



Universidade Federal  
de Campina Grande



# SISTEMAS DIGITAIS MULTIPROCESSADORES

Prof. Thommas Kevin Sales Flores

# SUMÁRIO

- ✓ Motivação
- ✓ Introdução
- ✓ Taxonomia
- ✓ Arquitetura (UMA, NUMA, COMA)
- ✓ Coerência de Memória Cache

# MOTIVAÇÃO

- Processamento de Dados;
- Simulações Científicas e Modelagem;
- Renderização Gráfica e Animação;
- Processamento de Vídeo e Imagem.

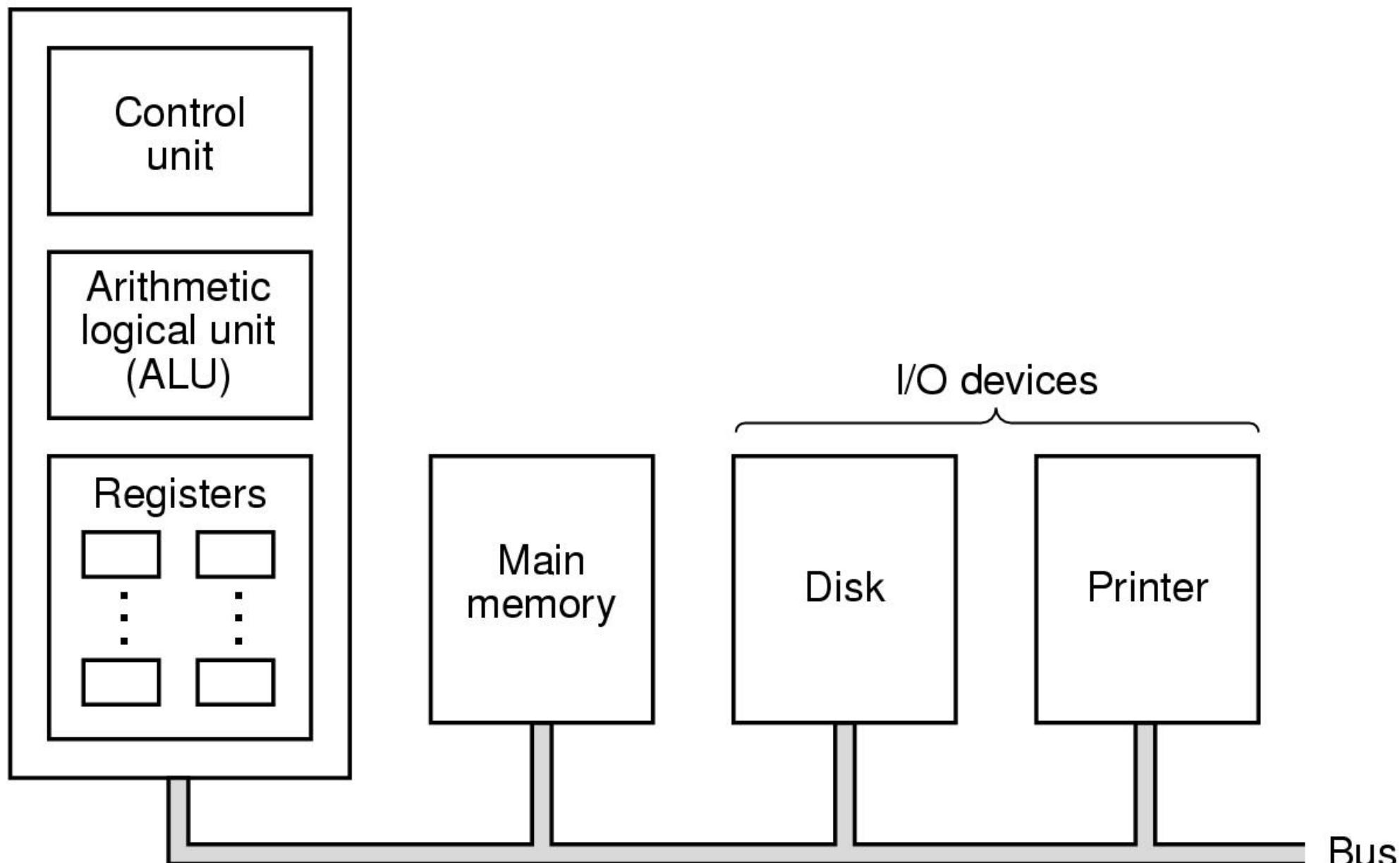
# BIG DATA



# INTRODUÇÃO

- **Processador ou CPU**

Central processing unit (CPU)

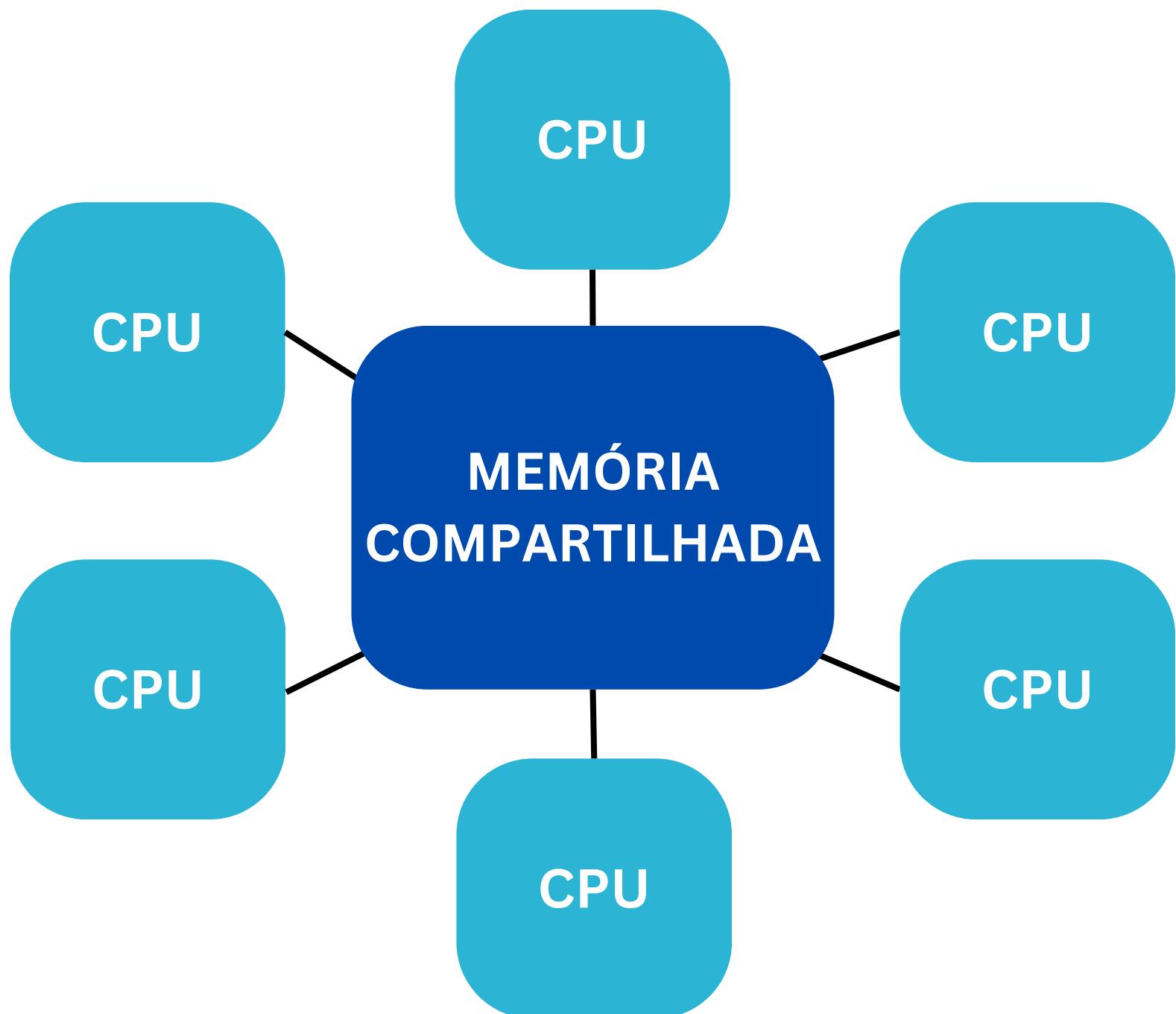


*“É o encapsulamento da Unidade de Controle, Unidade Lógica e Aritmética e dos Registradores”*

