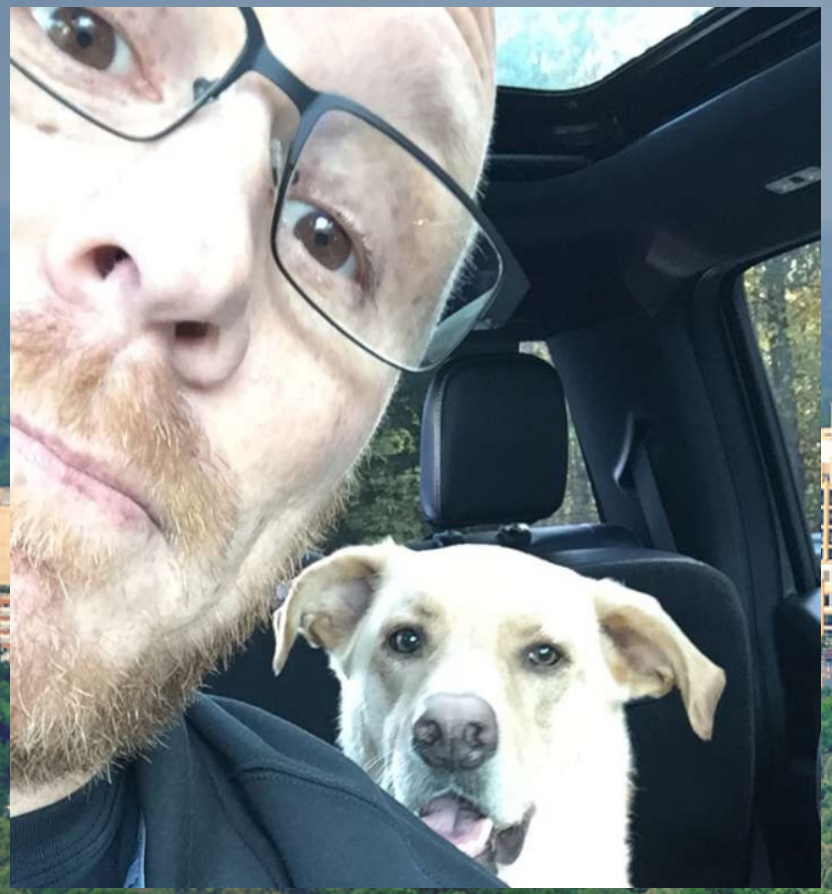
# Introduction To ReactJS

January 27, 2018

# Steve Pietrek




App Dev Practice Manager
Front End Developer
SharePoint Application Architect

@spietrek
spietrek@gmail.com
Cardinal Solutions
Raleigh/Durham


**Presentation References**
https://github.com/spietrek/triangle-reactjs-intro-to-reactjs

# Topics

| | | |
|---|---|---|
| React Overview | Virtual DOM & Components | JSX |
| Create React App | ES6 / ECMAScript 6 | Lifecycle Methods |

