Multiprocessadores com Memória Compartilhada

Prof. Rômulo Calado Pantaleão Camara

Multiprocessadores

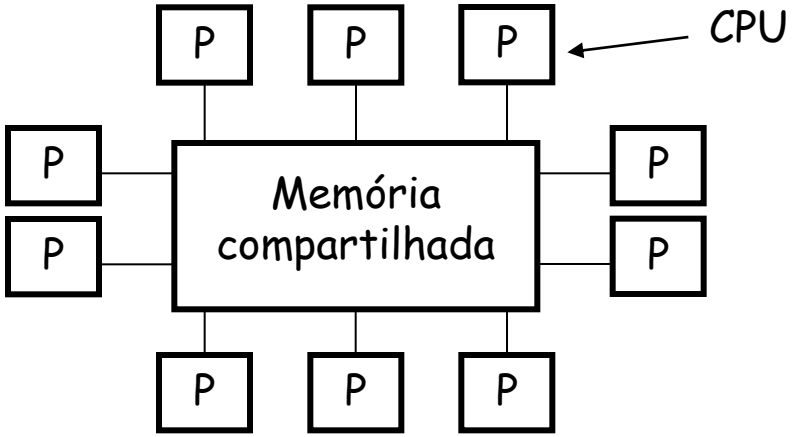
Carga Horária: 2h/60h

Introdução

- ✓ Sistemas com várias CPUs - MIMD.
  - **Multiprocessadores;**
  - Multicomputadores.

Multiprocessadores

- ✓ **Multiprocessador** → Computador paralelo onde todas as CPUs compartilham uma memória comum.



Multiprocessadores

- ✓ **Multiprocessador** → Formado por vários dispositivos de E/S.
- ✓ Em alguns sistemas, apenas algumas CPUs possuem acesso aos dispositivos de E/S → **CPUs com função de E/S especial.**
- ✓ Quando cada CPU tem igual acesso a todos os módulos de memória e a todos os dispositivos de E/S, o sistema é denominado de **multiprocessador simétrico – SMP.**

Multiprocessadores

- ✓ Existem três tipos de multiprocessadores, distinguidos pelo modo como a memória compartilhada é implementada.
  - Multiprocessadores **UMA** (Uniform Memory Access – Acesso à Memória Uniforme).
  - Multiprocessadores **NUMA** (NonUniform Memory Access – Acesso Não-Uniforme à Memória).
  - Multiprocessadores **COMA** (Cache Only Memory Access – Acesso Somente à Memória Cache).

Multiprocessadores

- ✓ Em grandes multiprocessadores, a memória é normalmente dividida em vários módulos.
- ✓ **Multiprocessadores UMA** → Cada CPU tem o mesmo tempo de acesso a todos os módulos de memória.
  - Se for tecnicamente impossível, a velocidade das palavras de memória mais rápidas são reduzidas para o valor das mais lentas – **Uniforme**.
  - Uniformidade torna o desempenho previsível.

Multiprocessadores

- ✓ Multiprocessadores NUMA e COMA a propriedade de uniformidade não é válida.
  - Existem módulos de memória perto de cada CPU - Acessar estes módulos é mais rápido do que os mais distantes.
  - **Desempenho** – O local onde o código e os dados são posicionados é importante.

Multiprocessadores

- ✓ Nos multiprocessadores, existem vários módulos de memória, cada um contendo uma parte da memória física, e as CPUs e memórias são conectadas por uma rede complexa de interconexão.
  - É possível que várias CPUs tentem ler uma palavra de memória ao mesmo tempo em que várias outras CPUs estão tentando escrever a mesma palavra.
  - É possível também que mensagens de requisições sejam ultrapassadas por outras em trânsito e serem entregues em ordens diferentes.

Semântica da Memória

- ✓ **Semântica de memória** → Um contrato entre software e hardware de memória.
  - Se o software concordar em obedecer certas regras, a memória concorda em entregar certos resultados.
  - Quais regras?
  - **Modelos de Consistência.**

Semântica da Memória

- ✓ **Exemplo:** suponha que a CPU 0 escreve o valor 1 em alguma palavra de memória e, um pouco mais tarde, a CPU 1 escreve o valor 2 para a mesma palavra. Agora a CPU 0 lê a palavra e obtém o valor 2.
  - O PC está com defeito?
    - Isso depende do que a memória prometeu (contrato).

Semântica da Memória

- ✓ **Modelos de consistência.**
  - Consistência estrita.
  - Consistência seqüencial.
  - Consistência de processador.
  - Consistência fraca.

Semântica da Memória

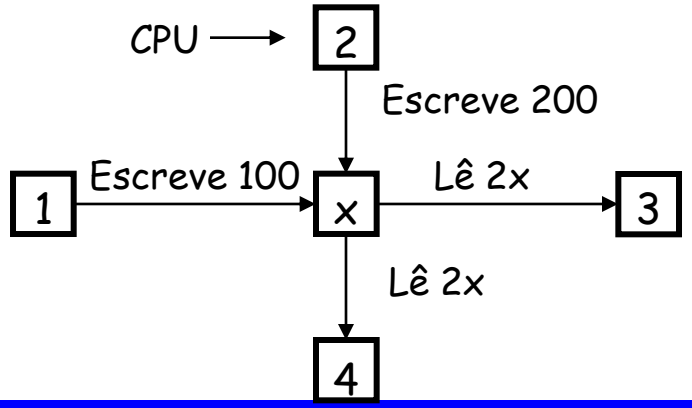
- ✓ **Consistência Estrita.**
  - Neste modelo, qualquer leitura para uma localização x sempre retorna o valor da escrita mais recente de x.
  - Para implementá-la, as requisições deveriam ser atendidas segundo a política primeiro a chegar, primeiro a ser atendido.

Semântica da Memória

- ✓ **Consistência Seqüencial.**
  - Quando houver múltiplas requisições de leitura e escrita, o hardware escolhe (sem determinismo) alguma intercalação de todas as requisições, mas todas as CPUs vêem a mesma ordem.

Semântica da Memória

- ✓ **Consistência Seqüencial.**
  - **Exemplo:** suponha que a CPU 1 escreve o valor 100 para a palavra x, e 1 ns mais tarde a CPU 2 escreve o valor 200 para a palavra x. Agora, suponha que 1 ns após a segunda escrita ter sido emitida (mas não necessariamente ainda concluída) duas outras CPUs, 3 e 4, lêem a palavra x duas vezes cada uma em rápida sucessão.



Semântica da Memória

- ✓ **Consistência Seqüencial.**
  - **Exemplo:** seis eventos ocorreram (duas escritas e quatro leituras).
  - Considere duas possíveis ordenações dos seis eventos:

(a)	(b)
W100	W100
W200	R3=100
R3=200	W=200
R3=200	R4=200
R4=200	R3=200
R4=200	R4=200

Semântica da Memória

- ✓ **Consistência Seqüencial.**
  - Uma memória seqüencialmente consistente nunca permitirá que a CPU 3 obtenha (100, 200) enquanto a CPU 4 obtém (200, 100).
  - Se isto ocorresse, de acordo com a CPU 3, a escrita de 100 pela CPU 1 concluiu antes da escrita de 200 pela CPU 2. Mas, de acordo com a CPU 4, a escrita de 200 pela CPU 2 concluiu antes da escrita de 100 pela CPU 1.
    - A consistência seqüencial garante que há uma única ordenação global de todas as escritas que é visível para as CPUs. Se a CPU 3 observar que 100 foi escrito em primeiro lugar, então a CPU 4 também deve ver essa ordem.

Semântica da Memória

- ✓ **Consistência de Processador.**
  - Possui duas propriedades:
    1. Escritas por qualquer CPU são vistas por todas as CPUs na ordem em que foram emitidas.
       - Se a CPU 1 emitir escritas com valores 1A, 1B e 1C para alguma localização de memória nessa seqüencia. Todas as outras CPUs também as vêem nessa mesma ordem.
    2. Para cada palavra de memória, todas as CPUs vêem todas as escritas para ela na mesma ordem.
       - Garante que toda palavra de memória possui um valor não ambíguo após várias CPUs escreverem para ela e, por fim, pararem.

Semântica da Memória

- ✓ **Consistência de Processador.**
  - **Exemplo:** considere que a CPU 1 emite as escritas 1A, 1B e 1C concorrentemente com a CPU 2, que emite as escritas 2A, 2B e 2C.
  - Outras CPUs que estão ocupadas lendo memória, observarão alguma intercalação de seis escritas tal como, 1A, 1B, 2A, 2B, 1C, 2C ou 2A, 1A, 2B, 2C, 1B, 1C ou outras.
  - A consistência de processador não garante que toda CPU vê a mesma ordenação. Mas a ordem em que cada CPU escreveu é garantida (1A, 1B, 1C...)

Semântica da Memória

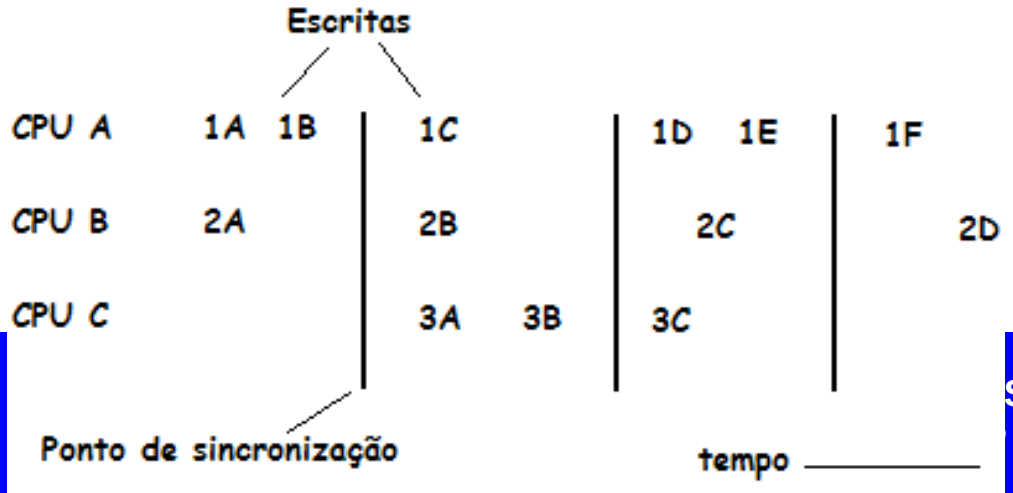
- ✓ **Consistência Fraca.**
  - Não garante que as escritas de uma única CPU sejam vistas em ordem.
  - Em uma memória fracamente consistente, uma CPU poderia ver 1A antes de 1B e uma outra CPU poderia ver 1A depois de 1B.
  - Para realizar uma ordenação, variáveis de sincronização são utilizadas.
    - Quando uma sincronização é executada, todas as escritas pendentes são terminadas e nenhuma nova é iniciada até que todas as antigas estejam concluídas, assim como a própria sincronização.

Semântica da Memória

- ✓ **Consistência Fraca.**
  - O que a sincronização faz?
    - "Descarrega o pipeline" e leva a memória a um estado estável sem nenhuma operação pendente.
  - Operações de sincronização são seqüencialmente consistentes. Quando várias CPUs as emitem, alguma ordem é escolhida, mas todas as CPUs vêem a mesma ordem.

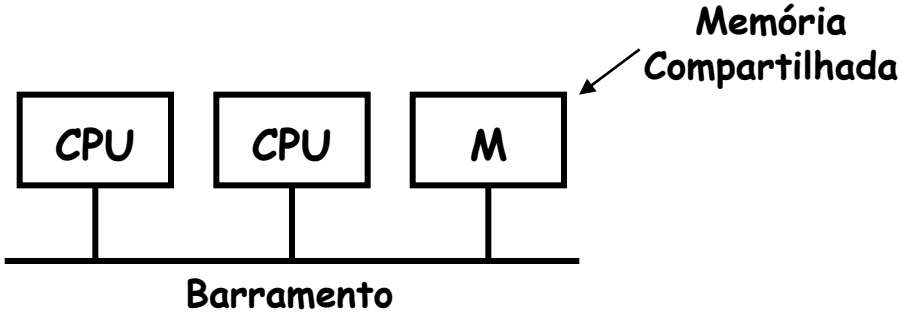
Semântica da Memória

- ✓ **Consistência Fraca.**
  - **Funcionamento** → O tempo é dividido em slots bem definidos delimitados pelas sincronizações.
  - De acordo com a figura, nenhuma ordem relativa é garantida para 1A e 1B, ou seja, diferentes CPUs podem ver as duas escritas em ordens diferentes, mas todas as CPUs vêem 1B antes de 1C, devido a sincronização.



Arquiteturas Simétricas de Multiprocessador UMA

- ✓ Os multiprocessadores mais simples são baseados em um único barramento.



