SSC0611
Arquitetura de Computadores

21ª Aula – Arquiteturas Paralelas

Arquitetura MIMD com Memória Compartilhada

Coerência de Cache

Profa. Sarita Mazzini Bruschi
sarita@icmc.usp.br

Memórias Cache
Políticas de Atualização

- As memórias caches possuem dois modos básicos para trabalhar em relação à atualização dos dados na memória principal durante uma escrita:
  - Write-through: Os dados são atualizados tanto na memória cache como na memória principal
  - Write-back: Os dados são atualizados apenas na memória cache, e copiados para a memória principal, apenas quando da substituição do bloco/linha modificado na cache

Problema de Coerência de Cache

- Os sistemas multiprocessados com memória compartilhada apresentam os seguintes problemas:
  - Conteção de memória: o módulo de memória pode manipular somente uma requisição de memória por vez
  - Contenção de comunicação: contenção dos links de comunicação, mesmo que deseje-se acessar módulos diferentes da memória
  - Tempo de latência: tempo de comunicação utilizando as redes de interconexão tende a aumentar quando o número de processadores aumenta e a interconexão se torna mais complexa

Tempo de acesso à memória

Multiprocessador	Ano de lançamento	SMP ou NUMA	Máximo de processadores	Rede de interconexão	Tempo típico de acesso à memória remota (ns)
Servidores Sun Starfire	1996	SMP	64	Barramentos de endereço múltiplos, switch de dados	500
SGI Origin 3000	1999	NUMA	512	Hipercubo largo	500
Cray T3E	1996	NUMA	2048	Toro 3D de duas vias	300
Série HP V	1998	SMP	32	Crossbar 8 × 8	1000
Compaq AlphaServer GS	1999	SMP	32	Barramentos comutados	400
Sun V880	2002	SMP	8	Barramentos comutados	240
HP Superdome 9000	2003	SMP	64	Barramentos comutados	275

Figura 9.1.3 Tempos de acesso remoto típicos para recuperar uma word de uma memória remota em multiprocessadores de memória compartilhada.

Fonte: Organização e Projeto de Computadores (Patterson & Hennesy), 2003

Problema de Coerência de Cache

- Para solucionar os problemas citados, utiliza-se memórias cache, objetivando reduzir o número de acessos à memória principal
- Problema: várias cópias de um mesmo bloco compartilhado por diversos processos ao mesmo tempo
  - As várias cópias devem estar consistentes
  - Problema denominado “*Problema de coerência de cache*”

Exemplo de algoritmos com compartilhamento de variáveis

- Problema do Produtor/Consumidor

```
Producer:
    if count <= N then
    mutexbegin
        buffer[ in ] := item;
        in := in + 1 mod N;
        count := count + 1;
    mutexend
```

```
Consumer:
    if count <> 0 then
    mutexbegin
        item := buffer[ out ];
        out := out + 1 mod N;
        count := count - 1;
    mutexend
```

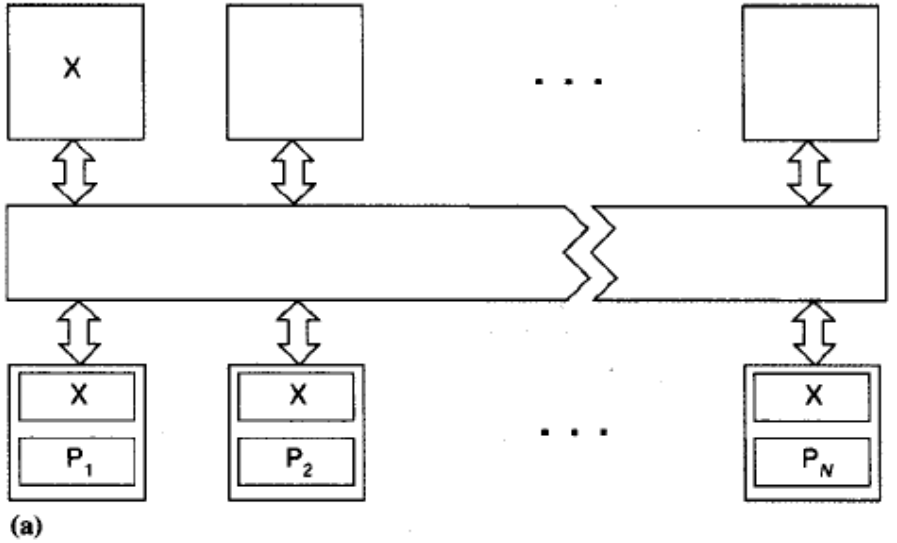
- Variáveis compartilhadas: *buffer*, *in*, *out* e *count*

