

# Memória Compartilhada Distribuída

- Visão Geral
- Implementação
- Produtos

# Memória Compartilhada Distribuída

- Mecanismos tradicionais de comunicação via RPC/RMI ou mensagens deixam explícitas as interações entre processos
  - Processos interagem para trocar dados de modo a manter um estado global consistente
  - É possível abstrair completamente as interações entre processos, de modo que todos os processos de uma aplicação distribuída tenham acesso ao mesmo estado global, replicado localmente?

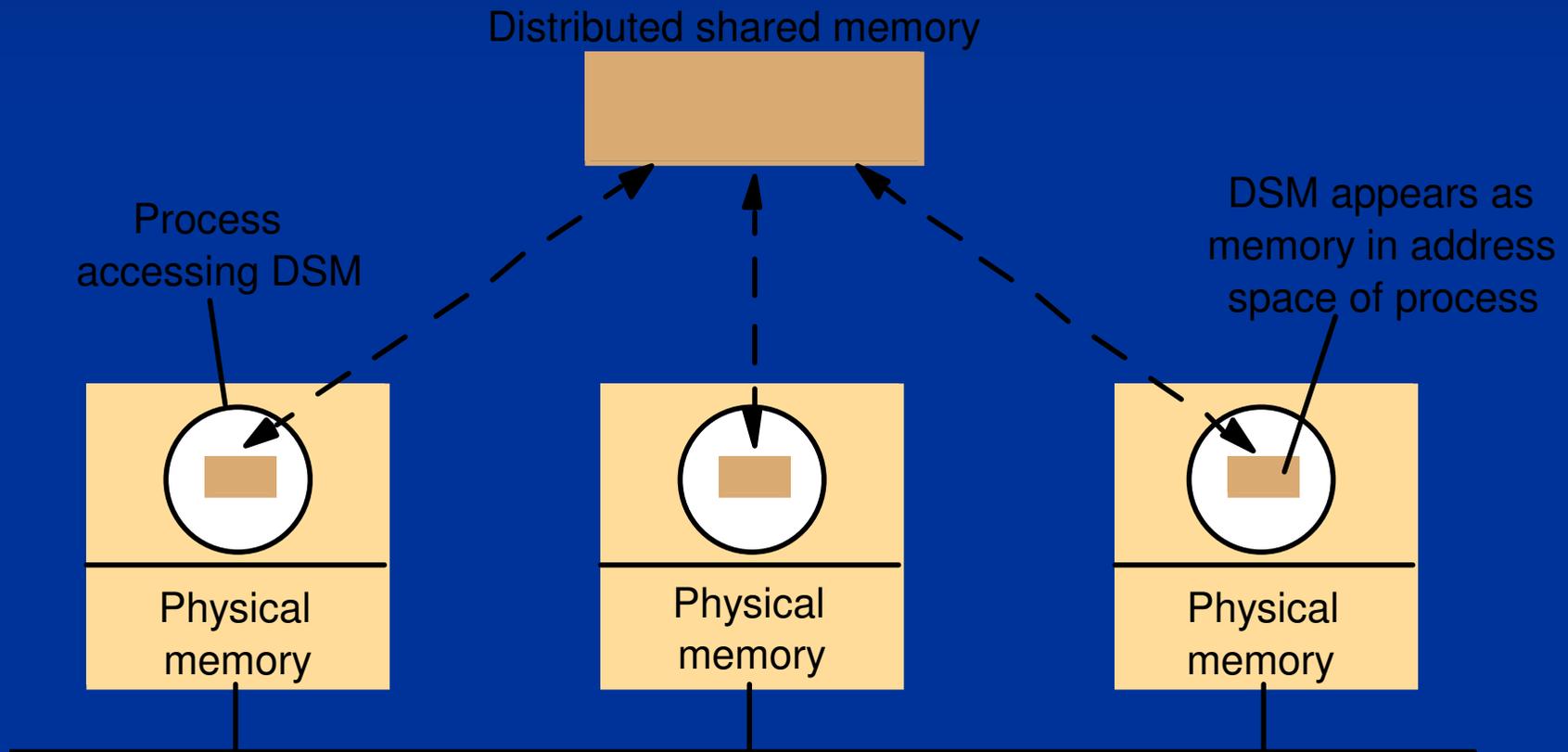
# Memória Compartilhada Distribuída

- DSM (*Distributed Shared Memory*)
  - “Abstração usada para o compartilhamento de dados entre computadores que não compartilham memória física.”

