REACT PARA PRINCIPIANTES

TEMA 4: PROPS & STATE

Material obtenido de Open Webinars

OpenWebinars

OpenWebinars

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

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

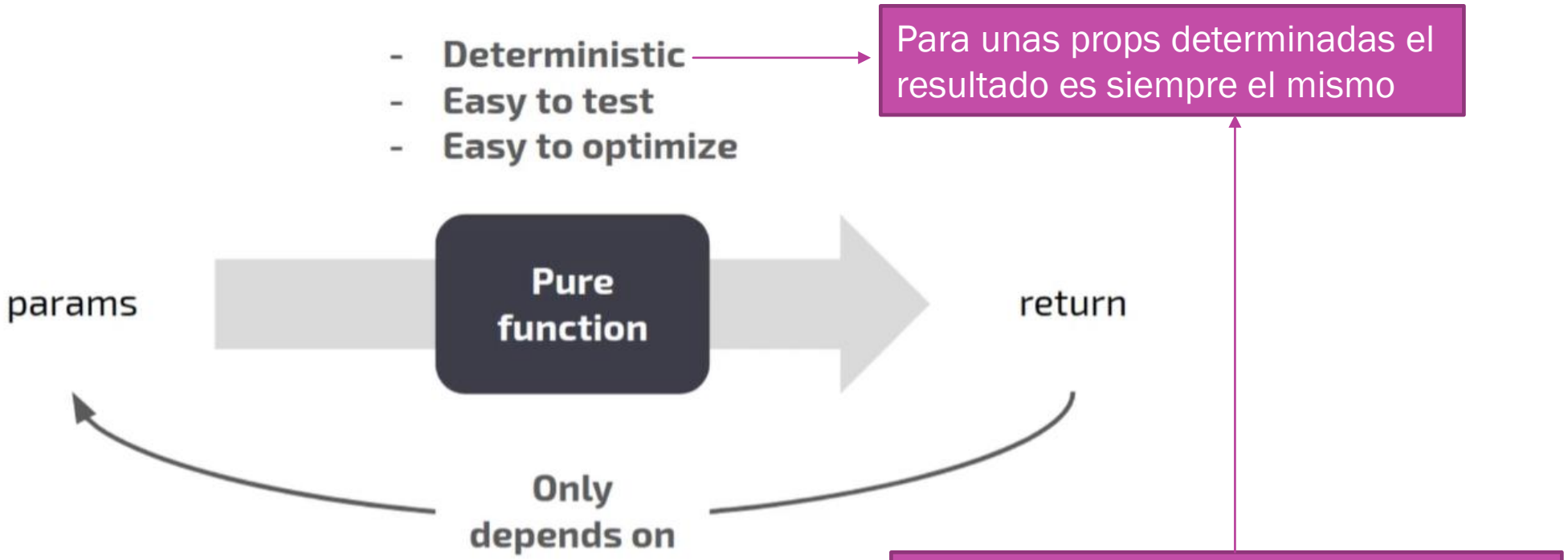
RECORDATORIO COMPONENTES

- 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>;
```



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

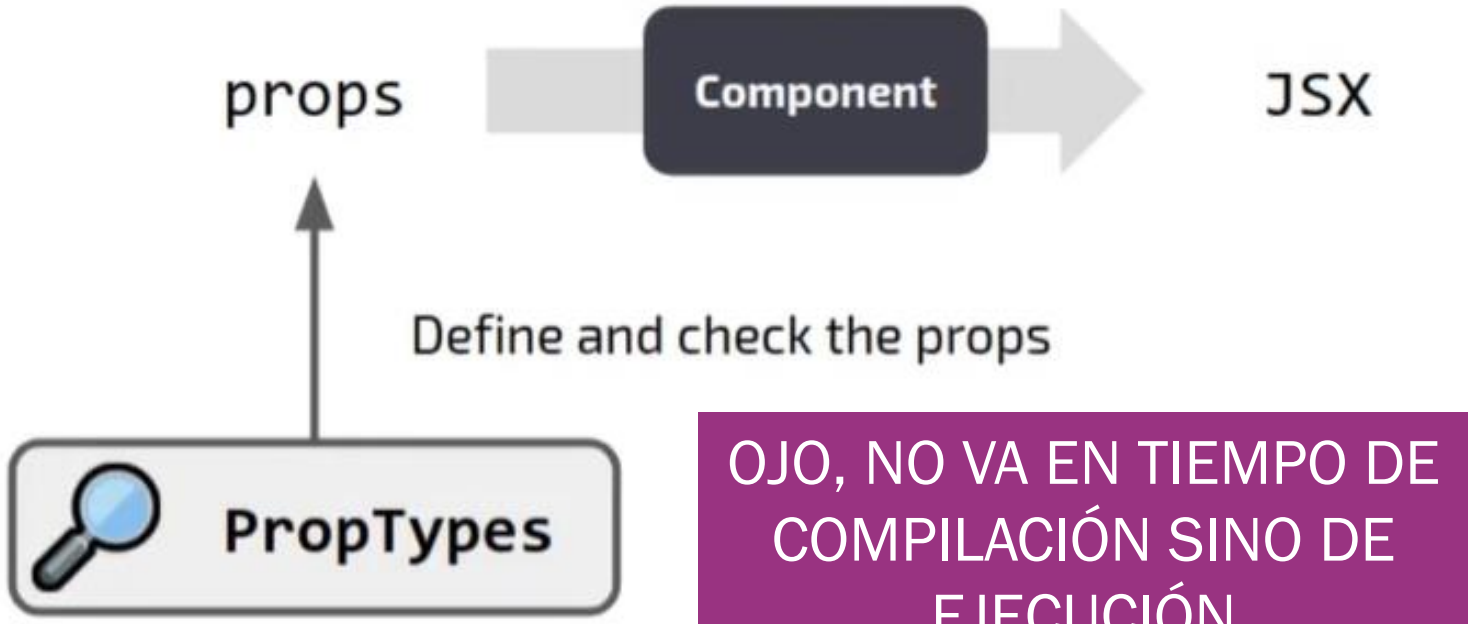
```
class Header extends Component {
  render() {
    const { label } = this.props;
    return <h1>{label}</h1>;
  }
}
```

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



OJO, NO VA EN TIEMPO DE COMPILACIÓN SINO DE EJECUCIÓN

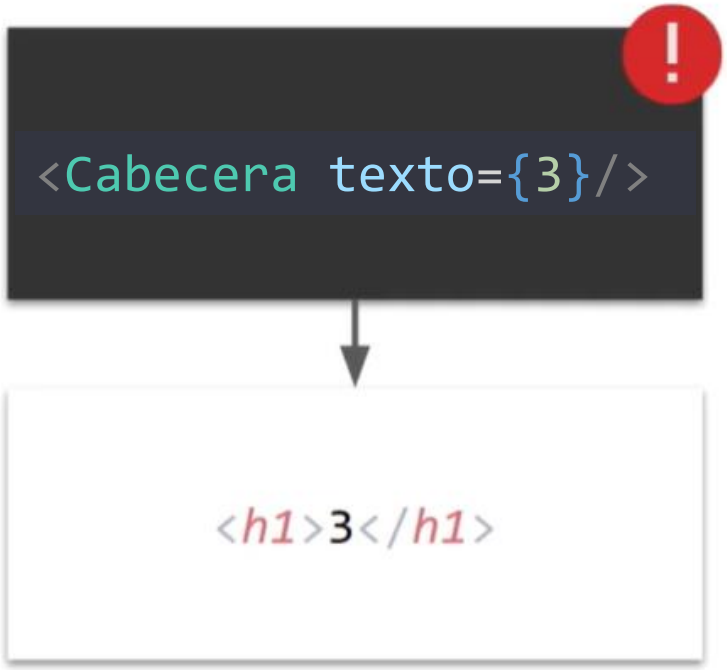
```
npm install --save prop-types
```

```
▶Warning: Failed prop type: Invalid prop `texto` of type `number` supplied to `Cabecera`, expected `string`.
    at Cabecera (http://localhost:3000/static/js/bundle.js:262:23)
