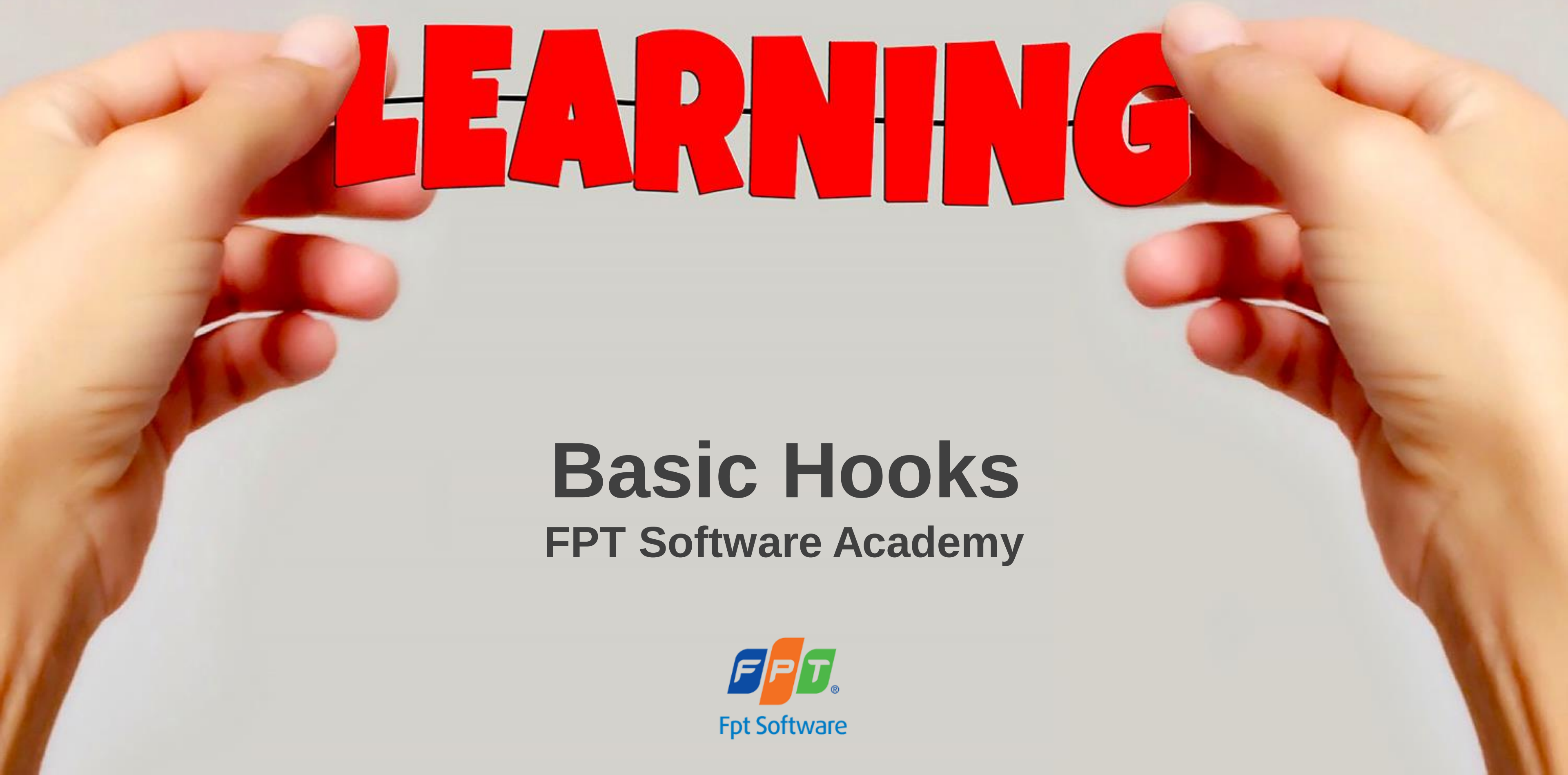


Basic Hooks

FPT Software Academy

Fpt Software

Contents

Fpt Software

1. Overview

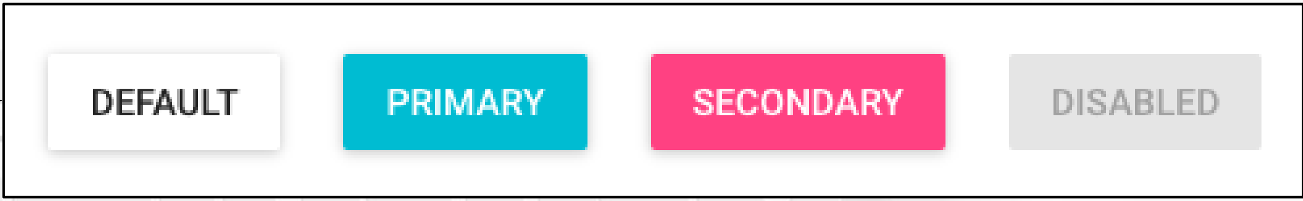
Fpt Software

1. Overview

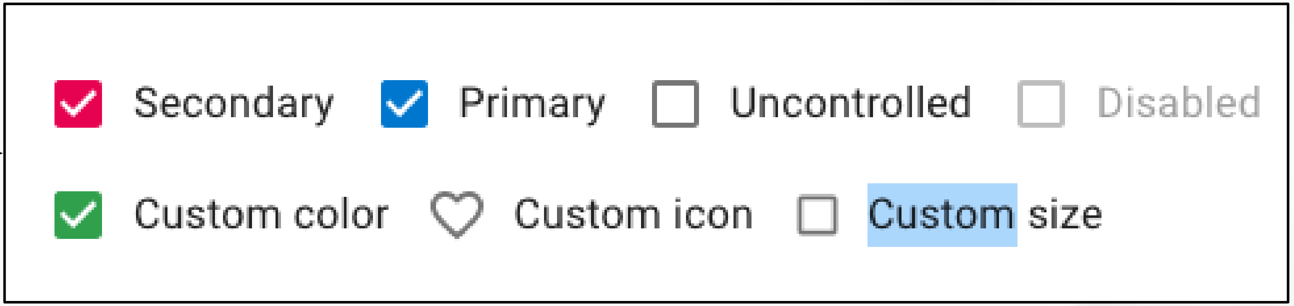
- Recall what is React Component ?
  - Components are building block of every React app
  - Components provide a certain feature (Button, Dropdown)
  - Components are independent and reusable bits of code (HTML, CSS and JS)