*Colouris, Dollimore & Kindberg*
  - Processos lêem e escrevem na memória compartilhada como se estivessem acessando seu espaço de endereçamento de memória
  - Evita a necessidade de lidar diretamente com os mecanismos de comunicação, tornando a manutenção de estado global transparente

# Memória Compartilhada Distribuída

## ■ Visão do Usuário da DSM



# Memória Compartilhada Distribuída

- Principal Utilização: Computação de Alto Desempenho
  - Com DSM, as máquinas de um sistema distribuído podem se comportar como se fossem um multiprocessador com memória compartilhada
  - Usando DSM, programas desenvolvidos para multiprocessadores com memória compartilhada podem ser facilmente portados para multiprocessadores sem memória compartilhada, clusters e grids

# Memória Compartilhada Distribuída

- Outras Formas de Utilização:
  - Aplicações distribuídas nas quais dados são compartilhados e podem ser acessados diretamente pelos processos envolvidos
  - Algumas aplicações do tipo cliente/servidor, onde o servidor pode compartilhar uma área de memória para acesso de todos os clientes

# Memória Compartilhada Distribuída

## ■ Vantagens

- **Transparência:** usuário não precisa lidar diretamente com mecanismos de comunicação
- **Desenvolvimento:**
  - Programador lida com conceitos com os quais já está familiarizado
  - Aplicações centralizadas podem ser facilmente paralelizadas/distribuídas

# Memória Compartilhada Distribuída

## ■ Limitações

- Suporte de execução:
  - Faz atualizações na memória compartilhada
  - Deve estar presente em todos os nós
- Desempenho:
  - Acesso à DSM pode ser várias ordens de grandeza mais lento que o acesso à memória física, degradando o desempenho
  - Desempenho é ruim principalmente quando há muitas alterações nos valores dos dados
- Falta: um processo falto pode corromper os dados compartilhados e comprometer o funcionamento de todo o sistema

# Memória Compartilhada Distribuída

- Comparação com outras tecnologias
  - Dados são compartilhados diretamente por processos, não precisando ser passados como mensagens (MOM) ou parâmetros (RPC/RMI)
  - Requer o uso de mecanismos de controle de concorrência para garantir a consistência dos dados na área de memória compartilhada

# Implementação

- Implementação em Hardware
  - Usada em arquiteturas de multiprocessadores do tipo NUMA (*Non-Uniform Memory Access*)
  - Grupos de processadores (tipicamente 4), ligados por um barramento de alta velocidade, compartilham um banco de memória local
  - Processos acessam espaço de endereçamento que reúne memória local e remota
  - Acesso à memória remota é mais lento que o acesso à memória local
  - Hardware especializado executa as instruções de acesso à memória, usando uma *cache* local e efetuando acesso remoto, se necessário

# Implementação

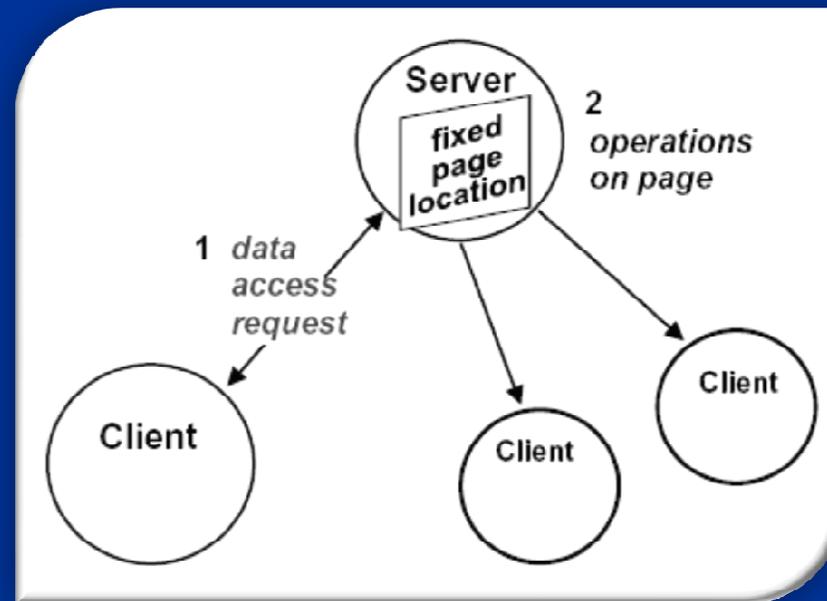
- Implementação por *Software*
  - *Middleware* é responsável por efetuar a comunicação entre os nós para manter a memória compartilhada atualizada e consistente
  - Processos chamam o *middleware* para acessar dados compartilhados
  - Requer comunicação entre os nós envolvidos
  - Em geral não manipula endereços de memória diretamente, mas usa abstrações de alto nível, como variáveis/objetos compartilhados

