Comunicação

Santiago Azevedo Robles

setembro/2019

METROCAMP | DeVry Brasil

Elementos básicos de comunicação

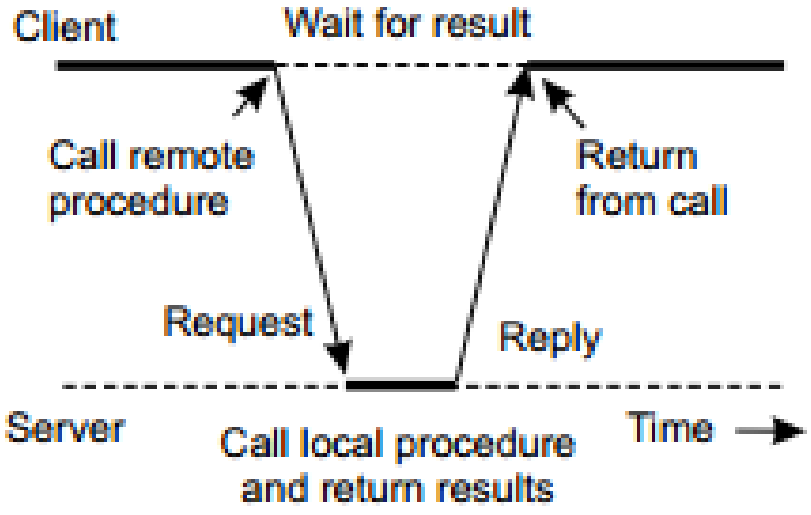
- Transmissão de dados
- Endereçamento
- Sincronismo
- Enfileiramento (Bufferização)
- Confiabilidade

Tipos de Comunicação

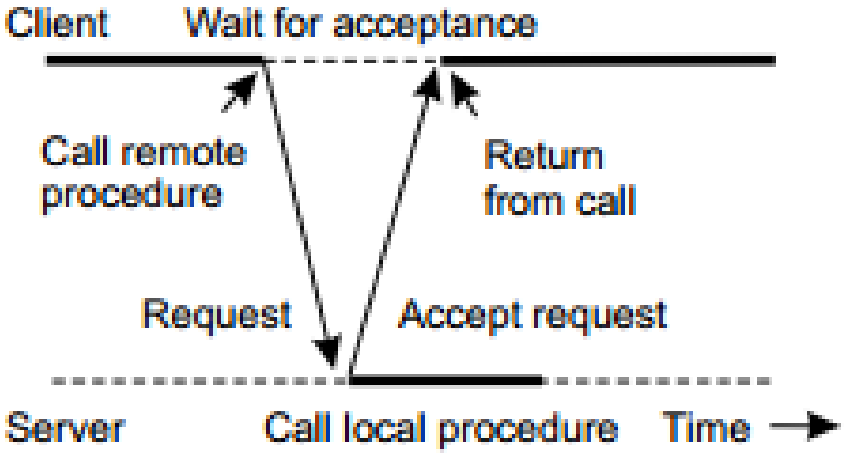
- síncrona
  - processos remetente e destino são sincronizados a cada mensagem
  - operação de *send* causa bloqueio
    - processo remetente desbloqueia apenas quando:
      - a mensagem é passada ao middleware
      - a mensagem é recebida pelo processo destino
      - a mensagem é processada pelo destino que então retorna uma resposta
- assíncrona
  - operação *send* é não-bloqueante
    - mensagem é copiada para um buffer local
    - transmissão em paralelo com o processamento no remetente
  - operação *receive* bloqueante/não-bloqueante
    - processo bloqueado enquanto mensagem não chega
    - processo continua execução podendo ser notificado (p.ex. por uma interrupção) da chegada de uma mensagem

Tipos de Comunicação

- persistente
  - a mensagem transmitida é armazenada pelo sistema de comunicação até que ela seja entregue ao receptor
  - não é necessário que a aplicação transmissora continue executando após ter transmitido a mensagem e a aplicação receptora não precisa estar executando quando da transmissão da mensagem
  - ex. sistema de correio eletrônico
- transiente
  - a mensagem é armazenada pelo sistema de comunicação somente enquanto as aplicações transmissora e receptora estão executando
  - ex. serviços de comunicação no nível de transporte

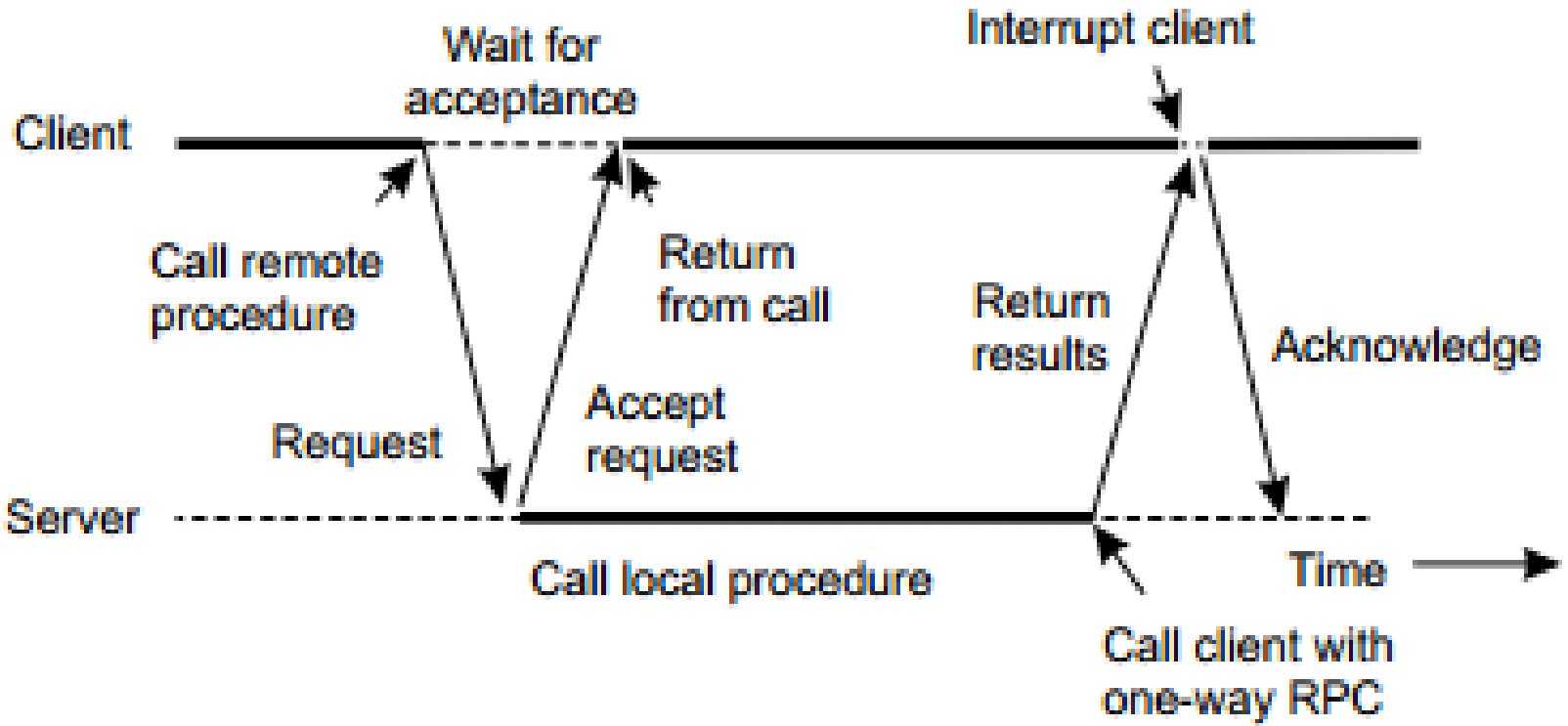

Client
Wait for result
Call remote procedure
Return from call
Request
Reply
Server
Call local procedure and return results
Time

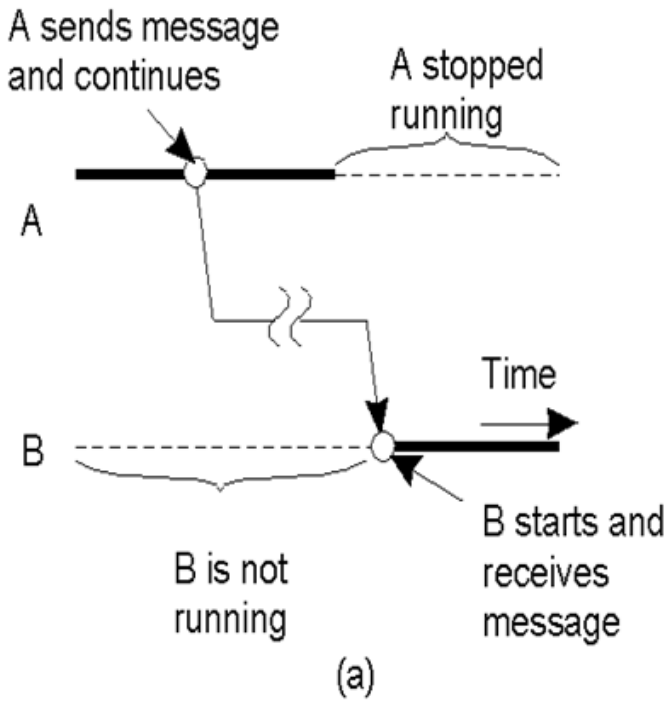
(a)

