

Programação & Serviços Web

React – Aula 2

2019/2020

State/Props

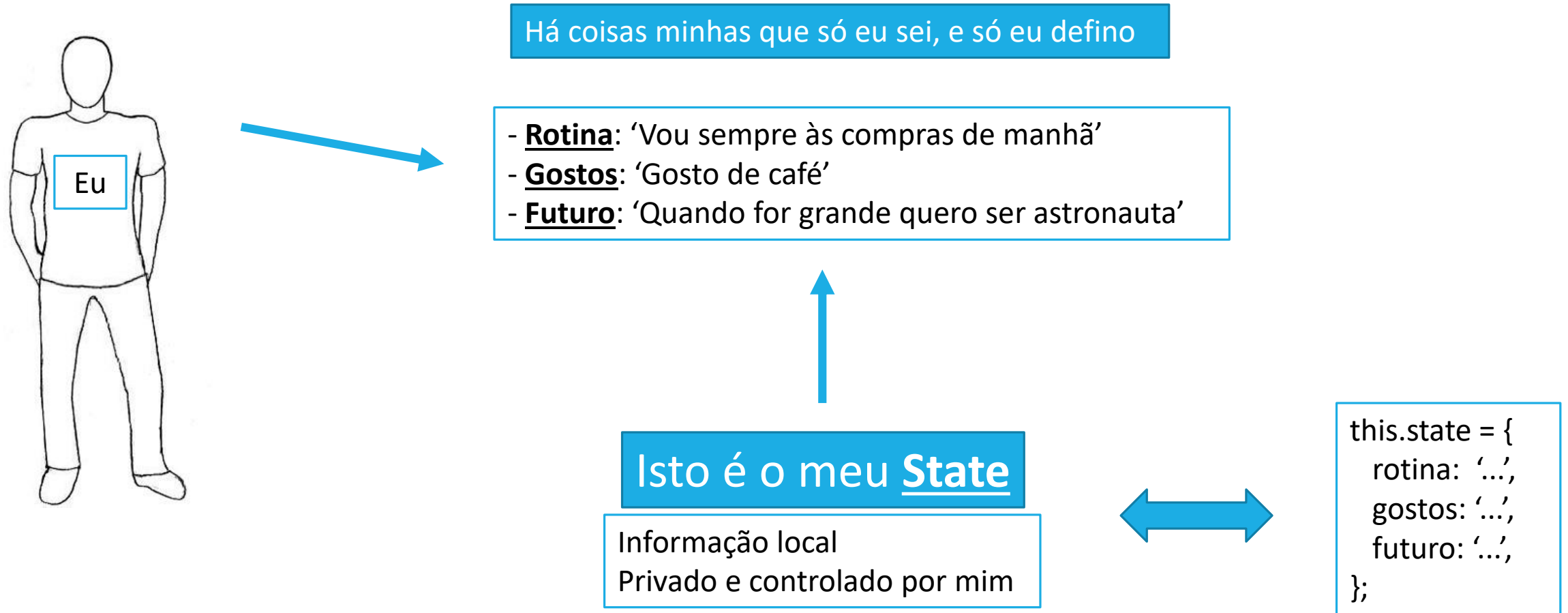
Gestão de Eventos

Aula 2