# Implementação

- Representação dos dados na DSM
  - Cada implementação de DSM define o formato usado para representar dados compartilhados
  - Formatos comumente utilizados:
    - Bytes: devem ser transformados nos formatos de dados usados pela aplicação
    - Tuplas: mantém tuplas de valores tipados em um espaço de tuplas; são imutáveis – criadas, lidas e removidas, nunca alteradas
    - Objetos: dados encapsulados manipulados através de invocações a métodos; limita a portabilidade entre plataformas/linguagens

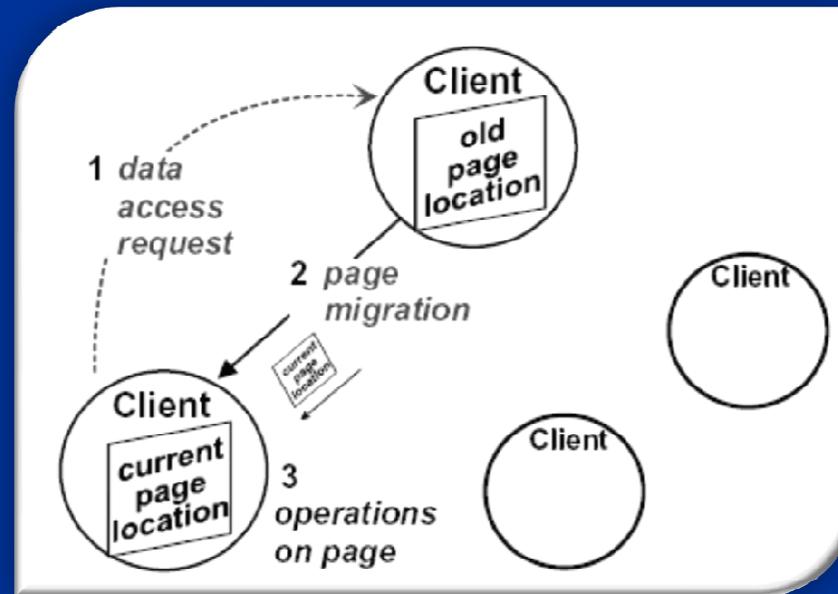
# Implementação

- Implementação Centralizada
  - As páginas de memória compartilhadas são mantidos em um servidor central, que executa as operações requisitadas pelos clientes



# Implementação

- Implementação por Migração
  - Páginas compartilhadas são migradas de um nó para outro; nó de posse da página executa localmente as operações desejadas



# Implementação

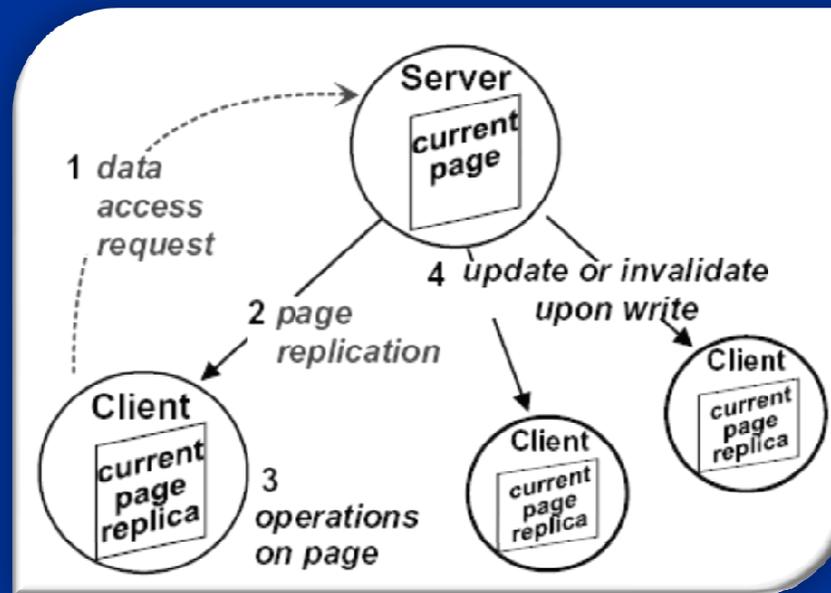
- Estratégias baseadas em Replicação
  - Cada nó mantém uma cópia local dos dados
  - Mais eficientes que as estratégias centralizada e por migração, pois permitem o acesso concorrente aos dados
  - Exigem extremo cuidado para manter os dados consistentes e sincronizados
  - Mecanismos de controle de concorrência (*Locks*, monitores, semáforos...) devem ser usados para garantir a consistência

