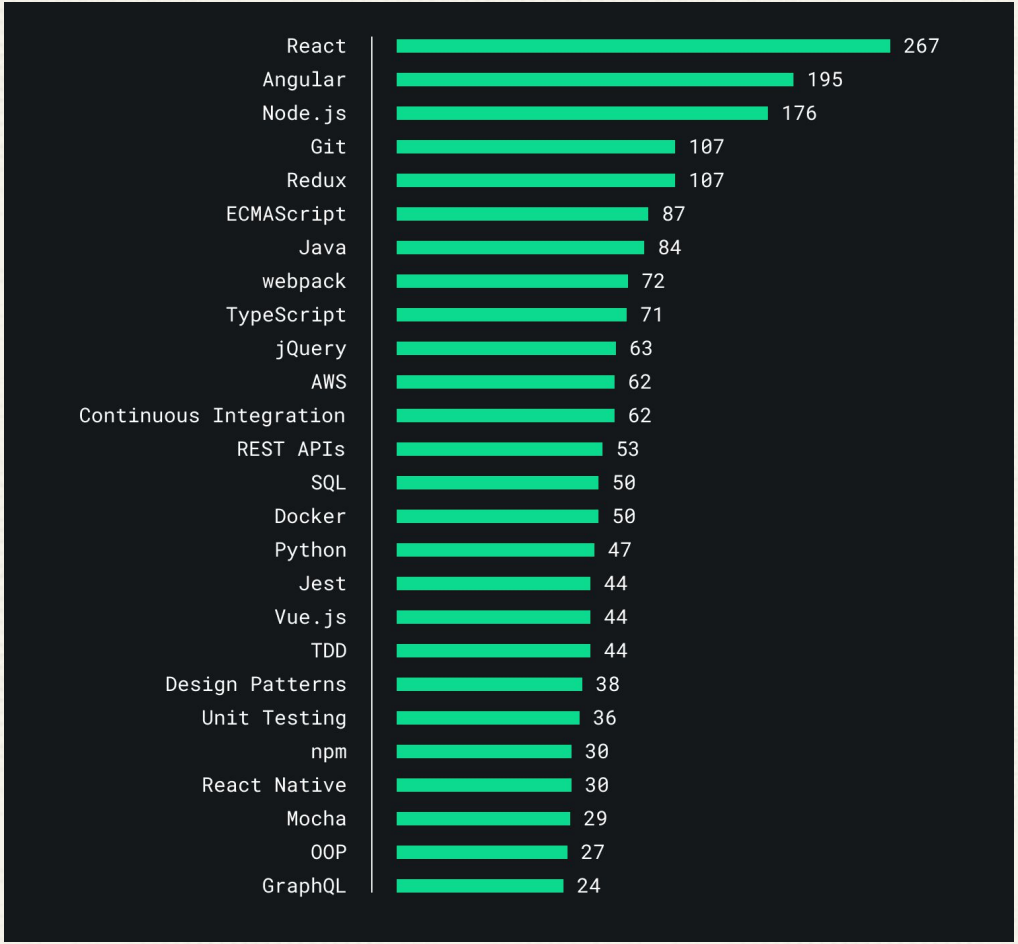
