Chapter 2

Communication

References

- Andrew S. Tanenbaum and Maarten Van Steen, "Distributed Systems: Principles and Paradigms", 2nd edition, Pearson Education.
- George Coulouris, Jean Dollimore, Tim Kindberg, "Distributed Systems: Concepts and Design", 4th Edition, Pearson Education, 2005.

Content….

2.1 Layered Protocols,
Interprocess communication (IPC): MPI,
Remote Procedure Call (RPC),
Remote Object Invocation,
Remote Method Invocation (RMI)

2.2 Message Oriented Communication,
Stream Oriented Communication,
Group Communication

Introduction

- A communication network provides data exchange between two (or more) end points. Early examples: telegraph or telephone system.
- In a computer network, the end points of the data exchange are computers and/or terminals. (nodes, sites, hosts, etc., …)
- Networks can use switched, broadcast, or multicast technology

Introduction

- In a distributed system, processes run on different machines.
- Processes can only exchange information through message passing.
  - harder to program than shared memory communication
- Successful distributed systems depend on communication models that hide or simplify message passing

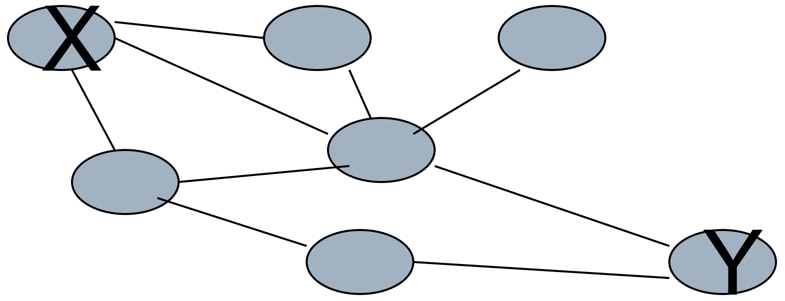
Introduction

- Message-Passing Protocols
  - OSI reference model
  - TCP/IP
  - Others (Ethernet, token ring, …)

- Higher level communication models
  - Remote Procedure Call (RPC)
  - Message-Oriented Middleware (time permitting)
  - Data Streaming (time permitting)

Network Communication Technologies : Switched Networks

- Usual approach in wide-area networks
- Partially (instead of fully) connected
- Messages are switched from one segment to another to reach a destination.
- Routing is the process of choosing the next segment.



Circuit Switching v Packet Switching

- Circuit switching is *connection-oriented* (think traditional telephone system)
  - Establish a dedicated path between hosts
  - Data can flow continuously over the connection
- Packet switching divides messages into fixed size units (packets) which are routed through the network individually.
  - different packets in the same message may follow different routes.

Pros and Cons

- Advantages of packet switching:
  - Requires little or no state information
  - Failures in the network aren't as troublesome
  - Multiple messages share a single link
- Advantages of circuit switching:
  - Fast, once the circuit is established
- Packet switching is the method of choice since it makes better use of bandwidth.

A Compromise

Virtual circuits: based on packet-switched networks, but allow users to establish a connection (usually static) between two nodes and then communicate via a stream of bits, much as in true circuit switching

- Slower than actual circuit switching because it operates on a shared medium
- More secure, or more efficient…

Protocols

- A protocol is a set of rules that defines how two entities interact.
  - For example: HTTP, FTP, TCP/IP,
- Layered protocols have a hierarchical organization
- Conceptually, layer n on one host talks directly to layer n on the other host, but in fact the data must pass through all layers on both machines.

Open Systems Interconnection Reference Model

- Identifies/describes the issues involved in low-level message exchanges
- Divides issues into 7 levels, or layers, from most concrete to most abstract
- Each layer provides an interface (set of operations) to the layer immediately above
- Supports communication between open systems
- Defines functionality – not specific protocols

Layered Protocols

High level 7□

Create message, 6 □ string of bits

Establish Comm. 5□

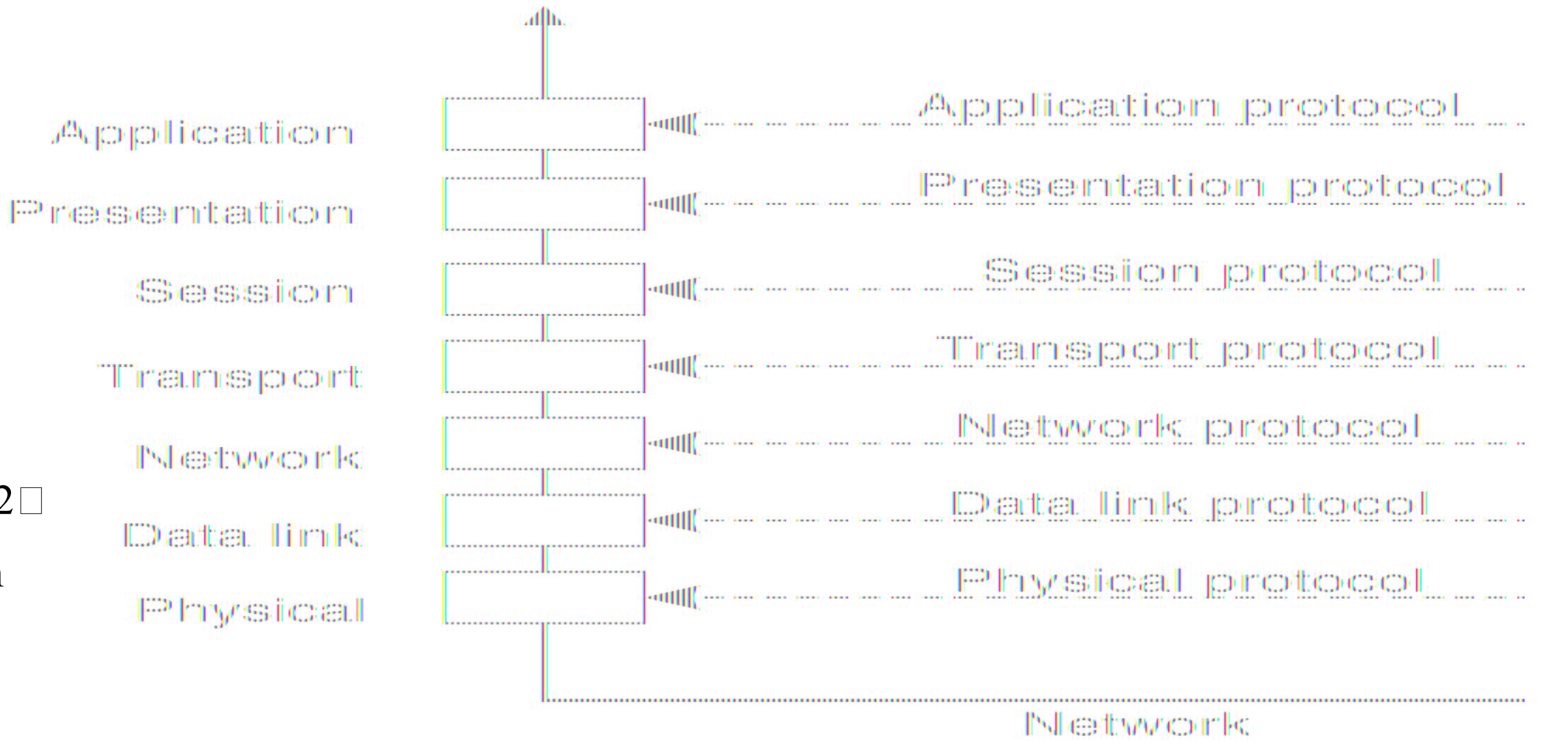
Create packets 4□

Network routing 3□

Add header/footer tag + checksum 2□

Transmit bits via 1□ comm. medium

(e.g. Copper, Fiber, wireless)



Drawbacks

- Focus on message-passing only
- Often unneeded or unwanted functionality
- Violates access transparency

Low-level layers

Recap

- Physical layer: contains the specification and implementation of bits, and their transmission between sender and receiver
- Data link layer: prescribes the transmission of a series of bits into a frame to allow for error and flow control
- Network layer: describes how packets in a network of computers are to be routed.

Observation

- For many distributed systems, the lowest-level interface is that of the network layer.

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
UDP: unreliable (best-effort) datagram communication

Layered Protocols

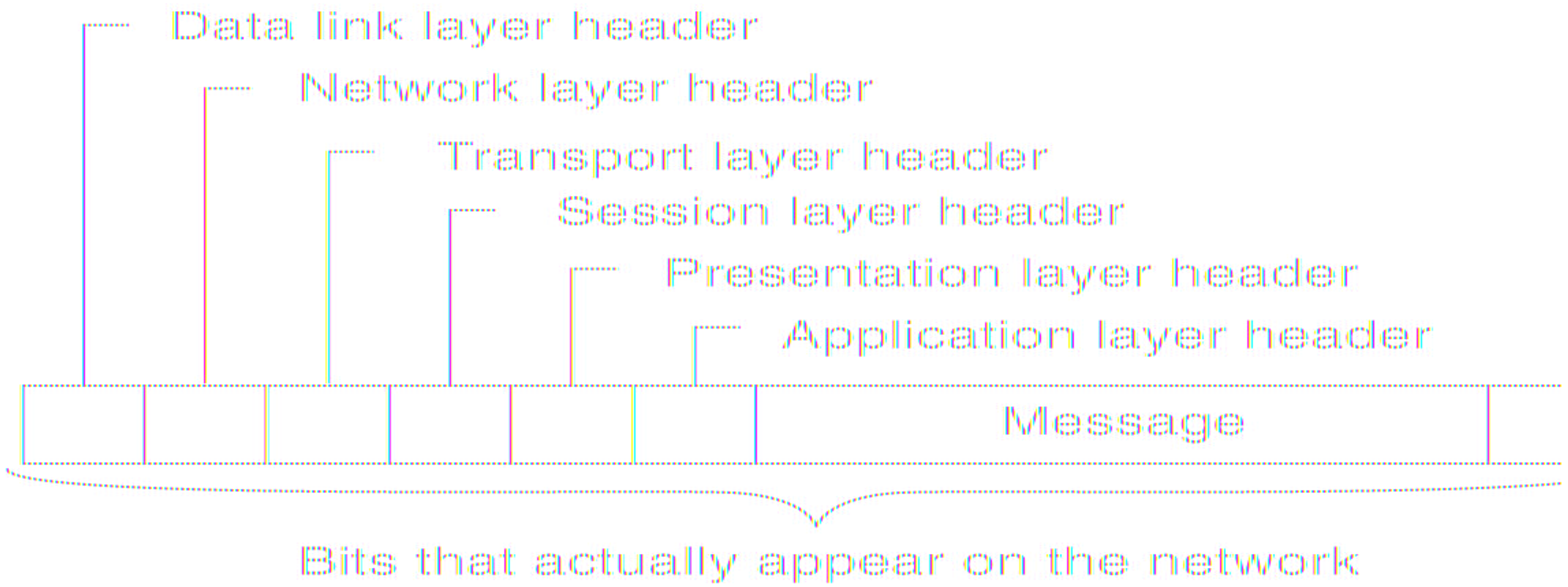


Figure : A typical message as it appears on the network.

