Communication

Message-Oriented Communication
and Streams

Message-Oriented Middleware: MOM

- As a communications mechanism, RPC/RMI is often inappropriate.
- For example: 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*.
- Something else is needed: **Messaging**.

Some DS Comms. Terminology

Persistent Communications:

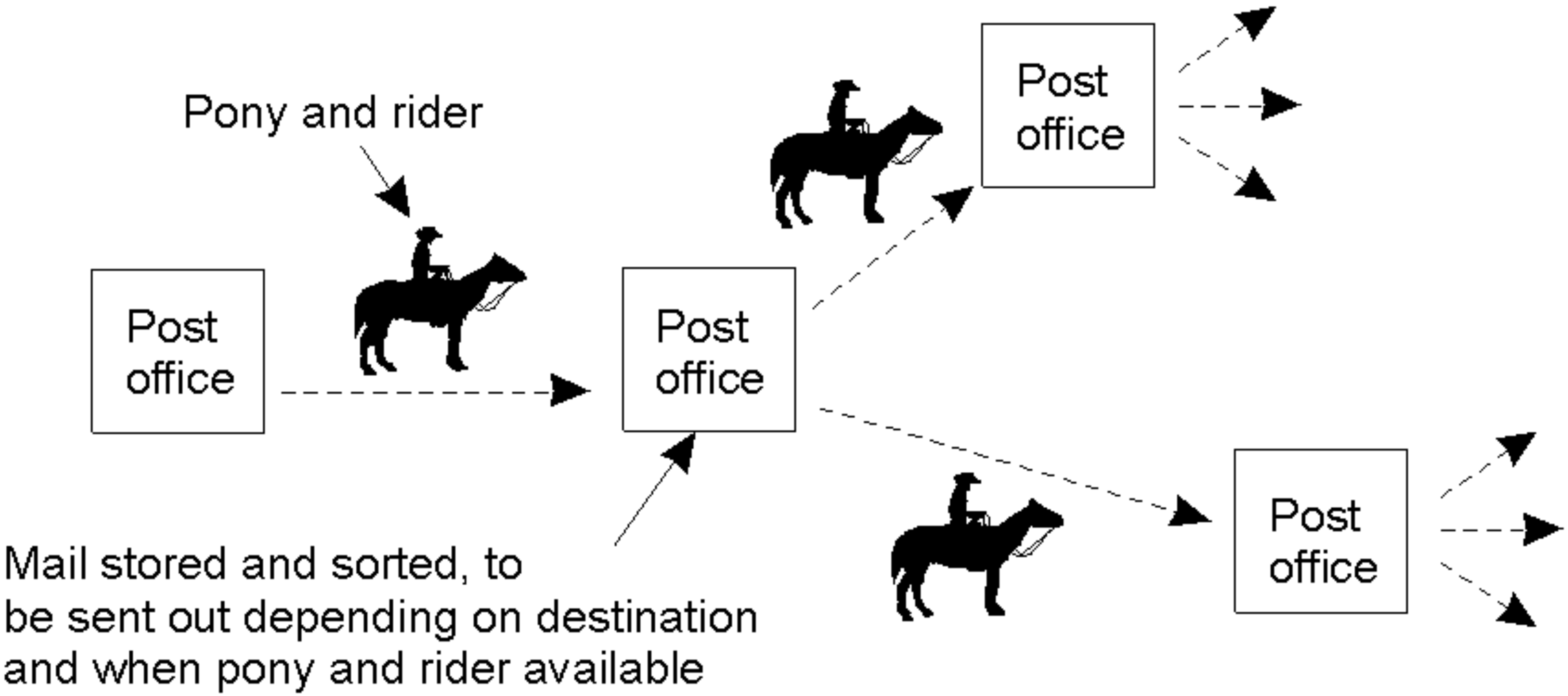
Once sent, the "sender" can stop executing. The "receiver" need not be operational at this time – the commmunications system **buffers** the message as required (until it can be delivered).

[Can you think of an example?]

Contrast to *Transient Commuications:*

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


Pony and rider
Post office
Post office
Post office
Post office
Mail stored and sorted, to
be sent out depending on destination
and when pony and rider available

