Distributed Systems

Inter-Process Communication

Dynamic and Distributed
Information Systems

Today's Agenda

- Communication layers
  - The ISO-OSI layers
  - Connection-oriented vs. Connectionless
- Working with Sockets
  - System calls for TCP/IP
- Remote Procedure Calls

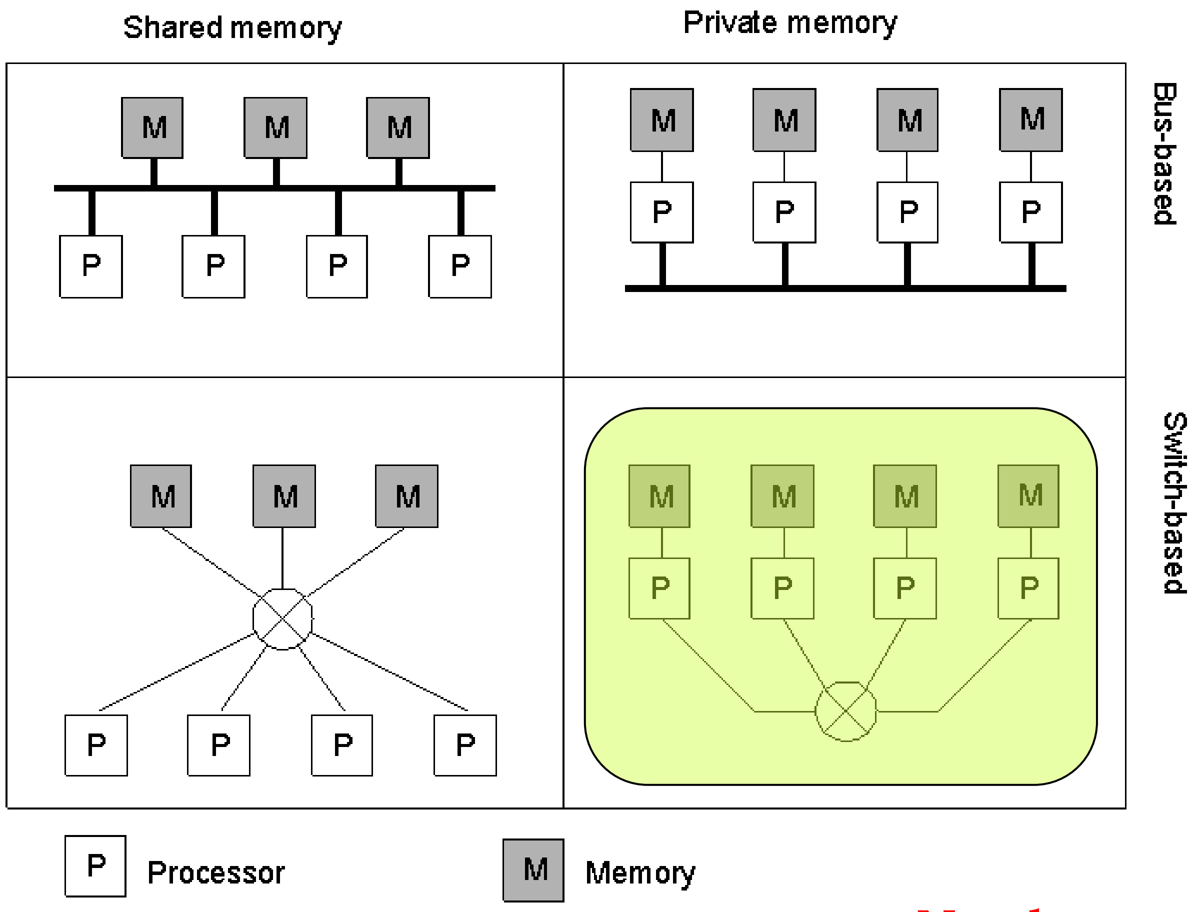
Communication layers

Communication

- Communication is at the heart of distributed systems
- Processes on different computers need to exchange information
- Information is exchanged over the network:
  - Transferred by means of primitive electrical or optical pulses
  - Unreliable
  - Complex
  - Possibly 1000's or millions of processes
- Processes need a simplified abstraction
  - concentrate on what data to exchange and with whom
  - ignore how that data is transferred

