Computação Distribuída

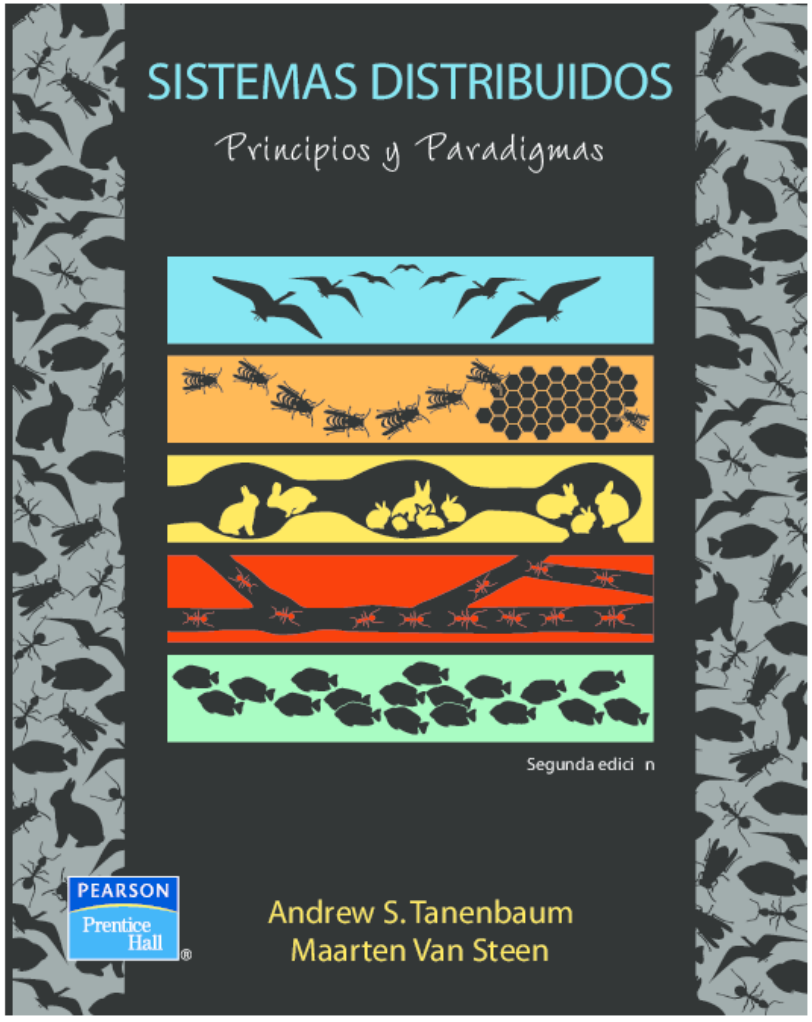
Comunicação

CHAPTER 4

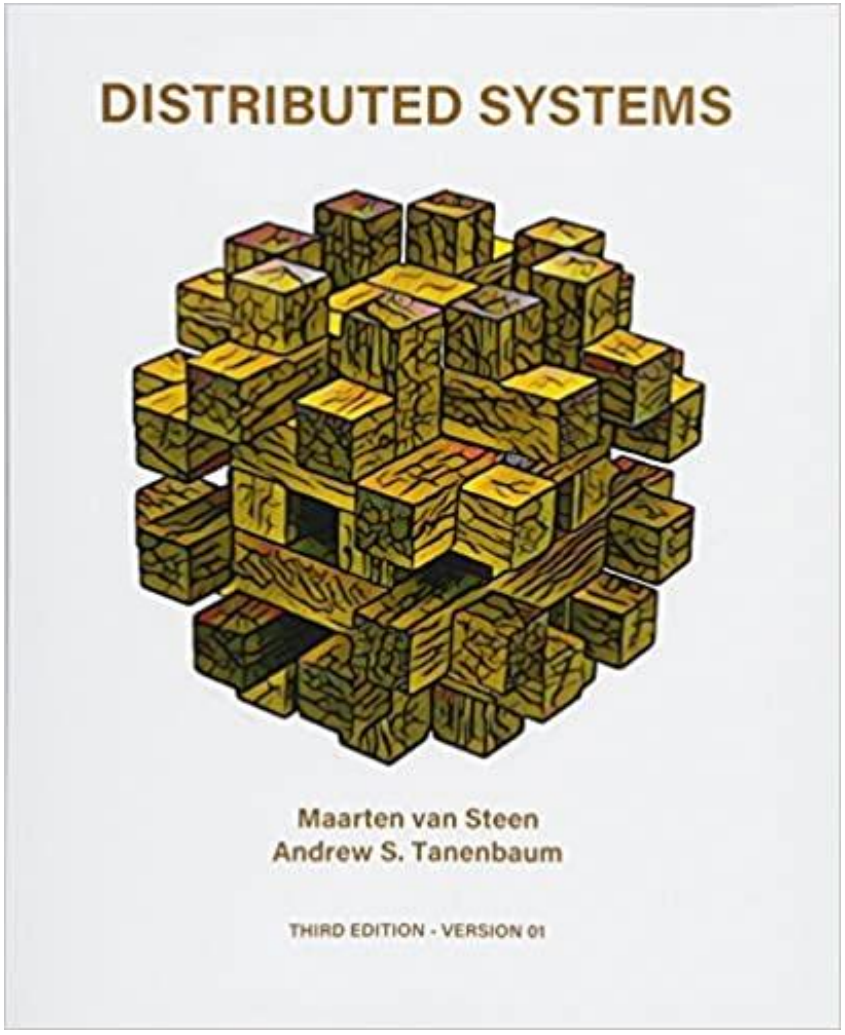
COMMUNICATION

Vladimir Rocha (Vladi)

CMCC - Universidade Federal do ABC


SISTEMAS DISTRIBUIDOS
Principios y Paradigmas
Segunda edici n
PEARSON
Prentice
Hall
Andrew S. Tanenbaum
Maarten Van Steen

2007


DISTRIBUTED SYSTEMS
Maarten van Steen
Andrew S. Tanenbaum
THIRD EDITION - VERSION 01

2017

Disclaimer

- Estes slides foram baseados nos do professor **Emilio Francesquini** para o curso de Sistemas Distribuídos na UFABC.
- Este material pode ser usado livremente desde que sejam mantidos, além deste aviso, os créditos aos autores e instituições.
- Estes slides foram adaptados daqueles originalmente preparados (e gentilmente cedidos) pelo professor **Daniel Cordeiro, da EACH-USP** que por sua vez foram baseados naqueles disponibilizados online pelos autores do livro "Distributed Systems", 3ª Edição em: https://www.distributed-systems.net.

Agenda

- Fundamentos
- Comunicação orientada a procedimento (RPC)
- Comunicação orientada a mensagem (MOM)
- Comunicação multicast (FIFO/Causal/Total – Gossip/Flooding)
- Comunicação orientada a fluxo

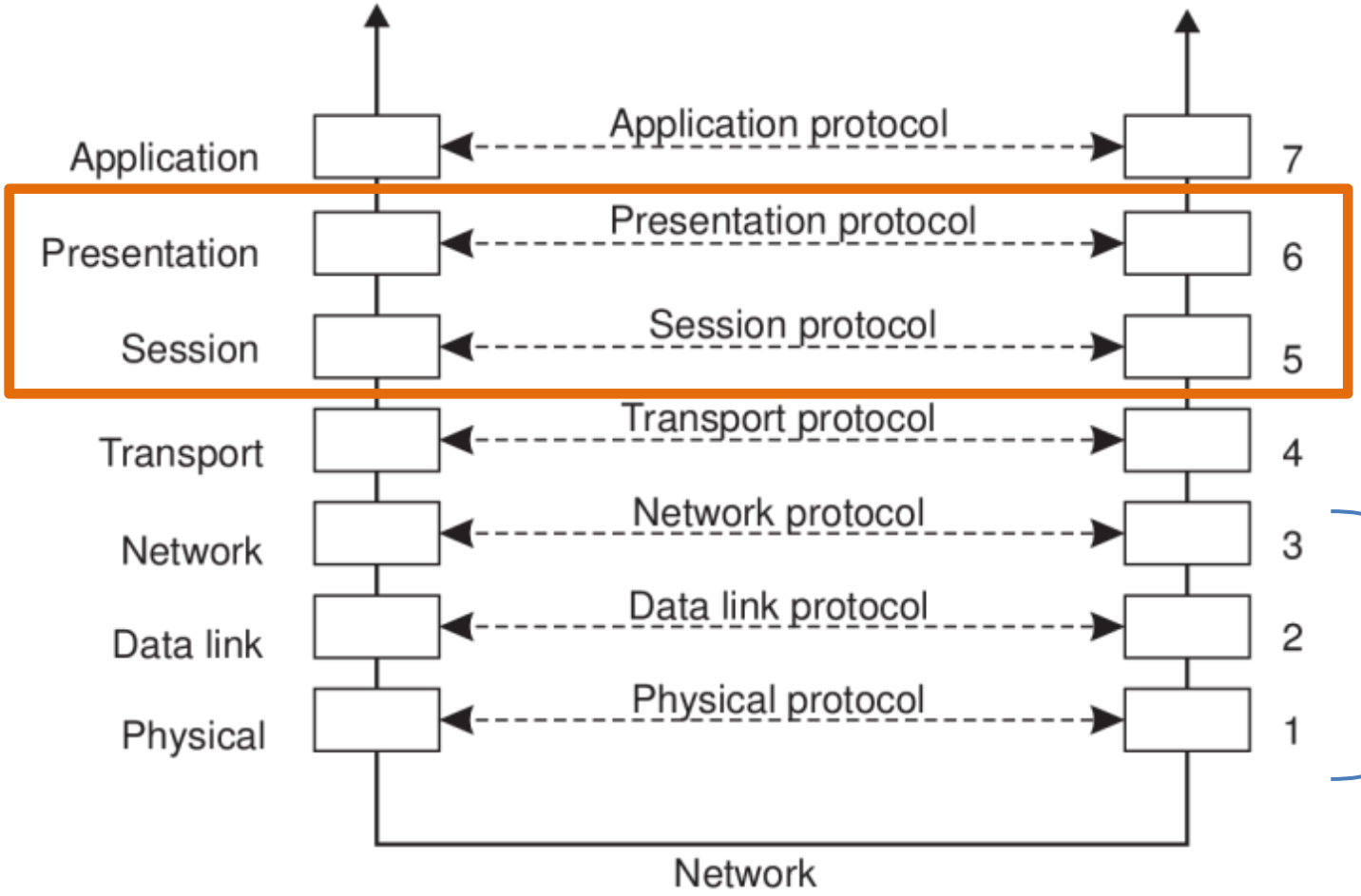
Agenda

- Fundamentos
- Comunicação orientada a procedimento (RPC)
- Comunicação orientada a mensagem (MOM)
- Comunicação multicast (Gossip / Flooding)
- Comunicação orientada a fluxo

Protocolos em camadas

- Camadas de baixo nível
- Camada de transporte
- Camada do middleware
- Camada de aplicação
- Tipos de comunicação

Modelo de comunicação básico



Desvantagens:

- Funciona apenas com passagem de mensagens
- Frequentemente possuem funcionalidades desnecessárias
- Viola a transparência de acesso

Camada de transporte

Importante:

A camada de transporte fornece as ferramentas de comunicação efetivamente utilizadas pela maioria dos sistemas distribuídos.

Protocolos padrões da Internet

- TCP: orientada a conexão, confiável, comunicação orientada a fluxo de dados
- UDP: comunicação de datagramas não confiável (*best-effort*)

Nota:

IP multicasting é normalmente considerado um serviço padrão (mas essa é uma hipótese perigosa)

Camada de middleware

Middleware foi inventado para prover serviços e protocolos frequentemente usados que podem ser utilizados por várias aplicações diferentes.

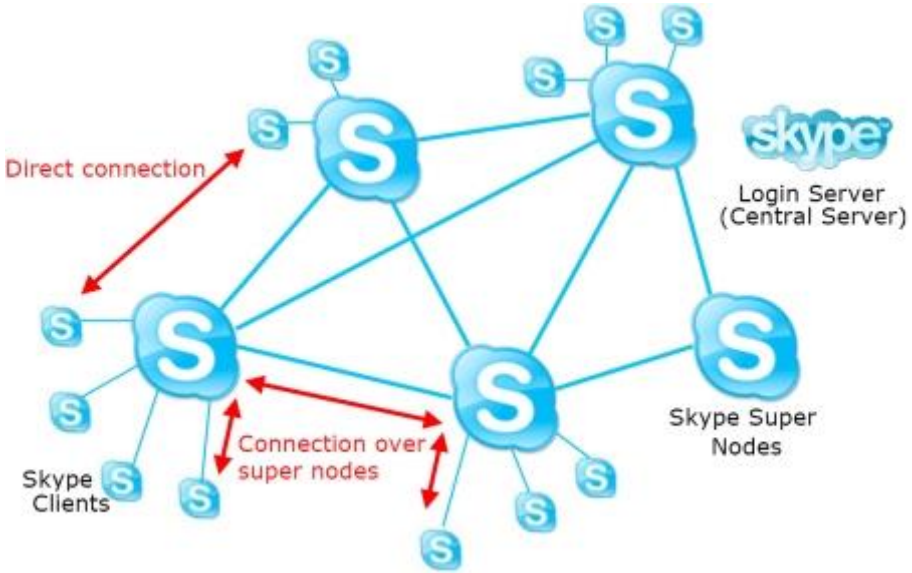
- Um conjunto rico de protocolos de comunicação
- (Des)empacotamento [*(un)marshaling*] de dados, necessários para a integração de sistemas
- Protocolos de gerenciamento de nomes, para auxiliar o compartilhamento de recursos
- Protocolos de segurança para comunicações seguras
- Mecanismos de escalabilidade, como replicação, sharding e caching

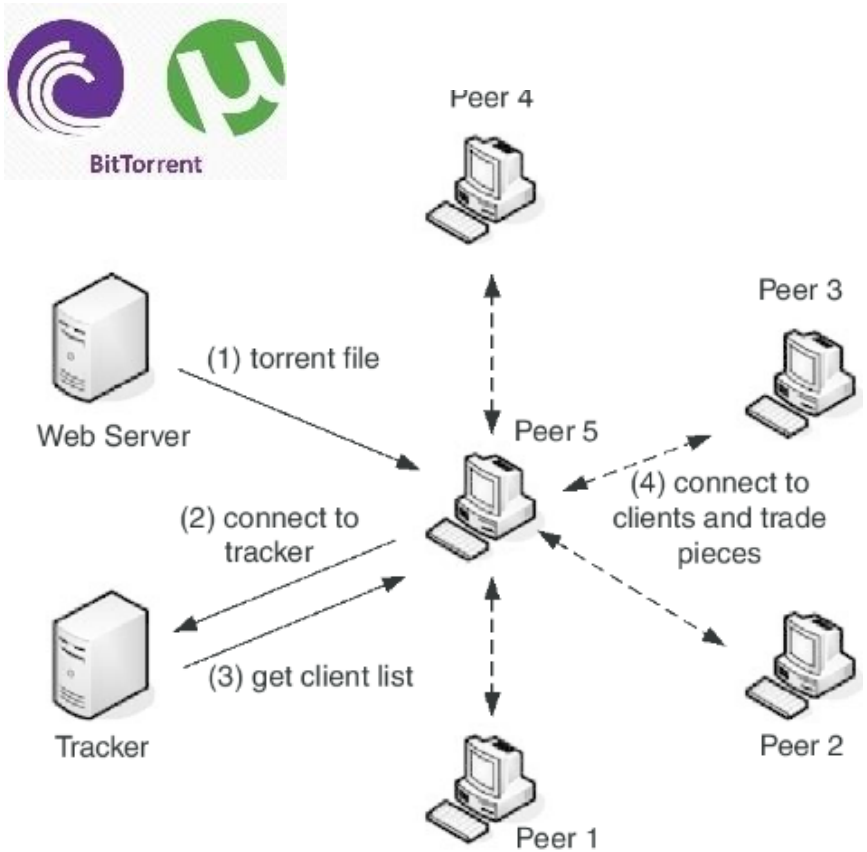
Observação:

O que realmente sobra são protocolos específicos de aplicação.

