Schedule

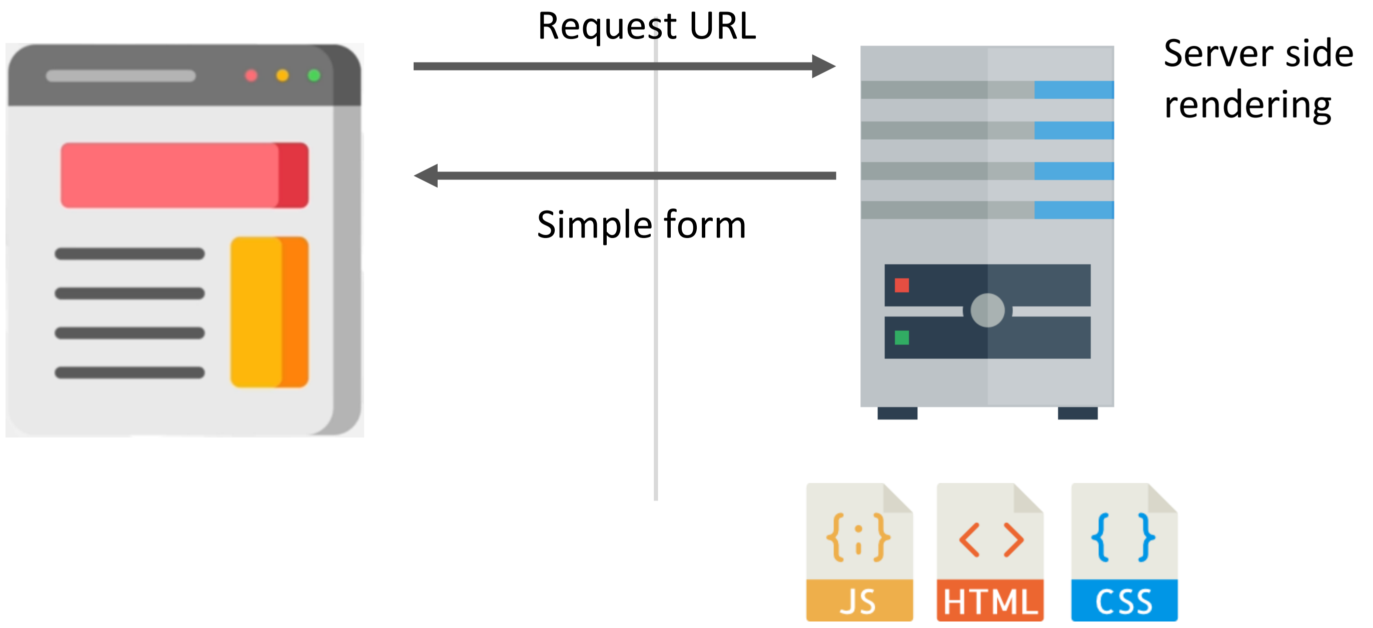
ReactJS:

- The birth of React
- Getting started with React
- React key concepts
- How to become a good React developer

The birth of React

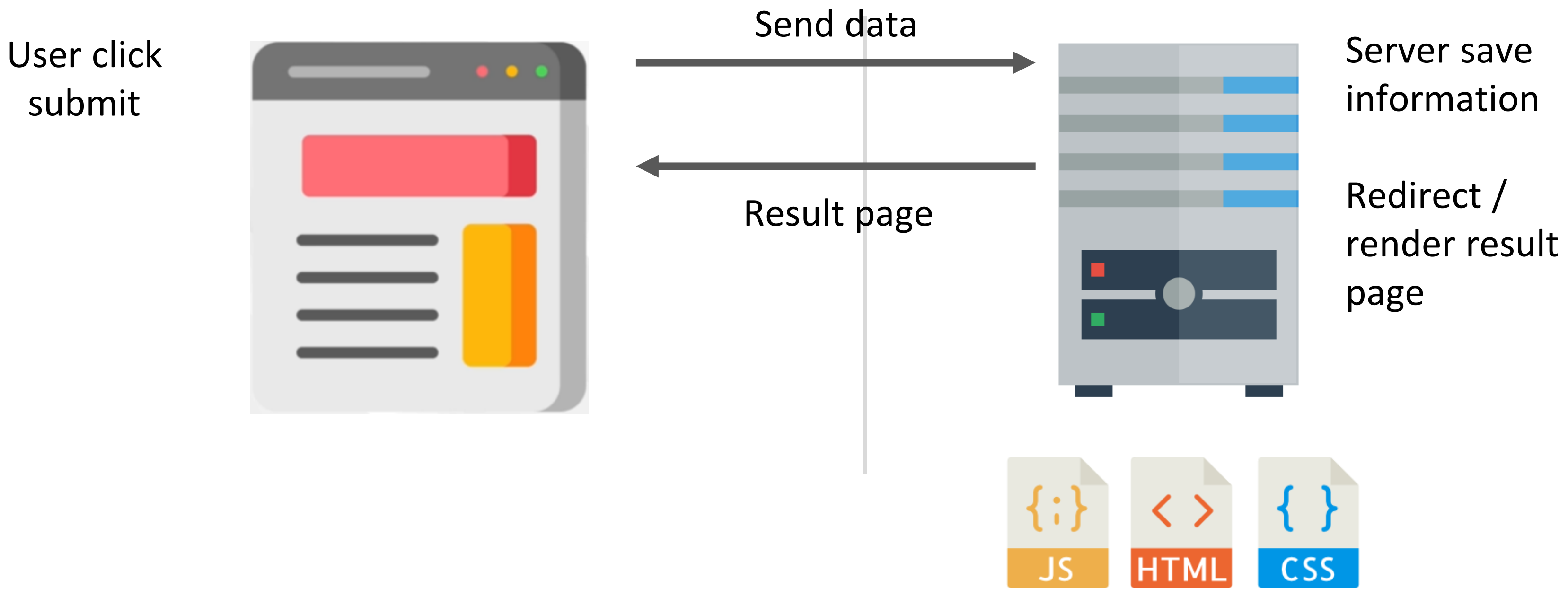
Traditionally...

