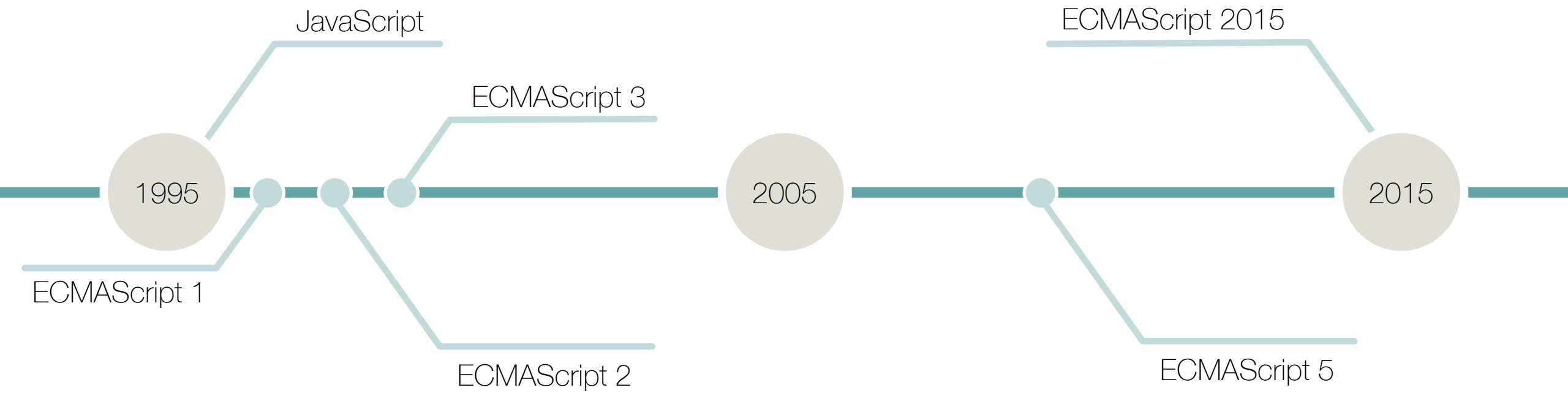
Catching up with JavaScript

– ES 2015 edition

History of JavaScript



Problem 1

- How do we organize and structure our code?

Classes

ES 2015

```
class Volkswagen extends Car {

  constructor(model, isTestRunning) {
    super(model);
    this.isTestRunning = isTestRunning;
  }

  getEmission() {
    var emission = super.getEmission();
    return this.isTestRunning ?
      emission / 2 : emission;
  }

  static getOrigin() { return "Germany"; }
}
```

ES5

```
'use strict';var _createClass = (function () { function defineProperties(target, props) { for
(var i = 0; i < props.length; i++) { var descriptor = props[i]; descriptor.enumerable =
descriptor.enumerable || false; descriptor.configurable = true; if ("value" in descriptor)
descriptor.writable = true; Object.defineProperty(target, descriptor.key, descriptor); } }
return function (Constructor, protoProps, staticProps) { if (protoProps)
defineProperties(Constructor.prototype, protoProps); if (staticProps)
defineProperties(Constructor, staticProps); return Constructor; }; })();var _get = function
get(object, property, receiver) { if (object === null) object = Function.prototype; var desc =
Object.getOwnPropertyDescriptor(object, property); if (desc === undefined) { var parent =
Object.getPrototypeOf(object); if (parent === null) { return undefined; } else { return
get(parent, property, receiver); } } else if ("value" in desc) { return desc.value; } else { var
getter = desc.get; if (getter === undefined) { return undefined; } return getter.call(receiver);
} };function _classCallCheck(instance, Constructor) { if (!(instance instanceof Constructor)) {
throw new TypeError("Cannot call a class as a function"); } }function
_possibleConstructorReturn(self, call) { if (!self) { throw new ReferenceError("this hasn't been
initialised - super() hasn't been called"); } return call && (typeof call === "object" || typeof
call === "function") ? call : self; }function _inherits(subClass, superClass) { if (typeof
superClass !== "function" && superClass !== null) { throw new TypeError("Super expression must
either be null or a function, not " + typeof superClass); } subClass.prototype =
Object.create(superClass && superClass.prototype, { constructor: { value: subClass, enumerable:
false, writable: true, configurable: true } }); if (superClass) Object.setPrototypeOf ?
Object.setPrototypeOf(subClass, superClass) : subClass.__proto__ = superClass; }var Volkswagen =
(function (_Car) {  _inherits(Volkswagen, _Car);  function Volkswagen(year, model) {
_classCallCheck(this, Volkswagen);    var _this = _possibleConstructorReturn(this,
Object.getPrototypeOf(Volkswagen).call(this, year, model));    _this.origin = 'Germany';
return _this;  }  _createClass(Volkswagen, [{    key: 'getEmission',    value: function
getEmission() {      var emission = _get(Object.getPrototypeOf(Volkswagen.prototype),
'getEmission', this).call(this);      return this.isTestRunning ? emission / 2 : emission;    }
}]);  return Volkswagen;})(Car);
```

Babel

- A transpiler for JavaScript
  - ES 2015, ES 2016, JSX…
  - You choose which transforms to apply using plugins
  - Presets for common uses
- Other tools exists
  - But not really…

.babelrc

```
{
  "presets": ["es2015"]
}
```

```
> npm install babel-cli
> npm install babel-preset-es2015
> ./node_modules/.bin/babel script.js
```

Modules

- In JavaScript the default way is to include and structure code manually using `<script>`

- Other solutions are available

  - CommonJS (Node.js)
  - Asynchronous Module Definition (RequireJS)
  - Universal Module Definition

Modules

lib/module1.js

```
export function hello() {
  console.log("Hello world");
};

export var helloPhrase = "Hello world";

export class Volkswagen { … }
```

app/main.js

```
import * as mod1 from "../lib/module1";

mod1.hello();
mod1.hello() === say.helloPhrase

var car = new mod1.Volkswagen()

import { hello, helloPhrase, Volkswagen }
  from "../lib/module1";

hello();
hello() === helloString;

var car = new Volksvagen();
```