Camada de aplicação

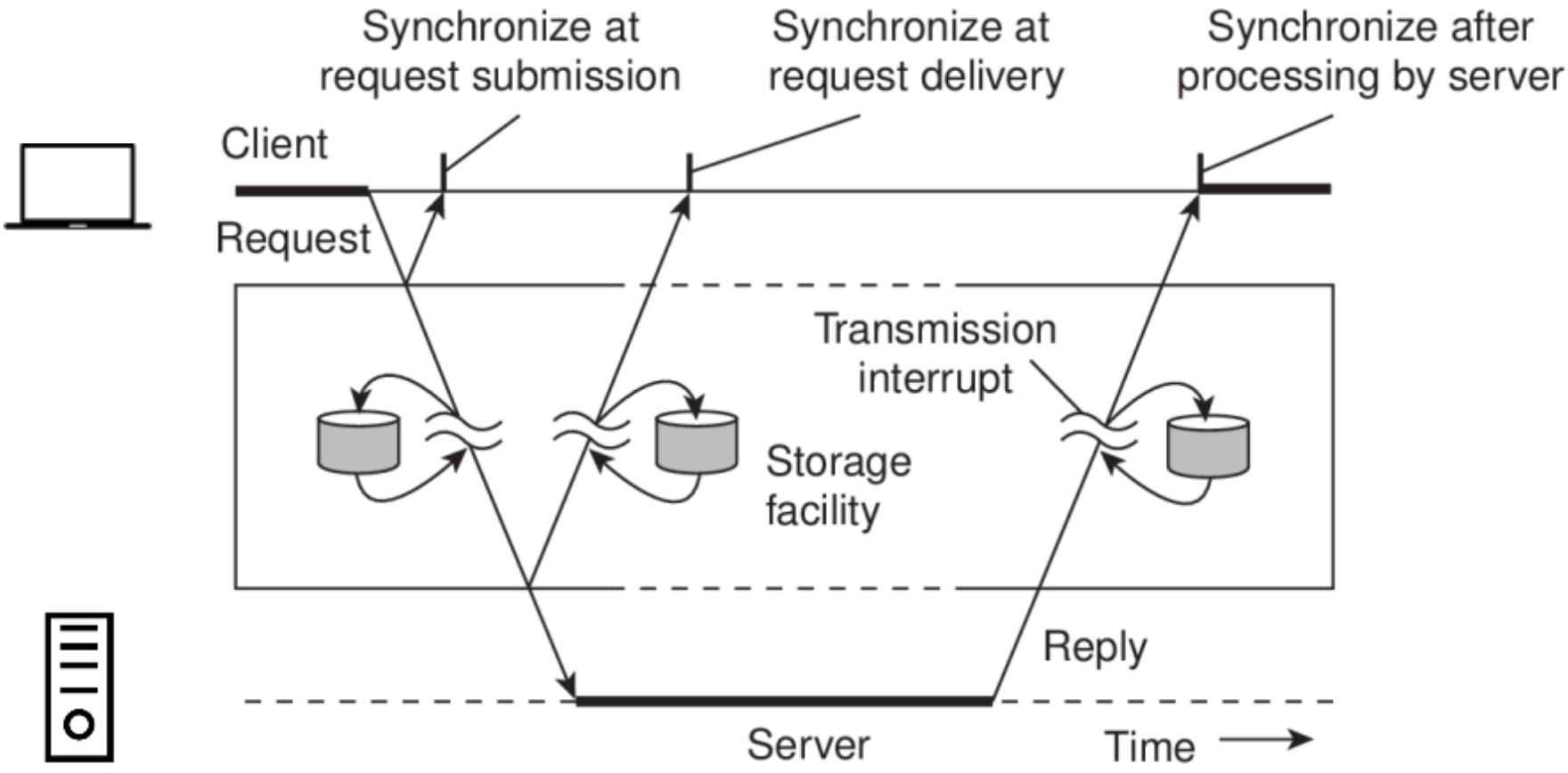
www.img.lx.it.pt%2F~fp%2Fcomunicacao_multimedia%2Fano%2520lectivo_2016_2017%2FTrabalhos_CMul_2016_2017%2FGrupo%25203%2Farquitetura.html





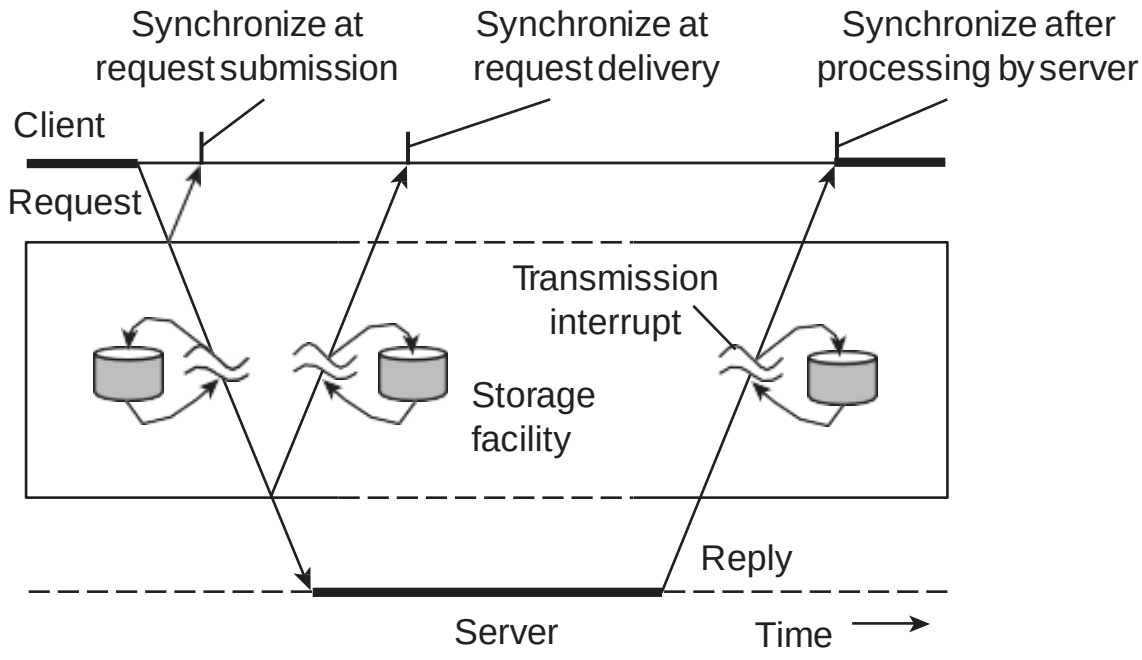
https://www.researchgate.net/publication/225633414_Modeling_energy_efficiency_of_file_sharing/figures?lo=1

Tipos de comunicação



- Comunicação transiente vs. persistente
- Comunicação assíncrona vs. síncrona

Tipos de comunicação

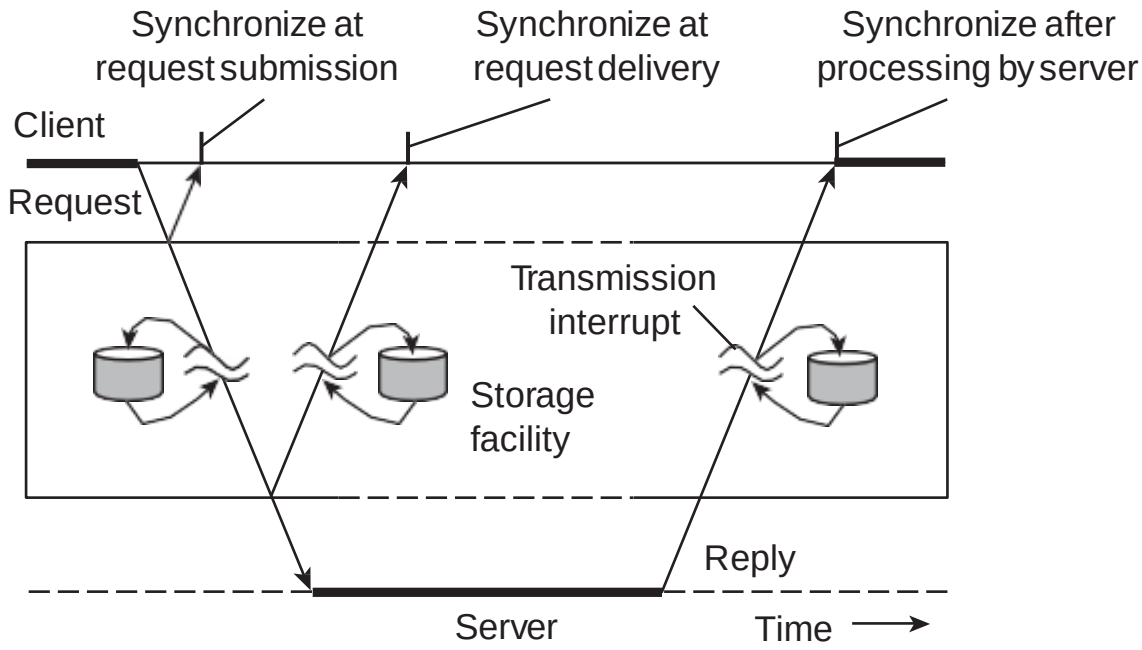


Transiente vs. persistente

Comunicação transiente: remetente descarta a mensagem se ela não puder ser encaminhada para o destinatário

Comunicação persistente: uma mensagem é guardada no remetente pelo tempo que for necessário, até ser entregue no destinatário

Tipos de comunicação



Pontos de sincronização

- No envio da requisição
- Na entrega da requisição
- Após o processamento da requisição

Agenda

- Fundamentos
- Comunicação orientada a procedimento (RPC)
- Comunicação orientada a mensagem (MOM)
- Comunicação multicast (Gossip / Flooding)
- Comunicação orientada a fluxo

RMI

Chamadas a procedimentos remotos (RPC)

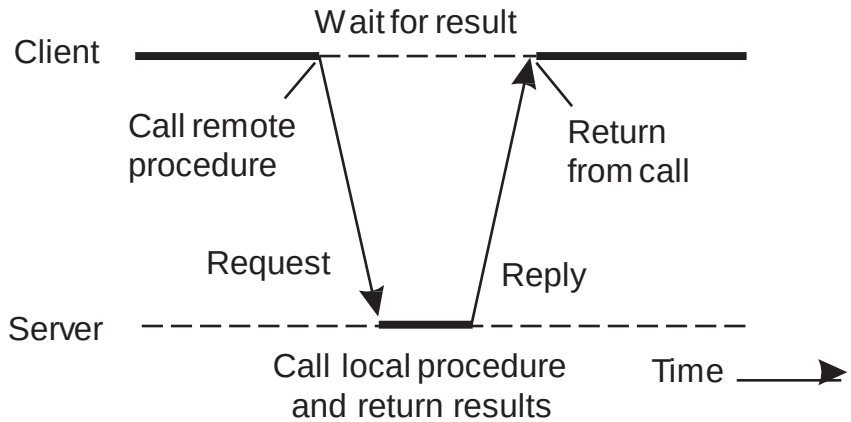
- Funcionamento básico de RPCs [Nelson 1984]
- Passagem de parâmetros
- Tipos de comunicação
- Exemplo

RPC: Funcionamento básico

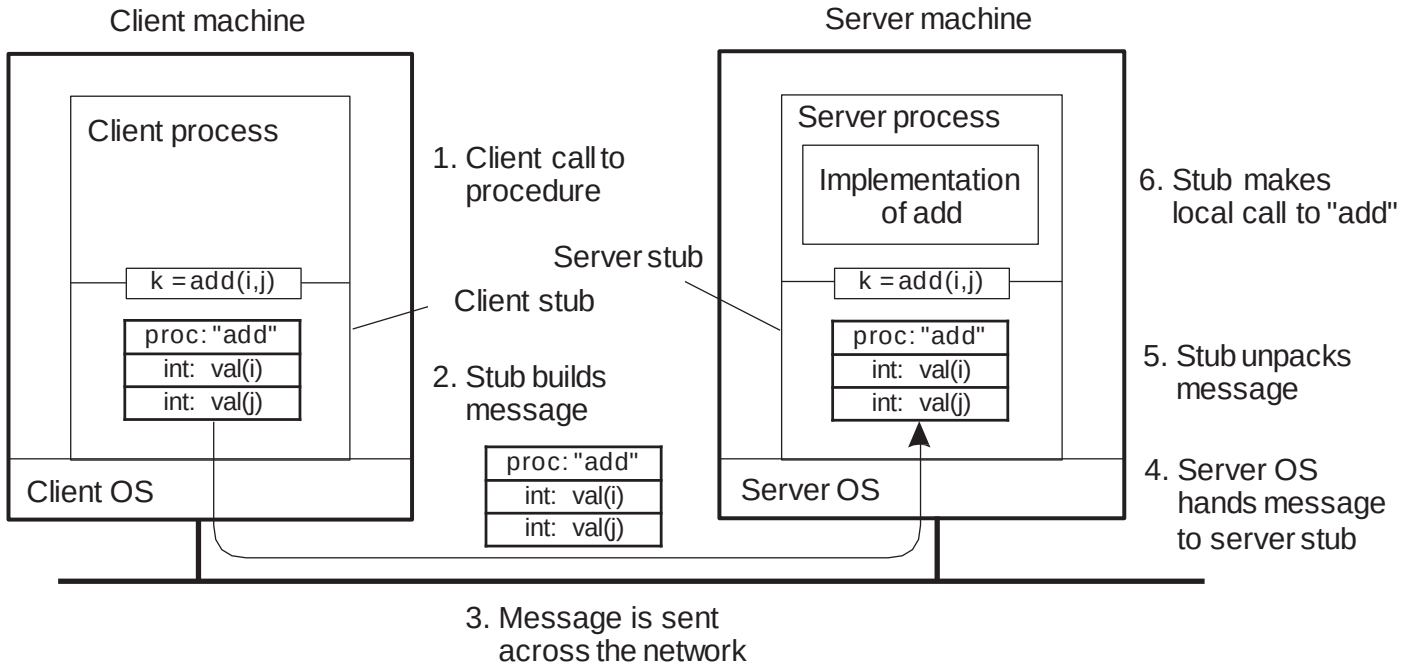
- Desenvolvedores estão familiarizados com o modelo de procedimentos
- Procedimentos bem projetados operam isoladamente (*black box*)
- Então não há razão para não executar esses procedimentos em máquinas separadas

Conclusão

Comunicação entre o chamador & chamado podem ser escondida com o uso de mecanismos de chamada a procedimentos.



RPC: Funcionamento básico



1. Procedimento no cliente chama o *stub* do cliente
2. *Stub* constrói mensagem; chama o SO local
3. SO envia msg. para o SO remoto
4. SO remoto repassa mensagem para o *stub*
5. *Stub* desempacota parâmetros e chama o servidor
6. Servidor realiza chamada local e devolve resultado para o *stub*
7. *Stub* constrói mensagem; chama SO (volta)
8. SO envia mensagem para o SO do cliente
9. SO do cliente repassa msg. para o *stub*
10. *Stub* do cliente desempacota resultado e devolve para o cliente

RPC: concorrência

Que acontece quando vários clientes acessam o método ?

Depende da implementação. Normalmente (e.g., Java RMI) utiliza-se o modelo dispatcher/worker.

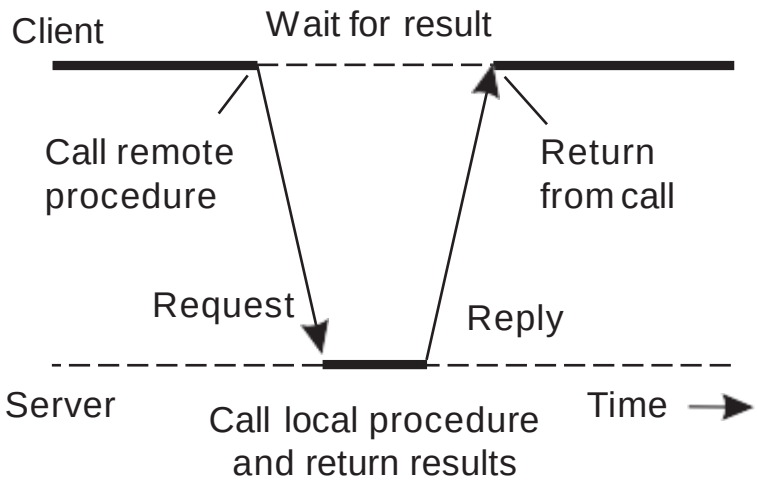
Ou seja, cada requisição do cliente é atendida por uma thread diferente.

Isso quer dizer que é necessário proteger as variáveis e estruturas de acessos concorrentes.

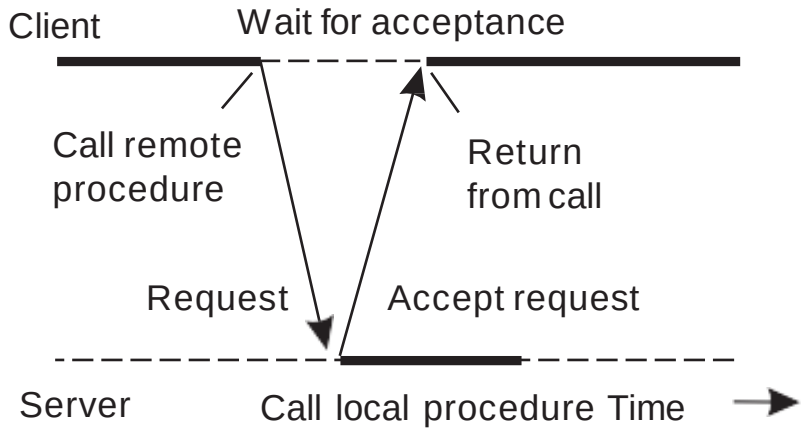
RPC: tipos de comunicação →assíncrono

Ideia geral

Tentar se livrar do comportamento estrito de requisição–resposta, mas permitir que o cliente continue sem esperar por uma resposta do servidor.

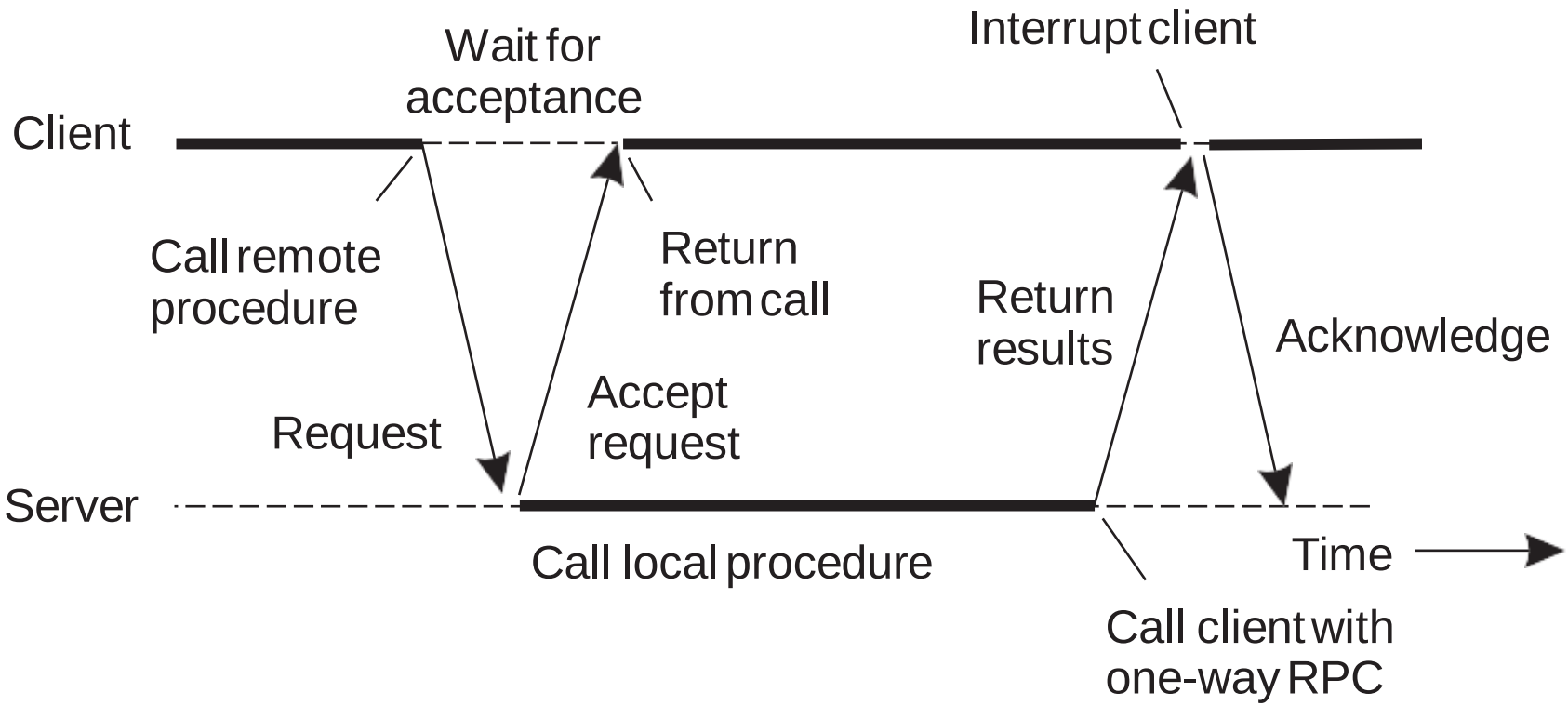


(a)



