cin.ufpe.br

Centro de Informática

U · F · P · E

Universidade Federal de Pernambuco

Coerência de Cache

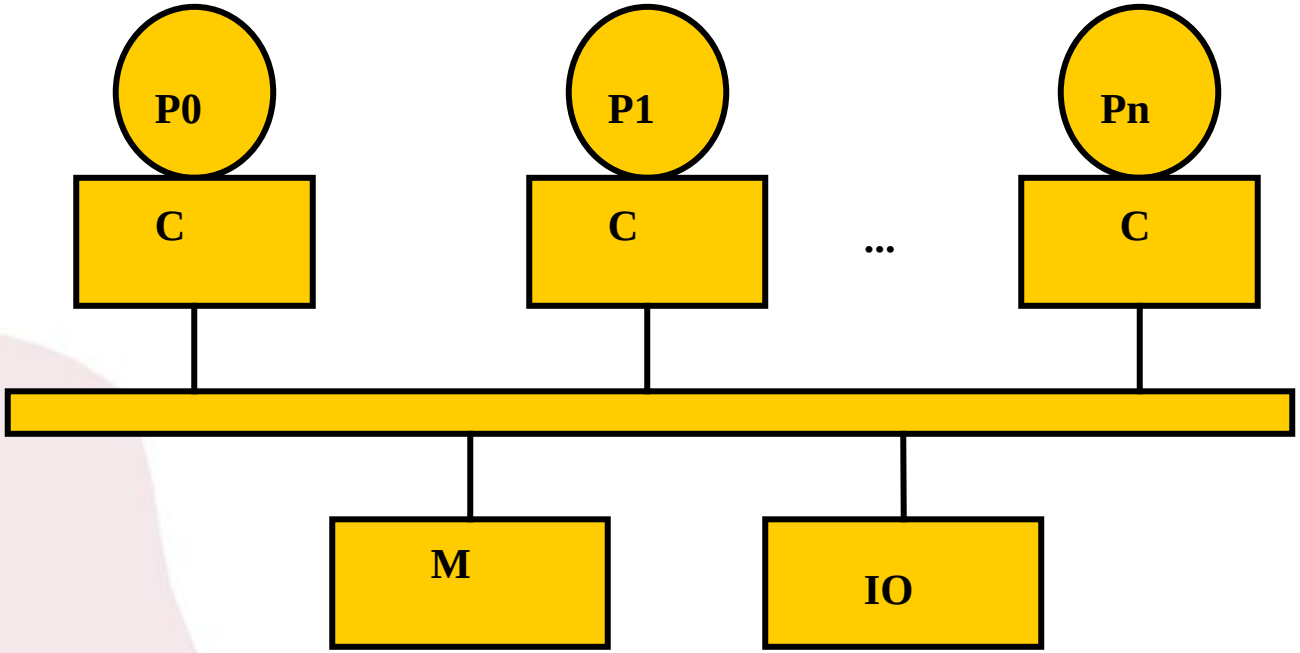
Protocolo de Coerência para processadores multi-cores baseados em barramentos

Roteiro da Aula

- Multiprocessador Simétrico de Memória Compartilhada - SMP
- Problema Coerência e consistência de memória em SMPs
- Técnica para Coerência SMPs: Snooping
- Desafios e Limitações de SMPs com Snooping

Multiprocessadores Simétricos de Memória Centralizada SMPs

- Comunicação entre processadores através de barramento



Multi-core

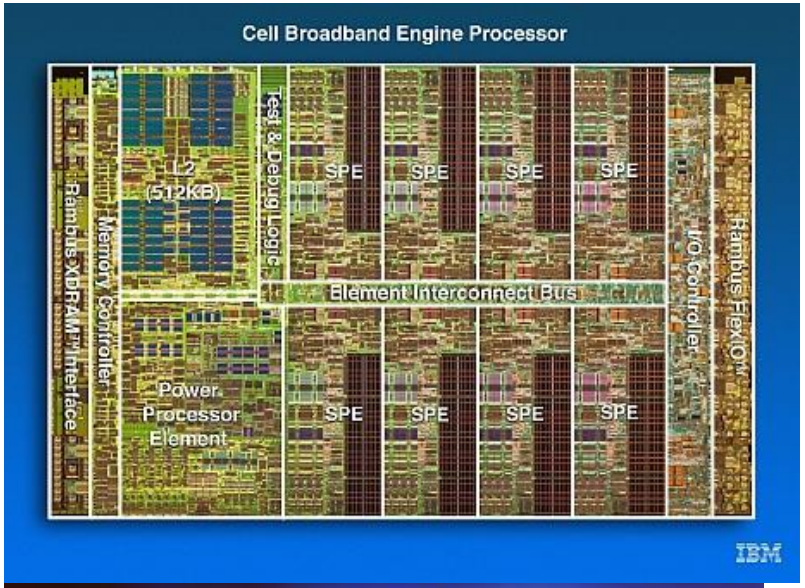
- Multiprocessador implementado em um único circuito integrado

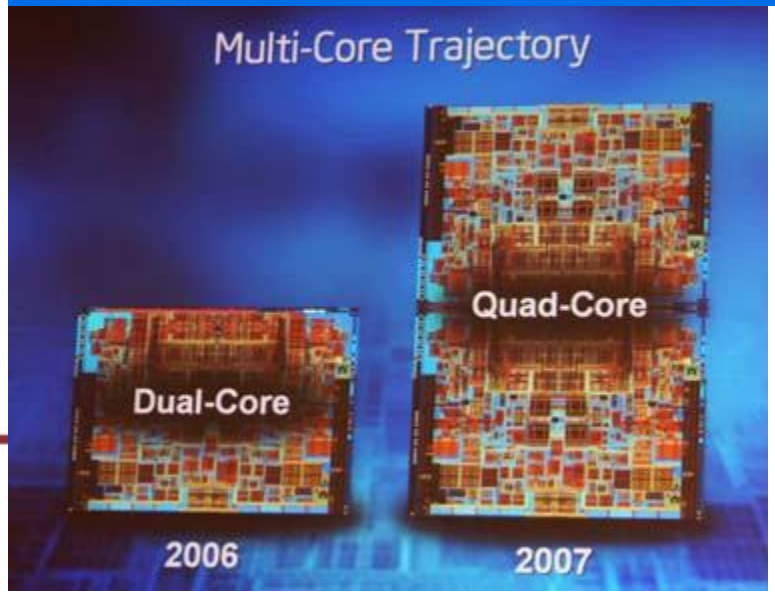
Multi-Core Processor

Centro de Informática U·F·P·E

IBM

Power5

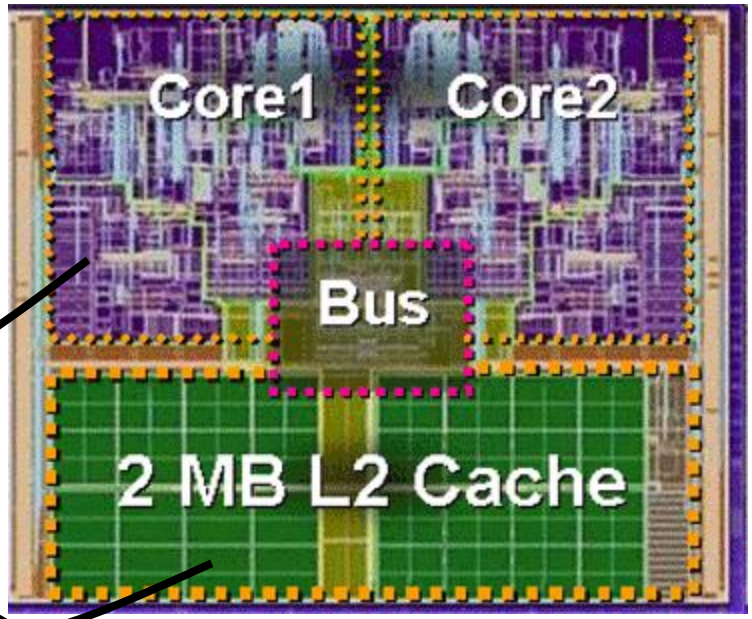




BROADCOM
Connecting everything®

Intel Core 2 Duo

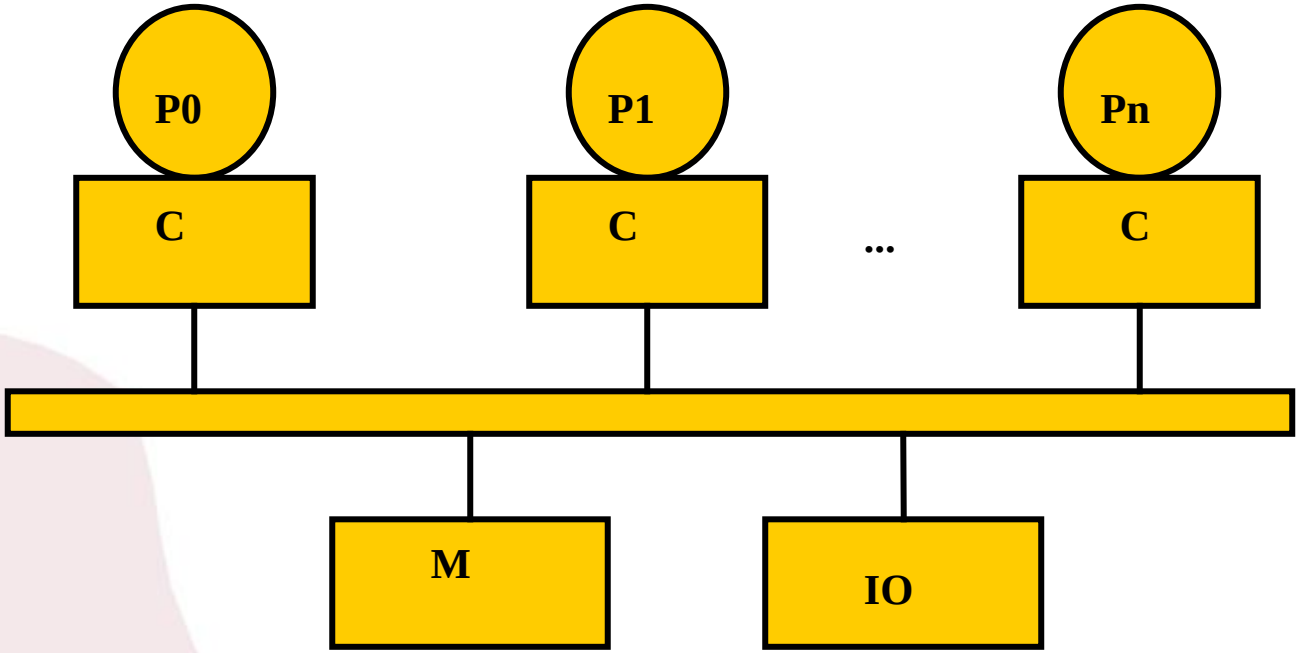
- Cores homegêneos
  - Superscalares
    - (escalonamento dinâmico, especulação, multiple issue)
- Interconexão baseada em barramento
- Cada “core” tem cache local (L1)
- Memória compartilhada
  - (cache L2) no chip



