DAY 1: FOUNDATION

React

SECTION No.

0

WHAT ARE WE COVERING TODAY

WHAT ARE WE COVERING TODAY

1. Overview of React
2. Node / NPM
3. Dev Environment
4. ES6
5. JSX
6. Components
7. State
8. HMR
9. Webpack
10. Lodash
11. Promises
12. API Access

ITS GOING TO BE A FAST DAY

CHAPTER No.

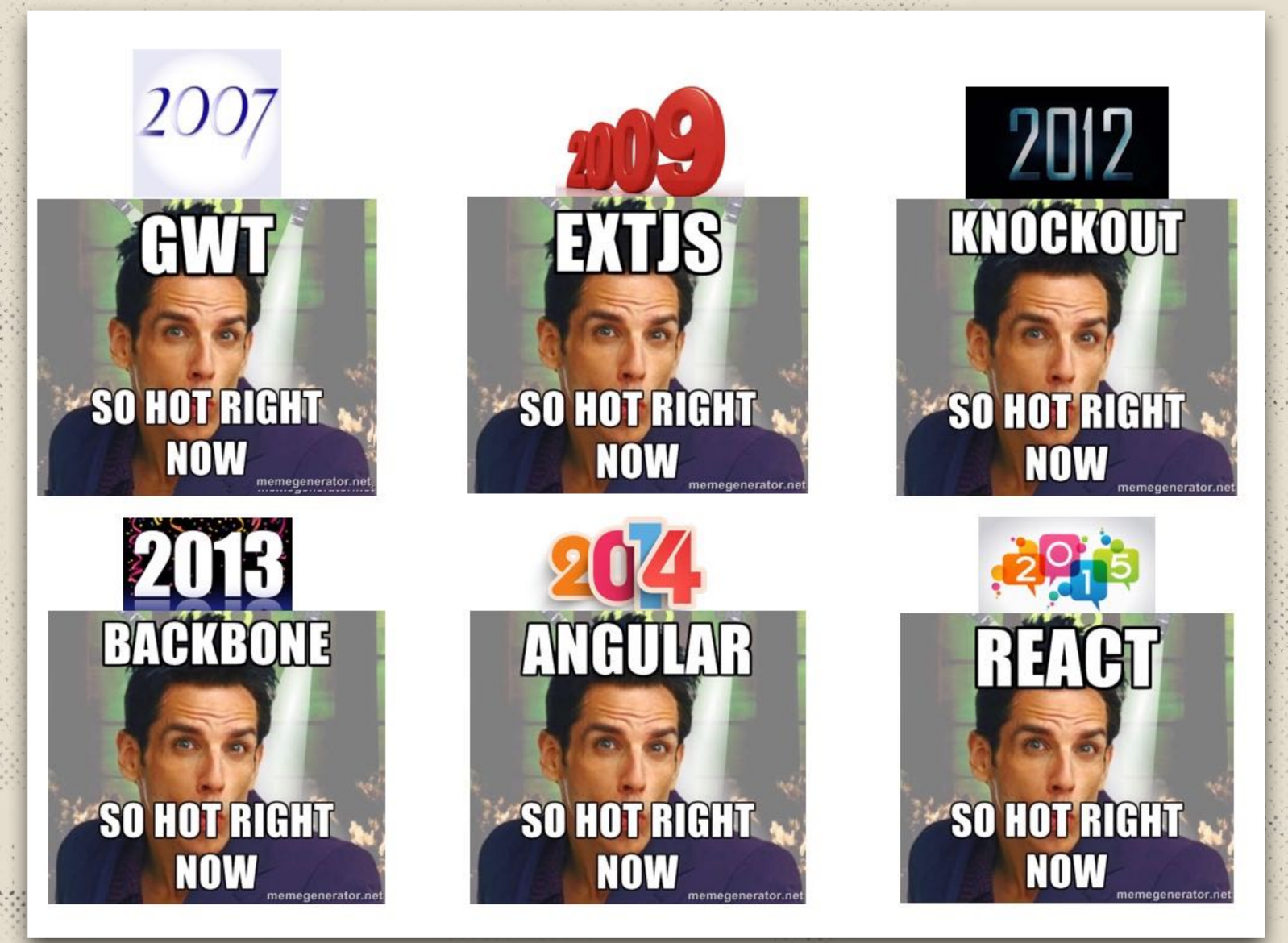
WHAT IS REACT?

WHAT IS REACT?

★ ★ ★ ★ ★ ★ ★ ★ ★

- React is a Javascript view framework

NOT ANOTHER FRAMEWORK



WHAT IS REACT?

★ ★ ★ ★ ★ ★ ★ ★ ★

- ★ React is a Javascript view framework
- ★ Declarative
- ★ SPA
- ★ Composed of components
- ★ Not opinionated
- ★ Can be used to build web and mobile
- ★ Focuses on enabling quick build cycles

CHAPTER No. 2

NODE / NPM

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
- ★ React, ReactDom, ReactNative are all NPM packages

A QUICK RABBIT TRAIL

CHAPTER No. 3

DEV ENVIRONMENT

EDITOR

Since JS files
are just text ...

HOW DOES IT WORK TOGETHER?

