System Models

Architectural System Model

- An architectural model of a distributed system is concerned with the placement of its parts and relationships between them.
- Examples
  - Client-server
  - Peer-to-peer
- The client-server system can be modified by:
  - Partition of data or replication at cooperating servers
  - Caching of data by proxy servers or clients
  - Use of mobile code
  - Support of addition/removal of mobile devices

Architectural System Model

- Interaction Model
  - Deals with performance and the difficulty to set time limits (e.g., in message delivery).
- Failure Model
  - (gives a precise specification of the faults of the processes and the links -> reliable communication – correct processes)
- Security Model
  - Possible threats (for processes, links)

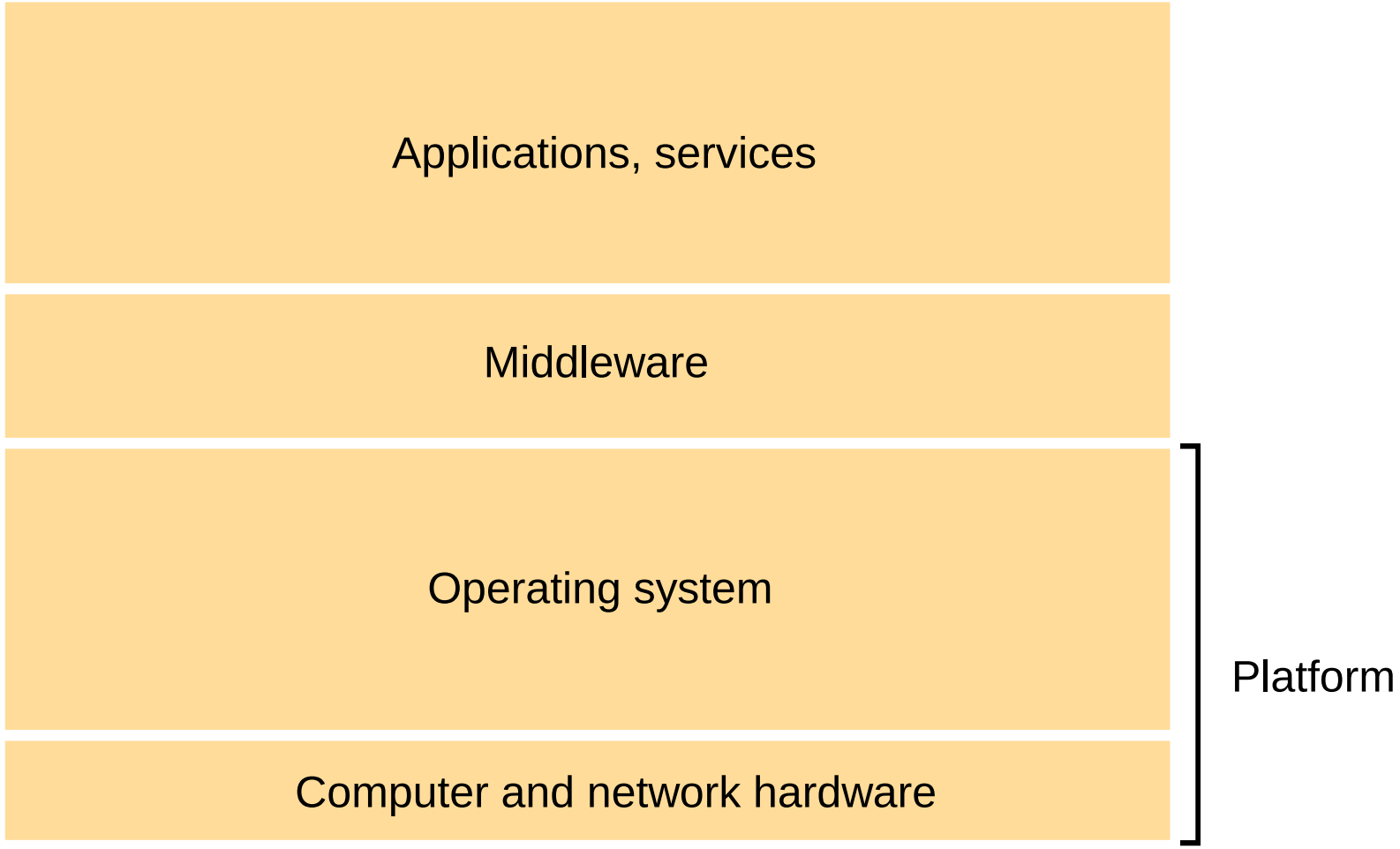
Architectural Models

- Architecture is the structure of a system in terms of separately specified components.
- It abstracts the functions of the individual components.
- It considers the placement of the components across the computer network.
- Considers the interrelationships between the components.
- Classification of processes:
  - Servers, clients, peers
  - Identifies responsibilities, helps to assess their workloads, determines the impact of failures in each of them.

Software Layers

- Software architecture refers to services offered and requested between processes located in the same or different computers.
  - Service layers
- Distributed service
  - One or more server processes
  - Client processes
- Platform -> the lowest level hardware and software layers
  - Examples: Intel x86/Windows, Intel x86/Solaris, PowerPC/MAC OS, Intel x86/Linux
- Middleware
  - A layer of software which masks heterogeneity and provides a convenient programming model to application programmers.
  - Provides useful building blocks:
    - Remote method invocation, communication between a group of processes, notification of events, partitioning, placement and retrieval of data objects, replication, transmission of multimedia data in real time

Software and hardware service layers in distributed systems



Middleware

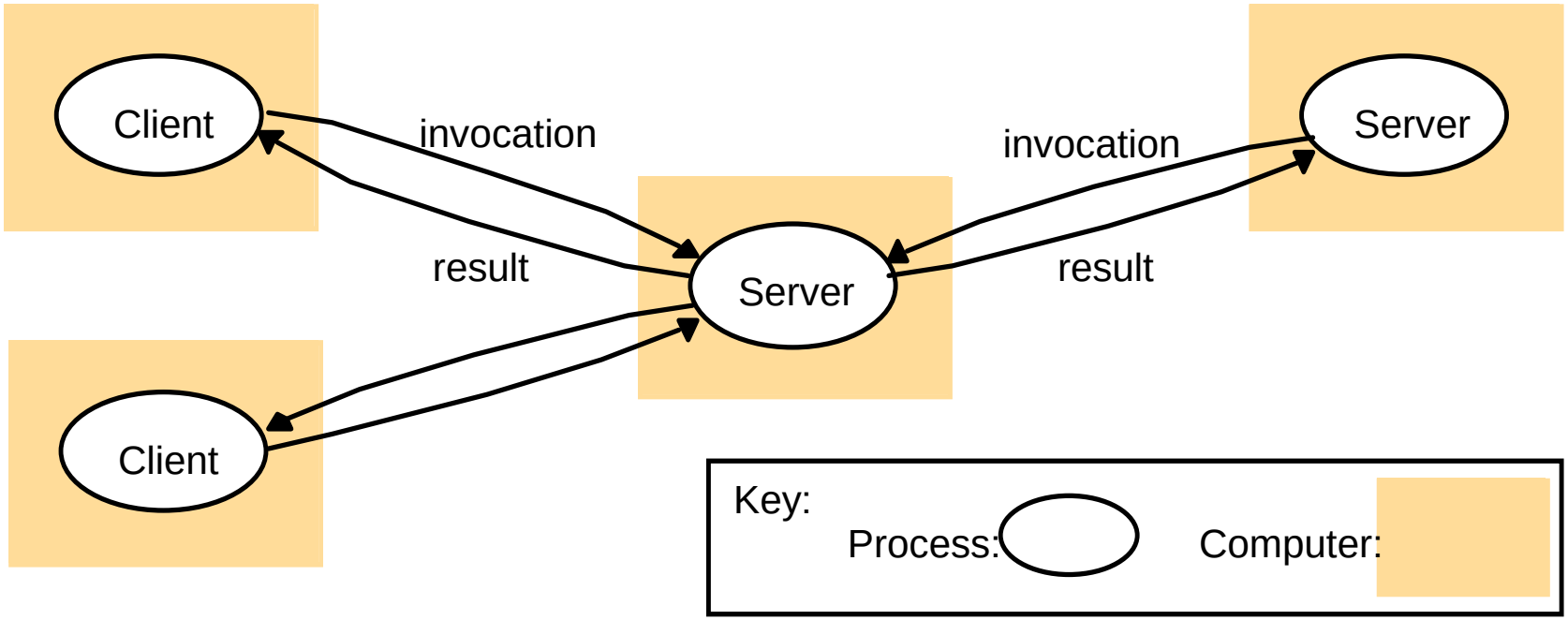
Examples

- Remote procedure calling packages
  - Sun RPC
- Group Communication Systems
  - ISIS
- CORBA
- JAVA RMI