(b)

RPC: tipos de comunicação → síncrono diferido

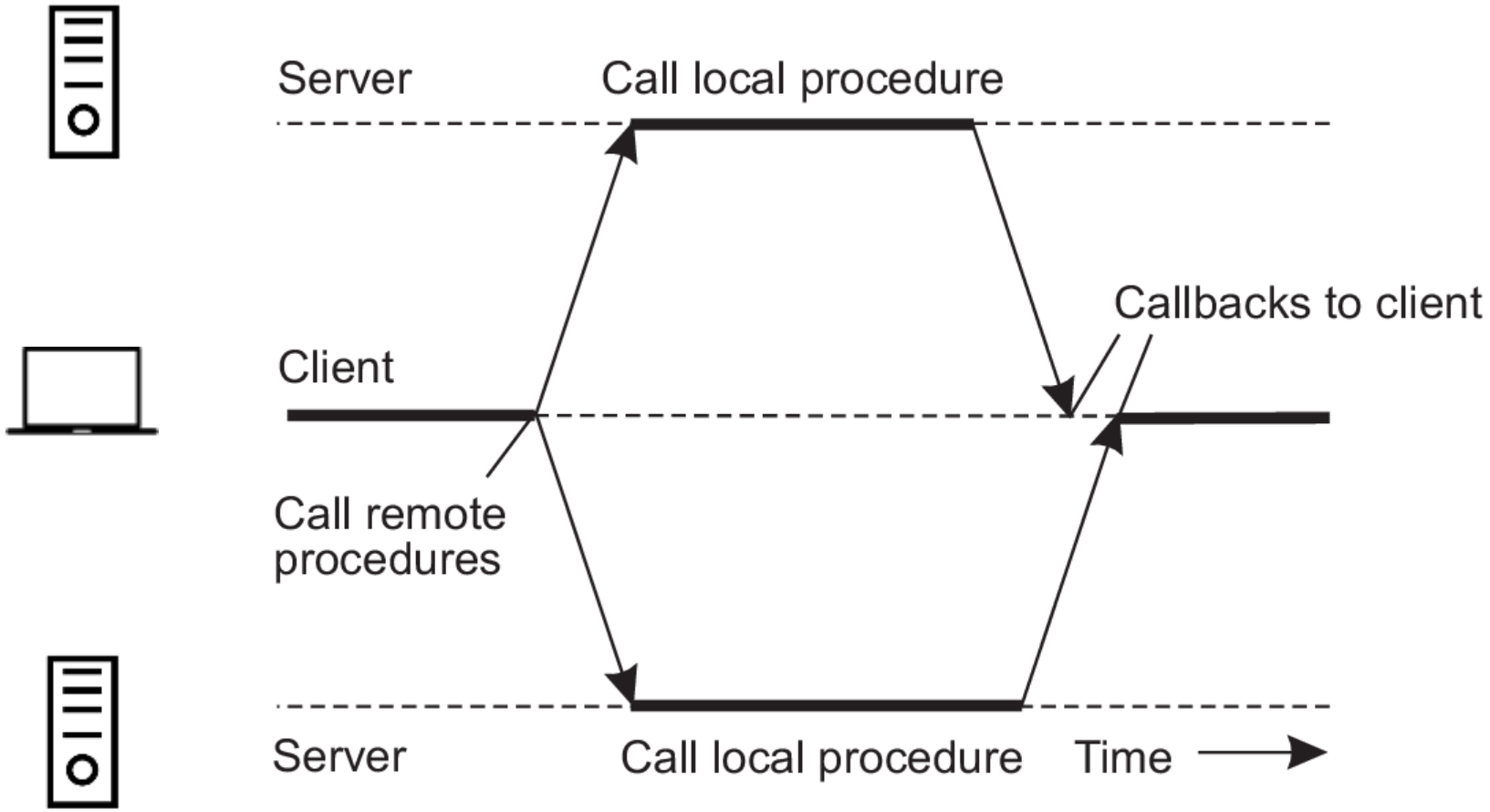


Variação

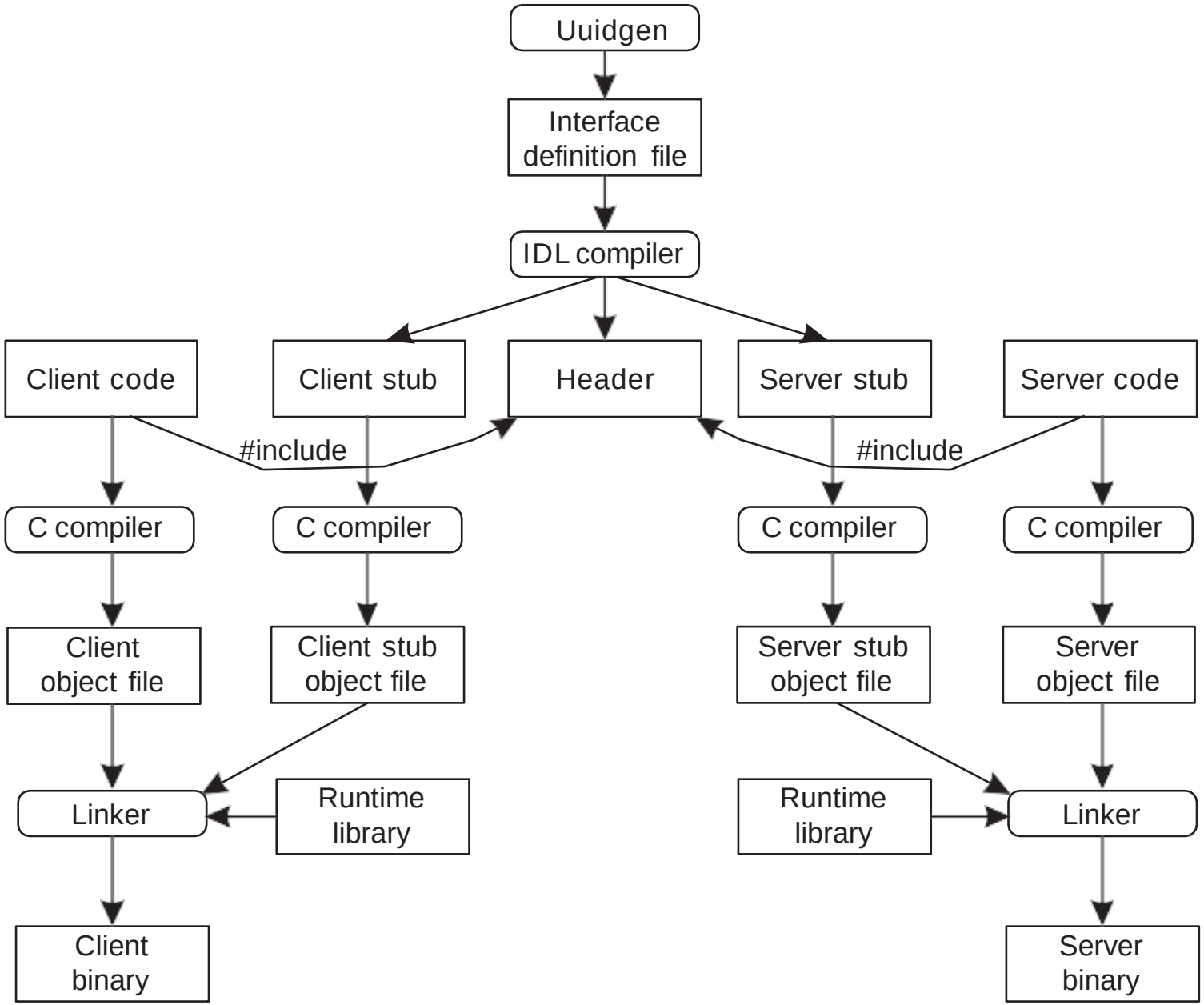
Cliente pode também realizar uma consulta (*poll*) (bloqueante ou não) para verificar se os resultados estão prontos.

RPC: tipos de comunicação → Enviando várias

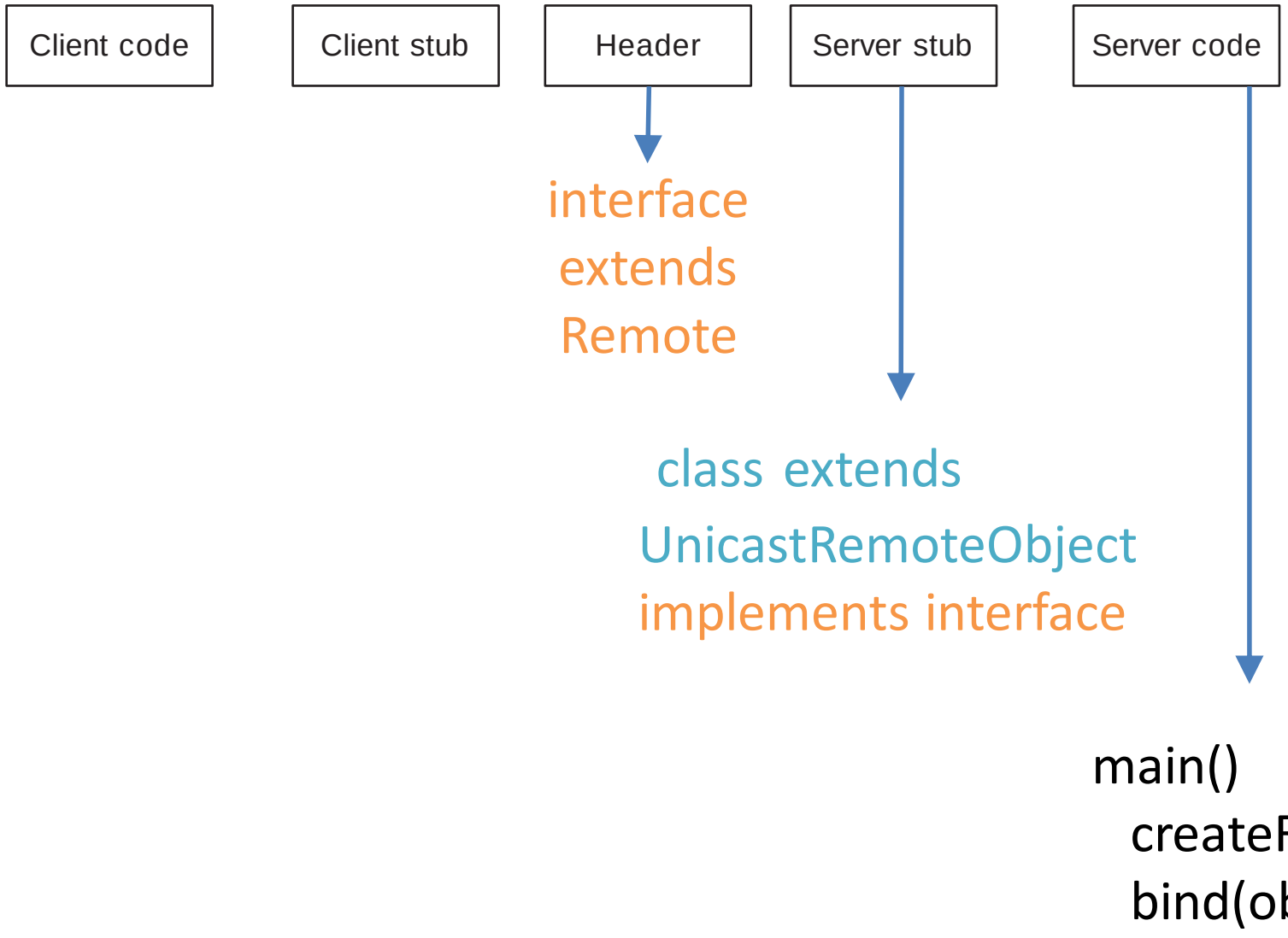
Enviando um pedido de RPC a um grupo de servidores



RPC: Exemplo


Uuidgen
Interface definition file
IDL compiler
Client code
Client stub
Header
Server stub
Server code
#include
#include
C compiler
C compiler
C compiler
C compiler
Client object file
Client stub object file
Server stub object file
Server object file
Linker
Runtime library
Runtime library
Linker
Client binary
Server binary

RPC: Exemplo (RMI – Remote Method Invocation)

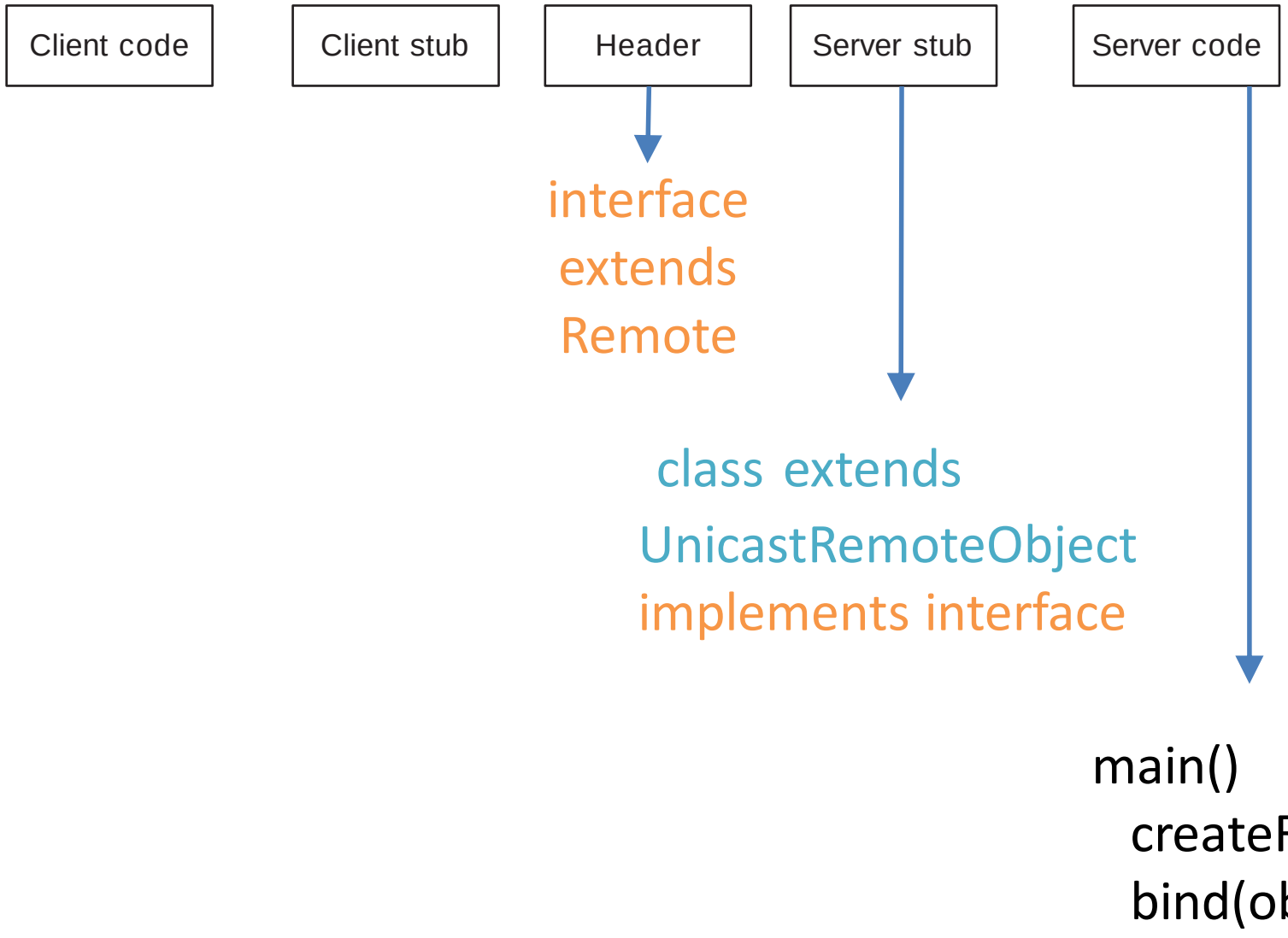


RPC: Exemplo (RMI – Remote Method Invocation)

```
public interface ServicoMatriz extends Remote {

  public int[][] inverte(int[][] matrizGigante)
                                    throws RemoteException;
}
```

RPC: Exemplo (RMI – Remote Method Invocation)



