SISTEMAS DISTRIBUÍDOS

Capítulo 8 - Sistemas de comunicação indireta

Aula 1

NOTA PRÉVIA

A estrutura da apresentação é semelhante e utiliza algumas das figuras do livro de base do curso

G. Coulouris, J. Dollimore, T. Kindberg, G. Blair
Distributed Systems - Concepts and Design,
Addison-Wesley, 5th Edition, 2011

Para saber mais:

secção 6.1-6.4, 15.4 – apenas parte relativa aos tipos de multicast (não entrar na parte de implementação)

COMUNICAÇÃO INDIRETA

Na comunicação indireta, as mensagens não são endereçadas a processos de forma direta ou explícita.

O envio e a recepção das mensagens está desacoplado, no espaço e no tempo

O emissor e o(s) receptor(es) podem não se conhecer;

O emissor e o(s) receptor(es) podem não executar em simultâneo;

A comunicação/interação é suportada por recurso a outras entidades

- comunicação em grupo
- message queues

etc.

FORMAS DE COMUNICAÇÃO : FACETAS

ponto-a-ponto (ou **unicast**) [**1x1**]

– entre um emissor e um único receptor

multi-ponto

broadcast ou difusão total [*1xn*] – envio de uma mensagem de um emissor para todos os receptores

multicast ou comunicação em grupo ou difusão parcial [$1xk \leq n$] – um emissor para um grupo de receptores

funcional ou anycasting [1x1 de n] – envio de uma mensagem de um emissor para um de múltiplos possíveis receptores

geocast [1xk de n] – envio de uma mensagem de um emissor para os receptores de uma dada área geográfica

COMUNICAÇÃO MULTI-PONTO: TCP/IP

A comunicação em difusão total, parcial e difusão funcional podem ser suportados diretamente ao nível da rede TCP/IP.

ex:

broadcast: ping 255.255.255.255 ; 192.168.1.255

multicast: 224.0.0.0-239.255.255.255

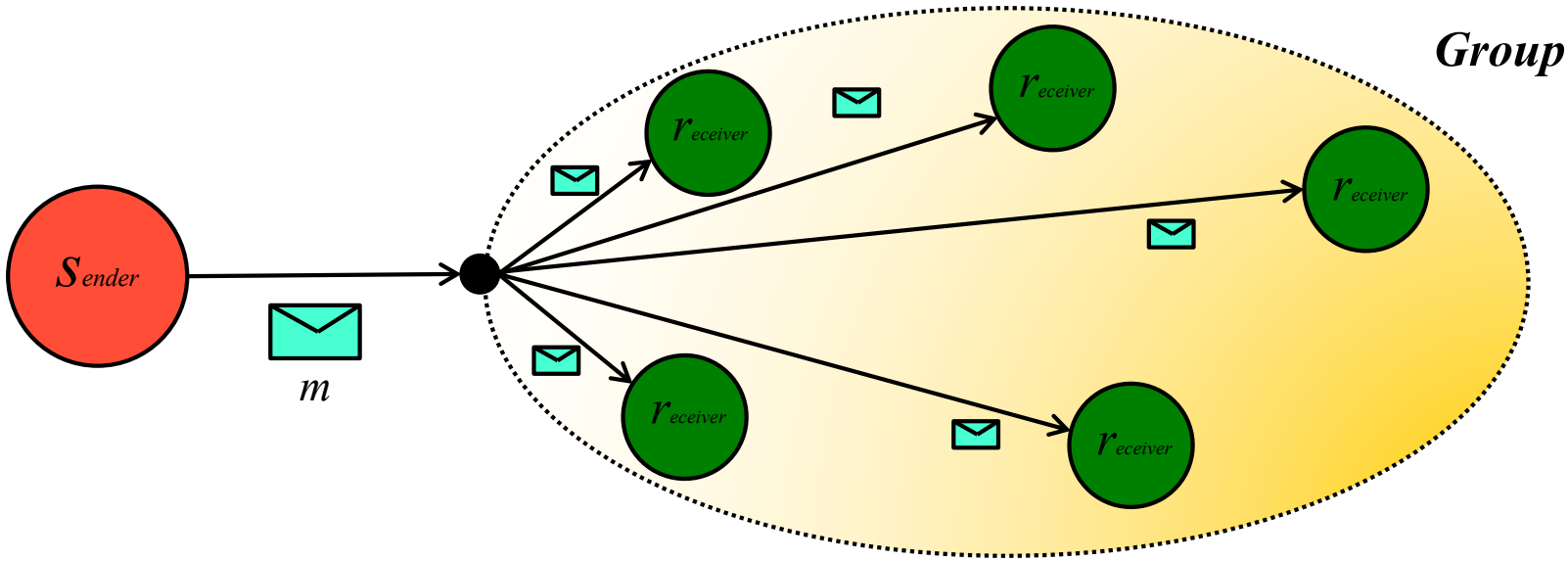
anycast: endereços unicast tratados de forma especial no encaminhamento

Na Internet, anycast é implementado anunciando o mesmo endereço IP a partir de multiplos locais.

Como o encaminhamento escolhe a rota mais curta, o caminho seguido dependerá da origem

Comunicação em grupo : Modelo (1)

Na comunicação em grupo, o grupo é um conjunto **dinâmico** de *communication end-points* (portas, processos, servidores, etc.) que é tratado pelo sistema como uma entidade singular do ponto vista da comunicação.



COMUNICAÇÃO EM GRUPO : MODELO (2)

Características transversais:

O alvo das mensagens é o grupo, designado através de um endereço/identificador **único;**

O emissor **pode** ou **não pertencer** ao grupo;

A composição do grupo é **dinâmica**, não necessariamente visível

A filiação evolui de forma independente dos emissores

API

Típica: send, receive, joinGroup, leaveGroup, createGroup, destroyGroup

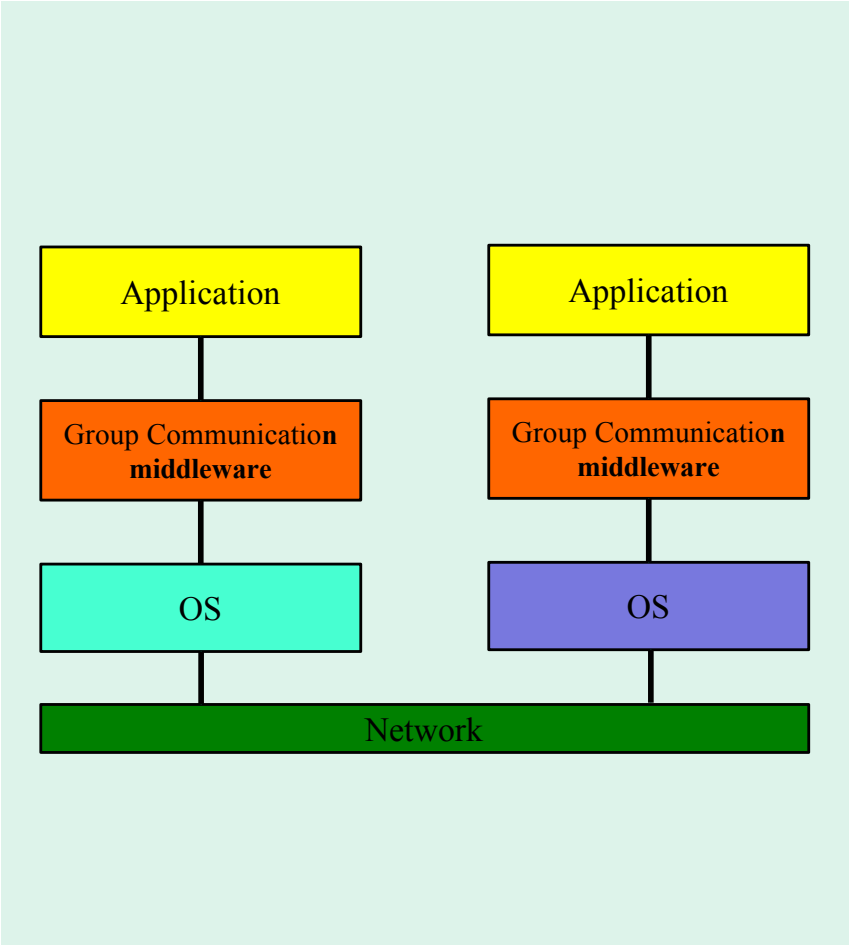
Especializada: PUB$_{lish}$/SUB$_{scribe}$ (modelo editor/assinante)

COMUNICAÇÃO EM GRUPO : IMPLEMENTAÇÃO EM MIDDLEWARE

A comunicação em grupo exposta diretamente pelo OS oferece funcionalidades limitadas:

- garantias de entrega pobres;
  não fiável, sem ordem, ...
- âmbito local (devido a restrições administrativas ao nível da rede)

A implementação em *middleware* dedicado é a forma usual de endereçar estas limitações.



