REACT

Day 1

Hi, I'm Josh

CTO @ TRUEFIT

FATHER

MAKER

FIND ME:

@joshgretz

github/jgretz

gretzlab.com

TODAY'S AGENDA

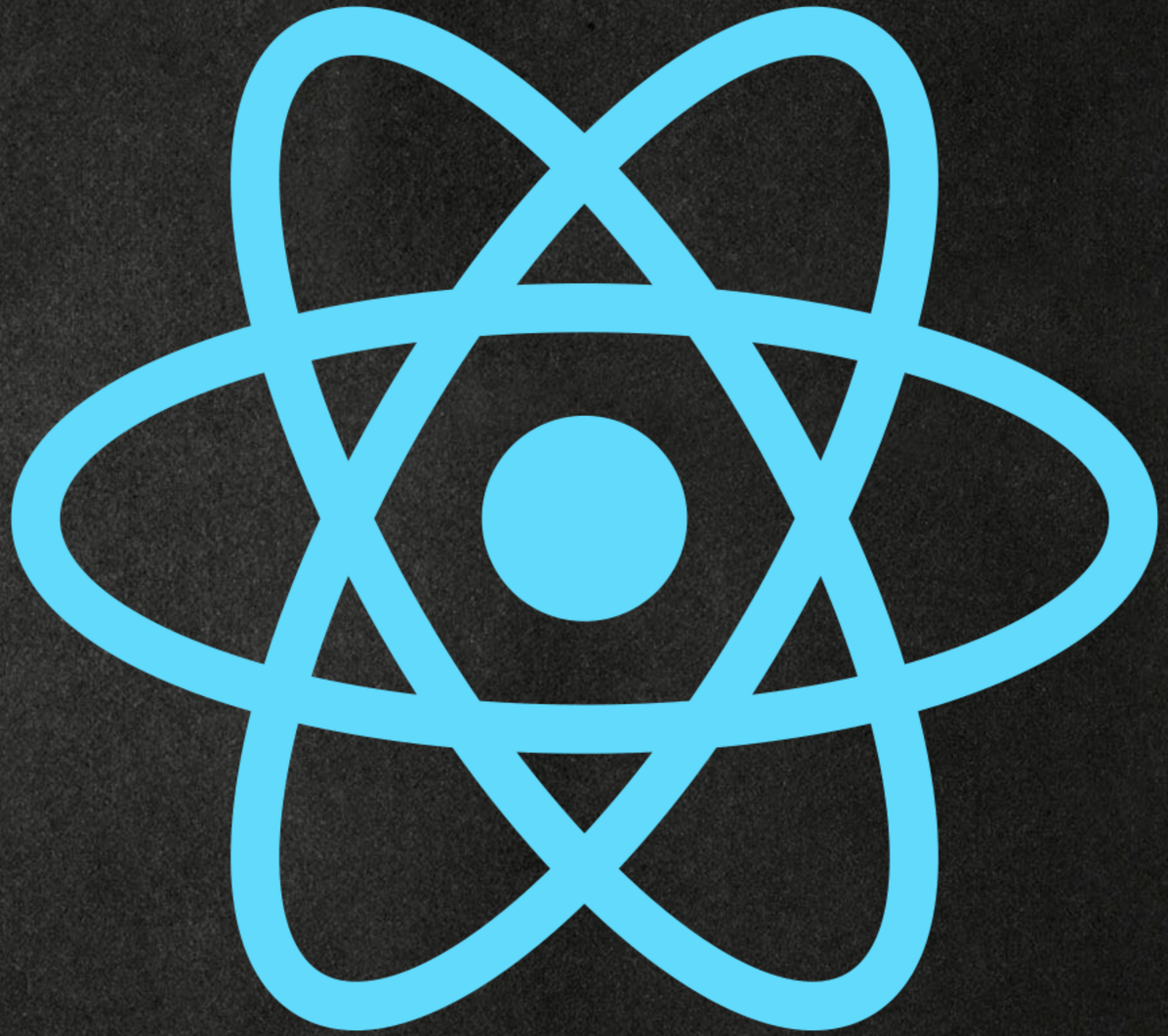
SECTION No. 0

Today's Agenda

1. What's React
2. Node, NPM, & Yarn
3. Dev Environment
4. ES6
5. Promises

Today's Agenda

6. Components
7. JSX
8. State
9. HMR



Today's Agenda

10. API Access
11. Lodash

Today’s Agenda

In the morning,
I’m going to talk
a lot …

Today's Agenda

In the afternoon, you're going to code a lot …

WHAT IS REACT?

