Day 58

Rickety, Rackety, React

The Coding Bootcamp | June 3, 2016

Component Refresher

A Moment to Ponder…

Instead of separating “layout and logic”,
React JS uses what <u>alternative paradigm</u>?

A Moment to Ponder…

Instead of separating “layout and logic”, **React JS uses what <u>alternative paradigm</u>**?

<u>Components*!</u>

* If you only learn one thing from our lessons on React it should be this.

A Moment to Ponder…

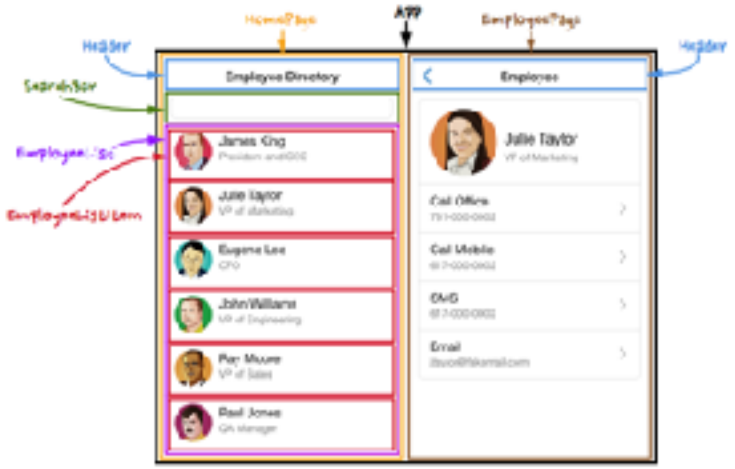
What ***exactly*** are components again?

Power of Components!

By separating elements out into components...

- Layout and Logic are kept bundled together in a self-contained package.
- Components can easily be re-used in various points in the application without needing to be re-coded.
- Components can be more easily tested. (i.e. having one re-usable component means only one UI element needs to be tested).

<u>For complex applications each of these can be critical in finding bugs and saving time.</u>



Power of Components!

A Moment to Ponder…

What method do we use to create a
new component?

A Moment to Ponder…

What method do we use to create a **new component**?

var ClassName* = React.createClass()

** Remember that React components must be capitalized*

A Moment to Ponder…

What method do we use to create a **render our components to the DOM**?

A Moment to Ponder…

What method do we use to create a **render our components to the DOM?**

ReactDOM.render()*

** Note that you will only be rendering a single component into the DOM. Every other component will be a child to that component.*

A Moment to Ponder…

New Question:

Every component **must** contain which method?

A Moment to Ponder…

New Question:

Every component **must** contain which method?

render: function() { }*

** Our render function will define what our component will look like. It will be in JSX syntax*

A Moment to Ponder…

New Question:

How do we deploy a component in our JSX?

A Moment to Ponder…

New Question:

How do we deploy a component in our JSX?

```
<ComponentName />
```

A Moment to Ponder…

New Question:
How would we deploy a component *inside* of another component?

A Moment to Ponder…

New Question:
How would we deploy a component *inside* of another component?

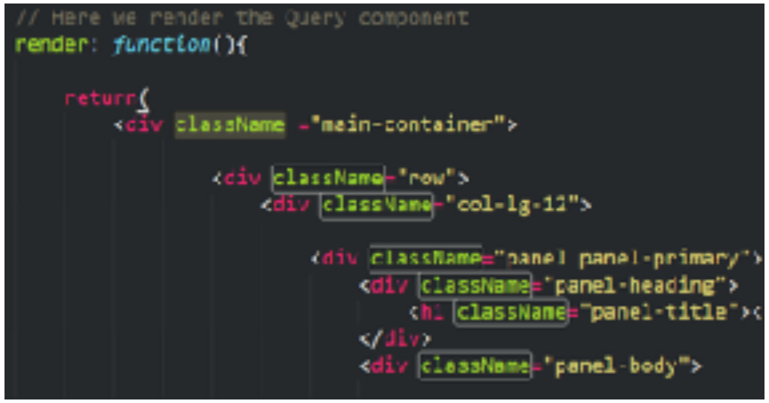
```
<ParentComponent>
                <ComponentName />
</ParentComponent>
```

JSX Gotcha’s

Incorporating Classes

It turns out…

- *You can incorporate classes into JSX*
- *You just need to call them "<u>className</u>"*
- *This is because "class" is a reserved keyword in Javascript*

