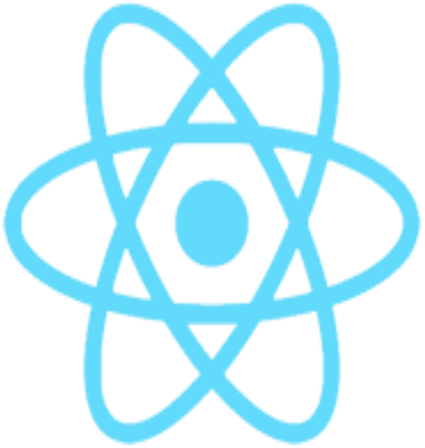


React

Manos a la obra

Enrique Barra

Indice

Nuestro primer componente

- Se crea definiendo una clase
- Tiene varios métodos: único obligatorio el método render
- Tiene unas propiedades (props) y estado (state)
- Se insertan en el DOM con:
  - ReactDOM.render(component, domElement)

```
class HelloWorld extends React.Component {
    render() {
        return (
            <p>
                Hello, <input type="text" placeholder="Your name here" />!
                It is {this.props.date.toTimeString() }
            </p>
        );
    }
});

setInterval(function () {
    ReactDOM.render(<HelloWorld date={new Date() }/>, document.getElementById('example‘));
}, 500);
```

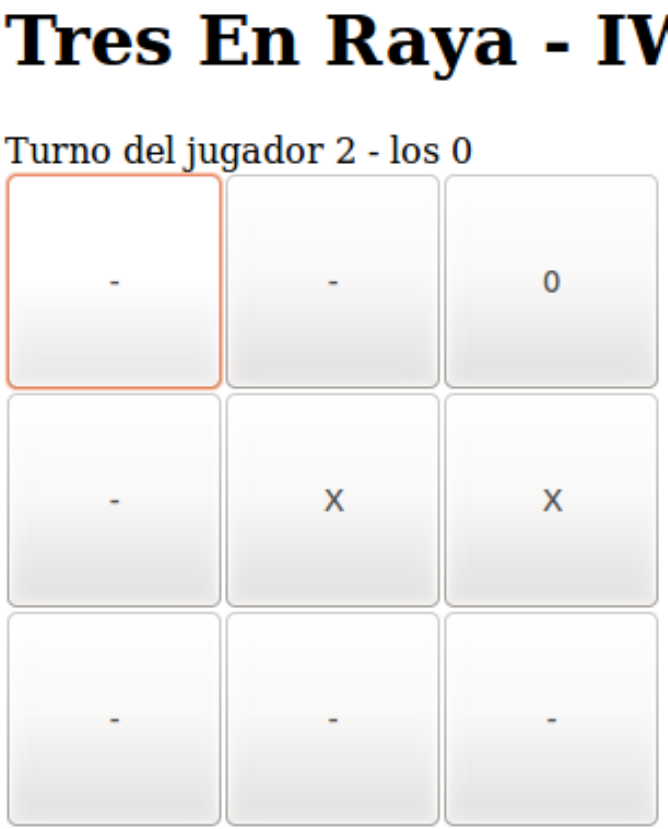
Props y state

- Los componentes tienen unas propiedades y un estado
- Propiedades (props) -> inmutables
- Estado (state) -> mutable
- Las propiedades las pasa el componente padre con un atributo
- El estado cambia con el método setState()

```
class LikeButton extends React.Component {
    getInitialState() {
        return { liked: false };
    }

    handleClick(event) {
        this.setState({ liked: !this.state.liked });
    }

    render() {
        var text = this.state.liked ? 'like' : 'haven\'t liked';
        return (
            <p onClick={this.handleClick}>You {text} this.Click to toggle.</p>
        );
    }
};

ReactDOM.render(<LikeButton />, document.getElementById('example'));
```

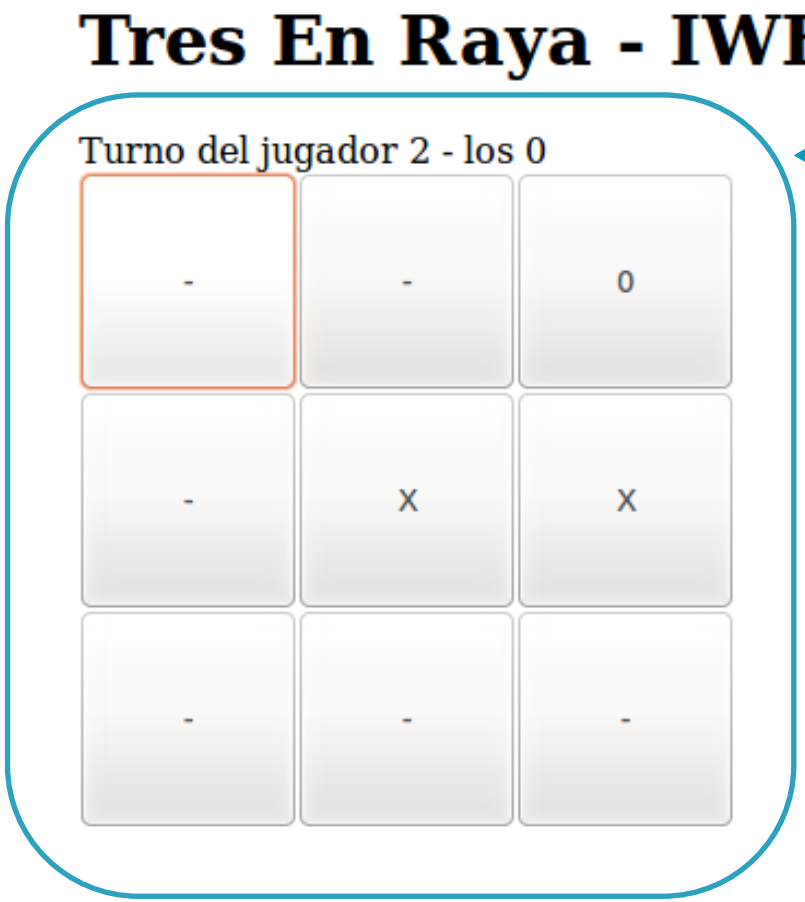
Tres en Raya

- Para aprender los conceptos básicos de React vamos a hacer un tres en raya
- El primer paso es separar la jerarquía de los componentes de nuestra app
- Seguiremos el “principio de responsabilidad única”. Cada componente idealmente hace una cosa, si crece se debería descomponer en subcomponentes