1. Overview

Button Component

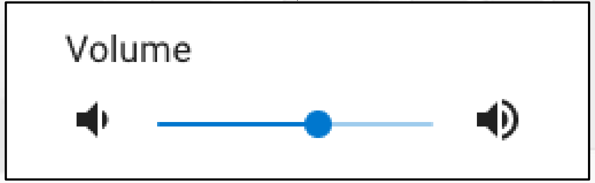


Checkbox Component

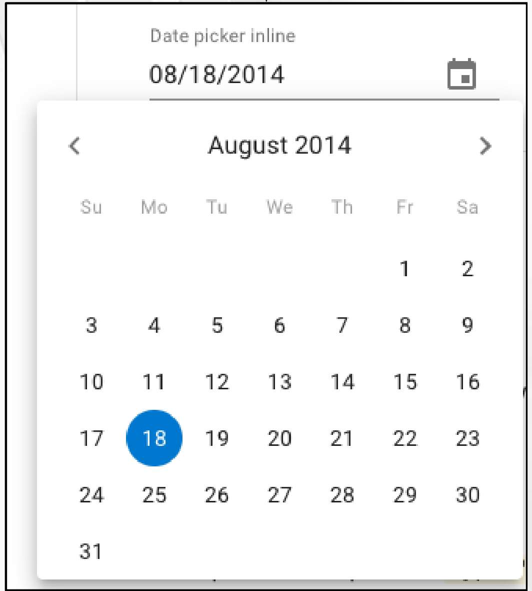


1. Overview

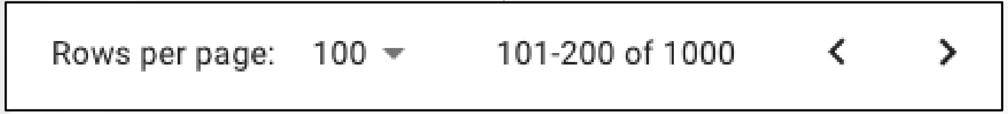
Slider Component



Datepicker Component

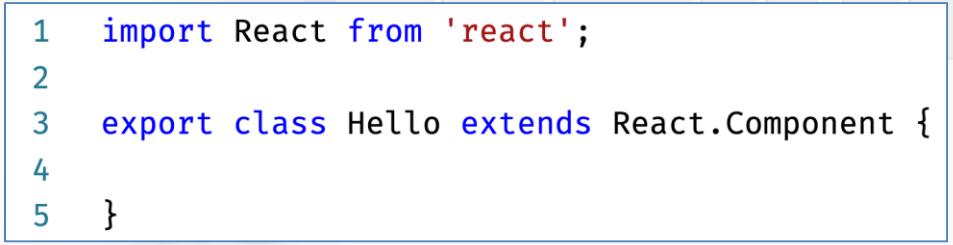


Pagination Component

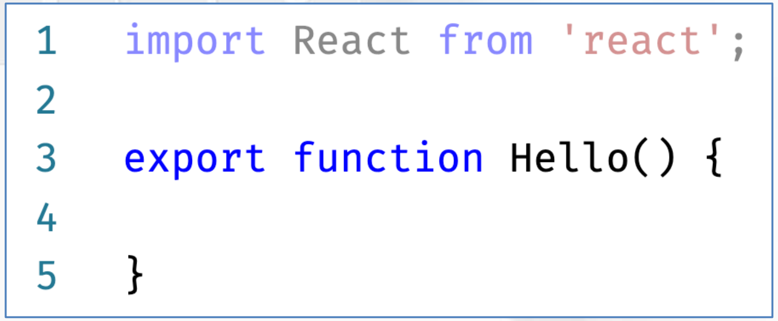


1. Overview

- How many types of Components in React ?
  - Classical Component
  - Functional Component