Políticas de coerência de cache

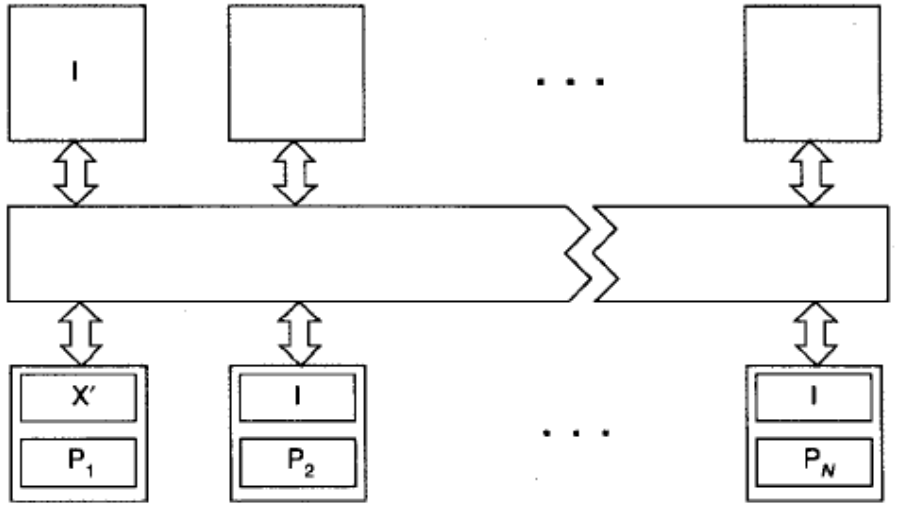
- *Write-invalidate*: política que mantém a consistência da seguinte maneira:
  - Requisições de leitura são tratadas localmente caso já exista uma cópia local do bloco
  - Todas as outas cópias são invalidadas caso o bloco seja atualizado
    - Uma próxima atualização pelo mesmo processador pode ser feita localmente, já que as outras cópias estão invalidadas
  - Múltiplos leitores, um escritor

Políticas de coerência de cache

- *Write-invalidate*



(a)



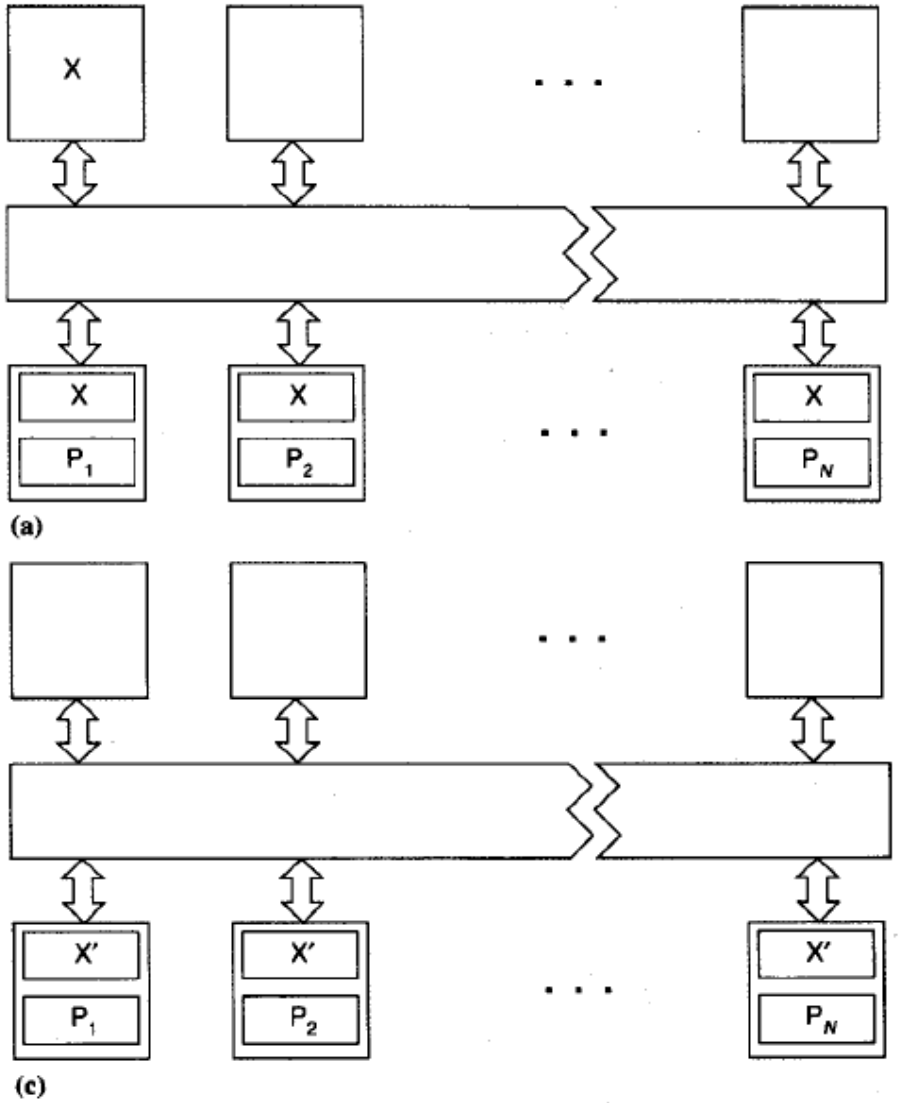
(b)

