Hierarquia de Memória II

Prof. Gustavo Girão

*Baseado no material do Professor Leandro Galvão

Questões Básicas na Hierarquia de Memória

I. **Posicionamento do bloco**.
Onde o bloco deve ser colocado na memória de nível mais alto?

II. **Substituição de bloco**.
Quais blocos serão trocados em um miss?

III. **Estratégia de gravação**.
O que acontece em uma escrita?

I – Posicionamento do bloco

- **Mapeamento direto (*directed-mapped*)**
  - Cada bloco pode ser colocado em uma única posição na cache
  - Vantagem: hardware mais simples
    - Para determinar se houve *cache hit* basta consultar UMA linha da cache.
  - Desvantagem: potencial desperdício de espaço
    - Pode existir o caso em que exista espaço livre na cache, porém são espaços que não podem ser ocupados pelo bloco vindo da memória (regra de mapeamento)

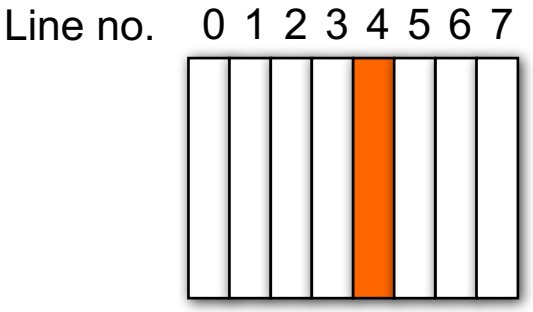
I – Posicionamento do bloco

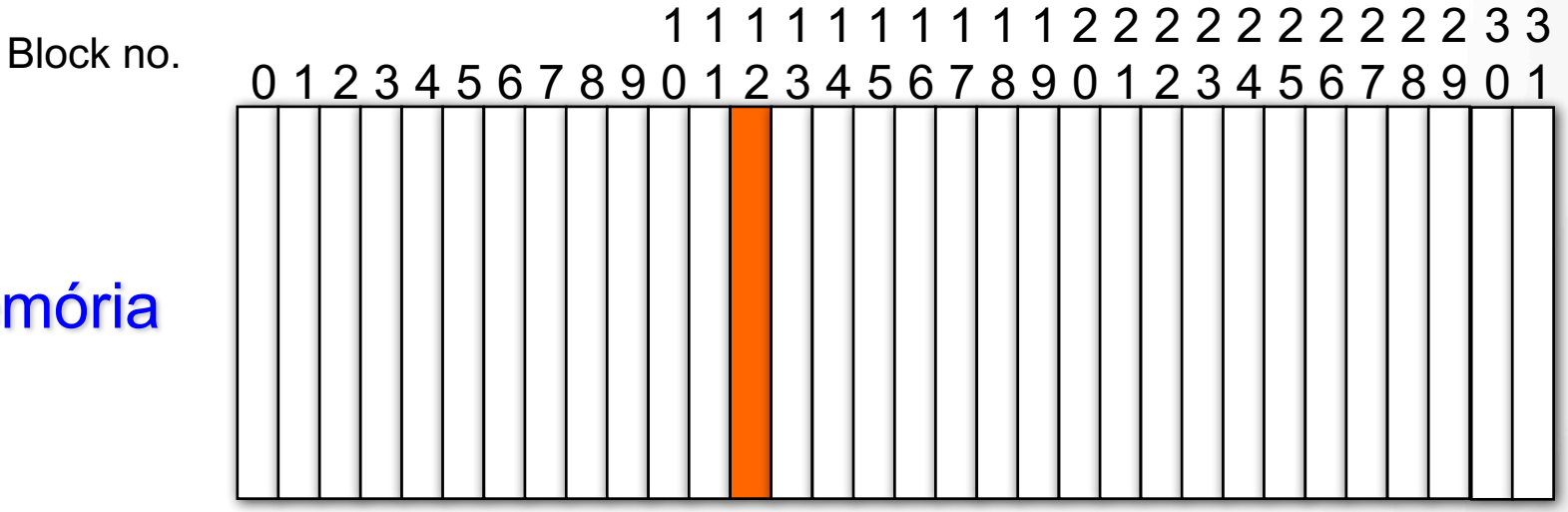
Mapeamento direto

Regra de mapeamento:

do bloco % # de linhas da cache

(12 % 8) = 4





I – Posicionamento do bloco

- **Totalmente associativa (*fully associative*)**
  - Cada bloco pode ser colocado em qualquer posição na cache
  - Vantagem: Nunca há desperdício de espaço
    - Enquanto houver espaço disponível na cache, o bloco vindo da memória pode ser alocado.
  - Desvantagem: hardware mais complexo
    - Para saber se houve *cache hit* é preciso comparar TODAS as linhas da cache.
    - Utilizada somente em memórias muito pequenas.

