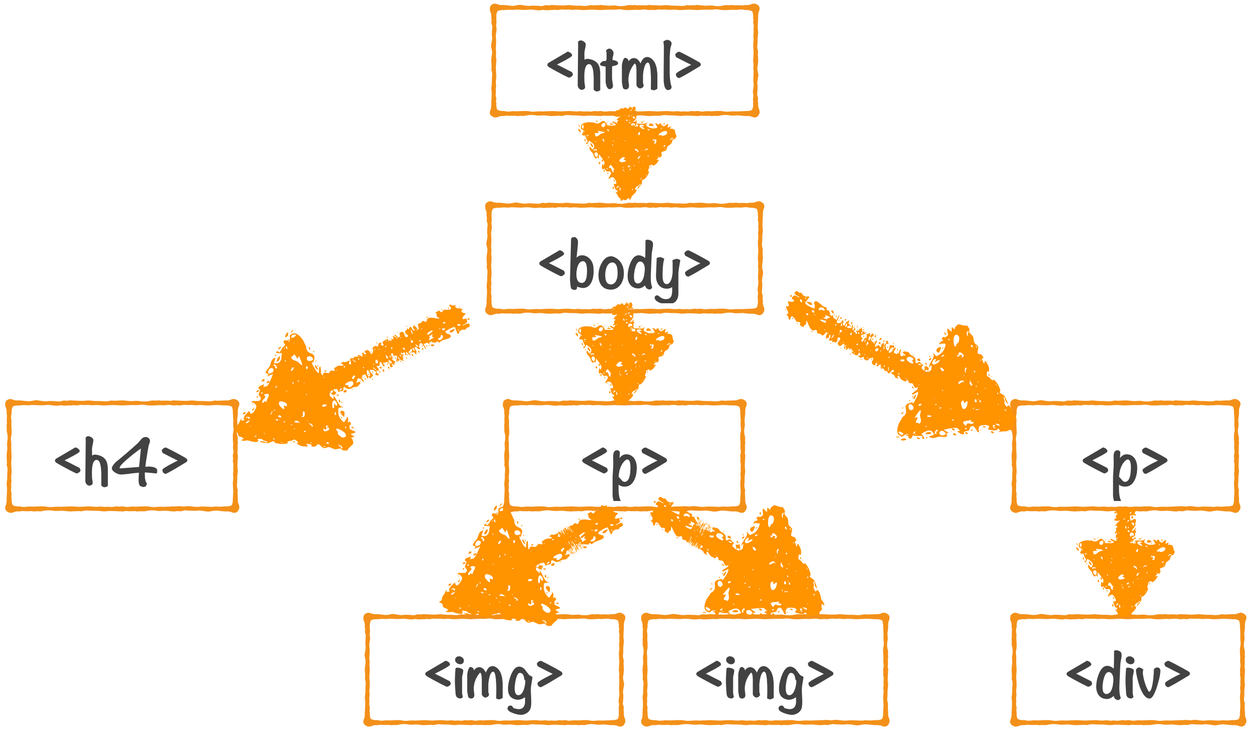
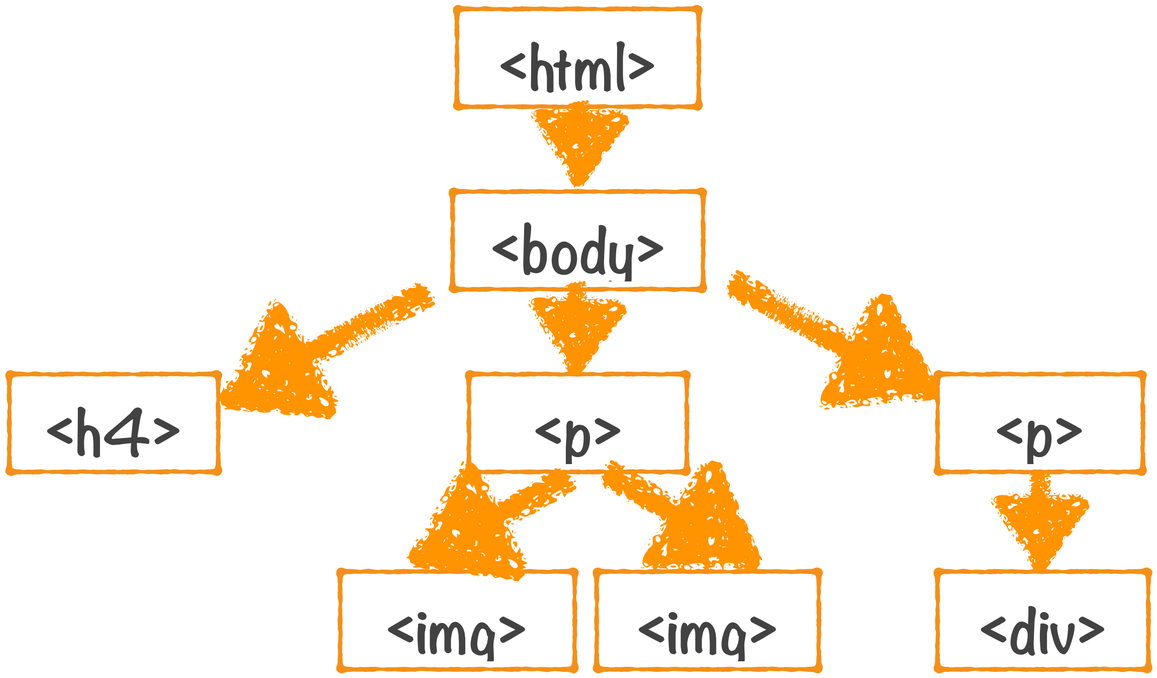
REACT JS

