React

Tres en raya con React

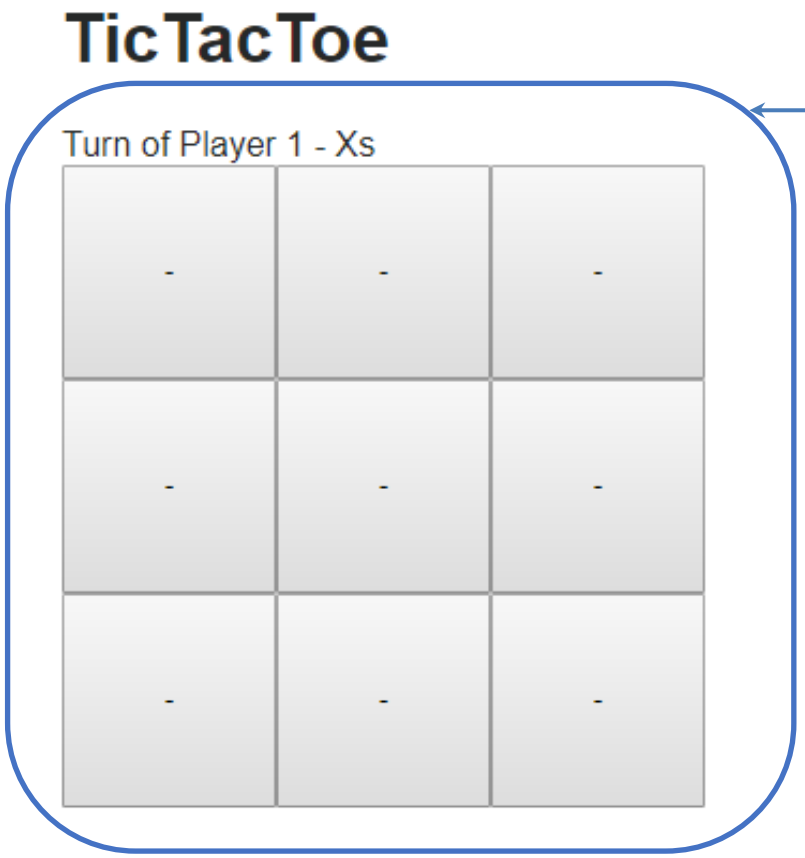
Enrique Barra Arias
Álvaro Alonso González

¿Qué vamos a hacer?

- Para profundizar en los conceptos básicos de React (estado, props, flujo de datos y de eventos, etc), vamos a implementar un juego de TicTacToe (o Tres en Raya)
- El primer paso antes de empezar a programar es separar la jerarquía de componentes de nuestra aplicación
- Seguiremos el "principio de responsabilidad única". Cada componente idealmente hace una cosa: si crece debe ser dividido en subcomponentes

¿Qué vamos a hacer?

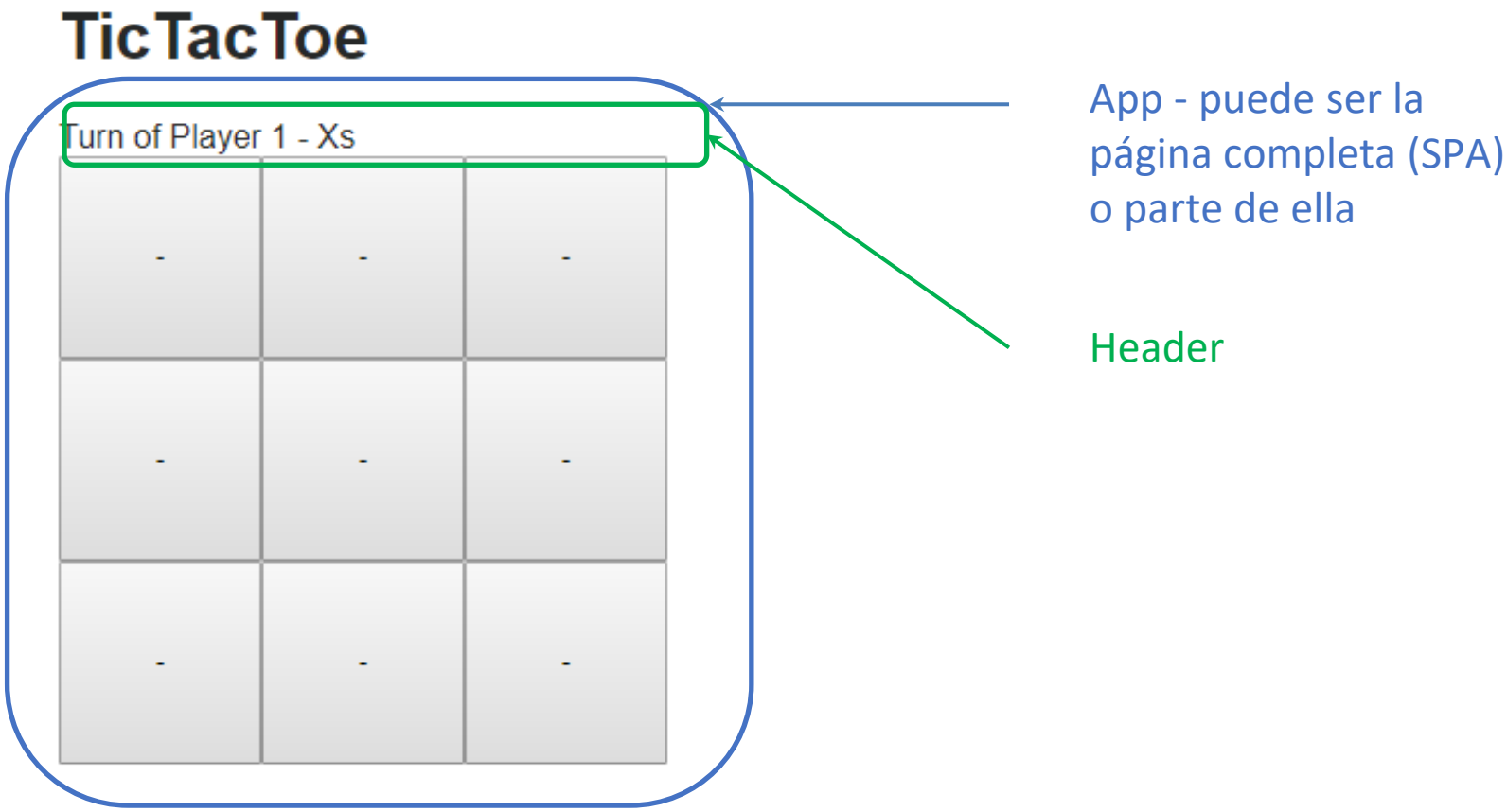
- Para aprender los conceptos básicos de React, vamos a implementar un juego de TicTacToe (o Tres en Raya)
- El primer paso antes de empezar a programar es separar la jerarquía de componentes de nuestra aplicación
- Seguiremos el "principio de responsabilidad única". Cada componente idealmente hace una cosa: si crece debe ser dividido en subcomponentes



