

REACT PARA PRINCIPIANTES

TEMA 4: PROPS & STATE

Material obtenido de Open Webinars

TEMAS DEL CURSO

REACT PARA

PRINCIPIANTES



1. **FUNDAMENTOS:** qué saber antes de iniciarse con React
2. **SETUP:** crear un proyecto React desde cero.
3. **RENDERIZADO:** cómo aprovechar las capacidades de renderizado de React
4. **PROPS & STATE:** comunicación de componentes

ÍNDICE DEL TEMA 3

INTRODUCCIÓN A REACT. 4 - PROPS & STATE



Props

Propiedades: Manera de recibir info de fuera (de su padre)



State

Datos que maneja cada componente



Lifecycle

Ciclo de vida desde que nace hasta que se destruye

RECORDATORIO COMPONENTES

- Se pueden equiparar a funciones JavaScript
- Es como una función simple que devuelve jsx

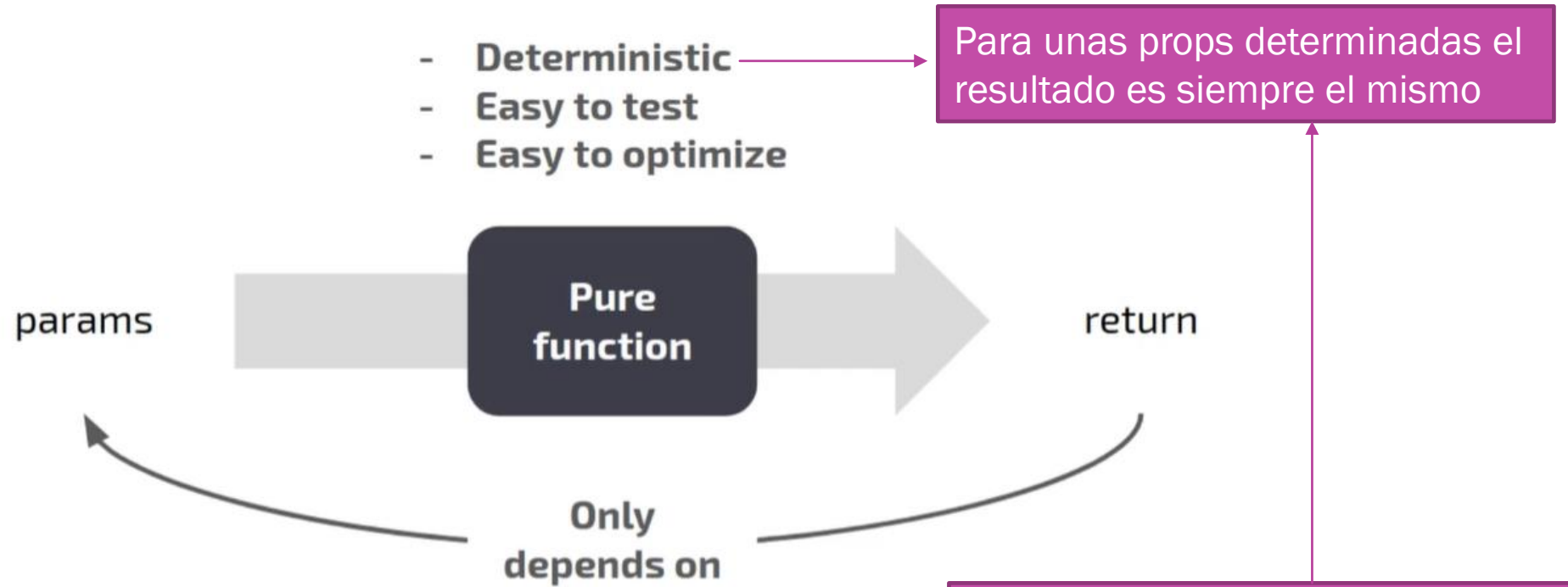


Parámetros de React se llaman props



Ejemplo de componente muy sencillo llamado Header

```
const Header = () => <div> My Wishlist </div>;
```

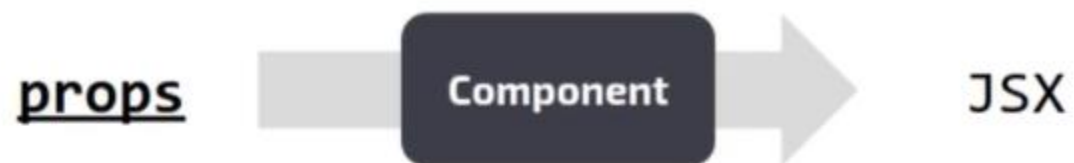


Por tanto, intentaremos que nuestros componentes sean funciones puras

FUNCIONES PURAS

SON AQUELLAS CUYO VALOR DE RESPUESTA DEPENDE ÚNICAMENTE DE LOS PARÁMETRO DE ENTRADA

FLUJO DE COMPONENTES



```
const Header = props => <h1> {props.label} </h1>;
```

Declaration

```
const Header = props =>  
  <h1> {props.label} </h1>;
```

Usage

```
<Header label="My wishlist"/>
```

EJEMPLO

- Recordemos que la comunicación es siempre de padres a hijos

Functional component

```
const Header = props =>  
  <h1> {props.label} </h1>;
```

