

DISCIPLINA

Introdução à Computação Paralela - Conceitos Importantes de Hardware e Software-

Prof. Kayo Gonçalves

BACHARELADO EM TECNOLOGIA DA INFORMAÇÃO

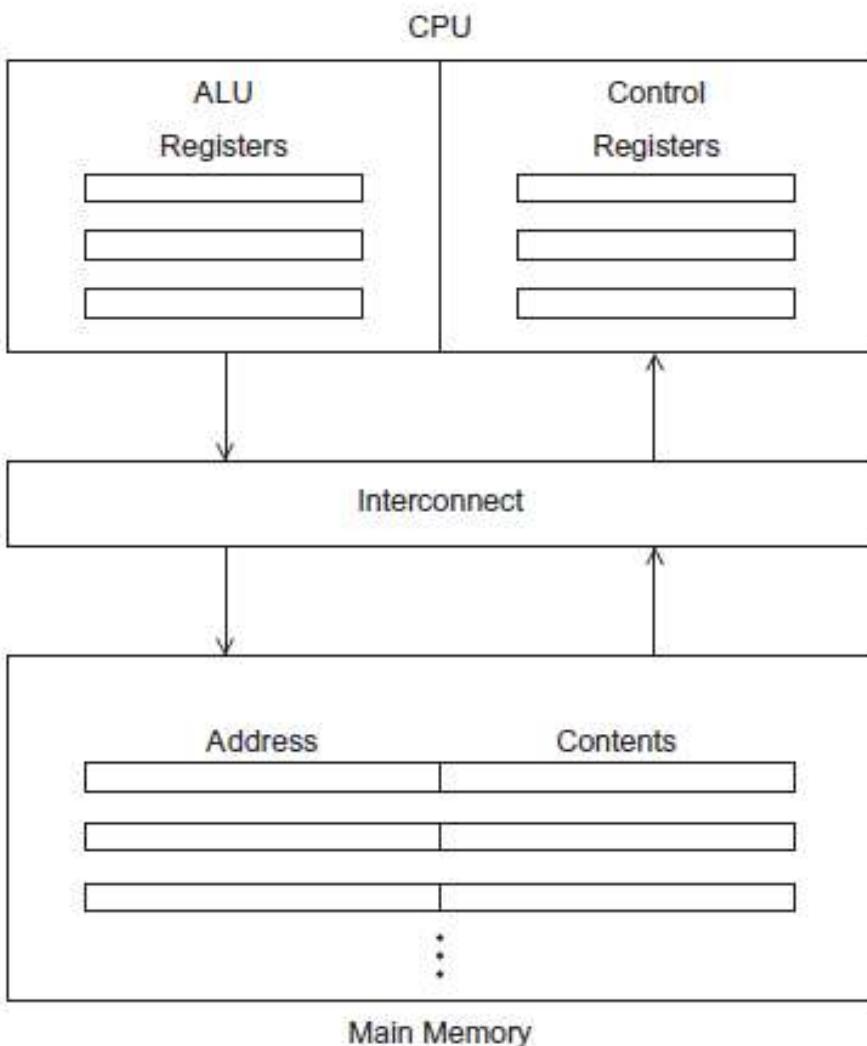


Adaptado do livro “An Introduction to Parallel Programming” do Peter Pacheco



Execução Serial

Computador foi concebido
inicialmente para executar um
programa por vez



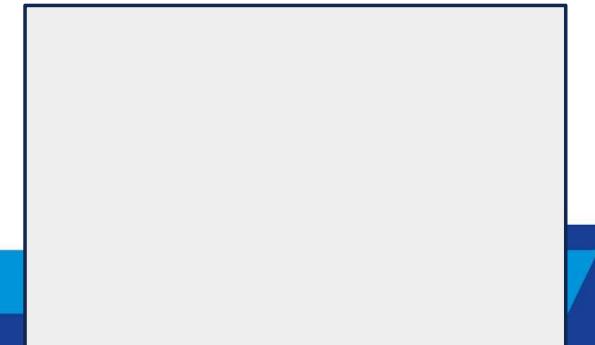
Arquitetura de von Neumann

Unidade Central de Processamento CPU

- Dividido em duas partes.

Unidade de controle – responsável por decidir qual instrução em um programa deve ser executada. (o chefe)

Unidade aritmética e lógica (ALU) - responsável pela execução das instruções reais. (o trabalhador)

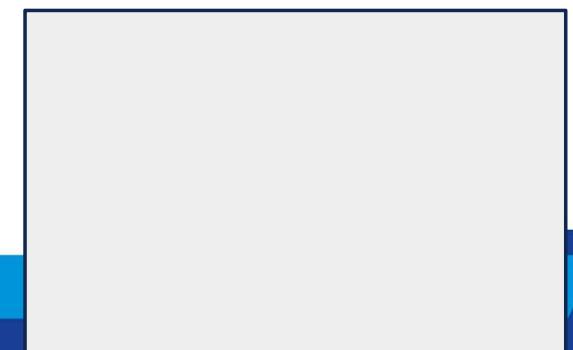


Fetch / Read / Leitura

CPU

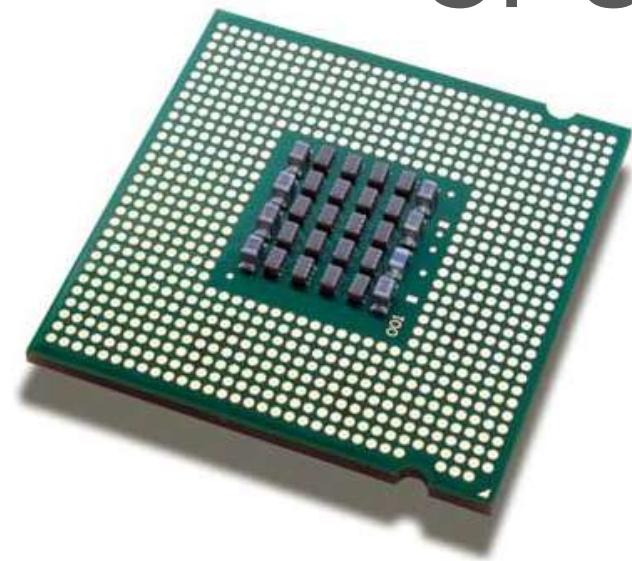


Memória

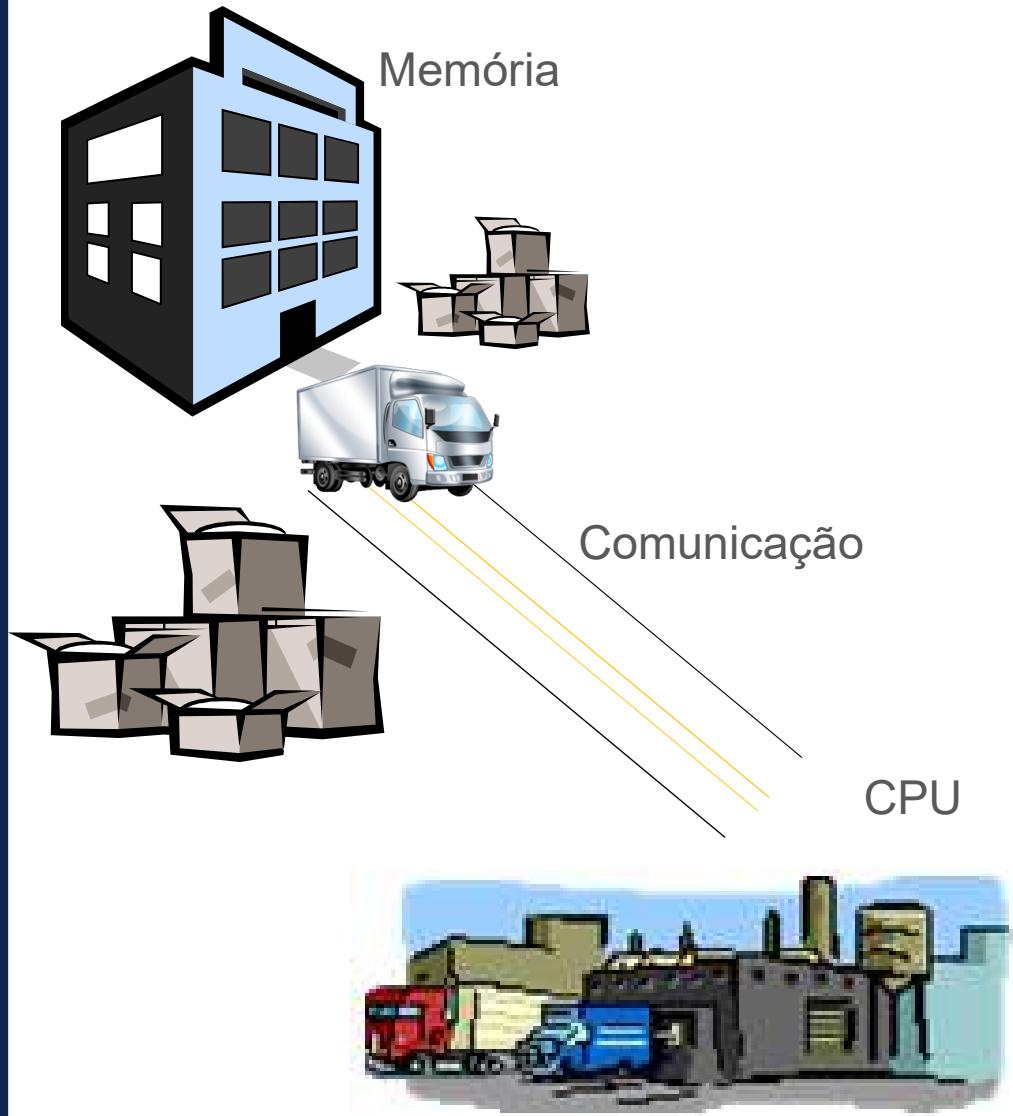


write / Store / Escrita

Memória



CPU



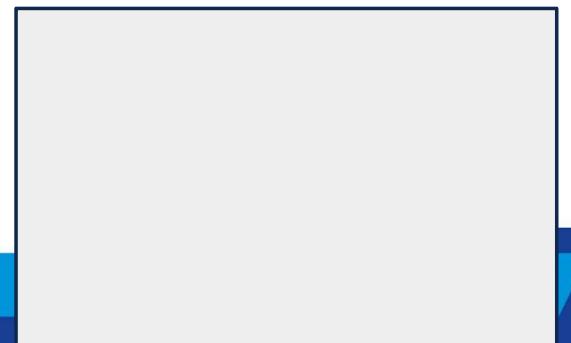
Gargalo de von Neumann

Alto processamento, baixa troca de dados.

Multitarefa e Threads

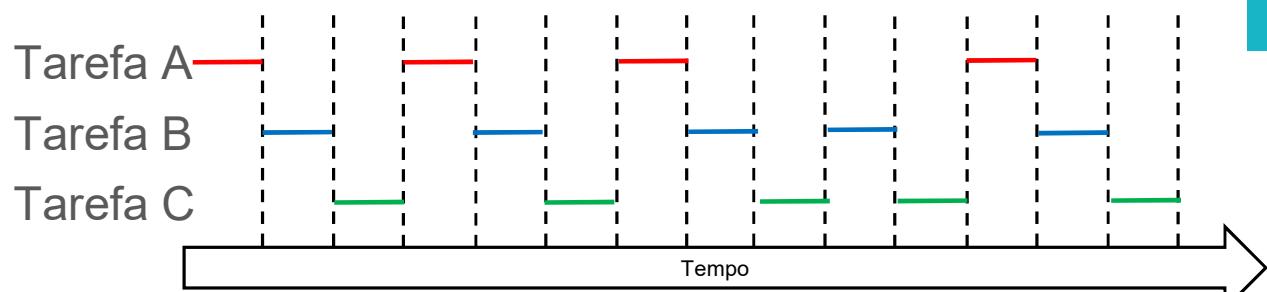
Um “processo” do SO

- Uma instância de um programa de computador que está sendo executado.
- Componentes de um processo:
 - O programa de linguagem de máquina executável.
 - Um bloco de memória.
 - Descritores de recursos que o SO alocou para o processo.
 - Informação segura.
 - Informações sobre o estado do processo
 - Outros



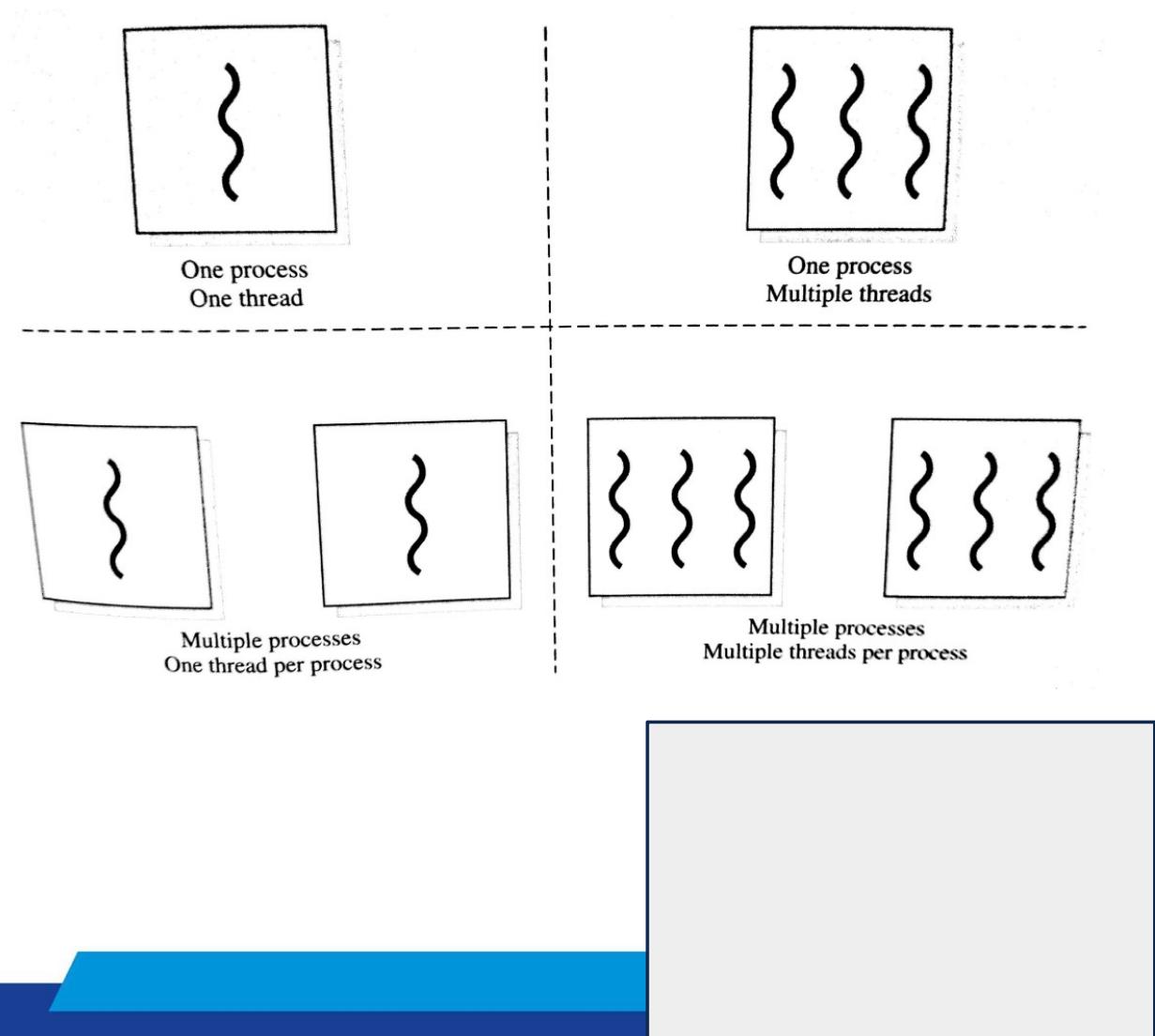
Multitasking / Multitarefa

- Dá a ilusão de que um sistema de processador único está executando vários programas simultaneamente.
- Os processos revezam em execução. (parte do tempo / time slice)

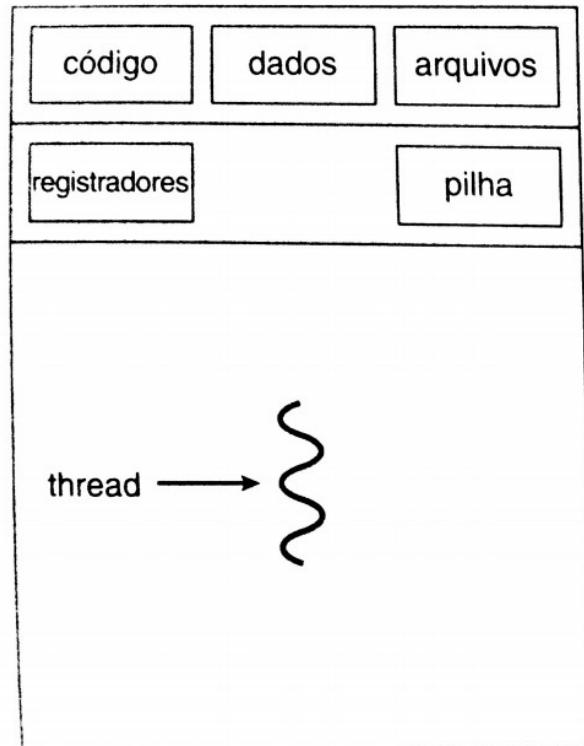


Threading

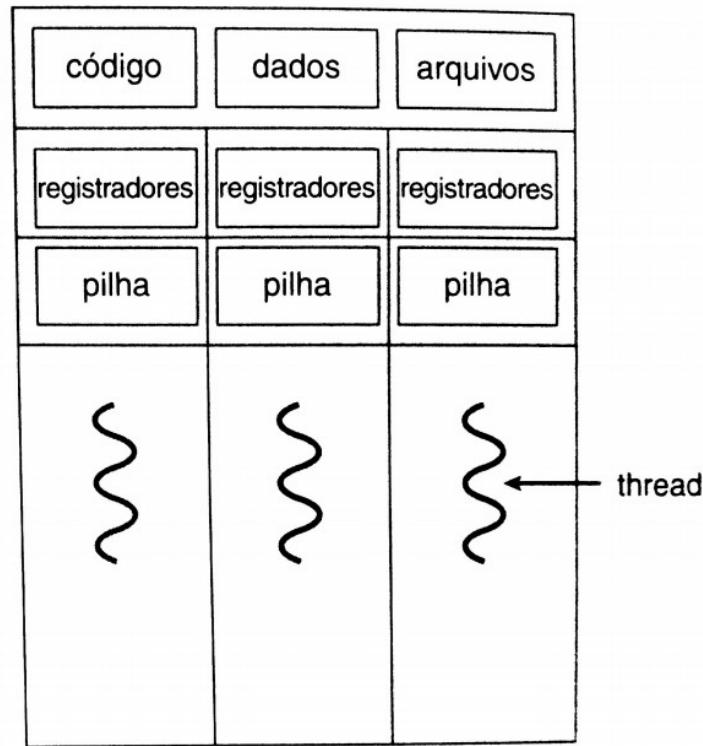
- Threads (linha de execução) estão contidos num processo.
- Permite que os programadores dividam seus programas em (mais ou menos) tarefas independentes.
- A esperança é que, quando uma thread bloqueia, outra poderá executar.



