Efeitos Colaterais e useEffect

Labenu_

O que vamos ver hoje?

- O que são efeitos colaterais (ou *side effects*) no React
- Hook **useEffect**
- Substituir métodos de **lifecycle** usando useEffect
- Erros comuns

Efeitos Colaterais

Labenu_

O que são Efeitos Colaterais? 🏌️

- Um Efeito Colateral (ou *Side Effect*) é quando uma função altera algo **fora do seu próprio escopo**

- **Exemplo:** Trocar de casaco
  - Ação independente
  - Consequência global
  - Gera efeito colateral

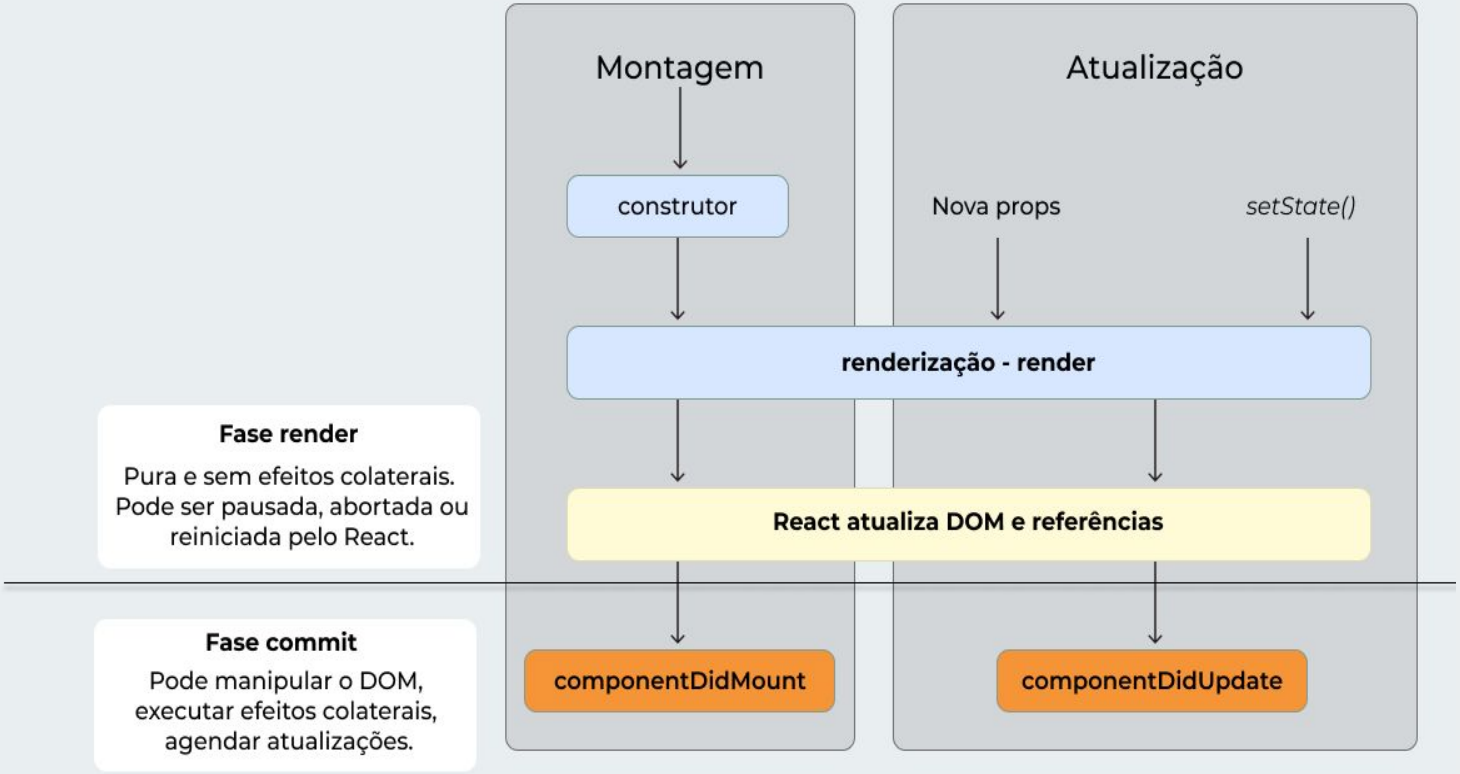
O que são Efeitos Colaterais? 🤾

- No React, um Efeito Colateral é quando uma função afeta algo fora do seu escopo