App - puede ser la página completa (SPA) o parte de ella

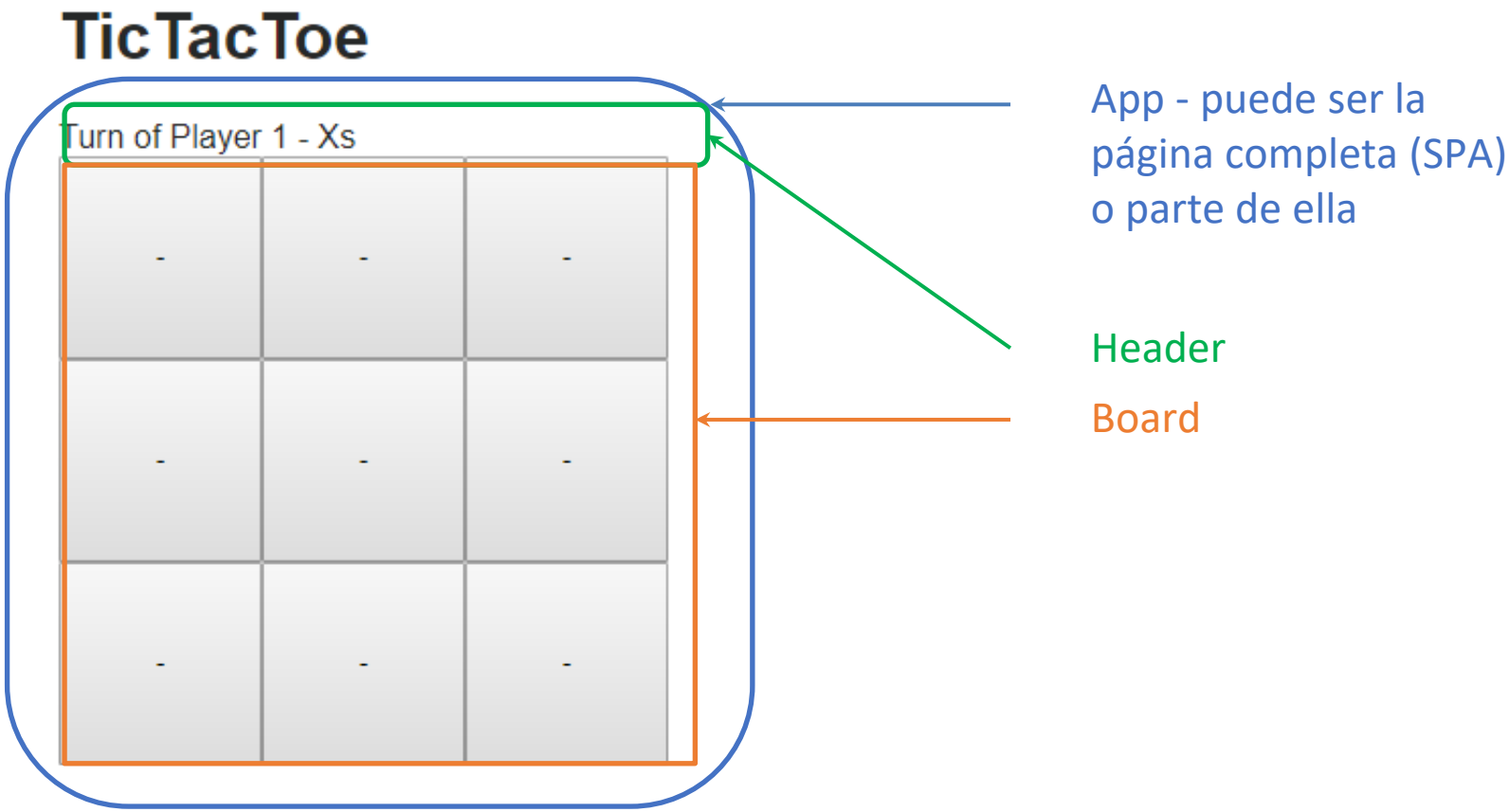
¿Qué vamos a hacer?

