

Sistemas Distribuídos

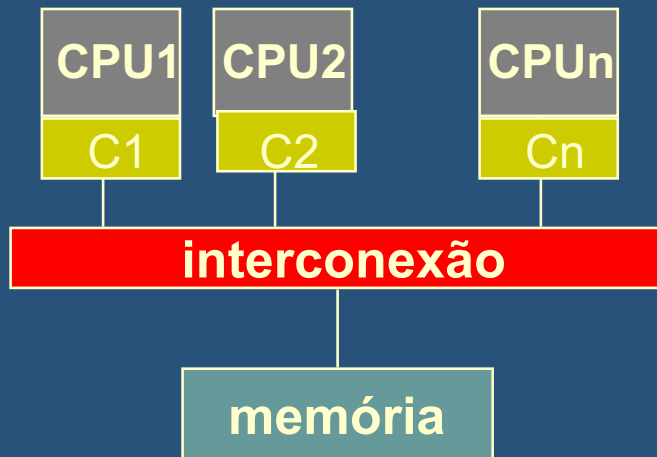
Módulo 8 – Distributed Shared Memory



Memória Compartilhada Distribuída

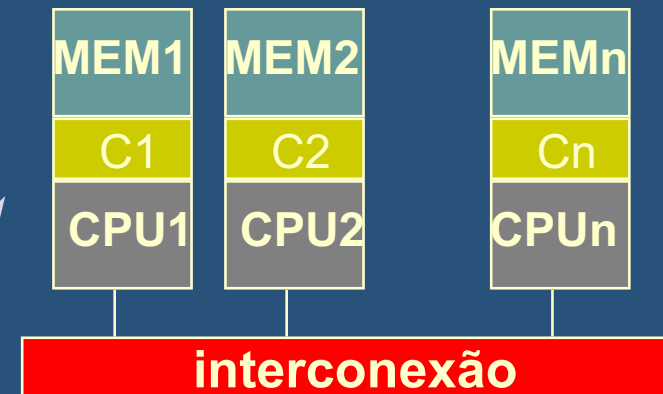


- Arquiteturas Fortemente acopladas



Variáveis Compartilhadas

- ◆ Arquiteturas Fracamente acopladas

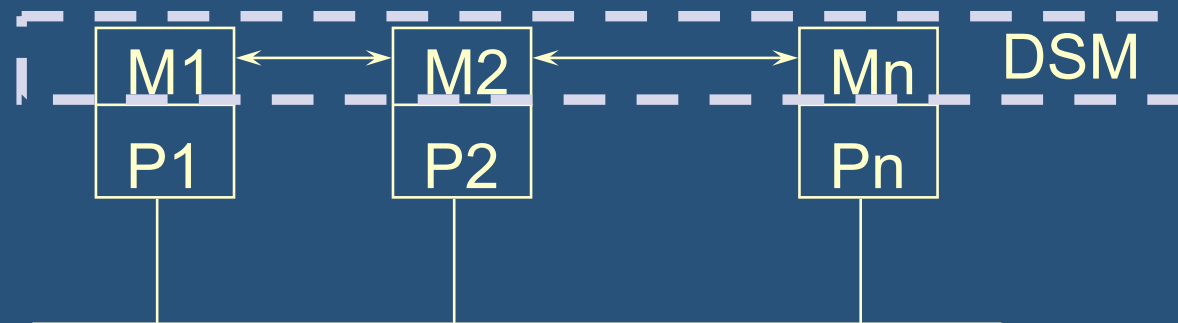


Troca de mensagens

Memória Compartilhada Distribuída



- A memória compartilhada distribuída (Distributed Shared Memory) é uma abstração que cria uma memória global, acessível por todos os processadores.



Memória Compartilhada Distribuída

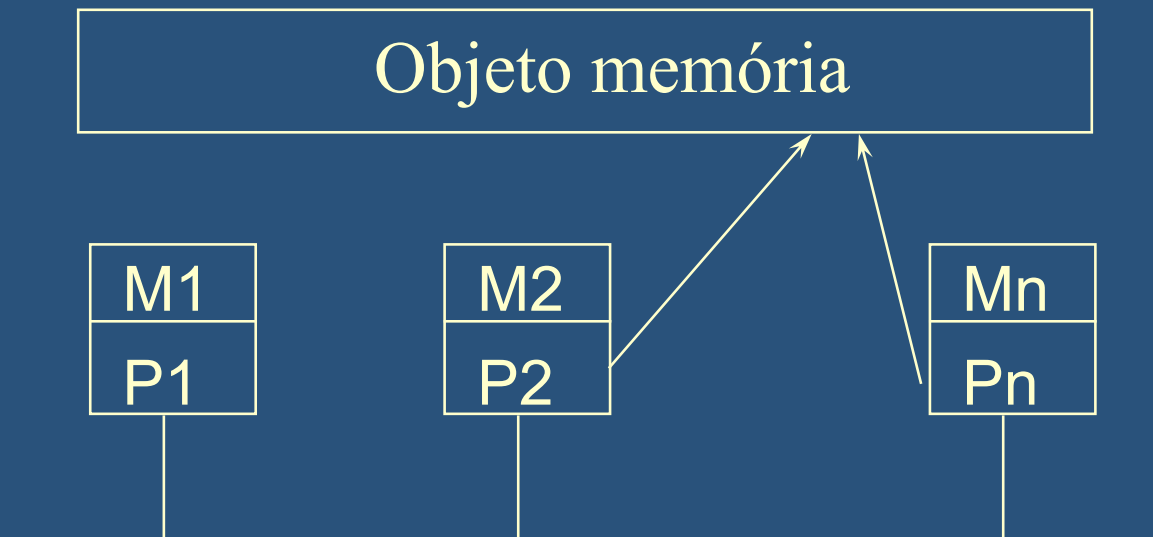


- A DSM pode ser implementada basicamente por duas abordagens:
 - Objetos distribuídos
 - Memória virtual compartilhada

DSM - Objetos Distribuídos



- Neste caso, a abstração de memória compartilhada é tratada como um objeto ou um conjunto de objetos.



DSM - Objetos Distribuídos



- As operações definidas sobre o objeto memória são as operações básicas de leitura e escrita. Estas operações serão efetuadas somente no interior do objeto.
- Em geral, as operações sobre um determinado objeto são indivisíveis. Esta suposição simplifica o tratamento da sincronização.

