Front-End Entwicklung mit React

Mail: jonas.bandi@ivorycode.com

Twitter: @jbandi

Client Side Routing

Client Side Routing in a SPA

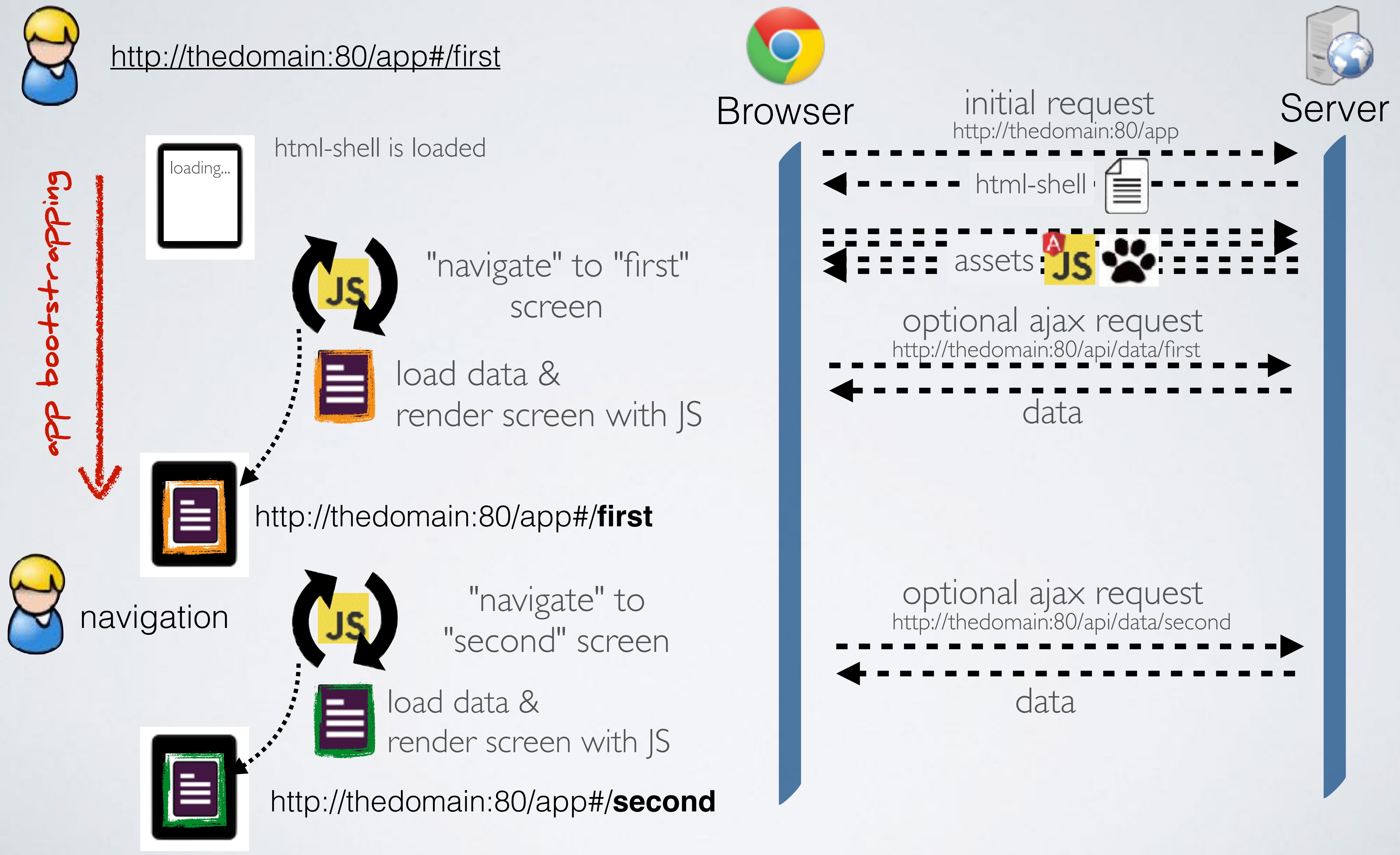
The traditional web is built on the concept of linked documents (URL, bookmarking, back-button …)

In a SPA the document is just the shell for an application. When you navigate away from the document, the application is "stopped".

