Distributed Systems
COMP 212

Lecture 15

Othon Michail

UNIVERSITY OF LIVERPOOL

RPC/RMI vs Messaging

- RPC/RMI great in hiding communication in DSs
- But in some cases they are inappropriate
  - What happens if we cannot assume that the receiving side is “awake” and waiting to communicate?
  - Also, the default “synchronous, blocking” nature of RPC/RMI is often too restrictive

In such cases, something else is needed:

- Messaging
  - Message-Oriented Middleware (MOM)

Message-Oriented Communication and Streams

Some DS Comms. Terminology

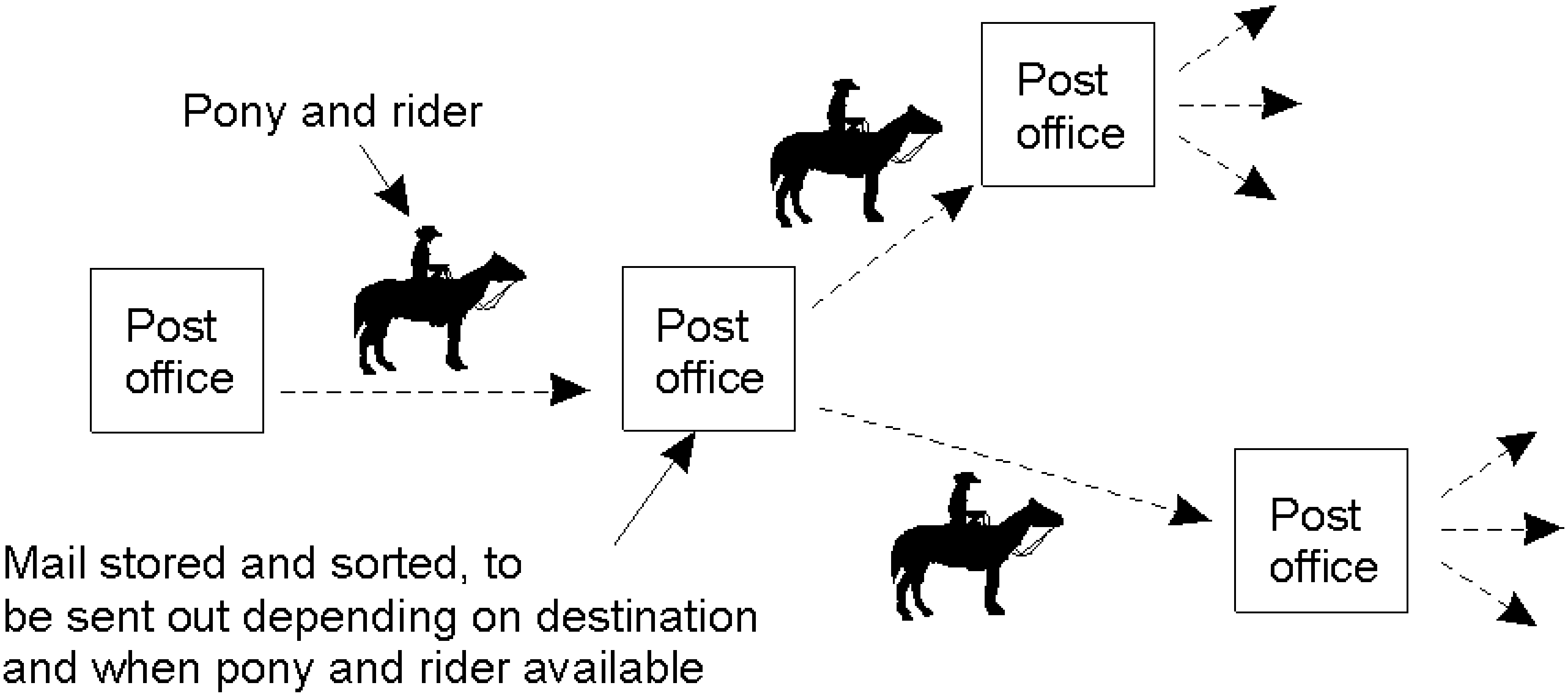
Persistent Communications:

- Once sent, the “sender” can stop executing. The “receiver” need not be operational at this time – the communications system buffers the message as required (until it can be delivered).
- [Can you think of an example?]

Contrast to Transient Communications:

- The message is only stored as long as the “sender” and “receiver” are executing. If problems occur, the message is simply discarded …

Persistence and Synchronicity in Communication



- Persistent communication of letters back in the days of the Pony Express

More DS Comms. Terminology

Asynchronous Communications:

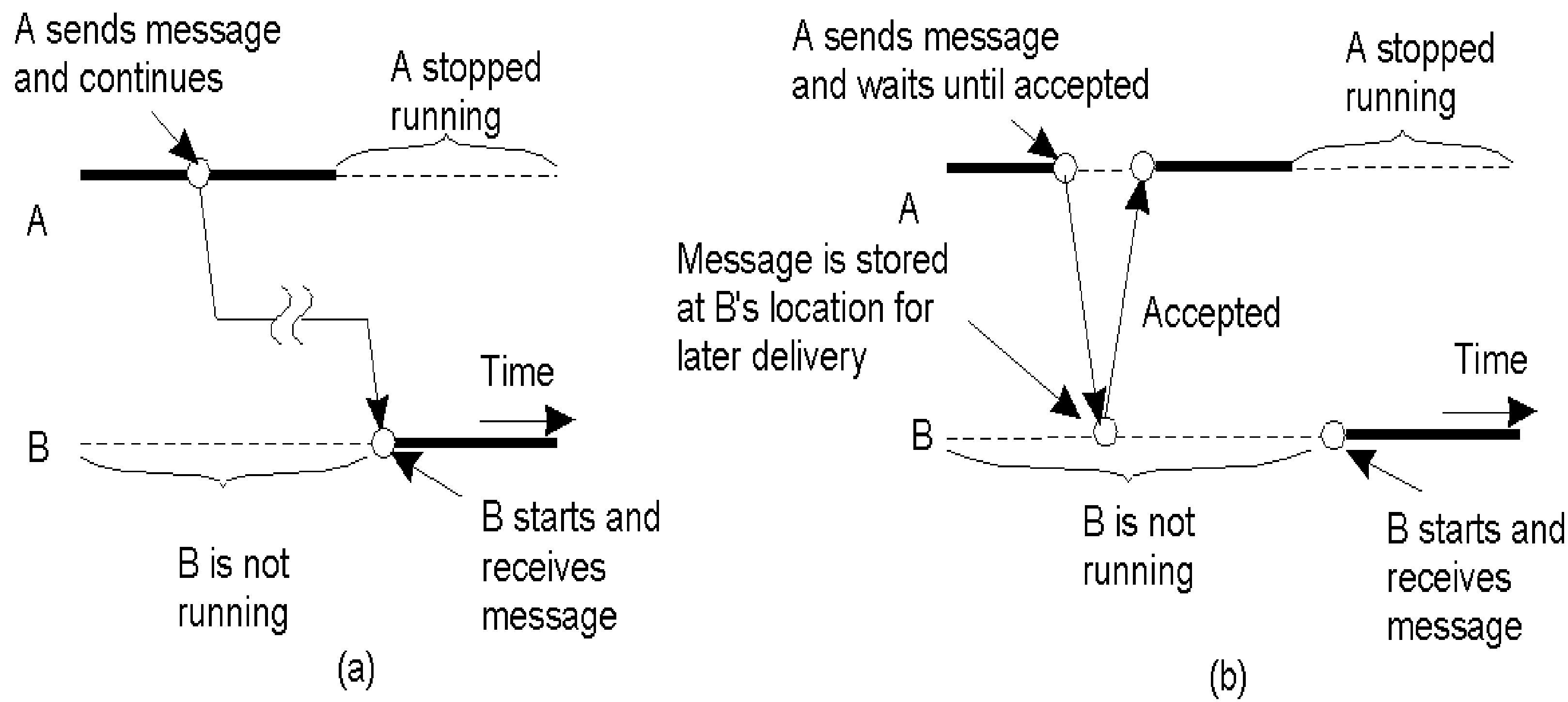
- A sender continues with other work immediately upon sending a message to the receiver

Synchronous Communications:

- A sender blocks, waiting for a reply from the receiver before doing any other work
  - This tends to be the default model for RPC/RMI technologies

- We will have a look now at combinations

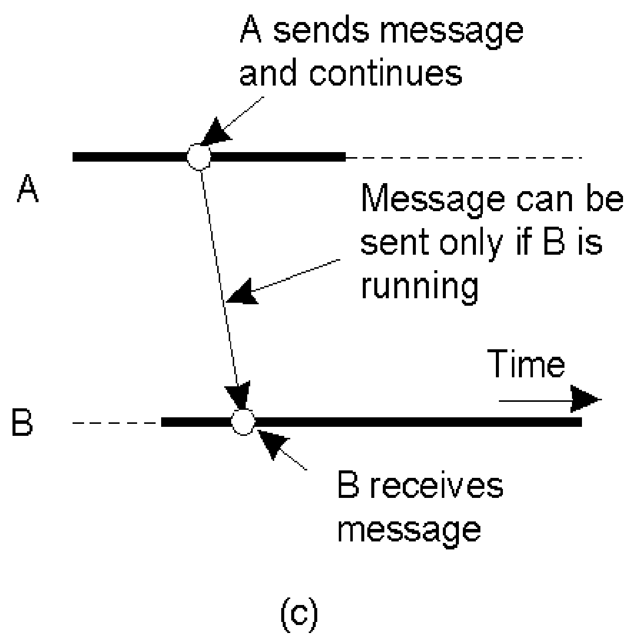
Classifying Distributed Communications (1)

