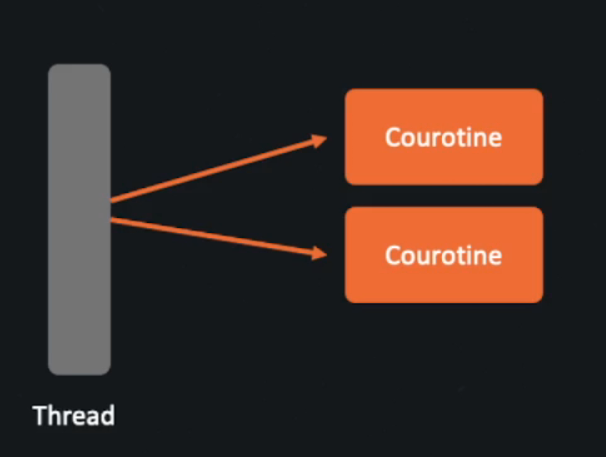


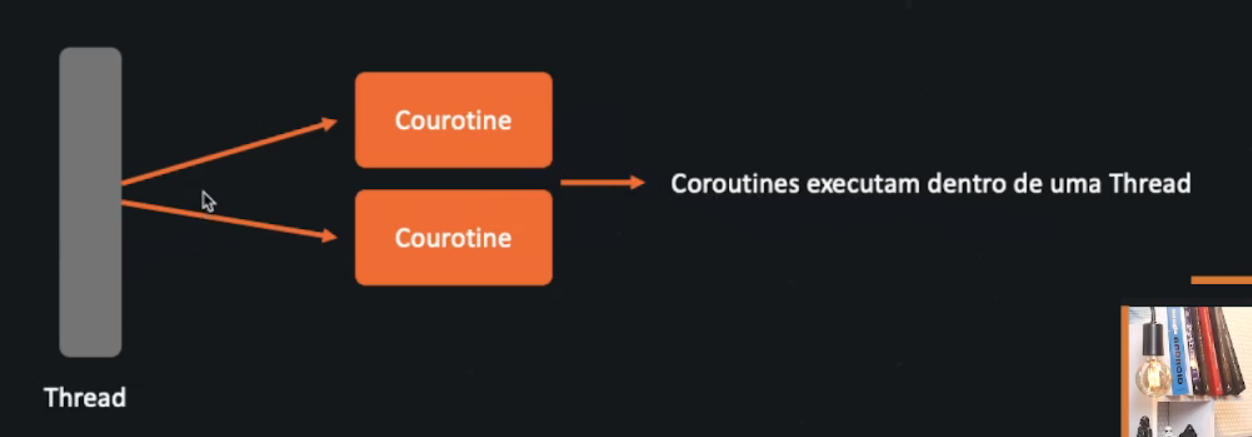
Resumindo lembra das tarefas assíncronas? Que fazemos tudo junto? Então é basicamente isso.

Imagina que vamos processar os postos de combustível do aplicativo, então teríamos que criar uma thread para isso, mas com o coroutines isso fica mais fácil/simples e é nossa melhor opção.

Vantagens de usar a corroutines



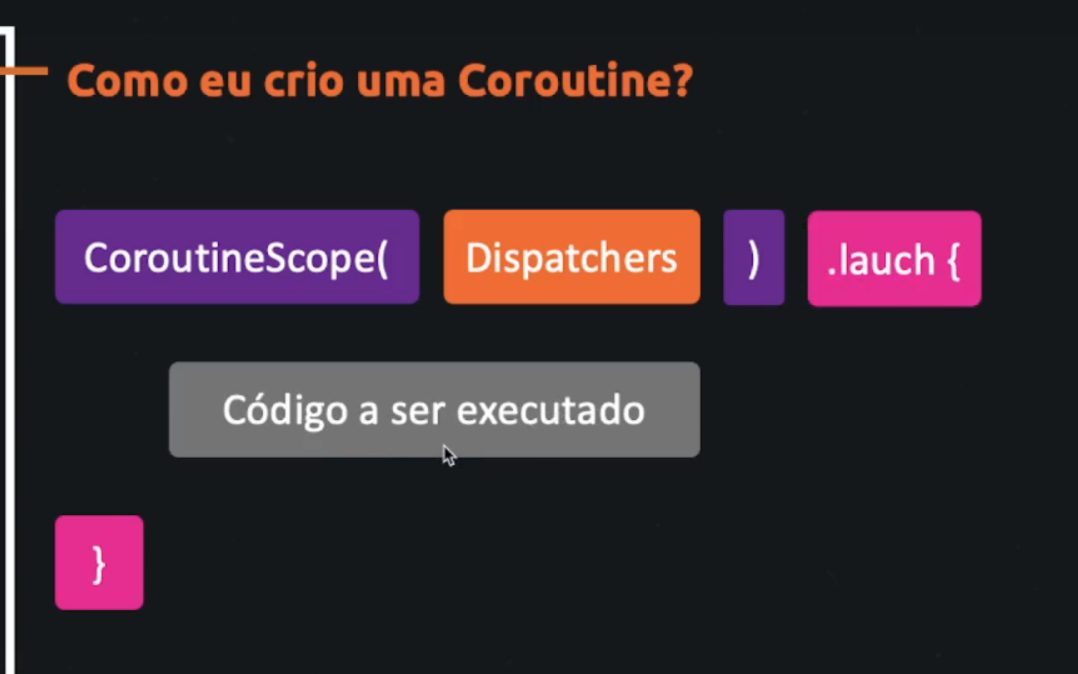
Imagina que vamos criar varias threads para executar varias tarefas diferentes, isso iria trazer um problema de lentidão porque a thread usa bastante recurso, já a coroutine ela faz isso de maneira mais leve/mais otimizada.