- 90s & early 2000s



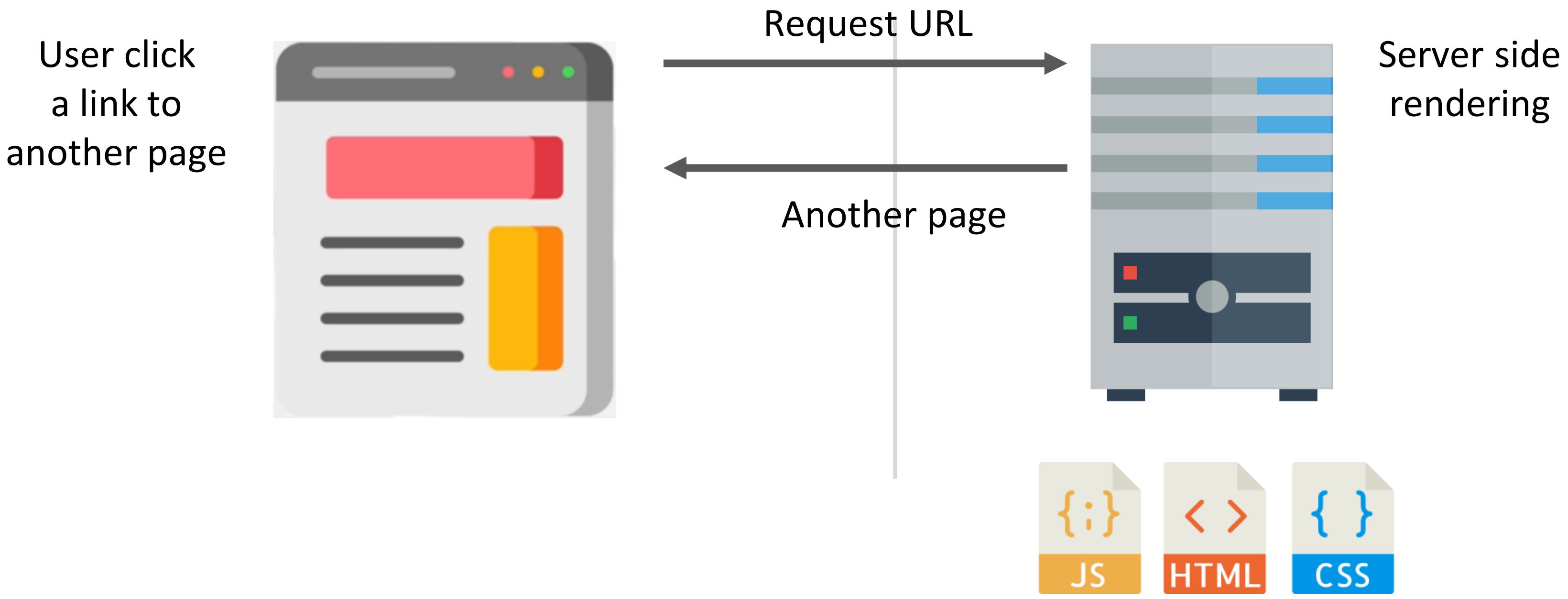
Traditionally…

- 90s & early 2000s



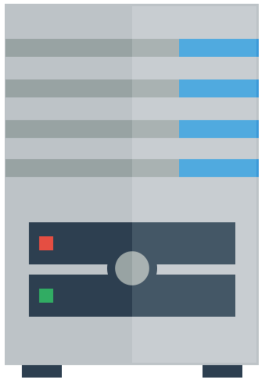
Traditionally…

- 90s & early 2000s



Traditionally…

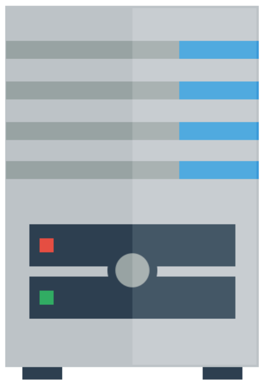
- 90s & early 2000s



JS – manipulate with the DOM ← that's all about JS

- User interaction: e.g. toggle show/ hide elements
- AJAX / fetch
    - HTML → add/ replace into DOM tree
    - JSON + template engine: handlebars

Compatibility problem

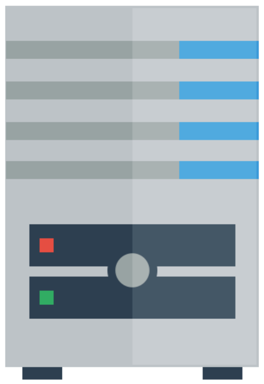


More & more JS?

- Different teams
- Different engines / DOM API

→ sometimes work differently from each other

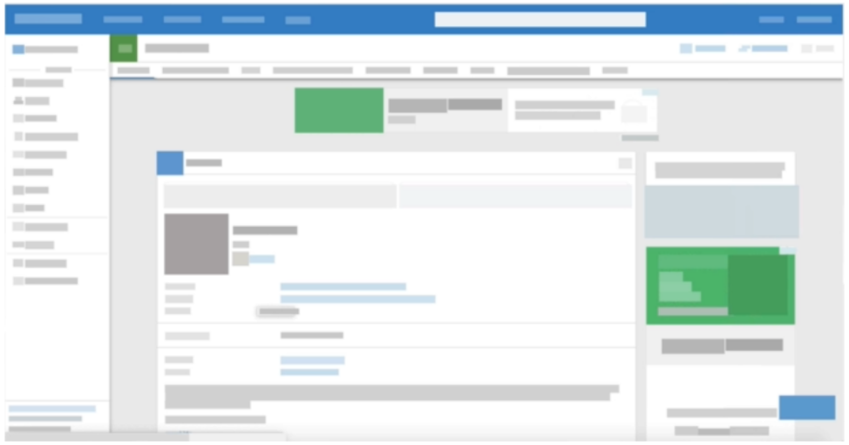
JQuery – easily DOM interact across browsers



DOM API → JQuery API

- Same function → works consistently across browsers

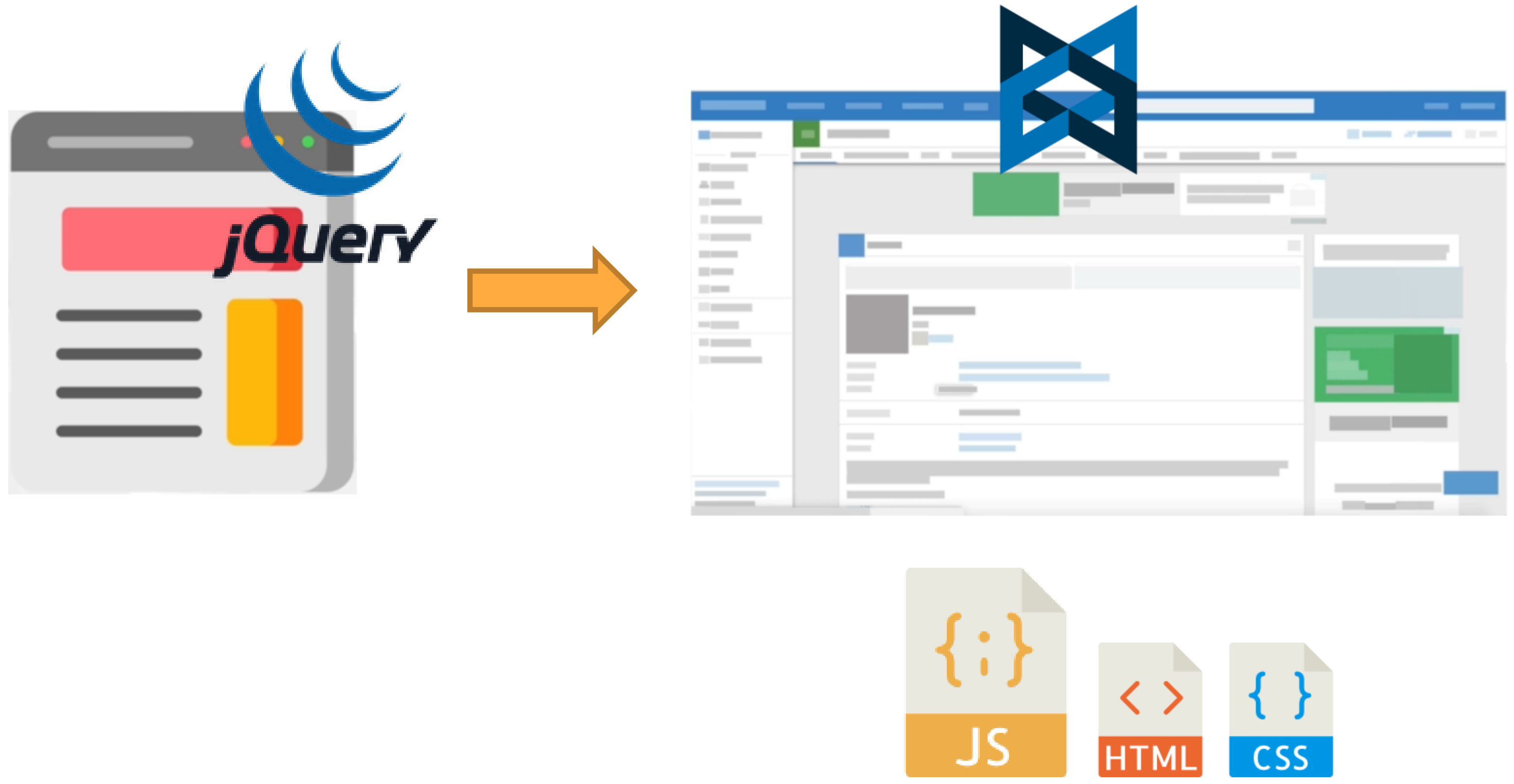
Bigger & bigger applications...



