REACT JS

Mixins

Mixins

Components are generally the units of reuse in React

Components are composed into larger components to build stuff up

Mixins

Sometimes there might be a smaller unit of code that you might want to reuse

a helper function

some default set up

Mixins

smaller reusable unit of code

a helper function

some default set up

Mixins allow you to reuse code across components without composition

Mixins should be used sparingly because they break the composition model

Mixins

used sparingly

Leads to implicit dependences rather than explicit composition dependencies

Complexity snowballs as mixins end up as a parallel hierarchy

Mixins

The new ES6 classes for React do not support mixins

EXAMPLE 29

Mixins.html

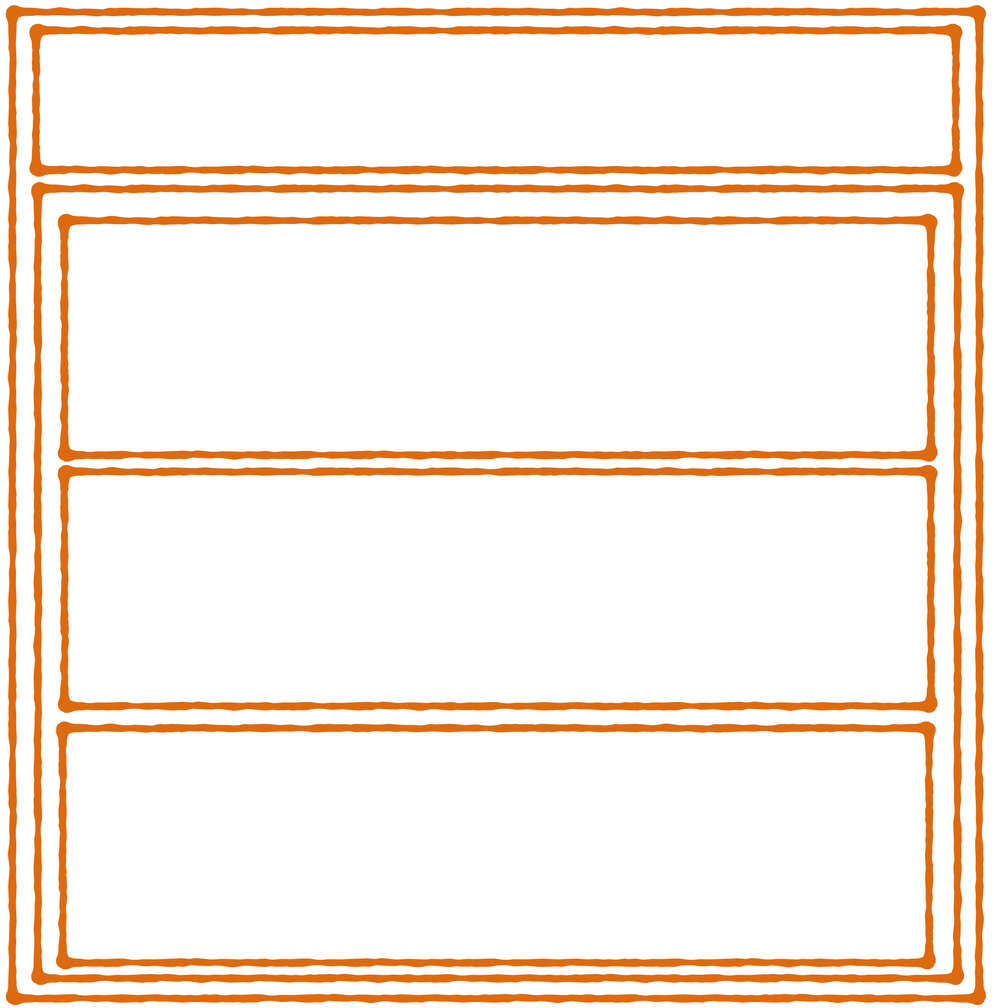
EXAMPLE 30

NestedAndMultipleMixins.html

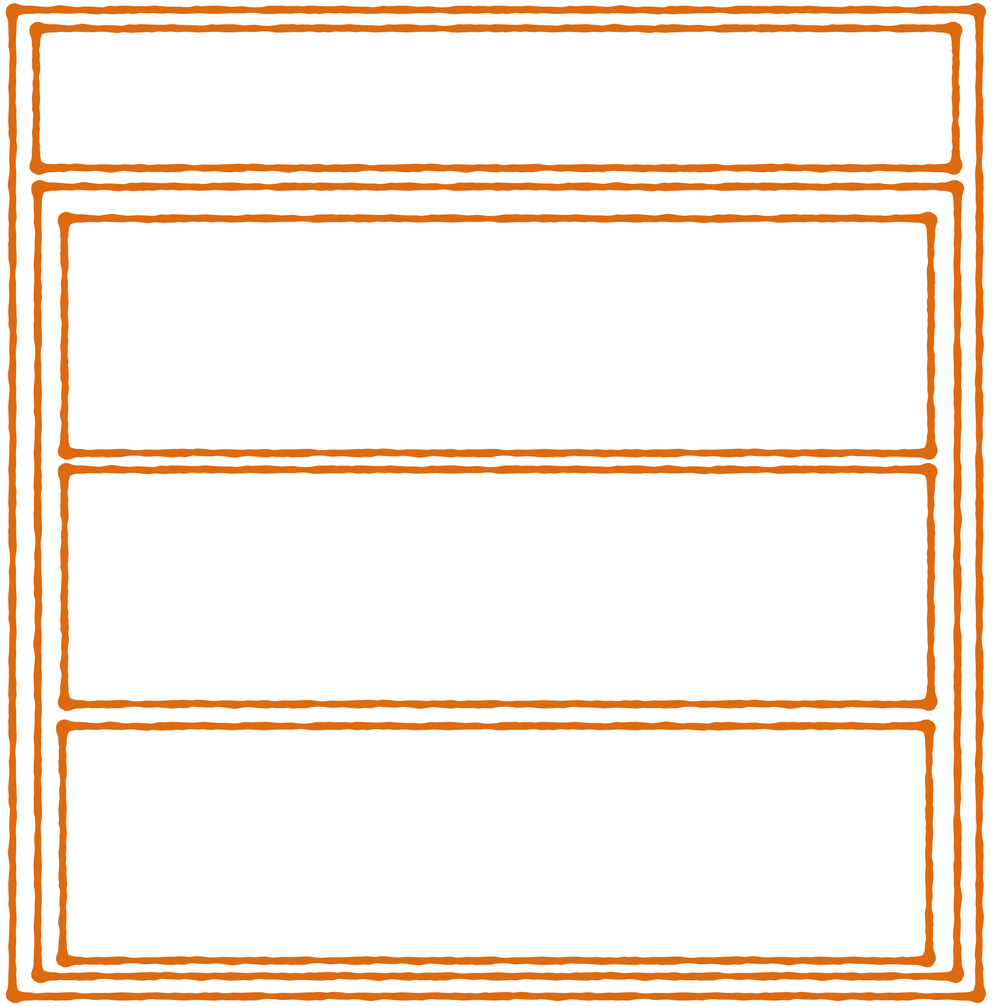
REACT JS

ES6 And Classes

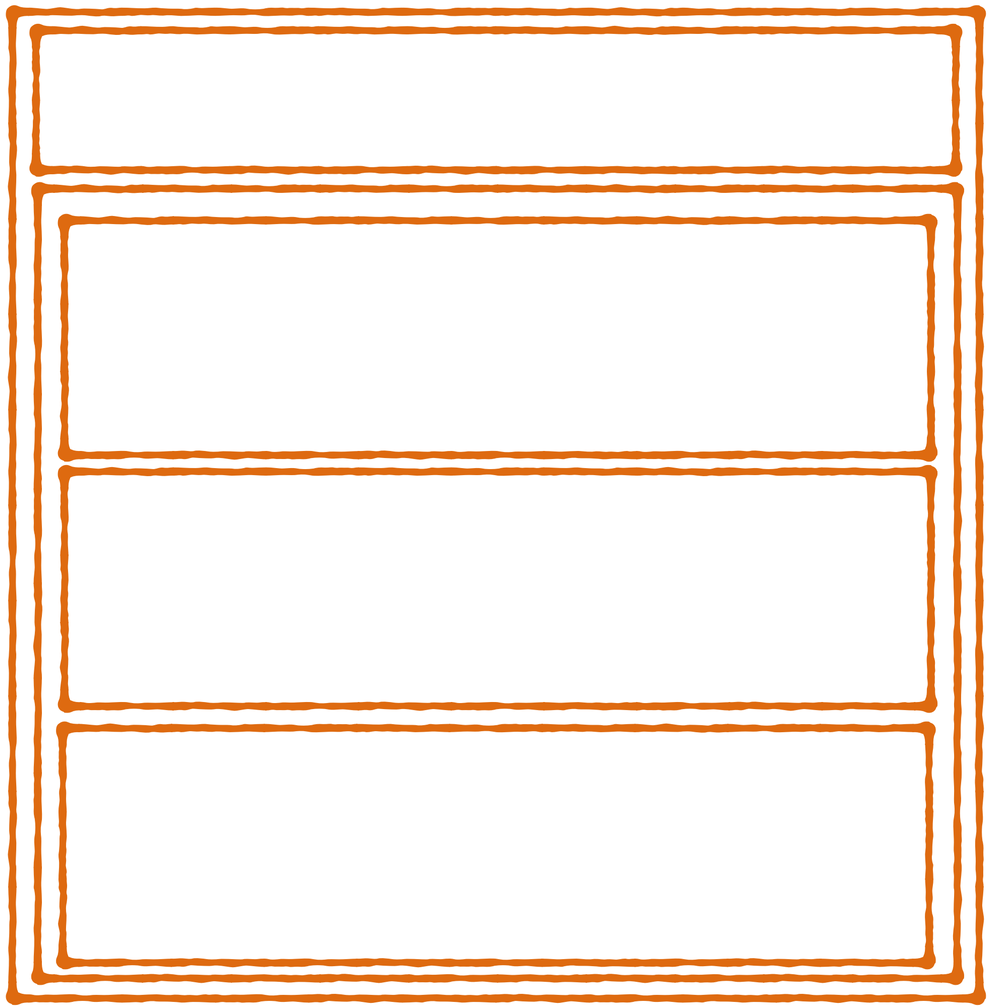
React breaks up the UI into components



Each component knows how to **render itself** using its props and state



Each component is a **reusable** unit, which can be developed in **isolation**



ES6 And Classes

ECMAScript 6, is the latest specification of the Javascript

It has special syntactic sugar which makes writing components easier

`React.Component`

An abstract base class to represent a component

`React.Component`

This uses Javascript's prototype inheritance under the hood - it just feels like a OO class

`React.Component`

Each function on the createClass object specification has an equivalent in this class

EXAMPLE 31

Es6Class.html

REACT JS

Forms

Elements in forms are typically those which accept input

The element values are editable, and internally store their own state

Integrating forms with React requires:

Making the component's state
the single source of truth

Intercepting form submission
and getting access to form data

Controlled components

`<input>` `<textarea>` `<select>`

These are components whose internal state is **merged** with React's state

EXAMPLE 32

ControlledComponents.html

EXAMPLE 33

FormSetup.html

EXAMPLE 34

FormValidation.html

REACT JS

Accessing DOM Elements

Accessing DOM Elements

`this.props` = accesses a component's properties

`this.state` = accesses a component's state

`this.refs`

`this.refs`

Allows access to DOM elements within a component

`this.refs`

Only those elements which have been set up with a reference

EXAMPLE 35

AccessingTheDOM.html

EXAMPLE 36

AccessingTheDOMInAComponent.html

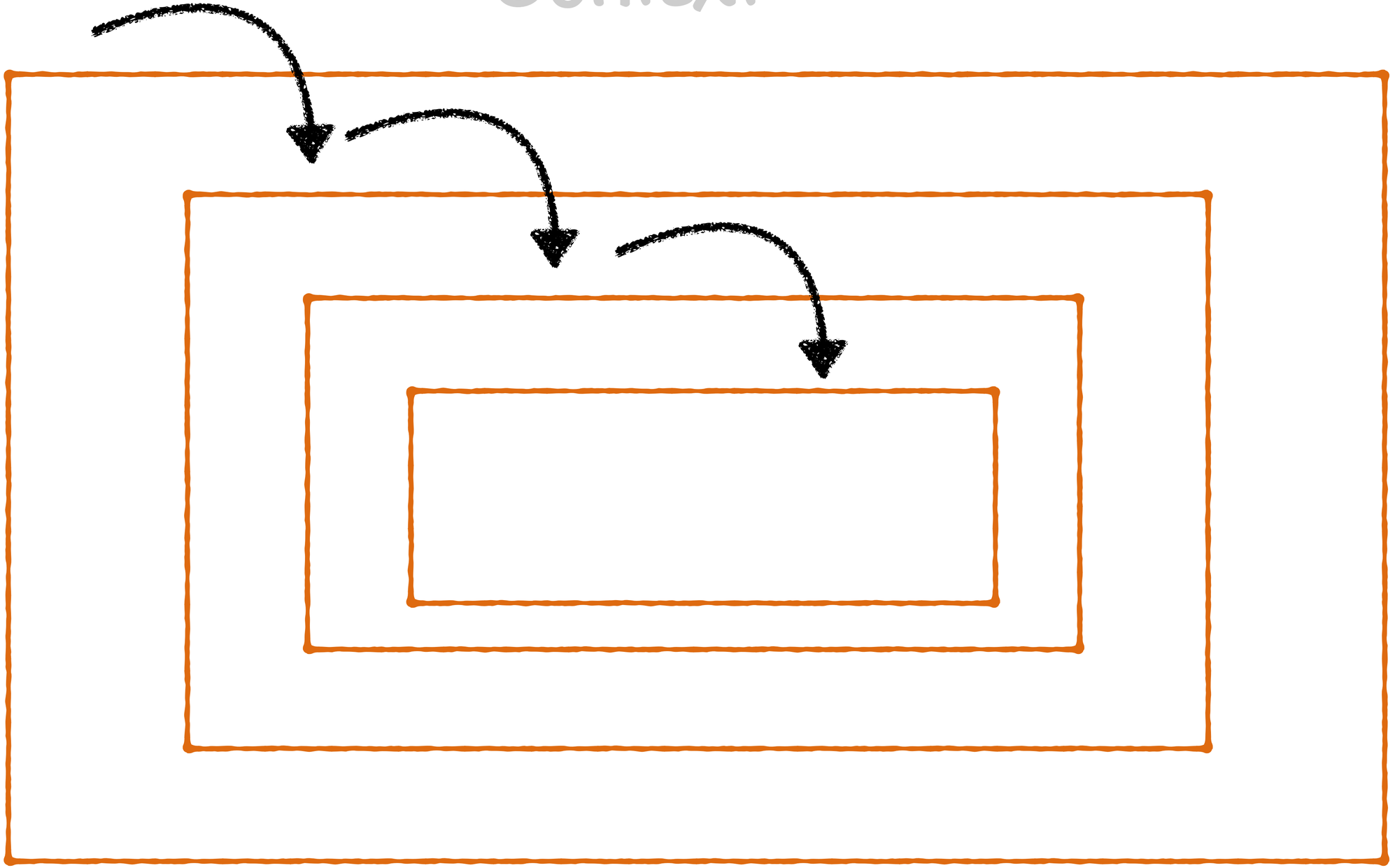
REACT JS

Context

Context

Properties in React flow from top-level components to lower-level components

Context



This makes React components structured and clear (you can always see what props a particular component has!)