    at App
```

EJEMPLO DE PROPTYPES

DEFINIMOS QUE ES DE TIPO STRING Y REQUERIDA

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

const Cabecera = (props) => {
    const texto=props.texto;
    return (<h1> {texto} </h1>);
}

Cabecera.propTypes = {
    texto: PropTypes.string.isRequired,
}
```



Se ponen bajo la definición del componente

Lo pinta, pero da warning en la consola del browser

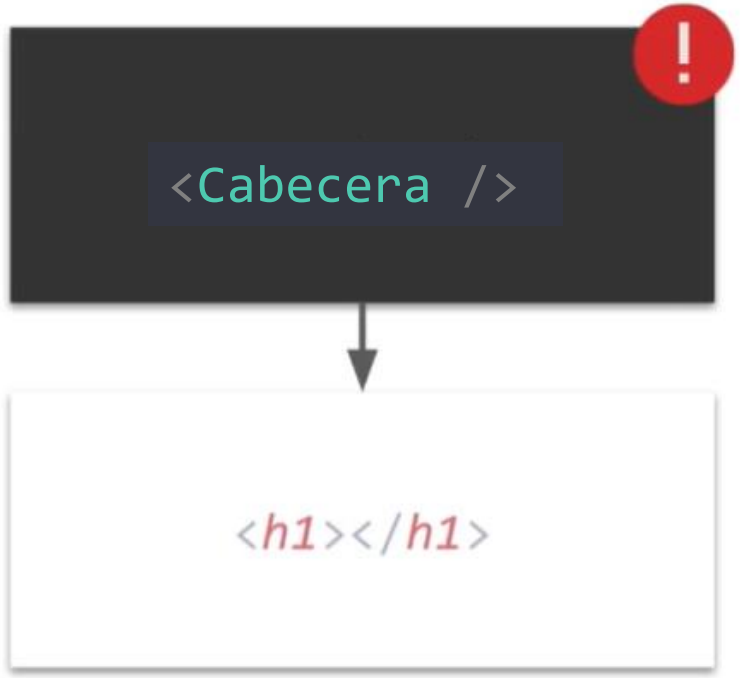
https://reactjs.org/docs/typechecking-with-proptypes.html

```
▶Warning: Failed prop type: The prop `texto` is marked as required in `Cabecera`,
but its value is `undefined`.
    at Cabecera (http://localhost:3000/static/js/bundle.js:262:23)
    at App
```

EJEMPLO DE PROPTYPES

DEFINIMOS QUE ES DE TIPO STRING Y REQUERIDA

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

const Cabecera = (props) => {
    const texto=props.texto;
    return (<h1> {texto} </h1>);
}