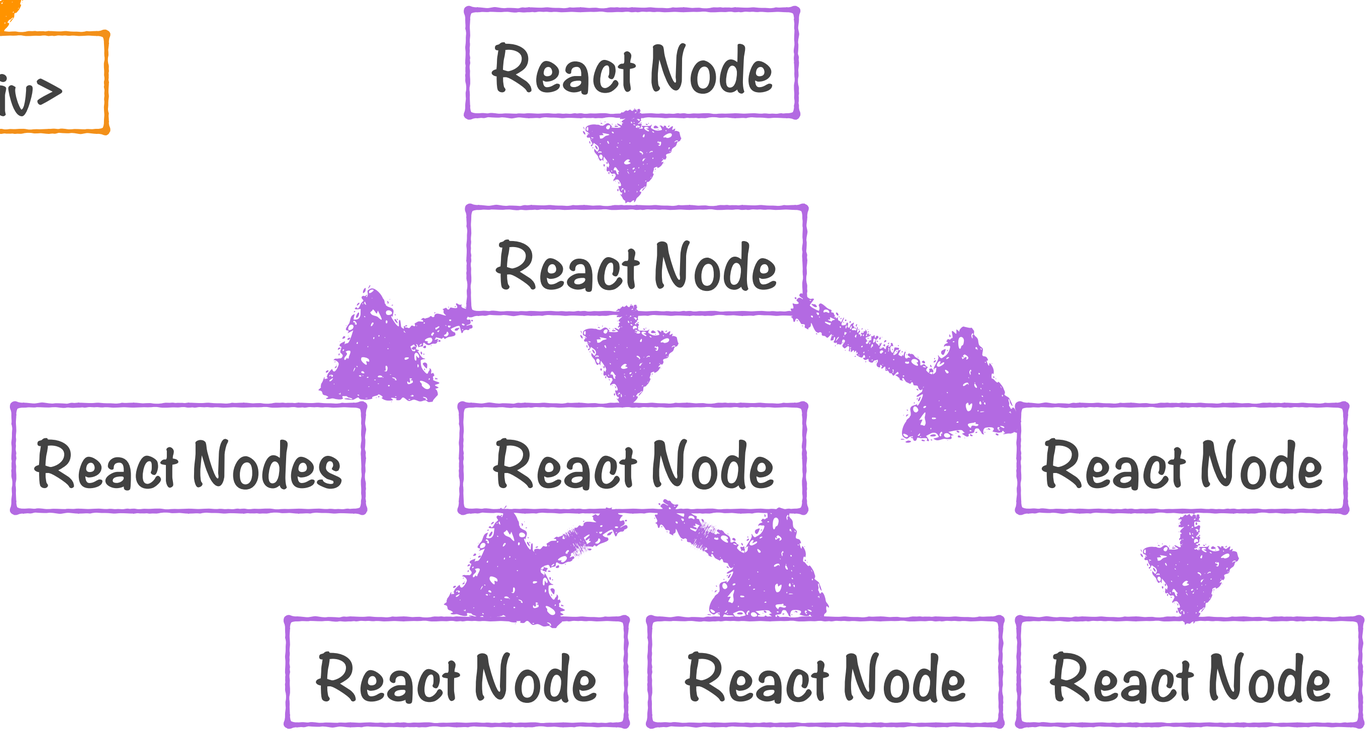
Terms in React



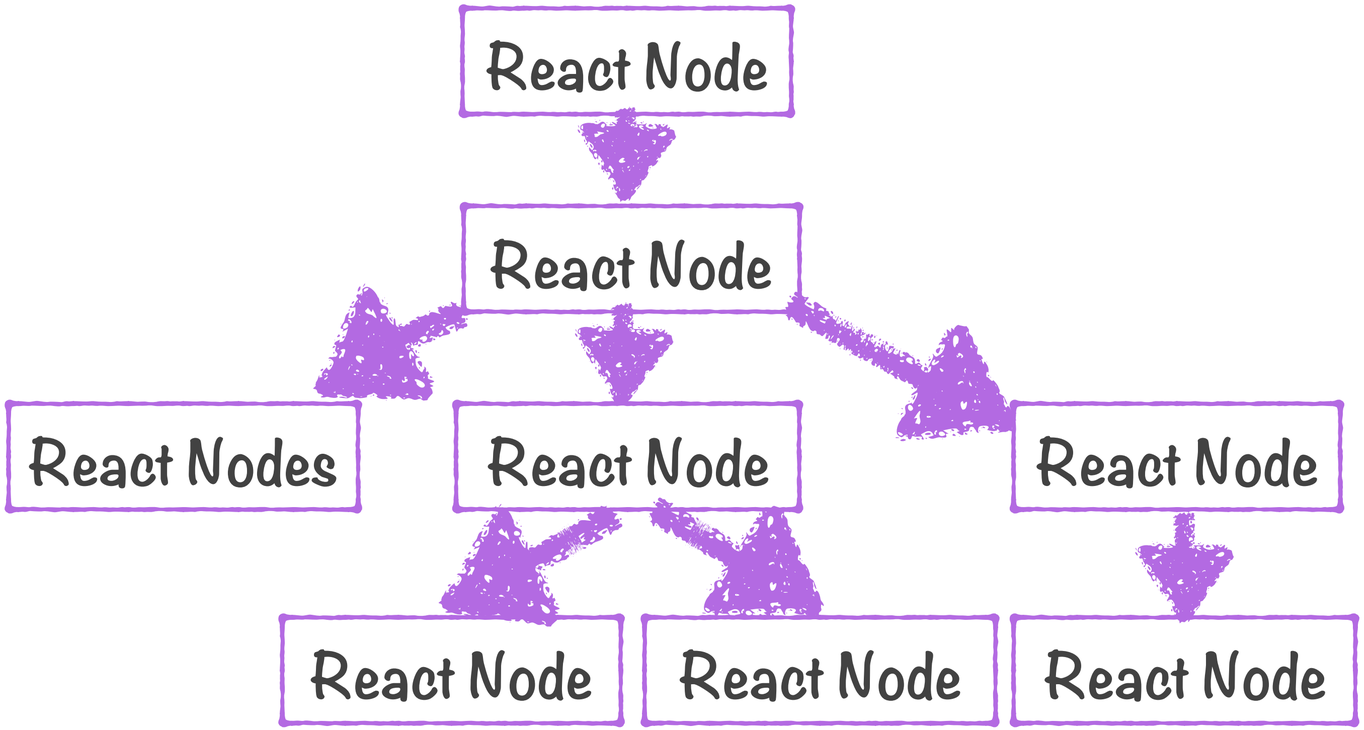
Every web page is made up of a DOM hierarchy



React creates an in-memory representation of this: the virtual DOM

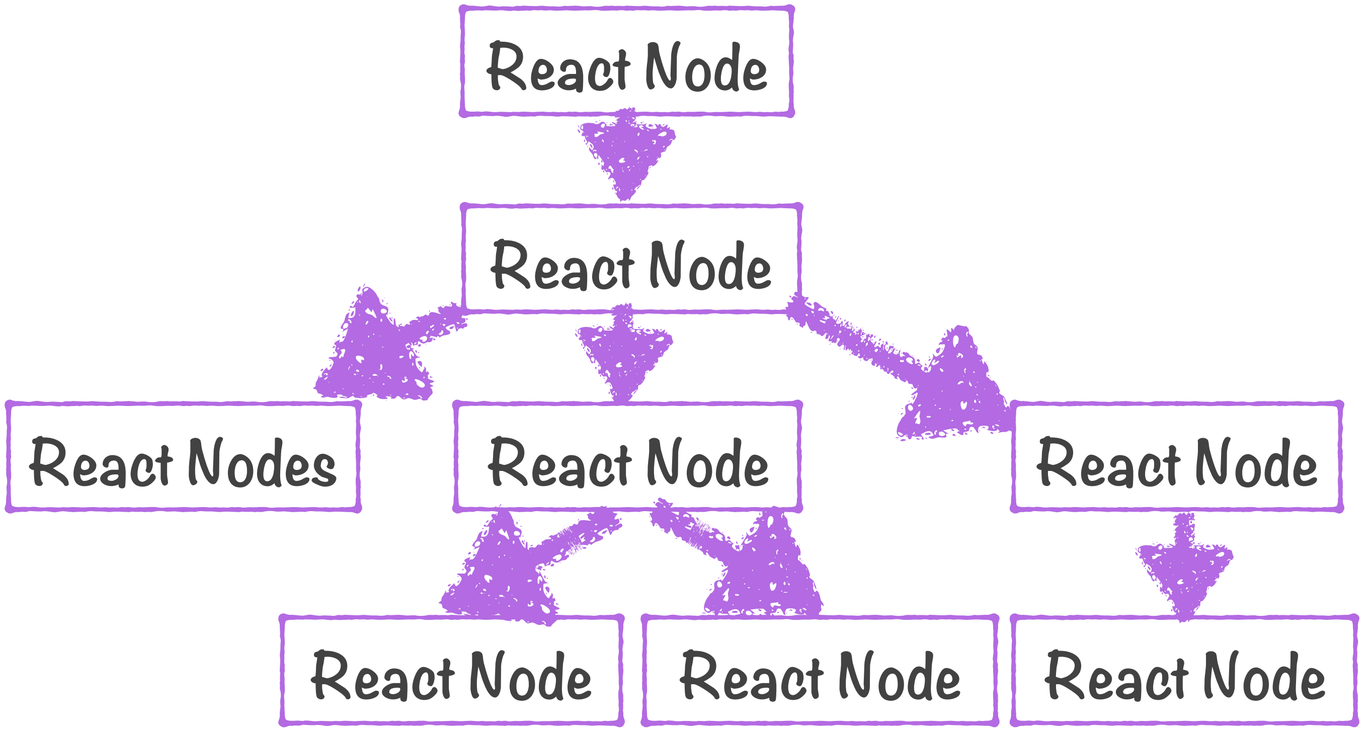


Terms in React



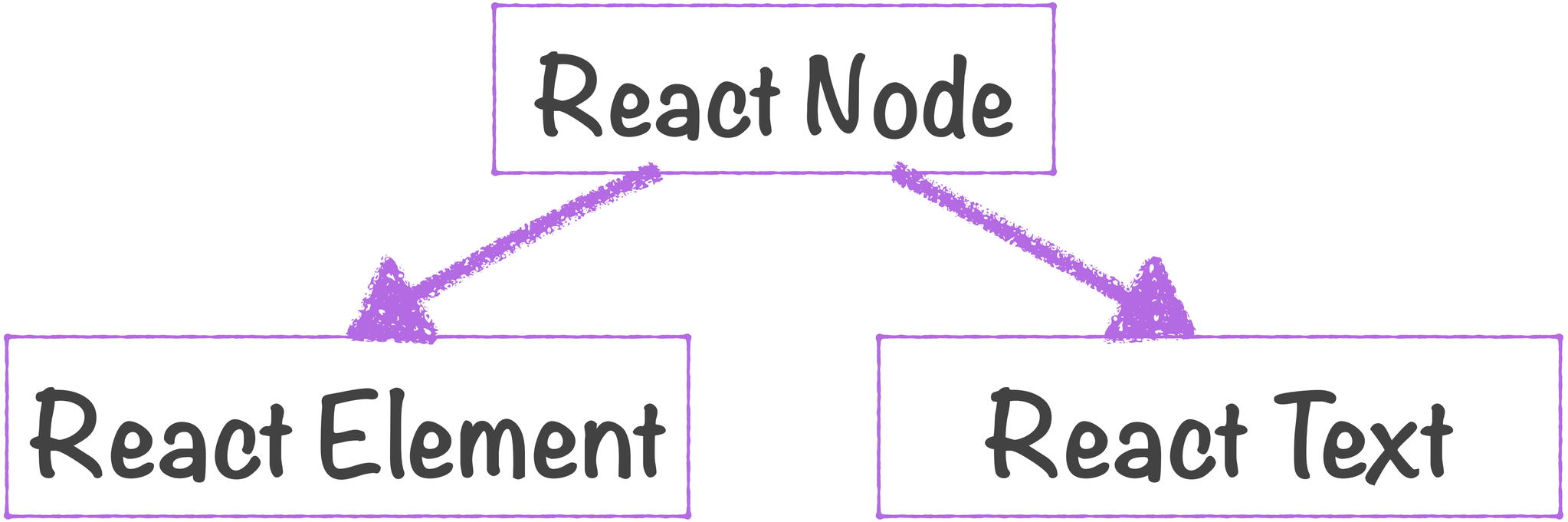
The virtual DOM is made up of React Nodes

