Communication: MOM & Streams

CS403/534

Distributed Systems

Erkay Savas
Sabanci University

Message-Oriented Communication

- Synchronous versus asynchronous communication
  - RPC, RMI are synchronous
- Message-Queuing Systems
- Streams

Synchronous Communication

- Client/sever model is generally based on synchronous communication:
  - Both client and server have to be active at the time of the communication
  - Client blocks
  - Server blocks
- Drawbacks of synchronous communication:
  - Client cannot do any other work while waiting for reply
  - Failures have to be dealt with immediately (since the client is waiting)
  - In many cases, the model is simply not appropriate (mail, news)

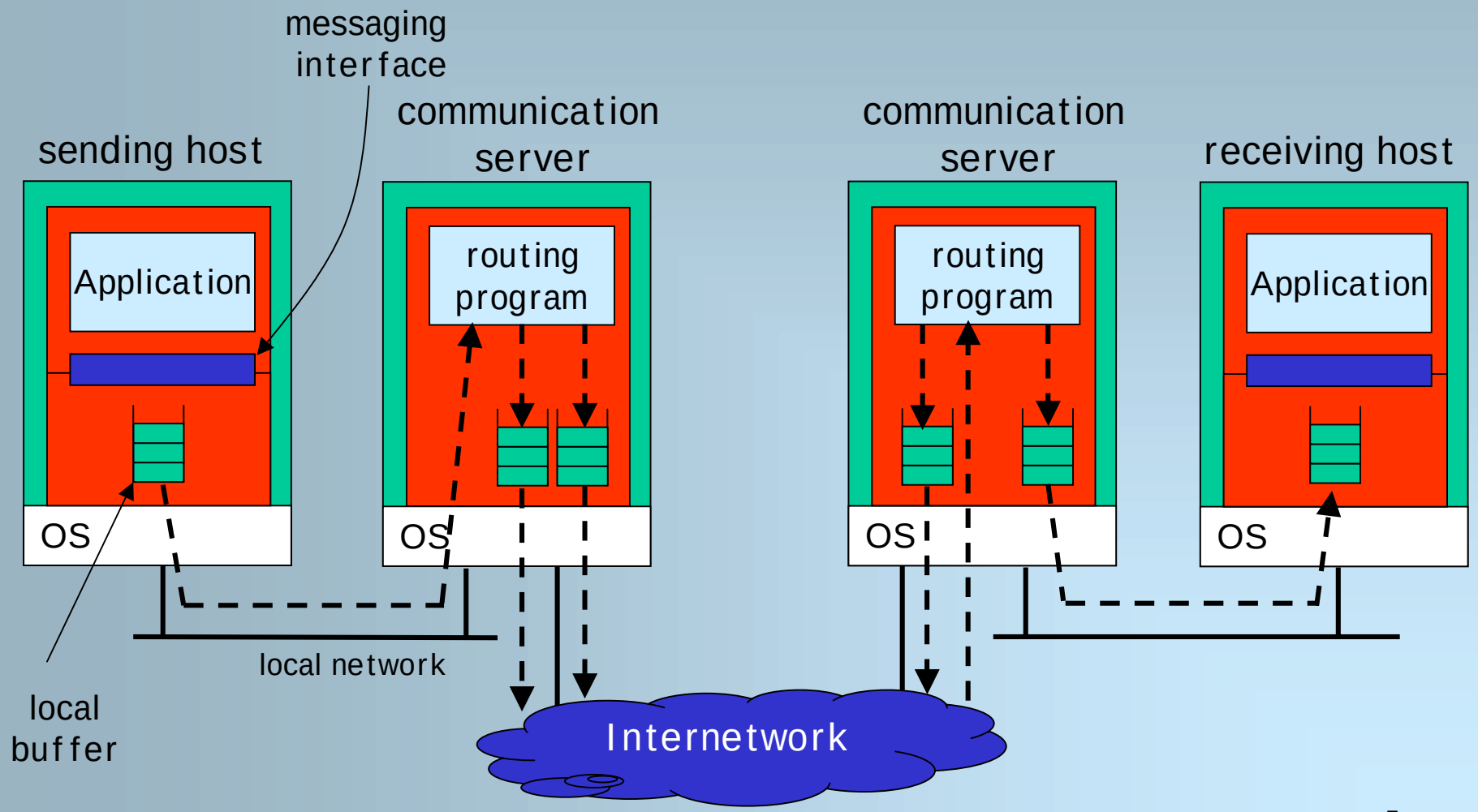
Asynchronous Communication: Messaging

Message-oriented middleware: aims at high-level asynchronous communication:

- Processes send each other messages, which are queued (in buffers)
- Sender need not wait for immediate reply, but can do other things
- Receiver needs not be active at the time communication is initiated.
- Sender needs not be active at the time the actual communication takes place.
- Middleware often ensures fault tolerance