Cabecera.propTypes = {
    texto: PropTypes.string.isRequired,
}
```



EJEMPLO DE PROPTYPES

DEFINIMOS QUE ES DE TIPO STRING Y REQUERIDA

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

const Cabecera = (props) => {
    const texto=props.texto;
    return (<h1> {texto} </h1>);
}

Cabecera.propTypes = {
    texto: PropTypes.string.isRequired,
}
```

PROPTYPES. VALOR POR DEFECTO

```
import PropTypes from 'prop-types';
//Podemos acortar:
const Cabecera = ({texto}) =>
    <h1> {texto} </h1>

Cabecera.propTypes = {
    texto: PropTypes.string.isRequired,
}
Cabecera.defaultProps = {
    texto: 'My Wishlist',
}
export default Cabecera;
```

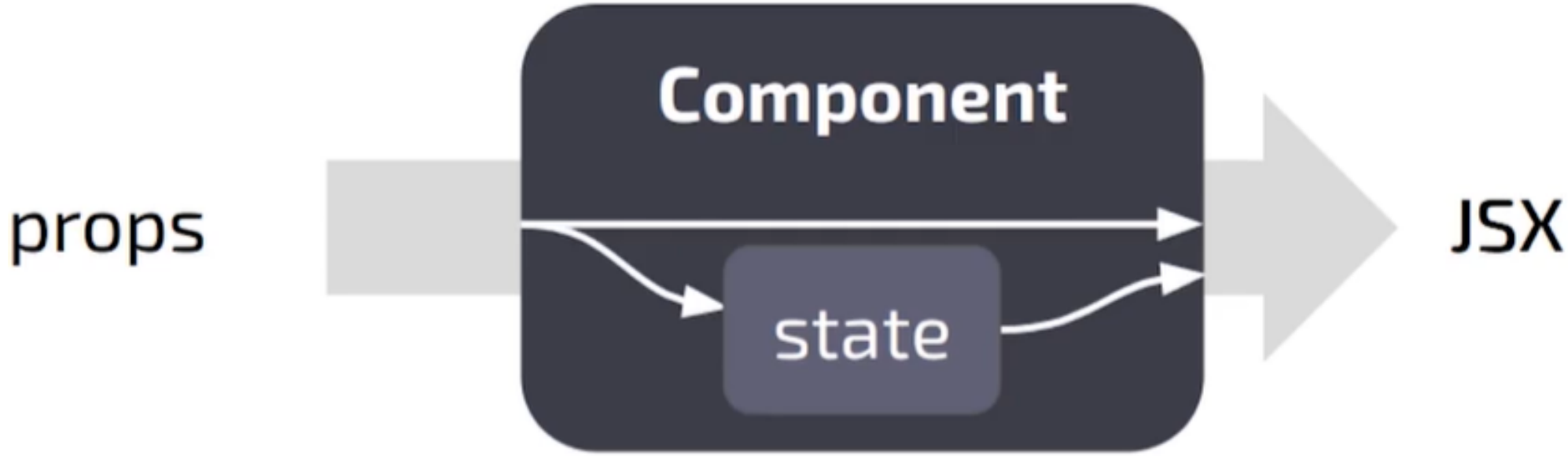
EVENTO SENCILLO

```
const BotonArchivar = () => {
    const handleClick = () => {alert ('has pinchado');}
    return (
        <button type="button" className='deseos-clear' onClick={handleClick}>
            Archivar deseos cumplidos.
        </button>
    );
}
```