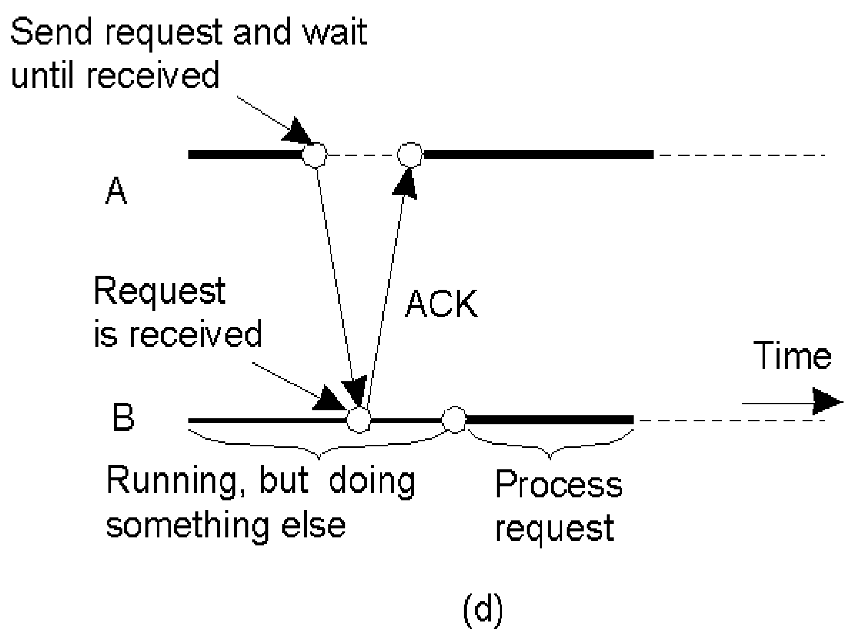


a) Persistent asynchronous communication

b) Persistent synchronous communication

Classifying Distributed Communications (2)

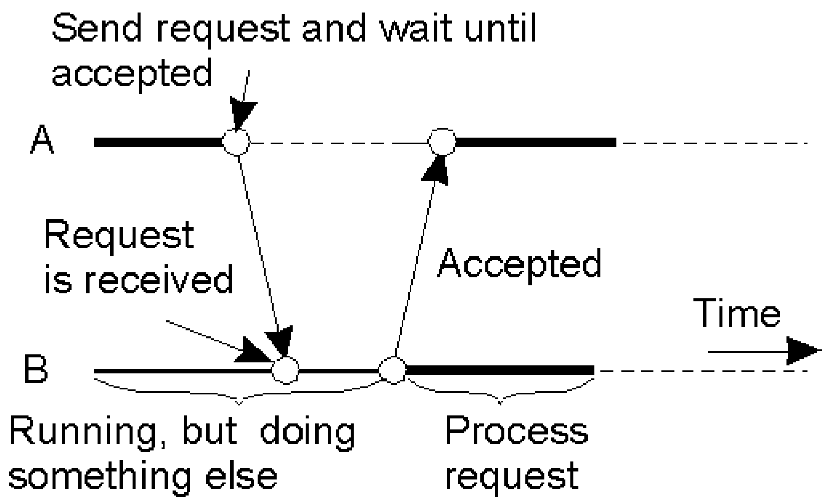




c) Transient asynchronous communication

d) Receipt-based transient synchronous communication

Classifying Distributed Communications (3)



e) Delivery-based transient synchronous communication at message delivery

f) Response-based transient synchronous communication

Example: Message-Passing Interface

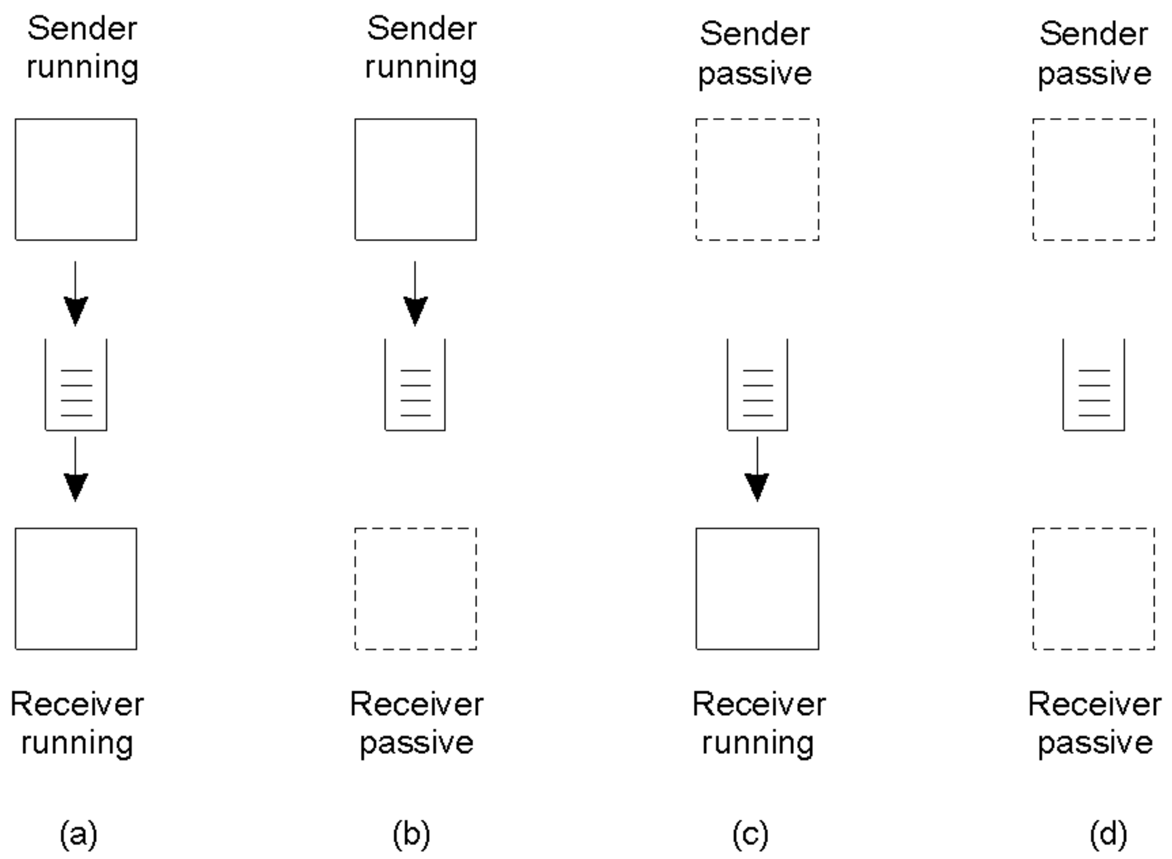
- For high-performance multicomputers
- Specific network protocols (not TCP/IP)
- Message-based communication
- Primitives for all 4 forms of transient communication (+ variations)
- Over 100 functions