Communication System

General Organization



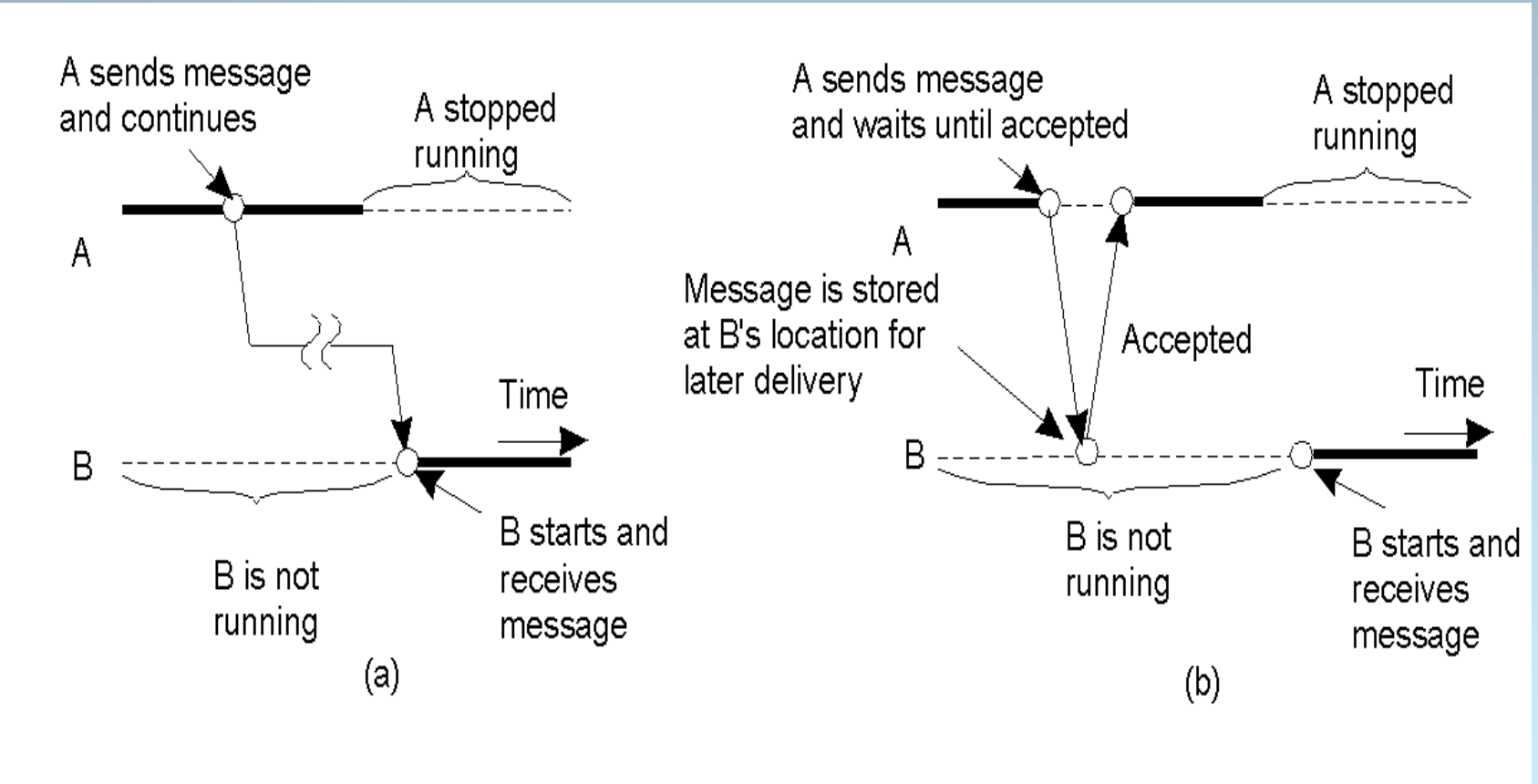
Persistent vs. Transient Communication

- **Persistent communication:** a message that has been submitted for transmission is stored by communication system as long as it takes to deliver it to the receiver.
  - Electronic mail system
- **Transient communication:** Both sender and receiver must be active
  - If a communication server cannot deliver the message to the receiver, the message will be discarded.
  - All transport-level communication services offer only transient communication
  - In this case, communication server corresponds to a traditional store-and-forward router

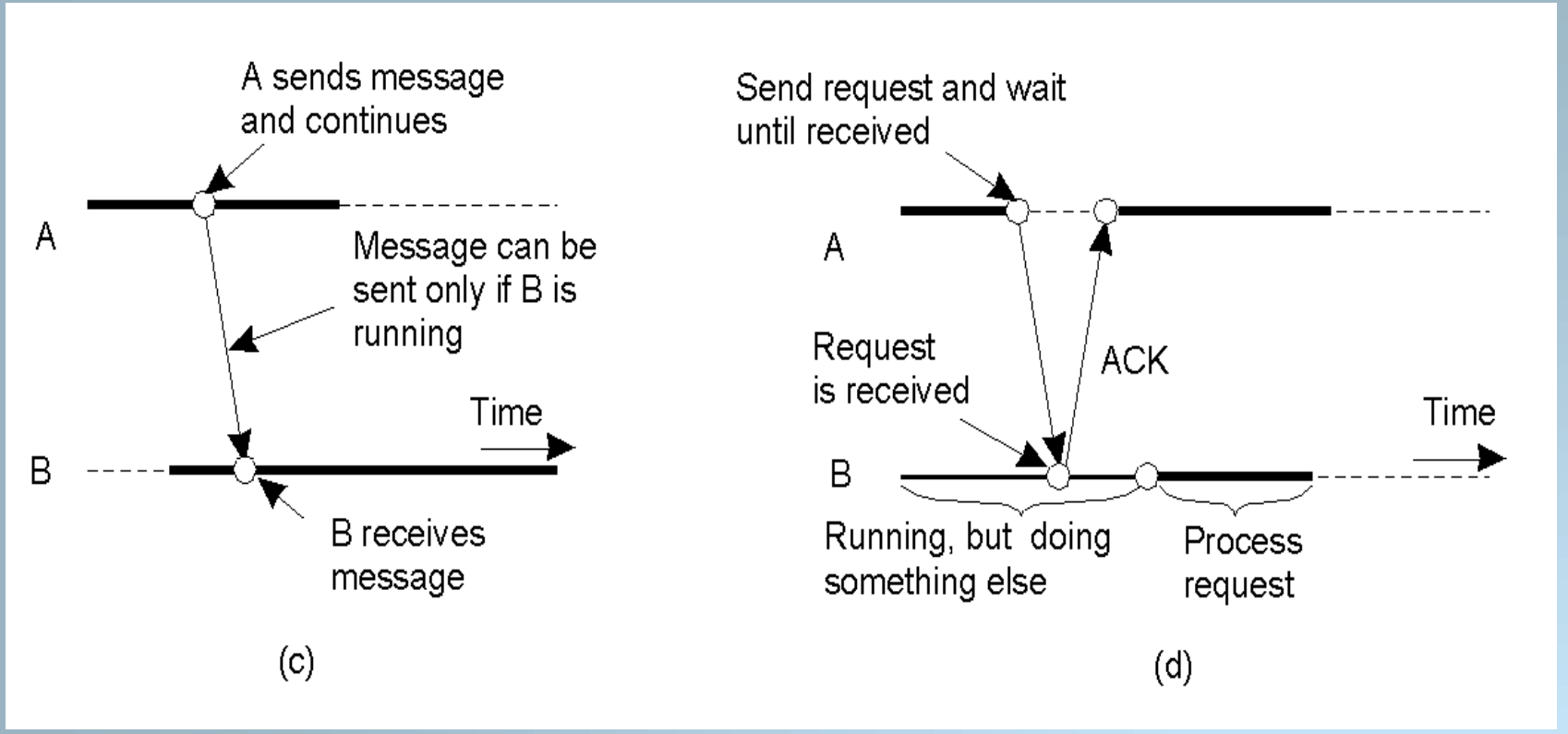
Forms of Communication (1)

a) Persistent asynchronous communication

b) Persistent synchronous communication



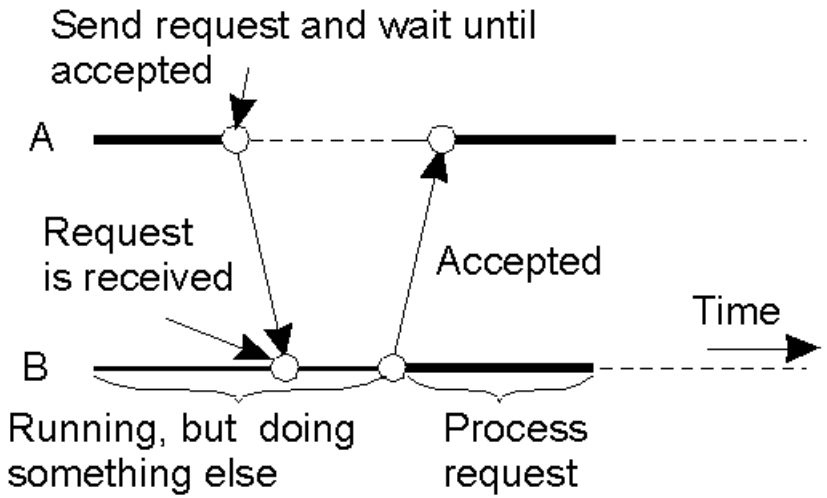
Forms of Communication (2)



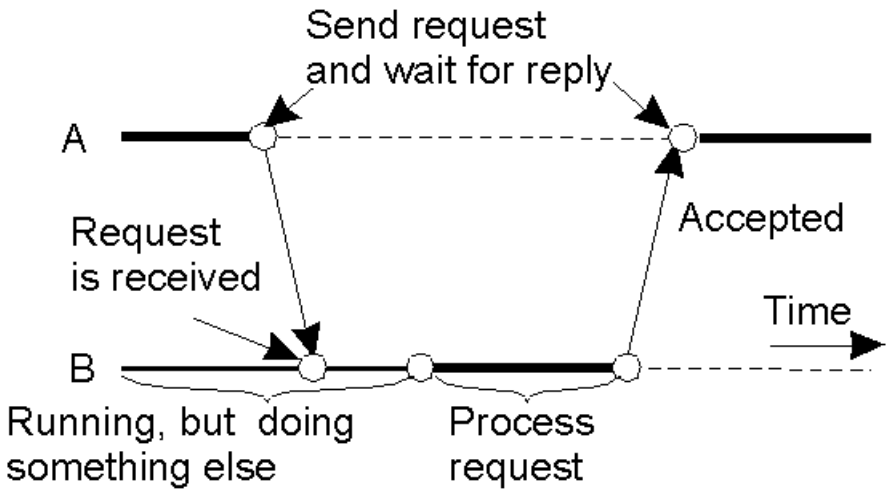
c) Transient asynchronous communication
(e.g. UDP, one way asynchronous RPC)

d) Receipt-based transient synchronous communication (weakest)

