Sistemas Distribuídos baseados em Coordenação

Pedro Ferreira

DI - FCUL

Ciências ULisboa | Informática

Introdução

- Uma visão diferente sobre os sistemas distribuídos:
  - Consideramos sistemas inerentemente distribuídos que se modificam com o passar do tempo (entram e saem processos)
  - O grande problema a ser resolvido é como coordenar as interações entre os processos deste sistema

- O paradigma de coordenação promove a separação entre atividades de computação e coordenação:
  - **Computação:** execuções internas num processo
  - **Coordenação:** interações entre processos

Taxonomia dos Modelos de Coordenação

- Podem ser classificados de acordo com o seu grau de acoplamento (ou desacoplamento)

- O acoplamento pode ser:
  - **Referencial:** relativo à localização (por endereçamento) dos processos
    - » **Acoplado:** entidades conhecem o endereço umas das outras;
    - » **Desacoplado:** entidades desconhecem o endereço umas das outras.
  - **Temporal:** relativo ao estado das entidades comunicantes;
    - » **Acoplado:** entidades estão ativas ao mesmo tempo durante a interação
    - » **Desacoplado:** não necessitam estar ativas ao mesmo tempo para que a interação ocorra

Taxonomia dos Modelos de Coordenação

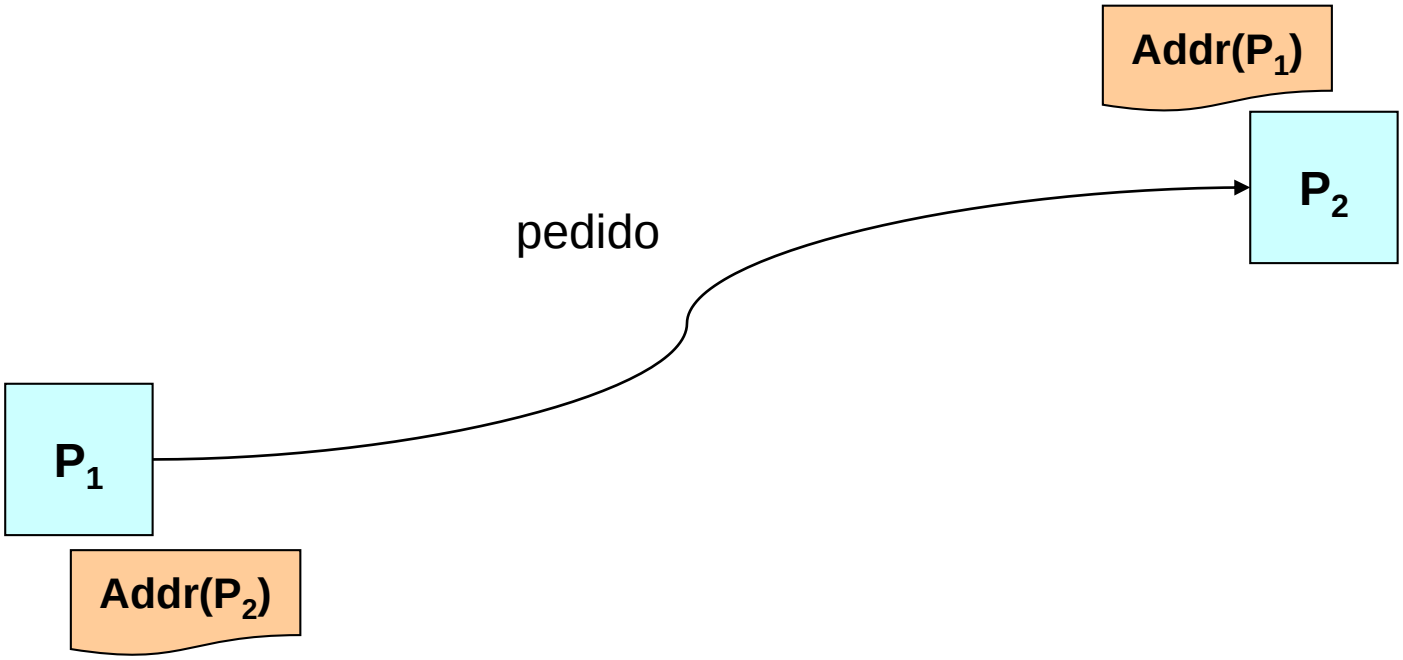
Referential \ Temporal	Coupled	Decoupled
Coupled	Direct	Mailbox
Decoupled	Meeting oriented	Generative communication