- Para aprender los conceptos básicos de React, vamos a implementar un juego de TicTacToe (o Tres en Raya)
- El primer paso antes de empezar a programar es separar la jerarquía de componentes de nuestra aplicación
- Seguiremos el "principio de responsabilidad única". Cada componente idealmente hace una cosa: si crece debe ser dividido en subcomponentes



¿Qué vamos a hacer?

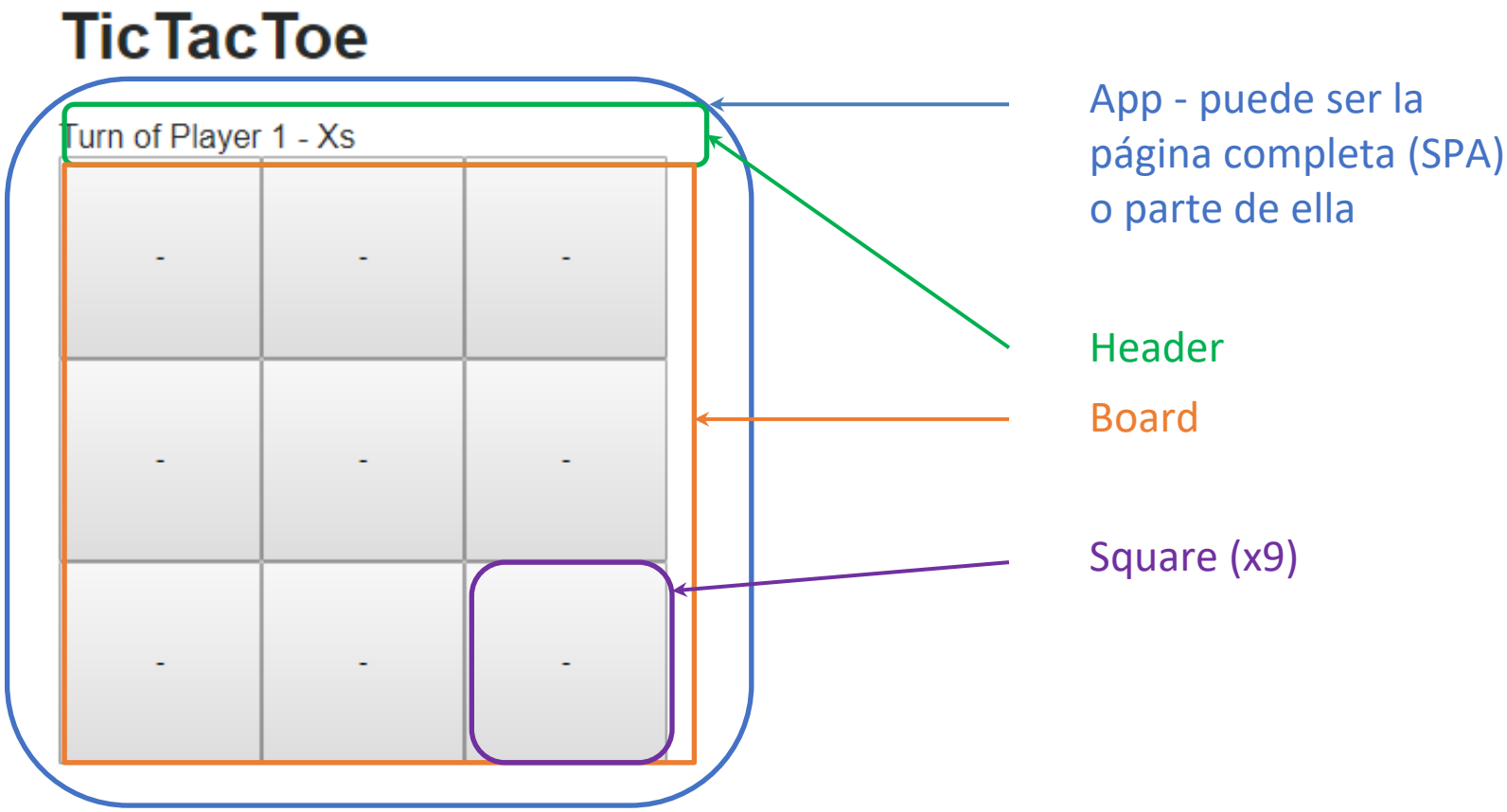
- Para aprender los conceptos básicos de React, vamos a implementar un juego de TicTacToe (o Tres en Raya)
- El primer paso antes de empezar a programar es separar la jerarquía de componentes de nuestra aplicación
- Seguiremos el "principio de responsabilidad única". Cada componente idealmente hace una cosa: si crece debe ser dividido en subcomponentes



5

¿Qué vamos a hacer?

