香港中文大學
The Chinese University of Hong Kong

AN INTRODUCTION TO REACTJS

CSCI2720 2022-23 Term 1

Building Web Applications

Dr. Chuck-jee Chau
chuckjee@cse.cuhk.edu.hk

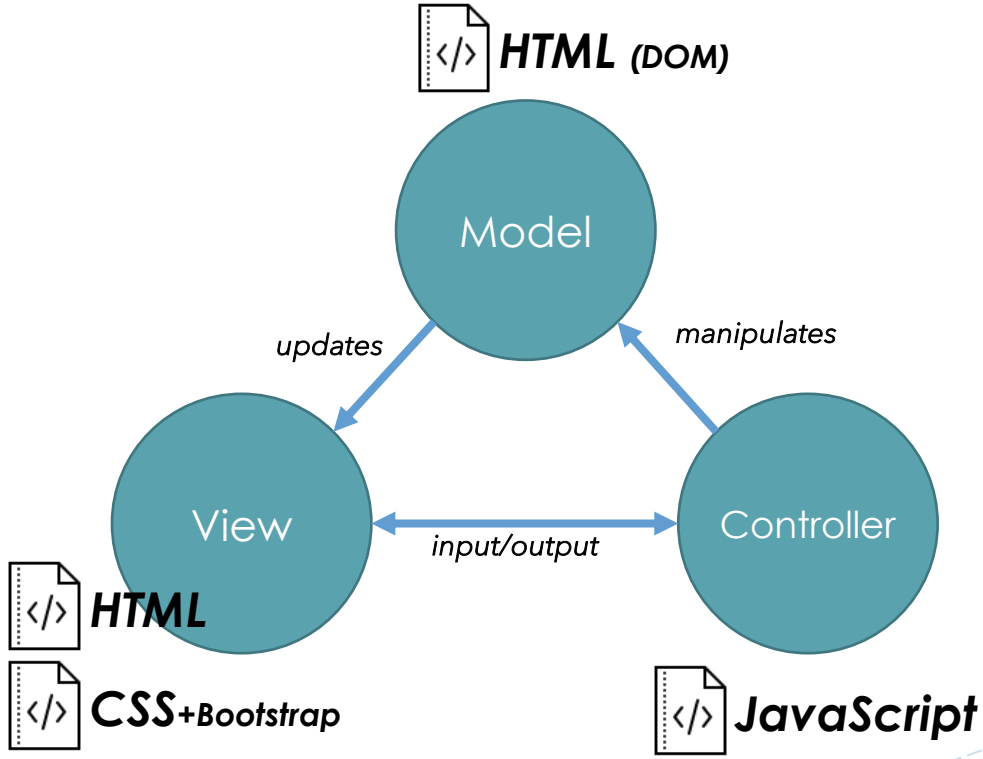
OUTLINE

- Basics of the Web
- Frontend frameworks and libraries
- Starting with React
- Virtual DOM and JSX
- Components
- Props and states
- Events
- Conditional rendering
- List and keys
- Forms
- Lifecycle methods
- Learn more for React

BASICS OF THE WEB

- Markup + Styling + Scripts
- ***HTML*** + ***CSS*** + ***JavaScript***

- Most modern libraries or frameworks only ***helps you generate*** or manipulate these