Forms of Communication (3)



(e)



(f)

e) Delivery-based transient synchronous communication at message delivery

f) Response-based transient synchronous communication

Transport-Level Sockets

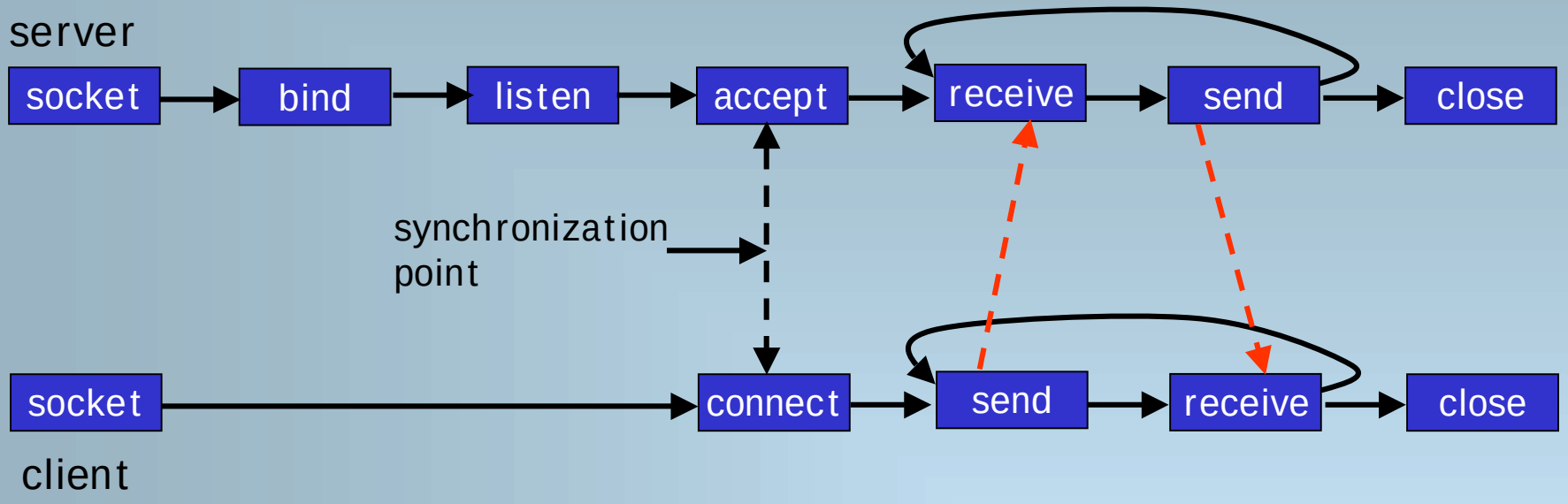
- Transient communication employs transport-level sockets
- Sockets interface
- **Socket:** A communication endpoint
  - to which an application can write data that are to be sent over the underlying network, and from which incoming data can be read
- An abstraction over the actual communication endpoint
- Creating a socket means that the local OS reserves resources to accommodate sending and receiving messages for the specified protocol

Socket Primitives for TCP/IP

Primitive	Meaning
socket	Create a new communication endpoint
bind	Attach a local address to a socket (IP + port number)
listen	Announce willingness to accept connections (nonblocking) Creates a queue associated with the socket
accept	Block caller until a connection request arrives (blocking)
connect	Actively attempt to establish a connection (blocks client)
send	Send some data over the connection
receive	Receive some data over the connection
close	Release the connection

Berkeley Sockets

Connection-oriented communication pattern using sockets.



- Accept blocks the caller
- When a request arrives, a new socket created with the same properties with original one
- New socket is handed to the caller

Message-Passing Interface (MPI)

- Sockets are insufficient:
  1. Not supporting different communication models
  2. They are designed to communicate across networks using TCP/IP protocol stack. They are not suitable for high-speed interconnection networks used in, for example, MPPs or COWs.
- **MPI** is designed for developing high-performance parallel applications.
  - Higher level of abstraction
  - No communication server → transient communication
  - Communication takes place within a known group of processes (*groupID*, *processID*)

MPI: Message-Passing Primitives

Primitive	Meaning
MPI_bsend	Append outgoing message to a local send buffer
MPI_send	Send a message and wait until copied to local or remote buffer
MPI_ssend	Send a message and wait until receipt starts
MPI_sendrecv	Send a message and wait for reply
MPI_isend	Pass reference to outgoing message, and continue
MPI_issend	Pass reference to outgoing message, and wait until receipt starts
MPI_recv	Receive a message; block if there are none
MPI_irecv	Check if there is an incoming message, but do not block

MPI: Communication Models

- **MPI_bsend:** transient asynchronous communication
  - The message is copied to a local buffer in MPI runtime system
  - The sender immediately continues and the message is removed when the receiver has called a receive primitive
- **MPI_send:** blocks the sender. Two implementation types:
  1. Receipt-based transient synchronous communication (blocks until the message is copied to MPI runtime system at receiver site)
  2. Delivery-based transient synchronous communication (blocks until the receiver initiates receive operation)

MPI: Communication Models

- **MPI_ssend:** blocks the sender until its request is accepted for further processing
  - Delivery-based transient synchronous communication
- **MPI_sendreceive:** blocks the sender
  - Response-based transient synchronous communication
- **MPI_recv:** Blocking
- **MPI_irecv:** Non-blocking