- É recomendado que efeitos colaterais aconteçam sempre **depois da renderização do componente**

- Em componentes de classe, fazemos isso usando **métodos de ciclo de vida**
  - `componentDidMount()`
  - `componentDidUpdate()`

Ciclo de vida em Componentes de Classe



Ciclo de vida - componentDidMount

```
1  class UsersList extends React.Component {
2    state = {
3      usersList: []
4    }
5
6    componentDidMount() {
7      this.getUsers()
8    }
9
10   getUsers = async () => {
11     try {
12       const response = await axios.get(`${baseUrl}/users`)
13       this.setState({usersList: response.data})
14     } catch(error) {
15       console.log(error)
16     }
17   }
18
19   render() {...}
20 }
```

Ciclo de vida - componentDidUpdate

```
class UserInfo extends React.Component {
  state = {
    user: {}
  }

  componentDidMount() {
    this.getUserById(this.props.userId)
  }

  componentDidUpdate(prevProps) {
    // componentDidUpdate recebe prevProps que contém o valor das
    // props na renderização anterior.
    // Deve ser usado para tomar ações somente quando uma prop mudar.
    if(prevProps.userId !== this.props.userId) {
      this.getUserById(this.props.userId)
    }
  }

  getUserById = async (id) => {
    try {
      const response = await axios.get(`${baseUrl}/users/${id}`)
      this.setState({user: response.data})
    } catch(error) {
      console.log(error)
    }
  }

  render() {...}
}
```

useEffect()

Labenu_

useEffect 🧦

- O useEffect() é uma função responsável por **executar outra função** após renderização do componente

- É o Hook que nos permite **reproduzir** a funcionalidade dos métodos `componentDidMount()` e `componentDidUpdate()` em componentes funcionais
  - Não é **exatamente** equivalente!
  - Meios diferentes de atingir o **mesmo fim**

useEffect 🧦

- Ele deve ser importado do React entre chaves

```
import React, {useEffect} from 'react'
```

Adicionar isso ao topo do arquivo quando quisermos usar o useEffect()

useEffect 🧦

- Recebe dois parâmetros:
  - Função a ser executada (**não** pode ser async)
  - Array de dependências

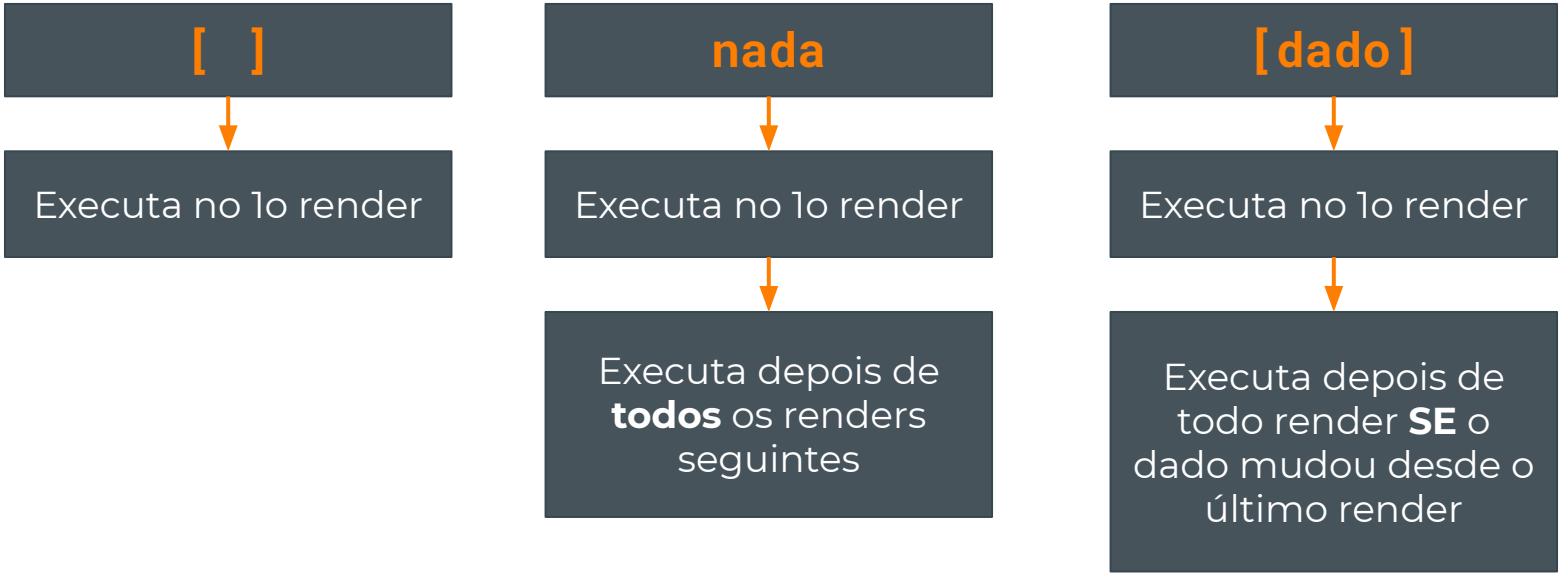
```
useEffect(() => {
  //função de efeito colateral
  getUsers()

}, [])
```