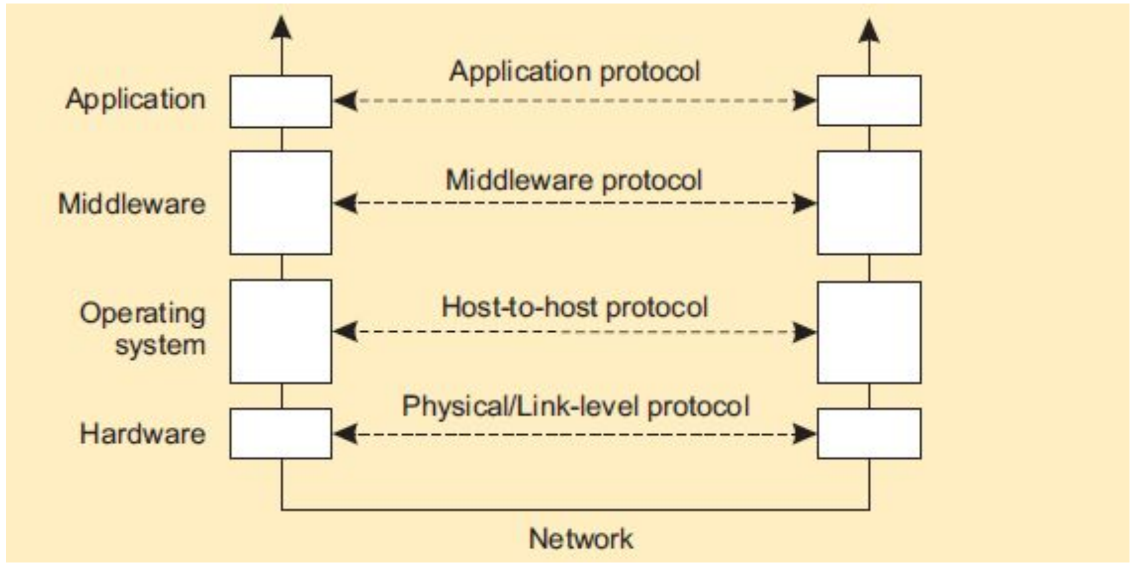
Middleware layer

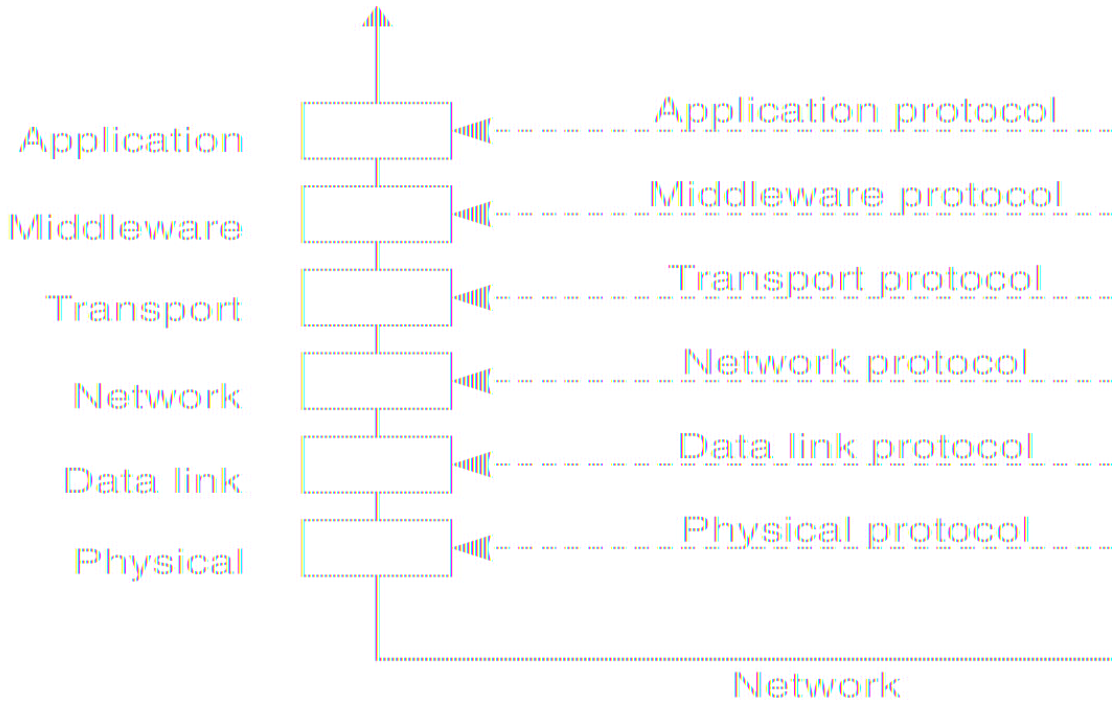
Observation :

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

- A rich set of communication protocols
- (Un)marshaling of data, necessary for integrated systems
- Naming protocols, to allow easy sharing of resources
- Security protocols for secure communication
- Scaling mechanisms, such as for replication and caching

An adapted layering scheme

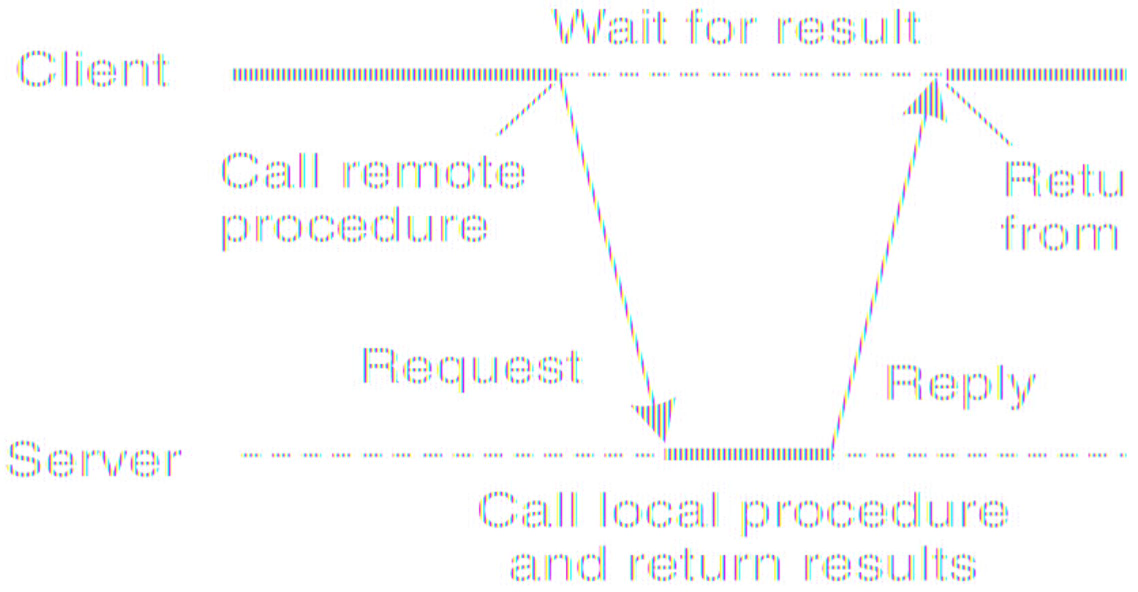




Basic RPC operation

Observations

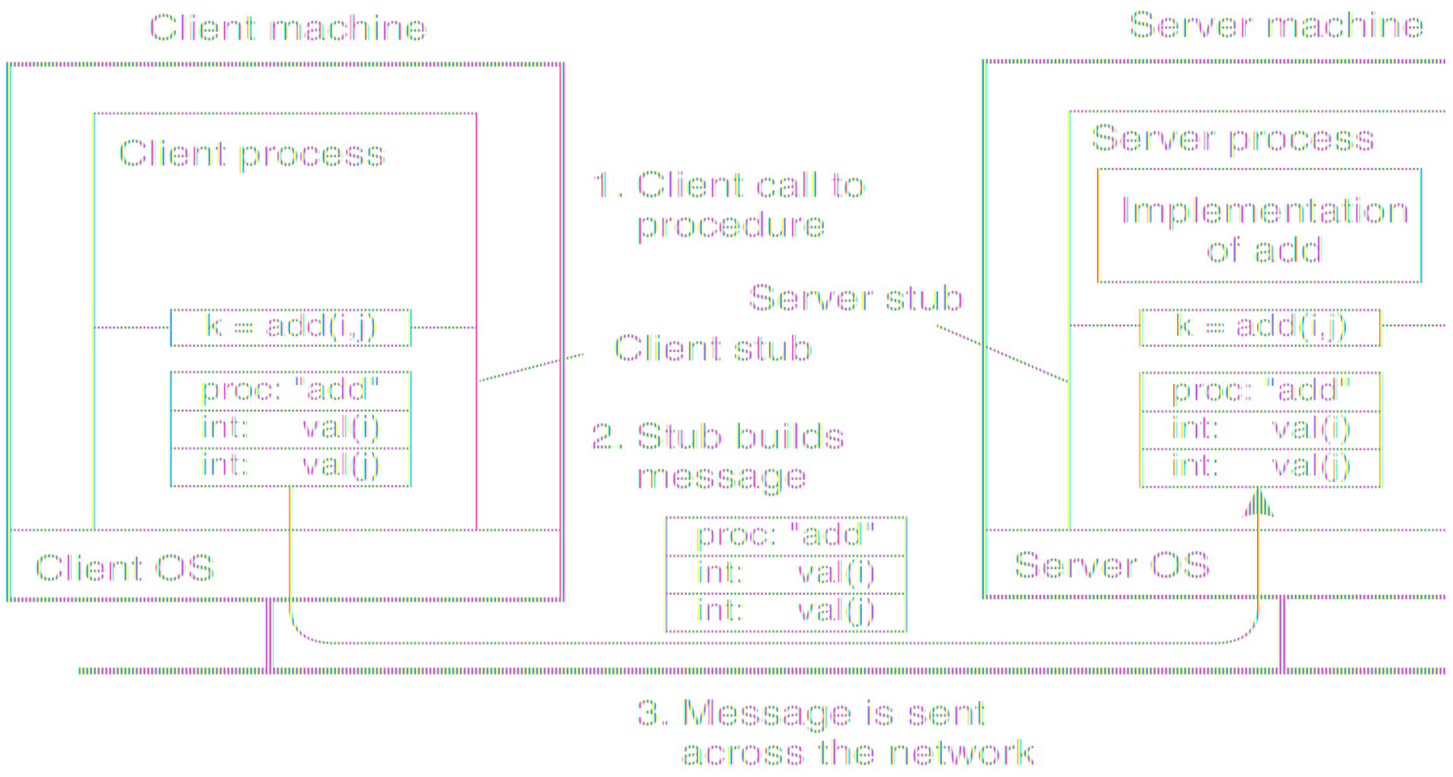
- Application developers are familiar with simple procedure model Well-engineered procedures operate in isolation (black box).
- In short we want RPC in transparent.



Conclusion

•Communication between caller & callee can be hidden by using procedure-call mechanism.

Basic RPC operation



1. Client procedure calls client stub.
2. Stub builds message; calls local OS.
3. OS sends message to remote OS.
4. Remote OS gives message to stub.
5. Stub unpacks parameters; calls server.
6. Server does local call; returns result to stub.
7. Stub builds message; calls OS.
8. OS sends message to client's OS.
9. Client's OS gives message to stub.
10. Client stub unpacks result; returns to client.