Tres en Raya

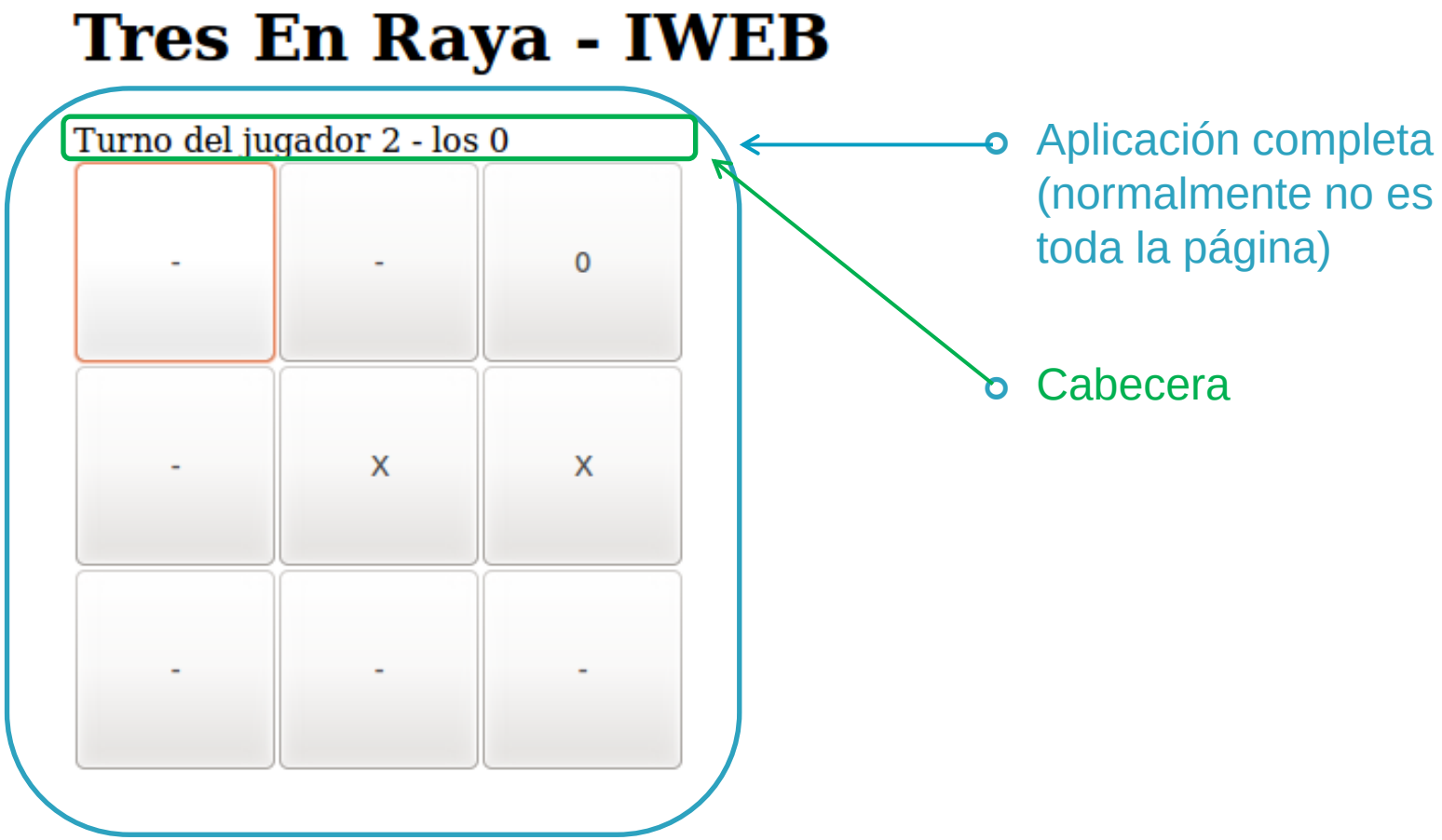
- Para aprender los conceptos básicos de React vamos a hacer un tres en raya
- El primer paso es separar la jerarquía de los componentes de nuestra app
- Seguiremos el “principio de responsabilidad única”. Cada componente idealmente hace una cosa, si crece se debería descomponer en subcomponentes



Aplicación completa (normalmente no es toda la página)

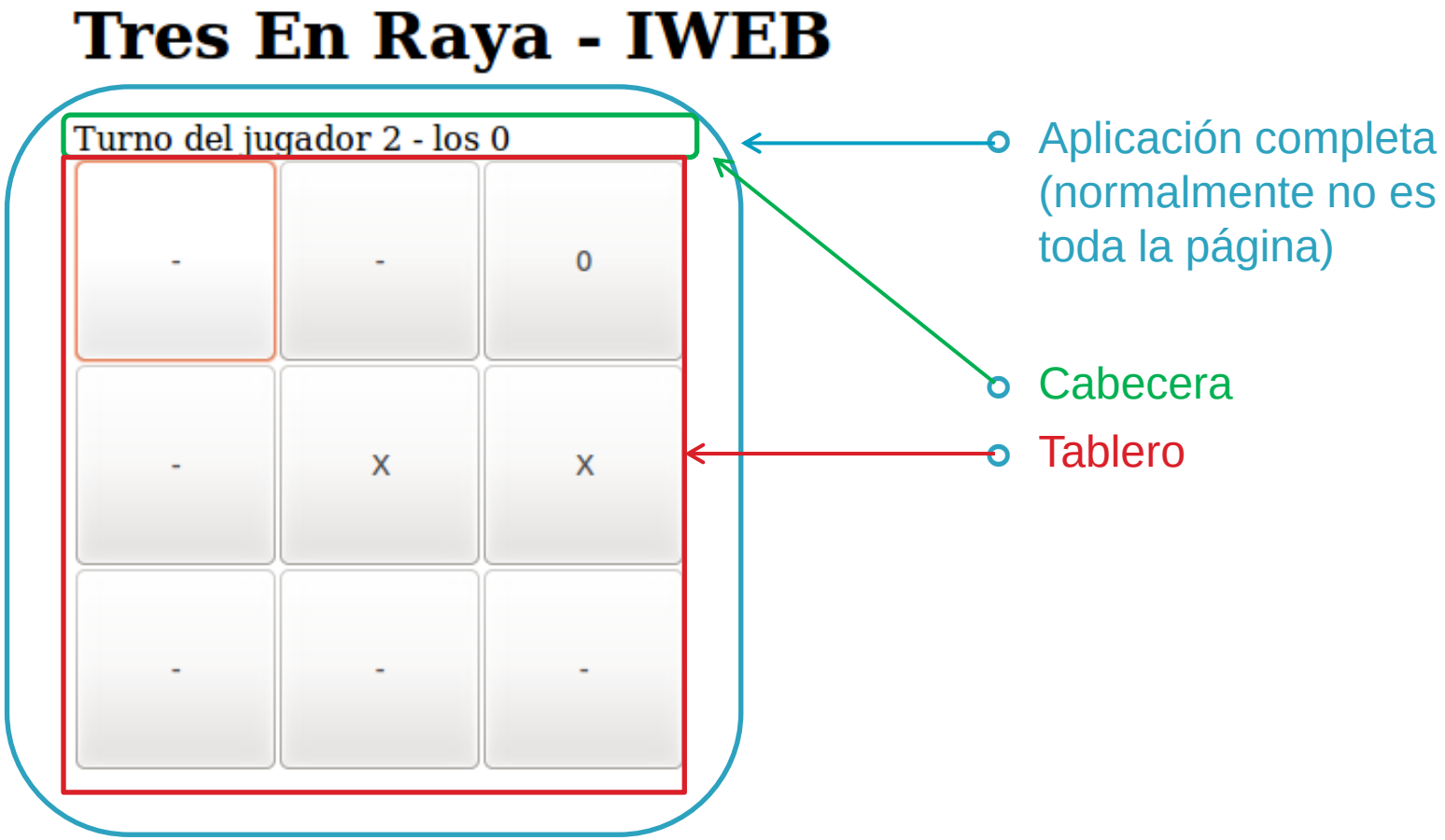
Tres en Raya

- Para aprender los conceptos básicos de React vamos a hacer un tres en raya
- El primer paso es separar la jerarquía de los componentes de nuestra app
- Seguiremos el “principio de responsabilidad única”. Cada componente idealmente hace una cosa, si crece se debería descomponer en subcomponentes