★ ★ ★ ★ ★ ★ ★ ★ ★

- ★ package.json
- ★ webpack
- ★ eslint

DEMO / LAB

CHAPTER No. 4

ESS

ES6

★ “Latest” version of JS, first update since 2007
★ Mixed support levels … Babel
★ Lots of syntactic sugar
★ Most React samples will be written in ES6
★ ES7 … async / await

ES6 – CONST / LET

```
var foo = ‘foo’;
```

```
const foo = ‘foo’;

let bar = ‘bar’;
bar = ‘bear’;
```

ES6 – FUNCTION

```
function foo() {
 console.log('foo');
}
```

```
() => {
 console.log('foo');
}
```

ES6 – STRING INTERPOLATION

```
‘Hello ’ + foo + ‘ World’;
```

```
`Hello ${foo} World’;
```

ES6 – SPREAD OPERATOR

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

```
const args = [0, 1];
const second = [ …args, 2];
```

ES6 – SPREAD OPERATOR

```
var person = {
 fName: ‘Jon’,
 lName: ‘Smith’,
};

var jr = {
{
 fName: person.fName,
 lName: person.lName,
};
```

```
const person = {
 fName: ‘Jon’,
 lName: ‘Smith’,
 };

const jr = { …person };
```

ES6 – PROPERTY SHORTHAND

```
var obj = { x: x, y: y };
```

```
const obj = { x, y };
```

ES6 – DEFAULT PARAMETERS

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

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

ES6 – OBJECT DESTRUCTURING

```
var obj = x.obj;
var foo = x.foo;
var bar = x.bar;
```

```
const { obj, foo, bar } = x;
```

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 – CLASSES

```
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, width, height) {
        super(x, y);
        this.width  = width;
        this.height = height;
    }
}
```

ES6 – IMPORT / EXPORT

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

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

ES6 – IMPORT / EXPORT

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

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

DEMO / LAB


I'D LIKE TO START WITH A CARTOON.
IT'S ABOUT A GUY WHO SHOWS A CARTOON BEFORE GIVING A BORING PRESENTATION.
BUT IT DOESN'T WORK BECAUSE THE CARTOON HAS NO PUNCHLINE.
www.dilbert.com scottadams@aol.com
3/4/94 © 1998 United Feature Syndicate, Inc.

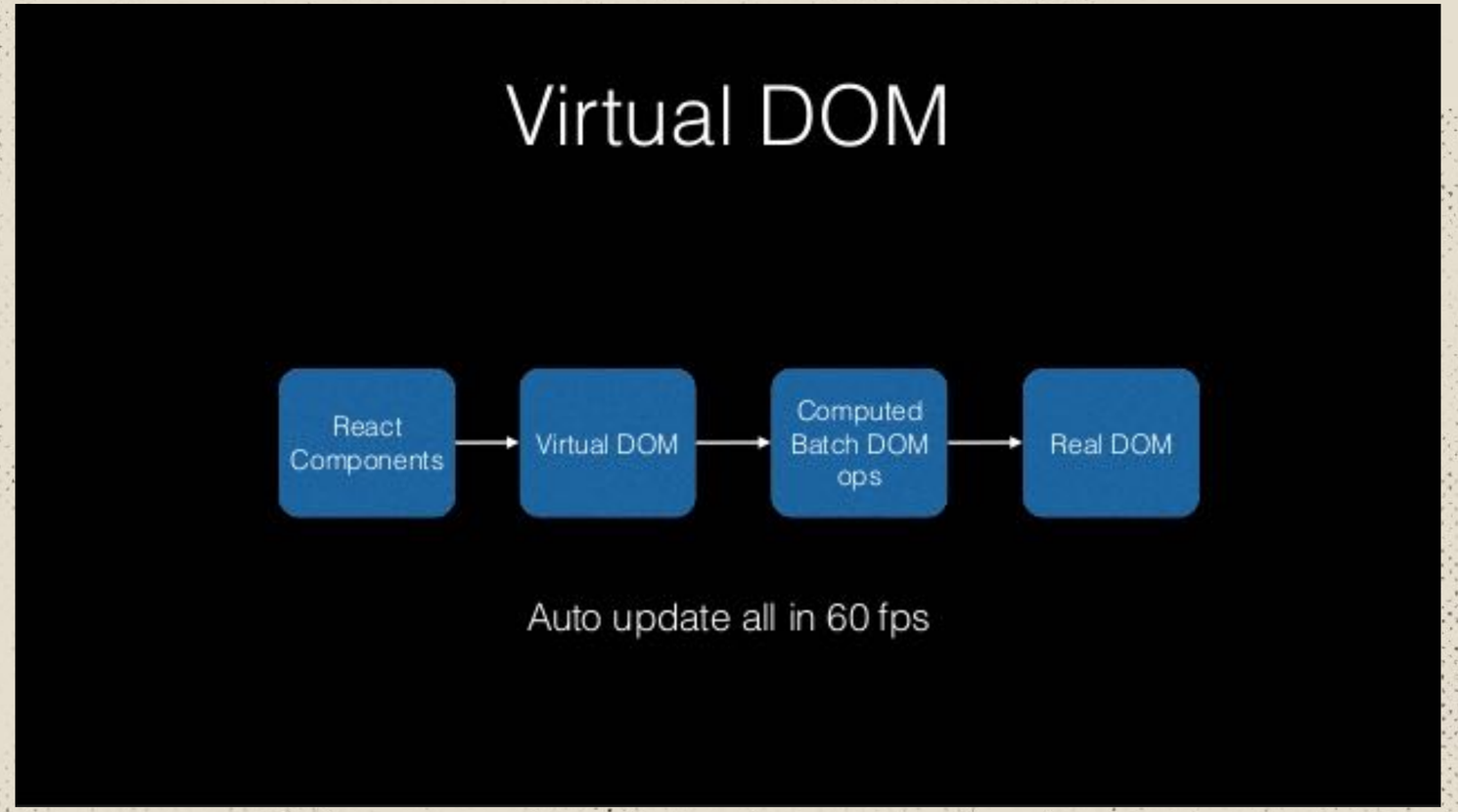
CHAPTER No. 5

SIX

JSX

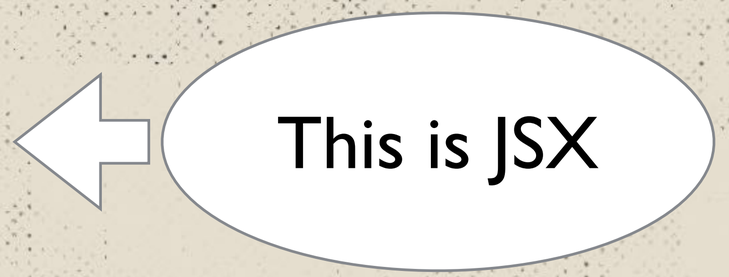
- JSX is how we express the eventual view
- JSX !== HTML element, rather it represents objects
- JSX is interpreted by React into the view

