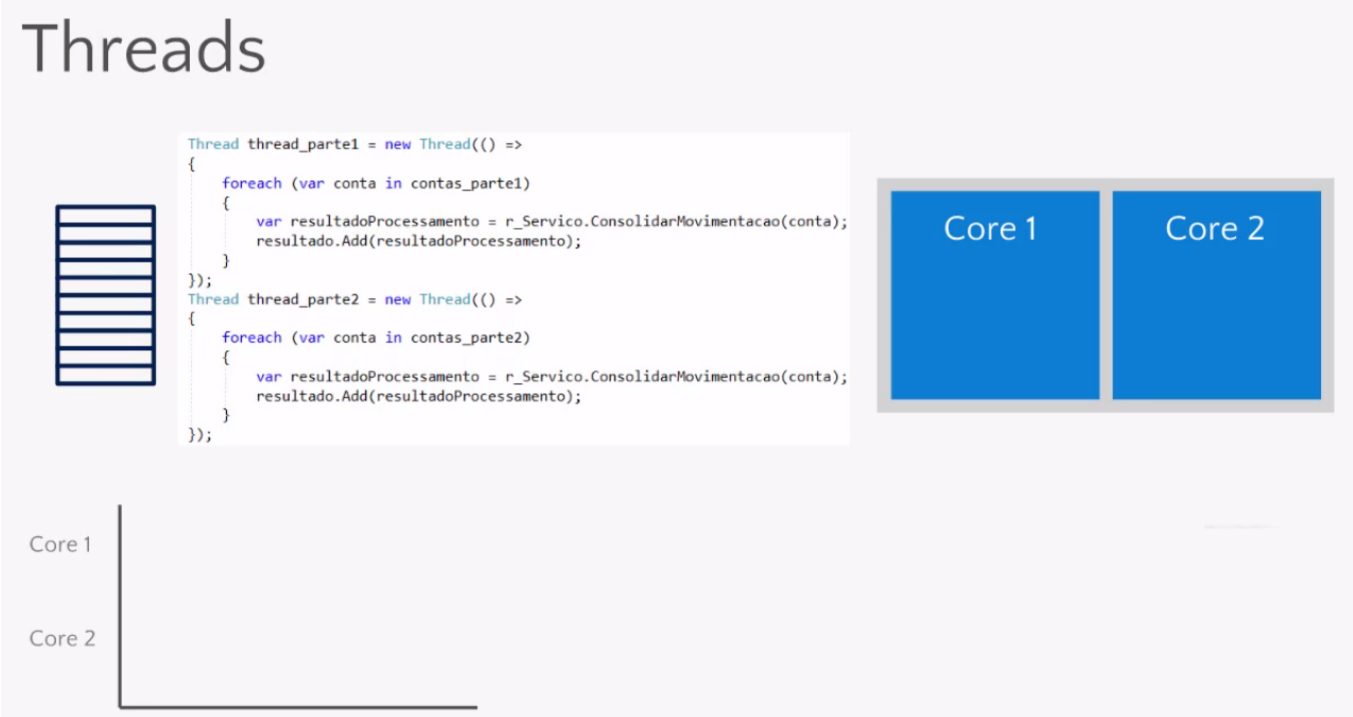
01

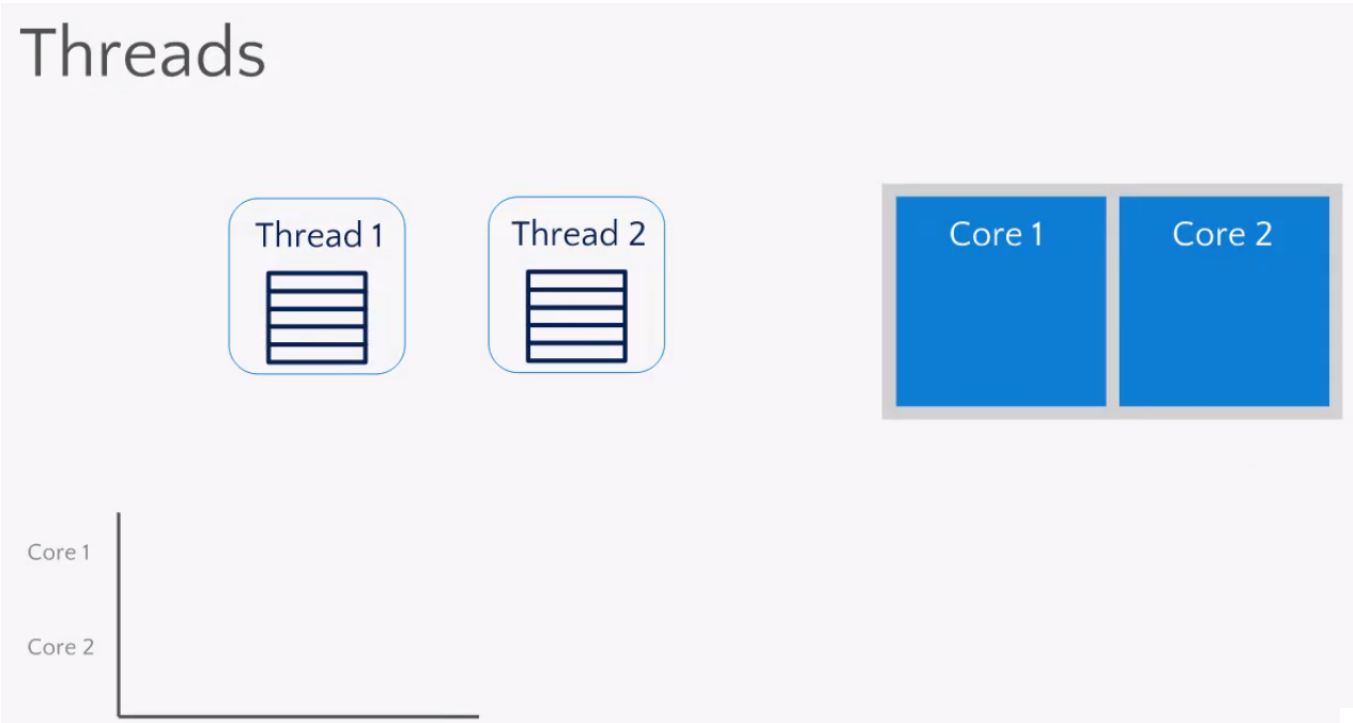
Abstraindo tarefas

Transcrição

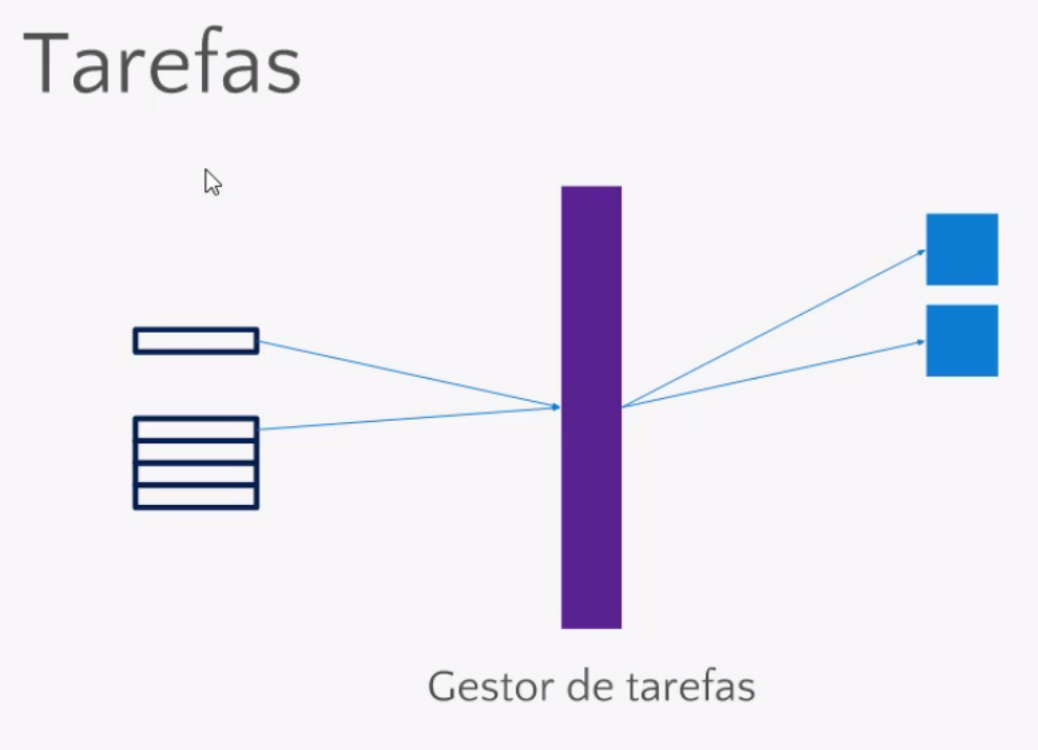
Estávamos discutindo sobre como tornar a aplicação mais rápida fazendo uso do Paralelismo em C# e .NET. Qual é o nosso problema atual? Temos um conjunto de contas no ByteBank, que em nosso exemplo em desenvolvimento são `10`. Colocamos uma pilha composta de dez retângulos representando as contas, e começamos a dividi-los pelos núcleos através do código, delegando para cada *thread* a resolução de cinco contas, metade da lista inteira.