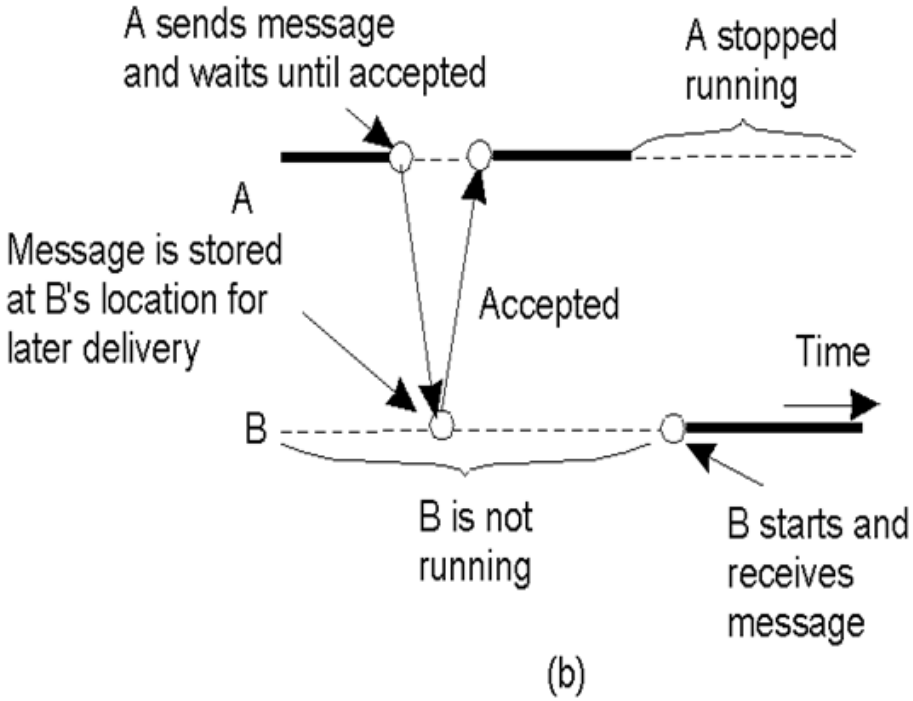

Client
Wait for acceptance
Call remote procedure
Return from call
Request
Accept request
Server
Call local procedure
Time

(b)

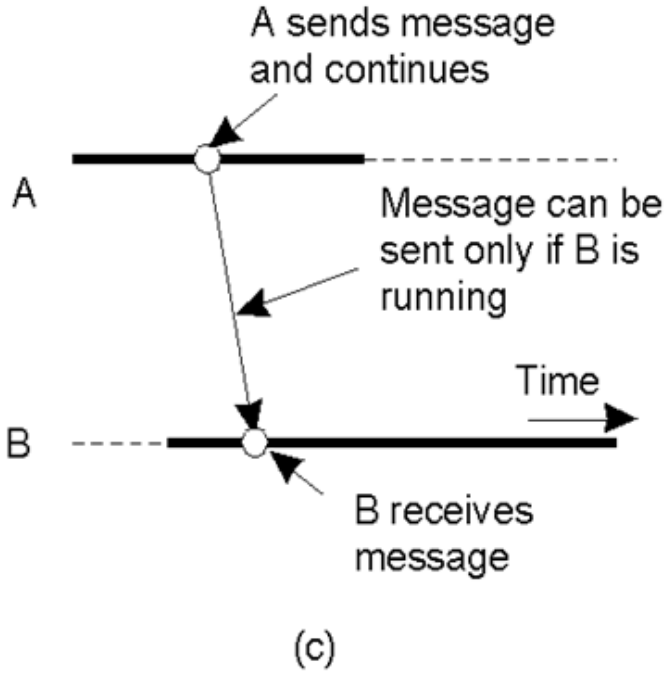
Assincrona


Wait for acceptance
Interrupt client
Client
Call remote procedure
Return from call
Return results
Acknowledge
Request
Accept request
Server
Call local procedure
Time
Call client with one-way RPC

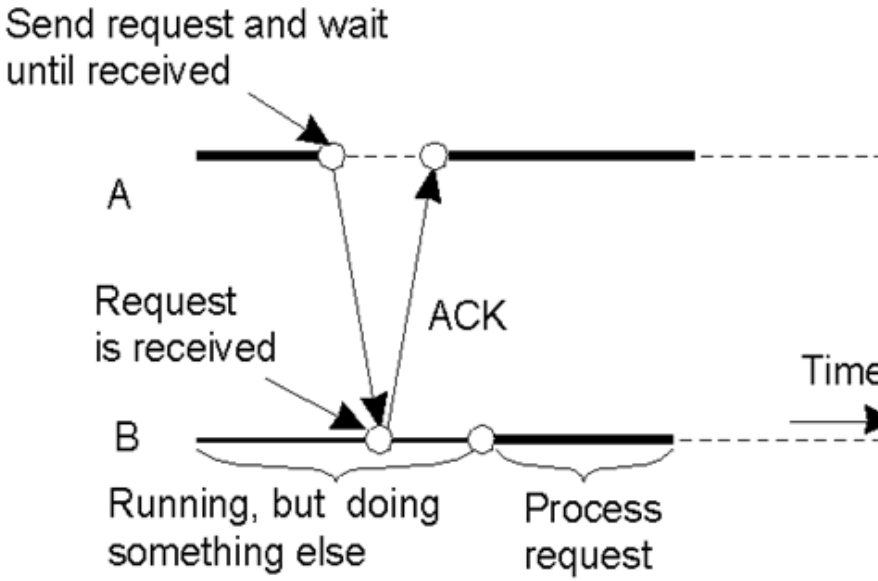




a) Persistent asynchronous communication
b) Persistent synchronous communication



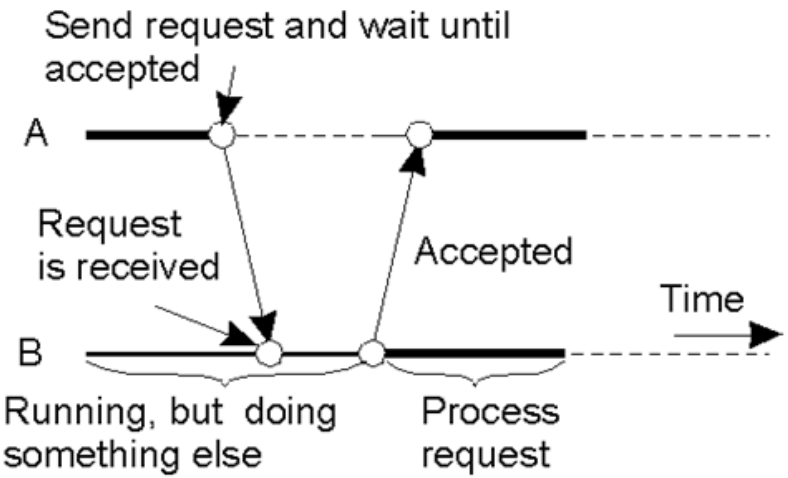
(c)



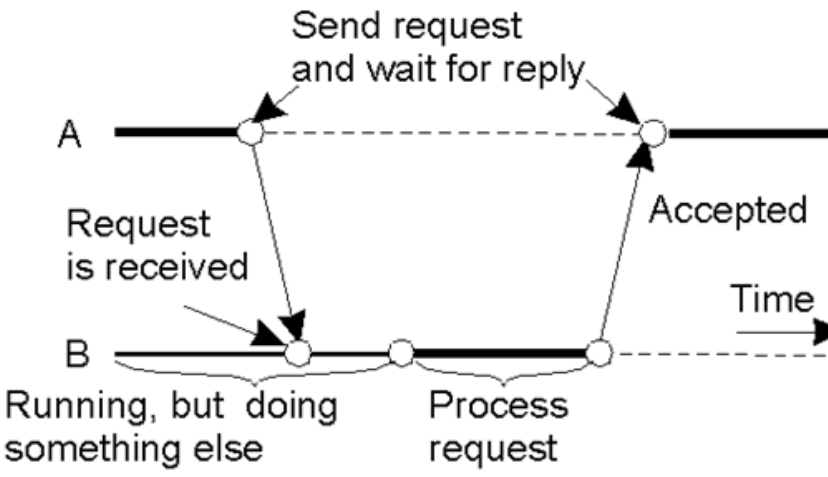
(d)

c) Transient asynchronous communication

d) Receipt-based transient synchronous communication



(e)



