

# ARQUITETURA E ORGANIZAÇÃO DE COMPUTADORES

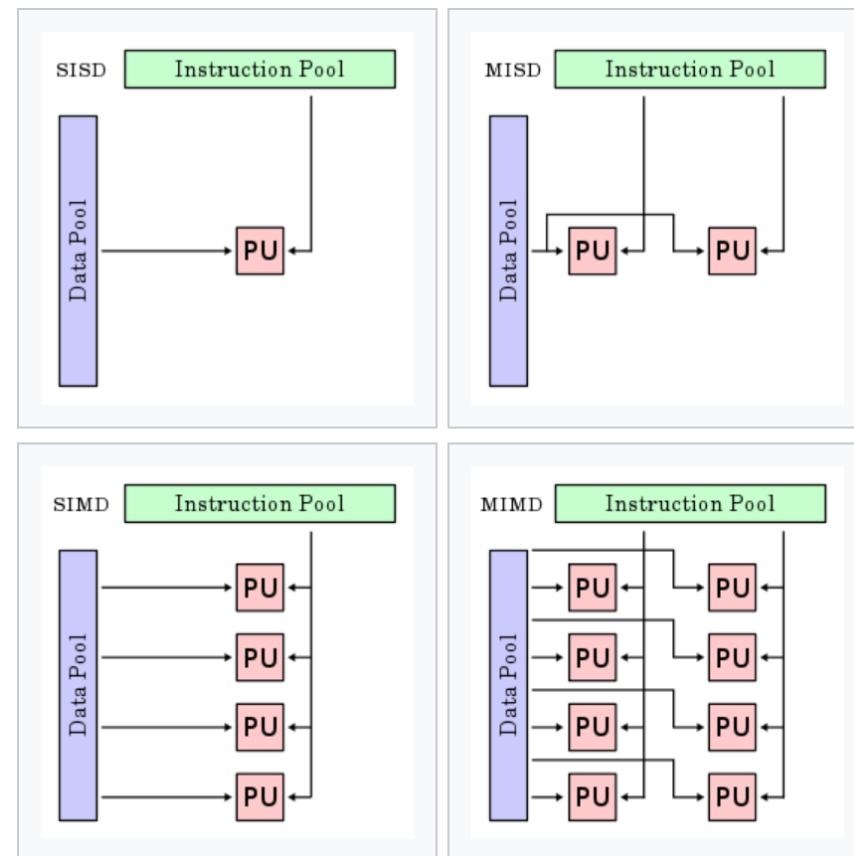
Taxonomia de Flynn

# CLASSIFICANDO OS COMPUTADORES

A taxonomia de Flynn é uma classificação de arquiteturas de computadores, proposta por Michael J. Flynn em 1966.

O sistema de classificação ficou preso e tem sido usado como uma ferramenta no design de processadores modernos e suas funcionalidades.

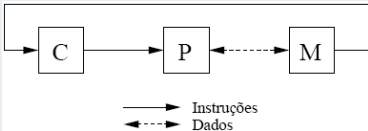
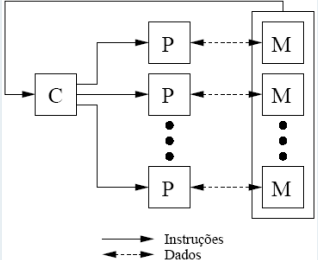
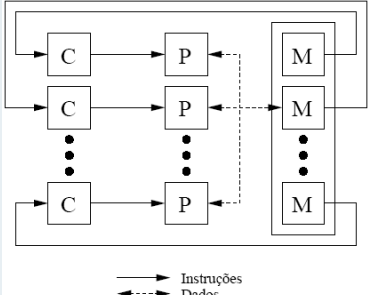
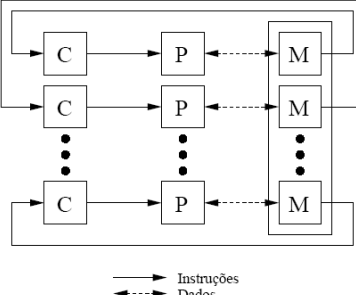
Desde o surgimento das unidades centrais de processamento multiprocessamento (CPUs), um contexto de multiprogramação evoluiu como uma extensão do sistema de classificação.



# VISITANDO A TAXONOMIA DE FLYNN

\*[Flynn, 1972]

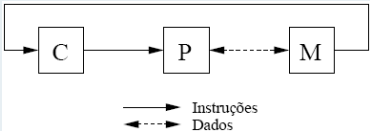
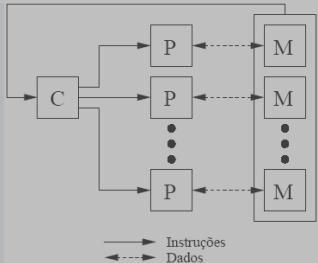
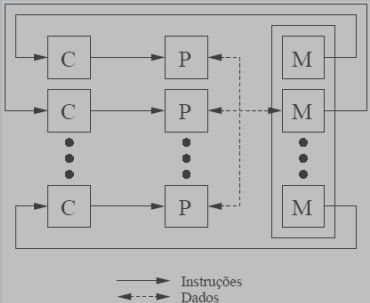
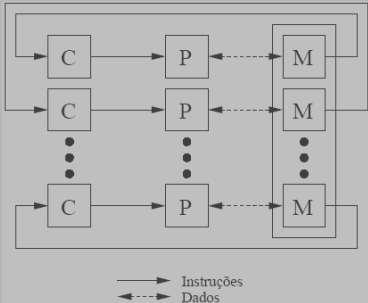
\*\*[De Rose, 2003]

*	SD (Single Data)	MD (Multiple Data)
<b>SI</b> <b>(Single Instruction)</b>	<p>* *</p>  <p>SISD (Máquinas von Neumann)</p>	<p>* *</p>  <p>SIMD (Máquinas Vetoriais)</p>
<b>MI</b> <b>(Multiple Instruction)</b>	<p>* *</p>  <p>MISD (Sem representante / Arquiteturas Sistólicas)</p>	<p>* *</p>  <p>MIMD (Multiprocessadores e Multicomputadores)</p>

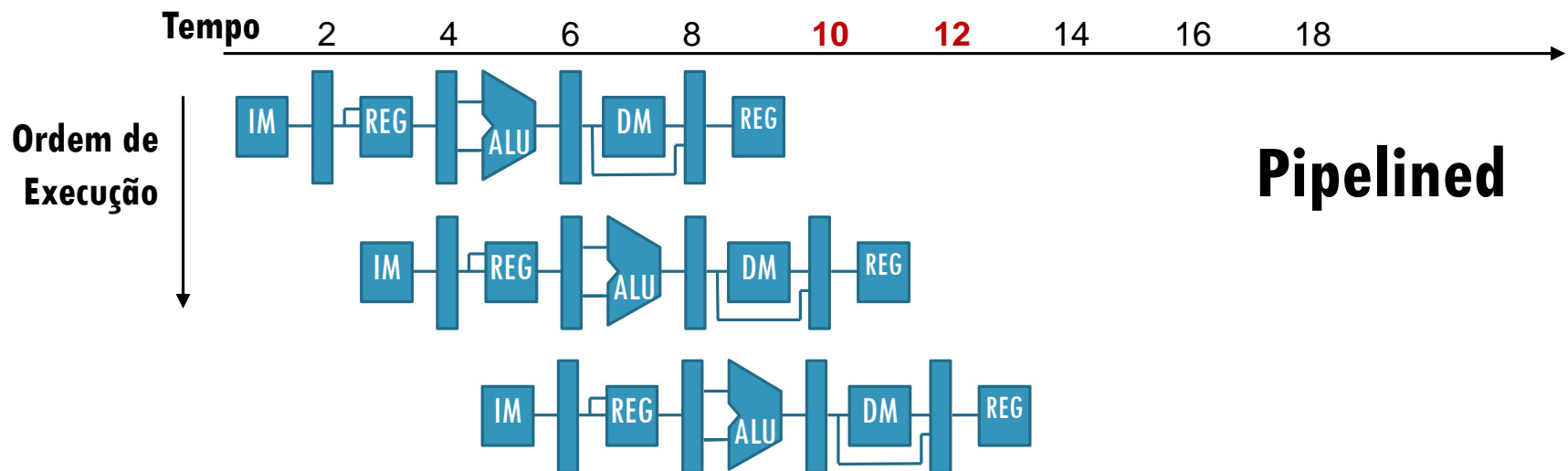
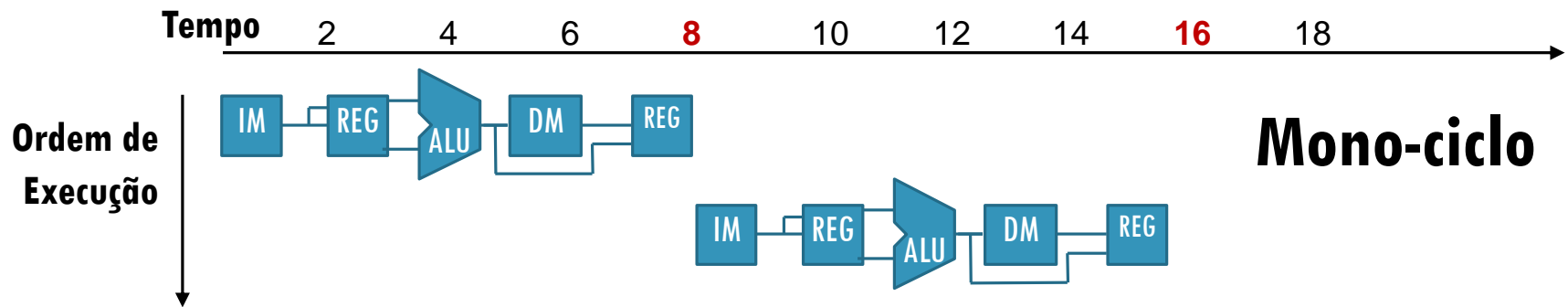
# VISITANDO A TAXONOMIA DE FLYNN