More & more JS

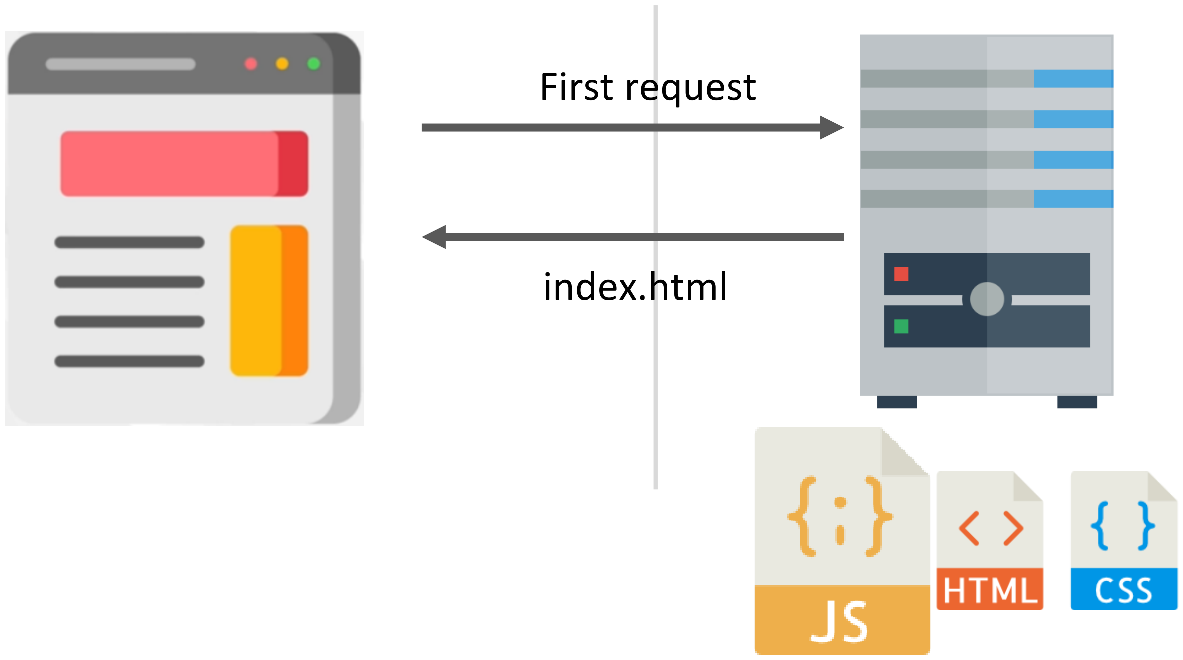
- e.g: Facebook: pages, feeds, chatting with friends...

Backbone.js – organize JS files



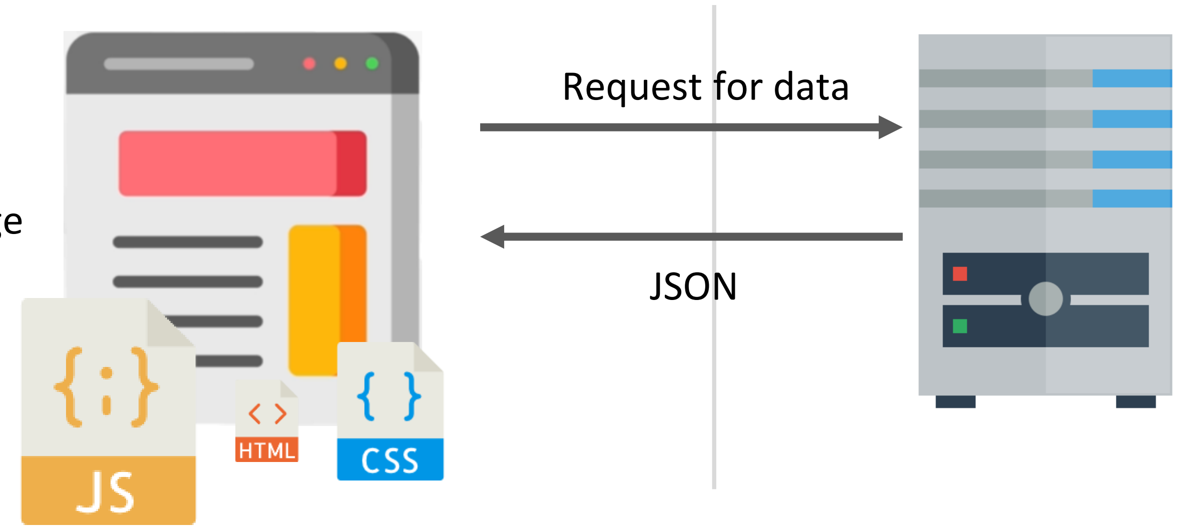
More & more JS

The birth of SPA



The birth of SPA

User click
a link to
another page



JS updates the DOM

AngularJS

- 2010 AngularJS by Google
- MVC pattern – easier to work as teams get larger & larger → **popular**

Problem

- Thing started getting more & more complicated

→ Harder & harder to debug the code &
understand how parts affect the others

Facebook dev group

- Experience this problem (esp. Facebook ads app)

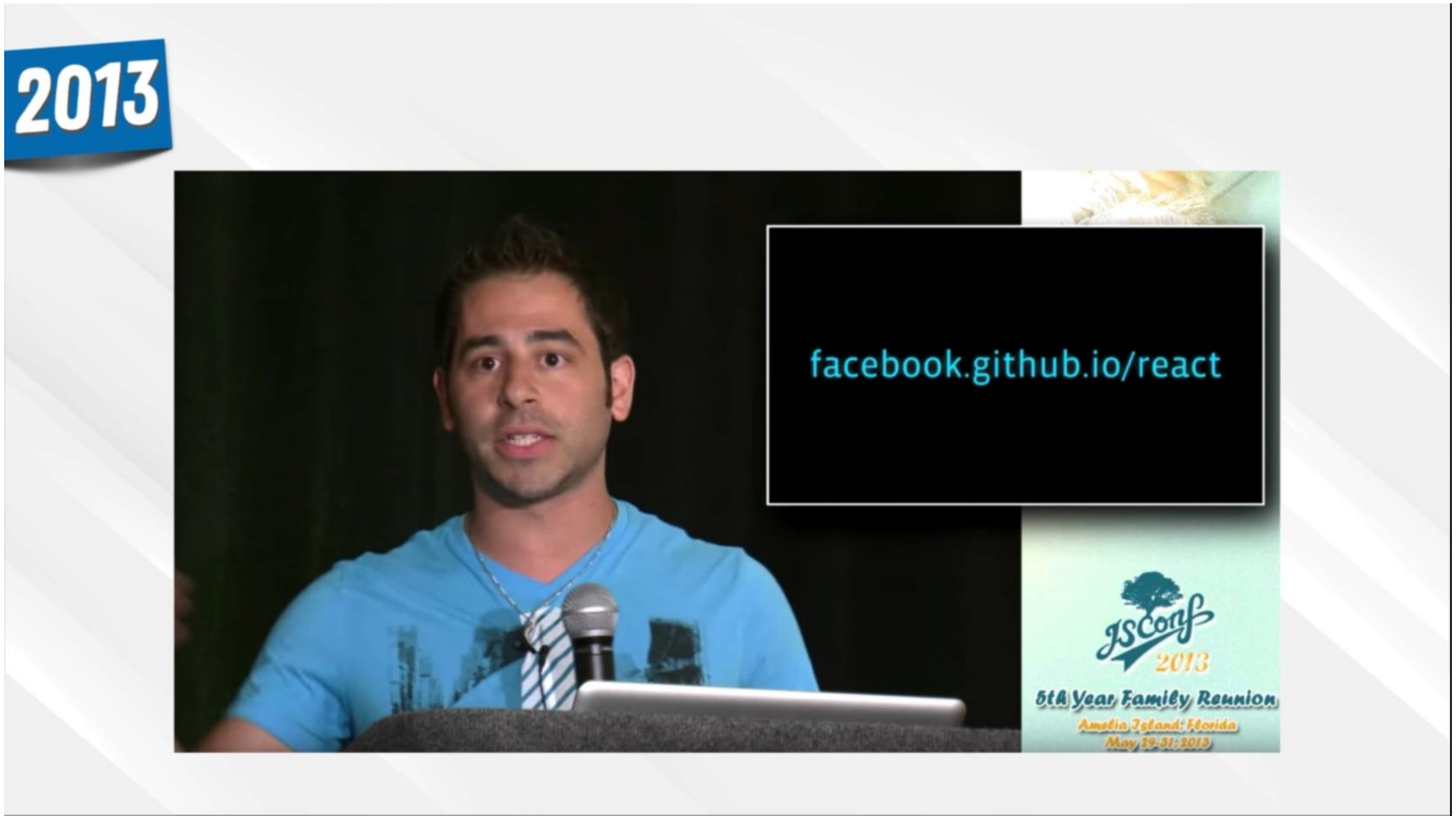
→ Need a good architecture

- How to organize the code
- How to manipulate data
- **How that data flows in our program**

→ **Come up with the solution ← works really well**

The birth of ReactJS

- Release to community in JS conference
- 2014 – Google rewrite AngularJS → **Angular**



Getting started with React

update: What is React?