**Andrew S. Tanenbaum**

# INTRODUÇÃO

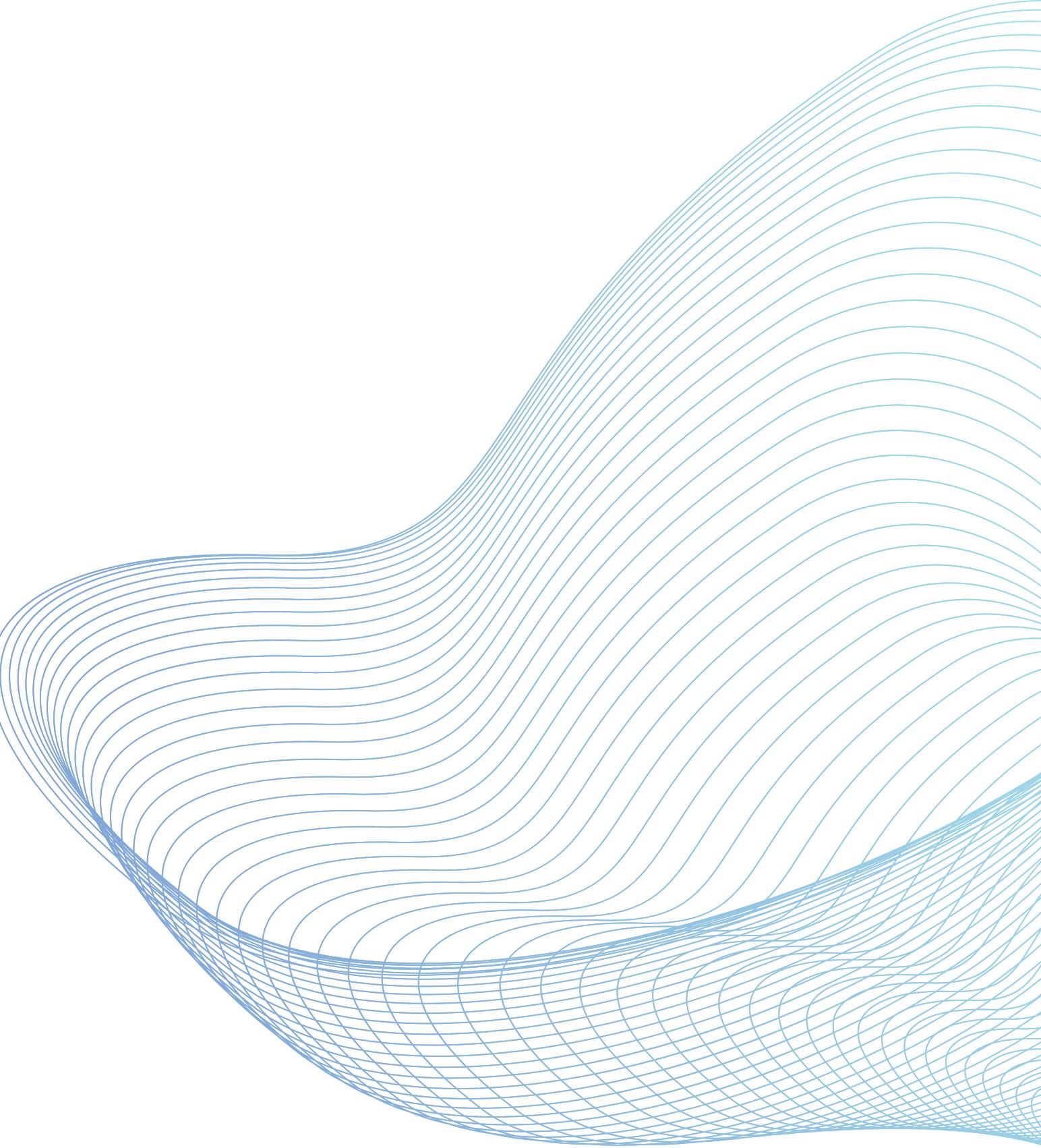
- Definição de multiprocessador



*“É um computador paralelo, no qual as CPU's compartilham uma memória comum”*

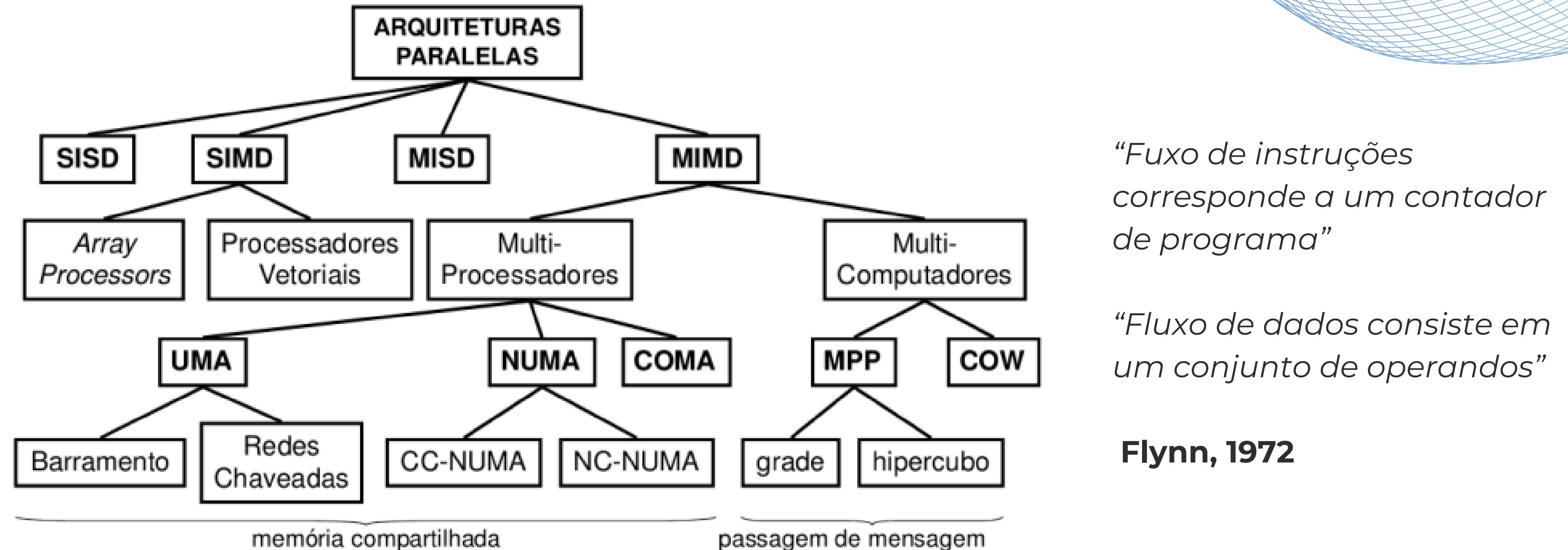
**Andrew S. Tanenbaum**

# MULTIPROCESSADORES TAXONOMIA

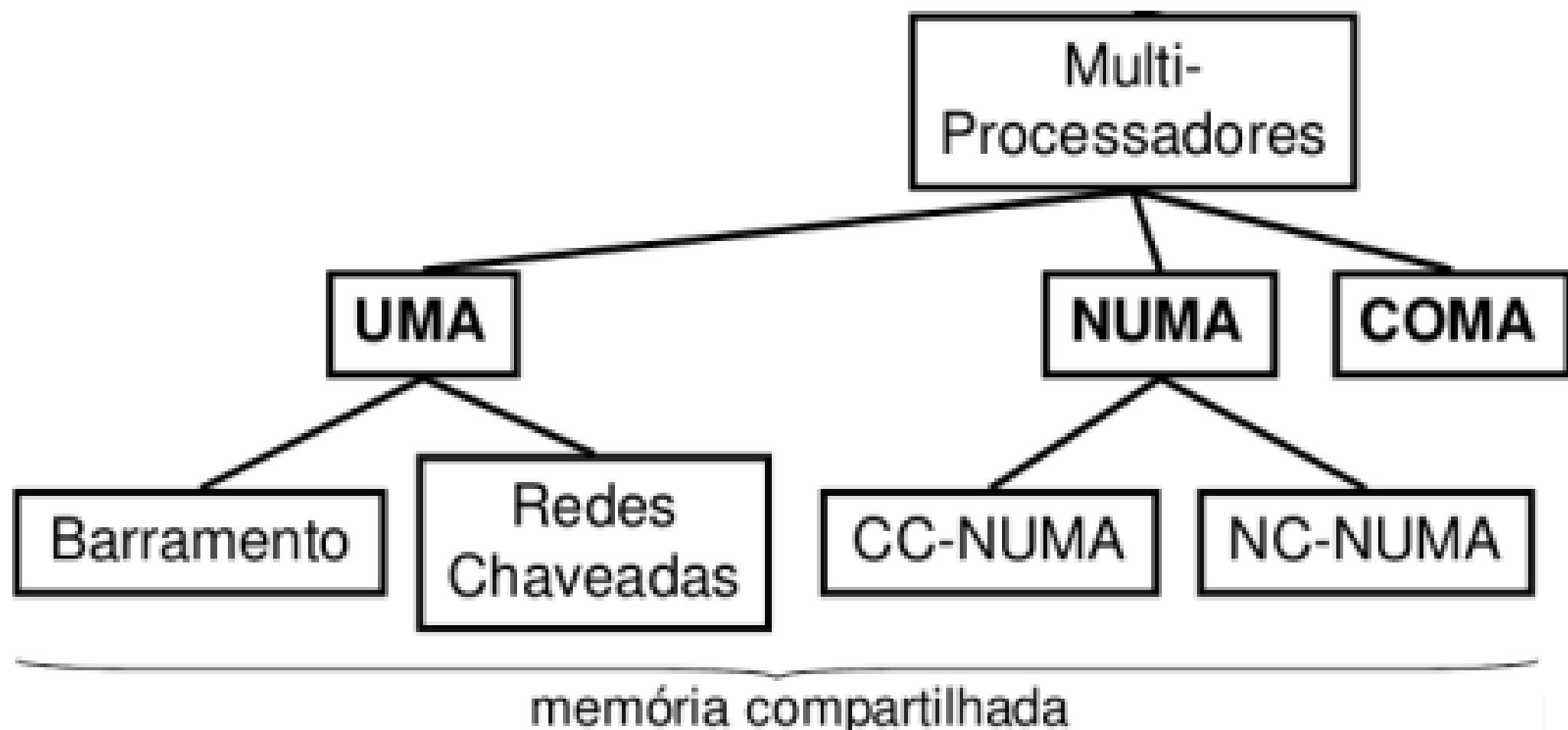


# TAXONOMIA DE COMPUTADORES PARALELOS (FLYNN)

- Flynn é baseada em dois conceitos: fluxos de instruções e fluxos de dados



# TAXONOMIA DE COMPUTADORES PARALELOS (FLYNN)



“UMA (Uniform Memory Access)”

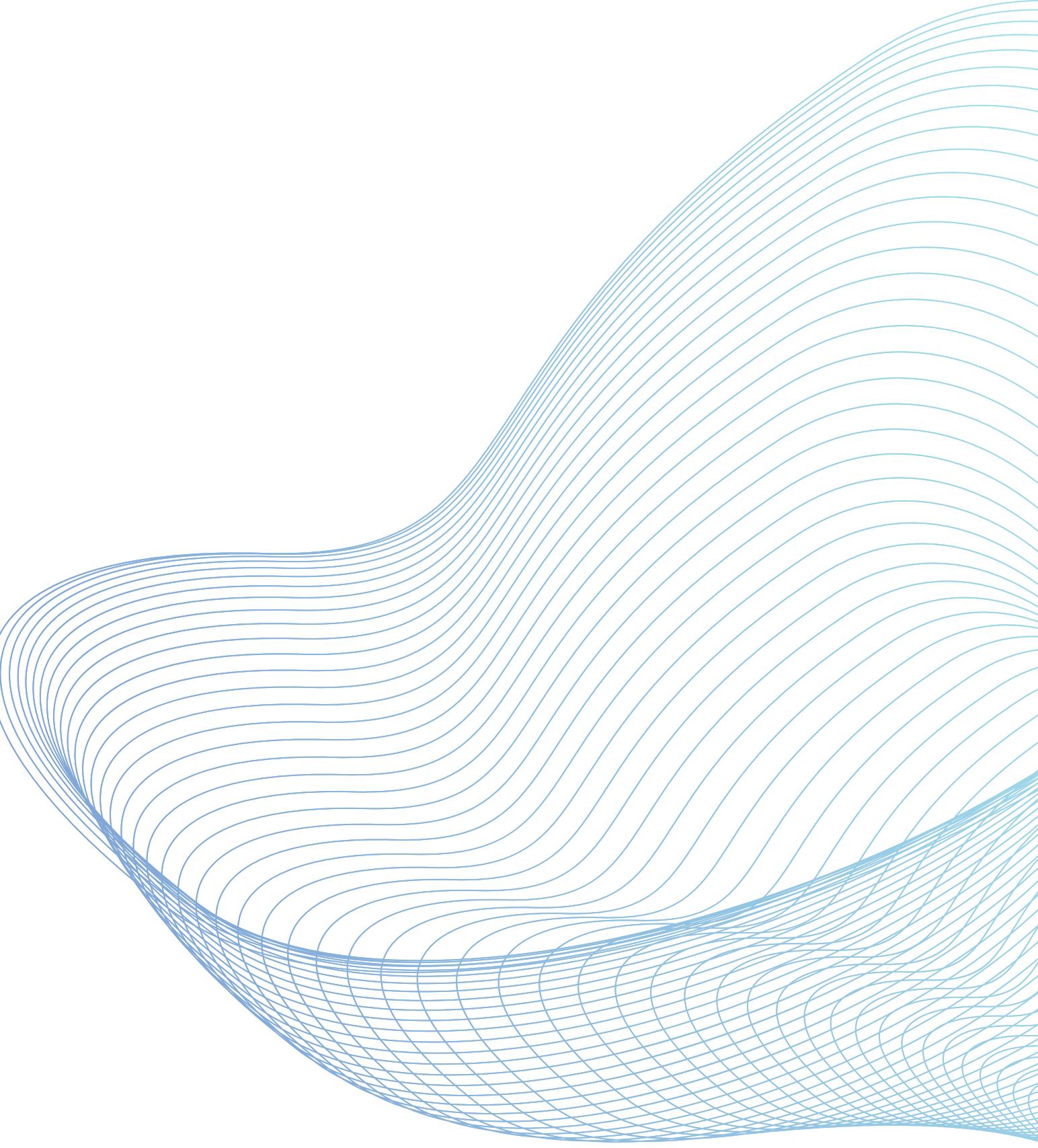
“NUMA (NonUniform Memory Access)”

“COMA (Cache Only Memory Access)”

(Flynn, 1972)

