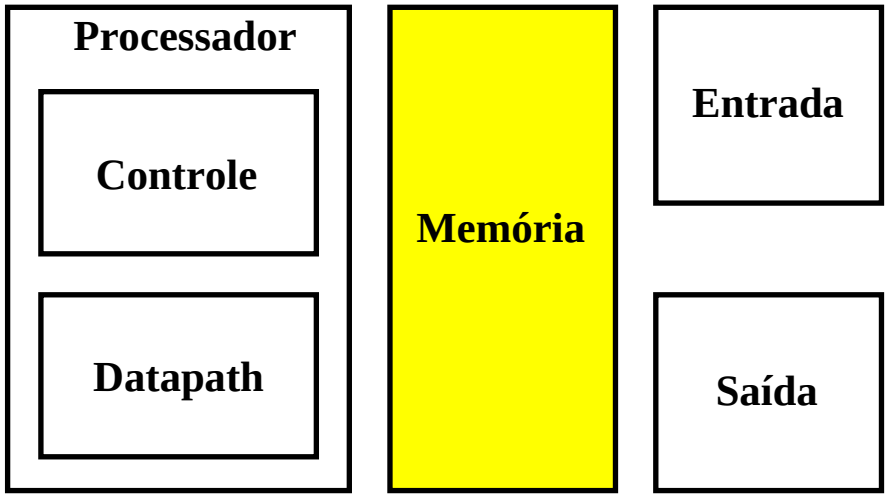
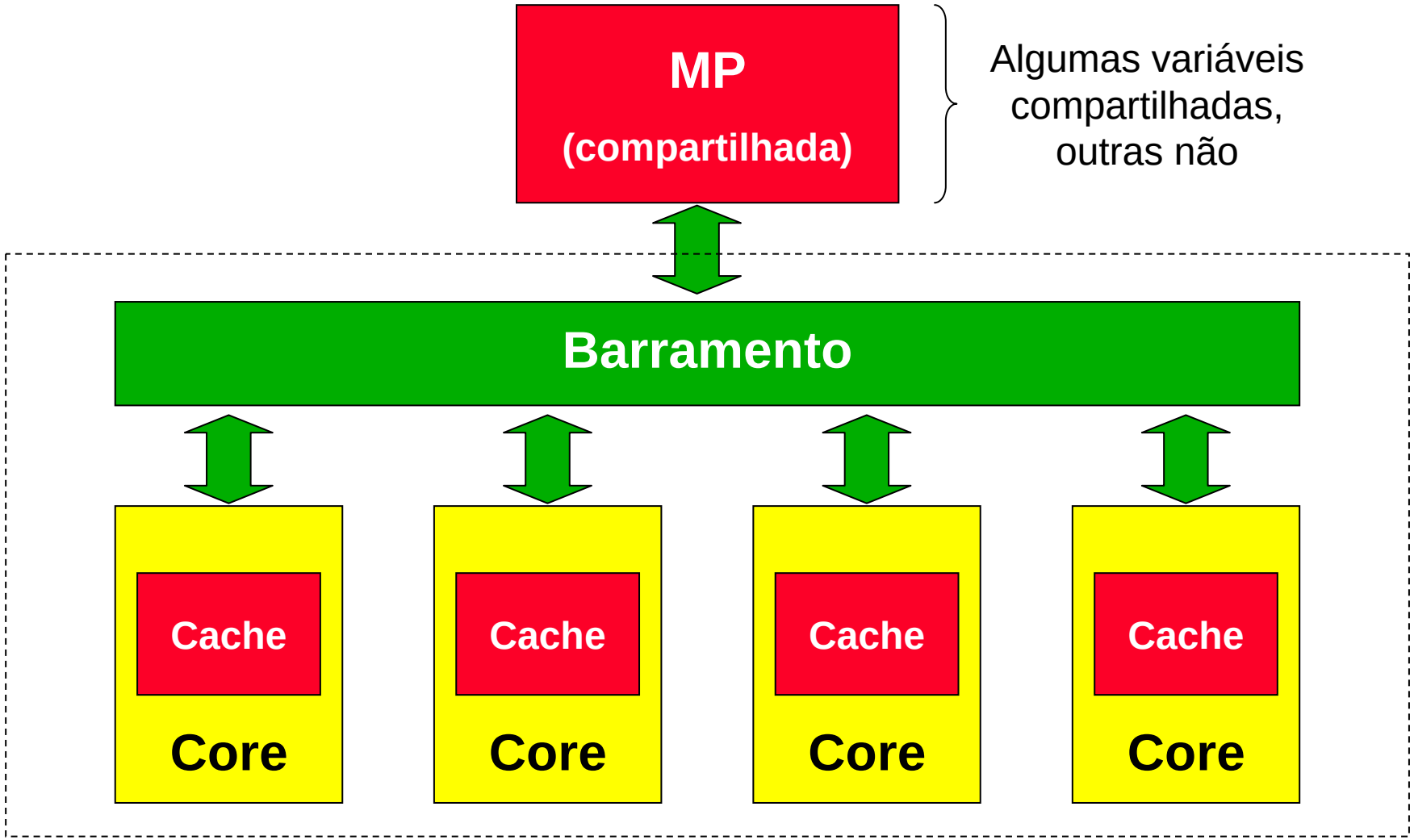
Paralelismo e hierarquias de memória: coerência de cache