\*[Flynn, 1972]

\*\*[De Rose, 2003]

*	SD (Single Data)	MD (Multiple Data)
<p><b>SI</b> (Single Instruction)</p>	<p>* *</p>  <p>SISD (Máquinas von Neumann)</p>	<p>* *</p>  <p>SIMD (Máquinas Vetoriais)</p>
<p><b>MI</b> (Multiple Instruction)</p>	<p>* *</p>  <p>MISD (Sem representante / Arquiteturas Sistólicas)</p>	<p>* *</p>  <p>MIMD (Multiprocessadores e Multicomputadores)</p>

# MONOCICLO VS. PIPELINE



# SINGLE CORE - PIPELINING

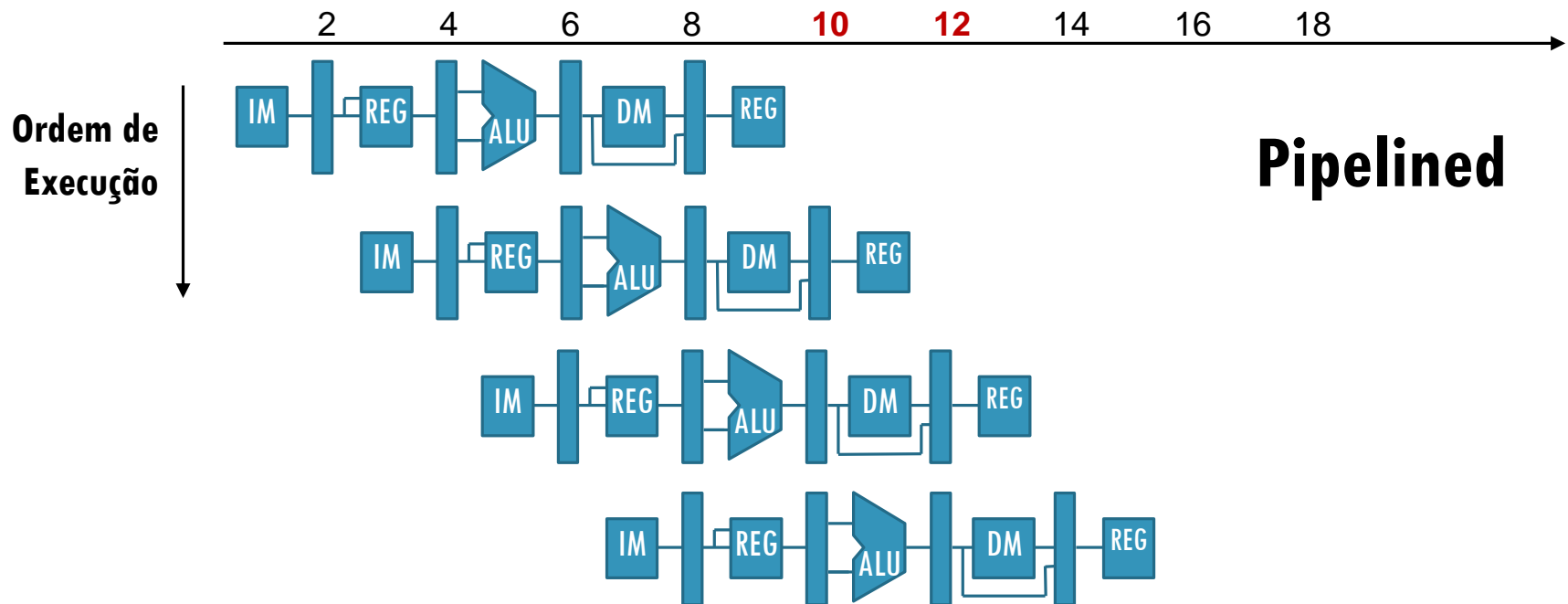
Sem Paralelismo.

Reuso do hardware.

CPI ideal = 1 (após encher o pipeline).

**Como resolvemos:**

- Dependência de recursos?
- Dependência de dados?

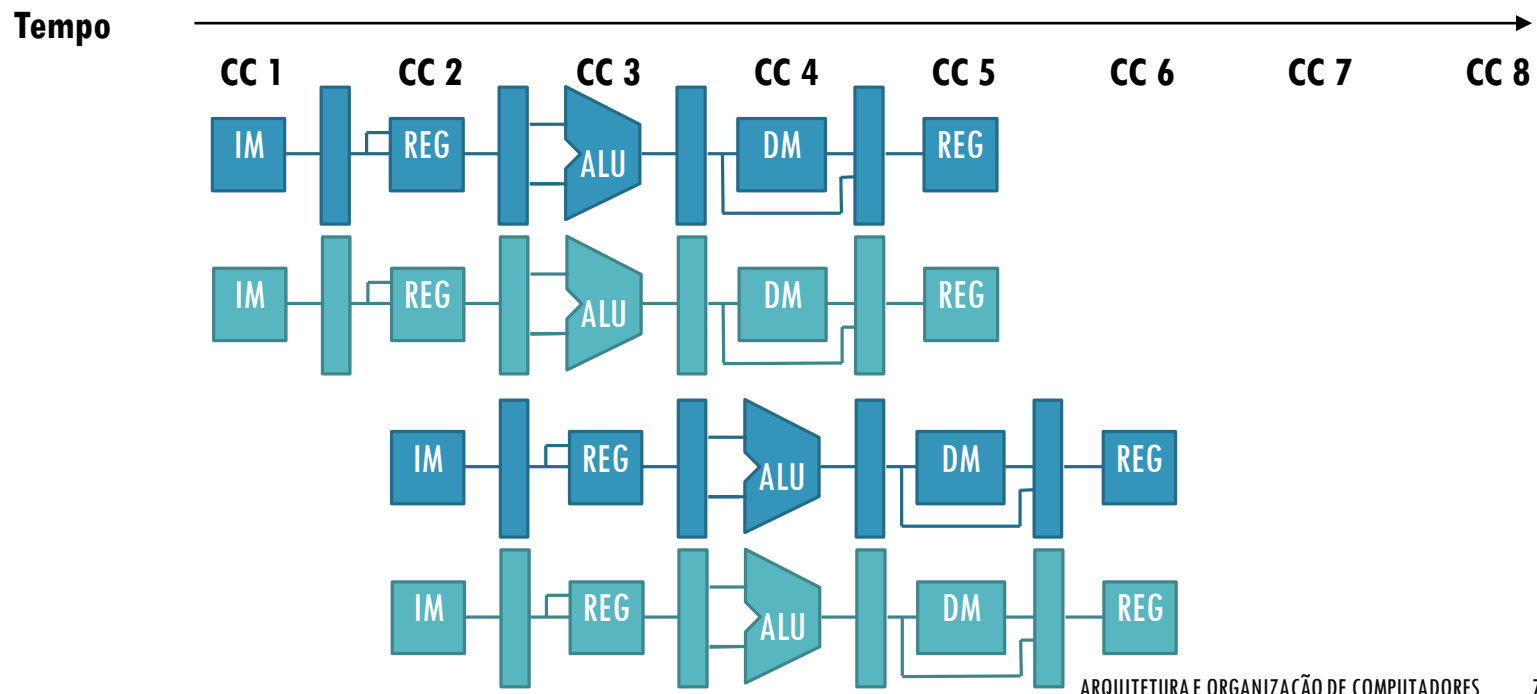


# SINGLE CORE - PIPELINE SUPERSCALAR

Pipelining continua ativo

IPC ideal  $> 1$

ILP Paralelismo de instruções.

