

SWE 432 -Web Application Development

Fall 2021



George Mason
University

Dr. Kevin Moran

Week 7: Web App Deployment





Administrivia

- Midterm - Congrats on Finishing! We will have the Midterm grades back to you within the next week.
- HW Assignment 3 - Out now, Due October 26th, before class
 - Please accept the Assignment on GitHub Classroom!!



HW Assignment 3

Step 1: Sign up on GitHub Classroom to Clone the Starter Project

Please follow the instructions for setting up this homework assignment in GitHub Classroom and deployment of your project via Heroku. The starter project includes code for both a React Front-End and an Express back-end. You may reuse your code from HW2 for your backend to satisfy requirements for this assignment.

[Click Here to View HW 3 Tutorial](#)

Step 2: Use Persistence to Store Data Gathered through Fetch

In your node backend, you will again gather data from an external API through fetch. In this HW, rather than store data as a local variable in memory, you will instead use Firebase to persist data. You should take data generated by interacting with your 3rd party API, store the data in Firebase, and then retrieve this data later in order to handle requests made to your microservice.



HW Assignment 3

Step 3: Using React as a Template Engine to Display Data

In this step, you will build a simple frontend to display data from your microservice. Your frontend will be organized into several React components.

For example, for a cities web app, you might have an index.html page that displays a list of all cities and their names. Clicking on a single city loads a second view displaying data for that city. Clicking a link for weather on that page loads a third view displaying current weather data for that city.

Requirements:

- Use fetch to retrieve a dataset from a remote web service.
 - Data should be persisted (see Persistence below) so that the same data is only retrieved from the remote web service once during the lifetime of your microservice.
 - You should handle at least one potential error generated by the third-party API.
- Endpoints
 - Include at least 2 GET endpoints.
- Persistence
 - Within your node backend, ensure that all state which is reused across different HTTP requests is persisted into a Firebase datastore.
 - Within your node backend, retrieve data from your Firebase datastore to handle client requests.
- HTML
 - Create at least 3 separate React components corresponding to different views. These may be structured as a single page or as 3 separate pages.
 - Use at least 3 different semantic markup elements, for example:

```
1 | <nav><article><aside><section><figcaption><address><cite><strong><abbr>
```



- React
 - On each of 3 views, after the page loads, use fetch to retrieve appropriate data from your backend and then use React to generate HTML for this data.



Class Overview

- Big picture: from ideas to great products
 - How do we structure the process that gets us those products?
- Buzzwords:
 - DevOps, Continuous Integration, Continuous Deployment, Continuous Delivery, and how we got there
- No specific technologies!

For further reading:

[Chuck Rossi \(Facebook\) on Continuous Mobile Release](#)

<http://blog.christianposta.com/deploy/blue-green-deployments-a-b-testing-and-canary-releases/>



What is a software process?

- A structured set of activities required to develop a software product
 - Specification
 - Design and implementation
 - Validation
 - Evolution (operation and maintenance)
- Goal: Minimize Risk
 - Falling behind schedule
 - Changes to requirements
 - Bugs/unintended effects of changes



Software Design & Implementation

- The process of converting the system specification into an executable system.
- Software design
 - Design a software structure that realizes the specification;
- Implementation
 - Translate this structure into an executable program;
 - The activities of design and implementation are closely related and may be inter-leaved.



Software Validation

- Verification and validation (V & V) is intended to show that a system conforms to its specification and meets the requirements of the customer(s).
- Involves checking and review processes, and acceptance or beta testing.
- Custom software: Acceptance testing involves executing the system with test cases that are derived from the real data to be processed by the system in the customer's environment.
- Generic software: Beta testing executes the system in many customers' environments under real use.



Software Evolution

- Software is inherently flexible and can change.
- As requirements change due to changing business circumstances, the software that supports the business must also evolve and change.
- Although there has historically been a demarcation between development and evolution, this is increasingly irrelevant as fewer and fewer systems are completely new.

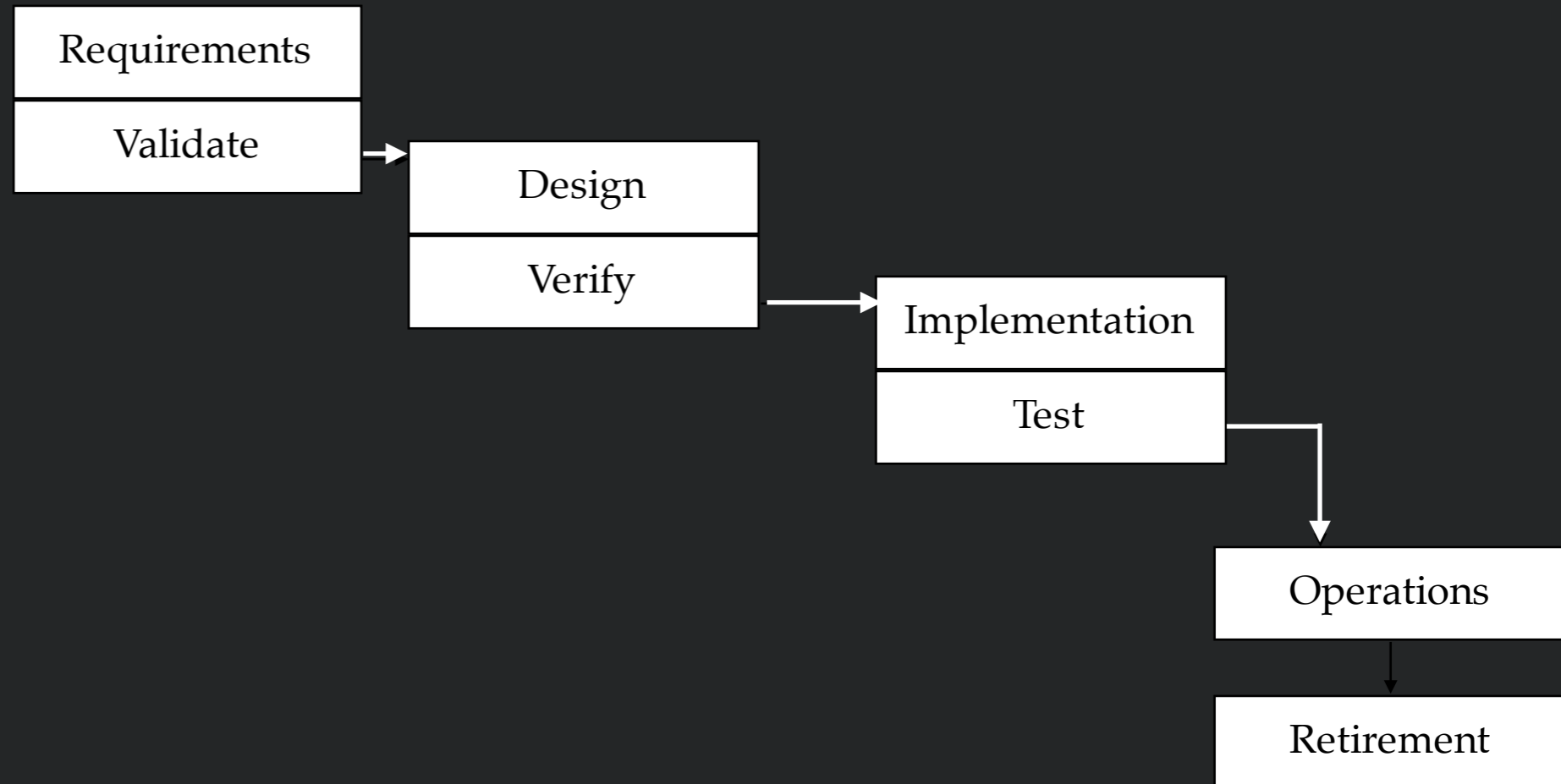


Process Models

- If we say that building software requires:
 - Specification
 - Design/Implementation
 - Validation
 - Evolution
- How do we structure our organization/development teams/tasks to do this most efficiently?



