

# SWE 432 -Web Application Development

Fall 2022

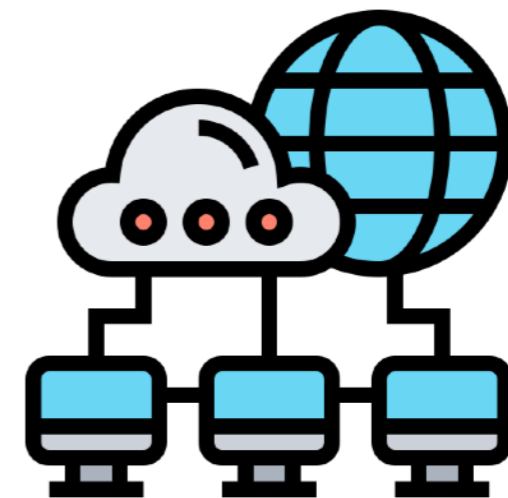


George Mason  
University

---

Dr. Kevin Moran

## Week 4: Asynchronous Programming II





# Administrivia

- *HW Assignment 1* - Due Today Before Class
- *HW Assignment 2* - Out today, will discuss next week (but feel free to get started)

# Class Overview





# Class Overview

- (Last Week) Part 1 - Asynchronous

Programming I: Communicating between web app components

- (Today) - Part 2 - Asynchronous Programming

II: More communication strategies

# Asynchronous Programming II





# Review: Asynchronous

- Synchronous:
  - Make a function call
  - When function call returns, the work is done
- Asynchronous:
  - Make a function call
  - Function returns immediately, before completing work!



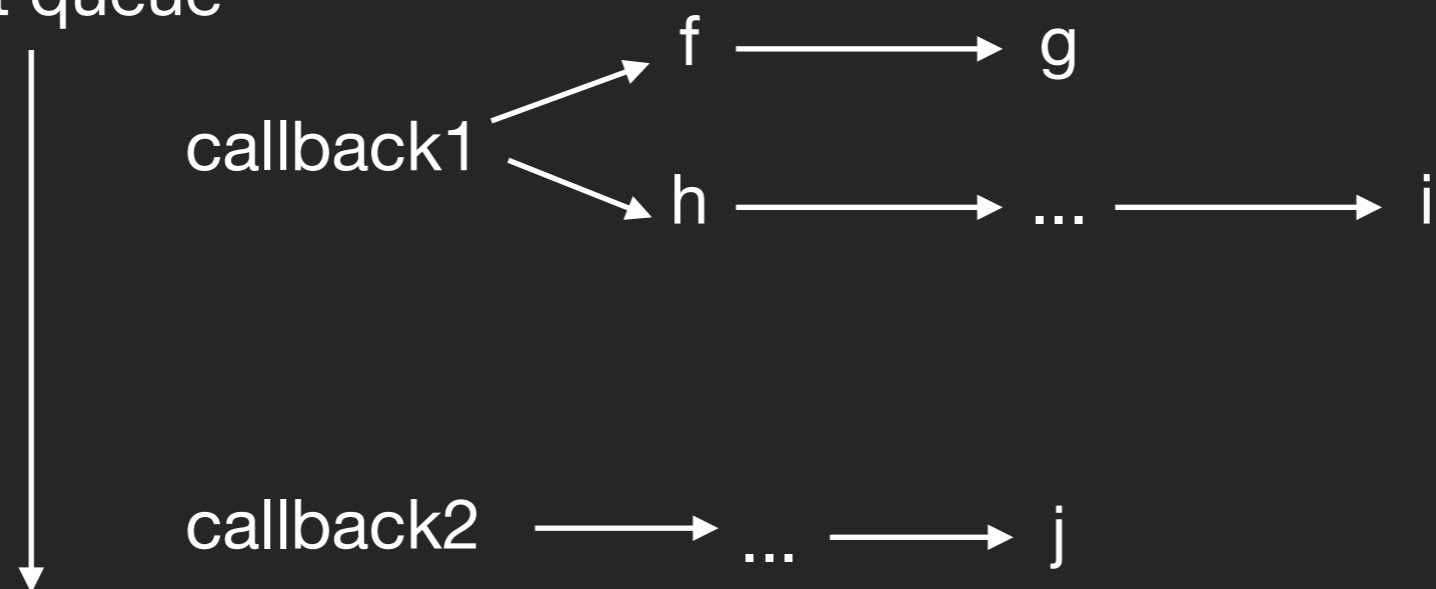
# Review: Asynchronous

- How we do multiple things at a time in JS
- NodeJS magically handles these asynchronous things in the background
- Really important when doing file/network input/output

