6 – Comunicação Indireta

Prof. Marco Aurélio S Birchal

PUC Minas

6 - Introdução

Paradigmas de Comunicação:

- comunicação entre processos (cap.4);
- invocação remota (cap.5);
- **comunicação indireta**

6 - Introdução

Comunicação Indireta:

- **comunicação em grupo**, abstração de grupo, sem que o remetente saiba a identidade dos destinatários;
- **sistemas publicar-assinar (publish-subscriber),** estratégias de disseminar eventos para vários destinatários por meio de um intermediário;
- **sistemas de fila de mensagens**, abstração de fila, com os destinatários extraindo mensagens dessas filas;
- **memória compartilhada**, uma abstração de memória compartilhada global para os programadores.

6.1 - Introdução

Indireção (coneito):

“ Todos os problemas na ciência da computação podem ser resolvidos por outro nível de indireção.”

Roger Needham, Maurice Wilkes e David Wheeler (Cambridge)

Comunicação Indireta é definida como a comunicação entre entidades de um sistema distribuído por meio de um intermediário, sem nenhum acoplamento direto entre o remetente e o destinatário (ou destinatários).

6.1 - Introdução

DESACOPLAMENTO

Desacoplamento espacial, no qual o remetente não sabe ou não precisa saber a identidade do destinatário (ou destinatários) e vice-versa. Por causa do desacopla- mento espacial, o desenvolvedor de sistema tem muitos graus de liberdade para lidar com alterações: os participantes (remetentes ou destinatários) podem ser substituídos, atualizados, duplicados ou migrados.

Desacoplamento temporal, no qual o remetente e o destinatário (ou destinatários) podem ter tempos de vida independentes. Em outras palavras, o remetente e o destinatário (ou destinatários) não precisam existir ao mesmo tempo para se comunicar. Isso tem vantagens importantes, por exemplo, em ambientes mais voláteis, onde remetentes e destinatários podem ir e vir.

6.1 - Introdução

	Acoplamento temporal	*Desacoplamento temporal*
Acoplamento espacial	*Propriedades:* comunicação direcionada para determinado destinatário (ou destinatários); o destinatário (ou destinatários) deve existir nesse momento no tempo. *Exemplos:* passagem de mensagens, invocação remota (consulte os Capítulos 4 e 5).	*Propriedades:* comunicação direcionada para determinado destinatário (ou destinatários); o remetente (ou remetentes) e o destinatário (ou destinatários) podem ter tempos de vida independentes. *Exemplos:* consulte o Exercício 6.3.
Desacoplamento espacial	*Propriedades:* o remetente não precisa conhecer a identidade do destinatário (ou destinatários); o destinatário (ou destinatários) deve existir nesse momento no tempo. *Exemplos: multicast* IP (consulte o Capítulo 4).	*Propriedades:* o remetente não precisa conhecer a identidade do destinatário (ou destinatários); o remetente (ou remetentes) e o destinatário (ou destinatários) podem ter tempos de vida independentes. *Exemplos:* a maioria dos paradigmas de comunicação indireta abordados neste capítulo.

Figura 6.1 Acoplamento espacial e temporal em sistemas distribuídos.

6.1 - Introdução

Maioria dos paradigmas de Comunicação Direta (comunic. entre processos e invocação remota)

	Acoplamento temporal	*Desacoplamento temporal*
Acoplamento espacial	*Propriedades:* comunicação direcionada para determinado destinatário (ou destinatários); o destinatário (ou destinatários) deve existir nesse momento no tempo. *Exemplos:* passagem de mensagens, invocação remota (consulte os Capítulos 4 e 5).	*Propriedades:* comunicação direcionada para determinado destinatário (ou destinatários); o remetente (ou remetentes) e o destinatário (ou destinatários) podem ter tempos de vida independentes. *Exemplos:* consulte o Exercício 6.3.
Desacoplamento espacial	*Propriedades:* o remetente não precisa conhecer a identidade do destinatário (ou destinatários); o destinatário (ou destinatários) deve existir nesse momento no tempo. *Exemplos: multicast* IP (consulte o Capítulo 4).	*Propriedades:* o remetente não precisa conhecer a identidade do destinatário (ou destinatários); o remetente (ou remetentes) e o destinatário (ou destinatários) podem ter tempos de vida independentes. *Exemplos:* a maioria dos paradigmas de comunicação indireta abordados neste capítulo.

Figura 6.1 Acoplamento espacial e temporal em sistemas distribuídos.

6.1 - Introdução

Maioria dos paradigmas de comunicação indireta

	Acoplamento temporal	*Desacoplamento temporal*
Acoplamento espacial	*Propriedades:* comunicação direcionada para determinado destinatário (ou destinatários); o destinatário (ou destinatários) deve existir nesse momento no tempo. *Exemplos:* passagem de mensagens, invocação remota (consulte os Capítulos 4 e 5).	*Propriedades:* comunicação direcionada para determinado destinatário (ou destinatários); o remetente (ou remetentes) e o destinatário (ou destinatários) podem ter tempos de vida independentes. *Exemplos:* consulte o Exercício 6.3.
Desacoplamento espacial	*Propriedades:* o remetente não precisa conhecer a identidade do destinatário (ou destinatários); o destinatário (ou destinatários) deve existir nesse momento no tempo. *Exemplos: multicast* IP (consulte o Capítulo 4).	*Propriedades:* o remetente não precisa conhecer a identidade do destinatário (ou destinatários); o remetente (ou remetentes) e o destinatário (ou destinatários) podem ter tempos de vida independentes. *Exemplos:* a maioria dos paradigmas de comunicação indireta abordados neste capítulo.

Figura 6.1 Acoplamento espacial e temporal em sistemas distribuídos.

6.1 - Introdução

Multicast IP (comunicação direta entre processos em grupo)

	Acoplamento temporal	*Desacoplamento temporal*
Acoplamento espacial	*Propriedades:* comunicação direcionada para determinado destinatário (ou destinatários); o destinatário (ou destinatários) deve existir nesse momento no tempo. *Exemplos:* passagem de mensagens, invocação remota (consulte os Capítulos 4 e 5).	*Propriedades:* comunicação direcionada para determinado destinatário (ou destinatários); o remetente (ou remetentes) e o destinatário (ou destinatários) podem ter tempos de vida independentes. *Exemplos:* consulte o Exercício 6.3.
Desacoplamento espacial	*Propriedades:* o remetente não precisa conhecer a identidade do destinatário (ou destinatários); o destinatário (ou destinatários) deve existir nesse momento no tempo. *Exemplos: multicast* IP (consulte o Capítulo 4).	*Propriedades:* o remetente não precisa conhecer a identidade do destinatário (ou destinatários); o remetente (ou remetentes) e o destinatário (ou destinatários) podem ter tempos de vida independentes. *Exemplos:* a maioria dos paradigmas de comunicação indireta abordados neste capítulo.

Figura 6.1 Acoplamento espacial e temporal em sistemas distribuídos.

6.1 - Introdução

email

	Acoplamento temporal	*Desacoplamento temporal*
Acoplamento espacial	*Propriedades:* comunicação direcionada para determinado destinatário (ou destinatários); o destinatário (ou destinatários) deve existir nesse momento no tempo. *Exemplos:* passagem de mensagens, invocação remota (consulte os Capítulos 4 e 5).	*Propriedades:* comunicação direcionada para determinado destinatário (ou destinatários); o remetente (ou remetentes) e o destinatário (ou destinatários) podem ter tempos de vida independentes. *Exemplos:* consulte o Exercício 6.3.
Desacoplamento espacial	*Propriedades:* o remetente não precisa conhecer a identidade do destinatário (ou destinatários); o destinatário (ou destinatários) deve existir nesse momento no tempo. *Exemplos: multicast* IP (consulte o Capítulo 4).	*Propriedades:* o remetente não precisa conhecer a identidade do destinatário (ou destinatários); o remetente (ou remetentes) e o destinatário (ou destinatários) podem ter tempos de vida independentes. *Exemplos:* a maioria dos paradigmas de comunicação indireta abordados neste capítulo.

Figura 6.1 Acoplamento espacial e temporal em sistemas distribuídos.

6.2 – Comunicação em Grupo

Introdução

- Serviço por meio do qual uma mensagem é enviada para um grupo e, então, entregue a todos os membros do grupo.
- O remetente não conhece a identidade dos destinatários.
- A comunicação em grupo representa uma abstração em relação à comunicação por multicast melhorando o gerenciamento de participantes do grupo e a detecção de falhas e as garantias de confiabilidade e ordenação.
- Com as garantias reforçadas, a comunicação em grupo está para o multicast IP assim como o TCP está para o serviço ponto a ponto em IP.

6.2 – Comunicação em Grupo

Aplicações

- disseminação confiável de informações para números grandes de clientes, incluindo o setor financeiro;
- suporte para aplicativos colaborativos, em que, os eventos precisam ser disseminados para vários usuários preservando uma visão comum (jogos multiusuário);
- suporte à estratégias de tolerância a falhas, como replicação de dados ou de servidores altamente disponíveis;
- suporte para monitoramento e gerenciamento de sistemas (balanceamento de carga).

6.2 – Comunicação em Grupo

6.2.1. O modelo de programação multicast

- **conceito central** é o de um grupo com atribuições de membros associadas por meio das quais os processos podem ingressar no grupo ou sair dele;
    - *Multicast* – envio para um grupo
    - *Broadcast* – envio para todos
    - *Unicast* – envio para um único membro
- A característica fundamental da comunicação em grupo é que um processo executa somente uma operação de *multicast* para enviar uma mensagem para cada processo de um grupo de processos (em vez de executar várias operações de envio para processos individuais).

6.2 – Comunicação em Grupo

6.2.1. O modelo de programação multicast
Grupos de Processos e Grupos de Objetos

- **grupos de processos**: grupos em que as entidades que se comunicam são processos. Esses serviços são de nível relativamente baixo:
  - as mensagens são entregues para processos e nenhum outro suporte para entrega é fornecido.
  - as mensagens são vetores de byte não estruturados, sem suporte para empacotamento de tipos de dados complexos (conforme o fornecido, por exemplo, em RPC ou RMI).

6.2 – Comunicação em Grupo

6.2.1. O modelo de programação multicast
Grupos de Processos e Grupos de Objetos

- **grupos de objetos**: fornecem uma estratégia de nível mais alto para a computação em grupo.
- é um conjunto de objetos (normalmente instâncias da mesma classe) que processam o mesmo conjunto de invocações concorrentemente, com cada um retornando respostas.
- Os objetos clientes não precisam estar cientes da replicação.
- Eles ativam operações em um único objeto local, o qual atua como proxy para o grupo. O proxy usa um sistema de comunicação em grupo para enviar as invocações para os membros do grupo de objetos.

6.2 – Comunicação em Grupo

6.2.1. O modelo de programação multicast
Grupos Fechados e Grupos Abertos

- grupo fechado: se somente membros do grupo podem enviar mensagem para ele.
- grupo é aberto: se processos de fora do grupo podem enviar mensagens para ele.
- Os grupos fechados de processos são úteis, por exemplo, para servidores em cooperação enviarem, uns para os outros, mensagens que somente eles devem receber. Os grupos abertos são úteis, por exemplo, para entregar eventos para grupos de processos interessados.

6.2 – Comunicação em Grupo

6.2.1. O modelo de programação multicast

Grupos Fechados e Grupos Abertos

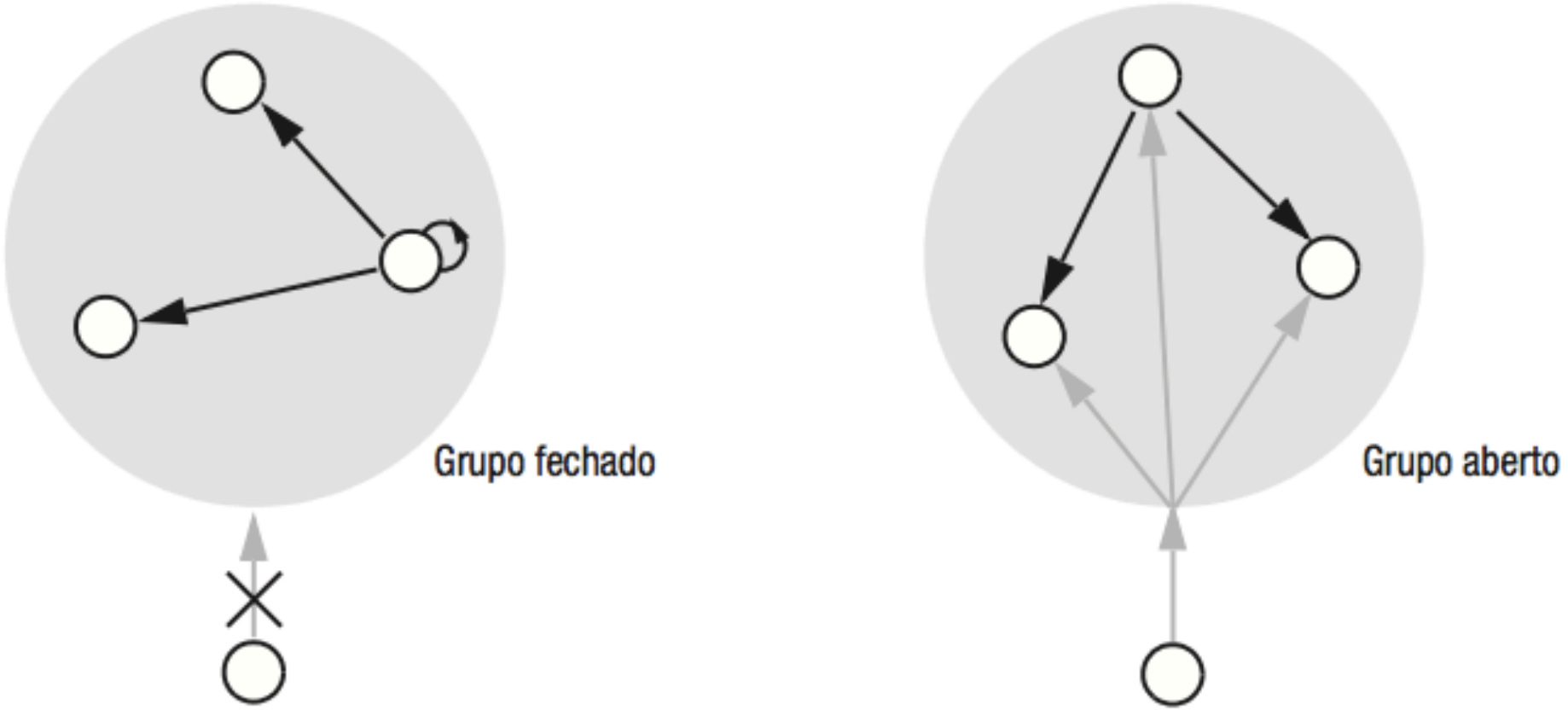


Figura 6.2 Grupos abertos e fechados.