SECTION No. 1

What is React?

- React is a JS View Framework
- Declarative
- SPA, Mobile, and Desktop
- Not Opinionated
- Composed of many small pieces
- Focused on quick build cycles

What is React?

NODE, NPM, & YARN

SECTION No. 2

Node

- NodeJS is a popular, javascript based, cross platform runtime
- Often used to “host” react apps (could be .Net, Rails, Webpack, etc …)
- Not much to it OOB, but it has a HUGE community …

NPM

- NPM stands for Node Package Manager
- Ships as part of node
- Same purpose as Nuget, Gems, CocoaPods, etc
- Allows the program to be built on a series of explicit parts

Yarn

- Yarn is a replacement for the npm cli
- It's commands are pretty much 1:1 with npm (npm install === yarn install)
- It's really, really, really super fast
- It supports parallel installation

DEV ENVIRONMENT

SECTION No. 3

Dev Environment

The great thing about React is it’s not opinionated …

The hardest thing about React is it’s not opinionated …

Editors

How does it hook together?

- package.json
- webpack
- babel
- eslint

package.json

- Part of NPM
- Declares all of the dependencies
- Can contain name “scripts”

webpack

- code “bundler”
- lots of plugins
- can be obtuse

babel

- code “transpiler”
- allows us to use ES6 / ES7 features now

eslint

- code analysis tool
- enforces language requirements
- enforces team style conventions

Let’s take a look …

ES6

SECTION No. 4

ES6

- “Modern” JavaScript
- Lot’s of syntactic sugar
- Most React samples are now written in ES6

ES6 - cosnt & let

```
var foo = ‘foo’;
```

```
const foo = ‘foo’;

let bar = ‘bear’;
bar = ‘bar’;
```

ES6 - Function

```
function foo() {}
```

```
const foo = () => {}
```

ES6 - String Interpolation

```
var greeting = ‘Hello ‘ + foo + ‘ !!!’;
```

```
const greeting = `Hello ${foo} !!!`;
```

ES6 - Property Shorthand

```
var args {a: a, b: b, c: c};
```

```
const args = {a, b, c};
```

ES6 - Spread Operator

```
var args = [0, 1];
var args2 = args.concat(2);
```

```
const args = [0, 1];
const arg2 = [...args, 2];
```

ES6 - Spread Operator

```
var args = {a: 1, b: 2, c: 3};

var args2 = {
 a: args.a,
 b: args.b,
 c: args.c,
 };
```

```
const args = {a: 1, b: 2, c: 3};

const args2 = {...args};
```

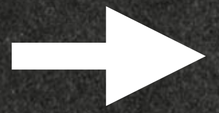
ES6 - Object Desctructuring

```
var args = {a: 1, b: 2, c: 3};

var a = args.a;
var b = args.b;
var c = args.c;
```

```
const args = {a: 1, b: 2, c: 3};

cosnt {a, b, c} = args;
```

ES6 - Default Parameters

```
function foo(x) {
 if (!x) {
  x = 1;
 }
}
```



```
const foo = (x = 1) => {
};
```

ES6 - Classes

```
var Shape = function(x, y) {
 this.x = x;
 this.y = y;
};

Shape.prototype.move =
function (x, y) {
 this.x = x;
 this.y = y;
};
```

```
class Shape {
 constructor(x, y) {
  this.move(x, y);
 }

 move(x, y) {
  this.x = x;
  this.y = y;
 }
};
```

ES6 - Inheritance

```
var Rectangle = function(x, y, width, height) {
    Shape.call(this, x, y);
    this.width  = width;
    this.height = height;
};

Rectangle.prototype =
 Object.create(Shape.prototype);

Rectangle.prototype.constructor = Rectangle;
```

```
class Rectangle extends Shape {
 constructor(x, y, w, h) {
  super(x, y);

  this.width = w;
  this.height = h;
 }
};
```

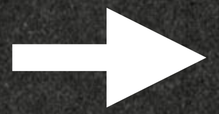
ES6 - Export default

```
module.exports = function() {
};
```

```
export default () => {};
```

ES6 - Export named

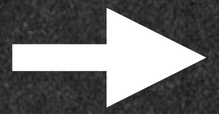
```
var foo = function() {};
foo.bar = function() {};

module.exports = foo;
```



```
export const bar = () => {};
```

ES6 - Import

```
var React = require('react');
var Component = React.Component;
```



```
import React, {Component} from 'react';
```

SECTION No. 5

PROMISES

What are Promises

- Promises are a pattern for writing asynchronous code
- Say for example, we need to make a call to an api ...