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

More DS Comms. Terminology

- *Asynchronous Communications:*

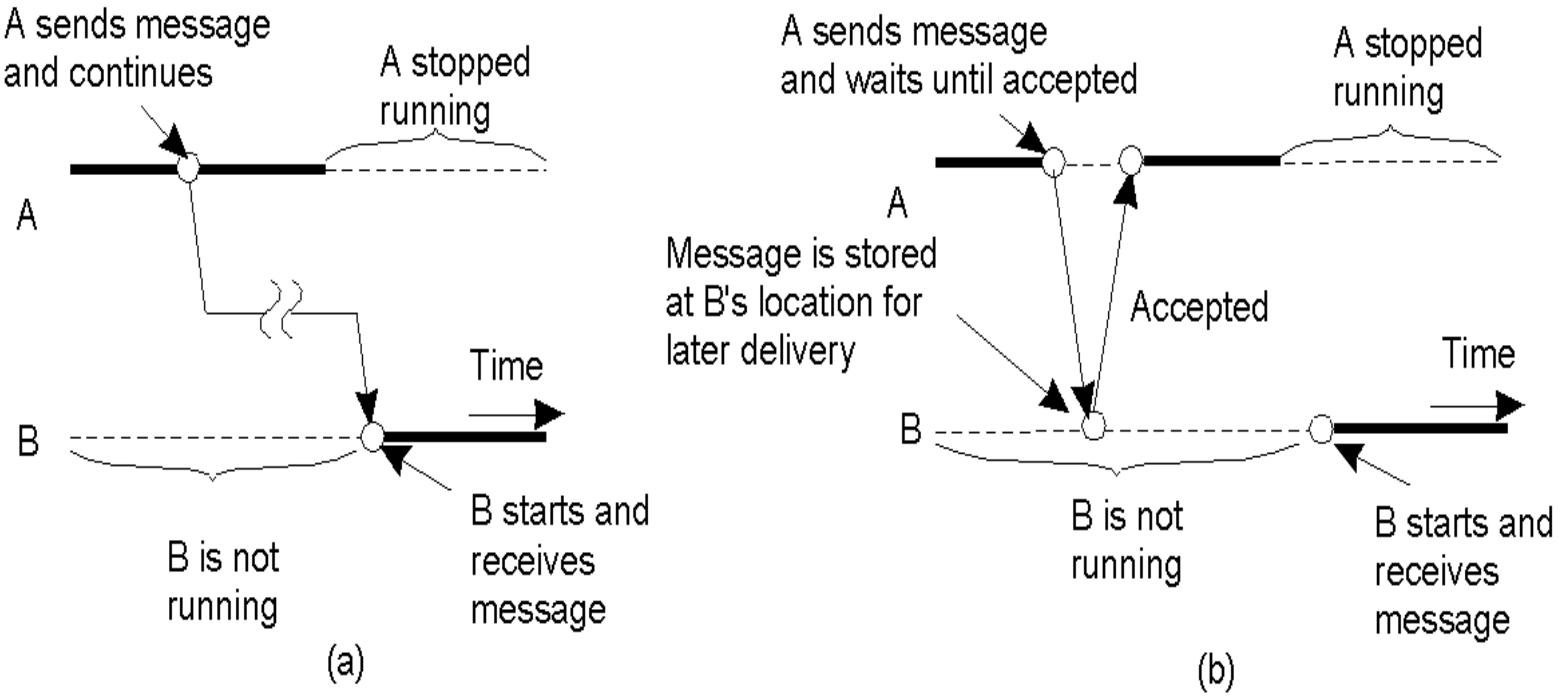
  A sender **continues** with other work immediately upon sending a message to the receiver.

- *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 have a look now on combinations

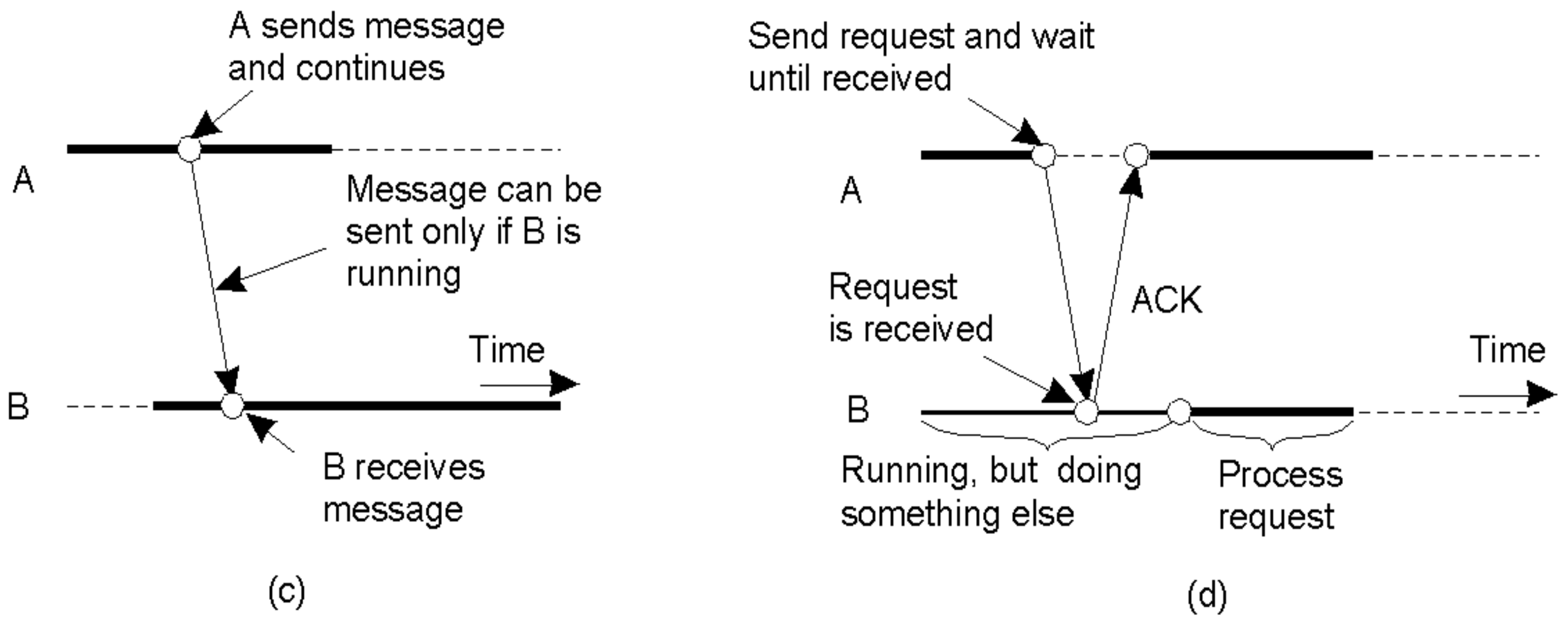
Classifying Distributed Communications (1)