# MULTIPROCESSADORES

## UMA - UNIFORM MEMORY ACCESS

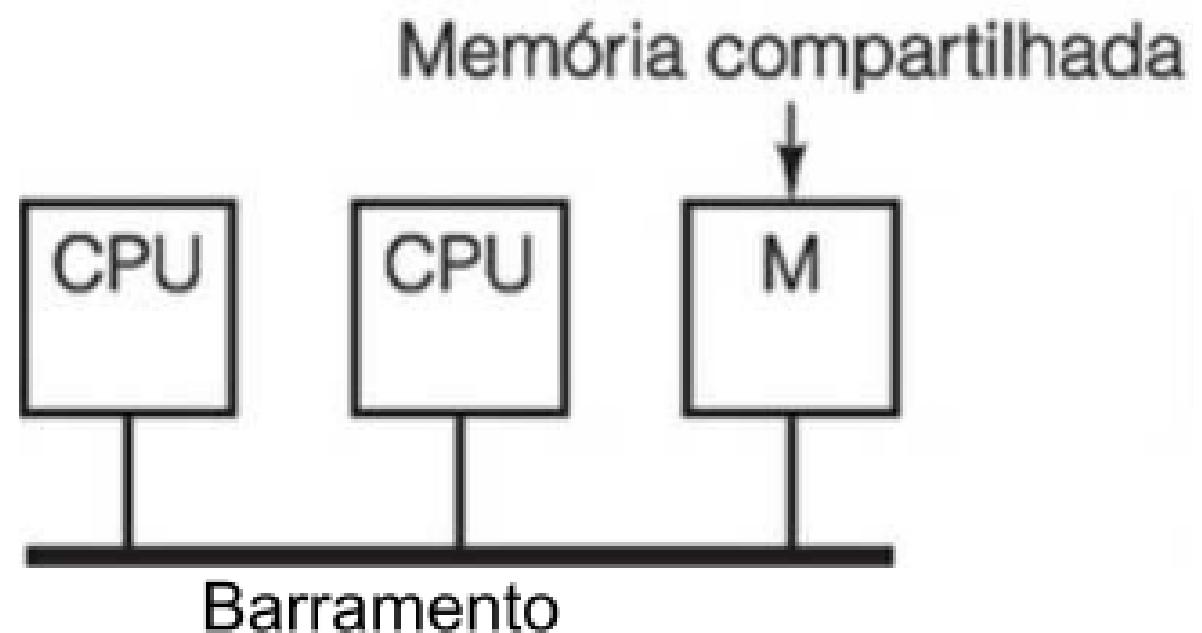


# UMA - UNIFORM MEMORY ACCESS

## BARRAMENTO

### Arquiteturas de multiprocessadores simétricos UMA

- São baseados em um único barramento;
- Quando uma CPU quer ler uma palavra de memória, ela primeiro verifica se o barramento está ocupado;
- Para um grande número de CPUs, o processamento é limitado pela largura de banda do barramento.

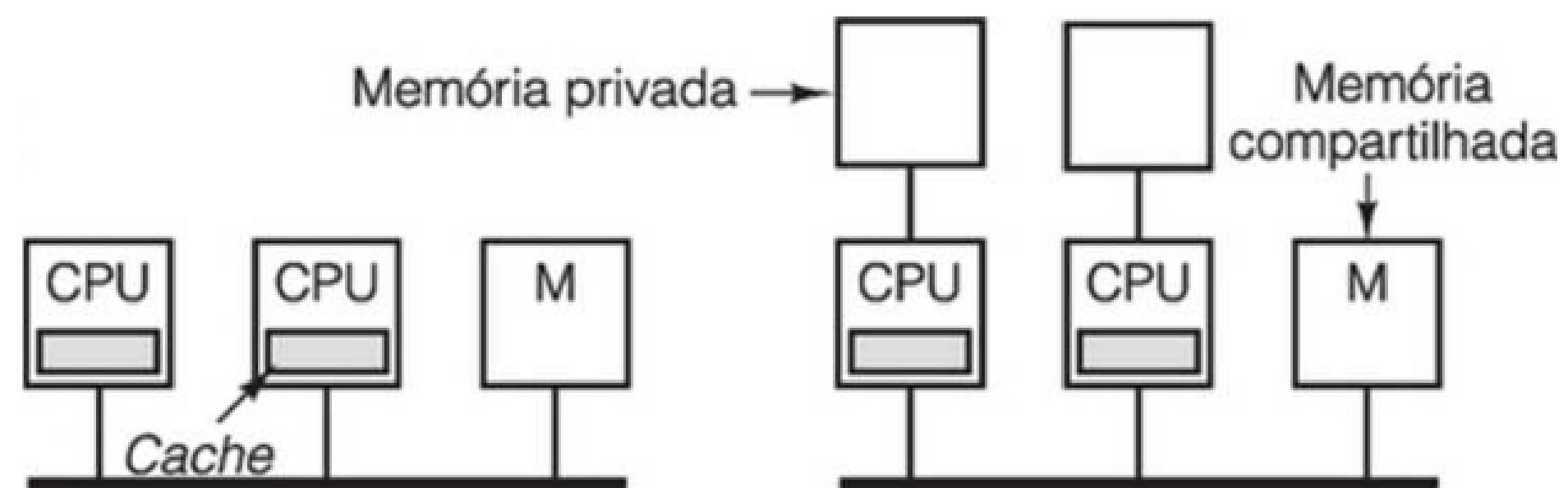


# UMA - UNIFORM MEMORY ACCESS

## BARRAMENTO

### Solução para a largura de banda do barramento

- A solução para esse problema é acrescentar uma cache a cada CPU;
- A cache pode estar dentro do chip da CPU, próxima ao chip da CPU, na placa do processador ou alguma combinação de todas as três;
- Não só uma cache, mas também uma memória local e privada que ela acessa por um barramento dedicado.



# UMA - UNIFORM MEMORY ACCESS

## REDES CHAVEADAS

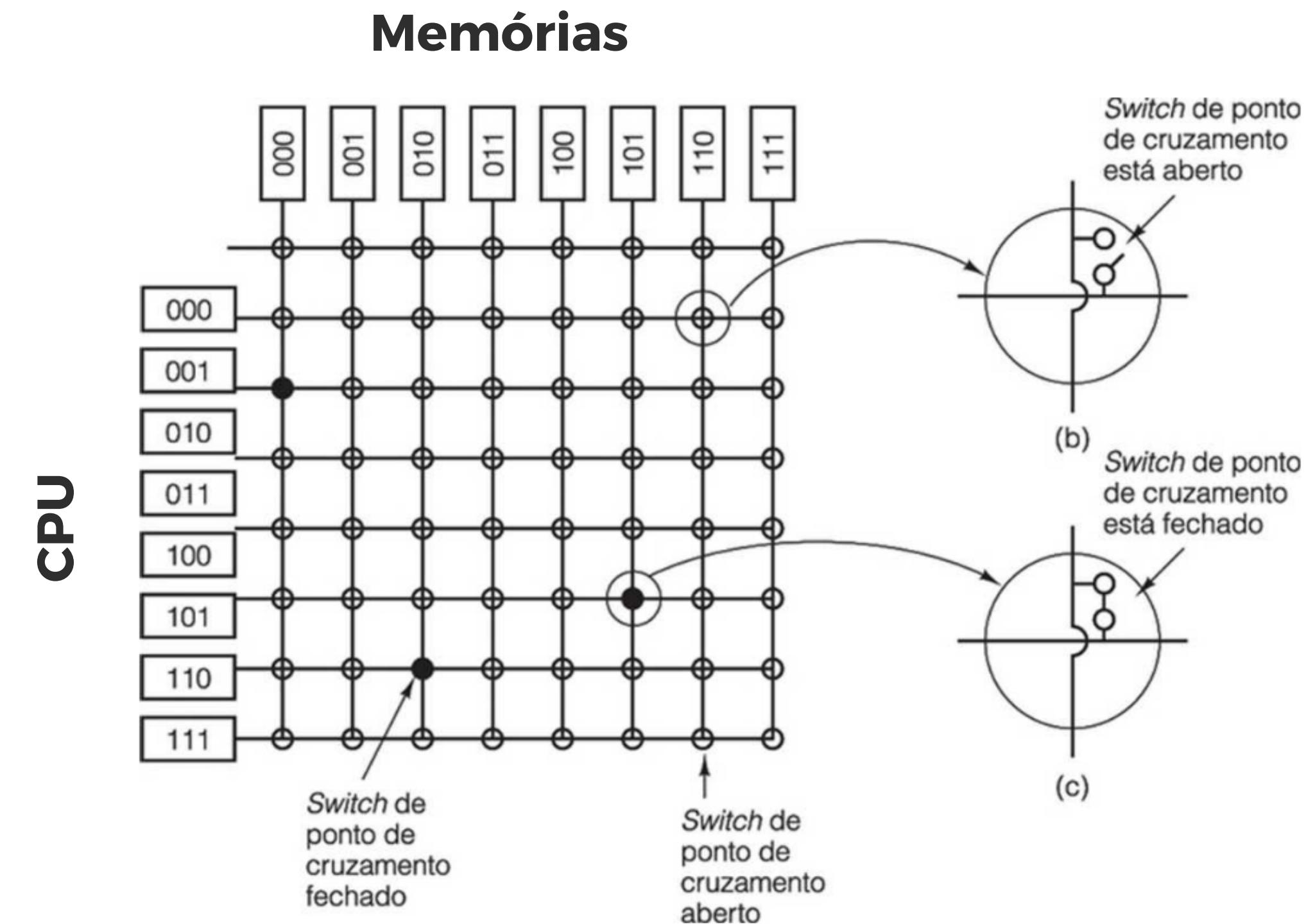
### Definições:

- A utilização de um **único barramento** limita o tamanho do multiprocessador UMA a cerca de 16 ou 32 CPUs;
- Uma solução é utilizar diferentes tipos de rede de interconexão;
- O circuito mais simples para conectar  $n$  CPUs a  $k$  memórias é o switch crossbar;
- Cada intersecção de uma linha horizontal (de entrada) com uma linha vertical (de saída) está um ponto decruzamento (pequeno switch que pode ser aberto ou fechado eletricamente).

# UMA - UNIFORM MEMORY ACCESS

## REDES CHAVEADAS

### Estrutura



# **UMA - UNIFORM MEMORY ACCESS**

## **REDES CHAVEADAS**



### **Vantagens**

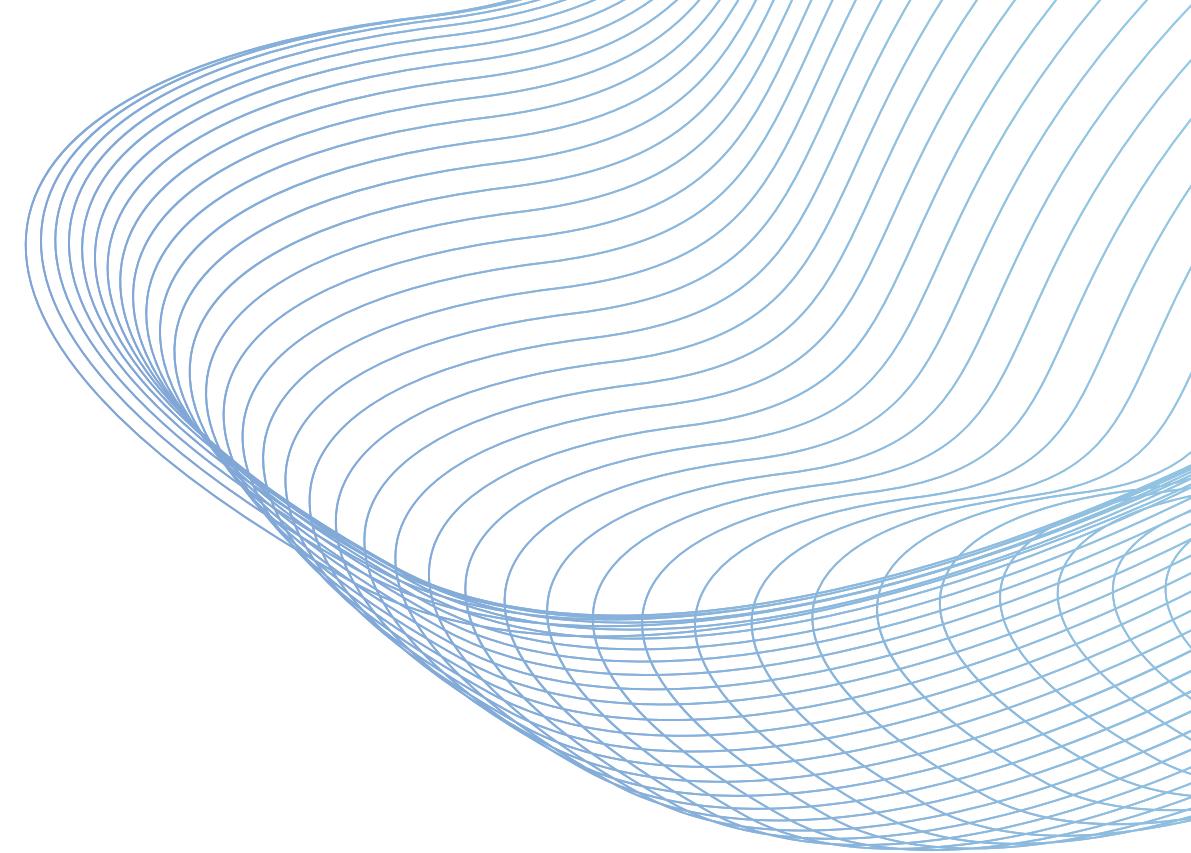
- Rede sem bloqueio, o que significa que a nenhuma CPU é negada a conexão de que necessita porque algum ponto de cruzamento ou linha já está ocupado;
- Não é preciso planejamento antecipado: sempre é possível conectar a CPU restante à memória restante;

