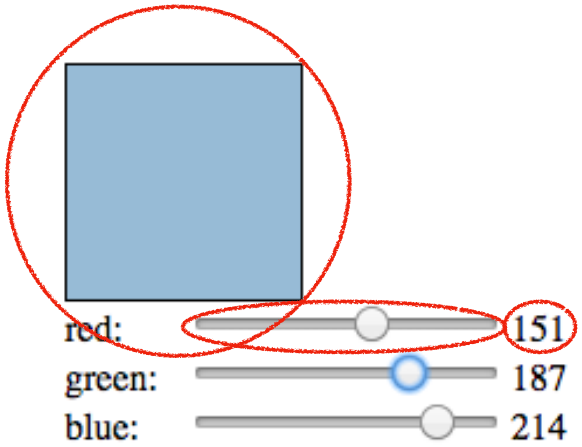
CS 312 Software Development

Introduction to React

Color picker example

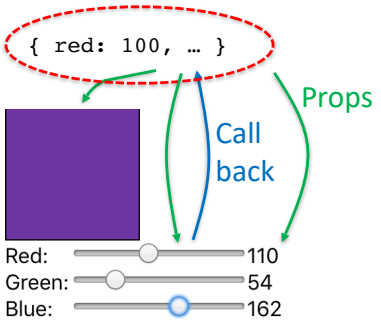


Frameworks

- Event based (e.g., Backbone, Vue)
  - Changing the data triggers an event
  - Views register event handlers
- Two-way binding (e.g. Angular)
  - Assigning to a value propagates to dependent components and vice versa
- Efficient re-rendering (e.g. React)
  - Re-render all subcomponents when data changes

Philosophy of React

- There is a single source of truth (the state)
- Render the UI as it should appear for any given state of the application
- Update the state as a result of user actions
- Repeat (i.e., re-render the UI with the new state)