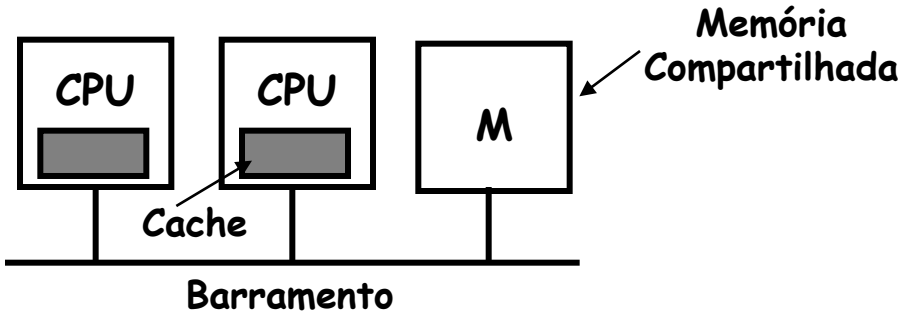
- ✓ Independente da quantidade de CPUs e memórias, todos os componentes utilizam o mesmo barramento.

Arquiteturas Simétricas de Multiprocessador UMA

- ✓ Considere que uma CPU deseja ler uma palavra de memória.
  - Verifica o estado do barramento.
    - **Livre** → CPU coloca o endereço da palavra, ativa alguns sinais de controle e espera o resultado.
    - **Ocupado** → CPU espera barramento ficar livre.
- ✓ **Problema** → Se houver muitas CPUs (mais de 32), a espera devido a contenção do barramento será insuportável.
  - O sistema ficará limitado pela largura de banda do barramento.
  - A maioria das CPUs ficará ociosa a maior parte do tempo.

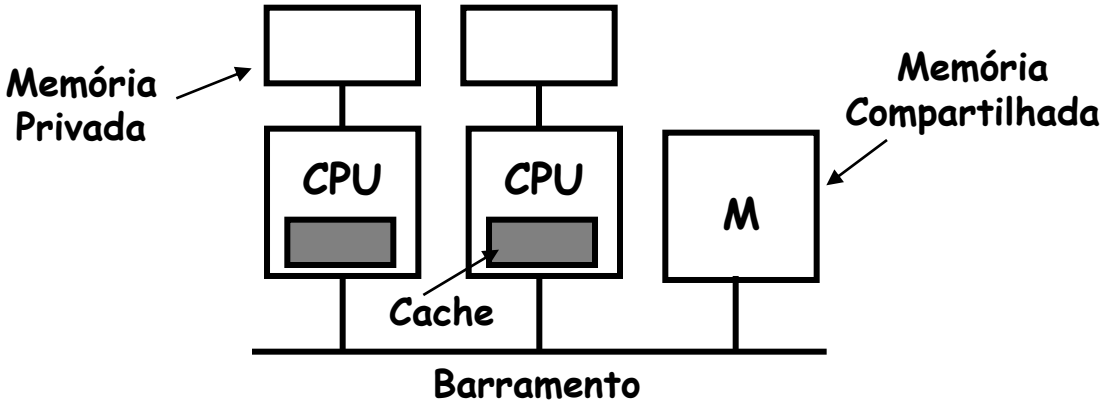
Arquiteturas Simétricas de Multiprocessador UMA

- ✓ **Soluções.**
  - Acrescentar uma cache a cada CPU.
    - Diminuição no tráfego do barramento → Muitas leituras podem ser satisfeitas pela cache local.
    - O sistema pode ter mais CPUs.



Arquiteturas Simétricas de Multiprocessador UMA

- ✓ **Soluções.**
  - Além da cache, adicionar uma memória local e privada a cada CPU.



  - Para otimizar o desempenho, o compilador deve colocar os programas e dados na memória local.
  - Memória compartilhada → Variáveis compartilhadas.

Arquiteturas Simétricas de Multiprocessador UMA

- ✓ **Mais problemas...**
  - Suponha que a memória seja seqüencialmente consistente.
    - O que acontece se a CPU 1 tiver uma linha em sua cache e então a CPU 2 tentar ler uma palavra na mesma linha de cache?
      - A CPU 2 obterá uma cópia na sua cache.
    - Suponha agora que a CPU 1 modifica a linha e, em seguida, a CPU 2 lê a sua cópia da linha a partir de sua cache.
      - Dados estarão desatualizados (**dados velhos**) → **Coerência de cache** ou **consistência de cache**.

Arquiteturas Simétricas de Multiprocessador UMA

- ✓ O problema da coerência de cache é importante.
  - Conviver com esse problema torna o sistema defeituoso.
  - Sem cache, os problemas da largura de banda limitada e das CPUs ociosas continuam, Assim, não podemos ter muitas CPUs nesses sistemas.
- ✓ Muitas soluções foram propostas – **Protocolos de coerência de cache**.
  - **Idéia básica** → Impedir que versões diferentes da mesma linha de cache apareçam simultaneamente em duas ou mais caches.

Arquiteturas Simétricas de Multiprocessador UMA

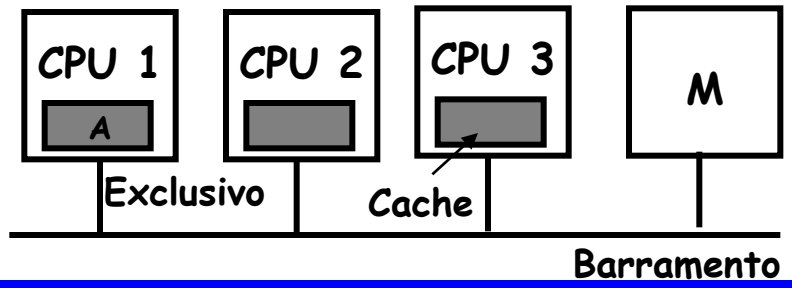
- ✓ Em todas as soluções, existe um controlador de cache (**caches de escuta**) projetado para monitorar todas as requisições do barramento de outras CPUs e caches, e executar alguma ação em certos casos.
- ✓ O conjunto de regras implementado pelas caches, CPUs e memória formam o **protocolo de coerência de cache**.

Arquiteturas Simétricas de Multiprocessador UMA

- ✓ **Protocolo MESI de Coerência de Cache.**
  - Utilizado para monitorar o barramento no Pentium 4.
  - Cada entrada de cache pode estar em um dos seguintes quatro estados:
    - **Inválido** – A entrada da cache não contém dados válidos.
    - **Compartilhado** – Múltiplas caches podem conter a linha. A memória está atualizada.
    - **Exclusivo** – Nenhuma outra cache contém a linha. A memória está atualizada.
    - **Modificado** – A entrada é válida. A memória é inválida. Não existem cópias.

Arquiteturas Simétricas de Multiprocessador UMA

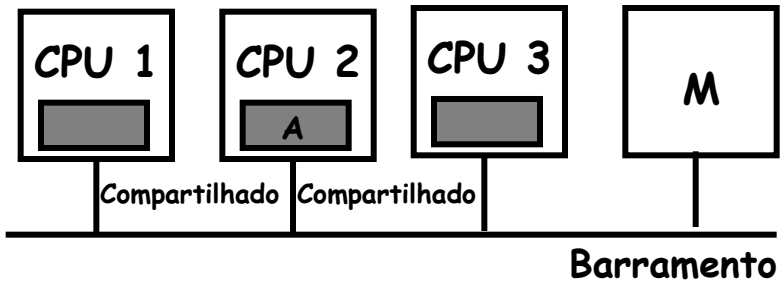
- ✓ **Protocolo MESI de Coerência de Cache.**
  - **Funcionamento.**
    - CPU é iniciada pela 1ª → Todas as entradas de cache são marcadas como **INVÁLIDAS** (Não contém dados válidos).
    - Memória lida pela 1ª vez → A linha referenciada **(A)** é marcada como **EXCLUSIVO** (Memória atualizada), uma vez que ela é a única cópia dentro de uma cache (Veja CPU 1).



CPU 1 lê bloco A.

Arquiteturas Simétricas de Multiprocessador UMA

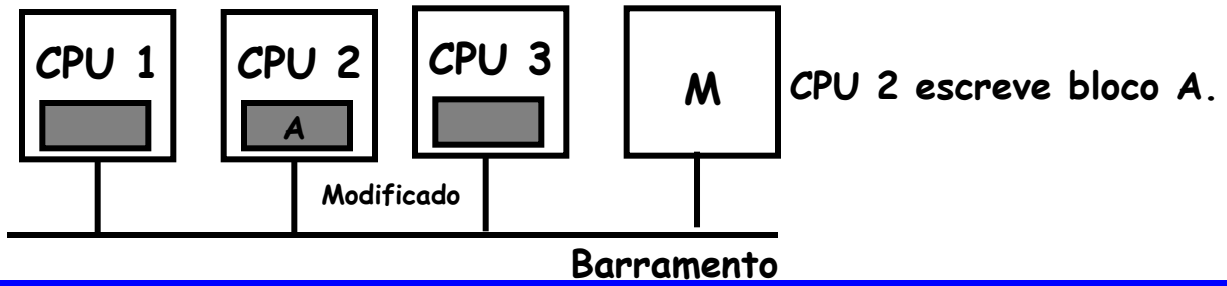
- ✓ **Protocolo MESI de Coerência de Cache.**
  - **Funcionamento.**
    - Uma outra CPU também pode buscar a mesma linha **(A)** e colocá-la na cache mas, por causa da escuta, o portador original (CPU 1) vê que não está mais sozinho e anuncia no barramento que ele também tem uma cópia. Ambas as cópias são marcadas como **COMPARTILHADO** (Memória atualizada).



CPU 2 lê bloco A.

Arquiteturas Simétricas de Multiprocessador UMA

- ✓ **Protocolo MESI de Coerência de Cache.**
  - **Funcionamento.**
    - Se a CPU 2 escrever na linha de cache **(A)** 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 nova cópia em cache passa para o estado **MODIFICADO** (Entrada é válida e a memória é inválida).



Arquiteturas Simétricas de Multiprocessador UMA

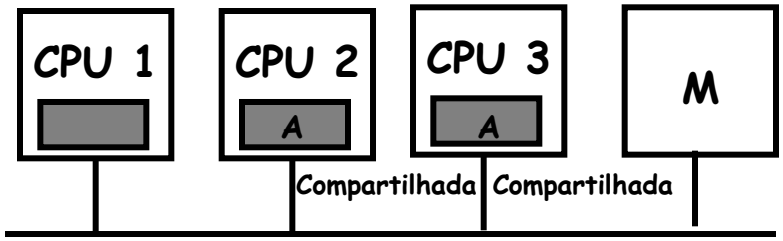
- ✓ **Protocolo MESI de Coerência de Cache.**
  - **Funcionamento.**
    - **Obs:** Se uma linha estiver no estado **EXCLUSIVO** quando for escrita, nenhum sinal é necessário para invalidar outras caches.

Arquiteturas Simétricas de Multiprocessador UMA

- ✓ **Protocolo MESI de Coerência de Cache.**
  - **Funcionamento.**
    - O que ocorre se a CPU 3 ler a linha **(A)** que agora pertence a CPU 2 e está no estado **MODIFICADO**?
      - A CPU 2 envia um sinal no barramento informando à CPU 3 que a linha na memória é inválida, e pede que aguarde até que ela escreva a linha de volta na memória.
        - » **Obs:** a linha é válida apenas para a cache da CPU 2. Quando a CPU 2 atualiza à memória, o novo estado da linha será **EXCLUSIVO** mas, imediatamente, a CPU 3 lê a memória, e a linha vai para o estado **COMPARTILHADO**.

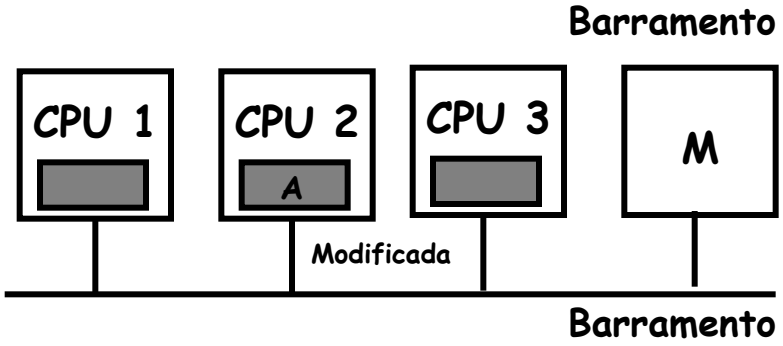
Arquiteturas Simétricas de Multiprocessador UMA

- ✓ **Protocolo MESI de Coerência de Cache.**
  - **Funcionamento.**



CPU 3 lê bloco A.