Message-Oriented Persistent Communication

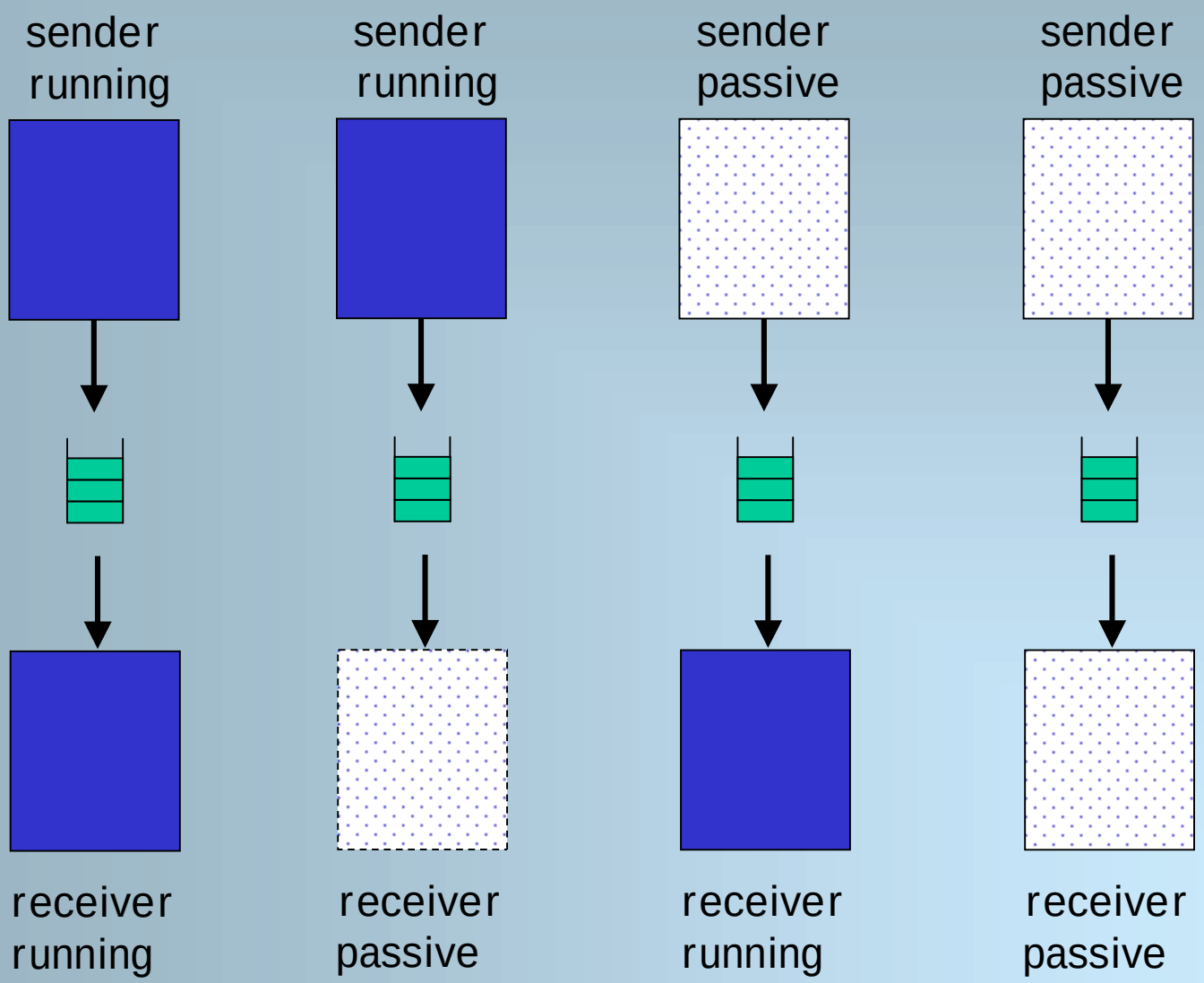
Message-Queuing Systems or Message-Oriented Middleware (MOM)

- They provide extensive support for persistent asynchronous communication
- They offer intermediate-term storage capacity for messages
- Neither sender nor receiver have to be active during message transmission
- For message transfers that typically take minutes (not seconds or milliseconds)
- Communication servers play an active role.

Message-Queuing Model

- The persistency is fulfilled by the various queues in the system
- Each application has a private queue to which other applications can send messages
- Message-queuing systems guarantee that the message eventually will be inserted in recipient's queue
  - No guarantee about when it is delivered
  - No guarantee whether the message will ever be read at all
- Loosely-coupled communication
- Systemwide unique name of the destination queue is used for addressing

Loosely-Coupled Communication

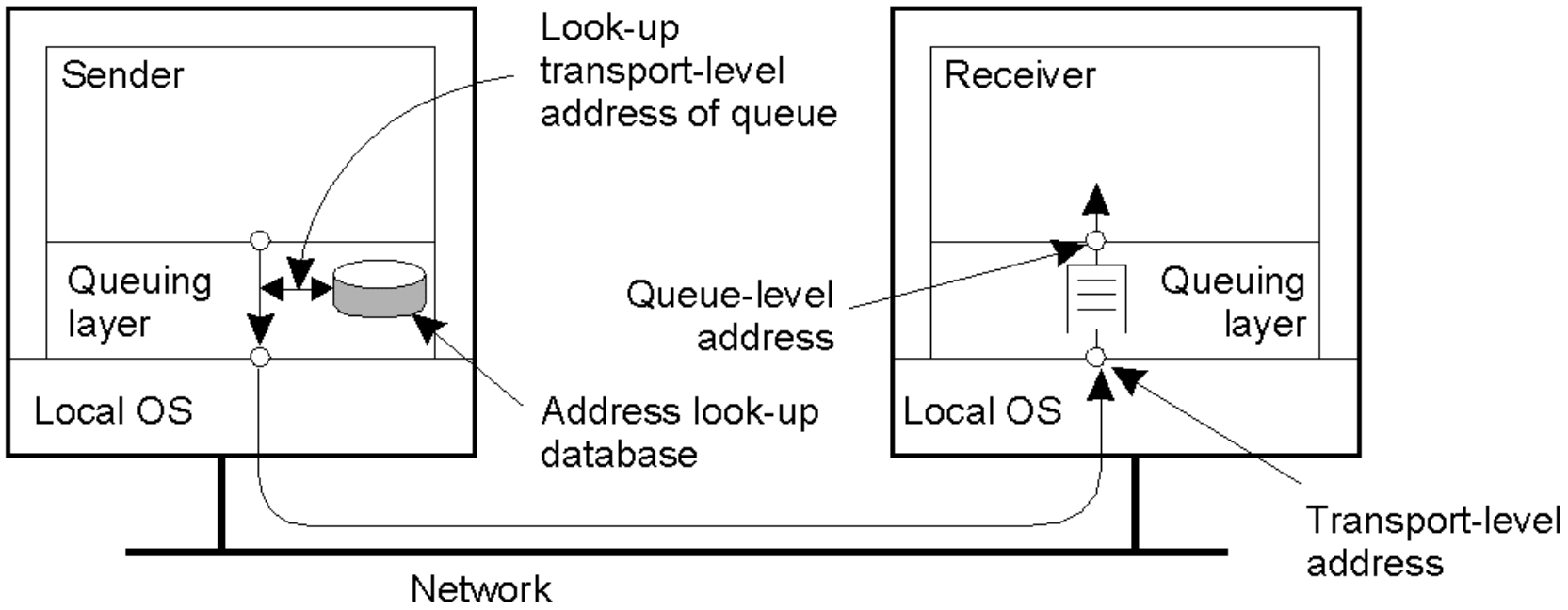


Basic Interface

Primitive	Meaning
`put`	Append a message to a specified queue (*nonblocking*)
`get`	Block until the specified queue is nonempty, and remove the first message (*blocking if queue empty*)
`poll`	Check a specified queue for messages, and remove the first. Never block.
`notify`	Install a handler (as a *callback function*) to be called when a message is put into the specified queue.