JSX


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

JSX

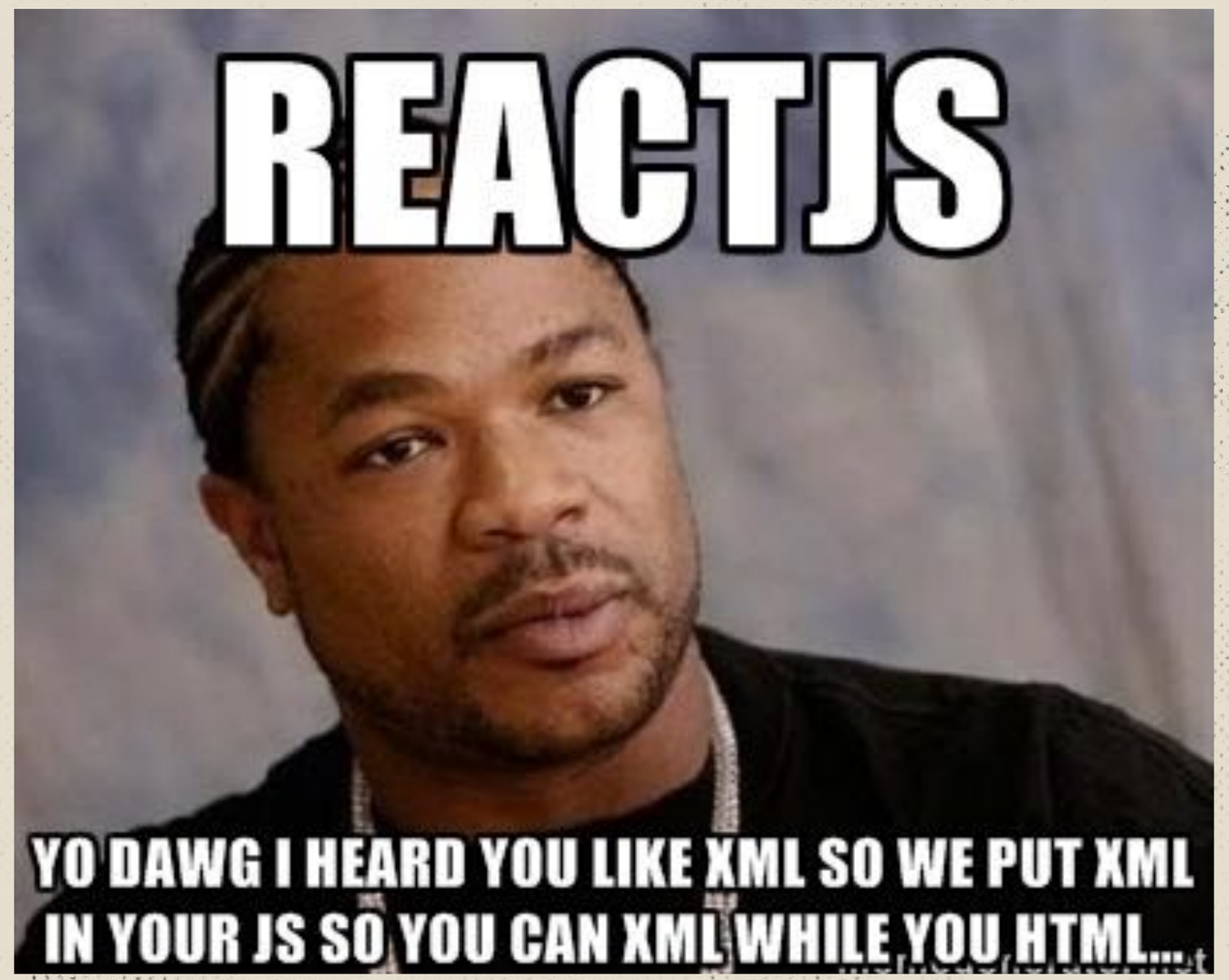
```
export default class Hello extends Component {
  render() {
    return (
      <div>Hello World</div>
    );
  }
}
```



JSX

```
export default class Hello extends Component {
  render() {
    return (
      <Row>
        <Col xs={12}>
          <ul className="list">
            {
              list.map((item) => {
                <li>{item.name}</li>
              }
            }
          </ul>
        </Col>
      </Row>
    );
  }
}
```

JSX


REACTJS
YO DAWG I HEARD YOU LIKE XML SO WE PUT XML IN YOUR JS SO YOU CAN XML WHILE YOU HTML...