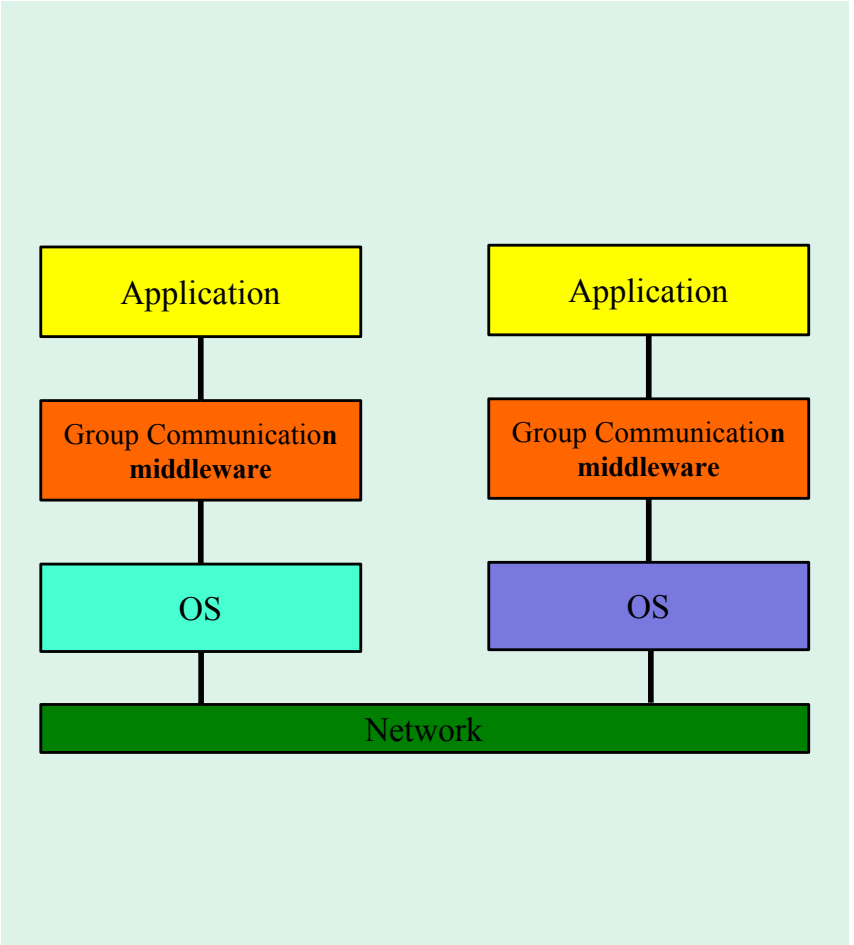
COMUNICAÇÃO EM GRUPO : IMPLEMENTAÇÃO EM MIDDLEWARE

O middleware de comunicação em grupo fica responsável por questões como:

Gestão da filiação, incluindo a expansão dos *endpoints* dos membros do grupo

group_id -> {receiver1, receiver2, ...}

Gestão da comunicação: envio, encaminhamento, buferização das mensagens, ACKs, NACKs, etc.

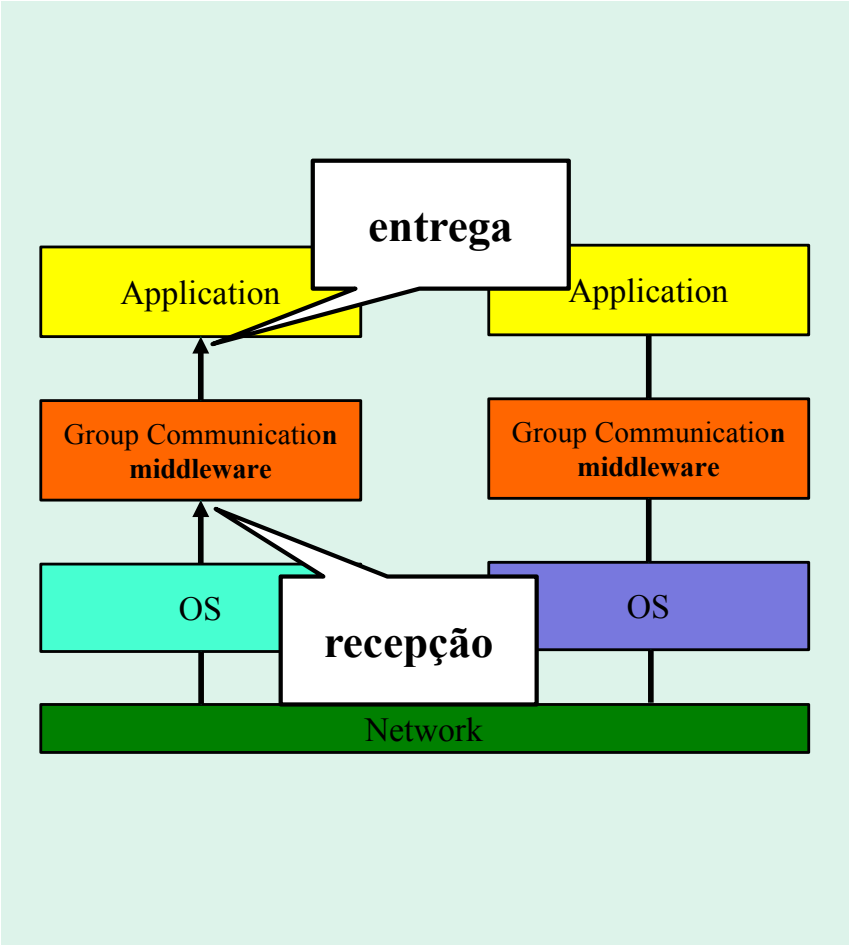


COMUNICAÇÃO EM GRUPO : ENTREGA DAS MENSAGENS

Diz-se que o sistema de comunicação **entrega** uma mensagem quando ela é entregue à aplicação para ser consumida

O middleware pode **atrasar a entrega** das mensagens, **reordená-las**, a fim de garantir as propriedades desejadas

Em cada instância do *middleware*, haverá para esse efeito, uma fila local de mensagens recebidas, ainda por entregar



CARACTERIZAÇÃO DO MULTICAST: FIABILIDADE

Multicast não fiável (*unreliable multicast*) – em caso de falha, não existem garantias sobre a entrega das mensagem aos vários elementos do grupo.

i.e., a mensagem pode ser não entregue a nenhum ou a alguns dos membros do grupo.

e.g. IP mutlicast

Multicast fiável (*reliable multicast*) – uma mensagem enviada para um grupo ou é entregue a todos os membros correctos (que não falham) ou a nenhum

Membros que falham podem ter entregue a mensagem ou não

Implementação simples (e errada): o emissor emite, individualmente, para cada um dos elementos do grupo de forma fiável (acks+retransmissões)

Porquê errada?