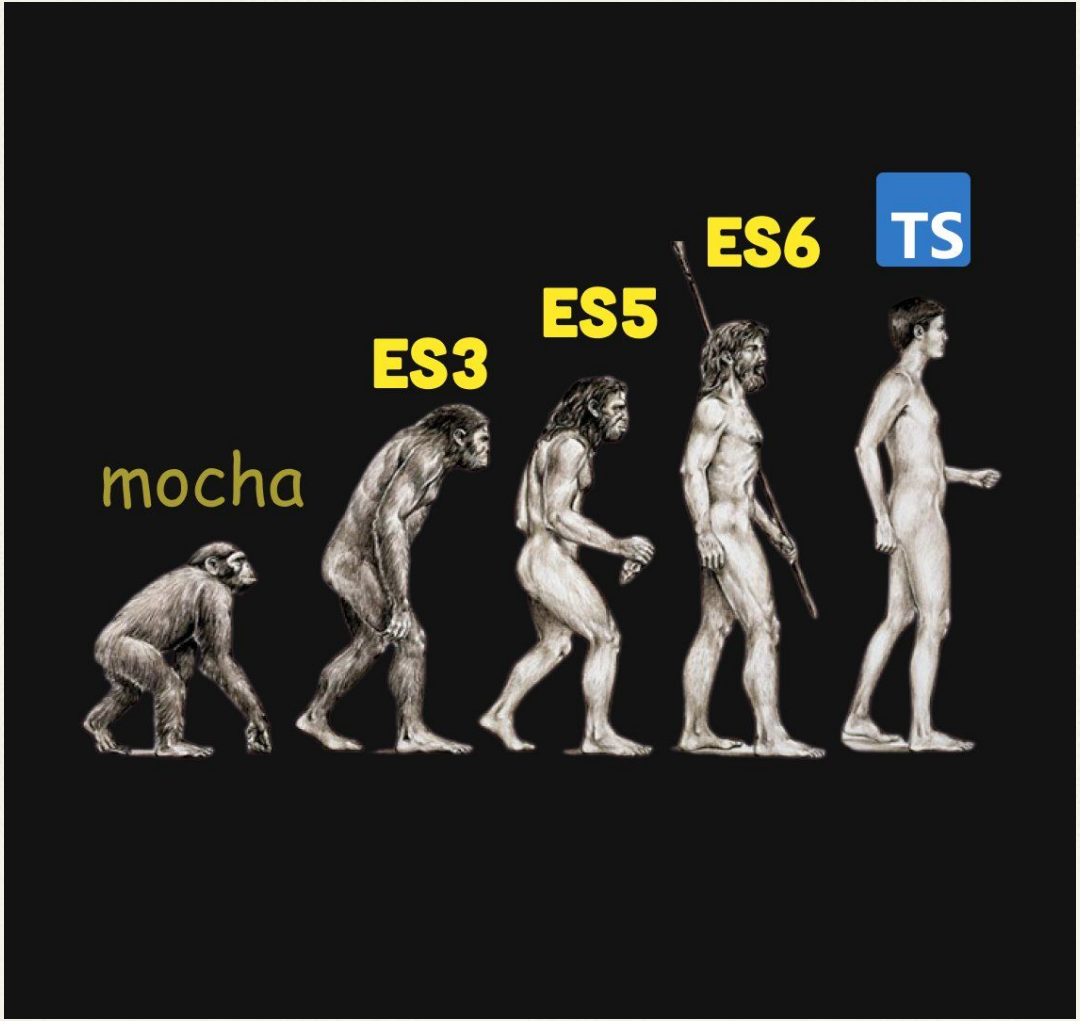
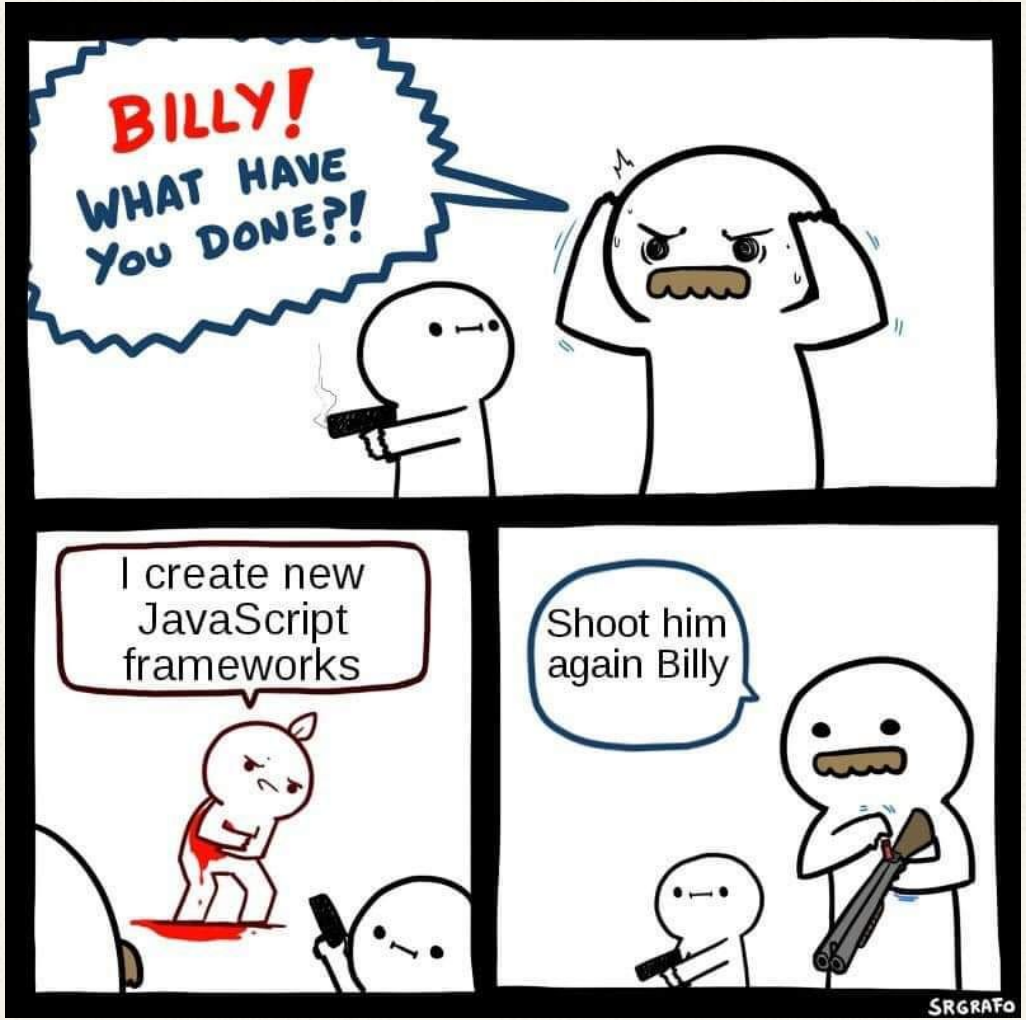