- Vendors
  -- IBM, Intel, TMC, Meiko, Cray, Convex, Ncube

Message-Oriented Persistent Comms.

- Also known as: "message-queuing systems"
- They support persistent, asynchronous communications
- Typically, transport can take minutes (hours?) as opposed to seconds/milliseconds
- The basic idea: applications communicate by putting messages into and taking messages out of "message queues"
- Only guarantee: your message will eventually make it into the receiver's message queue
- This leads to "loosely-coupled" communications

Message-Queuing Models



- Four combinations for “loosely-coupled” communications which use message-queues

Simple Interface

- Messages can contain any data
- Only important aspect: Must be properly addressed
  - Unique names of destination queues
- The system takes care of fragmenting and assembling large messages in a way transparent to applications
  - Extremely simple interface offered to applications

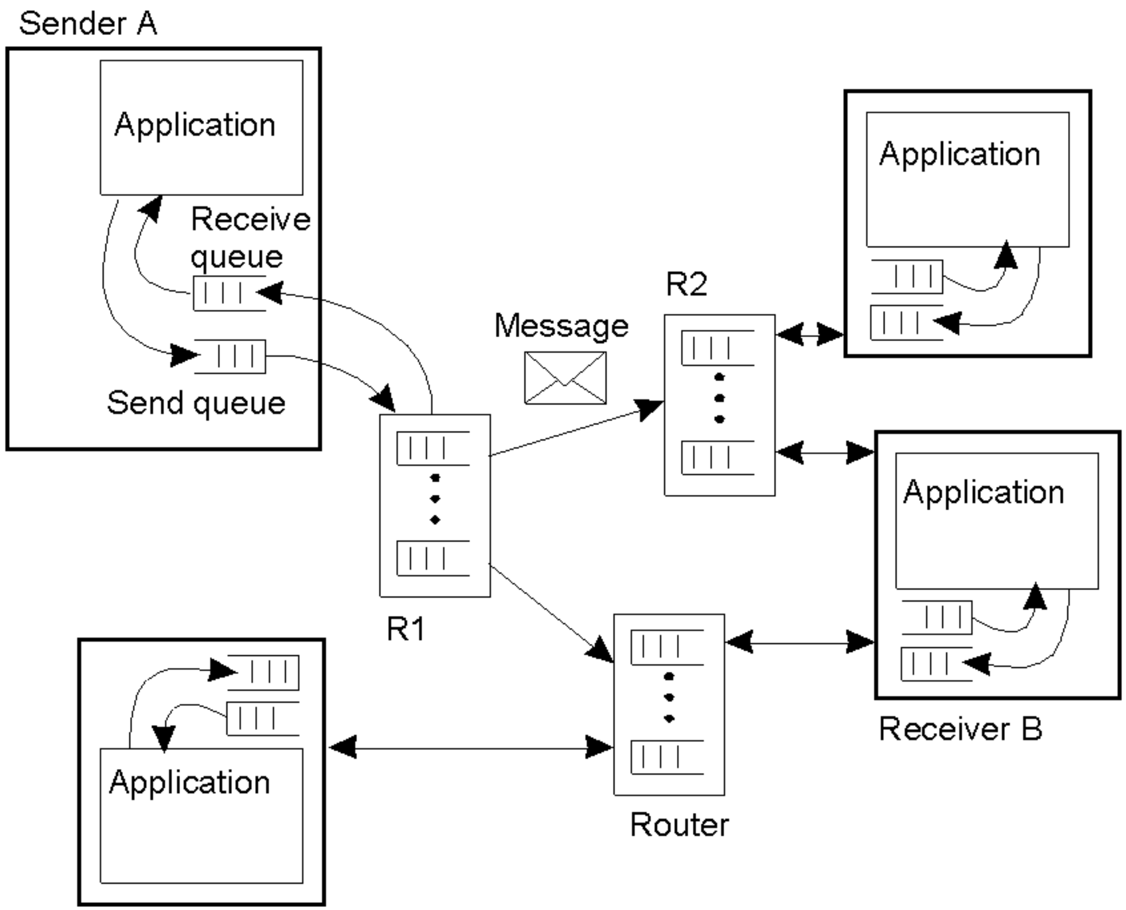
Primitive	Meaning
Put	Append a message to a specified queue
Get	Block until the specified queue is nonempty, and remove the first message
Poll	Check a specified queue for messages, and remove the first. Never block.
Notify	Install a handler to be called when a message is put into a specified queue

