SWE 432 -Web Application Development

Fall 2021

George Mason University

Dr. Kevin Moran

Week 6: Security & HTML

Administrivia

- *HW Assignment 2* - Due today Before Class
- *Midterm Exam* - In class next week
  - We will review today

Midterm Exam

- 3 Parts, In-class exam, closed book, 200 points total
  - ***Part 1:*** Multiple Choice Questions
  - ***Part 2:*** Short Answer
    - Either provide program output, or answer in a few short sentences
  - ***Part 3:*** Multi-Part Code Question (*implementing a simple microservice*)
  - Covers material from weeks 1-6, from both lectures and readings
  - You will have the ***entire*** class period to complete

Class Overview

- *Part 1 - Security:* What is it, authentication, and important types of attacks
- ***10 minute Break***
- *Part 2 -Intro to Frontend:* Templates, Databinding, and HTML
- *Part 3 - Midterm Exam Review:* Looking back at key concepts

Web Security

Security

- Why is it important?
  - Users' data is on the web
    - Blog comments, FB, Email, Banking, …
  - Can others steal it?
    - or who already has access?
  - Can others impersonate the user?
    - e.g., post on FB on the user's behalf

Security Requirements for Web Apps

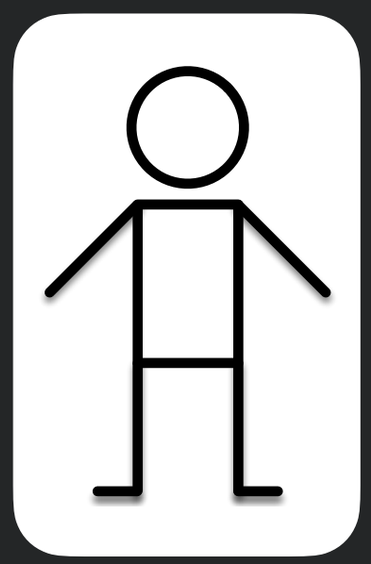
1. Authentication
   - Verify the ***identify*** of the parties involved
   - Who is it?
2. Authorization
   - Grant ***access*** to resources only to allowed users
   - Are you allowed?
3. Confidentiality
   - Ensure that ***information*** is given only to authenticated parties
   - Can you see it?
4. Integrity
   - Ensure that information is ***not changed*** or tampered with
   - Can you change it?

Threat Models

- What is being defended?
  - What resources are important to defend?
  - What malicious actors exist and what attacks might they employ?

- Who do we trust?
  - What entities or parts of system can be considered secure and trusted
  - Have to trust **something**!

Web Threat Models: Big Picture



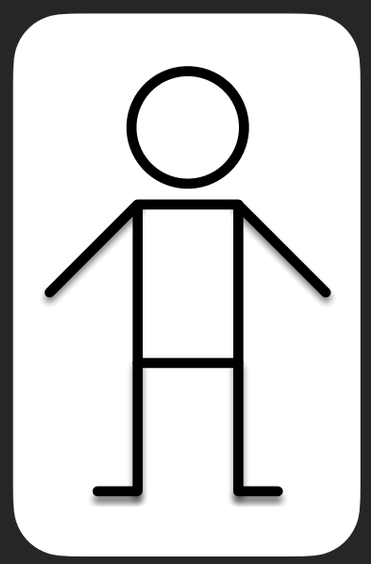
client page
(the “user”)

HTTP Request

HTTP Response

server

Web Threat Models: Big Picture



HTTP Request

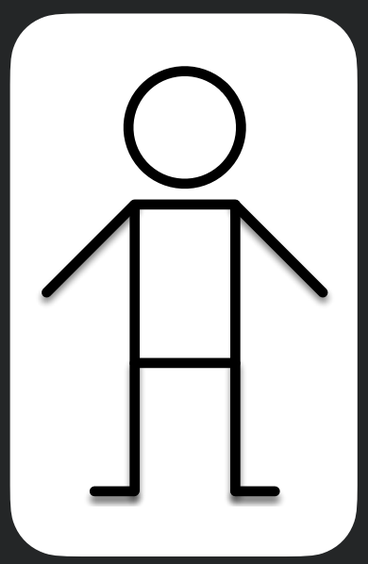
HTTP Response

client page
(the "user")

server

Do I trust that this request *really* came from the user?

Web Threat Models: Big Picture



HTTP Request

HTTP Response

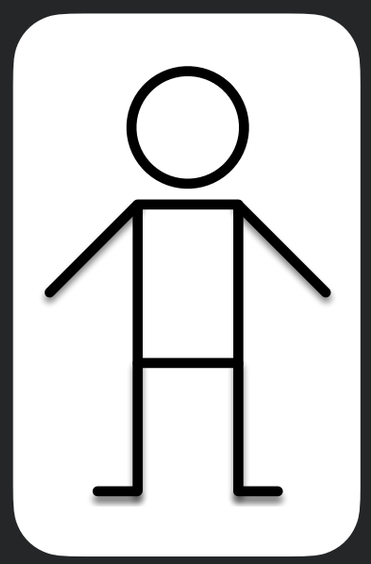
client page
(the "user")

server

Do I trust that this response *really* came from the server?

Do I trust that this request *really* came from the user?

Web Threat Models: Big Picture



HTTP Request

HTTP Response

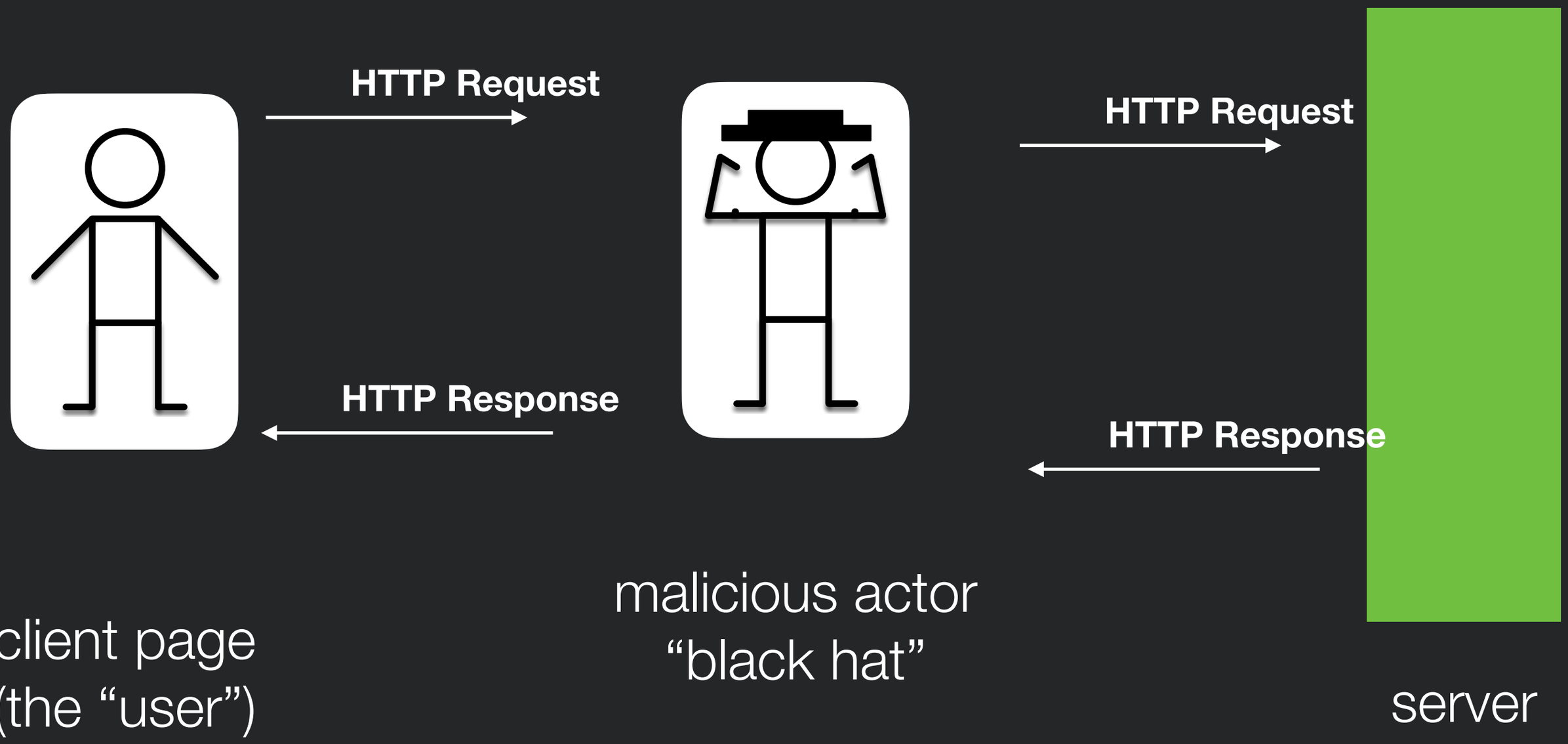
client page
(the "user")

server

Do I trust that this response *really* came from the server?

Do I trust that this request *really* came from the user?

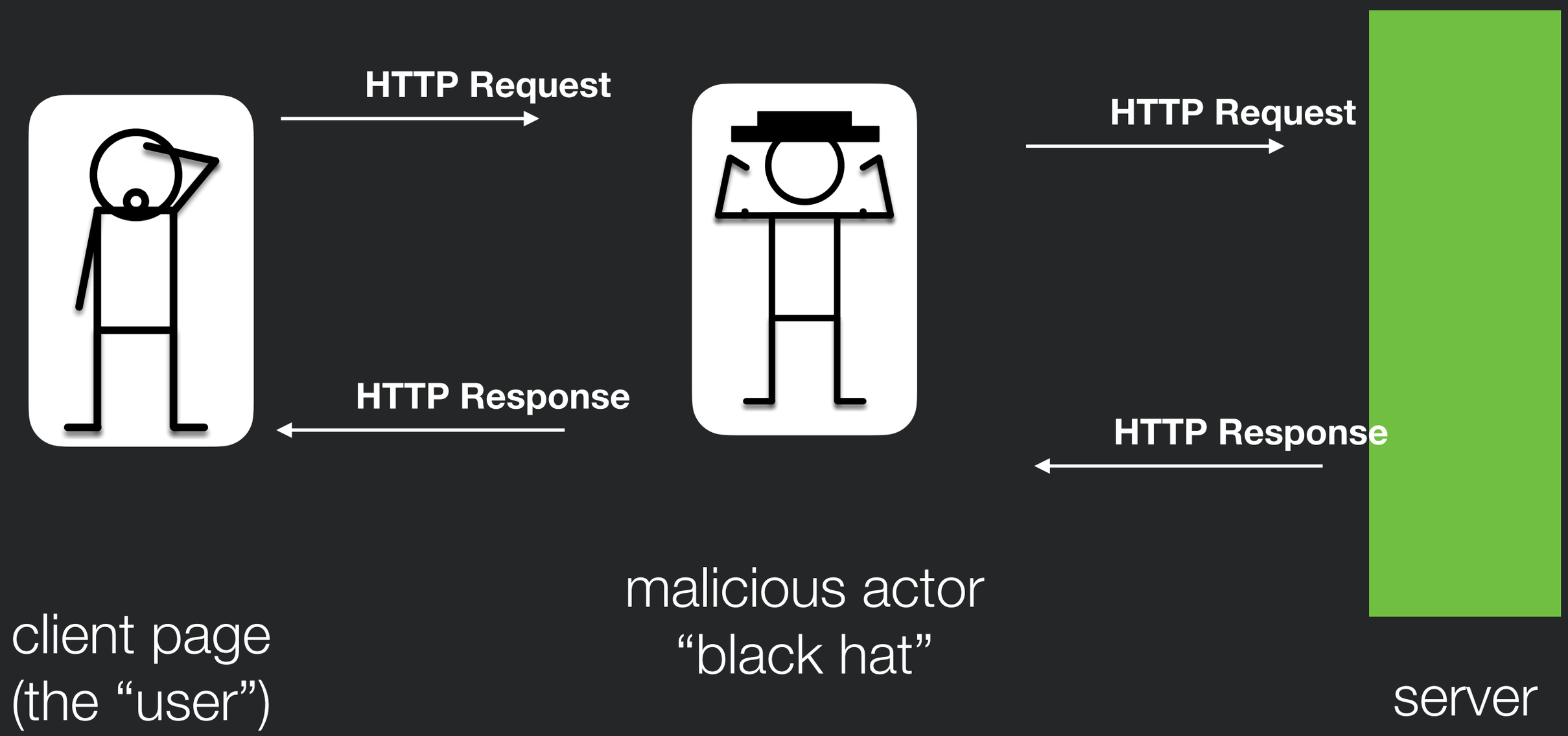
Web Threat Models: Big Picture



Do I trust that this response *really* came from the server?

Do I trust that this request *really* came from the user?

Web Threat Models: Big Picture

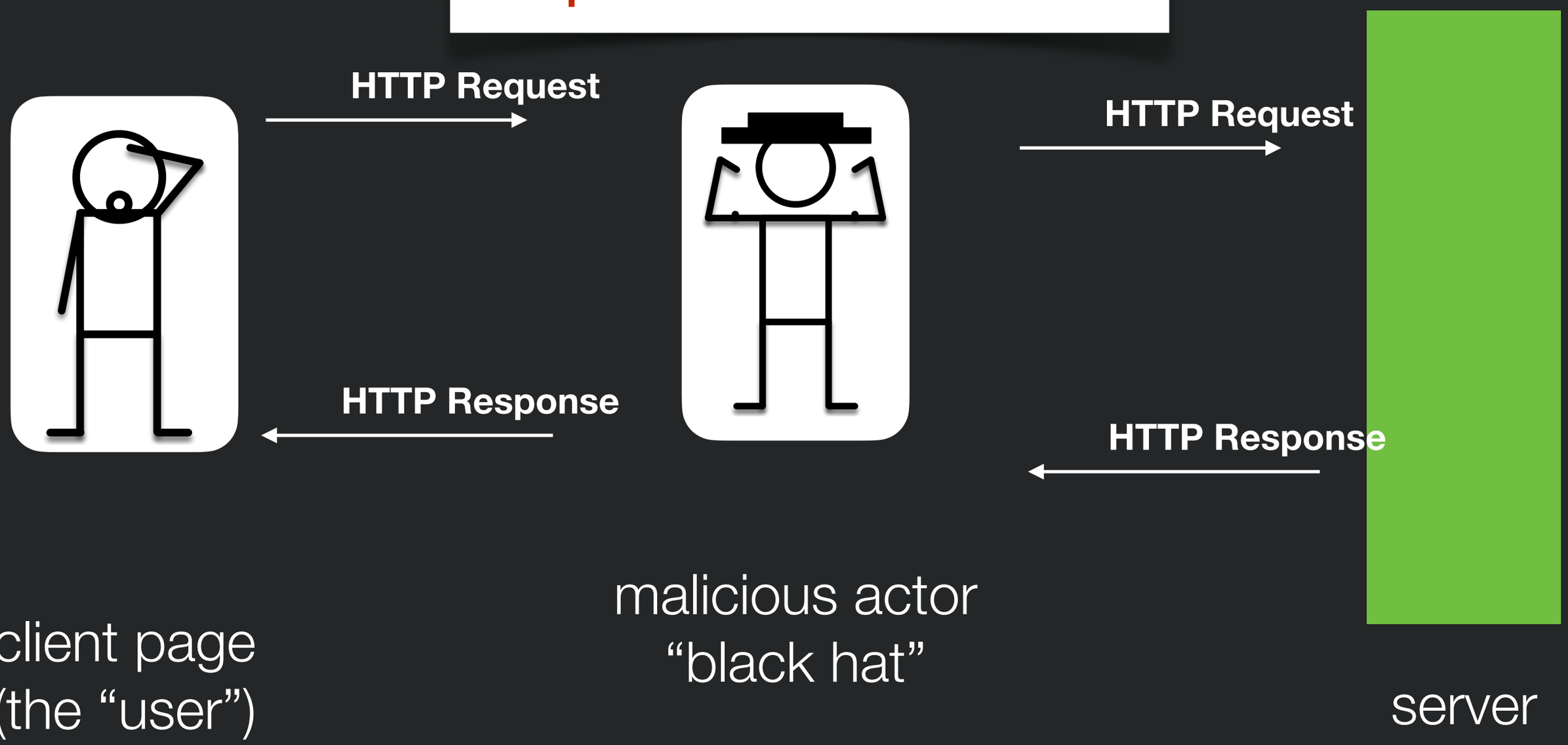


Do I trust that this response *really* came from the server?

Do I trust that this request *really* came from the user?

Web Threat M

Might be "man in the middle" that intercepts requests and impersonates user or server.



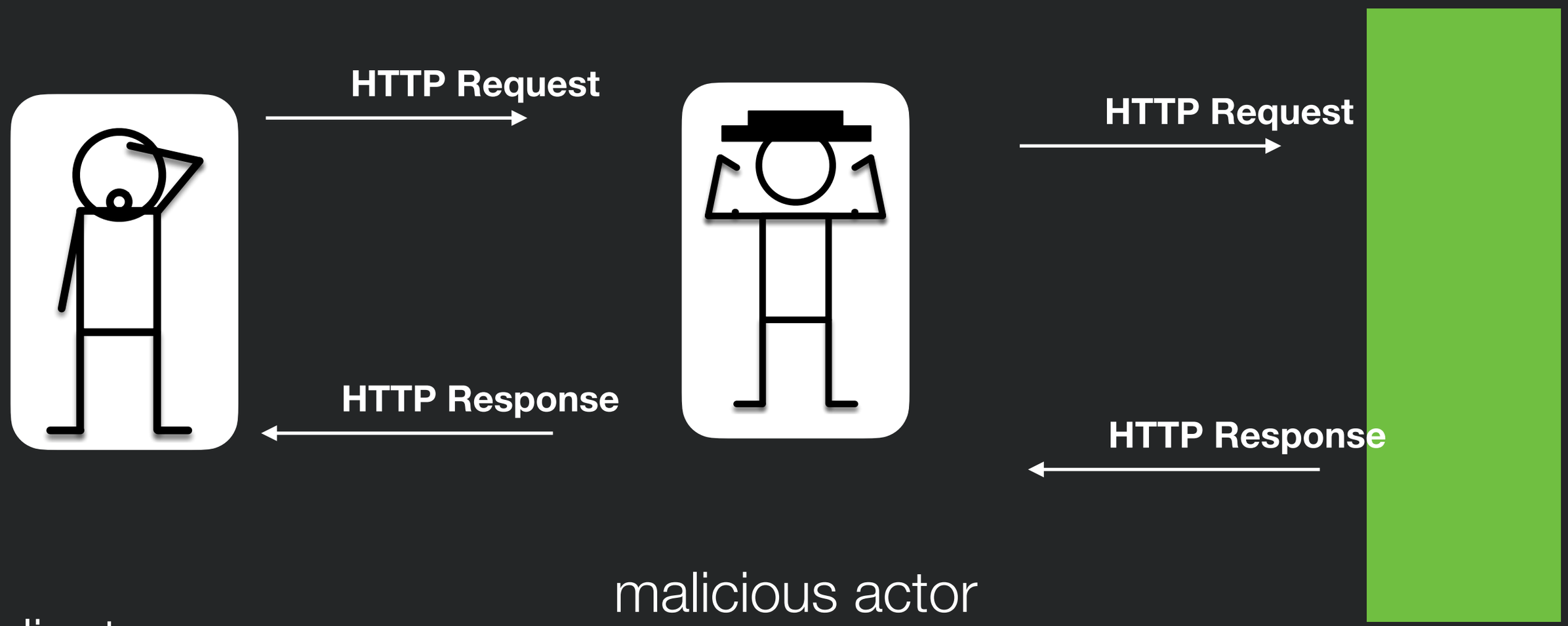
Do I trust that this response *really* came from the server?

Do I trust that this request *really* came from the user?

Security Requirements for Web Apps

1. Authentication
   - Verify the ***identify*** of the parties involved
   - Threat: Impersonation. A person pretends to be someone they are not.
2. Authorization
3. Confidentiality
   - Ensure that ***information*** is given only to authenticated parties
   - Threat: Eavesdropping. Information leaks to someone that should not have it.
4. Integrity
   - Ensure that information is ***not changed*** or tampered with
   - Threat: ***Tampering***.

Web Threat Models: Big Picture



client page
(the "user")

malicious actor
"black hat"

server

What if malicious actor impersonates server?

Man in the Middle

- Requests to server intercepted by man in the middle
  - Requests forwarded
  - But… response containing code edited, inserting malicious code
- Or could
  - Intercept and steal sensitive user data

HTTPS: HTTP over SSL

- Establishes secure connection from client to server
  - Uses SSL to encrypt traffic
- Ensures that others can’t impersonate server by establishing certificate authorities that vouch for server.
- Server trusts an HTTPS connection iff
  - The user trusts that the browser software correctly implements HTTPS with correctly pre-installed certificate authorities.
  - The user trusts the certificate authority to vouch only for legitimate websites.
  - The website provides a valid certificate, which means it was signed by a trusted authority.
  - The certificate correctly identifies the website (e.g., certificate received for “https://example.com" is for "example.com" and not other entity).

Using HTTPS

- If using HTTPS, important that all scripts are loaded through HTTPS
  - If mixed script from untrusted source served through HTTP, attacker could still modify this script, defeating benefits of HTTPS
- Example attack:
  - Banking website loads Bootstrap through HTTP rather than HTTPS
  - Attacker intercepts request for Bootstrap script, replaces with malicious script that steals user data or executes malicious action

Authentication

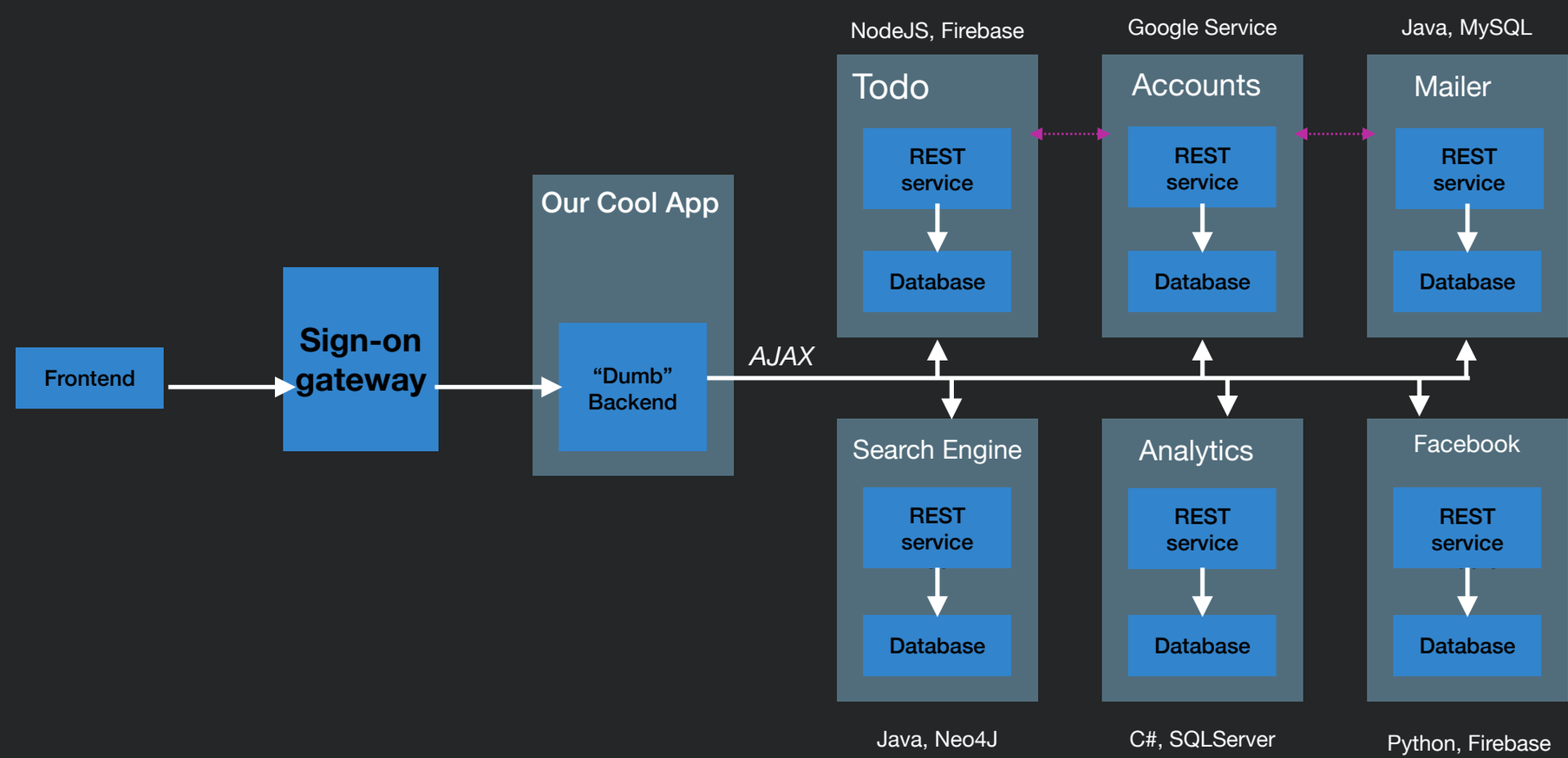
- How can we know the identify of the parties involved
- Want to customize experience based on identity
  - But need to determine identity first!
- Options
  - Ask user to create a new username and password
    - Lots of work to manage (password resets, storing passwords securely, …)
    - Hard to get right (#2 on the OWASP Top 10 Vulnerability List)
    - User does not really want another password…
  - Use an authentication provider to authenticate user
    - Google, FB, Twitter, Github, …

Authentication Provider

- Creates and tracks the identity of the user
- Instead of signing in directly to website, user signs in to authentication provider
  - Authentication provider issues token that uniquely proves identity of user

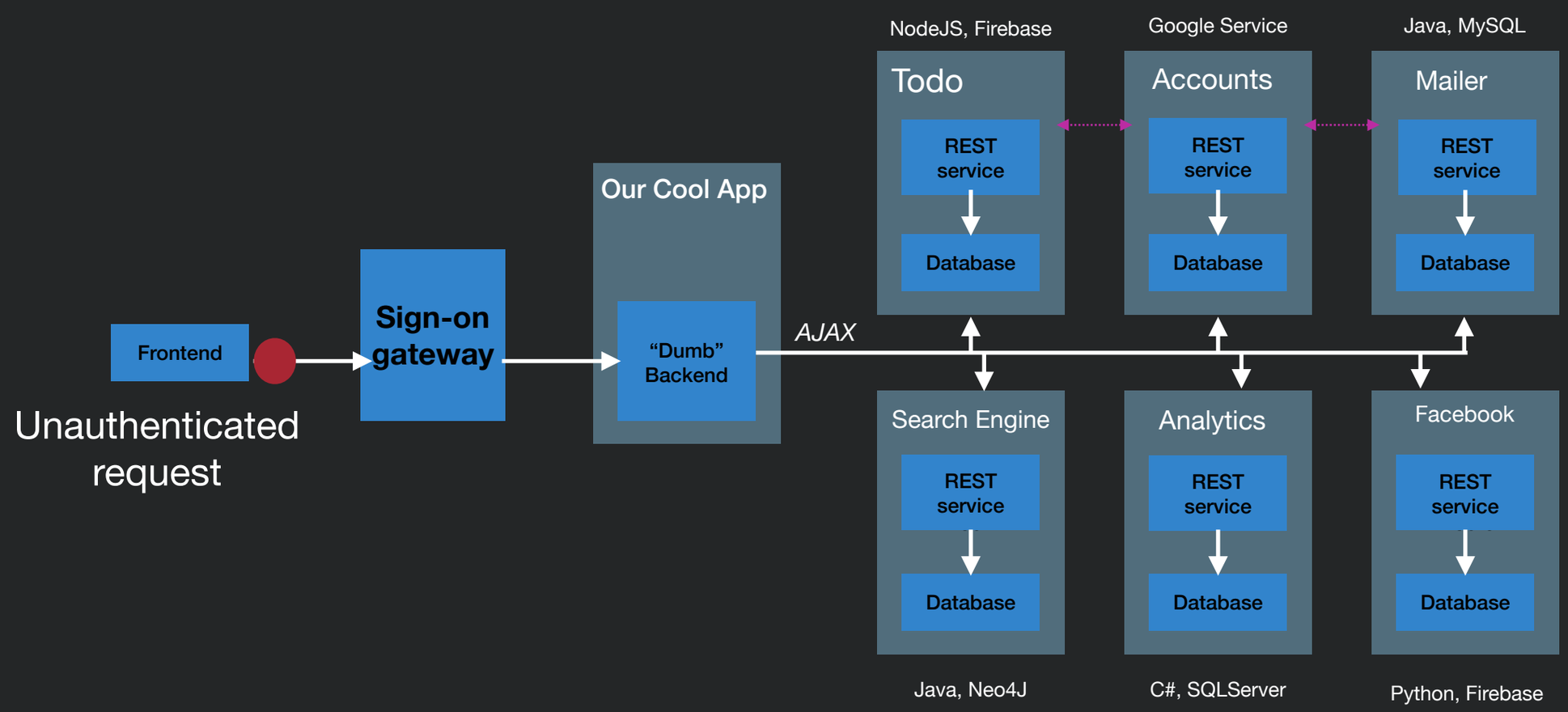
Sign-on Gateway

- Can place some magic “sign-on gateway” before out app - whether it’s got multiple services or just one



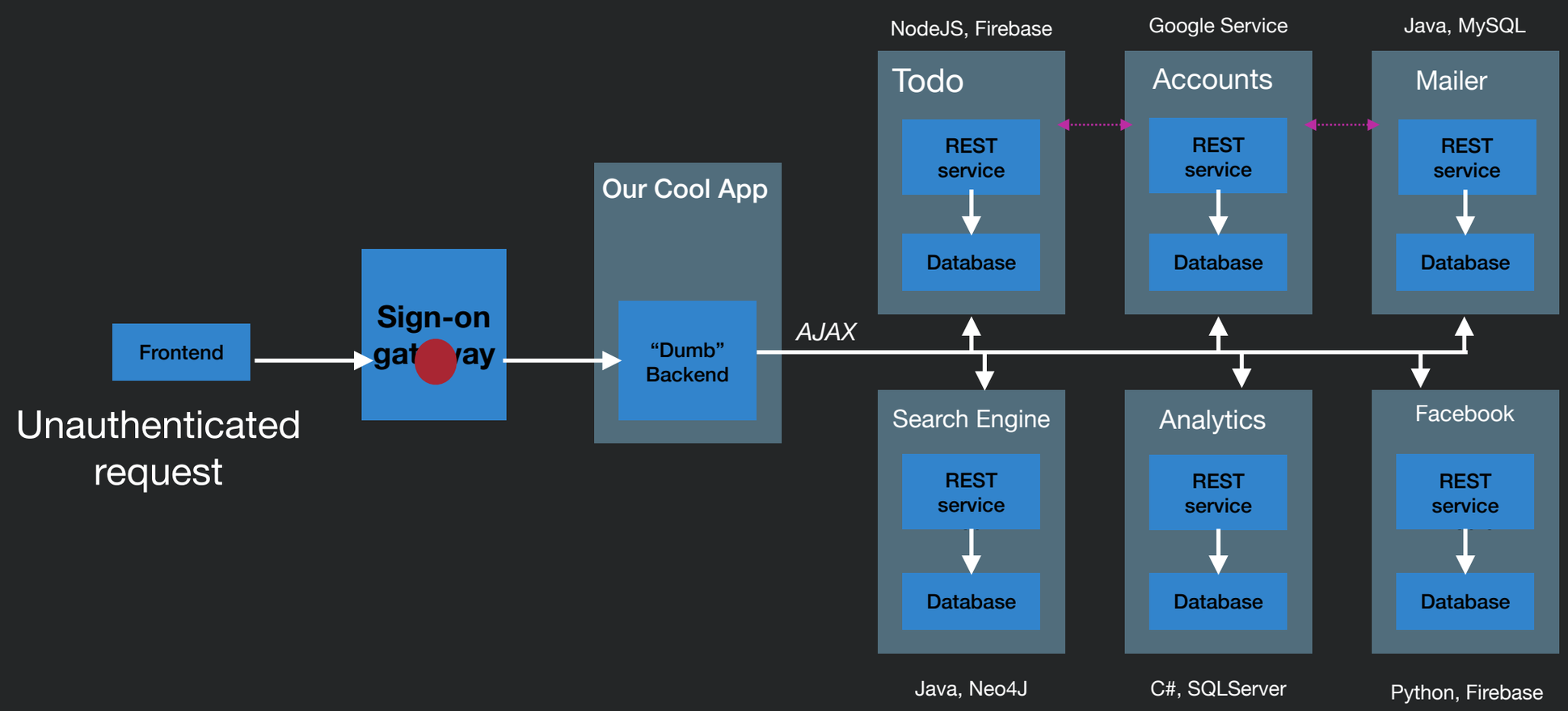
Sign-on Gateway

- Can place some magic “sign-on gateway” before out app - whether it’s got multiple services or just one



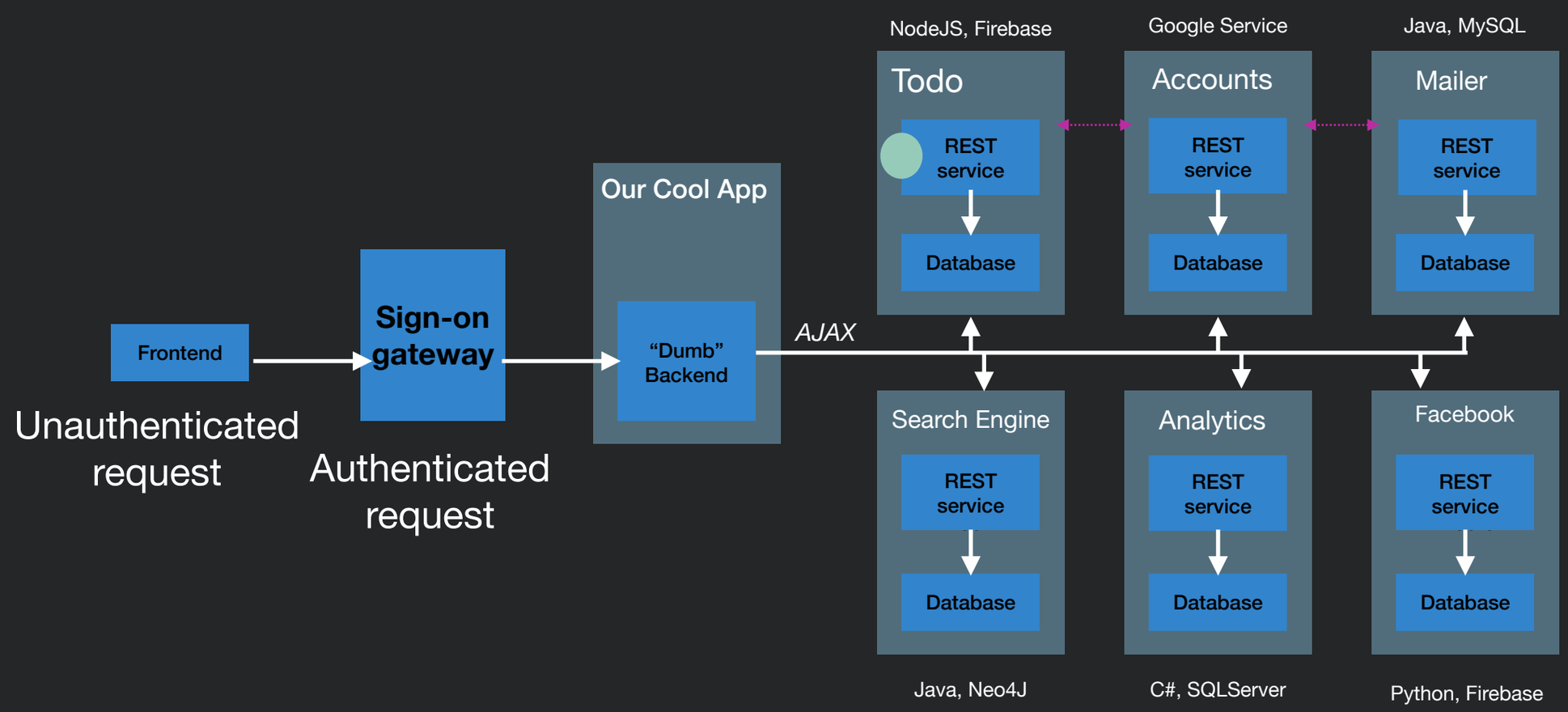
Sign-on Gateway

- Can place some magic “sign-on gateway” before out app - whether it’s got multiple services or just one



Sign-on Gateway

- Can place some magic “sign-on gateway” before out app - whether it’s got multiple services or just one

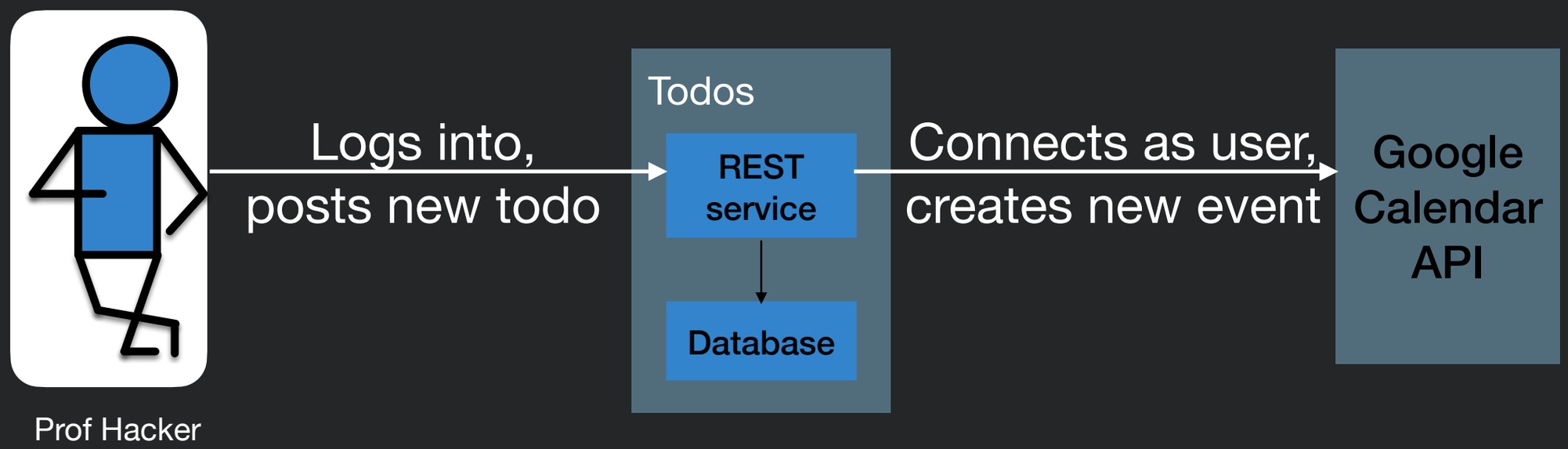


Bigger Picture - Authentication with Multiple Service Providers

- Let's consider updating a Todos app so that it can automatically put calendar events on a Google Calendar

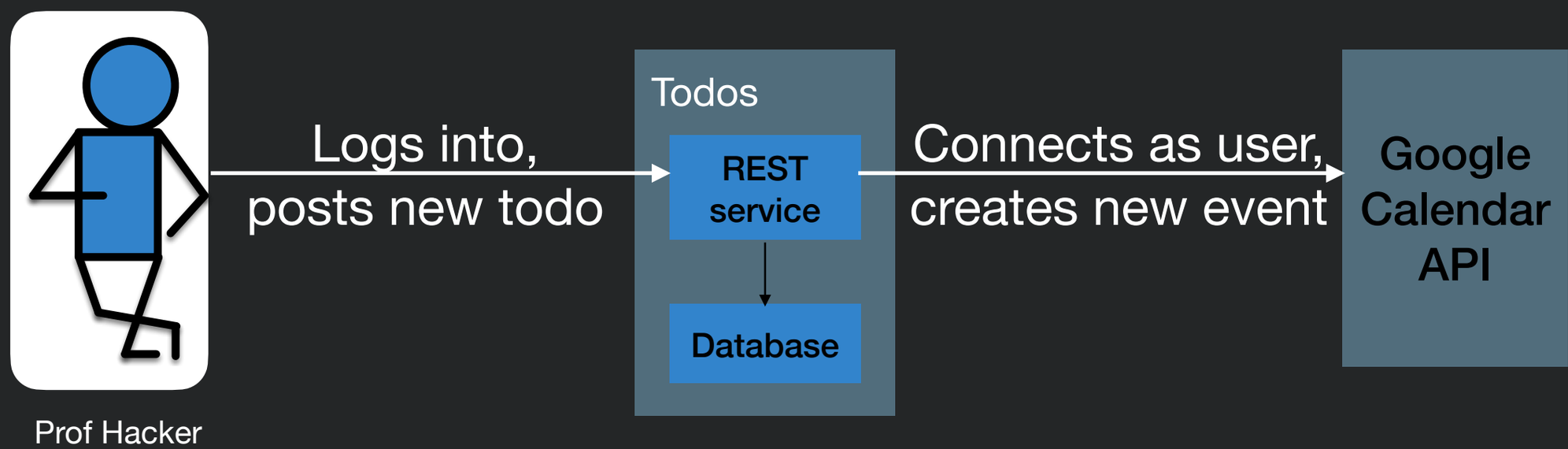
Bigger Picture - Authentication with Multiple Service Providers

- Let's consider updating a Todos app so that it can automatically put calendar events on a Google Calendar



Bigger Picture - Authentication with Multiple Service Providers

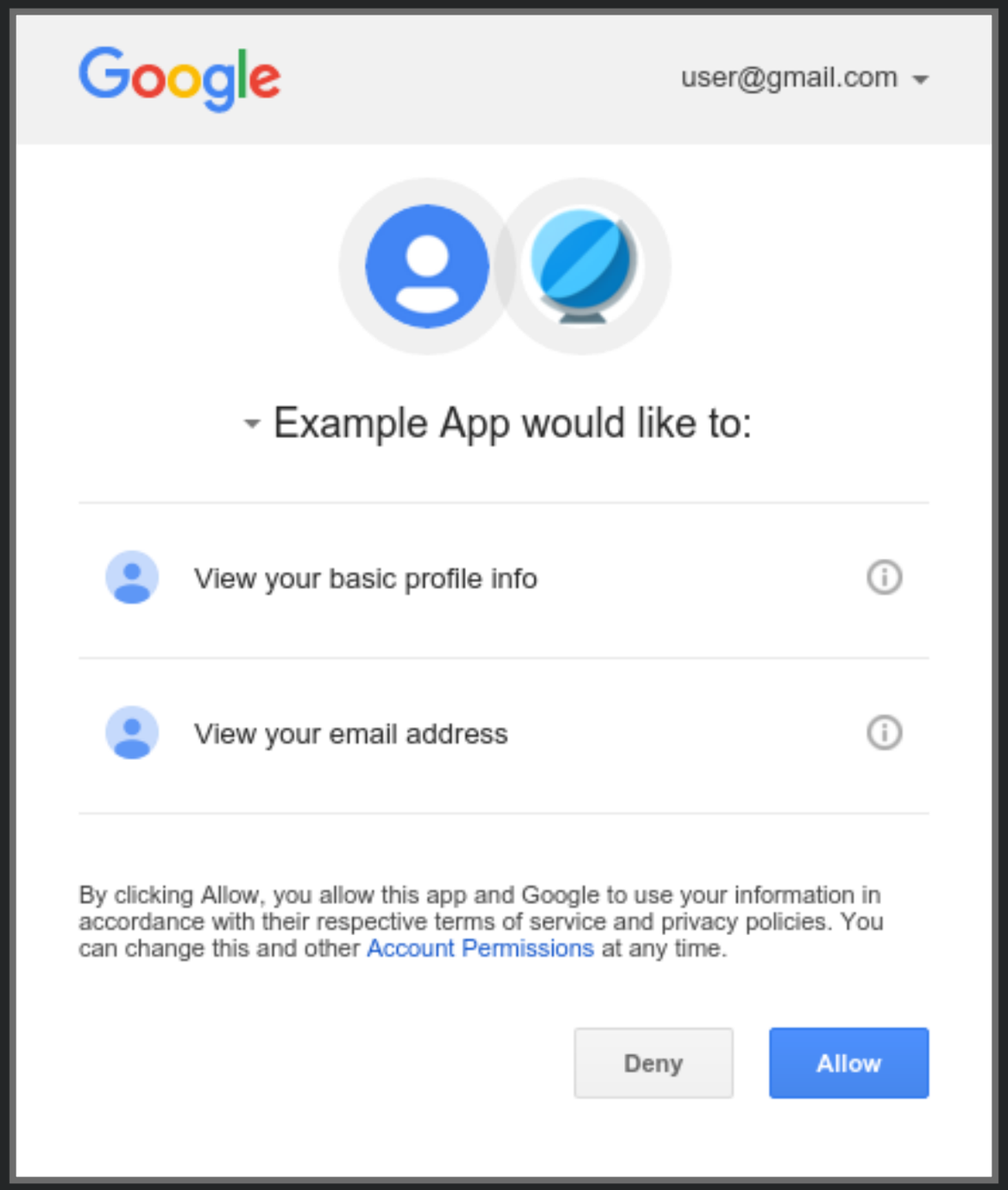
- Let's consider updating a Todos app so that it can automatically put calendar events on a Google Calendar



How does Todos tell Google that it's posting something for Prof Hacker?
Should Prof Hacker tell the Todos app her Google password?

We've Got Something for That…

We've Got Something for That…



OAuth

OAuth

- OAuth is a standard protocol for sharing information about users from a “service provider” to a “consumer app” ***<u>without</u>*** them disclosing their password to the consumer app

OAuth

- OAuth is a standard protocol for sharing information about users from a “service provider” to a “consumer app” ***<u>without</u>*** them disclosing their password to the consumer app

- 3 key actors:

OAuth

- OAuth is a standard protocol for sharing information about users from a “service provider” to a “consumer app” ***<u>without</u>*** them disclosing their password to the consumer app
- 3 key actors:
  - User, consumer app, service provider app

OAuth

- OAuth is a standard protocol for sharing information about users from a “service provider” to a “consumer app” ***<u>without</u>*** them disclosing their password to the consumer app
- 3 key actors:
  - User, consumer app, service provider app
  - E.x. “Prof Hacker,” “Todos App,” “Google Calendar”

OAuth

- OAuth is a standard protocol for sharing information about users from a “service provider” to a “consumer app” ***<u>without</u>*** them disclosing their password to the consumer app
- 3 key actors:
  - User, consumer app, service provider app
  - E.x. “Prof Hacker,” “Todos App,” “Google Calendar”
- Service provider issues a ***<u>token</u>*** on the user’s behalf that the consumer can use

OAuth

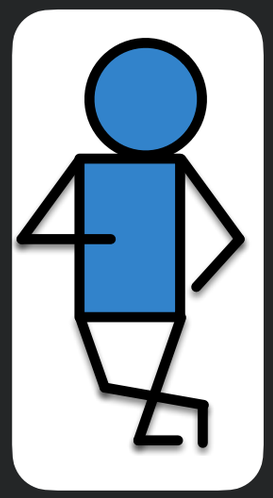
- OAuth is a standard protocol for sharing information about users from a “service provider” to a “consumer app” ***<u>without</u>*** them disclosing their password to the consumer app
- 3 key actors:
  - User, consumer app, service provider app
  - E.x. “Prof Hacker,” “Todos App,” “Google Calendar”
- Service provider issues a ***<u>token</u>*** on the user’s behalf that the consumer can use
- Consumer holds onto this token on behalf of the user

OAuth

- OAuth is a standard protocol for sharing information about users from a “service provider” to a “consumer app” ***<u>without</u>*** them disclosing their password to the consumer app
- 3 key actors:
  - User, consumer app, service provider app
  - E.x. “Prof Hacker,” “Todos App,” “Google Calendar”
- Service provider issues a ***<u>token</u>*** on the user’s behalf that the consumer can use
- Consumer holds onto this token on behalf of the user
- Protocol could be considered a conversation…

An OAuth Conversation

Goal: `TodosApp` can post events to `User's` calendar. `TodosApp` never finds out `User's` email or password



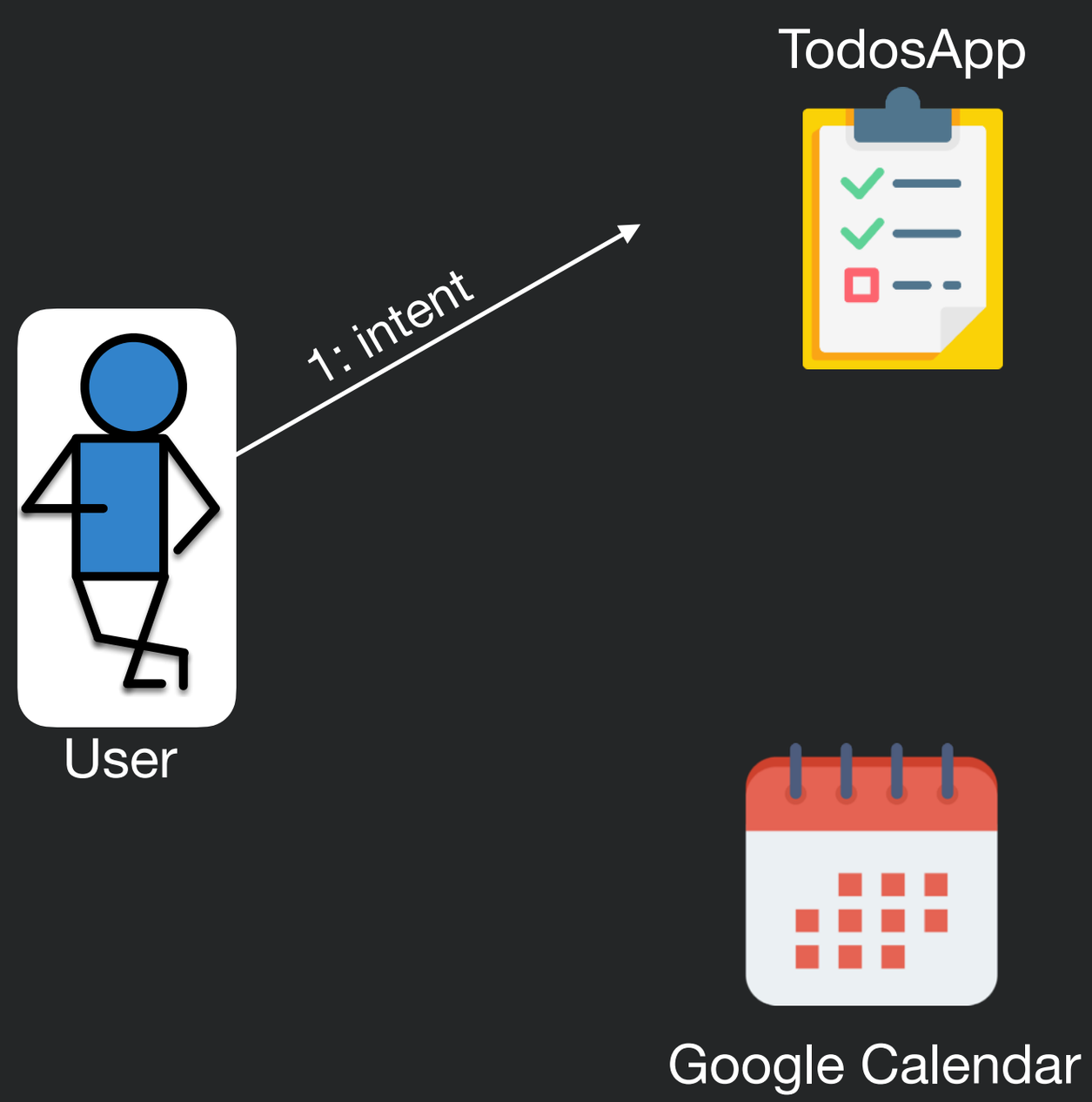
User

TodosApp

Google Calendar

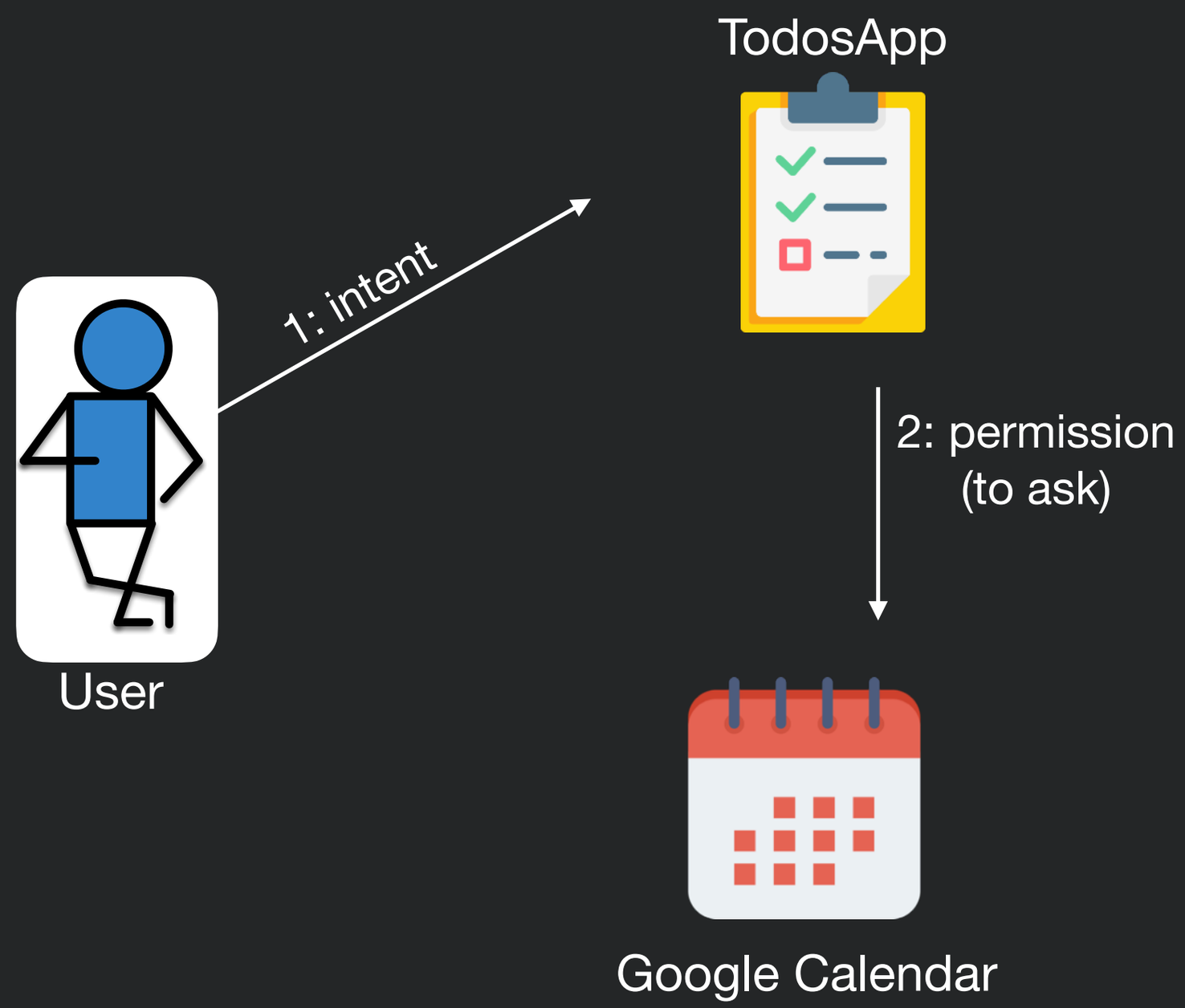
An OAuth Conversation

Goal: `TodosApp` can post events to `User's` calendar. `TodosApp` never finds out `User's` email or password



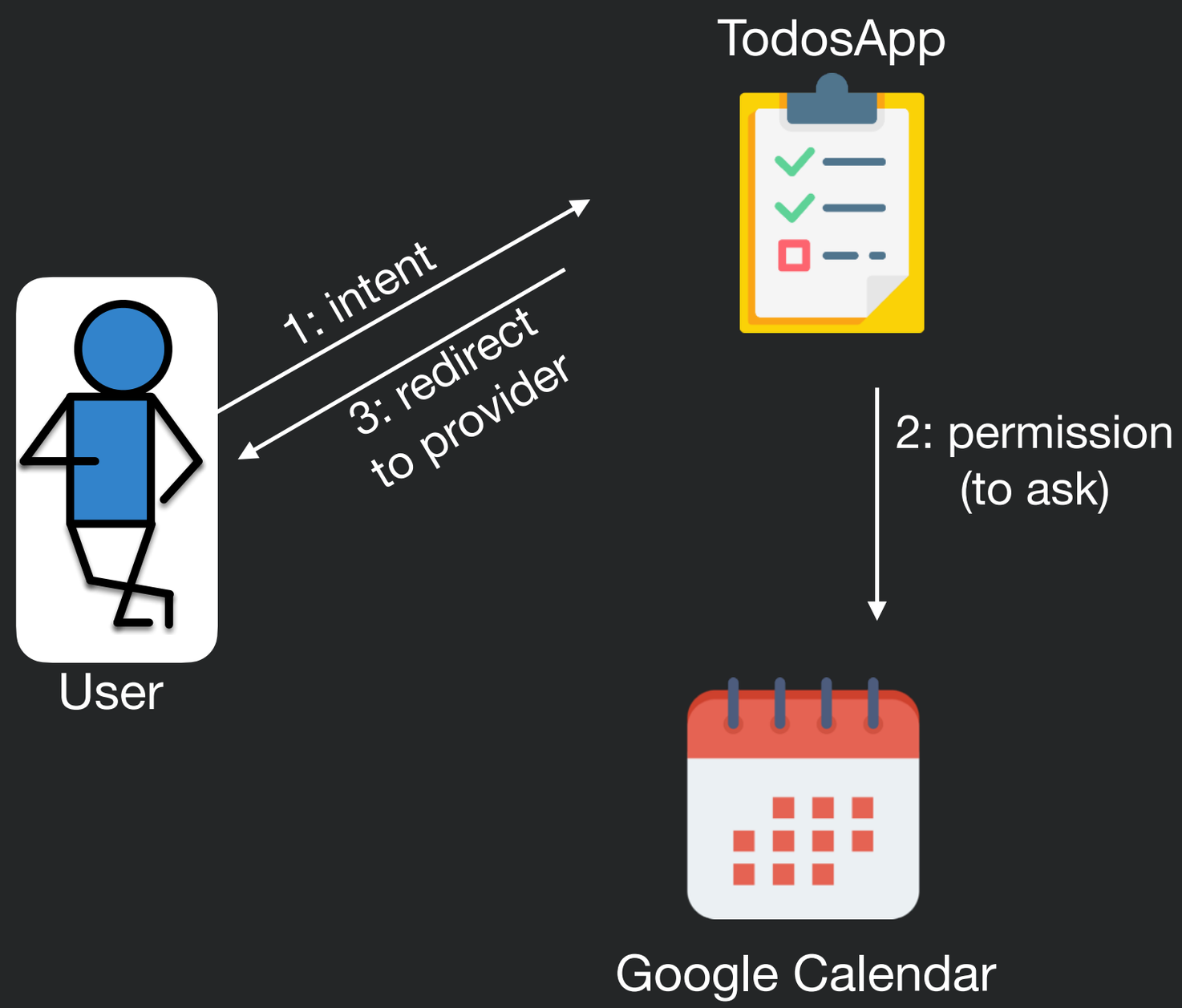
An OAuth Conversation

Goal: `TodosApp` can post events to `User's` calendar. `TodosApp` never finds out `User's` email or password



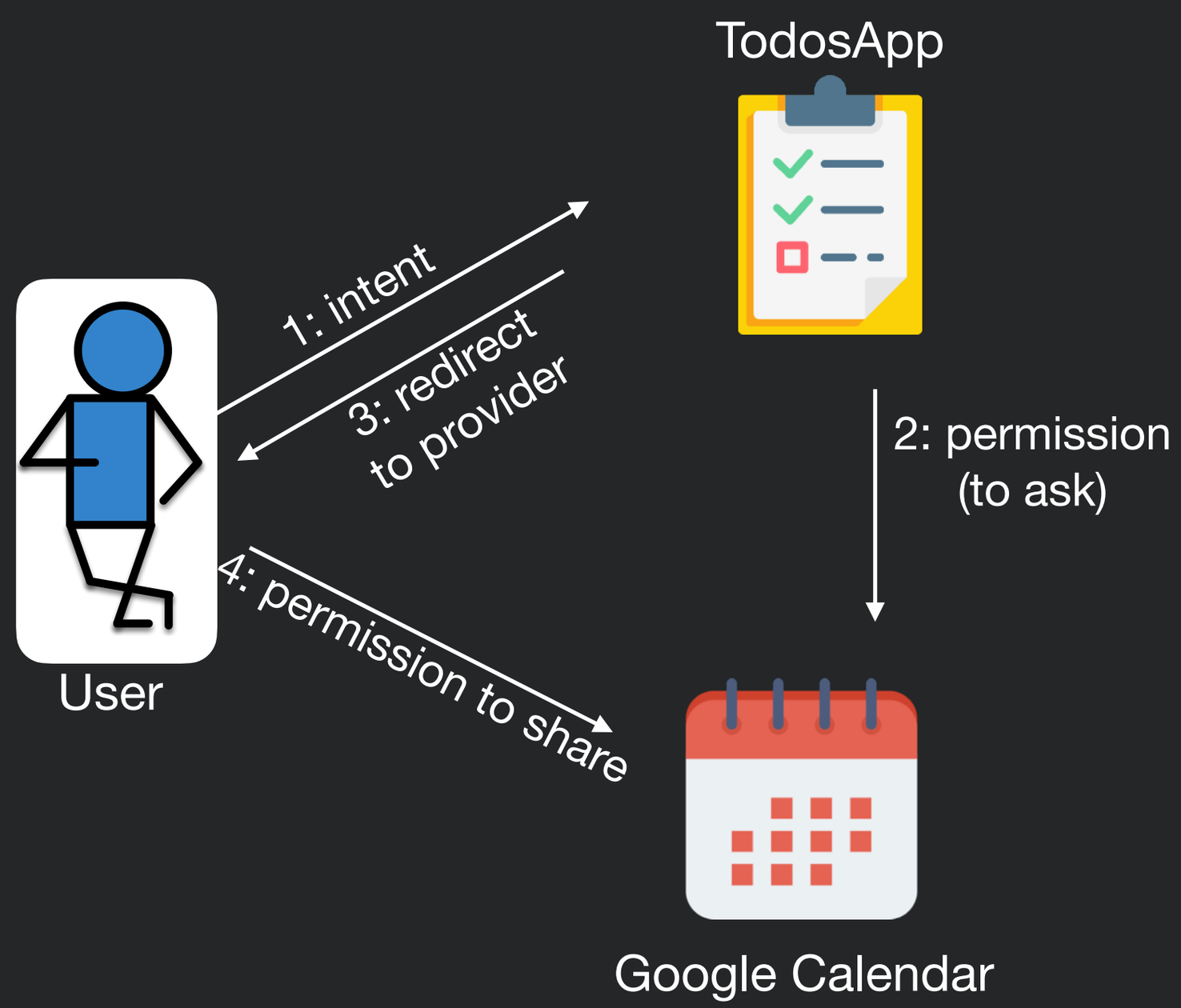
An OAuth Conversation

Goal: `TodosApp` can post events to `User's` calendar. `TodosApp` never finds out `User's` email or password



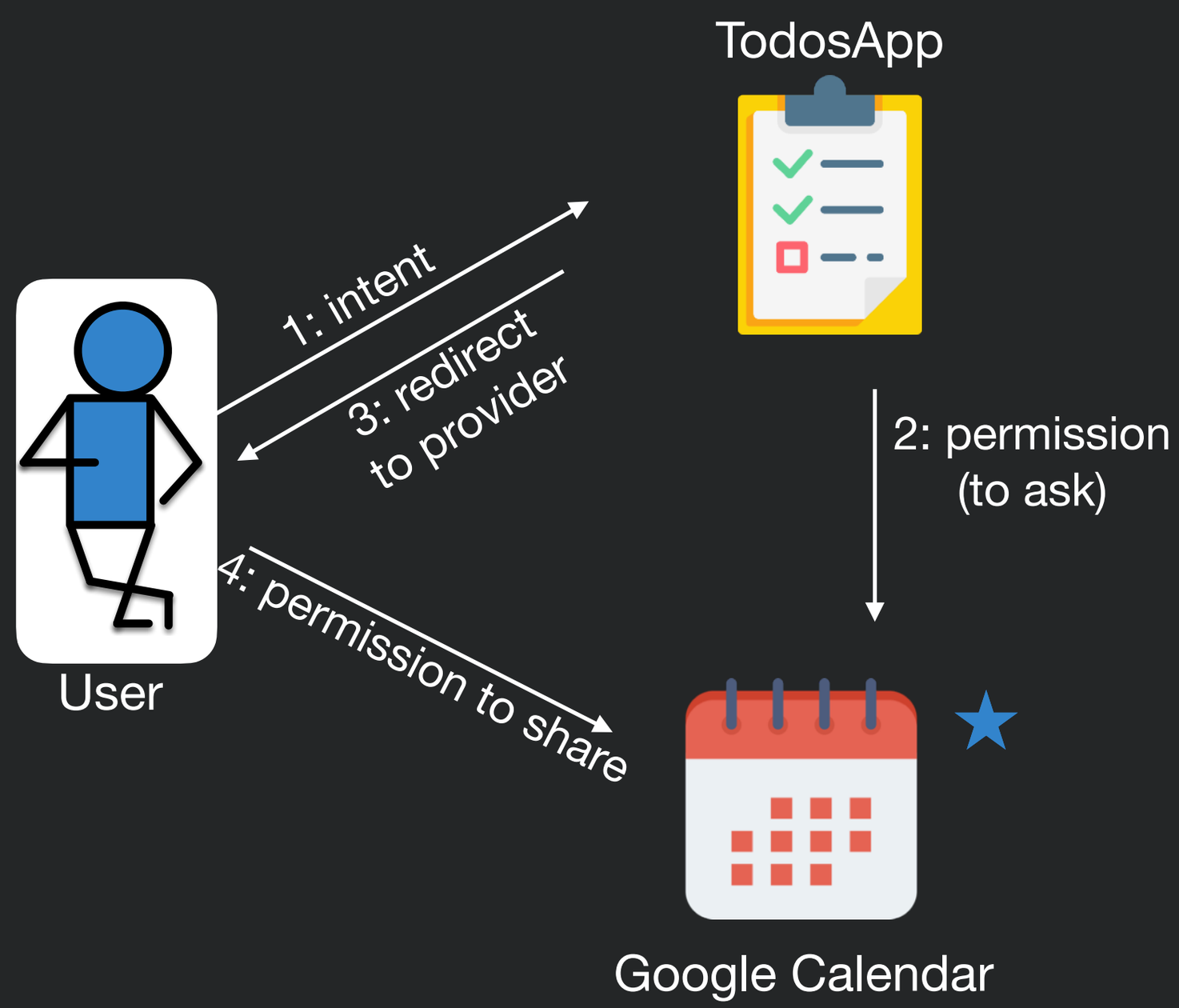
An OAuth Conversation

Goal: `TodosApp` can post events to `User's` calendar. `TodosApp` never finds out `User's` email or password