JSX

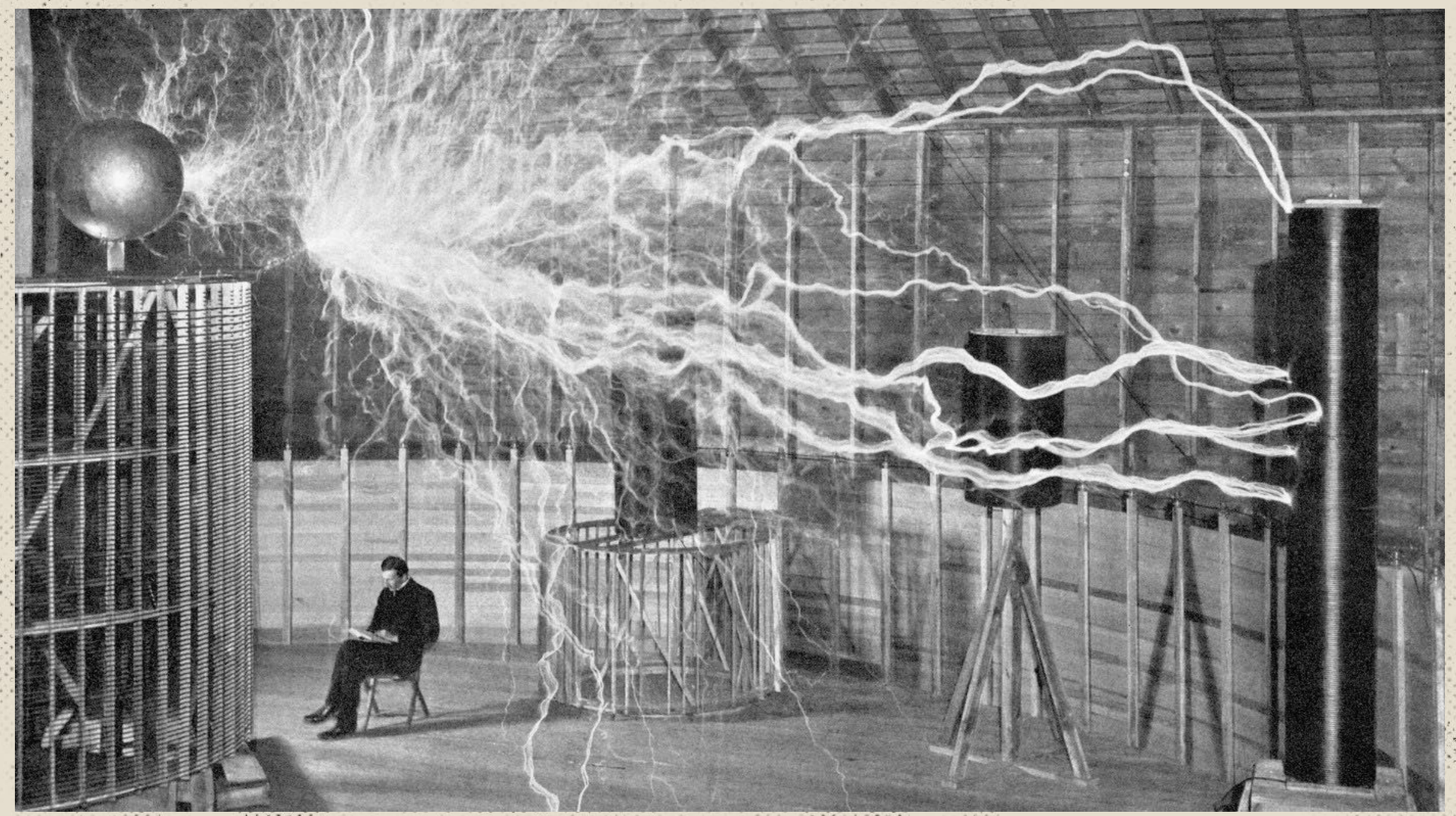
JSX


Everything is going to be all right.
Everything is going to be all right.
Everything is going to be all right.
Everything is going to be all right.
Everything is going to be all right.
Everything is going to be all right.

JSX

- You get to use JS that you know rather than 3rd party DSL
- Descriptive: data and code live together
- Dev time support can be powerful because JSX elements are object

DEMO / LAB



CHAPTER No. 6

COMPONENTS


COMPOSE
ALL THE THINGS!

COMPONENTS

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

COMPONENTS

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

EVENTS

- Certain components expose events, for example the click of a button
- Just like in html, you can attach a function to handle these events

EVENTS

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

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

LIFECYCLE

- Class based components can take advantage of the lifecycle events in the base component
- componentDidMount
- componentDidUnmount
- componentWillReceiveProps
- shouldComponentUpdate
- componentWillUpdate
- render
- componentDidUpdate

CONTAINERS VS COMPONENTS

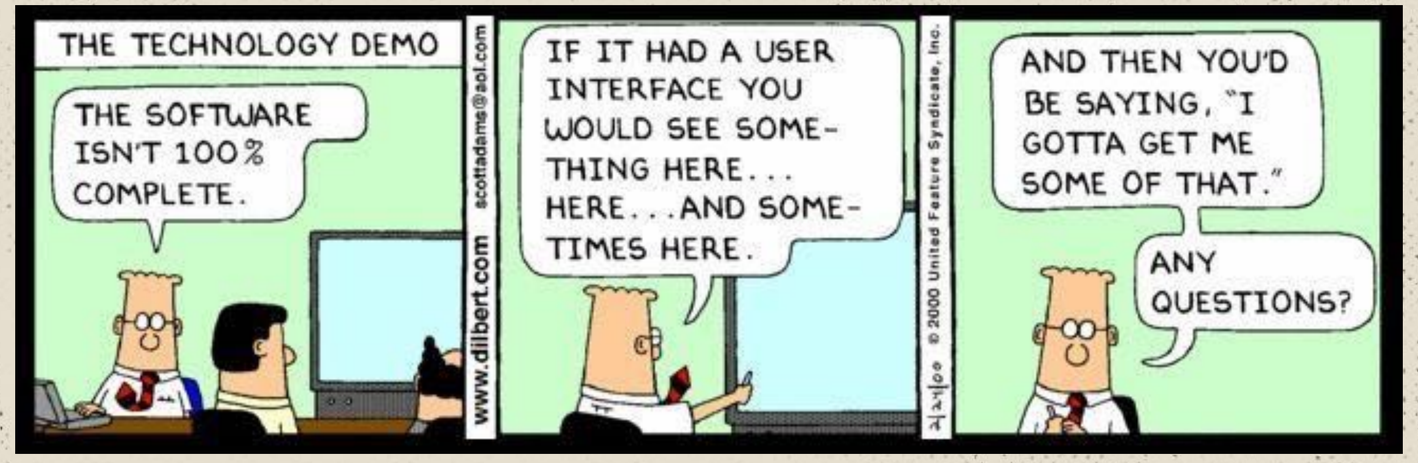
- Components that load and manager their own state are sometimes referred to as “Containers” or “Smart Components”
- Components that are dependent on their parents to pass in their state are referred to as “Components” or “Dumb Components”
- Some people will split their code base folders by these designations, BUT we haven’t found value in that, so we don’t …

PROPTYPES

★ ★ ★ ★ ★ ★ ★ ★ ★

- Short for property types
- Used to provide context for required and option “attributes” on JSX elements
- Our default linter requires them
- func, string, number, array, object, etc ...
- .isRequired

DEMO / LAB


THE TECHNOLOGY DEMO
THE SOFTWARE ISN'T 100% COMPLETE.
www.dilbert.com scottadams@aol.com
IF IT HAD A USER INTERFACE YOU WOULD SEE SOMETHING HERE... HERE...AND SOMETIMES HERE.
2-23-00 © 2000 United Feature Syndicate, Inc.
AND THEN YOU'D BE SAYING, "I GOTTA GET ME SOME OF THAT."
ANY QUESTIONS?

