香 港 中 文 大 學
The Chinese University of Hong Kong

CSCI2720 - Building Web Applications

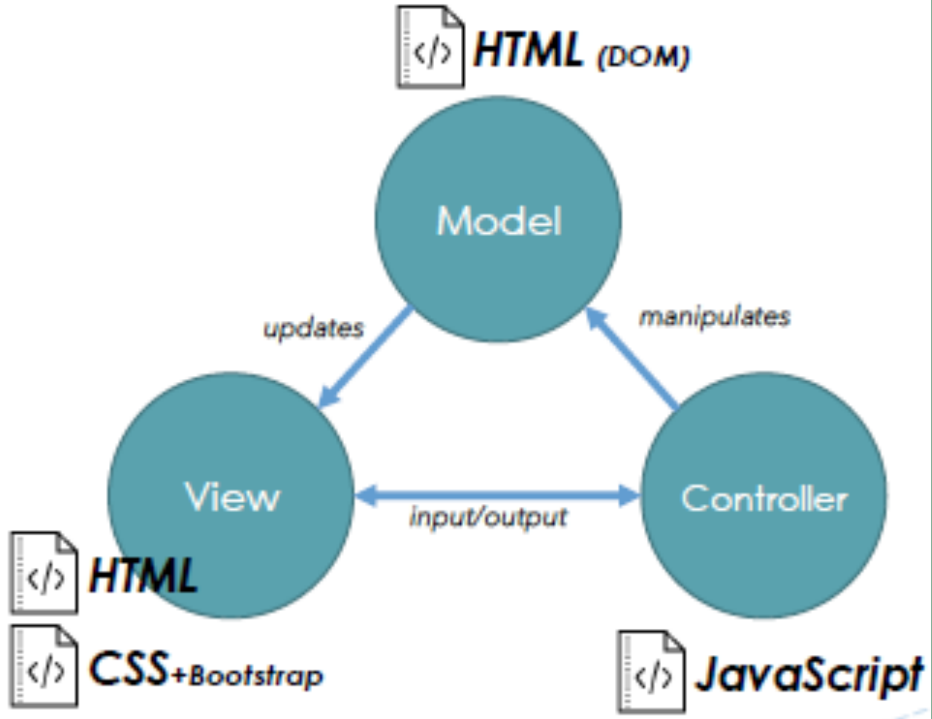
Lecture 10: ReactJS

Dr Colin Tsang

Outline

Basics of the Web

- Markup + Styling + Scripts = HTML + CSS + JavaScript
- Many modern libraries or frameworks help you *generate* these. (i.e., we can be lazy!)