Políticas de coerência de cache

- *Write-update*: política que mantém a consistência da seguinte maneira:
  - Ao invés de invalidar as cópias, esta política atualiza todas as outras cópias
  - Múltiplos leitores e escritores

Políticas de coerência de cache

- *Write-update*



Exemplo 1
Produtor/consumidor

- Considere um processo produtor *P* e um processo consumidor *C*
  - Cada processo está executando em uma máquina diferente
  - O processo *P* executa *k* entradas na região crítica (inserções) seguidas
  - O processo *C* executa *k* entradas na região crítica (remoções) seguidas

Exemplo 1
Produtor/consumidor

- Comparação do custo de comunicação para acesso à variável *count* para *k* execuções consecutivas na região crítica

Communication Cost		Bounded-Buffer Problem
Write-invalidate	Invalidations	1
	Misses	1
Write-update	Updates	*K*

Exemplo 2

- Considere a seguinte seqüencia de instruções

Etapa	Processador	Atividade da memória	Endereço de memória
1	P1	Leitura	100
2	P2	Escrita	104
3	P1	Leitura	100
4	P2	Leitura	104
5	P1	Leitura	104
6	P2	Leitura	100

- Conte o número de transações do barramento utilizando os protocolos *write-update, write-back* e que o tamanho do bloco é igual a uma palavra (4 bytes)