https://es.reactjs.org/docs/handling-events.html

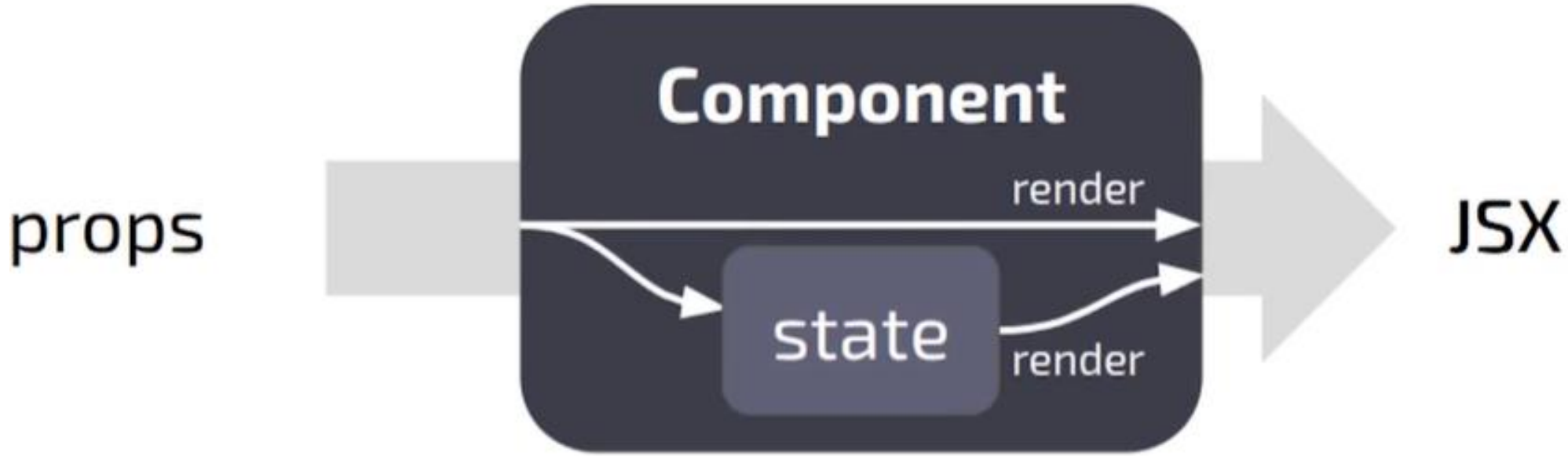
ESTADO INTERNO DE COMPONENTES

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



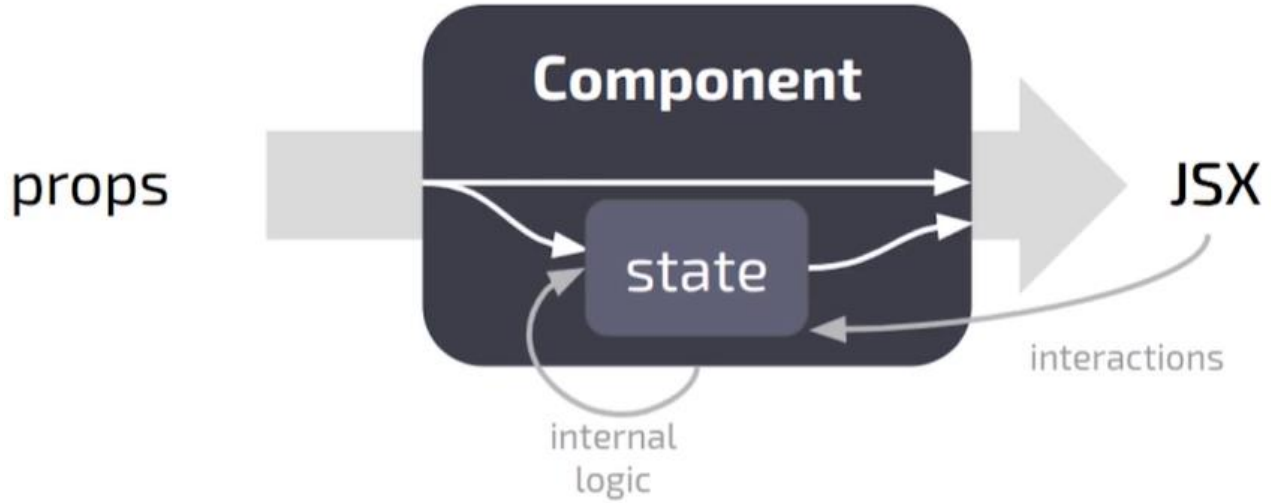
ESTADO INTERNO DE COMPONENTES

- 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



El estado en la página de React se explica con las clases, ya que antes era la única forma.
https://es.reactjs.org/docs/state-and-lifecycle.html

GESTIÓN DEL ESTADO

USESTATE HOOKS

- El hook de useState (desde 2018) permite usar el estado en componentes funcionales (antes solo clases)
- *defaultValue*: valor por defecto de la variable de estado.
- Devuelve un array con dos elementos:
  - *value:* valor de estado a fijar
  - *setValue*: función que se invoca para setear el valor de *value*

