React intermedio

Gestión del estado

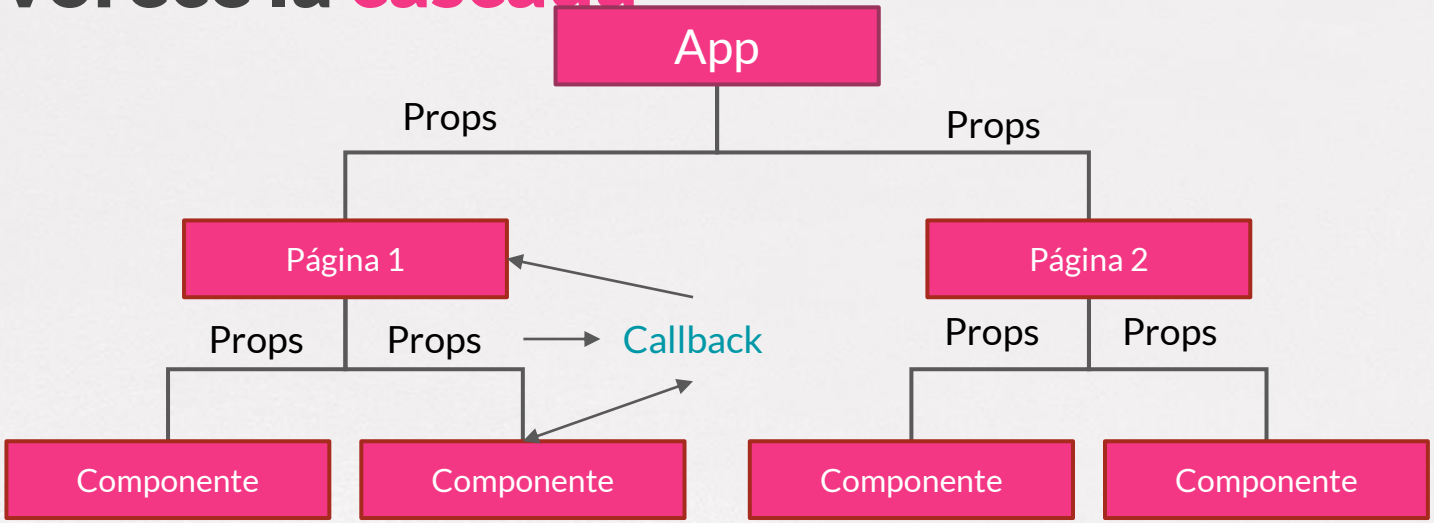
Estado y buenas prácticas

OpenWebinars

ÍNDICE

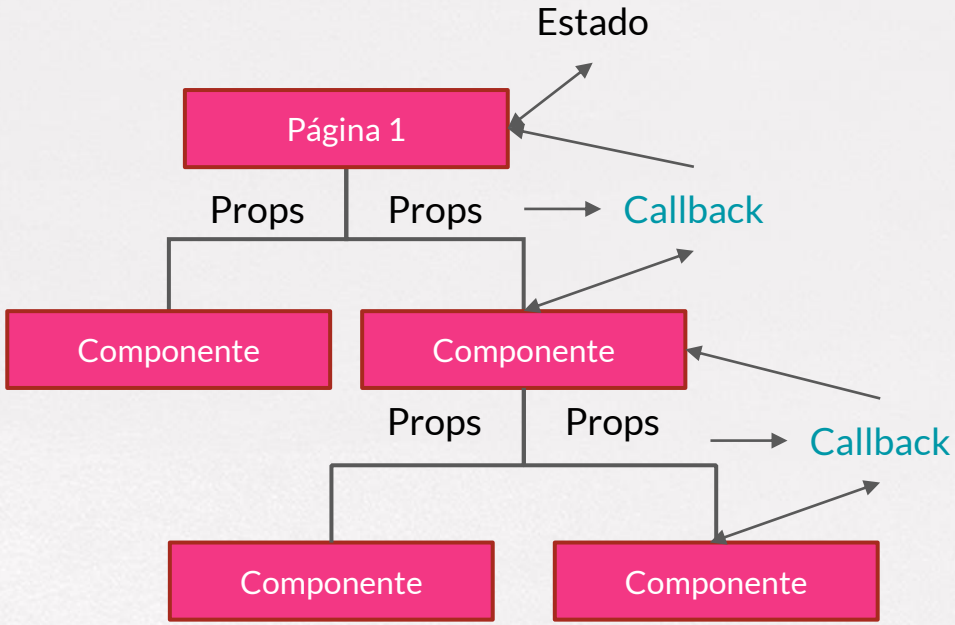
- Favorece la cascada
- Mantén los componentes puros
- Centraliza si es necesario

Favorece la cascada



- **Seguir la cascada de props** es una forma habitual de evitar que la aplicación tenga un estado descontrolado

