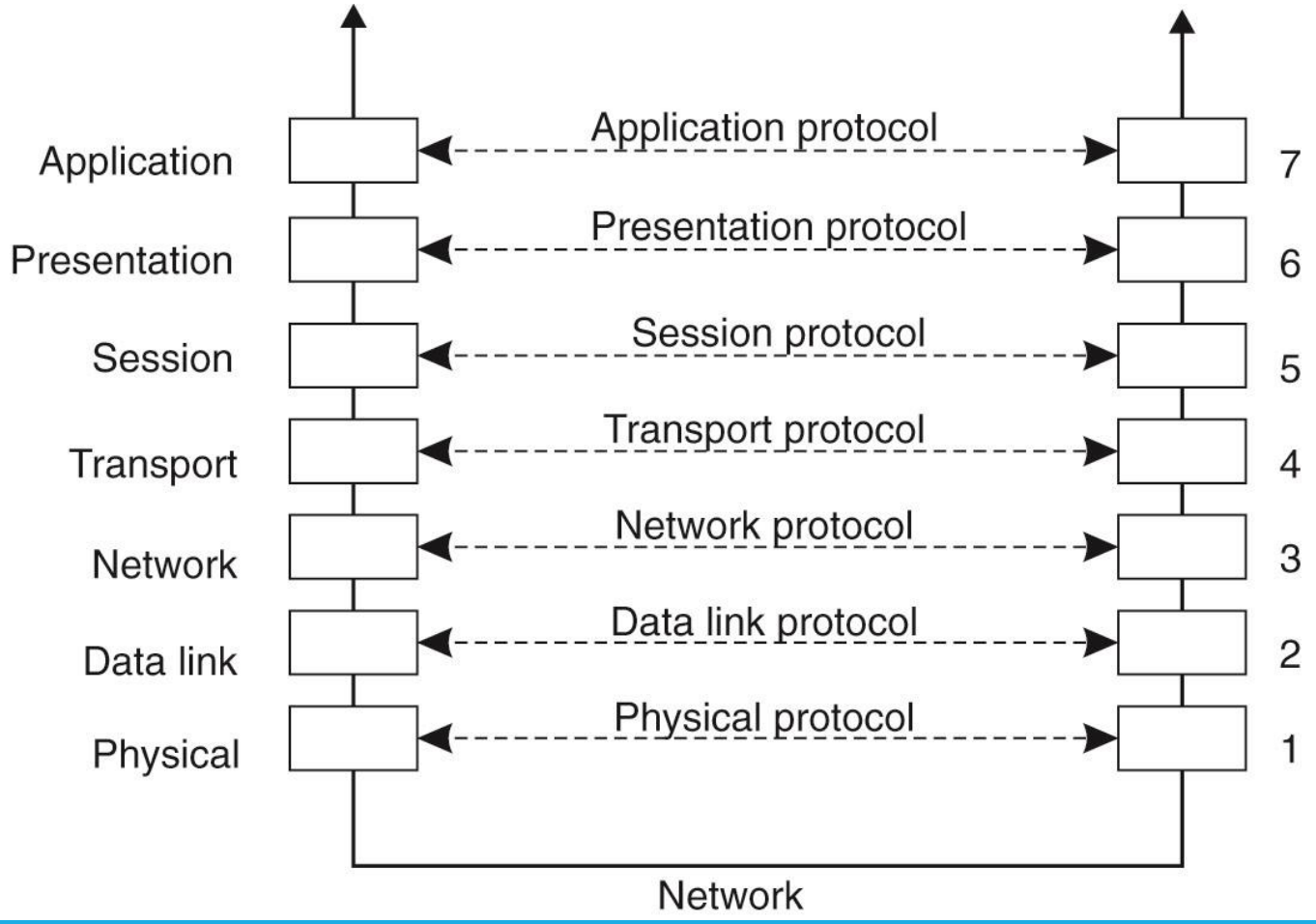
DISTRIBUTED SYSTEMS
Principles and Paradigms

Chapter 4
Communication

Layered Protocols (OSI Model)



Lower Layers

- **Physical Layer**
  - Contains the specification and implementation of **bit**, and their **transmission** between **sender and receiver**
- **Data link layer**

  -prescribes the transmission of a series of **bits into frame** to allow for **error** and **flow control**.
- **Network layer**

  -Describes how **packets** in a network of computers are to be **routed**

Transport Layer

Important

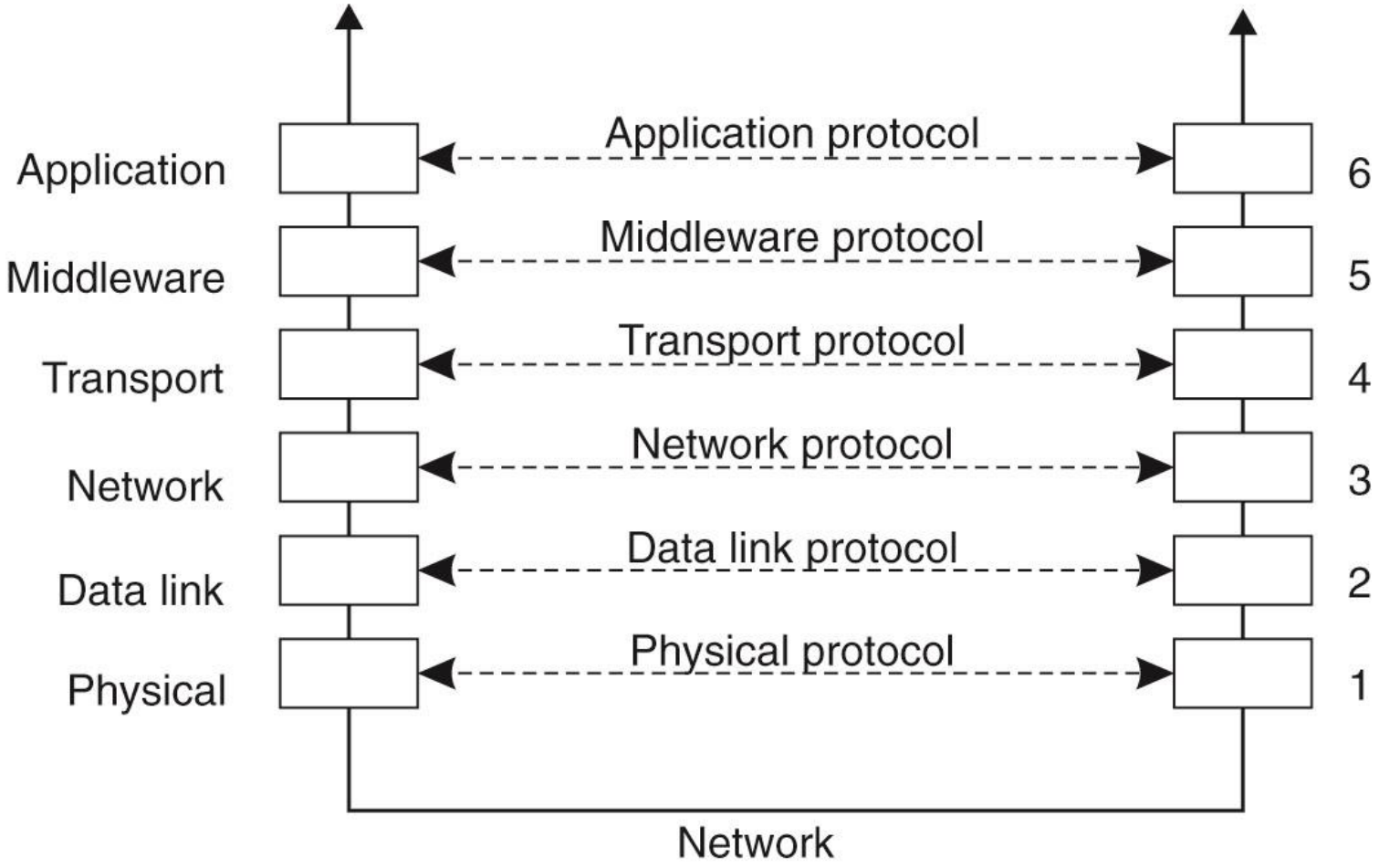
-The transport layer provides **the actual communication** facilities for most **distributed systems**.

Standard Internet protocols

-**TCP:** connection-oriented, reliable, stream-oriented communication

-**UPD:** unreliable(best-effort) datagram communication

Middleware Protocols



Middleware Layer

Middleware is invented to provide **common services and protocols** that can be used by many different applications

-A rich set of communication protocols

-Authentication and Authorization

-Naming protocols, to allow easy sharing of resources

-Security protocols for secure communication

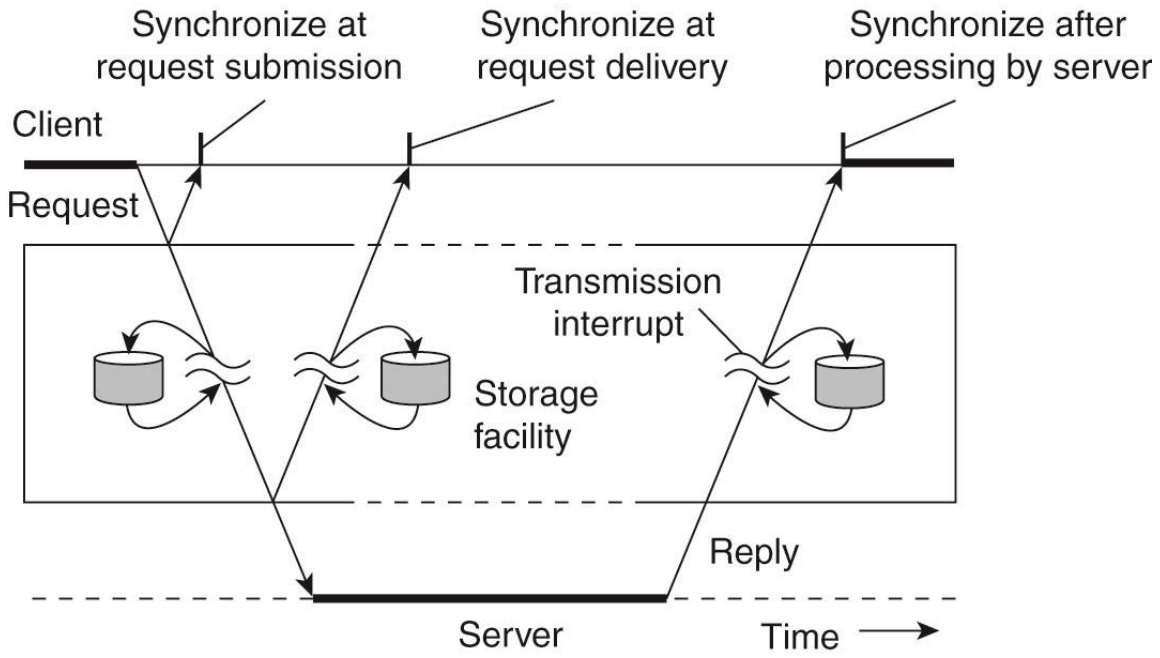
-Distributed commit and distributed lock