### **Desvantagem**

- Número de pontos de cruzamento cresce com  $n$  ao quadrado;
- O projeto de switches crossbar é viável para sistemas de tamanho médio.;

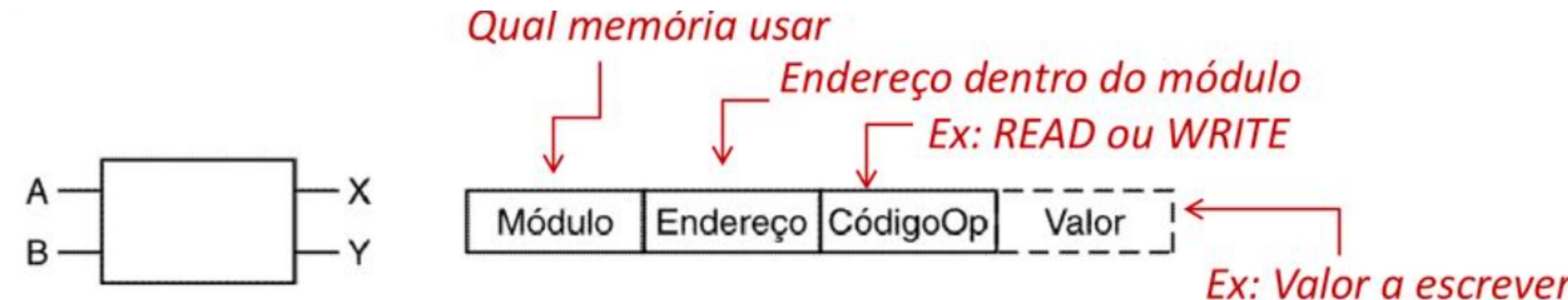
# UMA - UNIFORM MEMORY ACCESS

## REDES DE COMUTAÇÃO MULTIFESTÁGIOS



### Definição:

- Baseado no modesto switch 2x2
- Mensagens que chegam a qualquer uma das linhas de entrada podem ser comutadas para qualquer das linhas de saída.

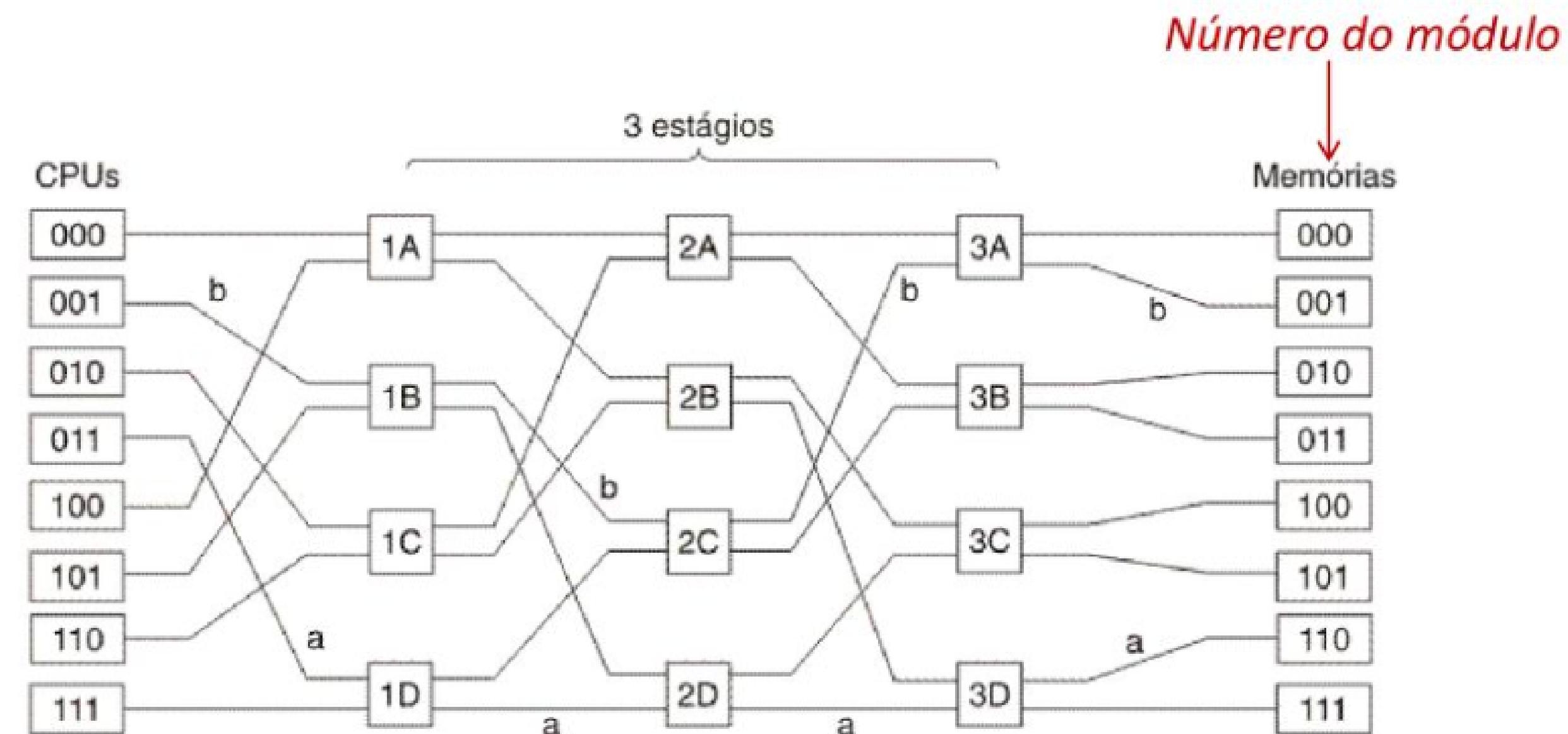


# UMA - UNIFORM MEMORY ACCESS

## REDES DE COMUTAÇÃO MULTIFESTÁGIOS

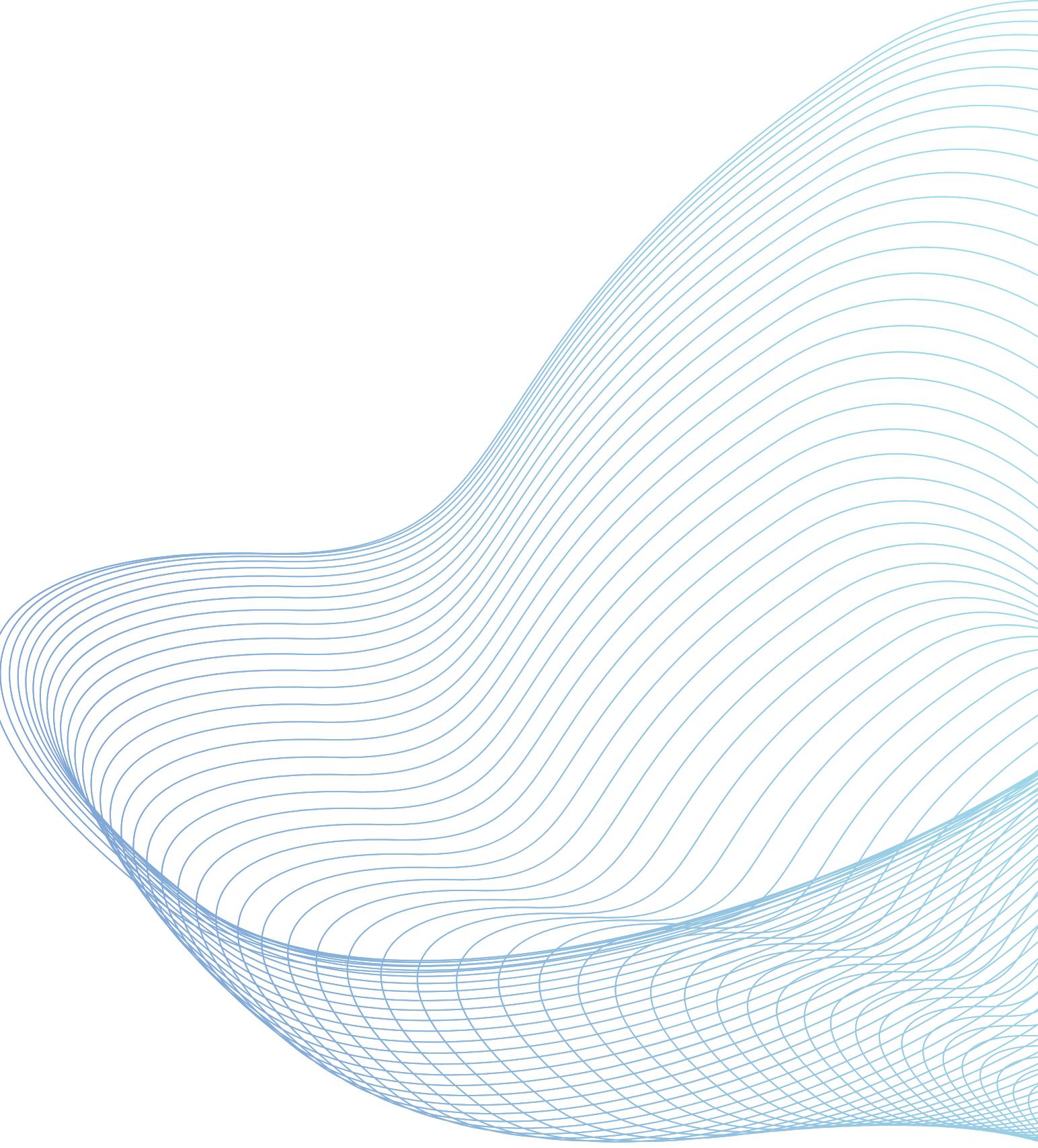
### Rede de comutação ômega

- Em Cada estágio, bit 1 roteia para baixo, bit 0 roteia para cima. Por exemplo, CPU 011 enderaça módulo 110 (**Caminho a**);
- A rede ômega é uma rede com bloqueio



# MULTIPROCESSADORES

## NUMA - NON UNIFORM MEMORY ACCESS



# NUMA- NON UNIFORM MEMORY ACCESS

## Definição:

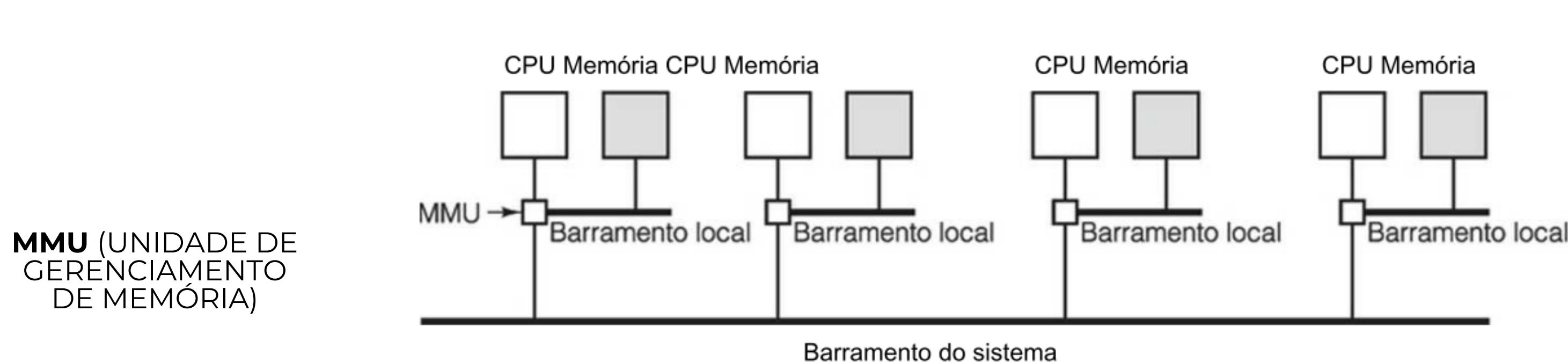
- Há um único espaço de endereço visível a todas as CPUs;
- O acesso à memória remota é feito usando instruções LOAD e STORE;
- O acesso à memória remota é mais lento do que o acesso à memória local.
  - **NC-NUMA:** Quando não há cache o sistema é denominado;
  - **CC-NUMA:** Quando estão presentes caches coerentes.