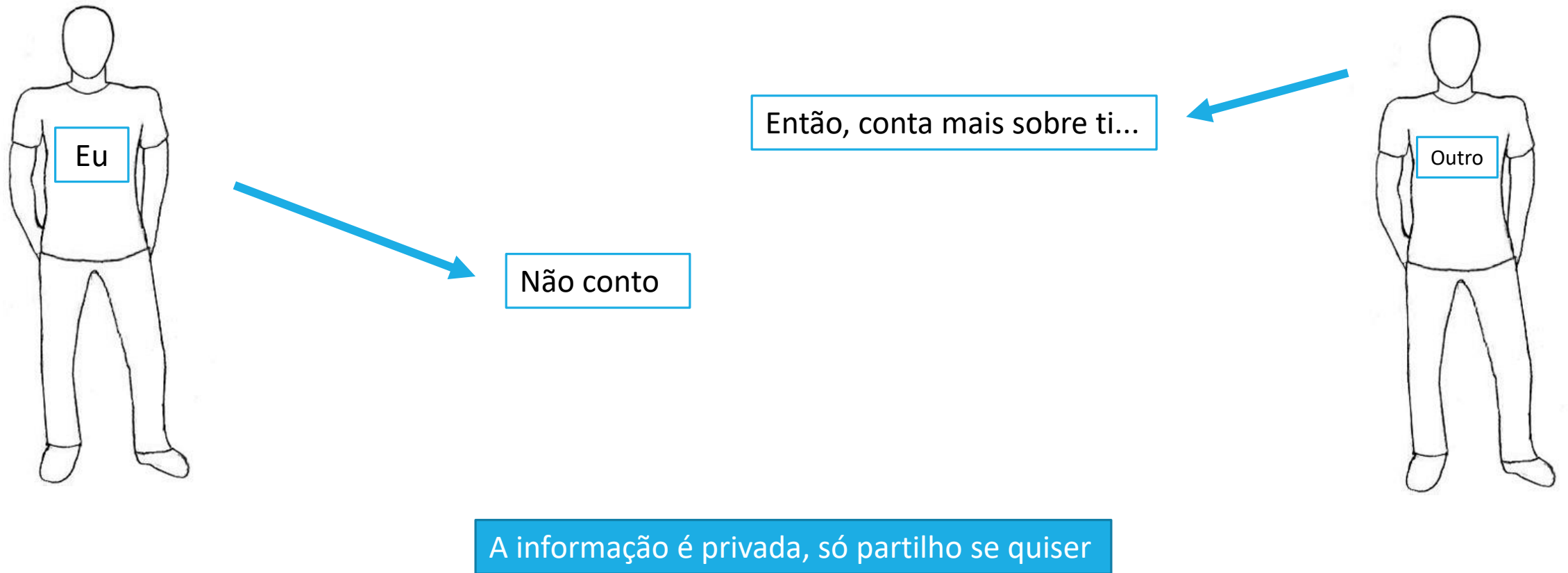
- Estados / Propriedades (State / Props)
- Listas & Chaves
- Manipulação de Eventos
- Revisão Componentes
- Render Condicional