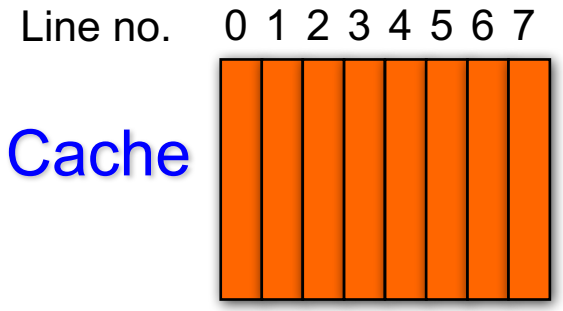
I – Posicionamento do bloco

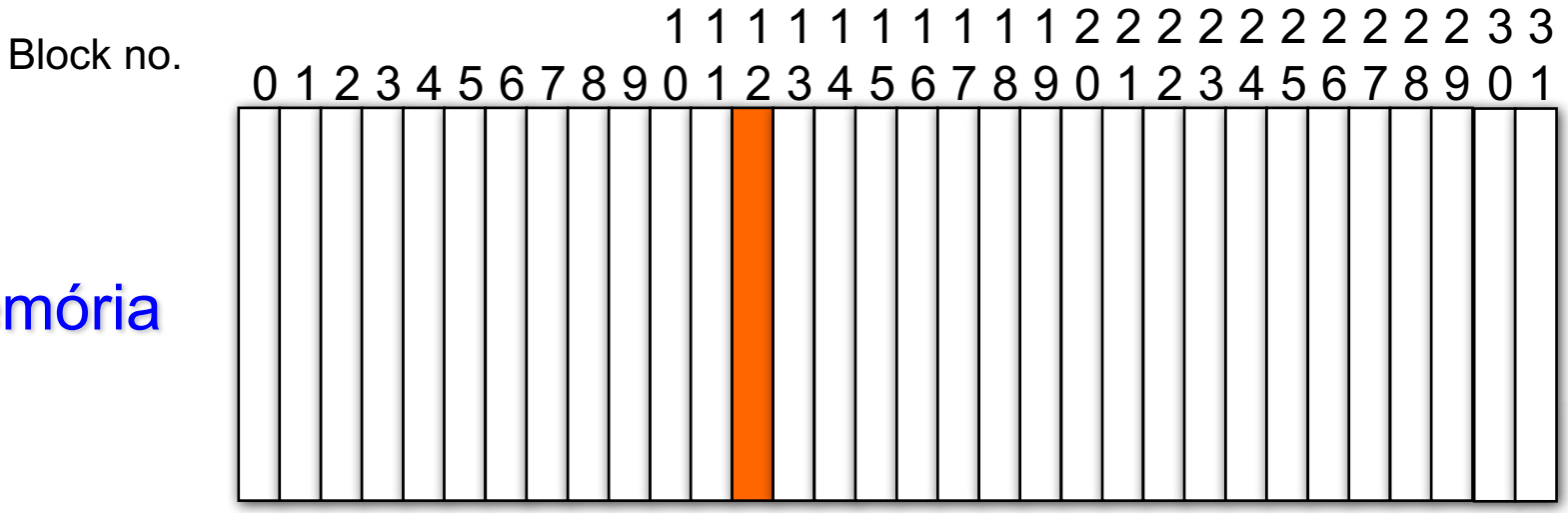
Totalmente Associativa

Mapeamento Totalmente Associativo

Regra de mapeamento:

A próxima linha disponível





I – Posicionamento do bloco

- **Associativa por conjunto (*n-way set-associative*)**
  - Cada bloco pode ser colocado em um conjunto restrito de posições na cache.
  - Solução de compromisso entre as duas anteriores.
    - Se o conjunto for do tamanho da própria cache
      - Completamente Associativo
    - Se o conjunto for de tamanho 1
      - Direto
  - "way" (ou **vias**) identifica quantos **linhas** por conjunto existem.

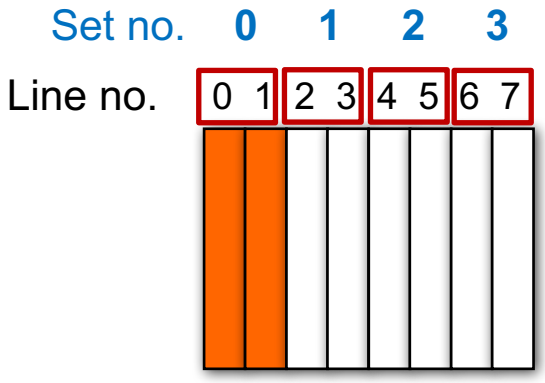
I – Posicionamento do bloco

Associativa por Conjunto:

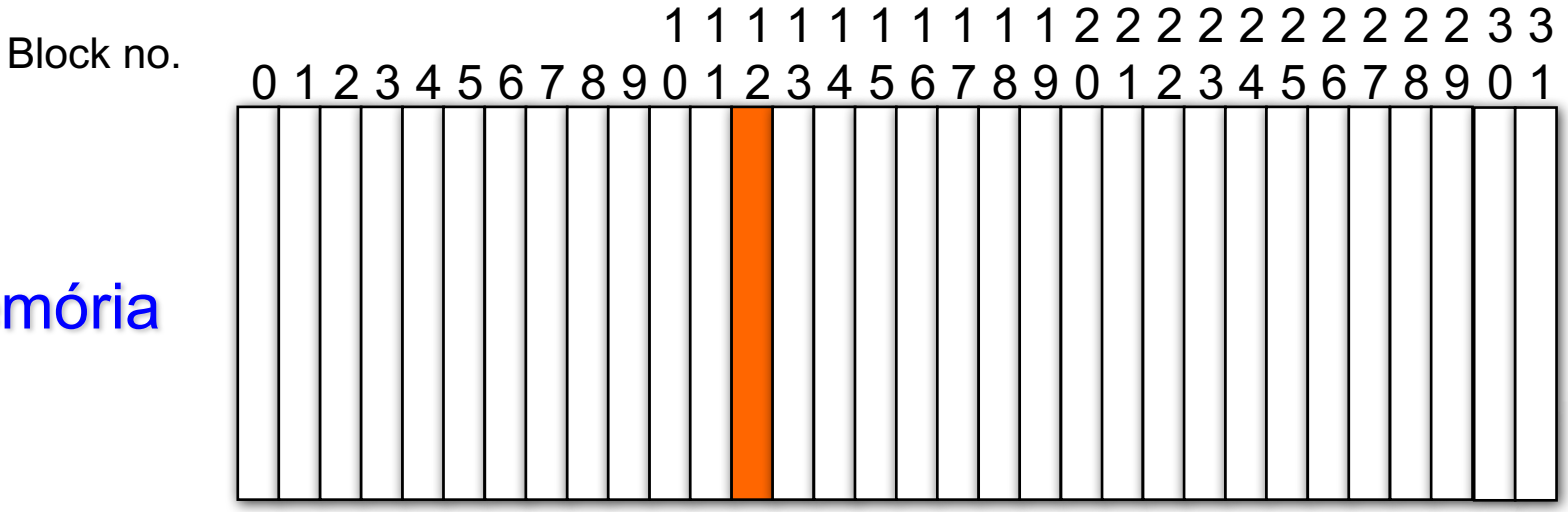
Regra de mapeamento:

do bloco % # de conjuntos

(12 % 4) = Set 0



Memória



I – Posicionamento do bloco

- Em processadores Intel core i7:
  - Cache de dados 32kB L1: associativas de conjunto 8 vias (8-way).
  - Cache de dados 256kB L2: associativas de conjunto 8 vias.
  - Cache 8MB L3 off-core: associativas de conjunto 16 vias.

- O número de vias aumenta o número de locais na cache onde o conteúdo de cada endereço de memória pode ser colocado.

- Assim, faz mais sentido ter mais locais para instruções, onde saltos e desvios abrem maior intervalo de endereçamento.

II – Substituição do bloco

- Uma falha de cache (ou miss) ocorre quando o processador requisita dados que não estão presentes na memória cache.
- Nesse caso, o processador congela seu funcionamento (*stall*), até que a memória cache busque no nível inferior o dado requisitado.
  - Existem algumas alternativas à parada do pipeline. Uma delas é a mudança de contexto para a execução de uma thread diferente como vimos em aulas passadas (*multithreading*)

II – Substituição do bloco

- Mapeamento direto: somente o bloco não encontrado é substituído, simplificando o hardware.
- Demais mapeamentos:
  - Aleatória:
    o bloco a substituir é escolhido aleatoriamente.
  - Menos recentemente usado (LRU):
    substitui-se o bloco que não é usado há mais tempo.
  - Menos frequentemente usado (LFU):
    substitui-se o bloco que foi menos utilizado.
  - Primeiro a entrar, primeiro a sair (FIFO):
    substitui-se o bloco mais antigo (ainda que tenha sido recentemente usado).

II – Substituição do bloco

- Como são Implementados?
  - Aleatória:
    Através de um gerador de números aleatórios (depende da máquina) o bloco a ser removido é escolhido.
    - Baixa custo de implementação
  - Menos recentemente usado (LRU):
    Uma referencia temporal é associada a cada bloco quando este chega na cache ou é acessado, esta referencia é atualizada (incrementada). O bloco a ser retirado é aquele com menor referencia temporal
    - Alto custo de implementação: um registrador de referencia temporal para cada bloco.
    - De tempos em tempos precisa ser redimensionado

II – Substituição do bloco

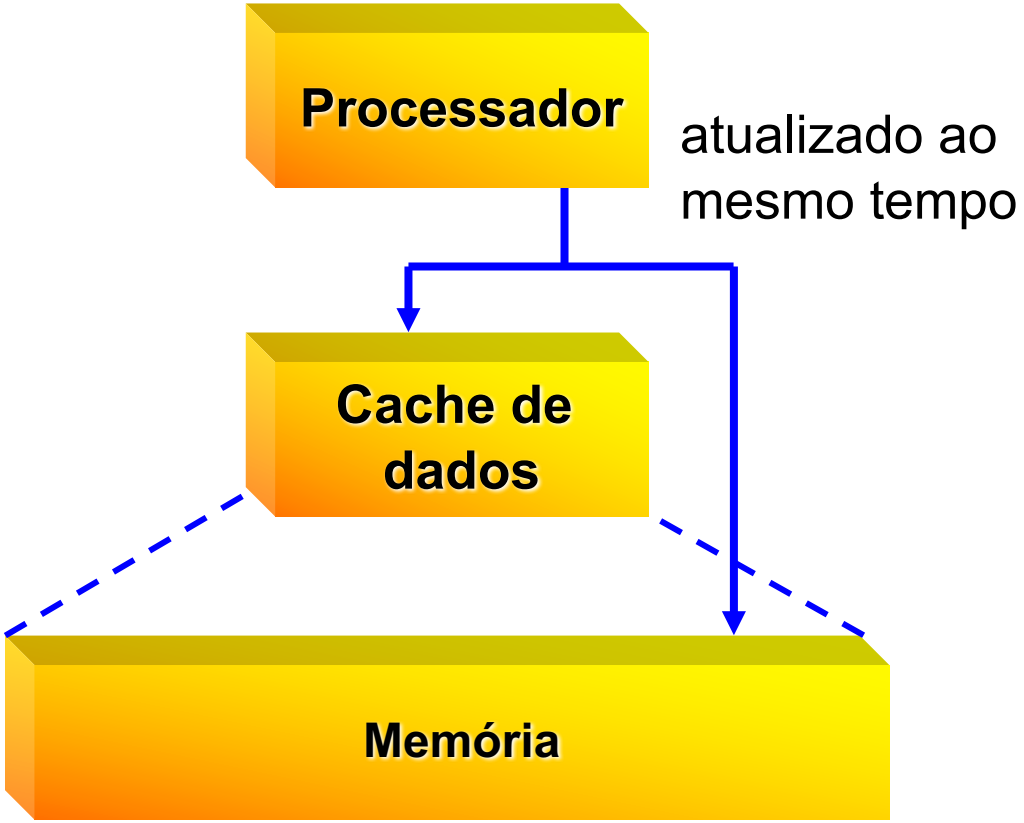
- Como são Implementados?
  - Menos frequentemente usado (LFU):
    Faz uso de um contador de uso (registrador). Sempre que o bloco é acessado (i.e. lido ou escrito) o contador é incrementado. A linha a ser retirada é a de menor contador.
    - Alto custo de implementação: um contador por linha
    - Também precisa de redimensionamento devido à saturação do contador
  - Primeiro a entrar, primeiro a sair (FIFO):
    Utiliza-se uma referencia de ordem dentro da cache. Cada linha trazida para um bloco da cache recebe uma referencia de quando chegou na cache. A linha a ser retirada é a de menor valor. Ao ser retirada as referencias de todas as outras linhas mudam.
    - Custo moderado de implementação e manutenção: um registrador para cada linha, porém tem resolução (numero de bits), pequena.