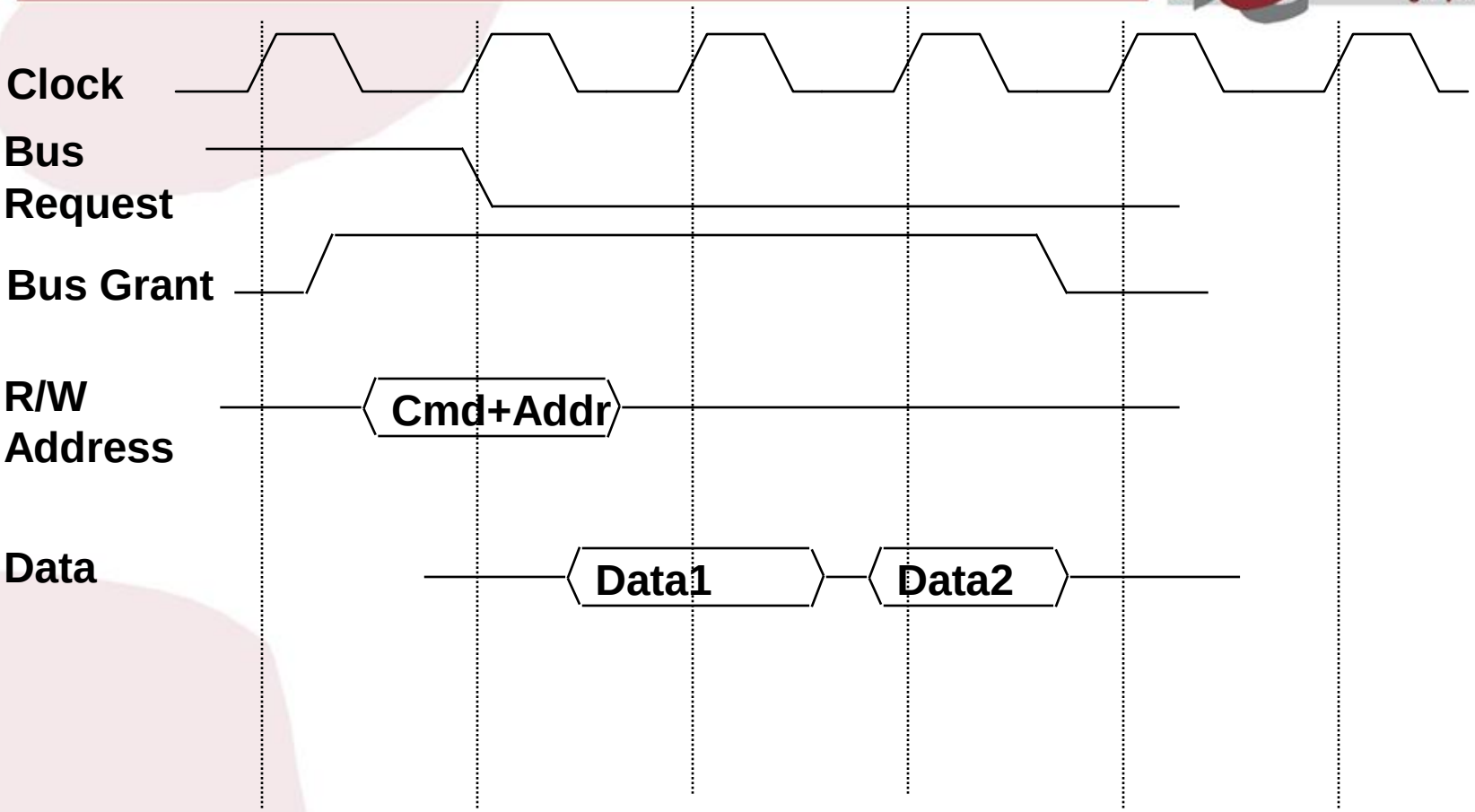
Source: Intel Corp.

Multiprocessadores Simétricos de Memória Centralizada SMPs

- Comunicação entre processadores através de barramento



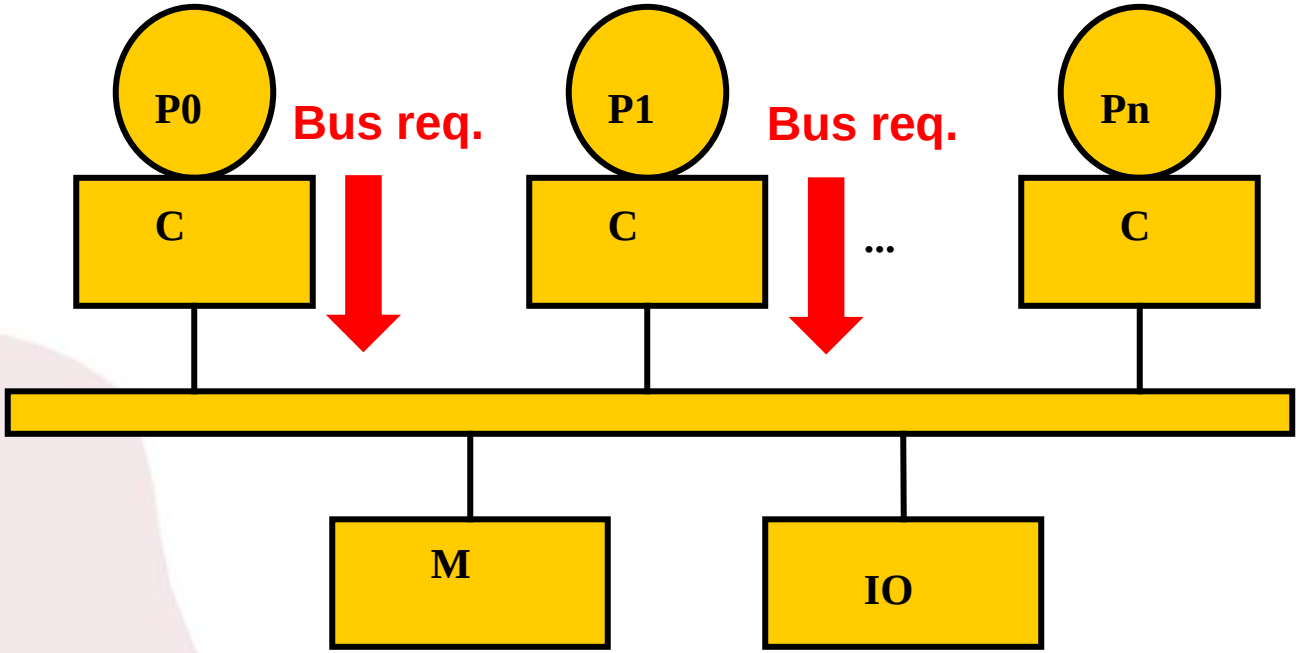
Protocolo de Barramento



- Só um processador usa o barramento por vez

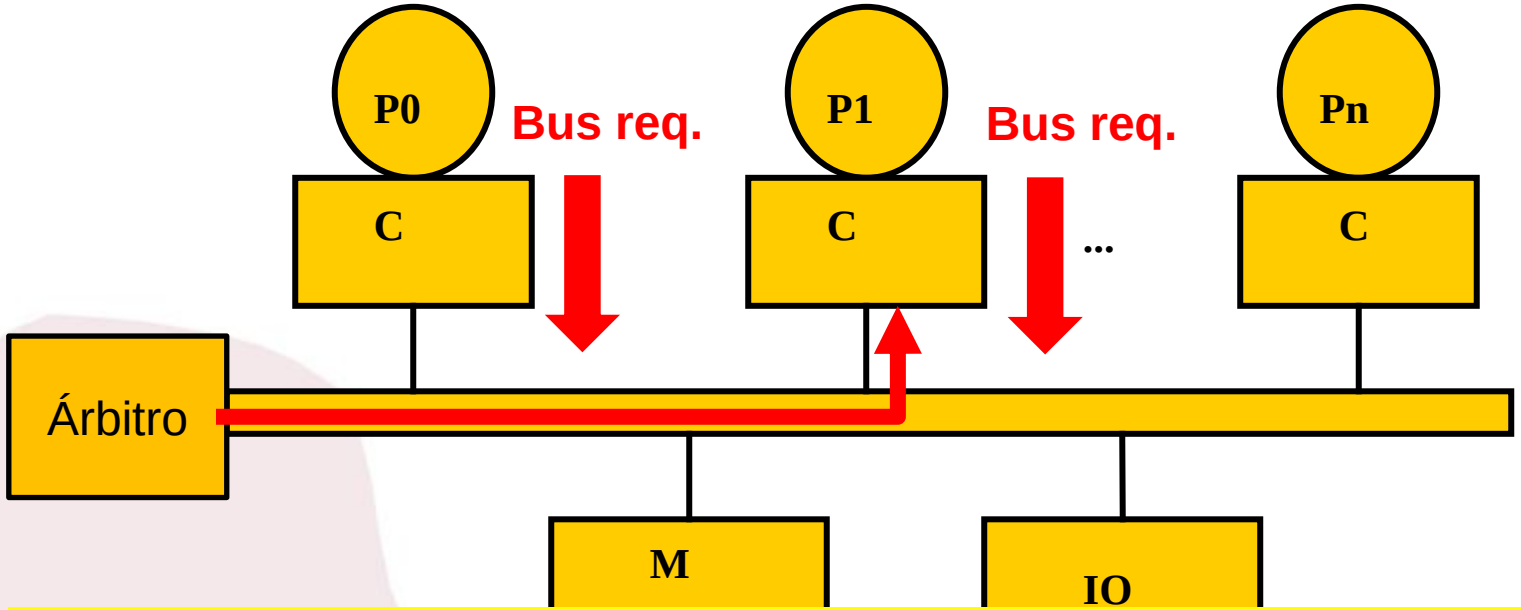
Contenção de Barramento

- Dois processadores querem usar o barramento ao mesmo tempo



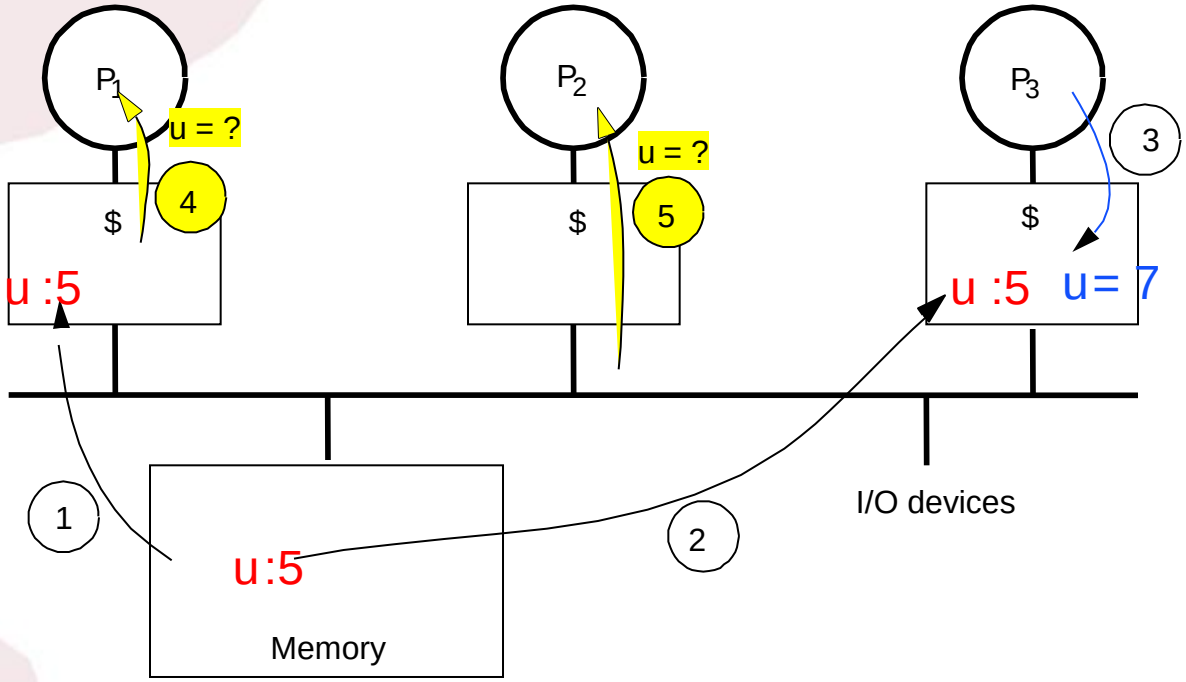
Contenção de Barramento

- Dois processadores querem usar o barramento ao mesmo tempo



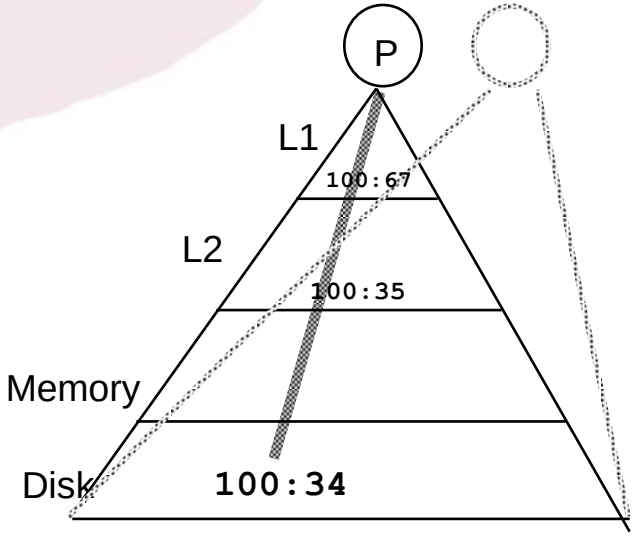
- O processador P0 tem que esperar

Problema de Coerência de Cache em SMPs



- Processadores verão diferentes valores de u após evento 3
- Caches write-through: valores inconsistentes
- Caches write back: valor escrito na memória depende do momento que que ocorre o flush de cache e os valores são atualizados na memória

Modelo de Memória



- Leitura de um enederço deveria retornar o último valor escrito no endereço

- Dois conceitos:

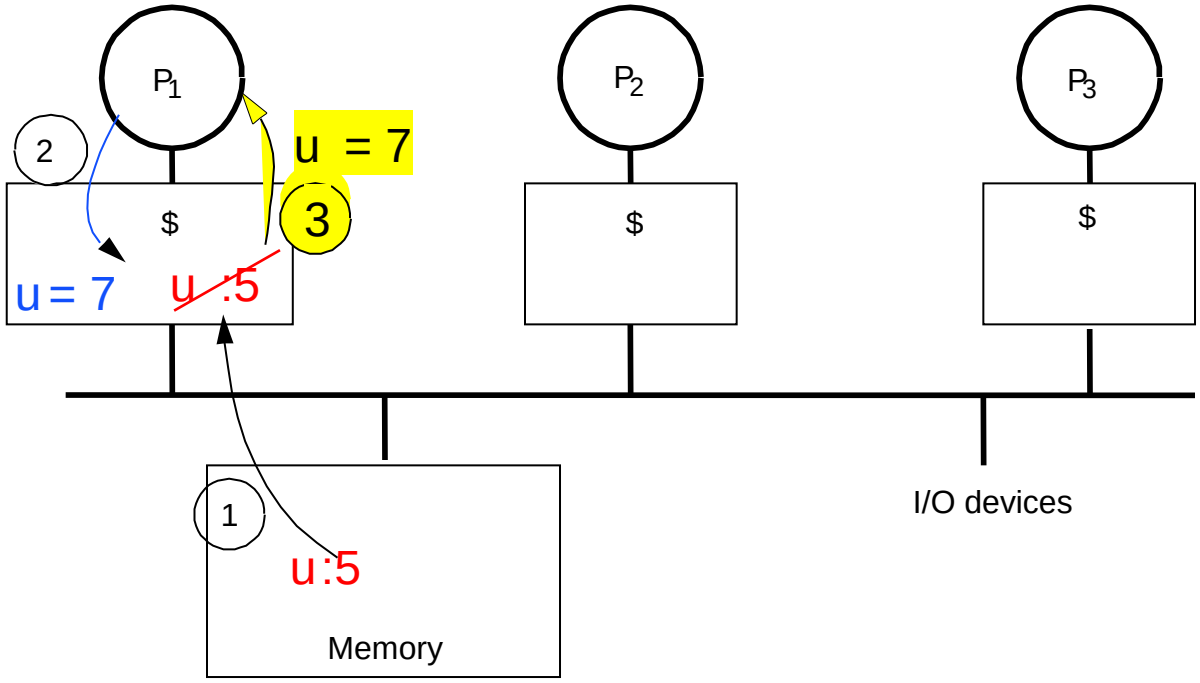
1. Coerência define os valores retornados de uma leitura
2. Consistência determina quando uma valor escrito deverá ser retornado por uma leitura

Coerência vs. Consistência

- Coerência: comportamento de leituras e escritas no mesmo endereço
- Consistência: comportamento de leituras e escritas em endereços diferentes de memória.

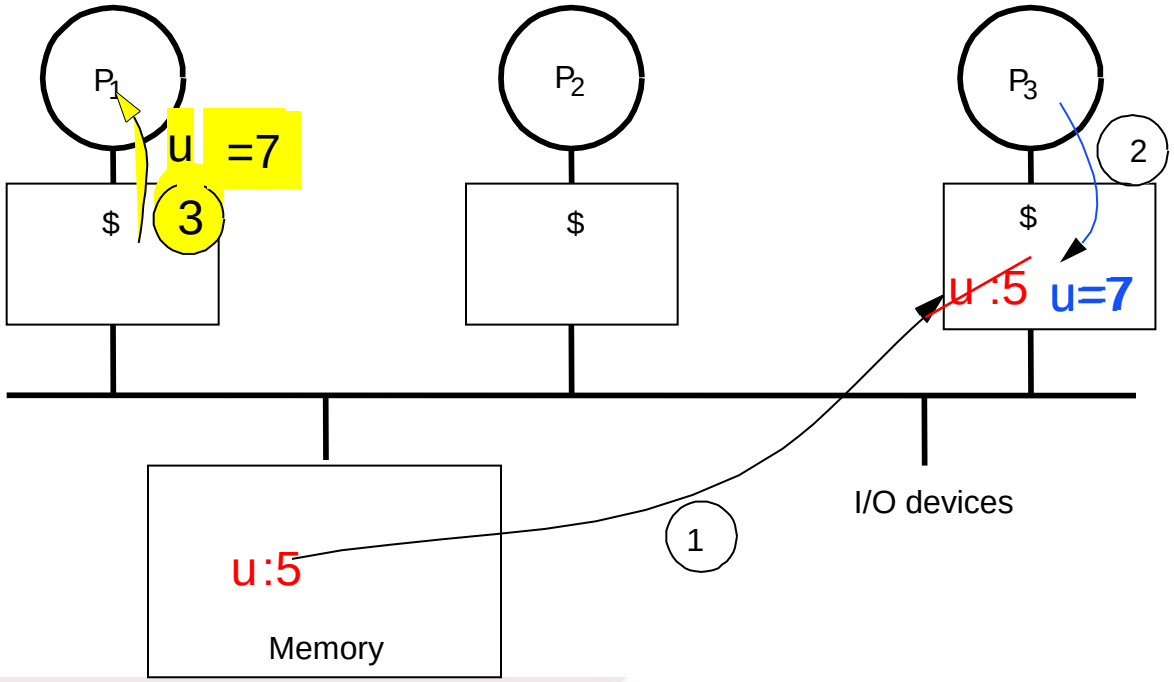
Memória Coerente

- Preservação da Ordem do Programa (escritas):
  - Uma leitura por processador P a um endereço X que segue uma escrita por P em X, com nenhuma escrita de X por qualquer outro processador ocorrendo entre a escrita e a leitura por P, sempre retorna o valor escrito por P



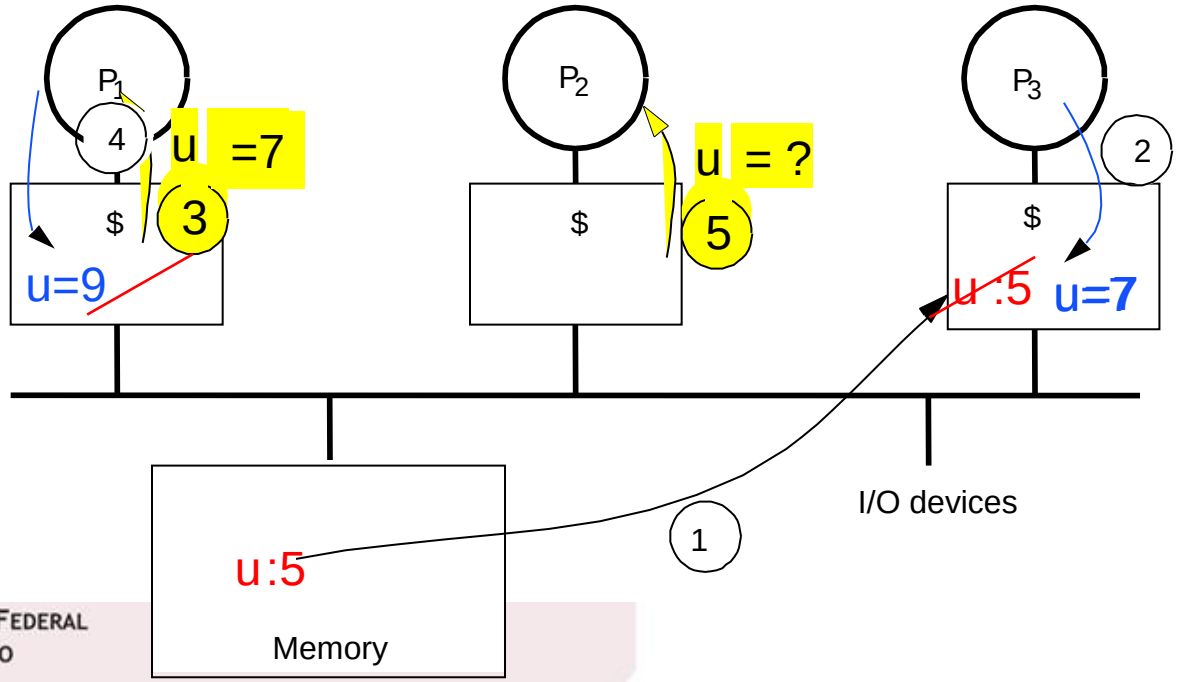
Memória Coerente

- Visão Coerente da Memória: Uma leitura por um processador a um endereço X que acontece após uma escrita por outro processador no endereço X retorna o valor escrito se a leitura e escrita forem suficientemente separadas no tempo e nenhuma outra escrita a X ocorre entre os dois acessos.



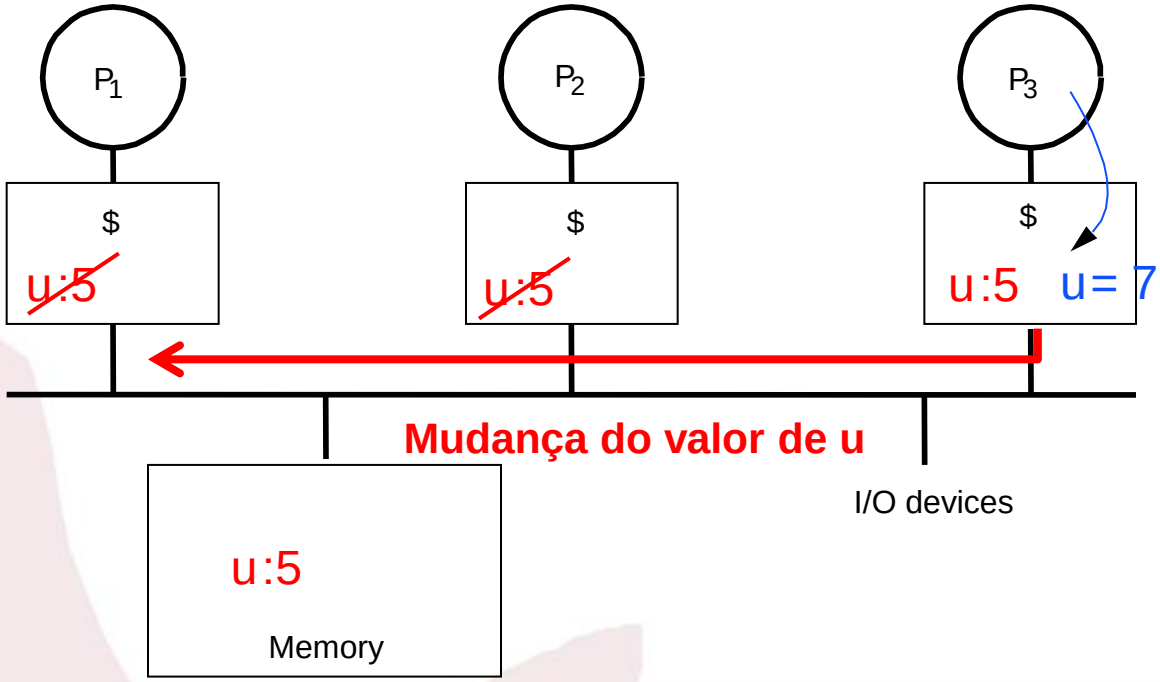
Memória Coerente

- Serialização das Escritas : 2 escritas no mesmo endereço por quaisquer 2 processadores devem ser vistas na mesma ordem por todos os processadores
  - Por exemplo, se os valores 1 e 2 são escritos em um endereço, processadores nunca podem ler o valor como 2 e então ler o valor 1



Consistência de Escrita

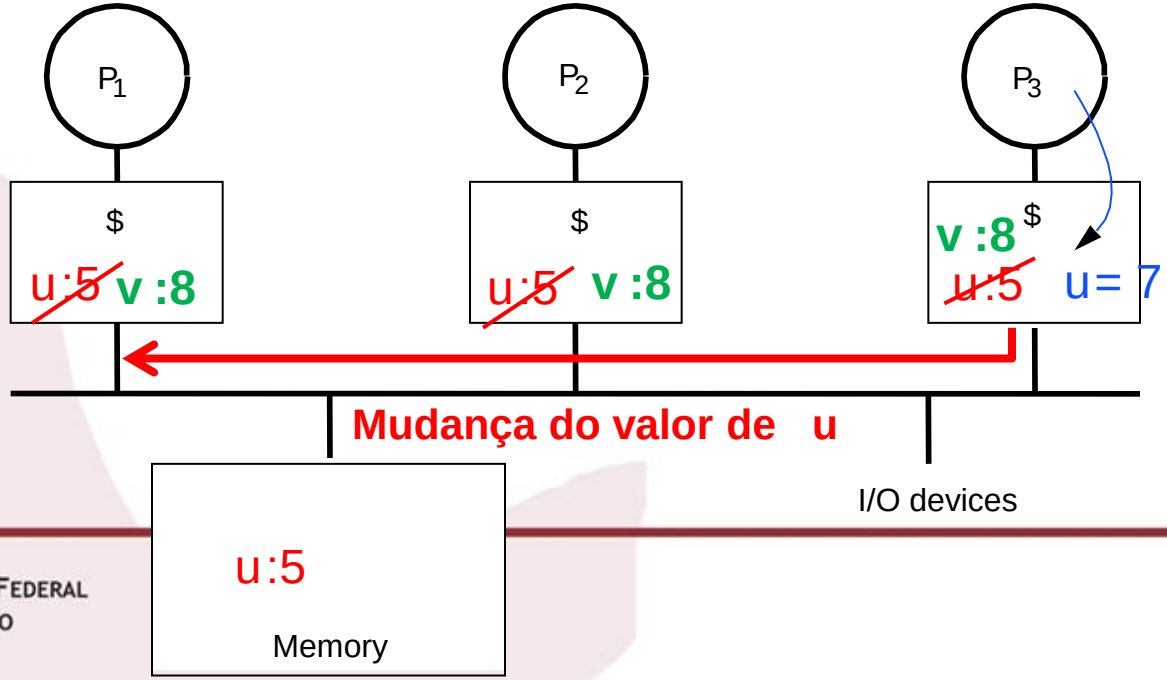
- Uma escrita não termina (não se poder permitir outra escrita) até que todos os processadores estejam “cientes” dos efeitos daquela escrita.



