centro universitário FEI

CCP150
DES. DE APLICATIVOS MÓVEIS

Prof. ISAAC
isaacjesus@fei.edu.br

DESENVOLVIMENTO DE APLICAÇÕES MOBILE

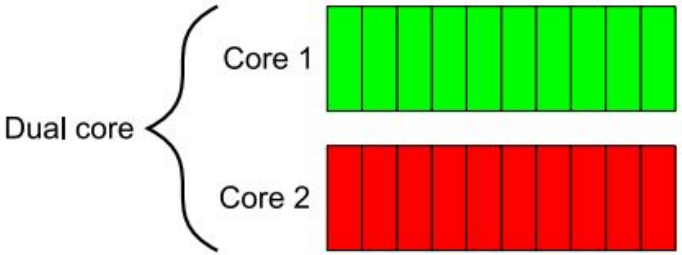
PROGRAMAÇÃO ASSÍNCRONA / CONCORRENTE
PROMISES

Programação Assíncrona

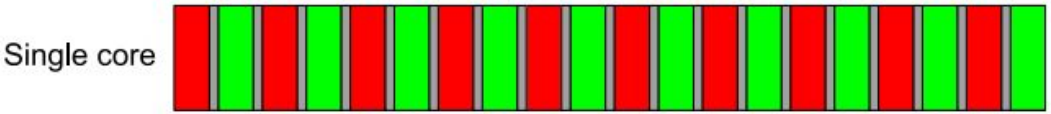
- Podemos ter vários programas rodando ao mesmo tempo no computador
- **Mesmo com somente um único processador (single-core)**
- Cada programa roda por um tempo; Então, sua execução é parada para que outro programa execute um pouco - **Programação Concorrente**
- Todo o processo de troca é muito rápido e temos a impressão que tudo está acontecendo em paralelo, mas na verdade, não está

Programação Assíncrona

- Duas formas de **concorrência**:



Execução paralela em máquina dual-core

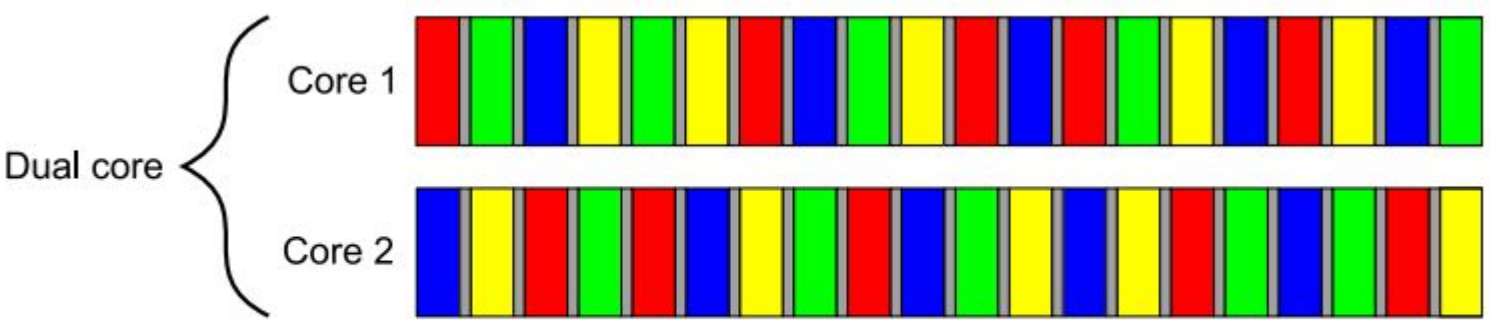


Alternância de tarefas em máquina single-core

Fonte: C++ Concurrency in Action - Anthony Williams

Programação Assíncrona

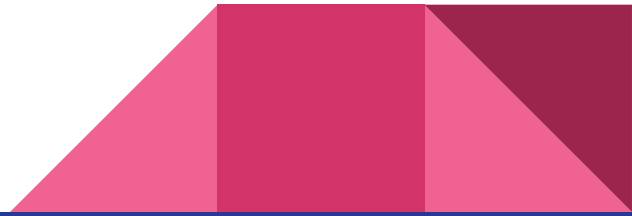
- Alternância entre 4 tarefas em dois núcleos (dual-core):



Fonte: C++ Concurrency in Action - Anthony Williams

Programação Assíncrona

- De forma geral, o computador todo funciona assim
- Os programas utilizam **sinais e interrupções** para avisarem ao processador que precisam de atenção
- Portanto, **tudo pode ser assíncrono**:
  - Os programas iniciam e ficam executando suas tarefas;
    Quando precisam de atenção enviam um sinal / interrupção



Programação Assíncrona

- Um dos problemas disso é que as linguagens de programação são, normalmente, síncronas
- Muitas delas conseguem assincronicidade por meio bibliotecas, criando ***threads***:
    - A criação de uma nova ***threads*** resulta em um ***novo fluxo de processo***