III – Estratégia de gravação

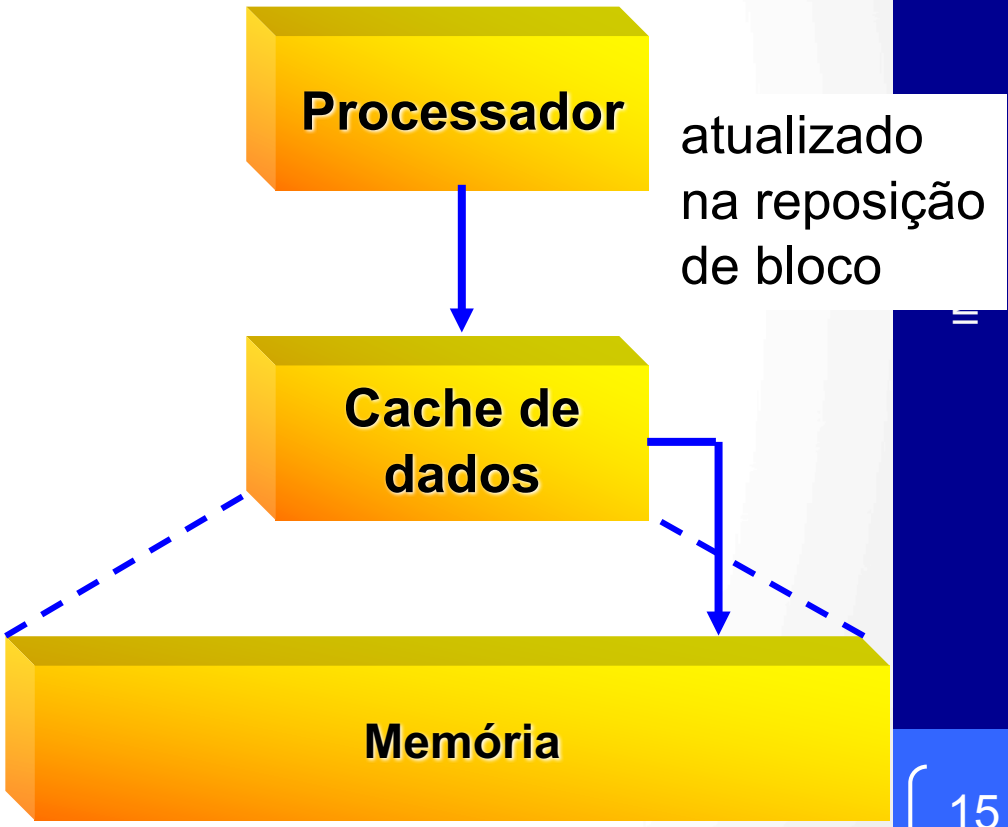
- Dois modos de escrita de dados:
  - Write-through: a informação é escrita tanto para o bloco da cache quanto para a memória de nível mais baixo.
  - Write-back: a informação é escrita somente para o bloco da cache. Este bloco só escrito na memória de nível inferior quando for trocado.

III – Estratégia de gravação

Write-through



Write-back



III – Estratégia de gravação

- Write-back
  - Servidores, por consumir menos largura de banda de memória
  - Sistemas embarcados, por poupar energia ao utilizar menos recursos de hardware para escrita de dados
- Write-Through
  - Dispositivos com duplo processamento, onde ambos compartilham uma memória comum.

Em Sistemas Multiprocessados

- EM sistemas multiprocessados deve se considerar outras preocupações, como por exemplo:
  - Modelo de compartilhamento de caches a ser utilizado
    - Que níveis de cache são privados (i.e. cada core tem o seu individual) e quais são compartilhados entre todos os cores?
    - Como isso afeta o princípio da localidade?
  - Garantir a coerência de cache quando se tem caches privadas