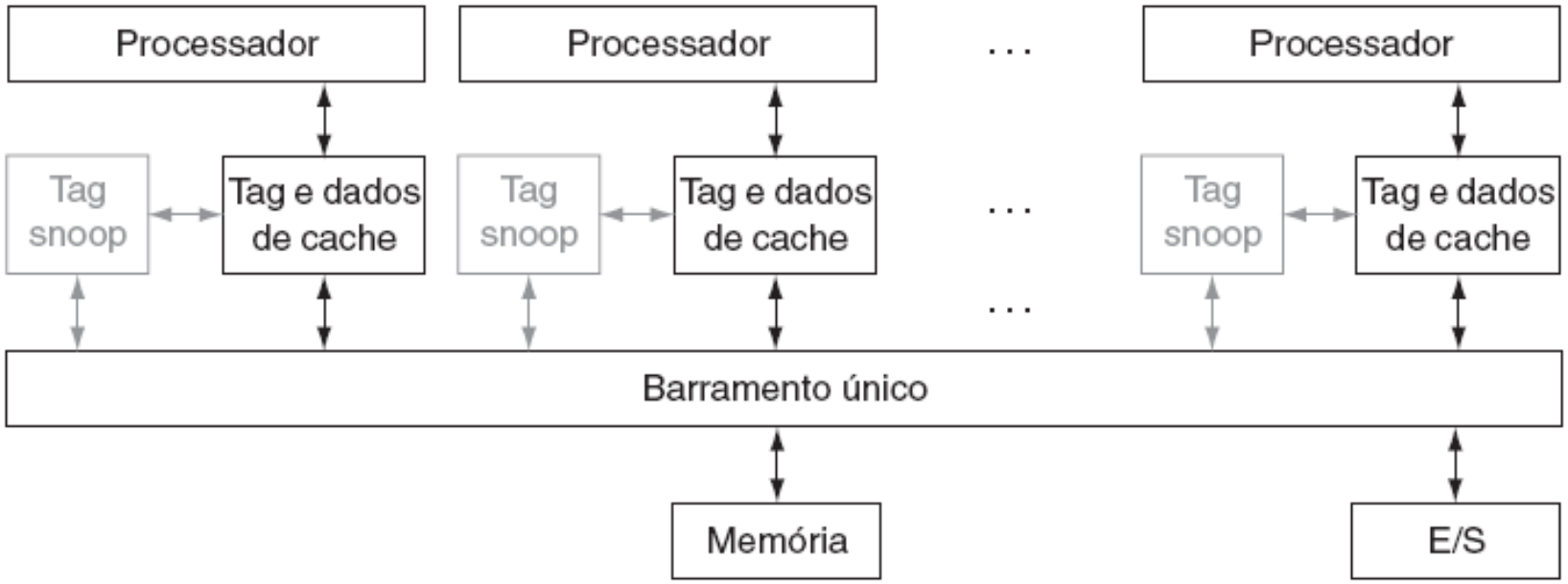
Protocolos para manter a coerência

- Independente da política utilizada (*write-invalidate* ou *write-update*), as mensagens para manter a coerência dos dados devem ser enviadas para pelo menos as que possuem uma cópia do dado
- Para redes onde o custo de se fazer um broadcast não é muito alto, utiliza-se um protocolo denominado *Snoopy Cache Protocol*
- Para redes de interconexão mais complexas utiliza-se um protocolo denominado *Directory Protocol*

Snoopy Cache Protocol ou *snooping*

- Utilizado em sistemas baseados em barramento, onde todos os processadores “observam” as transações de memória e tomam a atitude correta, dependendo da política
  - Invalidar (política *write-invalidate*) ou
  - Atualizar (política *write-update*)
- O *snooping* distribui a responsabilidade da coerência entre os controles de cache, os quais reconhecem que a linha é compartilhada
- Tem que ter a possibilidade de se fazer *brodcast*
- Bastante utilizado em máquinas de escala pequena, como os processadores atuais

Snoopy Cache Protocol ou *snooping*



Fonte: Organização e Projeto de Computadores (Patterson & Hennesy), 2003

Directory Protocol

- Utilizado quando os processadores não são conectados por barramento
- Existe um controlador central, o qual faz parte do controlador da memória principal, que coleta e mantém informações sobre as cópias dos dados nas caches
- Quando uma linha da cache é refenciada, a base de dados é verificada para saber onde é “limpa” (shared) ou “suja” (modified)

Directory Protocol

- Antes de um processador atualizar os dados compartilhados, este deve requisitar ao controlador acesso exclusivo à linha desejada
- Para obter o acesso exclusivo, o controlador envia uma mensagem a todos os processadores com uma cópia cacheada dessa linha, forçando cada processador a invalidar sua cópia

Directory Protocol

- O controlador tem que receber uma mensagem de cada processador antes de dar o acesso exclusivo
- Quando um processador tenta ler uma linha que está com acesso exclusivo a outro processador, é enviada uma notificação de falha de acesso para o controlador
- O controlador requisita que o processador que tem acesso exclusivo faça a atualização da linha na memória principal e passa o acesso exclusivo ao processador que requisitou a linha e compartilhada para leitura para o processador que detinha o acesso exclusivo