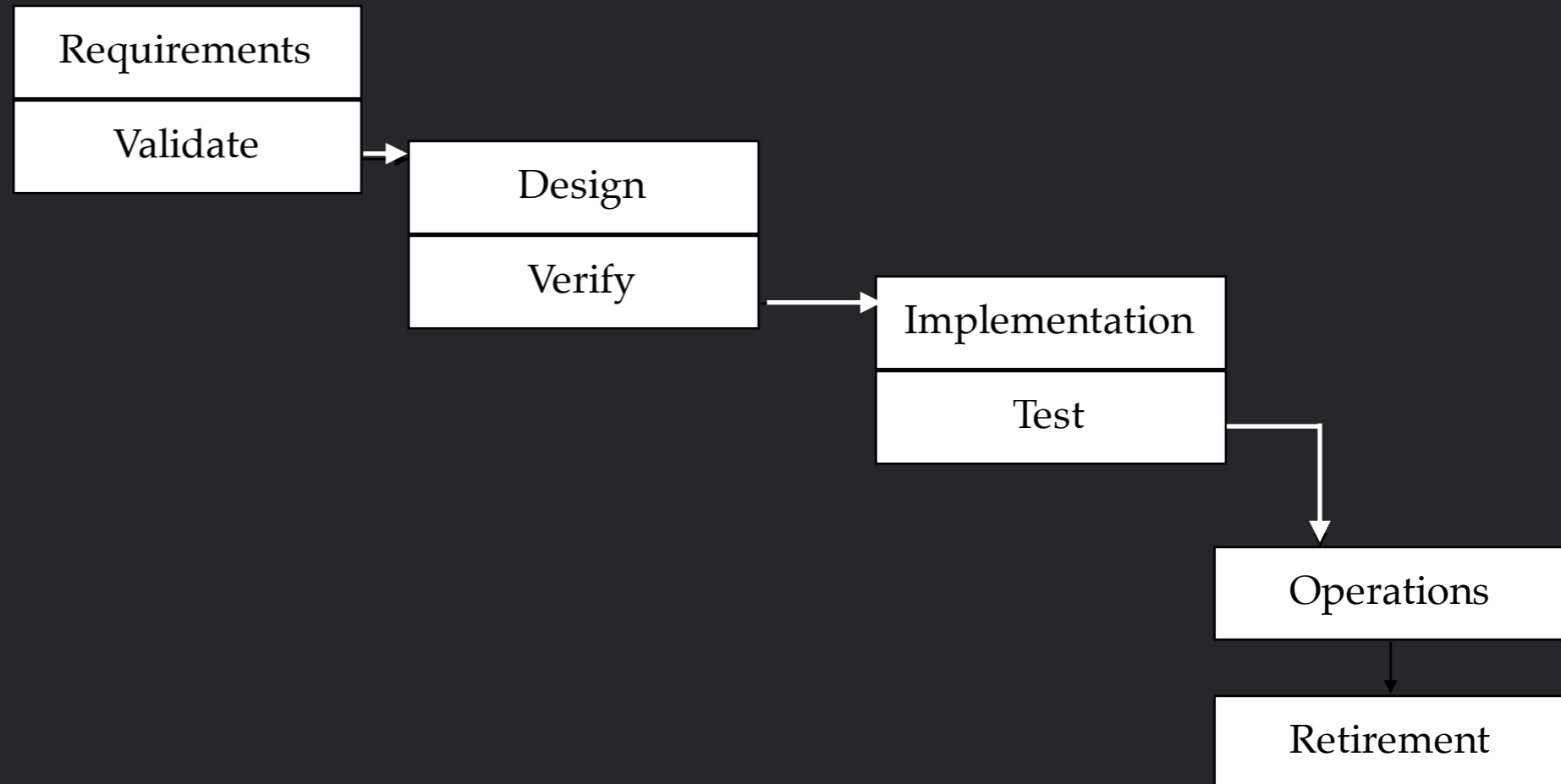
Waterfall Model





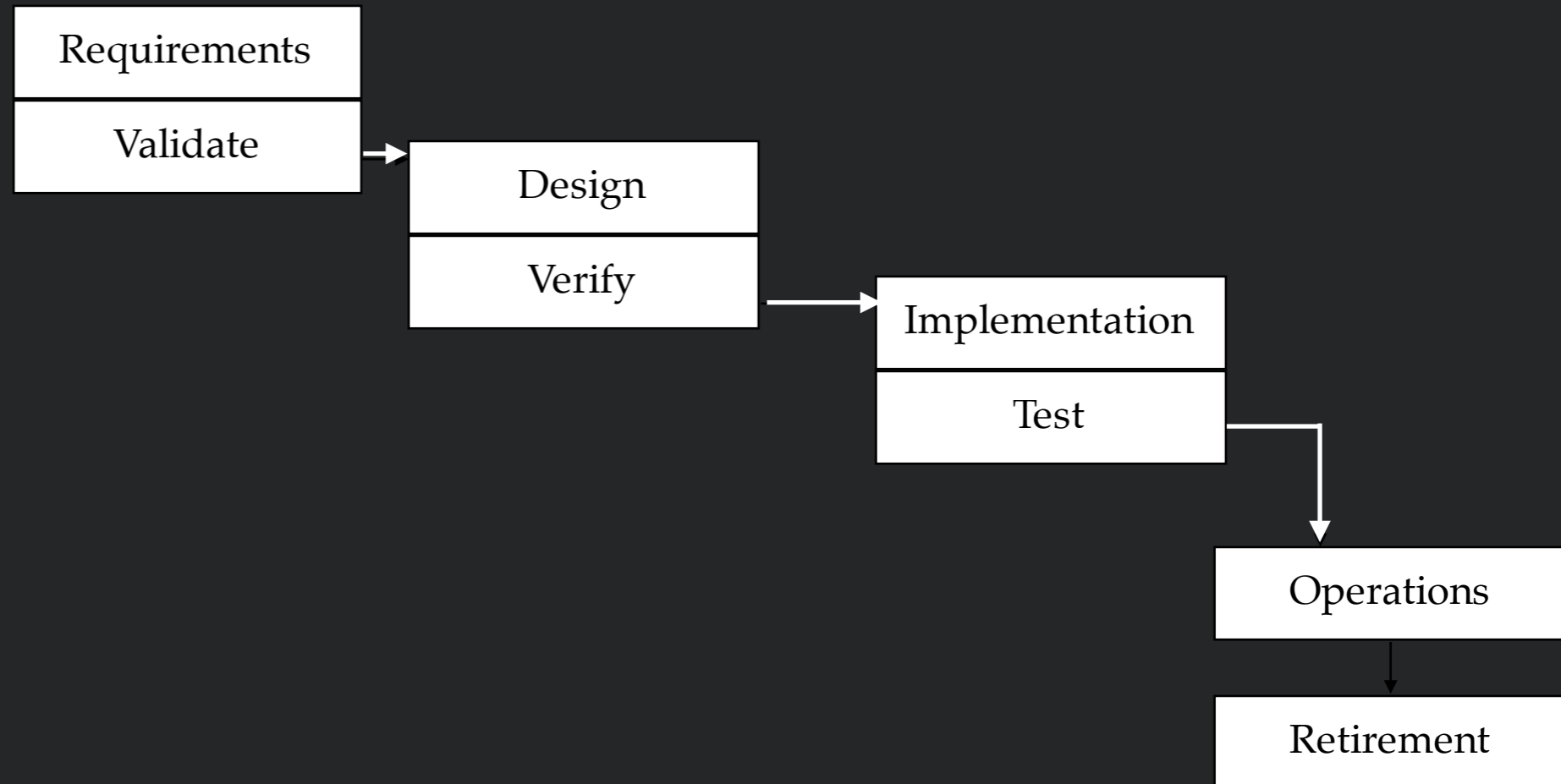
Waterfall Model

- Widely used today



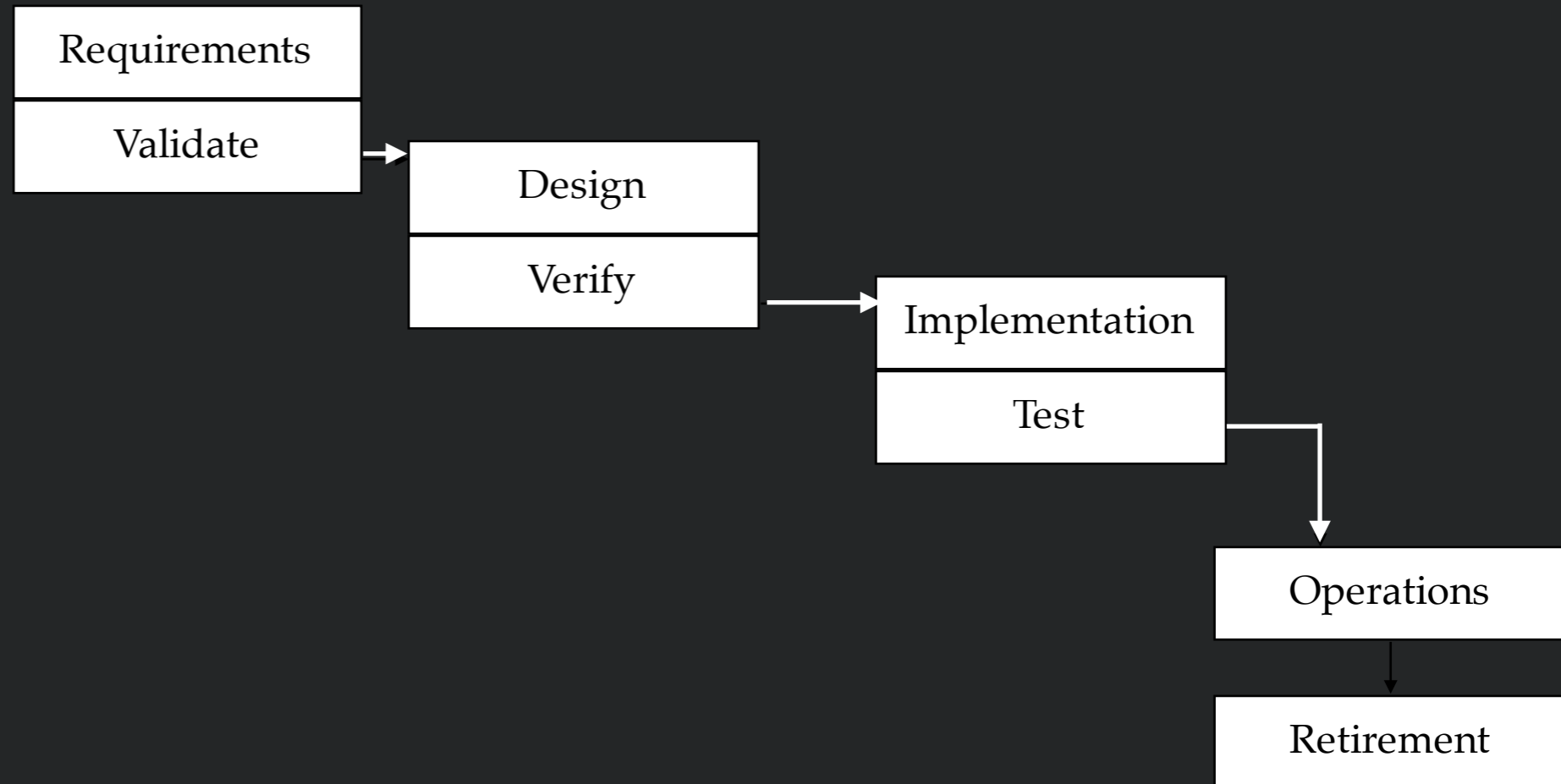
Waterfall Model

- Widely used today
- Advantages



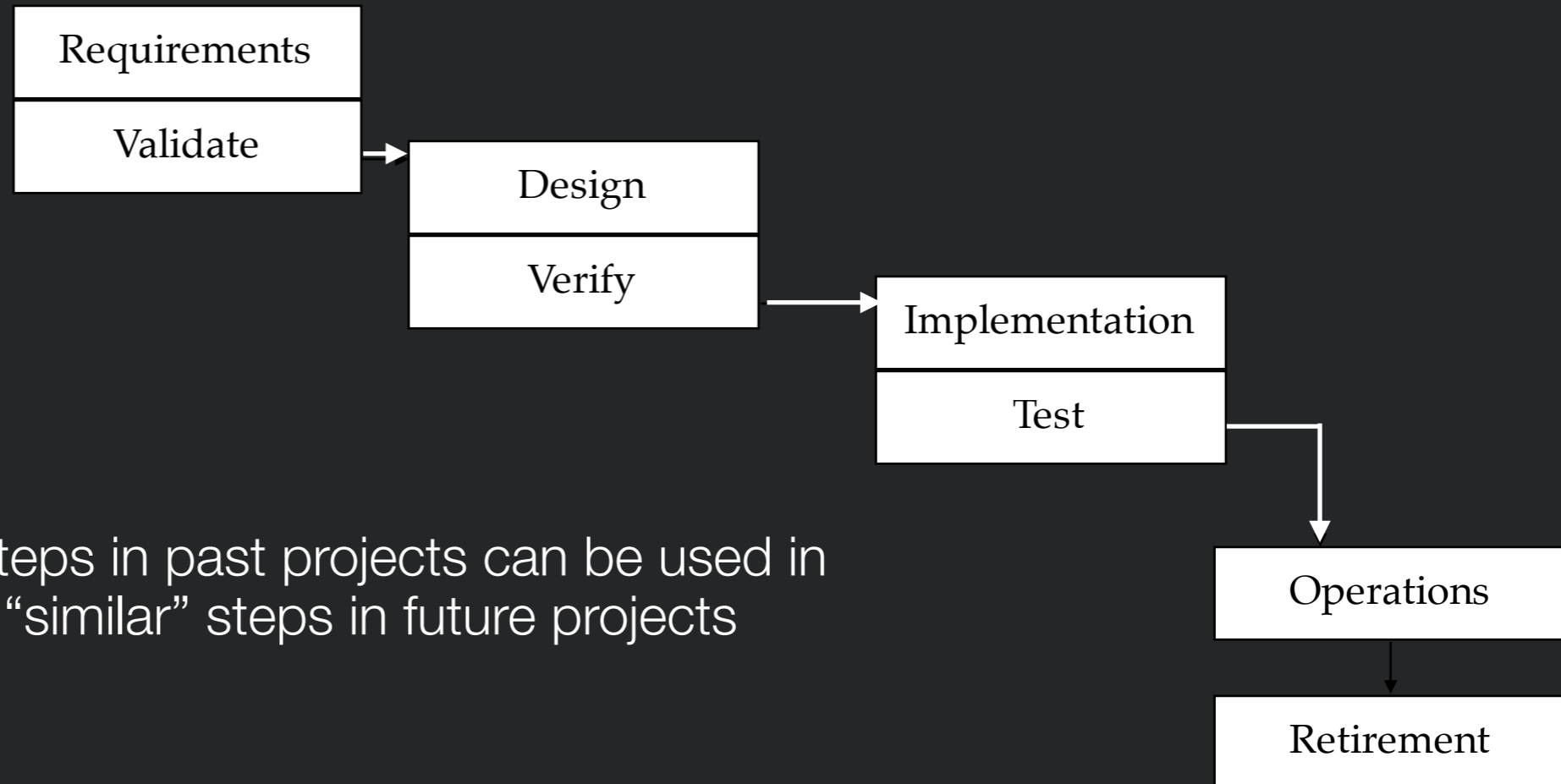
Waterfall Model

- Widely used today
- Advantages
 - Measurable progress



Waterfall Model

- Widely used today
- Advantages
 - Measurable progress
 - Experience applying steps in past projects can be used in estimating duration of “similar” steps in future projects

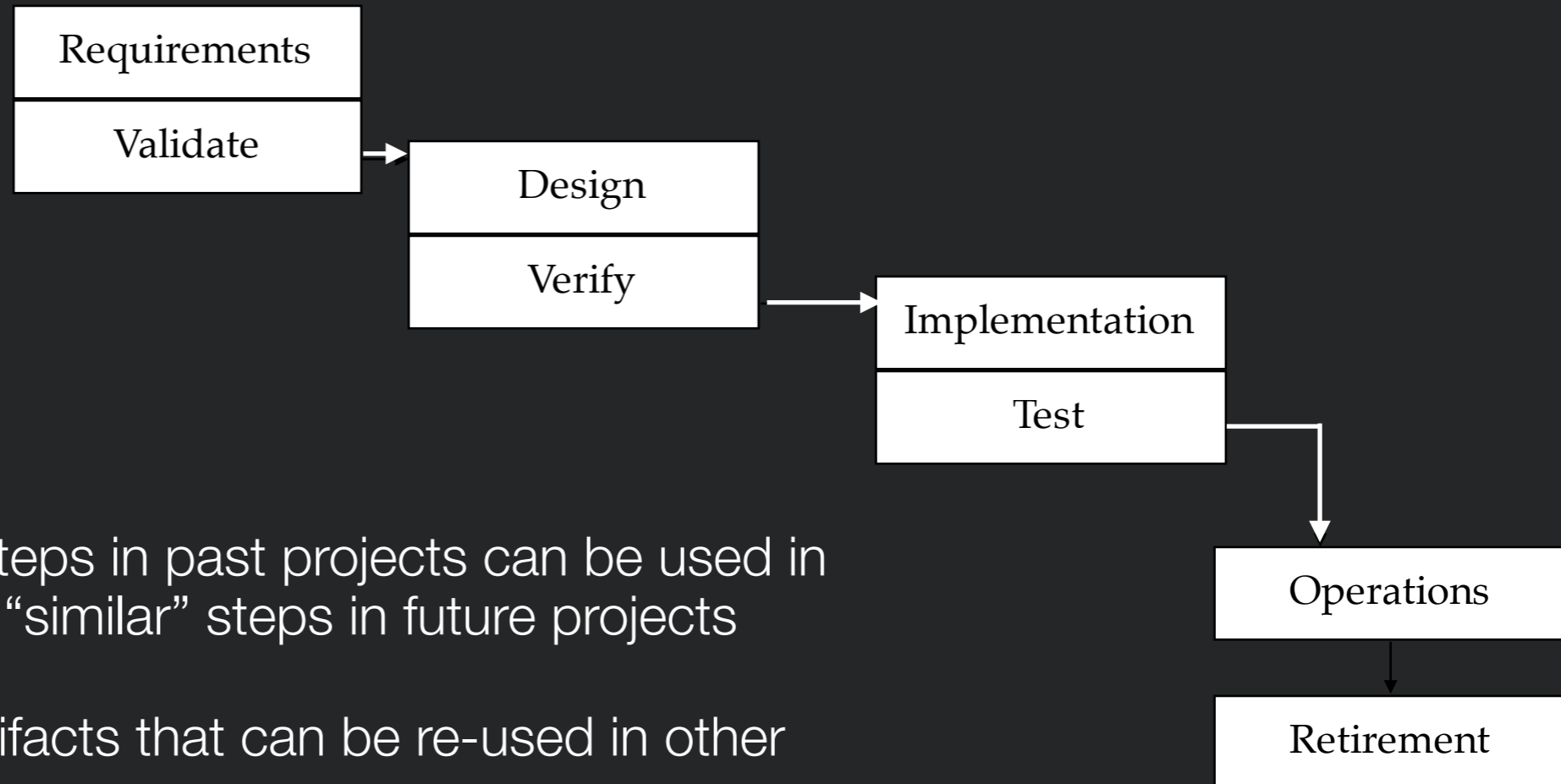


Waterfall Model

- Widely used today

- Advantages

- Measurable progress
- Experience applying steps in past projects can be used in estimating duration of “similar” steps in future projects
- Produces software artifacts that can be re-used in other projects