Taxonomia dos Modelos de Coordenação

- **Directa:** *sockets*, RPC

Taxonomia dos Modelos de Coordenação

- **Directa:** *sockets*, RPC

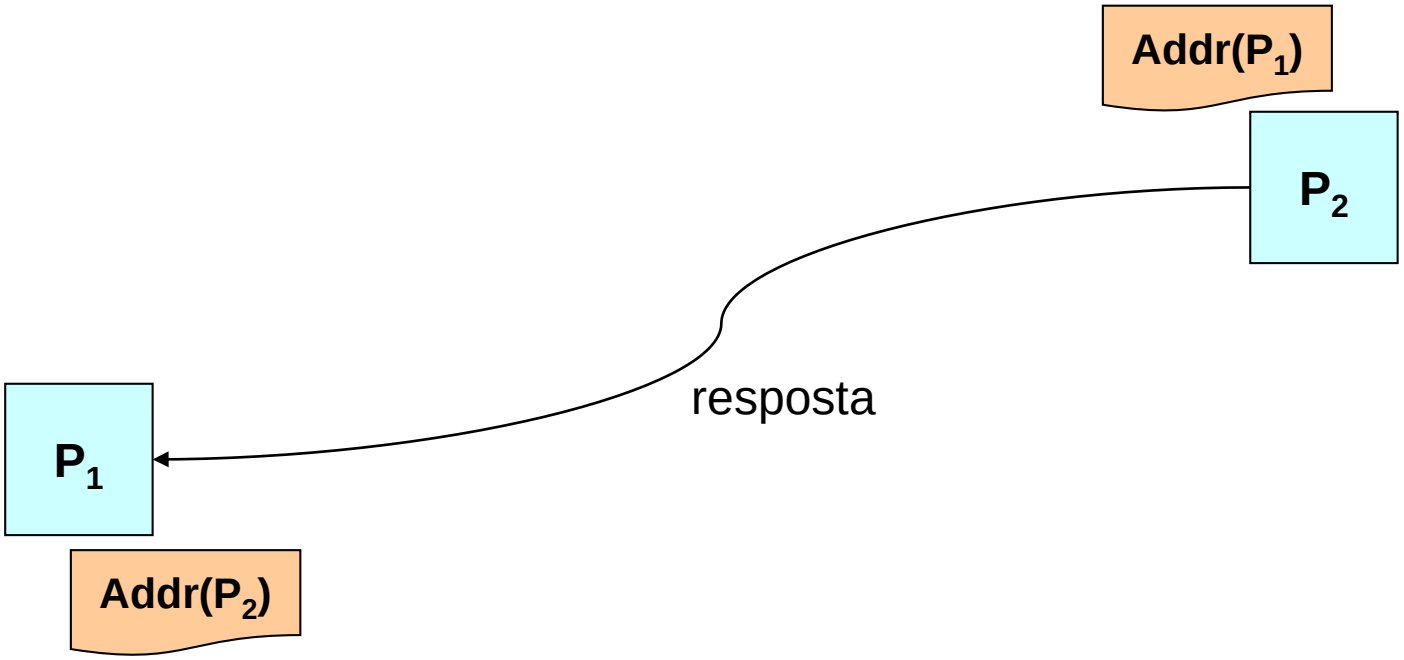


Taxonomia dos Modelos de Coordenação

- **Directa:** *sockets*, RPC

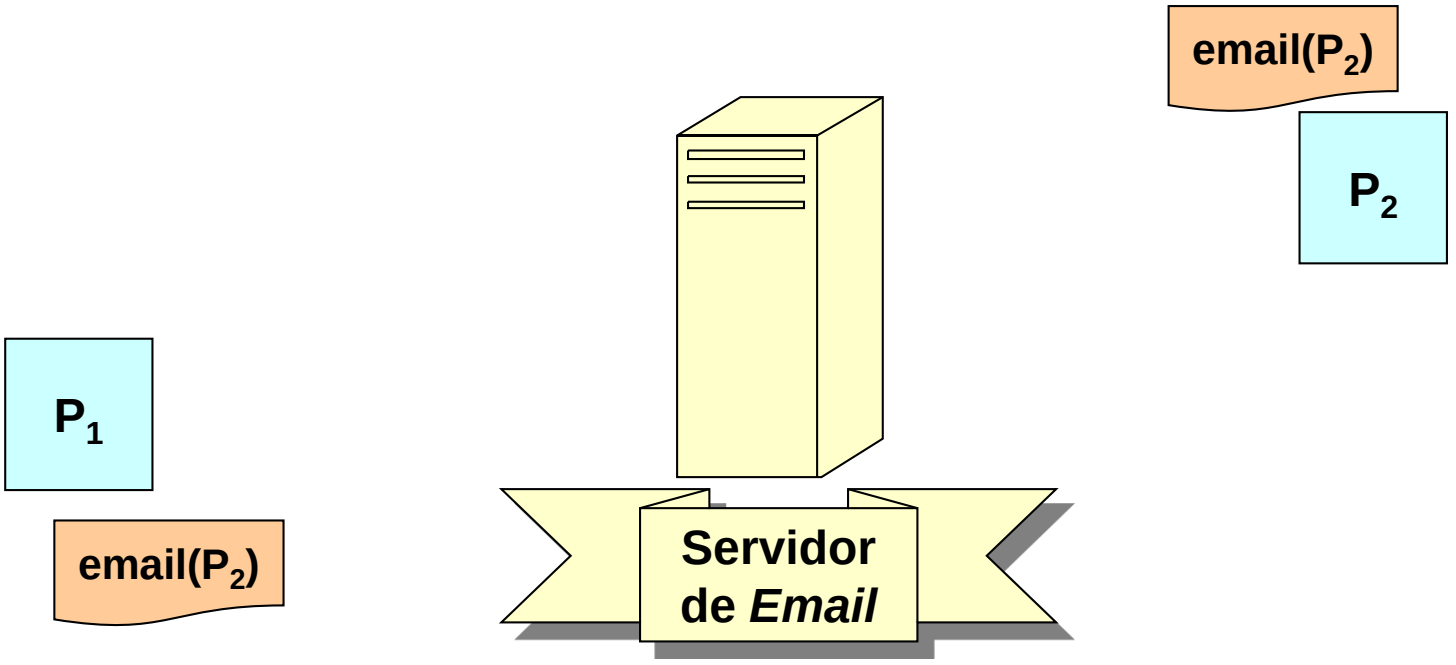
Taxonomia dos Modelos de Coordenação

- **Directa:** *sockets*, RPC



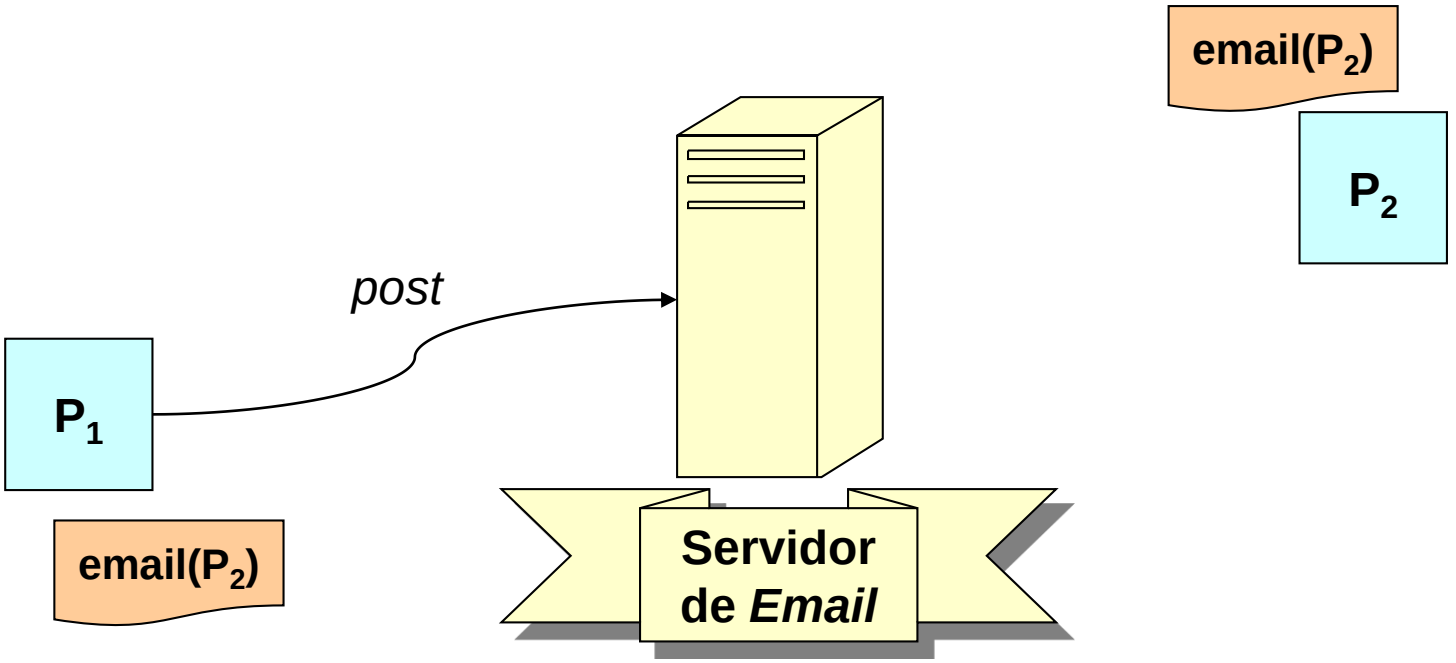
Taxonomia dos Modelos de Coordenação

- **Caixa de Correio:** *email*, quadro negro, filas de mensagens



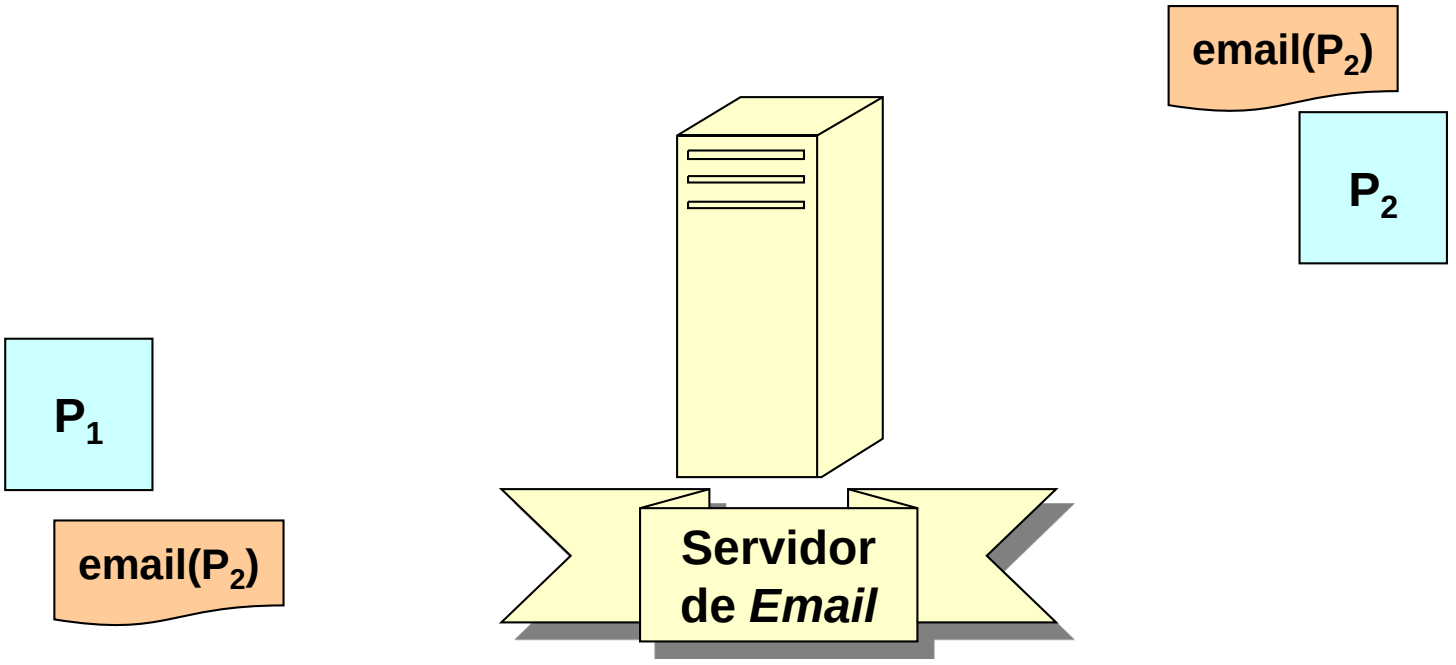
Taxonomia dos Modelos de Coordenação

- **Caixa de Correio:** *email*, quadro negro, filas de mensagens



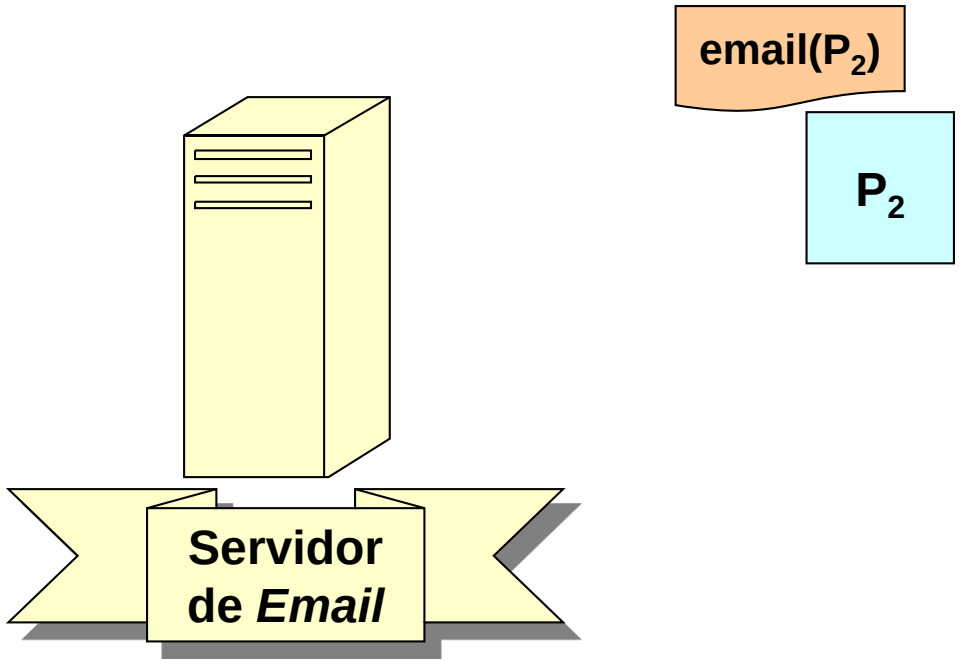
Taxonomia dos Modelos de Coordenação

- **Caixa de Correio:** *email*, quadro negro, filas de mensagens



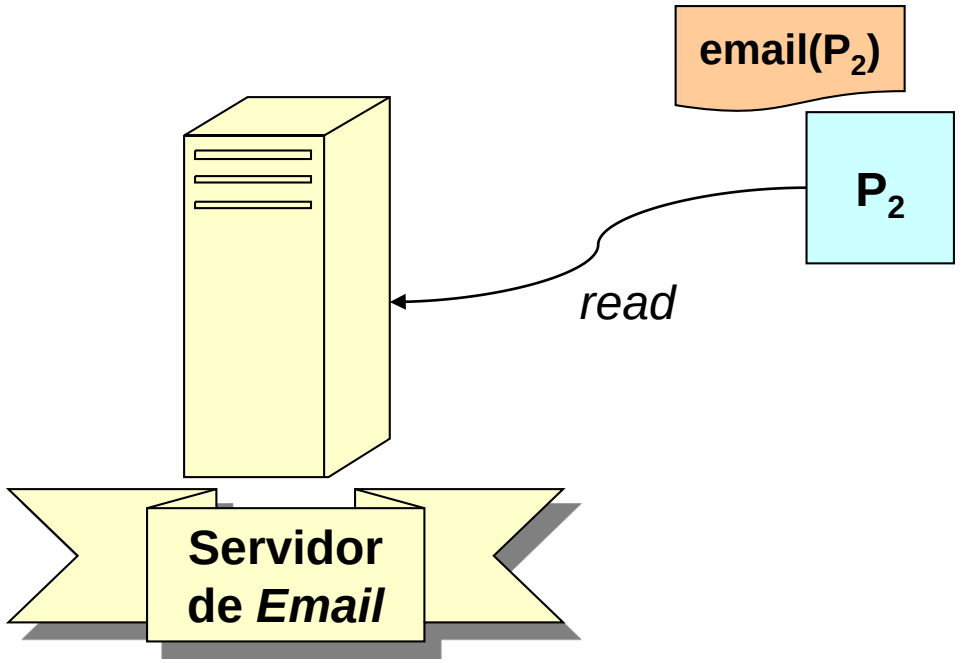