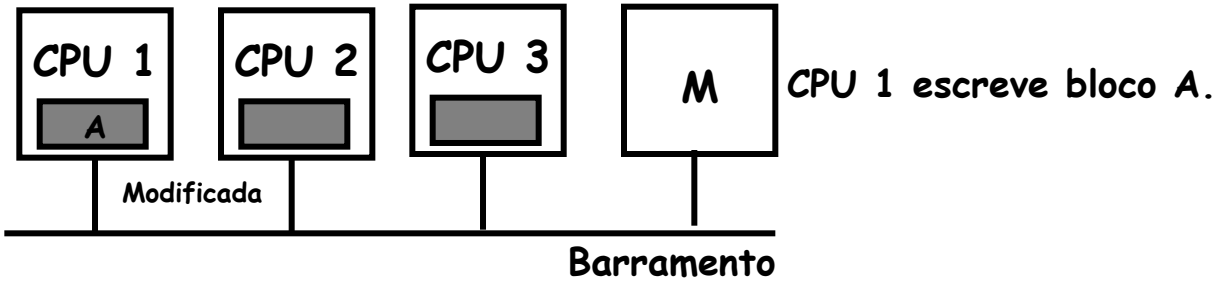
- A CPU 2 escreve na linha novamente, e inválida a cópia da CPU 3. A memória torna-se inválida.



CPU 2 escreve bloco A.

Arquiteturas Simétricas de Multiprocessador UMA

- ✓ **Protocolo MESI de Coerência de Cache.**
  - **Funcionamento.**
    - Se a CPU 1 escrever uma palavra na linha **A** → A CPU 2 vê a tentativa de escrita e emite um sinal via o barramento dizendo a CPU 1 para esperar até que ele atualize a memória. Após atualizar, a CPU 2 marca sua própria cópia como inválida.



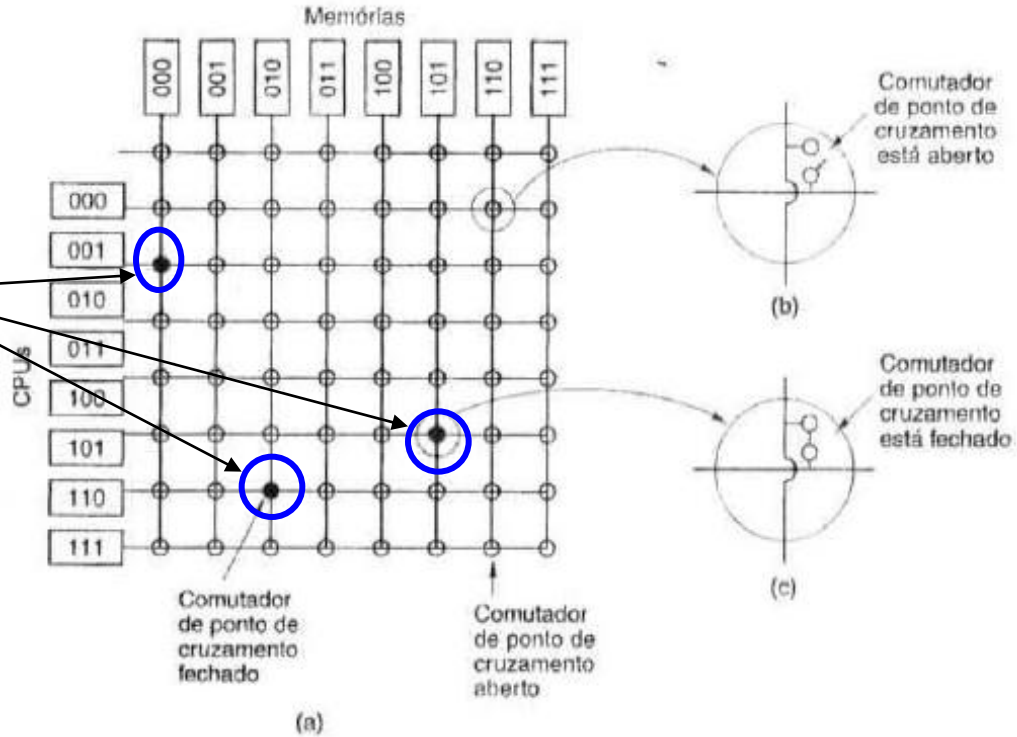
Multiprocessador UMA - Comutadores Crossbar

- ✓ Apesar de todas as otimizações, a utilização de um único barramento limita a quantidade de CPUs em um multiprocessador UMA há cerca de 16 à 32 CPUs.
- ✓ Para aumentar este número, é preciso utilizar um tipo diferente de rede de interconexão.
- ✓ O circuito mais simples para conectar n CPUs a k memórias é o **comutador crossbar**.

Multiprocessador UMA - Comutadores Crossbar

- ✓ Em cada interseção de uma linha horizontal com uma vertical está um **ponto de cruzamento**.
- ✓ Um **ponto de cruzamento** é um pequeno comutador que pode ser aberto ou fechado eletricamente, dependendo da conexão entre as linhas horizontal e vertical.

Pontos de cruzamento fechados ao mesmo tempo



Multiprocessador UMA - Comutadores Crossbar

- ✓ **Comutador crossbar** → É uma rede sem bloqueio.
  - A conexão necessária a uma CPU nunca será negada caso algum ponto de cruzamento ou linha já esteja ocupada.
    - É necessário apenas que o módulo de memória desejado esteja disponível.
- ✓ **Desvantagens**.
  - O número de pontos de cruzamento cresce n^2.
    - Com mil CPUs e mil módulos de memórias, precisamos de 1 000 000 de pontos de cruzamentos, o que não é viável financeiramente.

Multiprocessadores NUMA

- ✓ Até aqui, vimos que multiprocessadores UMA de um único barramento são limitados a não mais que algumas dezenas de CPUs e que multiprocessadores crossbar são caros.
- ✓ Para obter mais de cem CPUs, é necessário abandonar a idéia de que todos os módulos de memória possuem o mesmo tempo de acesso – **Multiprocessadores NUMA**.

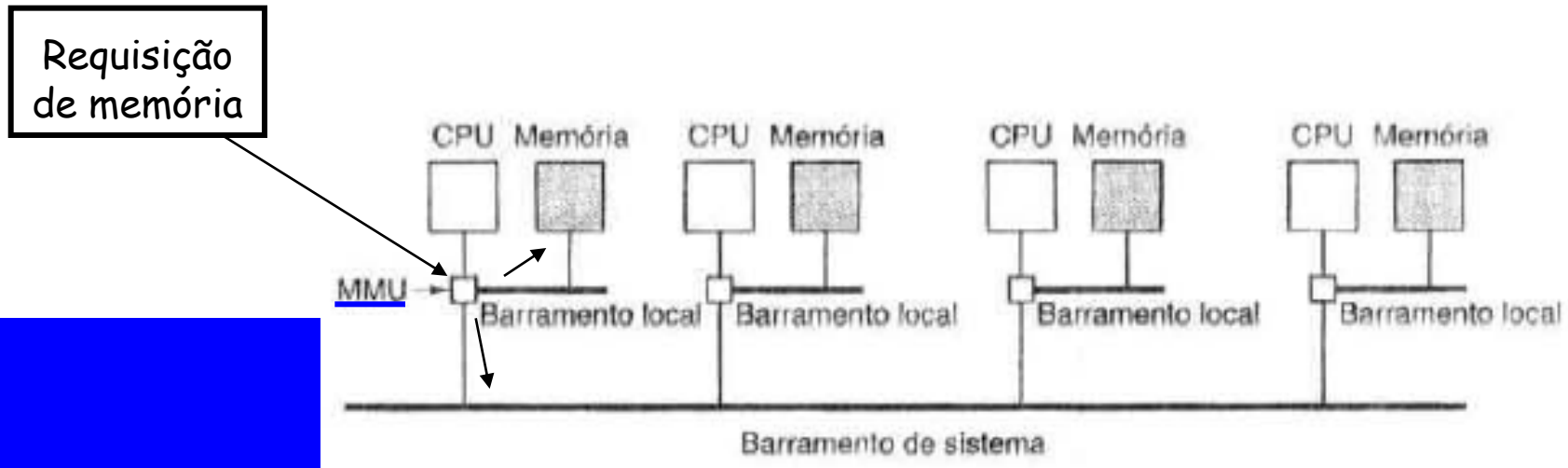
Multiprocessadores NUMA

- ✓ Características dos multiprocessadores NUMA.
  - Assim como os UMA, o espaço de endereço é único para todas as CPUs.
  - Acesso à memória remota é feita usando instruções LOAD e STORE.
  - Acesso à memória remota é mais lento do que o acesso à memória local.

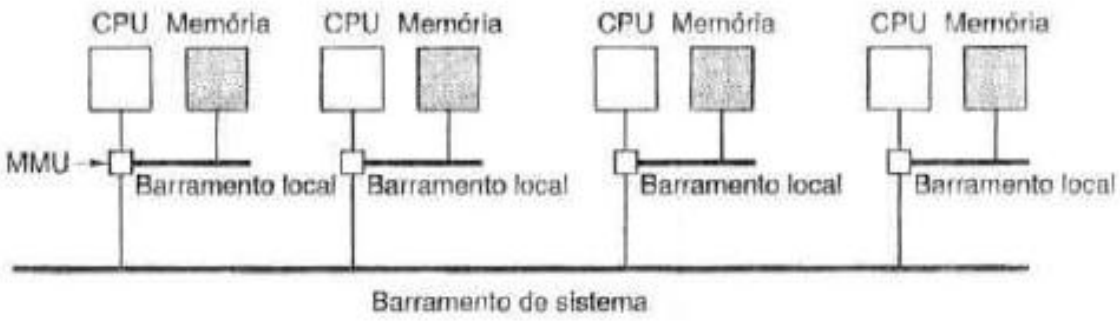
.

Multiprocessadores NUMA

- ✓ Tipos de multiprocessadores NUMA.
  - Quando o tempo de acesso à memória remota não é oculto, porque não há cache, o sistema é chamado de **NC-NUMA**.
  - Quando existem caches coerentes, o sistema é chamado de ***CC-NUMA***.
    - Primeira máquina NC-NUMA, chamada de Cm*.



Multiprocessadores NUMA com Coerência de Cache



Máquina NC-NUMA com dois níveis de barramento.

- ✓ Devido a ausência de cache, a ampliação resulta em problemas de desempenho.
  - **Problema** → Acessar a memória remota toda vez que não estiver na memória local.
  - **Solução** → Adicionar cache → Coerência de cache → CC-NUMA.

Multiprocessadores NUMA com Coerência de Cache

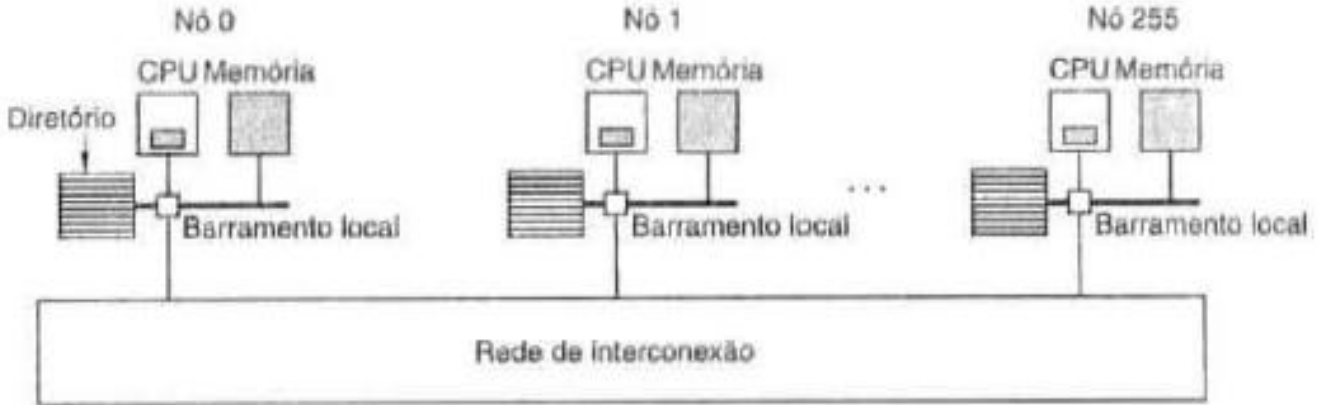
- ✓ Vimos nos multiprocessadores UMA que um modo de proporcionar coerência de cache é "escutar" o barramento do sistema – **Controlador de cache (Cache de escuta)**.
  - **Problema** → A utilização dos controladores torna-se inviável se o número de processadores crescer muito.
  - **Solução** → Multiprocessador baseado em diretório.
    - **Idéia básica** → Manter um banco de dados que informa onde está cada linha de cache e em qual estado ela está.

Multiprocessadores NUMA com Coerência de Cache

- ✓ **Multiprocessador baseado em diretório.**
  - Quando uma linha de cache for referenciada, o banco de dados é pesquisado para descobrir onde ela está e se está limpa ou suja (modificada).
  - O banco de dados deve ser mantido em hardware muito rápido capaz de responder em uma fração de um ciclo de barramento.