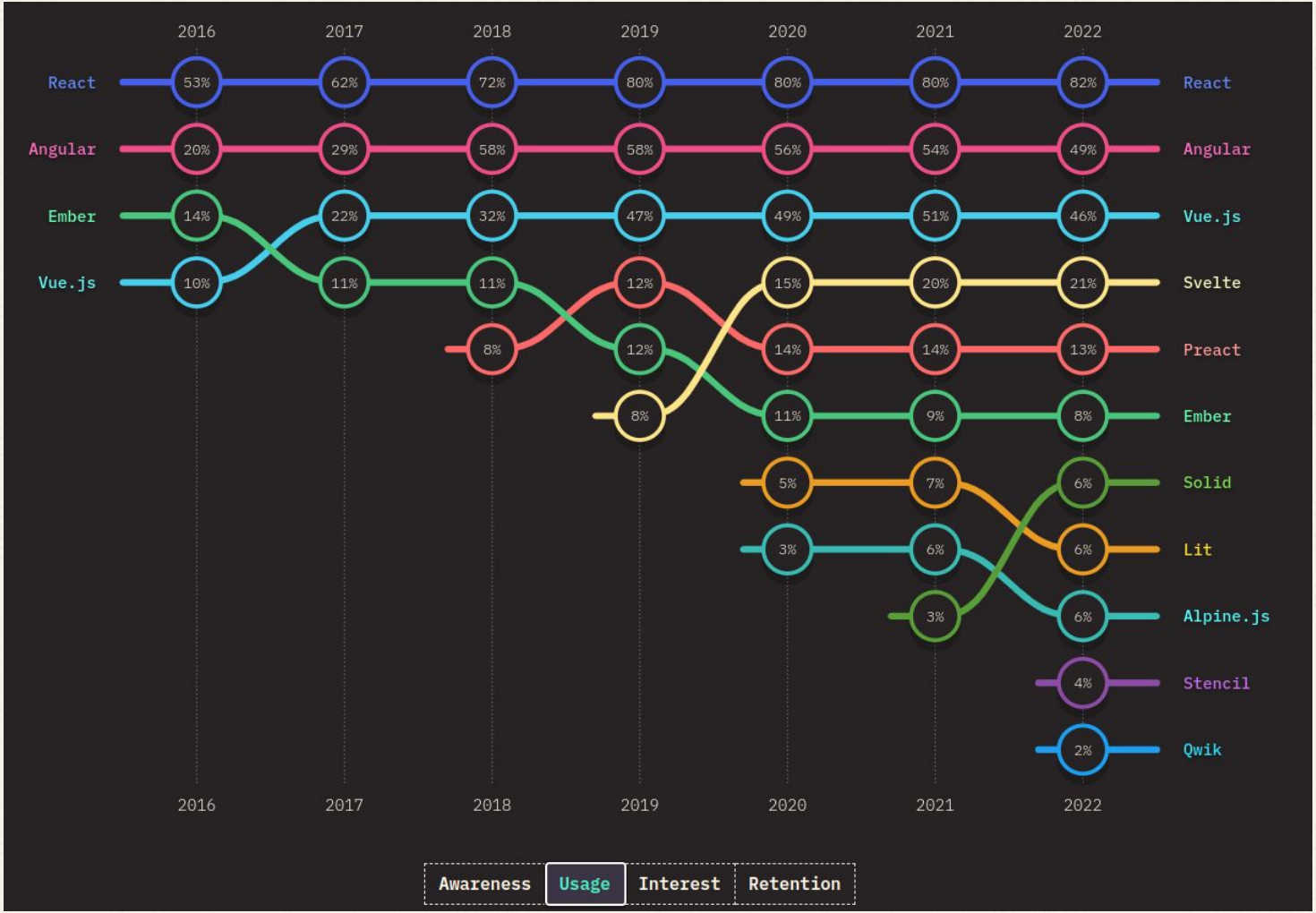
React workshop