Coerência de cache

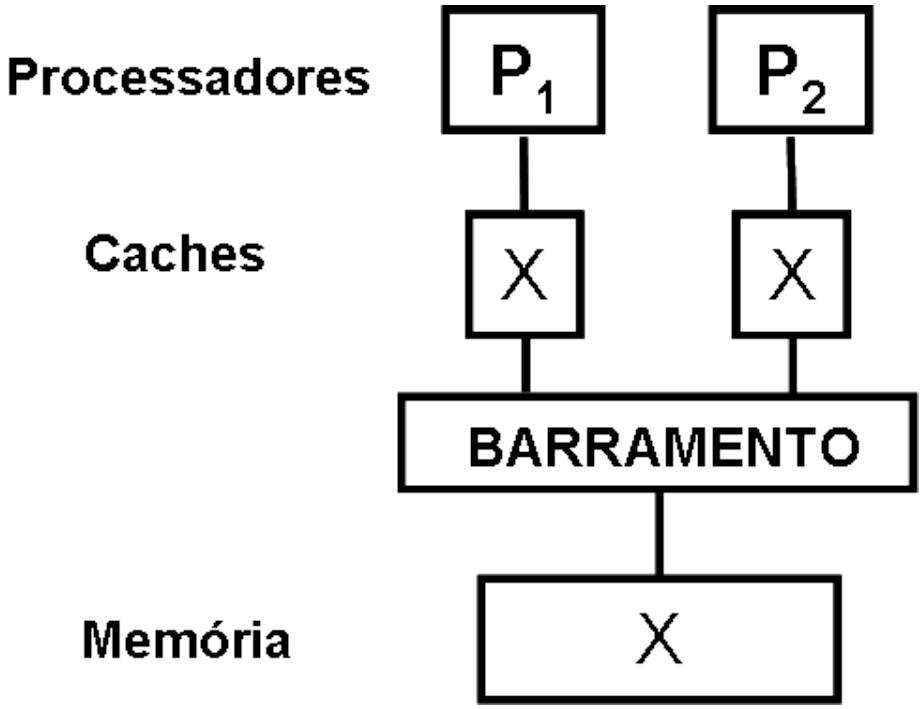
- Definição
  - Considerando um sistema com **mais** de um processador com caches privadas
  - Um dado em uma cache é modificado e uma ou mais caches tem este dado desatualizado
  - A inconsistência também pode ocorrer entre cache e a memória

Coerência de cache

- O problema pode ocorrer por vários fatores.
- Mais comuns:
  - **Uso de dados compartilhados**
  - Migração de processos

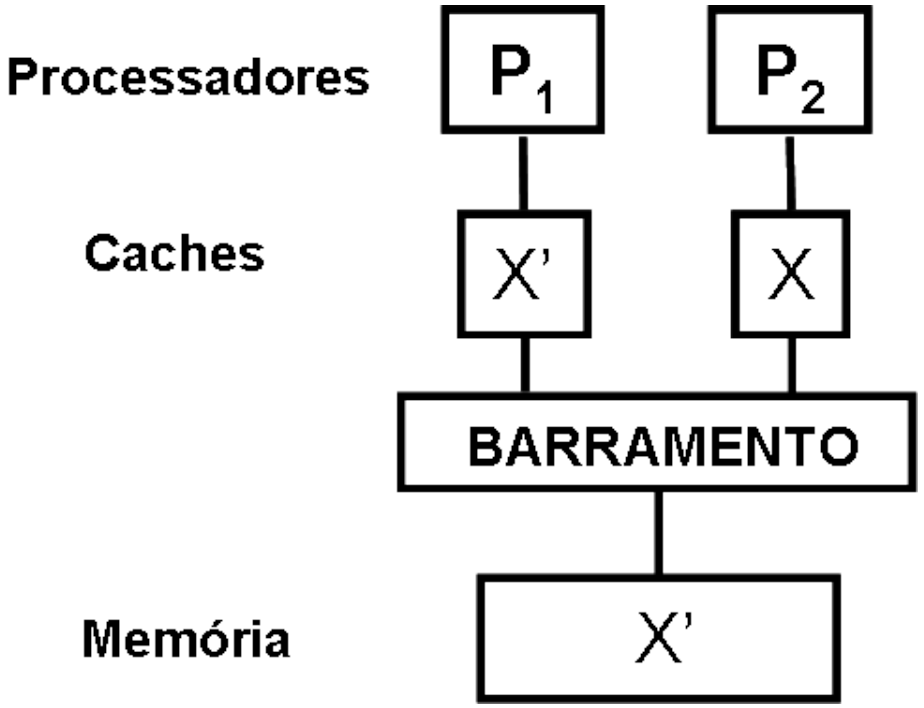
Coerência de cache

- Dados compartilhados (Situação inicial)