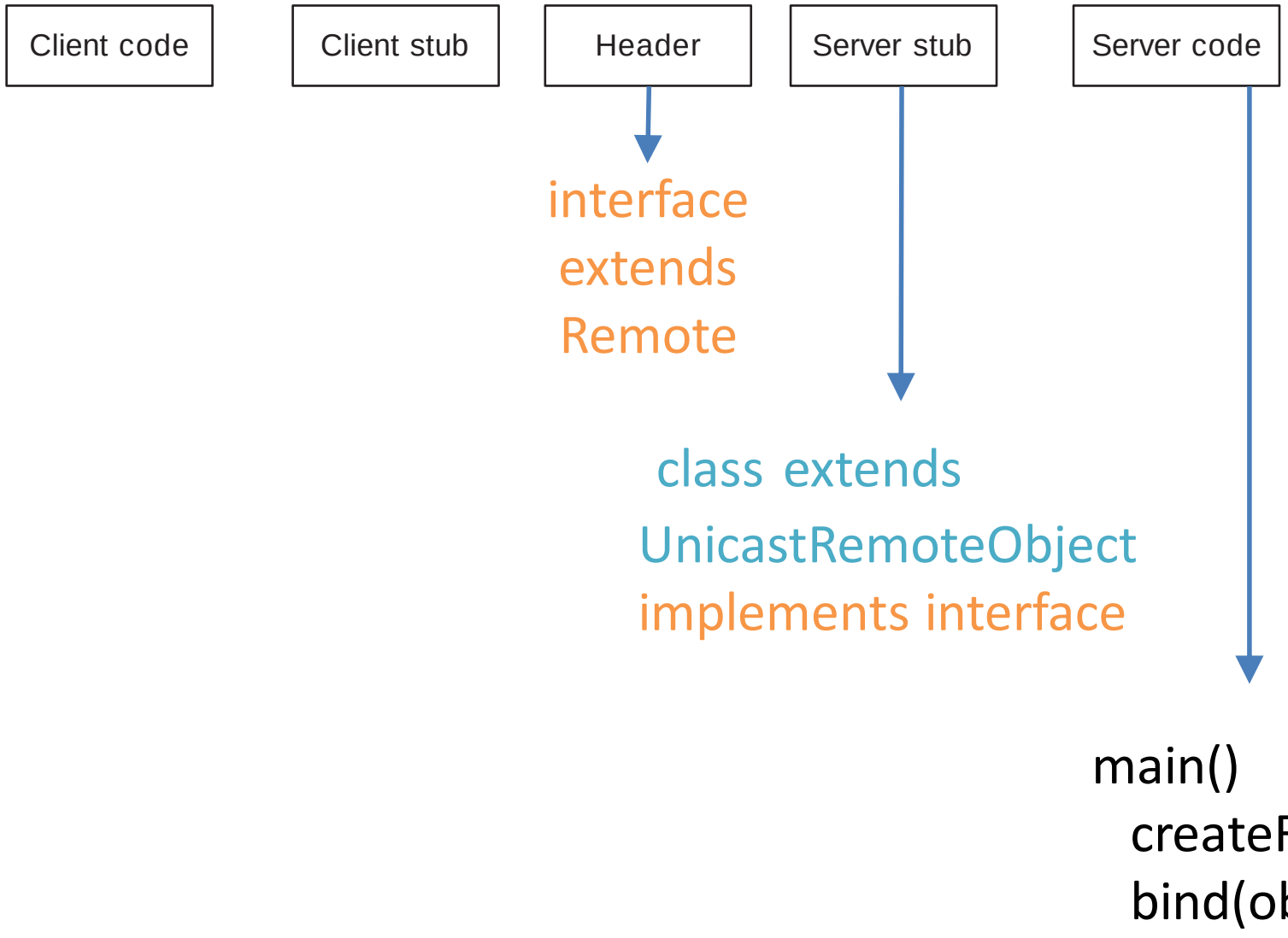
RPC: Exemplo (RMI – Remote Method Invocation)

```
import etml.*
public class ServicoImpl
                extends UnicastRemoteObject
                implements ServicoMatriz {

    public int[][] inverte(int[][] matrizGigante)
                                throws RemoteException {

        Matrix m = new Matrix(matrizGigante);
        return m.invert();
    }
}
```

RPC: Exemplo (RMI – Remote Method Invocation)



RPC: Exemplo (RMI – Remote Method Invocation)

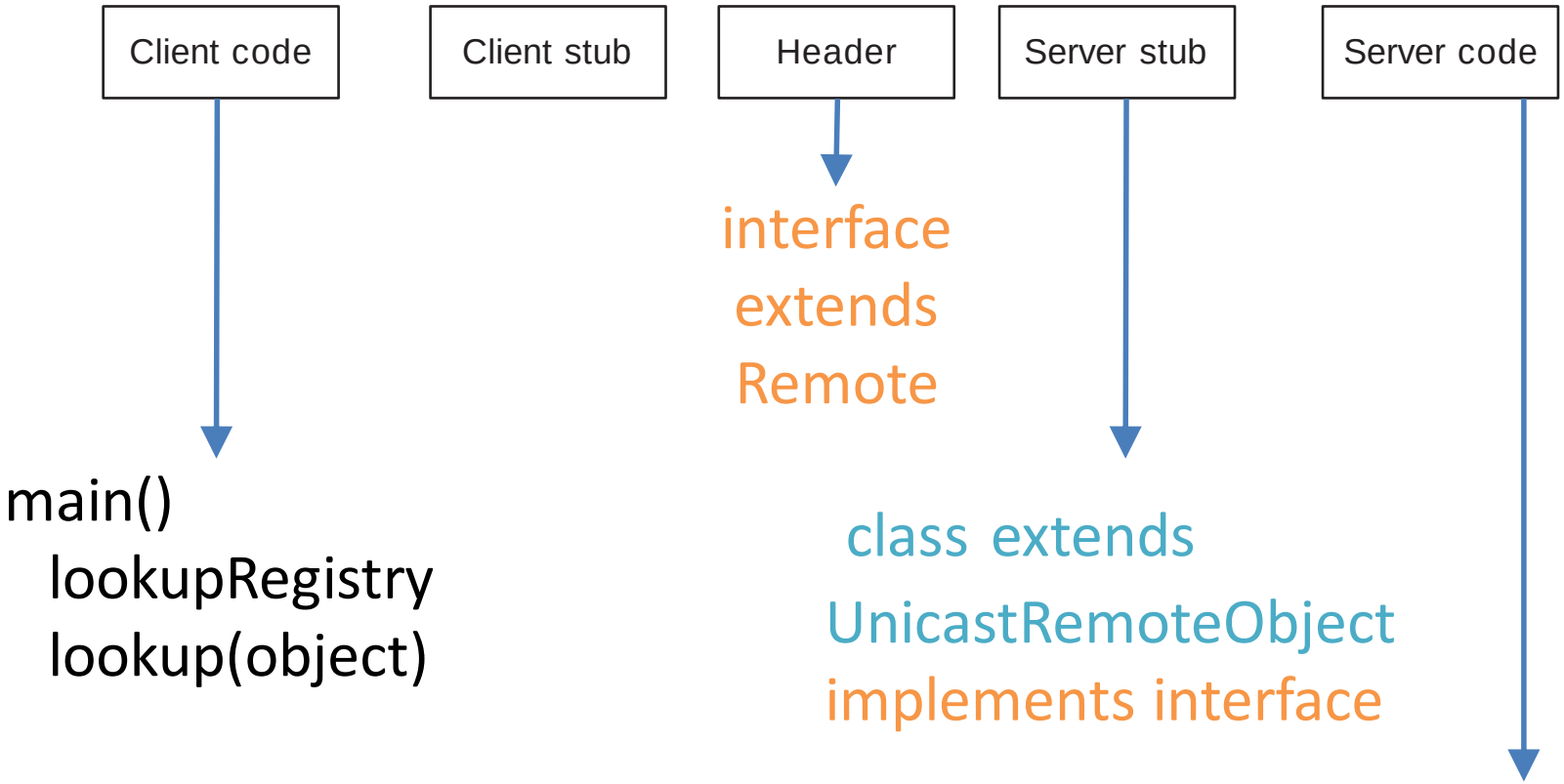
```
public class Servidor {

  public static void main(String[] args) {

     ServicoMatriz sm = new ServicoImpl();
     LocateRegistry.createRegistry(1099);

     Registry reg = LocateRegistry.getRegistry();
     reg.bind("rmi://127.0.0.1/servicomat", sm);
  }
}
```

RPC: Exemplo (RMI – Remote Method Invocation)



RPC: Exemplo (RMI – Remote Method Invocation)

```
public class Cliente {

    public static void main(String[] args) {
        Registry reg = LocateRegistry.getRegistry();
        ServicoMatriz sm = (ServicoMatriz)
        reg.lookup("rmi://127.0.0.1/servicomat");

        int [][] mat = {};
        int[][] inv = sm.inverte(mat);
    }
}
```

RPC: Exemplo (gRPC – Google RPC)

Google Remote Procedure Call

Desenvolvido pela Google em 2015.

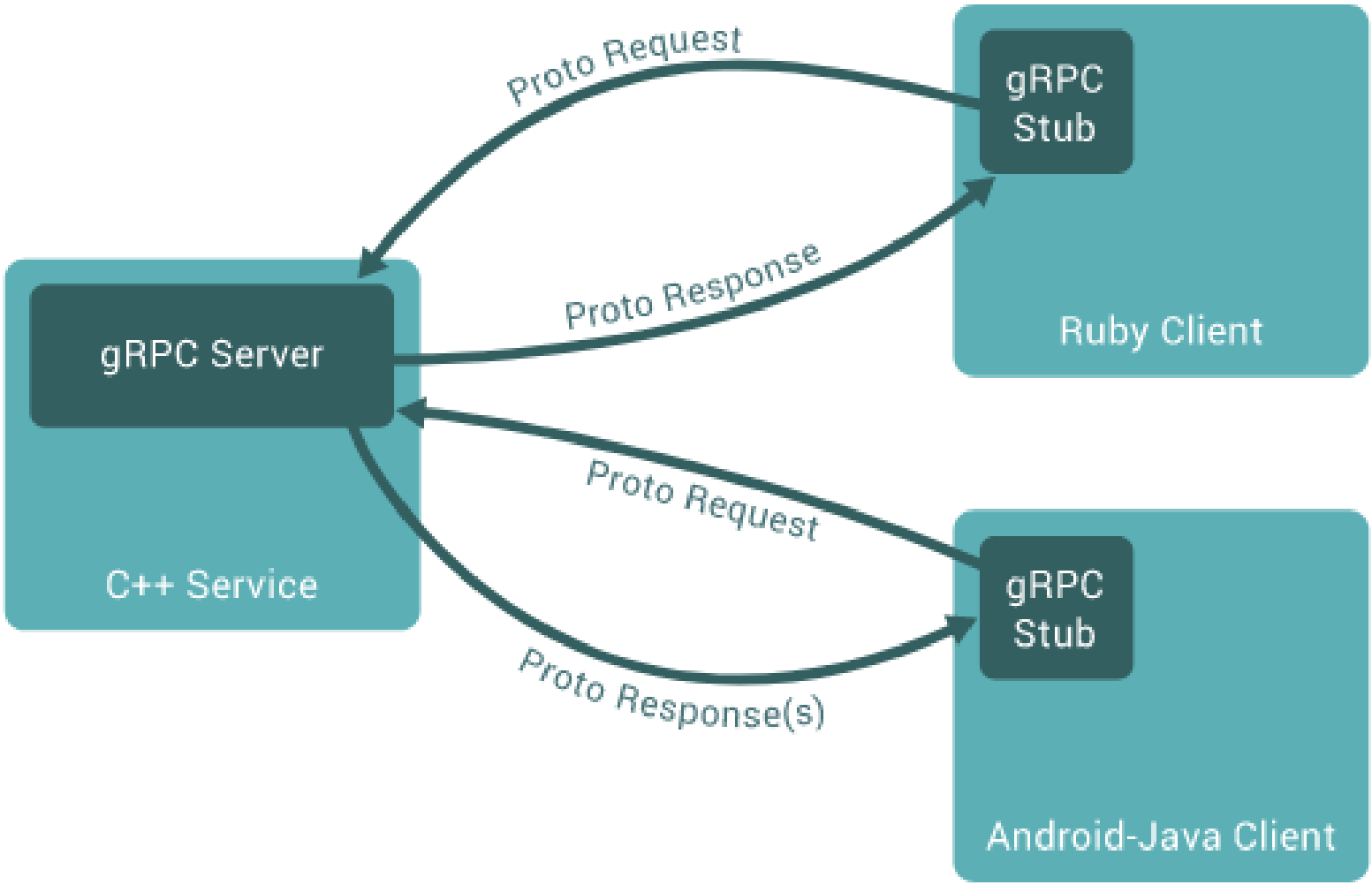
- Mensagens 3 a 10 dez vezes menores
- Enviadas 20 a 100 vezes mais rápido

Cenários comuns de uso:

- Quando precisar de baixa latência e alta escalabilidade.
- Clientes móveis comunicando-se com a nuvem
- Permitir facil extensão para outros componentes (auth, balanceamento, etc).

Usado por empresas tais como: Uber, Netflix, IBM, Dropbox, Cisco ...

RPC: Exemplo (gRPC – Google RPC)

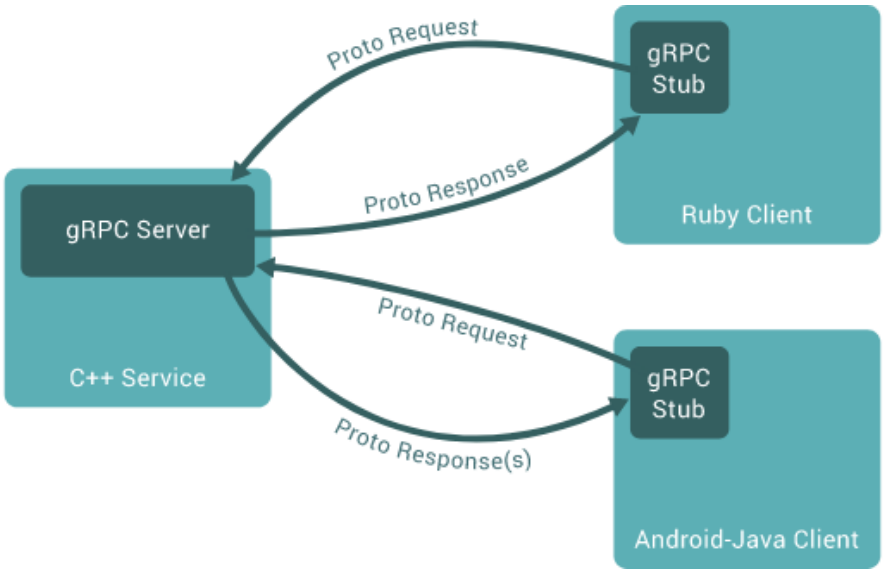


RPC: Exemplo (gRPC – Google RPC)

```
// The greeter service definition.
service Greeter {
    // Sends a greeting
    rpc SayHello (HelloRequest) returns (HelloReply) {}
}

// The request message containing the user's name.
message HelloRequest { string name = 1; }

// The response message containing the greetings
message HelloReply { string message = 1; }
```



Agenda

- Fundamentos
- Comunicação orientada a procedimento (RPC)
- Comunicação orientada a mensagem (MOM) *
- Comunicação multicast (Gossip / Flooding)
- Comunicação orientada a fluxo

* Usados como: event-streaming platform

Comunicação orientada a mensagens

- Middleware orientado a mensagens (filas)
- Message broker
- Exemplo: Apache Kafka

Middleware orientado a mensagens

Ideia geral

Comunicação assíncrona e persistente graças ao uso de filas gerenciadas pelo middleware. Filas correspondem a buffers em servidores de comunicação.

PUT	Adiciona uma mensagem à fila especificada
GET	Bloqueia até que a fila especificada tenha alguma mensagem e remove a primeira mensagem
POLL	Verifica se a fila especificada tem alguma mensagem. Nunca bloqueia
NOTIFY	Instala um tratador para ser chamado sempre que uma mensagem for inserida em uma dada fila

Message broker

Observação:

Sistemas de filas de mensagens assumem um protocolo comum de troca de mensagens: todas as aplicações usam o mesmo formato de mensagem (i.e., estrutura e representação de dados)

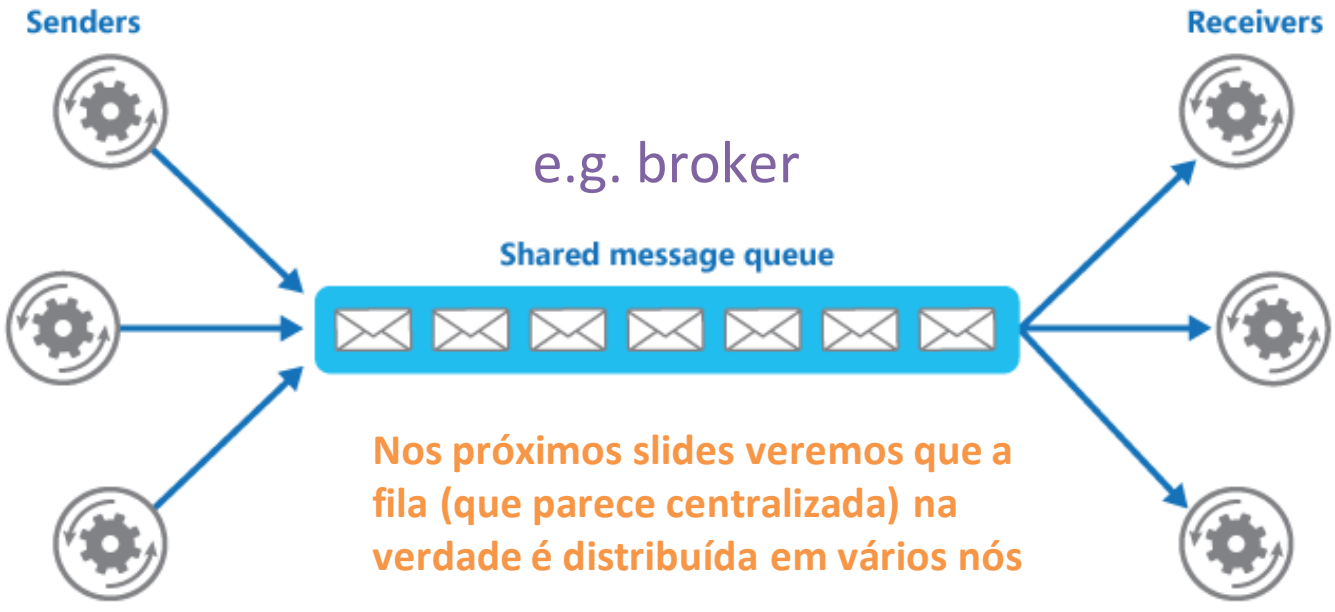
Message broker

Componente centralizado que lida com a heterogeneidade das aplicações:

- transforma as mensagens recebidas para o formato apropriado
- frequentemente funciona como um gerenciador e/ou armazenador da fila

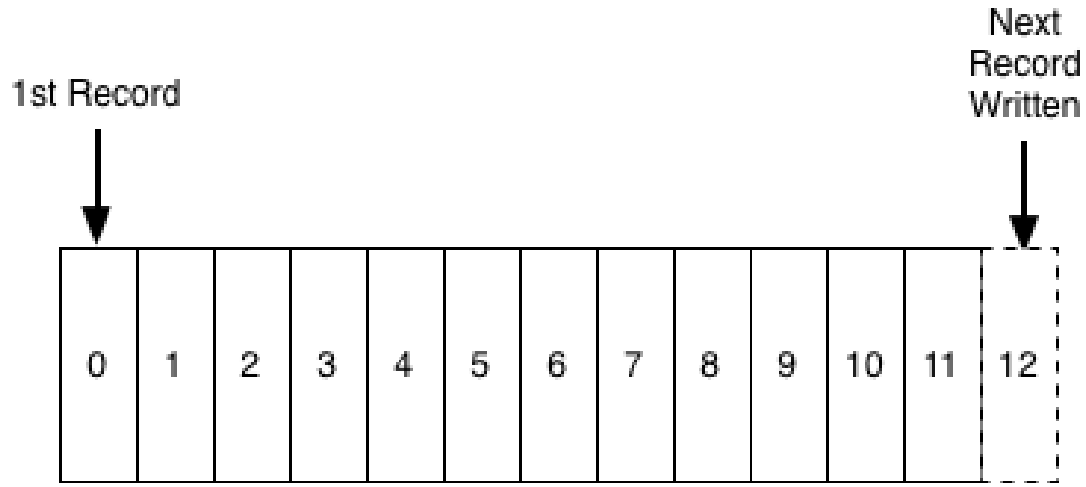
Message broker

- Mensagens específicas da aplicação são colocadas e removidas de filas
- As filas são controladas por um gerenciador de filas (e.g., um broker)
- Processos podem inserir mensagens em filas locais ou remotas, usando um mecanismo de RPC