CARACTERIZAÇÃO DO MULTICAST: FIABILIDADE

Multicast não fiável (unreliable multicast) – em caso de falha, não existem garantias sobre a entrega das mensagem aos vários elementos do grupo

- e.g. IP mutlicast

Multicast fiável (reliable multicast) – uma mensagem enviada para um grupo ou é entregue por todos os membros correctos (que não falham) ou por nenhum

- Membros que falham podem ter entregue a mensagem ou não
- Implementação simples (e errada): o emissor emite para cada um dos elementos do grupo de forma fiável (acks+retransmissões)
  - Em caso de falha do emissor, é necessário que os elementos do grupo propaguem as mensagens recebidas para os elementos que ainda não as receberam

Uniform agreement: se algum processo que falha entrega a mensagem, todos os processos correctos devem entregar a mensagem

CARACTERIZAÇÃO DO MULTICAST: ORDEM

Sem ordem – as mensagens podem ser entregues por diferentes ordens em diferentes processos

Ordem FIFO – as mensagens de um processo são entregues pela ordem de emissão em todos os processos

Na implementação recorre-se a números de sequência gerados em cada emissor…

CARACTERIZAÇÃO DO MULTICAST: ORDEM

Ordem causal – se **m1** pode causar **m2**, **m1** deve ser entregue sempre antes de **m2**

Se duas mensagens forem **emitidas pelo mesmo processo**, considera-se que o **envio da primeira pode causar o envio da segunda**

A entrega da primeira deve acontecer antes da entrega da segunda

Se duas mensagens forem **emitidas em processos diferentes**, a **primeira pode causar a segunda se for entregue antes do envio da segunda** (ou transitivamente incluindo outras mensagens)

Se uma mensagem não pode causar a outra, qualquer ordem de entrega respeita a ordem causal

NOTA: a noção de causalidade estende-se a qualquer tipo de comunicação

CARACTERIZAÇÃO DO MULTICAST: ORDEM

Ordem causal – se **m1** pode causar **m2**, **m1** deve ser entregue sempre antes de **m2**

A implementação da ordem causal pode ser conseguida por recurso a relógios lógicos (+ ACKs) ou relógios vetoriais.

CARACTERIZAÇÃO DO MULTICAST: ORDEM

Ordem total – as mensagens m1, m2 são entregues por ordem total, se forem entregues pela **mesma ordem em todos os processos**

- ∀pi, receive(m1,pi) < receive(m2,pi) **ou**
- ∀pi, receive(m1,pi) > receive(m2,pi)

NOTA: a ordem de entrega das mensagens de um mesmo emissor pode ser diferente da ordem de emissão

Um sistema de multicast fiável no qual as mensagens são entregues por ordem total chama-se sistema de **multicast atómico**

Ordem total causal – ordem total + ordem causal

As mensagens são entregues pela **mesma ordem** em **todos** os processos, segundo uma ordem que respeita a **causalidade**.

CARACTERIZAÇÃO DO MULTICAST: ORDEM

Ordem total – as mensagens m1, m2 são entregues por ordem total, se forem entregues pela **mesma ordem em todos os processos**

- ∀pi, receive(m1,pi) < receive(m2,pi) **ou**
- ∀pi, receive(m1,pi) > receive(m2,pi)

NOTA: a ordem de entrega das mensagens de um mesmo emissor pode ser diferente da ordem de emissão

Um sistema de multicast fiável no qual as mensagens são entregues por ordem total chama-se sistema de **multicast atómico**

Ordem total causal – ordem total + ordem causal

As mensagens são entregues pela **mesma ordem** em **todos** os processos, segundo uma ordem que respeita a **causalidade**.

CARACTERIZAÇÃO DO MULTICAST: ORDEM

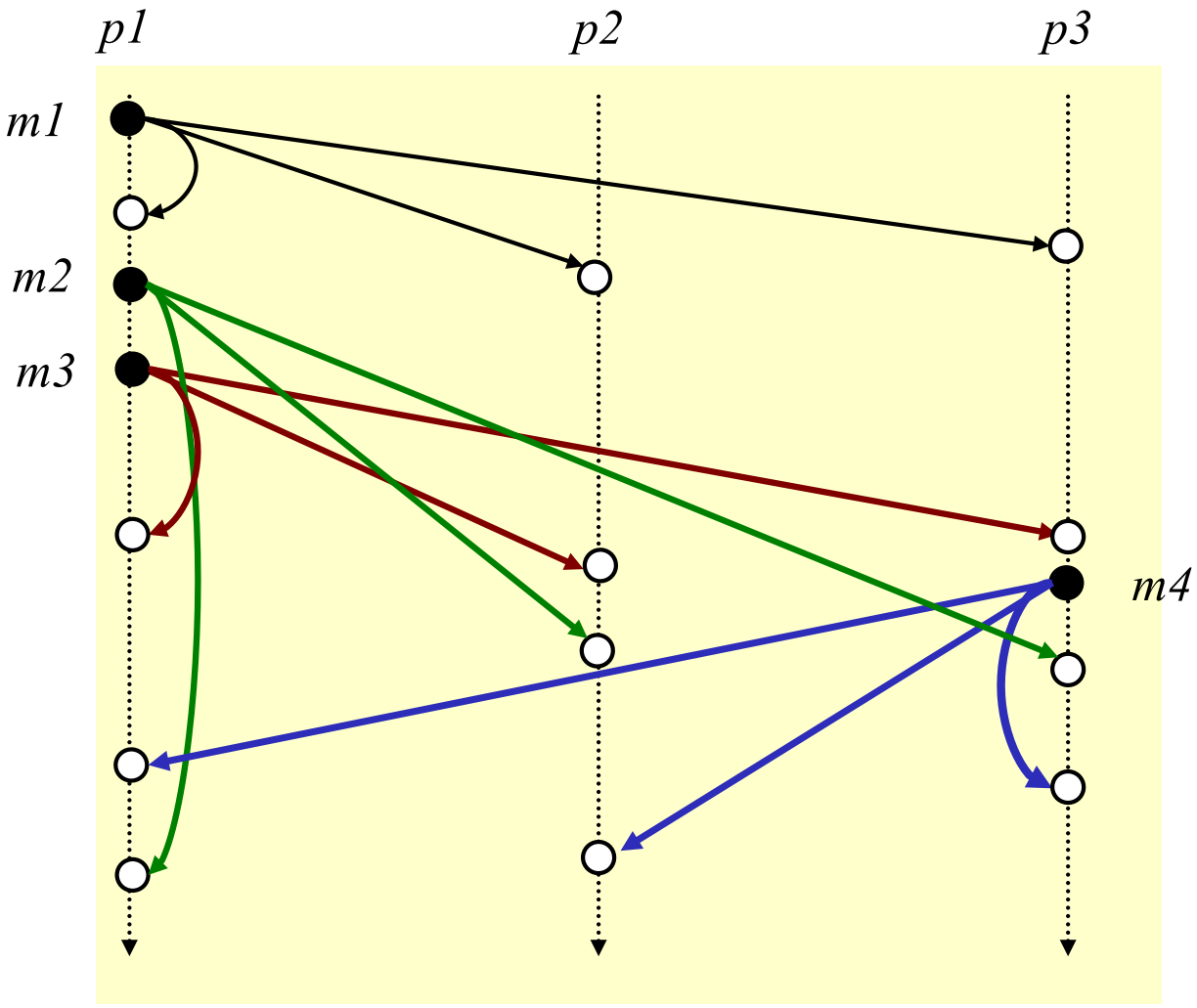
Ordem total – as mensagens m1, m2 são entregues por ordem total, se forem entregues pela **mesma ordem em todos os processos**

Implementação pode ser conseguida à custa de um **sequenciador**.

- solução centralizada – as mensagens são enviadas ao sequenciador e este impõe a ordem (transformando a ordem total em ordem FIFO).

- solução descentralizada – o sequenciador apenas emite uma "stream" adicional que indica a ordem de entrega a observar em todos os receptores.