A sends message and continues
A stopped running
A
B
Time
B is not running
B starts and receives message
(a)
A sends message and waits until accepted
A stopped running
A
Message is stored at B's location for later delivery
Accepted
Time
B
B is not running
B starts and receives message
(b)

a) Persistent asynchronous communication.

b) Persistent synchronous communication.

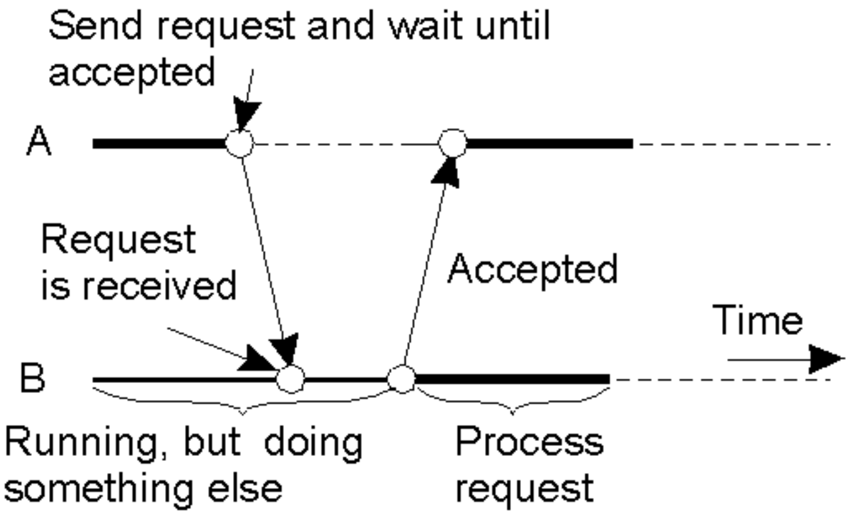
Classifying Distributed Communications (2)


A sends message and continues
A
Message can be sent only if B is running
Time
B
B receives message
(c)
Send request and wait until received
A
Request is received
ACK
Time
B
Running, but doing something else
Process request
(d)

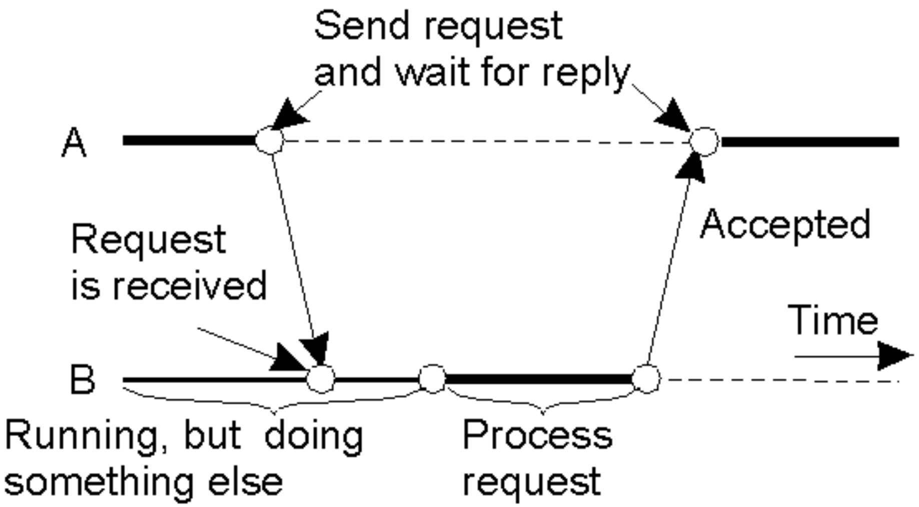
c) Transient asynchronous communication.

d) Receipt-based transient synchronous communication.

Classifying Distributed Communications (3)


Send request and wait until accepted
A
Request is received
Accepted
Time
B
Running, but doing something else
Process request

(e)


Send request and wait for reply
A
Request is received
Accepted
Time
B
Running, but doing something else
Process request

(f)

e) Delivery-based transient synchronous communication at message delivery.

f) Response-based transient synchronous communication.