Terms in React



The React Node is the building block for the virtual DOM

Terms in React



A React Node can be of two types

Terms in React

React Element

The primary node type in React

Terms in React

React Element

React Node

React Text

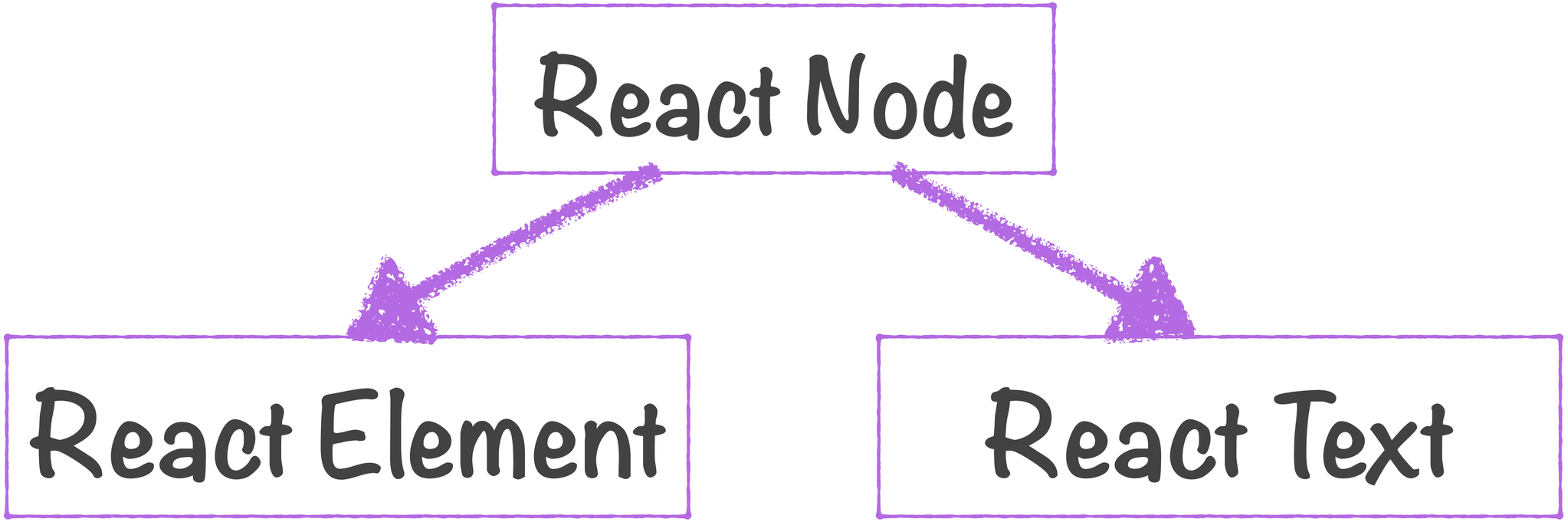
Represents a DOM element

Terms in React

React Element

It is a light, stateless, immutable virtual element

Terms in React



A React Node can be of two types

Terms in React

React Text

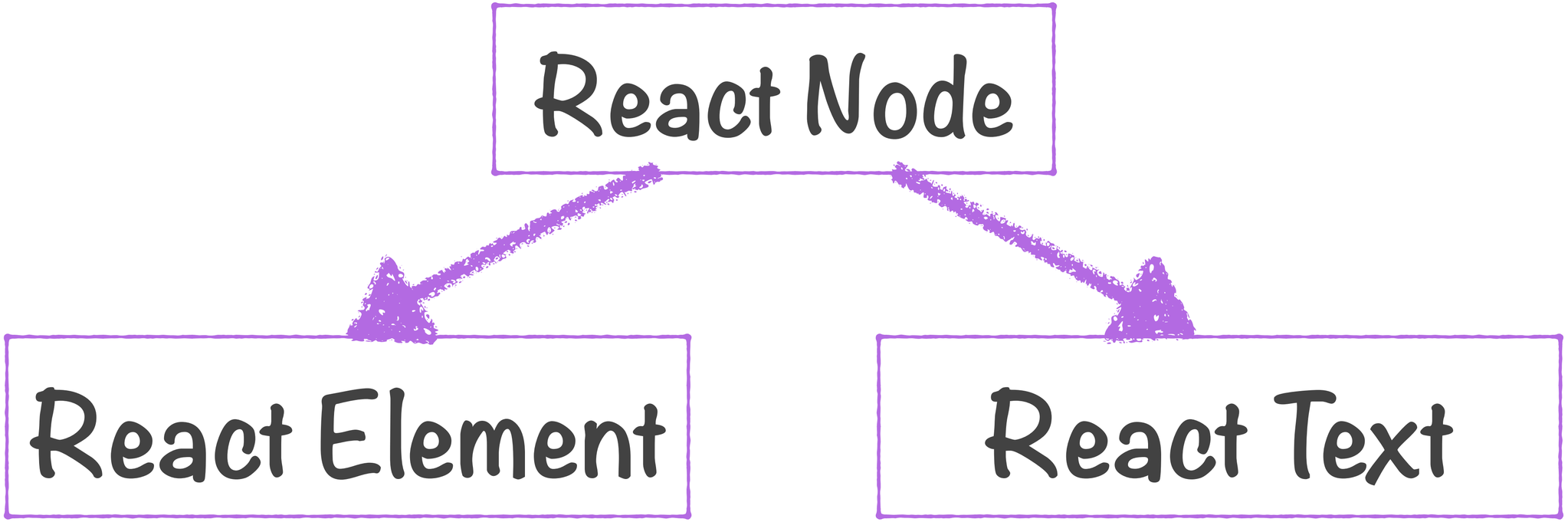
This represents text content - it can be a string or a number

Terms in React

React Text

It is a virtual representation of a Text Node in the DOM

Terms in React

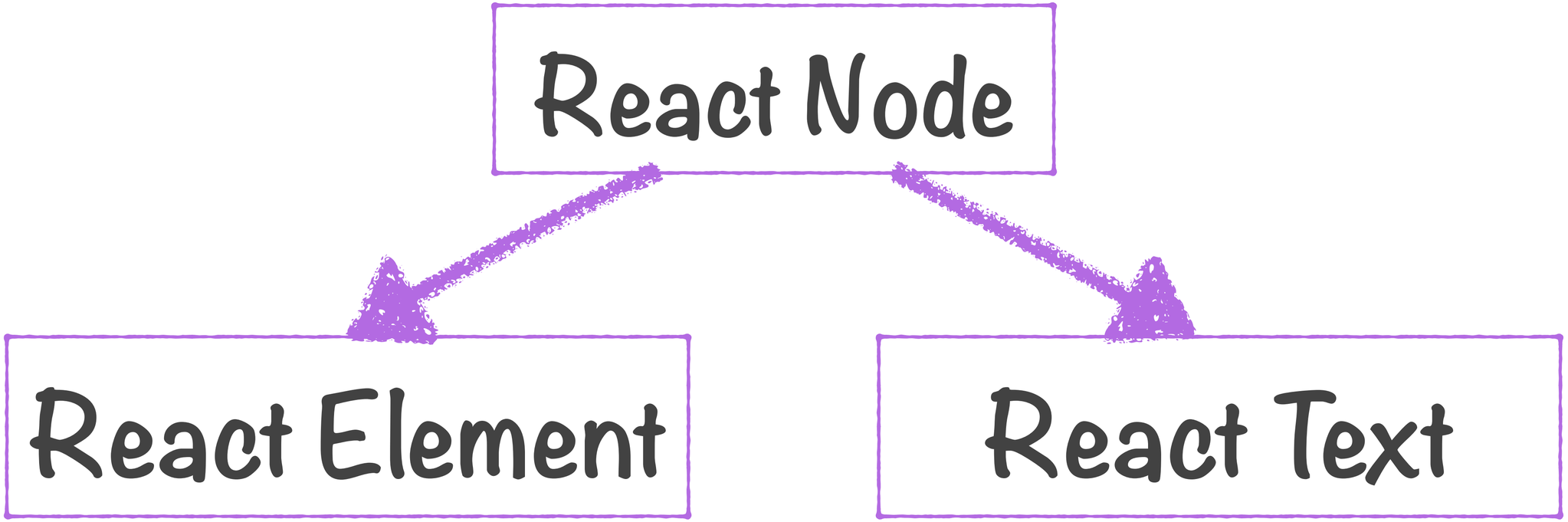


How are you?

React Element

How are you?

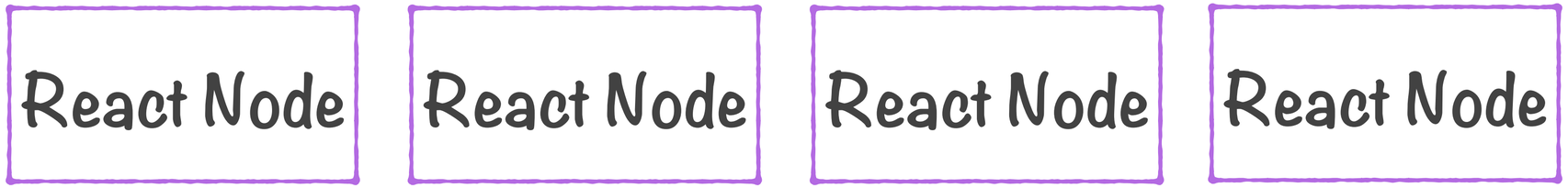
Terms in React



How are you?

