# 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

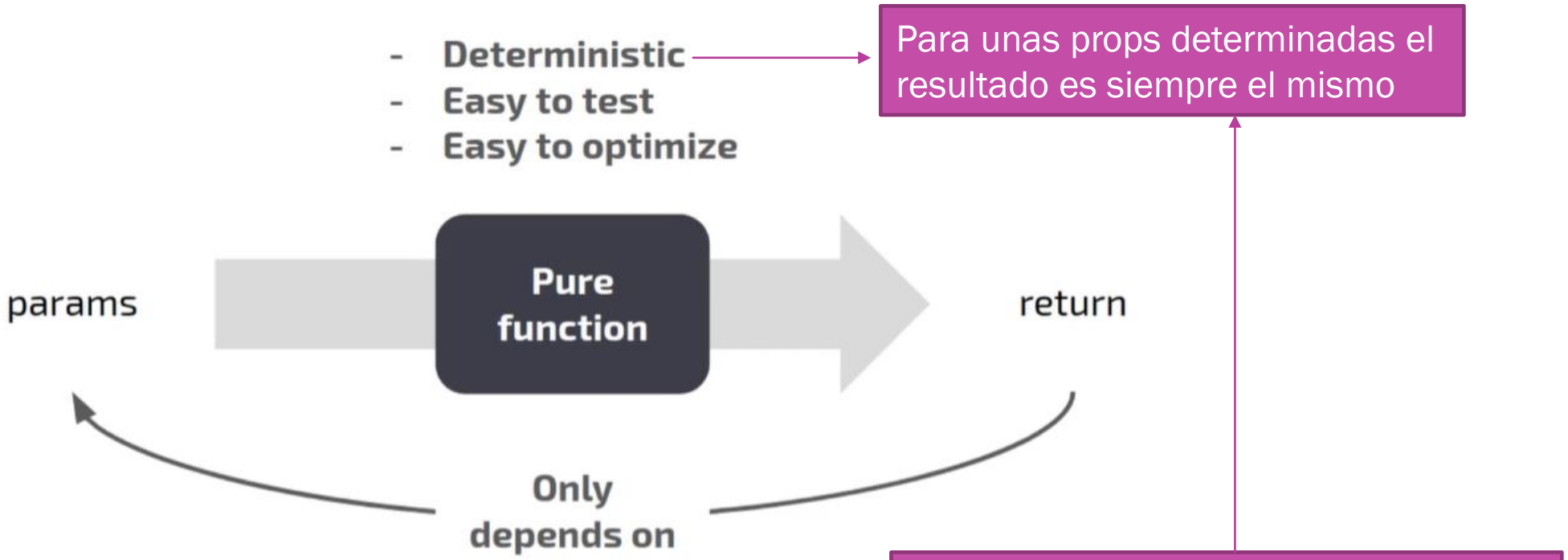
# RECORDATORIO COMPONENTES

- 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

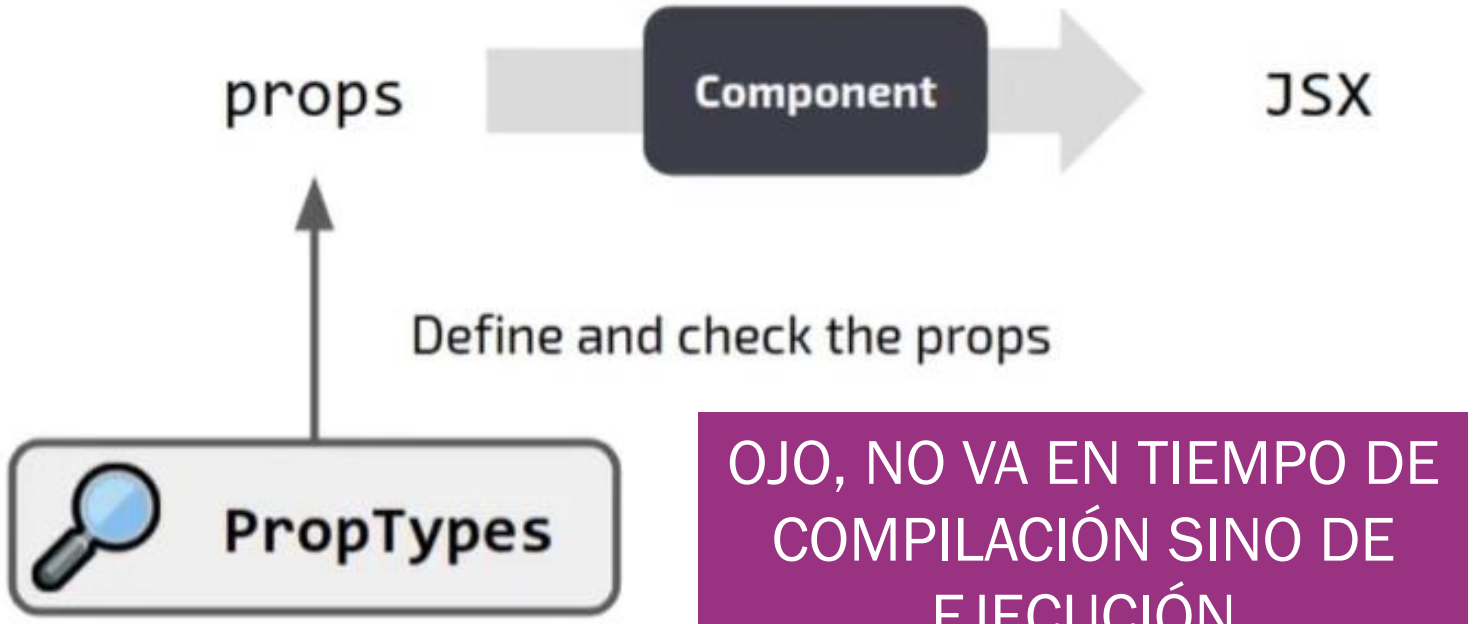
```
class Header extends Component {
  render() {
    const { label } = this.props;
    return <h1>{label}</h1>;
  }
}
```

