

React &

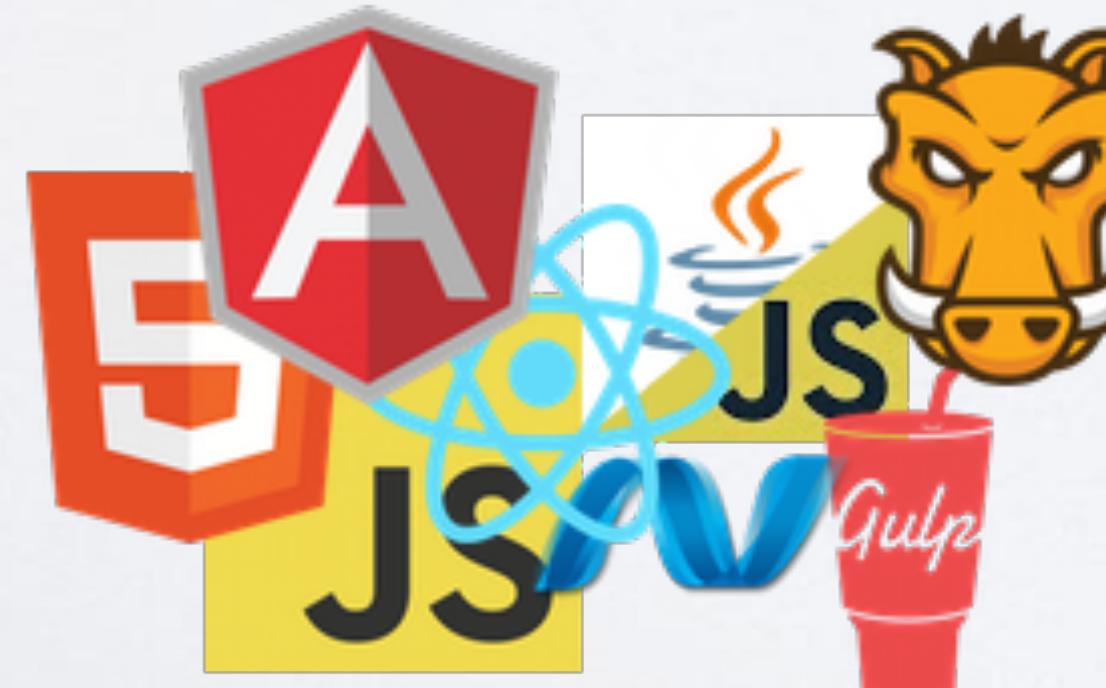


Redux

ABOUT ME

Jonas Bandi
jonas.bandi@ivorycode.com
Twitter: @jbandi

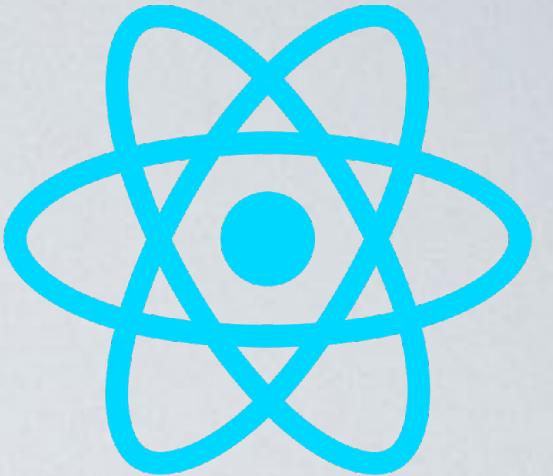
- Freelancer: www.ivorycode.com
- Dozent an der Berner Fachhochschule seit 2007
- In-House Kurse & Coachings: Web-Technologien im Enterprise (UBS, Postfinance, Mobiliar, BIT, SBB ...)



JavaScript / Angular 2 / React
Schulungen & Coachings
Project-Setup & Proof-of-Concept:
<http://ivorycode.com/#schulung>
jonas.bandi@ivorycode.com

Agenda

- React: Introduction & Basic Concepts
- React Demo
- Redux: Introduction & Basic Concepts
- Redux Demo
- Opinionated Comparison with Angular



What is React?

React is a JavaScript library for building user interfaces.

Created by Facebook in 2013.

The DOM is created programmatically.

JSX enables declarative DOM programming.

A virtual DOM abstracts the real DOM.

One-way data flow instead of two-way data binding.

React is Very Popular



amazon
Microsoft



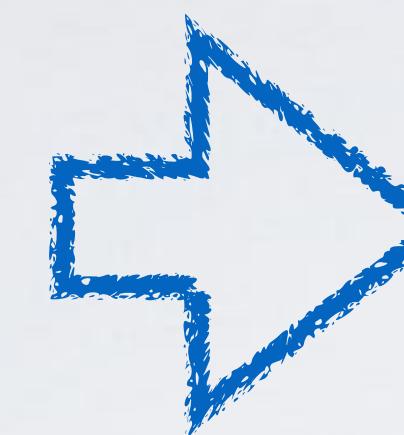
facebook / react	Star 75,060
vuejs / vue	Star 65,846
angular / angular	Star 27,553

JSX

```
class GreeterComponent extends React.Component {  
  
  state = {time: new Date()};  
  
  render() {  
    const now = new Date();  
  
    return (  
      <div>  
        <h1>{this.props.title}</h1>  
        <p>{this.props.message}</p>  
        <p>The current time is:</p>  
        <p>{this.state.time.toLocaleString()}</p>  
      </div>  
    );  
  }  
}
```

- JSX is a XML-like syntax extension to ECMAScript which defines a for defining tree structures
- JSX can declare HTML in JavaScript
- JSX is not an embedded HTML template. It gets compiled to JavaScript statements which use the React API to manipulate the DOM.

A React Component

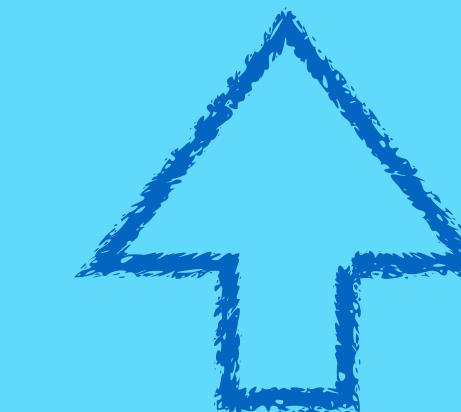


properties

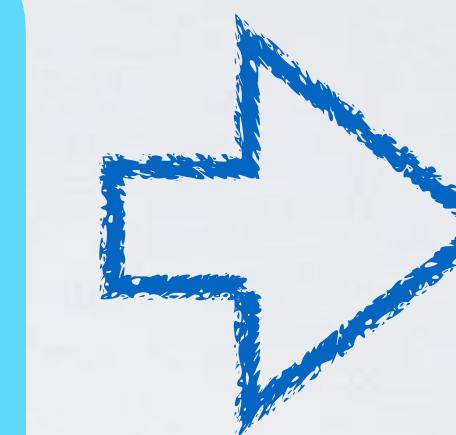
properties represent data
owned by someone else and
should not be changed

If properties or state
change, the component is
(re-)rendered

```
class HelloMessage extends React.Component{  
  ...  
  render() {  
    return <div>  
      Hello {this.props.props}  
      + {this.state.name}  
    </div>;  
  }  
};
```



state



html

components can be stateful

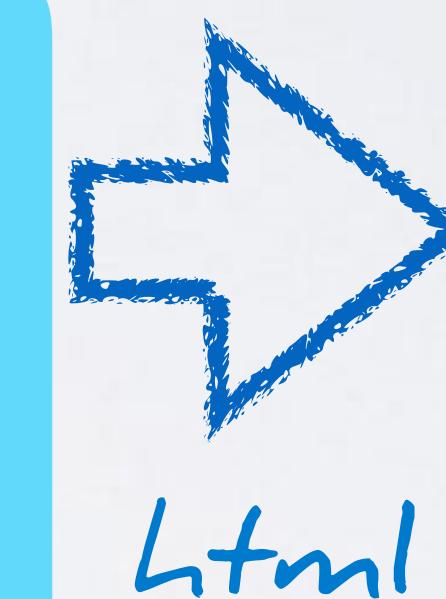
state is managed and changed
by the component itself

A Stateless Functional Component

Stateless components can be written as plain JavaScript functions.

