

---

# CSE 331

# Software Design & Implementation

Kevin Zatloukal  
Fall 2020  
Modern Web UIs

# Problems

---

This is better, but it still has problems...

1. Still no checking of HTML (opaque strings)
2. Modularity is still poor
  - need to join strings into one big string
3. More boilerplate
  - minimized JS file would change function names
  - need to call `btn.addEventListener` by hand

# JSX

---

- Fix the first problem by adding HTML as a JS type
- This is supported in `.jsx` files:

```
let x = <p>Hi, {name}</p>;
```

- Compiler can now check that this is valid HTML
- `{...}` replaced with string value of expression

# JSX Gotchas

---

- Put `(..)` around HTML if it spans multiple lines
- Cannot use `class="btn"` in your HTML
  - `class`, `for`, **etc.** are reserved words in JS
  - **use** `className`, `htmlFor`, **etc.**
- Must have a single top-level tag:
  - **not:** `return <p>one</p><p>two</p>;`
  - usually fixed by wrapping those parts in a `div`

# Problems

---

This is even better, but it still has problems...

1. Modularity is still poor
  - need to join strings into one big string
2. More boilerplate
  - minimized JS file would change function names
  - need to call `btn.addEventListener` by hand

# React

---

- Regain modularity by allowing custom tags

```
let app = (  
  <div>  
    <TitleBar name="My App" />  
    <EditPane rows="80" />  
  </div>);
```

- TitleBar **and** EditPane can be separate modules
  - their HTML gets substituted in these positions

# React

---

- Custom tags implemented using classes

```
class TitleBar extends React.Component {
```

- **Attributes** (`name="My App"`) passed in `props` arg
- Method `render` produces the HTML for component
- Framework joins all the HTML into one blob
  - can update in a single call to `innerHTML = ...`

# React Example

---

register-react/...



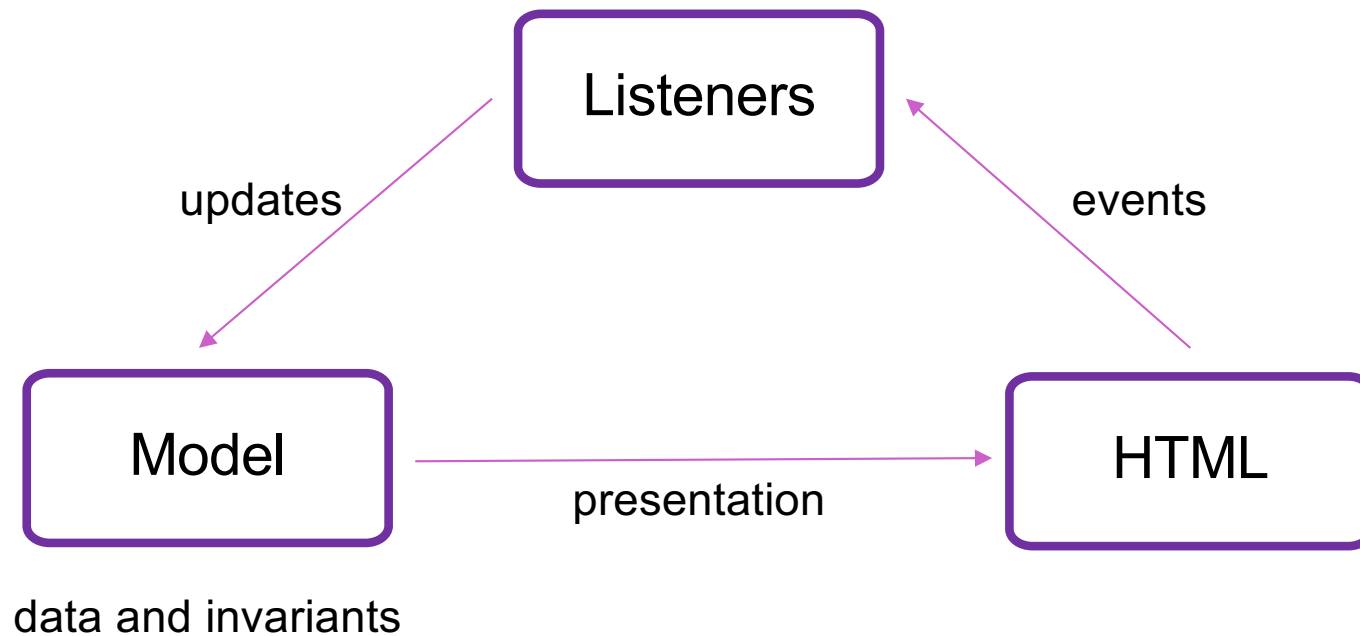
# React State

---

- Last example was not dynamic!
  - there was no model
  - (why have classes then?)