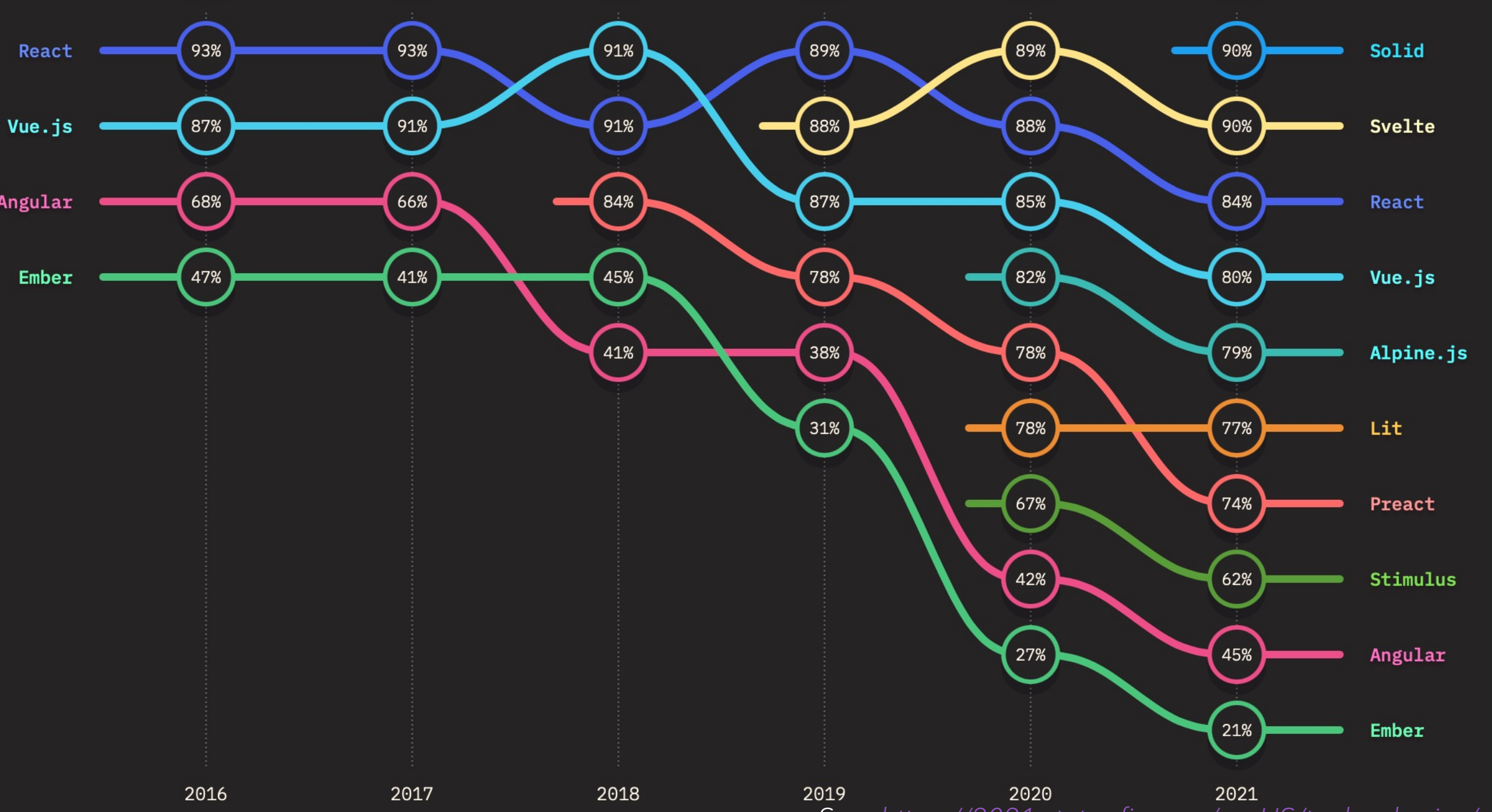


TRANSPILING

- Web standards: HTML, CSS, JavaScript
- Enhancement and ***syntactic sugar*** – make programming *easier*
  - Template engines: easily generated HTML, e.g., Emmet
  - CSS preprocessors: Sass, less
  - JavaScript flavors: TypeScript, JSX, CoffeeScript, …
    - *See: https://scotch.io/tutorials/javascript-transpilers-what-they-are-why-we-need-them*
- Extra transpiling (source-to-source compiling) is needed, to generate files browsers can read