# NUMA- NON UNIFORM MEMORY ACCESS

## NC-NUMA

### Troca de informações:

- A MMU em cada referência à memória verifica se o endereço pertencia à memória local:
  - Se sim, a requisição é feita à memória local, via barramento local;
  - Se não, é feito através do barramento de sistema.



# NUMA- NON UNIFORM MEMORY ACCESS

## NC-NUMA

### Observações:

- A coerência dos dados é garantida porque não existe caching;
- Porém, um dado no lugar “errado” gera penalidades ao sistema, busca direta ao barramento de sistema;
- Para amenizar o problema, usam-se esquemas implementados em software.
  - Scanner: verifica se a página está no lugar errado, se sim, ele remove esta página da memória para que da proxima vez ocorra um “page fault”;
  - Posteriormente coloca essa página em uma memória diferente daquela em que estava antes.

# NUMA- NON UNIFORM MEMORY ACCESS

CC-NUMA

## Definição

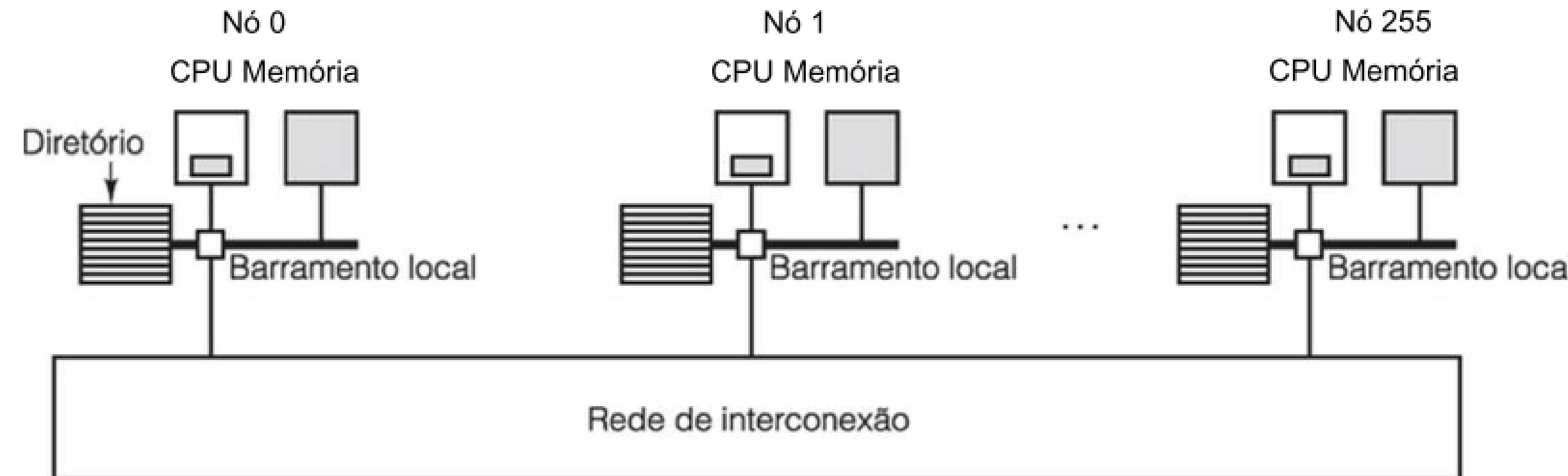
- A ideia é manter um banco de dados que informa onde está cada linha de cache e em qual estado ela está;
- Quando uma linha de cache é referenciada, o banco de dados é pesquisado para descobrir onde ela está e se está limpa ou suja (modificada);
- Para amenizar o problema, usam-se esquemas implementados em software.
  - Scanner: verifica se a página está no lugar errado, se sim, ele remove esta página da memória para que da proxima vez ocorra um “page fault”;
  - Posteriormente coloca essa página em uma memória diferente daquela em que estava antes.

# NUMA- NON UNIFORM MEMORY ACCESS

## CC-NUMA

### Exemplo:

- Sistema com 256 nós, 1 processador por nó, 16 MB por nó (  $256 \times 16 = 4\text{GB}$  RAM = endereços de 32 bits). A memória dos nós é acssada em blocos de 64 bytes.



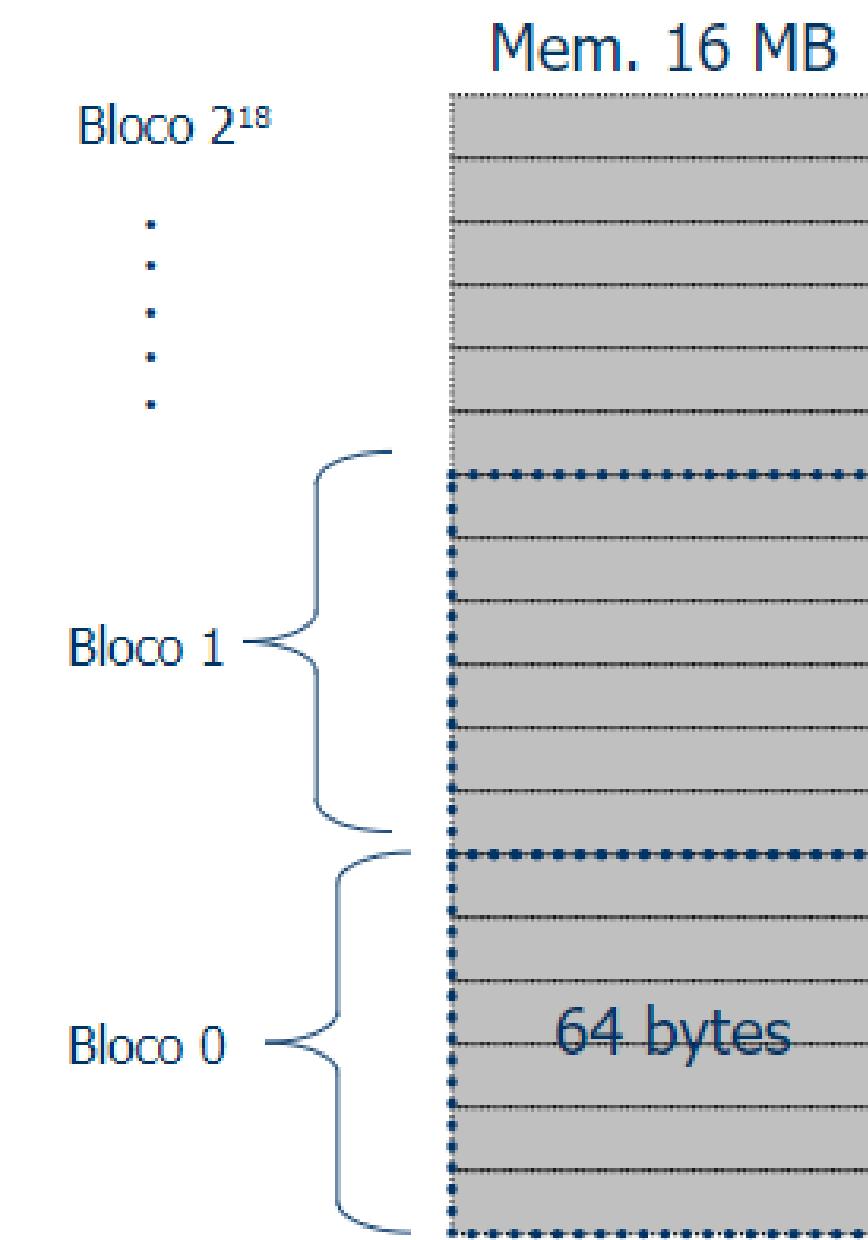
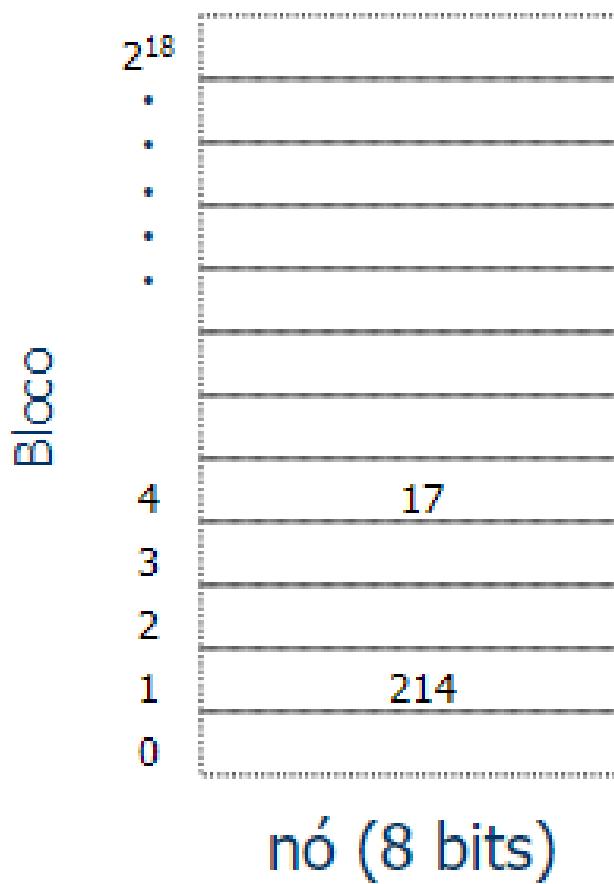
# NUMA- NON UNIFORM MEMORY ACCESS

CC-NUMA

Exemplo:

NÓ 89

Diretório

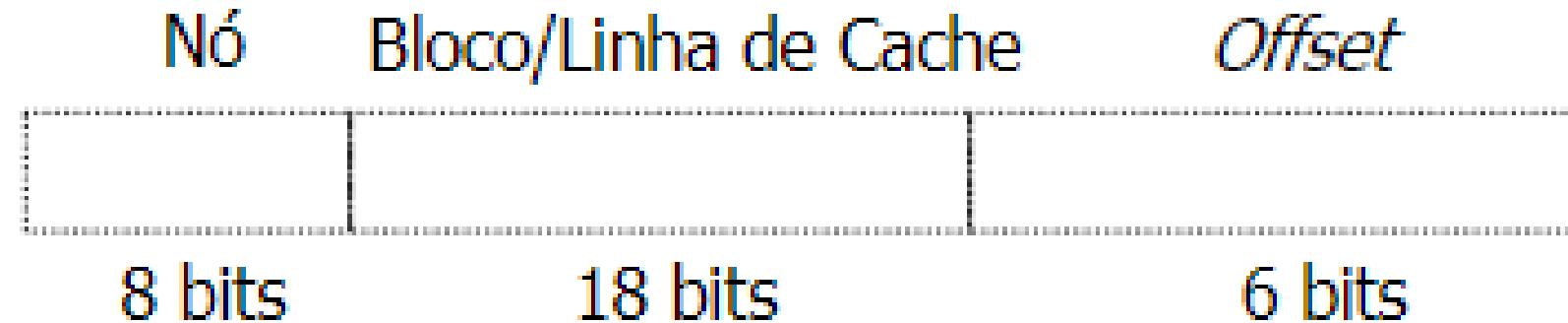


# NUMA- NON UNIFORM MEMORY ACCESS

## CC-NUMA

### Exemplo:

- CPU 20 executa a instrução:
  - LOAD 0x24000108 (em decimal: nó 36, linha 4 e deslocamento 8)
- MMU converte o endereço lógico em físico;



