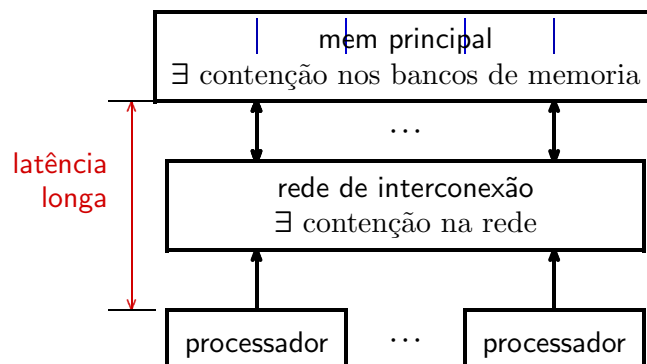


## Processamento Paralelo & Multiprocessadores

- **Motivação**
- **Tipos de máquinas paralelas**
- Coerência entre caches

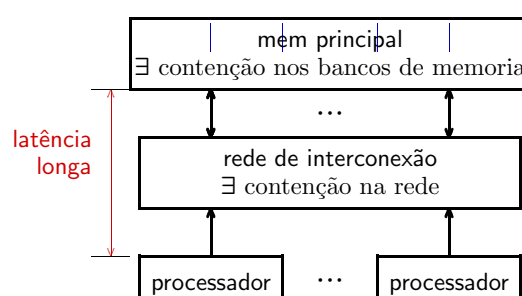
### UMA – Uniform Memory Access

- latência no acesso à memória é a mesma para todos processadores  
*mas pode ser longa*
- latências crescem com tamanho do sistema  
*aumentar escala é difícil*



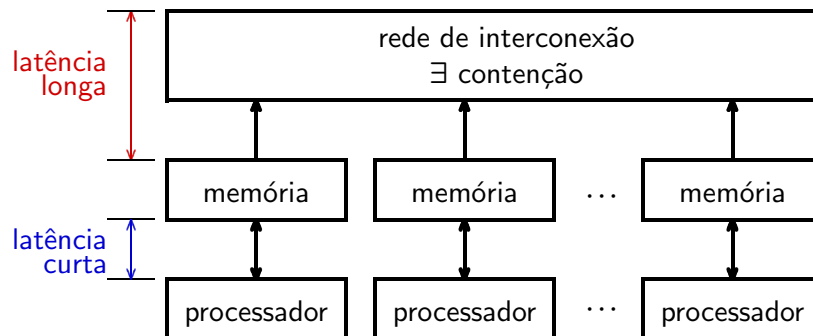
### UMA – Uniform Memory Access

- Localização dos dados é gerenciada automaticamente
- contenção limita vazão → na rede e na memória
- geralmente usa caches
- usado em pequenos multiprocessadores = MPs simétricos  
*Symmetric MultiProcessors ou SMPs*



## NUMA – Non-Uniform Memory Access

- Latência pequena no acesso à **memória local**
- latência grande no acesso à **memória remota**  $P \rightarrow M_{loc} \rightarrow R \rightarrow M_{rem}$
- vazão para memória local pode ser mais alta que para remota
- contenção na rede e no acesso à memória  $P_{loc} \times (N-1)P_{rem}$

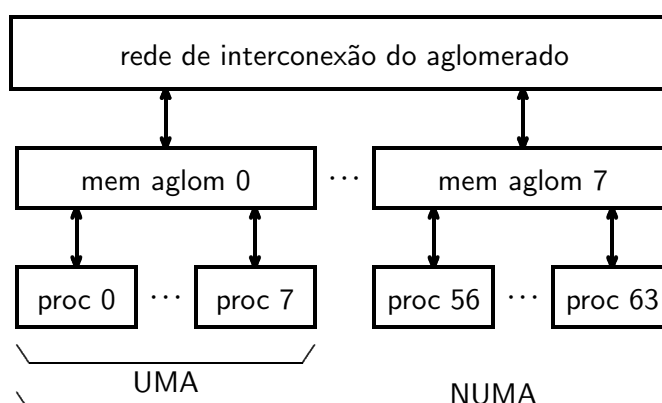


## Multiprocessadores NUMA Non-Uniform Memory Access

- Memória **logicamente compartilhada** mas **fisicamente distribuída**
    - ▷ um espaço de endereçamento lógico
    - ▷ pode ser tratado como memória compartilhada
  - desempenho depende fortemente da localização dos dados
- 
- Multicomputadores
    - ▷ cada processador tem espaço de endereçamento privativo
    - ▷ comunicação através de troca de mensagens (explícita)

## Aglomerados (clusters)

- Nós **UMA pequenos** num **sistema NUMA grande**
- híbrido? aglomerado de aglomerados?