Consistência de Escrita

- O processador não deve mudar a ordem de qualquer escrita com respeito a qualquer outro acesso à memória.

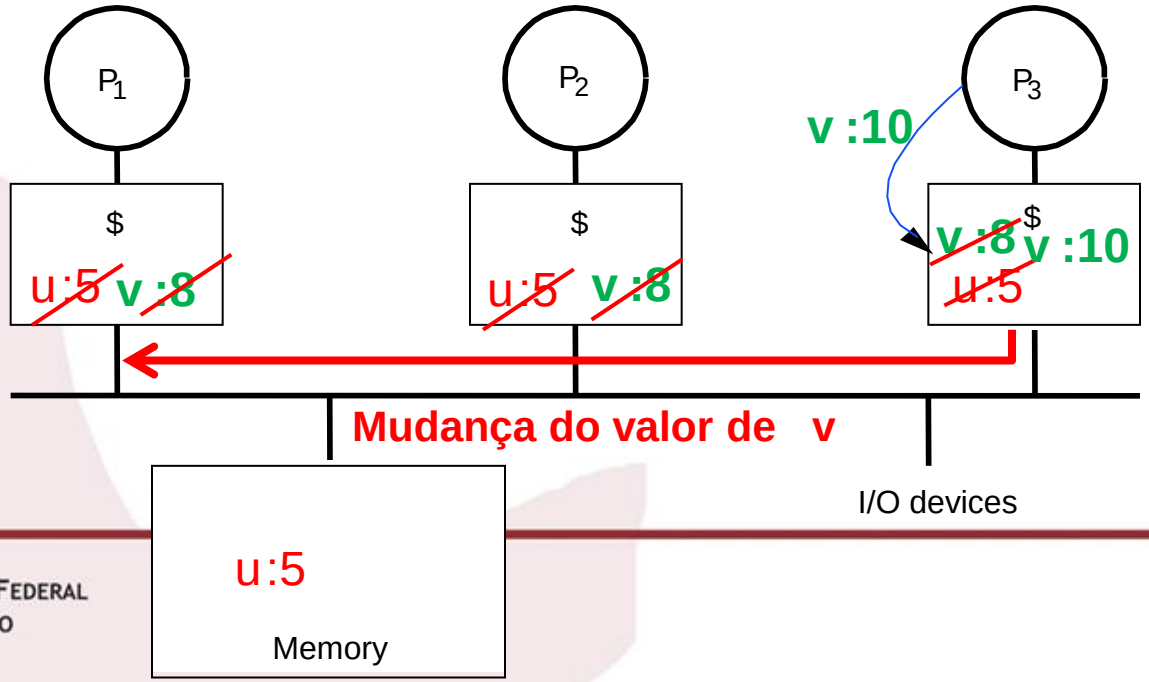
⇒ Se um processador escreve no endereço A seguido pela escrita no endereço B, qualquer processador que veja o novo valor em B deve também ver o novo valor em A.



Consistência de Escrita

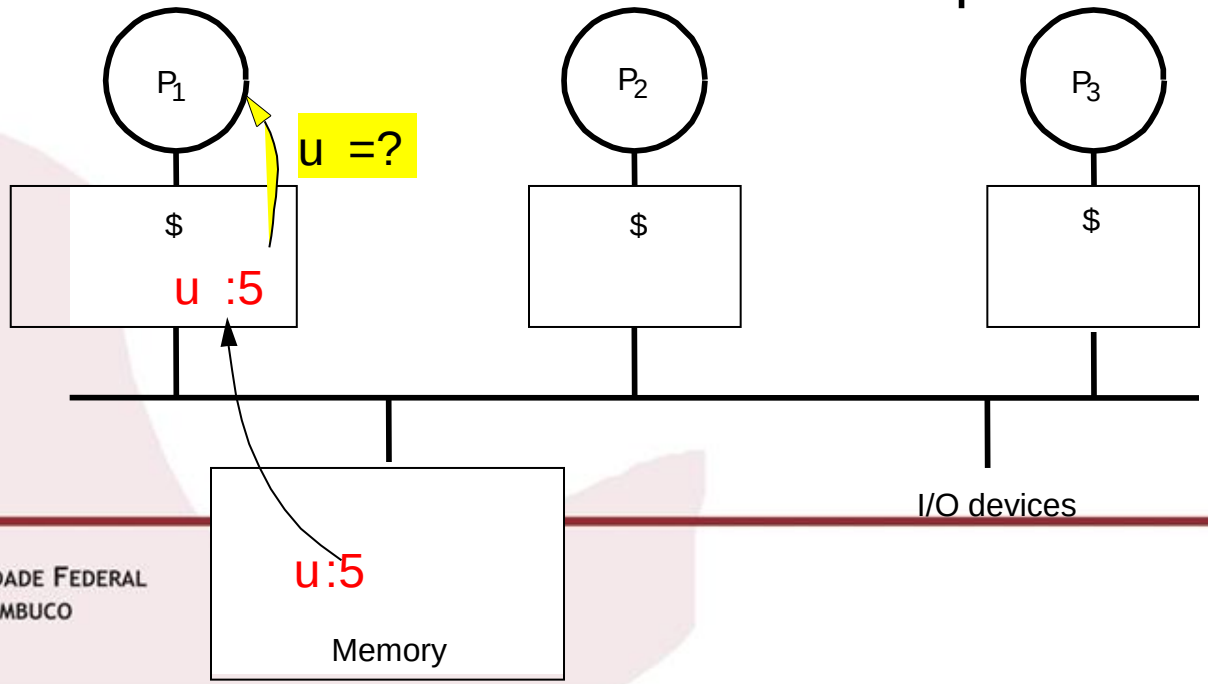
- O processador não deve mudar a ordem de qualquer escrita com respeito a qualquer outro acesso à memória.

⇒ Se um processador escreve no endereço A seguido pela escrita no endereço B, qualquer processador que veja o novo valor em B deve também ver o novo valor em A.



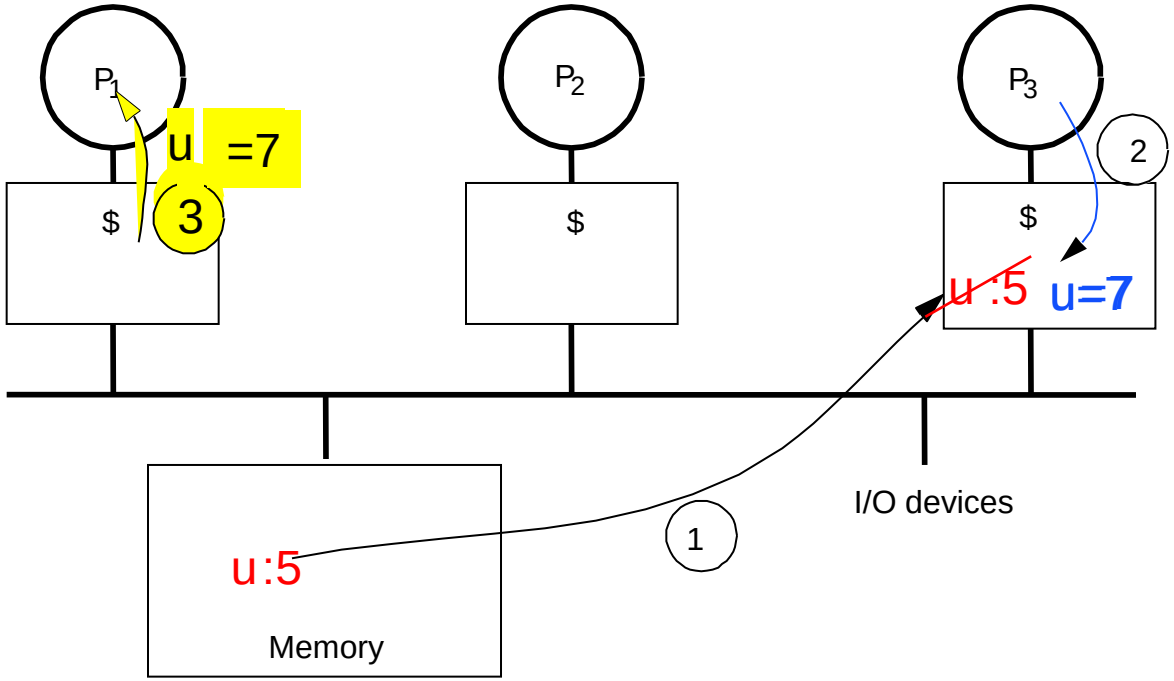
Garantindo a Coerência

- Como caches suportam o compartilhamento com coerência?
- Replicação – para dados compartilhados que estão na memória são feitas cópias nas caches que estão usando o dado
  - Reduz latência de acesso e utilização simultânea do barramento na leitura de dado compartilhado.



Garantindo a Coerência

- Como caches suportam o compartilhamento com coerência?
- Migração – o valor mais atualizado da variável é movido entre a cache que tem o valor mais atual e a cache que precisa do dado
  - Reduz a latência devido a leitura dos dados na memória



Garantindo a Coerência

- Como manter coerência de dados migrados ou replicados?
- Protocolos implementados pelo controlador da cache e/ou pelo controlador de memória que permitem o rastreamento do status de compartilhamento.