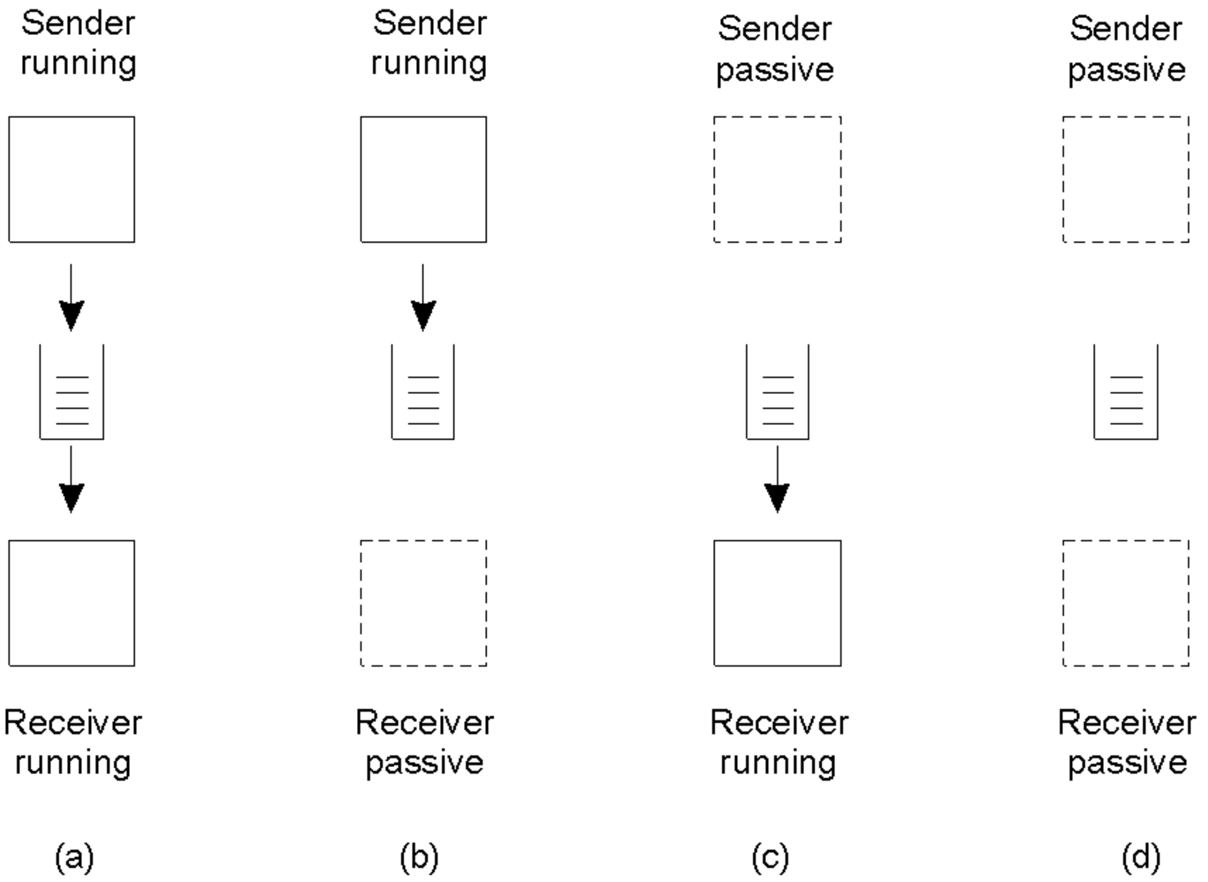
Example: Message-Passing Interface

- For hight-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

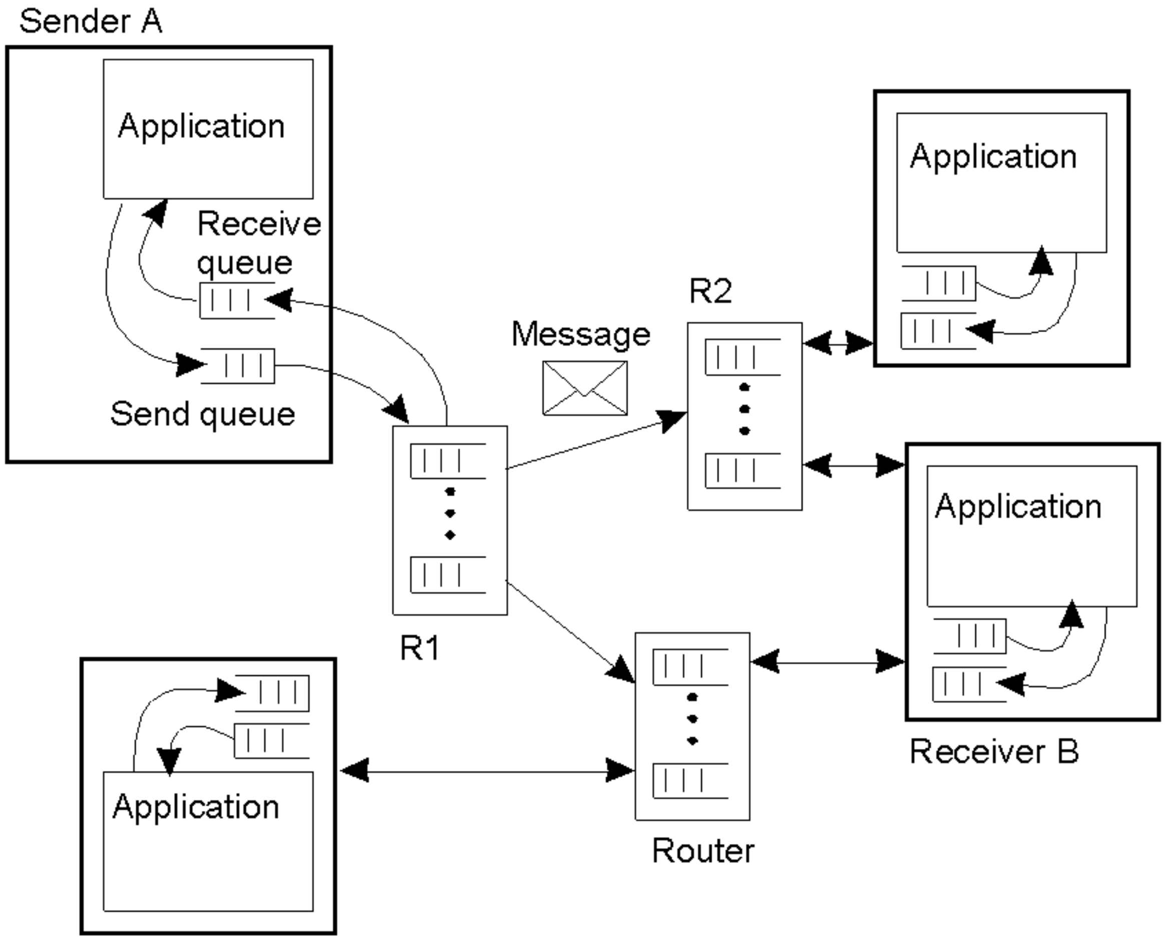


Four combinations for “loosely-coupled” communications which use message-queues.

Message-Queuing System Architecture

- Messages are “put into” a *source queue*.
- They are then “taken from” a *destination queue*.
- Obviously, 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*.
- These are message-queuing “relays” that interact with the distributed applications and with each other. Not unlike routers, these devices support the notion of a DS “overlay network”.