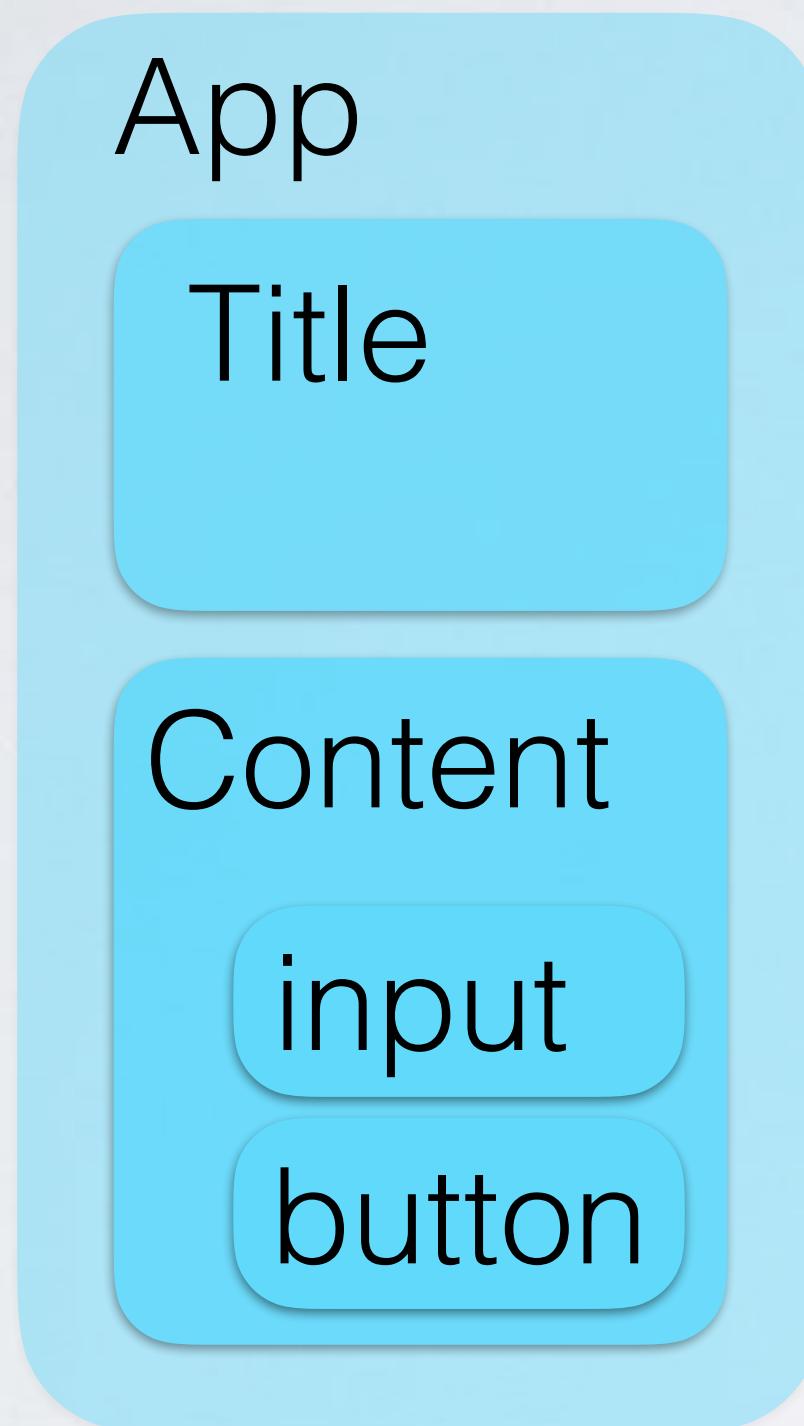


```
const AppComponent = (props) => (
  <div>
    <h1>{props.title}</h1>
    <p>{props.message}</p>
  </div>
);
```



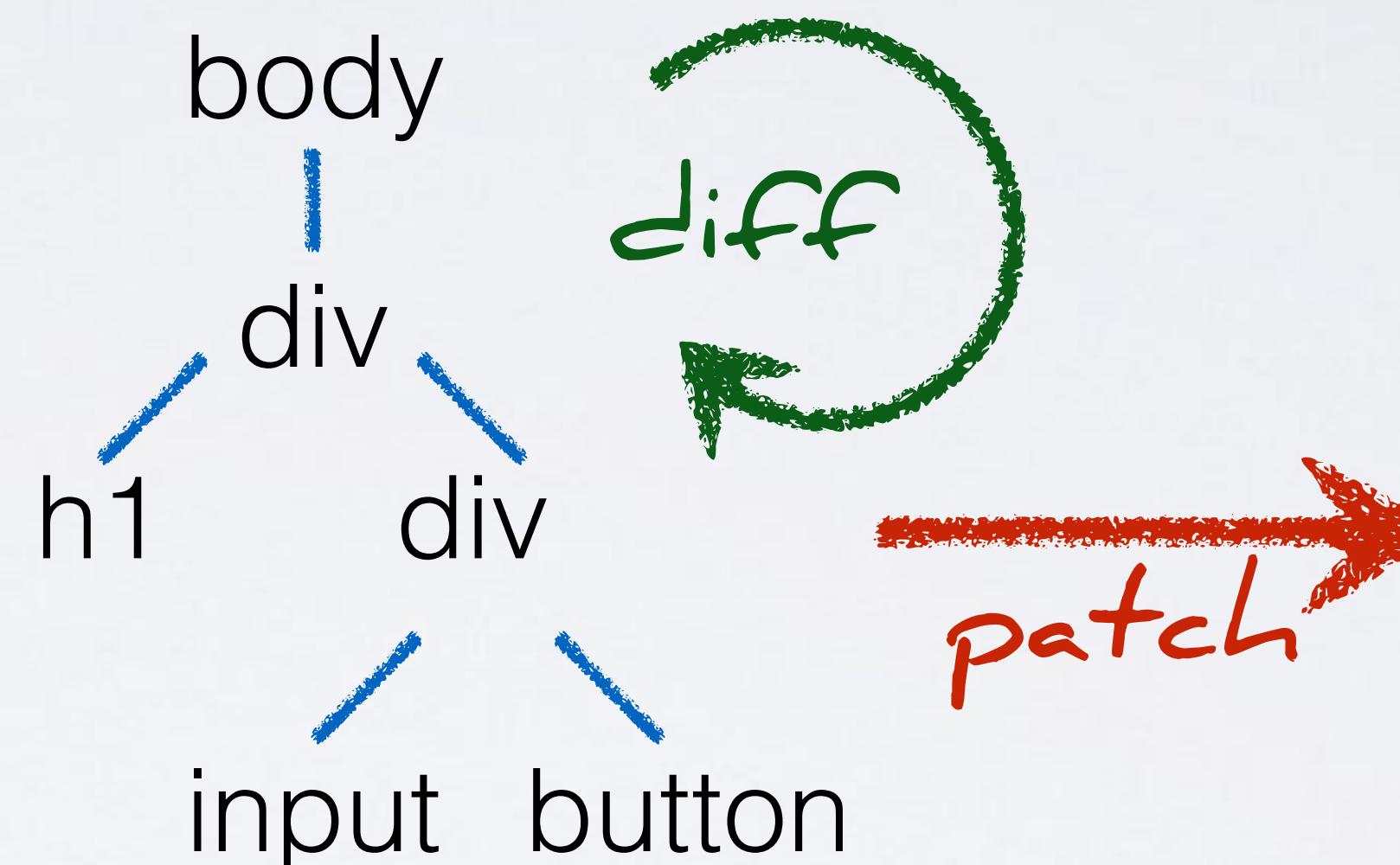
The Virtual DOM

Components



Virtual DOM

In-Memory, implemented in JavaScript



Browser DOM

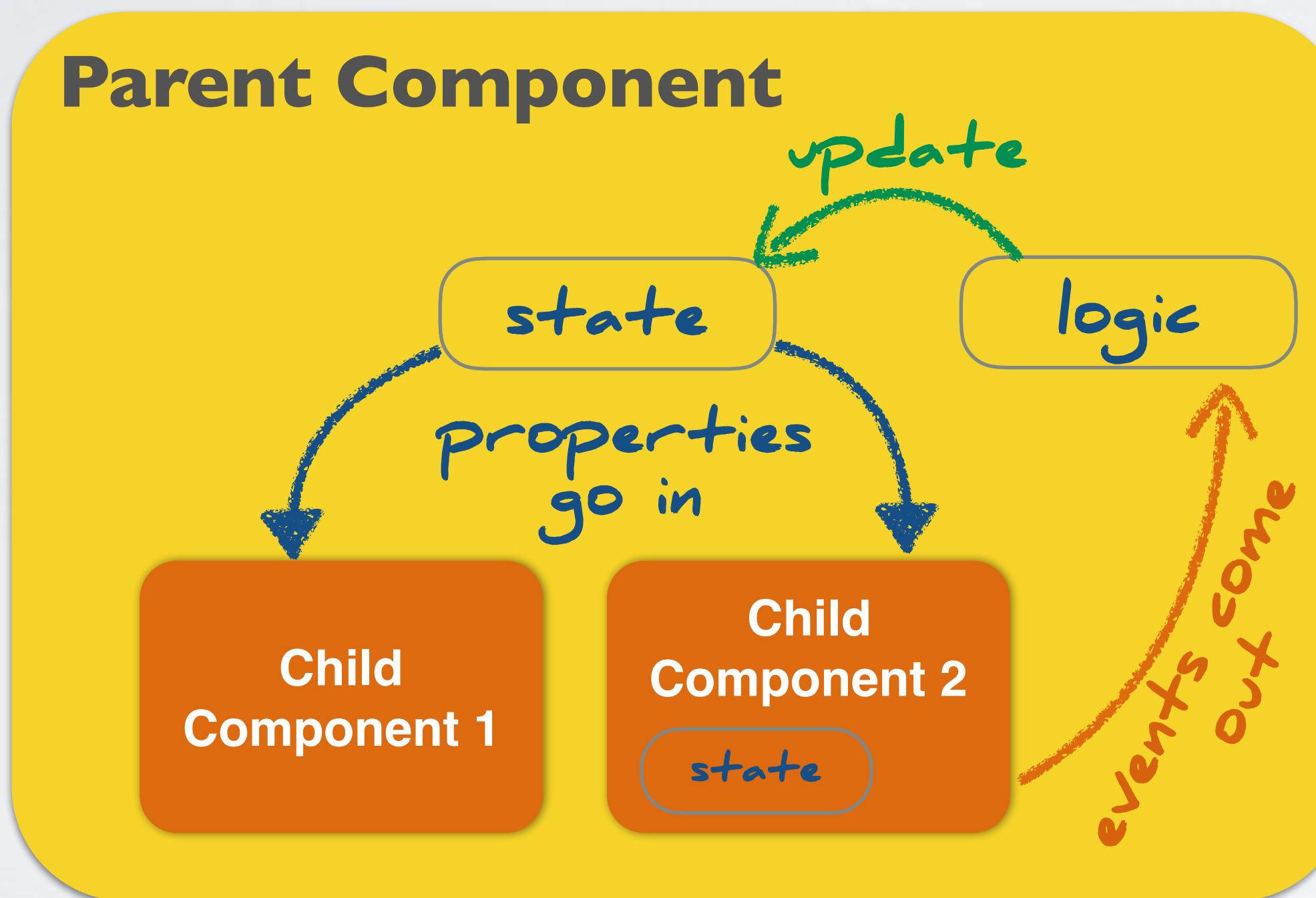
```
<body>
  <div>
    <h1></h1>
    <div>
      <input/>
      <button>
        </button>
      </div>
    </div>
  </body>
```