Taxonomia dos Modelos de Coordenação

- **Caixa de Correio:** *email*, quadro negro, filas de mensagens



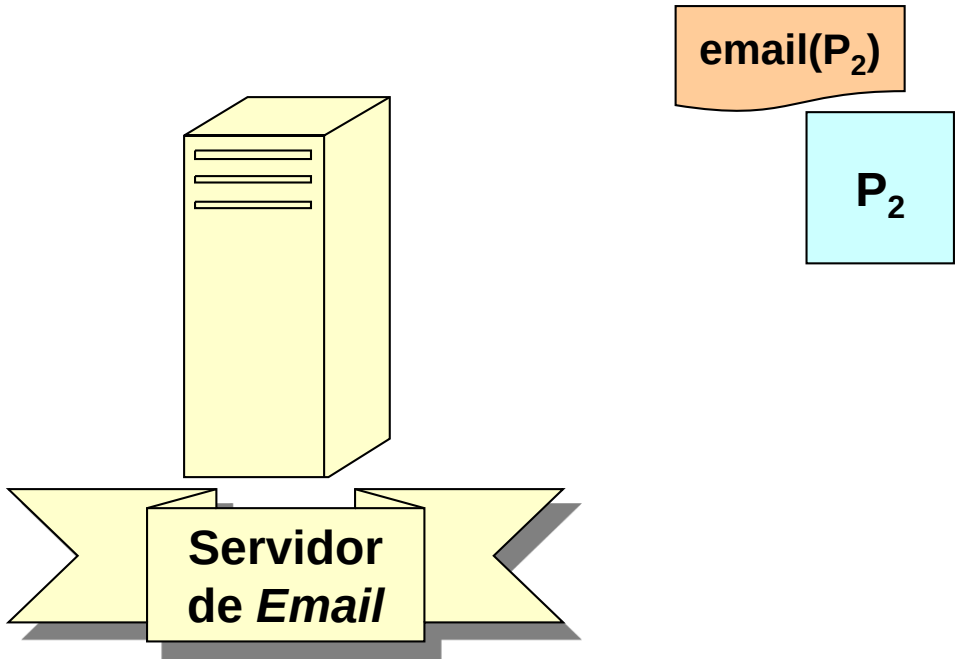
Taxonomia dos Modelos de Coordenação

- **Caixa de Correio:** *email*, quadro negro, filas de mensagens



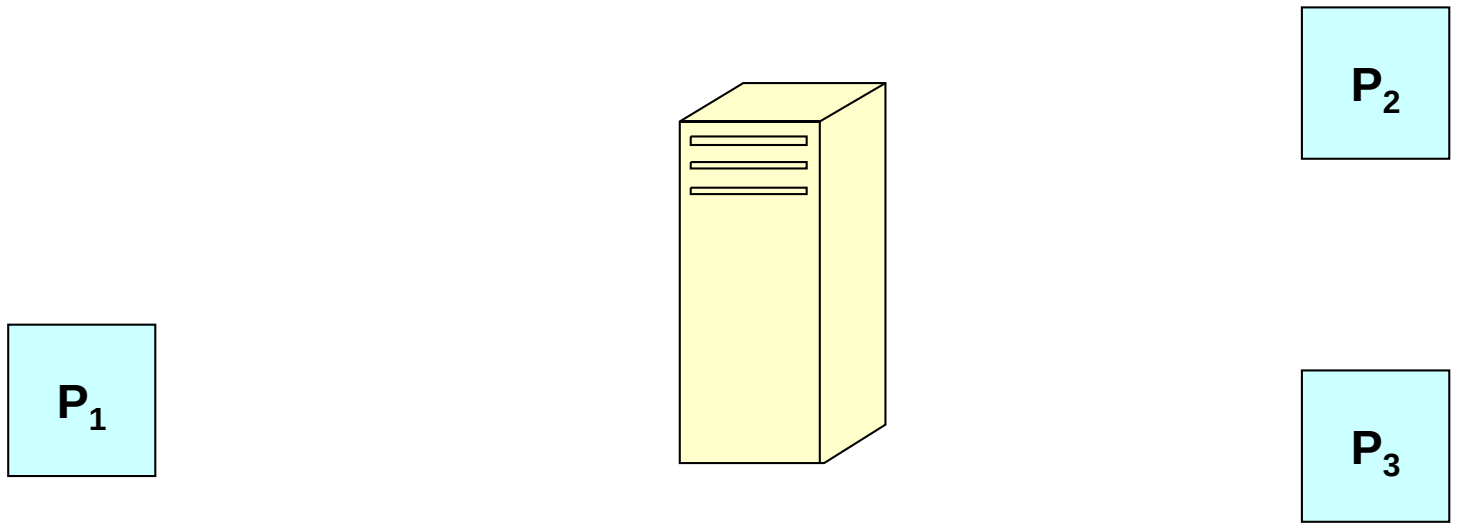
Taxonomia dos Modelos de Coordenação

- **Caixa de Correio:** *email*, quadro negro, filas de mensagens



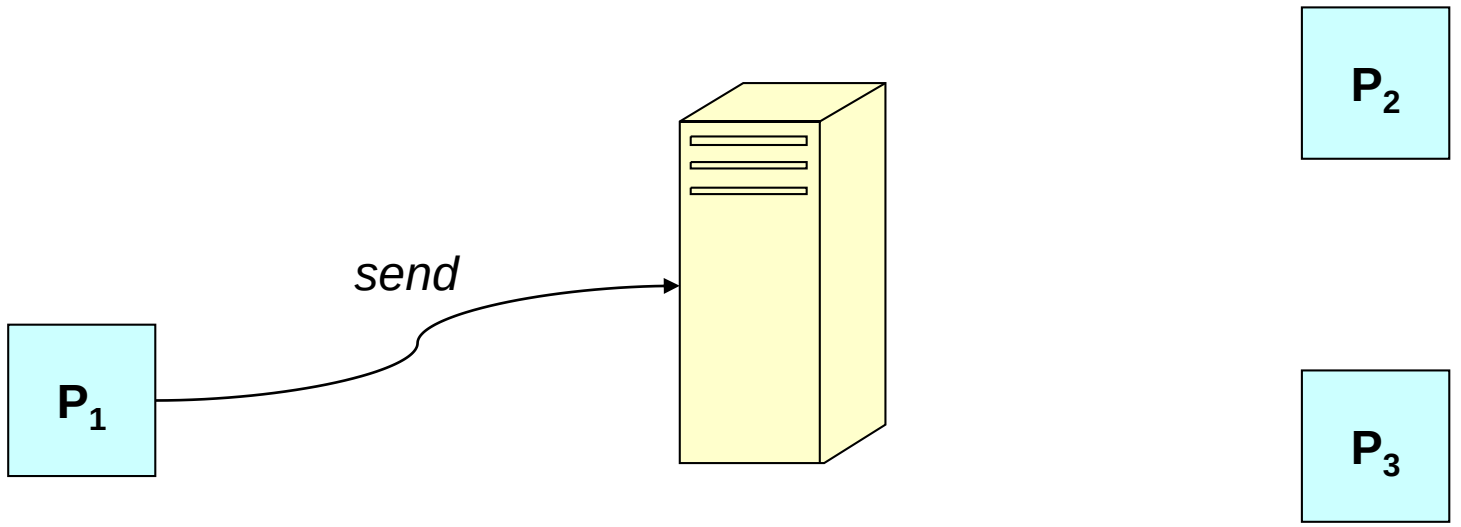
Taxonomia dos Modelos de Coordenação

- **Orientada a Encontro (*publisher/subscriber*):** serviços de eventos e notificação do CORBA, JMS, IBM MQ (*Message Queue*) Series.



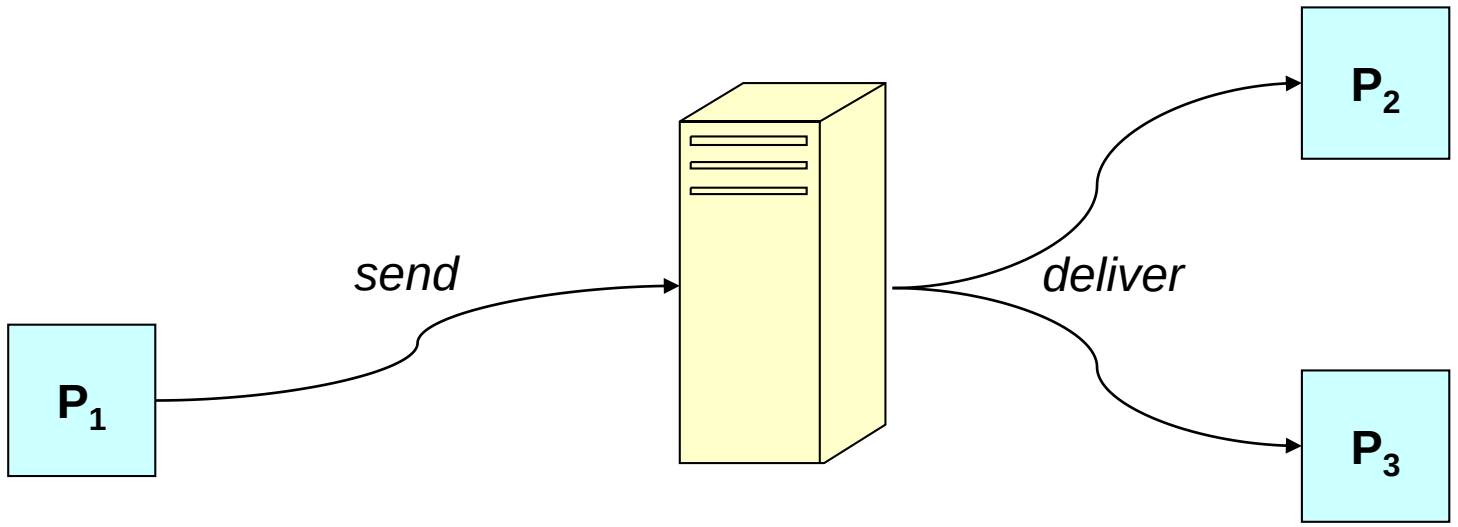
Taxonomia dos Modelos de Coordenação

- **Orientada a Encontro (*publisher/subscriber*):** serviços de eventos e notificação do CORBA, JMS, IBM MQ (*Message Queue*) Series.