General Architecture of a Message-Queuing System

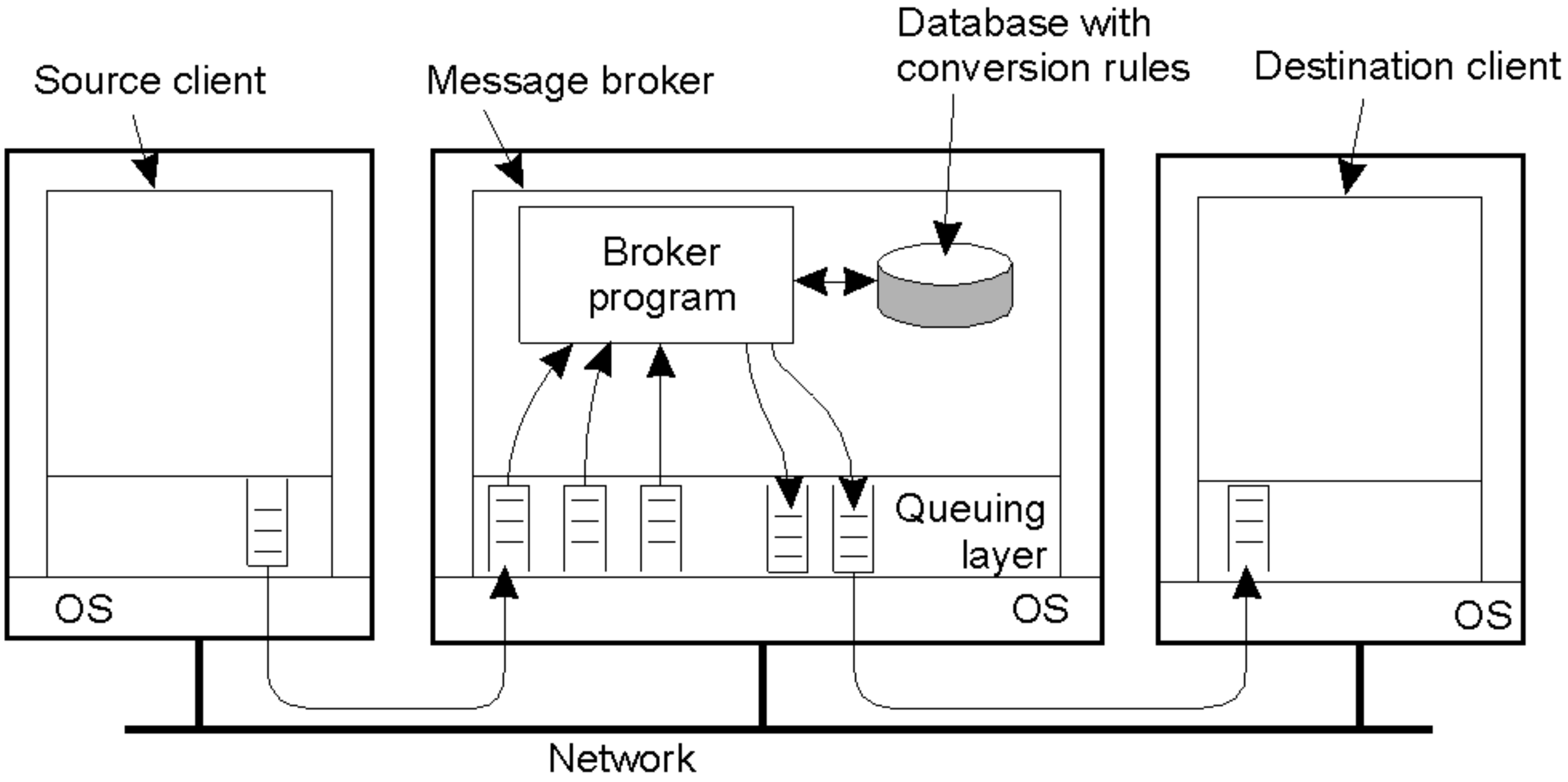


The general organization of a message-queuing system with routers. The Queue Managers can reside within routers as well as within the DS end-systems.

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 live with diversity (there's no choice).
- **How?**
- Meet the "Message Broker".

Message Broker Organization