React 267
Angular 195
Node.js 176
Git 107
Redux 107
ECMAScript 87
Java 84
webpack 72
TypeScript 71
jQuery 63
AWS 62
Continuous Integration 62
REST APIs 53
SQL 50
Docker 50
Python 47
Jest 44
Vue.js 44
TDD 44
Design Patterns 38
Unit Testing 36
npm 30
React Native 30
Mocha 29
OOP 27
GraphQL 24


mocha
ES3
ES5
ES6
TS


BILLY!
WHAT HAVE YOU DONE?!
I create new JavaScript frameworks
Shoot him again Billy
SRGRAFO


2016
2017
2018
2019
2020
2021
2022
React
Angular
Ember
Vue.js
Svelte
Preact
Solid
Lit
Alpine.js
Stencil
Qwik
53%
62%
72%
80%
80%
80%
82%
20%
29%
58%
58%
56%
54%
49%
14%
22%
32%
47%
49%
51%
46%
10%
11%
11%
12%
15%
20%
21%
8%
12%
14%
14%
13%
8%
11%
9%
8%
5%
7%
6%
3%
6%
6%
3%
6%
4%
2%
Awareness
Usage
Interest
Retention

React advantages

- Easy to learn and use
- Creating dynamic Web Applications becomes easier
- Reusable components
- Performance enhancement
- The support of handy tools
- Known to be SEO friendly
- The benefit of having JavaScript library
- Scope for testing the code

React disadvantages

- The high pace of development
- Poor Documentation
- View Part
- JSX as a barrier

Array destructuring

```
const foo = ['one', 'two', 'three'];

const [red, yellow, green] = foo;
let [x, y, z, q] = foo;

[x, y] = [y, x];
```

Object destructuring

```
const user = {
  id: 42,
  isVerified: true
};
const {id, isVerified, job = “se”} = user;

function userId({id}) {
  return id;
}
console.log(userId(user));
```

Spread syntax

```
const parts = ['shoulders', 'knees'];
const lyrics = ['head', ...parts, 'and', 'toes'];

const arr = [1, 2, 3];
const arr2 = [...arr]; // like arr.slice()

const arr3 = [0, 1, 2];
const arr4 = [3, 4, 5];

arr5 = [...arr3, ...arr4];
```

Spread syntax

```
const obj1 = { foo: 'bar', x: 42 };
const obj2 = { foo: 'baz', y: 13 };

const clonedObj = { ...obj1 };
const mergedObj = { ...obj1, ...obj2 };
```

Array methods

```
// Arrow function
map((element) => { /* ... */ })
map((element, index) => { /* ... */ })
map((element, index, array) => { /* ... */ })


const numbers = [1, 4, 9];
const roots = numbers.map((num) => Math.sqrt(num));
const logs = numbers.map((num, i) => `Number ${num} is at
index ${i}`);
```

