Linguagem de Programação II

Carlos Eduardo Batista

Centro de Informática - UFPB
bidu@ci.ufpb.br

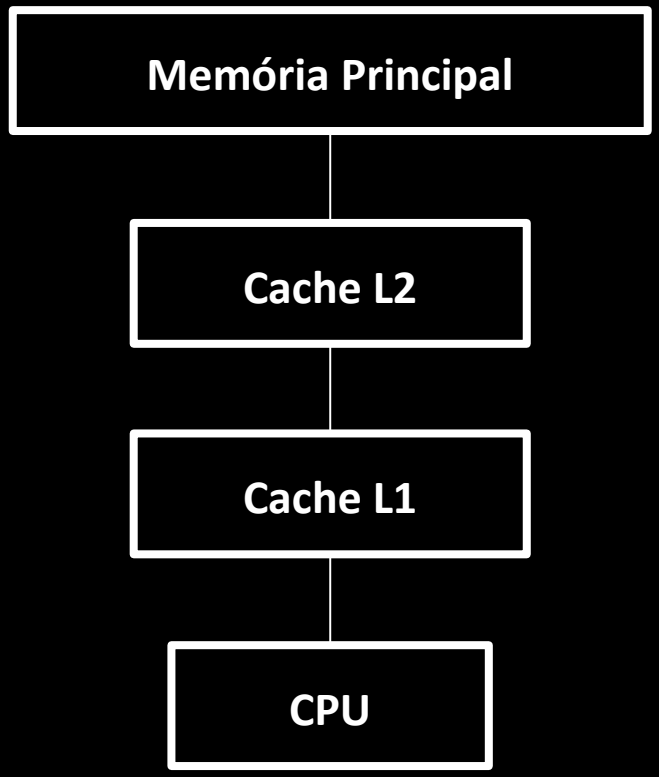
Motivação

- Adaptar a estrutura lógica de um problema (Ex.: Servidores Web).
- Lidar com dispositivos independentes (Ex.: S.O.).
- Aumentar o desempenho das aplicações (Ex.: aplicações de Big Data, SETI@Home).
- Arquiteturas de computadores paralelos
  - Avanços mais recentes das Arquitetura de Computadores têm implicado muito mais no incremento de paralelismo (multiprocessadores ou multicores) do que no aumento velocidade (*clock*).

Arquitetura de Computadores

- **Memória *cache***
  - Mais rápida e cara.
  - Localidade temporal e espacial.
  - **Localidade temporal**
    - Uma vez acessada uma determinada posição de memória, há uma grande probabilidade dessa posição ser novamente acessada em um curto intervalo de tempo.
  - **Localidade espacial**
    - Quando uma determinada posição de memória é referenciada, existe uma grande probabilidade de que as posições vizinhas também sejam acessadas.