Hardware Concepts

Different basic organizations and memories in distributed computer systems



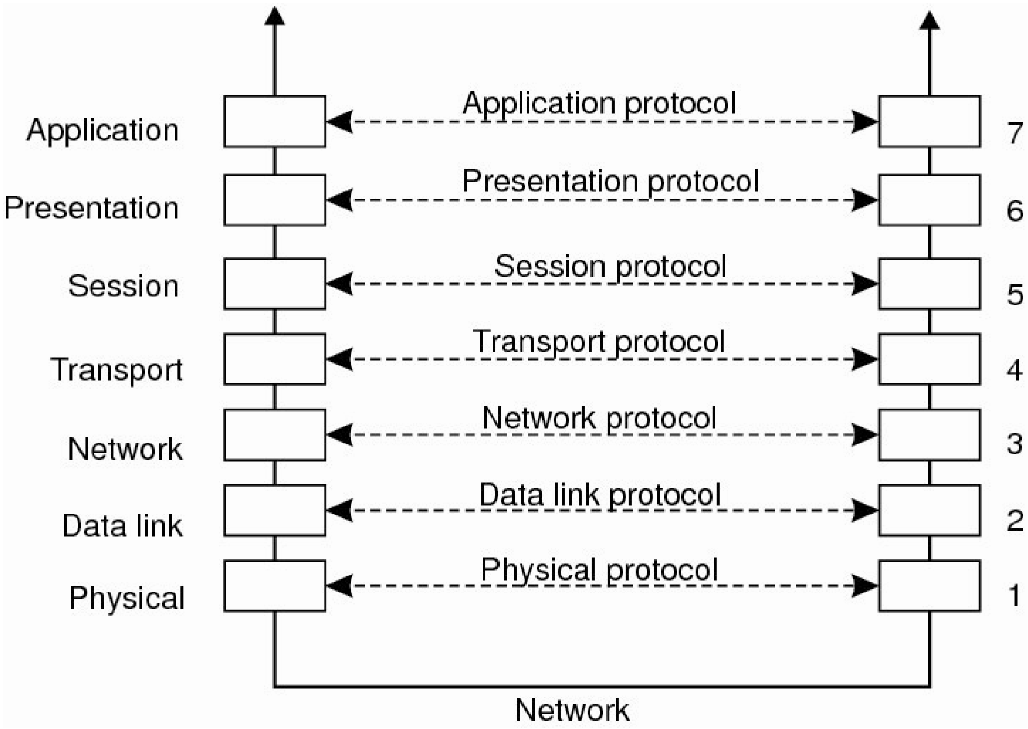
Non-homogeneous multicomputers

Communication

- Communication takes place by exchanging messages
- A process creates a message (some bytes) in user space, and invokes a system call to send it to a destination process on another computer
- Sounds simple, but:
  - What voltage levels represent 0 and 1?
  - How does the receiver know that the message has ended?
  - How does the sender know that the msg was received correctly?
  - What if a msg is lost or damaged?
  - How long are numbers? Strings? Data structures?
  - …
- Many agreements are needed
  - At many different levels!