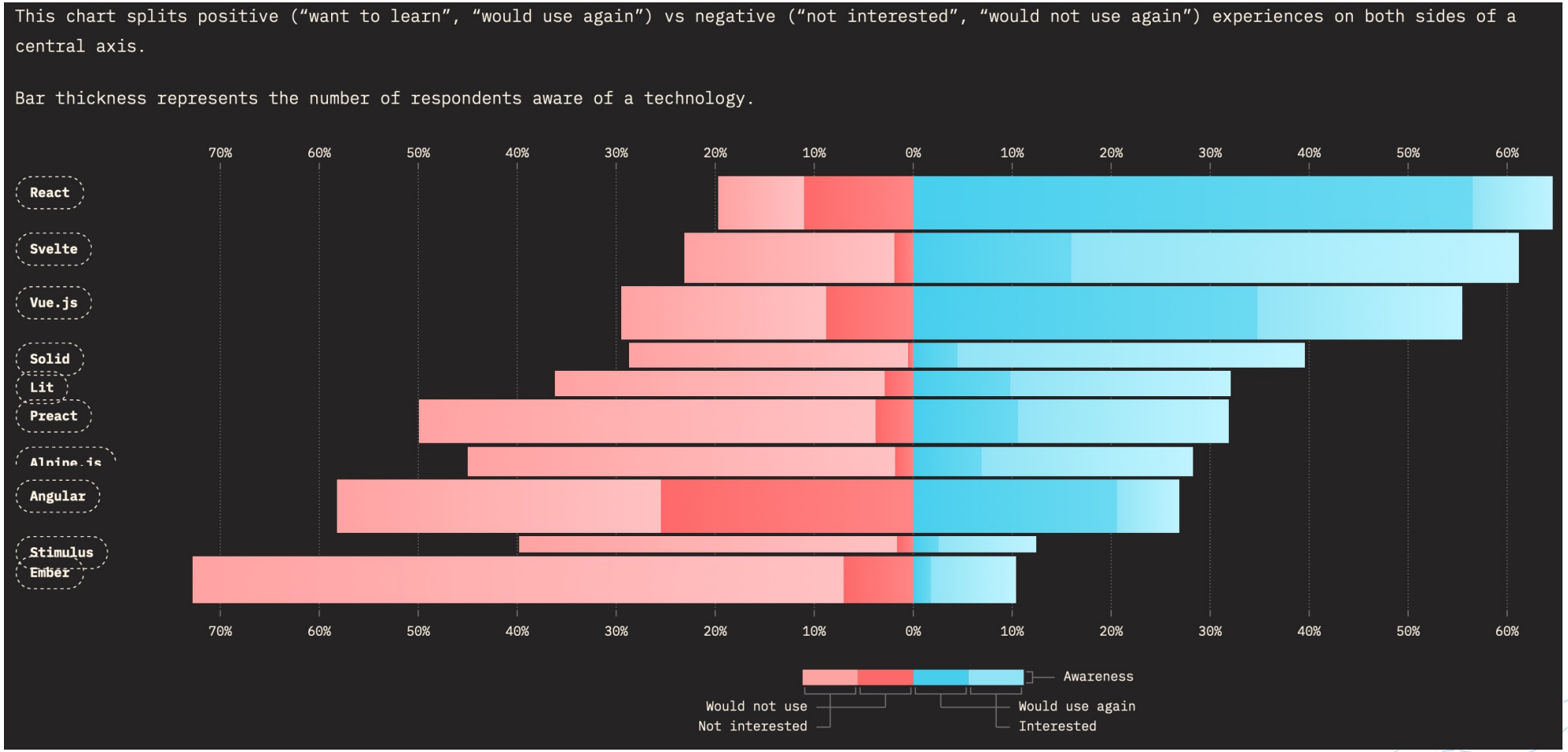
FRONTEND FRAMEWORKS AND LIBRARIES

See: https://hackernoon.com/angular-vs-react-vs-vue-which-is-the-best-choice-for-2019-16ce0deb3847
See: https://www.codeinwp.com/blog/angular-vs-vue-vs-react/



See: https://2021.stateofjs.com/en-US/technologies/

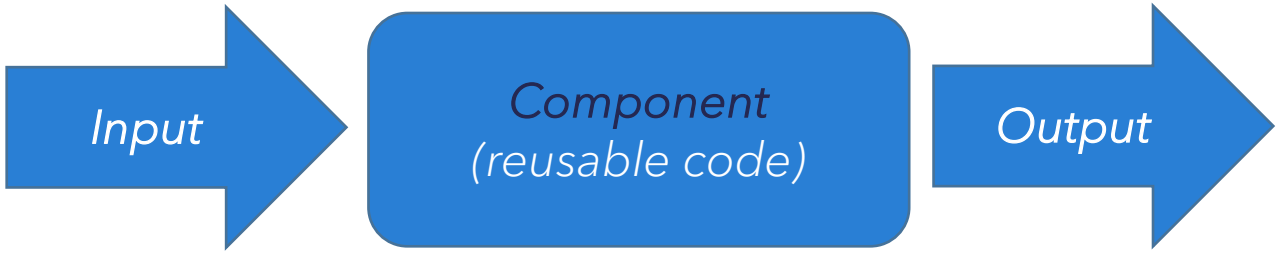
FRONTEND FRAMEWORKS AND LIBRARIES



See: https://2021.stateofjs.com/en-US/technologies/front-end-frameworks/

Angular	React	Vue
Since 2010	Since 2013	Since 2014
by Google	by Facebook	by ex-Google engineer
AngularJS (v1) was a library, and *Angular* (v2+) is a framework governing more than just the frontend (*opinionated*)	Frontend library, focusing on user interface	Lightweight framework "taking the best from Angular", with some features similar to React

AN OVERSIMPLIFIED COMPARISON



REACTJS

- Created by Jordan Walke, a Facebook engineer in 2011
- Deployed in Facebook and Instagram since then
  - Open source in 2013
- Current version: 18.2 in June 2022

ADVANTAGES OF REACT

- Fast
  - Quick and responsive by selective rendering
- Modular
  - Small and reusable modules which are easier for maintenance
- Scalable
  - Especially suitable for lots of changing data
- Flexible
  - It's not only useful for web apps!
- *Read more: https://www.freecodecamp.org/news/best-react-javascript-tutorial/*

WHAT DOES REACT GIVE YOU?