Arquitetura de Computadores



Arquitetura de Computadores Modernos

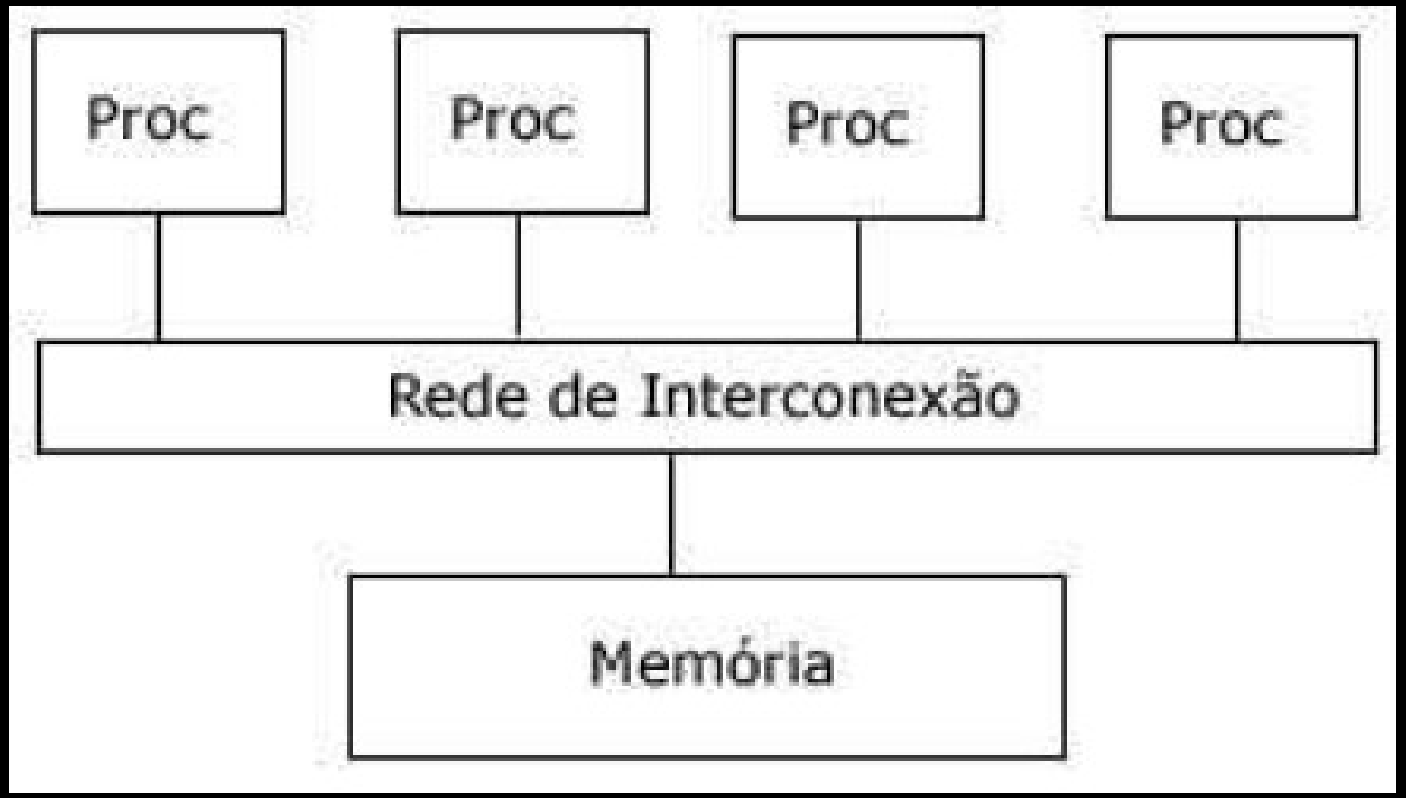
- Computadores paralelos
  - Multiprocessador de Memória Compartilhada
  - Multiprocessador de Memória Distribuída
- Taxonomia de Flynn
  - Single Instruction Single Data (SISD)
  - Multiple Instruction, Single Data stream (MISD)
  - Single Instruction, Multiple Data streams (SIMD)
  - Multiple Instruction, Multiple Data streams (MIMD)

Arquitetura de Computadores

- Leitura sugerida:
  - Flynn's taxonomy (Wikipedia)
    - http://en.wikipedia.org/wiki/Flynn%27s_taxonomy
  - "Some Computer Organizations and Their Effectiveness" (FLYNN, M.J.)
    - http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=5009071

Multiprocessador de memória compartilhada

Máquinas UMA (*Uniform Memory Access*)