Seguiremos la versión funcional

~

Class component

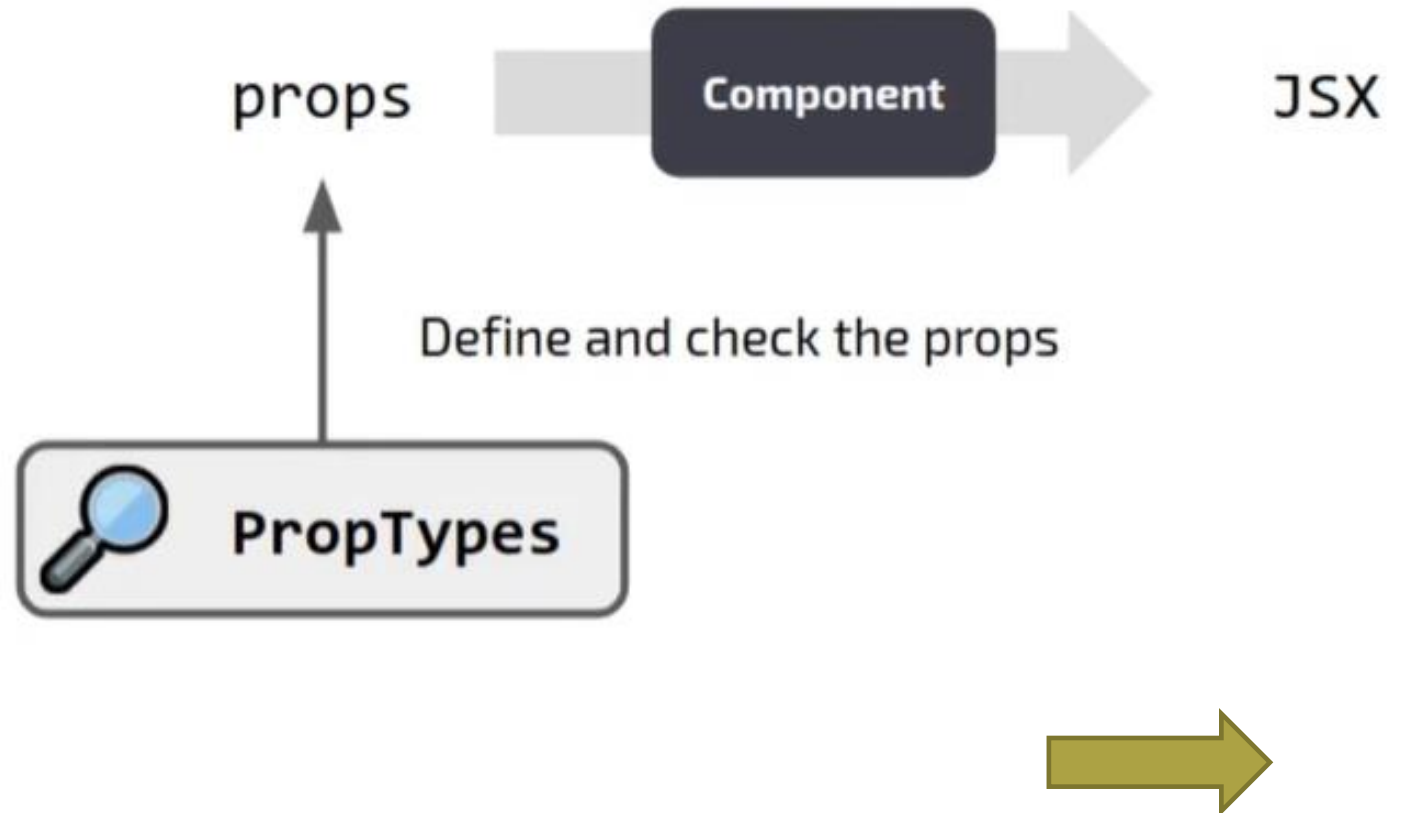
```
class Header extends Component {  
  render() {  
    const { label } = this.props;  
    return <h1>{label}</h1>;  
  }  
}
```

COMPARACIÓN CON VERSIÓN DE CLASES

AQUÍ YA SE HACE USO DE THIS

PROPTYPES

- Permite especificar cómo van a ser nuestras propiedades.
- Tenemos una librería PropTypes que nos lo permite



EJEMPLO DE PROPTYPES

DEFINIMOS QUE ES DE TIPO STRING Y REQUERIDA

```
import PropTypes from 'prop-types';

const Header = ({label}) => <h1>{label}</h1>

Header.propTypes = {
  label: PropTypes.string.isRequired
}
```



`<Header label={3} />`

`<h1>3</h1>`

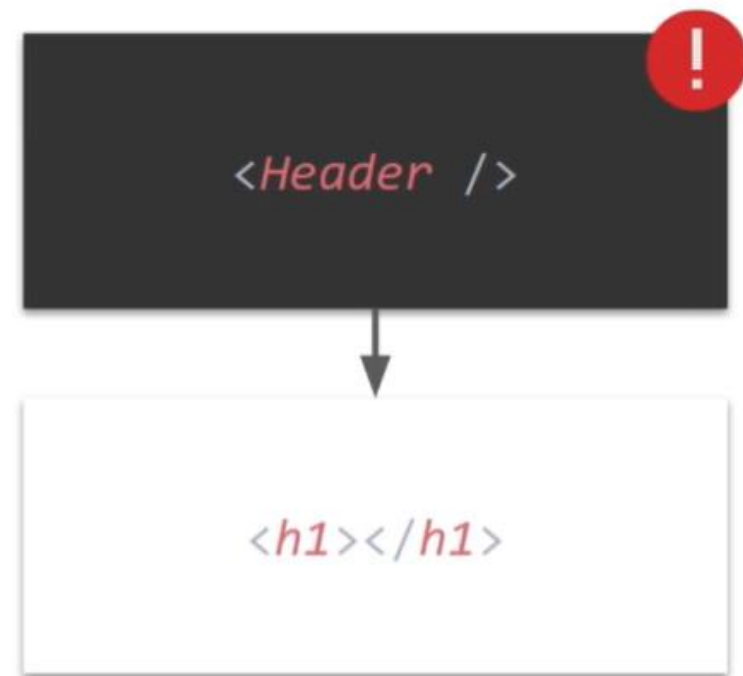
EJEMPLO DE PROPTYPES

DEFINIMOS QUE ES DE TIPO STRING Y REQUERIDA

```
import PropTypes from 'prop-types';

const Header = ({label}) => <h1>{label}</h1>

Header.propTypes = {
  label: PropTypes.string.isRequired
}
```