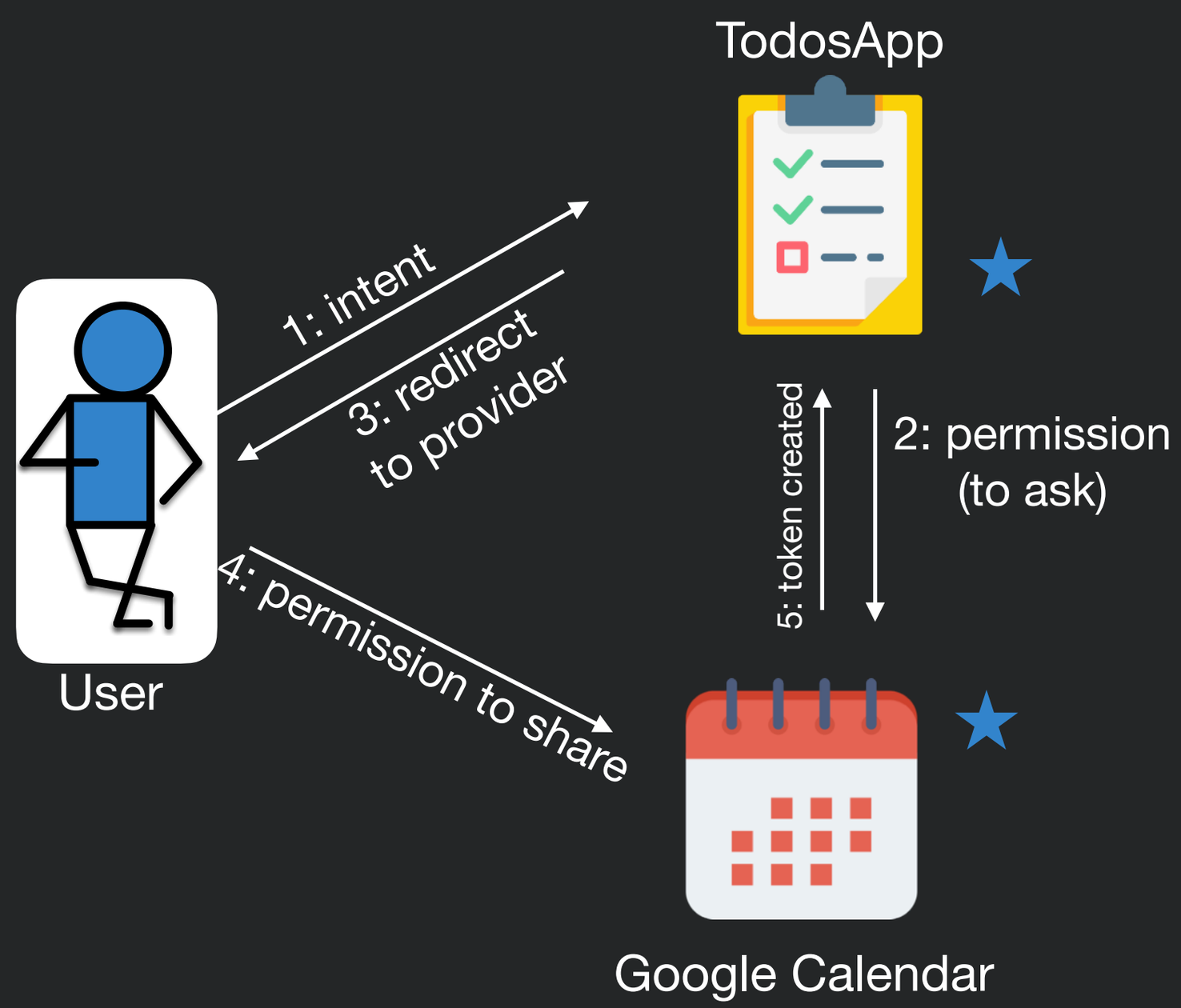
An OAuth Conversation

Goal: `TodosApp` can post events to `User's` calendar. `TodosApp` never finds out `User's` email or password



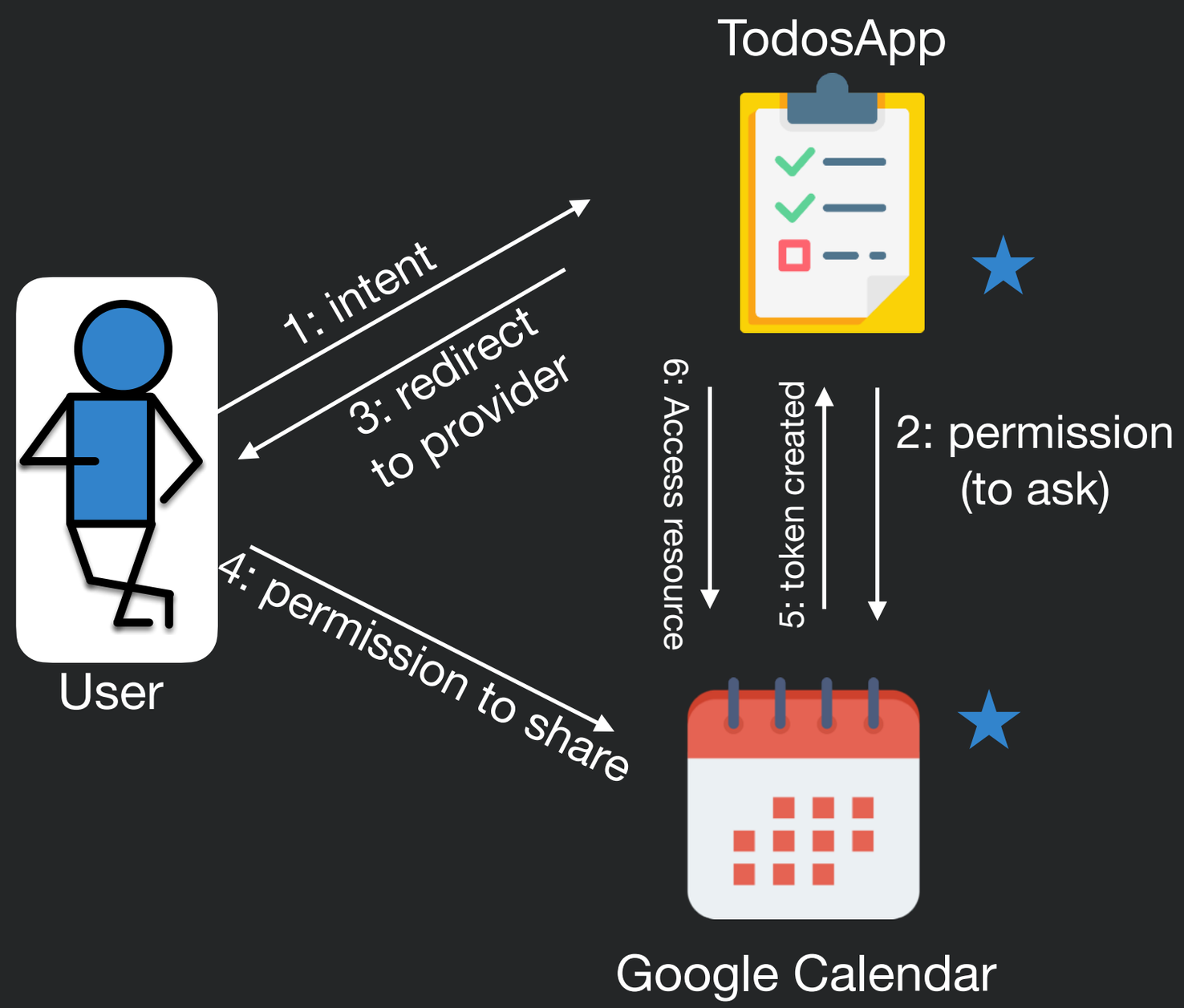
An OAuth Conversation

Goal: `TodosApp` can post events to `User's` calendar. `TodosApp` never finds out `User's` email or password



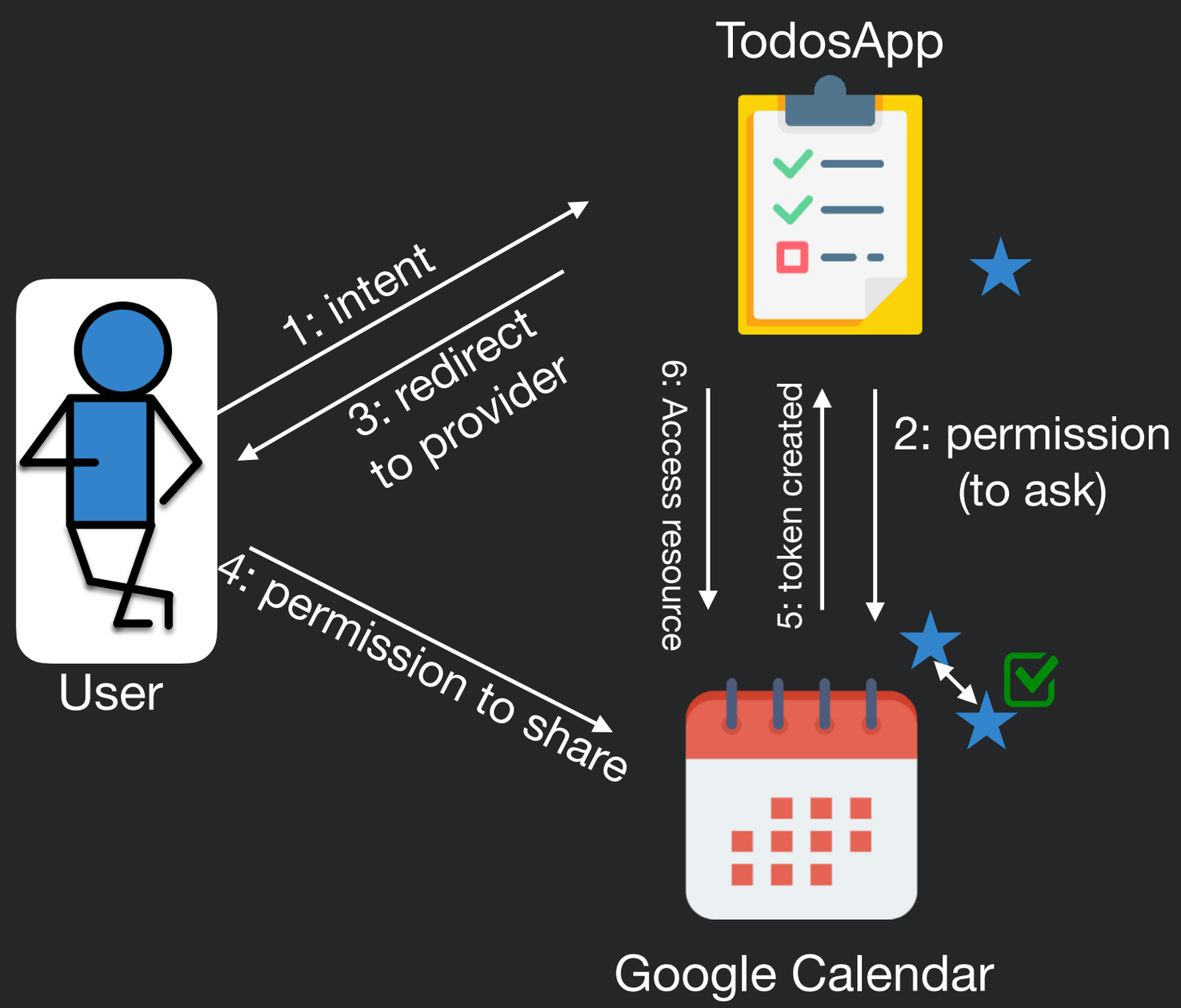
An OAuth Conversation

Goal: `TodosApp` can post events to `User's` calendar. `TodosApp` never finds out `User's` email or password



An OAuth Conversation

Goal: `TodosApp` can post events to `User's` calendar. `TodosApp` never finds out `User's` email or password



Tokens?

A token is a **secret value**. Holding it gives us access to some privileged data. The token identifies our users and app.

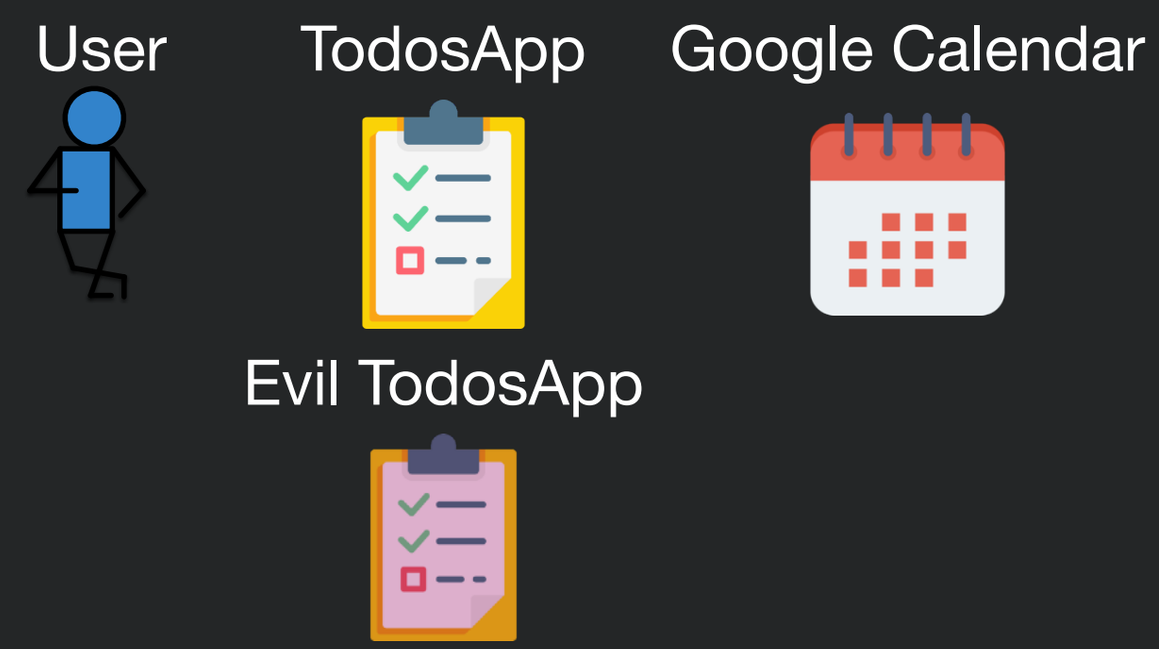
Example token:


```
eyJhbGciOiJSUzI1NiIsImtpZCI6ImU3Yjg2NjFjMGUwM2Y3ZTk3NjQyNGUxZWFiMzI5OWIxNzRhNGVlNWUifQ.eyJpc3MiOiJodHRwczovL3NlY3VyZXRva
2VuLmdvb2dsZS5jb20vYXV0aGRlbW8tNzJhNDIiLCJuYW1lIjoiSm9uYXRoYW4gQmVsbCIsInBpY3R1cmUiOiJodHRwczovL2xoNS5nb29nbGV1c2VyY29ud
GVudC5jb20vLW0tT29jRlU1R0x3L0FBQUFBQUFBQUFJL0FBQUFBQUFBQUgwL0JVV2tONkRtTVJrL3Bob3RvLmpwZyIsImF1ZCI6ImF1dGhkZW1vLTcyYTQyI
iwiYXV0aF90aW1lIjoxNDc3NTI5MzcxLCJ1c2VyX2lkIjoiSk1RclFpdTlTUlRkeDY0YlR5Z0EzeHhEY3VIMiIsInN1YiI6IkpNUXJRaXU5U1JUZHg2NGJUe
WdBM3h4RGN1SDIiLCJpYXQiOjE0Nzc1MzA4ODUsImV4cCI6MTQ3NzUzNDQ4NSwiZW1haWwiOiJqb25iZWxsd2l0aG5vaEBnbWFpbC5jb20iLCJlbWFpbF92Z
XJpZmllZCI6dHJ1ZSwiZmlyZWJhc2UiOnsiaWRlbnRpdGllcyI6eyJnb29nbGUuY29tIjpbIjEwOTA0MDM1MjU3NDMxMjE1NDIxNiJdLCJlbWFpbCI6WyJqb
25iZWxsd2l0aG5vaEBnbWFpbC5jb20iXX0sInNpZ25faW5fcHJvdmlkZXIiOiJnb29nbGUuY29tIn19.rw1pPK377hDGmSaX31uKRphKt4i79aHjceepnA8A
2MppBQnPJlCqmgSapxs-Pwmp-1Jk382VooRwc8TfL6E1UQUl65yi2aYYzSx3mWMTWtPTHTkMN4E-GNprp7hX-
pqD3PncBh1bq1dThPNyjHLp3CUlPPO_QwaAeSuG5xALhzfYkvLSINty4FguD9vLHydpVHWscBNCDHACOqSeV5MzUs6ZYMnBIitFhbkak6z5OClvxGTGMhvI8
m11hIHdWgNGnDQNNoosiifzlwMqDHiF5t3KOL-mxtcNq33TvMAc43JElxnyB4g7qV2hJIOy4MLtLxphAfCeQZA3sxGf7vDXBQ
```


Decoded:

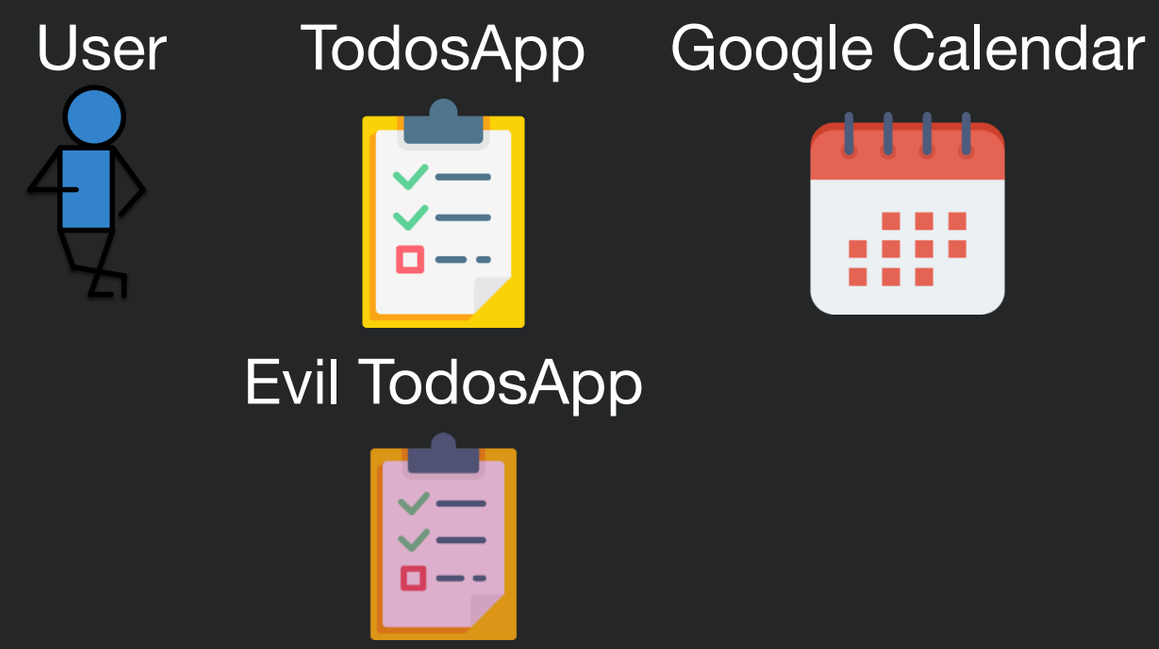

```
{
    "iss": "https://securetoken.google.com/authdemo-72a42",
    "name": “Alsyssa P Hacker”,
    "picture": "https://lh5.googleusercontent.com/-m-OocFU5GLw/AAAAAAAAAAI/AAAAAAAAAH0/BUWkN6DmMRk/photo.jpg",
    "aud": "authdemo-72a42",
    "auth_time": 1477529371,
    "user_id": "JMQrQiu9SRTdx64bTygA3xxDcuH2",
    "sub": "JMQrQiu9SRTdx64bTygA3xxDcuH2",
    "iat": 1477530885,
    "exp": 1477534485,
    "email": "alyssaphacker@gmail.com",
    "email_verified": true,
    "firebase": {
    "identities": {
        "google.com": ["109040352574312154216"],
            "email": ["alyssaphacker@gmail.com"]
    },
    "sign_in_provider": "google.com"
},
    "uid": "JMQrQiu9SRTdx64bTygA3xxDcuH2"
}
```

Trust in OAuth



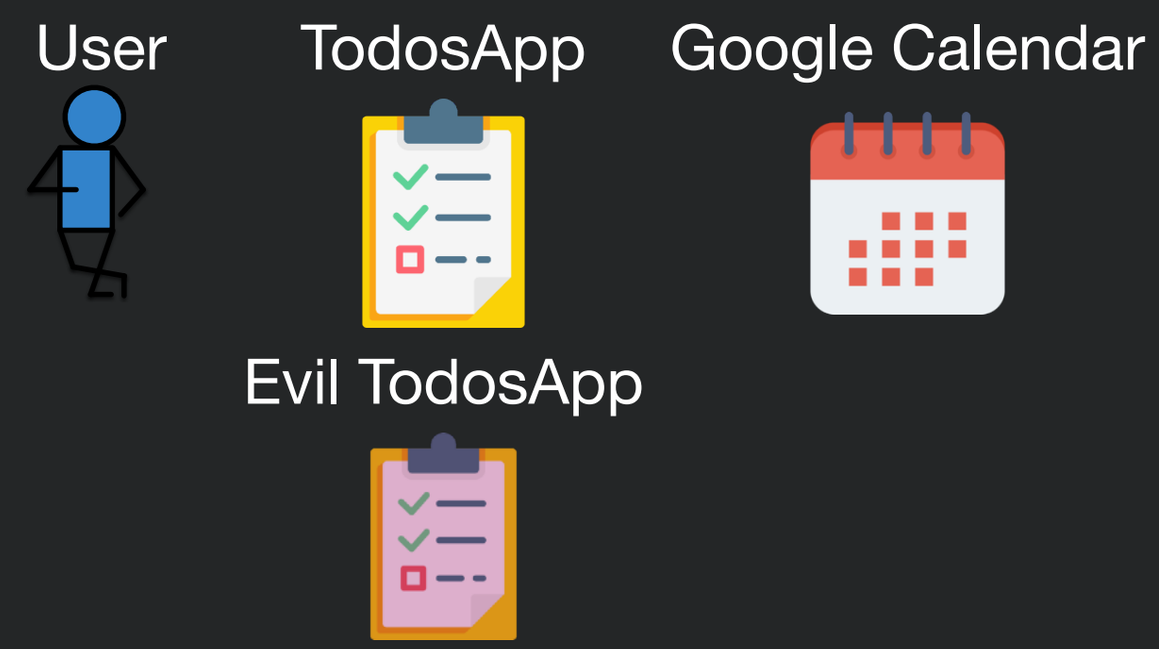
Trust in OAuth

- How does the Service provider (Google calendar) know what the TodosApp is?



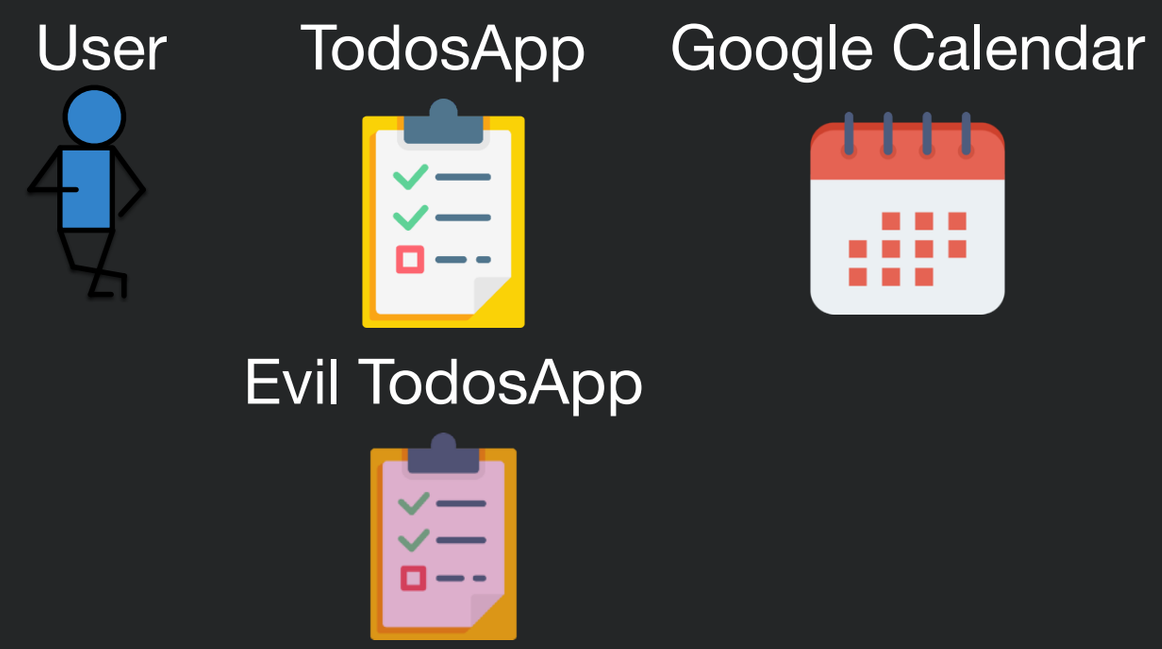
Trust in OAuth

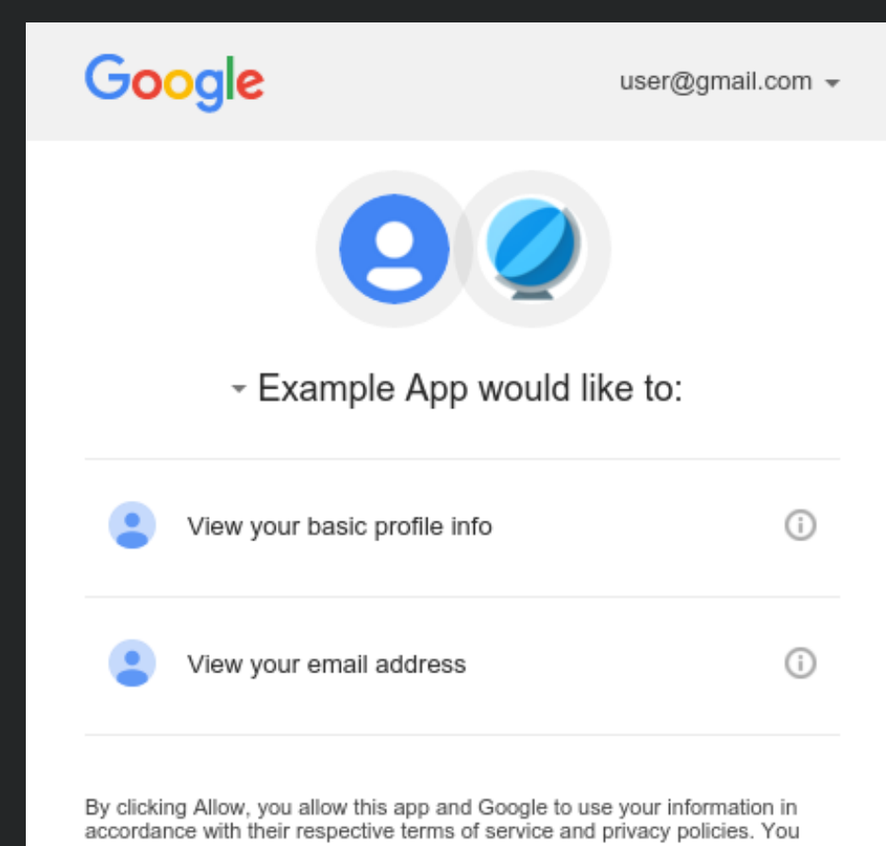
- How does the Service provider (Google calendar) know what the TodosApp is?
- Solution: When you set up OAuth for the first time, you must register your consumer app with the service provider



Trust in OAuth

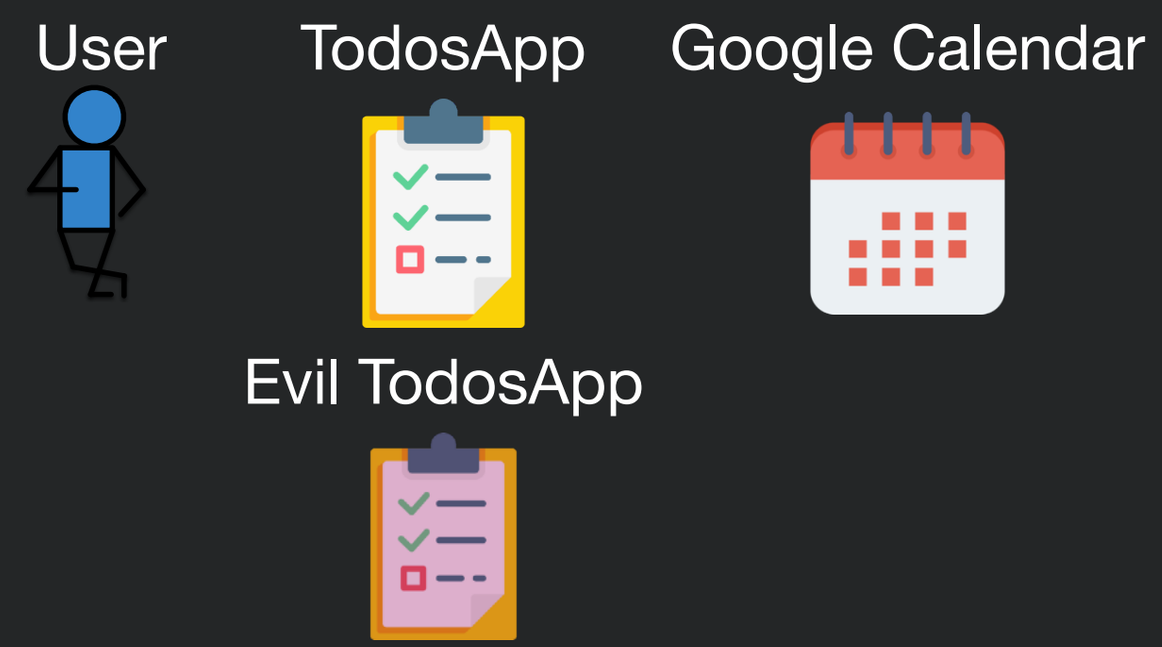
- How does the Service provider (Google calendar) know what the TodosApp is?
- Solution: When you set up OAuth for the first time, you must register your consumer app with the service provider
- Let the user decide

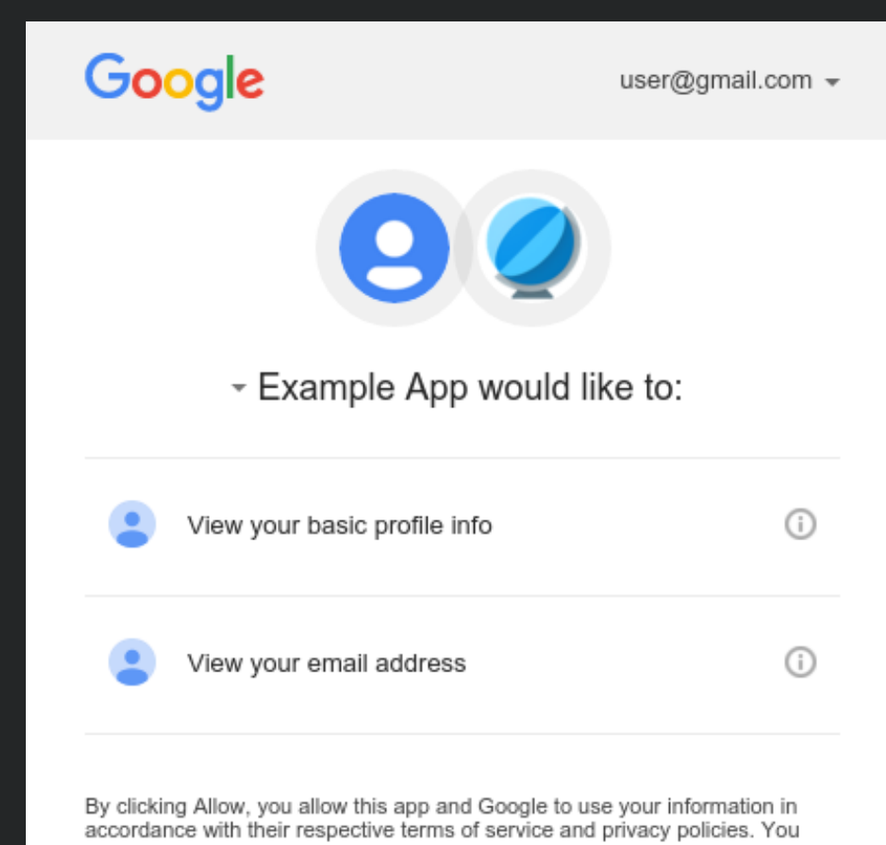




Trust in OAuth

- How does the Service provider (Google calendar) know what the TodosApp is?
- Solution: When you set up OAuth for the first time, you must register your consumer app with the service provider
- Let the user decide
  - … they were the one who clicked the link after all

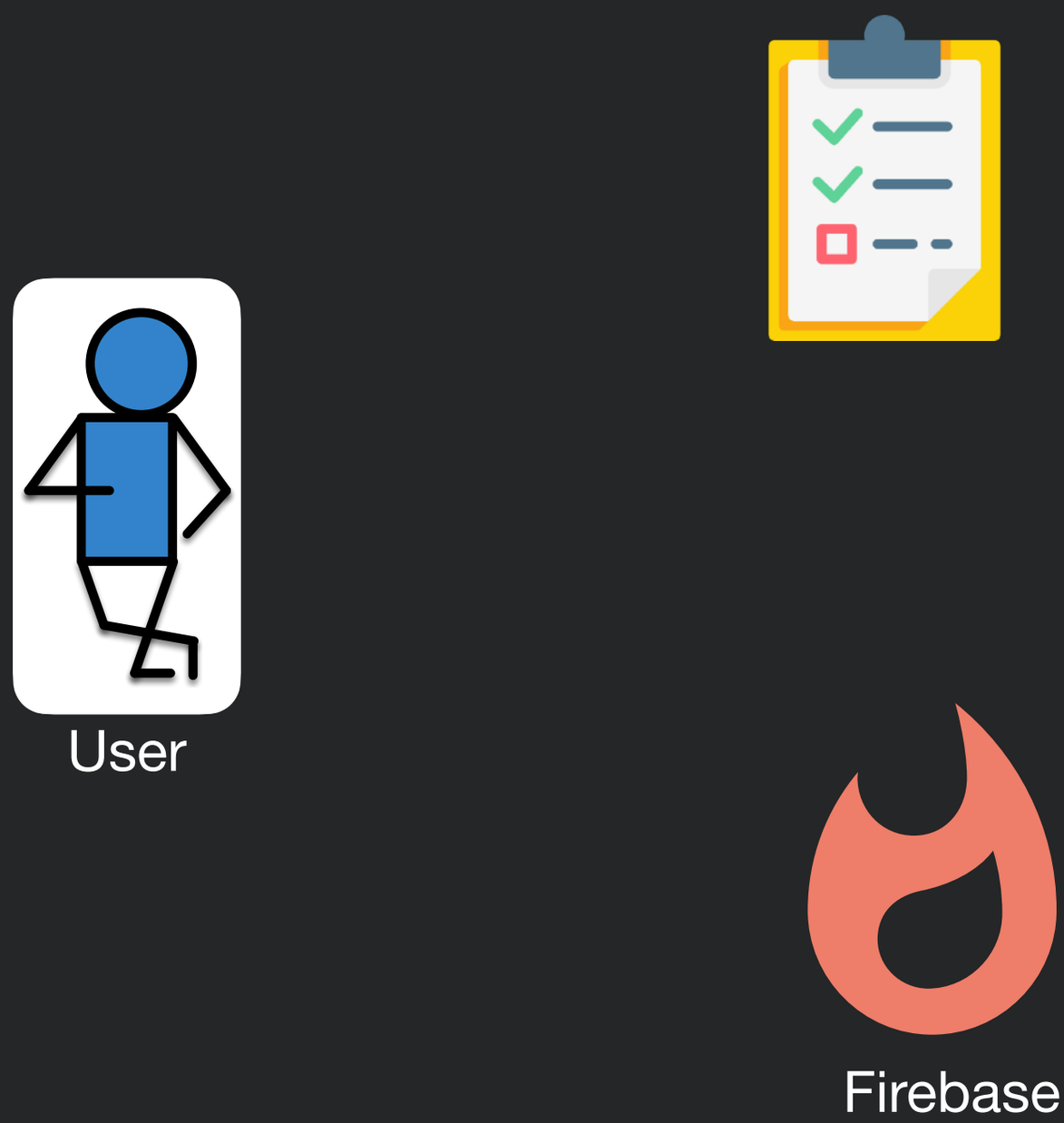




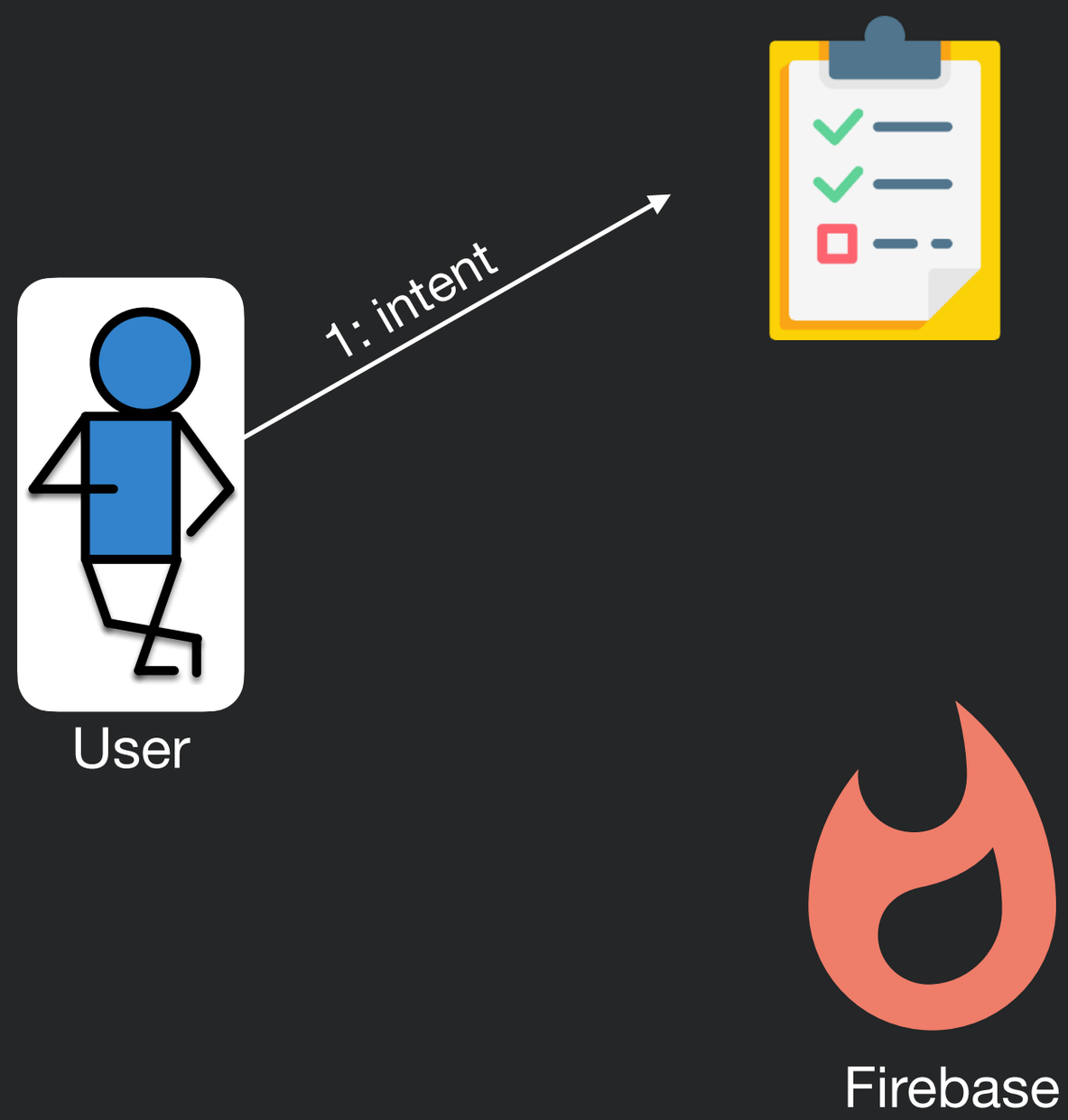
Authentication as a Service

- Whether we are building “microservices” or not, might make sense to farm out our authentication (user registration/logins) to another service
- Why?
  - Security
  - Reliability
  - Convenience
- We can use OAuth for this!

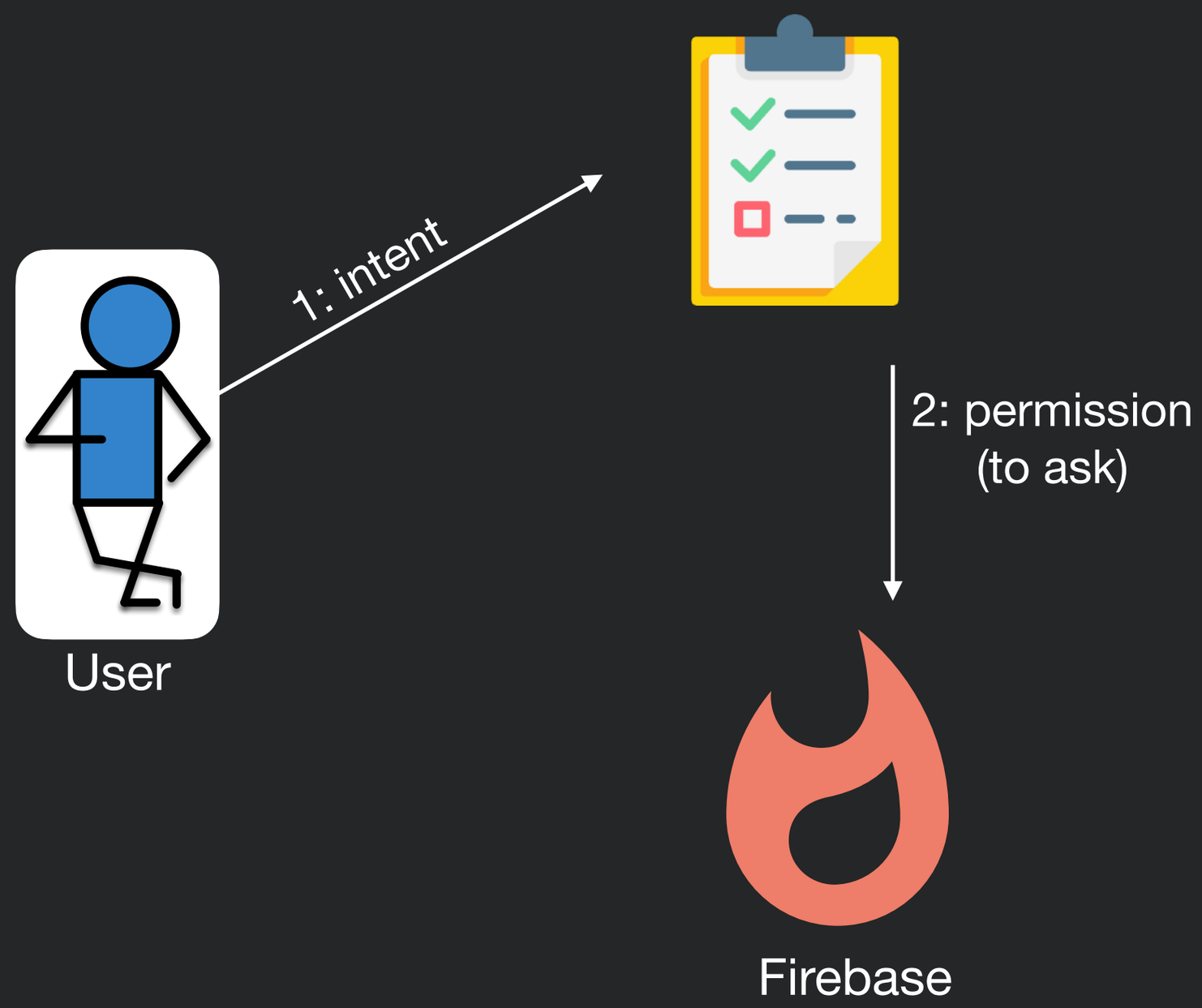
Using an Authentication Service



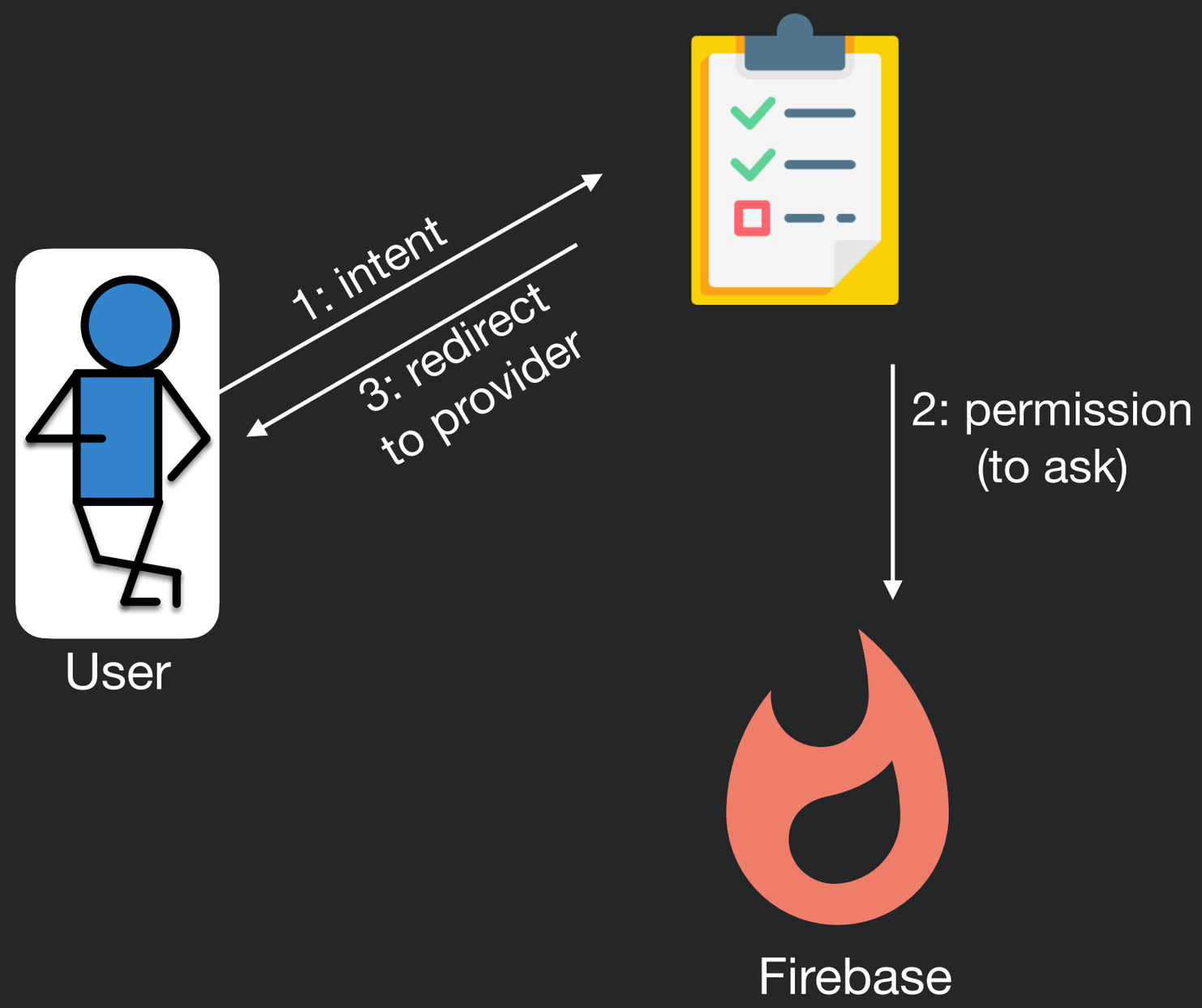
Using an Authentication Service



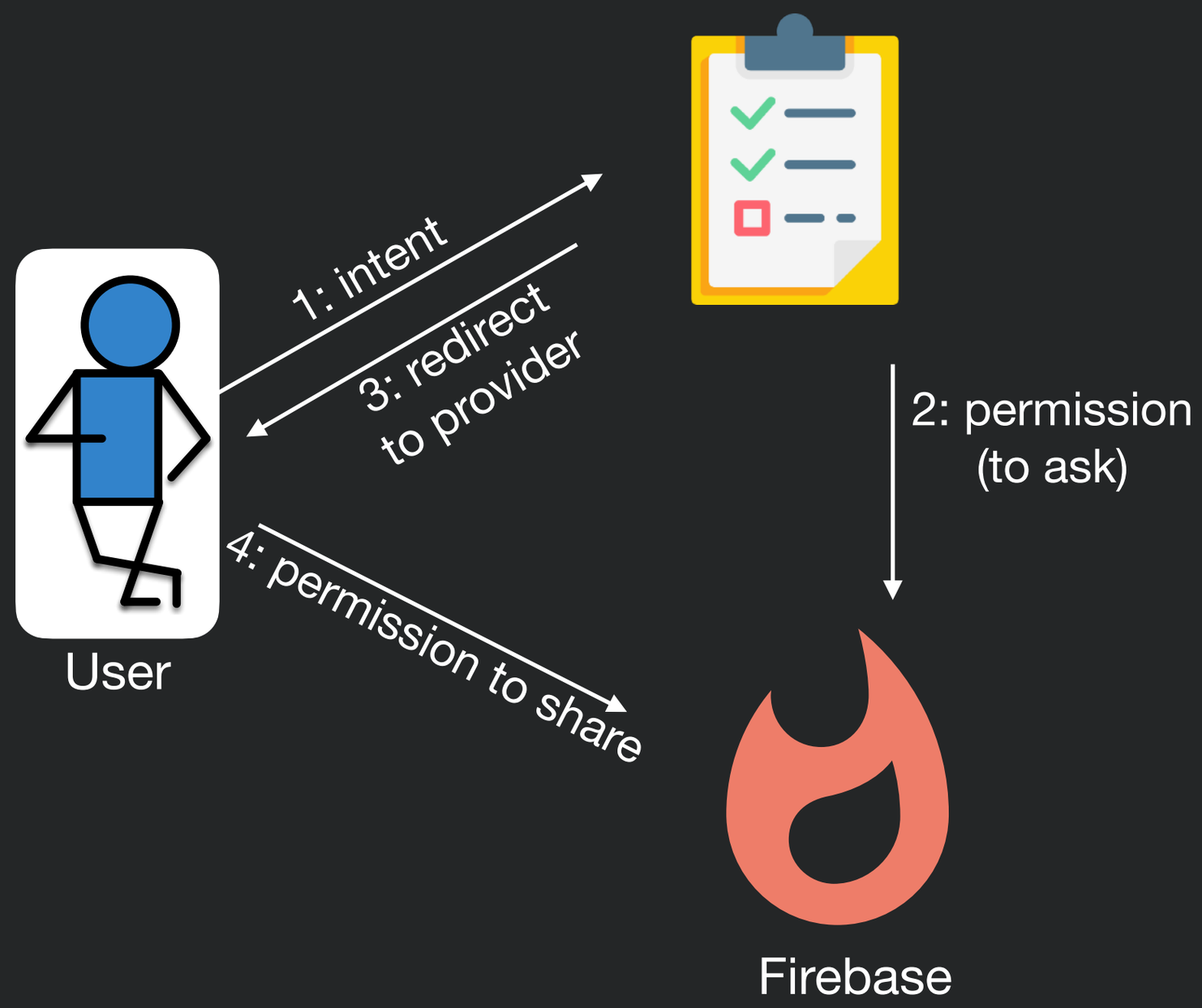
Using an Authentication Service



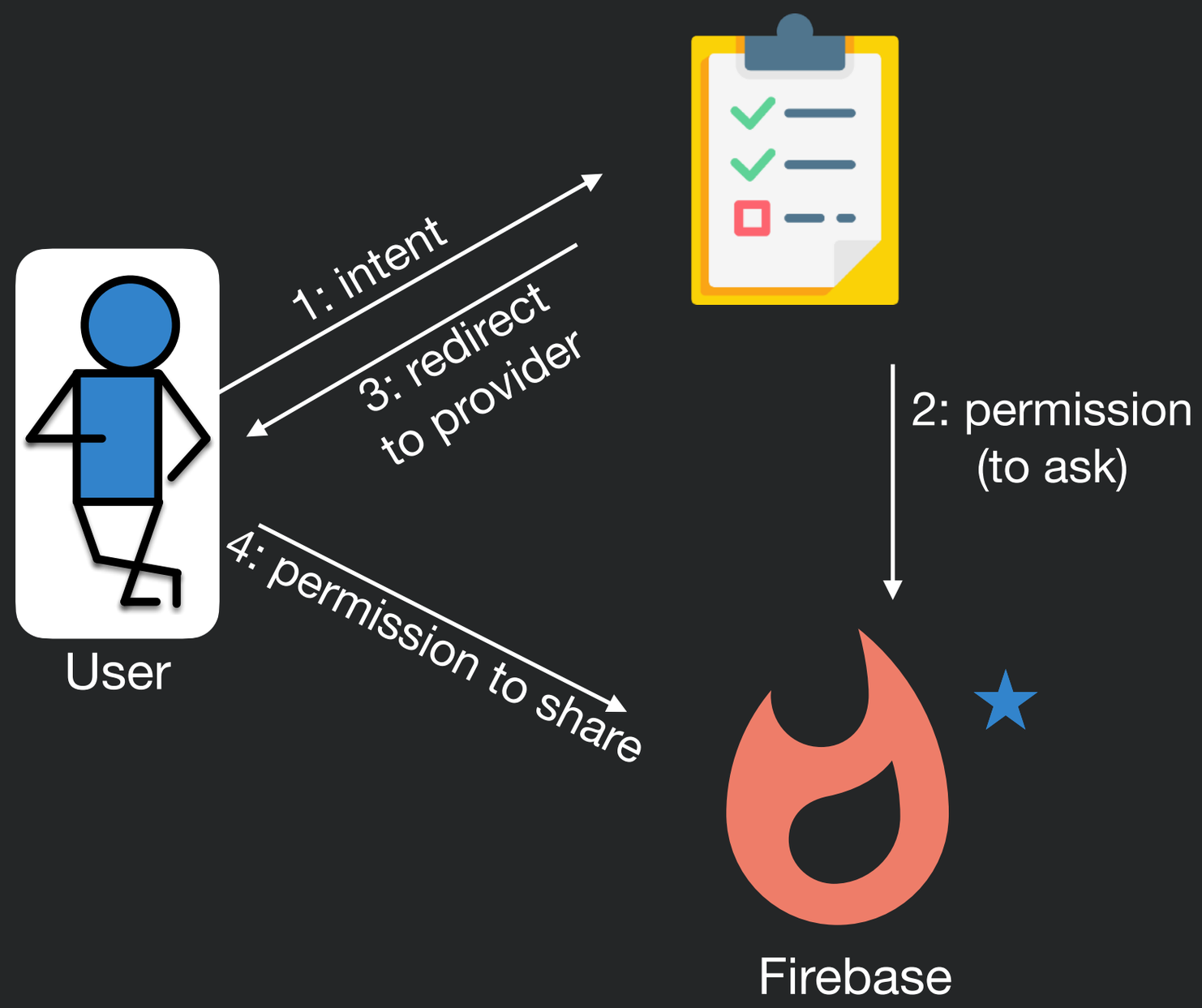
Using an Authentication Service



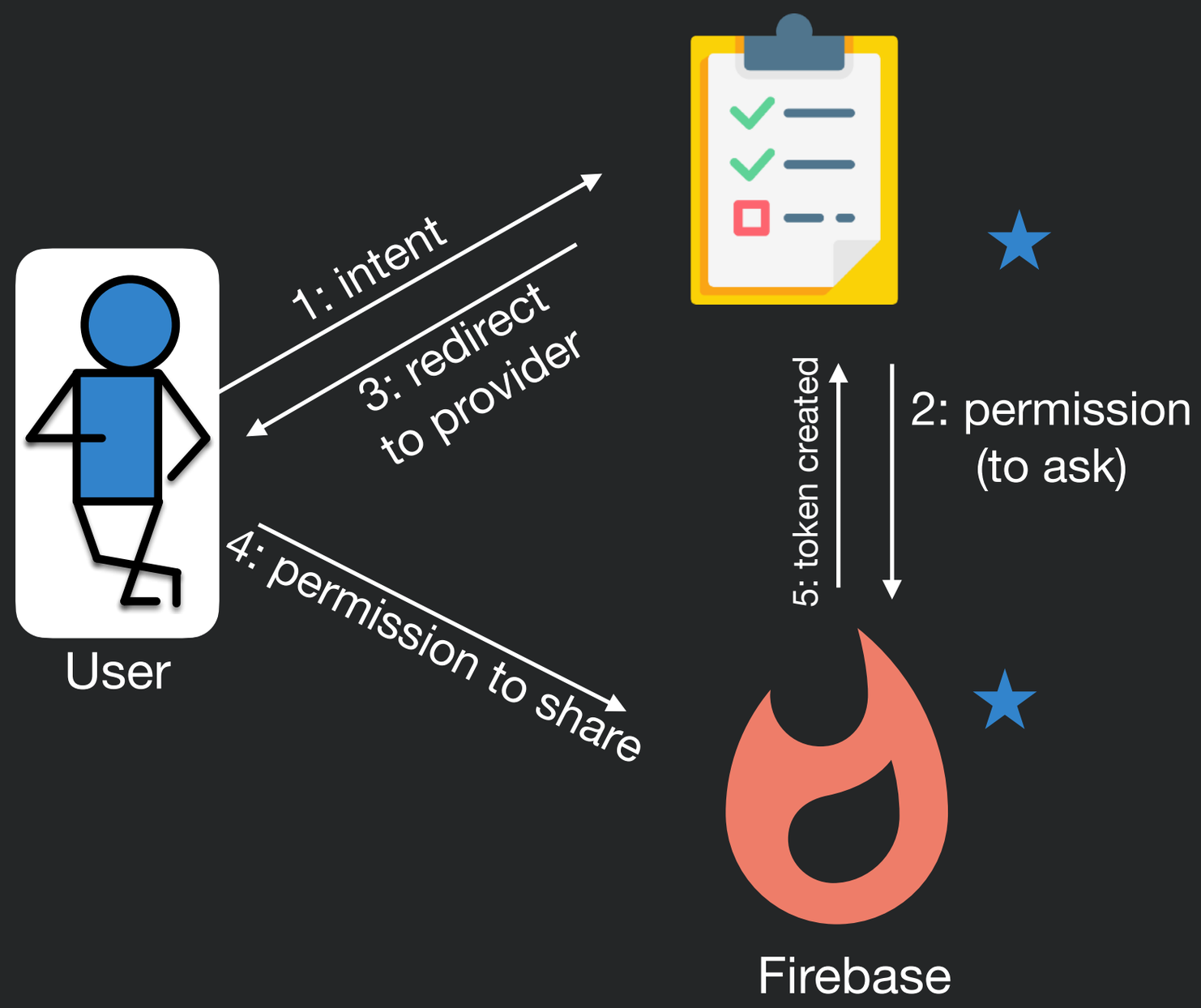
Using an Authentication Service



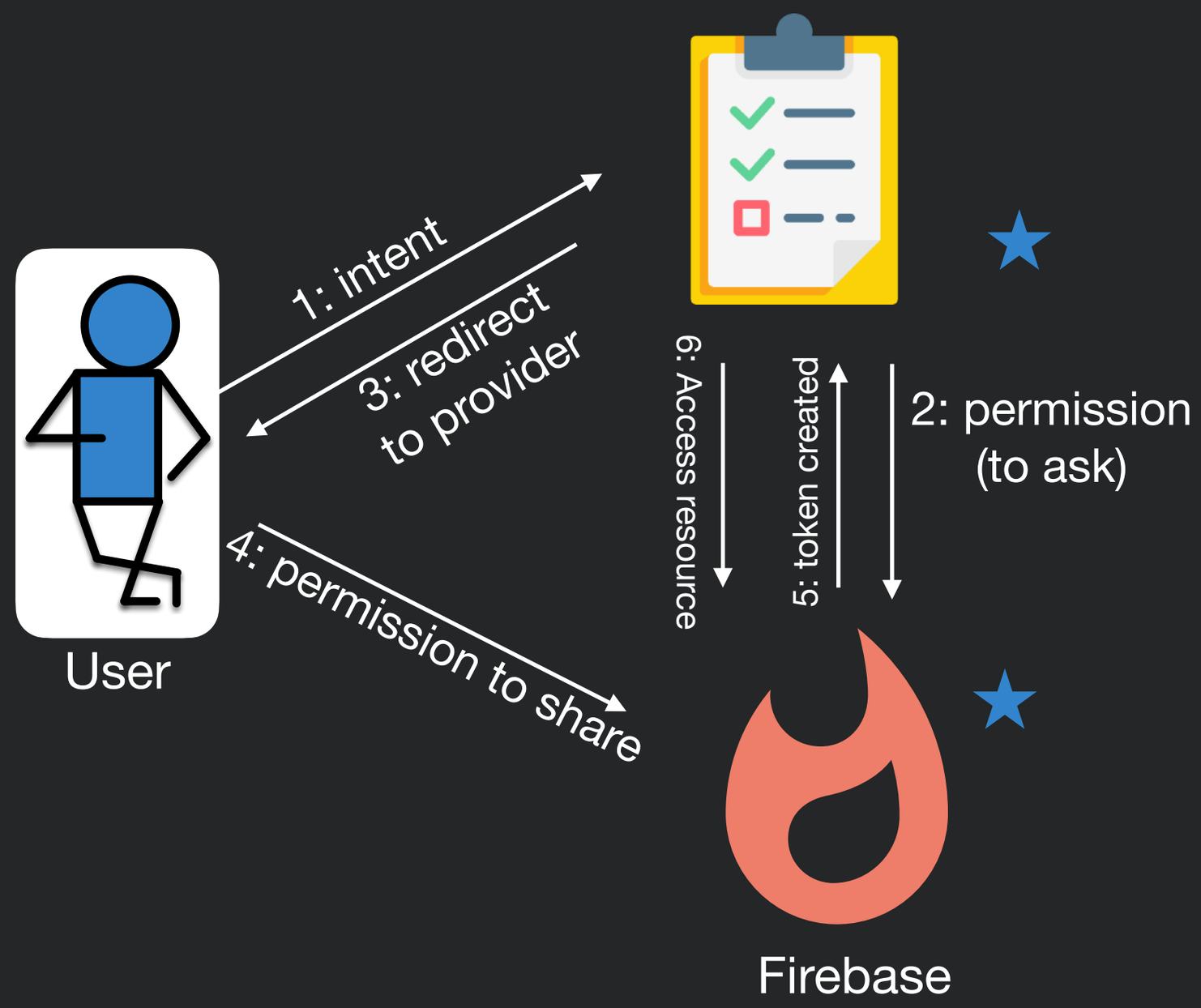
Using an Authentication Service



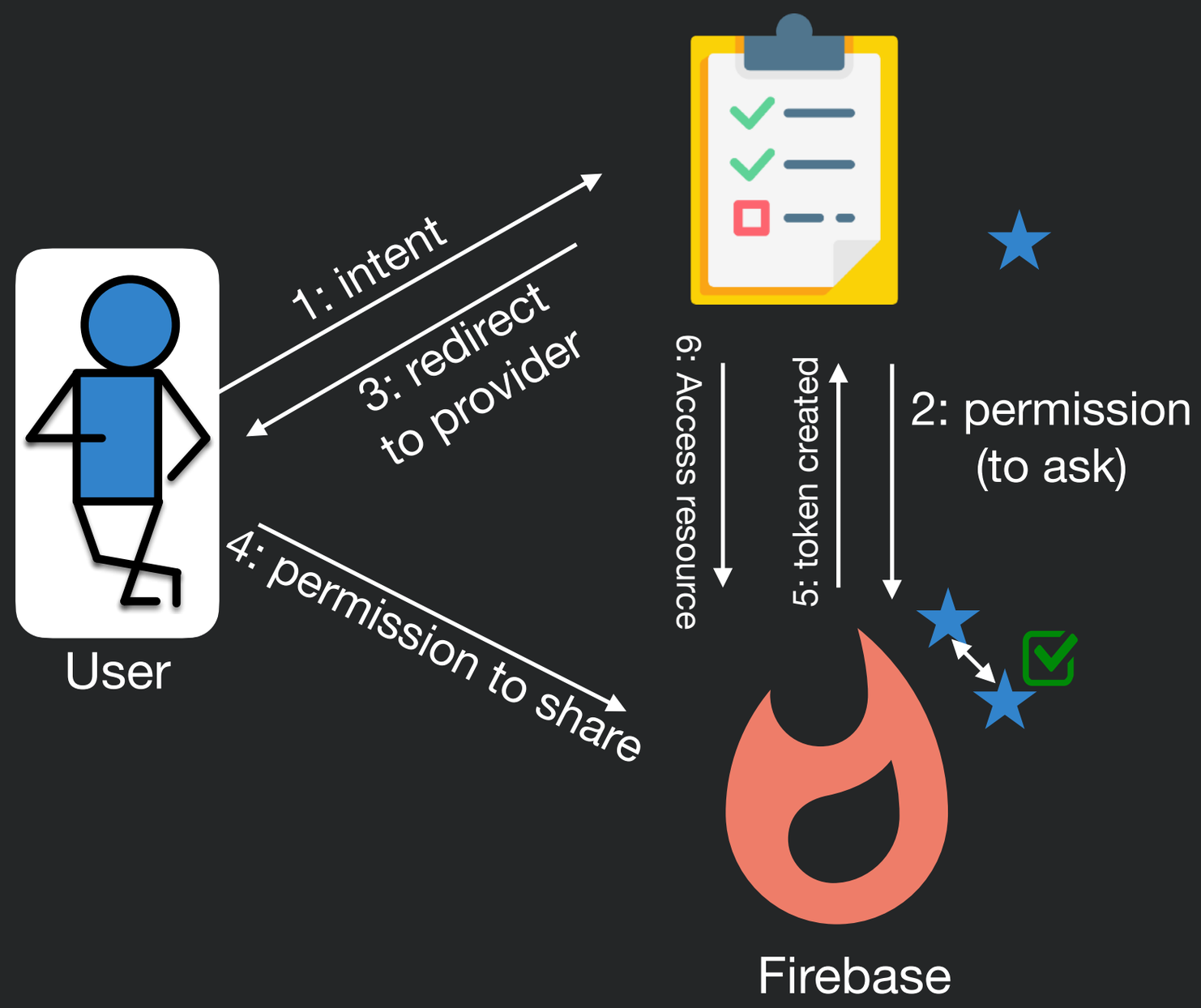
Using an Authentication Service



Using an Authentication Service



Using an Authentication Service



Firebase Authentication

- Firebase provides an entire suite of authentication services you can use to build into your app
- Can either use “federated” logins (e.g. login with google, facebook, GitHub credentials) or simple email/password logins. Use whichever you want.
- Getting started guide: https://github.com/firebase/FirebaseUI-Web
- Firebase handles browser local storage to track that the user is logged in across pages (woo)

Top 3 Web Vulnerabilities

- OWASP collected data on vulnerabilities
  - Surveyed 7 firms specializing in web app security
  - Collected 500,000 vulnerabilities across hundreds of apps and thousands of firms
  - Prioritized by prevalence as well as exploitability, detectability, impact

https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project

#3 - XSS: Cross Site Scripting

- User input that contains a *client-side* script that does not belong
  - A todo item:

```
/><script>alert("LASAGNA FOR PRESIDENT”);</script>
```

- Works when user input is used to render DOM elements without being escaped properly
- User input saved to server may be served to other users
  - Enables malicious user to execute code on other’s users browser
  - e.g., click ‘Buy’ button to buy a stock, send password data to third party, …

#2 - Broken Authentication and Session Management

- Building authentication is hard
  - Logout, password management, timeouts, secrete questions, account updates, …
- Vulnerability may exist if
  - User authentication credentials aren't protected when stored using hashing or encryption.
  - Credentials can be guessed or overwritten through weak account management functions (e.g., account creation, change password, recover password, weak session IDs).
  - Session IDs are exposed in the URL (e.g., URL rewriting).
  - Session IDs don't timeout, or user sessions or authentication tokens, particularly single sign-on (SSO) tokens, aren't properly invalidated during logout.
  - Session IDs aren't rotated after successful login.
  - Passwords, session IDs, and other credentials are sent over unencrypted connections.