# 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

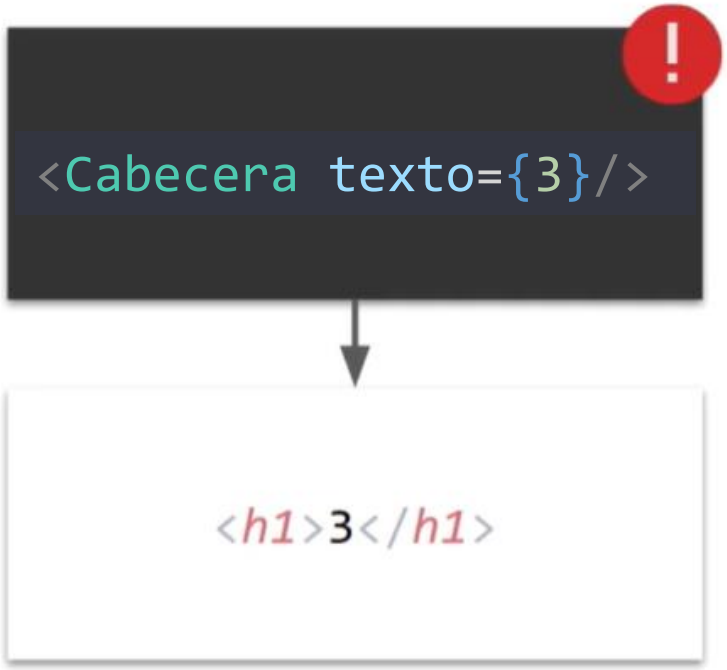
```
npm install --save prop-types
```

```
▶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

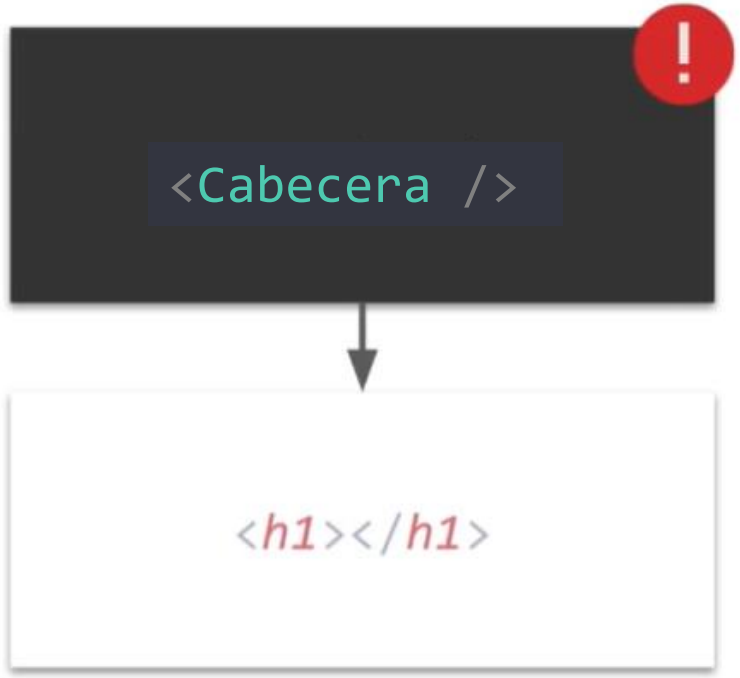
https://reactjs.org/docs/typechecking-with-proptypes.html

```
▶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;
```

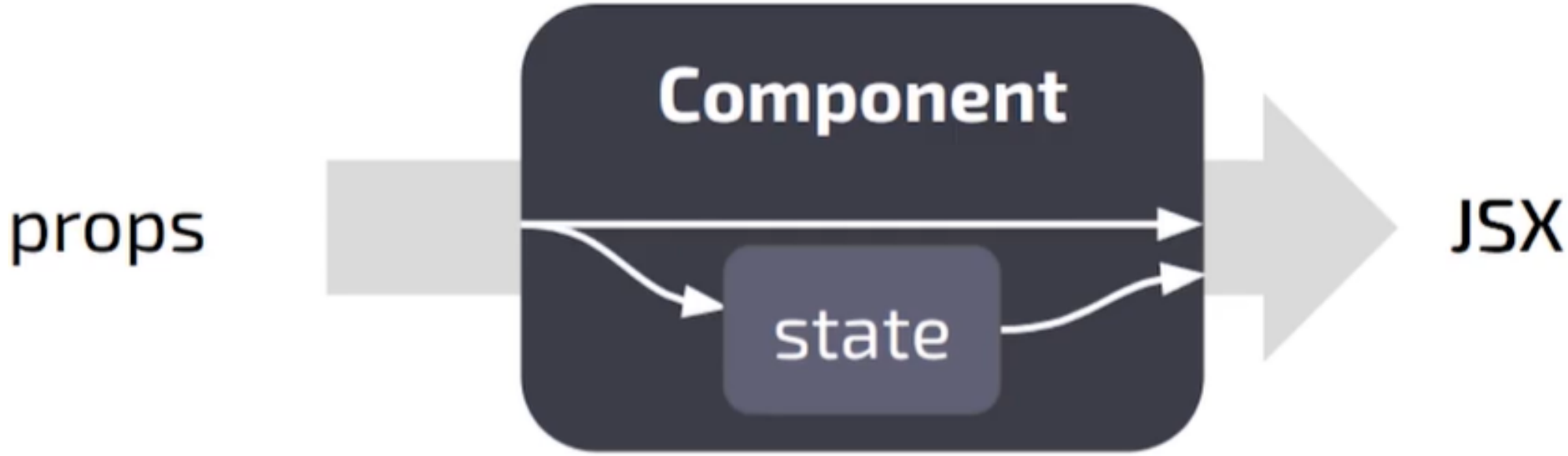
# EVENTO SENCILLO