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

EJEMPLO DE HOOK

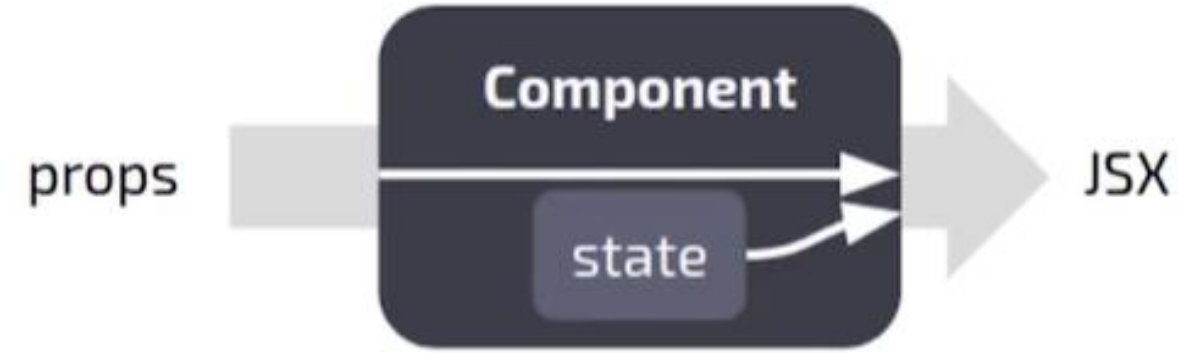
- Punto de partida: item de una lista

```
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’

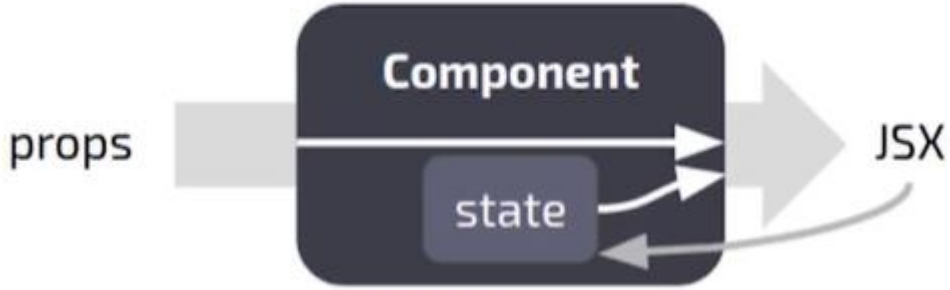
```
const TodoItem = ({ label }) => {
  const [checked] = useState(false);
  return (
    <p>
      {checked ? '✓' : 'X'} {label}
    </p>
  );
};
```



EJEMPLO DE HOOK

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

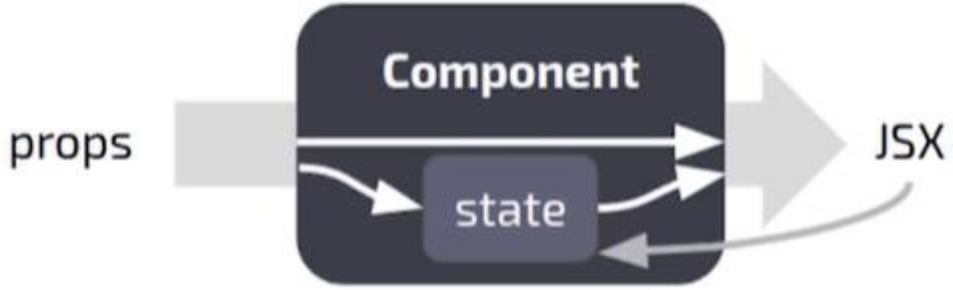
```
const TodoItem = ({ label }) => {
  const [checked, setChecked] = useState(false);