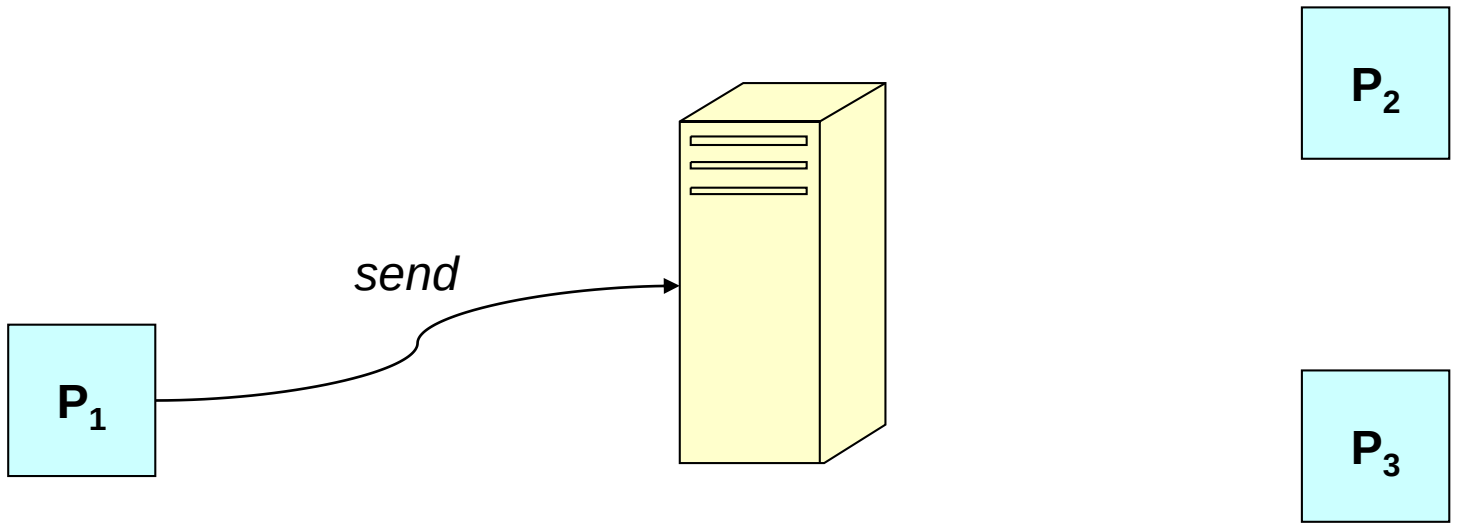
Taxonomia dos Modelos de Coordenação

- **Orientada a Encontro (*publisher/subscriber*):** serviços de eventos e notificação do CORBA, JMS, IBM MQ (*Message Queue*) Series.



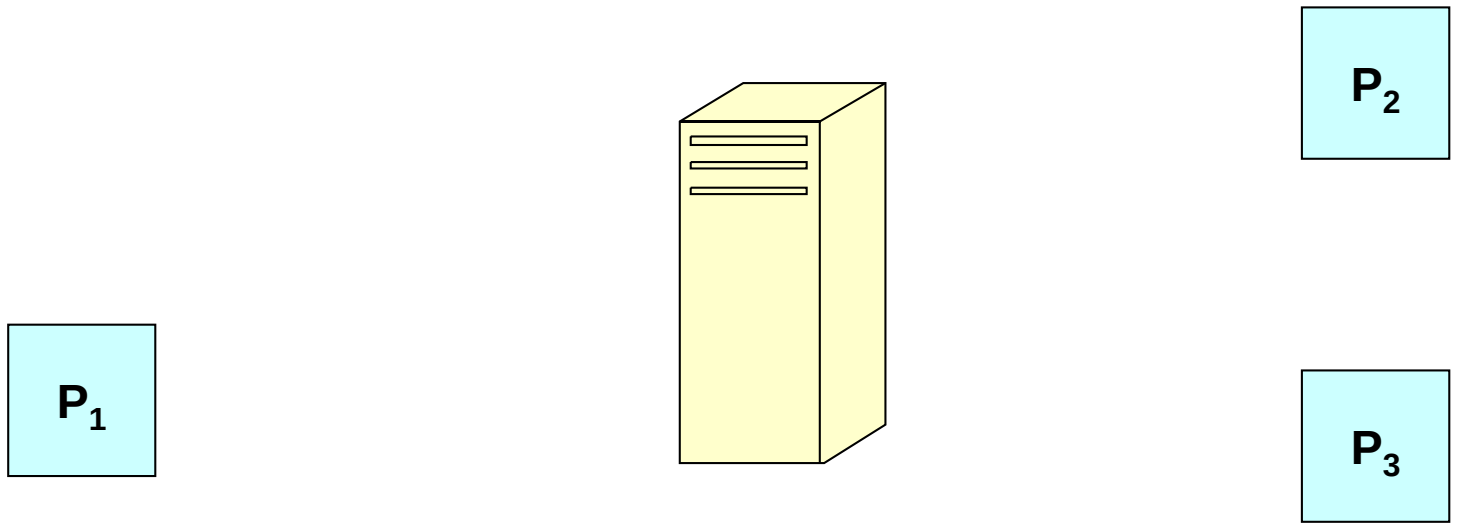
Taxonomia dos Modelos de Coordenação

- **Orientada a Encontro (*publisher/subscriber*):** serviços de eventos e notificação do CORBA, JMS, IBM MQ (*Message Queue*) Series.



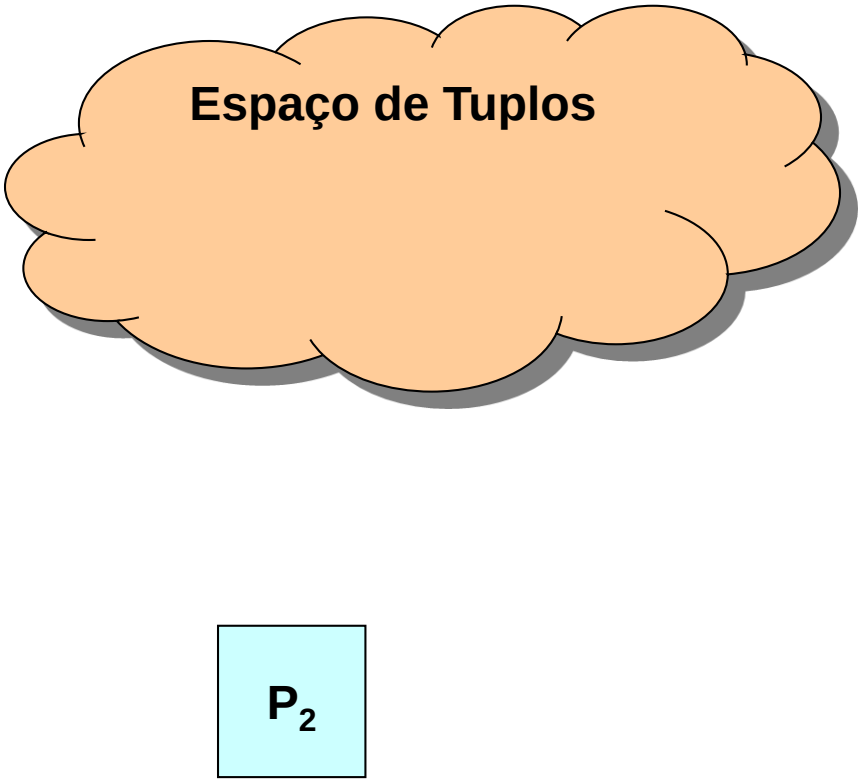
Taxonomia dos Modelos de Coordenação

- **Orientada a Encontro (*publisher/subscriber*):** serviços de eventos e notificação do CORBA, JMS, IBM MQ (*Message Queue*) Series.



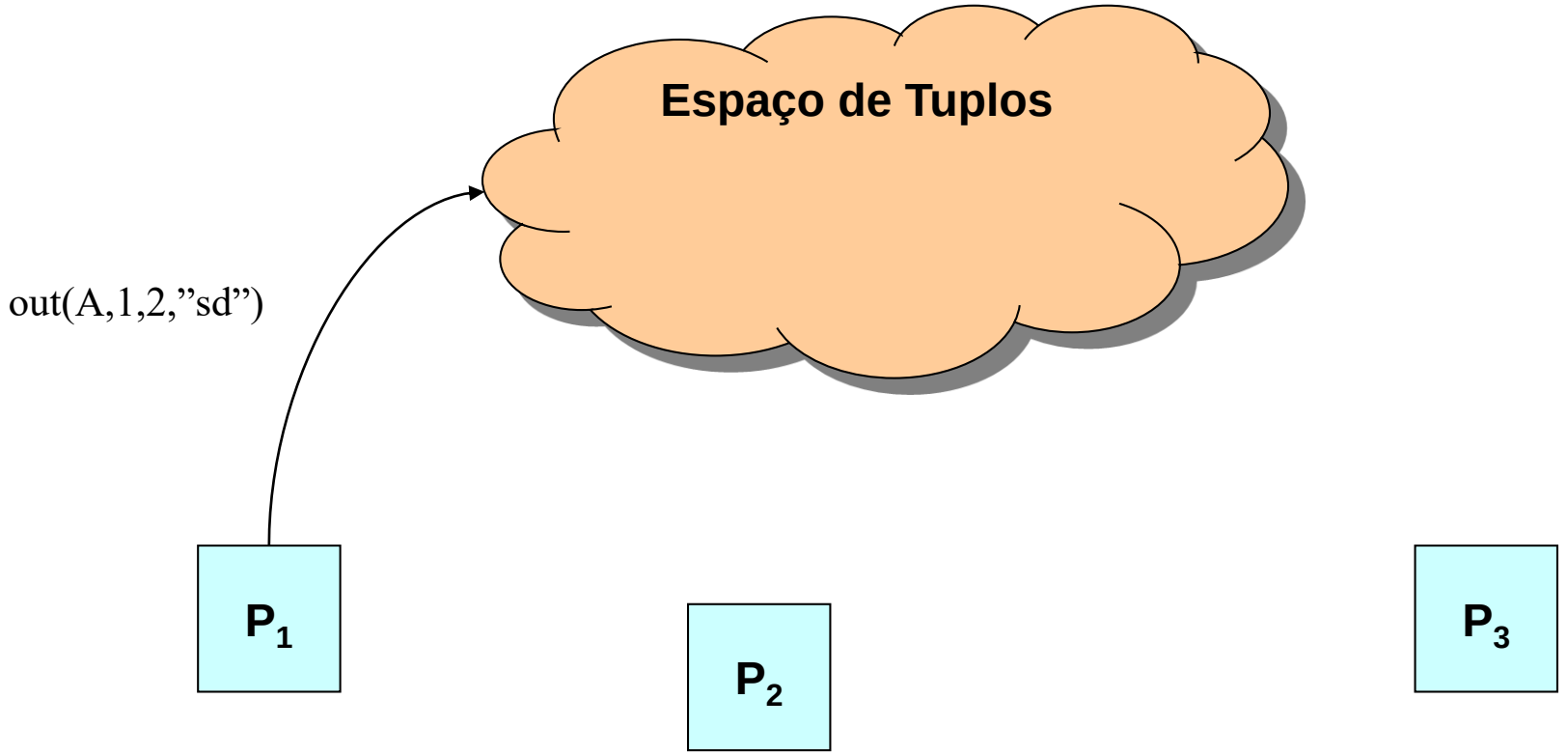
Taxonomia dos Modelos de Coordenação

- **Comunicação generativa:** Linda, JavaSpaces, IBM TSpace.