- React is a **JavaScript** library created by **Facebook**
- React is a **User Interface** (UI) library
- React is a tool for building **UI components**

```
import React from 'react';
import ReactDOM from 'react-dom';

class Test extends React.Component {
  render() {
    return <h1>Hello World!</h1>;
  }
}

ReactDOM.render(<Test />, document.querySelector('#root'));
```

Setting react

- Install create-react-app

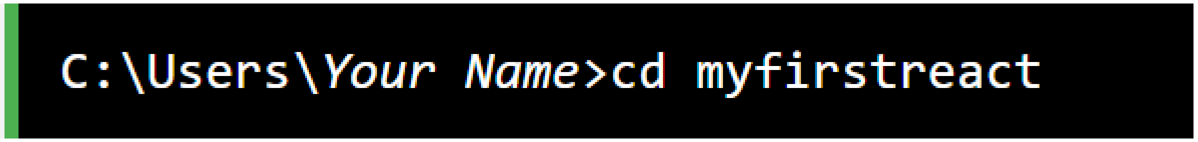
```
C:\Users\Your Name>npm install -g create-react-app
```

- To create a React application, eg. ***myfirstreact***.

```
C:\Users\Your Name>npx create-react-app myfirstreact
```

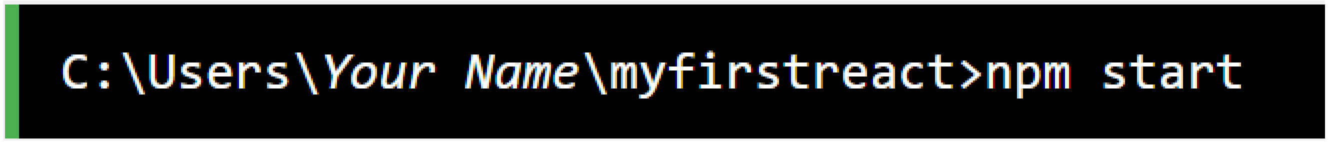
Running react

- Move to the ***myfirstreact*** directory



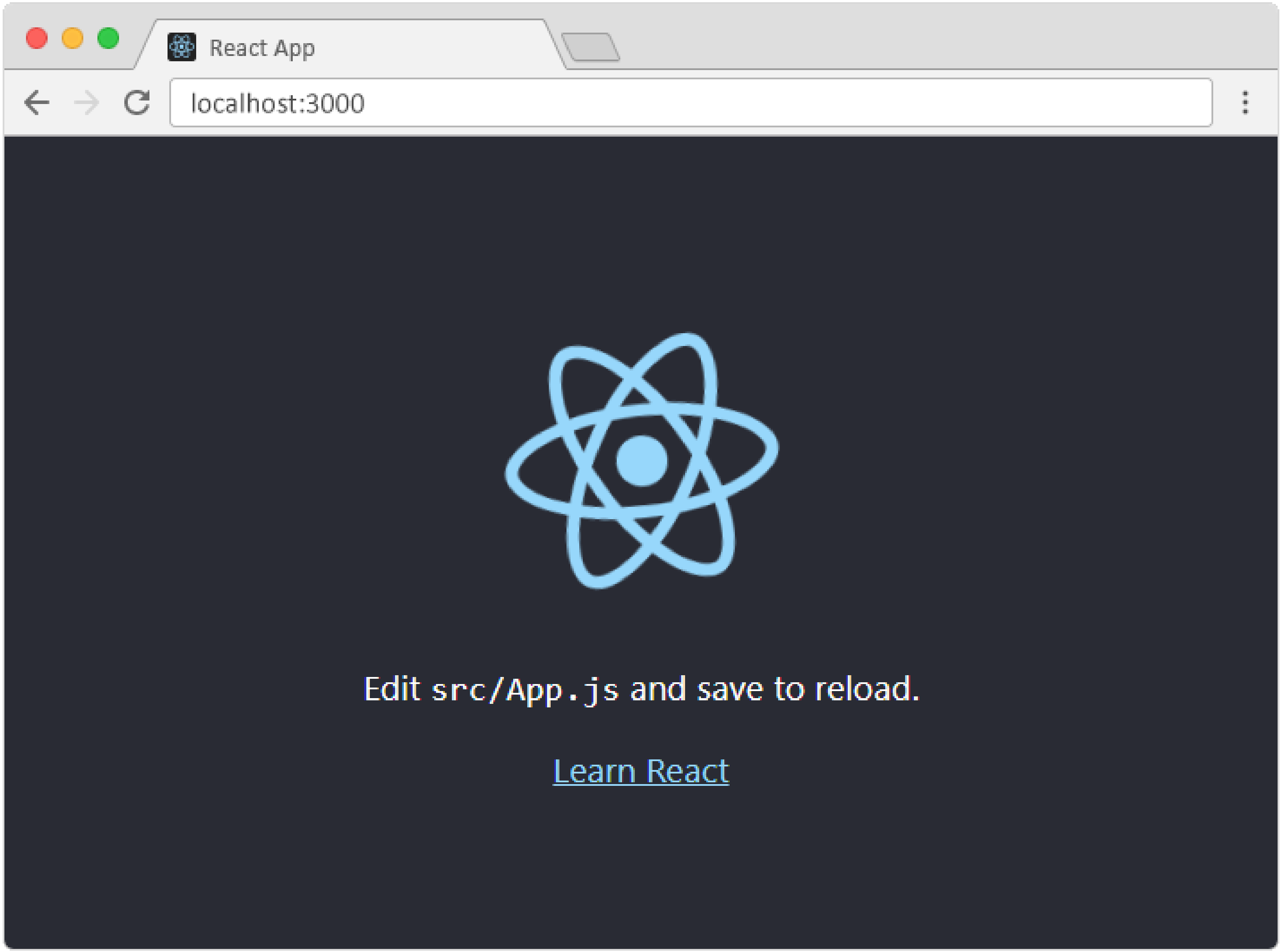
```
C:\Users\Your Name>cd myfirstreact
```

- Run application



```
C:\Users\Your Name\myfirstreact>npm start
```

Running react



Edit `src/App.js` and save to reload.

