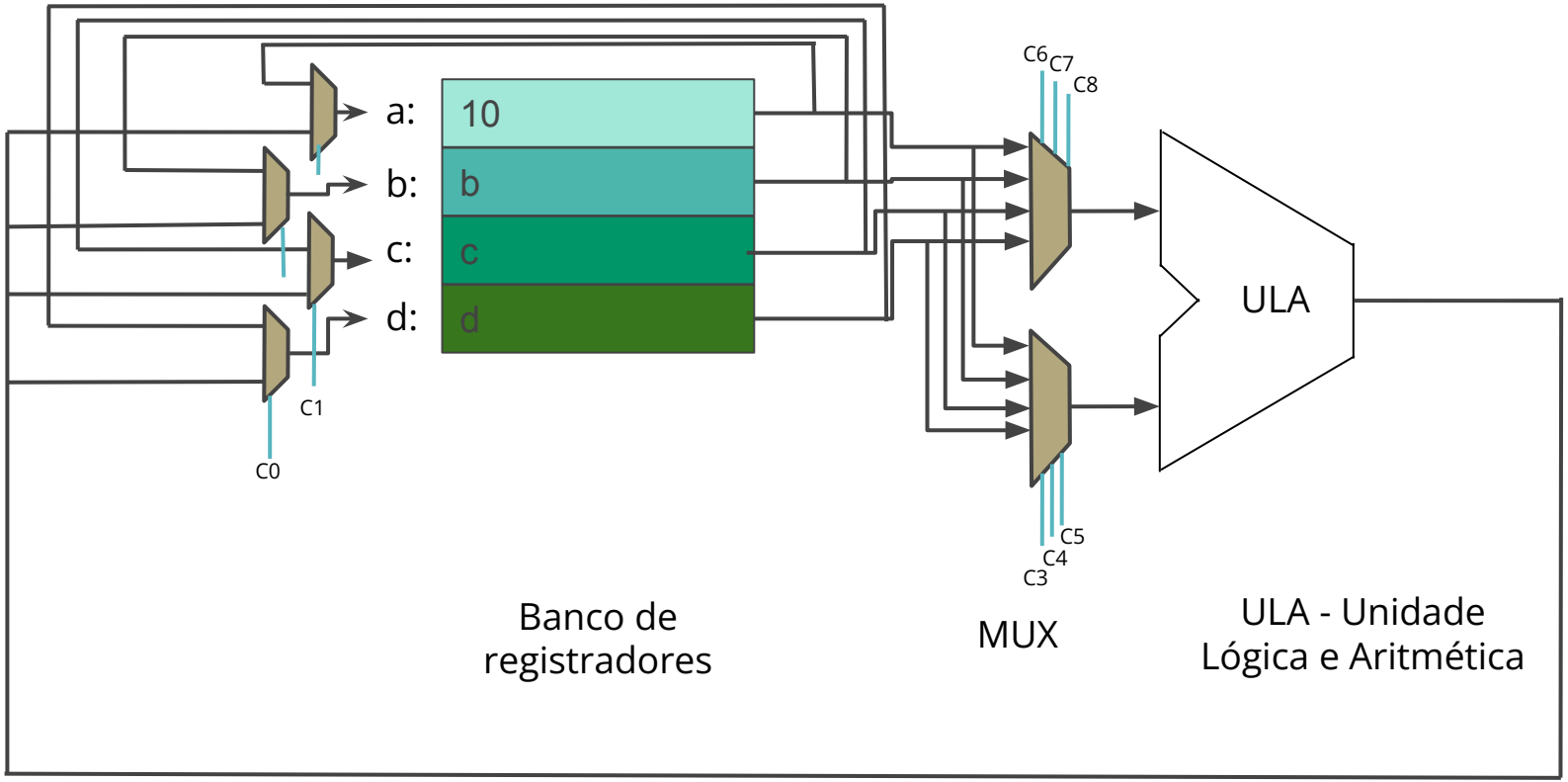
Processadores superescalares e taxonomia de Flynn