```
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

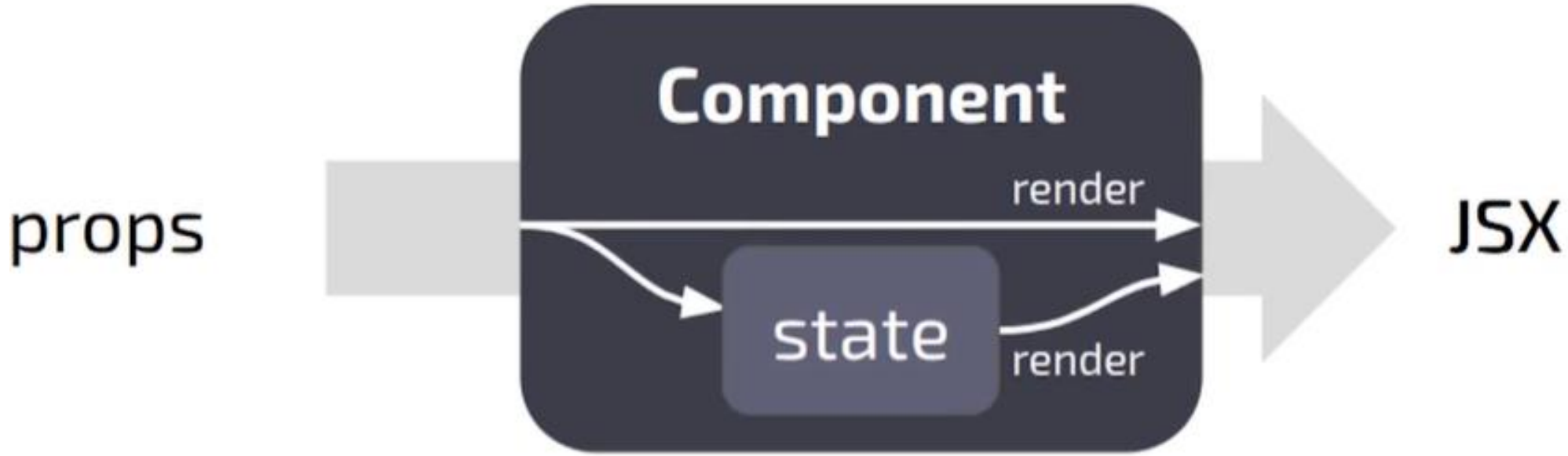
# ESTADO INTERNO DE COMPONENTES

- 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



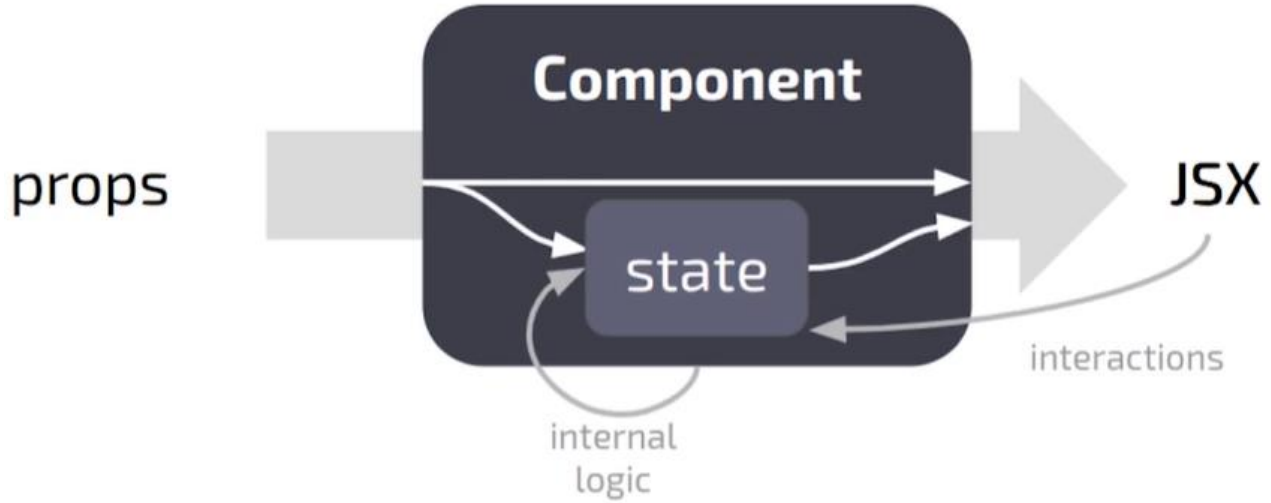
# ESTADO INTERNO DE COMPONENTES

- 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

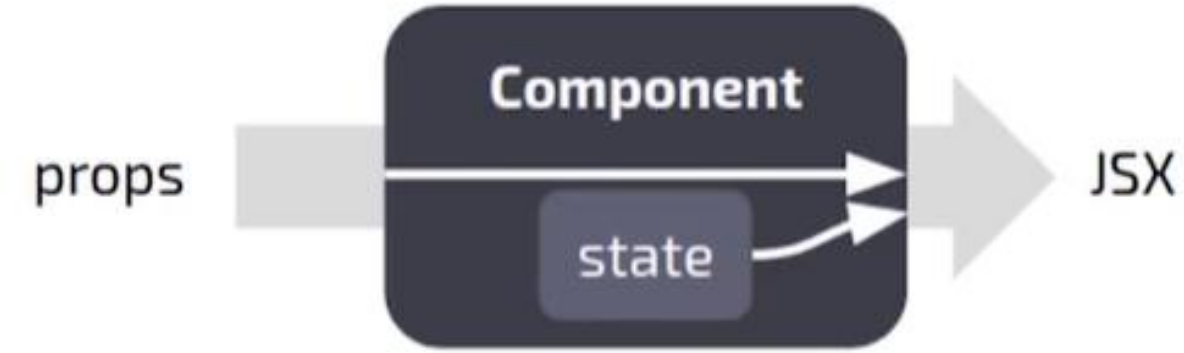
- Punto de partida: item de una lista

```
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’

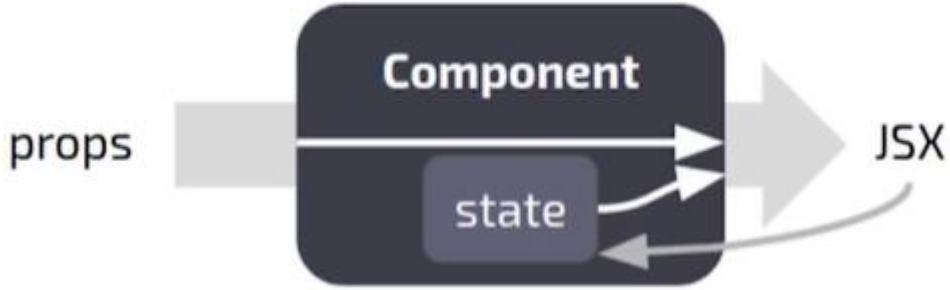
```
const TodoItem = ({ label }) => {
  const [checked] = useState(false);
  return (
    <p>
      {checked ? '✓' : 'X'} {label}
    </p>
  );
};
```



# EJEMPLO DE HOOK

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