React Text

How are you?



An array of React Nodes is a React Fragment

Terms in React

React Node

React Element

React Text

React Fragment

EXAMPLE 3

FactoryFunctions.html

REACT JS

JSX

JSX

```
var listItem1 = React.DOM.li({key: 'item-1'}, 'Albus Dumbledore');
var listItem2 = React.DOM.li({key: 'item-2'}, 'Dolores Umbridge');
var listItem3 = React.DOM.li({key: 'item-3'}, 'Severus Snape');
var listItem4 = React.DOM.li({key: 'item-4'}, 'Minerva McGonagall');

var reactFragment = [listItem1, listItem2, listItem3, listItem4];

var listElement = React.DOM.ul({className: 'container'}, reactFragment);

ReactDOM.render(listElement, document.getElementById('react-app'));
```

Setting up elements in this manner is pretty tedious

JSX

```
var listItem1 = React.DOM.li({key: 'item-1'}, 'Albus Dumbledore');
var listItem2 = React.DOM.li({key: 'item-2'}, 'Dolores Umbridge');
var listItem3 = React.DOM.li({key: 'item-3'}, 'Severus Snape');
var listItem4 = React.DOM.li({key: 'item-4'}, 'Minerva McGonagall');

var reactFragment = [listItem1, listItem2, listItem3, listItem4];

var listElement = React.DOM.ul({className: 'container'}, reactFragment);

ReactDOM.render(listElement, document.getElementById('react-app'));
```

It's hard to read, hard to see what the actual HTML structure is

JSX

```
var listItem1 = React.DOM.li({key: 'item-1'}, 'Albus Dumbledore');
var listItem2 = React.DOM.li({key: 'item-2'}, 'Dolores Umbridge');
var listItem3 = React.DOM.li({key: 'item-3'}, 'Severus Snape');
var listItem4 = React.DOM.li({key: 'item-4'}, 'Minerva McGonagall');

var reactFragment = [listItem1, listItem2, listItem3, listItem4];

var listElement = React.DOM.ul({className: 'container'}, reactFragment);

ReactDOM.render(listElement, document.getElementById('react-app'));
```

Code like this is a nightmare to maintain

This why React uses an XML like syntax to define HTML elements

JSX

Javascript Syntax eXtension

JSX

Javascript Syntax eXtension

This is a concise and familiar syntax to define tree structures with attributes

JSX

Javascript Syntax eXtension

concise and familiar

Large trees representing the DOM are easier to read and maintain

JSX

Javascript Syntax eXtension

concise and familiar

easier to read and maintain

Maintains the semantics of Javascript

JSX

Javascript Syntax eXtension

concise and familiar

easier to read and maintain

semantics of Javascript

What does JSX look like?

very similar to HTML

```
var headerElement = <h3> Hello world! </h3>;
ReactDOM.render(headerElement, document.body);
```

But embedded in a JS file or <script> tag

JSX

```
var headerElement = <h3> Hello world! </h3>;
ReactDOM.render(headerElement, document.body);
```

Notice that HTML elements are specified within Javascript - this is a snippet of Javascript

JSX

```
var headerElement = <h3> Hello world! </h3>;
ReactDOM.render(headerElement, document.body);
```

The browser does not understand JSX!

JSX

```
var headerElement = <h3> Hello world! </h3>;
ReactDOM.render(headerElement, document.body);
```

The browser does not understand JSX!

Which means that this bit of code cannot run directly on the browser

JSX

```
var headerElement = <h3> Hello world! </h3>;
ReactDOM.render(headerElement, document.body);
```

This has to be transformed to Javascript before the browser can understand and render the element

JSX

```
var headerElement = <h3> Hello world! </h3>;
ReactDOM.render(headerElement, document.body);
```

```
var headerElement = React.createElement(
  "h3",
  null,
  " Hello world! "
);
ReactDOM.render(headerElement, document.body);
```

JSX

```
var headerElement = <h3> Hello world! </h3>;
ReactDOM.render(headerElement, document.body);
```

This transformation is done by a Javascript compiler called Babel

```
var headerElement = React.createElement(
  "h3",
  null,
  " Hello world! "
);
ReactDOM.render(headerElement, document.body);
```

This transformation is done by a
Javascript compiler called Babel

Run Babel on your Javascript code which contains JSX

To get pure Javascript!

There is another reason to use Babel with React

There is another reason
to use Babel with React

React is compatible with new features
of ECMAScript6 or ES6

There is another reason
to use Babel with React

React is compatible with new features
of ECMAScript6 or ES6

Not all browsers support ES6!

There is another reason
to use Babel with React

Not all browsers support ES6!

Babel can be used to convert ES6 code to
ES5 so it is compatible with all browsers

Babel can be used to convert ES6 code to ES5 so it is compatible with all browsers

We'll see this later on this course

REACT JS

Babel REPL Environment

Babel REPL Environment

Babel offers an in-browser Read, Evaluate, Print, Loop (REPL) environment where you can try stuff out

Babel REPL Environment

Just type in JSX and see what the React Javascript equivalent is!

This should give you an idea of how JSX can be written within Javascript code

Babel Demo Video

REACT JS

Babel for transforming JSX to Javascript

Babel for transforming JSX to Javascript

This transformation can be done in 2 ways

on the fly, in the browser

rapid prototyping during development

at build time

production environments

Babel for transforming JSX to Javascript

on the fly, in the browser

rapid prototyping during development

Babel will run on the Javascript just before the browser renders the page

Babel for transforming JSX to Javascript

on the fly, in the browser

rapid prototyping during development

This is inefficient and a major performance hit

This is not acceptable on production systems

Babel for transforming JSX to Javascript

on the fly, in the browser

rapid prototyping during development

This is not acceptable on production systems

On the fly transformation is used during development when you quickly want something working

Babel for transforming JSX to Javascript

This transformation can be done in 2 ways

on the fly, in the browser

rapid prototyping during development

at build time

production environments

Babel for transforming JSX to Javascript

at build time

production environments

This involves transforming JSX code at build time by compiling them using Babel

Babel for transforming JSX to Javascript

at build time

production environments

These pre-built files are served to clients in production

Babel for transforming JSX to Javascript

at build time

production environments

The additional transformation phase when users access the webpage is eliminated

Babel for transforming JSX to Javascript

This transformation can be done in 2 ways

on the fly, in the browser

rapid prototyping during development

at build time

production environments

REACT JS

Standalone Babel For In-Browser Transformation

Let's see how you can use Babel for prototyping during development

StandaloneBabel video

EXAMPLE 4

ElementsWithJSX.html

REACT JS

Components

Components

We've user React elements along with JSX to build up static web pages

That's pretty useless...
where is the powerful stuff?

Components

That's pretty useless...
where is the powerful stuff?

What we want is the ability to build interactive components - those which **react** to user input

Reactive components are those which change state in response to user and server events

React Node

React Element

React Text

React Fragment

Do not have state!

React Component

= React Element + State Machine

React Component

= React Element + State Machine

Every state in the React Component can be represented by a different React element