CHAPTER No.

1

STATE

STATE

★ ★ ★ ★ ★ ★ ★ ★ ★

BECAUSE YOUR DATA
NEEDS TO LIVE SOMEWHERE

PROPS VS STATE

★ ★ ★ ★ ★ ★ ★ ★ ★

- You will see data live in two places in React
- Props are passed into the component
- State is owned and managed by the component

STATE VS STATELESS

- Carefully think about where your state needs to live
- Does the component fundamentally need to own the data or just need to know about it to render
- For example: list items vs button “pressed”
- Default to stateless, then carefully add state as needed

UNIDIRECTION DATA FLOW

★ ★ ★ ★ ★ ★ ★ ★ ★

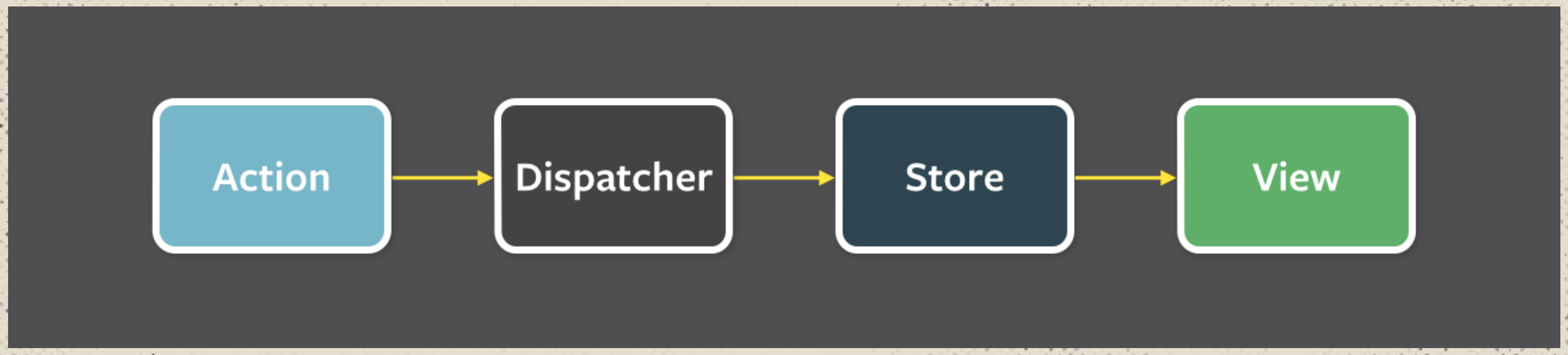
- ★ Simplify data flow
- ★ Eliminates unexpected side effects
- ★ State is explicit
- ★ State is modular
- ★ Forces separation of logic

UNIDIRECTION DATA FLOW

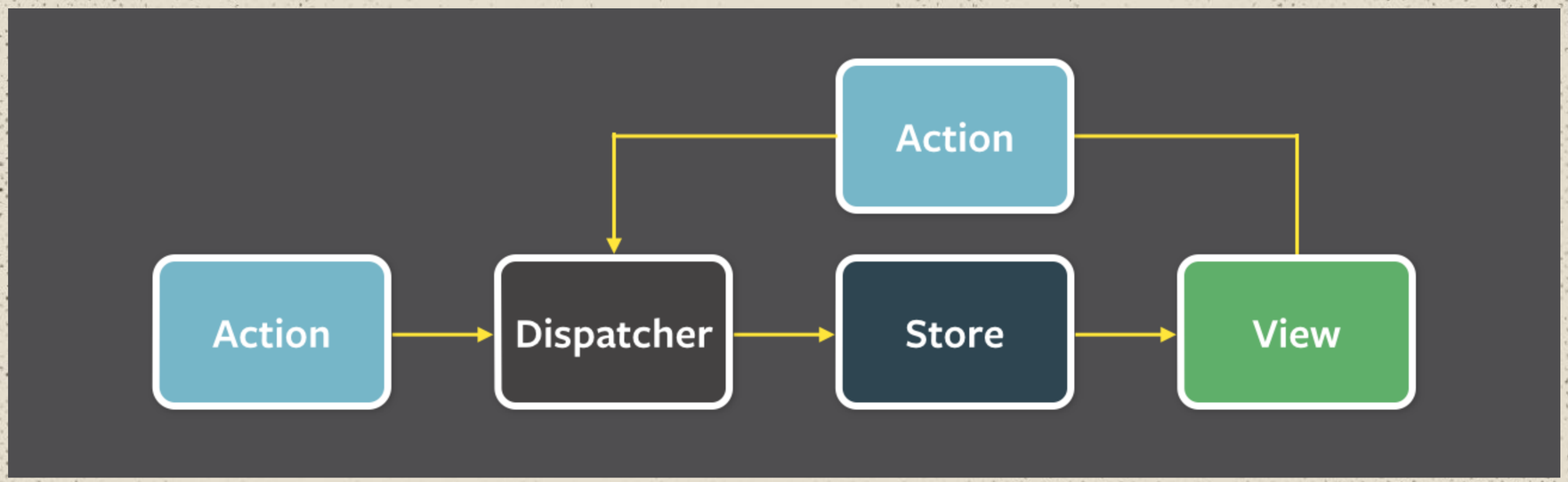
★ ★ ★ ★ ★ ★ ★ ★ ★

- ★ Never update view directly
- ★ Never update state directly
- ★ Emit new state via setState, and the component is notified to re-render