Old Way - Callbacks

```
getJson('http://www.google.com', (json) => {
  parseTheJson(json, (objects) => {
    doSomeCalculations(objects, (results) => {
      diplayResults(results);
    });
  });
});
```

Old Way - Callbacks with Error Handling

```
try {
  getJson('http://www.google.com', (json) => {
    try {
      parseTheJson(json, (objects) => {
        try {
          doSomeCalculations(objects, (results) => {
            try {
              diplayResults(results);
            } catch(e) {}
          });
        } catch(e) {}
      });
    } catch(e) {}
  });
} catch(e) {}
```

What are Promises

- This is referred to as “callback hell”
- Promises allow you to “chain” the calls together, in order, and have a single catch
- So …

Promises

```
getJson('http://www.google.com')
  .then(parseTheJson)
  .then(doSomeCalculations)
  .then(displayResults)
  .catch(logError);
```

Promises

- Most times you will simply be consuming promises from someone else's code (say an web client, disk access, etc ...).
- Occasionally though, you will need to create a promise, here's how ...

Promises

```
new Promise((resolve, reject) => {
  try {
    getJson('http://www.google.com', (json) => {
      resolve(json);
    });
  } catch(e) {
    reject(e);
  }
});
```

Break Time

COMPONENTS

SECTION No. 6

What are Components

- Components are the building blocks of React
- Components render some part of the view
- Components are made of other components which are made up of other components which are ...

Components

Functional or Pure Component

```
export default () => {
  <div>Hello World</div>
};
```

Class based Component

```
export default class Hello extends Component {
  render() {
    return (
      <div>Hello World</div>
    );
  }
}
```

Component Lifecycle

- Only class based components can currently take advantage of lifecycle events (change is coming ...)
- componentDidMount
- componentDidUnmount
- componentWillReceiveProps
- shouldComponentUpdate
- componentWillUpdate
- render
- componentDidUpdate

Handling Events

```
export default class Hello extends Component {
  handleClick() {
   alert('Hello World');
  }

  render() {
    return (
      <button onClick={this.handleClick}>
       Click Me
      </button>
    );
  }
}
```

Components - Lists

```
export default class Foo extends Component {
  renderList() {
   const data = [1, 2, 3, 4, 5];
   return data.map(i => <li key={i}>{i}</li>);
  }

  render() {
    return (
      <ul>{this.renderList()}</ul>
    );
  }
}
```

Handling Events - List

```
export default class Foo extends Component {
  renderList() {
    const data = [1, 2, 3, 4, 5];
    return data.map(i => {
      const handleClick = () => {
        doSomething(i);

        return (
          <li key={i} onClick={handleClick}>{i}</li>
        );
  }

  render() {
    return (
      <ul>{this.renderList()}</ul>
    );
  }
}
```

Components

```
class Foo extends Component {
  render() {
    return (
      <div>Hello World</div>
    );
  }
}

export default () => {
 return (
  <Foo />
 );
};
```

Props

- Following the pattern of all HTML components, React components can have properties (shortened to props).
- Props can be used to pass data & handlers from parent to child

Props

```
const Foo = ({target}) => {
 return (
  <div>Hello {target}</div>
 );
};

export default () => {
 return (
  <Foo target="World" />
 );
};
```

Props

```
class Foo extends Component {
  render() {
    return (
      <div>Hello {this.props.target}</div>
    );
  }
}

export default () => {
 return (
  <Foo target="World" />
 );
};
```

JSX

SECTION No. 7

What is JSX

- If we take a look again at the functional component - it looks a lot like HTML

Functional or Pure Component

```
export default () => {
  <div>Hello World</div>
};
```