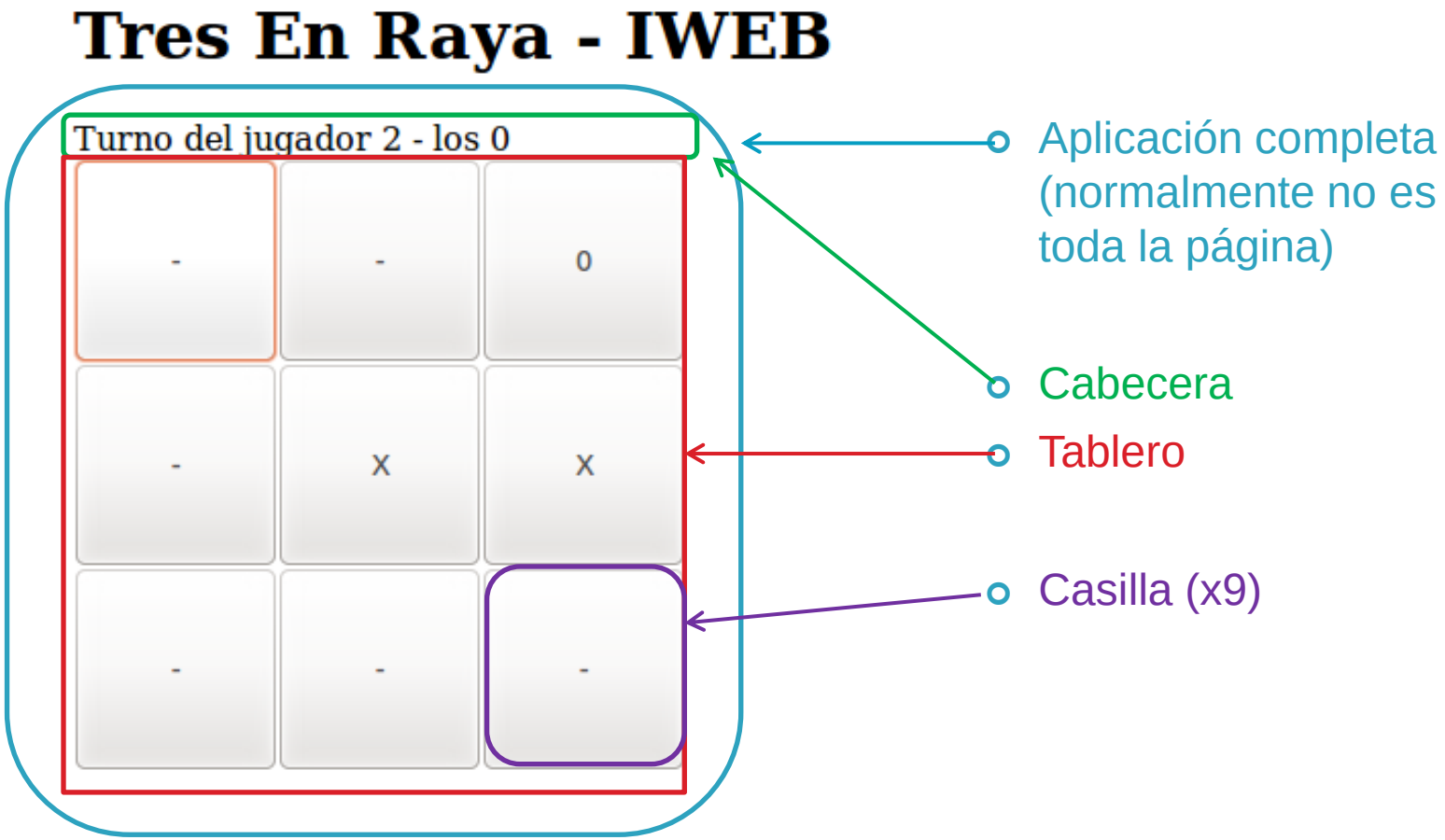
Tres en Raya

- Para aprender los conceptos básicos de React vamos a hacer un tres en raya
- El primer paso es separar la jerarquía de los componentes de nuestra app
- Seguiremos el “principio de responsabilidad única”. Cada componente idealmente hace una cosa, si crece se debería descomponer en subcomponentes



