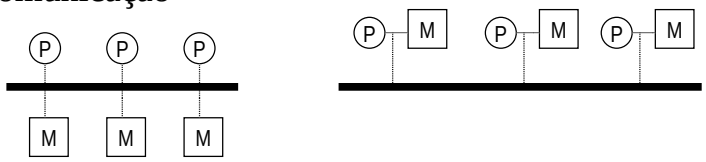
Arquiteturas Paralelas
Small-Scale Shared Memory Multiprocessors

Modelo de Comunicação

- *Shared Memory* (centralizada ou distribuída)
  - Processadores comunicam com espaço de endereçamento compartilhado
  - Fácil em máquinas *small-scale*
  - Vantagens:
    - Escolhido para uniprocessadores, MPs small-scale
    - Fácil de programar
    - Baixa latência
    - Mais fácil para usar hardware de controle de cache
- *Message passing*
  - Processadores possuem memórias privadas e comunicam-se via mensagens
  - Vantagens:
    - Menos hardware, pode ser implementado por sw
    - Escalabilidade
- HW pode suportar os dois modelos

Arquiteturas com Espaço de Endereçamento Compartilhado

- Todos os processadores podem acessar todas as posições de memória do sistema, provendo um mecanismo conveniente e rápido para comunicação



Centralizada
(small-scale)

Distribuída
(large-scale)

- Conhecidas também como *shared memory*

Modelo de Compartilhamento de Memória

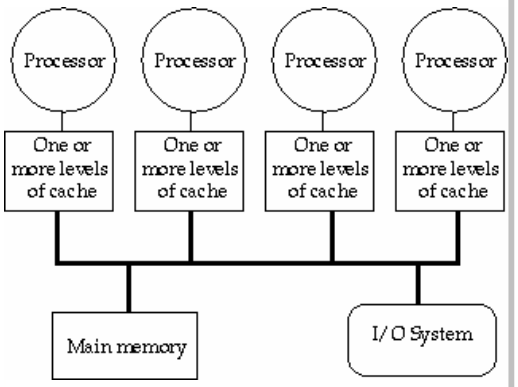
- Comunicação, compartilhamento e sincronização existem via load/store em variáveis compartilhadas
- Modelo de programação relativamente fácil (uniprocessador + sincronização)
- Desvantagem potencial e escalabilidade

Arquiteturas Baseadas em Troca de Mensagens

- Processadores só podem acessar memória local, e toda a comunicação e sincronização ocorrem via troca de mensagens explícita
- Fácil de se construir (máquinas uniprocessadoras + redes de comunicação)
- Escalabilidade é alta

Small-Scale—Shared Memory

- Caches servem para:
  - Reduzir necessidade de bandwidth alto entre processador/memória
  - Reduzir latência de acesso
  - Bom para dados privados e compartilhados
- Qual o problema de caches em MPs?



Coerência de Caches

Time	Event	Cache CPU A	Cache CPU B	Memória (X)
0				1
1	A reads (X)	1		1
2	B reads (X)	1	1	1
3	A stores 0 into X	0	1	0

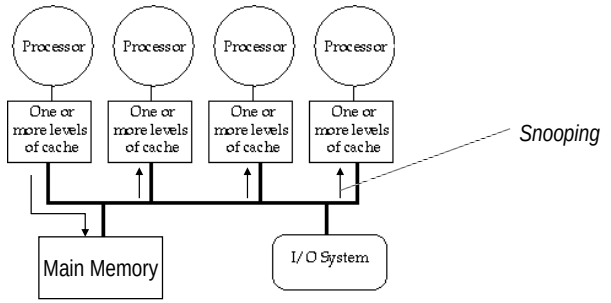
Coerência de Caches

- Sistema de memória é coerente se o dado retornado após uma leitura é o dado que foi escrito mais recentemente
  - *Coerência:* Quais valores podem ser retornados durante uma leitura
  - *Consistência:* Quando valor escrito vai ser retornado durante uma escrita

Mecanismos para Garantir Coerência de Caches

- *Snooping*: Cada cache que possui cópia própria do dado compartilhado possui também o estado do bloco, e nenhum estado centralizado é mantido.
  - Os controladores da cache ficam "espionando" o barramento compartilhado para verificar o estado atual do bloco
- *Directory based:* O estado do bloco é mantido em um diretório em uma única localização

Snoopy Caches

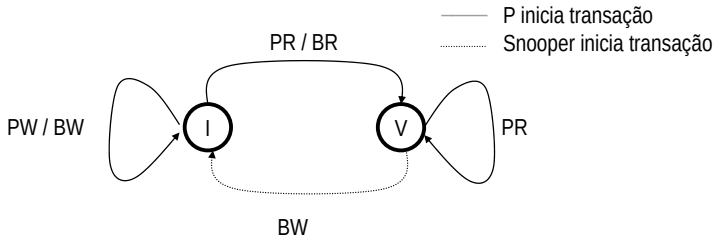


- Cada controlador "espiona" todas as transações do barramento
  - Ação depende do estado atual e do protocolo usado
- Algoritmo distribuído
  - Meio de broadcast facilita projeto, mas limita escalabilidade

Protocolos para Snoopy

- Controlador responde a eventos do processador e do barramento
  - Máquina de estados finitos
- Opções de projeto
  - Write-through vs. Write back
  - Invalidate vs. update