# Implementação

- Estratégias baseadas em Replicação (cont.)
  - Trocas de mensagens são efetuadas para manter os dados sincronizados
  - Quando um dado for alterado por um nó, duas estratégias podem ser empregadas para sincronização das réplicas
    - Atualizar todas as réplicas
    - Invalidar as demais réplicas, obrigando que sejam atualizadas antes do próximo acesso

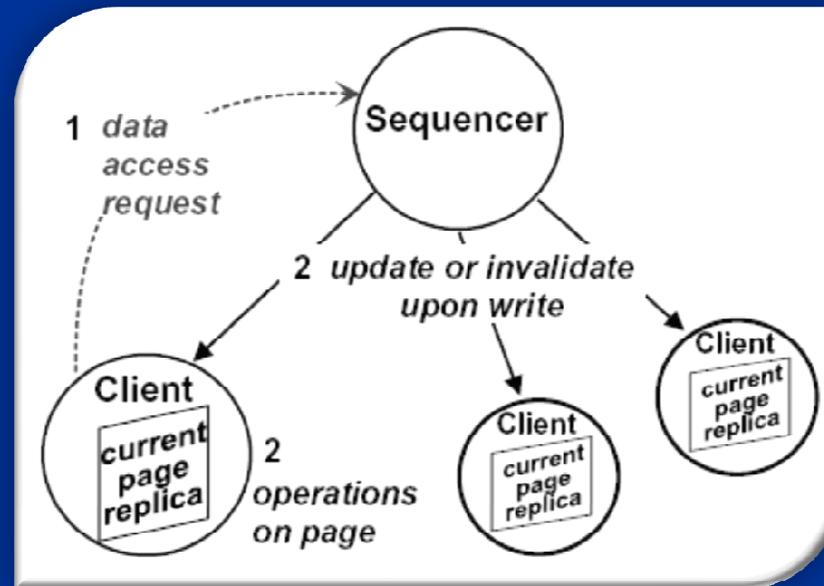
# Implementação

- Replicação Somente-Leitura
  - Dados podem ser lidos concorrentemente
  - Alterações nos dados são coordenadas por um servidor, que atualiza/invalida as réplicas



# Implementação

- Replicação Leitura/Escrita
  - Dados são lidos e alterados em paralelo
  - Seqüenciador é responsável por atualizar/invalidar cópias desatualizadas