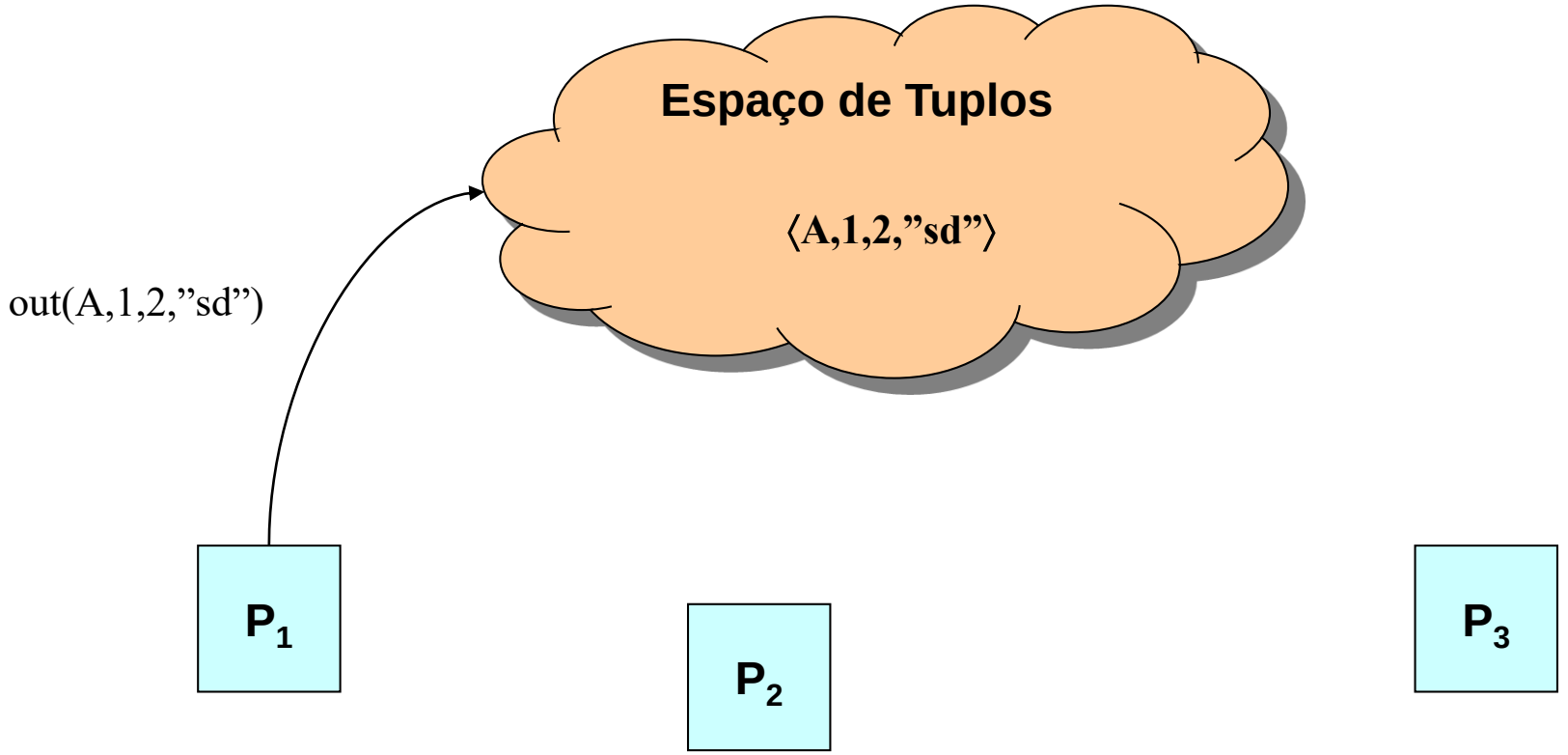
Taxonomia dos Modelos de Coordenação

- **Comunicação generativa:** Linda, JavaSpaces, IBM TSpace.



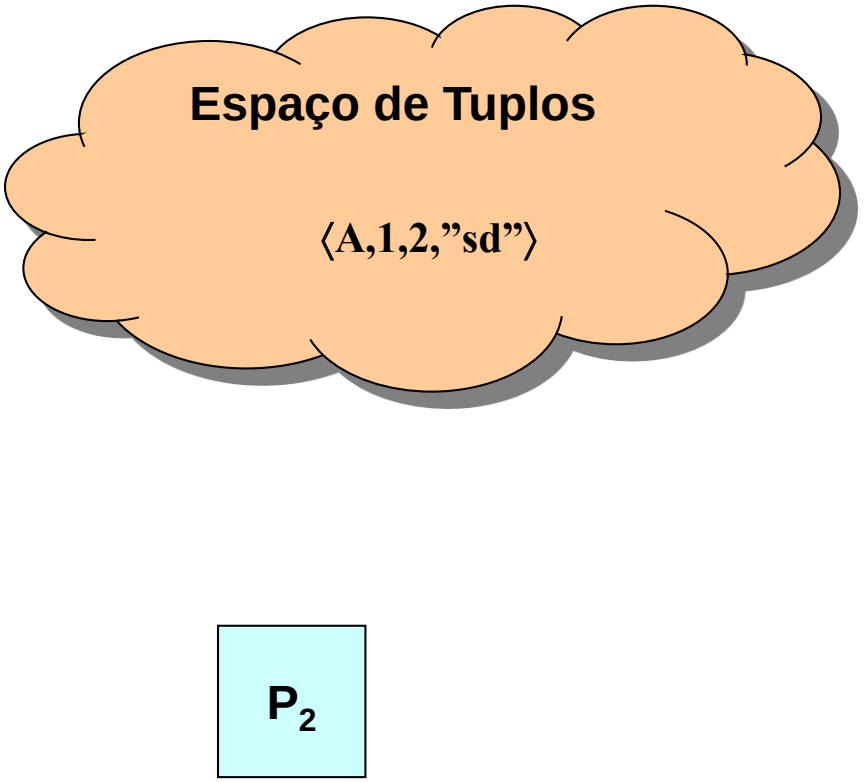
Taxonomia dos Modelos de Coordenação

- **Comunicação generativa:** Linda, JavaSpaces, IBM TSpace.



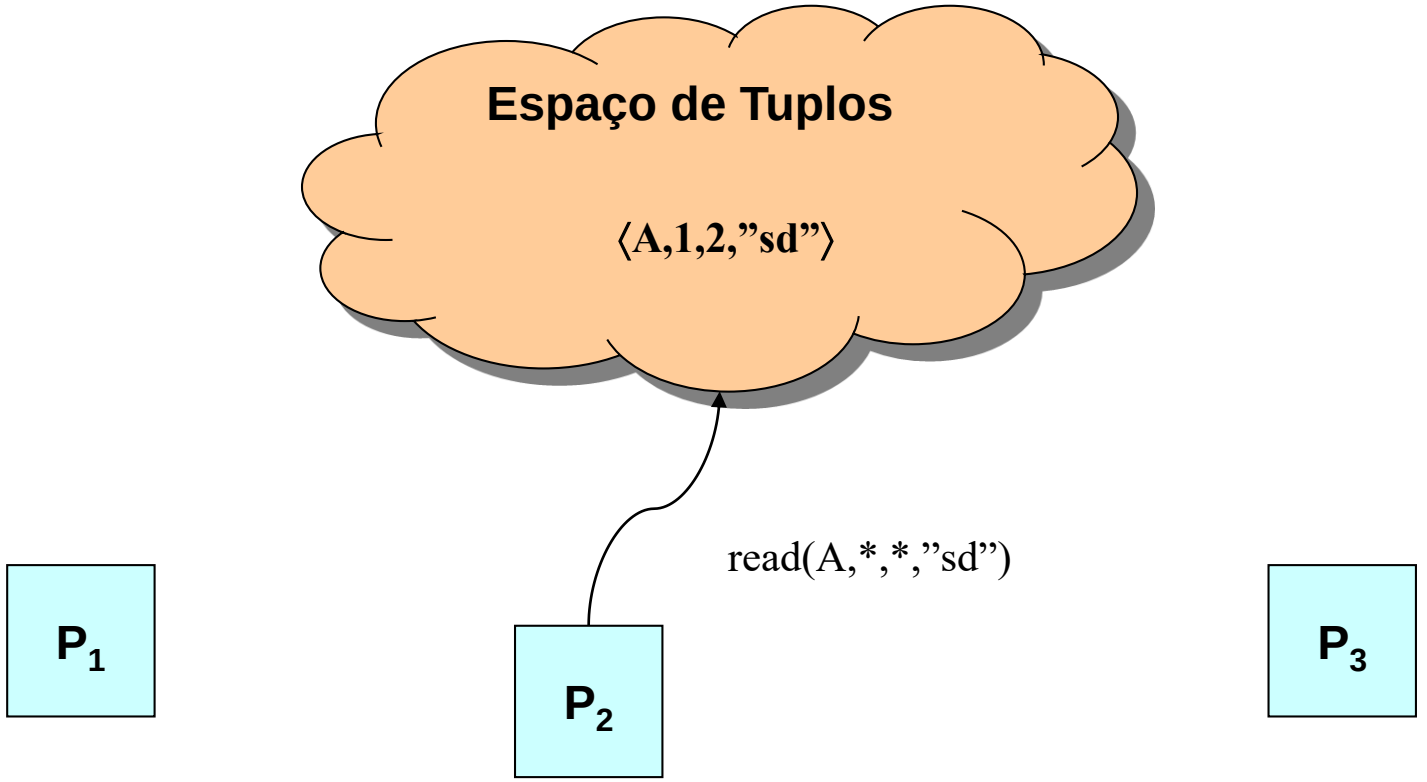
Taxonomia dos Modelos de Coordenação

- **Comunicação generativa:** Linda, JavaSpaces, IBM TSpace.