Types of Communication

Distinguish

-**Transient** versus **persistent** communication

-**Asynchronous** versus **synchronous** communication



Types of Communication

- Transient versus persistent
  - **Transient communication**: communication server discards message when it cannot be delivered at the next server, or at the receiver,
  - **persistent communication**: A message is stored at a communication server as long as it takes to deliver it.
- Places for synchronization
  - At request submission
  - At request delivery
  - After request processing

Client / Server Systems

Some observations

-Client/Server computing is generally based on a model of **transient synchronous** communication:

.Client and server have to **be active at time of common**

.Client issues request and block until it receives reply

.Server essentially waits only for incoming request, and subsequently processes them

-Drawbacks synchronous communication

.Client cannot do any other work while waiting for reply

.Failures have to be handled immediately

.the model may simply not be appropriate(mail, news)

Messaging

Message-oriented middleware

Aims at high-level persistent asynchronous communication:

-Processes send each other messages, which are queued

-Sender need not wait for immediate reply, but can do other things

-Middleware often ensures fault tolerance

Remote Procedure Call(RPC)

Observations

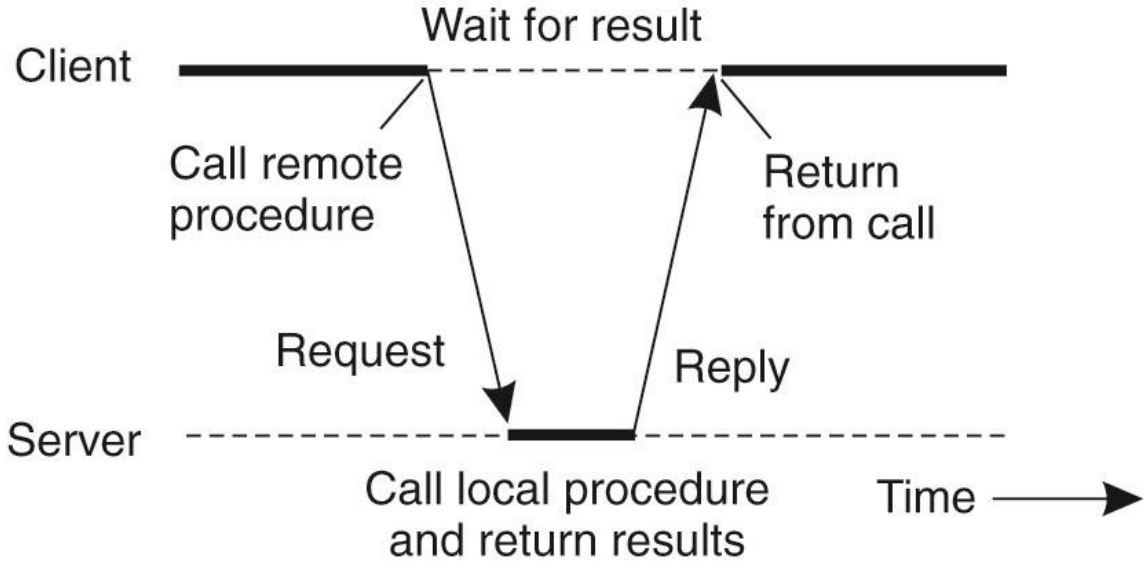
-Application developers are familiar with simple procedure model

-Well-engineered procedures operate in isolation(black box)

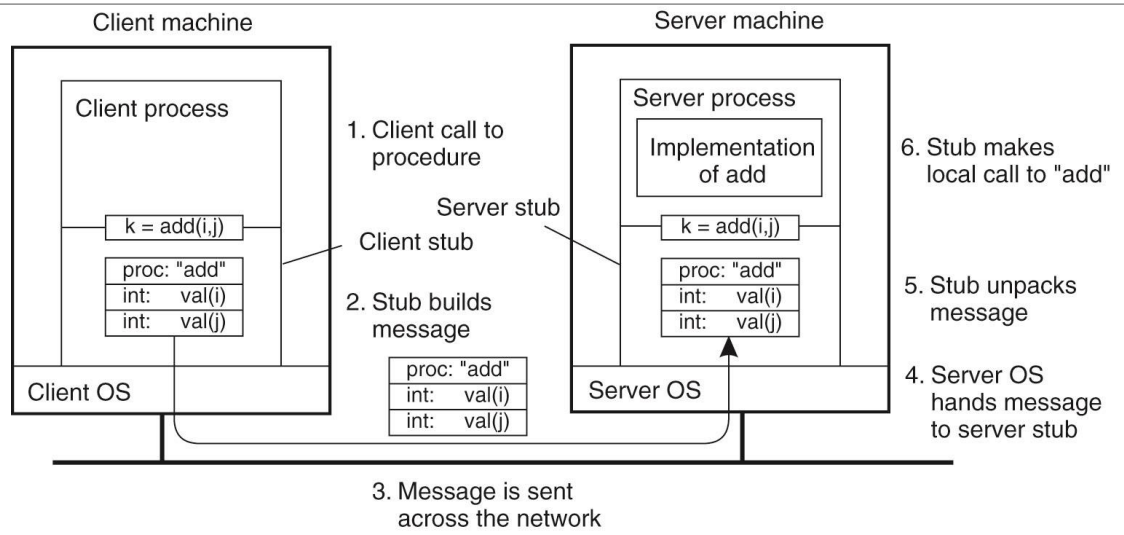
There is no fundamental reason not to execute procedures on separate machine

.Conclusion

-Communication between caller & callee can be hidden by using Procedure call mechanism



Remote Procedure Calls Operation



1. client procedure calls the client stub
2. stub builds a message and calls the local OS
3. OS sends the message to the remote OS.
4. remote OS gives the message to the server stub.
5. The server stub unpacks the parameters and calls the server.
6. server returns the result to the stub.
7. stub builds message and calls its local OS.
8. The server's OS sends the message to the client's OS.
9. The client's OS gives the message to the client stub.
10. The stub unpacks the result and returns to the client.

RPC: Parameter passing

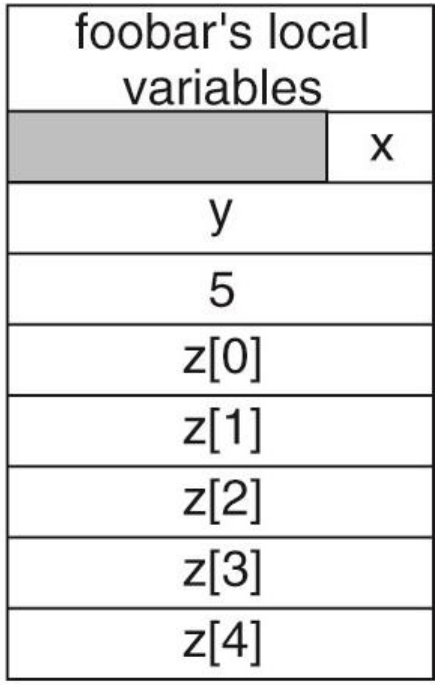
Parameter marshaling

- There's more than just wrapping parameters into a message:
  - Client and server machines may have different data representations(think of byte ordering)
  - Wrapping a parameter means transforming a value into a sequence of bytes
  - Client and sever have to agree on the same encoding
    - How are basic data value represented(integer, floats, characters)
    - How are complex data values represented(arrays, unions)
- Client and server need to properly interpret messages.

Parameter Specification and Stub Generation

```
foobar( char x; float y; int z[5] )
{
   ....
}
```

(a)



(b)