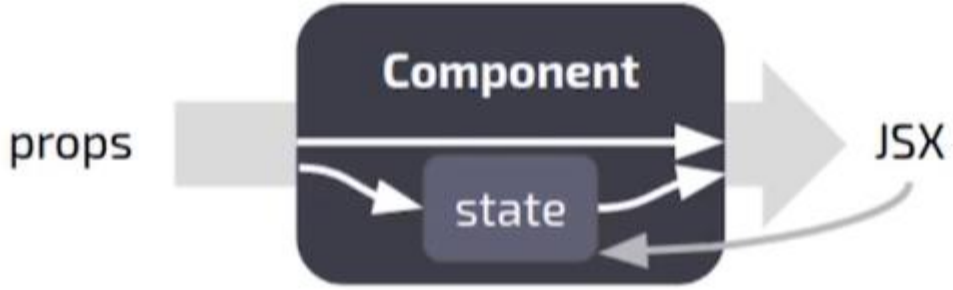
```
const TodoItem = ({ label }) => {
  const [checked, setChecked] = useState(false);
  return (
    <p onClick={() => setChecked(!checked)}>
      {checked ? '✓' : 'X'} {label}
    </p>
  );
};
```



# 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

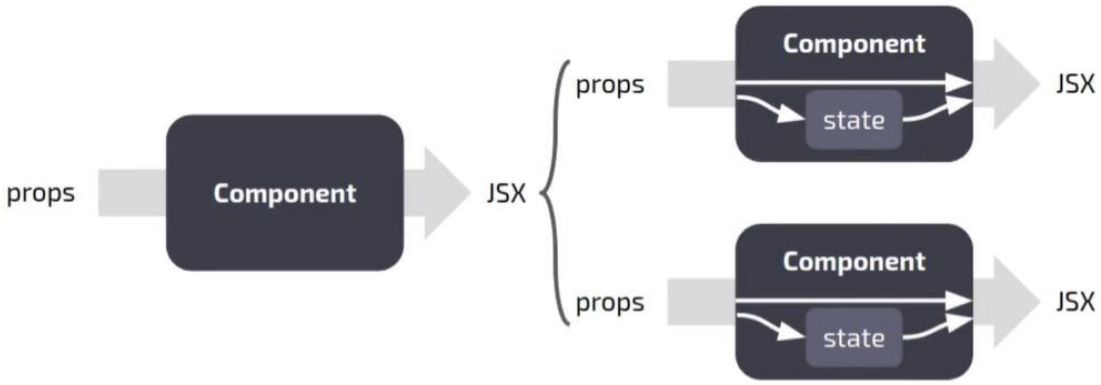
```
const TodoItem = ({ label, defChk }) => {
 const [checked, setChecked] = useState(defChk);
 return (
   <p onClick={() => setChecked(!checked)}>
     {checked ? '✓' : 'X'} {label}
   </p>
 );
};
```

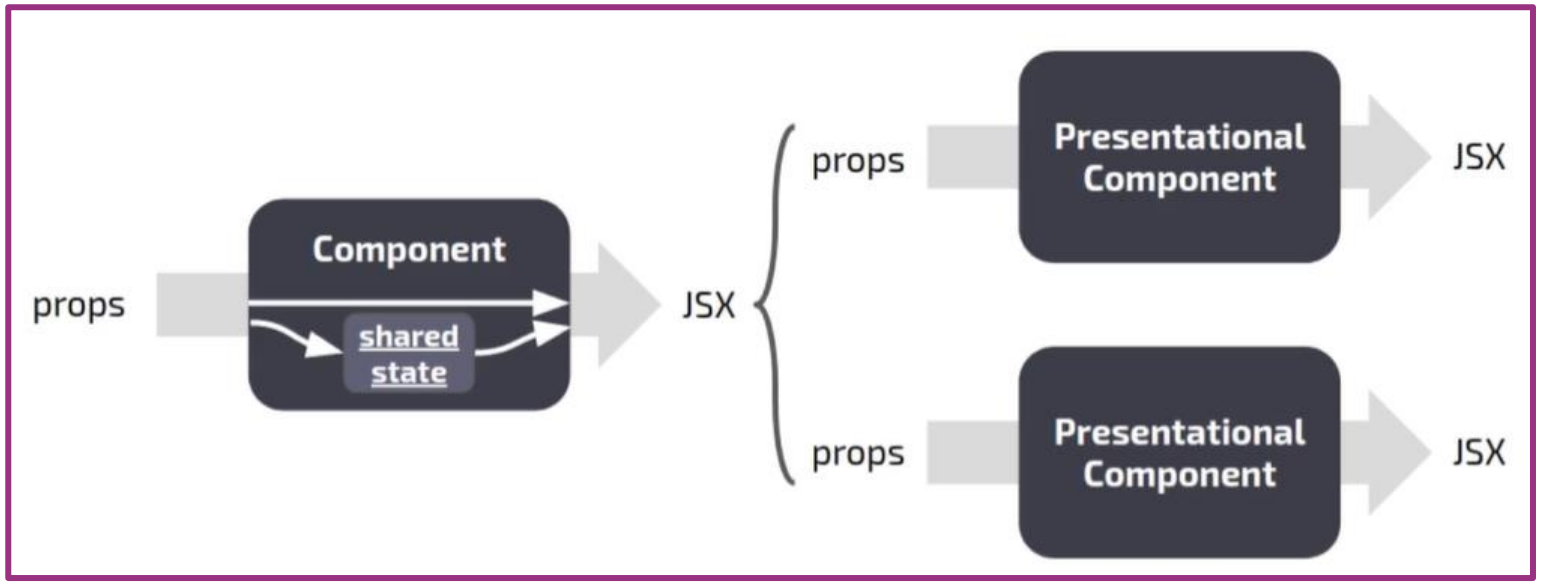
Class component

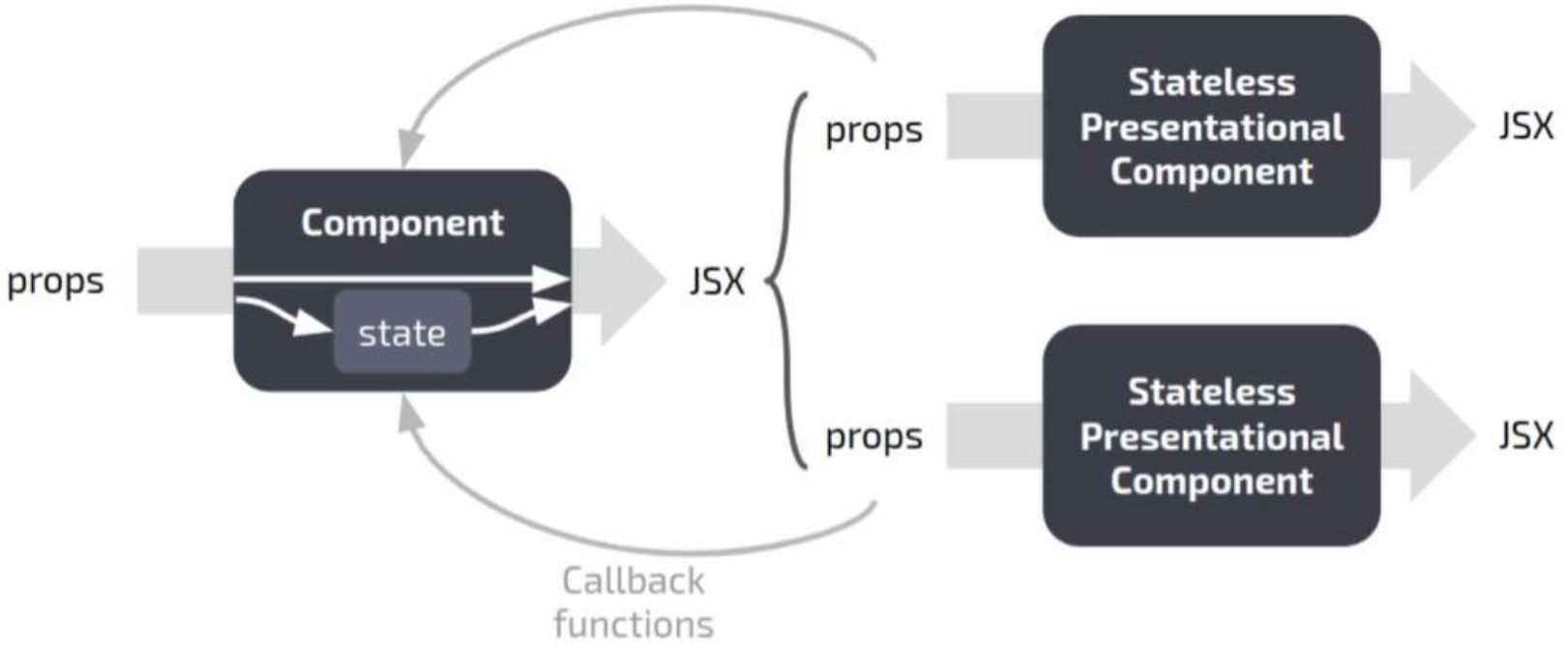
```
class TodoItem extends Component {
 state = { checked: this.props.defChk };