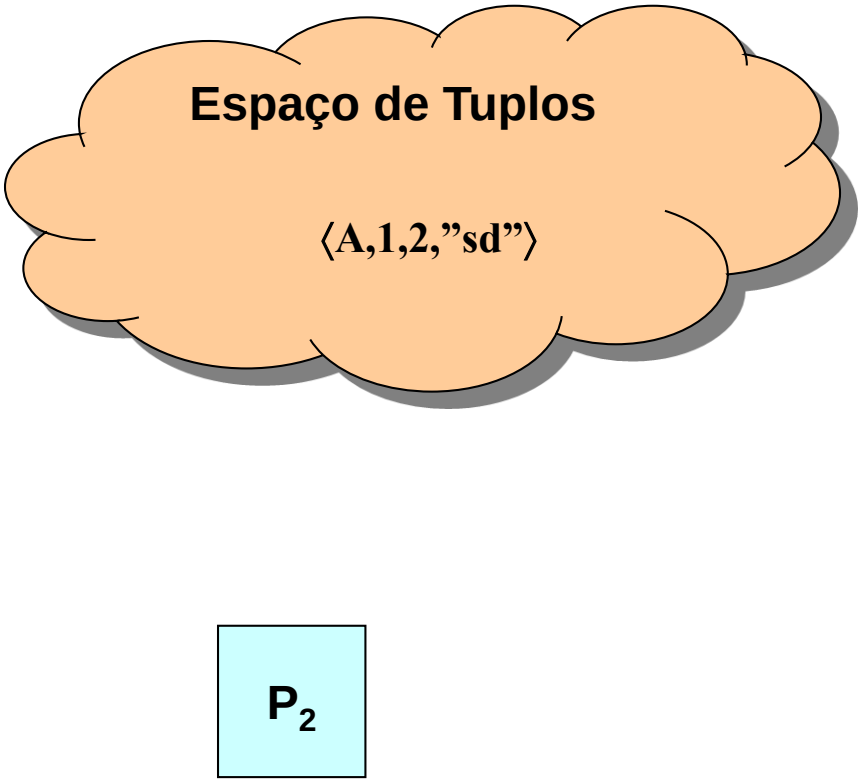
Taxonomia dos Modelos de Coordenação

- **Comunicação generativa:** Linda, JavaSpaces, IBM TSpace.



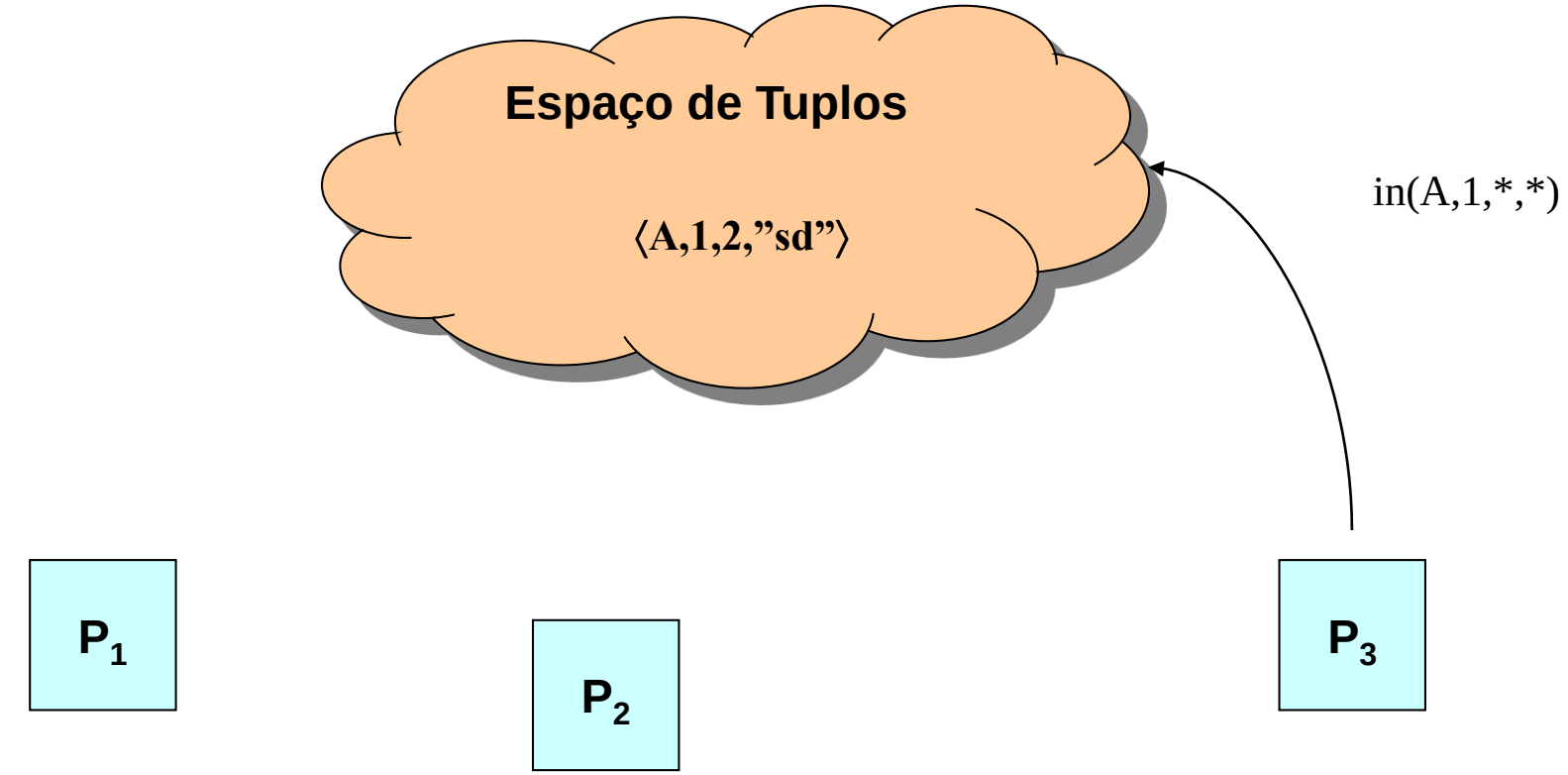
Taxonomia dos Modelos de Coordenação

- **Comunicação generativa:** Linda, JavaSpaces, IBM TSpace.



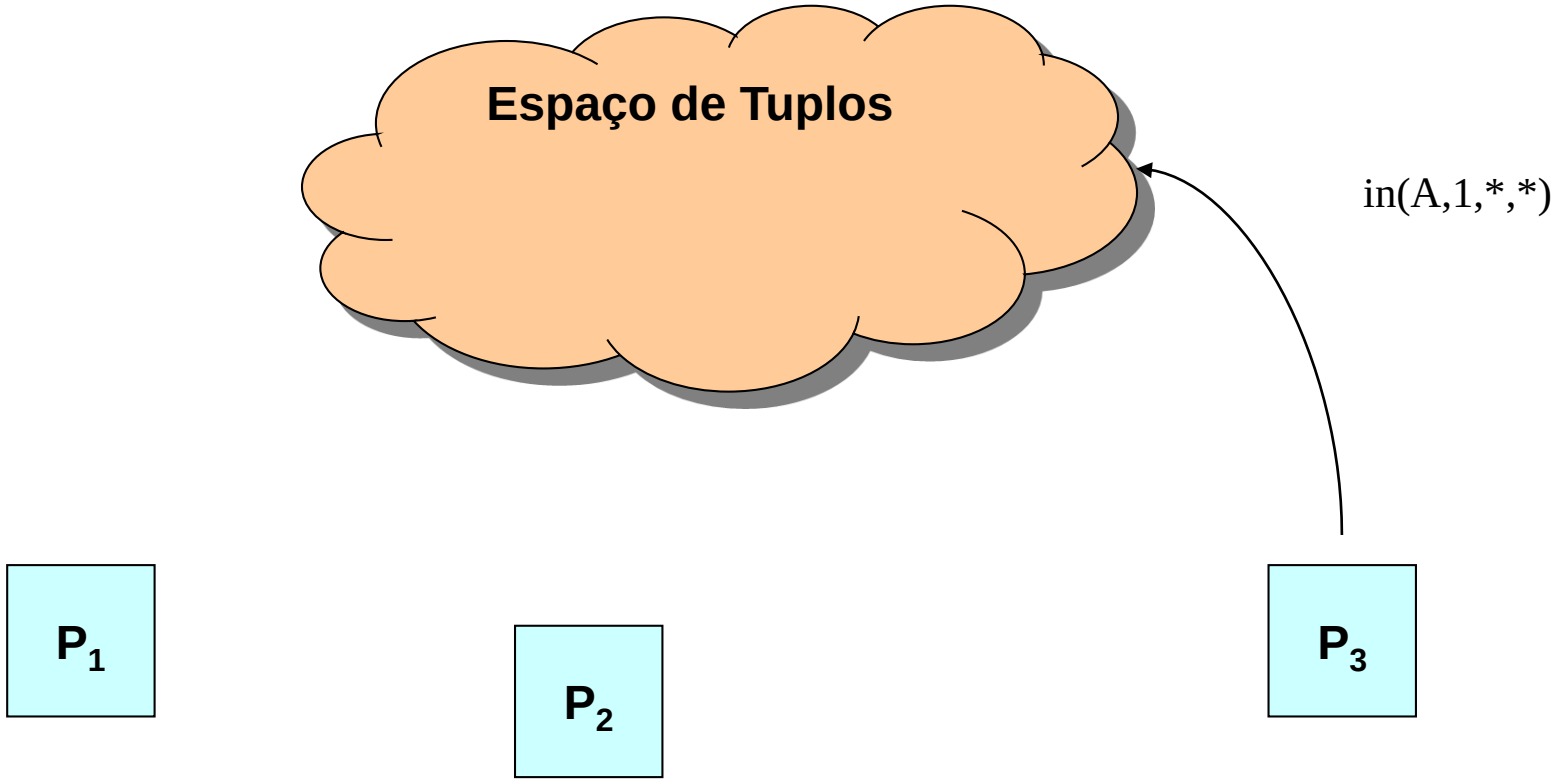
Taxonomia dos Modelos de Coordenação

- **Comunicação generativa:** Linda, JavaSpaces, IBM TSpace.