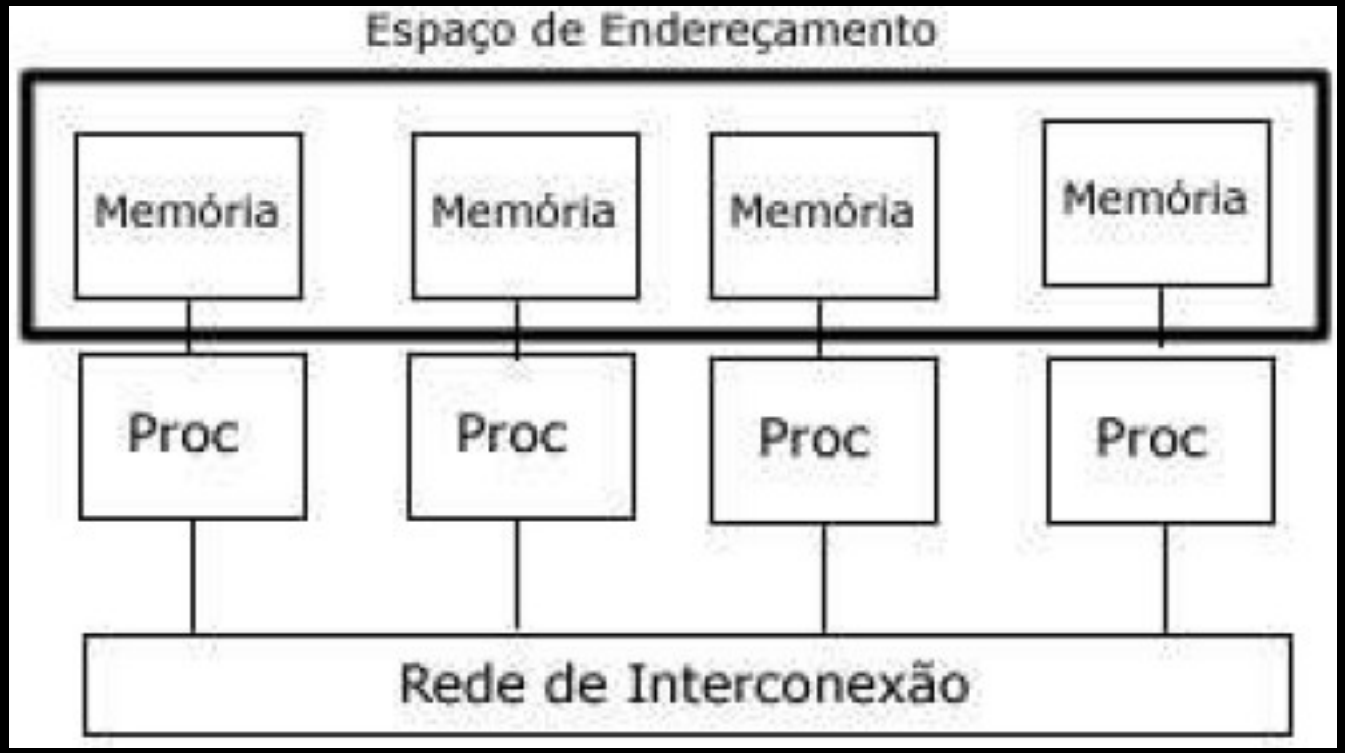


Multiprocessador de memória compartilhada

– IBM p690 (32 processadores Power4+)

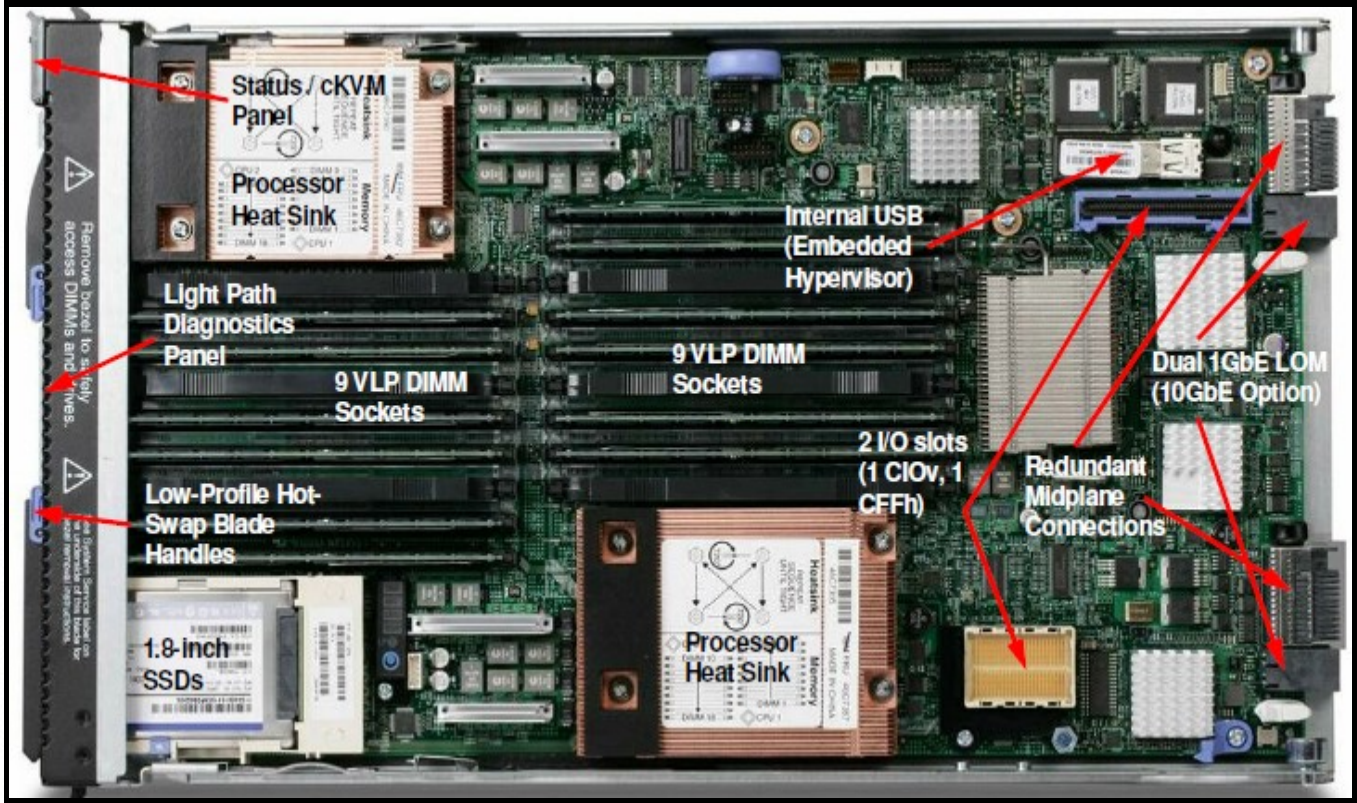
Multiprocessador de memória compartilhada

