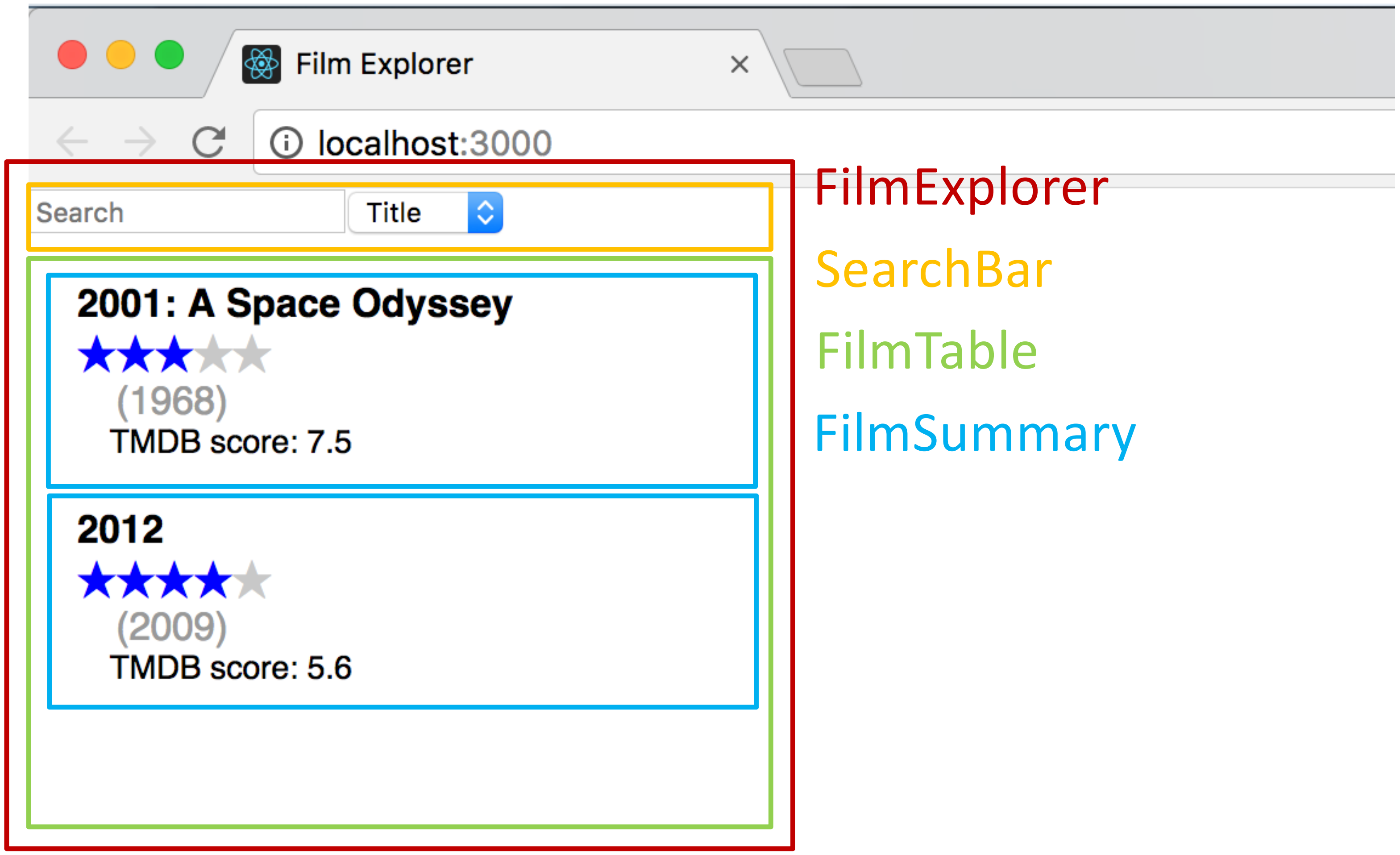
Recall: “Thinking in React”