UNIDIRECTIONAL DATA FLOW



UNIDIRECTIONAL DATA FLOW



WHAT DOES 'THIS' MEAN

- Javascript: The Bad Parts
- 'this' can refer to the instance or the global program
- Outside of the lifecycle methods, this refers to the global program, which isn't helpful.
- You need to bind this to the context of the function so that it knows what to do

WHAT DOES 'THIS' MEAN

```
export default class Hello extends Component {
  constructor(props) {
    super(props);

    this.handleClick = this.handleClick.bind(this);
  }

  handleClick() {
    alert(`Hello World ${this.state.count}`);
  }

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

WHAT DOES 'THIS' MEAN

CLASS—AUTOBIND

```
export default class Hello extends Component {
  constructor(props) {
    super(props);
    autobind(this);
  }

  handleClick() {
    alert(`Hello World ${this.state.count}`);
  }

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

DEMO / LAB

CHAPTER No. 8

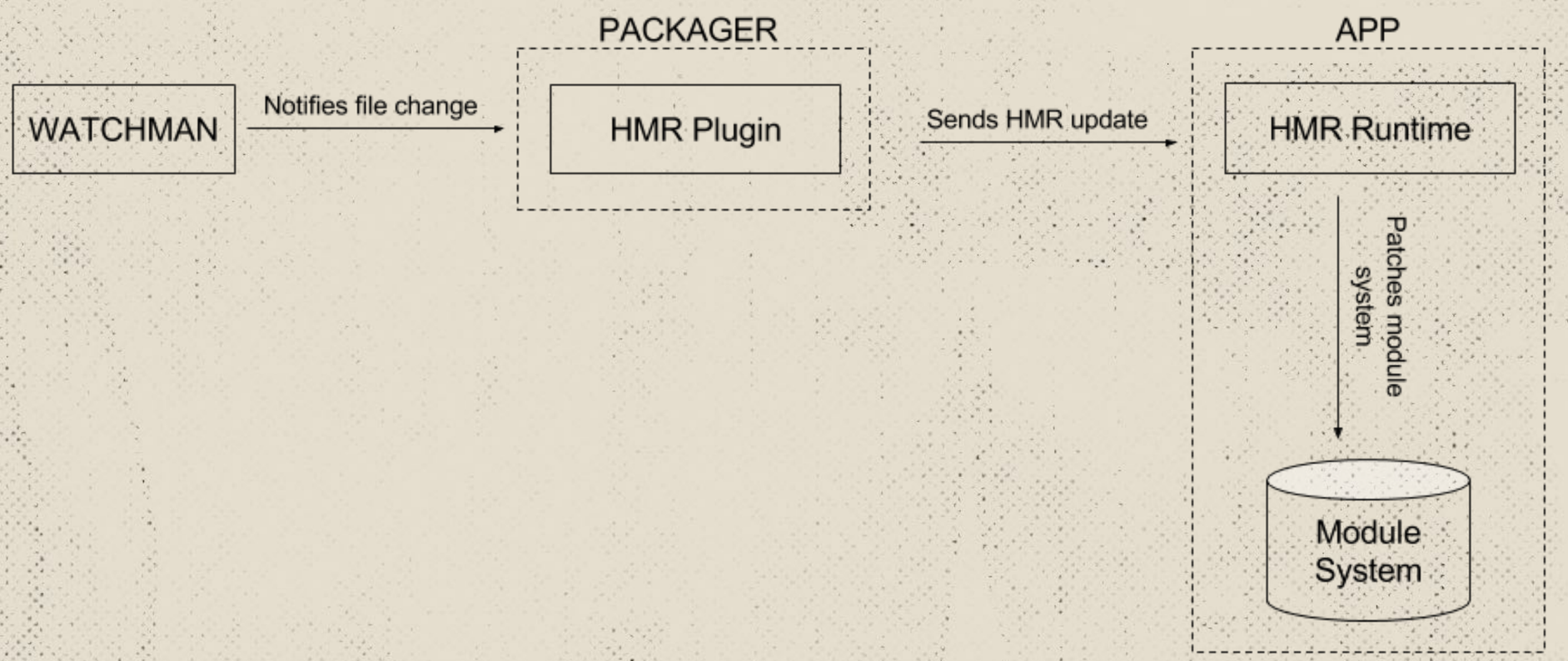
RIVER

HMR

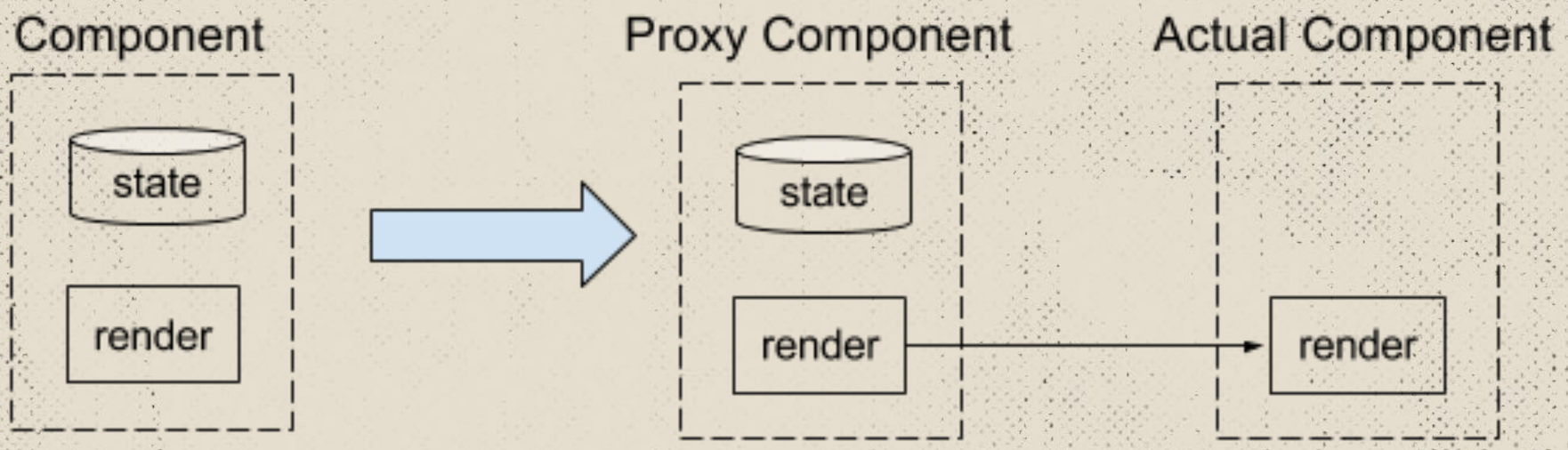
★ ★ ★ ★ ★ ★ ★ ★ ★

- ★ HMR is the reason I fell in love with React
- ★ HMR stands for Hot Module Replacement
- ★ HMR is a feature to inject updated modules into the active runtime.
- ★ It's like LiveReload for EVERY module.