Directory Protocol

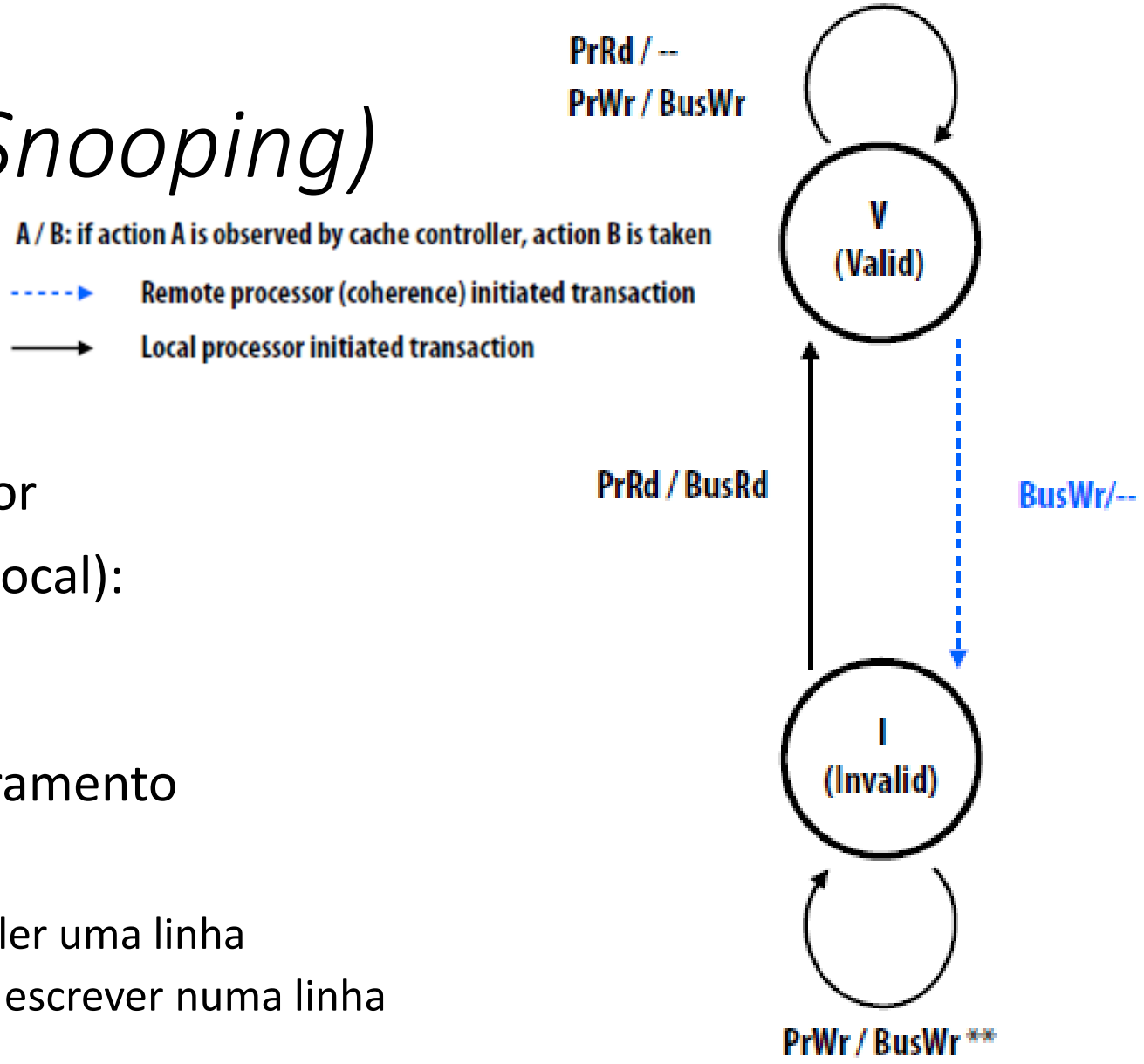
- Problema: é centralizado, criando um gargalo central e sobrecarregando a comunicação entre os vários controladores de cache e o controlador central
- São eficientes em sistemas de larga escala, os quais envolvem múltiplos barramentos ou interconexões mais complexas

Write-through e *write-invalidate (Snooping)*

- Dois estados:
  - V: válido
  - I: inválido
- Duas operações do processador

  (disparadas pelo processador local):
  - PrRd: leitura do processador
  - PRWr: escrita do processador
- Duas transações vistas no barramento