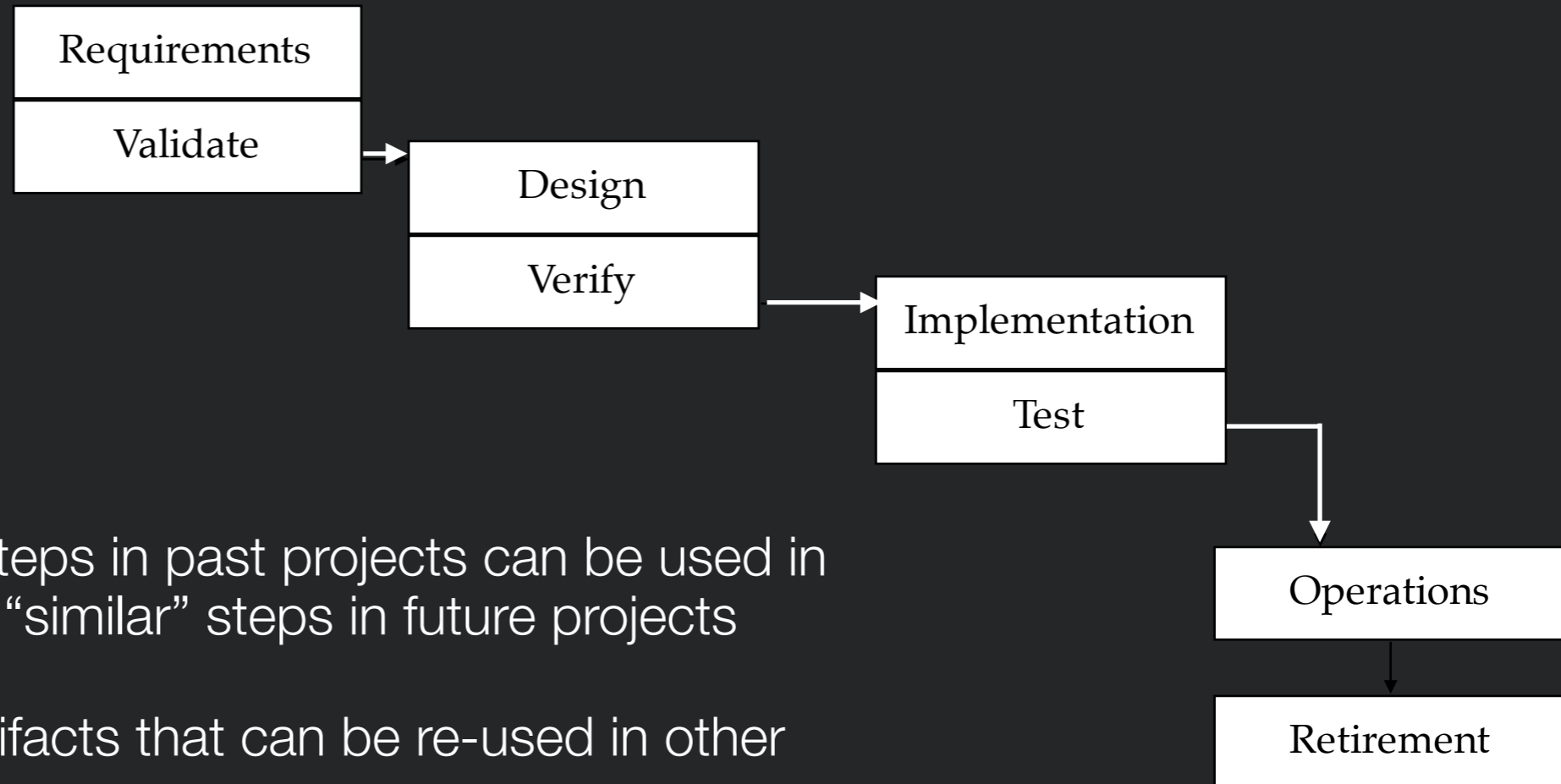
Waterfall Model

- Widely used today

- Advantages

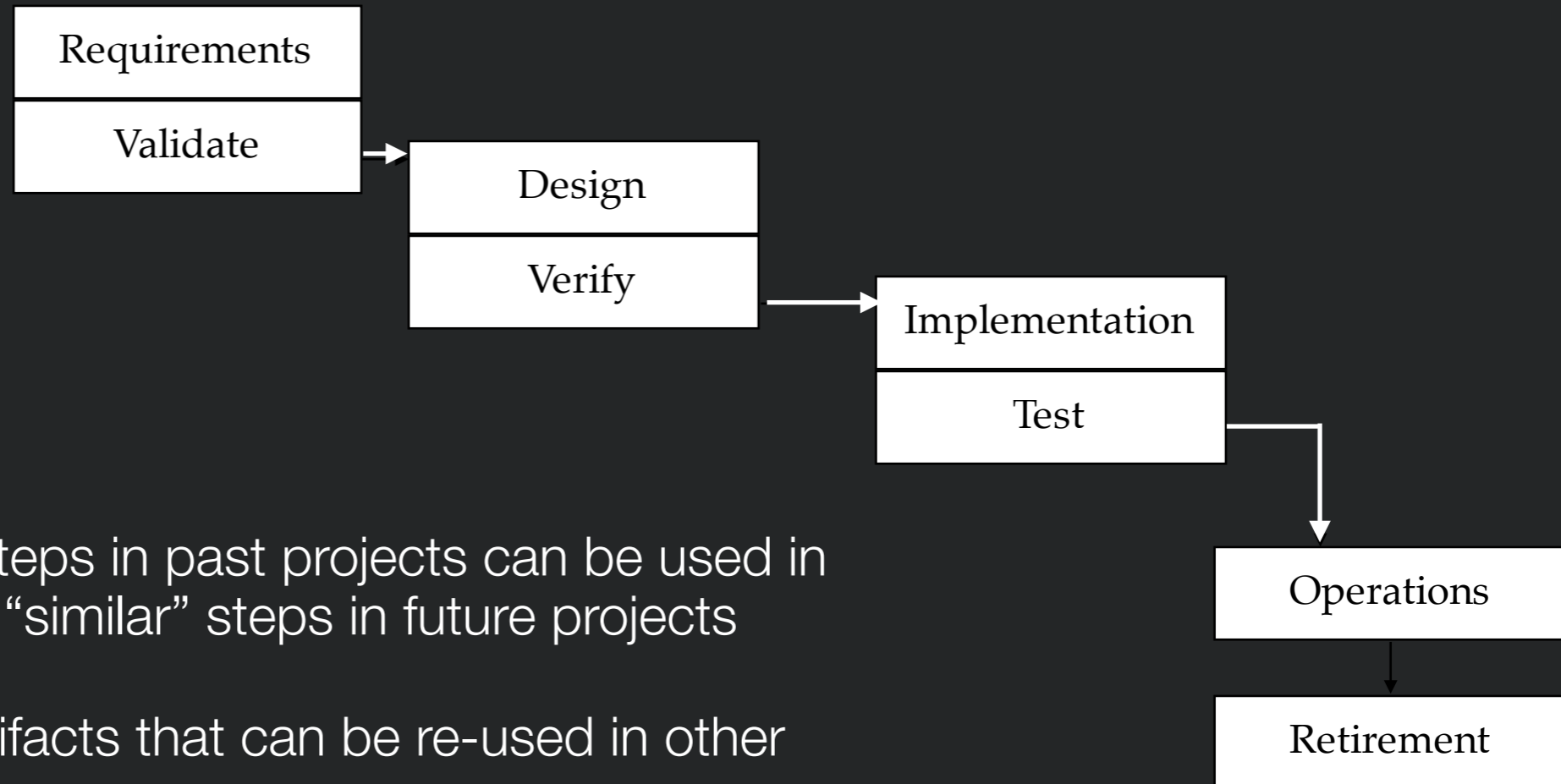
- Measurable progress
- Experience applying steps in past projects can be used in estimating duration of “similar” steps in future projects
- Produces software artifacts that can be re-used in other projects

- Disadvantages

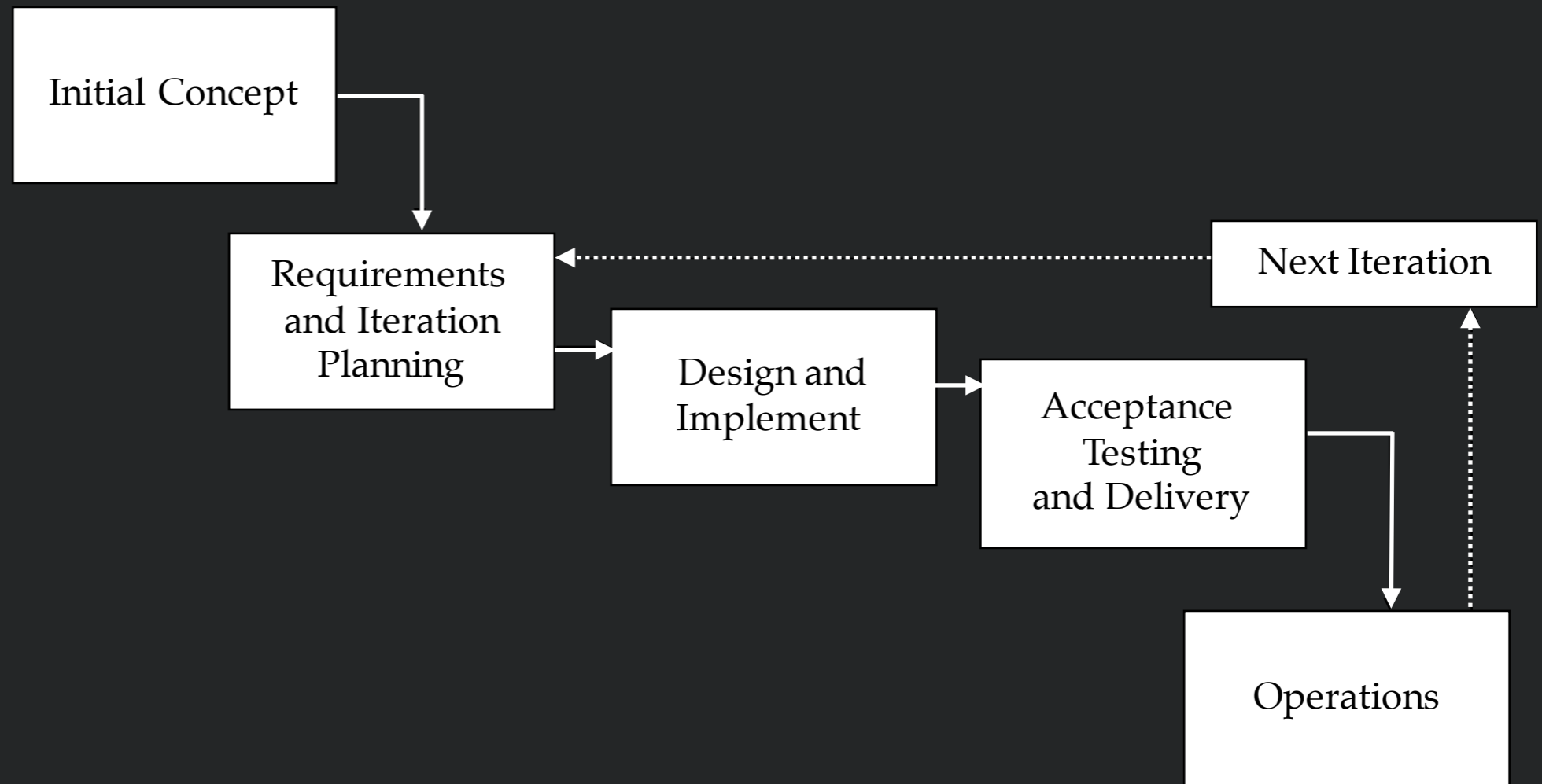


Waterfall Model

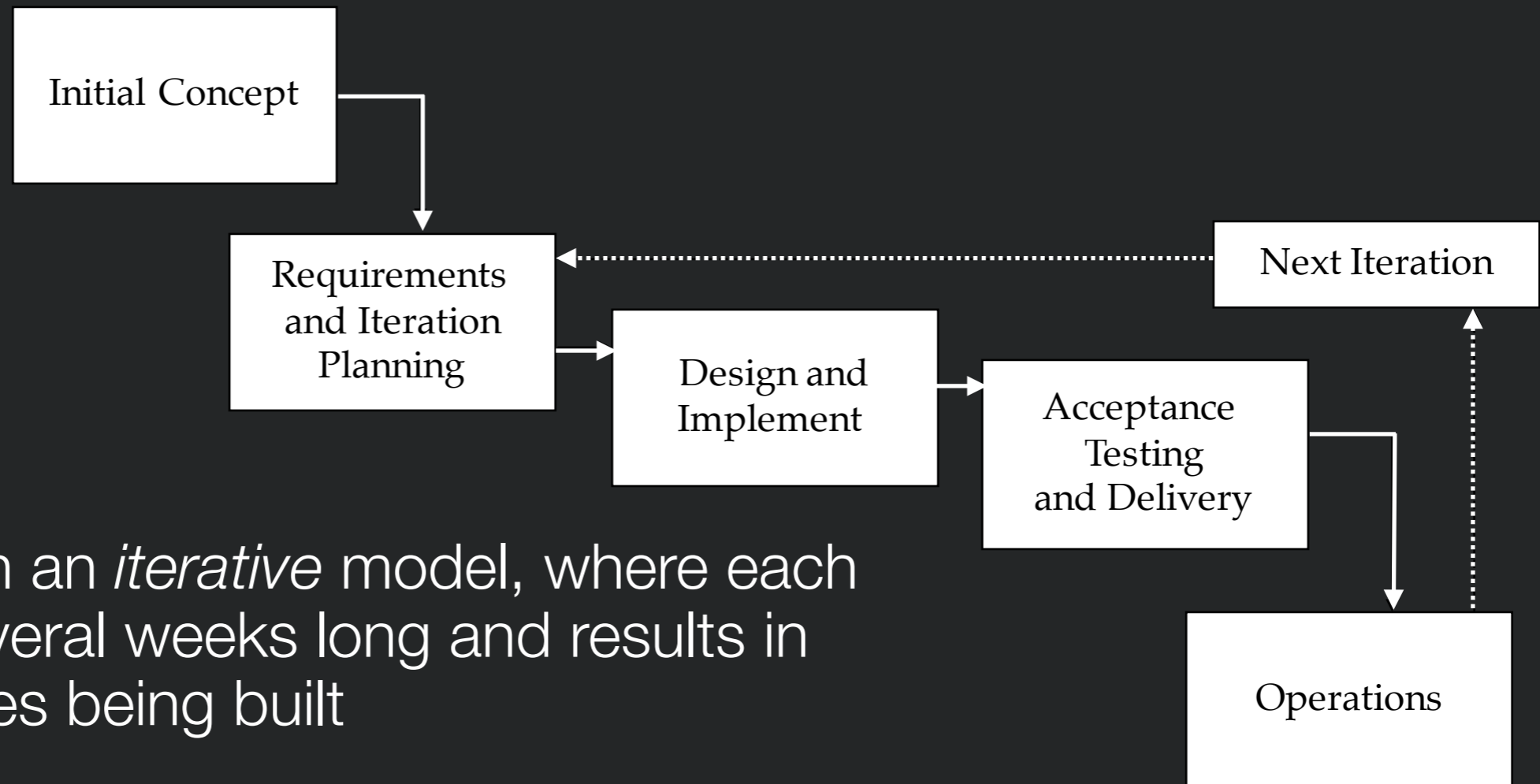
- Widely used today
- Advantages
 - Measurable progress
 - Experience applying steps in past projects can be used in estimating duration of “similar” steps in future projects
 - Produces software artifacts that can be re-used in other projects
- Disadvantages
 - Difficulty of accommodating change after the process is underway: One phase has to be complete before moving onto the next phase.