  return (
    <p onClick={() => setChecked(!checked)}>
      {checked ? '✓' : 'X'} {label}
    </p>
  );
};
```



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

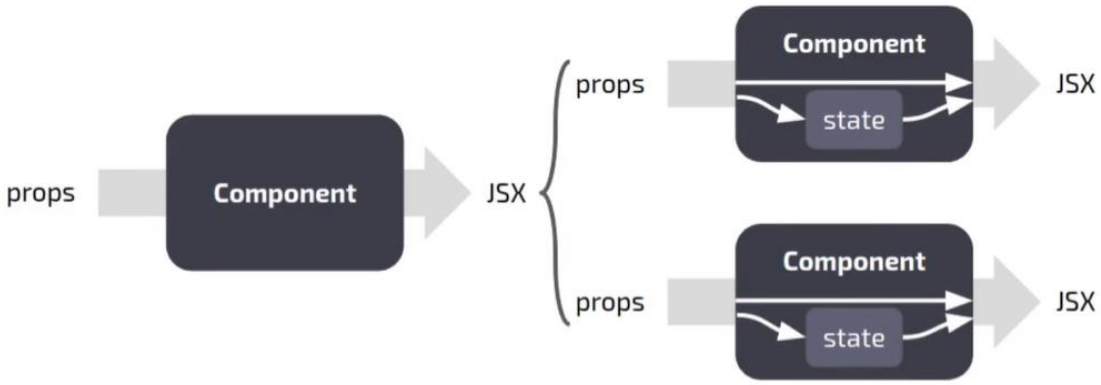
```
const TodoItem = ({ label, defChk }) => {
 const [checked, setChecked] = useState(defChk);
 return (
   <p onClick={() => setChecked(!checked)}>
     {checked ? '✓' : 'X'} {label}
   </p>
 );
};
```

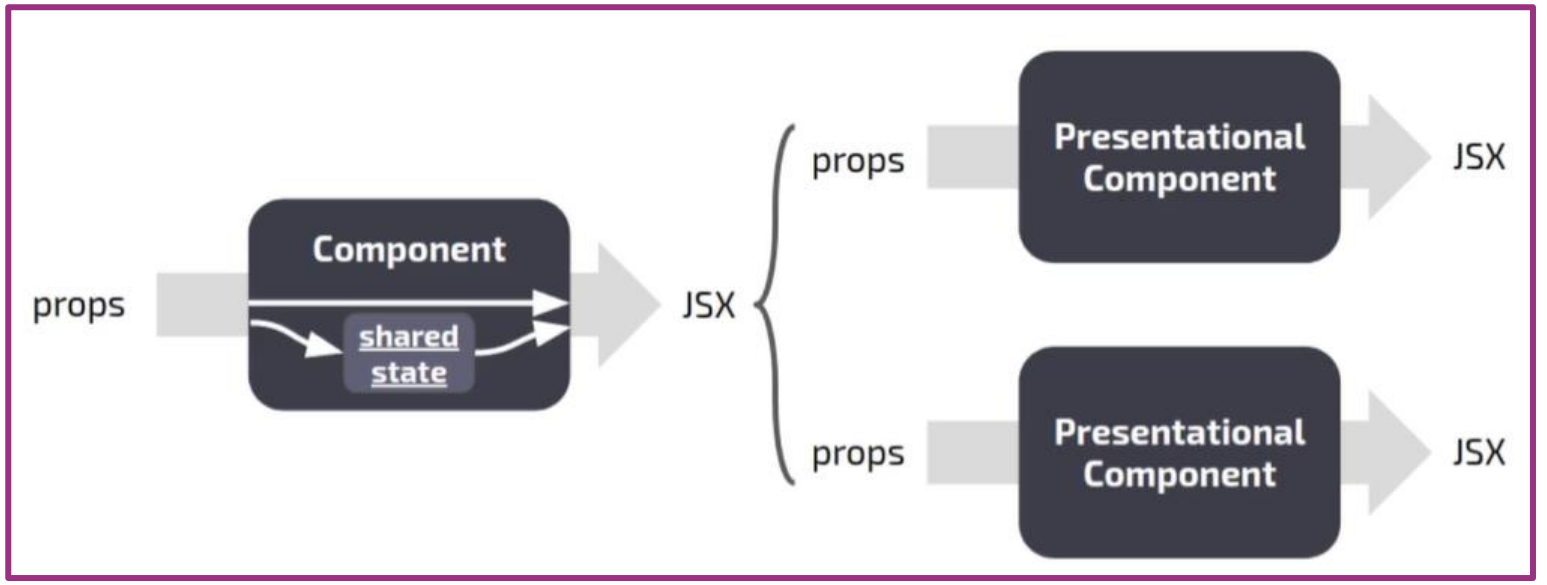
Class component

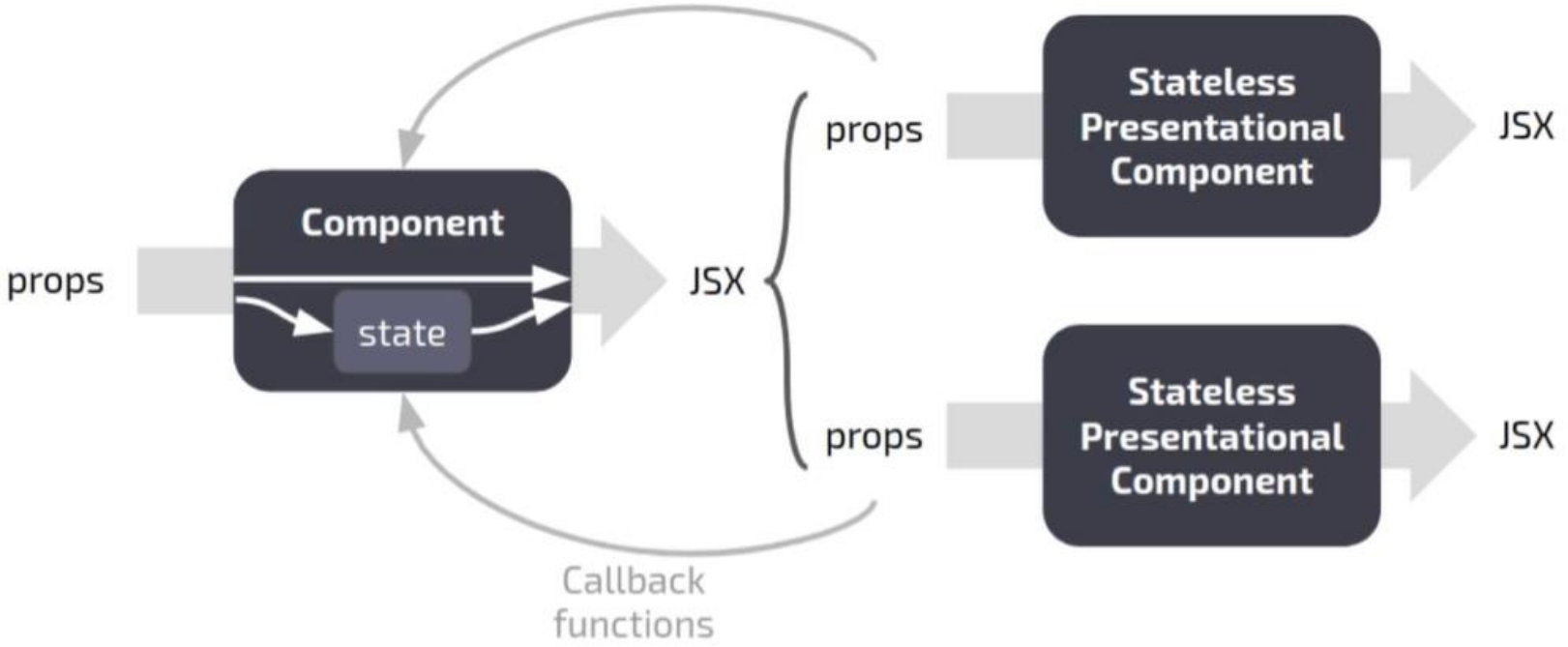
```
class TodoItem extends Component {
 state = { checked: this.props.defChk };