Virtual DOM also enables server-side rendering.

Nested Components: Unidirectional Data-Flow

State should be explicitly owned by a component.

No 2-way databinding!



- A parent component passes state to children
- Children should not edit state of their parent
- Children “notify” parents (events, actions ...)

React formalises unidirectional data-flow via properties and events.

Separation of Concerns

Separation of concerns is not equal to separation of file types!
Keep things together that change together.

You can split a component into a controller and a view:

```
import View from './View';

export class GiphySearch extends Component {
  ... // controller logic & state
  render(){
    return <View data={...} ...
  }
}
```

GiphySearch.js

```
const View = ({data}) => (
  <div>
    {data.message}
  </div>
);
export default View;
```

View.js

Container vs. Presentation Components

Application should be decomposed in container- and presentation components:

Container

Little to no markup
Pass data and actions down
typically stateful / manage state

Presentation

Mostly markup
Receive data & actions via props
mostly stateless
better reusability

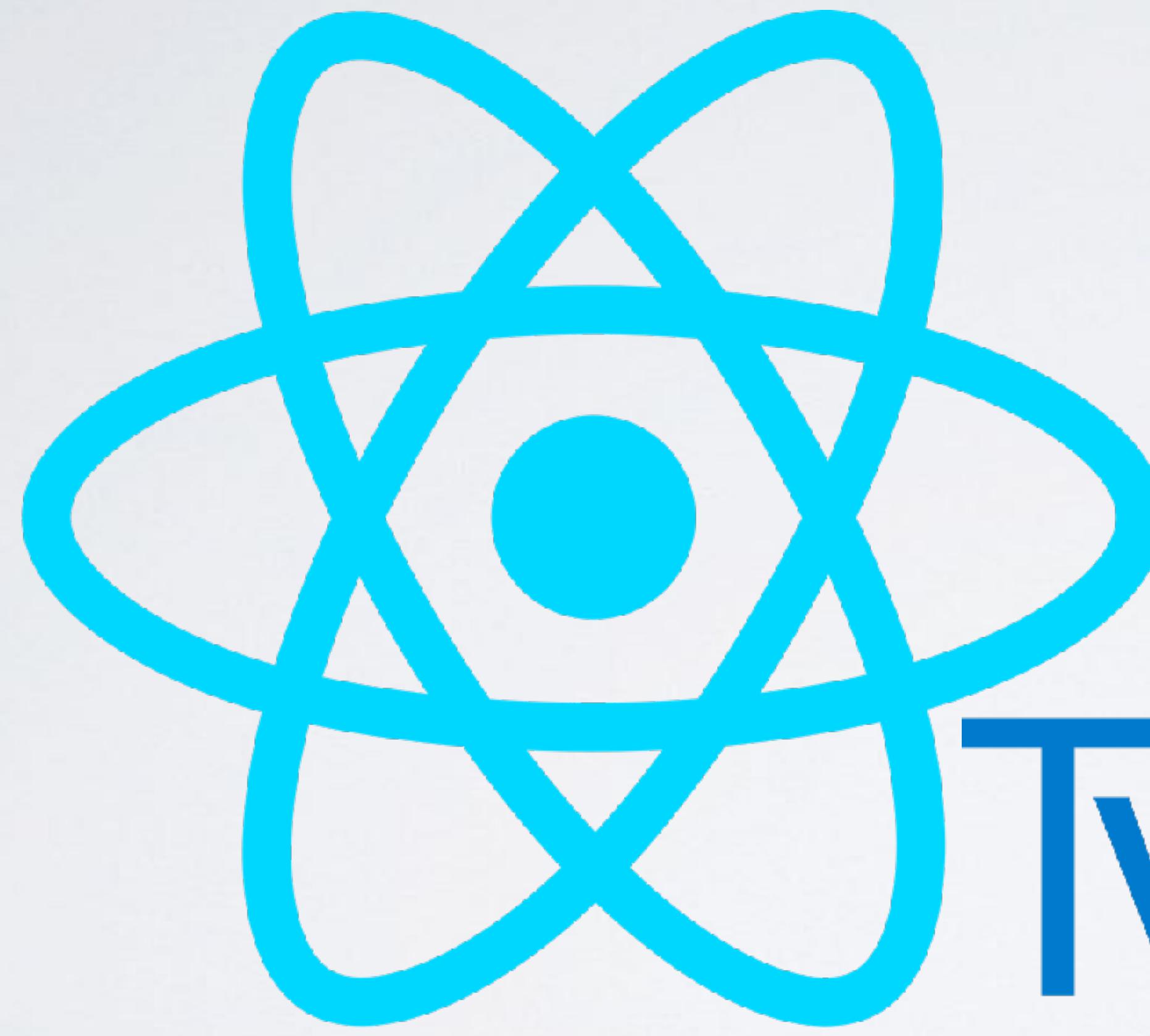
aka: Smart- vs. Dumb Components

https://medium.com/@dan_abramov/smart-and-dumb-components-7ca2f9a7c7d0

<https://medium.com/curated-by-versett/building-maintainable-angular-2-applications-5b9ec4b463a1>

<https://www.robinwieruch.de/learn-react-before-using-redux/>

React works great with TypeScript!





What is Redux?

Redux is a predictable state container for JavaScript apps.

Created in 2015 by Dan Abramov.

Redux itself is the implementation of a very simple concept.

There is a big ecosystem around Redux (middleware).

Redux is very small (~2KB).

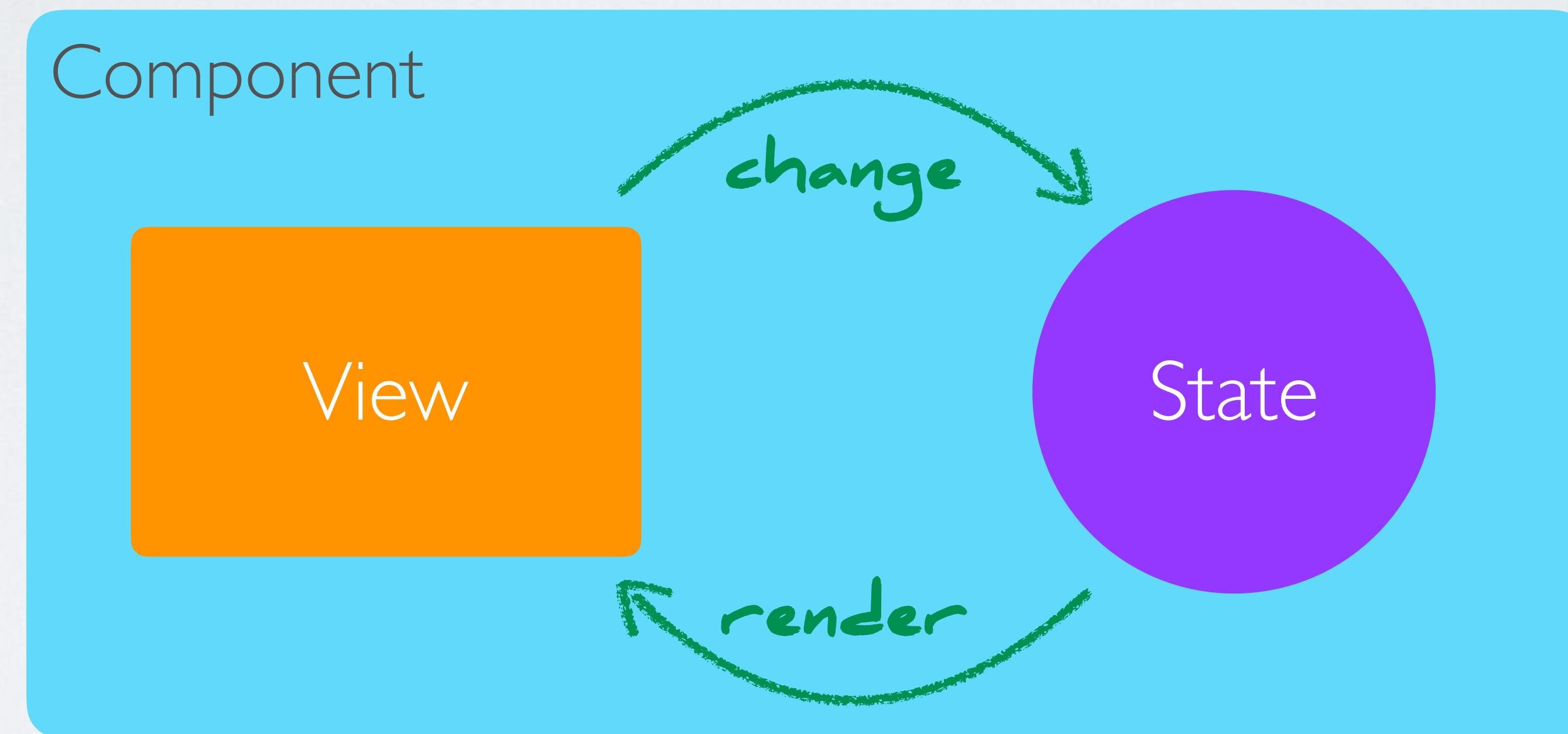
Redux is widely used in the React community. Technically it is not tied to React.