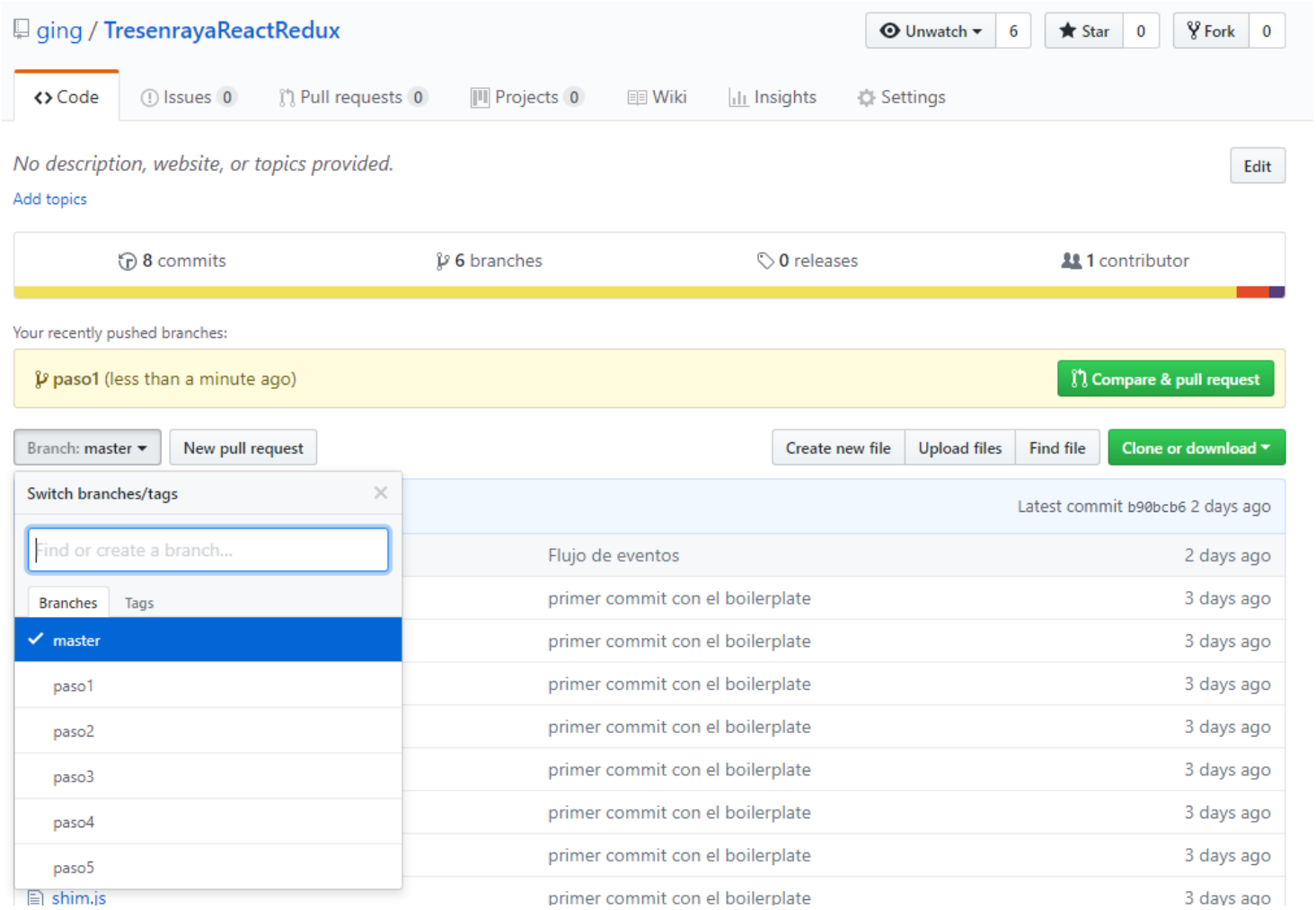
Tres en Raya

- Para aprender los conceptos básicos de React vamos a hacer un tres en raya
- El primer paso es separar la jerarquía de los componentes de nuestra app
- Seguiremos el “principio de responsabilidad única”. Cada componente idealmente hace una cosa, si crece se debería descomponer en subcomponentes



Proyecto en GitHub

- Proyecto completo en: https://github.com/ging/TresenrayaReactRedux



Paso 0 – preparar el entorno/proyecto

- Software necesario:
  - node (> 8.0.X)
  - npm (> 4.0.X y <5.0) (IMPORTANTE: no usar la versión 5 de npm porque no funciona React Native)
  - Si no tenemos node y npm, lo instalamos con las versiones que se indican.
  - Si ya lo tenemos lo actualizamos (yo tengo node 8.9.1 y npm 4.6.1 en windows 10)

- Creamos la carpeta para el proyecto:
  - `mkdir tresenraya`
  - `cd tresenraya`

- Descargar como zip el boilerplate y descomprimirlo en la carpeta tresenraya
  - https://github.com/sonsoleslp/react-iweb-boilerplate
  - Alternativamente se puede descargar con git y borrar el directorio .git (oculto) para desvincularlo del boilerplate original

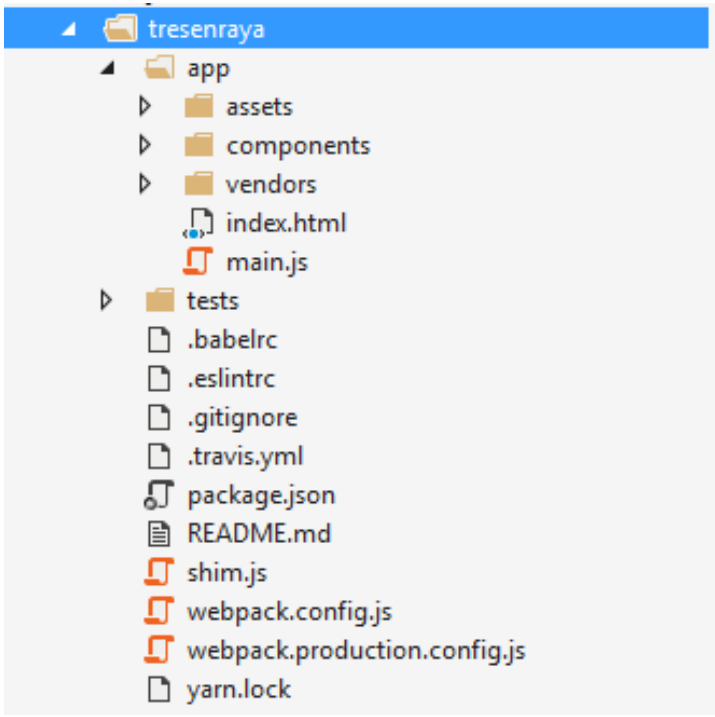
Paso 0 – preparar el entorno/proyecto

- Ya tenemos la estructura de directorios

- Abrimos un terminal (Shell). Nos movemos al directorio tresenraya y ejecutamos
  - `npm install`
  - Aparecerá un directorio node_modules

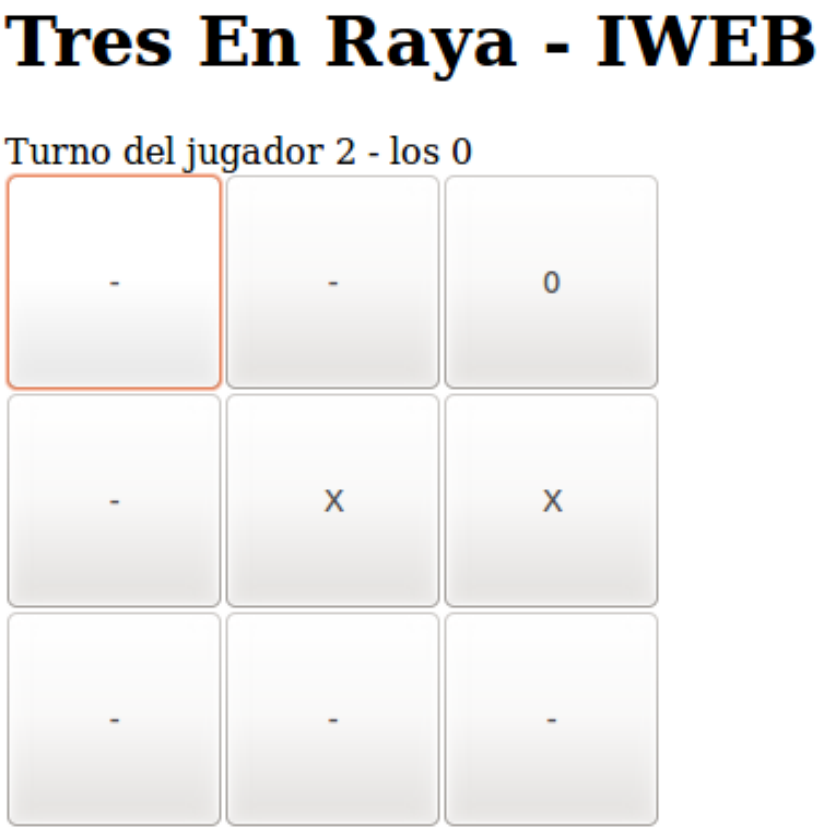
- Ejecutamos a continuación
  - `npm start`
  - Arrancará un servidor de desarrollo y abrirá la página en http://localhost:8080

- Inspeccionamos los distintos directorios y ficheros para ver qué tienen. En el fichero app/components/App.jsx cambiamos “Hello ReactJS” por un texto de nuestra elección, guardamos el fichero y visualizamos los cambios en el navegador.