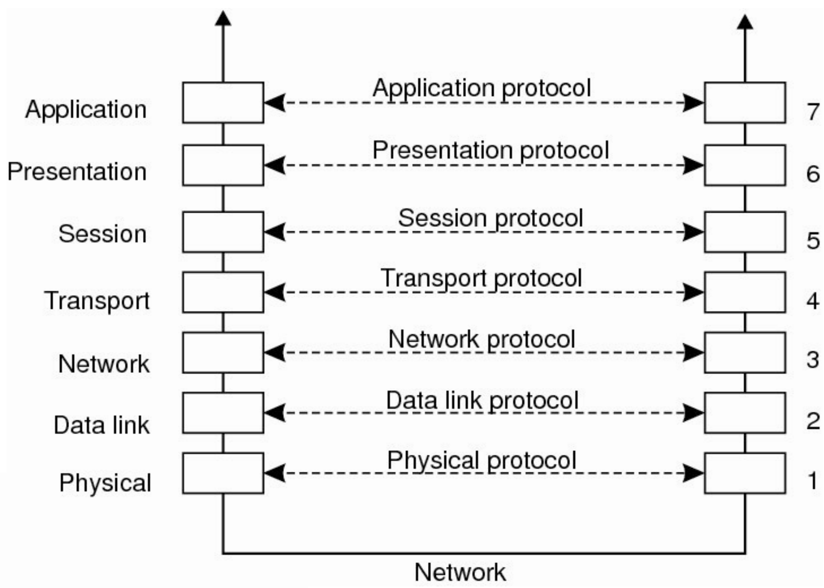
The seven ISO-OSI layers

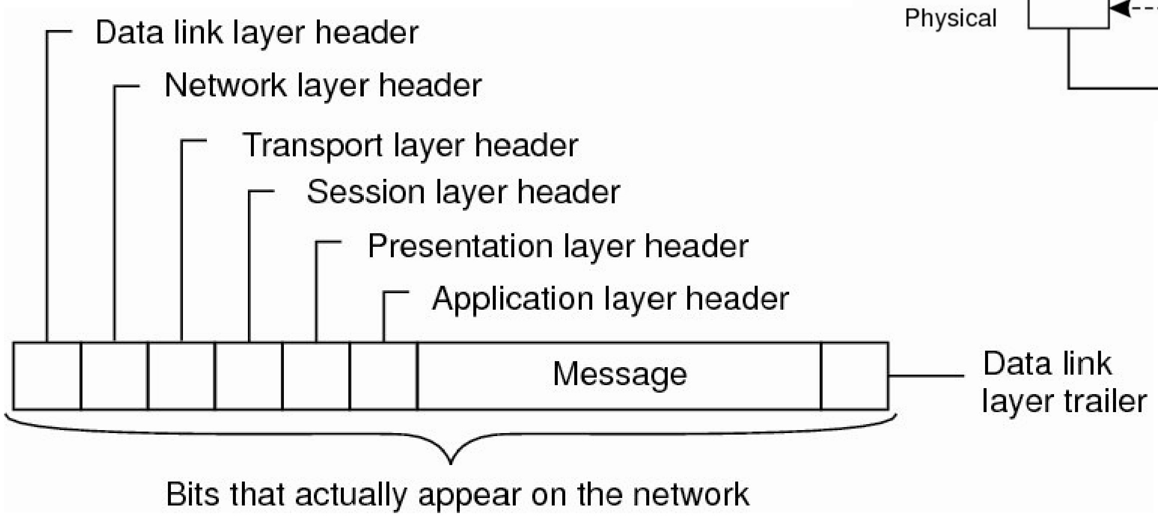
- ISO: **I**nternational **S**tandards **O**rganization
- OSI: **O**pen **S**ystems **I**nterconnection



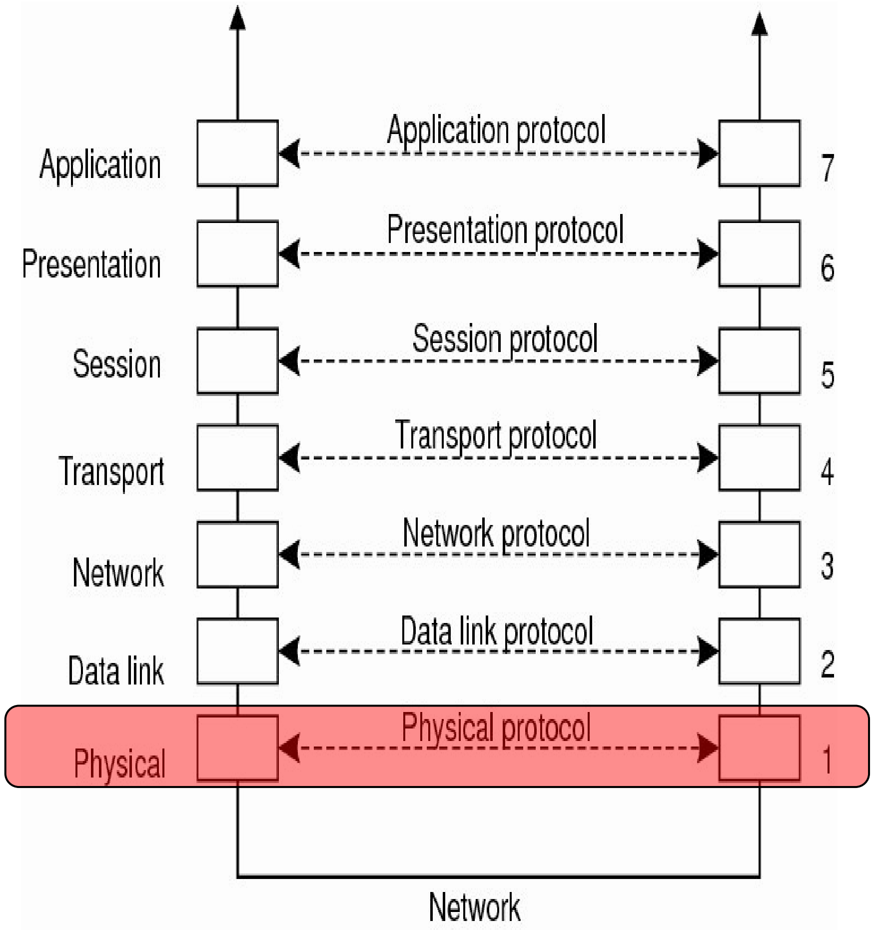
- Each layer
  - deals with a specific aspect of communication
  - provides an abstraction to the layer right above it