 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

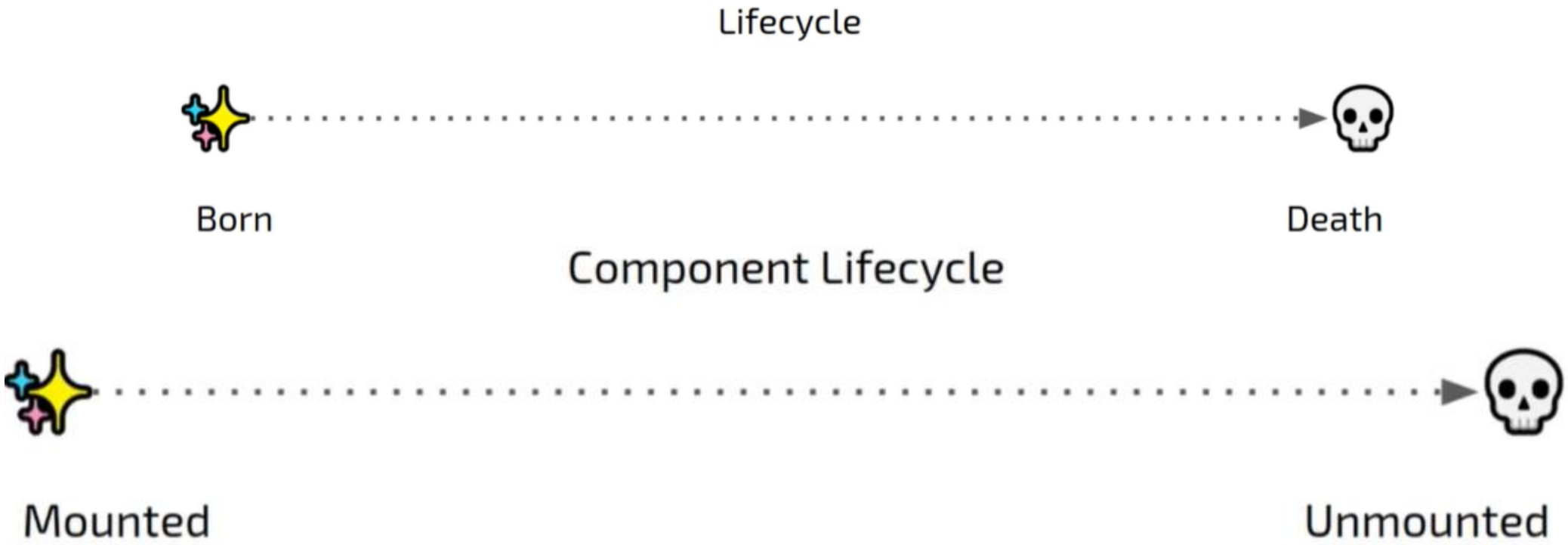
SE HACE MEDIANTE CALLBACKS

EJEMPLO DE ESTADO DELEGADO

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

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

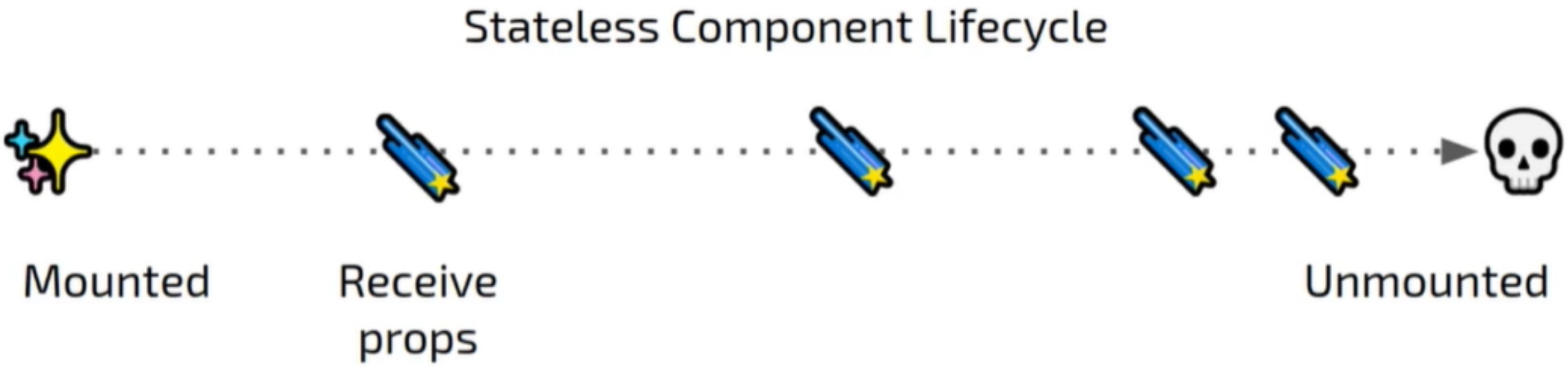
```jsx
const SearchDisplay = ({ search, onClear }) => (
 <div>
   <p>Current search: {search}</p>
   <button onClick={onClear}>Clear</button>
 </div>
);
```