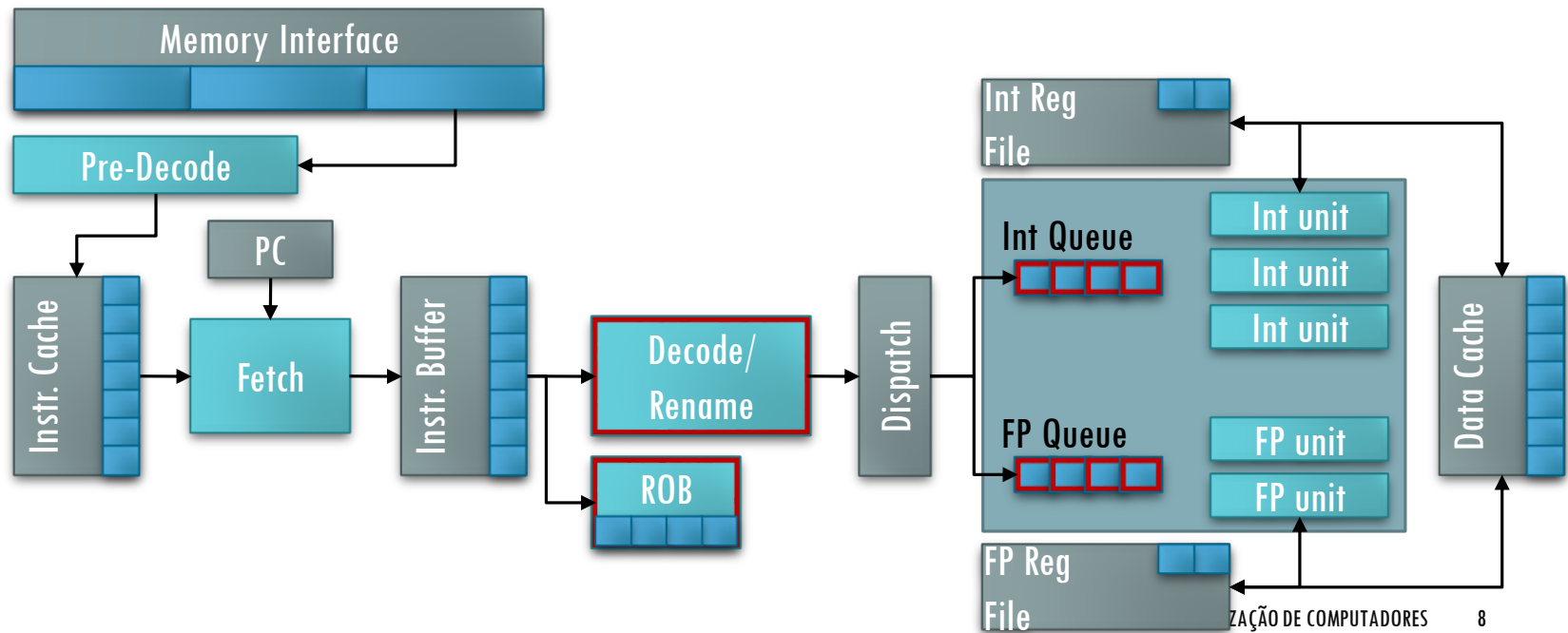


# SINGLE CORE - PIPELINE SUPERSCALAR O<sup>3</sup> (OUT-OF-ORDER)

Início do Paralelismo (**ILP – Instruction Level Parallelism**)

Busca, decodificação e execução de mais de uma instrução por ciclo.

Paralelismo agressivo com despacho e execução fora-de-ordem (OOO).





# PROCESSADORES VLIW

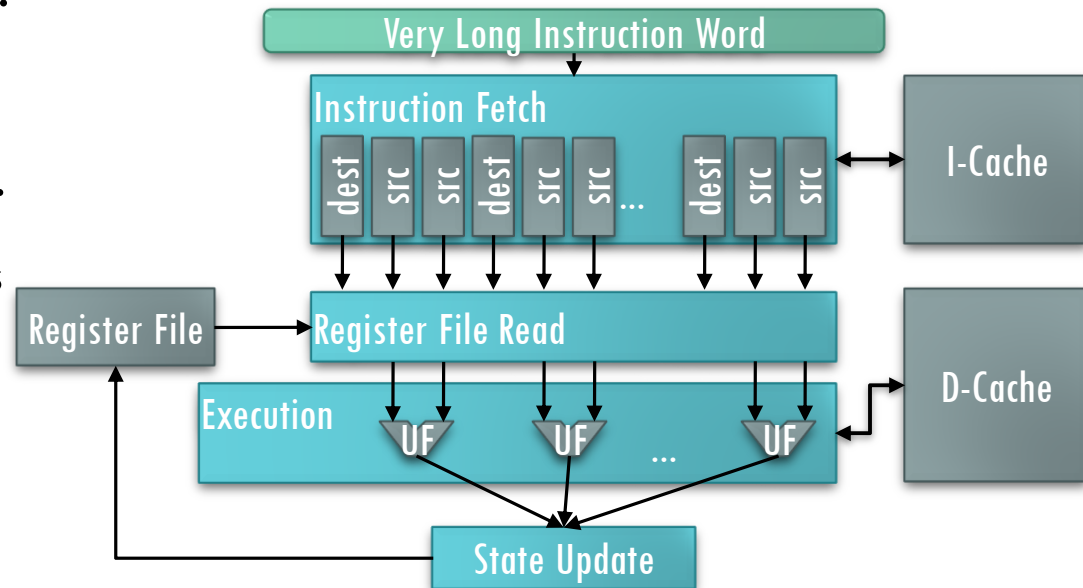
Compilador entende de arquitetura.

Controle e despacho simplificados.

Não mantém retro-compatibilidade.

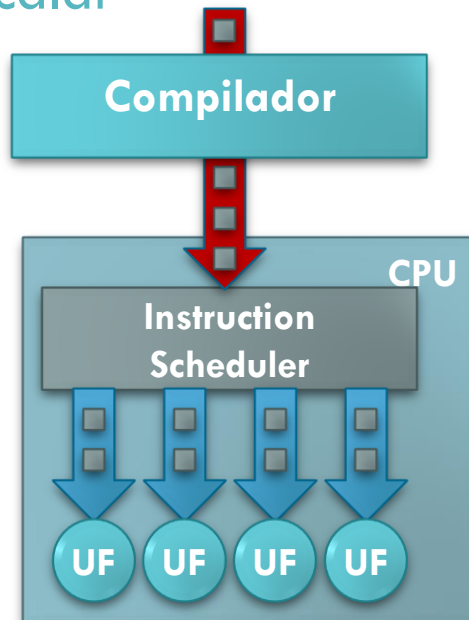
Menor densidade do código (muitos nops).

Requer muitas portas de acesso a dados (cache).

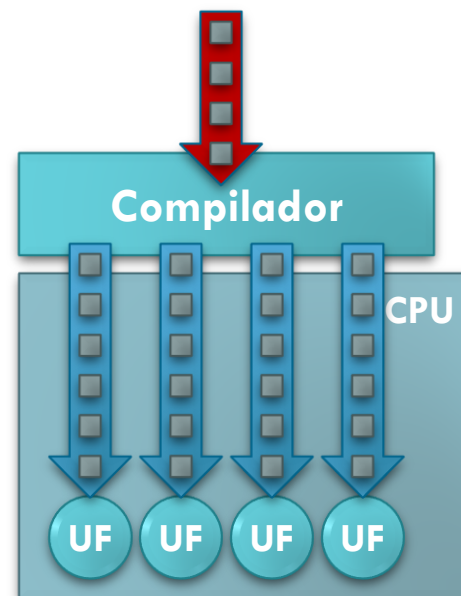


# PROCESSADORES VLIW

Superescalar



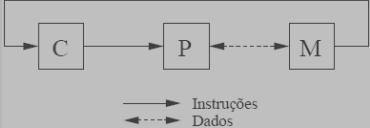
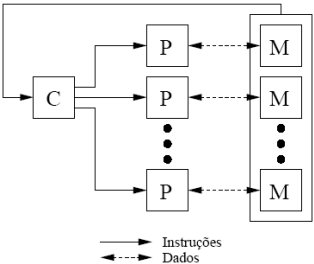
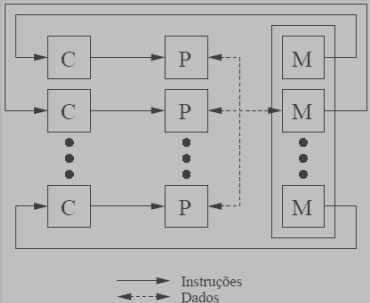
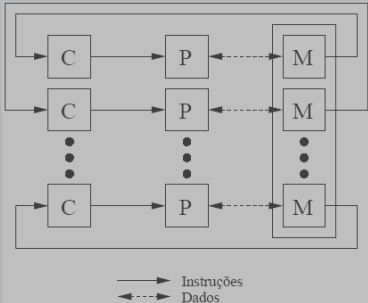
VLIW



# TAXONOMIA DE FLYNN

\*[Flynn, 1972]

\*\*[De Rose, 2003]

*	SD (Single Data)	MD (Multiple Data)
<p><b>SI</b> (Single Instruction)</p>	<p>* *</p>  <p>SISD (Máquinas von Neumann)</p>	<p>* *</p>  <p>SIMD (Máquinas Vetoriais)</p>
<p><b>MI</b> (Multiple Instruction)</p>	<p>* *</p>  <p>MISD (Sem representante / Arquiteturas Sistólicas)</p>	<p>* *</p>  <p>MIMD (Multiprocessadores e Multicomputadores)</p>

