Nhóm 10

NHIỆM VỤ 3

LIFECYCLE METHODS & USEEFFECT

Nhóm 10

LIFECYCLE METHODS & USEEFFECT

Thành viên thực hiện:

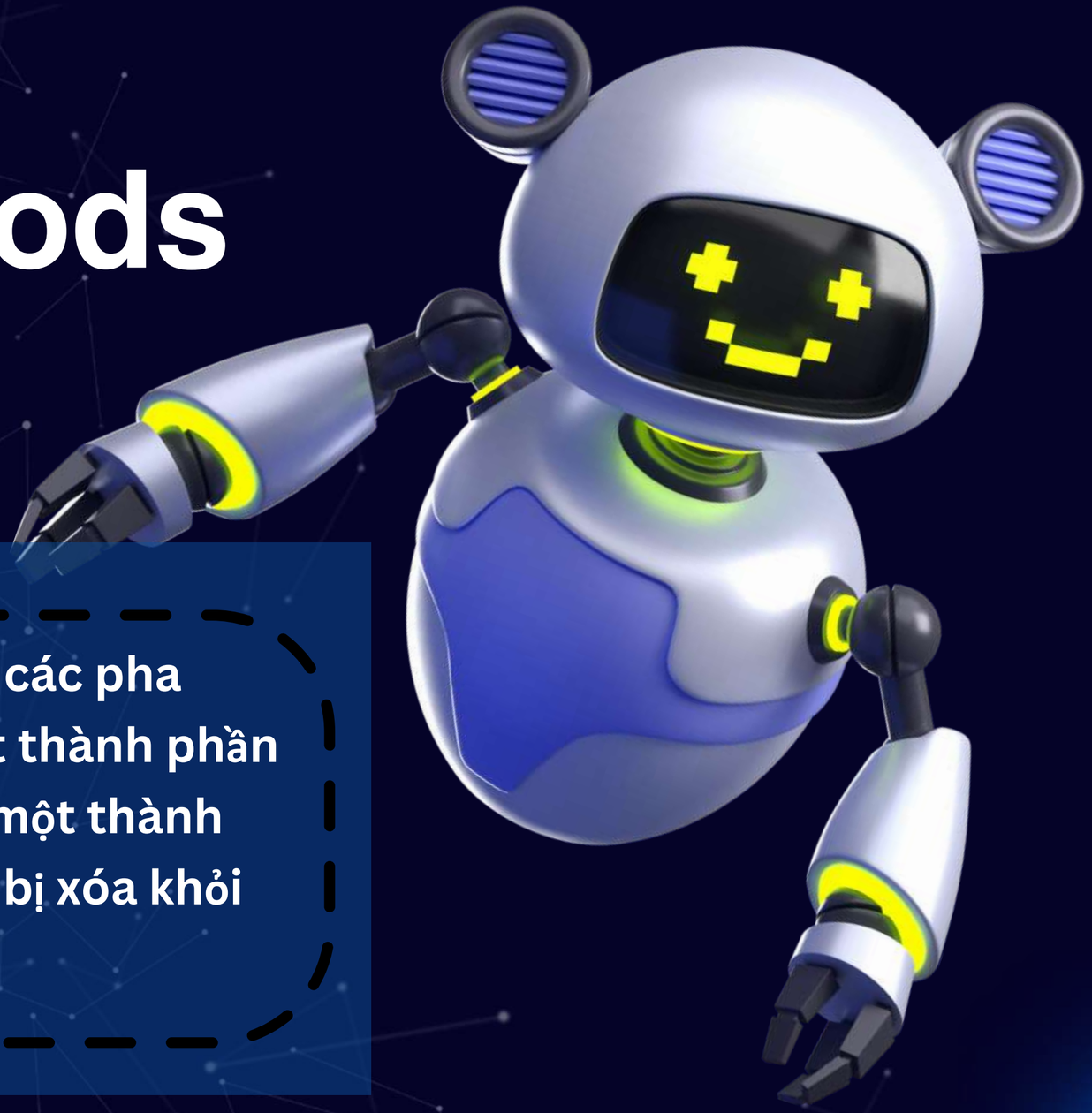
Nguyễn Văn Huỳnh

Lý Đình Sơn

Nguyễn Văn Trường

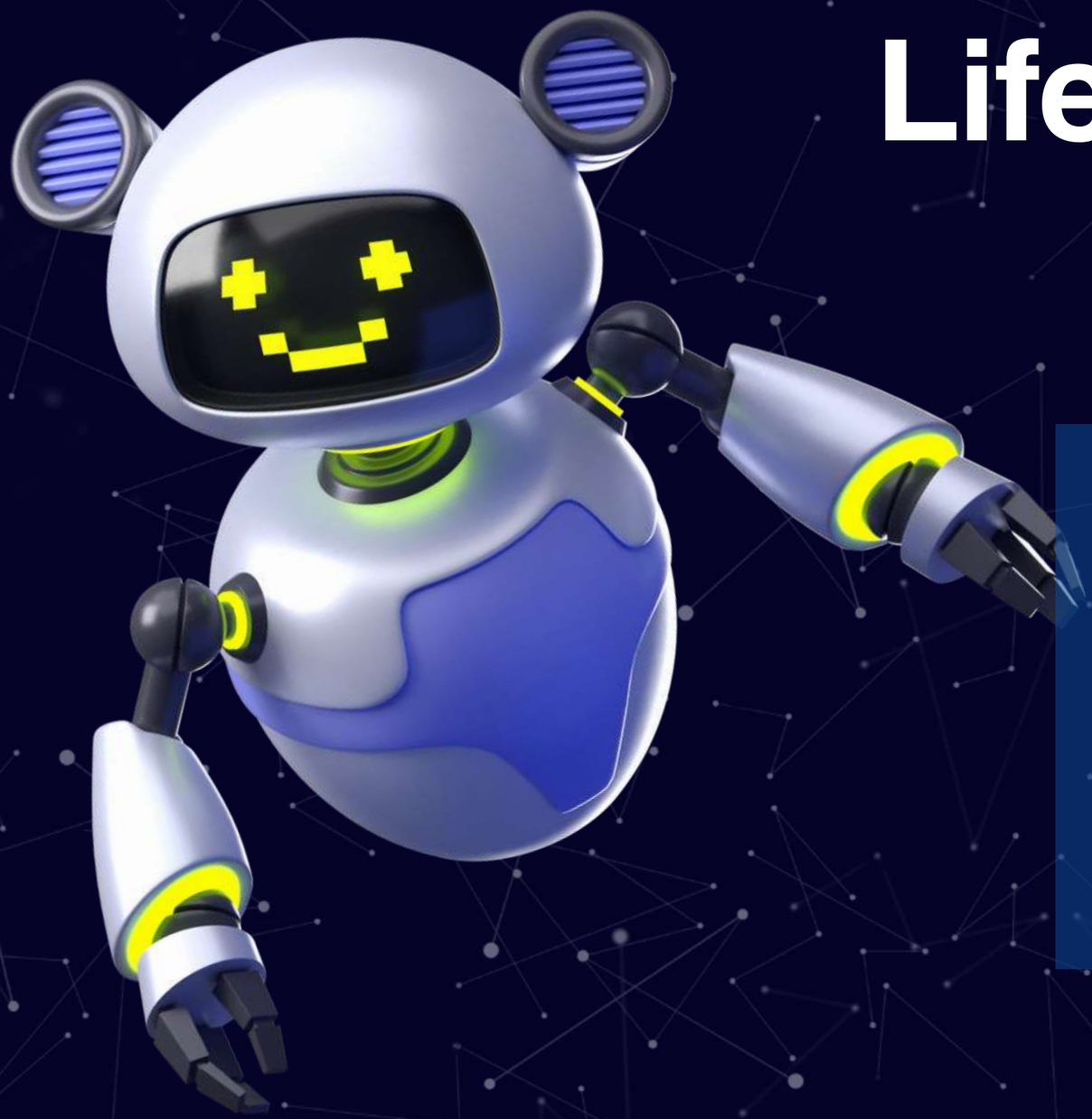
Gv hướng dẫn: ThS .Tạ Chí Hiếu

LifeCycle Methods



LifeCycle Mothods là gì ?

Lifecycle trong React là chuỗi các pha trong quá trình tồn tại của một thành phần React. Các pha này xảy ra khi một thành phần được tạo, cập nhật, hoặc bị xóa khỏi cây DOM.



LifeCycle Methods

Mục đích

- Quản lý trạng thái component
- Tối ưu hóa hiệu suất
- Xử lý side effects
- Dọn dẹp tài nguyên
- Đồng bộ hóa với external data

Các pha trong LifeCycle của Component

Mounting

constructor

render

React updates DOM and refs

componentDidMount

Updating

New props

setState()

forceUpdate()

render

React updates DOM and refs

componentDidUpdate

Unmounting

componentWillUnmount

Các pha trong LifeCycle của Component

Mounting

constructor

render

React updates DOM and refs

componentDidMount

Mounting

Giao đoạn component được tạo ra và chèn vào DOM lần đầu tiên.

3 phương thức quan trọng trong giai đoạn này bao gồm:

```
// Constructor là nơi khởi tạo state, props
constructor(props) {
  super(props); // gọi hàm constructor của lớp cha (React.Component)
  this.state = {
    count: 0,
  };
  console.log("constructor: Khởi tạo component");
}
```

```
render() {
  console.log("render: Hiển thị giao diện");
  return (
    <div>
      <h1>Count: {this.state.count}</h1>
      <button onClick={() => this.setState({ count: this.state.count + 1 })}>
        Tăng
      </button>
    </div>
  );
}
```

```
// Giai đoạn MOUNTING: gọi sau constructor và trước khi hiển thị lên DOM
componentDidMount() {
  console.log("componentDidMount: Component đã gắn vào DOM");
}
```

Mounting

Giao đoạn component được tạo ra và chèn vào DOM lần đầu tiên.