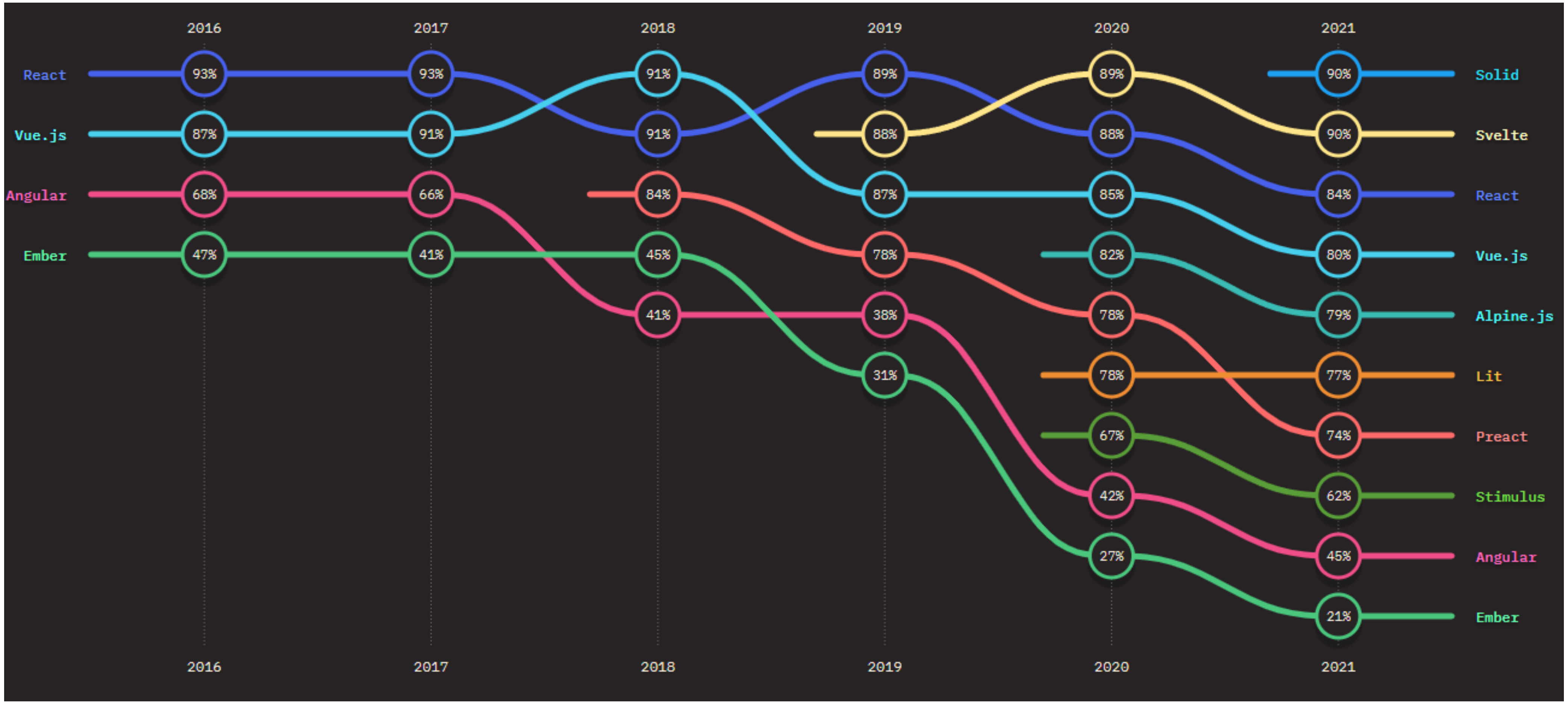
Transpiling

- Web standards: HTML, CSS, JavaScript
- Enhancement and syntactic sugar – make programming easier
  - Template engines: easily generated HTML, e.g., Emmet
  - CSS preprocessors: Sass or Less
  - JavaScript flavors: TypeScript, JSX, CoffeeScript, …
    - See: https://www.digitalocean.com/community/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