Agile Model

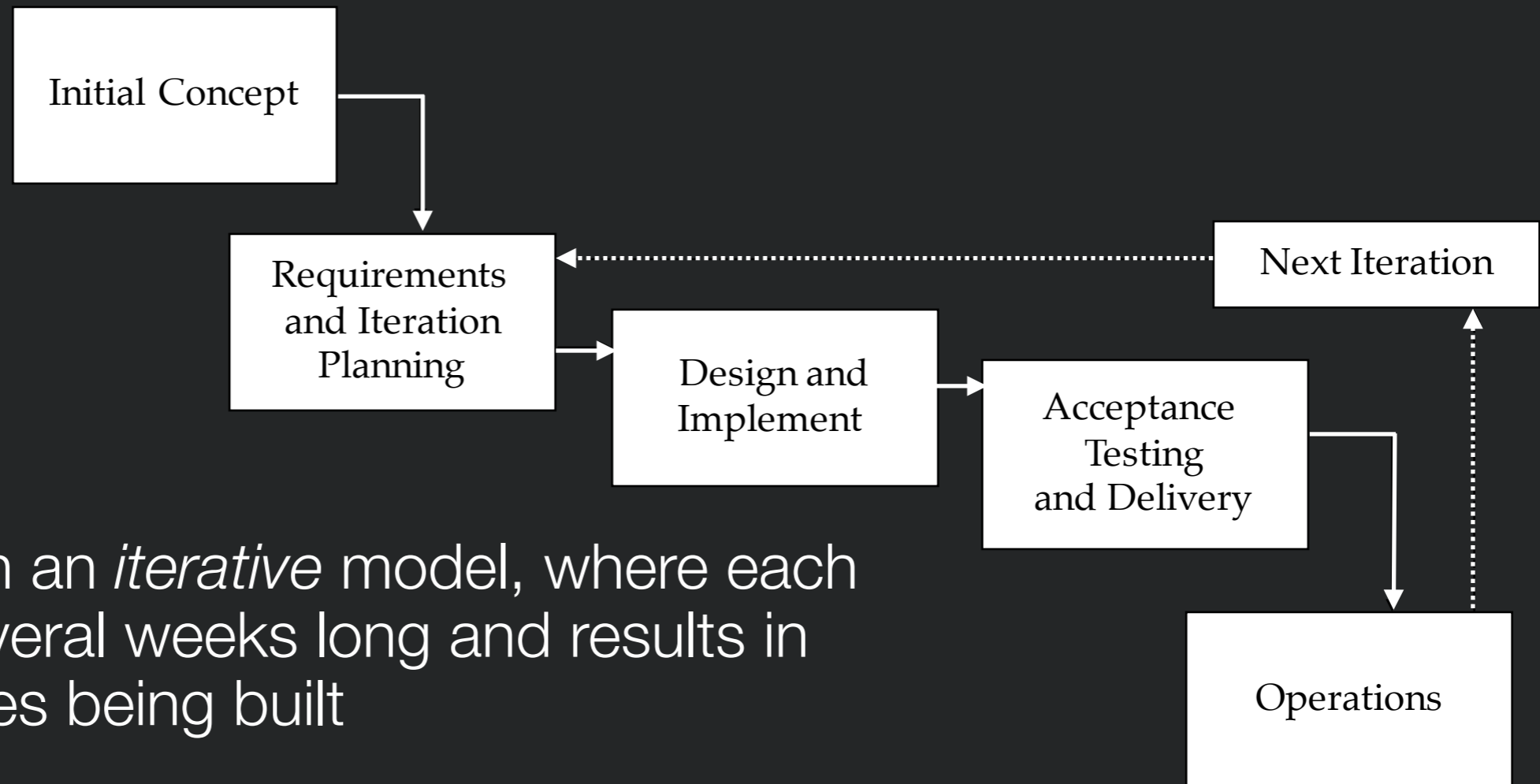


Agile Model



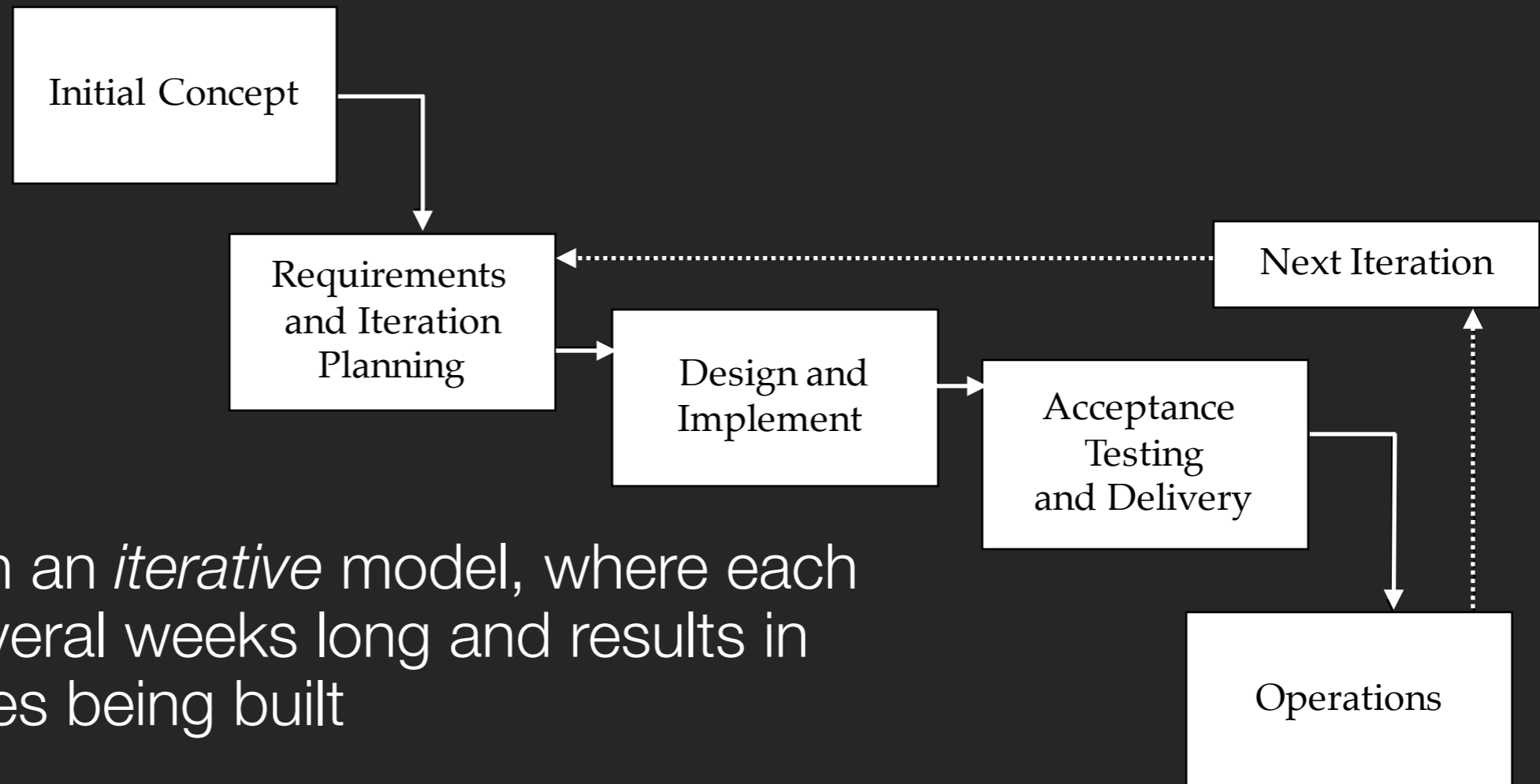
- Agile results in an *iterative* model, where each iteration is several weeks long and results in several features being built

Agile Model



- Agile results in an *iterative* model, where each iteration is several weeks long and results in several features being built
- Recognize that requirements **ALWAYS** evolve as you are trying to build something