 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

Callback: onSearchChange para avisar al padre desencadenando evento de que debería cambiar el valor del search. Así los componentes hijos no cambian el estado, avisan con el evento y lo cambia el padre (delegación).

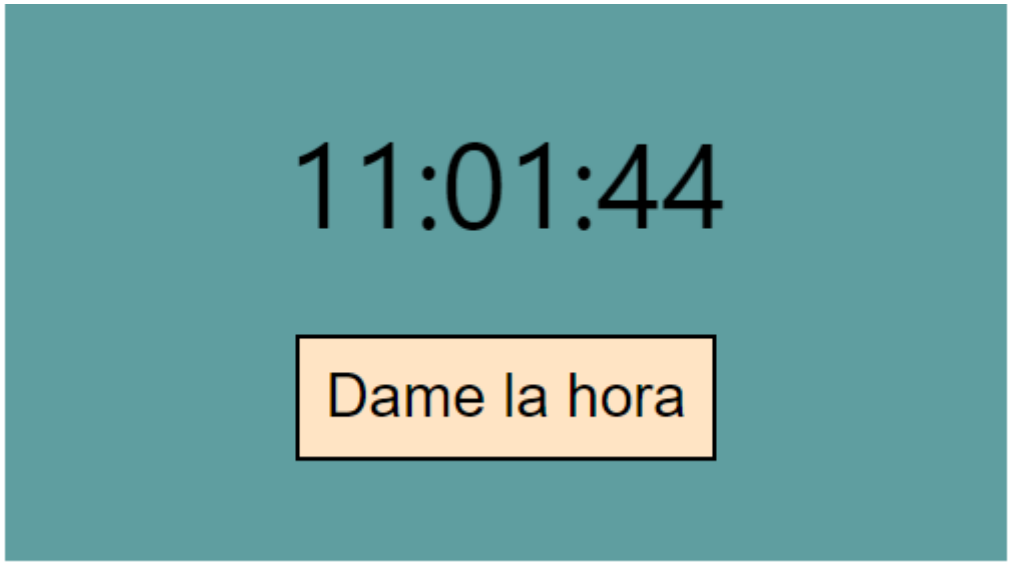
```jsx
const Search = () => {
 const [search, setSearch] = useState('');
 return (
   <div>
     <SearchInput
       search={search}
       onSearchChange={setSearch}
     />
     <SearchDisplay
       search={search}
       onClear={() => setSearch('')}
     />
   </div>
 );
};
```