General Architecture

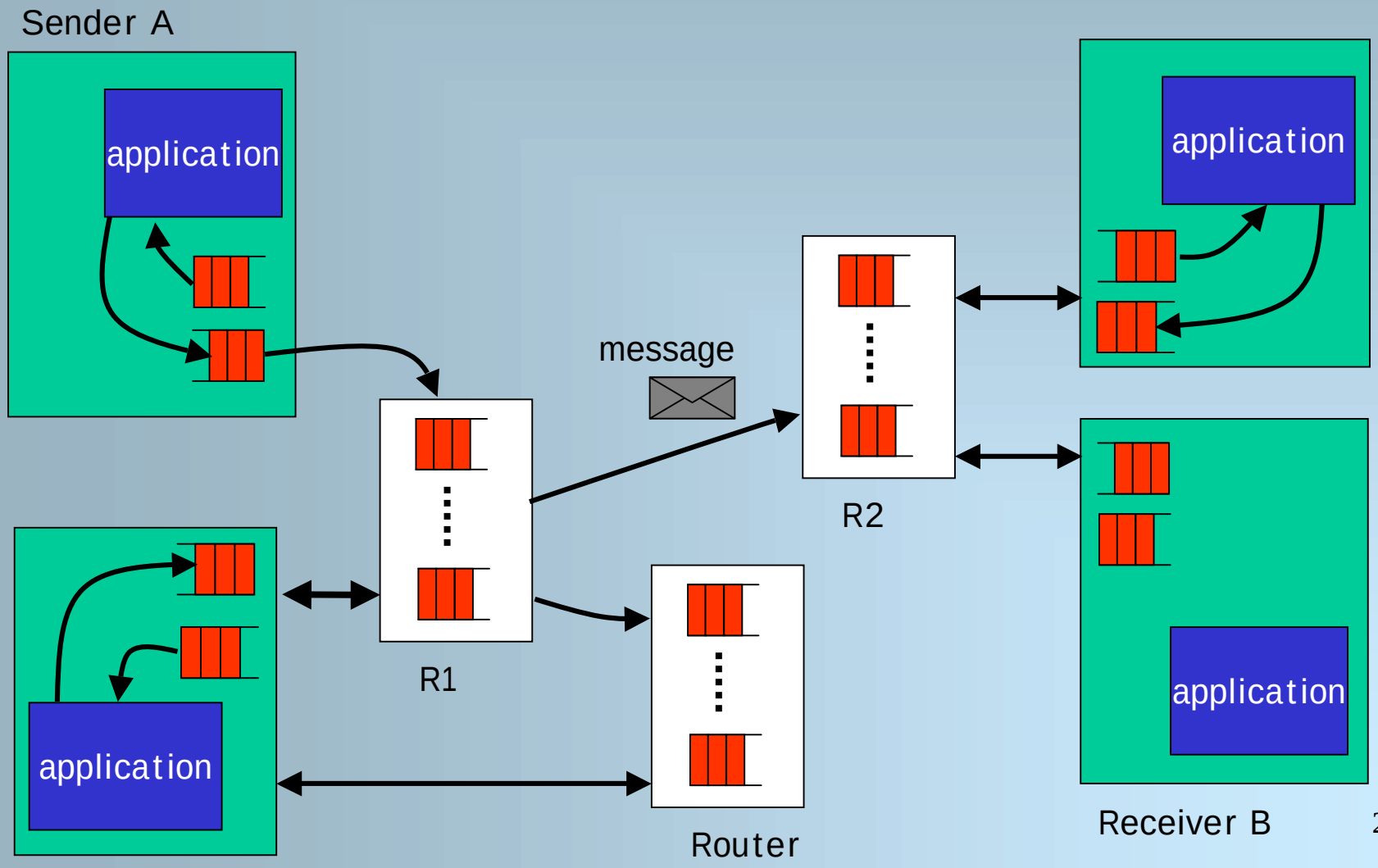
The relationship between queue names and network-level addressing.



Queues are managed by queue managers

Message-Queuing System with Routers

- Few routers (relays) that know about the network topology
- Queue manager needs not to store queue-to-address mapping



Benefits of Using Routers

- Relays maintain a static routing scheme
- Manual configuration of routers may get problematic if queuing network grows.
  - In many message-queuing systems, there is no general naming service available
- Relays are good for secondary processing
  - Logging for security or fault tolerance
  - Transforming messages into a format that can be understood by the receiver
  - multicasting

Message-Queuing: Applications

E-mail:

- Direct support for end-users
- Additional requirements such as message filtering, advanced messaging database

General message-queuing systems:

- No special emphasis on direct support for end-users
- Primary goal is to set up persistent communication between processes
- Guaranteed delivery, fault tolerance, logging facilities, load balancing, efficient multicasting
- e-mail, workflow, groupware, batch processing
- Integration of (possibly widely dispersed) database applications into a multidatabase system
  - Subqueries are packed into messages and sent to individual databases

Reading Assignment

1. Message Brokers, on pages 113-114 from the textbook
2. IBM MQSeries, on pages 115-119 from the textbook

Stream-Oriented Communication

- For communication types we have learned so far, the timing has no effect on correctness.
- Support for continuous data
  - Communication of time dependent information
  - Playing CD quality audio and video streams
- Streams in distributed systems
- Stream management

Continuous Media

- **Observation:** All communication facilities discussed so far are essentially based on a *discrete*, that is *time-independent* exchange of information
  - Temporal relationship between data items are NOT fundamental to correctly interpreting data
- **Continuous media:** characterized by the fact that values are time dependent:
  - Temporal relationship between data items ARE fundamental to correctly interpreting data
  - Audio, Video, Animations, Sensor data (temperature, pressure, etc)
  - Correct order and a specific rate

Data Streams

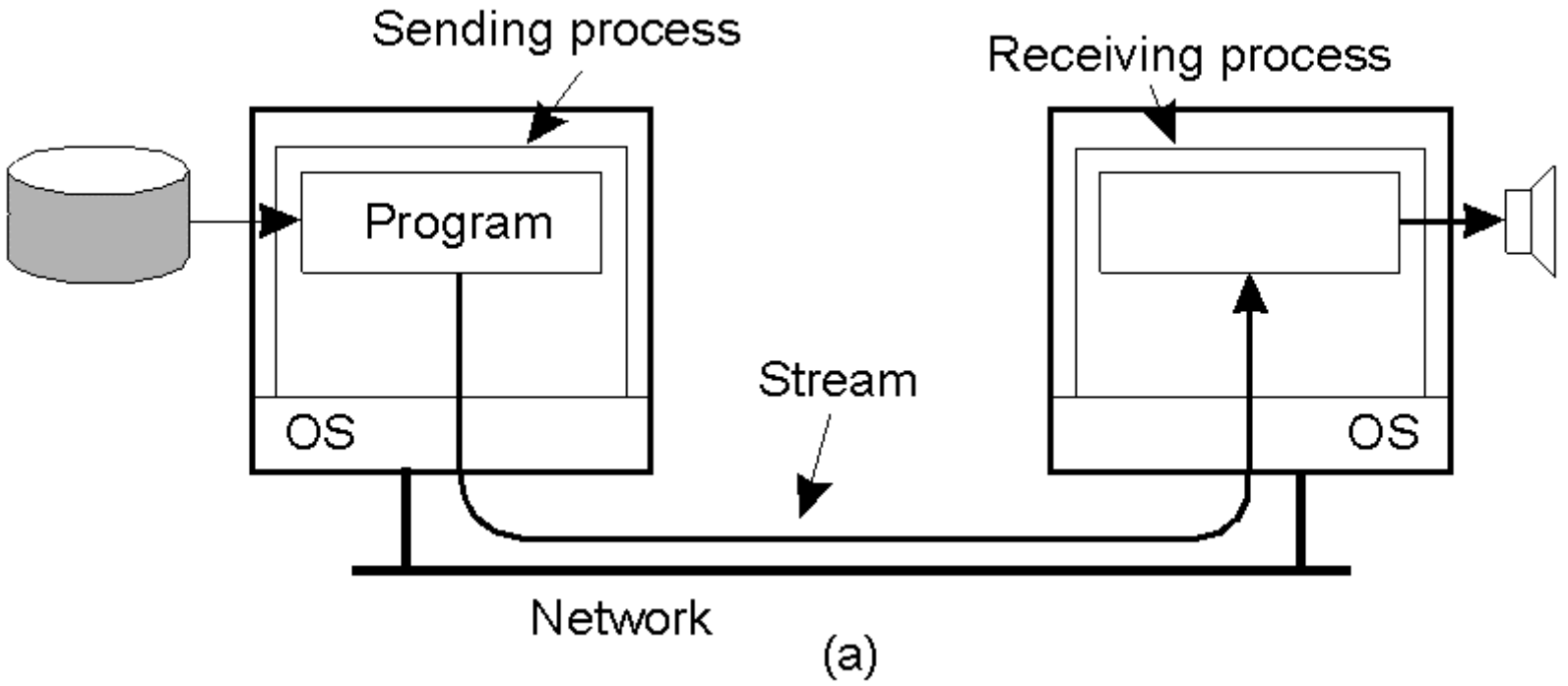
- A data stream is sequence of data units
- Discrete data streams:
  - TCP/IP connections, UNIX pipes
- **Transmission modes:** Different timing guarantees with respect to data transfer
  - **Asynchronous:** no restriction with respect to *when* data is to be delivered
  - **Synchronous:** define a maximum end-to-end delay for individual packets
  - **Isochronous:** define maximum and minimum end-to-end delays (*bounded jitter*)