Redux can be used with Angular. For Angular there is also ngrx, which is an alternative implementation of the same concept.

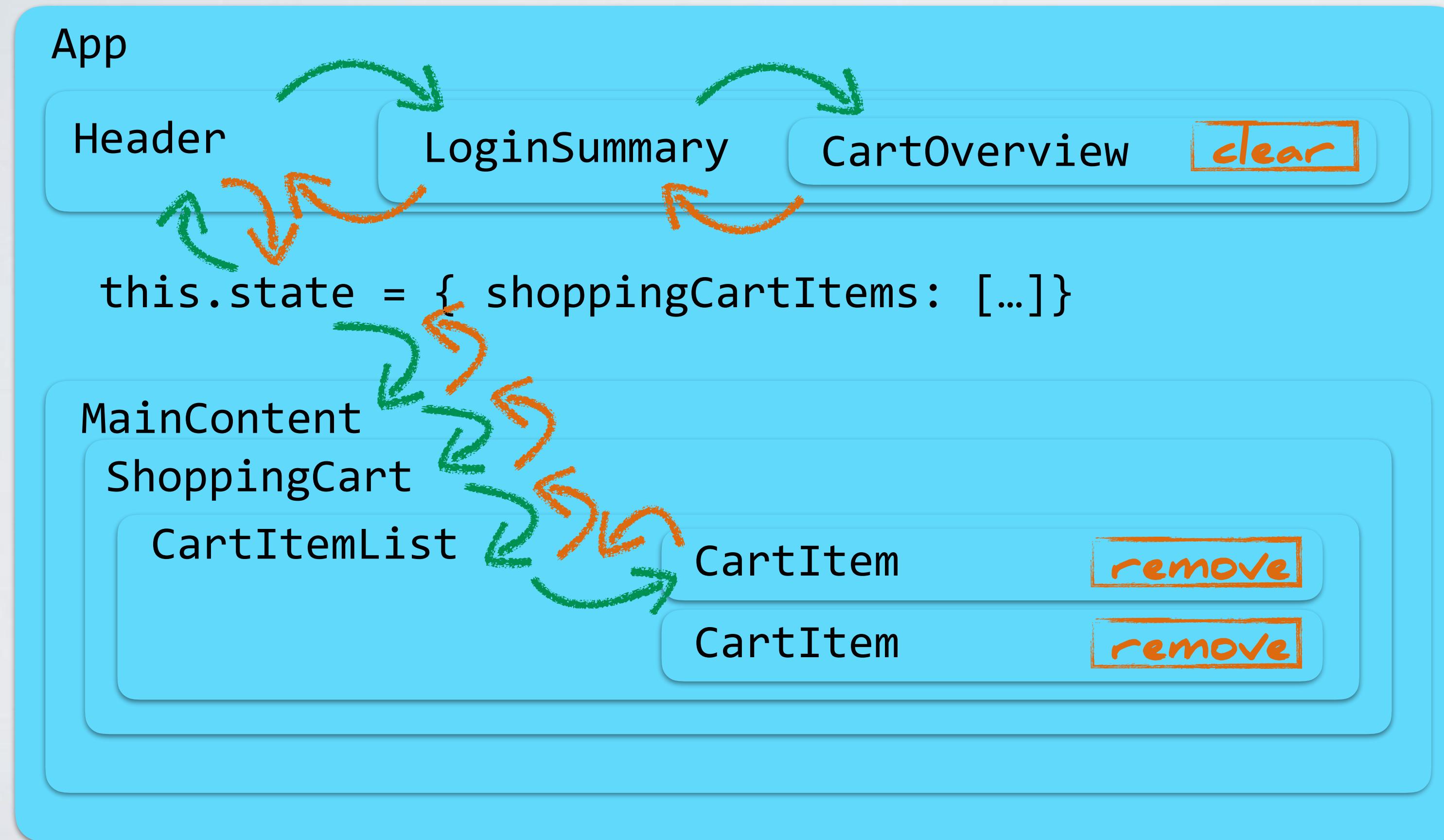
(<https://github.com/angular-redux/ng-redux> and <https://github.com/ngrx>)

A single component

Managing state in a component is simple:



State Management

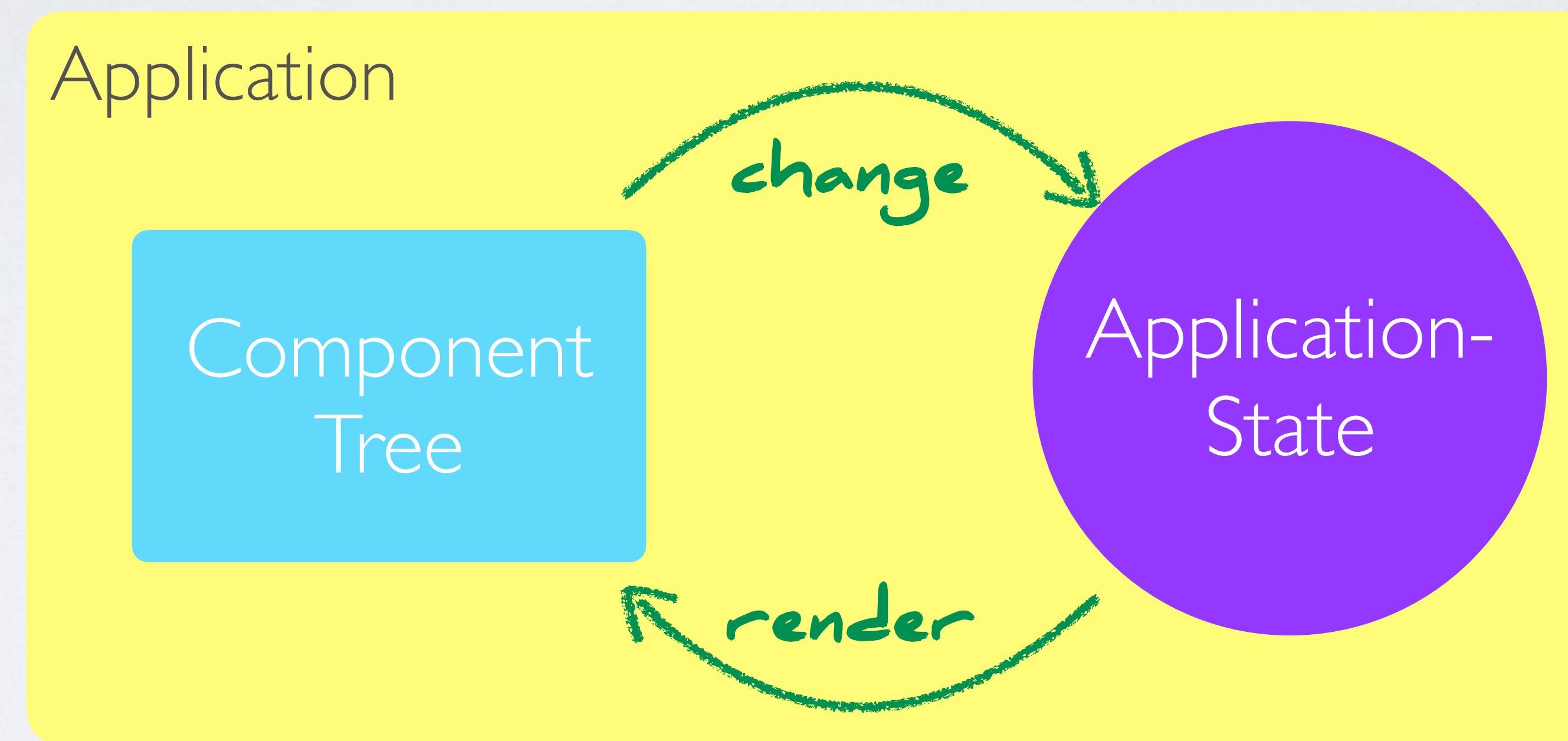


Challenges when managing state in a component tree:

- Multiple components may depend on the same piece of state.
- Different components may need to mutate the same piece of state.