Máquinas NUMA (*Non-Uniform Memory Access*)



Multiprocessador de memória compartilhada

- IBM HS22 (2 processadores Xeon 5500 da



Arquitetura de Computadores Modernos

- Quando dois processadores fazem referência a uma mesma posição na memória compartilhada, pode ocorrer um problema de **inconsistência de cache**.
- Operações de escrita podem levar a um problema de **inconsistência de memória**.
- Manter a sequência natural das instruções em um programa executado de forma paralela é *caro* e muitas vezes complexo.

Coerência de Cache

- Diversas Arquiteturas de máquinas paralelas atuais são construídas com processadores produzidos em larga escala - Reduzir os custos de projeto
- Máquinas paralelas com múltiplos processadores incorporam *caches* em suas arquiteturas
- Presença de *caches* privadas em multiprocessadores necessariamente introduz problemas de coerência de *cache*

Coerência de Cache

- Descrição do problema
  - Múltiplas cópias de mesma posição de memória podem existir em diferentes *caches*
  - Cada processador atualiza sua cópia local, não se preocupando com a existência de outras cópias em outros processadores
  - Cópias de mesmo endereço de memória poderão possuir valores diferentes → **Caracteriza uma situação de inconsistência de dados**

Coerência de Cache

- Coerência de cache em multicomputadores
  - **Não** ocorre pois cada nó possui hierarquia de memória inteira
  - **Não** existe espaço de endereçamento global compartilhado por todos