- Para aprender los conceptos básicos de React, vamos a implementar un juego de TicTacToe (o Tres en Raya)
- El primer paso antes de empezar a programar es separar la jerarquía de componentes de nuestra aplicación
- Seguiremos el "principio de responsabilidad única". Cada componente idealmente hace una cosa: si crece debe ser dividido en subcomponentes



Proyecto en GitHub

Proyecto completo en:

https://github.com/REACT-UPM/ejemplo_tres_en_raya

Paso – Preparar el entorno

- Software necesario
  - node (16 o 18)
  - create-react-app

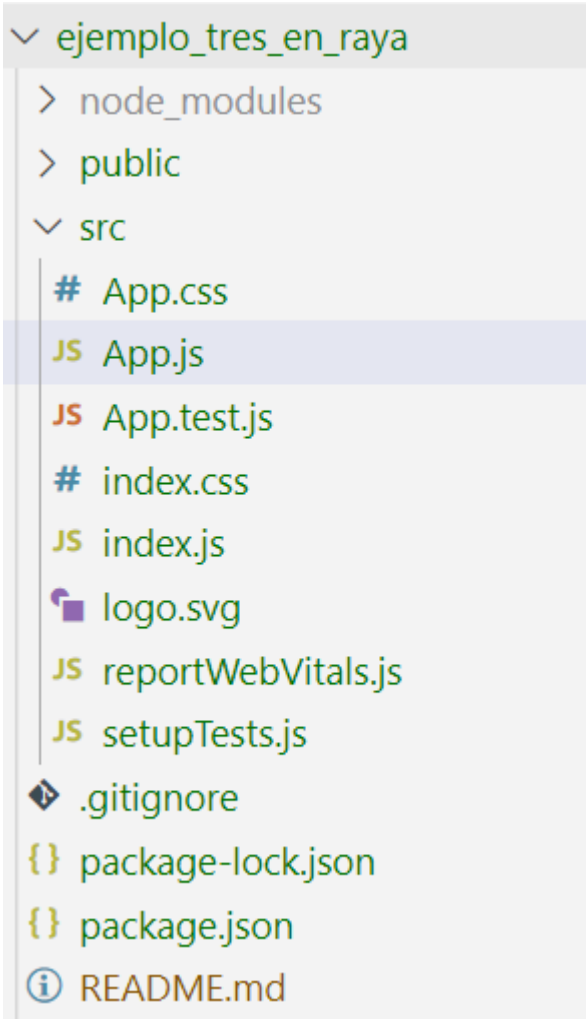
- **Crear el scaffold con create-react-app**
  - npx create-react-app tictactoe
  - cd tictactoe
  - npm install
  - npm start

- En este momento deberías poder
  - ver el proyecto inicial en el navegador

Paso – Preparar el entorno

- Ya tenemos las carpetas del proyecto

- Recuerda que si cambiamos el texto “Edit src/App.js and save to reload.” en src/App.js se debería recargar automáticamente la página



Paso – Definir el estado

- Vamos a definir **el estado** del Tictactoe
- Tenemos que preguntarnos "¿Qué **datos** necesitaríamos para volver a pintar la aplicación en otro dispositivo?" (por supuesto teniendo el html, css y js.
- Nota: En el futuro el estado crecerá según se vayan añadiendo funcionalidades

Paso – Definir el estado

- Vamos a definir **el estado** del Tictactoe
- Tenemos que preguntarnos "¿Qué **datos** necesitaríamos para volver a pintar la aplicación en otro dispositivo?" (por supuesto teniendo el html, css y js.
- Nota: En el futuro el estado crecerá según se vayan añadiendo funcionalidades


```
{
    turn: PLAYERX,
    values: [
        ['-','-','-'],
        ['-','-','-'],
        ['-','-','-']
    ]
}
```

Tipos de componentes

- **Componentes de presentación** (skinny, dumb)
  - Se preocupan por cómo se ven las cosas
  - Usualmente tienen etiquetas HTML y estilos propios...
  - No tienen ninguna dependencia del resto de la aplicación, como acciones o tiendas
  - Reciben datos y llamadas exclusivamente a través de las props
  - Rara vez tienen su propio estado (cuando lo tienen, es el estado de UI más que los datos)
- **Componentes contenedores** (fat, smart)
  - Se preocupan por cómo funcionan las cosas
  - Proporcionan los datos y el comportamiento a la presentación u otros componentes del contenedor
  - Llaman a las acciones y las proporcionan como callbacks a los componentes de presentación
  - Suelen tener estado propio, ya que tienden a servir como fuentes de datos
- Más info: https://medium.com/@dan_abramov/smart-and-dumb-components-7ca2f9a7c7d0

Paso – Componentes Header y Board

- Para evitar acumular todo el código en el componente **App**. crearemos subcomponentes o componentes hijos

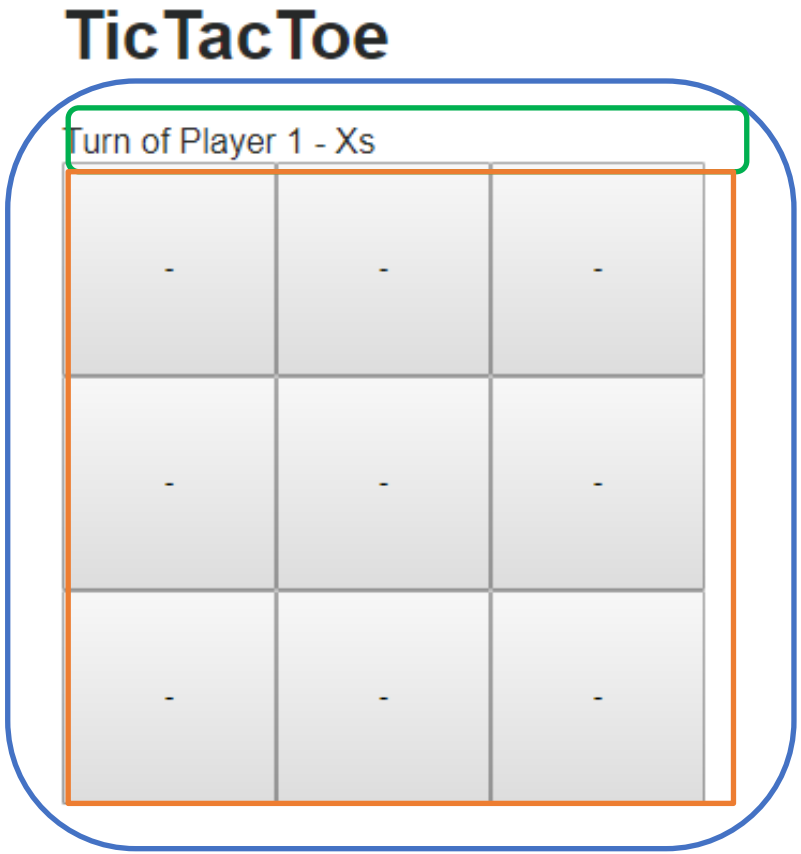
- **App** será nuestro componente contenedor y el resto serán componentes de presentación