Threading



processo com um único thread

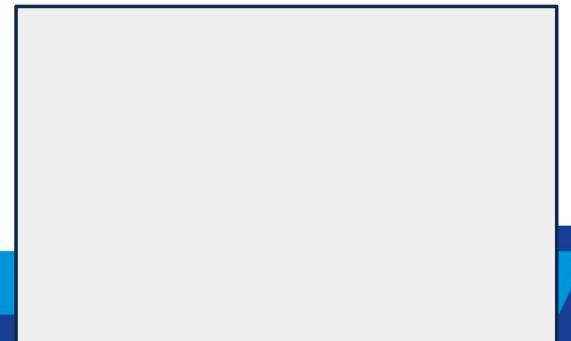


processo com vários threads

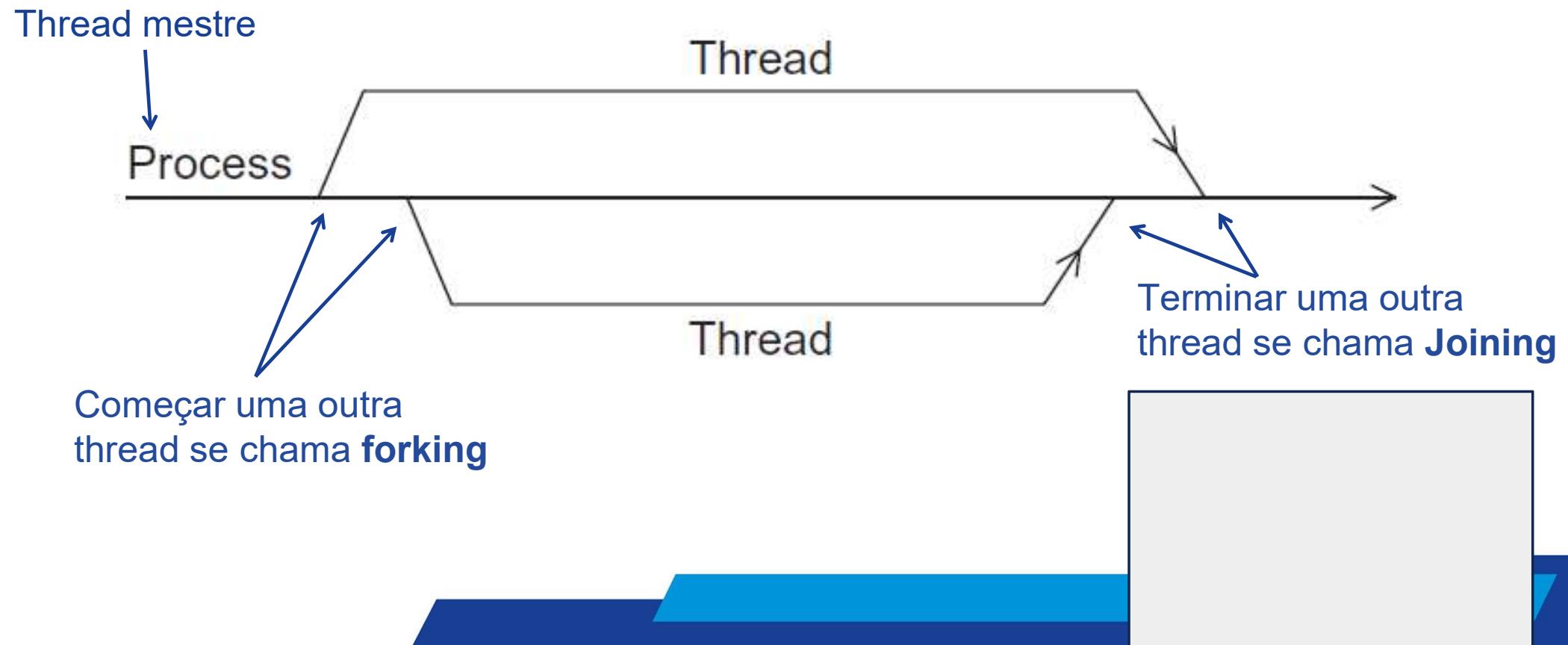


Por que usar Threading?

- Alivia a carga de criação (100x), gerência, destruição e mudança de contexto
- Informações facilmente compartilhadas
- Conceito de Thread
 - Pilha + PC + Registradores de uso geral
 - **Abstração similar a processos**
 - Comunicação via variáveis globais

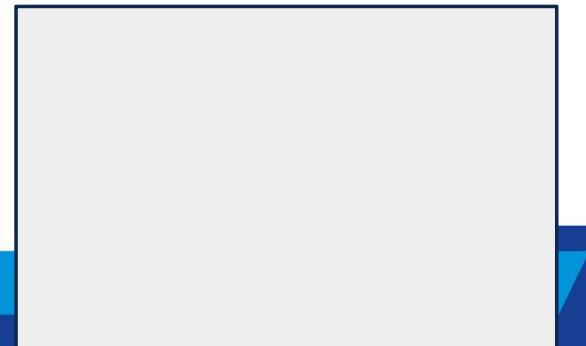


Um processo, três threads



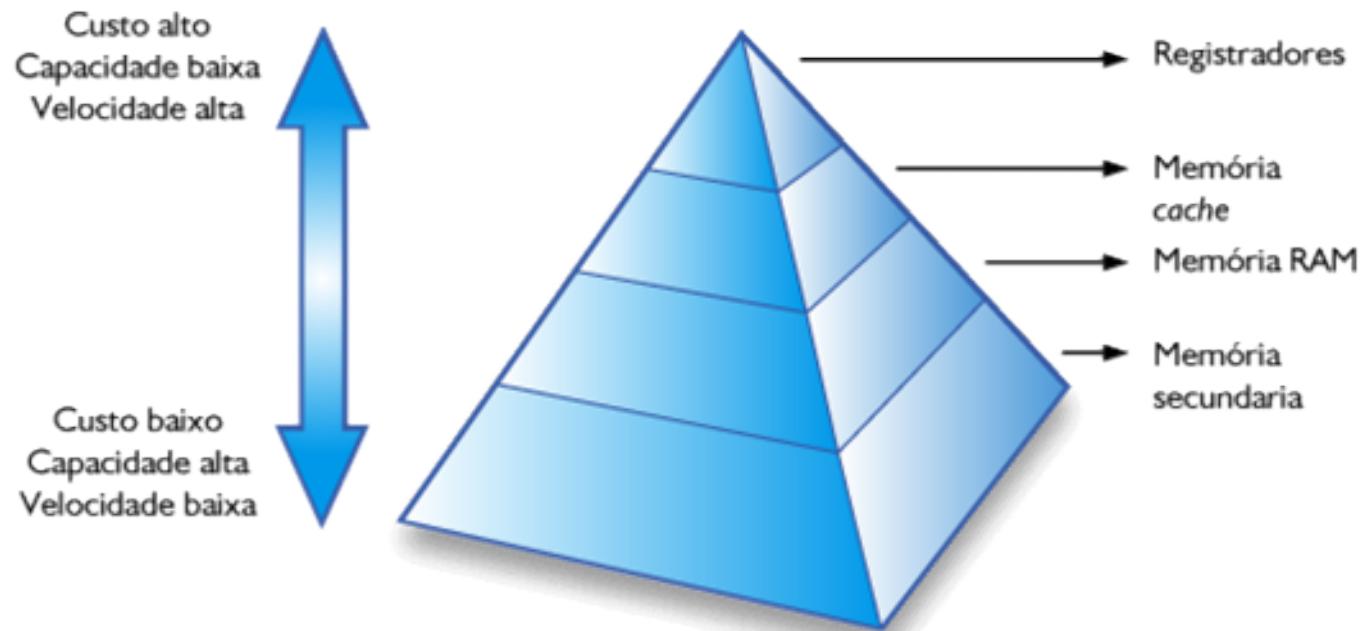
Onde queremos chegar?

- Mais threads do que cores:
 - Situação indesejada.
 - Pode reduz desempenho (disputa do processador entre as threads)
- Menos threads do que cores:
 - Situação indesejada.
 - Poder computacional desprezado
- Número igual de threads e cores
 - Situação ideal



Hierarquia de Memória

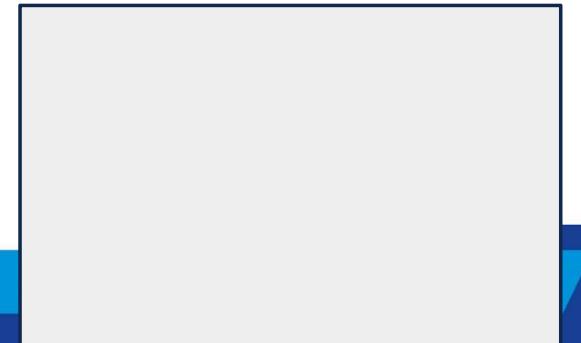
Hierarquia de Memória



Fonte: curso técnicos IMD - UFRN, acesso
em 20/07/2020

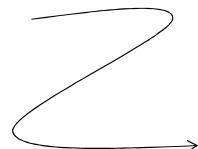
Noções de Memória Cache

- Uma coleção de locais de memória que podem ser acessados em menos tempo do que alguns outros locais de memória.
 - Cache L1, L2 e L3
- Um cache da CPU normalmente está localizado no mesmo chip ou em um que pode ser acessado muito mais rápido que a memória comum.



Níveis de cache

menor & mais rápida



L1

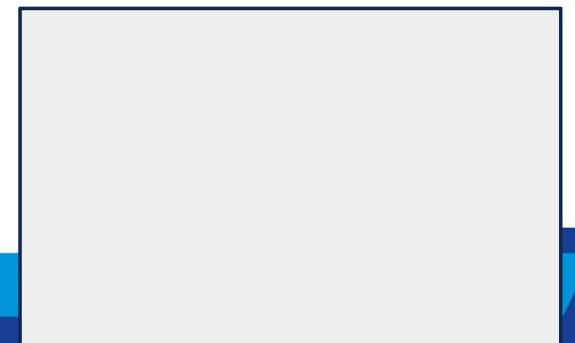


L2

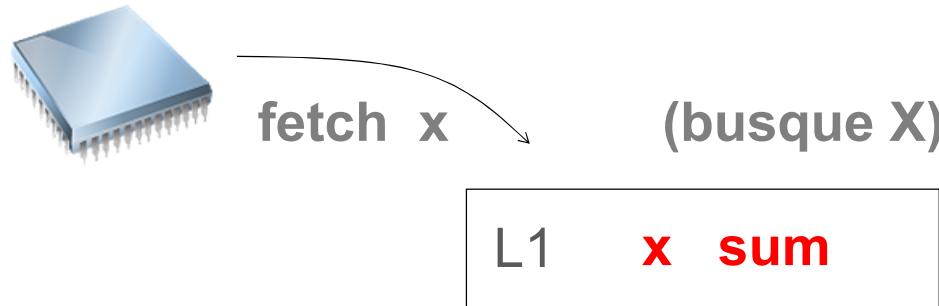
L3



maior & mais lenta



Cache hit (acerto)

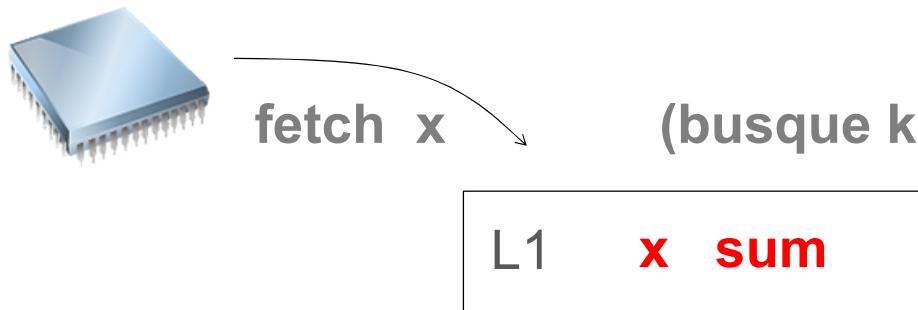


L3 **x sum y z A[] radius r1**

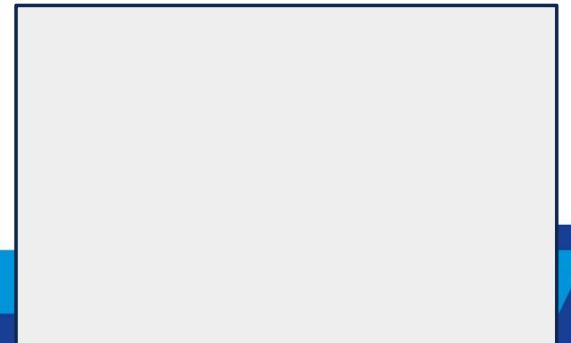
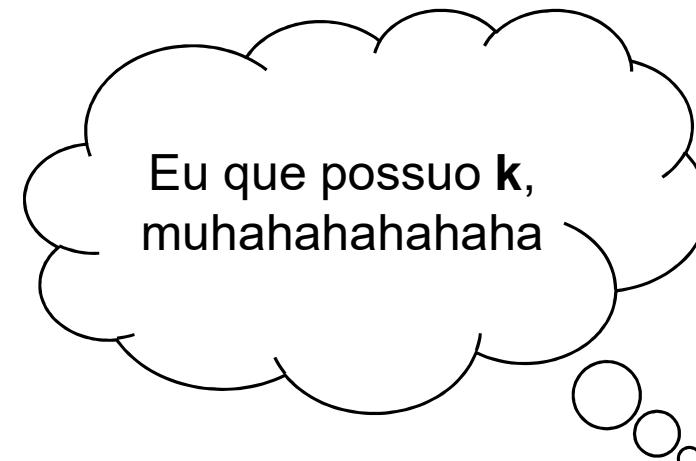


IMPORTANTE: Nível superior de memória contém uma porção do conteúdo de um nível inferior.

Cache miss (erro)



IMPORTANTE: Um conjunto de dados, o que inclui o **k**,
será copiado.



Princípio da Localidade

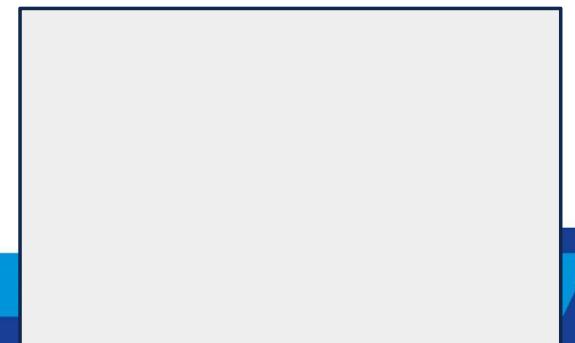
Relativo a tempo

O acesso a um local é **seguido** pelo acesso a um **local próximo**.

Relativo a espaço

Localidade espacial - acessando um local próximo.

Localidade temporal - acessando em um futuro próximo.



```
float z[1000];  
...  
sum = 0.0;  
for (i = 0; i < 1000; i++)  
    sum += z[i];
```

Princípio da Localidade

Maior exemplo!

Estratégias de gravação da cache

Quando uma CPU grava dados no cache, o valor no cache **pode ser inconsistente** com o valor na memória principal.

Caches Write-through tratam disso atualizando os dados na memória principal no momento em que são gravados no cache.

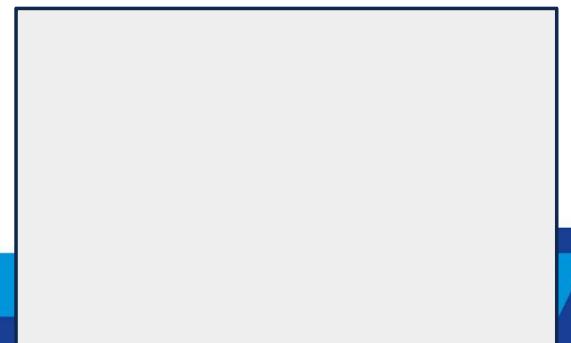
Caches Write-back marcam os dados no cache como sujos. Quando a linha de cache é substituída por uma nova linha de cache da memória, a linha suja é gravada na memória.



Memória Virtual

Problema

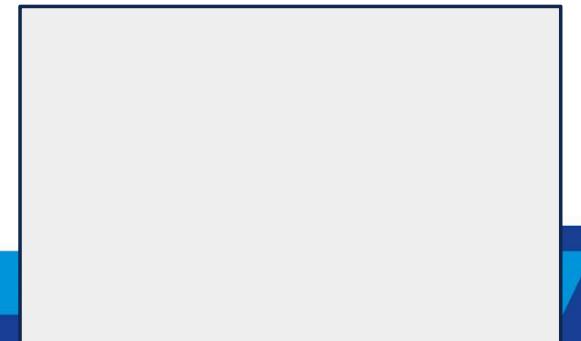
- Se executarmos um programa **muito grande** ou um programa que acesse conjuntos de dados muito grandes, todas as instruções e dados **poderão não caber na memória principal**.
- Como ter memória principal “**INFINITA**”?
Basta usar a memória secundária (HD/SSD) como extensão da memória principal (RAM)



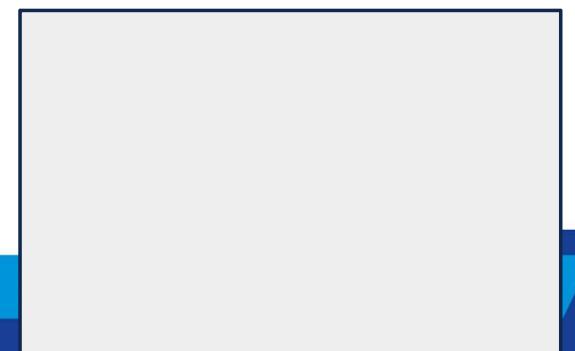
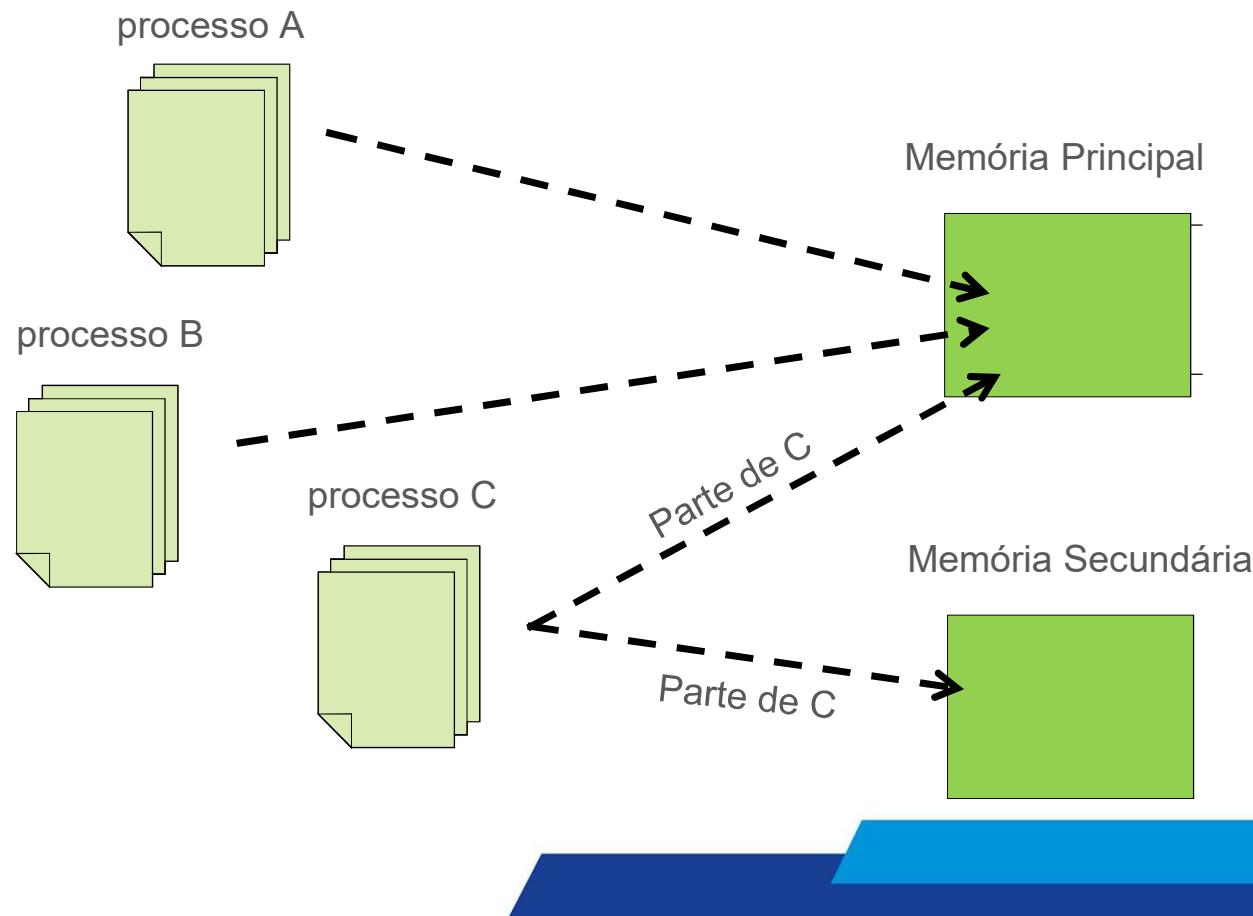
Solução

- Serve como **extensão** da memória RAM.
- Dá a **ilusão** ao programador que a RAM é “infinita”.