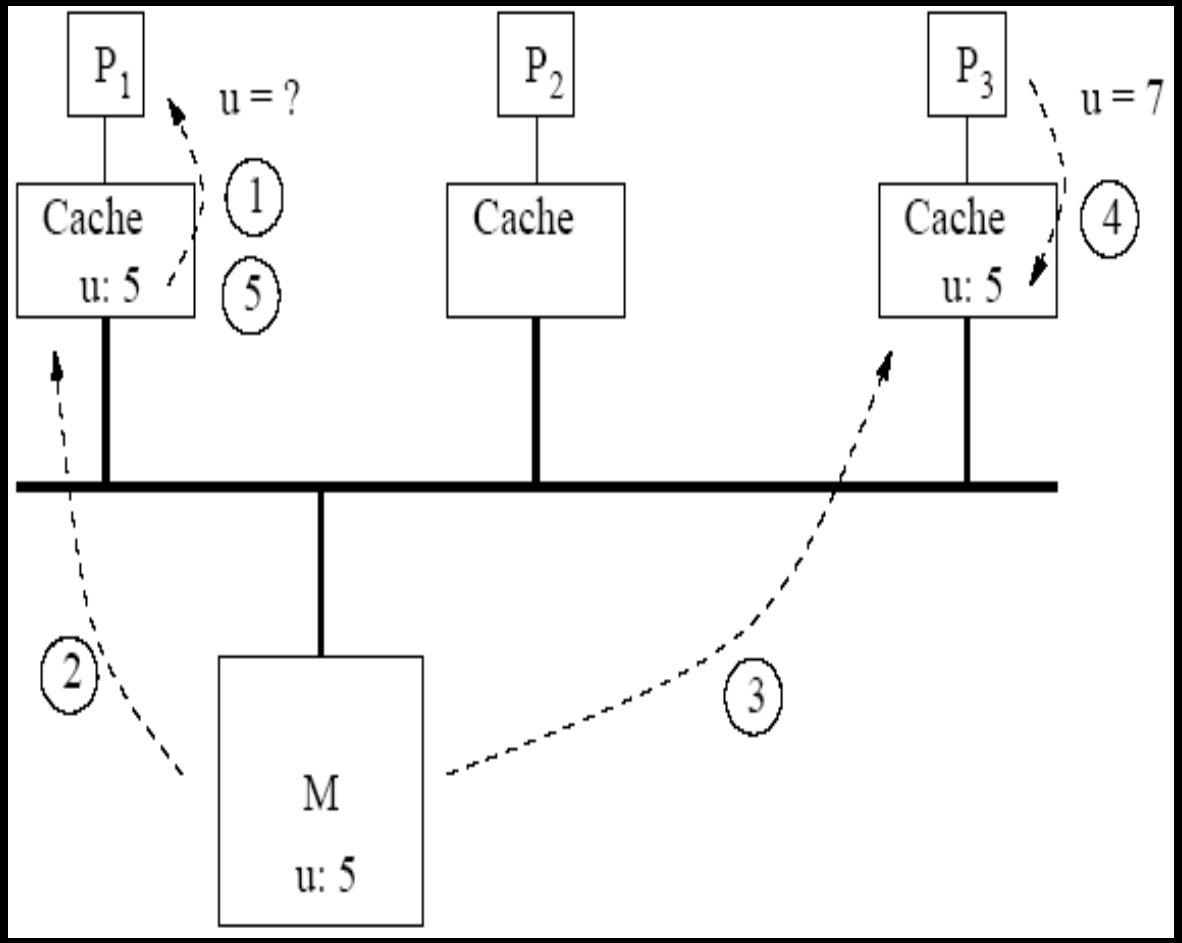
Coerência de Cache

- Uma arquitetura multiprocessada com caches privadas é coerente se e somente se uma leitura de uma posição x de memória efetuada por qualquer processador i retorne o valor mais recente desse endereço

Coerência de Cache

- Toda vez que uma escrita for efetuada por um processador i em um endereço de memória x, tem que ser garantido que todas as leituras subseqüentes de x, independentemente do processador, forneçam o novo conteúdo de x

Coerência de Cache



Coerência de Cache

- Estratégias básicas para tratar coerência de cache
  - Escrita na cache resulta atualização de outras cópias desse dado nas demais caches (*write-update*)
  - Escrita na cache resulta invalidação de outras cópias desse dado nas demais caches (*write-invalidate*)

Coerência de Cache

- Invalidação tem custo menor, mas maior latência de acesso caso as cópias invalidadas sejam novamente acessadas
  - Necessita buscar da memória principal
- Atualização tem custo mais alto, especialmente em máquinas com muitos processadores (muitas cópias potenciais de mesmo endereço), mas menor latência
  - Novo acesso a cópias é resolvido em nível de cache

Arquitetura de Computadores Modernos

- Consistência de memória é a política que determina como e quando mudanças feitas por um processador são vistas pelos outros processadores do sistema.
- Um modelo de consistência de memória define um contrato entre o software e o sistema de memória
  - Se o software obedecer certas regras, o sistema de memória funcionará corretamente.

Arquitetura de Computadores Modernos

- Modelos de consistência de memória
  - Consistência sequencial
  - Consistência de processador
  - Consistência de liberação

Arquitetura de Computadores Modernos

- Consistência sequencial
  - A execução paralela de um programa é sequencialmente consistente se qualquer uma de suas execuções é equivalente a uma execução entrelaçada em um único processador

Arquitetura de Computadores Modernos

- **Consistência sequencial**

Initially Flag1 = Flag2 = 0

P1	P2
Flag1 = 1	Flag2 = 1
if (Flag2 == 0)	if (Flag1 == 0)
critical section	*critical section*

(a)