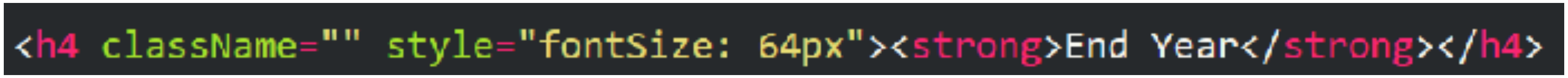


```
// Here we render the Query component
render: function(){

    return(
        <div className ="main-container">

            <div className="row">
                <div className="col-lg-12">

                    <div className="panel panel-primary">
                        <div className="panel-heading">
                            <h1 className="panel-title"><
                        </div>
                        <div className="panel-body">
```

Incorporating Styles

It also turns out…

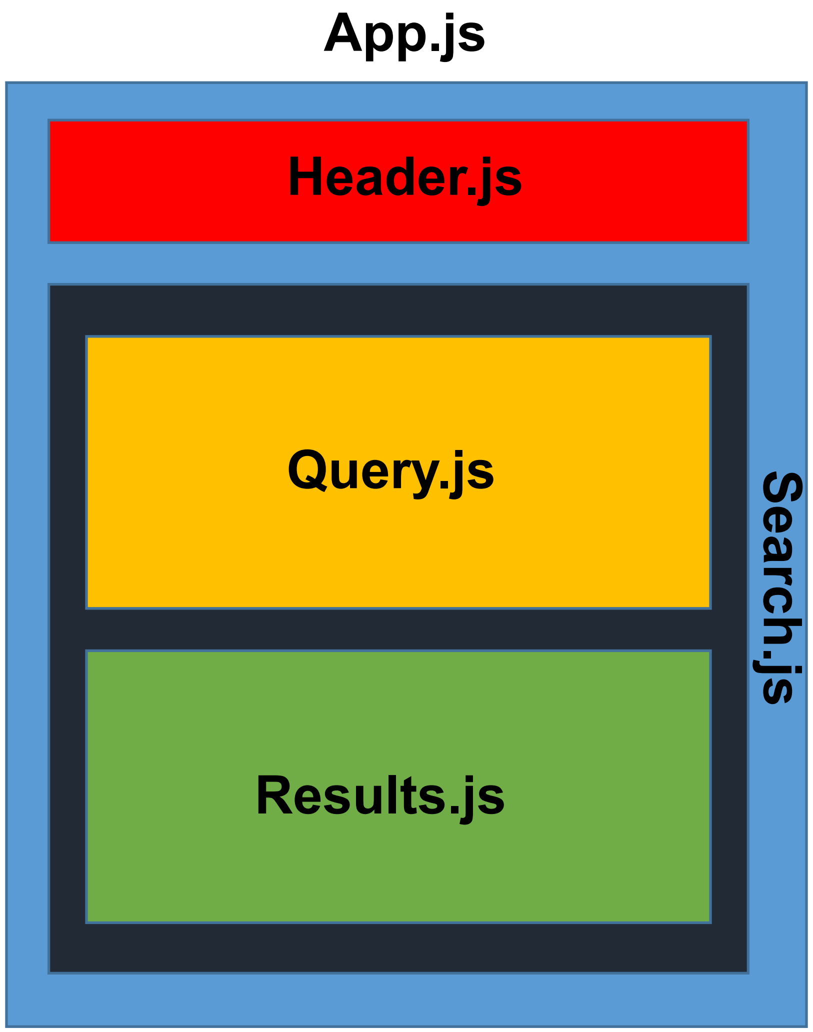
- *You can incorporate CSS styles into JSX*
- *You just need to ditch the hyphen and camelcase the property.*
- *Ex: font-size → fontSize*

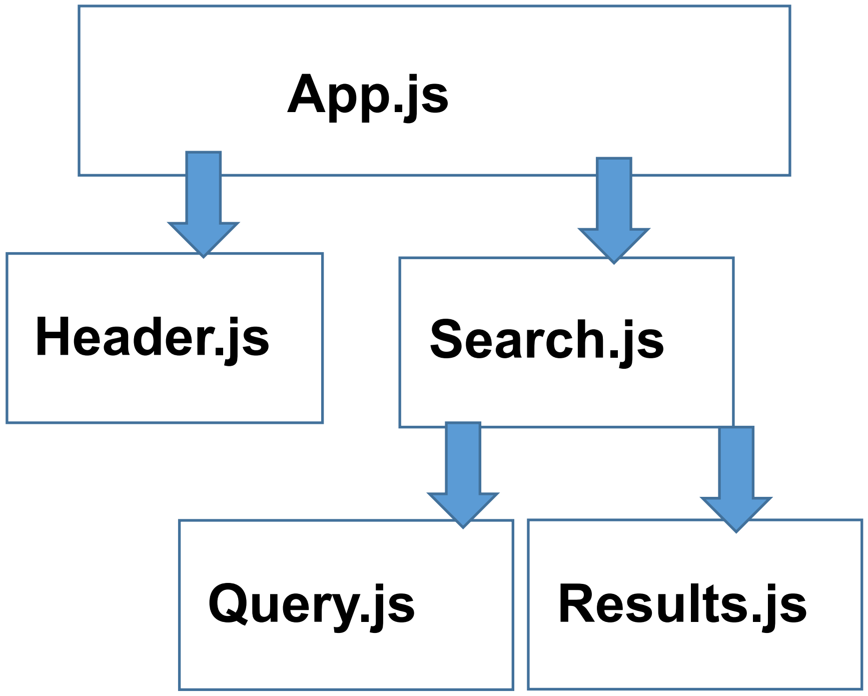


```
<h4 className="" style="fontSize: 64px"><strong>End Year</strong></h4>
```