Context

There might be times that you want to pass data directly to a child without manually passing properties at every level

Context

You can!

Using the new experimental context API

This can change in future React releases so be careful when you use it!

Using context is primarily for experienced developers

EXAMPLE 37

Context.html

REACT JS

DOM Reconciliation

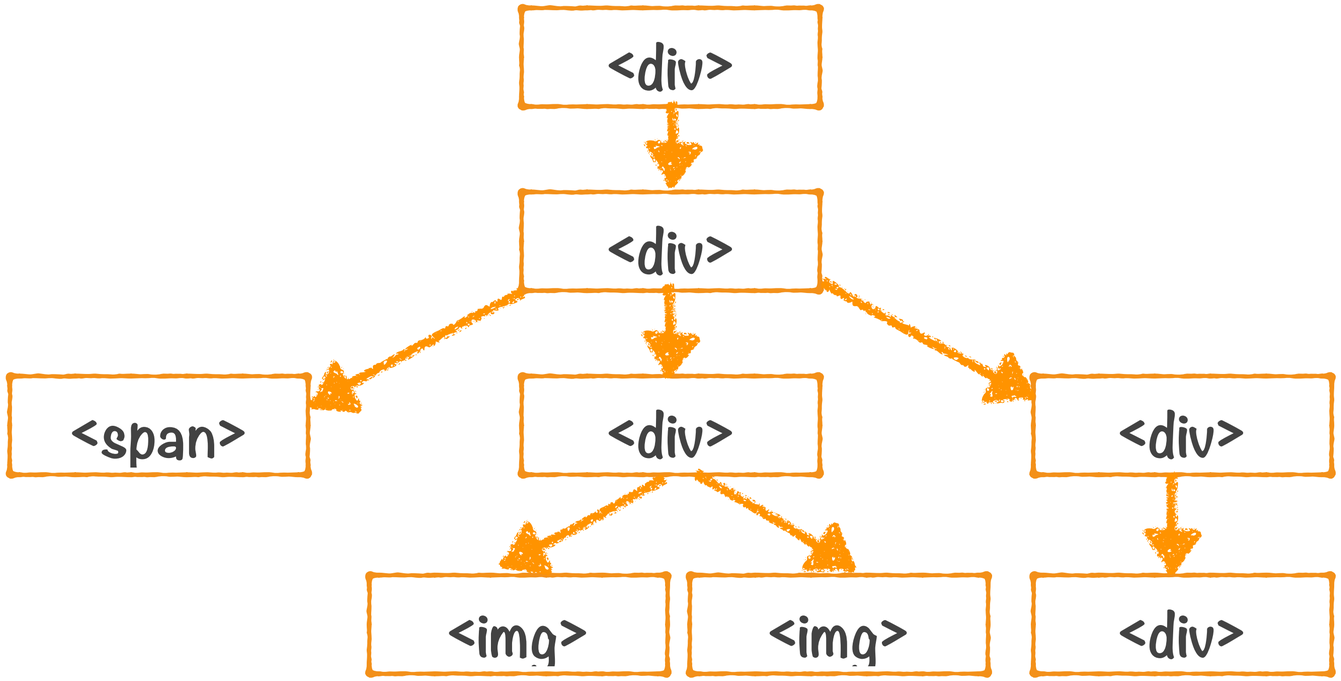
DOM Reconciliation

React has a declarative API

As a developer you just indicate what you want displayed

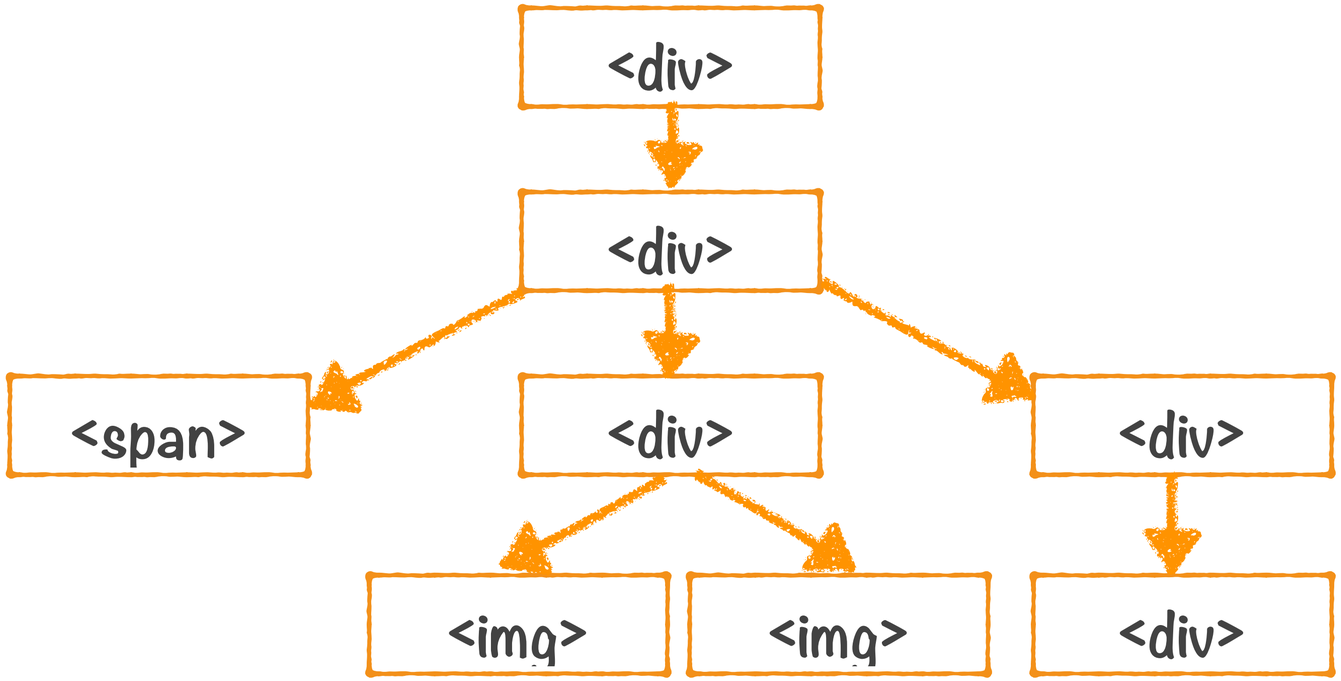
And leave it to React to make the changes in an optimal way