Initially A = B = 0

P1	P2	P3
A = 1		
	if (A ==1)	
	B = 1	
		if (B==1)
		register1 = A

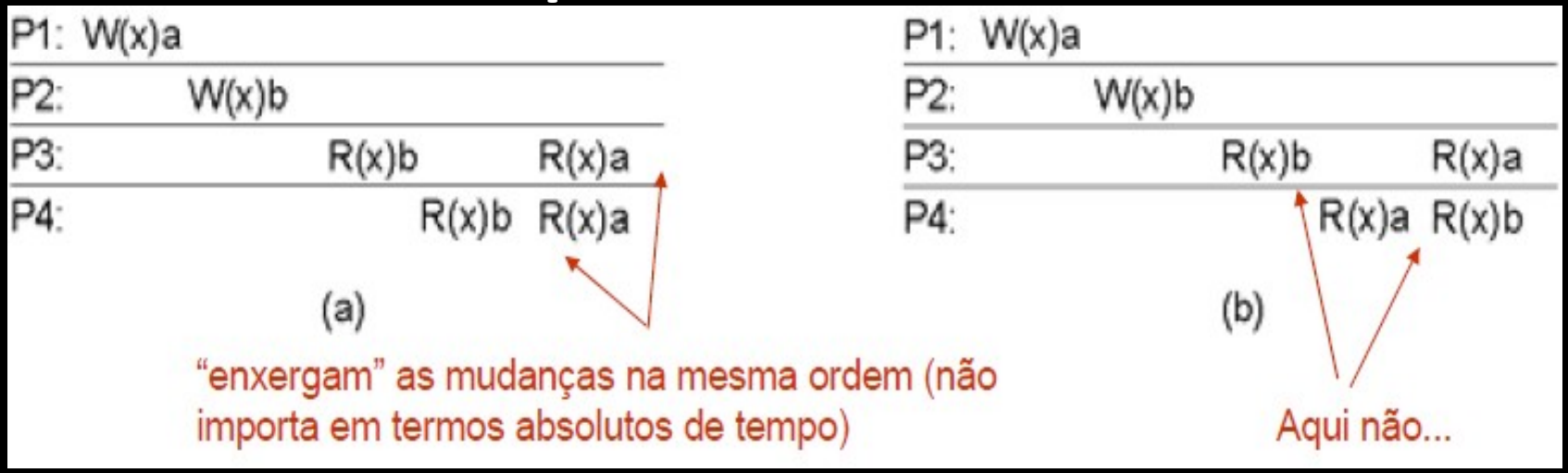
(b)

A ordem das operações em cada processador deve ser preservada .

O efeito das operações deve ser percebido pelos processadores no mesmo tempo.

Arquitetura de Computadores Modernos

Consistência sequencial

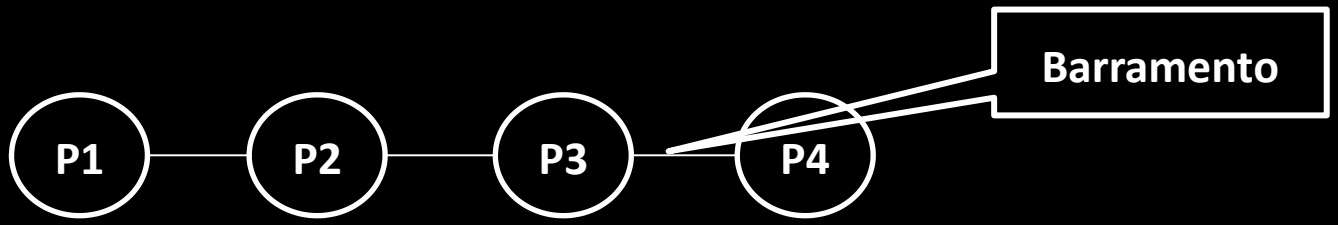


Arquitetura de Computadores Modernos

- **Consistência de processador (Goodman)**
  - Operações de escrita de um único processador são recebidas por todos os outros processadores na ordem em que são executadas, enquanto que operações de escrita de processadores distintos podem ser enxergadas em ordens diferentes por processadores diferentes.

Arquitetura de Computadores Modernos

- **Consistência de processador**



  - O processador P1 propaga sua operação de escrita, que chega primeiro em P2 e depois em P3.
  - O inverso ocorre com a operação de escrita propagada por P4.

Arquitetura de Computadores Modernos

- **Consistência de processador (Gharachorloo et al)**
  - Garante apenas que as operações de escrita ocorrerão na memória na ordem que são consideradas pelo processador.
  - Permite que uma operação de leitura posterior a uma de escrita seja realizada antes que essa operação de escrita seja percebida por todos os processadores.
  - Otimiza o desempenho do *hardware*.