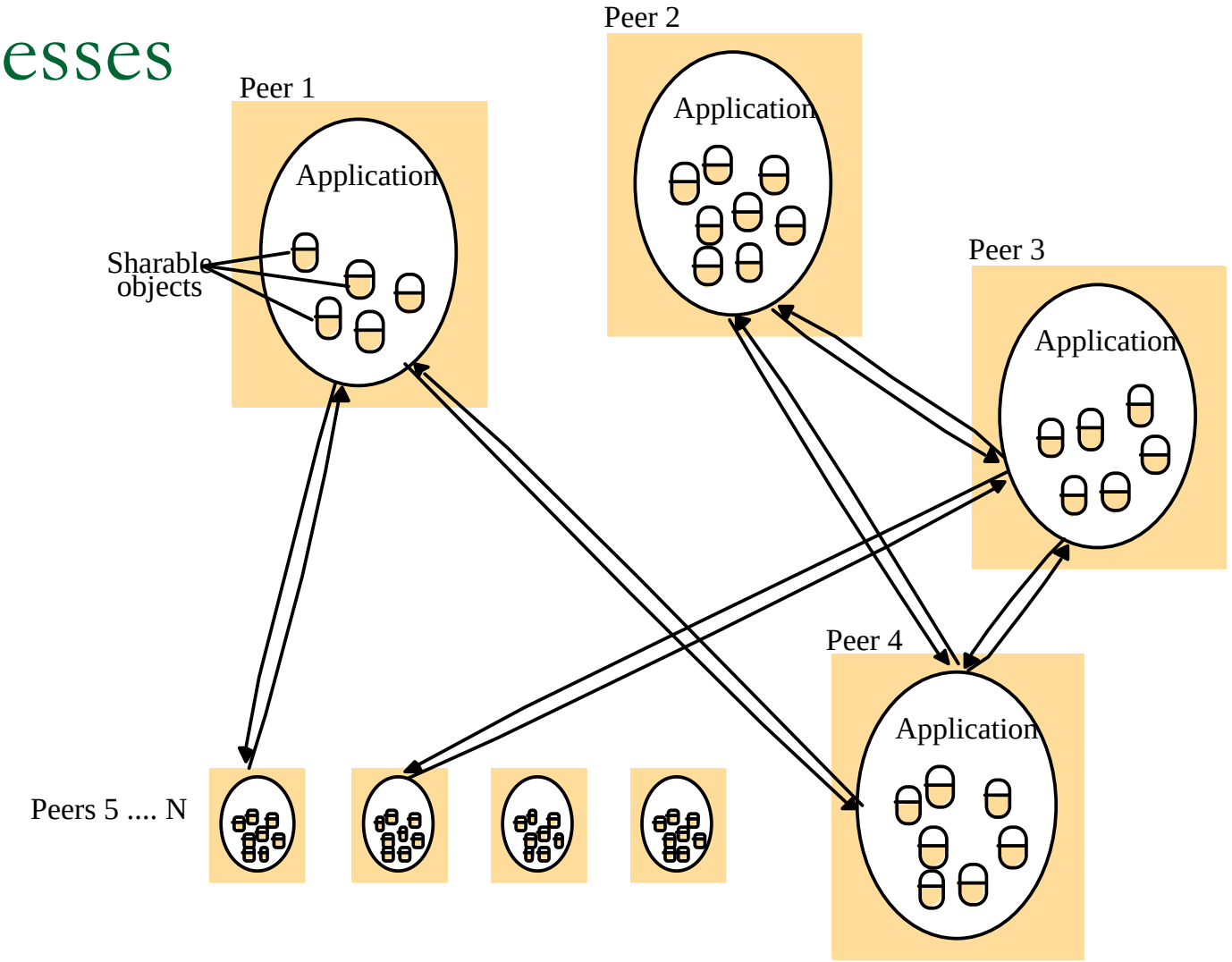
Middleware Limitations

- Some aspects of the dependability of systems require support at the application level.
  - Transfer of large electronic mail messages (or files) from the mail host of the sender to that of the recipient.
    - Using TCP may not be enough since there may occur some major network interruption.
    - The service should provide additional levels of fault tolerance (e.g., a record of progress, etc.)
- Correct behavior depends upon checks, error-correction mechanisms and security measures at many levels, some of which require access to data within the application's address space.

Clients invoke individual servers



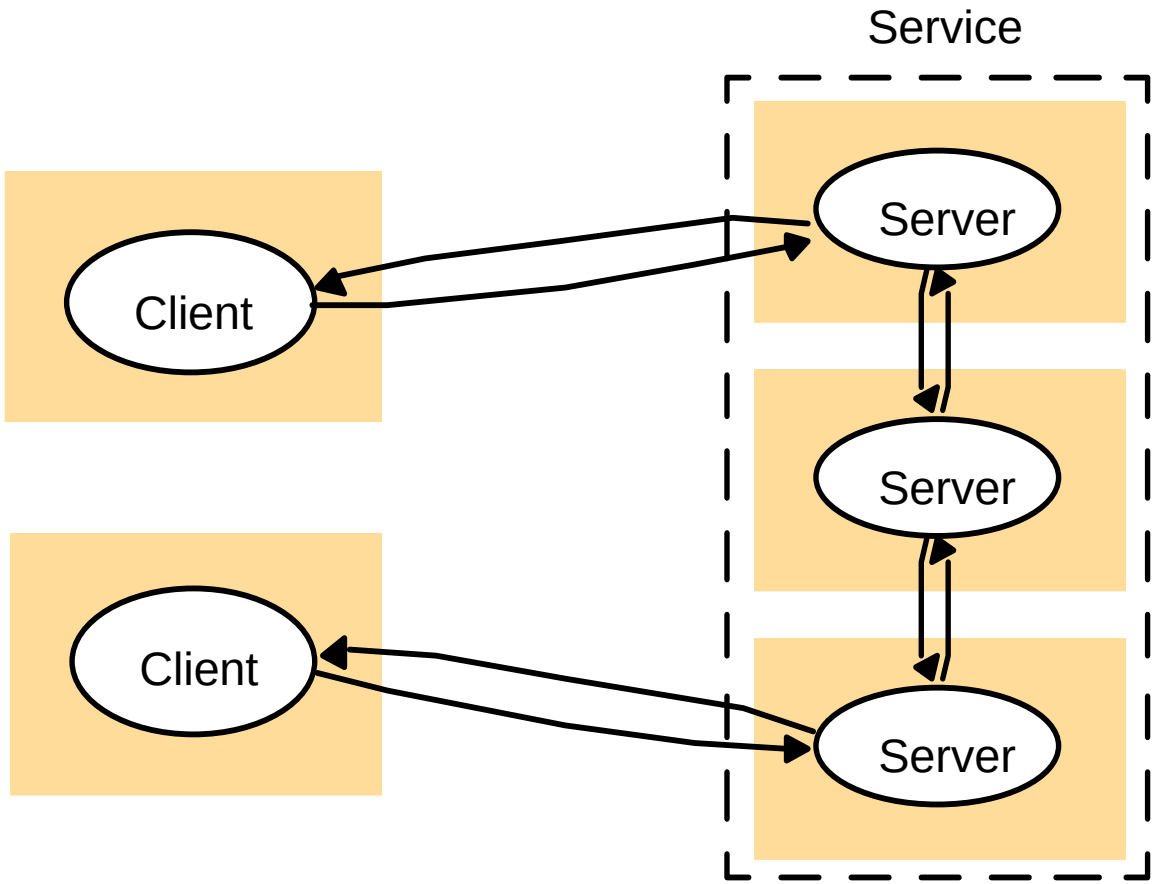
A distributed application based on peer processes



Variations

- Use of multiple servers and caches to increase performance and resilience
- Use of mobile code and mobile agents
- User's need for low-cost computers with limited hardware resources that are simple to manage
- Requirement to add and remove mobile devices in a convenient manner

A service provided by multiple servers



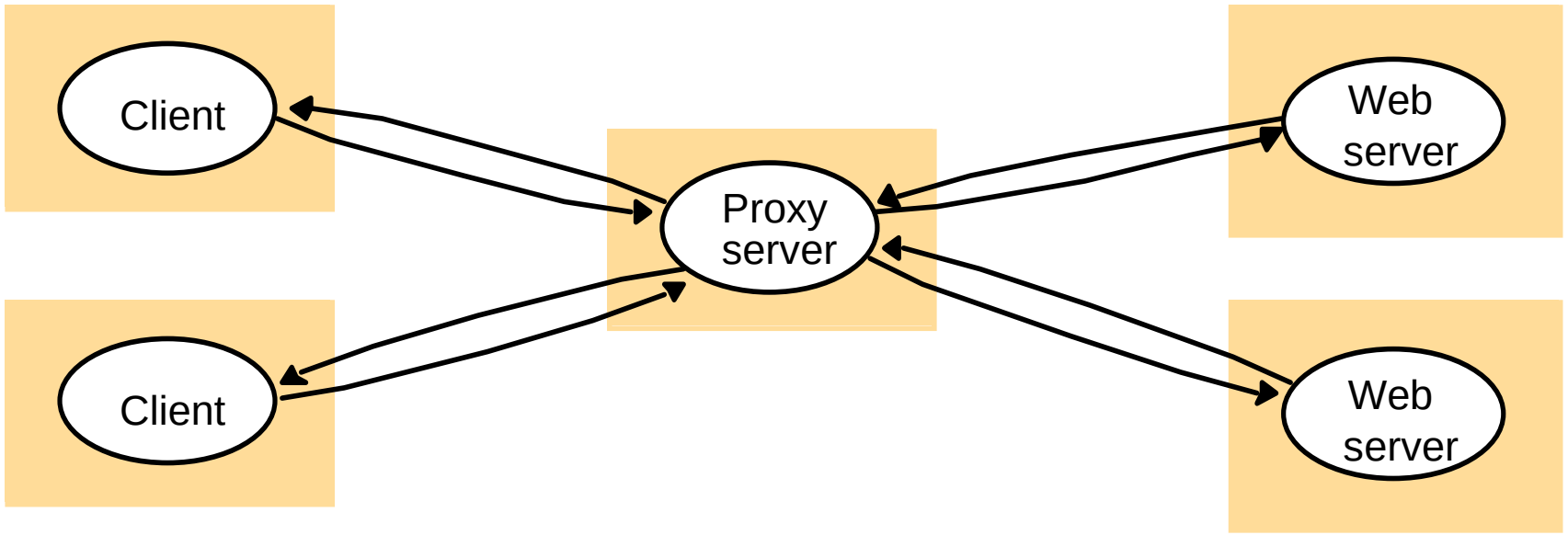
-Partition the set of objects and distribute them between themselves

- maintain replicated copies

- Web (1st approach)

-SUN Network Information Service (replication of password file at each site)

Web proxy server



A cache is a store of recently used data objects that is closer to the objects themselves.

Web proxy servers provide a shared cache of web resources for the client machines at a site or across several sites.

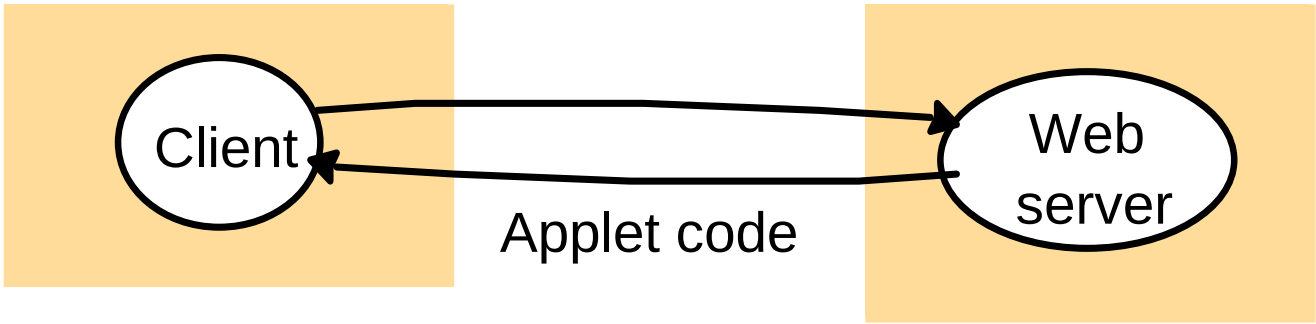
•Increase availability

•Increase performance

•Reduce the load on the wide-area network and web server

Mobile Code - Web applets

a) client request results in the downloading of applet code



b) client interacts with the applet

- Good interactive response

Mobile Agents

A mobile agent is a running program that travels from one computer to another in a network carrying out a task on someone's behalf.

• collecting information

A mobile agent may make many invocations to local resources at each site it visits (e.g., access to individual database entries).