EJEMPLO DE PROPTYPES

DEFINIMOS QUE ES DE TIPO STRING Y REQUERIDA

```
import PropTypes from 'prop-types';

const Header = ({label}) => <h1>{label}</h1>

Header.propTypes = {
  label: PropTypes.string.isRequired
}
```



```
<Header label="Hey" />
```


Diagram showing the transformation of a React component call. A box at the top contains the code `<Header label="Hey" />` with a green checkmark icon in the top right corner. An arrow points down from this box to another box below it. The bottom box contains the rendered output `<h1>Hey</h1>`.

```
<h1>Hey</h1>
```

PROPTYPES. VALOR POR DEFECTO

```
import PropTypes from 'prop-types';
const Header = ({label}) => <h1>{label}</h1>

Header.propTypes = {
  label: PropTypes.string
}
Header.defaultProps = {
  label: 'My Wishlist'
}
```



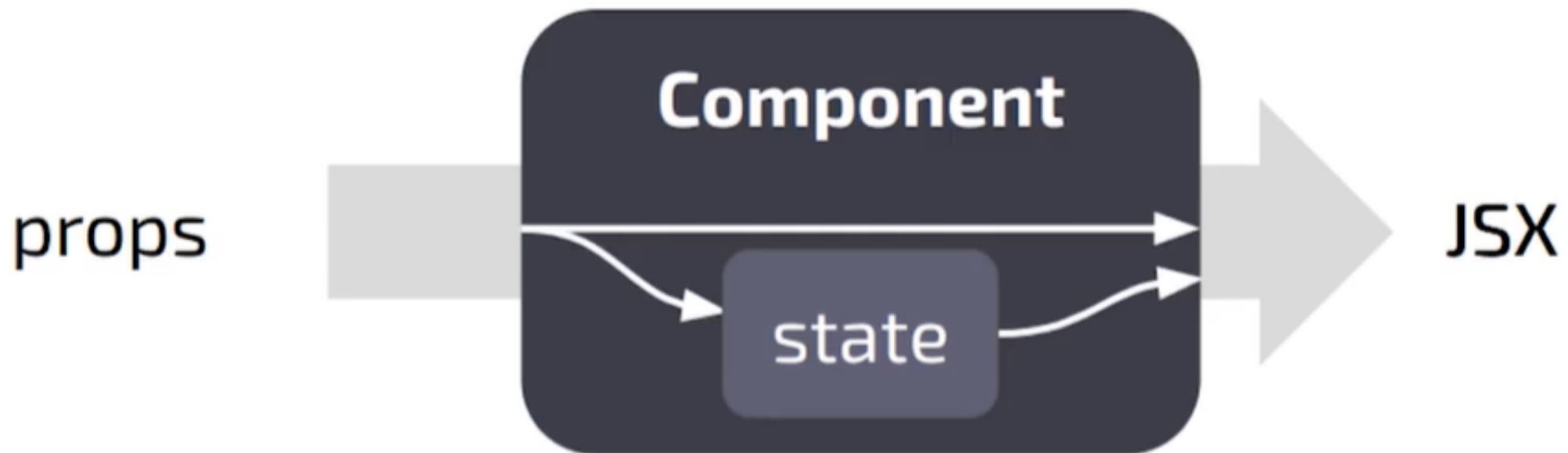
```
<Header />
```

↓

```
<h1>My Wishlist</h1>
```

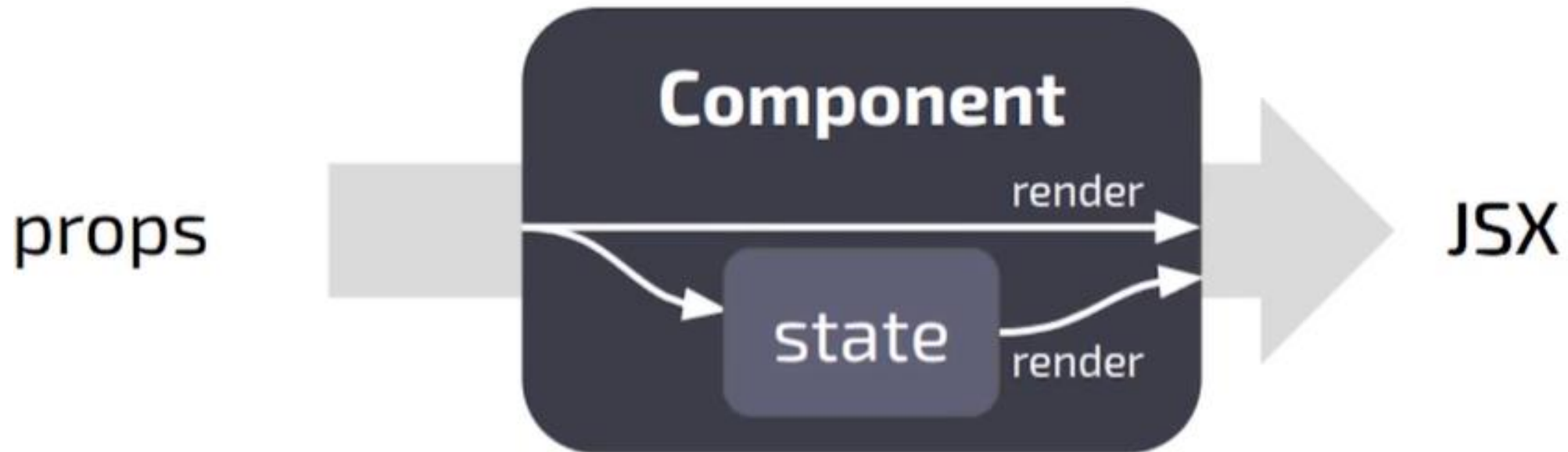
ESTADO INTERNO DE COMPONENTES