Ao infinito
e além!

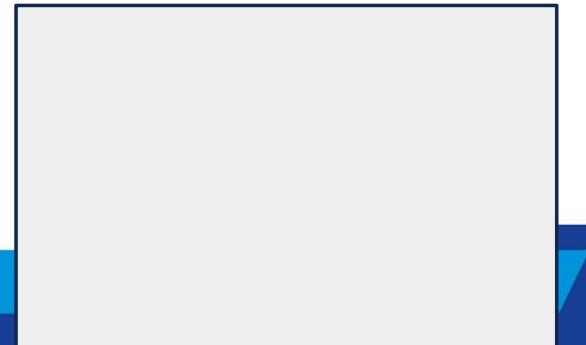


Memória Virtual



Características da MV

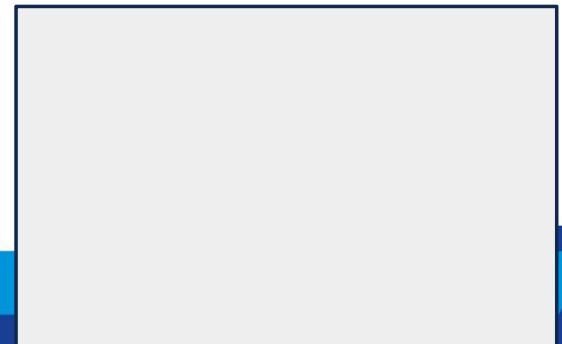
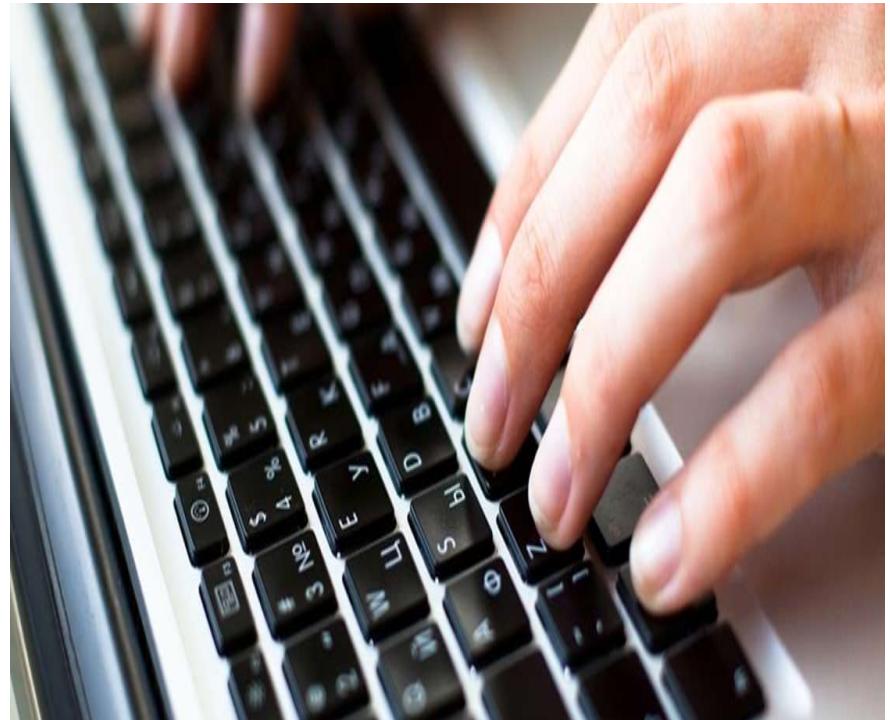
- Explora o princípio da localidade espacial e temporal.
- Ela apenas mantém as partes ativas dos programas em execução na memória principal.
- **Processamento paralelo NÃO DESEJA usar memória virtual**
 - Acesso a disco secundário (HD) é lenta;
 - Reduz desempenho do algoritmo paralelo;

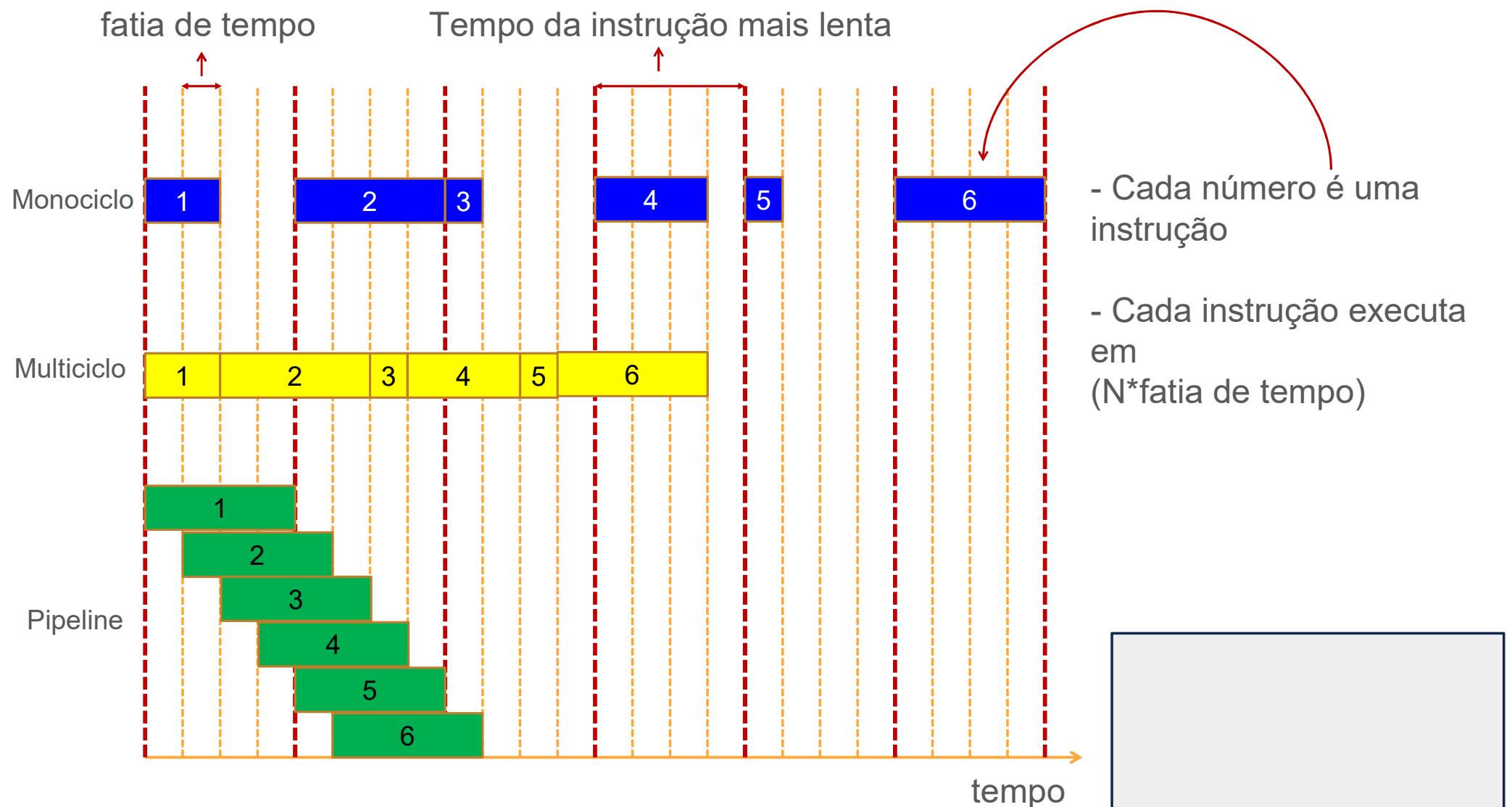


Tipos de Processamento

Premissas

- Um processador possui **diversas** instruções (comandos) para executar
- Instruções **podem** ter tempos de execução **diferentes**
- Para facilitar o entendimento, vamos considerar que **uma instrução** é completamente executado em **múltiplas fatias de tempo**
 - Uma fatia de tempo possui um **tempo constante**

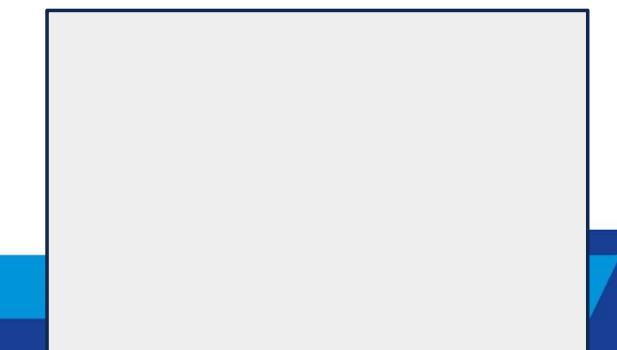




Pipeline

O que é Pipeline?

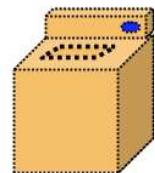
- É uma solução que visa melhorar o desempenho do processador, com vários componentes ou unidades funcionais, executando instruções simultaneamente
- Unidades funcionais são organizados em etapas
- Pipeline já é um paralelismo em nível de hardware



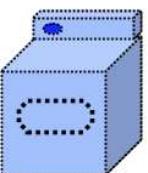
Exemplo de Pipeline

Lavanderia

4 pessoas (A, B, C, D) tem que lavar, secar e dobrar



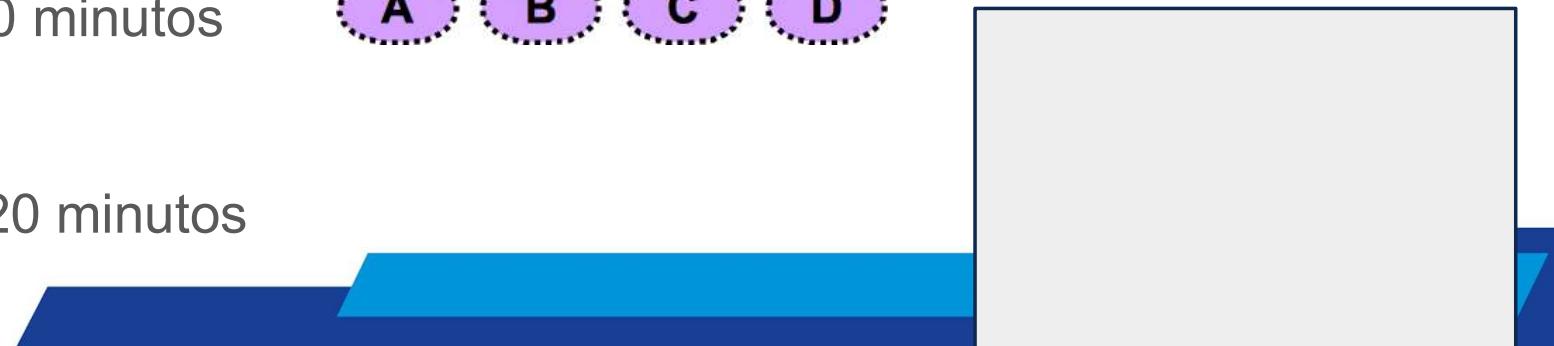
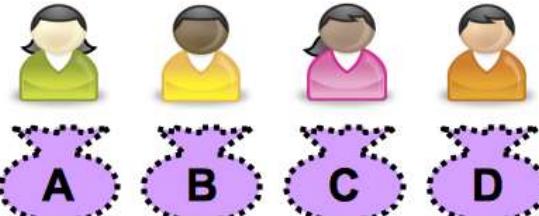
Lavar: 30 minutos

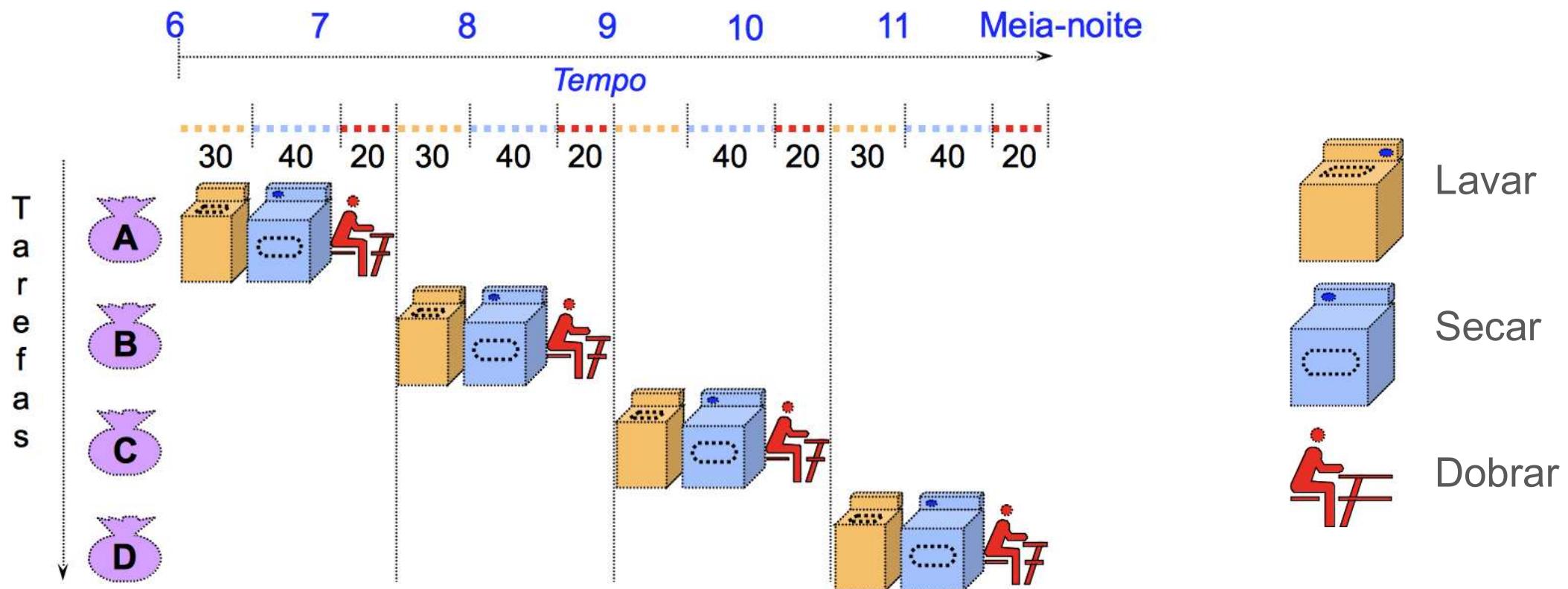


Secar: 40 minutos

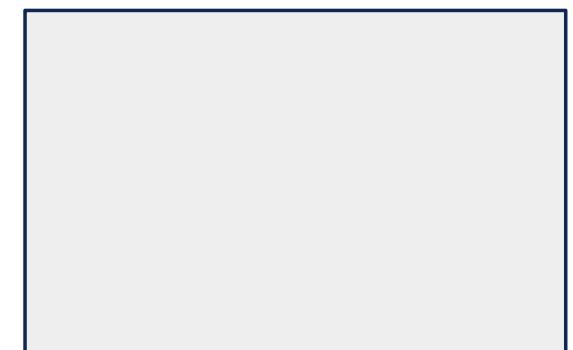


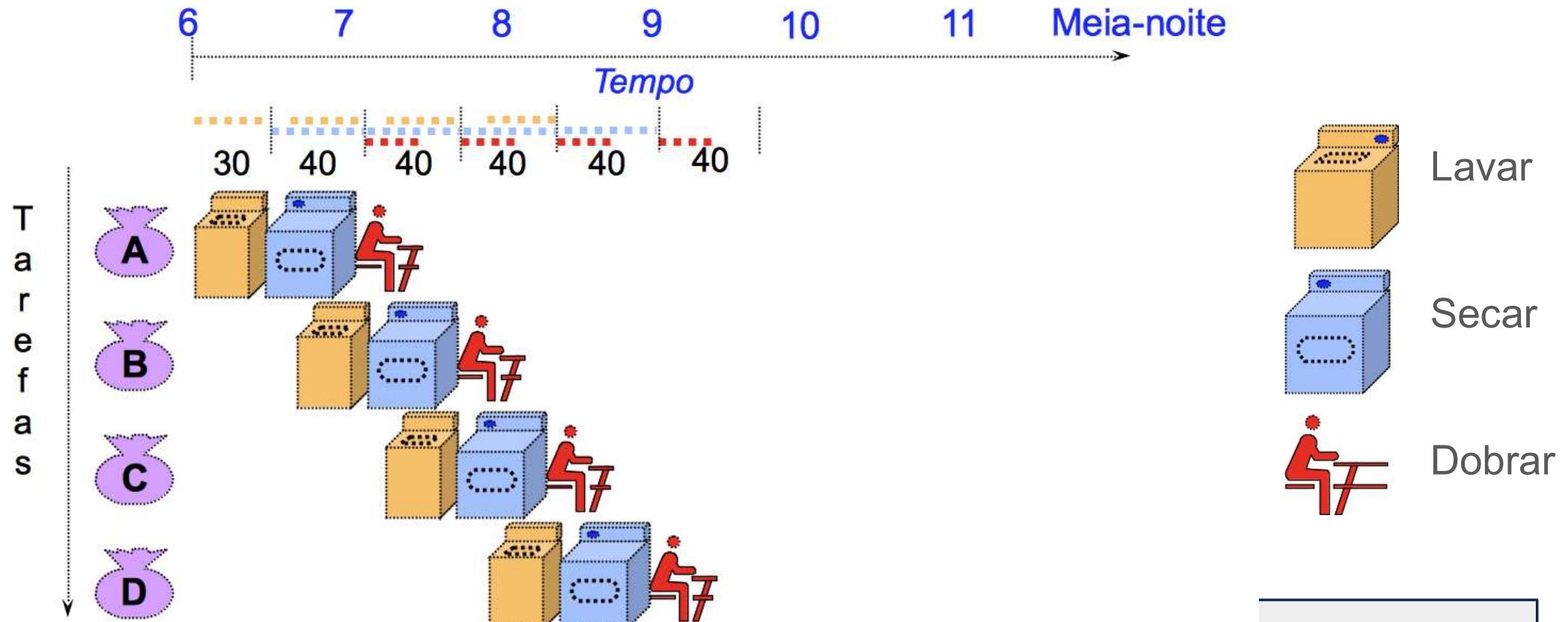
Dobrar: 20 minutos





6 horas sem pipeline



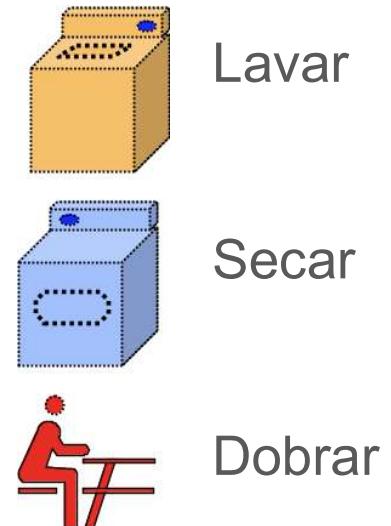


3h50 com pipeline

Informações & Conclusões

- Latency (Latência): tempo de executar **uma instrução (uma lavagem+secagem+dobra)** completamente ~90 minutos (30+40+20)