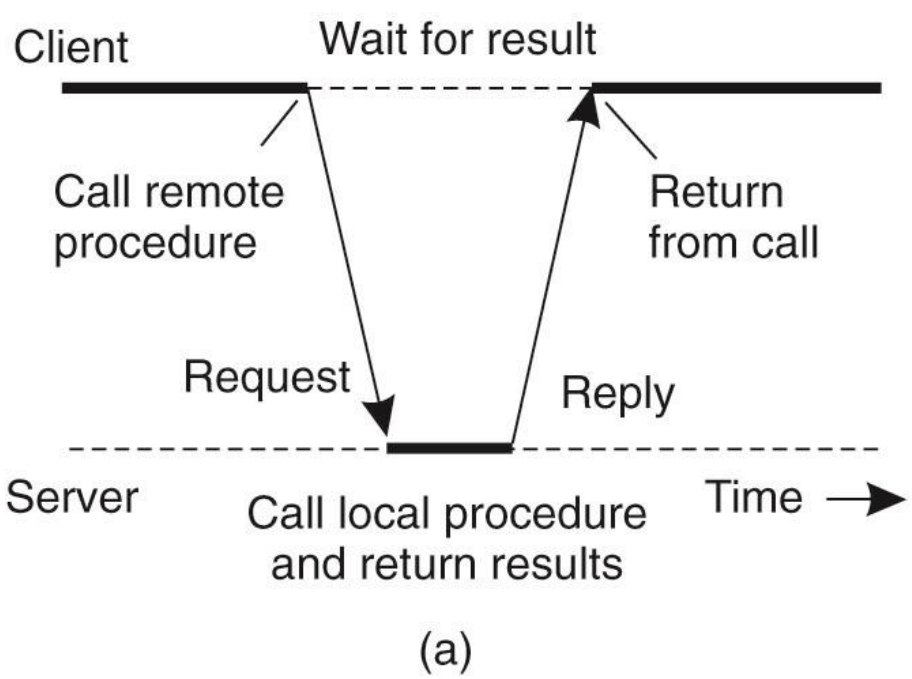
Reference and Pointer Parameters

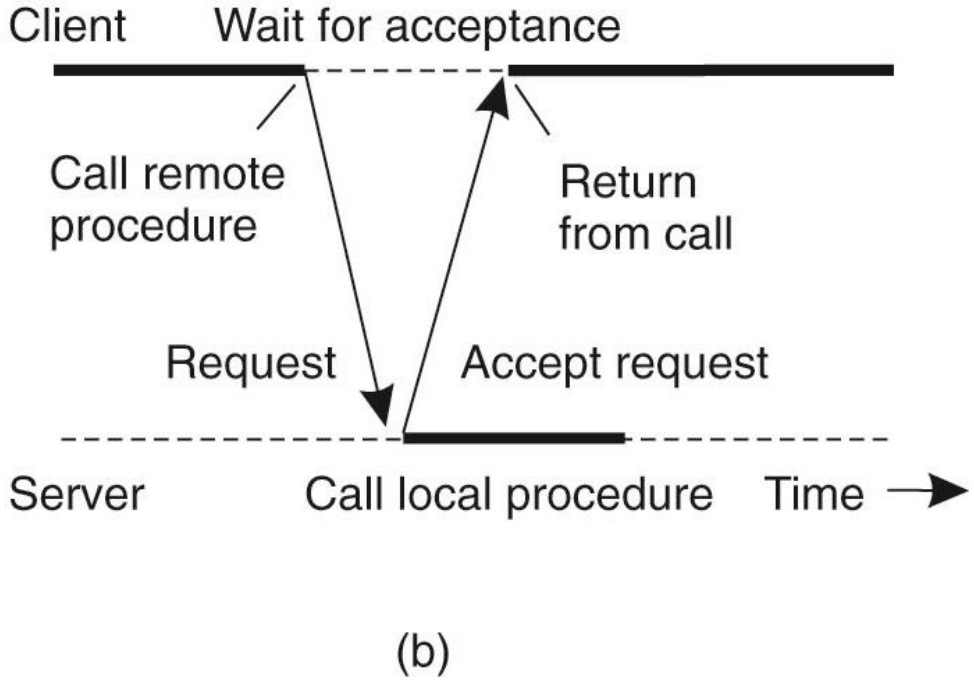
How to pass reference and pointers

- One solution: replace call-by-reference by **copy/restore**
- Handles pointers to simple arrays and structures, but cannot handle the general case of an arbitrary data structure such as graph
- Another solution: **remote references,** i. e sending the real pointer to the server and generating special codes in the server to access client data

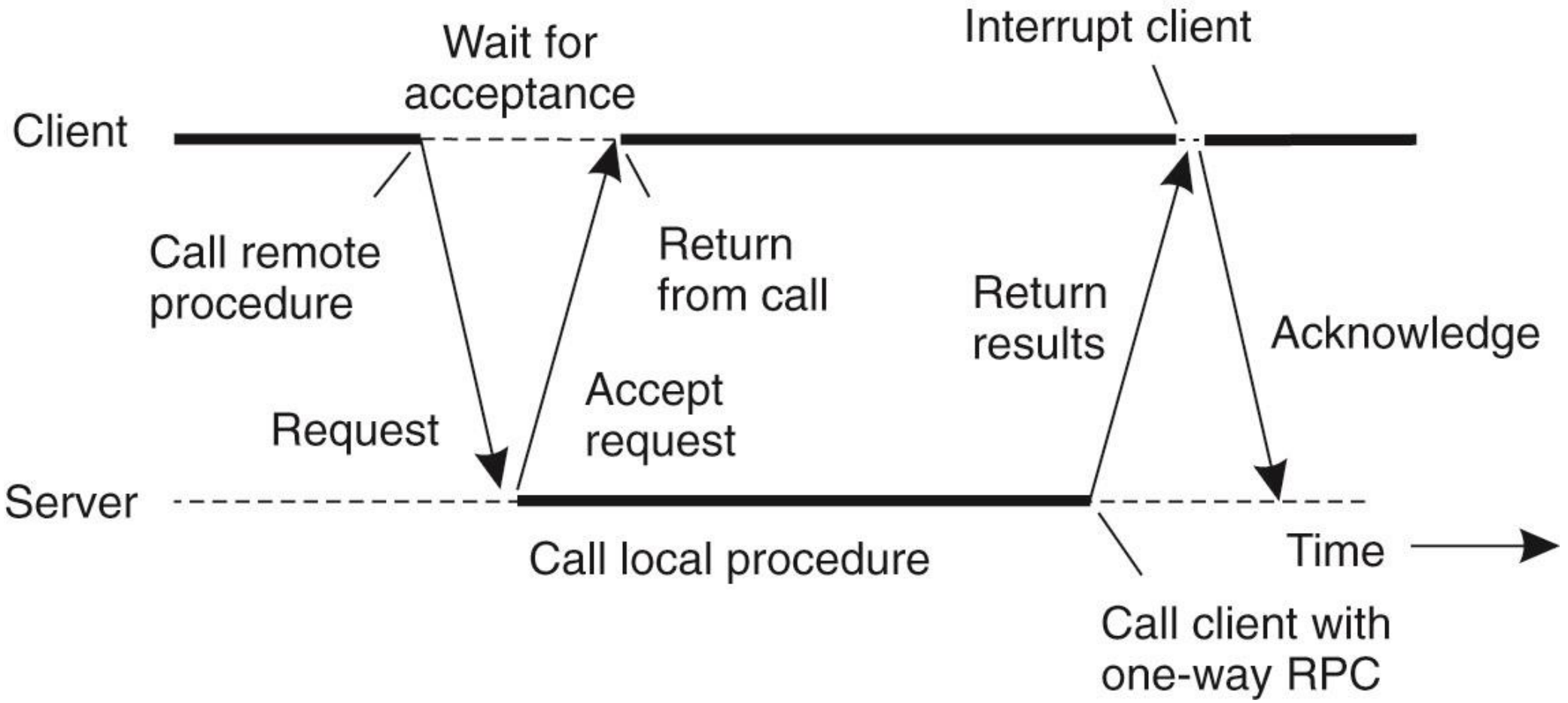
Asynchronous RPC

Try to get rid of the strict request- reply behavior, but let the client continue without waiting for an answer from the server.