# MÁQUINAS SIMD

Processador opera sobre vetores de dados

## Controle único

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

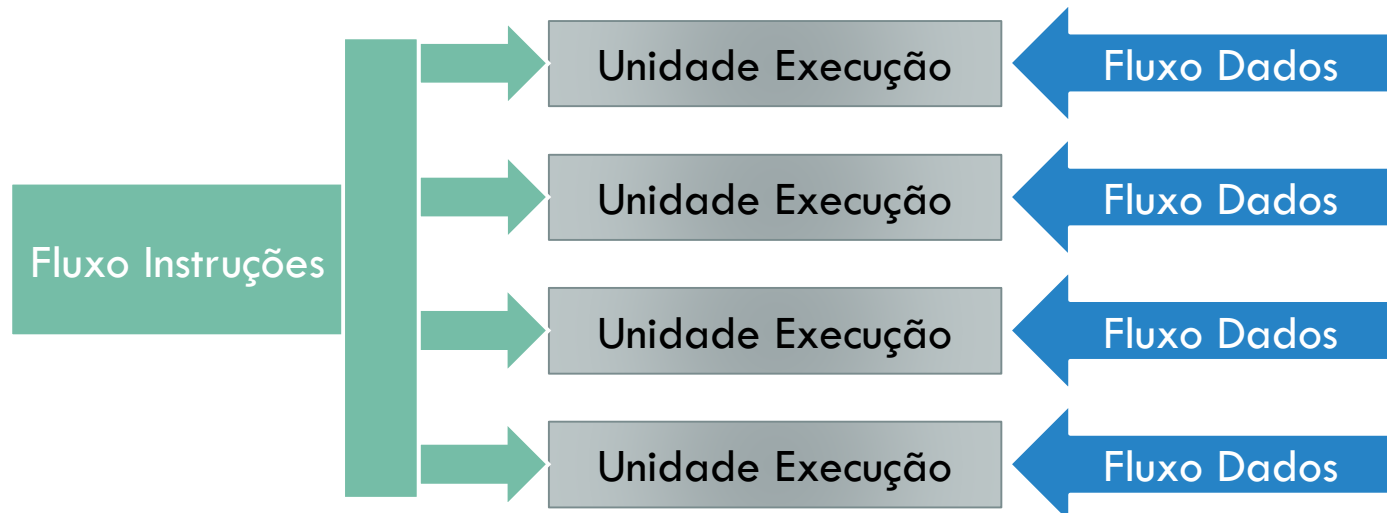
## Múltiplas unidades de execução (blocos operacionais)

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

# MÁQUINAS SIMD (2)

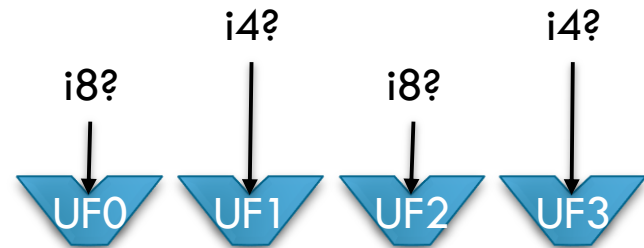

## Processador hospedeiro SISD

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



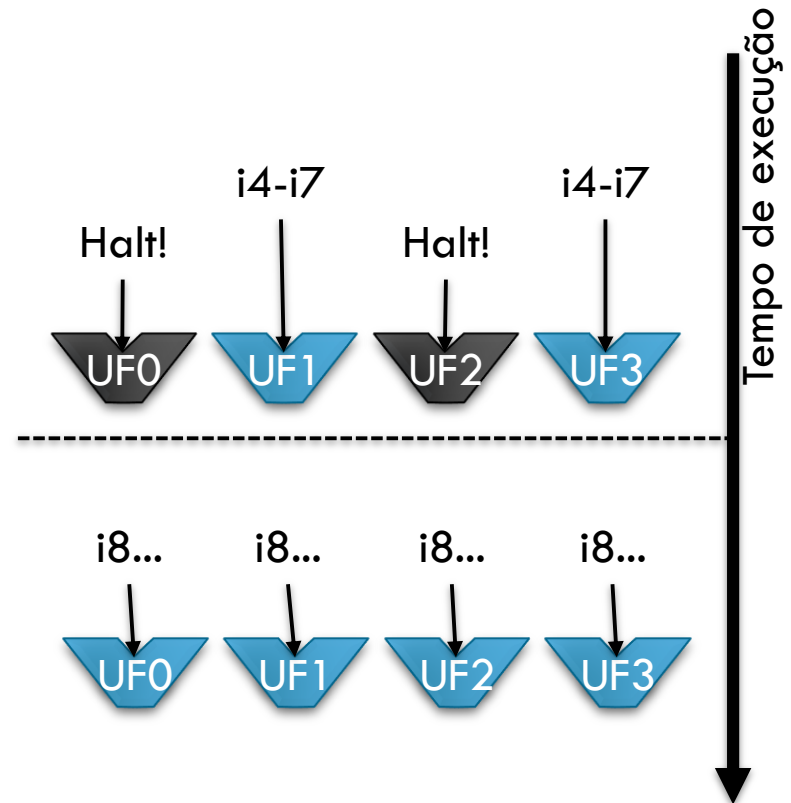

# EFICIÊNCIA DE ARQUITETURAS SIMD

i0  
i1  
i2  
i3 (br) **if (id % 2 == 0) goto i8**  
i4  
i5  
i6  
i7  
i8  
i9  
i10



# EFICIÊNCIA DE ARQUITETURAS SIMD

i0  
i1  
i2  
i3 (br) **if (id % 2 == 0) goto i8**  
i4  
i5  
i6  
i7  
i8  
i9  
i10



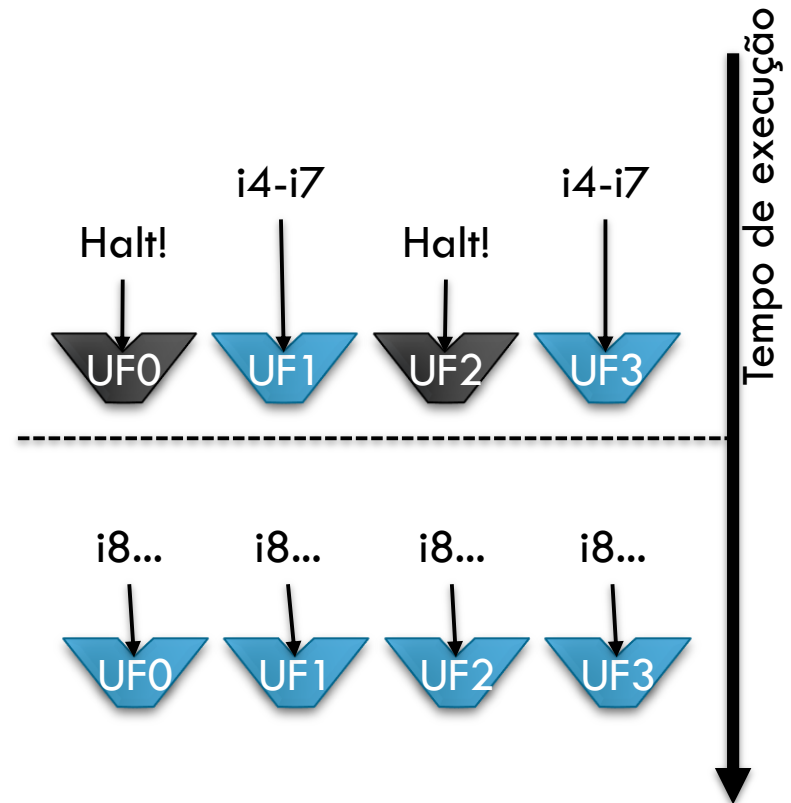
# EFICIÊNCIA DE ARQUITETURAS SIMD

SIMD são eficientes quando processam arrays em laços “for”

- Eficientes quando aplicação tem paralelismo de dados massivo

SIMD são ineficientes em aplicações do tipo “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