Array methods

```
function isBigEnough(value) {
  return value >= 10
}

const filtered = [12, 5, 8, 130, 44].filter(isBigEnough)

const fruits = ['apple', 'banana', 'grapes', 'mango', 'orange']
function filterItems(arr, query) {
  return arr.filter(function(el) {
    return el.toLowerCase().indexOf(query.toLowerCase()) !== -1
  })
}
console.log(filterItems(fruits, 'ap'))
console.log(filterItems(fruits, 'an'))
```

https://github.com/KubiGR/react-todo-workshop

Prerequisites: node

Checkout different branches to follow along

- chapter-1
- chapter-2
- ...
- master

In order to install dependencies, you need to run: npm install

And in order to start the development server, you need to run npm start

Recommended: React Developer Tools (browser extension)

JSX

```
// javascript xml
const myelement = <h1>I Love JSX!</h1>;
const myelement = React.createElement('h1', {}, 'I Love JSX!');
```

JSX

```
// expressions -> { }
return (<h1>React is {5 + 5} times better with JSX</h1>);
return (<img src={user.avatarUrl}></img>);
const myelement = (<h1>Hello, {formatName(user)}!</h1>);
```

JSX

```
// return exactly one DOM element
return (
  <div>
    <h1>I am a Header.</h1>
    <h1>I am a Header too.</h1>
  </div>
);
```

JSX

```
const title1 = React.createElement('h1', {}, 'I am a Header.');

const title2 = React.createElement('h1', {}, 'I am a Header
too.');

const container = React.createElement('div', {}, [title1,
title2]);
```

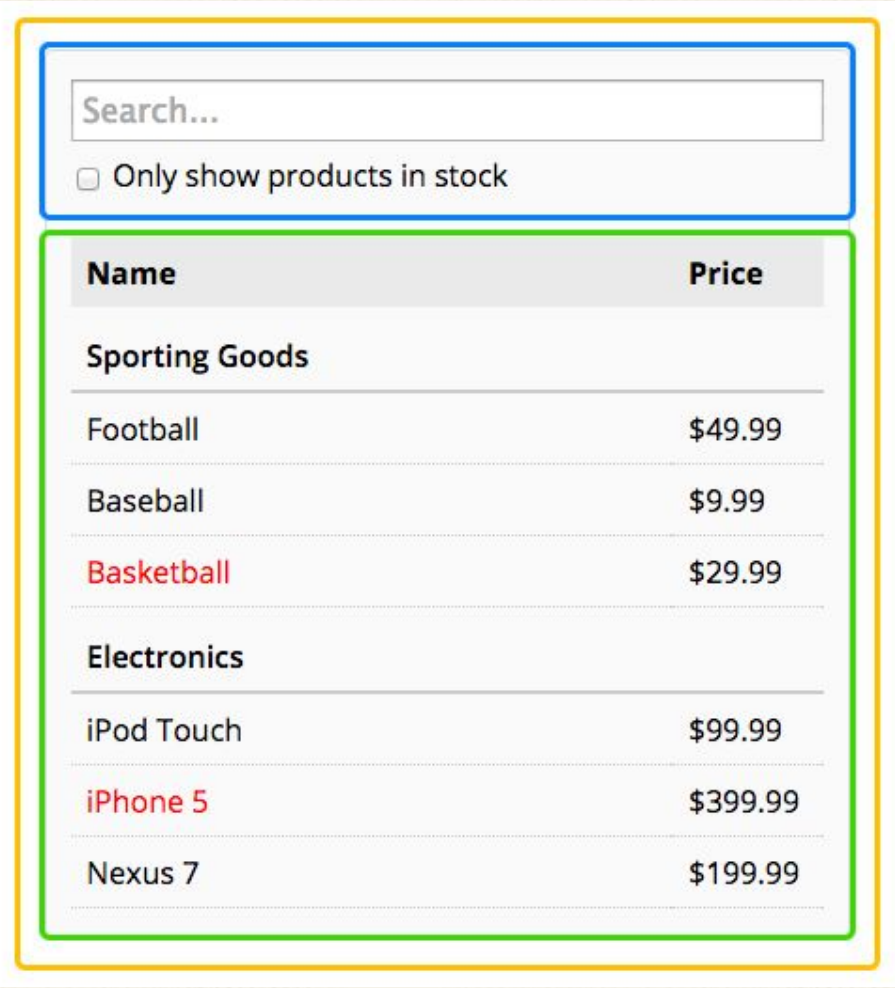
Components

Components are independent and reusable bits of code.

Components come in two types:

- class
- function

React components need to return JSX via a render() function (or return JSX directly, if they are function components).