Application with State Container

A state container extracts the shared state out of the components, and manages it in a global singleton.

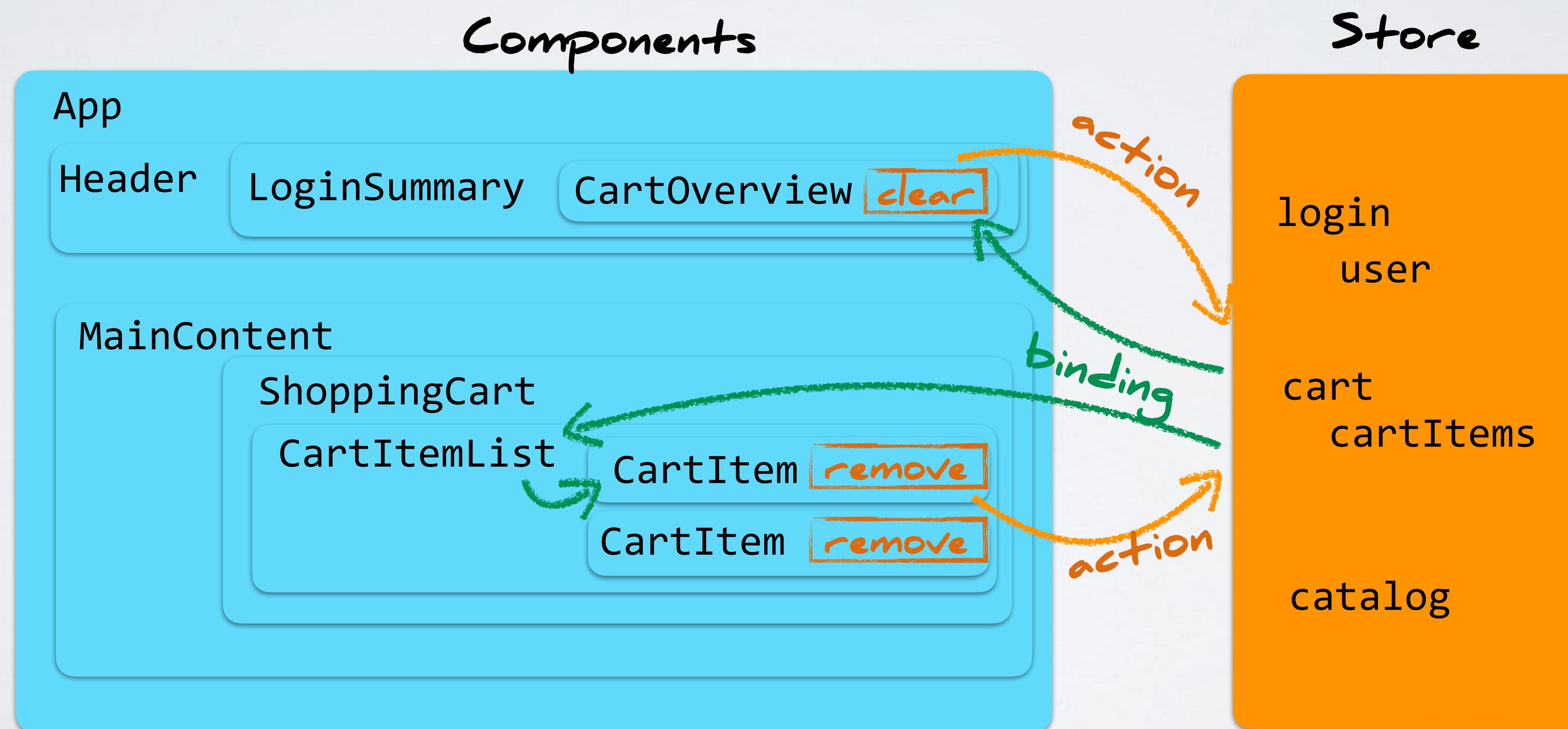


The component tree becomes a big "view", and any component can access the state or trigger actions, no matter where they are in the tree!

Managing State with a State Container

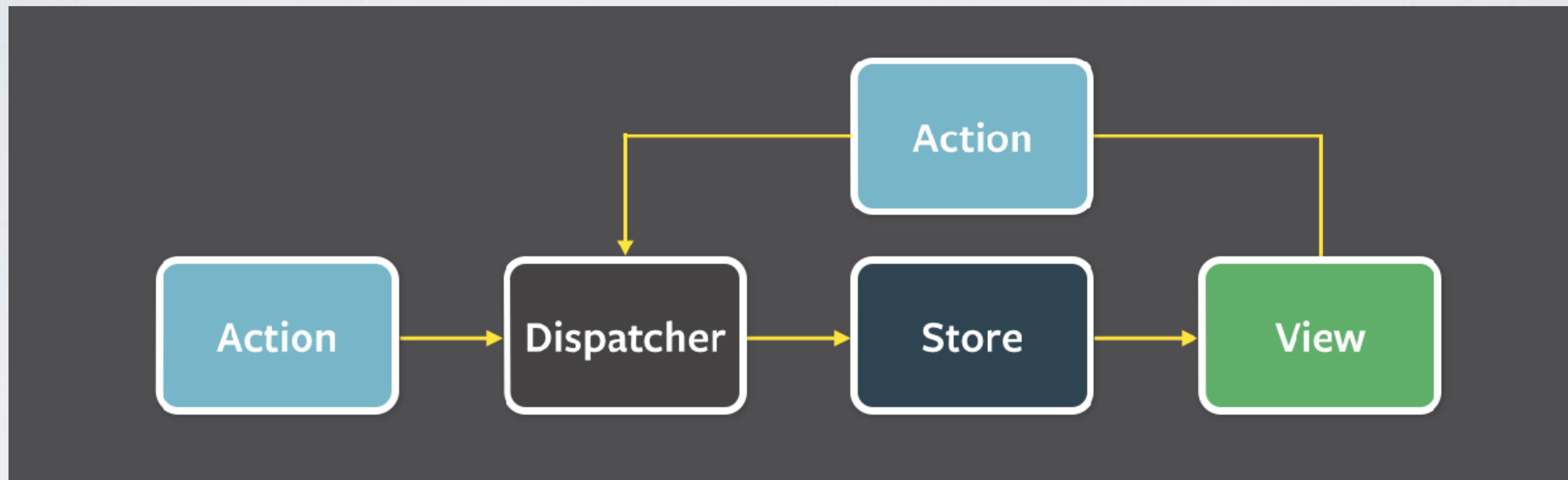
State can be managed outside the components.

Components are bound to state.