- El estado de un componente se puede gestionar de manera interna
- El resultado JSX puede estar condicionado además de por las propiedades, por el estado interno.
- El estado también puede estar condicionado por las propiedades



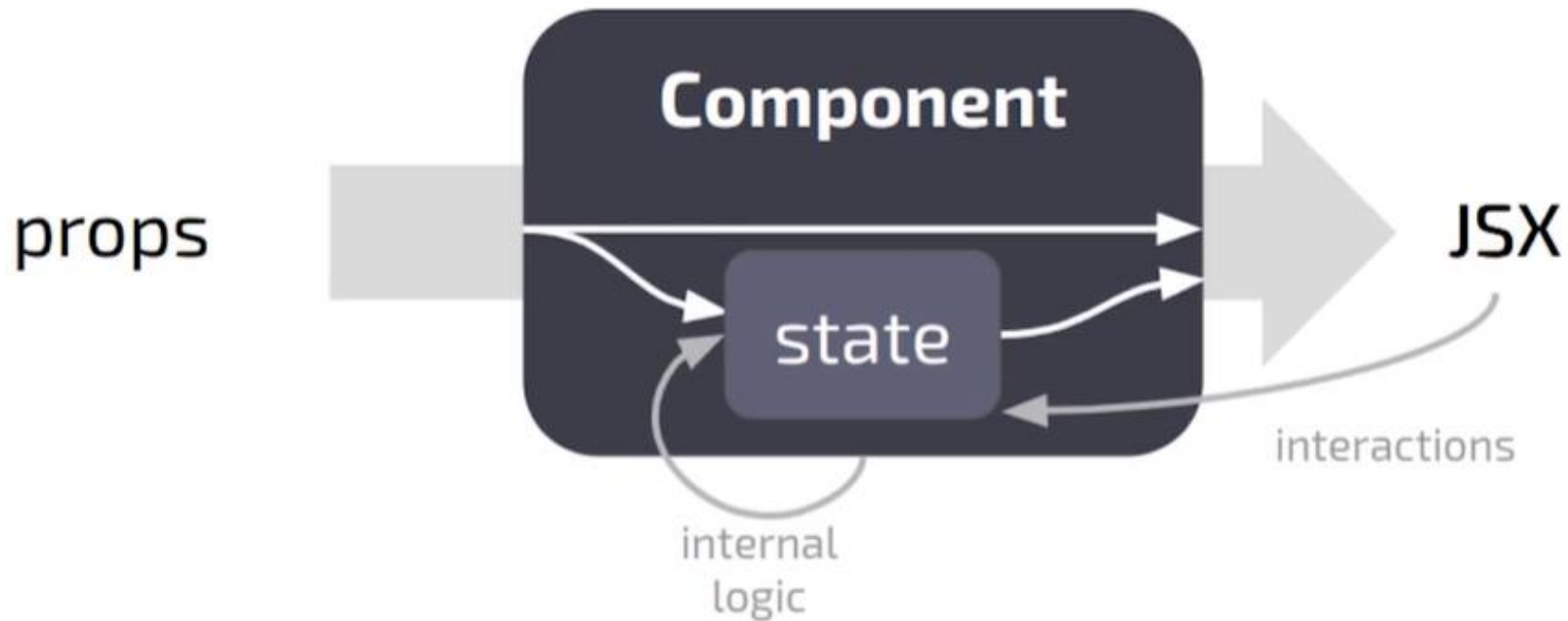
ESTADO INTERNO DE COMPONENTES

- El renderizado nuevo del JSX solo ocurre al modificar las props o estado.



ESTADO INTERNO DE COMPONENTES

- El estado se puede cambiar desde dentro:
 - Interacciones del usuario con la interfaz (eventos tipo click, change, etc)
 - Cualquier otra lógica que me permita cambiar su estado final



GESTIÓN DEL ESTADO

USESTATE

HOOKS

- El hook de useState (concepto propuesto en 2018) habilita la gestión del estado en componentes funcionales (antes solo podían en las clases)
- defaultValue: valor por defecto de la variable de estado.
- Devuelve un array con dos elementos
- setValue: función que se invoca para setear el valor de value

```
[value, setValue] = useState(defaultValue)
```