- En este paso vamos a crear el componente **Board** que dibuja el tablero y el componente **Header** que dibuja la cabecera de texto sobre el tablero con información sobre el turno

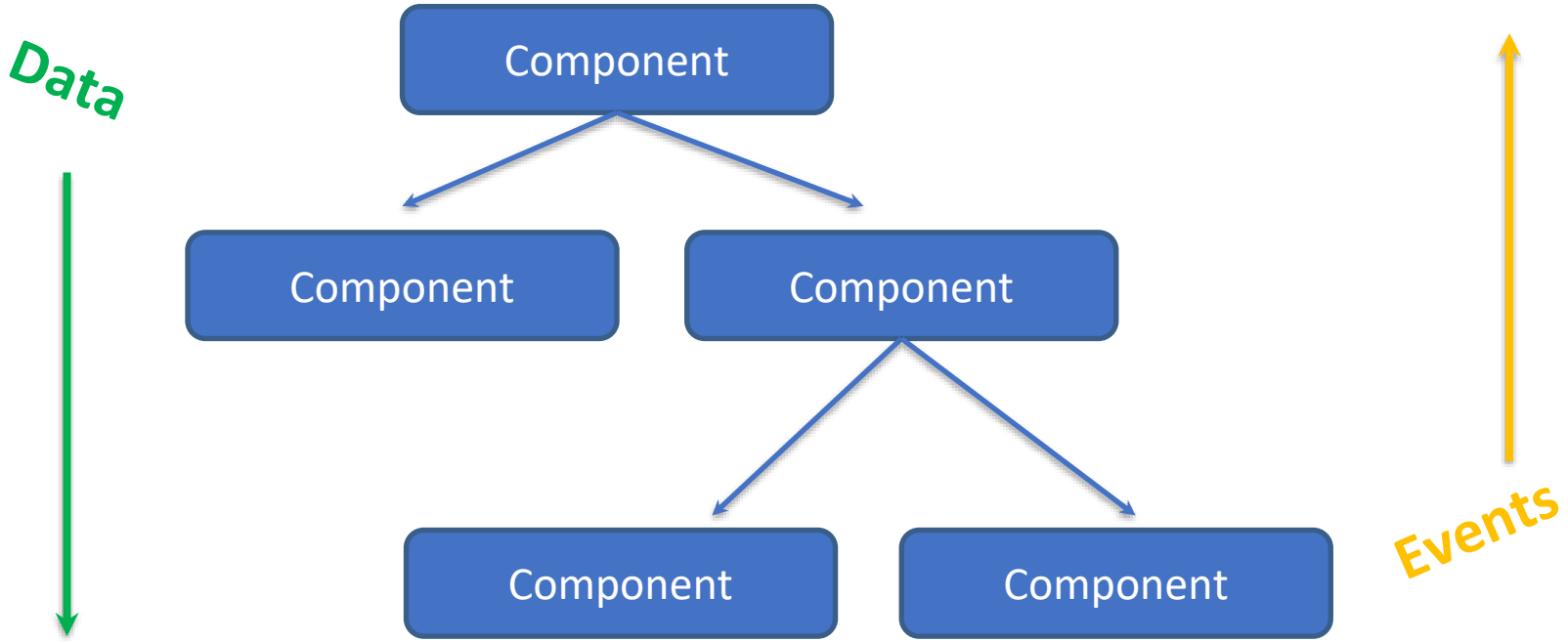
- Reciben como "props" lo que necesitan:
  - Partes del estado
  - Y después funciones para llamar cuando sea necesario (cuando ocurra un evento en ellas)

Paso – Componentes Header y Board

- **Header**: pondrá el mensaje (“Turn of player 1 – Xs” o “Turn of player 2 – Os”)
- **Board**: manejará las casillas (Square)