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 / LAB

STATE VS STATELESS

HMR DOESN'T UNDERSTAND FUNCTIONAL COMPONENTS ... SO IT JUST REPLACES THEM

CHAPTER No. 9

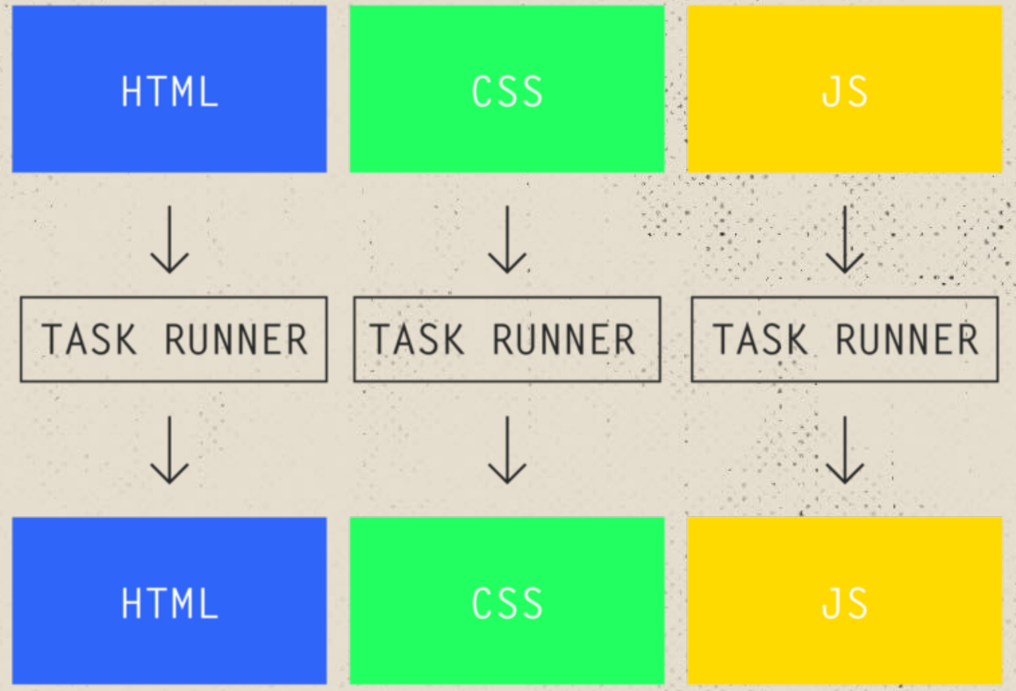
WEBPACK

WEBPACK

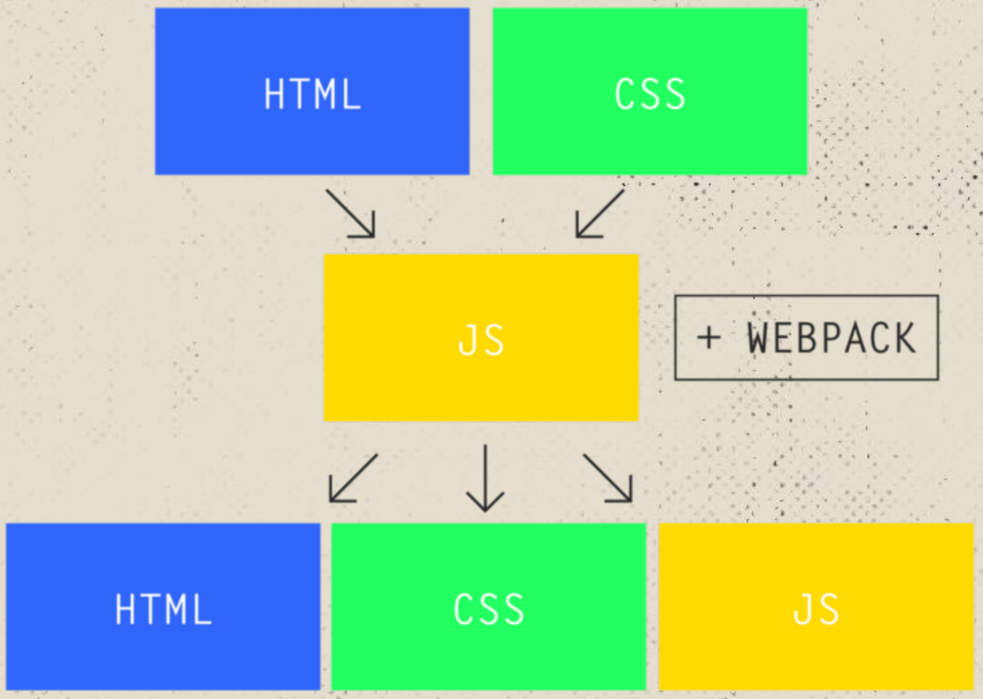
★ ★ ★ ★ ★ ★ ★ ★ ★

- Webpack is at its root a file bundler
- It has plugins to do SOO much more
- Combined with NPM scripts, its is our task runner, bundler, and dev server
- We aren't going to go too deep …
- In short, think of it as a set of inputs gets mapped, parsed, transformed into outputs that are the actual website