Programação Assíncrona

- A assincronicidade e o modelo de concorrência do JavaScript é baseado no ***Event Loop*** (Laço de Eventos)
  - Uma propriedade muito interessante do ***event loop***, é que o JavaScript, ao contrário de muitas outras linguagens, nunca bloqueia
  - Por exemplo, E/S são tratadas com callbacks, portanto outras tarefas podem ser executadas enquanto se espera

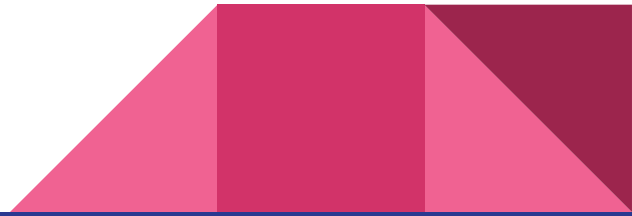
http://latentflip.com/loupe/

Programação Assíncrona

- Dessa forma, para trabalhar com a assincronicidade no JavaScript podemos utilizar:
  - ***callbacks*** e
  - ***promises***

Callback

- O ***callback*** é uma função utilizada para ser invocada no futuro, quando algum evento for detectado
- Ele não é, normalmente, sequencial ao resto do código

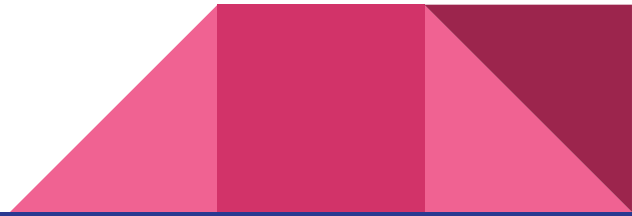


Promises

- No ECMAScript 6, o conceito de ***promises*** aparece
- ***Promise*** é uma ***promessa de execução***
- Promise é uma função que promete uma resposta e quando a resposta acontece ela invoca novas funções
- Quando você lança uma função pela promise ela não bloqueia o código, ela enfileira as funções que estão por vir

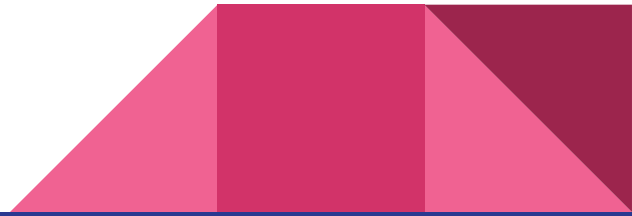
Promises

- Objeto que representa a conclusão de uma operação assíncrona



Promises

- Criamos com ***new Promise( )***
  - Utilizamos com ***then( )*** e ***catch( )***



Exemplos

https://snack.expo.dev/@isaacjesus/aula05-exemplo01-sem_promise

Exemplos

https://snack.expo.dev/@isaacjesus/exemplo01-promise

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

    let minhaPromise = this.ola();
    minhaPromise
    .then(
      (msg) => { console.log(msg); })
    .catch(
      (erro) => { console.log(erro);}
    );
    console.log("final do construtor")
  }
```

Exemplos

https://snack.expo.dev/@isaacjesus/exemplo02-promise

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

    let minhaPromise = this.ola();
    minhaPromise
    .then(
      (msg) => { console.log(msg); return this.ola2()})
    .then(
      (msg) => { console.log(msg);})
    .catch(
      (erro) => { console.log(erro);}
    );
    console.log("final do construtor")
  }
```

Promises

- Problema:
  - Podemos ter muitas promises encadeadas, com vários ***then***, portanto

Promises

- Então no ES6, surge o ***async*** e o ***await***
  - O ***async*** constrói uma promise
  - O ***await*** sincroniza eventos assíncronos *(só pode ser utilizado em blocos assíncronos)*

Exemplos

https://snack.expo.dev/@isaacjesus/exemplo02-async_promise

```
//teste também sem o async e await
async assincrono(){
  try{
    const resposta = await this.ola();
    const resposta2 = await this.ola2();
    console.log(resposta);
    console.log(resposta2);
  }catch(erro){
    console.log(erro);
  }
}
```

Exemplos

https://snack.expo.dev/@isaacjesus/exemplo-aula02-asyncstorage

```
31    async ler(){
32      try{
33        let senha = await AsyncStorage.getItem(this.state.usuario);
34        if(senha != null){
35          if(senha == this.state.senha){
36            alert("Logado!!!");
37          }else{
38            alert("Senha Incorreta!");
39          }
40        }else{
41          alert("Usuário não foi encontrado!");
42        }
43      }catch(erro){
44        console.log(erro);
45      }
46    }
47  }
```

Exemplos

```
60    async gravar(){
61      try{
62        await AsyncStorage.setItem(this.state.user, this.state.password);
63        alert("Salvo com sucesso!!!")
64      }catch(erro){
65        alert("Erro!")
66      }
67    }
68
```