3 phương thức quan trọng trong giai đoạn này bao gồm:

```
// Constructor là nơi khởi tạo state, props
constructor(props) {
  super(props); // gọi hàm constructor của lớp cha (React.Component)
  this.state = {
    count: 0,
  };
  console.log("constructor: Khởi tạo component");
}
```

Mounting

Giao đoạn component được tạo ra và chèn vào DOM lần đầu tiên.

3 phương thức quan trọng trong giai đoạn này bao gồm:

```
// Render luôn được gọi khi component mount hoặc update
render() {
  console.log("render: Hiển thị giao diện");
  return (
    <div>
      <h1>Count: {this.state.count}</h1>
      <button onClick={() => this.setState({ count: this.state.count + 1 })}>
        Tăng
      </button>
    </div>
  );
}
```

Mounting

Giao đoạn component được tạo ra và chèn vào DOM lần đầu tiên.

3 phương thức quan trọng trong giai đoạn này bao gồm:

```
// Giai đoạn MOUNTING: gọi sau constructor và trước khi hiển thị lên DOM
componentDidMount() {
  console.log("componentDidMount: Component đã gắn vào DOM");
}
```

Các pha trong LifeCycle của Component

Mounting

constructor

render

React updates DOM and refs

componentDidMount

Updating

New props

setState()

forceUpdate()

render

componentDidUpdate

Unmounting

componentWillUnmount

Các pha trong LifeCycle của Component

Updating

New props

setState()

forceUpdate()

render

React updates DOM and refs

componentDidUpdate

Updating

Giai đoạn component được re-render do props hoặc state thay đổi.

3 phương thức quan trọng trong giai đoạn này bao gồm:

```
// Giai đoạn UPDATING: gọi mỗi khi state hoặc props thay đổi
shouldComponentUpdate(nextProps, nextState) {
  console.log("shouldComponentUpdate: Có nên re-render không?");
  return true; // nếu false thì component sẽ không cập nhật
}
```

```
// Render luôn được gọi khi component mount hoặc update
render() {
  console.log("render: Hiển thị giao diện");
  return (
    <div>
      <h1>Count: {this.state.count}</h1>
      <button onClick={() => this.setState({ count: this.state.count + 1 })}>
        Tăng
      </button>
    </div>
  );
}
```

```
componentDidUpdate(prevProps, prevState, snapshot) {
  console.log("componentDidUpdate: Component đã cập nhật");
}
```

Updating

Giai đoạn component được re-render do props hoặc state thay đổi.

3 phương thức quan trọng trong giai đoạn này bao gồm:

```
// Giai đoạn UPDATING: gọi mỗi khi state hoặc props thay đổi
shouldComponentUpdate(nextProps, nextState) {
  console.log("shouldComponentUpdate: Có nên re-render không?");
  return true; // nếu false thì component sẽ không cập nhật
}
```

Updating

Giai đoạn component được re-render do props hoặc state thay đổi.

3 phương thức quan trọng trong giai đoạn này bao gồm:

```
// Render luôn được gọi khi component mount hoặc update
render() {
  console.log("render: Hiển thị giao diện");
  return (
    <div>
      <h1>Count: {this.state.count}</h1>
      <button onClick={() => this.setState({ count: this.state.count + 1 })}>
        Tăng
      </button>
    </div>
  );
}
```

Updating

Giai đoạn component được re-render do props hoặc state thay đổi.

3 phương thức quan trọng trong giai đoạn này bao gồm:

```
componentDidUpdate(prevProps, prevState, snapshot) {
  console.log("componentDidUpdate: Component đã cập nhật");
}
```

Các pha trong LifeCycle của Component

Mounting

constructor

render

React updates DOM and refs

componentDidMount

Updating

New props

setState()

forceUpdate()

render

React updates DOM and refs

componentDidUpdate

Unmounting

componentWillUnmount

Các pha trong LifeCycle của Component

Unmounting

componentWillUnmount

Unmounting

Giai đoạn component được loại bỏ khỏi DOM.

Phương thức quan trọng trong giai đoạn này:
componentWillUnmount

```
// Giai đoạn UNMOUNTING: trước khi component bị xóa khỏi DOM
componentWillUnmount() {
  console.log("componentWillUnmount: Component sẽ bị gỡ");
}
```

