How React Works

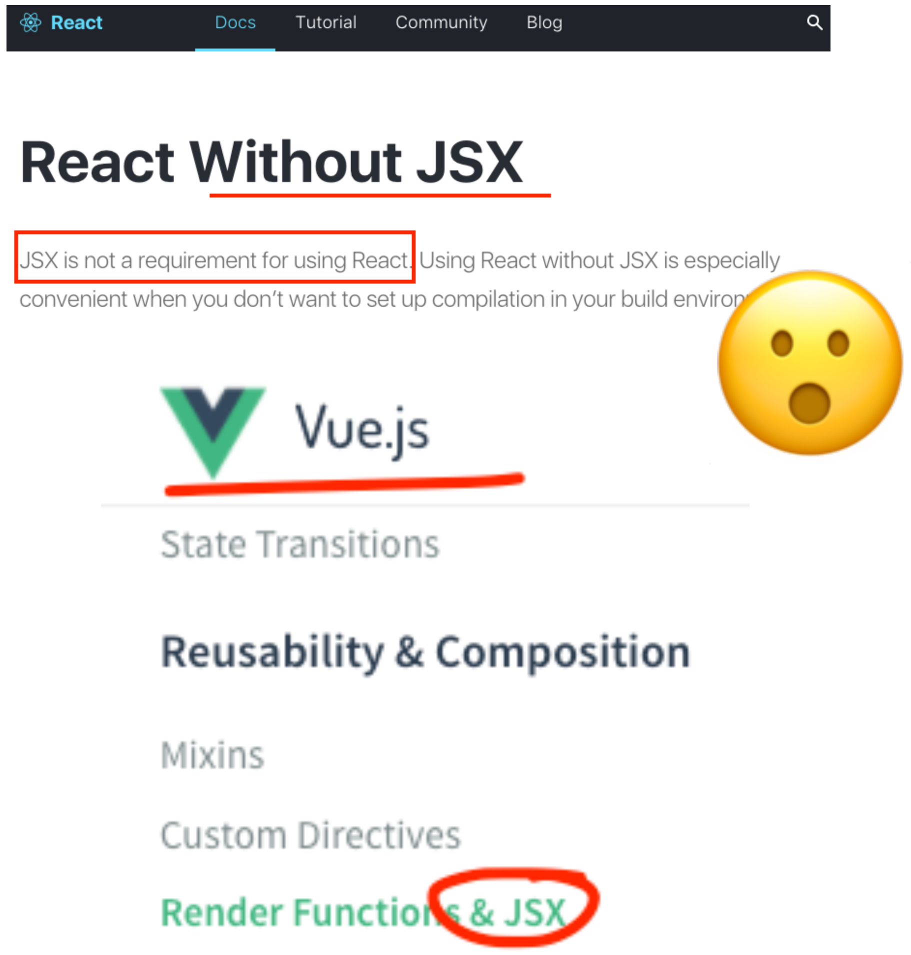
Kevin Li

Why React?

- I like JSX
- I like components
- Something something performance
- Something something virtual DOM
- Something something DOM diffing

Why React?

- ~~I like JSX~~
- I like components
- Something something performance
- Something something virtual DOM
- Something something DOM diffing

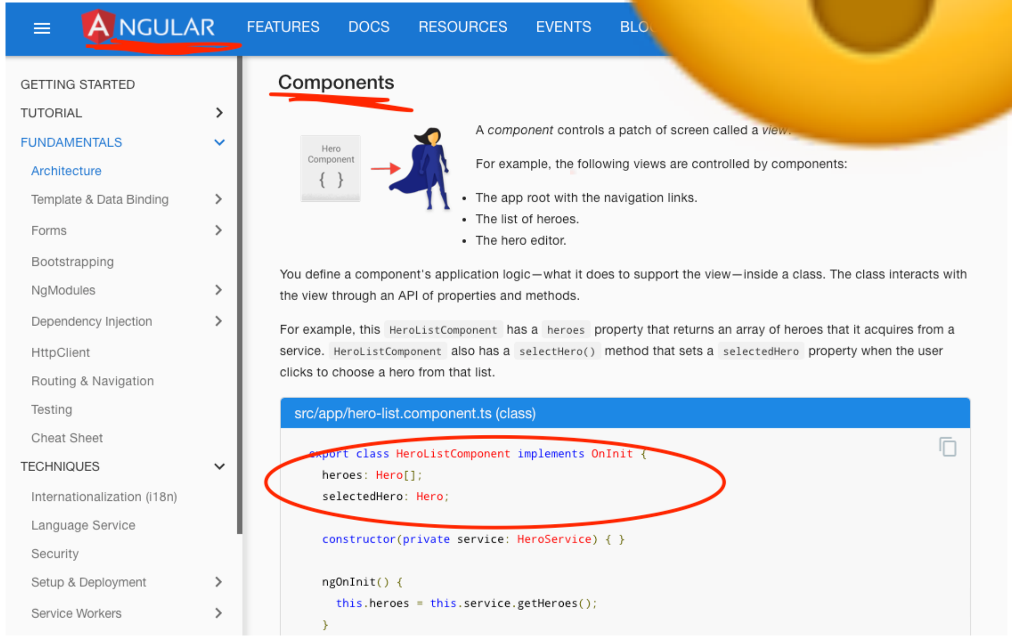


React Without JSX

JSX is not a requirement for using React. Using React without JSX is especially convenient when you don't want to set up compilation in your build environ

Why React?

- ~~I like JSX~~
- ~~I like components~~
- Something something performance
- Something something virtual DOM
- Something something DOM diffing



Components

A *component* controls a patch of screen called a *view*.

For example, the following views are controlled by components:

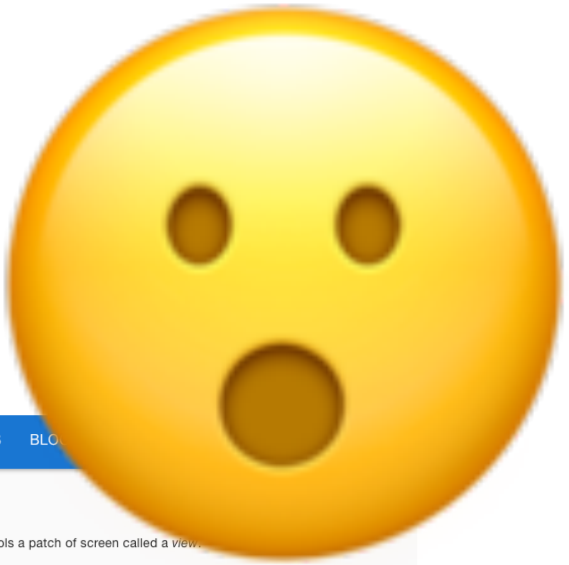
- The app root with the navigation links.
- The list of heroes.
- The hero editor.

You define a component's application logic—what it does to support the view—inside a class. The class interacts with the view through an API of properties and methods.

For example, this `HeroListComponent` has a `heroes` property that returns an array of heroes that it acquires from a service. `HeroListComponent` also has a `selectHero()` method that sets a `selectedHero` property when the user clicks to choose a hero from that list.