OBS: Um processo em um grupo fechado entrega para si mesmo qualquer mensagem que envie para o grupo.

6.2 – Comunicação em Grupo

6.2.2. Problemas de Implementação

- **Confiabilidade e ordenação em multicast**: todos os membros devem receber cópias das mensagens enviadas para o grupo, geralmente com garantias de entrega.

- As garantias incluem acordo sobre o conjunto de mensagens que todo processo do grupo deve receber e sobre a ordem de entrega para os membros do grupo.

6.2 – Comunicação em Grupo

6.2.2. Problemas de Implementação

Confiabilidade e ordenação em multicast

- Ordem FIFO: ou ordem de origem – preserva a ordem da perspectiva de um processo remetente; se um processo enviar uma mensagem antes de outro, então ela vai ser entregue nessa ordem em todos os processos do grupo.
- Ordem causal: leva em conta as relações causais entre as mensagens; se uma mensagem acontece antes de outra, essa relação causal vai ser preservada na entrega das mensagens em todos os processos
- Ordem total: se uma mensagem for entregue antes de outra em um processo, então a mesma ordem vai ser preservada em todos os processos.

6.2 – Comunicação em Grupo

6.2.2. Problemas de Implementação

- **Gerenciamento da participação no grupo**: Um serviço de participação como membro de um grupo tem quatro tarefas principais:
- Fornecer uma interface para mudanças de participação como membro do grupo: fornece operações para criar e destruir grupos de processos e para adicionar ou retirar um processo de um grupo.
- Detecção de falha: monitora (colapso, falha de comunicação) e marca como processos como Suspeitos ou Não suspeitos e decide pela participação ou exclusão do grupo.
- Notificar mudanças de participação no grupo: quando um processo é adicionado ou quando um processo é excluído.
- Realizar expansão de endereço de grupo: coordenar a entrega mesmo com mudanças na participação de membros no grupo. Isto é, ele pode decidir coerentemente onde vai entregar determinada mensagem, mesmo que a participação como membro tenha mudado durante a entrega.

6.2 – Comunicação em Grupo

6.2.2. Problemas de Implementação

Gerenciamento da participação no grupo

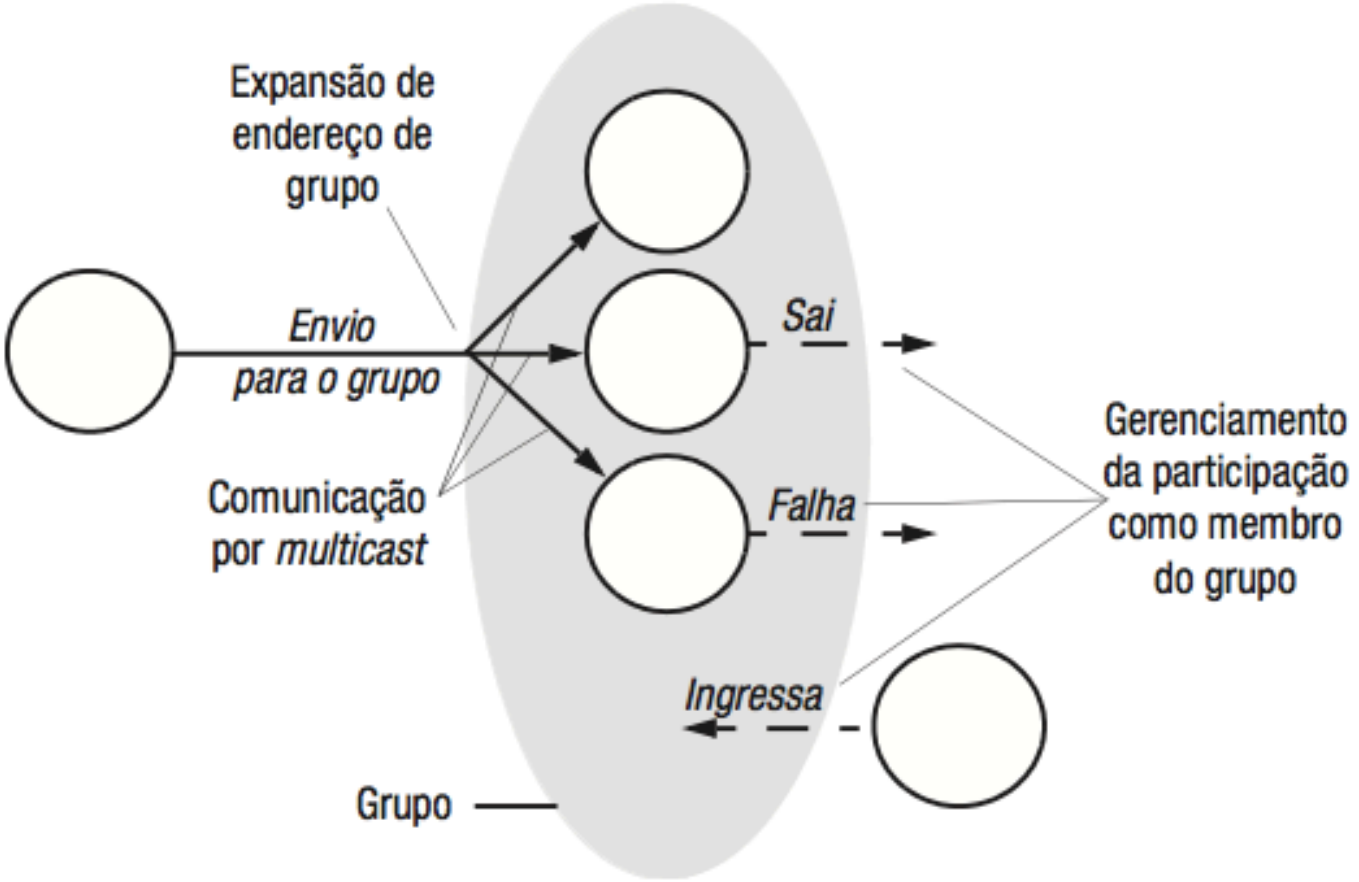


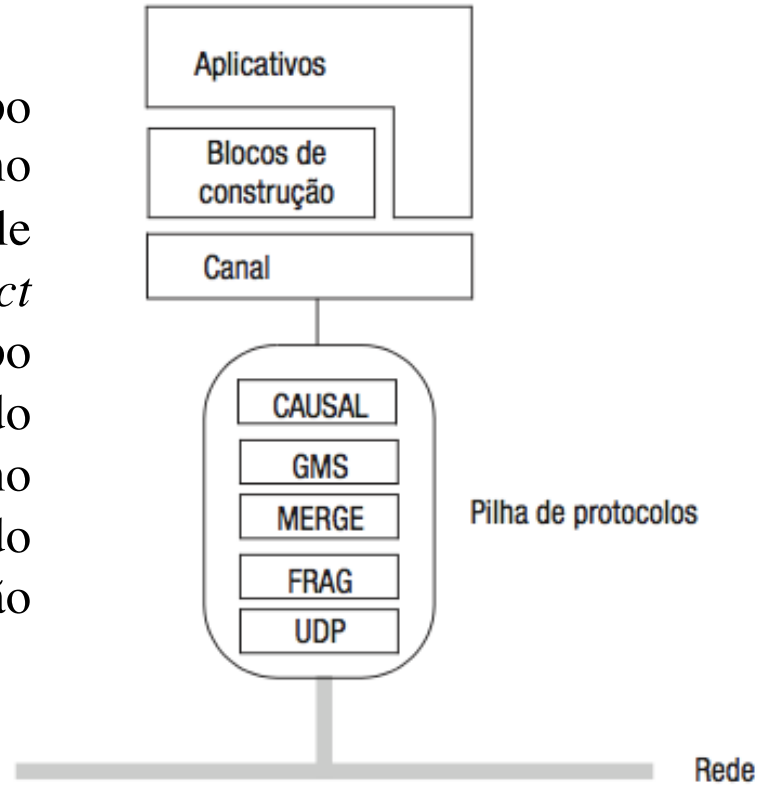
Figura 6.3 O papel do gerenciamento da participação como membro do grupo.

6.2 – Comunicação em Grupo

6.2.3. Estudo de caso – toolkit JGroups

- kit de ferramentas para comunicação em grupo confiável, escrito em Java. O kit faz parte da linhagem de ferramentas de comunicação em grupo desenvol- vidas na Cornell University

Canais • Um processo interage com um grupo por meio de um objeto *canal*, o qual atua como um tratador para um grupo. Quando criado, ele está desconectado, mas uma operação *connect* subsequente vincula esse tratador a um grupo nomeado em particular; se o grupo nomeado não existe, ele é criado implicitamente no momento da primeira conexão. Para sair do grupo, o processo executa a operação *disconnect* correspondente.



A arquitetura do JGroups.

6.2 – Comunicação em Grupo

6.2.3. Estudo de caso – toolkit JGroups

- Exemplo: um serviço por meio do qual um alarme de incêndio inteligente pode enviar uma mensagem “Fire!” por multicast para quaisquer destinatários registrados.
- O código do alarme de incêndio está mostrado:

```
import org.jgroups.JChannel;

public class FireAlarmJG {
    public void raise( ) {
        try {
            JChannel channel = new JChannel( );
            channel.connect("AlarmChannel");
            Message msg = new Message(null, null, "Fire!");
            channel.send(msg);
        }
        catch(Exception e) {
        }
    }
}
```

Classe Java *FireAlarmJG*.

O código para criar uma nova instância da classe *FireAlarmJG* e, então, disparar um alarme, seria:

```
FireAlarmJG alarm = new FireAlarmJG();
alarm.raise();
```

6.2 – Comunicação em Grupo

6.2.3. Estudo de caso – toolkit JGroups

- Exemplo: um serviço de alarme de incêndio inteligente

```
import org.jgroups.JChannel;

public class FireAlarmConsumerJG {
    public String await( ) {
        try {
            JChannel channel = new JChannel( );
            channel.connect("AlarmChannel");
            Message msg = (Message) channel.receive(0);
            return (String) msg.GetObject( );
        }
        catch(Exception e) {
            return null;
        }
    }
}
```

Classe Java *FireAlarmConsumerJG*.

Código do destinatário para esperar um alarme:

```
FireAlarmConsumerJG alarmCall = new FireAlarmConsumerJG();
String msg = alarmCall.await();
System.out.println("Alarm received: " + msg);
```

6.3 Sistemas publicar-assinar

- Um sistema baseado em publicar-assinar (publish-subscribe) é um sistema em que publicadores divulgam eventos estruturados para um serviço de evento e assinantes expressam interesse em eventos específicos por meio de assinaturas.
- A tarefa do sistema publicar-assinar é combinar as assinaturas com os eventos publicados e garantir a entrega correta de notificações de evento.
- Determinado evento vai ser entregue possivelmente para muitos assinantes e, assim, o publicar-assinar é, fundamentalmente, um paradigma de comunicação de um para muitos.

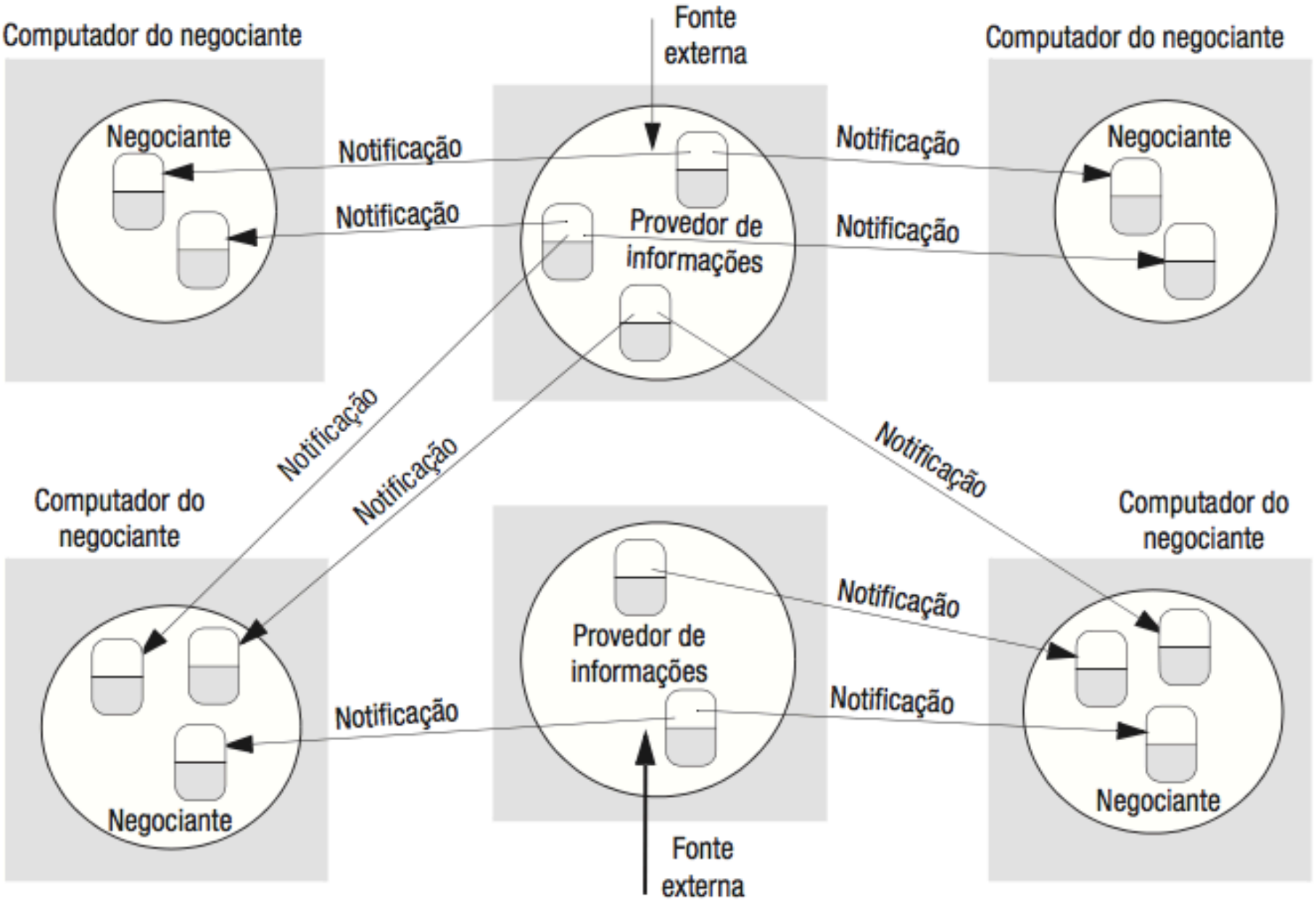
6.3 Sistemas publicar-assinar

Exemplo 1: sistema simples de sala de negociação.