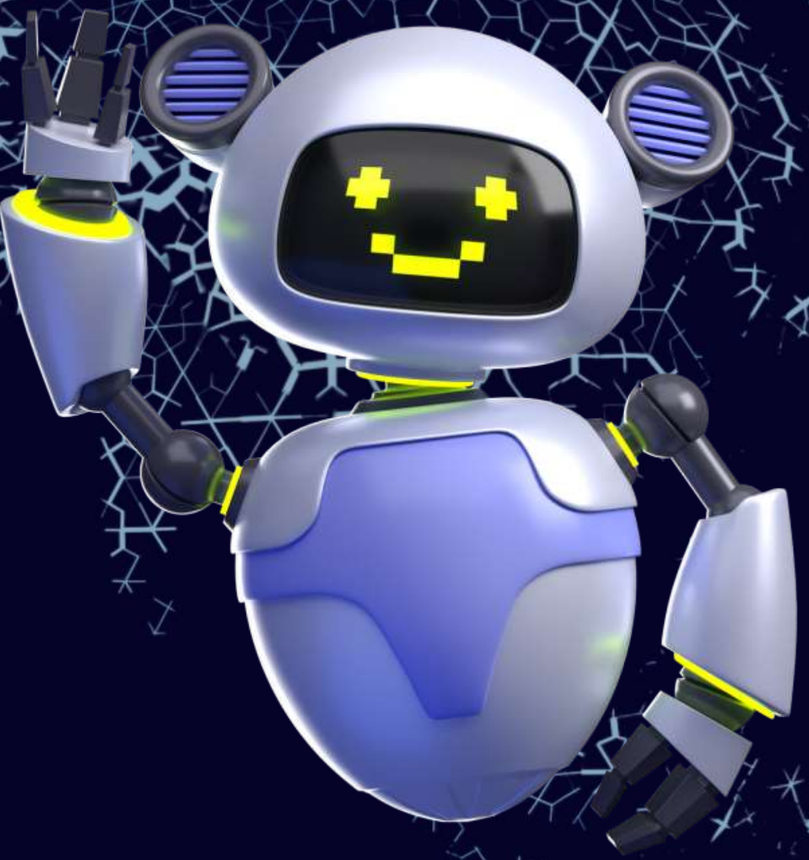
Ưu điểm của LifeCycle Methods

- Kiểm soát chi tiết vòng đời
- Tách biệt rõ ràng theo giai đoạn
- Dễ dàng tối ưu hóa hiệu suất
- Dễ dàng làm việc với DOM