Parameter passing : Conventional Procedure Call

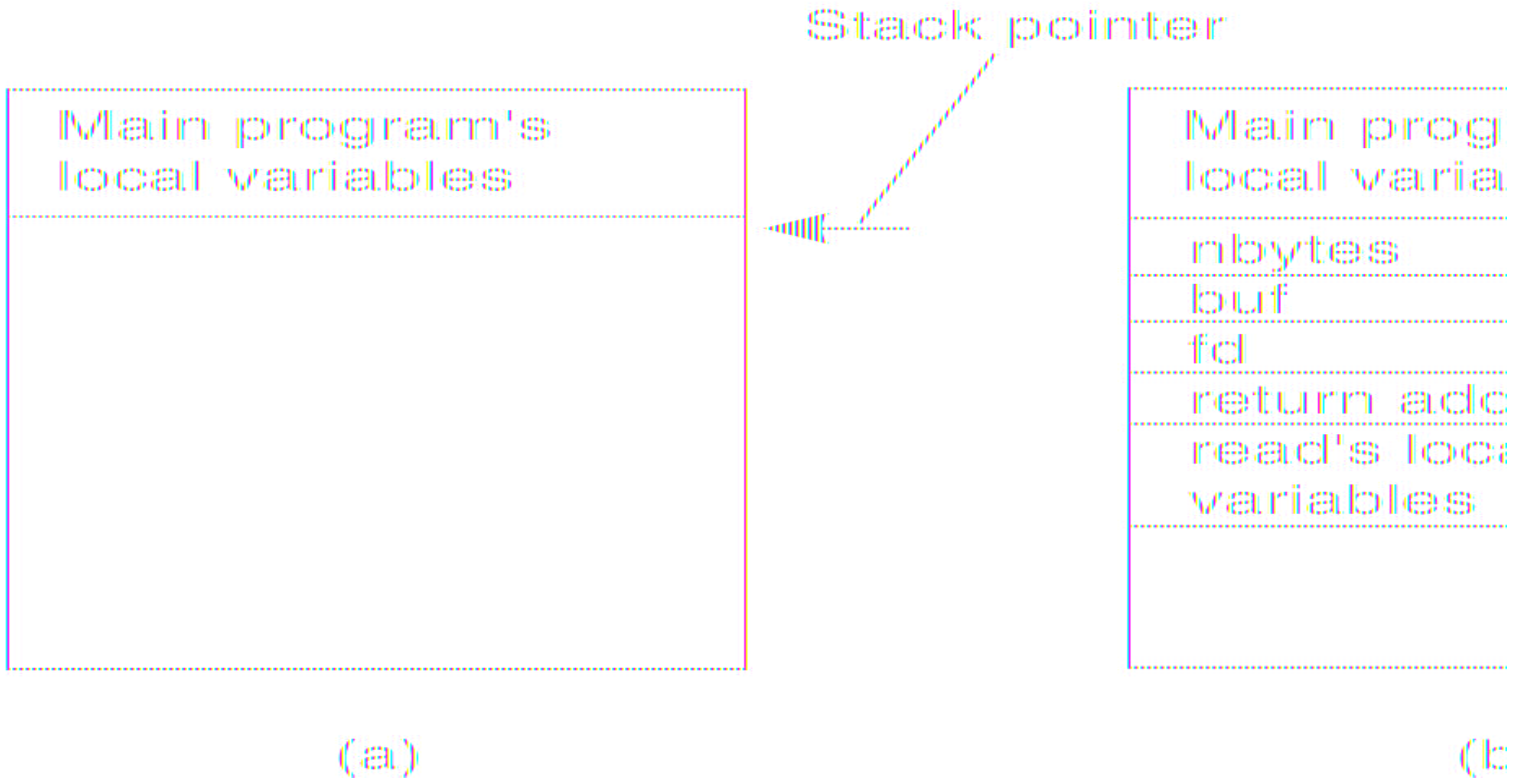


Figure (a) Parameter passing in a local procedure call: the stack before the call to read. (b) The stack while the called procedure is active.

RPC: Parameter passing

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 server have to agree on the same encoding:
    - How are basic data values represented (integers, floats, characters)
    - How are complex data values represented (arrays, unions)
- Solution : Use Standard representation e.g external data representation (XDR)

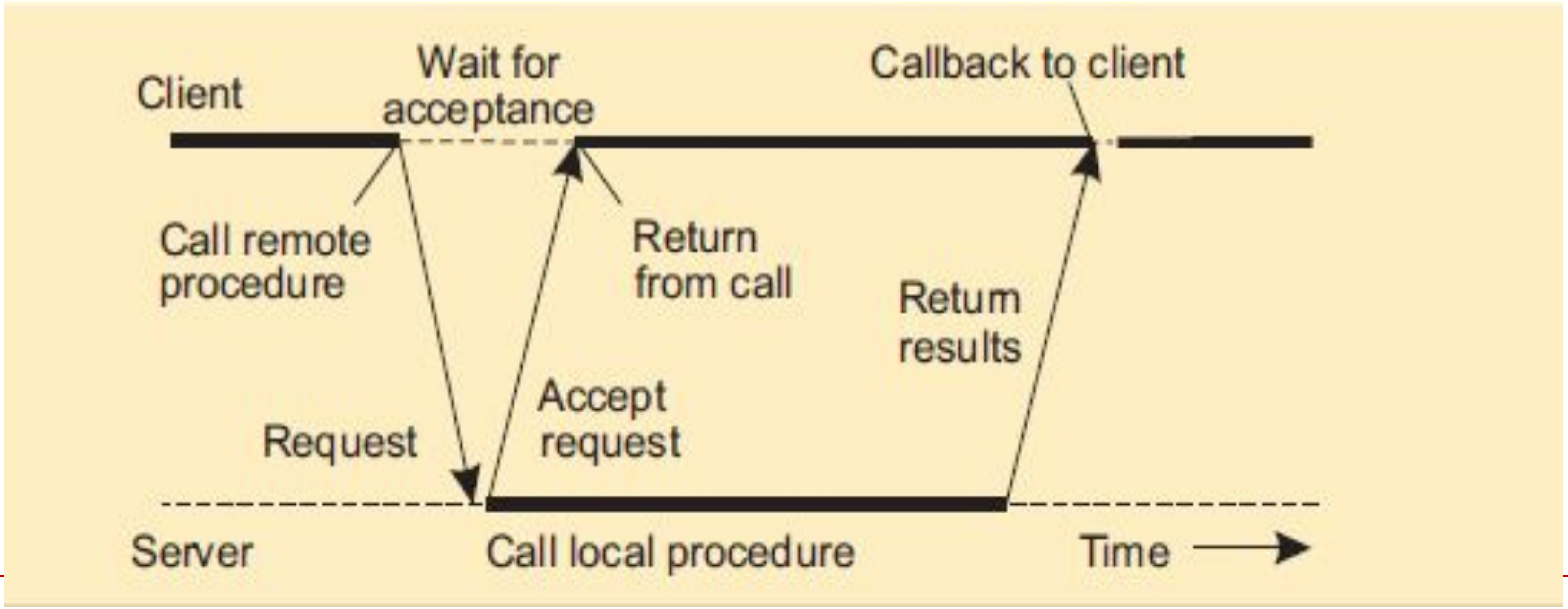
Conclusion

Client and server need to properly interpret messages, transforming them into machine-dependent representations.

RPC: Parameter passing

Essence

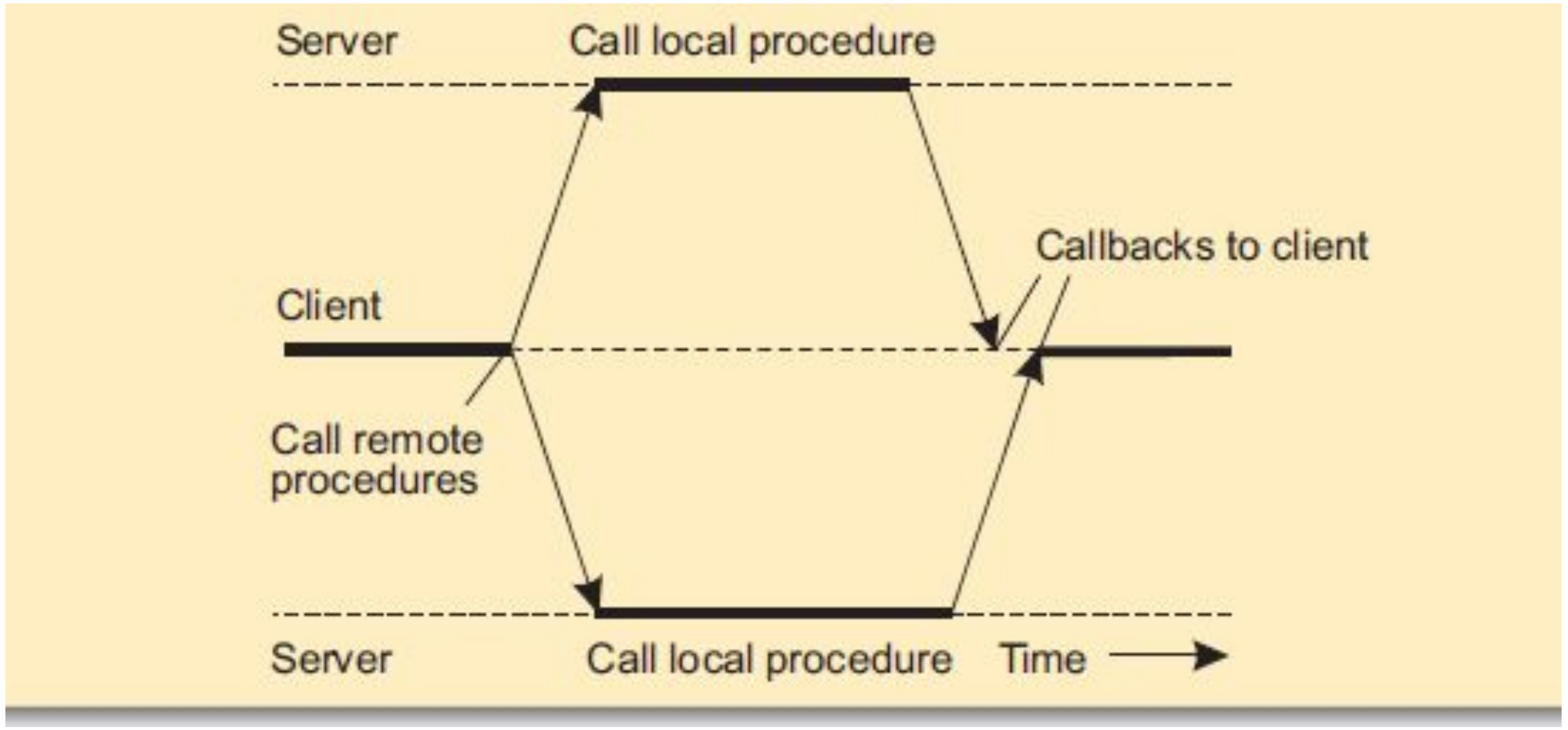
Try to get clear of the strict request-reply behavior, but let the client continue without waiting for an answer from the server **(Asynchronous RPC)** e.g. Money transfer online



RPC: Parameter passing

Essence

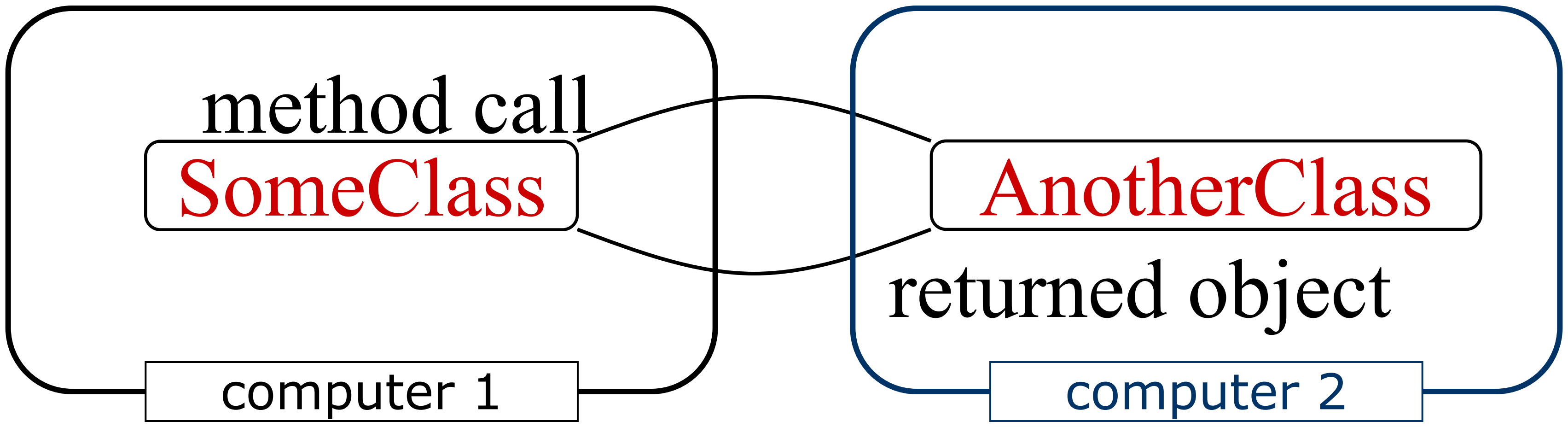
Sending an RPC request to a group of servers.



Remote Method Invocation (RMI)

Remote Method Invocation (RMI)

Consider the following program organization:



- If the network *is* the computer, we have to be able to put the two classes on different computers

•RMI is one technology that makes this possible

Remote Method Invocation (RMI)

- RPCs support procedural programming whereby only remote procedures or functions may be called.
- RMI is object based: It supports invocation of methods on remote objects.

- The parameters to remote procedures are ordinary data structures in RPC; with RMI it is possible to pass objects as parameters to remote methods.

- If the marshaled parameters are local (non remote) objects, they are passed by copy using a technique known as object serialization. Object serialization allowed the state of an object to be written to a byte stream.