EXAMPLE 5

StatelessReactComponent.html

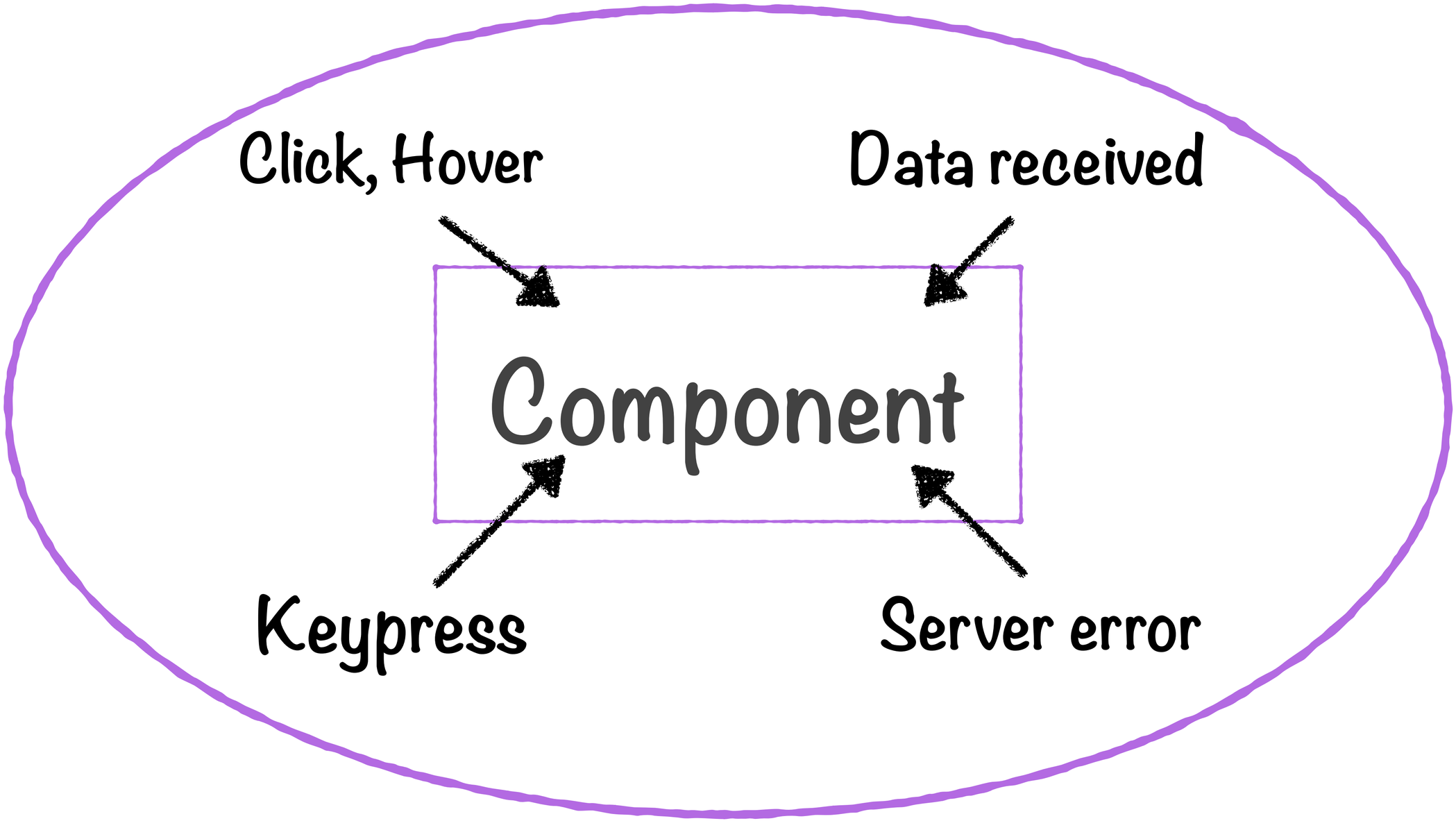
EXAMPLE 6

RenderFunction.html

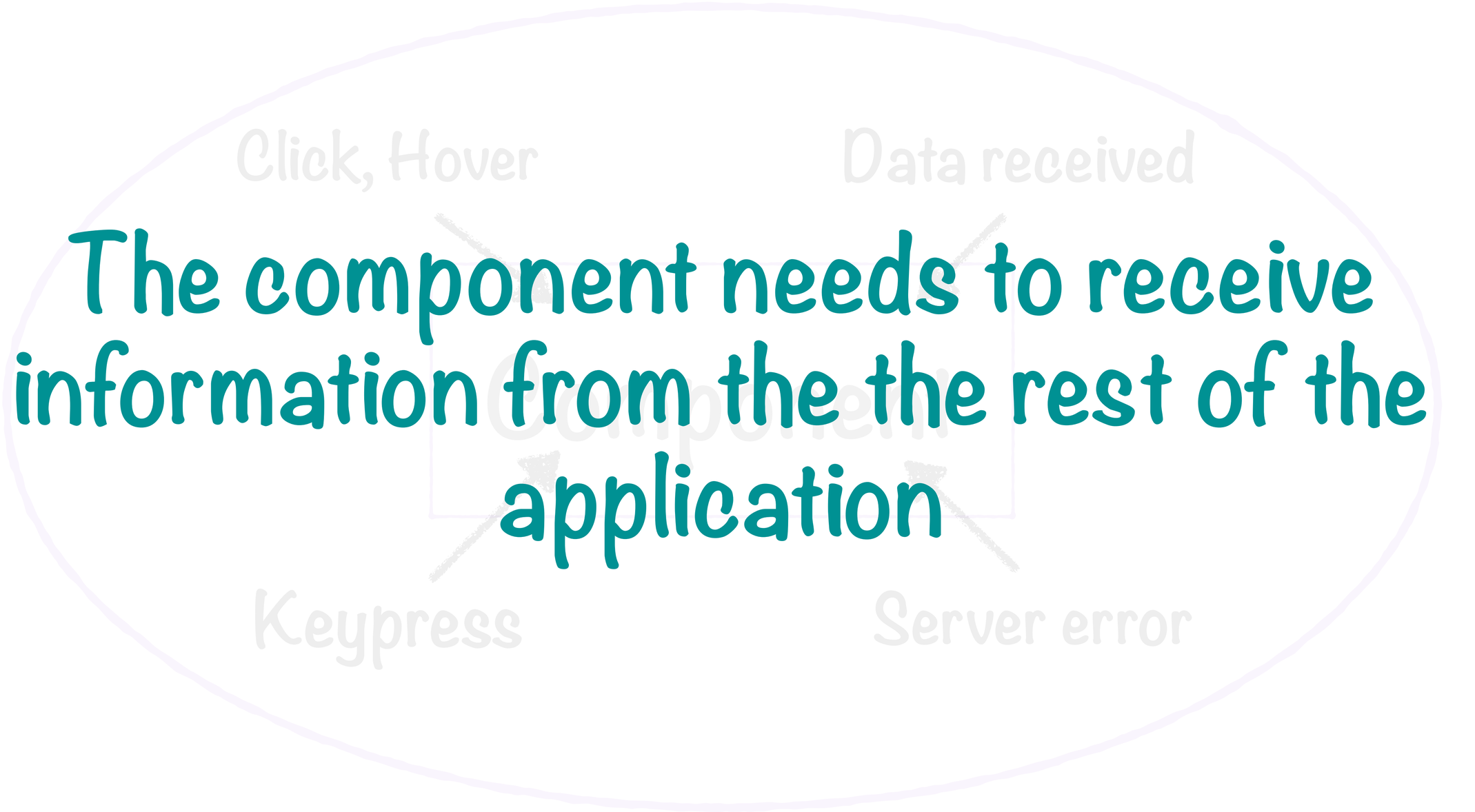
REACT JS

Passing data to React components

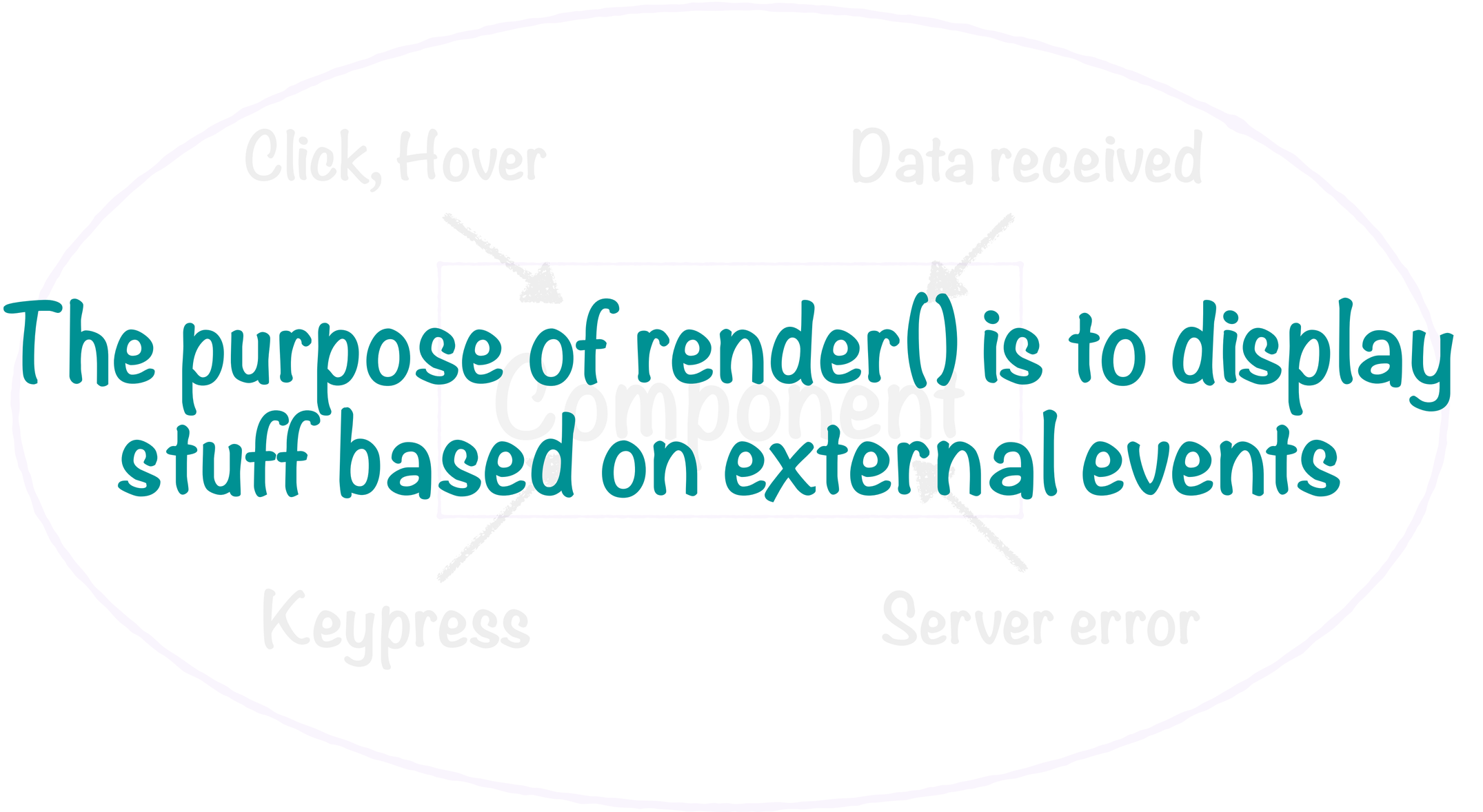
Passing data to React components



Passing data to React components



The component needs to receive information from the the rest of the application



The purpose of render() is to display stuff based on external events

There are 2 ways to pass data to React component

props

State

props

Read-only data passed from parent component to child components

Read-only data

props

Considered immutable and should not be changed by child components

Read-only data

immutable

props

Accessed using this.props within the render() function

There are 2 ways to pass data to React component

props

State

State

Stores data that is private to the component

private data

State