State / Props

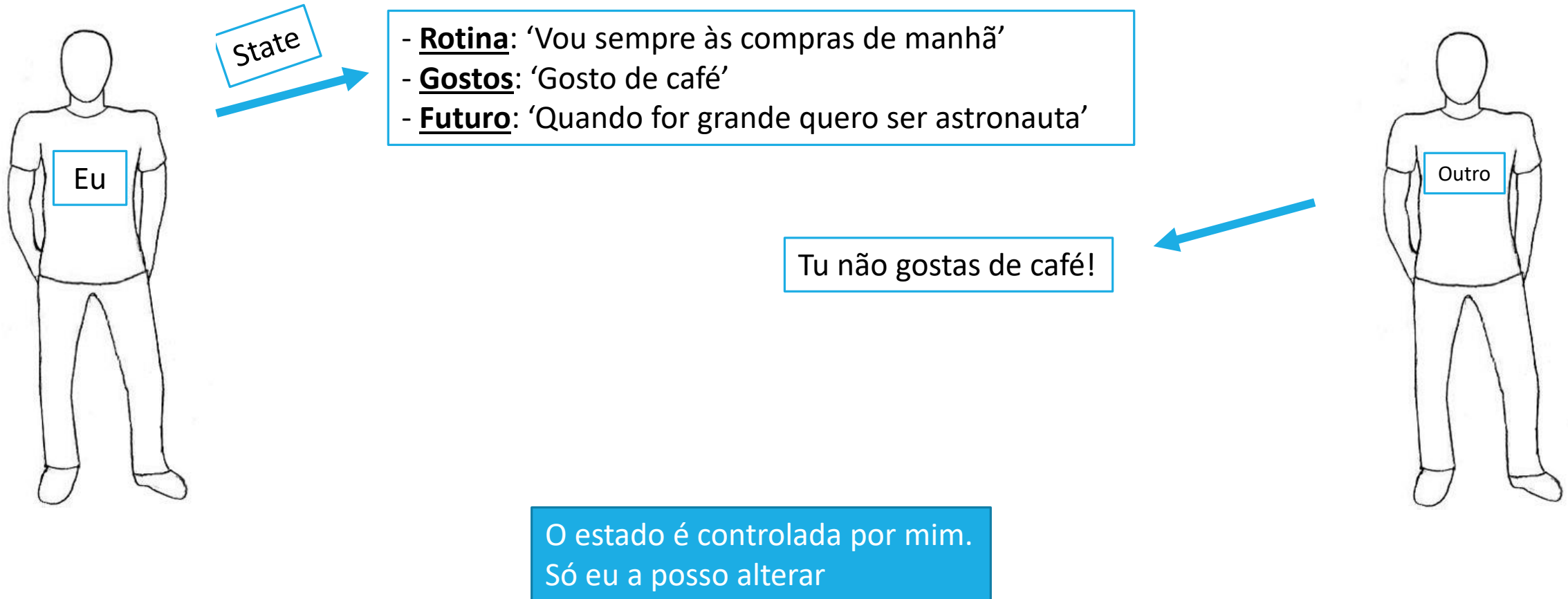
State



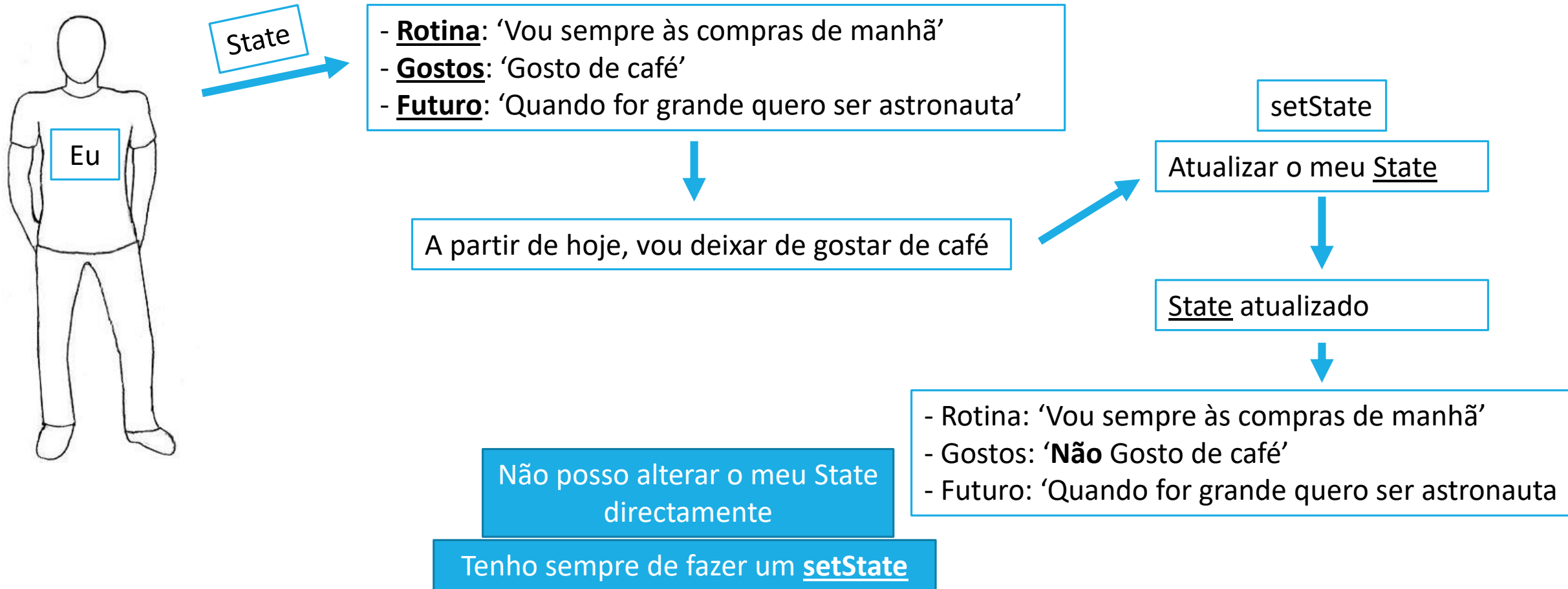
State



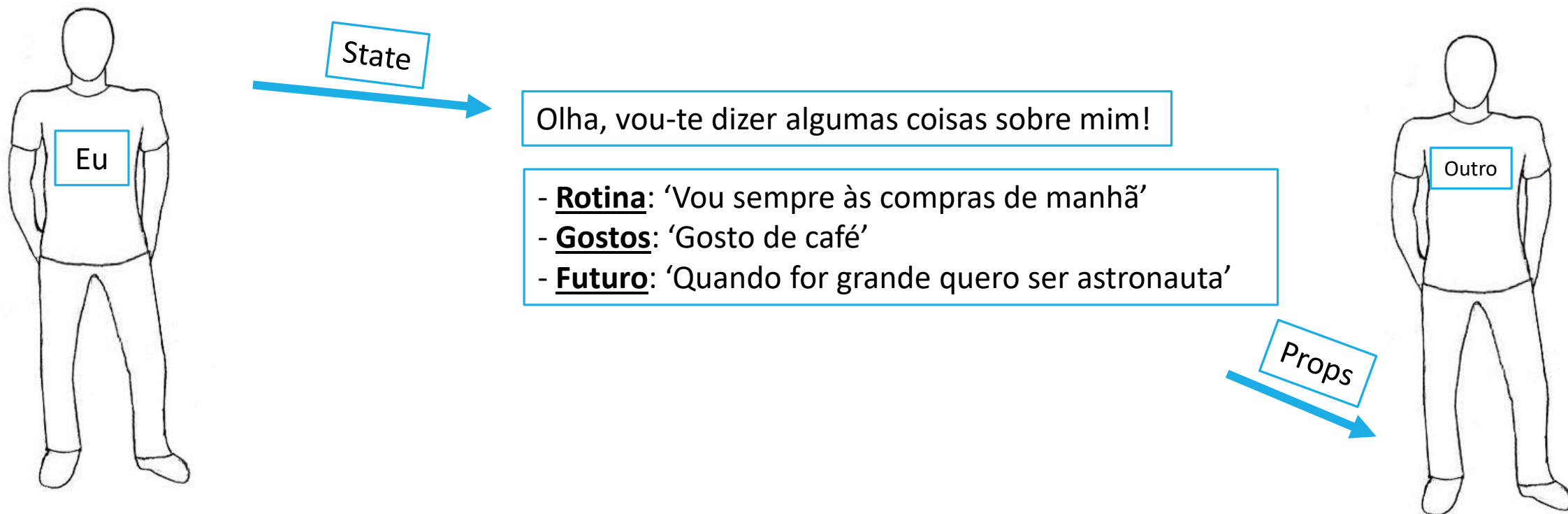
State



State

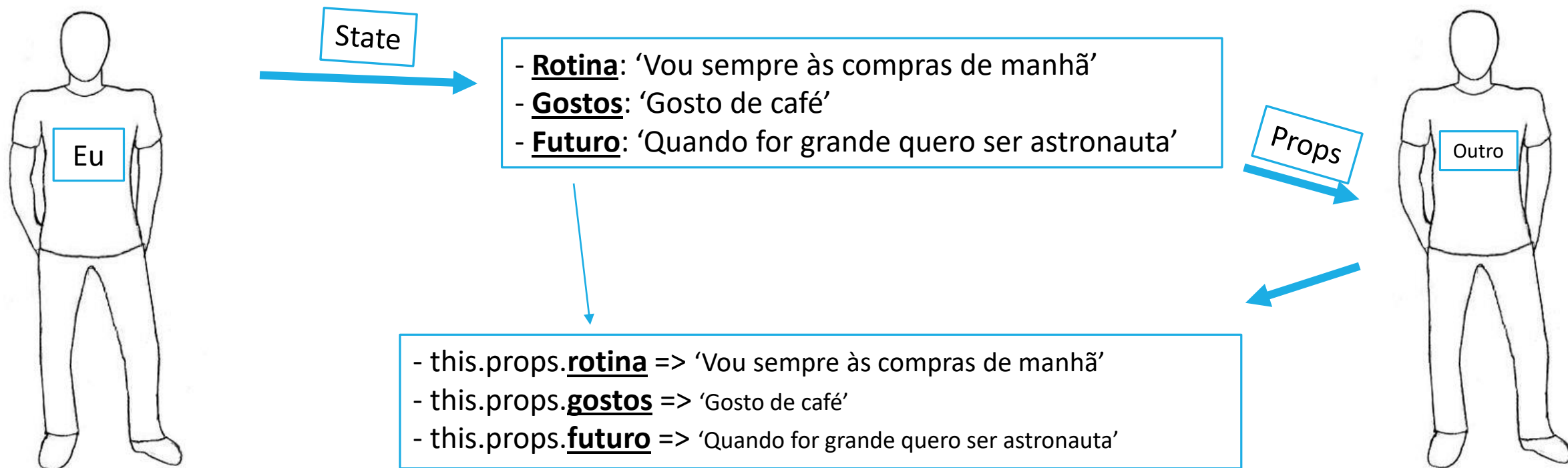


State



O meu state transforma-se no Props dos componentes para o qual envio

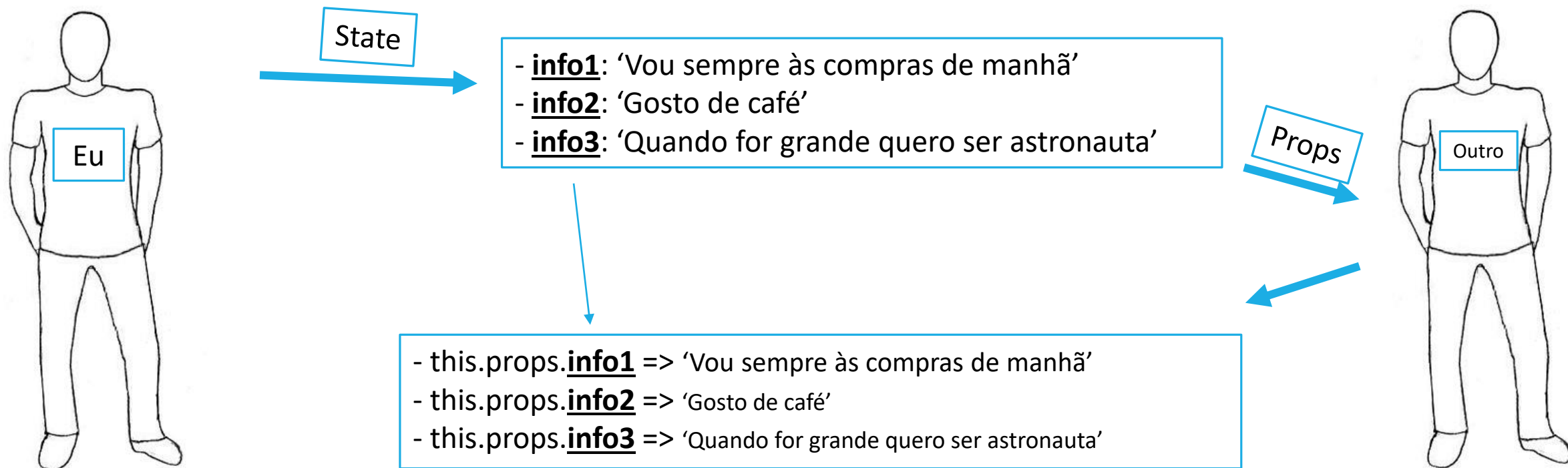
State



O meu state transforma-se no Props do Children

O nome dos campos enviados corresponde ao nome dos campos recebidos

State



O meu state transforma-se no Props do Children

O nome dos campos enviados corresponde ao nome dos campos recebidos

State

- Cada **componente** pode guardar informação local no seu **state**
 - **Privado** e **controlado** totalmente pelo componente
 - Pode ser passado como “props” para outros componentes
- Só apenas **Componentes de Classe** é que podem ter **State**

```
Constructor(props){  
  super(props);  
  
  this.state = {  