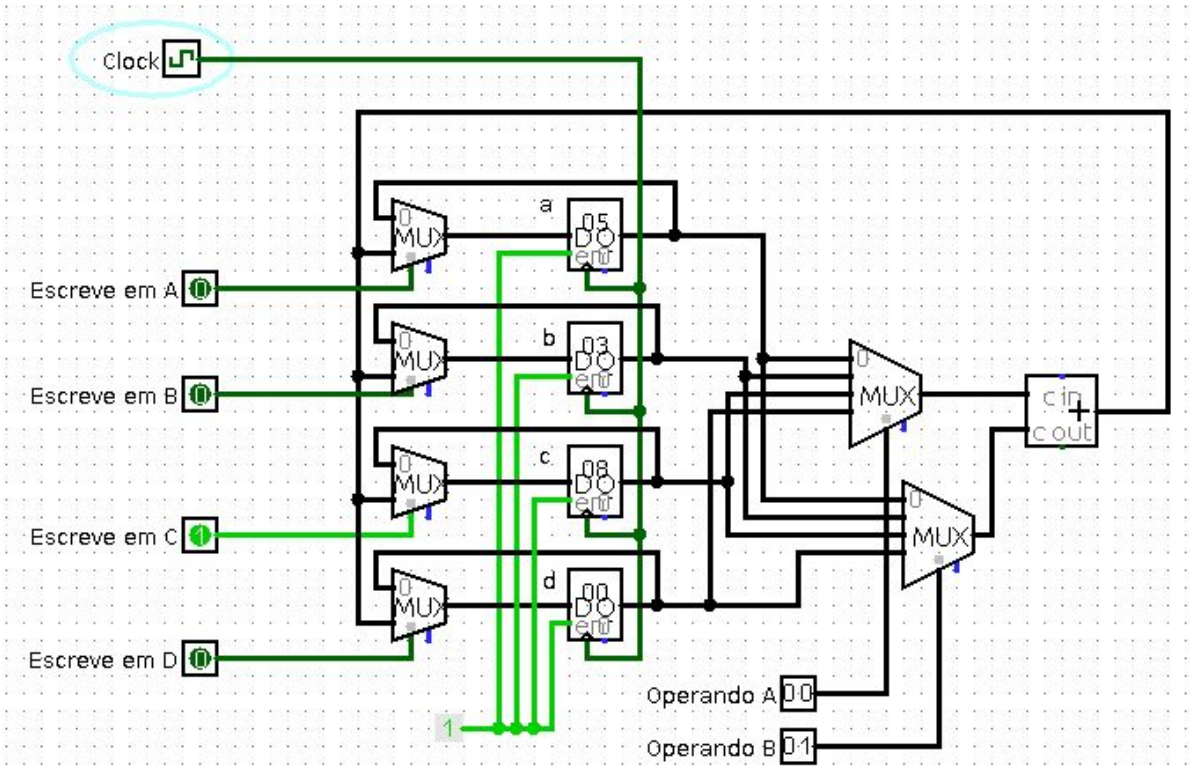
Arquitetura de
Sistemas Computacionais

Correções

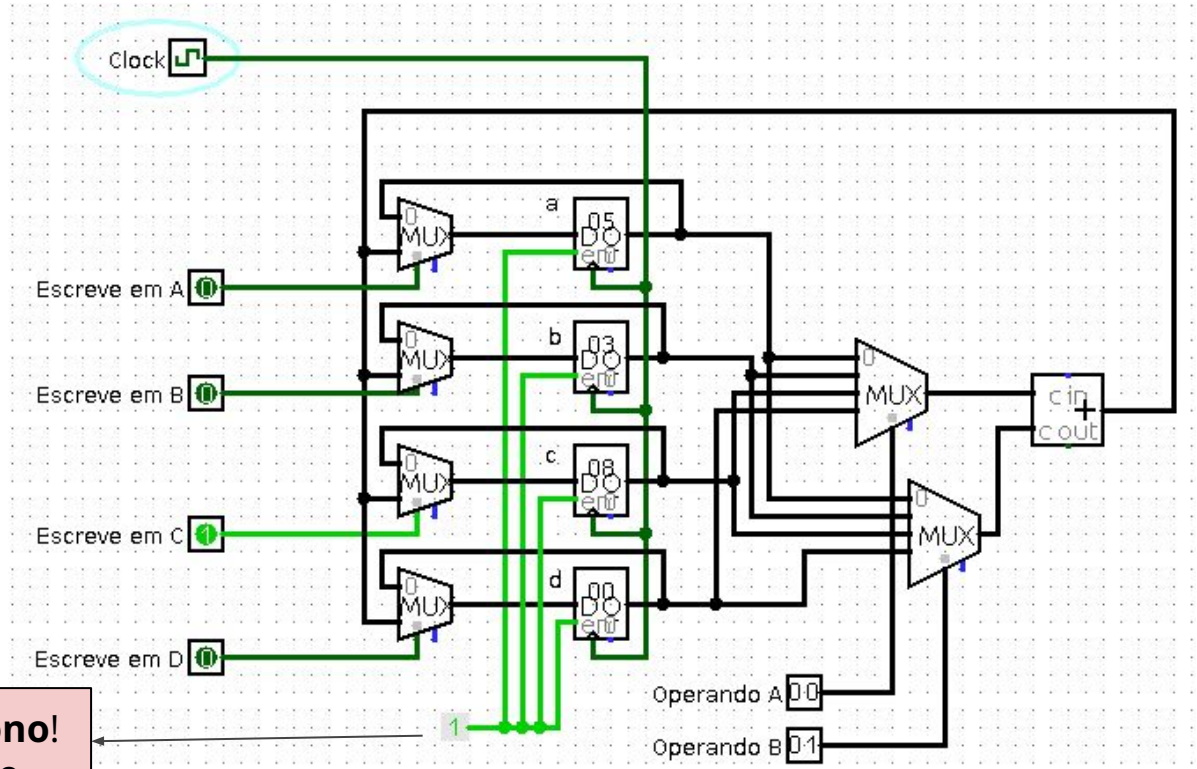
Leitura de registradores e execução



Leitura de registradores e execução



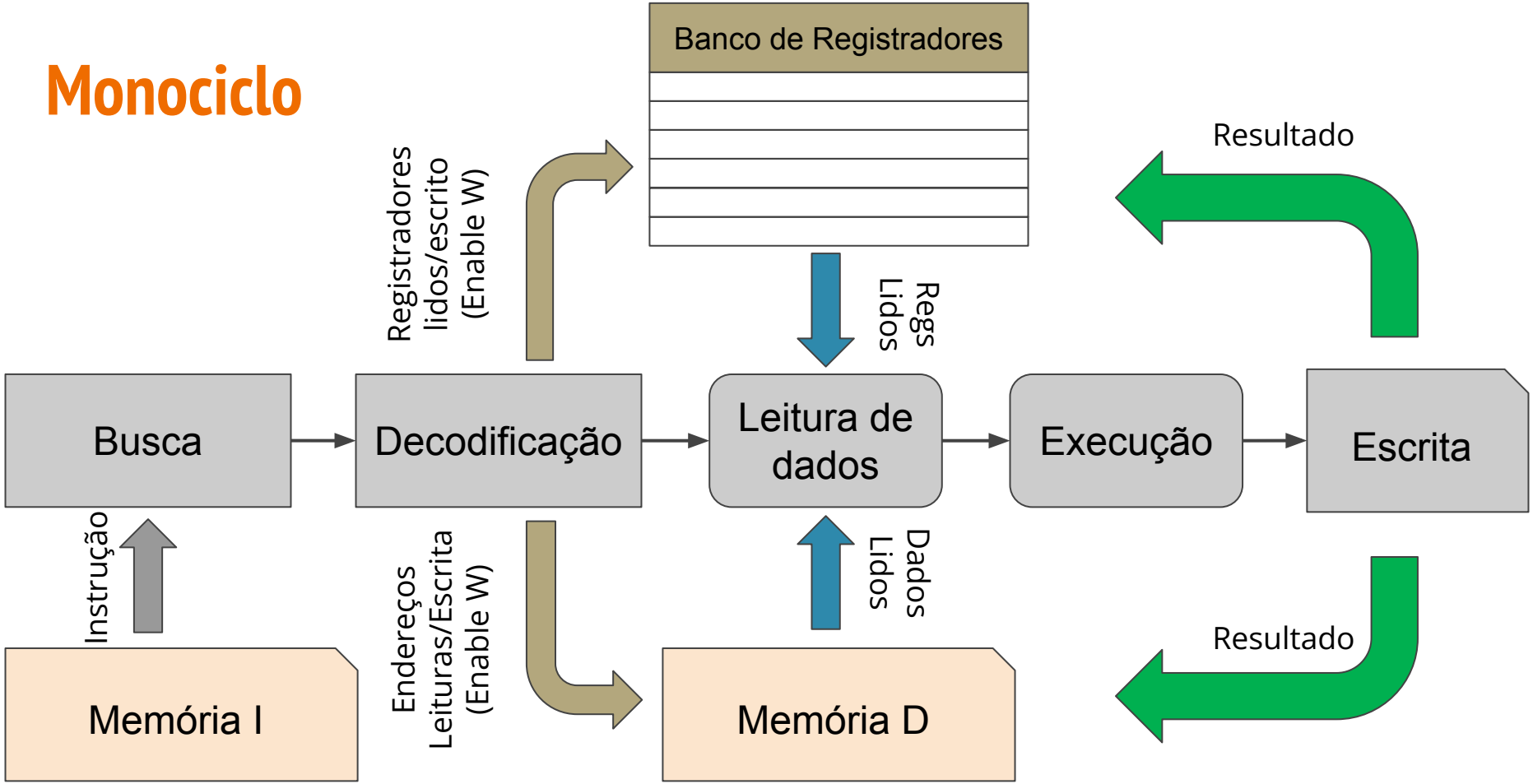
Leitura de registradores e execução



Monociclo é **síncrono**!
1 ciclo por instrução

Revisão

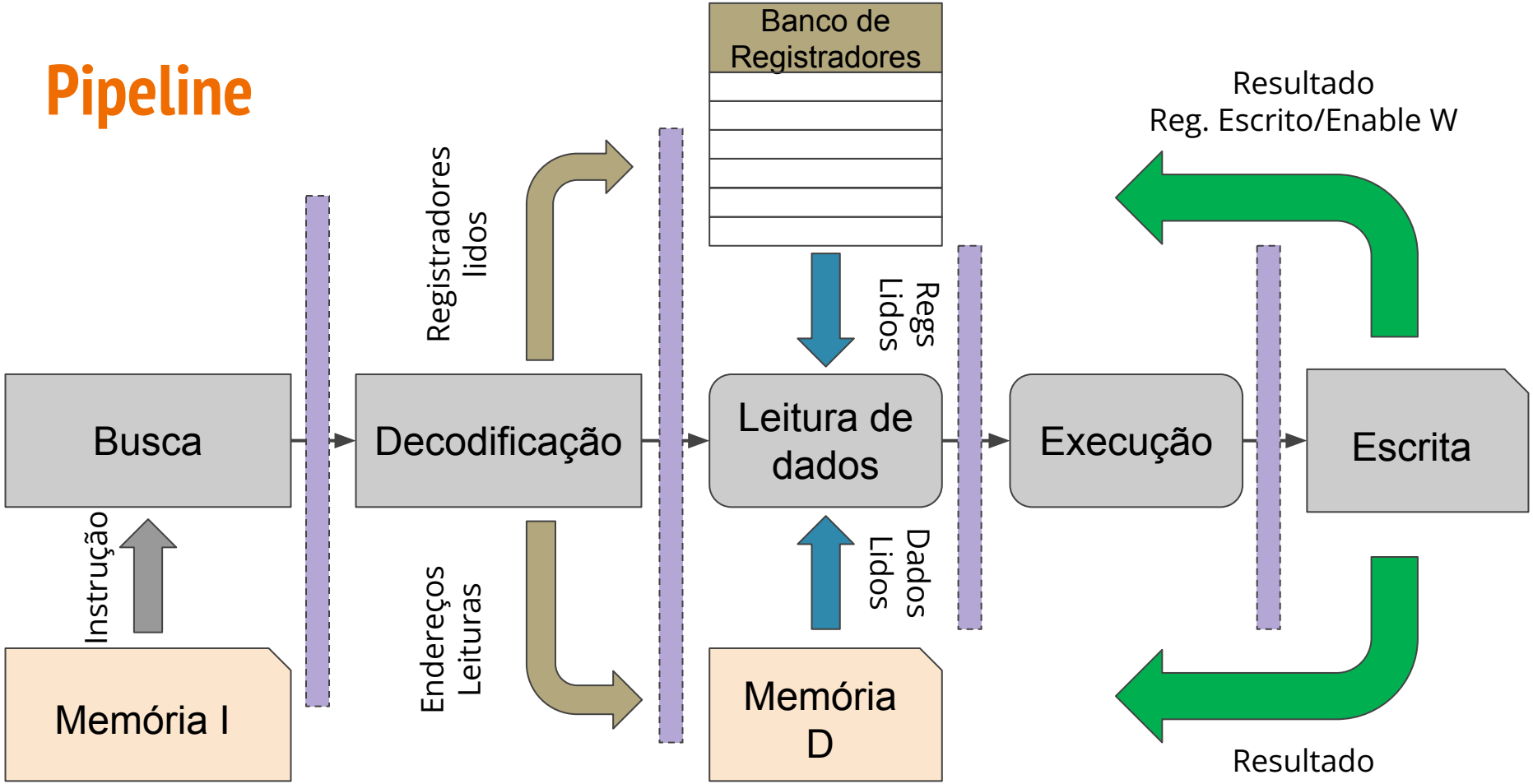
Monociclo


Banco de Registradores
Registradores lidos/escrito (Enable W)
Resultado
Regs Lidos
Busca
Decodificação
Leitura de dados
Execução
Escrita
Instrução
Endereços Leituras/Escrita (Enable W)
Dados Lidos
Resultado
Memória I
Memória D

Monociclo

- Hardware síncrono (clock)
  - Define o momento de leitura e escrita de dados
  - Cada instrução gasta 1 ciclo de relógio (clock) para executar
- Sensível a uma das bordas do clock
  - Cada borda definida (subida ou descida) conclui uma instrução e inicia a próxima
  - Isso permite que elementos de memória sejam lidos e escritos no mesmo ciclo
  - A saída de um elemento de memória fornece o valor escrito em um ciclo anterior
- Período de clock
  - Duração da instrução mais longa
  - Abordagem usada para determinar quando os dados estão estáveis para serem salvos

Pipeline


Banco de Registradores
Resultado
Reg. Escrito/Enable W
Registradores lidos
Regs Lidos
Busca
Decodificação
Leitura de dados
Execução
Escrita
Instrução
Endereços Leituras
Dados Lidos
Memória I
Memória D
Resultado
Reg. Escrito/Enable W

Pipeline

- Técnica de *hardware* que permite que a CPU execute simultaneamente diversas instruções
- O processamento de cada instrução é subdividido em etapas, sendo que cada etapa é executada por uma porção especializada da CPU
- Não reduz a latência de execução de uma única instrução
- Aumenta o throughput (vazão) de todo o workload

Pipeline

Para coordenar as operações é utilizado um sinal elétrico periódico: Clock

- A cada período de clock, uma instrução passa de um estágio a outro do pipeline
- O período de clock é determinado pelo estágio mais lento do pipeline

Deve-se procurar dividir a execução em estágios com o mesmo tempo de execução

Pipeline

O tempo gasto no processamento:

- **M** instruções
- Pipeline com **K** estágios
- Período de clock **t**

$$T = [K + (M - 1)] * t$$

Monociclo

O tempo gasto no processamento:

- **M** instruções
- Período de clock **t**

$$T = M * t$$

Exemplo

Um programa tem 1.000.000 de instruções.

Em uma arquitetura sem pipeline, o tempo de execução de cada instrução é 6,5 ns.

Qual a aceleração (speedup) na execução deste programa em um processador com pipeline de 5 estágios com ciclo de 2 ns?

Pipeline:

- **T = [K + (M –1)] * t**

Monociclo:

- **T = M * t**

$$\text{Speedup} = \frac{\text{Tempo antigo}}{\text{Tempo novo}}$$

Exercício (1)

Um programa tem 720.512 instruções.

Em uma arquitetura sem pipeline, a frequência de clock é de 2 MHz.

Qual a aceleração (speedup) na execução deste programa em um processador com pipeline de 8 estágios com frequência de 1 GHz?

Pipeline:

- **T = [K + (M –1)] * t**

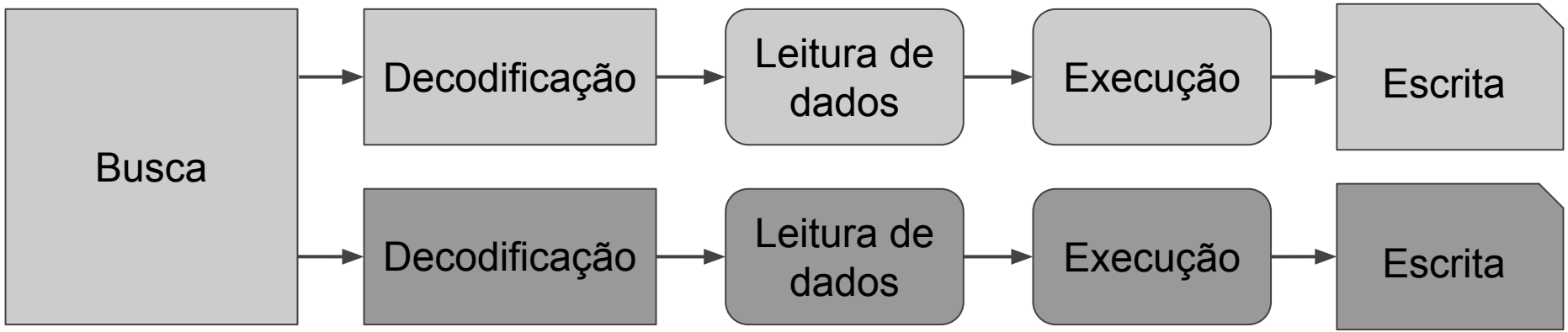
Monociclo:

- **T = M * t**

$$\text{Speedup} = \frac{\text{Tempo antigo}}{\text{Tempo novo}}$$

Pipeline superescalar

Pipeline superescalar



- Pipelining continua ativo
- IPC ideal > 1
- Instruction Level Parallelism (ILP)

Pipeline superescalar - Execução

Inst.	Estágio do pipeline					
1	B	D	L	Ex	Es	
2	B	D	L	Ex	Es	
3		B	D	L	Ex	Es
4		B	D	L	Ex	Es
Ciclo de clock	1	2	3	4	5	6

Pipeline superescalar

- Uma CPU com N pipelines é capaz de executar N instruções simultaneamente
- 1 unidade de busca de instruções busca pares de instruções, colocando cada uma delas em seu próprio pipeline completo
- Instruções paralelas não devem ter conflito de utilização de recursos
- Instruções paralelas não devem ter dependências de resultado
- Muito *hardware* duplicado para cada novo pipeline

Pipeline superescalar

O tempo gasto no processamento de uma aplicação ideal:

- **M** instruções
- Pipeline com **K** estágios
- **N** pipelines paralelos
- Período de clock **t**

Se M < N:

- **T = K * t**

Se M >= N:

- **T = (K + ⌈(M-N) / N⌉) * t**

Pipeline superescalar - Execução

Suponha um sistema um pipeline superescalar quádruplo de 20 estágios.

Considerando uma frequência de clock de 2 Hz.