Asynchronous RPC Deferred synchronous RPCS



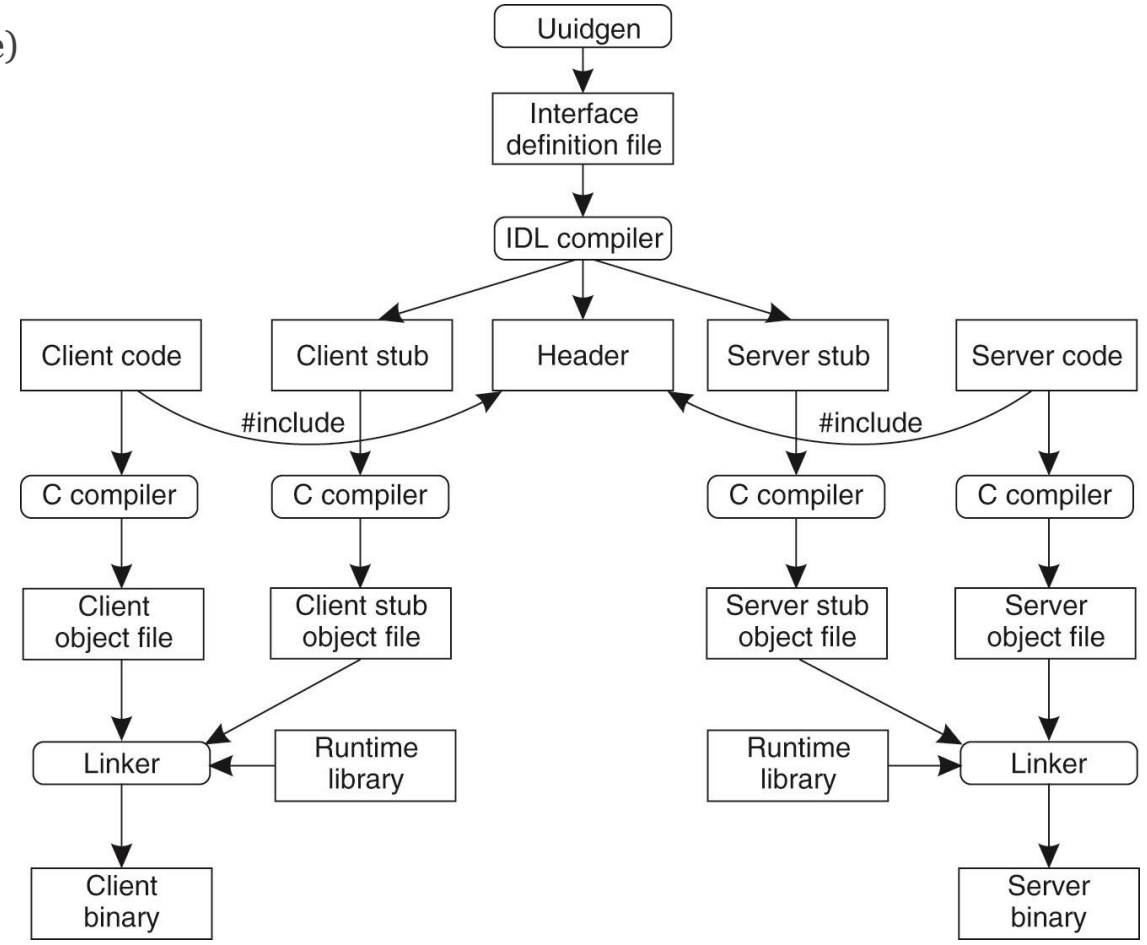
RPC in Practice

Distributed Computing Environment(DCE) RPC

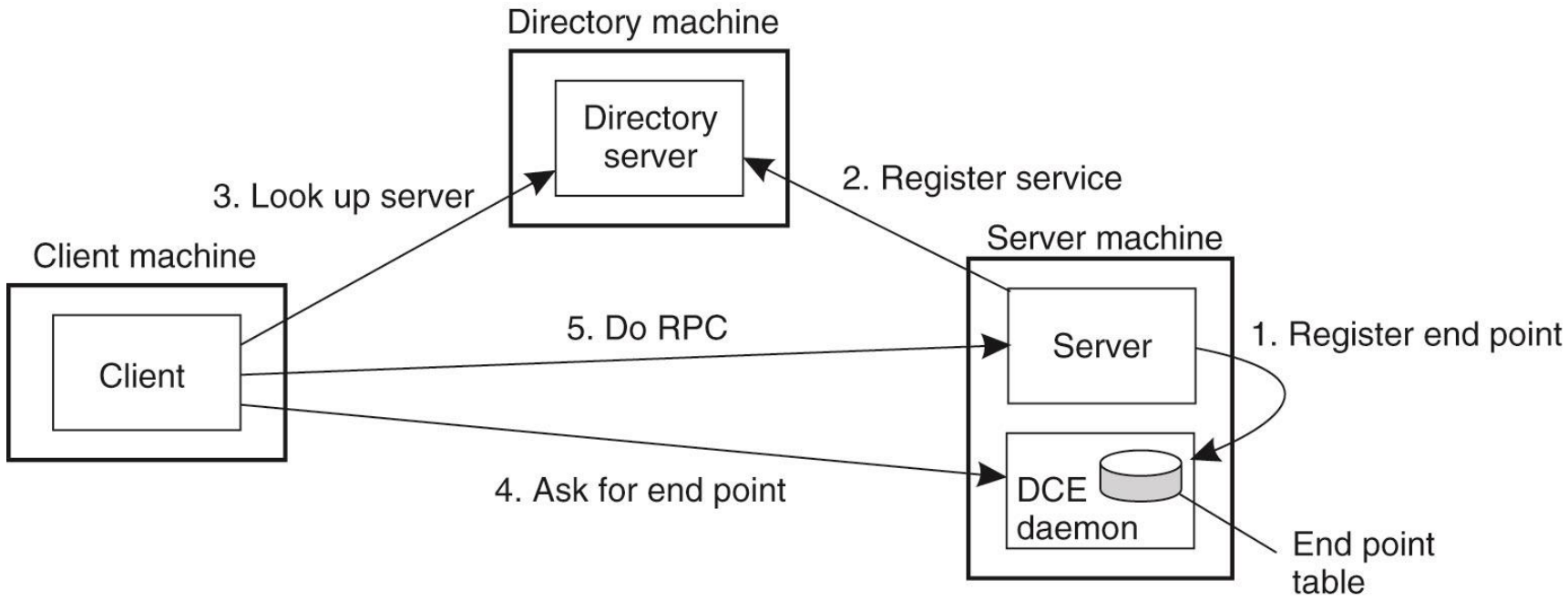
- DCE is a true middleware system in that it is designed to execute as a layer of abstraction between existing (network) operating systems and distributed applications.
- Developed by **Open software foundation**(OSF), now called the **Open Group**
- **Microsoft** used DCE RPC in **DCOM**.

DCE RPC Writing a Client and a Server

IDL(Interface Definition Language)



DCE RPC Binding a Client to a Server



Message Oriented Communication

Transient Messaging

Message Queuing System

Message Brokers

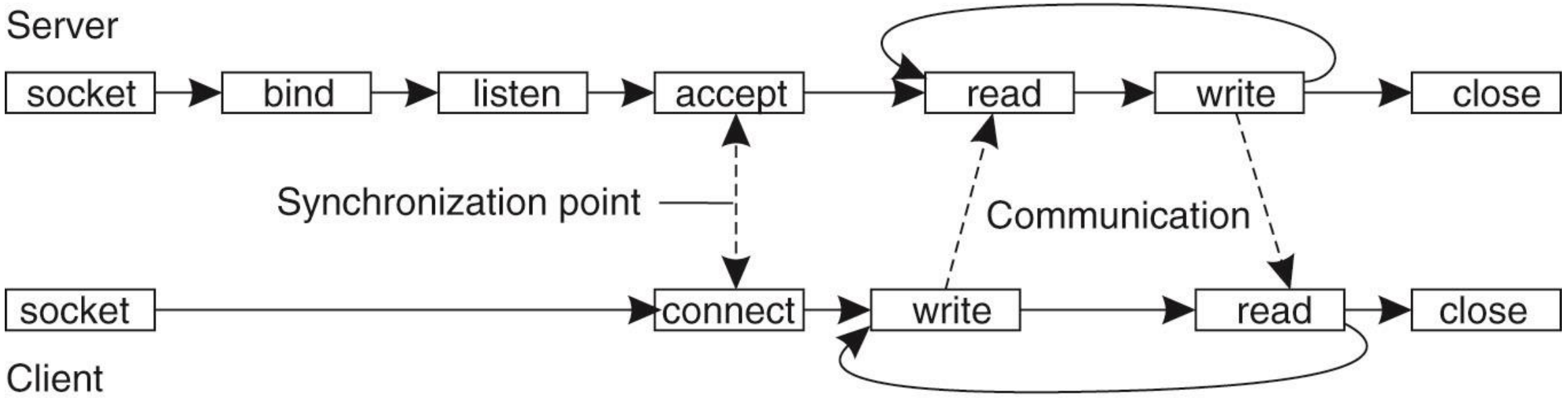
Example: IBM WebSphere

Transient Messaging : Sockets

The socket primitives for TCP/IP.

Primitive	Meaning
Socket	Create a new communication end point
Bind	Attach a local address to a socket
Listen	Announce willingness to accept connections
Accept	Block caller until a connection request arrives
Connect	Actively attempt to establish a connection
Send	Send some data over the connection
Receive	Receive some data over the connection
Close	Release the connection

Transient Messaging : Sockets



Transient Messaging: The Message-Passing Interface (MPI)

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 is none
MPI_irecv	Check if there is an incoming message, but do not block

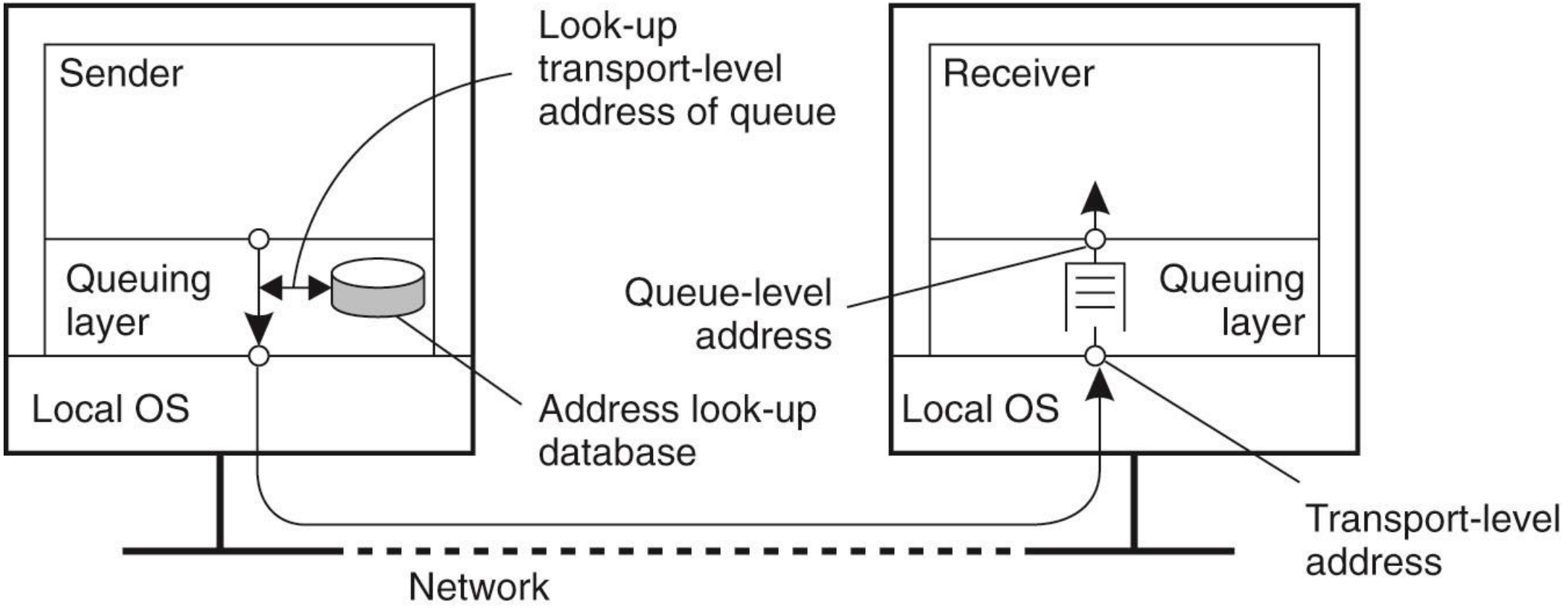
Message-Oriented Middleware(MOM)

Asynchronous persistent communication through support of middleware-level queues.