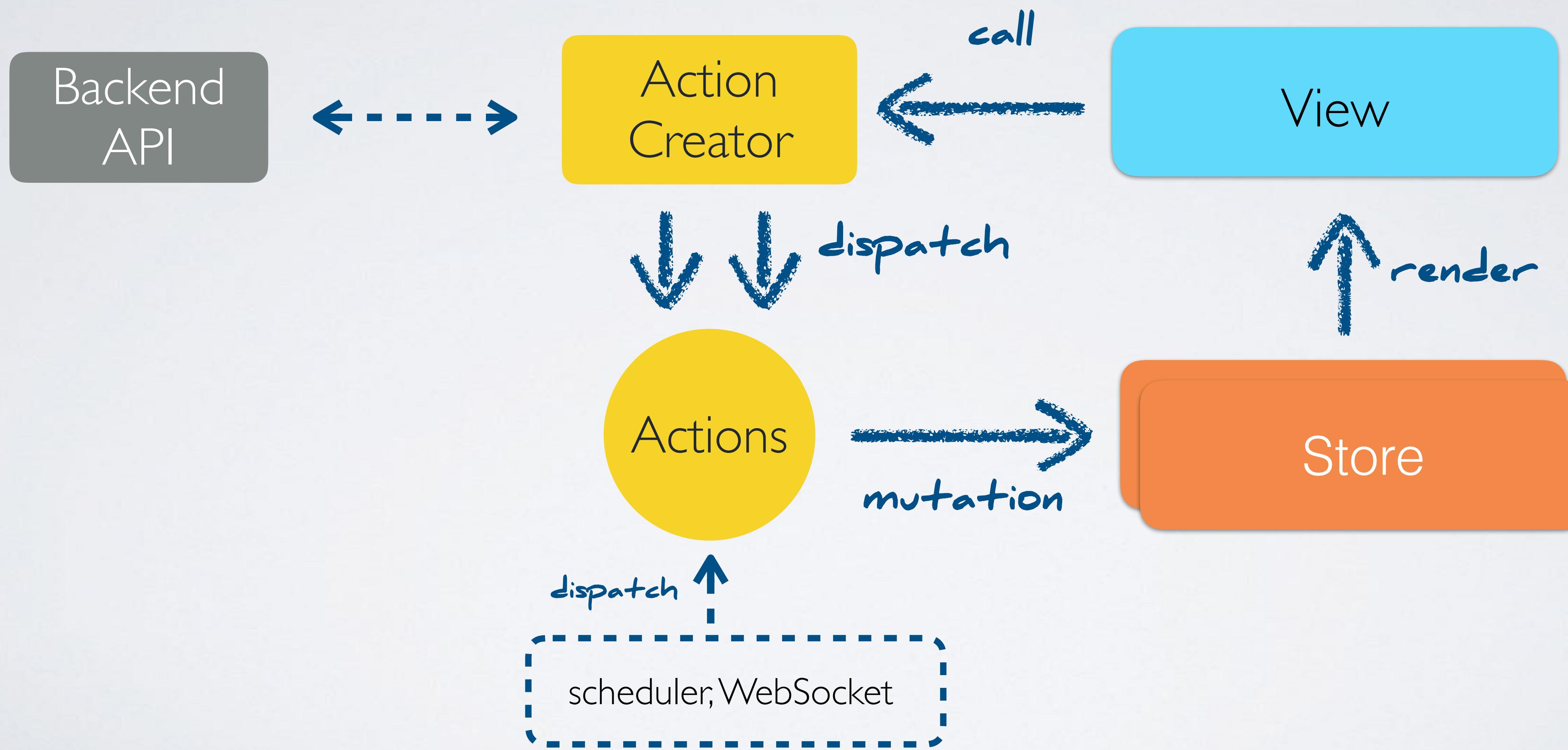
Flux Architecture

Formalization of unidirectional data-flow.

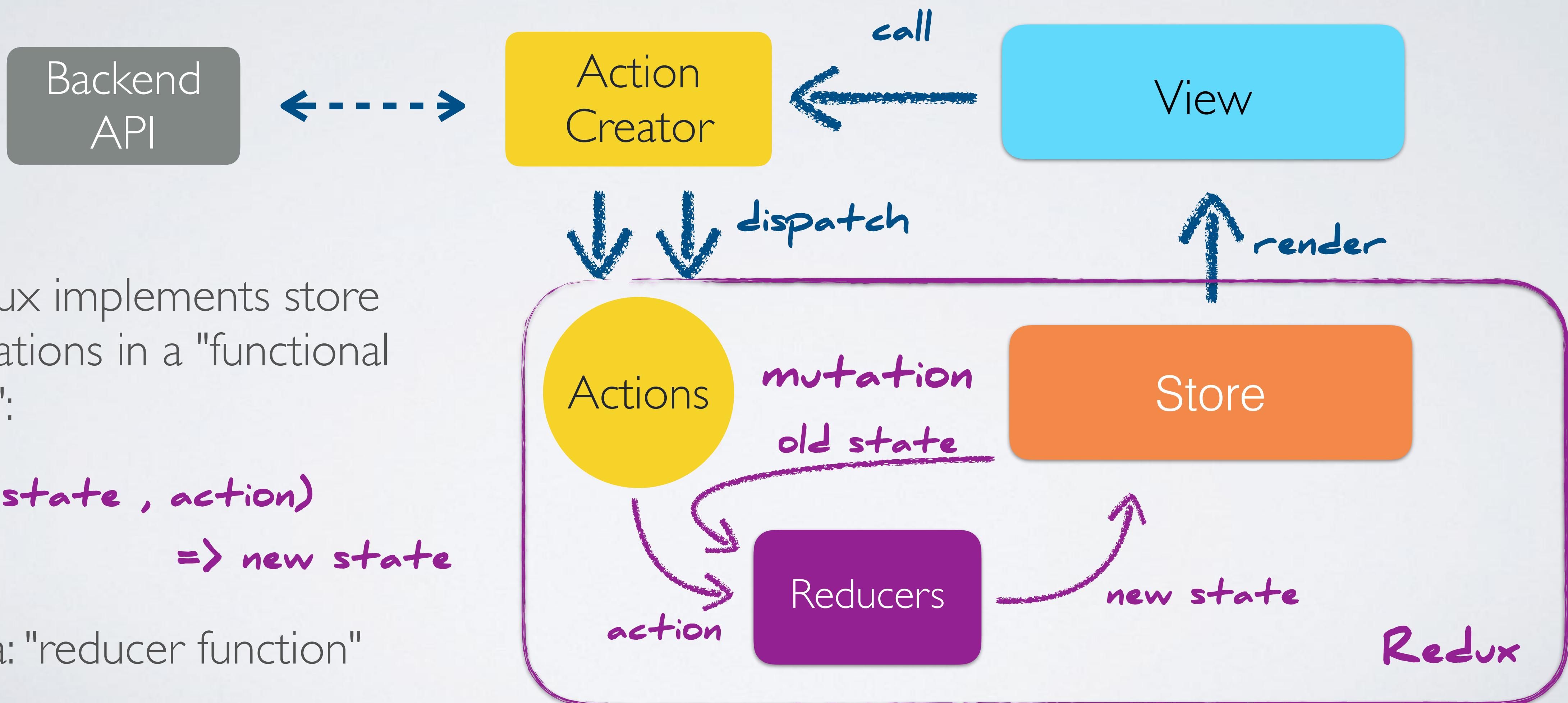


- Stores hold state that can be used from several views.
- Only actions can modify store states. Stores have no setters.
- Stores subscribe to actions and notify views if their state changed.
- Views trigger the rendering if a store changed.
- Actions are fire-and-forget.

Flux Architecture



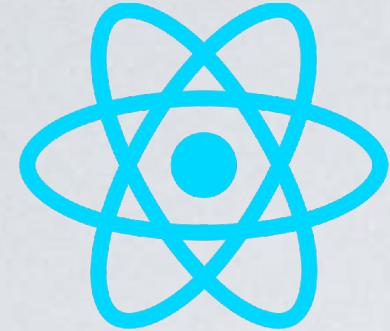
Redux Architecture





React vs. Angular
(personal opinion)

Framework vs. Library



React and Angular have different scopes:



UI-Library

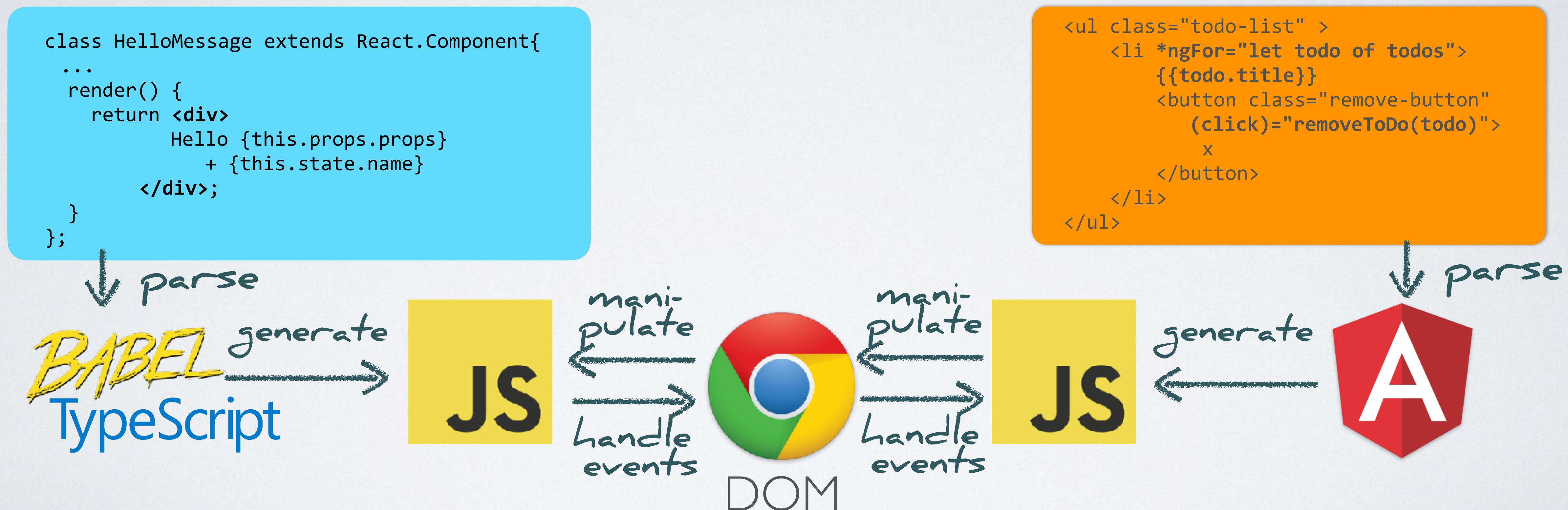
- UI-Components
- Unidirectional DataFlow Architecture

Complete SPA Framework

- UI-Components
- Unidirectional DataFlow Architecture
- Routing
- Backend-Access
- Architecture:
 - Services
 - Dependency-Injection
 - Modularization

The UI is rendered with JS

In Angular and in React the DOM is built with imperative JavaScript that is loaded into the browser.