Component Architecting

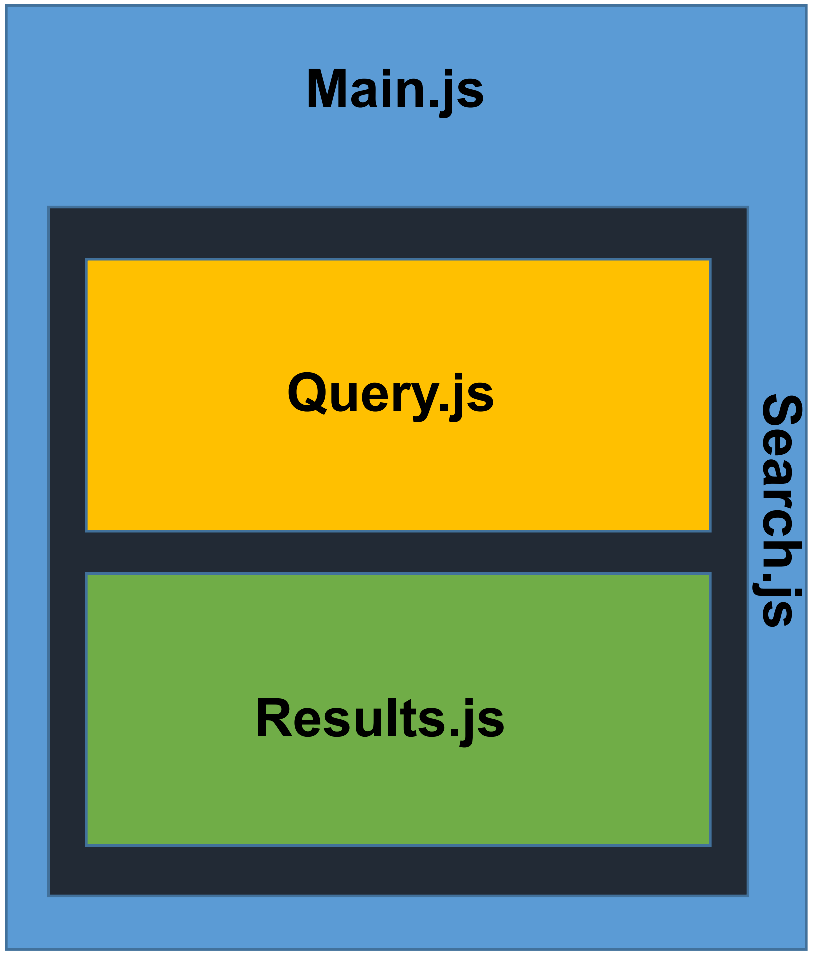
Parent-Child Relationships

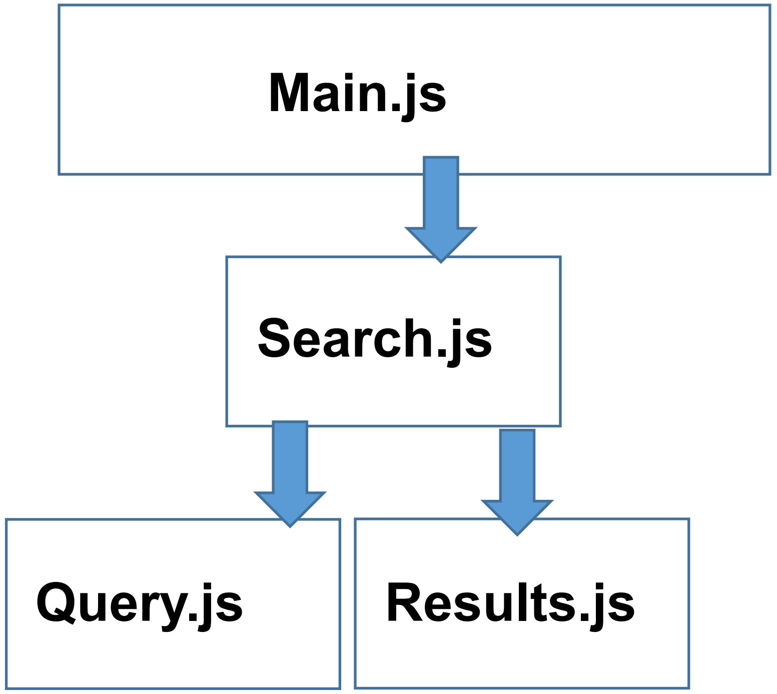




First step in building React applications is determining the component hierarchy.

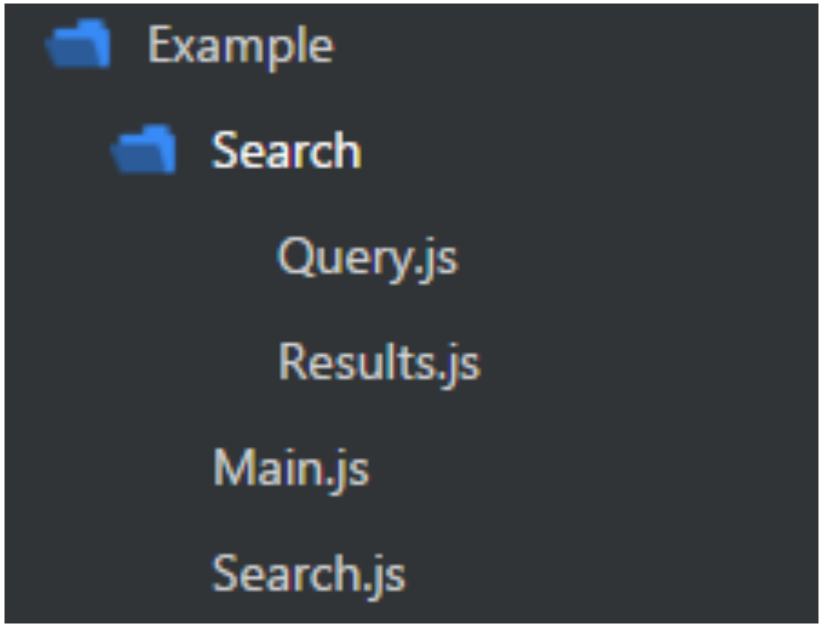
Parent-Child Relationships

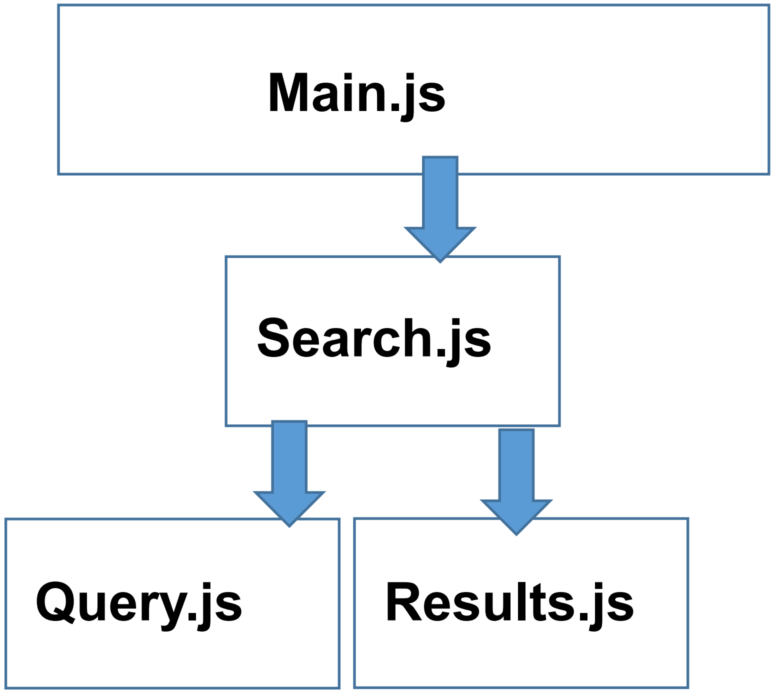




Sometimes, you can then simplify it by realizing certain components are static elements.