ORDENS: EXEMPLO



m1,**m2**: FIFO, total, causal
m1,**m3**: FIFO, total, causal
m1,**m4**: total, causal

m2,**m3**: total, não fifo, não causal

m2,**m4**: sem ordem

m3,**m4**: causal, total

CARACTERIZAÇÃO DO MULTICAST: VISTAS

Uma **vista** (view) é o conjunto de membros de um grupo, num dado momento.

- Nos sistemas de comunicação em grupo fiáveis, a mudança de vista é efectuada através do envio de uma mensagem multicast

Sincronia virtual (virtual synchrony) – um sistema de multicast fiável implementa sincronia virtual se:

- as mudanças de vista são entregues em todos os processos pela mesma ordem
- o conjunto de mensagens entregues entre a entrega de cada duas vistas consecutivas é idêntico para todos os processos que observam as duas vistas

FACETAS DA COMUNICAÇÃO MULTICAST: RESUMO

- Fiabilidade
  - Multicast fiável
  - Multicast não-fiável
- Ordenação das mensagen
  - Sem ordem
  - Ordem FIFO
  - Ordem total
  - Ordem casual
  - Ordem total causal
- Relativamente às vistas
  - Sincronia virtual

Sistemas de Comunicação em Grupo e Replicação de dados

Pode-se usar um sistema de comunicação em grupo para propagar e executar operações num conjunto de réplicas. Que semântica usar nos seguintes cenários:

- Pretende-se manter um forúm de mensagens, em que uma mensagem pode ter uma resposta. Uma operação que cria um nova mensagem/resposta deve ser entregue com que semântica?
- Pretende-se manter o stock dum produto, garantindo que o seu valor não excede um dado limite. Uma operação para modificar o stock deve ser entregue com que semântica?

SISTEMAS DISTRIBUÍDOS

Capítulo 8 - Sistemas de comunicação indireta

Aula 2

NA ÚLTIMA AULA: FACETAS DA COMUNICAÇÃO EM GRUPO

- Fiabilidade
  - Multicast fiável
  - Multicast não-fiável
- Ordenação das mensagen
  - Sem ordem
  - Ordem FIFO
  - Ordem total
  - Ordem casual
  - Ordem total causal
- Relativamente às vistas
  - Sincronia virtual

AGENDA PARA HOJE

Sistemas de disseminação de eventos: pub/sub

Sistemas de message queues

SISTEMAS DE DISSEMINAÇÃO DE EVENTOS: PUB/SUB

Pode ser visto como um sistema comunicação em grupo, com uma interface especializada.

Modelo Genérico:

- o(s) **editor**(es)/publisher(s) **publica**(m) mensagens/eventos, usualmente organizadas por **tópicos** ou **canais** temáticos;

- o(s) **assinante**(s)/subscriber(s)) manifestam interesse em receber certas mensagens, indicando o(s) **tópicos**, **canal** ou **padrão do conteúdo** das mensagens a receber

Os sistemas pub/sub que suportam filtragem baseada no conteúdo implementam o conceito de **content-based routing**

SISTEMAS DE DISSEMINAÇÃO DE EVENTOS: PUB/SUB

Os editores não necessitam de conhecer quais ou quantos subscritores existem ou se receberam as suas mensagens

Tal permite desacoplar o envio da recepção das mensagens/eventos;

As interações são assíncronas e podem ocorrer mesmo quando os editores e os assinantes não coexistem no tempo

PUBLISH/SUBSCRIBE: ARQUITECTURAS

Cliente/servidor

O(s) servidor(es) designa(m)-se de **broker(s)**

Tanto os editores como os assinantes operam como clientes do broker.

Permite implementar funcionalidades adicionais com relativa facilidade– e.g.: persistência, filtragem baseada no conteúdo, replicação, particionamento.

Peer-to-peer

Sem ponto central de falhas, potencialmente mais escalável

Pode recorrer a multicast tradicional, basear-se em comunicação epidémica, ou utilizar uma DHT

PUBLISH/SUBSCRIBE: EXEMPLO APACHE KAFKA

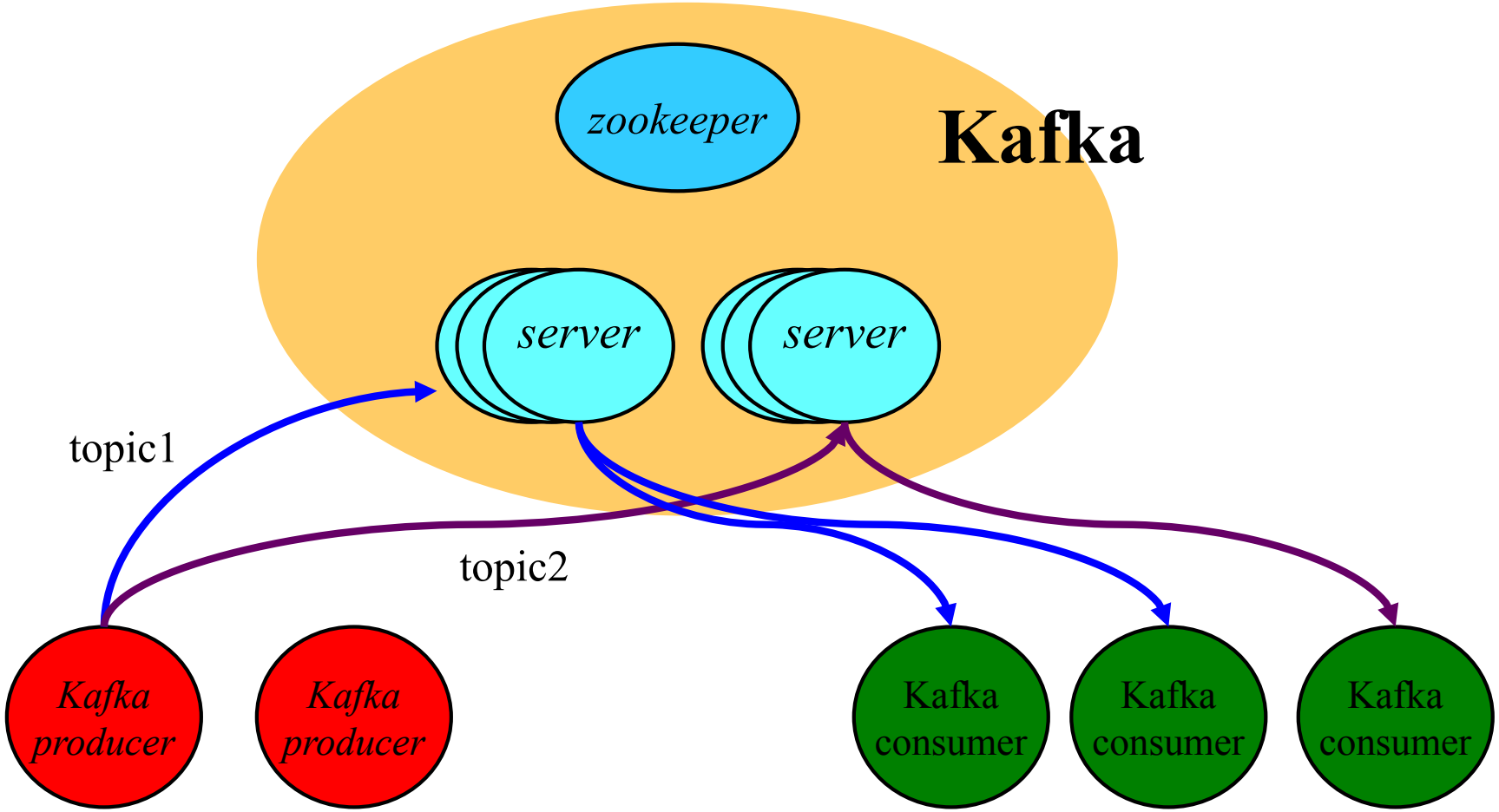
Apache Kafka é um sistema de comunicação em grupo (bastante sofisticado) que oferece uma API publish/subscribe, baseada em tópicos.