# Review: 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.

processing of  
event queue



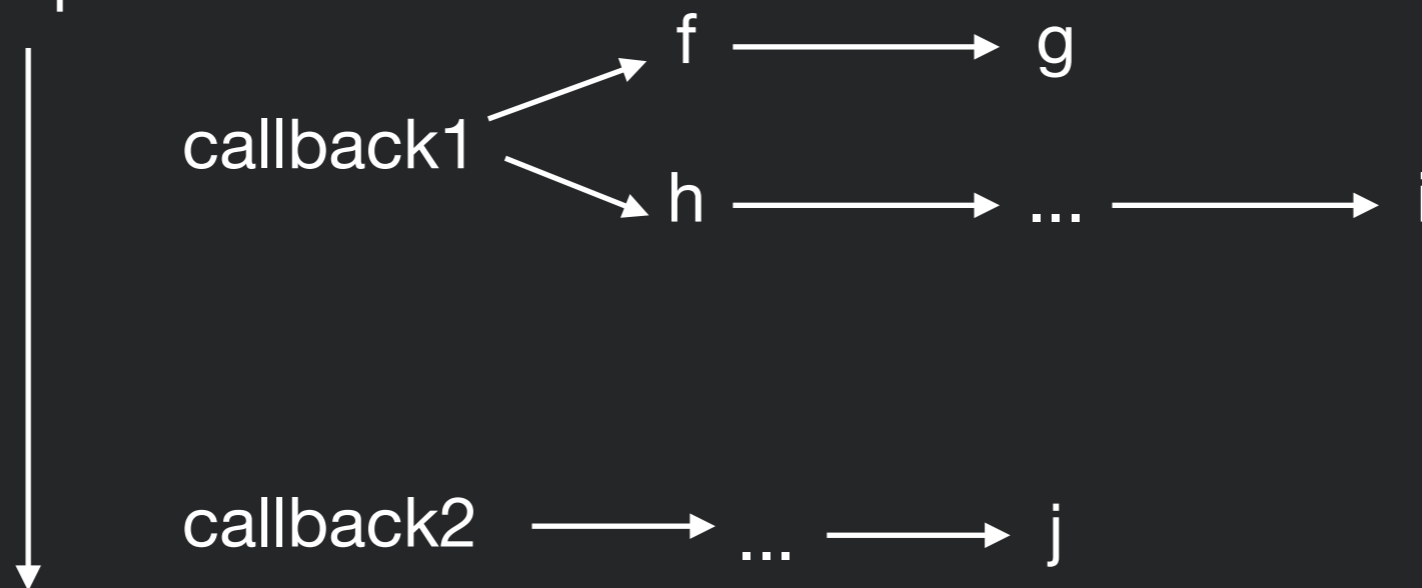




# Review: Implications of run-to-completion

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

processing of  
event queue



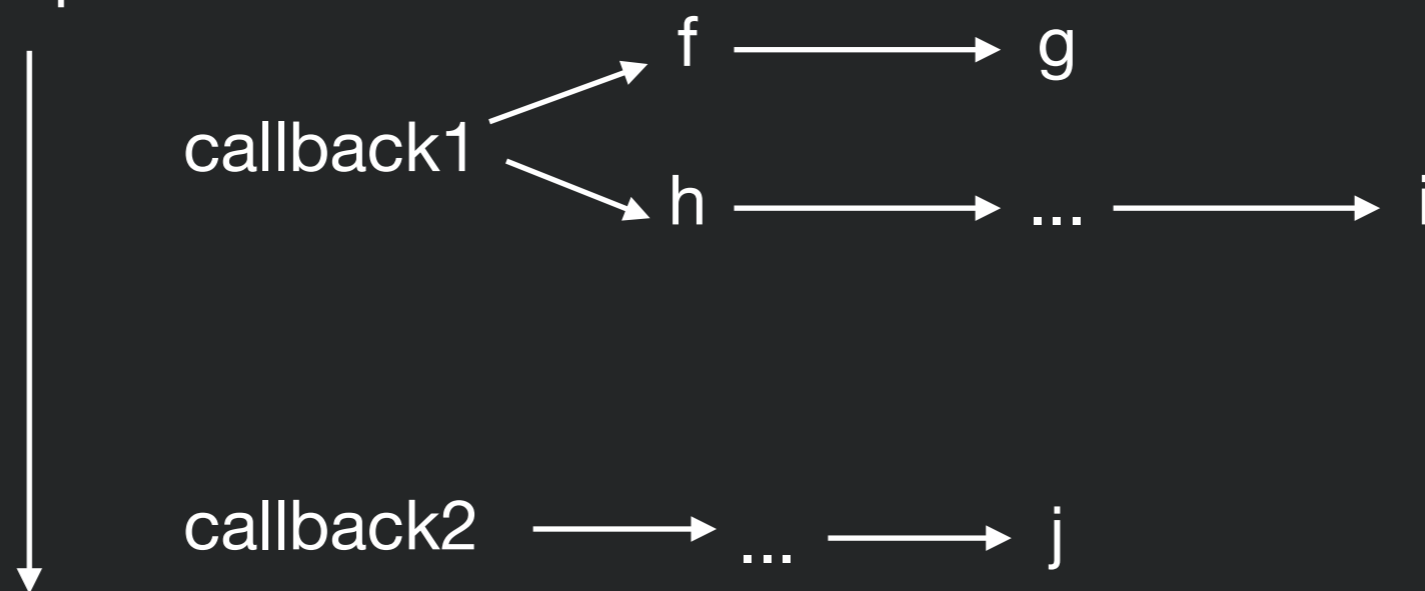
*j will not execute until after i*



# Review: 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

processing of  