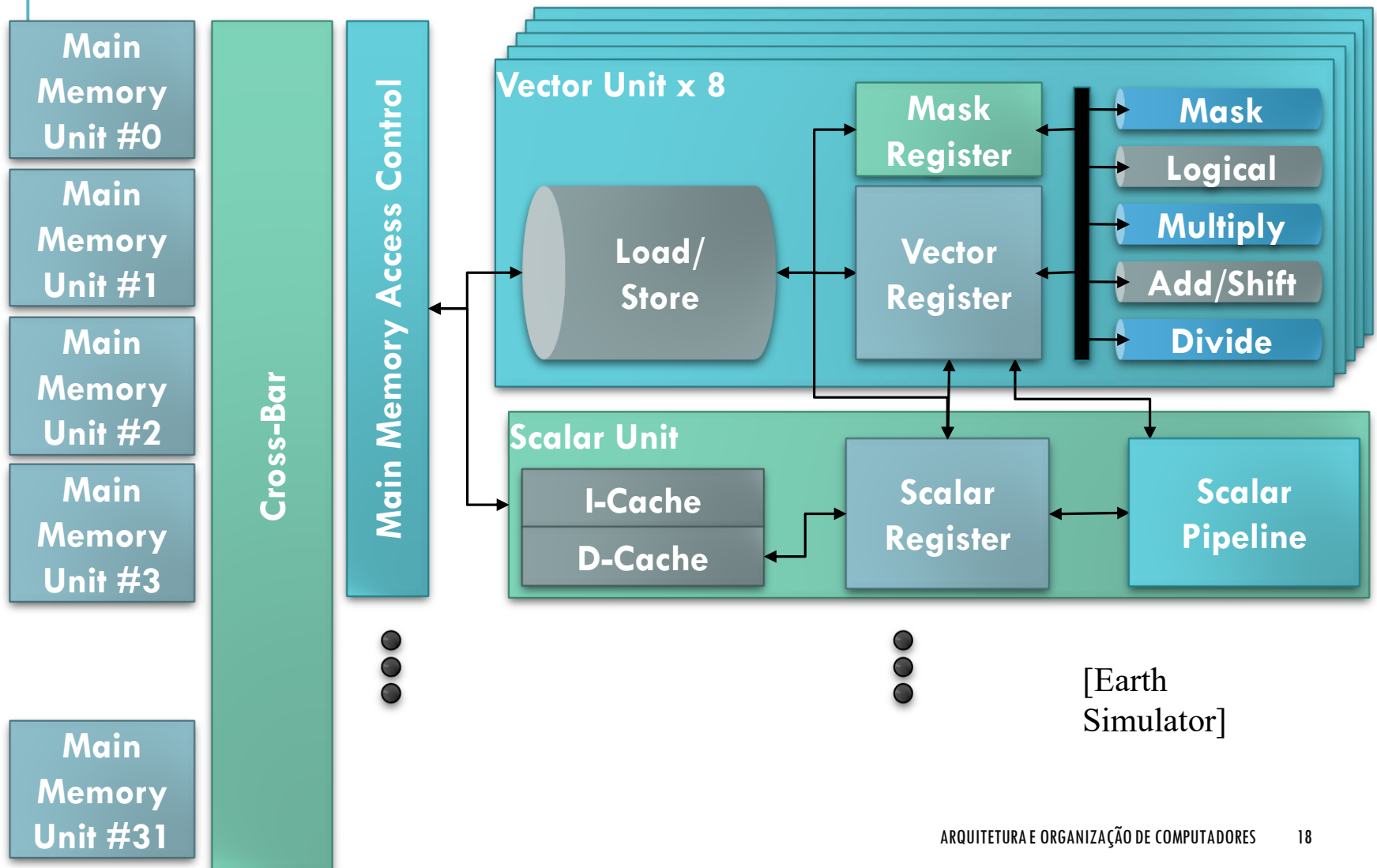
Exemplo: Cray C-90

- Registrador vetorial tem 64 x 64 bits
- 2 pipelines funcionais vetoriais

Outras máquinas vetoriais

- Convex C3, DEC VAX 9000, Fujitsu VP2000, Hitachi S-810, IBM 390/VF

# PROCESSADORES VETORIAIS



# EARTH SIMULATOR (BUILT BY NEC \$500MI) NO. 1 SYSTEM FROM JUNE 2002 TO JUNE 2004

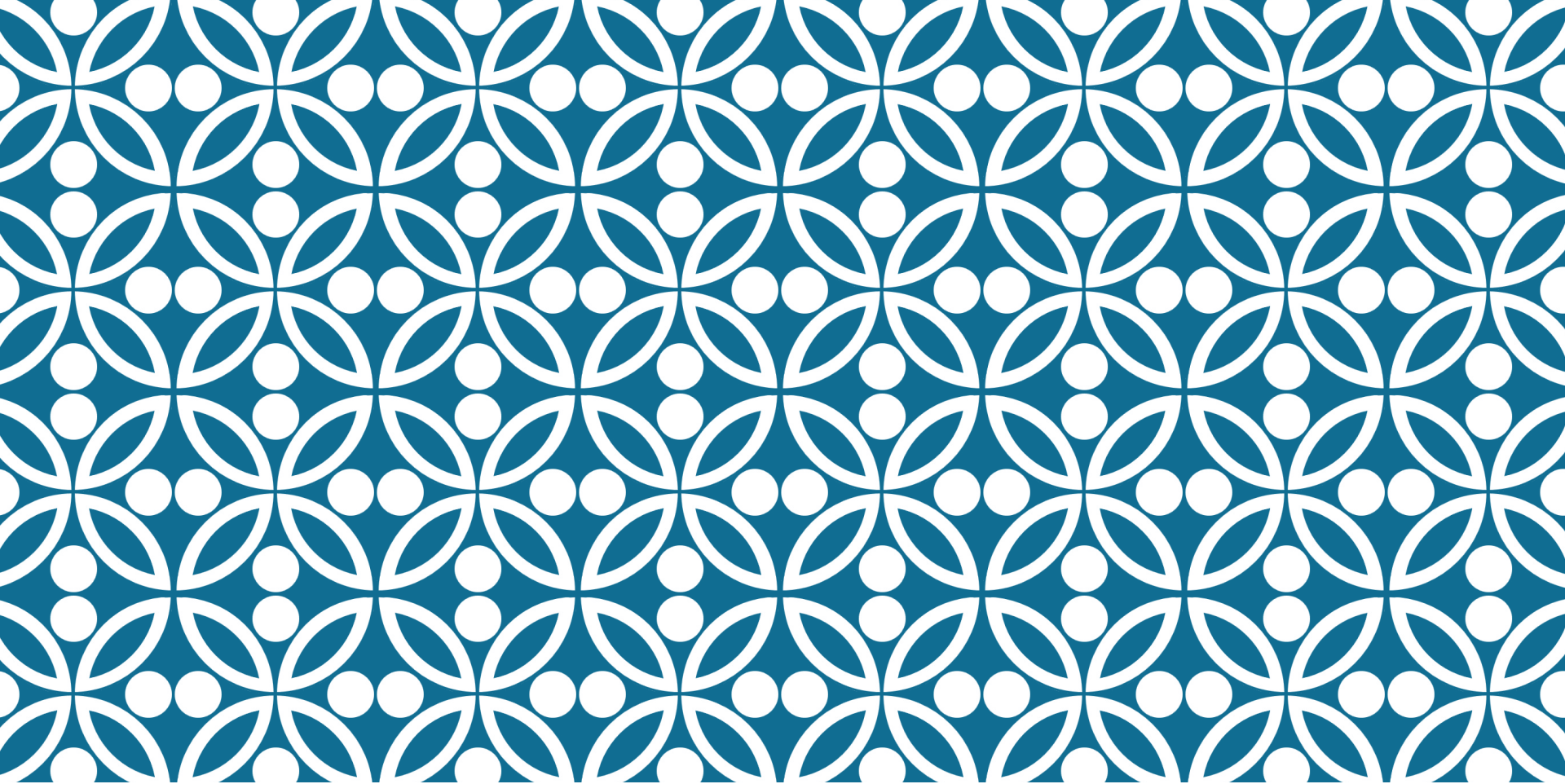




# EARTH SIMULATOR (BUILT BY NEC) NO. 1 SYSTEM FROM JUNE 2002 TO JUNE 2004

**NOT SURE IF EVERYTHING  
IS EXPENSIVE**

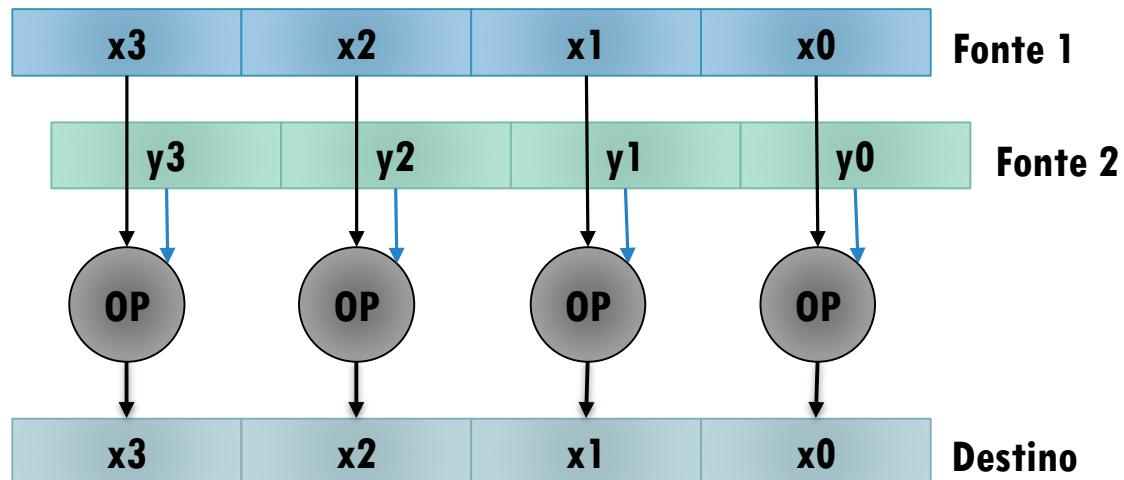
**OR I'M POOR**



# INSTRUÇÕES SIMD

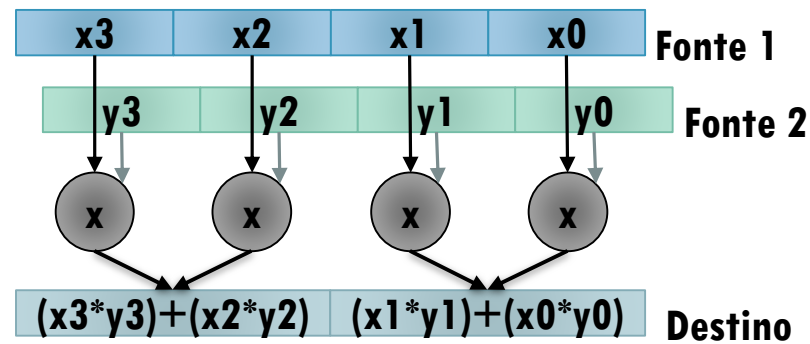
# INSTRUÇÕES SIMD

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

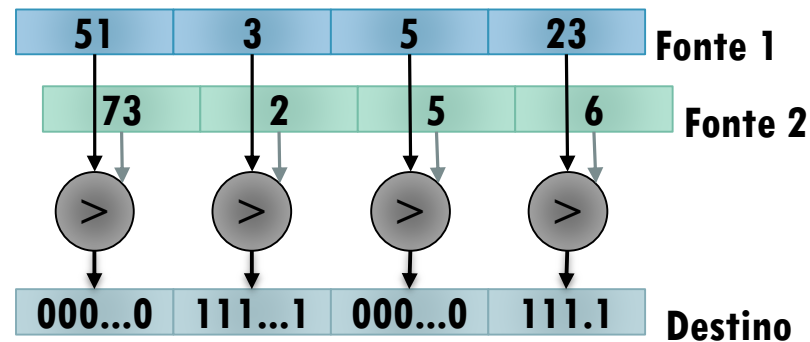


# EXEMPLOS

Mult/Add



Compares



# INTEL MMX – 8X64 BITS REGISTRADORES

(8bits x 8) Packed Bytes



(16bits x 4) Packed Words



(32bits x 2) Packed Doublewords



(64bits x 1) Packed Quadword





# INTEL SSE - 8X128 BITS REGISTRADORES

128-Bit Packed Double-Precision FP



128-Bit Packed Byte Integers



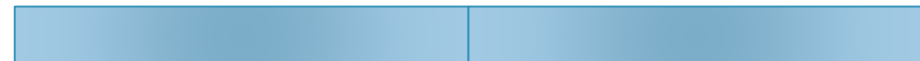
128-Bit Packed Word Integers



128-Bit Packed Doubleword Integers



128-Bit Packed Quadword Integers



# REGISTRADORES X86 (INTEL/AMD)

## Extensão Multimedia e Registradores de Ponto-Flutuante

MM0/ST0
MM1/ST1
MM2/ST2
MM3/ST3
MM4/ST4
MM5/ST5
MM6/ST6
MM7/ST7

## Fluxo SIMD Registradores de Extensão SSE

XMM0
XMM1
XMM2
XMM3
XMM4
XMM5
XMM6
XMM7
XMM8
XMM9
XMM10
XMM11
XMM12
XMM13
XMM14
XMM15

SSE/AVX-128 suportam loads de 128 bits  
AVX-256 suporta load de 256 bits  
AVX-512 carrega até 512 bits (64 bytes!!!)

# USANDO OPERAÇÕES VETORIAIS COM INTRINSICS



## Technologies

- ☐ MMX
- ☒ SSE
- ☐ SSE2
- ☐ SSE3
- ☐ SSSE3
- ☐ SSE4.1
- ☐ SSE4.2
- ☐ AVX
- ☐ AVX2
- ☐ FMA
- ☐ AVX-512
- ☐ KNC
- ☐ SVMML
- ☐ Other

## Categories

- ☐ Application-Targeted
- ☒ Arithmetic
- ☐ Bit Manipulation
- ☐ Cast
- ☐ Compare
- ☐ Convert
- ☐ Cryptography
- ☐ Elementary Math

## Functions

- ☐ General Support
- ☐ Load
- ☐ Logical
- ☐ Mask
- ☐ Miscellaneous

\_mm\_search

`__m128 _mm_add_ps (__m128 a, __m128 b)`

### Synopsis

```
__m128 _mm_add_ps (__m128 a, __m128 b)
#include "xmmintrin.h"
Instruction: addps xmm, xmm
CPUID Flags: SSE