    myname = "Nuno Carapito"  
  }  
}
```

Actualizar o state

- Os estados só podem ser modificados através da função setState

```
function onNameChange(newName){  
  this.setState({  
    my_name : newName  
  });  
}
```

- Nunca faça o seguinte:

```
This.state.my_name = newName;
```

Exercício 1

Adicione o construtor à classe App.js

Exercício 2

Adicione a key “my_name” ao state do App.js e atribua-lhe o seu nome

Props

- Os atributos JSX são passados de um componente para outro
 - Estão disponíveis no objeto “props” (this.props),
 - Pode-se enviar o número de elementos que se pretende
 - Os valores dos props não podem ser actualizados (são read-only)

Exemplo Props

```
<Informacao gostos={this.state.gostos} futuro={this.state.futuro} >
```

No componente informação, para aceder a:

- Gostos : `this.props.gostos`
- Futuro: `this.props.futuro`

```
<Informacao info1={this.state.gostos} info2={this.state.futuro} >
```

No componente informação, para aceder a:

- Gostos : `this.props.info1`
- Futuro: `this.props.info2`

Exercício 3

Envia o state “myname” do App.js para o Header.js

Mostra a key enviada no header.js

Listas/Vetores

- Listas/Vetores funcionam de forma semelhante ao Javascript

```
var list = {  
    key1: "Bom dia"  
    key2: "Boa tarde",  
    key3: "Boa noite"  
}
```




```
var listaIndividual = (  
    <div>  
        <span>{list.key1}</span>  
        <span>{list.key2}</span>  
        <span>{list.key3}</span>  
    </div>  
);
```

Exemplo Listas/Vetores

```
var list = [  
  {  
    key: "Bom dia"  
  },  
  {  
    key: "Boa tarde"  
  },  
  {  
    key: "Boa noite",  
  }  
]
```

```
var listaIndividual = (  
  { list.map((elementoDoVetor, pos) =>  
    <span key={pos}>{elementoDoVetor.key}</span>  
  }  
);
```



Quando se faz um ciclo, deve-se colocar sempre a "key", para que o react saiba qual o elemento a atualizar

Pode ser a posição do ciclo / ID que o elemento tenha

Gestão de Eventos

- Semelhante ao que se faz no Javascript / jQuery
- Usar camelCase para especificar eventos
- Passamos a função no handler do evento

- Exemplo:

```
<button onClick="changeType()" />
```

- Em react:

```
<button onClick={this.changeType} />
```

```
changeType() {  
  this.setState(state => ({  
    isToggleOn: !state.isToggleOn  
  }));  
}
```

Arrow Functions

- As arrow functions são uma forma mais simples de escrever expressões. Utilizam a seta (`=>`) e por isso é que têm o nome que têm. Para além disso, faz com que não seja preciso fazer bind ao this
- Para simplificar o uso do “this” no React, utilizam-se as Arrow Functions.

```
changeType = () => {  
  this.setState(state => ({  
    isToggleOn: !state.isToggleOn  
  }));  
}
```

```
<button onClick={() => this.changeType()} />
```