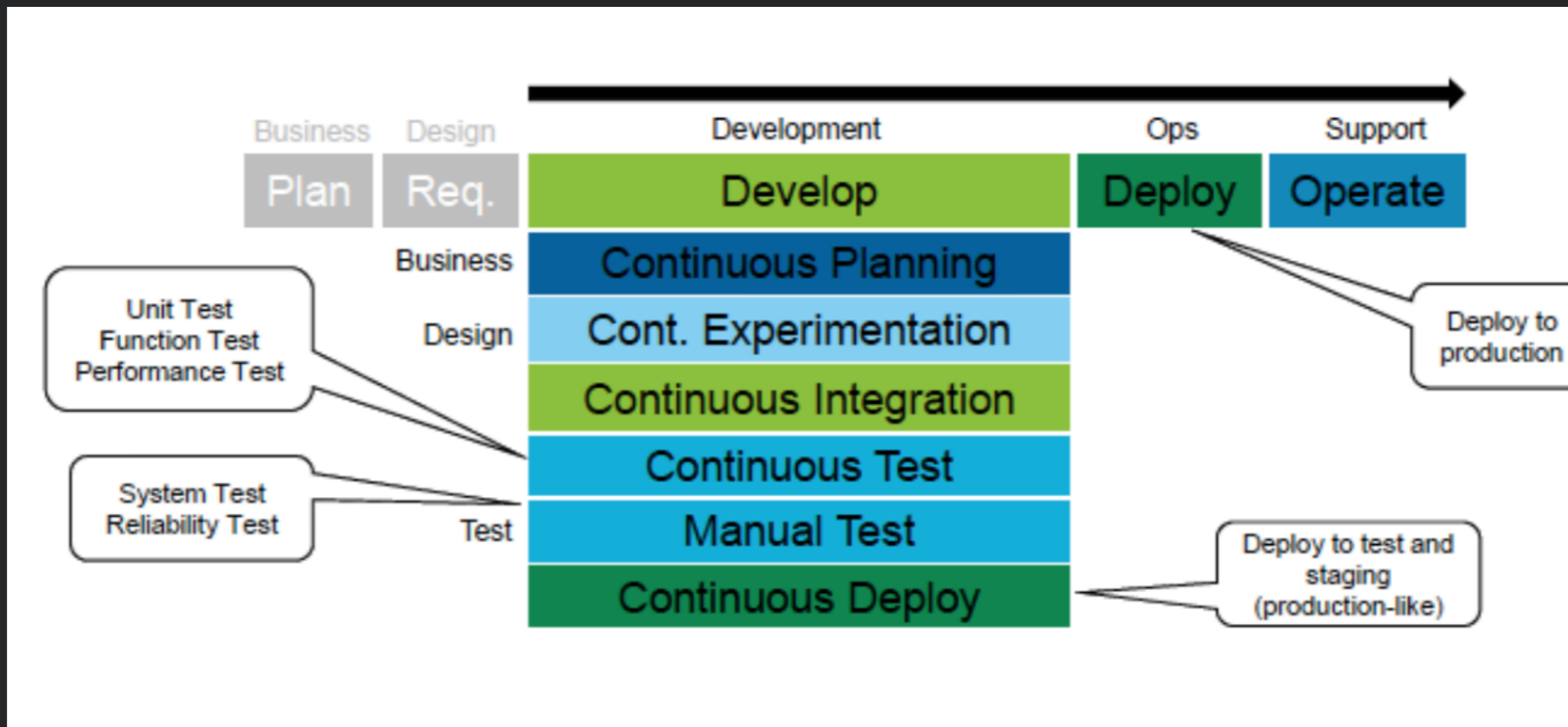
Agile Model



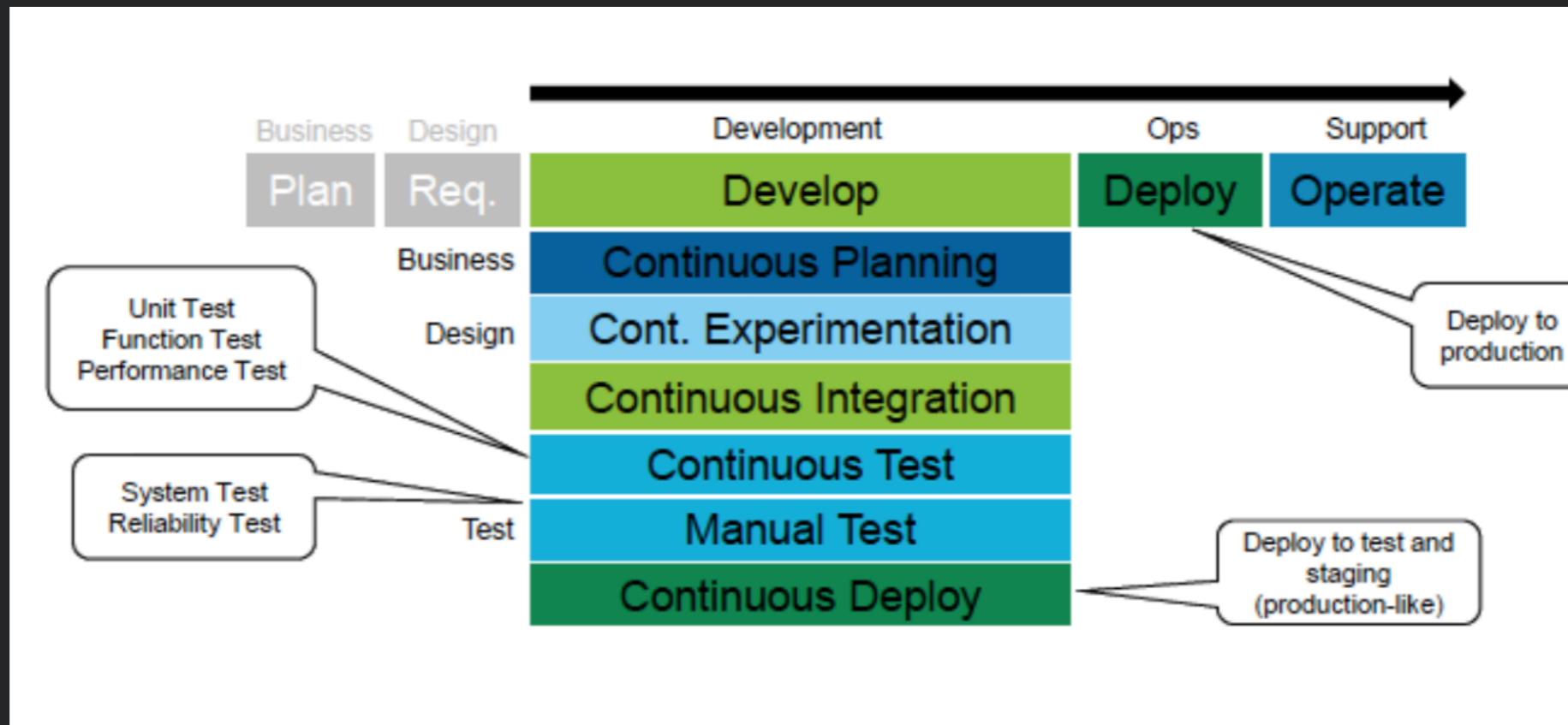
- Agile results in an *iterative* model, where each iteration is several weeks long and results in several features being built
- Recognize that requirements *ALWAYS* evolve as you are trying to build something
- Plus, maybe you can get useful feedback by delivering a partial app early



Continuous Development

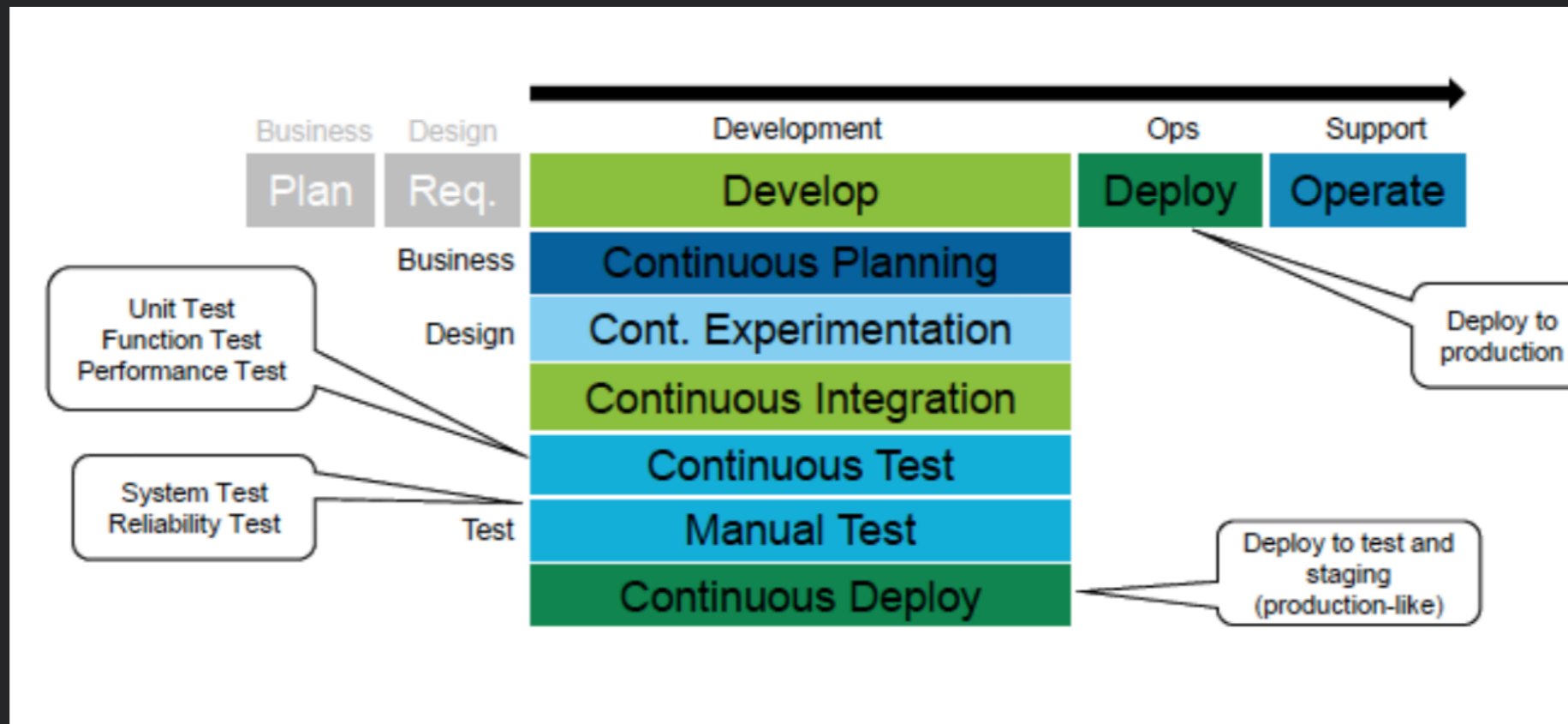


Continuous Development



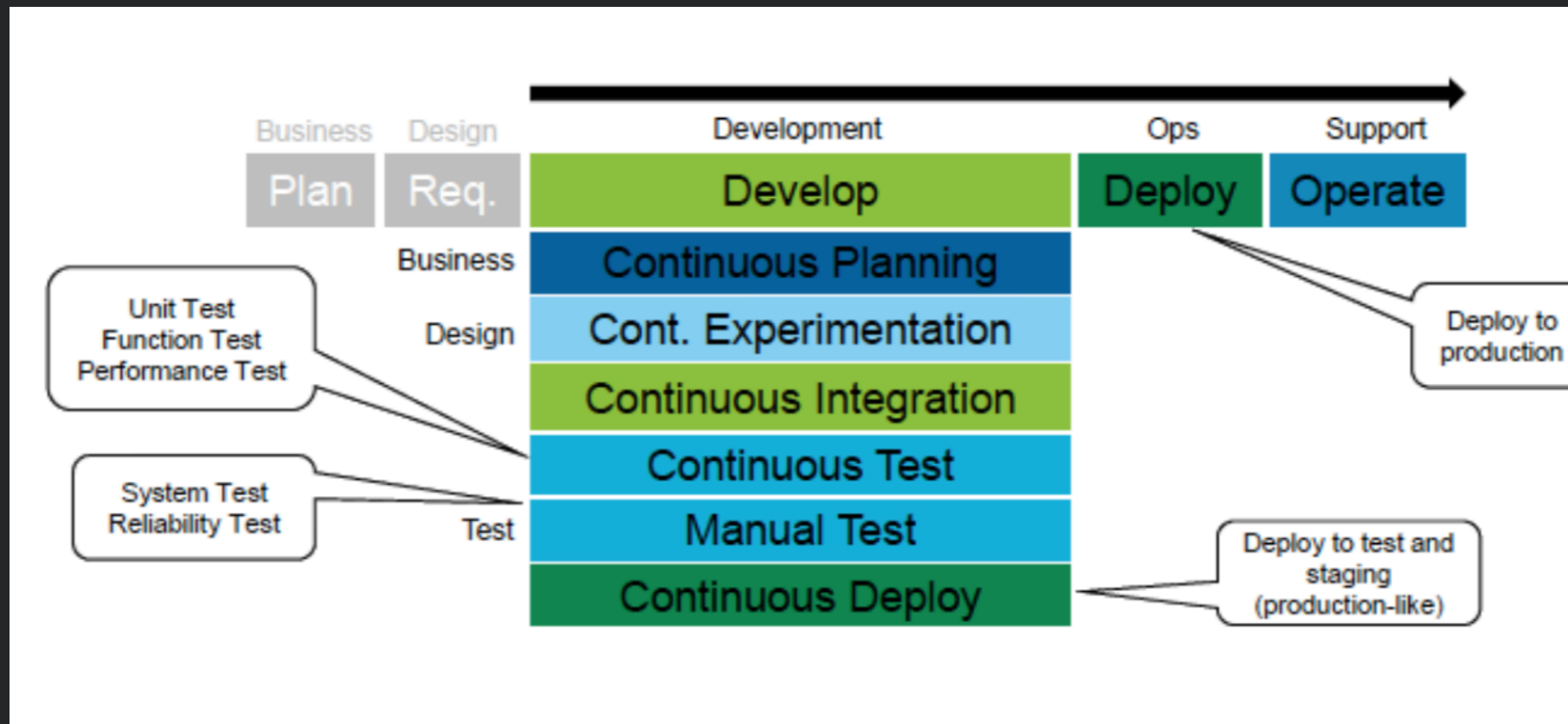
- Like agile, but...

Continuous Development



- Like agile, but...
- We are always working on different features

Continuous Development



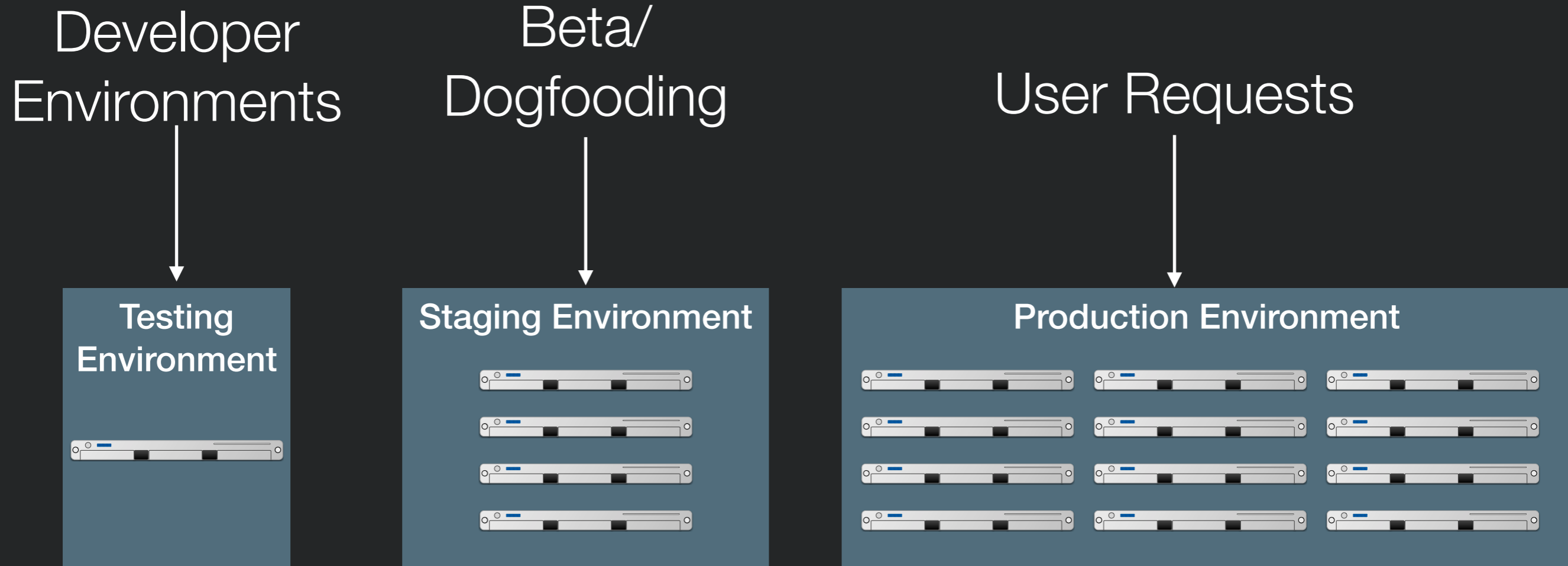
- Like agile, but...
 - We are always working on different features
 - We have a formal mechanism for deploying new versions of code and validating (test/staging/production)



The Value of the Staging Environment

- As software gets more complex with more dependencies, it's impossible to simulate the whole thing when testing
- ***Idea:*** Deploy to a complete production-like environment, but don't have everyone use it
 - Examples:
 - “Eat your own dogfood”
 - Beta/Alpha testers
- Lower risk if a problem occurs in staging than in production

Test-Stage-Production



Revisions are “promoted” towards production





Operations Responsibility

- Once we **deploy**, someone has to monitor software, make sure it's running OK, no bugs, etc
- Assume 3 environments:
 - Test, Staging, Production
- Whose job is it?

	Developers	Operators
Waterfall		
Agile		



Operations Responsibility

- Once we **deploy**, someone has to monitor software, make sure it's running OK, no bugs, etc
- Assume 3 environments:
 - Test, Staging, Production
- Whose job is it?

	Developers	Operators		
Waterfall		Test	Staging	Production
Agile				



Operations Responsibility

- Once we **deploy**, someone has to monitor software, make sure it's running OK, no bugs, etc
- Assume 3 environments:
 - Test, Staging, Production
- Whose job is it?