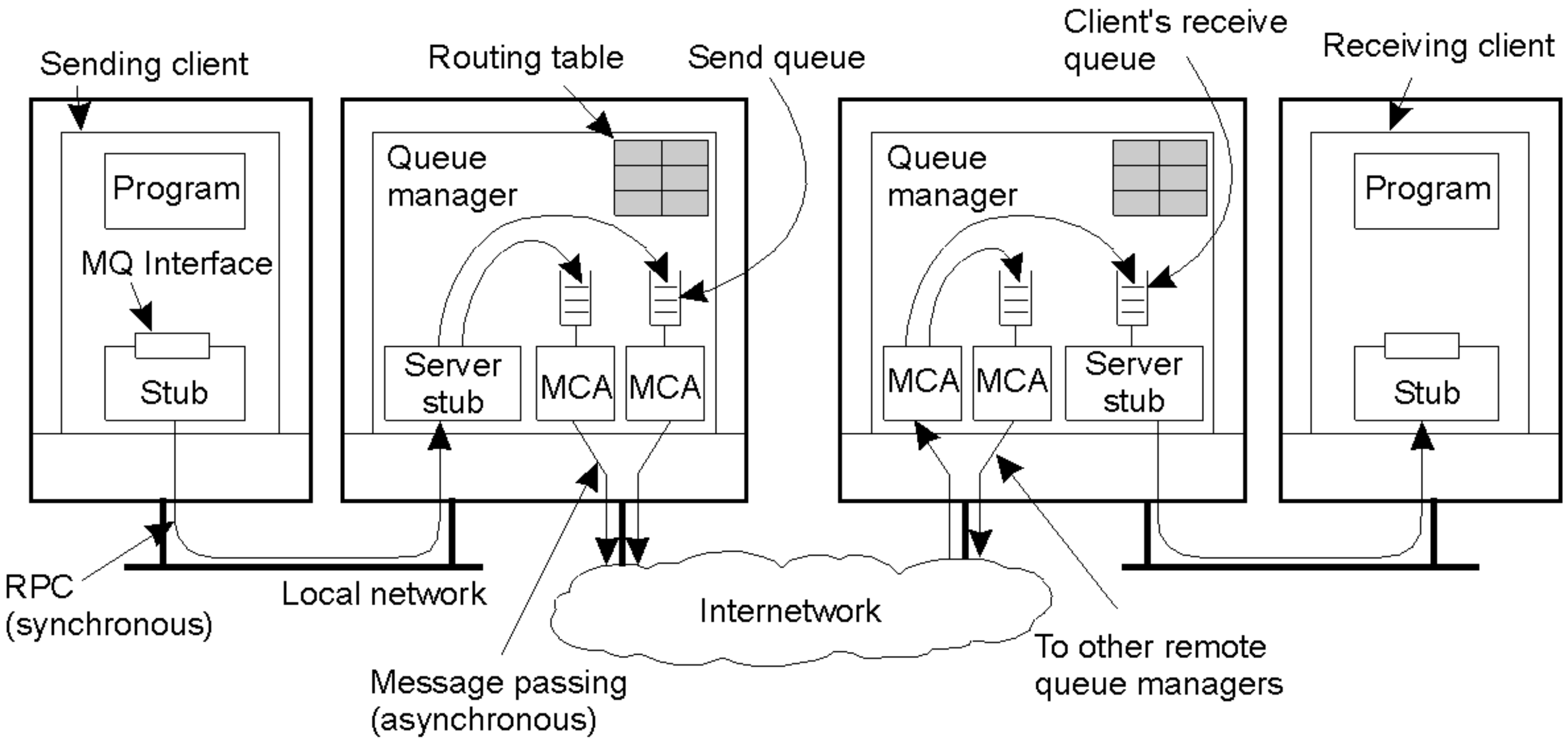


The general organization of a message broker in a message-queuing system – also known variously as an “interface engine”.

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 organization of IBM's 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.

Example: voice: PCM: 1/44100 sec intervals on playback.

Example: video: 30 frames per second (30-40 msec per image).