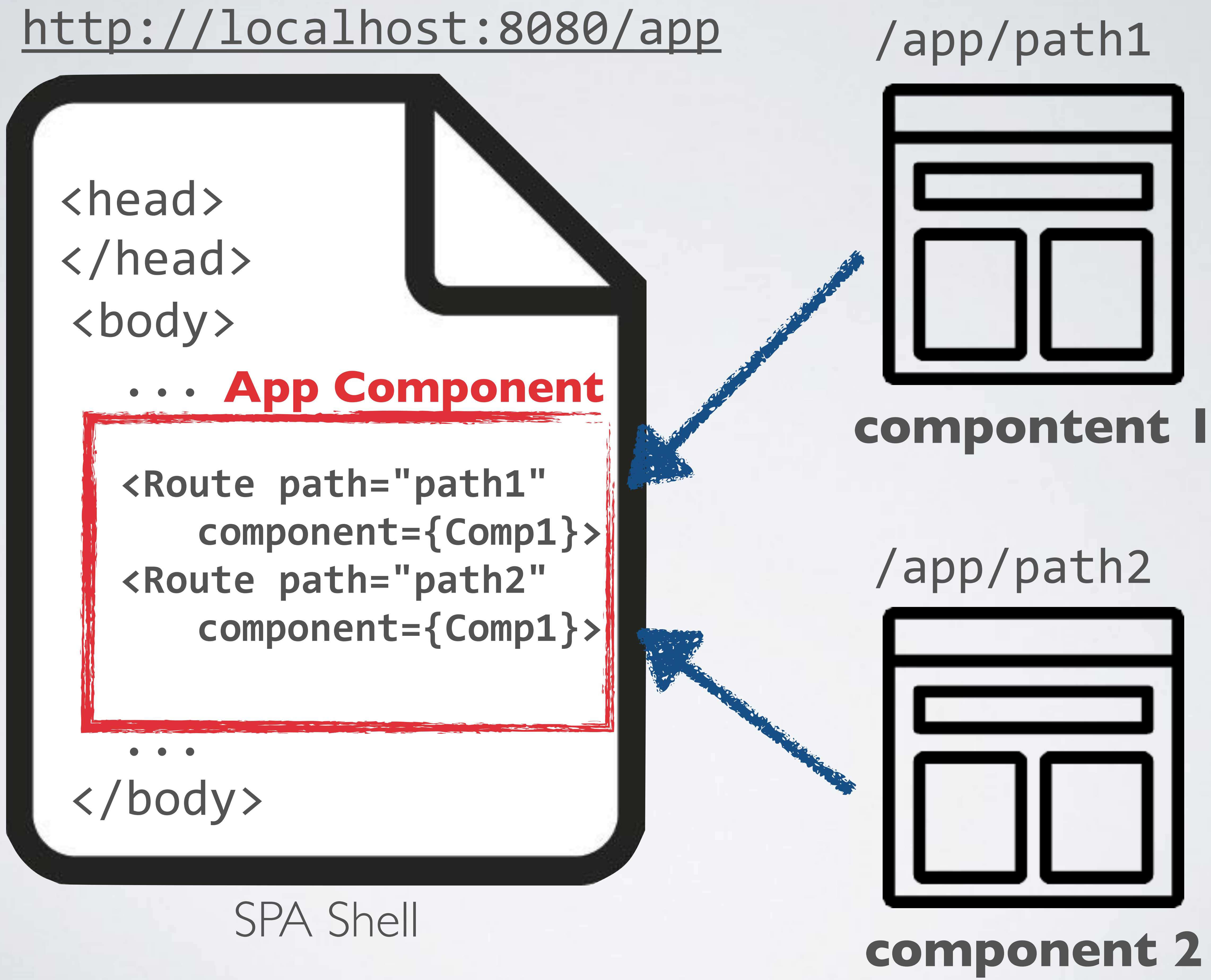
As a consequence a SPA should "emulate" the traditional user-experience on the web:

- navigate via urls, links & back-button
- bookmarks and deep links

Client-Side Routing in a SPA



http://thedomain:80/app#/first
loading...
html-shell is loaded
app bootstrapping
"navigate" to "first" screen
load data & render screen with JS
http://thedomain:80/app#/first
navigation
"navigate" to "second" screen
load data & render screen with JS
http://thedomain:80/app#/second
Browser
Server
initial request
http://thedomain:80/app
html-shell
assets
optional ajax request
http://thedomain:80/api/data/first
data
optional ajax request
http://thedomain:80/api/data/second
data



```
<head>
</head>
 <body>
    ... 
   <Route path="path1"
       component={Comp1}>
   <Route path="path2"
       component={Comp1}>

    ...
 </body>
```

React Router

React Router is a collection of navigational components.

With React Router routing is dynamic, it takes place as the app is rendering.
Routing is not configured statically during initialization.
This allows a very neat integration with the React component model.

```
npm install react-router-dom
```

https://reacttraining.com/react-router/

React Router

Routes are represented as components.

```
import {BrowserRouter as Router,
      Route, Link, Switch} from 'react-router-dom';

<Router>
  <div>
    <ul>
       <li><Link to="/">Home</Link></li>
       <li><Link to="/about">About</Link></li>
    </ul>
    <Switch>
       <Route exact path="/" component={Home}/>
       <Route exact path="/about" component={About}/>
    </Switch>
  </div>
</Router>
```

A `Router` is a container component:

A `route` renders a component if the route matches the path:

The `Router` provides some properties to all contained components: `match, location, history`

https://reacttraining.com/react-router/

Backend Access

axios

axios is a promise based HTTP client library for the browser and node.js

```
axios.get(API_URL)
   .then((response) => console.log(response))
   .catch((error) => console.log(error));
```

```
axios.post(API_URL, payload)
   .then((response) => console.log(response))
   .catch((error) => console.log(error));
```