- The virtual DOM
- JSX
- Components
- State and Props
- *and more…*

STARTING WITH REACT

- There are two+ ways to get React into your site!

1. Embedding React using `<script>`
   - Easier setup, but is not optimized for production sites
   - No special commands needed, no need for `import` in JS
   - *We will be using this method in this lecture*
2. JavaScript toolchains
   - Some more preparation, but allows automated dev/testing environment setup, and optimization for production
   - *e.g.*, `create-react-app`, Next.js, Gatsby, etc.

EMBEDDING REACT

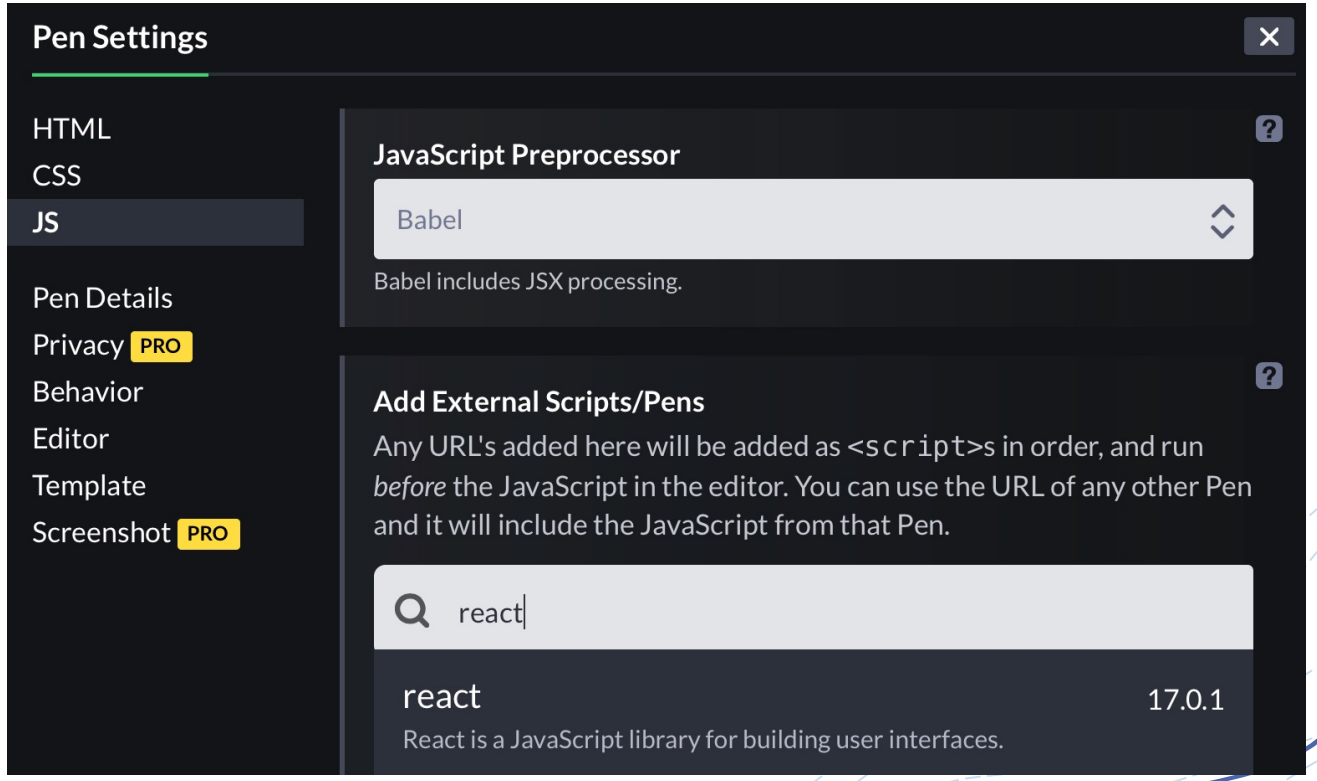
- The “simplest way”: Add these lines into the **<head>** of your HTML file

```
<head>
  ...
  <script src="https://unpkg.com/react@18/umd/react.development.js"
crossorigin></script> // @18 specifies the version to use
  <script src="https://unpkg.com/react-dom@18/umd/react-dom.development.js"
crossorigin></script>
  <script src="https://unpkg.com/babel-standalone@6/babel.min.js"></script>
  ...
</head>
```