- Um processo provedor de informação recebe continuamente novas informações de negócio de uma única fonte externa. Cada uma das atualizações é considerada um evento. O provedor de informação publica esses eventos no sistema publicar-assinar para entrega a todos os negociantes que tenham expressado interesse na mercadoria correspondente. Haverá um processo provedor de informação separado para cada fonte externa.
- Um processo negociante cria uma assinatura representando cada mercadoria nomeada que o usuário pede para ser exibida. Essa assinatura expressa o interesse em eventos relacionados a determinada mercadoria no provedor de informação

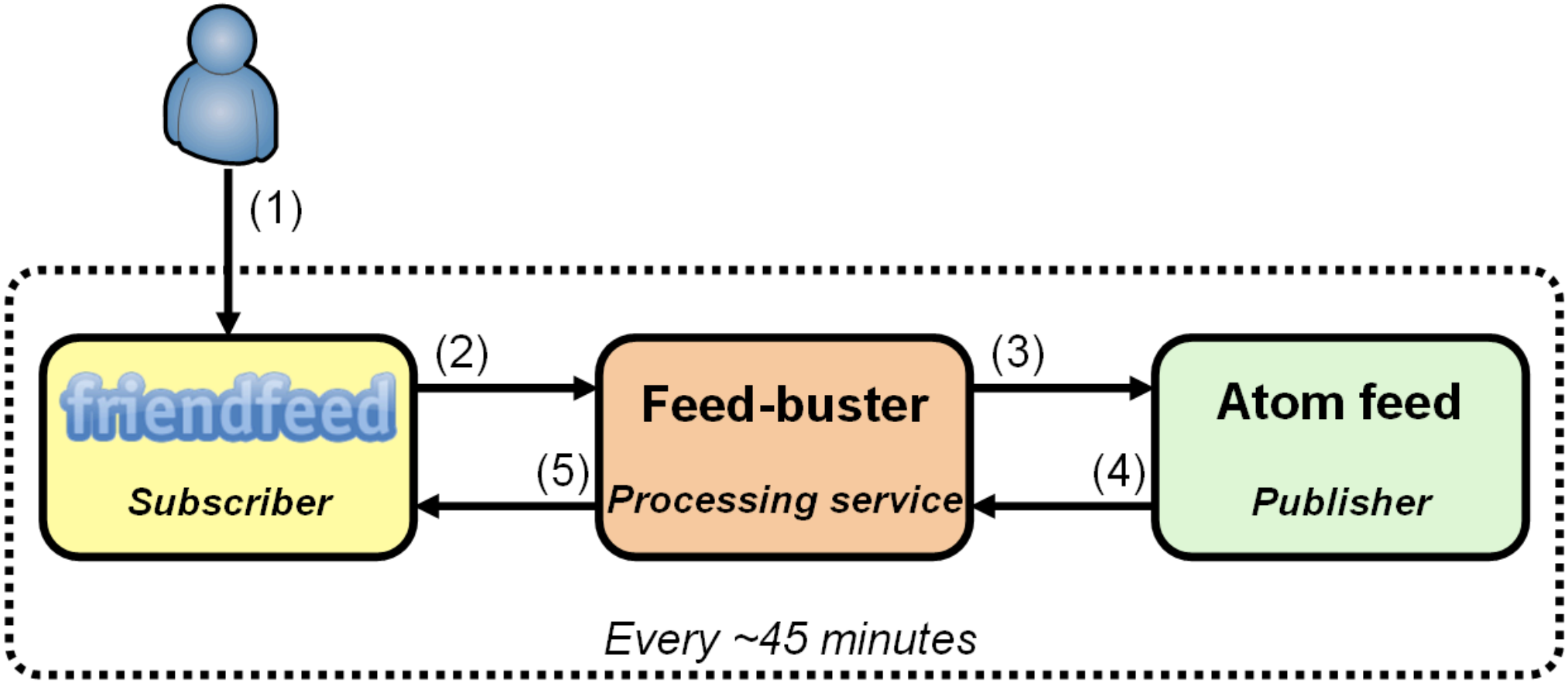
6.3 Sistemas publicar-assinar

Exemplo 1: sistema simples de sala de negociação.



6.3 Sistemas publicar-assinar

Exemplo 2: RSS - Real-time feed processing and filtering



(1) O usuário insere um link para o processo assinante de mensagens subscriber (2) O assinante faz uma requisição ao serviço diretório de serviços de mensagens (3) O serviço diretório localiza o publicador de mensagem Publisher que (4) devolve a mensagem para o diretório (5) que a entrega ao assinante.

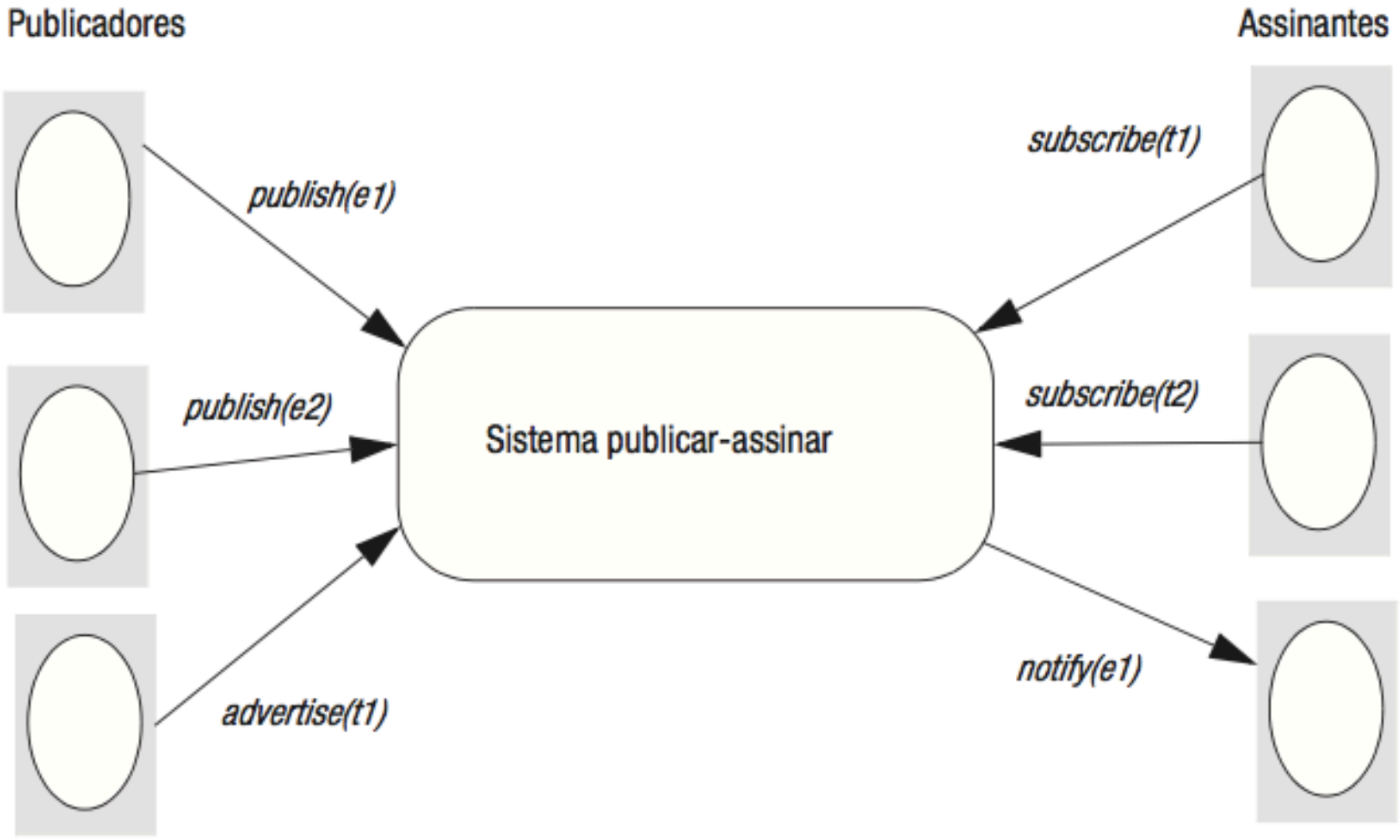
6.3 Sistemas publicar-assinar Características

- Heterogeneidade: quando notificações de evento são usadas como meio de comunicação, pode-se fazer com que componentes de um sistema distribuído que não foram projetados para operação conjunta funcionem juntos.
- Assíncronos: as notificações são enviadas de forma assíncrona, pelos publicadores que geram eventos, a todos os assinantes que expressaram interesse neles. Para evitar que os publicadores precisem estar sincronizados com os assinantes – os publicadores e os assinantes precisam estar desacoplados.

6.3 Sistemas publicar-assinar
6.3.1. Modelo de Programação

- Baseado em um pequeno conjunto de operações:
- e – evento
- publish(e) – publicador dissemina o evento e
- subscribe(f) – assinante assinam um certo conjunto de eventos, atrvés de um filtro de eventos f
- notify(e) – o sistema entrega um novo evento ao assinante
- unsubscribe(f) – o assinante revoga a assinatura f

6.3 Sistemas publicar-assinar
6.3.1. Modelo de Programação



O paradigma publicar - assinar

6.3 Sistemas publicar-assinar

6.3.1. Modelo de Programação

Modelos de Assinatura (filtros)

a expressividade dos sistemas publicar-assinar é determinada pelo modelo de assinatura (filtro), com vários esquemas definidos e considerados aqui em ordem crescente de sofisticação:

1. Baseado em canal
2. Baseado em tópico (ou baseado em assunto)
3. Baseado em Conteúdo
4. Baseado em tipo

6.3 Sistemas publicar-assinar
6.3.1. Modelo de Programação
Modelos de Assinatura (filtros)

1. *Baseado em canal:* publicadores divulgam eventos para canais nomeados; os assinantes se inscrevem em um deles para receber todos os eventos enviados para esse canal. Esse é um esquema bastante primitivo e o único que define um canal físico;

2. *Baseado em tópico (ou baseado em assunto):* nesta estratégia, supomos que cada notificação é expressa em termos de vários campos, com um deles denotando o tópico., As assinaturas são por tópico de interesse. É equivalente às estratégias baseadas em canal, com a diferença que os tópicos são definidos implicitamente no caso dos canais, mas declarados explicitamente como um dos campos nas estratégias baseadas em tópico.

6.3 Sistemas publicar-assinar

6.3.1. Modelo de Programação

Modelos de Assinatura (filtros)

3. *Baseado em conteúdo*: generalização das baseadas em tópico, permitindo a expressão de assinaturas sobre diversos campos em uma notificação de evento. Um filtro baseado em conteúdo é uma consulta definida em termos de composições de restrições sobre os valores de atributos de evento. essa estratégia é significativamente mais expressiva do que as estratégias baseadas em canal ou em tópico, porém com significativos novos desafios.
4. *Baseado em tipo*: ligadas às estratégias baseadas em objeto. As assinaturas são definidas em termos de tipos de eventos e a combinação é definida em termos de tipos ou subtipos do filtro dado. Têm expressividade semelhante às estratégias baseadas em conteúdo, com a vantagem de podem verificar a exatidão do tipo das assinaturas, eliminando alguns tipos de erros de assinatura.

6.3 Sistemas publicar-assinar

6.3.2 Problemas de implementação

Implementações centralizadas versus distribuídas

Centralizadas

- mais simples
- centralizar a implementação com um servidor intermediário de evento em um único nó.
- Os publicadores divulgam eventos para esse intermediário, e os assinantes enviam assinaturas para ele e recebem notificações em resposta.
- O serviço central mantém um repositório de assinaturas e faz a correspondência das notificações de evento com esse conjunto de assinaturas.
- A interação com o intermediário é por meio de uma série de mensagens ponto a ponto (por passagem de mensagens ou invocação remota)

6.3 Sistemas publicar-assinar

6.3.2 Problemas de implementação

Implementações centralizadas versus distribuídas

Distribuídas

- Rede de intermediários, que coopera para oferecer a funcionalidade desejada.
- Têm o potencial de sobreviver à falha do nó e têm-se mostrado capazes de funcionar bem em implantações na Internet.

- NOTA: Levando isso um passo adiante, é possível ter uma implementação totalmente *peer-to-peer* em que não há distinção entre publicadores, assinantes e intermediários; todos os nós atuam como intermediários, implementando cooperativamente a funcionalidade de roteamento de evento exigida.