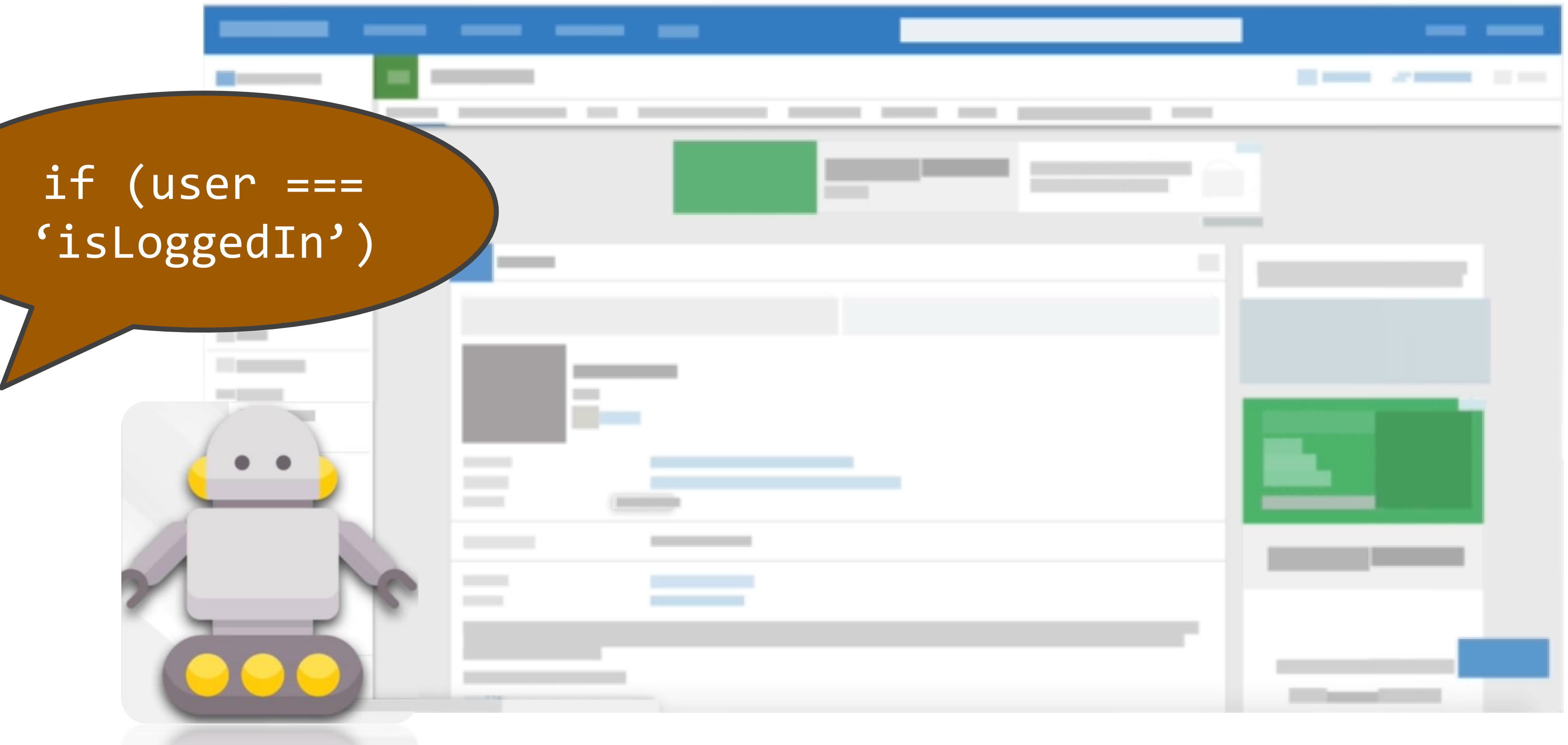
Learn React

React key concepts

1. Don’t touch the DOM. I’ll do it

Lib, framework before:

User events → directly change individual parts of your app

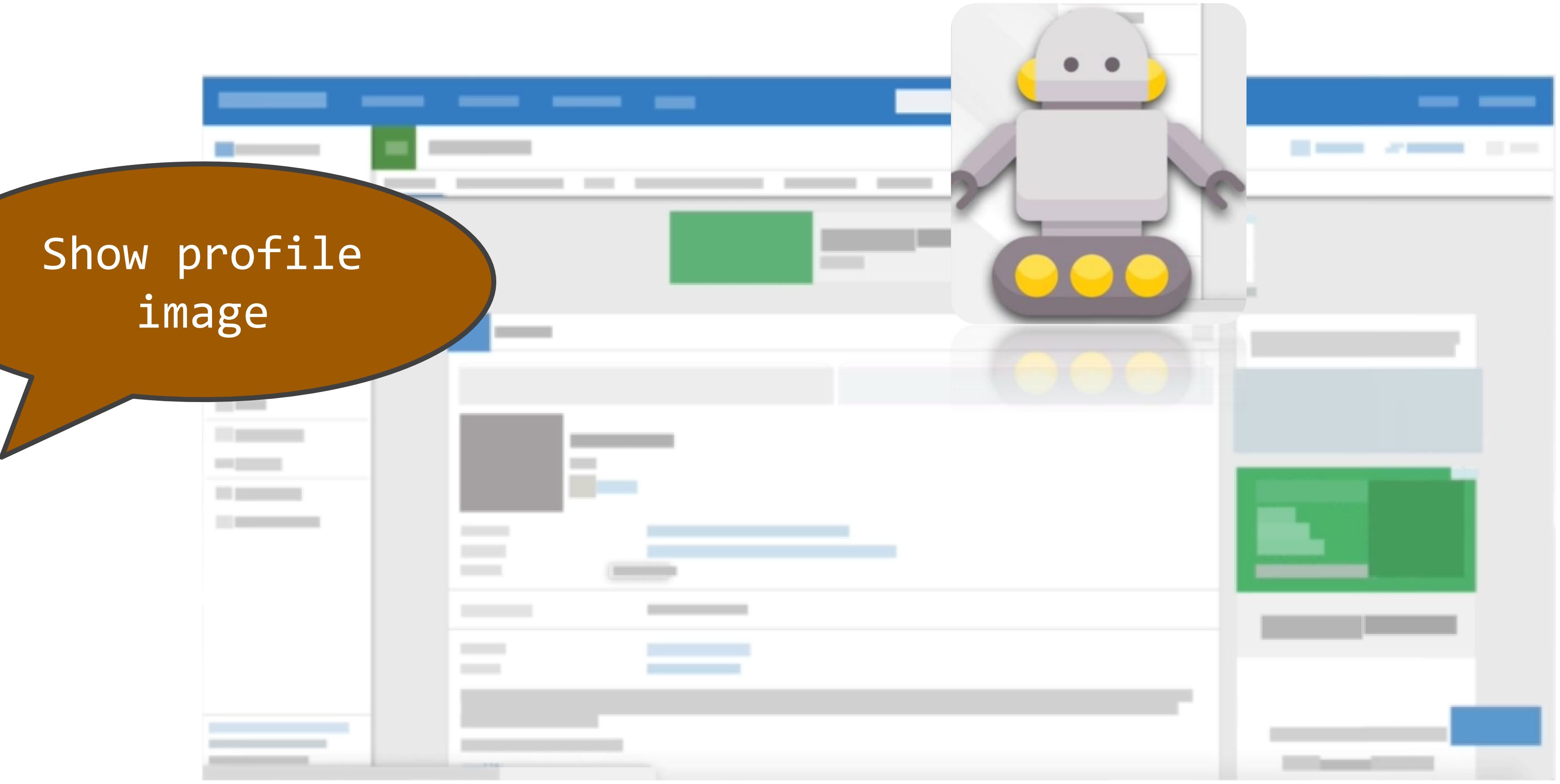


```
if (user ===
‘isLoggedIn’)
```

1. Don’t touch the DOM. I’ll do it

Lib, framework before:

User events → directly change individual parts of your app



1. Don’t touch the DOM. I’ll do it

Lib, framework before:

User events → directly change individual parts of your app

1. Don’t touch the DOM. I’ll do it

Lib, framework before:

User events → directly change individual parts of your app

Problem

- Hard to see relationship between events & all these cases
- Recall: AngularJS with relationships affecting each other & arrows pointing to different things

React virtual DOM

- DOM manipulation = take long time

eg. Changing the DOM

→ repaint changed/ added element → refloated layout

← expensive operation

- React find the best way to change the DOM **automatically** – just declare what your app looks like

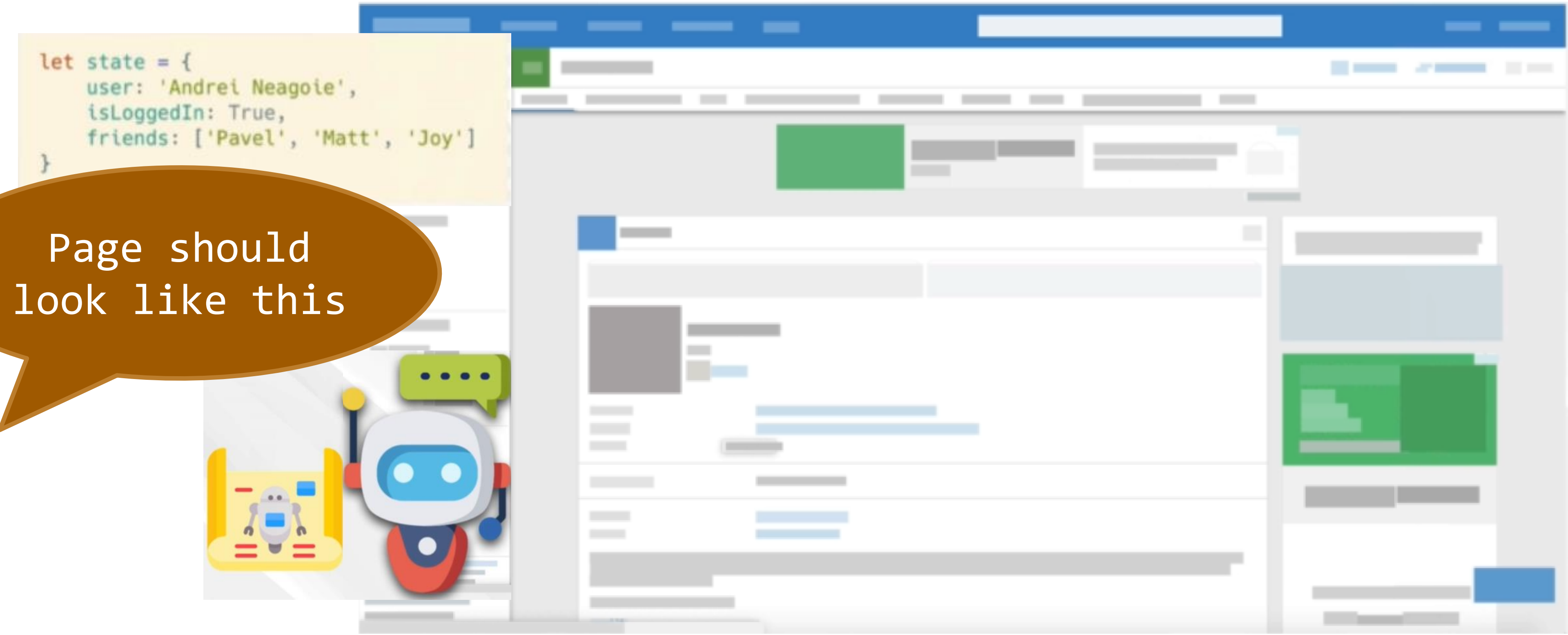
eg. JS object (state) = what I want the app to look