# React Overview

A JAVASCRIPT LIBRARY FOR BUILDING USER INTERFACES

FOCUSES ON THE “VIEW” IN MODEL VIEW CONTROLLER

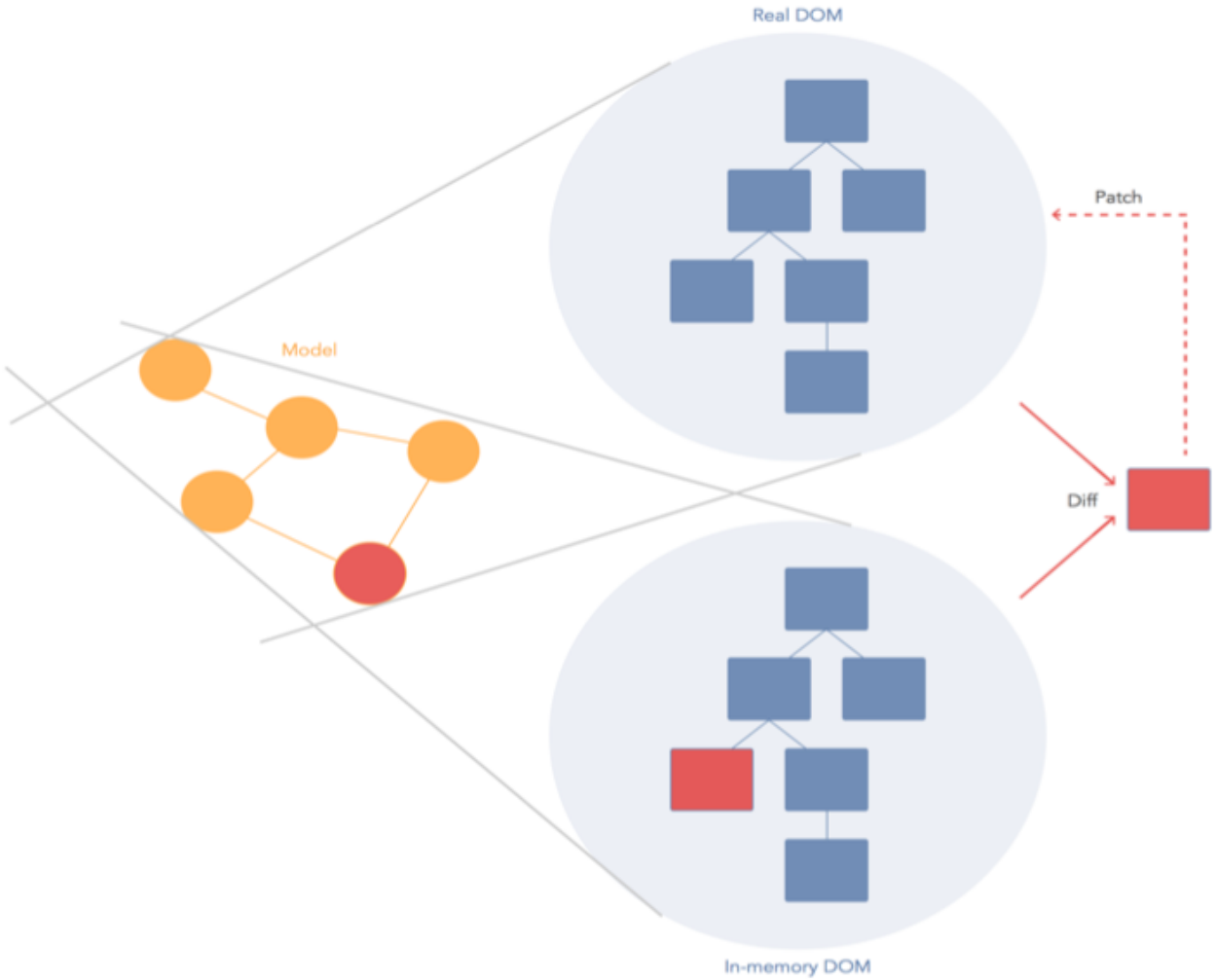
# React Key Concepts

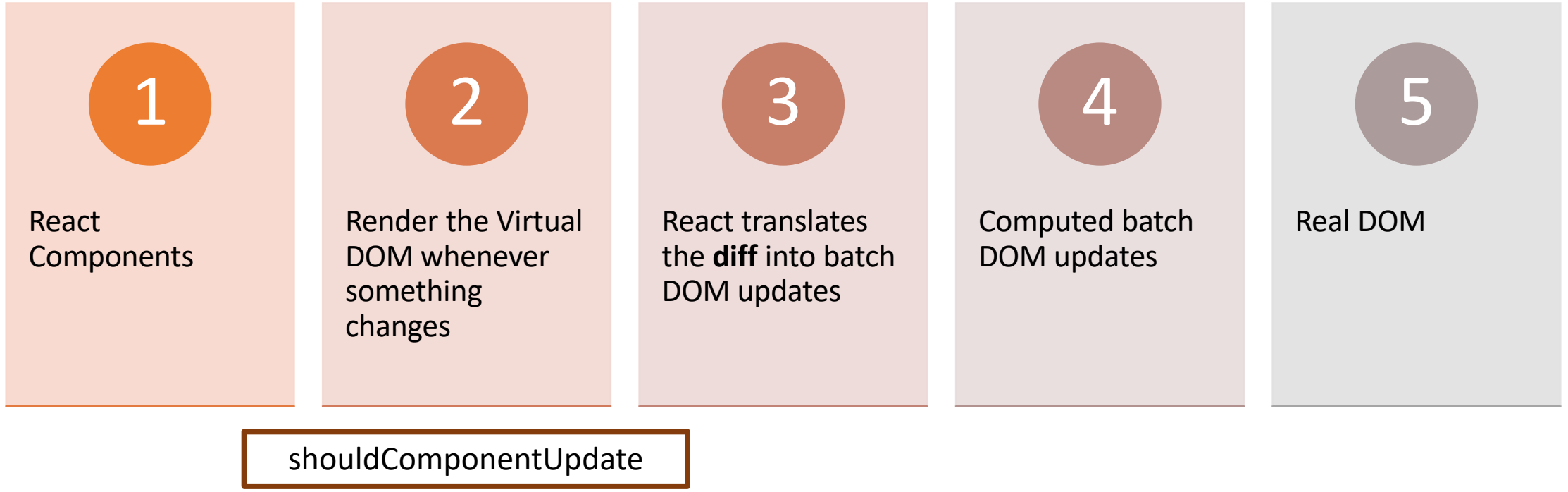
Virtual DOM

Components

# Virtual DOM

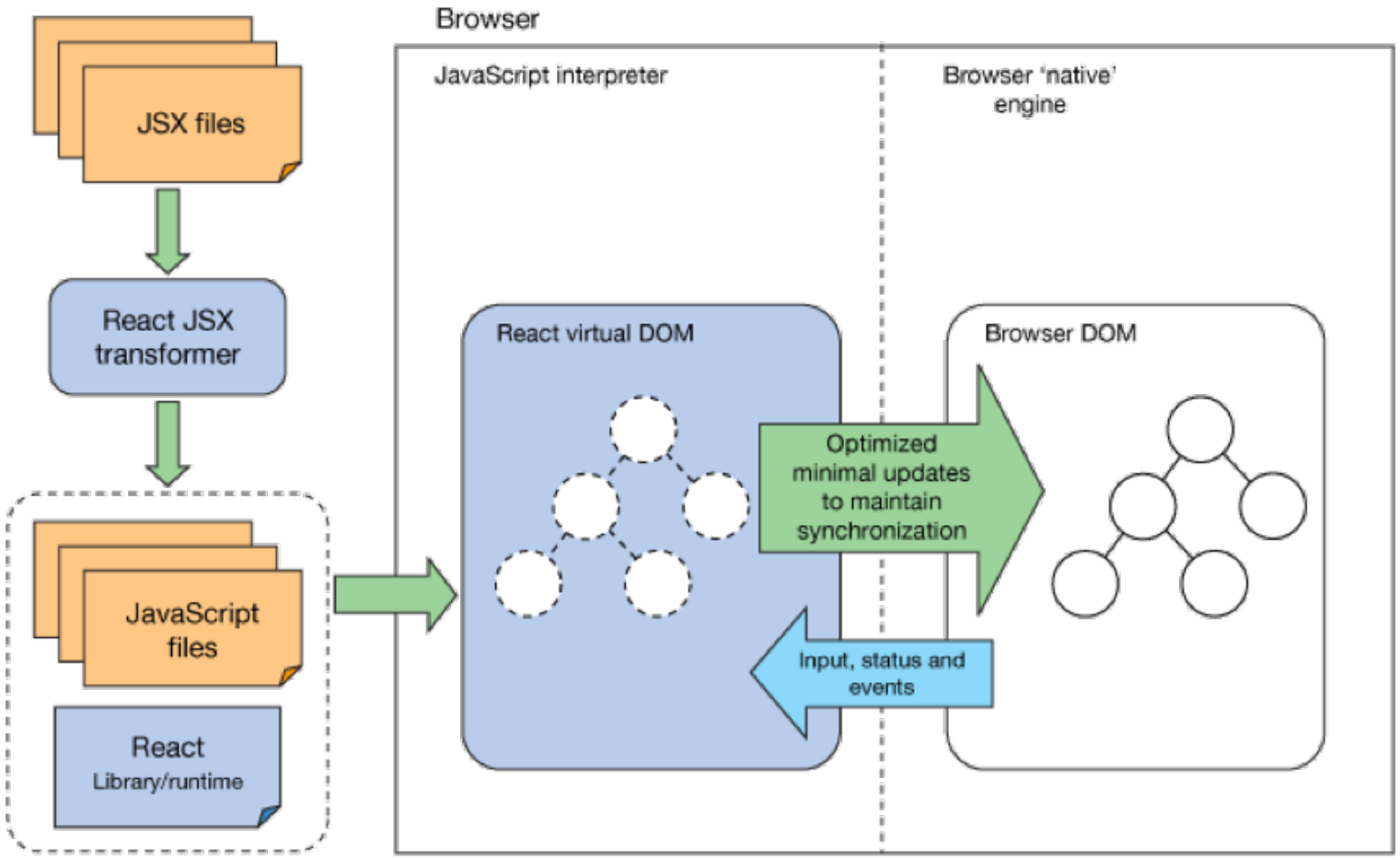
# Virtual DOM




1
React Components
2
Render the Virtual DOM whenever something changes
3
React translates the **diff** into batch DOM updates
4
Computed batch DOM updates
5
Real DOM
shouldComponentUpdate