Might be used to:

• install software on the computers of an organization

• compare the prices of products of vendors

Negative aspects

• mobile agents are a potential security threat to the resources

•they can be themselves vulnerable (they may not be able to complete their tasks if they are refused access to the information needed)

Other means by which the tasks can be performed

• remote invocations -> Web crawlers

Mobile Devices and Spontaneous Interoperation

- Mobile devices are hardware computing components that move between physical locations and thus networks, carrying software components with them.
- Mobile clients and systems may co-exist on mobile devices
  - Bus that moves around on a world tour
    - WiFi connectivity when available
    - Lower-bandwidth wide-area telecommunications connectivity elsewhere.
- Mobile transparency
- Variable connectivity
- Spontaneous interoperation (variation of client-server: associations between devices are routinely created and destroyed)

Design Requirements for Distributed Architectures

- Performance Issues
  - Responsiveness
    - The speed of a remote invocation depends on:
      - The load and performance of the server and network
      - Delays in all the software components (client and server operation systems and middleware, code of the process that implements the service)
    - Transfer of data is slow
  - Throughput
    - Rate at which computational work is done
    - Perform work for all users
  - Balancing of Computational Loads
    - Applets remove load from the server
    - Use several computers to host a single service

Design Requirements for Distributed Architectures

- Quality of Service
  - Reliability (the ability of a system to perform and maintain its function in routine circumstances, as well as hostile and unexpected circumstances.
  - Security
  - Performance
  - Adaptability (the ability of a system to adapt itself efficiently and fast to changed circumstances)
  - Resource Availability

- Some applications handle time-critical data – streams of data that are required to be processed or transferred from one process to another at a fixed rate.

Design Requirements for Distributed Architectures

- Caching and Replication
  - Cached copies of resources should be kept up-to-date when the resource at a server is updated.
  - A variety of cache-coherency protocols are used to suit different applications.
- Web Caching Protocol
  - Web browsers and proxy servers cache responses to client requests from web servers
  - The cache-consistency protocol can provide browsers with fresh copies of the resources held by the web server, but for performance reasons the freshness condition can be relaxed.
  - A browser or proxy can validate a datum with the server.
  - Web servers assign approximate expiry times to their resources
  - The expiry time of the resource and the current time at the server are attached to the response.

Design Requirements for Distributed Architectures

- Dependability Issues
  - Dependable applications should continue to function correctly in the presence of faults in hardware, software and networks.
- Attributes
  - Availability – readiness for correct service
  - Reliability is achieved through redundancy – the provision of multiple resources so that the system and application software can reconfigure to continue to perform its tasks in the presence of faults.
    - Replication
      - Multiple computers – multiple communication paths
      - Several replicas of a data item
      - Retransmission of messages, etc.
  - Safety – absence of incorrect consequences
  - Integrity – absence of improper system alteration
  - Maintainability – ability to undergo modifications and repairs

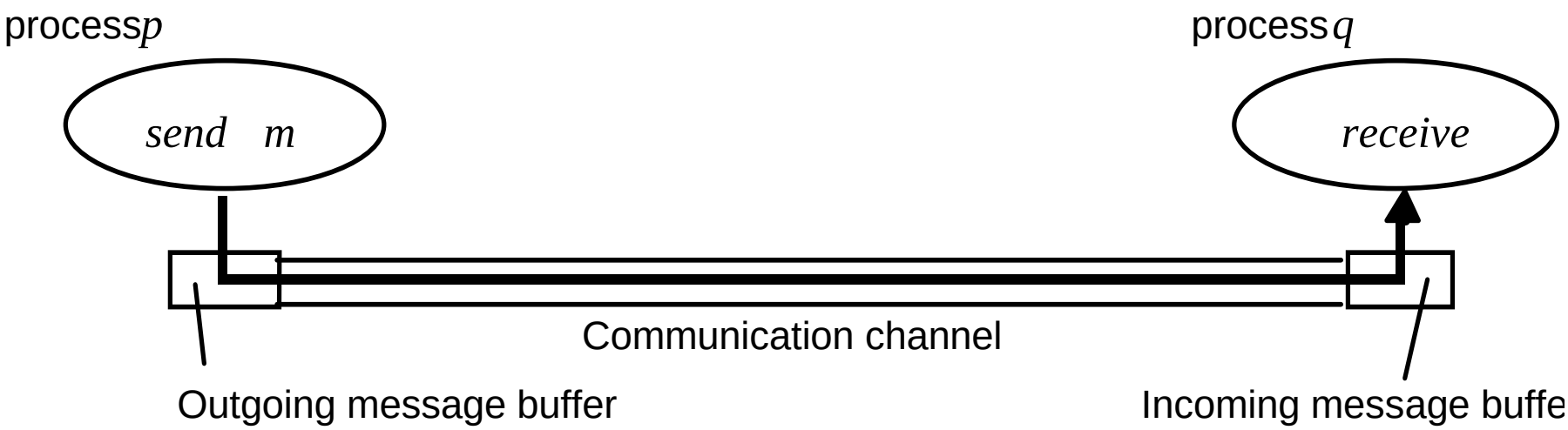
Fundamental Models

- A model contains only the essential ingredients needed to understand and reason about some aspects of a system's behavior.
- A system model has to address the following:
  - What are the main entities in the system?
  - How do they interact?
  - What are the characteristics that affect their individual and collective behavior?
- Purpose
  - Make explicit all the relevant assumptions about the system we are modeling
  - Make generalizations concerning what is possible or impossible, given those assumptions.
    - General purpose algorithms
    - Desirable properties
- Interaction
  - Communication takes place with delays
  - Maintaining the same notion of time across all nodes of a distributed system is difficilt.
- Failure

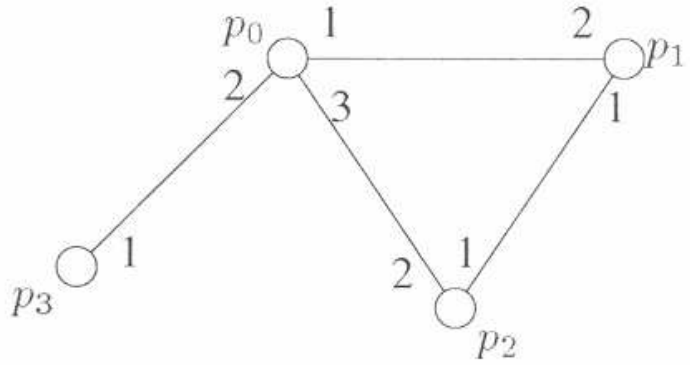
Interaction Model

- The behavior and state of distributed systems can be described by a distributed algorithm
  - A definition of the steps to be taken by each of the processes, including the transmission of messages between them.
- Messages are transmitted to transfer information between processes and to coordinate their activity.
- The computing rates of processes and the timing of the transmission of messages cannot in general be predicted.
- Each process has its own state, consisting of the set of data that it can access and update (i.e., its local variables).

Processes and Channels



Formal Model of Message-Passing Systems



- There are n processes in the
- system: p_0, .., p_{n-1}
- Each process is modeled as a state machine.
- The state of each process is comprised by its local variables and a set of arrays. For instance, for p_0, the state contains six arrays:
- $inbuf_0[1]$, …, $inbuf_0[3]$: contain messages that have been sent to p_0 by p_1, p_2 and p_3, respectively, but p_0 has not yet processed.
- $outbuf_0[1]$, …, $outbuf_0[3]$: messages that have been sent by p_0 to p_1, p_2, and p_3, respectively, but have not yet been delivered to them.

Formal Model of Message-Passing Systems

- The state of process p_i excluding the $outbuf_i[l]$ components, comprises the accessible state of p_i.
- Each process has an initial state in which all inbuf arrays are empty.
- At each step of a process, all messages stored in the inbuf arrays of the process are processed, the state of the process changes and a message to each other neighboring process can be sent.
- A configuration is a vector $C = (q_0, .., q_{n-1})$ where q_i represents the state of p_i.
- The states of the outbuf variables in a configuration represent the messages that are in transit on the communication channels.
- In an initial configuration all processes are in initial states.

Formal Model of Message-Passing Systems

- Computation event, **comp(i)**
  - Represents a computation step of process p_i in which p_i's transition function is applied to its current accessible state.
- Delivery Event, **del(i,j,m)**
  - Represents the delivery of message m from processor p_i to processor p_j.
- The behavior of a system over time is modeled as an execution, which is a sequence of configurations alternating with events.
- This sequence must satisfy a variety of conditions.
  - Safety conditions
    - Hold in every finite prefix of the execution (states that nothing bad has happened yet)
  - Liveness conditions
    - Hold a certain number of times (states that eventually something good must happen)

Performance of Communication Channel

- Latency

  The delay between the start of a message's transmission from one process and the beginning of its receipt by another.

- Bandwidth

  Total amount of information that can be transmitted over it in a given time. The latency includes:

  - Time taken for the first of a string of bits transmitted through the network to reach its destination + delay in accessing the network + time taken by the OS communication services at both sender and receiver

- Jitter

  The variation in the time taken to deliver a series of messages.

Computer Clocks and Timing Events

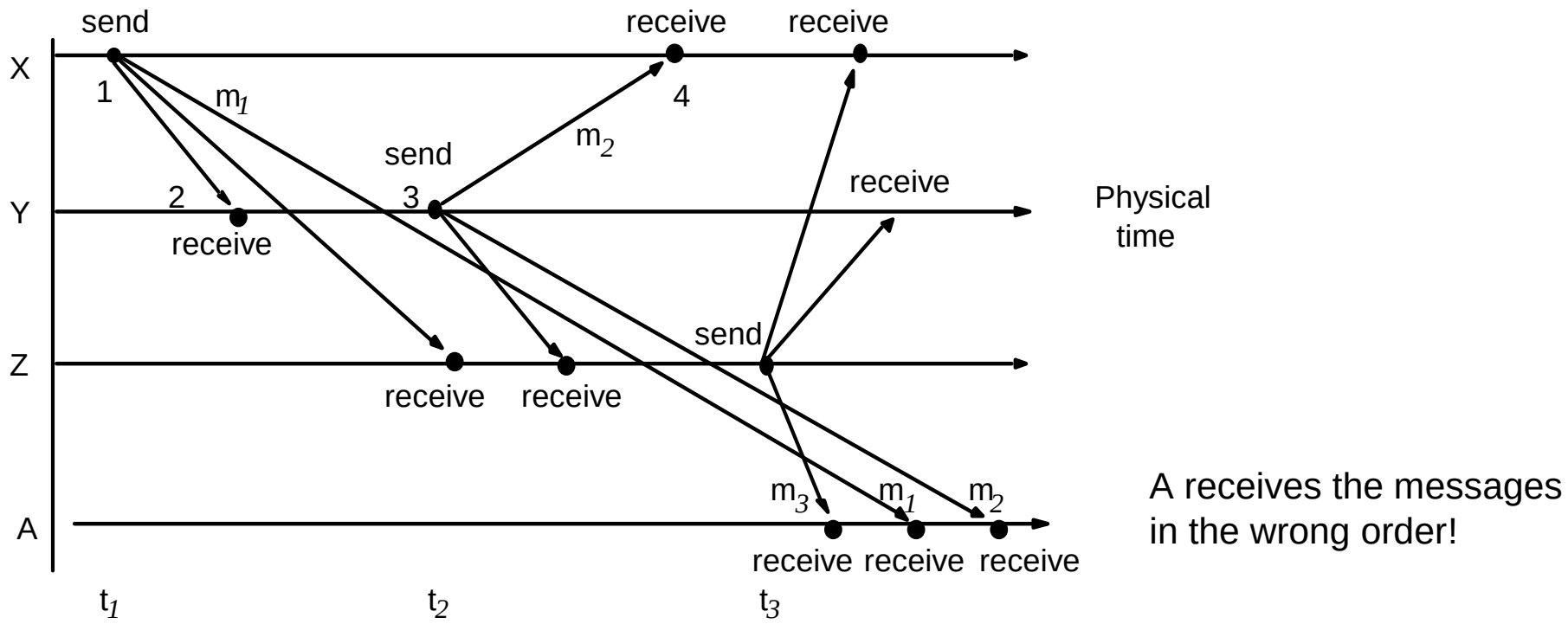
- Local clocks
- Clock drifts from perfect time and their drift rates differ from one another
- Clock Drift Rate: the relative amount that a computer clock differs from a perfect reference clock.
- Even if clocks are set at the same time initially, they would eventually vary unless corrections are applied periodically.