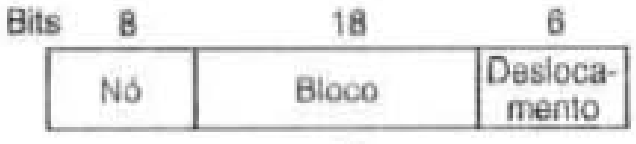
Multiprocessadores NUMA com Coerência de Cache

- ✓ **Exemplo.**
  - Considere um sistema com 256 nós, cada nó consistindo em uma CPU e 16 MB de RAM conectados à CPU por um barramento local.
  - A memória total é 2^{32} bytes, dividida em 2^{26} linhas de cache de 64 bytes. A memória é alocada estaticamente entre os nós, com 0-16MB no nó 0, 16-32M no nó 1 e assim por diante.



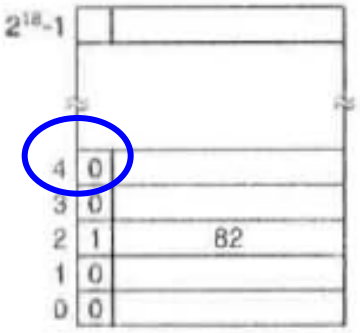
Multiprocessadores NUMA com Coerência de Cache

- ✓ **Exemplo.**
  - Cada nó também contém as entradas de diretório para as 2^{18} linhas de cache de 64 bytes abrangendo sua memória de 2^{24} bytes.
  - Por enquanto vamos considerar que uma linha pode ser contida, no máximo, em uma cache.
  - Suponha uma instrução LOAD da CPU 20 que referencia uma linha que está na cache.
  - A CPU 20 emite a instrução para a MMU, que a traduz para um endereço físico, por exemplo, 0x24000108. A MMU subdivide esse endereço em três partes – Neste caso, são nó 36, bloco 4 e deslocamento 8.



Multiprocessadores NUMA com Coerência de Cache

- ✓ **Exemplo.**
  - A MMU vê que a palavra de memória referenciada é do nó 36, e não do nó 20, portanto envia uma mensagem de requisição pela rede de interconexão ao nó 36 perguntando se sua linha 4 está em cache e, se estiver, onde está.
  - Quando a requisição chega ao nó 36, ela é roteada para o hardware de diretório. O hardware indexa para sua tabela de 2^{18} entradas, uma para cada uma das linhas de cache e extrai a entrada 4.



Multiprocessadores NUMA com Coerência de Cache

- ✓ **Exemplo.**
  - Como a linha 4 não está em cache, o hardware busca a linha 4 na RAM local, a envia de volta ao nó 20 e atualiza a entrada do diretório 4 para indicar que a linha agora está em cache no nó 20.
  - Considere agora uma segunda requisição, desta vez perguntando sobre a linha 2 do nó 36. De acordo com a Figura anterior, vemos que essa linha está no nó 82. Então, o hardware poderia atualizar a entrada de diretório 2 para informar que a linha está agora no nó 20 e então enviar uma mensagem ao nó 82 instruindo-o a passar a linha 20 para o nó 20 e invalidar sua cache.

Multiprocessadores NUMA com Coerência de Cache

- ✓ Uma limitação desse projeto é que uma linha só pode ser colocada em cache em um único nó.
  - Para permitir cache de linhas em vários nós, precisaríamos de algum modo de localizar todas elas, por exemplo, para invalidá-las ou atualizá-las em uma escrita.
  - **Solução 1** → Cada entrada de diretório ter k campos para especificar outros nós, permitindo assim a cache de cada linha em até k nós.
  - Existem diversas soluções.

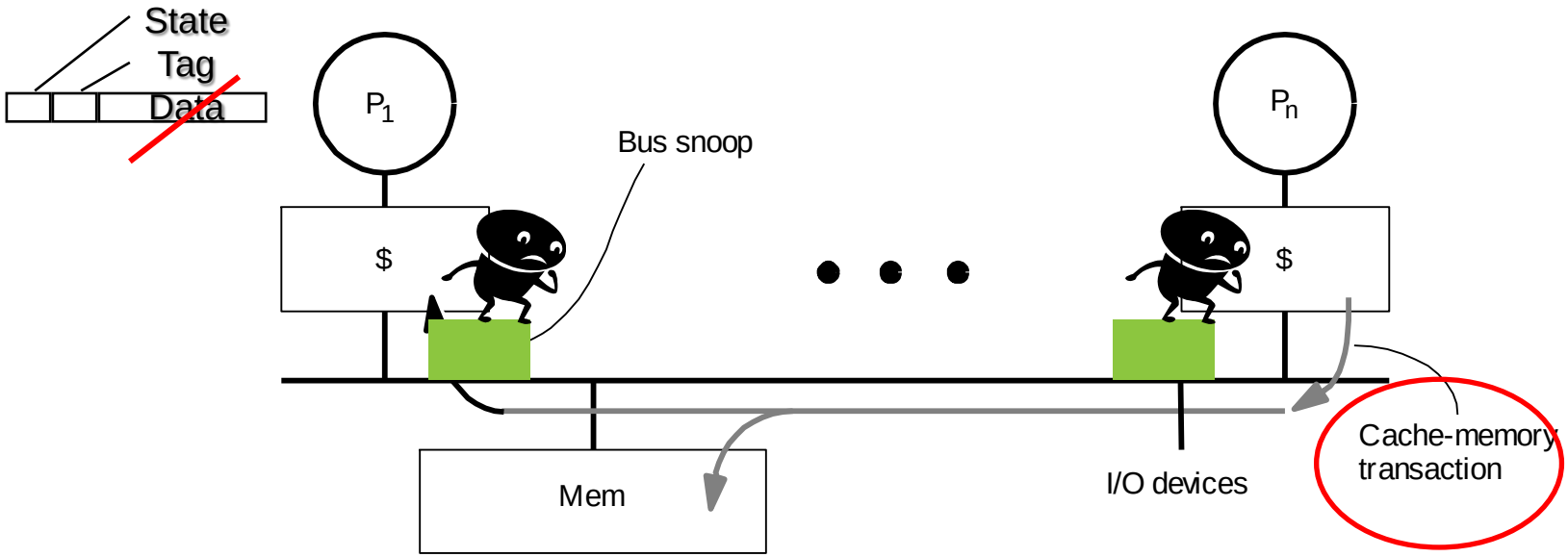
Coerência vs. Consistência

- ✓ **Coerência:** comportamento de leituras e escritas no mesmo endereço.
- ✓ QUAL?

- ✓ **Consistência:** comportamento de leituras e escritas em endereços diferentes de memória.
- ✓ Quando?

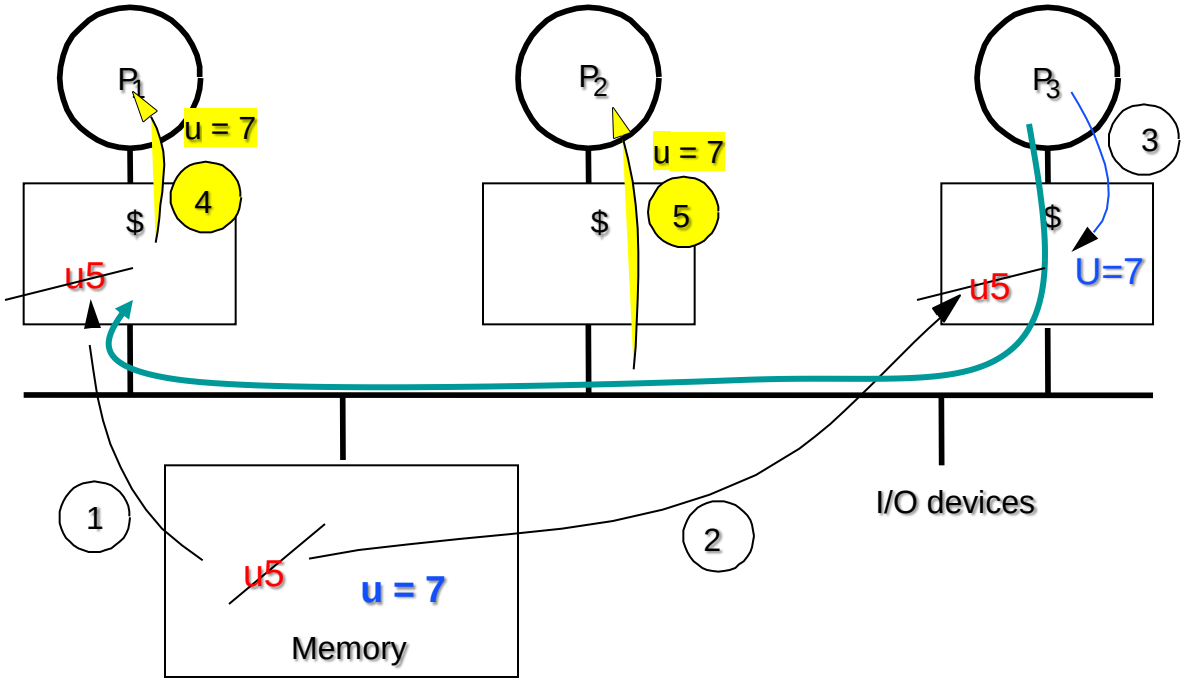
Protocolo Snooping

Protocolo Snooping de Coerência de Cache



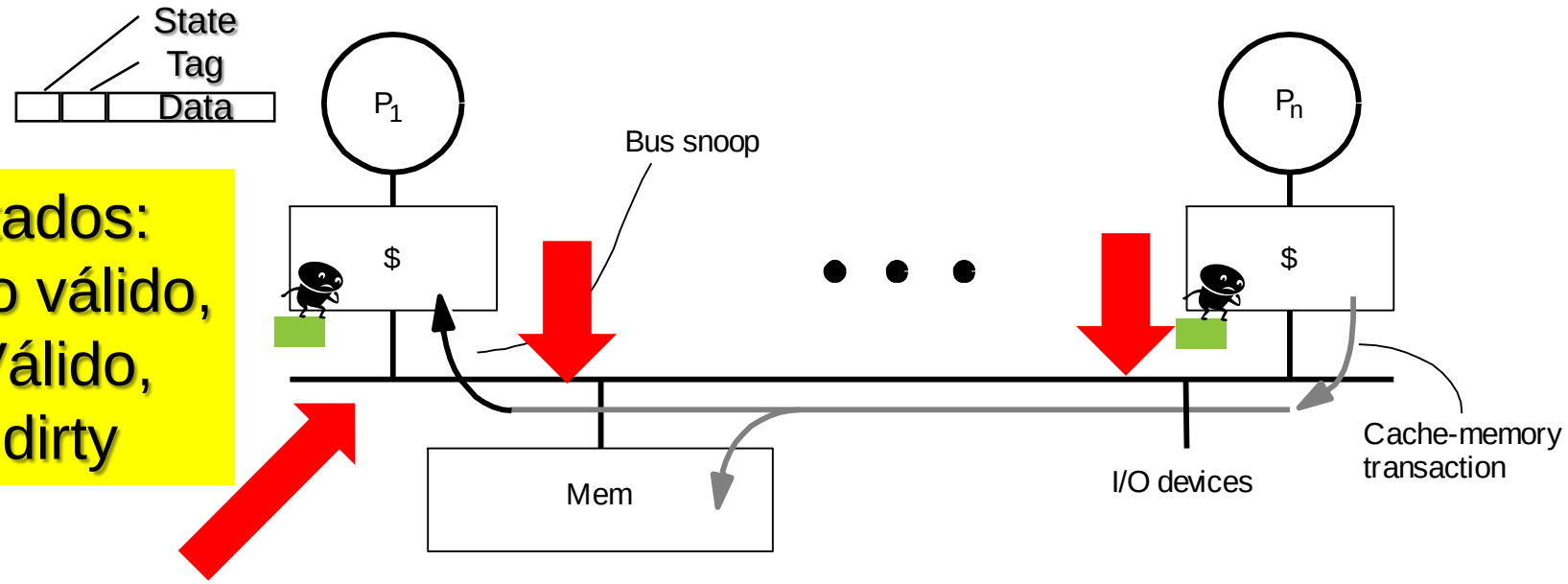
- Controladores de Cache "snoops" todas as transações no barramento
  - Transações relevantes : envolvem bloco que a cache possui
  - Realiza ação para garantir coerência
    - invalida, atualiza, ou fornece valor
  - Atualiza estado de compartilhamento do bloco de cache

Exemplo: Write-thru Invalidate



- ✓ P3 Deve invalidar todas as cópias antes da escrita
- ✓ Caches write through: simplicidade de implementação porém mais acessos à memória
- ✓ Caches write-back: menos acessos à memória, mas como encontrar o bloco que contem a cópia com valor atualizado?

