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?

FilmExplorer
SearchBar
FilmTable
FilmSummary
Review: React state placement

- 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
function FilmTable(props) {
  const films = props.films.map(film => {
    return (<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
How would you apply this design pattern to the toggling between FilmSummary and FilmDetail?

```javascript
FilmContainer

setShowDetail(false);

FilmSummary

Modify state in parent via callback passed to FilmSummary

FilmDetail

setShowDetail(true);

FilmContainer
```
Interlude: **Conditional rendering**

```jsx
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:**

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

map creates a new array

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

    return (<input
        type="text"
        value={title}
        onChange={(event) => setTitle(event.target.value)}
    />);
}
React: Controlled vs. Uncontrolled

(Familiar?) Controlled component:

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

+ Single source of truth
- Lots of callbacks

Uncontrolled component:

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

Reference to real DOM element

<table>
<thead>
<tr>
<th>Feature</th>
<th>Controlled</th>
<th>Uncontrolled</th>
</tr>
</thead>
<tbody>
<tr>
<td>One–time retrieval, e.g. on submit</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Validating on submit</td>
<td>✔</td>
<td>✔</td>
</tr>
<tr>
<td>Instant validation</td>
<td>✔</td>
<td>✘</td>
</tr>
<tr>
<td>Conditionally disabling submit</td>
<td>✔</td>
<td>✘</td>
</tr>
<tr>
<td>Several inputs for one piece of data</td>
<td>✔</td>
<td>✘</td>
</tr>
<tr>
<td>Dynamically modify data (e.g. capitalize)</td>
<td>✔</td>
<td>✘</td>
</tr>
<tr>
<td><code>&lt;input type=&quot;file&quot; /&gt;</code></td>
<td>✘</td>
<td>✔</td>
</tr>
</tbody>
</table>
React: Composition vs. Inheritance?

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?