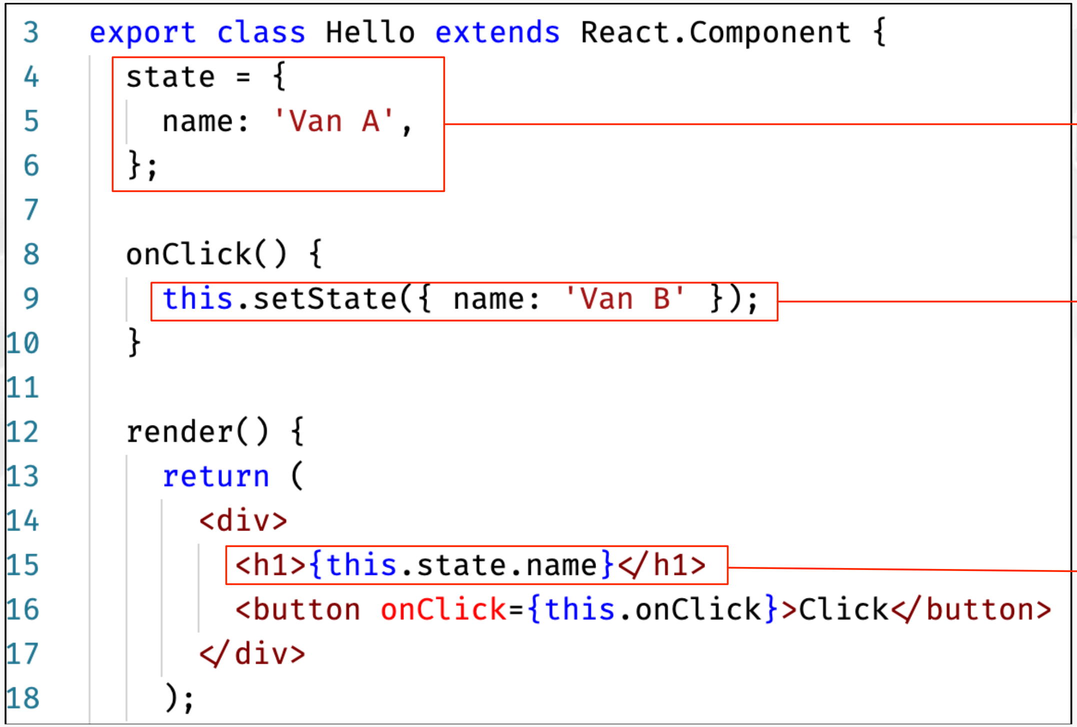
```
import React from 'react';

export class Hello extends React.Component {

}
```



```
import React from 'react';

export function Hello() {

}
```

1. Overview

- How do we store State (data that change overtime) in class Component ?



```
export class Hello extends React.Component {
  state = {
    name: 'Van A',
  };

  onClick() {
    this.setState({ name: 'Van B' });
  }

  render() {
    return (
      <div>
        <h1>{this.state.name}</h1>
        <button onClick={this.onClick}>Click</button>
      </div>
    );
```

Define state

Update state

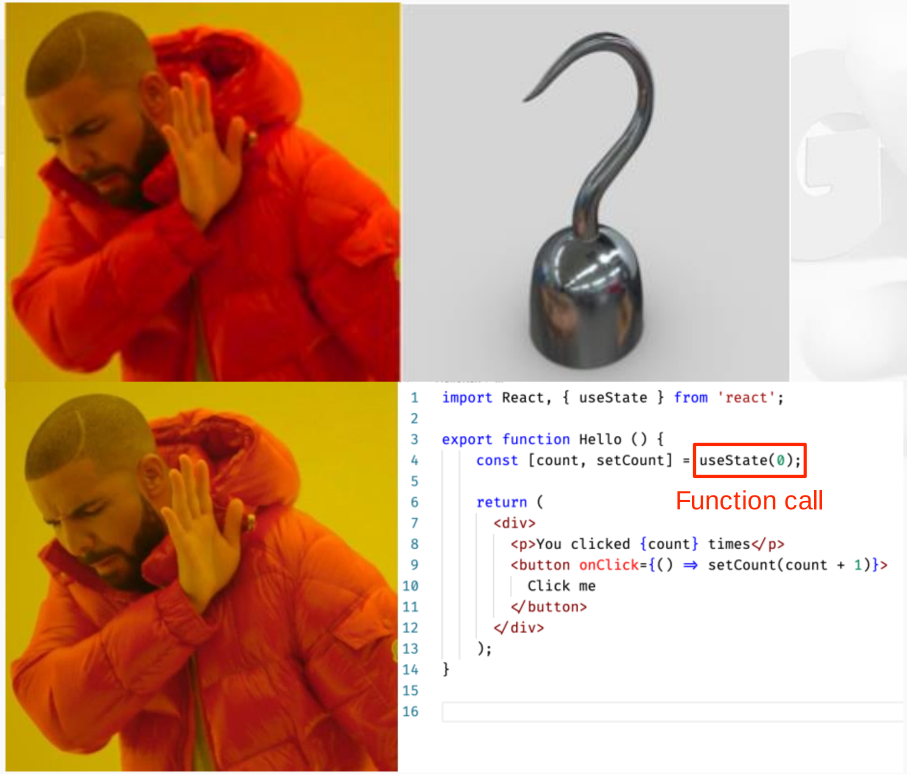
Access state

1. Overview

- Can we use State in Functional Component as well ?
  - Before React 16.8, it’s not possible
  - Good news in 16.8, Hooks are introduced
  - Functional Component can now use state and other features as well