```

### Description

Add packed single-precision (32-bit) floating-point elements in `a` and `b`, and store the results in `dst`.

### Operation

```
FOR j := 0 to 3
  i := j*32
  dst[i+31:i] := a[i+31:i] + b[i+31:i]
ENDFOR
```

### Performance

Architecture	Latency	Throughput (CPI)
Skylake	4	0.5
Broadwell	3	1
Haswell	3	1
Ivy Bridge	3	1

`__m128 _mm_add_ss (__m128 a, __m128 b)`

`__m128 _mm_div_ps (__m128 a, __m128 b)`

`__m128 _mm_div_ss (__m128 a, __m128 b)`

`__m128 _mm_mul_ps (__m128 a, __m128 b)`

`__m128 _mm_mul_ss (__m128 a, __m128 b)`

`__m64 _mm_mulhi_pu16 (__m64 a, __m64 b)`

`__m64 _m_pmulhuw (__m64 a, __m64 b)`

`__m64 _m_psadbw (__m64 a, __m64 b)`

# USANDO OPERAÇÕES VETORIAIS COM INTRINSICS

`__m128 _mm_add_ps (__m128 a, __m128 b)`

## Synopsis

- `__m128 _mm_add_ps (__m128 a, __m128 b)`
- `#include "xmmintrin.h"`
- Instruction: `addps xmm, xmm`
- CPUID Flags: SSE

## Description:

- Add packed single-precision (32-bit) floating-point elements in `a` and `b`, and store the results in `dst`.

## Operation

- FOR `j := 0 to 3`
- `i := j*32`
- `dst[i+31:i] := a[i+31:i] + b[i+31:i]`
- ENDFOR

## Performance

Architecture	Latency	Throughput (CPI)
Skylake	4	0.5
Broadwell	3	1
Haswell	3	1
Ivy Bridge	3	1

# USANDO OPERAÇÕES VETORIAIS COM INTRINSICS

```
#include "xmmintrin.h"

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

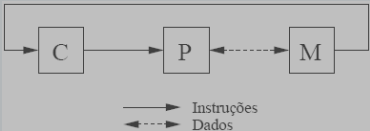
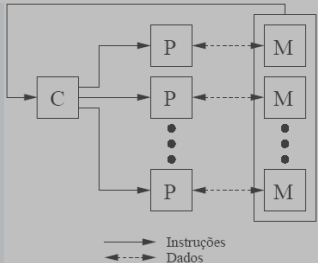
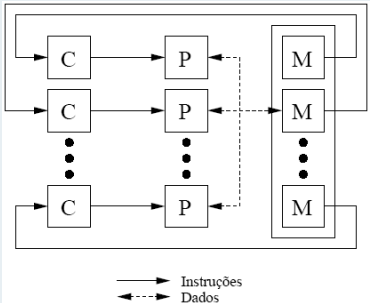
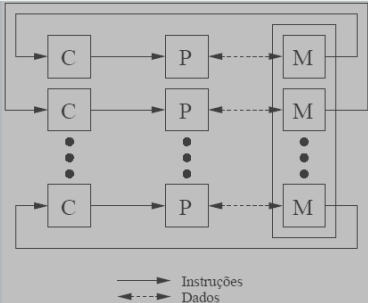
    for (int i=0; i<128; i+=4) {
        a = _mm_load_ps(&va[i]);
        b = _mm_load_ps(&vb[i]);
        c = _mm_add_ps(a, b);
        _mm_store_ps(&vc[i], c);
    }
}
```

Performance		
Architecture	Latency	Throughput (CPI)
Skylake	4	0.5
Broadwell	3	1
Haswell	3	1
Ivy Bridge	3	1

# TAXONOMIA DE FLYNN

\*[Flynn, 1972]

\*\*[De Rose, 2003]

*	SD (Single Data)	MD (Multiple Data)
<p><b>SI</b> (Single Instruction)</p>	<p>* *</p>  <p>SISD (Máquinas von Neumann)</p>	<p>* *</p>  <p>SIMD (Máquinas Vetoriais)</p>
<p><b>MI</b> (Multiple Instruction)</p>	<p>* *</p>  <p>MISD (Sem representante / Arquiteturas Sistólicas)</p>	<p>* *</p>  <p>MIMD (Multiprocessadores e Multicomputadores)</p>