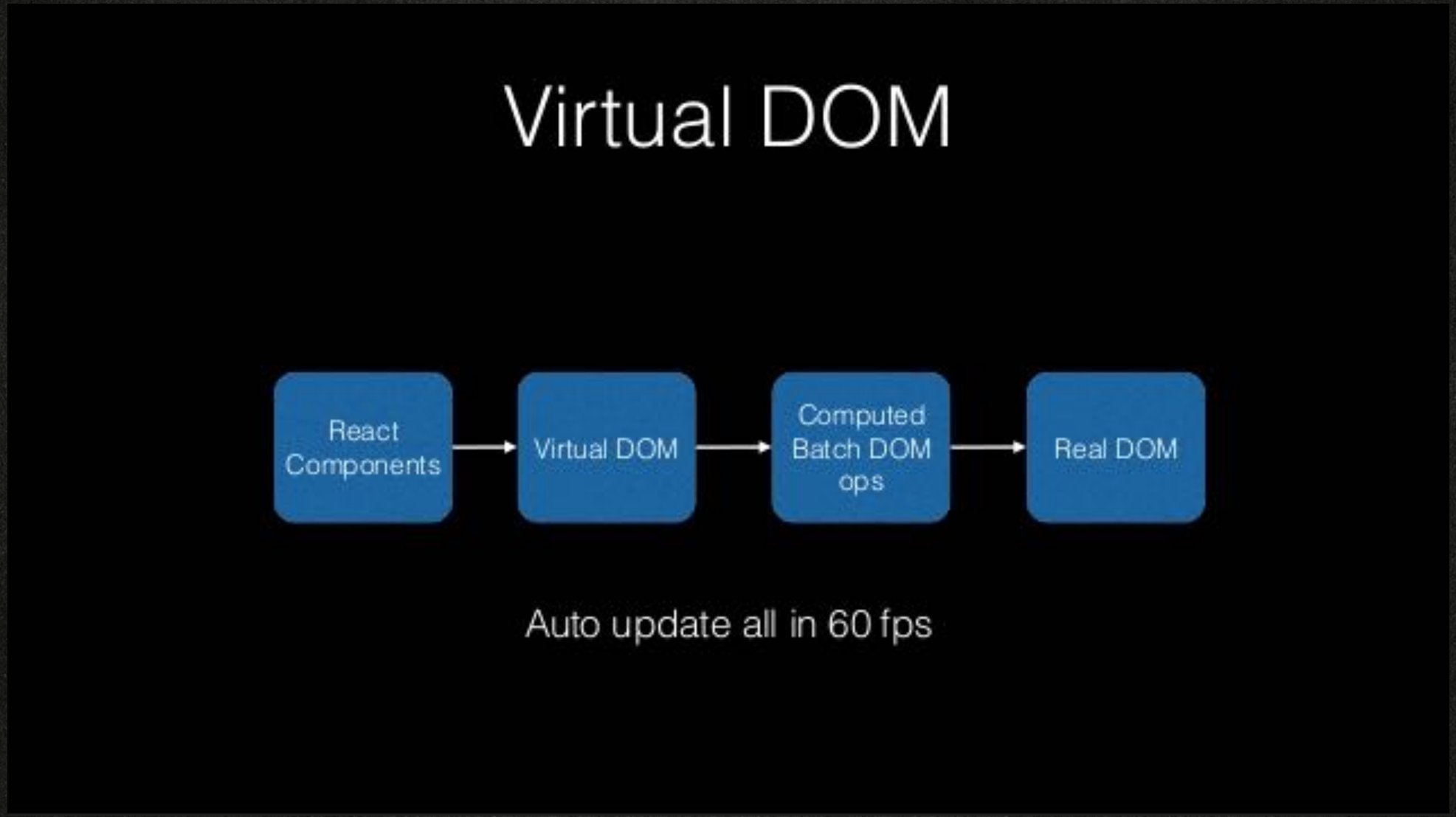
What is JSX

- BUT its not
- When writing websites, we use React-Dom
- React-Dom gives us components named the same as all of the standard HTML

What is JSX

- React then takes those components and maintains the “shadow dom” / “virtual dom”.
- React batches changes to the shadow dom, and then makes a single batch change to the actual dom

What is JSX

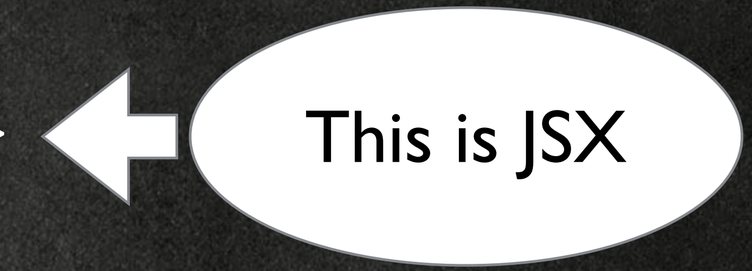

Virtual DOM
React Components
Virtual DOM
Computed Batch DOM ops
Real DOM
Auto update all in 60 fps

Why?