Modules

lib/module1.js

```
function hello() {
  console.log("Hello world");
};

var helloPhrase = "Hello world";

class Volkswagen { … }

export {
  hello,
  helloPhrase,
  Volkswagen
};
```

app/main.js

```
import * as mod1 from "../lib/module1";

mod1.hello();
mod1.hello() === say.helloPhrase

var car = new mod1.Volkswagen()

import { hello, helloPhrase, Volkswagen }
  from "../lib/module1";

hello();
hello() === helloString;

var car = new Volksvagen();
```

Modules

lib/module1.js

```
export default class Volkswagen { … }

export function hello() {
  console.log("Hello world");
};

export var helloPhrase = "Hello world";
```

app/main.js

```
import Volkswagen from "../lib/module1";
var car = new Volksvagen();

import { hello, helloPhrase }
  from "../lib/module1";

hello();
hello() === helloString;
```

Webpack

- Bundles modules and its dependencies as static assets
  - JavaScript
  - CSS, HTML, Images, …
- Complex but worth it
  - Server, Hot Reloading, Sourcemaps, …

Webpack.config.js

```
module.exports = {
  entry: "./app/main.js",
  output: {
    path: "./dist",
    filename: "app.js"
  },
  module: {
    loaders: [{
      test: /\.js$/,
      loader: "babel"
    }]
```

```
> npm install webpack
> ./node_modules/.bin/webpack
```

Problem 2

- Why is JavaScript such a crappy language?

Managing scope

challenge1.js

```
var txt = ["a", "b", "c"];

for (var i = 0; i < 3; i++) {

  var msg = txt[i];

  setTimeout(function () {
    console.log(msg);
  }, 1000);
}
```

```
> node challenge1.js
c c c
```

Actual

```
var msg;
var i;
var txt = ["a", "b", "c"];

for (i = 0; i < 3; i++) {

  msg = txt[i];

  setTimeout(function () {
    console.log(msg);
  }, 1000);
}
```

Managing scope

challenge2.js

```
var testVar = "I'm a global";

function challenge2 () {
  alert(testVar);
  var testVar = "I'm a local var";
  alert(testVar);
}

challenge2();
```

```
> node challenge2.js
undefined
I'm a local var
```

Actual

```
var testVar = "I'm a global";

function challenge2 () {

  var testVar;

  alert(testVar);
  testVar = "I'm a local var";
  alert(testVar);
}

challenge2();
```

Managing scope

challenge2.js

```
let testVar = "I'm a global";

function challenge2 () {
  alert(testVar);
  let testVar = "I'm a local var";
  alert(testVar);
}

challenge2();
```

```
> node challenge2.js
ReferenceError: testVar is not defined
```

Managing scope

challenge1.js

```
let txt = ["a", "b", "c"];

for (let i = 0; i < 3; i++) {

  let msg = txt[i];

  setTimeout(function () {
    console.log(msg);
  }, 1000);
}
```

```
> node challenge1.js
a b c
```

Managing scope

- Use `let` instead of `var`
- Use `const` if you can
  - The reference is contant
  - The content can still change
  - Use deepFreeze to freeze content

```
const name = "kokitotsos";

name = "kits";
TypeError: Assignment to constant  variable.

const name = {
  name: "kokitotsos"
};

name.name = "kits";
```

ESLint

- Should be present in all JavaScript projects
  - Used to find errors in your code
  - Lots of pluggable rules
  - Special rules for frameworks
- Has replaced JSHint

.eslintrc

```
{
  "parser": "babel-eslint",
  "parserOptions": {
    "ecmaVersion": 6,
    "sourceType": "module"
  },
```

```
> npm install eslint
> npm install babel-eslint

> ./node_modules/.bin/eslint main.js

1:1  error  Unexpected var, use let or const
instead  no-var
```

What is this?

challenge3.js

```
class Volkswagen {

  constructor(model) {
    this.model = model;
  }

  printModel() {
    console.log(this.model);
  }
}

(new Volkswagen("Golf")).printModel();
```

```
> node challenge3.js
Golf
```

What is this?

challenge3.js

```
class Volkswagen {

  constructor(model) {
    this.model = model;
  }

  printModel() {
    setTimeout(function () {
      console.log(this.model);
    }, 0);
  }
}

(new Volkswagen("Golf")).printModel();
```

```
> node challenge3.js
undefined
```

Arrow functions

- Shorthand syntax for functions
- Share the same lexical `this` as surrounding code

```
const hello = () => {
  console.log("hi");
}

const add = (a, b) => {
  return a + b;
}

const square = x => x * x;
```

Arrow functions

challenge3.js

```
class Volkswagen {

  constructor(model) {
    this.model = model;
  }

  printModel() {
    setTimeout(function () {
      console.log(this.model);
    }, 0);
  }
}

(new Volkswagen("Golf")).printModel();
```

```
> node challenge3.js
```

Arrow functions

challenge3.js

```
class Volkswagen {

  constructor(model) {
    this.model = model;
  }

  printModel() {
    setTimeout(() => {
      console.log(this.model);
    }, 0);
  }
}

(new Volkswagen("Golf")).printModel();
```

```
> node challenge3.js
Golf
```

Destructuring

```
const props = {
  username: "kokitotsos",
  show: true
};
```

ES 5

```
var username = props.username;
var show = props.show;
```

ES 2015

```
const { username, show } = props;
```

Destructuring

ES 2015

```
const obj = {
  a: {
    b: 1,
    c: 2,
    d: [3, 4, 5]
  }
};
```

```
const {
  a: {
    b,

    d: [, f, g]
  }
} = obj;
```

More features

- Template litterals

```
const a = `string with expressions ${1+2}`;
```

More features

- Template litterals
- Default parameters

```
const sayHiTo = (who = "world") => {
  console.log(`Hello ${who}`)
}
```