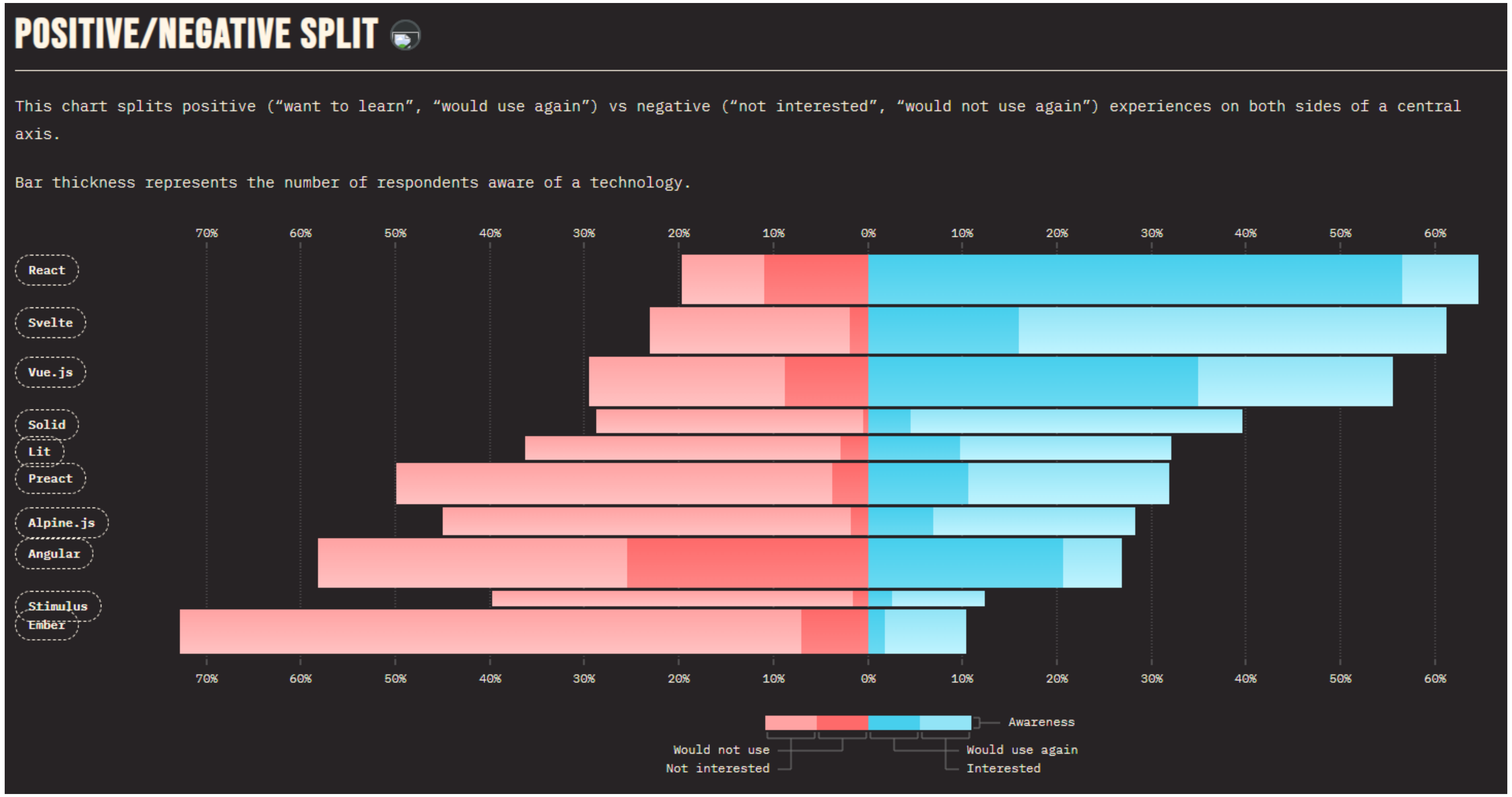
- See: https://hackernoon.com/angular-vs-react-vs-vue-which-is-the-best-choice-for-2019-16ce0deb3847
- See: https://www.codeinwp.com/blog/angular-vs-vue-vs-react/