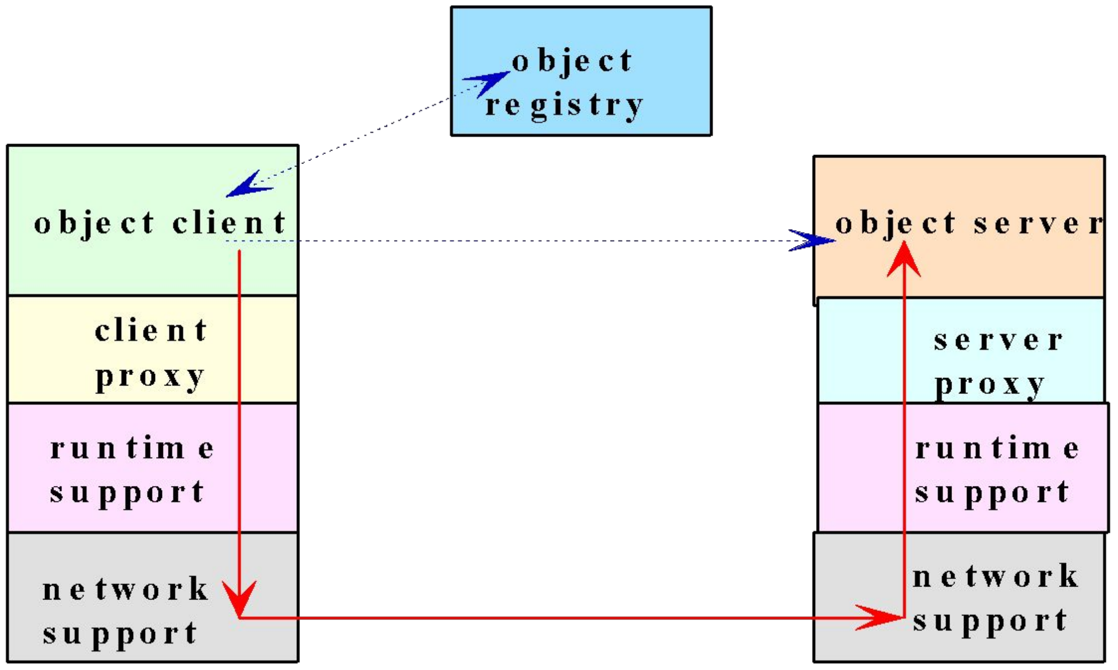
Terminology

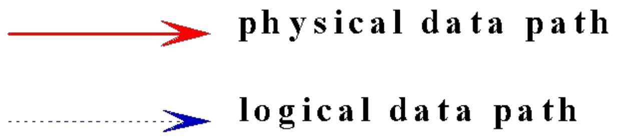
- A remote object is an object on another computer
- The client object is the object making the request (sending a message to the other object)
- The server object is the object receiving the request
- As usual, "client" and "server" can easily trade roles (each can make requests of the other)
- The rmiregistry is a special server that looks up objects by name
  - Hopefully, the name is unique!
- rmic is a special compiler for creating stub (client) and skeleton (server) classes

What is needed for RMI

- Java makes RMI (Remote Method Invocation) *fairly* easy, but there are some extra steps
- To send a message to a remote "server object,"
  - The "client object" has to *find* the object
    - Do this by looking it up in a registry
  - The client object then has to marshal the parameters (prepare them for transmission)
    - Java requires Serializable parameters
    - The server object has to unmarshal its parameters, do its computation, and marshal its response
  - The client object has to unmarshal the response
- Much of this is done for you by special software

An model of Distributed Objects System





- For RMI, you need to be running *three* processes
  - The Client
  - The Server
  - The Object Registry, rmiregistry, which is like a DNS service for objects
- You also need TCP/IP active

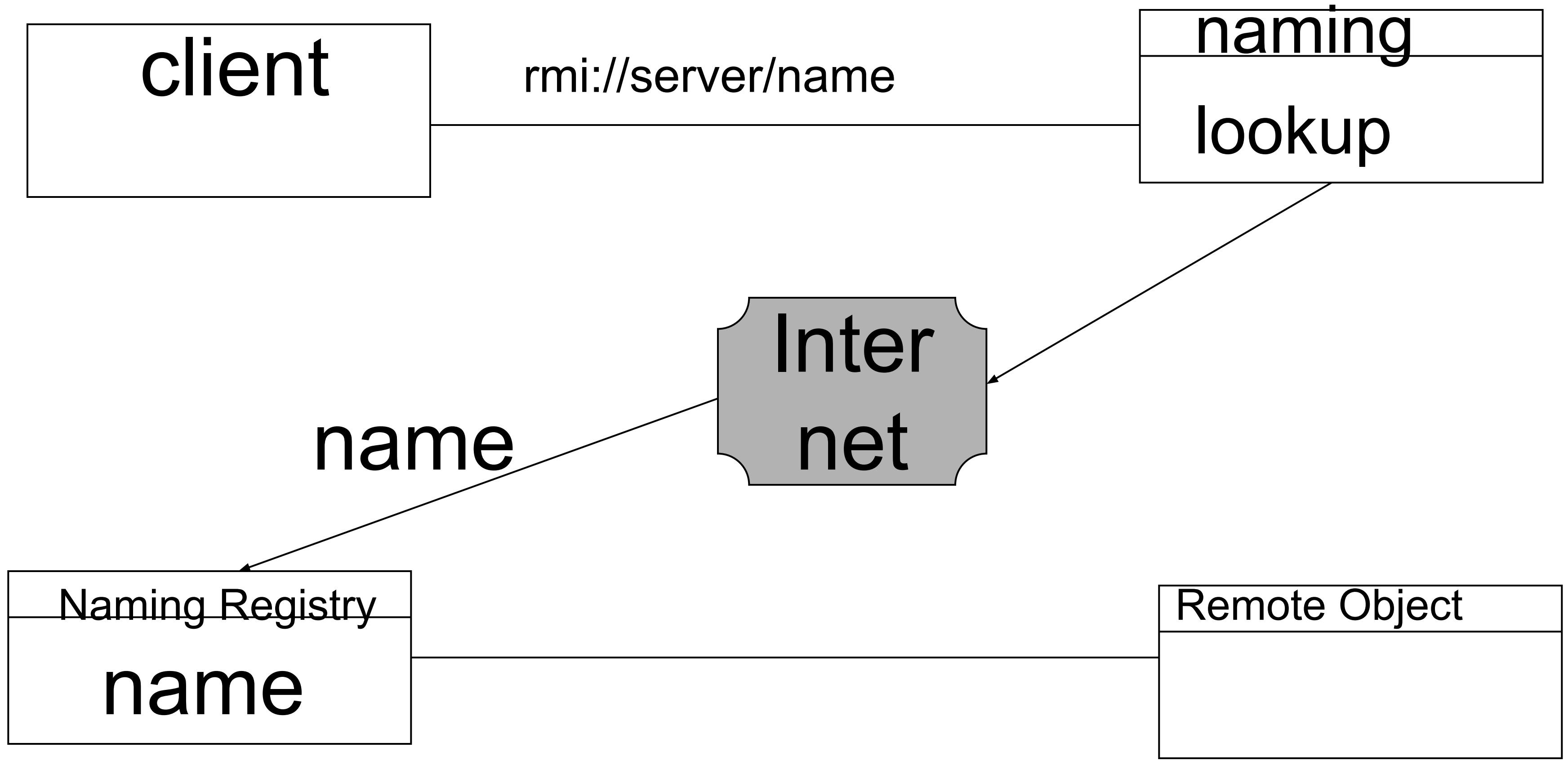
Proxy (Client)

- A proxy is an object at the client, which acts as the implementation of a remote interface
- It communicates with the real object over the network.
- RMI proxies are also named stubs.
- The class definition of a remote stub is automatically generated from the corresponding remote server class by the rmic compiler.

Naming – RMI registry

- Registry is a remote object that maps names to remote objects.
- Given a name by client, registry returns stub of the remote server.
- Remote object must register with naming service to allow clients to locate where is running.
- Client connects to a naming registry and asks for a reference to a service registered under a name.

Naming – RMI registry



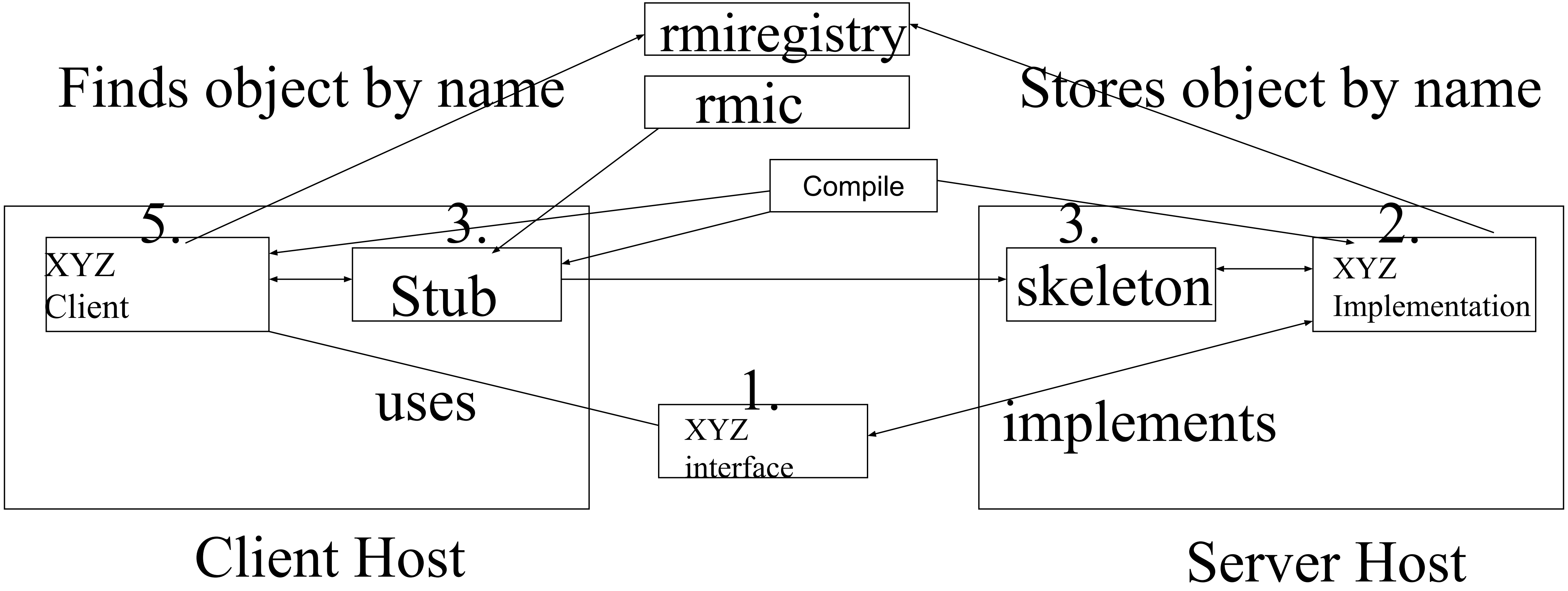
RMI Server

- An RMI server is a remote object which
  - Implements a remote interface
  - Is **`exported`** to the RMI system.
- RMI provides several base class which can be used to define server classes:
  - **RemoteObject**
    - Provides basic remote object semantics for servers and stubs.
  - **RemoteServer**
    - Provides getClientHost, getLog methods used by servers.
  - **UnicastRemoteObject**
    - Supports simple transient point-to-point RMI servers.

Steps in RMI-based Application

1. Design the interface for the service.
2. Implement the methods specified in the interface.
3. Generate the stub and the skeleton.
4. Register the service by name and location.
5. Use the service in an application.

Compile and Register Commands



Compile and Register Commands

- Once the object (or service) is registered, a client can look up that service.
- A client (application) receives a reference that allows the client to use the service (call the method).
- Syntax of calling is identical to a call to a method of another object in the same program.
- Transfer of parameters (or marshalling) is done by the RMI.
- Complex objects are streamed using Serialization.
- RMI model of networking for distributed system involves only Java.
- No need to learn IDL or any other language.