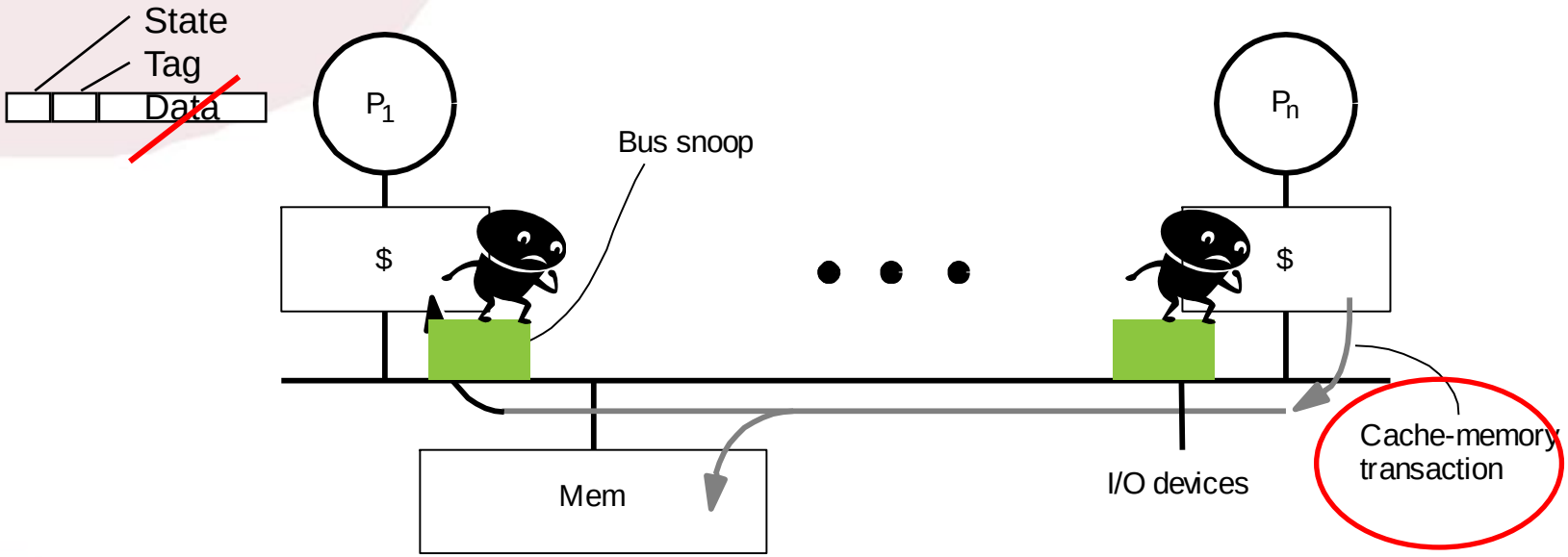
Protocolos de Coerência de Cache

1. Directory based — O status do compartilhamento de um bloco da memória física é armazenado em um local, diretório
2. Snooping — Cada cache que possui cópia de um dado também tem uma cópia do status de compartilhamento,
   - Todas as caches são acessíveis através de meio barramento
   - Todos os controladores de caches monitoram ou snoop o meio para determinar se eles possuem ou não uma cópia sendo requisitada pelo barramento

Protocolos de Coerência de Cache

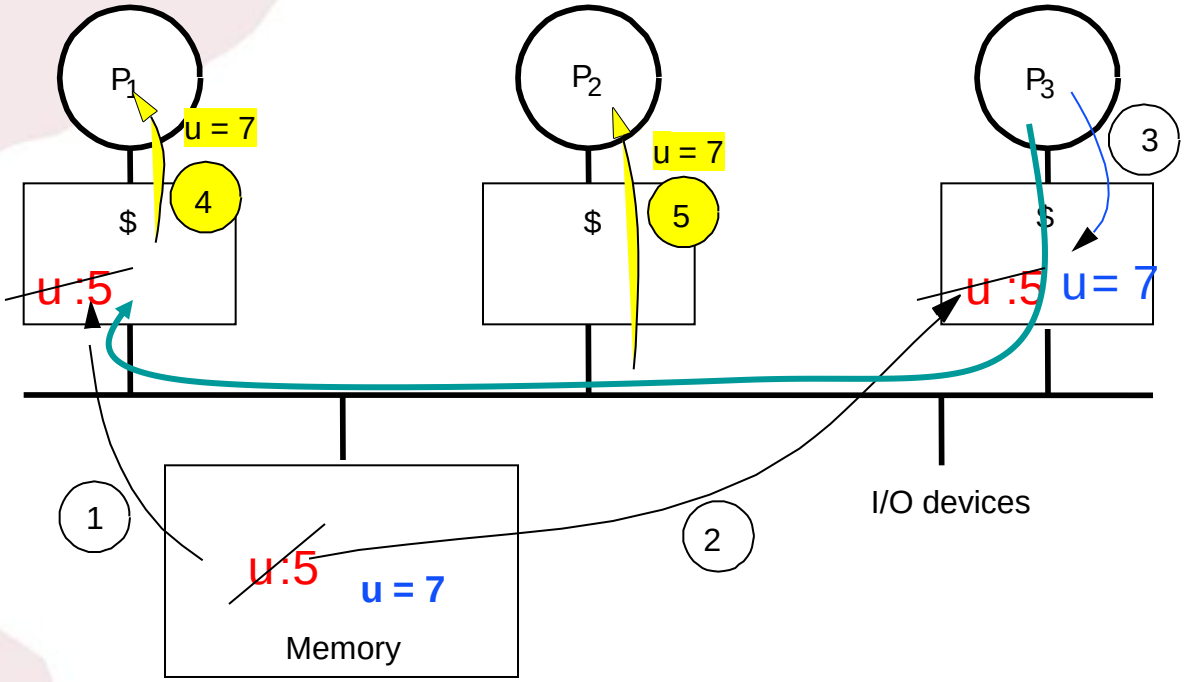
- Write Invalidate
  - As cópias nas demais caches são Invalidadas na ocorrência de uma escrita em uma das caches
- Write Update
  - As cópias das demais caches são atualizadas após a ocorrência de uma escrita em uma das caches
  - Write update causam maior utilização do barramento
- **Multiprocessadores atuais usam write invalidate**

Protocolo Snooping de Coerência de Cache



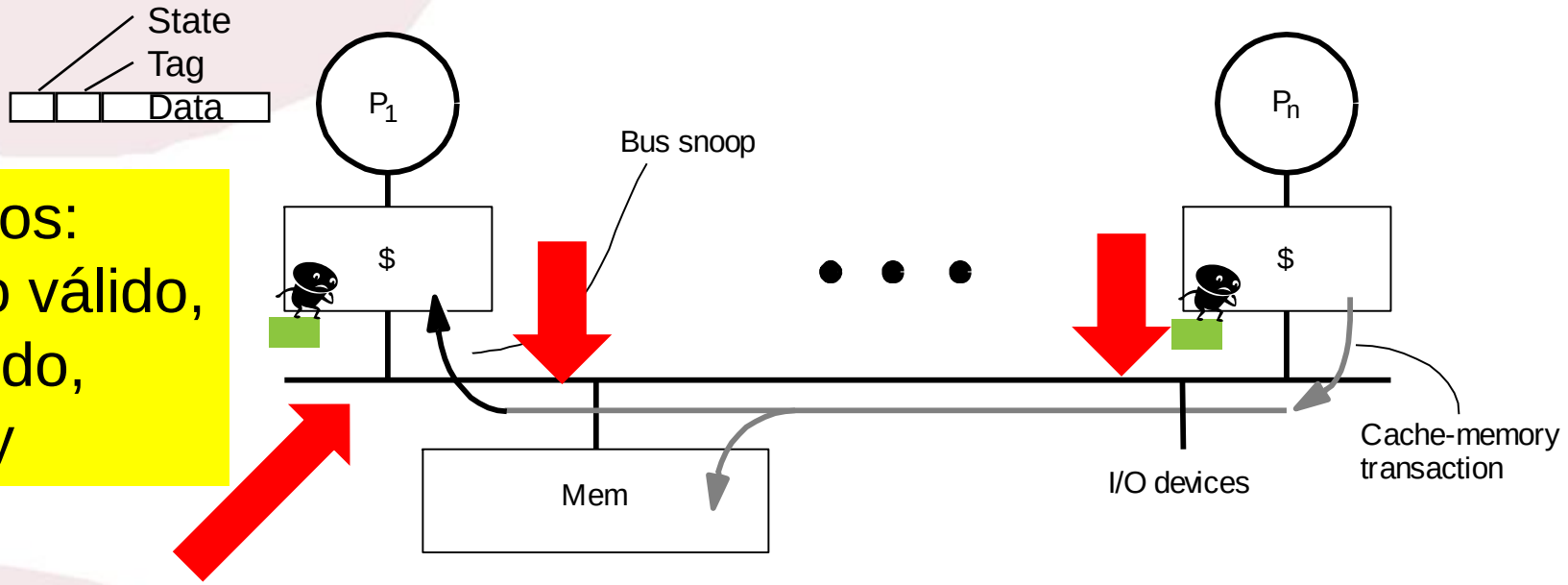
- Controladores de Cache "snoops" todas as transações no barramento
  - Transações relevantes : envolvem bloco que a cache possui
  - Realiza ação para garantir coerência
    - invalida, atualiza, ou fornece valor
  - Atualiza estado de compartilhamento do bloco de cache

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 $\Rightarrow$ Necessita saber se existem outras cópias do em outras caches
  - Não existem cópias $\Rightarrow$ Não necessita colocar invalidate no barramento
  - Se existem cópias $\Rightarrow$ 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 $\Rightarrow$ 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

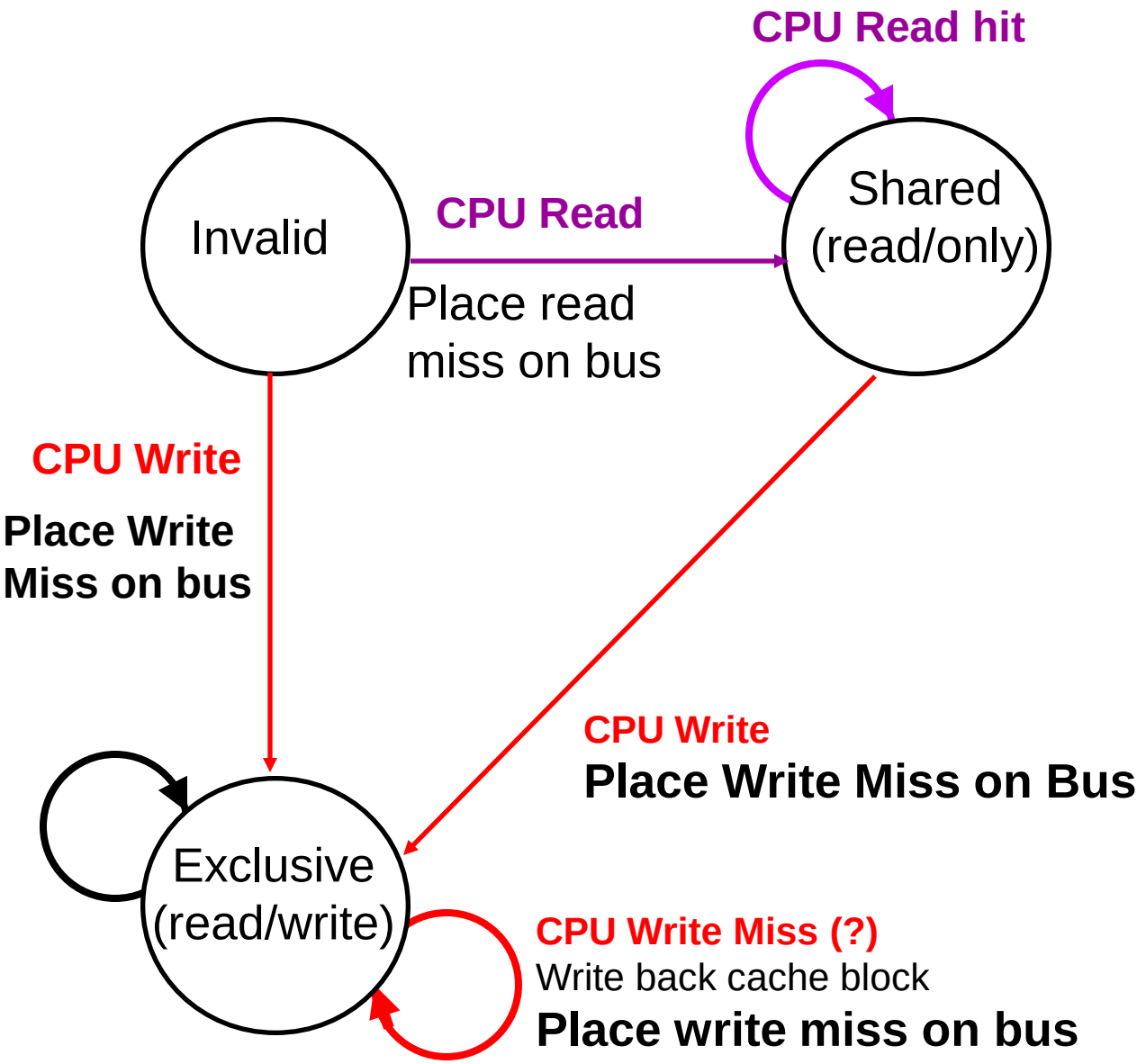
- Cada bloco de cache vai estar em UM dos estados:
  - Shared : bloco pode ser lido
  - OU Modified/Exclusive : cache tem somente uma cópia que pode ser escrita e dirty
  - OU Invalid : bloco não contem dado válido
- CPU solicita leitura:
- Se cache não tem cópia:
  - Controlador coloca Read Miss no 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