React enhances JavaScript in
order to declare the UI

Angular enhances HTML with a
new language

It's just JavaScript: Rendering

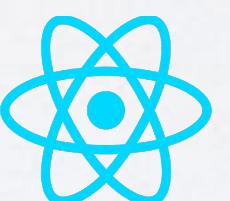
- Leverage the full power of JavaScript, no need for a new syntax.

Bonus: build-time syntax checkin of your templates

- JavaScript is much more versatile than HTML.

Trying to fit imperative constructs into a declarative language is cumbersome (does anybody remember Ant?)

- Angular templating/directives feel clumsy ...



```
render(){  
  let message;  
  if (this.props.loading) message = <p>Loading...</p>;  
  else message = <div>this.state.message</div>;  
  return message;  
}
```

if-else in React

```
<p *ngIf="loading; else notLoading">  
  Loading...  
</p>  
<ng-template #notLoading>  
  <div>{{message}}</div>  
</ng-template>
```

if-else added in Angular 4

Angular Templating: a new Language

```
<div [ngSwitch]="conditionExpression">  
<ng-template [ngSwitchCase]="case1Exp">...</ng-template>  
<ng-template ngSwitchCase="case2LiteralString">...</ng-template>  
<ng-template ngSwitchDefault>...</ng-template>  
</div>
```

```
<my-cmp [(title)]="name">  
<input [(ngModel)]="userName">
```

```
<div [style.width.px]="mySize">
```

```
<div [ngClass]="{{ 'active': isActive, 'disabled': isDisabled }}>
```

```
<li*ngFor="let contact of contacts | async; let o = odd; let e = even;">  
...  
</li>
```

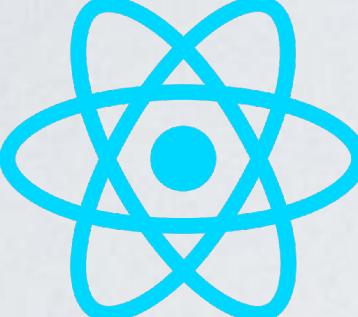
```
<ng-template ngFor let-contact [ngForOf]="contacts | async"> ... </ng-template>
```

```
<button (click)="do($event)">
```

```
<div *ngIf="things.car; let car">  
  Nice {{ car }}!  
</div>
```

It's just JavaScript: Dependencies

- Dependencies can be statically analyzed in JavaScript
- No additional module concept that 'magically' provides components



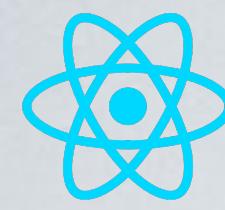
```
import MyComponent from './MyComponent';
...
<MyComponent
  data={currentValue}
  onSelect={handleSelect} />
```



```
<my-component
  [data]="currentValue"
  (onSelect)="handleSelect($target)" />
```

```
@Component({
  selector: 'my-component',
  ...
})
class MyComponent {...}
```

```
import {MyComponent} from './MyComponent';
...
@NgModule({
  declarations: [
    MyComponent
  ]
  ...
})
export class AppModule {}
```



Declarative all the Way!



In React rendering can only be triggered by changing the state:

```
class Modal extends React.Component {  
  render() {  
    // Render nothing if the "show" prop is false  
    if(!this.props.show) {  
      return null;  
    }  
    ...  
  }  
}  
  
class App extends Component {  
  state = { isOpen: false };  
  toggleModal = () => {  
    this.setState({isOpen: !this.state.isOpen});  
  }  
  render() {  
    return (  
      <Modal show={this.state.isOpen} />  
    );  
  }  
}
```

<https://daveceddia.com/open-modal-in-react/>

Angular mixes declarative and imperative rendering:

```
export class HomeComponent implements OnInit {  
  private bodyText: string;  
  
  constructor(private modalService: ModalService) {}  
  
  ngOnInit() {  
    this.bodyText = 'Mesage 1';  
  }  
  openModal(id: string){  
    this.modalService.open(id);  
  }  
}
```

<http://plnkr.co/edit/gPCTvV?p=preview>

```
export class DashboardComponent {  
  @ViewChild("mixedChart") mixedChart: UIChart;  
  ...  
  onDataUpdated(){  
    this.mixedChart.refresh();  
  }  
}
```

<https://github.com/glenasmith/pluralsight-primeng>

API Surface

- React has very few but powerful primitives
- The API is simple and consistent
- Angular has a huge API, and the APIs are often overlapping or inconsistent
- Angular introduces complexity with doubtful benefits

Properties and Events

The Browser:

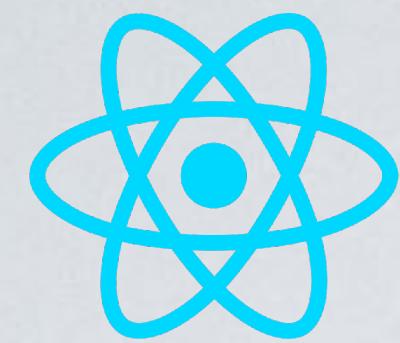
```
<input value="Test" onclick="alertHandler()">
```

React:

```
<input value={currentVal} onClick={alertHandler}>
```

Angular:

```
<input [value]="currentVal" (click)="alertHandler()">
```



Custom Components



```
import MyComponent from './MyComponent';
...
<MyComponent
  data={currentValue}
  onSelect={handleSelect} />
```

```
class MyComponent extends React.Component {
  ...
  trigger = () => {
    ...
    if (this.props.data != newVal){
      this.props.onSelect(newVal);
    }
  }
}
```

```
<my-component
  [data]="currentValue"
  (onSelect)="handleSelect($target)" />

@Component(
  selector: 'my-component',
  ...
)
class MyComponent
  @Input() data;
  @Output() onSelect = new EventEmitter();
  ...
  trigger(){
    ...
    if (this.data != newVal){
      this.onSelect.emit(newVal);
    }
  }
}
```

Complexity & Leaking Abstraction

 **Stephen Fluin**
@stephenfluin

[Follow](#)

How familiar are you with ngfactory files?

52% Not at all, should I be?

26% I think I heard of them

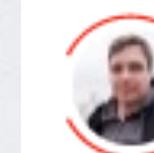
13% Yes and they don't matter

9% Yes and I hate them

128 votes • Final results

8:37 AM - 24 Aug 2017

 **Maxim Koretskyi** in *Angular In Depth*
Jul 10 • 3 min read

Would you buy a book on Angular change detection?



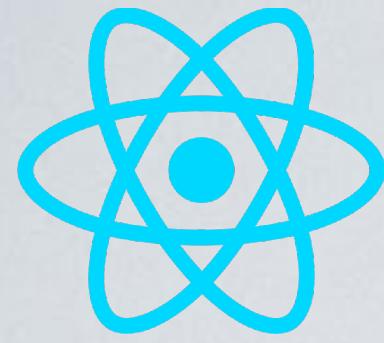
Read more...

 34

3 responses 



Unfortunate Design Decisions in Angular
(again: personal opinion)



Template Compilation



JSX is not supported in the browser.
A compiler (babel or typescript) is
used at build time to generate
JavaScript.

JavaScript is delivered to the browser.
Simple!

The official way to program Angular is to use
TypeScript.

TypeScript is not supported in the browser.
A compiler must be used at build time to
generate JavaScript.

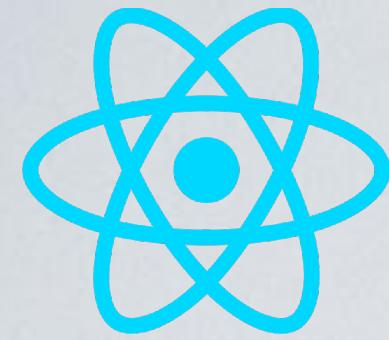
But we don't compile templates at build-time
... no no!

We ship the Angular compiler (70% of the
Angular code) to the browser.

We also ship templates as strings to the
browser.

Then we compile templates to JavaScript inside
the browser.

Let's give it a fancy name: JIT Mode.



Template Compilation ... continued!



Nothing more to say!

Ok, maybe there is a better way (we always had that in mind anyways ...). Let's compile at build time (ok, we are doing that already with TS anyway). Let's give it a fancy name: AOT mode.

Hmm ... let's make that the default in Angular 5.

Oh ... but we don't support the full JavaScript syntax in AOT mode ... sorry!

<https://github.com/rangle/angular-2-aot-sandbox>

Google: Consider replacing the function with a reference to an exported function

Change Detection

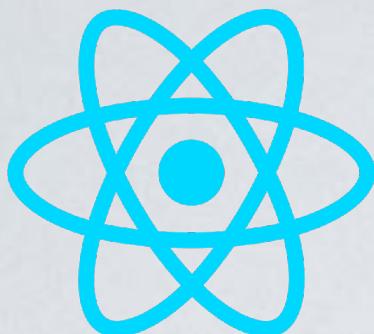
The goal of Angular change detection is to make development simple:
You change some JavaScript object and Angular updates the UI.
Magic!

In reality the abstraction is leaking and there are many reasons why
developers must start to care about change detection.