# Virtual DOM

# Virtual DOM

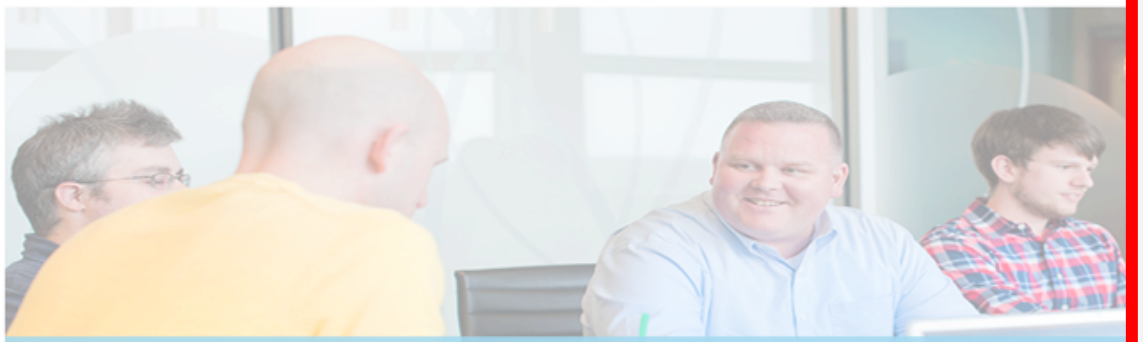

JSX files
React JSX transformer
JavaScript files
React
Library/runtime
Browser
JavaScript interpreter
Browser 'native' engine
React virtual DOM
Browser DOM
Optimized minimal updates to maintain synchronization
Input, status and events

# EVERYTHING IN REACT IS A COMPONENT

Office 365 | Sites

Drive everyday

Human Resources Dep



**Annual Employee Appreciation Celebration Planning i**

Click here to see some of the exciting events we have in store for you!

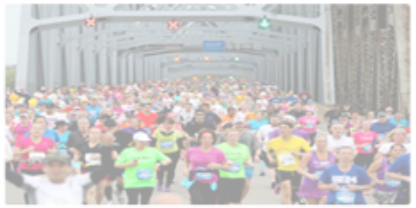
## News & Announcements


June 7, 2015

### The Benefits of Blogging

Blogging for business is not a new concept, but for many small businesses, especially service businesses, it can be an...

Read More.

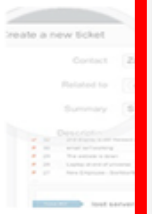


June 17, 2015

### Flying Pig 5K Marathon

It's the largest weekend party in town, and you can be part of it! Whether you're a runner or walker, whether you want to...

Read More.



June 23, 20

### Automa

We are e nounce a help des that will to your r

Read More.

## 12 Projects

Red Projects

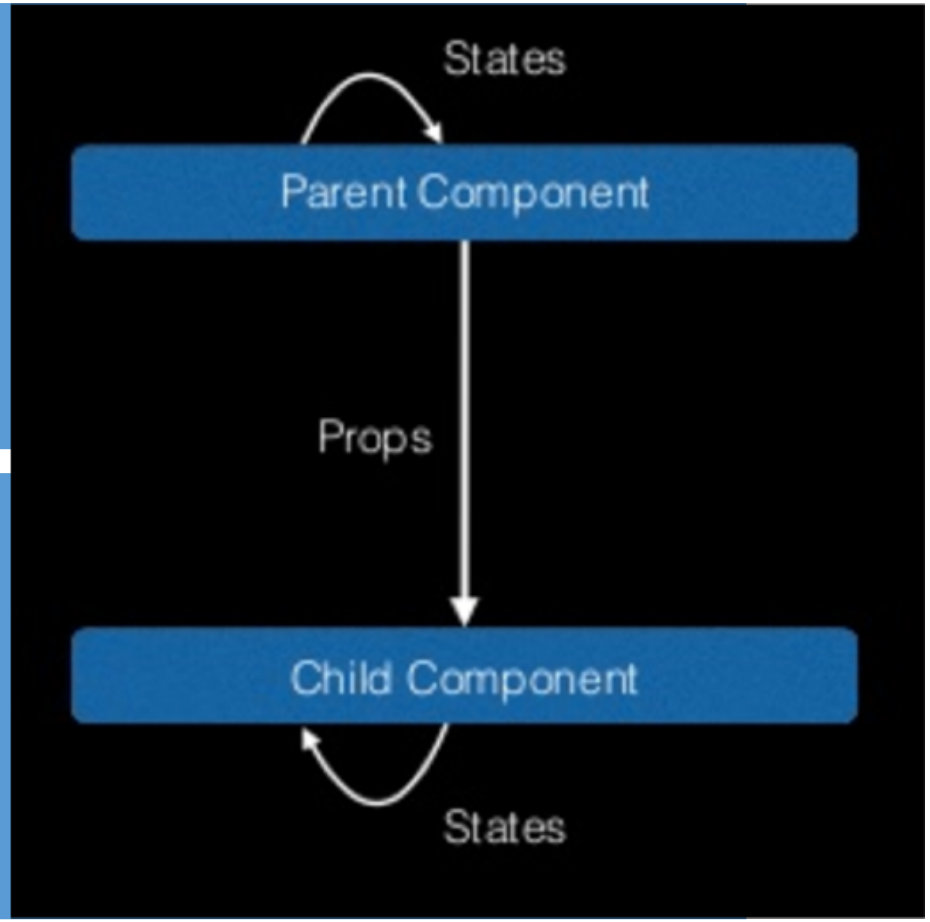
- Firetruck
- Irongate

All Projects

- Bladerunner
- Constantine
- Excalibur
- Firetruck
- Irongate
- Kryptonite
- Mad Hatter
- Moonshine
- Orion
- Revolution
- Sputnik
- Zircon

# One Way Data Flow

| | |
|---|---|
| Data flows down | Props are immutable<br>States are mutable<br>Anything can be passed to children as props |
| Events flow up | Through parent passed event handler props |



# Props vs State

| | **props** | **state** |
|---|---|---|
| Can use a plain JavaScript object to store data? | Yes | Yes |
| Can get initial value from parent component? | Yes | Yes |
| Can be changed by parent component? | Yes | No |
| Can set default values inside component? | Yes | Yes |
| Can change inside component? | No | Yes |
| Can set initial value for child components? | Yes | Yes |
| Does change trigger render event? | Yes | Yes |

# Typechecking with PropTypes

TypeScript or Flow can be used to typecheck your **whole** app

PropTypes ensure your props data is valid

A warning will display in the console if an invalid value was provided for a prop

defaultProps is used to set a default prop value when it is not required

```
class Greeting extends React.Component {
  render() {
    return (
      <h1>Hello, {this.props.name}</h1>
    );
  }
}

// Specifies the default values for props:
Greeting.defaultProps = {
  name: 'Stranger'
};

// Renders "Hello, Stranger":
ReactDOM.render(
  <Greeting />,
  document.getElementById('example')
);
```

```
import PropTypes from 'prop-types';

class Greeting extends React.Component {
  render() {
    return (
      <h1>Hello, {this.props.name}</h1>
    );
  }
}

Greeting.propTypes = {
  name: PropTypes.string
};
```

# propTypes / defaultProps Example

# Immutability

- An **immutable** object is an object whose state cannot be modified after it is created
- General rule: Treat state as if it were immutable
- You are circumventing React's state management > calling setState may replace your mutation
- Mutable array method: push
- Immutable array methods: Map, filter, concat