DSM - Objetos Distribuídos



- Como implementar os objetos distribuídos?
 - Objetos centralizados: todas as referências remotas são via mecanismos de comunicação (e.g. RPC)
 - Objetos migratórios: uma referência remota faz com que o objeto migre para o nodo que o referenciou
 - Objetos replicados: cada referência remota cria uma cópia do objeto no nodo local

DSM - Objetos Distribuídos

Exemplos

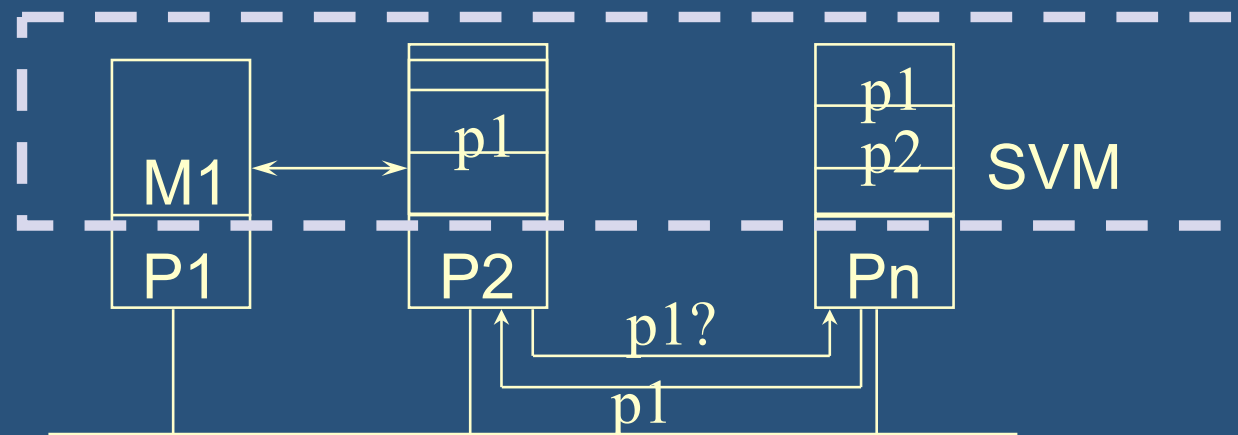


- Orca
 - Linguagem de programação orientada a objetos desenvolvida pelo grupo de Bal e Kaashoek na Holanda
 - <http://www.cs.vu.nl//vakgroepen/cs/orca.html>
- Aurora
 - Desenvolvida na Universidade de Toronto, propõe o comportamento dinâmico
 - <http://www.cs.utoronto.ca/~paullu/Aurora/aurora.html>

DSM - Memória Virtual Compartilhada



- Proposta por K. Li em Princeton, 1986
- A recuperação de dados é iniciada por page faults e feita por paginação entre as memórias locais



DSM - Memória Virtual Compartilhada



- A programação por SVM pode ser feita de maneira completamente transparente
- As operações de acesso à abstração de memória compartilhada são as operações de LOAD e STORE.

Memória Virtual Compartilhada

Localização da página



- Quando uma página não local é referenciada, ocorre um page fault. Em uma única máquina, se a página não está em memória local, ela está em disco.
- Em um ambiente DSM, a página pode estar em qualquer uma das memórias remotas ou no sistema de E/S.

DSM - Memória Virtual Compartilhada



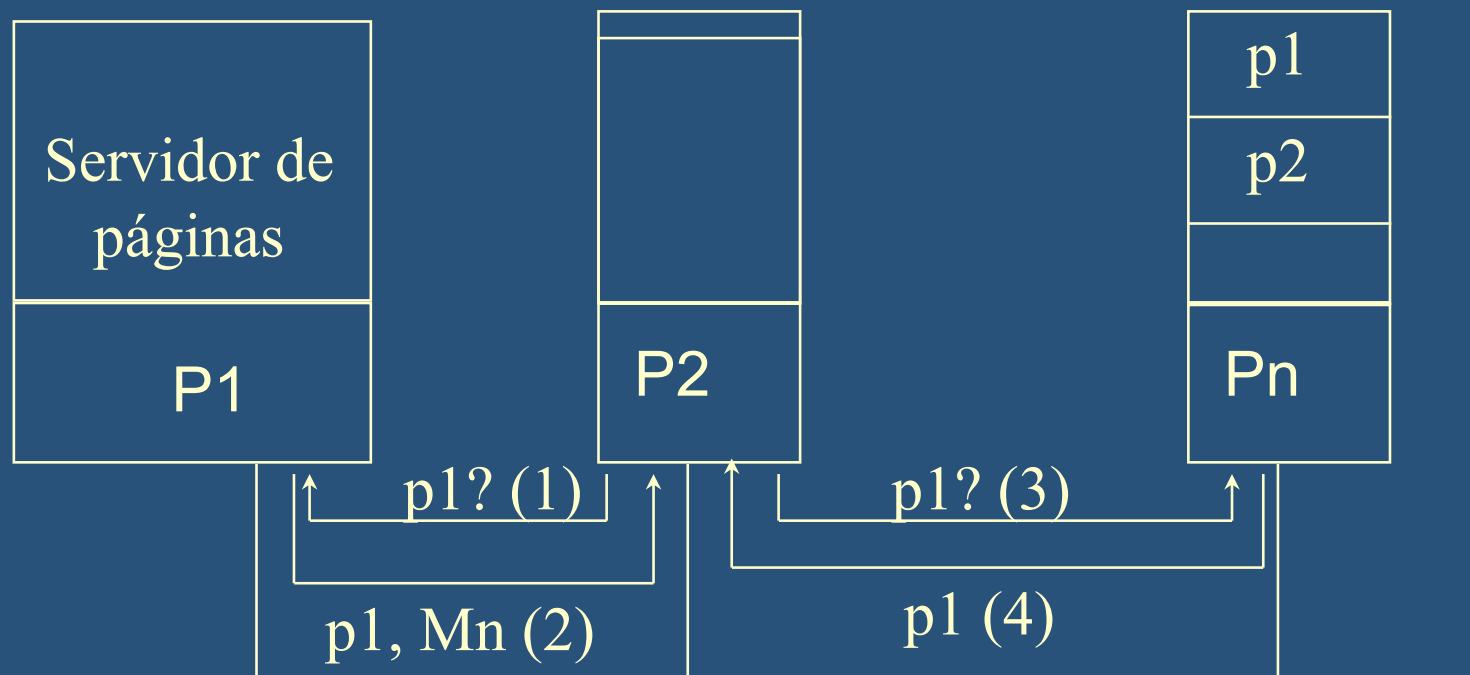
- ◆ Localização da página
 - ◆ Servidor centralizado
 - ◆ Servidor distribuído fixo
 - ◆ Servidor distribuído dinâmico