Apache Kafka

- Exemplo de MOM, desenvolvido pelo LinkedIn [2012]
- Uma das plataformas de mensagerias open source mais usadas
- Utiliza o conceito de *event log* (i.e., estrutura append-only) onde cada registro possui um identificador único que captura a ordem dos eventos.



https://engineering.linkedin.com/distributed-systems/log-what-every-software-engineer-should-know-about-real-time-datas-unifying

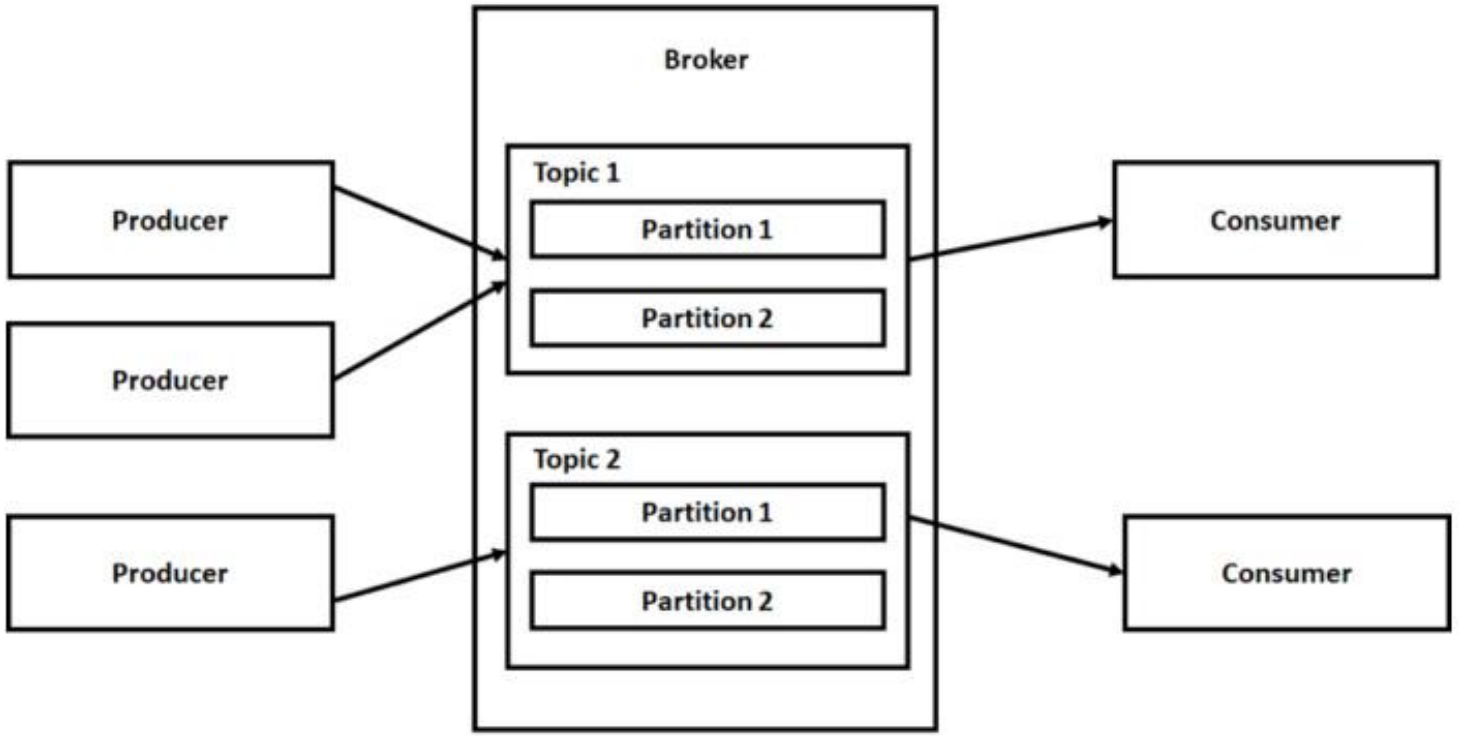
Apache Kafka

- Os logs (i.e., filas) são administrados/persistidos pelo Kafka broker

  O **kafka broker é stateless para requisições de clientes**
- O produtor envia os eventos ao broker de forma assíncrona
- Os eventos são armazenados de forma persistente no log.
- Podem ser lidos por vários consumidores.

  Cada consumidor especifica o log e o índice (offset).

No Kafka, o log é denominado de *Topic*



Apache Kafka

- Para a **escalabilidade**, kafka usa o particionamento (sharding) do tópico
  - Na criação você define a quantidade de partições
  - Permite a escalabilidade horizontal
  - Permite que produtores e consumidores paralelizem as reqs.

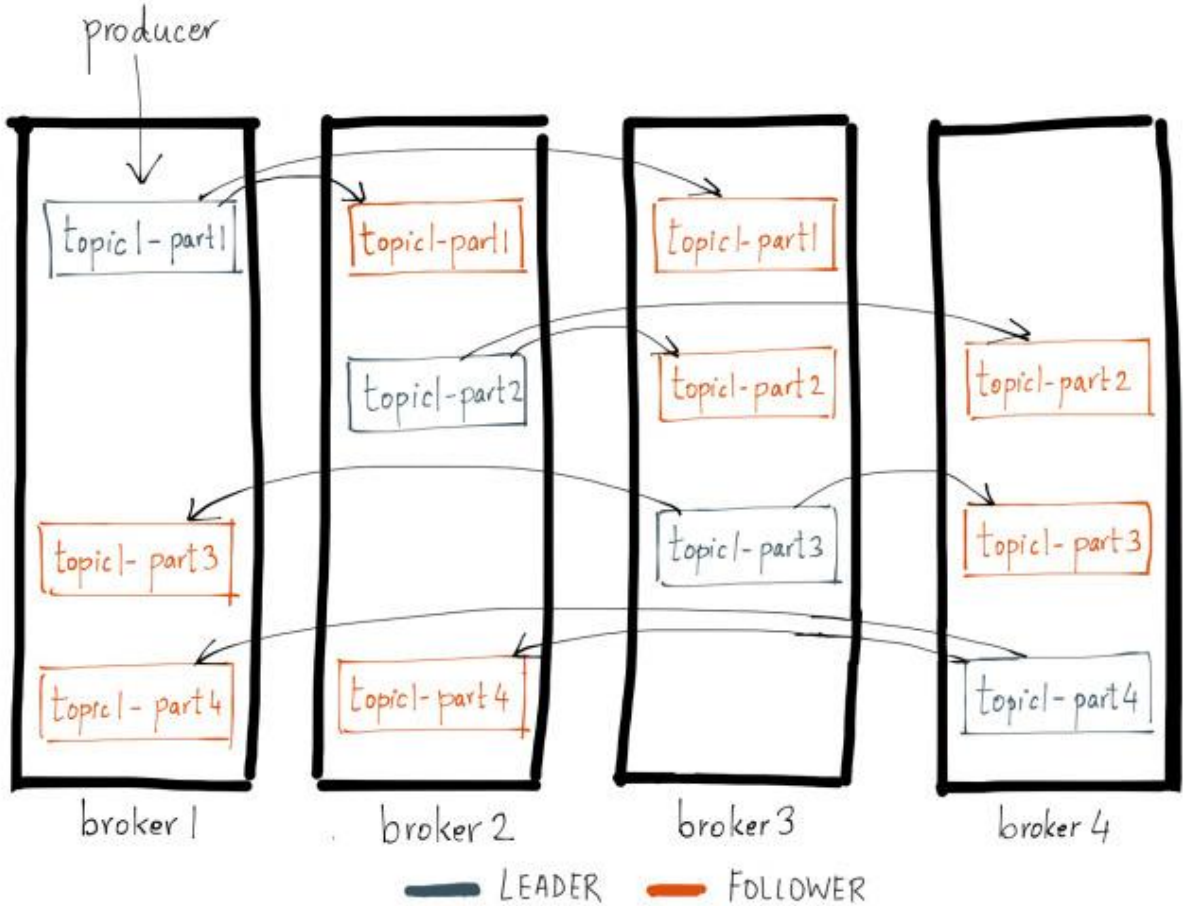
- Como o broker é stateless, cabe ao produtor escolher a partição onde será registrado o evento.

  E.g., Mensagens-SD-DA1, Mensagens-SD-DA2, ...

- Normalmente usa-se o hash de alguma chave (como id) que direciona à partição.

Apache Kafka

- Para a **disponibilidade**, kafka usa a replicação do tópico
  - Um líder, responsável por atender leituras e escritas.
  - N seguidores, responsáveis por resiliência a falhas do líder. (não aceitam leituras e/ou escritas)



https://www.confluent.io/blog/hands-free-kafka-replication-a-lesson-in-operational-simplicity/

Apache Kafka

slack engineering

- Slack (discord-like) usa kafka
  - Captura eventos de clientes Web que demandarão muito tempo para serem processados.
  - Em 2018, processou mais de 1 bilhão de mensagens por dia usando 16 brokers na AWS

    Instância i3.2xlarge: 8 vCPU, 61 GB RAM.
    Partições por tópico 32.
    Fator de replicação 3.

Apache ActiveMQ

O ActiveMQ é um dos mais populares provedores open source (licence Apache 2.0), utilizado para integrar diferentes linguagens, tais como: Java, .NET, C/C++/C#, Delphi, Perl, Python, entre outras, através do chamado "Cross Language Clients".

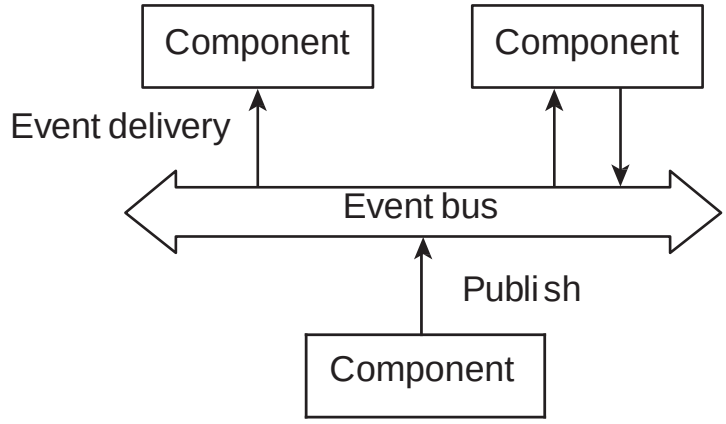
Desde 2004 até hoje

http://activemq.apache.org/

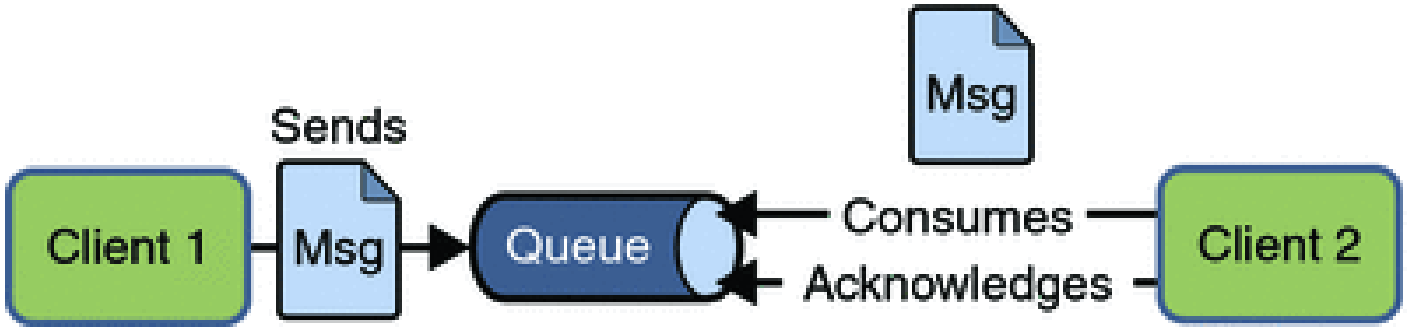
Apache ActiveMQ: modelos

- Modelo ponto a ponto
- Modelo publish/subscribe

Apache ActiveMQ: modelos

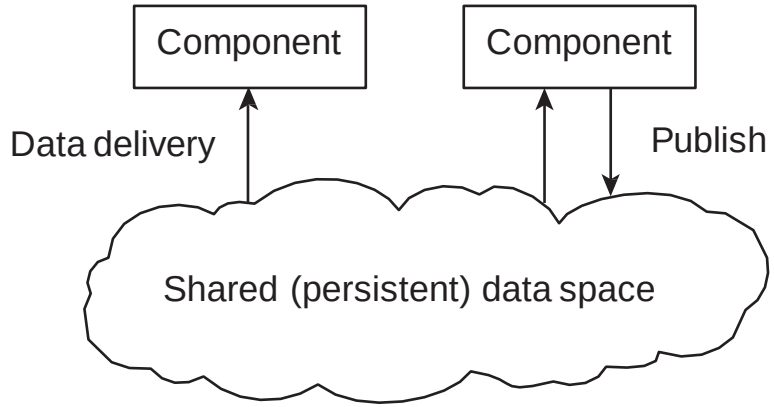


- Modelo ponto a ponto (queue)

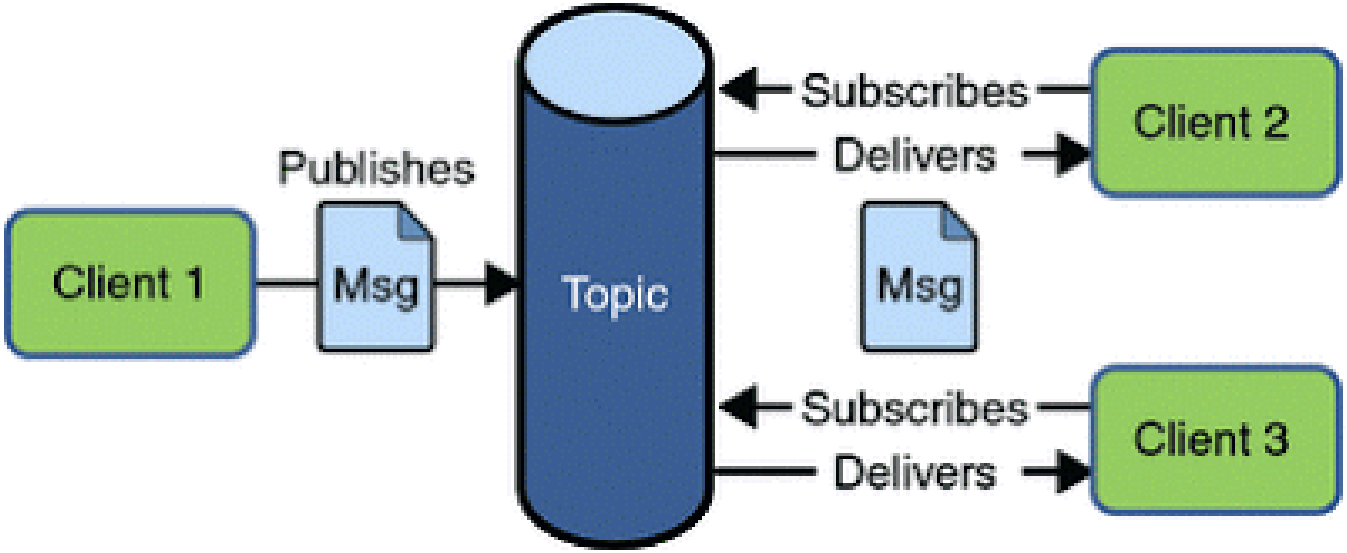


1. O produtor envia uma mensagem para a queue
2. Somente um dos consumidores (que estão escutando) recebe a mensagem

Apache ActiveMQ: modelos



- Espaço de dados compartilhados



1. O produtor envia uma mensagem para o servidor de tópicos
2. Todos os consumidores (que estão escutando) recebem a msg

Vantagens e Desvantagens

- Dentro da Aplicação

  Fácil de usar – Java ExecutorService

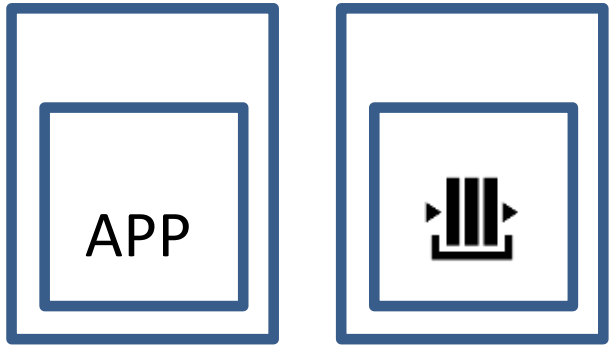
  Normalmente armazenamento em memória

  (perda de mensagens/tarefas se cair a app)

- Fora da Aplicação (separada)

  Armazenamento em disco – Kafka

  Escalabilidade limitada à máquina.

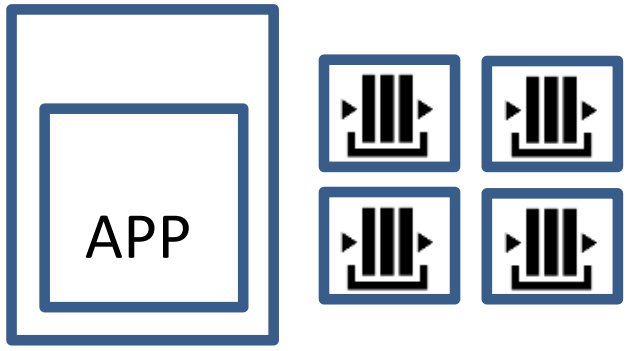


- Fora da Aplicação (distribuída)

  Armazenamento em disco, escalabilidade

  muito grande – Kafka/RabbitMQ/ApacheMQ

  Complexo de administrar, inconsistências



Agenda

- Fundamentos
- Comunicação orientada a procedimento (RPC)
- Comunicação orientada a mensagem (MOM)
- Comunicação multicast
  - FIFO/Causal/Total e sem ordem.
  - Gossip/Flooding
- Comunicação orientada a fluxo

Multicast

- Diferente do unicast (e.g., TCP) no multicast o emissor - sender - envia a mesma mensagem para um grupo de destinos - receivers.
- O multicast pode ser realizado pela camada de rede (difícil de ser implementado) ou pelo middleware.
- Pelo middleware, caberá a este verificar que as mensagens sejam entregues aos destinos

Agenda

- Fundamentos
- Comunicação orientada a procedimento (RPC)
- Comunicação orientada a mensagem (MOM)
- Comunicação multicast
  - FIFO/Causal/Total e sem ordem
  - Gossip/Flooding
- Comunicação orientada a fluxo

Multicast ordem FIFO

Se um processo *p* envia mensagens *m1* e depois *m2*, todos os processos entregarão primeiro *m1* antes de *m2*.