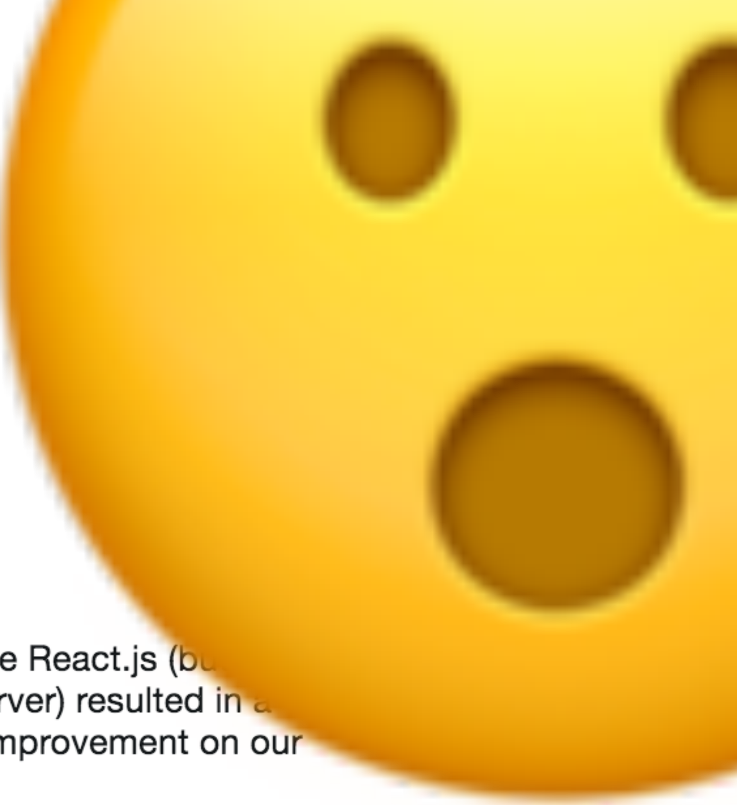
src/app/hero-list.component.ts (class)

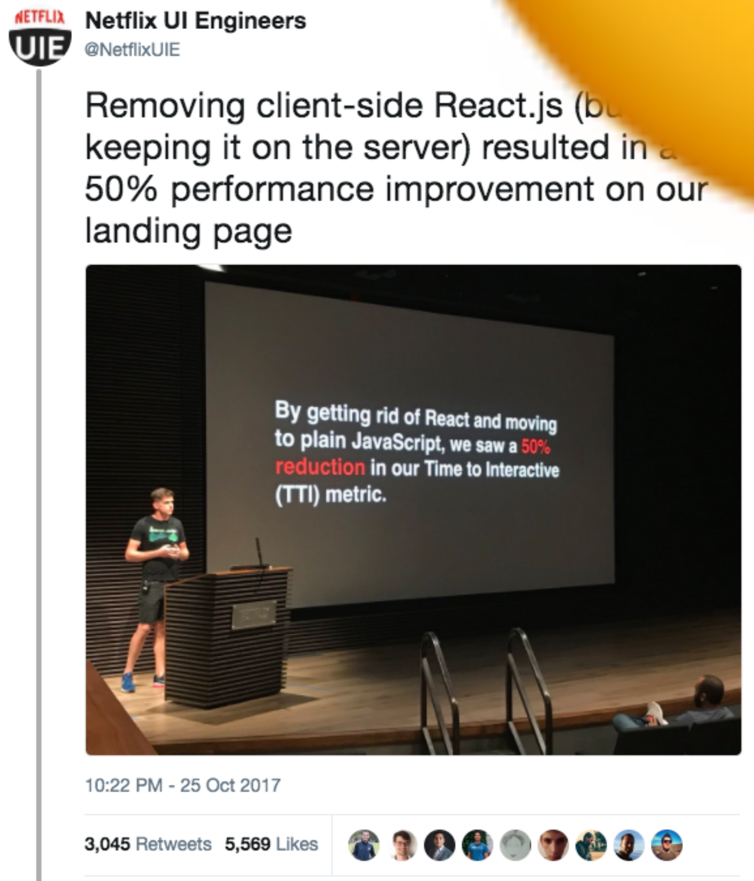
```
export class HeroListComponent implements OnInit {
  heroes: Hero[];
  selectedHero: Hero;

  constructor(private service: HeroService) { }

  ngOnInit() {
    this.heroes = this.service.getHeroes();
  }
```



Why React?

- ~~I like JSX~~
- ~~I like components~~
- Something something performance
- Something something virtual DOM
- Something something DOM diffing





Netflix UI Engineers
@NetflixUIE

Removing client-side React.js (b
keeping it on the server) resulted in
50% performance improvement on our
landing page

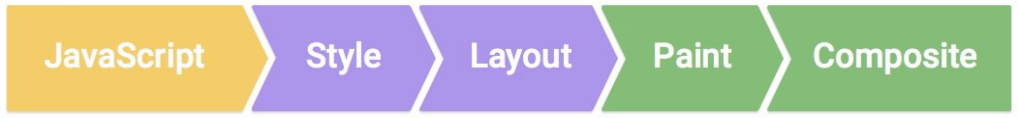
By getting rid of React and moving to plain JavaScript, we saw a 50% reduction in our Time to Interactive (TTI) metric.

10:22 PM - 25 Oct 2017

3,045 Retweets 5,569 Likes

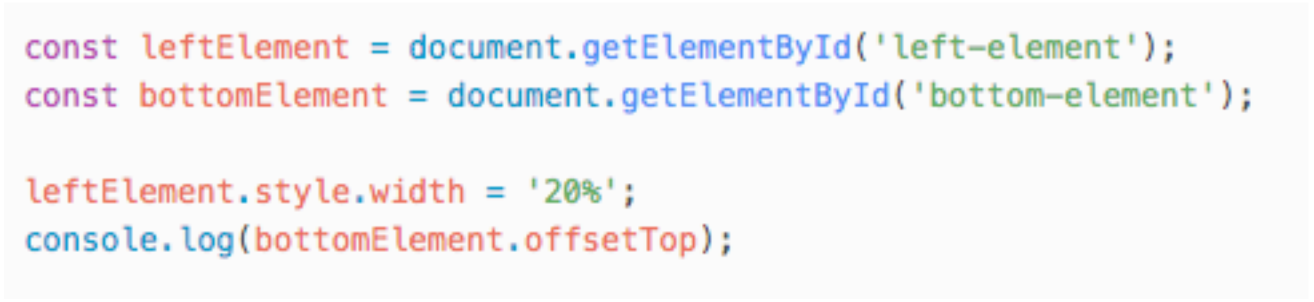
Browser Rendering Pipeline

- The browser goes through several steps to draw the page
- If you write JS or CSS that changes the style or layout or directly touches the DOM, you could trigger each of these calculations again!