EJEMPLO DE HOOK

- Punto de partida:
item de una lista

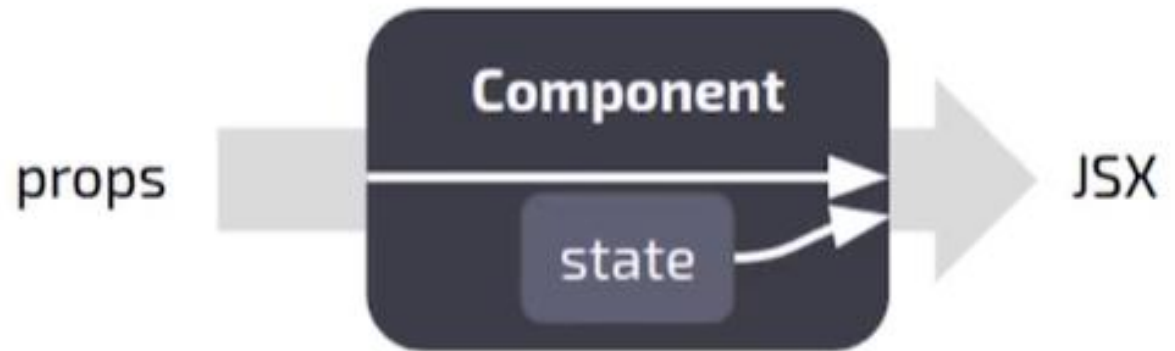
```
const TodoItem = ({ Label }) => {  
  return (  
    <p>  
      {label}  
    </p>  
  );  
};
```



EJEMPLO DE HOOK

- Creamos var de estado llamada checked.
- Por defecto va a estar como false (no chequeada) por tanto, inicialmente valdrá 'X'

```
const TodoItem = ({ label }) => {  
  const [checked] = useState(false);  
  return (  
    <p>  
      {checked ? '✓' : 'X'} {label}  
    </p>  
  );  
};
```



EJEMPLO DE HOOK

- Añadimos onClick al párrafo para que cada vez que pincha en él vaya cambiando el valor de checked.

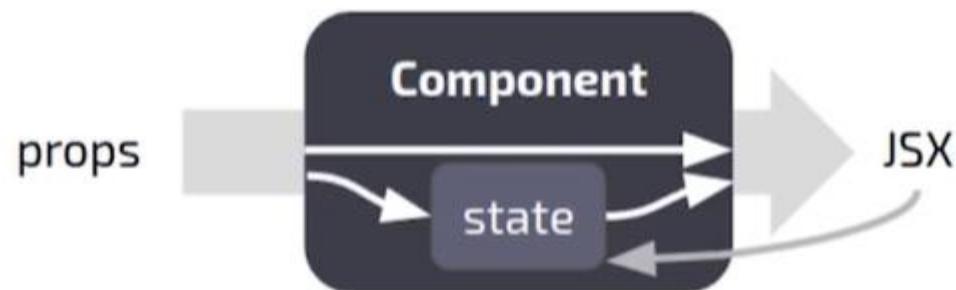
```
const TodoItem = ({ Label }) => {  
  const [checked, setChecked] = useState(false);  
  return (  
    <p onClick={() => setChecked(!checked)}>  
      {checked ? '✓' : 'X'} {label}  
    </p>  
  );  
};
```



EJEMPLO DE HOOK

- Podemos ampliar el comportamiento pasándole como prop un valor por defecto para el checked
- Me permite indicar su estado inicial (marcado o no)

```
const TodoItem = ({ label, defChk }) => {  
  const [checked, setChecked] = useState(defChk);  
  return (  
    <p onClick={() => setChecked(!checked)}>  
      {checked ? '✓' : 'X'} {label}  
    </p>  
  );  
};
```



COMPARACIÓN CON LA VERSIÓN TRADICIONAL (DE CLASE)

Functional component

```
const TodoItem = ({ label, defChk }) => {
  const [checked, setChecked] = useState(defChk);
  return (
    <p onClick={() => setChecked(!checked)}>
      {checked ? '✓' : 'X'} {label}
    </p>
  );
};
```