Interaction (Timing) Models

- **(Partially-Fully) Synchronous Systems**
  - There is a fixed upper bound Δ on the time for messages to be delivered (communication is synchronous).
  - There is a fixed upper bound Φ on the rate at which one processor's clock can run faster than another's (processors are synchronous).
  - If Φ=1 and Δ=1, we talk about a fully synchronous system. Then,
    - The sequence of alternating configurations and events can be partitioned into disjoint rounds.
    - A round consists of a deliver event for every message in an outbuf variable, until all outbuf variables are empty, followed by one computation event for every processor.
- **Asynchronous Systems**

  There is no fixed upper bound on how long it takes for a message to be delivered or how much time elapses between consecutive steps of a processor. Clock drift rates may be also arbitrary.

Real-Time Ordering of Events



A receives the messages in the wrong order!

Relative order in which events take place in a system, there are relationships between events in distributed systems (causality)

Logical clock, logical timestamp

Omission and arbitrary failures

Class of failure	*Affects*	*Description*
Fail-stop	Process	Process halts and remains halted. Other processes may detect this state.
Crash	Process	Process halts and remains halted. Other processes may not be able to detect this state.
Omission	Channel	A message inserted in an outgoing message buffer never arrives at the other end's incoming message buffer.
Send-omission	Process	A process completes a*send*, but the message is not put in its outgoing message buffer.
Receive-omission	Process	A message is put in a process's incoming message buffer, but that process does not receive it.
Arbitrary (Byzantine)	Process or channel	Process/channel exhibits arbitrary behaviour: it may send/transmit arbitrary messages at arbitrary times, commit omissions; a process may stop or take an incorrect step.

Timing failures

Class of Failure	*Affects*	*Description*
Clock	Process	Process's local clock exceeds the bounds on its rate of drift from real time.
Performance	Process	Process exceeds the bounds on the interval between two steps.
Performance	Channel	A message's transmission takes longer than the stated bound.

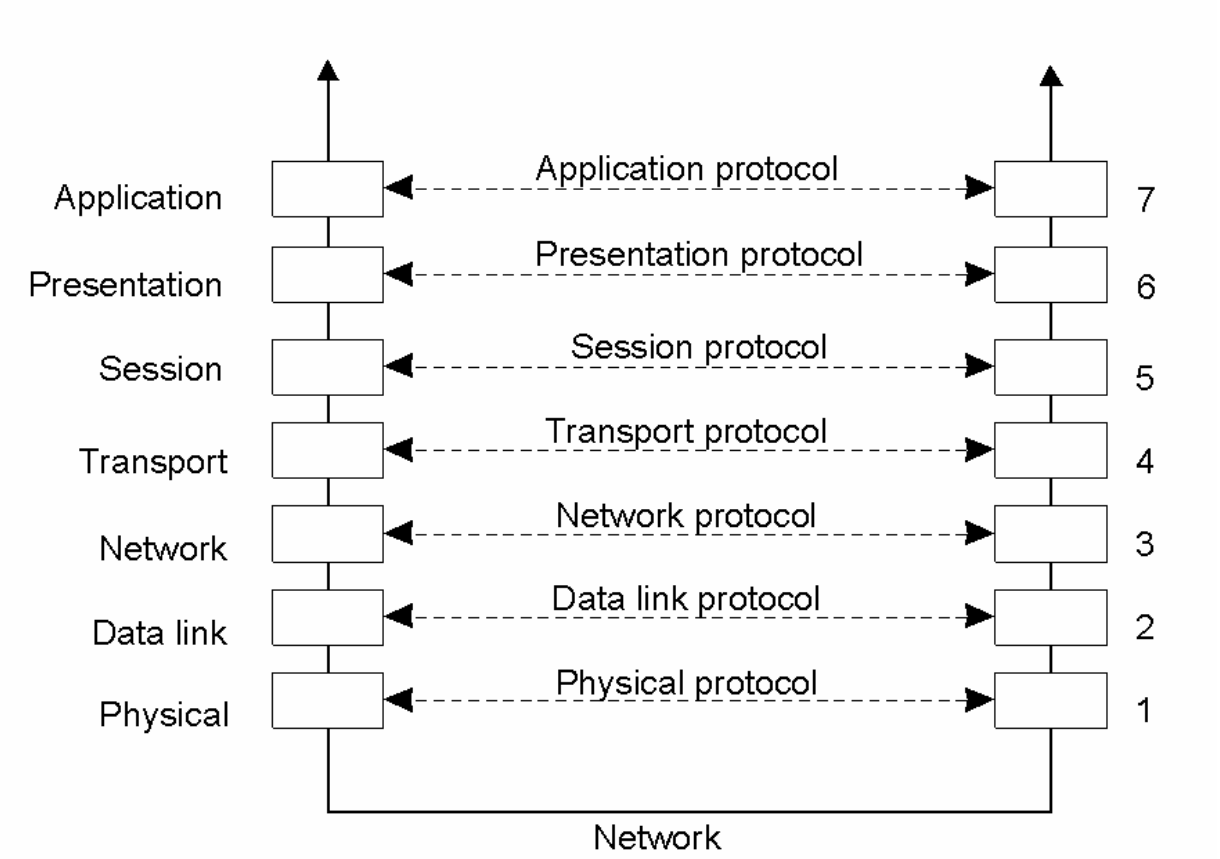
Masking Failures – Reliability of one-to-one Communication

- A service masks a failure, either by hiding it all together or by converting it into a more acceptable type of failure.
  - Checksums are used to mask corrupting messages -> a corrupted message is handled as a missing message
  - Message omission failures can be hidden by re-transmitting messages.
- The term reliable communication is defined in terms of validity and integrity as follows:
  - Validity: any message in the outgoing buffer is eventually delivered to the incoming message buffer
  - Integrity: the message received is identical to one sent, and no messages are delivered twice.

Communication

Layered Protocols

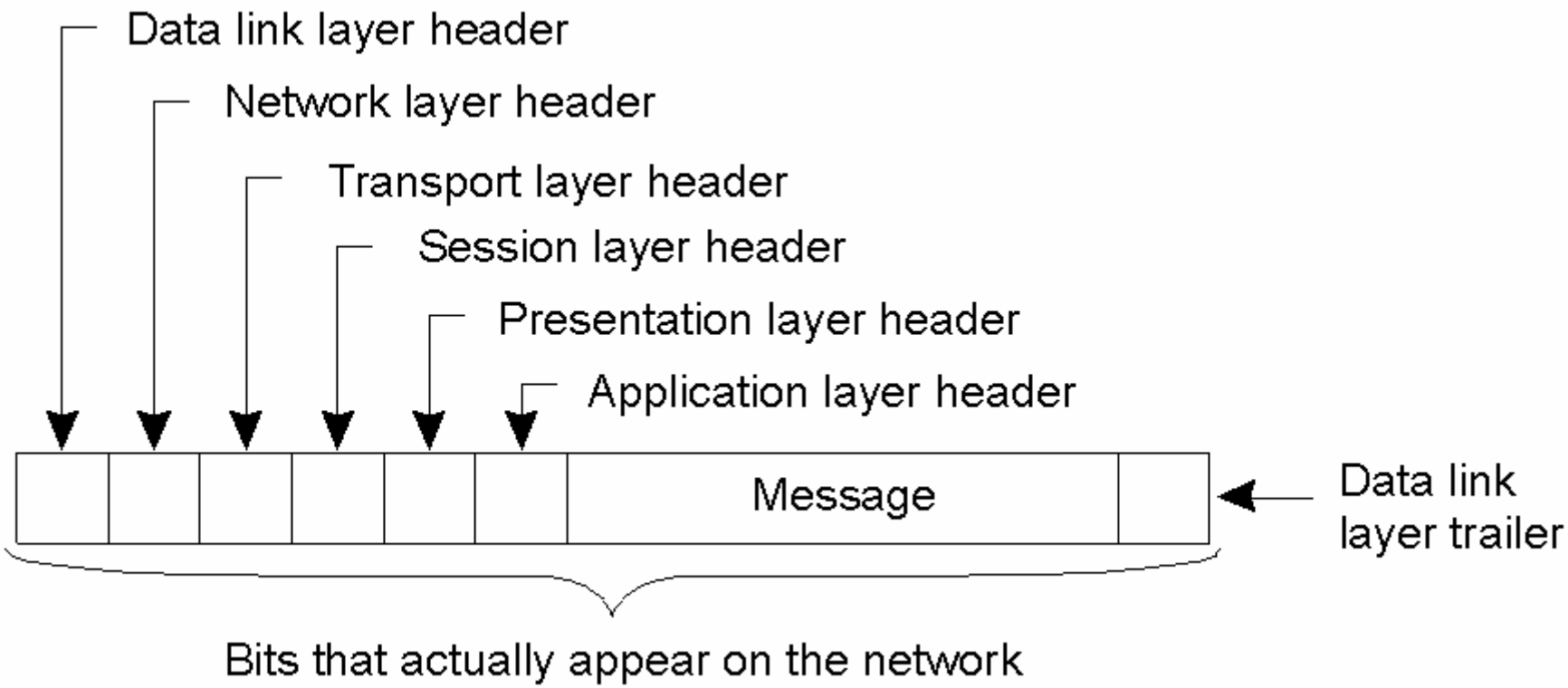
- Layers, interfaces, and protocols in the OSI model.