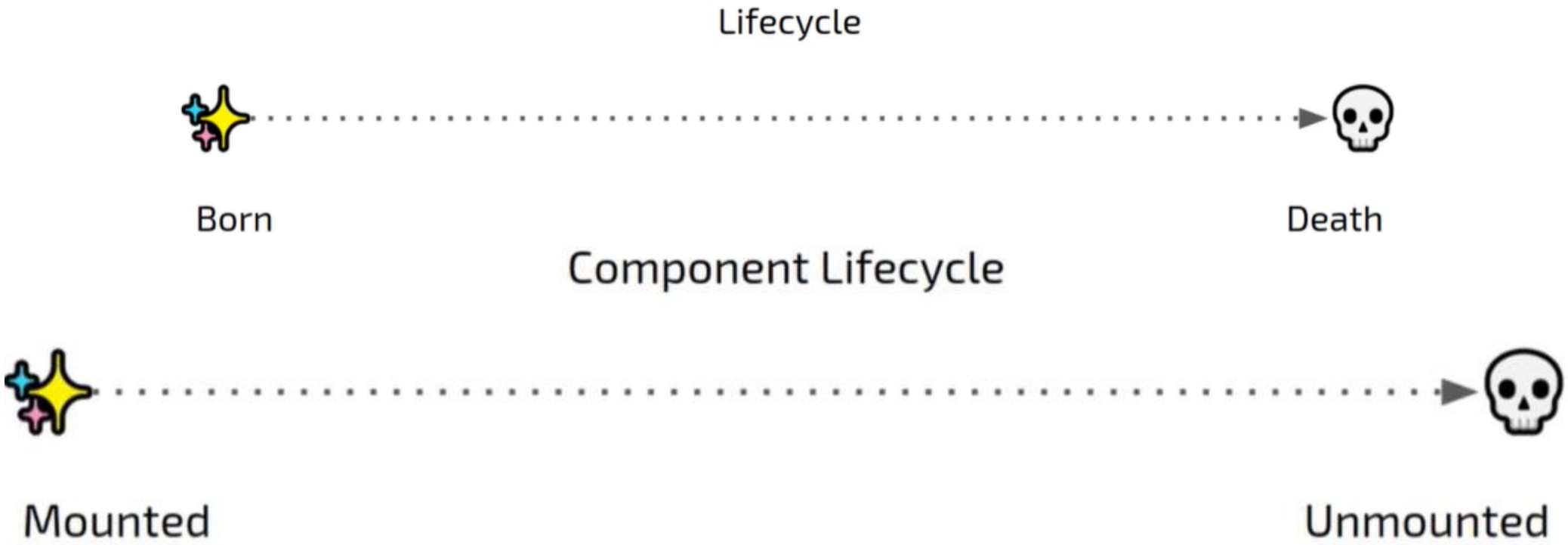
```jsx
const SearchInput = ({search, onSearchChange}) => (
 <input
   value={search}
   onChange={e => onSearchChange(e.target.value)}
 />
);
```

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

# ACTIVIDAD: COMPONENTE RELOJ

- Vamos a crear un nuevo proyecto llamado Reloj cuyo componente principal sea un Reloj como se muestra en la figura donde al pulsar el botón nos muestre la hora actual