(f)

e) Delivery-based transient synchronous communication at message delivery

f) Response-based transient synchronous communication

Endereçamento

- Esquemas:
  - Endereçamento máquina.processo
  - Endereçamento máquina.id-local
  - Descoberta de endereço via broadcasting (difusão)
  - Descoberta de endereço via um servidor de nomes

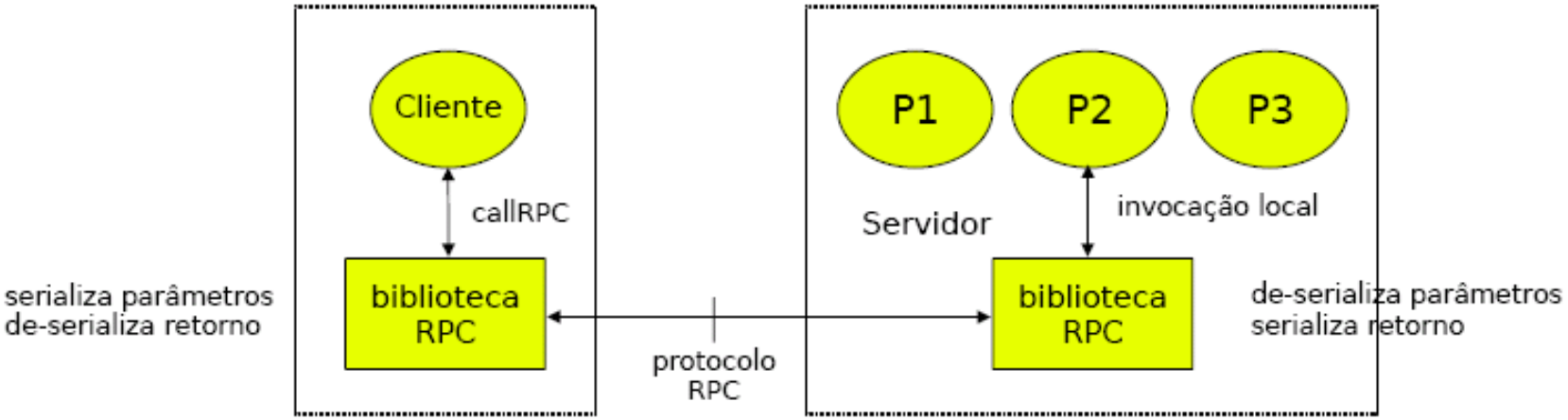
Problemas potenciais: transparência de localização, sobrecarga, escalabilidade

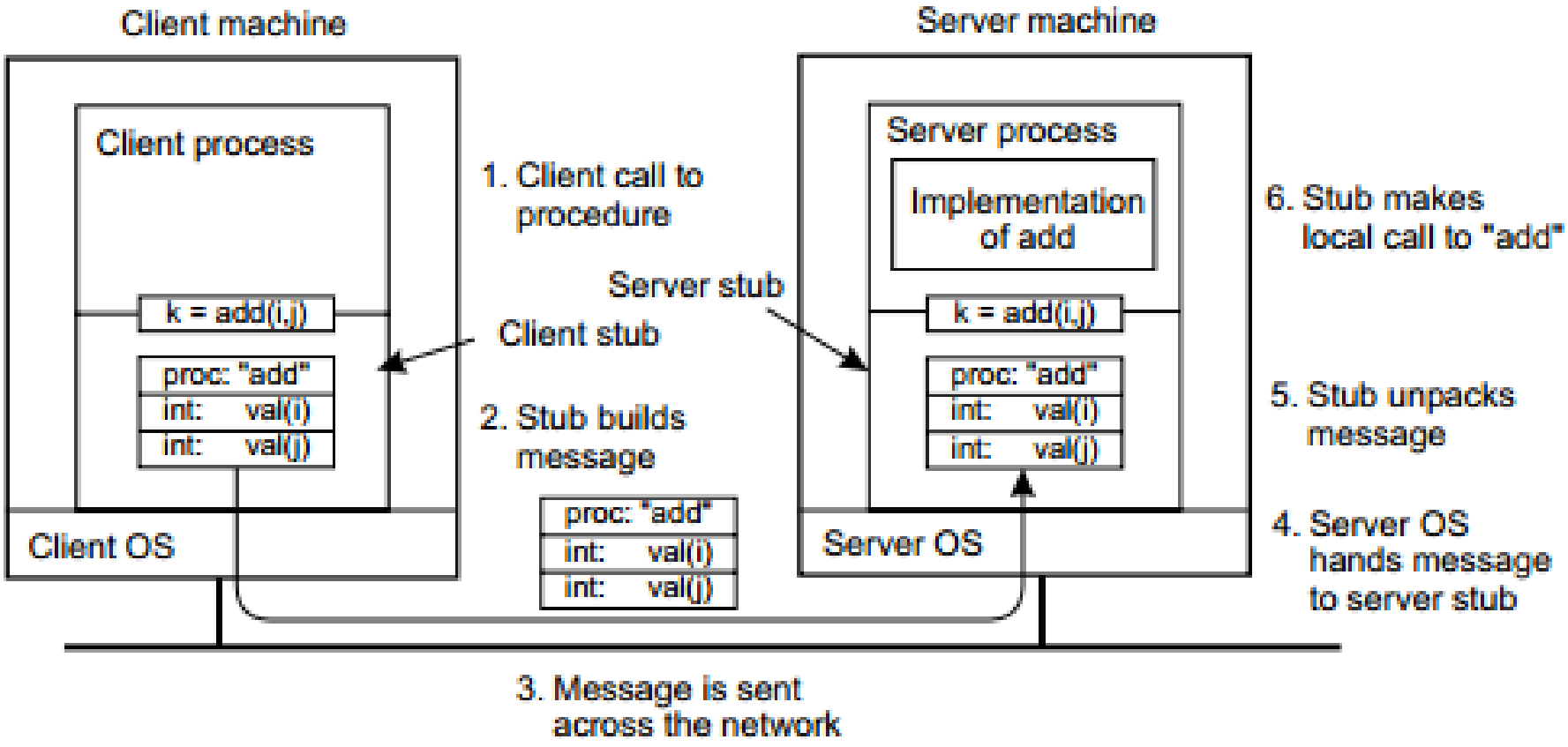
Confiabilidade

- Mensagens se perdem, atrasam, duplicam.
- Abordagens:
  - Send tem semântica não confiável: as aplicações devem garantir entrega de mensagens (ex: timeout)
  - Mensagem de acknowledgement enviada pelo servidor (no nível núcleo)
  - Mensagem de acknowledgement implícita na resposta do servidor

Remote Procedure Call (RPC)

- *Protocolo* de interação (comunicação) entre cliente/servidor;
- *Servidor* suporta conjunto de sub-rotinas invocadas dinâmicamente;
- passagem de parâmetros e retorno é sempre por valor;
- necessita de infra-estrutura para acesso remoto;
- Ex - infra- estrutura miníma:




Client machine
Server machine
Client process
Server process
Implementation of add
k = add(i,j)
k = add(i,j)
proc: "add"
int: val(i)
int: val(j)
proc: "add"
int: val(i)
int: val(j)
1. Client call to procedure
Server stub
Client stub
2. Stub builds message
proc: "add"
int: val(i)
int: val(j)
6. Stub makes local call to "add"
5. Stub unpacks message
4. Server OS hands message to server stub
Client OS
Server OS
3. Message is sent across the network