## Processamento Paralelo & Multiprocessadores

- Motivação
- Tipos de máquinas paralelas
- **Coerência entre caches**

### Execução atômica

Dois processadores executam em paralelo:

<i>/* P1 */</i>	<i>/* P2 */</i>
c = a = 0;	d = b = 0;
b = b + 1;	a = a + 1;
c = a + b;	d = c + b;
print c;	print d;

Quais os valores de “print c” e “print d”?

Comandos ‘atômicos’ em C  
**não são**  
executados atômicamente  
pelo processador

a = a + 1;  $\equiv$  lw r1,0(r2)  
addi r1,r1,1  
sw r1,0(r2)  
a++; a+=1;

### Sincronização e Atomicidade

- Operações atômicas **devem ser** serializadas pelos mecanismos de escrita em memória
- Problemas decorrentes de disputas:
  - ▷ **latência** sem contenção operações demoradas
  - ▷ **serialização** se há contenção latência + tempo na fila

→ **dificultam aumentar escala para sistemas maiores**

**Barramento** é um meio de comunicação compartilhado

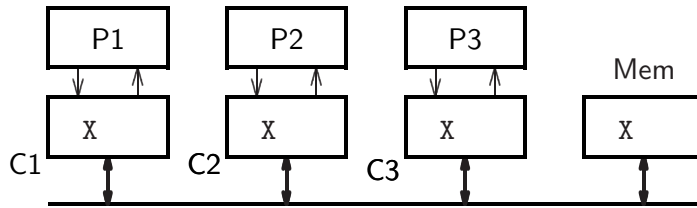
→ comunicação por **difusão**

*broadcast*

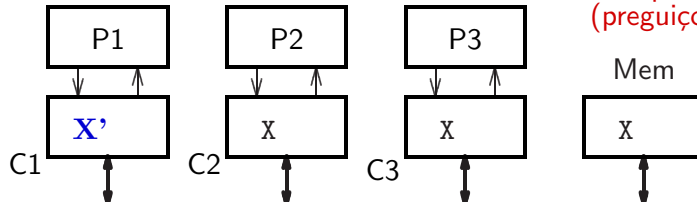
↪ comunicação é serializada através do barramento

## Coerência de caches

Inicialmente, três cópias idênticas de X em C1, C2 e C3



Processador P1 atualiza cópia para X'



X em C2 e C3 caducam com  $\forall$  política de escrita (preguiçosa/forçada)

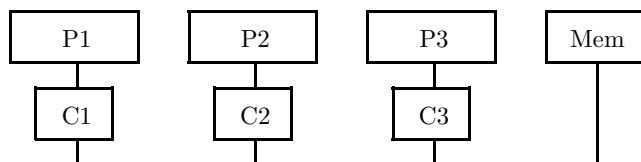
## Coerência de caches

P1, P2 e P3 carregam X em suas caches;  
 P1 atualiza sua versão;  
 C2 e C3 ficam com sua cópia desatualizada.  
 → isso ocorre **mesmo** que as caches usem  
 escrita forçada (ou escrita preguiçosa)

Informalmente:

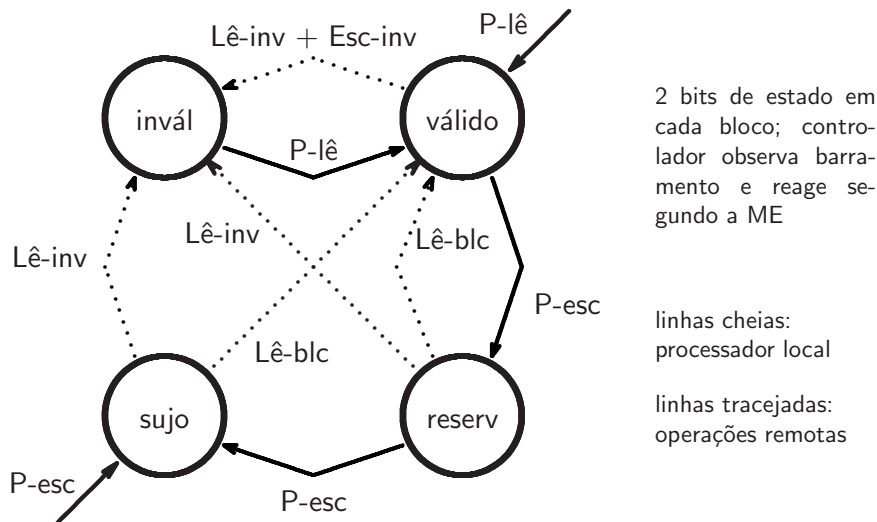
**coerência entre caches** é um método para garantir que os acessos à memória sejam coerentes, apesar das caches.