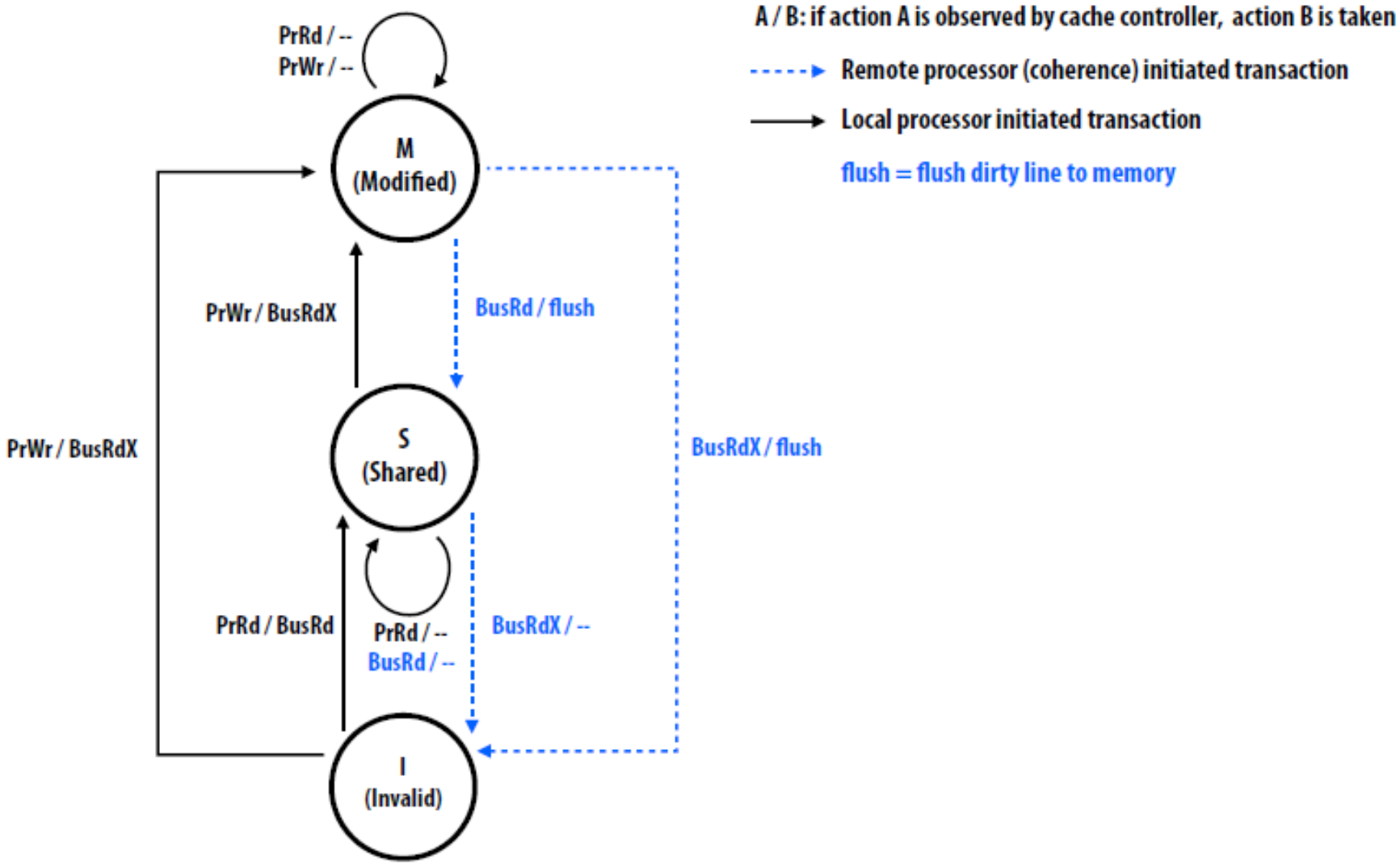
  (das caches remotas):
  - BusRd: outro processador quer ler uma linha
  - BusWr: outro processador quer escrever numa linha