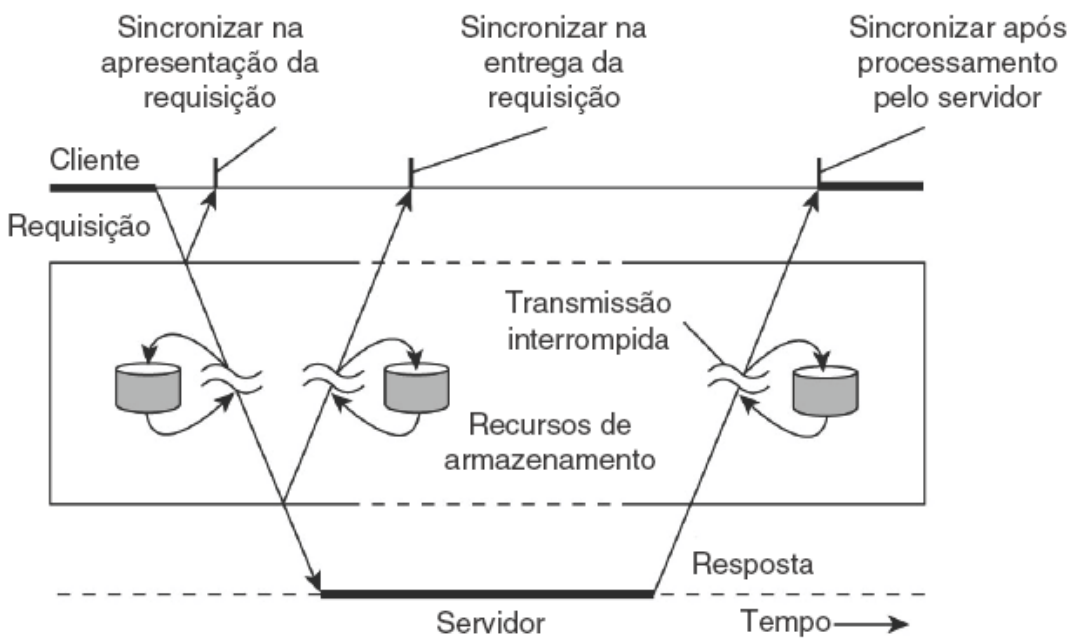
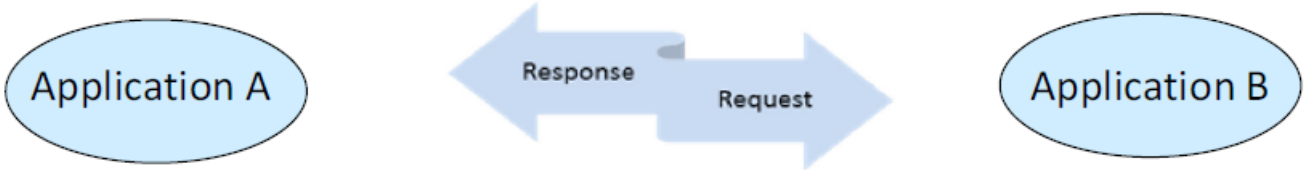
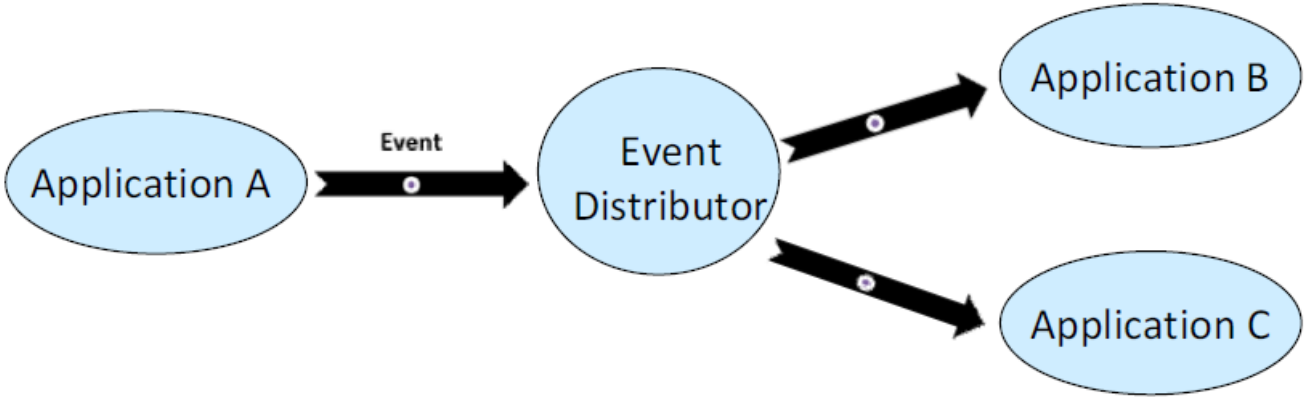


Figura 4.4 *Middleware visto como serviço intermediário (distribuído) na comunicação de nível de aplicação.*

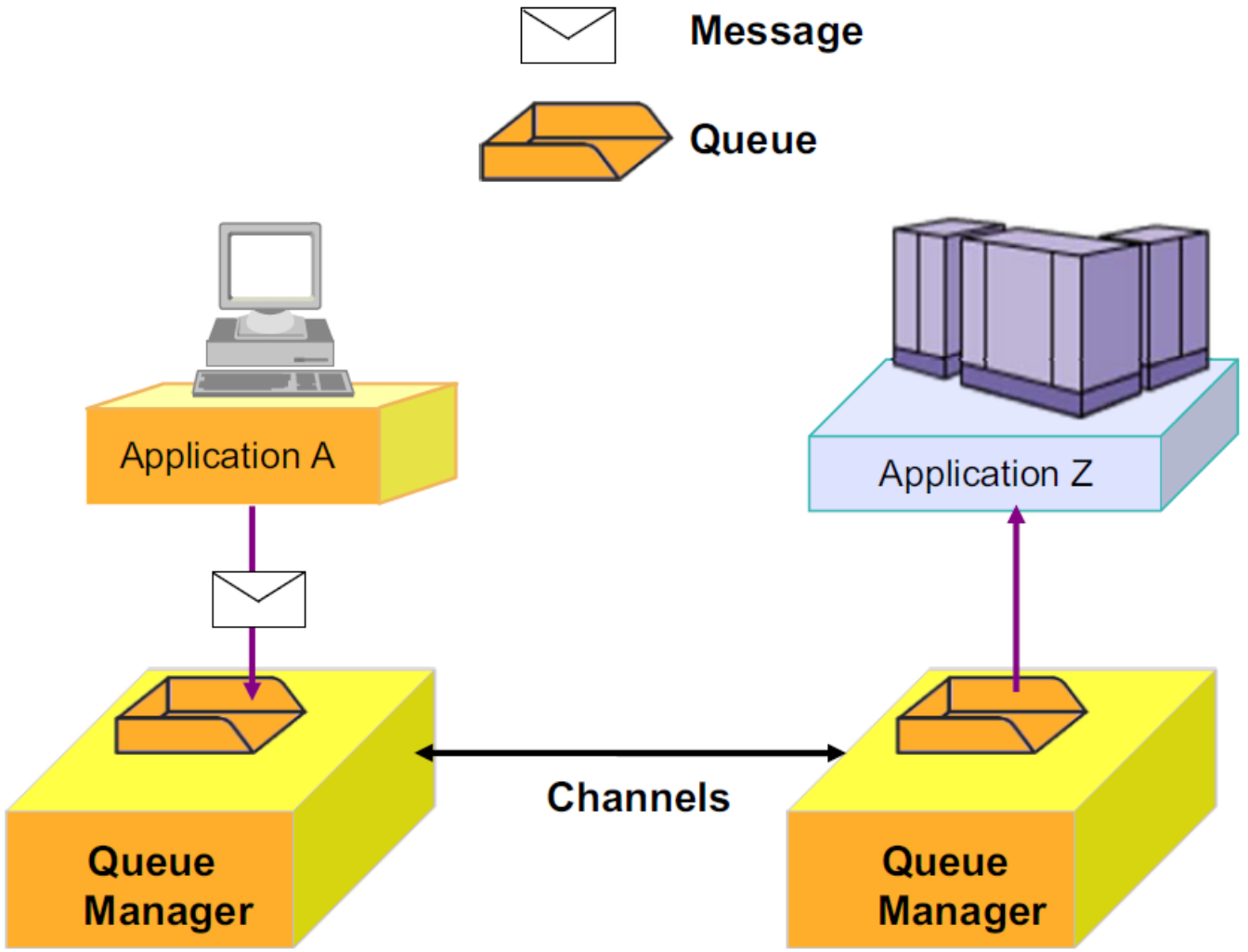
Application to application communication