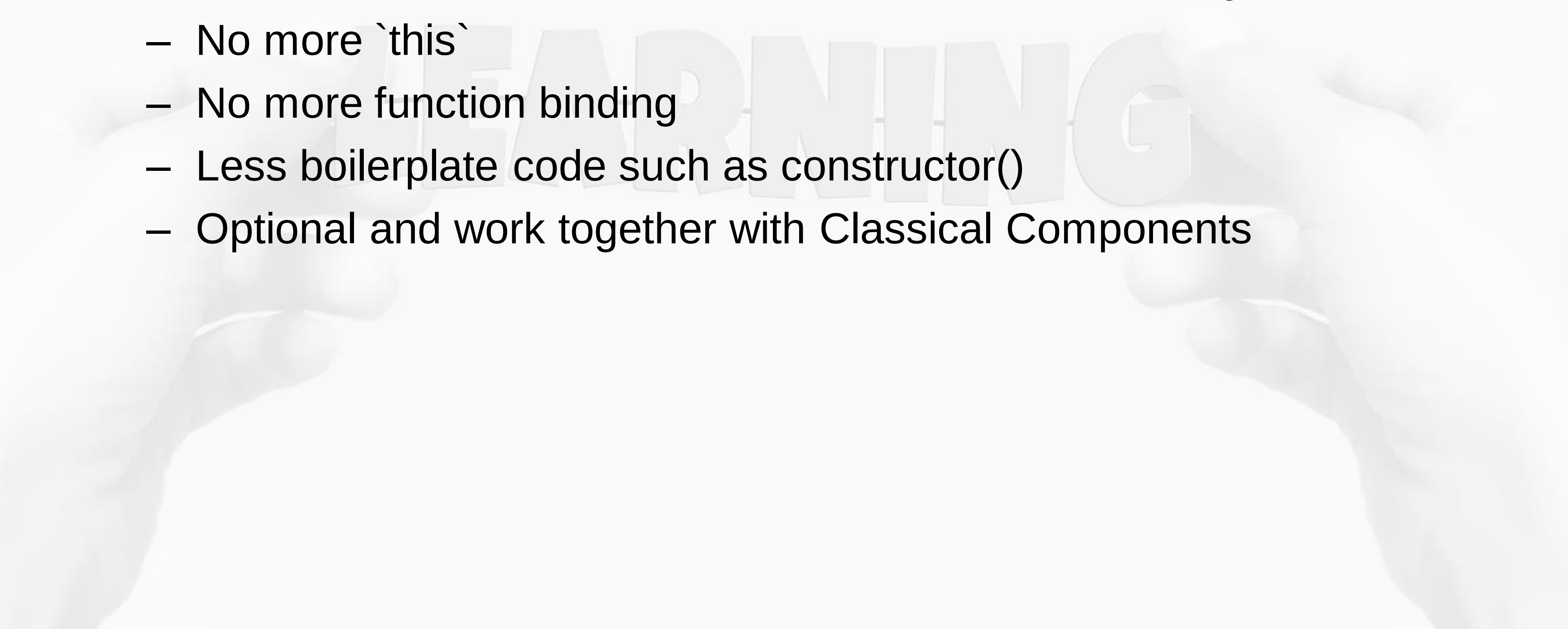
1. Overview

- What does React Hook look like ?



```
import React, { useState } from 'react';

export function Hello () {
    const [count, setCount] = useState(0);

    return (
      <div>
        <p>You clicked {count} times</p>
        <button onClick={() => setCount(count + 1)}>
          Click me
        </button>
      </div>
    );
}
```

1. Overview



- What are the benefits of Hooks over Classical syntax ?
  - No more `this`
  - No more function binding
  - Less boilerplate code such as constructor()
  - Optional and work together with Classical Components

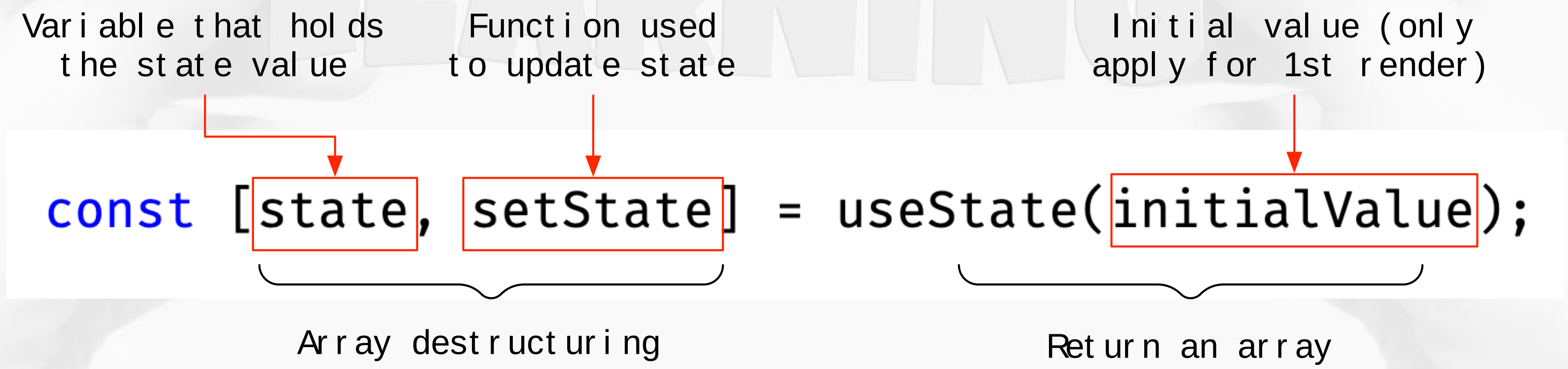
1. Summary

- Hooks are new features of React since version 16.8
- Hooks allow Functional Component to use State and other features such as componentDidMount
- Some benefits of Hooks over Classical Component:
  - No this
  - No function binding
  - Less boilerplate code
- Deep down, Hooks are function (Higher-order function)

2. useState

1. useState

- Allow Classical Component to have State
- Syntax:



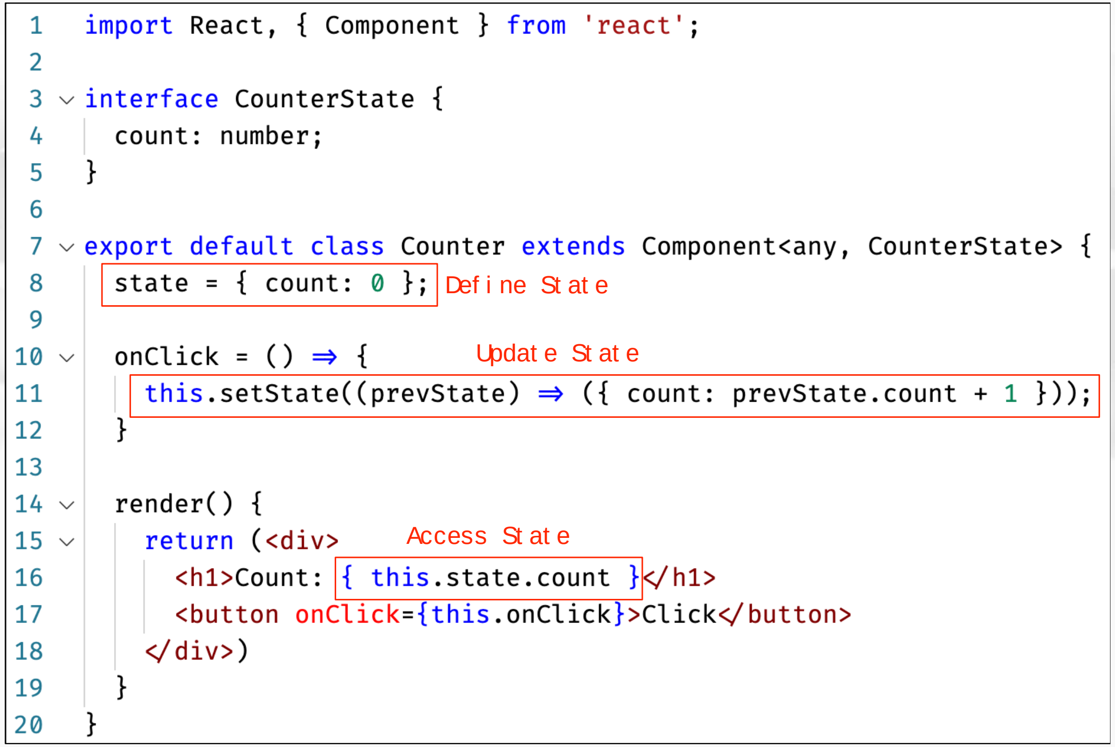
```
const [state, setState] = useState(initialValue);
```

Variable that holds the state value
Function used to update state
Initial value (only apply for 1st render)
Array destructuring
Return an array

Demo useState()

1. useState

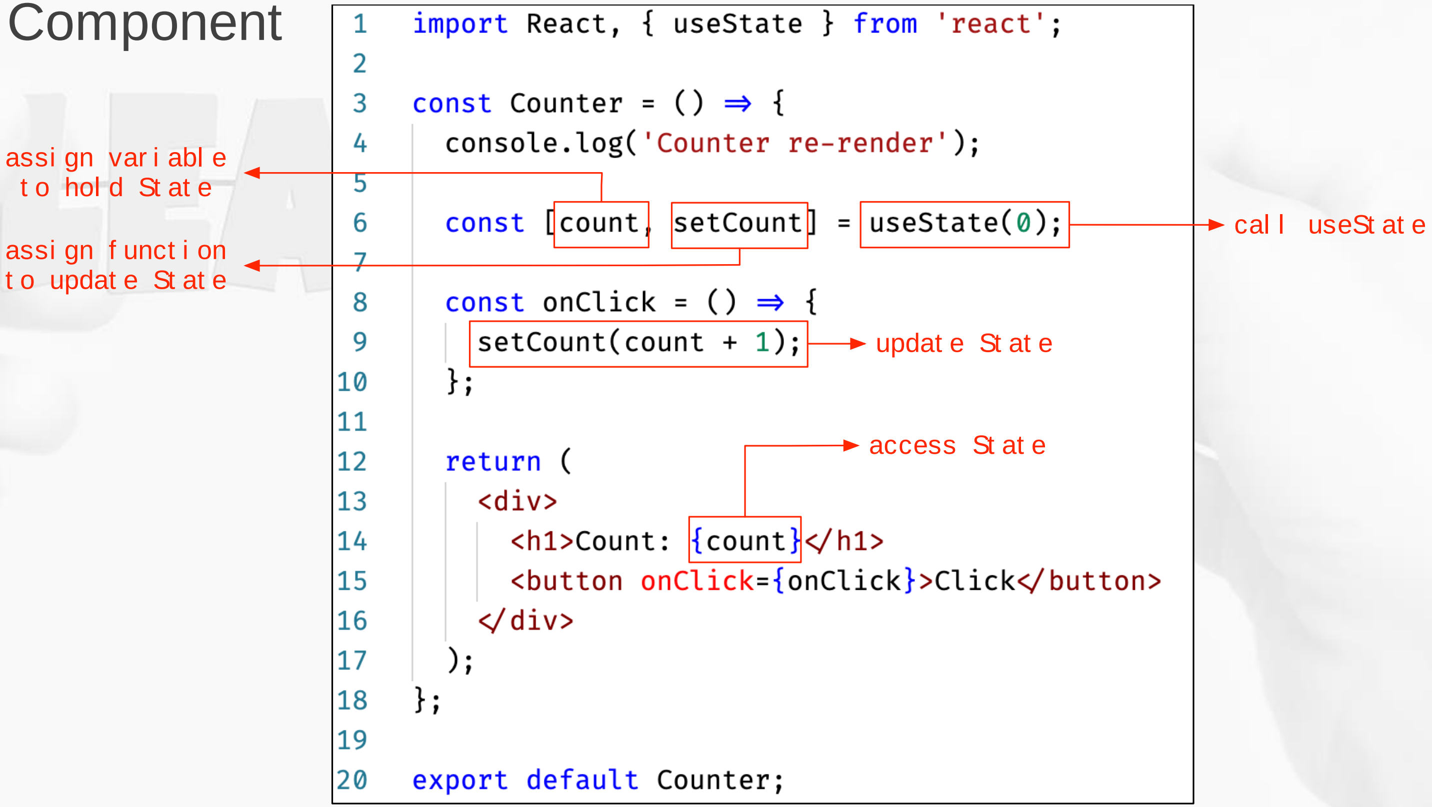
- Class Component



```
import React, { Component } from 'react';

interface CounterState {
  count: number;
}

export default class Counter extends Component<any, CounterState> {
  state = { count: 0 };

  onClick = () => {
    this.setState((prevState) => ({ count: prevState.count + 1 }));
  }

  render() {
    return (<div>
      <h1>Count: { this.state.count }</h1>
      <button onClick={this.onClick}>Click</button>
    </div>)
  }
}
```