Message-Queuing System Architecture

- Messages are put into a source queue
- To be transferred to a destination queue
- Queues are distributed across multiple machines
  - The message-queuing system has to maintain a mapping of queues to network locations
  - Distributed database of queue names to network locations
  - e.g., email to somebody@liverpool.ac.uk
    - The mailing system queries DNS to find the network address of the recipient mail server to use for the actual message transfer

- A mechanism has to exist to move a message from a source queue to a destination queue
- This is the role of the Queue Manager
  - May interact directly with sending/receiving applications
  - May operate as relays, forwarding messages to other queue managers
  - May form a complete, application level, overlay network, on top of an existing computer network

General Architecture of a Message-Queuing System

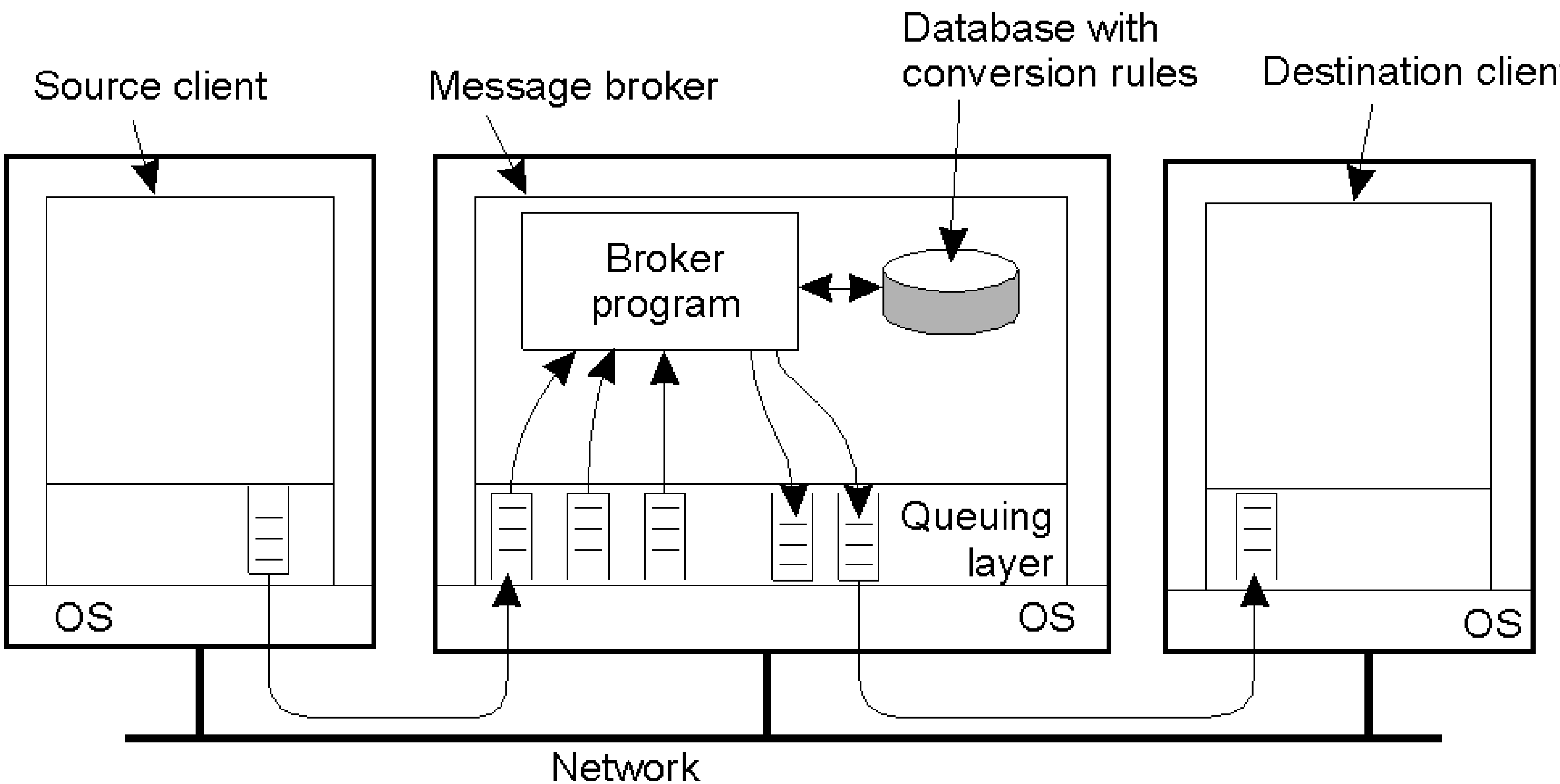


- The general organisation of a message-queuing system with routers
- The Queue Managers can reside within routers as well as within the DS end-systems

The Role of Message Brokers

- Often, there’s a need to integrate new/existing apps into a “single, coherent Distributed Information System (DIS)”
  - In other words, it is not always possible to start with a blank page
  - distributed systems have to live in the real world
- Problem: different message formats exist in legacy systems (cooperation and adherence to open standards was not how things were done in the past)
- It may not be convenient to “force” legacy systems to adhere to a single, global message format (cost!?)
- It is often necessary to learn to live with different formats
- How?
  - By Message Brokers
  - Their role is to convert incoming messages in a format that can be understood by the destination application