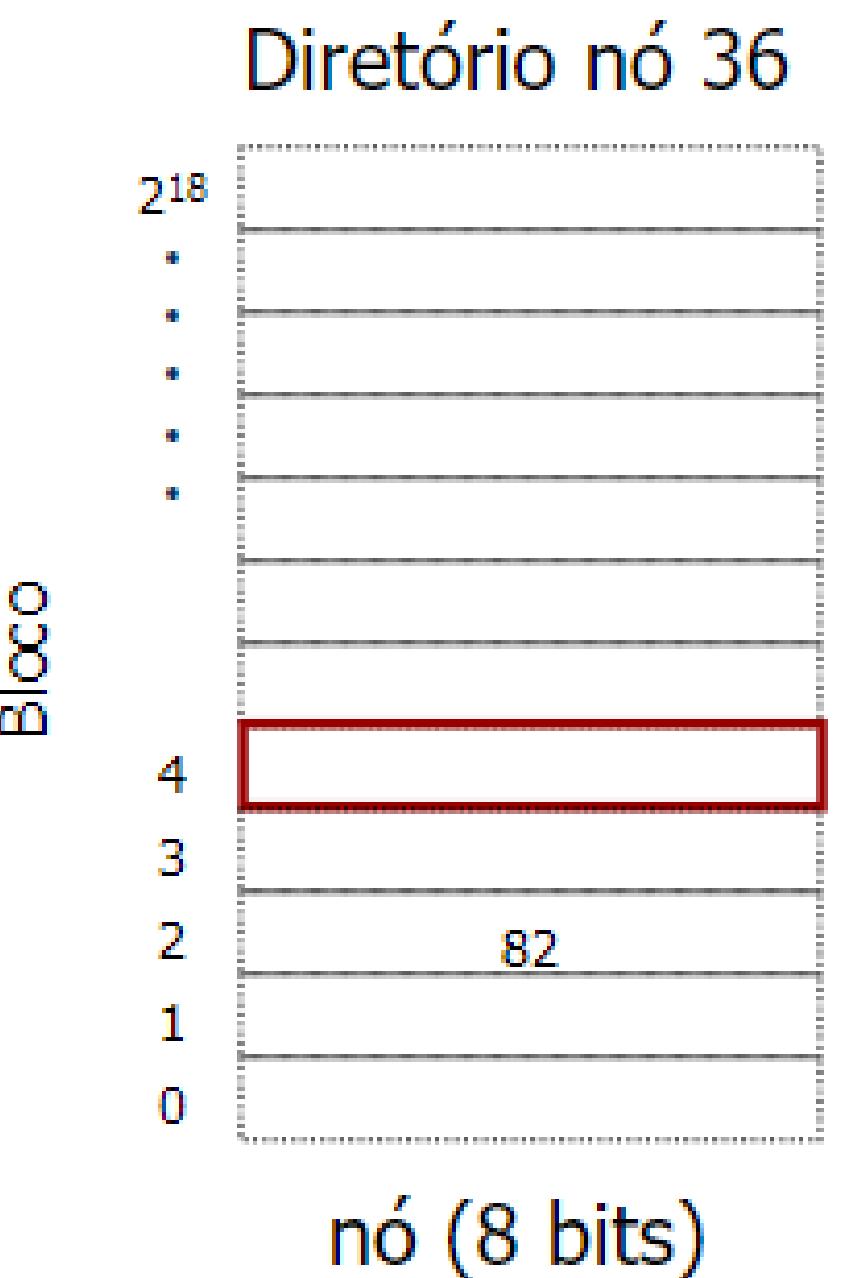
- Verifica sua memória cache;
- Não encontrando, evia requisição para nó 36 solicitando o bloco 4;

# NUMA- NON UNIFORM MEMORY ACCESS

CC-NUMA

## Exemplo:

- Hardware do nó 36 verifica o diretório;
- Entrada do bloco 4 vazia (não se encontra em cache);
- Hardware do diretório 36 envia o bloco 4 à CPU 20 e atualiza o diretório.



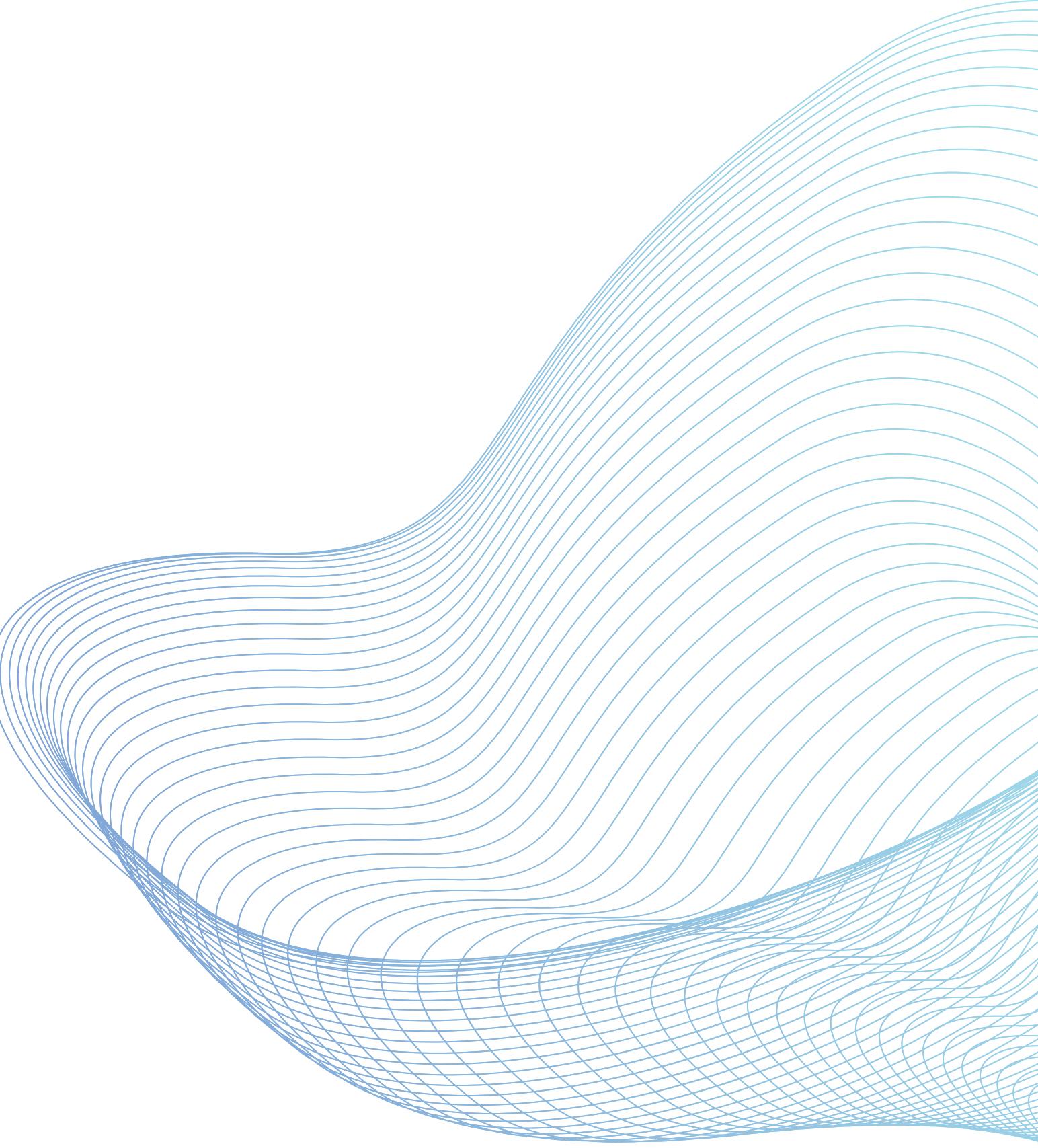
# NUMA- NON UNIFORM MEMORY ACCESS

## CC-NUMA

### Observações:

- Uma limitação é que uma linha só pode ser colocada em cache em um único nó
- Há várias opções para permitir cache em vários nós:
  - Dar a cada entrada de diretório k campos para especificar outros nós;
  - Substituir o número do nó por um mapa de bits, com um bit por nó;
  - Manter um campo de 8 bits em cada entrada de diretório e usá-lo como o cabeçalho de uma lista encadeada que enfileira todas as cópias da linha de cache.
- Monitorar se a linha de cache está limpa ou suja:
  - **Limpa:** o nó nativo pode cumprir a requisição;
  - **Suja:** deve ser passada para o nó que contém acópia válida.

# MULTIPROCESSADORES COMA - CACHE ONLY MEMORY ACCESS



# COMA - CACHE ONLY MEMORY ACCESS

## Definição:

- Usa a memória principal de cada CPU como uma cache;
- As páginas não têm máquinas nativas fixas, como acontece em máquinas NUMA e CC-NUMA;
- O espaço de endereço físico é subdividido em linhas de cache, que migram pelo sistema por demanda.

# COMA - CACHE ONLY MEMORY ACCESS

## Observação

- **Problema:** se a linha não estiver na cache verdadeira de hardware, não existe uma maneira fácil de dizer se ela está na memória principal;
- **Soluções para o problema da localização:**
  - Adicionado novo hardware para monitorar o rótulo de cada linha em cache;
  - Mapear páginas inteiras, mas não exigir que todas as linhas de cache estejam presentes (COMA Simples);
  - Dar a cada página uma máquina de residência para, no mínimo, localizar a linha de cache;
  - Organizar memória como uma árvore e procurar de baixo para cima até encontrar a linha;