No middleware

Receber: não há controle sobre o que se recebe pela camada de rede.
Entregar: após receber a mensagem, você controla o que fazer com ela
(e.g., entregar para o banco de dados fazer o update)

Entregar ou repassar, ambos termos são usados

Multicast ordem FIFO

Se um processo *p* envia mensagens *m1* e depois *m2*, todos os processos entregarão primeiro *m1* antes de *m2*.

tempo	P1	P2
1	Envia/Entrega m1	Entrega m1
2	Envia/Entrega m2	Entrega m2

tempo	P1	P2	P3
1	Envia/Entrega m1	Entrega m1	Entrega m2
2	Envia/Entrega m2	Entrega m2	Entrega m1

tempo	P1	P2	P3	P4
1	Envia/Entrega m1	Entrega m1	Entrega m3	Envia/Entrega m3
2	Envia/Entrega m2	Entrega m3	Entrega m1	Envia/Entrega m4
3	Entrega m3	Entrega m2	Entrega m2	Entrega m1
4	Entrega m4	Entrega m4	Entrega m4	Entrega m2

Multicast ordem CAUSAL

Se uma mensagem *m1* precede causalmente *m2*, todos os processos entregarão primeiro *m1* antes de *m2*.

No capítulo 6 será definido formalmente o termo causalidade.
Note que não depende do processo origem que enviou o multicast.

Multicast ordem CAUSAL

Implementação no capítulo 6

Se uma mensagem *m1* precede causalmente *m2*, todos os processos entregarão primeiro *m1* antes de *m2*.

tempo	P1	P2	P3
1	Envia m1		
2		Entrega m1	
3	Entrega m1	Envia m2	Entrega m1
4	Entrega m2		
5		Entrega m2	Entrega m2

tempo	P1	P2	P3
1	Envia m1		
2		Entrega m1	
3	Entrega m1	Envia m2	Entrega m2
4	Entrega m2		
5		Entrega m2	Entrega m1

Multicast ordem TOTAL

Implementação nos capítulos 6 e 8

Se um processo entrega *m1* antes de *m2*, todos os processos entregarão nessa mesma ordem.

tempo	P1	P2	P3	P4
1	Envia m1			Envia m2
2		Entrega m2	Entrega m2	
3	Entrega m2	Entrega m1	Entrega m1	Entrega m2
4	Entrega m1			Entrega m1

tempo	P1	P2	P3	P4
1	Envia m1			Envia m2
2		Entrega m2	Entrega m2	
3	Entrega m1	Entrega m1	Entrega m1	Entrega m2
4	Entrega m2			Entrega m1

Agenda

- Fundamentos
- Comunicação orientada a procedimento (RPC)
- Comunicação orientada a mensagem (MOM)
- Comunicação multicast
  - FIFO/Causal/Total
  - Gossip/Flooding
- Comunicação orientada a fluxo

Disseminação de dados via Gossip / Flooding

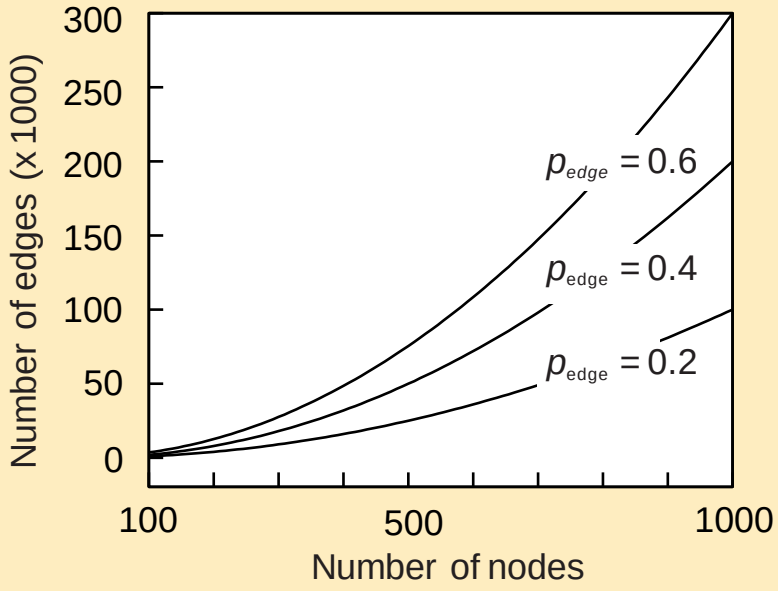
Ideia

Nó *P* envia uma mensagem *m* a seus vizinhos (via **gossip**). Cada vizinho a re-envia (via **flooding**) se não a viu antes.

Desempenho

Maior a quantidade de conexões, mais caro fica

Tamanho de uma overlay aleatória vs o nº de nós



Gossip: Protocolos Epidêmicos

Assuma não há conflito escrita-escrita

- Operações de atualização são realizadas em um só servidor
- Um nó passa informações (estado) a somente alguns nós
- Propagação de atualizações é lenta, i.e., não imediata
  Eventualmente, cada atualização atingirá cada nó

Duas formas de epidemia

Anti-entropy: Cada nó regularmente escolhe outro nó aleatoriamente e intercambia seu estado, levando a estados idênticos

Rumor: Um nó que foi atualizado (i.e., foi contaminado), avisa a outros nó sobre a atualização (contaminando outros).

O que é gossiping?

Node n

Nó, Thread ou Processo

```
n.run():
  while true:
    sleepMs(x)
    executeGossip()
```

Método main executado por Node *n*

Cada certo tempo acorda e faz algo (gossip)

```
n.executeGossip():
  p = selectNode()
  infoLocal = getLocalInfo()
  p.shareInfo(infoLocal, n)
```

Via RPC ao Nó *p*

```
n.shareInfo(info, q):
  addInfo(info)
  infoLocal = getLocalInfo()
  q.updateInfo(infoLocal)
```

Orientado a evento (muitos Nós podem chamá-lo ao mesmo tempo)

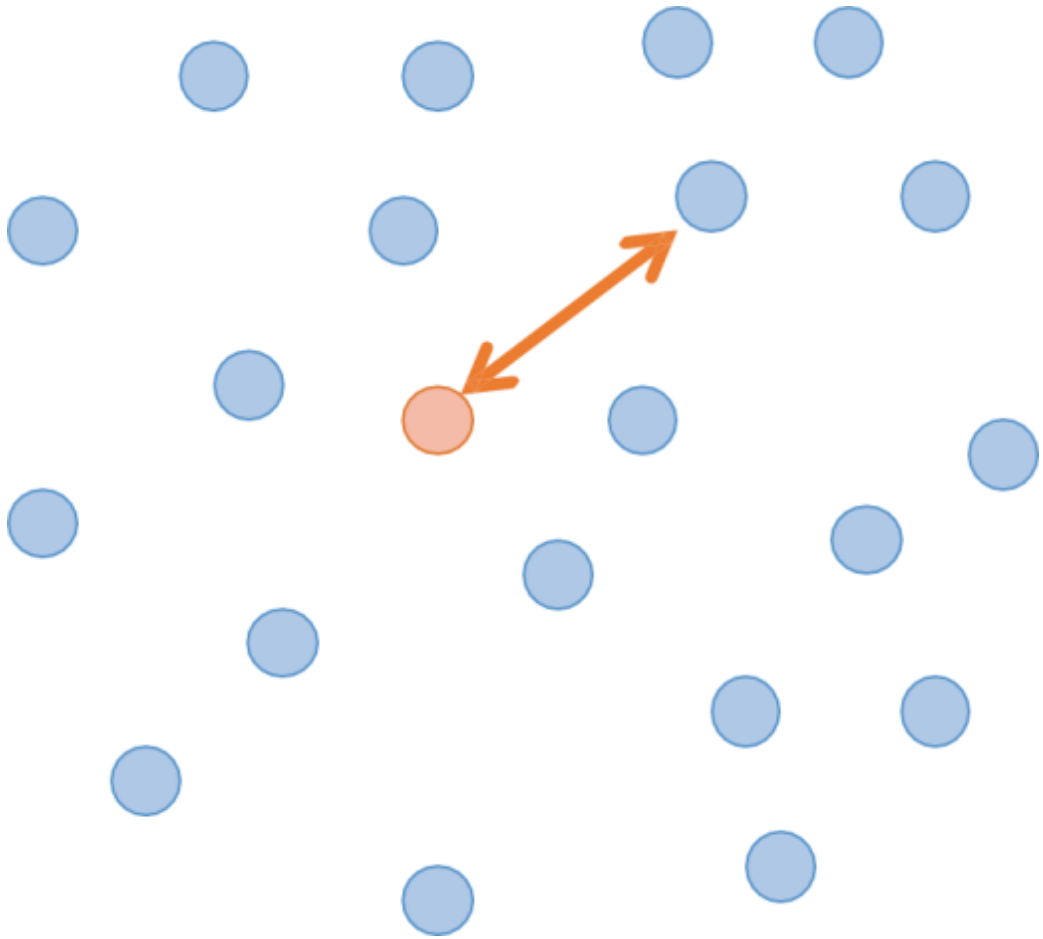
Via RPC ao Nó *q*

```
n.updateInfo(info):
  addInfo(info)
```

Orientado a evento (muitos Nós podem chamá-lo ao mesmo tempo)

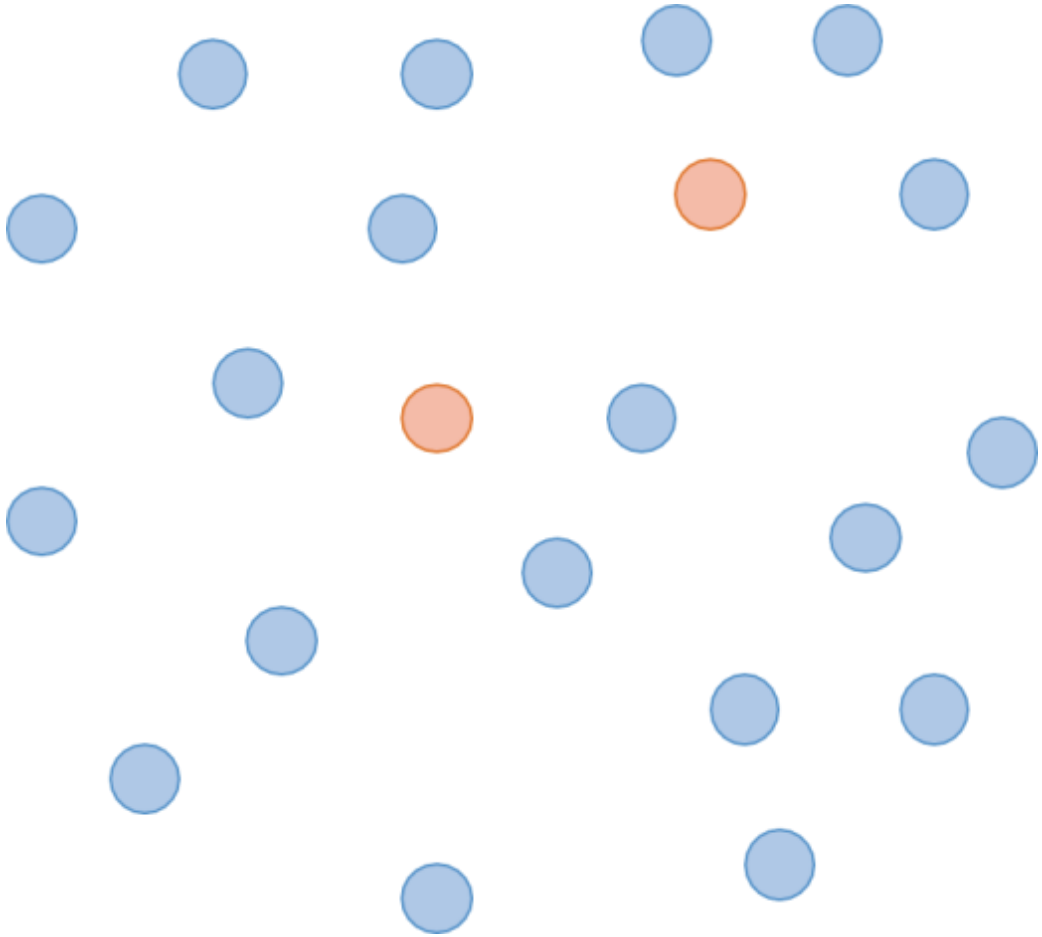
O que é gossiping?

Exemplo de Eugene Bagdasaryan
http://www.cs.cornell.edu/courses/cs6410/2016fa/slides/19-p2p-gossip.pdf



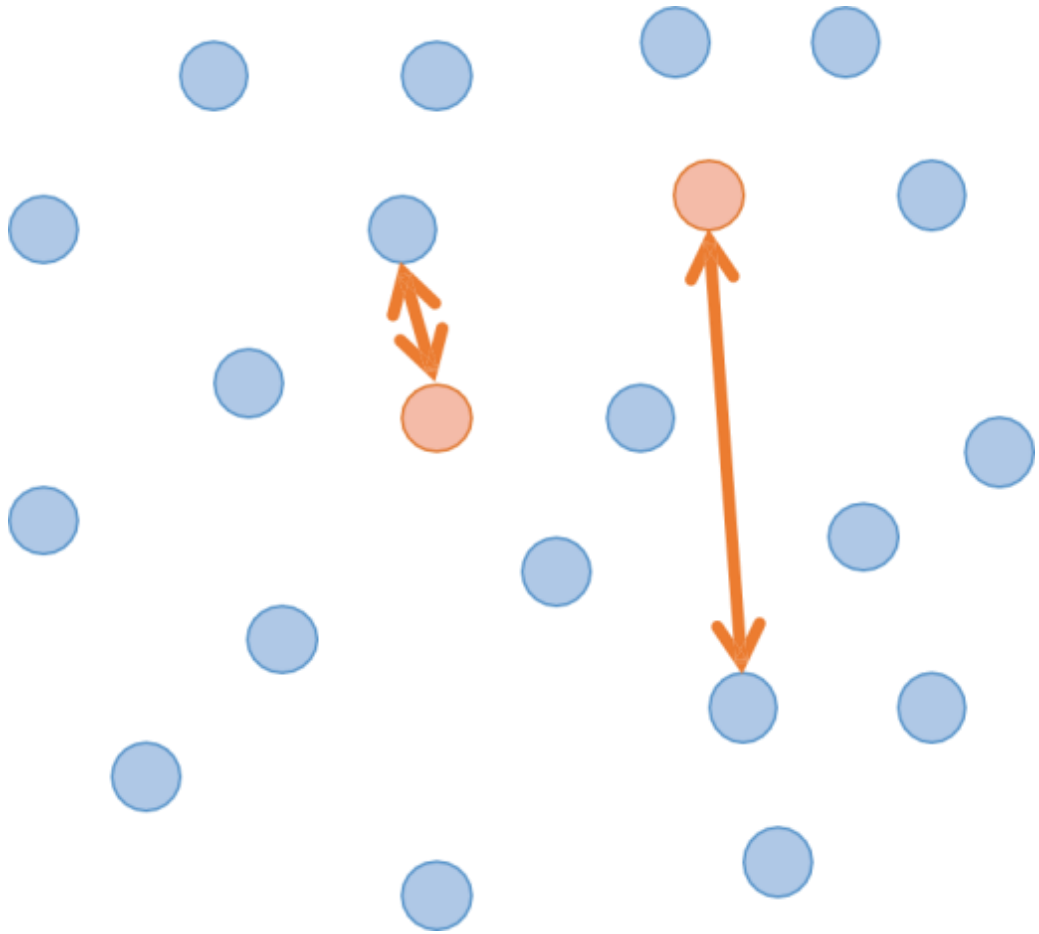
O que é gossiping?

Exemplo de Eugene Bagdasaryan
http://www.cs.cornell.edu/courses/cs6410/2016fa/slides/19-p2p-gossip.pdf



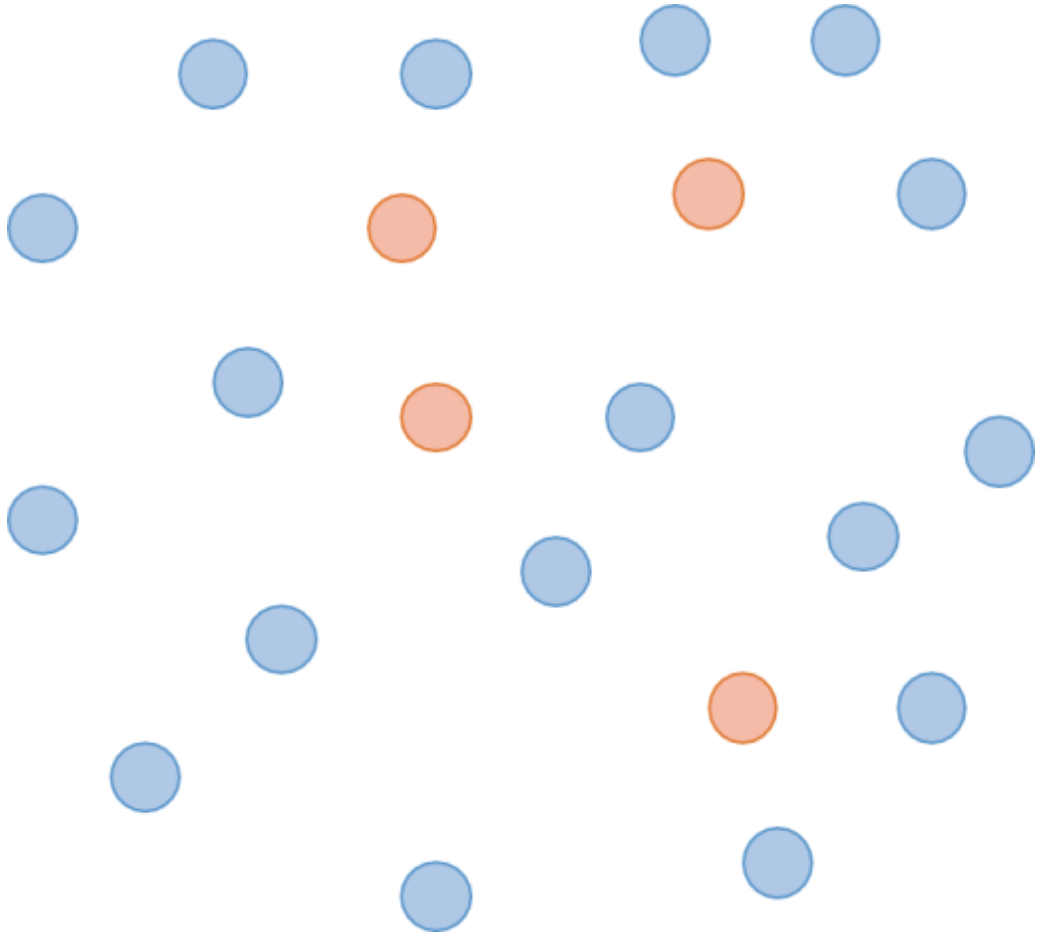
O que é gossiping?