## Métodos de Coerência para Barramentos (i)



- As caches usam escrita preguiçosa geralmente
  - ▷ porque estas minimizam tráfego no barramento
- Os blocos nas caches podem estar num destes estados típicos
  - ▷ **INVÁLIDO**: conteúdo do bloco não pode ser usado
  - ▷ **VÁLIDO**: não está sujo, compartilhado (há  $\geq 1$  cópia)
  - ▷ **SUJO**: única cópia suja
  - ▷ **RESERVADO**: não-sujo, única cópia (aumenta eficiência)

## Métodos de Coerência para Barramentos (ii)

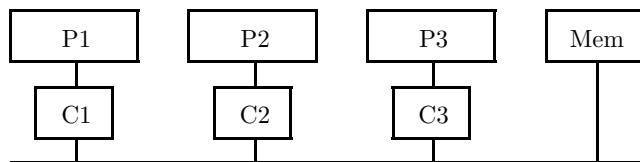


## Métodos de Coerência para Barramentos (iii)

- **Espionagem** (*snooping*)
  - ▷ todas as caches observam **todo** o tráfego no barramento
  - ▷ etiquetas com duas portas: CPU e barramento
  - ▷ ações de um controlador de cache são visíveis pelos demais CCs  
 ~> **difusão no barramento**
- O que ocorre nas escritas?

**invalidar** cópias nas outras caches: **protocolos de invalidação**

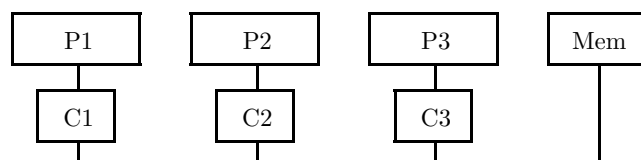
**atualizar** cópias nas outras caches: **protocolos de atualização**



## Protocolo de Invalidação

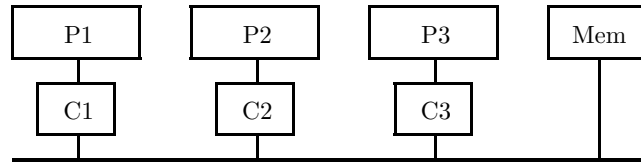
atividade no processador	atividade no barramento	cache C1	cache C2	memória ender X
				0
P1 lê X	falta em X	0		0
P2 lê X	falta em X	0	0	0
P1 faz X=1	invalidação para X	1	—	0
P2 lê X	falta em X *	1	1	1

\* P1 responde à falta em C2 e atualiza memória



## Protocolo de Atualização

atividade no processador	atividade no barramento	cache C1	cache C2	memória ender X
				0
P1 lê X	falta em X	0		0
P2 lê X	falta em X	0	0	0
P1 faz X=1	broadcast de X=1	1	1	1
P2 lê X	acerto em X	1	1	1



Há mais tráfego no barramento, se P2 não usar X novamente.

## Protocolo de Invalidação Simplificado

- Caches usam escrita preguiçosa
- Estados
  - ▷ **INVÁLIDO**: conteúdo do bloco não pode ser usado
  - ▷ **COMPART**-ilhado: não está sujo, compartilhado (há  $\geq 1$  cópia)
  - ▷ **EXCLUSIVO**: única cópia suja
- a cada transação no barramento, o controlador de cache:
  - ▷ verifica se bloco está na cache
  - ▷ se estiver na cache, efetua mudança de estado cfe ME (adiante)

## Desempenho de Protocolos para Barramentos

- Tipos de faltas nas caches:
  - ▷ capacidade (mais significativa)
  - ▷ conflitos
  - ▷ compulsórias
  - ▷ **coerência** (4º C)
- **Faltas por coerência**
  - são faltas causadas pelo protocolo de coerência
- **Falso compartilhamento**
  - ▷ a unidade de coerência é um **bloco** de cache
    - blocos grandes ( $\geq 32$  pals) para amortizar custos de comunicação e etiquetas
  - ▷ pode ocorrer que um **bloco inteiro** seja compartilhado,
    - mas palavras individuais não são compartilhadas
    - ↪ tráfego adicional desnecessário