Search...
Only show products in stock
Name
Price
Sporting Goods
Football
$49.99
Baseball
$9.99
Basketball
$29.99
Electronics
iPod Touch
$99.99
iPhone 5
$399.99
Nexus 7
$199.99

Props

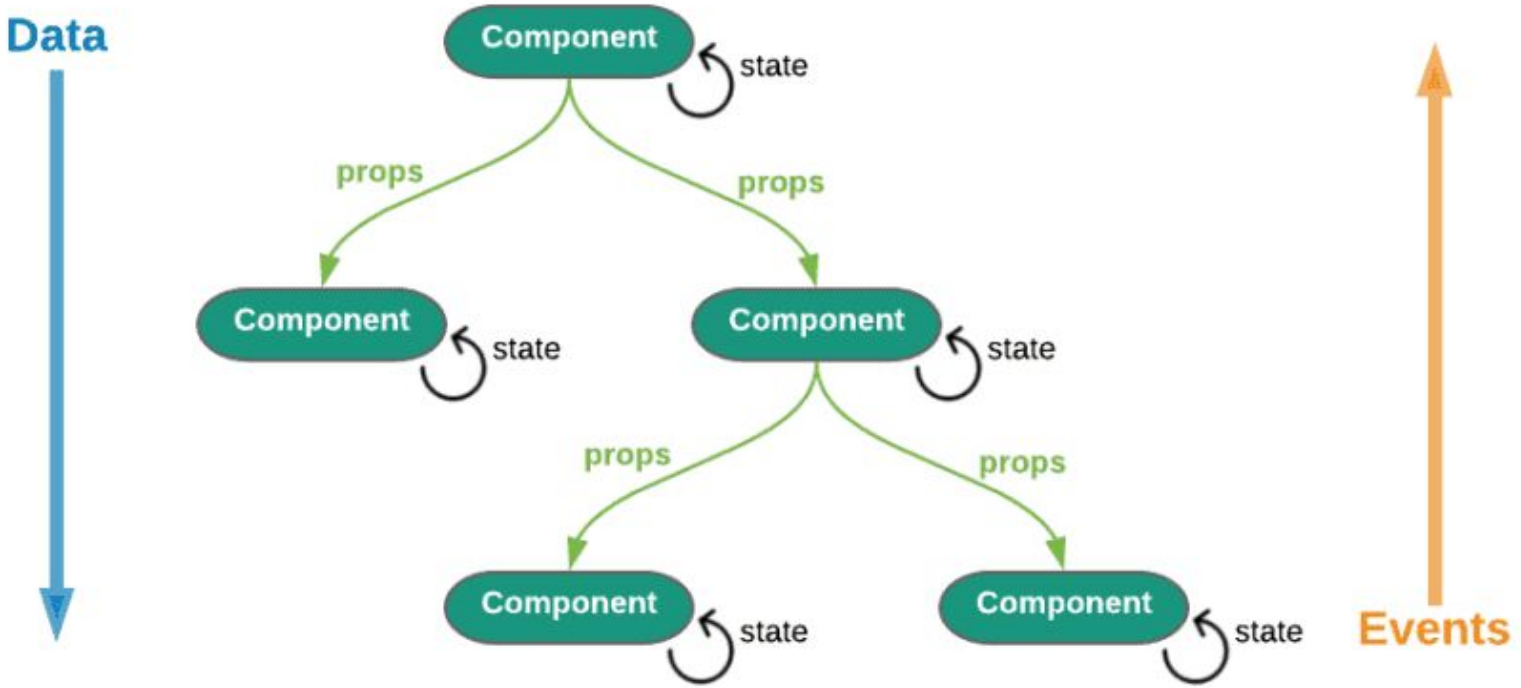
Props are arguments passed into React components.

Props are passed to components via HTML attributes.