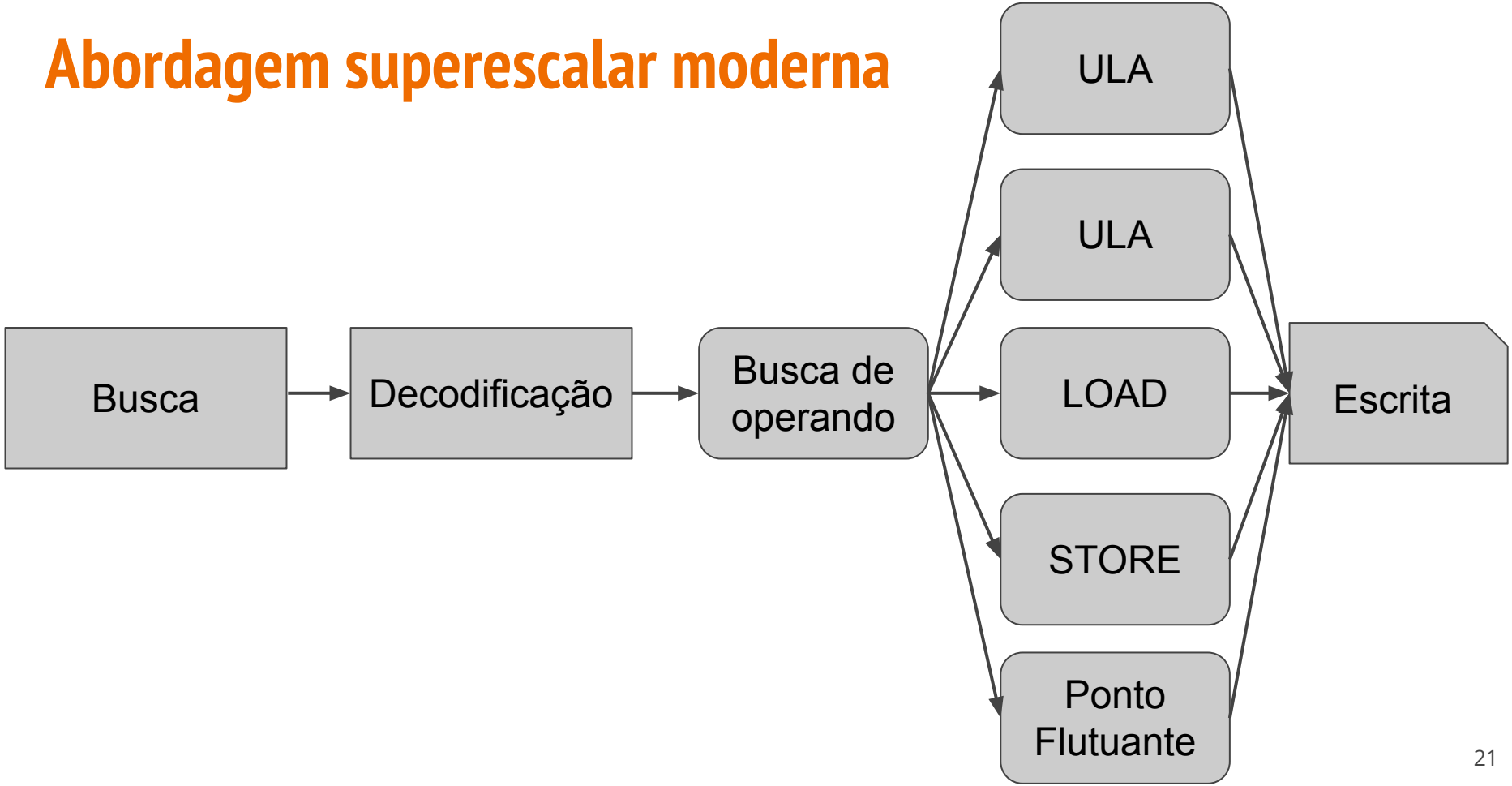
Qual o tempo de execução ideal de 100 instruções nesse sistema?

- **M** instruções
- Pipeline com **K** estágios
- **N** pipelines paralelos
- Período de clock **t**

Se M >= N:

- **$T = (K + \lceil (M-N) / N \rceil) * t$**

Abordagem superescalar moderna



Abordagem superescalar moderna

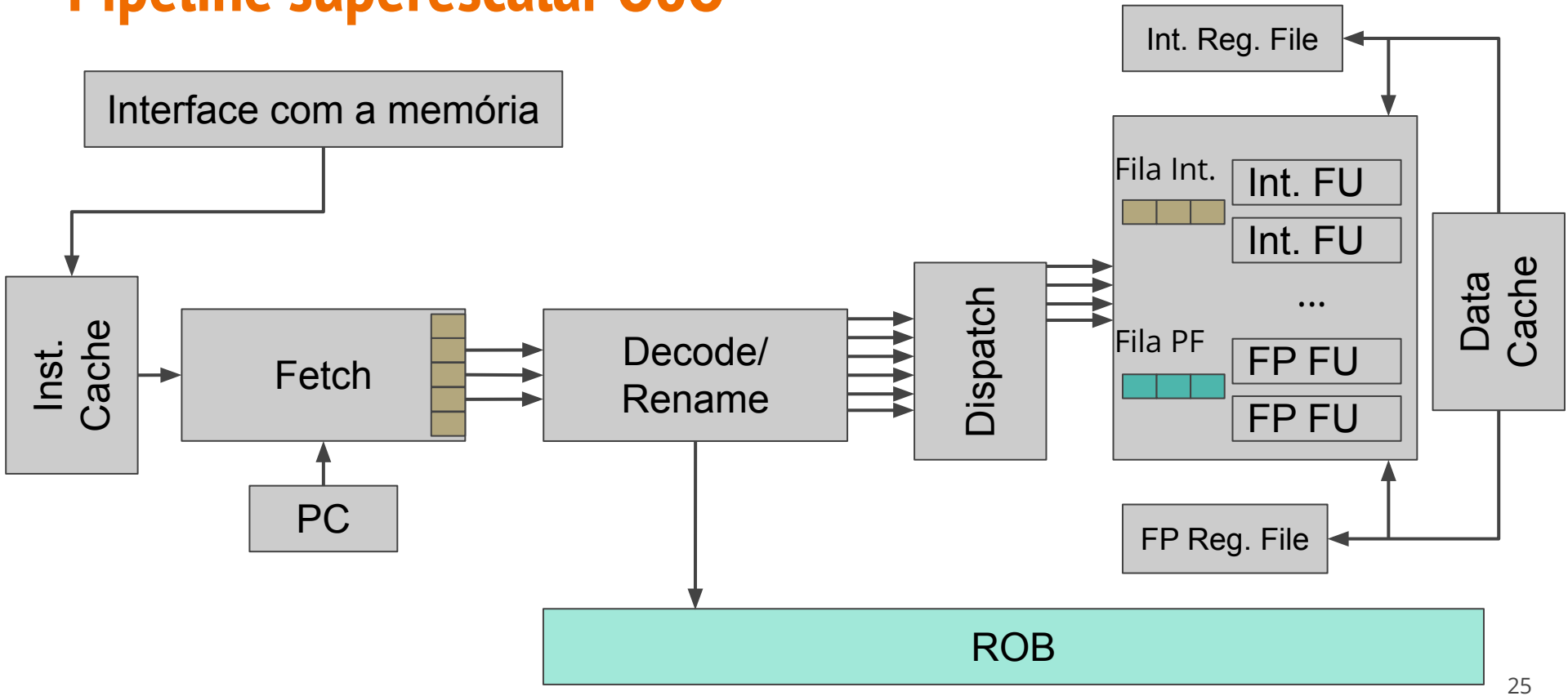
- Múltiplas instruções emitidas em um único ciclo de clock (4~6)
- Múltiplas instruções finalizadas em um único ciclo de clock (4~6)
- Um único pipeline com diversas Unidades Funcionais (UFs)
- As UFs podem levar vários ciclos para terminar a execução de uma instrução
- Os demais estágios emitem instruções mais rapidamente do que as UFs são capazes de executar

Pipeline superescalar Fora de Ordem (OoO)

Abordagem superescalar moderna

- Múltiplas instruções emitidas em um único ciclo de clock (4~6)
- Múltiplas instruções finalizadas em um único ciclo de clock (4~6)
- Um único pipeline com diversas Unidades Funcionais (UFs)
- As UFs podem levar vários ciclos para terminar a execução de uma instrução
- Os demais estágios emitem instruções mais rapidamente do que as UFs são capazes de executar
- **Instruções podem ser reordenadas antes de executar, evitando atrasos por dependências**

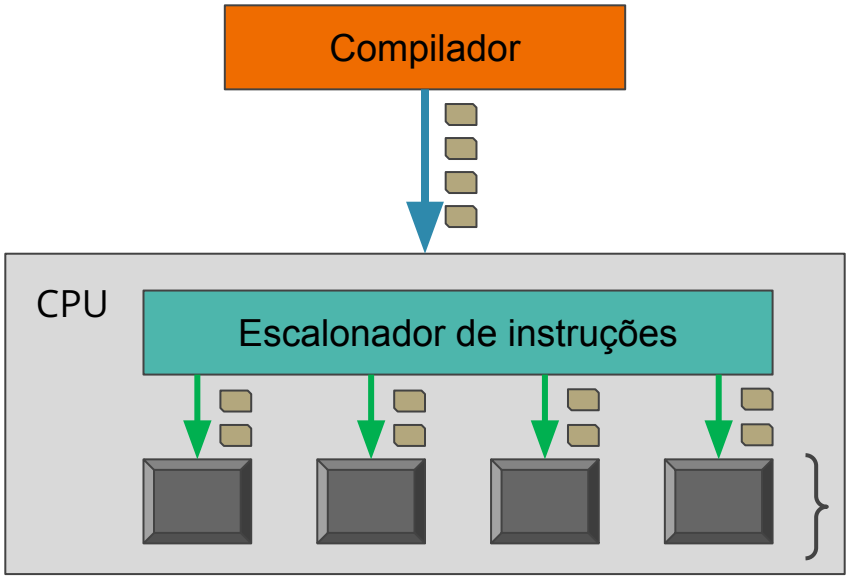
Pipeline superescalar OoO



Pipeline superescalar OoO

Fornece paralelismo a nível de instruções

- IPC ideal > 1
- Instruction Level Parallelism (ILP)

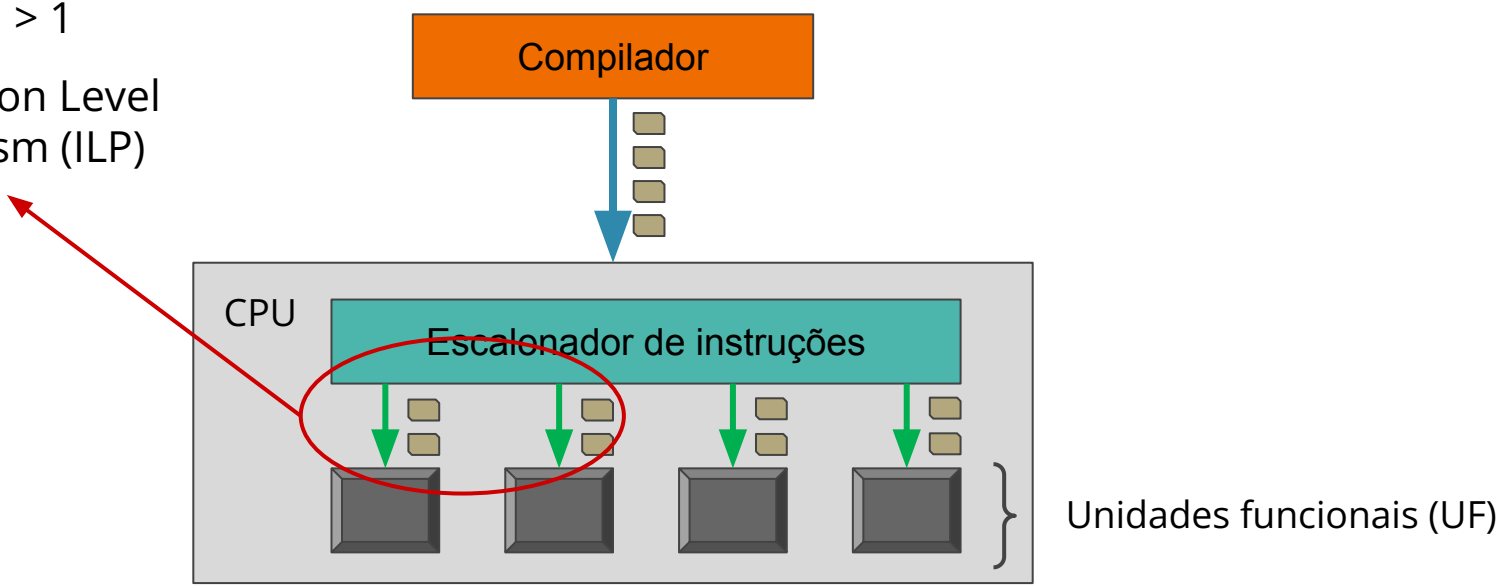


Unidades funcionais (UF)

Pipeline superescalar OoO

Fornece paralelismo a nível de instruções

- IPC ideal > 1
- Instruction Level Parallelism (ILP)



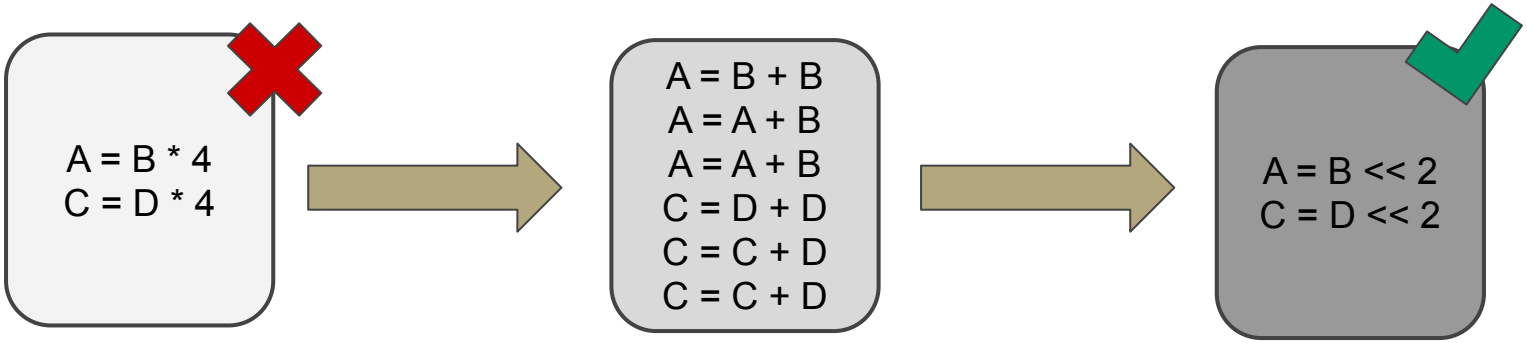
O que é mais rápido?

```
A = B * 4
C = D * 4
```

```
A = B + B
A = A + B
A = A + B
C = D + D
C = C + D
C = C + D
```

```
A = B << 2
C = D << 2
```

O que é mais rápido?