Função a ser executada

Array de dependências (opcional)

useEffect 🧦

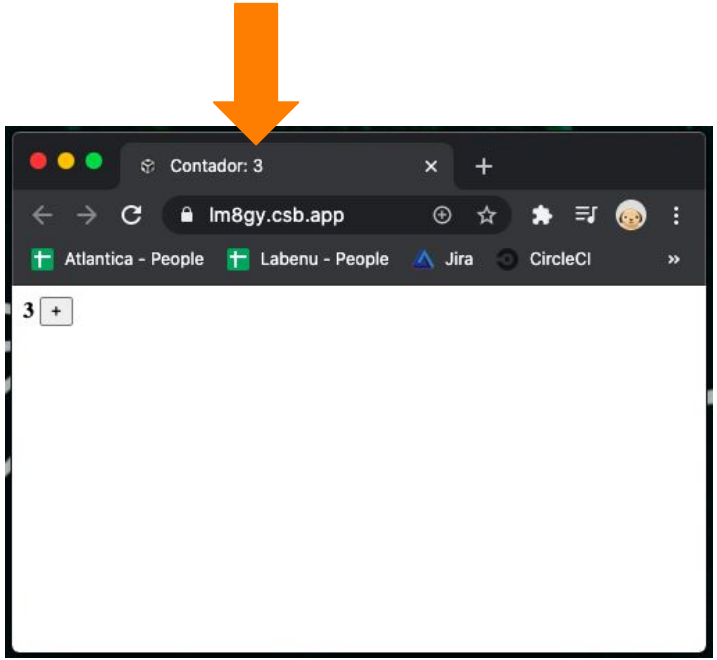
- Quando alguma das **dependências** do array mudar, a função (efeito colateral) é executada novamente.



Exercício 1

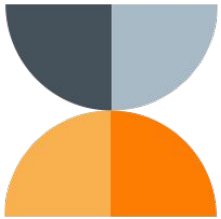
- Crie um componente funcional de contador que possui um botão de incrementar

- Toda vez que o contador mudar, mude também o nome da aba do seu site



Pausa para relaxar 😴

10 min



- Efeitos colaterais são funções que afetam algo **fora do seu escopo**
- No React, efeitos colaterais devem acontecer **pós-renderização**
- Em componentes funcionais, usamos o hook **useEffect()**

useEffect x didMount e didUpdate

Labenu_

componentDidMount x useEffect

```
class UsersList extends React.Component {
  state = {
    usersList: []
  }

  componentDidMount() {
    this.getUsers()
  }

  getUsers = async () => {
    try {
      const response = await axios.get(`${baseUrl}/users`)
      this.setState({usersList: response.data})
    } catch(error) {
      console.log(error)
    }
  }

  render() {...}
}
```

```
const UsersList = (props) => {
  const [users, setUsers] = useState([])

  getUsers = () => {
    axios.get(`${baseUrl}/users/`)
    .then(response => setUsers(response.data))
    .catch(error => console.log(error))
  }

  useEffect(() => {
    // Função que será executada após renderização
    getUsers()

    // Array de dependências (vazio)
    // Roda após primeiro render
  }, [])

  return(...)
}
```