#1 - Injection

- User input that contains *server-side* code that does not belong
- Usually comes up in context of SQL (which we aren't using)
  - e.g.,
  - `String query = "SELECT * FROM accounts WHERE custID='" + request.getParameter("id") + "'";`
- Might come up in JS in context of eval
  - `eval(request.getParameter("code"));`
  - Obvious injection attack - don't do this!

Validating User Input

- Escape Strings that originate from user
- Type of escaping depends on where data will be used
  - HTML - HTML entity encoding
  - URL - URL Escape
  - JSON - Javascript Escape
- Done automatically by some frameworks such as React
- More details: https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet

Authentication: Sharing Data Between Pages

- Browser loads many pages at the same time.
- Might want to share data between pages
  - Popup that wants to show details for data on main page
- Attack: malicious page
  - User visits a malicious page in a second tab
  - Malicious page steals data from page or its data, modifies data, or impersonates user

Solution: Same-Origin Policy

- Browser needs to differentiate pages that are part of same application from unrelated pages
- What makes a page similar to another page?
  - Origin: the **protocol**, **host**, and **port**

    http://www.example.com/dir/page.html
  - Different origins:

    https://www.example.com/dir/page.html

    http://www.example.com:80/dir/page.html

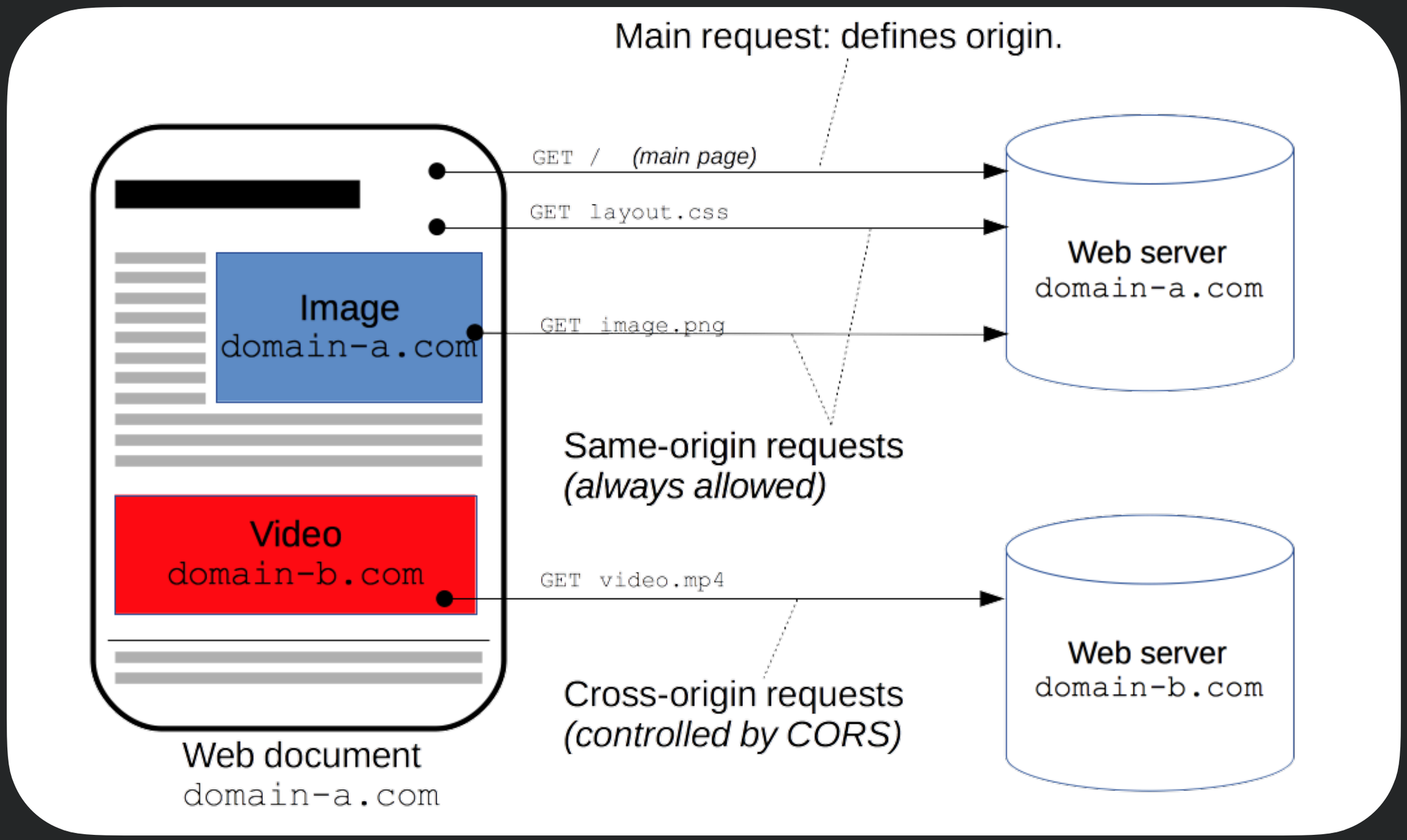
    http://en.example.com:80/dir/page.html

https://en.wikipedia.org/wiki/Same-origin_policy

Same-Origin Policy

- "Origin" refers to the *page that is executing it*, NOT where the data comes from
  - Example:
    - In one HTML file, I directly include 3 JS scripts, each loaded from a different server
    - -> All have same "origin"
  - Example:
    - One of those scripts makes an AJAX call to yet another server
    - -> AJAX call not allowed
- Scripts contained in a page may access data in a second web page (e.g., its DOM) if they come from the same origin

Cross Origin Requests



https://developer.mozilla.org/en-US/docs/Web/HTTP/Access_control_CORS

CORS: Cross Origin Resource Sharing

- Same-Origin might be safer, but not really usable:
  - How do we make AJAX calls to other servers?
- Solution: Cross Origin Resource Sharing (CORS)
- HTTP header:

```
Access-Control-Allow-Origin: <server or wildcard>
```

- In Express:

```
res.header("Access-Control-Allow-Origin", "*");
```

Takeaways

- Think about all potential threat models
  - Which do you care about
  - Which do you not care about
- What user data are you retaining
  - Who are you sharing it with, and what might they do with it

SWE 432 - Web Application Development

George Mason University

Instructor:
Dr. Kevin Moran

Teaching Assistant:
David Gonzalez Samudio

Class will start in:

10:00

SWE 432 - Web Application Development

George Mason University

Instructor:
Dr. Kevin Moran

Teaching Assistant:
David Gonzalez Samudio

Class will start in:

10:00

Templates, Databinding, & HTML

Today

- HTML
- Frontend JavaScript
- Intro to templating and React

HTML: HyperText Markup Language

- Language for describing *structure* of a document
- Denotes hierarchy of elements
- What might be elements in this document?

HTML History

HTML History

- 1995: HTML 2.0. Published as standard with RFC 1866

HTML History

- 1995: HTML 2.0. Published as standard with RFC 1866
- 1997: HTML 4.0 Standardized most modern HTML element w/ W3C recommendation
  - Encouraged use of CSS for styling elements over HTML attributes

HTML History

- 1995: HTML 2.0. Published as standard with RFC 1866
- 1997: HTML 4.0 Standardized most modern HTML element w/ W3C recommendation
  - Encouraged use of CSS for styling elements over HTML attributes
- 2000: XHTML 1.0
  - Imposed stricter rules on HTML format
    - e.g., elements needed closing tag, attribute names in lowercase

HTML History

- 1995: HTML 2.0. Published as standard with RFC 1866
- 1997: HTML 4.0 Standardized most modern HTML element w/ W3C recommendation
  - Encouraged use of CSS for styling elements over HTML attributes
- 2000: XHTML 1.0
  - Imposed stricter rules on HTML format
    - e.g., elements needed closing tag, attribute names in lowercase
- 2014: HTML5 published as W3C recommendation
  - New features for capturing more *semantic* information and *declarative* description of behavior
    - e.g., Input constraints
    - e.g., New tags that explain *purpose* of content
  - Important changes to DOM

HTML Elements

```
<p lang="en-us">This is a paragraph in English.</p>
```

HTML Elements

This is a paragraph in English.

"Start a paragraph element"

Opening tag begins an HTML element. Opening tags must have a corresponding closing tag.

HTML Elements

This is a paragraph in English.

name value

"Start a paragraph element"

Opening tag begins an HTML element. Opening tags must have a corresponding closing tag.

"Set the language to English"

HTML attributes are name / value pairs that provide additional information about the contents of an element.

HTML Elements

This is a paragraph in English.

name value

"Start a paragraph element"

Opening tag begins an HTML element. Opening tags must have a corresponding closing tag.

"Set the language to English"

HTML attributes are name / value pairs that provide additional information about the contents of an element.

"End a paragraph element"

Closing tag ends an HTML element. All content between the tags and the tags themselves compromise an HTML element.

HTML Elements

<input type="text" />

Some HTML tags can be self closing, including a built-in closing tag.

```
<!-- This is a comment.
Comments can be multiline. -->
```

HTML Elements

<input type="text" />

"Begin and end input element"

Some HTML tags can be self closing, including a built-in closing tag.

```
<!-- This is a comment.
Comments can be multiline. -->
```

A Starter HTML Document

```html
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>Hello World Site</title>
</head>
<body>
    Hello world!
</body>
</html>
```

Hello world!

A Starter HTML Document

"Use HTML5 standards mode"

```html
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>Hello World Site</title>
</head>
<body>
    Hello world!
</body>
</html>
```

Hello world!

A Starter HTML Document

"Use HTML5 standards mode"

"HTML content"

```html
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>Hello World Site</title>
</head>
<body>
    Hello world!
</body>
</html>
```

Hello world!

A Starter HTML Document

"Use HTML5 standards mode"

"HTML content"

"Header"

Information *about* the page

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>Hello World Site</title>
</head>
<body>
    Hello world!
</body>
</html>
```

Hello world!

A Starter HTML Document

"Use HTML5 standards mode"

"HTML content"

"Header"

Information *about* the page

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>Hello World Site</title>
</head>
<body>
    Hello world!
</body>
</html>
```

Hello world!

"Interpret bytes as UTF-8 characters"

Includes both ASCII & international characters.

A Starter HTML Document

"Use HTML5 standards mode"

"HTML content"

"Header"

Information *about* the page

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>Hello World Site</title>
</head>
<body>
    Hello world!
</body>
</html>
```

Hello world!

"Interpret bytes as UTF-8 characters"

Includes both ASCII & international characters.

"Title"

Used by browser for title bar or tab.

A Starter HTML Document

"Use HTML5 standards mode"

"HTML content"

"Header"

Information *about* the page

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>Hello World Site</title>
</head>
<body>
    Hello world!
</body>
</html>
```

Hello world!

"Interpret bytes as UTF-8 characters"

Includes both ASCII & international characters.

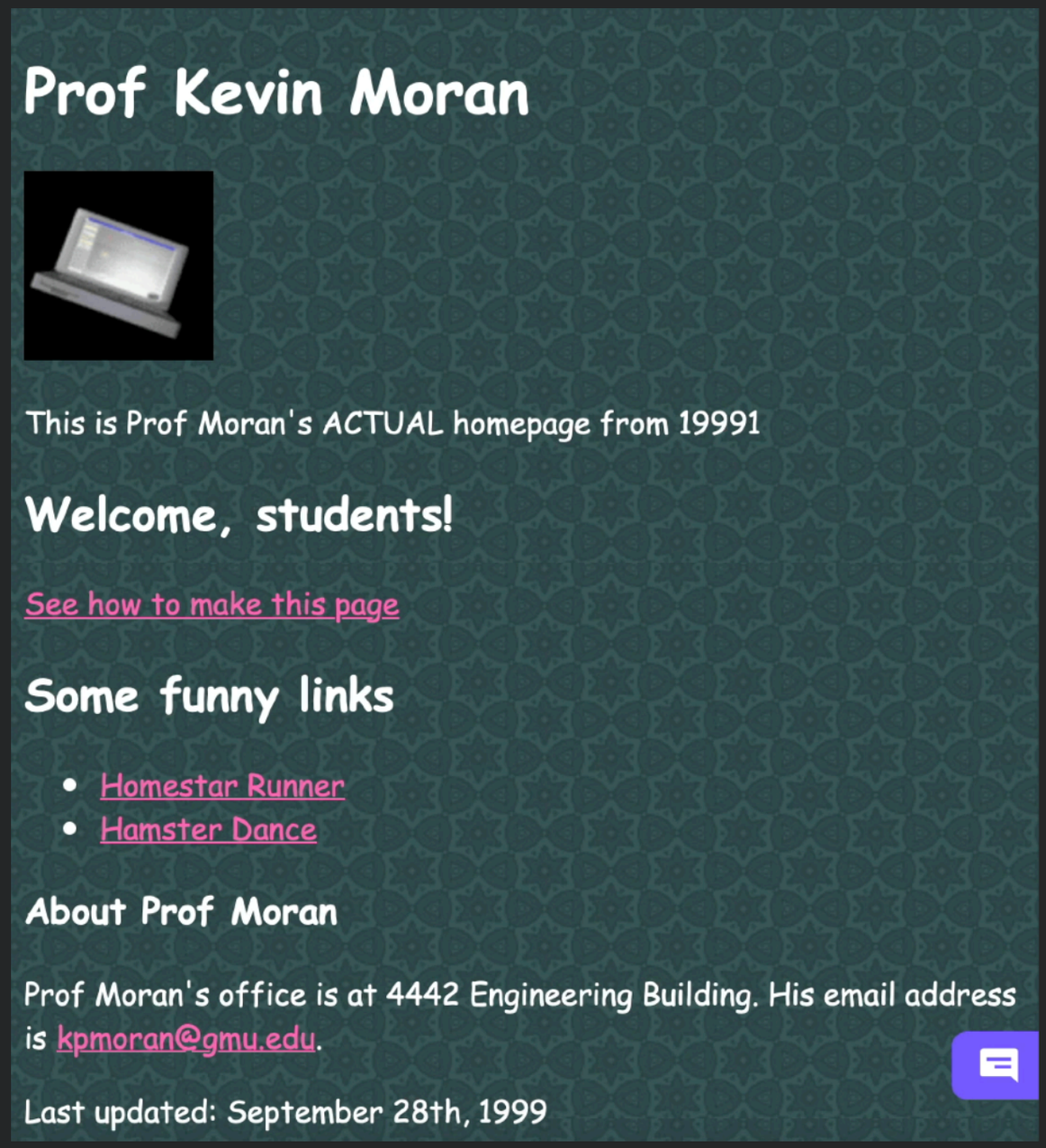
"Title"

Used by browser for title bar or tab.

"Document content"

HTML Example

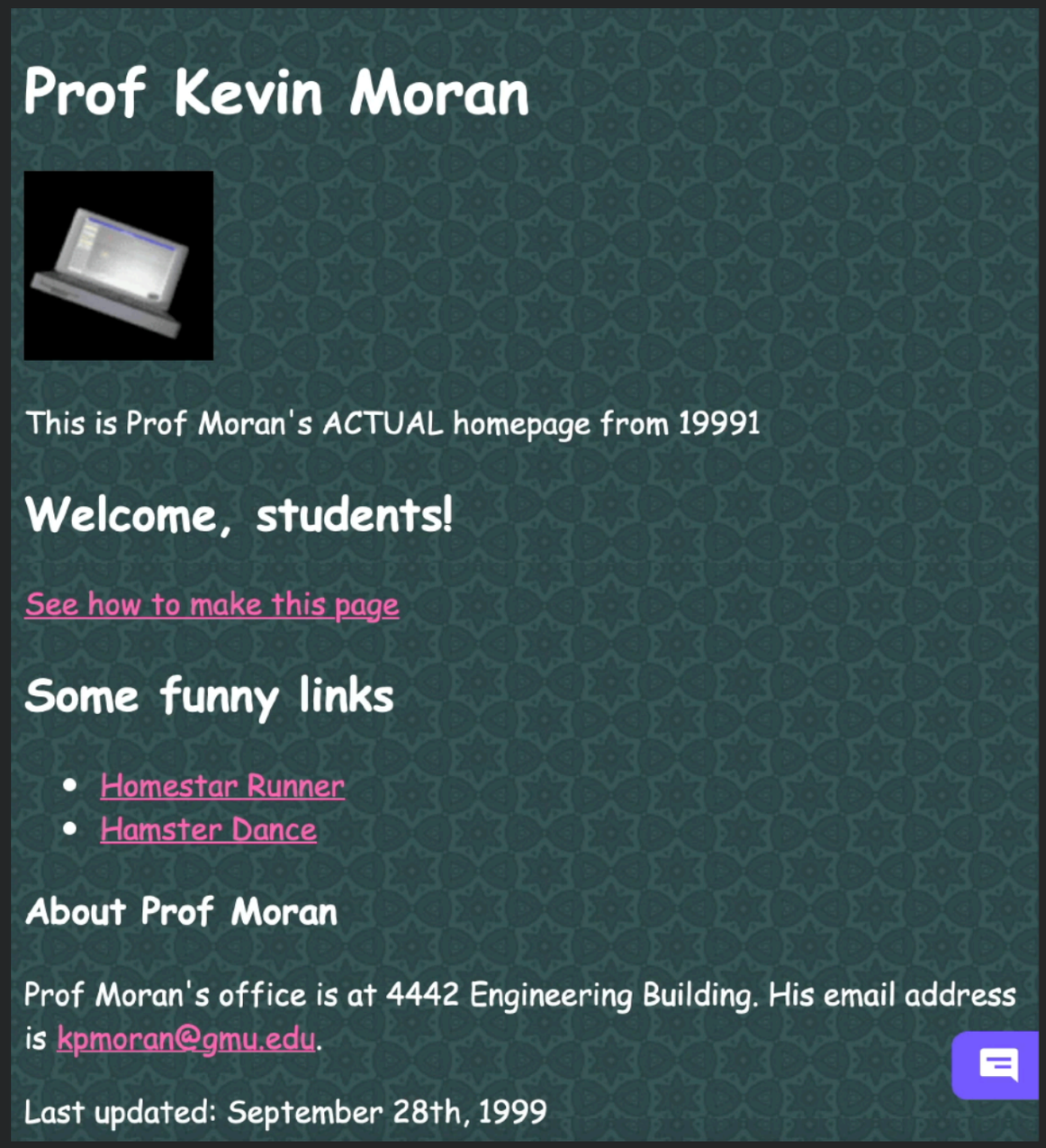
```html
<!DOCTYPE html>
<html>
  <head>
    <link rel="stylesheet" type="text/css" href="style.css">
    <title>Prof Moran's Webpage</title>
    </head>
    <body>
      <h1> Prof Kevin Moran </h1>
      <div>
        <p> <img alt="My really cool laptop"
        src="http://www.wonder-tonic.com/geocitiesizer/images/laptop-01.gif" />
        <br>
        <div class="marquee"> This is Prof Moran's ACTUAL homepage from 19991 </div> </p>
        <h2>Welcome, students!</h2>
        <p> <a href="https://www.youtube.com/watch?v=dOw4w9WgXW">See how to make this
        page</a> </p>
        <h2> Some funny links </h2>
<p>
<ul>
<li><a href="http://www.homestarrunner.com">Homestar Runner</a></li>
<li><a href="http://www.wb3w.net/The/6200riginal1420Hamsterdance.htm">Hamster
Dance</a></li> </ul>
</p> <h3> About Prof Moran </h3>

<p> Prof Moran's office is at 4442 Engineering Building. His email address is <a
href="mailto:kpmoran@gmu.edu">kpmoran@gmu.edu</a>. </p>

<p> Last updated: September 28th, 1999 </p> </div> </body>
</html>
```



https://replit.com/@kmoran/html-example#index.html

HTML Example

```html
<!DOCTYPE html>
<html>
  <head>
    <link rel="stylesheet" type="text/css" href="style.css">
    <title>Prof Moran's Webpage</title>
    </head>
    <body>
      <h1> Prof Kevin Moran </h1>
      <div>
        <p> <img alt="My really cool laptop"
        src="http://www.wonder-tonic.com/geocitiesizer/images/laptop-01.gif" />
        <br>
        <div class="marquee"> This is Prof Moran's ACTUAL homepage from 19991 </div> </p>
        <h2>Welcome, students!</h2>
        <p> <a href="https://www.youtube.com/watch?v=dOw4w9WgXW">See how to make this
        page</a> </p>
        <h2> Some funny links </h2>
<p>
<ul>
<li><a href="http://www.homestarrunner.com">Homestar Runner</a></li>
<li><a href="http://www.wb3w.net/The/6200riginal1420Hamsterdance.htm">Hamster
Dance</a></li> </ul>
</p> <h3> About Prof Moran </h3>

<p> Prof Moran's office is at 4442 Engineering Building. His email address is <a
href="mailto:kpmoran@gmu.edu">kpmoran@gmu.edu</a>. </p>

<p> Last updated: September 28th, 1999 </p> </div> </body>
</html>
```



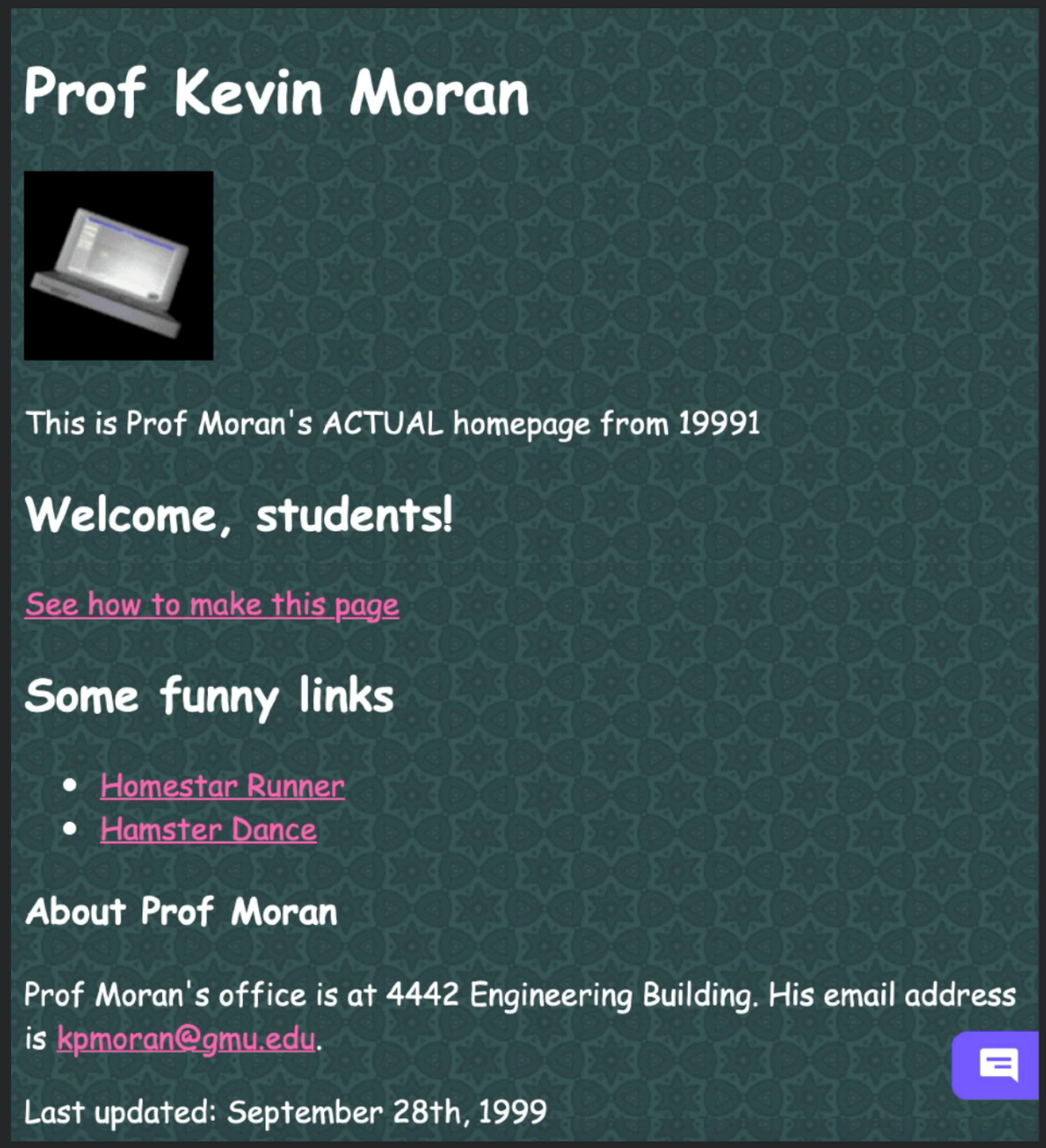
https://replit.com/@kmoran/html-example#index.html

HTML Example

```html
<!DOCTYPE html>
<html>
  <head>
    <link rel="stylesheet" type="text/css" href="style.css">
    <title>Prof Moran's Webpage</title>
```

Use <h1>, <h2>, …, <h5> for headings

```html
src="http://www.wonder-tonic.com/geocitiesizer/images/laptop-01.gif" />
<br>
<div class="marquee"> This is Prof Moran's ACTUAL homepage from 19991 </div> </p>
<h2>Welcome, students!</h2>
<p> <a href="https://www.youtube.com/watch?v=dOw4w9WgXW">See how to make this
page</a> </p>
<h2> Some funny links </h2>
<p>
<ul>
<li><a href="http://www.homestarrunner.com">Homestar Runner</a></li>
<li><a href="http://www.wb3w.net/The/6200riginal1420Hamsterdance.htm">Hamster
Dance</a></li> </ul>
</p> <h3> About Prof Moran </h3>

<p> Prof Moran's office is at 4442 Engineering Building. His email address is <a
href="mailto:kpmoran@gmu.edu">kpmoran@gmu.edu</a>. </p>

<p> Last updated: September 28th, 1999 </p> </div> </body>
</html>
```



https://replit.com/@kmoran/html-example#index.html

HTML Example

```html
<!DOCTYPE html>
<html>
  <head>
    <link rel="stylesheet" type="text/css" href="style.css">
    <title>Prof Moran's Webpage</title>
    </head>
    <body>
      <h1> Prof Kevin Moran </h1>
      <div>
        <p> <img alt="My really cool laptop"
        src="http://www.wonder-tonic.com/geocitiesizer/images/laptop-01.gif" />
        <br>
        <div class="marquee"> This is Prof Moran's ACTUAL homepage from 19991 </div> </p>
        <h2>Welcome, students!</h2>
        <p> <a href="https://www.youtube.com/watch?v=dOw4w9WgXW">See how to make this
        page</a> </p>
        <h2> Some funny links </h2>
<p>
<ul>
<li><a href="http://www.homestarrunner.com">Homestar Runner</a></li>
<li><a href="http://www.wb3w.net/The/6200riginal1420Hamsterdance.htm">Hamster
Dance</a></li> </ul>
</p> <h3> About Prof Moran </h3>

<p> Prof Moran's office is at 4442 Engineering Building. His email address is <a
href="mailto:kpmoran@gmu.edu">kpmoran@gmu.edu</a>. </p>

<p> Last updated: September 28th, 1999 </p> </div> </body>
</html>
```

Prof Kevin Moran

This is Prof Moran's ACTUAL homepage from 19991

Welcome, students!

See how to make this page

Some funny links

- Homestar Runner
- Hamster Dance

About Prof Moran

Prof Moran's office is at 4442 Engineering Building. His email address is kpmoran@gmu.edu.

Last updated: September 28th, 1999

https://replit.com/@kmoran/html-example#index.html

HTML Example

```html
<!DOCTYPE html>
<html>
  <head>
    <link rel="stylesheet" type="text/css" href="style.css">
    <title>Prof Moran's Webpage</title>
    </head>
    <body>
      <h1> Prof Kevin Moran </h1>
      <div>
        <p> <img alt="My really cool laptop"
        src="http://www.wonder-tonic.com/geocitiesizer/images/laptop-01.gif" />
        <br>
        <div class="marquee"> This is Prof Moran's ACTUAL homepage from 19991 </div> </p>
        <h2>Welcome, students!</h2>
        <p> <a href="https://www.youtube.com/watch?v=dOw4w9WgXW">See how to make this
        page</a> </p>
        <h2> Some funny links </h2>
<p>
<ul>
<li><a href="http://www.homestarrunner.com">Homestar Runner</a></li>
<li><a href="http://www.wb3w.net/The/6200riginal1420Hamsterdance.htm">Hamster
Dance</a></li> </ul>
</p> <h3> About Prof Moran </h3>

<p> Prof Moran's office is at 4442 Engineering Building. His email address is <a
href="mailto:kpmoran@gmu.edu">kpmoran@gmu.edu</a>. </p>

<p> Last updated: September 28th, 1999 </p> </div> </body>
</html>
```

Prof Kevin Moran

This is Prof Moran's ACTUAL homepage from 19991

Welcome, students!

See how to make this page

Some funny links

- Homestar Runner
- Hamster Dance

About Prof Moran

Prof Moran's office is at 4442 Engineering Building. His email address is kpmoran@gmu.edu.

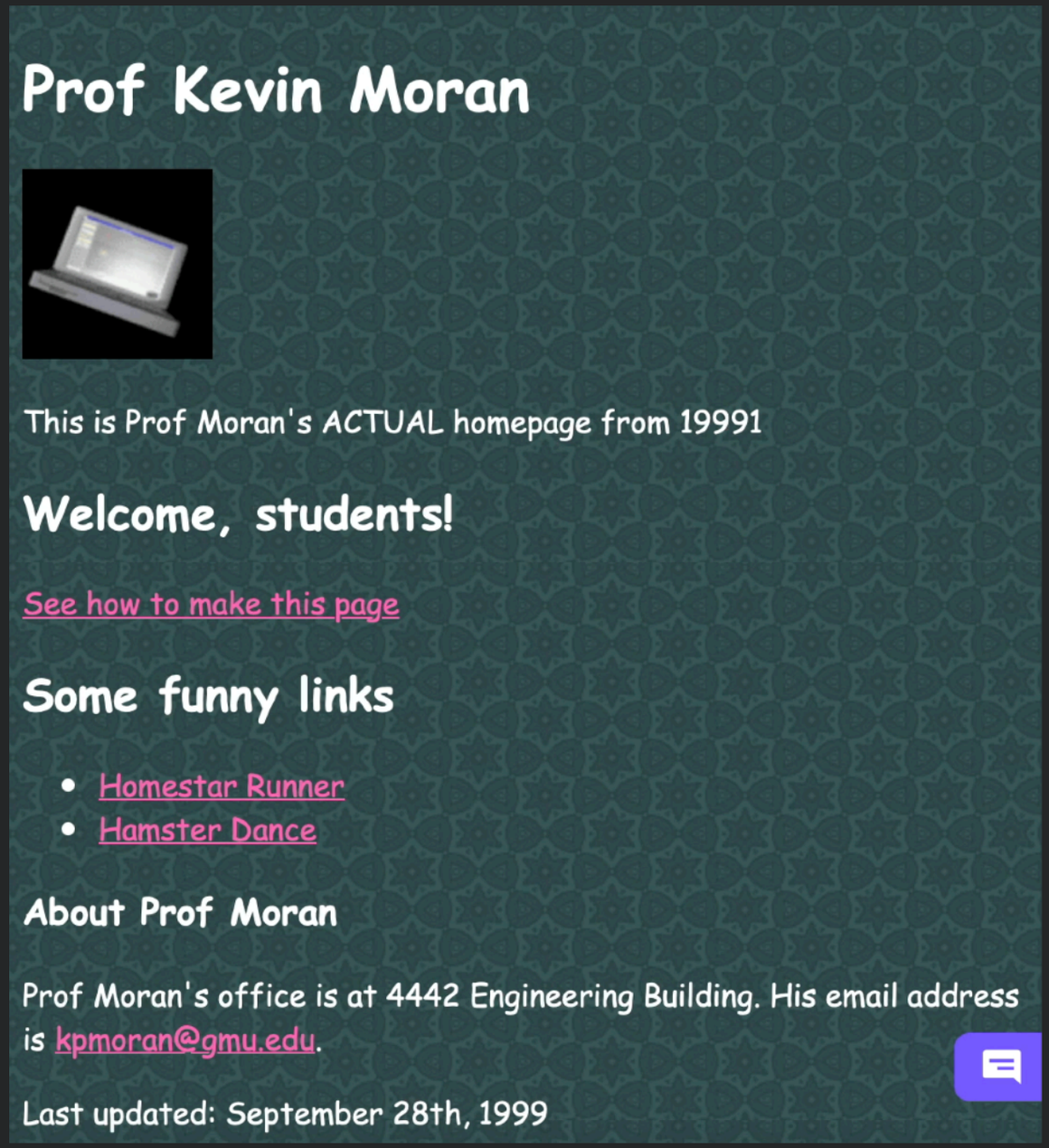
Last updated: September 28th, 1999

https://replit.com/@kmoran/html-example#index.html

HTML Example

```
<!DOCTYPE html>
<html>
  <head>
    <link rel="stylesheet" type="text/css" href="style.css">
    <title>Prof Moran's Webpage</title>
    </head>
    <body>
      <h1> Prof Kevin Moran </h1>
      <div>
        <p> <img alt="My really cool laptop"
        src="http://www.wonder-tonic.com/geocitiesizer/images/laptop-01.gif" />
        <br>
        <div class="marquee"> This is Prof Moran's ACTUAL homepage from 19991 </div> </p>

                                                                                this

<p
<u
<l
<li><a href="http://www.wb3w.net/The/6200riginal1420Hamsterdance.htm">Hamster
Dance</a></li> </ul>
</p> <h3> About Prof Moran </h3>

<p> Prof Moran's office is at 4442 Engineering Building. His email address is <a
href="mailto:kpmoran@gmu.edu">kpmoran@gmu.edu</a>. </p>

<p> Last updated: September 28th, 1999 </p> </div> </body>
</html>
```

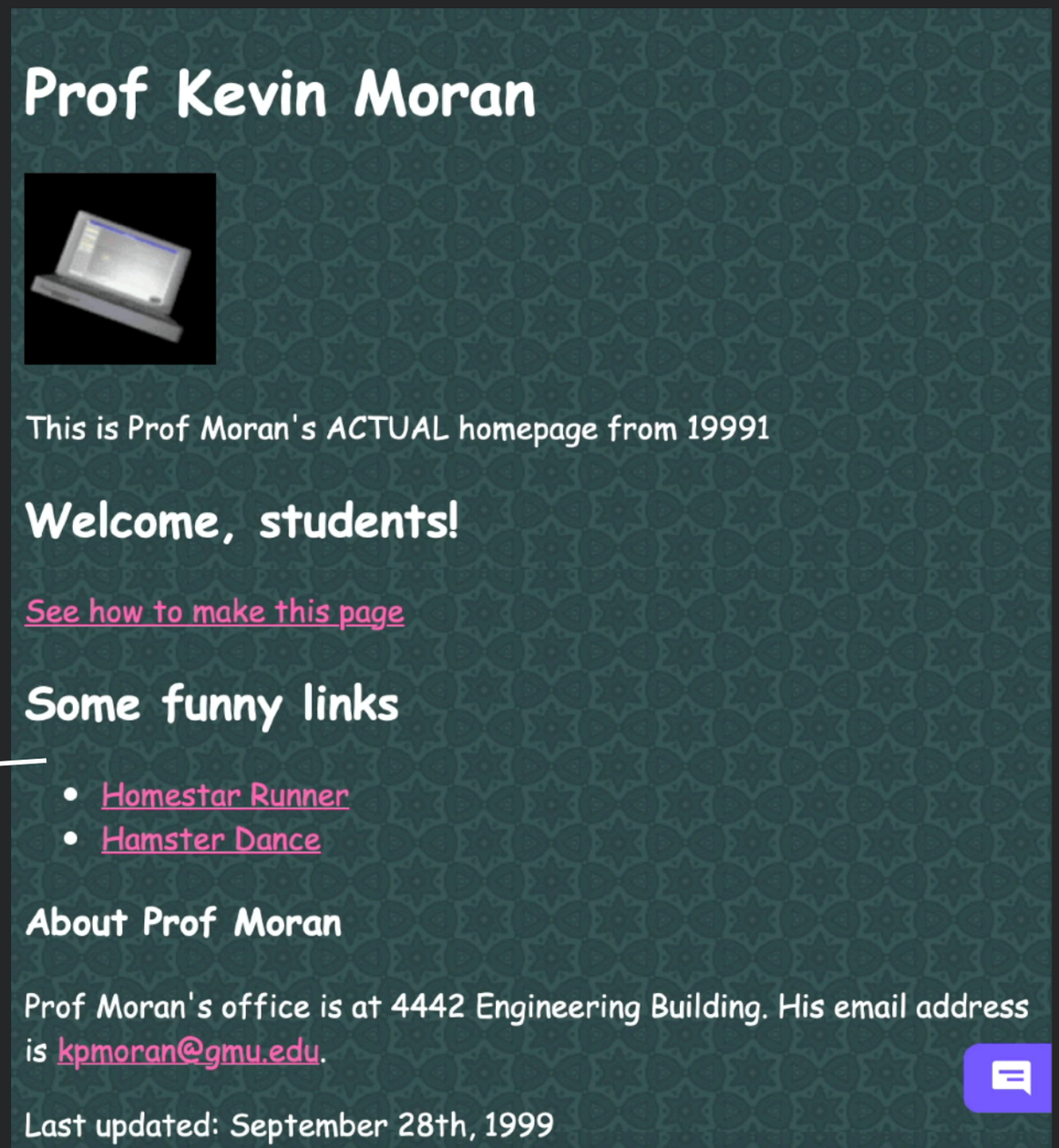
Paragraphs (<p>) consist of related content. By default, each paragraph starts on a new line.



https://replit.com/@kmoran/html-example#index.html

HTML Example

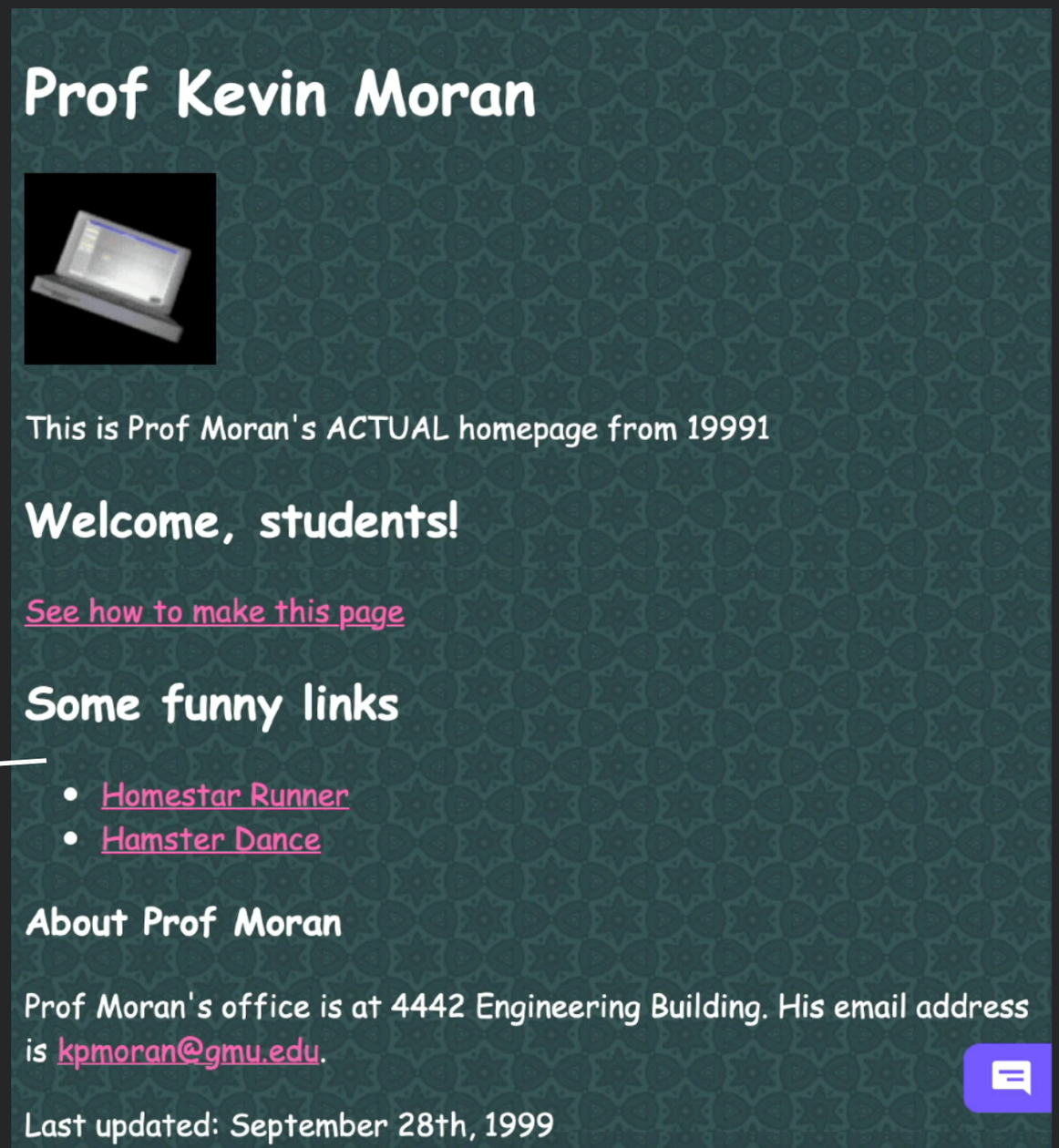
```html
<!DOCTYPE html>
<html>
  <head>
    <link rel="stylesheet" type="text/css" href="style.css">
    <title>Prof Moran's Webpage</title>
    </head>
    <body>
      <h1> Prof Kevin Moran </h1>
      <div>
        <p> <img alt="My really cool laptop"
        src="http://www.wonder-tonic.com/geocitiesizer/images/laptop-01.gif" />
        <br>
        <div class="marquee"> This is Prof Moran's ACTUAL homepage from 19991 </div> </p>
        <h2>Welcome, students!</h2>
        <p> <a href="https://www.youtube.com/watch?v=dOw4w9WgXW">See how to make this
        page</a> </p>
        <h2> Some funny links </h2>
<p>
<ul>
<li><a href="http://www.homestarrunner.com">Homestar Runner</a></li>
<li><a href="http://www.wb3w.net/The/6200riginal1420Hamsterdance.htm">Hamster
Dance</a></li> </ul>
</p> <h3> About Prof Moran </h3>

<p> Prof Moran's office is at 4442 Engineering Building. His email address is <a
href="mailto:kpmoran@gmu.edu">kpmoran@gmu.edu</a>. </p>

<p> Last updated: September 28th, 1999 </p> </div> </body>
</html>
```



https://replit.com/@kmoran/html-example#index.html

HTML Example

```html
<!DOCTYPE html>
<html>
  <head>
    <link rel="stylesheet" type="text/css" href="style.css">
    <title>Prof Moran's Webpage</title>
    </head>
    <body>
      <h1> Prof Kevin Moran </h1>
      <div>
        <p> <img alt="My really cool laptop"
        src="http://www.wonder-tonic.com/geocitiesizer/images/laptop-01.gif" />
        <br>
        <div class="marquee"> This is Prof Moran's ACTUAL homepage from 19991 </div> </p>
        <h2>Welcome, students!</h2>
        <p> <a href="https://www.youtube.com/watch?v=dOw4w9WgXW">See how to make this
        page</a> </p>
        <h2> Some funny links </h2>
<p>
<ul>
<li><a href="http://www.homestarrunner.com">Homestar Runner</a></li>
<li><a href="http://www.wb3w.net/The/6200riginal1420Hamsterdance.htm">Hamster
Dance</a></li> </ul>
</p> <h3> About Prof Moran </h3>

<p> Prof Moran's office is at 4442 Engineering Building. His email address is <a
href="mailto:kpmoran@gmu.edu">kpmoran@gmu.edu</a>. </p>

<p> Last updated: September 28th, 1999 </p> </div> </body>
</html>
```



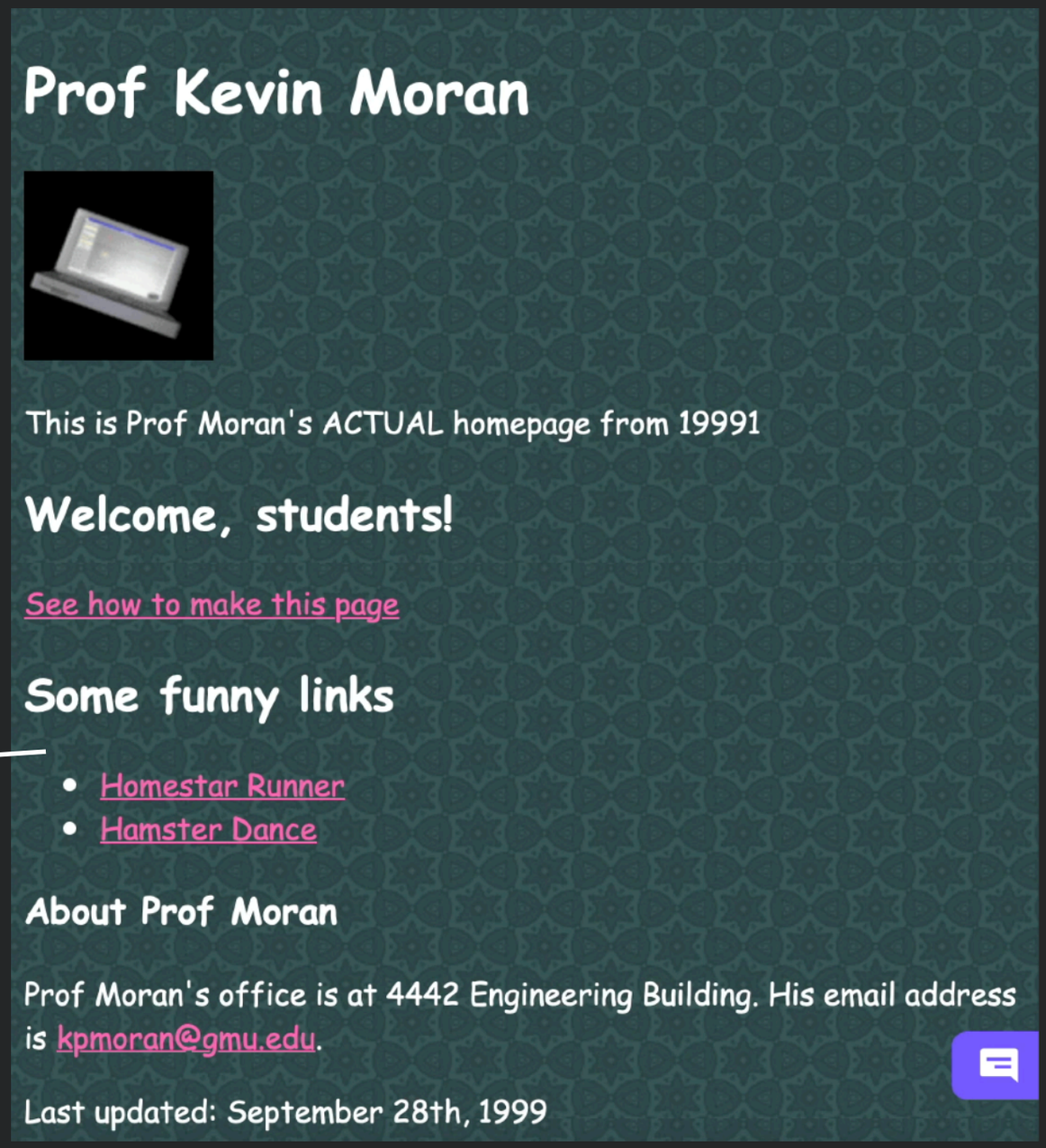
https://replit.com/@kmoran/html-example#index.html

HTML Example

```html
<!DOCTYPE html>
<html>
  <head>
    <link rel="stylesheet" type="text/css" href="style.css">
    <title>Prof Moran's Webpage</title>
    </head>
    <body>
      <h1> Prof Kevin Moran </h1>
      <div>
        <p> <img alt="My really cool laptop"
        src="http://www.wonder-tonic.com/geocitiesizer/images/laptop-01.gif" />
        <br>
        <div class="marquee"> This is Prof Moran's ACTUAL homepage from 19991 </div> </p>
        <h2>Welcome, students!</h2>
        <p> <a href="https://www.youtube.com/watch?v=dOw4w9WgXW">See how to make this
```

Unordered lists (<ul>) consist of list items (<li>) that each start on a new line. Lists can be nested arbitrarily deep.

```html
<p> Prof Moran's office is at 4442 Engineering Building. His email address is <a
href="mailto:kpmoran@gmu.edu">kpmoran@gmu.edu</a>. </p>

<p> Last updated: September 28th, 1999 </p> </div> </body>
</html>
```



https://replit.com/@kmoran/html-example#index.html

Text

```
9     <h1>Level 1 Heading</h1>
10    <h2>Level 2 Heading</h2>
11    <h3>Level 3 Heading</h3>
12    <h4>Level 4 Heading</h4>
13    <h5>Level 5 Heading</h5>
14    <h6>Level 5 Heading</h6>
15    Text can be made <b>bold</b> and
16        <i>italic</i>, or <sup>super</sup>
17        and <sub>sub</sub>scripts. White
18        space collapsing removes all
19        sequences of two more more spaces
20        and line breaks, allowing
21        the markup to use tabs
22        and whitespace for
23        organization.
24        Spaces can be added with
25            &amp;nbsp;.
26    <br/>New lines can be added with &lt
          ;BR/&gt;.
27
28    <p>A paragraph conssists of one or
          more sentences that form a self
          -contained unit of discourse. By
          default, a browser will show each
          paragraph on a new line.</p>
29
30    <hr/>
31    Text can also be offest with
32    horizontal rules.
33
34
```

Level 1 Heading

Level 2 Heading

Level 3 Heading

Level 4 Heading

Level 5 Heading

Level 5 Heading

Text can be made **bold** and *italic*, or ^{super} and $_{sub}$scripts.
White space collapsing removes all sequences of two more more spaces and line breaks, allowing the markup to use tabs and whitespace for organization. Spaces can be added with .
New lines can be added with <BR/>.

A paragraph conssists of one or more sentences that form a self-contained unit of discourse. By default, a browser will show each paragraph on a new line.

Text can also be offest with horizontal rules.

Semantic markup

Semantic markup

- Tags that can be used to denote the ***meaning*** of specific content

Semantic markup

- Tags that can be used to denote the ***meaning*** of specific content
- Examples
  - `<strong>` - An element that has importance.
  - `<blockquote>` - An element that is a longer quote.
  - `<q>` - A shorter quote inline in paragraph.
  - `<abbr>` - Abbreviation
  - `<cite>` - Reference to a work.
  - `<dfn>` - The definition of a term.
  - `<address>` - Contact information.
  - `<ins><del>` - Content that was inserted or deleted.
  - `<s>` - Something that is no longer accurate.

Links

```
<a href="http://www.google.com">Absolute link</a><br/>
<a href="movies.html">Relative URL</a><br/>
<a href="mailto:tlatoza@gmu.edu">Email Prof. LaToza</a><br/>
<a href="http://www.google.com" target="_blank">Opens in new
    window</a><br/>
<a href="#idName">Navigate to HTML element idName</a>
```

Absolute link
Relative URL
Email Prof. LaToza
Opens in new window
Navigate to HTML element idName

Controls

```html
<p>Text Input: <input type="text" maxlength="5" /></p>
<p>Password Input: <input type="password" /></p>
<p>Search Input: <input type="search"></p>
<p>Text Area: <textarea>Initial text</textarea></p>
<p>Checkbox:
    <input type="checkbox" checked="checked" /> Checked
    <input type="checkbox" /> Unchecked
</p>
<p>Drop Down List Box:
    <select>
        <option>Option1</option>
        <option selected="selected">Option2</option>
    </select>
</p>
<p>Multiple Select Box:
    <select multiple="multiple">
        <option>Option1</option>
        <option selected="selected">Option2</option>
    </select>
</p>
<p>File Input Box: <input type="file" />
<p>Image Button: <input type="image" src="http://cs.gmu.edu/~tlatoza
    /images/reachabilityQuestion.jpg" width="50"></p>
<p>Button: <button>Button</button></p>
<p>Range Input: <input type="range" min="0" max="100" step="10"
    value="30" /></p>
```

Text Input:

Password Input:

Search Input:

Text Area: Initial text

Checkbox: ☑ Checked ☐ Unchecked

Drop Down List Box: Option2

Multiple Select Box: Option1 Option2

File Input Box: Choose File No file chosen

Image Button:

Button: Button

Range Input:

Controls

```
<p>Text Input: <input type="text" maxlength="5" /></p>
<p>Password Input: <input type="password" /></p>
<p>Search Input: <input type="search"></p>
<p>Text Area: <textarea>Initial text</textarea></p>
<p>Checkbox:
    <input type="checkbox" checked="checked" /> Checked
    <input type="checkbox" /> Unchecked
</p>
<p>Drop Down List Box:
    <select>
        <option>Option1</option>
        <option selected="selected">Option2</option>
    </select>
</p>
<p>Multiple Select Box:
    <select multiple="multiple">
        <option>Option1</option>
        <option selected="selected">Option2</option>
    </select>
</p>
<p>File Input Box: <input type="file" />
<p>Image Button: <input type="image" src="http://cs.gmu.edu/~tlatoza
    /images/reachabilityQuestion.jpg" width="50"></p>
<p>Button: <button>Button</button></p>
<p>Range Input: <input type="range" min="0" max="100" step="10"
    value="30" /></p>
```

Search input provides clear button

Text Input:

Password Input:

Search Input:

Text Area: Initial text

Checkbox: ☑ Checked ☐ Unchecked

Drop Down List Box: Option2

Multiple Select Box: Option1 Option2

File Input Box: Choose File No file chosen

Image Button:

Button: Button

Range Input:

Block vs. Inline Elements

Block elements

Block elements appear on a new line.
Examples: <h1><p>
<table><form>

Inline elements

Inline elements appear to continue on the same line.
Examples: <a><b><input><img>

Block vs. Inline Elements

Block elements

Block elements appear on a new line.
Examples: <h1><p>
<table><form>

Inline elements

Inline elements appear to continue on the same line.
Examples: <a><b><input><img>

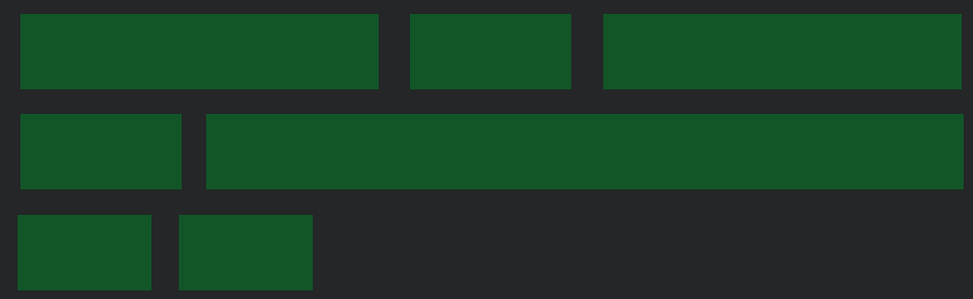
Block vs. Inline Elements

Block elements

Block elements appear on a new line.
Examples: <h1><p>
<table><form>

Inline elements

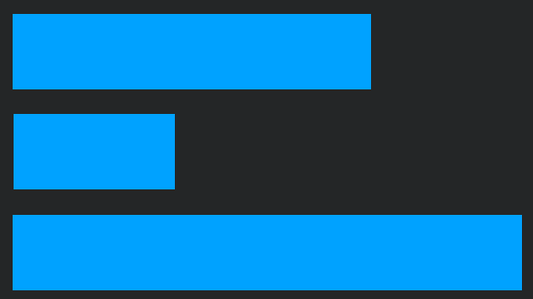
Inline elements appear to continue on the same line.
Examples: <a><b><input><img>



Block vs. Inline Elements

Block elements

Block elements appear on a new line.
Examples: <h1><p>
<table><form>



```
<h1>Hiroshi Sugimoto</h1>
<p>The dates for the ORIGIN OF ART exhbibition are as
    follows:</p>
<ul>
    <li>Science: 21 Nov- 20 Feb 2010/2011</li>
    <li>Architecture: 6 Mar - 15 May 2011</li>
</ul>
```

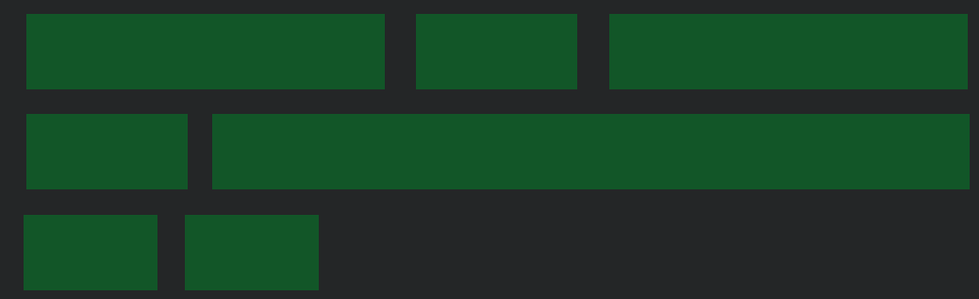
Hiroshi Sugimoto

The dates for the ORIGIN OF ART exhbibition are as follows:

- Science: 21 Nov- 20 Feb 2010/2011
- Architecture: 6 Mar - 15 May 2011

Inline elements

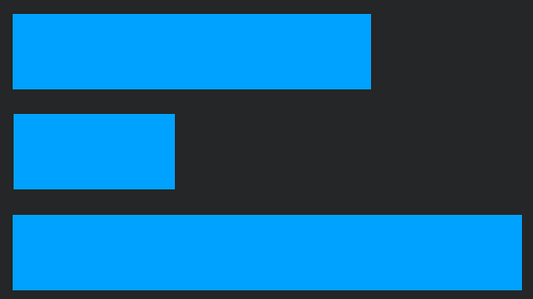
Inline elements appear to continue on the same line.
Examples: <a><b><input><img>



Block vs. Inline Elements

Block elements

Block elements appear on a new line.
Examples: <h1><p>
<table><form>



```
<h1>Hiroshi Sugimoto</h1>
<p>The dates for the ORIGIN OF ART exhbibition are as
    follows:</p>
<ul>
    <li>Science: 21 Nov- 20 Feb 2010/2011</li>
    <li>Architecture: 6 Mar - 15 May 2011</li>
</ul>
```

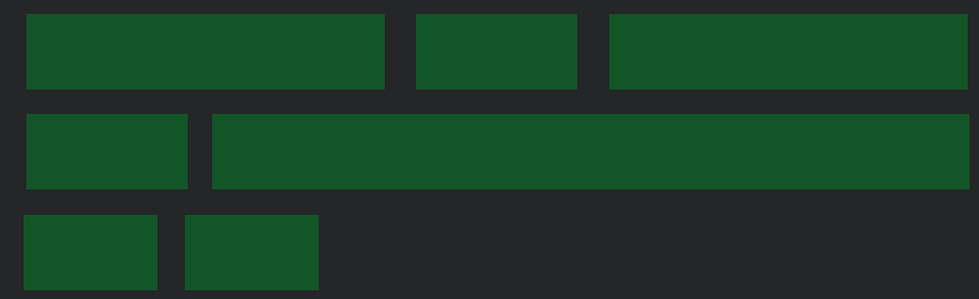
Hiroshi Sugimoto

The dates for the ORIGIN OF ART exhbibition are as follows:

- Science: 21 Nov- 20 Feb 2010/2011
- Architecture: 6 Mar - 15 May 2011

Inline elements

Inline elements appear to continue on the same line.
Examples: <a><b><input><img>



```
Timed to a single revolution of the planet around the sun
at a 23.4 degrees tilt that plays out the rhythm of the
seasons, this <em>Origins of Art</em> cycle is organized
around four themes: <b>science, architecture, history</b>,
and <b>relgion</b>.
```

Timed to a single revolution of the planet around the sun at a 23.4 degrees tilt that plays out the rhythm of the seasons, this *Origins of Art* cycle is organized around four themes: **science, architecture, history**, and **relgion**.

Frontend JavaScript

- Static page
  - Completely described by HTML & CSS
- Dynamic page
  - Adds interactivity, updating HTML based on user interactions
- Adding JS to frontend:

```
<script>
  console.log("Hello, world!");
</script>
```

  - We try to avoid doing this because:
    - Hard to organize
    - Different browsers support different things

DOM: Document Object Model

- API for interacting with HTML browser
- Contains objects corresponding to every HTML element
- Contains global objects for using other browser features

Reference and tutorials

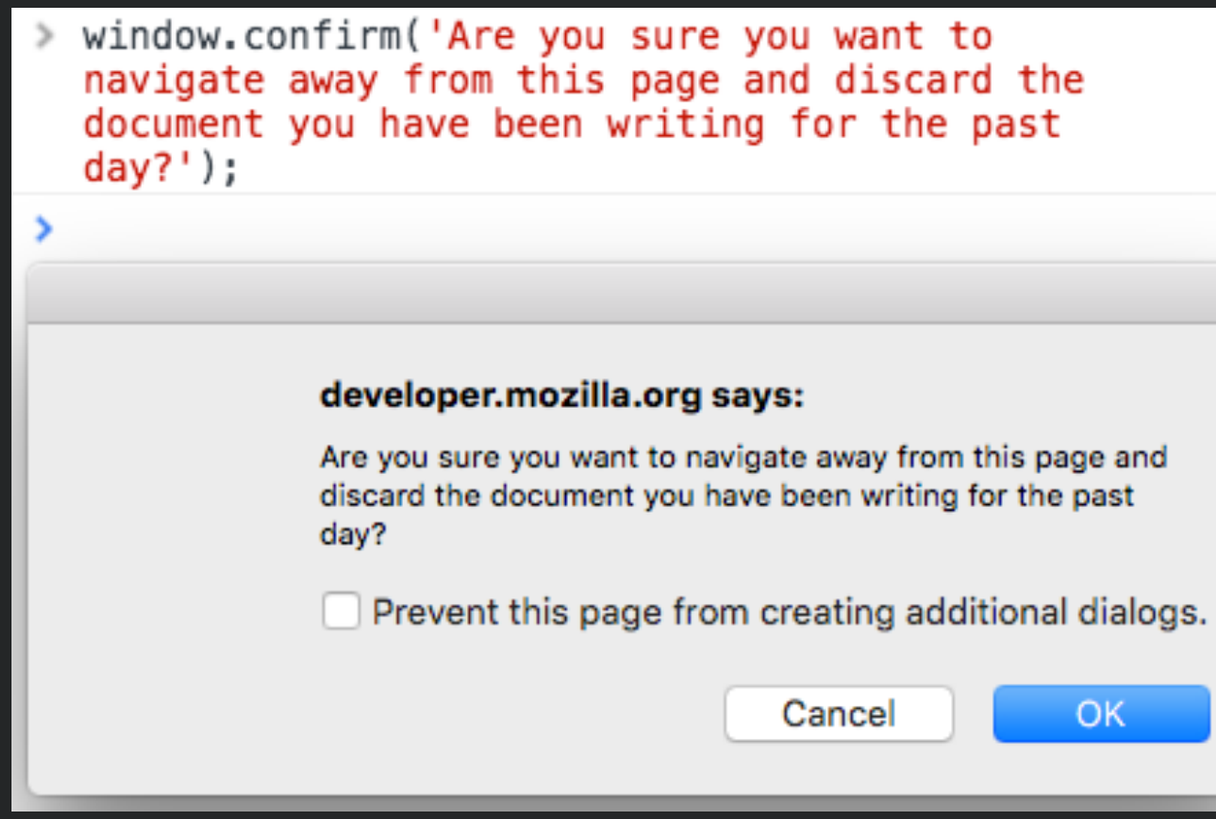
https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model

Global DOM objects

- ***window*** - the browser window
  - Has properties for following objects (e.g., window.document)
  - Or can refer to them directly (e.g., document)
- ***document*** - the current web page
- ***history*** - the list of pages the user has visited previously
- ***location*** - URL of current web page
- ***navigator*** - web browser being used
- ***screen*** - the area occupied by the browser & page

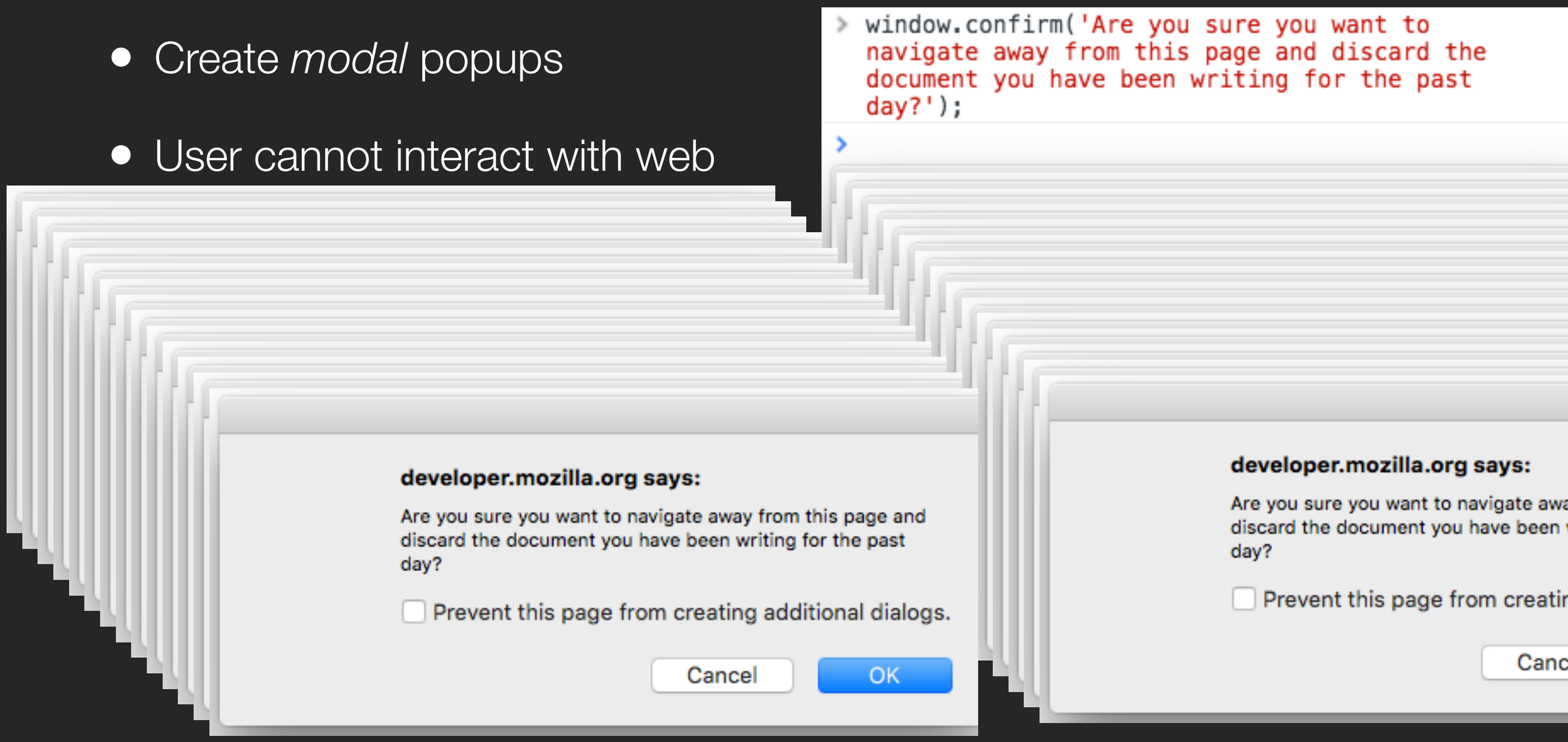
Working with Popups

- alert, confirm, prompt
  - Create *modal* popups
  - User cannot interact with web page until clears the popups
  - Only good style for messages that are *really* important



Working with Popups

- alert, confirm, prompt
  - Create *modal* popups
  - User cannot interact with web



Working with location

- Some properties
  - location.href - full URL of current location
  - location.protocol - protocol being used
  - location.host - hostname
  - location.port
  - location.pathname
- Can navigate to new page by updating the current location
  - location.href = ‘[new URL]’;

```
Location {hash: "", search: "", pathname:
"/~tlatoza/", port: "", hostname:
"cs.gmu.edu"…} i
▶ ancestorOrigins: DOMStringList
▶ assign: function ()
  hash: ""
  host: "cs.gmu.edu"
  hostname: "cs.gmu.edu"
  href: "http://cs.gmu.edu/~tlatoza/"
  origin: "http://cs.gmu.edu"
  pathname: "/~tlatoza/"
  port: ""
  protocol: "http:"
▶ reload: function reload()
```

Traveling Through History

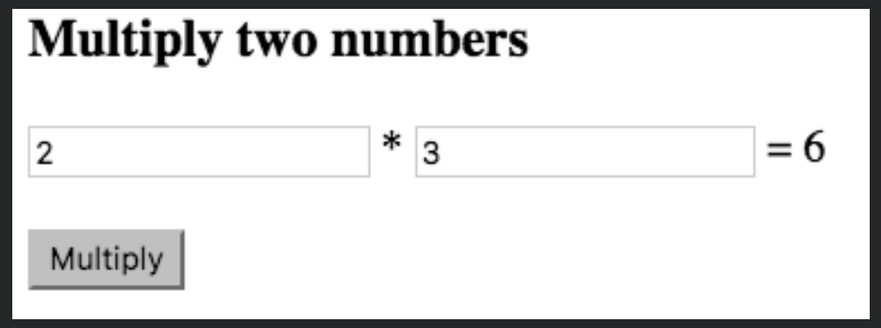
- history.back(), history.forward(), history.go(delta)
- What if you have an SPA & user navigates through different views?
  - Want to be able to jump between different views *within* a single URL
- Solution: manipulate history state
  - Add entries to history stack describing past views
  - Store and retrieve object using history.pushState() and history.state

```
> history.pushState( { activePane: 'main' }, ""
<· undefined
> history.state
<· ▶Object {activePane: "main"}
> history.back();
<· undefined
> history.state
<· null
```

DOM Manipulation

- We can also manipulate the DOM directly
- For this class, we will *not* focus on doing this, but will use React instead
- This is how React works though - it manipulates the DOM

DOM Manipulation




```
<h3>Multiply two numbers</h3>
<div>
    <input id="num1" type="number" /> *
    <input id="num2" type="number" /> =
    <span id="product"></span>
    <br/><br/>
    <button id="compute">Multiply</button>
</div>
```

```
document.getElementById('compute')
        .addEventListener("click", multiply);
function multiply()
{
    var x = document.getElementById('num1').value;
    var y = document.getElementById('num2').value;
    var productElem = document.getElementById('product');
    productElem.innerHTML = x * y;
}
```

May choose any event that the compute element produces. May pass the name of a function or define an anonymous function inline.