Como melhorar o desempenho em um processador (1)

Usando melhores **algoritmos** e **estruturas de dados**

- Reduzir a complexidade do algoritmo é fundamental

Reduzindo a **latência das instruções**

- Usar variáveis de 64 bits (8 bytes) apenas quando estritamente necessário
- Preferir instruções simples, sempre que possível

Como melhorar o desempenho em um processador (2)

Reduzindo a **dependência de dados**

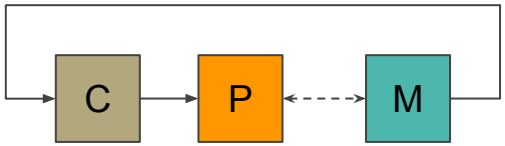
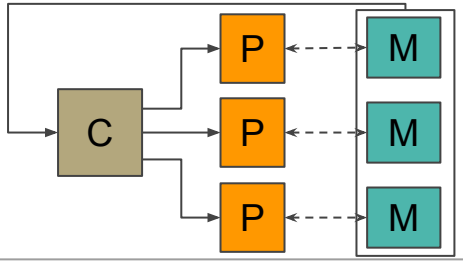
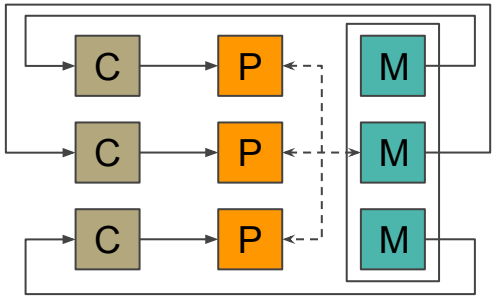
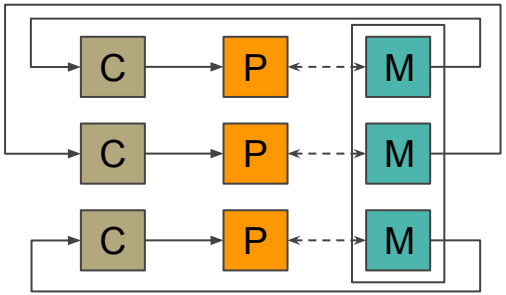
- Pensar em estruturas que não tenha muitas dependências de dados (ex. Array vs. Lista encadeada)

Reduzindo a **dependência de controle**

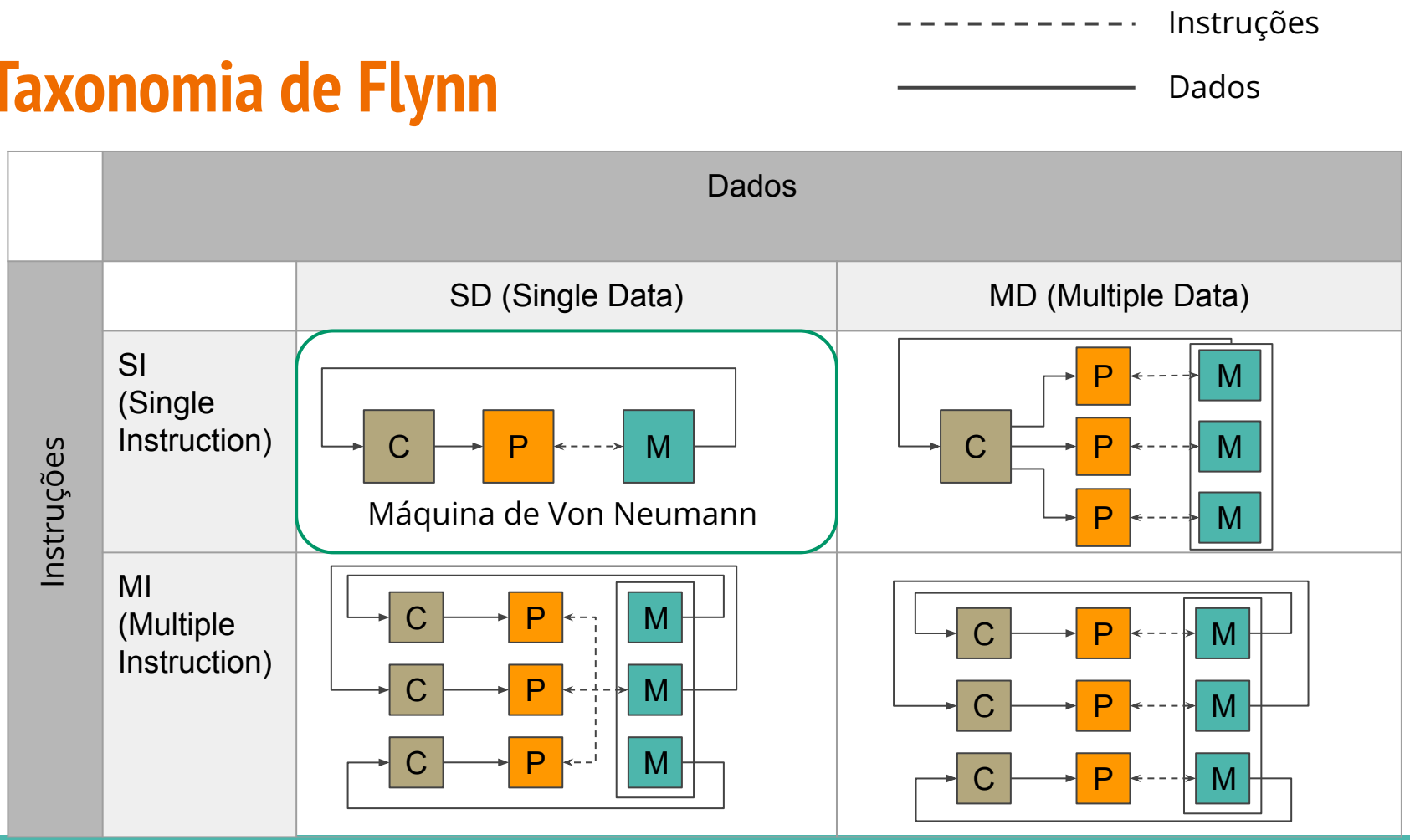
- Evitando usar muitos IFs (ex. remover IFs de dentro de laços)
- Evitando usar muitos Switch-Case
- Simplificando as condições (ex. colocar a primeira condição do IF, aquela que for mais restritiva.

Classificando arquiteturas

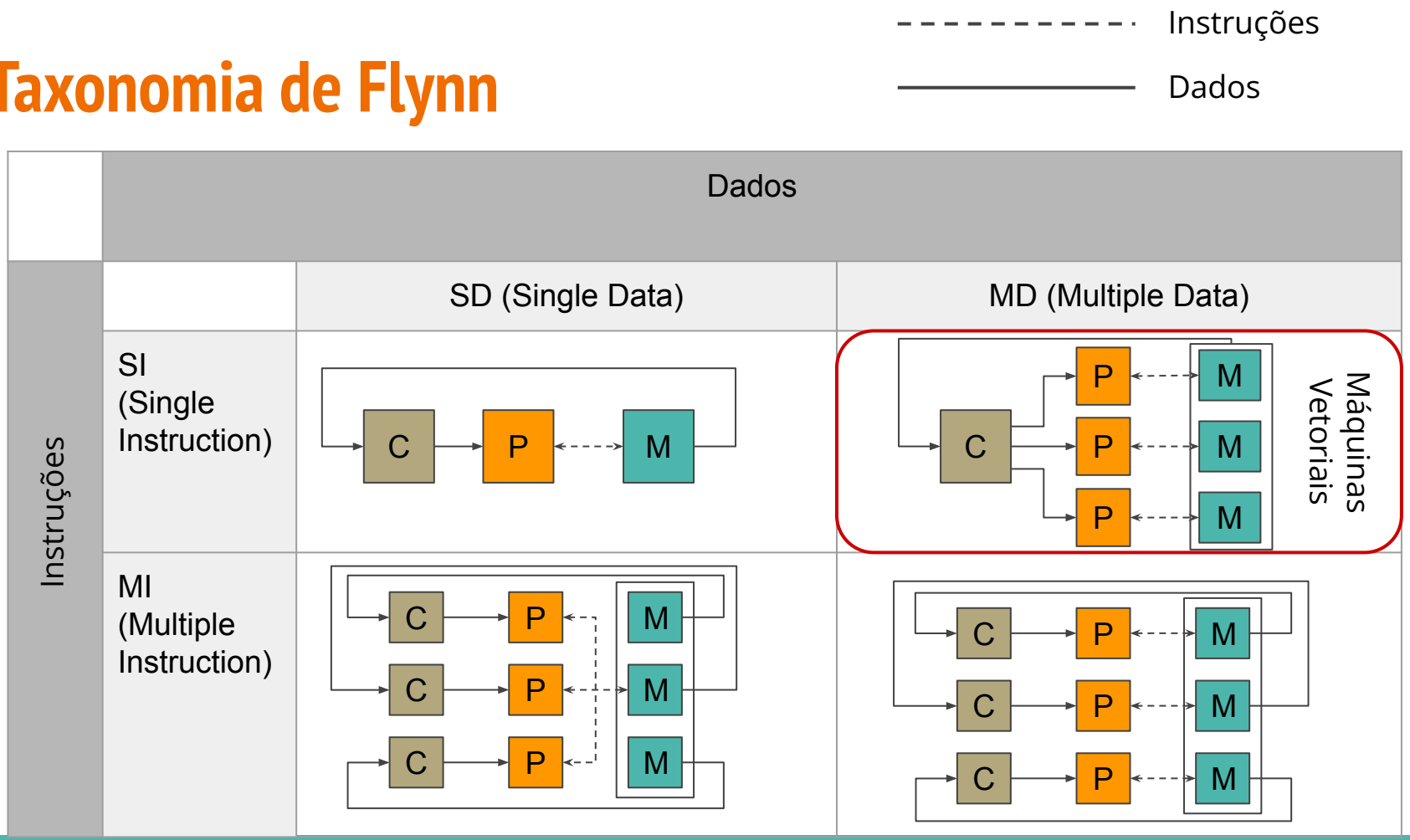
Taxonomia de Flynn

Instruções	Dados: SD (Single Data)	Dados: MD (Multiple Data)
SI (Single Instruction)		
MI (Multiple Instruction)		

Taxonomia de Flynn



Taxonomia de Flynn



Máquinas SIMD

Processador opera sobre vetores de dados

Controle único

- 1 Contador de programa
- 1 Bloco de controle
- 1 Instrução sendo executada

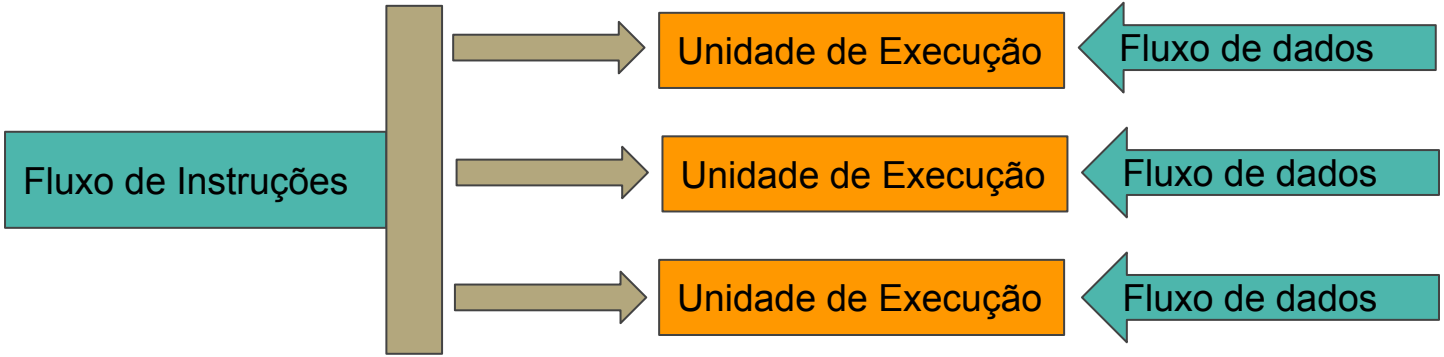
Múltiplas unidades de execução (blocos operacionais)