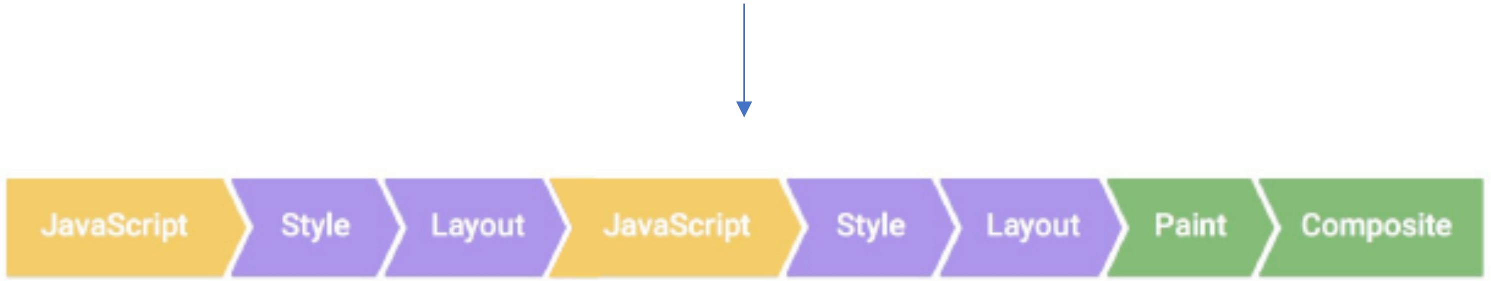


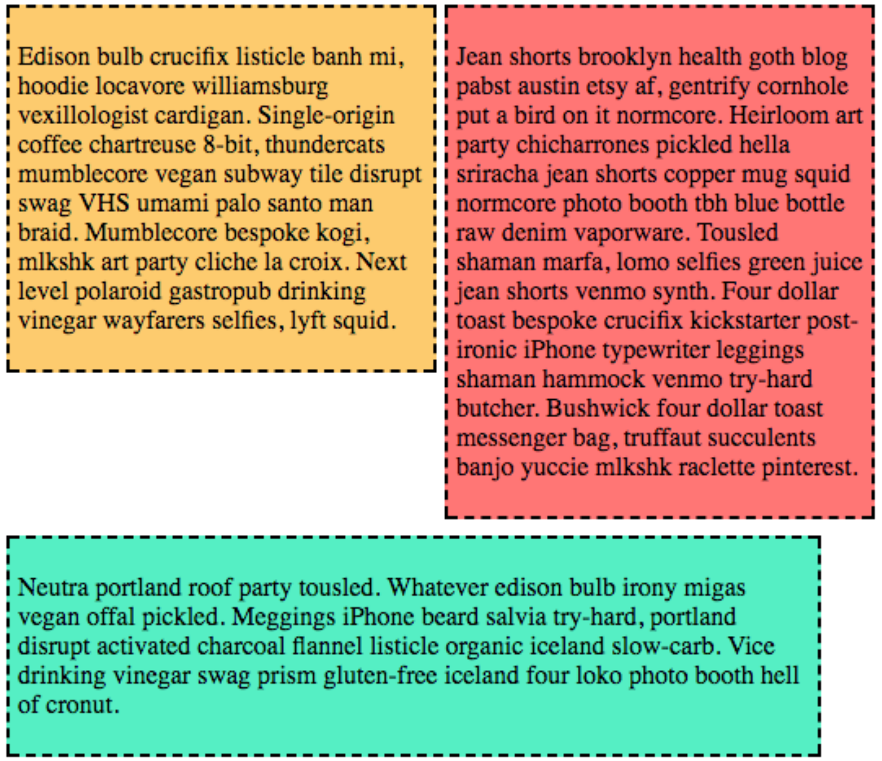
Source: https://developers.google.com/web/fundamentals/performance/rendering/

Problem



```
const leftElement = document.getElementById('left-element');
const bottomElement = document.getElementById('bottom-element');

leftElement.style.width = '20%';
console.log(bottomElement.offsetTop);
```



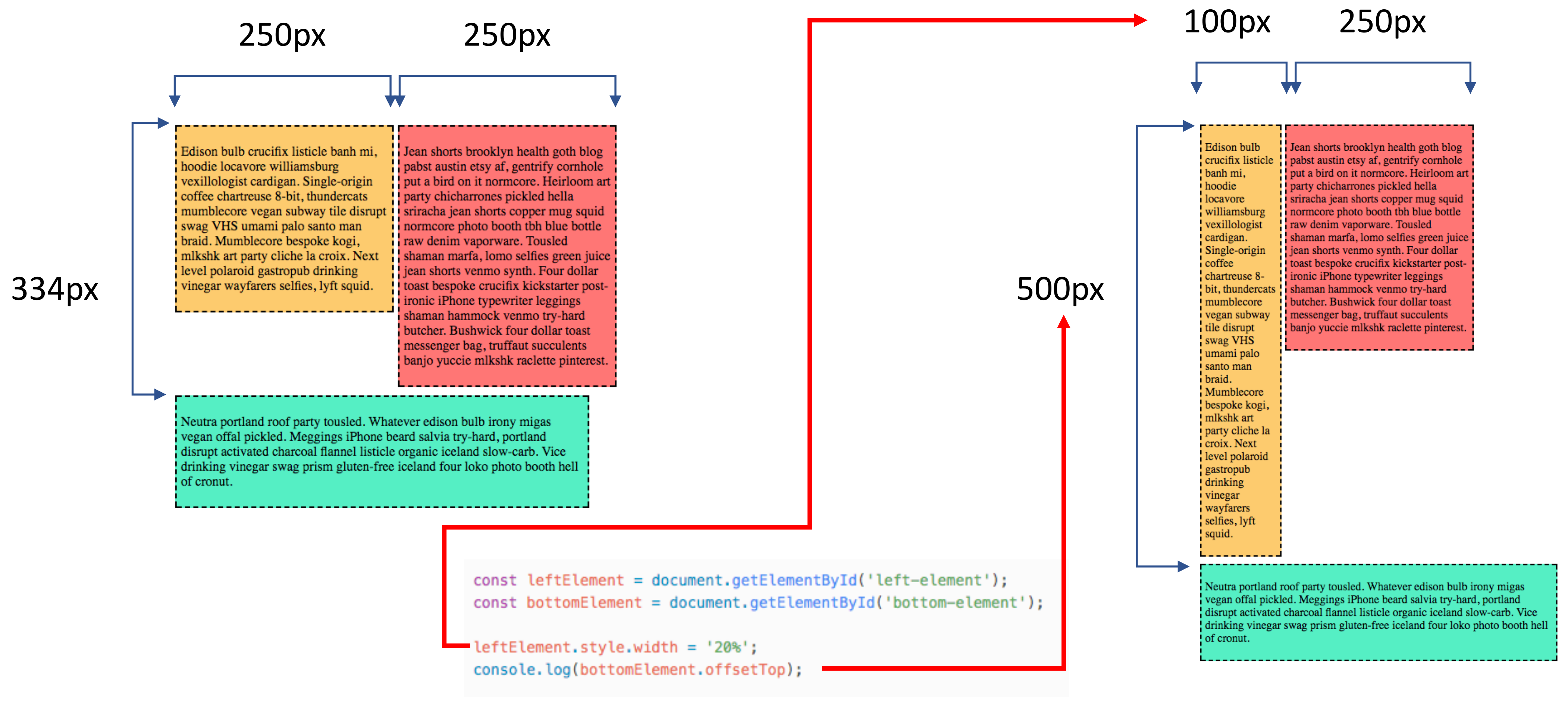


Edison bulb crucifix listicle banh mi, hoodie locavore williamsburg vexillologist cardigan. Single-origin coffee chartreuse 8-bit, thundercats mumblecore vegan subway tile disrupt swag VHS umami palo santo man braid. Mumblecore bespoke kogi, mlkshk art party cliche la croix. Next level polaroid gastropub drinking vinegar wayfarers selfies, lyft squid.