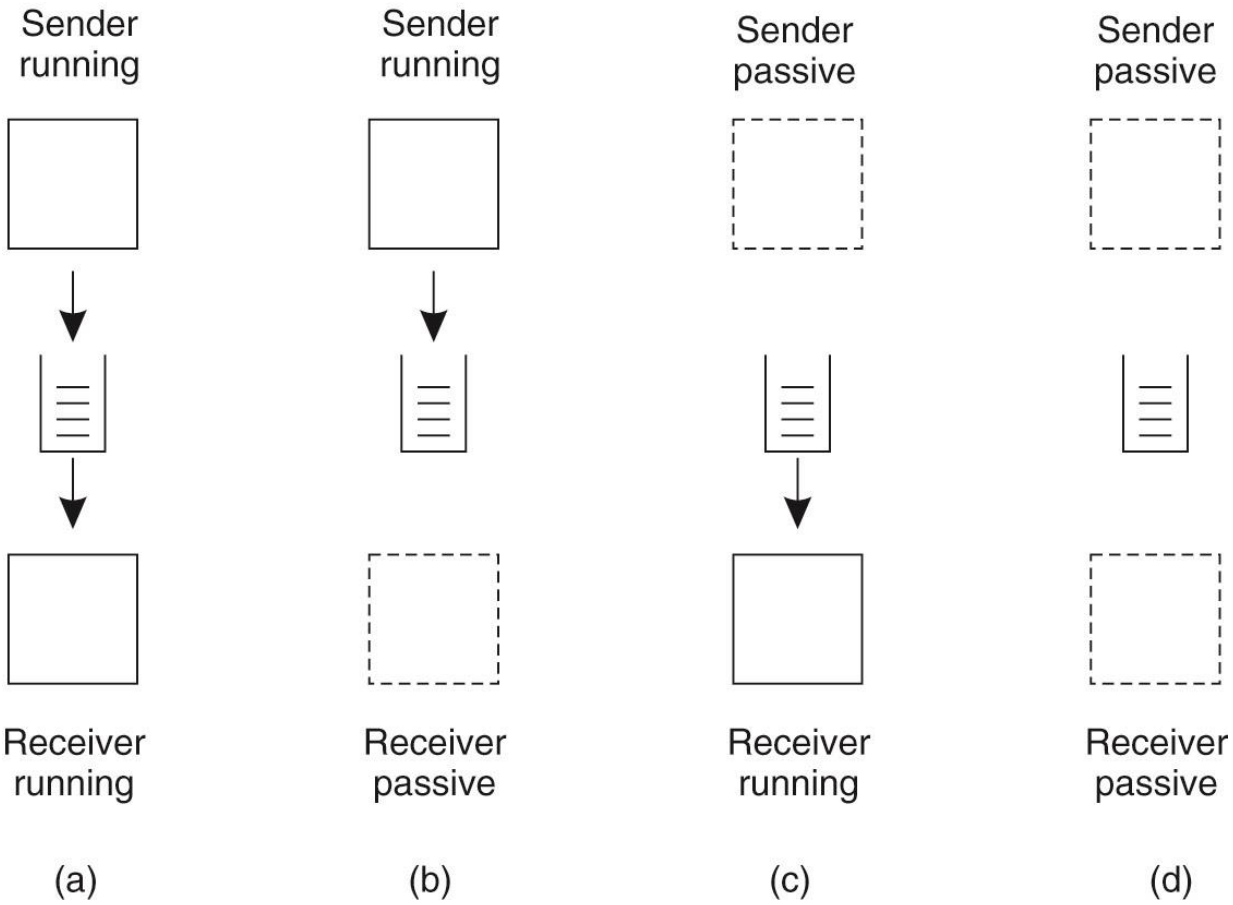
Queues correspond to buffers at communication servers

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

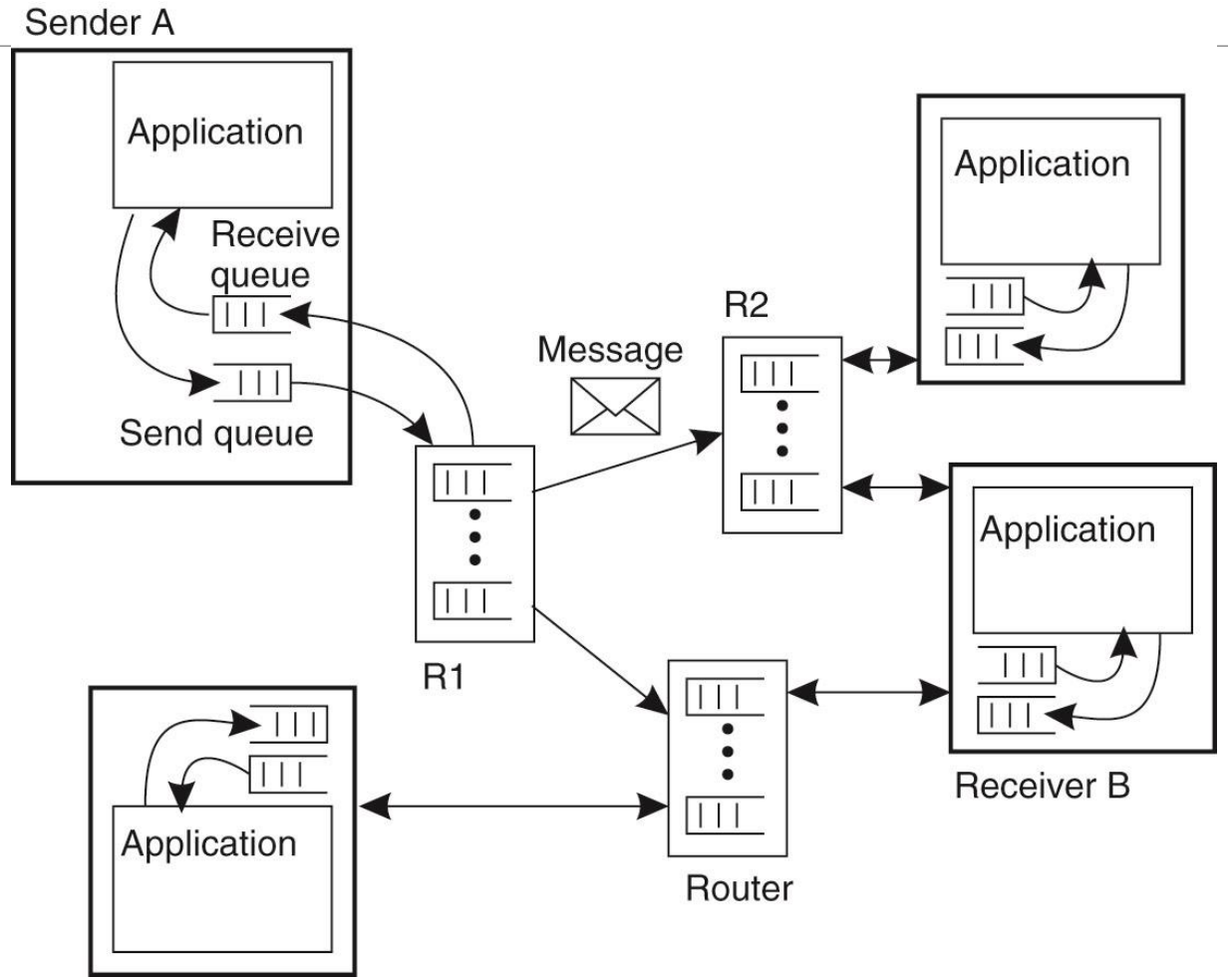
General Architecture of a Message-Queuing System



Message-Oriented Middleware(MOM)



General Architecture of a Message-Queuing System with Routers



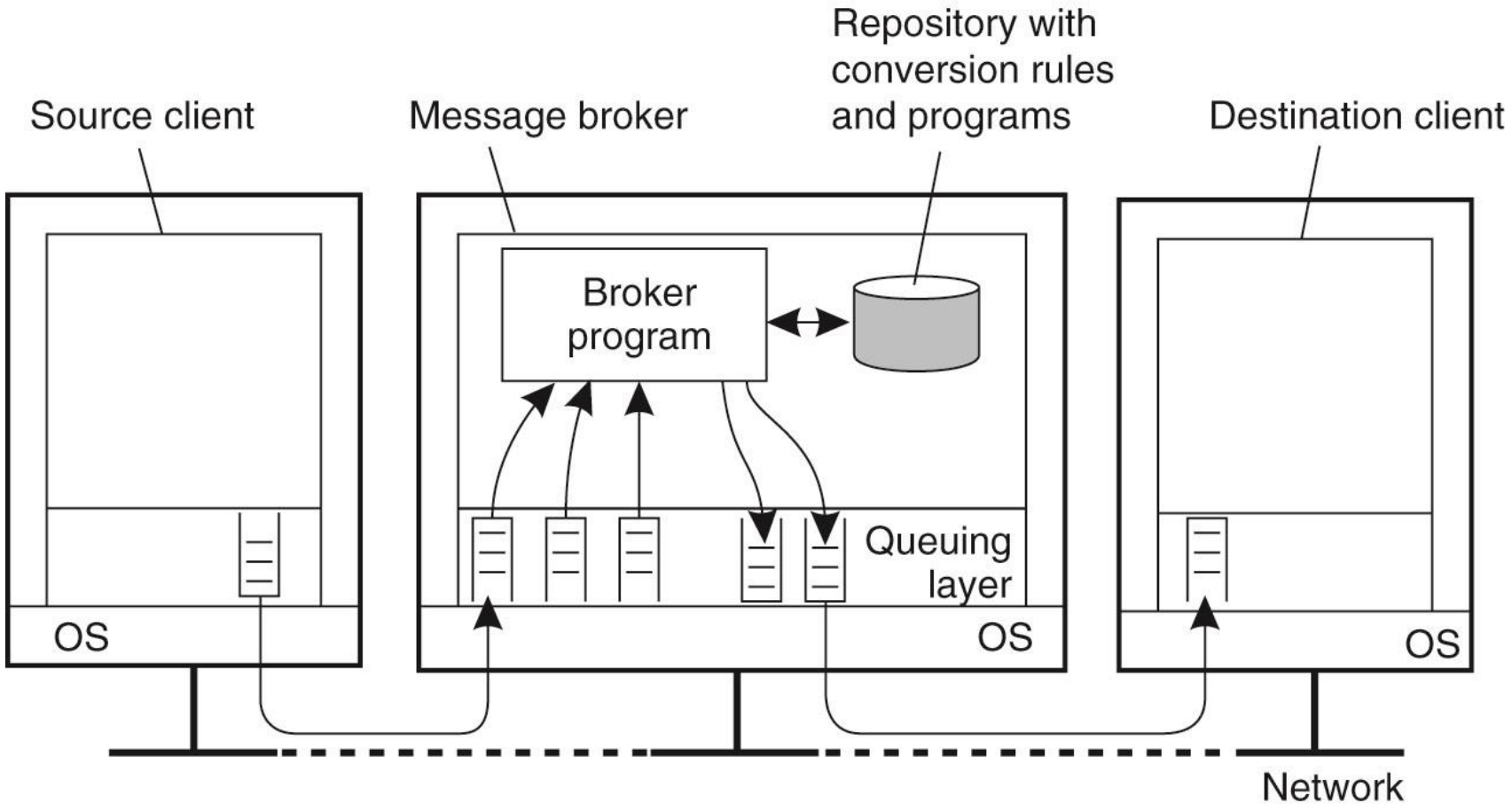
Message Broker

Message queuing systems assume a common messaging protocol: all applications agree on message format(i.e. structure and data representation)