- Throughput (vazão): instruções/tempo
Sem pipeline: 0.66 roupas / hora (4 roupas / 6 horas)
Com pipeline: 1.04 roupas / hora (4 roupas / 3.83 horas)

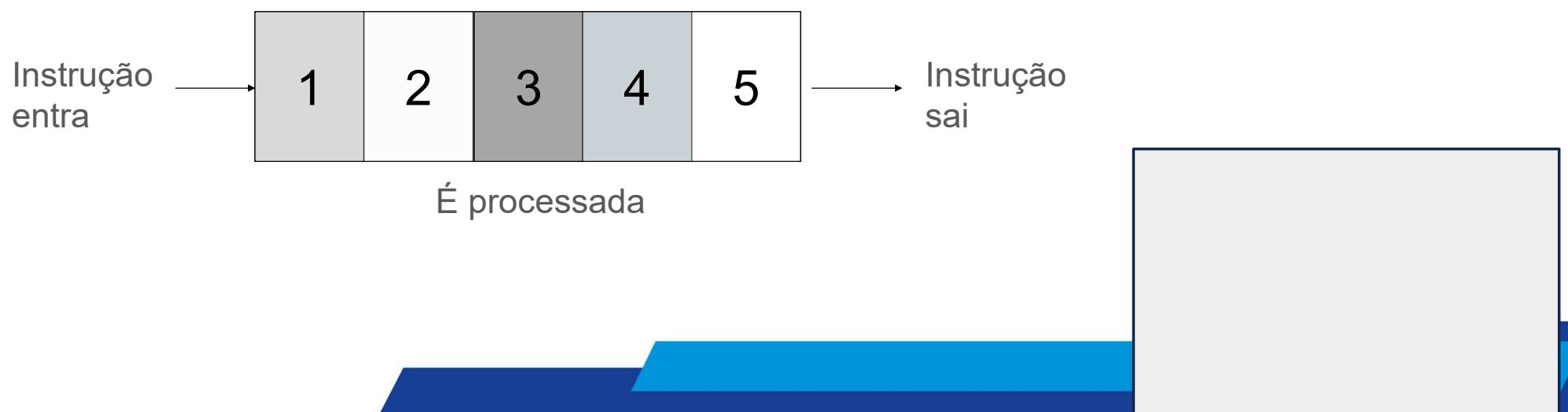


!!!!!! **PIPELINE NÃO MUDA LATÊNCIA** !!!!!!

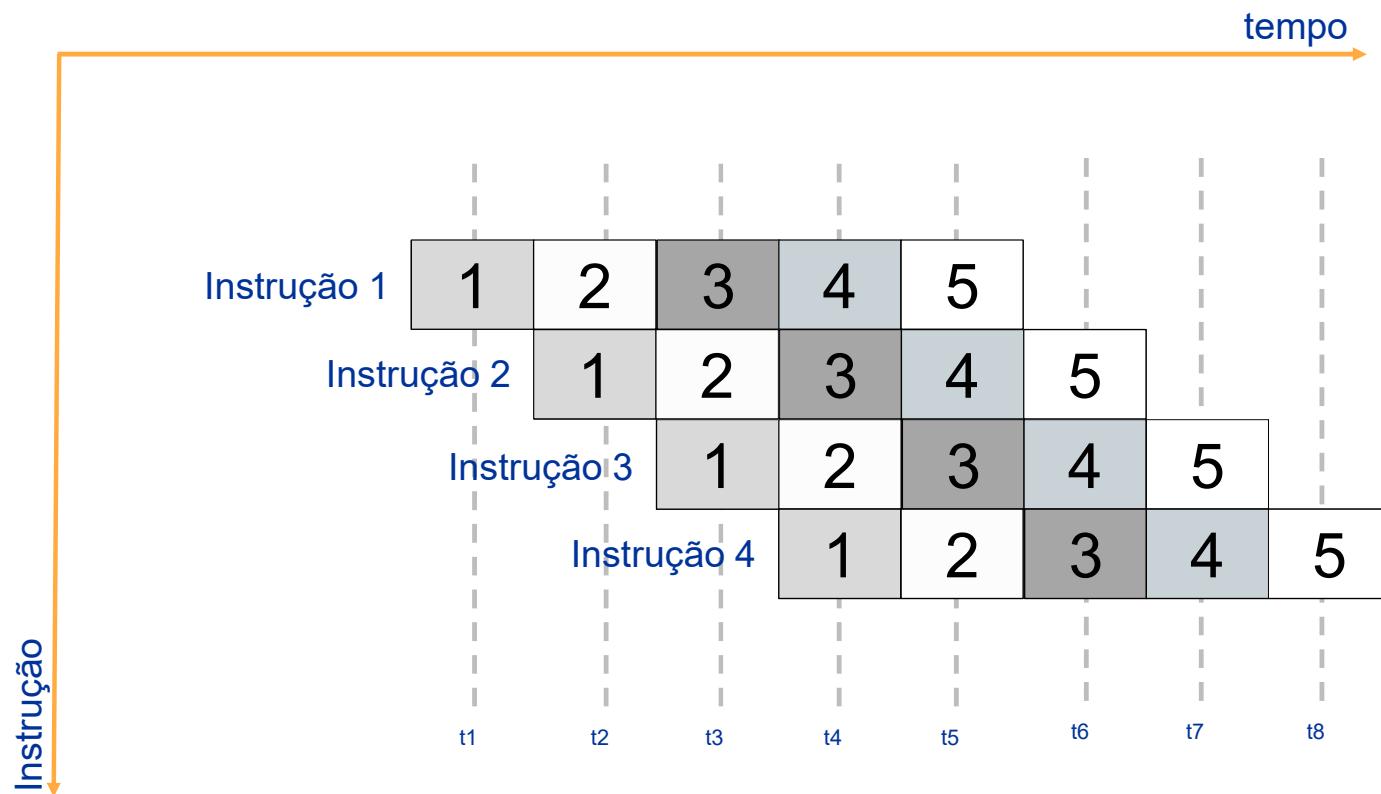
!!!!!! **PIPELINE AUMENTA VAZÃO** !!!!!!

Pipeline clássico

- [1] Busca da instrução
- [2] Decodificação da instrução
- [3] Busca dos operandos
- [4] Execução
- [5] Armazenamento do resultado



Pipeline clássico

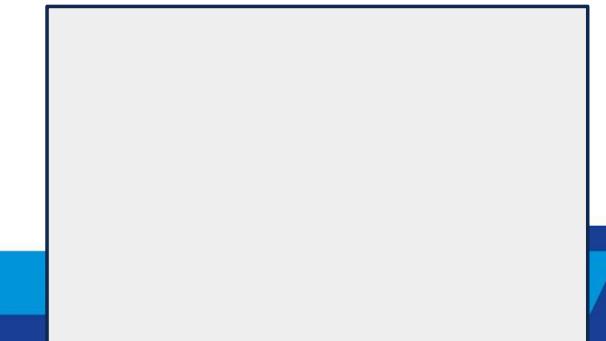


Todos os estágios do pipeline possuem o mesmo tempo. Ou seja...
 $(t_2 - t_1) = (t_3 - t_2) = (t_4 - t_3) = \dots$



Na prática

- Um Pipeline pode estar executando instruções de uma mesma thread ou de diferentes threads, seja do mesmo processo ou de diferentes processos.
- Uma instrução pode ser substituída por outra de qualquer outra thread em qualquer estágio do pipeline.
 - Chama-se Troca de contexto
 - Feito pelo Sistema Operacional (SO)



Conclusões

- Troca de contexto consome tempo e, portanto, reduz eficiência e speedup do algoritmo paralelo.
 - Troca de contexto pode gerar acessos à memória principal e, consequentemente, mais perda de desempenho.
- Dê condições para que o SO reduza ao máximo trocas de contexto.
 - **Utilize n° de cores = n° de threads**



Taxonomia de Flynn

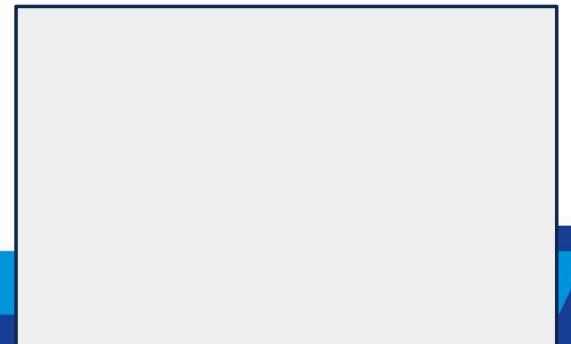
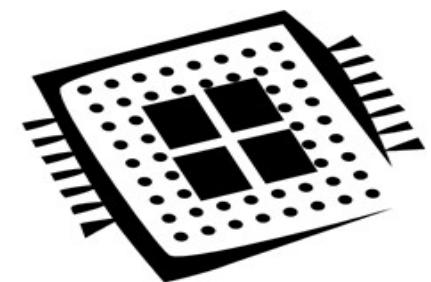
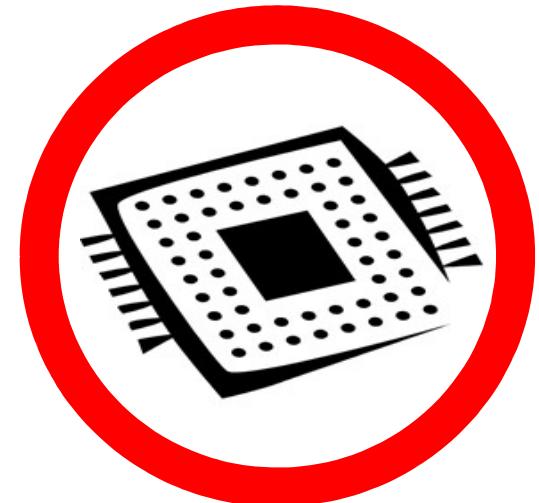
Taxonomia de Flynn

<i>clássico von Neumann</i>	<i>Processadores Vetoriais</i>
SISD Single Instruction stream Single Data stream	SIMD Single Instruction stream Multiple Data stream
MISD Multiple Instruction stream Single Data stream	MIMD Multiple Instruction stream Multiple Data stream
<i>Não coberto</i>	



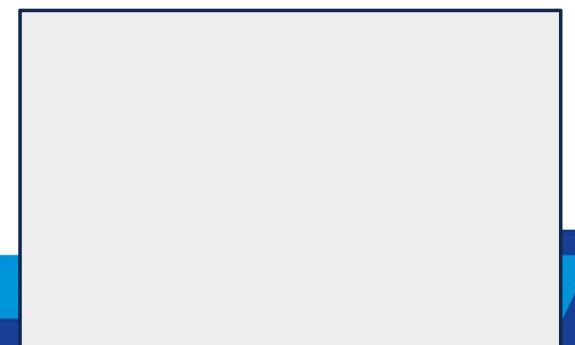
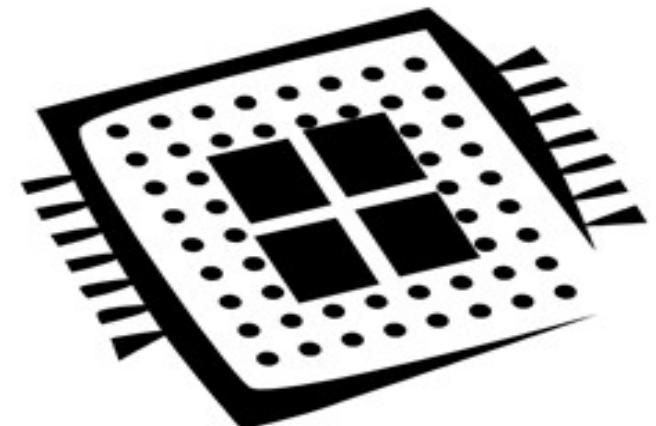
SISD

- Processador singlecore
- Um processador somente executa uma instrução por vez sobre um único dado
- Encontrado em Sistemas embarcados



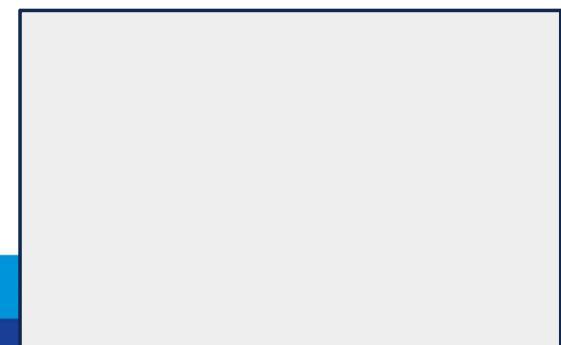
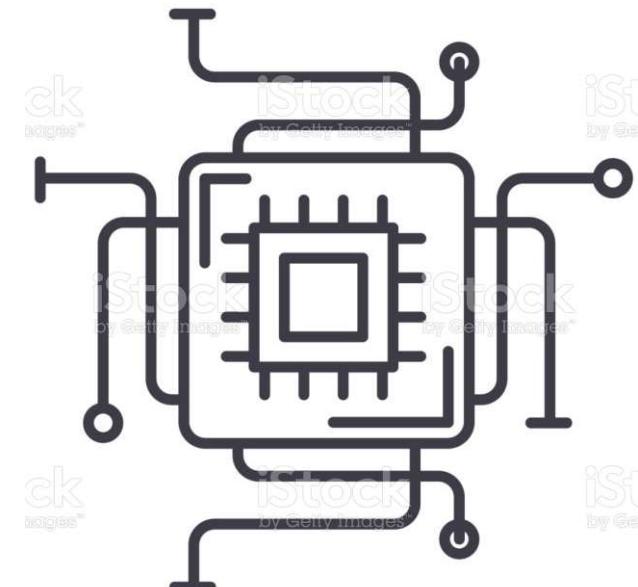
SIMD

- Paralelismo alcançado pela divisão de dados entre os cores.
- Aplica a **mesma instrução** a vários itens de dados diferentes.
- Chamada **paralelismo de dados**.



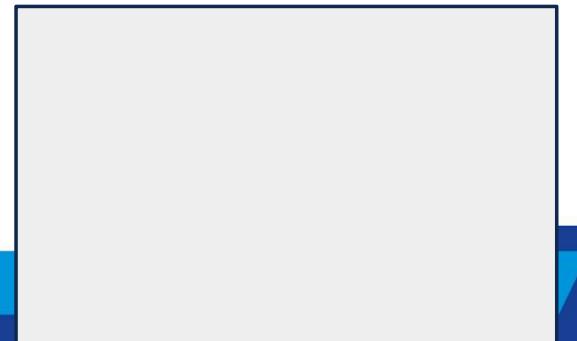
SIMD - Processadores Vetoriais

- Opere em matrizes ou vetores de dados, enquanto as CPUs convencionais operam em elementos de dados ou escalares individuais.
- Registradores de vetor.
Capaz de armazenar um vetor de operandos e operar simultaneamente em seu conteúdo.
- Instruções de vetor.
Operam em vetores em vez de escalares.



SIMD – GPU

- Linhas e triângulos da API para representar internamente a superfície de um objeto
- Um pipeline de processamento gráfico converte a representação interna em uma matriz de pixels que pode ser enviada para uma tela de computador.
- Muitos estágios deste pipeline (chamado **shader functions**) são programáveis



MIMD

- Suporta vários fluxos de instruções simultâneos operando em vários fluxos de dados.
- Normalmente consistem em uma coleção de unidades ou núcleos de processamento totalmente independentes, cada um com sua própria unidade de controle e sua própria ULA.