Jean shorts brooklyn health goth blog pabst austin etsy af, gentrify cornhole put a bird on it normcore. Heirloom art party chicharrones pickled hella sriracha jean shorts copper mug squid normcore photo booth tbh blue bottle raw denim vaporware. Tousled shaman marfa, lomo selfies green juice jean shorts venmo synth. Four dollar toast bespoke crucifix kickstarter post-ironic iPhone typewriter leggings shaman hammock venmo try-hard butcher. Bushwick four dollar toast messenger bag, truffaut succulents banjo yuccie mlkshk raclette pinterest.

Neutra portland roof party tousled. Whatever edison bulb irony migas vegan offal pickled. Meggings iPhone beard salvia try-hard, portland disrupt activated charcoal flannel listicle organic iceland slow-carb. Vice drinking vinegar swag prism gluten-free iceland four loko photo booth hell of cronut.

Problem



Edison bulb crucifix listicle banh mi, hoodie locavore williamsburg vexillologist cardigan. Single-origin coffee chartreuse 8-bit, thundercats mumblecore vegan subway tile disrupt swag VHS umami palo santo man braid. Mumblecore bespoke kogi, mlkshk art party cliche la croix. Next level polaroid gastropub drinking vinegar wayfarers selfies, lyft squid.

Jean shorts brooklyn health goth blog pabst austin etsy af, gentrify cornhole put a bird on it normcore. Heirloom art party chicharrones pickled hella sriracha jean shorts copper mug squid normcore photo booth tbh blue bottle raw denim vaporware. Tousled shaman marfa, lomo selfies green juice jean shorts venmo synth. Four dollar toast bespoke crucifix kickstarter post-ironic iPhone typewriter leggings shaman hammock venmo try-hard butcher. Bushwick four dollar toast messenger bag, truffaut succulents banjo yuccie mlkshk raclette pinterest.

Neutra portland roof party tousled. Whatever edison bulb irony migas vegan offal pickled. Meggings iPhone beard salvia try-hard, portland disrupt activated charcoal flannel listicle organic iceland slow-carb. Vice drinking vinegar swag prism gluten-free iceland four loko photo booth hell of cronut.

```
const leftElement = document.getElementById('left-element');
const bottomElement = document.getElementById('bottom-element');

leftElement.style.width = '20%';
console.log(bottomElement.offsetTop);
```

Edison bulb crucifix listicle banh mi, hoodie locavore williamsburg vexillologist cardigan. Single-origin coffee chartreuse 8-bit, thundercats mumblecore vegan subway tile disrupt swag VHS umami palo santo man braid. Mumblecore bespoke kogi, mlkshk art party cliche la croix. Next level polaroid gastropub drinking vinegar wayfarers selfies, lyft squid.

Jean shorts brooklyn health goth blog pabst austin etsy af, gentrify cornhole put a bird on it normcore. Heirloom art party chicharrones pickled hella sriracha jean shorts copper mug squid normcore photo booth tbh blue bottle raw denim vaporware. Tousled shaman marfa, lomo selfies green juice jean shorts venmo synth. Four dollar toast bespoke crucifix kickstarter post-ironic iPhone typewriter leggings shaman hammock venmo try-hard butcher. Bushwick four dollar toast messenger bag, truffaut succulents banjo yuccie mlkshk raclette pinterest.

Neutra portland roof party tousled. Whatever edison bulb irony migas vegan offal pickled. Meggings iPhone beard salvia try-hard, portland disrupt activated charcoal flannel listicle organic iceland slow-carb. Vice drinking vinegar swag prism gluten-free iceland four loko photo booth hell of cronut.

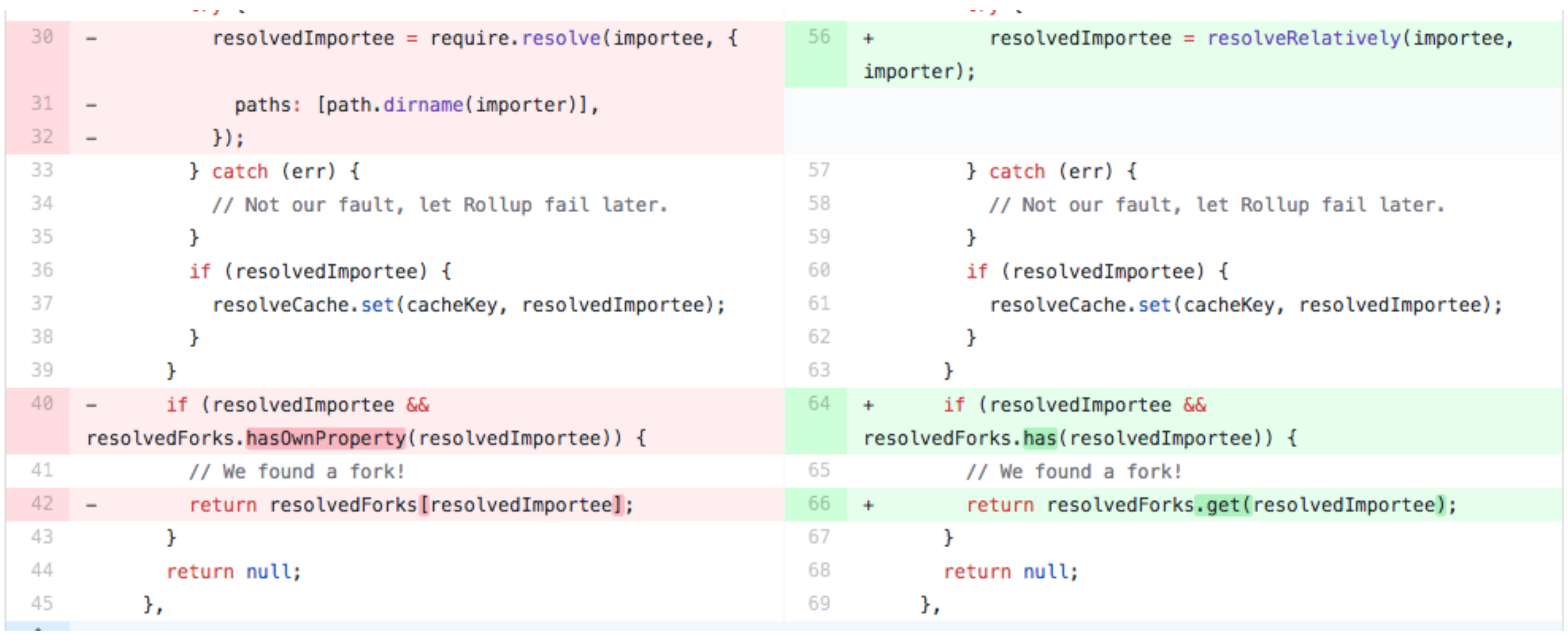
Imagine a Dumb JS Framework

We can make it fancier…

…but it's still dumb.