Message Broker Organisation



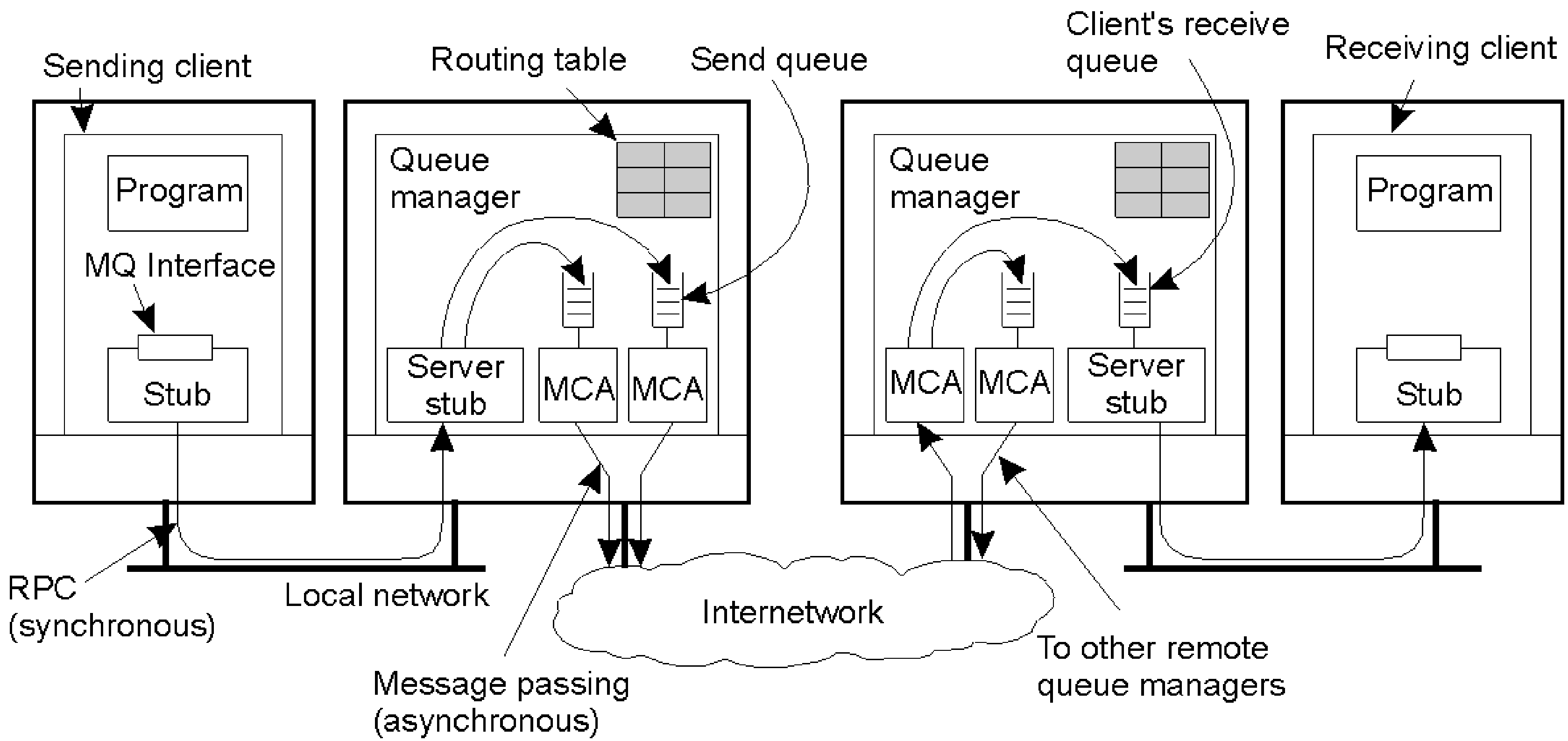
- The general organisation of a message broker in a message-queuing system
- Also known as an interface engine

Message-Queuing (MQ) Applications

- General-purpose MQ systems support a wide range of applications, including:
  - Electronic mail
  - Workflow
  - Groupware
  - Batch Processing
- Most important MQ application area:

The integration of a widely dispersed collection of database applications (which is all but impossible to do with traditional RPC/RMI techniques)

Example: IBM MQSeries



- General organisation of IBM's WebSphere MQSeries message-queuing system
  - Large-scale databases, finance

Stream-Oriented Communications

- With RPC, RMI and MOM, the effect that time has on correctness is of little consequence

- However, audio and video are time-dependent data streams
  - if the timing is off, the resulting “output” from the system will be incorrect

- Time-dependent information
  - known as continuous media communications

Examples:

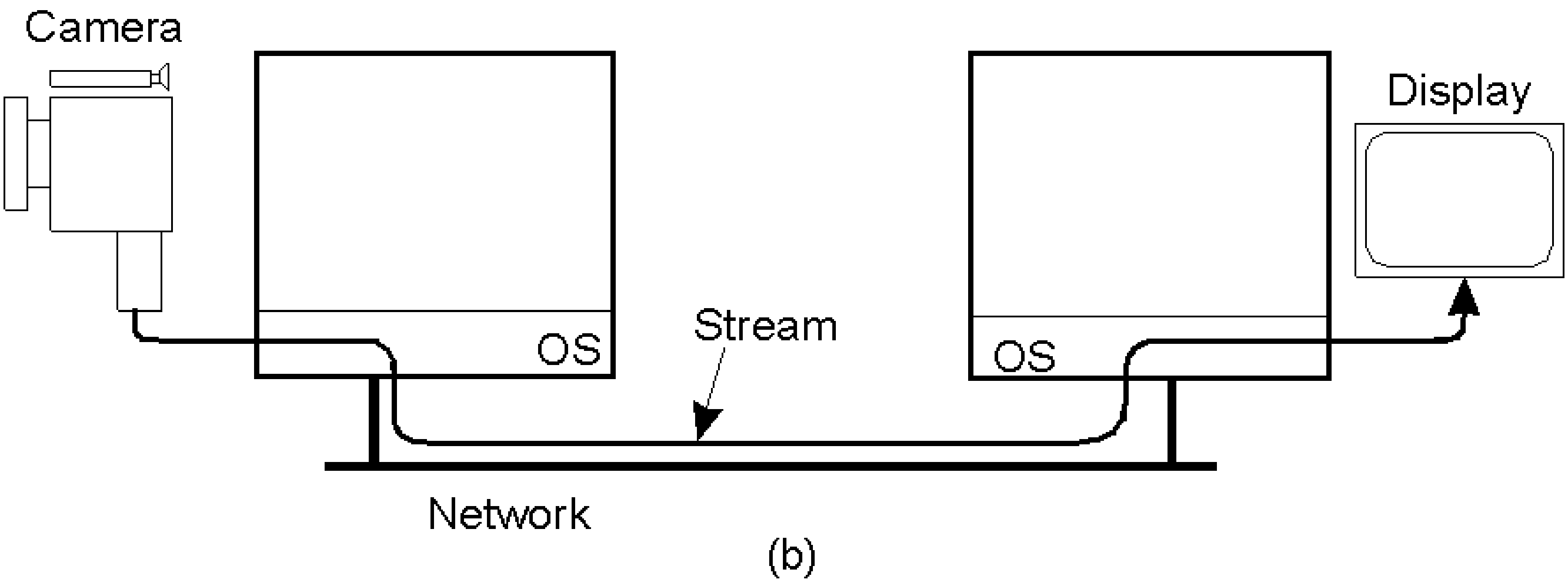
- Audio: (PCM) samples played at 1/44100 sec intervals (exactly)
- Video: 30 frames per second (30-40 msec per image)

- KEY MESSAGE: Timing is crucial!

Transmission Modes

- Asynchronous transmission mode
  - The data stream is transmitted in order, but there's no timing constraints placed on the actual delivery (e.g., File Transfer)
- Synchronous transmission mode
  - The maximum end-to-end delay is defined (but data can travel faster)
- Isochronous transmission mode –
  - Data transferred on time
  - There's a maximum and minimum end-to-end delay (known as bounded jitter)
  - Known as streams
  - Very useful for multimedia systems

End-device to End-device Streams



- Setting up a stream directly between two devices
  - i.e., no inter-networked processes

Two Types of Streams

- Simple Streams – one single sequence of data, for example: voice

- Complex Streams – several sequences of data (sub-streams) that are time-related
  - Think of a movie:
    - 1 stream for video
    - 2 streams for stereo sound
    - 1 stream for subtitles, …

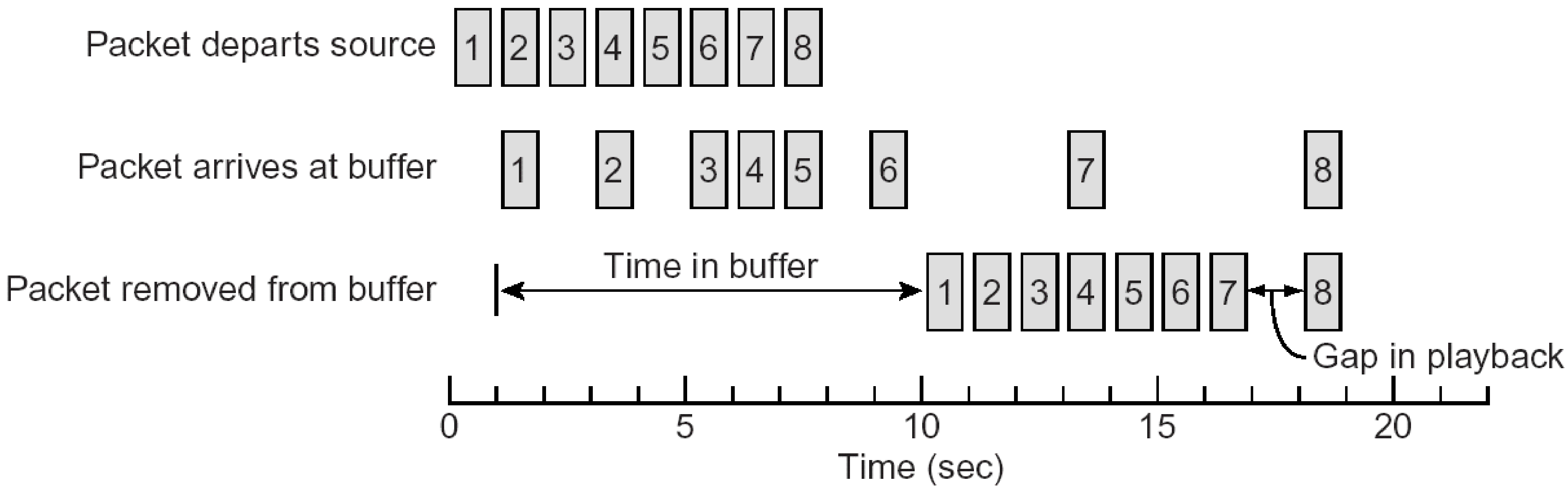
- This leads to data synchronisation problems … which are not at all easy to deal with