Layered Protocols

- International Standards Organization (ISO)
  - Developed a reference model that clearly identifies the various levels involved, gives them standard names, and points out which level should do which job.
- Open systems Interconnection Reference Model
- Protocols: rules determining how an open system can communicate with another open system.
- Connection-oriented protocols: Before exchanging data the sender and receiver first explicitly establish a connection, and possibly negotiate the protocol they will use
- Connectionless protocols: No setup in advance is needed.
  - Dropping a letter in a mailbox

Layered Protocols



Layered Protocols

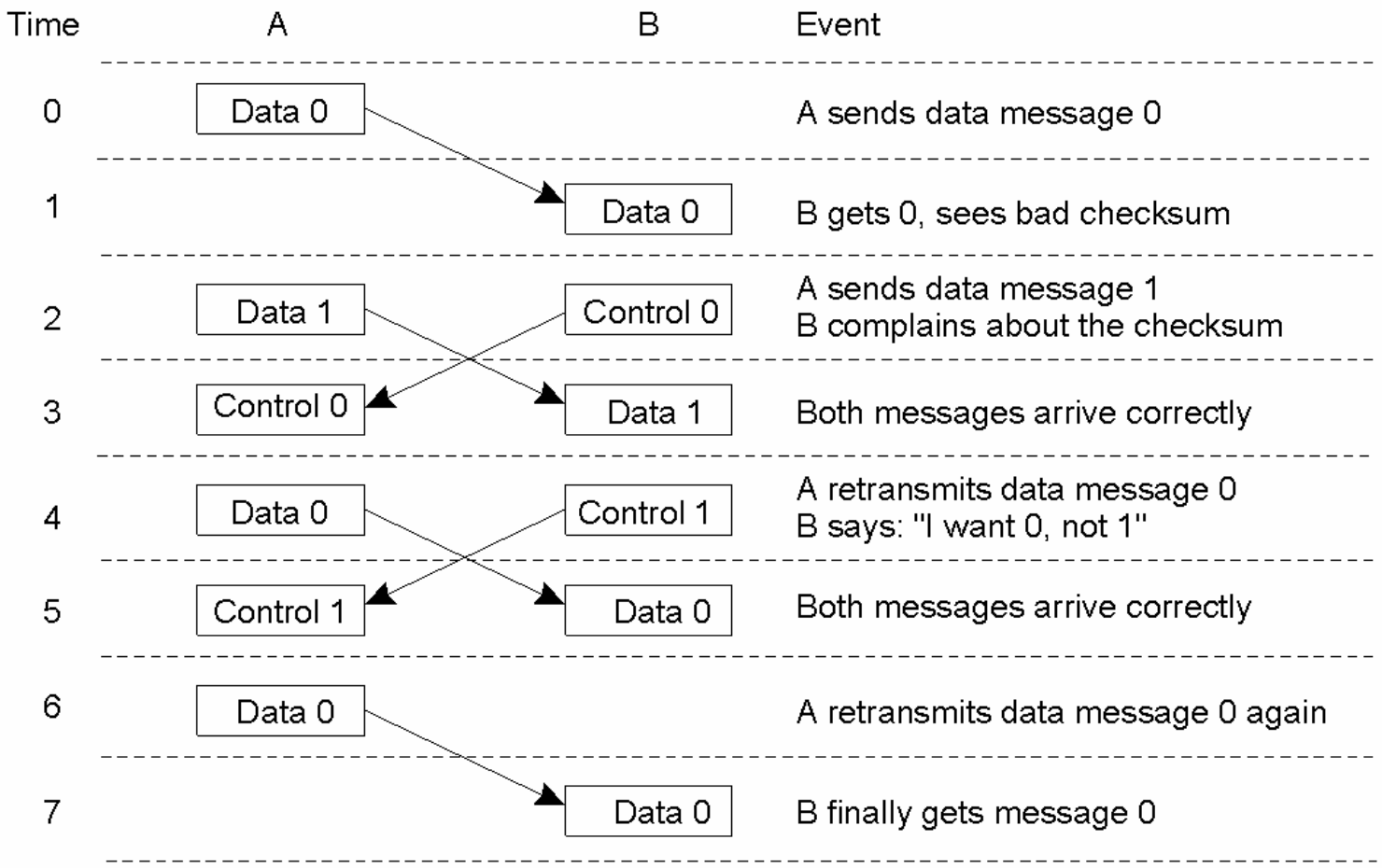
Physical Layer

- Undertakes the actual transmission of the message.
  - How many volts to use for 0 and 1?
  - How many bits per second can be sent?
  - Can transmission take place in both directions simultaneously?
  - Size and shape of the network connector
  - Number of pins and meanings of each

Data Link Layer

- Provide mechanisms to detect and correct errors during the bit transmission
  - Bits are grouped into units called frames.
  - A special bit pattern is placed at the beginning and at the end of each frame to mark it.
  - A checksum is computed by adding up all the bytes in the frame in a certain way.
  - Frames are assigned sequence numbers

Data Link Layer



- Discussion between a receiver and a sender in the data link layer.

Layered Protocols

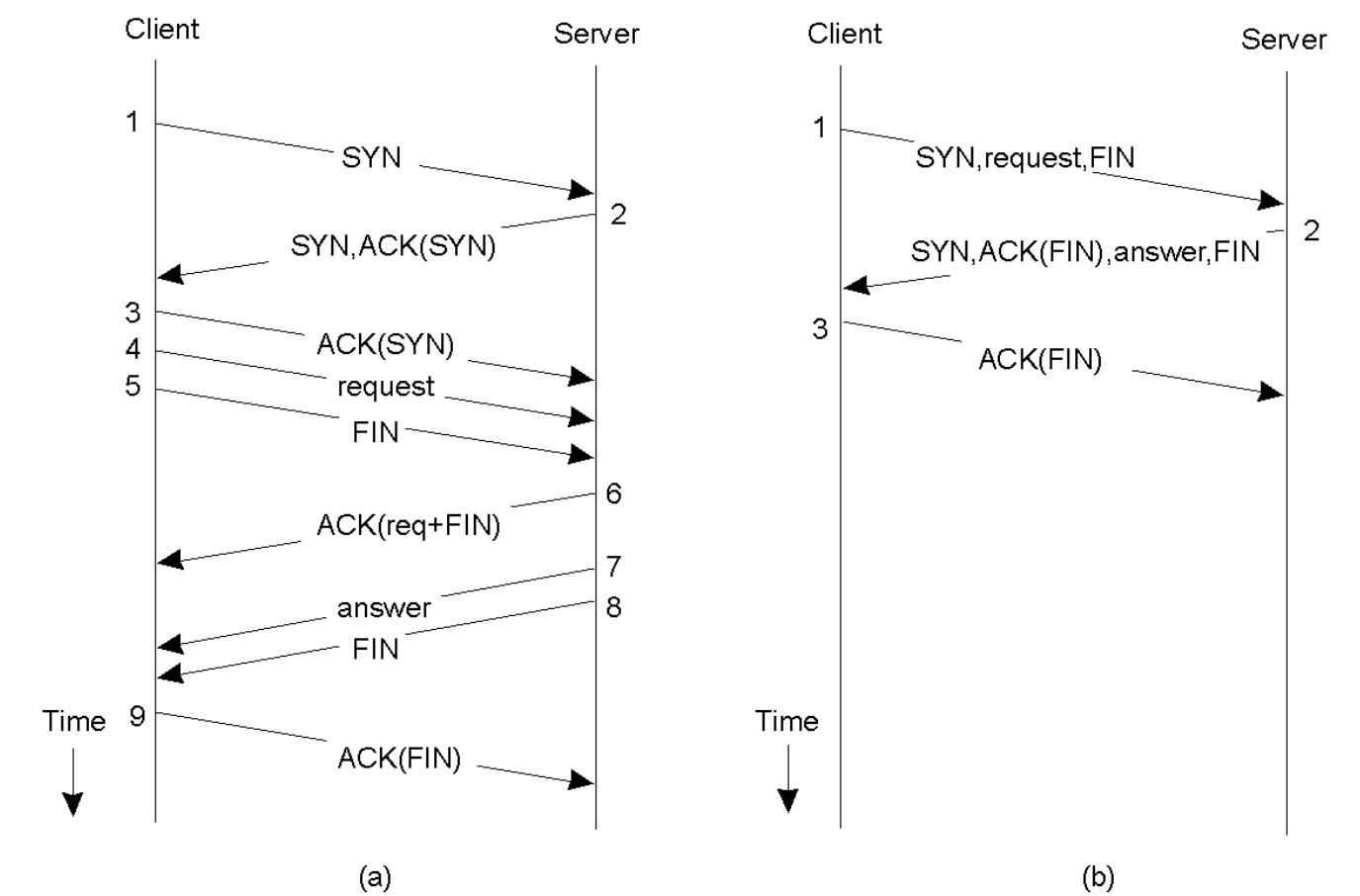
Network Layer

- The choosing of the best path from the sender to the receiver is called routing.
- Routing is the major task of the network layer.
- Example of Connectionless protocol: IP
  - No connection setup.
  - Each IP packet is routed to its destination independently of all others.
  - No internal path is selected and remembered.
- Example of Connection-oriented protocol – ATM Virtual Channels
  - A unidirectional connection from the source to the destination is established.
  - A collection of virtual channels between two hosts comprise a virtual path.

Transport Protocols

- Delivers messages without loss.
- Each message is broken into pieces called packets
- A sequence number is assigned to each packet
- Transport Layer Header
  - Which packets have been sent
  - Which have been received
  - How many more the receiver has room to accept
  - Which should be retransmitted

TCP & Client-Server TCP (TCP for Transactions, T/TCP)



Higher Level Protocols

Session Layer

- Provides dialog control, to keep track of which party is currently talking
- Provides synchronization facilities
  - Insert checkpoints into long transfers, so that in the event of a crash, it is necessary to go back only to the last checkpoint

Presentation Layer

- Is concerned with the meaning of the bits
  - It is possible to define records such a name, address, amount of money, and other valuable info that might be contained in a message, and have the sender notify the receiver that a message contains a particular record in a certain format.
  - Makes communication easier between machines with different internal representations