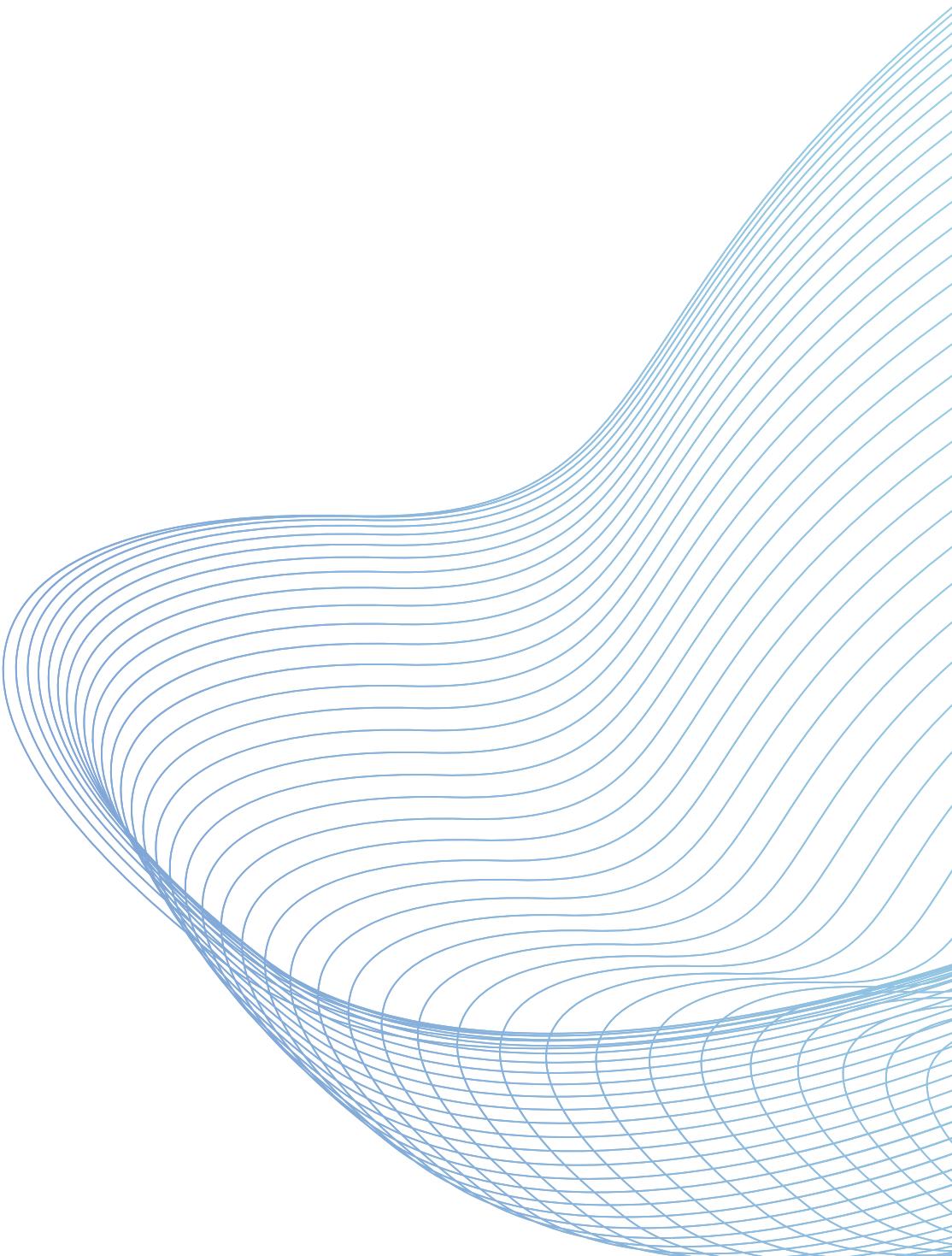
# COMA - CACHE ONLY MEMORY ACCESS

## Observação

- **Problema:** não remoção da última cópia;
- **Soluções para o problema da localização:**
  - Voltar ao diretório e verificar se há outras cópias
  - Identificar uma cópia de cada linha de cache como a cópia mestra e nunca jogá-la fora;

# MULTIPROCESSADORES

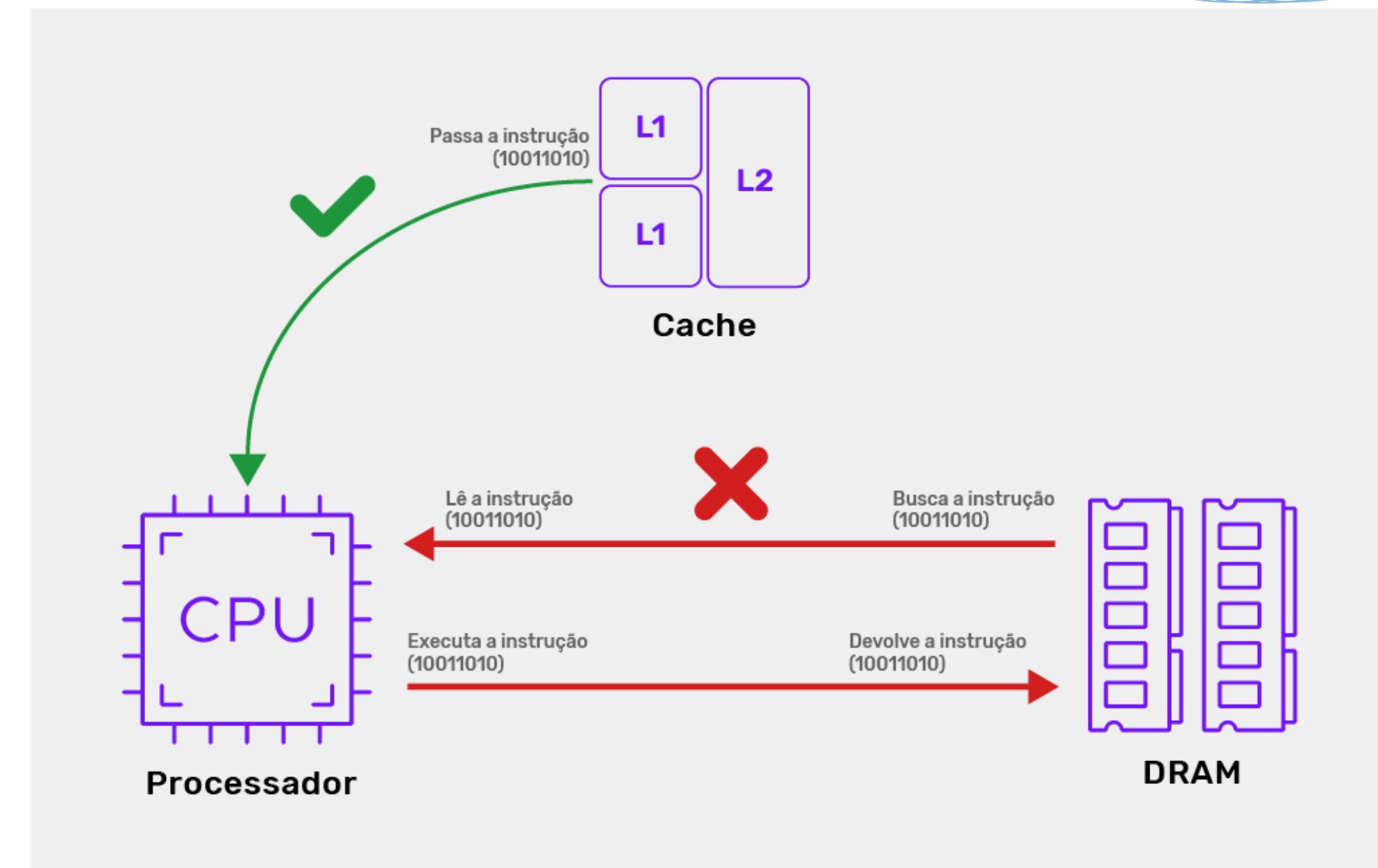
## COERÊNCIA DE MEMÓRIA CACHE



# COERÊNCIA DE MEMÓRIA CACHE

## Definição

- Funciona é uma memória de acesso rápido. Ela tem o objetivo de guardar dados, informações e processos temporários acessados com frequência e assim agilizar o processo de uso no momento em que são requisitados pelo usuário.



# COERÊNCIA DE MEMÓRIA CACHE

## Protocolo de coerência de cache de escrita direta

- Protocolo de coerência de cache de escrita direta. Os retângulos vazios indicam que nenhuma ação foi realizada.

Ação	Requisição local	Requisição remota
Ausência da <i>cache</i> para leitura	Busque dados da memória	
Presença na <i>cache</i> para leitura	Use dados da <i>cache</i> local	
Ausência da <i>cache</i> para escrita	Atualize dados na memória	
Presença na <i>cache</i> para escrita	Atualize <i>cache</i> e memória	Invalide entrada de <i>cache</i>

# COERÊNCIA DE MEMÓRIA CACHE

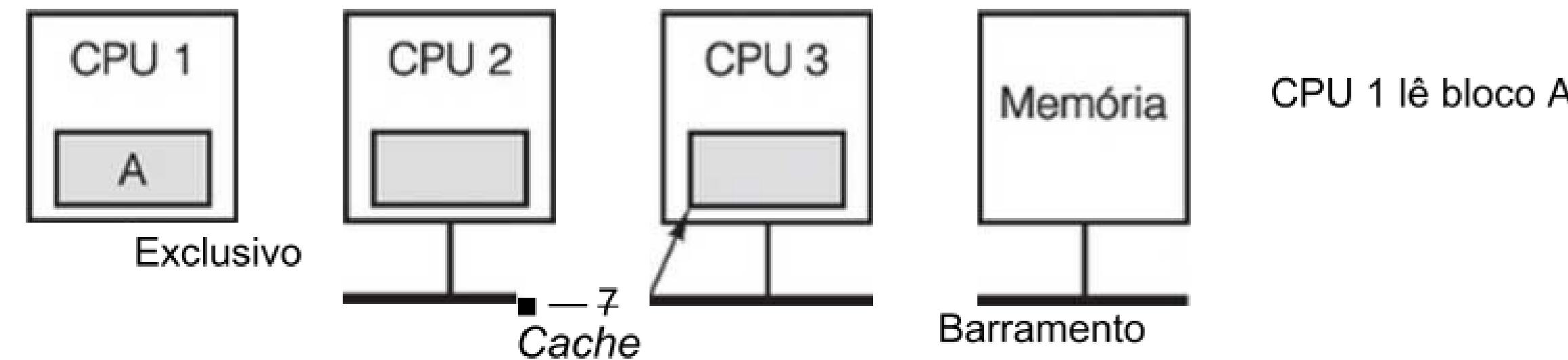
## Protocolo MESI de coerência de cache

- O protocolo MESI é usado pelo Core i7 e por muitas outras CPUs para espiar o barramento. Cada entrada de cache pode estar em um dos quatro estados:
  1. Inválido - A entrada da cache não contém dados válidos;
  2. Compartilhado (shared) - múltiplas caches podem conter a linha; a memória está atualizada;
  3. Exclusivo - nenhuma outra cache contém a linha; a memória está atualizada;
  4. Modificado - a entrada é válida; a memória é inválida; não existem cópias.

# COERÊNCIA DE MEMÓRIA CACHE

## Protocolo MESI de coerência de cache

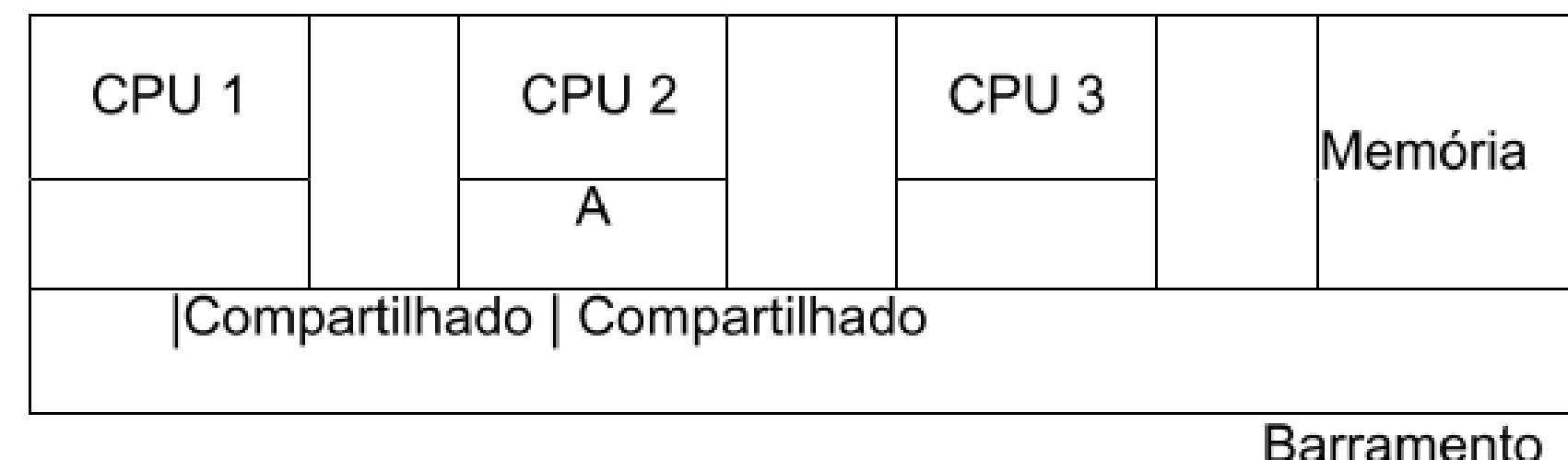
- Quando a CPU é iniciada pela primeira vez, todas as entradas de cache são marcadas como inválidas;
- Na primeira vez que a memória é lida, a linha referenciada é buscada na cache da CPU que está lendo a memória e marcada como no estado E (exclusivo)