DOM Diffing

- It's not this:



```
-          resolvedImportee = require.resolve(importee, {
-            paths: [path.dirname(importer)],
-          });
         } catch (err) {
           // Not our fault, let Rollup fail later.
         }
         if (resolvedImportee) {
           resolveCache.set(cacheKey, resolvedImportee);
         }
       }
-      if (resolvedImportee &&
resolvedForks.hasOwnProperty(resolvedImportee)) {
         // We found a fork!
-        return resolvedForks[resolvedImportee];
       }
       return null;
     },
```

```
+          resolvedImportee = resolveRelatively(importee,
importer);
         } catch (err) {
           // Not our fault, let Rollup fail later.
         }
         if (resolvedImportee) {
           resolveCache.set(cacheKey, resolvedImportee);
         }
       }
+      if (resolvedImportee &&
resolvedForks.has(resolvedImportee)) {
         // We found a fork!
+        return resolvedForks.get(resolvedImportee);
       }
       return null;
     },
```

DOM Diffing

- The rules are pretty simple:
  1. Did the element type change?
     - `<div />` is now `<button />` = CHANGE
  2. If the element type is the same, did the attributes change?
     - `<button title="Hello" />` is now `<button title="Goodbye" />` = CHANGE
- If there are child elements, recurse through those

Source: https://reactjs.org/docs/reconciliation.html#the-diffing-algorithm

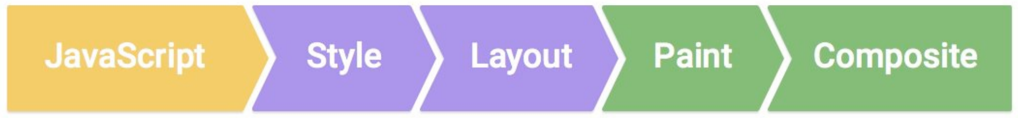
Let's Implement DOM Diffing

Baseline Performance:
Dumb Approach

First Attempt:
Naïve Approach

Naïve Approach: Why so much slower?

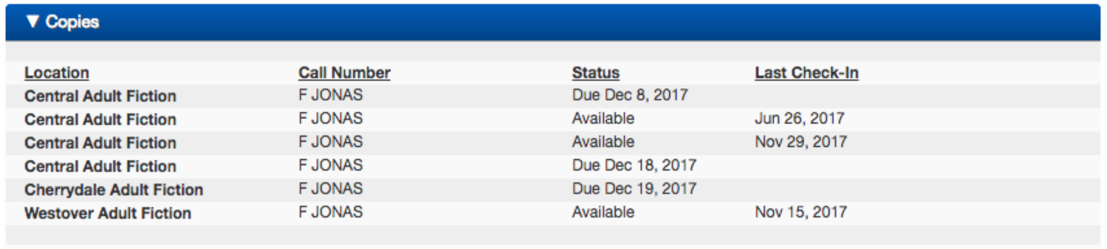
- Recall the browser rendering pipeline.
- Every iteration, we potentially perform two DOM operations:
  - *Read* the DOM to get the current value
  - *Write* to the DOM with the update element
- Every time we write to the DOM, we change it, so the browser's JS engine can't depend on its internal optimizations (cache) and has to recalculate the page's styles before it can read again.



Second Attempt:
Batched DOM Operations

Imagine going to a library…

or



▼ Copies

Location	Call Number	Status	Last Check-In
Central Adult Fiction	F JONAS	Due Dec 8, 2017	
Central Adult Fiction	F JONAS	Available	Jun 26, 2017
Central Adult Fiction	F JONAS	Available	Nov 29, 2017
Central Adult Fiction	F JONAS	Due Dec 18, 2017	
Cherrydale Adult Fiction	F JONAS	Due Dec 19, 2017	
Westover Adult Fiction	F JONAS	Available	Nov 15, 2017

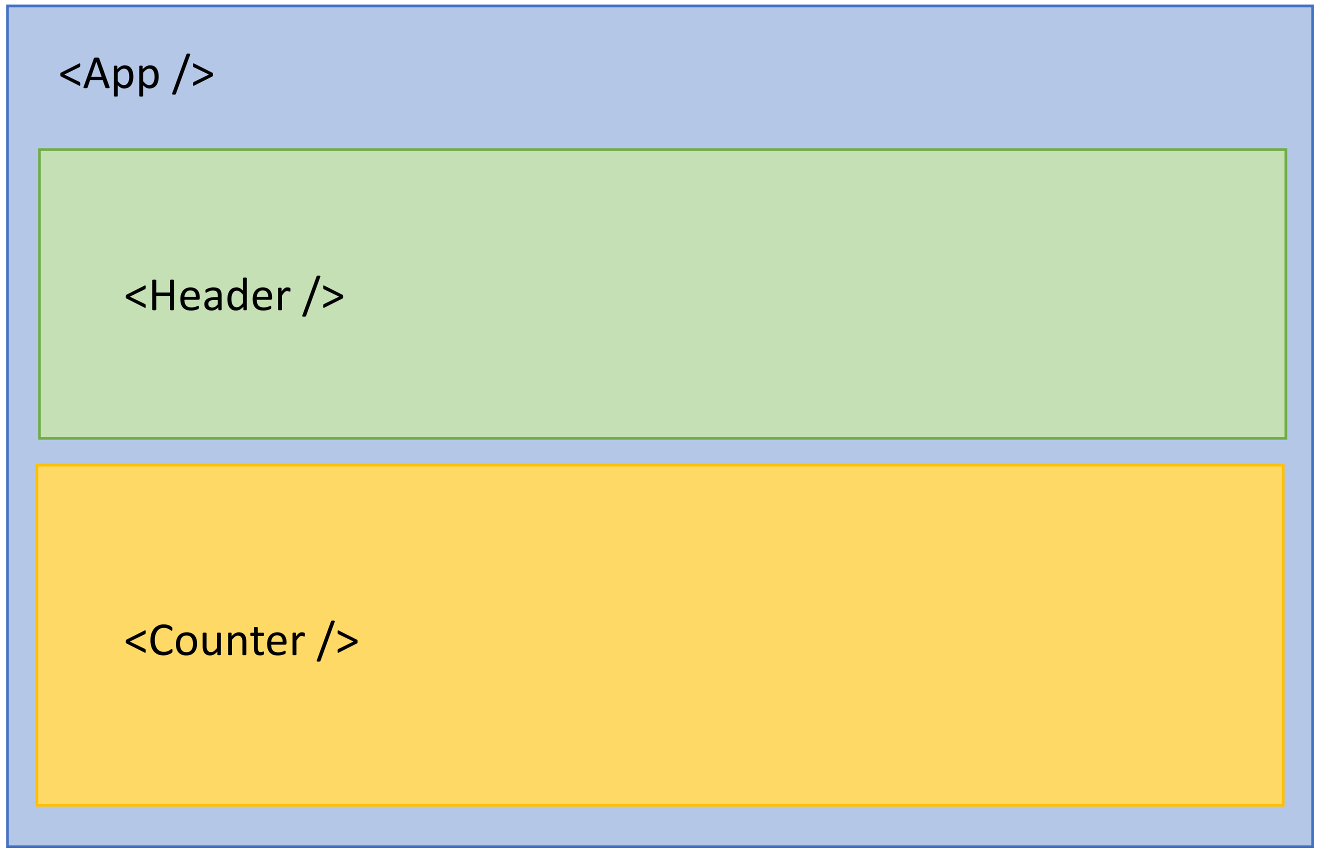
Final Attempt:
Virtual DOM

Why React?