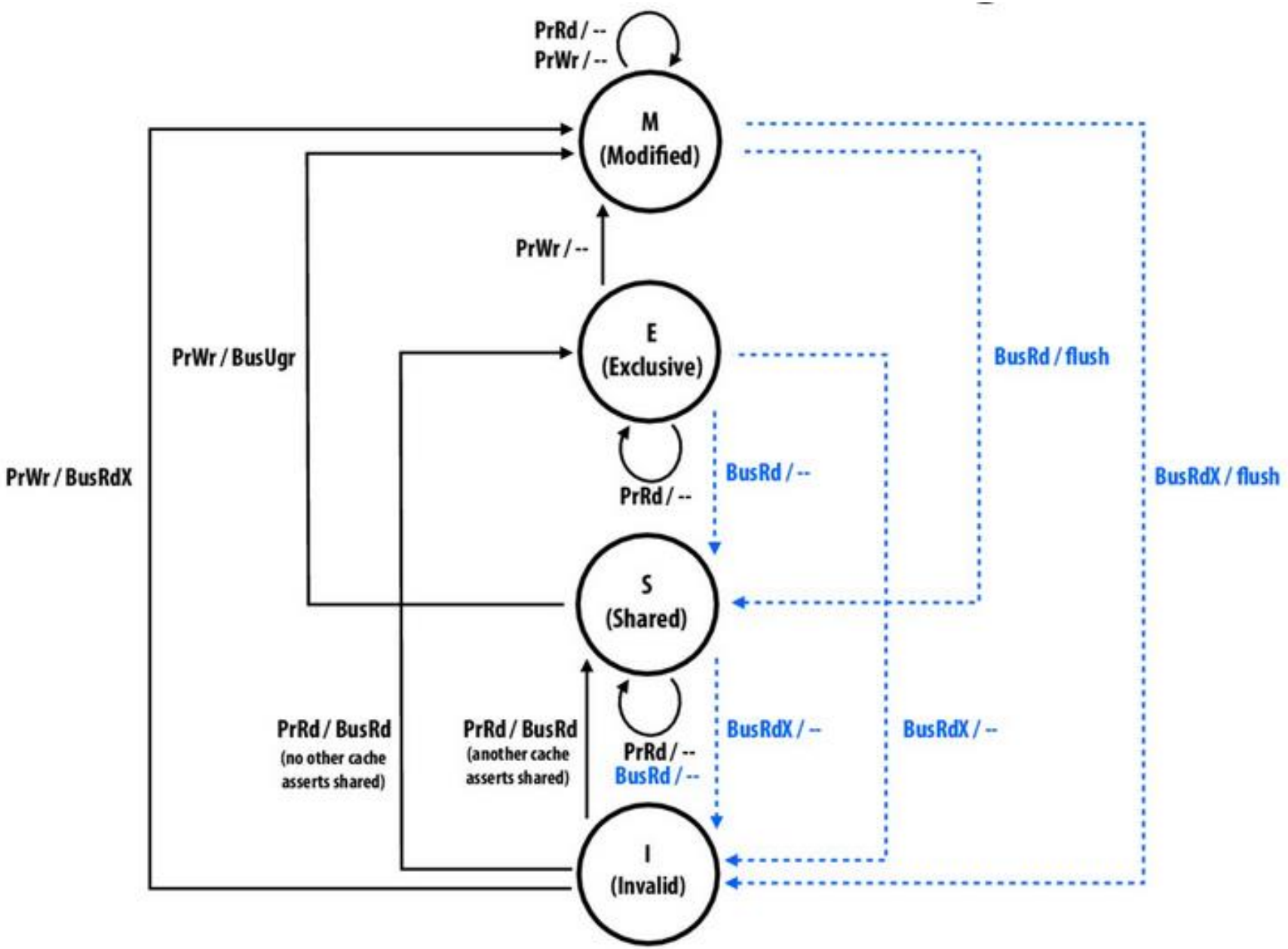
Write-back e *write-invalidate*

- Os exemplos das transições a seguir consideram a utilização de *snooping*
- Protocolo **MSI:** Baseia-se em uma máquina de estados finitos com 3 estados:
  - ***Shared*** (*Compartilhado* - somente leitura): esse bloco na cache é “limpo” (não escrito) e pode ser compartilhado
  - ***Modified*** (*Modificado* – leitura/escrita): esse bloco na cache é “sujo” (escrito) e pode *não* ser compartilhado
  - ***Invalid*** (*Inválido*): esse bloco de cache não possui dados válidos
- Protocolo **MESI:** Baseia-se em uma máquina de estados finitos com 4 estados:
  - Divide o estado ***Shared*** (compartilhado) em dois outros estados:
    - ***Shared*** (compartilhado): existem múltiplas cópias do bloco
    - ***Exclusive*** (exclusivo): existe apenas uma cópia
      - Existe apenas uma cópia do bloco e um acerto de escrita não precisa invalidar