Paso 1 - primer componente – App.jsx

Estado y constructor



Paso 1 - primer componente – App.jsx

Estado y constructor

- Vamos a definir el estado del tresenraya
- Aunque luego podrá ir creciendo
- Tendremos un turno para saber a quién le toca (X o O)
- Tendremos el estado del tablero que definimos como un array que contiene un array por cada fila
- El valor de cada casilla podrá ser “-”, “X” o “O”

```
this.state = {
    turno: JUGADORX,
    valores: [
    ['-', '-', '-'],
    ['-', '-', '-'],
    ['-', '-', '-'],
    ],
};
```

- El componente App.jsx tendrá un constructor, que solo llama a super(props) y fija el estado inicial

```
constructor(props) {
  super(props);
  this.state = {
      turno: JUGADORX,
      valores: [
      ['-', '-', '-'],
      ['-', '-', '-'],
      ['-', '-', '-'],
      ],
  };
}
```

Paso 1 - primer componente – App.jsx

Método Render de App.jsx

- Tan solo recorre el estado con un map dentro de un map (teníamos arrays dentro de arrays) para mostrar la información

```
render() {
    let texto = "Turno del " + this.state.turno;
    let tablero = this.state.valores.map((valoresFila, indiceFila) => {
      let fila = valoresFila.map((valor, indiceColumna) => {
          let mykey = "" + indiceFila + indiceColumna;
          return (
            <span key={mykey}>{valor}</span>
          );
      });
      return (
         <div key={"fila" + indiceFila}>{fila}</div>
      );
    });
    return (
      <div>
        <header className="cabecera">{texto}</header>
        {tablero}
      </div>
    );
}
```

Paso 2 – Componentes Cabecera.jsx y Tablero.jsx

- Para evitar acumular todo el código en el componente App.jsx creamos subcomponentes o componentes hijos

- Estos componentes serán “tontos”, solo saben pintar lo que se les pasa, no tienen estado

- Reciben como “props” la información que necesitan:
  - trozos del estado
  - funciones a las que llamar cuando ocurren acciones

- Haremos en este paso el componente Tablero.jsx que dibujará el tablero y el componente Cabecera.jsx que dibujará un texto encima del tablero con la información del turno

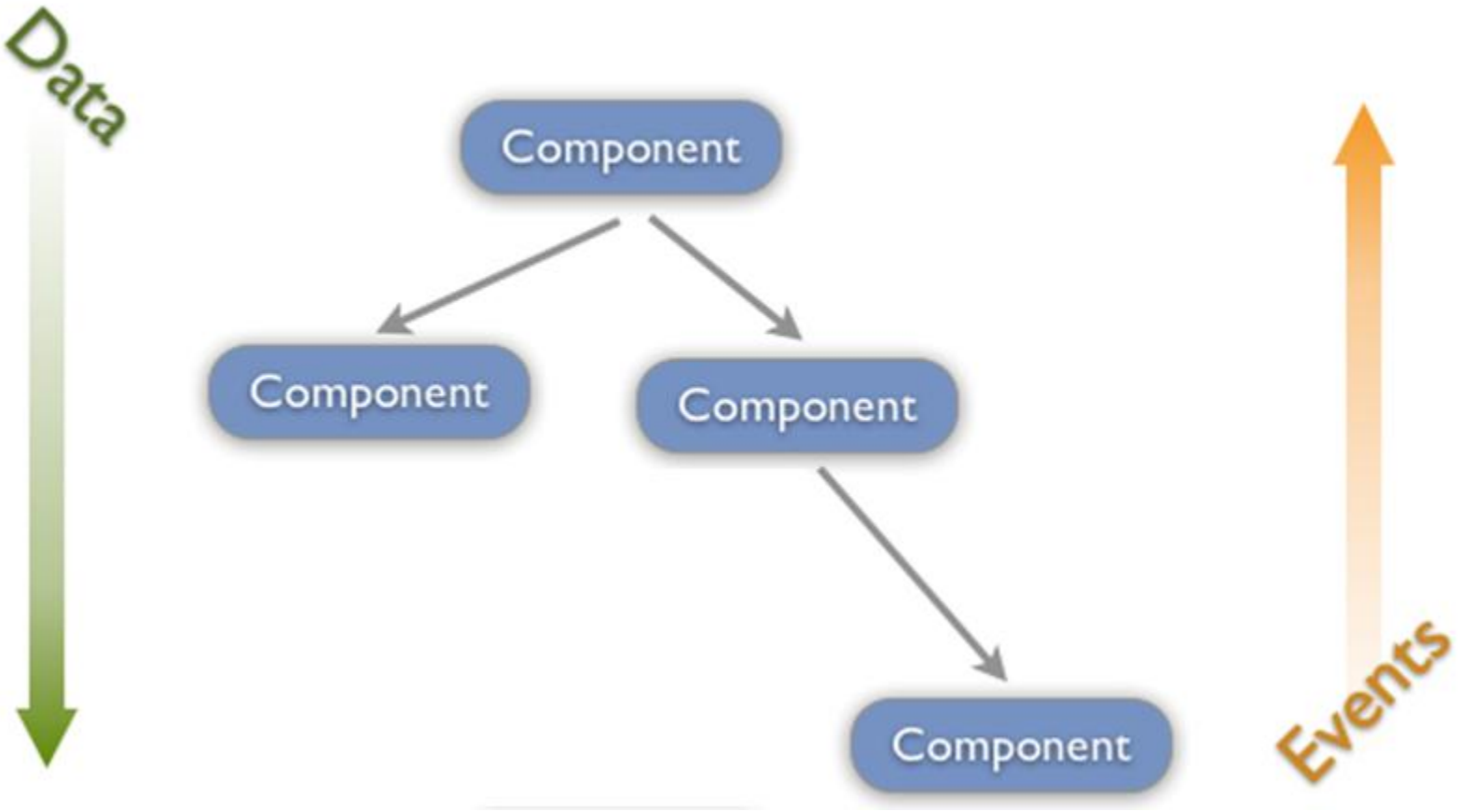
Paso 2 – Componentes Cabecera.jsx y Tablero.jsx

- "*cabecera*": pondrá el mensaje "Turno del jugador 1 - las X" o "Turno del jugador 2 - los 0"
- "*tablero*": gestionará las casillas

Paso 2 – Componentes Cabecera.jsx y Tablero.jsx

Flujo de datos y eventos en React

- Recordemos:

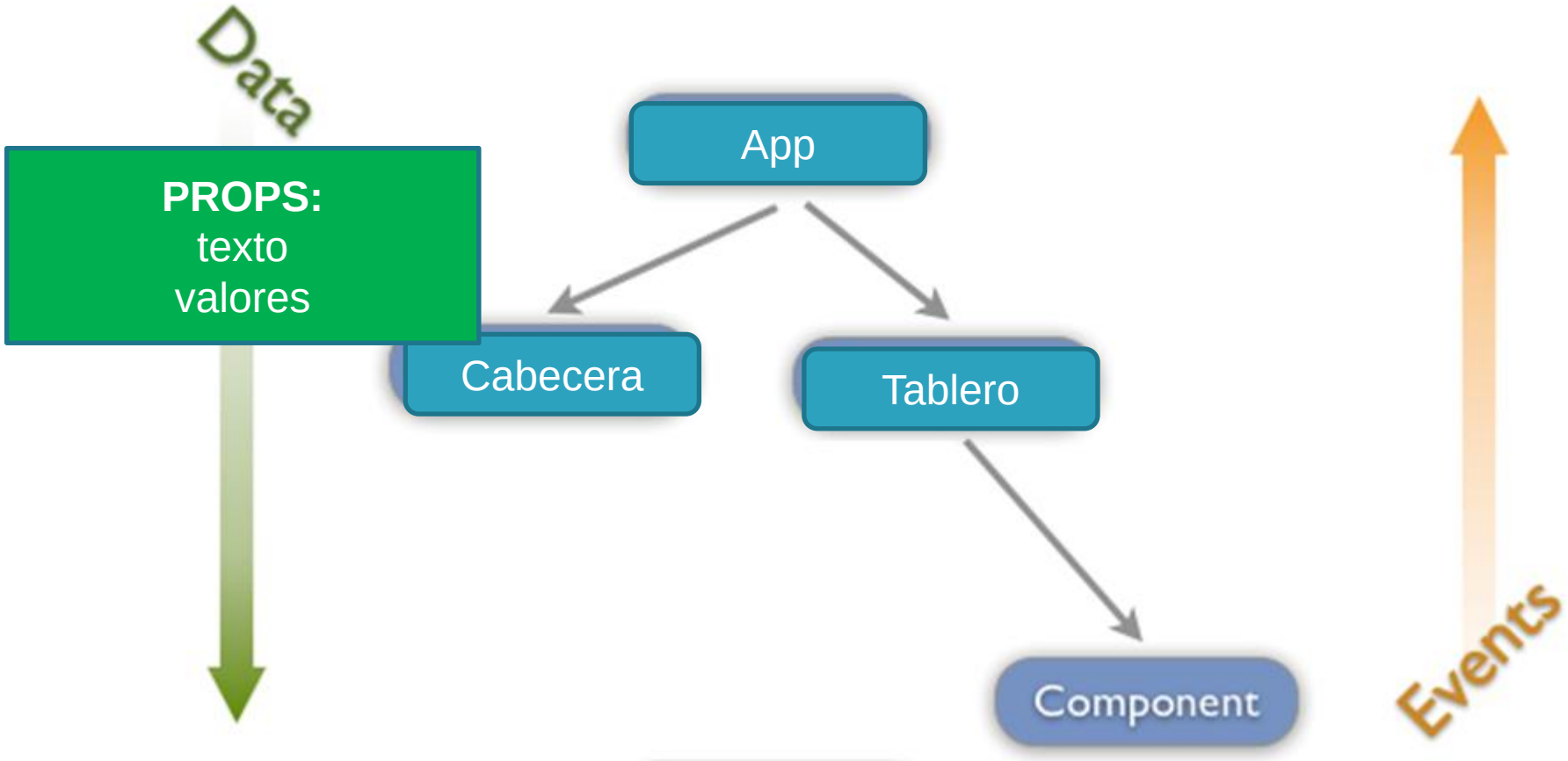


Fuente: https://facebook.github.io/react/img/blog/tutsplus.png

Paso 2 – Componentes Cabecera.jsx y Tablero.jsx

Flujo de datos y eventos en React

- Recordemos:



Fuente: https://facebook.github.io/react/img/blog/tutsplus.png

Paso 2 – Componentes Cabecera.jsx y Tablero.jsx

Cabecera.jsx

```
import React from 'react';

export default class Cabecera extends React.Component {
    render() {
        return (
          <header className="cabecera">
            {this.props.texto}
          </header>
        );
    }
}
```

Paso 2 – Componentes Cabecera.jsx y Tablero.jsx

Tablero.jsx

```
import React from 'react';

const casillaStyle = {
    height: '100px',
    width: '100px',
};
export default class Tablero extends React.Component {
    render() {
        let tablero = this.props.valores.map((valoresFila, indiceFila) => {
            let fila = valoresFila.map((valor, indiceColumna) => {
                let mykey = "" + indiceFila + indiceColumna;
                return (
                  <button style={casillaStyle} key={mykey}>{valor}</button>
                );
            });
            return (
               <div key={"fila" + indiceFila}>{fila}</div>
            );
        });

        return (
           <div>{tablero}</div>
        );
    }
}
```

Paso 2 – Componentes Cabecera.jsx y Tablero.jsx

App.jsx

```
import React from 'react';
import './../assets/scss/main.scss';
import Cabecera from './Cabecera.jsx';
import Tablero from './Tablero.jsx';

const JUGADORX = "jugador 1 - las X";
const JUGADOR0 = "jugador 2 - los 0";
export default class App extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
        turno: JUGADORX,
        valores: [
        ['-', '-', '-'],
        ['-', '-', '-'],
        ['-', '-', '-'],
        ],
    };
  }
  render() {
      let texto = "Turno del " + this.state.turno;
      return (
        <div>
          <Cabecera texto={texto}/>
          <Tablero valores={this.state.valores}/>
        </div>
      );
  }
}
```

Paso 3 – Componente Casilla.jsx

Casilla.jsx

```
import React from 'react';

const casillaStyle = {
    height: '100px',
    width: '100px',
};
export default class Casilla extends React.Component {
    render() {
        return(
          <button style={casillaStyle}>
            {this.props.valor}
          </button>
        );
    }
}
```