Parent-Child Relationships

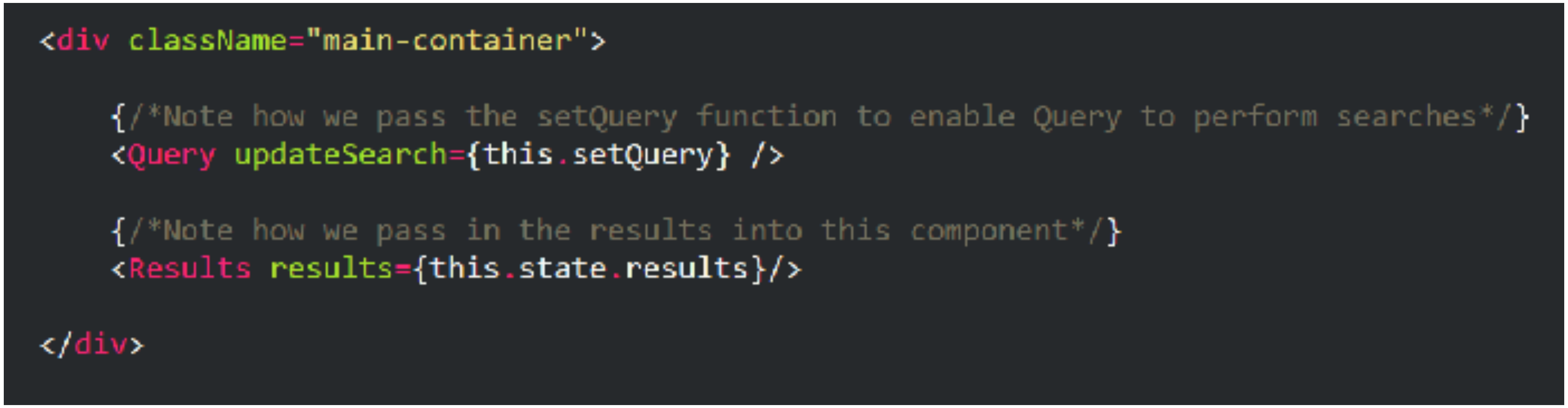




You then code out your components to match the same hierarchy.

States and Props

Passing State from Parent to Child

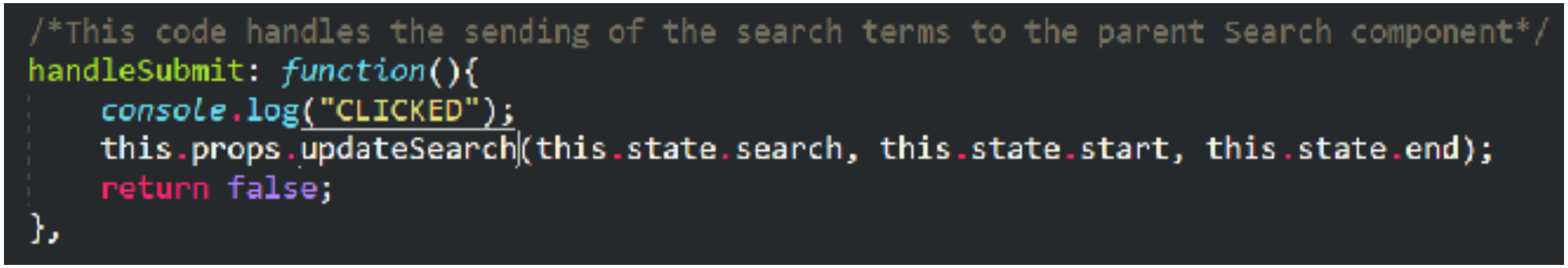


```
<div className="main-container">

    {/*Note how we pass the setQuery function to enable Query to perform searches*/}
    <Query updateSearch={this.setQuery} />

    {/*Note how we pass in the results into this component*/}
    <Results results={this.state.results}/>

</div>
```

- ***Parents can pass data (states and props) or methods to children.***

(It's a bit trickier to send it to parents).

Children Inherit Props

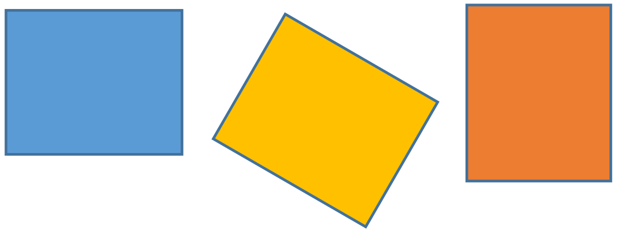


```
/*This code handles the sending of the search terms to the parent Search component*/
handleSubmit: function(){
    console.log("CLICKED");
    this.props.updateSearch(this.state.search, this.state.start, this.state.end);
    return false;
},
```

- *When children inherit the data or method it* ***ALWAYS comes in the form of a <u>prop</u>.***

- *Props can be specifically referenced using* ***<u>this.props.propName</u> syntax****.*

Props vs States: What's the Difference?