1. useState

- Functional Component



```
import React, { useState } from 'react';

const Counter = () => {
  console.log('Counter re-render');

  const [count, setCount] = useState(0);

  const onClick = () => {
    setCount(count + 1);
  };

  return (
    <div>
      <h1>Count: {count}</h1>
      <button onClick={onClick}>Click</button>
    </div>
  );
};

export default Counter;
```

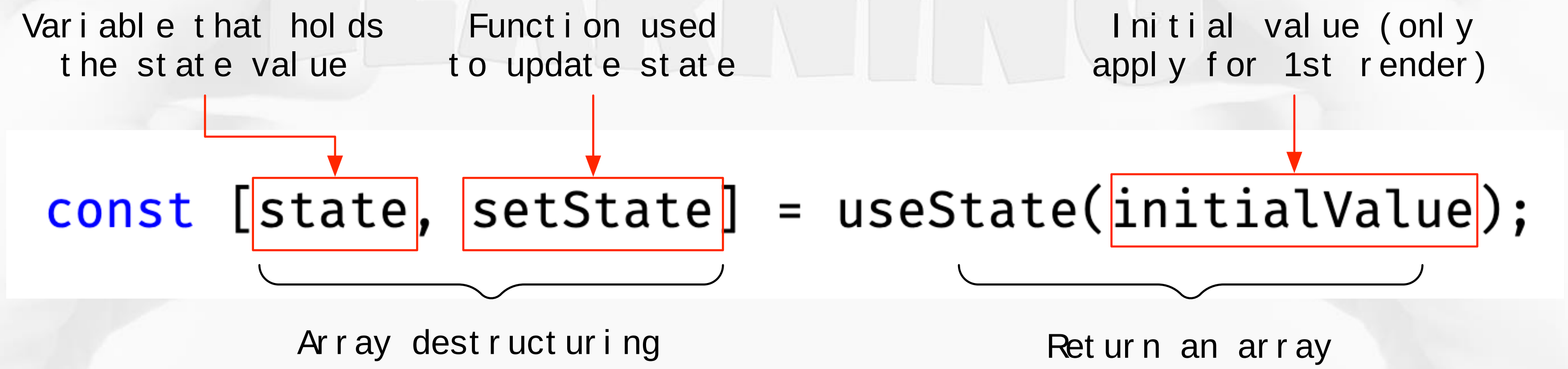
assign variable to hold State
assign function to update State
call useState
update State
access State

Demo useState() 2

Fpt Software

2. Summary

- useState allow Functional Components to use State
- Syntax for useState:



```
const [state, setState] = useState(initialValue);
```

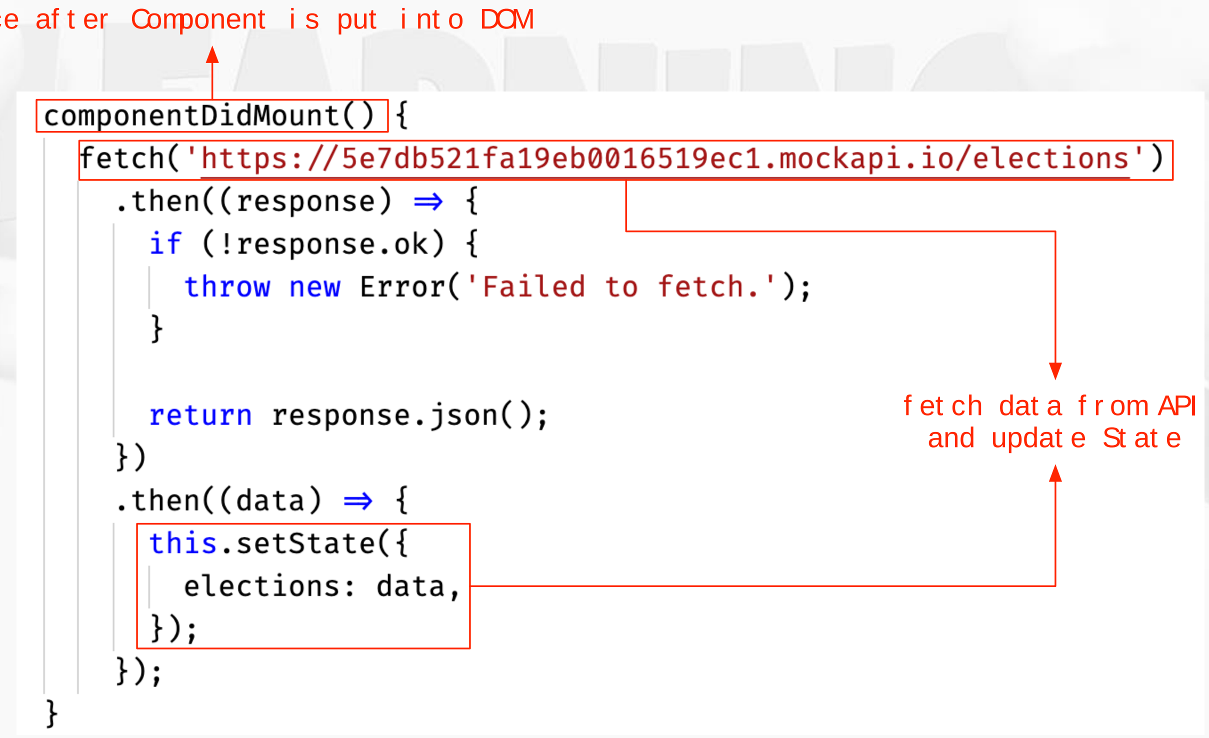
Variable that holds the state value
Function used to update state
Initial value (only apply for 1st render)
Array destructuring
Return an array