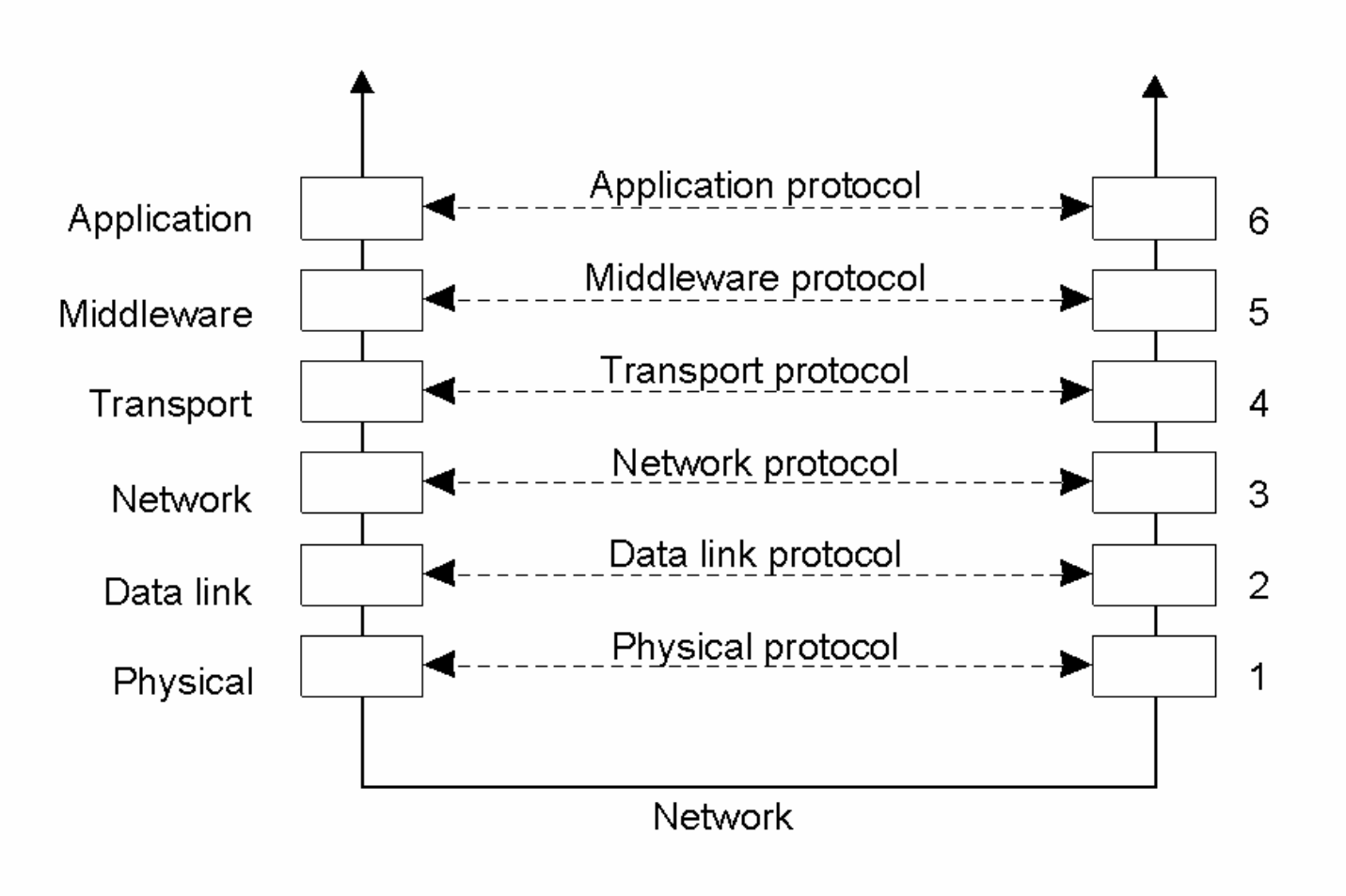
Application Protocols

- Contains a collection of standard network applications, like e-mail, file transfer, terminal emulation, etc.
- FTP
  - FTP protocol versus ftp program
- HTTP (HyperText Transfer Protocol)
  - Remotely manage and handle the transfer of Web pages
    - Web browsers and web servers
    - JAVA RMI

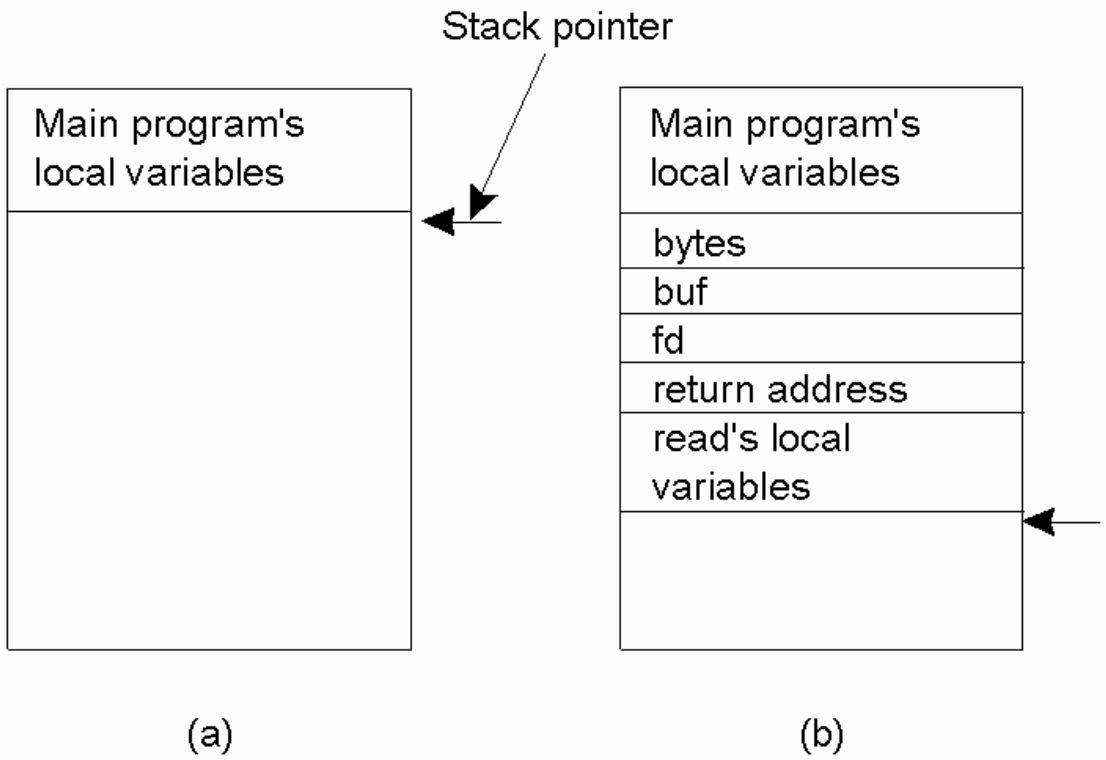
Middleware Protocols

- Middleware is an application.
- However, it often contains many general-purpose protocols that warrant their own layers.
  - Various ways to establish authentication
  - Distributed commit protocols
    - Atomicity: in a group of processes, either all processes carry out a particular operation, or that operation is not carried out at all.
      - Transactions – Fault-tolerant applications
  - Distributed Locking
  - Reliable multicast services

Middleware Protocols



Conventional Procedure Call



C call

count = read(fd,buf,nbytes)

- Call by value
- Call by reference
- Call by copy/restore

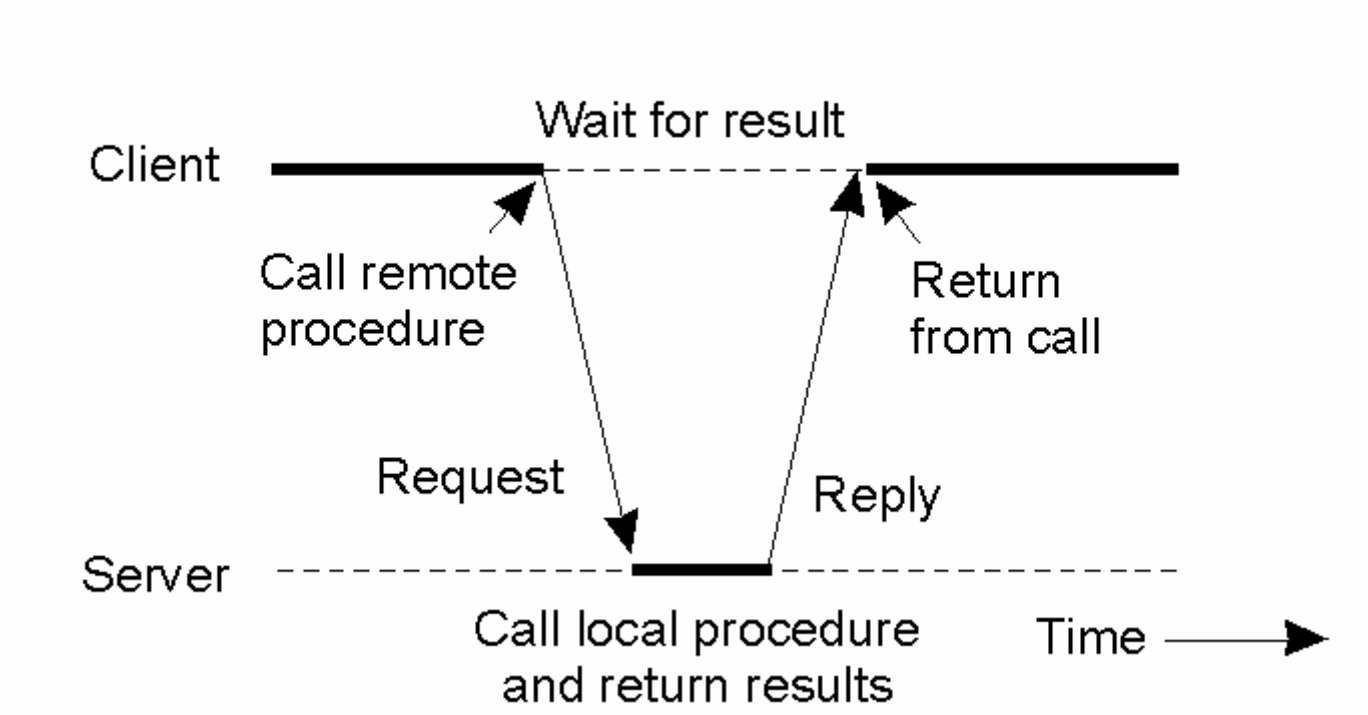
a) Parameter passing in a local procedure call: the stack before the call to read

b) The stack while the called procedure is active

RPC versus Calling a System Call - Read data from a file

Local file	**Remote file**
▪ The programmer calls `read` ▪ The `read` routine is extracted from the library by the linker ▪ It is a short procedure that calls a system call ▪ The `read` procedure is a kind of interface between the user code and the local OS.	▪ The programmer calls `read` ▪ the `read` routine is extracted from the library by the linker (it is now called client stub) ▪ **it packs the parameters into a message and requests that message to be sent to the server** ▪ The read procedure is again a kind of interface. The work hidden under it is however different. ▪ When the message arrives at the server, the OS passes it up to a server stub. ▪ The server stub unpacks the parameters and calls the server procedure in the usual way.