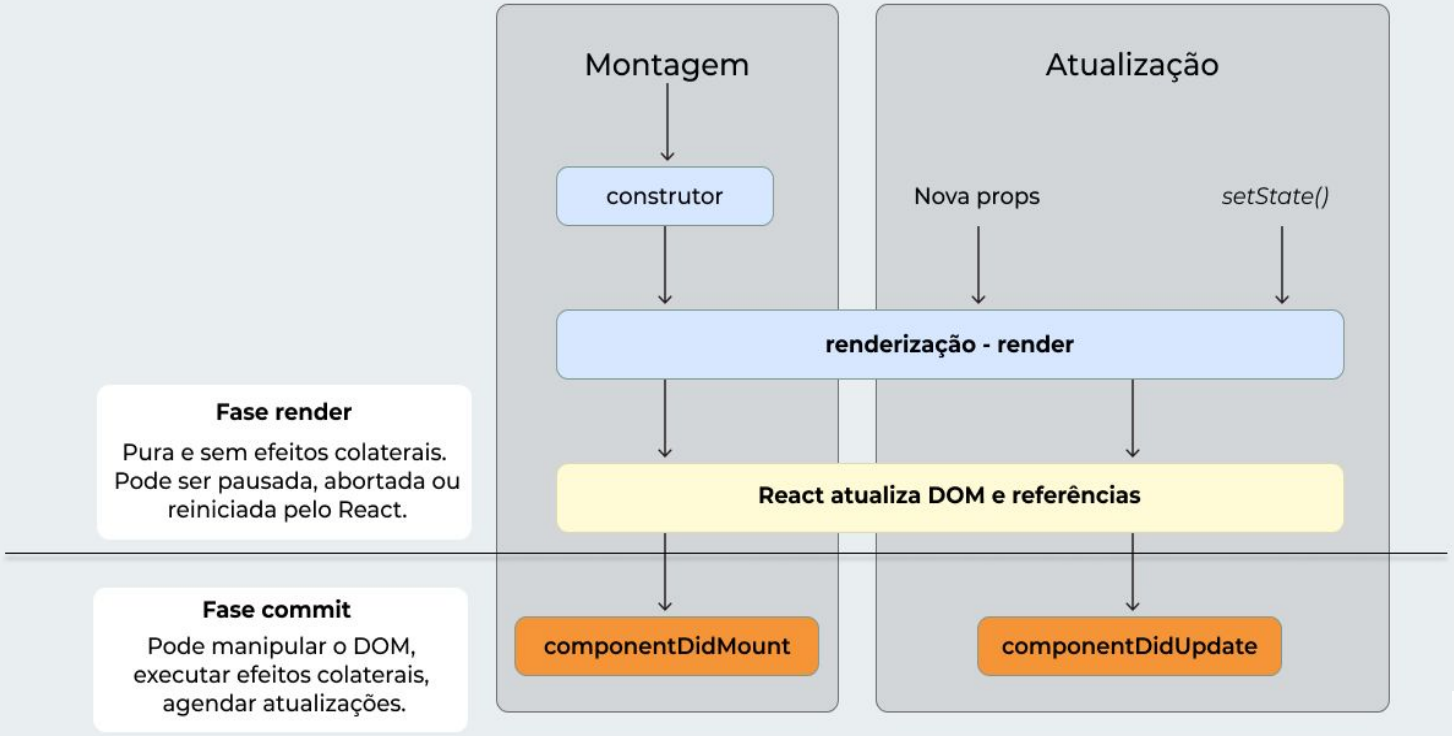
componentDidUpdate x useEffect

```
class UserInfo extends React.Component {
  state = {
    user: {}
  }

  componentDidMount() {
    this.getUserById(this.props.userId)
  }

  componentDidUpdate(prevProps) {
    // componentDidUpdate recebe prevProps que contém o valor das
    // props na renderização anterior.
    // Deve ser usado para tomar ações somente quando uma prop mudar.
    if(prevProps.userId !== this.props.userId) {
      this.getUserById(this.props.userId)
    }
  }

  getUserById = async (id) => {
    try {
      const response = await axios.get(`${baseUrl}/users/${id}`)
      this.setState({user: response.data})
    } catch(error) {
      console.log(error)
    }
  }

  render() {...}
}
```

```
const UserInfo = (props) => {
  const [user, setUser] = useState({})

  // Função que será executada após renderização
  useEffect(() => {
    // Declarando função assíncrona, pois useEffect
    // não pode ser assíncrono
    const getUserById = async () => {
      try {
        const response = await axios.get(`${baseUrl}/users/${props.userId}`)
        setUser(response.data)
      } catch(error) {
        console.log(error)
      }
    }

    // Chama função que acabou de ser declarada
    getUserById()
  }, [props.userId])

  return (...)
}
```