- unpkg.com is a free service providing CDN for libraries
  - Use ***production.min.js*** instead of ***development.js*** for deployment, which provides reduced error output and other optimizations
  - Learn more about UNPKG: *https://unpkg.com* and *http://unpkg.org*

USING REACT ON CODEPEN

- Codepen is another viable platform for testing and learning
  - In *JS settings*, choose **Babel** as the JS preprocessor, and add external scripts by searching for **`react`** and **`react-dom`**



THE ENTRY POINT

- You can pass the DOM control of part of your HTML to ReactDOM by specifying an element (e.g., **`<div>`**) with ID

```
<body>
  ...
  <div id="app">React is rendering...</div>
  <!-- div contents will be replaced when rendered by ReactDOM -->
  ...
</body>
```

```
const root = ReactDOM.createRoot(document.querySelector("#app"));
root.render( component );
```

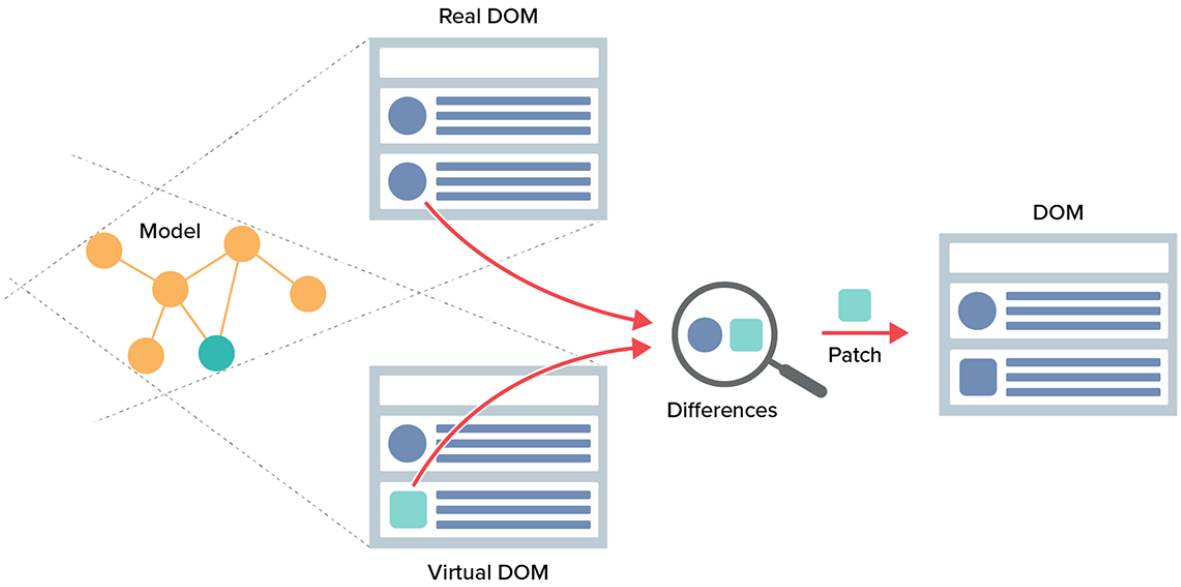
- This **`div`** will be updated by React automatically, when you specify it in **`render()`** later

THE VIRTUAL DOM

- The browser keeps the DOM tree to render and display HTML elements
- React has an extra in-memory data structure for the DOM as **`ReactDOM`**
  - Now, forget about the JS events in the DOM…
- *Question: What would happen if you render the same thing twice?*

THE VIRTUAL DOM

- When something has changed, entire UI is re-rendered in ReactDOM
- React find out the ***difference*** between the original and updated version
- The actual DOM is updated with only the calculated difference



See: https://medium.com/zenofai/beginners-guide-to-reactjs-3ca07f56d526

JSX

- A syntax extension of JavaScript
  - *Optional* for React, but everyone is using it, and so are we!
- You need to include the Babel transpiler to use JSX
  - Babel was embedded as the 3rd item a few slides ago
  - Babel also adds support of ES2015 code to old browsers
- JSX is used as type **`"text/babel"`**, and is usually stored with file name **`.jsx`**
  - Although using **`.js`** is fine, and some online IDE won't let you specify
- JSX produces React "elements", neither HTML nor string

COMBINING HTML+JS+JSX

- Learn more here: *https://reactjs.org/docs/introducing-jsx.html*

index.html

```
<div id="app">React is rendering...</div>
<script src="script.jsx"></script>
```

https://codepen.io/chuckjee/pen/qBbjpyQ

script.jsx

```
function formatName(u) {
  return u.firstName + ' ' + u.lastName;
}

const user = {  firstName: 'WebApp',  lastName: 'CUHK' };
const element = <h1>Hello, {formatName(user)}!</h1>;

const root = ReactDOM.createRoot(document.querySelector('#app'));
root.render(element);
```