Example 1 (array):

```
let updatedTodos = this.state.todos.concat(newTodo);
this.setState({todos: updatedTodos});
```

Example 2 (mutate object):

```
let updateItem = { …this.state.todo, isComplete:true };
this.setState({todo: updateItem});
```

- Check out **React Immutability Helpers**

# JSX

# JSX

Syntax extension to JavaScript which expresses the Virtual DOM

Declarative description of the UI inline in JavaScript code

Use inside if statements and for loops, assign to variables, accept as arguments, return from functions

Wrap JavaScript expressions in {} to be used within an element

Preprocessor translates syntax into plain JavaScript objects (Babel)

# JSX vs. Create Element Example

```
class Hello extends React.Component {
  render () {
    return (
      <div className="container">
        <h1>Getting Started</h1>
      </div>
    );
  }
}

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

```
class Hello extends React.Component {
  render () {
    return (
      React.createElement("div",
        { className: "container"},
         React.createElement("h1", null, "Getting Started")
      )
    );
  }
}
ReactDOM.render(React.createElement(Hello, null), mountNode);
```

# Advanced JSX Example

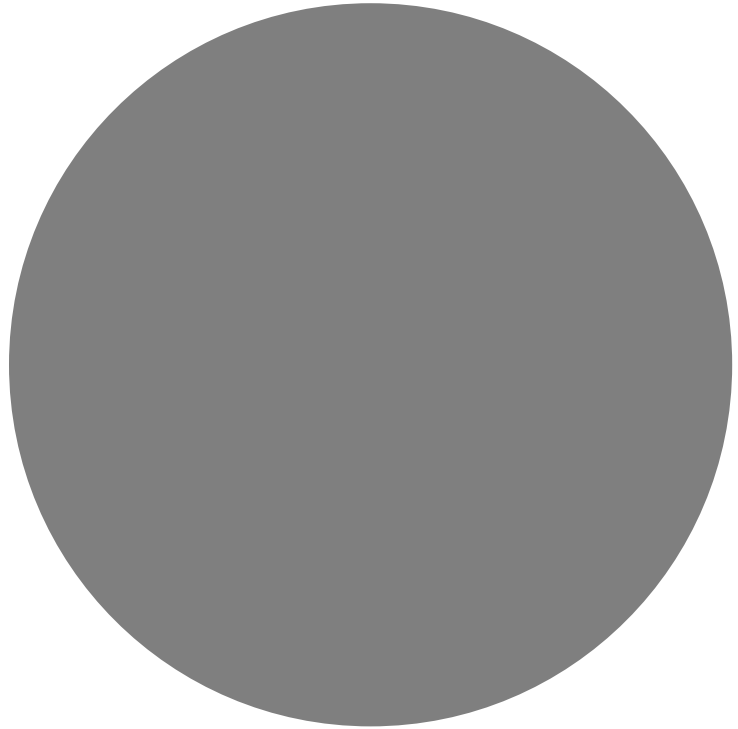
```
import React, { Fragment } from 'react';      Steve Pietrek, 6 days
import { Spinner } from '../common';
import TitlePage from './TitlePage';
import TableContentsPage from './TableContentsPage';
import PlanSections from './PlanSections';
import PlanFooter from './PlanFooter';

/* eslint-disable react/prop-types */
const TableResult = props => {
  const { filteredTableContents } = props;
  if (filteredTableContents.length > 0) {
    return (
      <Fragment>
        <TitlePage {...props} />
        <TableContentsPage {...props} />
        <PlanSections {...props} />
        <PlanFooter {...props} />
      </Fragment>
    );
  }
  return null;
};

const withSpinner = Comp => ({ isLoading, children, ...props }) =>
  isLoading ? Spinner() : <Comp {...props}>{children}</Comp>;
/* eslint-enable react/prop-types */

const DocumentResultWithSpinner = withSpinner(TableResult);

const PlanDocument = props => (
  <Fragment>
    <DocumentResultWithSpinner {...props} />
  </Fragment>
);

export default PlanDocument;
```

# Create React App

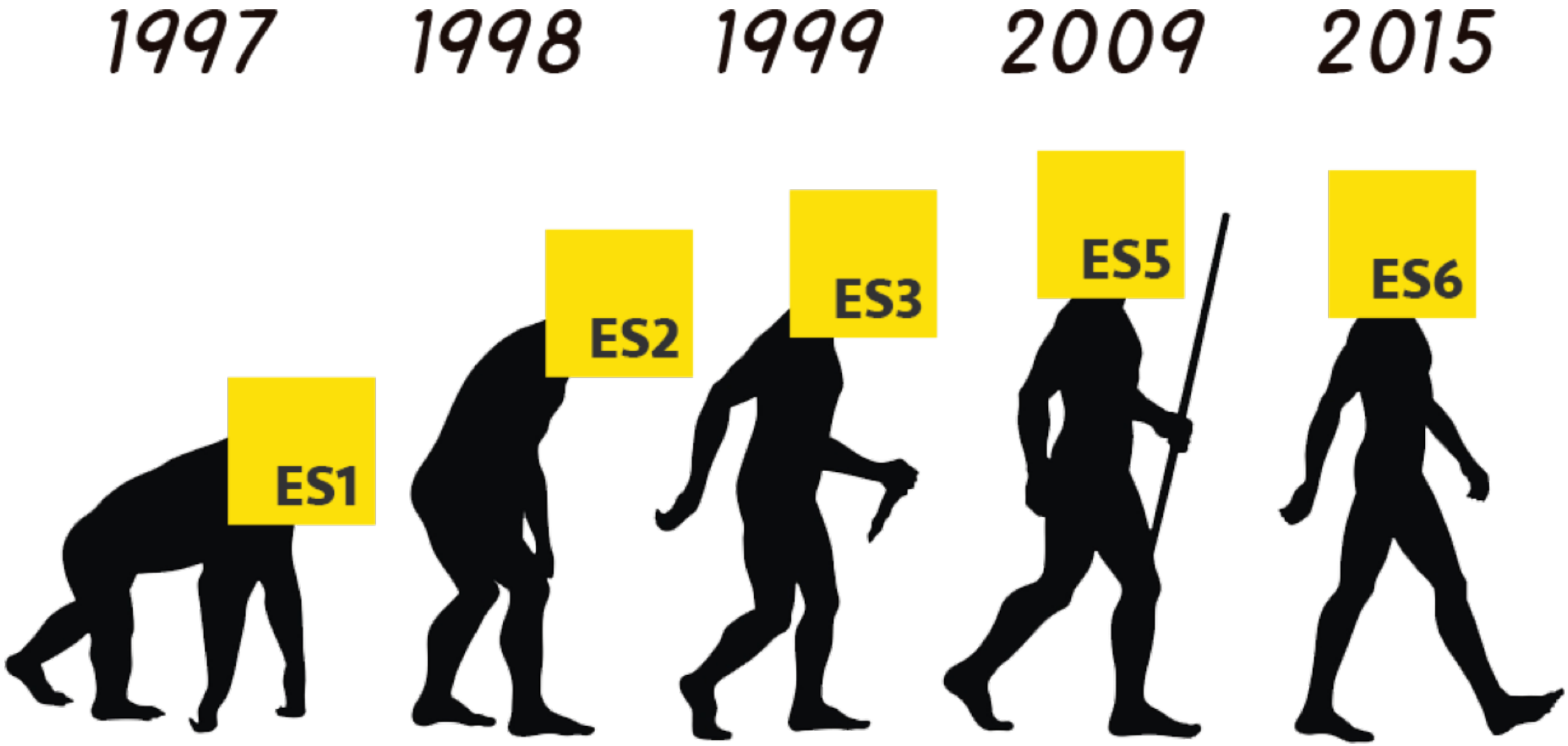
# Create React App

- Create React apps with no build configuration
- Every project built with it has a familiar base, reducing the ramp-up time as you and your teammates switch projects
- Quick start (in Terminal/Console):
  - npm install –g create-react-app
  - create-react-app my-app
  - cd my-app
  - npm start or yarn start
- Uses ES6, Webpack, Babel, and ESLint
- Can be upgraded in-place to get the latest config and tooling
- You can “eject” to a custom setup at any time. No going back.

# React Demo #1



# ES6 / ECMAScript 6


1997
1998
1999
2009
2015
ES1
ES2
ES3
ES5
ES6

# Block-Scoped Constructs Let and Const

- **var** is hoisted and function-scoped
- **let** is not hoisted and limited in scope to the block, statement, or expression where it is used
- **const** is immutable (read-only reference to a value) and limited in scope to the block, statement, or expression where it is used
- Recommendation: always use **const** until you need to mutate

| Construct | Scope | Reassignable | Mutable |
|---|---|---|---|
| const | Block | No | Yes |
| let | Block | Yes | Yes |
| var | Function | Yes | Yes |

# Template Literals

- Avoid using concatenation
- Use backtick

- ES5