Módulos da Arquitetura



Estados:
- Não válido,
- Válido,
- dirty

✓ Protocolo de barramento
- Requisição
- Comando
- Dado

- **Acesso simultâneo:**
  - Só um processador ganha o acesso
  - Decisão: árbitro
  - Invalidação das demais cópias

- Onde encontrar a cópia mais atualizada do bloco de cache?

Localizando cópia mais atualizada

- ✓ Caches Write-through: usa cópia da memória
  - Write through é mais simples porém causa muitos acessos à memória e maior utilização do barramento.
- ✓ Caches Write-back: deve localizar cópia mais recente nas caches.
  - É mais complicado de implementar
  - Reduz acessos à memória
  - A maioria dos multiprocessadores usam caches write-back

Localizando cópias em Caches Write Back

Solução: Usar o mesmo mecanismo de snooping para achar a cópia mais atual

- Blocos de cache Snoop todo endereço colocado no barramento
- Se processador possui cópia atual do bloco requisitado ele responde a requisição de leitura e aborta acesso à memória.

Recursos da Cache para WB Snooping

- ✓ Tags da cache podem ser usados para snooping
  - Bit de Validade do bloco facilita a invalidação
  - Faltas de Leitura usam snooping
- ✓ Writes ⇒ Necessita saber se existem outras cópias do em outras caches
  - Não existem cópias ⇒ Não necessita colocar invalidate no barramento
  - Se existem cópias ⇒ Necessita colocar invalidate no barramento

Recursos da Cache para WB Snooping

- ✓ Para sinalizar que bloco é compartilhado usa-se bit extra para cada bloco (exclusivo ou compartilhado):
  - Escrita em Bloco Compartilhado ⇒ Colocar invalidate no barramento e marcar bloco como exclusivo
  - O processador que escreve será o owner do bloco de cache.
  - Bloco de cache no owner muda do estado shared para exclusive

Exemplo de Protocolo

- ✓ Protocolo de Coerência de Snooping é usualmente implementado por um controlador especial para cada cache
- ✓ O controlador permite operações em blocos distintos
  - Uma operação pode ser iniciada antes que outra tenha sido concluída, mesmo porque é permitido apenas um acesso à cache ou barramento por vez.

Protocolo Snooping Write Back

- ✓ Cada bloco de cache vai estar em UM dos estados:
  - Shared : bloco pode ser lido
  - OU Modified/Exclusive : cache tem somente uma cópia que pode ser escrita e dirty
  - OU Invalid : bloco não contem dado válido

Protocolo Snooping Write Back

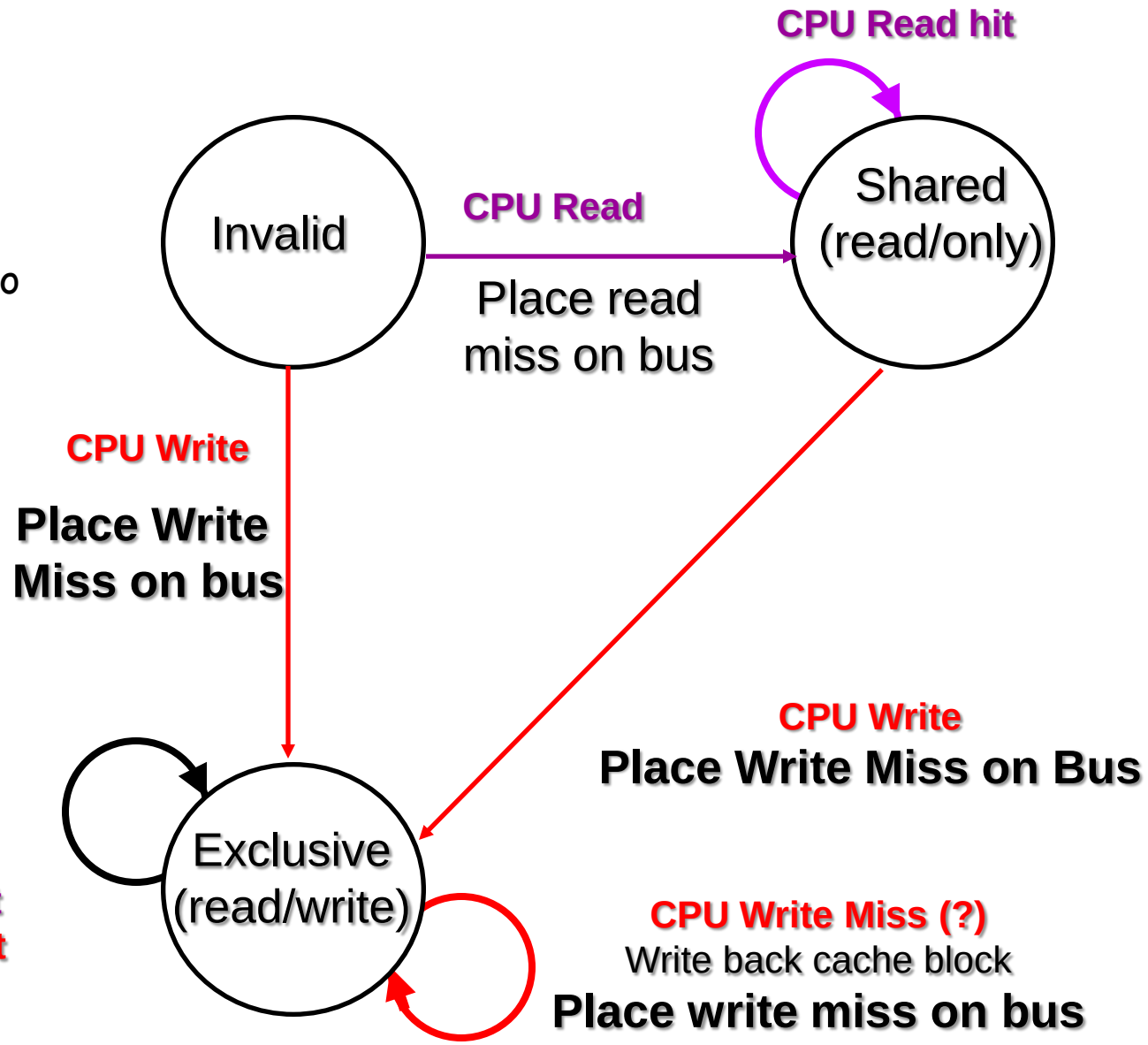
- ✓ Cada bloco de cache vai estar em UM dos estados:
  - Shared : bloco pode ser lido
  - OU Modified/Exclusive : cache tem somente uma cópia que pode ser escrita e dirty
  - OU Invalid : bloco não contem dado válido
- ✓ CPU solicita leitura:
- ✓ Se cache não tem cópia:
  - Controlador coloca Read Miss no barramento
- ✓ Outras caches:
- ✓ Read misses: todas as caches vão dar "snoop" no barramento
  - Controlador bisbilhota todo endereço colocado no barramento
  - Se a cache possui uma cópia Exclusive do bloco requisitado, fornece o bloco em resposta a requisição de leitura e aborta o acesso à memória.

Protocolo Snooping Write Back

- ✓ Cada bloco de cache vai estar em UM dos estados:
  - Shared : bloco pode ser lido
  - OU Modified/Exclusive : cache tem somente uma cópia que pode ser escrita e dirty
  - OU Invalid : bloco não contem dado válido
- ✓ CPU solicita escrita:
- ✓ Se cache não tem cópia:
  - Controlador coloca Write Miss no barramento
- ✓ Outras caches:
- ✓ Write misses: todas as caches vão dar "snoop" no barramento
  - Controlador bisbilhota todo endereço colocado no barramento
  - Se a cache possui uma cópia Exclusive do bloco requisitado, atualiza a memória e Invalida a cópia.
  - Se a cache possui uma cópia Shared do bloco requisitado invalida a cópia

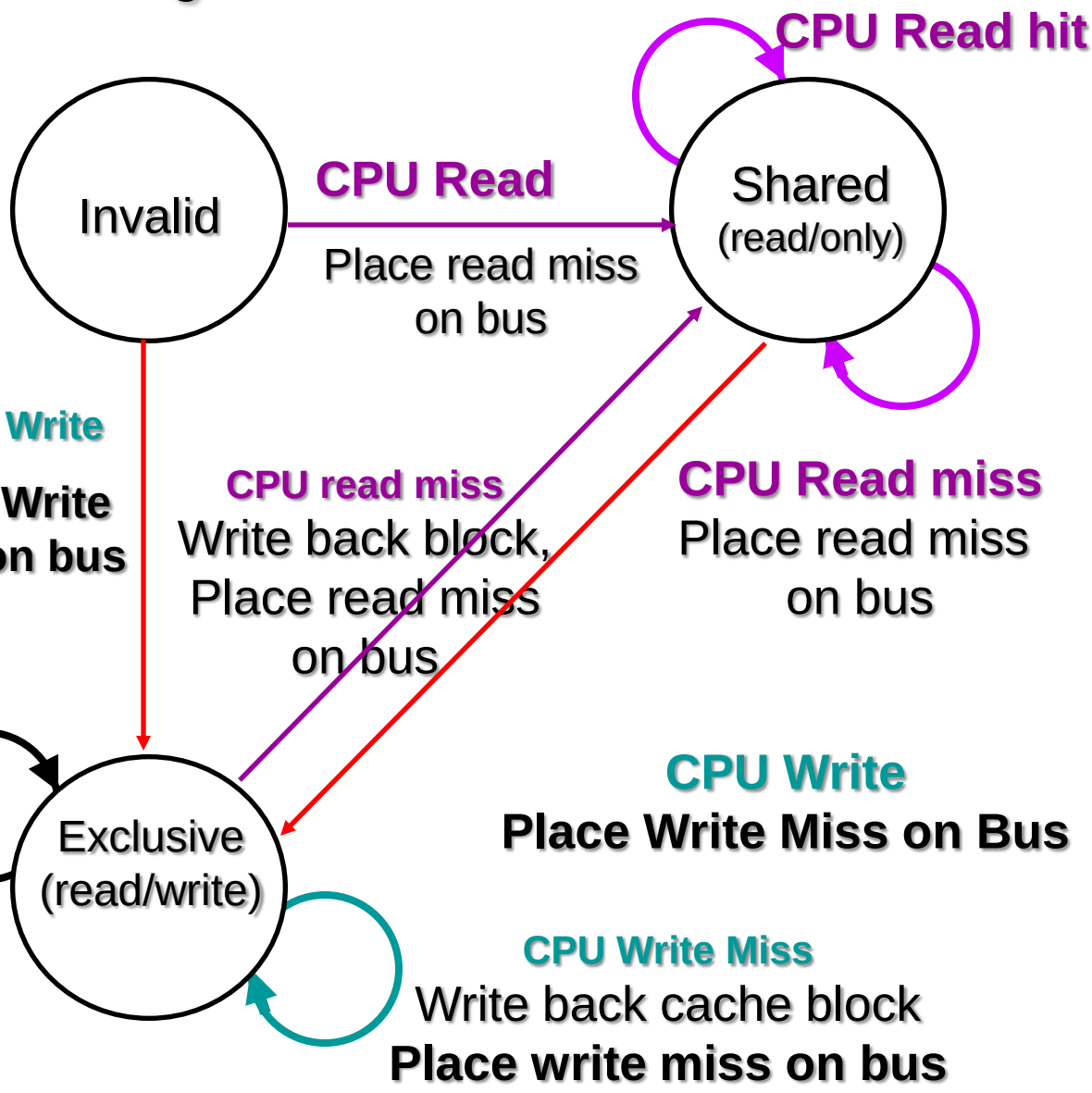
Snooping: Write-Back - CPU

- ✓ Mudança de estado considerando requisições da *CPU* para cada bloco de cache
- ✓ Blocos que não estão na cache não são validos