*Hey React, this is the **state** (data) of our app → display it*

React virtual DOM

Hey React, this is the ***new state*** *(data) of our app → make necessary changes to display it*

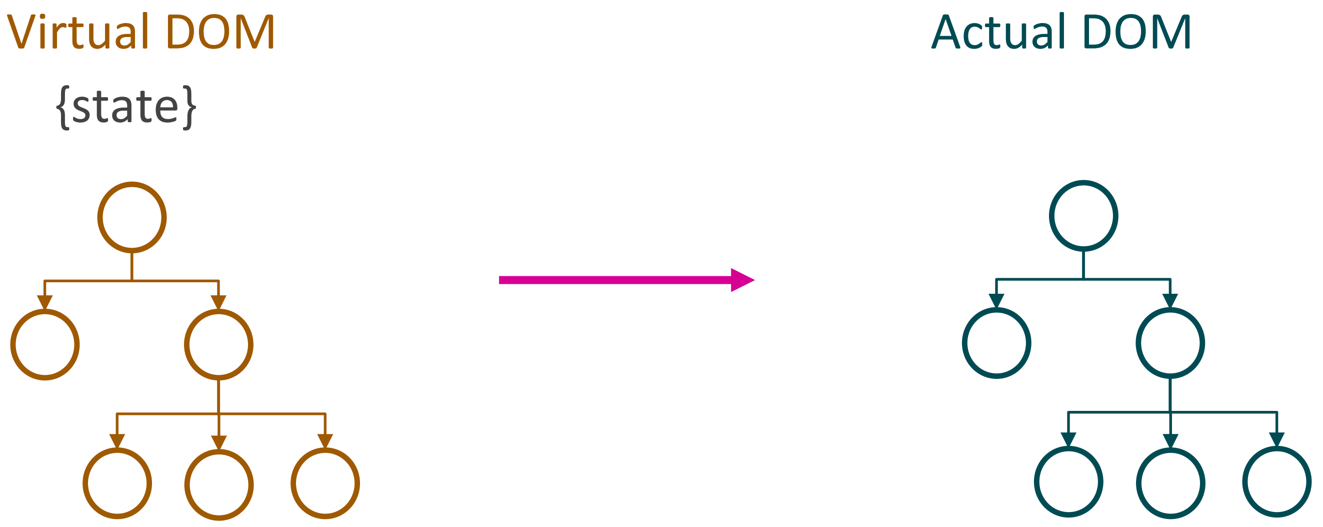
eg. user logged in → React already knows exactly what to update & what to do



```
let state = {
    user: 'Andrei Neagoie',
    isLoggedIn: True,
    friends: ['Pavel', 'Matt', 'Joy']
}
```

React virtual DOM

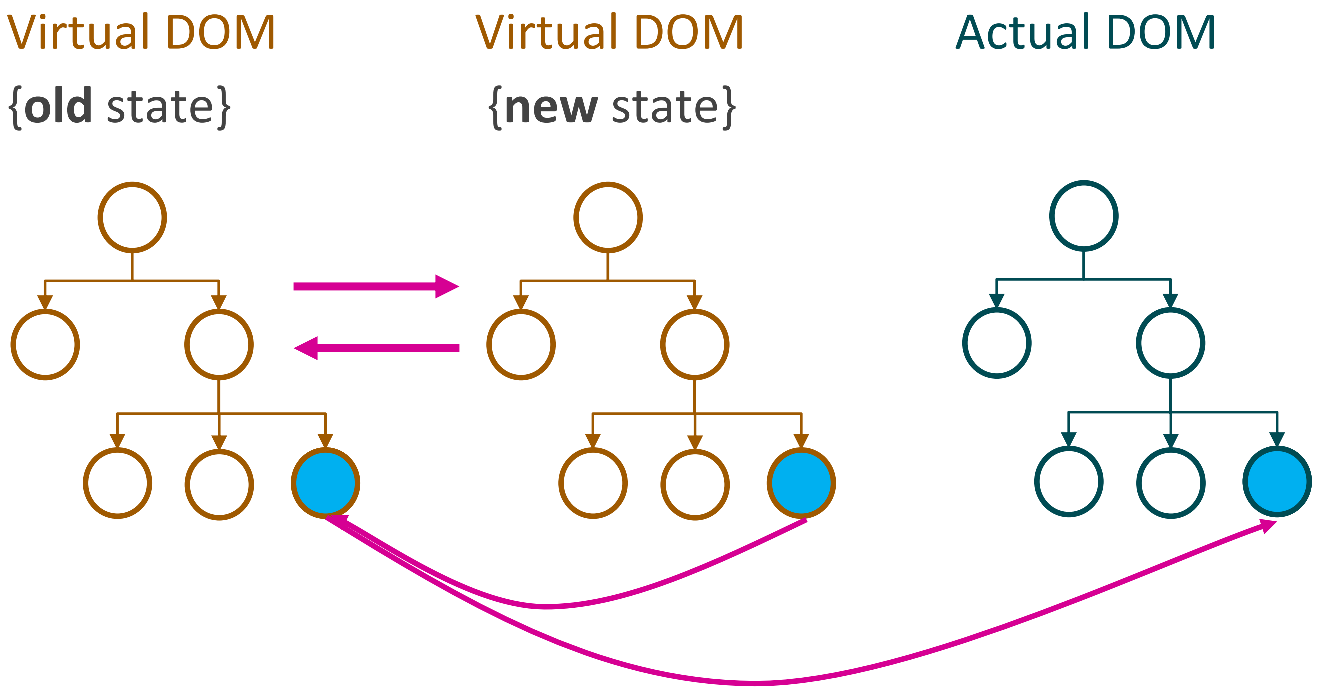
- **State**: one big JS object that describe how our app should look



* **React** = based on whatever the state or data of the app that describes the app → I just “**react**” to it & change everything for you so that you get the display you want.

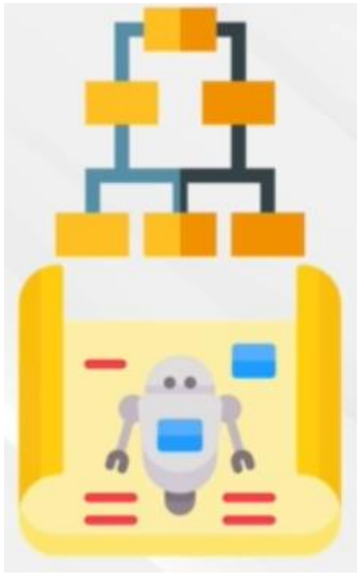
React virtual DOM

- **State**: one big JS object that describe how our app should look



* **React** = based on whatever the state or data of the app that describes the app → I just "**react**" to it & change everything for you so that you get the display you want.