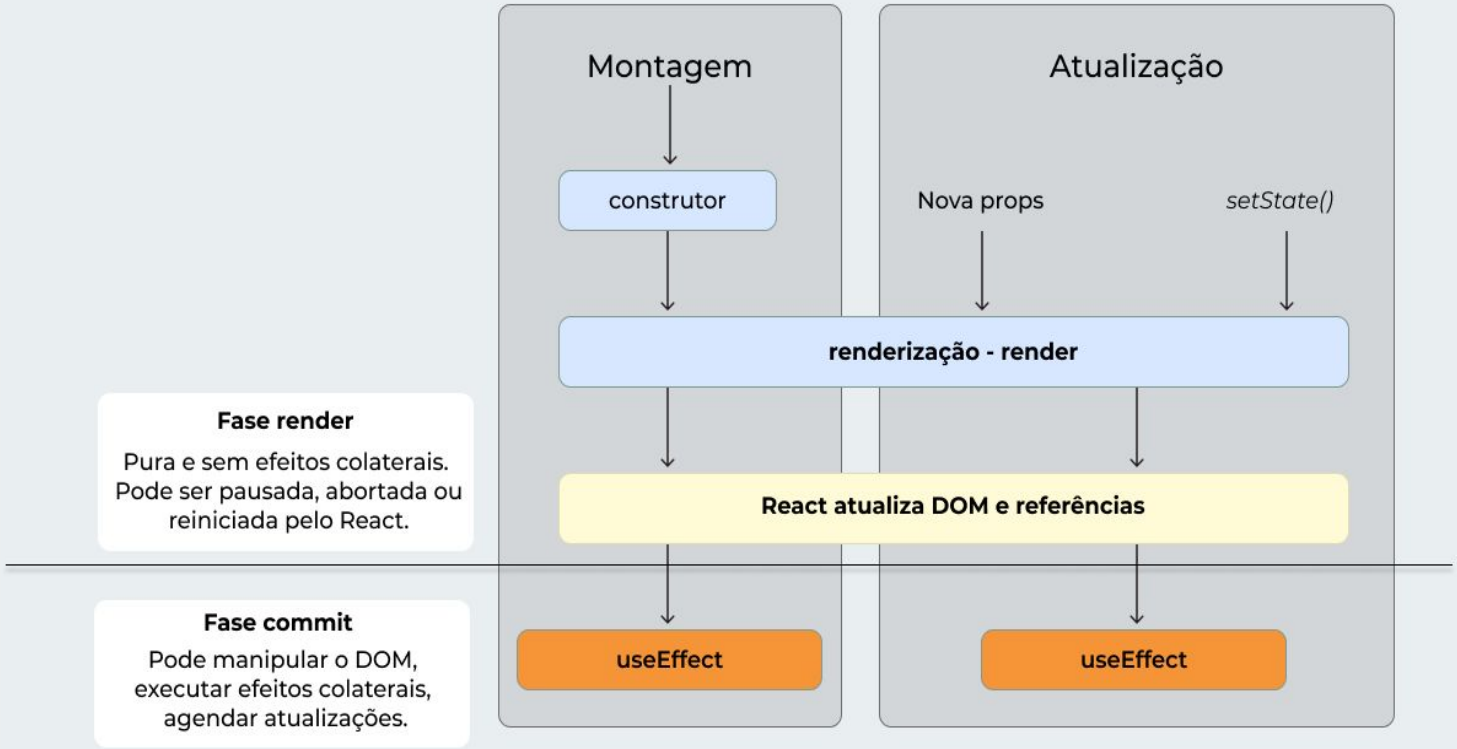
Labenu_

Ciclo de Vida - Componentes de Classe



benu_

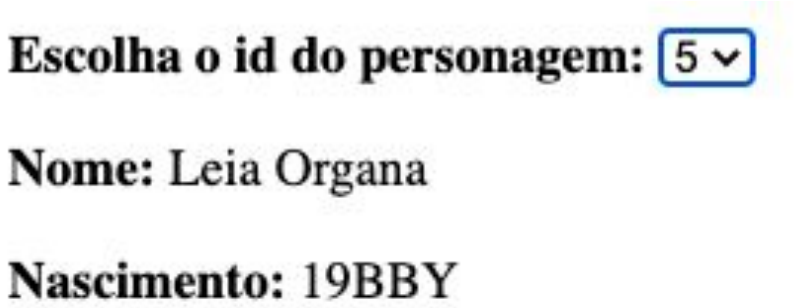
Ciclo de Vida - Componentes Funcionais



Exercício 2

- Usando a API de Star Wars, faça um site que permita o usuário selecionar os ids dos 5 primeiros personagens e mostre seu **nome** e seu **ano de nascimento**

- Link da API



Array de dependências

Labenu_

Array de Dependências 🧦

- Como vimos, o array de dependências é quem determina QUANDO a nossa função será executada

Array de Dependências 🧦

- Em geral, nós **sempre queremos ter um array de dependências**! Ele pode ser vazio ou ter valores.

- Se deixamos de passá-lo, a função de efeito colateral será executada toda vez que **qualquer prop ou estado do componente mudar**

- Isso pode, inclusive, gerar loops infinitos e outros bugs estranhos. Portanto, evitamos fazer isso

Array de Dependências 🧦

- Assim sendo, sobram os dois casos principais:

Array de Dependências - Vazio 🧦

- Se você só quer que a função seja executada **uma vez**, quando a tela abrir, o array fica vazio

- Um exemplo comum é quando queremos carregar dados na tela assim que ela abre

```
1 useEffect(() => {
2   getPostsList()
3 } , [])
```

Array de Dependências - Dados 🧦