```
var name = "My name is " + firstName + " " + lastName
```

- ES6

```
const name = `My name is ${firstName} ${lastName}`
```

# Arrow Functions

- Traditional function with a **.bind(this)**
- Biggest benefit > **this** will have the same value as the context of the function
- No more **that = this** or **self = this**

- ES5

```
var names = ['Steve', 'Laura'];
var messages = names.map(function (name, index) {
  return 'Name of ' + index + ' element is ' + name + ' '
})
```

- ES6

```
const names = ['Steve', 'Laura'];
const messages = names.map((name, index) => `Name of ${index} element is ${name} `)
```

# Rest Parameters/Spread Operator

- Allows an iterable to be expanded
- Use “real” array and can use array methods

- ES5

```
function max() {
  return Math.max.apply(null, arguments);
}
max(5,8,1,18);
```

- ES6 (spread)

```
function max() { return Math.max(...arguments); }
```

- ES6 (rest and spread)

```
const max = (...nums) => Math.max(...nums);
```

# Destructuring Assignment

- Breaks down objects and arrays into individual variables

Example 1:

```
const parts = ['shoulders', 'knees'];
const lyrics = ['head', ...parts, 'and', 'toes'];
console.log(lyrics); // ['head', 'shoulders', 'knees', 'and', 'toes']
const [first] = lyrics;
console.log(first); // head
```

Example 2:

```
const TodoListClearButton = ({ isFetching, todos, onClear }) => (
    isFetching || !todos.length ? null : <button onClick={onClear}>Clear Todos</button>
);
```

# Babel

- Babel is a JavaScript compiler.
- Use next generation JavaScript, today.
- Babel “transpiles” to ES5 JavaScript.
- Many browsers do not support all of ES6 features.

# More on Components

# Container vs. Presentational Components

- Note: React components do not need to emit DOM
- Container component does data fetching/business logic and renders its corresponding (shares the same name) sub-component
- Benefits:
  - Allow for separation of concerns
  - Allow for reusability of presentational components
  - Allow for better collaboration with designers and other developers

# Container vs. Presentational Components

| Container Component | Presentational Component |
| --- | --- |
| Concerned with how things work | Concerned with how things look |
| Provide data and behavior to presentational or other container components | Have no dependencies on the rest of the app (e.g. Redux, stores) |
| Interact with Redux or other state management libraries | Receive data and callbacks only through props |
| Generally stateful | Only state is UI state (not App state) |
| Generated using higher order components such as connect() (Redux) | Are generally written as functional components |

# Anti-Pattern

```
// CommentList.js
import React from "react";

class CommentList extends React.Component {
  constructor() {
    super();
    this.state = { comments: [] }
  }

  componentDidMount() {
    fetch("/my-comments.json")
      .then(res => res.json())
      .then(comments => this.setState({ comments }))
  }

  render() {
    return (
      <ul>
        {this.state.comments.map(({ body, author }) =>
          <li>{body}-{author}</li>
        )}
      </ul>
    );
  }
}
```

# Best Practice

## Container Component

```
// CommentListContainer.js
import React from "react";
import CommentList from "./CommentList";

class CommentListContainer extends React.Component {
  constructor() {
    super();
    this.state = { comments: [] }
  }

  componentDidMount() {
    fetch("/my-comments.json")
      .then(res => res.json())
      .then(comments => this.setState({ comments }))
  }

  render() {
    return <CommentList comments={this.state.comments} />;
  }
}
```

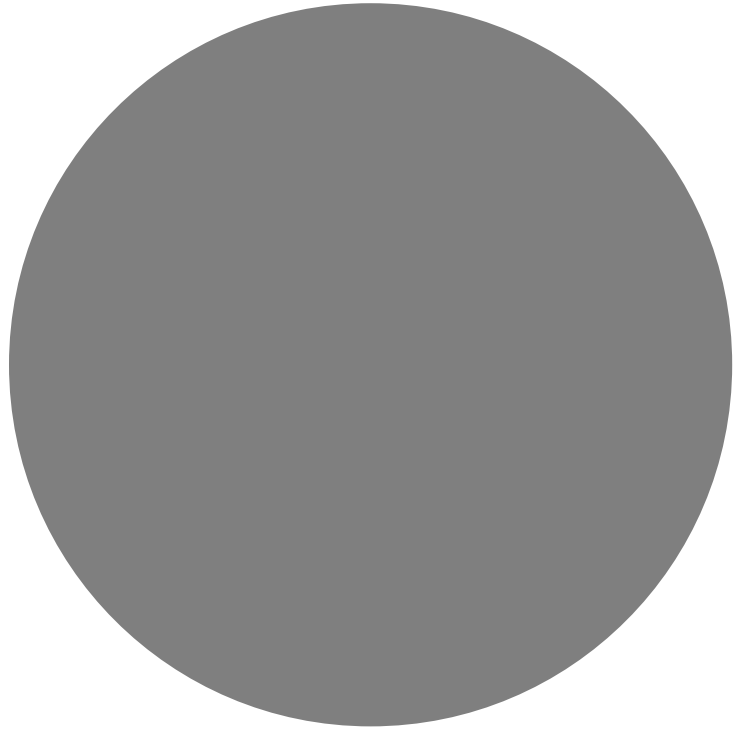
## Presentational Component