More features

- Template litterals
- Default parameters
- Spread operator

```
const a = [1, 2, 3];
const b = [...a, 4, 5] -> [1, 2, 3, 4, 5]
```

More features

- Template litterals
- Default parameters
- Spread operator
- Rest operator

```
const d = (e, f, ...args) => {
  console.log([e, f, ...args]);
}

d(1, 2, 3, 4, 5); -> [1, 2, 3, 4, 5]
```

More features

- Template litterals
- Default parameters
- Spread operator
- Rest operator
- Promises

```
const timeout = duration =>
  new Promise((resolve, reject) =>
    setTimeout(resolve, duration));

timeout(1000)
  .then(() => timeout(2000))
  .then(() => throw new Error("hmm"))
  .then(() => timeout(3000))
  .catch(err =>
    Promise.all([
      timeout(100),
      timeout(200)
    ]);
```

More features

- Template litterals
- Default parameters
- Spread operator
- Rest operator
- Promises
- Generators

```
function* generator(i) {
  yield i;
  yield i + 10;
}

const gen = generator(10);

console.log(gen.next().value); -> 10
console.log(gen.next().value); -> 20
```

ECMAScript 2016 & 2017

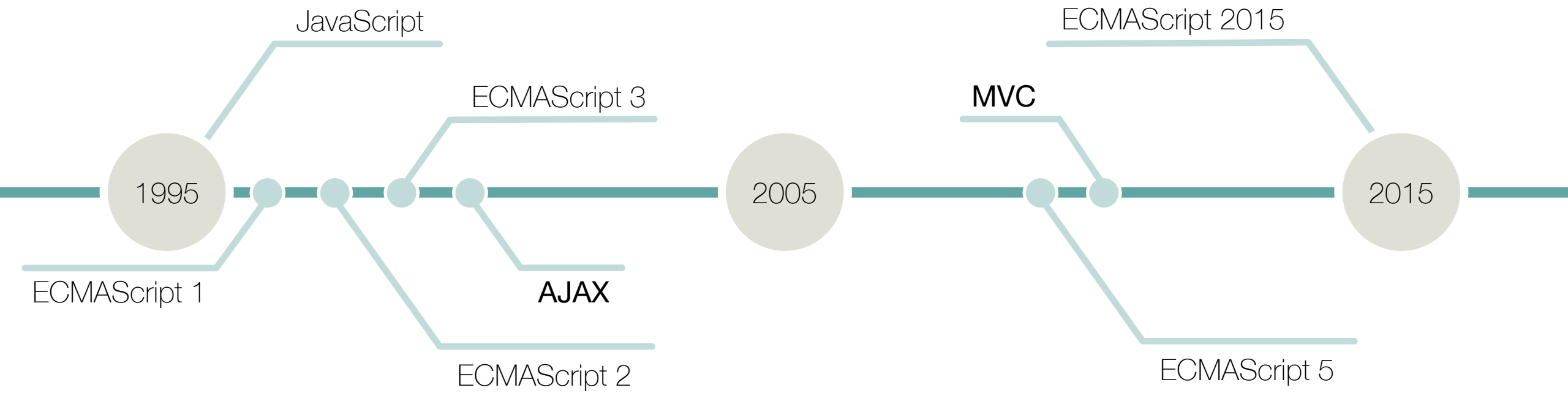
- Aiming for one new release every year
- A process for features

- All stages can be enabled in Babel
- Decorators and async functions are interesting

Architecture

- How do we build our applications?

History of JavaScript



Problem with MVC

- The model is too easy to modify
  - Two-way binding demos really well
  - The model is not always clear
  - The model can be changed from multiple places
- The controllers and views tend to grow
  - Really hard to maintain
  - Really hard to test

Components

- All frameworks move towards components
- A component should…

  …be small
  …do one isolated task
  …not have side effects if possible

Components

frameworklist.js

```
const FrameworkList =
  ({ frameworks = [] }) => (

  <ul>
    { frameworks.map(framework => {
        return <li>{ framework }</li>
      }
    }
  </ul>
);

export default FrameworkList;
```

frameworklist-spec.js

```
describe("<FrameworkList />", () => {

  it("renders three frameworks", () => {

    const frameworks =
      ["react", "ember", "angular"];
    const wrapper = shallow(
      <FrameworkList
        frameworks={ frameworks } />
    );

    expect(wrapper.find(li))
      .to.have.length(3);
  });
});
```