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

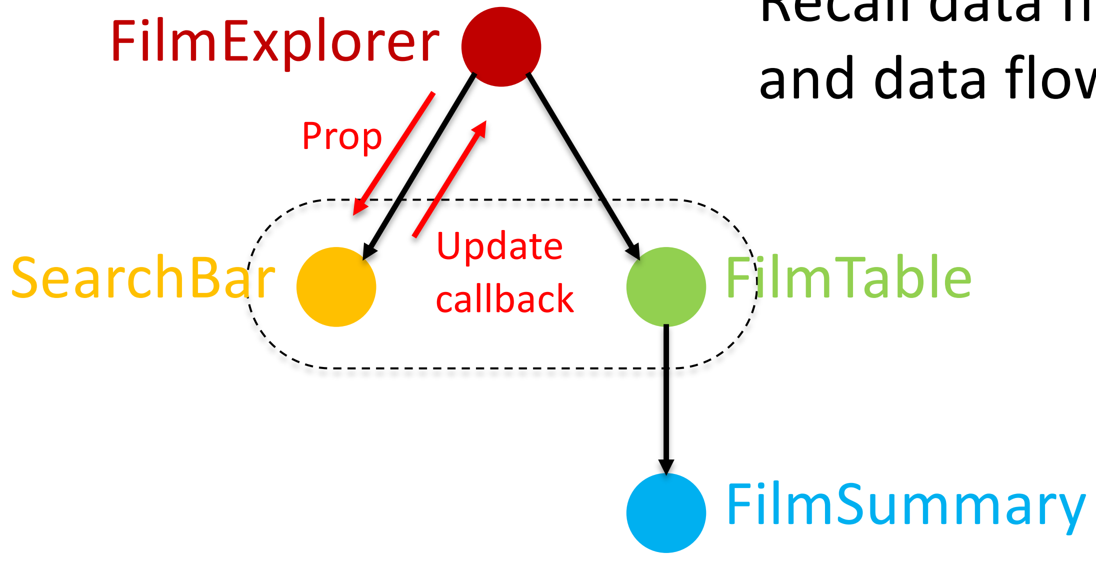
What is the component hierarchy?