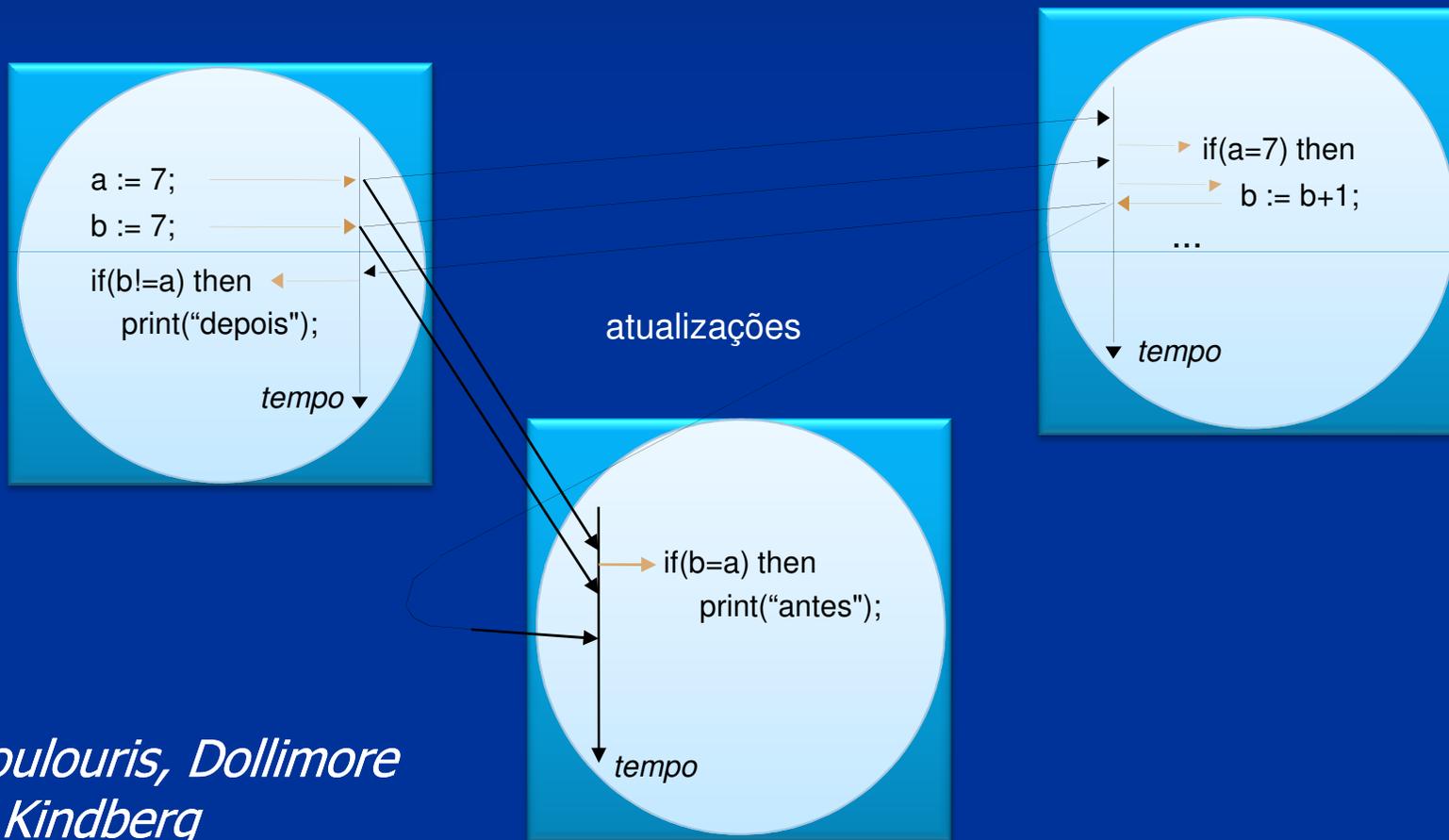
# Implementação

## ■ Comunicação

- A comunicação entre os nós pode afetar negativamente o desempenho das aplicações, principalmente quando for usada replicação
- Estratégias para diminuir a quantidade de mensagens enviadas pela rede:
  - Aguardar que um nó faça todas as alterações necessárias nos dados antes de propagá-las
  - Usar multicast para enviar atualizações
- O retardo causado pela propagação de mensagens na rede pode resultar em inconsistências momentâneas