Protocolo MSI



Protocolo MESI



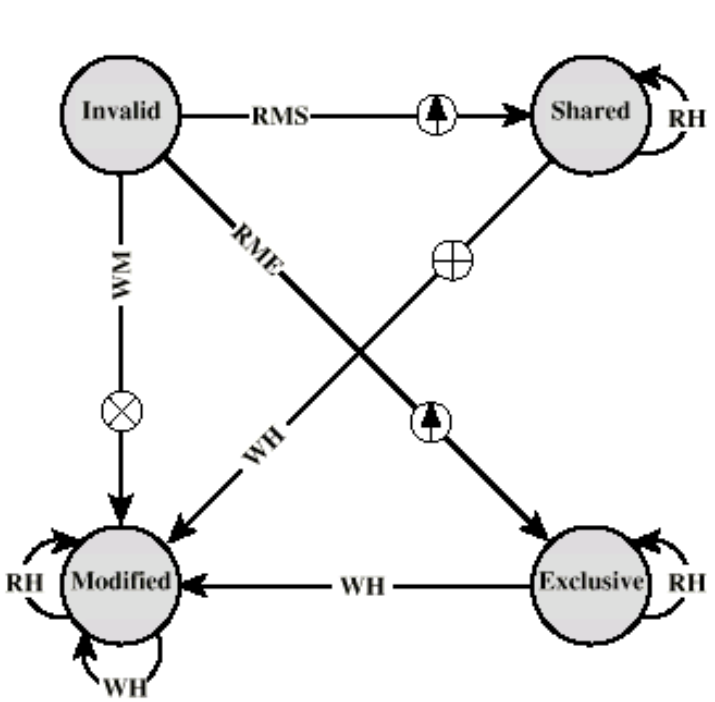
Protocolo MESI

- Resumo dos 4 estados

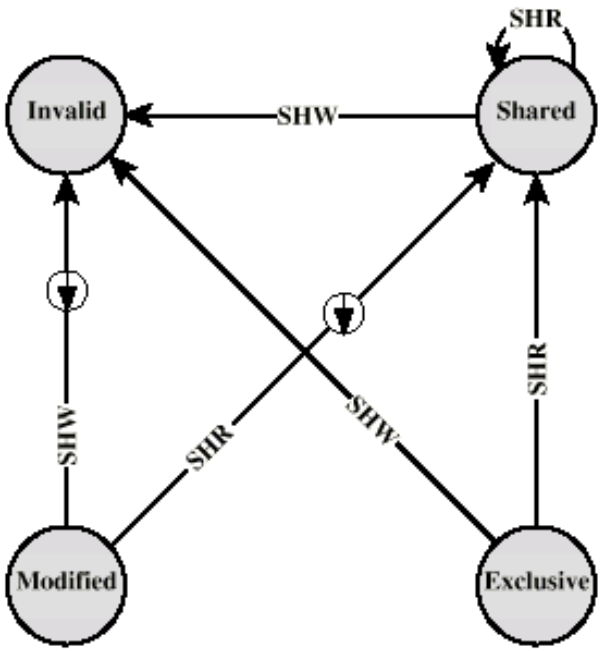
	M (Modified)	**E (Exclusive)**	**S (Shared)**	**I (Invalid)**
A linha é válida?	Sim	Sim	Sim	Não
A cópia da MP é...	Desatualizada	Válida	Válida	-
Existe cópia em outra cache?	Não	Não	Talvez	Talvez
Uma escrita nessa linha...	Não precisa do barramento	Não precisa do barramento	Precisa do barramento e atualiza a cache	Acessa direto o barramento

Fonte: Computer Organization and Architecture (William Stallings), 2003

Protocolo MESI



(a) Line in cache at initiating processor



(b) Line in snooping cache

RH	Read hit		Dirty line copyback
RMS	Read miss, shared		
RME	Read miss, exclusive		Invalidate transaction
WH	Write hit		
WM	Write miss		Read-with-intent-to-modify
SHR	Snoop hit on read		
SHW	Snoop hit on write or read-with-intent-to-modify		Cache line fill

Fonte: Computer Organization and Architecture (William Stallings), 2003

Protocolo MOESI

- Acrescenta o estado Owner, onde a cache onde está o bloco neste estado fornece o dado em caso de falha de leitura no barramento, uma vez que a memória não possui o dado atualizado
- Pode existir cópias do dado em outras caches

Exercício

- Conte o número de transações no barramento para a seqüência de instruções, explicitando as transições na máquina de estado finito de um protocolo MESI
- Considere que os processadores utilizam as políticas *write-invalidate*, *write-back*, tamanho de bloco de uma word e que os blocos não estão na memória cache (estado inicial *Invalid*)

Exercício

Etapa	Processador	Atividade da memória	Endereço de memória
1	P1	Leitura	100
2	P1	Escrita	100
3	P3	Leitura	104
4	P2	Escrita	100
5	P3	Escrita	104
6	P3	Escrita	100
7	P1	Leitura	100