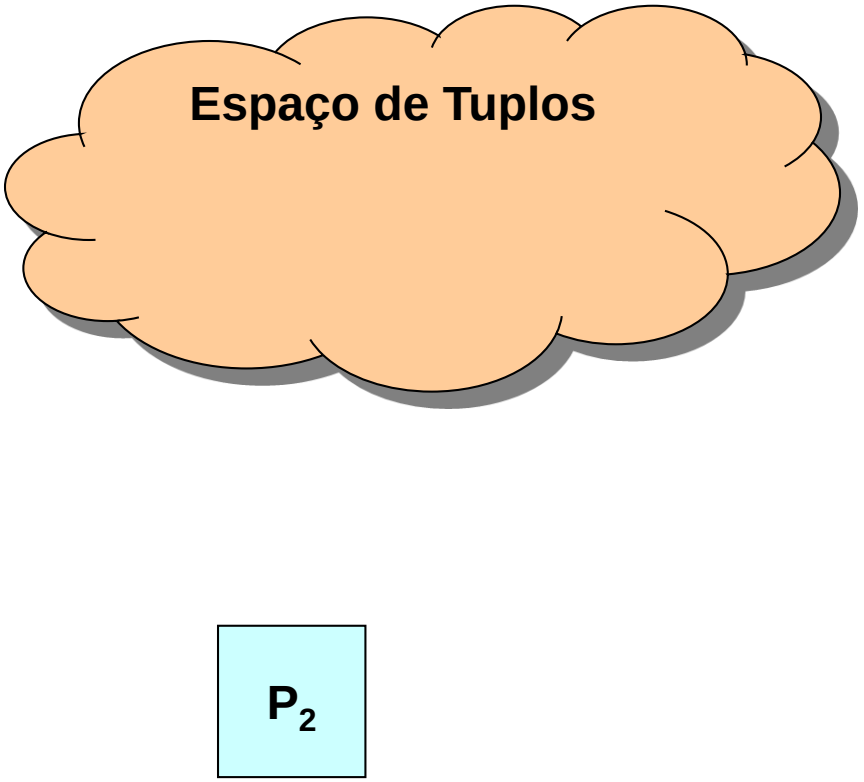
Taxonomia dos Modelos de Coordenação

- **Comunicação generativa:** Linda, JavaSpaces, IBM TSpace.



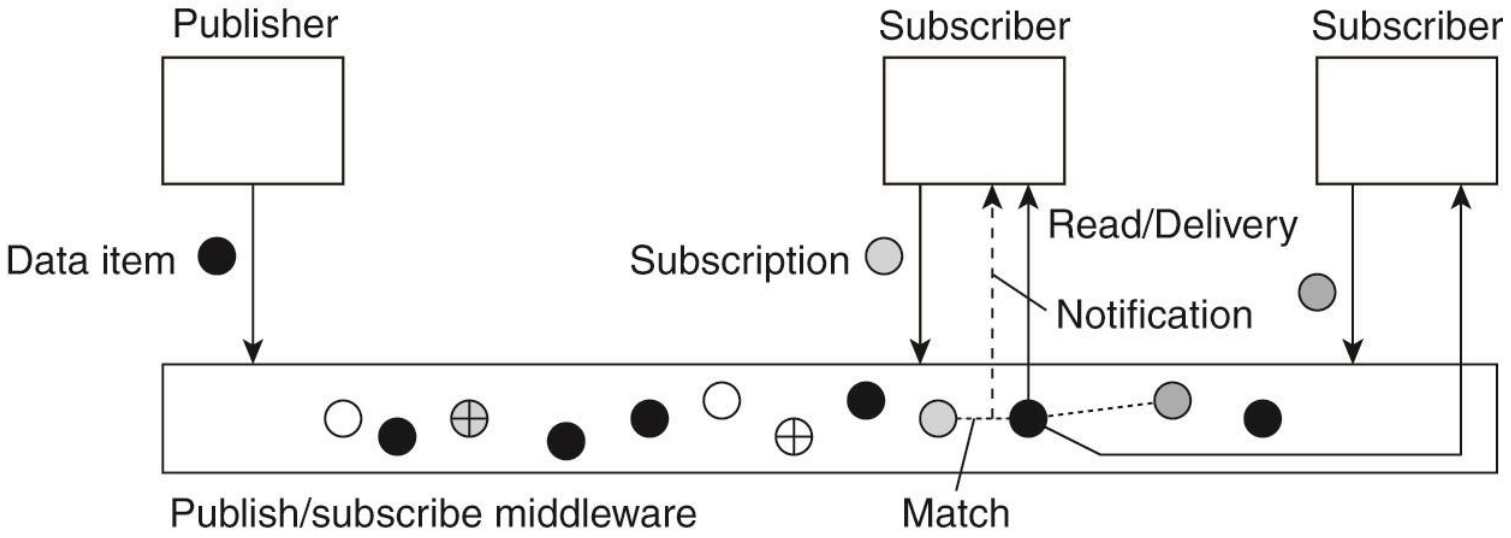
Taxonomia dos Modelos de Coordenação

- **Comunicação generativa:** Linda, JavaSpaces, IBM TSpace.



Publisher/Subscriber (Orientada a Encontro)

- Temos um **barramento de eventos** (*event bus*) onde:
  - os processos *publishers* publicam eventos
  - os processos *subscribers* se registam para obter certos tipos de eventos
    - » a subscrição pode ser restrita a eventos com certas propriedades
      - Só eventos que combinam com a subscrição são enviados ao processo
    - » a subscrição pode ser restrita a um certo canal
      - A ideia de canal é semelhante a um fórum ou lista de discussão
- A arquitetura desse tipo de *middleware* segue a das filas de mensagens



Coordenação Generativa

- Coordenação generativa (ou comunicação generativa) é um modelo de coordenação (ou comunicação) que utiliza um espaço de memória partilhada para **tuplos** (itens de dados genéricos)
- Esta memória é o espaço de tuplos:
  - É uma **memória partilhada real** porque qualquer processo deve poder referenciar um tuplo no espaço, independentemente da sua localização
  - É uma **memória partilhada lógica** porque não tem necessariamente de ser concretizada numa memória partilhada física
  - É uma **memória associativa** porque os tuplos não são acedidos pelo seu endereço mas sim pelo seu conteúdo
- Conclusão: o espaço de tuplos é memória partilhada, mas foi desenhado a pensar em redes de computadores

Coordenação Generativa

- Definições:
  - Tuplo:
    - » Conjunto de elementos $t = \langle e_1,..,e_j \rangle$
  - Molde:
    - » Tuplo *t'* onde alguns elementos podem ser indefinidos (*)
  - Combinação entre tuplo e molde:
    - » Mesmo número de elementos
    - » Os elementos definidos são iguais
  - Operações:
    - » **out(t)**: produz um tuplo no espaço
    - » **in(t')**: consome um tuplo do espaço que combina com *t'*
      - Ex: in(<a,b,*>) lê <a,b,c>
    - » **rd(t')**: lê um tuplo do espaço que combina com *t'*

Coordenação Generativa

- As operações são:
  - Bloqueantes:
    - » **in** e **rd** ficam bloqueadas até que exista um tuplo que combine com o molde passado como parâmetro
    - » Obs: existem variantes não-bloqueantes: **inp** e **rdp**
  - Atómicas:
    - » as operações **out**, **in** e **rd** são indivisíveis
  - Não deterministas:
    - » Se vários tuplos no espaço combinam com o molde passado em **in** ou **rd**, qualquer um deles pode ser retornado
    - » Se vários processos esperam para consumir um tuplo, qualquer um deles pode receber este tuplo quando ele estiver disponível no sistema

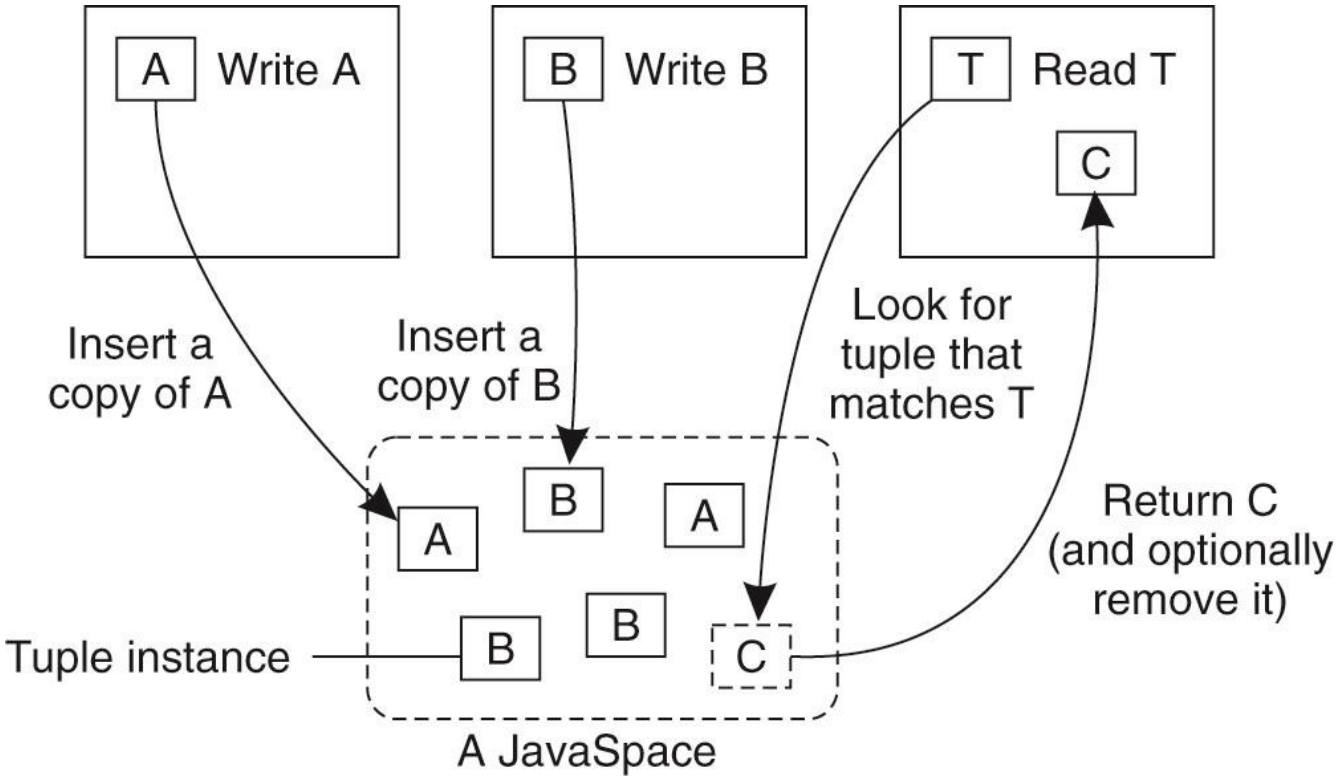
JavaSpaces

- Serviço de espaço de tuplos da arquitectura Jini

- O *JavaSpaces* suporta as operações usuais da coordenação generativa:
  - *write = out*
  - *read = rd*
  - *take = in*
  - Versões não bloqueantes: *takeIfExists* e *readIfExists*
  - Existe também a operação *notify*, que permite a recepção de eventos sobre a existência de um tuplo no espaço

- O *JavaSpaces* suporta transacções e notificações de eventos através da integração com os outros serviços do Jini

JavaSpaces



- A concretização usual de um *JavaSpace* consiste em um servidor acedido via JavaRMI que armazena tuplos