What the component renders changes based on its state

private data

determines
component rendering

State

The component can update
the state

private data

determines component rendering

component changes state

State

The component re-renders itself based on state changes

There are 2 ways to pass data to React component

props

State

EXAMPLE 7

Props.html

EXAMPLE 8

NestedElementProps.html

EXAMPLE 9

TransferringProps.html

EXAMPLE 10

TransferringPropsWithSpreadAttributes.html

EXAMPLE 11

DynamicTypesUsingProps.html

EXAMPLE 12

PropsWithTypes.html

EXAMPLE 13

JsxChildren.html

EXAMPLE 14

ExpressionsAsChildren.html

EXAMPLE 15

FunctionsAsChildren.html

REACT JS

State machines

There are 2 ways to pass data to React component

props

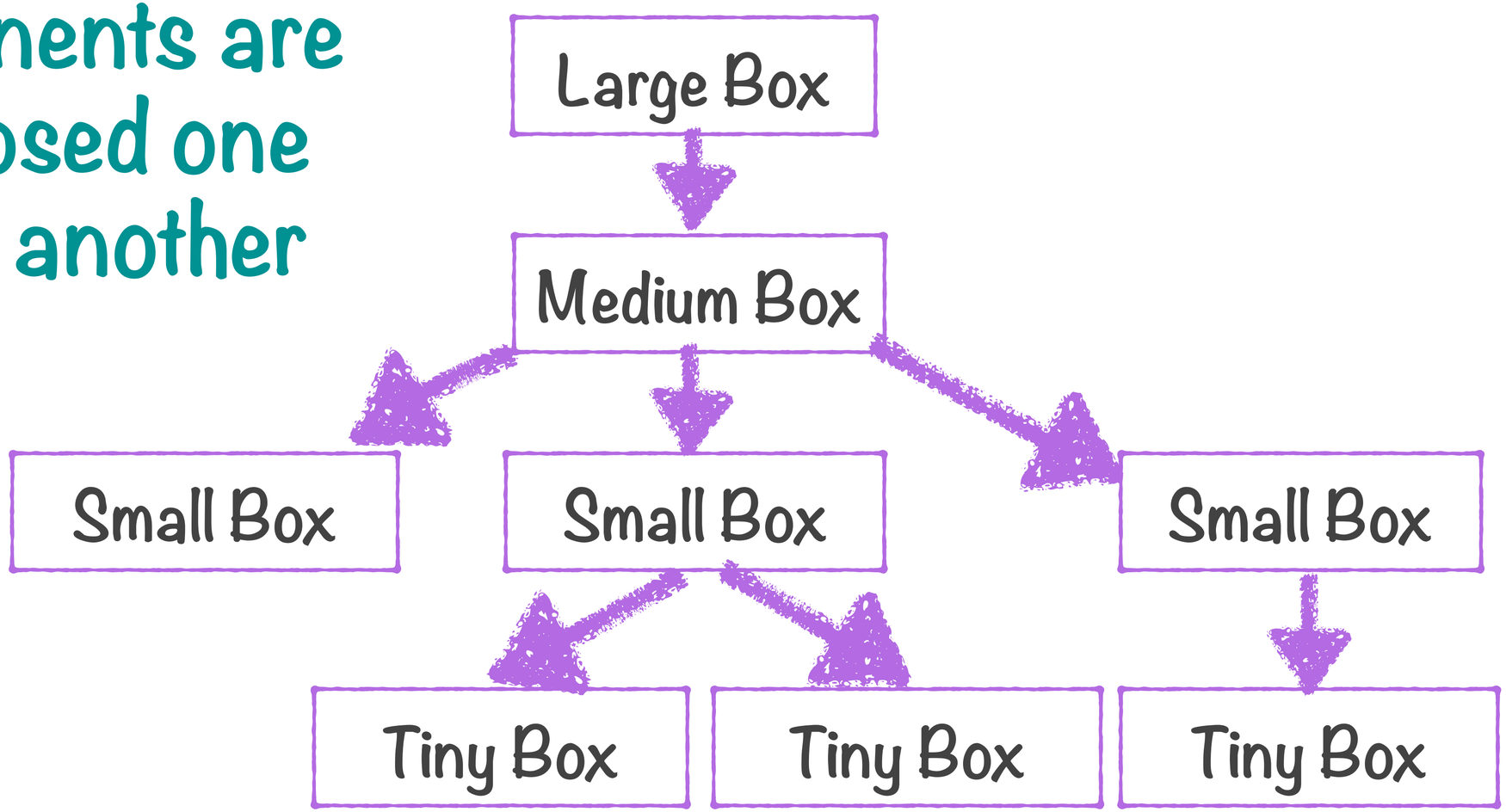
State

State

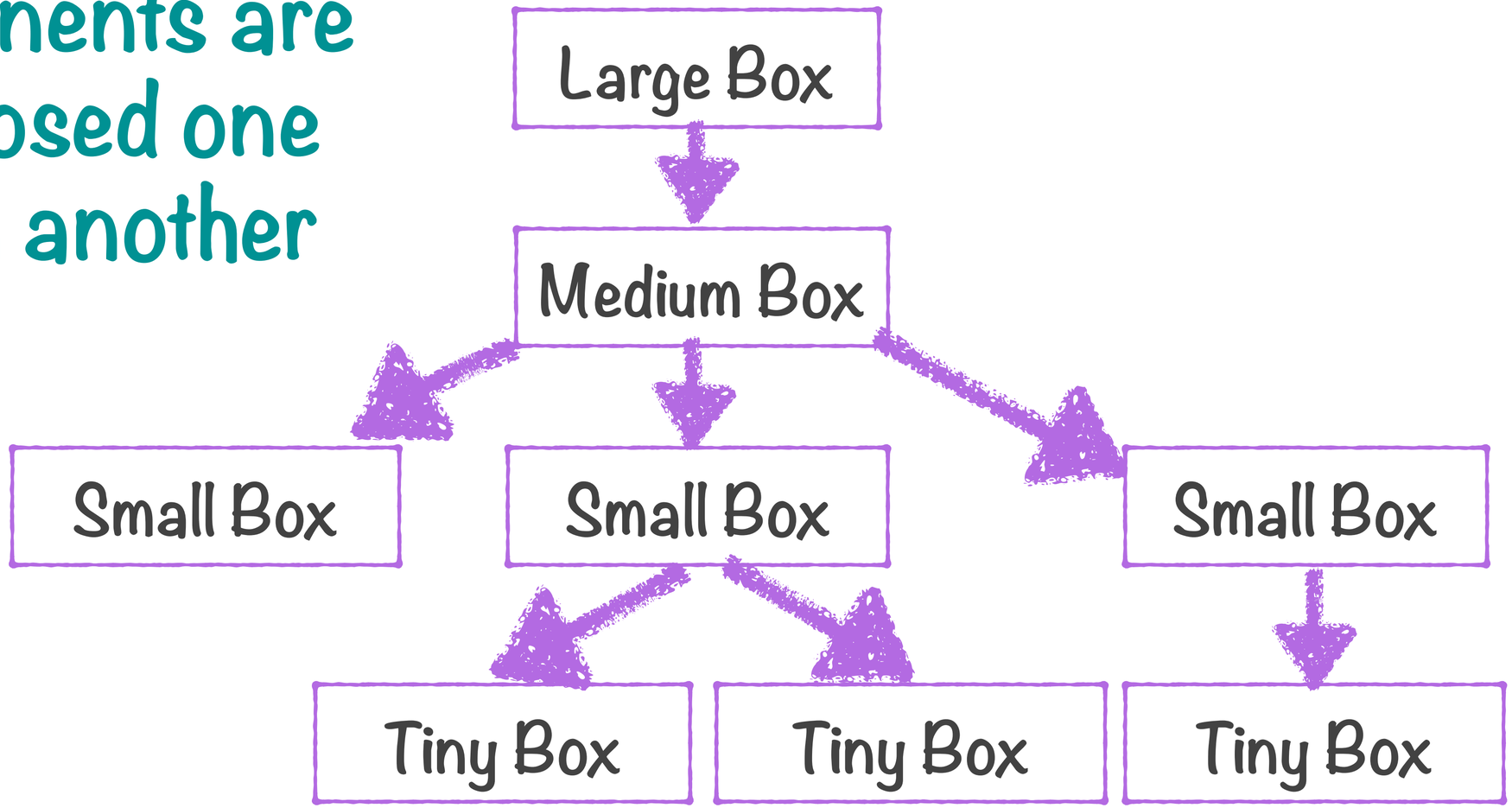
Every React component is a state machine and **can** have state

But it shouldn't unless it has to!