Com as Arrow Functions, onClick muda um pouco

Arrow Functions

```
<button onClick={(e) => this.deleteRow(id, e)}>Delete Row</button>  
<button onClick={this.deleteRow.bind(this, id)}>Delete Row</button>
```

Estas duas linhas são equivalentes:

- A primeira utiliza a arrow function (passa o **ID** e o **E**, que é o event)
- A segunda utiliza o this, para enviar para a função o que é que significa o “this”

Ao contrário das funções standard, o argumento E(event) têm de ser enviado para a função, no caso de o quisermos usar

Enviar funções para filhos

- As funções podem ser enviadas para um componente filho
- São enviadas da mesma maneira que qualquer outro parâmetro
- No componente filho, pode invocar a função chamando `this.props.Nome_funcao`

```
<Dashboard removeHero={this.removeHero}/>
```

```
removeHero = () => {  
  //TODO  
}
```

Componente Dashboard

```
<button onClick={() => this.props.removeHero(element.id)}>Remover</button>
```

Revisão Componentes

Tipos de Componentes

- Os componentes podem ser classificados pela forma como são usados:
 - Apresentação vs Conteúdo (Stateless vs Stateful)
 - Magros vs Gordos
 - Burros vs Inteligentes
 - Sem Estado vs Com estado

Ou

- Função vs Classe
- Todos estes nomes representam a mesma coisa (coluna esquerda vs coluna direita).
- A primeira e a última linha são as mais comum

Componentes de Apresentação / Função

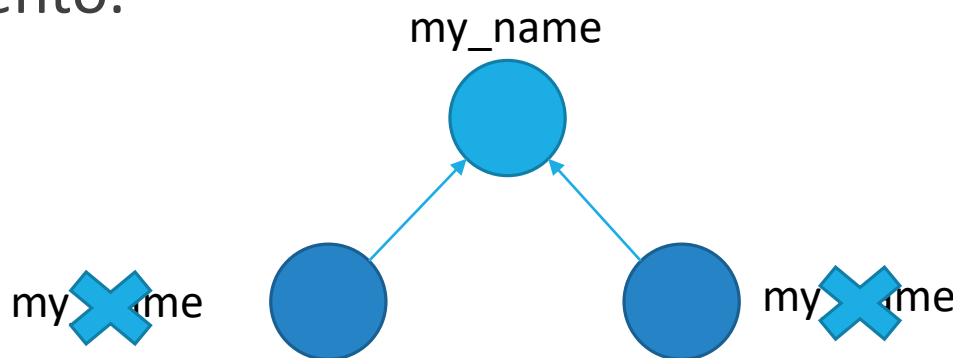
- Só se preocupam em renderizar a “view”
 - Como é que as coisas devem aparecer (markup, css)
- Renderiza a view baseado na informação que é enviada para eles
 - Através dos props
- Não têm state
 - Só se deve preocupar com a UI, e não com estados

Componentes de Conteúdo

- Responsável por fazer coisas funcionar
 - Ir buscar data, actualizar o estado, etc
- Utiliza/chama os componentes de apresentação no método Render
 - Não se preocupa com a interface, delega a responsabilidade nos componentes de apresentação
- Envia os dados para os componentes de **função**
- Gere o estado e comunica com os dados da aplicação

Variáveis – Subir de Nível

- Por vezes, alguns componentes podem ter dados que são comuns a vários sítios ou a mudança de dados num dos componentes precisa de ser refletida noutro componente
 - Imagine que quer ter o state `my_name` no header e no footer.. O que faz sentido nestes casos é coloca-lo no Content, para que depois seja enviado para os dois lados.
- O melhor é mover o state partilhado para um componente ancestral comum. Por vezes a aplicação precisa de ser re-adaptada ao longo do seu desenvolvimento.