Frontend frameworks and libraries



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

Frontend frameworks and libraries



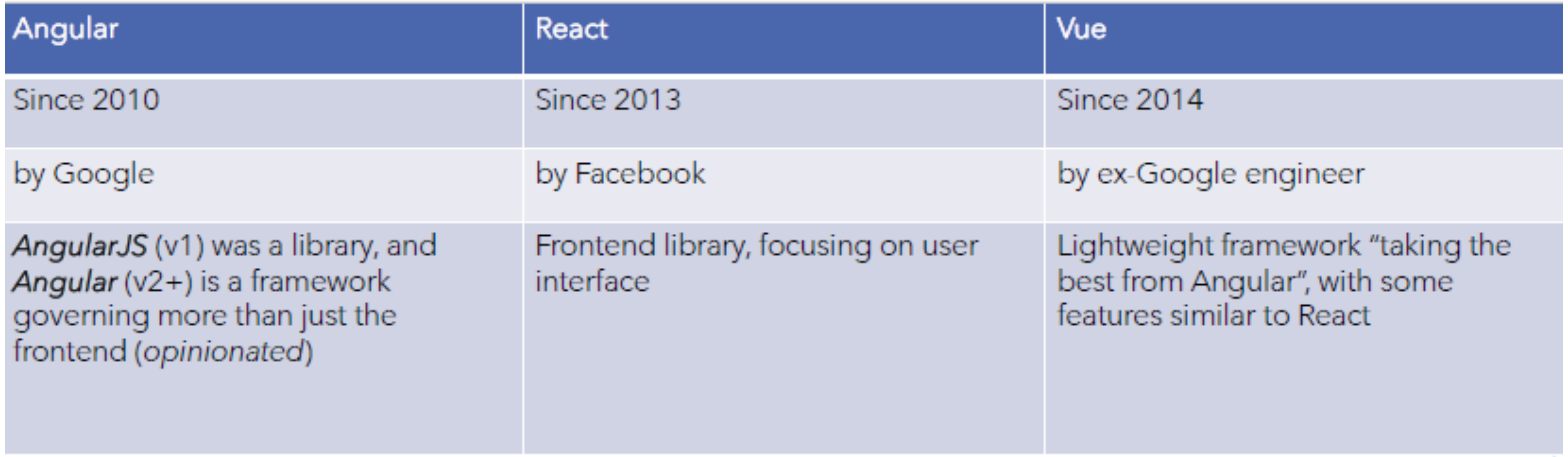
POSITIVE/NEGATIVE SPLIT

This chart splits positive ("want to learn", "would use again") vs negative ("not interested", "would not use again") experiences on both sides of a central axis.

Bar thickness represents the number of respondents aware of a technology.

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

Frontend frameworks and libraries



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

React

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

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
- See: https://www.freecodecamp.org/news/best-react-javascript-tutorial/

What does React give you?

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

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

Starting with React

- There two ways to get React into your website:

- Embedding React using **<script>**:
  - Easier setup but is not optimized for redarning.
  - No special commands needed, no need for *import* in JS.
  - *We will be using this method in this lecture*.

- JavaScript toolchains
  - Some more preparation, but allows automated 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 <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>
```

} Must import

} Recommended (for JSX)

- 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/

The first example

- You can pass the DOM control of your HTML to reactDOM by specifying an element with ID.

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

- The element with *id=app* will be updated by React automatically.
- Full demo is available on Blackboard.

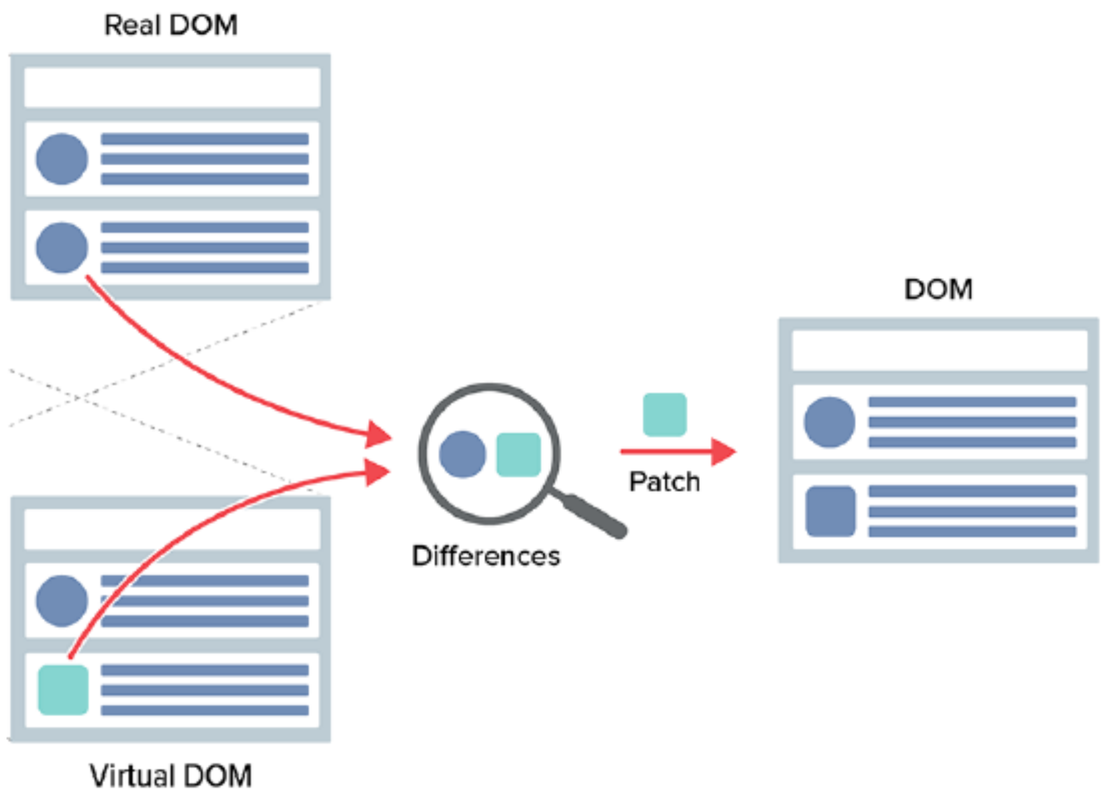
The virtual DOM

- 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.

- *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 us updated with only the calculated difference



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 to old browsers
- JSX is used as type *"text/babel"* and is usually stored with file name **.jsx**
  - Using **.js** is usually fine
- JSX produces *React elements*, neither HTML nor string
- Learn more: https://reactjs.org/docs/introducing-jsx.html

HTML + JS + JSX