CMP: “on-chip multiprocessing”



O problema da coerência de cache

- **Hipótese:**
  - **Uma única posição de memória (X)**
  - **Lida e escrita por dois processadores (A e B)**
  - **“Write through”**

Time step	Event	Cache contents for CPU A	Cache contents for CPU B	Memory contents for location X

O problema da coerência de cache

- **Hipótese:**
  - **Uma única posição de memória (X)**
  - **Lida e escrita por dois processadores (A e B)**
  - **“Write through”**

Time step	Event	Cache contents for CPU A	Cache contents for CPU B	Memory contents for location X
0				0

O problema da coerência de cache

- **Hipótese:**
  - **Uma única posição de memória (X)**
  - **Lida e escrita por dois processadores (A e B)**
  - **"Write through"**

Time step	Event	Cache contents for CPU A	Cache contents for CPU B	Memory contents for location X
0				0
1	CPU A reads X	0		0

O problema da coerência de cache

- **Hipótese:**
  - **Uma única posição de memória (X)**
  - **Lida e escrita por dois processadores (A e B)**
  - **"Write through"**

Time step	Event	Cache contents for CPU A	Cache contents for CPU B	Memory contents for location X
0				0
1	CPU A reads X	0		0
2	CPU B reads X	0	0	0

O problema da coerência de cache

- **Hipóteses:**
  - **Uma única posição de memória (X)**
  - **Lida e escrita por dois processadores (A e B)**
  - **"Write through"**

Time step	Event	Cache contents for CPU A	Cache contents for CPU B	Memory contents for location X
0				0
1	CPU A reads X	0		0
2	CPU B reads X	0	0	0
3	CPU A stores 1 into X	1	0	1

Coerência e consistência

- **Um sistema de memória é coerente quando a leitura de um item retorna o valor mais recentemente escrito naquele item?**
- **Definição é simplista**
- **Na verdade, contém dois diferentes aspectos do comportamento do sistema de memória**
- **Coerência:**
  - **Quais valores podem ser retornados por uma leitura**
- **Consistência:**
  - **Quando um valor escrito será retornado por uma leitura**

Coerência: requisito 1

- **O sistema de memória é coerente se preserva a ordem do programa, ou seja:**
- **Processador P escreve na posição X**
- **Nenhum outro processador faz escrita intermediária em X**
- **Processador P lê a posição X e o valor retornado é o próprio valor escrito por P**

Coerência: requisito 2

- **O sistema de memória é coerente se provê uma visão coerente da memória, ou seja:**
- **Processador P1 escreve na posição X**
- **Nenhum outro processador faz escrita intermediária em X**
- **O próximo acesso de leitura a X está suficientemente afastado de sua escrita**
- **Processador P2 lê a posição X e o valor retornado é o valor escrito por P1**

Coerência: requisito 3

- **O sistema de memória é coerente se provê uma a mesma ordem de escrita para todos os processadores, ou seja:**
- **Processador P1 escreve na posição X**
- **Processador P2 escreve na posição X**
- **Leituras de X feitas por P1 ou P2 enxergam os valores escritos na mesma ordem**