2001: A Space Odyssey
(1968)
TMDB score: 7.5

2012
(2009)
TMDB score: 5.6

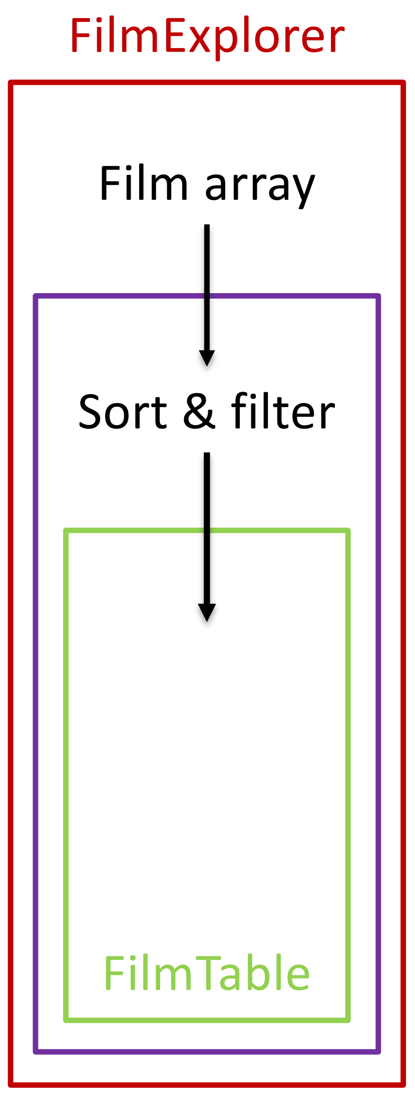
Review: React state placement



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, i.e., FilmExplorer

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

Interlude: Sequences in React

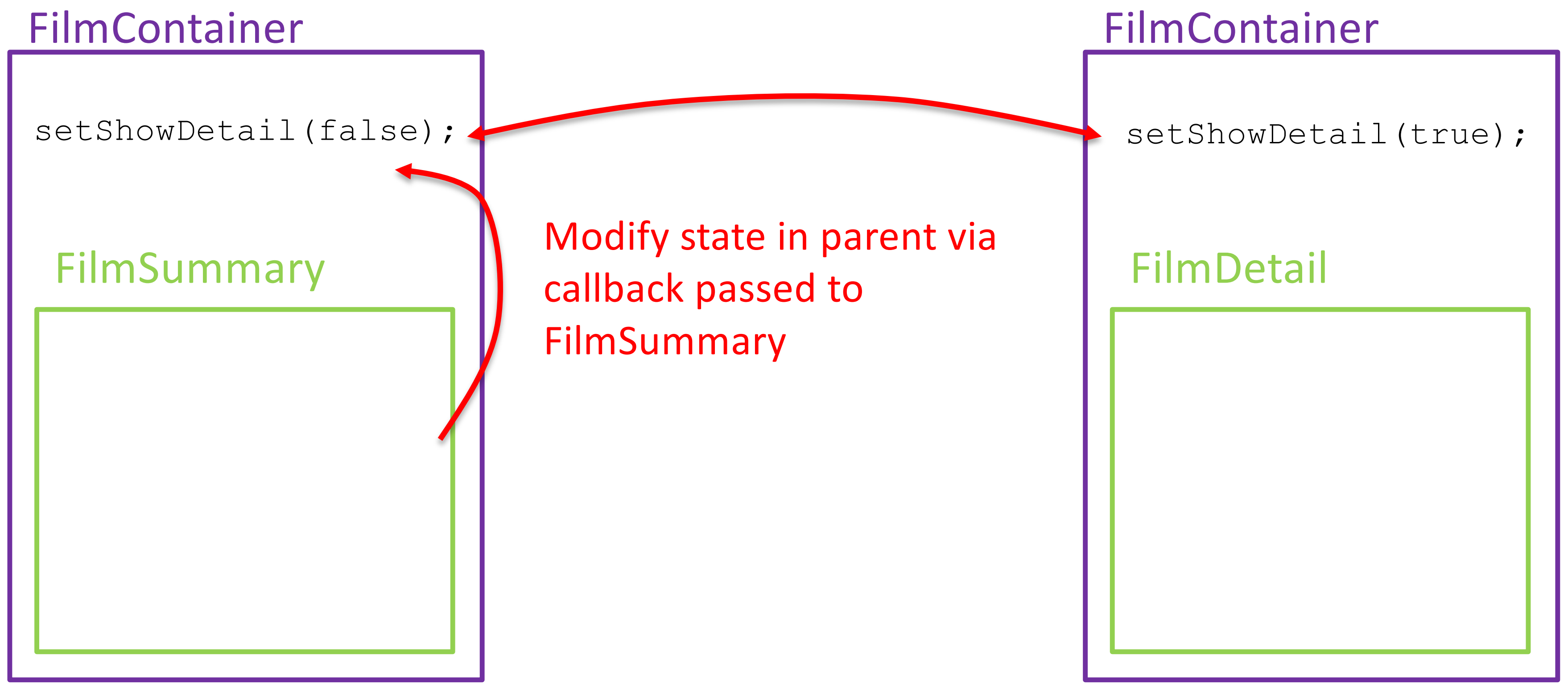
```
function FilmTable(props) {
  const films = props.films.map(film => (
    <FilmContainer
      key={film.id}
      {...film}
      setRatingFor={props.setRatingFor}
     />
  ));
  return <div>{films}</div>;
}
```

"Arrays" need key to uniquely identify components

"Keys help React identify which items have changed, are added, or are removed. Keys should be given to the elements inside the array to give the elements a stable identity. Most often you would use IDs from your data as keys" -ReactJS Docs