CICLO DE VIDA DE UN COMPONENTE

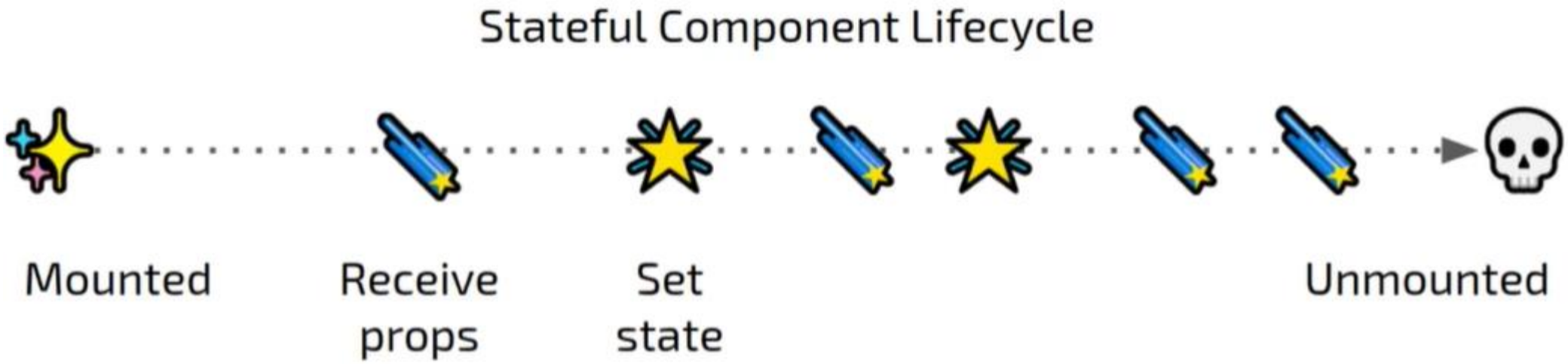
ESTADOS DEL CICLO DE VIDA

COMPONENTE SIN ESTADO: QUE RECIBA NUEVAS PROPIEDADES

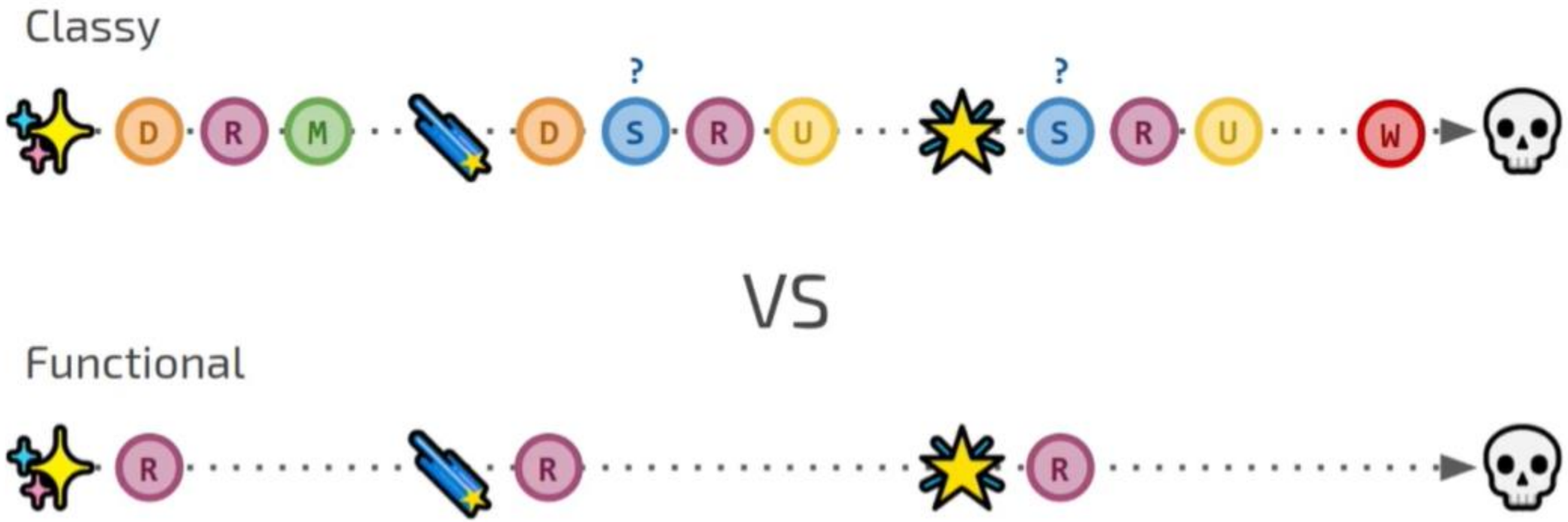


ESTADOS DEL CICLO DE VIDA

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

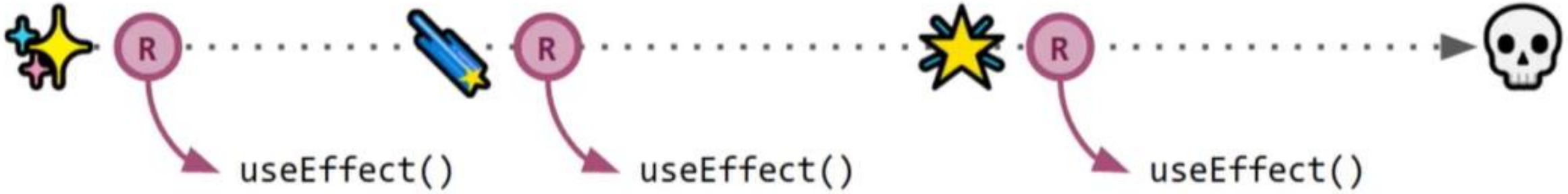


CICLO DE VIDA DE LOS COMPONENTES FUNCIONALES



CICLO DE VIDA DE LOS COMPONENTES FUNCIONALES

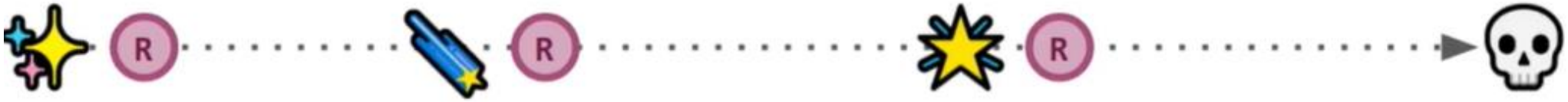
- Mediante el hook useEffect



USO DEL USEEFFECT

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

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).