Message Broker

- Centralized component that takes care of application heterogeneity in an MQ system
  - Transform incoming messages to target format
  - Very often acts as an application gateway
  - Enterprise application Integration: uses publish/subscribe model

Message Brokers



IBM's WebSphere MQ

Basic concepts

- Application specific message are put into, and removed from queues.
- Queues reside under the regime of a queue manager
- Processes can put messages only in local queues, through an RPC mechanism

IBM's WebSphere MQ

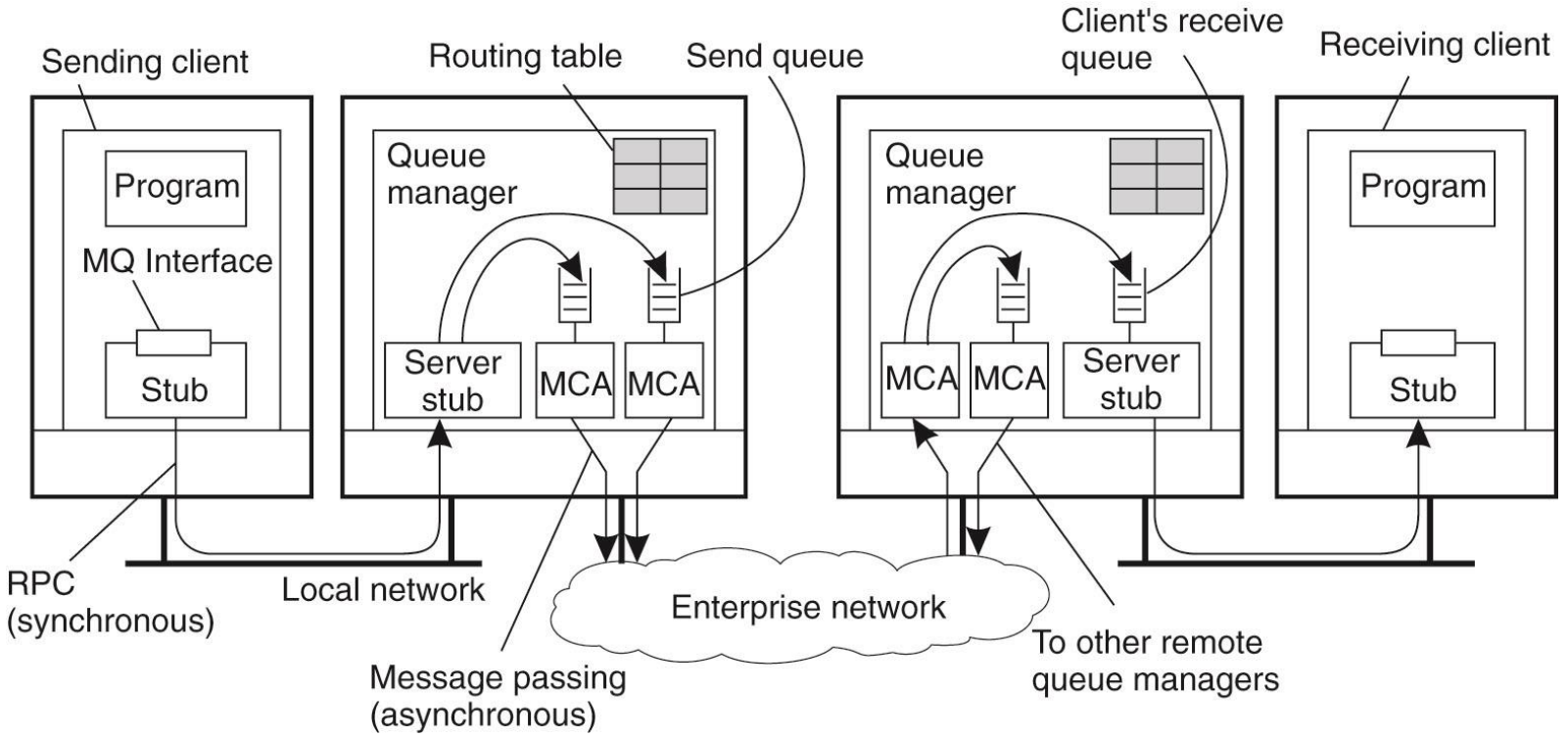
Message transfer

- Messages are transferred between queues
- Message transfer between queues at different processes, requires a **channel**
- At each endpoint of channel is a **message channel agent(MCA)**
- Message channel agents are responsible for:
- Setting up channels using lower-level network communication facilities(e.g, TCP/IP)
- (Un)wrapping messages from/in transport level packets
- Sending/receiving packets

IBM's WebSphere Message-Queuing System

Channels are inherently unidirectional

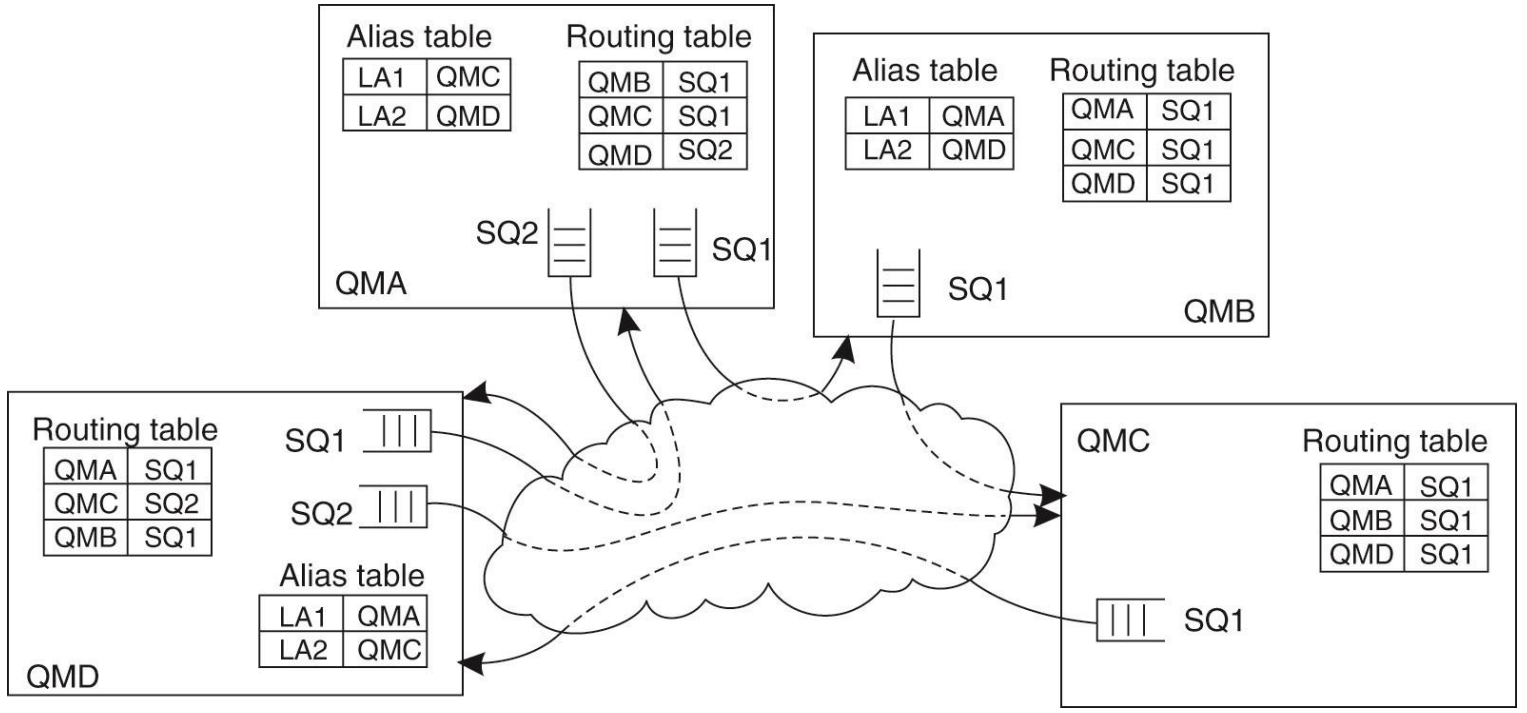
Routes are set up manually(system administration)



IBM's WebSphere MQ

Routing

- By using logical names, in combination with name resolution to local queues, it is possible to put a message in a remote queues.



Primitives in the Message queuing Interface

Primitive	Description
MQopen	Open a (possibly remote) queue
MQclose	Close a queue
MQput	Put a message into an opened queue
MQget	Get a message from a (local) queue

Stream oriented Communication

All communication facilities discussed so far are essentially based on a discrete, that is time independent exchange of information.