Oferece diversas garantias, incluindo persistência e múltiplas semânticas de entrega

Suporta particionamento e replicação dos servidores (brokers).

Depende do Zookeeper para tolerar falhas e resolver o *rendezvous* entre produtores e consumidores

APACHE KAFKA: ARQUITETURA



KAFKA: PUBLISHER (PRODUCER)

```
        Properties props = new Properties();

        props.put("bootstrap.servers", "localhost:9092");

        props.put("key.serializer", StringSerializer.class.getName());
        props.put("value.serializer", StringSerializer.class.getName());

        props.put("log.retention.ms", 1000);

        KafkaProducer<String, String> producer = new KafkaProducer<>(props);

        String topic = …;
        String event = …;

        ProducerRecord<String, String> data = new ProducerRecord<>(topic ,
event);
        producer.send(data);
        …
        producer.close();
```

KAFKA: SUBSCRIBER (CONSUMER)

```
Properties props = new Properties();

props.put("bootstrap.servers", "localhost:9092");

props.put("key.deserializer", StringDeserializer.class.getName());
props.put("value.deserializer", StringDeserializer.class.getName());

props.put("group.id", "gid" + System.nanoTime());

KafkaConsumer<String, String> consumer = new KafkaConsumer<>(props);

consumer.subscribe(Arrays.asList("topic0", "topic1", … ));

ConsumerRecords<String, String> records = consumer.poll(1000);
records.forEach( r -> {
        System.err.println( r.topic() + "/" + r.value());
});

consumer.close();
```

PUBLISH/SUBSCRIBE: PROPRIEDADES

Desacoplamento entre componentes

Vantagens:

- Emissor não necessita de conhecer receptor
- Emissor e receptor não necessitam de estar a funcionar ao mesmo tempo
- Emissor não necessita de esperar pelo receptor - assincronismo
- Melhor desempenho

Desvantagens:

- Mais complicado definir propriedades exactas do sistema

Modelo de programação *event-driven*

- Modelo reactivo - como nos GUIs

PUBLISH-SUBSCRIBE: SISTEMAS E *STANDARDS*

Web-services

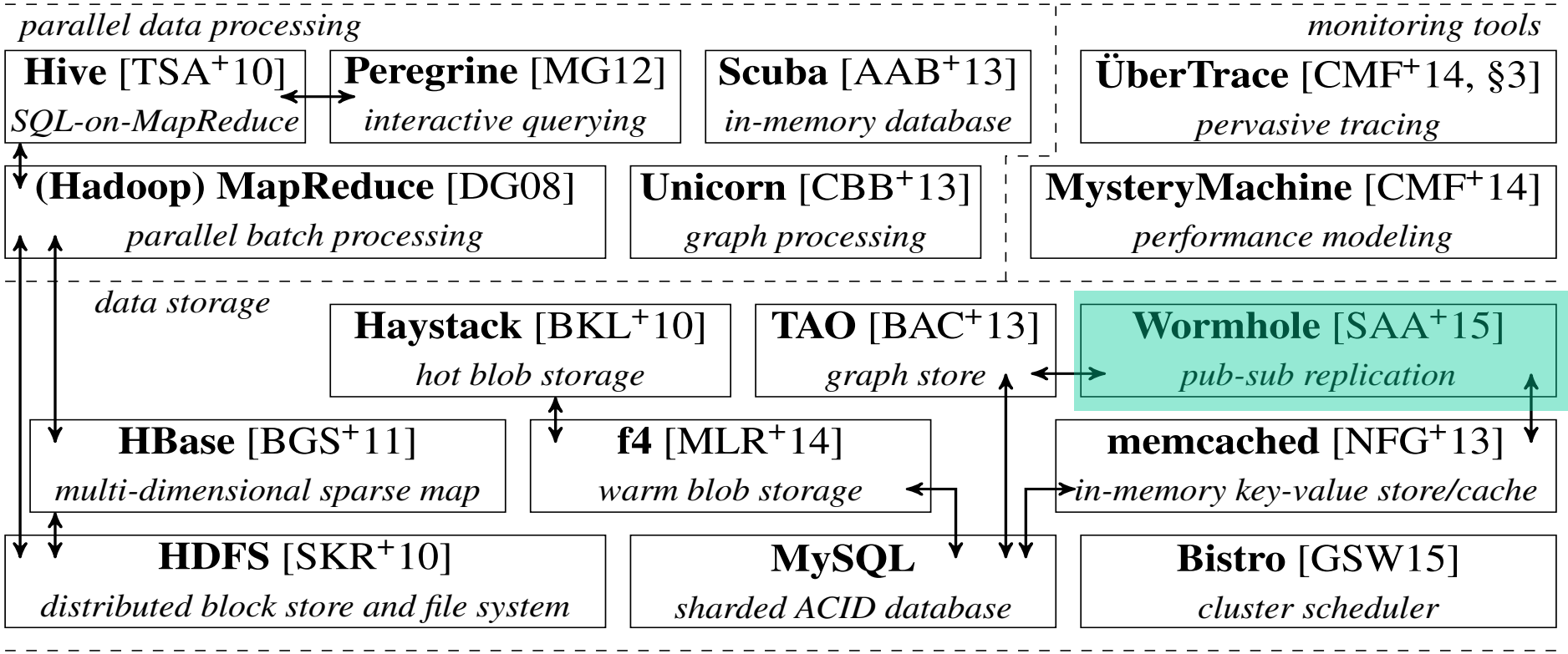
- WS-Notification
  - Tem mais funcionalidade que WS-Eventing, mas são mutuamente incompatíveis
- WS-Eventing

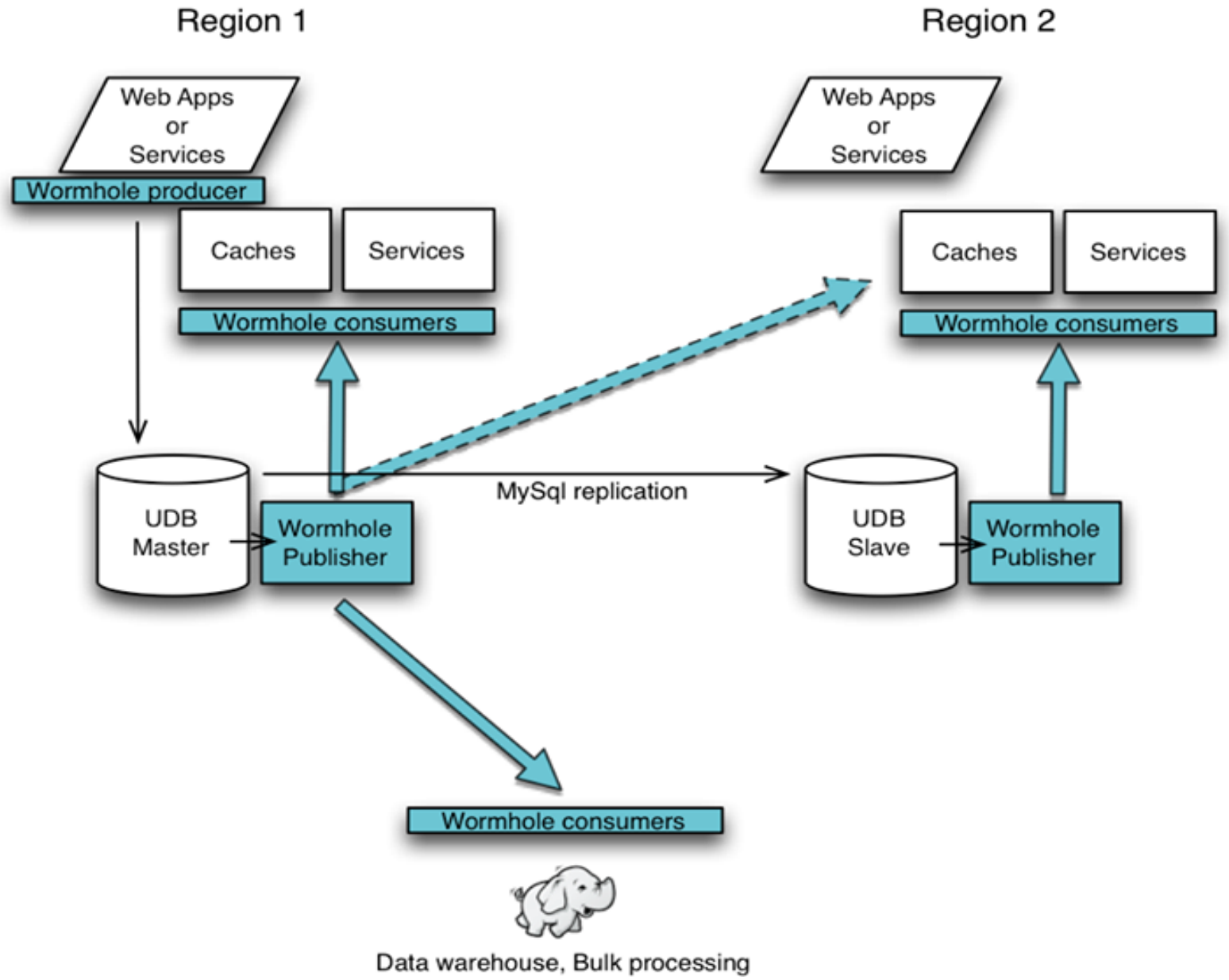
Java: JMS (Java Messaging System)