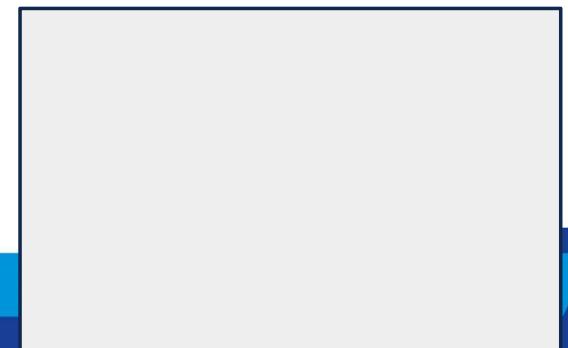


Processadores atuais

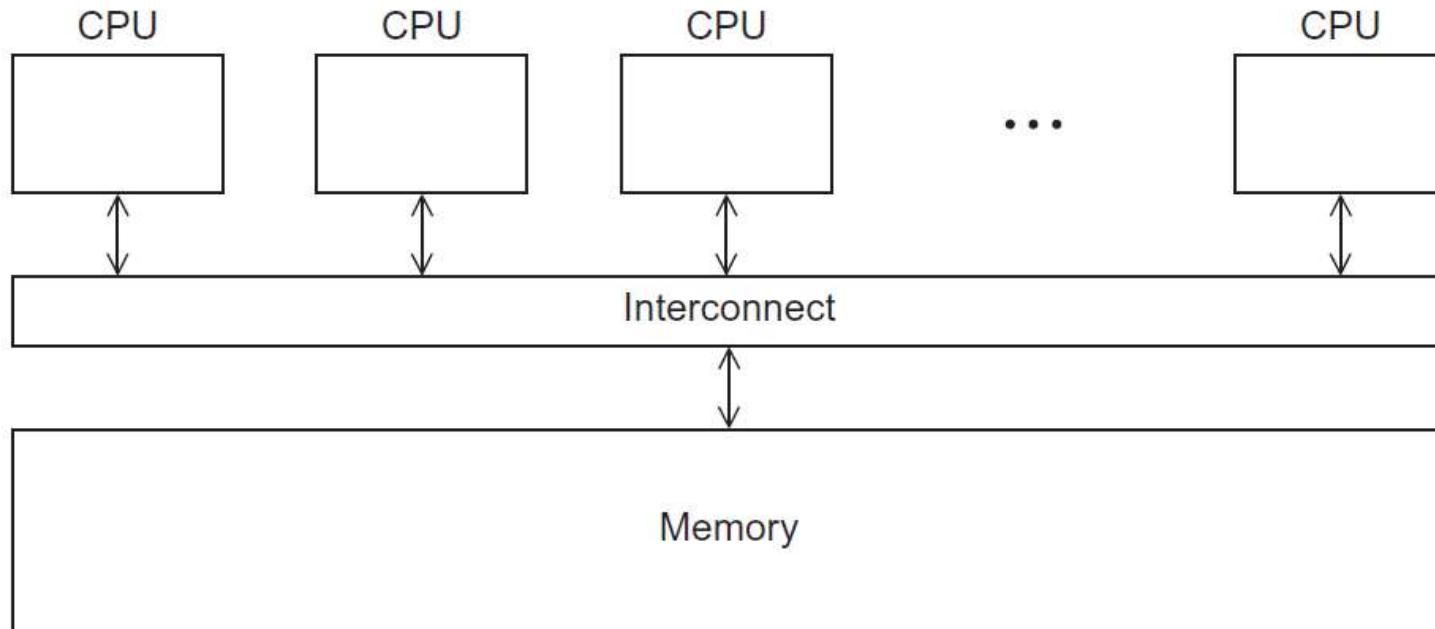
Acesso à Memória

Acesso à Memória

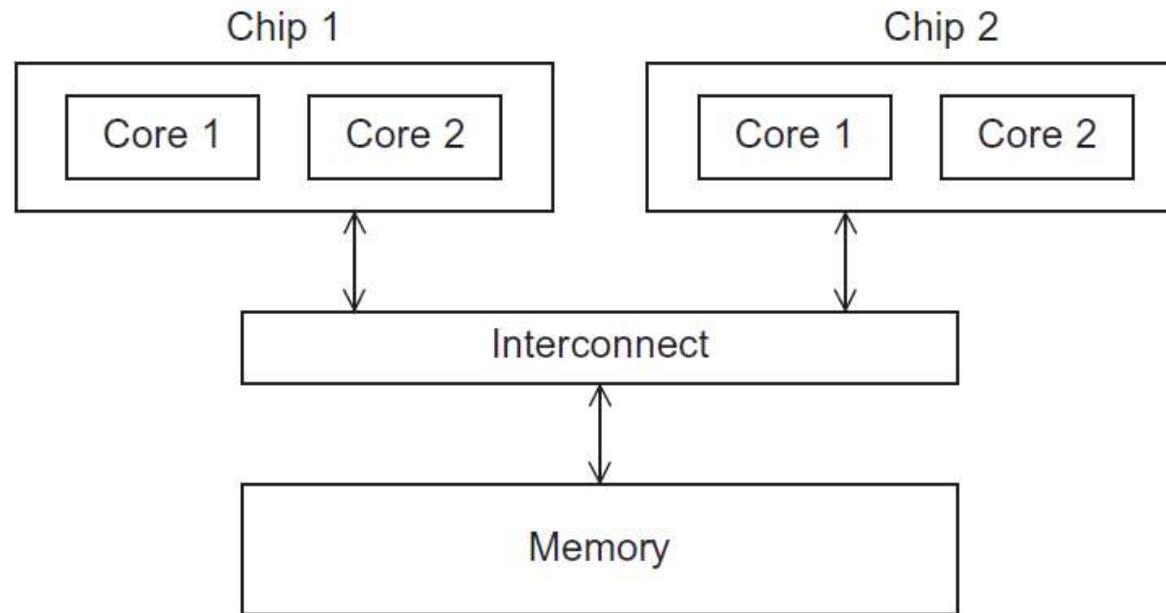
- Memória Compartilhada
- Memória Distribuída



Memória Compartilhada



Sistema Multicore UMA

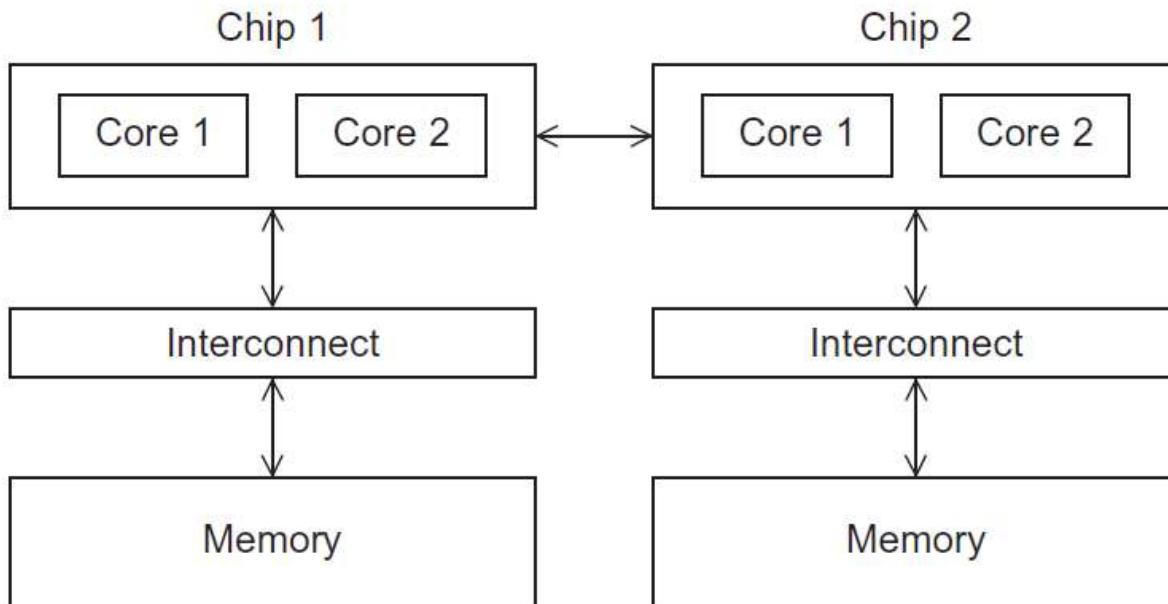


Tempo de acessar todos os locais de memória será o mesmo para todos os núcleos

UMA: Uniform Memory Access



Sistema Multicore NUMA



Um local de memória ao qual um núcleo está diretamente conectado pode ser acessado mais rapidamente do que um local de memória que deve ser acessado por outro chip.

NUMA: Non-Uniform
Memory Access



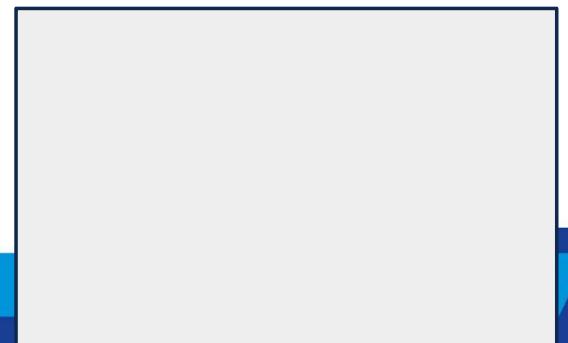
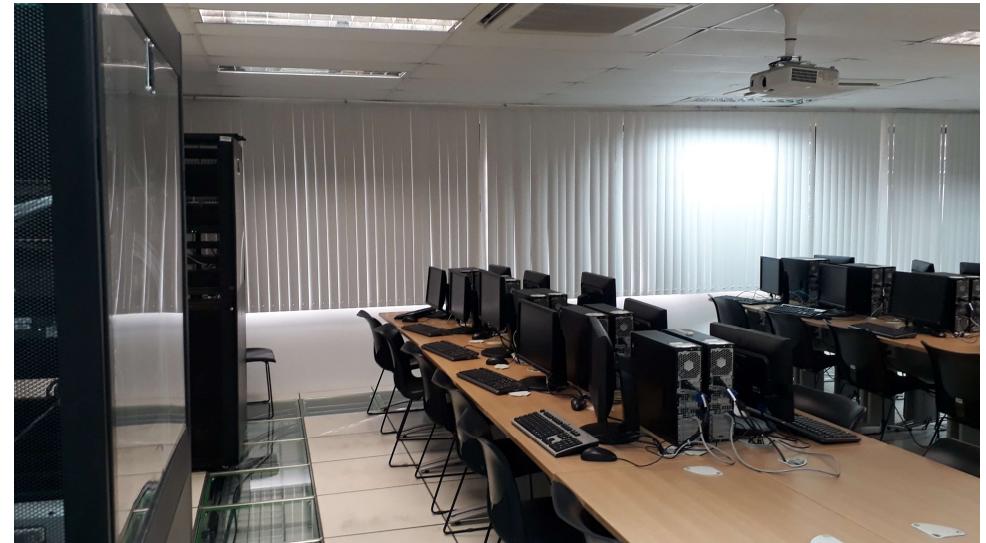
Memória Distribuída

- **Clusters (mais popular)**

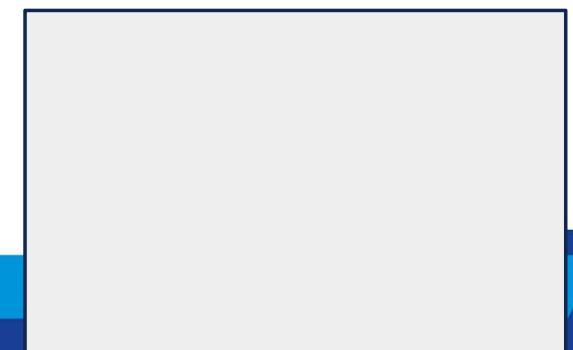
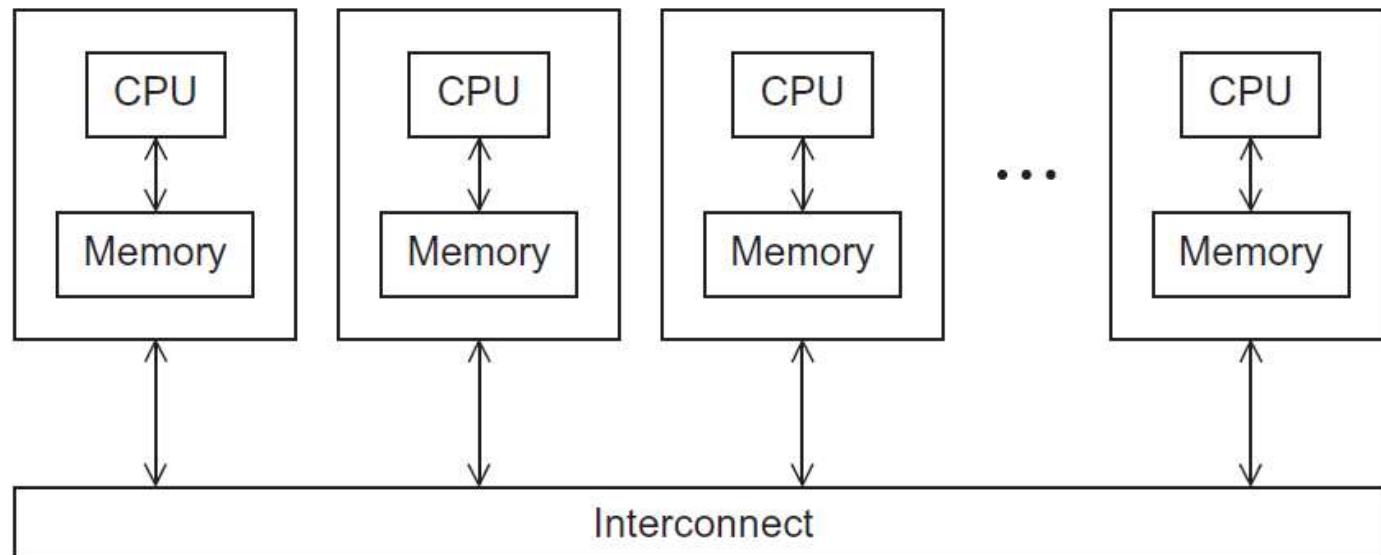
Uma coleção de sistemas computacionais conectados por uma rede de interconexão

- **Nodes (nós)** de um cluster são as unidades individuais de computações unidas pela rede de comunicação.

Conhecido como
Sistemas Híbridos



Memória Distribuída



Memória Distribuída - Definições

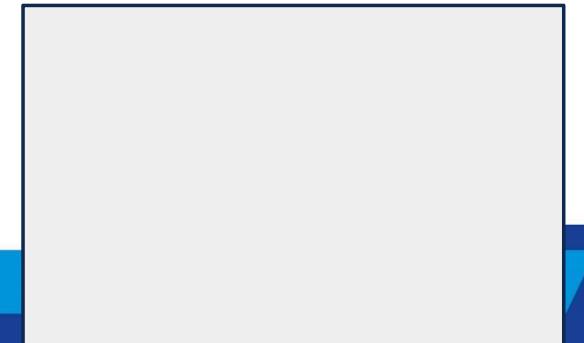
- Sempre que os dados são transmitidos, estamos interessados em quanto tempo levará para chegar ao destino.

- Latency (latência)**

O tempo decorrido entre o início da fonte para transmitir os dados e o destino começa a receber o primeiro byte

- Bandwidth (largura de banda)**

A taxa na qual o destino recebe dados depois de começar a receber o primeiro byte



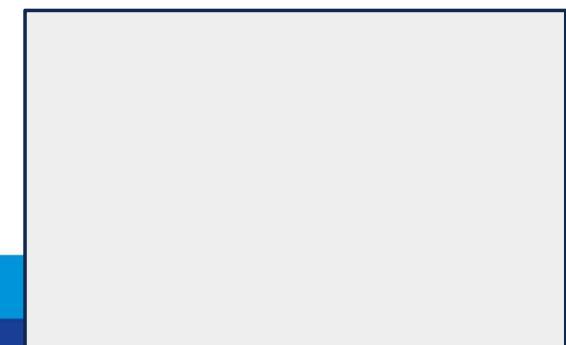
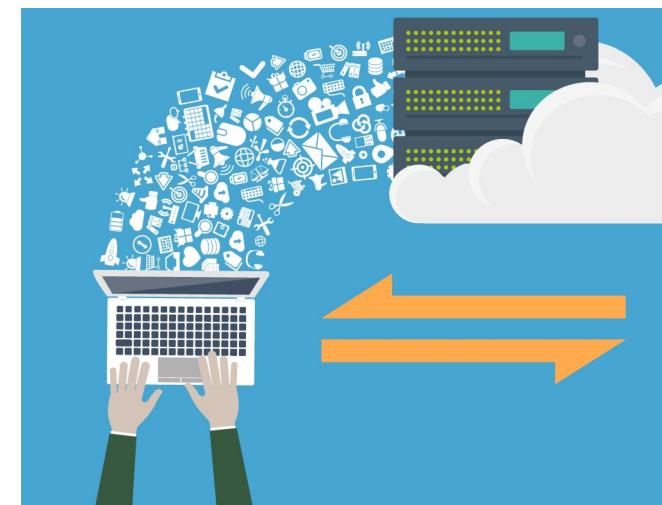
Memória Distribuída - Definições

Tempo de transmissão da msg = $| + n / b$

latência (segundos)

Tamanho da mensagem (bytes)

Largura de banda (bytes por segundo)





Não determinismo

```
...  
printf ( "Thread %d > my_val = %d\n" , my_rank , my_x );  
...
```



Thread 1 > my_val = 19
Thread 0 > my_val = 7



Thread 0 > my_val = 7
Thread 1 > my_val = 19



Desempenho – Speedup

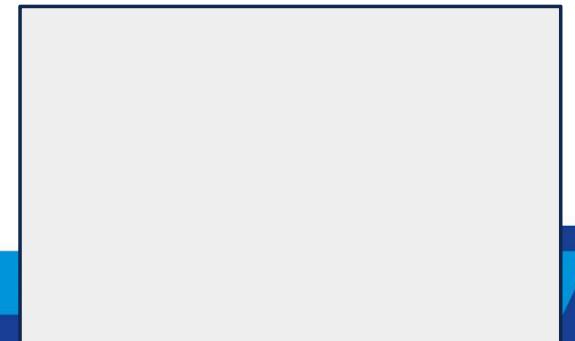
**“Queremos não somente a
solução correta, mas também
VELOCIDADE!”**