KEY MESSAGE: Timing is crucial!

Transmission Modes

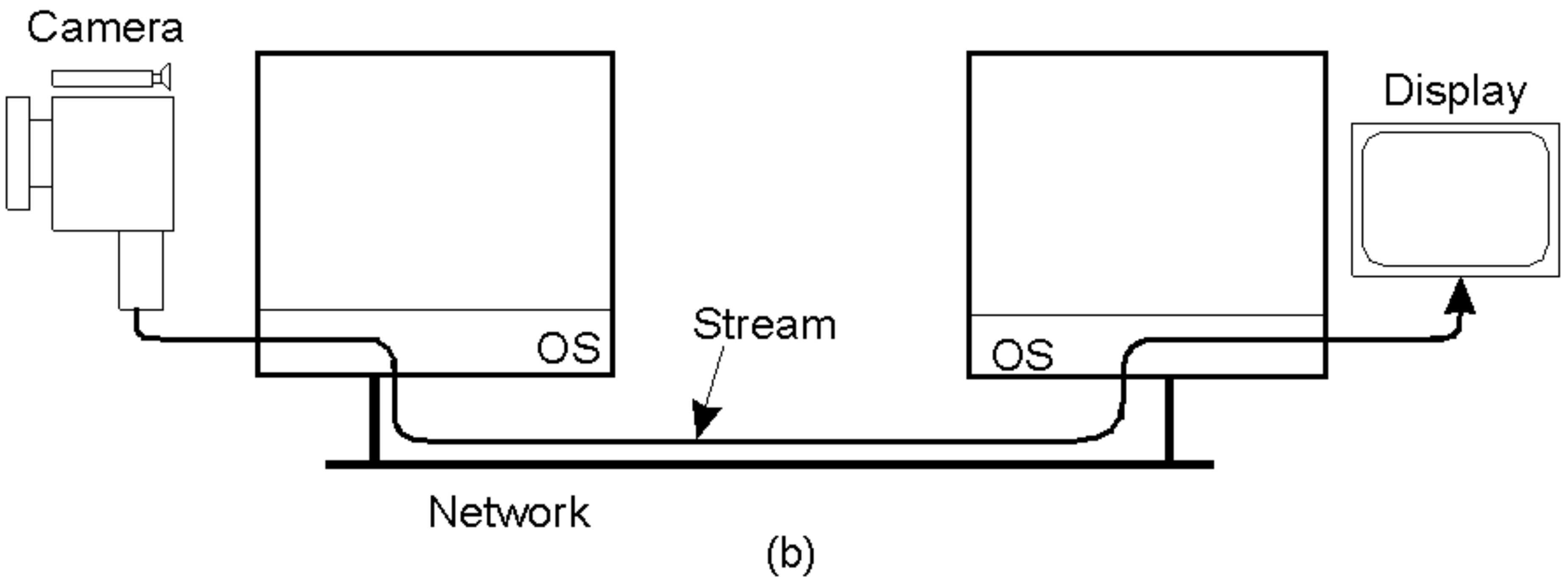
Asynchronous transmission mode – the data stream is transmitted in order, but there's no timing constraints placed on the actual delivery (e.g., File Transfer).

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" – isochronous transmission mode is 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 “related” by time. Think of a lip-synchronized movie, with sound and pictures, together with sub-titles …

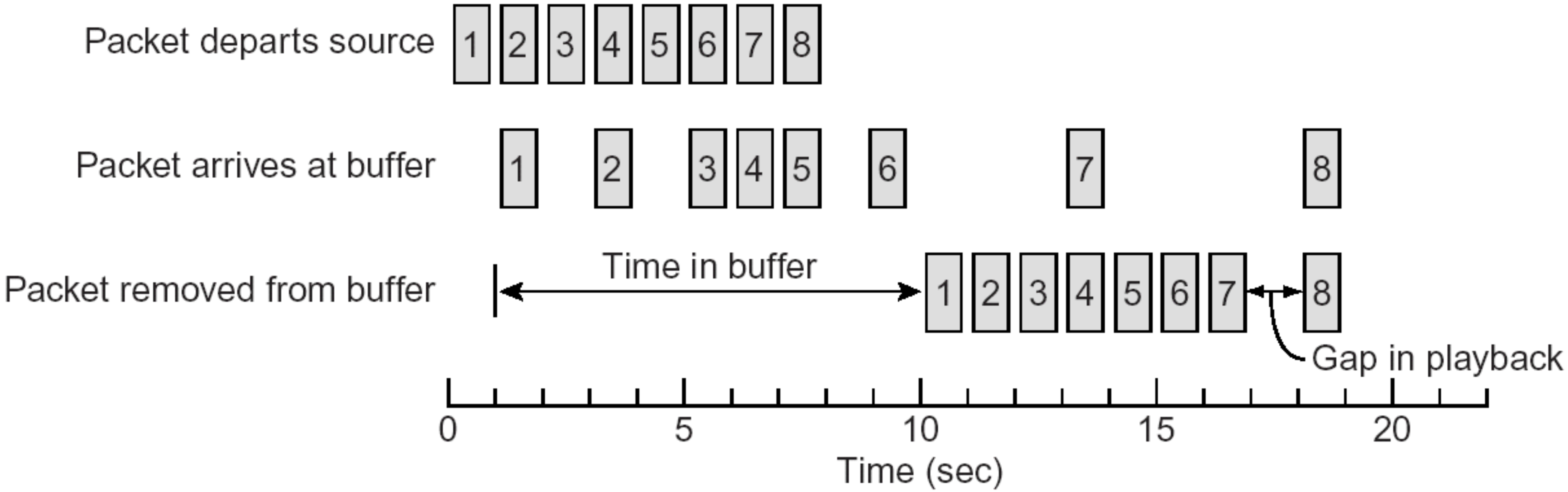
This leads to data synchronization problems … which are not at all easy to deal with.