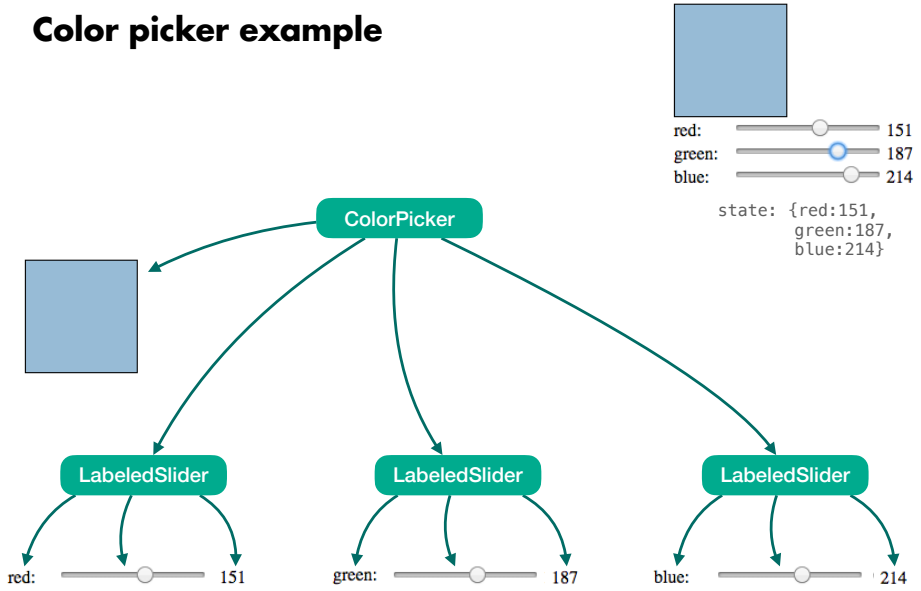


Thinking in React

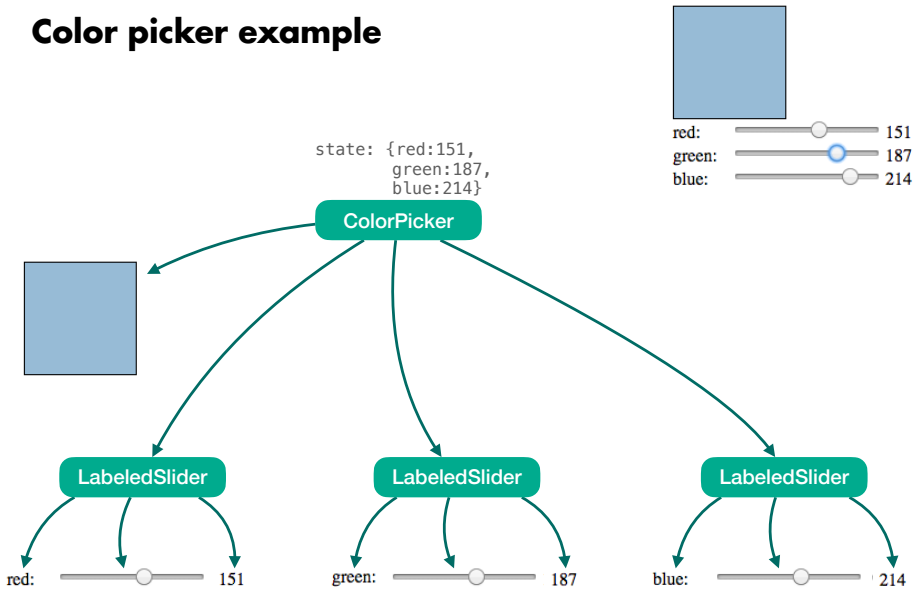
1. Break the UI into a component hierarchy
2. Build a static version in React
3. Identify the minimal (but complete) representation of state
4. Identify where your state should live
5. Add “inverse” data flow (data flows down, callbacks flow up)

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

Color picker example



Color picker example



ColorPicker

```
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">
      <div className="color-swatch" style={color} ></div>
      <LabeledSlider label="red"   value={red}   setValue={setRed}/>
      <LabeledSlider label="green" value={green} setValue={setGreen}/>
      <LabeledSlider label="blue"  value={blue}  setValue={setBlue}/>
    </div>
  );
}
```

ColorPicker

State with React Hooks (useState)

```
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">
      <div className="color-swatch" style={color} ></div>
      <LabeledSlider label="red"   value={red}   setValue={setRed}/>
      <LabeledSlider label="green" value={green} setValue={setGreen}/>
      <LabeledSlider label="blue"  value={blue}  setValue={setBlue}/>
    </div>
  );
}
```

```
const [currentValue, setter] = useState(initial value)
```

destructuring assignment

ColorPicker

Passing props

```
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">
      <div className="color-swatch" style={color} ></div>
      <LabeledSlider label="red"   value={red}   setValue={setRed}/>
      <LabeledSlider label="green" value={green} setValue={setGreen}/>
      <LabeledSlider label="blue"  value={blue}  setValue={setBlue}/>
    </div>
  );
}
```

passing state and setter as props

in JSX, we surround JS with {}

LabeledSlider

```
function LabeledSlider({ label, value, setValue }) {
  return (
    <div>
      <div className="color-label">{label}</div>
      <input type="range"
         min="0"
         max="255"
         value={value}
         onChange={(event)=>{setValue(parseInt(event.target.value,10))}}/>
      <span>{value}</span>
      </div>
  );
}
```

LabeledSlider

Props

single destructured argument “props”

```
function LabeledSlider({ label, value, setValue }) {
  return (
    <div>
      <div className="color-label">{label}</div>
      <input type="range"
         min="0"
         max="255"
         value={value}
         onChange={(event)=>{setValue(parseInt(event.target.value,10))}}/>
      <span>{value}</span>
      </div>
  );
}
```

LabeledSlider

Controlled components

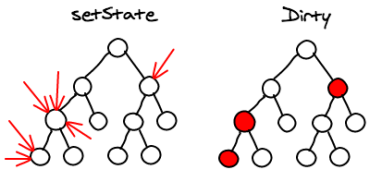
```
function LabeledSlider({ label, value, setValue }) {
  return (
    <div>
      <div className="color-label">{label}</div>
      <input type="range"
         min="0"
         max="255"
         value={value}
         onChange={(event)=>{setValue(parseInt(event.target.value,10))}}/>
      <span>{value}</span>
      </div>
  );
}
```

any change updates state, forcing a re-render

Form elements (like input) are **controlled**

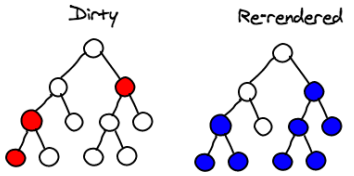
We exploit React's re-render loop to get interaction while maintaining a single source of truth

React mechanics



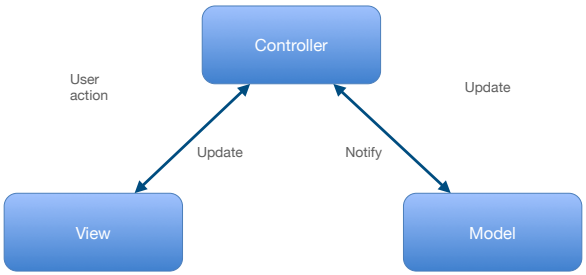
Batched updates

Sub-tree re-rendering



https://calendar.perfplanet.com/2013/diff/

Model View Controller (MVC)



React