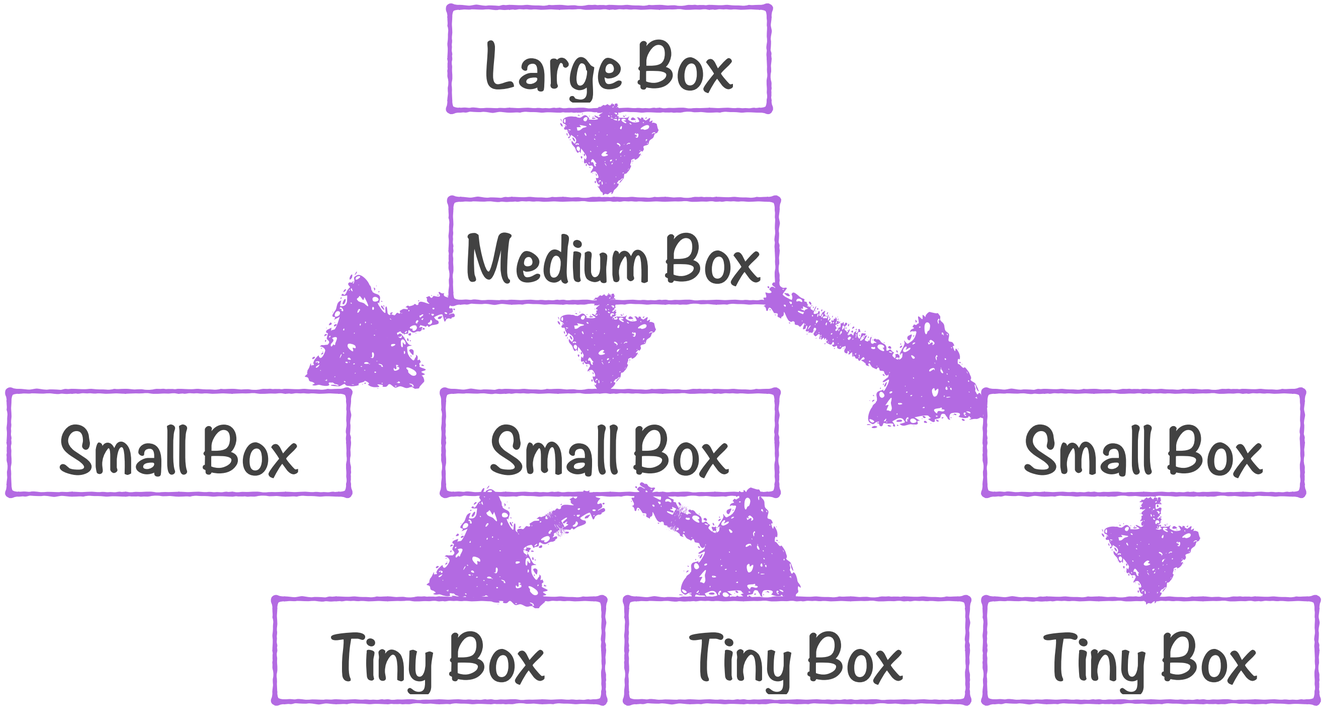
React components are composed one within another



React components are composed one within another

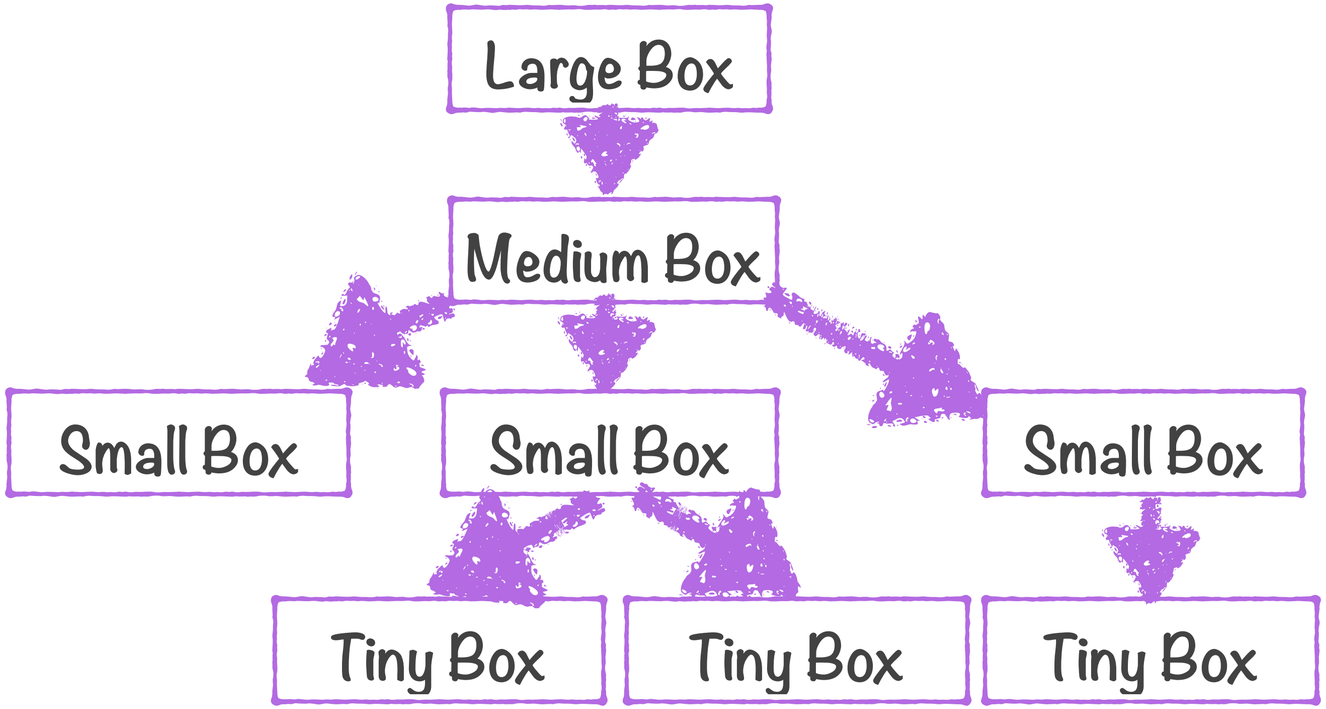


State machines



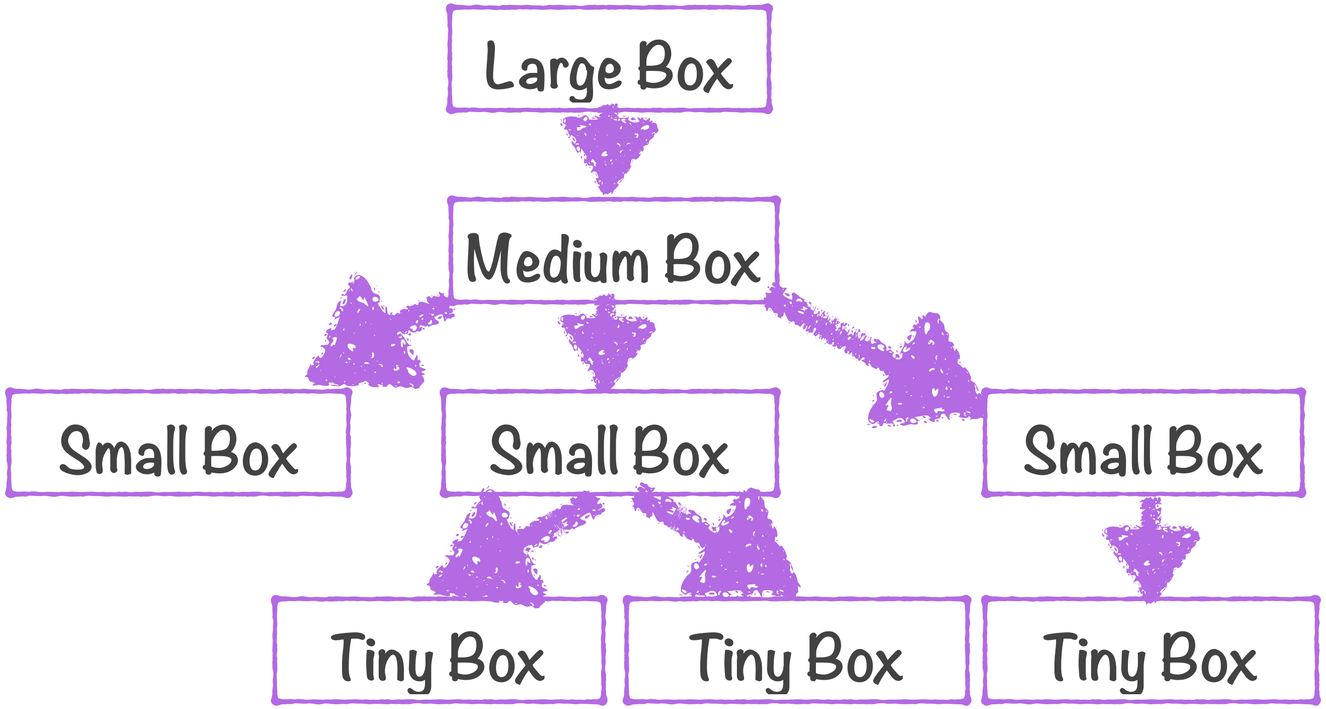
Most React components focus on how to render data