3. useEffect

Fpt Software

3. useEffect

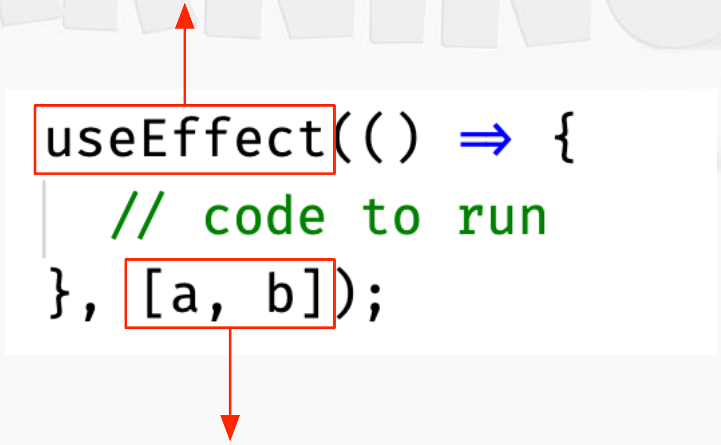
- How do we fetch data from API using Class Component ?


run once after Component is put into DOM
fetch data from API and update State

```
componentDidMount() {
  fetch('https://5e7db521fa19eb0016519ec1.mockapi.io/elections')
    .then((response) => {
      if (!response.ok) {
        throw new Error('Failed to fetch.');
      }

      return response.json();
    })
    .then((data) => {
      this.setState({
        elections: data,
      });
    });
}
```

3. useEffect

- How do we fetch data from API with Functional Component ?
  - useEffect to the rescue
- Syntax:



```
useEffect(() ⇒ {
    // code to run
}, [a, b]);
```

take a function as 1st parameter

array of dependencies. If one of the dependencies changes, the 1st parameter function will be called

Demo useEffect()

Fpt Software

3. useEffect

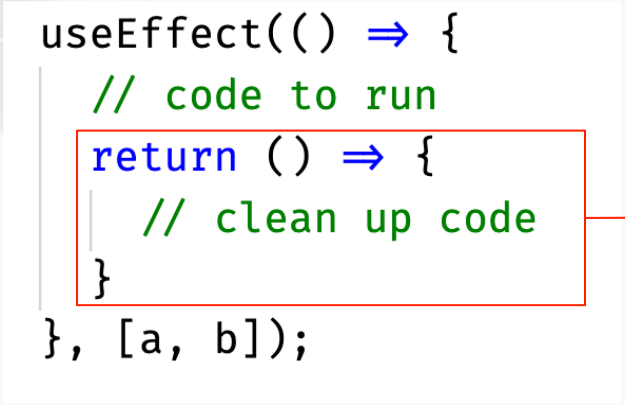
- How do we run code everytimes Components receive new props with Class Component ?
  - componentDidUpdate
- Can we do that with Functional Component ?
  - Yes, with useEffect()

Demo useEffect() 2

Fpt Software

3. useEffect

- How do we run cleanup code when Components is destroyed with Class Component ?
  - componentWillUnmout()
- Can we do that with Functional Component ?
  - Yes, with useEffect() **(again)**



```
useEffect(() => {
  // code to run
  return () => {
    // clean up code
  }
}, [a, b]);
```

If the 1st parameter of useEffect return a function. That function will be called when Component is destroyed