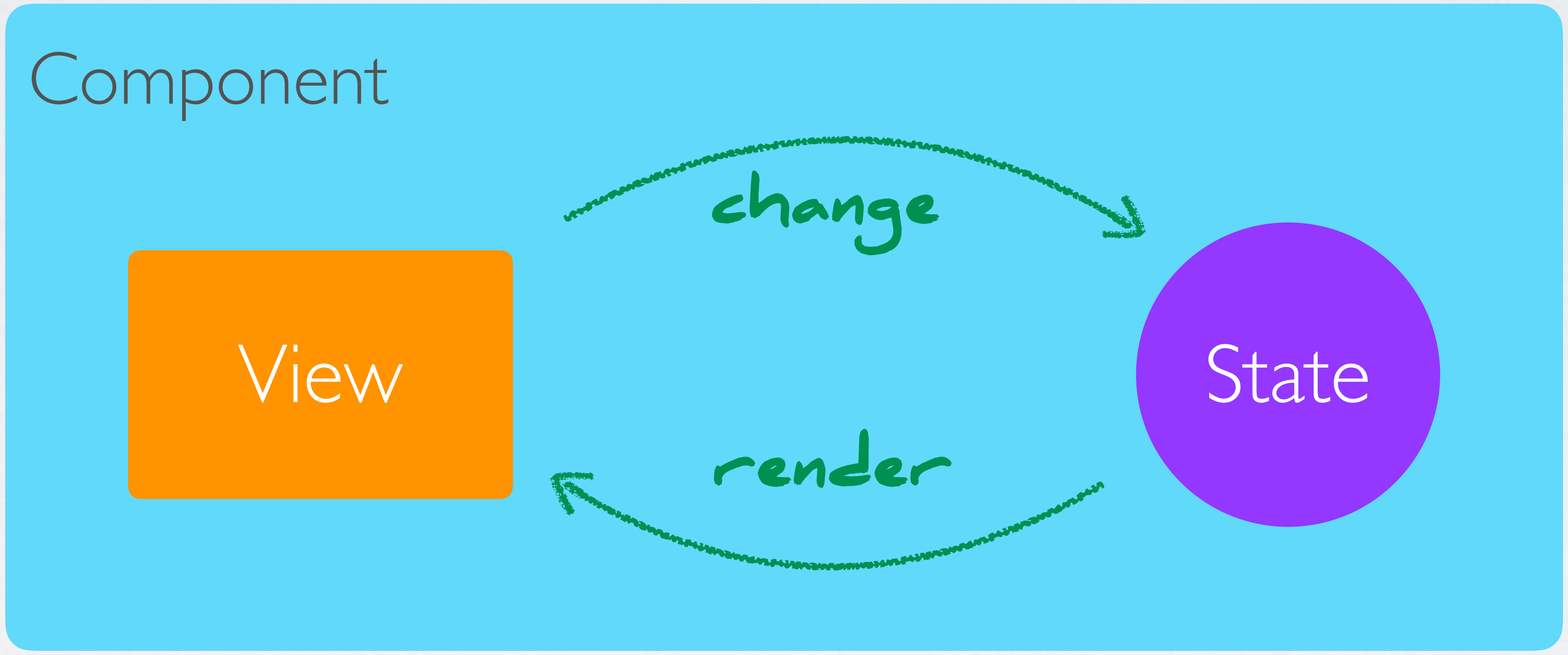
```
axios.delete(`${API_URL}/${id}`)
   .then((response) => console.log(response))
   .catch((error) => console.log(error));
```