Estrutura do Projeto (Pastas)

- Não existe uma regra a seguir na organização das pastas. Cada equipa de desenvolvimento tem a sua estrutura preferida.
 - Há quem divida os componentes por pastas (e, em cada uma destas pastas, tem o ficheiro do componente e o correspondente .css)
 - Há quem divida por componentes de função / de classe.
 - Há quem agrupe por feature
- A sugestão é não perder mais que 5 minutos ao discutir com a equipa como é que deve ser organizada a estrutura do projeto.
- Existem numerosas explicações no google, é uma questão de procurar e ver qual se adapta melhor.

Render Condicional

- Imagine que na sua aplicação quer que seja mostrada uma mensagem caso uma condição se verifique, ou outra caso não.

```
function Greeting(props) {  
  const isLoggedIn = props.isLoggedIn;  
  if (isLoggedIn) {  
    return <UserGreeting />;  
  }  
  return <GuestGreeting />;  
}
```

Neste exemplo é um componente de função. No entanto, o mesmo acontece em componente de classe. Para além dos props, pode-se usar o state ou quaisquer outras variáveis

Render condicional – Variáveis

```
let button;  
if (isLoggedIn) {  
  button = <LogoutButton onClick={this.handleLogoutClick} />;  
} else {  
  button = <LoginButton onClick={this.handleLoginClick} />;  
}  
return (  
  <div>  
    {element}  
  </div>  
)
```

Neste caso, está a declarar-se uma variável que vai ser um componente ou o outro, de acordo com a condição.
No final, um destes dois elementos será renderizado

Render Conditional – If/else Inline

- O if/else inline funciona da seguinte forma

condition ? true : false

Se a condição for **verdadeira**, vai acontecer o elemento à frente do **?**

Se a condição for **falsa**, vai acontecer o elemento à frente do **:**

- Adaptando isto a elementos do React:

```
const isLoggedIn = this.state.isLoggedIn;  
return (  
  <div>  
    The user is <b>{isLoggedIn ? 'currently' : 'not'}</b> logged in.  
  </div>  
);
```


Render Conditional – If/else Inline

- Ou outro exemplo

```
const isLoggedIn = this.state.isLoggedIn;  
return (  
  <div>  
    {isLoggedIn ? (  
      <LogoutButton onClick={this.handleLogoutClick} />  
    ) : (  
      <LoginButton onClick={this.handleLoginClick} />  
    )}  
  </div>  
);
```

Verdadeiro

Falso

Render Condicional – If/else Inline

- No caso de a vossa condição não tiver uma das condições, pode colocar null

```
const isLoggedIn = this.state.isLoggedIn;  
return (  
  <div>  
    {isLoggedIn ? (  
      <LogoutButton onClick={this.handleLogoutClick} />  
    ) : (  
      null  
    )}  
  </div>  
);
```

Verdadeiro

Falso

Render Condicional – If/else Inline

- No caso de querer que algum componente não renderize em alguma das condições, basta colocar um `return null`

```
function WarningBanner(props) {  
  if (!props.warn) {  
    return null;  
  }  
  
  return (  
    <div className="warning">  
      Warning!  
    </div>  
  );  
}
```

Neste caso, se a propriedade `props.warn` for falsa, não é mostrada a mensagem

Ficha 2

Leitura Complementar

- <https://reactjs.org/docs/components-and-props.html>
- <https://reactjs.org/docs/state-and-lifecycle.html> (Até à secção Adding Lifecycle Methods to a Class)
- <https://reactjs.org/docs/handling-events.html>
- <https://reactjs.org/docs/conditional-rendering.html>
- <https://reactjs.org/docs/lists-and-keys.html>
- <https://reactjs.org/docs/lifting-state-up.html>
- <https://reactjs.org/docs/composition-vs-inheritance.html>
- <https://reactjs.org/docs/faq-structure.html>