Memória Virtual Compartilhada

Localização da página



- Servidor centralizado

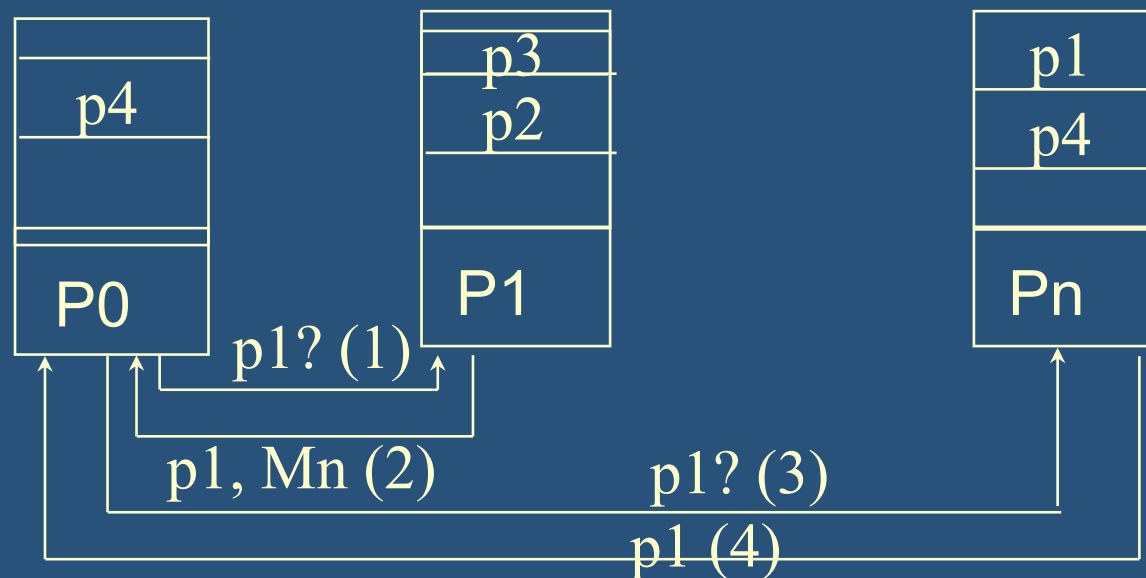


Memória Virtual Compartilhada

Localização da página



- Servidor distribuído fixo
 - Os processadores são numerados de 0 a n-1
 - Cada processador é o responsável pelo gerenciamento das páginas onde $(\text{page} \% n) = \text{número do processador}$.



Memória Virtual Compartilhada

Localização da página



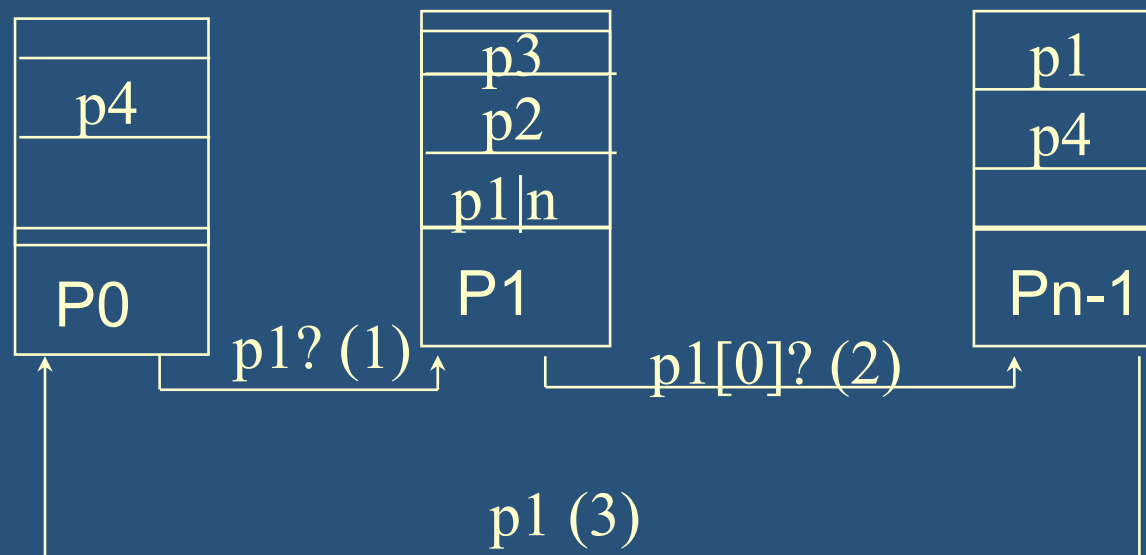
- Servidor distribuído fixo
 - Supondo um sistema com 10 processadores:
 - O processador 0 é o responsável pela gerência das páginas 0, 10, 20, 30
 - O processador 1 é o responsável pela gerência das páginas 1, 11, 21, 31
 - etc

Memória Virtual Compartilhada

Localização da página



- Servidor distribuído dinâmico
 - Só participam da gerência da página os processadores que já a tiveram ou a possuem em suas memórias.



Memória Virtual Compartilhada

Localização da página



- Servidor distribuído dinâmico
 - Na realidade, nós temos uma lista de prováveis proprietários da página (probable owner).
 - Se um nodo contido nesta lista não possuir a página, é garantido que o forward que ele vai oferecer leva ao proprietário real
 - Podem ser necessários vários forwards até que se chegue a um dono real da página



Memória Virtual Compartilhada

Localização da página



- Considerações
 - O servidor centralizado tem os problemas clássicos de gargalo e tolerância a falhas
 - O servidor distribuído dinâmico guarda dependência residual
 - O servidor distribuído fixo apresenta o melhor compromisso entre o desempenho e a simplicidade de implementação

Memória Distribuída Compartilhada - Consistência



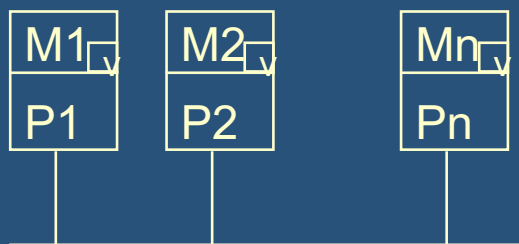
- Tanto a abordagem por objetos distribuídos como a abordagem SVM sofrem de problemas de desempenho relacionados à consistência dos acessos aos dados compartilhados.
- Consistência da memória: ordem na qual o conjunto de acessos à memória são percebidos pelo programador.