event queue



*j will not execute until i finishes*



# Review: 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){  
  
});
```



# Current Lecture

- Async/await
- Programming activity

# Promising many things

- Can also specify that \*many\* things should be done, and then something else
- Example: load a whole bunch of images at once:

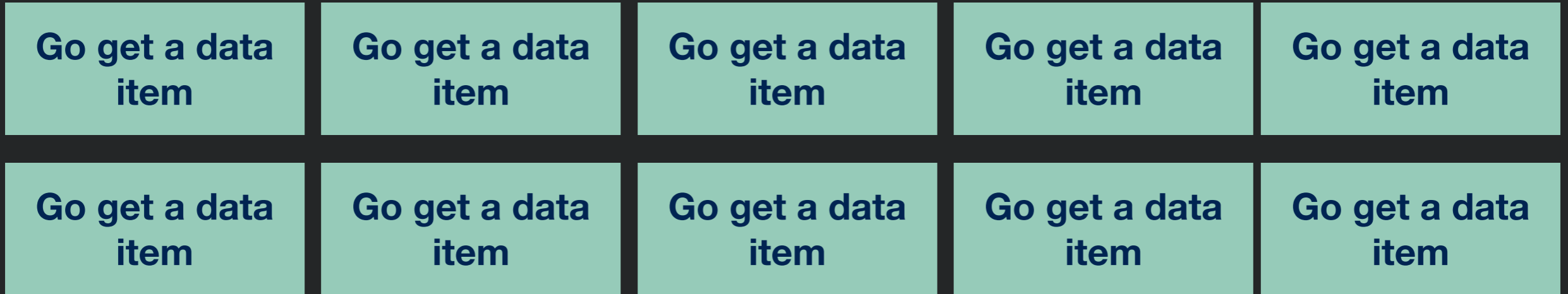
Promise