USING CSS IN REACT

- ***Important***: Writing JSX is not directly writing HTML, so some HTML attributes could be different…
- To use inline styles with a **`style`** attribute, special syntax is required:

```
// Result style: '10px'
<div style={{ height: 10 }}> Hello World! </div>

// Result style: '10%'
<div style={{ height: '10%' }}> Hello World! </div>
```

- To specify the `class` attribute, it is better to use **`className`** instead
- *Read more: https://reactjs.org/docs/dom-elements.html*

COMPONENTS

- ***Components*** can be anything in the UI, e.g.,
  - A paragraph, a list, a table, a button, or even invisible objects
  - Reusable modules as building blocks
  - Name starts with an Upper-case letter

Functional component

```
function Welcome(props) {
  return <h1>Hello, {props.name}</h1>;
}
function App() {
  return (
    <div>
      <Welcome name="chuckjee" />
      <Welcome name="student" />
    </div>
  );
}
const root = ReactDOM.createRoot(document.querySelector('#app'));
root.render(<App />);
```

Class component

```
// This is the same as...
class Welcome extends React.Component {
  render() {
    return <h1>Hello, {this.props.name}</h1>;
  }
}
```

A CLASS COMPONENT

```
class App extends React.Component {
  render() {
    return (
    <div className="container">
      <Item />
      <Item />
      <Item />
    </div>
   );
  }
}
class Item extends React.Component {
  render() { return <div className="box">CSCI</div>; }
}
const root = ReactDOM.createRoot(document.querySelector('#app'));
root.render(<App />);
```

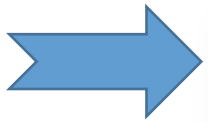
```
.container {
  background: #cccccc;
}
.box {
  background: #eeeeee;
  margin: 5px;
  width: 100px;
  display: inline-block;
}
```

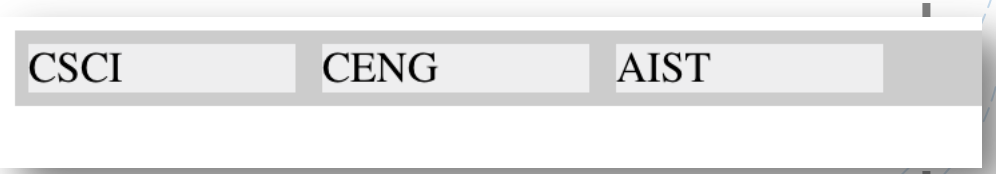
PROPS

- ***Props*** (*properties*) are immutable data in the component
  - Useful for parent components to pass data to children

https://codepen.io/chuckjee/pen/JjGJgWL

```
class App extends React.Component {
  render() {
    return (
    <div className="container">
      <Item subject="CSCI" />
      <Item subject="CENG" />
      <Item subject="AIST" />
    </div>
   );
  }
}
class Item extends React.Component {
  render() { return <div className="box">{this.props.subject}</div>; }
}
const root = ReactDOM.createRoot(document.querySelector('#app'));
root.render(<App />);
```





STATES

- The behaviour of a component at a given moment in time is defined by the ***states***
- Values in the state should ONLY be updated using `this.setState()`
  - Just usual JS *name:value* pairs
- When the state changes, affected components may be re-rendered
- *Note*: functional components were *stateless* before, but now are starting to support state with `useState()`
  - *See: https://reactjs.org/docs/hooks-state.html*

USING STATES

https://codepen.io/chuckjee/pen/OJMgKdy

```
class App extends React.Component {
  constructor() {
    super();
    this.state = { s1:"CSCI", s2:"CENG", s3:"AIST" };
  }
  render() {
    return (
    <div class="container">
      <Item subject={this.state.s1} />
      <Item subject={this.state.s2} />
      <Item subject={this.state.s3} />
    </div>
   );
  }
}
class Item extends React.Component {
  render() { return <div class="box">{this.props.subject}</div>; }
}

const root = ReactDOM.createRoot(document.querySelector('#app'));
root.render(<App/>);
```

EVENTS

- The syntax for React events are slightly different from JS
  - ***camelCase*** than lowercase
  - Passing an event handler function in JSX
  - The React event handler can be passed as a prop to a child
    - i.e., the child uses its parent's handler to handle an event

```
function ActionLink() {
  function handleClick(e) {
    e.preventDefault();
    console.log('The link was clicked.');
  }
  return (
    <a href="#" onClick={handleClick}>
      Click me
    </a>
  );
}
```