Testing

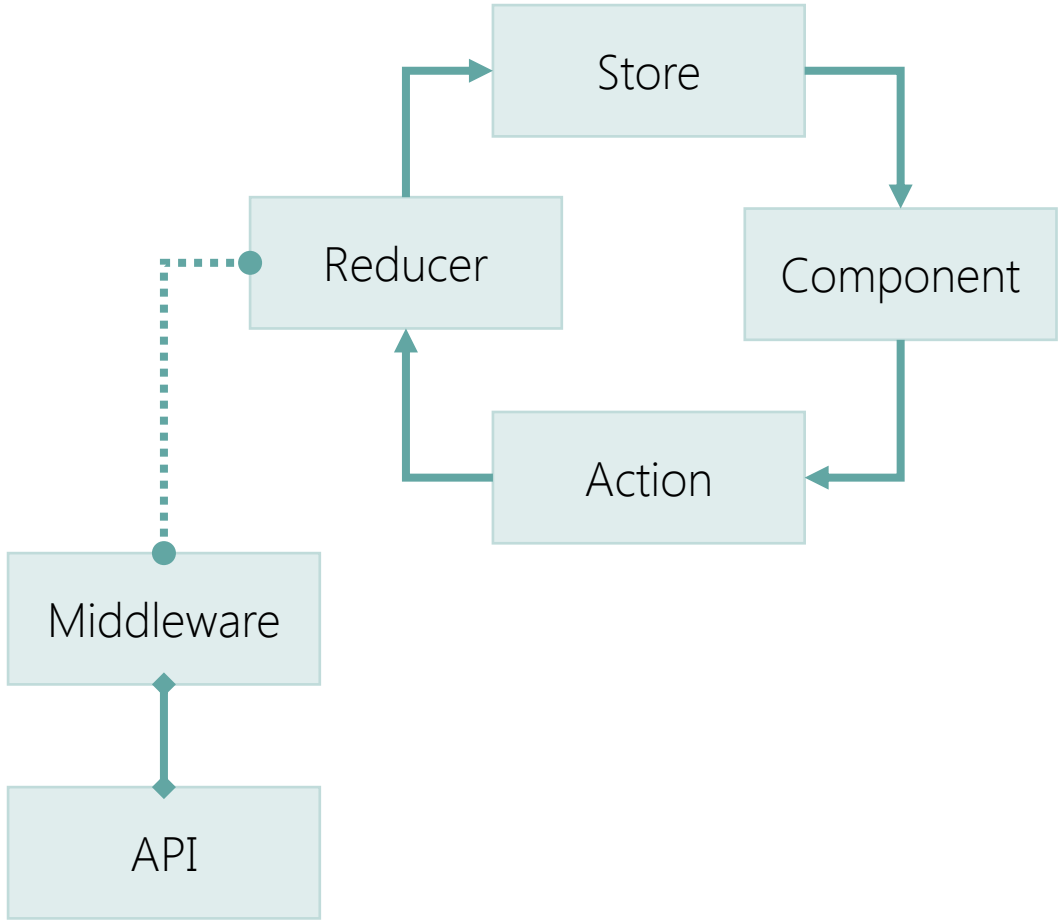
- Really important for JavaScript
- Lots of different tools but these are objectively the best
  - Mocha: Test runner
  - Chai: Assertions
  - Sinon: Spies
  - Enzyme: Component testing

package.json

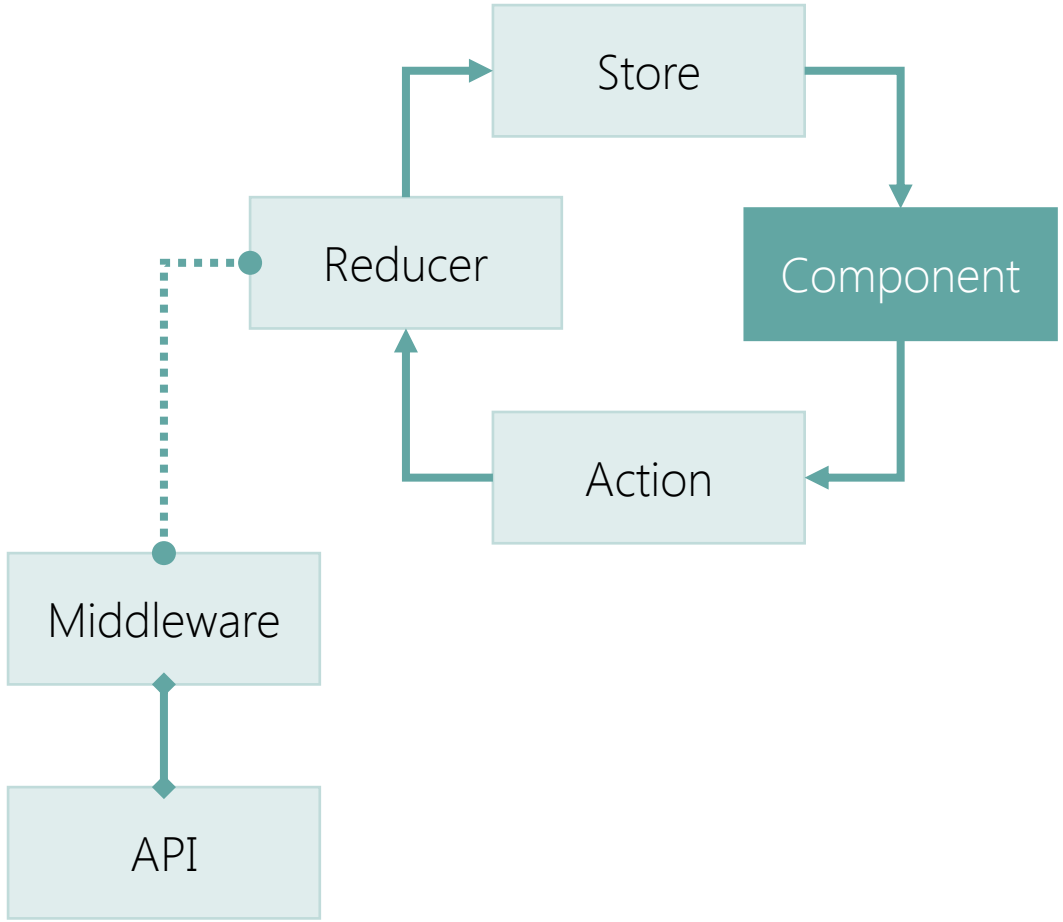
```
{
  ...
  "scripts": {
    "prepublish": "webpack",
    "start": "webpack-dev-server",
    "test": "mocha --compilers \
             js:babel-register"
  }
}
```

```
> npm test -- watch
<FrameworkList>
  ✓ renders all frameworks
```