Props

```
function Welcome(props) {
  return <h1>Hello, {props.name}</h1>;
}

function Welcome({ name }) {
  return <h1>Hello, {name}</h1>;
}

const element = <Welcome name="Sara" />;
```

React Data flow



State

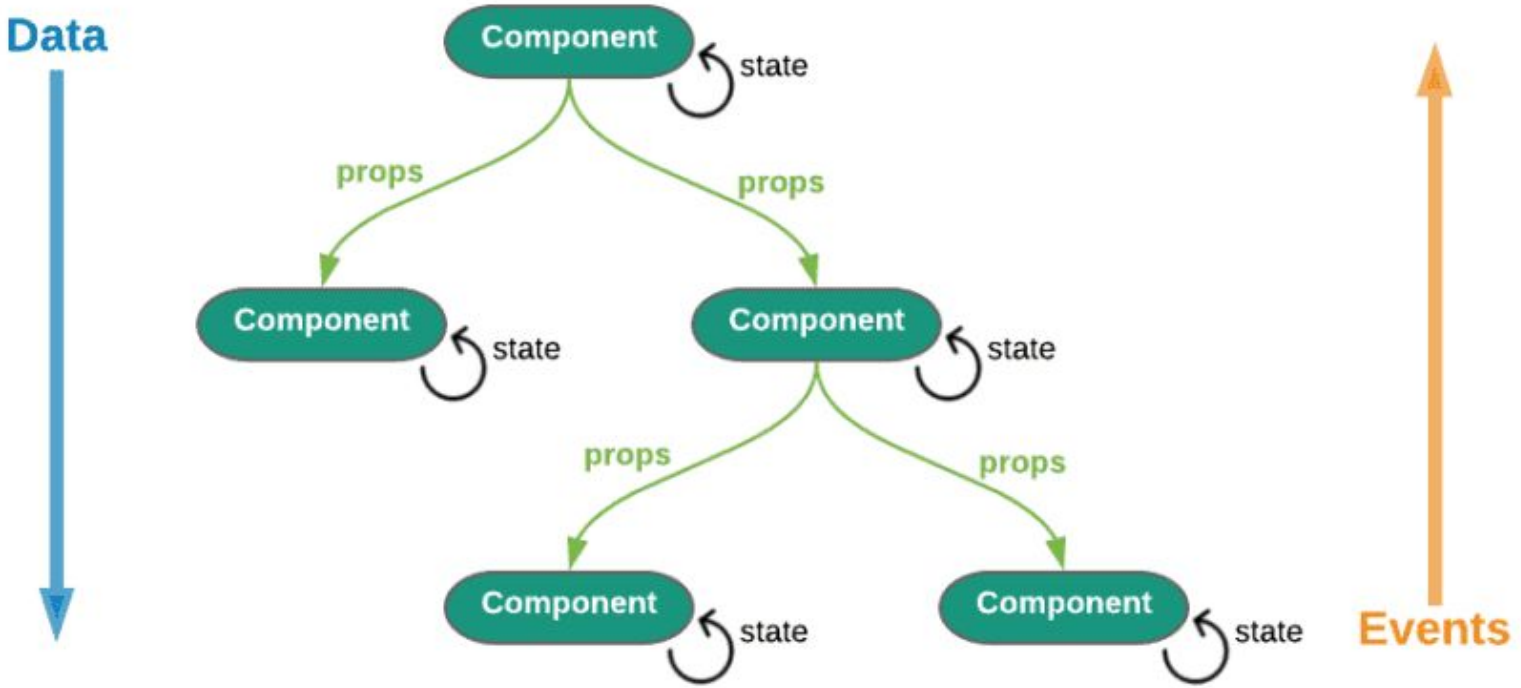
React components have a built-in state object.

The state object is where you store property values that belong to the component.

State

```
function Clock () {
  const [date, setDate] = useState(new Date());

  return (
    <div>
      <h2>
        It is {date.toLocaleTimeString()}.
      </h2>
    </div>
  );
}
```