# Structure of a React Application

---



# React State

---

- Last example was not dynamic!
  - there was no model
  - (why have classes then?)
- Components become dynamic by maintaining state
  - stored in fields of `this.state`
  - call `this.setState({field: value})` to update
- React will respond by calling `render` again
  - will automatically update the HTML to match the HTML produced by this call

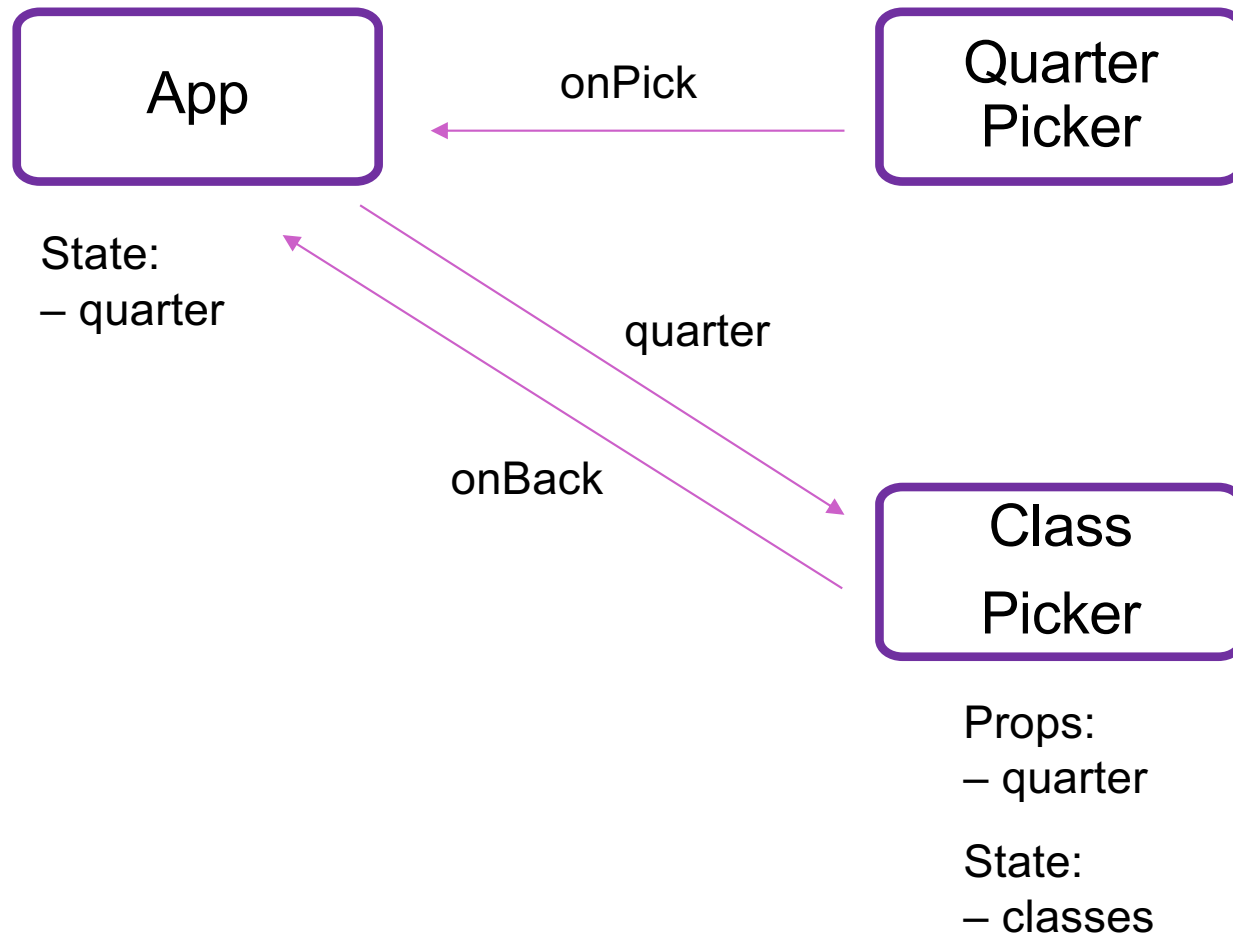
# Example 5

---

register-react2/...