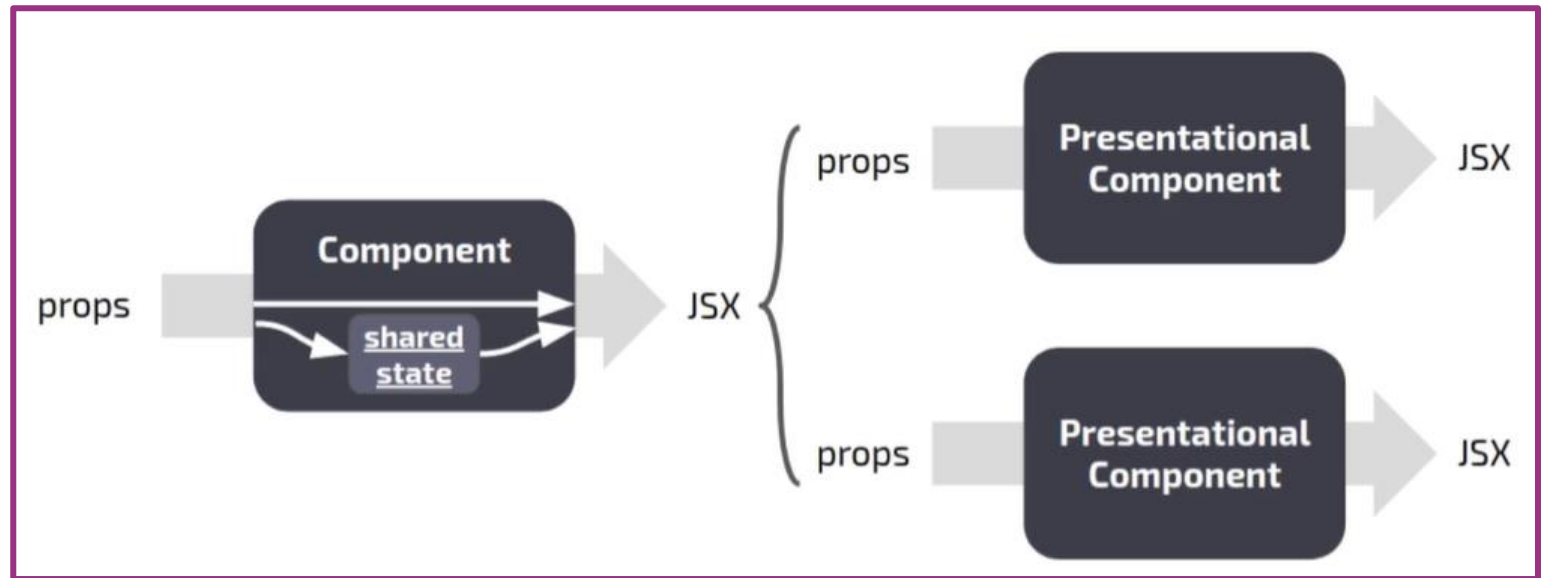
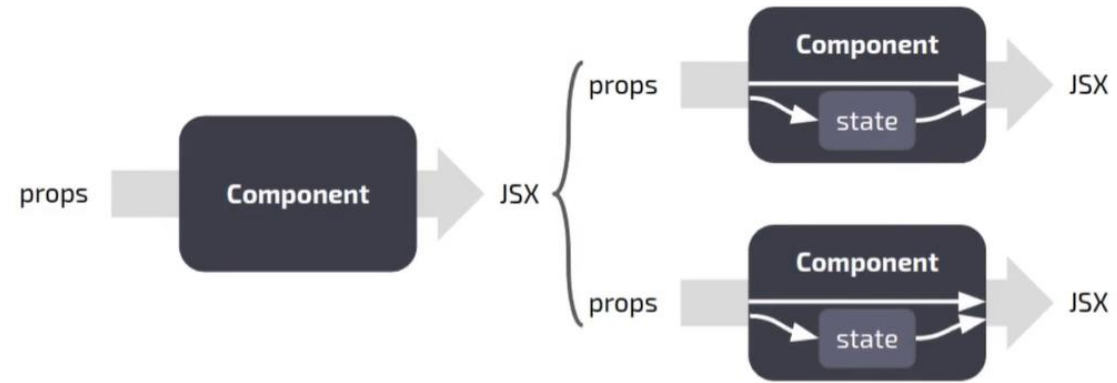
Class component

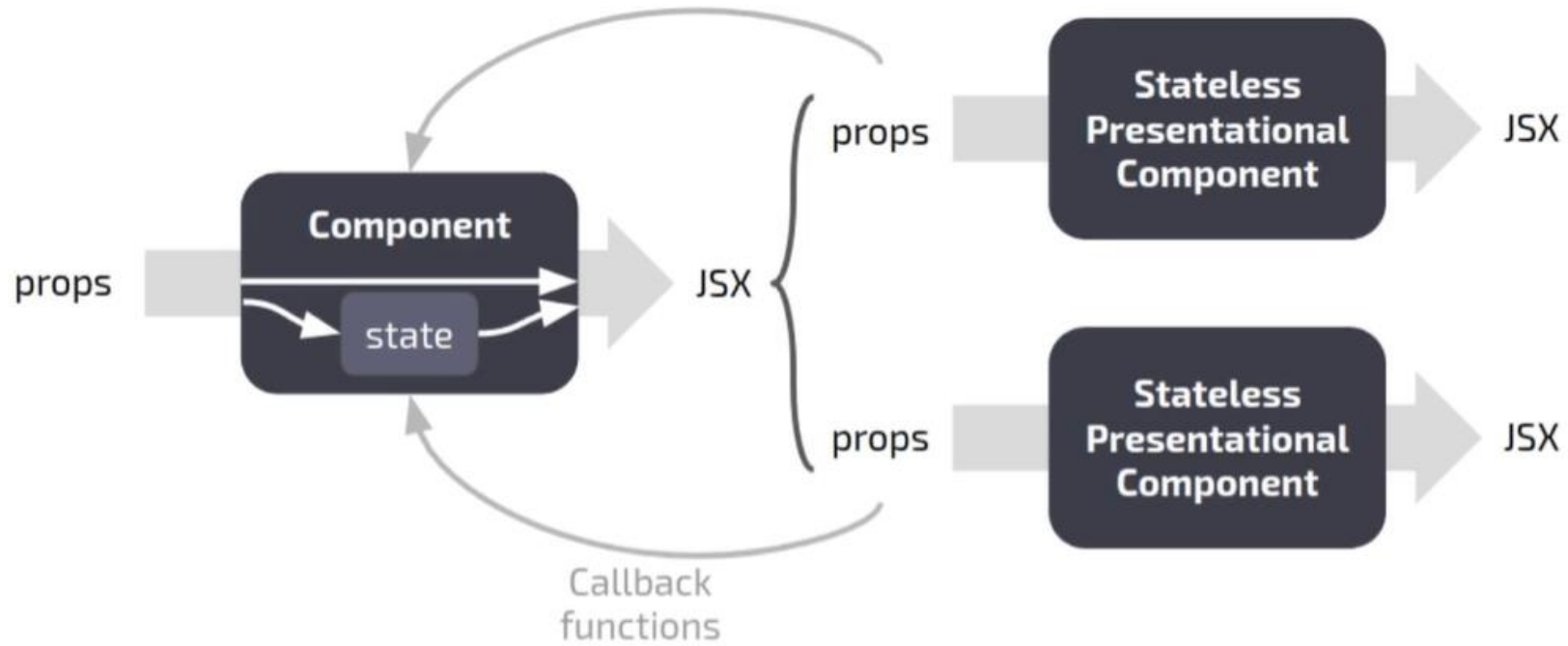
```
class TodoItem extends Component {
  state = { checked: this.props.defChk };

  render() {
    const { checked } = this.state;
    const { label } = this.props;
    return (
      <p onClick={() =>
        this.setState({ checked: !checked })
      }>
        {checked ? '✓' : 'X'} {label}
      </p>
    );
  }
}
```