6.3 Sistemas publicar-assinar
6.3.2 Problemas de implementação

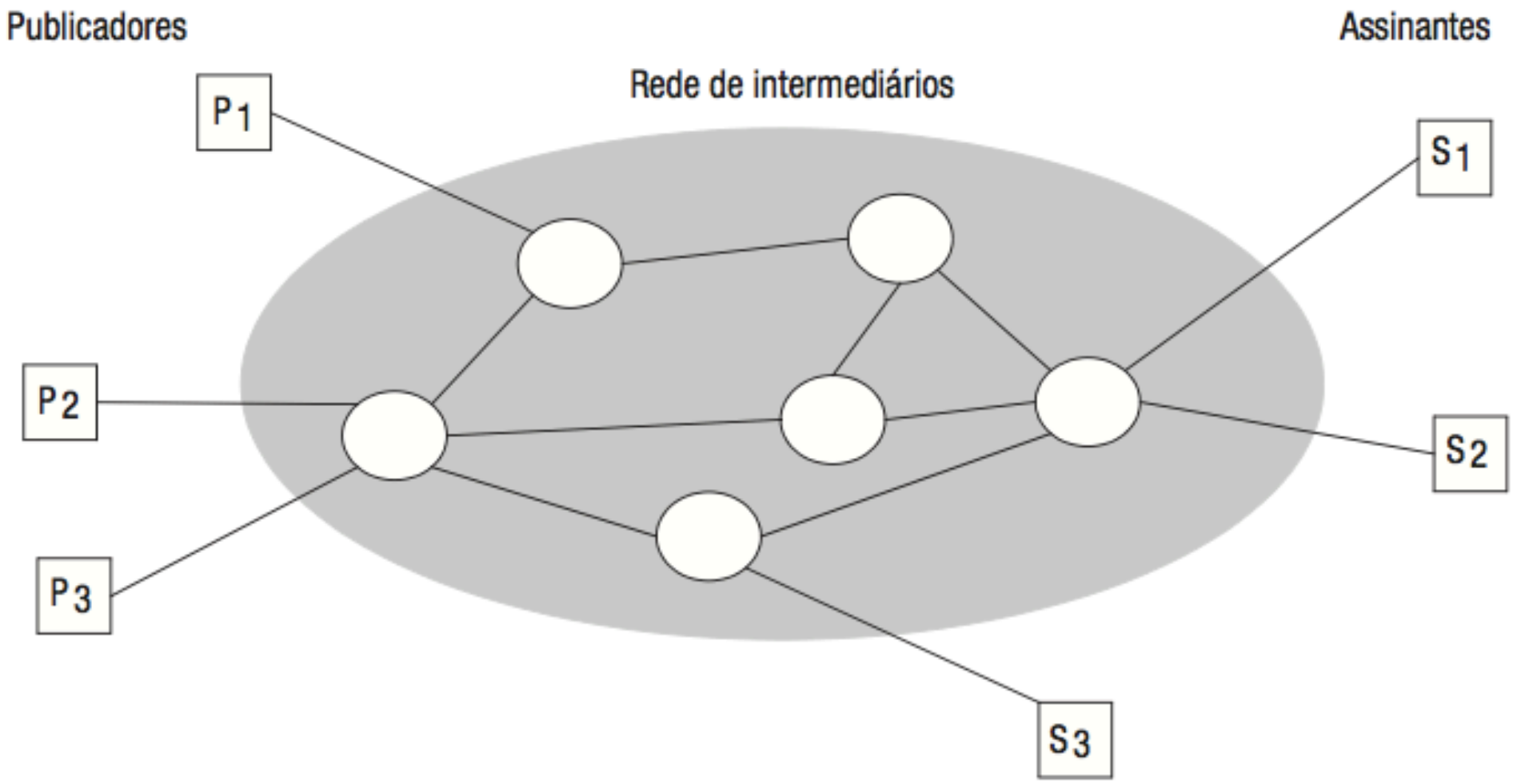


Figura 6.9 A rede de intermediários.

6.3 Sistemas publicar-assinar

6.3.2 Problemas de implementação

Arquitetura de sistemas global (para estratégias baseadas em conteúdo)

- Implementações baseadas em conteúdo são mais complexas e exibem várias possibilidades de implementação.

- Camadas:
  - Protocolos de rede
  - Redes de sobreposição
  - Roteamento de evento

6.3 Sistemas publicar-assinar

6.3.2 Problemas de implementação

Arquitetura de sistemas global
(para estratégias baseadas em conteúdo)

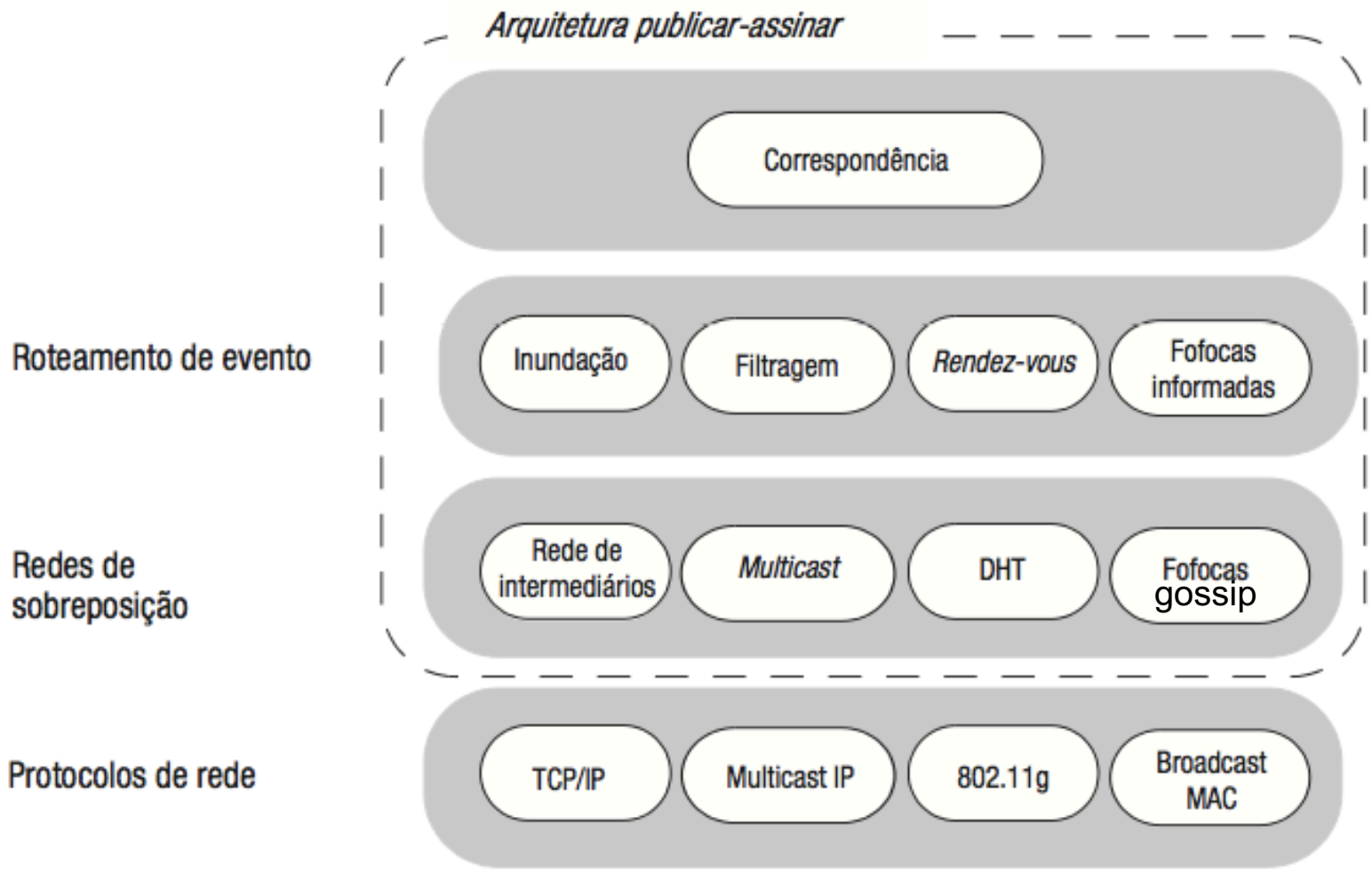


Figura 6.10 A arquitetura de sistemas publicar-assinar.

6.3 Sistemas publicar-assinar

6.3.2 Problemas de implementação

Arquitetura de sistemas global
(para estratégias baseadas em conteúdo)

- camada inferior.
- uso de diversos serviços de comunicação entre processos, como TCP/IP, *multicast* IP (onde estiver disponível) ou serviços mais especializados,, por exemplo, pelas redes sem fio.

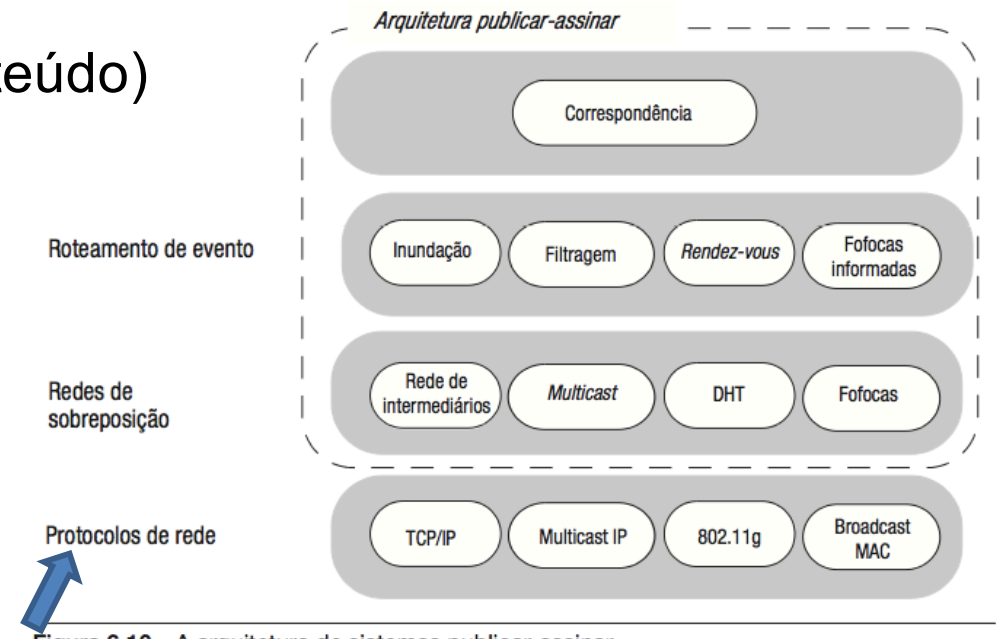


Figura 6.10 A arquitetura de sistemas publicar-assinar.

6.3 Sistemas publicar-assinar

6.3.2 Problemas de implementação

Arquitetura de sistemas global
(para estratégias baseadas em conteúdo)

- configura redes de intermediários ou estruturas peer-to-peer adequadas.

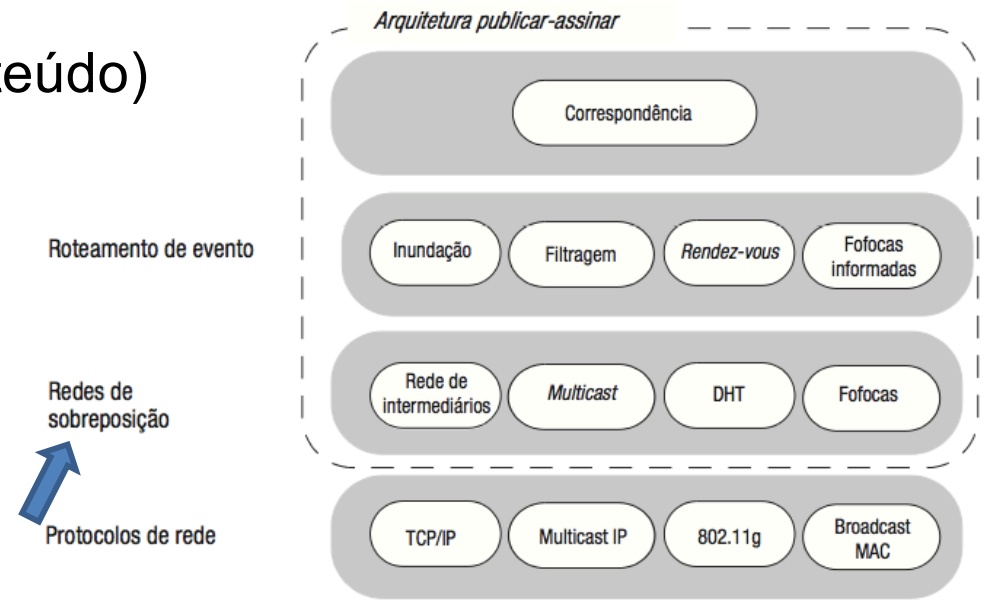


Figura 6.10 A arquitetura de sistemas publicar-assinar.

6.3 Sistemas publicar-assinar

6.3.2 Problemas de implementação

Arquitetura de sistemas global
(para estratégias baseadas em conteúdo)

- A parte principal da arquitetura.

- executa a tarefa de garantir que as notificações de evento sejam roteadas da forma mais eficiente possível para os assinantes apropriados

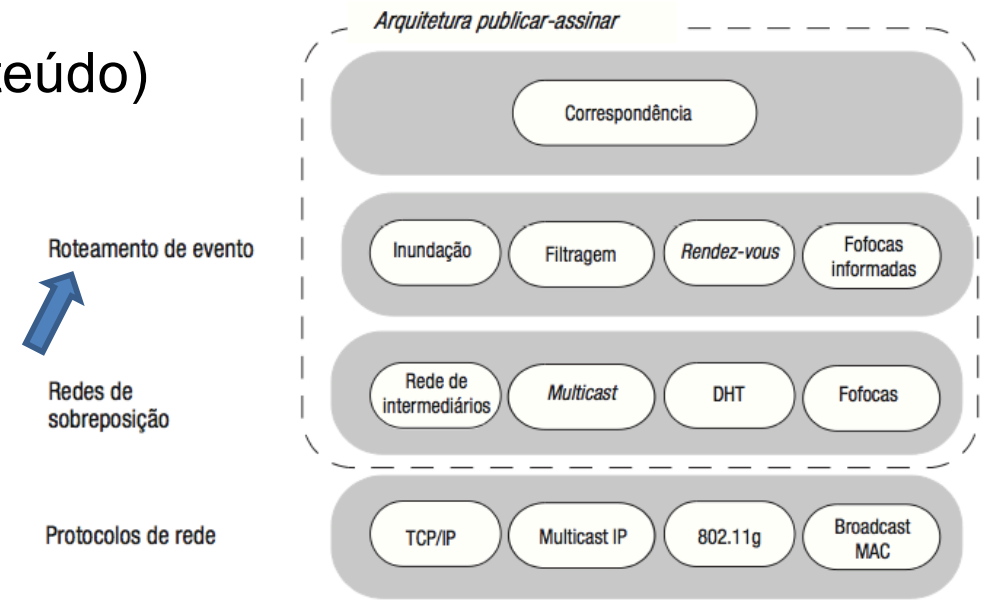


Figura 6.10 A arquitetura de sistemas publicar-assinar.

- Para estratégias baseadas em conteúdo, esse problema é referido como roteamento baseado em conteúdo (CBR, Content-Based Routing), com o objetivo de explorar as informações do conteúdo para rotear os eventos eficientemente para seus destinos.

6.3 Sistemas publicar-assinar

6.3.2 Problemas de implementação

Arquitetura de sistemas global
(para estratégias baseadas em conteúdo)

- Camada superior: correspondência
- implementa a correspon-dência: garante que os eventos correspondam a determinada assinatura.
- Embora isso possa ser implementado como uma camada separada, frequentemente a correspondência é rebaixada para os mecanismos de roteamento de evento.