Unidirectional dataflow

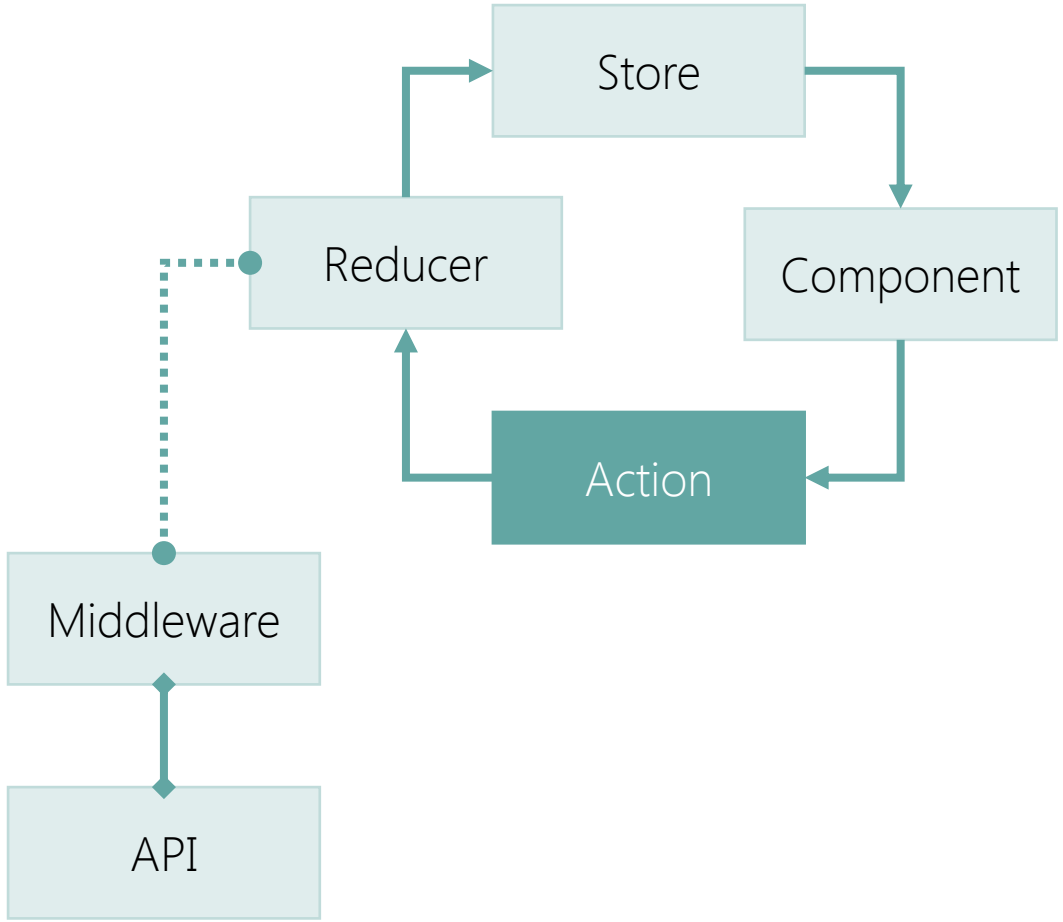


Unidirectional dataflow



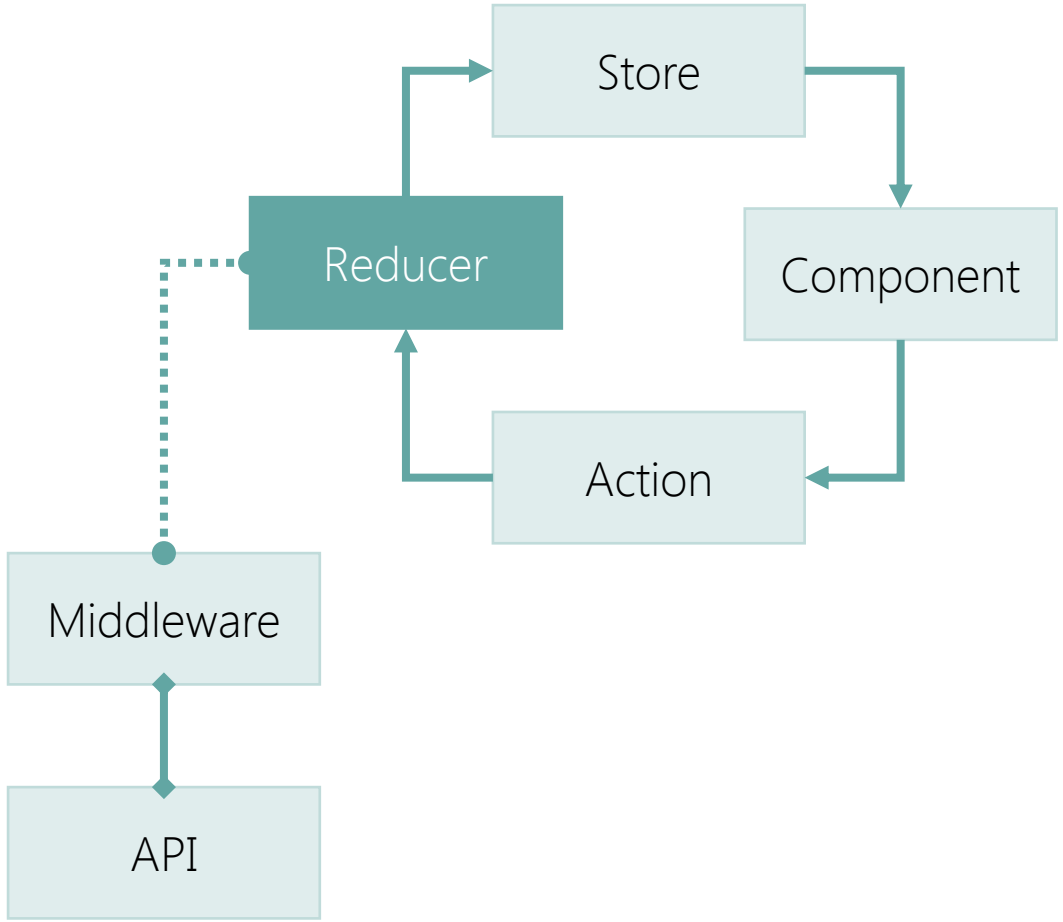

```
<Button onClick={ handleClick }>
  Add Todo
</Button>

const handleClick = () => {
  dispatch({
    type: "ADD_TODO",
    text: "Try Redux"
  });
}
```