Sistemas:

- TiBCO, IBM Gryphon, Apache pubscribe, Kafka, …

EXEMPLO: FACEBOOK




Region 1
Region 2
Web Apps or Services
Wormhole producer
Caches
Services
Wormhole consumers
Web Apps or Services
Caches
Services
Wormhole consumers
MySql replication
UDB Master
Wormhole Publisher
UDB Slave
Wormhole Publisher
Wormhole consumers
Data warehouse, Bulk processing

AGENDA PARA HOJE

Sistemas de disseminação de eventos: pub/sub

Sistemas de message queues

Message-Oriented Middleware (MOM) ou Message-Queuing systems

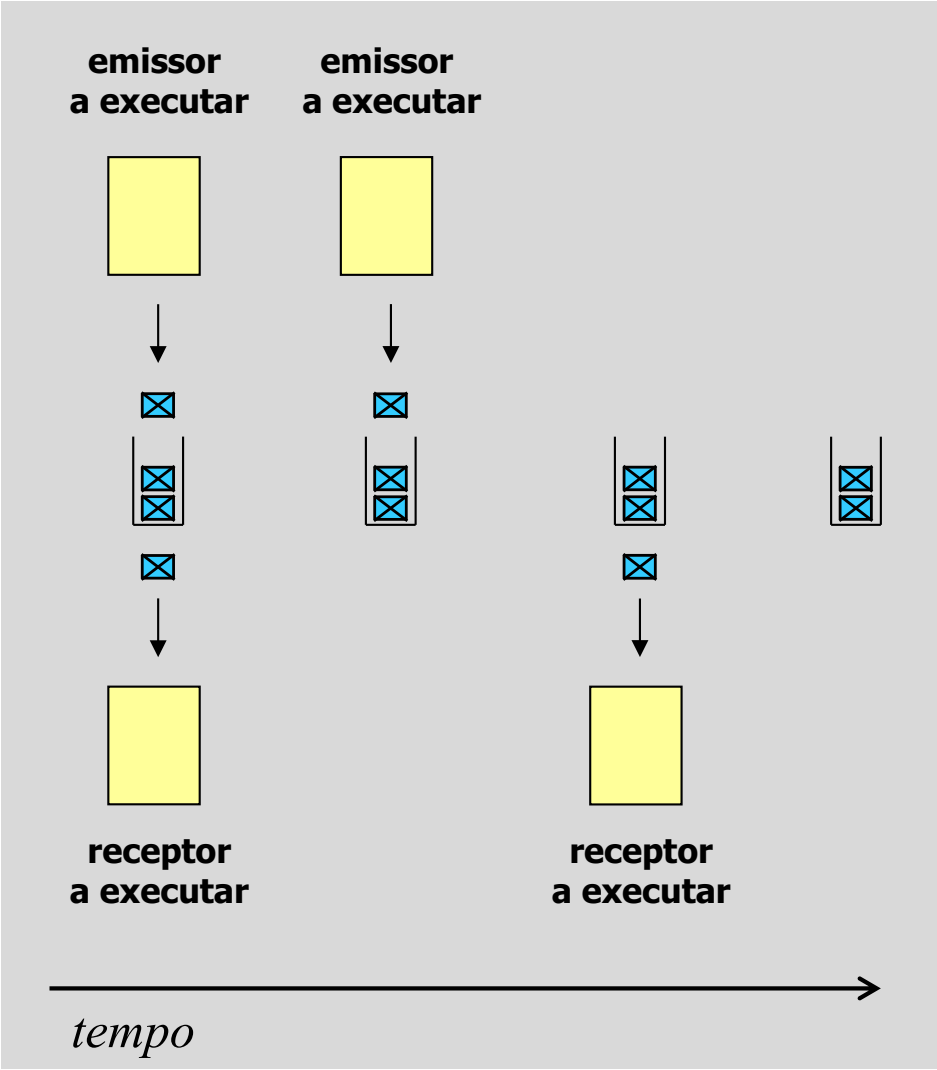
Os processos comunicam pela troca de mensagens usando um subsistema intermédio que assegura a persistência através de filas de mensagens (queues)

- Comunicação assíncrona
- Suporta transferência de mensagens que duram vários minutos

Uma mensagem é sempre endereçada por um processo para uma queue e só pode ser consumida da queue por um outro processo / aplicação (uma queue pode, no entanto, ser partilhada por vários processos / aplicações)