Quality of Service

- Definition: “ensuring that the temporal relationships in the stream can be preserved”.
- QoS is all about three things:
  a) Timeliness,
  b) Volume and
  c) Reliability.
- Most current operating systems and networks do not include the QoS management facilities
- Bleeding edge of the discipline

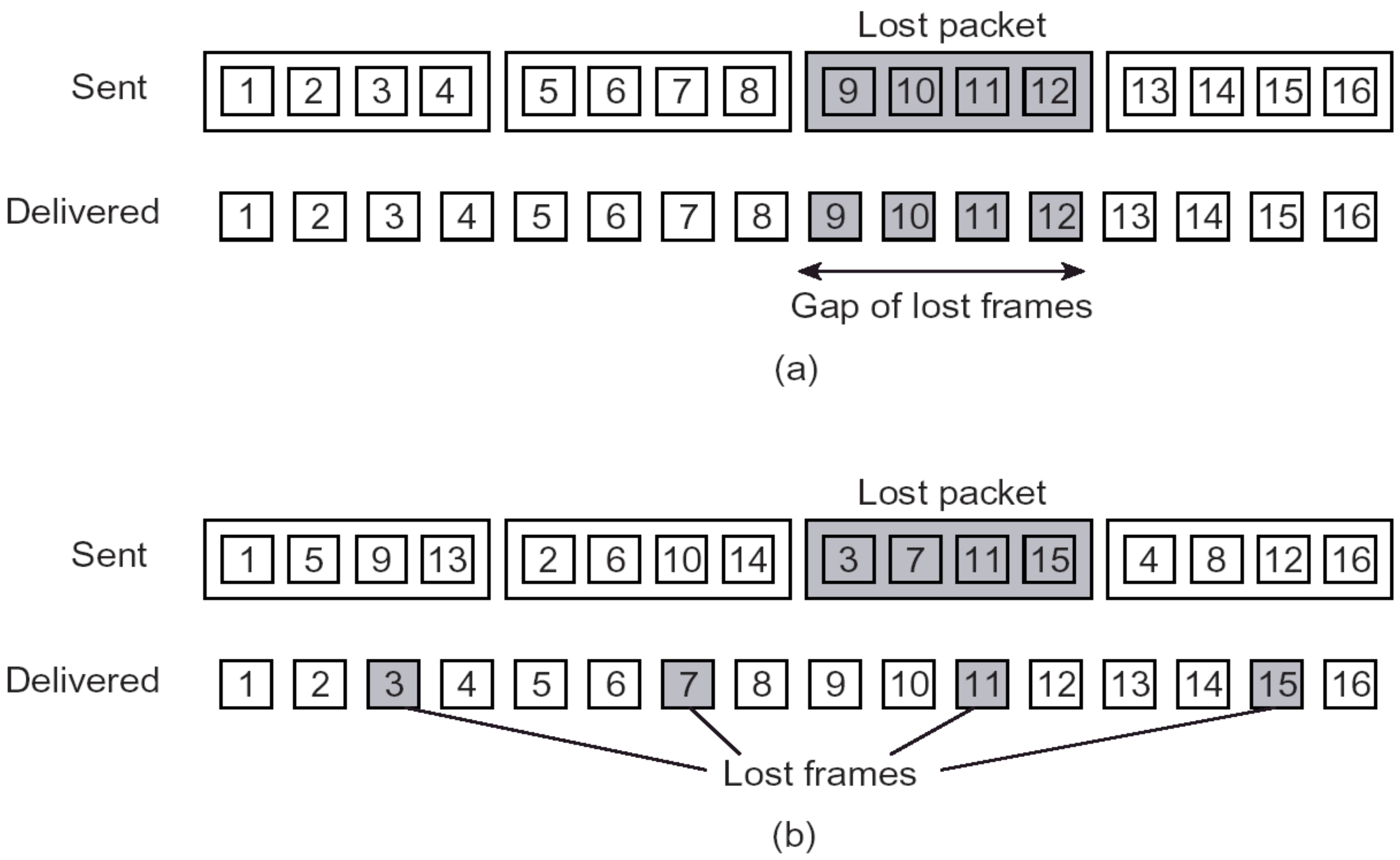
Enforcing QoS (1)

User **buffers** to reduce jitter



Enforcing QoS (2)

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: synchronized, transient.
- MOM: convenient, asynchronous, persistent.
- Streams: a special case, useful when dealing with “temporally related data” (not easy).