- Everything can live in 1 file
- You get to use JS to generate objects, rather than strings attached to html

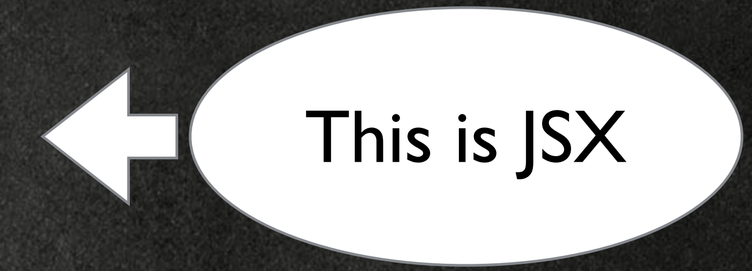
JSX

```
export default () => {
  <div>Hello World</div>
};
```



JSX

```
export default class Hello extends Component {
  render() {
    return (
      <div>Hello World</div>
    );
  }
}
```



STATE

SECTION No. 8

Where's the data

- Data lives in two places in React - props & state.
- Props is data that is passed to the component
- State is data that is contained "inside" the component
- The word "state" is a convention of React

State

```
export default class Hello extends Component {
  constructor() {
   this.state = {target: 'World'};
  }

  render() {
    return (
      <div>Hello {this.state.target}</div>
    );
  }
}
```

State vs Stateless

- Maintaining state can be expensive for both memory and maintainability - so think carefully about what owns the state and what can just be passed something to render

State

```
export default class Hello extends Component {
  constructor() {
   this.state = {target: 'World'};
  }

  render() {
    return (
      <div>Hello {this.state.target}</div>
    );
  }
}
```

Stateless

```
const Foo = ({target}) => {
 return (
  <div>Hello {target}</div>
 );
};

export default () => {
 return (
  <Foo target="World" />
 );
};
```

How to update state?

- Your first impulse is probably to say this.state.foo = “bar”;
- This introduces some problems though:
- How does the UI know to update?
- How does anything else dependent on foo know to update?
- Setter code is strewn throughout the codebase

How to update state?

- There were A LOT of frameworks written to try to answer the complexity of this approach, and most used a pattern named “two way databinding”
- Simply … the framework would introduce code to wrap getters, setters, and manage the dependency tree … so any time something updated a series of updates would go out

How to update state?

- This led to a lot of brittle code that was really difficult to reason about
- React forces you to go away from this pattern and use Unidirectional Data Flow

Unidirectional Data Flow

- Simplify the flow into something that can be read and reasoned about
- Eliminate side effects
- Separation of logic