Qualquer programador paralelo

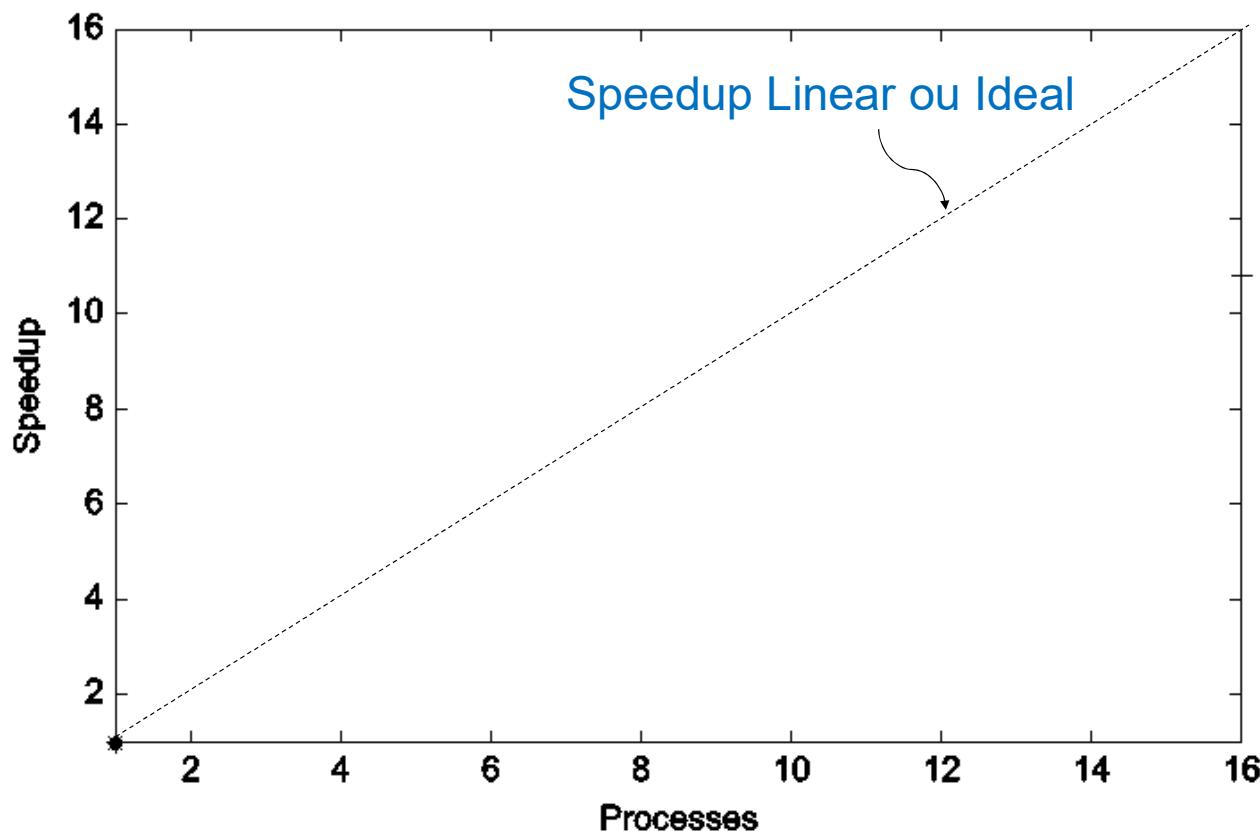
Speedup

- Speedup = S
- Tempo de execução serial = T_{serial}
- Tempo de execução paralelo = T_{paralelo}

$$S = \frac{T_{\text{serial}}}{T_{\text{paralelo}}}$$



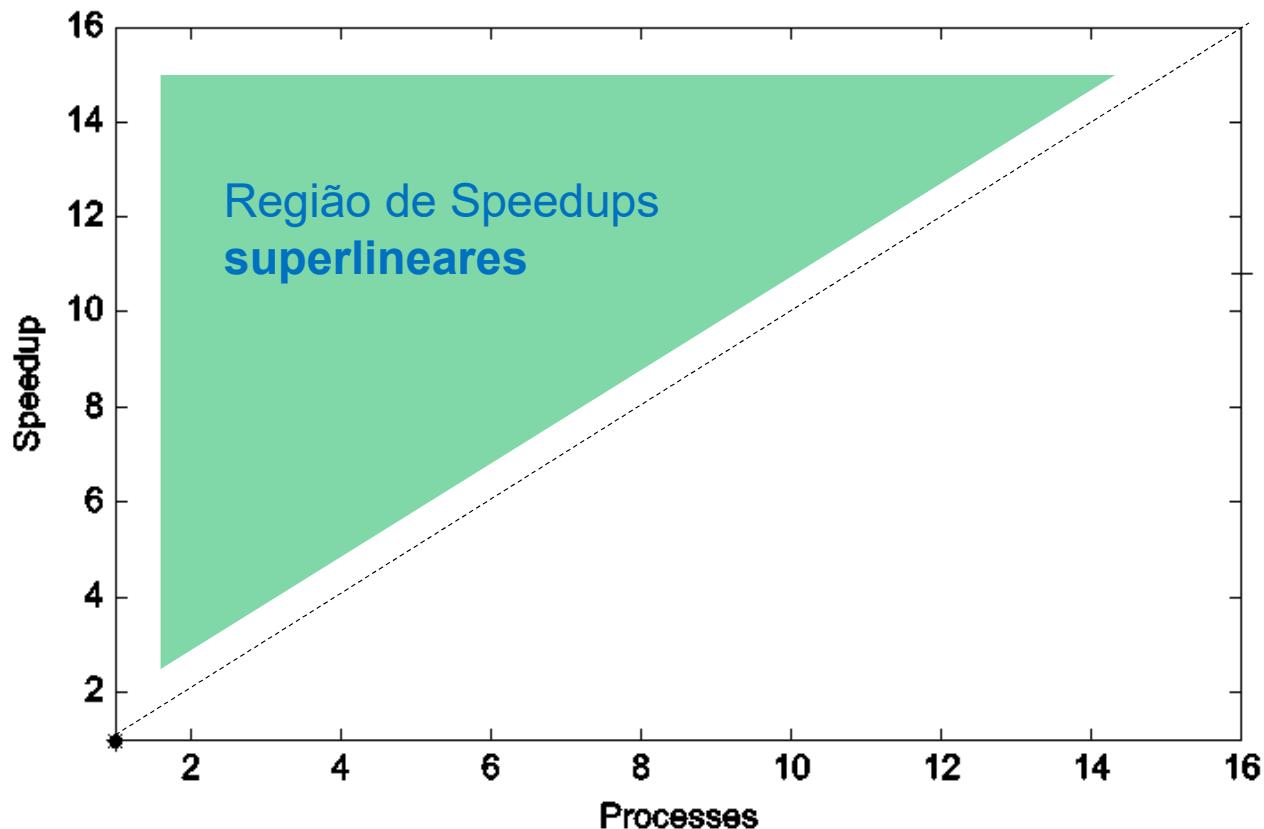
Speedup



Para um tamanho de problema fixo



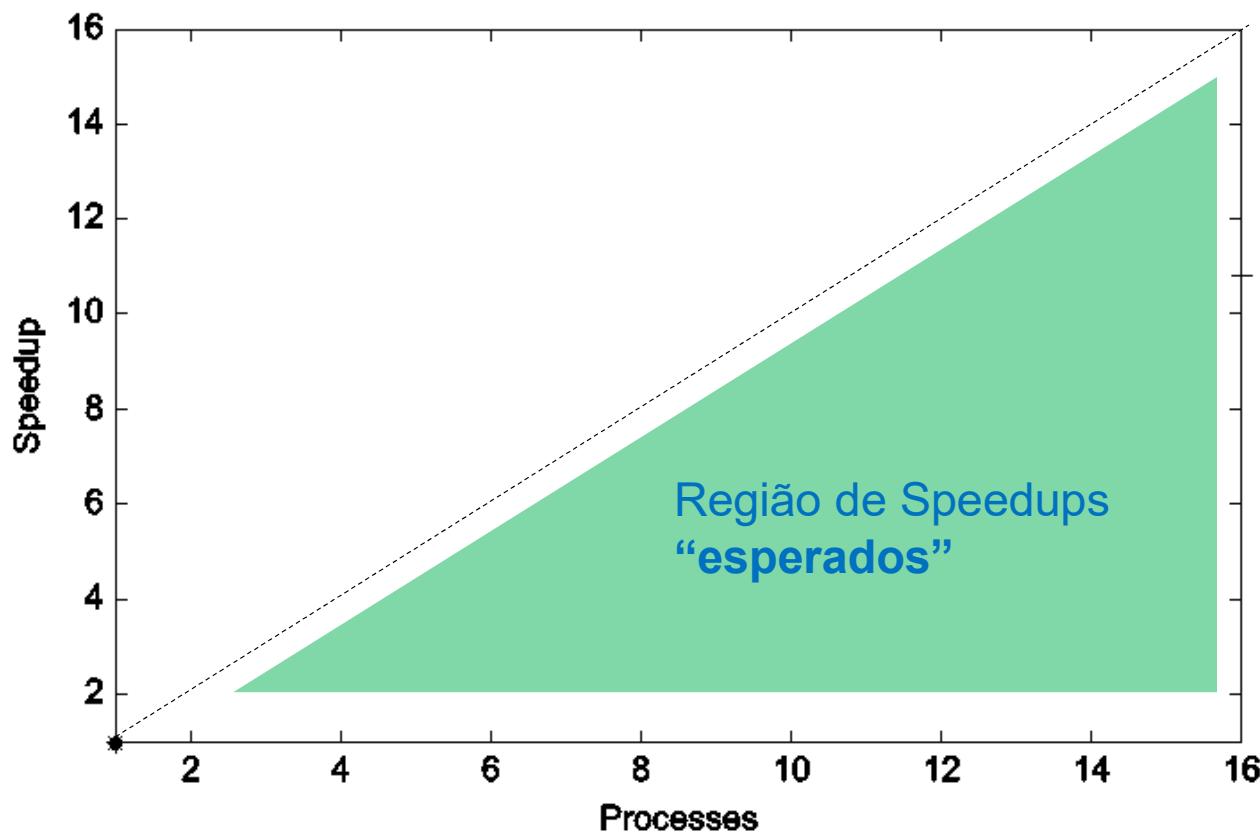
Speedup



Para um tamanho de problema fixo

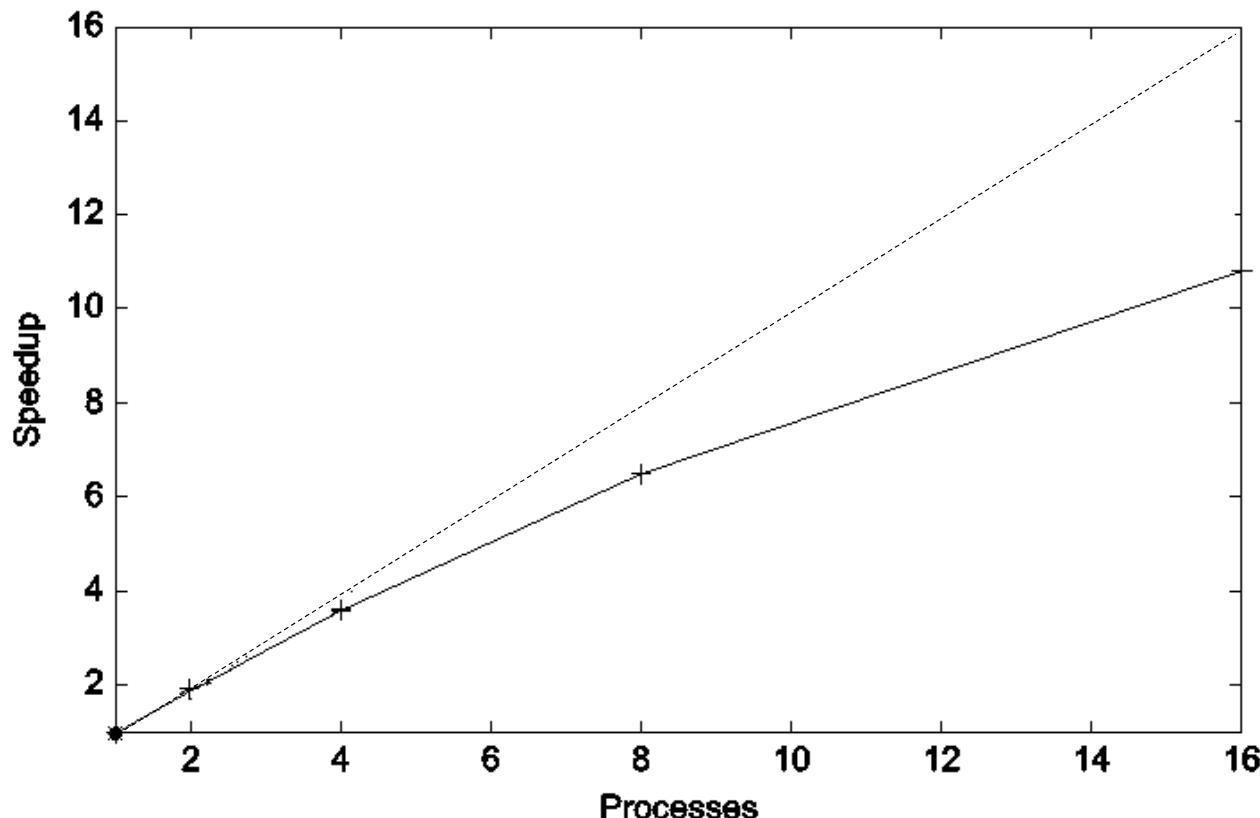


Speedup

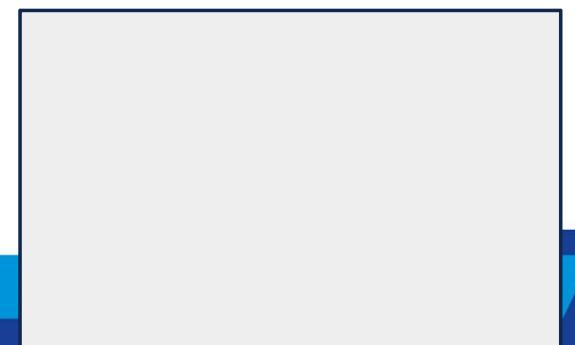


Para um tamanho de problema fixo

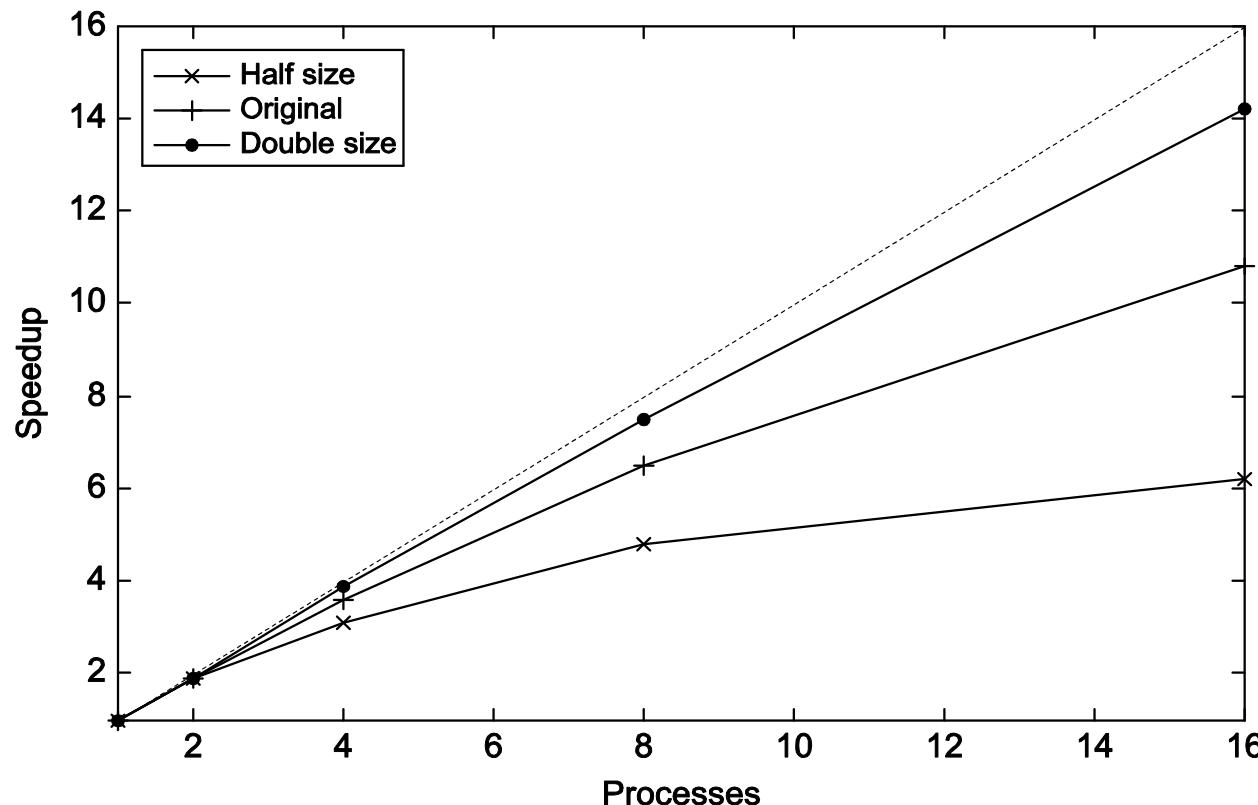
Speedup – Comportamento esperado



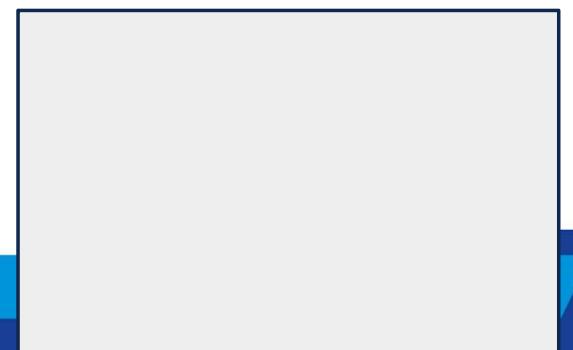
Para um tamanho de problema fixo



Speedup – Comportamento esperado

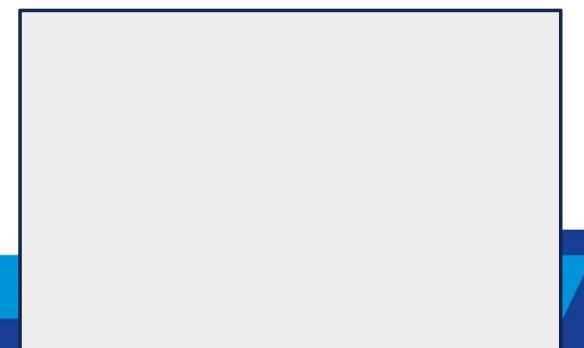
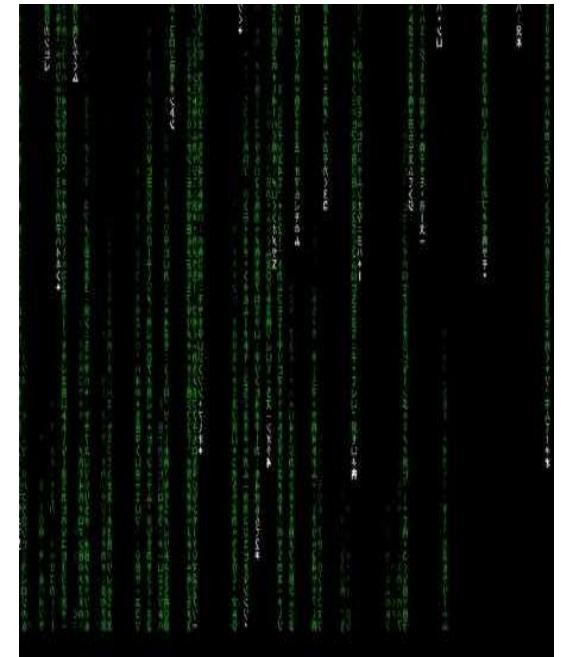
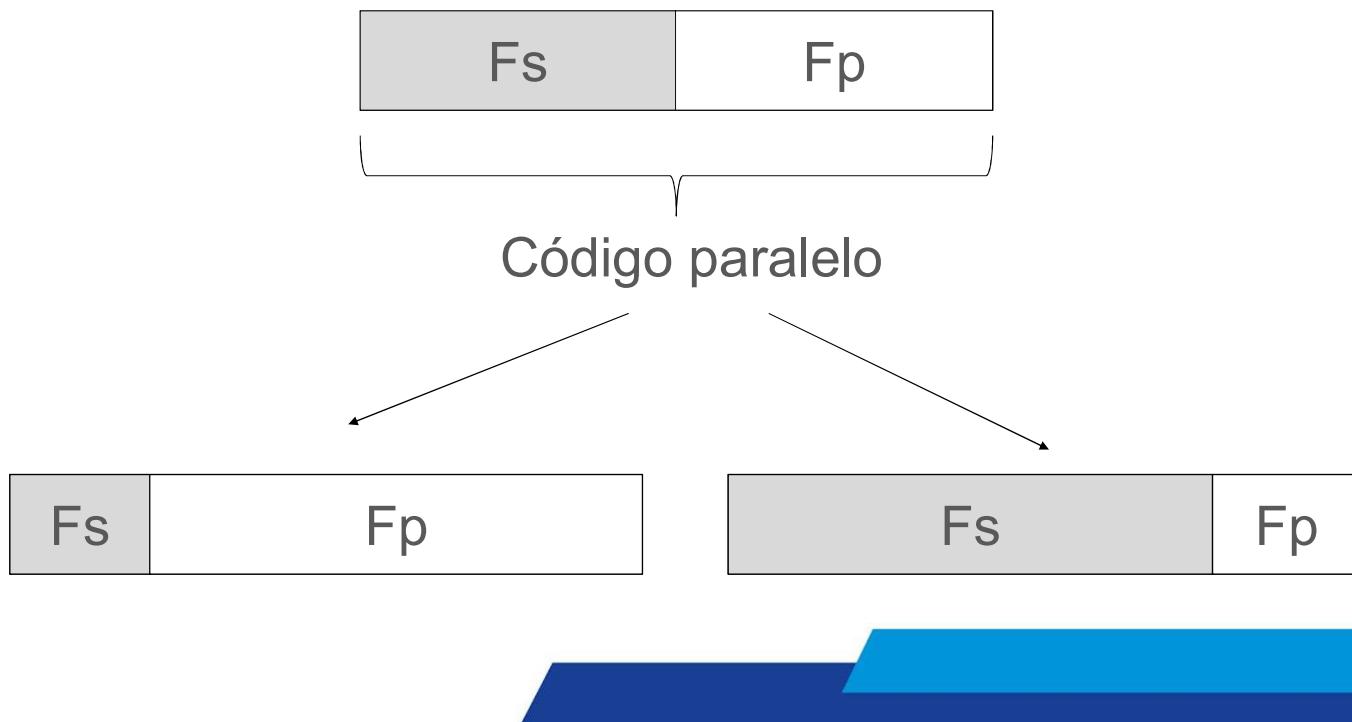