Event distribution on any scale

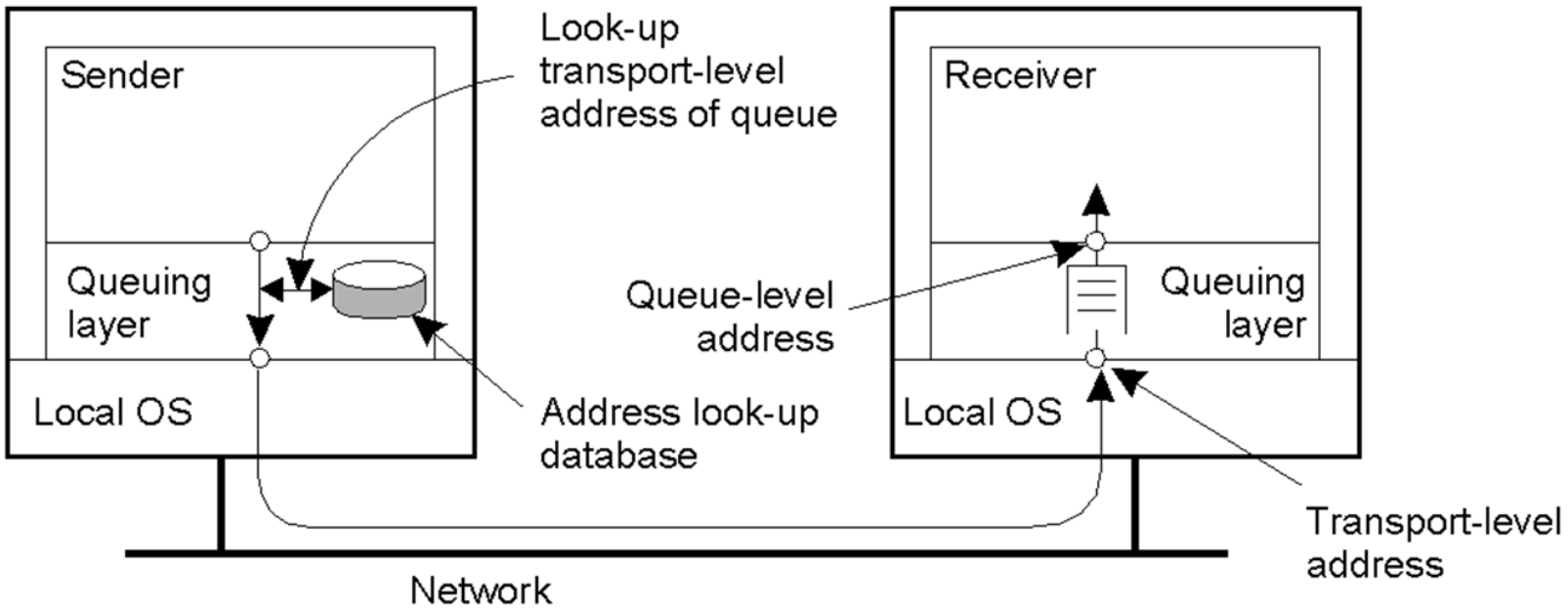


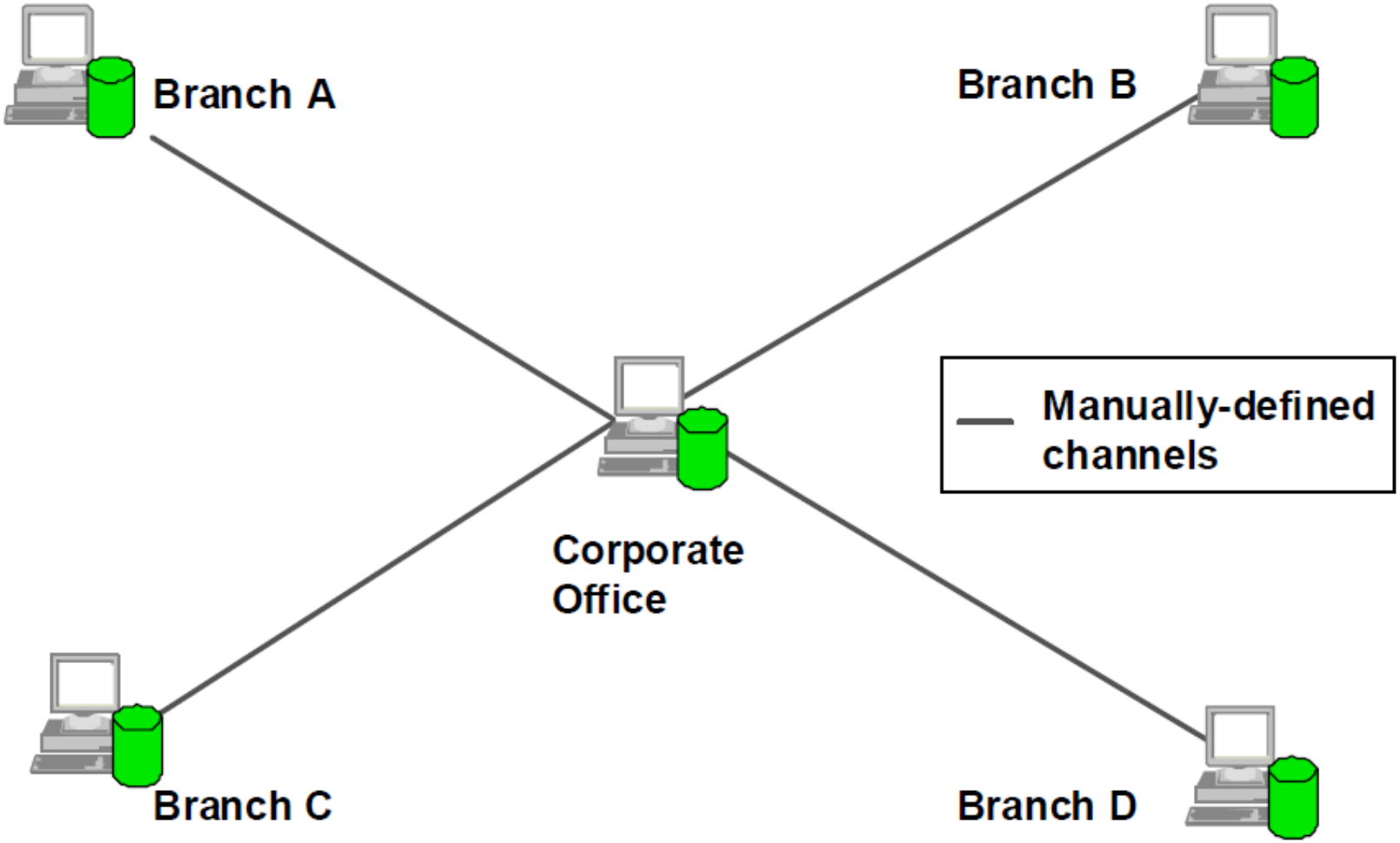
Comunicação por mensagens

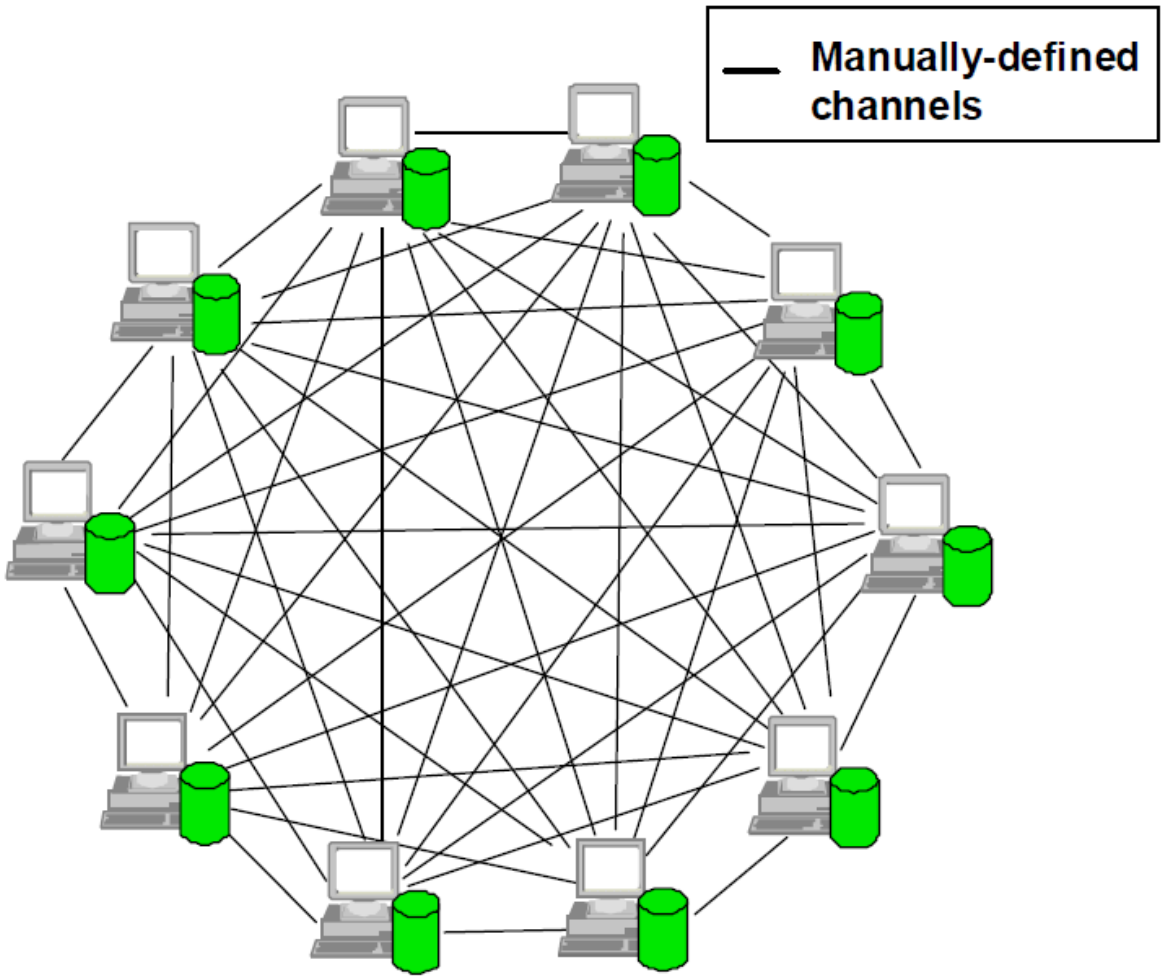


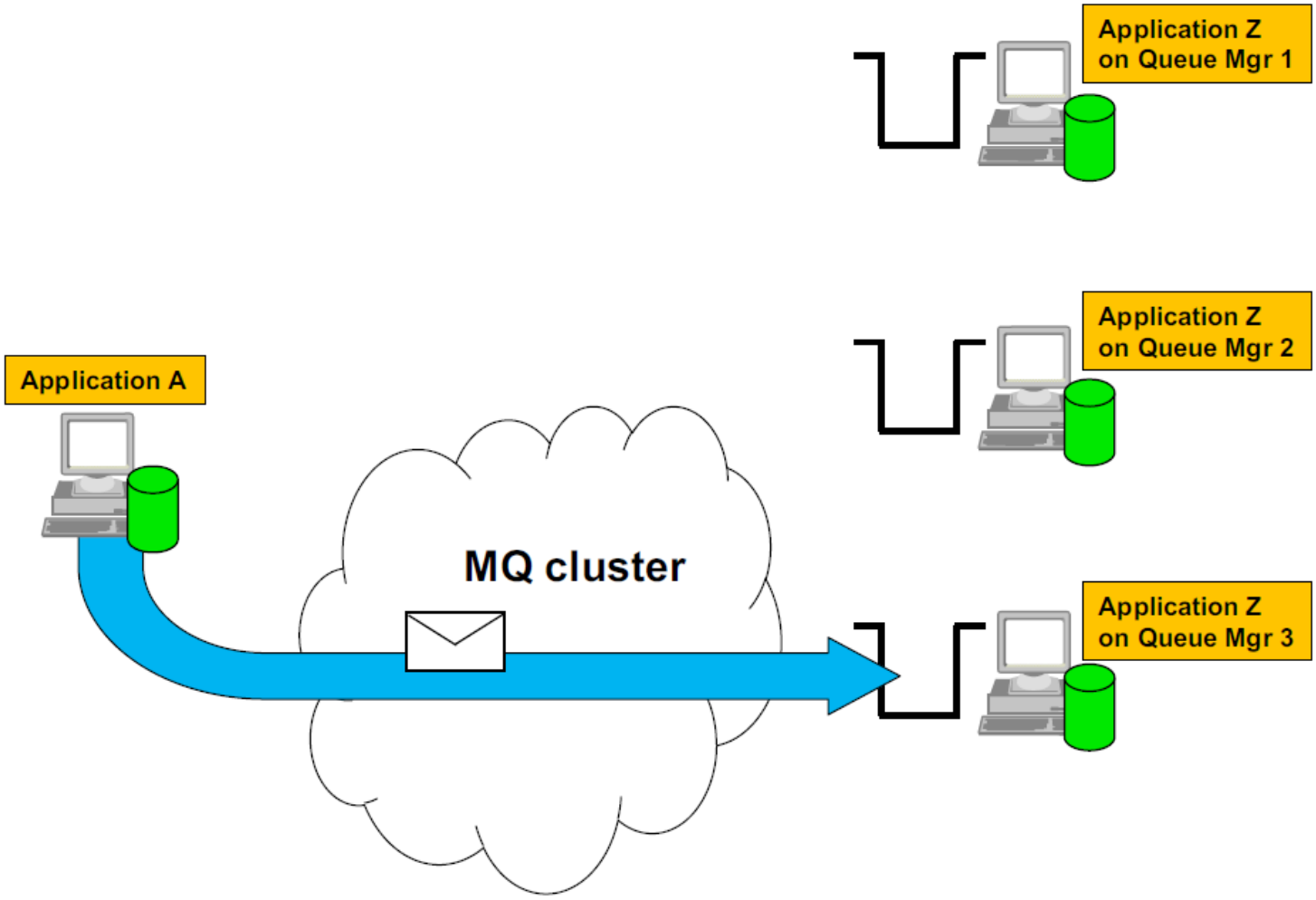
Comunicação por mensagens

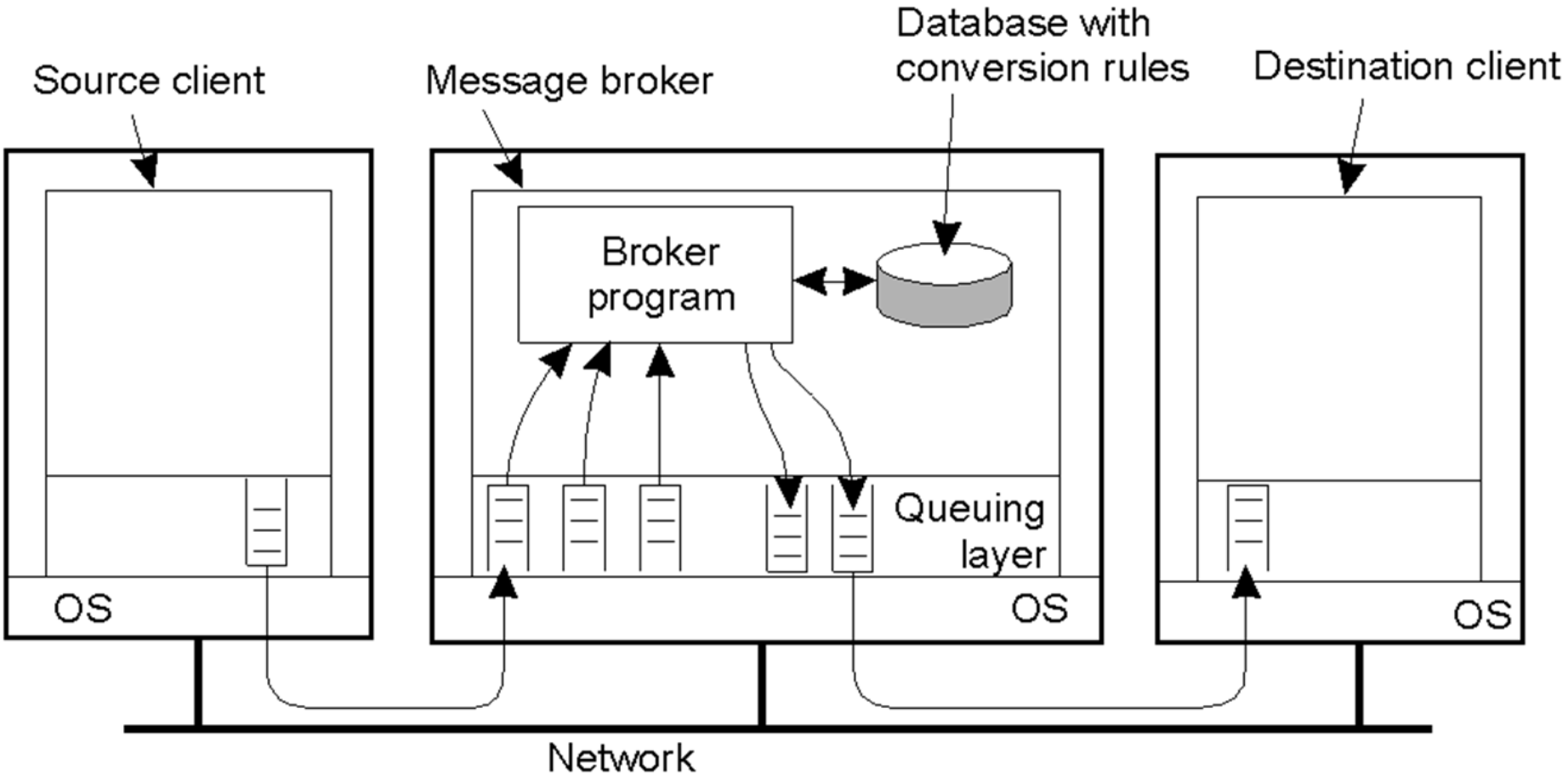
- Interface simples
- Foco nas “regras de negócio”
- Fraco acoplamento
- Código menos complexo
- Assíncrono (disponibilidade)
- Grande volume de dados
- Escalabilidade


Sender
Look-up
transport-level
address of queue
Receiver
Queuing
layer
Queue-level
address
Queuing
layer
Local OS
Address look-up
database
Local OS
Transport-level
address
Network