- ~~I like JSX~~
- ~~I like components~~
- Something something performance
- Something something virtual DOM
- Something something DOM diffing

~~How~~ Why React Works

App Components



Types of Components

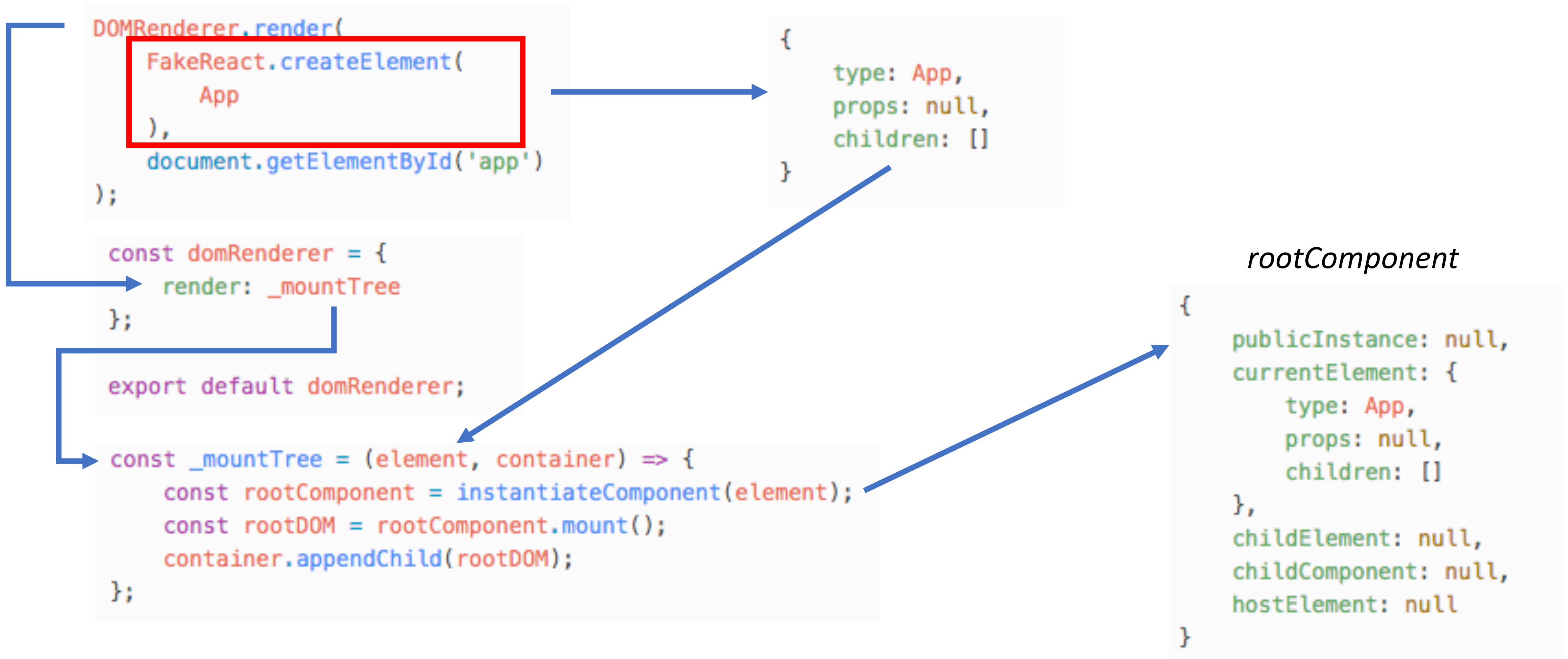
```
<MyComponent data={something} />
```

- Composite Component

```
<div className="my-css-class" />
```

- Host Component

Instantiating the App Component



```
DOMRenderer.render(
    FakeReact.createElement(
        App
    ),
    document.getElementById('app')
);
```

```
{
    type: App,
    props: null,
    children: []
}
```

```
const domRenderer = {
    render: _mountTree
};

export default domRenderer;
```

```
const _mountTree = (element, container) => {
    const rootComponent = instantiateComponent(element);
    const rootDOM = rootComponent.mount();
    container.appendChild(rootDOM);
};
```

rootComponent

```
{
    publicInstance: null,
    currentElement: {
        type: App,
        props: null,
        children: []
    },
    childElement: null,
    childComponent: null,
    hostElement: null
}
```

Mounting the App Component

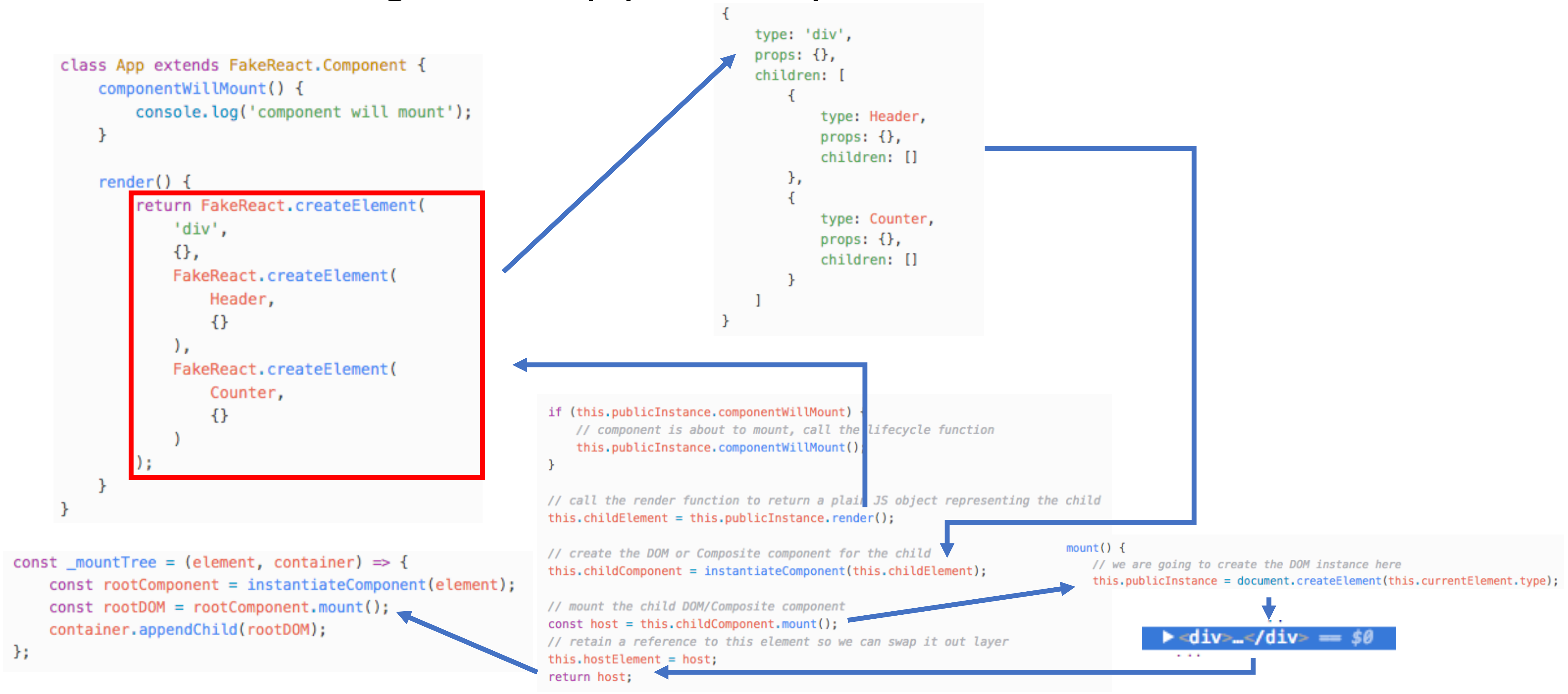
```
const _mountTree = (element, container) => {
    const rootComponent = instantiateComponent(element);
    const rootDOM = rootComponent.mount();
    container.appendChild(rootDOM);
};
```

```
{
    type: App,
    props: null,
    children: []
}
```

```
mount() {
    const element = this.currentElement;
    // instantiate an instance of the component class
    this.publicInstance = new element.type(element.props);
    // create a reference back to this component for future updating
    this.publicInstance._internalInstance = this;

    if (this.publicInstance.componentWillMount) {
        // component is about to mount, call the lifecycle function
        this.publicInstance.componentWillMount();
    }

    // call the render function to return a plain JS object representing the child
    this.childElement = this.publicInstance.render();

    // create the DOM or Composite component for the child
    this.childComponent = instantiateComponent(this.childElement);

    // mount the child DOM/Composite component
    const host = this.childComponent.mount();
    // retain a reference to this element so we can swap it out layer
    this.hostElement = host;
    return host;
}
```

```
class App extends FakeReact.Component {
    componentWillMount() {
        console.log('component will mount');
    }

    render() {
        return FakeReact.createElement(
            'div',
            {},
            FakeReact.createElement(
                Header,
                {}
            ),
            FakeReact.createElement(
                Counter,
                {}
            )
        );
    }
}
```