Quality of Service

- Definition: "Ensuring that the temporal relationships in the stream can be preserved"
- QoS is all about three things:
  1. Timeliness
  2. Volume
  3. Reliability
- Most current operating systems and networks do not include the QoS management facilities
- Bleeding edge of the discipline

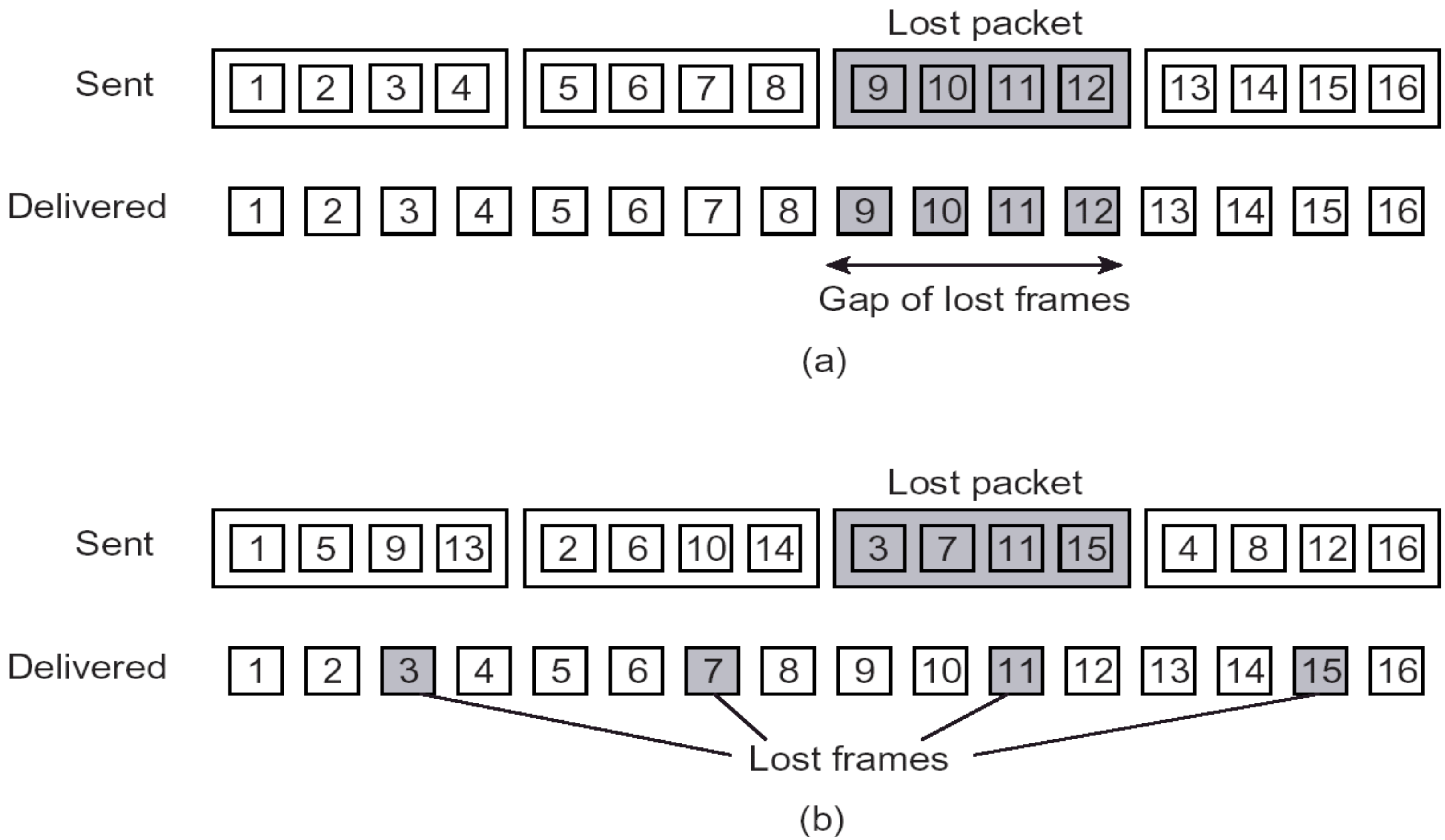
Enforcing QoS (1)

- Use buffers to reduce jitter
- Note: the larger the buffer the greater the initial delay of starting playback



Enforcing QoS (2)

- Underlying best-effort service: packets may be lost
- Reducing the effect of packet loss



Distributed Comms. - Summary

- Power and flexibility essential, as network programming primitives are too primitive
- Middleware Comms. Mechanisms – providing support for a higher-level of abstraction
- RPC and RMI: synchronised, transient
- MOM: convenient, asynchronous, persistent
- Streams: a special case, useful when dealing with temporally-related data (not easy)