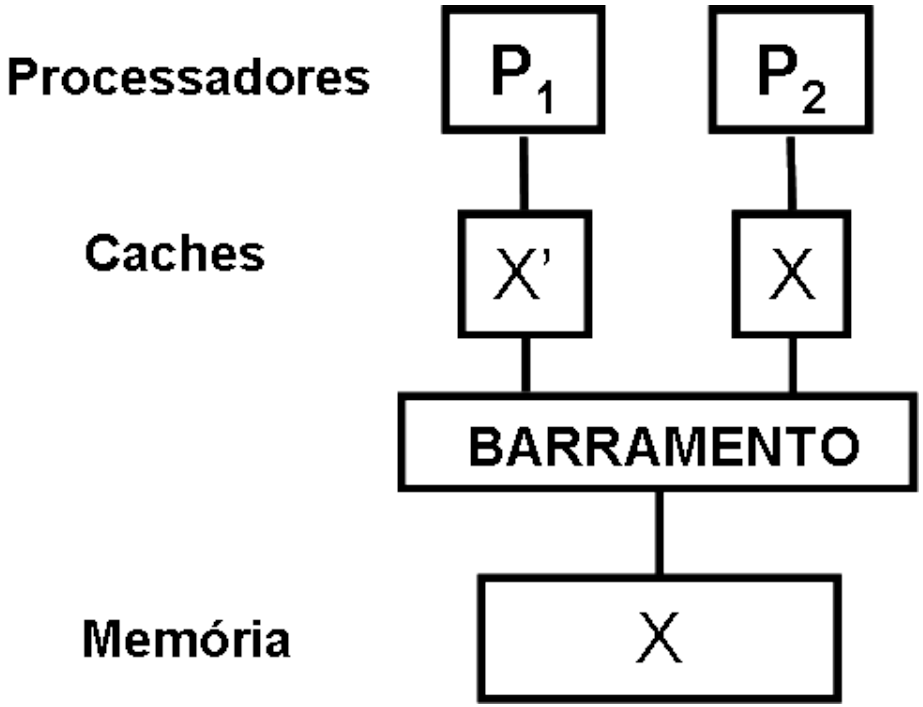
Coerência de cache

- Dados compartilhados (*write-through*)



Coerência de cache

- Dados compartilhados (*write-back*)



Coerência de cache

- Que soluções podem ser utilizadas?
  - Basicamente, as soluções se baseiam em manter um registro do estado da linha cache:
  - Se está exclusivamente na cache privada de um processador
  - Se está na cache privada de mais de um processador (i.e. compartilhada)
  - Se foi modificada e ainda não foi atualizada na memória...
  - Se é somente para leitura
  - Se tem permissão de escrita
- Estas informações precisam ser mantidas e atualizadas para cada cache de cada core

Coerência de cache

- Soluções de hardware
  - As soluções de hardware se dividem em duas categorias: *snoop e diretório*
  - *Snoop*: As informações do estado das linhas da cache são distribuídas
  - Diretório: As informações de estado são concentradas.

Coerência de cache

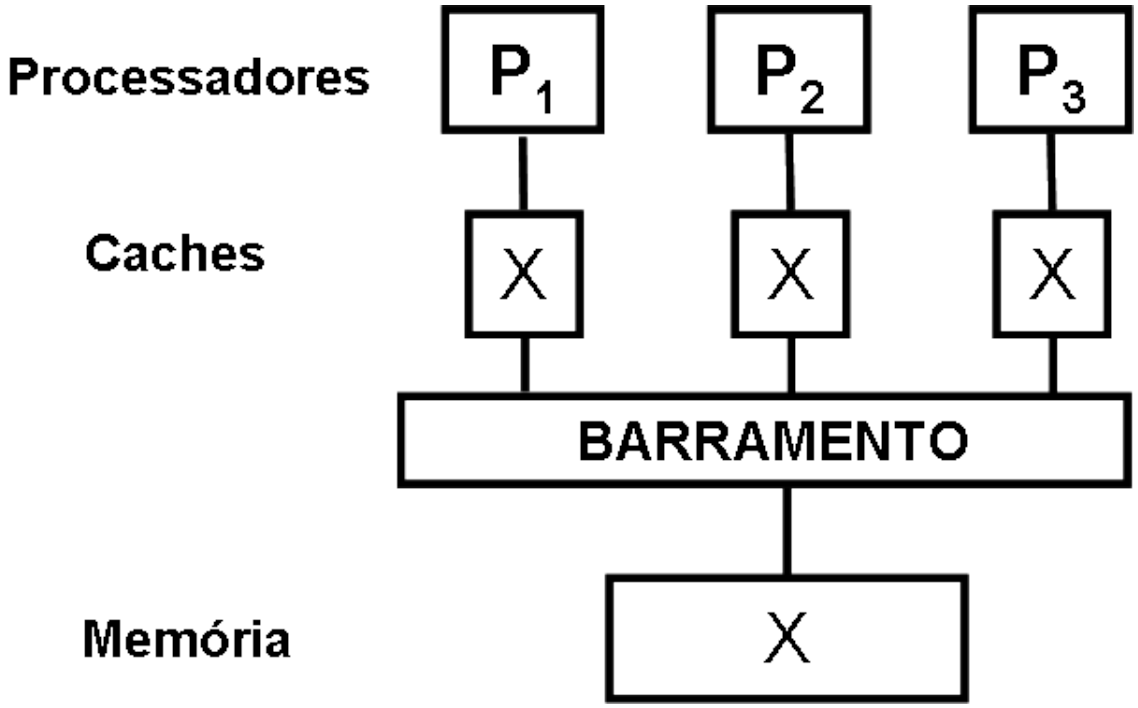
- *Snoop*
  - Solução distribuída
  - Cada cache tem um controlador para manter a coerência dos seus blocos
  - Cada transação é efetuada em *broadcast*

Coerência de cache

- *Snoop*
  - Podem ser utilizadas duas abordagens:
    - *Write-invalidate*
    - *Write-update*