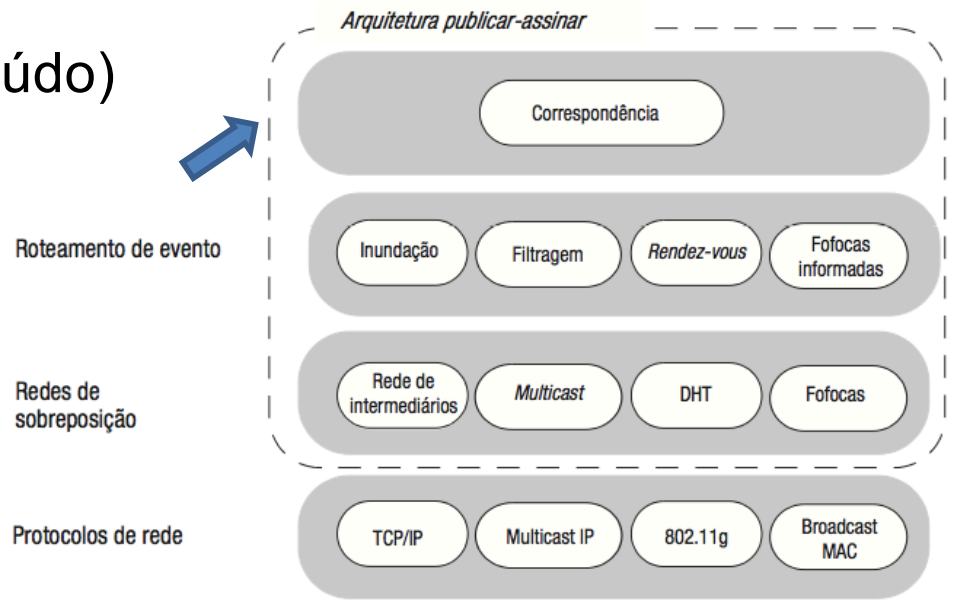


Figura 6.10 A arquitetura de sistemas publicar-assinar.

6.3 Sistemas publicar-assinar

6.3.2 Problemas de implementação

Arquitetura de sistemas global
(para estratégias baseadas em conteúdo)

Roteamento baseado em conteúdo – Implementações

- Inundação: estratégia mais simples: enviar uma notificação de evento para todos os nós da rede e, então, realizar a correspondência apropriada na extremidade assinante.
  - Como alternativa, a inundação pode ser usada para enviar assinaturas de volta para todos os publicadores possíveis, com a correspondência feita na extremidade publicadora e os eventos correspondentes enviados diretamente para os assinantes relevantes usando comunicação ponto a ponto.
  - A inundação pode ser implementada usando-se um recurso de broadcast ou multicast subjacente.
  - Pode resultar em muito tráfego de rede desnecessário

6.3 Sistemas publicar-assinar

6.3.2 Problemas de implementação

Arquitetura de sistemas global
(para estratégias baseadas em conteúdo)

Roteamento baseado em conteúdo – Implementações

- Filtragem: um princípio que serve de base para muitas estratégias é aplicar filtragem na rede de intermediários.
- Os intermediários mantém a informação de assinatura de assinantes de cada nó publicador e uma tabela de rotas para seus intermediários vizinhos.
- O intermediário só envia notificação para os publicadores em potencial.
- Pode gerar muito tráfego devido por enviar informação sobre assinaturas por inundação (broadcast)

6.3 Sistemas publicar-assinar

6.3.2 Problemas de implementação

Arquitetura de sistemas global
(para estratégias baseadas em conteúdo)

Roteamento baseado em conteúdo – Implementações

- Anúncios: Em sistemas em que há comunicação por anúnicos, reduz a carga na rede por enviar os anúncios para assinantes de maneira semelhante à propagação de assinaturas.
- Depende da existência deste tipo de operação (anúncios)

6.3 Sistemas publicar-assinar

6.3.2 Problemas de implementação

Arquitetura de sistemas global
(para estratégias baseadas em conteúdo)

Roteamento baseado em conteúdo – Implementações

- Rendez-vous: ver o conjunto de todos os eventos possíveis como um espaço de eventos e repartir a responsabildade por esse espaço de eventos entre o conjunto de intermediários na rede.
- Em particular, essa estratégia define nós de rendez-vous, que são nós intermediários responsáveis por determinado subconjunto do espaço de eventos.
- Obter um balenceamento de carga natural.

6.3 Sistemas publicar-assinar
6.3.3 Exemplos de sist. publicar-assinar

Sistemas (leituras recomendadas)	*Modelo de assinatura*	*Modelo de distribuição*	*Roteamento de eventos*
CORBA Event Service (Capítulo 8)	Baseado em canal	Centralizado	–
TIB Rendezvouz [Oki *et al.* 1993]	Baseado em tópicos	Distribuído	Filtragem
Scribe [Castro *et al.* 202b]	Baseado em tópicos	*Peer-to-peer* (DHT)	*Rendez-vouz*
TERA [Baldoni *et al.* 2007]	Baseado em tópicos	*Peer-to-peer*	Fofoca informada
Siena [Carzaniga *et al.* 2001]	Baseado em conteúdo	Distribuído	Filtragem
Gryphon [www.research.ibm.com]	Baseado em conteúdo	Distribuído	Filtragem
Hermes [Pietzuch e Bacon 2002]	Baseado em tópico e em conteúdo	Distribuído	*Rendez-vouz* e filtragem
MEDYM [Cao e Singh 2005]	Baseado em conteúdo	Distribuído	Inundação
Meghdoot [Gupta *et al.* 2004]	Baseado em conteúdo	*Peer-to-peer*	*Rendez-vouz*
Structure-less CBR [Baldoni *et al.* 2005]	Baseado em conteúdo	*Peer-to-peer*	Fofoca informada

6.4 Filas de Mensagens (distribuídas)

- Sistema ponto a ponto (ao contrário dos anteriores que são um para muitos): o remetente coloca a mensagem em uma fila que é, então, é removida por um único processo.
- Fila de mensagens como indireção: obtendo as propriedades do desacoplamento temporal e espacial.
- As filas de mensagem também são referidas como middleware orientado a mensagens.
- O principal uso desses produtos é na obtenção de integração de aplicativo empresarial. Eles também são amplamente usados como a base de sistemas de processamento de transações comerciais, em razão de seu suporte intrínsico para transações.

6.4 Filas de Mensagens (distribuídas)
6.4.1. Modelo de programação

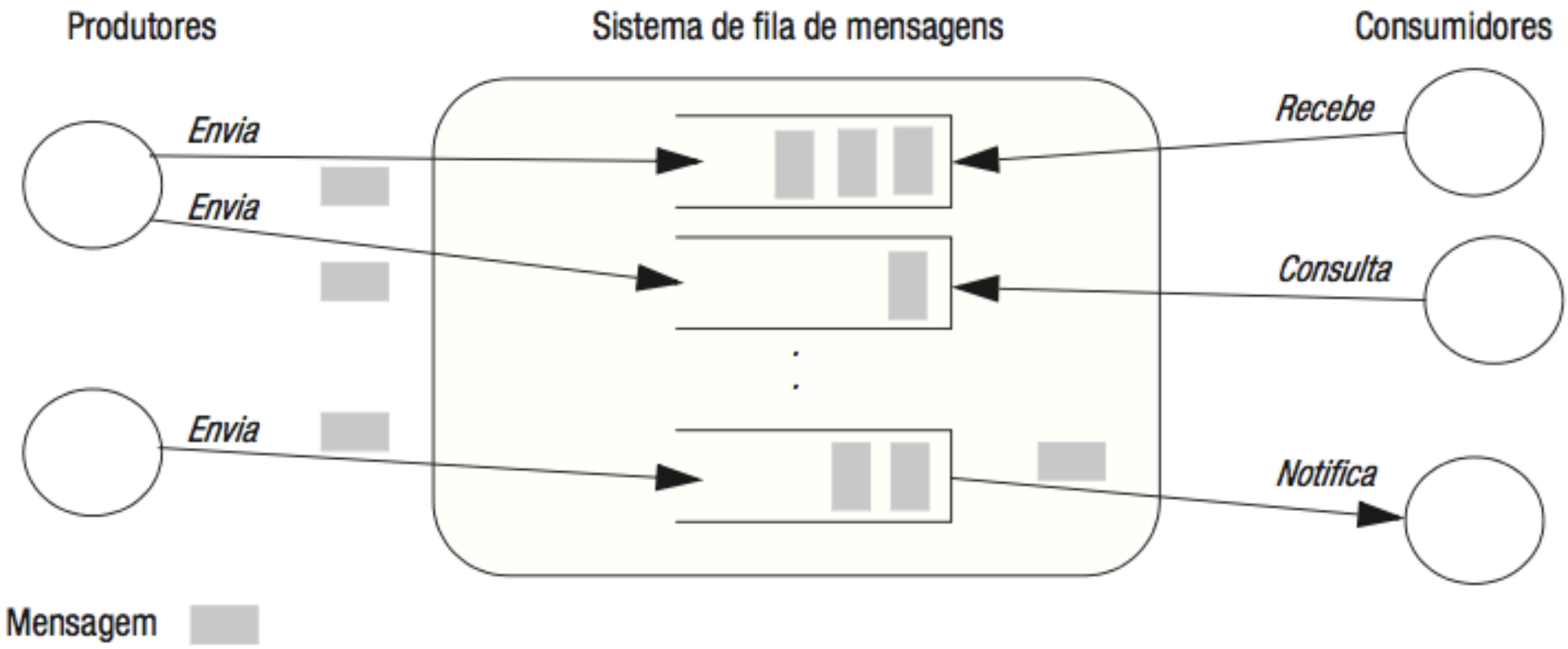
- Modelo simples: estratégia para comunicação em sistemas distribuídos por meio de filas.
- Processos produtores enviam mensagens para uma fila específica e processos consumidores)podem receber mensagens dessa fila.
- Geralmente, são suportados três estilos de recepção:
  - uma recepção com bloqueio, que bloqueará até que uma mensagem apropriada esteja disponível;
  - uma recepção sem bloqueio (operação de consulta), que verifica o status da fila (disponível / indisponível);
  - uma operação de notificação de evento quando uma mensagem estiver disponível na fila associada.

6.4 Filas de Mensagens (distribuídas)

6.4.1. Modelo de programação

- Vários processos podem enviar mensagens para a mesma fila e, do mesmo modo, vários destinatários podem remover mensagens de uma fila.
- Enfileiramento normalmente FIFO (first-in-first-out – primeiro a entrar, primeiro a sair), mas pode haver prioridade, com as mensagens de prioridade mais alta entregues primeiro.
- Os processos consumidores podem selecionar mensagens da fila com base em propriedades (uma mensagem consiste em um destino, metadados e o corpo da mensagem (normalmente opaco e não é analisado pelo sistema de fila de mensagens. O conteúdo associado é serializado).

6.4 Filas de Mensagens (distribuídas)
6.4.1. Modelo de programação



O paradigma da fila de mensagens

6.4 Filas de Mensagens (distribuídas)
6.4.1. Modelo de programação

- Propriedade fundamental: as mensagens são persistentes – isto é, as filas de mensagem armazenam as mensagens indefinidamente (até serem consumidas) e também as armazenam no disco para permitir a entrega confiável.
- Comunicação confiável: toda mensagem enviada é recebida (validade) e a mensagem recebida é idêntica à enviada, sendo que nenhuma mensagem é entregue duas vezes (integridade).
- Os sistemas de fila de mensagens garantem que as mensagens sejam entregues (e apenas uma vez), mas não podem dar informações sobre o momento da entrega.

6.4 Filas de Mensagens (distribuídas)

6.4.1. Modelo de programação

Filas de mensagens X Passagem de mensagem

De muitas maneiras são semelhantes. A diferença é que, enquanto os sistemas de passagem de mensagem têm filas **implícitas** associadas a remetentes e destinatários (buffers de mensagem), os sistemas de enfileiramento de mensagem têm filas **explícitas** que são entidades terceirizadas, separadas do remetente e do destinatário.

É essa importante diferença que torna as filas de mensagem um paradigma de comunicação indireta, com as propriedades fundamentais de desacoplamento espacial e temporal.

6.4 Filas de Mensagens (distribuídas)
6.4.2. Modelo de programação
Arquitetura Centralizada X Distribuída

- implementações centralizadas: com uma ou mais filas de mensagem controladas por um único gerenciador de fila localizado em determinado nó.
  - A vantagem desse esquema é a simplicidade, mas tais gerenciadores podem se tornar componentes bastante pesados e têm o potencial de se tornar um gargalo ou um único ponto de falha.

- Solução: implementações distribuídas.

6.4 Filas de Mensagens (distribuídas)

6.4.2. Modelo de programação

Estudo de Caso: JMS (Java Messaging Service)

- especificação padronizada para programas Java distribuídos, para comunicação indireta.
- unifica os paradigmas de sistemas publicar-assinar e fila de mensagens pelo menos superficialmente, suportando tópicos e filas como destinos de mensagens alternativos
- Uma ampla variedade de implementações da especificação comum está disponível, incluindo Joram da OW2, Java Messaging da JBoss, Open MQ da Oracle, Apache ActiveMQ e OpenJMS (Outras plataformas, incluindo Websphere MQ, também fornecem uma interface JMS em sua infraestrutura subjacente).

6.4 Filas de Mensagens (distribuídas)

6.4.2. Modelo de programação

Estudo de Caso: JMS (Java Messaging Service)

Funções do JMS:

- Cliente: é um programa ou componente Java que produz ou consome mensagens (produtor ou consumidor);
- Provedor JMS: é qualquer um dos vários sistemas que implementam a especificação JMS.
- Mensagem: é um objeto usado para comunicar informações entre clientes JMS (dos produtores para os consumidores).
- Destino: é um objeto que suporta a comunicação indireta no JMS. Tópico ou fila JMS.