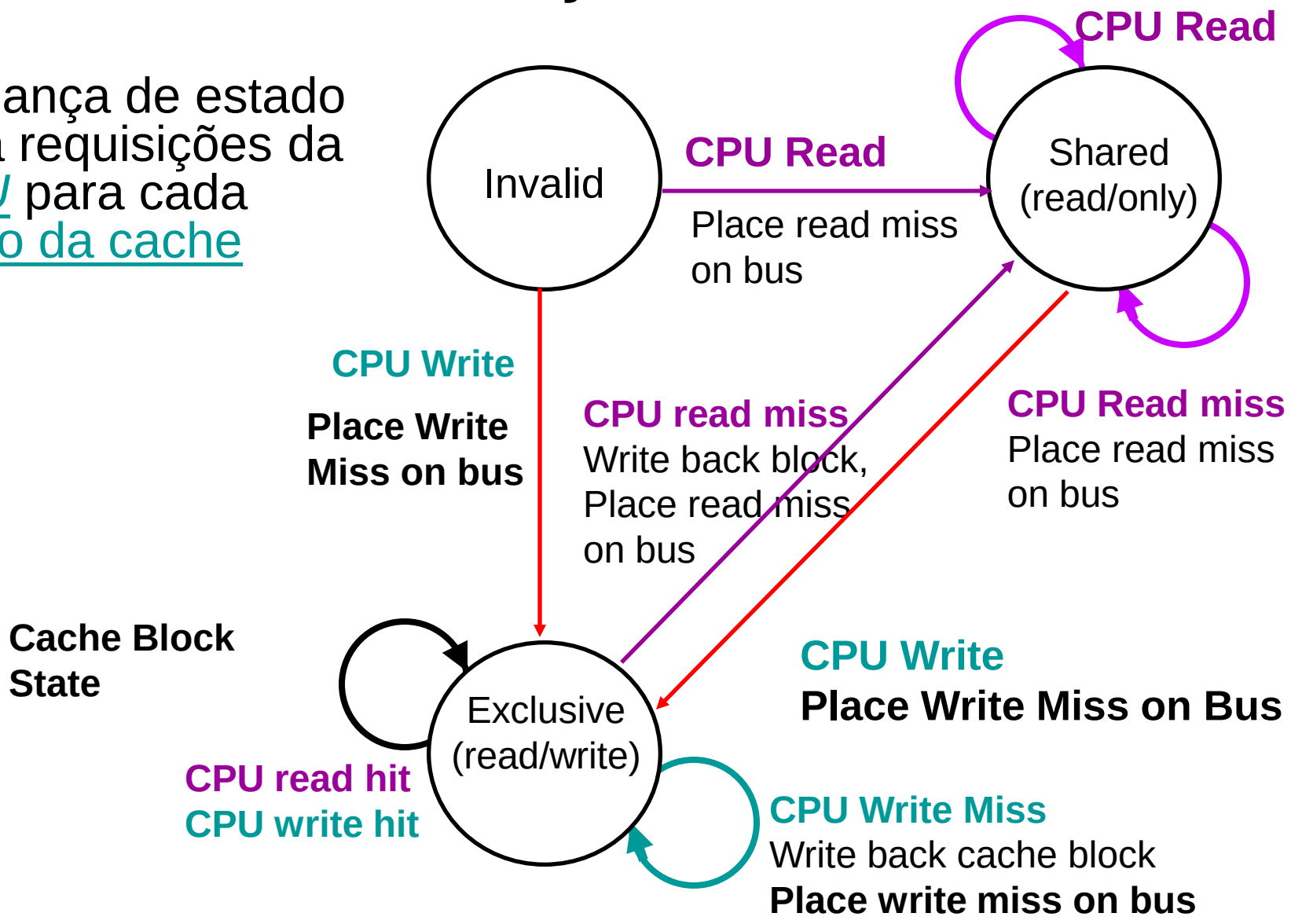
- Mudança de estado considerando requisições da *CPU* para cada bloco de cache
- Blocos que não estão na cache não são validos



Snooping: Write-Back
Substituição de Bloco

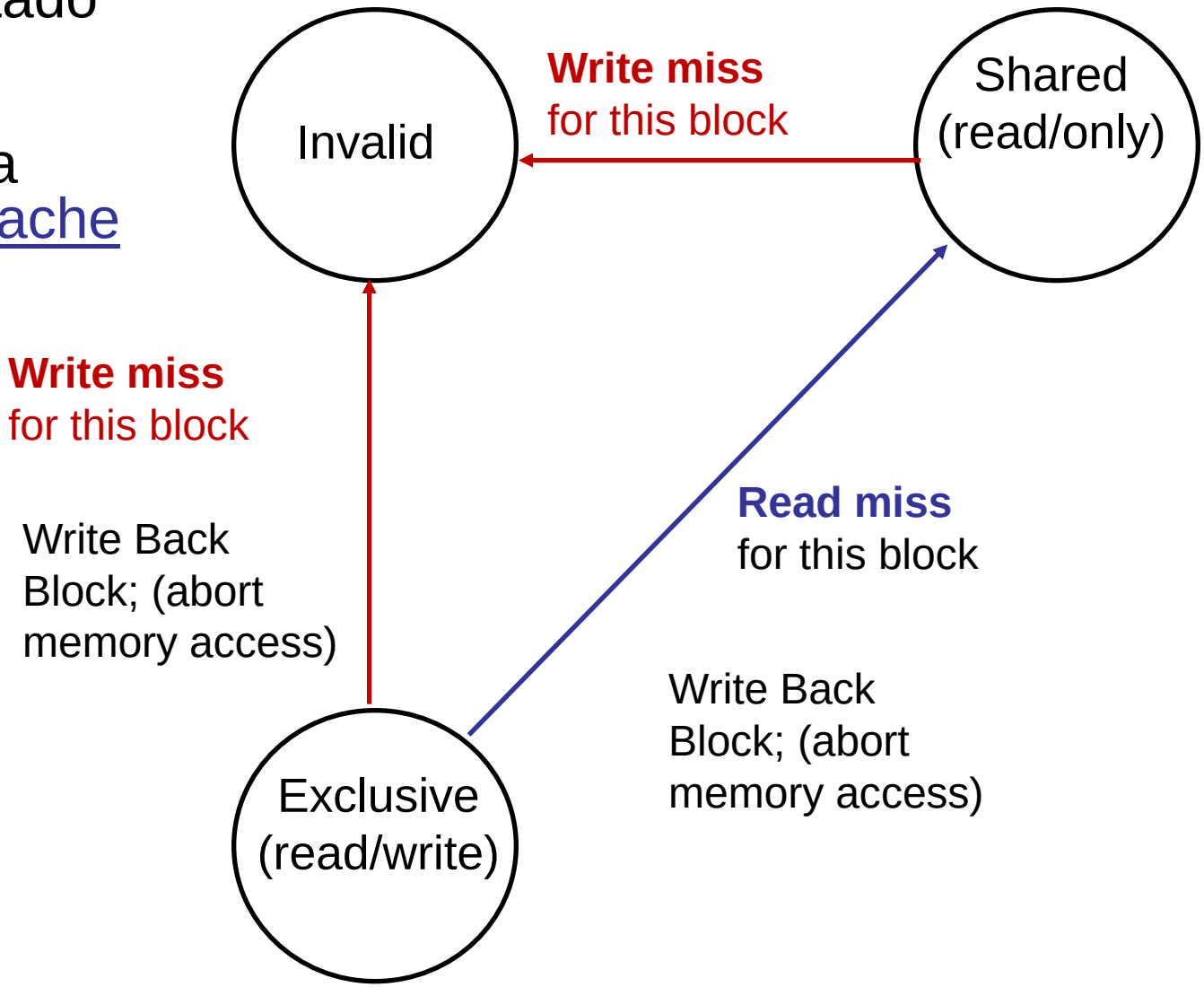
Centro de Informática

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



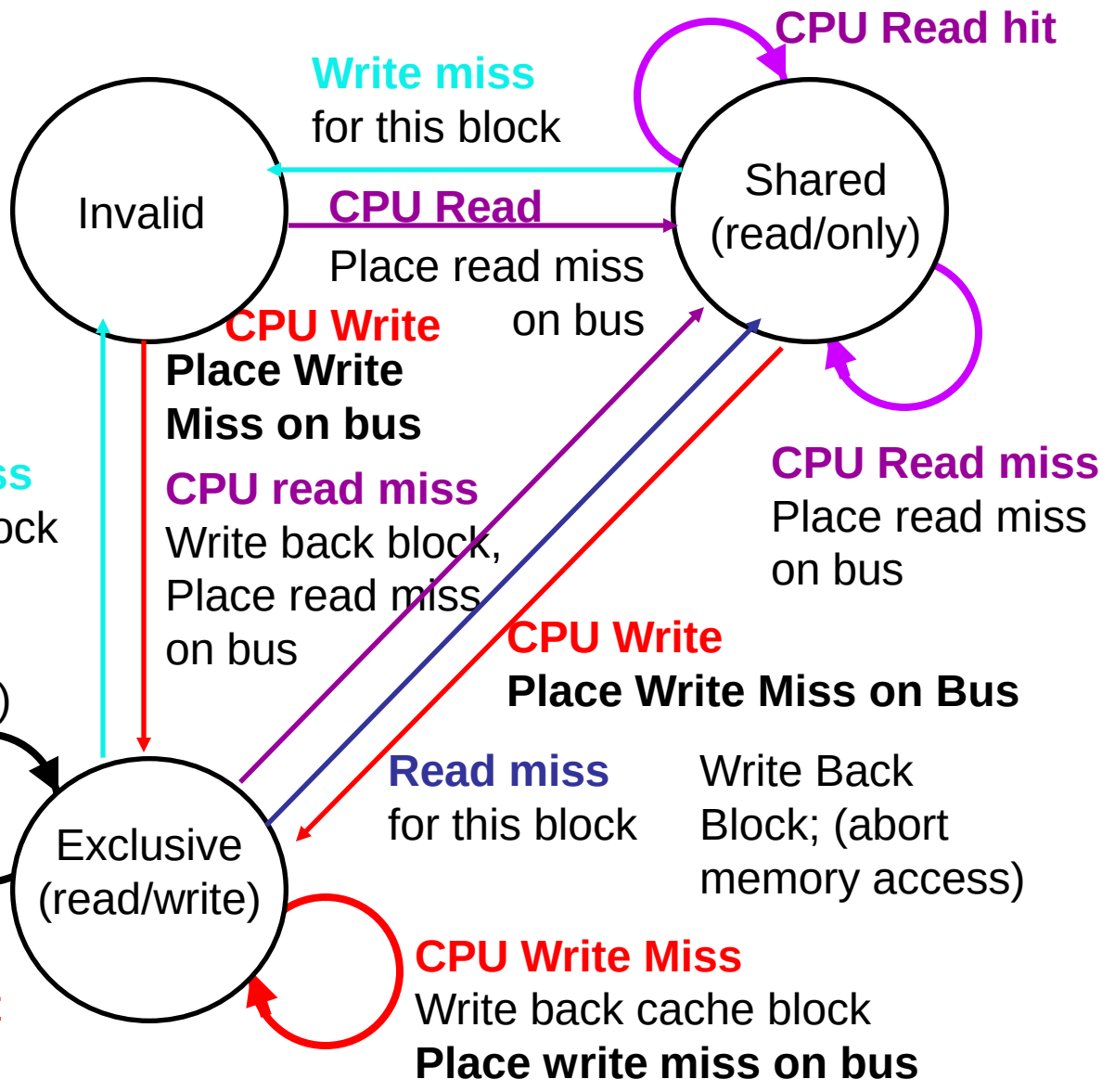
Snooping: Write-Back - Bus

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



Snooping Write-back

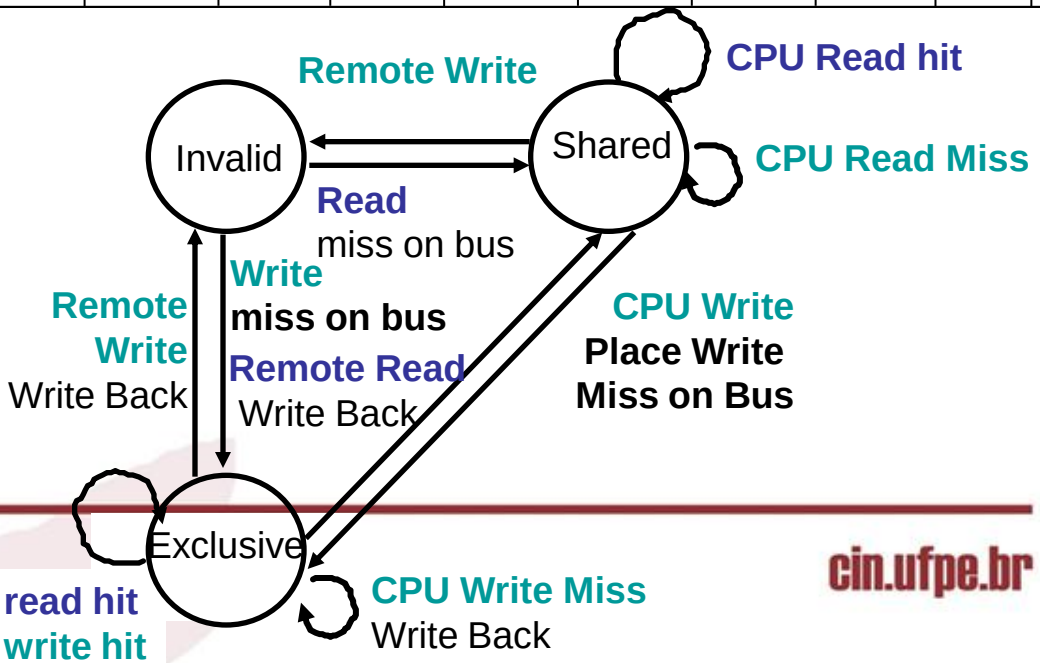
Mudança de estado para requisições da *CPU* para cada bloco da cache **e** para requisições do *barramento* para cada bloco de ache