```
<script type="text/babel">
    function formatName(u) {
    return u.firstName + ' ' + u.lastName;
    }
    const user = { firstName: 'WebApp', lastName: 'CUHK' };
    const element = <h1>Hello, {formatName(user)}! I am created by JSX!</h1>;
    const root = ReactDOM.createRoot(document.querySelector('#app'));
    root.render(element);
</script>
```

Using CSS in React

- Important warning: writing JSX is not directly writing HTML, so some HTML attributes could be different.
- To use inline styles in JSX with a style attribute, special syntax is required:

```
const myStyle = {
      color: 'blue',
      fontSize: '24px',
    };
const element = <h1 style={myStyle}>Hello, I am created by JSX!</h1>;
```

- In React without JSX:

```
const element = React.createElement("h1", {style: myStyle}, "Hello, I am from React with
CSS.");
```

- Read more: https://legacy.reactjs.org/docs/dom-elements.html

Components

$$\begin{tikzcd} & \text{Functional} \\ \text{Components} \arrow[ur, dash] \arrow[dr, dash] & \\ & \text{Class (More used)} \end{tikzcd}$$

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

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

Class component

- People usually use class component instead of functional 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 />);
```

going to be displayed

Props

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

```
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 />);
```

define props

States

- The behaviour of a component at a given moment in time is defined by the *state*
- Values in the state should only be updated using **this.setState()**
  - Just usual JS *key: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
- State is mutable.

Using State

Required to define state

```
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/>);
```

← Define state

Using States

- *this.state* inside the class is a class object accessible in the class scope
- Values can be read by calling *this.state.key*
- To change the state value, it must be through **this.setState()**