States:

- Mutable (i.e. changeable with UI).
- States can be changed using this.setState({})

Props:

- Immutable (i.e. unchangeable).
- Props are static elements. They may be static properties or static methods.

A Moment to Ponder…

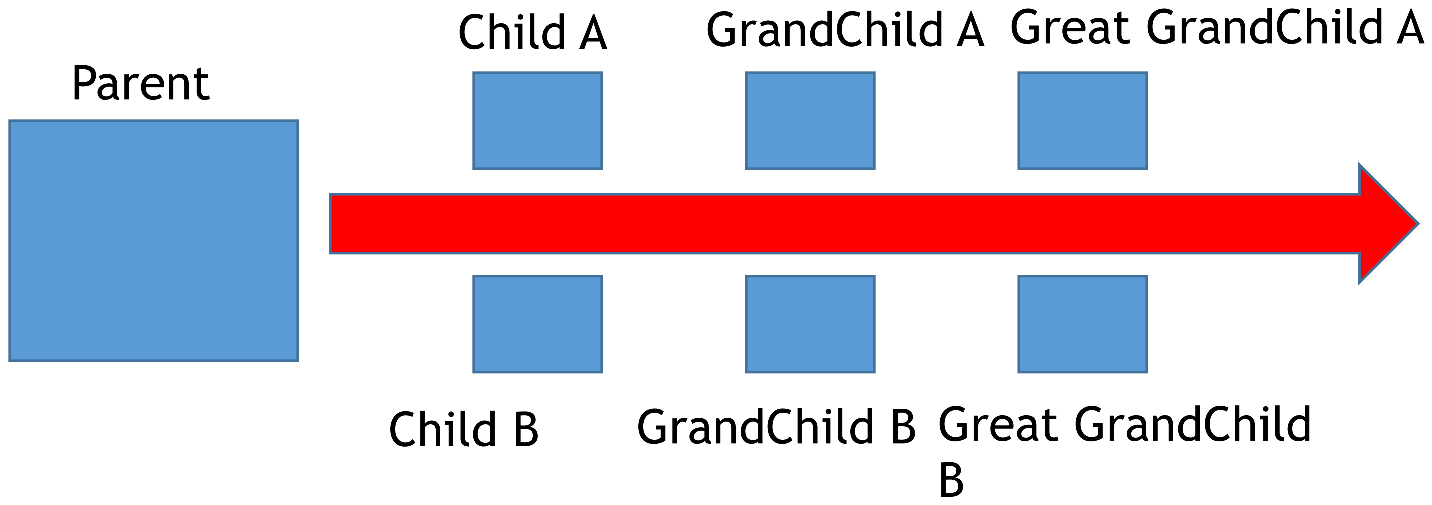
React.JS has a strong preference for passing <u>state</u> from parents to children…