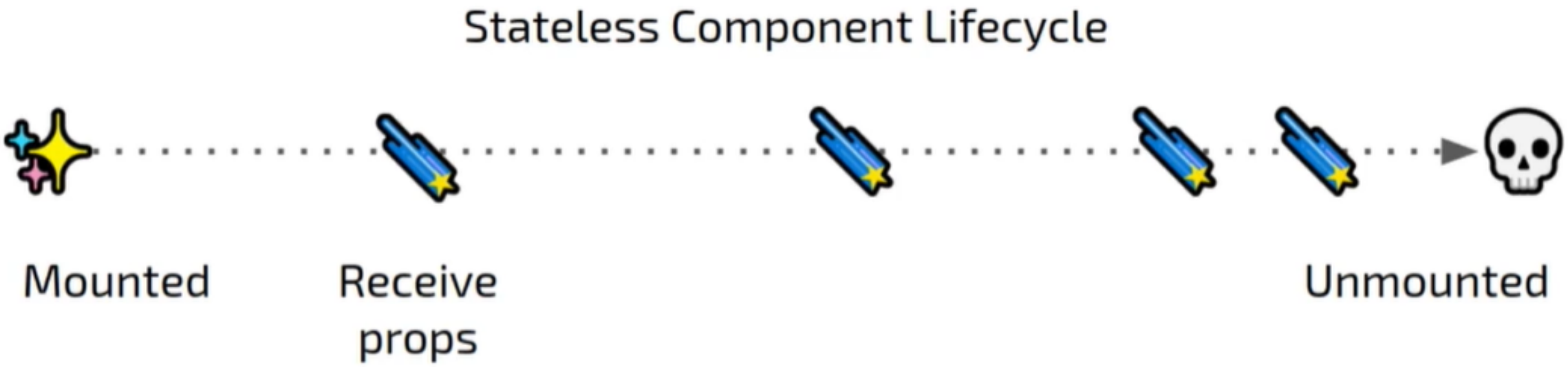





# CICLO DE VIDA DE UN COMPONENTE

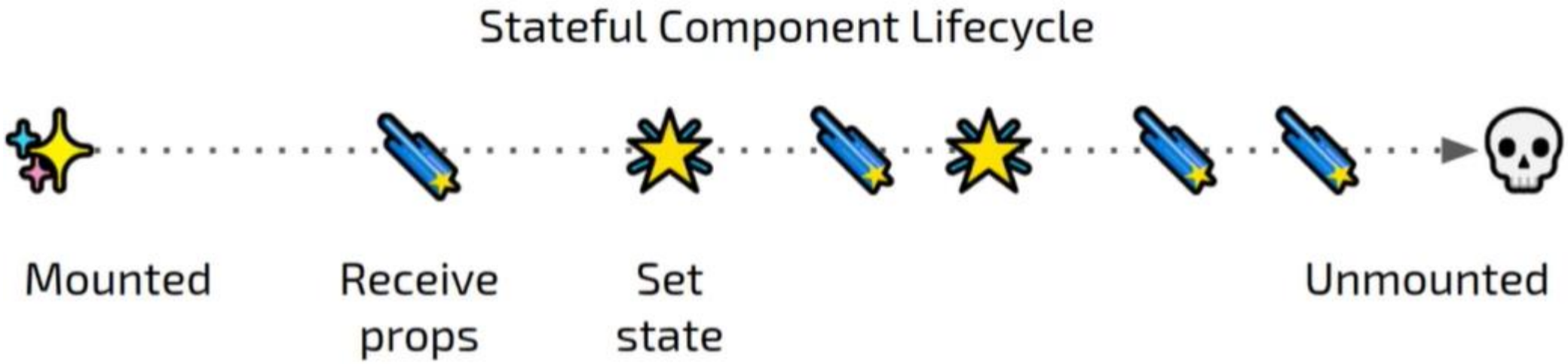
# ESTADOS DEL CICLO DE VIDA

## COMPONENTE SIN ESTADO: QUE RECIBA NUEVAS PROPIEDADES

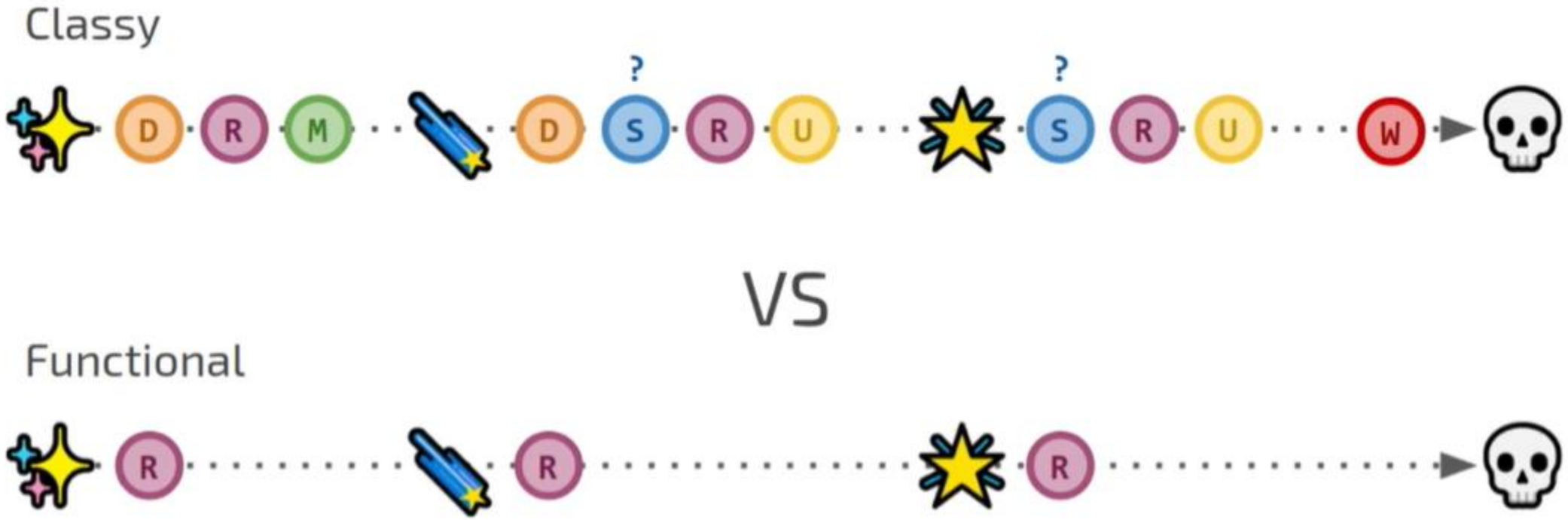


# ESTADOS DEL CICLO DE VIDA

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



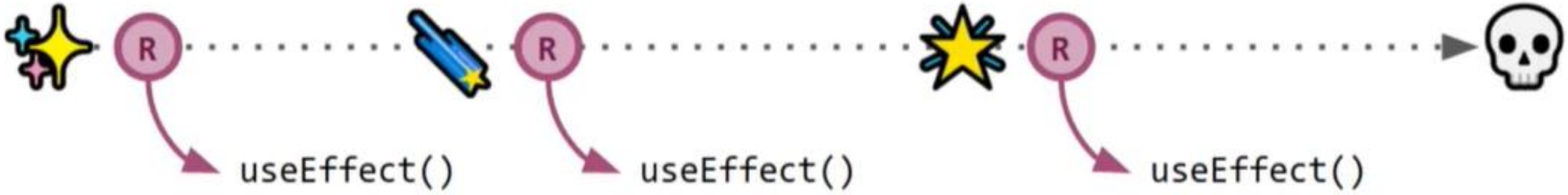
# CICLO DE VIDA DE LOS COMPONENTES FUNCIONALES



# EFECTOS

- Un efecto en React es código que se ejecuta después del render. Es decir, después de que el componente se monte y se ejecute el primer render.
- Esto antes se hacía con el ciclo de vida de las clases, ahora con el Hook useEffect

https://beta.reactjs.org/learn/synchronizing-with-effects



# USO DEL HOOK USEEFFECT

Nos permite ejecutar trozos de código según el momento en el que esté el ciclo de vida del componente.
SINTAXIS:

- Función que recibe como primer parámetro un callback (en este caso es una fc anónima)
- Dentro de ella realizamos las acciones que responderán a los diferentes eventos

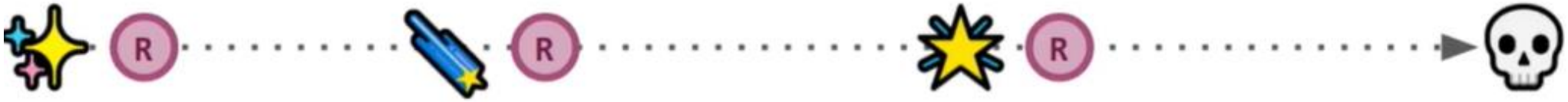
# EQUIVALENCIA CON CICLO DE VIDA DE CLASES

```
useEffect(
    ()=> {
        // Fn body
        return cleanFn;
    },
    [memo, deps]
)
```

```
componentDidMount()
componentDidUpdate()
shouldComponentUpdate()
componentWillUnmount()
```