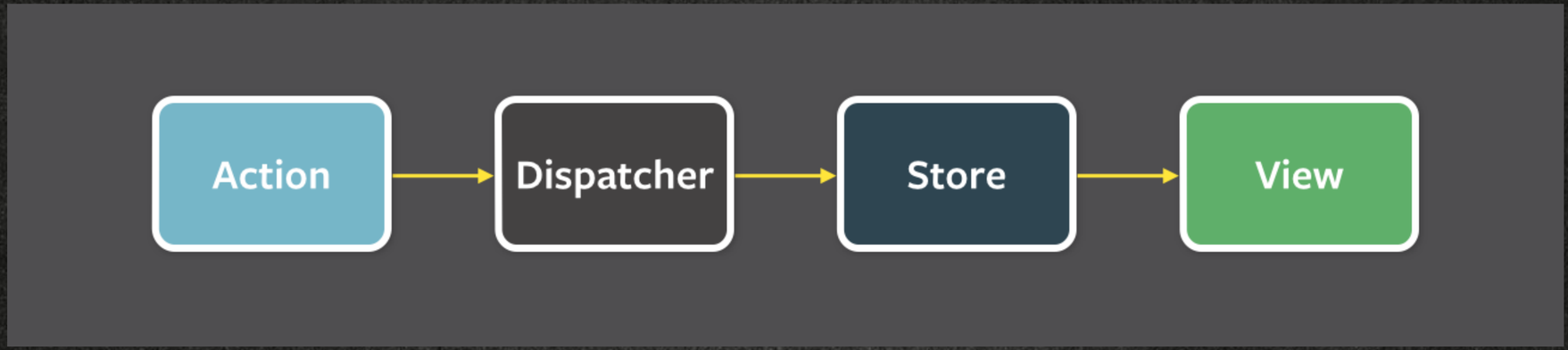
Unidirectional Data Flow

- Never update the view (i.e. label.text = “”)
- Never update the state (i.e. state.foo = “”)
- You tell react you have “new state”, and it handles the updating of everything for you

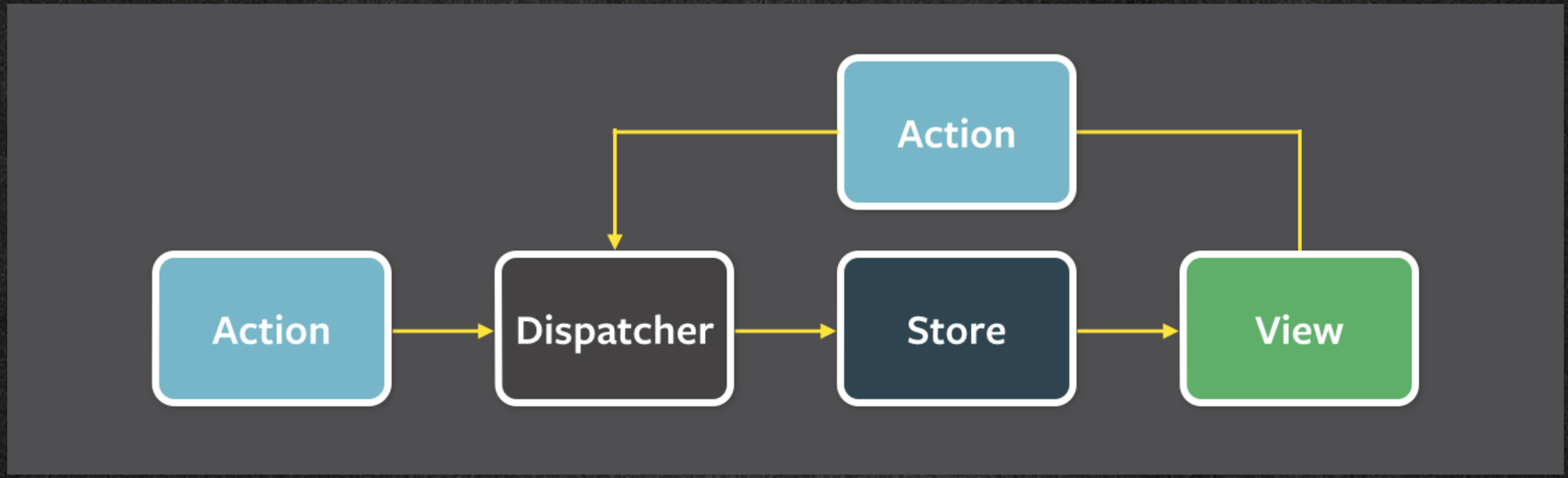
Unidirectional Data Flow

- Action = some event that changes state (page load, button click, etc ...)
- Dispatcher = the function you tell about the state change
- Store = React's copy of the "state"
- View = the result of your render method