Interfaces

- Interfaces define behavior
- Classes define implementation
  Therefore,
  - In order to use a remote object, the client must know its behavior (interface), but does not need to know its implementation (class)
  - In order to provide an object, the server must know both its interface (behavior) and its class (implementation)
- In short,
  - The interface must be available to both client and server
  - The class of any transmitted object must be on both client and server
  - The class whose method is being used should only be on the server

Classes

- A Remote class is one whose instances can be accessed remotely
  - On the computer where it is defined, instances of this class can be accessed just like any other object
  - On other computers, the remote object can be accessed via object handles
- A Serializable class is one whose instances can be marshaled (turned into a linear sequence of bits)
  - Serializable objects can be transmitted from one computer to another
- It probably isn't a good idea for an object to be both remote and serializable

Conditions for serializability

If an object is to be serialized:

- The class must be declared as public
- The class must implement Serializable
  - However, Serializable does not declare any methods
- The class must have a no-argument constructor
- All fields of the class must be serializable: either primitive types or Serializable objects
  - Exception: Fields marked transient will be ignored during serialization

Remote interfaces and class

A Remote class has two parts:

- **The interface (used by both client and server):**
  - Must be public
  - Must extend the interface java.rmi.Remote
  - Every method in the interface must declare that it throws java.rmi.RemoteException (other exceptions may also be thrown)
- **The class itself (used only by the server):**
  - Must implement the Remote interface
  - Should extend java.rmi.server.UnicastRemoteObject
  - May have locally accessible methods that are not in its Remote interface

Remote vs. Serializable

A Remote object lives on another computer (such as the Server)

- You can send messages to a Remote object and get responses back from the object
- All you need to know about the Remote object is its interface
- Remote objects don’t pose much of a security issue

You can transmit a *copy* of a Serializable object between computers

- The receiving object needs to know how the object is implemented; it needs the class as well as the interface
- There is a way to transmit the class definition
- Accepting classes *does* pose a security issue

The server class

- The class that defines the server object should extend UnicastRemoteObject
  - This makes a connection with exactly one other computer
  - If you must extend some other class, you can use exportObject() instead
  - Sun does *not* provide a MulticastRemoteObject class

- The server class needs to register its server object:
  - String url = "rmi://" + ***host*** + ":" + ***port*** + "/" + ***objectName***;
    - The default port is 1099
  - Naming.rebind(url, ***object***);

- Every remotely available method must throw a RemoteException (because connections can fail)
- Every remotely available method should be synchronized

Hello world server: interface

```
import java.rmi.*;

    public interface HelloInterface extends Remote {
      public String say() throws RemoteException;
    }
```

Hello world server: class

```
import java.rmi.*;
    import java.rmi.server.*;

    public class Hello extends UnicastRemoteObject
                     implements HelloInterface {
      private String message; // Strings are serializable

      public Hello (String msg) throws RemoteException {
        message = msg;
      }

      public String say() throws RemoteException {
        return message;
      }
    }
```

Registering the hello world server

```
class HelloServer {
        public static void main (String[] argv) {
           try {
              Naming.rebind("rmi://localhost/HelloServer",
                                    new Hello("Hello, world!"));
              System.out.println("Hello Server is ready.");
           }
           catch (Exception e) {
              System.out.println("Hello Server failed: " + e);
           }
        }
     }
```

The hello world client program

```
class HelloClient {
        public static void main (String[] args) {
          HelloInterface hello;
          String name = "rmi://localhost/HelloServer";
          try {
             hello = (HelloInterface)Naming.lookup(name);
             System.out.println(hello.say());
           }
           catch (Exception e) {
             System.out.println("HelloClient exception: " + e);
           }
        }
     }
```

rmic

- The class that implements the remote object should be compiled as usual
- Then, it should be compiled with rmic:
  - rmic Hello
- This will generate files Hello_Stub.class and Hello_Skel.class
- These classes do the actual communication
  - The "Stub" class must be *copied* to the client area
  - The "Skel" was needed in SDK 1.1 but is no longer necessary

Trying RMI

In three different terminal windows:

- Run the registry program:

  ```
  rmiregistry
  ```

- Run the server program:

  ```
  java HelloServer
  ```

- Run the client program:

  ```
  java HelloClient
  ```

If all goes well, you should get the “Hello, World!” message

Summary: RMI

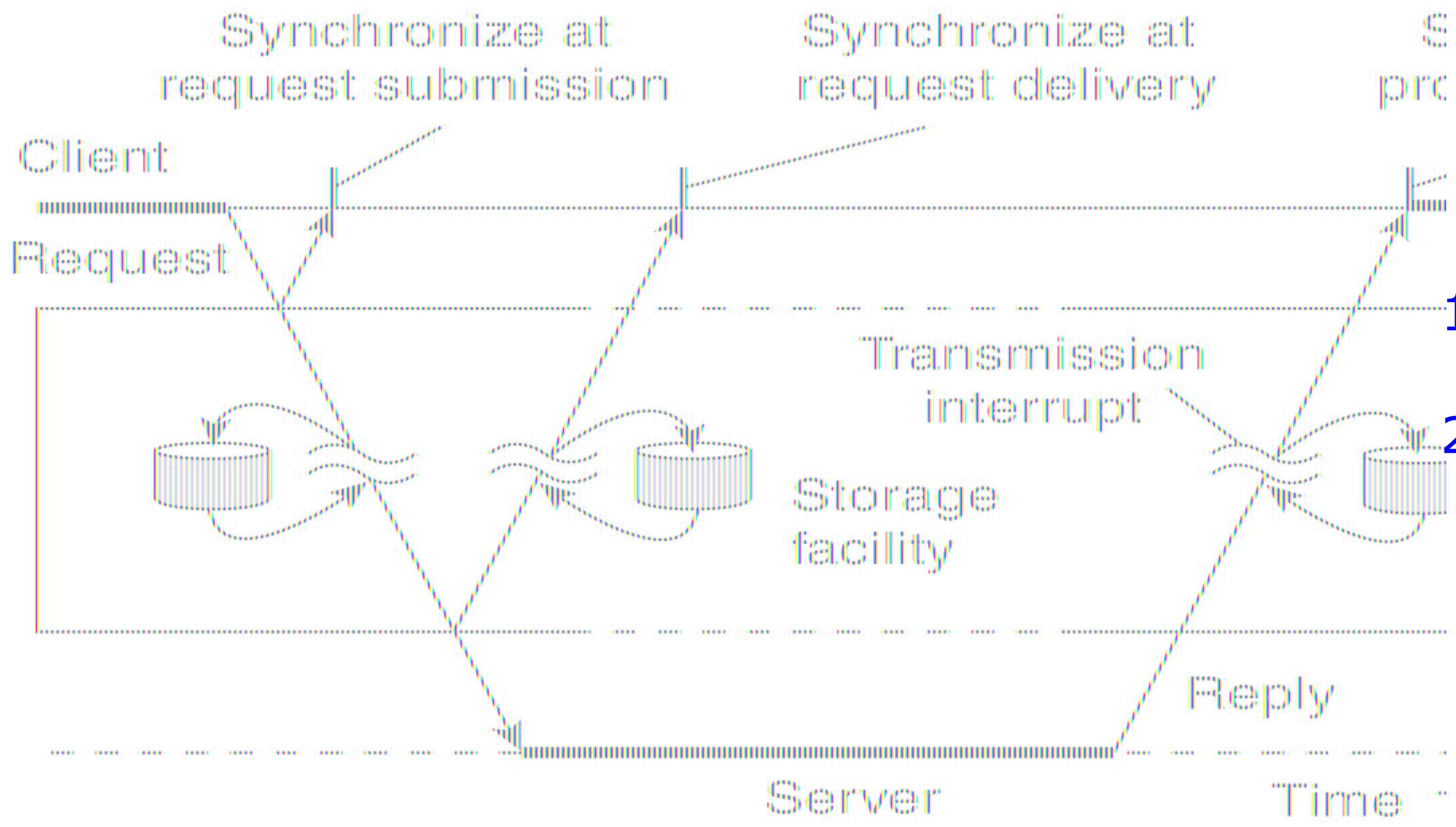
1. Start the registry server, rmiregistry
2. Start the object server

   The object server registers an object, with a name, with the registry server
3. Start the client

   The client looks up the object in the registry server
4. The client makes a request
   - The request actually goes to the Stub class
   - The Stub classes on client and server talk to each other
   - The client's Stub class returns the result

2.2 Message Oriented Communication, Stream Oriented Communication, Group Communication

Types of Communication



Note ----

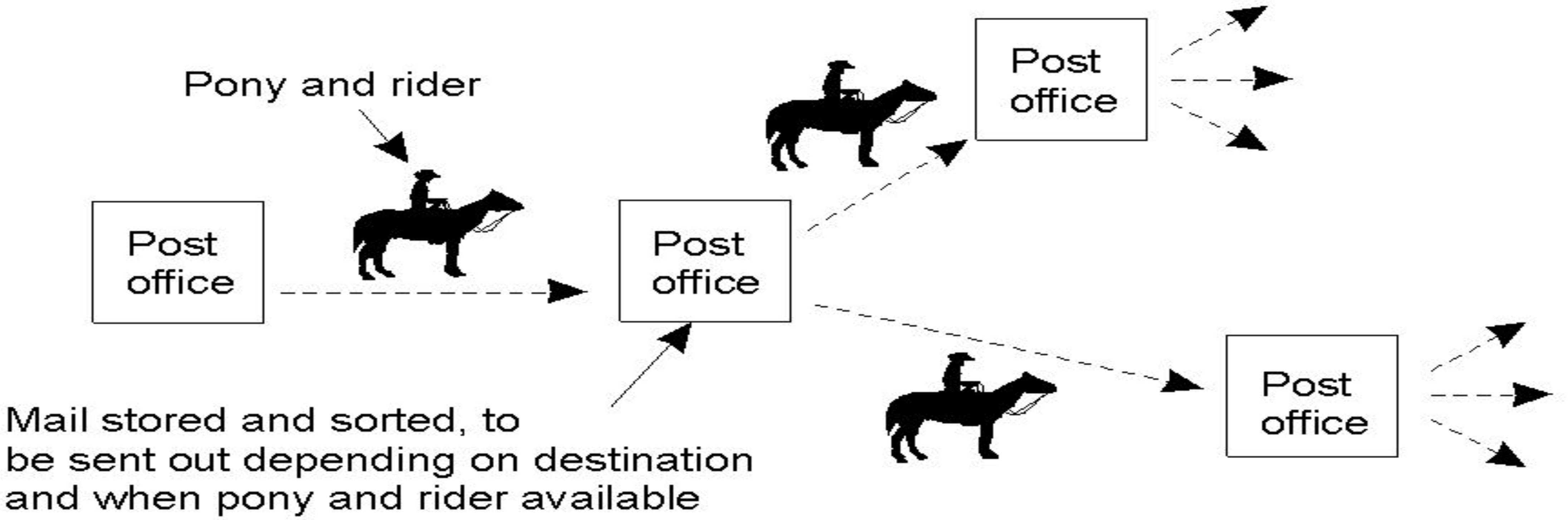
1. Transient versus persistent communication
2. Asynchronous versus synchronous communication

Fig: Viewing middleware as an intermediate (distributed) service in application-level communication.