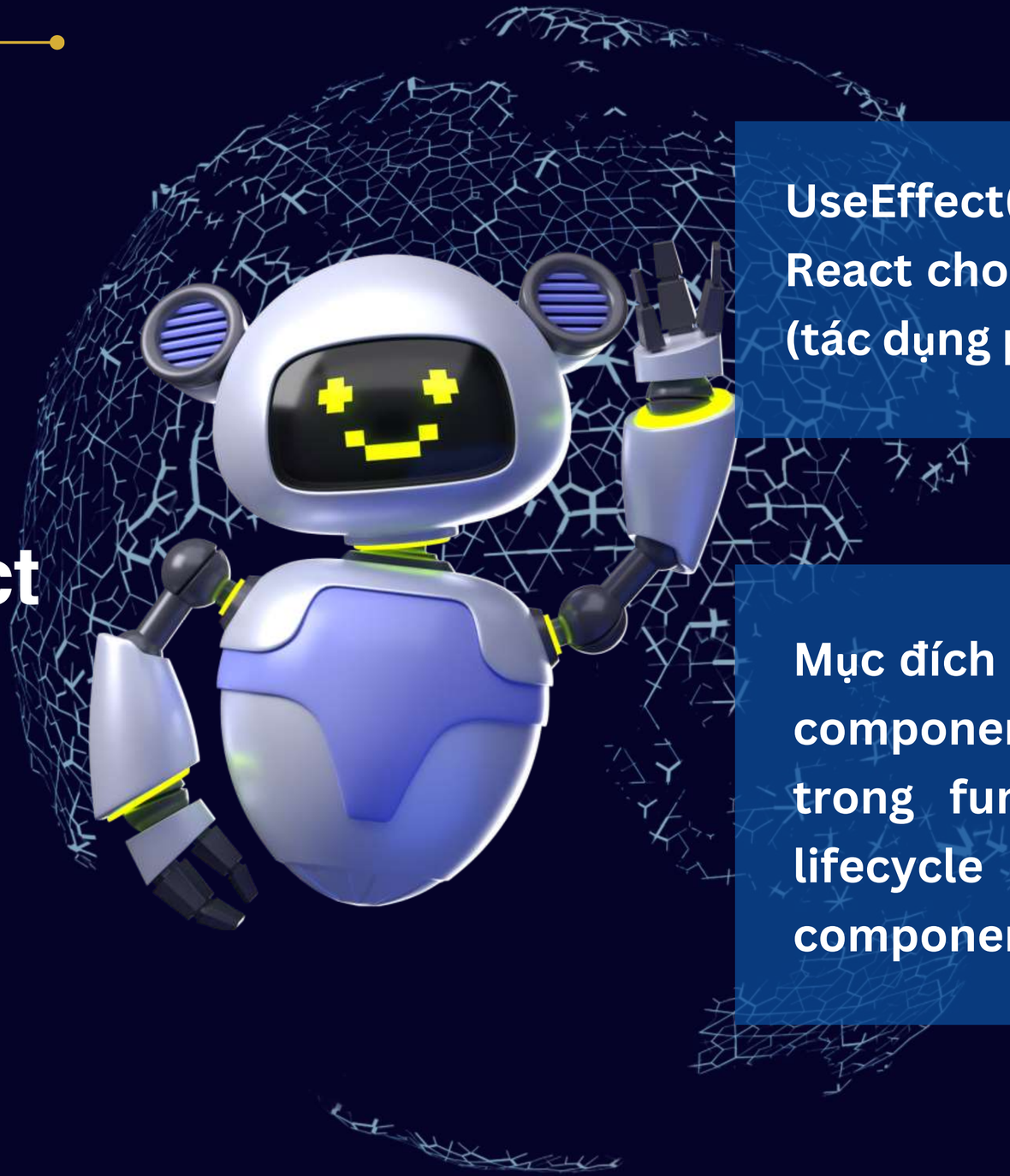
Nhược điểm của LifeCycle Methods

- Code bị chia nhỏ ra thành nhiều phương thức
- Tăng độ phức tạp với state
- Khó tái sử dụng logic giữa các component
- Khó cho người mới học

UseEffect



UseEffect



UseEffect() là một hook quan trọng trong React cho phép bạn thực hiện các side effect (tác dụng phụ) trong các component function

Mục đích : để quản lý vòng đời của của một component và nó phục vụ chúng ta sử dụng trong function component thay vì các lifecycle như trước đây trong class component..

Cú pháp

```
useEffect(effectFunction, dependencyArray);
```

Tham số thứ nhất

effectFunction

Tham số thứ hai

dependencyArray

Cú pháp

```
useEffect(effectFunction, dependencyArray);
```

effectFunction

- Một hàm callback chứa code thực hiện side effect.
- Bắt buộc phải có.

Cú pháp

```
useEffect(effectFunction, dependencyArray);
```

Tham số thứ nhất

effectFunction

Tham số thứ hai

dependencyArray

Cú pháp

```
useEffect(effectFunction, dependencyArray);
```

dependencyArray

1. Là một mảng tùy chọn (optional) chứa các giá trị phụ thuộc.
2. Quyết định khi nào effectFunction được thực thi lại:

- Không truyền
- Mảng rỗng []
- Có dependencies [dep1, dep2]

Các trường hợp Dependency array

01

02

03

Các trường hợp Dependency array

01 Không có dependency array (không truyền mảng)

```
useEffect(() => {
  console.log('Chạy sau mỗi lần render');
});
```

02

03

Các trường hợp Dependency array

01 Không có dependency array (không truyền mảng)

02 Dependency array rỗng ([])

```
useEffect(() => {
  console.log('Chỉ chạy 1 lần sau mount');
}, []);
```

03

Các trường hợp Dependency array

01 Không có dependency array (không truyền mảng)

02 Dependency array rỗng ([])

03 Dependency array có giá trị ([dep1, dep2])

```
useEffect(() => {
  // 1. Chạy sau lần render đầu tiên (mount)
  // 2. Chạy lại khi bất kỳ dependency nào thay đổi
}, [prop, state]); // Dependency array
```

Các trường hợp Dependency array

01 Không có dependency array (không truyền mảng)

02 Dependency array rỗng ([])

03 Dependency array có giá trị ([dep1, dep2])

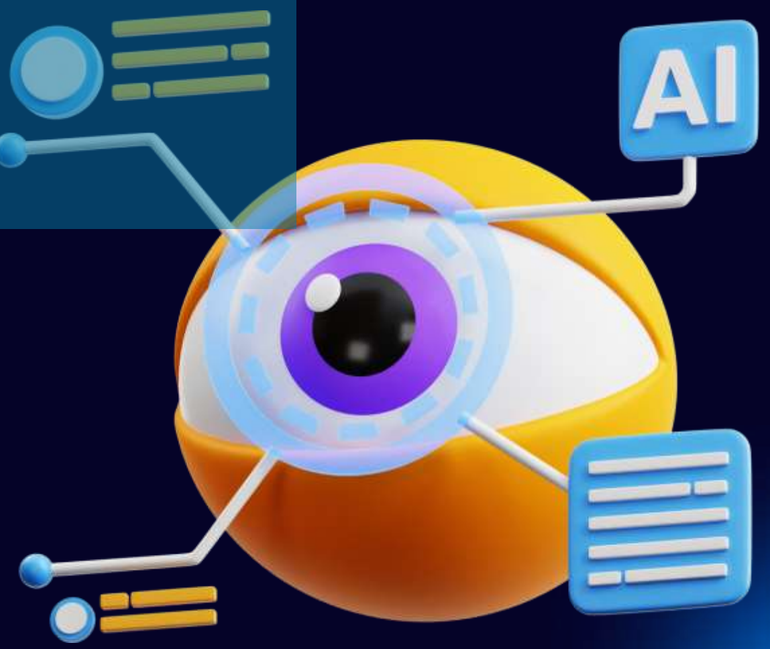
Cleanup function (Hàm dọn dẹp)