Comportamento do speedup com tamanhos diferentes do mesmo problema



Lei de Amdahl

“O speedup é limitado pela fração serial do código”

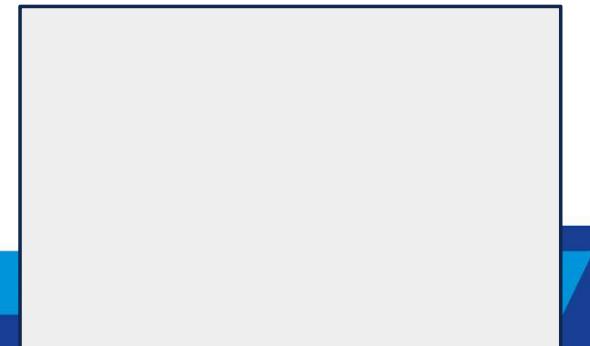


Exemplo 1: fictício

- Lista de exercícios com 50 exercícios feita por 5 alunos.
 - Um aluno precisa dizer quem pegará qual questão;
 - A quantidade de questões é dividida entre os alunos;
 - Um aluno reúne tudo para entregar ao professor;

Exemplo 1: fictício

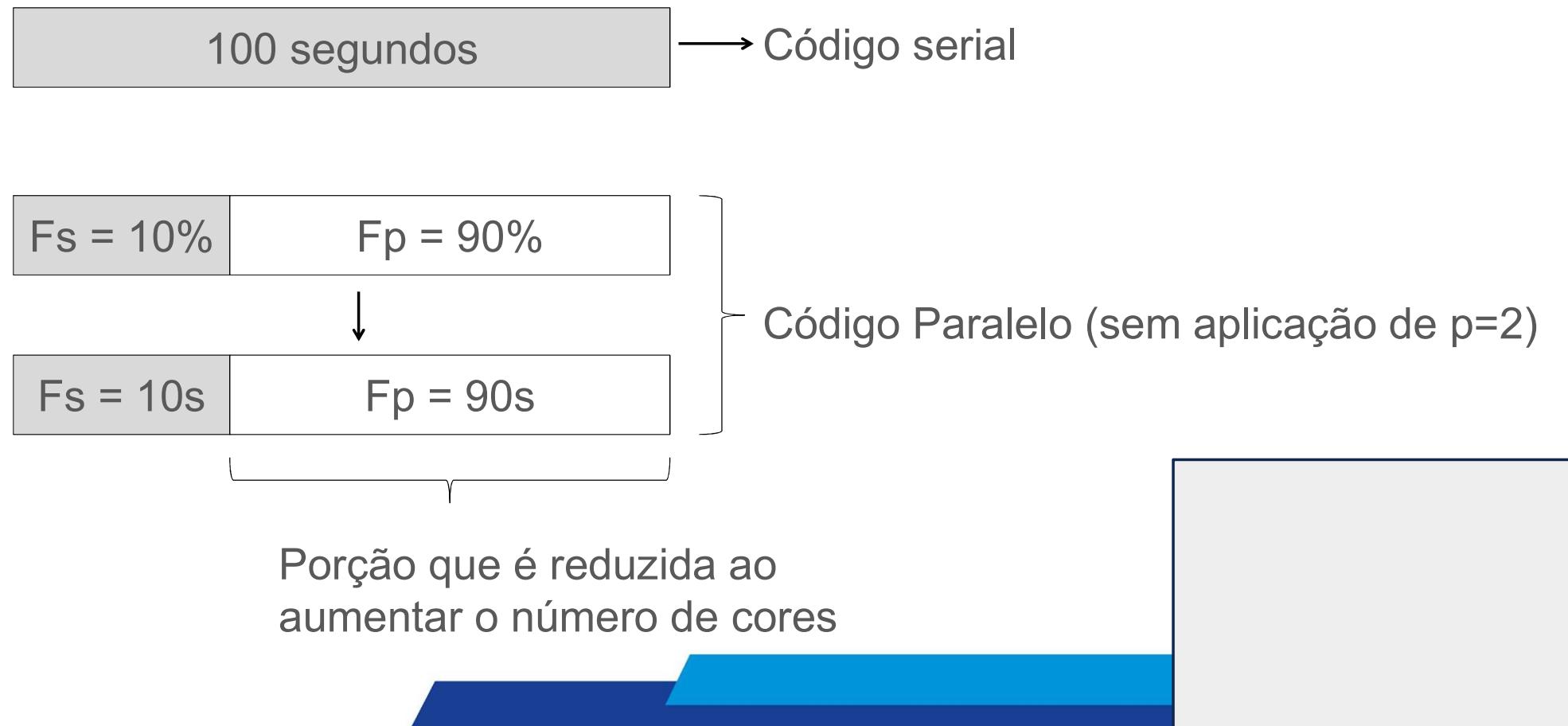
- Lista de exercícios com 50 exercícios feita por 5 alunos.
 - Um aluno precisa dizer quem pegará qual questão;
 - Feito serial. Não existe se 1 aluno faz a lista
 - A quantidade de questões é dividida entre os alunos;
 - Pode ser dividido pela qtd ou dificuldade das questões
 - Um aluno reúne tudo para entregar ao professor;
 - Feito serial. Não existe se 1 aluno faz a lista



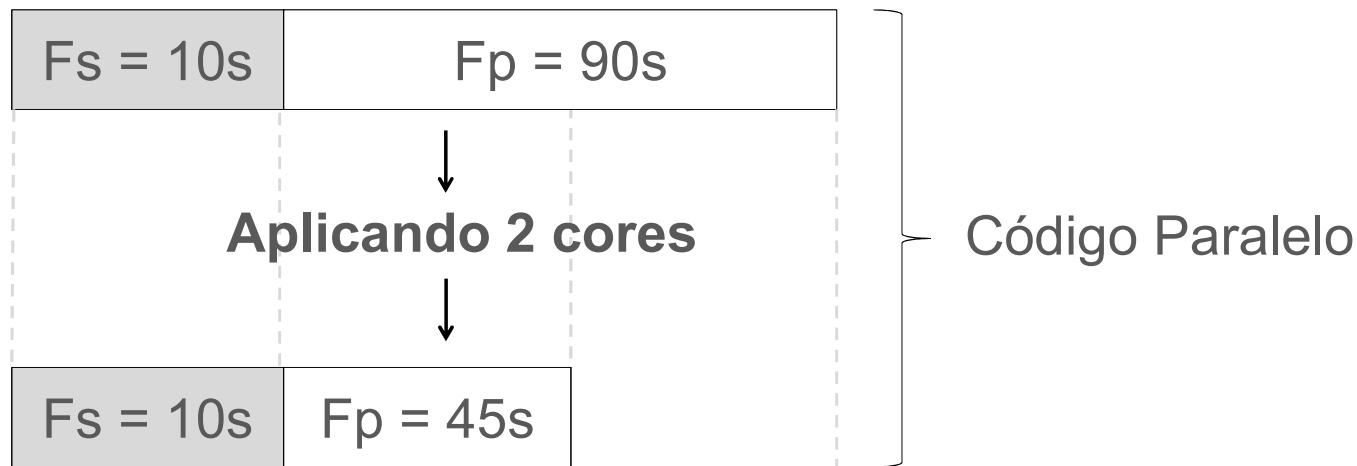
Exemplo 2:

- Suponha um código serial executa em 100 segundos e sua versão paralela é 90% paralelizável
- A paralelização é “perfeita” independentemente do número de cores
- Qual o speedup para 2 cores?

Exemplo 2:



Exemplo 2:



- CONCLUSÃO
 - $T_{\text{serial}} = 100\text{s}$
 - $T_{\text{paralelo}} = 55\text{s}$
 - $\text{Speedup} = T_{\text{serial}} / T_{\text{paralelo}} = 100/55 = 1,81$

Exemplo 3:

- Suponha um código serial executa em 100 segundos e sua versão paralela é 50% paralelizável
- A paralelização é “perfeita” independentemente do número de cores
- Qual o speedup para 2 cores ?
- **RESPOSTA**
- $T_{\text{serial}} = 100\text{s}$
- $T_{\text{paralelo}} = 50 + (50/2) = 50 + 25 = 75\text{s}$
- $\text{Speedup} = 100/75 = 1,33$

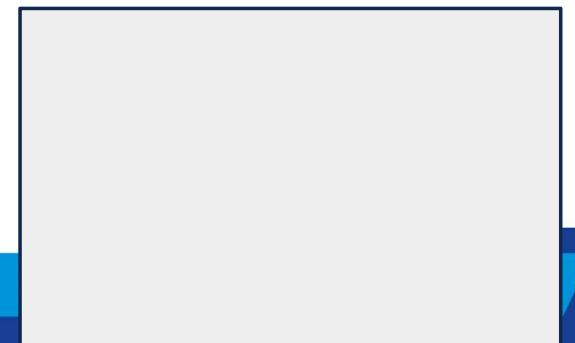
Desempenho – Eficiência

Eficiência

Comenta o **quão bem** os seus processadores estão sendo utilizados pelo código em relação ao speedup linear

- Eficiência 1 (100%) com 20 processadores = Speedup 20
 - Eficiência 0.5 (50%) com 20 processadores = Speedup 10
 - Eficiência 0.1 (10%) com 20 processadores = Speedup 2
-
- **Eficiência = E**
 - **Speedup = S**
 - **Número de cores = p**

$$E = \frac{S}{p}$$

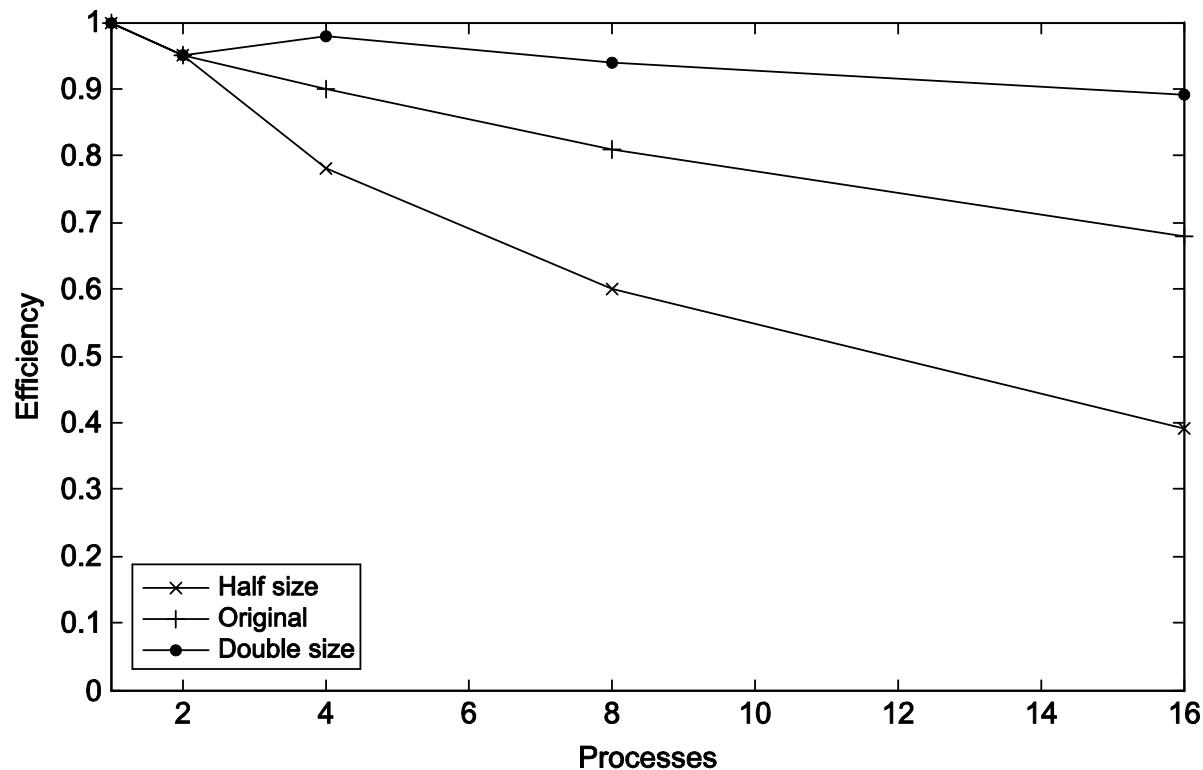


Exemplo

p	1	2	4	8	16
S	1.9	1.9	3.6	6.5	10.8
$E = S/p$	0.95	0.95	0.90	0.81	0.68

Para um tamanho de problema fixo

Eficiência - Comportamento

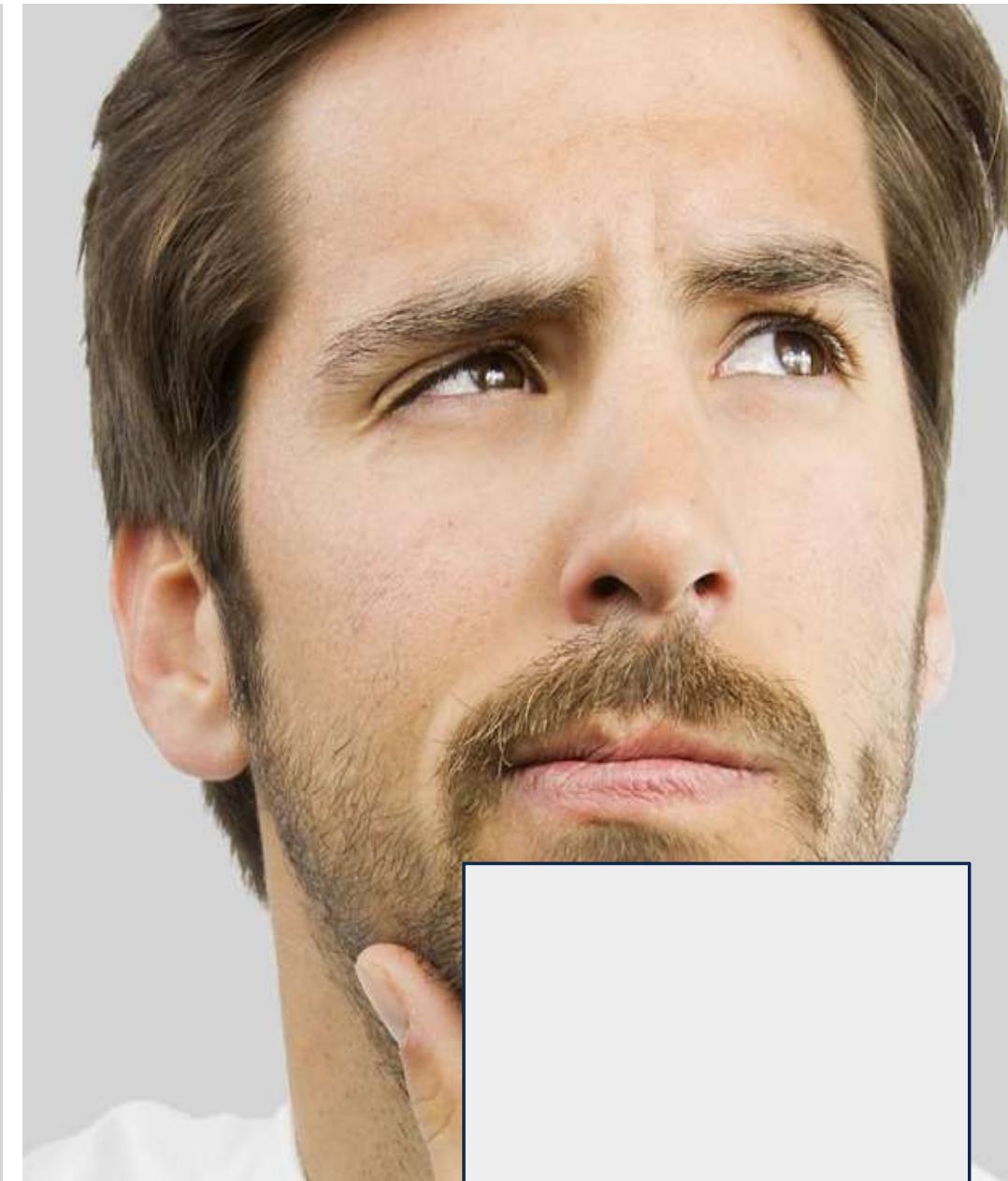


Para um tamanho de problema fixo





**E quem disse que eu preciso ter
um tamanho fixo?**



*“Quanto mais poder
computacional possuo, maiores e
mais complexos problemas eu
posso computar”*