The seven ISO-OSI layers



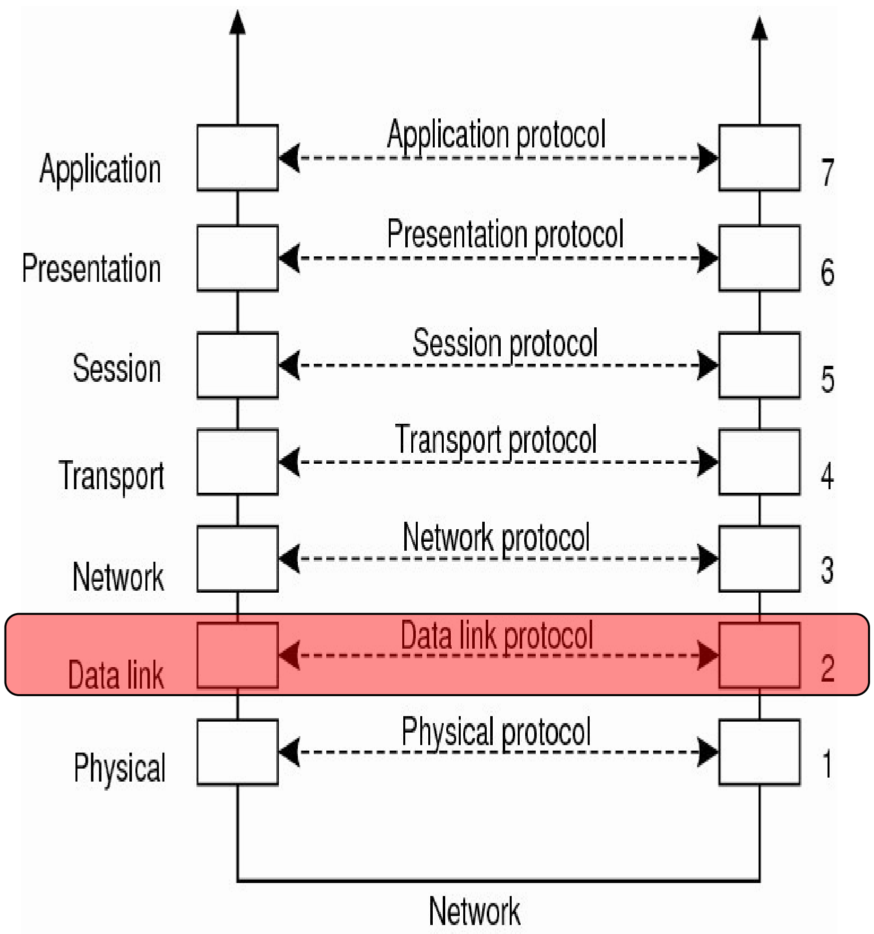


Layer 1: Physical



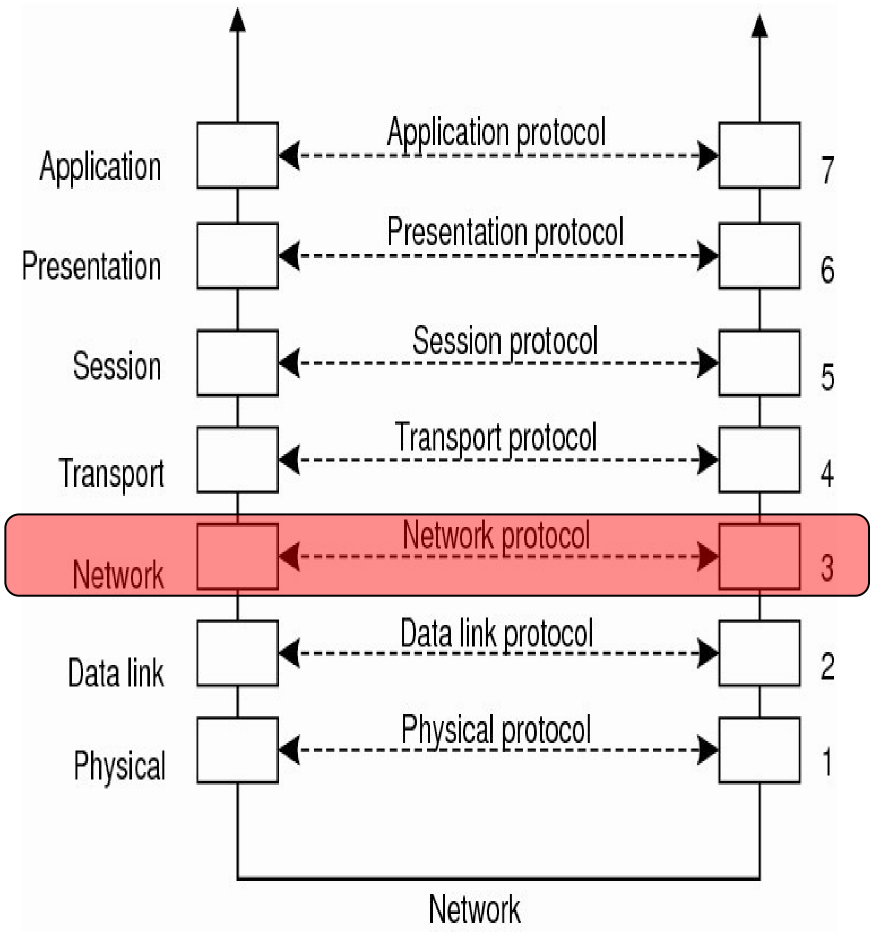
- How many volts represent 0 and 1
- Bits per second
- Half-duplex / Full-duplex