- O emissor apenas tem garantias que a mensagem foi entregue na queue

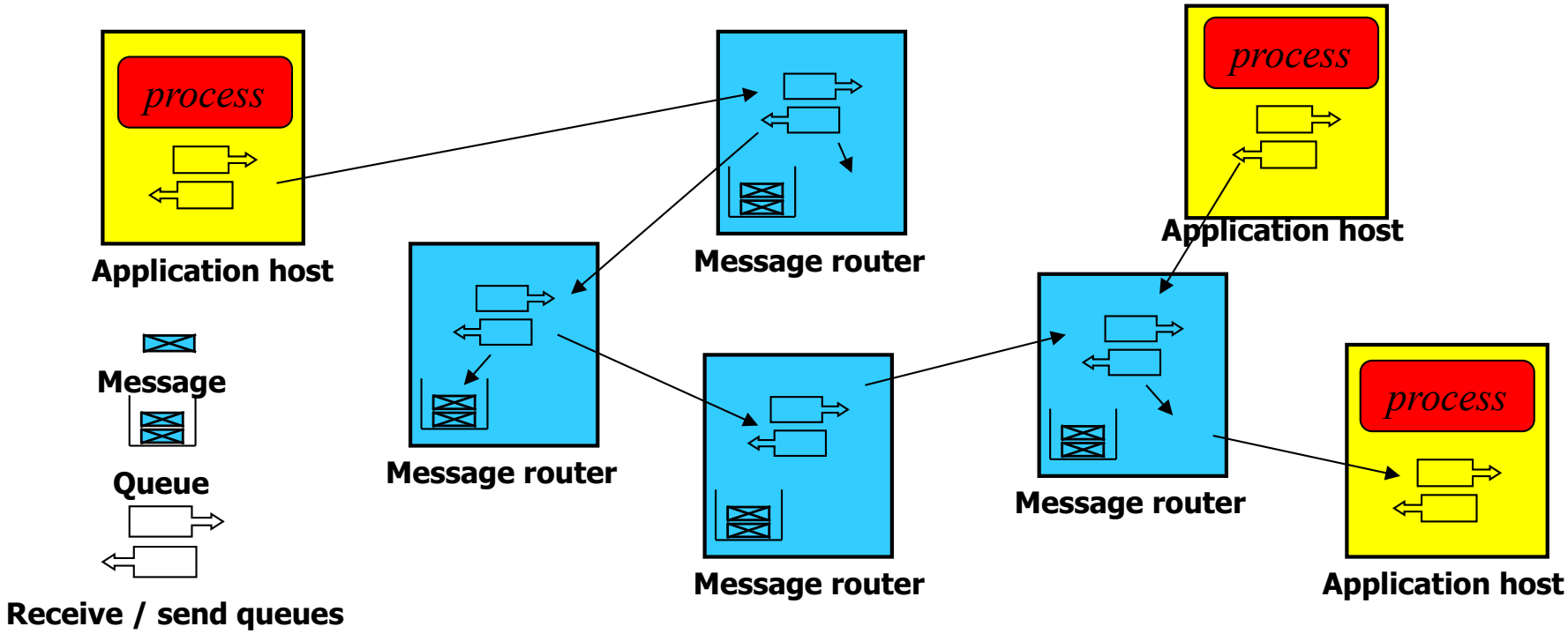
INTERACÇÃO EMISSOR / RECEPTOR



Primitivas:

- Put – adiciona uma mensagem a uma *queue*
- Get – bloqueia até que a *queue* não esteja vazia e remove a primeira mensagem
- Poll – verificar se existem mensagens na *queue*
- Notify – Instalar um *handler* para ser notificado se uma mensagem chegar

Estrutura geral de um MOM



Mensagens podem ser propagadas para o destino através de filas intermédias

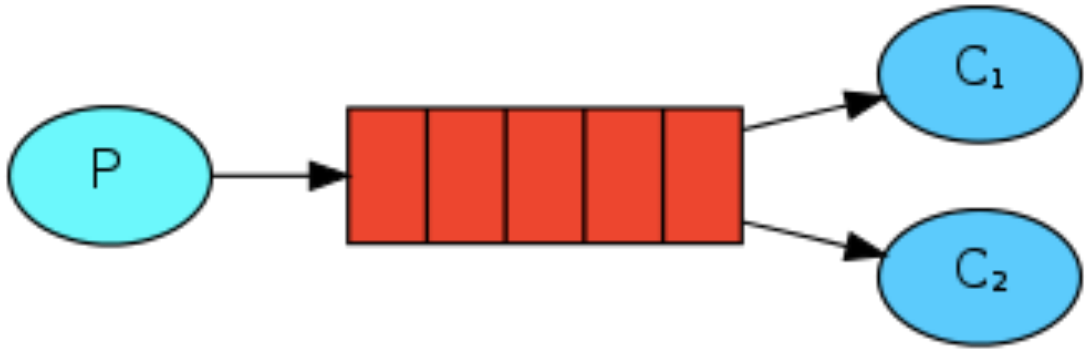
- Algumas máquinas podem ter filas e actuar apenas como *routers*
  - Permite implementar funcionalidade adicional – ex.: log para segurança ou tolerância a falhas, conversão de formato de mensagens/gateway (message brokers)

Produtos: RabbitMQ, IBM MQSeries

ALGUNS USOS DE SISTEMAS DE MESSAGE-QUEUE

(Figures from: https://www.rabbitmq.com/tutorials/tutorial-three-java.html)

Distribuir tarefas por múltiplos processos

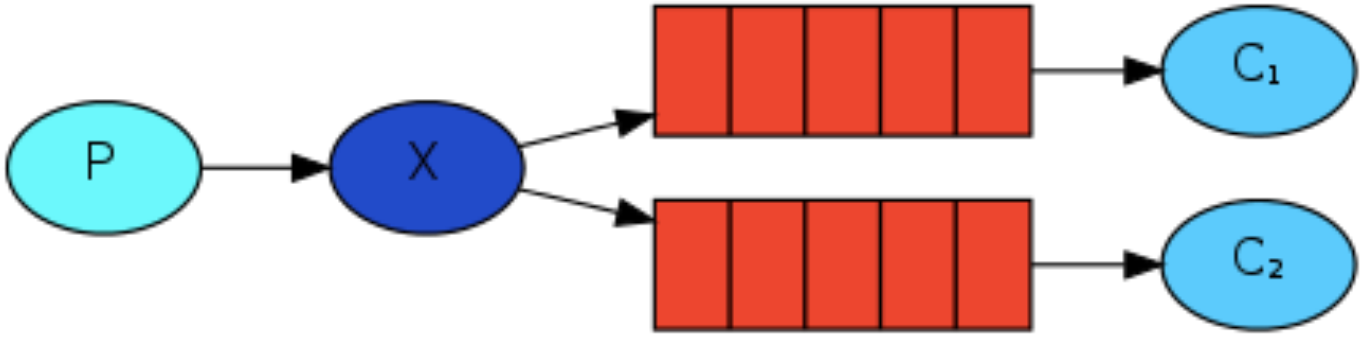


ALGUNS USOS DE SISTEMAS DE MESSAGE-QUEUE

(Figures from: https://www.rabbitmq.com/tutorials/tutorial-three-java.html)

Disseminar tarefas para múltiplos processos

Broadcast, Publish/Subscribe (filtragem no receptor)

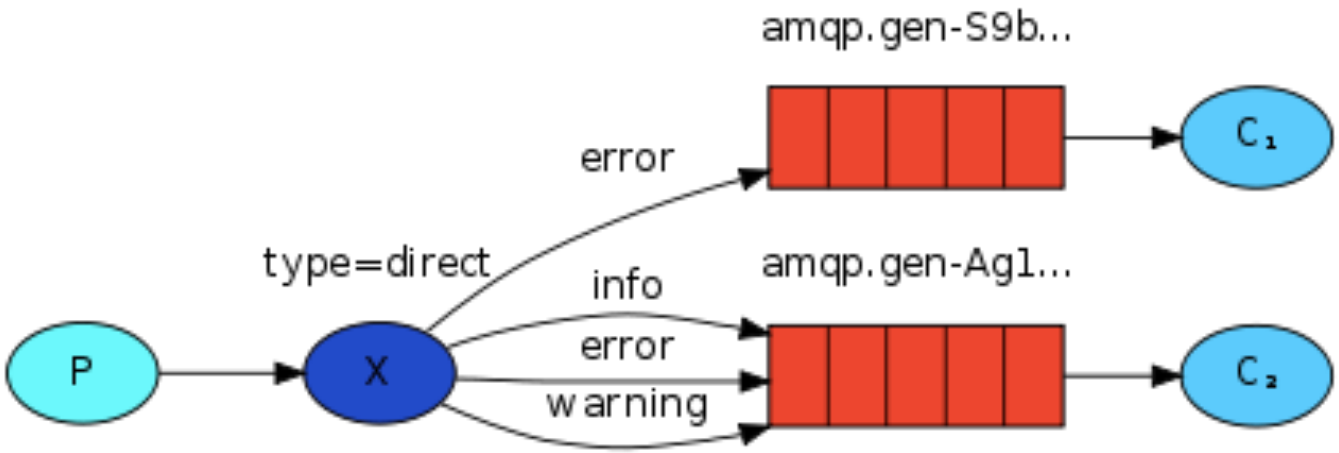


ALGUNS USOS DE SISTEMAS DE MESSAGE-QUEUE

(Figures from: https://www.rabbitmq.com/tutorials/tutorial-three-java.html)

Receber mensagens seletivamente

publish/subscribe (baseada em tópicos)