Demo useEffect() 3

Fpt Software

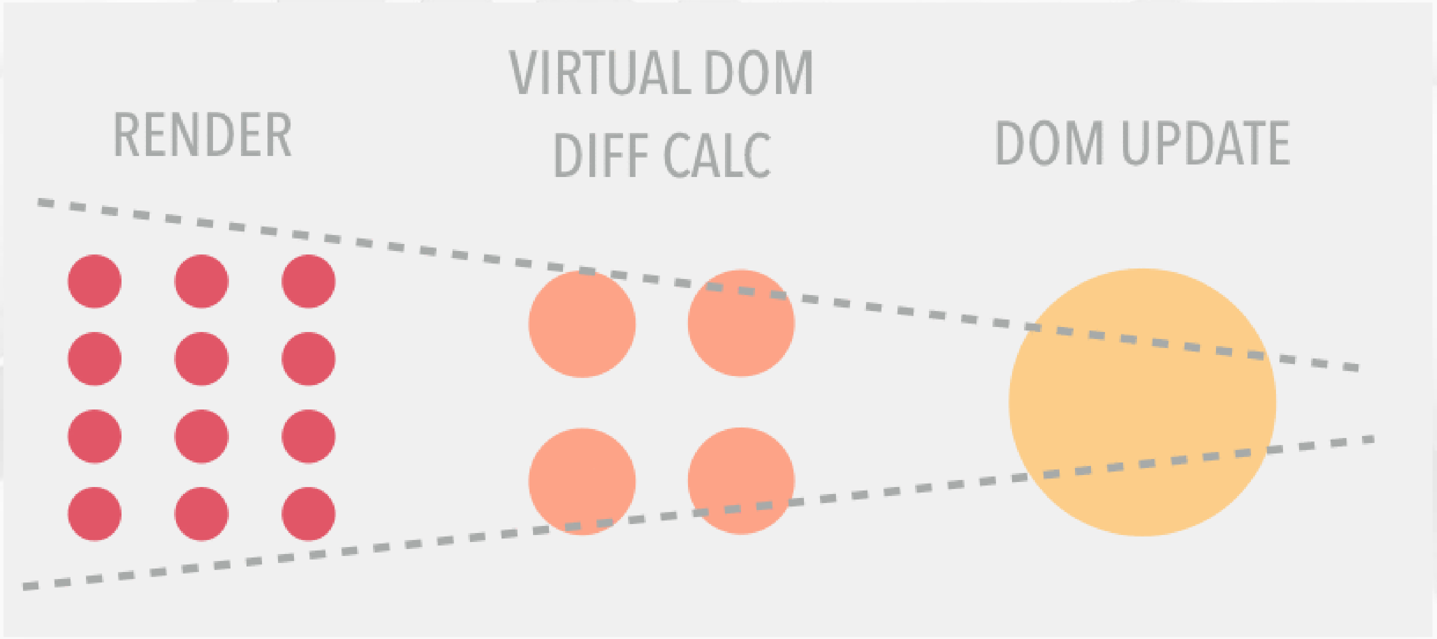
3. Summary

- useEffect allows Functional Component to do the following:
  - Run code once at Component start-up
  - Run code everytime Component receives new props
  - Run cleanup code when Component is destroyed

4. React.memo

4. React.memo

- Recall React Virtual DOM and re-render process ?

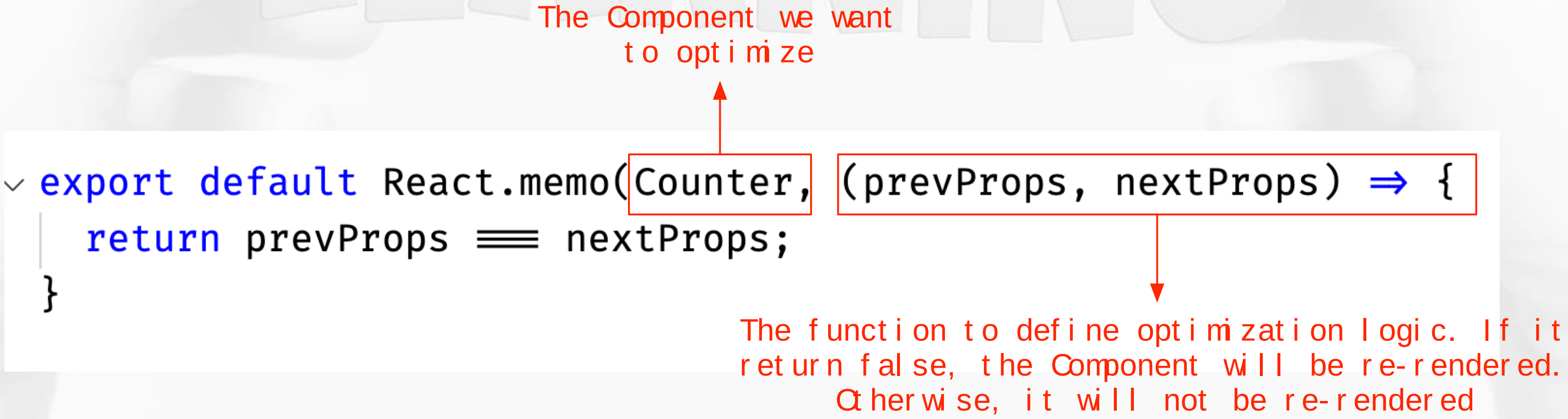


4. React.memo

- Can developer optimize the reconciliation process ?
  - Yes, with shouldComponentUpdate()
- shouldComponentUpdate() tell React whenever a Component should be re-render or not

4. React.memo

- Can Functional Component do that too ?
  - Easy with React.memo
- Syntax:



```
export default React.memo(Counter, (prevProps, nextProps) => {
  return prevProps === nextProps;
}
```

The Component we want to optimize

The function to define optimization logic. If it return false, the Component will be re-rendered. Otherwise, it will not be re-rendered

Demo React.memo()

4. Summary

- React.memo allows developers to reduce wasted re-rendering process
- React.memo takes:
  - The Component to be optimized
  - The function that defines optimization logics
- Note: we optimize the reconciliation process of React NOT the actual DOM update

Happy Coding!

Fpt Software