DOM Manipulation

Multiply two numbers

2 * 3 = 6

Multiply

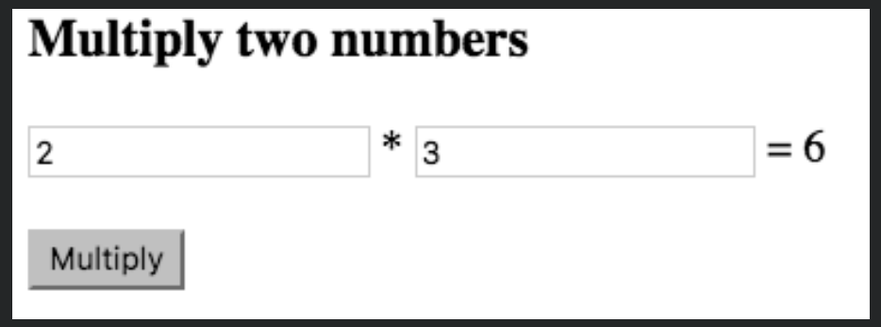
```
<h3>Multiply two numbers</h3>
<div>
    <input id="num1" type="number" /> *
    <input id="num2" type="number" /> =
    <span id="product"></span>
    <br/><br/>
    <button id="compute">Multiply</button>
</div>
```

```
document.getElementById('compute')
        .addEventListener("click", multiply);
function multiply()
{
    var x = document.getElementById('num1').value;
    var y = document.getElementById('num2').value;
    var productElem = document.getElementById('product');
    productElem.innerHTML = x * y;
}
```

"Get compute element"

May choose any event that the compute element produces. May pass the name of a function or define an anonymous function inline.

DOM Manipulation




```
<h3>Multiply two numbers</h3>
<div>
    <input id="num1" type="number" /> *
    <input id="num2" type="number" /> =
    <span id="product"></span>
    <br/><br/>
    <button id="compute">Multiply</button>
</div>
```

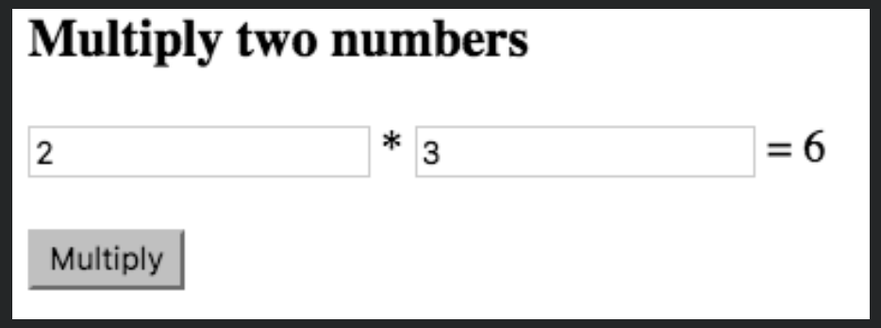
```
document.getElementById('compute')
        .addEventListener("click", multiply);
function multiply()
{
    var x = document.getElementById('num1').value;
    var y = document.getElementById('num2').value;
    var productElem = document.getElementById('product');
    productElem.innerHTML = x * y;
}
```

"Get compute element"

"When compute is clicked, call multiply"

May choose any event that the compute element produces. May pass the name of a function or define an anonymous function inline.

DOM Manipulation




```
<h3>Multiply two numbers</h3>
<div>
    <input id="num1" type="number" /> *
    <input id="num2" type="number" /> =
    <span id="product"></span>
    <br/><br/>
    <button id="compute">Multiply</button>
</div>
```

```
document.getElementById('compute')
        .addEventListener("click", multiply);
function multiply()
{
    var x = document.getElementById('num1').value;
    var y = document.getElementById('num2').value;
    var productElem = document.getElementById('product');
    productElem.innerHTML = x * y;
}
```

Manipulates the DOM by programmatically updating the value of the HTML content. DOM offers accessors for updating all of the DOM state.

DOM Manipulation

Multiply two numbers

2 * 3 = 6

Multiply

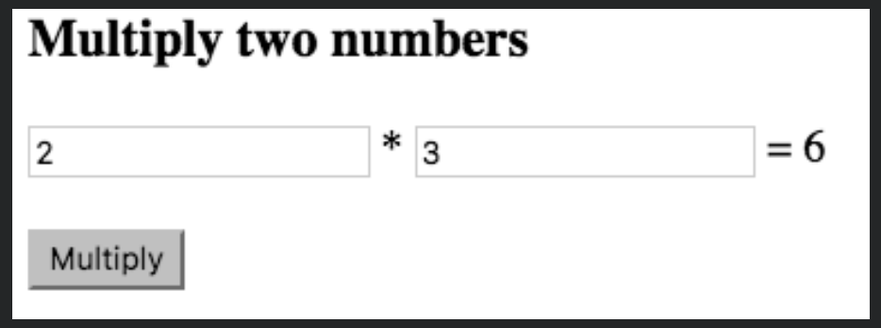
```
<h3>Multiply two numbers</h3>
<div>
    <input id="num1" type="number" /> *
    <input id="num2" type="number" /> =
    <span id="product"></span>
    <br/><br/>
    <button id="compute">Multiply</button>
</div>
```

```
document.getElementById('compute')
        .addEventListener("click", multiply);
function multiply()
{
    var x = document.getElementById('num1').value;
    var y = document.getElementById('num2').value;
    var productElem = document.getElementById('product');
    productElem.innerHTML = x * y;
}
```

"Get the current value of the num1 element"

Manipulates the DOM by programmatically updating the value of the HTML content. DOM offers accessors for updating all of the DOM state.

DOM Manipulation




```html
<h3>Multiply two numbers</h3>
<div>
    <input id="num1" type="number" /> *
    <input id="num2" type="number" /> =
    <span id="product"></span>
    <br/><br/>
    <button id="compute">Multiply</button>
</div>
```

```javascript
document.getElementById('compute')
        .addEventListener("click", multiply);
function multiply()
{
    var x = document.getElementById('num1').value;
    var y = document.getElementById('num2').value;
    var productElem = document.getElementById('product');
    productElem.innerHTML = x * y;
}
```

"Get the current value of the num1 element"

"Set the HTML between the tags of productElem to the value of x * y"

Manipulates the DOM by programmatically updating the value of the HTML content. DOM offers accessors for updating all of the DOM state.

DOM Manipulation Pattern

- Wait for some event
  - click, hover, focus, keypress, …
- Do some computation
  - Read data from event, controls, and/or previous application state
  - Update application state based on what happened
- Update the DOM
  - Generate HTML based on new application state
- Also: JQuery

Problems with Direct DOM Manipulation

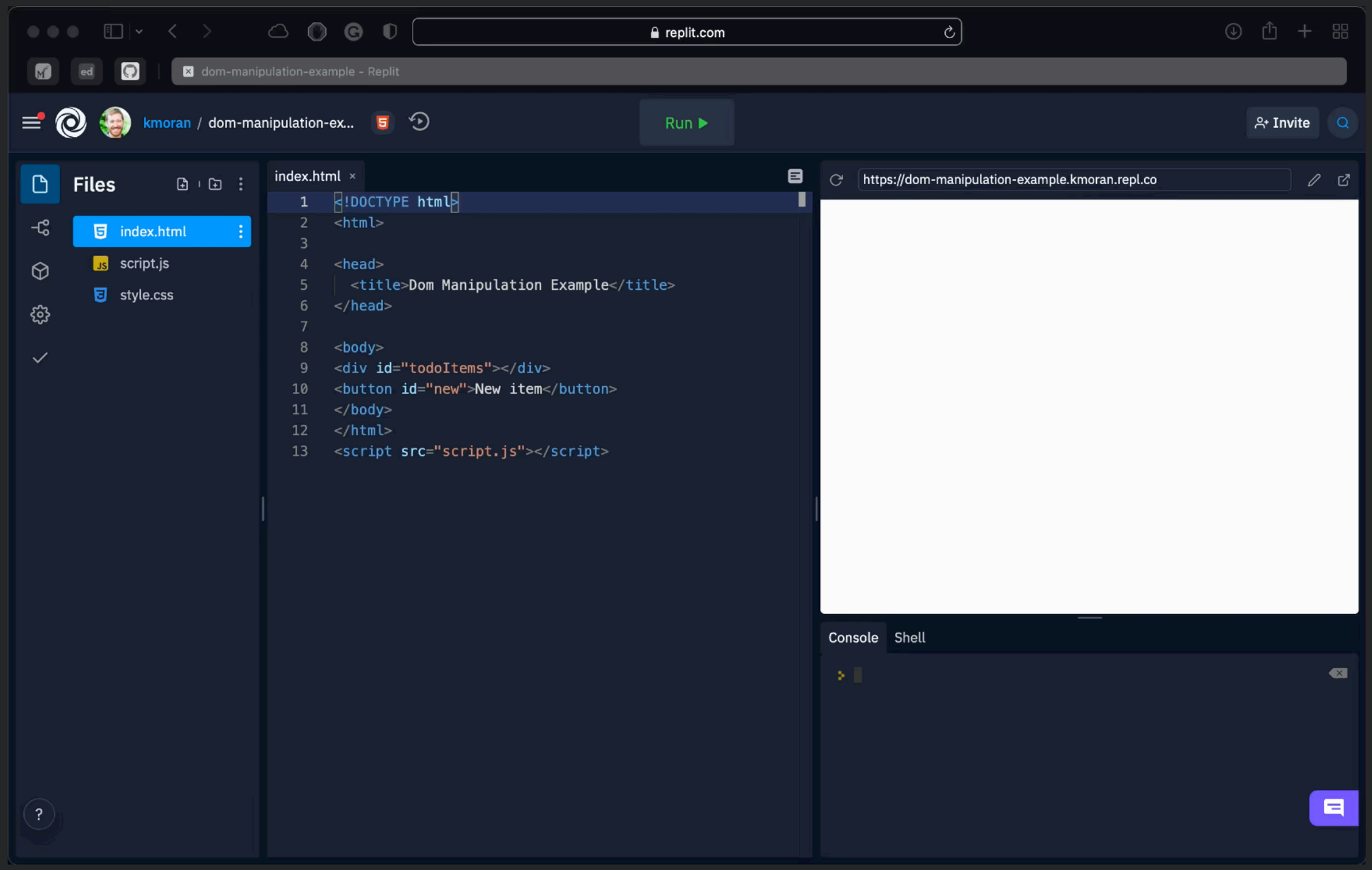
- Managing state becomes difficult for complex applications
- Directly Manipulating the DOM can be ***very*** slow
- ***Reasoning*** about the many ***different states*** in code can become difficult
- Working in a team trying to reason about many different states in code is ***even more difficult***
- Working directly with the DOM is possible, but requires discipline and great documentation.
- Modern web frameworks like Vue.js and React.js make this much easier.

Examples of events

- ***Form element events***
  - change, focus, blur
- ***Network events***
  - online, offline
- ***View events***
  - resize, scroll
- ***Clipboard events***
  - cut, copy, paste
- ***Keyboard events***
  - keydown, keypress, keypup
- ***Mouse events***
  - mouseenter, mouseleave, mousemove, mousedown, mouseup, click, dblclick, select

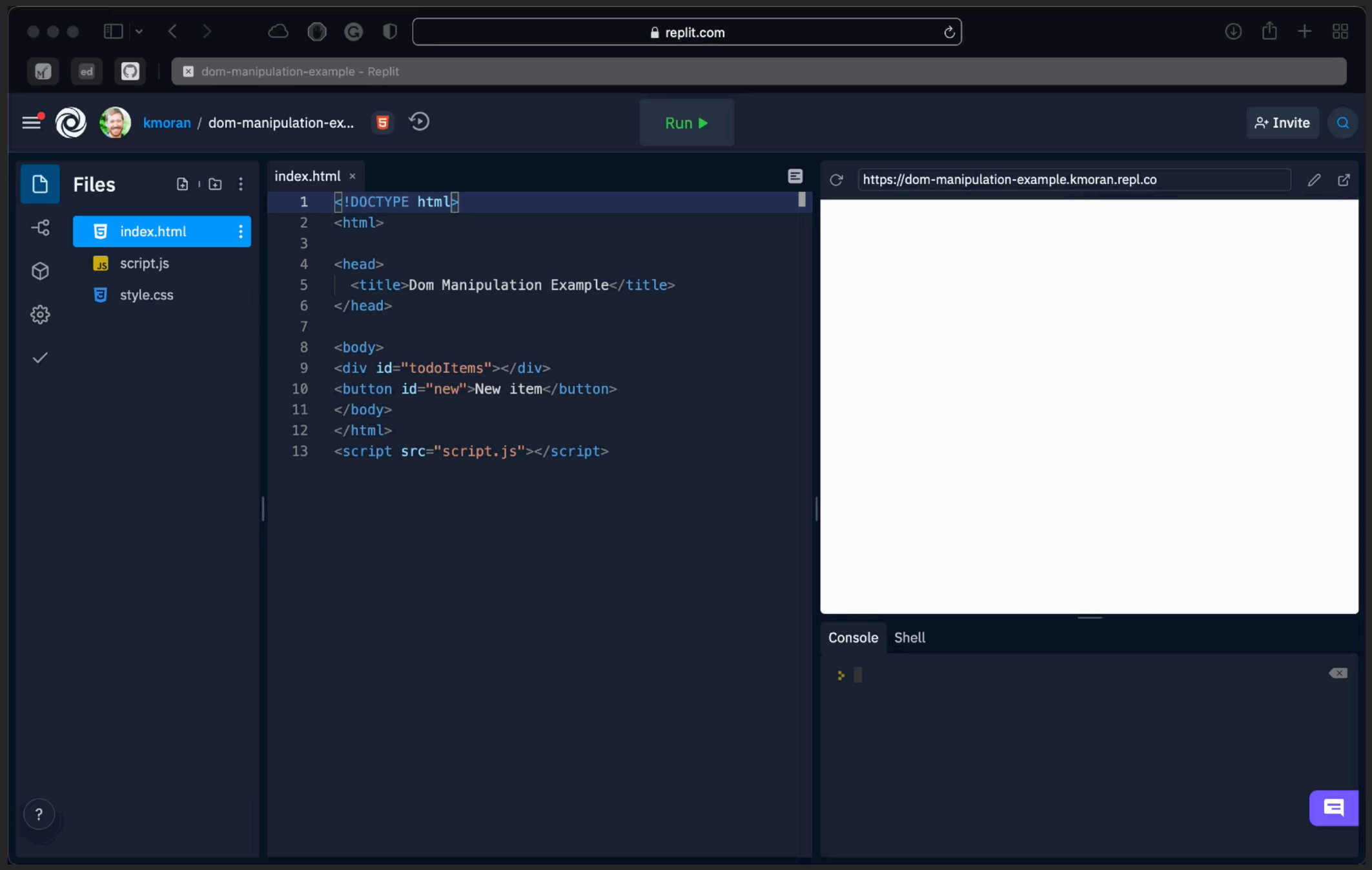
List of events: https://www.w3.org/TR/DOM-Level-3-Events/

DOM Manipulation Example



https://replit.com/@kmoran/dom-manipulation-example#index.html

DOM Manipulation Example



https://replit.com/@kmoran/dom-manipulation-example#index.html

Loading Pages

- What is the output of the following?

```
<script>

document.getElementById('elem').innerHTML =
'New content';
</script>

<div id="elem">Original content</div>
```

Loading Pages

- What is the output of the following?

```
<script>

document.getElementById('elem').innerHTML =
'New content';
</script>

<div id="elem">Original content</div>
```

- ***Answer:*** cannot set property innerHTML of undefined

Loading Pages

- What is the output of the following?

```
<script>

document.getElementById('elem').innerHTML =
'New content';
</script>

<div id="elem">Original content</div>
```

- ***Answer:*** cannot set property innerHTML of undefined

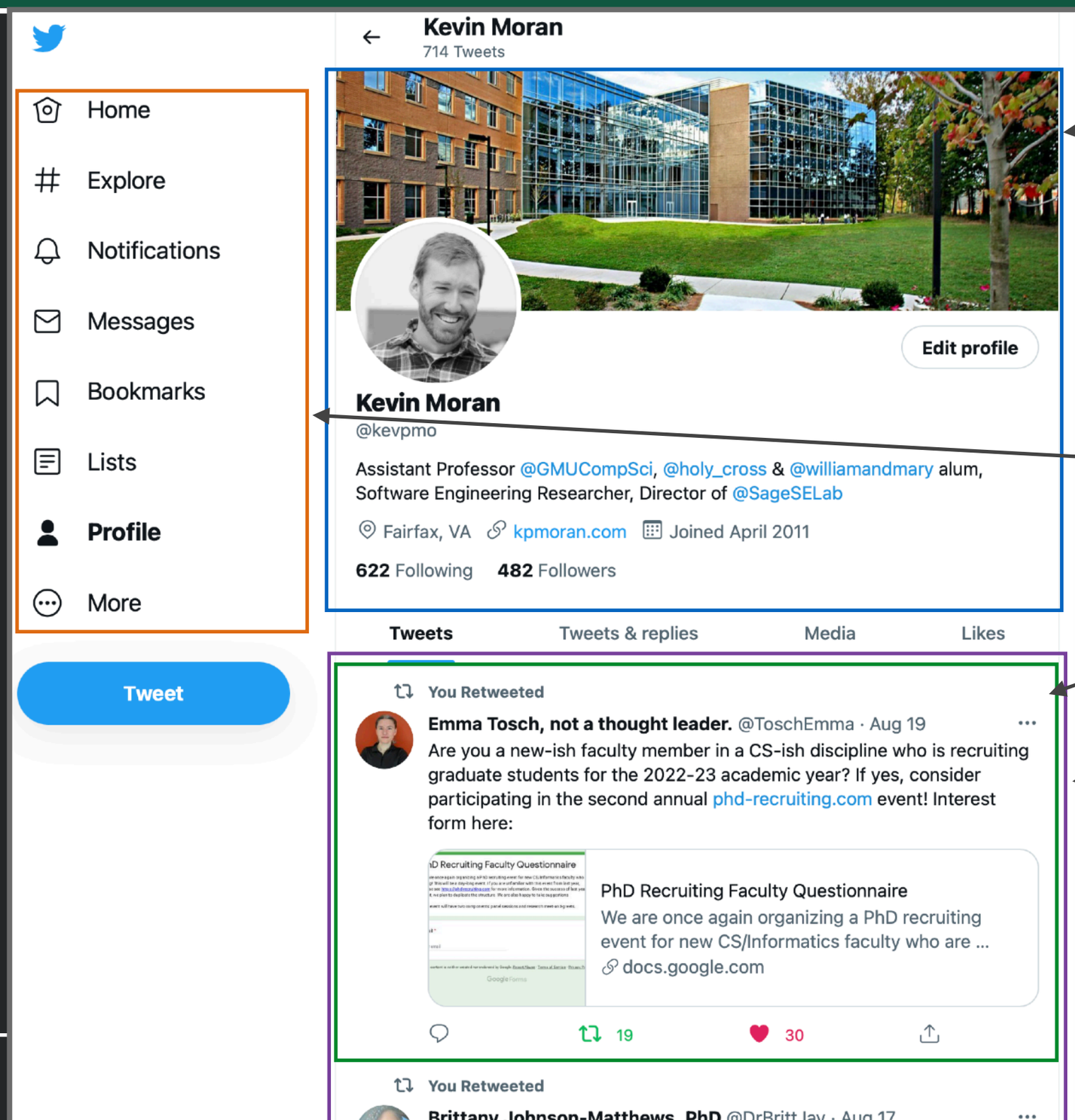
Loading Pages

- What is the output of the following?

```
<script>

document.getElementById('elem').innerHTML =
'New content';
</script>

<div id="elem">Original content</div>
```

- ***Answer:*** cannot set property innerHTML of undefined
- ***Solution:*** Put your script in after the rest of the page is loaded Or, perhaps better solution: don't do DOM manipulation

Anatomy of a Non-Trivial Web App



User profile widget

Menu Bar Widget

Feed widget

Feed item widget

Typical Properties of Web App UIs

Typical Properties of Web App UIs

- Each widget has both visual ***presentation*** & ***logic***
  - e.g., clicking on follow button executes some logic related to the containing widget
  - Logic and presentation of individual widget strongly related, loosely related to other widgets

Typical Properties of Web App UIs

- Each widget has both visual ***presentation*** & ***logic***
  - e.g., clicking on follow button executes some logic related to the containing widget
  - Logic and presentation of individual widget strongly related, loosely related to other widgets
- Some widgets occur ***more than once***
  - e.g., Follow widget occurs multiple times in Who to Follow Widget
  - Need to generate a copy of widget based on data

Typical Properties of Web App UIs

- Each widget has both visual ***presentation*** & ***logic***
  - e.g., clicking on follow button executes some logic related to the containing widget
  - Logic and presentation of individual widget strongly related, loosely related to other widgets
- Some widgets occur ***more than once***
  - e.g., Follow widget occurs multiple times in Who to Follow Widget
  - Need to generate a copy of widget based on data
- Changes to ***data*** should cause changes to ***widget***
  - e.g., following person should update UI to show that the person is followed. Should work even if person becomes followed through other UI

Typical Properties of Web App UIs

- Each widget has both visual ***presentation*** & ***logic***
  - e.g., clicking on follow button executes some logic related to the containing widget
  - Logic and presentation of individual widget strongly related, loosely related to other widgets
- Some widgets occur ***more than once***
  - e.g., Follow widget occurs multiple times in Who to Follow Widget
  - Need to generate a copy of widget based on data
- Changes to ***data*** should cause changes to ***widget***
  - e.g., following person should update UI to show that the person is followed. Should work even if person becomes followed through other UI
- Widgets are ***hierarchical***, with parent and child
  - Seen this already with container elements in HTML…

Idea 1: Templates

```
 document.getElementById('todoItems').innerHTML +=
         '<div class="todoItem" data-index="' + key
         + '"><input type="text" onchange="itemChanged(this)" value="'
+ value + '"><button onclick="deleteItem(this.parentElement)">&#x2716;</button></div>';
```

- Templates describe ***repeated*** HTML through a single ***common*** representation
  - May have ***variables*** that describe variations in the template
  - May have ***logic*** that describes what values are used or when to instantiate template
  - Template may be ***instantiated*** by binding variables to values, creating HTML that can be used to update DOM

Templates with Template Literals

```
document.getElementById('todoItems').innerHTML +=
        `<div class="todoItem" data-index="${key}">
             <input type="text" onchange="itemChanged(this)" value="${value}">
             <button onclick="deleteItem(this.parentElement)">&#x2716;</button>
         </div>`;
```

- Template literals reduce confusion of nested strings

Server Side vs. Client Side

- Where should template be instantiated?
- *Server-side* frameworks: Template instantiated on ***server***
  - Examples: JSP, ColdFusion, PHP, ASP.NET
  - Logic executes on server, generating HTML that is served to browser
- *Front-end* framework: Template runs in web ***browser***
  - Examples: React, Angular, Meteor, Ember, Aurelia, …
  - Server passes template to browser, browser generates HTML on demand

```
<html>
<head><title>First JSP</title></head>
<body>
  <%
    double num = Math.random();
    if (num > 0.95) {
  %>
      <h2>You'll have a luck day!</h2><p>(<%= num %>)</p>
  <%
    } else {
  %>
      <h2>Well, life goes on ... </h2><p>(<%= num %>)</p>
  <%
    }
  %>
```

```
class HelloMessage extends React.Component {
  render() {
    return (
      <div>
        Hello {this.props.name}
      </div>
    );
  }
}

ReactDOM.render(
  <HelloMessage name="John" />,
  mountNode
);
```

Server Side vs. Client Side

Server Side vs. Client Side

- Server side
  - Oldest solution.
  - True when “real” code ran on server, Javascript

Server Side vs. Client Side

- Server side
  - Oldest solution.
  - True when “real” code ran on server, Javascript
- Client side
  - Enables presentation logic to exist entirely in browser
  - e.g., can make call to remote web service, no need for server to be involved
  - (What we are looking at in this course).

Logic

- Templates require combining logic with HTML
  - Conditionals - only display presentation if some expression is true
  - Loops - repeat this template once for every item in collection
- How should this be expressed?
  - Embed code in HTML (ColdFusion, JSP, Angular)
  - Embed HTML in code (React)

Embed Code in HTML

```
<cfcomponent name = "PersonalChef">
    <cffunction name = "makeToast" returnType = "component">
        <cfargument name = "color" required="yes">

        <cfset this.makeToast = "Making your toast #arguments.color#!" />
        <cfreturn this />
    </cffunction>
</cfcomponent>
```

```
<html>
<head><title>First JSP</title></head>
<body>
  <%
    double num = Math.random();
    if (num > 0.95) {
  %>
      <h2>You'll have a luck day!</h2><p>(<%= num %>)</p>
  <%
    } else {
  %>
      <h2>Well, life goes on ... </h2><p>(<%= num %>)</p>
  <%
    }
  %>
```

- Template takes the form of an HTML file, with extensions
  - Custom tags (e.g., <% %>) enable logic to be embedded in HTML
  - Uses another language (e.g., Java, C) or custom language to express logic
  - Found in frameworks such as PHP, Angular, ColdFusion, ASP, ...

Embed HTML in Code

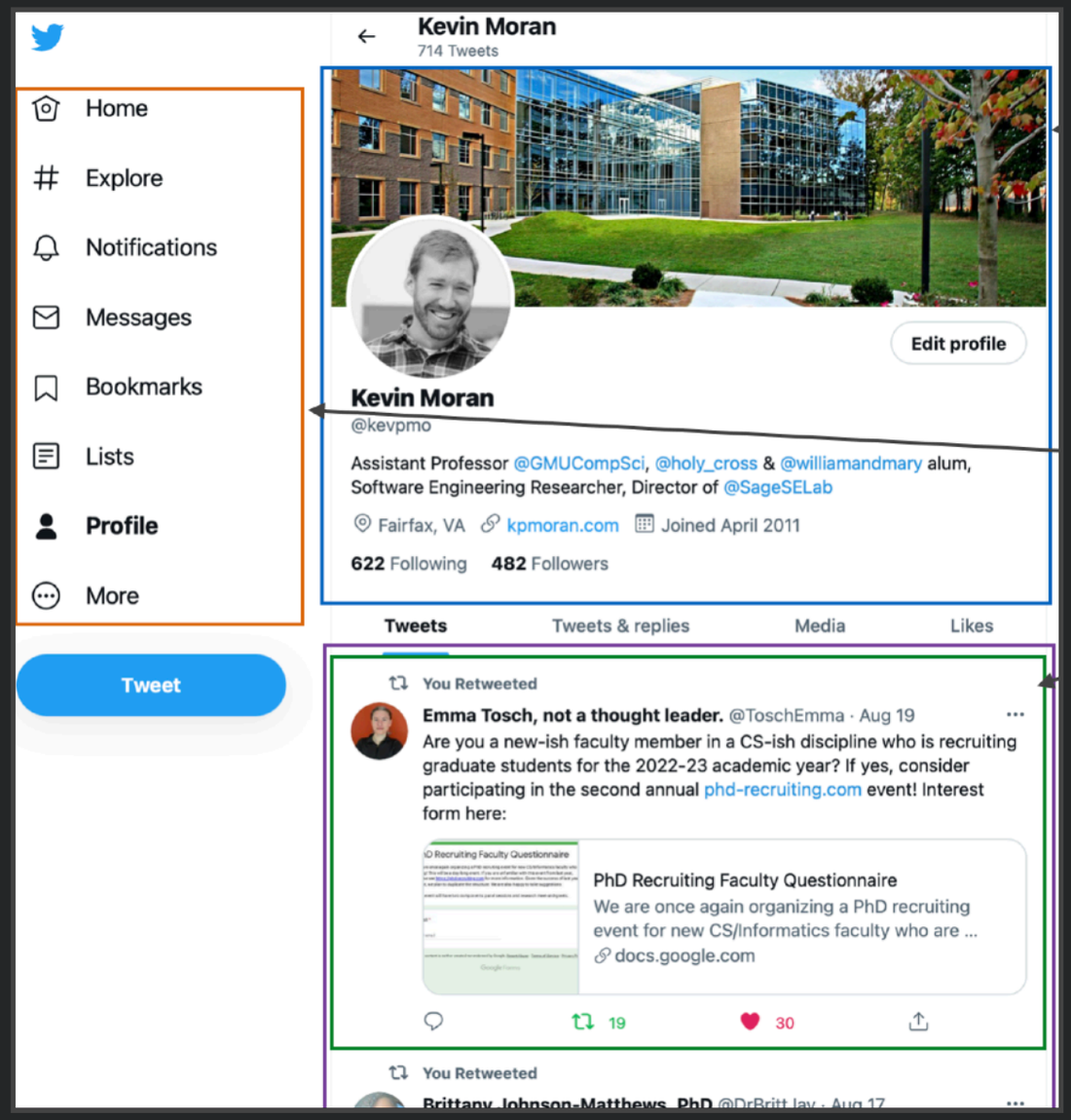
- Template takes the form of an HTML fragment, embedded in a code file
  - HTML instantiated as part of an expression, becomes a value that can be stored to variables
  - Uses another language (e.g., Javascript) to express logic
  - This course: ***React***

Templates Enable HTML to be Rendered Multiple Times

- Rendering takes a template, instantiates the template, outputs HTML
- Logic determines which part(s) of templates are rendered
- Expressions are evaluated to instantiate values
  - e.g., { this.props.name }
  - Different variable values ==> different HTML output

Idea 2: Components

- Web pages are complex, with lots of logic and presentation
- How can we organize web page to maximize modularity?
- Solution: Components
  - Templates that correspond to a specific widget
  - Encapsulates related logic & presentation using language construct (e.g., class)



Components

- Organize related logic and presentation into a single unit
  - Includes necessary *state* and the logic for updating this state
  - Includes presentation for *rendering* this state into HTML
  - Outside world *must* interact with state through accessors, enabling access to be controlled
- Synchronizes state and visual presentation
  - Whenever state changes, HTML should be rendered again
- Components instantiated through custom HTML tag

React: Front End Framework for Components

React

A JavaScript library for building user interfaces

- Originally built by Facebook
- Open-source frontend framework
- Powerful abstractions for describing frontend UI components
- Official documentation & tutorials
  - https://reactjs.org/

Example

```
class HelloMessage extends React.Component {
  render() {
    return (
      <div>
        Hello world!
      </div>
    );
  }
}

ReactDOM.render(
  <HelloMessage/>, mountNode
);
```

"Declare a HelloMessage component"

Declares a new component with the provided functions.

"Return the following HTML whenever the component is rendered"

Render generates the HTML for the component. The HTML is dynamically generated by the library.

"Render HelloMessage and insert in mountNode"

Instantiates component, replaces mountNode innerHTML with rendered HTML. Second parameter should always be a DOM element.

Example - Properties

```
class HelloMessage extends React.Component {
  render() {
    return (
      <div>
        Hello {this.props.name}
      </div>
    );
  }
}

ReactDOM.render(
  <HelloMessage name="John" />,
  mountNode
);
```

"Read this.props.name and output the value"

Evaluates the expression to a value.

"Set the name property of HelloMessage to John"

Components have a this.props collection that contains a set of properties instantiated for each component.

Embedding HTML in Javascript

```
return <div>Hello {this.props.name}</div>;
```

- HTML embedded in JavaScript
  - HTML can be used as an expression
  - HTML is checked for correct syntax
- Can use { expr } to evaluate an expression and return a value
  - e.g., { 5 + 2 }, { foo() }
- Output of expression is HTML

JSX

- How do you embed HTML in JavaScript and get syntax checking??
- Idea: extend the language: JSX
  - Javascript language, with additional feature that expressions may be HTML
  - Can be used with ES6 or traditional JS (ES5)
- It's a new(ish) language
  - Browsers *do not* natively run JSX
  - If you include a JSX file as source, you will get an error

reactjs.org

React – A JavaScript library for building user interfaces

React Docs Tutorial Blog Community Search v17.0.2 Languages GitHub

React

A JavaScript library for building user interfaces

Get Started Take the Tutorial

Declarative

React makes it painless to create interactive UIs. Design simple views for each state in your application, and React will efficiently update and render just the right components when your data changes.

Component-Based

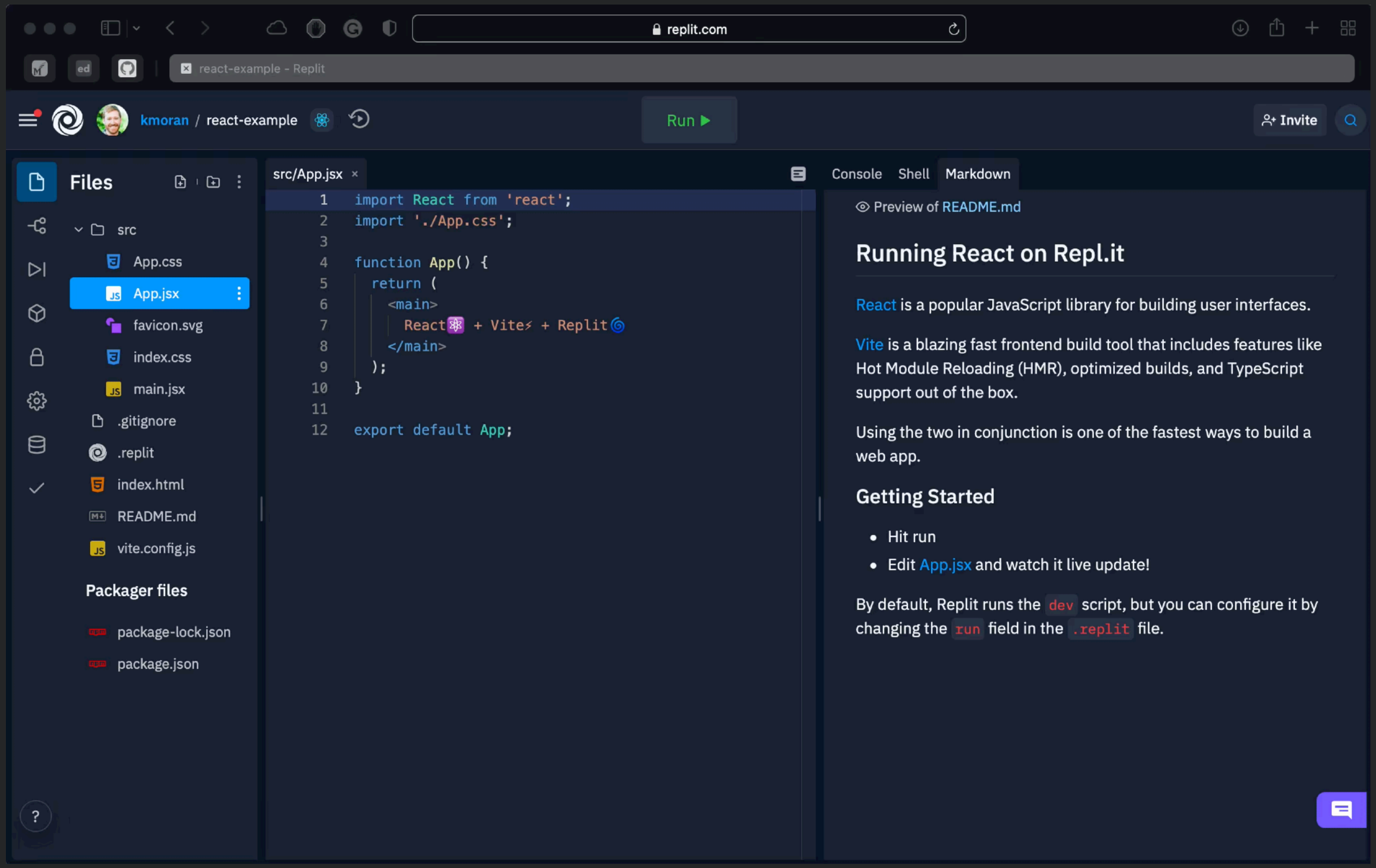
Build encapsulated components that manage their own state, then compose them to make complex UIs.

Since component logic is written in JavaScript

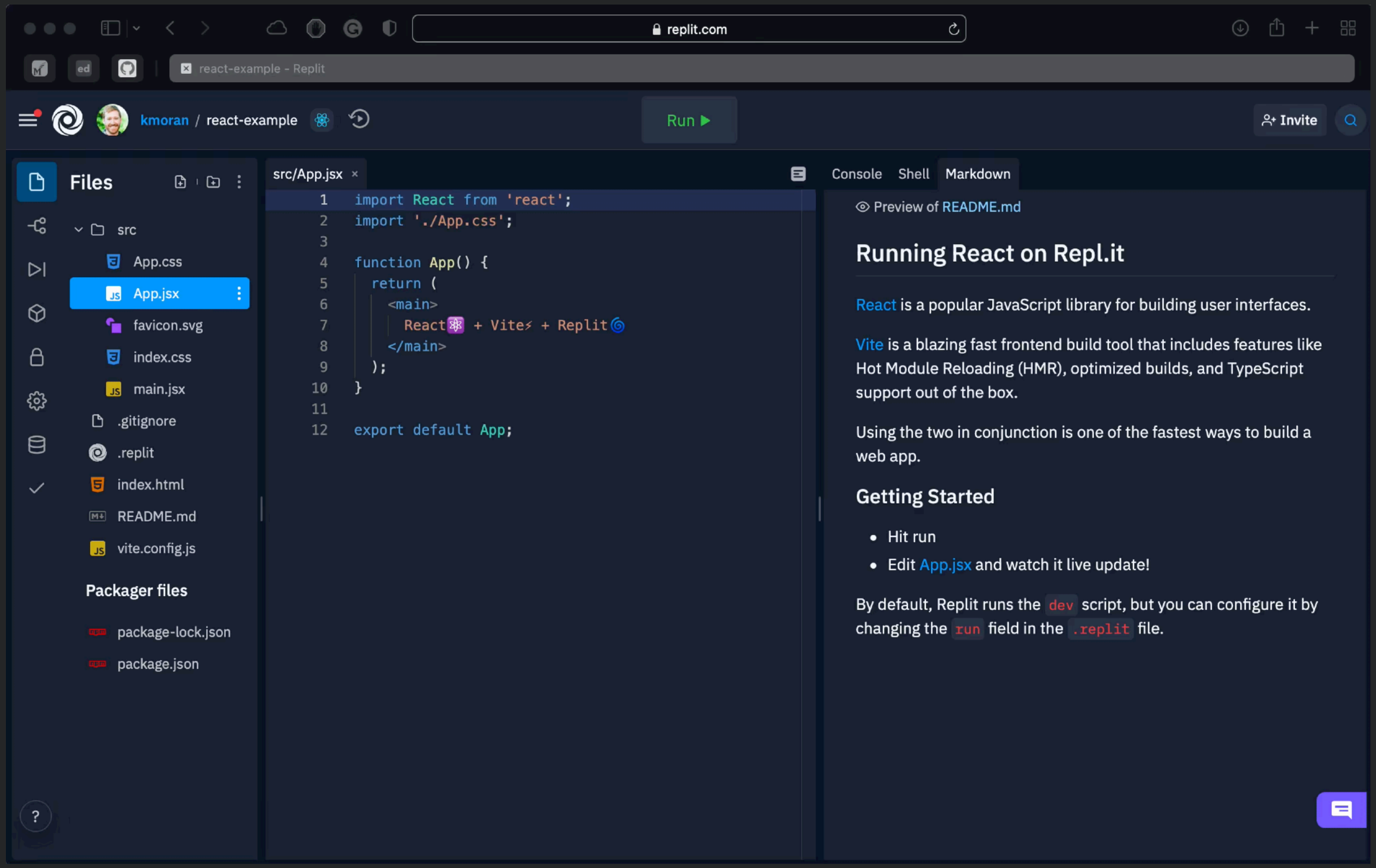
Learn Once, Write Anywhere

We don't make assumptions about the rest of your technology stack, so you can develop new features in React without rewriting existing code.

React can also render on the server using Node



- Pastebin sites such as Replit work with React
- Just need to include React first



- Pastebin sites such as Replit work with React
- Just need to include React first

Create React App

```
λ npx create-react-app my-app
npx: installed 114 in 4.308s
```

https://github.com/facebook/create-react-app

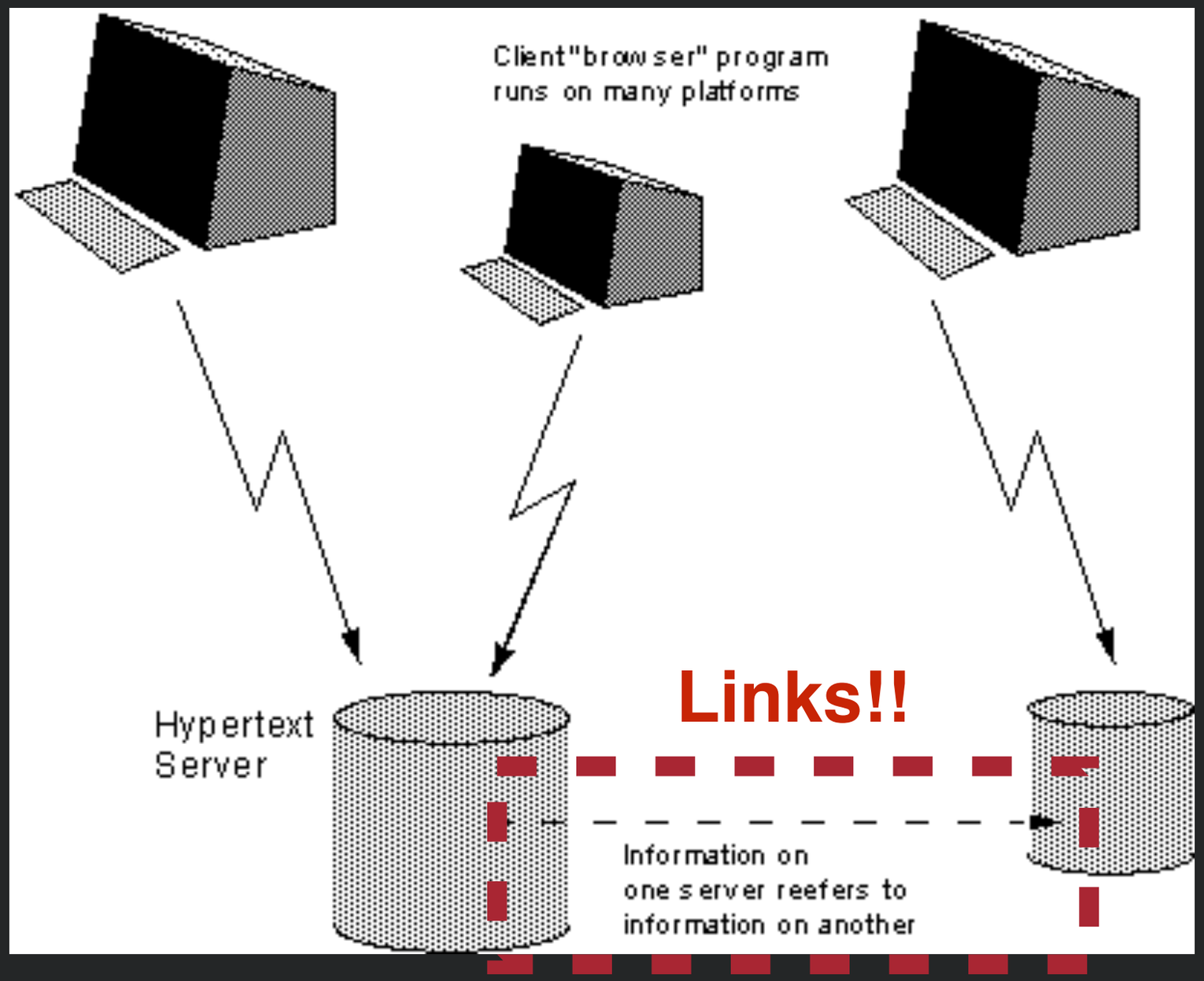
Create React App

```
λ npx create-react-app my-app
npx: installed 114 in 4.308s
```

https://github.com/facebook/create-react-app

Midterm Exam Review

Original WWW Architecture



URI: Universal Resource Identifier

URI: <scheme>://<authority><path>?<query>

http://cs.gmu.edu/~kpmoran/swe-432-f21.html

URI: Universal Resource Identifier

URI: <scheme>://<authority><path>?<query>

http://cs.gmu.edu/~kpmoran/swe-432-f21.html

"Use HTTP scheme"

Other popular schemes:
ftp, mailto, file

URI: Universal Resource Identifier

URI: <scheme>://<authority><path>?<query>

http://cs.gmu.edu/~kpmoran/swe-432-f21.html

"Use HTTP scheme"

Other popular schemes:
ftp, mailto, file

"Connect to cs.gmu.edu"

May be host name or an IP address
Optional port name (e.g., :80 for port 80)

URI: Universal Resource Identifier

URI: <scheme>://<authority><path>?<query>

http://cs.gmu.edu/~kpmoran/swe-432-f21.html

"Use HTTP scheme"

Other popular schemes:
ftp, mailto, file

"Connect to cs.gmu.edu"

May be host name or an IP address
Optional port name (e.g., :80 for port 80)

"Request ~kpmoran/swe-432-f21.html"

URI: Universal Resource Identifier

URI: <scheme>://<authority><path>?<query>

http://cs.gmu.edu/~kpmoran/swe-432-f21.html

"Use HTTP scheme"

Other popular schemes:
ftp, mailto, file

"Connect to cs.gmu.edu"

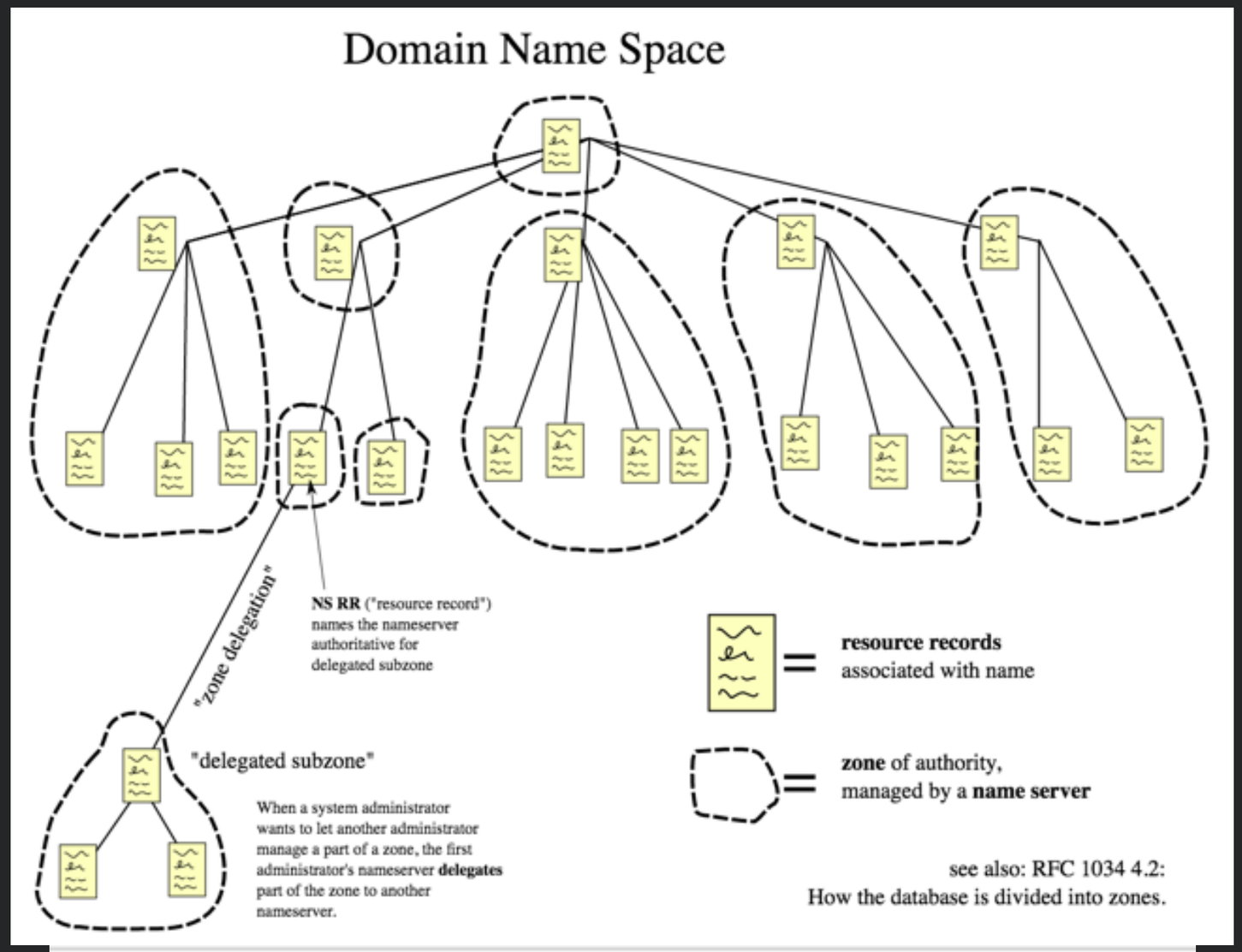
May be host name or an IP address
Optional port name (e.g., :80 for port 80)

"Request ~kpmoran/swe-432-f21.html"

More details: https://en.wikipedia.org/wiki/Uniform_Resource_Identifier

DNS: Domain Name System

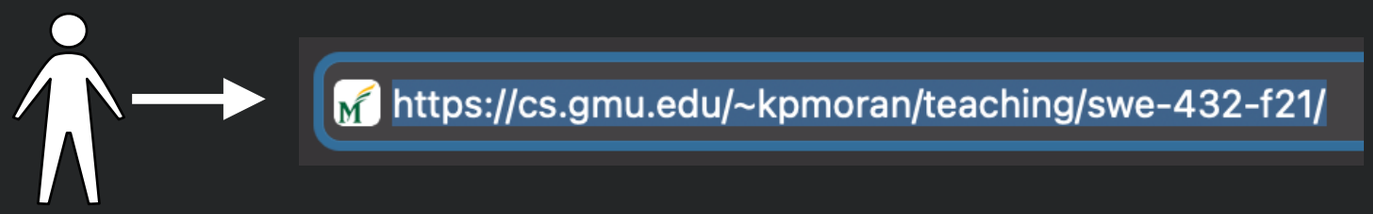
- Domain name system (DNS) (~1982)
  - Mapping from names to IP addresses
- E.g. cs.gmu.edu -> 129.174.125.139



The hierarchical Domain Name System for class *Internet*, organized into zones, each served by a name server

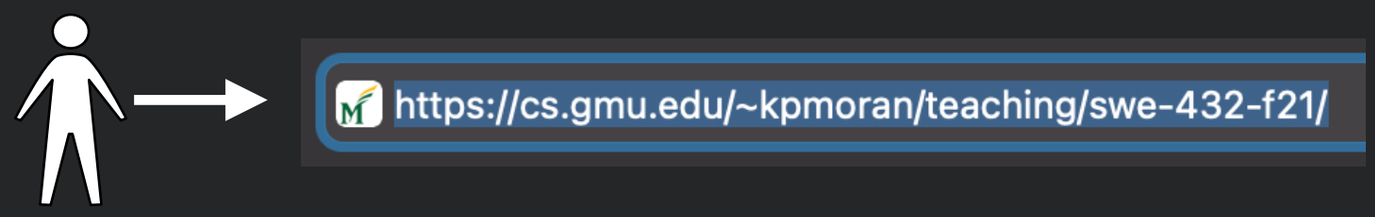
HTTP: HyperText Transfer Protocol

High-level protocol built on TCP/IP that defines how data is transferred on the web



HTTP: HyperText Transfer Protocol

High-level protocol built on TCP/IP that defines how data is transferred on the web

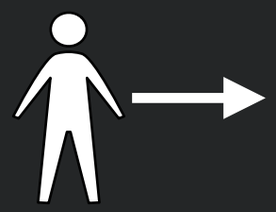


HTTP Request

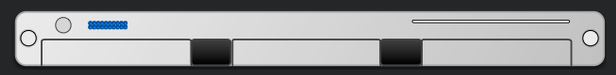
```
GET /~kpmoran/swe-432-f21.html HTTP/1.1
Host: cs.gmu.edu
Accept: text/html
```

HTTP: HyperText Transfer Protocol

High-level protocol built on TCP/IP that defines how data is transferred on the web



https://cs.gmu.edu/~kpmoran/teaching/swe-432-f21/



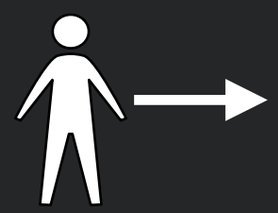
web server

HTTP Request

```
GET /~kpmoran/swe-432-f21.html HTTP/1.1
Host: cs.gmu.edu
Accept: text/html
```

HTTP: HyperText Transfer Protocol

High-level protocol built on TCP/IP that defines how data is transferred on the web



https://cs.gmu.edu/~kpmoran/teaching/swe-432-f21/

HTTP Request

```
GET /~kpmoran/swe-432-f21.html HTTP/1.1
Host: cs.gmu.edu
Accept: text/html
```

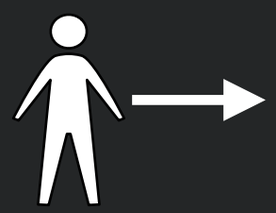
Reads file from disk

HTTP Response

```
HTTP/1.1 200 OK
Content-Type: text/html; charset=UTF-8

<html><head>...
```

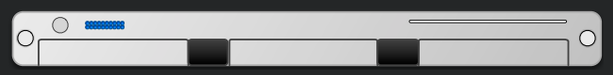
HTTP: HyperText Transfer Protocol

High-level protocol built on TCP/IP that defines how data is transferred on the web



https://cs.gmu.edu/~kpmoran/teaching/swe-432-f21/

web server



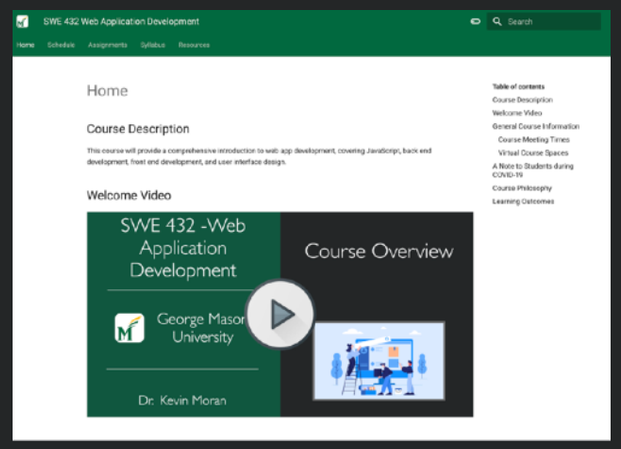
HTTP Request

```
GET /~kpmoran/swe-432-f21.html HTTP/1.1
Host: cs.gmu.edu
Accept: text/html
```

Reads file from disk

HTTP Response

```
HTTP/1.1 200 OK
Content-Type: text/html; charset=UTF-8

<html><head>...
```



HTTP Requests

HTTP Request

```
GET /~kpmoran/swe-432-f21.html HTTP/1.1
Host: cs.gmu.edu
Accept: text/html
```

Other popular types:
POST, PUT, DELETE, HEAD

- Request may contain additional *header lines* specifying, e.g. client info, parameters for forms, cookies, etc.
- Ends with a carriage return, line feed (blank line)
- May also contain a message body, delineated by a blank line

HTTP Requests

HTTP Request

```
GET /~kpmoran/swe-432-f21.html HTTP/1.1
Host: cs.gmu.edu
Accept: text/html
```

"GET request"

Other popular types:
POST, PUT, DELETE, HEAD

- Request may contain additional *header lines* specifying, e.g. client info, parameters for forms, cookies, etc.
- Ends with a carriage return, line feed (blank line)
- May also contain a message body, delineated by a blank line

HTTP Requests

HTTP Request

```
GET /~kpmoran/swe-432-f21.html HTTP/1.1
Host: cs.gmu.edu
Accept: text/html
```

"GET request"

"Resource"

Other popular types:
POST, PUT, DELETE, HEAD

- Request may contain additional *header lines* specifying, e.g. client info, parameters for forms, cookies, etc.
- Ends with a carriage return, line feed (blank line)
- May also contain a message body, delineated by a blank line

HTTP Responses

```
HTTP/1.1 200 OK
Date: Mon, 23 May 2005 22:38:34 GMT
Content-Type: text/html; charset=UTF-8
Content-Encoding: UTF-8
Content-Length: 138
Last-Modified: Wed, 08 Jan 2003 23:11:55 GMT
Server: Apache/1.3.3.7 (Unix) (Red-Hat/Linux)
ETag: "3f80f-1b6-3e1cb03b"
Accept-Ranges: bytes
Connection: close

<html>
<head>
  <title>An Example Page</title>
</head>
<body>
  Hello World, this is a very simple HTML document.
</body>
</html>
```

Response status codes:
1xx Informational
2xx Success
3xx Redirection
4xx Client error
5xx Server error

Common MIME types:
application/json
application/pdf
image/png

HTTP Responses

```
HTTP/1.1 200 OK
Date: Mon, 23 May 2005 22:38:34 GMT
Content-Type: text/html; charset=UTF-8
Content-Encoding: UTF-8
Content-Length: 138
Last-Modified: Wed, 08 Jan 2003 23:11:55 GMT
Server: Apache/1.3.3.7 (Unix) (Red-Hat/Linux)
ETag: "3f80f-1b6-3e1cb03b"
Accept-Ranges: bytes
Connection: close

<html>
<head>
  <title>An Example Page</title>
</head>
<body>
  Hello World, this is a very simple HTML document.
</body>
</html>
```

[HTML data]

"OK response"

Response status codes:
1xx Informational
2xx Success
3xx Redirection
4xx Client error
5xx Server error

"HTML returned content"

Common MIME types:
application/json
application/pdf
image/png

Properties of HTTP

- Request-response
  - Interactions always initiated by client request to server
  - Server responds with results
- Stateless
  - Each request-response pair independent from every other
  - Any state information (login credentials, shopping carts, etc.) needs to be encoded somehow

HTML: HyperText Markup Language

HTML is a **markup language** - it is a language for describing parts of a document

- NOT a programming language

HTML: HyperText Markup Language

HTML is a **markup language** - it is a language for describing parts of a document

- NOT a programming language
- Tags are added to markup the text, encompassed with <>'s

HTML: HyperText Markup Language

HTML is a **markup language** - it is a language for describing parts of a document

- NOT a programming language
- Tags are added to markup the text, encompassed with <>'s
- Simple markup tags: <b>,<i>, <u> (bold, italic, underline)

HTML: HyperText Markup Language

HTML is a **markup language** - it is a language for describing parts of a document

- NOT a programming language
- Tags are added to markup the text, encompassed with <>'s
- Simple markup tags: <b>,<i>, <u> (bold, italic, underline)

```
<b>This text is bold!</b>
```

HTML: HyperText Markup Language

HTML is a **markup language** - it is a language for describing parts of a document

- NOT a programming language
- Tags are added to markup the text, encompassed with <>'s
- Simple markup tags: <b>,<i>, <u> (bold, italic, underline)

```
<b>This text is bold!</b>
```

↓

This text is bold!

Variables

- Variables are *loosely* typed
  - String:
    ```
    var strVar = 'Hello';
    ```
  - Number:
    ```
    var num = 10;
    ```
  - Boolean:
    ```
    var bool = true;
    ```
  - Undefined:
    ```
    var undefined;
    ```
  - Null:
    ```
    var nulled = null;
    ```
  - Objects (includes arrays):
    ```
    var intArray = [1,2,3];
    ```
  - Symbols (named magic strings):
    ```
    var sym = Symbol(‘Description of the symbol’);
    ```
  - Functions (We’ll get back to this)
- Names start with letters, $ or _
- Case sensitive

Const

- Can define a variable that cannot be assigned again using const

```
const numConst = 10; //numConst can't be
changed
```

- For objects, properties may change, but object identity may not.

More Variables

- Loose typing means that JS figures out the type based on the value

```
 let x; //Type: Undefined
x = 2; //Type: Number
x = 'Hi'; //Type: String
```

- Variables defined with let (but not var) have block scope
  - If defined in a function, can only be seen in that function
  - If defined outside of a function, then global. Can also make arbitrary blocks:

```
{
    let a = 3;
}
//a is undefined
```

Loops and Control Structures

- `if` - pretty standard

```
if (myVar >= 35) {
    //...
} else if(myVar >= 25){
    //...
} else {
    //...
}
```

- Also get `while`, `for`, and `break` as you might expect

```
while(myVar > 30){
    //...
}

for(var i = 0; i < myVar; i++){
     //...
     if(someOtherVar == 0)
      break;
}
```

Operators

Operators

Operators

Operator	Meaning	Examples

Operators

```
var age = 20;
```

Operator	Meaning	Examples
==	Equality	`age == 20` `age == '20'`

Operators

```
var age = 20;
```

Operator	Meaning	Examples
==	Equality	`age == 20` `age == '20'`

Annoying

Operators

```
var age = 20;
```

Operator	Meaning	Examples
==	Equality	`age == 20` `age == '20'`
!=	Inequality	`age != 21`

Annoying

Operators

```
var age = 20;
```

Operator	Meaning	Examples
==	Equality	`age == 20` `age == '20'`
!=	Inequality	`age != 21`
>	Greater than	`age > 19`

Annoying

Operators

```
var age = 20;
```

Operator	Meaning	Examples
==	Equality	`age == 20` `age == '20'`
!=	Inequality	`age != 21`
>	Greater than	`age > 19`
>=	Greater or Equal	`age >= 20`

Annoying

Operators

```
var age = 20;
```

Operator	Meaning	Examples
==	Equality	`age == 20` `age == '20'`
!=	Inequality	`age != 21`
>	Greater than	`age > 19`
>=	Greater or Equal	`age >= 20`
<	Less than	`age < 21`

Annoying

Operators

```
var age = 20;
```

Operator	Meaning	Examples
==	Equality	`age == 20` `age == '20'`
!=	Inequality	`age != 21`
>	Greater than	`age > 19`
>=	Greater or Equal	`age >= 20`
<	Less than	`age < 21`
<=	Less or equal	`age <= 20`

Annoying

Operators

```
var age = 20;
```

Operator	Meaning	Examples
==	Equality	`age == 20` `age == '20'`
!=	Inequality	`age != 21`
>	Greater than	`age > 19`
>=	Greater or Equal	`age >= 20`
<	Less than	`age < 21`
<=	Less or equal	`age <= 20`
===	Strict equal	`age === 20`

Annoying

Operators

```
var age = 20;
```

Operator	Meaning	Examples
==	Equality	`age == 20` `age == '20'`
!=	Inequality	`age != 21`
>	Greater than	`age > 19`
>=	Greater or Equal	`age >= 20`
<	Less than	`age < 21`
<=	Less or equal	`age <= 20`
===	Strict equal	`age === 20`
!==	Strict Inequality	`age !== '20'`

Annoying

Functions

- At a high level, syntax should be familiar:

```
function add(num1, num2) {
    return num1 + num2;
}
```

- Calling syntax should be familiar too:

```
var num = add(4,6);
```

- Can also assign functions to variables!

```
var magic = function(num1, num2){
    return num1+num2;
}
var myNum = magic(4,6);
```

- Why might you want to do this?