Bản chất : Cleanup function là cơ chế **"dọn dẹp trước khi rời đi"** trong React, hoạt động như một bảo hiểm, để :

- Ngăn memory leaks
- Tránh lỗi thực thi trên component đã unmount
- Hủy bỏ các tác vụ không cần thiết
- Duy trì tính nhất quán của ứng dụng

Các trường hợp sử dụng phổ biến của cleanup function

- Hủy subscriptions (WebSocket, API, RxJS, v.v.)
- Đóng modals hoặc popups
- Gỡ bỏ event listeners (e.g. window.addEventListener)
- Clear timeout hoặc interval (clearTimeout, clearInterval)



Cleanup function được gọi trong các trường hợp sau

3.1 Trước khi component unmount

3.2 Trước khi effect chạy lại (khi có dependencies thay đổi)

3.3 Trong Strict Mode (development)

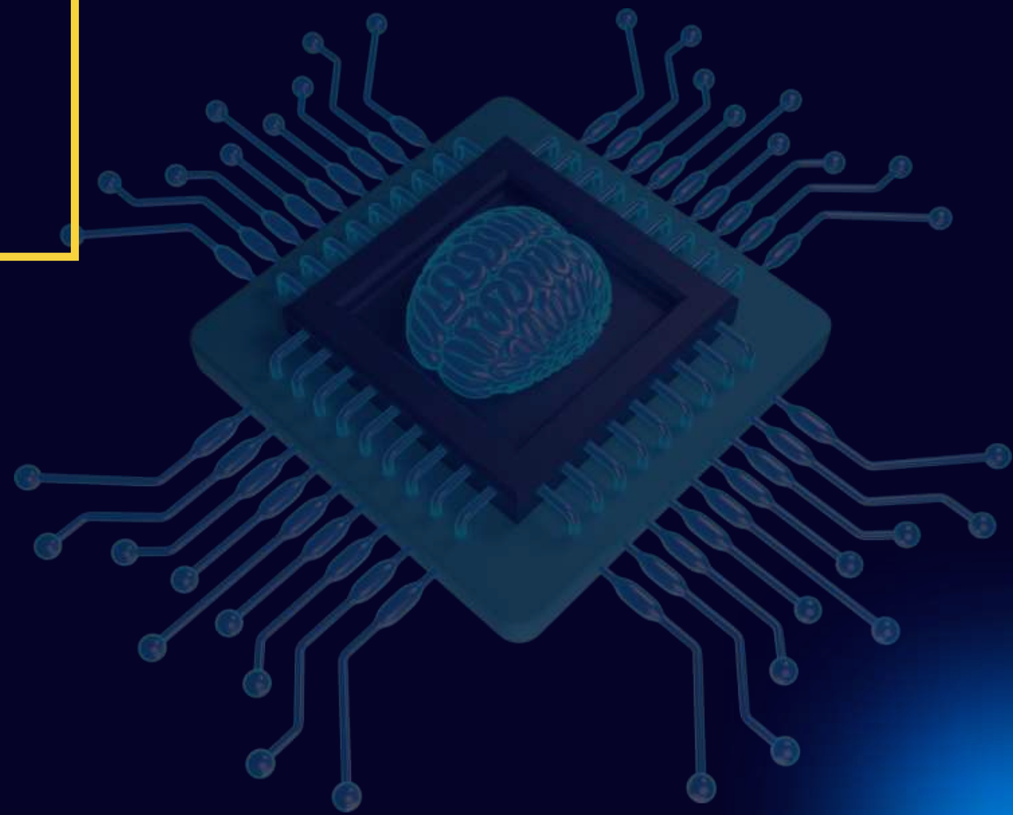
Cú Pháp

```
import React, { useEffect } from "React";

useEffect(() => {
  // Your effect
  return () => {
    // Cleanup
  };
}, []);
```