# COERÊNCIA DE MEMÓRIA CACHE

## Protocolo MESI de coerência de cache

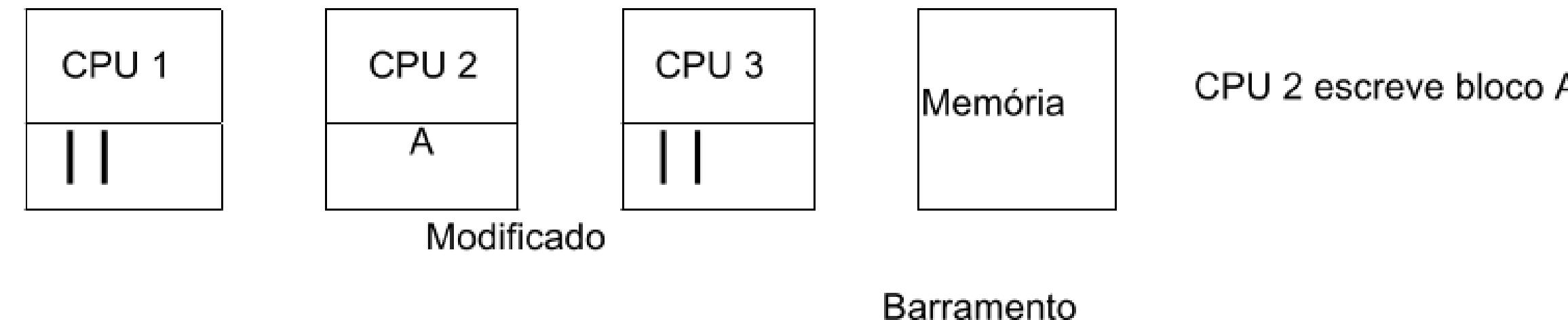
- Outra CPU também pode buscar a mesma linha e colocá-la na cache;
- Portador original (CPU 1) vê que não está mais sozinho e anuncia no barramento que ele também tem uma cópia;
- Ambas as cópias são marcadas como estado S (compartilhado).



# COERÊNCIA DE MEMÓRIA CACHE

## Protocolo MESI de coerência de cache

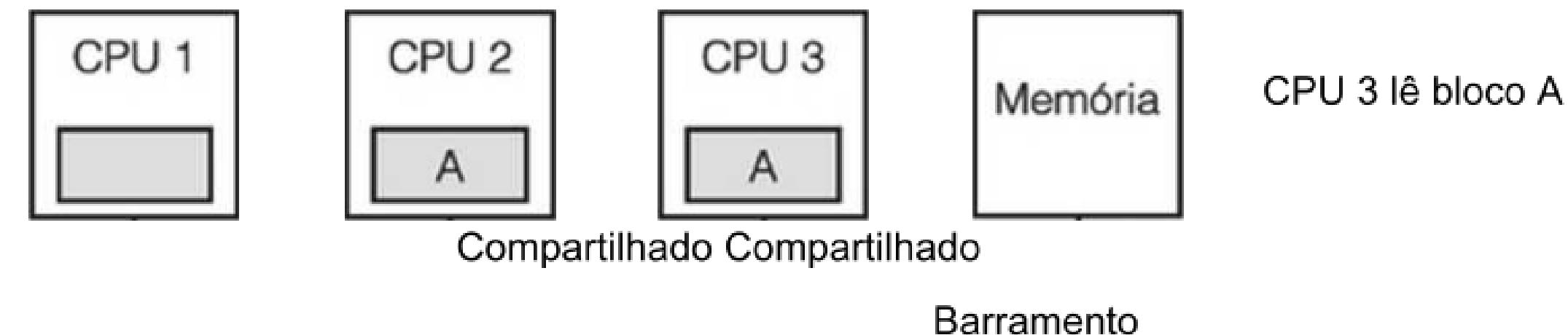
- CPU 2 escrever para a linha de cache que ela está mantendo no estado Compartilhado;
- Ela emite um sinal de invalidação no barramento, informando a todas as outras CPUs para descartar suas cópias;
- A cópia que está em cache agora passa para o estado M (modificado).



# COERÊNCIA DE MEMÓRIA CACHE

## Protocolo MESI de coerência de cache

- Considere o que acontece se a CPU 3 ler a linha;
- A CPU 2, que agora possui a linha, sabe que a cópia na memória não é válida;
- A CPU 2 ativa um sinal no barramento para à CPU 3 esperar enquanto ela escreve sua linha de volta para a memória;
- Quanto a CPU 2 concluir, a CPU 3 busca uma cópia e a linha é marcada como compartilhada em ambas as caches;

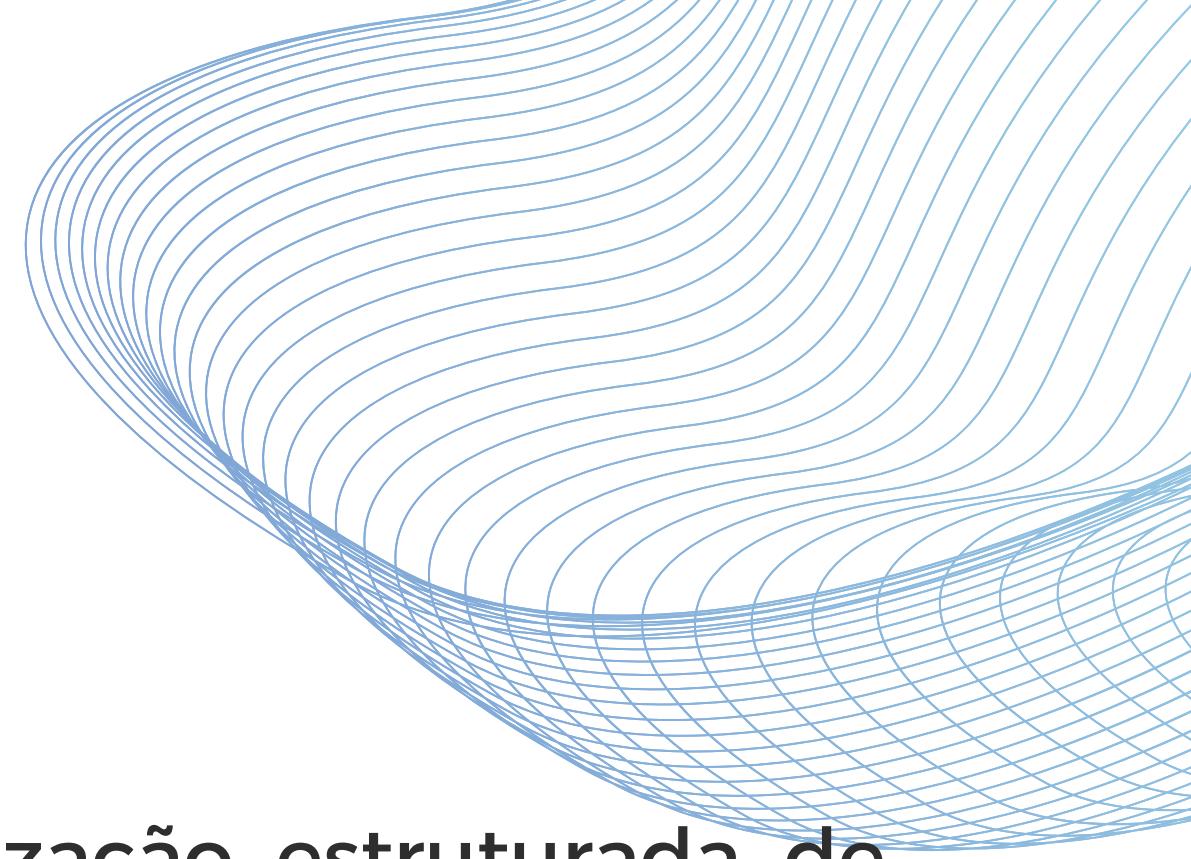


# COERÊNCIA DE MEMÓRIA CACHE

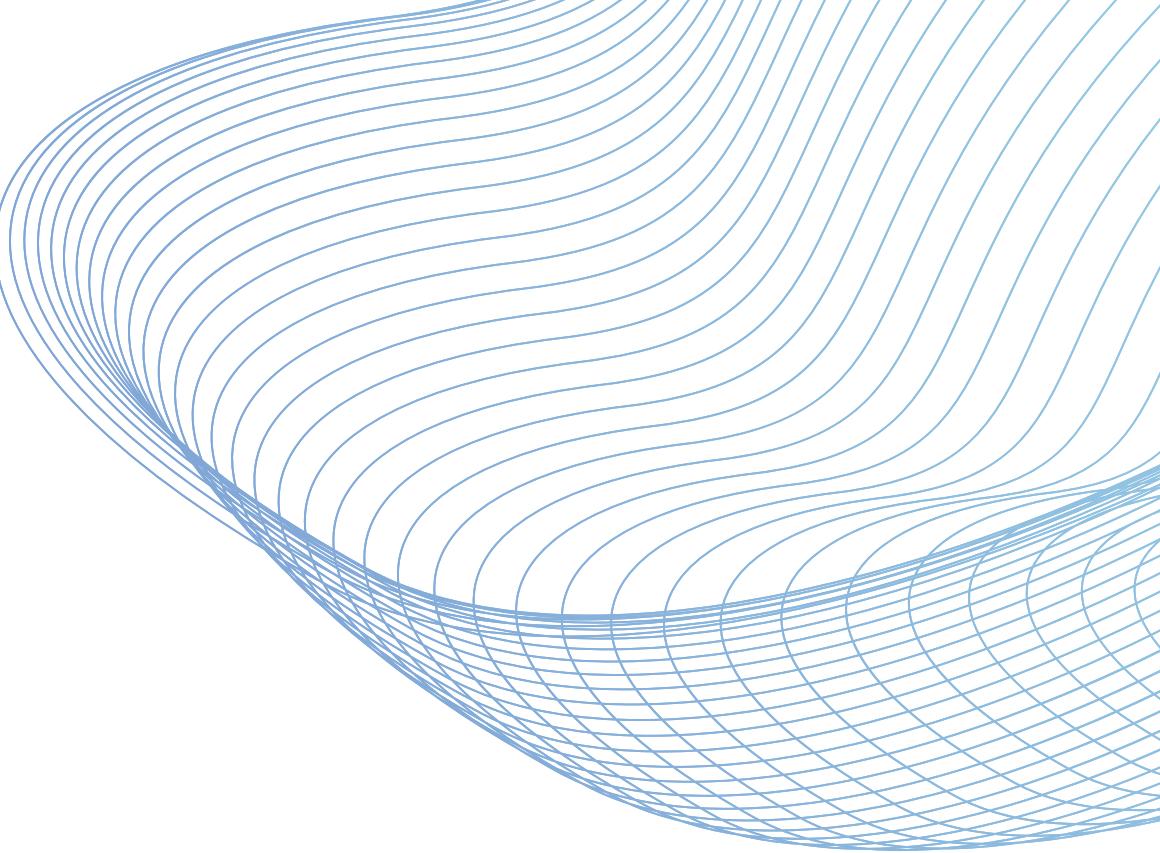
## Trabalhos recentes sobre coerência de cache

- WILKINS, Michael et al. WARDen: Specializing Cache Coherence for High-Level Parallel Languages. In: Proceedings of the 21st ACM/IEEE International Symposium on Code Generation and Optimization. 2023. p. 122-135.
- UMA, V.; MARIMUTHU, Ramalatha. D-wash-A dynamic workload aware adaptive cache coherence protocol for multi-core processor system. Microelectronics Journal, v. 132, p. 105675, 2023.

# BIBLIOGRAFIA



- TANENBAUM, Andrew S.; ZUCCHI, Wagner Luiz. Organização estruturada de computadores. Pearson Prentice Hall, 2009.
- BIENIA, Christian. Benchmarking modern multiprocessors. Princeton University, 2011.
- STALLINGS, William. Arquitetura e Organização de Computadores 8a Edição. 2010.



**OBRIGADO!**  
**ALGUMA DÚVIDA?**

E-mail: thommaskevin@gmail.com