Default Values

```
function add(num1=10, num2=45) {
    return num1 + num2;
}
```

Default Values

```
function add(num1=10, num2=45) {
    return num1 + num2;
}
```

Default Values

```
function add(num1=10, num2=45) {
    return num1 + num2;
}

var r = add(); // 55
```

Default Values

```
function add(num1=10, num2=45) {
    return num1 + num2;
}

var r = add(); // 55
var r = add(40); //85
```

Default Values

```
function add(num1=10, num2=45) {
    return num1 + num2;
}

var r = add(); // 55
var r = add(40); //85
var r = add(2,4); //6
```

Rest Parameters

```
function add(num1, ... morenums) {
    var ret = num1;
    for(var i = 0; i < morenums.length; i++)
        ret += morenums[i];
    return ret;
}
```

Rest Parameters

```
function add(num1, ... morenums) {
    var ret = num1;
    for(var i = 0; i < morenums.length; i++)
        ret += morenums[i];
    return ret;
}
```

Rest Parameters

```
function add(num1, ... morenums) {
    var ret = num1;
    for(var i = 0; i < morenums.length; i++)
        ret += morenums[i];
    return ret;
}


add(40,10,20); //70
```

=> Arrow Functions

- Simple syntax to define short functions *inline*
- Several ways to use

```
var add = (a,b) => {
    return a+b;
}
```

=> Arrow Functions

- Simple syntax to define short functions *inline*
- Several ways to use

Parameters

```
var add = (a,b) => {
    return a+b;
}
```

=> Arrow Functions

- Simple syntax to define short functions *inline*
- Several ways to use

Parameters

```
var add = (a,b) => {
    return a+b;
}


var add = (a,b) => a+b;
```

If your arrow function only has one expression, JavaScript will automatically add the word “return”

Objects

Objects

- What are objects like in other languages? How are they written and organized?

Objects

- What are objects like in other languages? How are they written and organized?
- Traditionally in JS, no *classes*

Objects

- What are objects like in other languages? How are they written and organized?
- Traditionally in JS, no *classes*
- Remember - JS is not really typed… if it doesn't care between a number and a string, why care between two kinds of objects?

Objects

- What are objects like in other languages? How are they written and organized?
- Traditionally in JS, no *classes*
- Remember - JS is not really typed… if it doesn't care between a number and a string, why care between two kinds of objects?

```
var profHacker = {
    firstName: "Alyssa",
    lastName: “P Hacker",
    teaches: "SWE 432",
    office: "ENGR 6409”,
    fullName: function(){
        return this.firstName + " " + this.lastName;
    }
};
```

Working with Objects

```
var profMoran = {
    firstName: “Alyssa",
    lastName: “P Hacker",
    teaches: "SWE 432",
    office: "ENGR 4448”,
    fullName: function(){
        return this.firstName + " " + this.lastName;
    }
};
```

Our Object

Working with Objects

```
var profMoran = {
    firstName: “Alyssa",
    lastName: “P Hacker",
    teaches: "SWE 432",
    office: "ENGR 4448”,
    fullName: function(){
        return this.firstName + " " + this.lastName;
    }
};
```

Our Object

```
console.log(profHacker.firstName); //Alyssa
console.log(profHacker[“firstName”]); //Alyssa
```

Accessing Fields

Working with Objects

```
var profMoran = {
    firstName: “Alyssa",
    lastName: “P Hacker",
    teaches: "SWE 432",
    office: "ENGR 4448”,
    fullName: function(){
        return this.firstName + " " + this.lastName;
    }
};
```

Our Object

```
console.log(profHacker.firstName); //Alyssa
console.log(profHacker[“firstName”]); //Alyssa
```

Accessing Fields

```
console.log(profHacker.fullName()); //Alyssa P Hacker
```

Calling Methods

Working with Objects

```
var profMoran = {
    firstName: “Alyssa",
    lastName: “P Hacker",
    teaches: "SWE 432",
    office: "ENGR 4448”,
    fullName: function(){
        return this.firstName + " " + this.lastName;
    }
};
```

Our Object

```
console.log(profHacker.firstName); //Alyssa
console.log(profHacker[“firstName”]); //Alyssa
```

Accessing Fields

```
console.log(profHacker.fullName()); //Alyssa P Hacker
```

Calling Methods

```
console.log(profHacker.fullName);
```

Working with Objects

```
var profMoran = {
    firstName: “Alyssa",
    lastName: “P Hacker",
    teaches: "SWE 432",
    office: "ENGR 4448”,
    fullName: function(){
        return this.firstName + " " + this.lastName;
    }
};
```

Our Object

```
console.log(profHacker.firstName); //Alyssa
console.log(profHacker[“firstName”]); //Alyssa
```

Accessing Fields

```
console.log(profHacker.fullName()); //Alyssa P Hacker
```

Calling Methods

```
console.log(profHacker.fullName);//function...
```

Working with Objects

```
var profMoran = {
    firstName: “Alyssa",
    lastName: “P Hacker",
    teaches: "SWE 432",
    office: "ENGR 4448”,
    fullName: function(){
        return this.firstName + " " + this.lastName;
    }
};
```

Our Object

```
console.log(profHacker.firstName); //Alyssa
console.log(profHacker[“firstName”]); //Alyssa
```

Accessing Fields

```
console.log(profHacker.fullName()); //Alyssa P Hacker
```

Calling Methods

```
console.log(profHacker.fullName);//function...
```

JSON: JavaScript Object Notation

Open standard format for transmitting *data* objects.

- No functions, only key / value pairs
- Values may be other objects or arrays

```
var profHacker = {
    firstName: "Alyssa",
    lastName: “P Hacker",
    teaches: "SWE 432",
    office: “ENGR 6409”,
    fullName: function(){
        return this.firstName + " " + this.lastName;
    }
};
```

Our Object

```
var profHacker = {
    firstName: "Alyssa",
    lastName: “P Hacker",
    teaches: "SWE 432",
    office: "ENGR 6409",
    fullName: {
        firstName: “Alyssa”,
        lastName: “P Hacker”}
};
```

JSON Object

Interacting w/ JSON

- Important functions
- JSON.parse(jsonString)
  - Takes a *String* in JSON format, creates an *Object*
- JSON.stringify(obj)
  - Takes a Javascript *object*, creates a JSON *String*
- Useful for persistence, interacting with files, debugging, etc.
  - e.g., console.log(JSON.stringify(obj));

Arrays

- Syntax similar to C/Java/Ruby/Python etc.
- Because JS is loosely typed, can mix types of elements in an array
- Arrays automatically grow/shrink in size to fit the contents

Arrays

- Syntax similar to C/Java/Ruby/Python etc.
- Because JS is loosely typed, can mix types of elements in an array
- Arrays automatically grow/shrink in size to fit the contents

Arrays

- Syntax similar to C/Java/Ruby/Python etc.
- Because JS is loosely typed, can mix types of elements in an array
- Arrays automatically grow/shrink in size to fit the contents

```
var students = ["Alice", "Bob", "Carol"];
```

Arrays

- Syntax similar to C/Java/Ruby/Python etc.
- Because JS is loosely typed, can mix types of elements in an array
- Arrays automatically grow/shrink in size to fit the contents

```
var students = ["Alice", "Bob", "Carol"];
var faculty = [profHacker];
```

Arrays

- Syntax similar to C/Java/Ruby/Python etc.
- Because JS is loosely typed, can mix types of elements in an array
- Arrays automatically grow/shrink in size to fit the contents

```
var students = ["Alice", "Bob", "Carol"];
var faculty = [profHacker];
```

Arrays are actually objects… and come with a bunch of "free" functions

Arrays

- Syntax similar to C/Java/Ruby/Python etc.
- Because JS is loosely typed, can mix types of elements in an array
- Arrays automatically grow/shrink in size to fit the contents

```
var students = ["Alice", "Bob", "Carol"];
var faculty = [profHacker];
var classMembers = students.concat(faculty);
```

Arrays are actually objects… and come with a bunch of "free" functions

Some Array Functions

- Length

```
var numberOfStudents = students.length;
```

- Join

```
var classMembers = students.concat(faculty);
```

- Sort

```
var sortedStudents = students.sort();
```

- Reverse

```
var backwardsStudents = sortedStudents.reverse();
```

- Map

```
var capitalizedStudents = students.map(x =>
                                x.toUpperCase());
  //["ALICE","BOB","CAROL"]
```

For Each

For Each

- JavaScript offers two constructs for looping over arrays and objects

For Each

- JavaScript offers two constructs for looping over arrays and objects
- For **of** (iterates over values):

For Each

- JavaScript offers two constructs for looping over arrays and objects
- For **of** (iterates over values):

For Each

- JavaScript offers two constructs for looping over arrays and objects
- For **of** (iterates over values):

```
for(var student of students)
{
    console.log(student);
} //Prints out all student names
```

For Each

- JavaScript offers two constructs for looping over arrays and objects
- For **of** (iterates over values):

```
for(var student of students)
{
    console.log(student);
} //Prints out all student names
```

- For **in** (iterates over keys):

For Each

- JavaScript offers two constructs for looping over arrays and objects
- For **of** (iterates over values):

```
for(var student of students)
{
    console.log(student);
} //Prints out all student names
```

- For **in** (iterates over keys):

For Each

- JavaScript offers two constructs for looping over arrays and objects
- For **of** (iterates over values):

```
for(var student of students)
{
    console.log(student);
} //Prints out all student names
```

- For **in** (iterates over keys):

```
for(var prop in profHacker){
    console.log(prop + ": " + profHacker[prop]);
}
```

For Each

- JavaScript offers two constructs for looping over arrays and objects
- For **of** (iterates over values):

```
for(var student of students)
{
    console.log(student);
} //Prints out all student names
```

- For **in** (iterates over keys):

```
for(var prop in profHacker){
    console.log(prop + ": " + profHacker[prop]);
}
```

Output:
firstName: Alyssa
lastName: P Hacker
teaches: SWE 432
office: ENGR 6409

Arrays vs Objects

- Arrays are Objects
- Can access elements of both using syntax

```
var val = array[idx];
```

- Indexes of arrays must be integers
- Don’t find out what happens when you make an array and add an element with a non-integer key :)

String Functions

- Includes many of the same String processing functions as Java
- Some examples
  - var stringVal = ‘George Mason University’;
  - stringVal.endsWith(‘University’) // returns true
  - stringVal.match(….) // matches a regular expression
  - stringVal.split(‘ ‘) // returns three separate words

- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String

Template Literals

- Enable embedding expressions **inside** strings

```
var a = 5;
var b = 10;
console.log(`Fifteen is ${a + b} and
not ${2 * a + b}.`);
// "Fifteen is 15 and not 20."
```

- Denoted by a back tick grave accent `, **not** a single quote

Map Collection

```javascript
var myMap = new Map();

var keyString = 'a string',
    keyObj = {},
    keyFunc = function() {};

// setting the values
myMap.set(keyString, "value associated with 'a string'");
myMap.set(keyObj, 'value associated with keyObj');
myMap.set(keyFunc, 'value associated with keyFunc');

myMap.size; // 3

// getting the values
myMap.get(keyString);    // "value associated with 'a string'"
myMap.get(keyObj);       // "value associated with keyObj"
myMap.get(keyFunc);      // "value associated with keyFunc"

myMap.get('a string');   // "value associated with 'a string'"
                         // because keyString === 'a string'
myMap.get({});           // undefined, because keyObj !== {}
myMap.get(function() {}) // undefined, because keyFunc !== function () {}
```

Week 2: Class Intro & Javascript

Design Goals

- Within a component
  - Cohesive
  - Complete
  - Convenient
  - Clear
  - Consistent
- Between components
  - Low coupling

Cohesion and Coupling

- Cohesion is a property or characteristic of an individual unit
- Coupling is a property of a collection of units
- High cohesion GOOD, high coupling BAD
- Design for change:
  - Reduce interdependency (coupling): You don't want a change in one unit to ripple throughout your system
  - Group functionality (cohesion): Easier to find things, intuitive metaphor aids understanding

Design for Reuse

- Why?
  - Don’t duplicate existing functionality
  - Avoid repeated effort
- How?
  - Make it easy to extract a single component:
    - Low ***coupling*** between components
    - Have high ***cohesion*** within a component

Design for Change

- Why?
  - Want to be able to add new features
  - Want to be able to easily ***maintain*** existing software
    - Adapt to new environments
    - Support new configurations
- How?
  - Low ***coupling*** - prevents unintended side effects
  - High ***cohesion*** - easier to find things

Organizing Code

How do we structure things to achieve good organization?

	Java	Javascript

Organizing Code

How do we structure things to achieve good organization?

	Java	Javascript
Individual Pieces of Functional Components		

Organizing Code

How do we structure things to achieve good organization?

	Java	Javascript
Individual Pieces of Functional Components	Classes	

Organizing Code

How do we structure things to achieve good organization?

	Java	Javascript
Individual Pieces of Functional Components	Classes	Classes

Organizing Code

How do we structure things to achieve good organization?

	Java	Javascript
Individual Pieces of Functional Components	Classes	Classes
Entire libraries		

Organizing Code

How do we structure things to achieve good organization?

	Java	Javascript
Individual Pieces of Functional Components	Classes	Classes
Entire libraries	Packages	

Organizing Code

How do we structure things to achieve good organization?

	Java	Javascript
Individual Pieces of Functional Components	Classes	Classes
Entire libraries	Packages	Modules

Classes

- ES6 introduces the `class` keyword
- Mainly just syntax - still not like Java Classes

Classes

- ES6 introduces the `class` keyword
- Mainly just syntax - still not like Java Classes

Old

```
function Faculty(first, last, teaches, office)
{
  this.firstName = first;
  this.lastName = last;
  this.teaches = teaches;
  this.office = office;
  this.fullName = function(){
    return this.firstName + " " + this.lastName;
  }
}
var prof = new Faculty("Kevin", "Moran", "SWE432", "ENGR 4448”);
```

Classes

- ES6 introduces the `class` keyword
- Mainly just syntax - still not like Java Classes

Old

```
function Faculty(first, last, teaches, office)
{
  this.firstName = first;
  this.lastName = last;
  this.teaches = teaches;
  this.office = office;
  this.fullName = function(){
    return this.firstName + " " + this.lastName;
  }
}
var prof = new Faculty("Kevin", "Moran", "SWE432", "ENGR 4448”);
```

New

```
class Faculty {
    constructor(first, last, teaches, office)
    {
        this.firstName = first;
        this.lastName = last;
        this.teaches = teaches;
        this.office = office;
    }
    fullname() {
        return this.firstName + " " + this.lastName;
    }
}
var prof = new Faculty("Kevin", "Moran", "SWE432", "ENGR 4448”);
```

Modules (ES6)

- With ES6, there is (finally!) language support for modules
- Module must be defined in its own JS file
- Modules **`export`** declarations
  - Publicly exposes functions as part of module interface
- Code **`imports`** modules (and optionally only parts of them)
  - Specify module by path to the file

Modules (ES6) - Export Syntax

```
var faculty = [{name:"Prof Johnson", section: 2}, {name:"Prof Moran”,
section:1}];
export function getFaculty(i) {
    // ..
}
export var someVar = [1,2,3];
```

Label each declaration with “export”

Modules (ES6) - Export Syntax

```
var faculty = [{name:"Prof Johnson", section: 2}, {name:"Prof Moran”,
section:1}];
export function getFaculty(i) {
    // ..
}
export var someVar = [1,2,3];
```

Label each declaration with “export”

Modules (ES6) - Export Syntax

```
var faculty = [{name:"Prof Johnson", section: 2}, {name:"Prof Moran”,
section:1}];
export function getFaculty(i) {
    // ..
}
export var someVar = [1,2,3];
```

Label each declaration with “export”

```
var faculty = [{name:"Prof Johnson", section: 2}, {name:"Prof Moran”,
section:1}];
var someVar = [1,2,3];
function getFaculty(i) {
    // ..
}
export {getFaculty, someVar};
```

Modules (ES6) - Export Syntax

```
var faculty = [{name:"Prof Johnson", section: 2}, {name:"Prof Moran”,
section:1}];
export function getFaculty(i) {
    // ..
}
export var someVar = [1,2,3];
```

Label each declaration with “export”

```
var faculty = [{name:"Prof Johnson", section: 2}, {name:"Prof Moran”,
section:1}];
var someVar = [1,2,3];
function getFaculty(i) {
    // ..
}
export {getFaculty, someVar};
```

Modules (ES6) - Export Syntax

```
var faculty = [{name:"Prof Johnson", section: 2}, {name:"Prof Moran",
section:1}];
export function getFaculty(i) {
    // ..
}
export var someVar = [1,2,3];
```

Label each declaration with "export"

```
var faculty = [{name:"Prof Johnson", section: 2}, {name:"Prof Moran",
section:1}];
var someVar = [1,2,3];
function getFaculty(i) {
    // ..
}
export {getFaculty, someVar};
```

Or name all of the exports at once

Modules (ES6) - Export Syntax

```
var faculty = [{name:"Prof Johnson", section: 2}, {name:"Prof Moran",
section:1}];
export function getFaculty(i) {
    // ..
}
export var someVar = [1,2,3];
```

Label each declaration with “export”

```
var faculty = [{name:"Prof Johnson", section: 2}, {name:"Prof Moran",
section:1}];
var someVar = [1,2,3];
function getFaculty(i) {
    // ..
}
export {getFaculty, someVar};

export {getFaculty as aliasForFunction, someVar};
```

Or name all of the exports at once

Modules (ES6) - Export Syntax

```
var faculty = [{name:"Prof Johnson", section: 2}, {name:"Prof Moran",
section:1}];
export function getFaculty(i) {
    // ..
}
export var someVar = [1,2,3];
```

Label each declaration with “export”

```
var faculty = [{name:"Prof Johnson", section: 2}, {name:"Prof Moran",
section:1}];
var someVar = [1,2,3];
function getFaculty(i) {
    // ..
}
export {getFaculty, someVar};
```

Or name all of the exports at once

Can rename exports too

```
export {getFaculty as aliasForFunction, someVar};
```

Modules (ES6) - Export Syntax

```
var faculty = [{name:"Prof Johnson", section: 2}, {name:"Prof Moran",
section:1}];
export function getFaculty(i) {
    // ..
}
export var someVar = [1,2,3];
```

Label each declaration with “export”

```
var faculty = [{name:"Prof Johnson", section: 2}, {name:"Prof Moran",
section:1}];
var someVar = [1,2,3];
function getFaculty(i) {
    // ..
}
export {getFaculty, someVar};
```

Or name all of the exports at once

Can rename exports too

```
export {getFaculty as aliasForFunction, someVar};


export default function getFaculty(i){...
```

Modules (ES6) - Export Syntax

```
var faculty = [{name:"Prof Johnson", section: 2}, {name:"Prof Moran",
section:1}];
export function getFaculty(i) {
    // ..
}
export var someVar = [1,2,3];
```

Label each declaration with “export”

```
var faculty = [{name:"Prof Johnson", section: 2}, {name:"Prof Moran",
section:1}];
var someVar = [1,2,3];
function getFaculty(i) {
    // ..
}
export {getFaculty, someVar};
```

Or name all of the exports at once

Can rename exports too

```
export {getFaculty as aliasForFunction, someVar};
```

```
export default function getFaculty(i){...
```

Default export

Modules (ES6) - Import Syntax

Modules (ES6) - Import Syntax

- Import specific exports, binding them to the same name

Modules (ES6) - Import Syntax

- Import specific exports, binding them to the same name

```
import { getFaculty, someVar } from "myModule";
getFaculty()...
```

Modules (ES6) - Import Syntax

- Import specific exports, binding them to the same name

```
import { getFaculty, someVar } from "myModule";
getFaculty()...
```

- Import specific exports, binding them to a new name

Modules (ES6) - Import Syntax

- Import specific exports, binding them to the same name

```
import { getFaculty, someVar } from "myModule";
getFaculty()...
```

- Import specific exports, binding them to a new name

```
import { getFaculty as aliasForFaculty } from "myModule";
aliasForFaculty()...
```

Modules (ES6) - Import Syntax

- Import specific exports, binding them to the same name

```
import { getFaculty, someVar } from "myModule";
getFaculty()...
```

- Import specific exports, binding them to a new name

```
import { getFaculty as aliasForFaculty } from "myModule";
aliasForFaculty()...
```

- Import default export, binding to specified name

Modules (ES6) - Import Syntax

- Import specific exports, binding them to the same name

```
import { getFaculty, someVar } from "myModule";
getFaculty()...
```

- Import specific exports, binding them to a new name

```
import { getFaculty as aliasForFaculty } from "myModule";
aliasForFaculty()...
```

- Import default export, binding to specified name

```
import theThing from "myModule";
theThing()... -> calls getFaculty()
```

Modules (ES6) - Import Syntax

- Import specific exports, binding them to the same name

```
import { getFaculty, someVar } from "myModule";
getFaculty()...
```

- Import specific exports, binding them to a new name

```
import { getFaculty as aliasForFaculty } from "myModule";
aliasForFaculty()...
```

- Import default export, binding to specified name

```
import theThing from "myModule";
theThing()... -> calls getFaculty()
```

- Import all exports, binding to specified name

Modules (ES6) - Import Syntax

- Import specific exports, binding them to the same name

```
import { getFaculty, someVar } from "myModule";
getFaculty()...
```

- Import specific exports, binding them to a new name

```
import { getFaculty as aliasForFaculty } from "myModule";
aliasForFaculty()...
```

- Import default export, binding to specified name

```
import theThing from "myModule";
theThing()... -> calls getFaculty()
```

- Import all exports, binding to specified name

```
import * as facModule from "myModule";
facModule.getFaculty()...
```

Cascade Pattern

Cascade Pattern

- aka “chaining”

Cascade Pattern

- aka “chaining”
- Offer set of operations that mutate object and returns the “this” object

Cascade Pattern

- aka “chaining”
- Offer set of operations that mutate object and returns the “this” object
  - Build an API that has single purpose operations that can be combined easily

Cascade Pattern

- aka “chaining”
- Offer set of operations that mutate object and returns the “this” object
  - Build an API that has single purpose operations that can be combined easily
  - Lets us read code like a sentence

Cascade Pattern

- aka “chaining”
- Offer set of operations that mutate object and returns the “this” object
  - Build an API that has single purpose operations that can be combined easily
  - Lets us read code like a sentence
- Example (String):

Cascade Pattern

- aka “chaining”
- Offer set of operations that mutate object and returns the “this” object
  - Build an API that has single purpose operations that can be combined easily
  - Lets us read code like a sentence
- Example (String):

Cascade Pattern

- aka “chaining”
- Offer set of operations that mutate object and returns the “this” object
  - Build an API that has single purpose operations that can be combined easily
  - Lets us read code like a sentence
- Example (String):

```
str.replace("k","R").toUpperCase().substr(0,4);
```

Cascade Pattern

- aka “chaining”
- Offer set of operations that mutate object and returns the “this” object
  - Build an API that has single purpose operations that can be combined easily
  - Lets us read code like a sentence
- Example (String):

```
str.replace("k","R").toUpperCase().substr(0,4);
```

- Example (jQuery):

Cascade Pattern

- aka “chaining”
- Offer set of operations that mutate object and returns the “this” object
  - Build an API that has single purpose operations that can be combined easily
  - Lets us read code like a sentence
- Example (String):

```
str.replace("k","R").toUpperCase().substr(0,4);
```

- Example (jQuery):

Cascade Pattern

- aka “chaining”
- Offer set of operations that mutate object and returns the “this” object
  - Build an API that has single purpose operations that can be combined easily
  - Lets us read code like a sentence
- Example (String):

```
str.replace("k","R").toUpperCase().substr(0,4);
```

- Example (jQuery):

```
$(“#wrapper")
        .fadeOut()
        .html(“Welcome")
        .fadeIn();
```

Cascade Pattern

```
function number(value) {
  this.value = value;

  this.plus = function (sum) {
    this.value += sum;
    return this;
  };

  this.return = function () {
    return this.value;
  };

  return this;
}

console.log(new number(5).plus(1).return());
```

Closures

- Closures are expressions that work with variables in a specific context
- Closures contain a function, and its needed state
  - Closure is that function and a ***stack frame*** that is allocated when a function starts executing and ***not freed*** after the function returns

Closures & Stack Frames

- What is a stack frame?
  - Variables created by function in its execution
  - Maintained by environment executing code

Closures & Stack Frames

- What is a stack frame?
  - Variables created by function in its execution
  - Maintained by environment executing code

```
function a() {
    var x = 5, z = 3;
    b(x);
}
function b(y)  {
    console.log(y);
}
a();
```

Closures & Stack Frames

- What is a stack frame?
  - Variables created by function in its execution
  - Maintained by environment executing code

```
function a() {
    var x = 5, z = 3;
    b(x);
}
function b(y)  {
    console.log(y);
}
a();
```

Function called: stack frame created

Closures & Stack Frames

- What is a stack frame?
  - Variables created by function in its execution
  - Maintained by environment executing code

```
function a() {
    var x = 5, z = 3;
    b(x);
}
function b(y)  {
    console.log(y);
}
a();
```

Contents of memory:

```
a:    x: 5
      z: 3
```

Stack frame

Function called: stack frame created

Closures & Stack Frames

- What is a stack frame?
  - Variables created by function in its execution
  - Maintained by environment executing code

```
function a() {
    var x = 5, z = 3;
    b(x);
}
function b(y)  {
    console.log(y);
}
a();
```

```
a:    x: 5
      z: 3
```

Stack frame

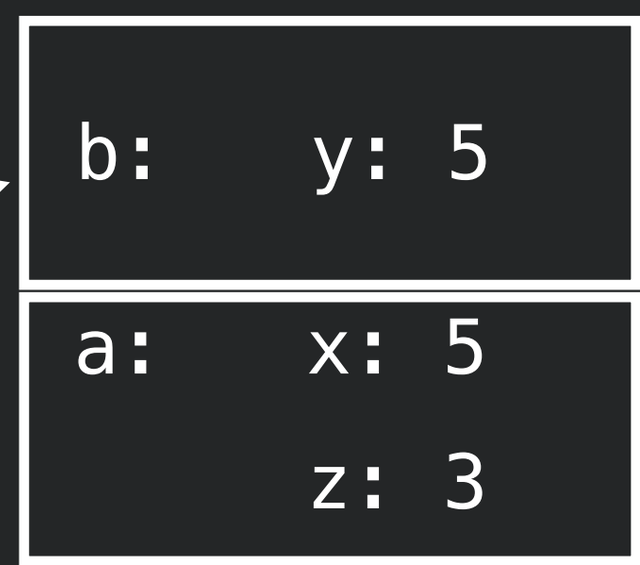
Function called: stack frame created

Closures & Stack Frames

- What is a stack frame?
  - Variables created by function in its execution
  - Maintained by environment executing code

```
function a() {
    var x = 5, z = 3;
    b(x);
}
function b(y)  {
    console.log(y);
}
a();
```

Contents of memory:



Stack frame

Function called: stack frame created

Closures & Stack Frames

- What is a stack frame?
  - Variables created by function in its execution
  - Maintained by environment executing code

```
function a() {
    var x = 5, z = 3;
    b(x);
}
function b(y)  {
    console.log(y);
}
a();
```

Stack frame

Function called: stack frame created

Closures & Stack Frames

- What is a stack frame?
  - Variables created by function in its execution
  - Maintained by environment executing code

```
function a() {
    var x = 5, z = 3;
    b(x);
}
function b(y)  {
    console.log(y);
}
a();
```

Contents of memory:

```
a:    x: 5
      z: 3
```

Stack frame

Function called: stack frame created

Closures

- Closures are expressions that work with variables in a specific context
- Closures contain a function, and its needed state
  - Closure is a stack frame that is allocated when a function starts executing and not freed after the function returns
- That state just refers to that state by name (sees updates)

Closures

- Closures are expressions that work with variables in a specific context
- Closures contain a function, and its needed state
  - Closure is a stack frame that is allocated when a function starts executing and not freed after the function returns
- That state just refers to that state by name (sees updates)

```
var x = 1;
function f() {
    var y = 2;
    return function() {
            console.log(x + y);
            y++;
    };
}
var g = f();
g();            // 1+2 is 3
g();            // 1+3 is 4
```

Closures

- Closures are expressions that work with variables in a specific context
- Closures contain a function, and its needed state
  - Closure is a stack frame that is allocated when a function starts executing and not freed after the function returns
- That state just refers to that state by name (sees updates)

```
var x = 1;
function f() {
    var y = 2;
    return function() {
            console.log(x + y);
            y++;
    };
}
var g = f();
g();            // 1+2 is 3
g();            // 1+3 is 4
```

This function attaches itself to x and y so that it can continue to access them.

Closures

- Closures are expressions that work with variables in a specific context
- Closures contain a function, and its needed state
  - Closure is a stack frame that is allocated when a function starts executing and not freed after the function returns
- That state just refers to that state by name (sees updates)

```
var x = 1;
function f() {
    var y = 2;
    return function() {
        console.log(x + y);
        y++;
    };
}
var g = f();
g();            // 1+2 is 3
g();            // 1+3 is 4
```

This function attaches itself to x and y so that it can continue to access them.

Closures

- Closures are expressions that work with variables in a specific context
- Closures contain a function, and its needed state
  - Closure is a stack frame that is allocated when a function starts executing and not freed after the function returns
- That state just refers to that state by name (sees updates)

```
var x = 1;
function f() {
    var y = 2;
    return function() {
        console.log(x + y);
        y++;
    };
}
var g = f();
g();            // 1+2 is 3
g();            // 1+3 is 4
```

This function attaches itself to x and y so that it can continue to access them.

It "**closes up**" those references

Closures

Closures

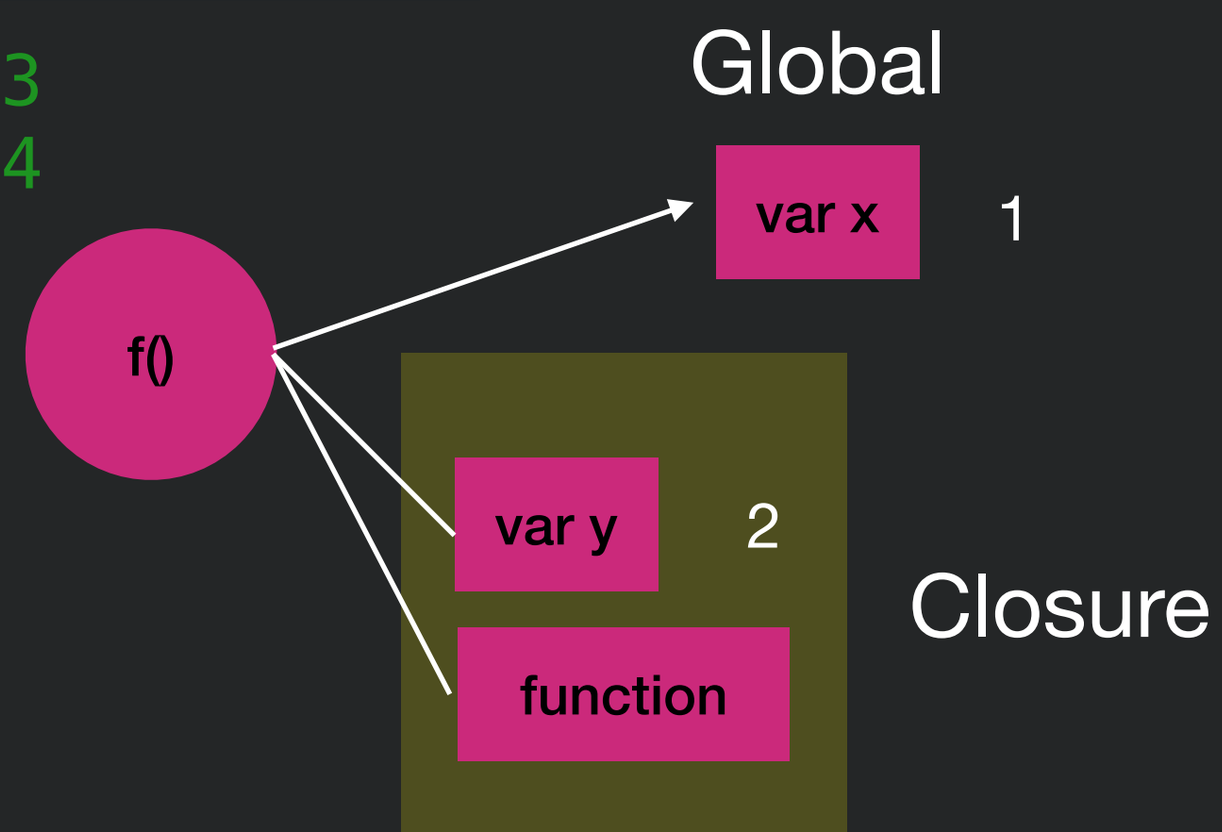
```
var x = 1;
function f() {
    var y = 2;
    return function() {
            console.log(x + y);
            y++;
    };
}
var g = f();
g();            // 1+2 is 3
g();            // 1+3 is 4
```

Closures

```
var x = 1;
function f() {
    var y = 2;
    return function() {
        console.log(x + y);
        y++;
    };
}
var g = f();
g();            // 1+2 is 3
g();            // 1+3 is 4
```

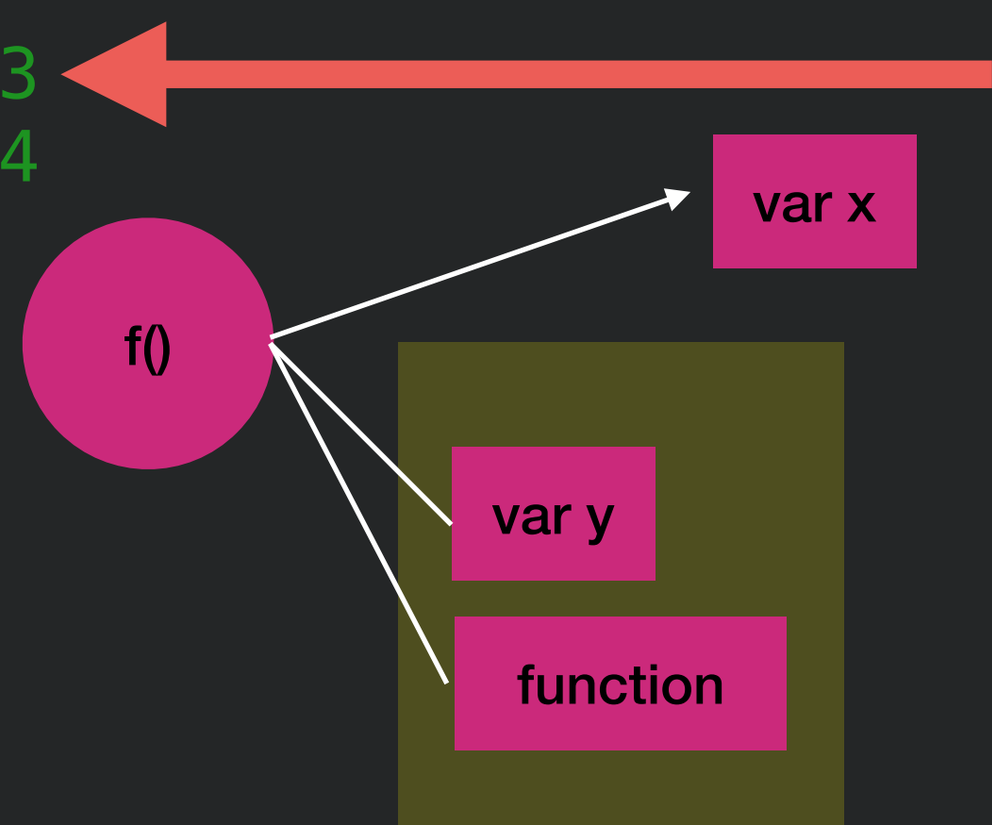
Closures

```
var x = 1;
function f() {
    var y = 2;
    return function() {
        console.log(x + y);
        y++;
    };
}
var g = f();
g();            // 1+2 is 3
g();            // 1+3 is 4
```



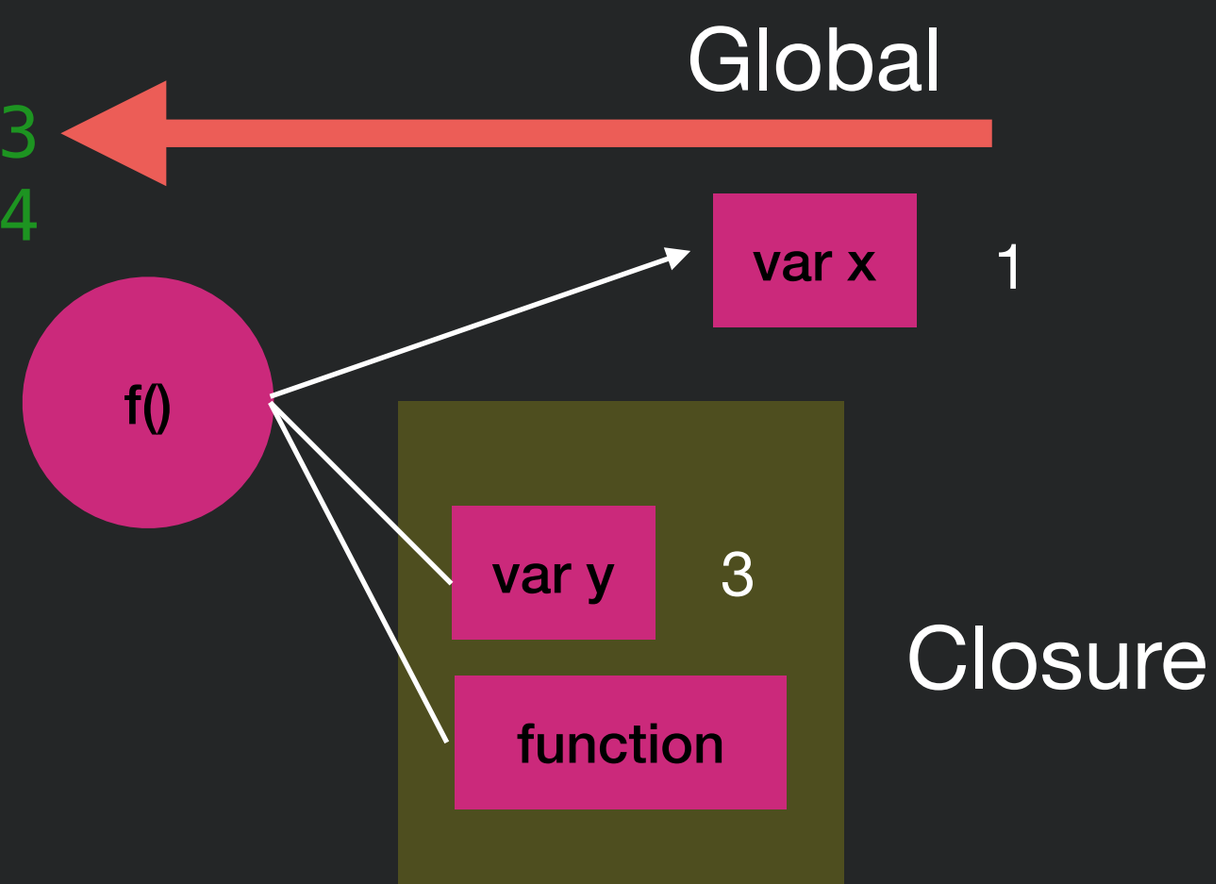
Closures

```
var x = 1;
function f() {
    var y = 2;
    return function() {
        console.log(x + y);
        y++;
    };
}
var g = f();
g();            // 1+2 is 3
g();            // 1+3 is 4
```



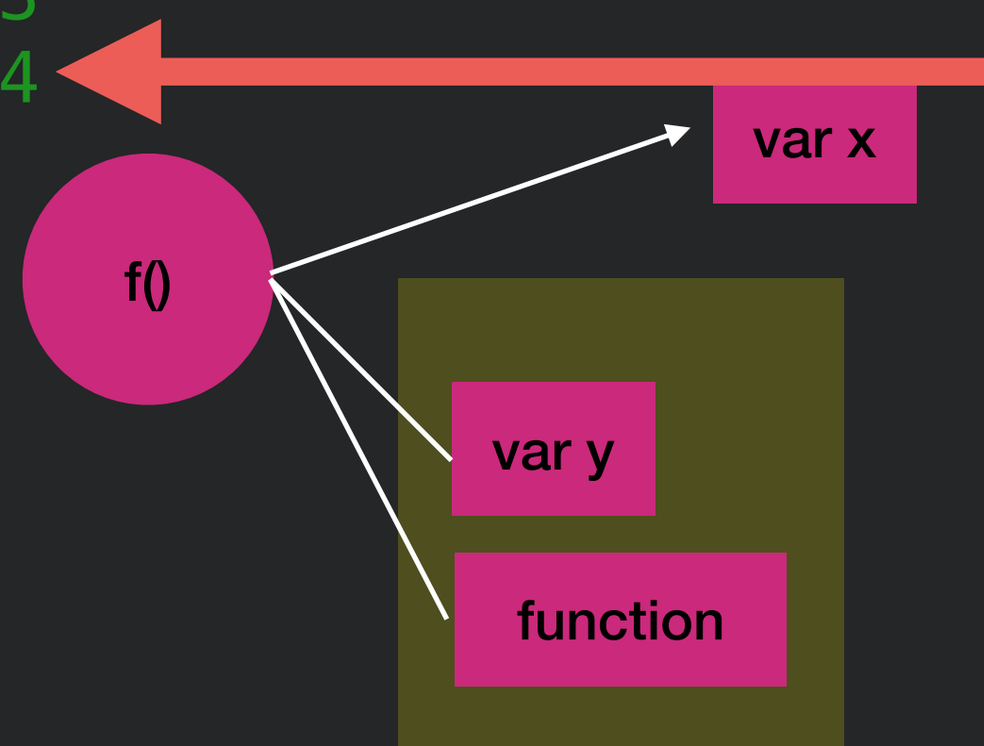
Closures

```
var x = 1;
function f() {
    var y = 2;
    return function() {
        console.log(x + y);
        y++;
    };
}
var g = f();
g();            // 1+2 is 3
g();            // 1+3 is 4
```



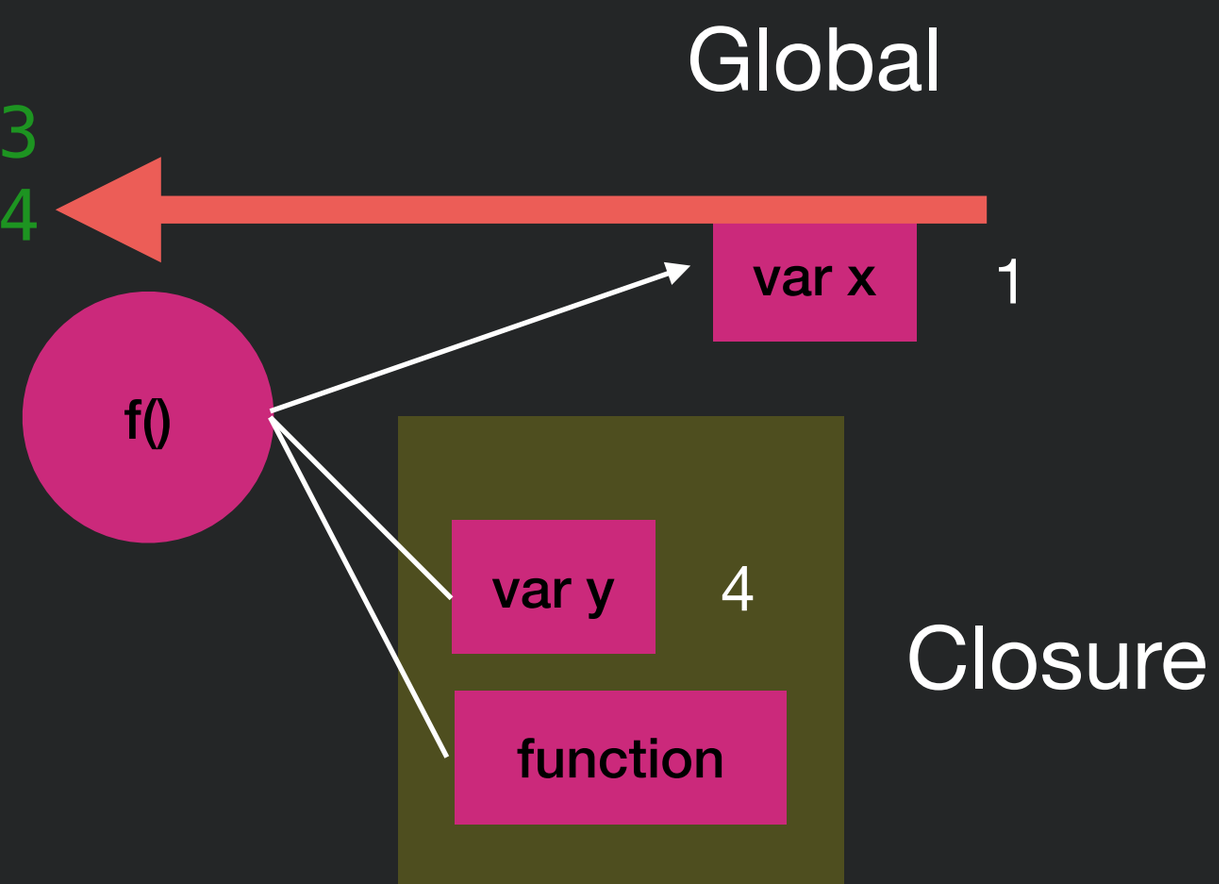
Closures

```
var x = 1;
function f() {
    var y = 2;
    return function() {
        console.log(x + y);
        y++;
    };
}
var g = f();
g();            // 1+2 is 3
g();            // 1+3 is 4
```



Closures

```
var x = 1;
function f() {
    var y = 2;
    return function() {
        console.log(x + y);
        y++;
    };
}
var g = f();
g();            // 1+2 is 3
g();            // 1+3 is 4
```



Modules with Closures

```
var facultyAPI = (function(){
  var faculty = [{name:"Prof Johnson", section: 2}, {name:"Prof
Moran", section:1}];

  return {
   getFaculty : function(i){
    return faculty[i].name + " (" + faculty[i].section + ")";
   }

 };
})();

console.log(facultyAPI.getFaculty(0));
```

Modules with Closures

```
var facultyAPI = (function(){
  var faculty = [{name:"Prof Johnson", section: 2}, {name:"Prof
Moran", section:1}];

  return {
   getFaculty : function(i){
    return faculty[i].name + " (" + faculty[i].section + ")";
   }

 };
})();

console.log(facultyAPI.getFaculty(0));
```

This works because inner functions have visibility to all variables of outer functions!

Closures Gone Awry

```
var result = [];
for (var i = 0; i < 5; i++) {
    result[i] = function() {
        console.log(i);
  };
}
```

```
result[0]() // 5, expected 0
result[1]() // 5, expected 1
result[2]() // 5, expected 2
result[3]() // 5, expected 3
result[4]() // 5, expected 4
```

Closures Gone Awry

```
var result = [];
for (var i = 0; i < 5; i++) {
    result[i] = function() {
        console.log(i);
  };
}
```

What is the output of `result[0]()`?

```
result[0]() // 5, expected 0
result[1]() // 5, expected 1
result[2]() // 5, expected 2
result[3]() // 5, expected 3
result[4]() // 5, expected 4
```

Closures Gone Awry

```
var result = [];
for (var i = 0; i < 5; i++) {
    result[i] = function() {
        console.log(i);
  };
}
```

What is the output of `result[0]()`?

```
result[0](); // 5, expected 0
result[1](); // 5, expected 1
result[2](); // 5, expected 2
result[3](); // 5, expected 3
result[4](); // 5, expected 4
```

Why?

Closures Gone Awry

```
var result = [];
for (var i = 0; i < 5; i++) {
    result[i] = function() {
        console.log(i);
  };
}
```

What is the output of `result[0]()`?

```
result[0]() // 5, expected 0
result[1]() // 5, expected 1
result[2]() // 5, expected 2
result[3]() // 5, expected 3
result[4]() // 5, expected 4
```

Why?

Closures Gone Awry

```
var result = [];
for (var i = 0; i < 5; i++) {
    result[i] = function() {
        console.log(i);
  };
}
```

What is the output of `result[0]()`?

```
result[0]() // 5, expected 0
result[1]() // 5, expected 1
result[2]() // 5, expected 2
result[3]() // 5, expected 3
result[4]() // 5, expected 4
```

Why?

Closures retain a *pointer* to their needed state!

Closures Under Control

Solution: IIFE - Immediately-Invoked Function Expression

```
function makeFunction(n)
{
    return function(){ return n; };
}
for (var i = 0; i < 5; i++) {
  result[i] = makeFunction(i);
}
```

Closures Under Control

Solution: IIFE - Immediately-Invoked Function Expression

```
function makeFunction(n)
{
    return function(){ return n; };
}
for (var i = 0; i < 5; i++) {
  result[i] = makeFunction(i);
}
```

Closures Under Control

Solution: IIFE - Immediately-Invoked Function Expression

```
function makeFunction(n)
{
    return function(){ return n; };
}
for (var i = 0; i < 5; i++) {
  result[i] = makeFunction(i);
}
```

```
result[0]() ; // 0, expected 0
result[1]() ; // 1, expected 1
result[2]() ; // 2, expected 2
result[3]() ; // 3, expected 3
result[4]() ; // 4, expected 4
```

Closures Under Control

Solution: IIFE - Immediately-Invoked Function Expression

```
function makeFunction(n)
{
    return function(){ return n; };
}
for (var i = 0; i < 5; i++) {
  result[i] = makeFunction(i);
}
```

```
result[0]() ; // 0, expected 0
result[1]() ; // 1, expected 1
result[2]() ; // 2, expected 2
result[3]() ; // 3, expected 3
result[4]() ; // 4, expected 4
```

Why does it work?

Closures Under Control

Solution: IIFE - Immediately-Invoked Function Expression

```
function makeFunction(n)
{
    return function(){ return n; };
}
for (var i = 0; i < 5; i++) {
  result[i] = makeFunction(i);
}
```

```
result[0]();  // 0, expected 0
result[1]();  // 1, expected 1
result[2]();  // 2, expected 2
result[3]();  // 3, expected 3
result[4]();  // 4, expected 4
```

Why does it work?

Each time the anonymous function is called, it will create a ***new variable*** **n**, rather than reusing the same variable `i`

Closures Under Control

Solution: IIFE - Immediately-Invoked Function Expression

```
function makeFunction(n)
{
    return function(){ return n; };
}
for (var i = 0; i < 5; i++) {
  result[i] = makeFunction(i);
}
```

```
result[0]() ; // 0, expected 0
result[1]() ; // 1, expected 1
result[2]() ; // 2, expected 2
result[3]() ; // 3, expected 3
result[4]() ; // 4, expected 4
```

Why does it work?

Each time the anonymous function is called, it will create a ***new variable*** **n**, rather than reusing the same variable `i`

Shortcut syntax:

```
var result = [];
for (var i = 0; i < 5; i++) {
  result[i] = (function(n) {
    return function() { return n; }
  })(i);
}
```

Closures Under Control

Solution: IIFE - Immediately-Invoked Function Expression

```
function makeFunction(n)
{
    return function(){ return n; };
}
for (var i = 0; i < 5; i++) {
  result[i] = makeFunction(i);
}
```

```
result[0]() ; // 0, expected 0
result[1]() ; // 1, expected 1
result[2]() ; // 2, expected 2
result[3]() ; // 3, expected 3
result[4]() ; // 4, expected 4
```

Why does it work?

Each time the anonymous function is called, it will create a ***new variable*** **n**, rather than reusing the same variable `i`

Shortcut syntax:

```
var result = [];
for (var i = 0; i < 5; i++) {
  result[i] = (function(n) {
    return function() { return n; }
  })(i);
}
```

NPM: Not an acronym, but the Node Package Manager

NPM: Not an acronym, but the Node Package Manager

- Bring order to our modules and dependencies

NPM: Not an acronym, but the Node Package Manager

- Bring order to our modules and dependencies
- Declarative approach:

NPM: Not an acronym, but the Node Package Manager

- Bring order to our modules and dependencies
- Declarative approach:
  - “My app is called helloworld”

NPM: Not an acronym, but the Node Package Manager

- Bring order to our modules and dependencies
- Declarative approach:
  - “My app is called helloworld”
  - “It is version 1”

NPM: Not an acronym, but the Node Package Manager

- Bring order to our modules and dependencies
- Declarative approach:
  - “My app is called helloworld”
  - “It is version 1”
  - You can run it by saying “node index.js”

NPM: Not an acronym, but the Node Package Manager

- Bring order to our modules and dependencies
- Declarative approach:
  - “My app is called helloworld”
  - “It is version 1”
  - You can run it by saying “node index.js”
  - “I need express, the most recent version is fine”

NPM: Not an acronym, but the Node Package Manager

- Bring order to our modules and dependencies
- Declarative approach:
  - “My app is called helloworld”
  - “It is version 1”
  - You can run it by saying “node index.js”
  - “I need express, the most recent version is fine”
- Config is stored in json - specifically package.json

NPM: Not an acronym, but the Node Package Manager

- Bring order to our modules and dependencies
- Declarative approach:
  - “My app is called helloworld”
  - “It is version 1”
  - You can run it by saying “node index.js”
  - “I need express, the most recent version is fine”
- Config is stored in json - specifically package.json

Generated by `npm` commands:

```
{
  "name": "helloworld",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test
specified\" && exit 1"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "express": "^4.14.0"
  }
}
```

Installing packages with NPM

- `` `npm install <package> --save` `` will download a package and add it to your package.json
- `` `npm install` `` will go through all of the packages in package.json and make sure they are installed/up to date
- Packages get installed to the `` `node_modules` `` directory in your project

Using NPM

https://docs.npmjs.com/index

Using NPM

- Your “project” is a directory which contains a special file, package.json

https://docs.npmjs.com/index

Using NPM

- Your “project” is a directory which contains a special file, package.json
- Everything that is going to be in your project goes in this directory

https://docs.npmjs.com/index

Using NPM

- Your “project” is a directory which contains a special file, package.json
- Everything that is going to be in your project goes in this directory
- Step 1: Create NPM project

```
npm init
```

https://docs.npmjs.com/index

Using NPM

- Your “project” is a directory which contains a special file, package.json
- Everything that is going to be in your project goes in this directory
- Step 1: Create NPM project
  ```
  npm init
  ```
- Step 2: Declare dependencies
  ```
  npm install <packagename> --save
  ```

https://docs.npmjs.com/index

Using NPM

- Your “project” is a directory which contains a special file, package.json
- Everything that is going to be in your project goes in this directory
- Step 1: Create NPM project
  ```
  npm init
  ```
- Step 2: Declare dependencies
  ```
  npm install <packagename> --save
  ```
- Step 3: Use modules in your app
  ```
  var myPkg = require(“packagename”)
  ```

https://docs.npmjs.com/index

Using NPM

- Your “project” is a directory which contains a special file, package.json
- Everything that is going to be in your project goes in this directory
- Step 1: Create NPM project
  ```
  npm init
  ```
- Step 2: Declare dependencies
  ```
  npm install <packagename> --save
  ```
- Step 3: Use modules in your app
  ```
  var myPkg = require(“packagename”)
  ```
- Do NOT include node_modules in your git repo! Instead, just do
  ```
  npm install
  ```

https://docs.npmjs.com/index

Using NPM

- Your “project” is a directory which contains a special file, package.json
- Everything that is going to be in your project goes in this directory
- Step 1: Create NPM project
  ```
  npm init
  ```
- Step 2: Declare dependencies
  ```
  npm install <packagename> --save
  ```
- Step 3: Use modules in your app
  ```
  var myPkg = require(“packagename”)
  ```
- Do NOT include node_modules in your git repo! Instead, just do
  ```
  npm install
  ```
  - This will download and install the modules on your machine given the existing config!

https://docs.npmjs.com/index

Unit Testing

- Unit testing is testing some program unit in isolation from the rest of the system (which may not exist yet)
- Usually the programmer is responsible for testing a unit during its implementation
- Easier to debug when a test finds a bug (compared to full-system testing)

Integration Testing

- ***Motivation:*** Units that worked in isolation may not work in combination
- Performed after all units to be integrated have passed all unit tests
- Reuse unit test cases that cross unit boundaries (that previously required stub(s) and/or driver standing in for another unit)

Jest Lets You Specify Behavior in *Specs*

Jest Lets You Specify Behavior in *Specs*

- Specs are written in JS

Jest Lets You Specify Behavior in *Specs*

- Specs are written in JS
- Key functions:
  - `describe`, `test`, `expect`

Jest Lets You Specify Behavior in *Specs*

- Specs are written in JS
- Key functions:
  - `describe`, `test`, `expect`
- `Describe` a high level scenario by providing a name for the scenario and function(s) that contains some `tests` by saying what you `expect` it to be

Jest Lets You Specify Behavior in *Specs*

- Specs are written in JS
- Key functions:
  - `describe`, `test`, `expect`
- `Describe` a high level scenario by providing a name for the scenario and function(s) that contains some `tests` by saying what you `expect` it to be
- Example:

```
describe("Alyssa P Hacker tests", () => {
    test("Calling fullName directly should always work", () => {
        expect(profHacker.fullName()).toEqual("Alyssa P Hacker");
    });
}
```

Writing Specs

- Can specify some code to run before or after checking a spec

```
var profHacker;
beforeEach(() => {
    profHacker = {
        firstName: "Alyssa",
        lastName: "P Hacker",
        teaches: "SWE 432",
        office: "ENGR 6409",
        fullName: function () {
            return this.firstName + " " + this.lastName;
        }
    };
});
```

Making it work

- Add `jest` library to your project (npm install --save-dev jest)
- Configure NPM to use jest for test in package.json

```
"scripts": {
  "test": "jest"
},
```

- For file `x.js`, create `x.test.js`
- Run `npm test`

Multiple Specs

- Can have as many tests as you would like

```
test("Calling fullName directly should always work", () => {
     expect(profHacker.fullName()).toEqual("Alyssa P Hacker");
 });

test("Calling fullName without binding but with a function ref is undefined", () => {
    var func = profHacker.fullName;
    expect(func()).toEqual("undefined undefined");
});
test("Calling fullName WITH binding with a function ref works", () => {
    var func = profHacker.fullName;
    func = func.bind(profHacker);
    expect(func()).toEqual("Alyssa P Hacker");
});
test("Changing name changes full name", ()=>{
    profHacker.firstName = "Dr. Alyssa";
    expect(profHacker.fullName()).toEqual("Dr. Alyssa P Hacker");
})
```

Nesting Specs

- "When its current price is higher than the paid price:
  - It should have a positive return of investment
  - It should be a good investment"
- How do we describe that?

```
describe("when its current price is higher than the paid price", function() {
    beforeEach(function() {
      stock.sharePrice = 40;
    });
    test("should have a positive return of investment", function() {
      expect(investment.roi()).toBeGreaterThan(0);
    });
    test("should be a good investment", function() {
      expect(investment.isGood()).toBeTruthy();
    });
  });
});
```

Matchers

- How does Jest determine that something is what we expect?

```
expect(investment.roi()).toBeGreaterThan(0);
expect(investment).isGood().toBeTruthy();
expect(investment.shares).toEqual(100);
expect(investment.stock).toBe(stock);
```

- These are “matchers” for Jest - that compare a given value to some criteria
- Basic matchers are built in:
  - toBe, toEqual, toContain, toBeNaN, toBeNull, toBeUndefined, >, <, >=, <=, !=, regular expressions
- Can also define your own matcher

Matchers

```
test('null', () => {
  const n = null;
  expect(n).toBeNull();
  expect(n).toBeDefined();
  expect(n).not.toBeUndefined();
});

const shoppingList = [
  'diapers',
  'kleenex',
  'trash bags',
  'paper towels',
  'beer',
];

test('the shopping list has beer on it', () => {
  expect(shoppingList).toContain('beer');
  expect(new Set(shoppingList)).toContain('beer');
});
```

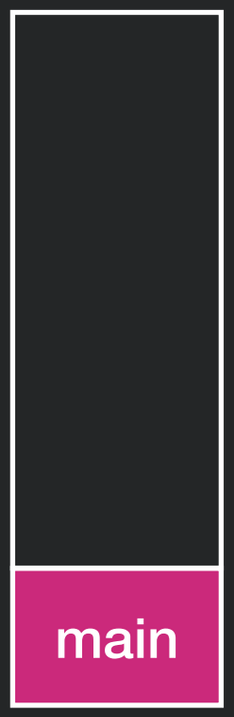
Week 3: Asynchronous Programming

Multi-Threading in Java

- Multi-Threading allows us to do more than one thing at a time
- Physically, through multiple cores and/or OS scheduler
- Example: Process data while interacting with user

Multi-Threading in Java

- Multi-Threading allows us to do more than one thing at a time
- Physically, through multiple cores and/or OS scheduler
- Example: Process data while interacting with user

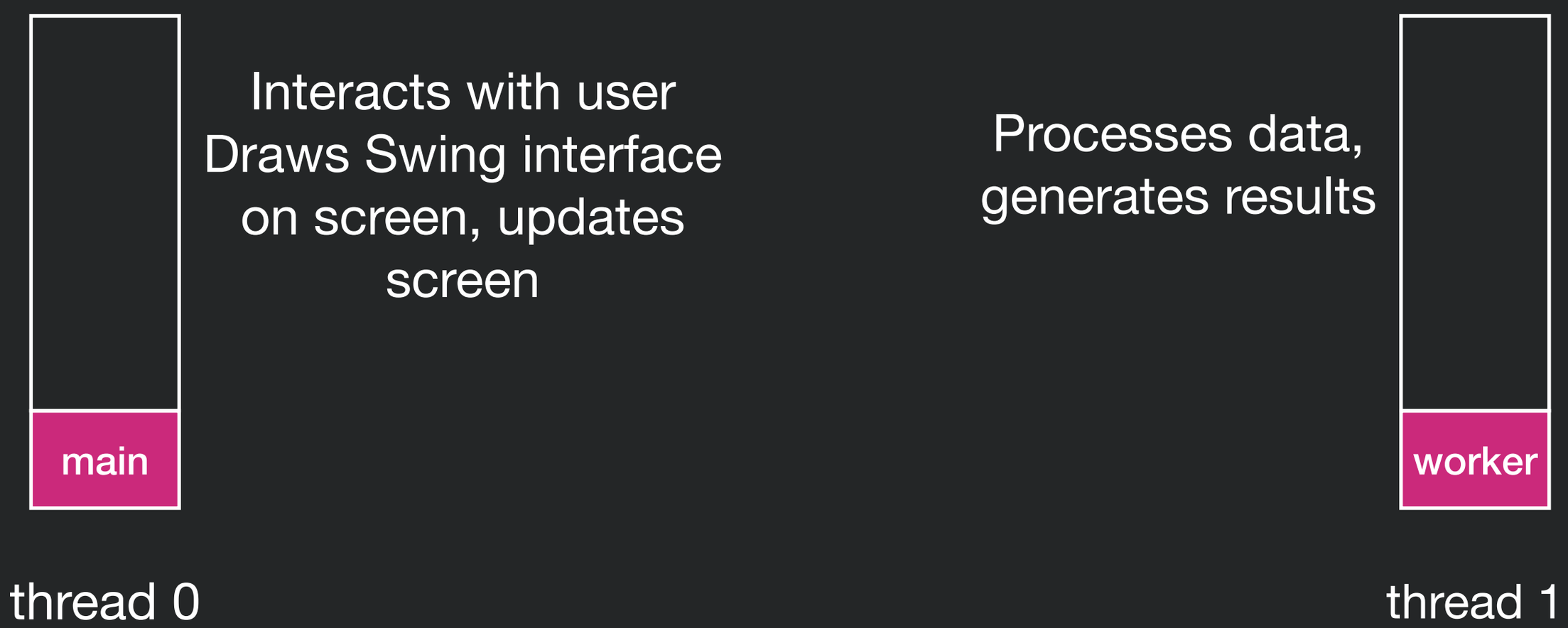


Interacts with user
Draws Swing interface
on screen, updates
screen

thread 0

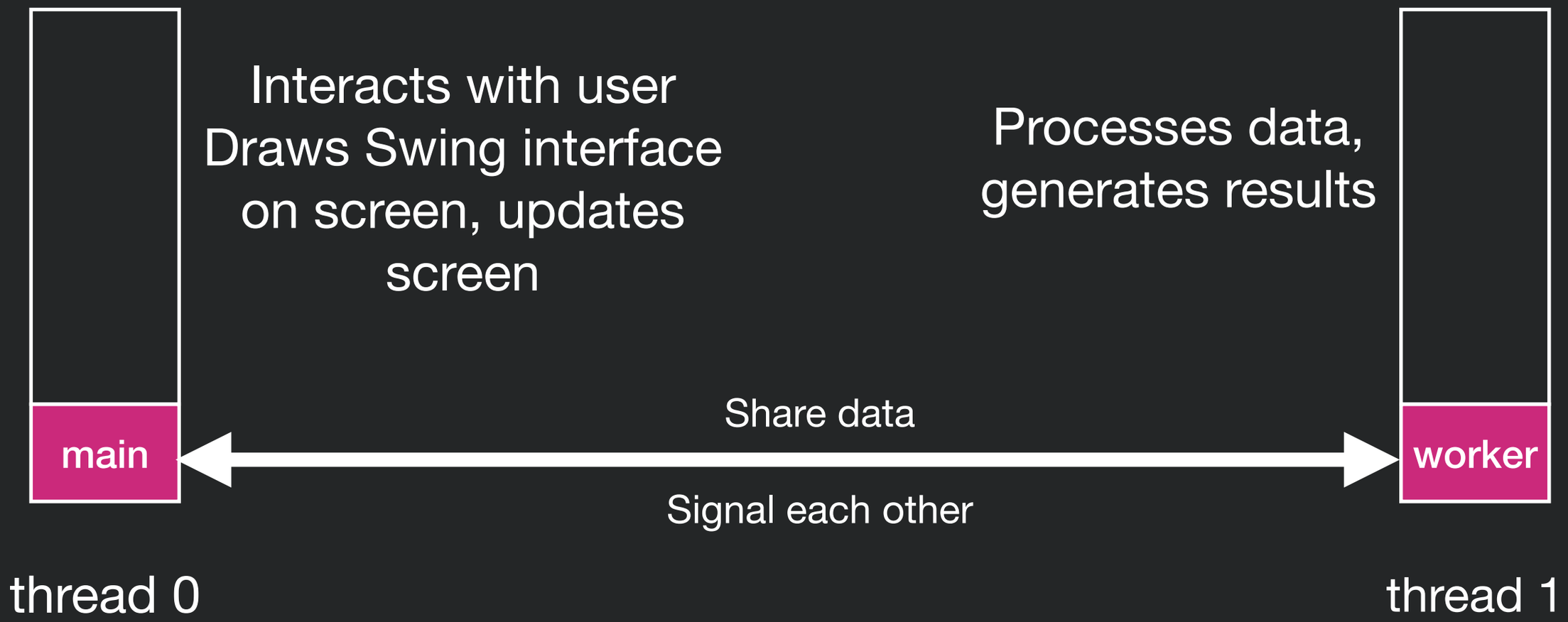
Multi-Threading in Java

- Multi-Threading allows us to do more than one thing at a time
- Physically, through multiple cores and/or OS scheduler
- Example: Process data while interacting with user



Multi-Threading in Java

- Multi-Threading allows us to do more than one thing at a time
- Physically, through multiple cores and/or OS scheduler
- Example: Process data while interacting with user



Woes of Multi-Threading

```
public static int v;
public static void thread1()
{
   v = 4;
   System.out.println(v);
}
```

```
public static void thread2()
{
   v = 2;
}
```

This is a data race: the `println` in thread1 might see either 2 OR 4

Woes of Multi-Threading

```
public static int v;
public static void thread1()
{
   v = 4;
   System.out.println(v);
}
```

```
public static void thread2()
{
   v = 2;
}
```

This is a data race: the `println` in thread1 might see either 2 OR 4

Thread 1	Thread 2

Woes of Multi-Threading

```
public static int v;
public static void thread1()
{
   v = 4;
   System.out.println(v);
}
```

```
public static void thread2()
{
   v = 2;
}
```

This is a data race: the `println` in thread1 might see either 2 OR 4

Thread 1	Thread 2
Write V = 4	

Woes of Multi-Threading

```
public static int v;
public static void thread1()
{
   v = 4;
   System.out.println(v);
}
```

```
public static void thread2()
{
   v = 2;
}
```

This is a data race: the `println` in thread1 might see either 2 OR 4

Thread 1	Thread 2
Write V = 4	
	Write V = 2

Woes of Multi-Threading

```
public static int v;
public static void thread1()
{
   v = 4;
   System.out.println(v);
}
```

```
public static void thread2()
{
   v = 2;
}
```

This is a data race: the `println` in thread1 might see either 2 OR 4

Thread 1	Thread 2
Write V = 4	
	Write V = 2
Read V (2)	

Woes of Multi-Threading

```
public static int v;
public static void thread1()
{
   v = 4;
   System.out.println(v);
}
```

```
public static void thread2()
{
   v = 2;
}
```

This is a data race: the `println` in thread1 might see either 2 OR 4

Thread 1	Thread 2
Write V = 4	
	Write V = 2
Read V (2)	

Thread 1	Thread 2

Woes of Multi-Threading

```
public static int v;
public static void thread1()
{
   v = 4;
   System.out.println(v);
}
```

```
public static void thread2()
{
   v = 2;
}
```

This is a data race: the `println` in thread1 might see either 2 OR 4

Thread 1	Thread 2
Write V = 4	
	Write V = 2
Read V (2)	

Thread 1	Thread 2
	Write V = 2

Woes of Multi-Threading

```
public static int v;
public static void thread1()
{
   v = 4;
   System.out.println(v);
}
```

```
public static void thread2()
{
   v = 2;
}
```

This is a data race: the `println` in thread1 might see either 2 OR 4

Thread 1	Thread 2
Write V = 4	
	Write V = 2
Read V (2)	

Thread 1	Thread 2
	Write V = 2
Write V = 4	

Woes of Multi-Threading

```
public static int v;
public static void thread1()
{
   v = 4;
   System.out.println(v);
}
```

```
public static void thread2()
{
   v = 2;
}
```

This is a data race: the `println` in thread1 might see either 2 OR 4

Thread 1	Thread 2
Write V = 4	
	Write V = 2
Read V (2)	

Thread 1	Thread 2
	Write V = 2
Write V = 4	
Read V (4)	

Multi-Threading in JS

```
var request = require('request');
request('http://www.google.com', function (error, response,
body) {
    console.log("Heard back from Google!");
});
console.log("Made request");
```

Request is an ***asynchronous call***

Multi-Threading in JS

```
var request = require('request');
request('http://www.google.com', function (error, response,
body) {
    console.log("Heard back from Google!");
});
console.log("Made request");
```

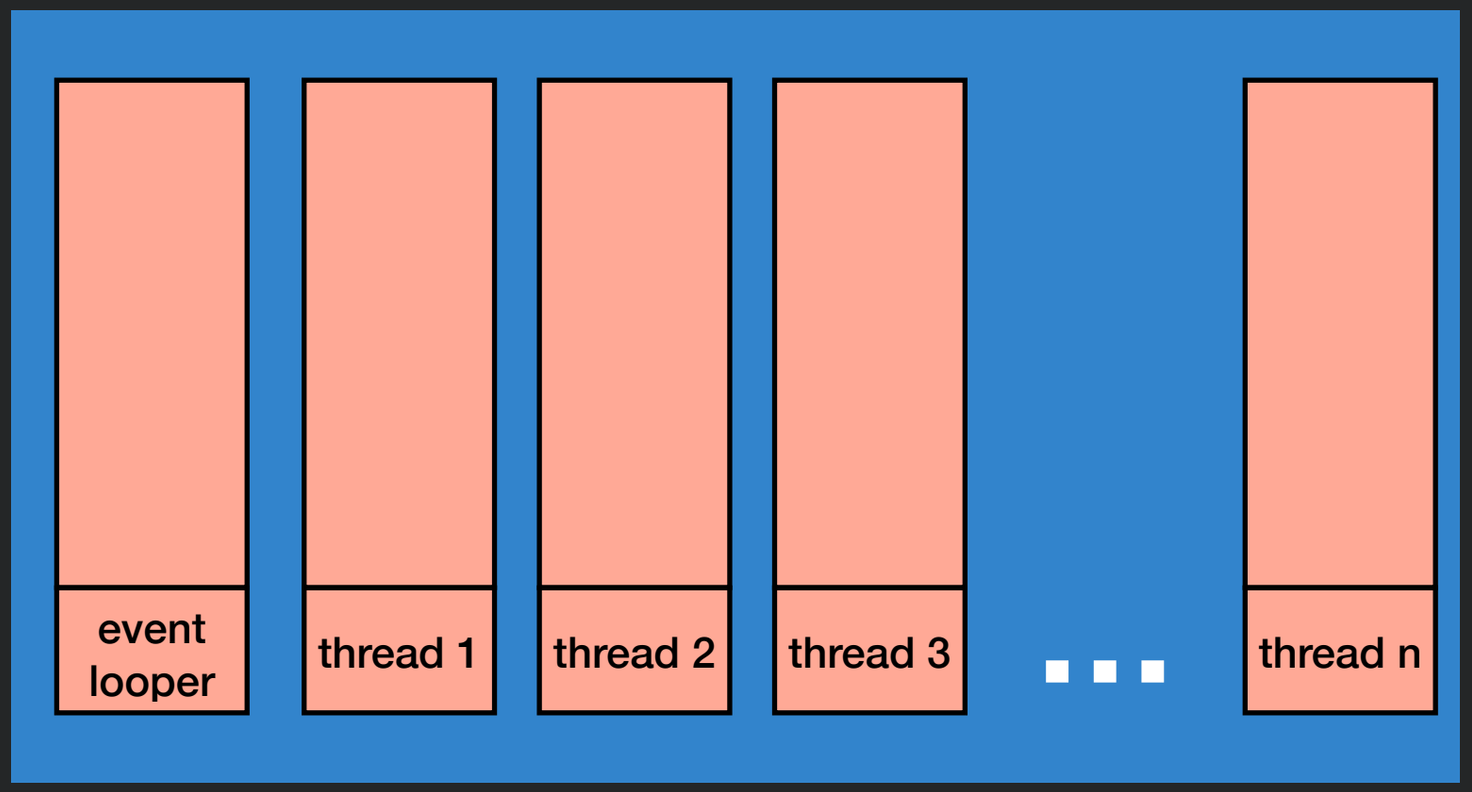
Output:

Made request
Heard back from Google!