So sánh LifeCycle Methods với useEffect

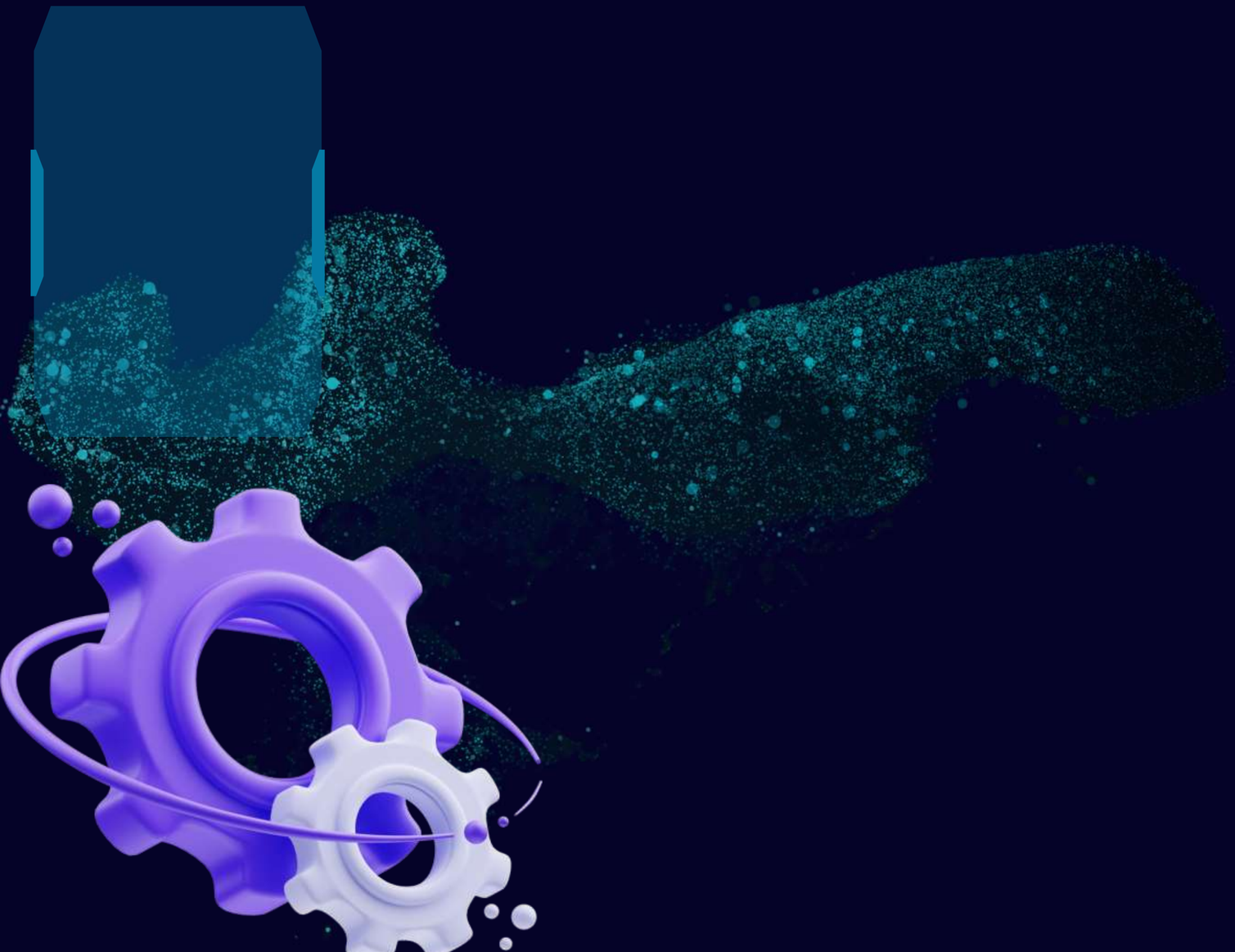
So sánh tổng quan

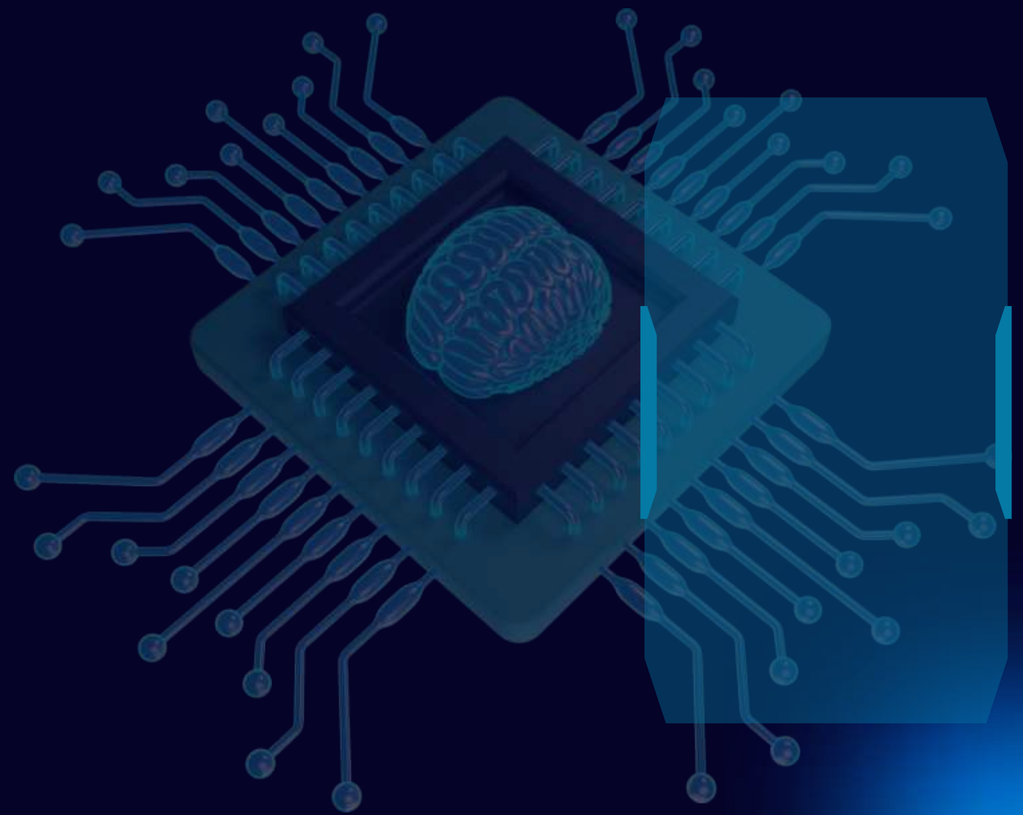


So sánh LifeCycle Methods với useEffect

Tiêu chí	Lifecycle Methods	useEffect
Cách dùng	Dùng các phương thức vòng đời như: componentDidMount, componentDidUpdate, componentWillUnmount,... ()	Dùng hook useEffect() để xử lý mọi giai đoạn của component
Loại component áp dụng	Chỉ dùng với class Component	Dùng với function component
Tổ chức code	Phân tách logic theo từng method	Gom logic liên quan vào cùng một hoặc nhiều useEffect