- Electrical / mechanical / optical signaling interfaces

Layer 2: Data Link



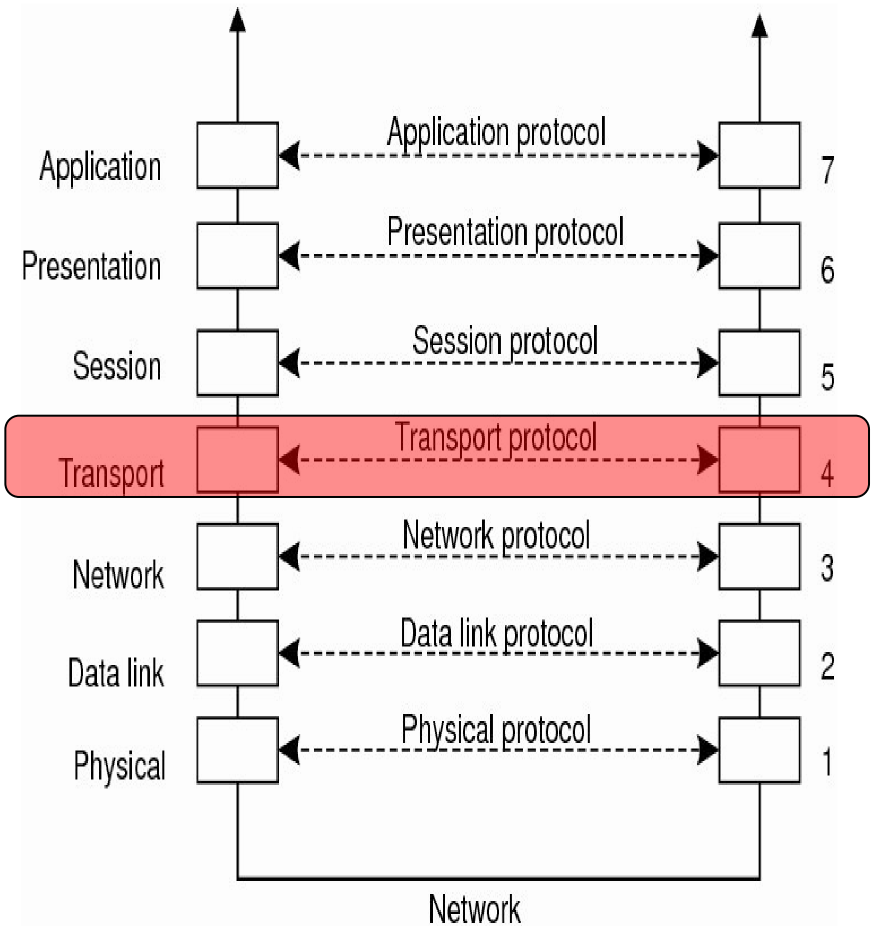
- Groups bits into frames
- In each frames adds
  - Starting and ending bit patterns
  - A sequence number
  - A checksum for error detection