<u>What is the implication of this?</u>

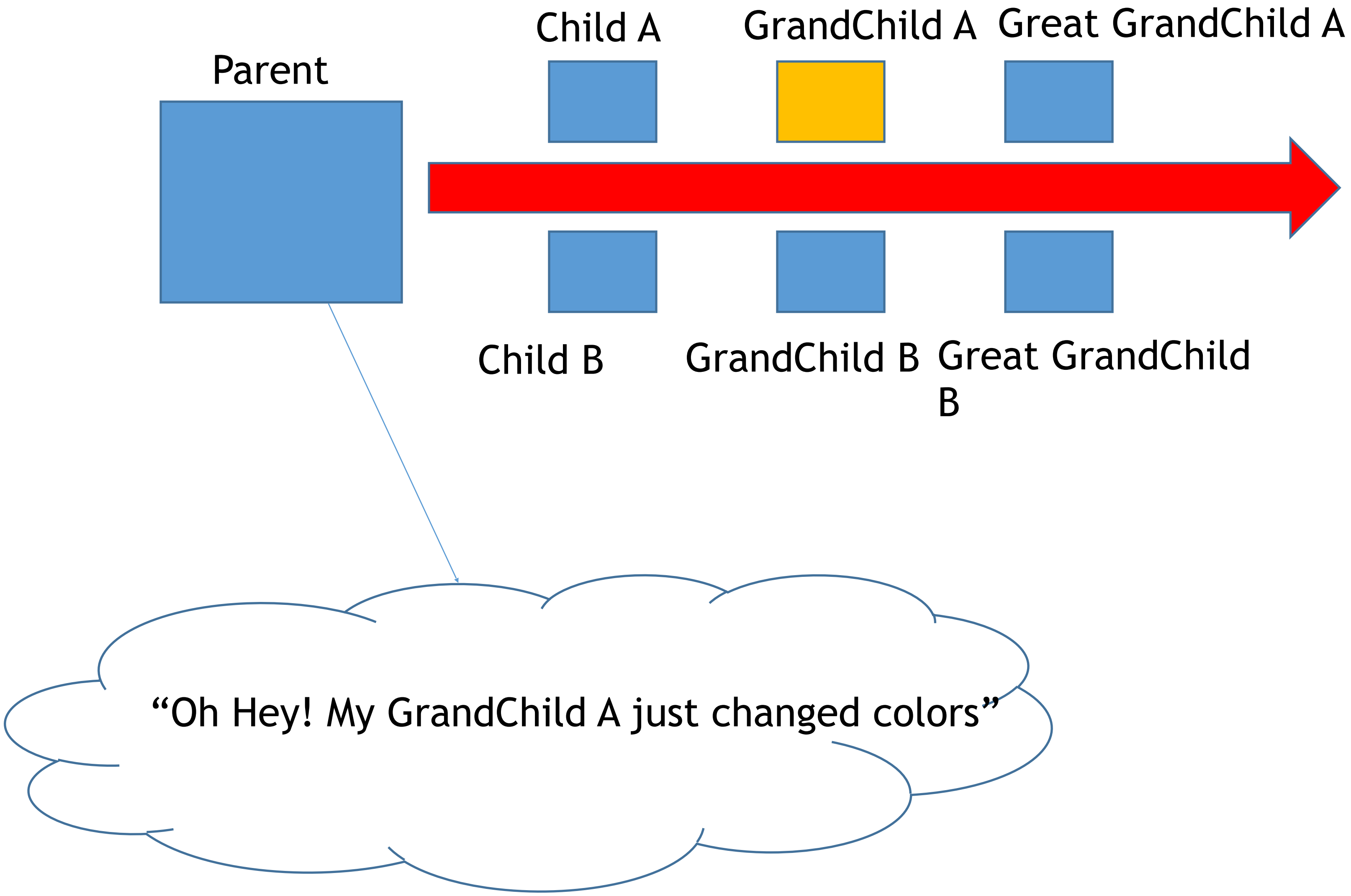
Unidirectional Data Flow

Unidirectional Data Flow

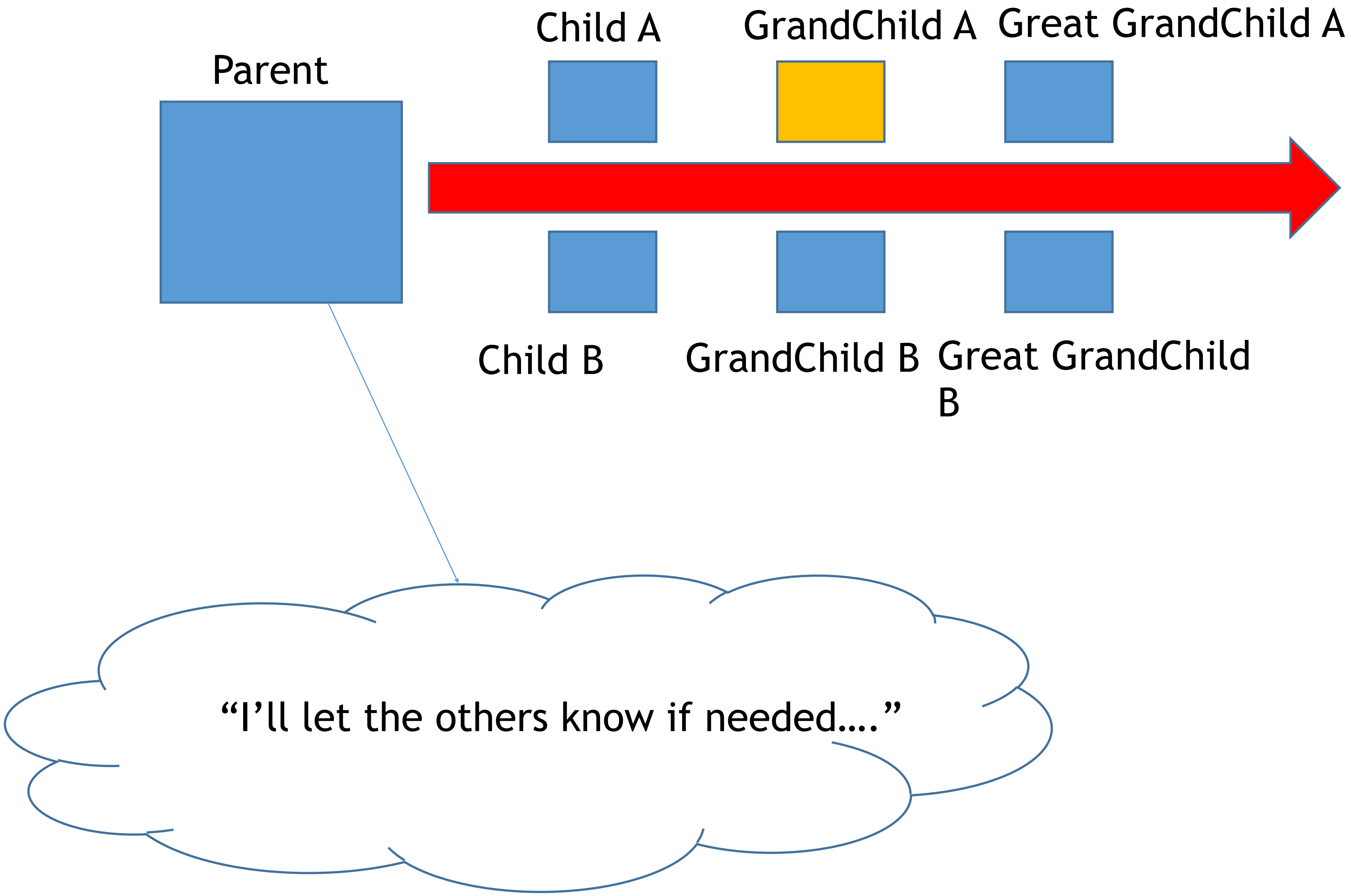
- *React.JS has a strong preference for **unidirectional data flow.***
- *This means that the variables that get manipulated are controlled by parents.*
- *Parents are then responsible for **<u>divvying the data (and state changes) to the children</u>**.*