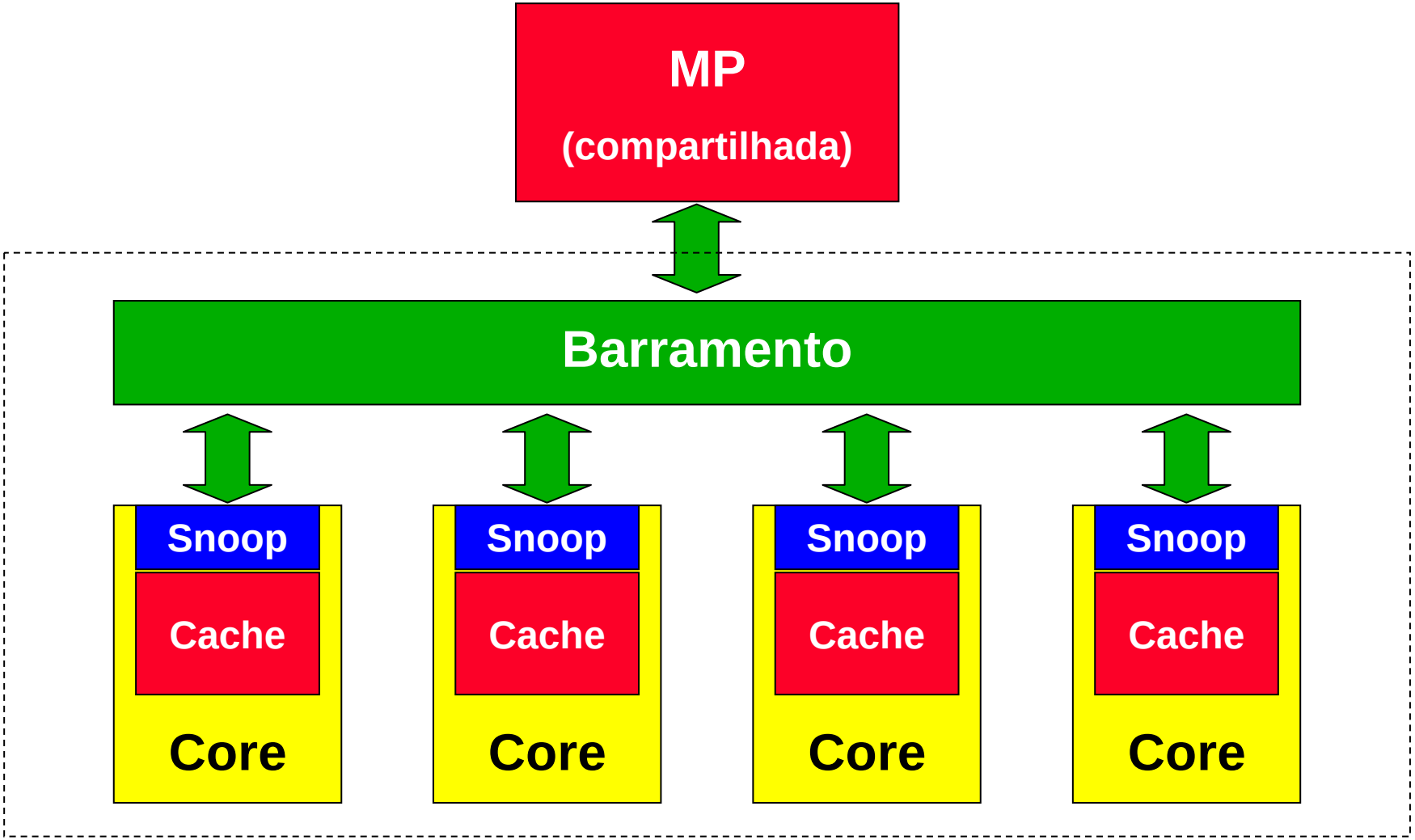
Técnicas para garantir coerência

- **Protocolos de coerência de cache**
  - **Distribuído: “Snooping”**
    - » **Menor “overhead”**
    - » **Uso: pequeno número de “cores”**
  - **Centralizado: Diretório**
    - » **Maior “overhead”**
    - » **Uso: grande número de “cores”**
- **Chave para implementação**
  - **Capturar o estado de qualquer compartilhamento de um bloco**

“Snooping”

- **Requisito 1: meio compartilhado que faz “broadcasting” de faltas na cache**
  - **Exemplo: barramento compartilhado**
- **Requisito 2: controladores de cache são estendidos para “espionar” o barramento**

CMP: “on-chip multiprocessing”



“Snooping”

- **Idéia-chave:**
  - **Ao escrever um item da cache, um processador deve invalidar outras cópias daquele item em outras caches**
- **Consequências:**
  - **Protocolo do tipo “write-invalidate”**
  - **Um processador tem exclusividade de acesso a um item antes de nele escrever**

Exemplo da ação do “snooping”

- **Hipóteses:**
  - **Posição X lida e escrita por processadores A e B**
  - **“Write back” (quando bloco torna-se compartilhado)**