Arquitetura de Computadores Modernos

- Consistência de processador

```
P1:   W(x)1 W(x)2
-------------------------------
P2:                 R(x)2 R(x)1
```

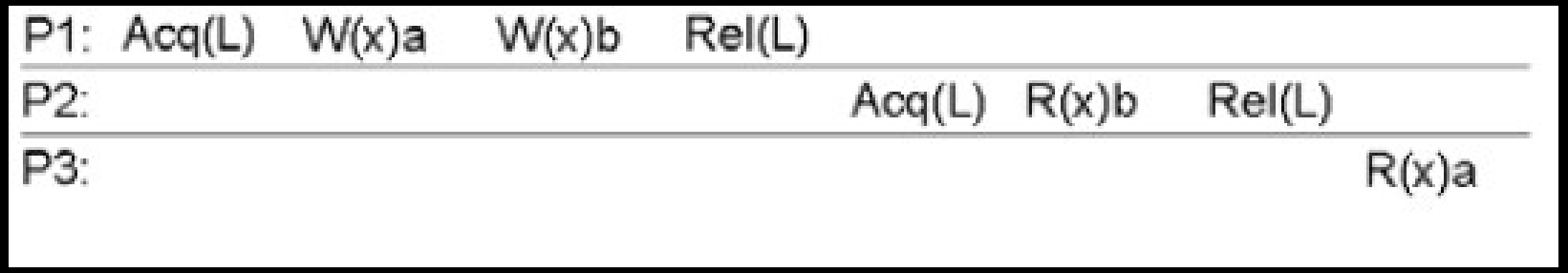
Cenário não condizente com este modelo.

Arquitetura de Computadores Modernos

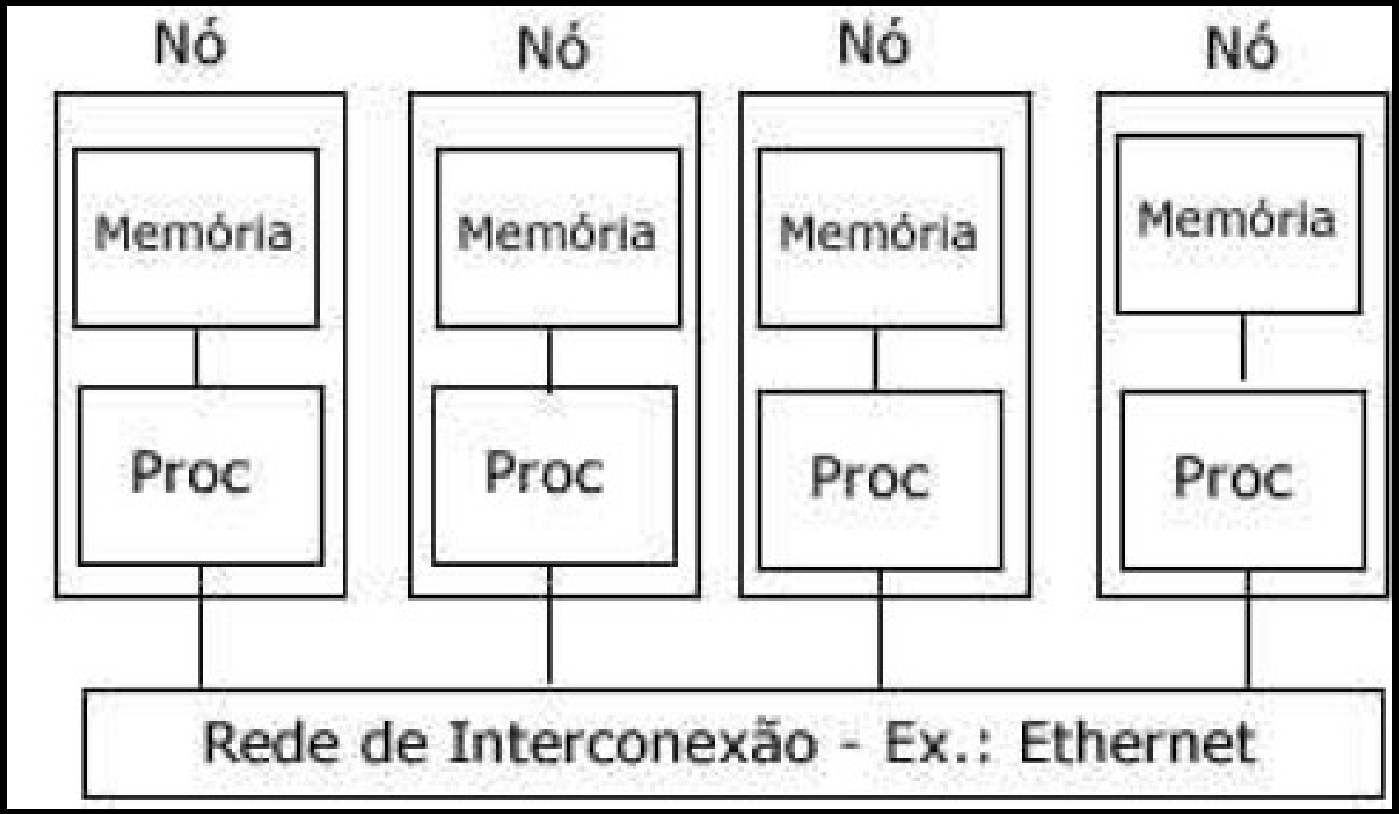
- **Consistência de liberação**
  - Antes de qualquer acesso a memória por operações de leitura ou escrita, todas as operações de obtenção (*acquire*) devem ter sido executadas.
  - Antes de um acesso de liberação (*release*) ser executado, todas as operações de leitura ou escrita na memória devem ser executadas.
  - Acessos especiais como obtenção (*acquire*) ou liberação (*release*) devem ser sequencialmente consistentes.