```
.all([loadImage("GMURGB.jpg"), loadImage("CS.jpg")])  
.then(function (imgArray) {  
    imgArray.forEach(img => {document.body.appendChild(img)})  
})  
.catch(function (e) {  
    console.log("Oops");  
    console.log(e);  
});
```



# Async Programming Example

1 second each

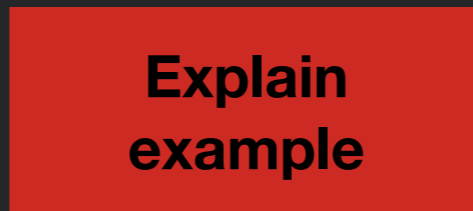
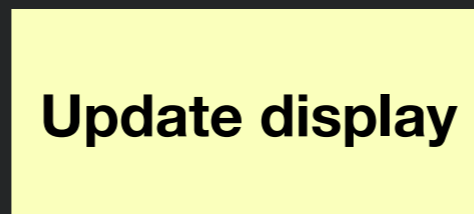


2 seconds each

thenCombine



when done



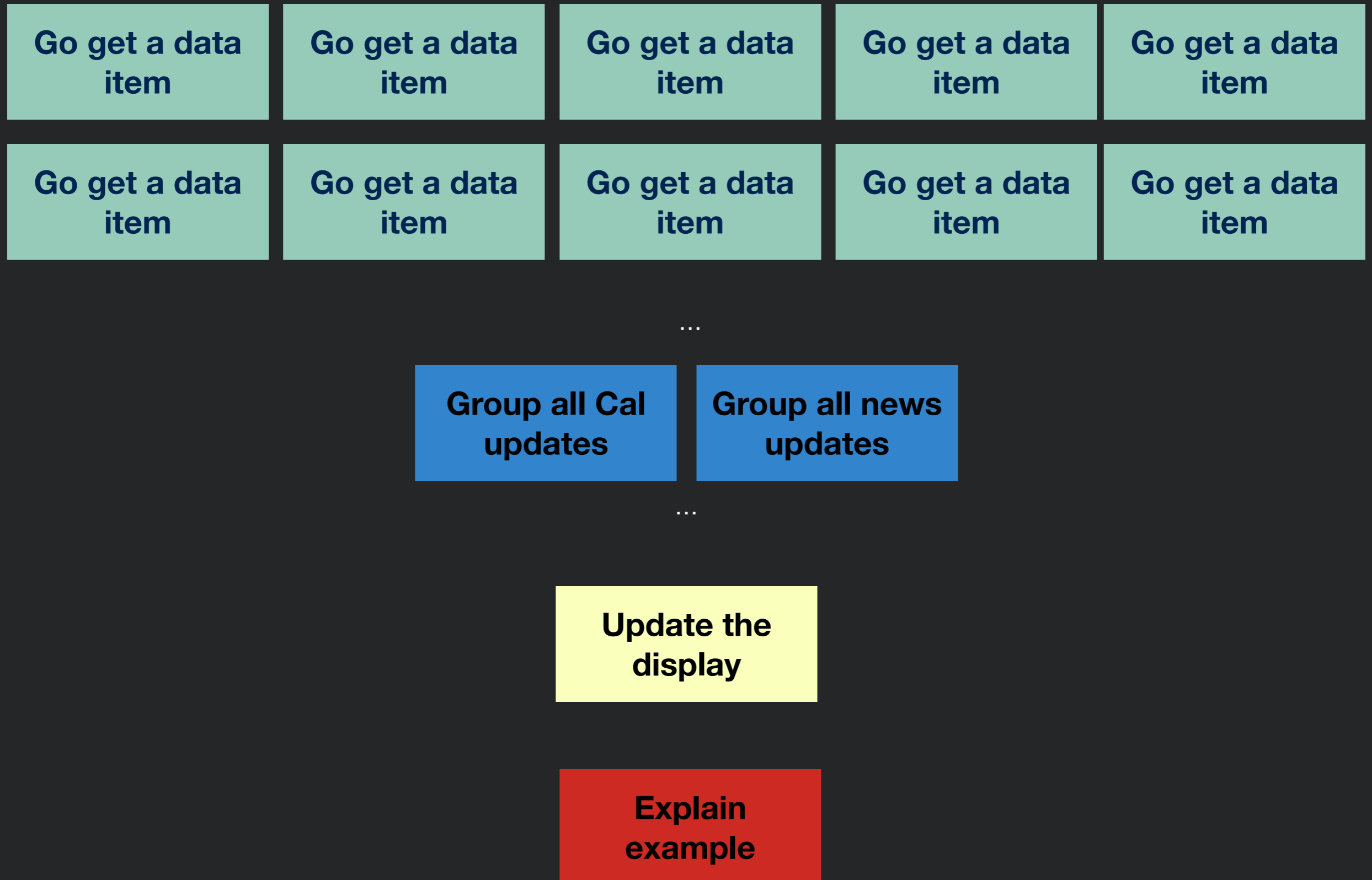


# Synchronous Version





# Asynchronous Version







# Async Programming Example (Sync)

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

let thingsToFetch = ['t1', 't2', 't3', 's1', 's2',
  's3', 'm1', 'm2', 'm3', 't4'];
let stuff = [];
for(let thingToGet of thingsToFetch)
{
  stuff.push(lib.getSync(thingToGet));
  console.log("Got a thing");
}
//Got all my stuff
let ts = lib.groupSync(stuff, "t");
console.log("Grouped");
let ms = lib.groupSync(stuff, "m");
console.log("Grouped");
let ss = lib.groupSync(stuff, "s");
console.log("Grouped");

console.log("Done");
```

```
node v12.16.1
```

```
█
```