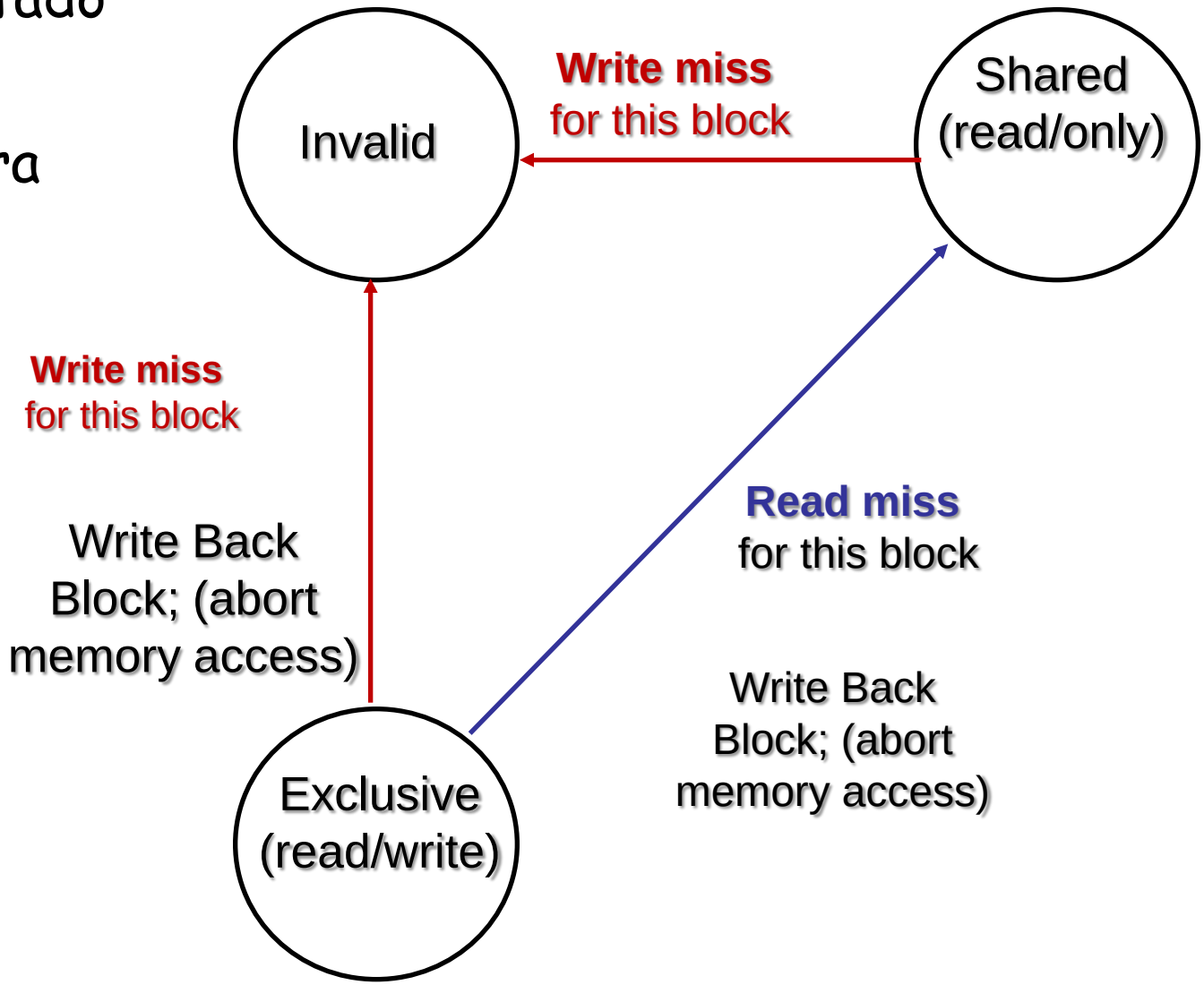
Snooping: Write-Back
Substituição de Bloco

Mudança de estado para requisições da *CPU* para cada bloco da cache



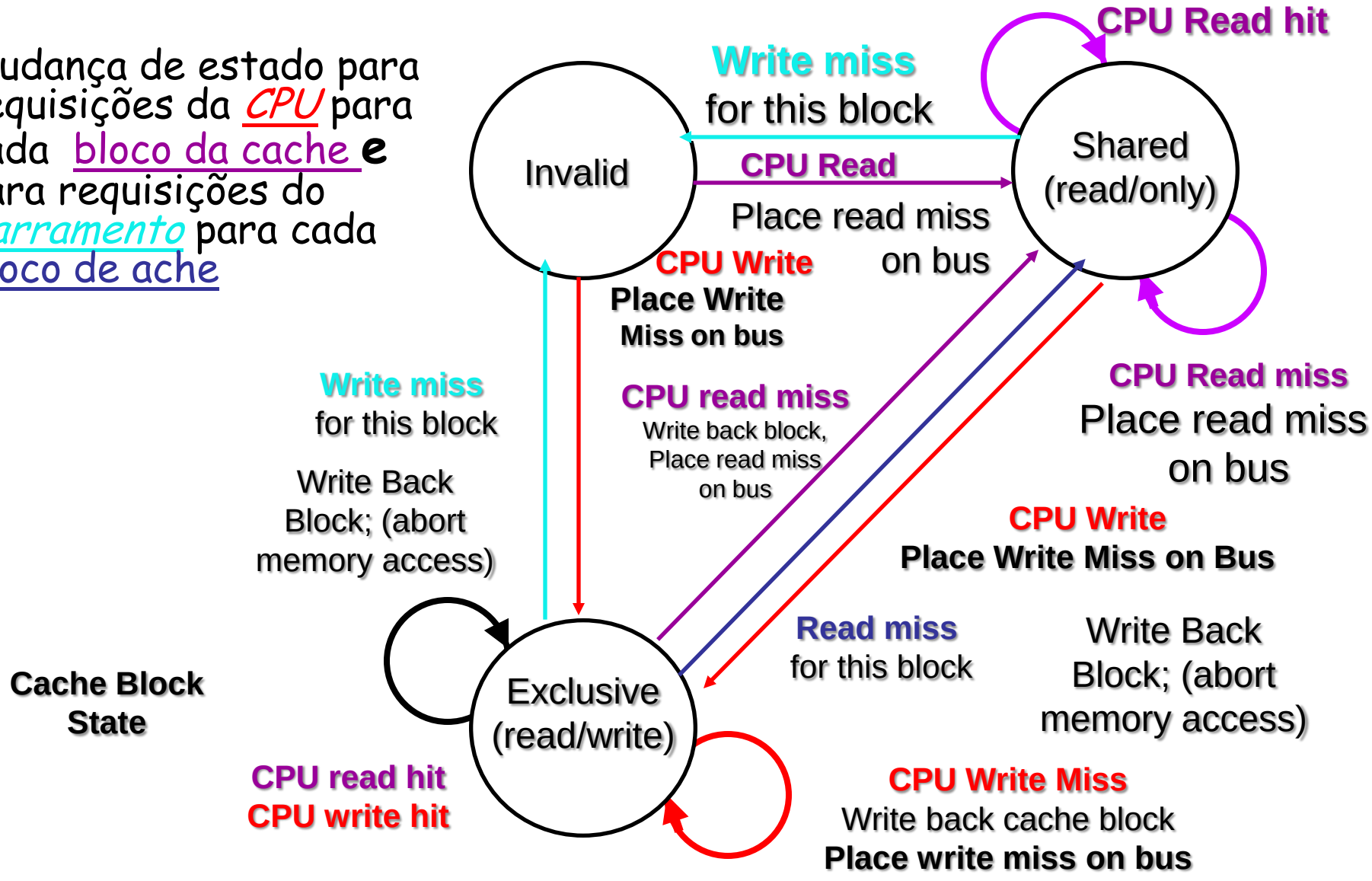
Snooping: Write-Back - Bus

Mudança de estado considerando operações do *barramento* para cada bloco de cache



Snooping Write-back

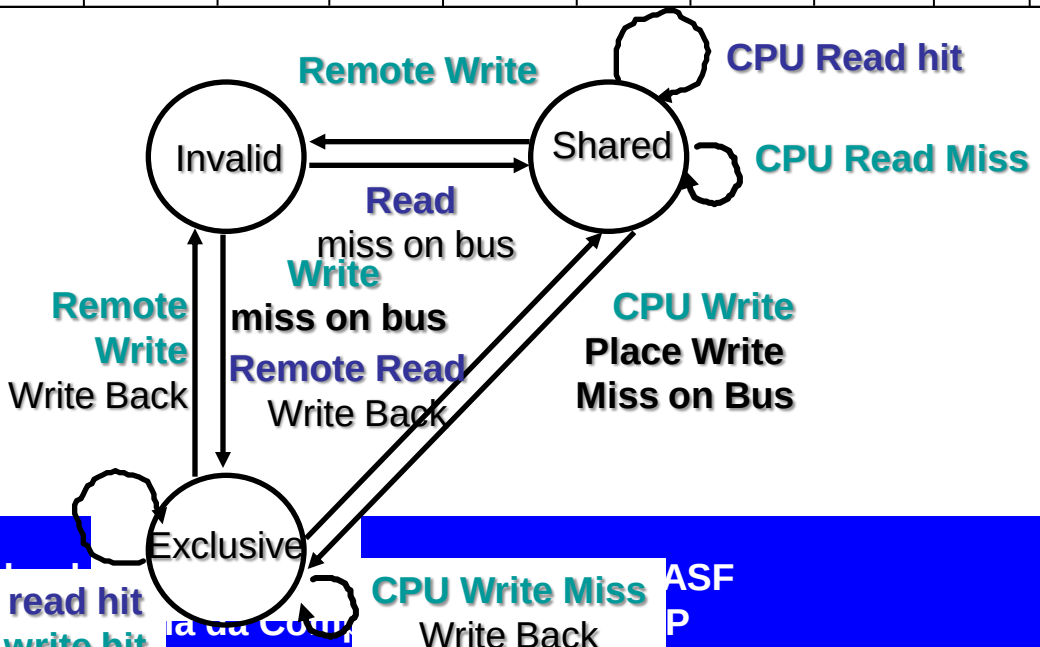
Mudança de estado para requisições da *CPU* para cada bloco da cache **e** para requisições do *barramento* para cada bloco de ache



Exemplo

	Processor 1			Processor 2			Bus				Memory	
	P1			P2			Bus				Memory	
step	State	Addr	Value	State	Addr	Value	Action	Proc.	Addr	Value	Addr	Value
P1: Write 10 to A1												
P1: Read A1												
P2: Read A1												
P2: Write 20 to A1												
P2: Write 40 to A2												

- Assuma que estado inicial da cache é “não válido”
- A1 e A2 mapeiam para o mesmo slot de cache mas A1 != A2

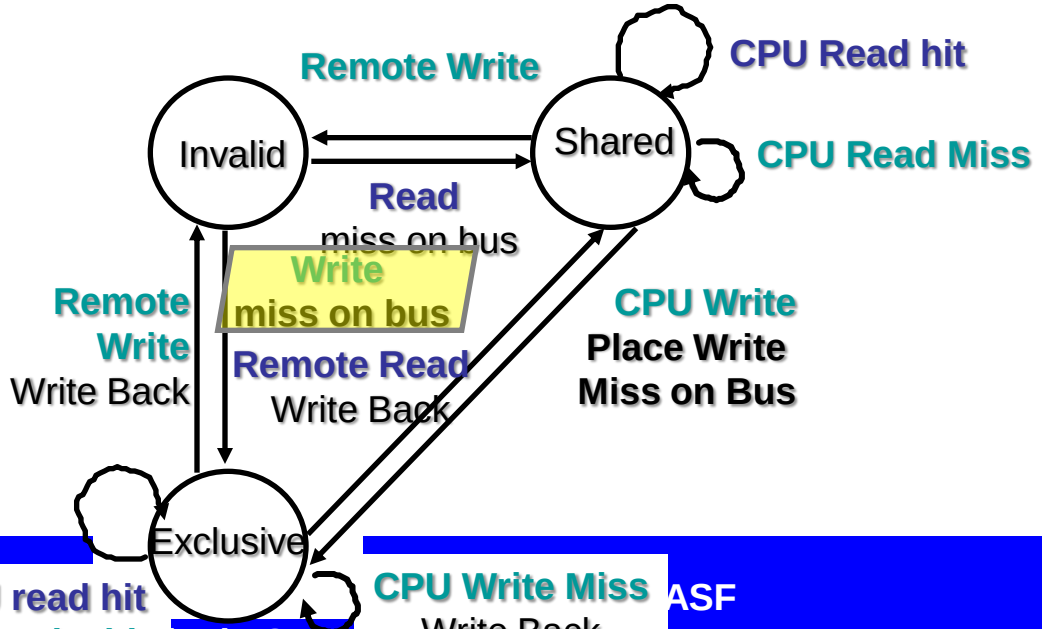


Exemplo: Passo 1

	P1			P2			Bus				Memory	
step	State	Addr	Value	State	Addr	Value	Action	Proc.	Addr	Value	Addr	Value
P1: Write 10 to A1	Excl.	A1	10				WrMs	P1	A1			
P1: Read A1												
P2: Read A1												
P2: Write 20 to A1												
P2: Write 40 to A2												

- Assuma que estado inicial da cache é "não válido"
- A1 e A2 mapeiam para o mesmo slot de cache mas A1 != A2