React virtual DOM



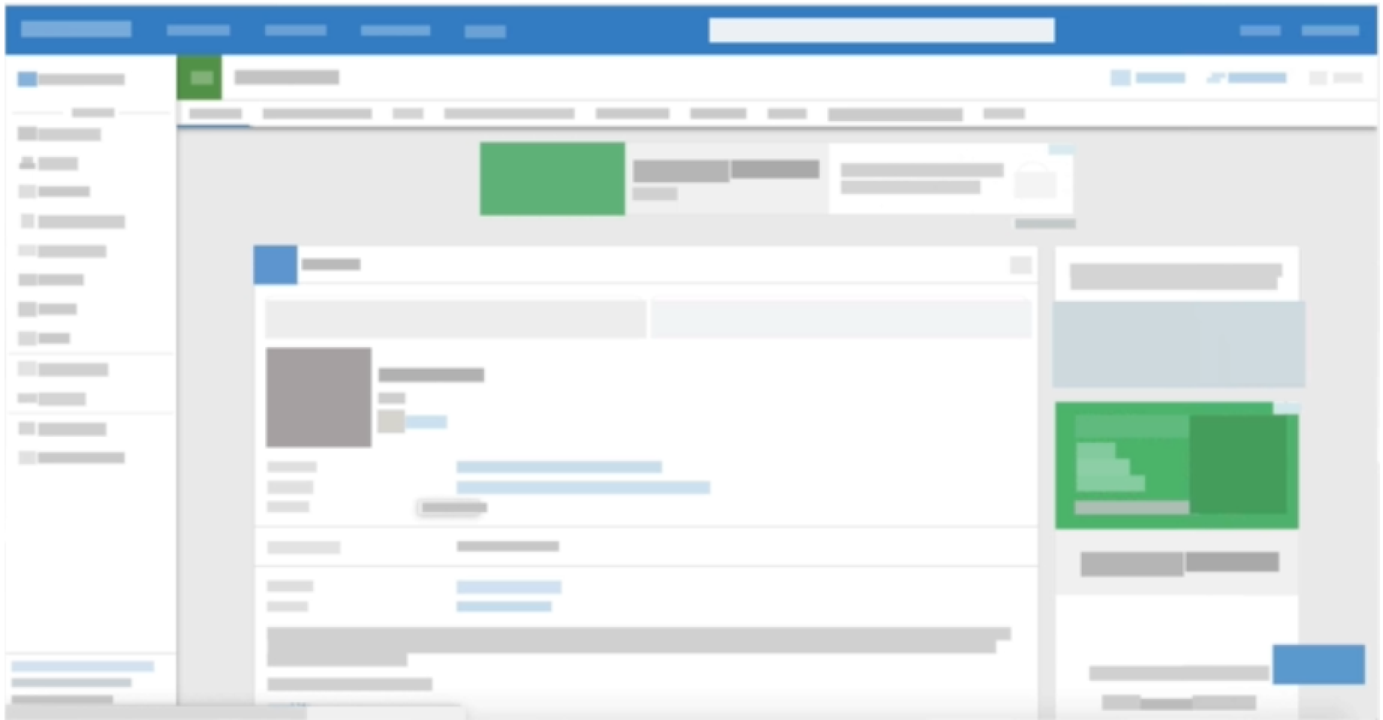
→ Less complexity

→ Better code quality

→ Faster developer times

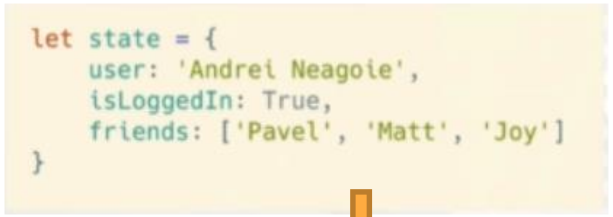
2. Build website like LEGO blocks

- Reusable components
  - e.g. https://material-ui.com/components/buttons/
- Small components – put together → bigger component
- Even move over to different projects

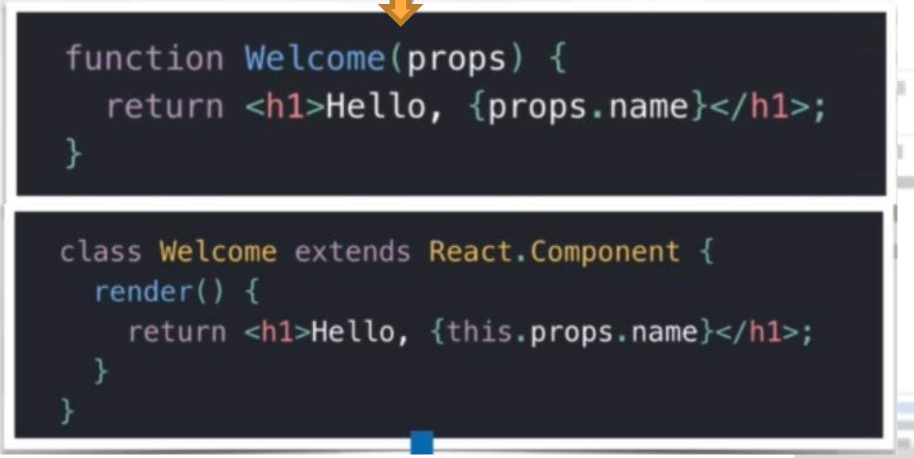


2. Build website like LEGO blocks

- Component – from data – simply as a **function** or a **class**
  - Input: some attributes (props)
  - Return: HTML inside JS

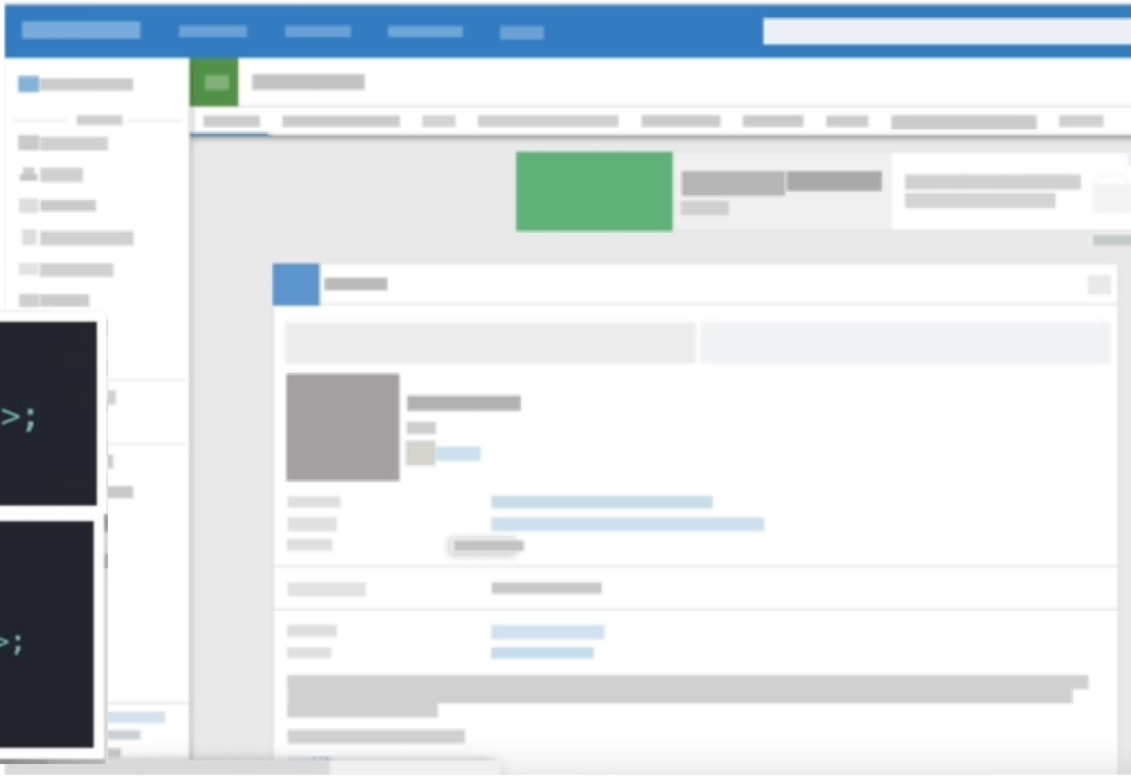


```
let state = {
    user: 'Andrei Neagoie',
    isLoggedIn: True,
    friends: ['Pavel', 'Matt', 'Joy']
}
```



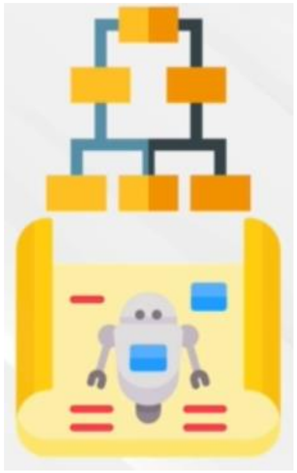
```
function Welcome(props) {
  return <h1>Hello, {props.name}</h1>;
}
```