Unidirectional Data Flow

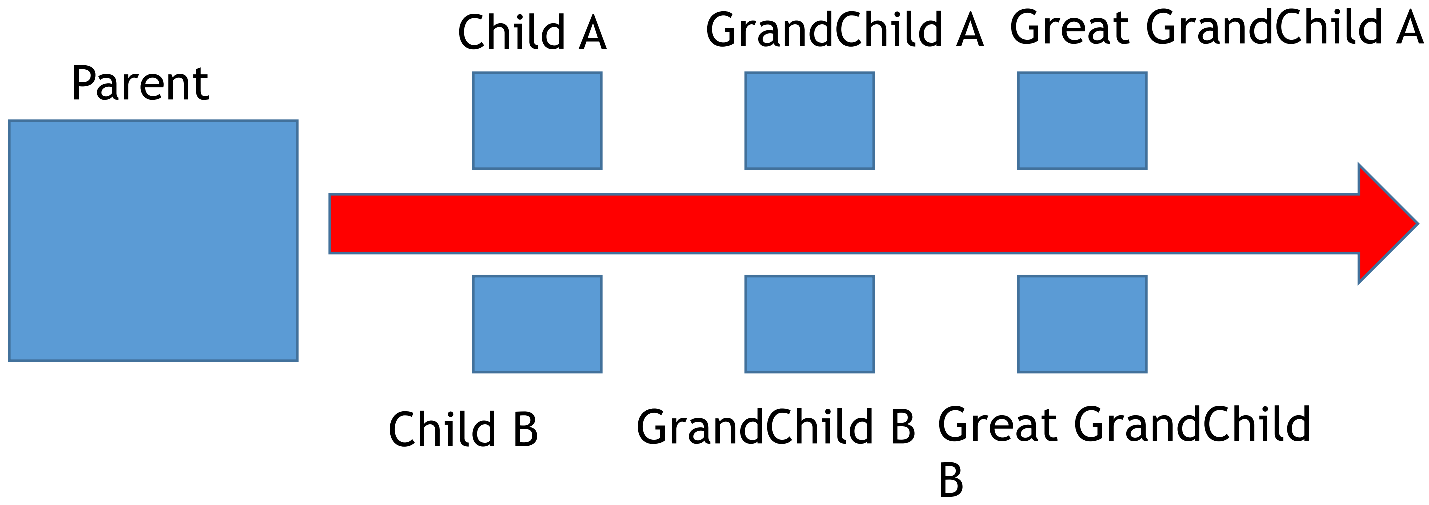


Unidirectional Data Flow



Unidirectional Data Flow

- *This approach is meant to create a level of* ***manageability*** *when it comes to data flow and UI changes.*



Time to Code!

Questions