Exemplo de Eugene Bagdasaryan
http://www.cs.cornell.edu/courses/cs6410/2016fa/slides/19-p2p-gossip.pdf



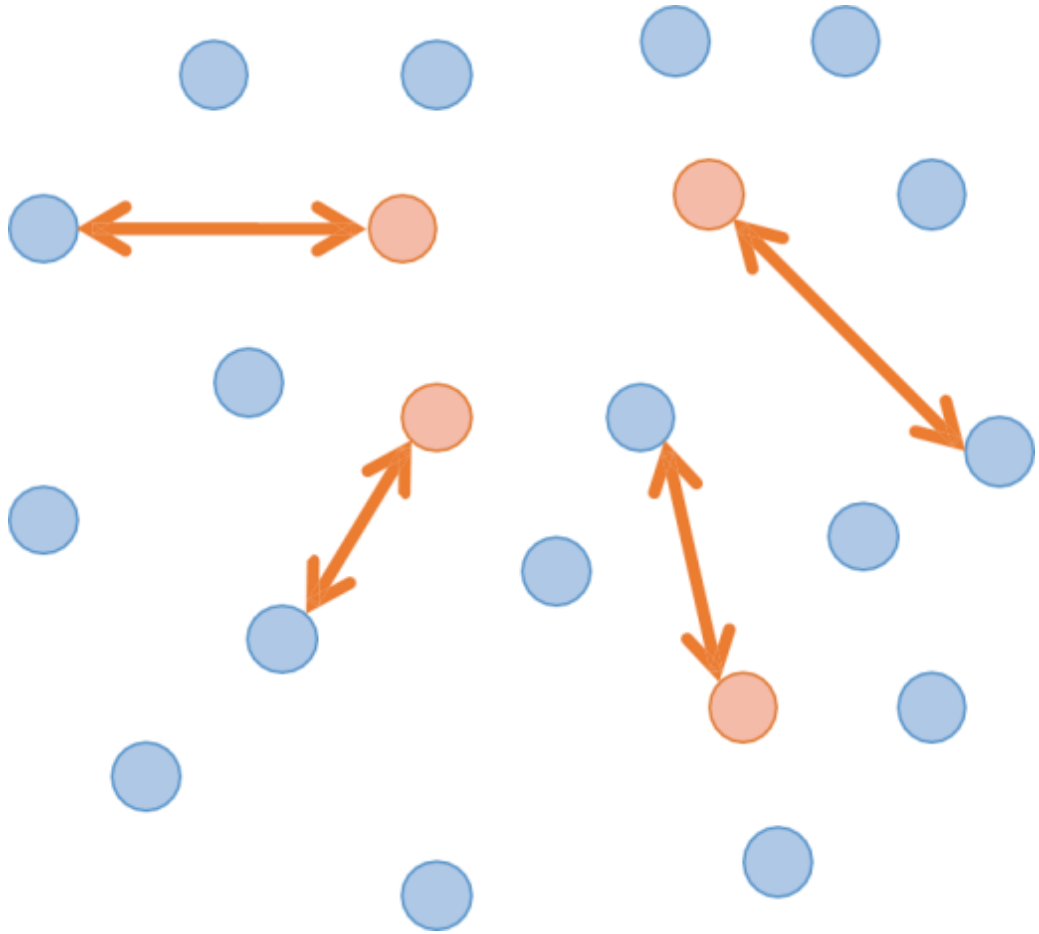
O que é gossiping?

Exemplo de Eugene Bagdasaryan
http://www.cs.cornell.edu/courses/cs6410/2016fa/slides/19-p2p-gossip.pdf



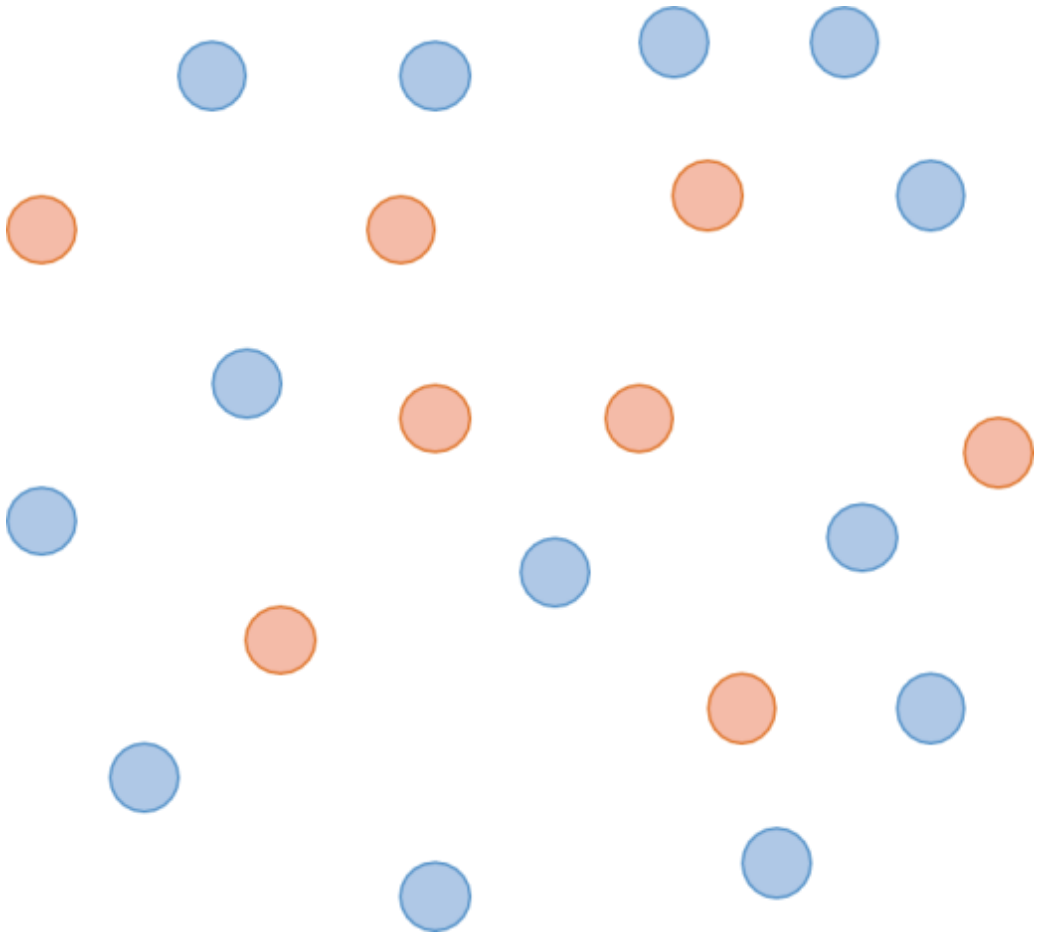
O que é gossiping?

Exemplo de Eugene Bagdasaryan
http://www.cs.cornell.edu/courses/cs6410/2016fa/slides/19-p2p-gossip.pdf



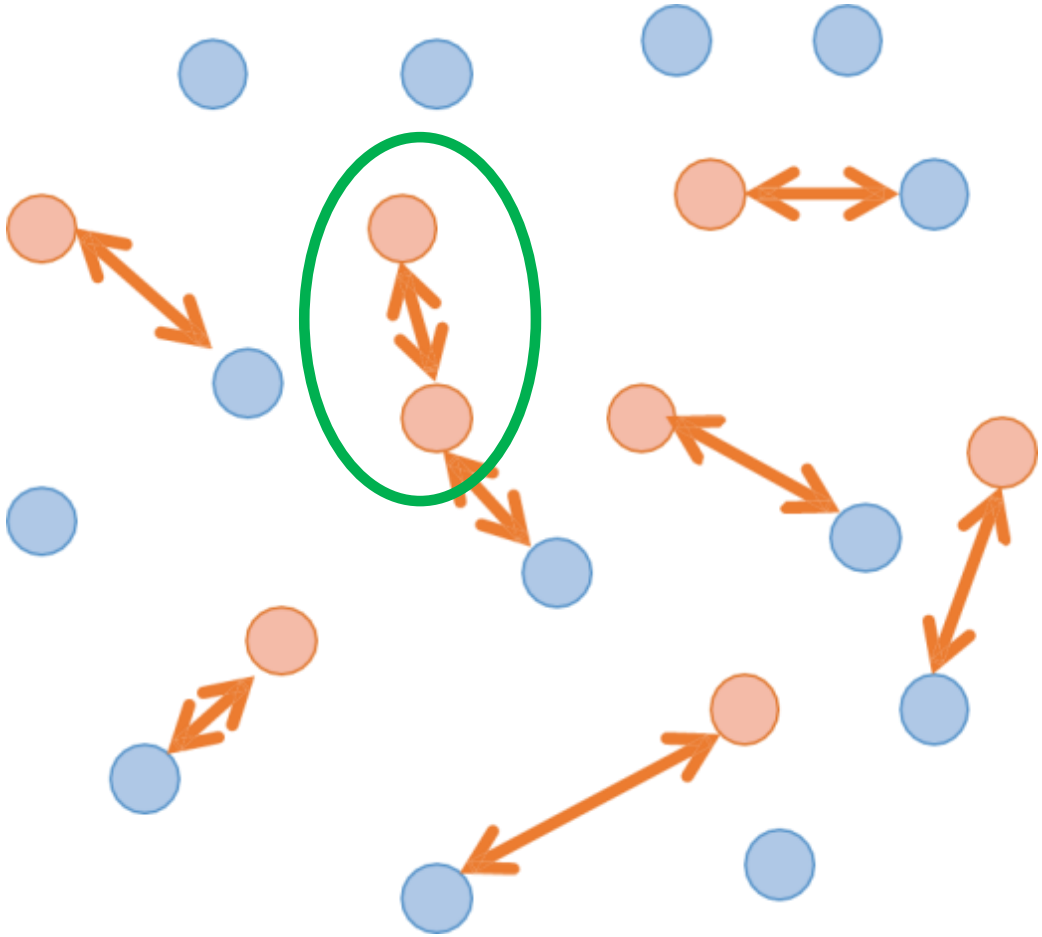
O que é gossiping?

Exemplo de Eugene Bagdasaryan
http://www.cs.cornell.edu/courses/cs6410/2016fa/slides/19-p2p-gossip.pdf



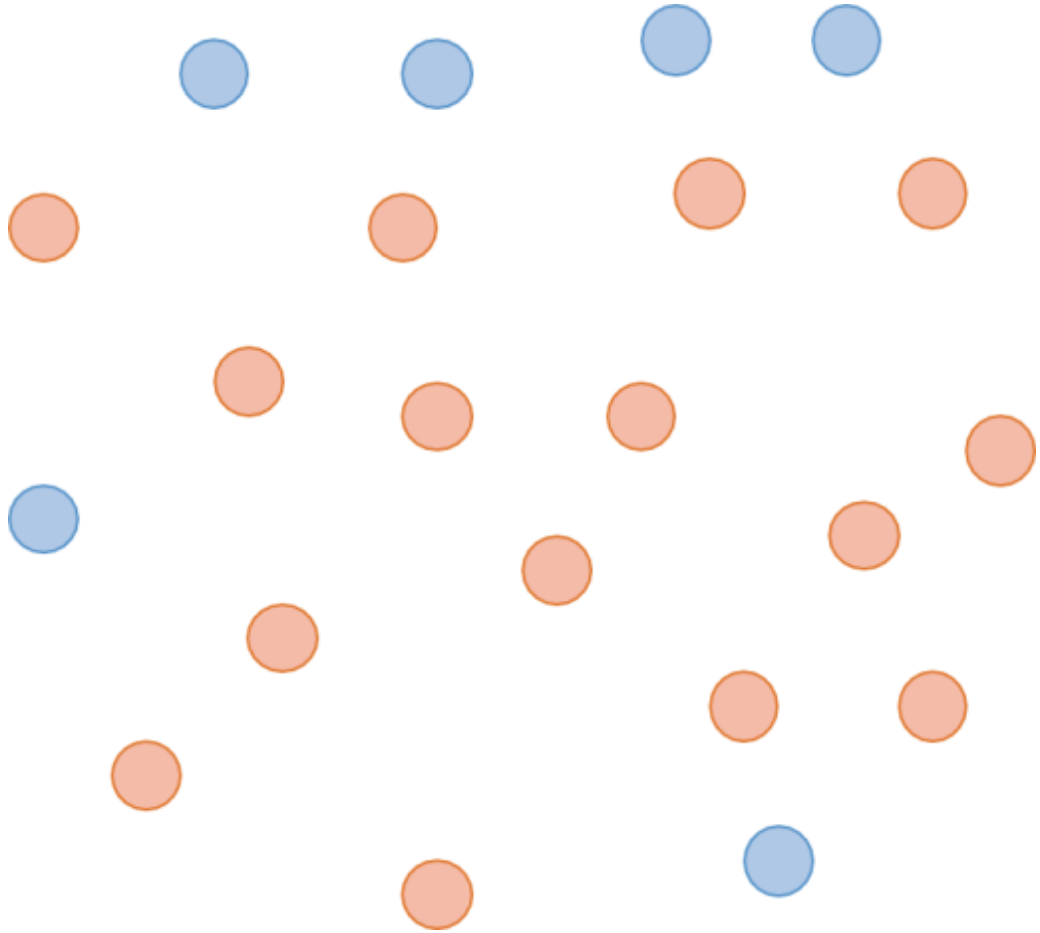
O que é gossiping?

Exemplo de Eugene Bagdasaryan
http://www.cs.cornell.edu/courses/cs6410/2016fa/slides/19-p2p-gossip.pdf



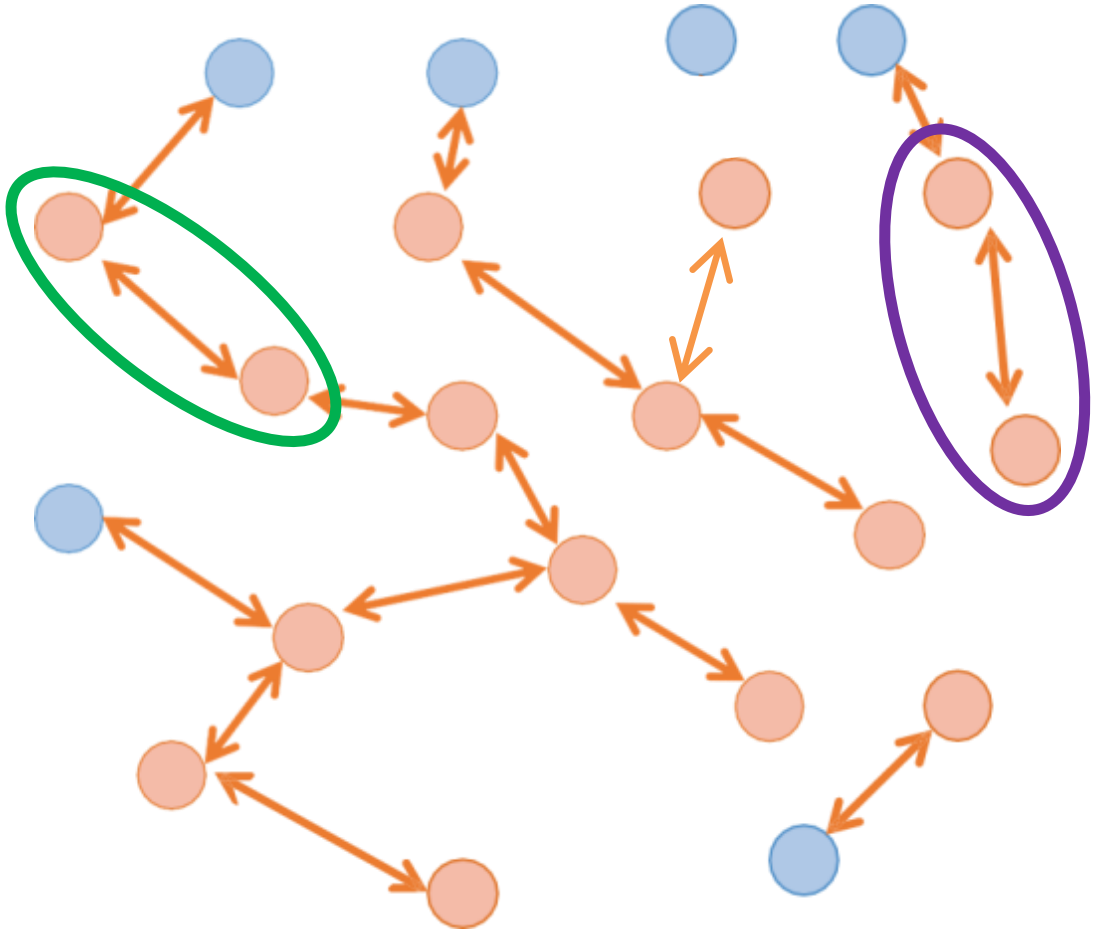
O que é gossiping?