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);
  if (showDetail) {
    return <FilmDetail {...props} onClick={() => setShowDetail(false)} />;
  } else {
    return <FilmSummary {...props} onClick={() => setShowDetail(true)} />;
  }
}
```

A React function is code and so you can use conditionals to change views

Some other common conditional patterns:

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

What are some React technical "dichotomies"?

Stateful vs. Stateless

- CC are typically stateful, PC typically stateless (but not always)

Class vs. Functional Components

- Classes can have state! And lifecycle methods.
- Functions are suggested unless you need Class features since they are simpler and may be optimized in the future
- Function components are suggested in all situations (using Hooks if stateful)

Interlude: Rules of Hooks

- Only call Hooks at the top level of a function

  Don’t call Hooks inside loops, conditions, or nested functions

- Only call Hooks from React functions or custom Hooks

  Don’t call Hooks from regular JavaScript functions

https://reactjs.org/docs/hooks-rules.html

Hooks in place of class components

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

  // Render components
}


class ColorPicker extends React.Component {
  constructor(props) {
    super(props);
    this.state = { red: 0, green: 0, blue: 0,};
  };

  render() {
   …
   this.setState({ red: value })
   …
  }
};
```

Make copies instead of mutating state or props

map creates a new array

```
const setRating = (filmid, rating) => {
  const newFilms = films.map((film) => {
    if (film.id === filmid) {
      // or return Object.assign({}, film, { rating: rating});
      return { ...film, rating };
    }
    return film;
  });
  setFilms(newFilms);
}
```

Create a new object instead of mutating

Now newFilms != films, even with shallow (reference) compare

Take home message: Don’t mutate state or props, create new objects

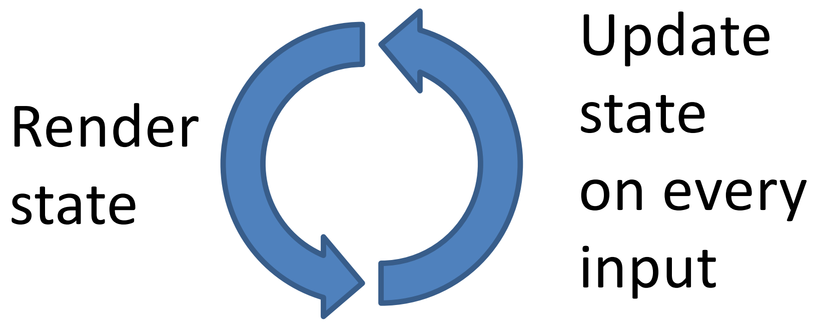
- Mutated props will not compare as different objects and so may not trigger a re-render
- Assigning to state does not trigger a re-render

```
// Wrong (in a class component)
this.state.comment = 'Hello';

// Correct (in a class component)
this.setState({ comment: 'Hello' });

// Hooks prevents the above error (but not calling setComment
// with the same object)
const [comment, setComment] = useState('');
comment = 'Hello'; // Javascript error
```

Review: React controlled components



```
function Example(props) {
  const [title, setTitle] = useState('');

  return (<input
    type="text”
    value={title}
    onChange={(event) => setTitle(event.target.value})}
  />);
}
```

Input value determined by React state

Change updates state, which re-renders input with new value

React: Controlled vs. Uncontrolled

+ Single source of truth
- Lots of callbacks

(Familiar?) Controlled component:

```
<input type="text" value={…} onChange={…}/>
```