Continuous Media

- Characterized by the fact that values are time dependent
  - Audio
  - Video
  - Animation
  - Sensor data(temperature, pressure , etc)

Continuous Media

Transmission modes

Different timing guarantees with respect to data transfer:

- Asynchronous: no restrictions with respect to when data is to be delivered.
- Synchronous: define a maximum end to end delay for individual data packets.
- Isochronous: define a maximum and minimum end to end delay(jitter is bounded)

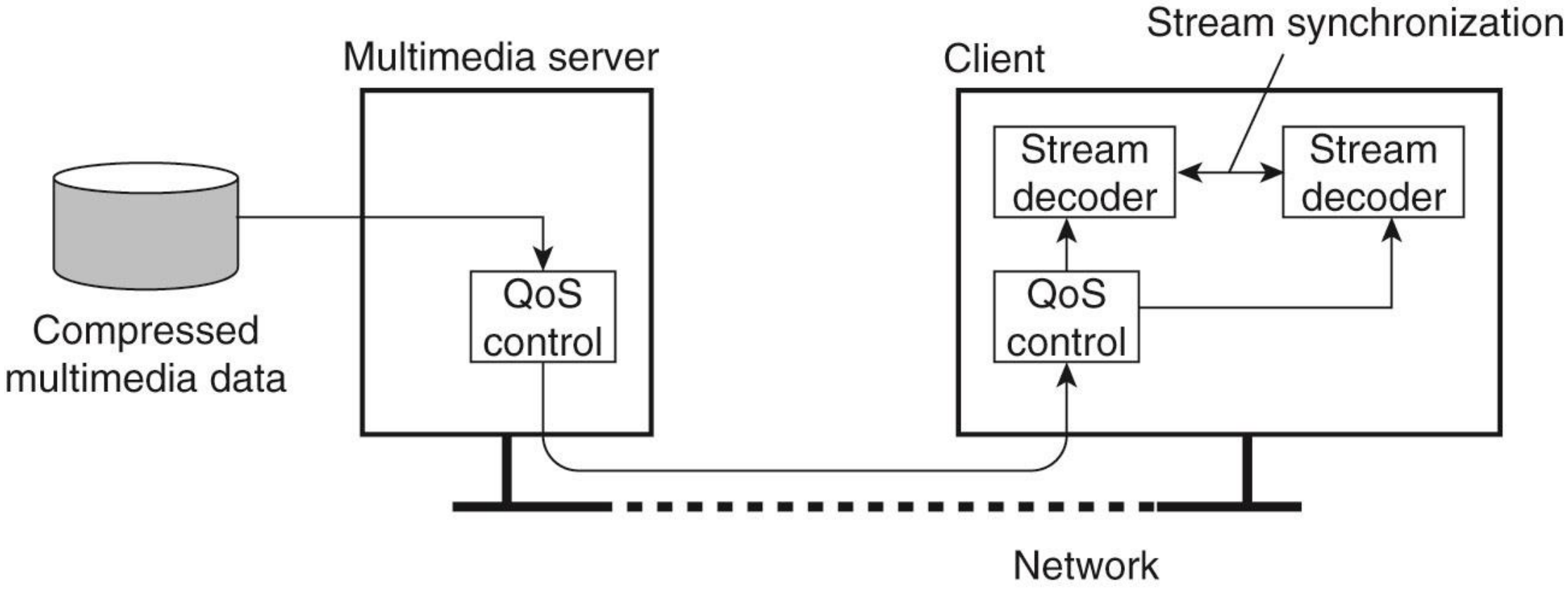
Data Stream

A(continuous) data stream is a connection oriented communication facility that supports isochronous data transmission

Some common stream characteristics

- Streams are unidirectional
- There is generally a single source, and one or more sinks
- Often, either the sink and / or source is a wrapper around hardware(e.g. camera, CD device , Tv Monitor)
- Simple stream: a single flow of data, e. g audio and video
- Complex stream: multiple data flows. e.g. stereo audio or combination audio/video

Data Stream


Stream synchronization
Multimedia server
Client
Stream decoder
Stream decoder
QoS control
QoS control
Compressed multimedia data
Network

Streams and QoS

Streams are all about timely delivery of data. How do you specify this **Quality of Service(QoS)**?

- The required bit rate at which data should be transported .
- The maximum delay until a session has been set up(i.e. when an application can start sending data)
- The maximum end to end delay(i.e. how long it take until a data unit makes it to a recipient)
- The maximum delay variance, or jitter.
- The maximum round trip delay

Enforcing QoS

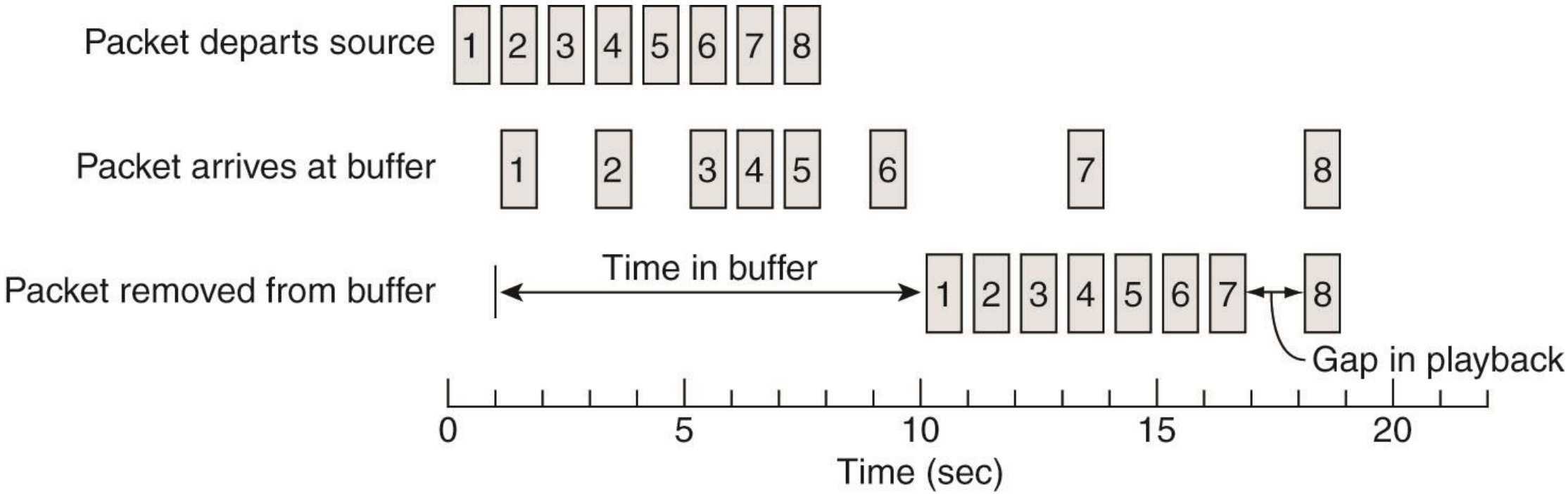
Network level tools

- Usually best effort, i.e has no mechanism for QoS
- There are some tools such as differentiated service by which certain packet can be periodized

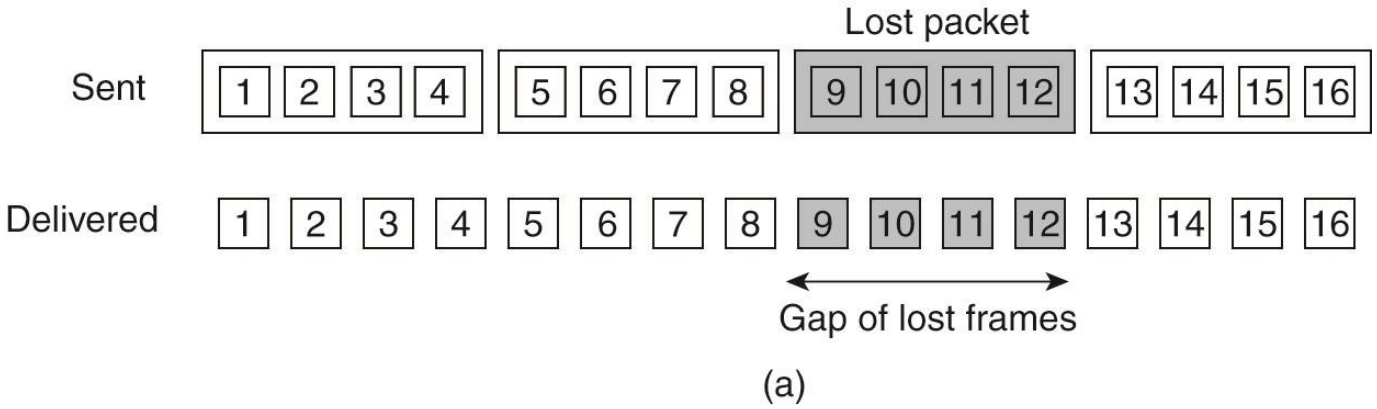
Distributed middleware level tools

- Using buffers
- Forward Error correction(FEC)
- interleaving

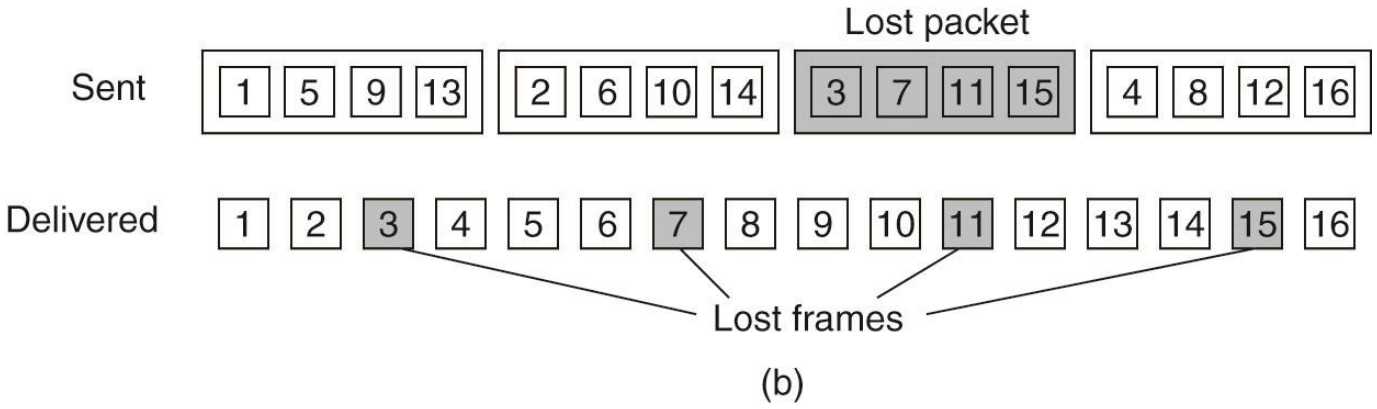
Buffering


Packet departs source
1 2 3 4 5 6 7 8
Packet arrives at buffer
1 2 3 4 5 6 7 8
Packet removed from buffer
Time in buffer
1 2 3 4 5 6 7 8
Gap in playback
0 5 10 15 20
Time (sec)

Interleaving



(a)