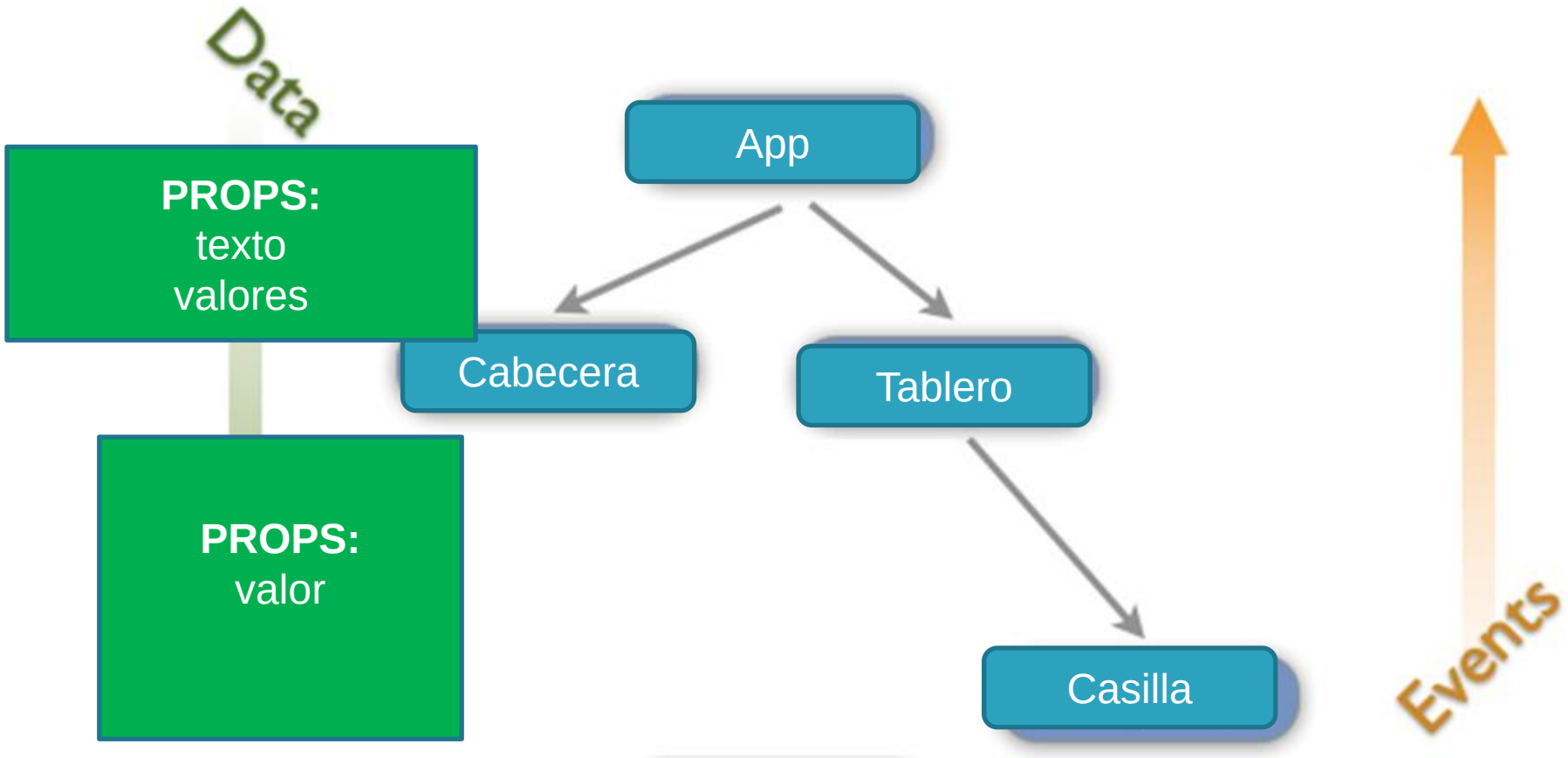
Paso 3 – Componente Casilla.jsx

Tablero.jsx

```
import React from 'react';
import Casilla from './Casilla.jsx';
const casillaStyle = {
    height: '100px',
    width: '100px',
};
export default class Tablero extends React.Component {
    render() {
        let tablero = this.props.valores.map((valoresFila, indiceFila) => {
            let fila = valoresFila.map((valor, indiceColumna) => {
                let mykey = "" + indiceFila + indiceColumna;
                return (
                  <Casilla valor={valor} key={mykey}/>
                );
            });
            return (
               <div key={"fila" + indiceFila}>{fila}</div>
            );
        });
        return (
           <div>{tablero}</div>
        );
    }
}
```

Paso 4 – Flujo de eventos

Lo que tenemos hasta ahora



- Vamos a pasar como props la función a la que tienen que llamar los componentes inferiores para que sean tontos
- El estado se moficará en el componente superior

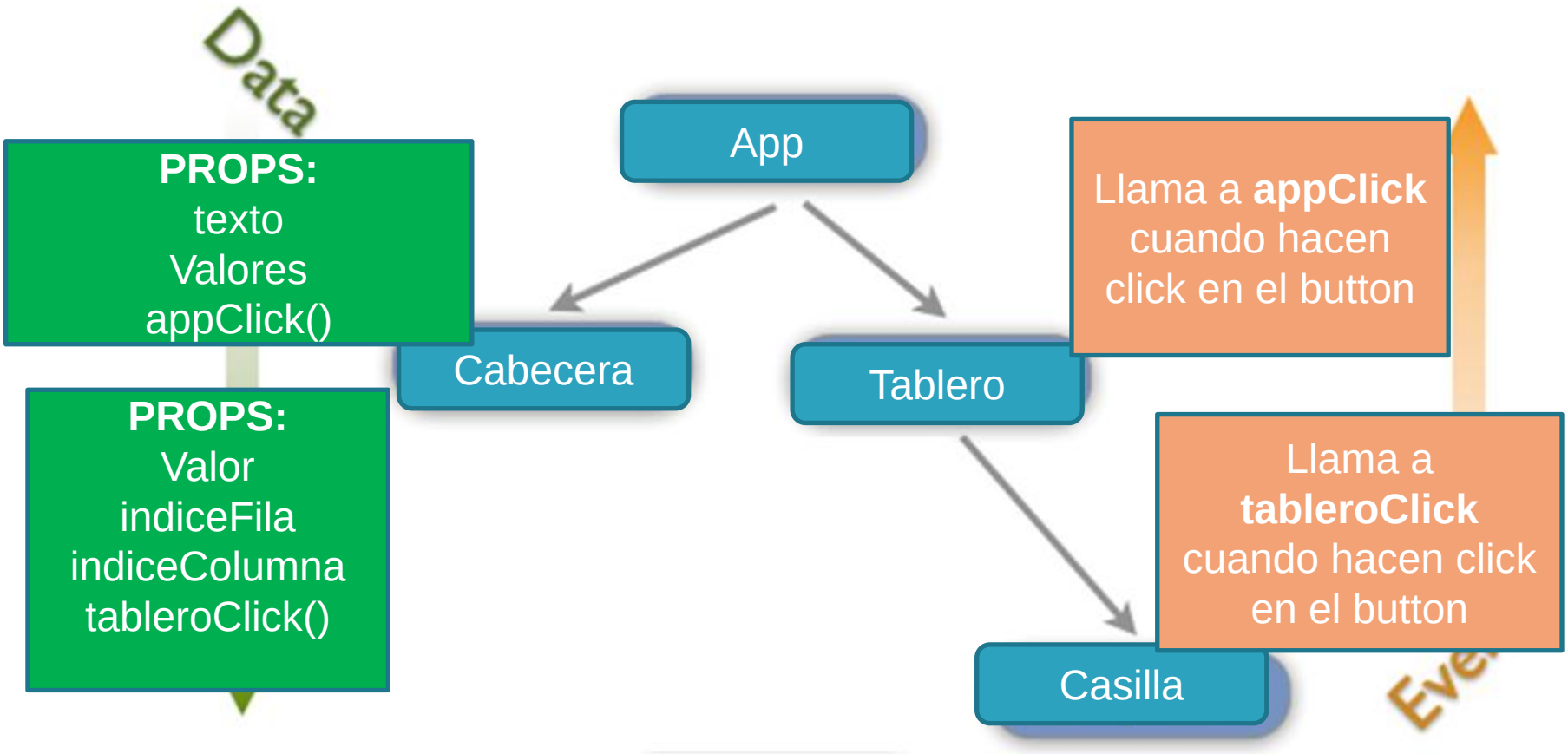
Paso 4 – Flujo de eventos

Función appClick de App.jsx

- Qué tiene que hacer la función appClick() que se llamará cada vez que se hace click en un botón
  - Cambiar el estado para que refleje la nueva situación
  - Precaución no mutar el estado (immutability)
  - ¿Qué datos necesita para cambiar el estado? ->
    - el turno (lo tiene en el propio estado)
    - la posición (x e y) de la casilla pulsada (habrá que pasársela a la casilla como prop)

Paso 4 – Flujo de eventos

Lo que vamos a añadir en este paso



Paso 4 – Flujo de eventos

Cambios App.jsx

- Método appClick que recibe numeroFila y numeroColumna:

```
appClick(numeroFila, numberoColumna) {
    let nuevosvalores = this.state.valores.slice();
    let nuevoValor = this.state.turno === JUGADORX ? 'X' : '0';
    nuevosvalores[numeroFila][numberoColumna] = nuevoValor;
    this.setState({
        turno: this.state.turno === JUGADORX ? JUGADOR0 : JUGADORX,
        valores: this.state.valores,
    });
}
```

- En el constructor hay que pasarle “this” con un bind para que pueda hacer this.state.

```
this.appClick = this.appClick.bind(this);
```