Exemplo de Eugene Bagdasaryan
http://www.cs.cornell.edu/courses/cs6410/2016fa/slides/19-p2p-gossip.pdf



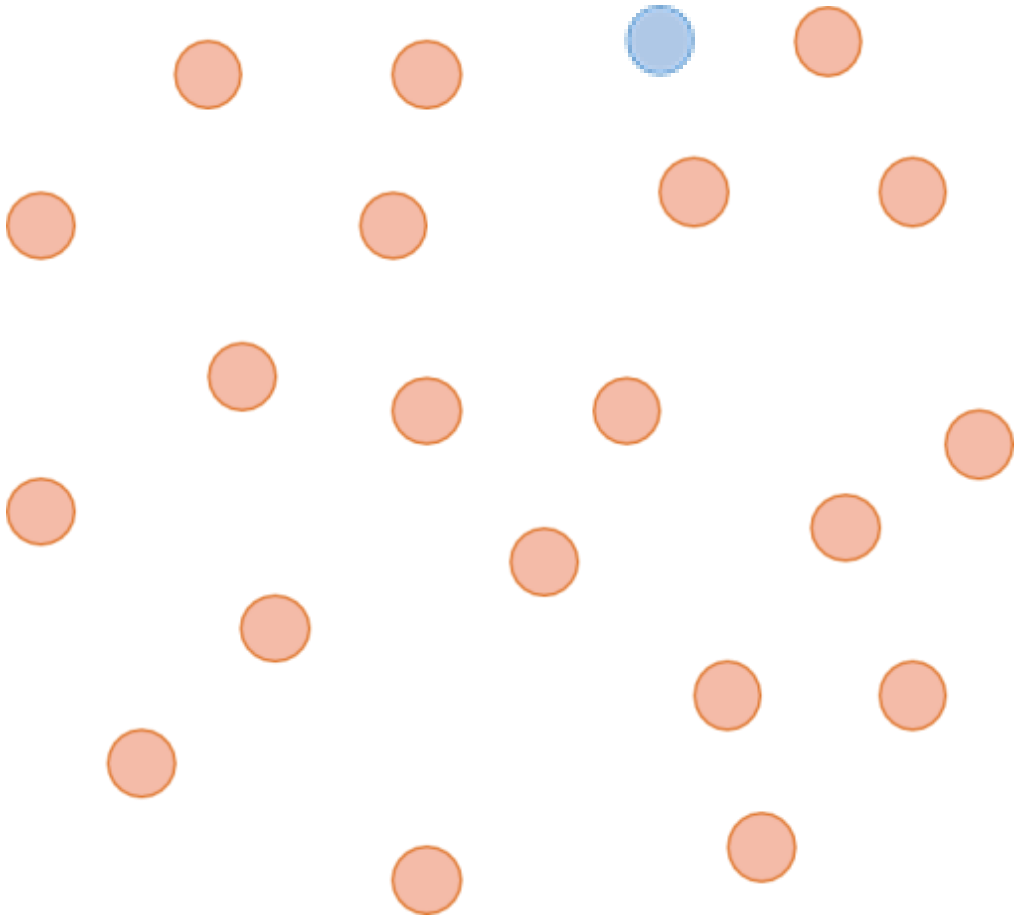
O que é gossiping?

Exemplo de Eugene Bagdasaryan
http://www.cs.cornell.edu/courses/cs6410/2016fa/slides/19-p2p-gossip.pdf



O que é gossiping?

Exemplo de Eugene Bagdasaryan
http://www.cs.cornell.edu/courses/cs6410/2016fa/slides/19-p2p-gossip.pdf



Faltou um

Gossip: Protocolos Epidêmicos → Anti-entropy

Princípios

Um nó *P* seleciona outro nó *Q* do sistema aleatoriamente.

Pull (puxar): *P* somente pede novas atualizações de *Q.*

Push (empurrar): *P* somente envia novas atualizações a *Q*, sem Q ter pedido.

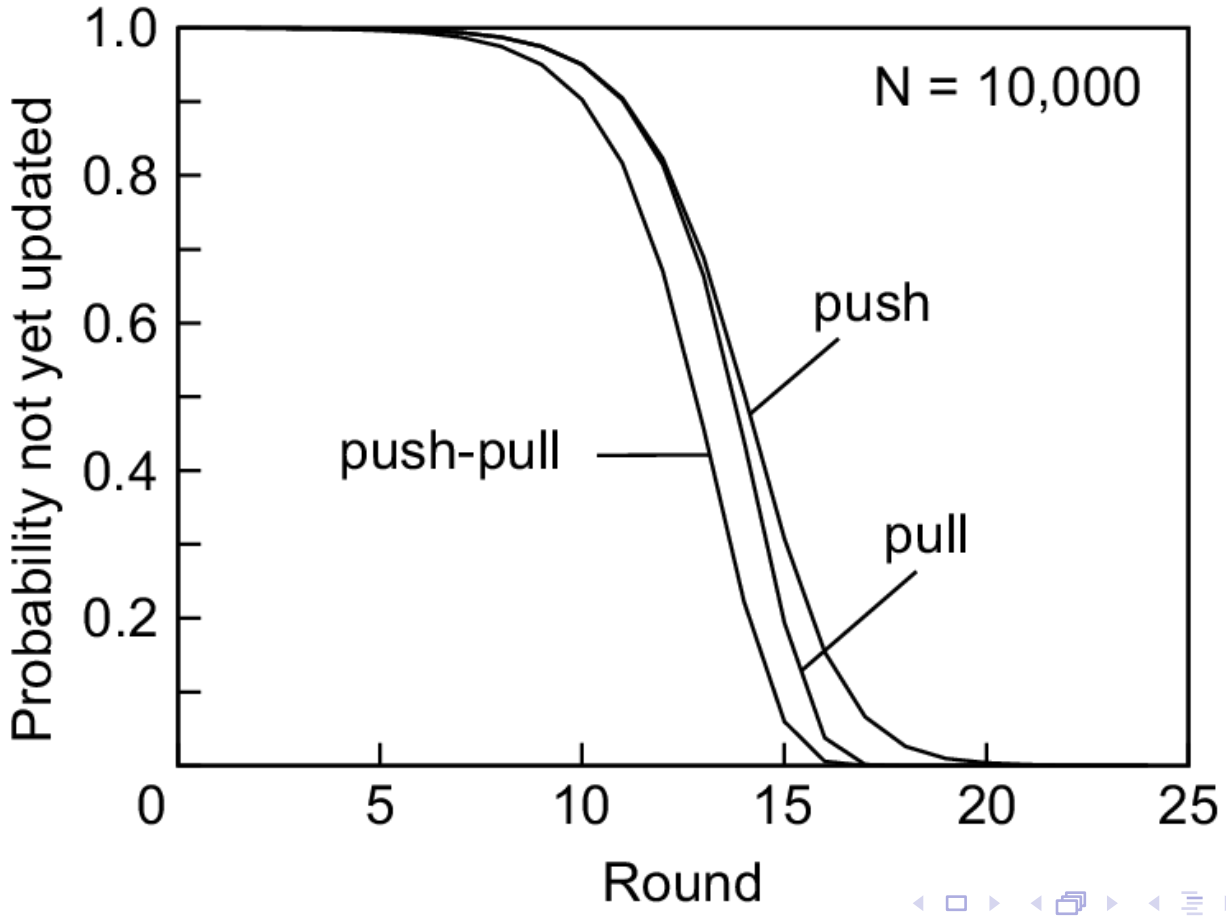
Push-pull: *P* e *Q* enviam atualizações um a outro.

Gossip: Protocolos Epidêmicos → Anti-entropy

Observação

push-pull usa $O(log(N))$ rodadas para disseminar as atualizações nos N nós (rodada = quando cada nó tomou a iniciativa de iniciar o intercâmbio).

Desempenho



Gossip: Protocolos Epidêmicos → Rumor (boato)

Modelo básico

Um nó *S*, com uma atualização a disseminar, contata outros nós. Se o nó contatado já possui a atualização, *S* para de contatar outros nós com probabilidade p_{stop}.

Observação

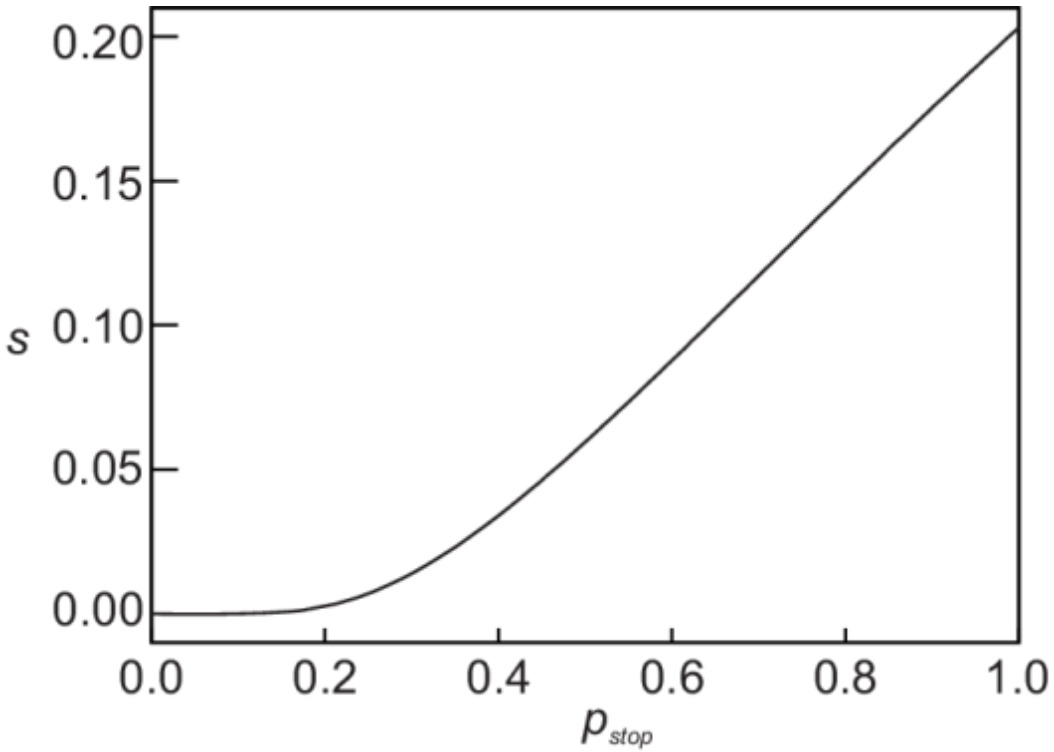
Se *s* é a fração de nós ignorantes (i.e., não tem a atualização), pode ser demonstrado (via SIR) que, para muitos servidores:

$$s = e^{-(1/p_{stop}+1)(1-s)}$$

Gossip: Protocolos Epidêmicos → Rumor (boato)

O efeito de parar

$$s = e^{-(1/p_{stop}+1)(1-s)}$$

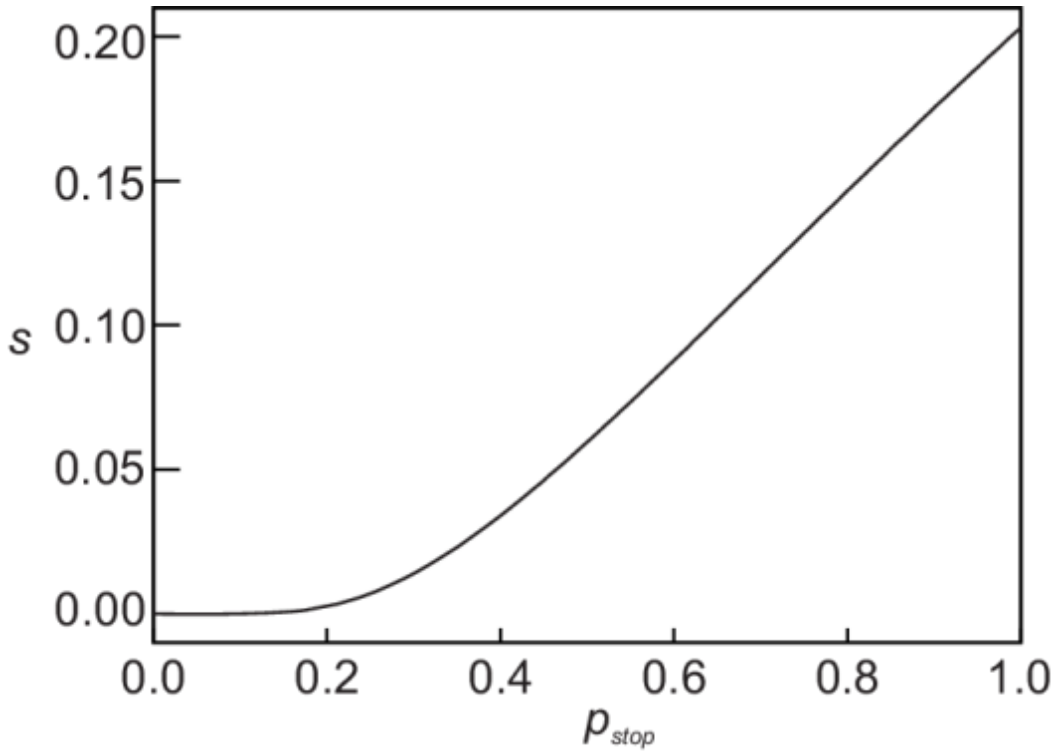


Considere 10,000 nós		
$1/p_{stop}$	s	N_s
1	0.203188	2032
2	0.059520	595
3	0.019827	198
4	0.006977	70
5	0.002516	25
6	0.000918	9
7	0.000336	3

Gossip: Protocolos Epidêmicos → Rumor (boato)

O efeito de parar

$$s = e^{-(1/p_{stop}+1)(1-s)}$$



Considere 10,000 nós		
$1/p_{stop}$	s	N_s
1	0.203188	2032
2	0.059520	595
3	0.019827	198
4	0.006977	70
5	0.002516	25
6	0.000918	9
7	0.000336	3

Nota

Se realmente quer estar seguro de atualizar TODOS os nós, o rumor por si só não é suficiente

Gossip: Remoção de dados

Problema Fundamental

Não é possível remover um valor antigo de um nó e esperar que isso se dissemine. Pelo contrário, a remoção será desfeita pelo uso do algoritmo epidêmico

Solução

A remoção deve ser registrada como uma atualização “especial” usando um death certificate

Conceitos adquiridos

- Comunicação transiente/persistente, assíncrona/síncrona.
- Comunicação RPC.
- Comunicação MOM (Event Streaming).
- Kafka.
- Multicast com ordem FIFO, Causal e Total.
- Comunicação Gossip: anti-entropy e rumor.

Agenda

- Fundamentos
- Comunicação orientada a procedimento (RPC)
- Comunicação orientada a mensagem (MOM)
- Comunicação multicast (Gossip / Flooding)
- Comunicação orientada a fluxo

Comunicação orientada a fluxo (streams)

- Suporte para mídia contínua
- Fluxo em sistemas distribuídos
- Qualidade de Serviço (QoS)

Suporte para mídia contínua

Observação

Toda a comunicação apresentada foi baseada em um intercâmbio de informação discreto e independente do tempo

Mídia Contínua

Caracterizada por ter valores dependentes do tempo:

- Áudio
- Vídeo
- Animações
- Sensores (temperatura, pressão, etc.)

Suporte para mídia contínua

Modos de Transmissão

Diferentes garantias de tempo na entrega de dados:

- Asynchronous: sem restrições de quando a **o pacote** deve ser entregue
- Synchronous: define um delay (end-to-end) máximo para a entrega do pacote
- Isochronous: define um delay (end-to-end) máximo e mínimo para a entrega do pacote (*jitter*)

Fluxo em sistemas distribuídos

Definição

Um fluxo (contínuo) de dados é uma comunicação orientada à conexão que suporta transmissão de dados isochronous.

Características

Fluxos são unidirecionais

Geralmente há uma única fonte

Fluxo simple: único fluxo de dados, e.g., áudio ou vídeo

Fluxo complexo: múltiplos fluxos de dados, e.g., áudio estéreo ou combinação de áudio/vídeo

Qualidade de Serviço

Ideia

Como especificar a Qualidade de Serviço (QoS)?

Exemplos:

- Bit rate requerido no transporte.
- Delay máximo até criar a sessão (i.e., quando a aplicação pode começar a enviar dados).
- Delay máximo do jitter

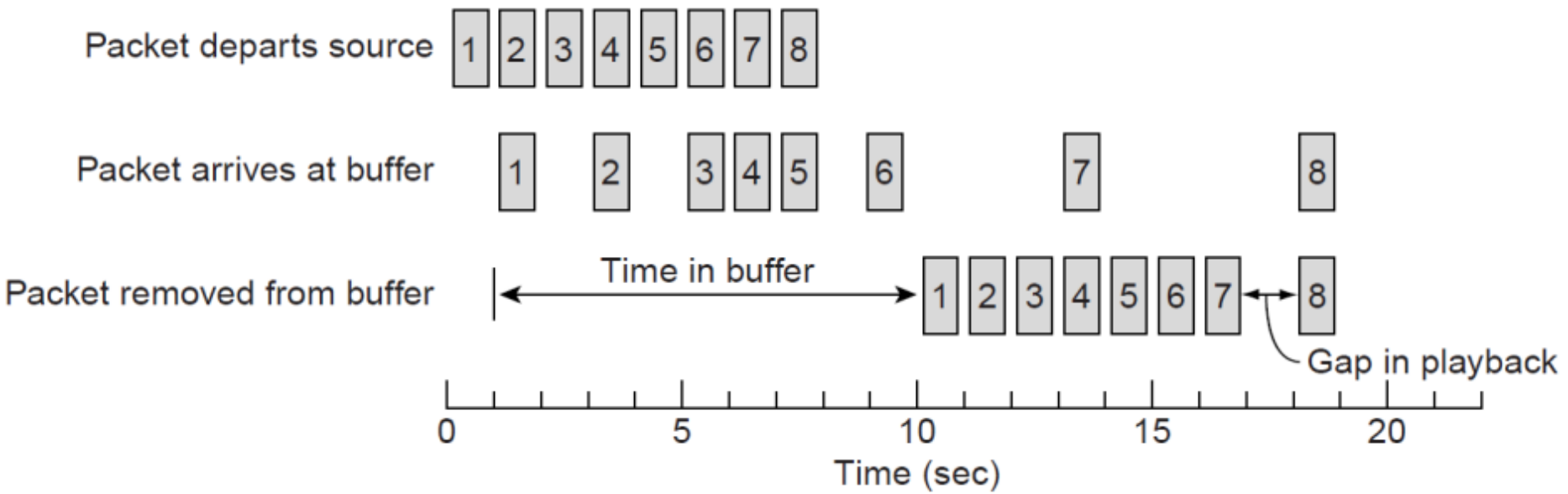
Qualidade de Serviço

Observação

Os pacotes podem ser priorizados por ferramentas inseridas na camada de rede

Também

Uso de buffers na camada de aplicação para reduzir o jitter:



Qualidade de Serviço

Problema

Como reduzir o efeito de pacotes perdidos usando UDP (quando múltiplos quadros estão em um mesmo pacote)?

Exemplo, 4 quadros de áudio em cada pacotes.

Qualidade de Serviço