Favorece la cascada



Favorece la cascada

```
const Pagina1 = () => {
  const [count, setCount] = useState(0);
  return (
    <>
      <div>
        <Componente callback={setCount}></Componente>
      </div>
    </>
  );
};

export default Pagina1;
```

Favorece la cascada

```
const Componente = (props) => {
  return (
    <div onclick={props.callback(count => count + 1)}></div>
  )
}

export default Componente;
```

Mantén los componentes puros

- Un componente puro es **aquel cuya renderización sólo depende de las propiedades** y sólo cambia cuando sus propiedades cambian
- Se basa en e**l concepto de programación funcional de función pura**, es decir, aquella función cuyo resultado depende exclusivamente de sus parámetros
- La ventaja de los componentes puros es que son **puramente deterministas**, **no introducen efectos secundarios** y **su renderización es optimizable**

Mantén los componentes puros

```
class Ejemplo extends React.Component {
  constructor(props) {
    super(props);
  }

  render() {
    return {
      <div>
        <h1>Hola, me llamo {this.props.name}</h1>
      </div>
    }
  }
}

export default Ejemplo;
```

- El componente sólo necesita renderizarse si la propiedad **name** cambia
- Por tanto sería lógico implementar **shouldComponentUpdate** para que tenga esto en cuenta

Mantén los componentes puros

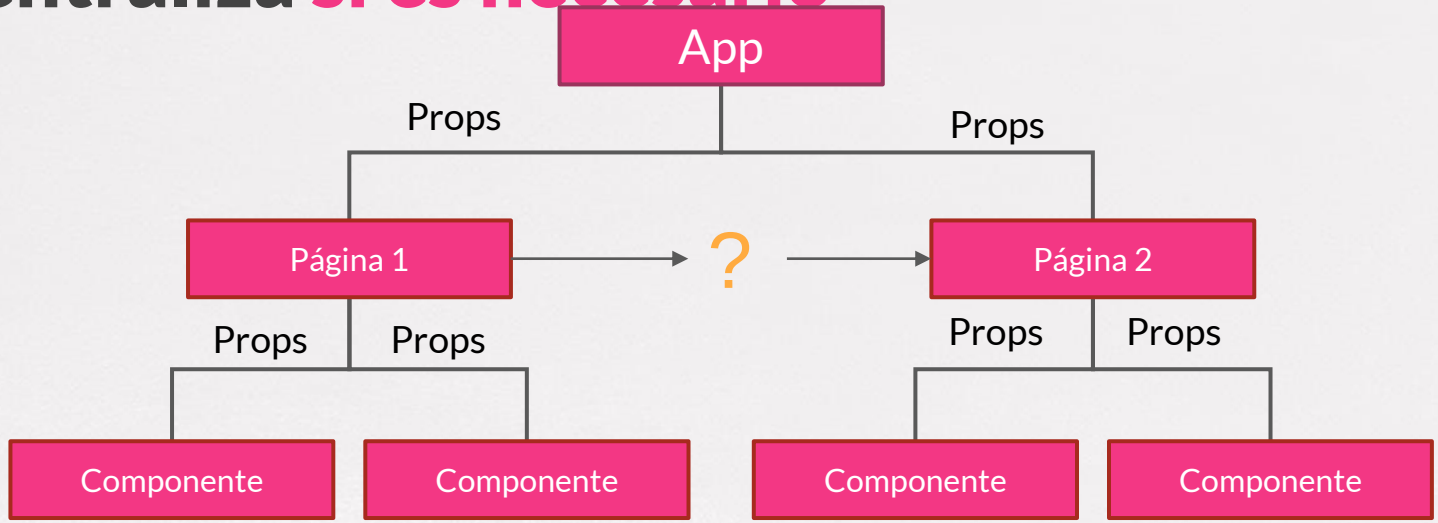
```
class Ejemplo extends React.PureComponent {
  constructor(props) {
    super(props);
  }

  render() {
    return {
      <div>
        <h1>Hola, me llamo {this.props.name}</h1>
      </div>
    }
  }
}

export default Ejemplo;
```

- Heredando de **PureComponent** tendremos una implementación por defecto para componentes puros
- En ella el método **shouldComponentUpdate** sólo devuelve **true** cuando una propiedad cambia, y **false** en caso contrario

Centraliza si es necesario



- La mayor parte de las veces sabremos que es necesaria alguna estrategia de centralización c**uando encontremos comunicación horizontal**

PARA RESUMIR

- Favorecer la cascada y mantener el mínimo posible de **componentes con estado o stateFull** simplifica la gestión del estado en puntos controlables
- Mantener los componentes puros, cuando sea posible, **no sólo favorece a la cascada sino que nos permite introducir optimizaciones de la librería**
- **Centralizar el estado puede ser positivo incluso aunque no se use un patrón concreto**, simplemente tener alguna estrategia simplificará el desarrollo