	Developers	Operators		
Waterfall		Test	Staging	Production
Agile	Test			



Operations Responsibility

- Once we **deploy**, someone has to monitor software, make sure it's running OK, no bugs, etc
- Assume 3 environments:
 - Test, Staging, Production
- Whose job is it?

	Developers	Operators		
Waterfall		Test	Staging	Production
Agile	Test		Staging	Production



Operations Responsibility

- Once we **deploy**, someone has to monitor software, make sure it's running OK, no bugs, etc
- Assume 3 environments:
 - Test, Staging, Production
- Whose job is it?

	Developers	Operators		
Waterfall		Test	Staging	Production
Agile	Test		Staging	Production
DevOps				



Operations Responsibility

- Once we **deploy**, someone has to monitor software, make sure it's running OK, no bugs, etc
- Assume 3 environments:
 - Test, Staging, Production
- Whose job is it?

	Developers	Operators		
Waterfall		Test	Staging	Production
Agile	Test		Staging	Production
DevOps	Test Staging			



Operations Responsibility

- Once we **deploy**, someone has to monitor software, make sure it's running OK, no bugs, etc
- Assume 3 environments:
 - Test, Staging, Production
- Whose job is it?

	Developers	Operators		
Waterfall		Test	Staging	Production
Agile	Test		Staging	Production
DevOps	Test Staging			Production



Operations Responsibility

- Once we **deploy**, someone has to monitor software, make sure it's running OK, no bugs, etc
- Assume 3 environments:
 - Test, Staging, Production
- Whose job is it?

	Developers	Operators		
Waterfall		Test	Staging	Production
Agile	Test		Staging	Production
DevOps	Test	Staging	Production	Production



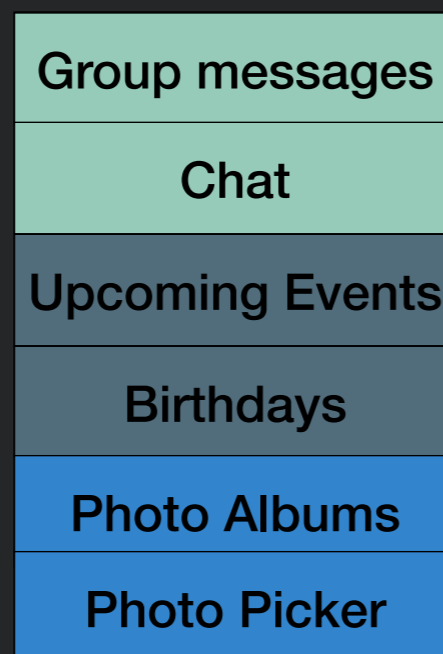
DevOps Values

- No silos, no walls, no responsibility "pipelines"
- One team owns changes "from cradle to grave"
- *You* are the support person for your changes, regardless of platform
- Example: Facebook mobile teams

Engineering Teams

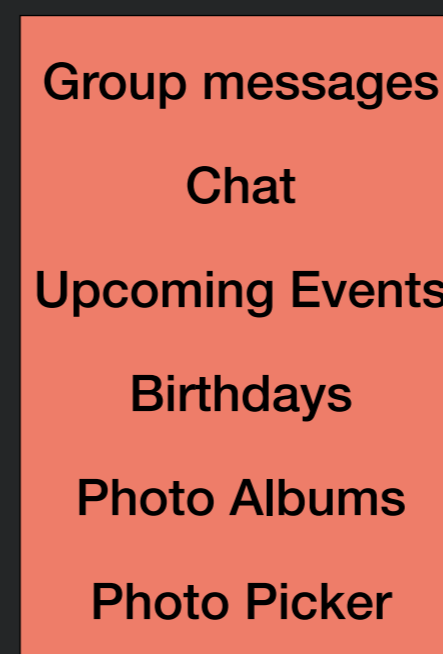
- Messages
- Events
- Photos
- Android
- iOS

Desktop/Web



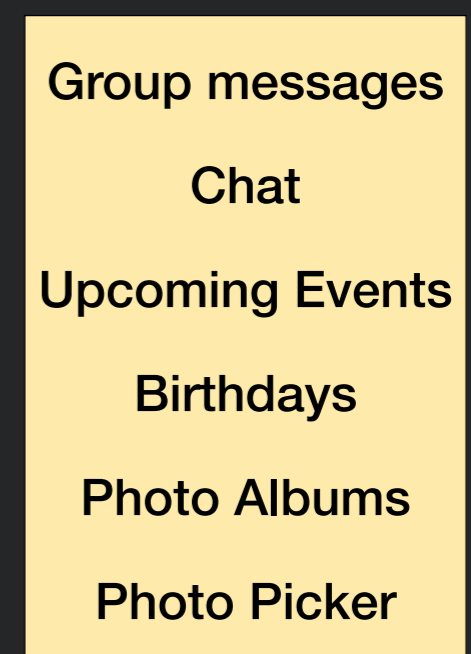
Product Experts

Android



Platform Experts

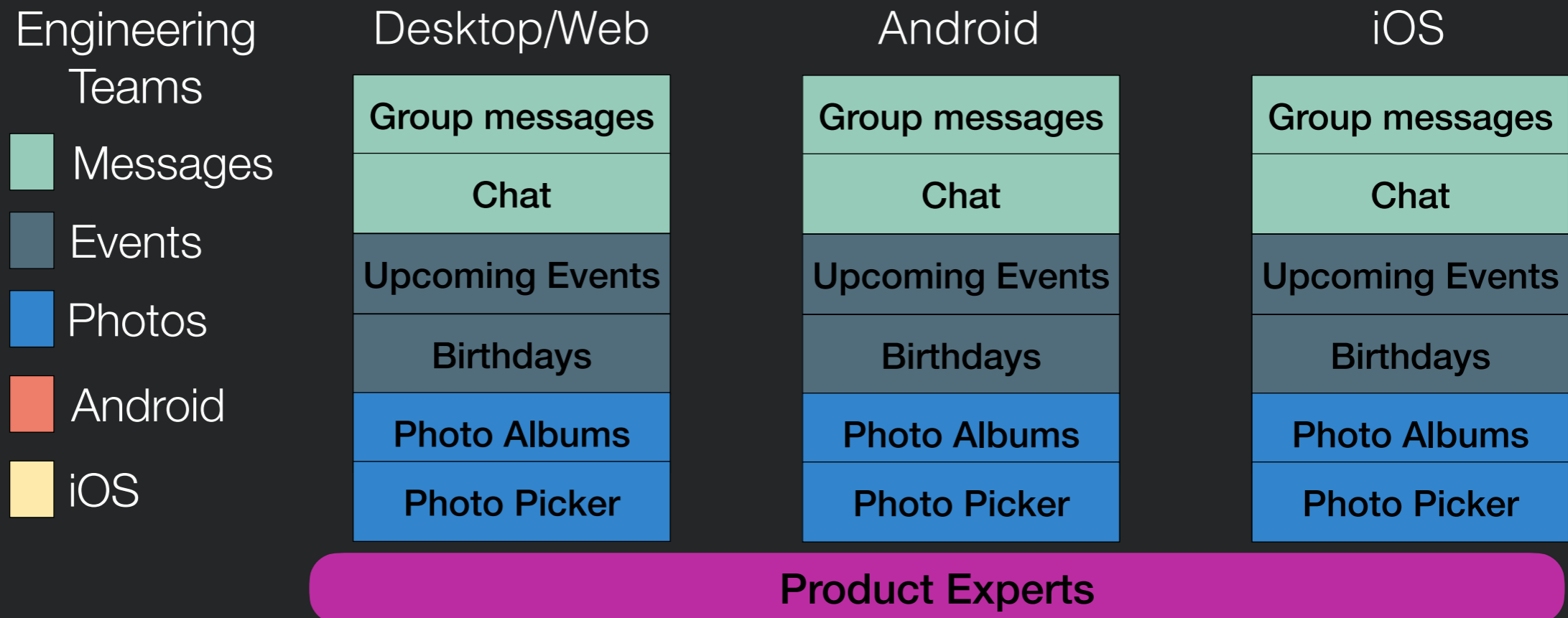
iOS





DevOps Values

- No silos, no walls, no responsibility "pipelines"
- One team owns changes "from cradle to grave"
- *You* are the support person for your changes, regardless of platform
- Example: Facebook mobile teams

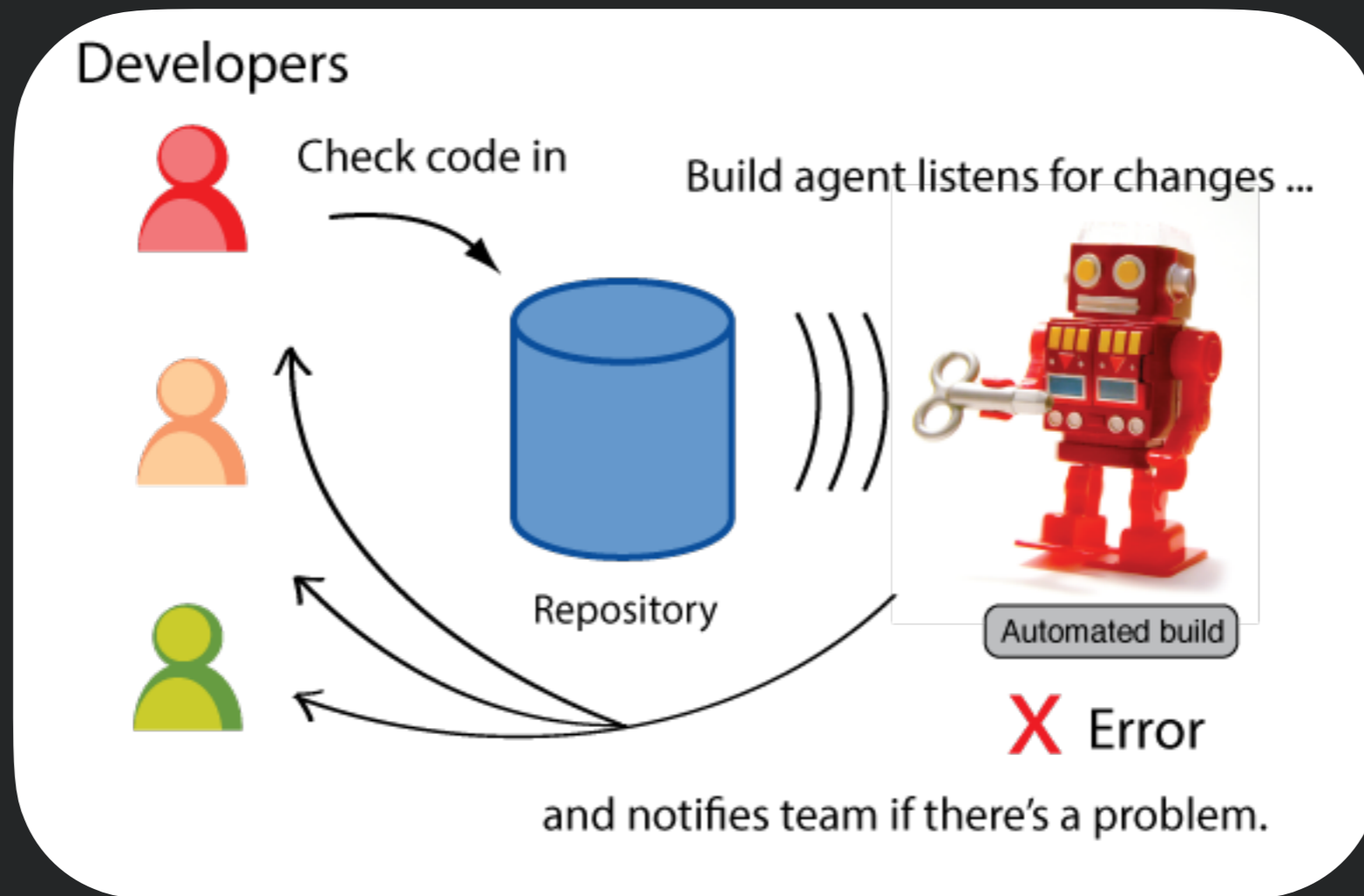




Continuous X

- Continuous Integration:
 - A practice where developers automatically build, test, and analyze a software change in response to every software change committed to the source repository.
- Continuous Delivery:
 - A practice that ensures that a software change can be delivered and ready for use by a customer by testing in production-like environments.
- Continuous Deployment:
 - A practice where incremental software changes are automatically tested, vetted, and deployed to production environments.