Memória Distribuída

Compartilhada - Consistência



- Coerência de cache: todas as cópias de um mesmo dado devem possuir o mesmo valor (quando observadas)



$$v(m1)=v(m2)=\dots=v(mn)$$

- Consistência da memória: diz respeito à ordem na qual o conjunto de acessos à memória são percebidos pelo programador. O problema de consistência da memória engloba o problema de coerência de cache

Memória Distribuída Compartilhada - Consistência



- Modelo de consistência de uma memória única: *as operações de acesso à memória são percebidas pelo programador como feitas atomicamente, na ordem especificada pelo programa* (consistência sequencial)
- Mantendo esse modelo, várias otimizações foram feitas nos monoprocessadores: write buffers e re-escalonamento de instruções

programa

a=1;

b=2;

c=a;

execução

b=2;

a=1;

c=a;

Memória Distribuída

Compartilhada - Consistência



- Para manter a consistência sequencial em uma máquina paralela, essas otimizações internas do processador não podem ser utilizadas.
- Todos os acessos à memória virtual compartilhada devem ser feitos na ordem do programa em todas as memórias locais
 - -> graves problemas de desempenho

nodo x

```
a=1;  
b=2;  
c=a;
```

nodo y

```
if (b==2)  
    print(a);  
a??
```

Memória Distribuída Compartilhada - Consistência



- Os primeiros sistemas implementam a consistência forte (Ivy, Mirage). Baixo desempenho
- Proliferação de modelos de consistência menos fortes

*programação
+ fácil*



consistência sequencial
consistência causal
consistência do processador
memória lenta

consistência do release
consistência de entrada

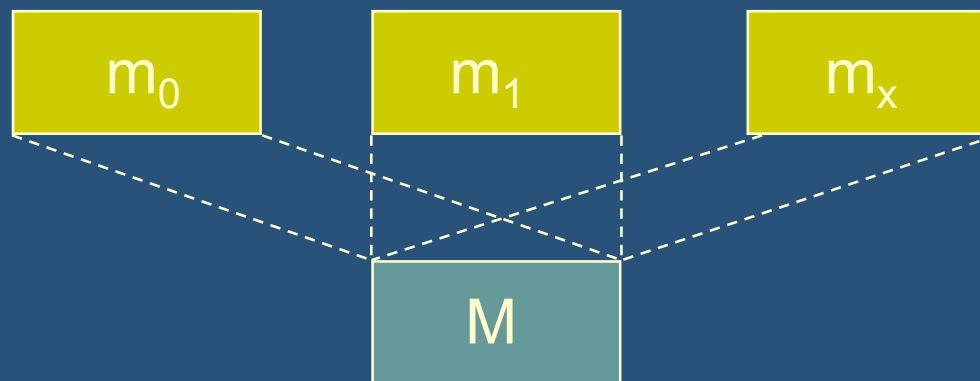


*maior
desempenho*

Modelos de Consistência - Formalização



- ♣ Um programa paralelo é executado por um conjunto de processadores $P=\{p_1, p_2 \dots p_x\}$ que acessam uma memória global compartilhada (M).
- ♣ Cada processador p_i possui sua memória local m_i , que é uma cache de todas as posições de M.



Modelos de Consistência - Formalização



- Cada operação é iniciada e depois terminada.
- Existem duas operações não atômicas básicas em M : leitura (r) e escrita (w).
- Uma escrita $w(x)1$ é na verdade um conjunto de escritas do valor 1 na posição x de cada memória m_i .

Modelos de Consistência

Consistência Atômica



- A consistência atômica é o modelo mais forte e mais antigo
- Todos os processadores percebem todos os acessos na mesma ordem (ordem global)
- Leva em consideração o tempo físico para dividir a execução de operações em intervalos

P1:	w(x)1	r(y)0	
P2:	w(x)2		
P3:			w(y)3

Modelos de Consistência

Consistência Sequencial



- A Consistência sequencial foi definida por Lamport: *um sistema é consistente segundo a consistência sequencial se o resultado de uma execução é equivalente à execução de todas as operações segundo uma ordem sequencial e as operações de cada processador aparecem nesta ordem sequencial segundo a ordem do seu programa.*
- Ordem do programa: se o_1 e o_2 são efetuadas pelo mesmo processador p_i e o_1 vem antes de o_2 no código de p_i , então o_1 precede o_2 segundo a ordem do programa

Modelos de Consistência

Consistência Sequencial



- Na consistência sequencial, todos os processadores devem perceber a mesma ordem de todos os acessos à memória (ordem global)

P1: $w(x)1 \quad w(z)3$

P2: $w(y)2$

P3: $r(y)2 \quad r(x)0 \quad r(x)1$

Ordem global: $w(y)2 \rightarrow r(y)2 \rightarrow r(x)0 \rightarrow w(x)1 \rightarrow r(x)1 \rightarrow w(z)3$

Modelos de Consistência

Consistência Causal



- A Consistência causal é um modelo de consistência relaxado. Não é mais necessário que os processos percebam a mesma ordem de todos os acessos (ordem parcial)
- A Consistência causal segue a relação de causalidade potencial definida por Lamport. Nesta relação, se um valor escrito por um processador ($w(x)v$) é lido por outro processador ($r(x)v$) então $w(x)v$ precede $r(x)v$ segundo a ordem causal.

Modelos de Consistência

Consistência Causal



P1:	$w(x)1$	$w(x)3$
P2:	$r(x)1$	$w(x)2$
P3:		$r(x)2$ $r(x)3$
P4:		$r(x)3$ $r(x)2$

- ordem P1: $w(x)1 \rightarrow w(x)2 \rightarrow w(x)3$
- ordem P2: $w(x)1 \rightarrow r(x)1 \rightarrow w(x)2 \rightarrow w(x)3$
- ordem P3: $w(x)1 \rightarrow r(x)1 \rightarrow w(x)2 \rightarrow r(x)2 \rightarrow w(x)3 \rightarrow r(x)3$
- ordem P4: $w(x)1 \rightarrow r(x)1 \rightarrow w(x)3 \rightarrow r(x)3 \rightarrow w(x)2 \rightarrow r(x)2$

Modelos de Consistência

Consistência do Processador



- A Consistência do Processador é na realidade uma família de modelos de Consistência (Pipelined RAM, PCGoodman, etc).
- Em PRAM, todos os processadores devem ver todos os writes feitos por um mesmo processador na ordem do seu programa.

Modelos de Consistência

Consistência PRAM



P1: $w(x)1$ $r(y)3$

P2: $r(x)1$ $w(y)3$

P3: $r(y)3$ $r(x)0$

- ordem P1: $w(x)1 \rightarrow w(y)3 \rightarrow r(y)3$
- ordem P2: $w(x)1 \rightarrow r(x)1 \rightarrow w(y)3$
- ordem P3: $w(y)3 \rightarrow r(y)3 \rightarrow r(x)0 \rightarrow w(x)1$

Modelos de Consistência

Consistência PCGoodman



- A Consistência PCGoodman é a consistência PRAM acrescida da coerência de cache.
- PCGoodman é então mais forte que a consistência PRAM.
- Outros modelos da família Processor Consistency: Total Store Order, Partial Store Order.