# Async Programming Example (Callbacks, no parallelism)

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

let thingsToFetch = ['t1', 't2', 't3', 's1', 's2', 's3', 'm1', 'm2', 'm3', 't4'];
let stuff = [];
let ts, ms, ss;
let outstandingStuffToGet = thingsToFetch.length;

lib.getASync(thingsToFetch[0], (v) => {
  stuff.push(v);
  console.log("Got a thing")
  lib.getASync(thingsToFetch[1], (v) => {
    stuff.push(v);
    console.log("Got a thing")
    lib.getASync(thingsToFetch[2], (v) => {
      stuff.push(v);
      console.log("Got a thing")
      lib.getASync(thingsToFetch[3], (v) => {
        stuff.push(v);
        console.log("Got a thing")
        lib.getASync(thingsToFetch[4], (v) => {
          stuff.push(v);
          console.log("Got a thing")
          lib.getASync(thingsToFetch[5], (v) => {
            stuff.push(v);
            console.log("Got a thing")
            lib.getASync(thingsToFetch[6], (v) => {
              stuff.push(v);
              console.log("Got a thing")
              lib.getASync(thingsToFetch[7], (v) => {
                stuff.push(v);
                console.log("Got a thing")
                lib.getASync(thingsToFetch[8], (v) => {
                  stuff.push(v);
                  console.log("Got a thing")
                  lib.getASync(thingsToFetch[9], (v) => {
                    stuff.push(v);
                    console.log("Got a thing")
                    lib.groupAsync(stuff, "t", (t) => {
                      ts = t;
                      console.log("Grouped");
                      lib.groupAsync(stuff, "m", (m) => {
                        ss = s;
                        console.log("Grouped");
```

```
node v12.16.1
```

```
█
```



# Async Programming Example (Callbacks)

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

let thingsToFetch = ['t1', 't2', 't3', 's1', 's2', 's3', 'm1', 'm2', 'm3', 't4'];
let stuff = [];
let ts, ms, ss;
let outstandingStuffToGet = thingsToFetch.length;
for (let thingToGet of thingsToFetch) {
  lib.getAsync(thingToGet, (v) => {
    stuff.push(v);
    console.log("Got a thing")
    outstandingStuffToGet--;
    if (outstandingStuffToGet == 0) {
      let groupsOfStuffToGetStill = 3;
      lib.groupAsync(stuff, "t", (t) => {
        ts = t;
        console.log("Grouped");
        groupsOfStuffToGetStill--;
        if (groupsOfStuffToGetStill == 0)
          console.log("Done");
      });
      lib.groupAsync(stuff, "m", (m) => {
        ms = m;
        console.log("Grouped");
        groupsOfStuffToGetStill--;
        if (groupsOfStuffToGetStill == 0)
          console.log("Done");
      });
      lib.groupAsync(stuff, "s", (s) => {
        ss = s;
        console.log("Grouped");
        groupsOfStuffToGetStill--;
        if (groupsOfStuffToGetStill == 0)
          console.log("Done");
      });
    }
  });
}
```

node v12.16.1

```
[ ]
```



# Async Programming Example (Promises, no parallelism)

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

let thingsToFetch = ['t1', 't2', 't3', 's1', 's2', 's3', 'm1', 'm2', 'm3', 't4'];
let stuff = [];
let ts, ms, ss;
let outstandingStuffToGet = thingsToFetch.length;
lib.getPromise(thingsToFetch[0]).then(
  (v)=>{
    stuff.push(v);
    console.log("Got a thing");
    return lib.getPromise(thingsToFetch[1]);
  }
).then(
  (v)=>{
    stuff.push(v);
    console.log("Got a thing");
    return lib.getPromise(thingsToFetch[1]);
  }
).then(
  (v)=>{
    stuff.push(v);
    console.log("Got a thing");
    return lib.getPromise(thingsToFetch[1]);
  }
).then(
  (v)=>{
    stuff.push(v);
    console.log("Got a thing");
    return lib.getPromise(thingsToFetch[2]);
  }
).then(
  (v)=>{
    stuff.push(v);
    console.log("Got a thing");
    return lib.getPromise(thingsToFetch[3]);
  }
).then(
  (v)=>{
    stuff.push(v);
    console.log("Got a thing");
    return lib.getPromise(thingsToFetch[4]);
  }
);
```