Streams (1)

- **Definition:** A (continuous) data stream is a connection-oriented communication facility that supports isochronous data transmission
- **Stream characteristics:**
  - Streams are unidirectional
  - Generally a single **source**, one or more **sinks**
  - either the sink and source could be a wrapper around hardware (e.g. camera, CD device, TV monitor)
- **Stream types:**
  - **Simple:** consists of a single flow of data, e.g. audio or video
  - **Complex**: multiple data flows, e.g. stereo audio and movie ➔substreams (synchronization of substreams is important)
  - Relation between the substreams are complex and time-dependent

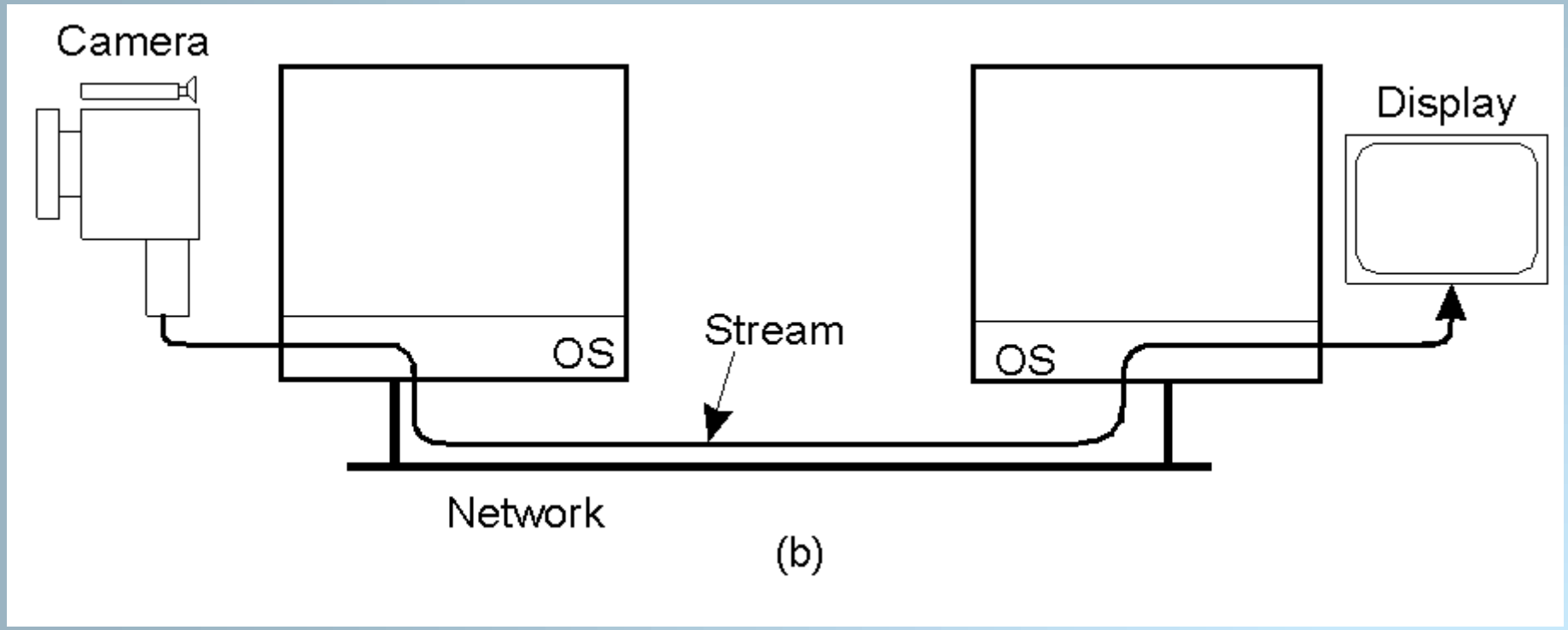
Streams (2)

Setting up a stream between two processes across a network.



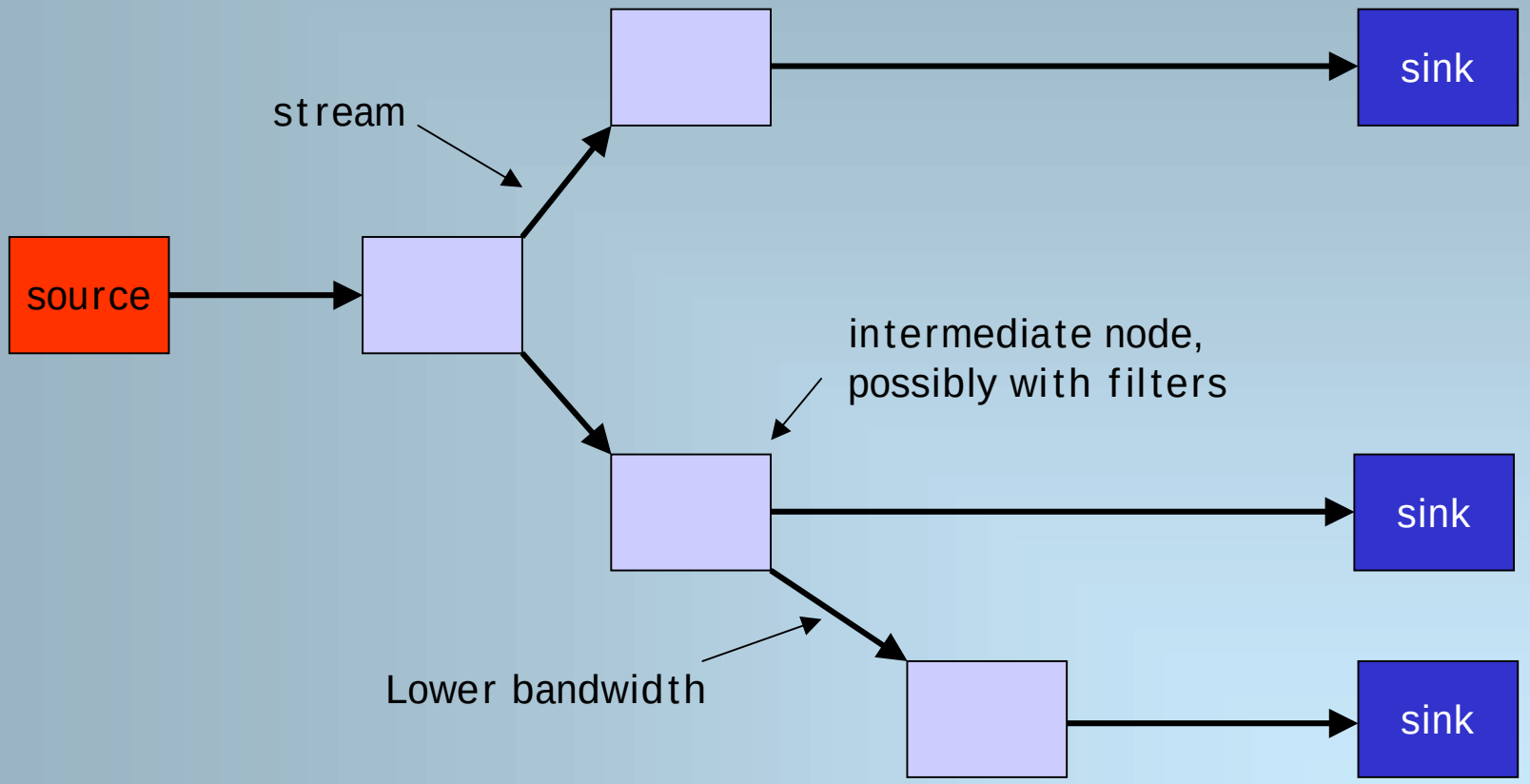
Streams (3)