6.4 Filas de Mensagens (distribuídas)

6.4.2. Modelo de programação

Estudo de Caso: JMS (Java Messaging Service)

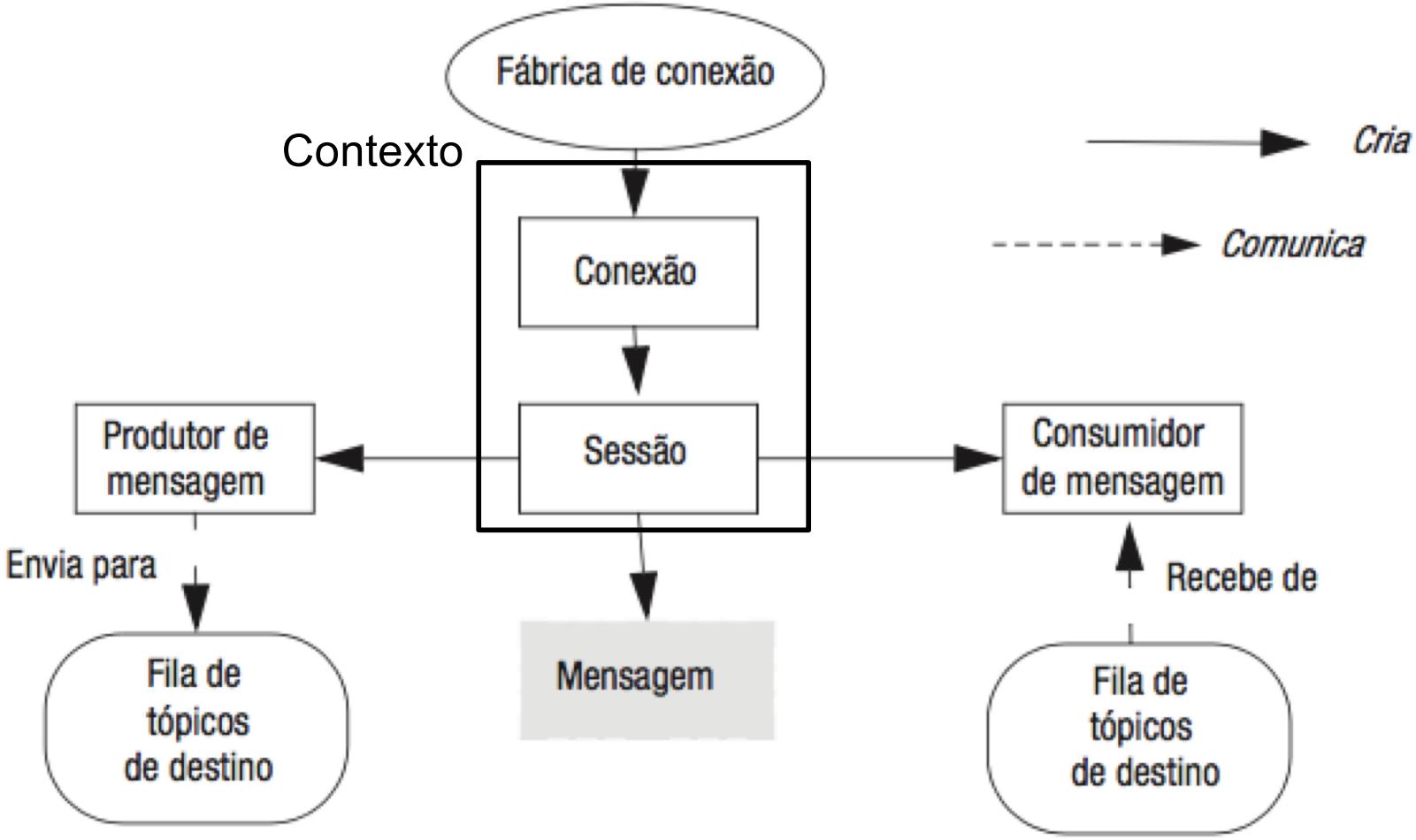
Programação com JMS

- Fábrica de conexão: um serviço responsável por estabelecer conexões cliente e o provedor com as propriedades exigidas (conexão resultante é um canal lógico entre o cliente e o provedor)
- Modo de operação: dois tipos de conexão:
    - TopicConnection (publish-subscribe)
    - QueueConnection (Filas de mensagens)
- Sessões: série de operações envolvendo criação, produção e consumo de mensagens relacionadas a uma tarefa lógica.

6.4 Filas de Mensagens (distribuídas)

6.4.2. Modelo de programação

Estudo de Caso: JMS (Java Messaging Service)



O modelo de programação oferecido pelo JMS

6.4 Filas de Mensagens (distribuídas)

6.4.2. Modelo de programação

Estudo de Caso: JMS (Java Messaging Service)

Exemplo – Alarme de Incêndio JMS Publicar-Assinar

- Aplicativo um para muitos: o alarme (produtor) produz mensagem de alerta para muitos consumidores
- O código para criar uma nova instância da classe FireAlarmJMS e disparar um alarme é (Publicador):

```
FireAlarmJMS alarm = new FireAlarmJMS()
alarm.raise();
```

- O código para criar uma nova instância de assinante é:

```
FireAlarmConsumerJMS alarmCall = new FireAlarmConsumerJMS();
String msg = alarmCall.await();
System.out.println("Alarm received: "+msg);
```

6.4 Filas de Mensagens (distribuídas)

6.4.2. Modelo de programação

Estudo de Caso: JMS (Java Messaging Service)

Publicador

```
import javax.jms.*;
import javax.naming.*;

public class FireAlarmJMS {

    public void raise( ) {
    try {                                                              1
        Context ctx = new InitialContext( );                           2
        TopicConnectionFactory topicFactory =                          3
            (TopicConnectionFactory)ctx.lookup("TopicConnectionFactory");  4
        Topic topic = (Topic)ctx.lookup("Alarms");                     5
        TopicConnection topicConn =                                    6
            topicFactory.createTopicConnection( );                     7
        TopicSession topicSess = topicConn.createTopicSession(false,   8
                                        Session.AUTO_ACKNOWLEDGE);     9
        TopicPublisher topicPub = topicSess.createPublisher(topic);    10
        TextMessage msg = topicSess.createTextMessage( );              11
        msg.setText("Fire!");                                          12
        topicPub.publish(msg);                                         13
        } catch (Exception e) {                                        14
    }                                                                  15
}
```

6.4 Filas de Mensagens (distribuídas)

6.4.2. Modelo de programação

Estudo de Caso: JMS (Java Messaging Service)

Assinante

```
import javax.jms.*;
import javax.naming.*;

public class FireAlarmConsumerJMS {
    public String await( ) {
    try {                                                              1
        Context ctx = new InitialContext( );                           2
        TopicConnectionFactory topicFactory =                          3
            (TopicConnectionFactory)ctx.lookup("TopicConnectionFactory"); 4
        Topic topic = (Topic)ctx.lookup("Alarms");                     5
        TopicConnection topicConn =                                    6
            topicFactory.createTopicConnection( );                     7
        TopicSession topicSess = topicConn.createTopicSession(false,   8
                                        Session.AUTO_ACKNOWLEDGE);     9
        TopicSubscriber topicSub = topicSess.createSubscriber(topic);  10
        topicSub.start( );                                             11
        TextMessage msg = (TextMessage) topicSub.receive( );           12
        return msg.getText( );                                         13
        } catch (Exception e) {                                        14
            return null;                                               15
    }                                                                  16
}
```

6.4 Filas de Mensagens (distribuídas)
6.4.2. JMS – Fila de Msg - HelloWorld

```
import javax.jms.*;
public class HelloMsg {
   public static void main(String argv[]) throws Exception {
      // The producer and consumer need to get a connection factory and use it to set up
      // a connection and a session
      QueueConnectionFactory connFactory = new com.sun.messaging.QueueConnectionFactory();
      QueueConnection conn = connFactory.createQueueConnection();
      // This session is not transacted, and it uses automatic message acknowledgement
      QueueSession session = conn.createQueueSession(false, Session.AUTO_ACKNOWLEDGE);
      Queue q = new com.sun.messaging.Queue("world");
      // Sender
      QueueSender sender = session.createSender(q);
      // Text message
      TextMessage msg = session.createTextMessage();
      msg.setText("Hello there!");
      System.out.println("Sending the message: "+msg.getText());
      sender.send(msg);
      // Receiver
      QueueReceiver receiver = session.createReceiver(q);
      conn.start();
      Message m = receiver.receive();
      if(m instanceof TextMessage) {
         TextMessage txt = (TextMessage) m;
         System.out.println("Message Received: "+txt.getText());
      }
      session.close();
      conn.close();
   }
}
```

```
Sending the message: Hello there!
Message Received: Hello there!
```

6.5 Estratégias de Memória Compartilhada

- Memória Compartilhada Distribuída – DSM - Distributed Shared Memory

- Comunicação via espaço de tuplas
    - Estudo de caso: JavaSpaces

6.5 Estratégias de Memória Compartilhada
6.5.1 Memória Compartilhada Distribuída

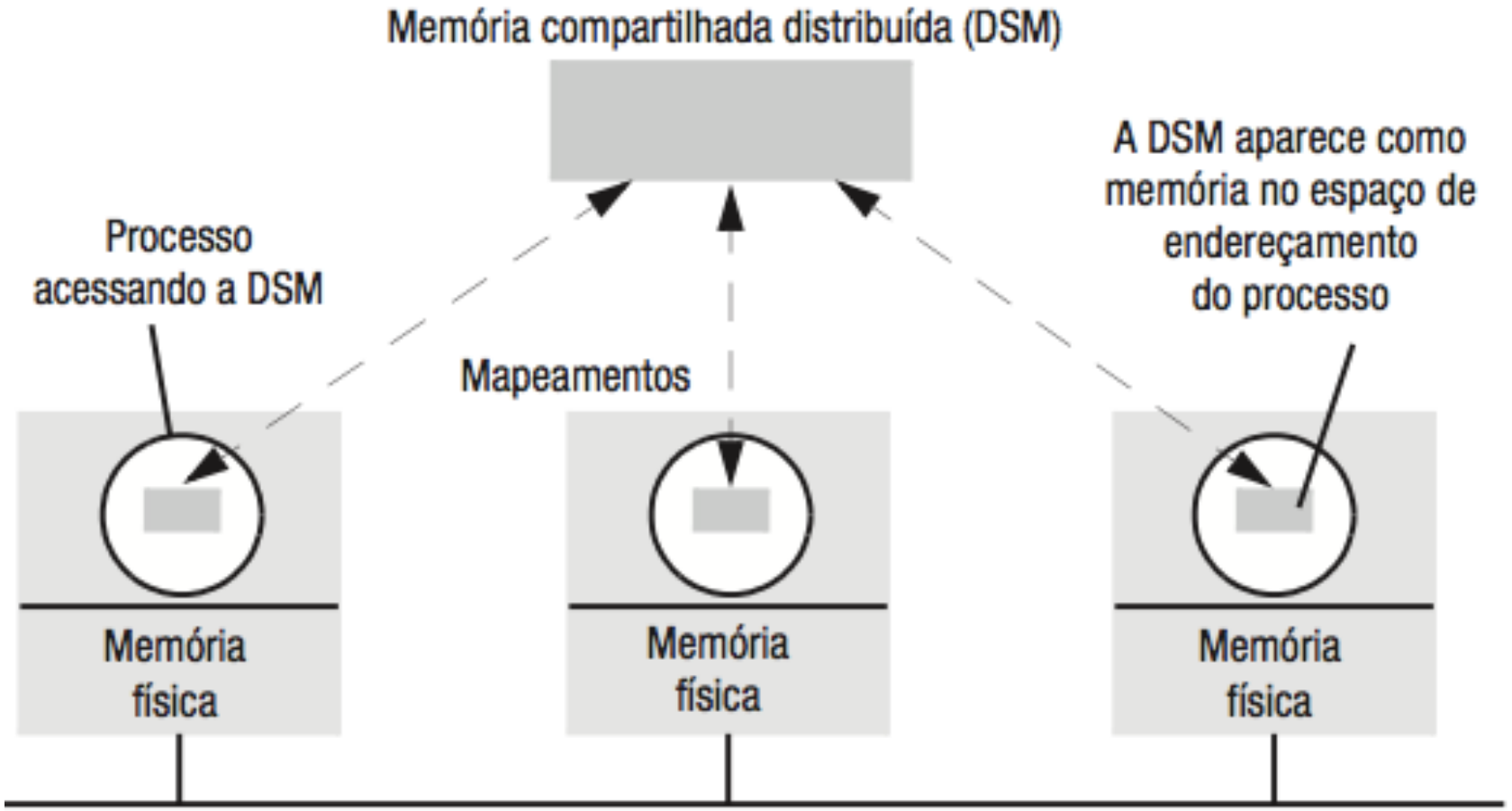
DSM - Distributed Shared Memory

- Abstração para compartilhar dados entre computadores que não compartilham memória física (RAM).
- Os processos acessam a DSM por meio de leituras e atualizações no que parece ser memória normal dentro de seus espaços de endereçamento. Contudo, um suporte de execução runtime subjacente garante, de forma transparente, que os processos sendo executados em diferentes computadores observem as atualizações feitas pelos outros.
- É como se os processos acessassem uma única memória compartilhada, mas na verdade a memória física é distribuída.