Branch A
Branch B
Corporate Office
Manually-defined channels
Branch C
Branch D


Manually-defined channels


Application Z
on Queue Mgr 1
Application Z
on Queue Mgr 2
Application A
MQ cluster
Application Z
on Queue Mgr 3


Source client
Message broker
Database with
conversion rules
Destination client
Broker
program
Queuing
layer
OS
OS
OS
Network

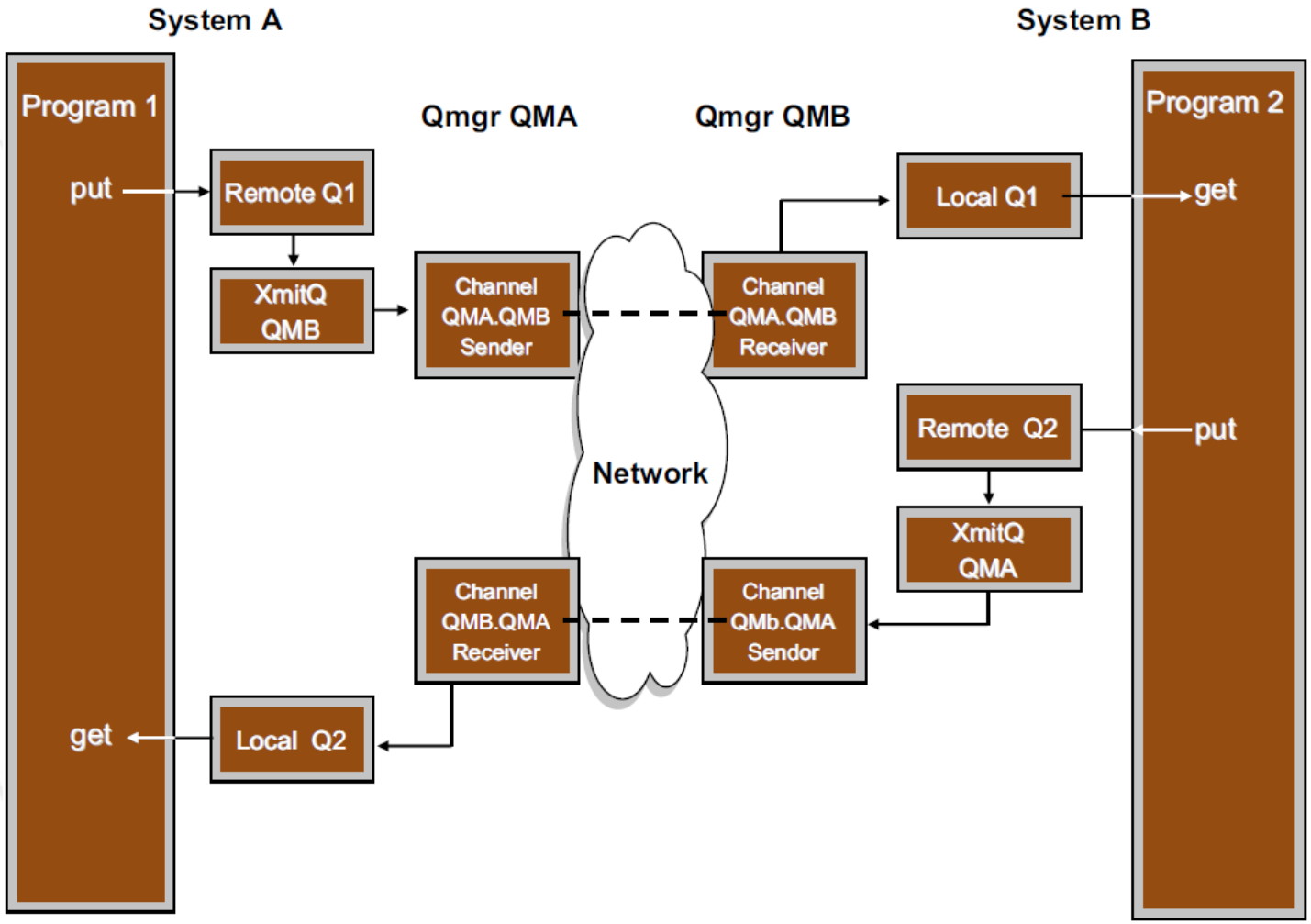
Primitive	Meaning
Put	Append a message to a specified queue
Get	Block until the specified queue is nonempty, and remove the first message
Poll	Check a specified queue for messages, and remove the first. Never block.
Notify	Install a handler to be called when a message is put into the specified queue.

Messages

A *message* is a container that consists of three parts:

- WebSphere MQ Message Descriptor (MQMD): Identifies the message and contains additional control information, such as the type of message and the priority that is assigned to the message by the sending application.
- Message properties: An optional set of user-definable elements that describes the message without being part of the payload. Applications that receive messages can choose whether to inspect these properties.
- Message data: Contains the application data. The structure of the data is defined by the application programs that use it; WebSphere MQ is largely unconcerned with its format or content.

An individual message can contain up to 100 MB of data. There are no constraints on the format of this data. It can be readable text, binary, XML, or any combination.


System A
System B
Program 1
Qmgr QMA
Qmgr QMB
Program 2
put
Remote Q1
Local Q1
get
XmitQ
QMB
Channel
QMA.QMB
Sender
Channel
QMA.QMB
Receiver
Remote Q2
put
Network
XmitQ
QMA
Channel
QMB.QMA
Receiver
Channel
QMb.QMA
Sendor
get
Local Q2

Amazon Simple Queue Service

Fully managed message queues for microservices, distributed systems, and serverless applications

http://aws.amazon.com/sqs/

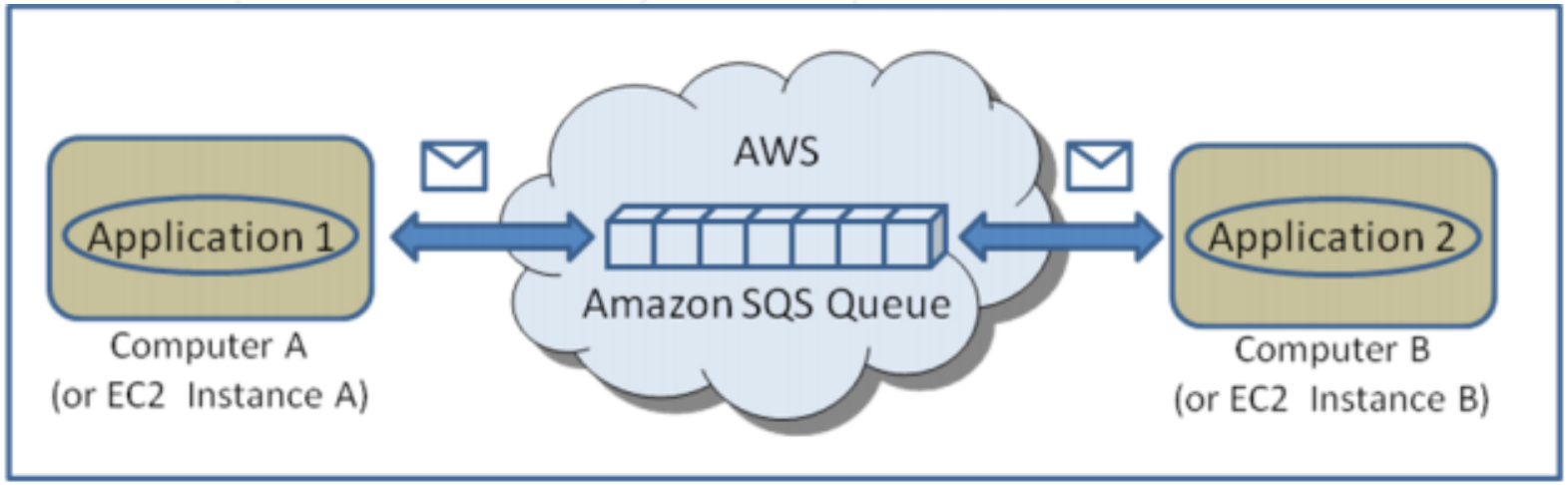


Figure 1: Two distributed applications communicating asynchronously by passing messages through an Amazon SQS queue.

IBM

https://developer.ibm.com/messaging/ibm-mq/

What IBM MQ can do for your business

Make every message count

Some messaging solutions deliver messages twice, or not at all. What does that mean to you when a single message contains critical information? IBM MQ delivers messages once and only once.

Work with all your existing applications

Time spent hardcoding applications is time lost. You shouldn't have to amend applications so they communicate. With IBM MQ, disparate applications can communicate with ease — no code changes needed.

Offer security-rich messaging, everywhere

Working on the mainframe? With appliances? Software? On one cloud or many? Wherever you need to connect — from mainframe to mobile, on premises and across clouds — IBM MQ is the one solution you need.