- Al Tablero hay que pasarle como prop el método appClick

```
return (
  <div>
    <Cabecera texto={texto}/>
    <Tablero valores={this.state.valores} appClick={this.appClick}/>
  </div>
);
```

Paso 4 – Flujo de eventos

Cambios Tablero.jsx

- Método tableroClick que recibe numeroFila y numeroColumna y llama a appClick

```
tableroClick(numeroFila, numeroColumna) {
    this.props.appClick(numeroFila, numeroColumna);
}
```

- En el constructor hay que pasarle “this” con un bind para que pueda hacer this.props.

```
this.tableroClick = this.tableroClick.bind(this);
```

- A la Casilla hay que pasarle como prop el método tableroClick y también indiceFila e indiceColumna

```
let fila = valoresFila.map((valor, indiceColumna) => {
    let mykey = "" + indiceFila + indiceColumna;
    return (
      <Casilla valor={valor} key={mykey}
        indiceFila={indiceFila} indiceColumna={indiceColumna}
        tableroClick={this.tableroClick}/>
    );
});
```

Paso 4 – Flujo de eventos

Cambios Casilla.jsx

- Método casillaClick que usa numeroFila y numeroColumna de las props para llamar a tableroClick

```
casillaClick() {
    if(this.props.valor === "-") {
        this.props.tableroClick(this.props.indiceFila, this.props.indiceColumna);
    }
}
```

- En el constructor hay que pasarle “this” con un bind para que pueda hacer this.props.

```
this.casillaClick = this.casillaClick.bind(this);
```

- Podemos añadir una clase al button (con className) que dependa de su valor. Y añadir esas clases al fichero scss /assets/

```
className={this.props.valor === "-" ? "clickable" : "no_clickable"}
```

```
.clickable {
  cursor: pointer;
}

.no_clickable {
  cursor: not-allowed;
}
```

Paso 5 – Mas acciones

Contador de movimientos

- Habrá que añadir el número de movimientos al estado

```
this.state = {
    turno: JUGADORX,
    valores: [
    ['-', '-', '-'],
    ['-', '-', '-'],
    ['-', '-', '-'],
    ],
    movimientos: 0
};
```

- Y pintarlos en el render (con un h1 o con un nuevo componente)

```
render() {
    let texto = "Turno del " + this.state.turno;
    return (
      <div>
        <Cabecera texto={texto}/>
        <Tablero valores={this.state.valores} appClick={this.appClick}/>
        <h3>Número de movimientos: {this.state.movimientos}</h3>
      </div>
    );
}
```

Paso 5 – Mas acciones

Contador de movimientos

- Por último en appClick (único sitio donde se modifica el estado) habrá que actualizar el valor de “movimientos”

```
appClick(numeroFila, numberoColumna) {
    let nuevosvalores = this.state.valores.slice();
    let nuevoValor = this.state.turno === JUGADORX ? 'X' : '0';
    nuevosvalores[numeroFila][numberoColumna] = nuevoValor;
    this.setState({
        turno: this.state.turno === JUGADORX ? JUGADOR0 : JUGADORX,
        valores: this.state.valores,
        movimientos: this.state.movimientos + 1
    });
}
```

Paso 5 – Mas acciones

Botón reset

- Queremos hacer un botón “reset” o “reiniciar partida”
- Lo que tiene que hacer es poner el estado inicial de nuevo
- Hacemos un método resetClick() en App.jsx que reinicia el estado

```
resetClick(){
  this.setState({
    turno: JUGADORX,
    valores: [
    ['-', '-', '-'],
    ['-', '-', '-'],
    ['-', '-', '-'],
    ],
    movimientos: 0
  });
}
```

- No olvidar el bind(this) en el constructor:

```
this.resetClick = this.resetClick.bind(this);
```

- En el render añadimos un componente Reset (recordar también importarlo con `import Reset from './Reset.jsx';`) y le pasamos este método para que pueda llamarlo

```
<Reset resetClick={this.resetClick}></Reset>
```

Paso 5 – Mas acciones

Botón reset

- Hacemos el nuevo componente Reset.jsx que recibe como prop el método a llamar como siempre:

```
import React from 'react';

export default class Reset extends React.Component {
    constructor(props) {
        super(props);
        this.click = this.click.bind(this);
    }
    click() {
        this.props.resetClick();
    }
    render() {
        return(
          <button onClick={this.click} >
            Reset
          </button>
        );
    }
}
```

Paso 6 – React Bootstrap

Qué es Bootstrap - instalación

- Bootstrap es un framework para desarrollo web
  - Provee principalmente CSS, HTML y JS
  - Ejemplos: grid, modales, botones, tabs, breadcrumbs, …
  - http://getbootstrap.com/
- React-bootstrap es lo mismo pero reescrito para React
  - https://react-bootstrap.github.io/
- Se instala con:
  - npm install --save react-bootstrap
- También hay que añadir el CSS de Bootstrap y un tema

```
<!-- Latest compiled and minified CSS -->
<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/latest/css/bootstrap.min.css">
<!-- Optional theme -->
<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/latest/css/bootstrap-theme.min.css">
```

Paso 6 – React Bootstrap

Qué es Bootstrap - instalación

- En los archivos en los que queramos utilizar componentes de React-bootstrap primero los importamos y luego los introducimos en el render
- Importante mirar la documentación para ver que props tienen
  - https://react-bootstrap.github.io/components.html
- Ejemplos:

Paso 6 – React Bootstrap

Qué es Bootstrap - Ejemplo

- En los archivos en los que queramos utilizar componentes de React-Bootstrap primero los importamos y luego los introducimos en el render
- Importante mirar la documentación para ver que props tienen
  - https://react-bootstrap.github.io/components.html
- Ej Reset.jsx con button de Bootstrap (https://react-bootstrap.github.io/components.html#buttons):

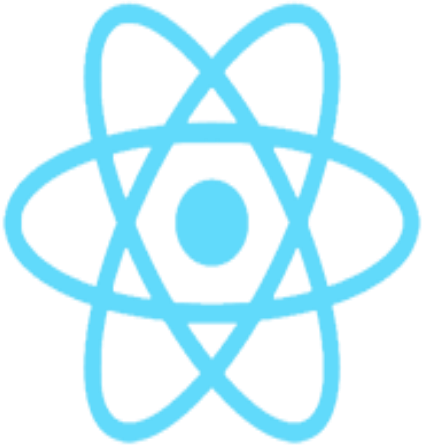
```
import React from 'react';
import { Button } from 'react-bootstrap';

export default class Reset extends React.Component {
    constructor(props) {
        super(props);
        this.click = this.click.bind(this);
    }
    click() {
        this.props.resetClick();
    }
    render() {
        return(
          <Button bsStyle="info" onClick={this.click}>Reset</Button>
        );
    }
}
```

Paso 6 – React Bootstrap

Qué es Bootstrap - Ejemplo

- Ejemplo en la cabecera poner un panel (https://react-bootstrap.github.io/components.html#panels-contextual)

```
import React from 'react';
import { Panel } from 'react-bootstrap';

const title = (
  <h3>Turno</h3>
);

export default class Cabecera extends React.Component {
    render() {
        return (
          <Panel header={title} bsStyle="info" className="turno">
            {this.props.texto}
          </Panel>
        );
    }
}
```



React

¡Muchas gracias!