# Structure of Example React App

---



# React State

---

- Custom tag also has its own events
- Updating data in a parent:
  - sends parent component new data via event
  - parent updates state with `setState`
  - React calls parent's `render` to get new HTML
    - result can include new children
    - result can include changes to child props

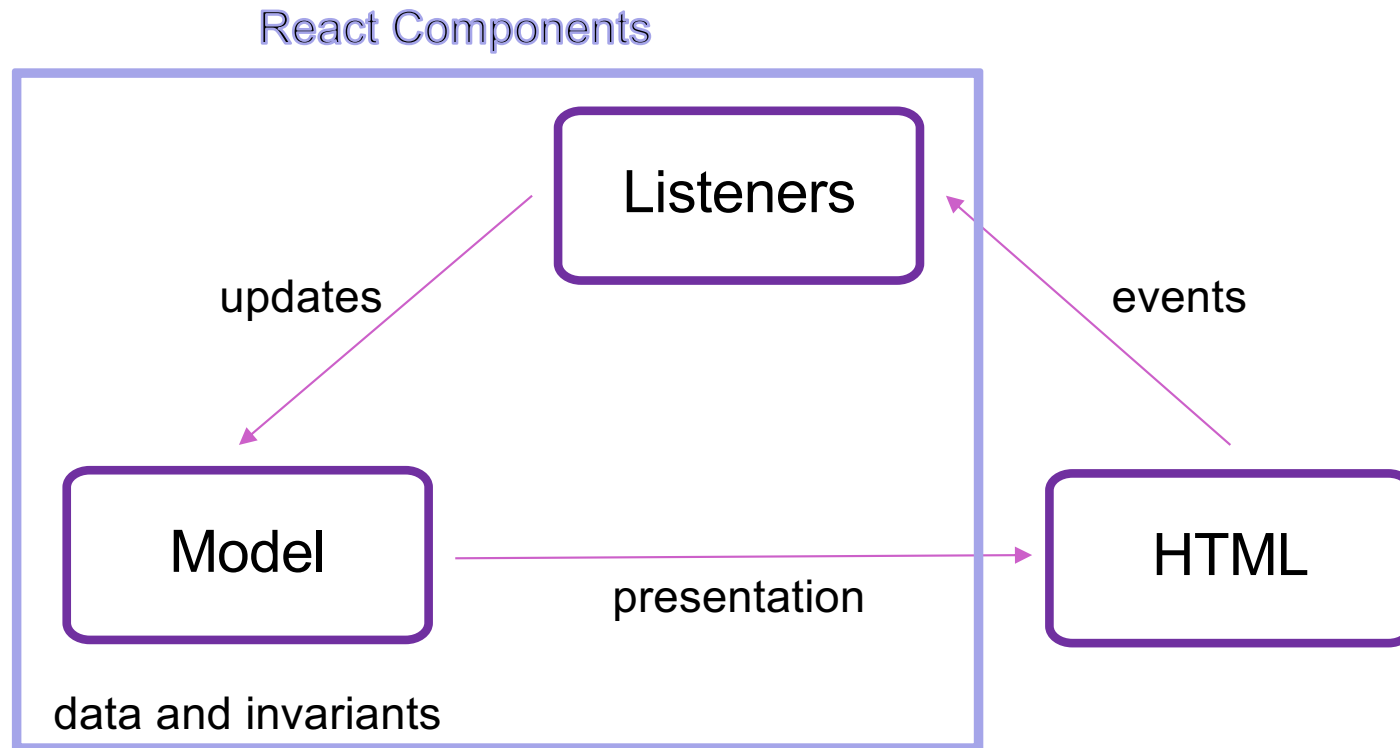
# Splitting the Model

---

- State should exist in the **lowest common parent** of all the components that need it
  - sent down to children via *props*
- Children change it via *events*
  - sent up to the parent so it can change its state
- Parent's render creates new children with new props

# Structure of a React Application

---





# Structure of a React Application

---

- Model must store all data necessary to generate the exact UI on the screen
  - react may call `render` at any time
  - must produce identical UI
- Any state in the HTML components must be mirrored in the model
  - e.g., every text field's `value` must be part of some React component's state
  - render produces

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

# React setState

---

- `setState` does not update state instantly:

```
// this.state.x is 2
this.setState({x: 3});
console.log(this.state.x); // still 2!
```

- Update occurs after the event finishes processing
  - `setState` adds a new event to the queue
  - work is performed when that event is processed
- React can batch together multiple updates

# React Gotchas

---

- `render` should not have side-effects
  - only *read* `this.state` in render
- Never modify `this.state`
  - use `this.setState` instead
- Never modify `this.props`
  - read-only information about parent's state
- Not following these rules may introduce bugs that will be hard to catch!

# React Performance

---

- React re-computes the tree of HTML on state change
  - can compute a “diff” vs last version to get changes
- Surprisingly, this is not slow!
  - slow part is calls into browser methods
  - pure-JS parts are very fast in modern browsers
  - processing HTML strings is also incredibly fast

# React Tools

---

- Use of compilers etc. means new tool set
- `npm` does much of the work for us
  - installs third-party libraries
  - runs the compiler(s)