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

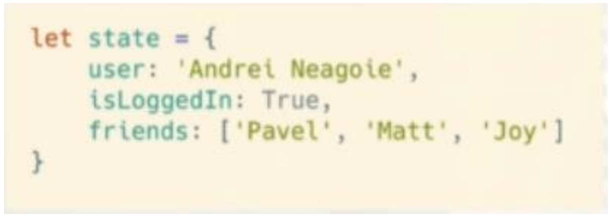


3. Unidirectional data flow

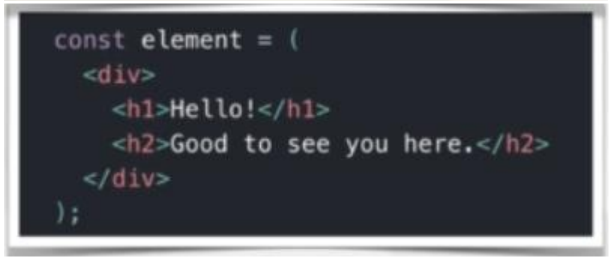
Virtual DOM



State



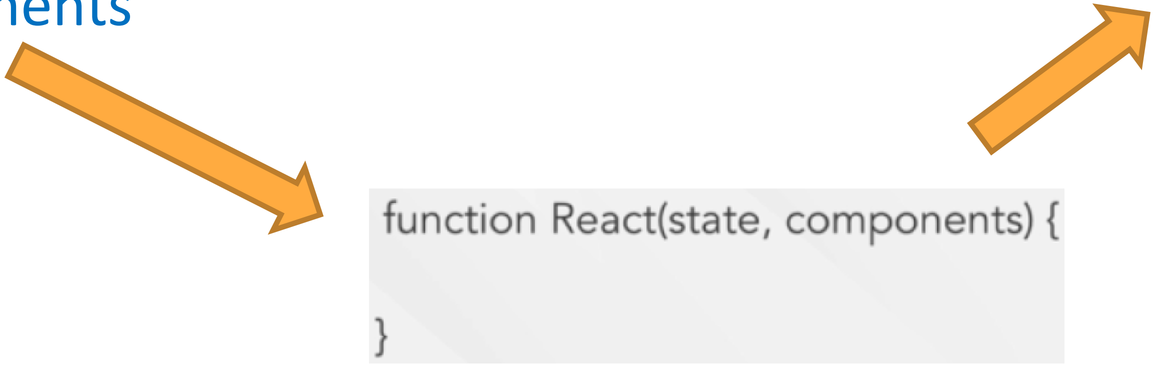
```
let state = {
    user: 'Andrei Neagoie',
    isLoggedIn: True,
    friends: ['Pavel', 'Matt', 'Joy']
}
```



```
const element = (
  <div>
    <h1>Hello!</h1>
    <h2>Good to see you here.</h2>
  </div>
);
```

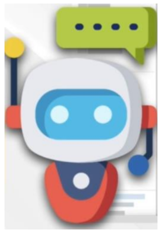
JSX

Components



```
function React(state, components) {

}
```

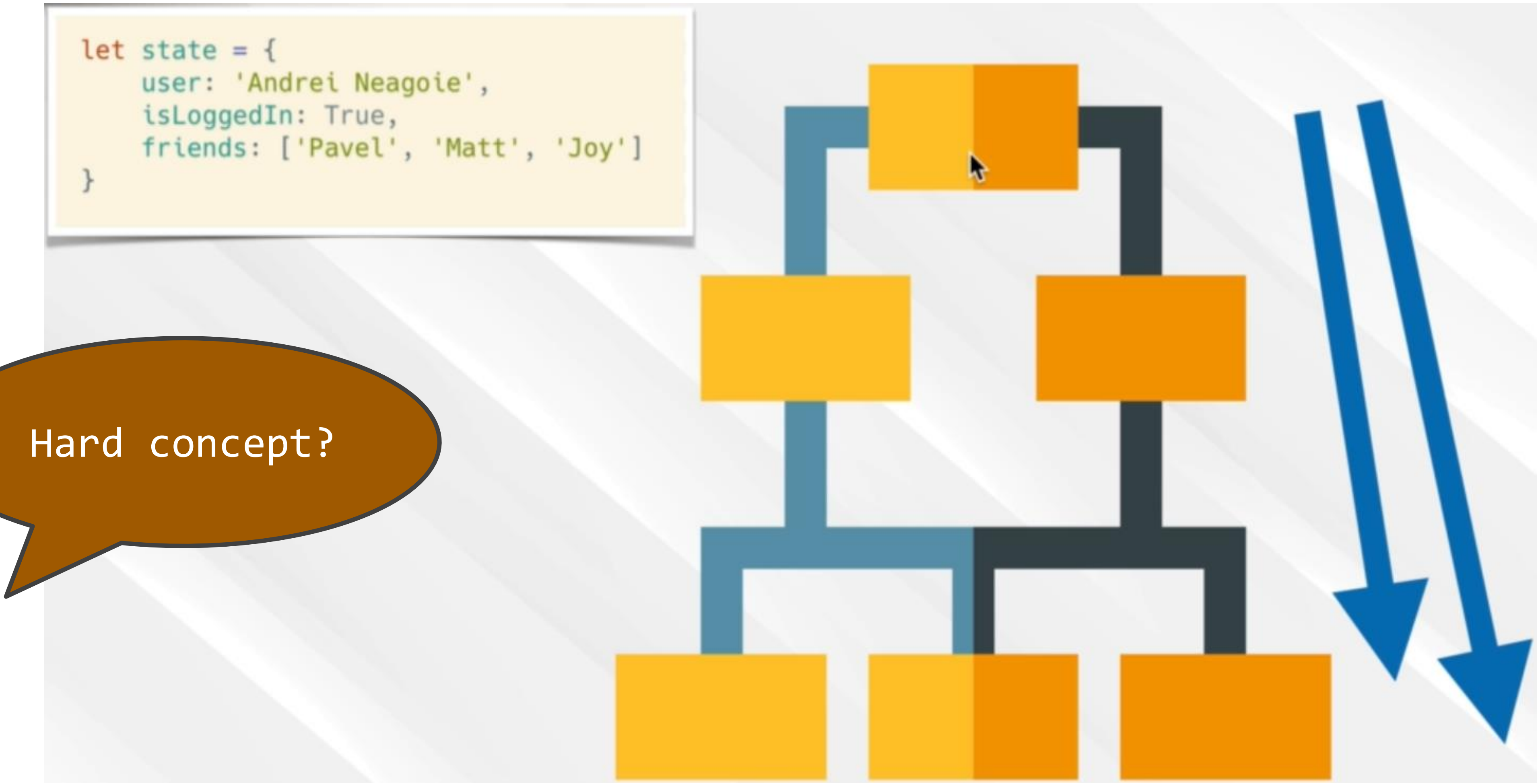


3. Unidirectional data flow

Anytime we want to **change the webpage**

→ **change the state**

3. Unidirectional data flow



```
let state = {
    user: 'Andrei Neagoie',
    isLoggedIn: True,
    friends: ['Pavel', 'Matt', 'Joy']
}
```

3. Unidirectional data flow

Data **never move up**
All the changes can **only trigger down**

4. UI, The rest is up to you

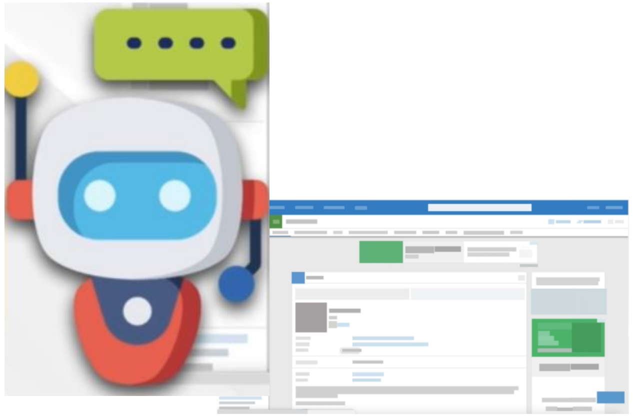
- AngularJS – a framework
- React: UI library (view only)

→ **React everywhere:**

- same principles with JS → build Cross platform

→ (React native – mobile, VR, React desktop, terminal)

React core lib: general robot



React DOM library: specific robot for DOM

How to become
a good React developer

React Keywords

State

Declarative

JSX

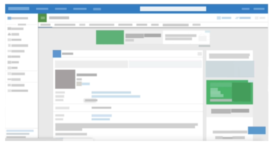
VirtualDOM

Components

Props

The job of a React Developer

1. Decide on Components



2. Decide the State and where it lives

3. What changes when state changes