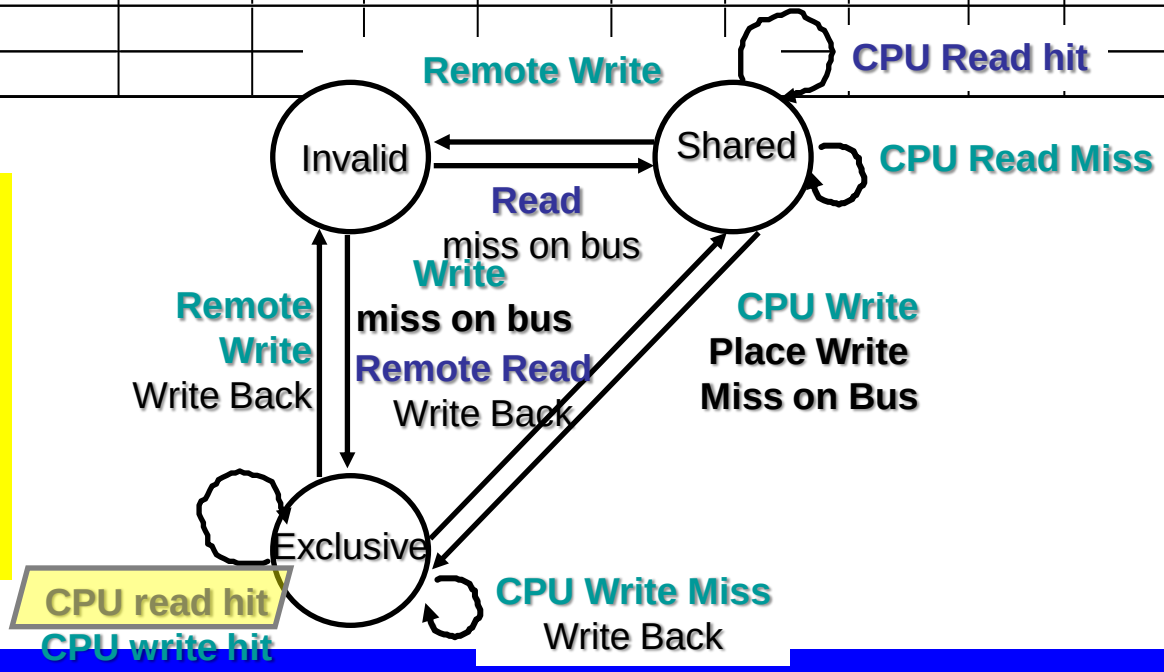
Estado ativo:



Exemplo: Passo 2

	P1			P2			Bus				Memory	
step	State	Addr	Value	State	Addr	Value	Action	Proc.	Addr	Value	Addr	Value
P1: Write 10 to A1	Excl.	A1	10				WrMs	P1	A1			
P1: Read A1	Excl.	A1	10									
P2: Read A1												
P2: Write 20 to A1												
P2: Write 40 to A2												

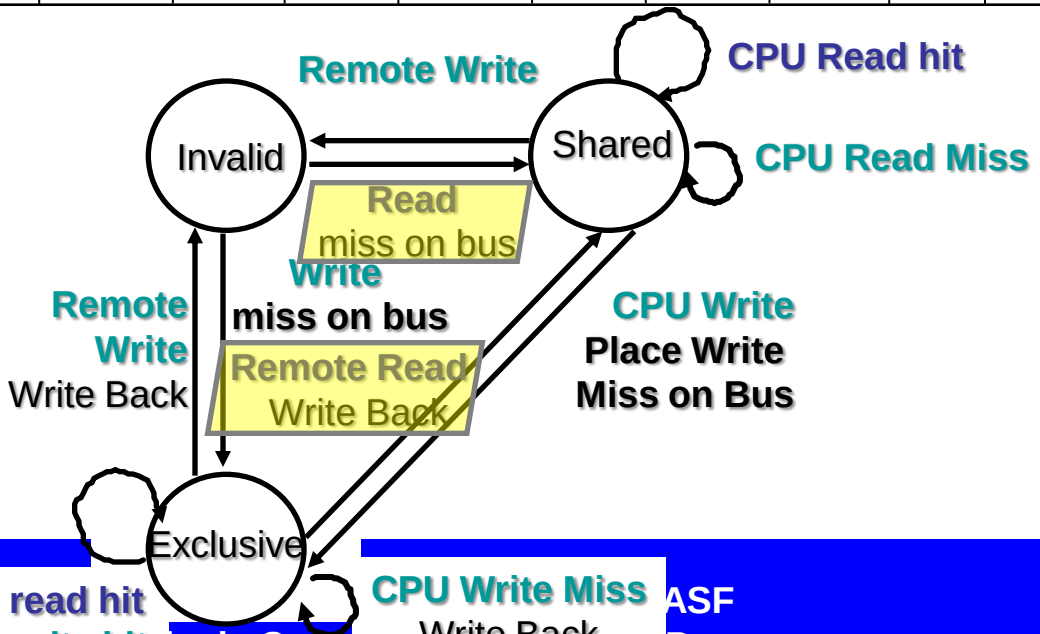
- Assuma que estado inicial da cache é “não válido”
- A1 e A2 mapeiam para o mesmo slot de cache mas A1 != A2



Exemplo: Passo 3

	P1			P2			Bus				Memory	
step	State	Addr	Value	State	Addr	Value	Action	Proc.	Addr	Value	Addr	Value
P1: Write 10 to A1	Excl.	A1	10				WrMs	P1	A1			
P1: Read A1	Excl.	A1	10									
P2: Read A1				Shar.	A1		RdMs	P2	A1		A1	
	Shar.	A1	10				WrBk	P1	A1	10	A1	10
				Shar.	A1	10	RdDa	P2	A1	10		
P2: Write 20 to A1												
P2: Write 40 to A2												
												10

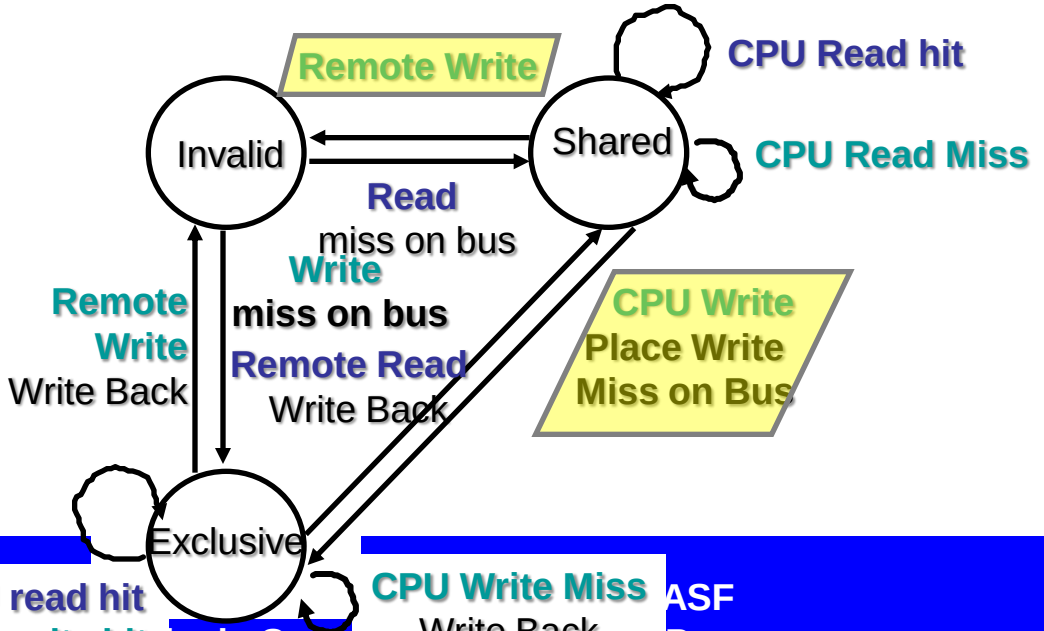
- Assuma que estado inicial da cache é “não válido”
- A1 e A2 mapeiam para o mesmo slot de cache mas A1 != A2



Exemplo: Passo 4

	P1			P2			Bus				Memory	
step	State	Addr	Value	State	Addr	Value	Action	Proc.	Addr	Value	Addr	Value
P1: Write 10 to A1	Excl.	A1	10				WrMs	P1	A1			
P1: Read A1	Excl.	A1	10									
P2: Read A1				Shar.	A1		RdMs	P2	A1		A1	
	Shar.	A1	10				WrBk	P1	A1	10	A1	10
				Shar.	A1	10	RdDa	P2	A1	10	A1	10
P2: Write 20 to A1	Inv.			Excl.	A1	20	WrMs	P2	A1			
P2: Write 40 to A2												
												10

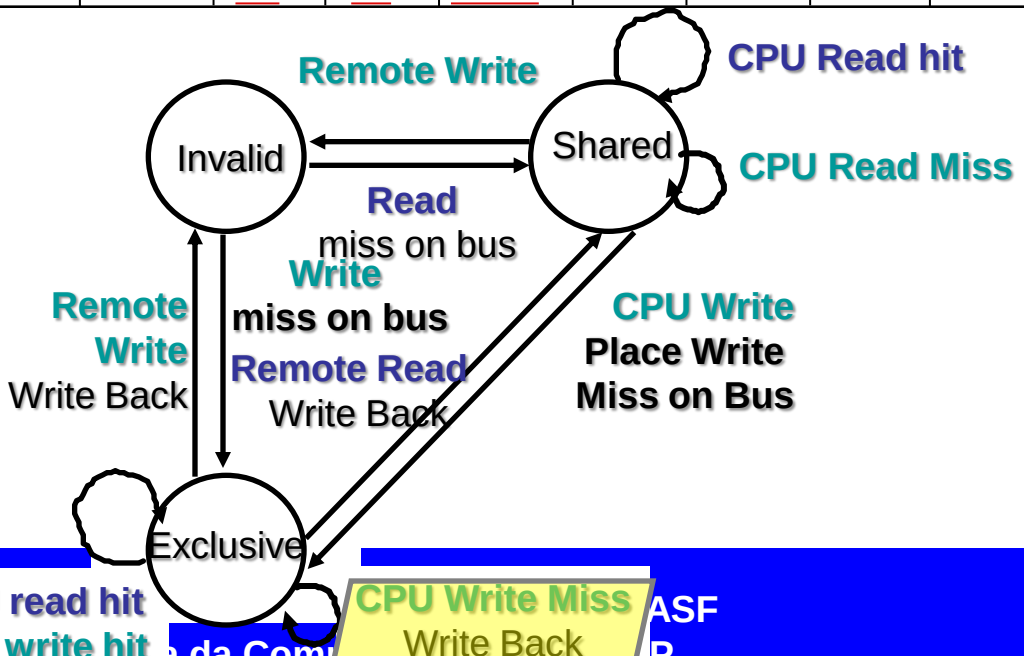
- Assuma que estado inicial da cache é “não válido”
- A1 e A2 mapeiam para o mesmo slot de cache mas A1 != A2



Exemplo: Passo 5

	P1			P2			Bus				Memory	
step	State	Addr	Value	State	Addr	Value	Action	Proc.	Addr	Value	Addr	Value
P1: Write 10 to A1	Excl.	A1	10				WrMs	P1	A1			
P1: Read A1	Excl.	A1	10								A1	
P2: Read A1				Shar.	A1		RdMs	P2	A1		A1	
	Shar.	A1	10				WrBk	P1	A1	10	A1	10
				Shar.	A1	10	RdDa	P2	A1	10	A1	10
P2: Write 20 to A1	Inv.			Excl.	A1	20	WrMs	P2	A1		A1	10
P2: Write 40 to A2							WrMs	P2	A2			10
				Excl.	A2	40	WrBk	P2	A1	20		20

- Assuma que estado inicial da cache é "não válido"
- A1 e A2 mapeiam para o mesmo slot de cache mas A1 != A2



Problemas na Implementação

- ✓ Conflitos na escrita:
  - A cache não pode ser atualizada até que o barramento seja obtido
    - Senão outro processador pode obter o barramento e escrever no mesmo bloco de cache
  - Dois passos:
    - Conseguir o barramento (controlado pela arbitragem)
    - Colocar miss no barramento e completar a operação
  - Se um miss ocorrer enquanto esperando pelo barramento, o miss deve ser tratado (talvez seja necessário invalidar bloco) e deve se tentar novamente