Persistent vs. Transient communication

- **Persistent communication**
  - a message is stored by the communication system as long as it takes to deliver it never lost or thrown away.
  - e.g e-mail, SMS
- **Transient communication**
  - a message is stored by the communication system only as long as the sending and receiving application are executing
  - Discard message if it can’t be delivered to next server/receiver
  - Example: transport-level communication services offer transient communication
  - Example: Typical network router – discard message if it can’t be delivered next router or destination

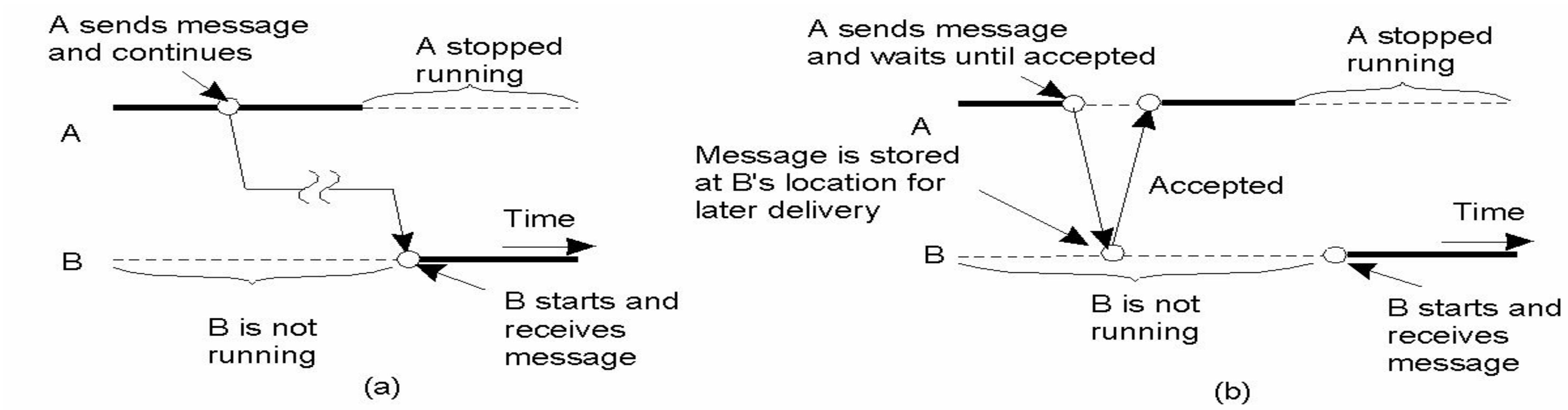
Example - Persistent communication of letters


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

Synchronous v/s Asynchronous communication

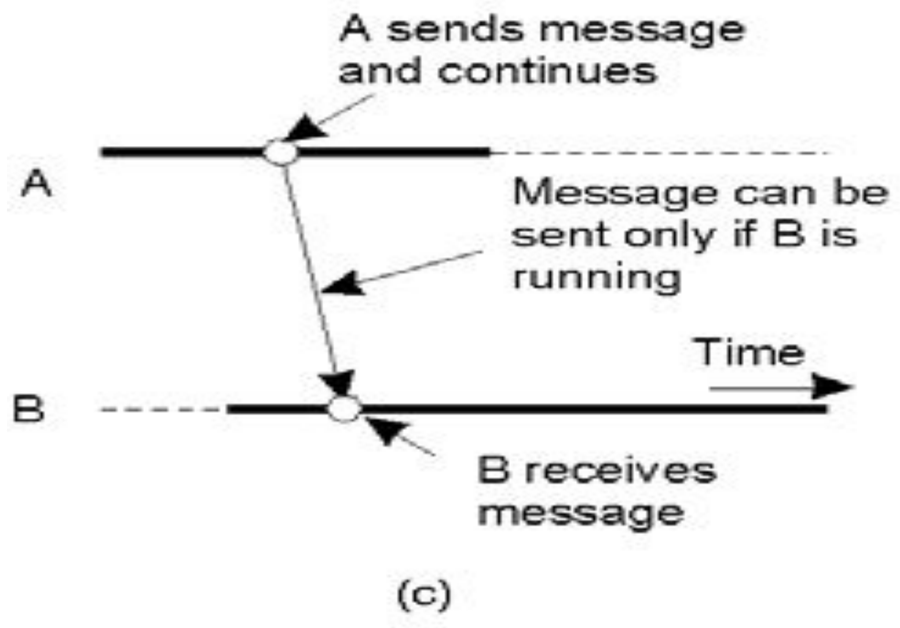
- Synchronous communication blocks the process until the message is received or the sender gets a response from the server.
- Synchronous = happens at the same time.
- Asynchronous communication means that the sender is doing non-blocking sending. It continues immediately after submitting the message. It doesn't require the recipient's immediate attention, allowing them to respond to the message at their convenience
- Asynchronous = doesn't happen at the same time.

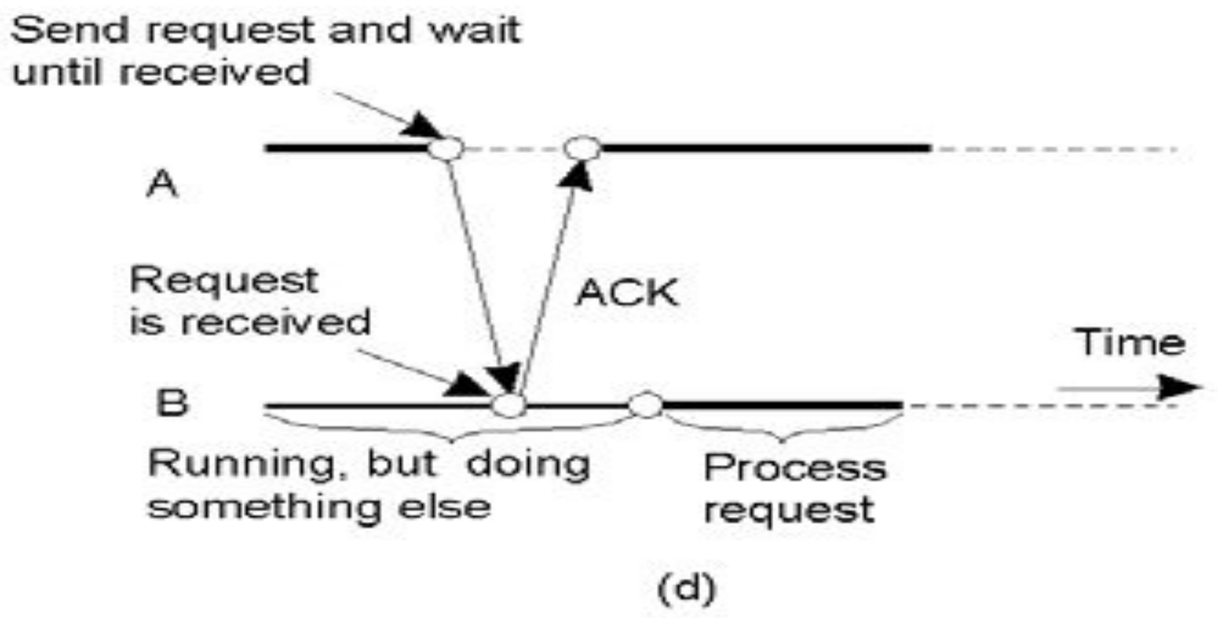
Forms of communication


A sends message and continues
A stopped running
A
Time
B
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 (e.g., email)
b) Persistent synchronous communication (Some instant message applications, such as Blackberry messenger, are good examples. When you send out a message, the app shows you the message is ”delivered” but not ”read”. After the message is read, you will receive another acknowledgement.)

Forms of communication

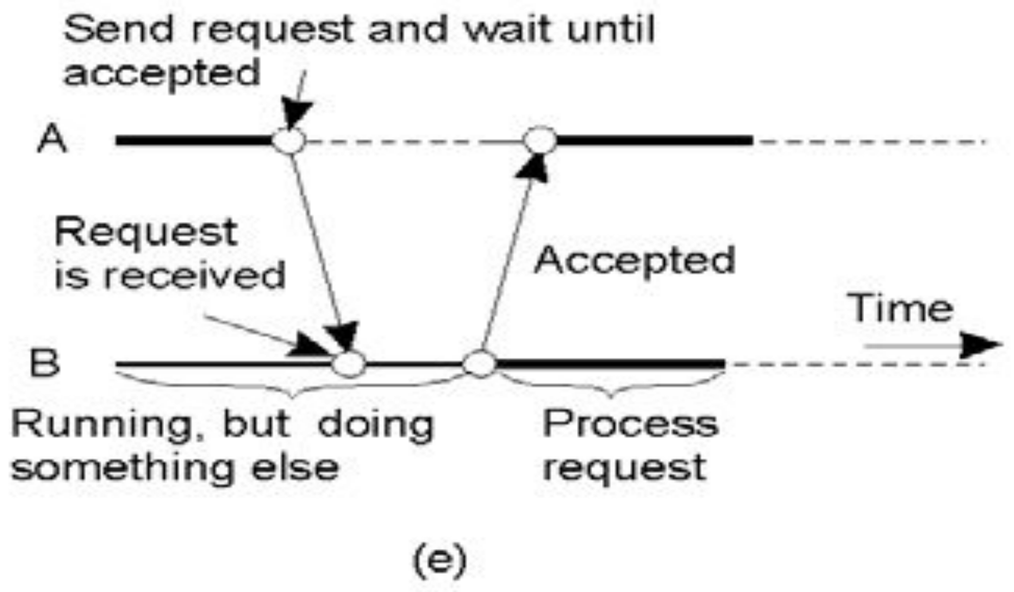

A sends message and continues
A
Message can be sent only if B is running
Time
B
B receives message
(c)

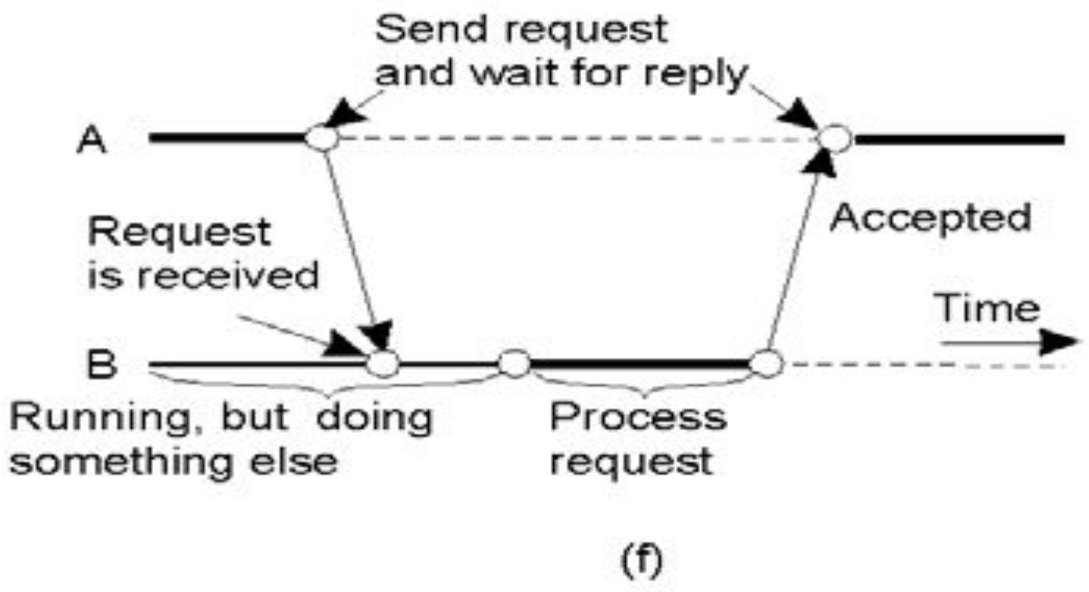

Send request and wait until received
A
Request is received
ACK
Time
B
Running, but doing something else
Process request
(d)

c) Transient asynchronous communication (e.g., UDP)

d) Receipt-based transient synchronous communication (eg RPC, RMI)

Forms of communication


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 (e.g., asynchronous RCP – Banking system)

f) Response-based transient synchronous communication (RPC)

Client/Server computing

Observation :

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

- Client and server have to be active at time of communication
- Client issues request and blocks until it receives reply
- Server essentially waits only for incoming requests, and subsequently processes them

Drawbacks synchronous communication

- Client cannot do any other work while waiting for reply
- Failures have to be handled immediately: the client is waiting
- The model may simply not be appropriate (mail, news)

Message - oriented communication

- **Message-oriented transient communication**
  - Berkeley Sockets (Transport)
  - Message Passing Interface (MPI)
  - Latency : milisec to sec.
- **Message-oriented persistent communication**
  - Message Queuing Model
  - Does not required either sender/receiver to active during message transmission
  - Latency : sec to min.