```
// CommentList.js
import React from "react";

const Commentlist = comments => (
  <ul>
    {comments.map(({ body, author }) =>
      <li>{body}-{author}</li>
    )}
  </ul>
)
```

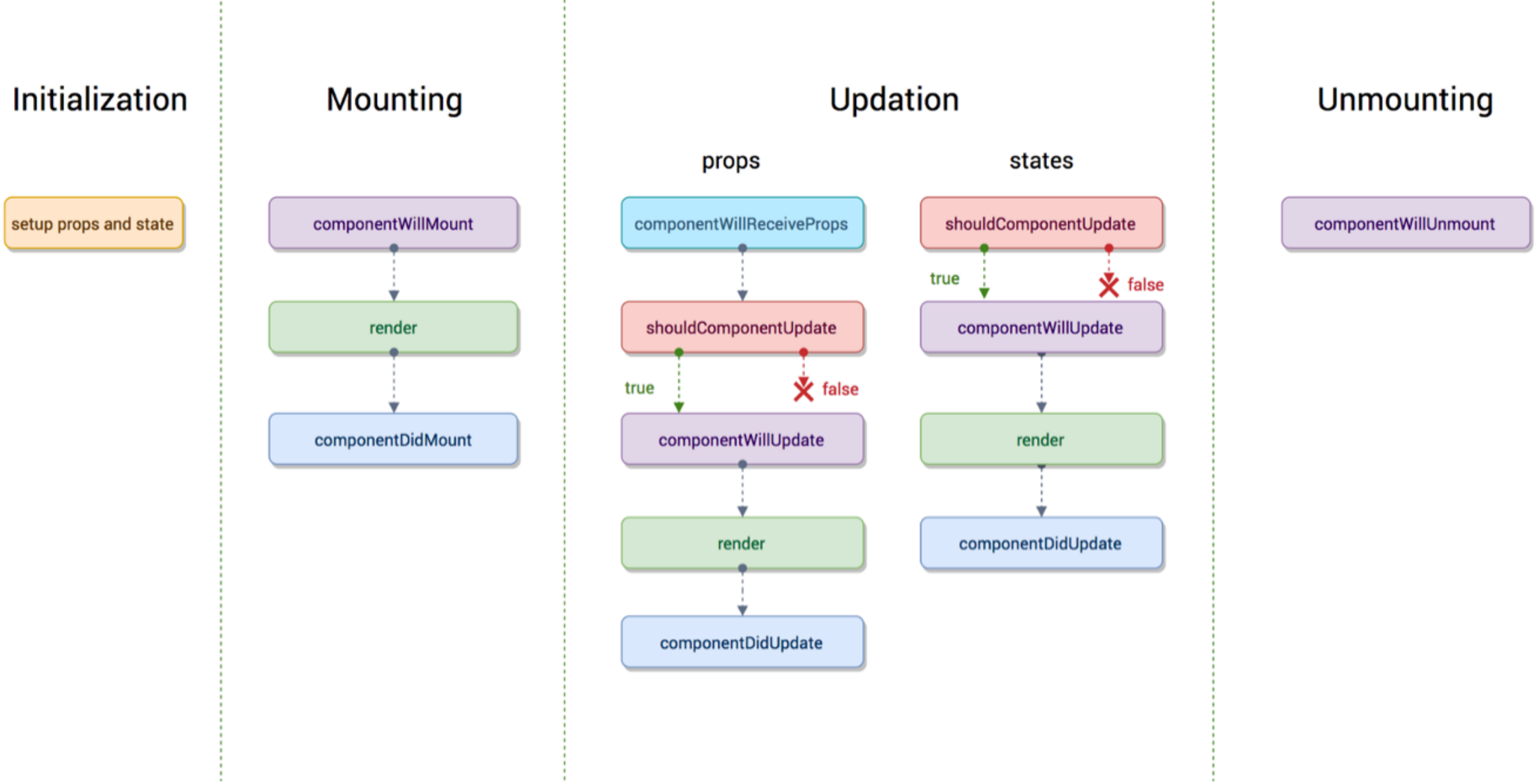
# Stateless vs. Component vs. Pure Component

| | |
|---|---|
| Stateless component | Created using a function<br>No state<br>No lifecycle methods<br>Presentational components |
| Normal component | Allows state<br>Lifecycle methods<br>Generally Container components |
| Pure component | Similar to Normal components<br>Implements shouldComponentUpdate automatically<br>Shallow compare of new props and old props<br>Allows state<br>Lifecycle methods<br>Be careful when using due to shallow compare |

# React Demo #2



# Lifecycle Methods




componentDidCatch(error, info)

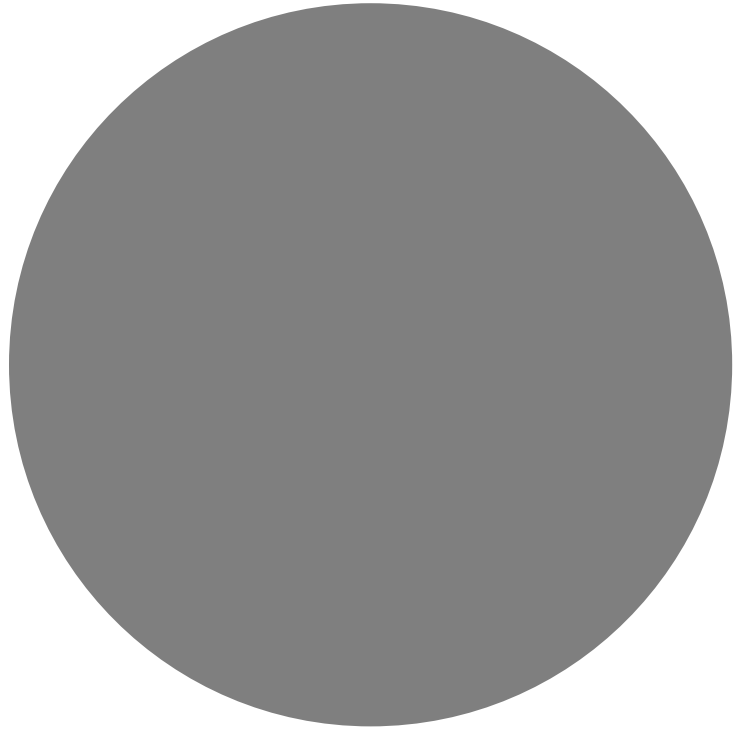
# React Lifecycle Best Practices

- The constructor() and componentWillMount() methods can usually be avoided by using class properties instead.
- Do NOT do async things or add listeners inside of componentWillMount(). Use componentDidMount() instead.
- render() gets gets called a ton. Try to keep it pretty lightweight. (avoid sorting large lists, etc).
- Always cancel any listeners, timeouts, intervals, fetch requests, etc in componentWillUnmount().

| method | phase | setState? | SSR* |
|---|---|---|---|
| constructor | mounting | ✗ | ✓ |
| componentWillMount | mounting | ✓ | ✓ |
| render | mounting, updating | ✗ | ✓ |
| componentDidMount | mounting | ✓ | ✗ |
| componentWillReceiveProps | updating | ✓ | ✗ |
| shouldComponentUpdate | updating | ✗ | ✗ |
| componentWillUpdate | updating | ✗ | ✗ |
| componentDidUpdate | updating | ✓ | ✗ |
| componentWillUnmount | unmounting | ✗ | ✗ |
| componentDidCatch | errors | ✓ | ✓ |

**SSR: Server Side Rendering*

# React Demo #3



# Tools

# Node.js & NPM

- Node.js
  - Open source, cross-platform runtime environment for server-side and networking applications
  - Applications are written in JavaScript
  - Based on the Google's V8 JavaScript Engine
  - Uses an event-driven, single threaded, and non-blocking I/O model => lightweight and efficient