Modelos de Consistência

Memória Lenta



- Este é um dos modelos mais relaxados
- Na memória lenta, só é necessário que os processadores tenham a mesma visão das escritas feitas pelo mesmo processador na mesma posição de memória.
- Não garante a exclusão mútua lógica.

Modelos de Consistência

Memória Lenta

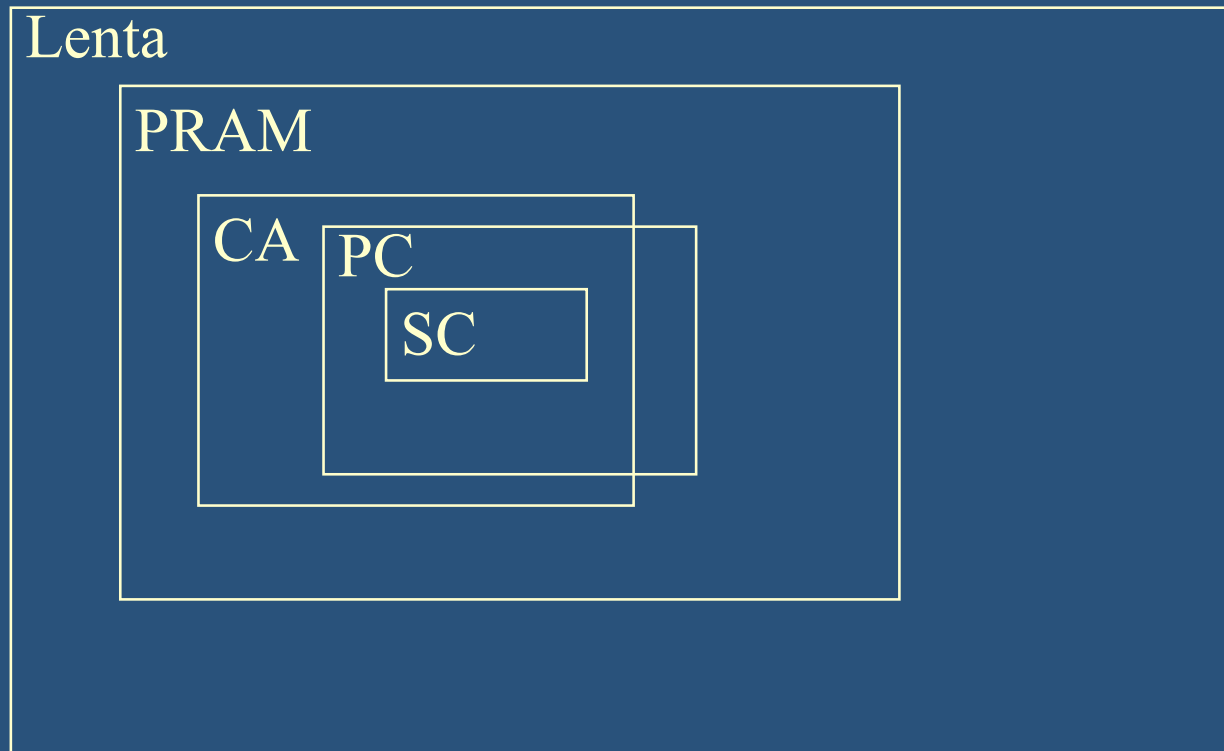


P1: $\frac{w(x)1 \quad w(y)3}{\quad}$

P2: $\frac{r(y)3 \quad r(x)0}{\quad}$

- ordem P1: $w(x)1 \rightarrow w(y)3$
- ordem P2: $w(y)3 \rightarrow r(y)3 \rightarrow r(x)0 \rightarrow w(x)1$

Relação entre modelos uniformes



Cada retângulo representa o conjunto de resultados possíveis em um determinado modelo.

SC=c. sequencial, PC=c. pcgoodman, CA=c. causal, PRAM= c. PRAM, Lenta=memória lenta

Modelos de Consistência Híbridos



- O paradigma de programação por memória compartilhada possui basicamente duas partes:
 - acesso a dados compartilhados
 - sincronização
- Existem alguns modelos de consistência, ditos híbridos, que determinam a execução de operações de consistência no momento da sincronização.
- Exemplos de modelos híbridos:
 - Weak Ordering, Release Consistency, Entry Consistency, Scope Consistency

Modelos de Consistência Híbridos



- ➡ Nos modelos híbridos, nós temos pelo menos 2 ordens:
 - ordenação dos acessos básicos em relação aos acessos de sincronização
 - ordenação dos acessos de sincronização entre eles



Consistência Fraca

- A consistência fraca, ou weak ordering, divide os tipos de acessos em: leitura (r), escrita (w) e sincronização (sync)
- Quando um sync é executado, todos os acessos anteriores são vistos por todos os processadores e nenhum acesso posterior ao sync é visto por nenhum processador.
- O sync funciona como uma barreira.
- Os acessos de sincronização obedecem a consistência sequencial

Consistência Fraca



P1: sync(l)1 w(x)1 w(y)2 sync(l)2

P2: sync(l)3 r(x)1 r(y)2 sync(l)4

- ordem P1: sync(l)1 -> w(x)1 -> w(y)2 -> sync(l)2 -> sync(l)3 -> sync(l)4
- ordem P2: sync(l)1 -> w(y)2 -> w(x)1 -> sync(l)2 -> sync(l)3 -> r(x)1 -> r(y)2 -> sync(l)4

Modelos de Consistência

Consistência do Release



- A consistência do release é o modelo de consistência mais utilizados atualmente.
- O acesso sync proposto na Consistência fraca é dividido em dois acessos: acquire e release.
- O release garante que os acessos anteriores são terminados.
 - As modificações são vistas no momento do release.
- O acquire garante que os acessos posteriores não são começados.

Modelos de Consistência

Consistência do Release



- Na consistência do release, os acessos são divididos em:
 - ordinários
 - leituras
 - escritas
 - especiais
 - não-sincronização
 - acquire
 - release

Modelos de Consistência

Consistência do Release



- Os acessos acquire e release podem ser vistos como a obtenção e a liberação de locks, respectivamente.
- Os acessos especiais de não-sincronização são acessos que se comportam de maneira mais forte que os acessos ordinários, porém não possuem o propósito de sincronização.