- See the *state_demo.html* from Blackboard

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 the event

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

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

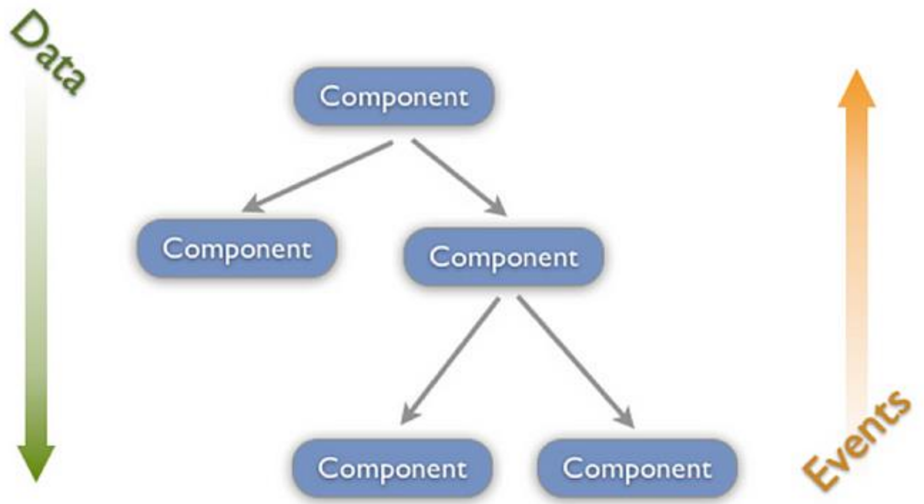
Events

- Always mind the subtle difference between functional vs class components
  - Only class component events are called with this

```
class ActionLink extends React.Component {
  handleClick(e) {
    e.preventDefault();
    console.log('The link was clicked.');
  }

  render() {
    return (
      <a href="#" onClick={this.handleClick}>
        Click me
      </a>
    );
  }
}
```

U~~n~~idirectional 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 (not discussed in this course)
  - 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

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

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

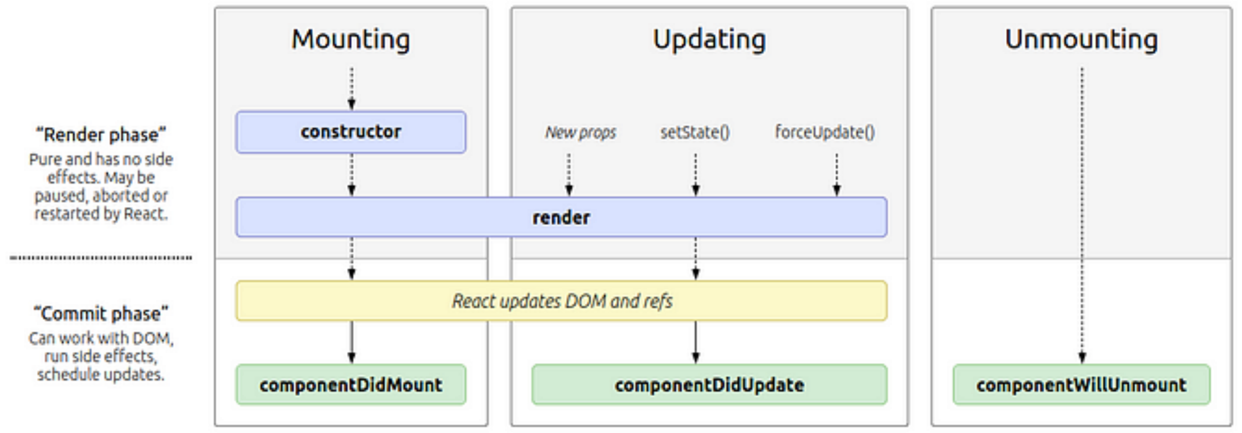
```
const numbers = [1, 2, 3, 4, 5];
const listItems = numbers.map((number) =>
   <li key={number.toString()}>{number}</li>
);


ReactDOM.render(
   <ul>{listItems}</ul>,
   document.getElementById('root')
);
```

```
• 1
• 2
• 3
• 4
• 5
```

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 browser
- Check blackboard demo
- Two major advantages:
  - You can bypass the default actions
  - The user input is stored into *state*
- See: https://reactjs.org/docs/forms.html

Lifecycle methods



- Lifecycle of a react component
  - Mounting -> updating -> unmounting
- *Mounting* is the process of creating an instance of a component and inserting it into DOM.
- *Updating* is the process of making changes to a component's *state* or *props* and reflecting those changes in the rendered output.
- *Unmounting* is the process of removing a component instance from the DOM.

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()**

- Learn more in: https://www.newline.co/fullstack-react/30-days-of-react/day-7/

I cannot cover everything in this course……

- 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

Further reading

- Beginner’s guide to ReactJS (v16)
    - https://medium.com/zenofai/beginners-guide-to-reactjs-3ca07f56d526

- Getting started with React
    - https://legacy.reactjs.org/docs/getting-started.html