Processor activity	Bus activity	Contents of CPU A’s cache	Contents of CPU B’s cache	Contents of memory location X
				0

Inicialmente, nenhuma das caches contém o valor de X

Exemplo da ação do “snooping”

- **Hipóteses:**
  - **Posição X lida e escrita por processadores A e B**
  - **“Write back” (quando bloco torna-se compartilhado)**

Processor activity	Bus activity	Contents of CPU A’s cache	Contents of CPU B’s cache	Contents of memory location X
				0
CPU A reads X	Cache miss for X	0		0

A lê X

Exemplo da ação do “snooping”

- **Hipóteses:**
  - **Posição X lida e escrita por processadores A e B**
  - **“Write back” (quando bloco torna-se compartilhado)**

Processor activity	Bus activity	Contents of CPU A’s cache	Contents of CPU B’s cache	Contents of memory location X
				0
CPU A reads X	Cache miss for X	0		0
CPU B reads X	Cache miss for X	0	0	0

B lê X

Exemplo da ação do “snooping”

- **Hipóteses:**
  - **Posição X lida e escrita por processadores A e B**
  - **“Write back” (quando bloco torna-se compartilhado)**

Processor activity	Bus activity	Contents of CPU A’s cache	Contents of CPU B’s cache	Contents of memory location X
				0
CPU A reads X	Cache miss for X	0		0
CPU B reads X	Cache miss for X	0	0	0
CPU A writes a 1 to X	Invalidation for X	1		0

A escreve no bloco em sua cache e invalida bloco na cache de B

Exemplo da ação do “snooping”

- **Hipóteses:**
  - **Posição X lida e escrita por processadores A e B**
  - **“Write back” (quando bloco torna-se compartilhado)**

Processor activity	Bus activity	Contents of CPU A’s cache	Contents of CPU B’s cache	Contents of memory location X
				0
CPU A reads X	Cache miss for X	0		0
CPU B reads X	Cache miss for X	0	0	0
CPU A writes a 1 to X	Invalidation for X	1		0
CPU B reads X	Cache miss for X	1	1	1

Falta na cache de B: **A fornece o novo valor a B e o atualiza na MP**

Cenário 1: leitura sob “write-back”

- **PA executa “write invalidate” de um item X**
  - **Em CA**
- **Quando, mais tarde, PB referencia X para leitura**
  - **Ocorrerá falta em CB**
- **Forçando**
  - **PA a atualizar MP**
    - » **Pois o bloco tornou-se compartilhado**

Cenário 1: leitura sob “write-back”

- **PA executa “write invalidate” de um item X**
  - **Em CA**
- **Quando, mais tarde, PB referencia X para leitura**
  - **Ocorrerá falta em CB**
- **Forçando**
  - **PA a atualizar MP e atualizar CB**
    - » **Pois o bloco tornou-se compartilhado**
    - » **Evitando ter que esperar para lê-lo a partir de MP**

Cenário 2: escrita sob “write-back”

- **PA executa “write invalidate” de um item X**
  - **Em CA**
- **Quando, mais tarde, PB referencia X para escrita**
  - **Ocorrerá “write invalidate” em CB**
- **Forçando**
  - **PA a atualizar MP**
    - » **Antes de o bloco “sujo” ser invalidado em CA**
    - » **Pois bloco a ser invalidado não foi atualizado em MP**

Cenário 2: escrita sob “write-back”

- **PA executa “write invalidate” de um item X**
  - **Em CA**
- **Quando, mais tarde, PB referencia X para escrita**
  - **Ocorrerá “write invalidate” em CB**
- **Forçando**
  - **PA a atualizar MP e atualizar CB**
    - » **Antes de o bloco “sujo” ser invalidado em CA**
    - » **Pois bloco a ser invalidado não foi atualizado em MP**
    - » **Evitando ter que lê-lo a partir da MP*...**
    - » **Antes de PB escrever em CB**

Cenário 3: leitura sob “write-through”

- **PA executa “write invalidate” de um item X**
  - **Em CA**
- **Quando, mais tarde, PB referencia X para leitura**
  - **Ocorrerá falta na CB**
- **Forçando**
  - **CB a ler o bloco invalidado a partir da MP**

Cenário 4: escrita sob “write-through”

- **PA executa “write invalidate” de um item X**
  - **Em CA**
- **Quando, mais tarde, PB referencia X para escrita**
  - **Ocorrerá “write invalidate” na CB**
- **Forçando**
  - **PB a atualizar MP**