Unidirectional dataflow



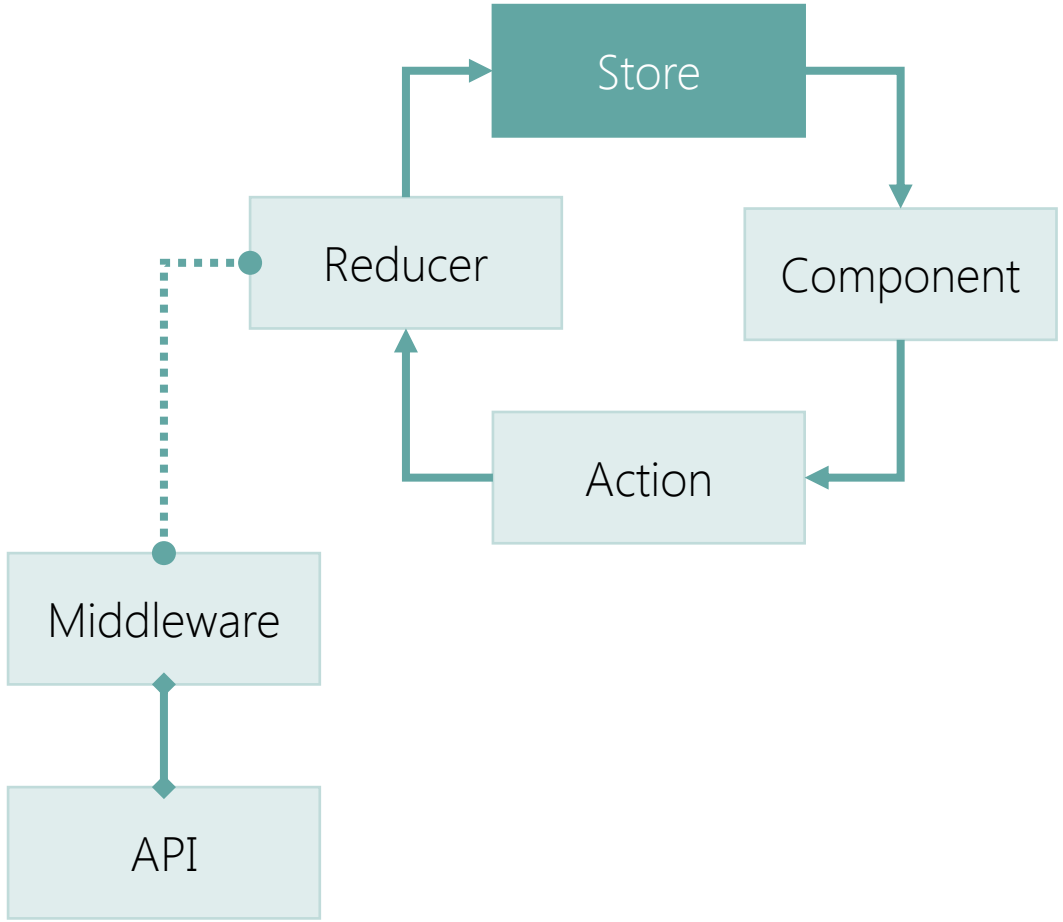

```
{
  type: "ADD_TODO",
  text: "Try Redux"
}
```

Unidirectional dataflow



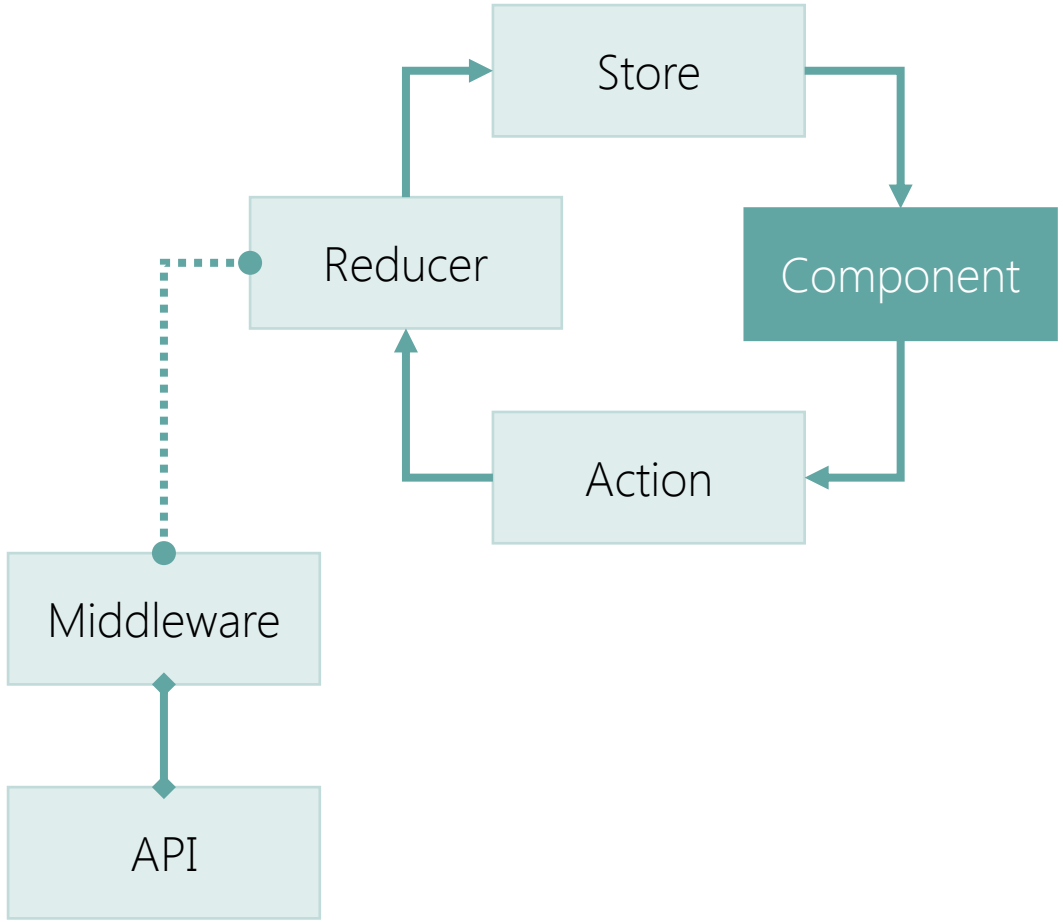

```
const initial = {
  todos: []
};

const todoApp = (state = initial, action) => {
  switch (action.type) {

    case "ADD_TODO": {
      return Object.assign({}, state, {
        todos: [
          ...state.todos,
          action.text
        });
    }

    default: return state;
  }
}
```