Unidirectional Data Flow



Unidirectional Data Flow



Demo time …

Break Time

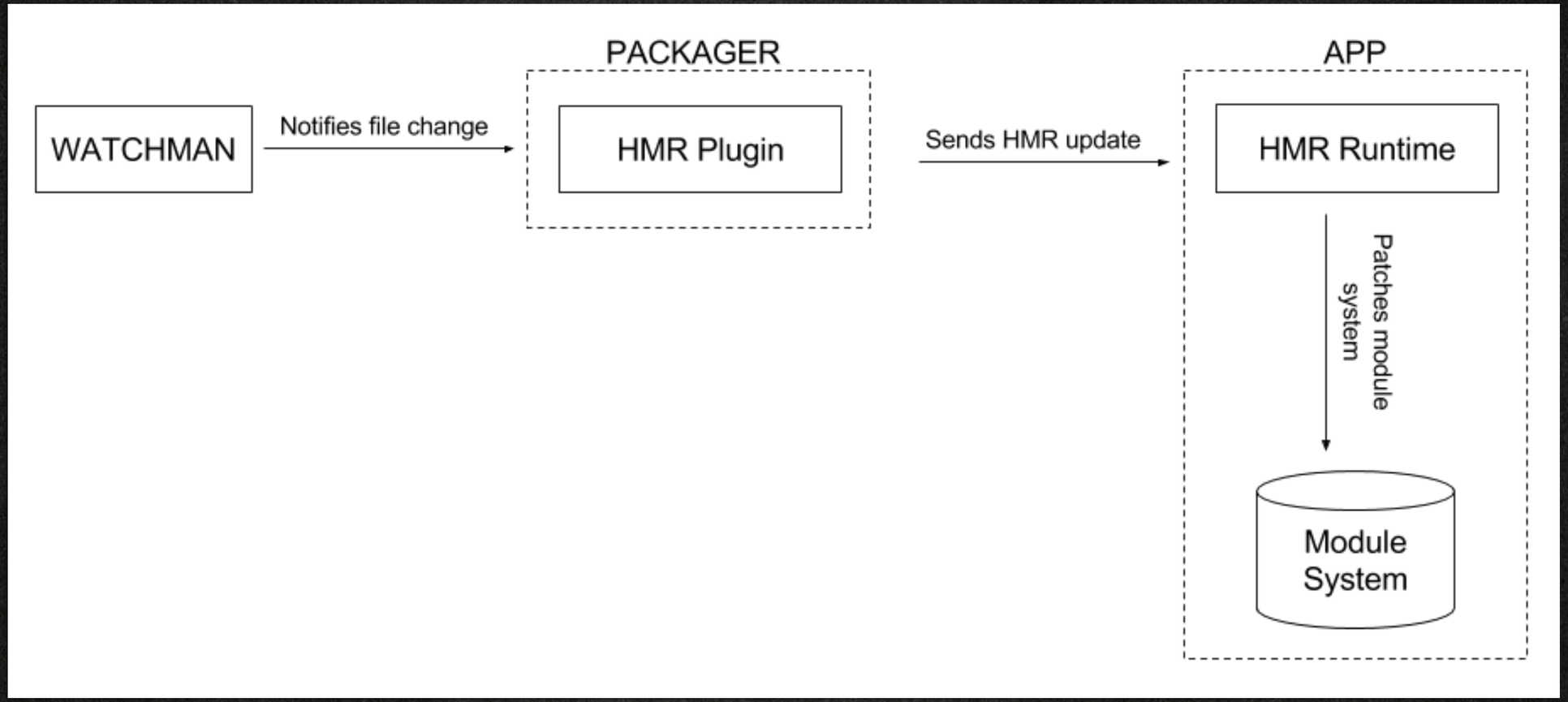
HMR

SECTION No. 9

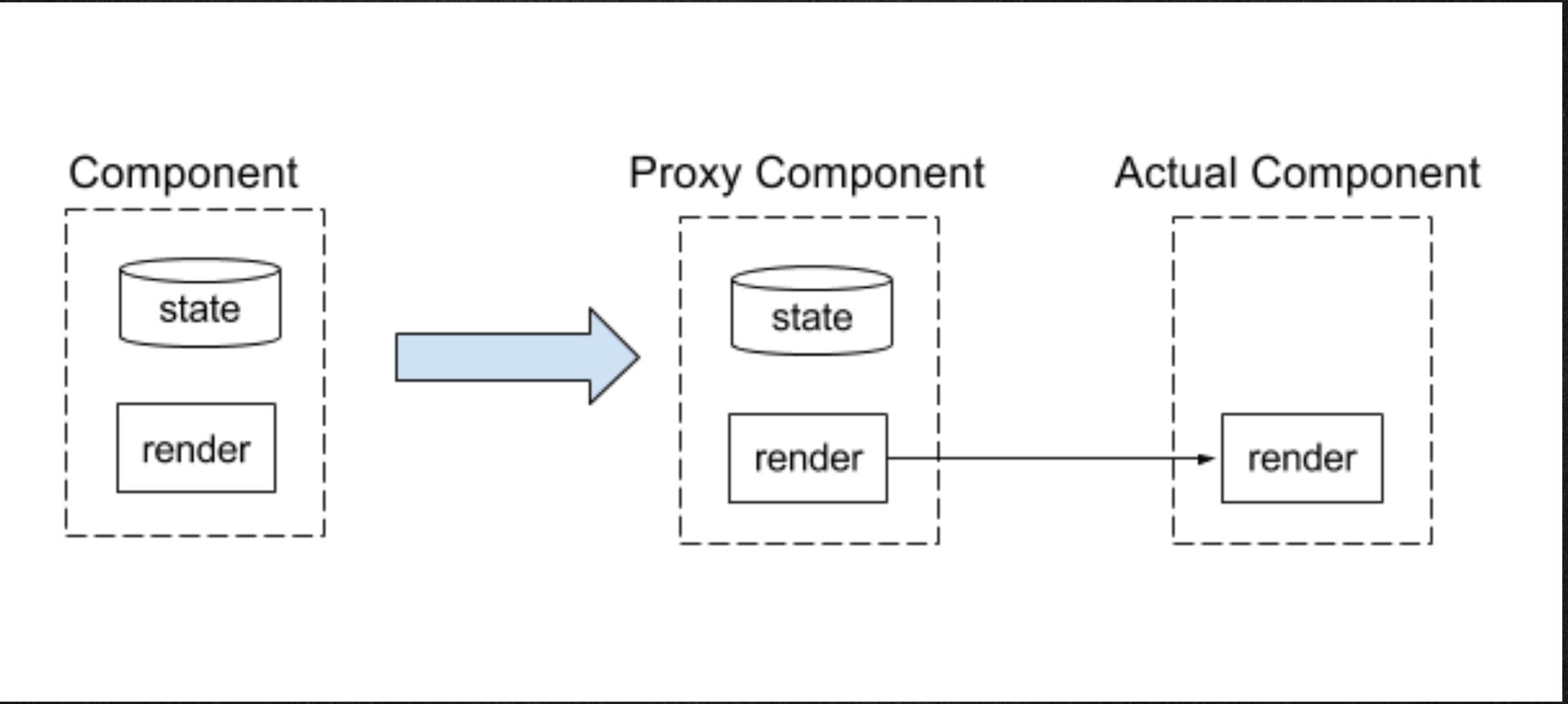
HMR

- HMR stands for Hot Module Replacement (or Reload)
- It's the reason I fell in love with React
- It allows code and styles to be updated at runtime live, without reloading the page

HMR


PACKAGER
APP
WATCHMAN
Notifies file change
HMR Plugin
Sends HMR update
HMR Runtime
Patches module system
Module System

HMR


Component
state
render
Proxy Component
state
render
Actual Component
render

Demo time …

API ACCESS

SECTION No.

API Access

- Remember React is a view framework ...
- You need to get your data to display from somewhere ... and that where is almost always APIs.

Fetch

- JS provides a simple way to make a API call
- fetch(‘http://www.google.com')
- returns a promise

Fetch / Request

- Sometimes the request needs more than just an address (headers, cors, different verb, etc)
- Fetch can also accept a Request ...

Fetch / Request

```
var myHeaders = new Headers();

var myInit = { method: 'GET',
               headers: myHeaders,
               mode: 'cors',
               cache: 'default' };

var myRequest = new Request(‘http://www.google.com', myInit);

fetch(myRequest)
.then(function(response) {
  console.log(response.data);
});
```

Fetch / Request

- If you have to do this a lot though …
- This type of boilerplate is where the ecosystem of NPM really shines …

Axios

```
axios({
 method: ‘post’,
 url: ‘http://www.google.com",
 data: { search: ‘foo’ },
}).then((response) => {
 console.log(response.data);
});
```

LODASH

SECTION No. 11

Lodash

- lodash is a series of utility functions
- Arrays: take, drop, concat, filter, find, flatten, …
- Functions: after, before, curry, delay, once, throttle, …
- Lanaguage: isArray, isObject, isString, conformsTo, …
- Math: min, max, minBy, maxBy, sum, …
- Number: inRange
- Object: assign, forOwn, merge, omit, …
- Strings: startsWith, endsWith, toLower, toUpper, …
- Util: attempt, flow, mixin

Lodash

```
import _ from 'lodash';

const array = [1, 2, 3];

export default () =>
  _.filter(array, x => x > 1).map(x => <li>{x}</li>);
```

Demo time …

Now it's your turn ...