Para a `thread_parte1`, o código delega, então, a execução da primeira parte, e este código, que escrevemos no vídeo anterior, vai resolver a segunda metade. Com isto, conseguimos executá-las simultaneamente.



Apertamos "*Start*" na `thread_parte1`, e o sistema operacional escolhe um dos *cores* do processador para fazer sua execução. Faremos o mesmo com a `thread_parte2`, e o sistema operacional arranja outra *thread*, ou a mesma, dependendo do estado do computador, executando-se ambas ao mesmo tempo. Notamos que isto aumenta a performance consideravelmente.



No entanto, queremos usar mais de nossa máquina, pois ela possui oito *cores*. Portanto, começamos a utilizar quatro, sendo que o trabalho é dividido a partir da quantidade de contas, quebrando-a para definirmos a porção que cada *thread* resolverá. De que maneira fazemos esta divisão? Pela divisão do total da contagem das contas por quatro.

O *count* de uma lista retorna um número inteiro. Dividindo-se por outro número inteiro, o resultado também será um número inteiro. `10` dividido por `4` não será `2,5`, e sim `2`. Sendo assim, a primeira *thread* resolveu `2`, a segunda pulou a parte anterior e pegou outras `2`. A terceira pulou as partes das outras, pegando outras `2`. Até aí, o total pego foi de `6`, e a última *thread* fica com a restante, que são `4`. Desta maneira, uma das *threads* ficou com mais trabalho que as outras, porque não percebemos este *bug*.