See: https://reactjs.org/docs/handling-events.html

EVENTS

- Always mind the subtle difference between *functional* vs. *class* components
  - e.g., only class component events are called with **`this`**

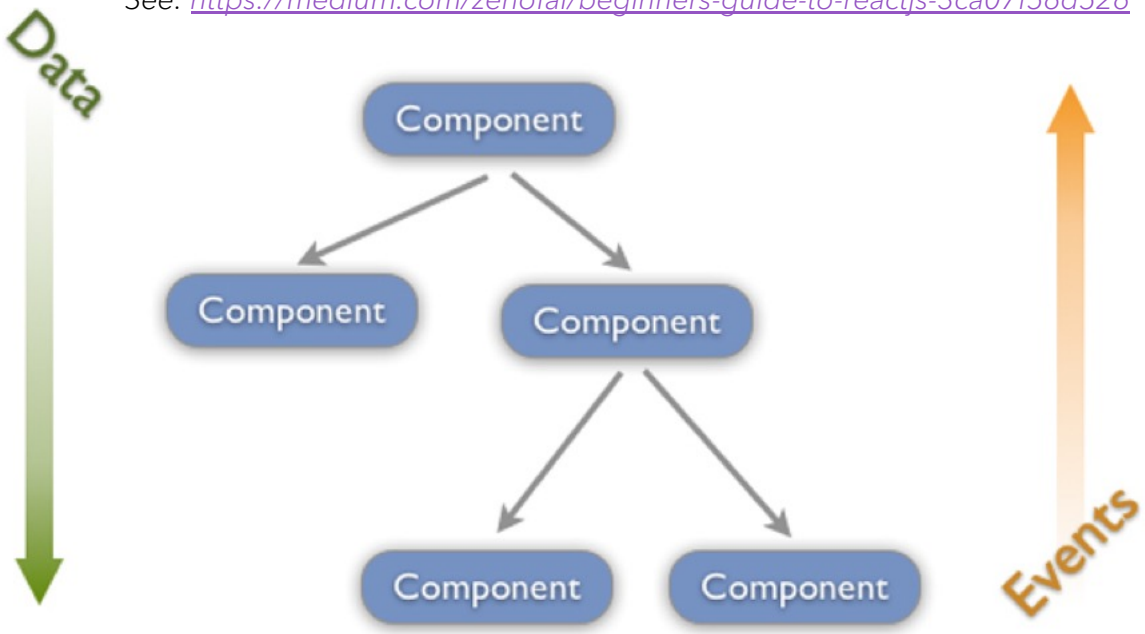
```
class App extends React.Component {  ... }

class Item extends React.Component {
  handleClick() {
    alert("click");
  }
  render() {
    return <div class="box" onClick={this.handleClick}>
             {this.props.subject}</div>;
  }
}
```

UNIDIRECTIONAL DATA FLOW

- *"Properties flow down; actions flow up"*
  - Data are passed to children as *props*
  - Events are handled by parents, as the handler has been passed as props
  - If information needs to be passed to the parent, the technique of "*lifting state up*" could be used

See: https://medium.com/zenofai/beginners-guide-to-reactjs-3ca07f56d526



CONDITIONAL RENDERING

- It is common to decide whether something should be displayed based on a *boolean*
  - *`condition ? true : false`*
  - `if-else` structure

```
render() {
  const isLoggedIn = this.state.isLoggedIn;
  return (
    <div>
      User is <b>{isLoggedIn ? 'currently' : 'not'}</b> logged in.
    </div>
  );
}
```

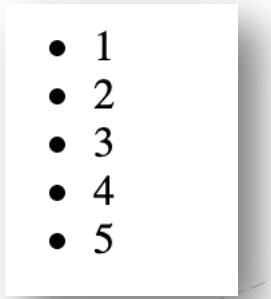
See: https://reactjs.org/docs/conditional-rendering.html

LISTS AND KEYS

- You can easily loop through arrays to create lists
  - A **key** is usually generated for the ReactDOM to identify items and check whether they are modified
- *See: https://reactjs.org/docs/lists-and-keys.html*

https://codepen.io/gaearon/pen/jrXYRR

```
const numbers = [1, 2, 3, 4, 5];
const listItems = numbers.map((number) =>
  <li key={number.toString()}>{number}</li>
);
. . .
. . .
ReactDOM.render(
  <ul>{listItems}</ul>,
  document.getElementById('root')
);
```



FORMS

- React prefers ***controlled components*** instead of HTML default behaviour for forms
  - Single source of truth for form *contents and rendering*, e.g.
    - `handleChange()` will decide what should happen when the form element has new input
    - `handleSubmit()` will decide what should happen when the form is submitted
  - `event.preventDefault()` to avoid default actions (e.g., submit) handled by the browser
- *See: https://reactjs.org/docs/forms.html*