Modelos de Consistência

Consistência do Release



- ♣ A consistência do release necessita que todo acesso à memória compartilhada seja rotulado como leitura, escrita, acquire, release ou nsync.
- ♣ Um programa é dito propriamente rotulado se todo acesso compartilhado sem conflito for rotulado como leitura ou escrita, todo acesso compartilhado onde houver conflito for rotulado como especial e existirem acquires e releases suficientes para delimitar todas as seções críticas
 - ♣ Foi mostrado que um programa propriamente rotulado na RC comporta-se como se estivesse se executando na consistência sequencial

Modelos de Consistência

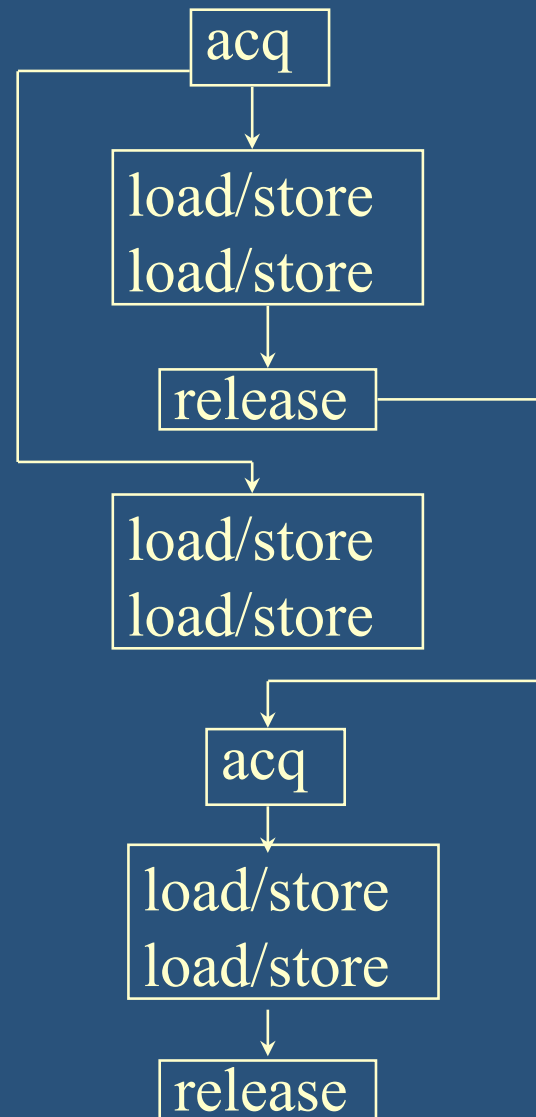
Consistência do Release



- Condições para a garantia da consistência do release (Rcsc):
 - Antes que um acesso ordinário possa ser executado, todos os acessos acquire precedentes devem ter sido terminados
 - Antes que um release possa ser executado, todos as leituras e escritas anteriores devem ter sido terminadas
 - Acessos especiais obedecem a consistência sequencial.

Modelos de Consistência

Consistência do Release



Modelos de Consistência

Consistência do Release

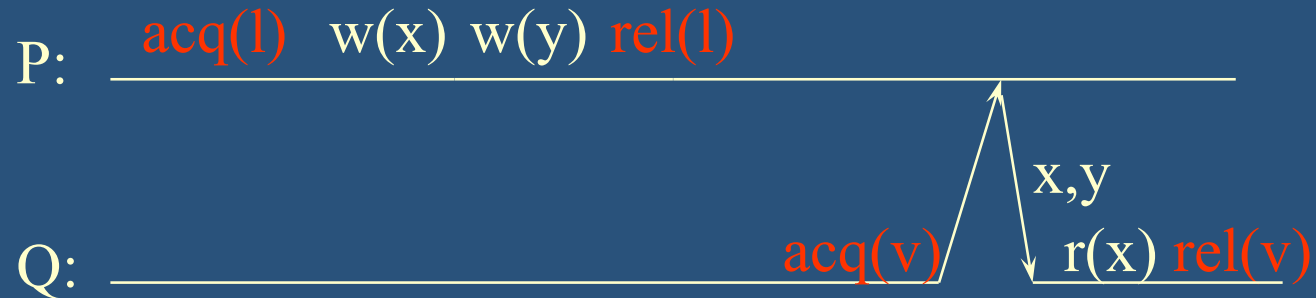


P1: acquire(l)1 w(x)1 w(y)2 release(l)2

P2: _____ acquire(l)3 r(x)1 r(y)2 release(l)4

- ordem P1: acq(l)1 → w(x)1 → w(y)2 → rel(l)2 → acq(l)3 → rel(l)4
- ordem P2: acq(l)1 → w(y)2 → w(x)1 → rel(l)2 → acq(l)3 → r(x)1 → r(y)2 → rel(l)4

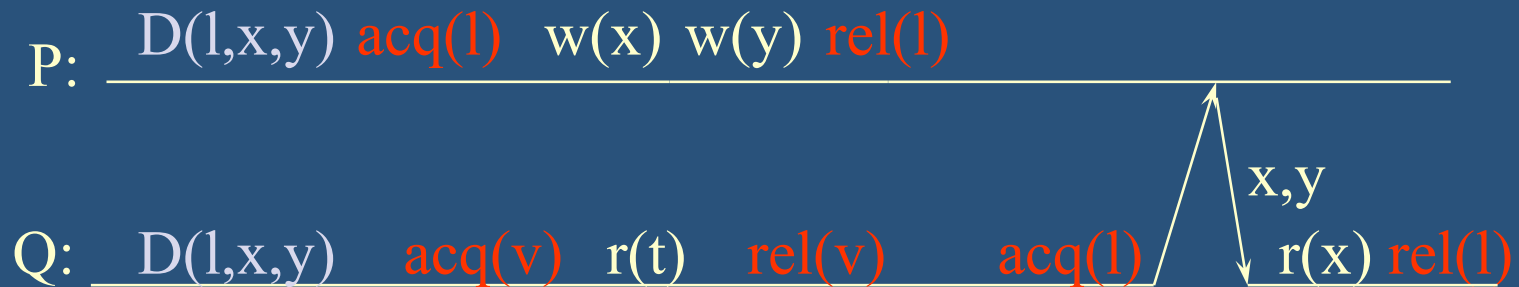
Lazy Release Consistency



- As modificações são visíveis no momento do próximo acquire
- Foi proposta como uma implementação da consistência do release.