Continuous Integration



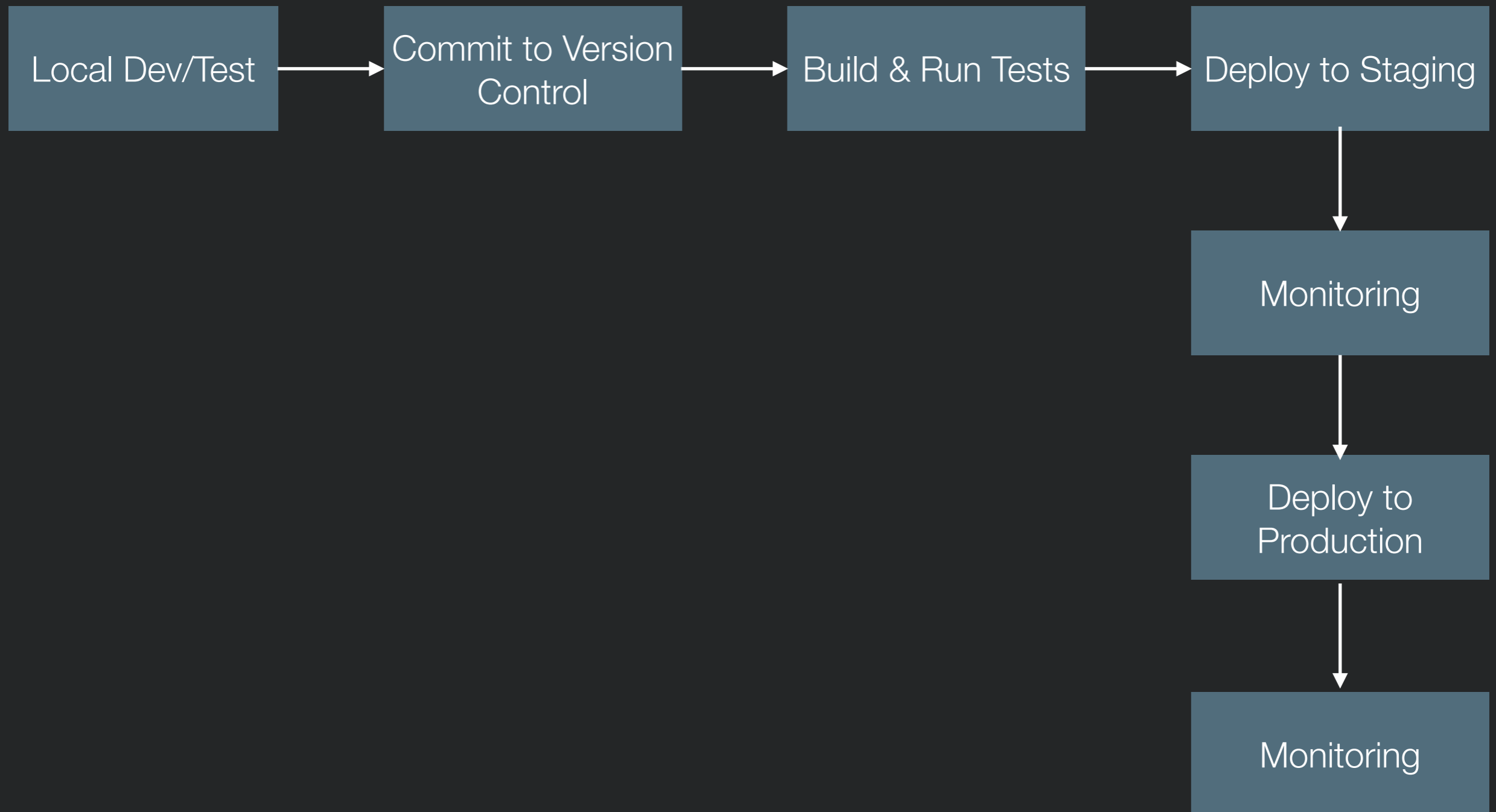


Continuous Integration

- Commit Code Frequently
- Don't commit broken code
- Fix broken builds immediately
- Write automated developer tools
- All tests and inspections must pass
- Run private builds
- Avoid getting broken code



Deployment Pipeline





Deployment Pipeline

- Even if you are deploying every day, you still have some latency
- A new feature I develop today won't be released today
- But, a new feature I develop today can begin the *release pipeline* today (minimizes risk)
- *Release Engineer*: gatekeeper who decides when something is ready to go out, oversees the actual deployment process



Deployment Example: Facebook.com

Developers working in their own branch



Deployment Example: Facebook.com

Developers working in their own branch

When feature is ready, push as 1 change to master branch

~1 week of development

master branch



Deployment Example: Facebook.com

Developers working in their own branch

When feature is ready, push as 1 change to master branch

~1 week of development

master branch

3 days

Stabilize

Weekly

*All changes from week
that are ready for release*

release branch



Deployment Example: Facebook.com

Developers working in their own branch

When feature is ready, push as 1 change to master branch

~1 week of development

master branch

All changes that survived stabilizing

3 days

4 days

Stabilize

Release Branch

Weekly

*All changes from week
that are ready for release*

release branch



Deployment Example: Facebook.com

Developers working in their own branch

When feature is ready, push as 1 change to master branch

~1 week of development

master branch

All changes that survived stabilizing

3 days

4 days

Weekly

Stabilize

Release Branch

All changes from week that are ready for release

release branch

Twice Daily

production



Deployment Example: Facebook.com

Developers working in their own branch

When feature is ready, push as 1 change to master branch

~1 week of development

master branch

All changes that survived stabilizing

3 days

4 days

Weekly

Stabilize

Release Branch

All changes from week that are ready for release

release branch

production

Twice Daily

Your change doesn't go out unless you're there that day at that time to support it!



Deployment Example: Facebook.com

Developers working in their own branch

When feature is ready, push as 1 change to master branch

~1 week of development

master branch

All changes that survived stabilizing

3 days

4 days

Weekly

Stabilize

Release Branch

All changes from week that are ready for release

release branch

production

Twice Daily

Your change doesn't go out unless you're there that day at that time to support it!

"When in doubt back out"



Deployment Example: Facebook.com

Developers working in their own branch

When feature is ready, push as 1 change to master branch

~1 week of development

~1 week of development

master branch

Weekly

All changes from week that are ready for release

3 days

Stabilize

4 days

Release Branch

All changes that survived stabilizing

release branch

production

Twice Daily

Your change doesn't go out unless you're there that day at that time to support it!

"When in doubt back out"

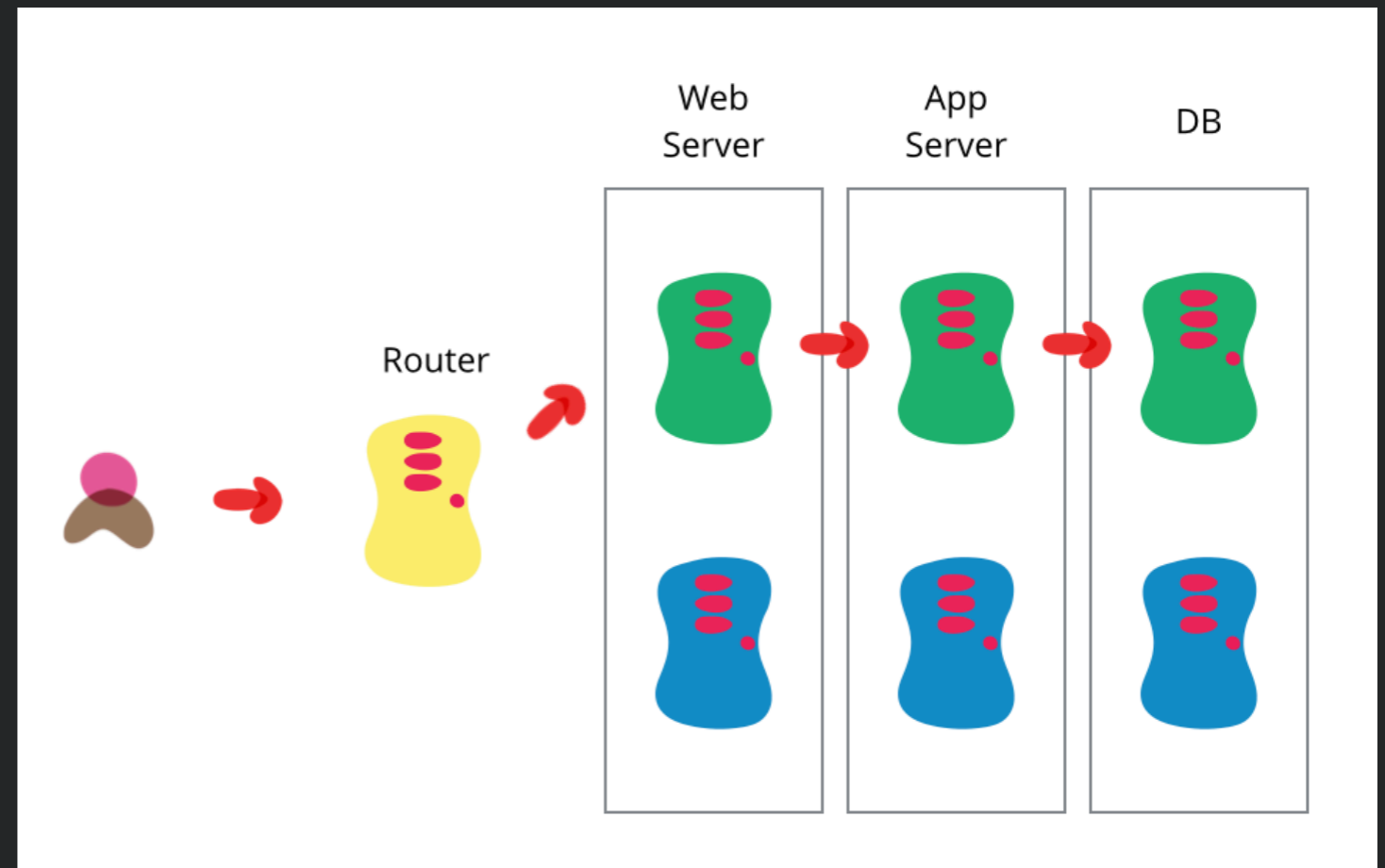


Continuous Integration & Continuous Deployment

- Thousands of changes coming together at once
- To isolate problems:
 - Every time that every change is potentially going to be introduced, the entire system is integrated and tested
- Facebook does 20,000-30,000 complete integrations PER DAY for mobile alone
- General rule:
 - Cost of compute time to run tests more often is way less than the cost of a failure

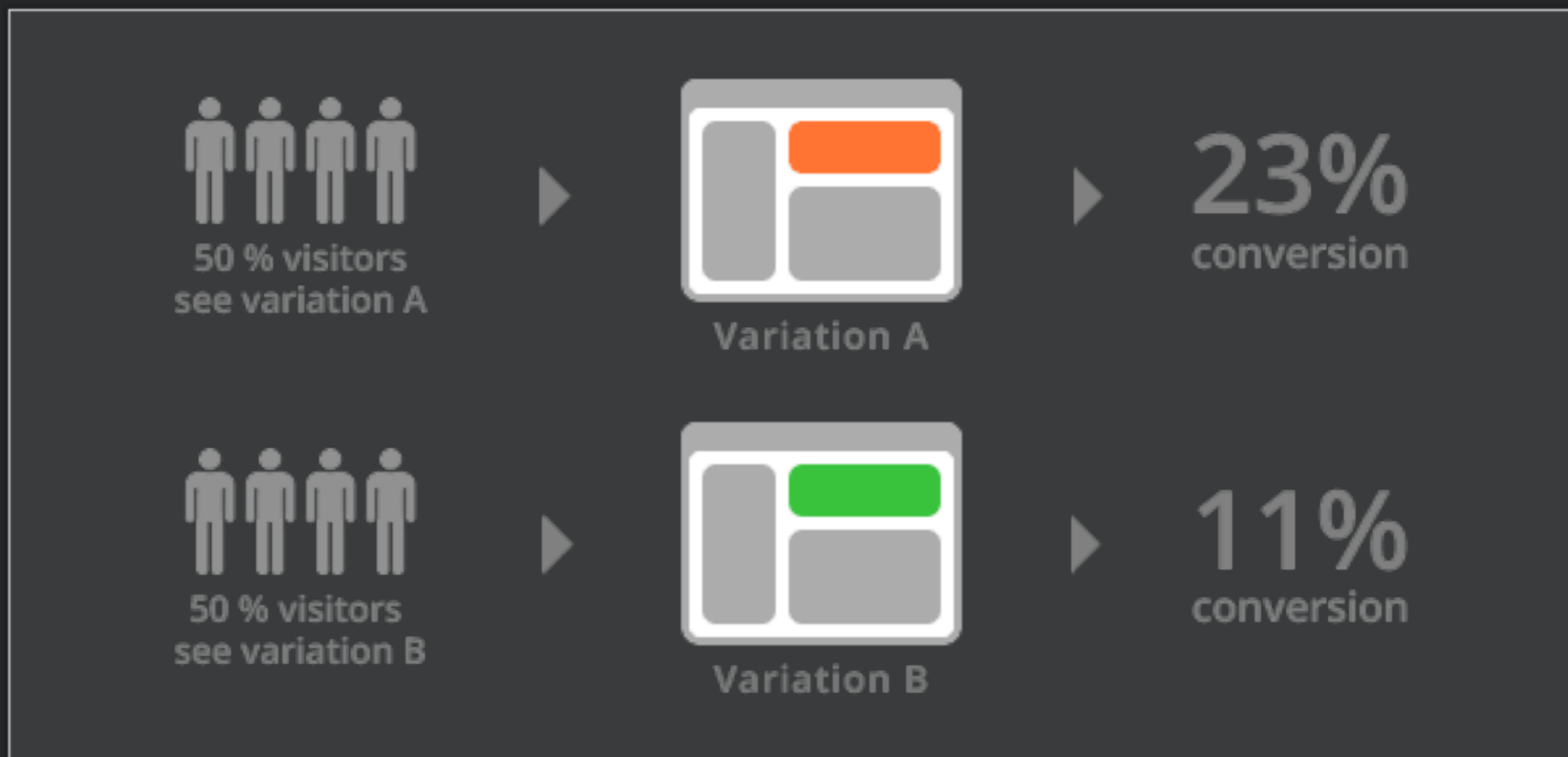
Blue-Green Deployment

- Always have 2 complete environments ready:
 - One that you're using now
 - One that you're just about ready to use
- Easily switch which is handling requests



A/B Testing

- Ways to test new features for usability, popularity, performance
- Show 50% of your site visitors version A, 50% version B, collect metrics on each, decide which is better