Unidirectional dataflow



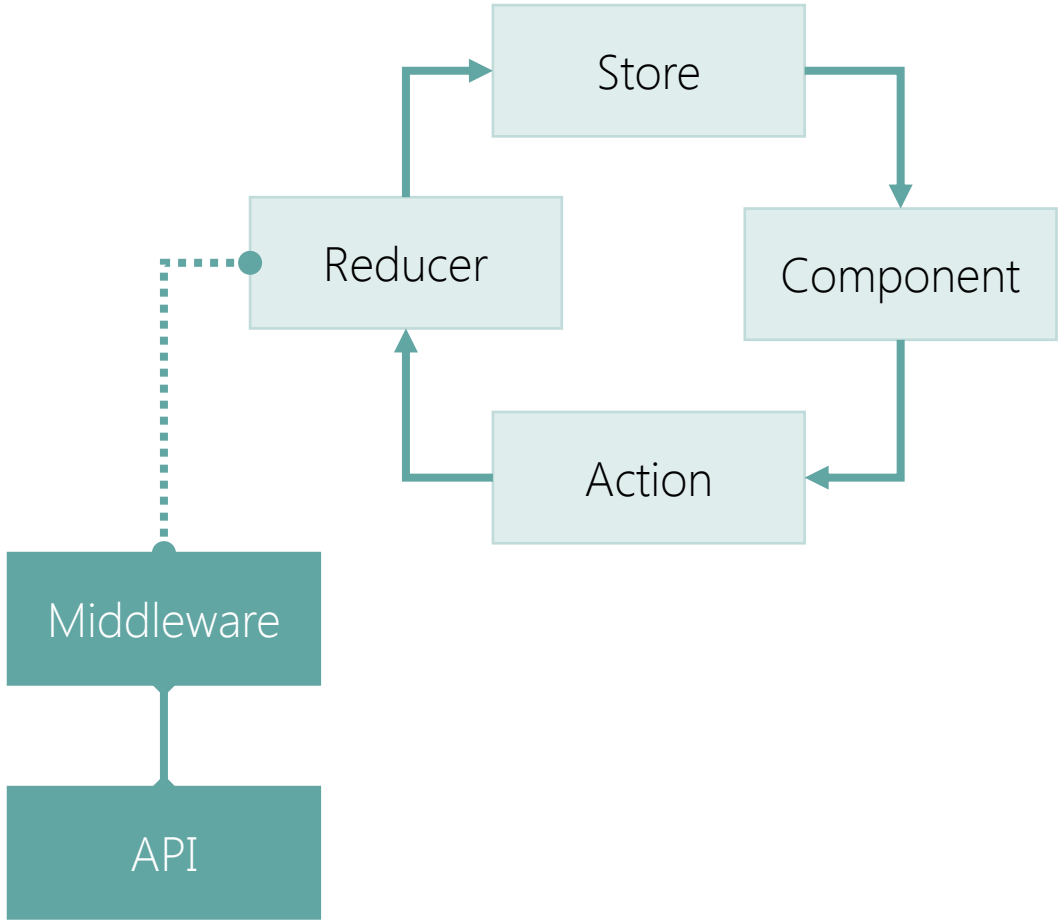

```
import { createStore } from "redux";
import { todoApp } from "./reducers";

const store = createStore(todoApp);
```

Unidirectional dataflow

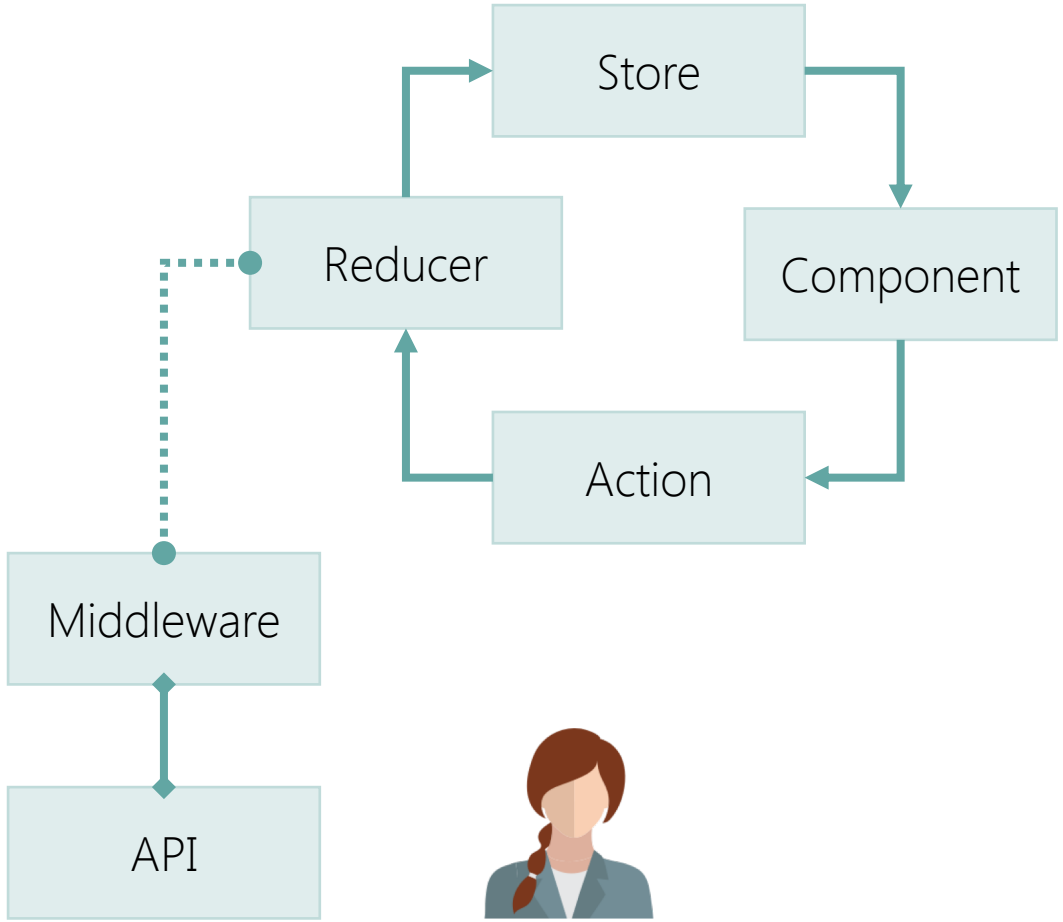



```
<Button onClick={ handleClick }>
  Add Todo
</Button>

const handleClick = () => {
  store.dispatch({
    type: "ADD_TODO",
    text: "Try Redux"
  });
}

store.subscribe(() => {
  store.getState();
  // Update component
});
```