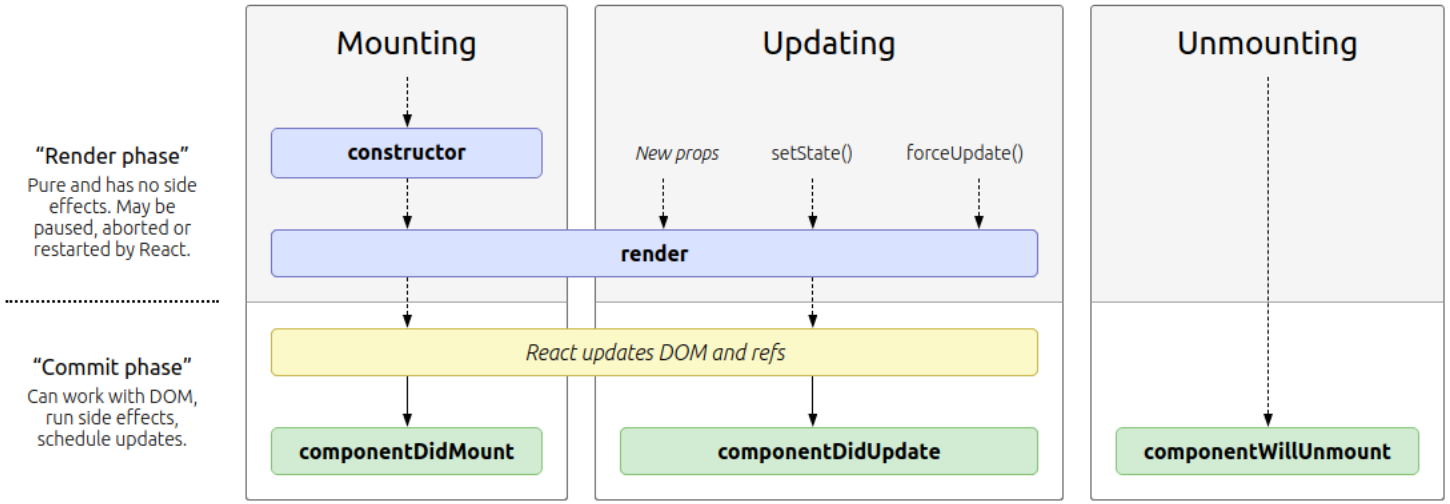
FORMS

https://codepen.io/gaearon/pen/VmmPgp

```
class NameForm extends React.Component {
  constructor(props) {
    super(props);
    this.state = {value: ''};
    this.handleChange = this.handleChange.bind(this);
    this.handleSubmit = this.handleSubmit.bind(this);
  }
  handleChange(event) { this.setState({value: event.target.value}); }
  handleSubmit(event) {
    alert('A name was submitted: ' + this.state.value);
    event.preventDefault();
  }
  render() { return (
    <form onSubmit={this.handleSubmit}>
      <label>
        Name: <input type="text" value={this.state.value} onChange={this.handleChange} />
      </label>
      <input type="submit" value="Submit" />
    </form>
  ); }
}
```

LIFECYCLE METHODS

- Lifecycle of a React component
  - Mounting → Updating → Unmounting
- These methods will be called when the time arrives



See: https://medium.com/zenofai/beginners-guide-to-reactjs-3ca07f56d526

LIFECYCLE METHODS

- The lifecycle methods are useful to insert your own functionalities in the component's lifecycle
  - `componentWillMount()` / `componentDidMount()`
  - `componentWillUpdate()` / `componentDidUpdate()`
  - `componentWillReceiveProps()`
  - `componentWillUnmount()`
- *See: https://www.newline.co/fullstack-react/30-days-of-react/day-7/*

LEARN MORE FOR REACT

- React Router
  - Deciding what to display based on URL in a single-page app (SPA)
  - *See: https://www.freecodecamp.org/news/react-router-in-5-minutes/*
- React-Redux
  - State manager for communication between objects
  - *See: https://medium.com/@christiannaths/from-zero-to-redux-8db779b6ed01*
- React Native
  - Build UI on iOS and Android using React and JSX
  - *See: https://itnext.io/from-react-to-react-native-what-you-need-to-know-to-jump-ship-61320df96557*

READ FURTHER…

Beginner's Guide to ReactJS (v16)

https://medium.com/zenofai/beginners-guide-to-reactjs-3ca07f56d526

Getting Started with React

https://reactjs.org/docs/getting-started.html