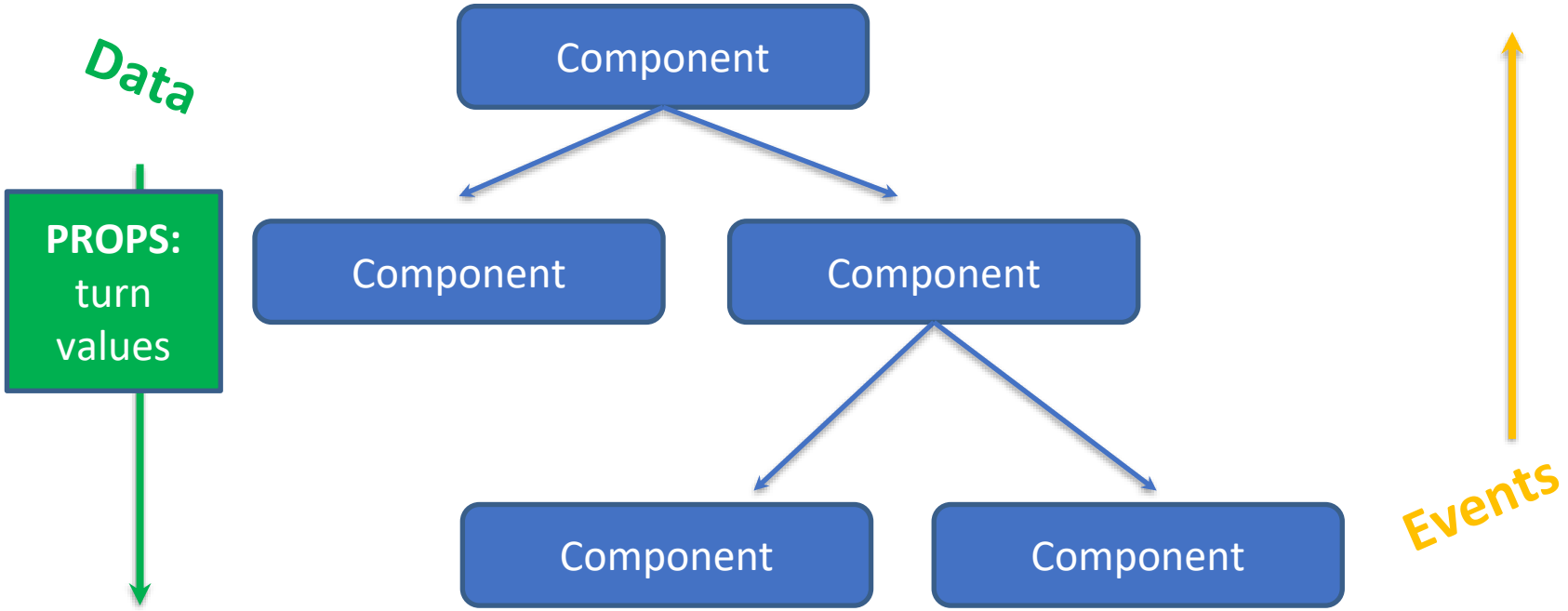
- Recuerda:


Data
Component
Component
Component
Component
Component
Events

Paso – Componentes Header y Board

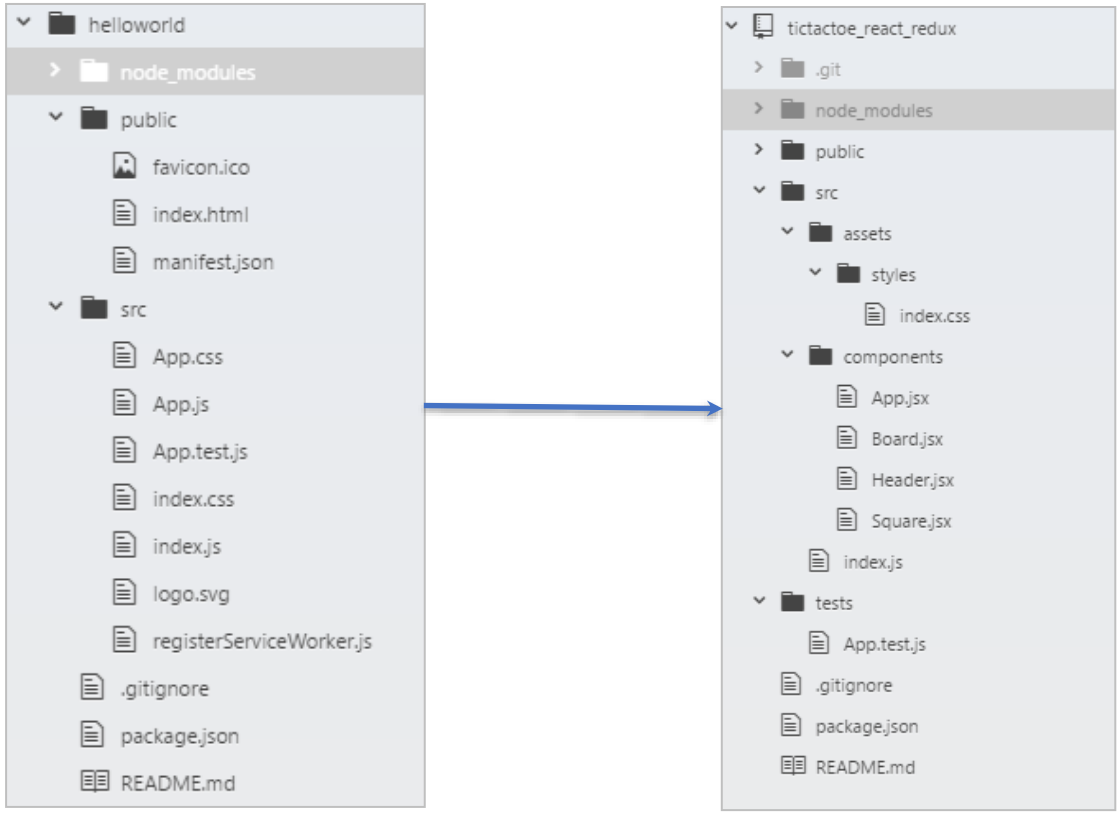
Flujo de eventos vs. Flujo de datos en React