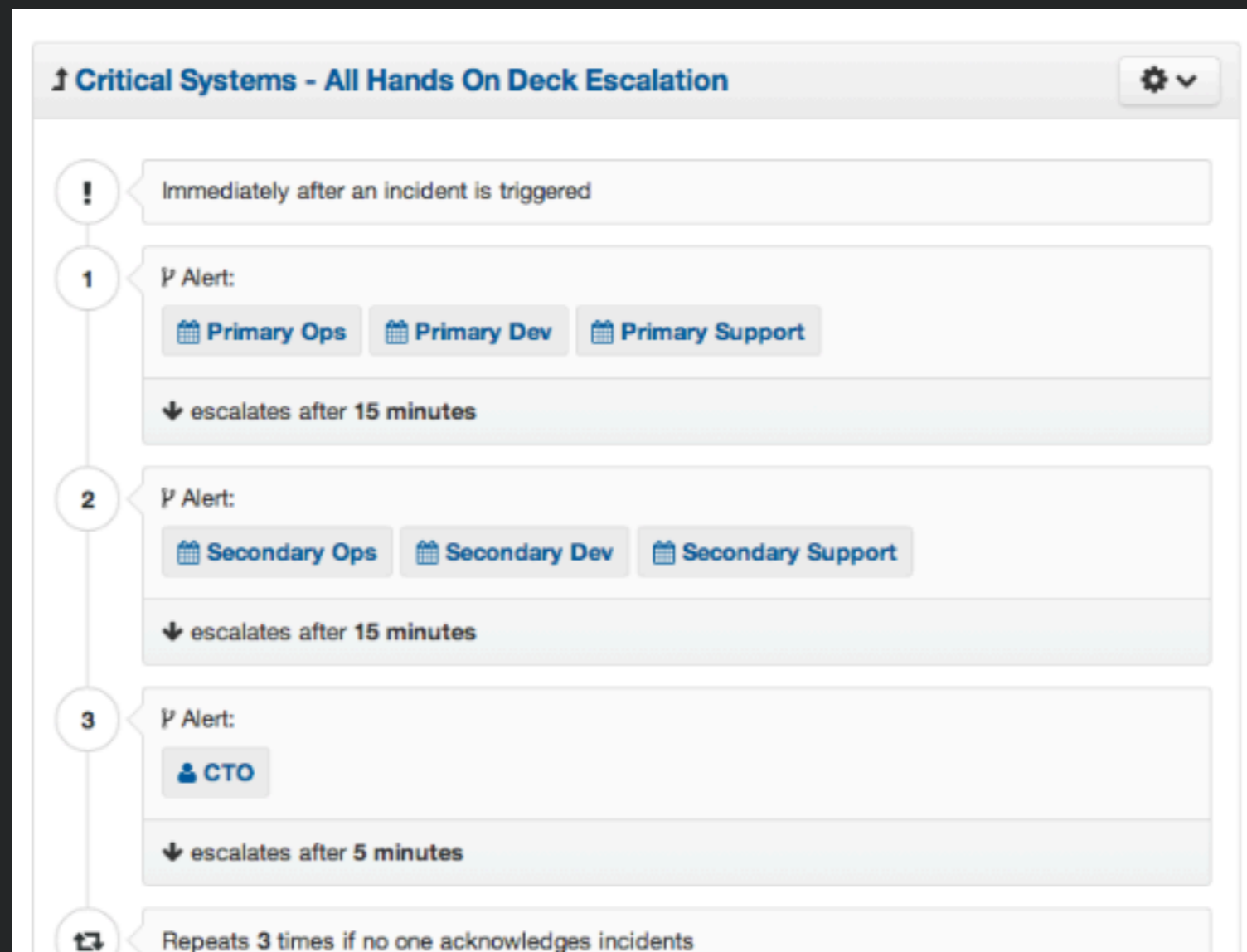


Monitoring

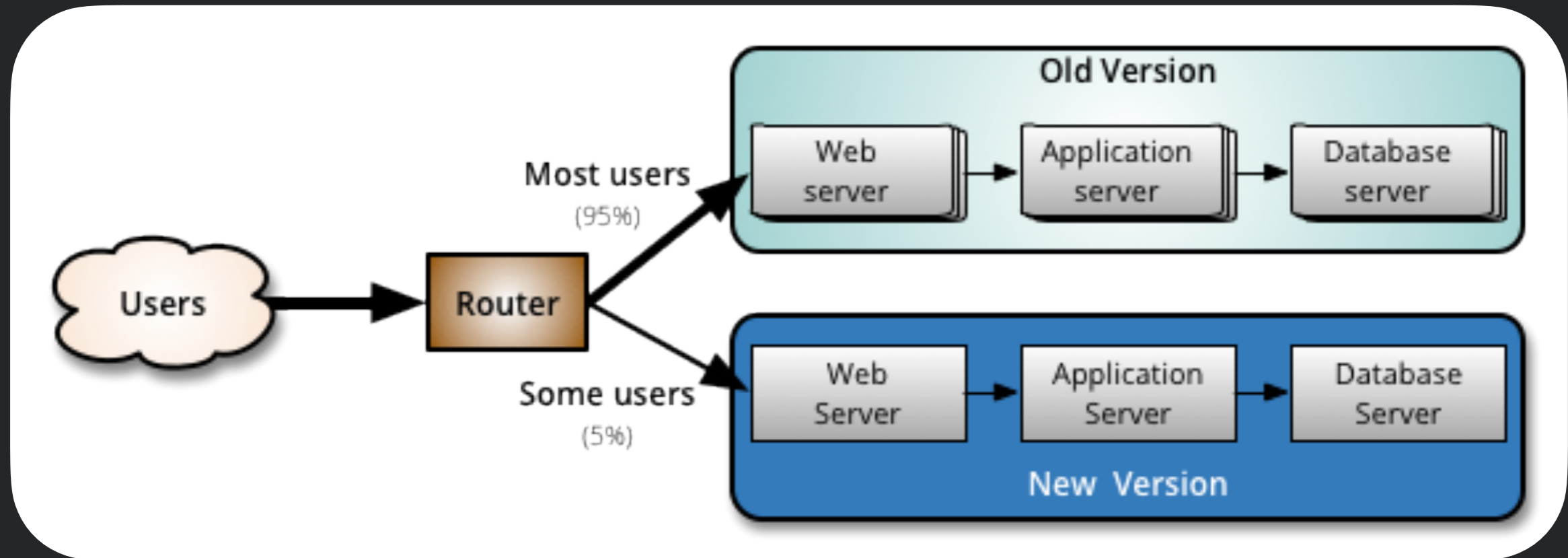
- Hardware
 - Voltages, temperatures, fan speeds, component health
- OS
 - Memory usage, swap usage, disk space, CPU load
- Middleware
 - Memory, thread/db connection pools, connections, response time
- Applications
 - Business transactions, conversion rate, status of 3rd party components

When Things Go Wrong

- Automated monitoring systems can notify “on-call” staff of a problem
- Triage & escalation



Canaries



Monitor both:
But minimize impact of problems in new version



Making it Happen

- Build Tools
- Test Automation
- Build Servers
- Deployment Tools



Build Tools

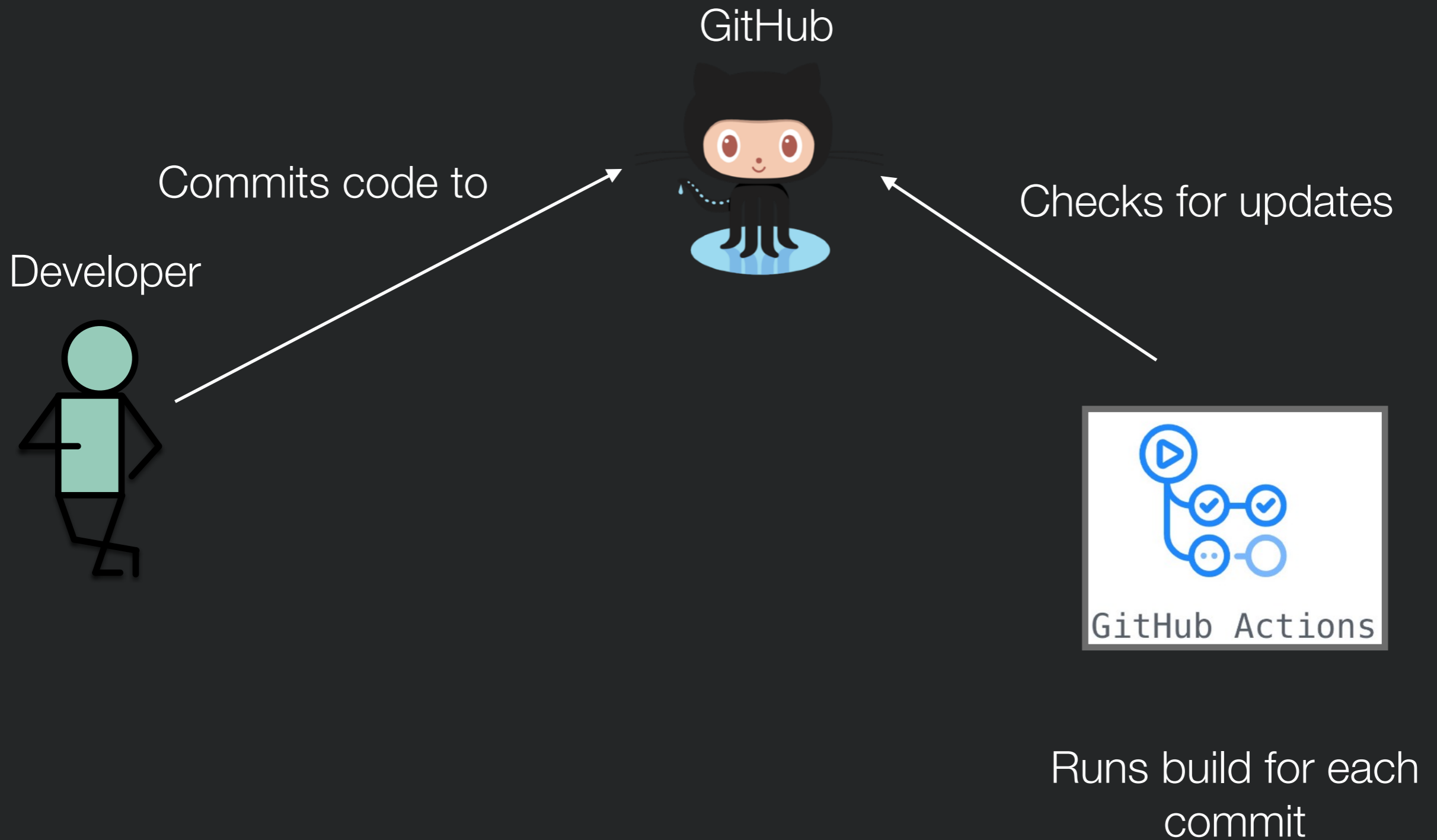
- Need to be able to automate construction of our executable software...
Example:
 - “Install d3 with bower with grunt with npm with brew.” *pew*
- We'd like a general method for describing and executing build tasks:
 - Minify my code
 - Run my tests
 - Generate some documentation
 - Deploy to staging
- Ensure that builds are repeatable, reproducible and standard



Build Servers

- Once we have a standard mechanism for describing how to build our code, no reason to only build it on our own machine
- Continuous Integration servers run these builds in the cloud
 - Bamboo, Hudson/Jenkins, TravisCI, GitHub Actions
- Easy to use - typically monitors your source repository for changes, then runs a build
- Really helps with organizing tests and results
- Can scale the build server independently of the rest of your processes

GitHub Actions





GitHub Actions

- Can see history and status per commit

The screenshot shows the GitHub Actions interface for a repository named "Week Week 6 Lecture Materials". The repository is on the "main" branch with commit hash "60eb608". The "Actions" tab is selected in the top navigation bar. The workflow "ci" is shown, with the "deploy" job selected. The "deploy" job is shown as "succeeded 2 days ago in 42s". The job log is displayed, showing the following steps:

- Set up job** (3s):
 - 1 Current runner version: '2.283.2'
 - 2 ▶ Operating System
 - 6 ▶ Virtual Environment
 - 11 ▶ Virtual Environment Provisioner
 - 13 ▶ GITHUB_TOKEN Permissions
 - 26 Prepare workflow directory
 - 27 Prepare all required actions
 - 28 Getting action download info
 - 29 Download action repository 'actions/checkout@v2' (SHA:5a4ac9002d0be2fb38bd78e4b4dbde5606d7042f)
 - 30 Download action repository 'actions/setup-python@v2' (SHA:dc73133d4da04e56a135ae2246682783cc7c7cb6)
- Run actions/checkout@v2** (5s):
 - 1 ▶ Run actions/checkout@v2
 - 13 Syncing repository: kpmoran-teaching/swe-432-f21
 - 14 ▶ Getting Git version info
 - 18 Deleting the contents of '/home/runner/work/swe-432-f21/swe-432-f21'
 - 19 ▶ Initializing the repository
 - 33 ▶ Disabling automatic garbage collection
 - 35 ▶ Setting up auth
 - 41 ▶ Fetching the repository
 - 154 ▶ Determining the checkout info



GitHub Actions

- Can see history and status per commit

The screenshot shows the GitHub Actions interface for a repository named "Week Week 6 Lecture Materials". The repository is on the "main" branch with commit hash "60eb608". The "Actions" tab is selected in the top navigation bar. The workflow "ci" is shown, with the "deploy" job selected. The "deploy" job is shown as "succeeded 2 days ago in 42s". The job log is displayed, showing the following steps:

- Set up job** (3s):
 - 1 Current runner version: '2.283.2'
 - 2 ▶ Operating System
 - 6 ▶ Virtual Environment
 - 11 ▶ Virtual Environment Provisioner
 - 13 ▶ GITHUB_TOKEN Permissions
 - 26 Prepare workflow directory
 - 27 Prepare all required actions
 - 28 Getting action download info
 - 29 Download action repository 'actions/checkout@v2' (SHA:5a4ac9002d0be2fb38bd78e4b4dbde5606d7042f)
 - 30 Download action repository 'actions/setup-python@v2' (SHA:dc73133d4da04e56a135ae2246682783cc7c7cb6)
- Run actions/checkout@v2** (5s):
 - 1 ▶ Run actions/checkout@v2
 - 13 Syncing repository: kpmoran-teaching/swe-432-f21
 - 14 ▶ Getting Git version info
 - 18 Deleting the contents of '/home/runner/work/swe-432-f21/swe-432-f21'
 - 19 ▶ Initializing the repository
 - 33 ▶ Disabling automatic garbage collection
 - 35 ▶ Setting up auth
 - 41 ▶ Fetching the repository
 - 154 ▶ Determining the checkout info



Summary

- DevOps: Developers as Operators
- Continuous Integration & Deployment: Techniques for reducing time to get features out the door
- Staging environments reduce risk
- Build Systems and Services help automate CI