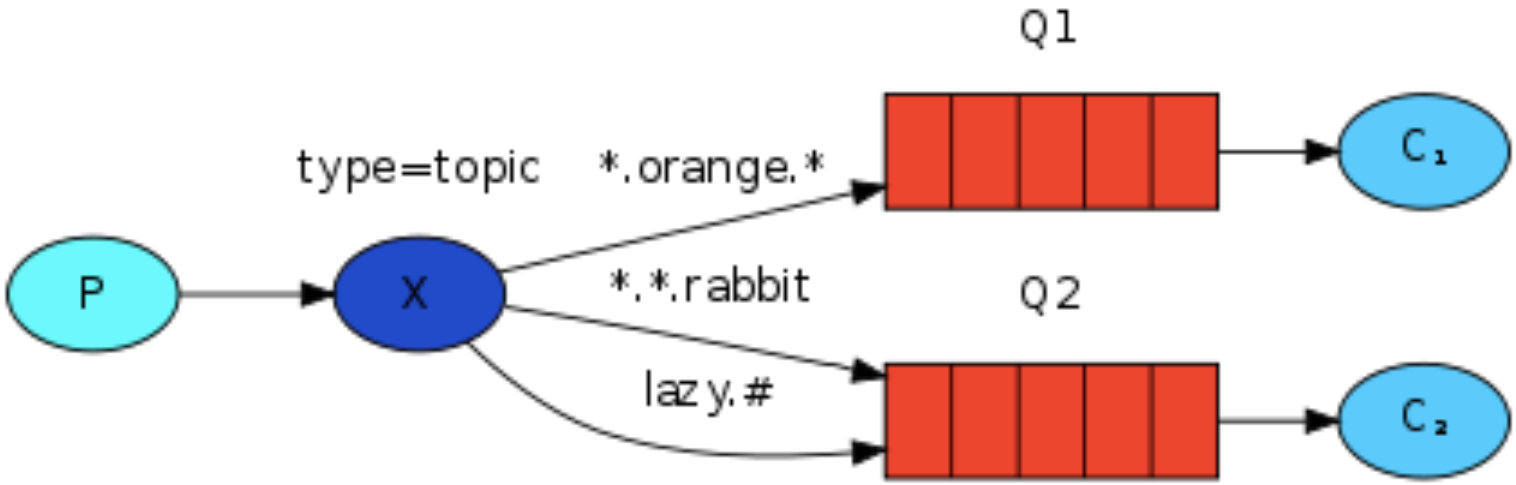


ALGUNS USOS DE SISTEMAS DE MESSAGE-QUEUE

(Figures from: https://www.rabbitmq.com/tutorials/tutorial-three-java.html)

Receber mensagens baseado num padrão

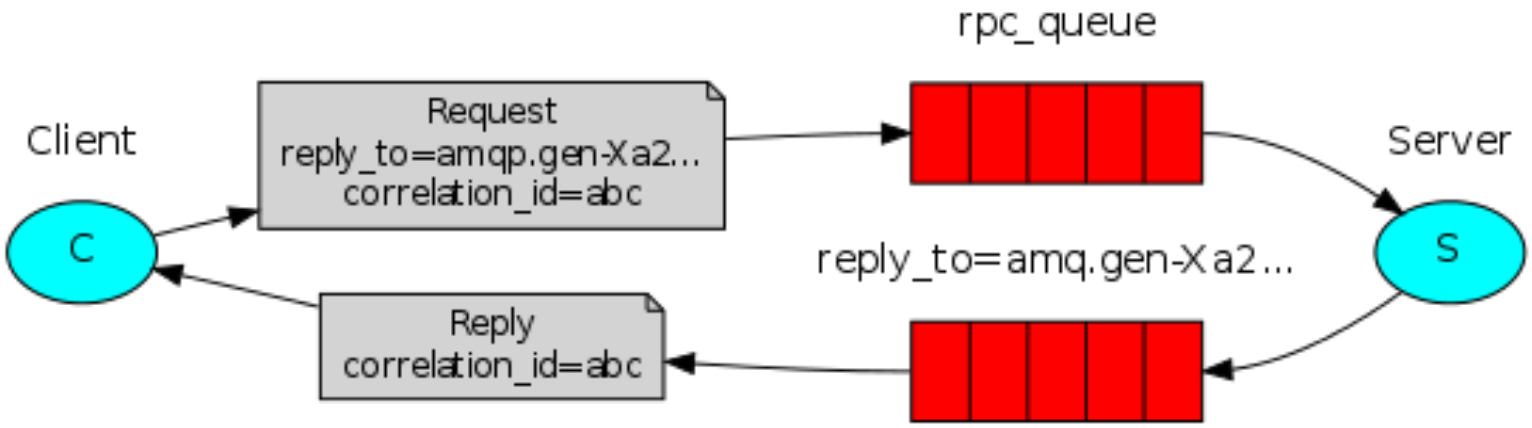
Content-based routing



ALGUNS USOS DE SISTEMAS DE MESSAGE-QUEUE

(Figures from: https://www.rabbitmq.com/tutorials/tutorial-three-java.html)

Invocação remota (c/ duas queues)



RabbitMQ

MOM que implementa o protocolo AMPQ 1.0

AMPQ (Advanced Message Queuing Protocol) é um protocolo aberto e standardizado;

A arquitetura é cliente/servidor, com os servidores escritos em Erlang;

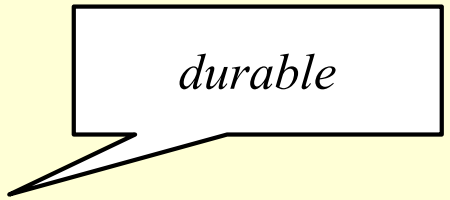
Clientes para diversas linguagens: Java, Python, PHP, Go, etc.

O transporte usa um formato binário, sobre o qual se pode usar, por exemplo JSON.

Set-up queues

```
ConnectionFactory factory = new ConnectionFactory();
Connection conn = factory.newConnection("localhost");
Channel channel = conn.createChannel();

channel.queueDeclare("my_queue", true);
```



RabbitMQ : Consumer/Receiver

```
ConnectionFactory factory = new ConnectionFactory();
factory.setHost("localhost");
Connection connection = factory.newConnection();

Channel channel = connection.createChannel();

channel.queueDeclare(QUEUE_NAME, false, false, false, null);

String message = "Hello World!";
channel.basicPublish("", QUEUE_NAME, null, message.getBytes());

channel.close();
connection.close();
```

RabbitMQ : Consumer/Receiver

```
ConnectionFactory factory = new ConnectionFactory();
factory.setHost("localhost");
Connection connection = factory.newConnection();
Channel channel = connection.createChannel();

channel.queueDeclare(QUEUE_NAME, false, false, false, null);

Consumer consumer = new DefaultConsumer(channel) {
   public void handleDelivery(String cTag, Envelope env, AMQP.BasicProperties props, byte[] body) throws IOException {
      …
   }
};

for(;;)
    channel.basicConsume(QUEUE_NAME, true, consumer);

// channel.close();
// connection.close();
```

SISTEMA: COMO IMPLEMENTAR?

Sistema de disseminação de informação da bolsa

SISTEMAS: COMO IMPLEMENTAR?

Suponham que têm um backbone de caches de internet, uma por ISP em cada país

(nota: por exemplo, a akamai tem esta infraestrutura)

Como actualizar as caches?

==========

Sistema de controlo de tráfego áereo

Radares monitorizam espaço aéreo

Quem está interessado na informação? Controlo de tráfego aérea, companhias, aeroportos

==========

Editor de texto cooperativo

==========

Sistema de disseminação de informação da bolsa

PARA SABER MAIS

G. Coulouris, J. Dollimore, T. Kindberg, G. Blair, Distributed Systems –Concepts and Design, Addison-Wesley, 5th Edition, 2011

capítulo 6.1-6.4, 15.4