https://github.com/axios/axios

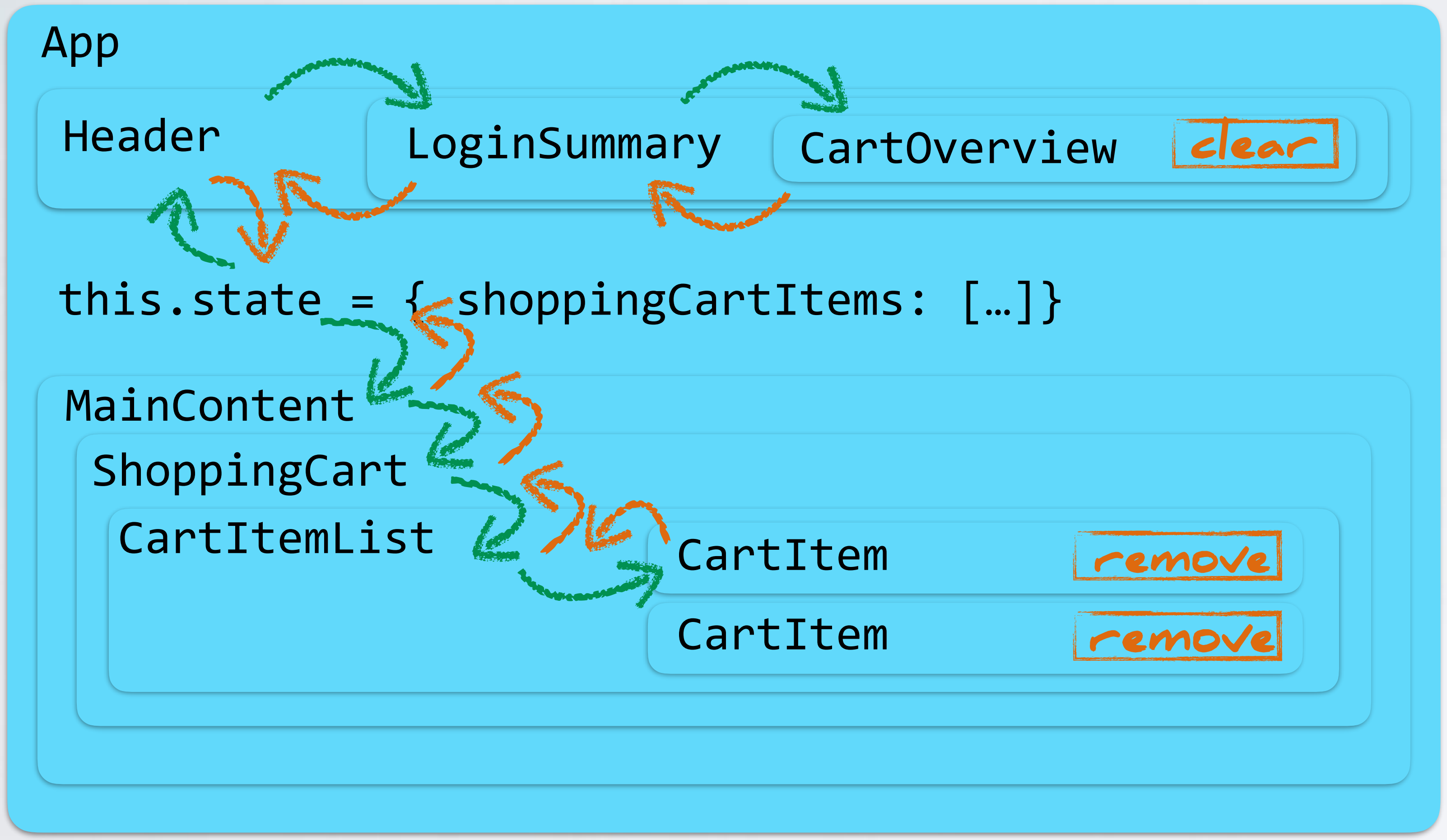
State Management with Redux

A single component

Managing state in a component is simple:



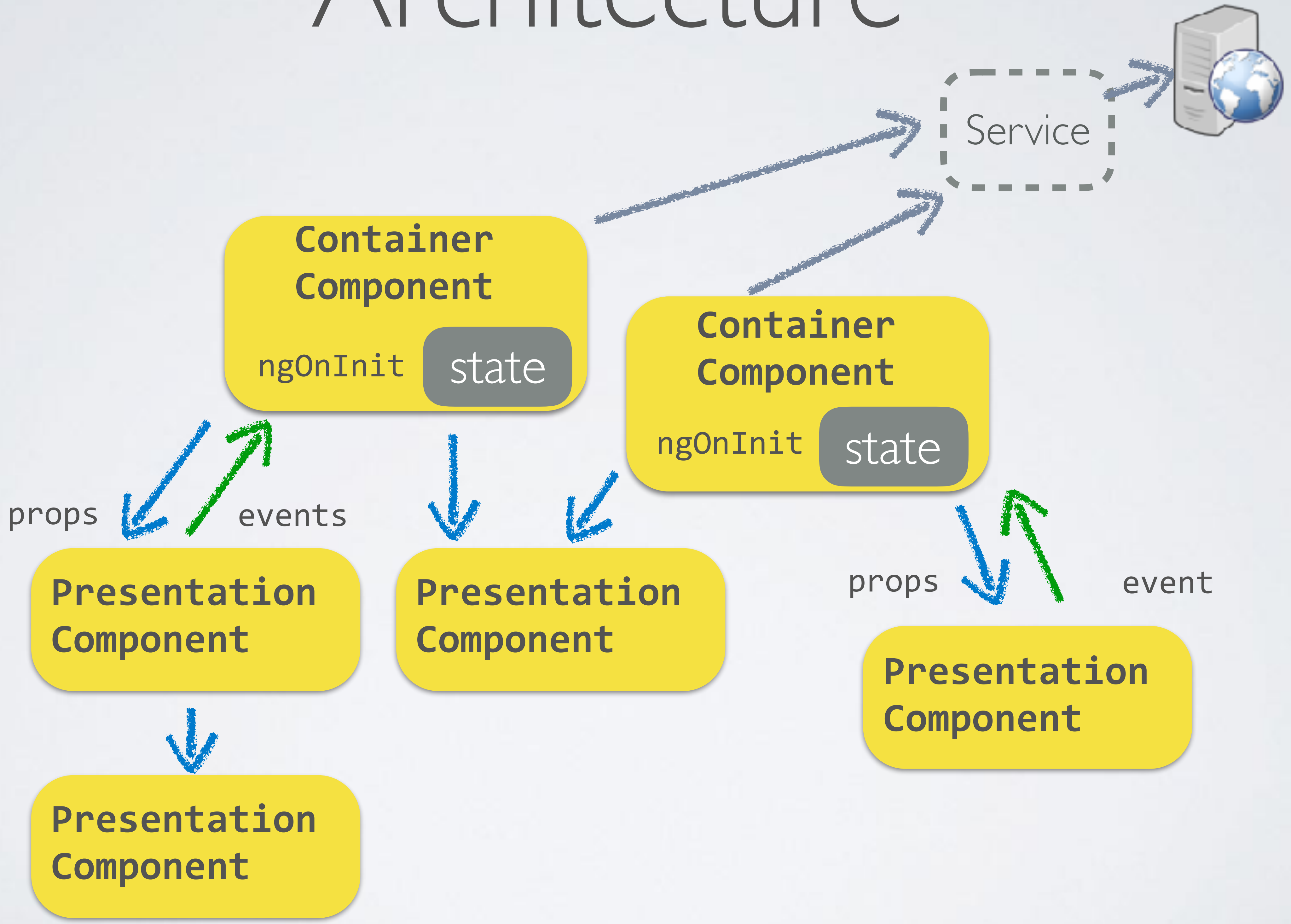
State Management



Challenges when managing state in a component tree:

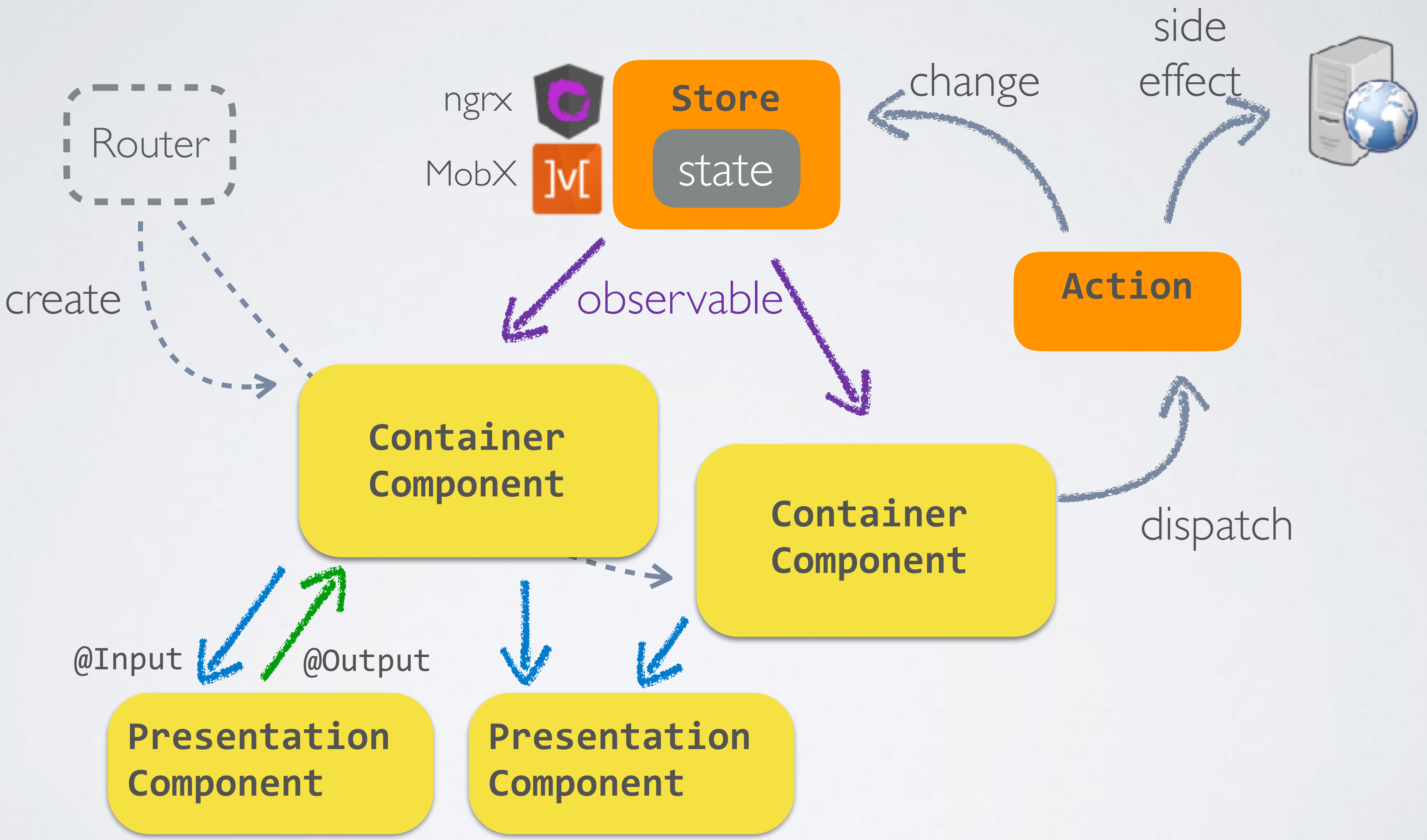
- Multiple components may depend on the same piece of state.
- Different components may need to mutate the same piece of state.