DELEGACIÓN DE ESTADOS

- Así nos evitamos tener que gestionar el estado interno en cada uno de los hijos
- Además así entre hermanos no podrían compartir info o estado puesto que en React SOLO el padre puede mandar al hijo





DELEGACIÓN DE ESTADOS

SE HACE MEDIANTE CALLBACKS

EJEMPLO DE ESTADO DELEGADO

```
const Search = () => {  
  const [search, setSearch] = useState('');  
  return (  
    <div>  
      <SearchInput  
        search={search}  
        onChange={setSearch}  
      />  
      <SearchDisplay  
        search={search}  
        onClear={() => setSearch('')}  
      />  
    </div>  
  );  
};
```



```
const SearchInput = ({search, onChange}) => (  
  <input  
    value={search}  
    onChange={e => onChange(e.target.value)}  
  />  
);
```

```
const SearchDisplay = ({ search, onClear }) => (  
  <div>  
    <p>Current search: {search}</p>  
    <button onClick={onClear}>Clear</button>  
  </div>  
);
```

Lifecycle



Born



Death

Component Lifecycle



Mounted

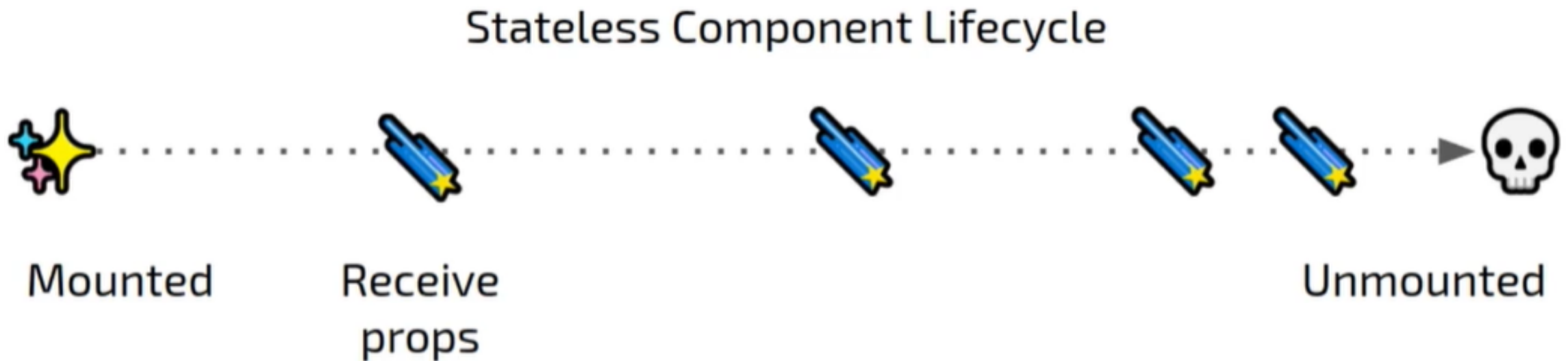


Unmounted

CICLO DE VIDA DE UN COMPONENTE

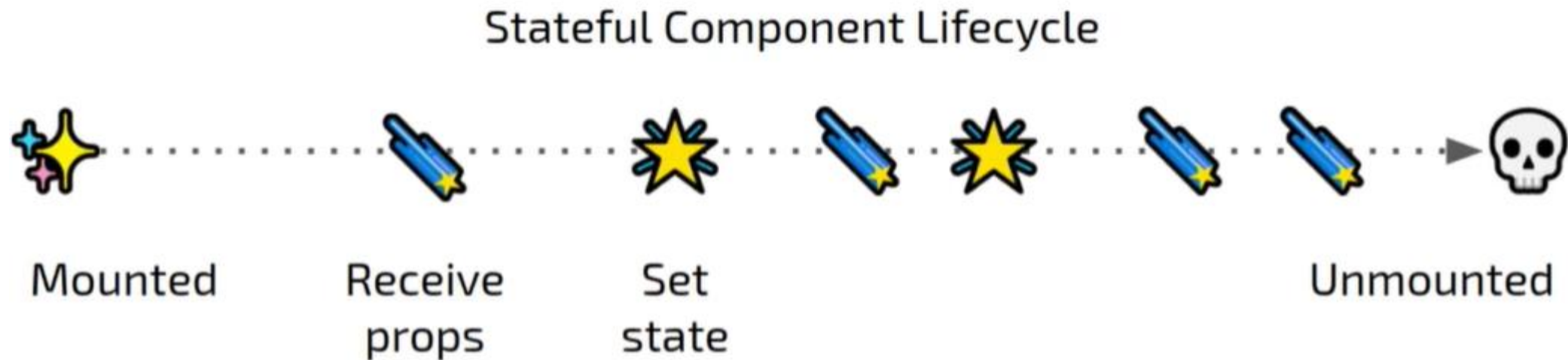
ESTADOS DEL CICLO DE VIDA

COMPONENTE SIN ESTADO: QUE RECIBA NUEVAS PROPIEDADES

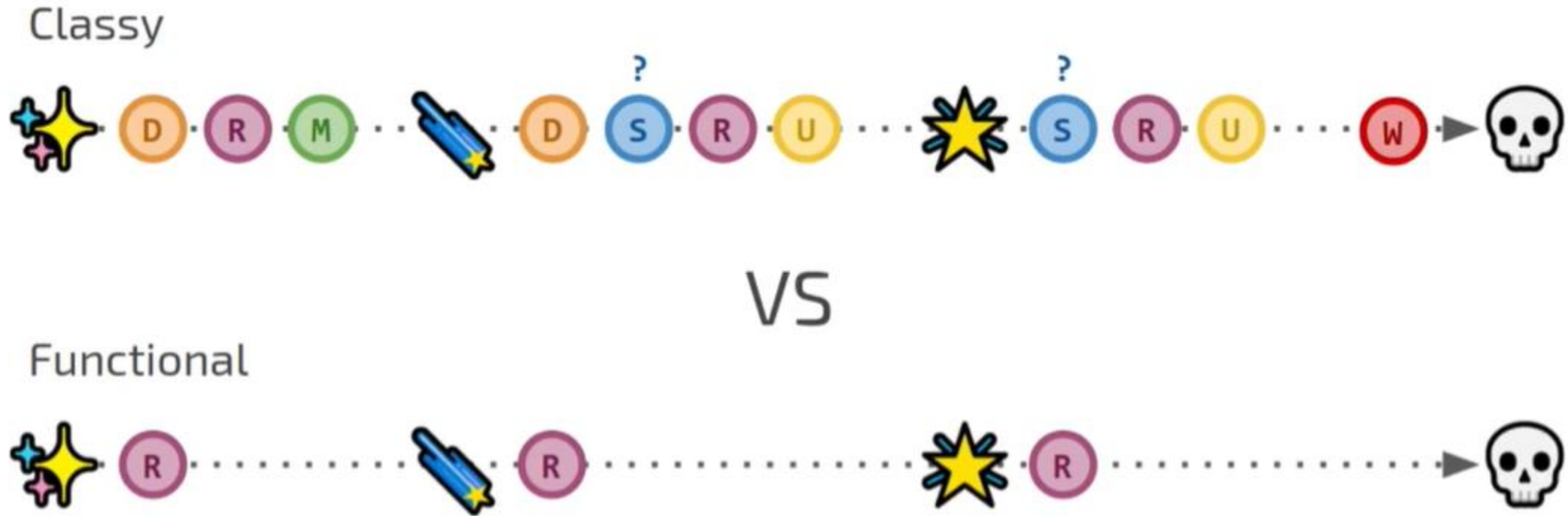


ESTADOS DEL CICLO DE VIDA

COMPONENTE CON ESTADO: QUE RECIBA NUEVAS PROPIEDADES Y CAMBIOS DE ESTADO

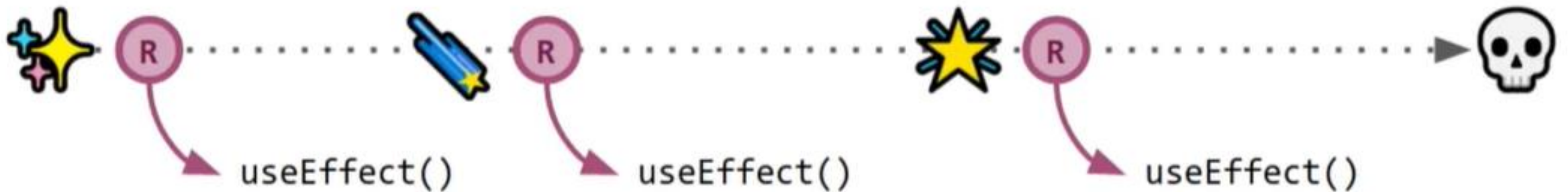


CICLO DE VIDA DE LOS COMPONENTES FUNCIONALES



CICLO DE VIDA DE LOS COMPONENTES FUNCIONALES

- Mediante el hook `useEffect`



USO DEL USEEFFECT



```
const MyComponent = () => {  
  useEffect(() => {  