React Data flow



State dos and don’ts

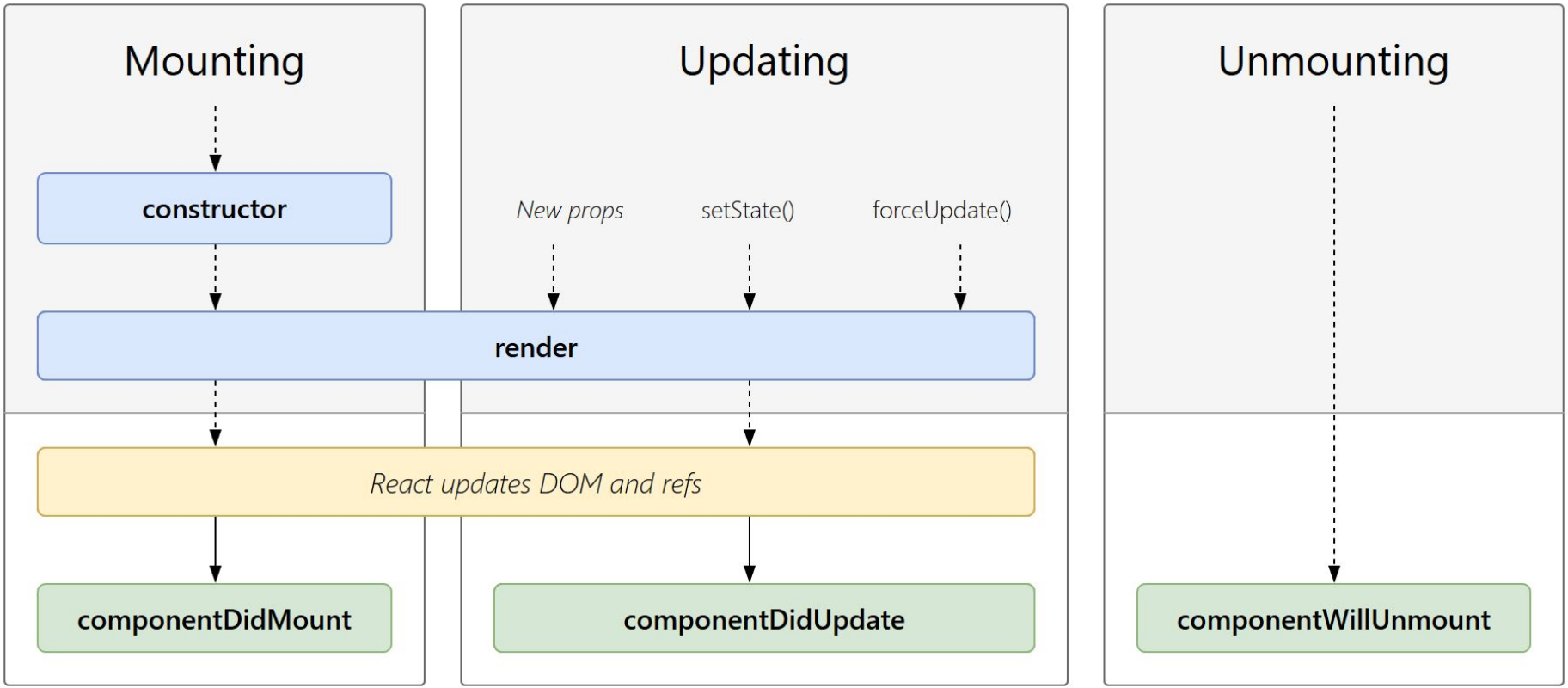
Do Not Modify State Directly (you won't get a re-render)

```
const [comment, setComment] = useState();

comment = 'Hello'; // wrong

setComment('Hello'); // correct
```

State Updates May Be Asynchronous (you depend on invalid data)

React may batch multiple setState() calls into a single update for performance.

```
setCounter(
  counter + increment,
);


setCounter((previousCounter) =>
  previousCounter + increment
);
```


Mounting
constructor
Updating
New props
setState()
forceUpdate()
render
Unmounting
React updates DOM and refs
componentDidMount
componentDidUpdate
componentWillUnmount

Effect hook

```
useEffect(() => {
  document.title = `You clicked ${count} times`;
});

componentDidMount() {
  document.title = `You clicked ${count} times`;
}
componentDidUpdate() {
  document.title = `You clicked ${count} times`;
}
```

Effect hook

```
useEffect(() => {
  ChatAPI.subscribeToFriendStatus(id, handleChange);

  return () => {
    ChatAPI.unsubscribeFromFriendStatus(id, handleChange );
  };
});
```

Ref hook

Essentially, useRef is like a “box” that can hold a mutable value in its .current property.

Mutating the .current property doesn’t cause a re-render.

If you pass a ref object to React, React will set its .current property to the corresponding DOM node whenever that node changes.

Ref hook

```
function TextInputWithFocusButton() {
  const inputEl = useRef(null);
  const onButtonClick = () => {
    inputEl.current.focus();
  };
  return (
    <>
      <input ref={inputEl} type="text" />
      <button onClick={onButtonClick}>
        Focus the input
      </button>
    </>
  );
}
```