Modelos de Consistência

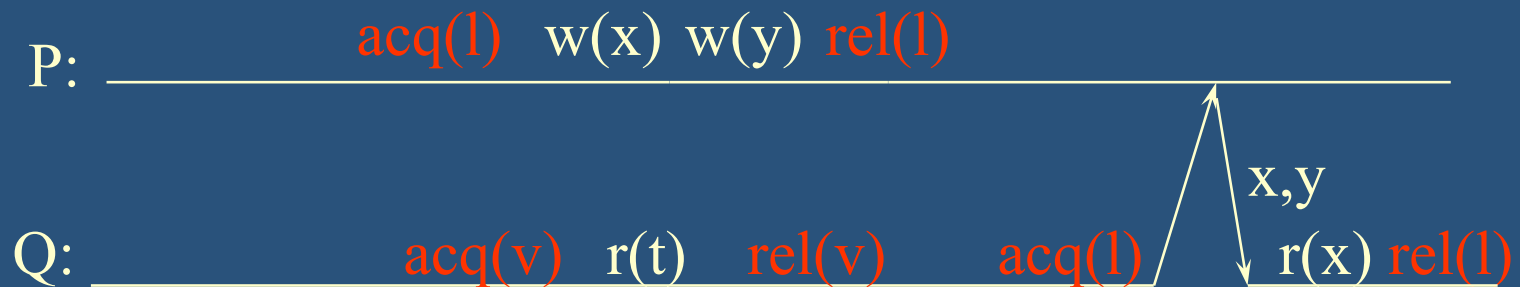
Entry Consistency (EC)



- ◆ Neste modelo, as atualizações nos dados guardados pelo lock l , só são enviadas para o próximo processo que obtem o mesmo lock.

Modelos de Consistência

Scope Consistency (ScC) (1996)



- ◆ Neste modelo, as atualizações nos dados que estão na seção crítica limitada pelo lock l , só são enviadas para o próximo processo que obtem o mesmo lock.

Modelos de Consistência

Qual escolher?



- ◆ A escolha de um modelo de consistência é um compromisso entre a performance e a facilidade de programação
- ◆ O desempenho de um modelo de consistência depende de como uma aplicação paralela acessa os dados compartilhados
- ➡ Sistemas que suportam múltiplos modelos de consistência
 - Midway
 - Mermera
 - DiVA



Protocolos de Coerência

- ◆ O Modelo de Consistência define **quando** as atualizações devem ser feitas.
- ◆ O protocolo de coerência define **como** as atualizações serão feitas
 - Multiple Reader Single Writer (MRSW)
 - Multiple Reader Multiple Writers (MRMW)

Protocolos de Coerência

MRSW

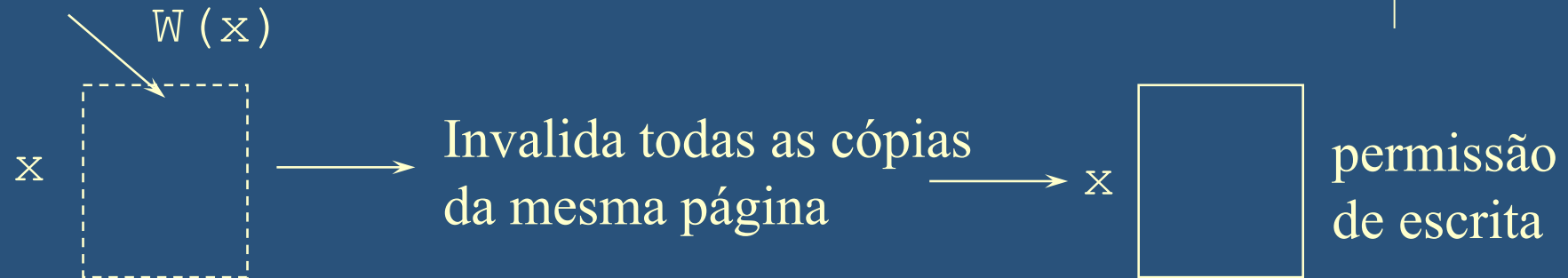
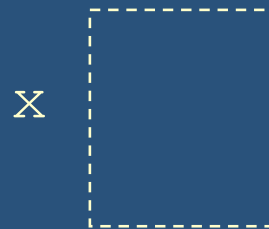


Diagram illustrating a read operation $R(x)$:

A process x initiates a read operation $R(x)$ (indicated by an arrow). The result is a process x with a question mark, indicating a read operation.