Client and Server Stubs

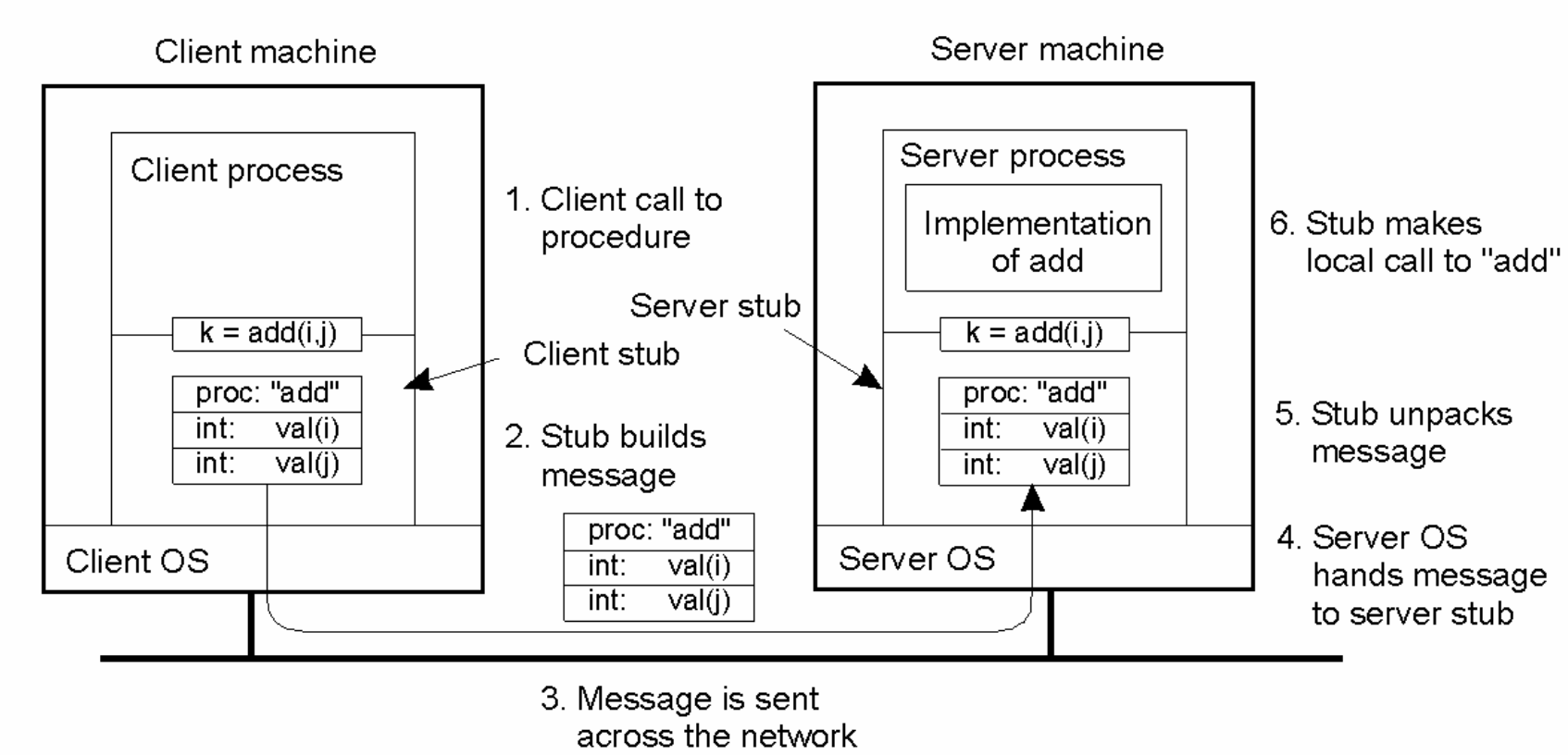


- Principle of RPC between a client and server program.

Steps of a Remote Procedure Call

1. Client procedure calls client stub in normal way
2. Client stub builds message, calls local OS
3. Client's OS sends message to remote OS
4. Remote OS gives message to server stub
5. Server stub unpacks parameters, calls server
6. Server does work, returns result to the stub
7. Server stub packs it in message, calls local OS
8. Server's OS sends message to client's OS
9. Client's OS gives message to client stub
10. Stub unpacks result, returns to client

Passing Value Parameters (1)

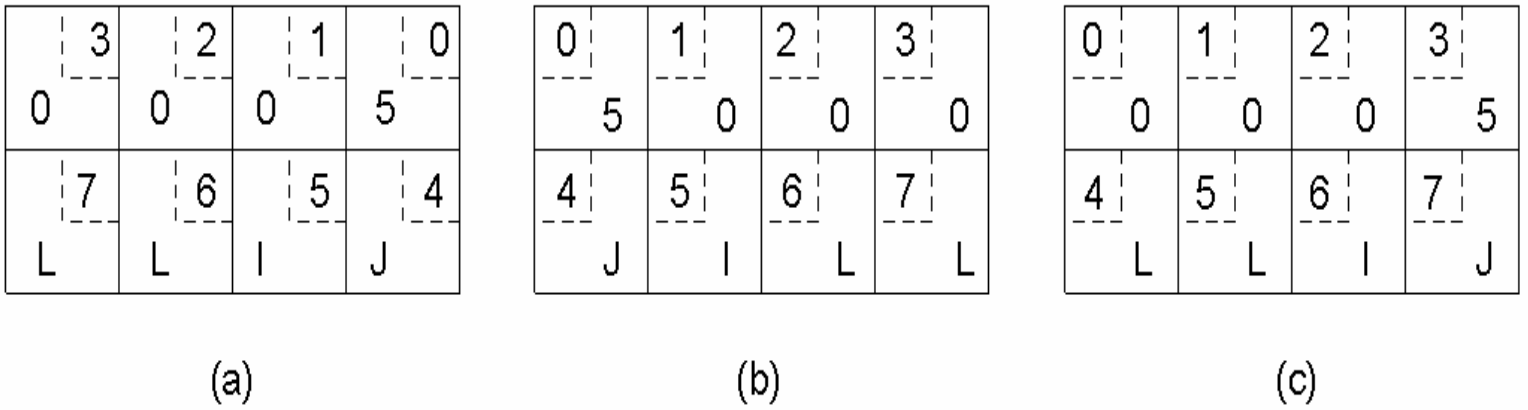


Packing parameters into a message is called parameter marshalling.

Passing Value Parameters (2)

Each machine has its own representation for numbers, characters, and other data items. IBM mainframes use the EBCDIC character code, whereas IBM PCs use ASCII

Intel Pentium numbers their bytes from right to left (little endian), Sun SPARC number them the other way (big endian).



(a) (b) (c)

a) Original message on the Pentium
b) The message after receipt on the SPARC
c) The message after being inverted. The little numbers in boxes indicate the address of each byte

Passing Reference Parameters

- How are pointers or in general references passed?
- With the greatest of difficulties if at all.
- Possible Solutions
  - Forbid pointers
  - Copy the object pointed to by the pointer into the message and send it to the server
  - The server copies it at some place in its memory space, and calls the routine passing a pointer to it.
  - When the routine ends, the server copies back the object's value into one parameter of the message and sends the message to the client.
  - If the stubs know whether the object is an input or output parameter to the server, one of the two copies can be avoided.
  - The above approach does not work with complex objects (i.e, dynamic arbitrary data structures).

Parameter Specification and Stub Generation

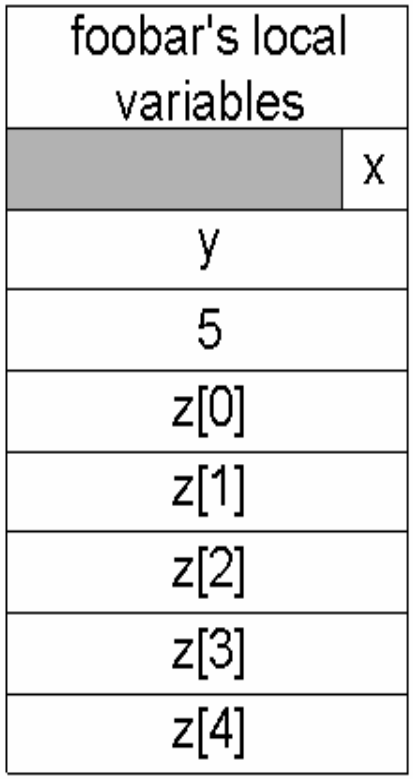
a) A procedure

b) The corresponding message:

- A character is placed in the rightmost byte of a word
- A float is transmitted as a whole word
- An array as a group of words equal to the array length, preceded by a word giving the length.

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

(a)



(b)

Parameter Specification and Stub Generation

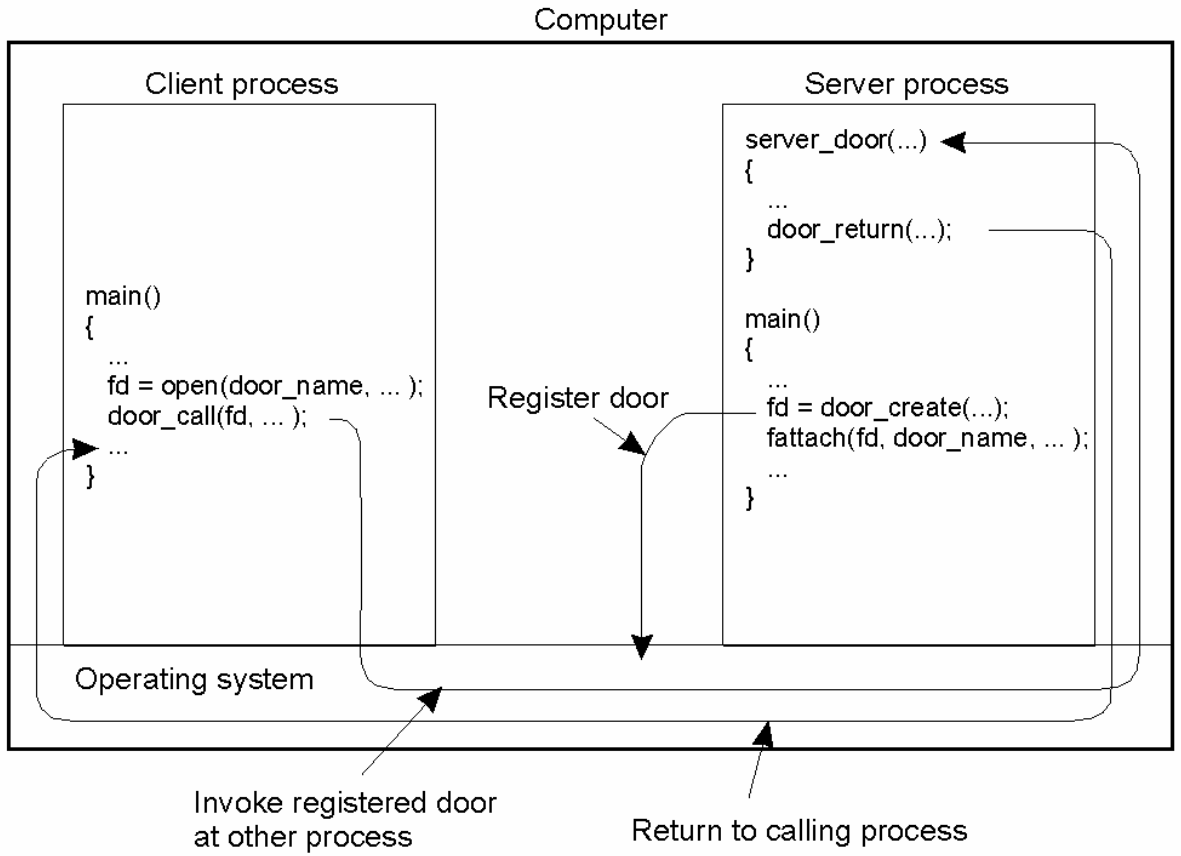
The caller and the callee agree on:

- The format of the messages
- The representation of simple data structures
  - integers are represented in two's complement, characters in 16-bit Unicode, floats in IEEE standard #754 format, etc.
- Stubs for the same RPC protocol but different procedures generally differ only in their interface to the applications.
- An interface consists of a collection of procedures that can be called by a client, and which are implemented by a server.
- Interfaces are often specified by means of an Interface Definition Language (IDL),
  - then compiled into a client stub and a server stub, along with the appropriate compile-time or run-time interfaces.