Setting up a stream directly between two devices



Streams (4)

An example of multicasting a stream to several receivers.



Issue: receivers with different requirements with respect to the quality of the stream

Streams & QoS

- **Essence:** Streams are all about timely delivery of data
  - Time dependent (or other nonfunctional) requirements are generally expressed as **Quality of Service** (**QoS**) requirements
  - These requirements describe what is needed from the underlying distributed system in order to preserve the temporal relationship in a stream
  - QoS for continuous data streams mainly concerns *timeliness*, *volume*, and *reliability*
  - Make distinction between **specification** and **implementation** of QoS

Specifying QoS

- **Provide a precise Flow Specification:** bandwidth requirements, transmission rates, delays, etc

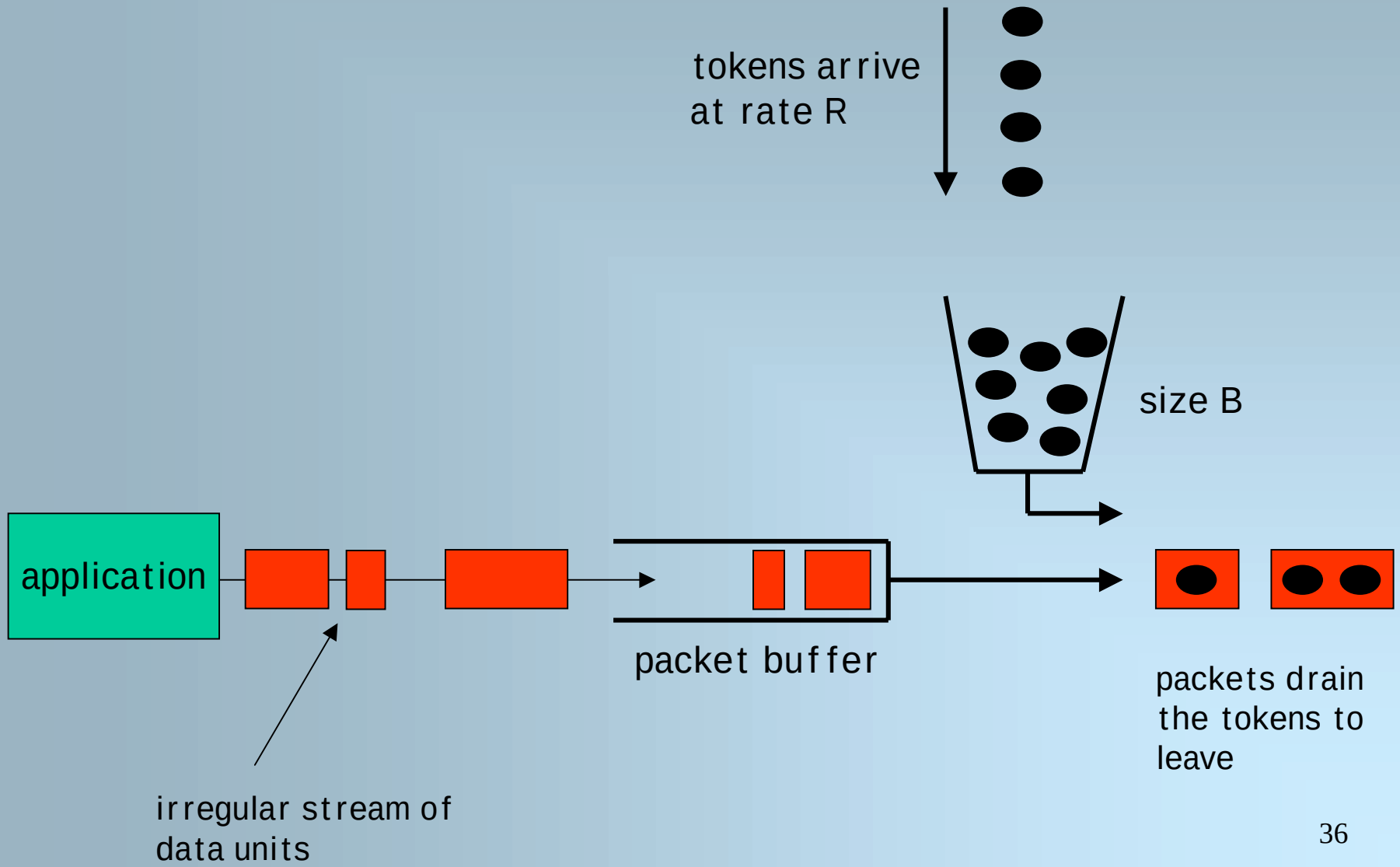
Characteristics of the Input	Service Required
• maximum data unit size (bytes) • Token bucket rate – R (bytes/sec) • Token bucket size – B (bytes) • Maximum transmission rate (bytes/sec)	• Loss sensitivity (bytes) • Loss interval (µsec) • Burst loss sensitivity (data units) • Minimum delay noticed (µsec) • Maximum delay variation (µsec) • Quality of guarantee

Partridge's Model

- In Partridge's model, the characteristics of the stream are formulated in terms of token bucket algorithm,
- Token bucket algorithm specifies how the stream will shape its network traffic
- The basic idea is that tokens are generated at a constant rate and buffered in a bucket
- Bucket has a limited capacity
- A token represents a fixed number of bytes
- When the bucket is full, tokens will be dropped

Token Bucket Algorithm (1)

The principle



Token Bucket Algorithm (2)

- Each time an application wants to send a data size of S it removes sufficient tokens from the bucket that jointly represent at least S bytes.
- The effect is that the data is passed to the network at a relatively constant rate.
- Bursts of data may happen
  - Rt+B over a time interval of t
- Token bucket algorithm is used in network routers and gateways to perform traffic shaping or policing
- In a flow specification, the application will offer data units to the communication system according to the output of the token bucket algorithm

Specifying QoS: Services Required

- Loss sensitivity + loss interval specifies maximum acceptable loss rate (e.g. 1 byte per minute)
- burst loss sensitivity specifies how many consecutive data units may be lost
- Minimum delay noticed specifies how long the network can delay the delivery of a data unit before the receiver notices the delay
- Maximum delay variation specifies the maximum tolerated jitter
- Quality of guarantee is a number indicating how serious the service requirements should be taken
  - Low number indicates no harm done, if the communication system cannot provide the required services

A Flow Spec: Internet RFC 1363

Eleven 16-bit numeric values

	Protocol Version
Bandwidth	Maximum transmission unit
	Token bucket rate
	token bucket size
	maximum transmission rate
Delay	minimum delay noticed
	maximum delay variation
Loss	Loss sensitivity
	Burst loss sensitivity
	loss interval
	Quality of guarantee

Implementing QoS

- Essence: QoS specifications translate to resource reservations in underlying communication system
  - Resources: Bandwidth, buffers, processing capacity
- Problem: There is no single best model for
  1. Specifying QoS parameters
  2. Generically describing resources in any communication system
  3. Translating QoS parameters to resource usage
- QoS in distributed systems mainly dependent on services the underlying network can offer
- QoS Negotiation

QoS Negotiations (1)

- QoS manager at each node
- A source component initiates the negotiation by sending out the flow spec to its local QoS manager
- The flow spec is forwarded to the next node where the resources are required
- The flow spec traverses all the nodes until the final target is reached.