State machines

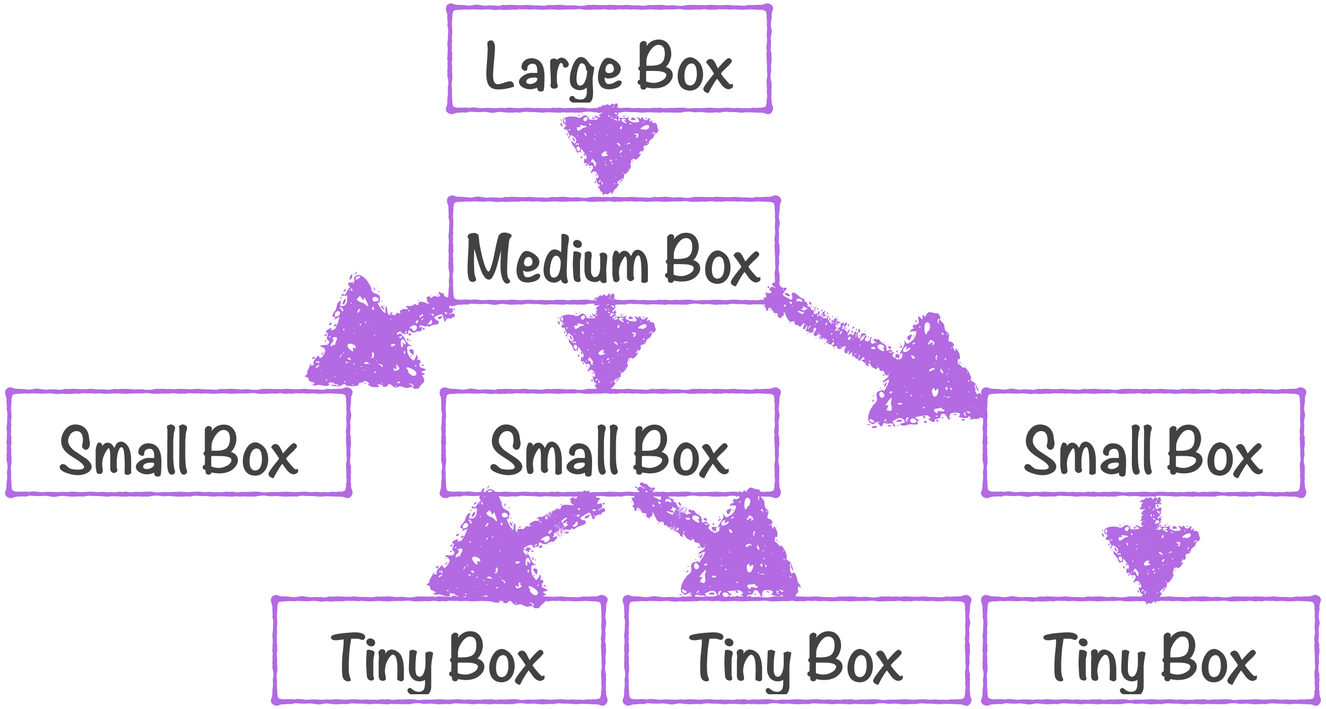


They receive state via **props** and render the display accordingly

State machines

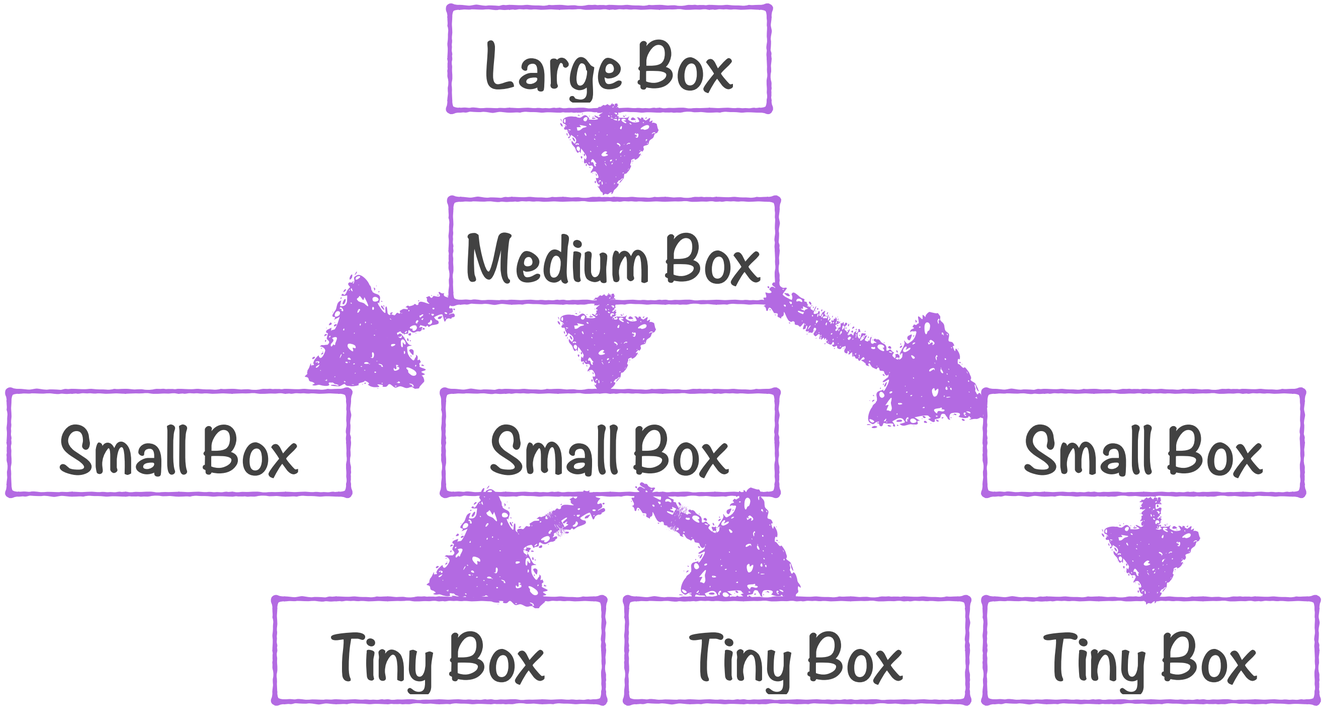


Sparingly choose a few components to have state



These components should ideally be at the top of the hierarchy

State machines



Higher level components calculate state and pass this information using props

EXAMPLE 16

State.html

EXAMPLE 17

UpdateState.html

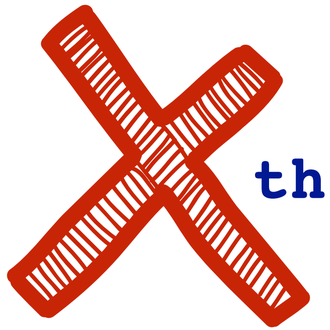
EXAMPLE 18

AccessingPreviousState.html

REACT JS

Properties of state

State should only be modified by calling setState()



```
this.state.textColor = 'white';
```

```
this.setState({
    textColor: ‘white'
});
```

The view is re-rendered only when setState() is called

```
this.setState({
    textColor: ‘white'
});
```

State and props updates might be asynchronous

```
this.setState({
    counter: this.state.counter + this.props.increment
});
```

```
this.setState(function(prevState, props) {
    counter: prevState.counter + props.increment
});
```

Use a function which passes in the previous state and the last value props value as arguments

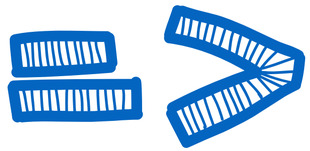
```
this.setState(function(prevState, props) {
    return {
      counter: prevState.counter + props.increment
    }
});
```

Arrow notation for specifying the state update function

```
this.setState((prevState, props) => ({
    counter: prevState.counter + props.increment
}));
```

State updates are merged to get the new state

```
{
    propertyA: 'value',
    propertyB: 'value'
}
```

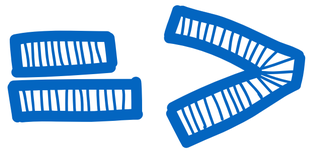


```
this.setState({
    propertyA: 'valueA',
});
```

```
{
    propertyA: 'valueA',
    propertyB: 'value'
}
```

Only properties which are updated change, not others

```
{
    propertyA: 'value',
    propertyB: 'value'
}
```



```
this.setState({
    propertyB: 'valueB',
});
```

```
{
    propertyA: 'value',
    propertyB: 'valueB'
}
```

What should be stored in the state?

Properties of state

What should be stored in the state?

Values which change with user or server events

Values which cause the UI to re-render

Properties of state

What should be stored in the state?

Values which change with user or server events

Values which cause the UI to re-render

Anything else should be removed or moved to props