Rendering the App Component



```
class App extends FakeReact.Component {
    componentWillMount() {
        console.log('component will mount');
    }

    render() {
        return FakeReact.createElement(
            'div',
            {},
            FakeReact.createElement(
                Header,
                {}
            ),
            FakeReact.createElement(
                Counter,
                {}
            )
        );
    }
}
```

```
{
    type: 'div',
    props: {},
    children: [
        {
            type: Header,
            props: {},
            children: []
        },
        {
            type: Counter,
            props: {},
            children: []
        }
    ]
}
```

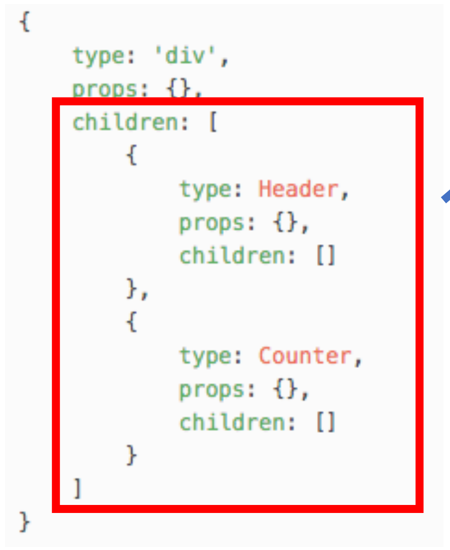
```
if (this.publicInstance.componentWillMount) {
    // component is about to mount, call the lifecycle function
    this.publicInstance.componentWillMount();
}

// call the render function to return a plain JS object representing the child
this.childElement = this.publicInstance.render();

// create the DOM or Composite component for the child
this.childComponent = instantiateComponent(this.childElement);

// mount the child DOM/Composite component
const host = this.childComponent.mount();
// retain a reference to this element so we can swap it out layer
this.hostElement = host;
return host;
```

```
mount() {
    // we are going to create the DOM instance here
    this.publicInstance = document.createElement(this.currentElement.type);
```

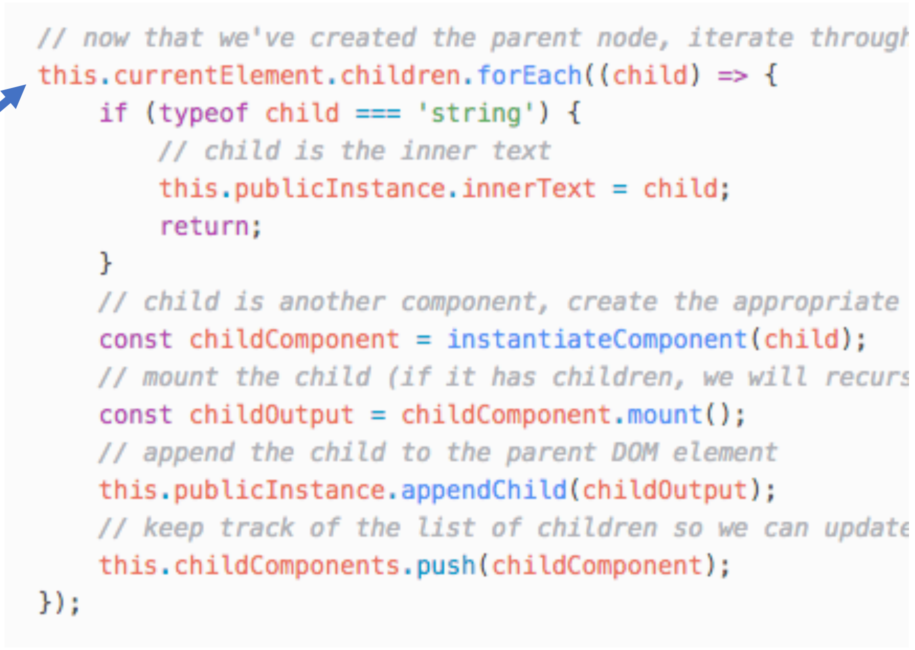
```
▶ <div>…</div> == $0
```

```
const _mountTree = (element, container) => {
    const rootComponent = instantiateComponent(element);
    const rootDOM = rootComponent.mount();
    container.appendChild(rootDOM);
};
```

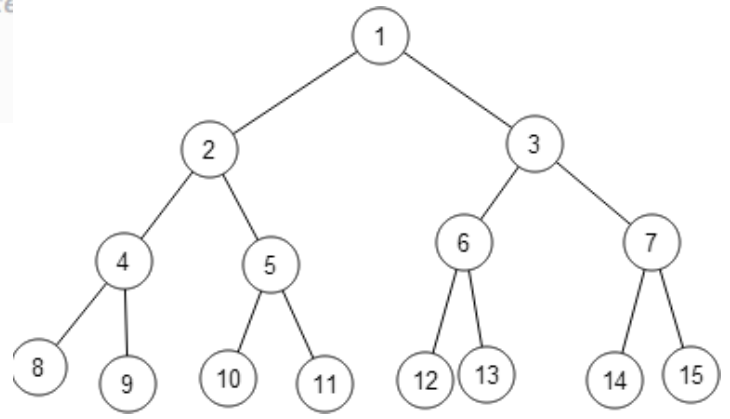
Handling Children



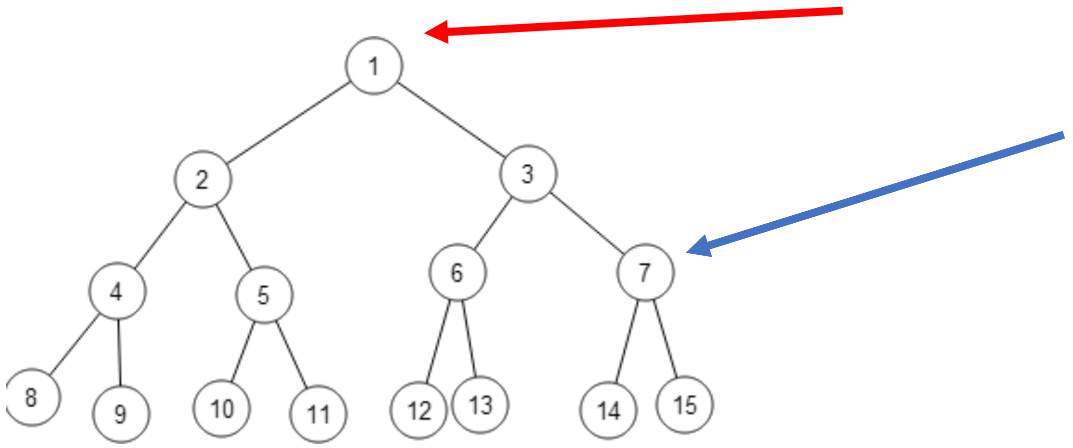
```
{
    type: 'div',
    props: {},
    children: [
        {
            type: Header,
            props: {},
            children: []
        },
        {
            type: Counter,
            props: {},
            children: []
        }
    ]
}
```