JavaSpaces

[589-595]

```
public interface JavaSpace {
        Lease write(Entry e, Transaction txn, long lease);
        Entry read(Entry tmpl, Transaction txn, long timeout);
        Entry readIfExists(Entry tmpl, Transaction txn, long timeout);
        Entry take(Entry tmpl, Transaction txn, long timeout);
        Entry takeIfExists(Entry tmpl, Transaction txn, long timeout);
        EventRegistration notify(Entry tmpl, Transaction txn,
                                 RemoteEventListener listener, long lease);
        Entry snapshot(Entry e);
}
```

Interface do serviço *JavaSpaces* (simplificada)

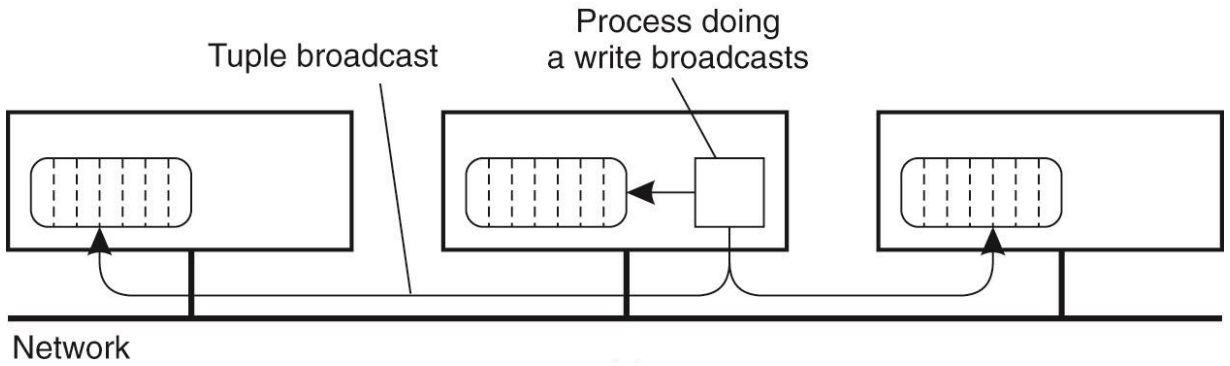
JavaSpaces: Concretização Distribuída

- Uma questão fundamental quando se pensa no modelo de espaço de tuplos é como garantir que o espaço vai estar sempre disponível e não será um *bottleneck*
  - A resposta para essa questão passa pela concretização distribuída do espaço
  - Vamos analisar como isso pode ser feito com o *JavaSpaces*

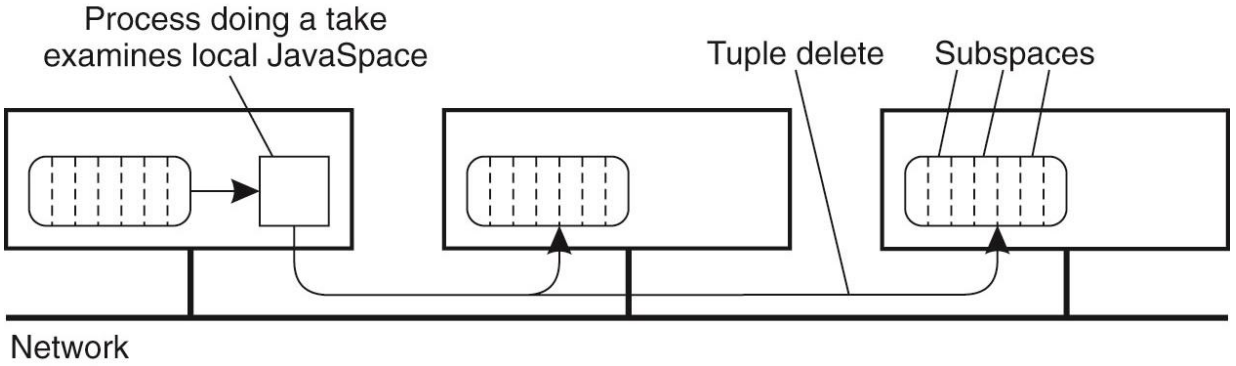
- O *JavaSpaces* pode ser replicado com o objectivo de:
  - Aumentar a **disponibilidade** dos tuplos (tolerância a faltas)
  - Melhorar a **escalabilidade** do sistema (balanceamento de carga)

- A concretização requer a resolução de dois problemas:
  1. Como simular o endereçamento associativo sem usar *flooding*
  2. Como distribuir os tuplos pelas máquinas e localizá-las depois

JavaSpaces: Replicação de Tuplos



Um tuplo é inserido em todas as instâncias do espaço na rede



As leituras podem ser feitas localmente, mas as remoções devem ser feitas em todas as máquinas

Vantagem:

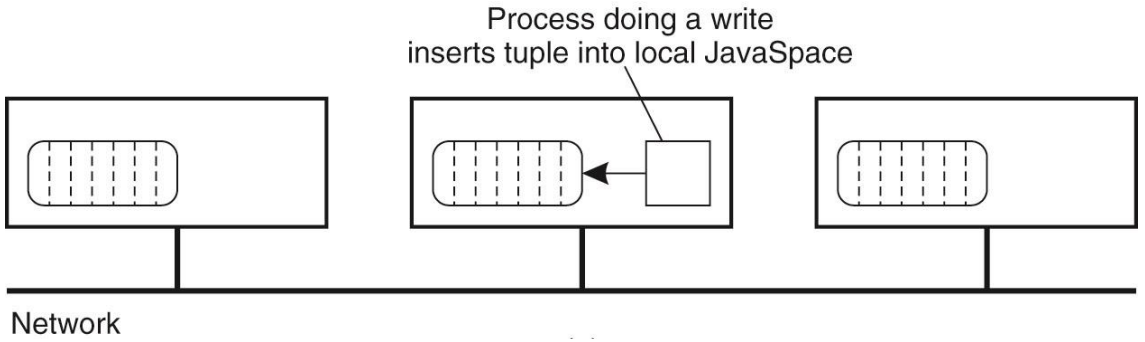
- **Tolera faltas**

Desvantagens:

- **Inserção/remoção pesada**
- **Má utilização de memória**
- **Não há consistência sequencial**

Pode-se concretizar usando algum dos algoritmos para replicação vistos nas aulas, de forma a satisfazer consistência sequencial.

JavaSpaces: Distribuição de Tuplos



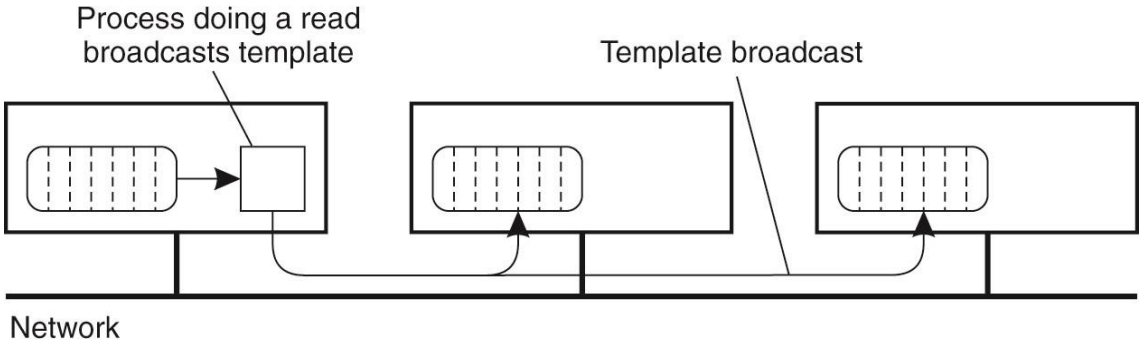
Um tuplo é inserido localmente e removido em apenas uma máquina

Vantagem:

- **Melhor utilização da memória**
- **Satisfaz consistência sequencial**

Desvantagens:

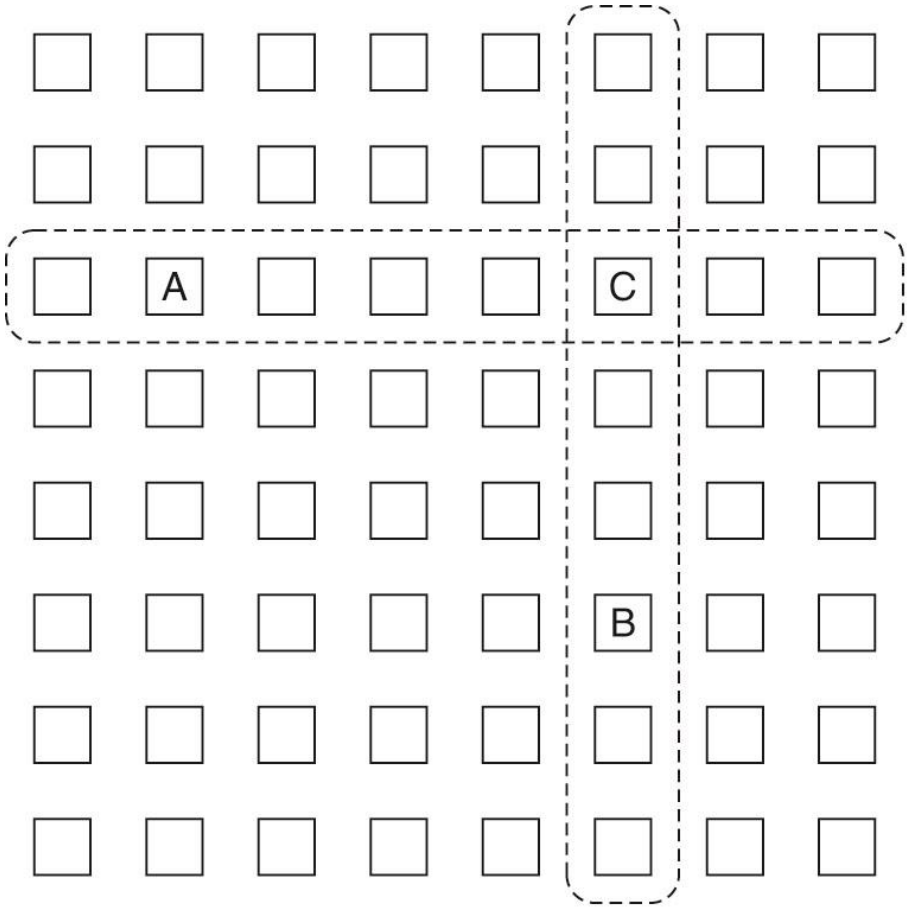
- **Não tolera faltas**
- **Leitura pesada (*flooding*)**



As leituras devem ser enviadas a todas as instâncias do espaço na rede

JavaSpaces: Concretização Híbrida

[607-611]



Insere os tuplos numa linha.

Busca tuplos numa coluna.

Vantagens:
- **Tolera faltas**
- **Distribui carga**

Desvantagens:
- **Requer mais máquinas**

Este modelo tem relação com os sistemas de quórum que vimos nas aulas de tolerância a faltas