Arquitetura de Computadores Modernos

- **Consistência de liberação**
  - São necessárias operações de sincronização explícitas para garantir a ordem em que as operações de leitura e escrita se realizarão.



Multiprocessador de memória distribuída



Arquitetura de Computadores Modernos

- **Multiprocessadores de memória distribuída**
  - **Multicomputadores**
    - Máquinas fortemente acopladas.
    - Processadores fisicamente próximos, conectados por um barramento de alta velocidade.
    - Ex.: Mesh ou hipercubo.
  - **Redes**
    - Máquinas fracamente acopladas.
    - Nós conectados através de uma rede de comunicação.
    - Ex.: Beowulf.

Arquitetura de computadores modernos

- Beowulf

Arquitetura de Computadores Modernos

- **Multiprocessador de memória distribuída**
  - Cada processador possui sua própria memória privada.
  - A comunicação entre os processador se dá por meio de troca de mensagens.
  - Como não há memória compartilhada, não há problemas de consistência de *cache* e de memória.

Desafios da Programação Concorrente

- Construir programas de computador que trabalhem juntos e que possibilitem o uso eficiente das arquiteturas de *hardware* de processamento paralelo.
- As aplicações concorrentes são divididas em três grandes grupos:
  - **Sistemas *Multithreaded***
  - **Sistemas Distribuídos**
  - **Sistemas Paralelos**

Desafios da Programação Concorrente

- **Sistemas *Multithreaded***
  - Sistemas que apresentam mais *threads* do que processadores. Ex.: Gerenciador de janelas.
- **Sistemas Distribuídos**
  - Sistemas que cooperam entre si, residindo em computadores distintos que se comunicam através de uma rede. Ex.: A Web.
- **Sistemas Paralelos**
  - Sistemas que necessitam de grande poder de processamento. Ex.: Problemas de otimização.

Desafios da Programação Concorrente

- Para desenvolver programas concorrente é necessário conhecer três elementos:
  - **Regras**
    - Modelos formais que ajudam a entender e desenvolver programas corretos.
  - **Ferramentas**
    - Mecanismos oferecidos pelas linguagens de programação para descrever computações concorrentes.
  - **Estratégias**
    - Paradigmas de programação adequados para as aplicações tratadas

Desafios da Programação Concorrente

- As aplicações concorrentes devem ser vistas por dois ângulos:
  - **Computabilidade**
    - Princípios que abordam o que pode ser computado em um ambiente concorrente.
  - **Desempenho**
    - Avaliação do aumento no desempenho.

Leitura sugerida

- Parallel computing (Wikipedia)
  - http://en.wikipedia.org/wiki/Parallel_computing
- Amdahl's law (Wikipedia)
  - http://en.wikipedia.org/wiki/Amdahl's_law
- Scientist out to break Amdahl's law
  - http://www.pcworld.com/article/2042256/scientist-out-to-break-amdahls-law.html

Referências

- Notas de Aula do Prof. Bruno Pessoa
- Andrews, G. *Foundations of Multithreaded, Parallel, and Distributed Programming*, Addison-Wesley, 2000.
- Rosseto, S. , “Cap. I: Introdução e histórico da programação concorrente”. Disponível em: <http://www.dcc.ufrj.br/~silvana/compconc/cap-1.pdf>.