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

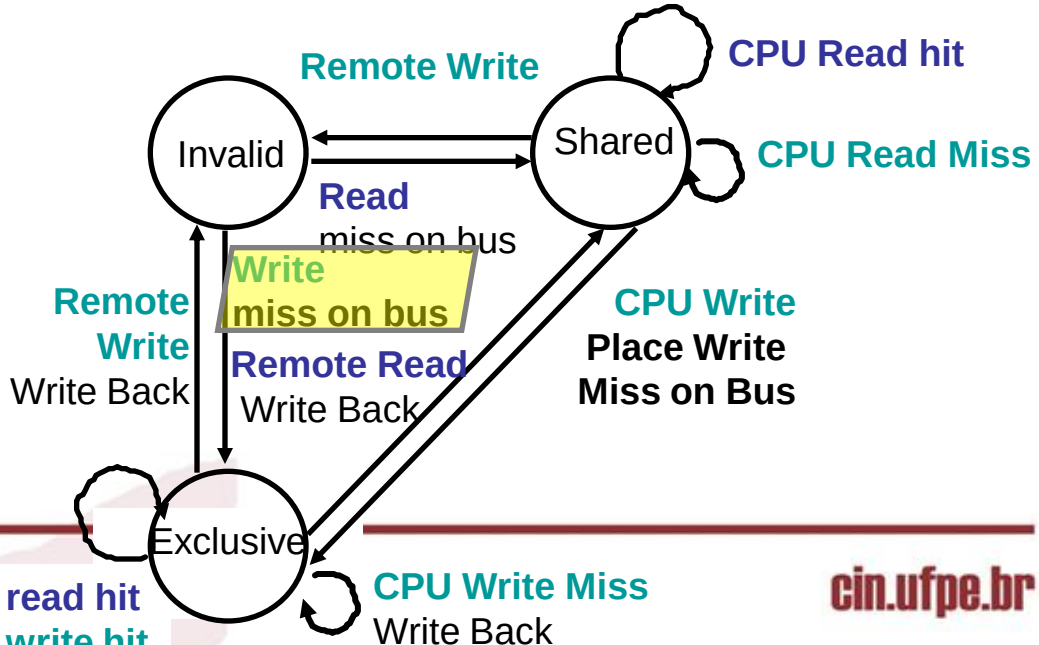


Exemplo: Passo 1

	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

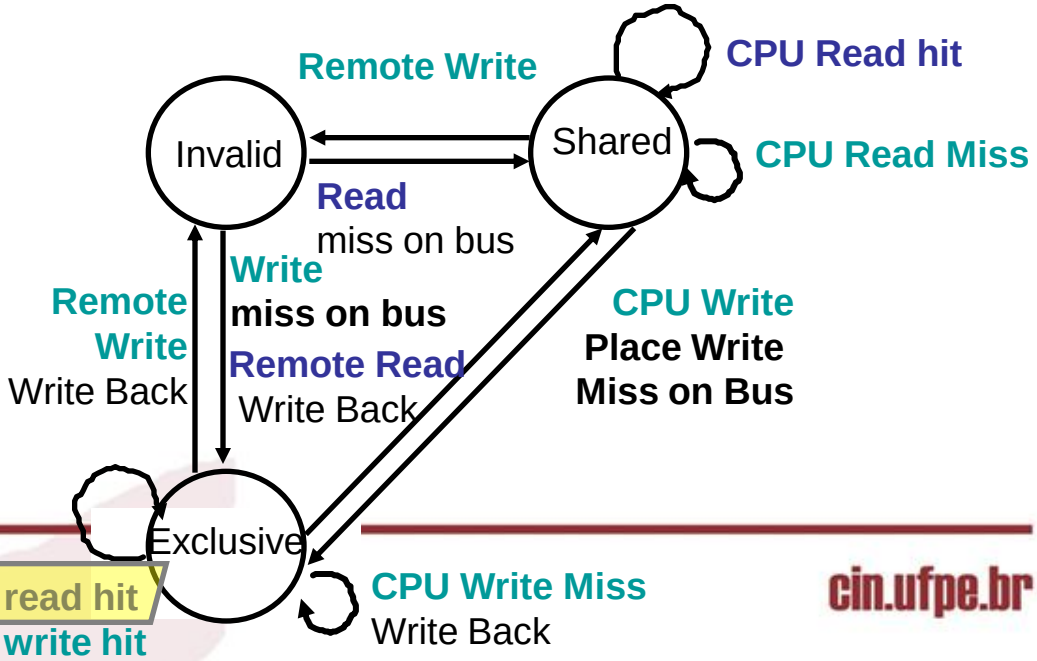
Estado ativo:



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												

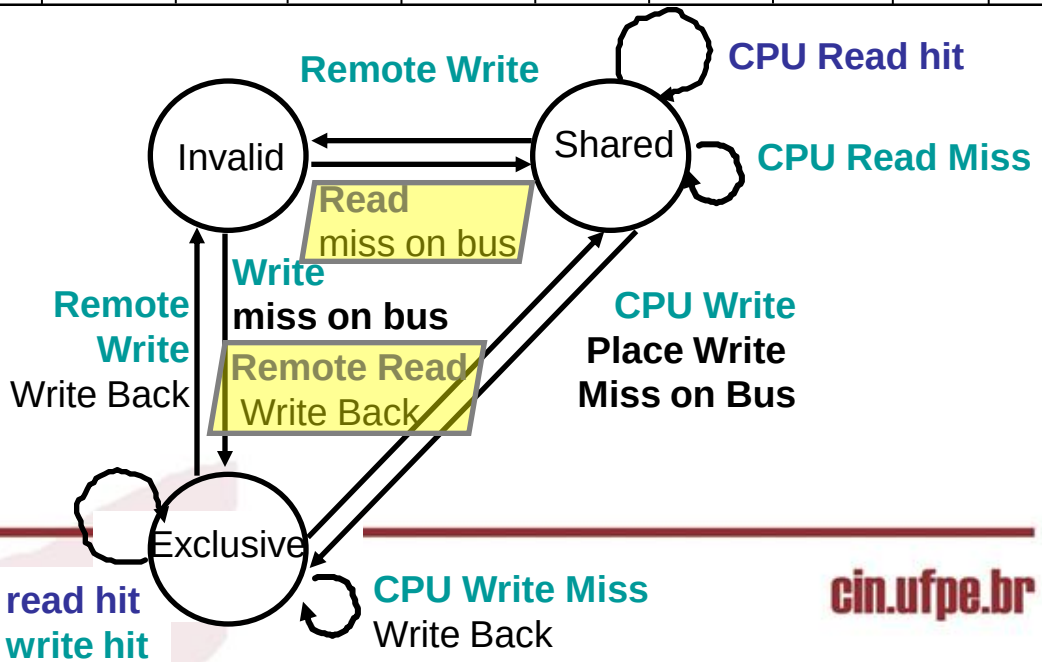
- Assuma que estado inicial da cache é “não válido”
- A1 e A2 mapeiam para o mesmo slot de cache mas A1 != A2



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

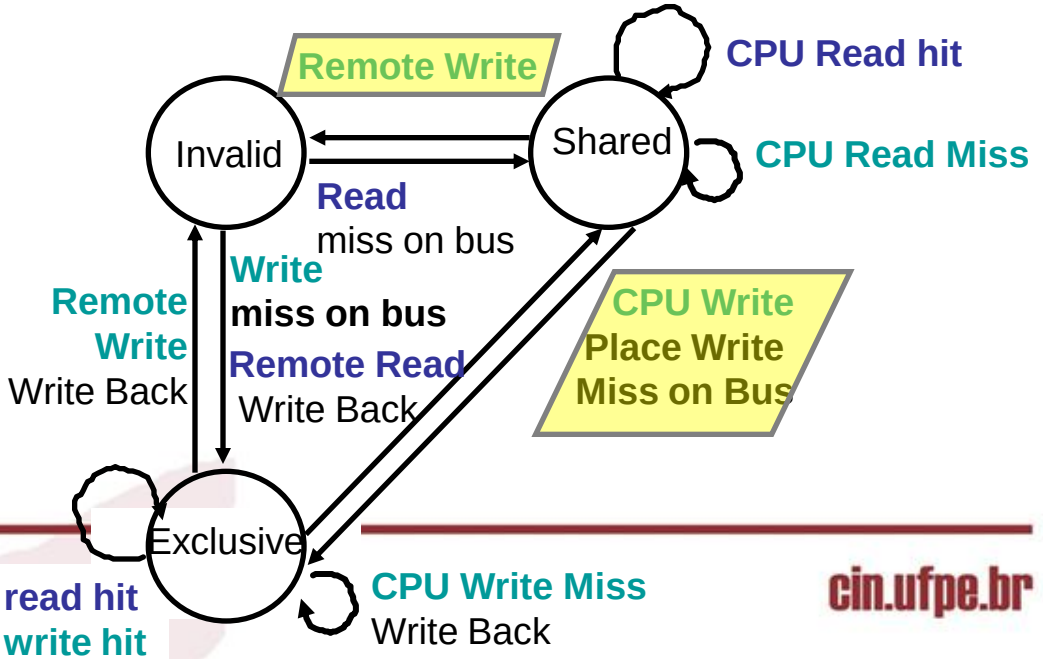
- Assuma que estado inicial da cache é “não válido”
- A1 e A2 mapeiam para o mesmo slot de cache mas A1 != A2



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

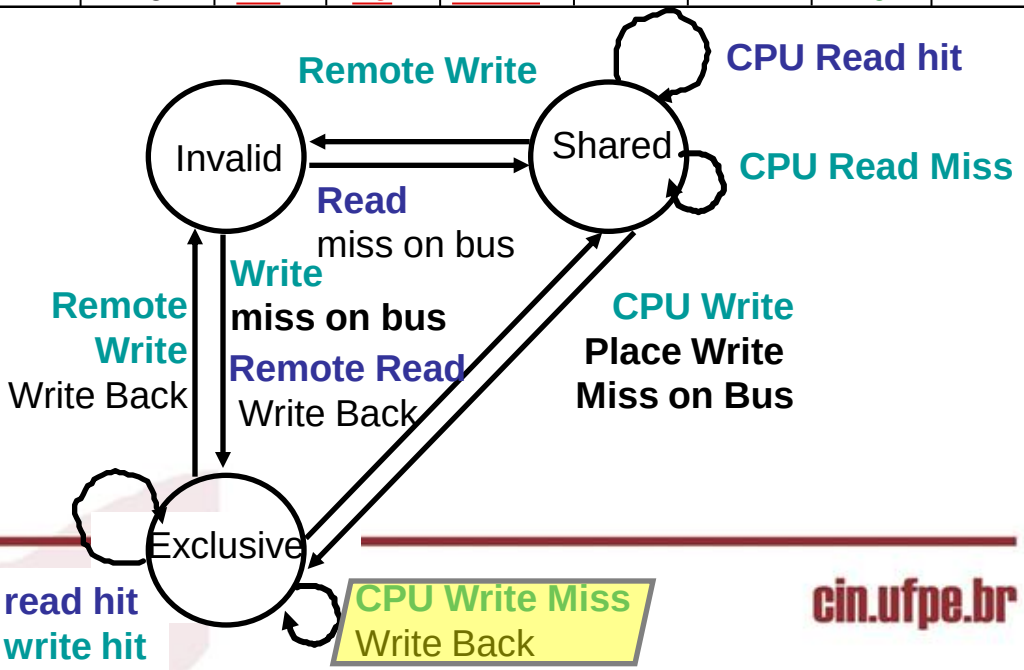
- Assuma que estado inicial da cache é “não válido”
- A1 e A2 mapeiam para o mesmo slot de cache mas A1 != A2



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									
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