State Management: Component Architecture



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

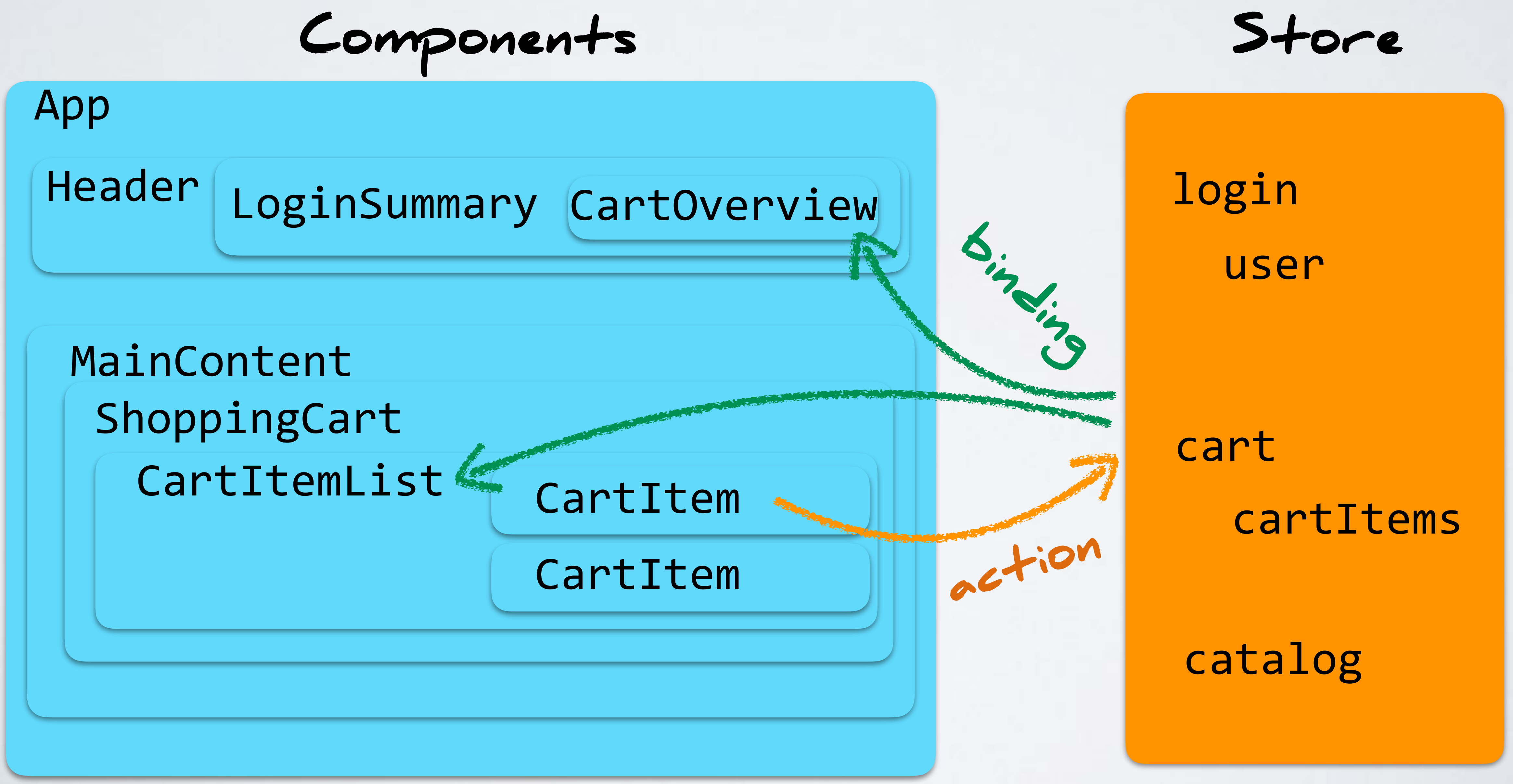
State Management: State Container



ngrx: https://github.com/ngrx/
MobX: https://mobx.js.org/

Managing State with a State Container

State can be managed outside the components.
Components can be bound to state.



Do I need a state container?

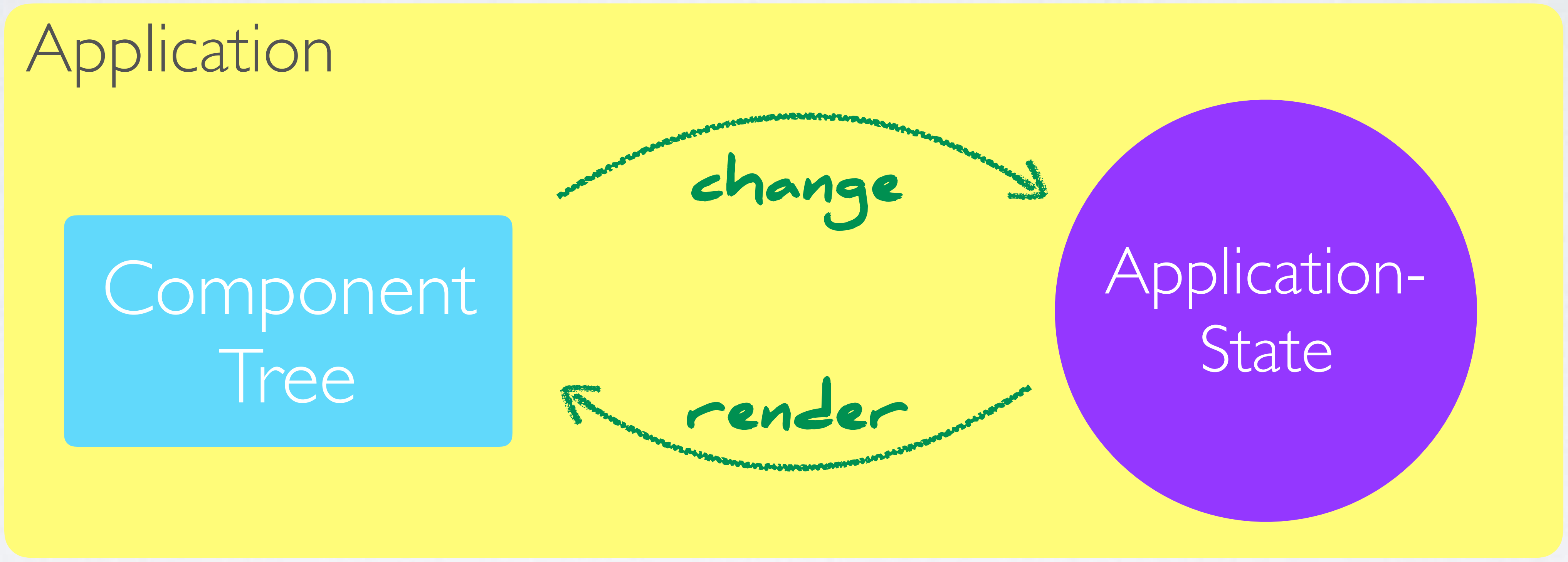
Holding state in React components works well when the state is hierarchical and more or less matches the component structure.