- Recuerda:



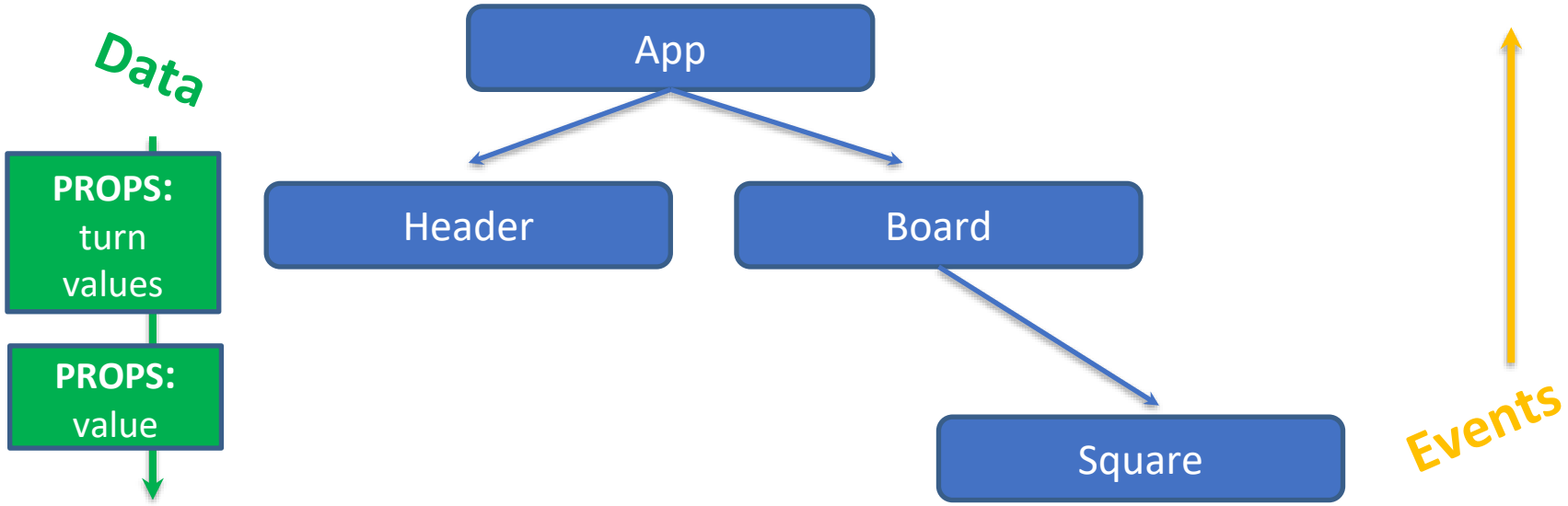
Paso – Reestructurar el código

- Opcional aunque recomendable si la aplicación crece
- Podemos refactorizar el código como queramos siempre y cuando mantengamos la carpeta **src** y el archivo i**ndex.js** dentro de ella
- Cuando movemos los ficheros a una nueva carpeta tenemos que editar los imports de esos ficheros, así como los de los ficheros que hacen referencia a ellos.



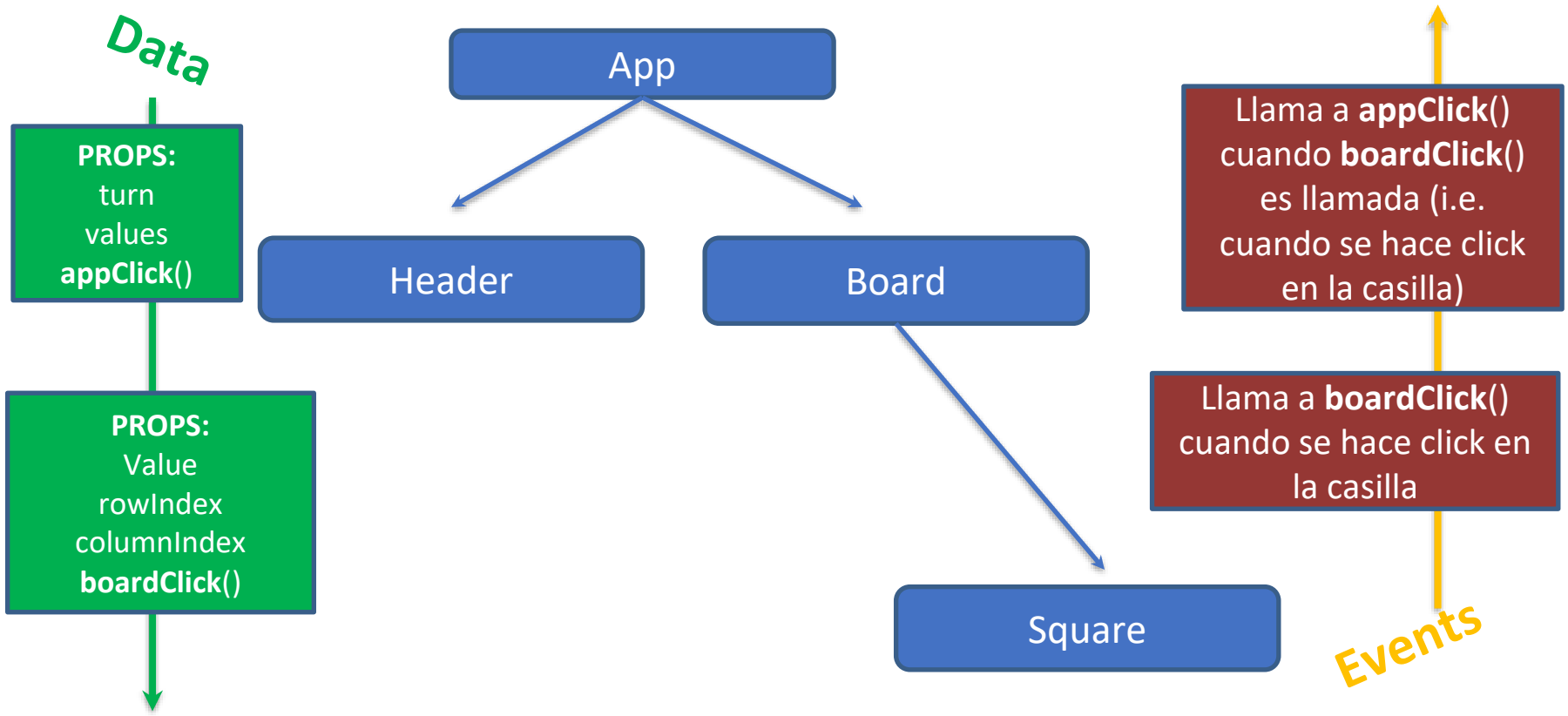
Paso – Flujo de eventos
Lo que tenemos hasta ahora