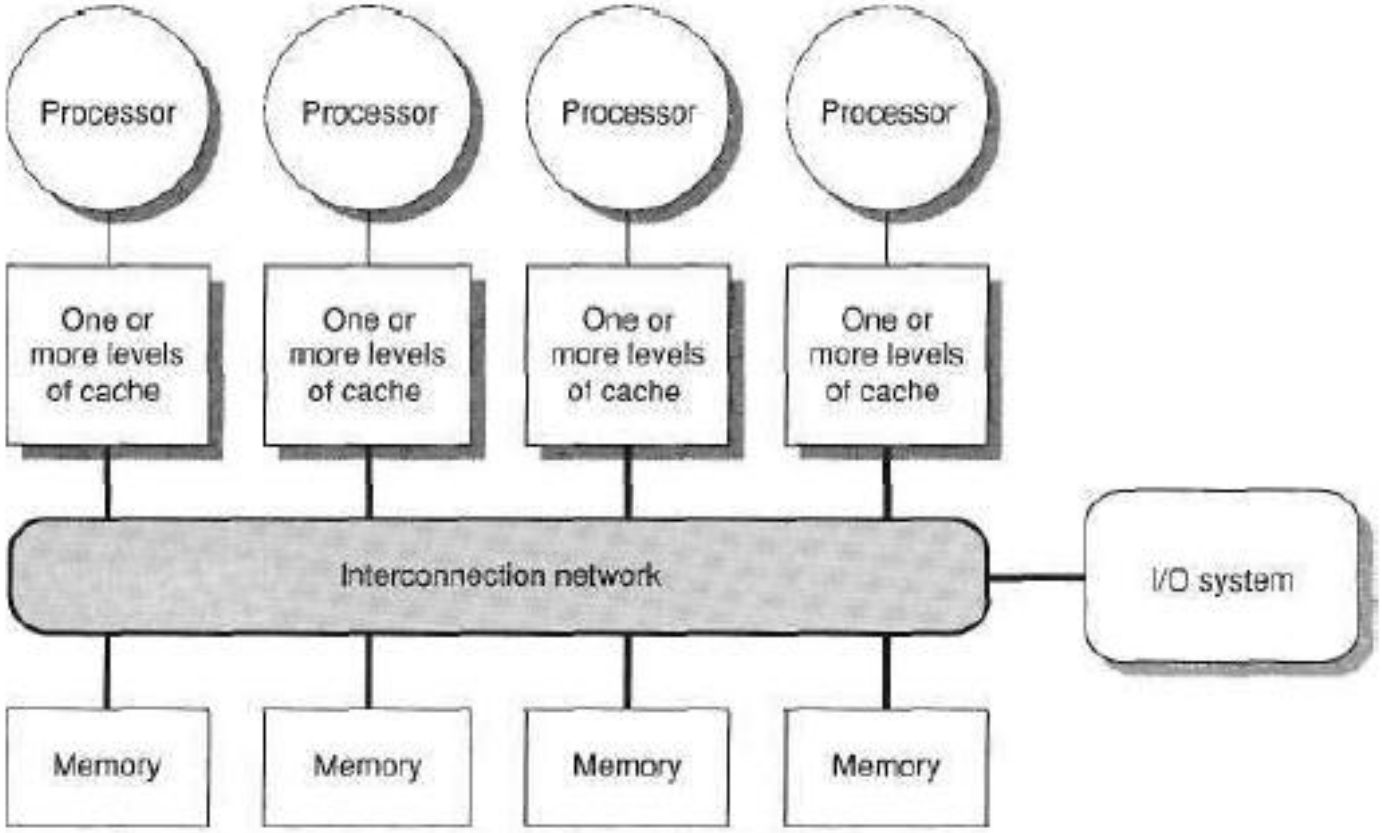
- Controlador de cache deve implementar novos comandos para realizar coerência
- Controladores de cache deve monitorar continuamente o barramento (endereços)
  - Se tag do endereço coincidir com algum bloco de cache deve invalidar (no caso de escrita)
- Esta checagem contínua dos tags na cache com os 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?

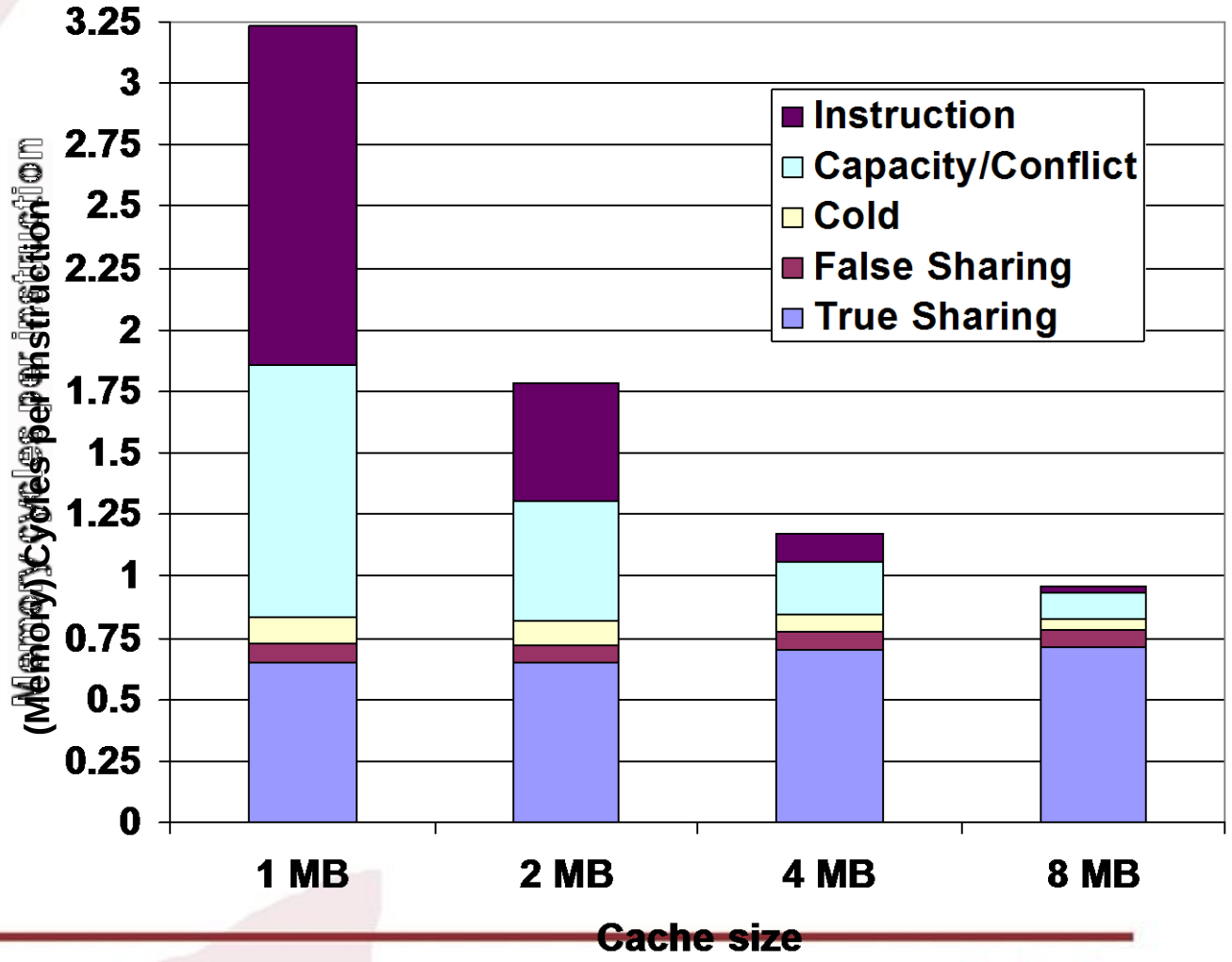
- **Assuma que x1 e x2 estão no mesmo bloco de cache**
- **P1 e P2 possuem cópias de x1 e x2.**

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

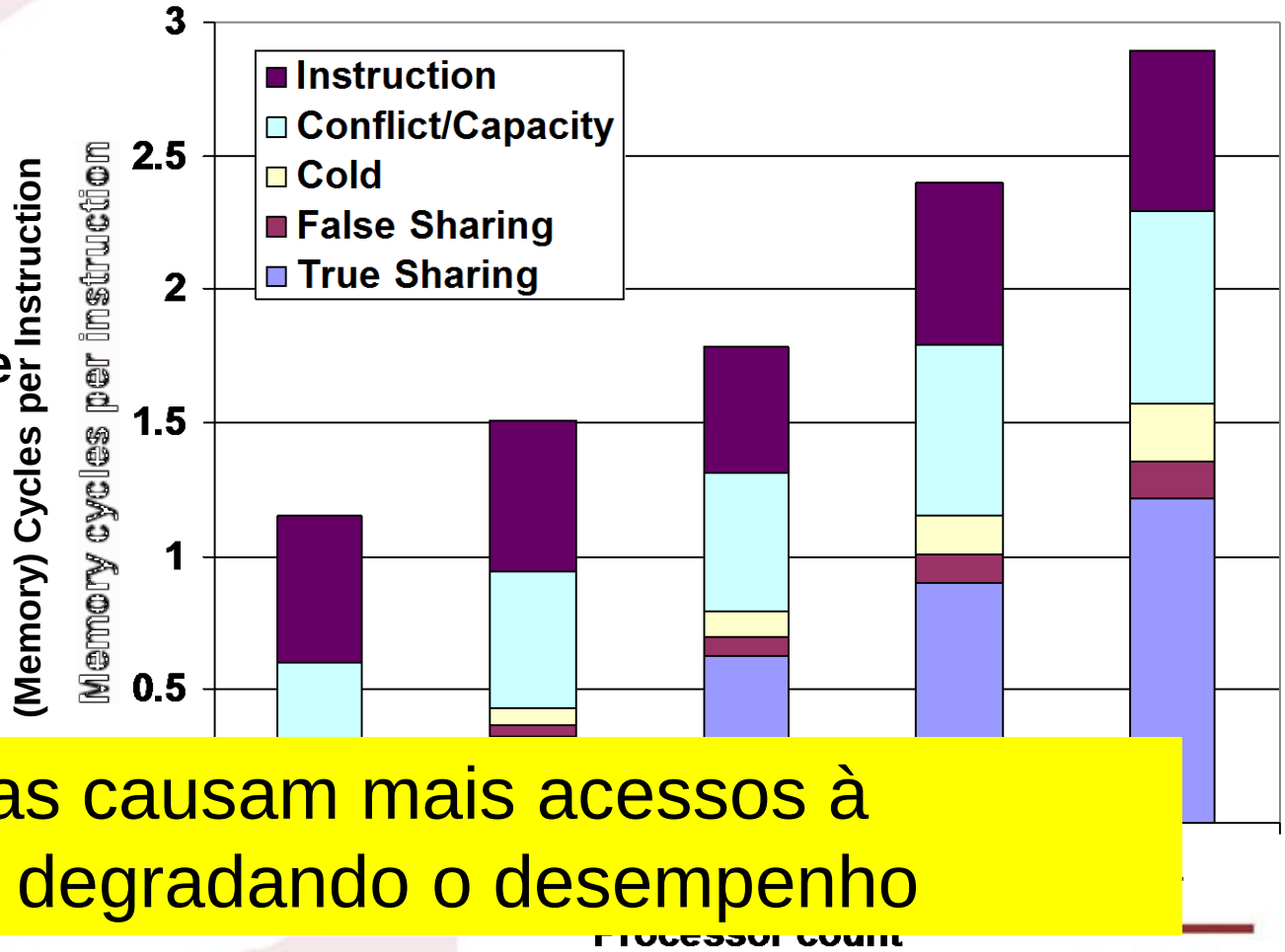
- **True sharing e false sharing não mudam com aumento da cache**
  - **1 MB para 8 MB (L3 cache)**
- **Faltas da aplicação diminuem com aumento da cache**
  - **(Instruction, Capacity/Conflict, Compulsory)**



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

Centro de Informática U·F·P·E

Como conseguir desempenho com maior número de processadores?

Conclusões

- Multiprocessadores Simétricos de Memória Compartilhada
- Processadores multi-core
- Interconexão através de barramentos
- Redução do custo de comunicação para Variáveis Compartilhadas
- Manutenção de cópias das variáveis nas caches de cada processador

Conclusões

- Problema da coerência e consistência
  - Coerência: valores retornados por uma leitura
  - Consistência: quando um valor escrito será retornado por uma leitura
- Multiprocessadores baseados em barramento
  - Meio compartilhado serializa as escritas $\Rightarrow$ Consistência de Escrita
  - Coerência garantida pelo Protocolo de Snooping
- Protocolo de Snooping
- Limitações dos SMPs