Doors

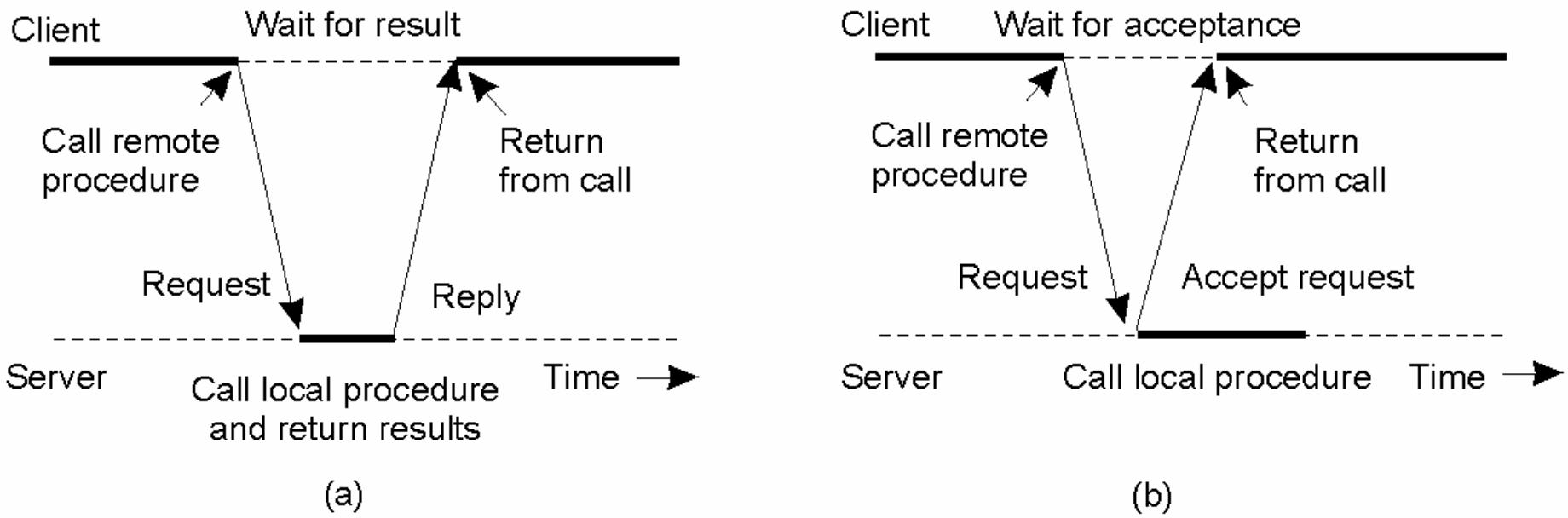
- A door is a generic name for a procedure in the address space of a server process that can be called by processes co-located with the server.

•**Benefit** : they allow the use of a single mechanism (namely procedure calls) for communication in a distributed system.



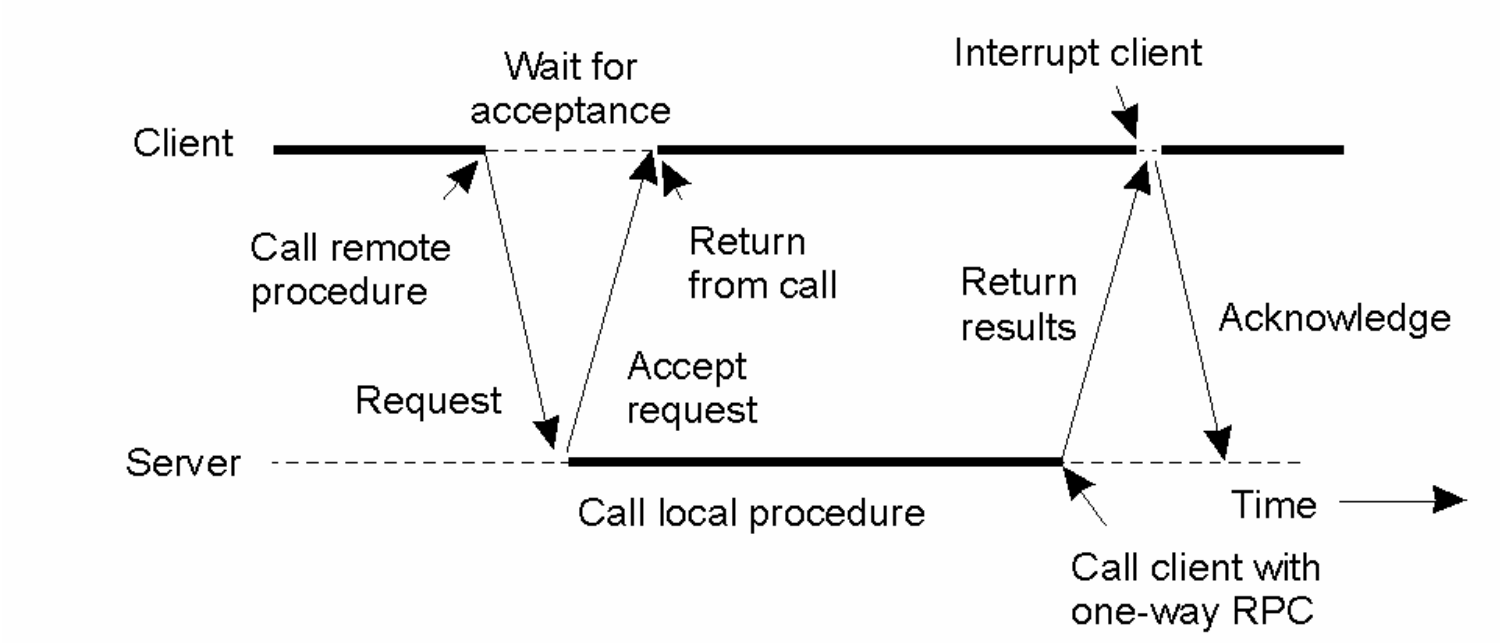
The principle of using doors as the Inter Process Communication (IPC) mechanism.

Asynchronous RPC (1)



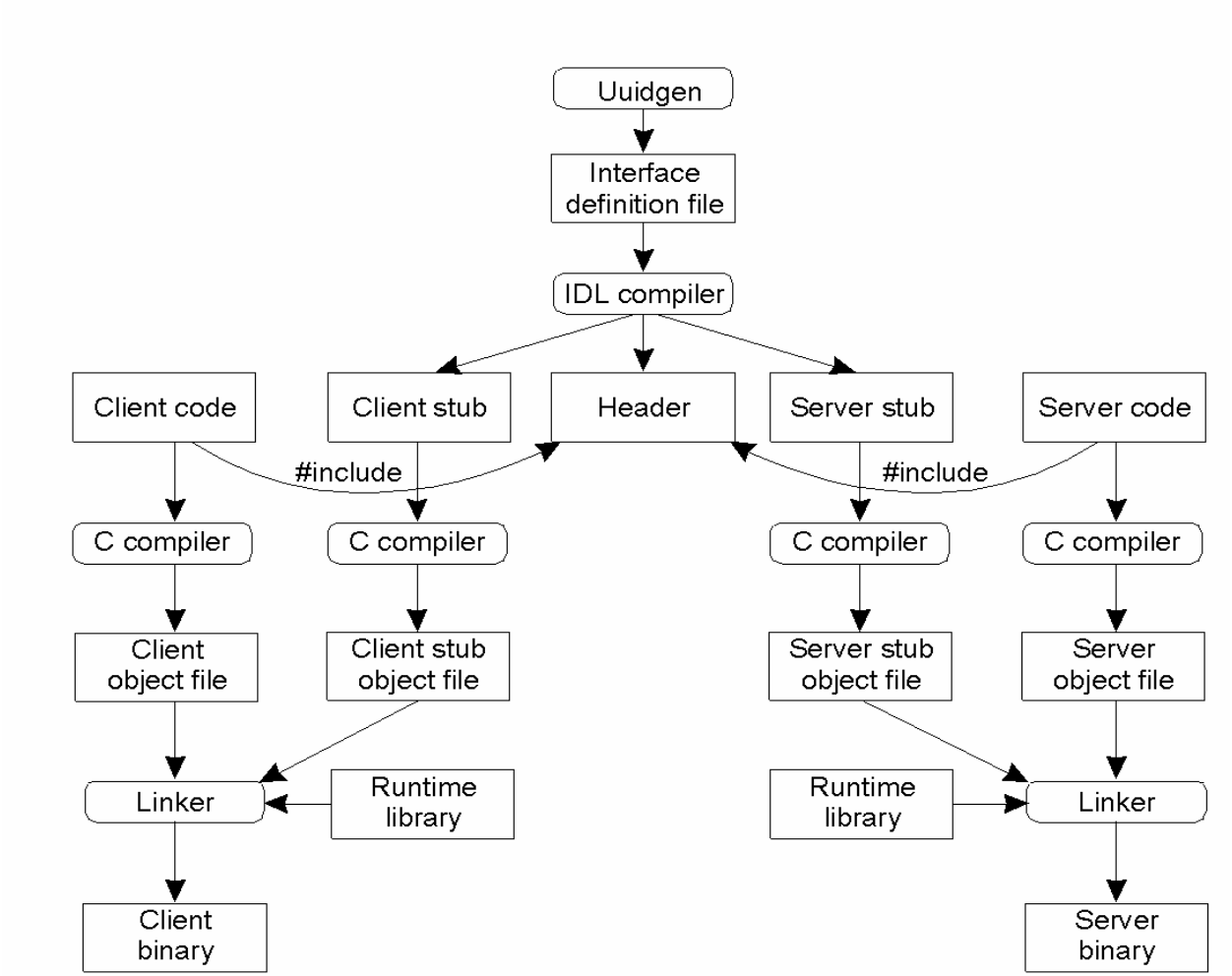
a) The interconnection between client and server in a traditional RPC
b) The interaction using asynchronous RPC

Asynchronous RPC (2)



- A client and server interacting through two asynchronous RPCs
- Combining two asynchronous RPCs is referred as a deferred synchronous RPC.

Writing a Client and a Server



- The steps in writing a client and a server in DCE RPC.

Binding a Client to a Server