Implementando Snooping Caches

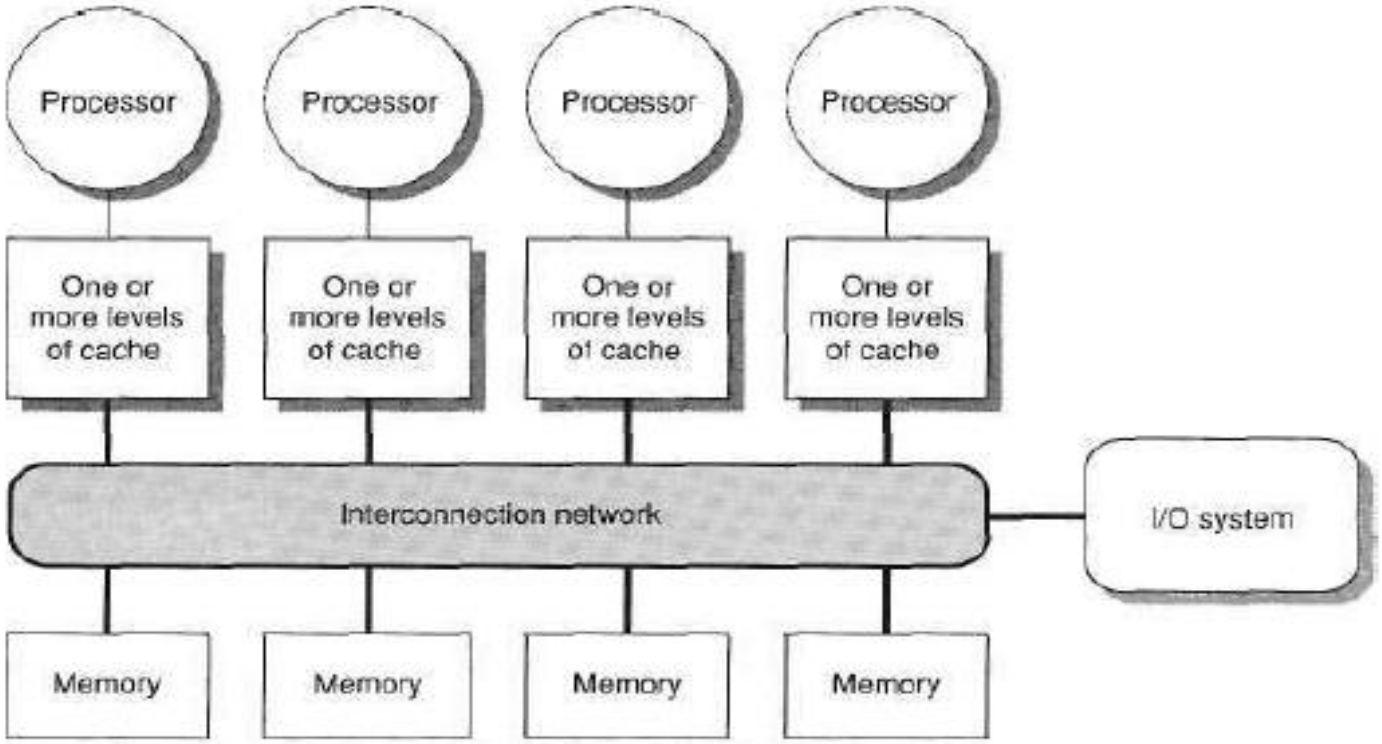
- ✓ Controlador de cache deve implementar novos comandos para realizar coerência
- ✓ Controladores de cache deve monitorar continuamente o barramento (endereços)
  - Se tag do endereço coincidir com algum bloco de cache deve invalidar (no caso de escrita)
- ✓ Esta checagem contínua dos tags na cache com os endereços de barramento pode interferir com a checagem do tag para mandar dado para a CPU.
  - solução 1: duplicar tags para caches L1 para permitir checagens paralelas
  - solução 2: Tag é checado na Cache L2 desde que L2 obedeça principio de inclusão com L1 cache

Limitações

- ✓ Técnica de Snooping é aplicável a multiprocessadores com memória compartilhada centralizada:
  - Memória ÚNICA para todas as CPUs
  - Multi-processador baseado em barramento
    - barramento deve suportar tanto acessos para coerência como para acessos normais de memória
- ✓ Fatores limitantes no número de processadores

Limitações

- ✓ Processadores mais rápidos e em maior número....
- ✓ Como suportar este novo cenário?



Desempenho: Sharing misses

- ✓ Desempenho da Cache é uma combinação
  - Cache misses de cada processador (aplicação)
  - Cache misses causados pela comunicação
    - Escrita em variável compartilhada resulta em invalidações e subsequentes cache misses

Desempenho: Sharing misses

- ✓ 4th C: *coherence miss*
  - Inclui: Compulsory, Capacity, Conflict
  - Dois tipos de coherence misses, (relacionados ao compartilhamento)
    - True sharing miss: palavra é escrita pelo processador, invalidando o bloco, e então é acessada por outro processador.
    - False sharing miss: palavra é escrita pelo processador, invalidando o bloco, e então uma outra palavra do mesmo bloco é acessada por outro processador.
  - Aumento no compartilhamento → aumento das coherence misses

Exemplo: True v. False Sharing v. Hit?

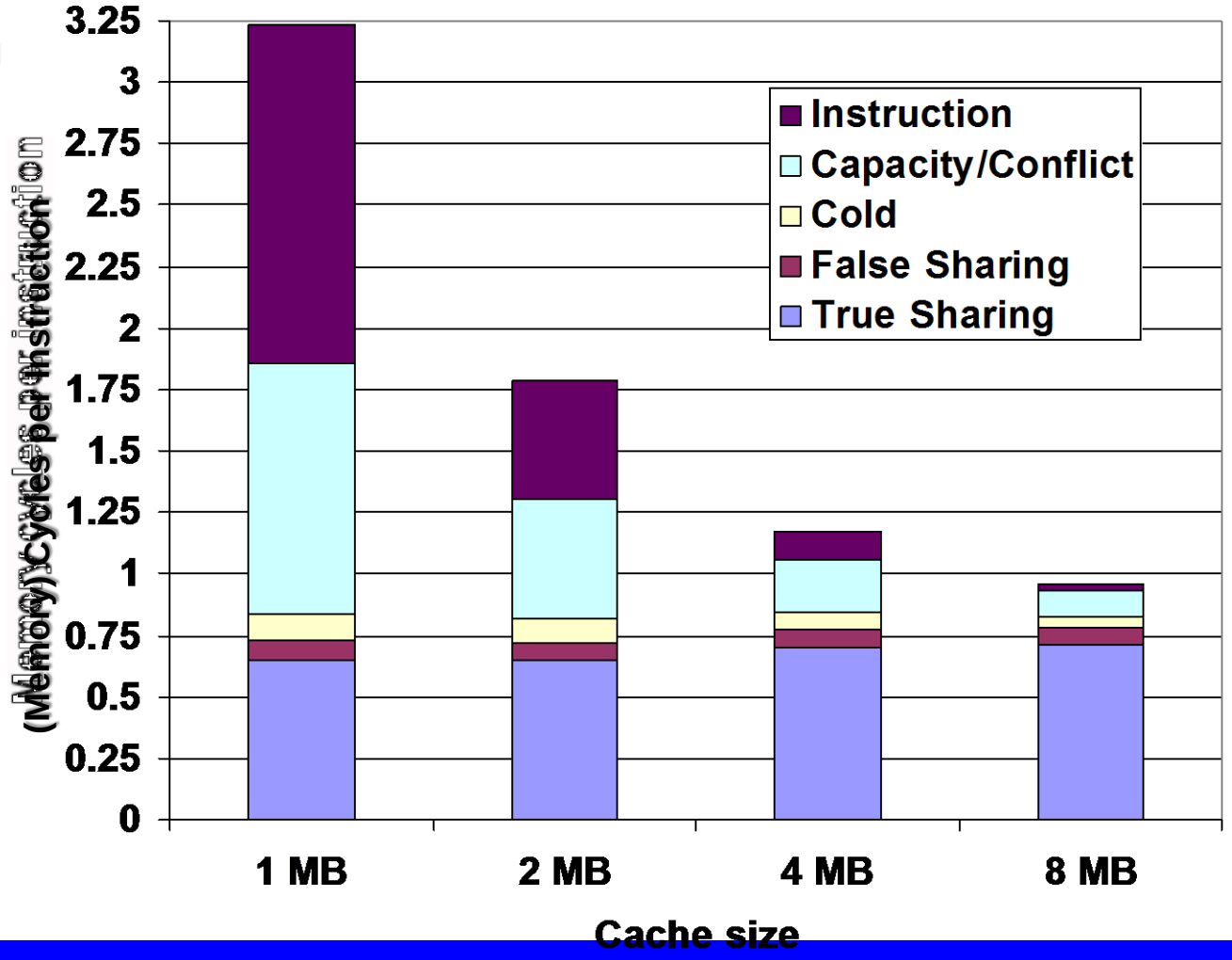
- **Assuma que x1 e x2 estão no mesmo bloco de cache**
- **P1 e P2 possuem cópias de x1 e x2.**

Tempo	P1	P2	True miss, False miss, Hit?
1	Write x1		True miss; invalidate x1 em P2
2		Read x2	False miss; x1 não é usado P2
3	Write x1		False miss; x1 não é usado P2
4		Write x2	False miss; x1 não é usado P2
5	Read x2		True miss; invalidate x2 em P1

Desempenho Multiprocessador com 4 Processadores

Benchmarck: OLTP, Decision Support (Database), Search Engine

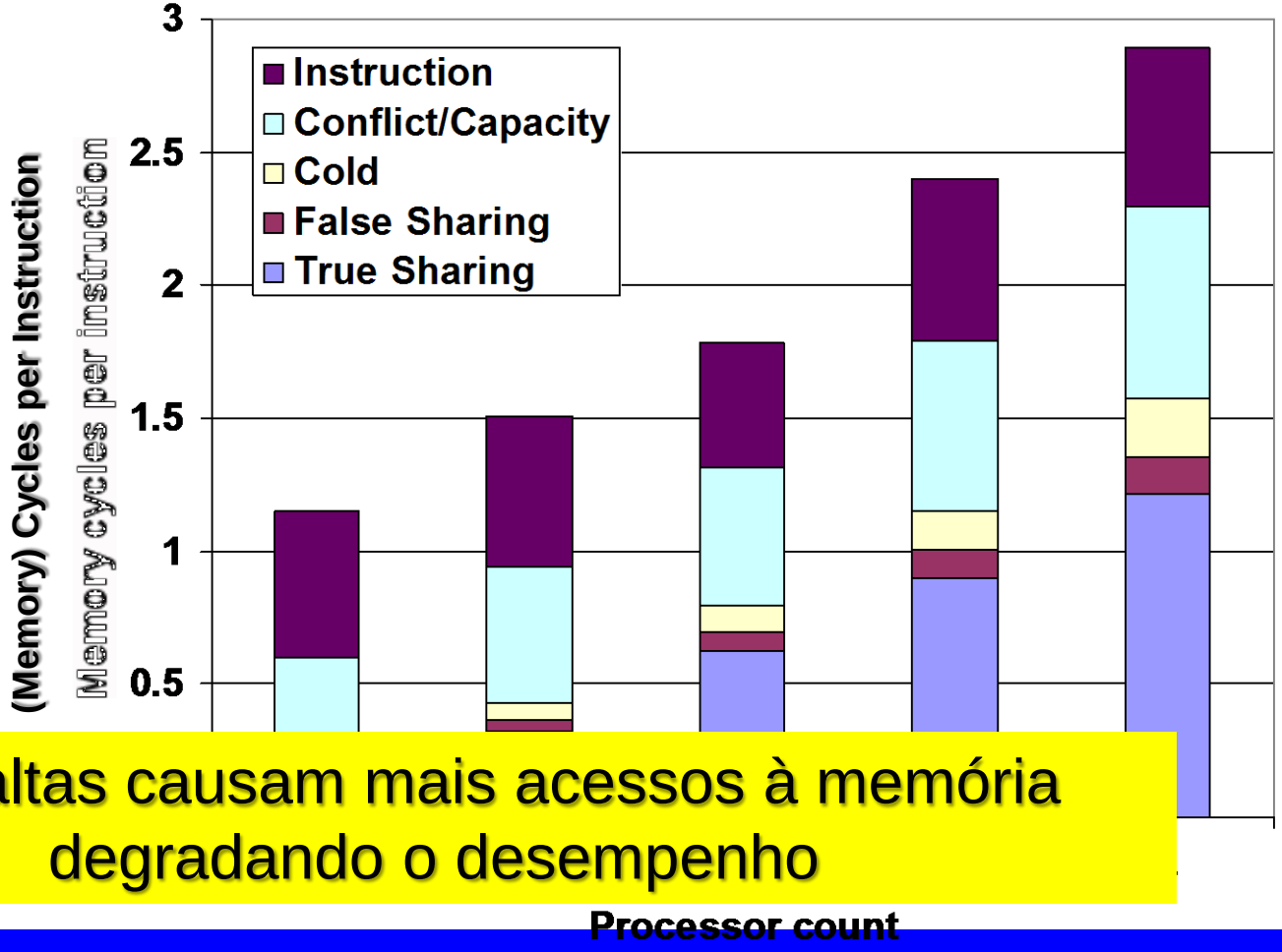
- True sharing e false sharing não mudam com aumento da cache
  - 1 MB para 8 MB (L3 cache)
- Faltas da aplicação diminuem com aumento da cache
  - (Instruction, Capacity/Conflict, Compulsory)



Desempenho Multiprocessador com 4 Processadores

Benchmarck: OLTP, Decision Support (Database), Search Engine

Faltas devido ao compartilhamento (True sharing, false sharing) aumentam quando aumenta número de processadores
- 1 para 8 CPUs



Mais faltas causam mais acessos à memória degradando o desempenho