Write-through vs. Write back



- Write-through é simples, pois todo evento é observável
- Mas usa muito BW do barramento
- MPs baseados em barramento utilizam caches write-back

Coerência de Caches
Write Invalidate vs. Write Update

- *Write invalidate* (write back): Invalida todas as cópias existentes durante uma escrita
  - Trabalha em nível de bloco de cache
  - Escritas locais na cache
- *Write update* ou *write broadcast* (write through): Faz a atualização de todas as cópias durante uma escrita
  - Trabalha em nível de palavra para reduzir o tempo de escrita
  - Escritas em todas as caches
  - Reduz atraso entre escrita em uma cache e leitura em outra

Questão básica: Quando um bloco é escrito múltiplas vezes por um P antes de ser lido por outros

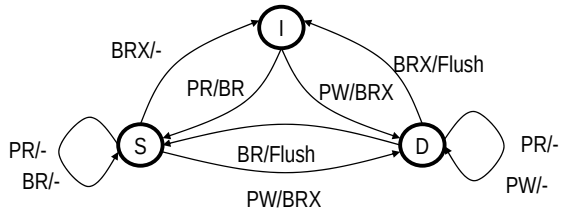
Write Invalidate

Activity Proc	Activity Bus	Cache CPU A	Cache CPU B	Memory (X)
				0
CPU A reads X	Cache miss for X	0		0
CPU B reads X	Cache miss for X	0	0	0
CPU A writes 1 to X	Invalidation for X	1		0
CPU B reads X	Cache miss for X	1	1	1

Write Update

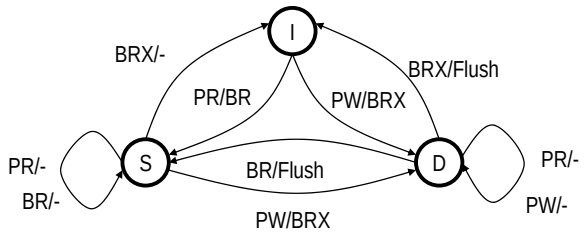
Activity Proc	Activity Bus	Cache CPU A	Cache CPU B	Memory (X)
				0
CPU A reads X	Cache miss for X	0		0
CPU B reads X	Cache miss for X	0	0	0
CPU A writes 1 to X	Write broadcast of X	1	1	1
CPU B reads X		1	1	1

Protocolo Writeback + Invalidate Básico



- Estados
  - Invalid (I), Shared (S) 1+, Dirty (D) 1
- Eventos do processador
  - PR (Processor Read), PW (Processor Write)
- Transações no barramento
  - BR (Bus Read), BRX (Bus Read + Write), BW (update memory)

Exemplo

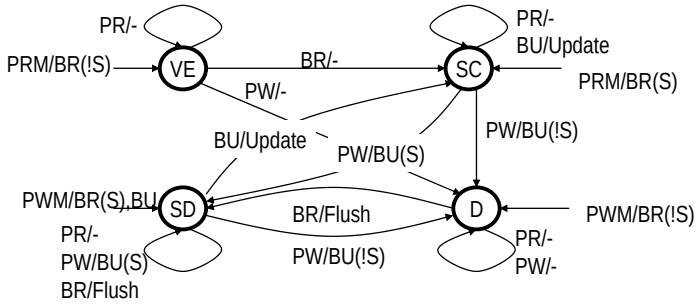


Processor Action	P1	P2	P3	Bus Action	Data Supplied By
P1 reads u	S	-	-	BR	Memory
P3 reads u	S	-	S	BR	Memory
P3 writes u	I	-	D	BRX	Memory
P1 reads u	S	-	S	BR	P3's cache

Variações do Protocolo de Invalidação

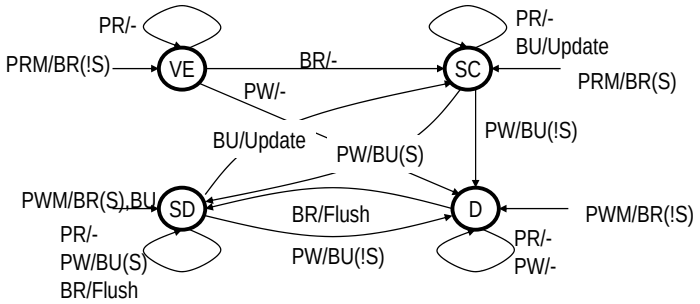
- Utiliza 4 estados para evitar transações múltiplas em R/W sem compartilhamento
- Estados
  - Invalid, Valid-exclusive (1 cópia somente), Shared (2+), Dirty
- Nova transição: I $\rightarrow$VE

Protocolo p/ Writeback + Update (Dragon)



- Estados
  - Valid exclusive (VE) [P + M], shared clean (SC) [P + O + M (talvez)], shared dirty (SD) [P + O + !M], dirty (D) [P]
- Novos eventos do processador
  - PRM (Processor Read Miss) e PWM (Processor Write Miss)
- Eventos do Barramento
  - R (read), W (write) e U (update)

Exemplo



Processor Action	P1	P2	P3	Bus Action	Data Supplied By
P1 reads u	VE	-	-	BR	Memory
P3 reads u	SC	-	SC	BR	Memory
P3 writes u	SC	-	SD	BU	P3
P1 reads u	SC	-	SD	-	-
P2 reads u	SC	SC	SD	BR	P3