6.5 Estratégias de Memória Compartilhada

6.5.1 Memória Compartilhada Distribuída

DSM - Distributed Shared Memory

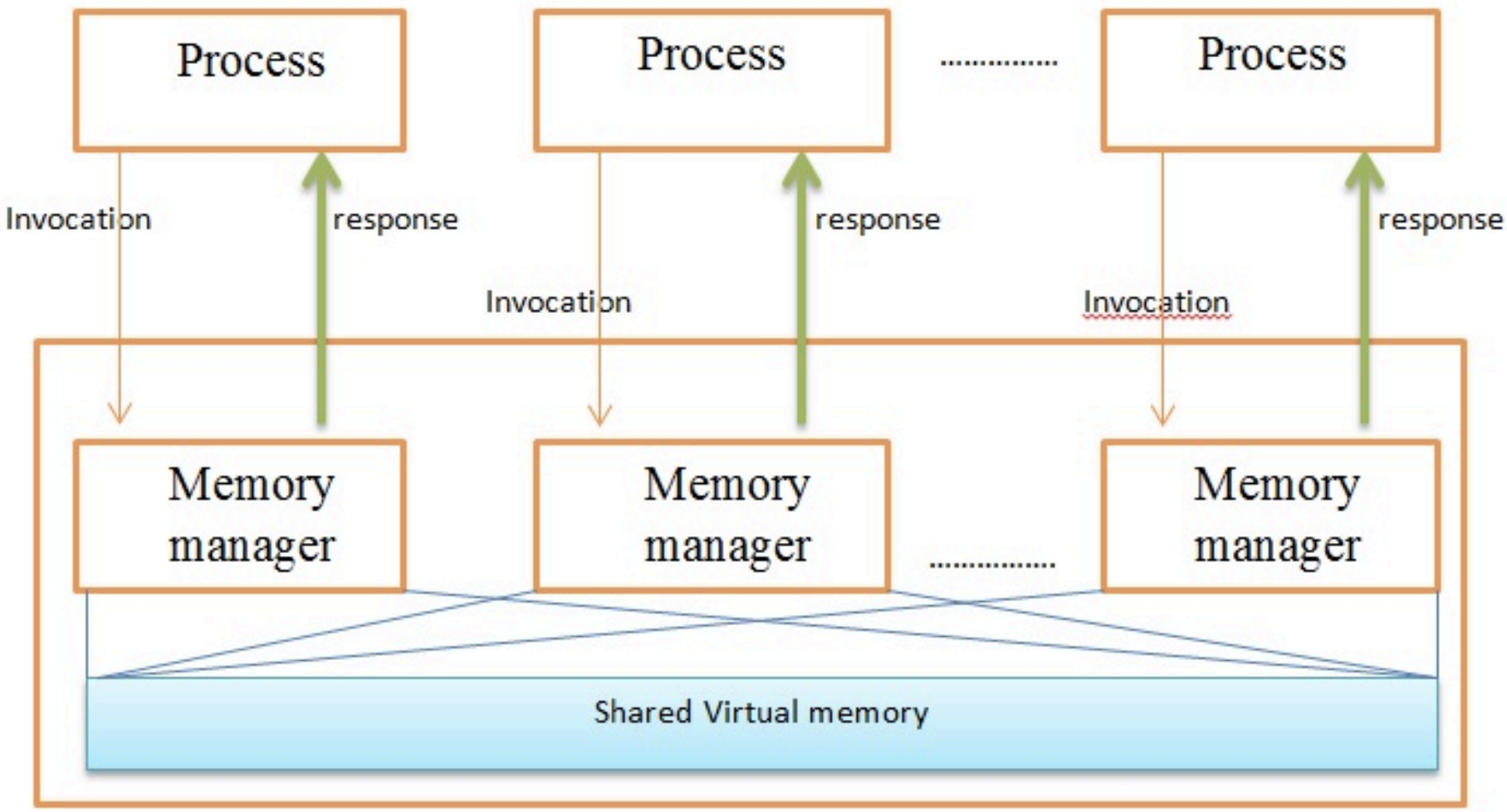


A abstração de memória compartilhada distribuída (DSM).

6.5 Estratégias de Memória Compartilhada

6.5.1 Memória Compartilhada Distribuída

DSM - Distributed Shared Memory



Distributed shared memory

6.5 Estratégias de Memória Compartilhada

6.5.1 Memória Compartilhada Distribuída

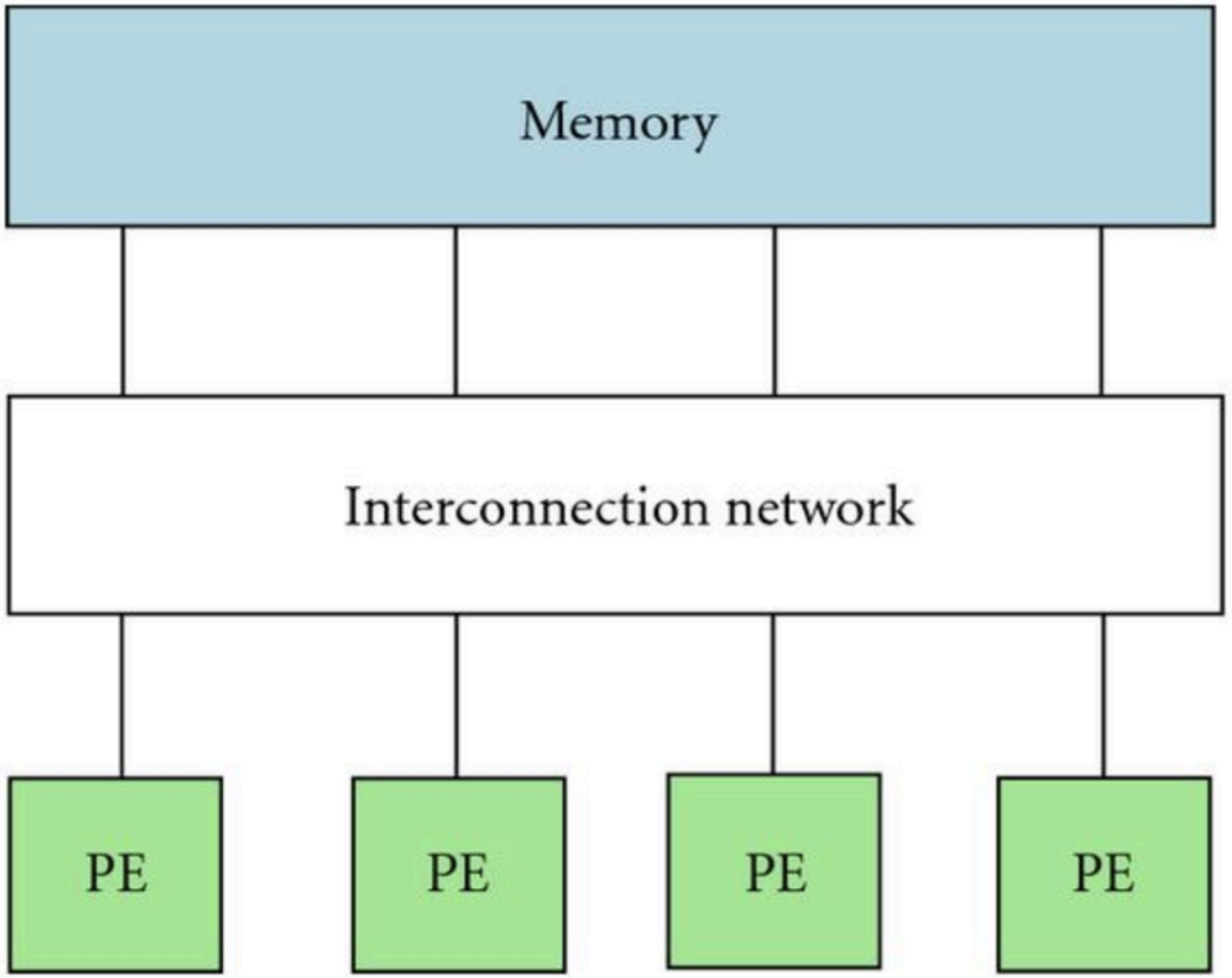
DSM - Distributed Shared Memory

- A principal vantagem da DSM é que ela dispensa o programador das preocupações com passagem de mensagens ao escrever aplicativos que, de outra forma, talvez tivessem que utilizá-la.
- A DSM é uma ferramenta para aplicativos paralelos ou distribuídos nos quais itens de dados compartilhados individuais podem ser acessados diretamente.
- Em geral, a DSM é menos adequada em sistemas cliente-servidor, em que os clientes normalmente veem os recursos mantidos no servidor como dados abstratos e os acessam por requisição (por motivos de modularidade e proteção).

6.5 Estratégias de Memória Compartilhada

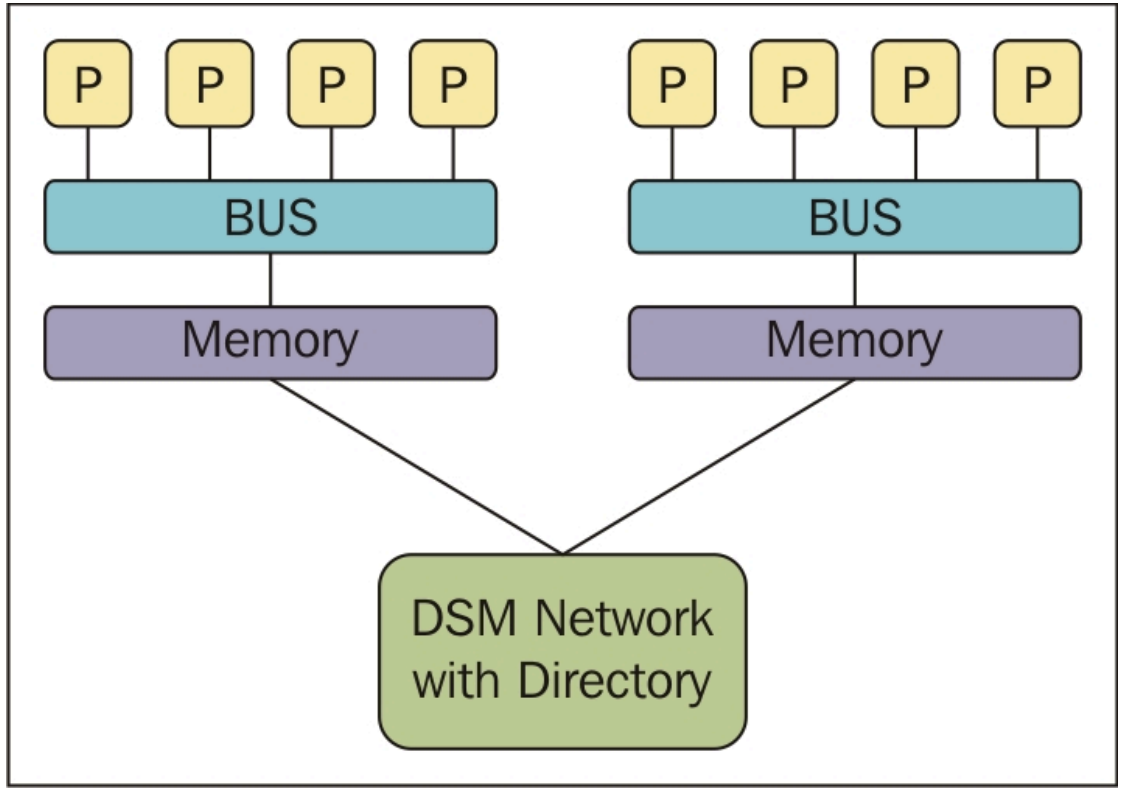
6.5.1 Memória Compartilhada Distribuída

Shared Memory Multiprocessor



6.5 Estratégias de Memória Compartilhada
6.5.1 Memória Compartilhada Distribuída

NUMA – Non-Uniform Memory Access



- arquitetura hierárquica na qual placas de quatro processadores são conectadas com um *switch* de alto desempenho ou com um barramento de nível mais alto.
- Os processadores veem um único espaço de endereçamento contendo toda a memória de todas as placas.
- No entanto, a latência de acesso da memória da placa é menor do que a de um módulo de memória em outra placa – daí o nome dessa arquitetura.

6.5 Estratégias de Memória Compartilhada
6.5.1 Memória Compartilhada Distribuída

Passagem de Mensagem *x* DSM

- Como mecanismo de comunicação, o DSM se compara com a passagem de mensagens (PM), pois sua aplicação em processamento paralelo, em particular, requer o uso de comunicação assíncrona.
- Em PM, as variáveis precisam ser empacotadas, transmitidas e desempacotadas em outras variáveis no processo de destino. Em contraste, DSM os processos envolvidos compartilham as variáveis diretamente, de modo que não é necessário nenhum empacotamento.
- como a DSM pode ser persistente, os processos que se comunicam podem executar com tempos não sobrepostos. Em contraste, os processos que se comunicam por meio de PM devem ser executados ao mesmo tempo.

6.5 Estratégias de Memória Compartilhada
6.5.2 Comunicação via espaço de tuplas

- Forma de computação distribuída baseada em *comunicação generativa* [Gelernter 1985].
- Os processos se comunicam indiretamente, colocando tuplas em um espaço de tuplas, enquanto outros processos podem ler ou remover tuplas desse espaço.
- As tuplas não têm endereço, mas são acessadas por casamento de padrões no conteúdo (memória endereçada pelo conteúdo).
- Sistemas:
    - Linda [Gelernter 1985]
    - Agora [Bisiani e Forin 1988]
    - JavaSpaces, da Oracle
    - TSpace, da IBM.

6.5 Estratégias de Memória Compartilhada

6.5.2 Comunicação via espaço de tuplas

Modelo de Programação

- Tuplas são uma sequência de um ou mais campos de dados tipados: <“fred”, 1958>, <“sid”, 1964>, <4, 9.8, “Yes”>.
- Processos se comunicam por meio de um espaço de tuplas: uma coleção de tuplas compartilhada.
- Qualquer combinação de tipos de tuplas pode existir no mesmo espaço de tuplas.
- Operações:
    - Write (out): adiciona uma tupla sem afetar as tuplas existentes no espaço.
    - read (rd): retorna o valor de uma tupla sem afetar o conteúdo do espaço de tuplas.
    - Take (in): retorna uma tupla, removendo-a do espaço de tuplas.