# PROCESSADORES SISTÓLICOS

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

Aplicações em processamento digital de sinais

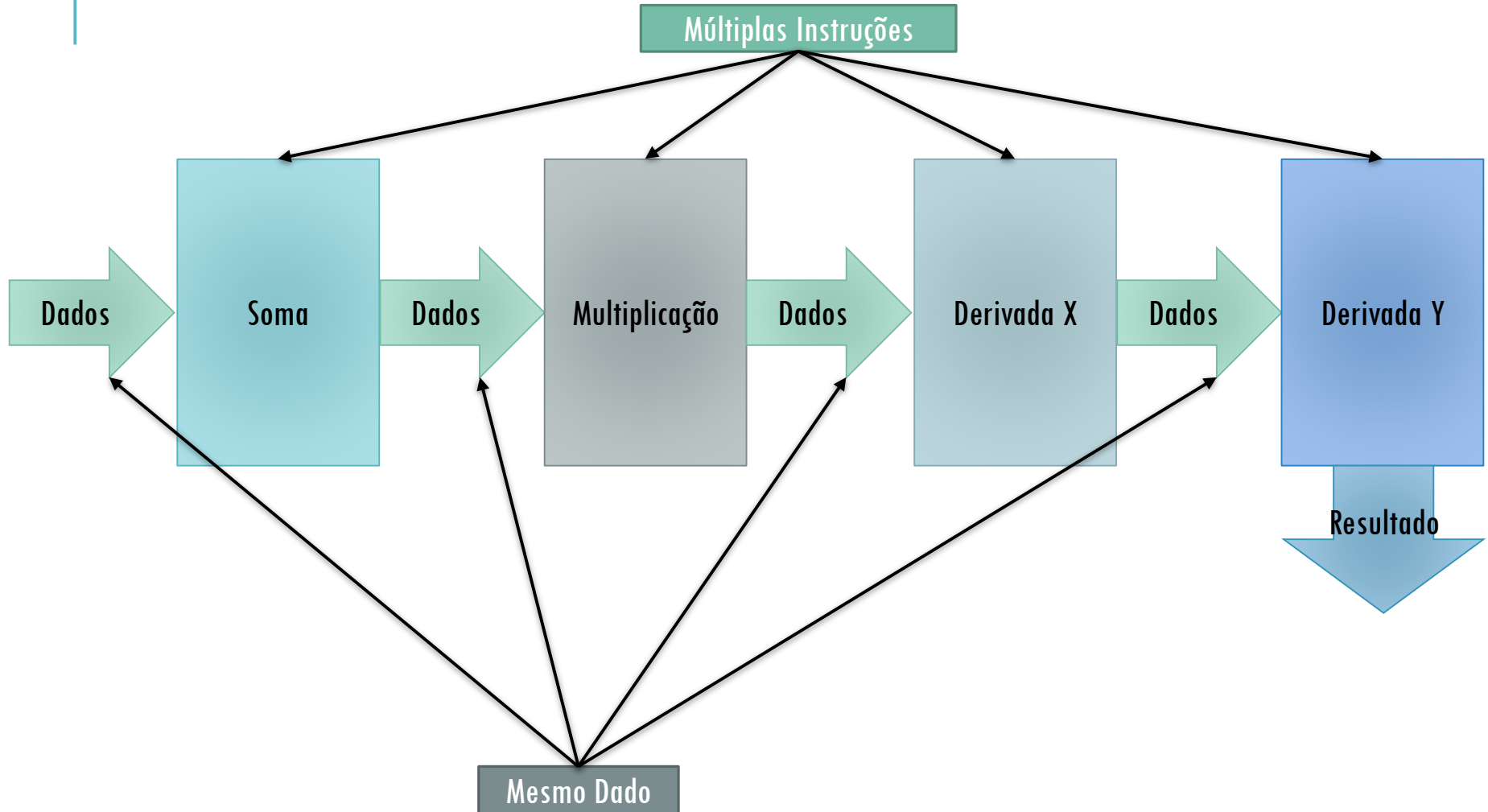
- Processamento de imagens, voz, ...

Integração num único chip

São utilizados em:

- Em sistemas eletrônicos dedicados
- Como unidade funcional especializada de um processador hospedeiro convencional

# PROCESSADORES SISTÓLICOS

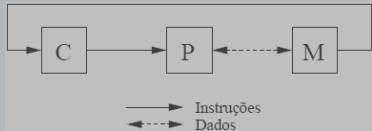
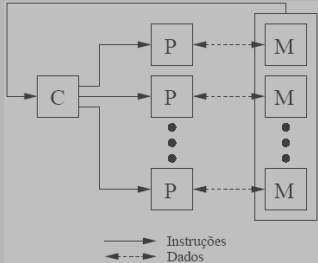
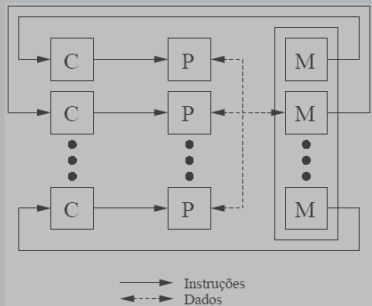
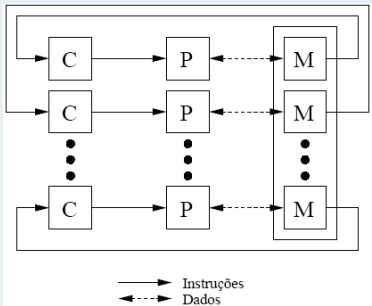




# TAXONOMIA DE FLYNN – MIMD

\*[Flynn, 1972]

\*\*[De Rose, 2003]

	SD (Single Data)	MD (Multiple Data)
SI (Single Instruction)	 <p>SISD (Máquinas von Neumann)</p>	 <p>SIMD (Máquinas Array)</p>
MI (Multiple Instruction)	 <p>MISD (Sem representante até agora / Arquiteturas Sistólicas)</p>	 <p>MIMD (Multiprocessadores e Multicomputadores)</p>

# MULTI-PROCESSORS



[Stallings, 2005]

# 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 ocorrerá condições de corrida.

## Espaço compartilhado

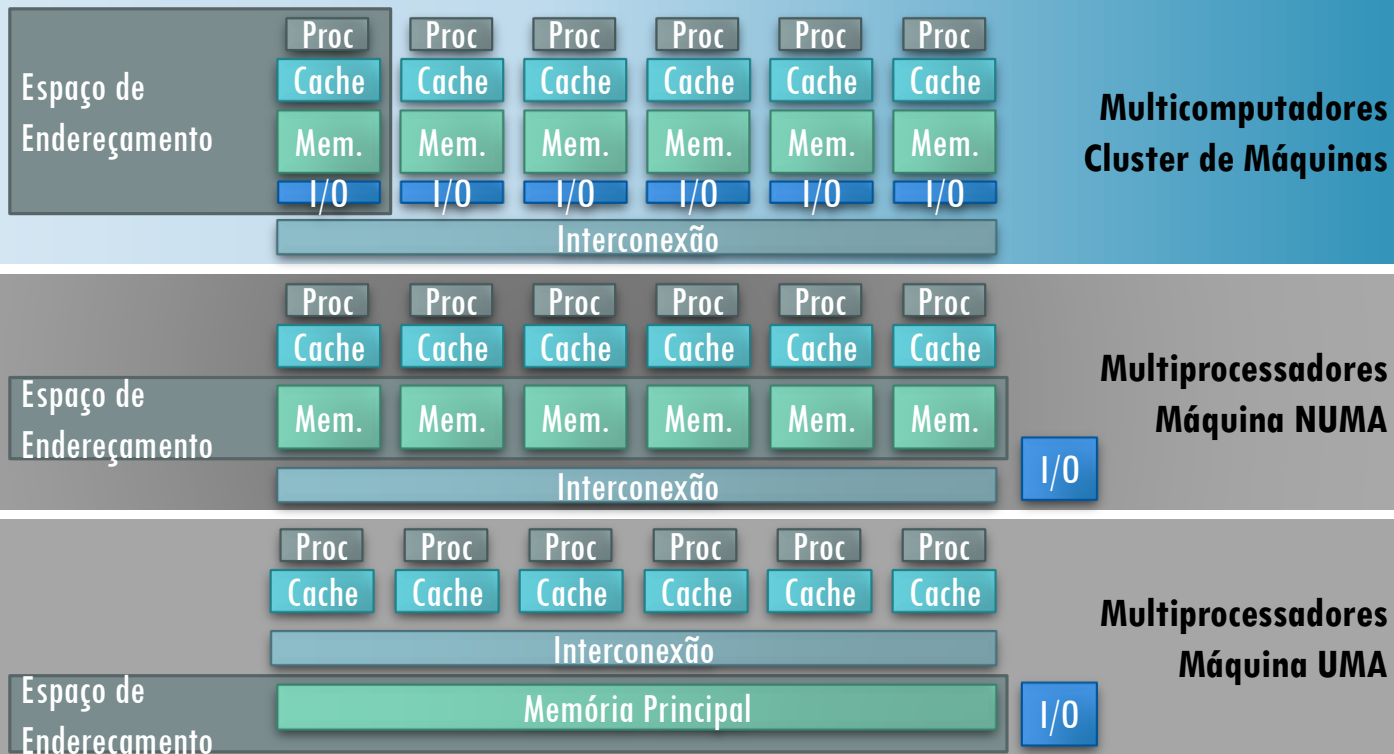
Todos os processadores tem a mesma visão da memória.

Mesma variável pode ser visível para todos.

Comunicação por variável compartilhada (mais rápido).

Haverá naturalmente condições de corrida (recursos compartilhados).

# MULTI-PROCESSORS



[Stallings, 2005]

# CLUSTER DE MÁQUINAS

Conhecidos como COW/NOW: Cluster/Network of Workstations.

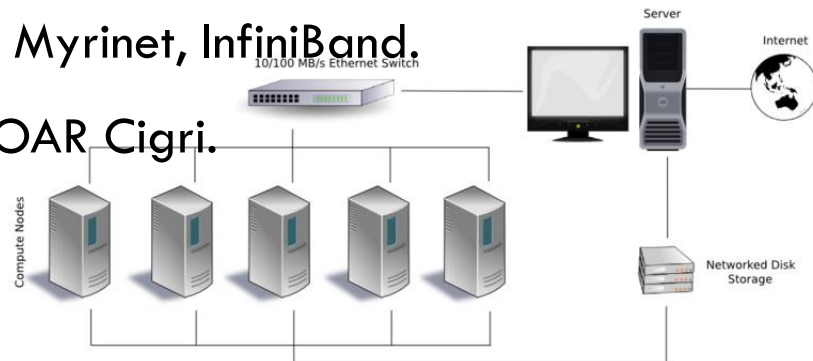
Paralelismo: Bastante claro em nível de processos.