Dọn dẹp side effect	Trong componentWillUnmount()	Trong hàm return của useEffect()
Tái sử dụng logic	Khó tái sử dụng (nhiều code trùng lặp)	Dễ tái sử dụng với custom Hooks

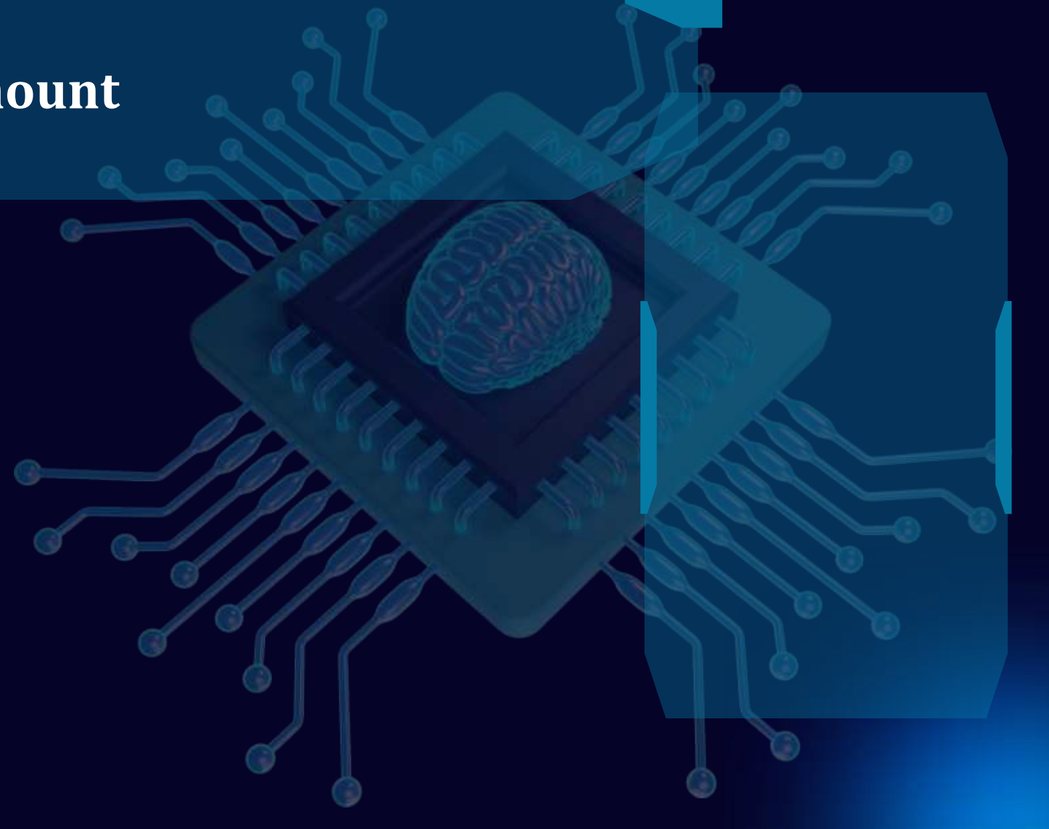
Tổng kết





Tổng kết

1. Thay thế các phương thức vòng đời:
2. useEffect kết hợp chức năng của nhiều phương thức vòng đời vào một Hook duy nhất
   - ✓ - Chạy sau mỗi lần render → giống componentDidUpdate
   - ✓ - Mảng phụ thuộc ([]) → gióng componentDidMount
   - ✓ - Hàm return bên trong → giống componentWillUnmount



Tổng kết

Quản lý side effects:

- Tránh gây gián đoạn render
- Dọn dẹp hiệu quả khi unmount hoặc re-render

Thank You!