- ALU
- Registradores de dados
- Registradores de endereço
- Memória de dados local
- Interconexões com unidades vizinhas

Máquinas SIMD (1)

Processador hospedeiro SISD

- Executa operações sequenciais
- Calcula endereços
- Acessa memória de instruções



Eficiência de arquiteturas SIMD

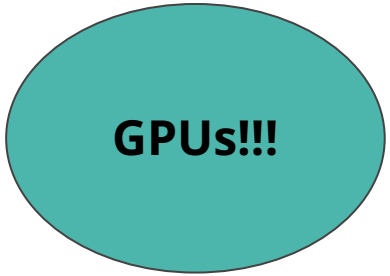
[+] SIMD são **eficientes** quando processam arrays em laços “**for**”

- Eficientes quando aplicação tem paralelismo de dados massivo

[-] SIMD são **ineficientes** em aplicações do tipo “**if/case**”

- Cada unidade de execução executa operação diferente, dependendo do dado

Processadores vetoriais



Caso aplicado de máquinas SIMD

“Processador vetorial” com pipeline associado a um processador hospedeiro escalar convencional

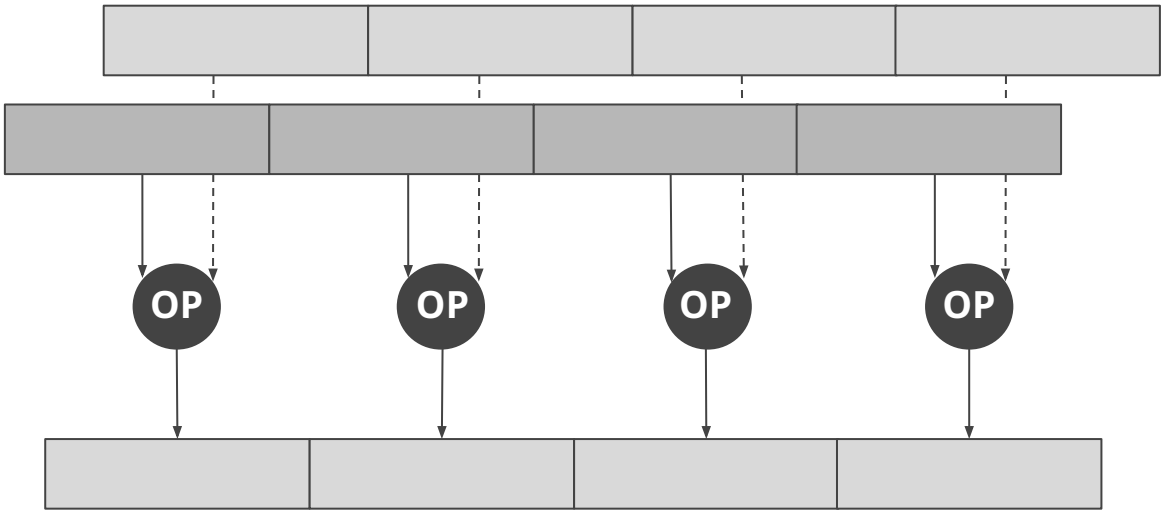
Processador vetorial ...

- Busca dados vetoriais de “registradores vetoriais” ou diretamente da memória
- Executa operações sobre os vetores através de 1 ou mais pipelines funcionais paralelos

Instruções SIMD

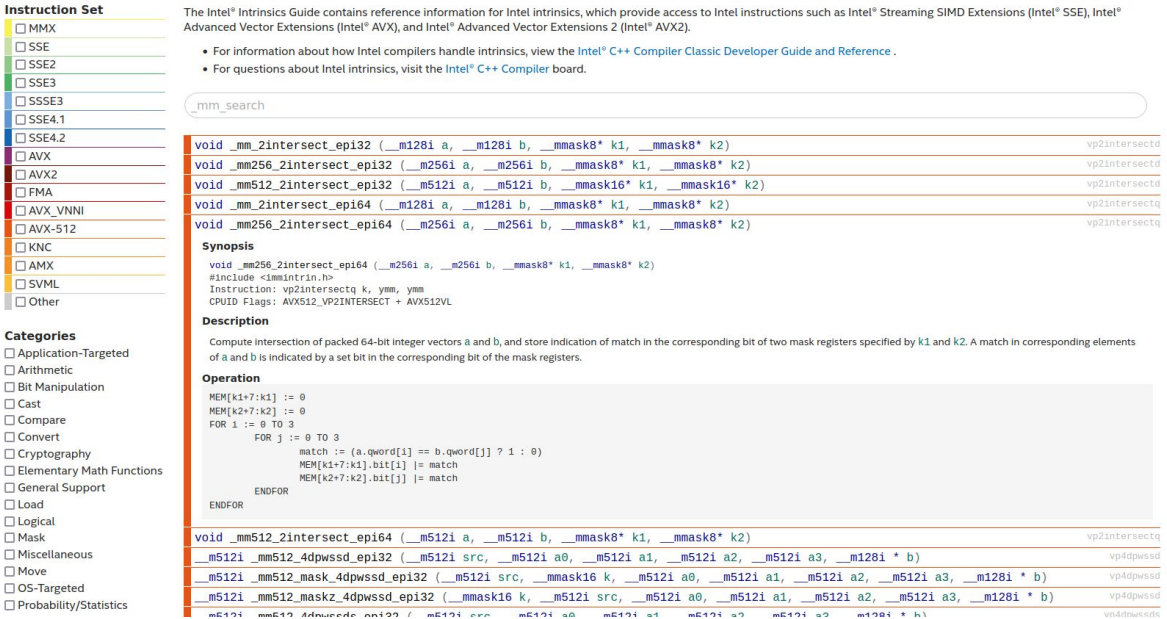
Instruções SIMD

Máquinas SISD/SIMD têm uma mistura de instruções SISD e SIMD



Instruções vetoriais da Intel

https://www.intel.com/content/www/us/en/docs/intrinsics-guide/index.html



Usando instruções vetoriais

```
#include "xmmintrin.h"

void main() {
  __m128 a, b, c;
  float va[128], vb[128], vc[128];

  for (int i=0; i<128; i+=4) {
    a = _mm_load_ps(&va[i]);
    b = _mm_load_ps(&vb[i]);
    c = _mm_add_ps(a, b);
    _mm_store_ps(&vc[i], c);
  }
}
```

Como melhorar o desempenho vetorial de um processador (1)

Utilizando bibliotecas que sejam vetorizadas

- Diversas bibliotecas python/C já apresentam formas vetorizadas
- A mesma tarefa pode ser expressa de duas formas, escalar ou vetorial

Aumentando a vazão das instruções

- Usar variáveis de 64 bits (8 bytes) apenas quando estritamente necessário
- Podemos desenrolar o laço manualmente (ex. for (i=0; i<MAX; i+=4))

Como melhorar o desempenho vetorial de um processador (2)

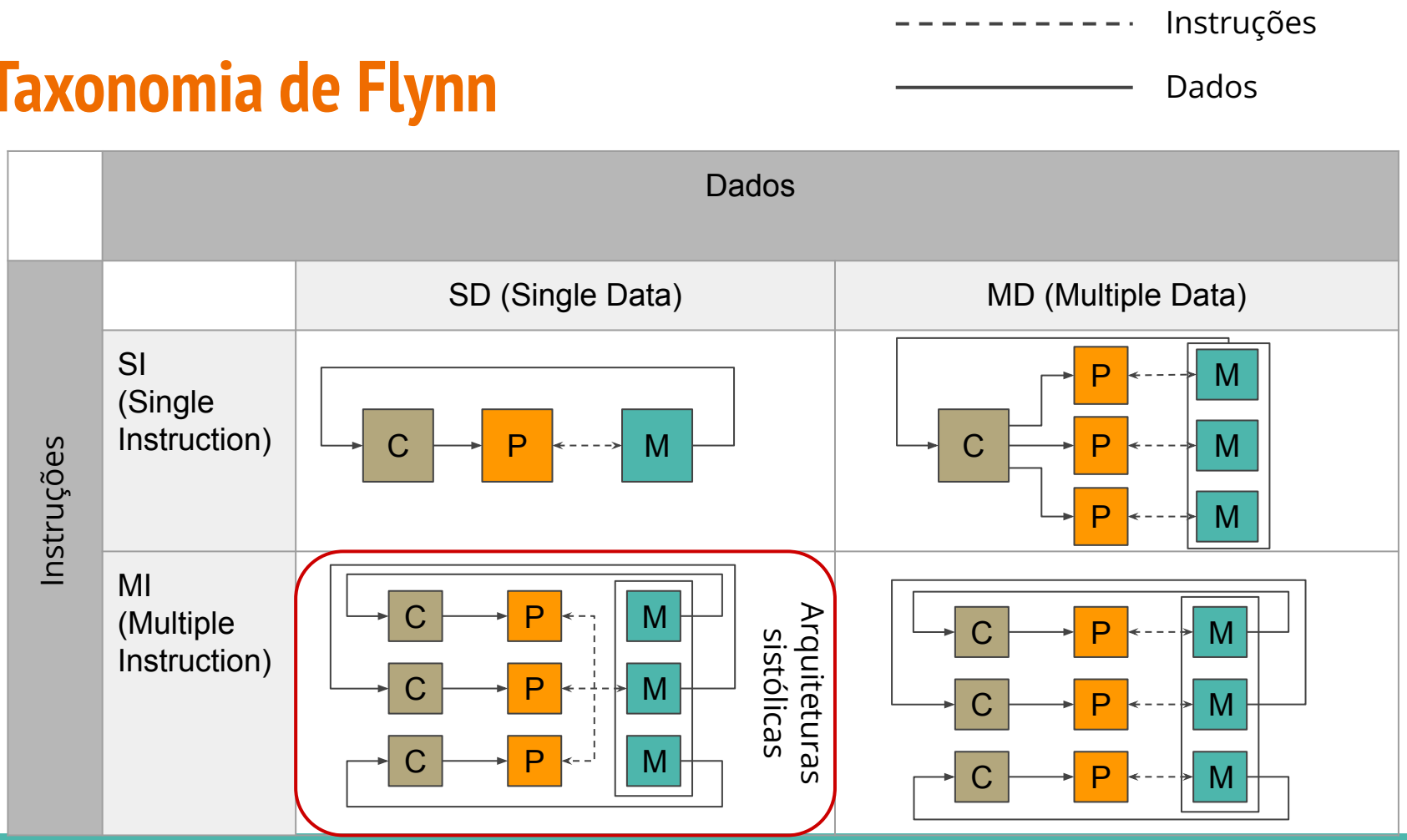
Reduzindo a dependência de dados

- Pensar em algoritmos que não tenha muitas dependências de dados (ex. Array vs. Lista encadeada)
- Removendo dependência entre iterações de um laço de repetição

Reduzindo a dependência de controle

- Evitando usar muitos IF's (ex. remover if's de dentro de laços)

Taxonomia de Flynn



Processadores sistólicos (1)

Considerado por alguns autores como máquinas MISD

Processamento “data-flow”

- Cada processador executa operação quando dados de entrada estão disponíveis

Processadores elementares

- Executam operação única, não programáveis

Processadores sistólicos (2)