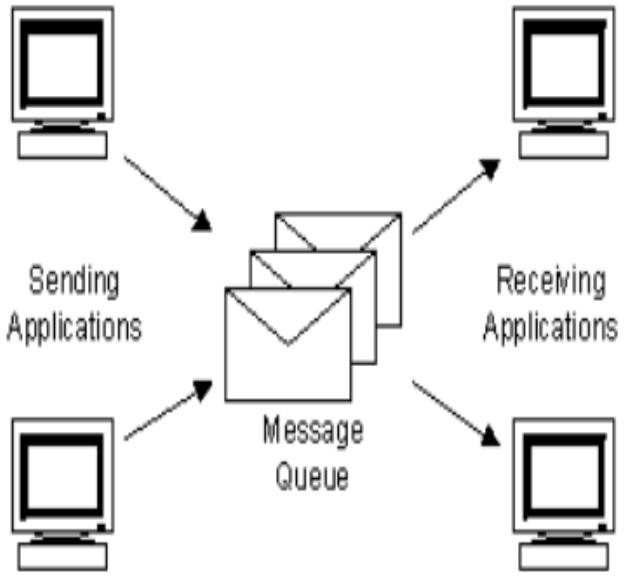
Support the environment you have today, and the one you'll have tomorrow

Are all your applications written in the same language and designed to work together? No? IBM MQ enables you to use new and existing applications, systems and services together. No rip and replace.

Microsoft Message Queuing (MSMQ)

https://msdn.microsoft.com/en-us/library/ms711472.aspx

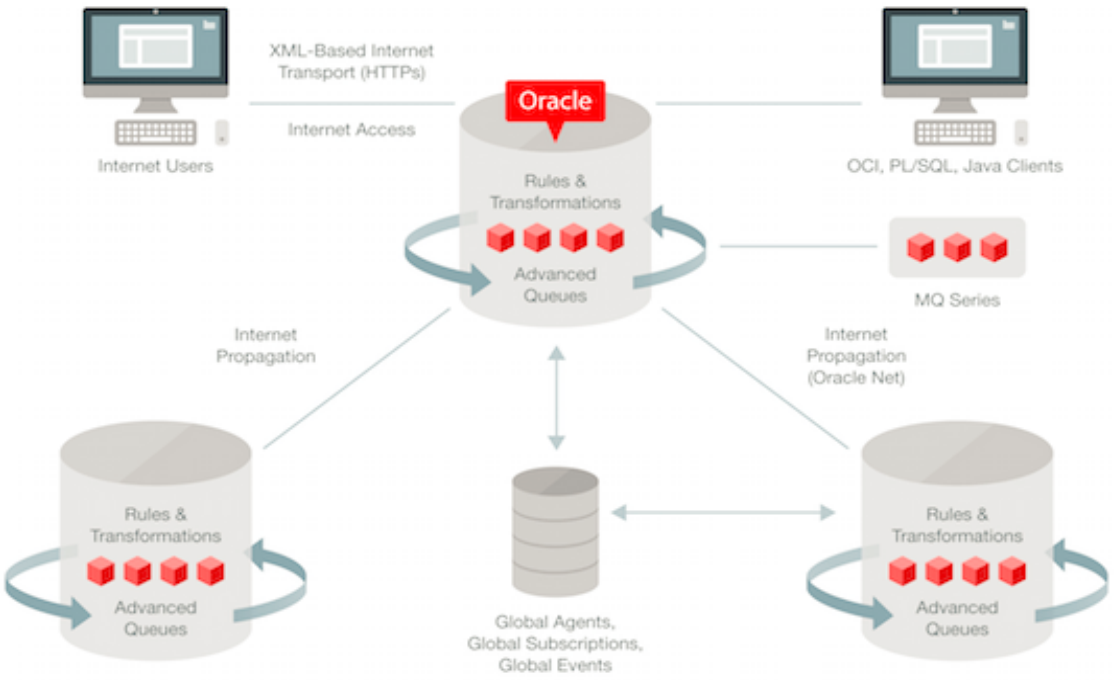
Message Queuing (MSMQ) technology enables applications running at different times to communicate across heterogeneous networks and systems that may be temporarily offline. Applications send messages to queues and read messages from queues. The following illustration shows how a queue can hold messages that are generated by multiple sending applications and read by multiple receiving applications.



http://www.oracle.com/technetwork/database/database-technologies/advanced-queuing/index.html

Oracle Database Advanced Queuing

Message Queuing in Oracle Database 12*c*

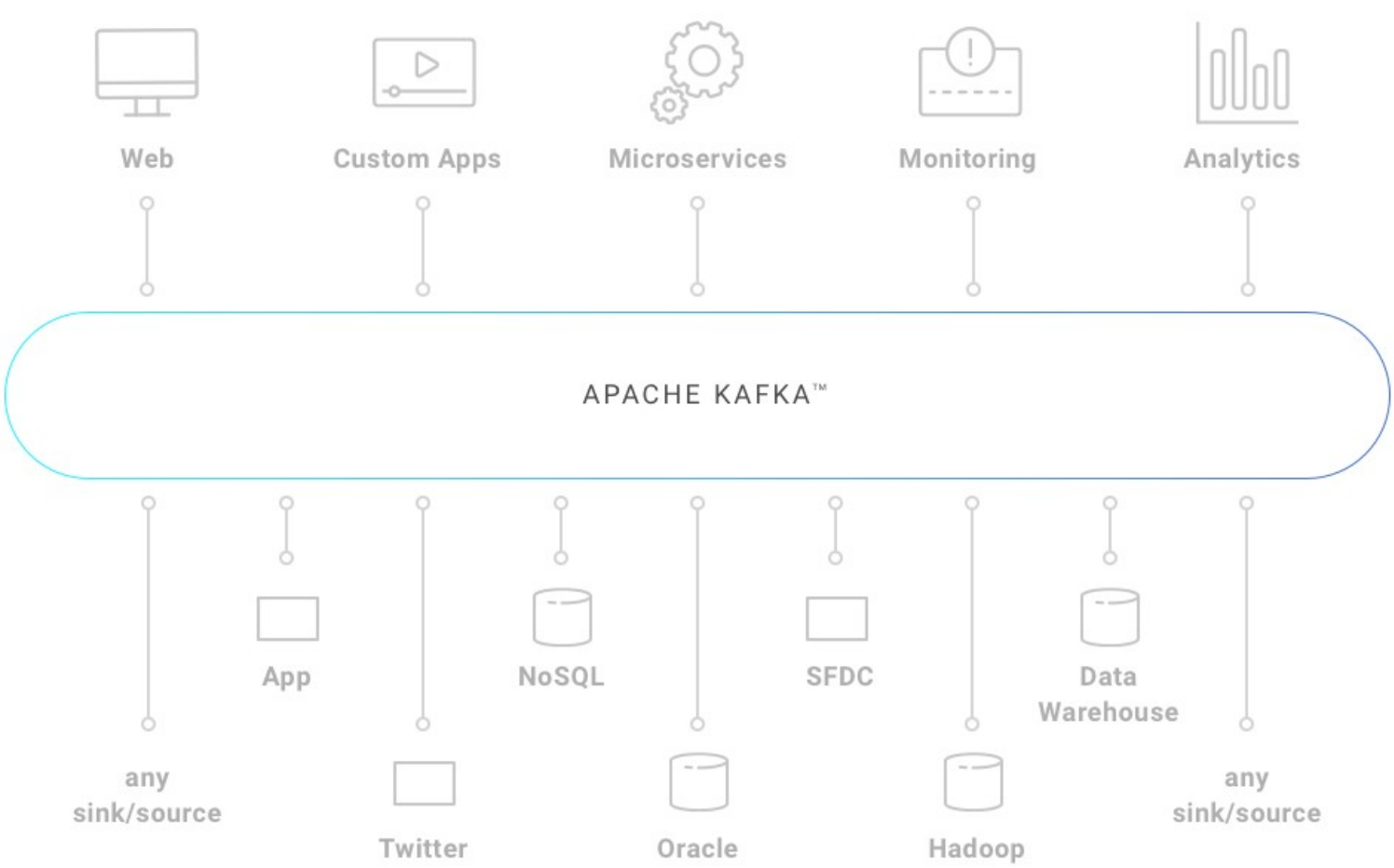


Oracle Advanced Queuing (AQ) provides enterprise message functionality across many industries and is widely used within the database itself. AQ supports both persistent and non-persistent messages, and AQ queues can be set up under different queuing models such as point-to-point and publish-subscribe to let business applications communicate with each other flexibly and reliably.

AQ asynchronously notifies interested clients about messages and database events. Messages can be asynchronously propagated between queues on the same or

Apache Kafka

Apache Kafka



RabbitMQ

https://www.rabbitmq.com/