- A outra possibilidade é colocando um (ou mais) dados no seu array de dependências

- A função será executada **todas as vezes que qualquer um desses dados mudar**

```
1 const [contador, setContador] = useState(0)
2
3 useEffect(() => {
4   alert("Você mudou o valor do contador!")
5 } , [contador])
```

Erros comuns 🧦⚠️

- Esquecer alguma dependência no array: EM GERAL quando usamos uma prop ou estado na função do useEffect, **queremos saber quando ela muda** para executar novamente a função

- Se você colocou um console.log dentro do seu useEffect e ele não está sendo chamado no momento que você queria, **verifique se não esqueceu de colocar alguma dependência no array**

Erros comuns 🧦⚠️

- Colocar na função uma mudança de estado e colocar o valor do estado no array de dependências causa um loop infinito

```
1 const [frase, setFrase] = useState("")
2
3 useEffect(() => {
4   setFrase("Bom dia turma!")
5 } , [frase])
```

Resumo

Labenu_

Resumo 📓

- Para substituir os métodos de ciclo de vida `componentDidMount()` e `componentDidUpdate()` em componentes de classe, usamos o hook `useEffect()`

- Ele recebe dois parâmetros:
  - **Função** a ser executada (efeito colateral)
  - Array de **dependências** (opcional)

Resumo 📓

componentDidMount

`[ ]`

Executa no 1o render

Cuidado!