  - Many development tools leverage Node.js
- Npm
  - Node Package Manager
  - Online repository for the publishing of open-source Node.js projects
  - Packages can be installed globally (across all projects) or locally (specific to project)
  - Command-line utility for interacting with repositories (package installation, version management, and dependency management)
  - Yarn is gaining traction as a package manager

# ESLint

| | | | |
|---|---|---|---|
| Help with code quality | Catch common errors (console.log, debugger) | Provide consistency between developers | Configure rules to validate code (Formatting Rules and Code-quality rules) |
| Rules can be used to ignore, warn, or error | .eslintrc is used to define rules | Most IDE's have ESLint plugins | Supports ES6 and JSX |

# Prettier

"Opinionated code formatter"

Focuses on formatting rules

Supports JavaScript (ES6, ES7), JSX, Flow, TypeScript, CSS, JSON

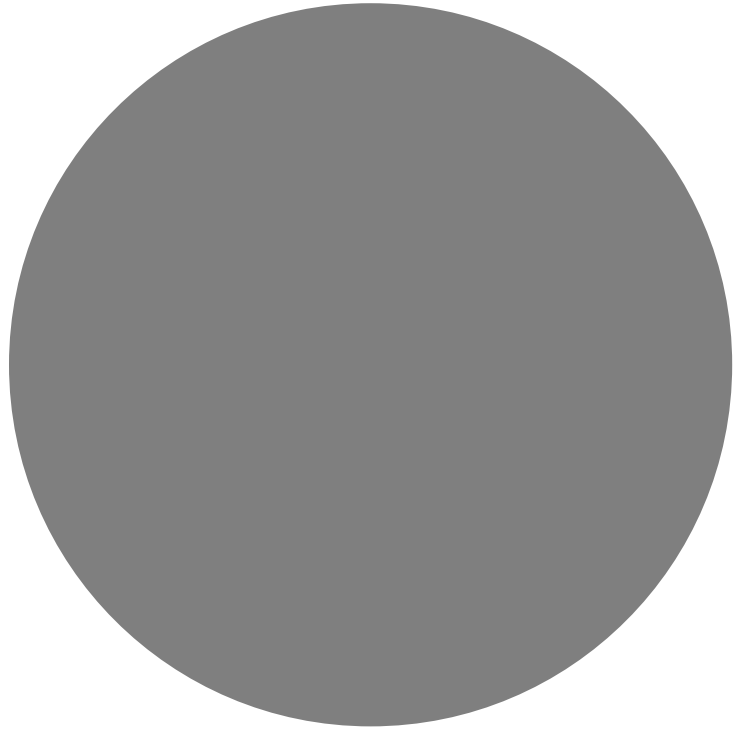
Enforces a consistent code style

Provide consistency between developers

ESLint => many configuration rules, Prettier => few configuration rules

Can use ESLint and Prettier together (eslint-plugin-prettier)

# React Demo #4



# Static Type Checking

# Static Type Checking

Static usually means “at compile time” or “without running a program”

Dynamic means “at runtime”

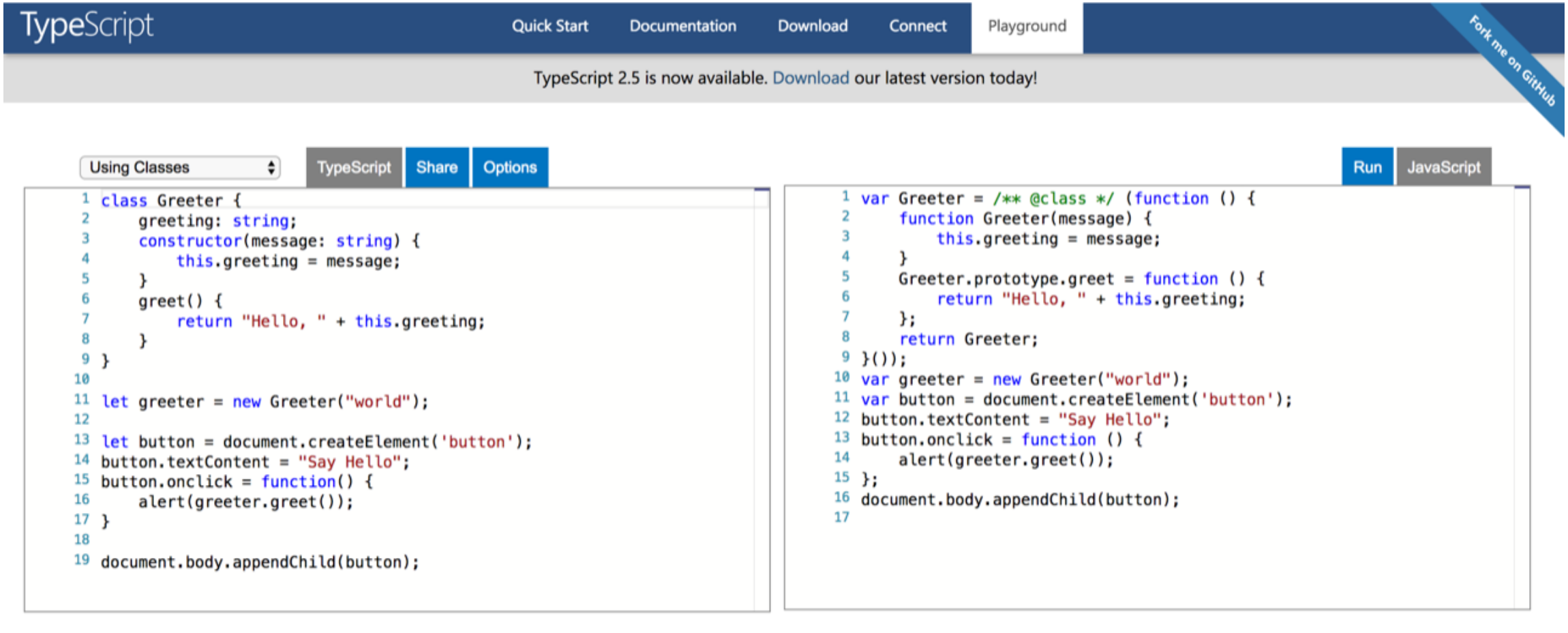
JavaScript is dynamic

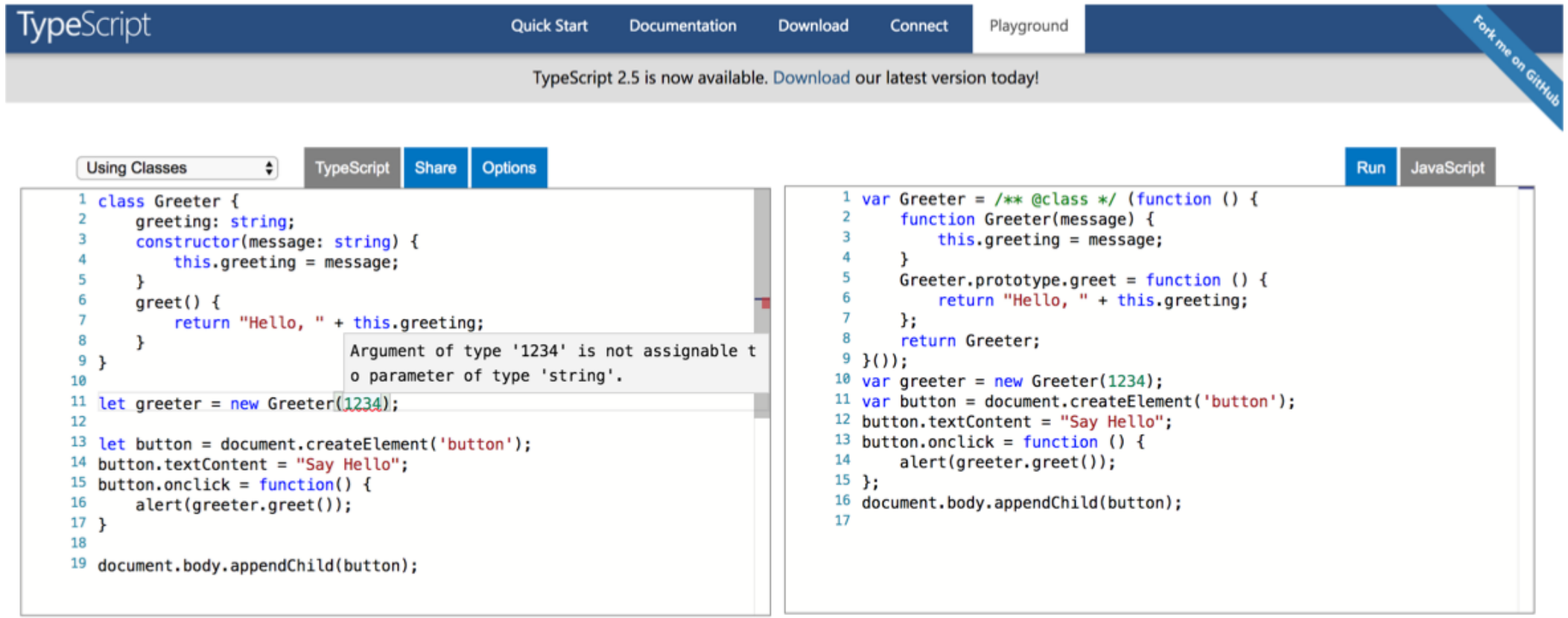
# Benefits of Static Type Checking

- You get more errors at “compile” time
- Usually faster than running unit tests and tends to catch a different category of errors
- It helps IDEs with auto-completion
- Type annotations are useful for documenting parts of an API
- Checking and documenting types helps large teams collaborate
- It gives you an additional way of specifying what you require from or provide for your collaborators

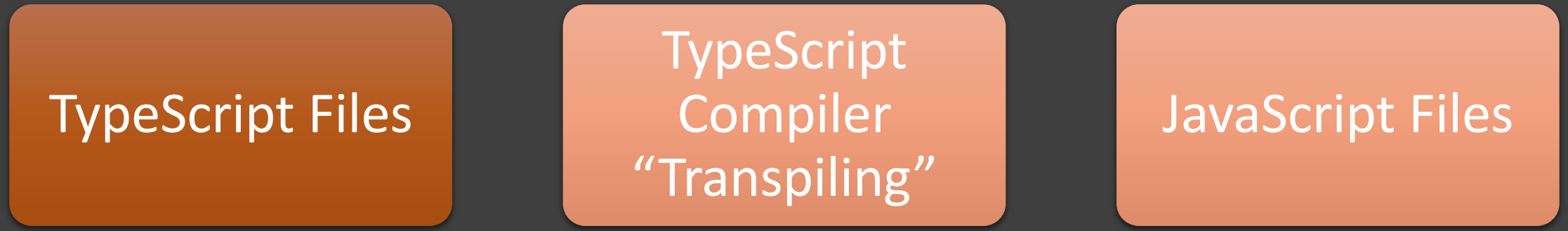


# Static Type Checkers


TypeScript
Quick Start
Documentation
Download
Connect
Playground
Fork me on GitHub