6.5 Estratégias de Memória Compartilhada

6.5.2 Comunicação via espaço de tuplas

Modelo de Programação

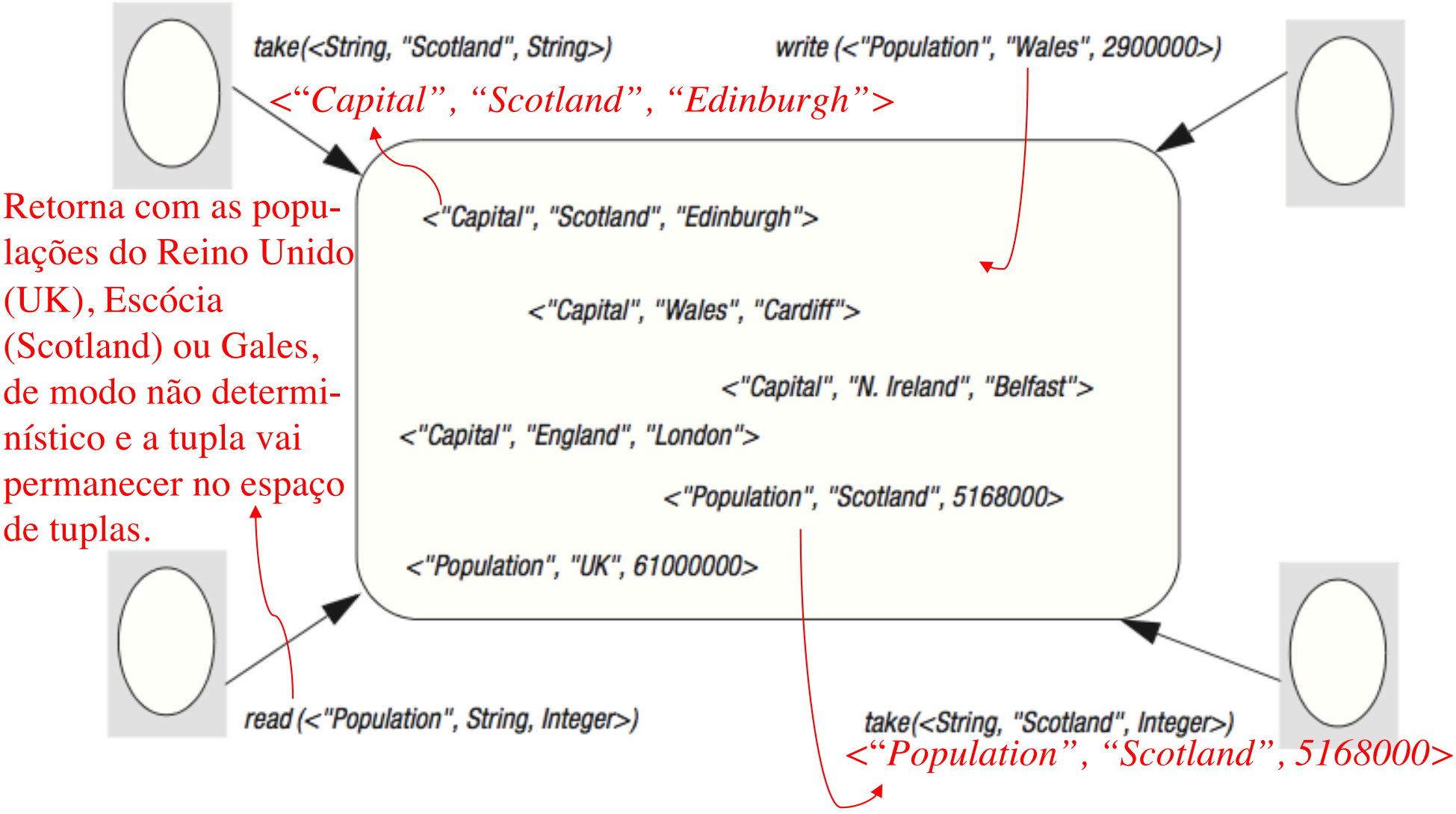
- Endereçamento associativo: um processo fornece uma especificação de tupla, e o espaço de tuplas retorna qualquer tupla que corresponda a essa especificação. Exemplos:
  - take(<String, integer>) poderia extrair <“fred”, 1958> ou <“sid”, 1964>;
  - take(<String, 1958>) extrairia somente <“fred”, 1958> dessas duas.
- As tuplas são imutáveis: nenhum acesso direto às tuplas é permitido, e os processos têm de substituir as tuplas no espaço de tuplas, em vez de modificá-las. Exemplo:
  - incrementar o contador em um espaço de tuplas myTS:

    <s, count>:= myTS.take(<“counter”, integer>);
    myTS.write(<“counter”, count+1>);

6.5 Estratégias de Memória Compartilhada

6.5.2 Comunicação via espaço de tuplas

Modelo de Programação



6.5 Estratégias de Memória Compartilhada

6.5.2 Comunicação via espaço de tuplas

Modelo de Programação

Propriedades associadas ao espaço de tuplas

- Desacoplamento espacial: uma tupla pode se originar de qualquer número de processos remetentes e pode ser entregue a qualquer um de vários destinatários em potencial.
- Desacoplamento temporal: uma tupla colocada no espaço de tuplas vai permanecer nesse espaço de tuplas até que seja removida (talvez indefinidamente) e, assim, o remetente e o destinatário não precisam se sobrepor no tempo.
- Juntos, esses pontos fornecem uma estratégia totalmente distribuída no espaço e no tempo e uma forma de compartilhamento distribuído de variáveis distribuídas por meio do espaço de tuplas.

6.5 Estratégias de Memória Compartilhada

6.5.2 Comunicação via espaço de tuplas

Problemas de implementação

- Solução centralizada: espaço de tuplas é gerenciado por um único servidor.
    - Vantagem: simplicidade
    - Desvantagens: não é tolerante a falhas e não muda de escala.
- Soluções distribuídas:
    - Replicação
    - Particionamento do espaço de tuplas
    - Peer-to-peer

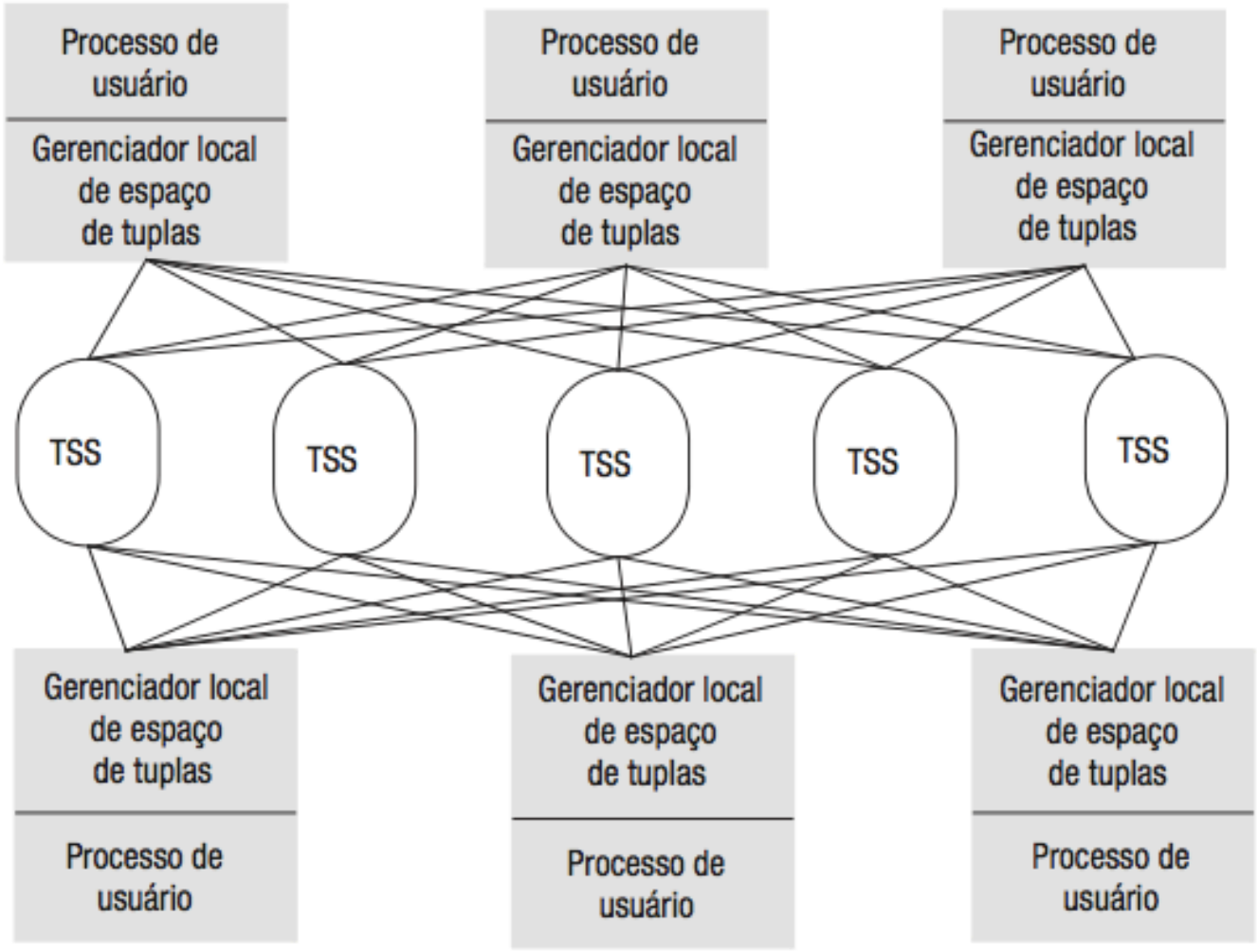
6.5 Estratégias de Memória Compartilhada

6.5.2 Comunicação via espaço de tuplas

Problemas de implementação – Soluções Distribuídas

Particionamento do espaço de tuplas

TSS: Tuple Space Server

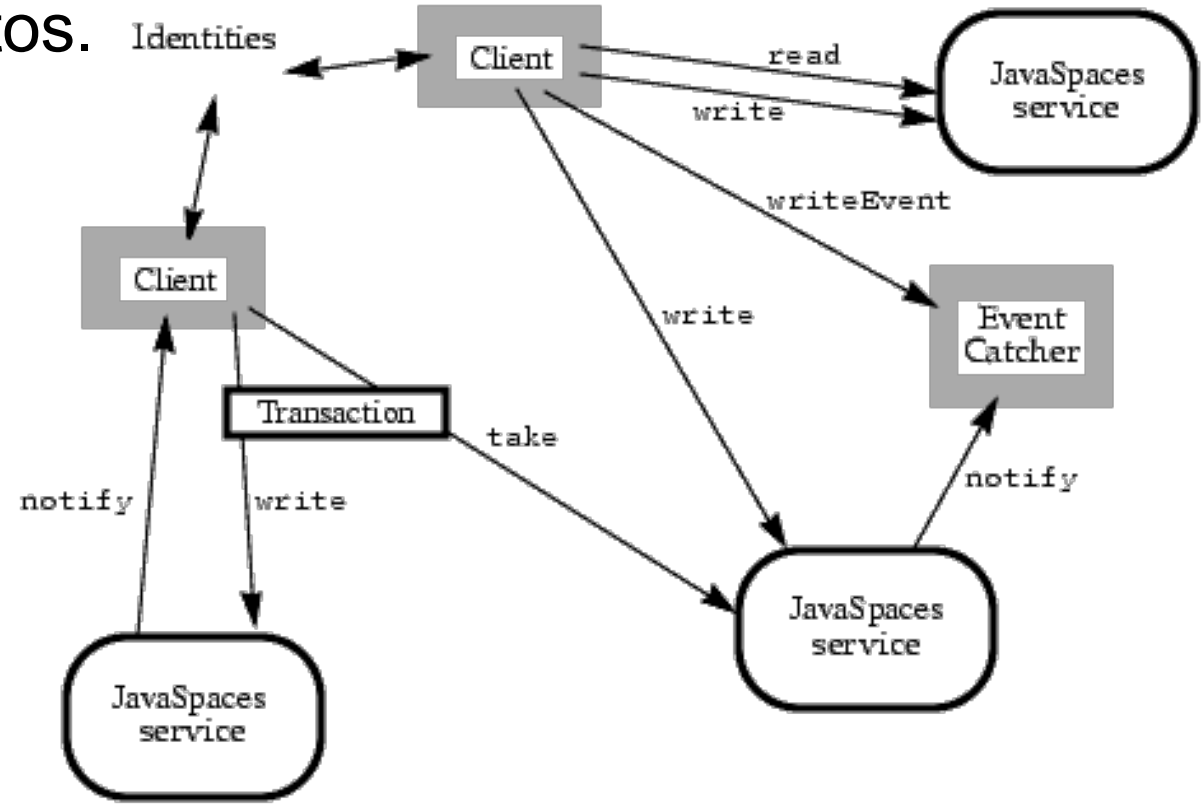


6.5 Estratégias de Memória Compartilhada

6.5.2 Comunicação via espaço de tuplas

Estudo de Caso: JavaSpaces

- O JavaSpaces é uma ferramenta para comunicação via espaço de tuplas desenvolvido pela (Sun) Oracle.
- Permite ao programador criar qualquer numero de instâncias de um espaço, onde espaço é um repositório persistente e compartilhado de objetos.



6.5 Estratégias de Memória Compartilhada

6.5.2 Comunicação via espaço de tuplas

Estudo de Caso: JavaSpaces

Operação	*Efeito*
Lease write(Entry e, Transaction txn, long lease)	Coloca uma entrada em um JavaSpace específico
Entry read(Entry tmpl, Transaction txn, long timeout)	Retorna uma cópia de uma entrada correspondente a um modelo especificado
Entry readIfExists(Entry tmpl, Transaction txn, long timeout)	Como o anterior, mas sem bloqueio
Entry take(Entry tmpl, Transaction txn, long timeout)	Recupera (e remove) uma entrada correspondente a um modelo especificado
Entry takeIfExists(Entry tmpl, Transaction txn, long timeout)	Como o anterior, mas sem bloqueio
EventRegistration notify(Entry tmpl, Transaction txn, RemoteEventListener listen, long lease, MarshalledObject handback)	Notifica um processo, caso uma tupla correspondente a um modelo especificado seja gravada em um JavaSpace

Figura 6.23 A API JavaSpaces.

6.5 Estratégias de Memória Compartilhada

6.5.2 Comunicação via espaço de tuplas

Estudo de Caso: JavaSpaces

Criação da Tupla – O objeto de entrada AlarmTupleJS

```
import net.jini.core.entry.*;

public class AlarmTupleJS implements Entry {
        public String alarmType;

        public AlarmTupleJS( ) {
        }

        public AlarmTupleJS(String alarmType) {
                this.alarmType = alarmType;
        }
}
```

Campo da tupla

Classe Java *AlarmTupleJS*.

6.5 Estratégias de Memória Compartilhada

6.5.2 Comunicação via espaço de tuplas

Estudo de Caso: JavaSpaces

Código de Alarme de Incêndio - Produtor de alarme

```
import net.jini.space.JavaSpace;

public class FireAlarmJS {

    public void raise( ) {
        try {
            JavaSpace space = SpaceAccessor.findSpace("AlarmSpace");
            AlarmTupleJS tuple = new AlarmTupleJS("Fire!");
            space.write(tuple, null, 60*60*1000);
        catch (Exception e) {
        }
    }
}
```

Classe Java *FireAlarmJS*.

Esse código pode, então, ser chamado usando-se o seguinte:

```
FireAlarmJS alarm = new FireAlarmJS();
alarm.raise();
```

6.5 Estratégias de Memória Compartilhada

6.5.2 Comunicação via espaço de tuplas

Estudo de Caso: JavaSpaces

Código de Alarme de Incêndio - Consumidor de alarme

```
import net.jini.space.JavaSpace;

public class FireAlarmConsumerJS {

    public String await( ) {
        try {
            JavaSpace space = SpaceAccessor.findSpace( );
            AlarmTupleJS template = new AlarmTupleJS("Fire!");
            AlarmTupleJS recvd = (AlarmTupleJS) space.read(template, null,
                                                Long.MAX_VALUE);
            return recvd.alarmType;
        }
        catch (Exception e) {
            return null;
        }
    }
}
```

Esse código pode, então, ser chamado usando-se:

```
FireAlarmConsumerJS alarmCall = new FireAlarmConsumerJS();
String msg = alarmCall.await();
System.out.println("Alarm received: "+msg);
```

Classe Java *FireAlarmReceiverJS*.