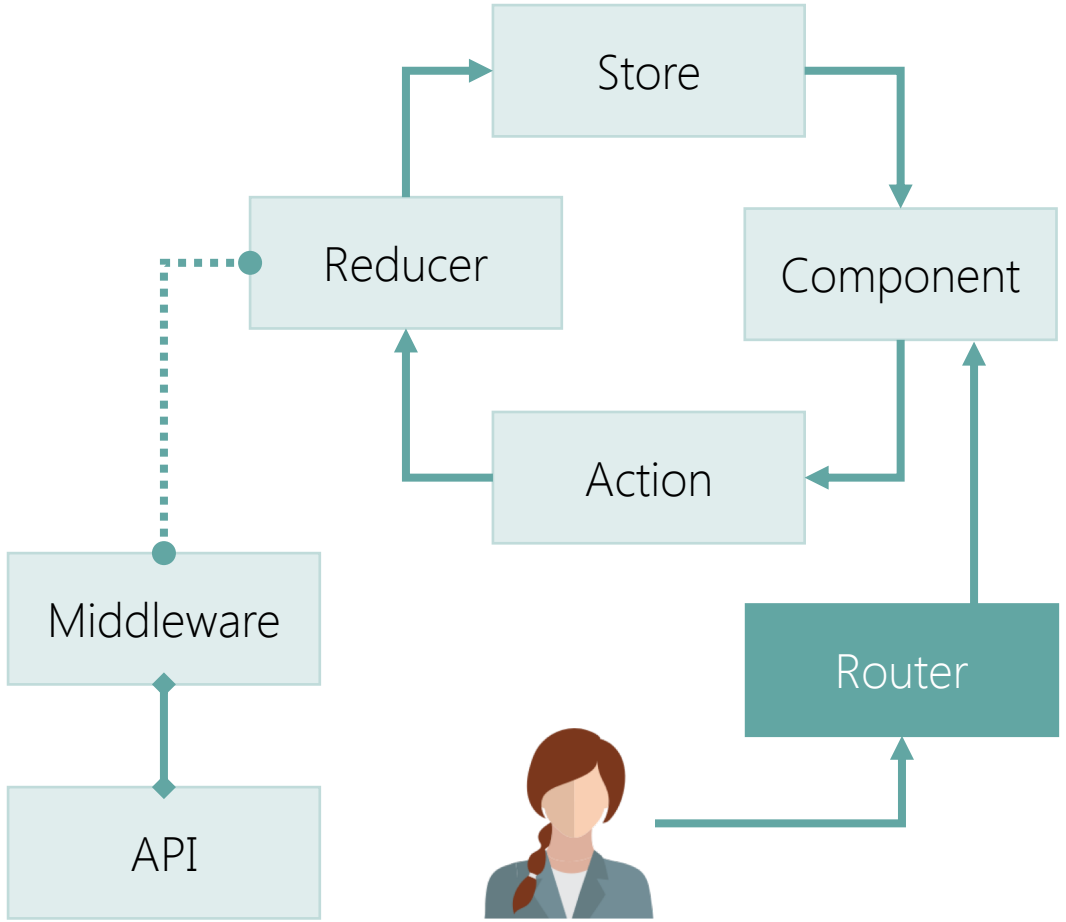
Unidirectional dataflow



What is missing?



Routing



Frameworks

- Which framework shall we use?

Frameworks

React

+ Is leading the way
+ Huge community
+ Mature

– Weak architecture
– Tool fatigue

Ember

+ Strict architecture
+ Mature
+ Productive

– No momentum
– Handlebars

Angular 2

+ Rethinking Angular
+ Components
+ TypeScript

– Immature
– I know Angular?

Summary

- JavaScript is growing up
  - The language is getting better
  - The tools are really great
  - Best practices are starting to emerge
- We need to grow up
  - Don’t focus on the frameworks
  - New frameworks pop up every day
  - Pick one and learn it really well

Thank you for coming

Links

- ECMAScript
  - https://github.com/lukehoban/es6features
  - https://github.com/tc39/ecma262
- Tools
  - http://babeljs.io
  - https://webpack.github.io
  - http://eslint.org
- Redux
  - http://redux.js.org
- Frameworks
  - http://facebook.github.io/react
  - http://emberjs.com
  - https://angular.io
- Test
  - http://mochajs.org
  - http://chaijs.com
  - http://sinonjs.org
  - https://github.com/airbnb/enzyme