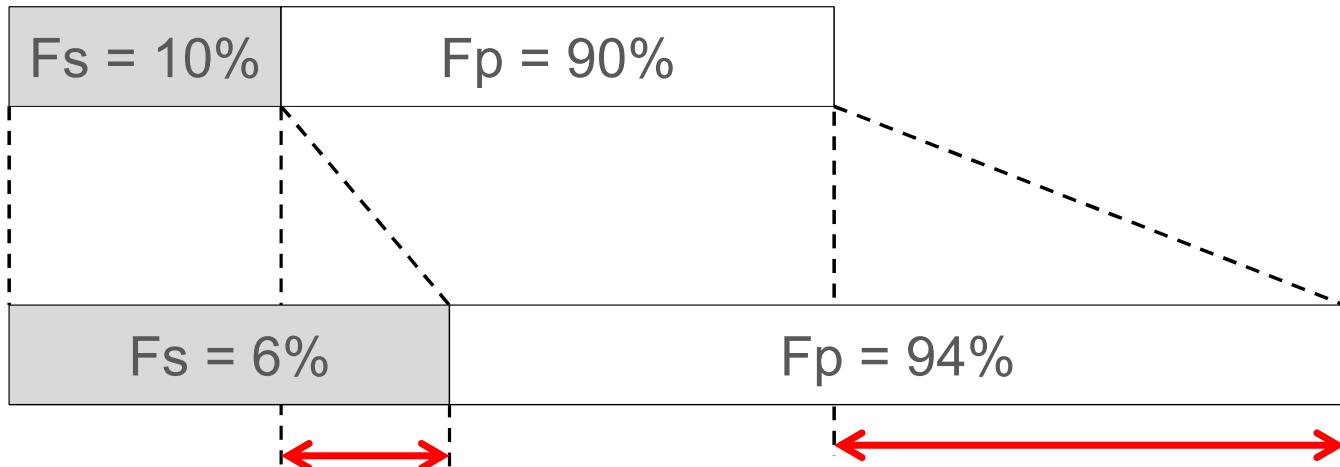
Aumentar o tamanho do problema

	p	1	2	4	8	16
Half	S	1.9	1.9	3.1	4.8	6.2
	E	0.95	0.95	0.78	0.60	0.39
Original	S	1.9	1.9	3.6	6.5	10.8
	E	0.95	0.95	0.90	0.81	0.68
Double	S	1.9	1.9	3.9	7.5	14.2
	E	0.95	0.95	0.98	0.94	0.89



Para um tamanho de problema variável

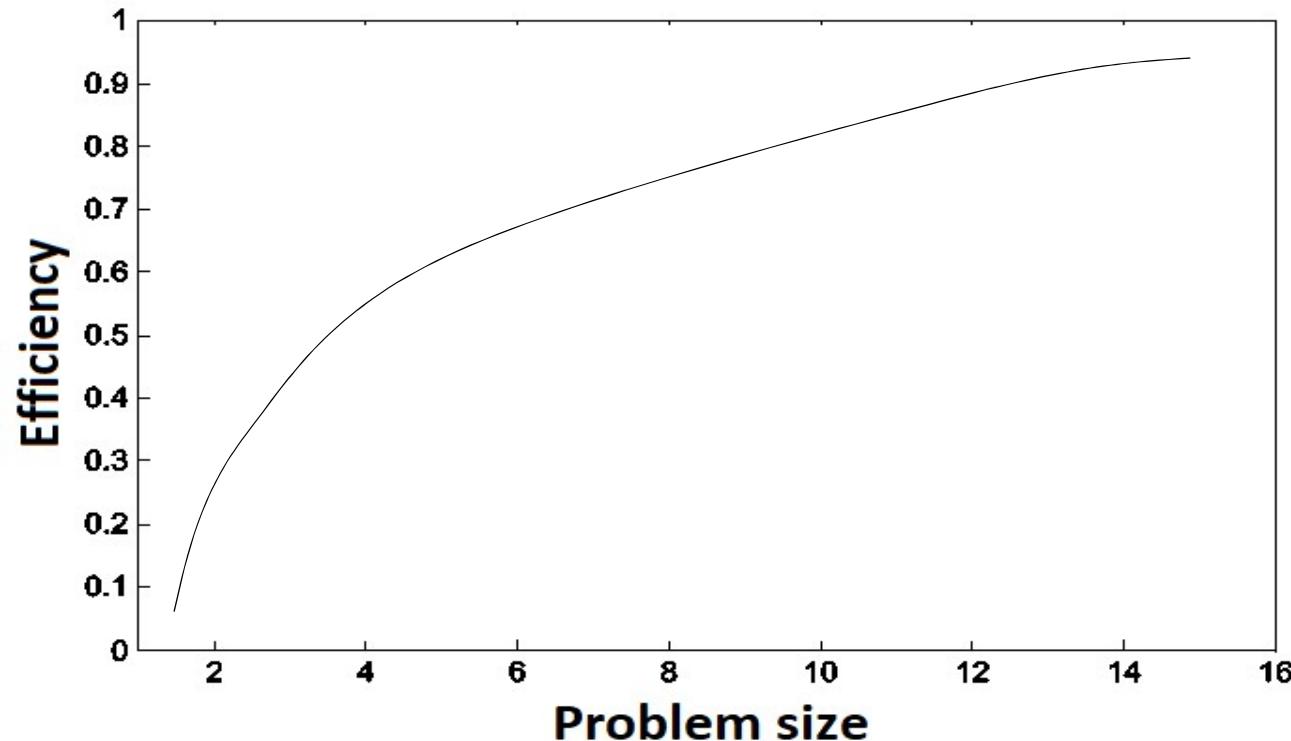
Importância tamanho do problema



Em bons algoritmos paralelos (escalonáveis), se aumentarmos o tamanho do problema, espera-se que F_s aumente menos do que F_p .

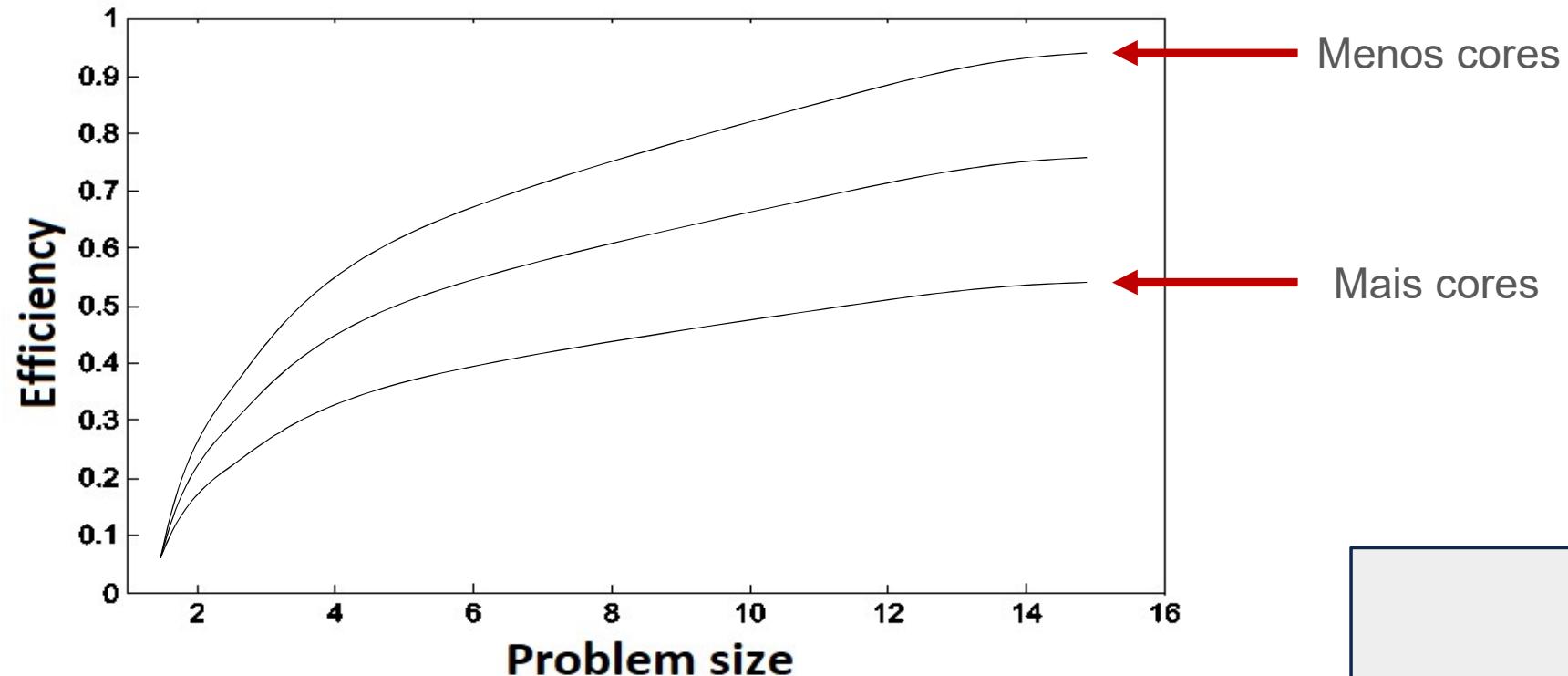
Com isso, Speedup aumenta e, consequentemente, Eficiência também

Eficiência - #cores fixo



Para um número de cores (p) fixo

Eficiência - #cores fixo



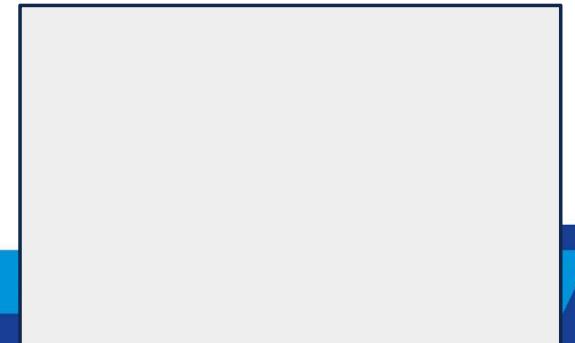
Para um número de cores (p) fixo

Classificação de escalabilidade

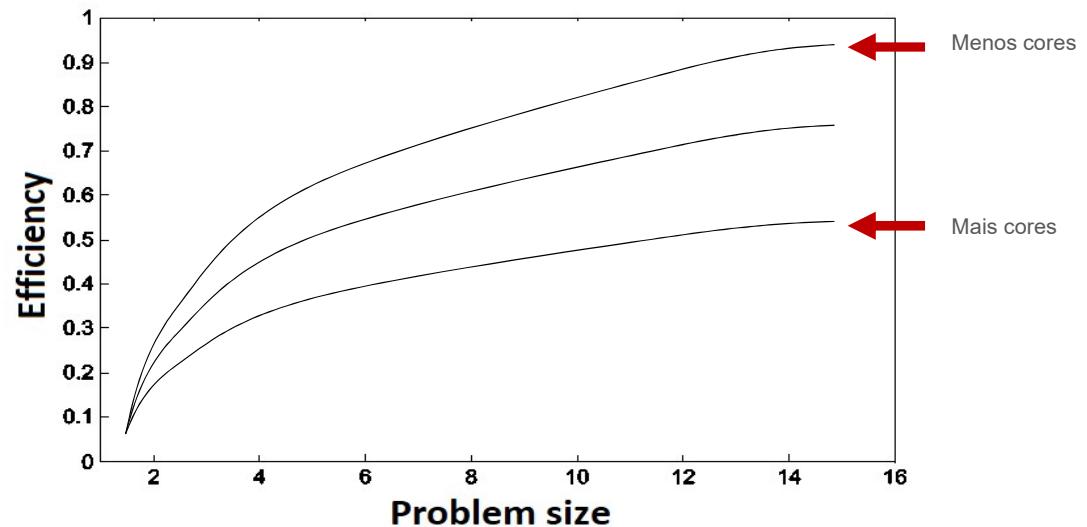
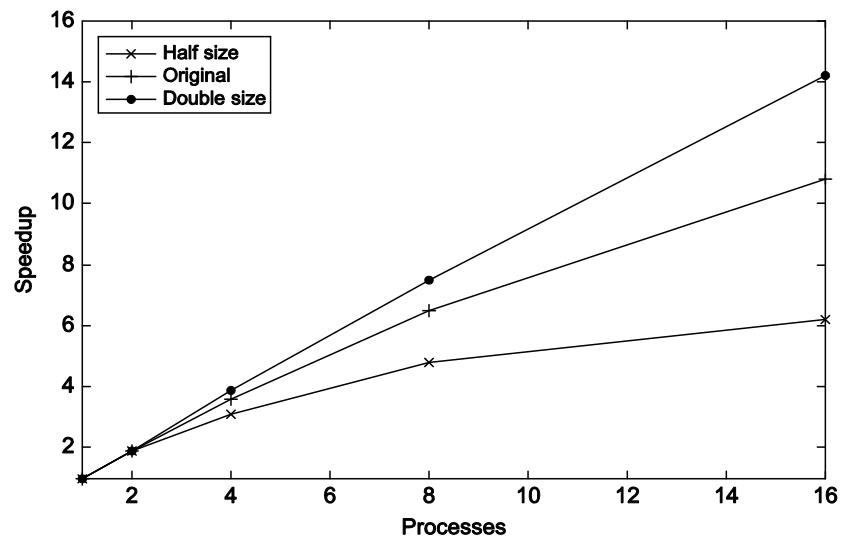
1) **Fortemente escalável:** Se eficiência aumenta ou permanece constante quando mantemos o tamanho do problema fixo e aumentamos o número de cores (**MUITO RARO**)

2) **Fracamente escalável:** Se a eficiência aumenta ou permanece constante quando aumentamos o tamanho do problema na mesma proporção que aumentamos o número de cores (**COMUM**)

3) **Algoritmo é escalável:** Se a eficiência aumenta ou permanece constante quando aumentamos o tamanho do problema na mesma proporção e aumentamos o número de cores (**COMUM**)



Resumo – O que eu uso?



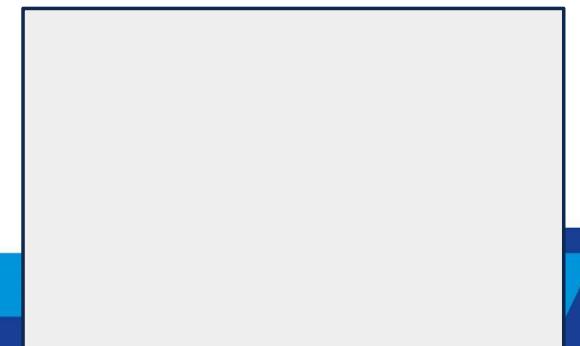
- 1) Analisa comportamento crescente do Speedup
- 2) Analisa o comportamento crescente da Eficiência
 - 2.1) Classifica o algoritmo como **fortemente escalável, fracamente escalável ou somente escalável.**

Metodología de Foster

Design de programas paralelos

Metodologia de Foster

- 1) Particionamento
- 2) Comunicação
- 3) Aglomeração ou agregação
- 4) Mapeamento



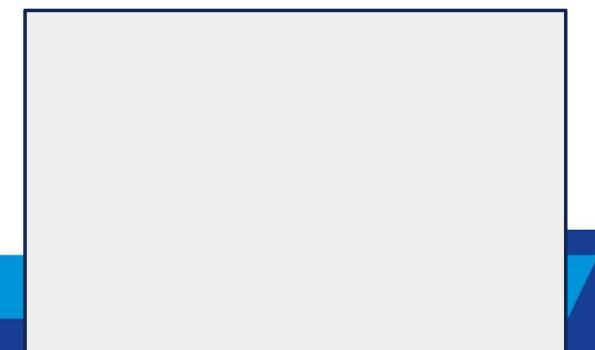
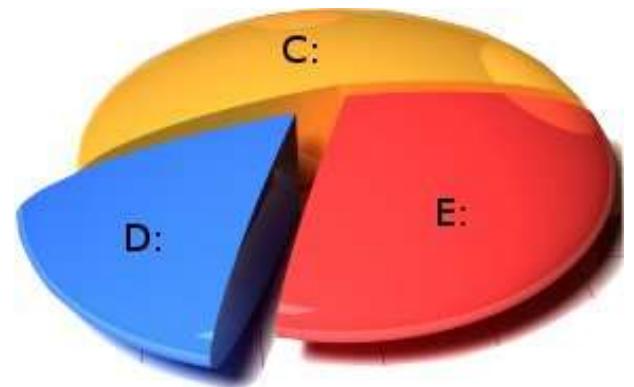
Design de programas paralelos

Metodologia de Foster

1) Particionamento

Divida a computação a ser executada e os dados operados pela computação em pequenas tarefas.

O foco aqui deve estar na identificação de tarefas que podem ser executadas em paralelo.

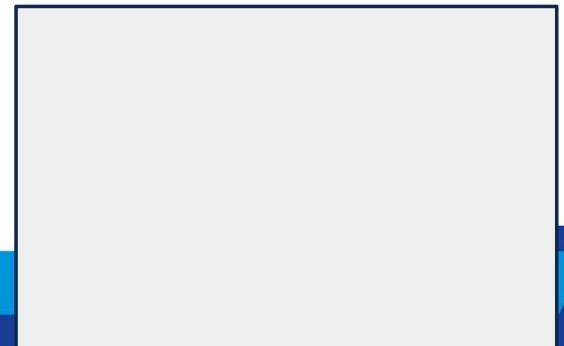


Design de programas paralelos

Metodologia de Foster

2) Comunicação

Determine qual comunicação deve ser realizada entre as tarefas identificadas na etapa anterior



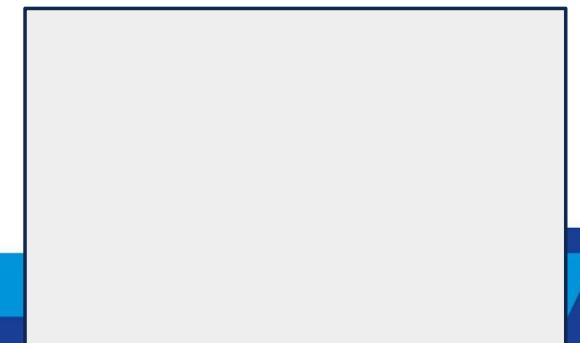
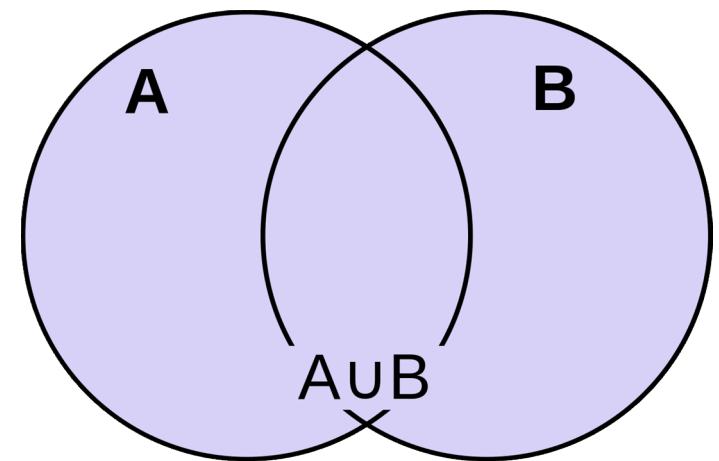
Design de programas paralelos

Metodologia de Foster

3) Aglomeração ou agregação

Combine tarefas e comunicações identificadas na primeira etapa em tarefas maiores.

Por exemplo, se a tarefa A deve ser executada antes da tarefa B, pode fazer sentido agregá-las em uma única tarefa composta.



Design de programas paralelos

Metodologia de Foster

4) Mapeamento

Atribua as tarefas compostas identificadas na etapa anterior aos cores.

Isso deve ser feito para que a comunicação seja minimizada e cada core obtenha aproximadamente a mesma quantidade de trabalho.