- Vamos a pasar como un apoyo a la función que el componente hijo tiene que llamar (por lo que es un componente de presentación y no contiene ninguna lógica)
- El estado se modificará en el componente superior que es donde se “aloja”



- Creamos una función **appClick()** en el componente App.jsx, que será llamada cada vez que hagamos click en una casilla
- Tendrá que hacer lo siguiente:
  - Cambiar el estado para reflejar la nueva situación
  - Advertencia: no cambiar el estado directamente ->
    Usar el setter que devuelve useState
  - ¿Qué debemos cambiar en el estado para reflejar el click? ->
    - Turno
    - Valor de la posición (x,y) de la casilla clickada

Paso – Flujo de eventos



Paso – Más acciones - Contador de movimientos

- Añadimos “moves” al estado
- En el método render de App añadimos el contador (con un h3 o un componente nuevo)
- A continuación tenemos que actualizar el valor de “moves” cada vez que hay un nuevo click. Esto lo hacemos en **appClick** (el único lugar donde modificamos el estado)

Paso – Más acciones - Botón de reset

- Queremos un botón de “reset” para reiniciar el juego
- Este botón sólo tendrá que volver a establecer el estado inicial
- Creamos un método resetClick en App.jsx que reinicie el juego
- En el return añadimos un nuevo componente Reset (recuerda también importarlo con import) y le pasamos este nuevo método de **resetClick** para que lo llame (será una prop)

Paso – Más acciones - Cargar datos de partida guardada al cargar juego

- Queremos hacer fetch a https://api.npoint.io/c734e05e43c5b87dd971 para cargar el estado que nos da el servidor y no empezar con la partida en blanco

- Hacemos un hook useEffect con fetchData

Paso – React Bootstrap
¿Qué es Bootstrap? - Instalación

- Bootstrap es un framework de desarrollo web
  - Provee CSS, HTML y JS. Ejemplos: modals, buttons, tabs, breadcrumbs…
- React-bootstrap es lo mismo pero reescrito para React
  - https://react-bootstrap.github.io/
  - https://react-bootstrap.github.io/getting-started/introduction
- Se instala mediante:
  - npm install react-bootstrap bootstrap
- Hay que añadir el CSS de Bootstrap y opcionalmente un theme
- Dos opciones para esto
- En el componente “App.js”:
  - `import 'bootstrap/dist/css/bootstrap.min.css';`
- En el fichero “public/index.html”:

```
<linkrel="stylesheet"href="https://cdn.jsdelivr.net/npm/bootstrap@5.2.0-
beta1/dist/css/bootstrap.min.css"integrity="sha384-
0evHe/X+R7YkIZDRvuzKMRqM+OrBnVFBL6DOitfPri4tjfHxaWutUpFmBp4vmVor"crossorigin="anonymous"/>
```

- En los componentes en los que vayamos a usar React-bootstrap tenemos que importar los componentes de la librería que vamos a utilizar y luego declararlos en el return
- Es importante leer la documentación y mirar qué props esperan
  - https://react-bootstrap.github.io/components/alerts/

Esto está implementado en la rama bootstrap del proyecto en github:

https://github.com/IWEB-UPM/ejemplo_tres_en_raya/tree/bootstrap

Paso – Router

¿Cómo hacer una página principal de juegos con rutas a cada juego?

Esto está implementado en la rama router del proyecto en github:

https://github.com/IWEB-UPM/ejemplo_tres_en_raya/tree/router

- Pasos seguidos para meter rutas
- App.js pasa a ser TicTacToe.js (nuestra app antigua ahora es un juevo en una pestaña pero habrá otros juegos…)
- Index.js añade “BrowserRouter” por fuera de App.js que nos lo da react-router-dom
- App.js es totalmente nuevo. Tiene tres links uno a cada ruta específica y tres routes que cada una carga un componente
- Dos componentes Juego2.js y Home.js