Message Oriented Communication

- RPC and RMI support access transparency, but aren't always appropriate
- Message-oriented communication is more flexible
- Built on transport layer protocols.
- Standardized interfaces to the transport layer include **sockets** (Berkeley UNIX) and XTI (X/Open Transport Interface), formerly known as TLI (AT&T model)

Sockets

- A communication endpoint used by applications to write and read to/from the network.
- Sockets provide a basic set of primitive operations
- Sockets are an abstraction of the actual communication endpoint used by local OS
- Socket address: IP# + port#

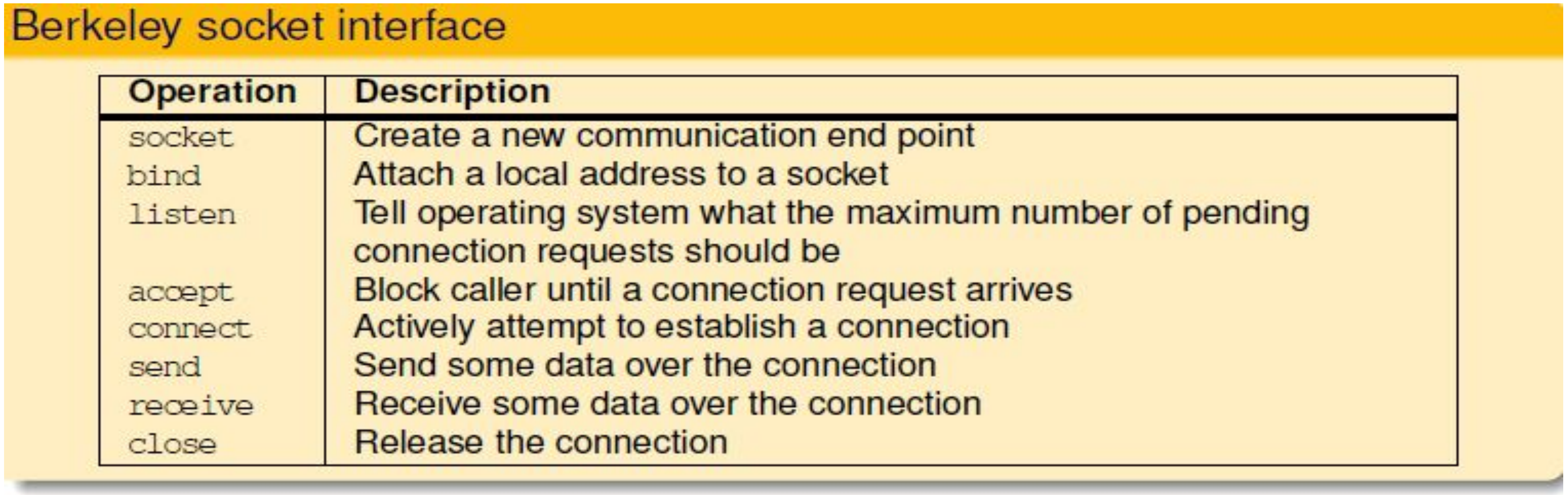
Socket Communication

- Using sockets, clients and servers can set up a connection-oriented communication session.
- Servers execute first four primitives (socket, bind, listen, accept) while clients execute socket and connect primitives)
- Then the processing is client/write, server/read, server/write, client/read, all close connection.

Message-oriented Transient Communication

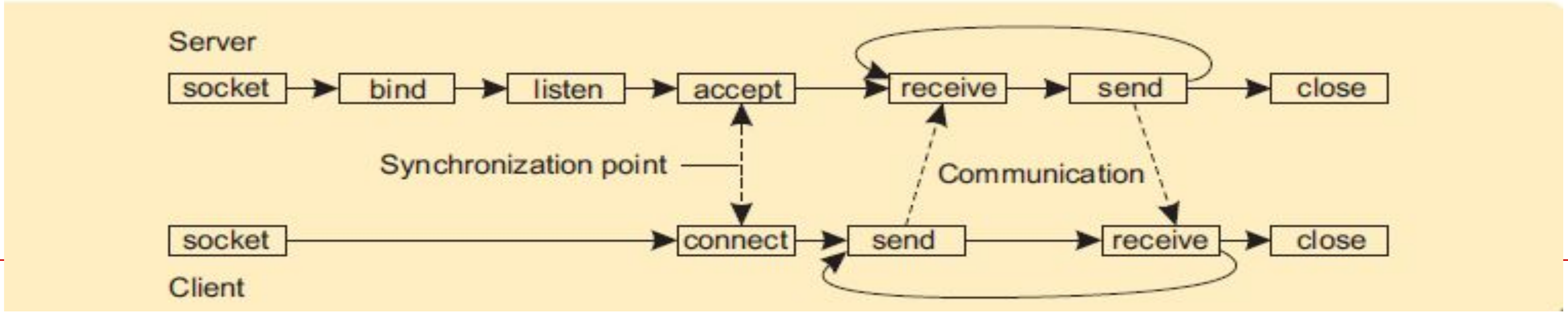
- Many distributed systems and applications are built directly on top of the simple message-oriented model offered by the transport layer.
- Berkeley Socket
- Socket: It is a communication end point.

Transient messaging : Socket



Berkeley socket interface

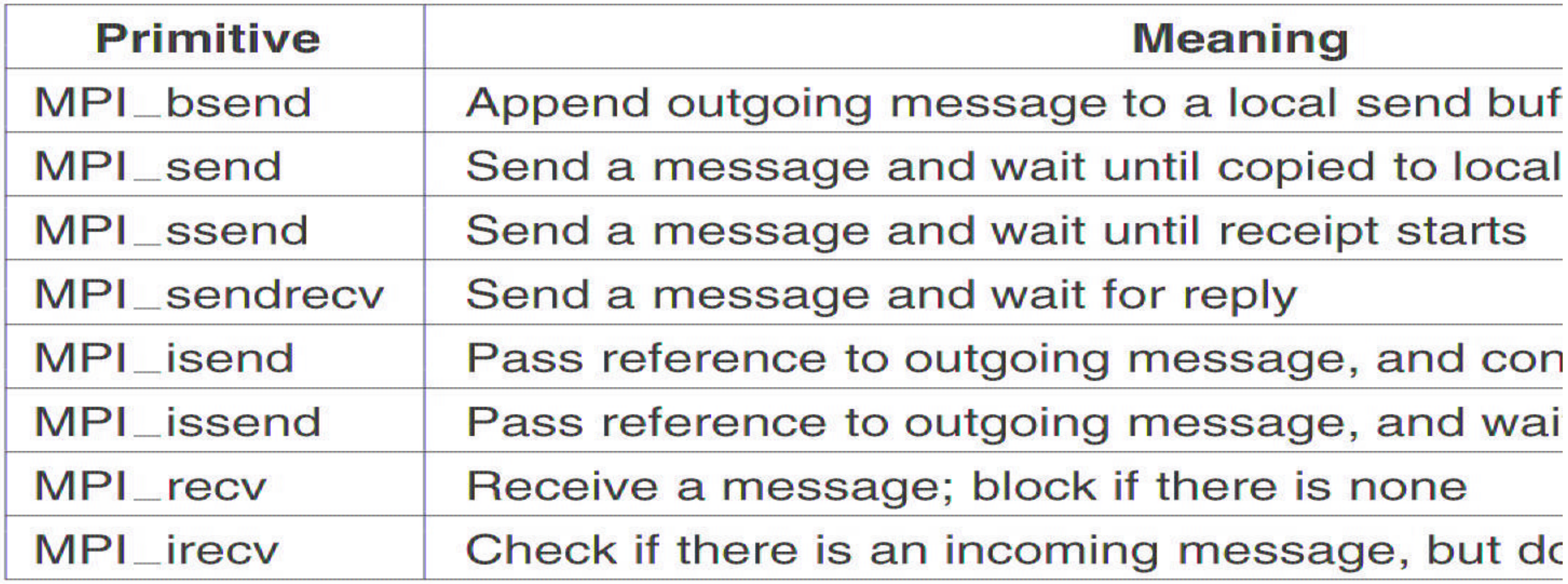
Operation	Description
socket	Create a new communication end point
bind	Attach a local address to a socket
listen	Tell operating system what the maximum number of pending connection requests should be
accept	Block caller until a connection request arrives
connect	Actively attempt to establish a connection
send	Send some data over the connection
receive	Receive some data over the connection
close	Release the connection



Message-Passing Interface (MPI)

- Message Passing Interface (MPI) is a communication protocol for parallel programming. MPI is specifically used to allow applications to run in parallel across a number of separate node/computers connected by a network.
- Disadvantages of Socket:
  - at the wrong level of abstraction by support only simple send and receive primitives.
  - designed to communicate across networks using general-purpose protocol stacks(TCP/IP).
  - not considered suitable for the proprietary protocols developed for **high-speed interconnection networks**, such as those used in high-performance server clusters.
- Those protocols required an interface.
  - For high-performance *multicomputer.*
  - to easily write highly efficient applications.
  - implementation incurs only minimal overhead.
  - hardware and platform independent
  - designed for parallel applications
  - communication takes place within group of processes.
  - higher level interface for transient async & sync communication

MPI: Lot of flexibility needed



Primitive	Meaning
MPI_bsend	Append outgoing message to a local send buf
MPI_send	Send a message and wait until copied to local
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 con
MPI_issend	Pass reference to outgoing message, and wai
MPI_recv	Receive a message; block if there is none
MPI_irecv	Check if there is an incoming message, but d

MPI Apps versus C/S

- Processes in an MPI-based parallel system act more like peers (or peer slaves to a master processor)
- Communication may involve message exchange in multiple directions.
- C/S communication is more structured.

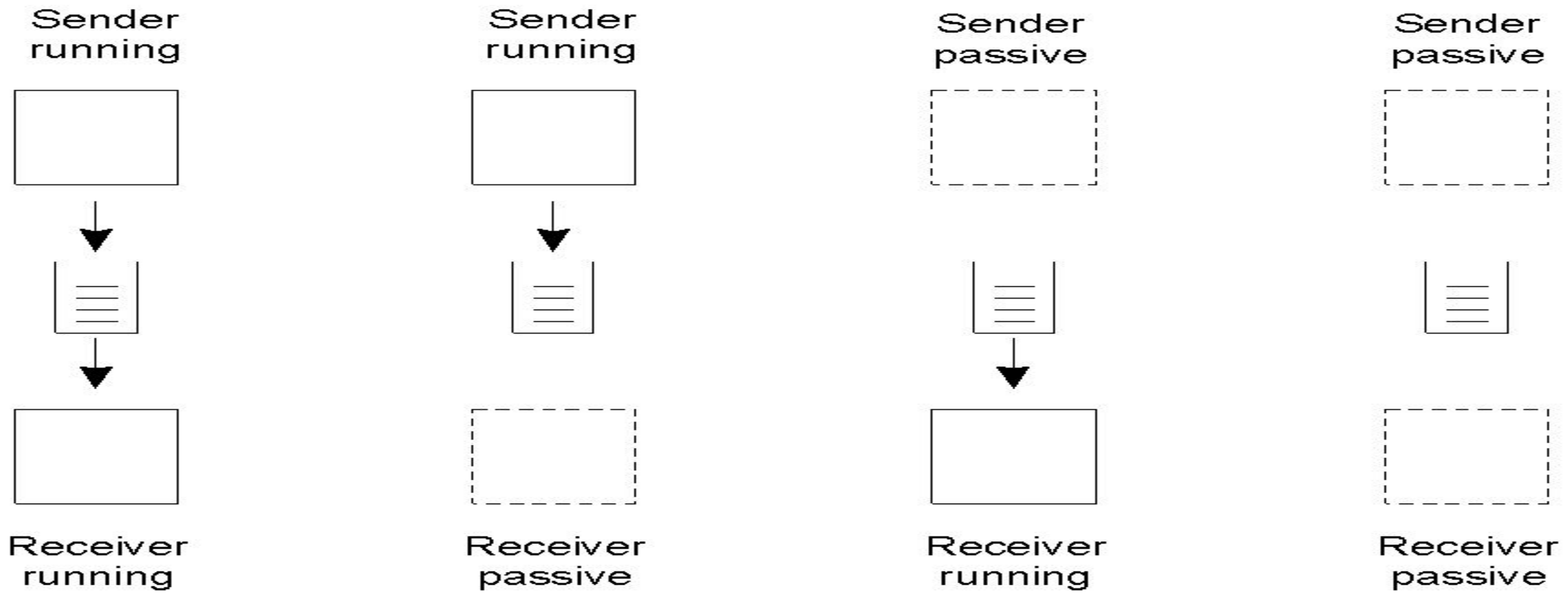
Message - oriented persistent communication