Caso 1 - Só existem cópias em leitura

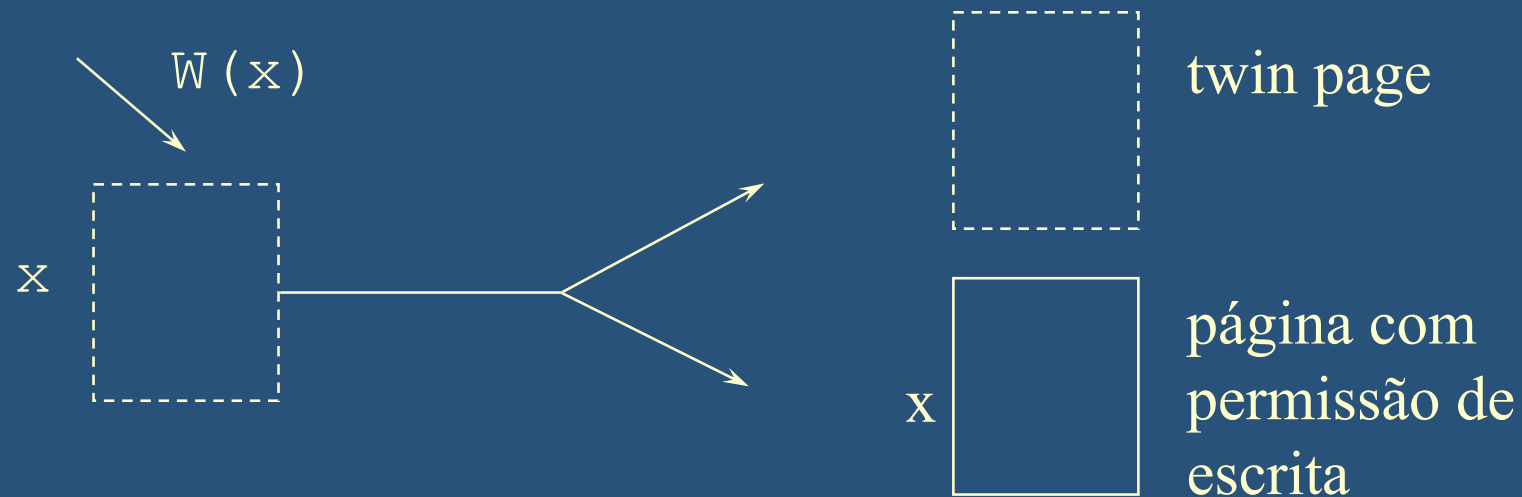


Caso 2 - Existe uma cópia em escrita

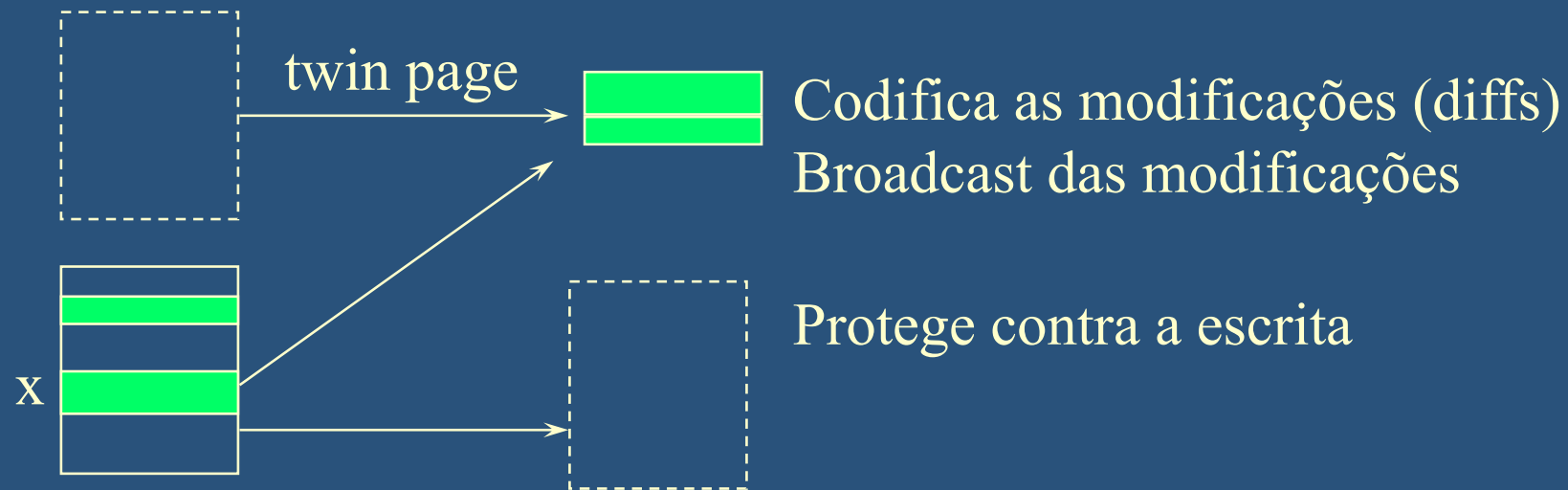


Protocolos de Coerência

MRMW



Release



Memória Compartilhada Distribuída



- ◆ Como é a programação da DSM?
 - Totalmente transparente (Ivy)
 - Com attach e detach de segmentos compartilhados (JIAJIA, TreadMarks)
 - Com chamadas à métodos remotos (ORCA)
 - Com o auxílio do compilador (Midway)

Programação da DSM Totalmente Transparente



`int a, b;`  `Compartilhado`

```
main()  
{  
    int c;  
  
    a = 1;  
    c = 2;  
    b = c;  
  
}
```


Programação da DSM

Attach e detach de segmentos



```
main()  
{  
    int c, *p;  
  
    p = attach (SEG_COMP);  
    *p = 1;  
    c = 2;  
    (*p) + 4 = c;  
    detach(p);  
}
```

Programação da DSM

Chamada remota a métodos




```
main()  
{  
    int c;  
  
    shared.atualiza("a",1);  
    c = 2;  
    shared.atualiza("b",c);  
}
```

Programação da DSM

Auxílio do Compilador



```
shared int a, b;  Novo tipo de dado  
float d,e;
```

```
main()  
{  
    int c;  
  
    a = 1;  
    c = 2;  
    b = c;  
  
}
```

Sincronização em sistemas DSM



- Os mecanismos de sincronização utilizados em sistemas DSM são geralmente os mesmos existentes em sistemas monoprocessados.
- O uso de spin locks em redes de interconexão genéricas pode causar um enorme overhead.
- Locks mutex são geralmente utilizados.

Sincronização em Sistemas DSM



- Principais questões envolvidas:
 - Localização dos objetos de sincronização
 - Política de atribuição de objetos de sincronização
 - Paradigma utilizado para implementar a sincronização

Localização dos Objetos de Sincronização - Locks



- Para localizar os locks, geralmente são utilizados os algoritmos de localização de páginas: servidor centralizado, servidor distribuído fixo ou servidor distribuído dinâmico.
- Como utilizar o algoritmo distribuído fixo, já que ele aplica uma função em um número (número da página) e os locks possuem nomes?
 - Associar um número a cada lock
 - Aplicar uma função sobre o nome do lock

Localização dos Objetos de Sincronização - Barreiras



- Na maior parte das implementações, as barreiras são gerenciadas por um gerente centralizado.
- A chamada à barreira pode ser:
 - Barreira: também conhecida como barreira global de sincronização, espera até que todos os processos da aplicação cheguem à barreira.
 - Barreira(n): espera até que n processos da aplicação cheguem à barreira.

Política de atribuição de objetos de sincronização



- O que fazer quando um lock é liberado e existem mais de um processador esperando?
 - Escolher randomicamente o processador que deterá o lock
 - Atribuir locks a processadores segundo a ordem FIFO



Paradigma utilizado para implementar a sincronização

- Para a implementação dos objetos de sincronização, podemos escolher entre dois paradigmas:
 - Memória compartilhada: usar a própria DSM para implementar a sincronização
 - Solução elegante, porém ineficiente
 - Troca de mensagens: implementar os objetos de sincronização através de troca de mensagens entre o gerente do lock e o processador da aplicação.
 - Solução mais utilizada

Exemplos de Sincronização em DSM



- Ivy: Contador de eventos
- TreadMarks: locks e barreiras
- JIAJIA: locks, variáveis de condição e barreiras
- Brazos: locks e barreiras
- Orca: locks e variáveis de condição
- Midway: locks e barreiras
- Shasta: locks, barreiras e variáveis de condição

Sistemas DSM mais recentes



- Kerrighed (2010): consistência sequencial
 - www.kerrighed.org/wiki/index.php/Download
- Concordia (2021): consistência forte (strong), uso da placa de rede
 - <https://www.usenix.org/system/files/fast21-wang.pdf>
- MENPS (2020): release consistency e RDMA
 - <https://github.com/endowataru/menps>