## USO DEL USEEFFECT.

# EJEMPLO



Este array nos indica de qué valores depende la ejecución de esta lógica interna.
SOLO CUANDO ALGUNO DE LOS DOS CAMBIE SE LANZARÁ ESTA FUNCIÓN.
Así evitamos que se lance en cada render.

```
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

Este callback tiene un return, lo que hace es devolver dentro del callback una nueva fc que se va a ejecutar cuando el componente se destruya (o se desmonte). Limpiamos lo que nos interesa antes de que se destruya el componente

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

Aquí encontramos un problema, porque se lanza el useEffect cada vez que se actualiza el counter, que a su vez se hace dentro.... Por lo que debería eliminar el counter como dependencia para que este useEffect solo se ejecute una vez

## USO DEL USEEFFECT.

# EJEMPLO DE USO MÁS AVANZADO


De este modo ya no dependemos del counter, por tanto, el useEffect solo se ejecutará una vez.

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

Para probarlo, lo podemos añadir a nuestro Proyecto Reloj

# ACTIVIDAD

- Actualiza el componente Reloj para que muestre la hora cada segundo, utilizando hooks de efecto tal que se ejecute una sola vez

# 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).

# CONSTRUIR EL DESEOINPUT

- Tratamos de que sea una cápsula independiente cuya funcionalidad sea crear nuevos deseos.
- Creamos *prop* de tipo *callback* al que queremos llamar cuando se **crea un deseo** con un evento →el prop se llamará *onNuevoDeseo*

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

const DeseoInput = ({onNuevoDeseo}) =>
   <fieldset className='deseo-input'>
     <legend className='deseo-input__label'>New wish: </legend>
     <input className='deseo-input__field' placeholder='Enter your wish here'/>
   </fieldset>

DeseoInput.propTypes = {
    onNuevoDeseo: PropTypes.func,
}
DeseoInput.defaultProp = {
    onNuevoDeseo: ()=>{},
}
export default DeseoInput;
```

Añadimos el callback *onNuevoDeseo* como *prop* y abajo sus *propTypes* (es una función y su valor por defecto es la función vacía)

# CREAMOS EL HOOK DE ESTADO

- Querremos que se lance la función cuando se pulse ENTER (onKeyUp)
- Mientras, vamos guardando en nuestro estado el valor del input de texto en: *nuevoDeseoTexto*

```
const [nuevoDeseoTexto,setNuevoDeseoTexto]=useState('');
```

- ¿Cuándo? Cuando el input cambie

```
<input className='deseo-input__field' placeholder='Enter your wish here'
value={nuevoDeseoTexto} onChange={e=>setNuevoDeseoTexto(e.target.value)}/>
```

FALTA onKeyUp

# EVENTO ONKEYUP

```
<input className='deseo-input__field' placeholder='Enter your wish here'
       value={nuevoDeseoTexto} onChange={e=>setNuevoDeseoTexto(e.target.value)}
       onKeyUp={e => {
              if (e.key === 'Enter' && nuevoDeseoTexto.length) {
                     onNuevoDeseo({texto:nuevoDeseoTexto, cumplido:false })
                     setNuevoDeseoTexto('');//limpiamos el input
              }
       }}
/>
```

- Ya podemos capturar desde fuera (desde App) el evento ***onNuevoDeseo***

Vamos a App.jsx

# RECOGEMOS EL EVENTO EN APP

- Necesitamos crear un estado para modificar la lista de deseos (la constante externa *deseosIniciales* no la podemos modificar)

```
const [deseos, setDeseos] = useState(deseosIniciales);
```

- Modificamos la llamada al componente para decirle lo que tiene que hacer cuando recibe el callback *onNuevoDeseo*. En nuestro caso, añadimos el nuevo deseo al principio de la lista

```
<DeseoInput onNuevoDeseo={deseo => setDeseos([deseo, ...deseos ])} />
```

# CONTINUAMOS..

- Nos falta solamente el botón archivar.
- Cuando lo hagamos nos daremos cuenta de algo raro. ¿qué está pasando?
- Cuando modificábamos el estado de los checkboxes no modificábamos la lista de deseos sino su parte visual. Debemos corregir esto, ahora que ya sabemos, para conseguir la funcionalidad completa.

Una vez está completa la funcionalidad nos falta añadir los hooks de efecto

# HOOK DE EFECTO

3. Every wish should be coloured depending on the time that has remained undone: orange (>10s), red (>20s).

- En cada WishItem o DeseoItem, por antigüedad, vamos a ir pasándolo de naranja a rojo.

- Creamos variable de estado con hook de estado: edad
- Añadimos dos nuevas clases en nuestro css y nuestro className usando los modificadores  - -warning  para naranja y  - - danger para rojo

```
const [edad, setEdad] = useState(0); //edad: segundos que lleva sin completarse


        <li className={classNames(
            'lista-deseos__item',{
                'lista-deseos__item--cumplido': cumplido,
                'lista-deseos__item--warning' : edad >= 10 && edad <20,
                'lista-deseos__item--danger' : edad >= 20,
            }
        )}>
```

- Para darle uso al setEdad empleamos el useEffect.
- Me interesa iniciar el temporizador cuando cambie el valor de cumplido en cada deseo, en concreto cuando es falso.

```
useEffect (()=> {
     let intervalo;
     if (cumplido)
          setEdad(0)
     else
          intervalo = setInterval (()=>{/*ejecutar cada segundo esto*/},1000);
},[cumplido]);
```

- Cada segundo incrementamos el valor anterior, así ya no dependemos de edad, sino del valor anterior→ *setEdad (c=>c+1)*
- Cuando el deseo se cumpla, dicho valor de temporizador debe resetearse → *if cumplido clearInterval (intervalo)*

```
useEffect (()=> {
            let intervalo;
            if (cumplido)
                setEdad(0)
            else
                intervalo = setInterval (()=>{
                    if (cumplido) clearInterval (intervalo);
                    else setEdad (c=>c+1); //mejor poner esto que edad + 1
                    },1000);
        },[cumplido]);
```

- Cuando el componente se destruya (desaparezca de la interfaz por algún motivo) debe limpiarse también el intervalo

```
useEffect (()=> {
            let intervalo;
            if (cumplido)
                setEdad(0)
            else
                intervalo = setInterval (()=>{
                    if (cumplido) {clearInterval (intervalo);}
                    else {setEdad (c=>c+1);}
                    },1000);

            return () => clearInterval (intervalo);
        },[cumplido]);
```

# FINAL

FINAL DEL TEMA DE INTRODUCCIÓN A REACT