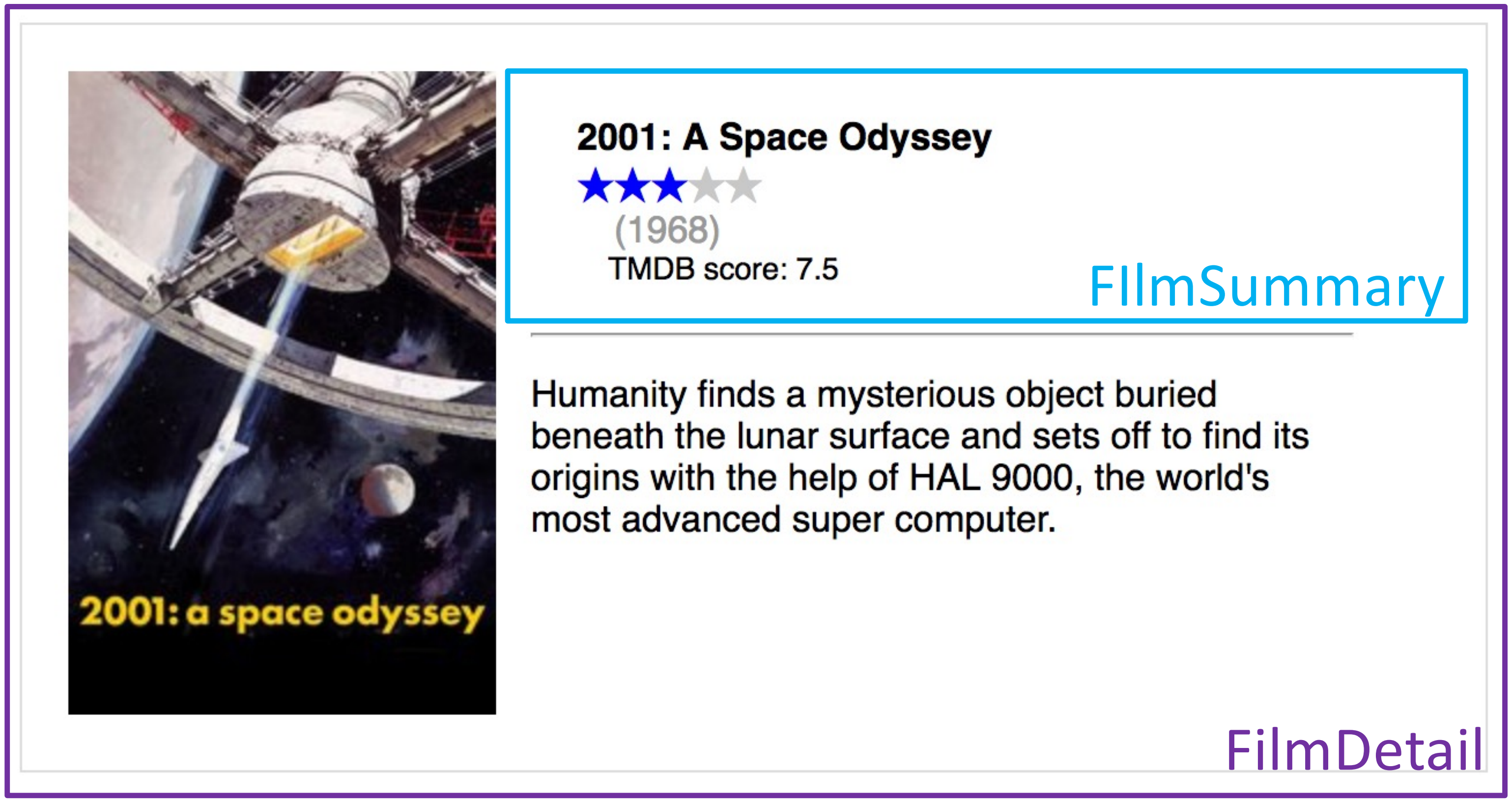
Uncontrolled component:

Reference to real DOM element

```
<input type="text" ref={(input) => this.input = input} />
```

Feature	**Controlled**	**Uncontrolled**
One–time retrieval, e.g. on submit	✓	✓
Validating on submit	✓	✓
Instant validation	✓	✗
Conditionally disabling submit	✓	✗
Several inputs for one piece of data	✓	✗
Dynamically modify data (e.g. capitalize)	✓	✗
<input type="file" />	✗	✓

React: Composition vs. Inheritance?



2001: A Space Odyssey
★★★☆☆
(1968)
TMDB score: 7.5

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.

If implemented as classes, 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?