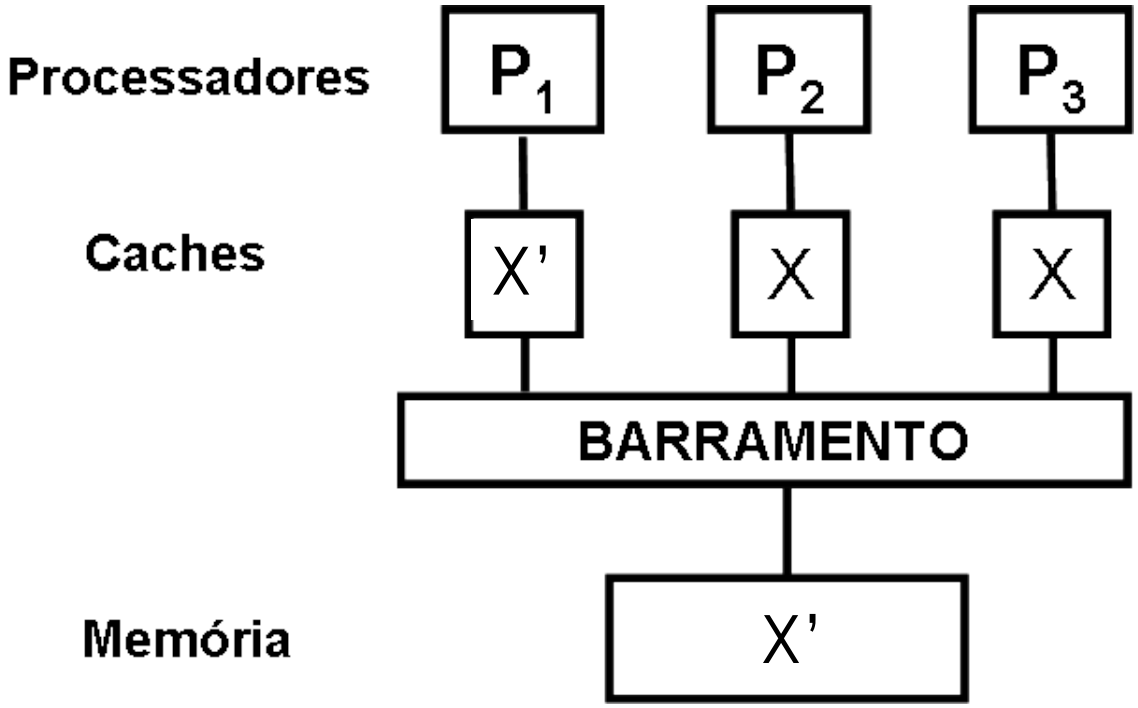
Coerência de cache

- Situação inicial



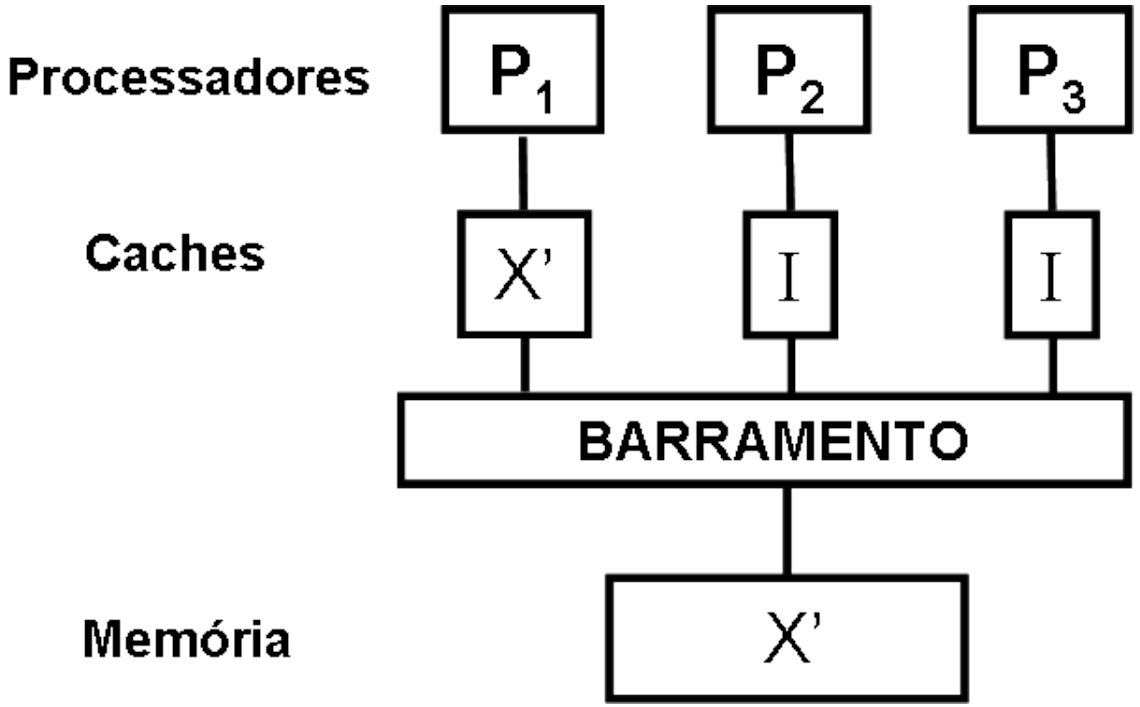
Coerência de cache

- Estado de **incoerência** dos dados!
  - O que fazer?