GRUNT / GULP



GRUNT / GULP



DEMO / LAB

CHAPTER No. 10

LODASH

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

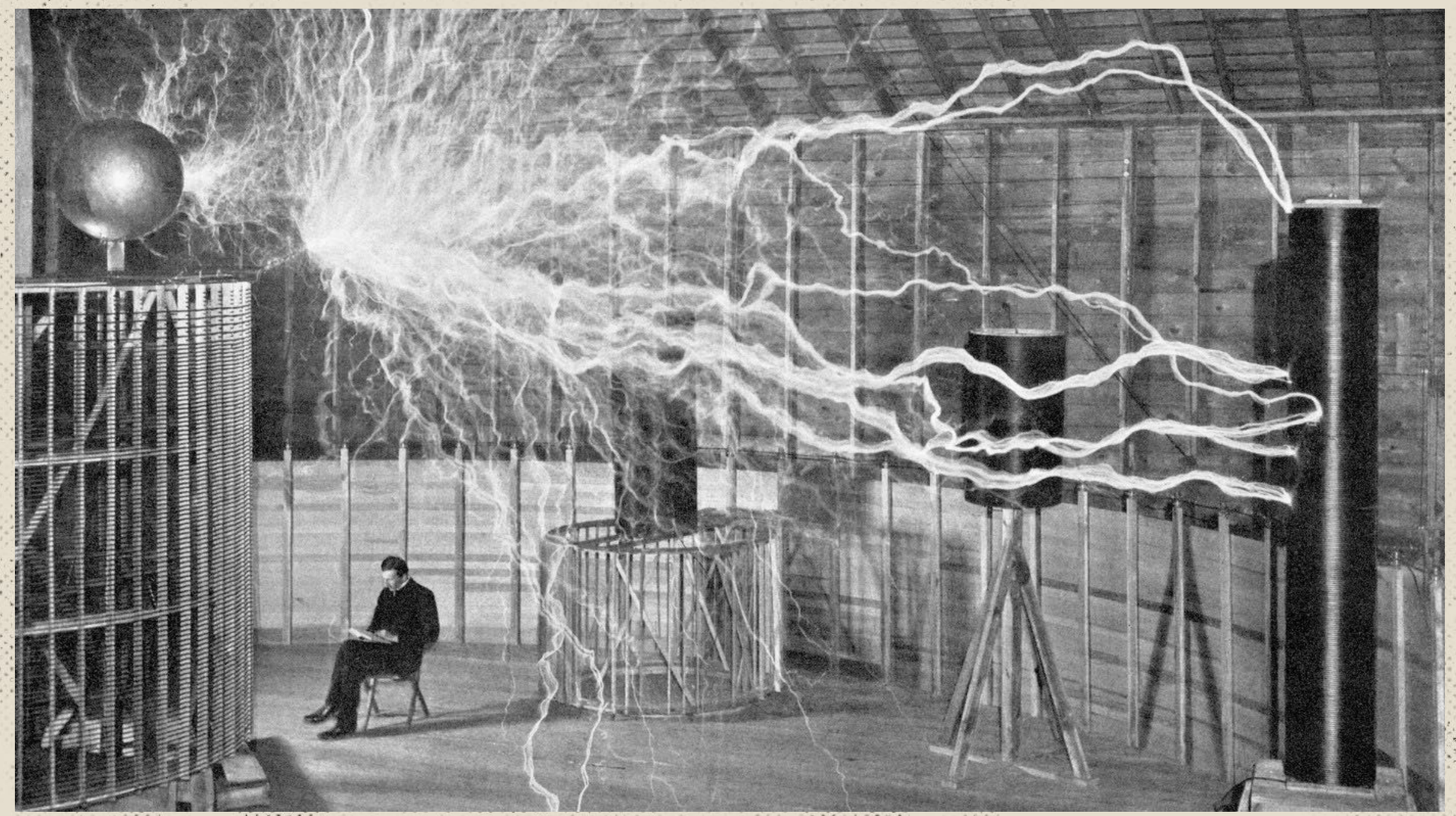
LODASH

```
import _ from 'lodash';

const array = [ 1, 2, 3 ];

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

DEMO / LAB



CHAPTER No.

11

PROMISES

PROMISES – THE PROBLEM

- Javascript is single threaded, so when we need to do things that take a while (like api calls), traditionally we used callbacks
- Unfortunately, this leads to sad code like:

PROMISES – THE PROBLEM

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

OR WORSE . . .

PROMISES – THE PROBLEM

```
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) {
}
```

PROMISES

- Promises simplify the callback “hell” by allowing you to chain the functions together while it manages the flow
- When consuming promises, the pattern is then, then, then, … with a single catch
- If any of the promises fail, the catch is called
- So that same code now looks like:

PROMISES

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

PROMISES

- To create a promise, you create a new Promise instance which takes as a function as the single parameter
- The function needs to take resolve and reject functions
- When you complete your action, call resolve with whatever data you want to return, this will call the next then
- If your action fails, call reject, and pass an error, this will call catch

PROMISES

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

PROMISES

- Sometimes you have multiple promises that you want to run “simultaneously” - Promise.all
- Sometimes you need to return a promise that just resolves (maybe to fulfill a contract) - Promise.resolve() or Promise.reject()

DEMO / LAB

CHAPTER No. 12

API ACCESS

FETCH

★ ★ ★ ★ ★ ★ ★ ★ ★

- JS provides us with a simple way to get basic calls
- fetch(‘http://www.google.com')
- Returns a promise

FETCH

```
fetch('http://www.google.com')
 .then((response) => {
  console.log(response.data);
 });
```

REQUEST

- Sometimes web calls aren't so simple ...
- You need may need to specify headers, cors, or execute a different verb like POST.
- fetch also accepts a Request object

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);
});
```

AXIOS

★ ★ ★ ★ ★ ★ ★ ★ ★

- Axios provides a friendly wrapper for making web calls
- Its most basic method is axios(config)
- Does provide friendly axios.get, axios.post, etc
- The config is made up of logical properties like: url, verb, headers, data
- Returns a promise

AXIOS

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

DEMO / LAB


FINISH