Diversos espaços de endereçamento.

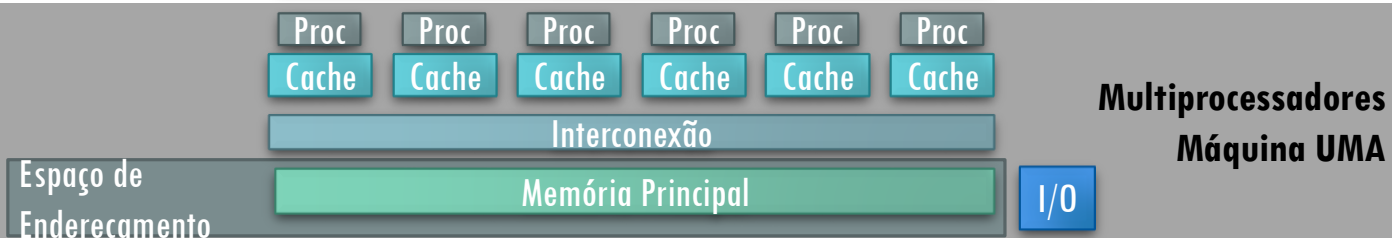
SO para cada máquina: Linux ou proprietário.

Tecnologias de Rede: Ethernet, Myrinet, InfiniBand.

Escalonador de Jobs: Globus, OAR Cigri.



# MULTI-PROCESSORS



[Stallings, 2005]

# MÁQUINAS NUMA

Na arquitetura NUMA as partições dos Cores e as memórias são agrupadas em Nós

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

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**.

# MÁQUINAS NUMA

## MAPEAMENTO DE ENDEREÇOS

Diferentes tipos de mapeamento de endereço para os nodos NUMA podem ser adotados:

Entrelaçado

0	1	2	3
4	5	6	7
8	9	10	11
...	...	...	...

Linear

0	10	20	30
1	11	21	31
2	12	22	32
...	...	...	...

Bloco Entrelaçado

0	5	10	15
~	~	~	~
4	9	14	19
20			
...	...	...	...



# MÁQUINAS NUMA

Exemplo de NUMA com capacidade total de 400 páginas de memória

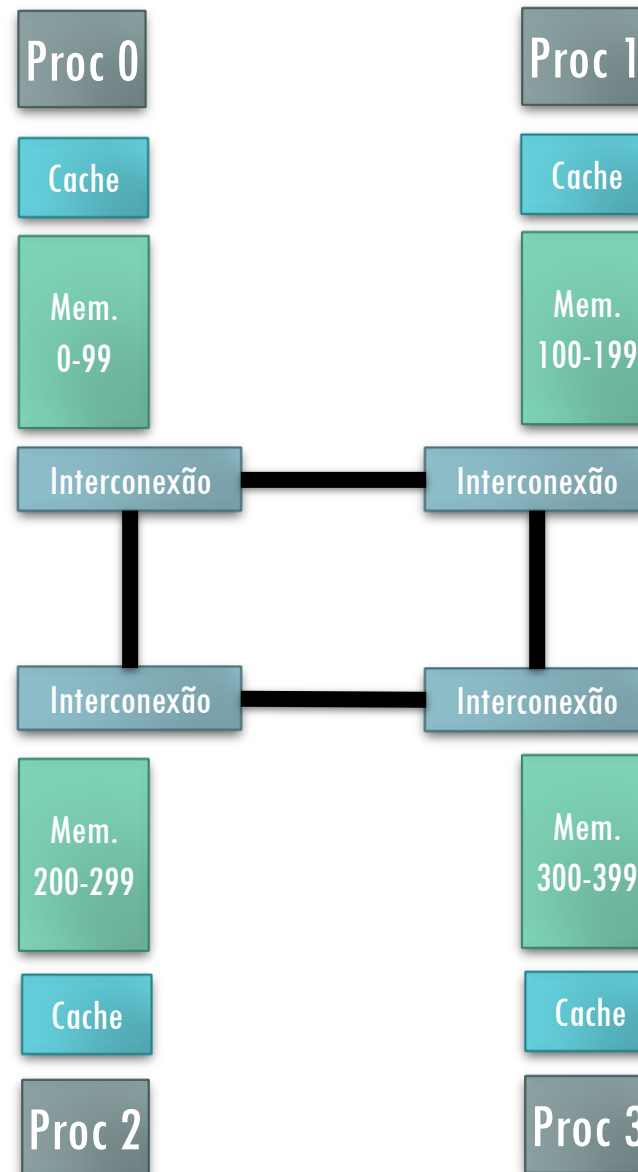
Cada nó contém um quadro de páginas contíguo

Considerando as latências:

- Caches/Memória igual a ***m ciclos***
- Interconexão/link igual a ***n ciclos***

O custo de acesso do Proc 0 acessar:

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



# MÁQUINAS NUMA

## NON-UNIFORM MEMORY ACCESS

### COMA (Cache Only Memory Architecture)

- Formado com memórias cache de alta capacidade.
- Coerência é obtida em hardware com a atualização simultânea em múltiplos nós dos dados.

### CC-NUMA (Cache Coherent NUMA)

- **Coerência de cache é garantida pelo hardware.**

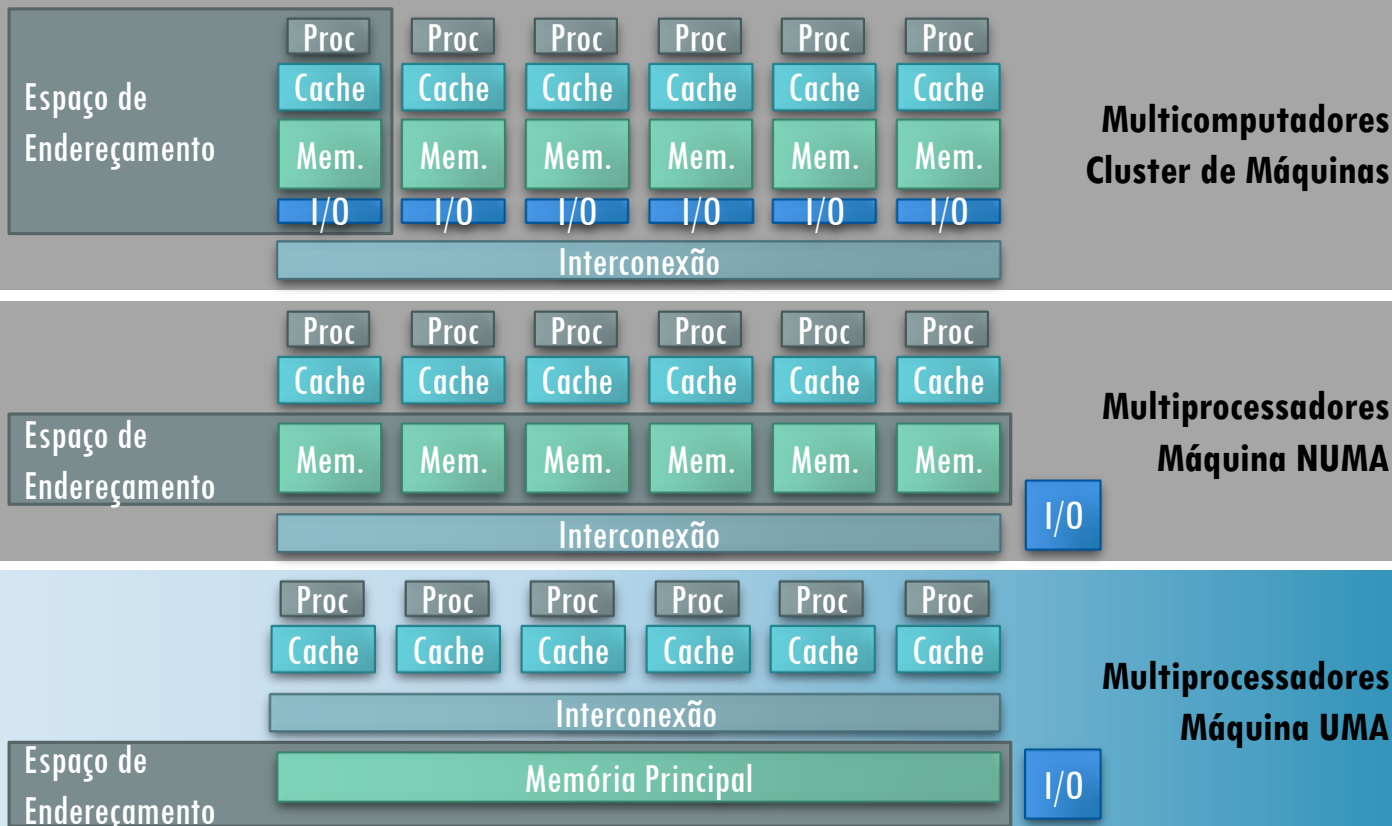
### NCC-NUMA (Non-Cache Coherent NUMA)

- Não existe garantia de coerência da cache ou não existe cache.

### SC-NUMA (Software Coherent NUMA) /DSM.

- Coerência de cache é garantida em software.
- NORMA ou NCC-NUMA com coerência por software.

# MULTI-PROCESSORS



[Stallings, 2005]