The concepts for change detection are very complex and a constant source
for confusion:

`ngDoCheck`, `ngOnChanges`, `ChangeDetectionStrategy.OnPush`,
`AppRef.tick()`, `NgZone.run()`, `ChangeDetectorRef.detach()`,
`ChangeDetectorRef.markForCheck()`

Routing & Unidirectional DataFlow

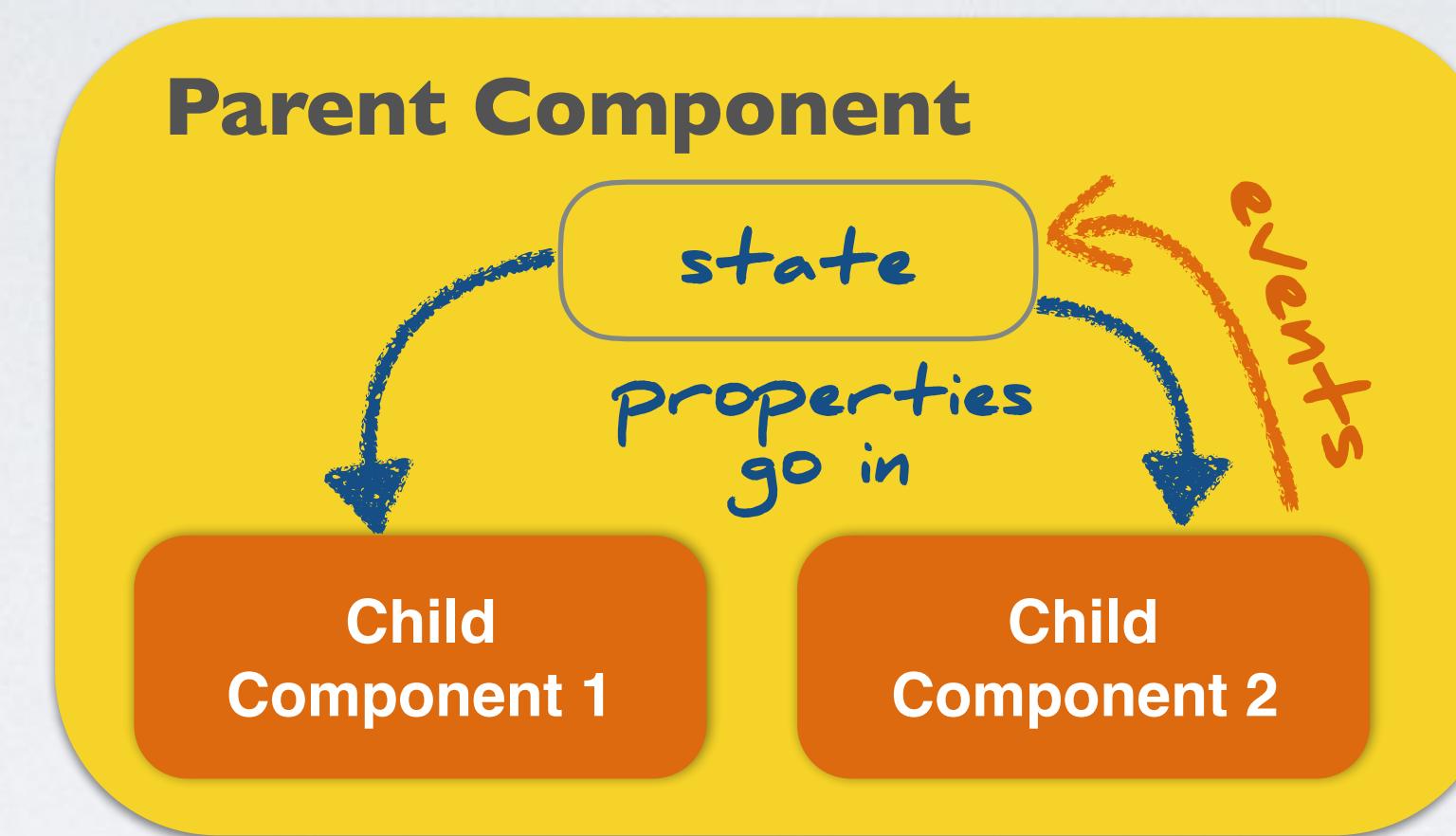


A route is just a component!

```
<Route exact path="/" render={(props) => (
  <Child1
    message={this.state.message1}
    notify={this.alert}
    {...props}
  />
)}
```



```
<Route path="/child2" render={(props) => (
  <Child2
    message={this.state.message2}
    notify={this.alert}
    {...props}/>
)}
```



Routing brakes the component architecture

```
const appRoutes: Routes = [
  { path: '/', component: Child1 },
  { path: 'child2', component: Child2 },
]
```

static route configuration

```
<h1>Parent</h1>
...
<router-outlet></router-outlet>
```

parent template

render function in parent



"no way for Parent to Pass properties or handle events"

Toolchain is not used internally at Google

- Google did not use TypeScript until beginning of this year
- Google is not using angular-cli
- Google is not using webpack and @ngtools

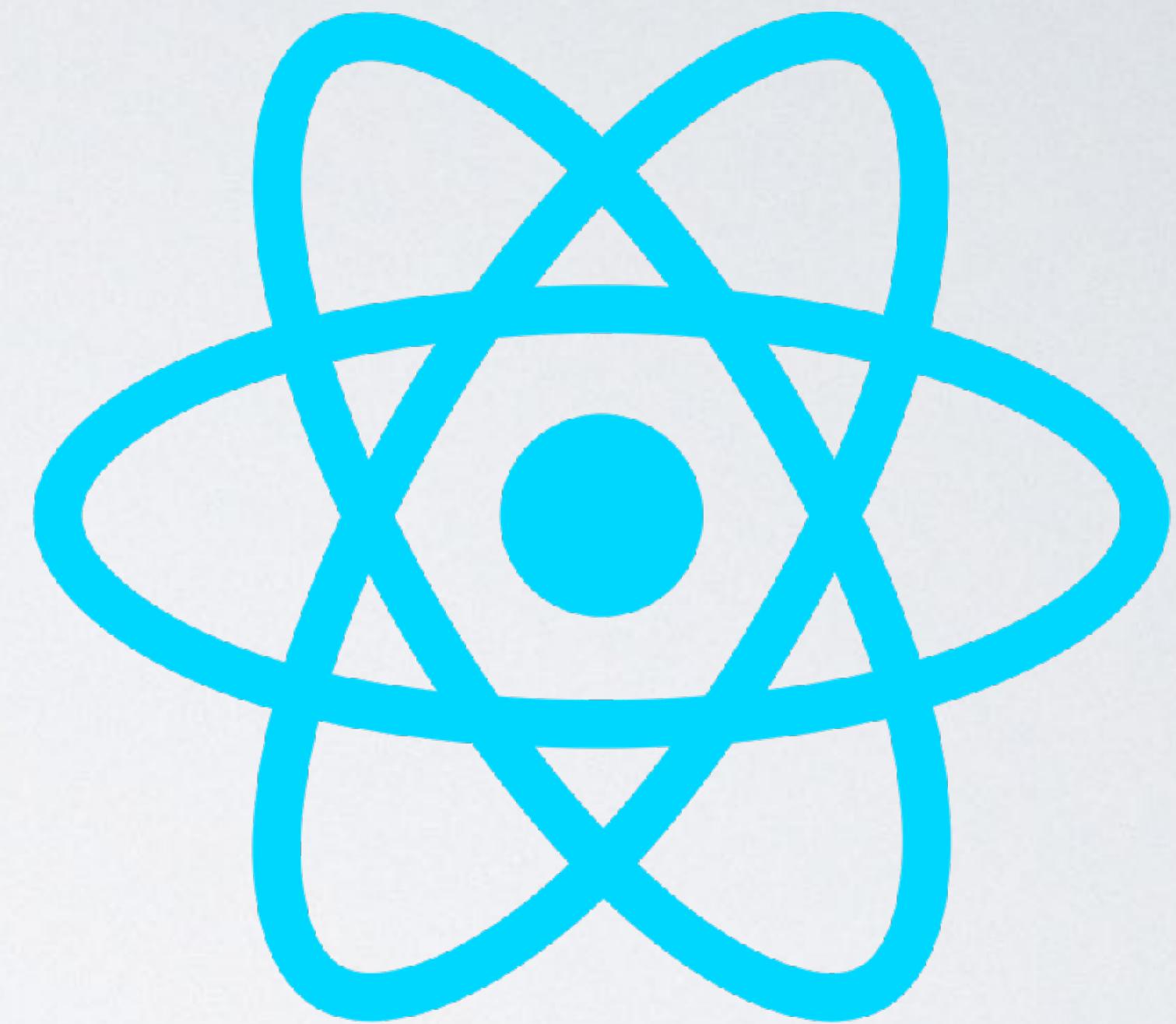
| and it would be a dream to no longer have to worry about the
| build process at all.

That's definitely where we'd like to be. At the moment, we're
heavily reliant on third parties (webpack et al), and it's tricky as
internally at Google we have an entirely different infrastructure for
build, so we're grateful for this kind of feedback as it lets us know
where we're falling short.

Either way



... have fun with
Angular ...



... or have a look at
React!

Fragen?

<https://github.com/jbandi/sbb-dev-day-2017>



JavaScript / Angular / React
Schulungen & Coachings,
Project-Setup & Proof-of-Concept:
<http://ivorycode.com/#schulung>
jonas.band@ivorycode.com