If distant distant parts of the app want to have access to the same state, you'll end up with a bunch of very large components at the top of the component tree that pass a myriad of props down through some intermediate components that don't use them, just to reach a few leaf components that actually care about that data.
In this scenario you can extract state & state management into an external container and connect leaf components directly.

http://stackoverflow.com/questions/36631761/when-should-i-add-redux-to-a-react-app
https://medium.com/@dan_abramov/you-might-not-need-redux-be46360cf367

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!

Step 4: Change the Global State in a "functional style"

Go to: `11-state-changes`

Inspect the example.

Run the example:

```
npm install
npm run:watch
```

Implement the functionality to remove a todo.

Solution:
`xx-backup/todos-step-04`

Array Reduce

```
var a = [1,2,3,4,5];

var result = a.reduce(
  // reducer function
  (acc, val) => {
    const sum = acc.sum + val;
    const count = acc.count + 1;
    const avg = sum/count;
    return {sum, count, avg};
  },
  // state object
  {sum:0, count:0, avg:0}
);

console.log('Statistics:', result);
```

The reduceer function is a pure function.

Step 5: Use Redux to manage the Global State

Go to: `12-state-redux`

Inspect the example.

Run the example:

```
npm install
npm run:watch
```

Implement the functionality to remove a todo.

Solution:
`xx-backup/todos-step-05`

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 alterative implementation of the same concept.
(https://github.com/angular-redux/ng-redux and https://github.com/ngrx)