Resumindo: criamos uma coroutine e ela própria vai fazer a gestão dessa thread.

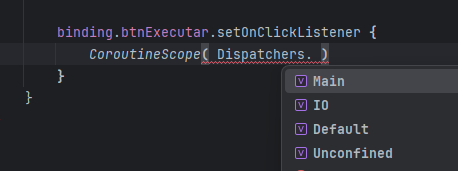
Ou seja, ela pode criar uma nova thread ou usar a mesma, a coroutine vai fazer o uso otimizado de uma thread.

# Como criar uma coroutine



## Dispatchers

Seria em qual contexto vamos executar a coroutine, ex: Qual thread vamos executar, vai ser na principal? Vai ser na thread x? e por ai adiante...



Aqui vamos ter esses 4 padroes, mas podemos criar o nosso também

 **Dispatchers.Main**:

* Usado principalmente para interações de IU (Interface do Usuário).
* Executa corrotinas no thread principal da aplicação Android.
* Ideal para operações rápidas que precisam atualizar a IU.

 **Dispatchers.IO**:

* Projetado para operações de E/S (Entrada/Saída) que podem ser bloqueantes, como leituras/gravações de arquivos ou acessos a banco de dados.
* Executa corrotinas em um pool de threads otimizadas para operações de I/O.

 **Dispatchers.Default**:

* Utilizado para operações que consomem bastante CPU, como cálculos intensivos ou processamento de dados.
* Executa corrotinas em um pool de threads que é ajustado conforme a carga de trabalho.

 **Dispatchers.Unconfined**:

* Inicia corrotinas no thread onde foram chamadas, mas pode se mover para outros threads após suspensão.
* Mais adequado para tarefas leves e não restritas, mas seu uso é menos comum em práticas recomendadas.

# Executando a Coroutine

binding.btnInciar.setOnClickListener **{** *CoroutineScope*( Dispatchers.Main).*launch* **{** */\*\* ATENÇÃO: Essa Coroutine vai rodar dentro da Thread Principal - Dispatchers.Main \*/  
  
 /\*\* Launch: Arremesar / Botar para jogo  
 \* o Launch vai fazer a execução  
 \*/* binding.btnInciar.*text* = "Executou"  
 **}**

Agora vamos executar também a IO para teste, mas sabemos que não podemos colocar o binding para executar na IO, porque ela não consegue atualizar a thread de interface apenas a MAIN que consegue.

## IO

binding.btnInciar.setOnClickListener **{** *CoroutineScope*( Dispatchers.IO).*launch* **{** *repeat*(15) **{**indice **->** Log.i("info\_coroutine", "Executando: $indice - T: ${Thread.currentThread().*name*}")  
 delay(1000) // Tempo que ele vai esperar para executar novamente.  
  
 **}  
  
  
 }  
}**

## Atualizando agora a interface

Repara no código a cima que estamos dentro da Thread IO, mas quando a gente quiser atualizar algo em outra interface, podemos utilizar o contexto e passar qual THREAD queremos atualizar.

Ficando dessa forma a baixo.

binding.btnInciar.setOnClickListener **{** *CoroutineScope*( Dispatchers.IO).*launch* **{** */\*\* Thread para atualizar dados  
 \* Não serve para atualizar interface  
 \* Ela é uma thread secundaria  
 \* Se quiser atualizar interface precisamos da Thread Main  
 \*/  
  
 repeat*(15) **{**indice **->** Log.i("info\_coroutine", "Executando: $indice - T: ${Thread.currentThread().*name*}")  
  
 withContext(Dispatchers.Main) **{** */\*\* Qual contexto queremos atualizar? qual thread?  
 \* Agora aqui dentro vamos executar no contexto da thread principal  
 \* Ela vai ser responsavel por atualizar o codigo a baixo que é atualizar o nome do botao  
 \* Que é uma atualização de VIEW/Interface  
 \*/* binding.btnInciar.*text* = "Executando"

// Aqui dentro desse bloco de código podemos fazer várias atualizações de VIEWS  
 **}** delay(1000) // Tempo que ele vai esperar para executar novamente.  
  
 **}  
  
  
 }  
}**