Request is an ***asynchronous call***

Multi-Threading in JS

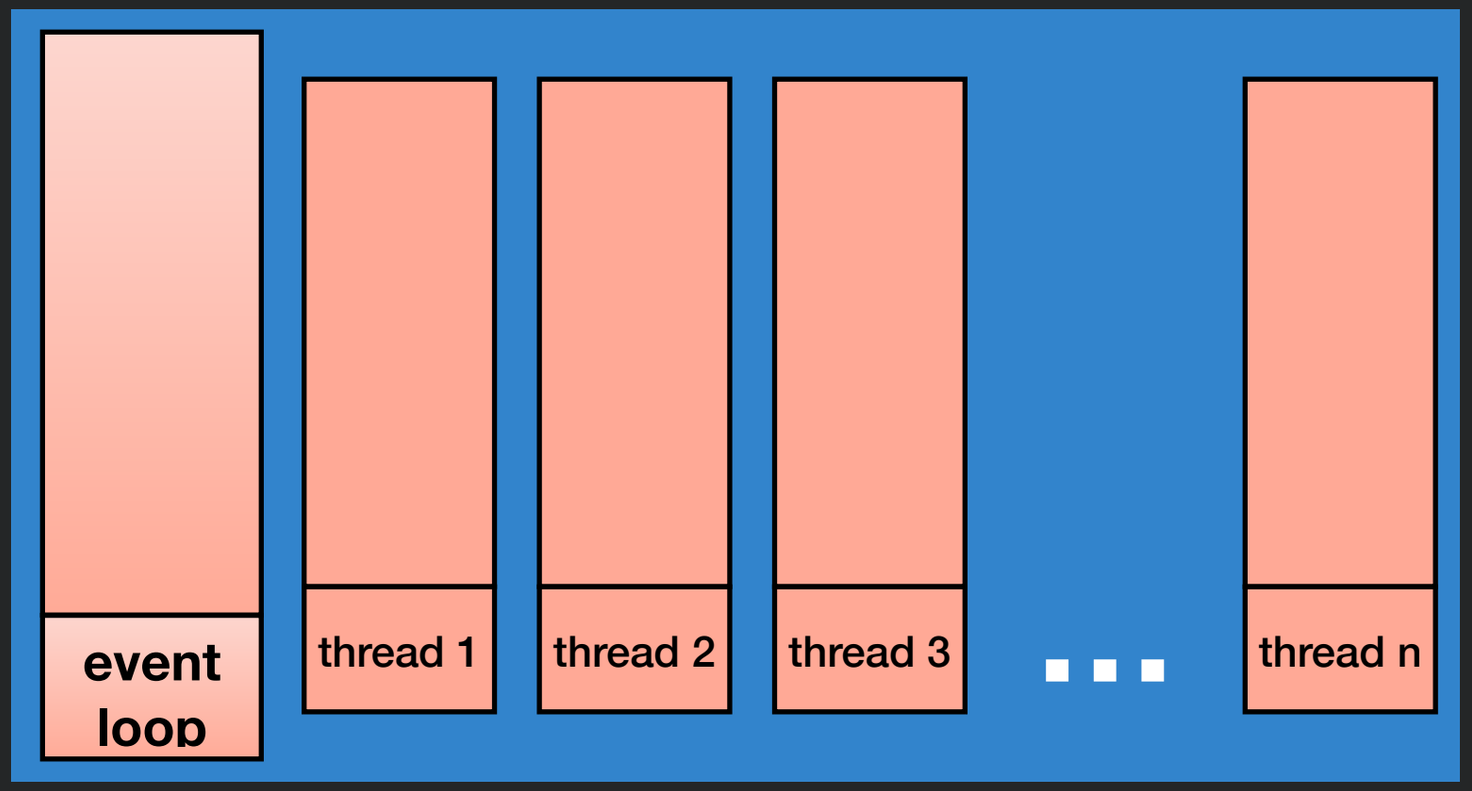
- Everything you write will run in a single thread* (event loop)
- Since you are not sharing data between threads, races don't happen as easily
- Inside of JS engine: many threads
- Event loop processes events, and calls your callbacks



JS Engine

Multi-Threading in JS

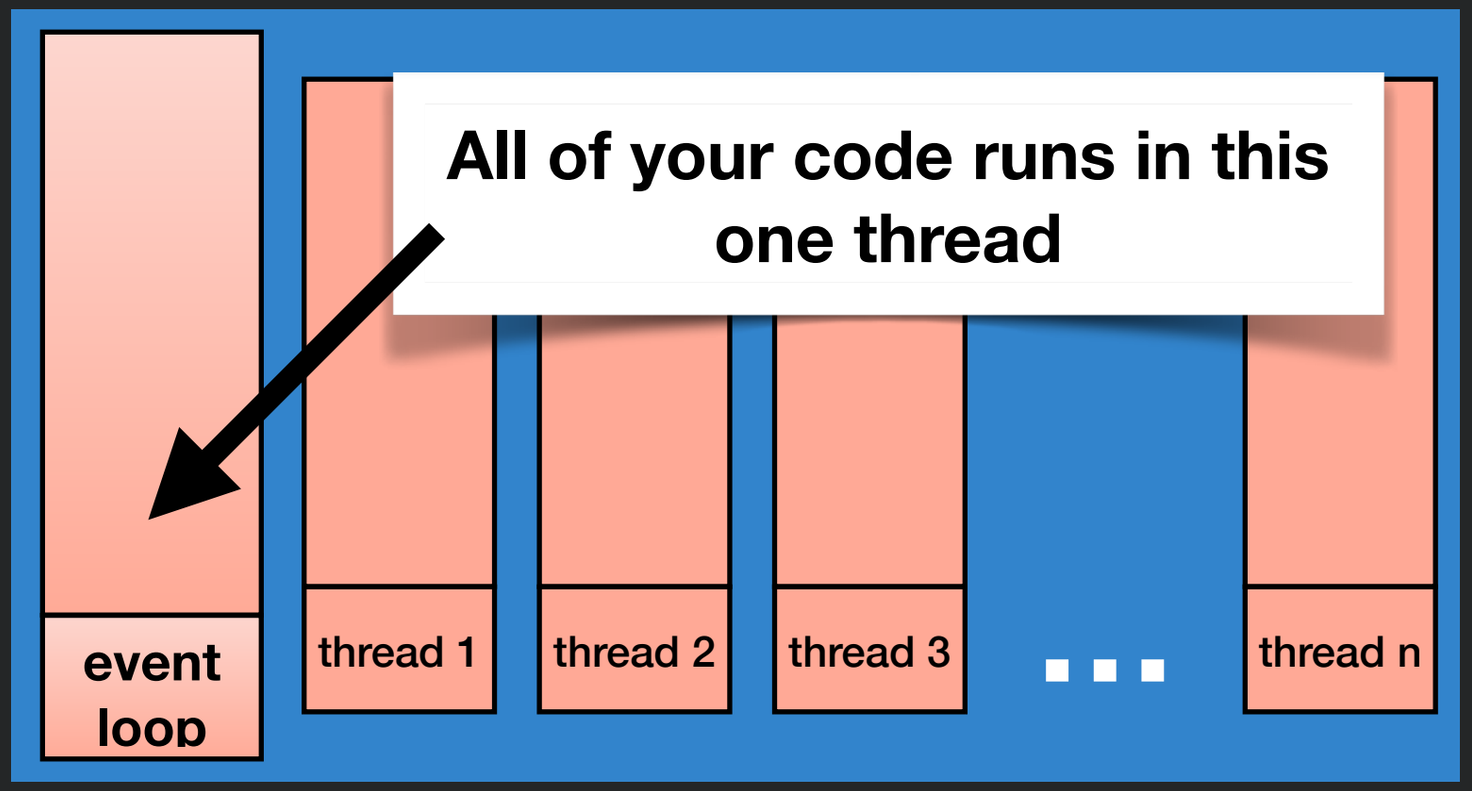
- Everything you write will run in a single thread* (event loop)
- Since you are not sharing data between threads, races don't happen as easily
- Inside of JS engine: many threads
- Event loop processes events, and calls your callbacks



JS Engine

Multi-Threading in JS

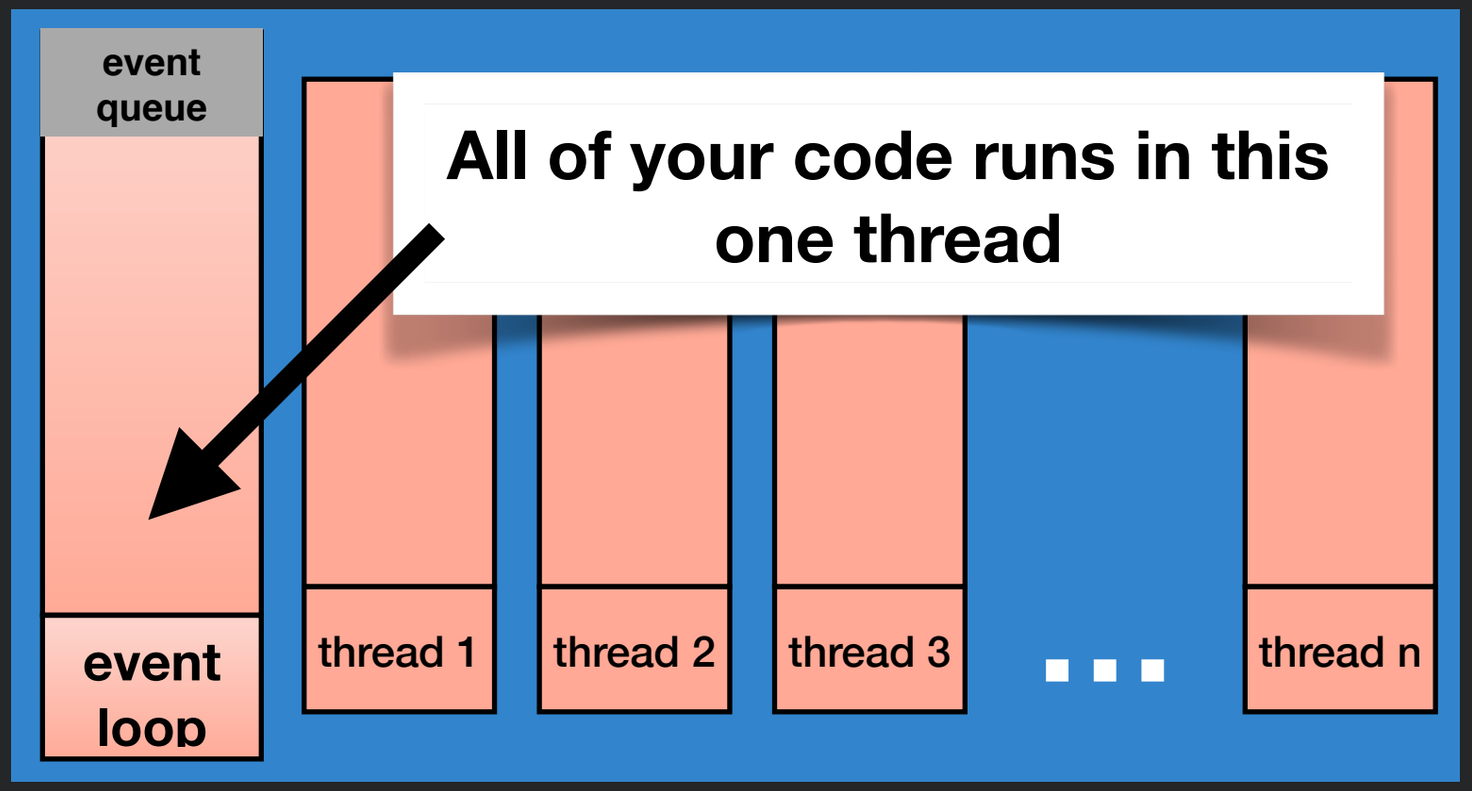
- Everything you write will run in a single thread* (event loop)
- Since you are not sharing data between threads, races don't happen as easily
- Inside of JS engine: many threads
- Event loop processes events, and calls your callbacks



JS Engine

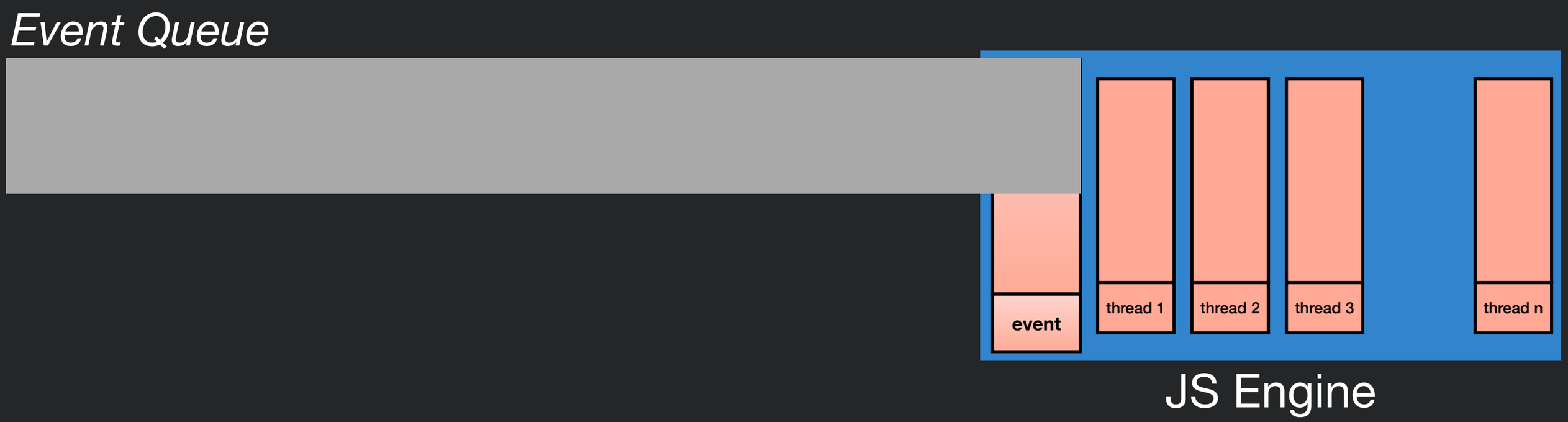
Multi-Threading in JS

- Everything you write will run in a single thread* (event loop)
- Since you are not sharing data between threads, races don't happen as easily
- Inside of JS engine: many threads
- Event loop processes events, and calls your callbacks

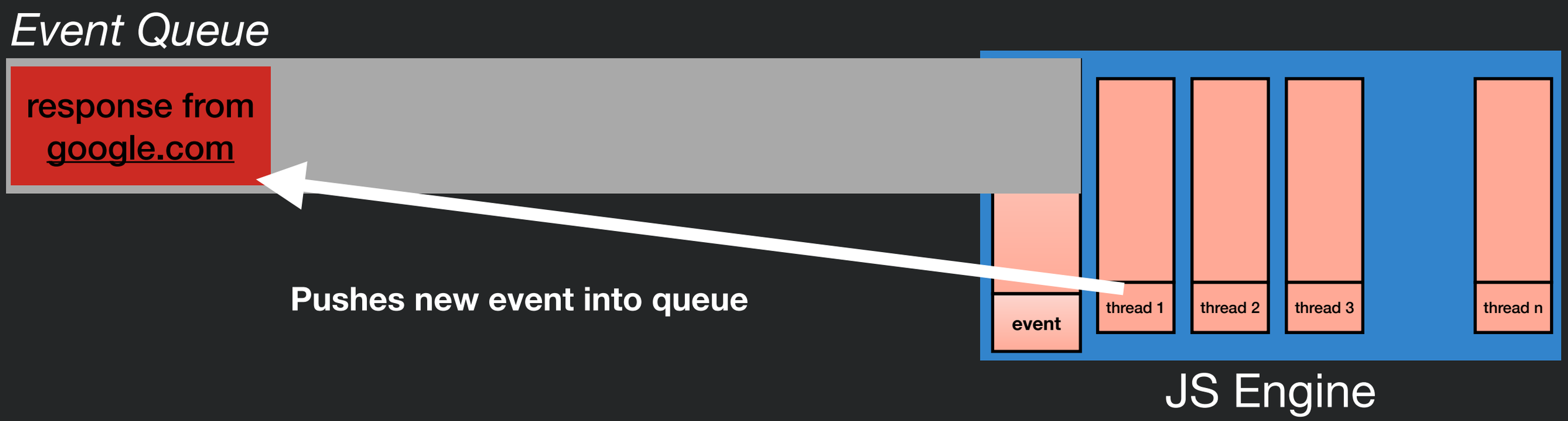


JS Engine

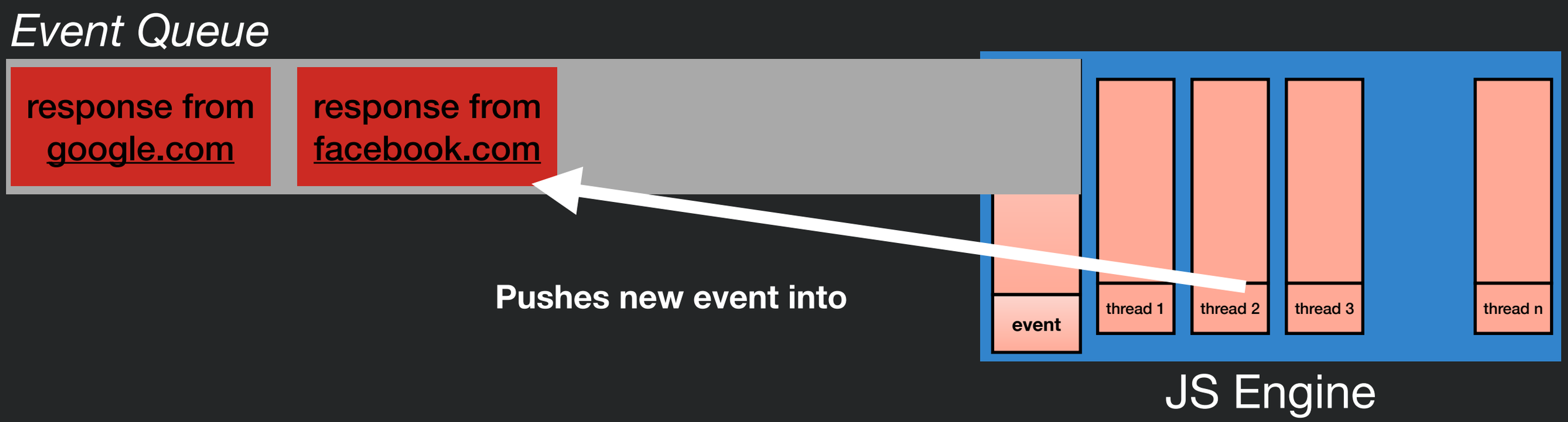
The Event Loop



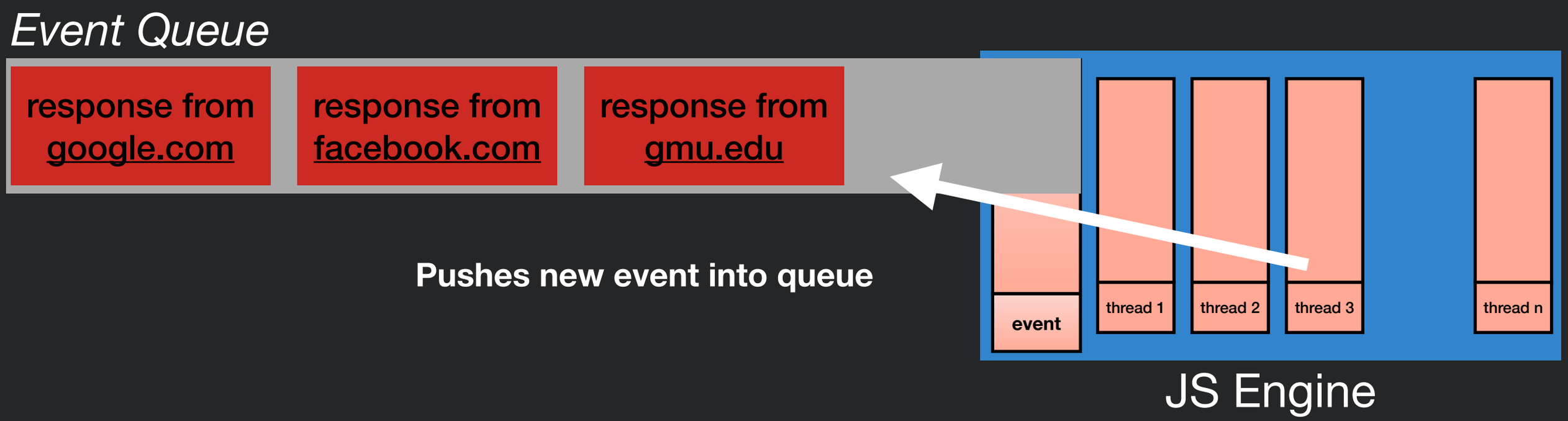
The Event Loop



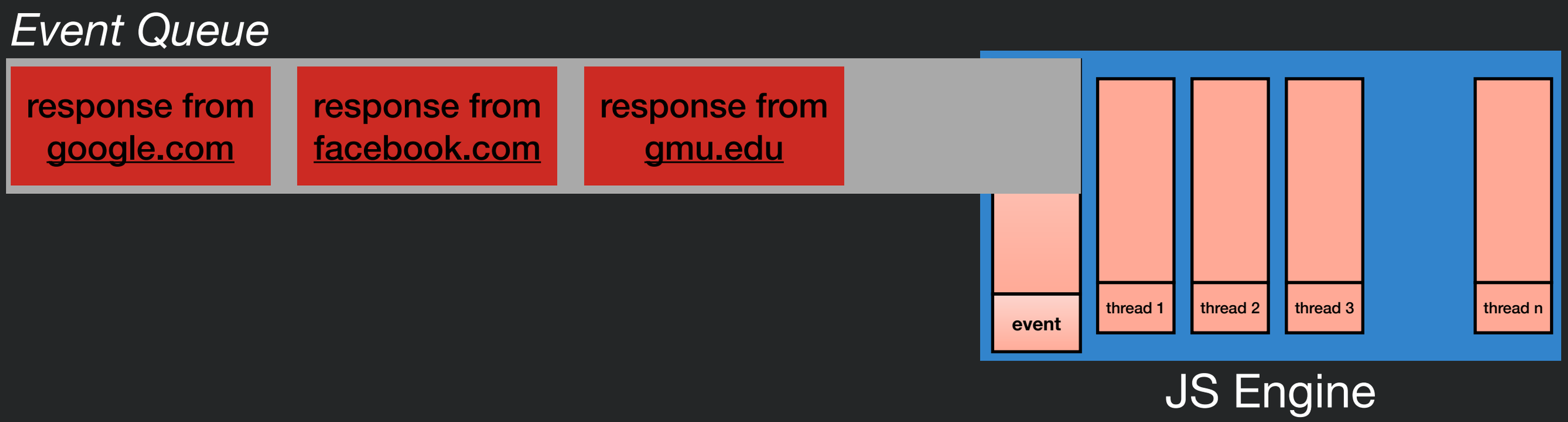
The Event Loop



The Event Loop



The Event Loop

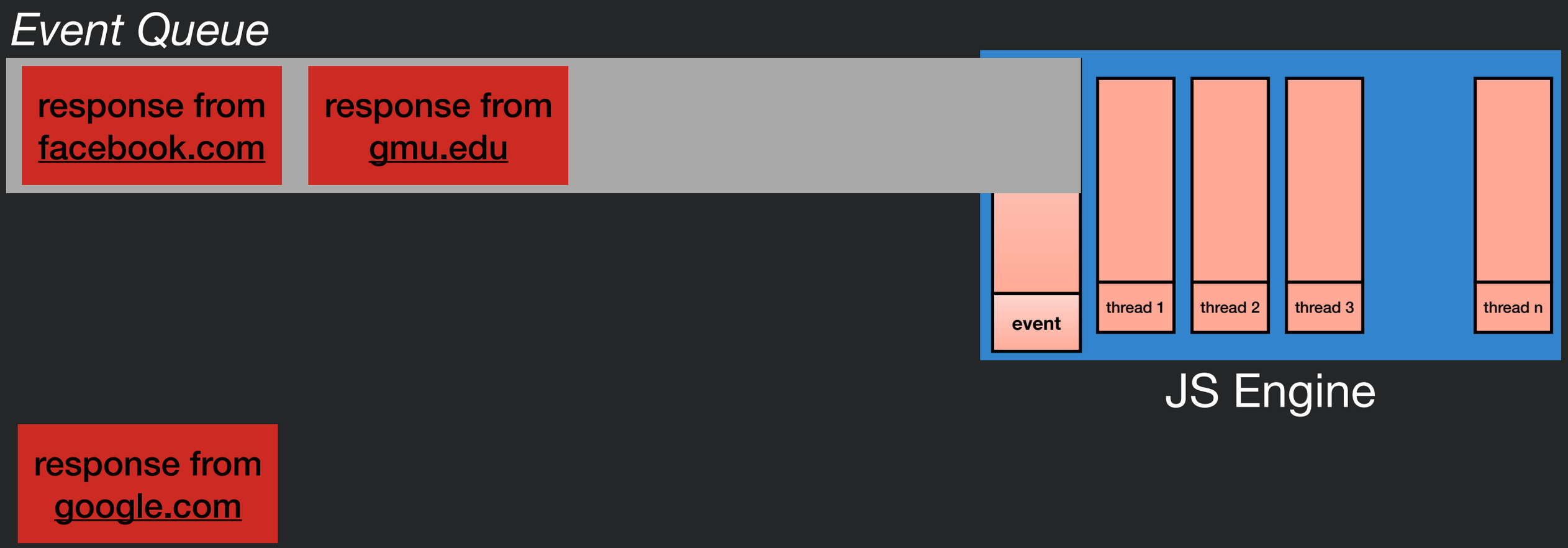


The Event Loop

Event Queue

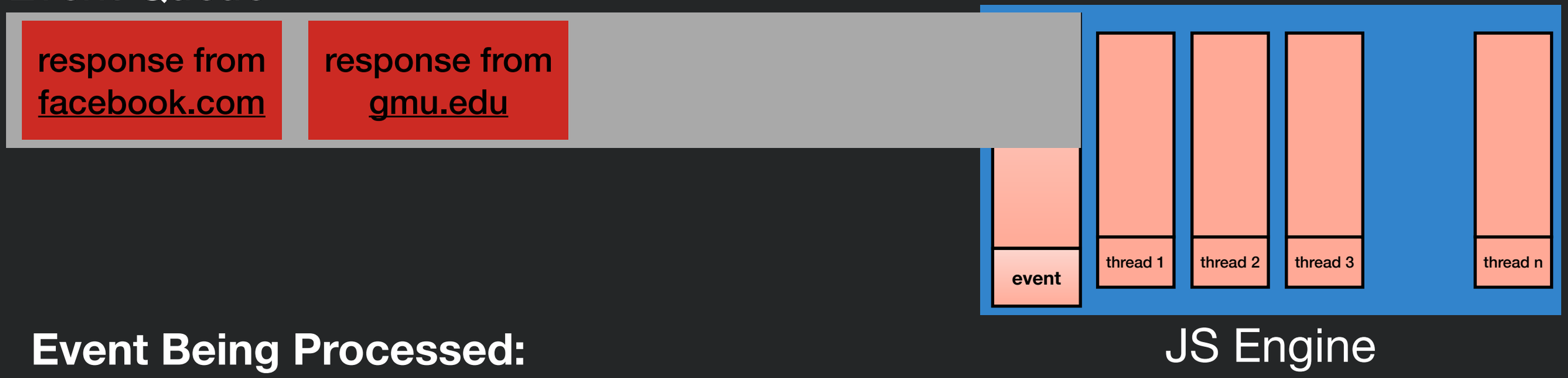
Event Being Processed:

The Event Loop



The Event Loop

Event Queue

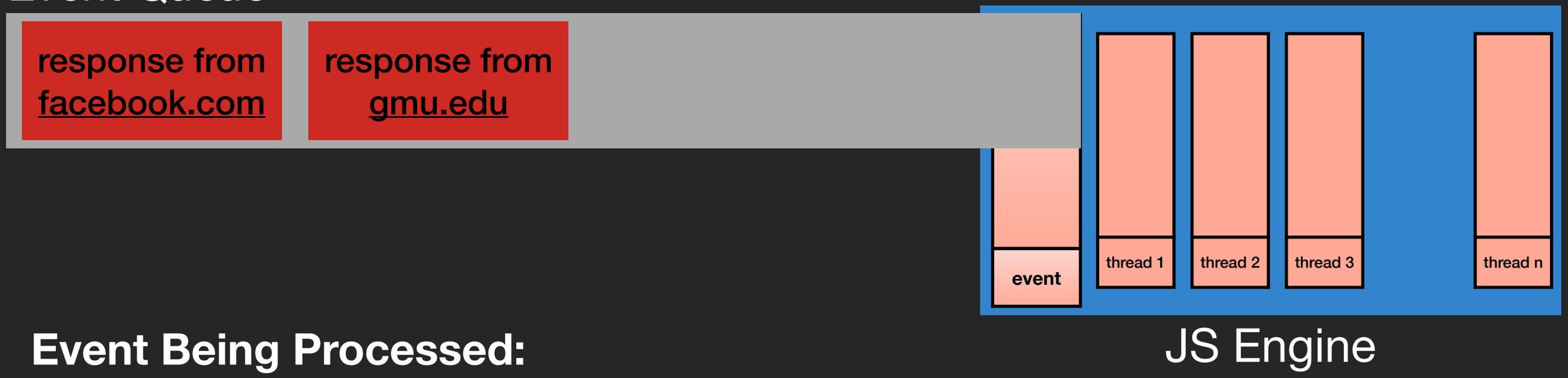


Event Being Processed:

response from google.com

The Event Loop

Event Queue



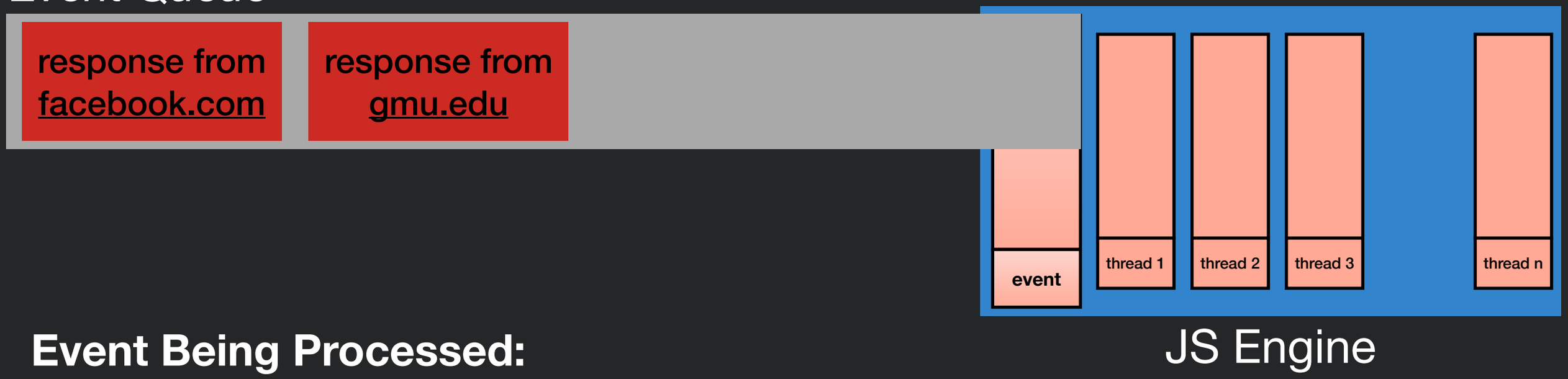
Event Being Processed:

response from google.com

Are there any listeners registered for this event?

The Event Loop

Event Queue



Event Being Processed:

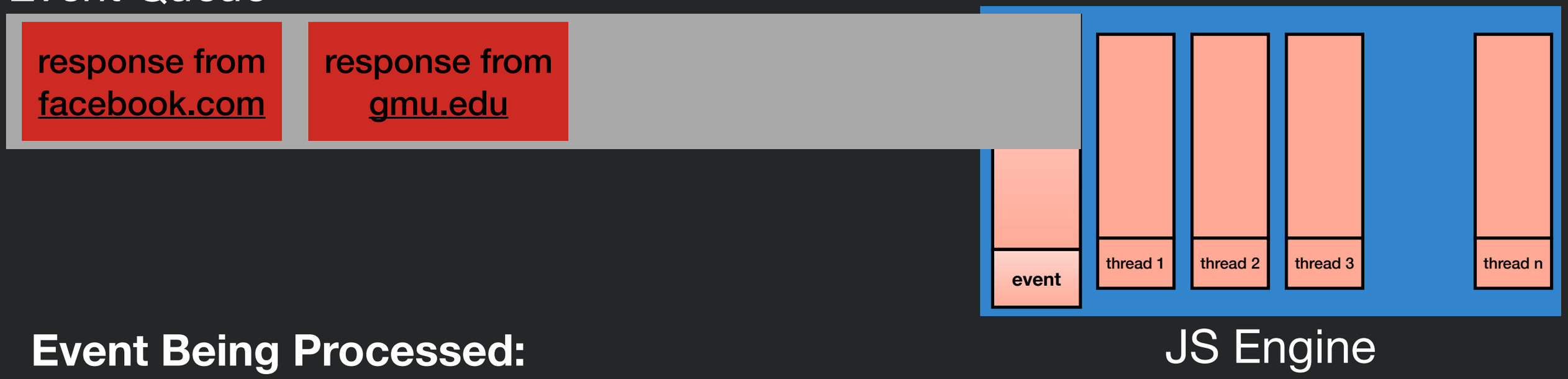
response from google.com

Are there any listeners registered for this event?

If so, call listener with event

The Event Loop

Event Queue



Event Being Processed:

response from google.com

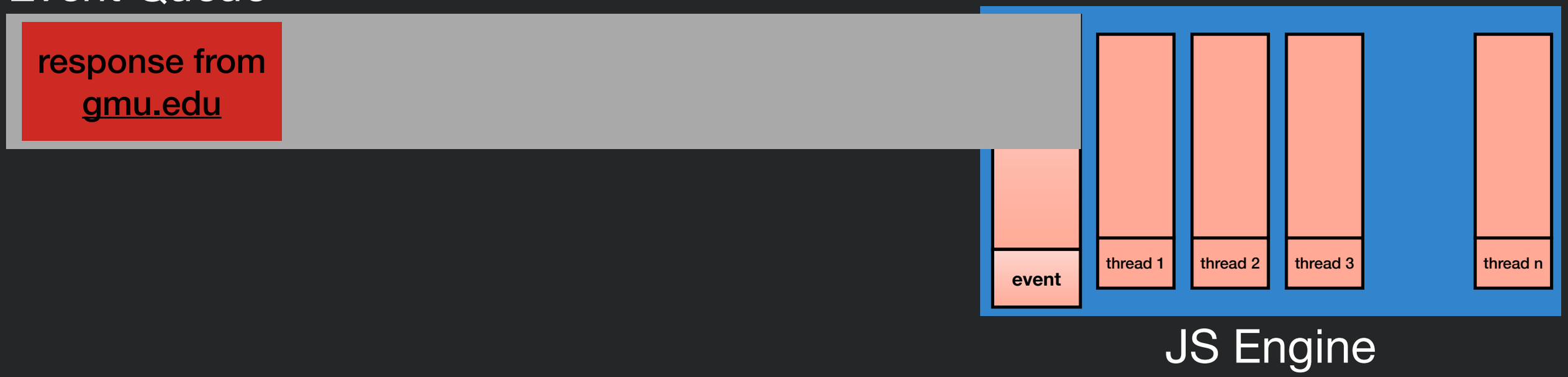
Are there any listeners registered for this event?

If so, call listener with event

After the listener is finished, repeat

The Event Loop

Event Queue

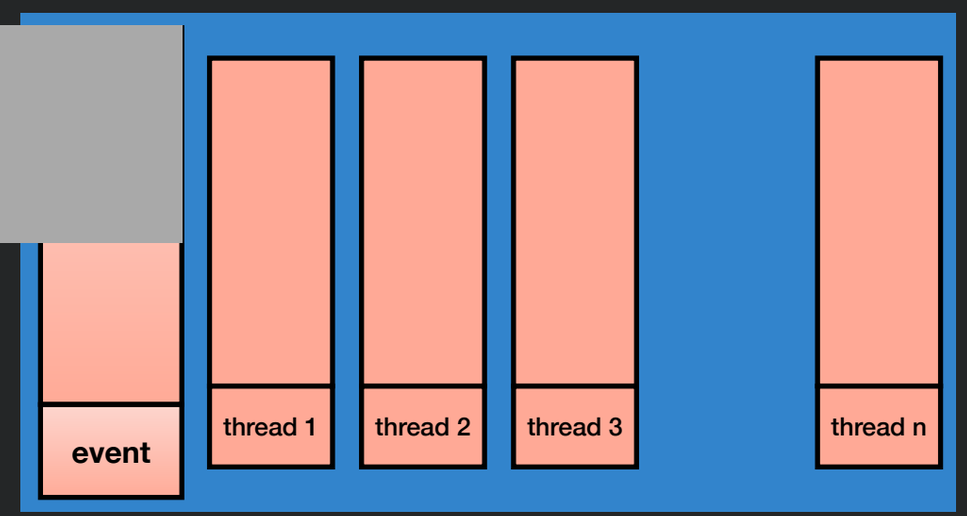


response from
facebook.com

The Event Loop

Event Queue

response from
gmu.edu



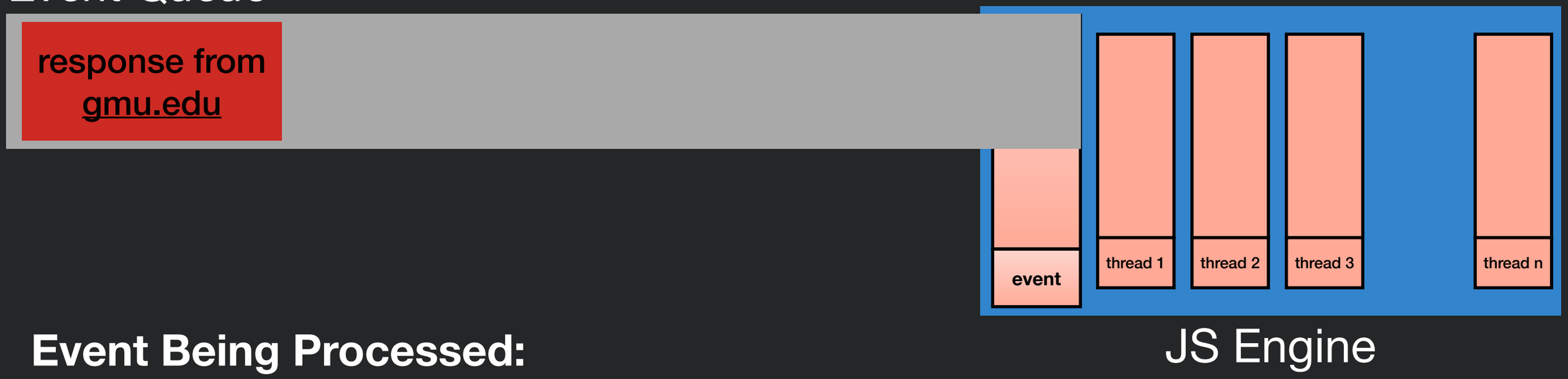
JS Engine

Event Being Processed:

response from
facebook.com

The Event Loop

Event Queue



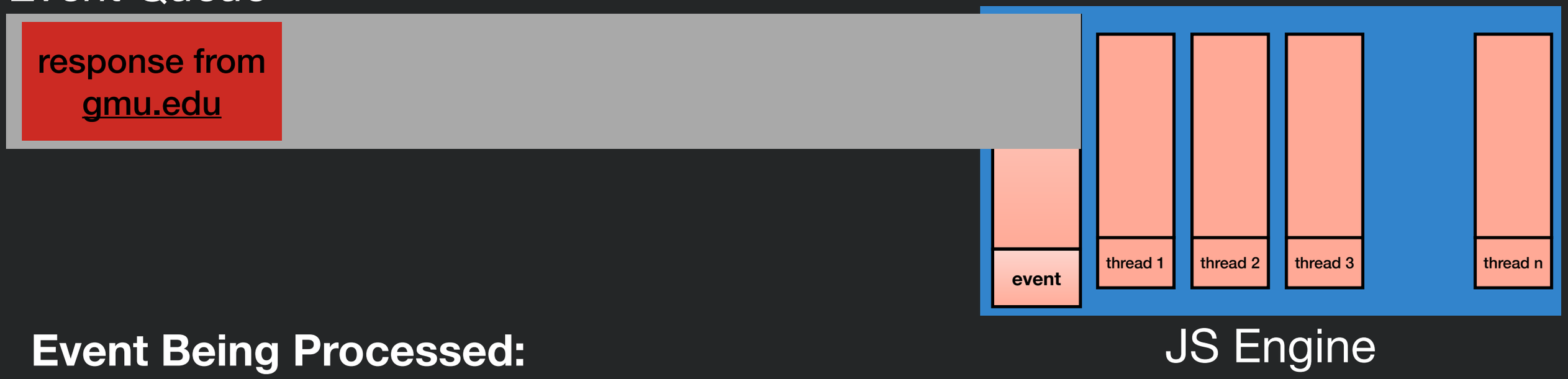
Event Being Processed:

response from
facebook.com

Are there any listeners registered for this event?

The Event Loop

Event Queue



Event Being Processed:

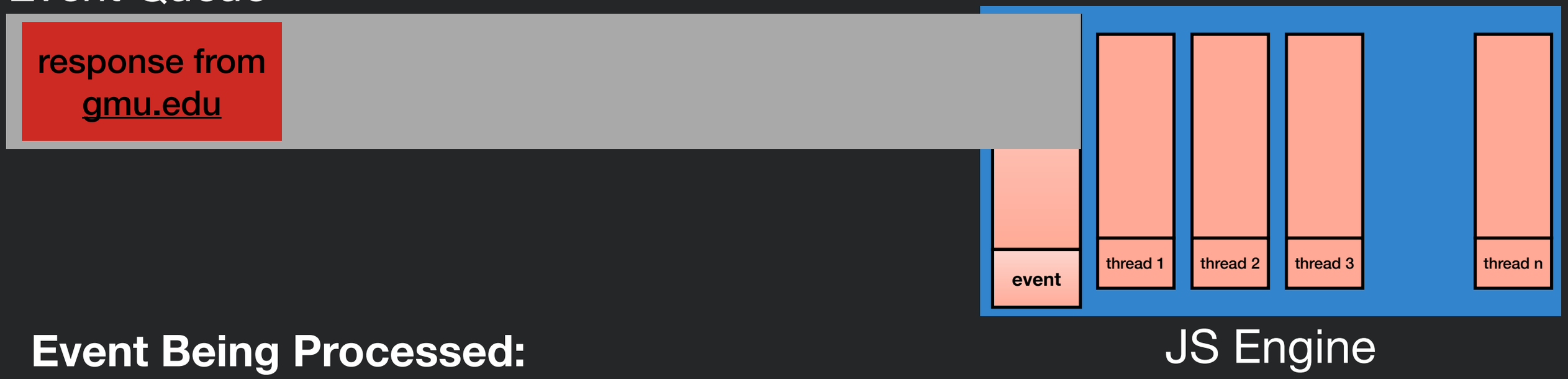
response from
facebook.com

Are there any listeners registered for this event?

If so, call listener with event

The Event Loop

Event Queue



Event Being Processed:

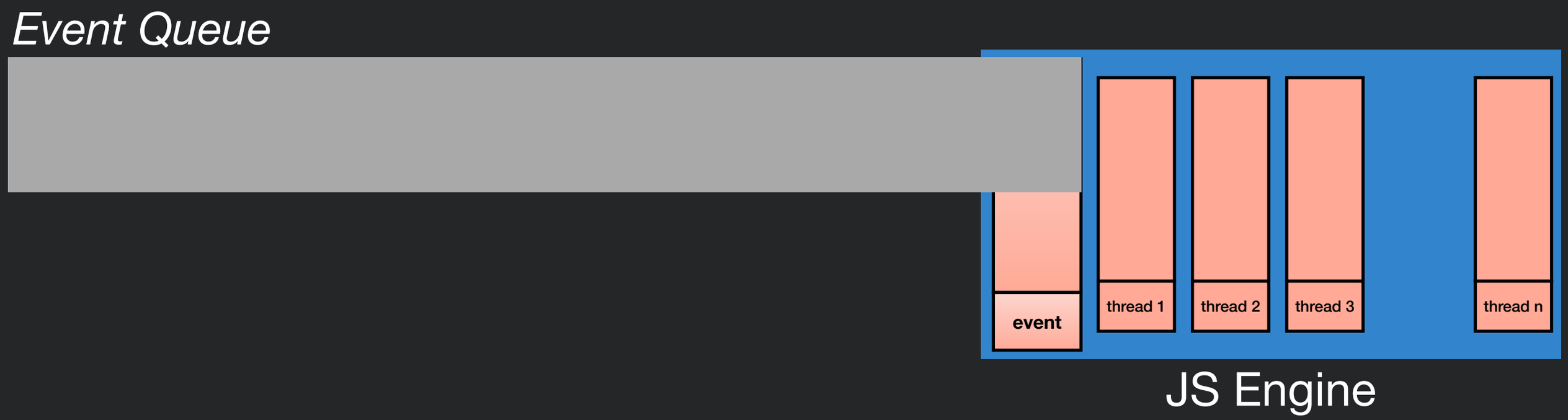
response from
facebook.com

Are there any listeners registered for this event?

If so, call listener with event

After the listener is finished, repeat

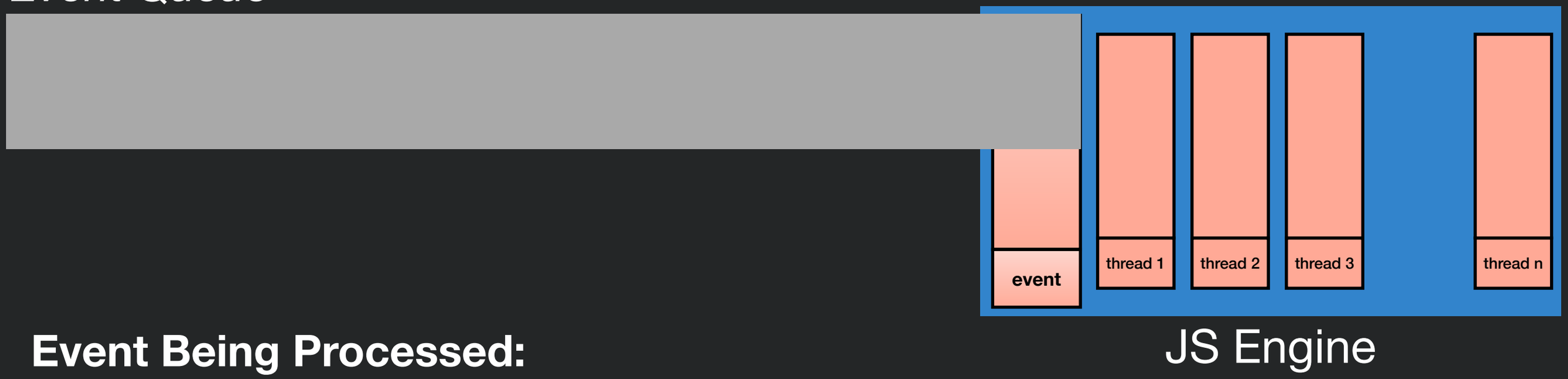
The Event Loop



response from
gmu.edu

The Event Loop

Event Queue

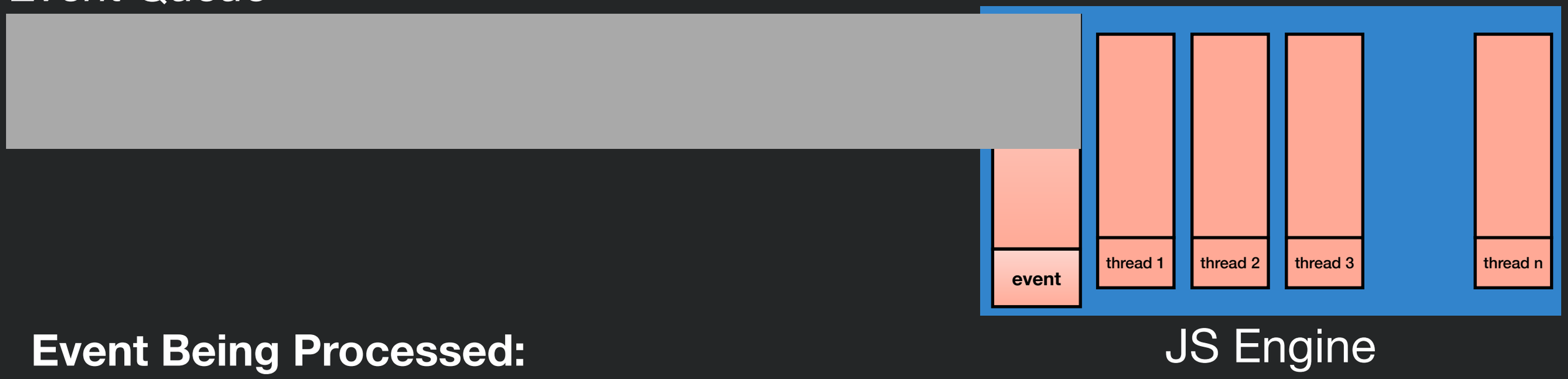


Event Being Processed:

response from gmu.edu

The Event Loop

Event Queue



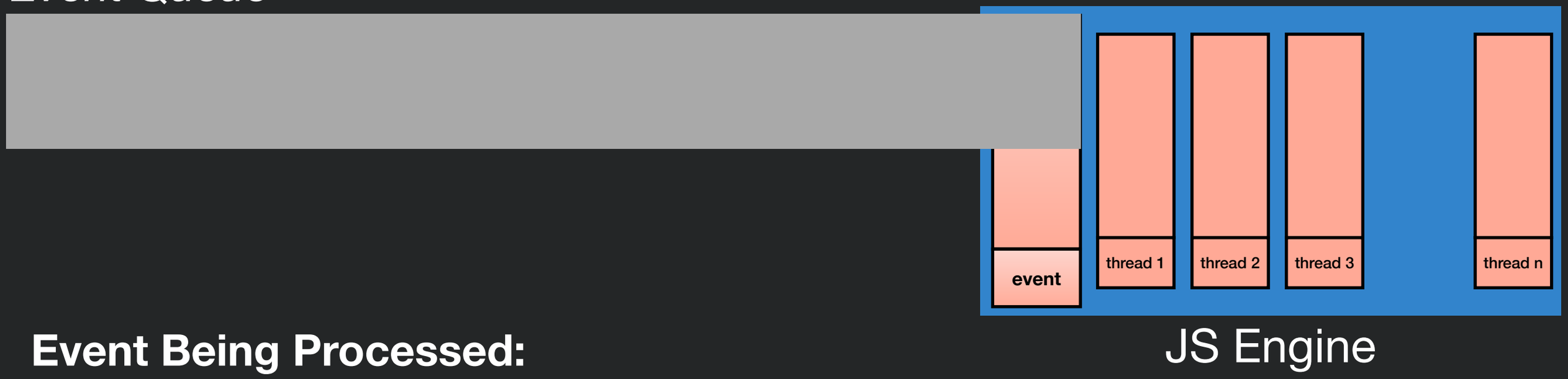
Event Being Processed:

response from gmu.edu

Are there any listeners registered for this event?

The Event Loop

Event Queue



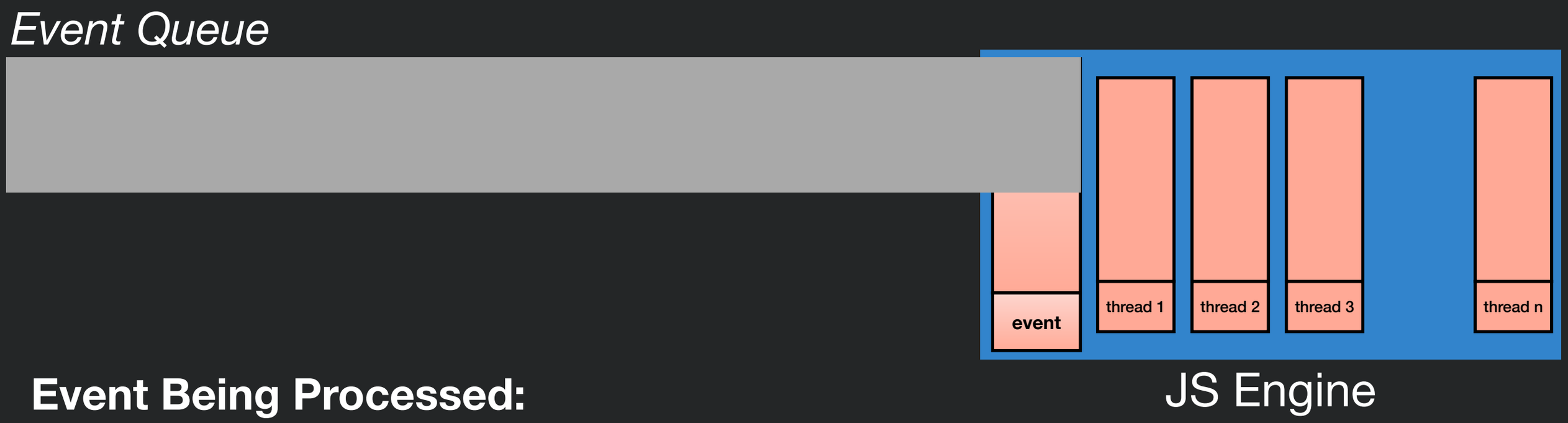
Event Being Processed:

response from gmu.edu

Are there any listeners registered for this event?

If so, call listener with event

The Event Loop



Event Being Processed:

response from gmu.edu

Are there any listeners registered for this event?

If so, call listener with event

After the listener is finished, repeat

The Event Loop

- Remember that JS is **event-driven**

```
var request = require('request');
request('http://www.google.com', function (error, response, body) {
    console.log("Heard back from Google!");
});
console.log("Made request");
```

- Event loop is responsible for dispatching events when they occur

- Main thread for event loop:

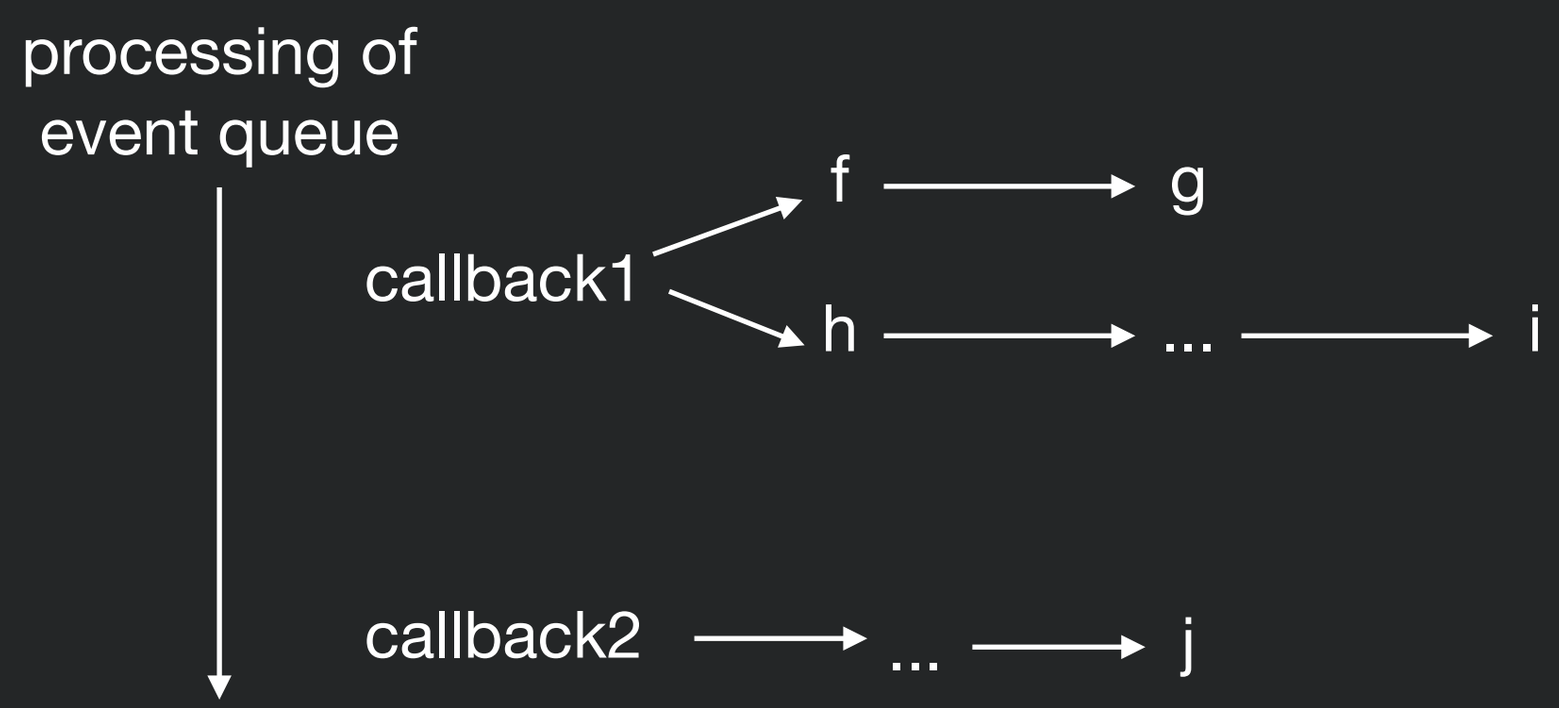
```
while(queue.waitForMessage()){
  queue.processNextMessage();
}
```

Benefits vs. Explicit Threading (Java)

- Writing your own threads is ***difficult*** to reason about and get right:
  - When threads share data, need to ensure they correctly ***synchronize*** on it to avoid race conditions
- Main downside to events:
  - Can not have slow event handlers
  - Can still have races, although easier to reason about

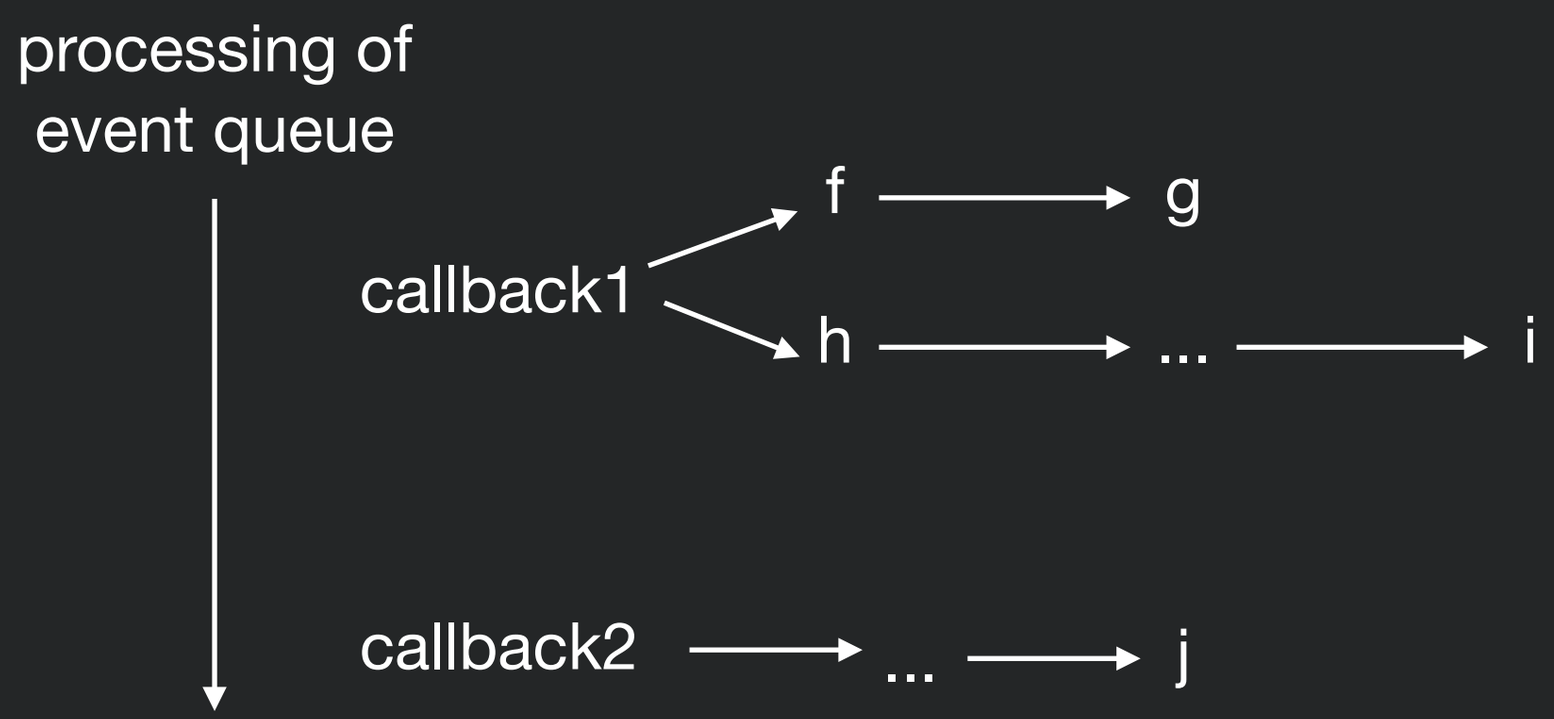
Run-to-Completion Semantics

- Run-to-completion
  - The function handling an event and the functions that it (transitively) synchronously calls will keep executing until the function finishes.
  - The JS engine will not handle the next event until the event handler finishes.