`nada`

Executa no 1o render

Executa depois de **todos** os renders seguintes

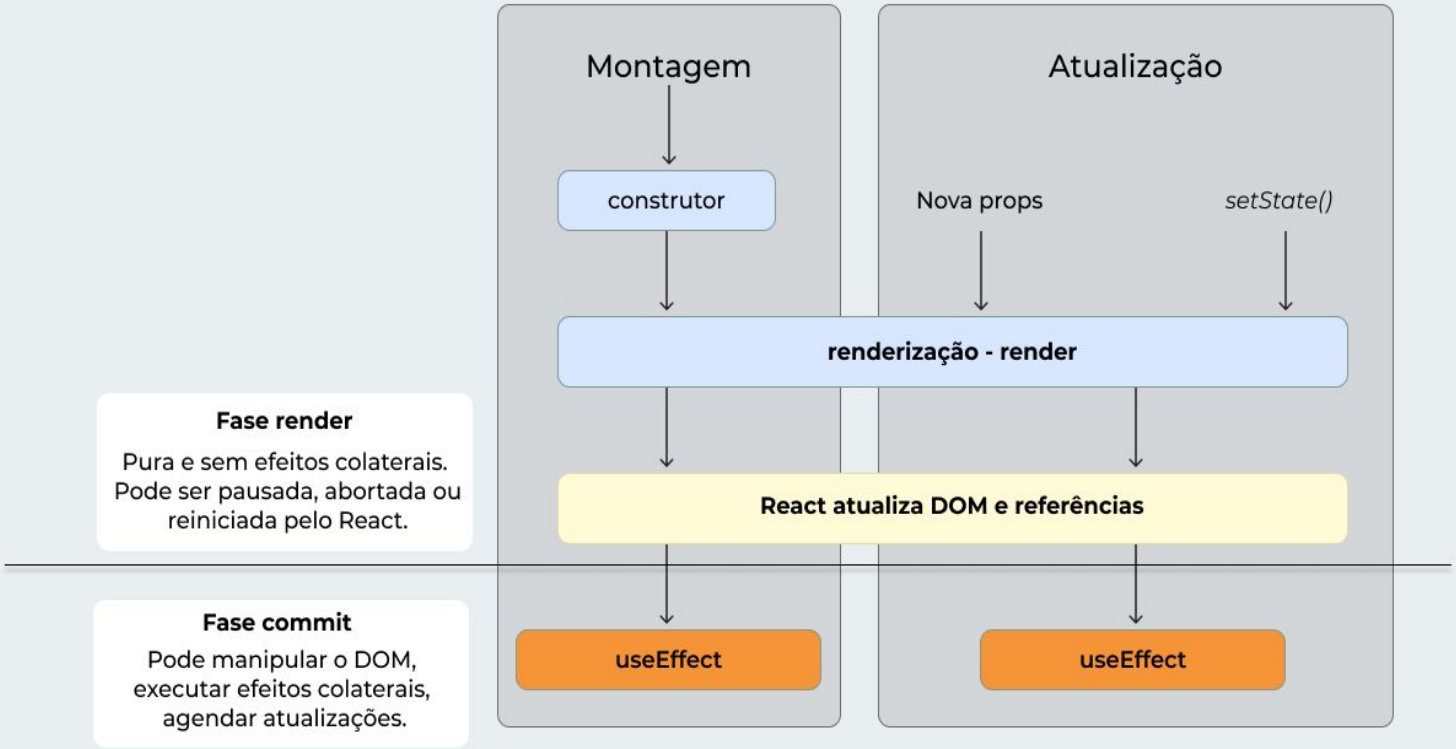
componentDidUpdate

`[dado]`

Executa no 1o render

Executa depois de todo render **SE** o dado mudou desde o último render

Resumo


Montagem
Atualização
construtor
Nova props
setState()
renderização - render
Fase render
Pura e sem efeitos colaterais. Pode ser pausada, abortada ou reiniciada pelo React.
React atualiza DOM e referências
Fase commit
Pode manipular o DOM, executar efeitos colaterais, agendar atualizações.
useEffect
useEffect

Dúvidas? 

Labenu_

Labenu_

Obrigado(a)!