TypeScript 2.5 is now available. Download our latest version today!
Using Classes
TypeScript
Share
Options
Run
JavaScript
class Greeter {
greeting: string;
constructor(message: string) {
this.greeting = message;
}
greet() {
return "Hello, " + this.greeting;
}
}
let greeter = new Greeter("world");
let button = document.createElement('button');
button.textContent = "Say Hello";
button.onclick = function() {
alert(greeter.greet());
}
document.body.appendChild(button);
var Greeter = /** @class */ (function () {
function Greeter(message) {
this.greeting = message;
}
Greeter.prototype.greet = function () {
return "Hello, " + this.greeting;
};
return Greeter;
}());
var greeter = new Greeter("world");
var button = document.createElement('button');
button.textContent = "Say Hello";
button.onclick = function () {
alert(greeter.greet());
};
document.body.appendChild(button);


TypeScript
Quick Start
Documentation
Download
Connect
Playground
Fork me on GitHub
TypeScript 2.5 is now available. Download our latest version today!
Using Classes
TypeScript
Share
Options
Run
JavaScript
class Greeter {
greeting: string;
constructor(message: string) {
this.greeting = message;
}
greet() {
return "Hello, " + this.greeting;
}
}
Argument of type '1234' is not assignable t
o parameter of type 'string'.
let greeter = new Greeter(1234);
let button = document.createElement('button');
button.textContent = "Say Hello";
button.onclick = function() {
alert(greeter.greet());
}
document.body.appendChild(button);
var Greeter = /** @class */ (function () {
function Greeter(message) {
this.greeting = message;
}
Greeter.prototype.greet = function () {
return "Hello, " + this.greeting;
};
return Greeter;
}());
var greeter = new Greeter(1234);
var button = document.createElement('button');
button.textContent = "Say Hello";
button.onclick = function () {
alert(greeter.greet());
};
document.body.appendChild(button);

# How does TypeScript work?



# Questions

# Resources

- React - https://facebook.github.io/react/
- Create React App - https://github.com/facebookincubator/create-react-app
- axios - https://github.com/mzabriskie/axios