Implications of Run-to-Completion

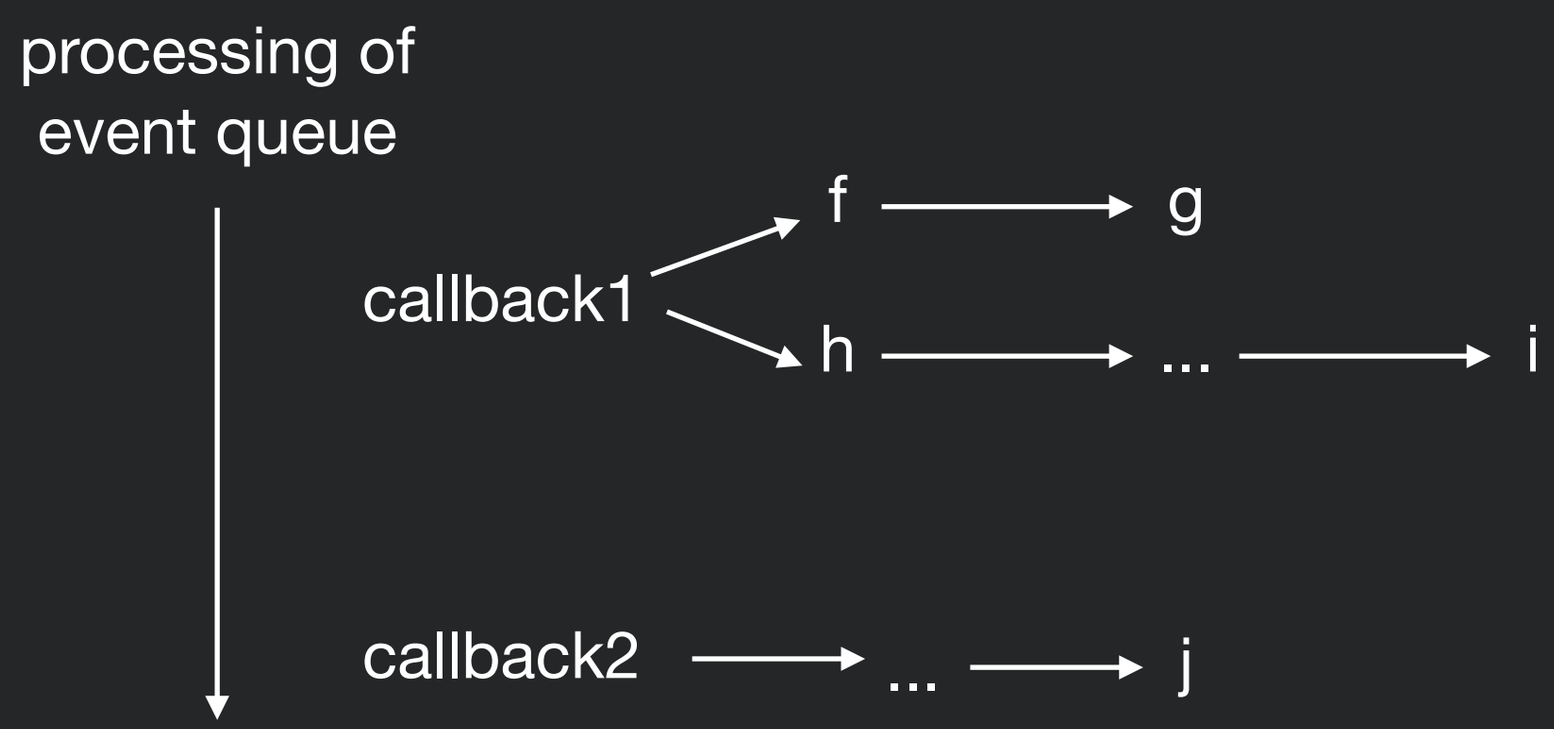
- Good news: no other code will run until you finish (no worries about other threads overwriting your data)



`j` will not execute until after `i`

Implications of Run-to-Completion

- Bad/OK news: Nothing else will happen until event handler returns
  - Event handlers should never block (e.g., wait for input) --> all callbacks waiting for network response or user input are **always** asynchronous
  - Event handlers shouldn't take a long time either



`j` will not execute until `i` finishes

Decomposing a long-running computation

- If you ***must*** do something that takes a long time (e.g. computation), split it into multiple events
  - `doSomeWork();`
  - ... [let event loop process other events]..
  - `continueDoingMoreWork();`
  - ...

Dangers of Decomposition

- Application state may ***change*** before event occurs
  - Other event handlers may be interleaved and occur before event occurs and mutate the same application state
  - --> Need to check that update still makes sense
- Application state may be in ***inconsistent*** state until event occurs
- leaving data in inconsistent state...
- Loading some data from API, but not all of it...

Sequencing events with Promises

- Promises are a ***wrapper*** around async callbacks
- Promises represents ***how*** to get a value
- Then you tell the promise what to do ***when*** it gets it
- Promises organize many steps that need to happen in order, with each step happening asynchronously
- At any point a promise is either:
  - Unresolved
  - Succeeds
  - Fails

Using a Promise

- Declare what you want to do when your promise is completed (`then`), or if there's an error (`catch`)

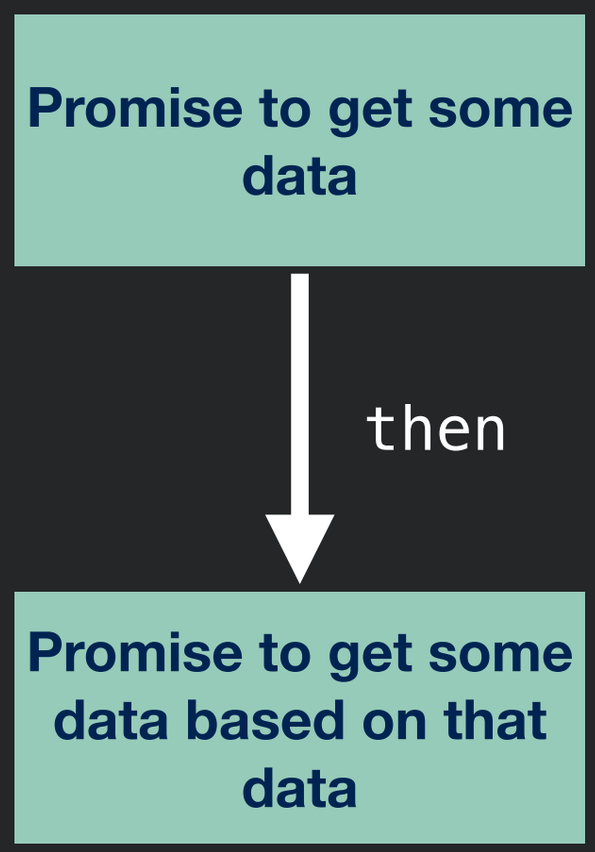
```
fetch('https://github.com/')
    .then(function(res) {
        return res.text();
    });
```

```
fetch('http://domain.invalid/')
    .catch(function(err) {
        console.log(err);
    });
```

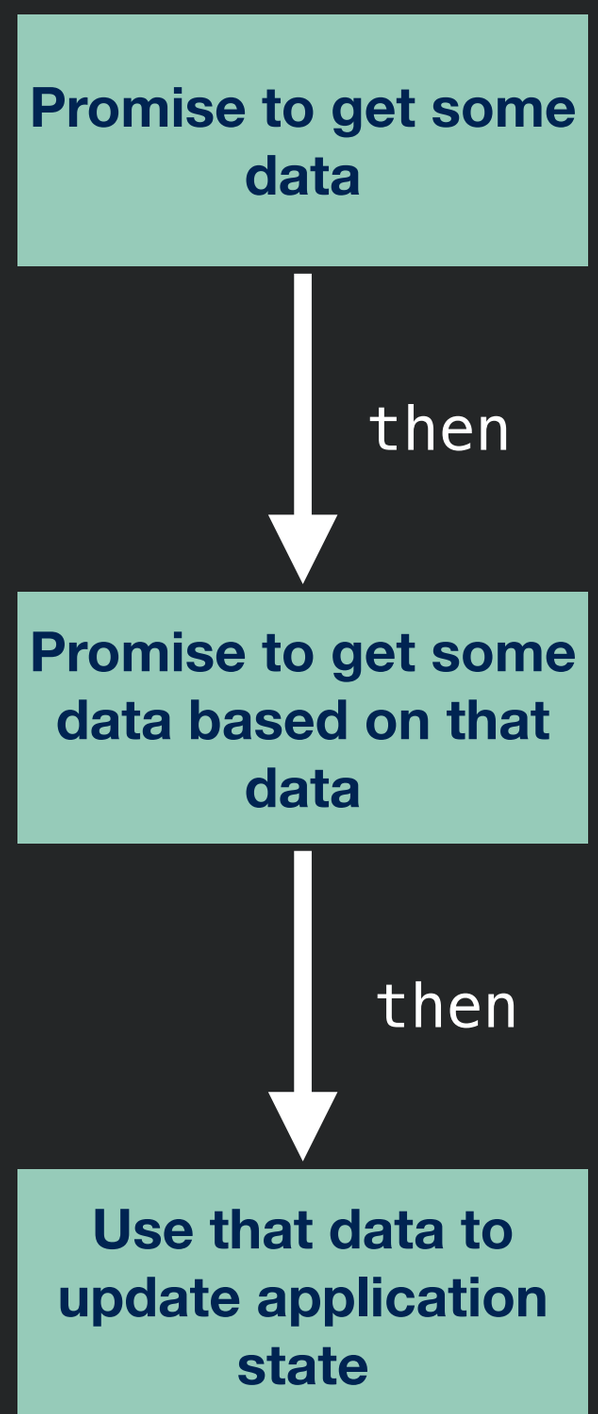
Promise One Thing Then Another

Promise to get some data

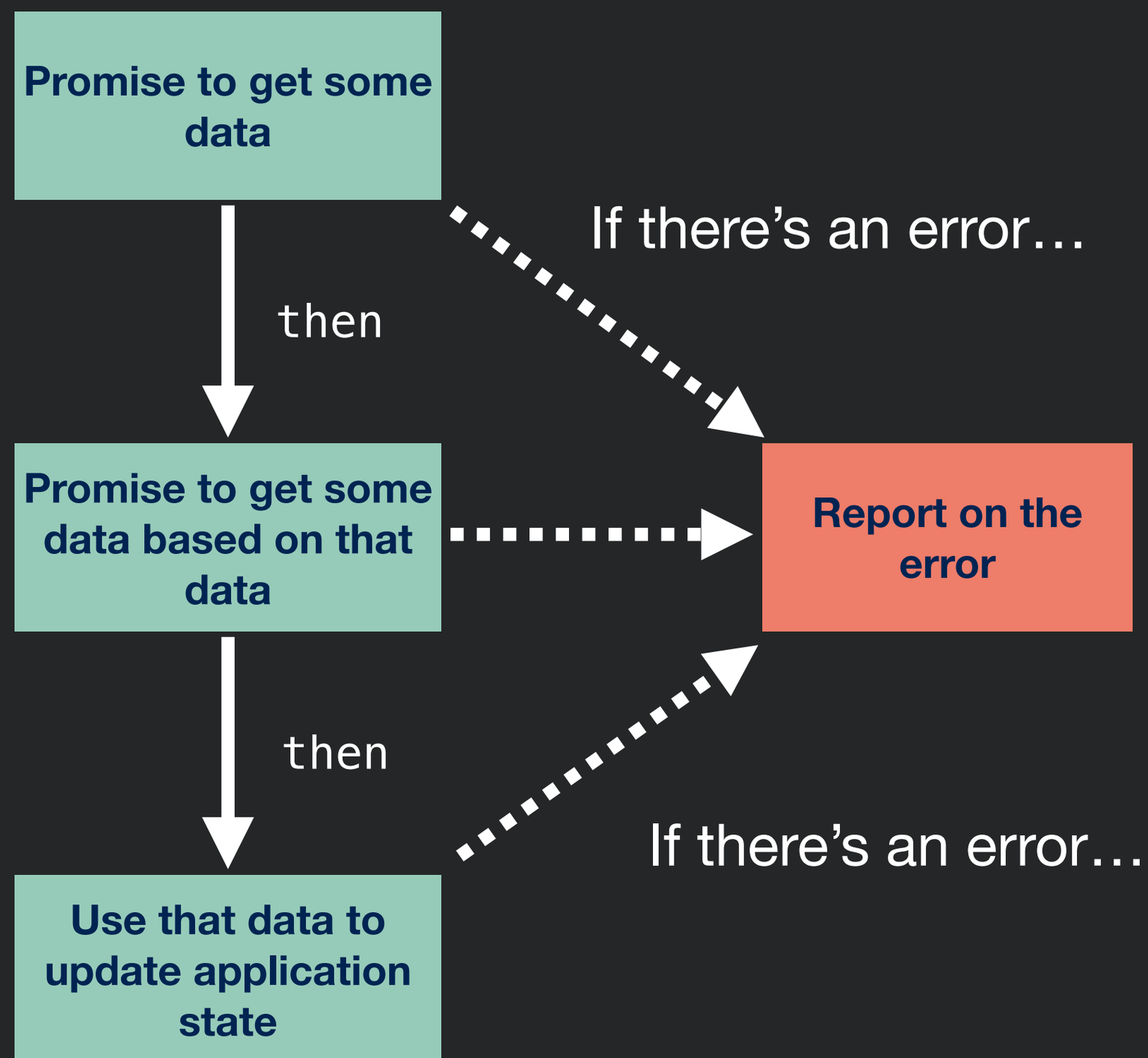
Promise One Thing Then Another



Promise One Thing Then Another



Promise One Thing Then Another



Chaining Promises

Chaining Promises

```
myPromise.then(function(resultOfPromise){
    //Do something, maybe asynchronously
    return theResultOfThisStep;
})
```

Chaining Promises

```
myPromise.then(function(resultOfPromise){
    //Do something, maybe asynchronously
    return theResultOfThisStep;
})
.then(function(resultOfStep1){
    //Do something, maybe asynchronously
    return theResultOfStep2;
})
```

Chaining Promises

```
myPromise.then(function(resultOfPromise){
    //Do something, maybe asynchronously
    return theResultOfThisStep;
})
.then(function(resultOfStep1){
    //Do something, maybe asynchronously
    return theResultOfStep2;
})
.then(function(resultOfStep2){
    //Do something, maybe asynchronously
    return theResultOfStep3;
})
```

Chaining Promises

```
myPromise.then(function(resultOfPromise){
    //Do something, maybe asynchronously
    return theResultOfThisStep;
})
.then(function(resultOfStep1){
    //Do something, maybe asynchronously
    return theResultOfStep2;
})
.then(function(resultOfStep2){
    //Do something, maybe asynchronously
    return theResultOfStep3;
})
.then(function(resultOfStep3){
    //Do something, maybe asynchronously
    return theResultOfStep4;
})
```

Chaining Promises

```
myPromise.then(function(resultOfPromise){
    //Do something, maybe asynchronously
    return theResultOfThisStep;
})
.then(function(resultOfStep1){
    //Do something, maybe asynchronously
    return theResultOfStep2;
})
.then(function(resultOfStep2){
    //Do something, maybe asynchronously
    return theResultOfStep3;
})
.then(function(resultOfStep3){
    //Do something, maybe asynchronously
    return theResultOfStep4;
})
.catch(function(error){

});
```

Writing a Promise

- Most often, Promises will be generated by an API function (e.g., `fetch`) and returned to you.
- But you can also create your own Promise.

```
var p = new Promise(function(resolve, reject) {
   if (/* condition */) {
      resolve(/* value */);  // fulfilled successfully
   }
   else {
      reject(/* reason */);  // error, rejected
   }
});
```

Example: Writing a Promise

- loadImage returns a promise to load a given image

```
function loadImage(url){
    return new Promise(function(resolve, reject) {
        var img  = new Image();
        img.src = url;
        img.onload = function(){
            resolve(img);
        }
        img.onerror = function(e){
            reject(e);
        }
    });
}
```

Once the image is loaded, we'll resolve the promise

If the image has an error, the promise is rejected

Writing a Promise

- Basic syntax:
  - do something (possibly asynchronous)
  - when you get the result, call resolve() and pass the final result
  - In case of error, call reject()

```
var p = new Promise( function(resolve,reject){
    // do something, who knows how long it will take?
    if(everythingIsOK)
    {
        resolve(stateIWantToSave);
    }
    else
        reject(Error("Some error happened"));
} );
```

Promises in Action

Promises in Action

- Firebase example: get some value from the database, then push some new value to the database, then print out “OK”

Promises in Action

- Firebase example: get some value from the database, then push some new value to the database, then print out “OK”

Promises in Action

- Firebase example: get some value from the database, then push some new value to the database, then print out “OK”

Promises in Action

- Firebase example: get some value from the database, then push some new value to the database, then print out “OK”

```
todosRef.child(keyToGet).once(‘value')
      .then(function(foundTodo){
          return foundTodo.val().text;
      })
      .then(function(theText){
          todosRef.push({'text' : "Seriously: " + theText});
      })
      .then(function(){
          console.log("OK!");
      })
      .catch(function(error){
          //something went wrong
      });
```

Promises in Action

- Firebase example: get some value from the database, then push some new value to the database, then print out “OK”

```
todosRef.child(keyToGet).once(‘value')
      .then(function(foundTodo){
          return foundTodo.val().text;
      })
      .then(function(theText){
          todosRef.push({'text' : "Seriously: " + theText});
      })
      .then(function(){
          console.log("OK!");
      })
      .catch(function(error){
          //something went wrong
      });
```

Do this

Promises in Action

- Firebase example: get some value from the database, then push some new value to the database, then print out “OK”

```
todosRef.child(keyToGet).once(‘value')
      .then(function(foundTodo){
          return foundTodo.val().text;
      })
      .then(function(theText){
          todosRef.push({'text' : "Seriously: " + theText});
      })
      .then(function(){
          console.log("OK!");
      })
      .catch(function(error){
          //something went wrong
      });
```

Do this

Then, do this

Promises in Action

- Firebase example: get some value from the database, then push some new value to the database, then print out “OK”

```
todosRef.child(keyToGet).once(‘value')
      .then(function(foundTodo){
          return foundTodo.val().text;
      })
      .then(function(theText){
          todosRef.push({'text' : "Seriously: " + theText});
      })
      .then(function(){
          console.log("OK!");
      })
      .catch(function(error){
          //something went wrong
      });
```

Do this

Then, do this

Then do this

Promises in Action

- Firebase example: get some value from the database, then push some new value to the database, then print out “OK”

```
todosRef.child(keyToGet).once(‘value')
      .then(function(foundTodo){
          return foundTodo.val().text;
      })
      .then(function(theText){
          todosRef.push({'text' : "Seriously: " + theText});
      })
      .then(function(){
          console.log("OK!");
      })
      .catch(function(error){
          //something went wrong
      });
```

Do this

Then, do this

Then do this

And if you ever had an error, do this

Async/Await

- The latest and greatest way to work with async functions
- A programming pattern that tries to make async code look more synchronous
- Just “await” something to happen before proceeding
- https://javascript.info/async-await

Async keyword

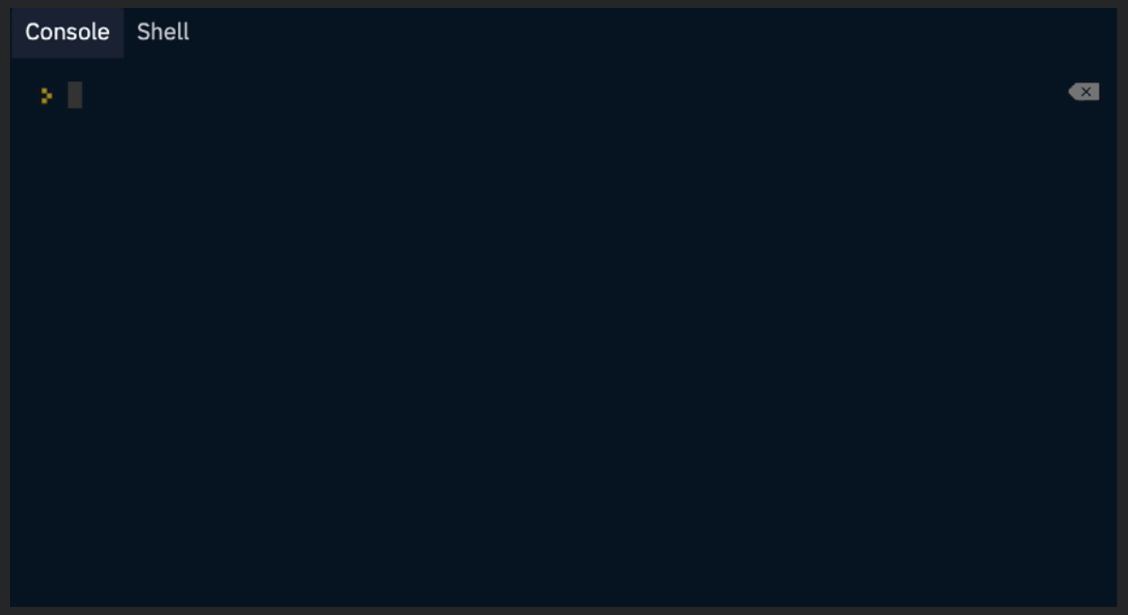
- Denotes a function that can block and resume execution later

```
async function hello() { return "Hello" };
hello();
```

- Automatically turns the return type into a Promise

Async/Await Example

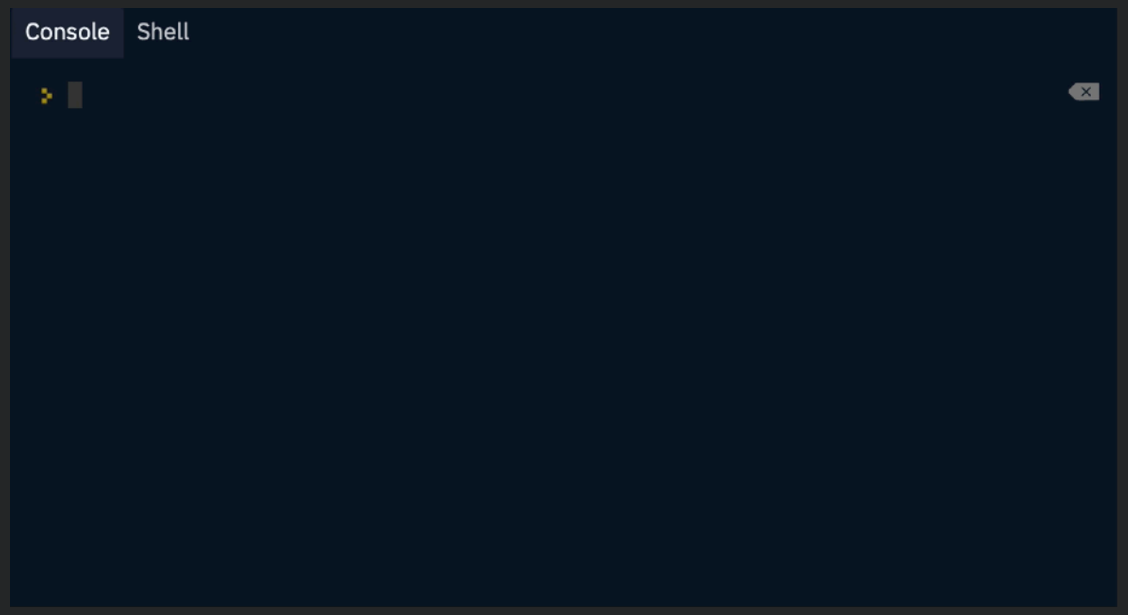
```
function resolveAfter2Seconds() {
    return new Promise(resolve => {
        setTimeout(() => {
            resolve('resolved');
        }, 2000);
    });
}

async function asyncCall() {
    console.log('calling');
    var result = await
resolveAfter2Seconds();
    console.log(result);
    // expected output: 'resolved'
}
```



https://replit.com/@kmoran/async-ex#script.js

Async/Await Example

```
function resolveAfter2Seconds() {
    return new Promise(resolve => {
        setTimeout(() => {
            resolve('resolved');
        }, 2000);
    });
}

async function asyncCall() {
    console.log('calling');
    var result = await
resolveAfter2Seconds();
    console.log(result);
    // expected output: 'resolved'
}
```



https://replit.com/@kmoran/async-ex#script.js

Async/Await -> Synchronous

```javascript
let lib = require("./lib.js");

async function getAndGroupStuff() {
    let thingsToFetch = ['t1', 't2', 't3', 's1', 's2',
‘s3’, 'm1', 'm2', 'm3', 't4'];
    let stuff = [];
    let ts, ms, ss;

    let promises = [];
    for (let thingToGet of thingsToFetch) {
        stuff.push(await lib.getPromise(thingToGet));
        console.log("Got a thing");
    }
    ts = await lib.groupPromise(stuff,"t");
    console.log("Made a group");
    ms = await lib.groupPromise(stuff,"m");
    console.log("Made a group");
    ss = await lib.groupPromise(stuff,"s");
    console.log("Made a group");
    console.log("Done");
}

getAndGroupStuff();
```

```
node v12.16.1
```

Async/Await -> Synchronous

```javascript
let lib = require("./lib.js");

async function getAndGroupStuff() {
    let thingsToFetch = ['t1', 't2', 't3', 's1', 's2',
‘s3’, 'm1', 'm2', 'm3', 't4'];
    let stuff = [];
    let ts, ms, ss;

    let promises = [];
    for (let thingToGet of thingsToFetch) {
        stuff.push(await lib.getPromise(thingToGet));
        console.log("Got a thing");
    }
    ts = await lib.groupPromise(stuff,"t");
    console.log("Made a group");
    ms = await lib.groupPromise(stuff,"m");
    console.log("Made a group");
    ss = await lib.groupPromise(stuff,"s");
    console.log("Made a group");
    console.log("Done");
}

getAndGroupStuff();
```

```
node v12.16.1
```

Async/Await

- Rules of the road:
  - You can only call **await** from a function that is **async**
  - You can only **await** on functions that return a **Promise**
  - Beware: await makes your code synchronous!

```
async function getAndGroupStuff() {
...
    ts = await lib.groupPromise(stuff,"t");
...
}
```

Week 4: Backend & HTTP Requests

Express

- Basic setup:
  - For get:

```
app.get("/somePath", function(req, res){
    //Read stuff from req, then call res.send(myResponse)
});
```

  - For post:

```
app.post("/somePath", function(req, res){
    //Read stuff from req, then call res.send(myResponse)
});
```

  - Serving static files:

```
app.use(express.static('myFileWithStaticFiles'));
```

    - Make sure to declare this *last*

- Additional helpful module - bodyParser (for reading POST data)

https://expressjs.com/

Demo: Hello World Server

1: Make a directory, `myapp`

Demo: Hello World Server

1: Make a directory, `myapp`

2: Enter that directory, type `npm init` (accept all defaults)

Creates a configuration file for your project

Demo: Hello World Server

1: Make a directory, `myapp`

2: Enter that directory, type `npm init` (accept all defaults)

3: Type `npm install express --save`

Creates a configuration file for your project

Tells NPM that you want to use express, and to save that in your project config

Demo: Hello World Server

1: Make a directory, `myapp`

2: Enter that directory, type `npm init` (accept all defaults)

Creates a configuration file for your project

3: Type `npm install express --save`

Tells NPM that you want to use express, and to save that in your project config

4: Create text file `app.js`:

```
var express = require('express');
var app = express();
var port = process.env.PORT || 3000;
app.get('/', function (req, res) {
  res.send('Hello World!');
});

app.listen(port, function () {
  console.log('Example app listening on port' + port);
});
```

Demo: Hello World Server

1: Make a directory, `myapp`

2: Enter that directory, type `npm init` (accept all defaults)

Creates a configuration file for your project

3: Type `npm install express --save`

Tells NPM that you want to use express, and to save that in your project config

4: Create text file `app.js`:

```
var express = require('express');
var app = express();
var port = process.env.PORT || 3000;
app.get('/', function (req, res) {
  res.send('Hello World!');
});

app.listen(port, function () {
  console.log('Example app listening on port' + port);
});
```

5: Type `node app.js`

6: Point your browser to http://localhost:3000

Runs your app

Demo: Hello World Server

```
var express = require(‘express');

var app = express();

var port = process.env.PORT || 3000;

app.get('/', function (req, res) {
  res.send('Hello World!');
});



app.listen(port, function () {
  console.log('Example app listening on port' + port);
});
```

Demo: Hello World Server

```
var express = require('express');  // Import the module express

var app = express();

var port = process.env.PORT || 3000;

app.get('/', function (req, res) {
  res.send('Hello World!');
});



app.listen(port, function () {
  console.log('Example app listening on port' + port);
});
```

Demo: Hello World Server

```
var express = require('express');  // Import the module express

var app = express();  // Create a new instance of express

var port = process.env.PORT || 3000;

app.get('/', function (req, res) {
  res.send('Hello World!');
});



app.listen(port, function () {
  console.log('Example app listening on port' + port);
});
```

Demo: Hello World Server

```
var express = require('express');  // Import the module express

var app = express();  // Create a new instance of express

var port = process.env.PORT || 3000;  // Decide what port we want express to listen on

app.get('/', function (req, res) {
  res.send('Hello World!');
});



app.listen(port, function () {
  console.log('Example app listening on port' + port);
});
```

Demo: Hello World Server

```
var express = require('express');  // Import the module express

var app = express();  // Create a new instance of express

var port = process.env.PORT || 3000;  // Decide what port we want express to listen on

app.get('/', function (req, res) {
  res.send('Hello World!');
});

app.listen(port, function () {
  console.log('Example app listening on port' + port);
});
```

// Create a callback for express to call when we have a "`get`" request to "`/`". That callback has access to the request (`req`) and response (`res`).

Demo: Hello World Server

```
var express = require('express');  // Import the module express

var app = express();  // Create a new instance of express

var port = process.env.PORT || 3000;  // Decide what port we want express to listen on

app.get('/', function (req, res) {
  res.send('Hello World!');
});
```

// Create a callback for express to call when we have a "`get`" request to "`/`". That callback has access to the request (`req`) and response (`res`).

```
app.listen(port, function () {
  console.log('Example app listening on port' + port);
});
```

// Tell our new instance of express to listen on `port`, and print to the console once it starts successfully

Core Concept: Routing

- The definition of end points (URIs) and how they respond to client requests.
  - app.METHOD(PATH, HANDLER)
  - METHOD: all, get, post, put, delete, [and others]
  - PATH: string (e.g., the url)
  - HANDLER: call back

```
app.post('/', function (req, res) {
  res.send('Got a POST request');
});
```

Route Paths

- Can specify strings, string patterns, and regular expressions
  - Can use ?, +, *, and ()
- Matches request to root route

```
app.get('/', function (req, res) {
  res.send('root');
});
```

- Matches request to /about

```
app.get('/about', function (req, res) {
  res.send('about');
});
```

- Matches request to /abe and /abcde

```
app.get('/ab(cd)?e', function(req, res) {
 res.send('ab(cd)?e');
});
```

Route Parameters

- Named URL segments that capture values at specified location in URL
  - Stored into `req.params` object by name
- Example
  - Route path */users/**:userId**/books/**:bookId***
  - Request URL *http://localhost:3000/users/34/books/8989*
  - Resulting `req.params: { "userId": "34", "bookId": "8989" }`

```
app.get('/users/:userId/books/:bookId', function(req, res)
{
  res.send(req.params);
});
```

Route Handlers

- You can provide multiple callback functions that behave like middleware to handle a request
- The only exception is that these callbacks might invoke next('route') to bypass the remaining route callbacks.
- You can use this mechanism to impose pre-conditions on a route, then pass control to subsequent routes if there's no reason to proceed with the current route.

```
app.get('/example/b', function (req, res, next) {
  console.log('the response will be sent by the next function ...')
  next()
}, function (req, res) {
  res.send('Hello from B!')
})
```

Request Object

- Enables reading properties of HTTP request
  - `req.body`: JSON submitted in request body (*must* define body-parser to use)
  - `req.ip`: IP of the address
  - `req.query`: URL query parameters

HTTP Responses

- Larger number of response codes (200 OK, 404 NOT FOUND)
- Message body only allowed with certain response status codes

```
HTTP/1.1 200 OK
Date: Mon, 23 May 2005 22:38:34 GMT
Content-Type: text/html; charset=UTF-8
Content-Encoding: UTF-8
Content-Length: 138
Last-Modified: Wed, 08 Jan 2003 23:11:55 GMT
Server: Apache/1.3.3.7 (Unix) (Red-Hat/Linux)
ETag: "3f80f-1b6-3e1cb03b"
Accept-Ranges: bytes
Connection: close

<html>
<head>
  <title>An Example Page</title>
</head>
<body>
  Hello World, this is a very simple HTML document.
</body>
</html>
```

HTTP Responses

- Larger number of response codes (200 OK, 404 NOT FOUND)
- Message body only allowed with certain response status codes

```
HTTP/1.1 200 OK
Date: Mon, 23 May 2005 22:38:34 GMT
Content-Type: text/html; charset=UTF-8
Content-Encoding: UTF-8
Content-Length: 138
Last-Modified: Wed, 08 Jan 2003 23:11:55 GMT
Server: Apache/1.3.3.7 (Unix) (Red-Hat/Linux)
ETag: "3f80f-1b6-3e1cb03b"
Accept-Ranges: bytes
Connection: close

<html>
<head>
  <title>An Example Page</title>
</head>
<body>
  Hello World, this is a very simple HTML document.
</body>
</html>
```

"OK response"

"HTML returned content"

[HTML data]

HTTP Responses

- Larger number of response codes (200 OK, 404 NOT FOUND)
- Message body only allowed with certain response status codes

```
HTTP/1.1 200 OK
Date: Mon, 23 May 2005 22:38:34 GMT
Content-Type: text/html; charset=UTF-8
Content-Encoding: UTF-8
Content-Length: 138
Last-Modified: Wed, 08 Jan 2003 23:11:55 GMT
Server: Apache/1.3.3.7 (Unix) (Red-Hat/Linux)
ETag: "3f80f-1b6-3e1cb03b"
Accept-Ranges: bytes
Connection: close

<html>
<head>
  <title>An Example Page</title>
</head>
<body>
  Hello World, this is a very simple HTML document.
</body>
</html>
```

"OK response"

Response status codes:
1xx Informational
2xx Success
3xx Redirection
4xx Client error
5xx Server error

"HTML returned content"

[HTML data]

HTTP Responses

- Larger number of response codes (200 OK, 404 NOT FOUND)
- Message body only allowed with certain response status codes

```
HTTP/1.1 200 OK
Date: Mon, 23 May 2005 22:38:34 GMT
Content-Type: text/html; charset=UTF-8
Content-Encoding: UTF-8
Content-Length: 138
Last-Modified: Wed, 08 Jan 2003 23:11:55 GMT
Server: Apache/1.3.3.7 (Unix) (Red-Hat/Linux)
ETag: "3f80f-1b6-3e1cb03b"
Accept-Ranges: bytes
Connection: close

<html>
<head>
  <title>An Example Page</title>
</head>
<body>
  Hello World, this is a very simple HTML document.
</body>
</html>
```

[HTML data]

"OK response"

Response status codes:
1xx Informational
2xx Success
3xx Redirection
4xx Client error
5xx Server error

"HTML returned content"

Common MIME types:
application/json
application/pdf
image/png

Response Object

- Enables a response to client to be generated
  - `res.send()` - send string content
  - `res.download()` - prompts for a file download
  - `res.json()` - sends a response w/ application/json Content-Type header
  - `res.redirect()` - sends a redirect response
  - `res.sendStatus()` - sends only a status message
  - `res.sendFile()` - sends the file at the specified path

```
app.get('/users/:userId/books/:bookId', function(req, res) {
  res.json({ “id”: req.params.bookID });
});
```

Describing Responses

- What happens if something goes wrong while handling HTTP request?
  - How does client know what happened and what to try next?
- HTTP offers response status codes describing the nature of the response
  - 1xx Informational: Request received, continuing
  - 2xx Success: Request received, understood, accepted, processed
    - 200: OK
  - 3xx Redirection: Client must take additional action to complete request
    - 301: Moved Permanently
    - 307: Temporary Redirect

https://en.wikipedia.org/wiki/List_of_HTTP_status_codes

Describing Errors

- 4xx Client Error: client did not make a valid request to server. Examples:
  - 400 Bad request (e.g., malformed syntax)
  - 403 Forbidden: client lacks necessary permissions
  - 404 Not found
  - 405 Method Not Allowed: specified HTTP action not allowed for resource
  - 408 Request Timeout: server timed out waiting for a request
  - 410 Gone: Resource has been intentionally removed and will not return
  - 429 Too Many Requests

Describing Errors

- 5xx Server Error: The server failed to fulfill an apparently valid request.
  - 500 Internal Server Error: generic error message
  - 501 Not Implemented
  - 503 Service Unavailable: server is currently unavailable

Error Handling in Express

- Express offers a default error handler
- Can specific error explicitly with status
  - `res.status(500);`

Persisting Data in Memory

- Can declare a global variable in node
  - i.e., a variable that is not declared inside a class or function
- Global variables persist between requests
- Can use them to store state in memory
- Unfortunately, if server crashes or restarts, state will be lost
  - Will look later at other options for persistence

Making HTTP Requests

- May want to request data from other servers from backend
- Fetch
  - Makes an HTTP request, returns a Promise for a response
  - Part of standard library in browser, but need to install library to use in backend
- Installing:

```
npm install node-fetch --save
```

- Use:

```
const fetch = require('node-fetch');

fetch('https://github.com/')
    .then(res => res.text())
    .then(body => console.log(body));

var res = await fetch('https://github.com/');
```

https://www.npmjs.com/package/node-fetch

Responding Later

- What happens if you'd like to send data back to client in response, but not until something else happens (e.g., your request to a different server finishes)?
- Solution: wait for event, then send the response!

```
fetch('https://github.com/')
    .then(res => res.text())
    .then(body => res.send(body));
```

REST: REpresentational State Transfer

- Defined by Roy Fielding in his 2000 Ph.D. dissertation
  - Used by Fielding to design HTTP 1.1 that generalizes URLs to URIs
  - http://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf
- *"Throughout the HTTP standardization process, I was called on to defend the design choices of the Web. That is an extremely difficult thing to do… I had comments from well over 500 developers, many of whom were distinguished engineers with decades of experience. That process honed my model down to a core set of principles, properties, and constraints that are now called REST."*
- Interfaces that follow REST principles are called RESTful

Properties of REST

- Performance
- Scalability
- Simplicity of a Uniform Interface
- Modifiability of components (even at runtime)
- Visibility of communication between components by service agents
- Portability of components by moving program code with data
- Reliability

Principles of REST

- Client server: separation of concerns (reuse)
- Stateless: each client request contains all information necessary to service request (scaling)
- Cacheable: clients and intermediaries may cache responses. (scaling)
- Layered system: client cannot determine if it is connected to end server or intermediary along the way. (scaling)
- Uniform interface for resources: a single uniform interface (URIs) simplifies and decouples architecture (change & reuse)

Uniform Interface for Resources

- Originally files on a web server
  - URL refers to directory path and file of a resource
- But… URIs might be used as an identity for any entity
  - A person, location, place, item, tweet, email, detail view, like
  - *Does not matter* if resource is a file, an entry in a database, retrieved from another server, or computed by the server on demand
  - Resources offer an *interface* to the server describing the resources with which clients can interact

URI: Universal Resource Identifier

- Uniquely describes a resource
  - https://mail.google.com/mail/u/0/#inbox/157d5fb795159ac0
  - https://www.amazon.com/gp/yourstore/home/ref=nav_cs_ys
  - http://gotocon.com/dl/goto-amsterdam-2014/slides/StefanTilkov_RESTIDontThinkItMeansWhatYouThinkItDoes.pdf
  - Which is a file, external web service request, or stored in a database?
    - It does not matter
- As client, only matters what actions we can *do* with resource, not how resource is represented on server

Intermediaries

Web “Front End”

“Origin” server

HTTP Request

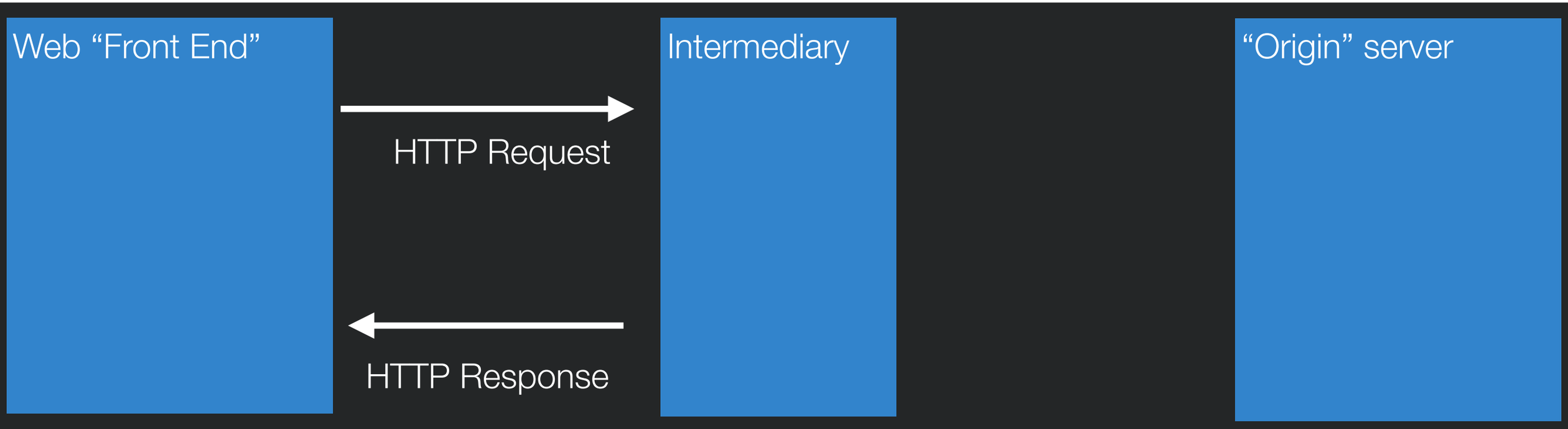
```
HTTP GET http://api.wunderground.com/api/
3bee87321900cf14/conditions/q/VA/Fairfax.json
```

HTTP Response

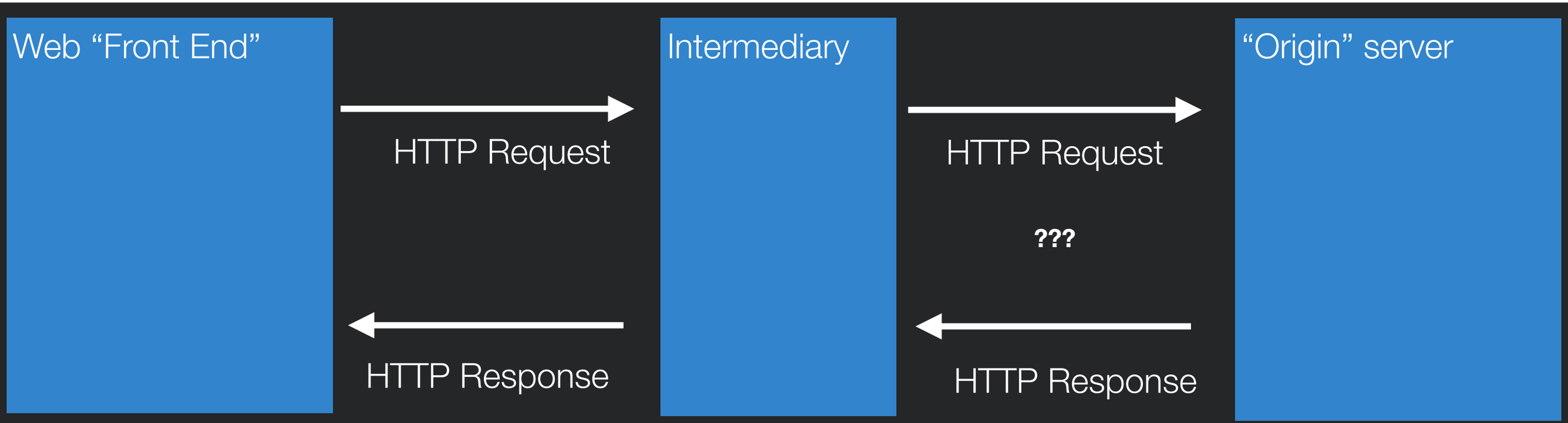
```
HTTP/1.1 200 OK
Server: Apache/2.2.15 (CentOS)
Access-Control-Allow-Origin: *
Access-Control-Allow-Credentials: true
X-CreationTime: 0.134
Last-Modified: Mon, 19 Sep 2016 17:37:52 GMT
Content-Type: application/json; charset=UTF-8
Expires: Mon, 19 Sep 2016 17:38:42 GMT
Cache-Control: max-age=0, no-cache
Pragma: no-cache
Date: Mon, 19 Sep 2016 17:38:42 GMT
Content-Length: 2589
Connection: keep-alive
```

```
{
  "response": {
  "version":"0.1"
```

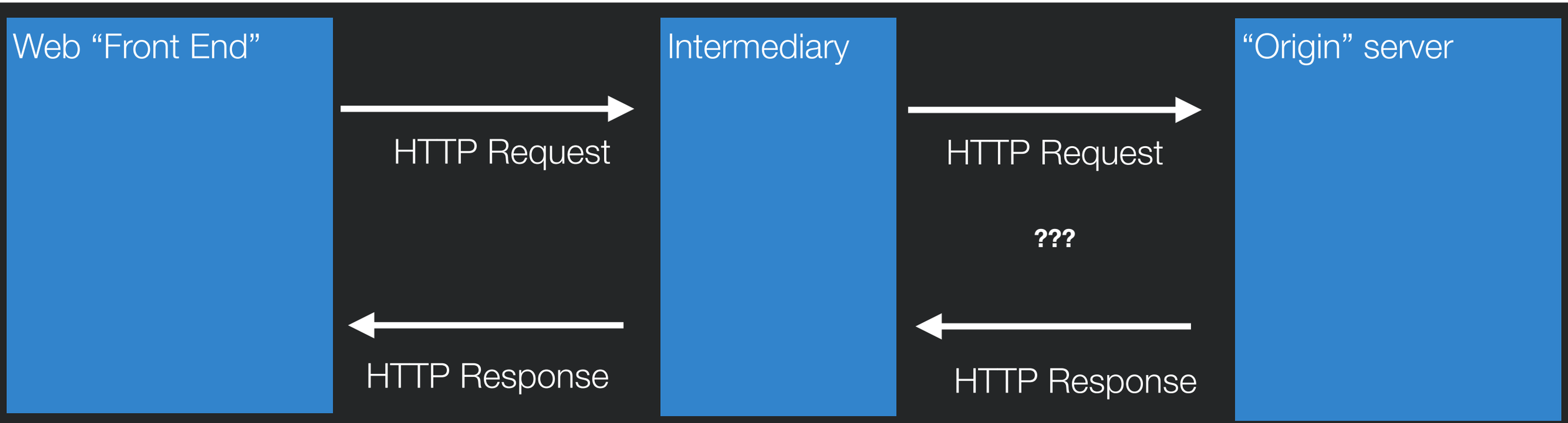
Intermediaries



Intermediaries



Intermediaries



- Client interacts with a resource identified by a URI
- But it never knows (or cares) whether it interacts with origin server or an unknown intermediary server
  - Might be randomly load balanced to one of many servers
  - Might be cache, so that large file can be stored locally
    - (e.g., GMU caching an OSX update)
  - Might be server checking security and rejecting requests

Challenges with intermediaries

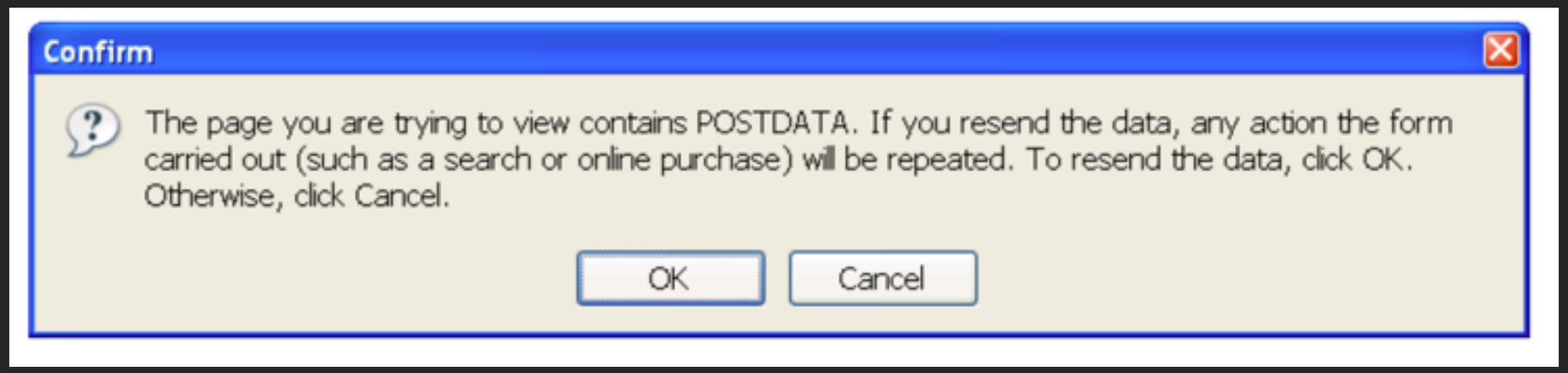
- But can all requests really be intercepted in the same way?
  - Some requests might produce a change to a resource
    - Can’t just cache a response… would not get updated!
  - Some requests might create a change every time they execute
    - Must be careful retrying failed requests or could create extra copies of resources

HTTP Actions

- How do intermediaries know what they can and cannot do with a request?
- Solution: HTTP Actions
  - Describes what will be done with resource
  - GET: retrieve the current state of the resource
  - PUT: modify the state of a resource
  - DELETE: clear a resource
  - POST: initialize the state of a new resource

HTTP Actions

- GET: safe method with no side effects
  - Requests can be intercepted and replaced with cache response
- PUT, DELETE: idempotent method that can be repeated with same result
  - Requests that fail can be retried indefinitely till they succeed
- POST: creates new element
  - Retrying a failed request might create duplicate copies of new resource



Week 5: Persistence & More Microservices

URI Design

- URIs represent a contract about what resources your server exposes and what can be done with them
- Leave out **anything that might change**
  - Content author names, status of content, other keys that might change
  - File name extensions: response describes content type through MIME header not extension (e.g., .jpg, .mp3, .pdf)
  - Server technology: should not reference technology (e.g., .cfm, .jsp)
- Endeavor to make all changes backwards compatible
  - Add new resources and actions rather than remove old
- If you must change URI structure, support old URI structure **and** new URI structure

Nouns vs. Verbs

- URIs should hierarchically identify **nouns** describing **resources** that exist
- Verbs describing actions that can be taken with resources should be described with an HTTP **action**

- PUT /cities/:cityID (nouns: cities, :cityID)(verb: PUT)
- GET /cities/:cityID (nouns: cities, :cityID)(verb: GET)

- Want to offer **expressive** abstraction that can be reused for many scenarios

Support Reuse

- You have your own frontend for cityinfo.org. But everyone now wants to build their own sites on top of your city analytics.

- Can they do that?

cityinfo.org

Microservice API

GET /cities

GET /populations

Support Reuse

cityinfo.org

Microservice API

/topCities GET
/topCities/:cityID/descrip PUT, GET

/city/:cityID GET, PUT, POST, DELETE
/city/:cityID/averages GET
/city/:cityID/weather GET
/city/:cityID/transitProvders GET, POST
/city/:cityID/transitProvders/:providerID GET, PUT, DELETE

What Happens When a Request has Many Parameters?

- `/topCities/:cityID/descrip` PUT
- Shouldn't this really be something more like
  - `/topCities/:cityID/descrip/:descriptionText/:submitter/:time/`

Solution 1: Query strings

```
var express = require('express');
var app = express();

app.put('/topCities/:cityID', function(req, res){
    res.send(`descrip: ${req.query.descrip} submitter: ${req.query.submitter}`);
});

app.listen(3000);
```

- Use req.query to retrieve
- Shows up in URL string, making it possible to store full URL
  - e.g., user adds a bookmark to URL
- Sometimes works well for short params

Solution 2: JSON Request Body

- PUT /topCities/Memphis
  { "descrip": "Memphis is a city of ...",
  "submitter": "Dan", "time": 1025313 }

- Best solution for all but the simplest parameters (and often times everything)

- Use body-parser package and req.body to retrieve

```
$npm install body-parser
```

```
var express    = require('express');
var bodyParser = require('body-parser');

var app = express();

// parse application/json
app.use(bodyParser.json());

app.put('/topCities/:cityID', function(req, res){
    res.send(`descrip: ${req.body.descrip} submitter: ${req.body.submitter}`);
});

app.listen(3000);
```


https://www.npmjs.com/package/body-parser

Storing state in a global variable

- **Global variables**

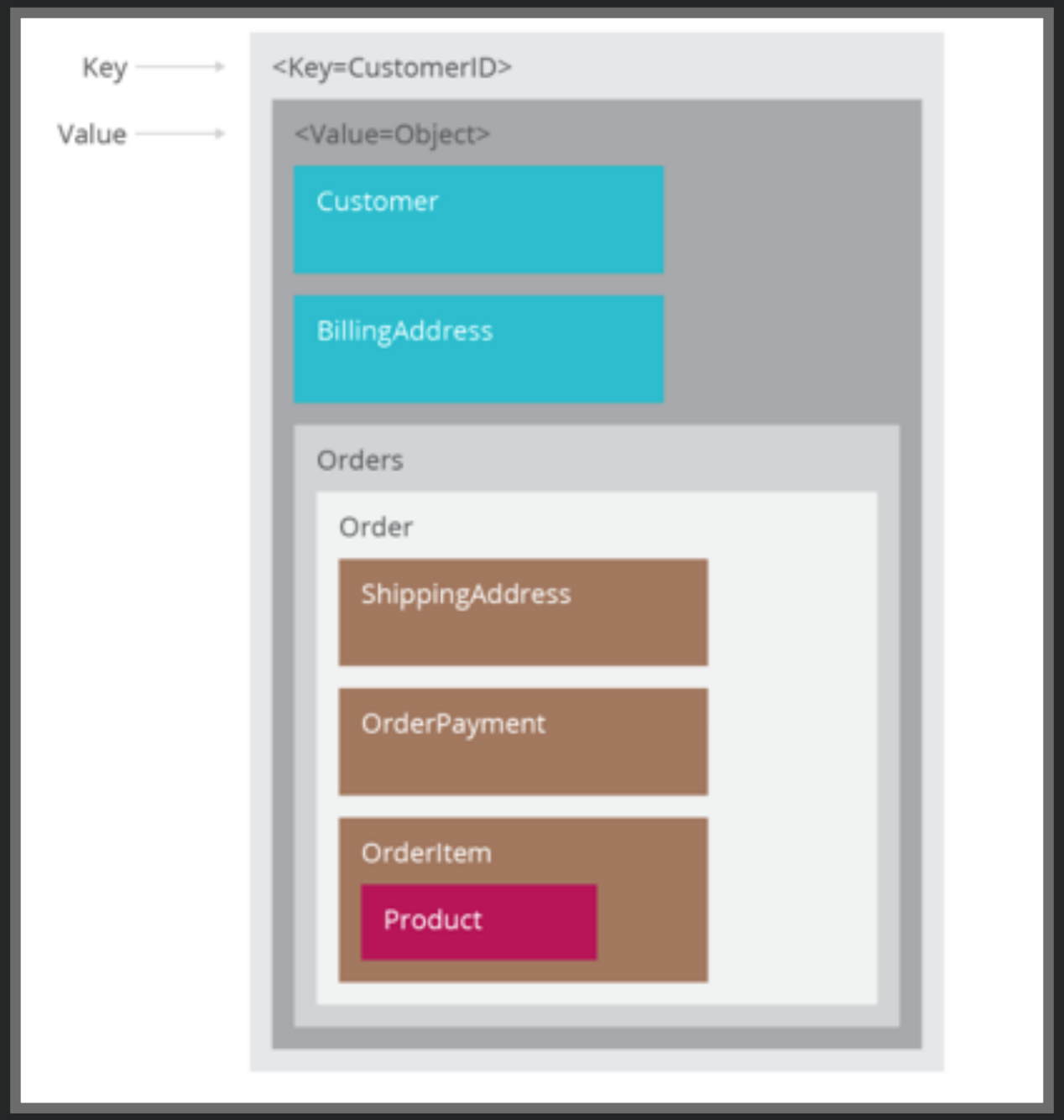
```
var express = require('express');
var app = express();
var port = process.env.port || 3000;

var counter = 0;
app.get('/', function (req, res) {
    res.send('Hello World has been said ' + counter + ' times!');
    counter++;
});

app.listen(port, function () {
    console.log('Example app listening on port' + port);
});
```

- Pros/cons?
  - Keep data between requests
  - *Goes away* when your server stops
    - Should use for transient state or as cache

NoSQL

- ***non SQL***, non-relational, "not only" SQL databases
- Emphasizes ***simplicity*** & ***scalability*** over support for relational queries
- Important characteristics
  - Schema-less: each row in dataset can have different fields (just like JSON!)
  - Non-relational: no structure linking tables together or queries to "join" tables
  - (Often) weaker consistency: after a field is updated, all clients *eventually* see the update but may see older data in the meantime
- Advantages: greater scalability, faster, simplicity, easier integration with code
- Several types. We'll look only at key-value.

Key-Value NoSQL



https://www.thoughtworks.com/insights/blog/nosql-databases-overview

Week 6: Security & HTML

Threat Models

- What is being defended?
  - What resources are important to defend?
  - What malicious actors exist and what attacks might they employ?

- Who do we trust?
  - What entities or parts of system can be considered secure and trusted
  - Have to trust **something**!

Security Requirements for Web Apps

1. Authentication
   - Verify the ***identify*** of the parties involved
   - Threat: Impersonation. A person pretends to be someone they are not.
2. Authorization
3. Confidentiality
   - Ensure that ***information*** is given only to authenticated parties
   - Threat: Eavesdropping. Information leaks to someone that should not have it.
4. Integrity
   - Ensure that information is ***not changed*** or tampered with
   - Threat: ***Tampering***.

HTTPS: HTTP over SSL

- Establishes secure connection from client to server
  - Uses SSL to encrypt traffic
- Ensures that others can’t impersonate server by establishing certificate authorities that vouch for server.
- Server trusts an HTTPS connection iff
  - The user trusts that the browser software correctly implements HTTPS with correctly pre-installed certificate authorities.
  - The user trusts the certificate authority to vouch only for legitimate websites.
  - The website provides a valid certificate, which means it was signed by a trusted authority.
  - The certificate correctly identifies the website (e.g., certificate received for “https://example.com" is for "example.com" and not other entity).