QoS Negotiations (2)

- The information on whether the desired QoS can be provided by the system is passed back to the source.
- Applications rarely have fixed QoS requirements
- If a stream has multiple sinks, the negotiation path forks according to the data flow.
  - Worst case QoS parameters
  - Largest loss rates, smallest bandwidth, longest delay
- Remedy is the receiver-initiated protocols

RSVP (1)

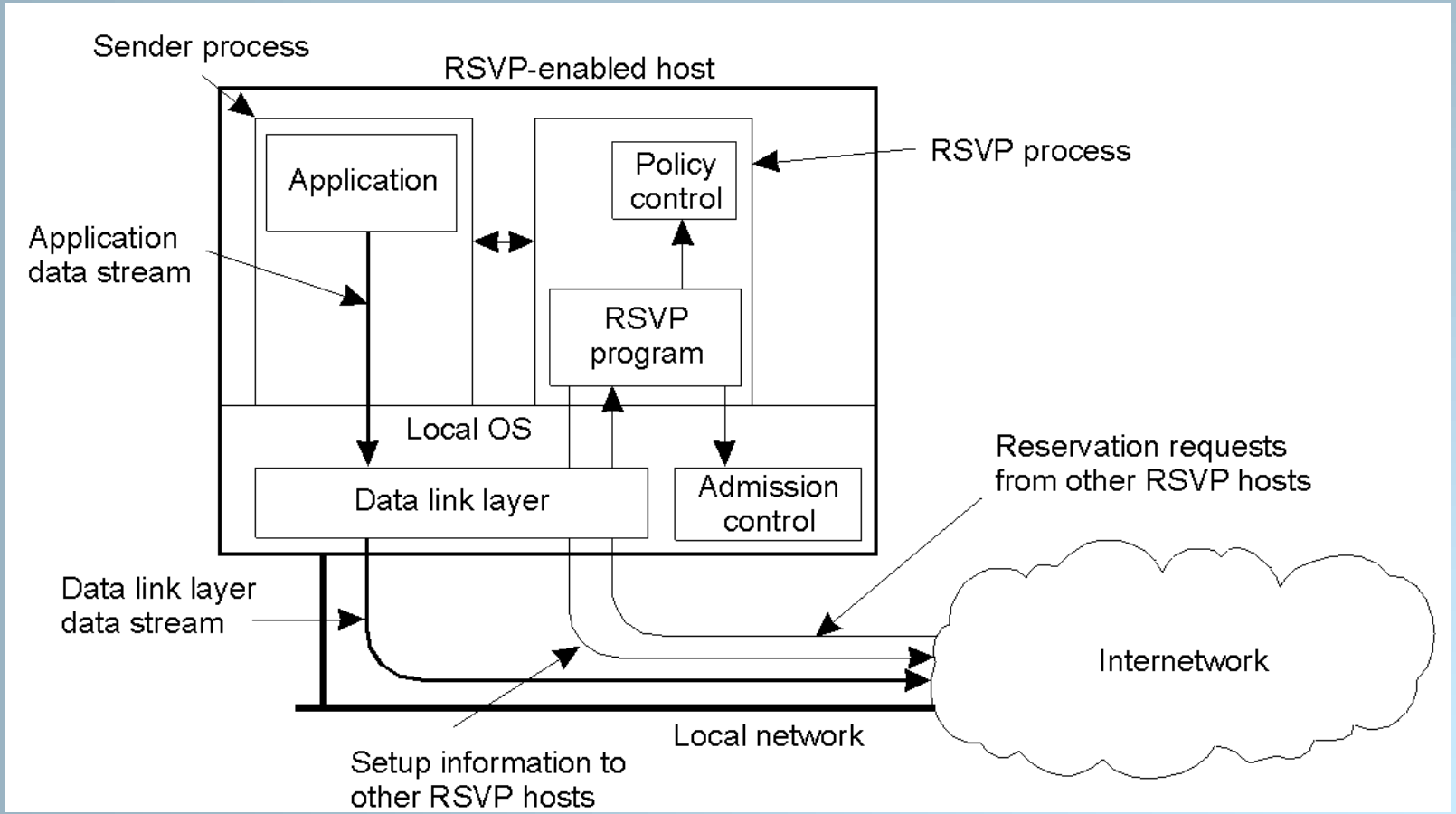
- Resource reSerVation Protocol
- Receiver-initiated QoS negotiation protocol
- RSVP prevents more resources from being allocated than are strictly necessary
- Senders provide flow specification characterizing the data in terms of bandwidth, delay, jitter, etc.
- A RSVP process that is co-located on the same machine as the sender stores this specification locally,
- Sender in RSVP first sets up a path to potential receivers and pass the flow specification to each intermediate node
- Then a target places a reservation request along its upstream path to the sender

RSVP (2)

- A RSVP process, on a node in the path where resources are needed, passes a reservation request to the admission control module to check if resources are available
- Policy control module checks if the receiver has the permission to make reservation

Setting Up a Stream

The basic organization of RSVP for resource reservation in a distributed system.



Stream Synchronization

- Problem: Given a complex stream, how do we keep the substreams in synch?
- Synchronization takes place at the level of data units
- High-quality Audio for Stereo effect
  - 16-bit samples, sampling rate is 44100 Hz
  - Synchronization could take place approx every 23 μs
- Video
  - NTSC displays video frames at a rate of 30 Hz
  - We can group the audio samples into logical units as long as a video frame is displayed (33 ms)
  - For lip synchronization, a logical audio data unit can be as large as 1470 samples

Synchronization Mechanisms

- Issues:
  - The basic mechanisms for synchronizing two streams
  - The distribution of those mechanisms in a networked environment
- One way is to achieve synchronization is operating on data units of simple streams
  - A process that executes read and write operations on several simple streams, ensure that those operations adhere to specific timing constraints

Example: Audio/Video Sync.

- **Video:**
  - Low-quality image of 320×240 pixels ➔76800 bytes
  - Frame rate of 30 Hz (or one image every 33 ms)
- **Audio:**
  - From previous example, the size of logical data unit corresponding 33 ms of audio is 2940 bytes
- Application should be able to process
  - (5880 + 76800) bytes of data every 33 ms or
  - ~ 2.5 Mbytes of data every second
- Process handles the synchronization by simply alternating between reading an image and reading a block of audio data samples every 33 ms

Distribution of Synchronization

- Receiving side of a complex stream needs to have a complete synchronization specification locally available
- Common practice is to multiplex substreams into a single stream containing all data units, including those for synchronization
  - MPEG (Motion Picture Experts Group)
  - In MPEG-2, an unlimited number of continuous and discrete streams can be merged into a single stream
  - Each input stream is first turned into a stream of packets that carry a timestamp based on 90-kHz system clock
  - These substreams are subsequently multiplexed into a program stream consisting of variable length packets, but they all the same time base