    // Do stuff  
  });  
  return <div>Hello world</div>;  
};
```

USO DEL USEEFFECT.

EJEMPLO DE USO



```
const FullName = ({ name, surname }) => {  
  const [fullName, setFullName] = useState();  
  useEffect(() => {  
    setFullName(`${name} ${surname}`);  
  }, [name, surname]);  
  return <div>Hello {fullName}</div>;  
};
```


USO DEL USEEFFECT.

EJEMPLO DE USO MÁS AVANZADO



```
const Counter = () => {  
  const [counter, setCounter] = useState(0);  
  useEffect(() => {  
    const interval = setInterval(  
      () => setCounter(counter + 1), 1000  
    );  
    return () => clearInterval(interval);  
  }, [counter]);  
  return <p>{counter}</p>;  
};
```

USO DEL USEEFFECT.

EJEMPLO DE USO MÁS AVANZADO



```
const Counter = () => {  
  const [counter, setCounter] = useState(0);  
  useEffect(() => {  
    const interval = setInterval(  
      () => setCounter(c => c + 1), 1000  
    );  
    return () => clearInterval(interval);  
  }, []);  
  return <p>{counter}</p>;  
};
```



EJERCICIO

AÑADIR LÓGICA FUNCIONAL A LA WISHLIST

INTRODUCCIÓN A REACT. 4 - PROPS & STATE



Exercises!

1. Separate our Wish List application in several components. Use props to pass data down.
 - a. `WishlistInput`
 - b. `WishlistItem`
2. Add functionality to our application
 - a. The input should create new wishes to add to the list
 - b. The wishes checkbox should mark the wish as done
 - c. Buttons to archive completed wishes should make the be removed from the list
3. Every wish should be coloured depending on the time that has remained undone: orange (>10s), red (>20s).