Using HTTPS

- If using HTTPS, important that all scripts are loaded through HTTPS
  - If mixed script from untrusted source served through HTTP, attacker could still modify this script, defeating benefits of HTTPS
- Example attack:
  - Banking website loads Bootstrap through HTTP rather than HTTPS
  - Attacker intercepts request for Bootstrap script, replaces with malicious script that steals user data or executes malicious action

Authentication

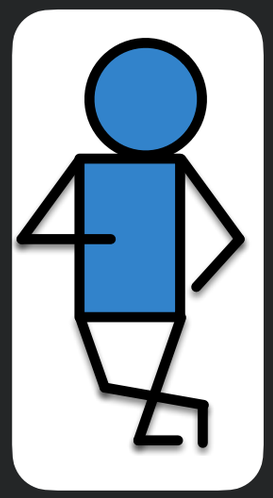
- How can we know the identify of the parties involved
- Want to customize experience based on identity
  - But need to determine identity first!
- Options
  - Ask user to create a new username and password
    - Lots of work to manage (password resets, storing passwords securely, …)
    - Hard to get right (#2 on the OWASP Top 10 Vulnerability List)
    - User does not really want another password…
  - Use an authentication provider to authenticate user
    - Google, FB, Twitter, Github, …

Authentication Provider

- Creates and tracks the identity of the user
- Instead of signing in directly to website, user signs in to authentication provider
  - Authentication provider issues token that uniquely proves identity of user

An OAuth Conversation

Goal: `TodosApp` can post events to `User's` calendar. `TodosApp` never finds out `User's` email or password



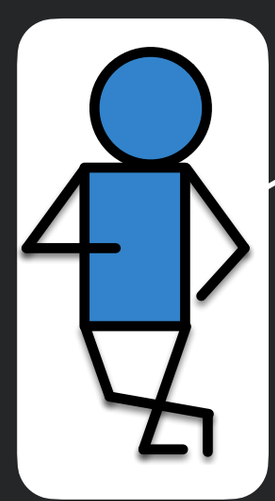
User

TodosApp

Google Calendar

An OAuth Conversation

Goal: `TodosApp` can post events to `User's` calendar. `TodosApp` never finds out `User's` email or password



User

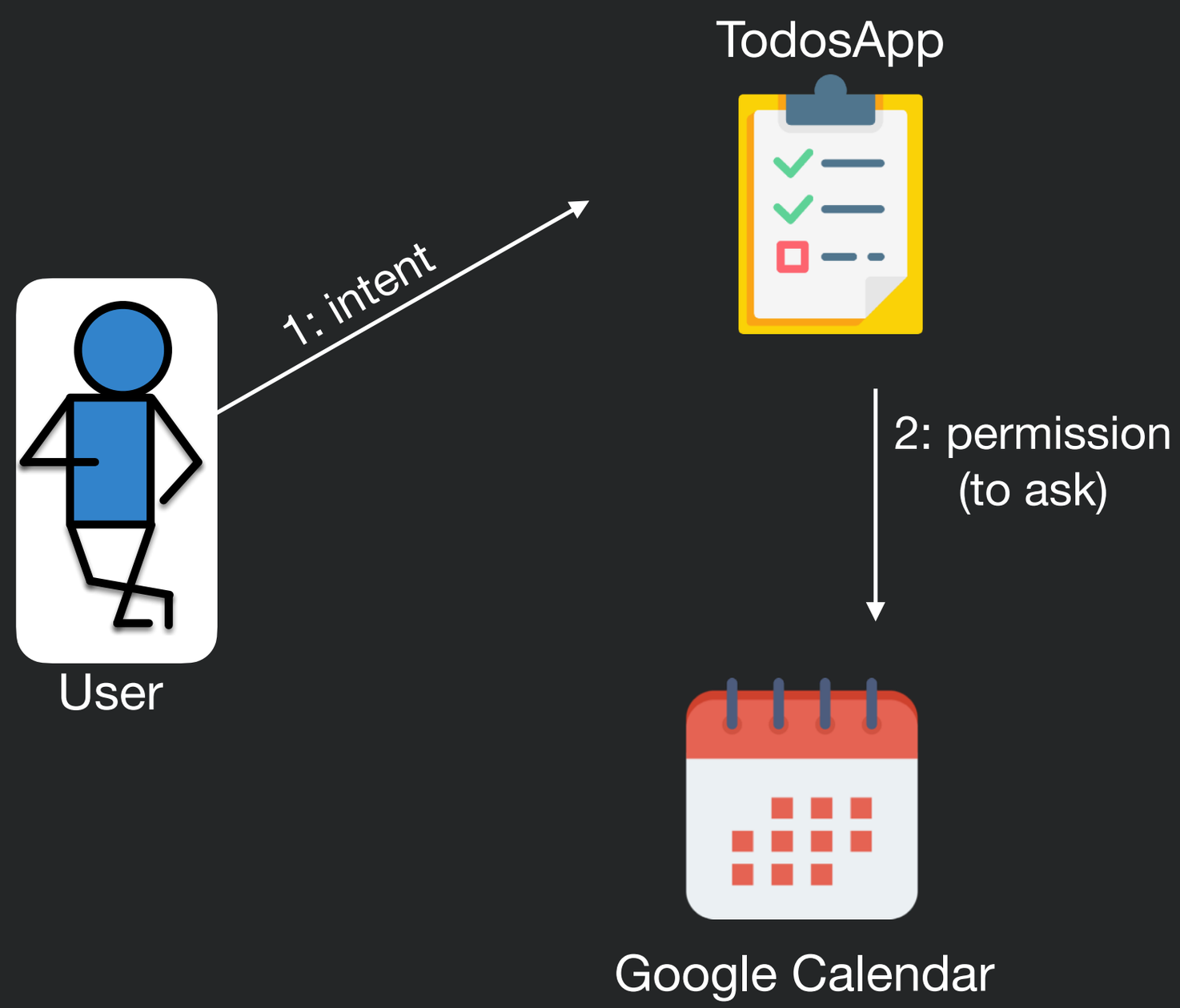
1: intent

TodosApp

Google Calendar

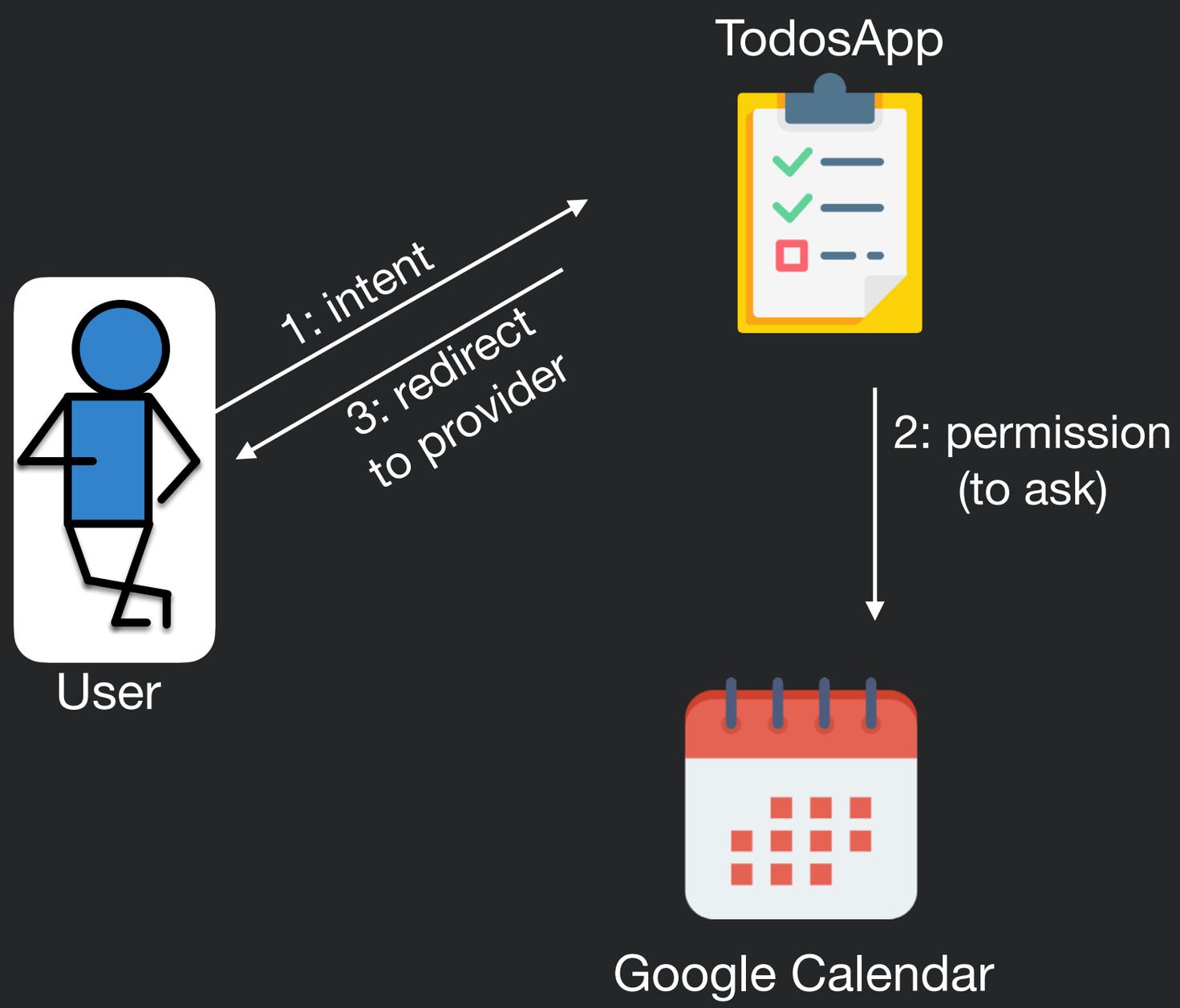
An OAuth Conversation

Goal: `TodosApp` can post events to `User's` calendar. `TodosApp` never finds out `User's` email or password



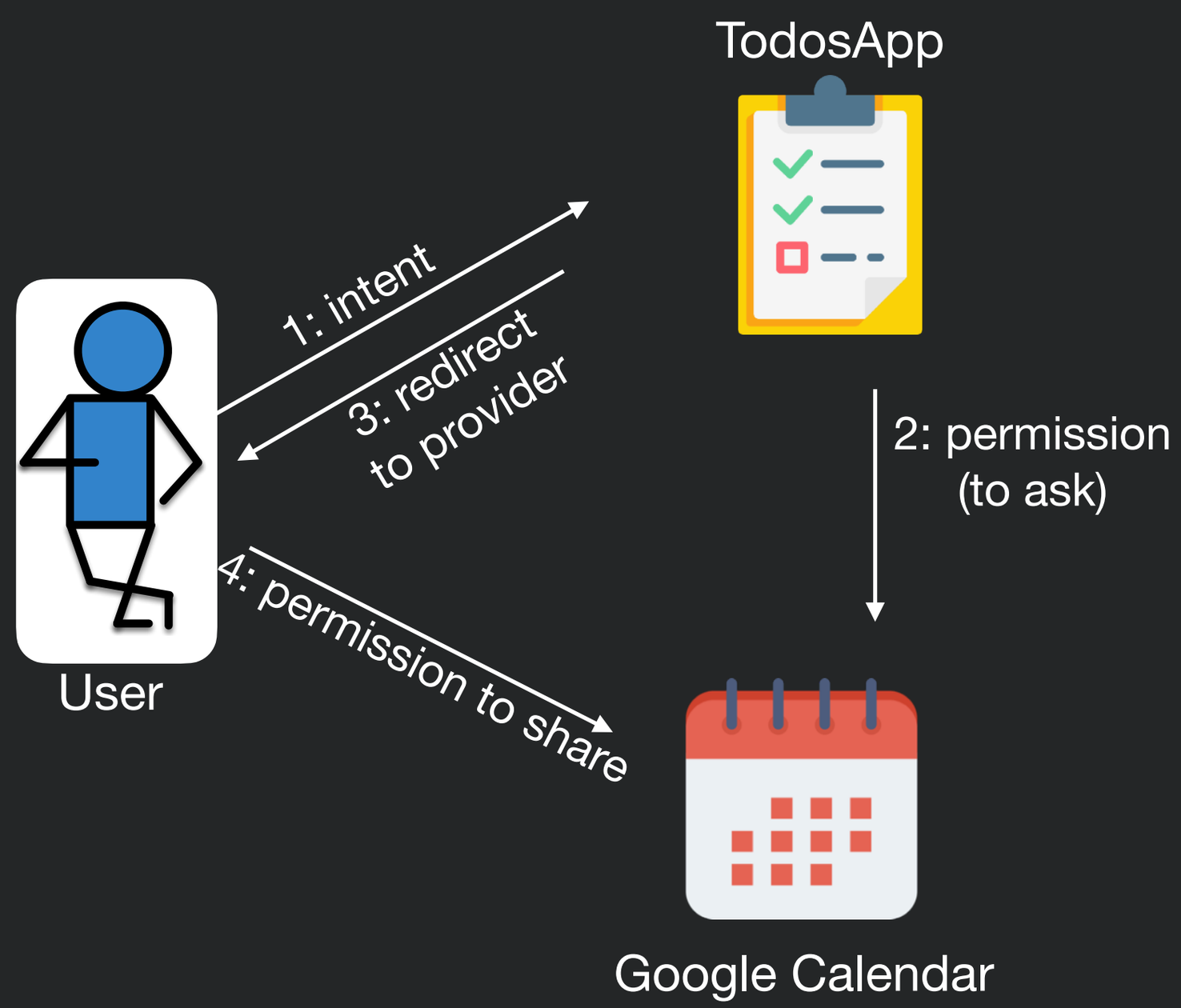
An OAuth Conversation

Goal: `TodosApp` can post events to `User's` calendar. `TodosApp` never finds out `User's` email or password



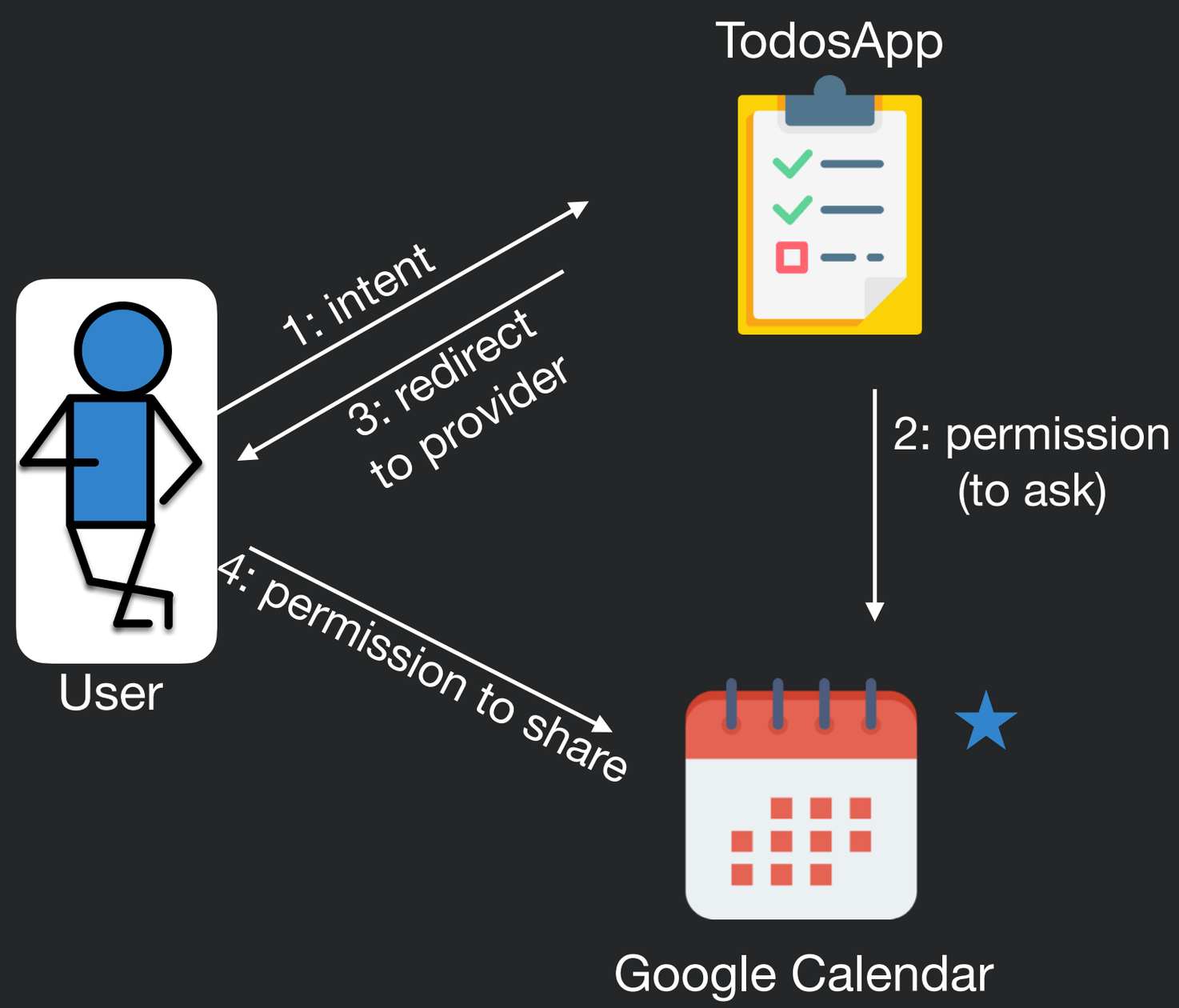
An OAuth Conversation

Goal: `TodosApp` can post events to `User's` calendar. `TodosApp` never finds out `User's` email or password



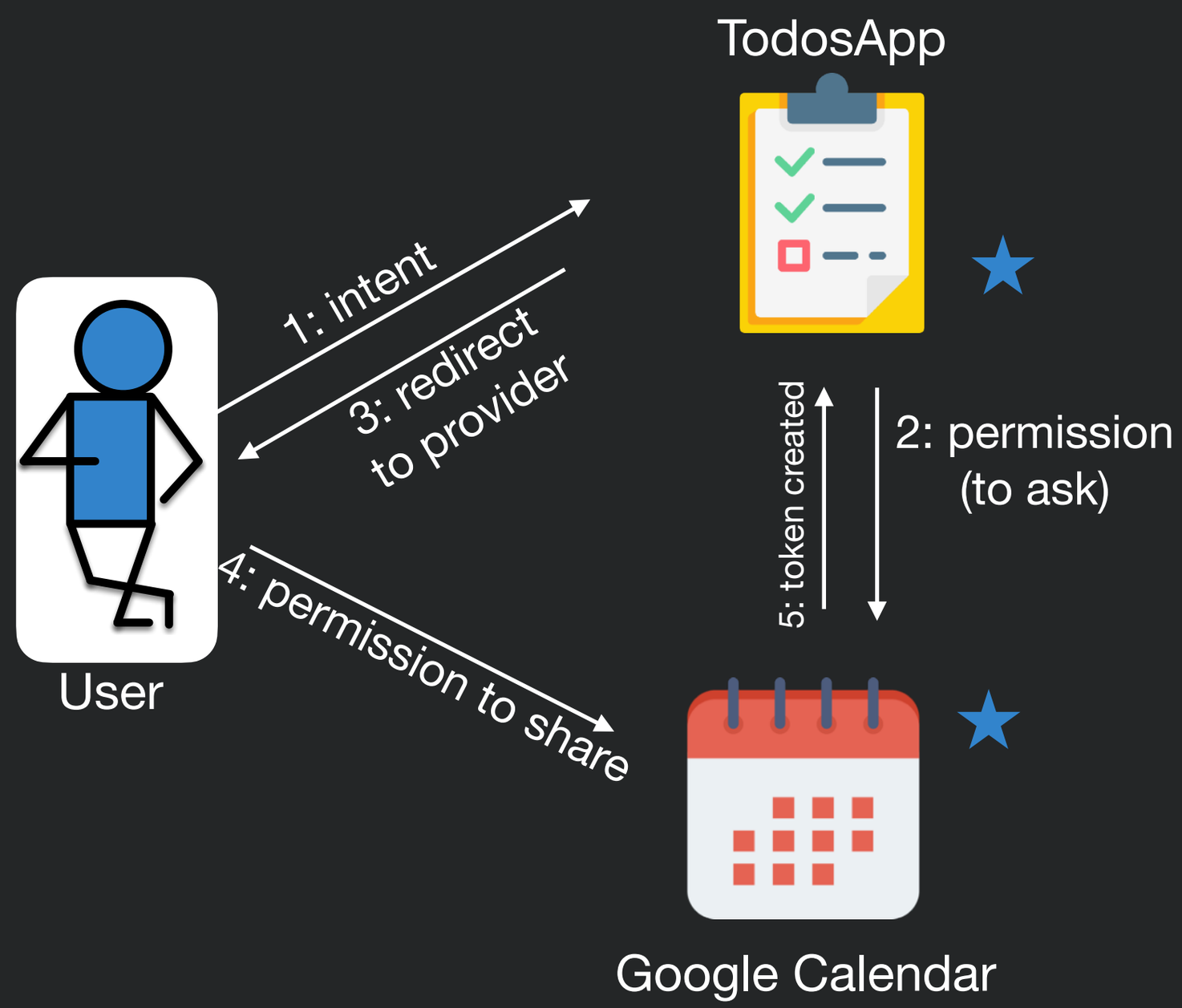
An OAuth Conversation

Goal: `TodosApp` can post events to `User's` calendar. `TodosApp` never finds out `User's` email or password



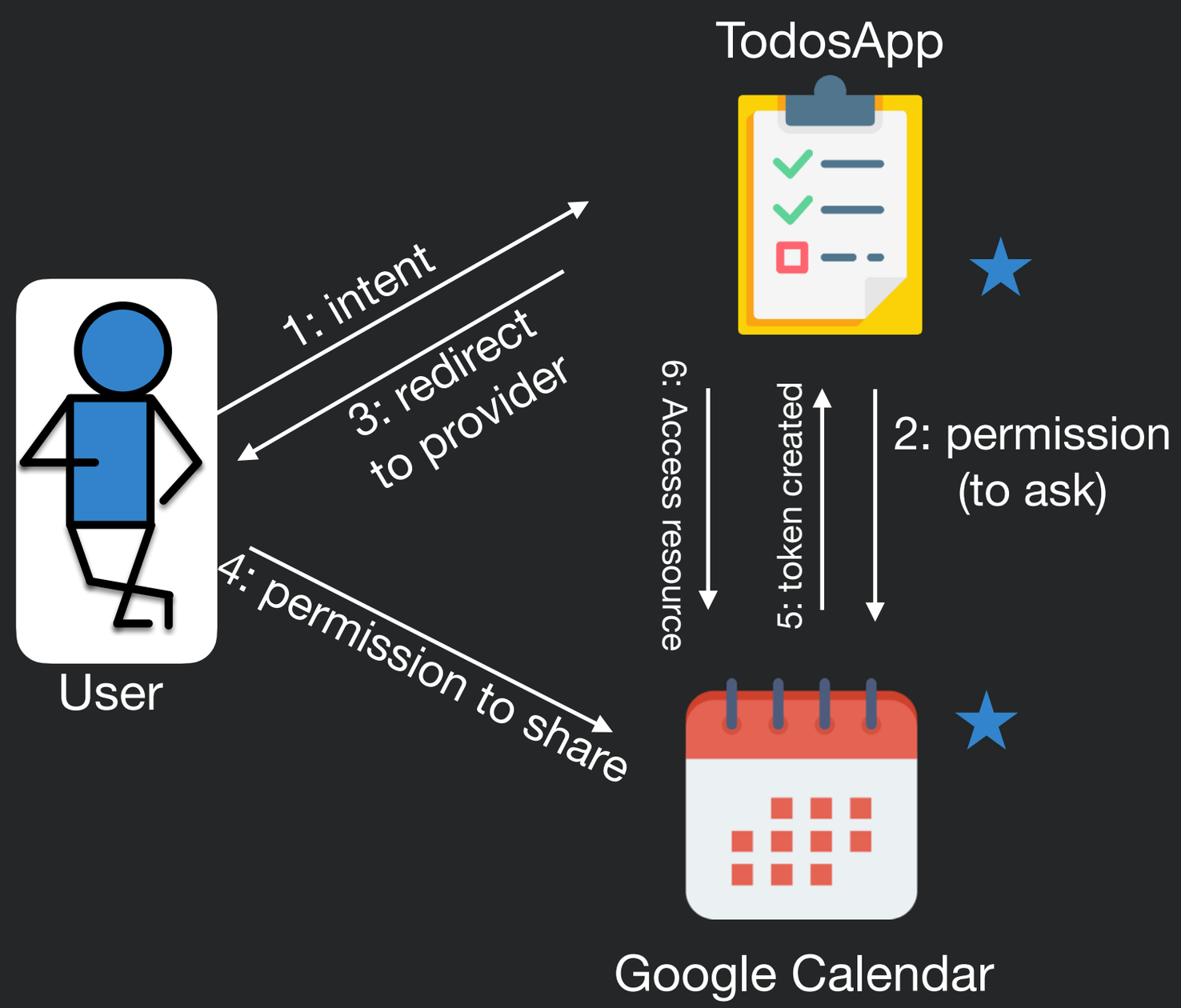
An OAuth Conversation

Goal: `TodosApp` can post events to `User's` calendar. `TodosApp` never finds out `User's` email or password



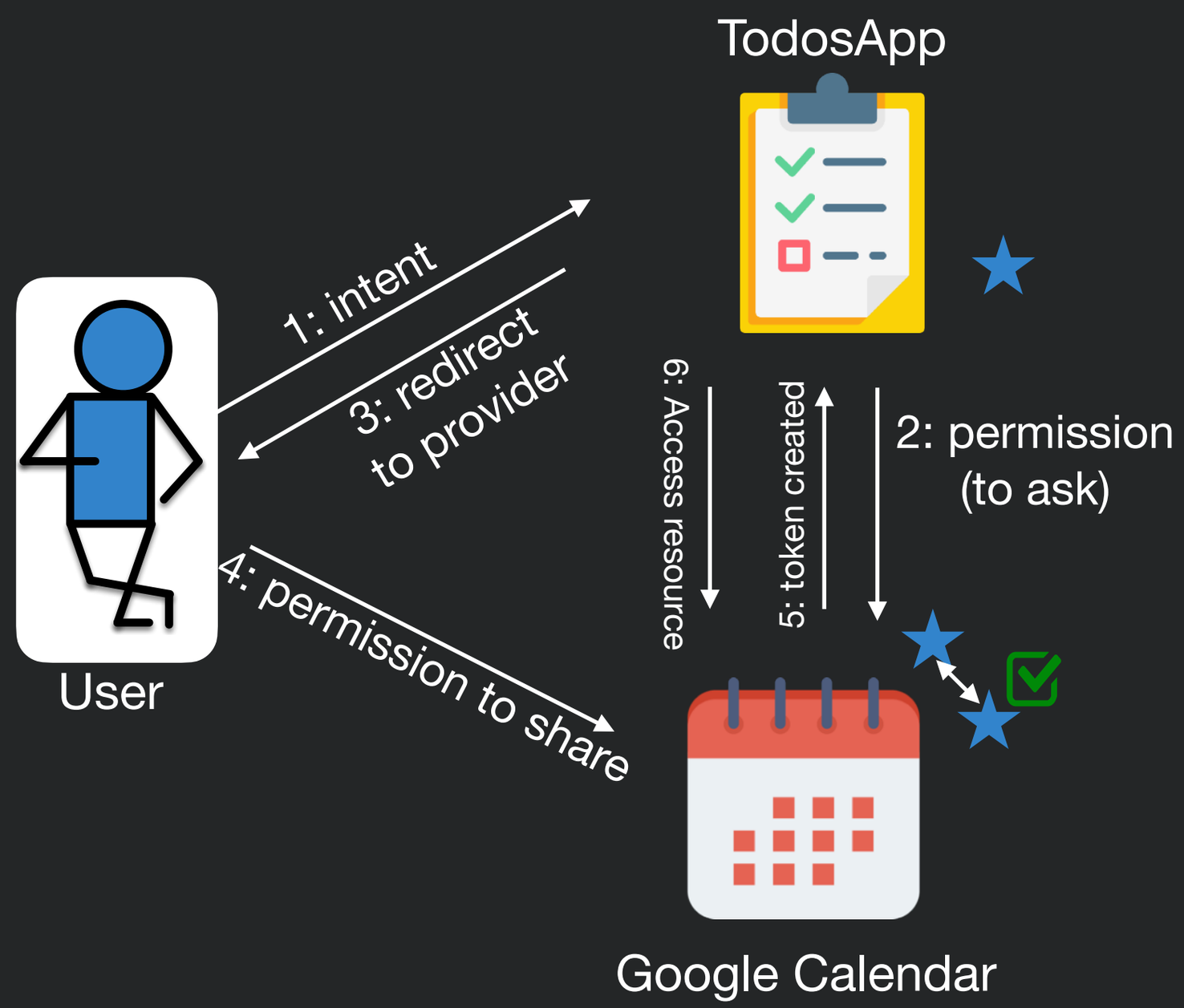
An OAuth Conversation

Goal: `TodosApp` can post events to `User's` calendar. `TodosApp` never finds out `User's` email or password

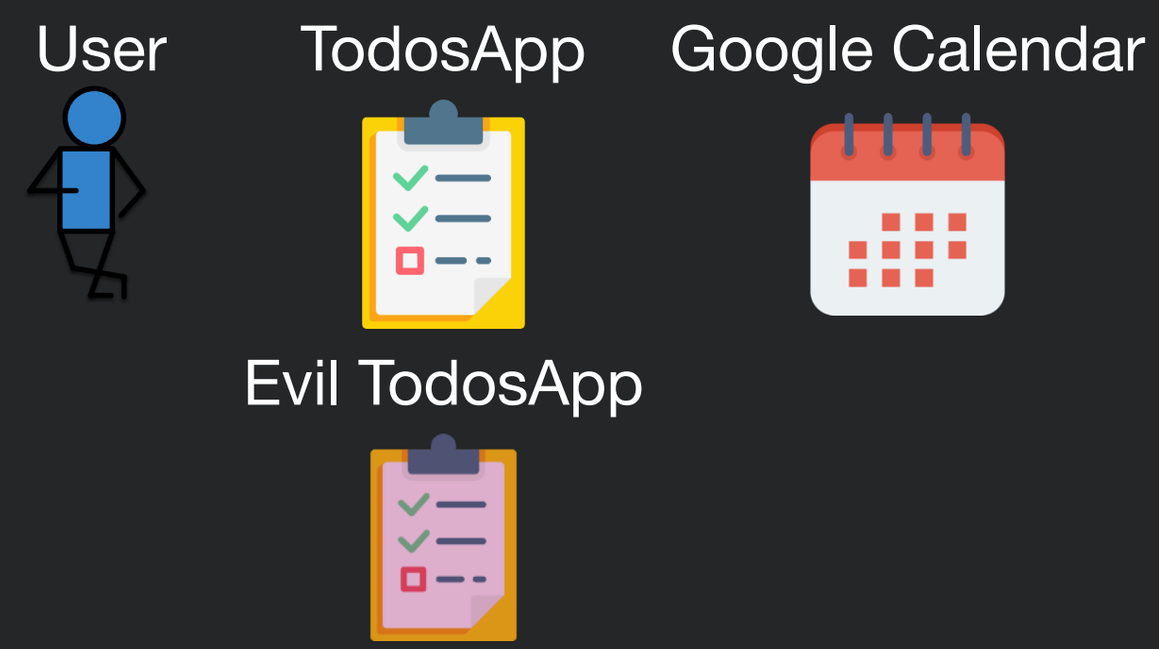


An OAuth Conversation

Goal: `TodosApp` can post events to `User's` calendar. `TodosApp` never finds out `User's` email or password

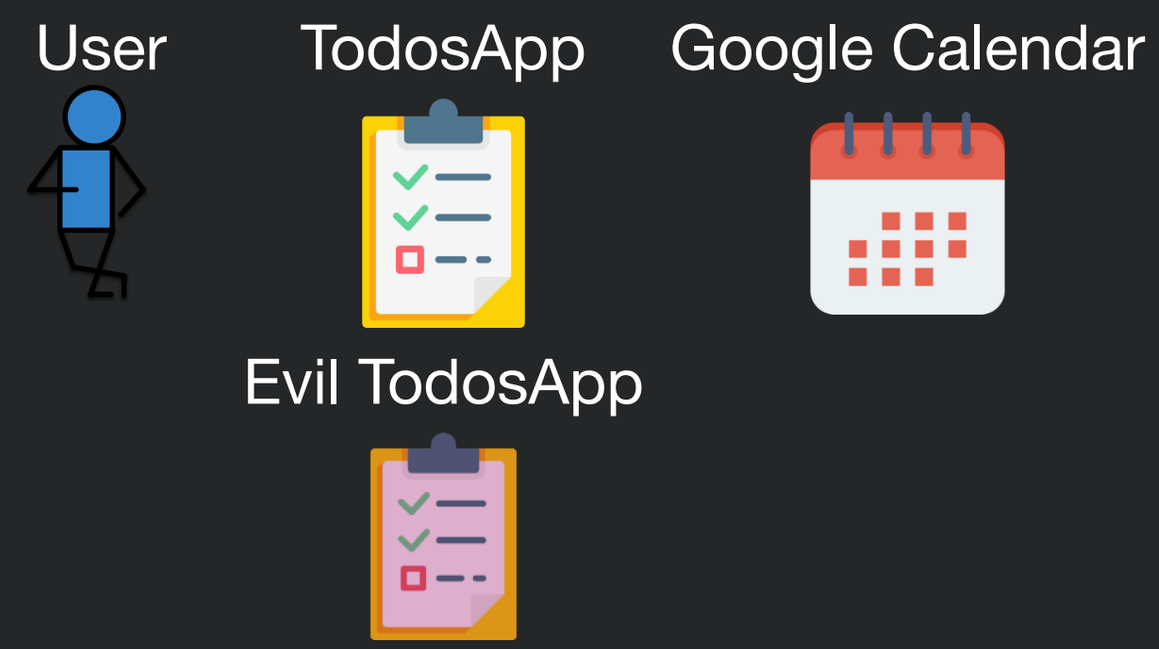


Trust in OAuth



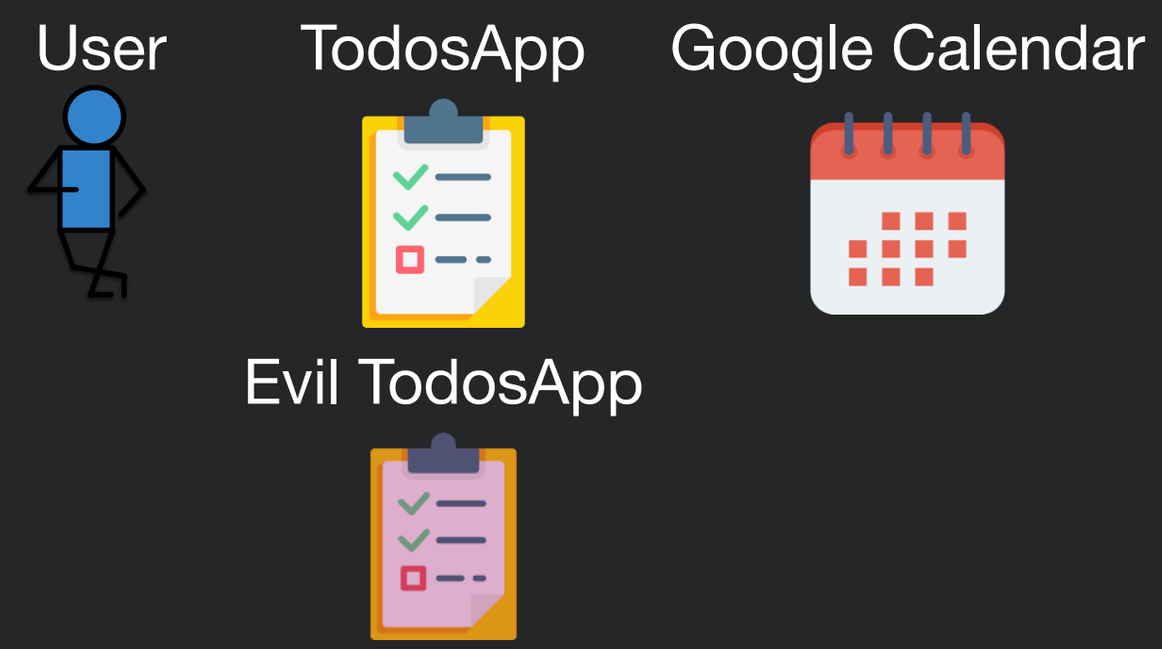
Trust in OAuth

- How does the Service provider (Google calendar) know what the TodosApp is?



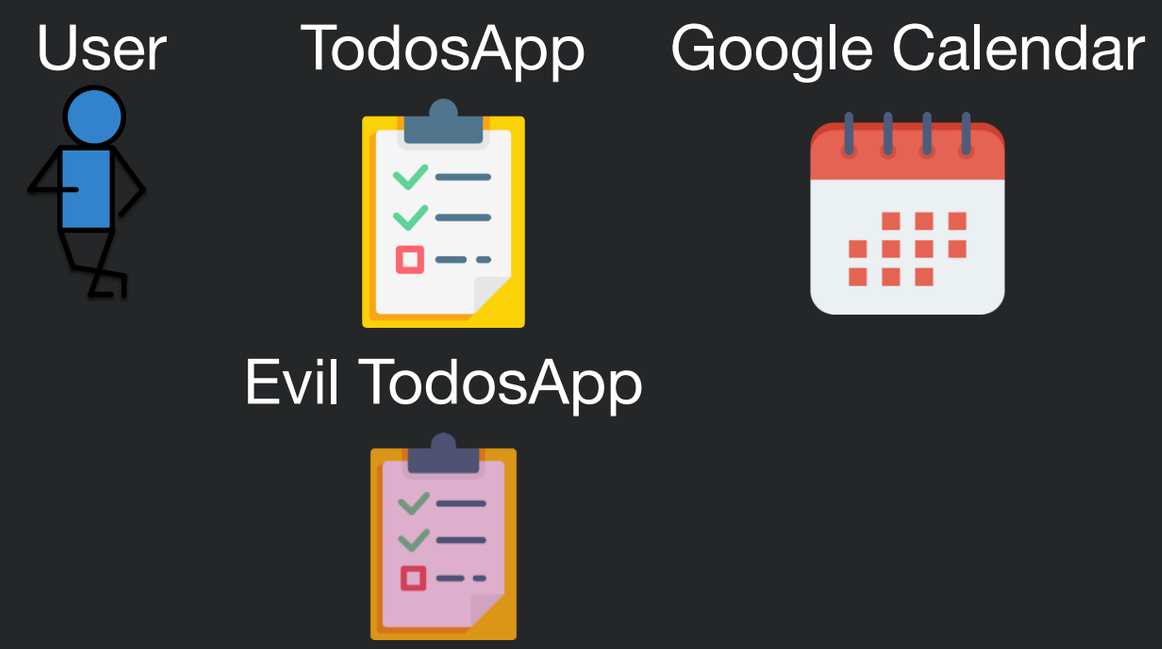
Trust in OAuth

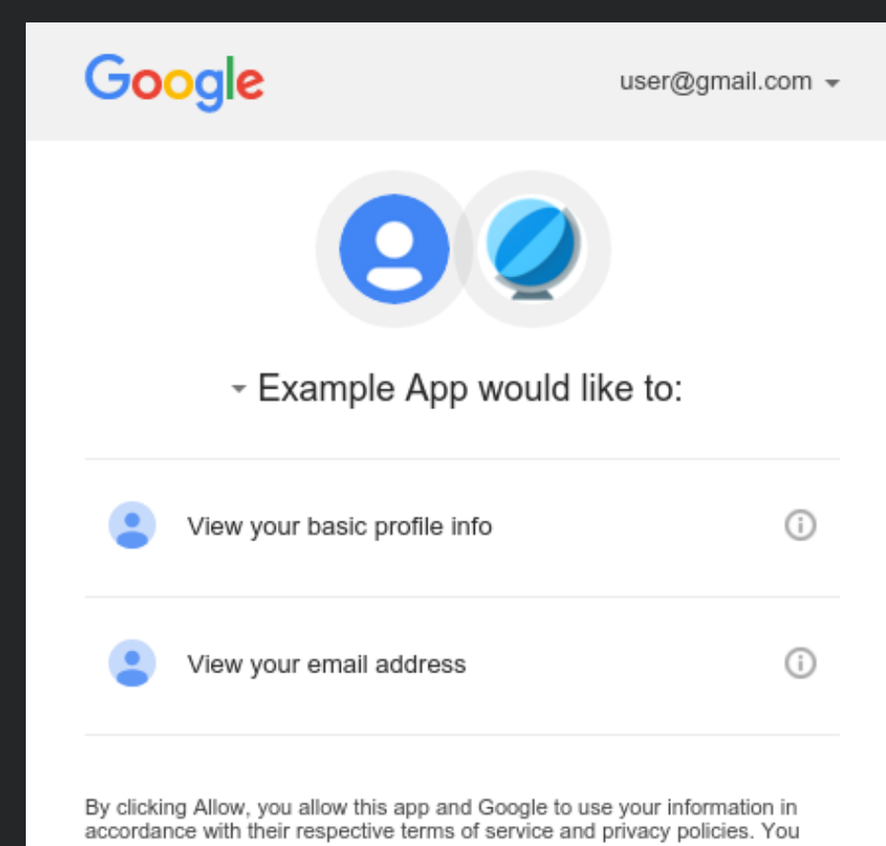
- How does the Service provider (Google calendar) know what the TodosApp is?
- Solution: When you set up OAuth for the first time, you must register your consumer app with the service provider



Trust in OAuth

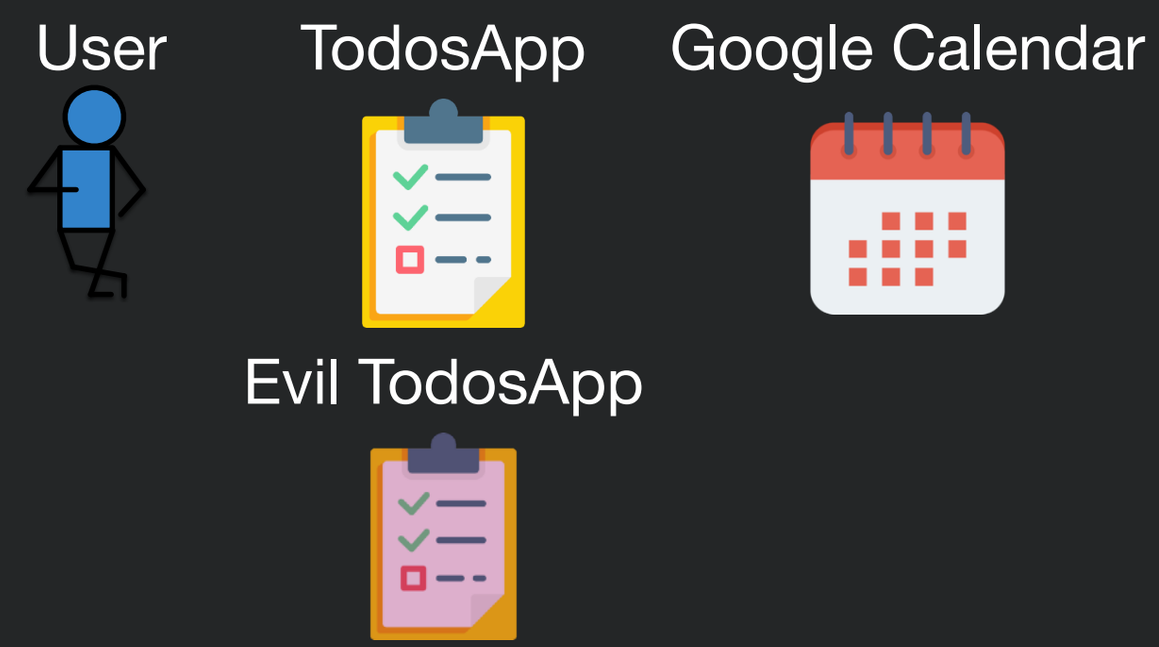
- How does the Service provider (Google calendar) know what the TodosApp is?
- Solution: When you set up OAuth for the first time, you must register your consumer app with the service provider
- Let the user decide

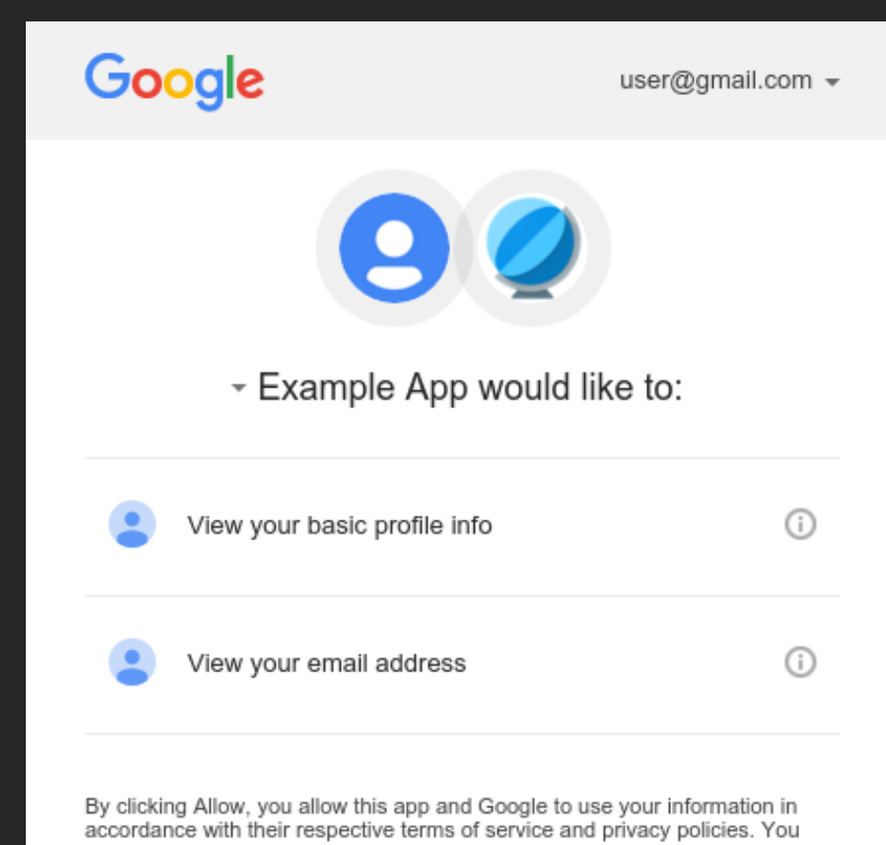




Trust in OAuth

- How does the Service provider (Google calendar) know what the TodosApp is?
- Solution: When you set up OAuth for the first time, you must register your consumer app with the service provider
- Let the user decide
  - … they were the one who clicked the link after all





Authentication as a Service

- Whether we are building “microservices” or not, might make sense to farm out our authentication (user registration/logins) to another service
- Why?
  - Security
  - Reliability
  - Convenience
- We can use OAuth for this!

Authentication: Sharing Data Between Pages

- Browser loads many pages at the same time.
- Might want to share data between pages
  - Popup that wants to show details for data on main page
- Attack: malicious page
  - User visits a malicious page in a second tab
  - Malicious page steals data from page or its data, modifies data, or impersonates user

Solution: Same-Origin Policy

- Browser needs to differentiate pages that are part of same application from unrelated pages
- What makes a page similar to another page?
  - Origin: the **protocol**, **host**, and **port**

    http://www.example.com/dir/page.html
  - Different origins:

    https://www.example.com/dir/page.html

    http://www.example.com:80/dir/page.html

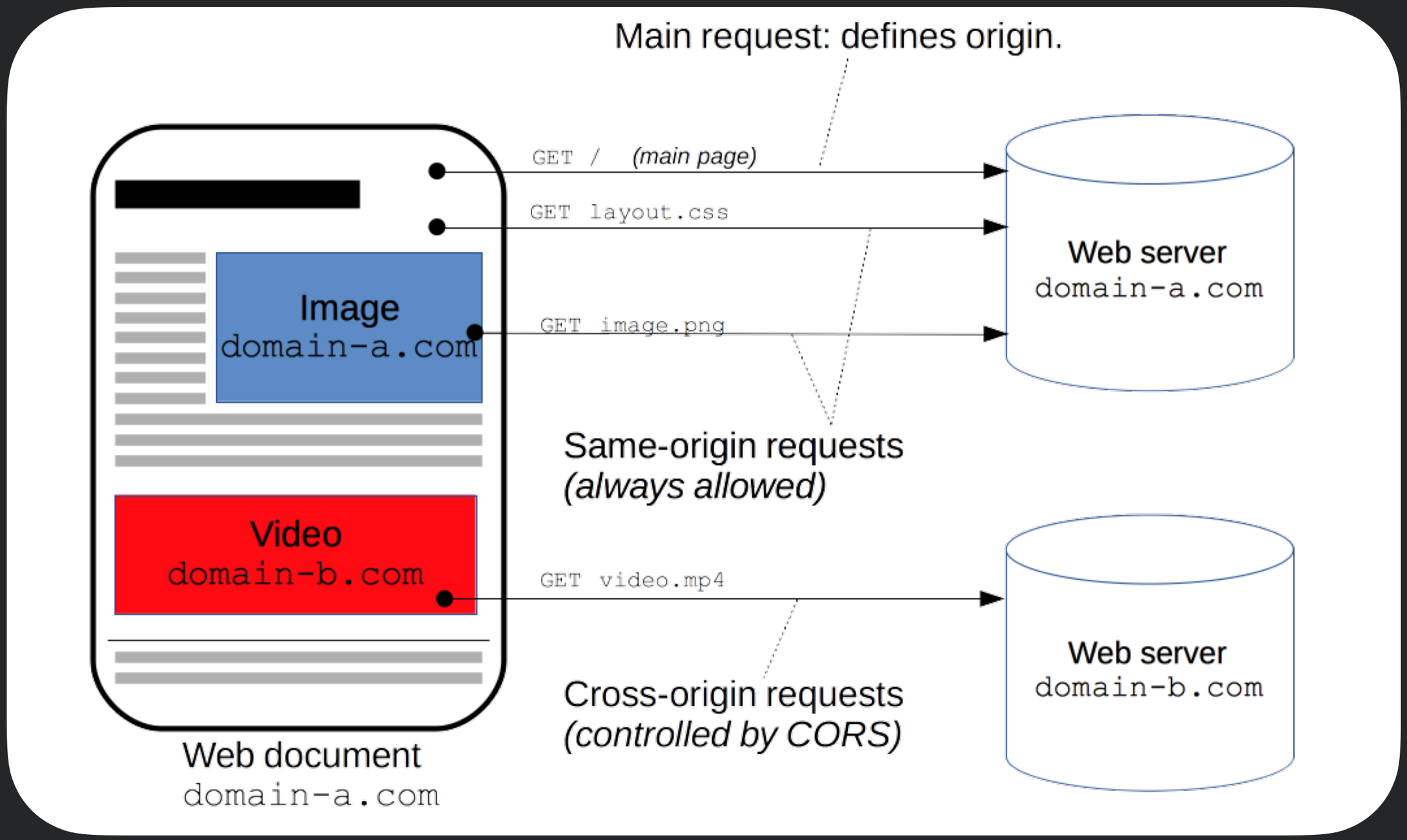
    http://en.example.com:80/dir/page.html

https://en.wikipedia.org/wiki/Same-origin_policy

Same-Origin Policy

- "Origin" refers to the *page that is executing it*, NOT where the data comes from
  - Example:
    - In one HTML file, I directly include 3 JS scripts, each loaded from a different server
    - -> All have same "origin"
  - Example:
    - One of those scripts makes an AJAX call to yet another server
    - -> AJAX call not allowed
- Scripts contained in a page may access data in a second web page (e.g., its DOM) if they come from the same origin

Cross Origin Requests



https://developer.mozilla.org/en-US/docs/Web/HTTP/Access_control_CORS

CORS: Cross Origin Resource Sharing

- Same-Origin might be safer, but not really usable:
  - How do we make AJAX calls to other servers?
- Solution: Cross Origin Resource Sharing (CORS)
- HTTP header:

```
Access-Control-Allow-Origin: <server or wildcard>
```

- In Express:

```
res.header("Access-Control-Allow-Origin", "*");
```

Takeaways

- Think about all potential threat models
  - Which do you care about
  - Which do you not care about
- What user data are you retaining
  - Who are you sharing it with, and what might they do with it

HTML Elements

```
<p lang="en-us">This is a paragraph in English.</p>
```

HTML Elements

This is a paragraph in English.

"Start a paragraph element"

Opening tag begins an HTML element. Opening tags must have a corresponding closing tag.

HTML Elements

This is a paragraph in English.

name value

"Start a paragraph element"

Opening tag begins an HTML element. Opening tags must have a corresponding closing tag.

"Set the language to English"

HTML attributes are name / value pairs that provide additional information about the contents of an element.

HTML Elements

This is a paragraph in English.

name value

"Start a paragraph element"

Opening tag begins an HTML element. Opening tags must have a corresponding closing tag.

"Set the language to English"

HTML attributes are name / value pairs that provide additional information about the contents of an element.

"End a paragraph element"

Closing tag ends an HTML element. All content between the tags and the tags themselves compromise an HTML element.

HTML Elements

<input type="text" />

Some HTML tags can be self closing, including a built-in closing tag.

```
<!-- This is a comment.
Comments can be multiline. -->
```

HTML Elements

<input type="text" />

"Begin and end input element"

Some HTML tags can be self closing, including a built-in closing tag.

```
<!-- This is a comment.
Comments can be multiline. -->
```

A Starter HTML Document

```html
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>Hello World Site</title>
</head>
<body>
    Hello world!
</body>
</html>
```

Hello world!

A Starter HTML Document

"Use HTML5 standards mode"

```html
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>Hello World Site</title>
</head>
<body>
    Hello world!
</body>
</html>
```

Hello world!

A Starter HTML Document

"Use HTML5 standards mode"

"HTML content"

```html
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>Hello World Site</title>
</head>
<body>
    Hello world!
</body>
</html>
```

Hello world!

A Starter HTML Document

"Use HTML5 standards mode"

"HTML content"

"Header"

Information *about* the page

```html
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>Hello World Site</title>
</head>
<body>
    Hello world!
</body>
</html>
```

Hello world!

A Starter HTML Document

"Use HTML5 standards mode"

"HTML content"

"Header"

Information *about* the page

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>Hello World Site</title>
</head>
<body>
    Hello world!
</body>
</html>
```

Hello world!

"Interpret bytes as UTF-8 characters"

Includes both ASCII & international characters.

A Starter HTML Document

"Use HTML5 standards mode"

"HTML content"

"Header"

Information *about* the page

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>Hello World Site</title>
</head>
<body>
    Hello world!
</body>
</html>
```

Hello world!

"Interpret bytes as UTF-8 characters"

Includes both ASCII & international characters.

"Title"

Used by browser for title bar or tab.

A Starter HTML Document

"Use HTML5 standards mode"

"HTML content"

"Header"

Information *about* the page

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8">
    <title>Hello World Site</title>
</head>
<body>
    Hello world!
</body>
</html>
```

Hello world!

"Interpret bytes as UTF-8 characters"

Includes both ASCII & international characters.

"Title"

Used by browser for title bar or tab.

"Document content"

Text

```
9     <h1>Level 1 Heading</h1>
10    <h2>Level 2 Heading</h2>
11    <h3>Level 3 Heading</h3>
12    <h4>Level 4 Heading</h4>
13    <h5>Level 5 Heading</h5>
14    <h6>Level 5 Heading</h6>
15    Text can be made <b>bold</b> and
16        <i>italic</i>, or <sup>super</sup>
17        and <sub>sub</sub>scripts. White
18        space collapsing removes all
19        sequences of two more more spaces
20        and line breaks, allowing
21        the markup to use tabs
22        and whitespace for
23        organization.
24        Spaces can be added with
25            &amp;nbsp;.
26    <br/>New lines can be added with &lt
          ;BR/&gt;.
27
28    <p>A paragraph conssists of one or
          more sentences that form a self
          -contained unit of discourse. By
          default, a browser will show each
          paragraph on a new line.</p>
29
30    <hr/>
31    Text can also be offest with
32    horizontal rules.
33
34
```

Level 1 Heading

Level 2 Heading

Level 3 Heading

Level 4 Heading

Level 5 Heading

Level 5 Heading

Text can be made **bold** and *italic*, or ^{super} and $_{sub}$scripts. White space collapsing removes all sequences of two more more spaces and line breaks, allowing the markup to use tabs and whitespace for organization. Spaces can be added with .
New lines can be added with <BR/>.

A paragraph conssists of one or more sentences that form a self-contained unit of discourse. By default, a browser will show each paragraph on a new line.

Text can also be offest with horizontal rules.

Semantic markup

Semantic markup

- Tags that can be used to denote the ***meaning*** of specific content

Semantic markup

- Tags that can be used to denote the ***meaning*** of specific content
- Examples
  - `<strong>` - An element that has importance.
  - `<blockquote>` - An element that is a longer quote.
  - `<q>` - A shorter quote inline in paragraph.
  - `<abbr>` - Abbreviation
  - `<cite>` - Reference to a work.
  - `<dfn>` - The definition of a term.
  - `<address>` - Contact information.
  - `<ins><del>` - Content that was inserted or deleted.
  - `<s>` - Something that is no longer accurate.

Controls

```
<p>Text Input: <input type="text" maxlength="5" /></p>
<p>Password Input: <input type="password" /></p>
<p>Search Input: <input type="search"></p>
<p>Text Area: <textarea>Initial text</textarea></p>
<p>Checkbox:
    <input type="checkbox" checked="checked" /> Checked
    <input type="checkbox" /> Unchecked
</p>
<p>Drop Down List Box:
    <select>
        <option>Option1</option>
        <option selected="selected">Option2</option>
    </select>
</p>
<p>Multiple Select Box:
    <select multiple="multiple">
        <option>Option1</option>
        <option selected="selected">Option2</option>
    </select>
</p>
<p>File Input Box: <input type="file" />
<p>Image Button: <input type="image" src="http://cs.gmu.edu/~tlatoza
    /images/reachabilityQuestion.jpg" width="50"></p>
<p>Button: <button>Button</button></p>
<p>Range Input: <input type="range" min="0" max="100" step="10"
    value="30" /></p>
```

Text Input:

Password Input:

Search Input:

Text Area: Initial text

Checkbox: Checked Unchecl

Drop Down List Box: Option2

Multiple Select Box: Option1 Option2

File Input Box: Choose File No file ch

Image Button:

Button: Button

Range Input:

Controls

```
<p>Text Input: <input type="text" maxlength="5" /></p>
<p>Password Input: <input type="password" /></p>
<p>Search Input: <input type="search"></p>
<p>Text Area: <textarea>Initial text</textarea></p>
<p>Checkbox:
    <input type="checkbox" checked="checked" /> Checked
    <input type="checkbox" /> Unchecked
</p>
<p>Drop Down List Box:
    <select>
        <option>Option1</option>
        <option selected="selected">Option2</option>
    </select>
</p>
<p>Multiple Select Box:
    <select multiple="multiple">
        <option>Option1</option>
        <option selected="selected">Option2</option>
    </select>
</p>
<p>File Input Box: <input type="file" />
<p>Image Button: <input type="image" src="http://cs.gmu.edu/~tlatoza
    /images/reachabilityQuestion.jpg" width="50"></p>
<p>Button: <button>Button</button></p>
<p>Range Input: <input type="range" min="0" max="100" step="10"
    value="30" /></p>
```

Search input provides clear button

Text Input:

Password Input:

Search Input:

Text Area: Initial text

Checkbox: Checked Unchecl

Drop Down List Box: Option2

Multiple Select Box: Option1 Option2

File Input Box: Choose File No file ch

Image Button:

Button: Button

Range Input:

Block vs. Inline Elements

Block elements

Block elements appear on a new line.
Examples: <h1><p>
<table><form>

Inline elements

Inline elements appear to continue on the same line.
Examples: <a><b><input><img>

Block vs. Inline Elements

Block elements

Block elements appear on a new line.
Examples: <h1><p>
<table><form>

Inline elements

Inline elements appear to continue on the same line.
Examples: <a><b><input><img>

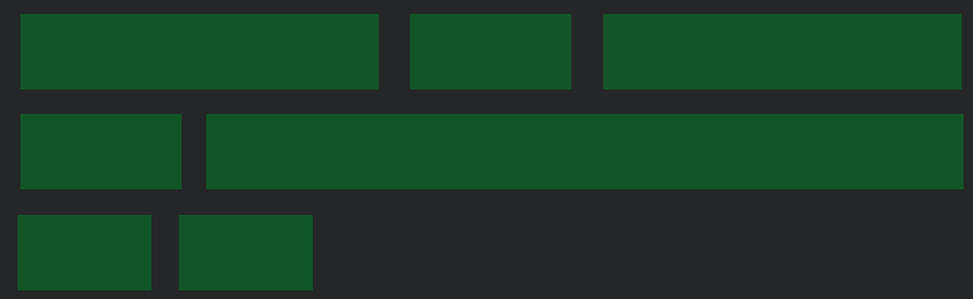
Block vs. Inline Elements

Block elements

Block elements appear on a new line.
Examples: <h1><p>
<table><form>

Inline elements

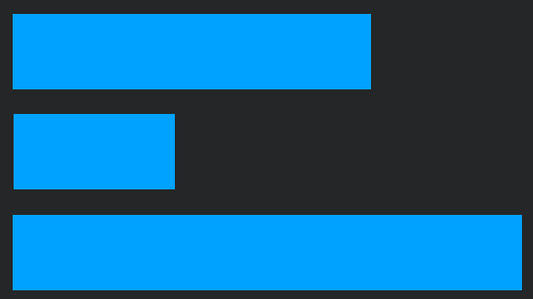
Inline elements appear to continue on the same line.
Examples: <a><b><input><img>



Block vs. Inline Elements

Block elements

Block elements appear on a new line.
Examples: <h1><p>
<table><form>



```
<h1>Hiroshi Sugimoto</h1>
<p>The dates for the ORIGIN OF ART exhbibition are as
    follows:</p>
<ul>
    <li>Science: 21 Nov- 20 Feb 2010/2011</li>
    <li>Architecture: 6 Mar - 15 May 2011</li>
</ul>
```

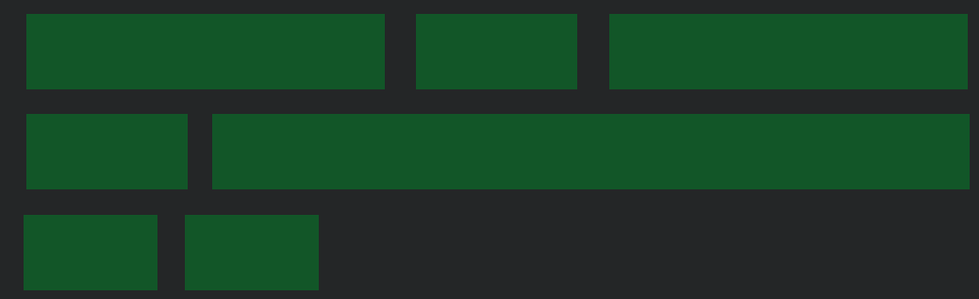
Hiroshi Sugimoto

The dates for the ORIGIN OF ART exhbibition are as follows:

- Science: 21 Nov- 20 Feb 2010/2011
- Architecture: 6 Mar - 15 May 2011

Inline elements

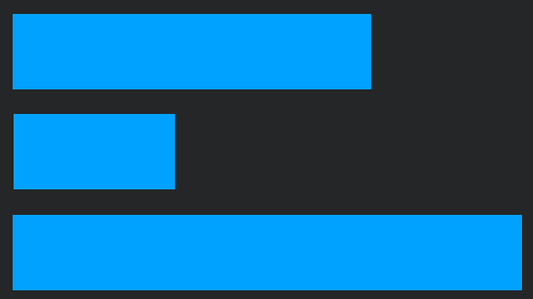
Inline elements appear to continue on the same line.
Examples: <a><b><input><img>



Block vs. Inline Elements

Block elements

Block elements appear on a new line.
Examples: <h1><p>
<table><form>



```
<h1>Hiroshi Sugimoto</h1>
<p>The dates for the ORIGIN OF ART exhbibition are as
    follows:</p>
<ul>
    <li>Science: 21 Nov- 20 Feb 2010/2011</li>
    <li>Architecture: 6 Mar - 15 May 2011</li>
</ul>
```

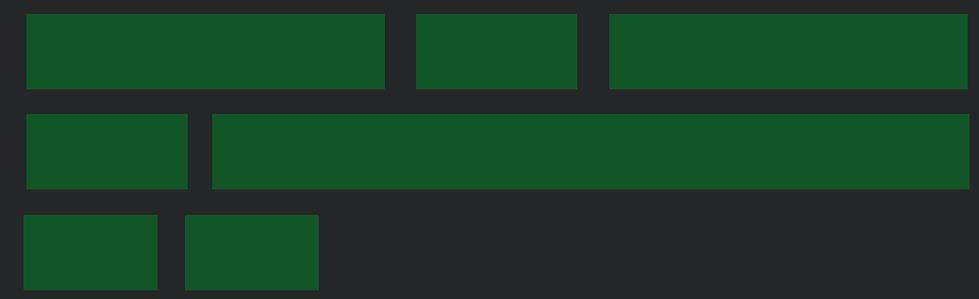
Hiroshi Sugimoto

The dates for the ORIGIN OF ART exhbibition are as follows:

- Science: 21 Nov- 20 Feb 2010/2011
- Architecture: 6 Mar - 15 May 2011

Inline elements

Inline elements appear to continue on the same line.
Examples: <a><b><input><img>



```
Timed to a single revolution of the planet around the sun
at a 23.4 degrees tilt that plays out the rhythm of the
seasons, this <em>Origins of Art</em> cycle is organized
around four themes: <b>science, architecture, history</b>,
and <b>relgion</b>.
```

Timed to a single revolution of the planet around the sun at a 23.4 degrees tilt that plays out the rhythm of the seasons, this *Origins of Art* cycle is organized around four themes: **science, architecture, history**, and **relgion**.

DOM: Document Object Model

- API for interacting with HTML browser
- Contains objects corresponding to every HTML element
- Contains global objects for using other browser features

Reference and tutorials

https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model

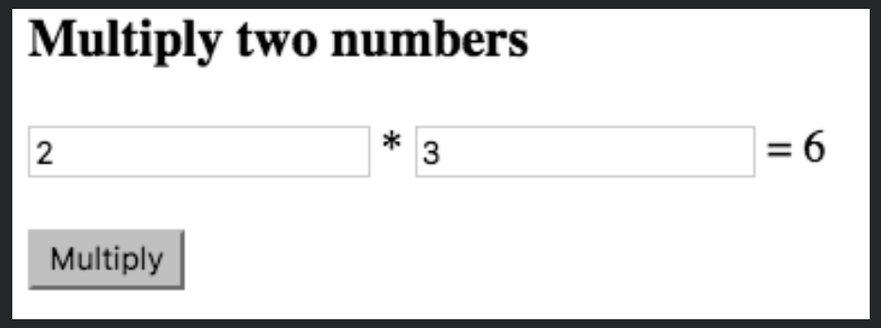
Global DOM objects

- ***window*** - the browser window
  - Has properties for following objects (e.g., window.document)
  - Or can refer to them directly (e.g., document)
- ***document*** - the current web page
- ***history*** - the list of pages the user has visited previously
- ***location*** - URL of current web page
- ***navigator*** - web browser being used
- ***screen*** - the area occupied by the browser & page

DOM Manipulation

- We can also manipulate the DOM directly
- For this class, we will *not* focus on doing this, but will use React instead
- This is how React works though - it manipulates the DOM

DOM Manipulation




```
<h3>Multiply two numbers</h3>
<div>
    <input id="num1" type="number" /> *
    <input id="num2" type="number" /> =
    <span id="product"></span>
    <br/><br/>
    <button id="compute">Multiply</button>
</div>
```

```
document.getElementById('compute')
        .addEventListener("click", multiply);
function multiply()
{
    var x = document.getElementById('num1').value;
    var y = document.getElementById('num2').value;
    var productElem = document.getElementById('product');
    productElem.innerHTML = x * y;
}
```

May choose any event that the compute element produces. May pass the name of a function or define an anonymous function inline.

DOM Manipulation

Multiply two numbers

2 * 3 = 6

Multiply

```
<h3>Multiply two numbers</h3>
<div>
    <input id="num1" type="number" /> *
    <input id="num2" type="number" /> =
    <span id="product"></span>
    <br/><br/>
    <button id="compute">Multiply</button>
</div>
```

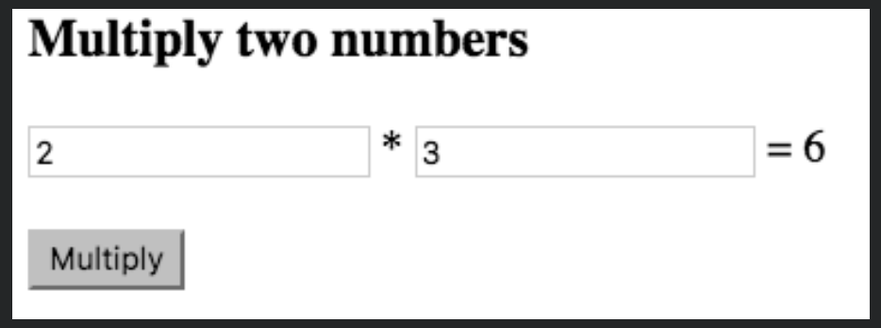
```
document.getElementById('compute')
        .addEventListener("click", multiply);
function multiply()
{
    var x = document.getElementById('num1').value;
    var y = document.getElementById('num2').value;
    var productElem = document.getElementById('product');
    productElem.innerHTML = x * y;
}
```

"Get compute element"

"When compute is clicked, call multiply"

May choose any event that the compute element produces. May pass the name of a function or define an anonymous function inline.

DOM Manipulation

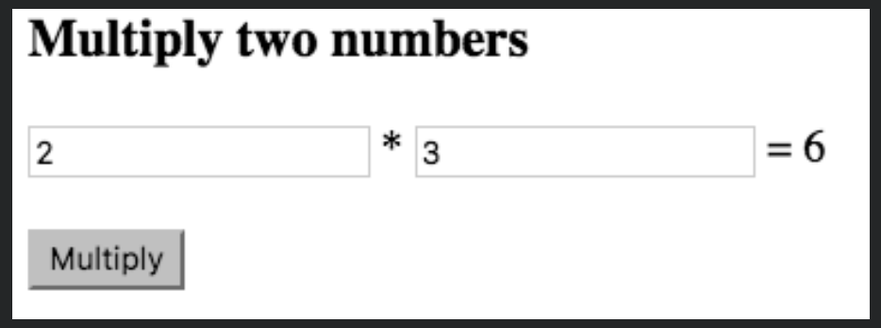



```
<h3>Multiply two numbers</h3>
<div>
    <input id="num1" type="number" /> *
    <input id="num2" type="number" /> =
    <span id="product"></span>
    <br/><br/>
    <button id="compute">Multiply</button>
</div>
```

```
document.getElementById('compute')
        .addEventListener("click", multiply);
function multiply()
{
    var x = document.getElementById('num1').value;
    var y = document.getElementById('num2').value;
    var productElem = document.getElementById('product');
    productElem.innerHTML = x * y;
}
```

Manipulates the DOM by programmatically updating the value of the HTML content. DOM offers accessors for updating all of the DOM state.

DOM Manipulation




```html
<h3>Multiply two numbers</h3>
<div>
    <input id="num1" type="number" /> *
    <input id="num2" type="number" /> =
    <span id="product"></span>
    <br/><br/>
    <button id="compute">Multiply</button>
</div>
```

```javascript
document.getElementById('compute')
        .addEventListener("click", multiply);
function multiply()
{
    var x = document.getElementById('num1').value;
    var y = document.getElementById('num2').value;
    var productElem = document.getElementById('product');
    productElem.innerHTML = x * y;
}
```

"Get the current value of the num1 element"

"Set the HTML between the tags of productElem to the value of x * y"

Manipulates the DOM by programmatically updating the value of the HTML content. DOM offers accessors for updating all of the DOM state.

DOM Manipulation Pattern

- Wait for some event
  - click, hover, focus, keypress, …
- Do some computation
  - Read data from event, controls, and/or previous application state
  - Update application state based on what happened
- Update the DOM
  - Generate HTML based on new application state
- Also: JQuery

Examples of events

- ***Form element events***
  - change, focus, blur
- ***Network events***
  - online, offline
- ***View events***
  - resize, scroll
- ***Clipboard events***
  - cut, copy, paste
- ***Keyboard events***
  - keydown, keypress, keypup
- ***Mouse events***
  - mouseenter, mouseleave, mousemove, mousedown, mouseup, click, dblclick, select

List of events: https://www.w3.org/TR/DOM-Level-3-Events/