DOM Reconciliation



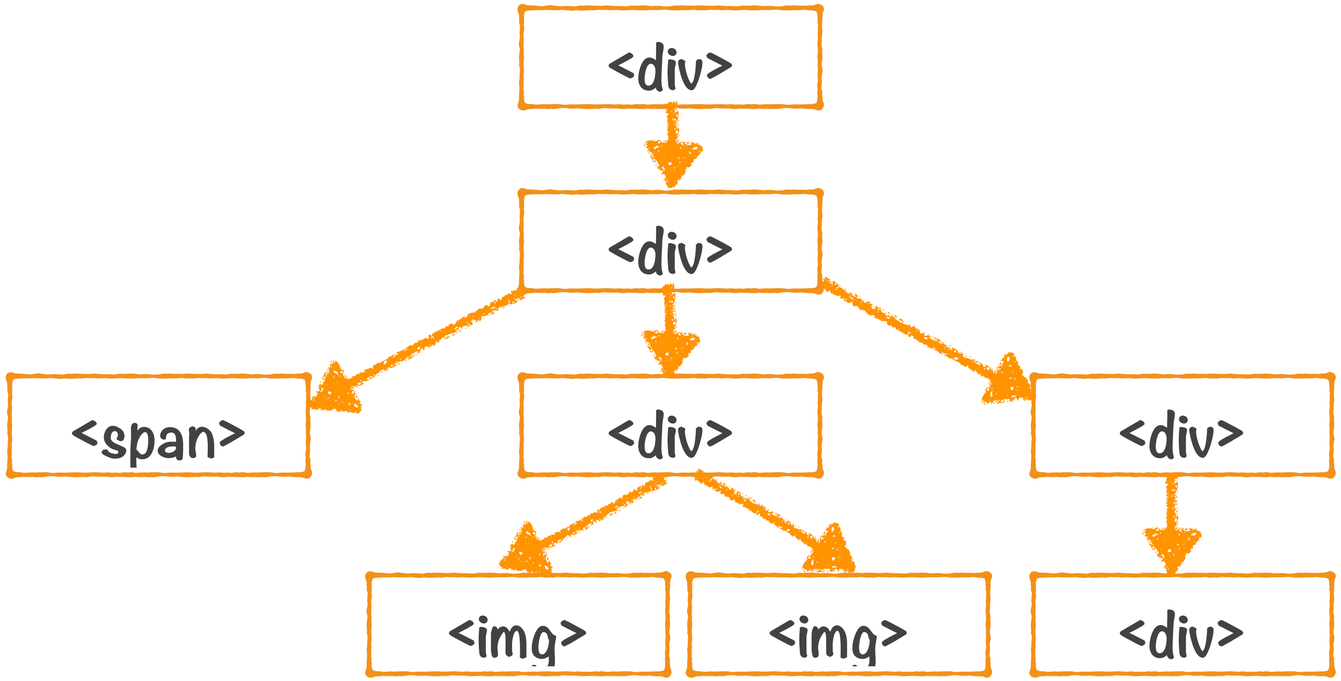
The render() function in a component is the root of a tree of elements