```
node v12.16.1
```

```
█
```



# Async Programming Example (Promises)

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

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) {
  promises.push(lib.getPromise(thingToGet));
}
Promise.all(promises).then((data) => {
  console.log("Got all things");
  stuff = data;
  return Promise.all([
    lib.groupPromise(stuff, "t"),
    lib.groupPromise(stuff, "m"),
    lib.groupPromise(stuff, "s")
  ])
})
  .then((groups) => {
    console.log("Got all groups");
    ts = groups[0];
    ms = groups[1];
    ss = groups[2];
    console.log("Done");
  });
```

```
node v12.16.1
█
```



# Problems with Promises

```
const makeRequest = () => {
  try {
    return promise1()
      .then(value1 => {
        // do something
      }).catch(err => {
        //This is the only way to catch async errors
        console.log(err);
      })
  } catch(ex) {
    //Will never catch async errors!!
  }
}
```



# 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();
```

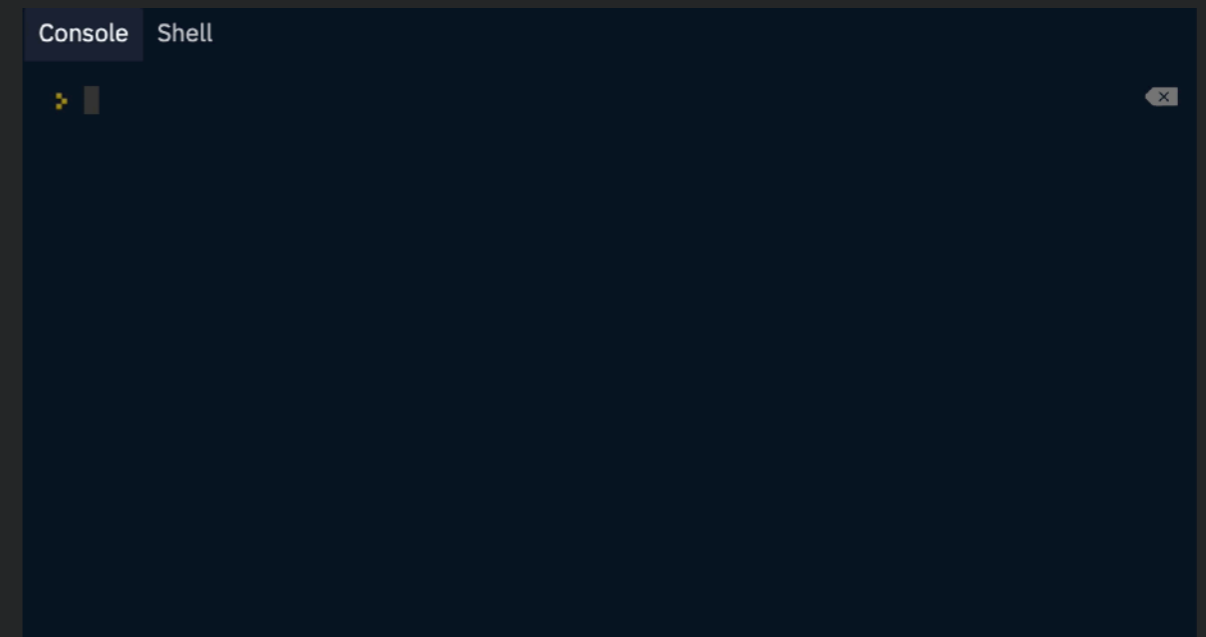
- Automatically turns the return type into a Promise





# 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 -> Synchronous

```
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");  
  ...  
}
```



# Async/Await Activity

Rewrite this code so that all of the things are fetched (in parallel) and then all of the groups are collected using async/await

```
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();
```

<https://replit.com/@kmoran/SWE-Week-3-Activity#index.js>

I will also post to Ed right now!



# Acknowledgements

Slides adapted from Dr. Thomas LaToza's  
SWE 632 course