Aplicações em processamento digital de sinais

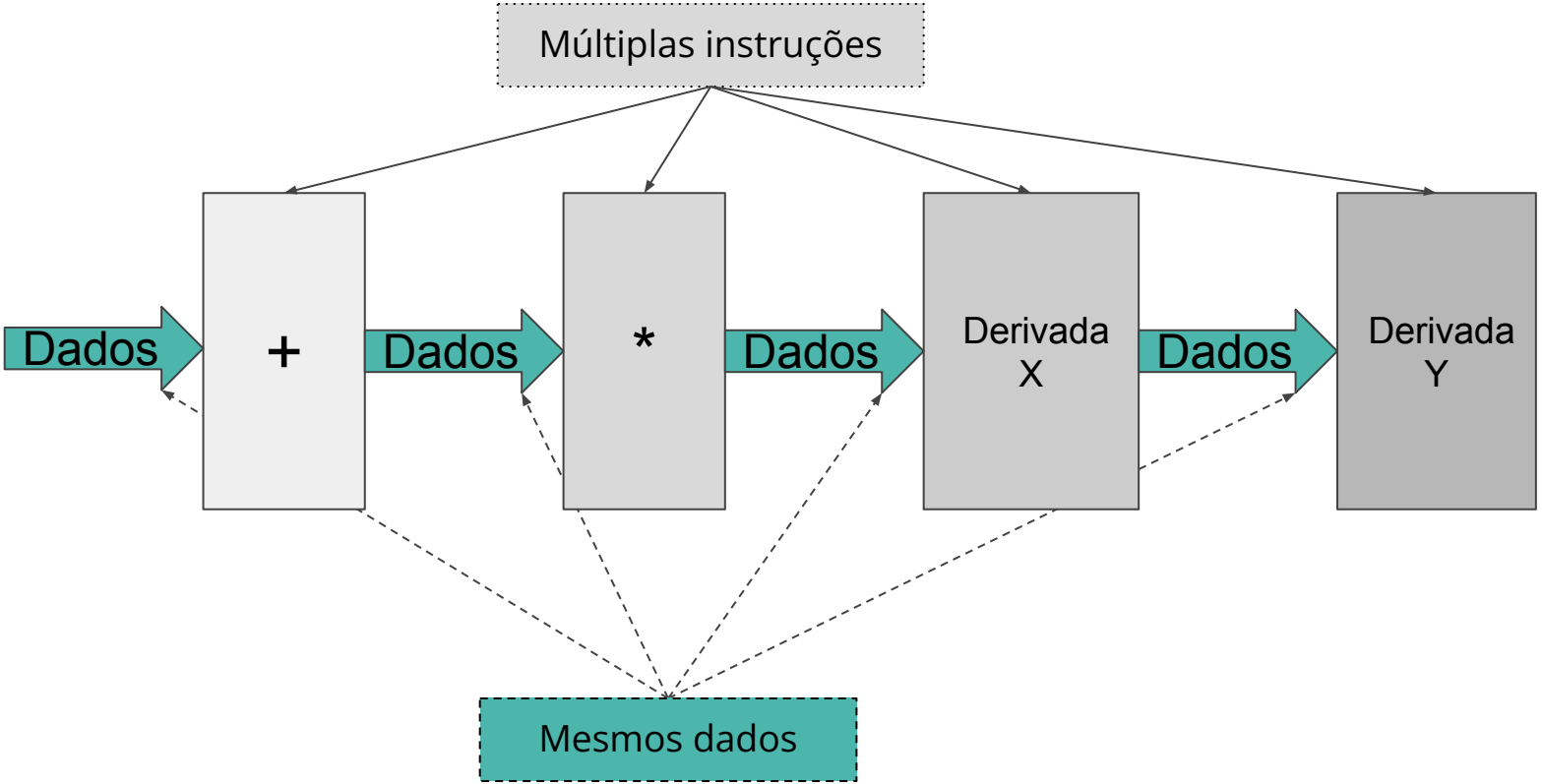
- Processamento de imagens, voz, ...

Integração num único chip

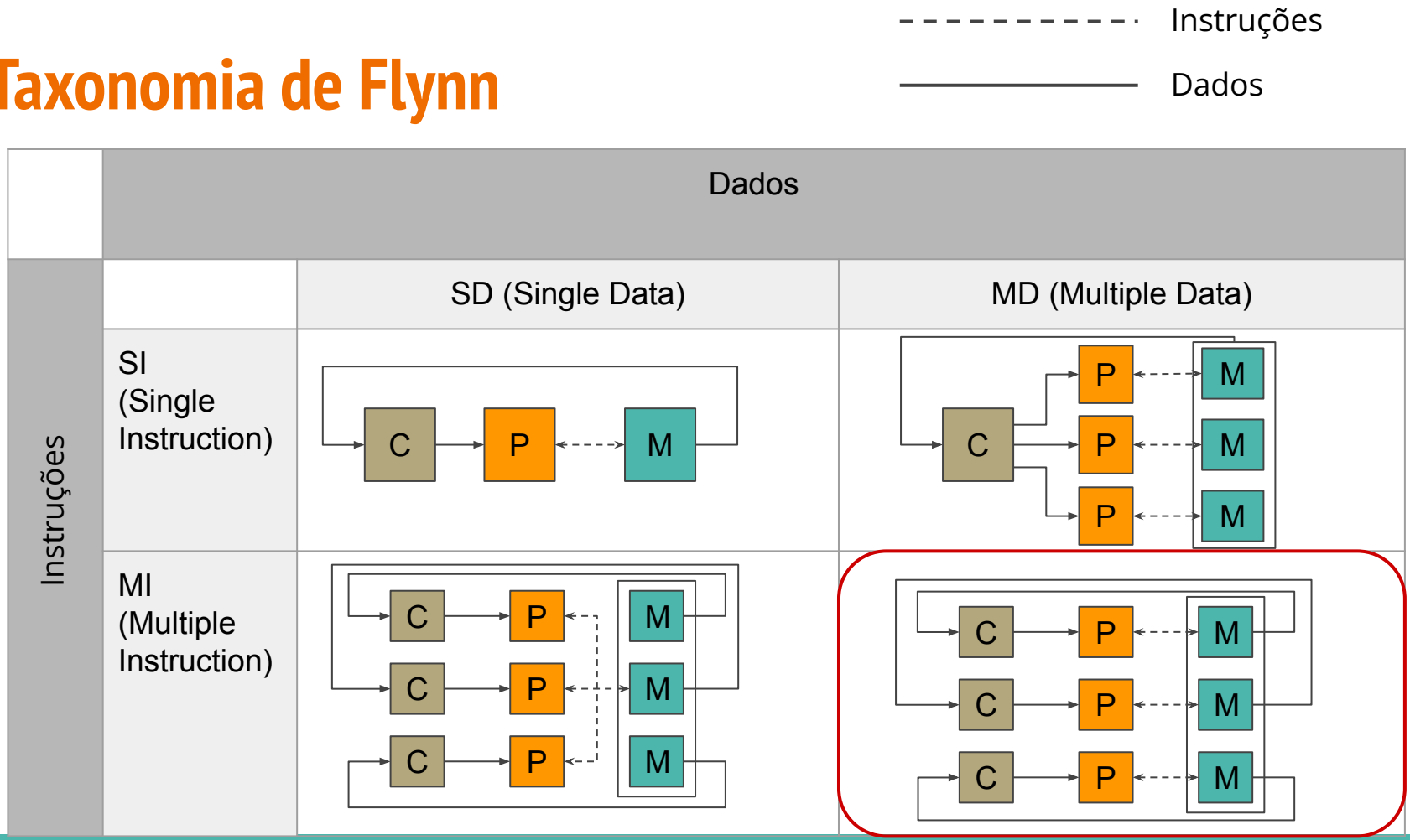
São utilizados em:

- Em sistemas eletrônicos dedicados como DSP (Digital Signal Processing)
- Como unidade funcional especializada de um processador hospedeiro convencional

Processadores sistólicos - Exemplo



Taxonomia de Flynn



Multiprocessadores/ Multicomputadores

Espaço de endereçamento privado vs. compartilhado

Espaço privado

Cada processador tem sua visão da memória.

Cada um terá sua variável privada.

Comunicação através da troca de mensagens (mais lento).

Normalmente não ocorrerão condições de corrida.

Espaço compartilhado

Todos os processadores tem a mesma visão da memória.

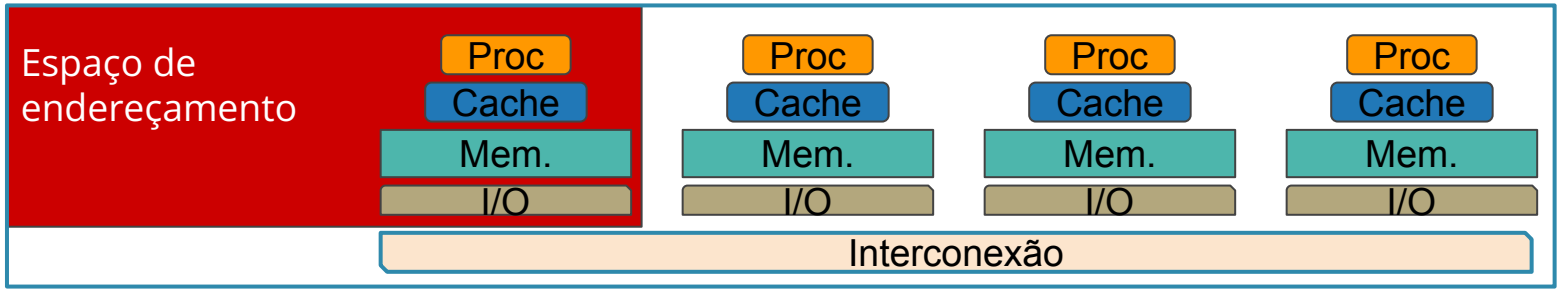
Mesma variável pode ser visível para todos.

Comunicação por variável compartilhada (mais rápido).

Haverá naturalmente condições de corrida (recursos compartilhados).