DOM Reconciliation



When the state or props change, a new tree is re-rendered

DOM Reconciliation

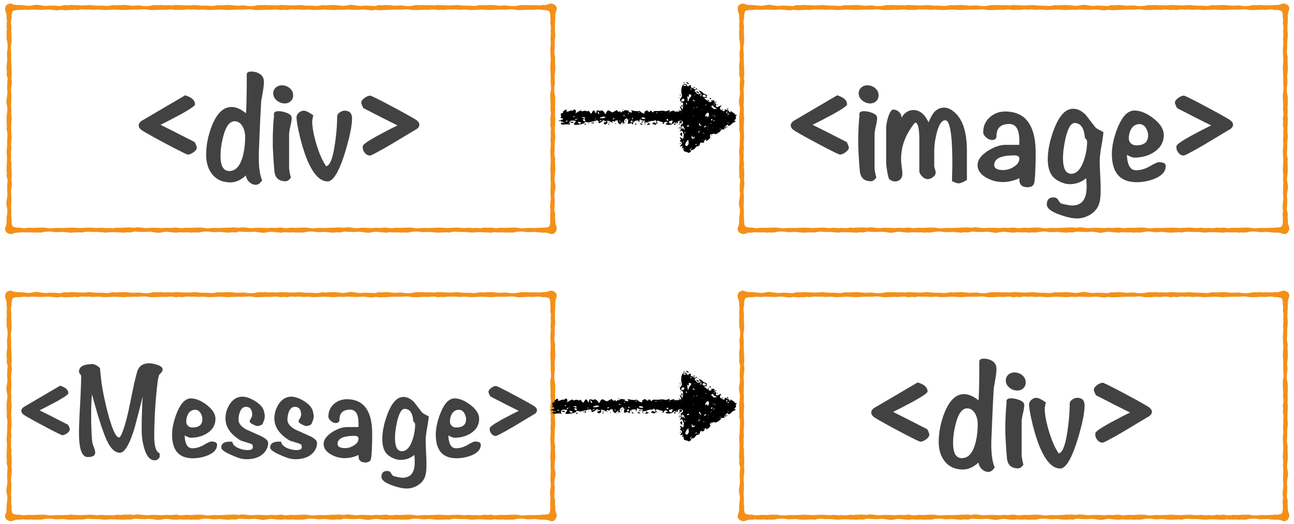


React needs to figure out how to efficiently re-render this tree

DOM Reconciliation

efficiently re-render this tree

1. If the root element is different, then rebuild the entire tree



DOM Reconciliation

efficiently re-render this tree

1. If the root element is different, then rebuild the entire tree

All old nodes are unmounted and destroyed

All old state is lost

DOM Reconciliation

efficiently re-render this tree

2. If the root DOM element is the same but with different attributes

```
<div style={{color: 'red', fontWeight: 'bold'}} />
```

```
<div style={{color: 'green', fontWeight: 'bold'}} />
```

efficiently re-render this tree

2. If the root DOM element is the same but with different attributes

Diff the attributes which have changed and only update those

efficiently re-render this tree

3. Same component with different state and props

<Box message={“Hello there!”} />

<Box message={“How are you?”} />

efficiently re-render this tree

3. Same component with different state and props

The component instance says the same - state is preserved across renderings

efficiently re-render this tree

3. Same component with different state and props

The render() function is called on the component

DOM Reconciliation

efficiently re-render this tree

1. If the root element is different, then rebuild the entire tree

2. If the root DOM element is the **same** but with different attributes

3. **Same** component with different state and props

DOM Reconciliation

efficiently re-render this tree

1. If the root element is different, then rebuild the entire tree
2. If the root DOM element is the same but with different attributes
3. Same component with different state and props

Recurse and apply these same rules to every child of the current node

DOM Reconciliation

efficiently re-render this tree

A special case when children are a collection of the same kind of element

DOM Reconciliation

efficiently re-render this tree

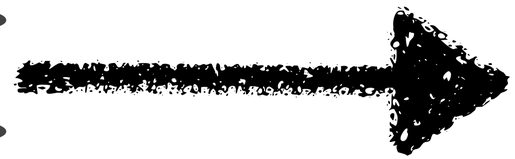
A special case when children are a collection of the same kind of element

```
<ul>
  <li>Obama</li>
  <li>Trump</li>
</ul>
```

DOM Reconciliation

efficiently re-render this tree

```
<ul>
   <li>Obama</li>
   <li>Trump</li>
</ul>
```



```
<ul>
   <li>Bush</li>
   <li>Obama</li>
   <li>Trump</li>
</ul>
```

DOM Reconciliation

efficiently re-render this tree

```
<ul>
   <li>Obama</li>
   <li>Trump</li>
</ul>
```

```
<ul>
   <li>Bush</li>
   <li>Obama</li>
   <li>Trump</li>
</ul>
```

An inefficient re-render can end up changing every child

DOM Reconciliation

efficiently re-render this tree

```
<ul>
   <li>Obama</li>
   <li>Trump</li>
</ul>
```

```
<ul>
   <li>Bush</li>
   <li>Obama</li>
   <li>Trump</li>
</ul>
```

If you use keys to identify these children uniquely

then React will use keys to optimize rendering

DOM Reconciliation

efficiently re-render this tree

```
<ul>
  <li key="2008">Obama</li>
  <li key="2016">Trump</li>
</ul>
```

```
<ul>
  <li key="2000">Bush</li>
  <li key="2008">Obama</li>
  <li key="2016">Trump</li>
</ul>
```

React will use these keys to see which elements are the same

DOM Reconciliation

efficiently re-render this tree

```
<ul>
  <li key="2008">Obama</li>
  <li key="2016">Trump</li>
</ul>
```

```
<ul>
  <li key="2000">Bush</li>
  <li key="2008">Obama</li>
  <li key="2016">Trump</li>
</ul>
```

It will not mutate unchanged elements!