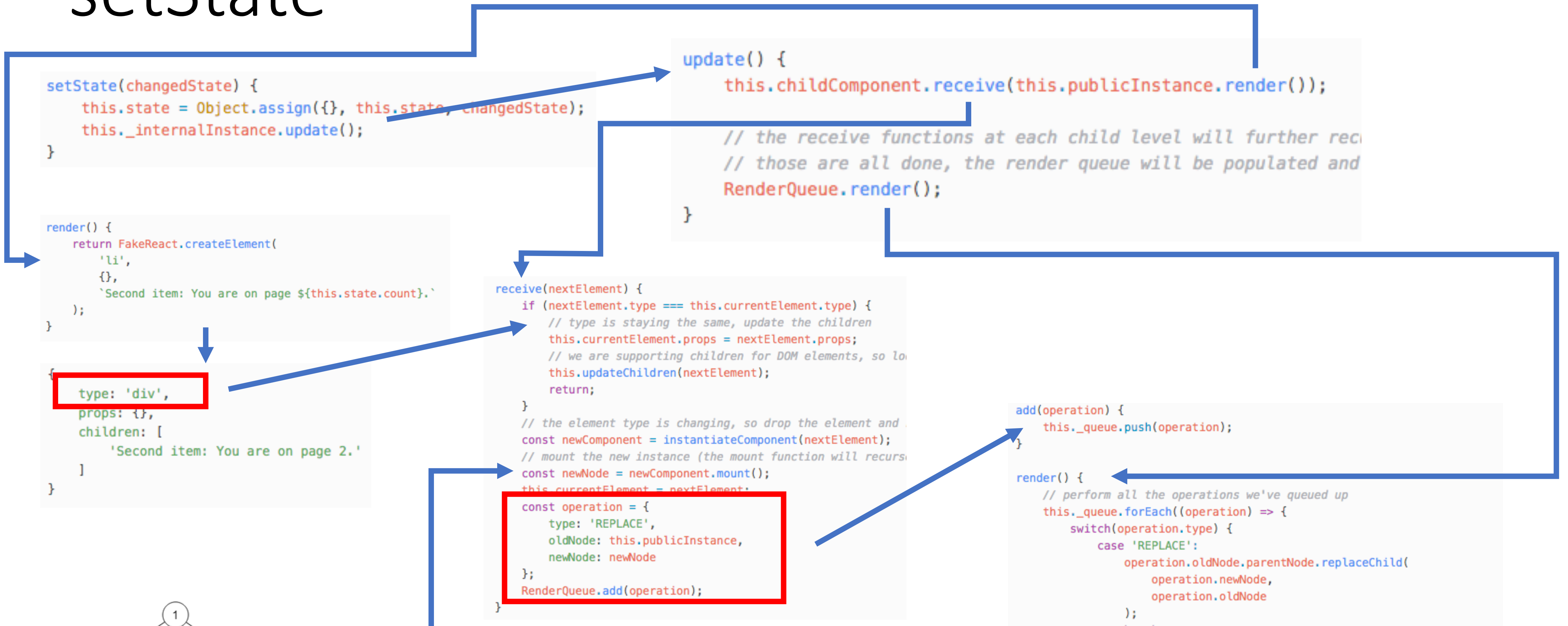
```
// now that we've created the parent node, iterate through
this.currentElement.children.forEach((child) => {
    if (typeof child === 'string') {
        // child is the inner text
        this.publicInstance.innerText = child;
        return;
    }
    // child is another component, create the appropriate
    const childComponent = instantiateComponent(child);
    // mount the child (if it has children, we will recurs
    const childOutput = childComponent.mount();
    // append the child to the parent DOM element
    this.publicInstance.appendChild(childOutput);
    // keep track of the list of children so we can update
    this.childComponents.push(childComponent);
});
```



setState



setState



```
setState(changedState) {
    this.state = Object.assign({}, this.state, changedState);
    this._internalInstance.update();
}
```

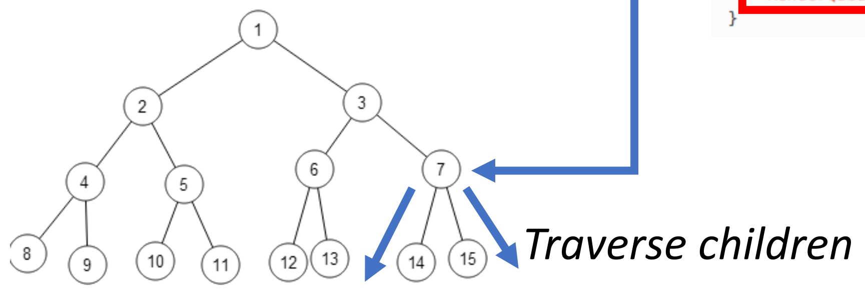
```
update() {
    this.childComponent.receive(this.publicInstance.render());

    // the receive functions at each child level will further rec
    // those are all done, the render queue will be populated and
    RenderQueue.render();
}
```

```
render() {
    return FakeReact.createElement(
        'li',
        {},
        `Second item: You are on page ${this.state.count}.`
    );
}
```

```
{
    type: 'div',
    props: {},
    children: [
        'Second item: You are on page 2.'
    ]
}
```

```
receive(nextElement) {
    if (nextElement.type === this.currentElement.type) {
        // type is staying the same, update the children
        this.currentElement.props = nextElement.props;
        // we are supporting children for DOM elements, so lo
        this.updateChildren(nextElement);
        return;
    }
    // the element type is changing, so drop the element and
    const newComponent = instantiateComponent(nextElement);
    // mount the new instance (the mount function will recurs
    const newNode = newComponent.mount();
    this.currentElement = nextElement;
    const operation = {
        type: 'REPLACE',
        oldNode: this.publicInstance,
        newNode: newNode
    };
    RenderQueue.add(operation);
}
```

```
add(operation) {
    this._queue.push(operation);
}

render() {
    // perform all the operations we've queued up
    this._queue.forEach((operation) => {
        switch(operation.type) {
            case 'REPLACE':
                operation.oldNode.parentNode.replaceChild(
                    operation.newNode,
                    operation.oldNode
                );
                break;
            case 'REMOVE':
                operation.node.parentNode.removeChild(operation.node);
                break;
        }
    });
```



More Details on the React Web Site

- https://reactjs.org/docs/implementation-notes.html
- https://reactjs.org/docs/reconciliation.html

- Demo code: https://github.com/kevinlig/how-react-works-demos