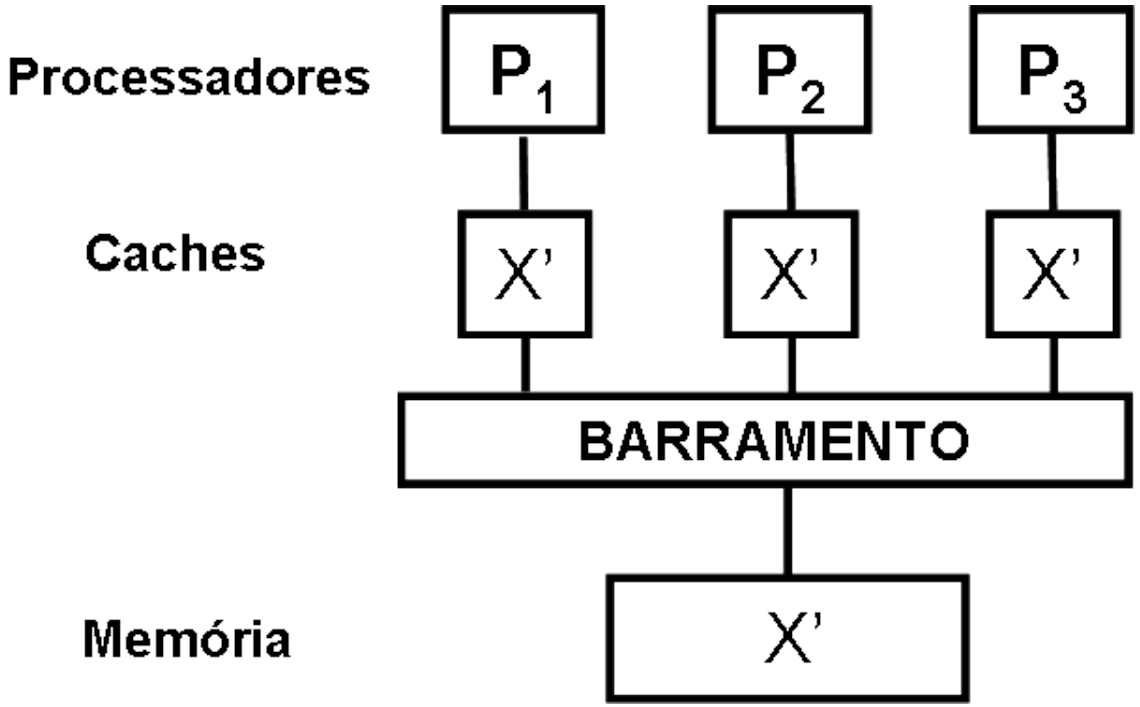
Coerência de cache

- *write-invalidate*
  - Invalida-se as linhas correspondentes nas outras caches



Coerência de cache

- *write-update*
  - Atualiza-se a nova informação nas linhas correspondetes das outras caches



Coerência de cache

- *Snoop*
  - MSI: protocolo criado para definir os estados dos blocos
    - MSI (*Modified*; *Shared*; *Invalid*)
  - A partir do protocolo MSI foram criados outros protocolos:
    - MESI (Modified; Exclusive; Shared; Invalid);
    - MOSI (Modified; Owner; Shared; Invalid);
    - MOESI (Modified; Owner; Exclusive; Shared; Invalid).

Coerência de cache

- Diretório
  - Solução centralizada
  - Geralmente modelado como um componente localizado na memória principal
  - Solução que não exige o uso de operações de *broadcast* (NoC)

Coerência de cache

- Diretório
  - Metodologia:
    - Manter uma "grande" tabela que indica os estados dos blocos e em que cache eles se encontram
      - Blocos podem ser sujos ou limpos:
        - Sujos: foram modificados e encontram-se em incoerência
        - Limpos: não foram modificados

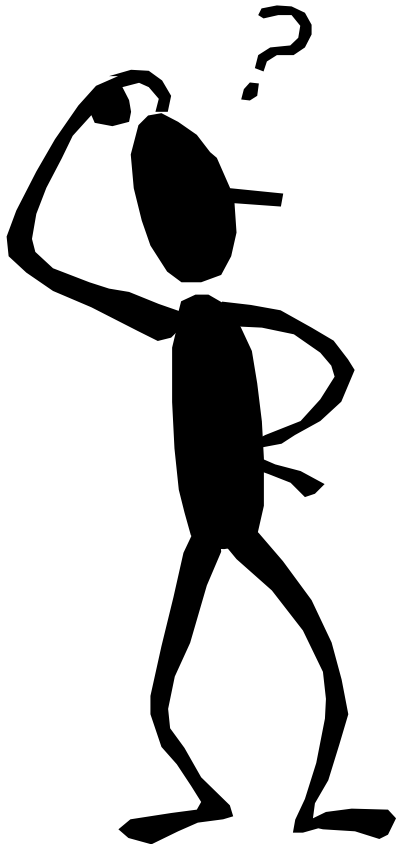
Bloco	P0	P1	P2	Sujo
0	0	1	0	1
1	1	1	1	0
2	1	1	0	0
3	0	0	1	0

Coerência de cache

- Diretório
  - Para manter a coerência, o diretório deve ser o responsável por enviar mensagens para a(s) cache(s) para que elas:
    - Invalide(m) um determinado bloco quando alguma outra cache solicitar uma escrita nele;
    - Envie um bloco que foi modificado por ela (caso a política de escrita seja *write-back*);

Soluções de coerência de cache

- De modo geral as duas técnicas são largamente utilizadas, porém:
  - A técnica de snoop
    - Não escala tão bem quanto a de diretório, i.e. quanto maior o numero de cores, pior o desempenho do sistema.
    - Em geral é utilizada em sistemas menores
  - A técnica de diretório
    - Apesar de escalar melhor, é mais complexa de se implementar.
    - Exige um controlador adicional centralizado na memória que gerencia as informações.
    - Faz uso de uma memória para armazenar sua tabela.
    - É bem mais eficiente ao ser implementada em um sistema que suporte comunicação paralela (i.e.: todos podem se comunicar ao mesmo tempo)



Bibliografia

- PATTERSON, D.A. & HENNESSY, J. L. **Organização e Projeto de Computadores** - A Interface Hardware/Software. 3ª ed. Campus, 2005. **CAPÍTULO 7**

- STALLINGS, William. Arquitetura e organização de computadores. 8. ed. São Paulo: Pearson, 2010. **Capítulo 4**

Próxima aula

- Memória Virtual