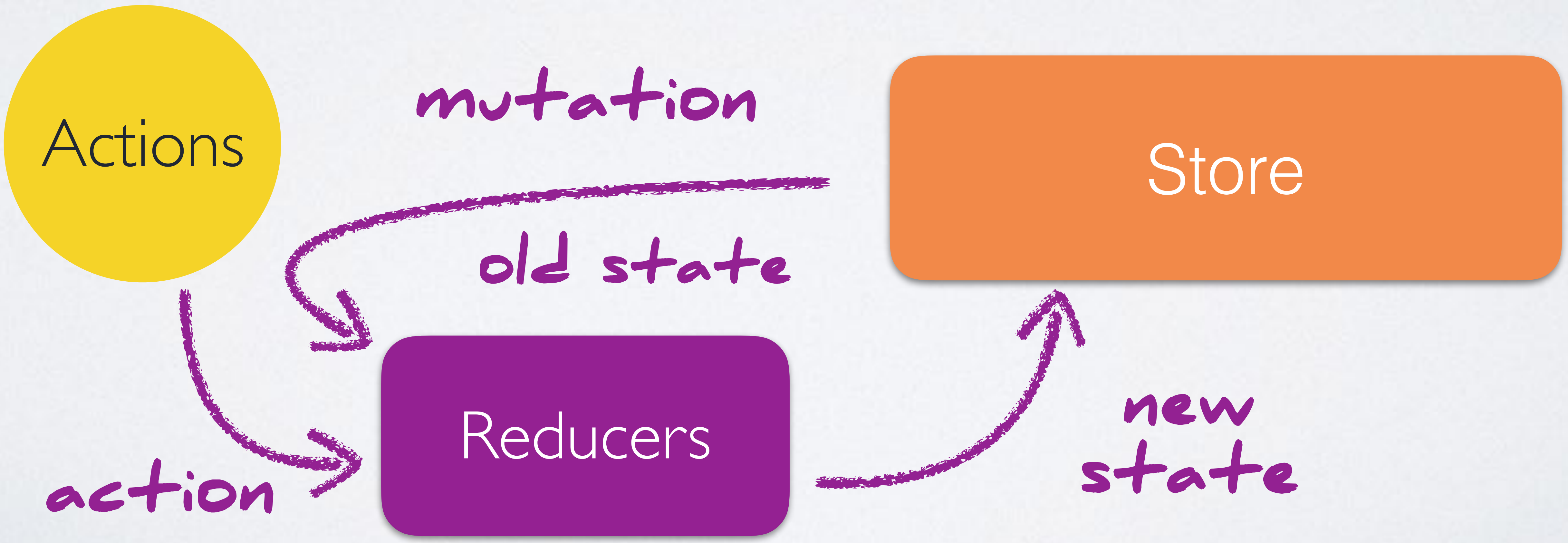
http://redux.js.org/

State Reduce

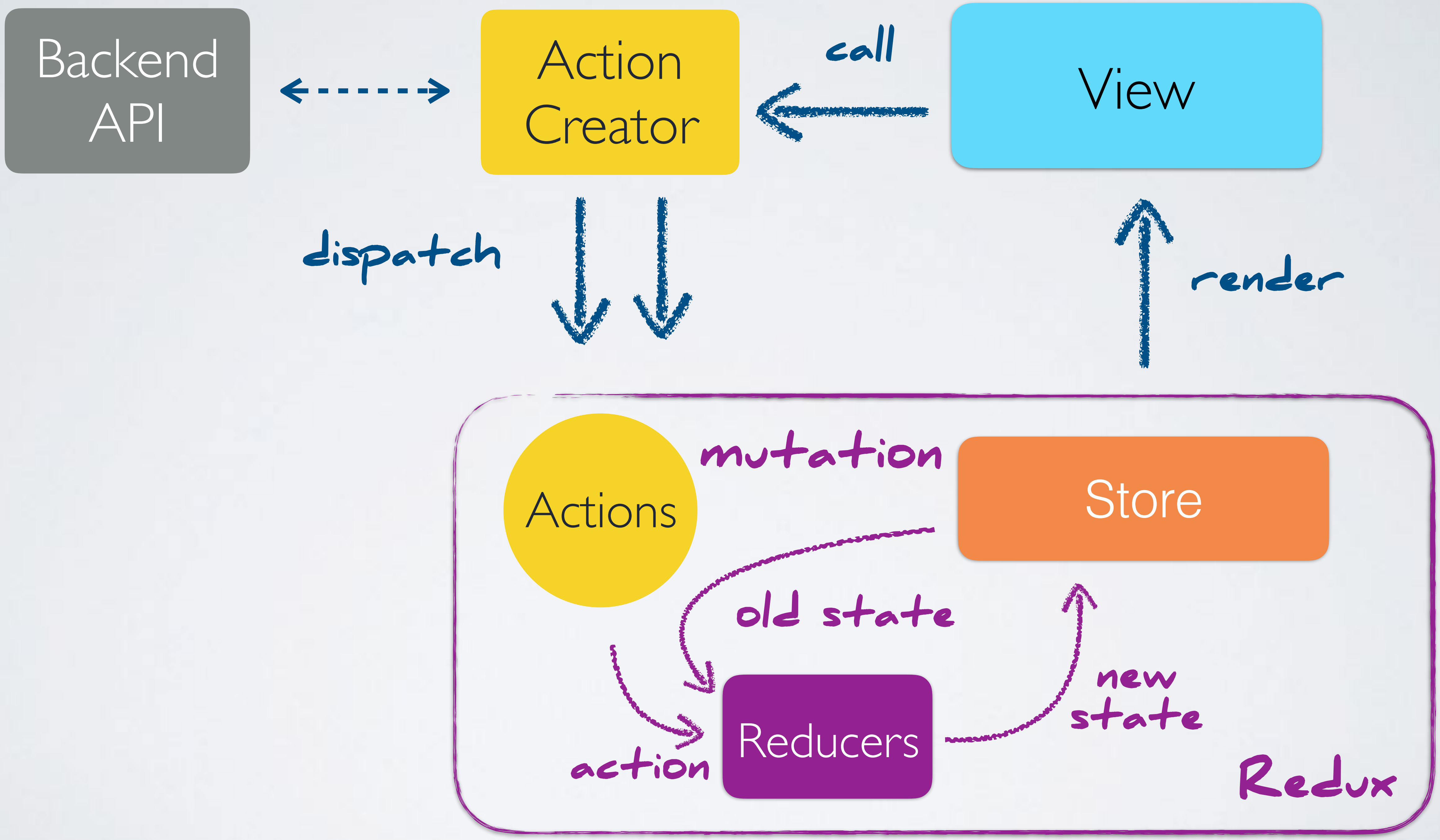
Redux implements store mutations in a "functional way".

(old state , action) => new state

aka: "reducer function"



Redux Architecture



Redux Thunk

Redux Thunk is a middleware for Redux that enables action creators can return a function.

With thunks you can:

- dispatch multiple actions
- dispatch actions asynchronously at a later point in time
- dispatch actions based on certain conditions

Alternative: Redux Sagas

https://github.com/gaearon/redux-thunk

MobX

MobX is a state management library.

https://mobx.js.org/

https://www.robinwieruch.de/redux-mobx-confusion/

React works great with TypeScript!

```
npm i -g create-react-app
create-react-app awesome-app --scripts-version=react-scripts-ts
cd awesome-app
npm start
```

Alternative: Flow
https://flow.org/

Viel Spass mit React

JavaScript / Angular / React
Schulungen & Coachings,
Project-Setup & Proof-of-Concept:

http://ivorycode.com/#schulung
jonas.bandi@ivorycode.com