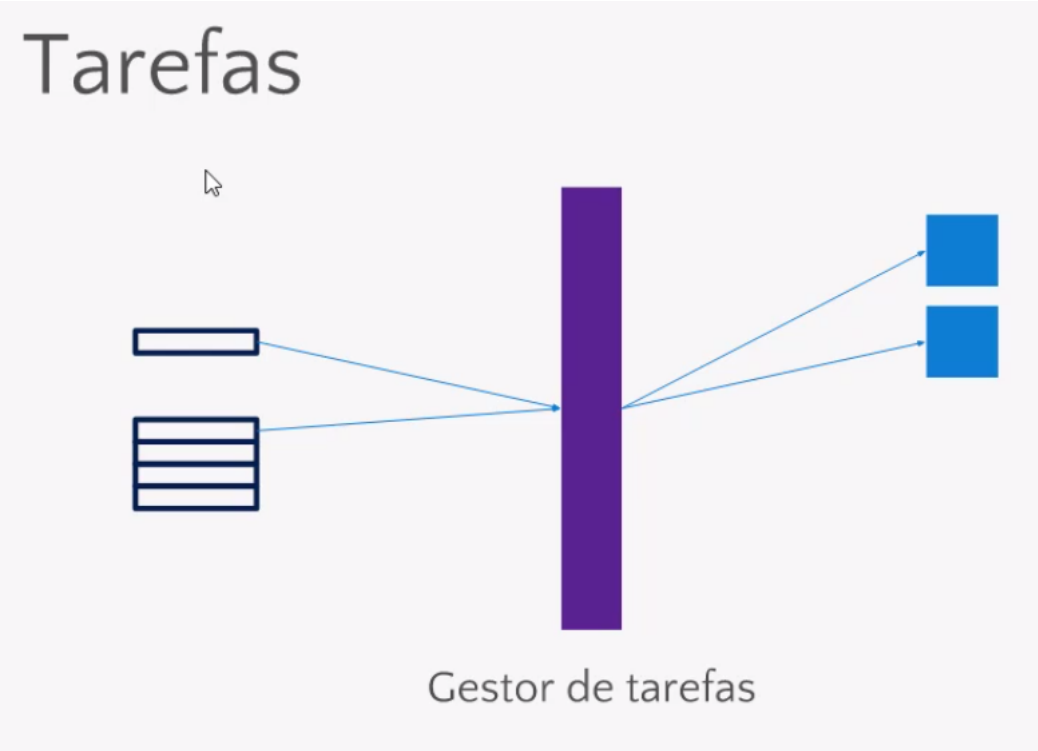
Resolveríamos isto criando uma quinta *thread*, sexta, sétima, oitava. Vocês se lembram de quantos lugares teríamos que alterar no código para conseguirmos incluir mais *threads*? Vamos relembrar: precisamos criar uma nova porção na lista de contas para cada parte, alterando-se também o divisor, a contagem de quantas partes seriam usadas, teríamos que criar nova *thread*, iniciando-se uma nova, ficando-se no laço de repetição...

Ou seja, são muitas alterações no código! Há maneira melhor de resolvermos isto, e é o que veremos neste vídeo. Vamos abstrair esta ideia de *threads, cores*, processador, e pensar naquilo que realmente precisamos fazer: temos várias tarefas, e elas precisam ser resolvidas.

Nossa tarefa é consolidar a conta de um cliente e, no caso, são `10`. Em vez de especificarmos sua execução à *thread* `1`, fazendo o mesmo em relação à *thread* `2`, ou seja, delegando funções para cada uma delas, por que não damos esta responsabilidade para algo intermediário, um gestor de tarefas? Assim, ele receberá as tarefas a serem realizadas, com uma quantidade de funcionários, trabalhadores (no caso do computador, será uma *thread*). De acordo com o número de trabalhadores que ele possui, ele vai manter todos trabalhando, resolvendo as tarefas.

Se temos apenas um trabalhador, o gestor de tarefas enviará todas as tarefas para ele. Conforme o trabalho necessário, o gestor vai enviando aos trabalhadores (ou *threads*). Abaixo, temos uma situação com uma tarefa pesada. Então, enquanto

uma *thread* fica trabalhando nela, não teremos problemas, pois o gestor irá delegar a execução de outras tarefas para outras *threads*.



De tal forma que não precisaremos mais nos preocupar se temos duas, três, quatro ou oito *cores* no processador da máquina, bastando que nos atentemos em delegar estas tarefas ao gestor. Vamos lá?