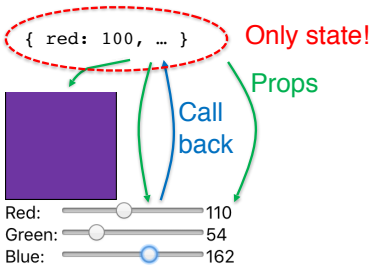
React philosophy (& design pattern)

- Single source of truth (the state)
- Render the HTML (view) you want for the current state
- Use callbacks (and `setState`) to update state and trigger re-rendering



Revisiting the color picker

```
function ColorPicker() {
  const [red, setRed] = useState(0);
  const [green, setGreen] = useState(0);
  const [blue, setBlue] = useState(0);

  const color = {background: `rgb(${red}, ${green}, ${blue})`};
  return (
    <div className="color-picker">
      <ColorSwatch style={color} />
      <LabeledSlider
       label="red"
       value={red}
       setValue={(value)=>{setRed(value)}}/>
    …
    </div>
  );
}
```

Passing state as prop

Passing a callback as prop

"Thinking in React"

1. Break the UI into a component hierarchy
2. Build a static version in React
3. Identify the minimal (but complete) representation of state
4. Identify where your state should live
5. Add "inverse" data flow (data flows down, callbacks flow up)

https://reactjs.org/docs/thinking-in-react.html

PropTypes in action

```
LabeledSlider.propTypes = {
  label: PropTypes.string.isRequired,
  value: PropTypes.oneOfType([
    PropTypes.string,
    PropTypes.number,
  ]).isRequired,
  setValue: PropTypes.func.isRequired,
};
```

Bit of a "code smell"

Catch errors and document component "signature"

How can we style our application

- Static CSS files
- “Import” CSS files like code

  `import './ColorPicker.css'`
- CSS-in-JS

```
const ColorLabel = styled.div`
    display: inline-block;
    width: 50px;
    text-align: left;
    `;
…
<ColorLabel>{props.label}:</ColorLabel>
```

“Style as code”

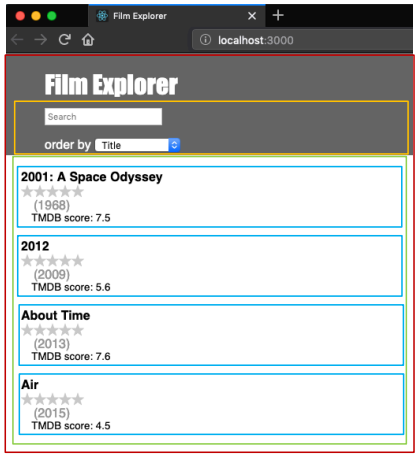
Really about separation of concerns (SoC)

SoC is a design principle that each "unit" in a program should address a different and non-overlapping concern

HTML is content (only), CSS is style (only)

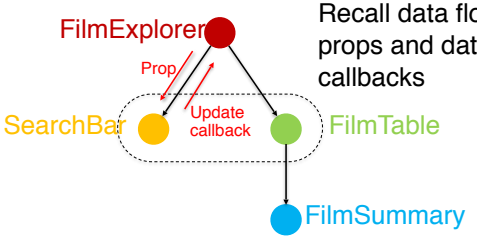
Each component should be separate

What is the component hierarchy?



FilmExplorer
SearchBar
FilmTable
FilmSummary

Review: React state placement



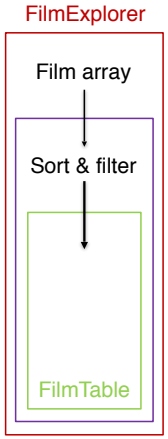
Recall data flows “down” via props and data flows “up” via callbacks

- SearchBar and FilmTable both need the “search term” and “sort type”
- State should “live” in the nearest common ancestor (FilmExplorer)

You are embedding the color picker in a drawing app (to pick the pen color), where should you maintain the color state?

A. In the ColorPicker, and use a callback to communicate changes to the parent drawing component
B. In the drawing component
C. Neither, I heard I am supposed to use Redux to manage state

Container components: Separating logic from UI



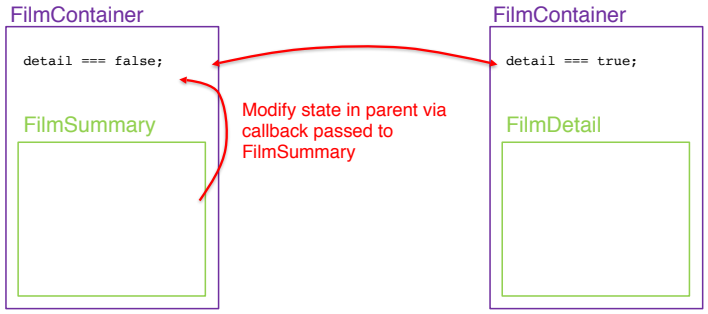
Separation of Concerns:

- Container Component (CC): Concerned with how the application works, i.e. implements logic
- Presentation Component (PC): Concerned with how the application looks. Typically generates DOM.

"Remember, components don't have to emit DOM. They only need to provide composition boundaries between UI concerns." Dan Abramov

CC applied: FilmContainer

How would you apply this design pattern to the toggling between FilmSummary and FilmDetail?



Interlude: Conditional rendering

```
function FilmContainer (props) {
  const [showDetail, setShowDetail] = useState(false);
  const View = showDetail ? FilmDetail : FilmSummary;
  return (
    <View {...props} onClick={()=>{setShowDetail(!showDetail);}} />
  );
}
```

Some common conditional patterns:

```
{boolean && <Component … />}
{boolean ? <Component1 … /> : <Component2 … />}
```

Interlude: Immutable data structures

- Immutable: Once created, a collection cannot be altered
- Persistent: Can create new collections from previous collection and a mutation. The original is still valid.
- Structural Sharing: New collections use the same structure as the original where possible to reduce copying

```
const { Map } = require('immutable');
const map1 = Map({ a: 1, b: 2, c: 3 });
const map2 = map1.set('b', 50)
`${map1.get('b')} vs ${map2.get('b')}` // ? vs ?
```

Adapted from Immutable.js

React: Composition vs. Inheritance?

2001: A Space Odyssey

★★★☆☆

(1968)

TMDB score: 7.5

FilmSummary

Humanity finds a mysterious object buried beneath the lunar surface and sets off to find its origins with the help of HAL 9000, the world's most advanced super computer.

FilmDetail

Should FilmDetail inherit from FilmSummary or contain a FilmSummary?

When do we use subtyping (inheritance)?

- Subtyping is described by an “is a” relationship, e.g. a car “is a” vehicle
- Composition is described by a “has a” relationship, e.g. a car “has an” engine

So FilmDetail “is a” FilmSummary or “has a” FilmSummary?

Formalizing subtyping: Liskov Substitution Principle

Let $\phi(x)$ be a property provable about objects x of type T. Then $\phi(y)$ should be true for objects y of type S where S is a subtype of T.

Turing Award Winner
Barbara Liskov

TL;DR; A method that works on an instance of type T, should also work on any subtype of T

When a Square is not a Rectangle

```
class Rectangle {
  constructor(w, h) {
    this.w = w;
    this.h= h;
  }
  setWidth(w) { this.w = w; }
}

class Square extends Rectangle {
  constructor(side) {
    super(side, side);
  }
  setWidth(w) {
    this.w = w;
    this.h = h;
  }
}
```