# Implementação

## ■ Comunicação: efeitos do retardo



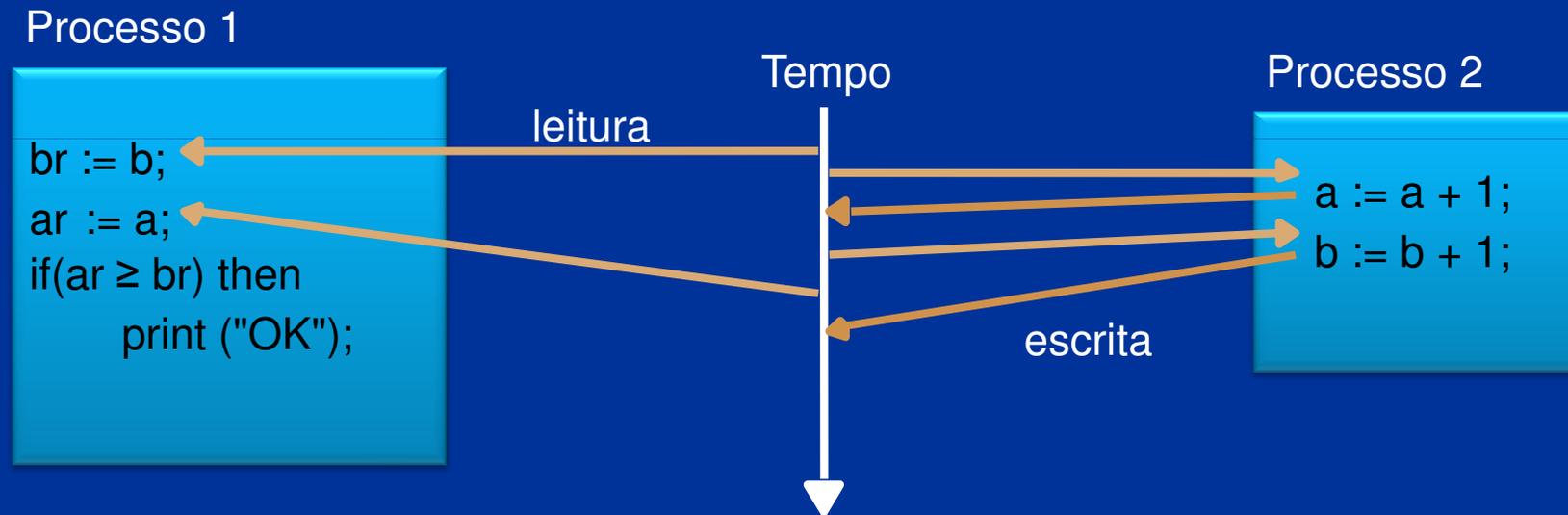
© Coulouris, Dollimore  
& Kindberg

# Implementação

- **Consistência**
  - A manutenção da consistência dos dados é crucial para o funcionamento da DSM
  - Consistência seqüencial garante que todos os nós recebem as alterações na mesma ordem
  - Garantir consistência seqüencial é dispendioso pois requer controle de concorrência e ordenação de mensagens
  - Em alguns casos, certo grau de inconsistência é tolerado para obter um melhor desempenho no acesso à memória compartilhada

# Implementação

## ■ Consistência Seqüencial



# Produtos

- Linda [*Gelernter*, 1985]
  - Linguagem que criou o conceito de espaço de tuplas (*tuplespace*) distribuído
  - Principais operações:
    - **In**: lê uma tupla e a remove do *tuplespace*
    - **Read**: lê uma tupla (sem removê-la)
    - **Out**: escreve uma tupla no *tuplespace*
  - Implementada como biblioteca para outras linguagens: C, C++, Java, Python, Ruby, ...
  - Estendida para suportar múltiplos espaços de tuplas, notificação, definição de tipos/classes de tuplas, coleta de lixo, *pattern matching*, etc.

# Produtos

## ■ JavaSpaces

- Implementação de espaço de tuplas em Java
- Parte da tecnologia Jini (Sun/Apache)
- Clientes conectam a servidores JavaSpaces
- Tuplas são objetos Java que implementam a interface *net.jini.core.Entry*
- Tuplas possuem um tempo de vida (*lease*)
- *Templates* (tuplas que podem ter alguns campos *null*) são usados na leitura/remoção
- Acesso ao espaço de tuplas pode ser feito no escopo de uma transação atômica
- Suporta notificações de inserção de tuplas

# Produtos

- Métodos do JavaSpace

- Escrita:

- write(tupla, transaction, lease);*

- Leitura de uma tupla, sem remoção:

- read(template, transaction, timeout);*

- readIfExists(template, transaction, txTimeout);*

- Leitura de uma tupla, com remoção:

- take(template, transaction, timeout);*

- takeIfExists(template, transaction, txTimeout);*

- Notificação:

- notify(template, transact, listener, timeout, id);*

# Produtos

- Outros Produtos
  - DIPC, Kerrighed
    - Alteram núcleo do Linux para suportar DSM
  - GigaSpaces
    - Implementação comercial do JavaSpaces, com espaços de tuplas replicados
  - TreadMarks
    - Implementação de DSM com suporte a diversos sistemas operacionais
  - ...