Message Queuing Systems

- Message Oriented Middleware – MOM
- Support asynchronous persistent communication
- basic idea: applications communicate by inserting messages in specific queues.
- messages are forwarded over a series of communication servers
- provides queues to sender and receiver
- senders and receivers do not need to be active at the same time
- guarantee for delivering, no guarantee for the delivering moment
- “Loosely coupled communication”

Message Queuing Model

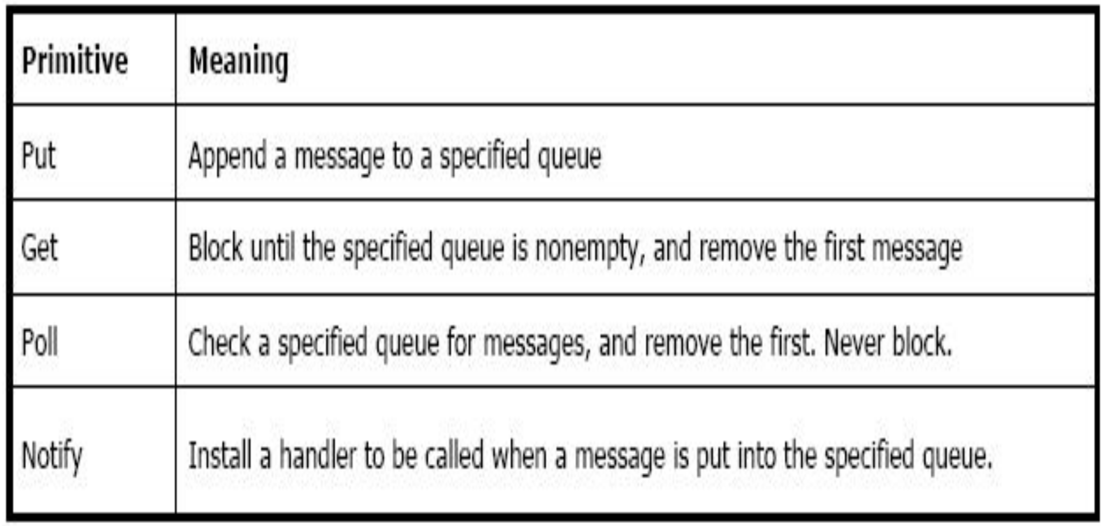
Four combinations for loosely-coupled communications using queues.



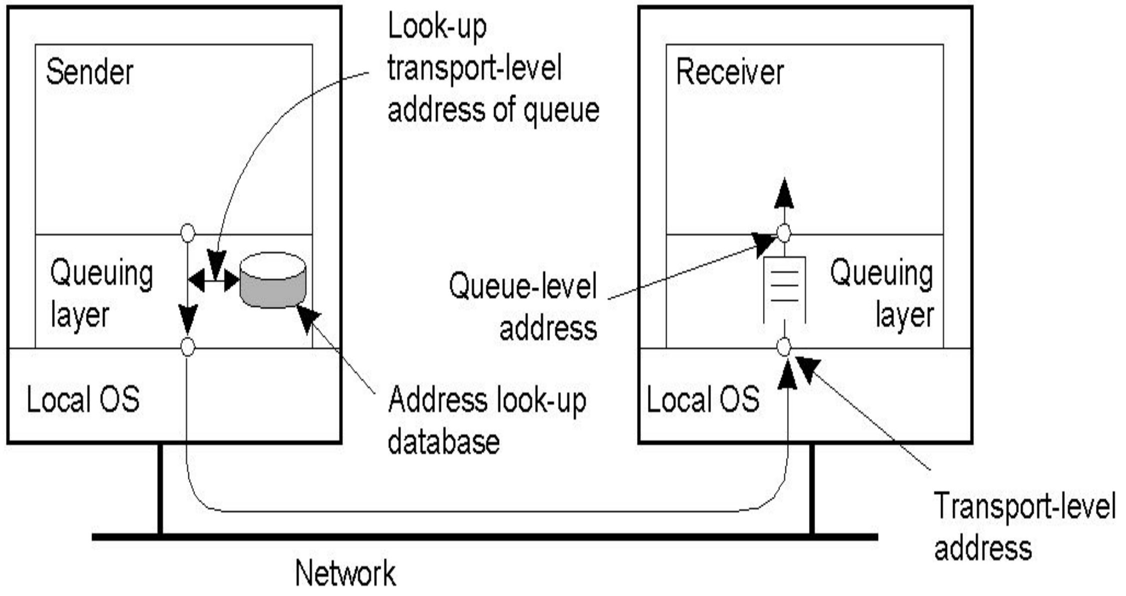
Architecture of a message-queuing system

General architecture of a message-queuing system

- Queues are managed by queue managers.
- Static mapping is easier, dynamic mapping is more complex.



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.

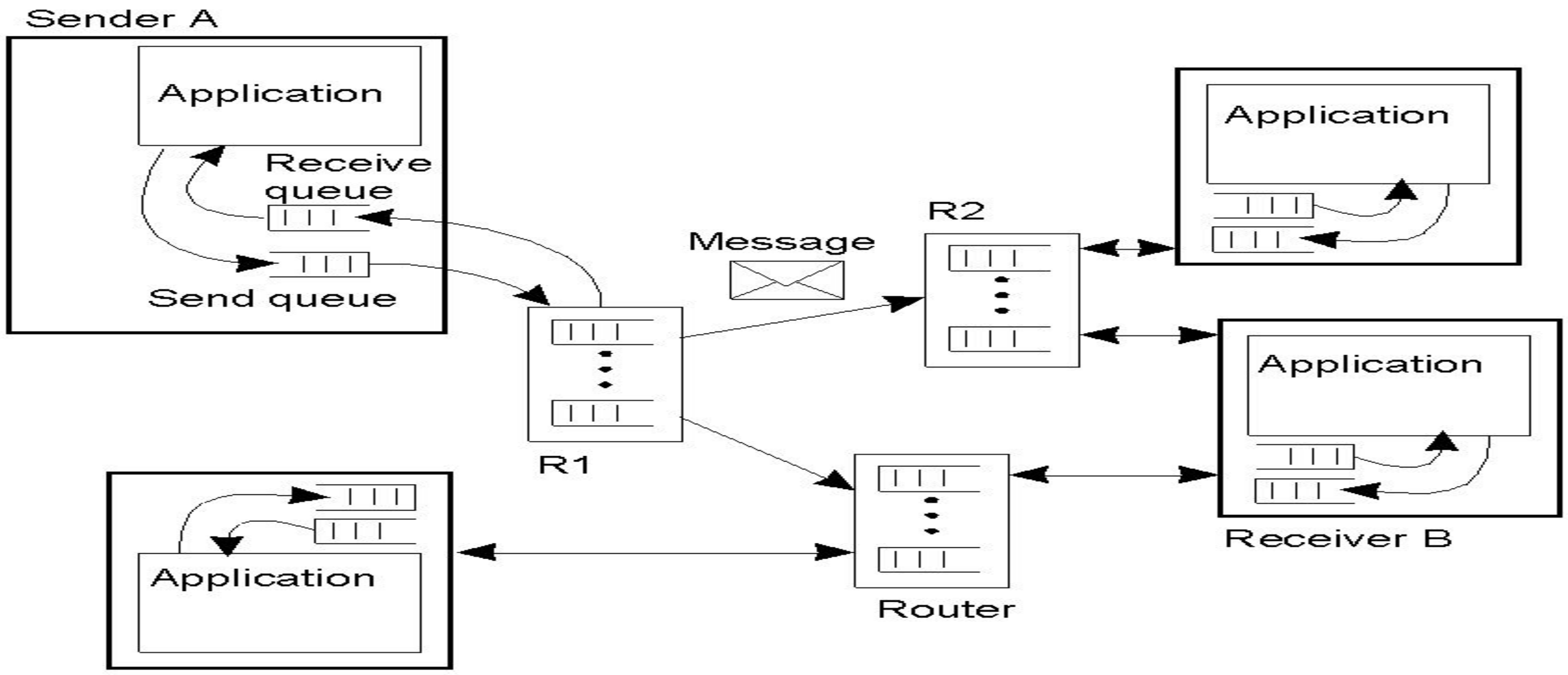


Message-Queuing System Architecture

- 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.

Message-Queuing System with routers

- Routers know about network, queue manager know the nearest router
- Only routers need to be updated when queues are added or removed



Stream-Oriented Communication

- RPC, RMI, message-oriented communication are based on the exchange of discrete messages
  - Timing might affect performance, but not correctness
- In stream-oriented communication the message content must be delivered at a certain rate, as well as correctly.
  - e.g., music or video

Data Streams

- Data stream = sequence of data items
- Can apply to discrete, as well as continuous media
    - e.g. TCP/IP connections which are both byte oriented (discrete) streams
- Audio and video require continuous data streams between file and device.

Streams

Types of Streams

- Simple streams have a single data sequence
- Complex streams have several substreams, which must be synchronized with each other; for example a movie with
  - One video stream
  - Two audio streams (for stereo)
  - One stream with subtitles

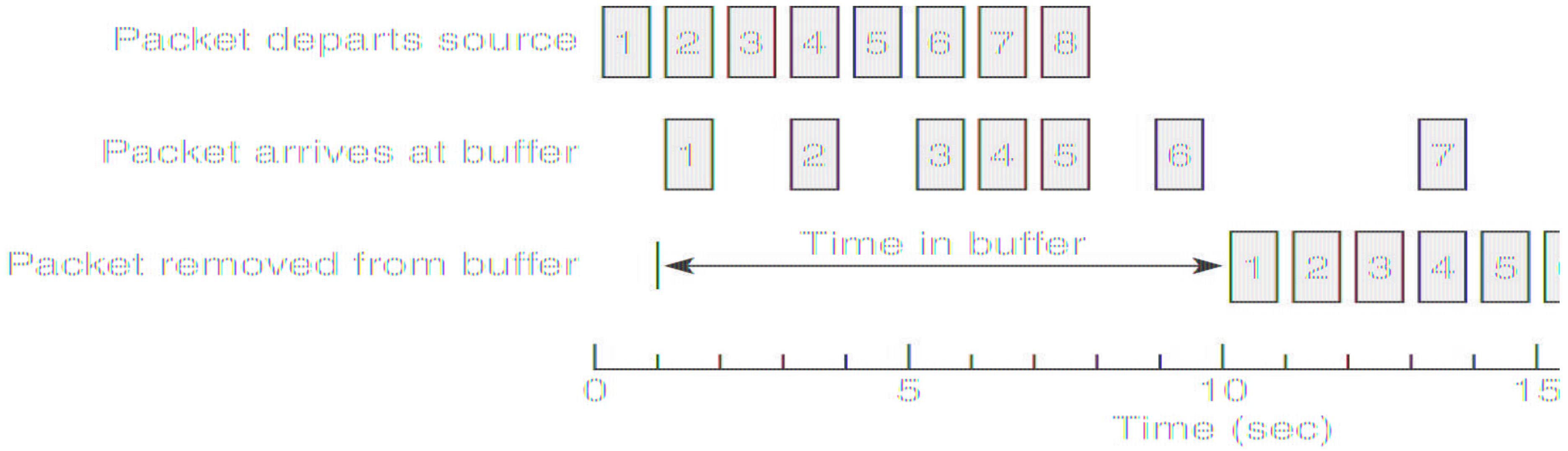
Data Streams

- **Asynchronous transmission mode:** the order is important, and data is transmitted one after the other.
- **Synchronous transmission** mode transmits each data unit with a guaranteed upper limit to the delay for each unit.
- **Isochronous transmission** mode have a maximum and minimum delay. A signal in which the time interval separating any two significant instants is equal to the unit interval
  - Not too slow, but not too fast either

Distributed System Support

Quality of the transmission

- Data compression, particularly for video
- Synchronization
- Reduce jitter Using a buffer



Group/Multicast Communication

- Multicast: sending data to multiple receivers simultaneously .
- Multicast may one to many or many to many .
- Network- and transport-layer protocols for multicast bogged down at the issue of setting up the communication paths to all receivers.
- Peer-to-peer communication using structured overlays can use application-layer protocols to support multicast

Thank you !!!