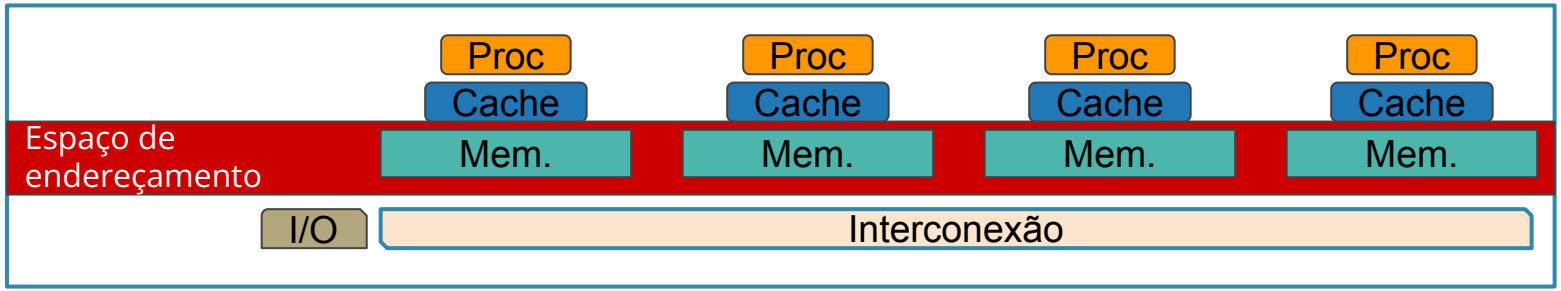
MIMD - Multicomputadores

Multicomputadores
Cluster de máquinas



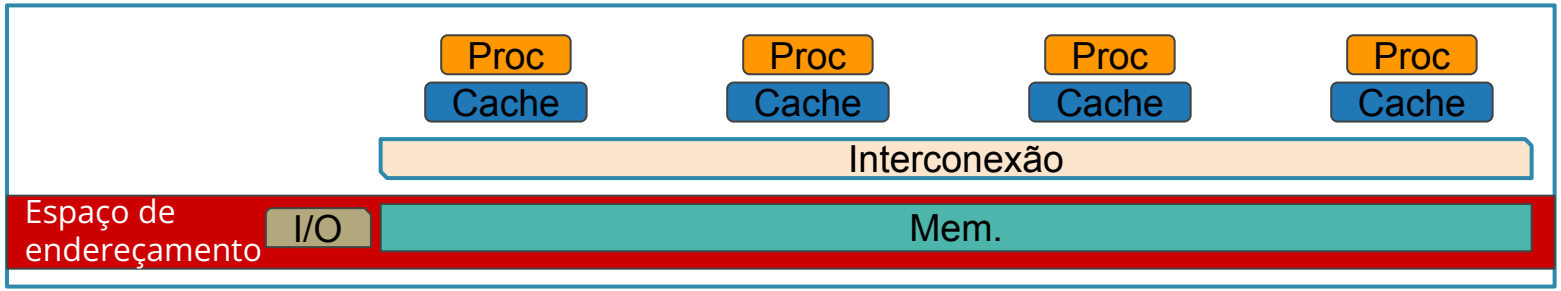
MIMD - Multiprocessadores NUMA

Multiprocessadores
Máquina NUMA

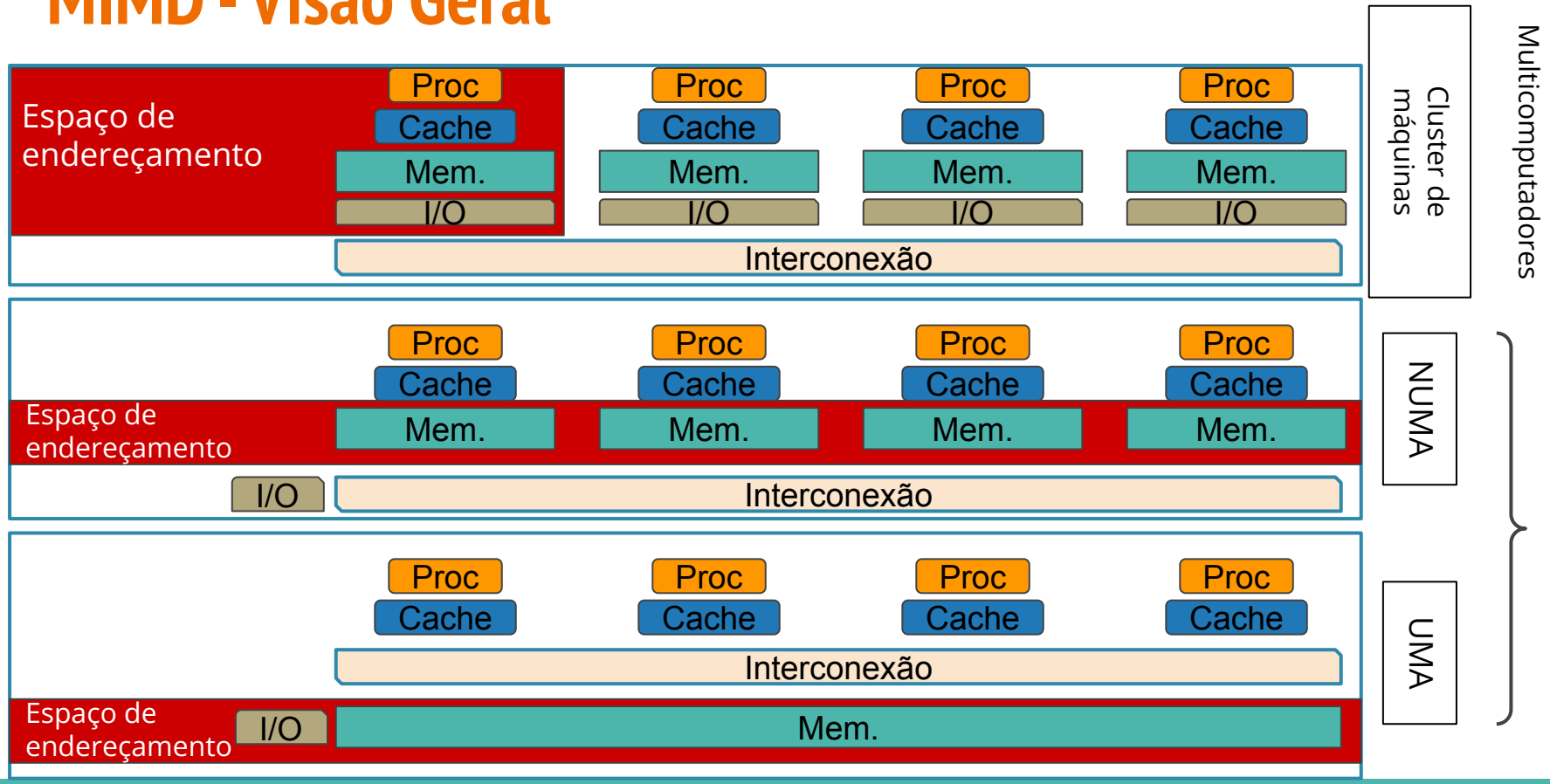


MIMD - Multiprocessadores UMA

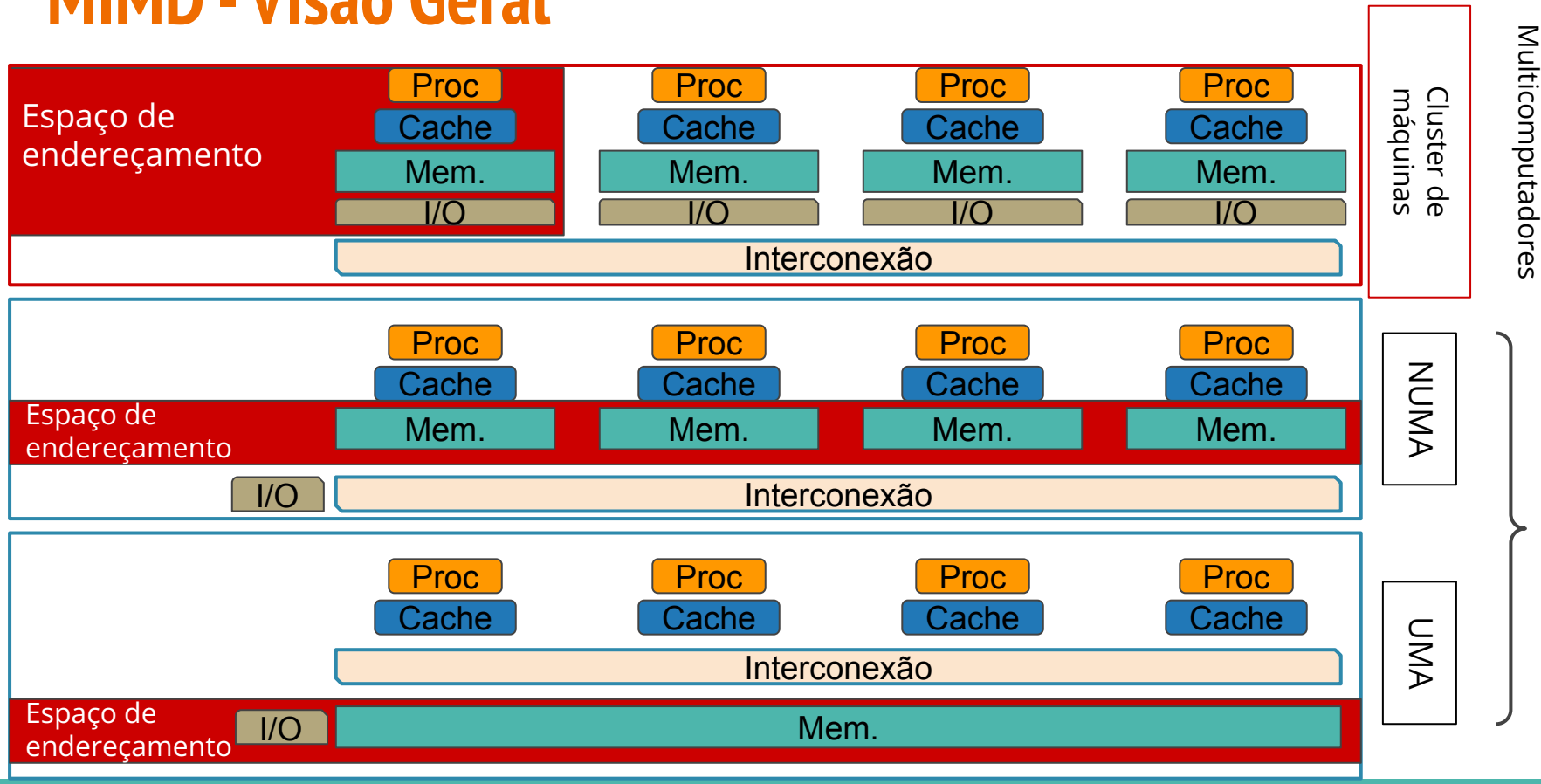
Multiprocessadores
Máquina UMA



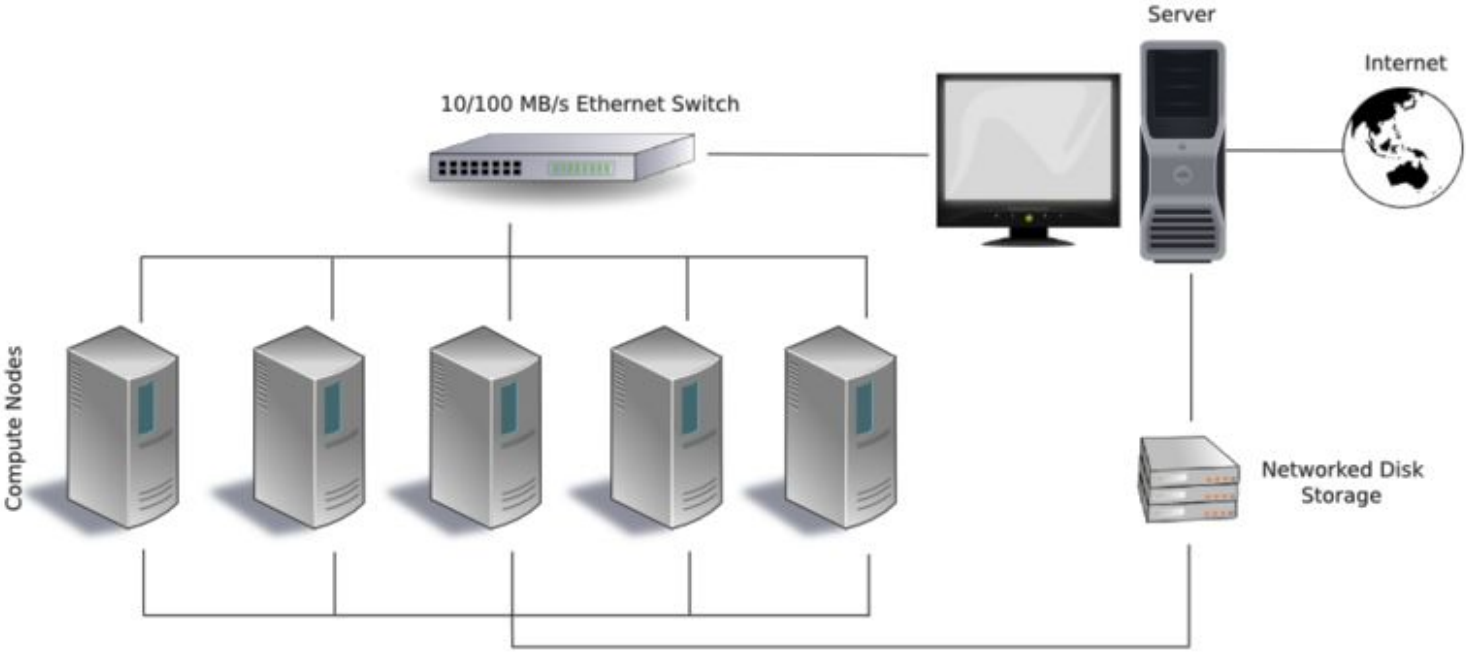
MIMD - Visão Geral



MIMD - Visão Geral



Multicomputadores - Cluster de máquinas



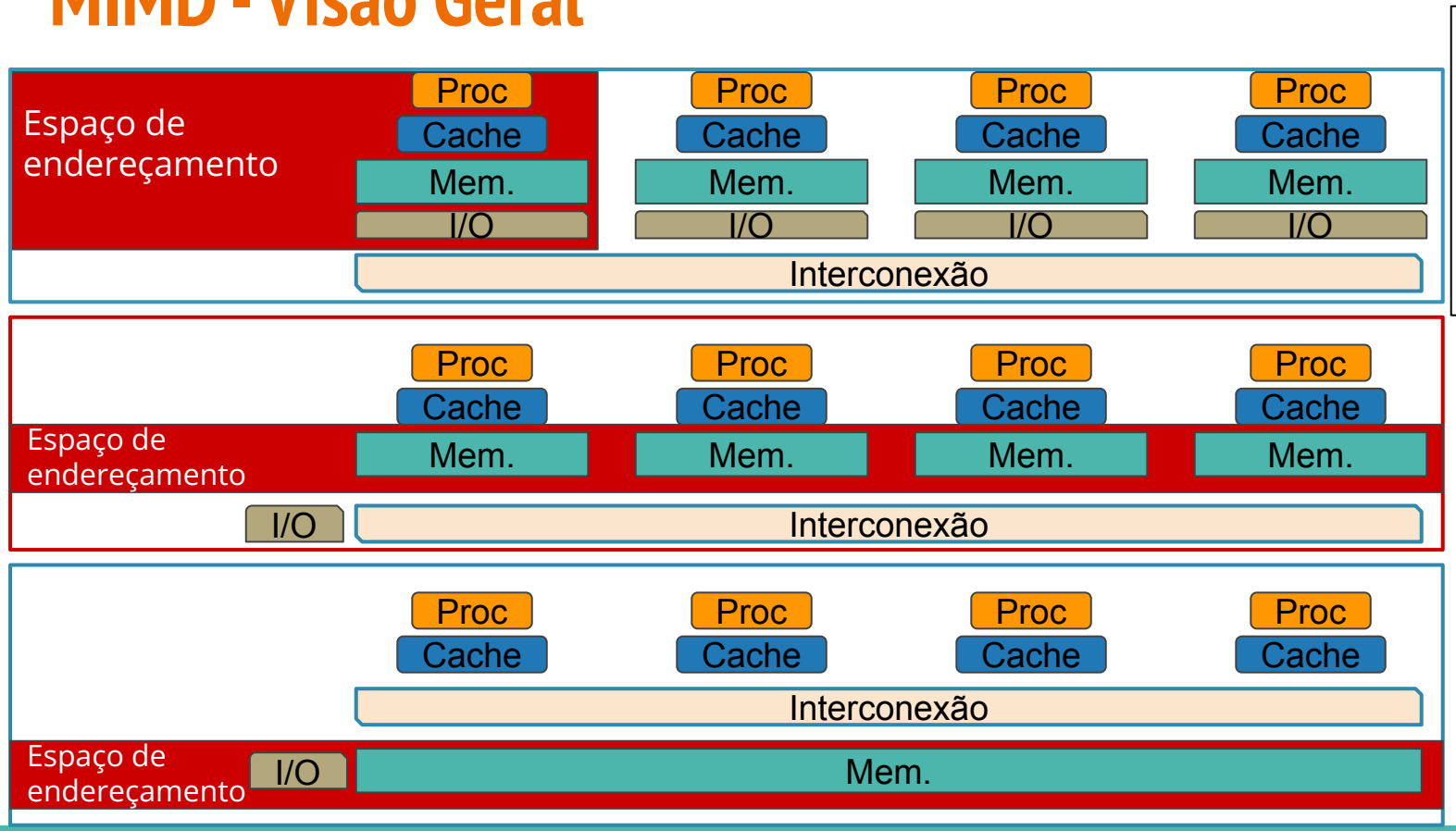
Multicomputadores - Cluster de máquinas

Conhecidos como COW/NOW: Cluster/Network of Workstations

Paralelismo: Bastante claro em nível de processos

- CPUs fracamente acopladas
- Diversos espaços de endereçamento
- SO para cada máquina: Linux ou proprietário
- Escalonador de Jobs
- Tecnologias de Rede: Ethernet, Myrinet, InfiniBand

MIMD - Visão Geral



MIMD - NUMA - Non-Uniform Memory Access (1)

Partições dos Cores e as memórias são agrupadas em Nós

CPUs fortemente acopladas

Uma máquina pode ter vários Nós NUMA, dependendo do modelo e quantidade de memória/processadores

A latência de acesso de um processador com a memória dentro de um Nó NUMA é muito baixa, pois o barramento torna eficiente o acesso dentro do mesmo Nó

MIMD - NUMA - Non-Uniform Memory Access (2)

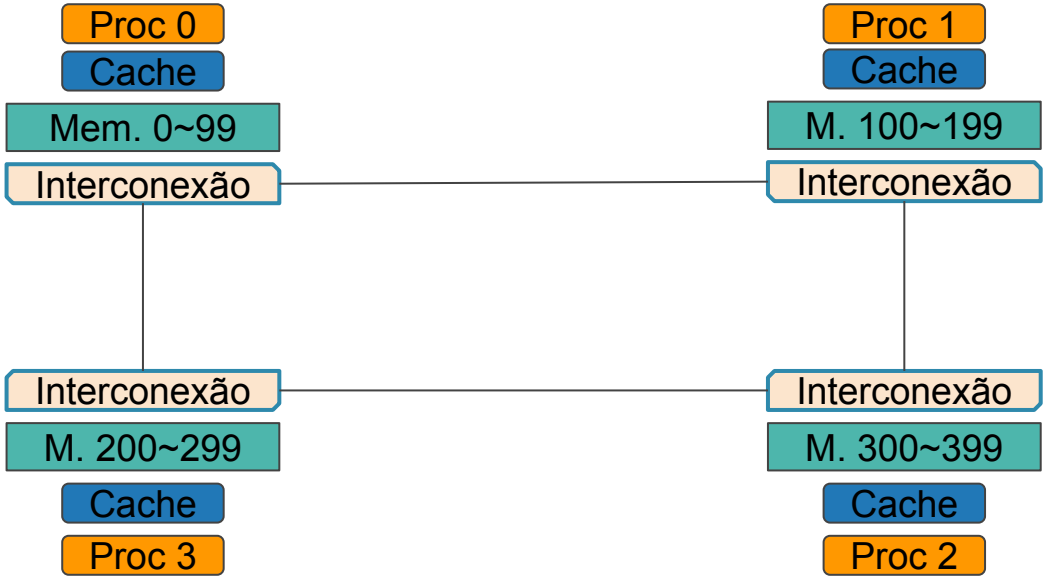
Entretanto o acesso do processador de um Nó X para a memória em um Nó Y é maior pois é necessário atravessar a interconexão que liga os diversos Nós

Pode parecer insignificante esta penalidade entretanto em acessos intensivos de memória a Nós remotos pode ter grande impacto no desempenho do sistema

A razão da diferença de latência entre o nodo local e um nodo remoto é chamado de ***NUMA factor***.

NUMA - Exemplo

Exemplo de NUMA com capacidade total de 400 páginas de memória



NUMA - Exemplo

Exemplo de NUMA com capacidade total de 400 páginas de memória