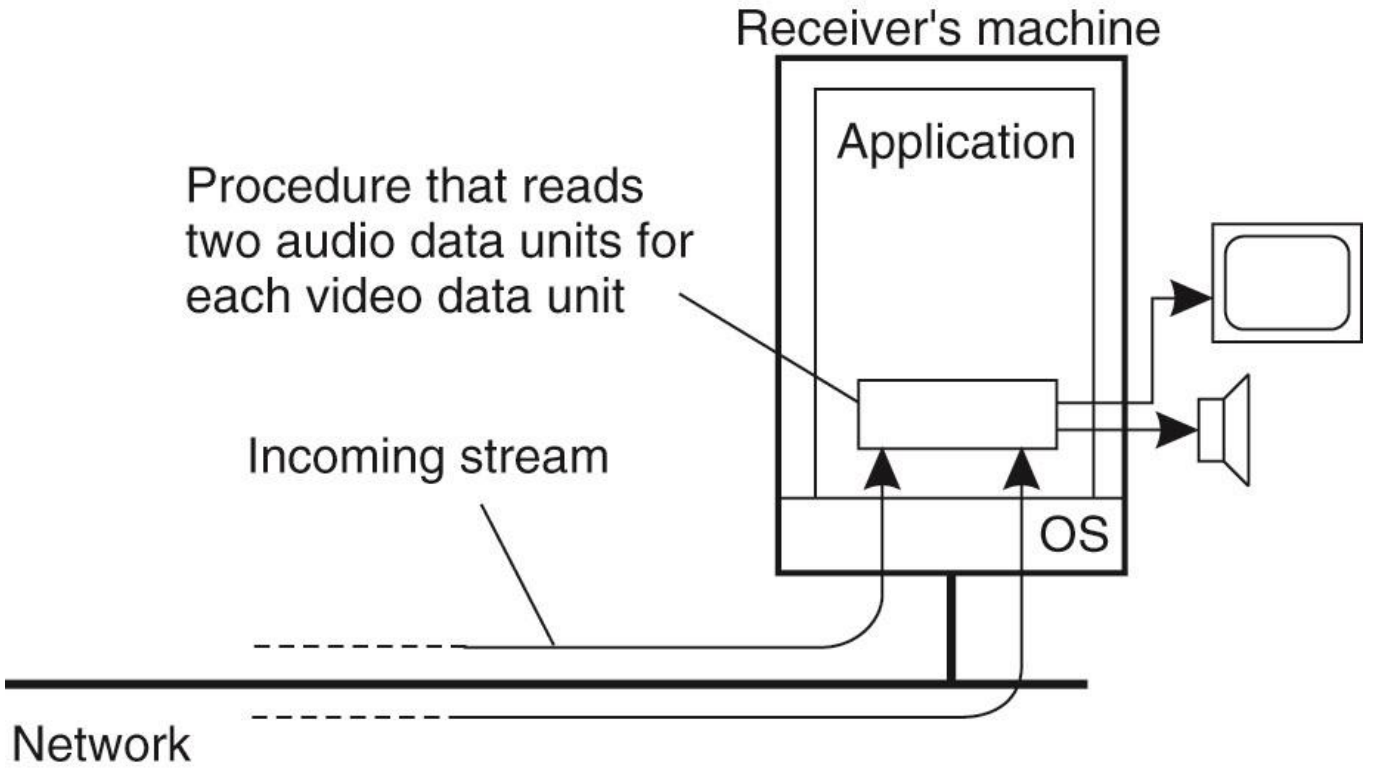


(b)

Stream Synchronization Mechanisms

Given a complex stream, how do you keep the different sub streams in synch?

- Example: playing out two channels, that together form stereo sound



Multicast Communication

Application level multicasting

Gossip based data dissemination

Application level multicasting

Essence

- Organize nodes of a distributed system into an overlay network and use that network to disseminate data

Overlay network organization

- Tree organization: there is only one path between each pair of nodes.
- Mesh network: each node has several neighbors

Chord based Tree Building

Initiator generates a **multicast identifier** mid

Lookup **succ(mid),** the node responsible for mid.

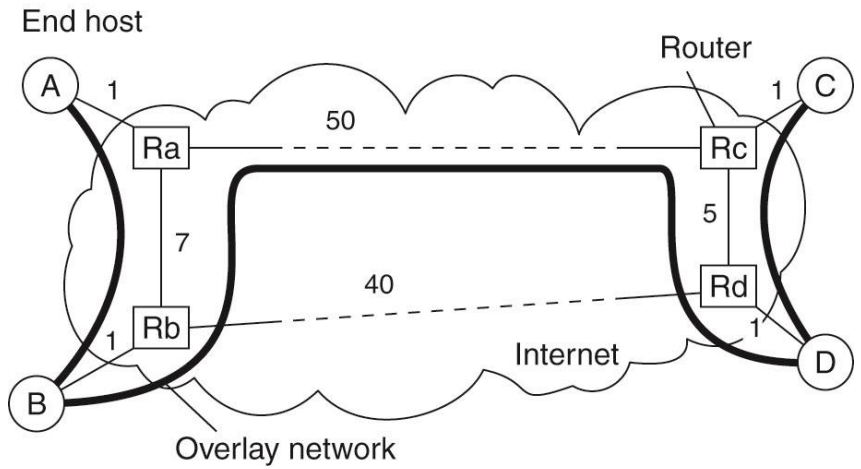
Request is routed to succ(mid), which will become the **root**

If P wants to Join, it sends a **join request** to the root.

When request arrives at Q:

- Q has not seen a join request before: it become forwarded P become chide of Q. **Join request continues to be forwarded.**
- Q knows about tree: P becomes child of Q. **No need to forward join request anymore.**

Overlay Construction



Link stress: how often does an ALM(Application Level Multicast) message cross the same physical link? Example: message from A to D needs to cross<Ra, Rb> twice.

Stretch: ratio in delay between ALM level path and network level path Example: message B to C follow path of length 59 at ALM, but 47 at network level : stretch= 59 /47

Tree cost: Finding the minimal spanning tree

Information Dissemination Models

Epidemic protocols

- **Observation:** how diseases spread among people
- **Goal:** to rapidly propagate information among a large collection of nodes using only local information
  - There is no central component by which information dissemination is coordinated
- **Basic idea:**
  - Update operation are performed at a single server
  - A replica passes updated state to only a few neighbors
  - Update propagation is, lazy i.e not immediate
  - Eventually, each update should reach every replica

Types of Epidemic Protocols

Anti entropy

- Each replica regularly chooses another replica at random, and exchanges state differences, leading to identical state at both afterwards

Gossiping:

- A replica which has just been updated(i.e, has been contaminated), tells a number of other replica about its update(contaminating them as well)

Anti entropy

Principle operations

- A node P selects another node Q from the system at random
- **Push:** P only sends its updates to Q
- **Pull:** p only retrieve updates from Q
- **Push pull:** P and Q exchange mutual updates (after which they hold the same information)

Observation

- For push pull it takes D(log(N)) rounds to disseminate updates to all N nodes(**round**=when every node as taken the initiative to start an exchange)

Gossiping

Basic model

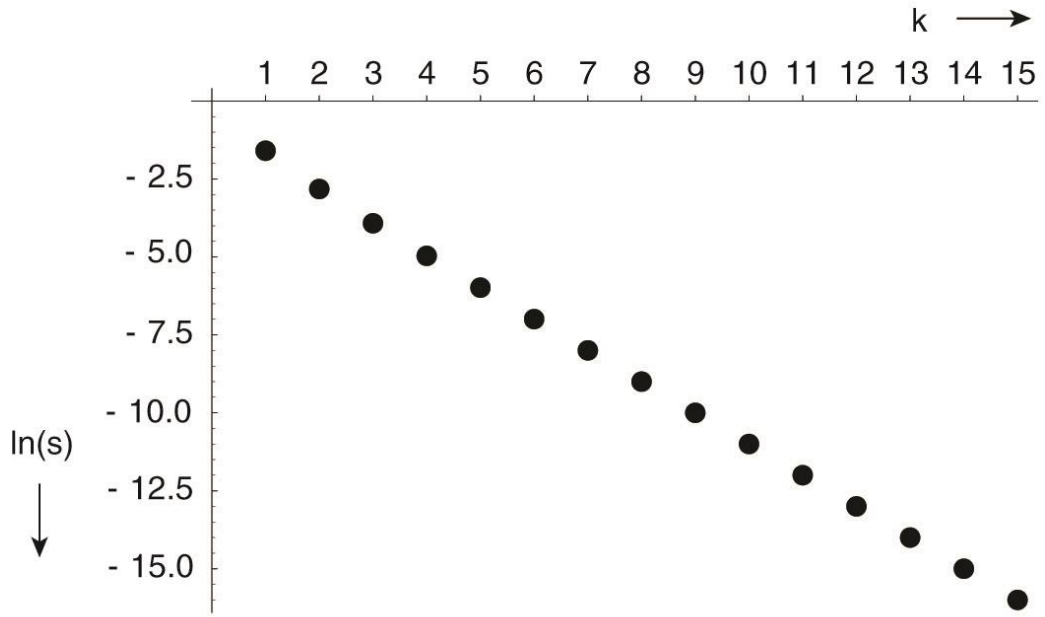
- A server S having an update to report, contacts other servers, if a server is contacted to which the update has already propagated, S stops contacting other servers with probability 1/k.

Observation

- If S is the fraction of ignorant servers(i.e. which are unaware of the update), it can be shown that with many servers

$$S=e^{-(k-1)(1-a)}$$

Gossiping



Consider 10,000 nodes		
R	s	Ns
1	0.203188	2032
2	0. 059520	595
3	0.019837	198
4	0.006977	70
5	0.002516	25
6	0.000918	9
7	0.0000336	3

Note

- If we really have to ensure that all servers are eventually updated, gossiping alone is not enough

Example Applications

Data dissemination: perhaps the most important one.

Aggregation: let every node I maintain a variable x_i when two nodes gossip, they each reset then variable to

$$x_i; x_j \leftarrow (x_i + x_j)/2$$

Result: in the end each node will have computed the average

Question: What happens if initially =1 and =0: j!=1?