- Cada nó contém um quadro de páginas contíguo

Considerando as latências:

- Caches/Memória: **m** ciclos
- Interconexão/link: **n** ciclos

O custo do Proc 0 acessar:

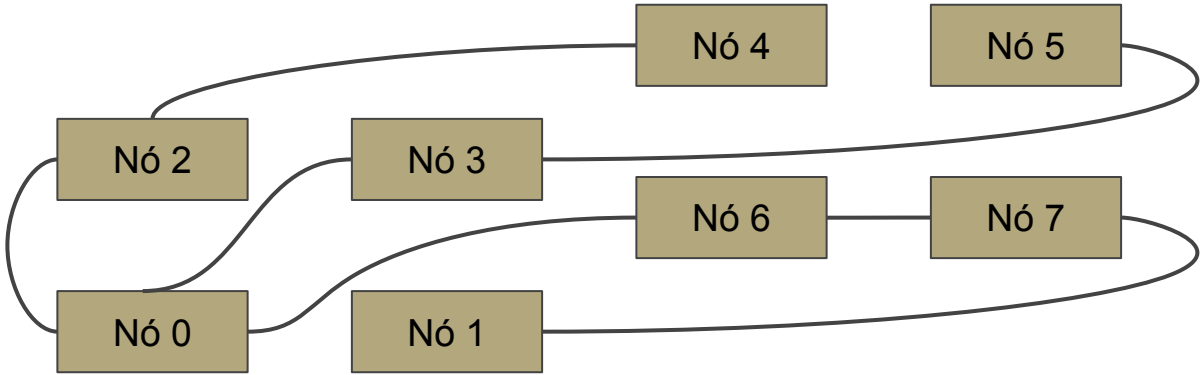
- Páginas (0~99): m ciclos
- Páginas (100~299): n + m ciclos
- Páginas (300~399): 2n + m ciclos

NUMA - Exercício

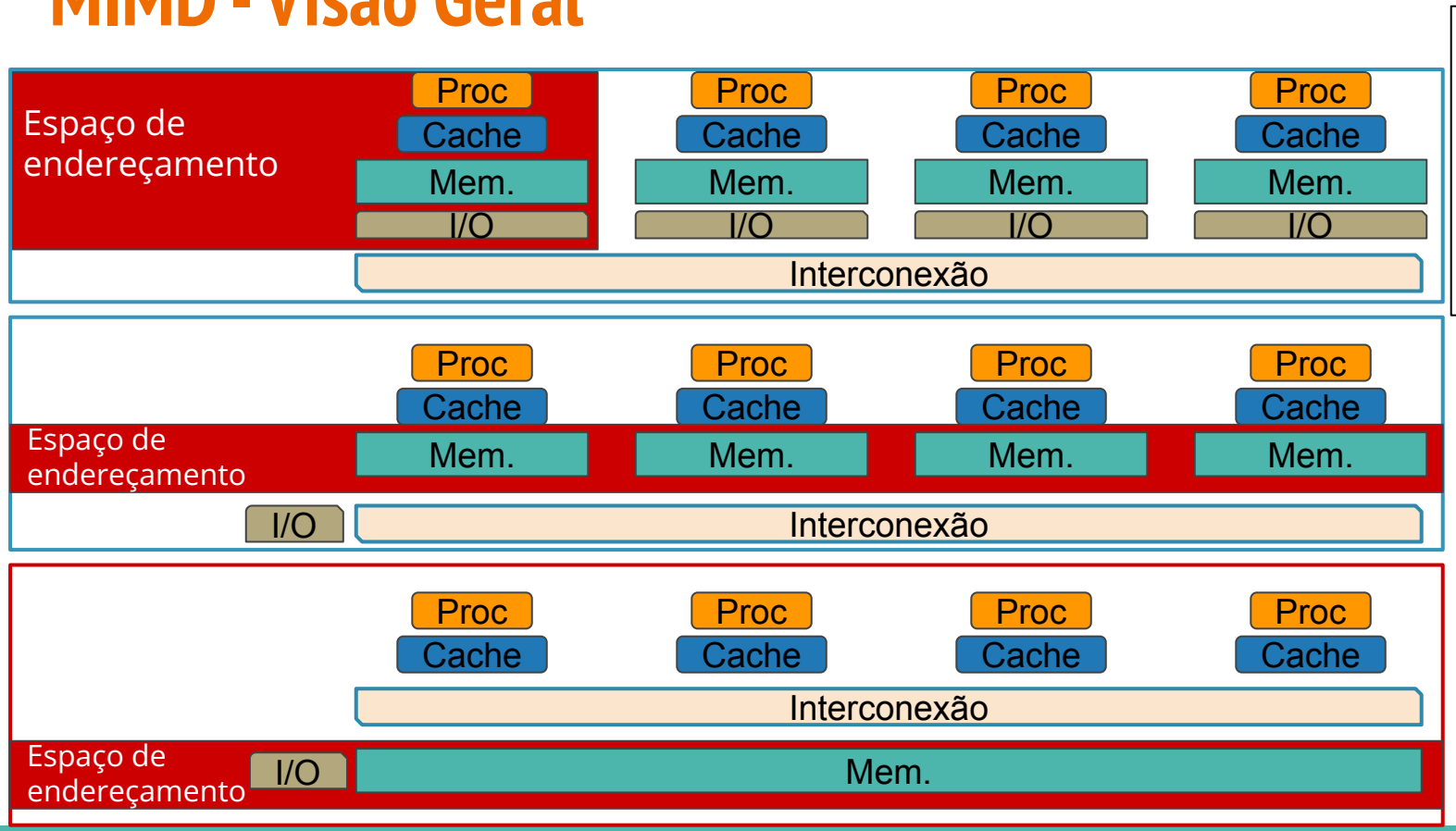
Considerando a seguinte topologia e arquitetura NUMA:

- Caches/Memória: **108** ciclos
- Interconexão/link: **72** ciclos

Qual o tempo de acesso à memória no pior cenário? Qual o NUMA factor?



MIMD - Visão Geral



Uniform Memory Access - UMA

Chip Multithreading

- Busca mascarar o tempo de espera a dados de uma aplicação
  - Busca em memória e dependências
- Permite que a CPU gerencie múltiplos threads de controle ao mesmo tempo
- Multithreading de granulação fina
- Multithreading de granulação grossa
  - Intervalo entre trocas
- Multithreading simultâneo (SMT) ou Hyper-Threading

Uniform Memory Access - UMA

Chip Multiprocessor

- O ganho de desempenho com multithreading é limitado pelo compartilhamento de recursos entre as threads
- Solução: Processadores independentes no mesmo chip
- Compartilham uma ou mais caches e memória principal
- Dobro de desempenho sem aumento significativo de custo para servidores caros

Multicomputadores x Multiprocessadores

Multicomputadores

- Mais fáceis de construir

Multiprocessadores (UMA e NUMA)

- Mais fáceis de programar

RISC vs CISC

Reduced Instruction Set Computer

- Pequeno número de instruções
- Todas as instruções comuns são executadas diretamente pelo hardware
- Maximize a taxa de execução das instruções
- Instruções devem ser fáceis de decodificar
- Somente LOAD e STORE devem referenciar a memória
- Providencie muitos registradores

Complex Instruction Set Computer

- Muitas instruções
- Instruções podem ser subdivididas em microinstruções
- Instruções de tamanho distinto