Layer 3: **Network**



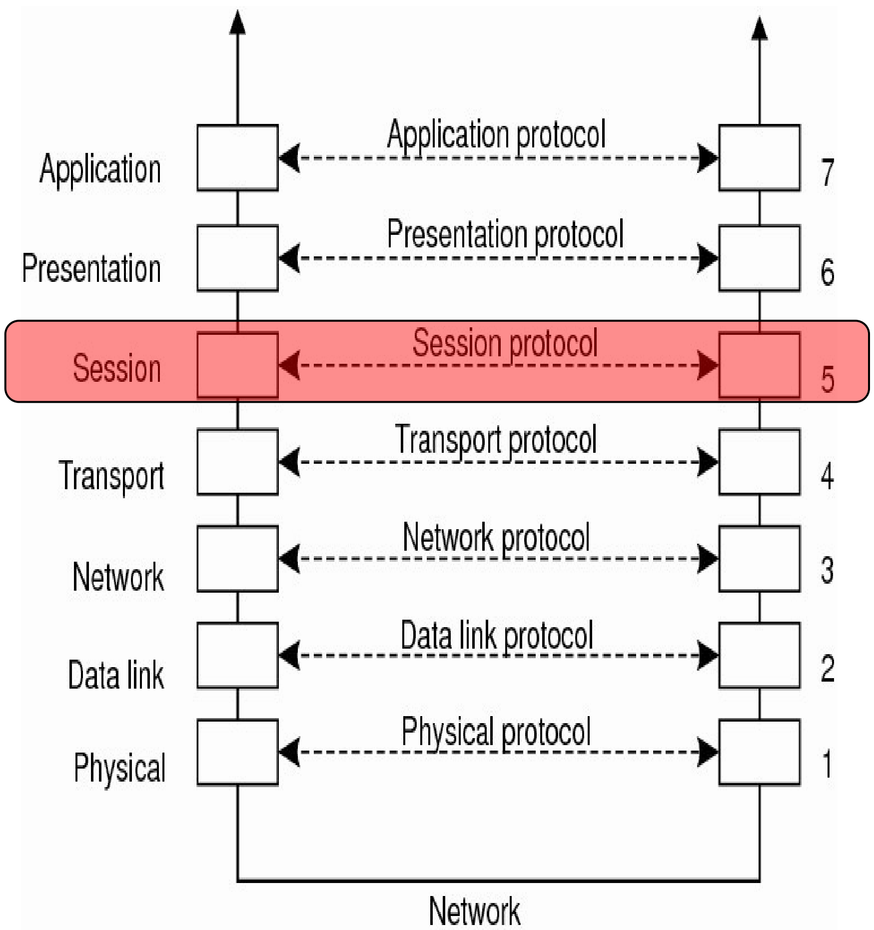
- Routes packets towards their destination
- Most common protocol: **IP (Internet Protocol)**
- Goal:
  - Find the shortest path to the destination
  - …or the optimal path (based on traffic conditions and link speeds)

Layer 4: Transport



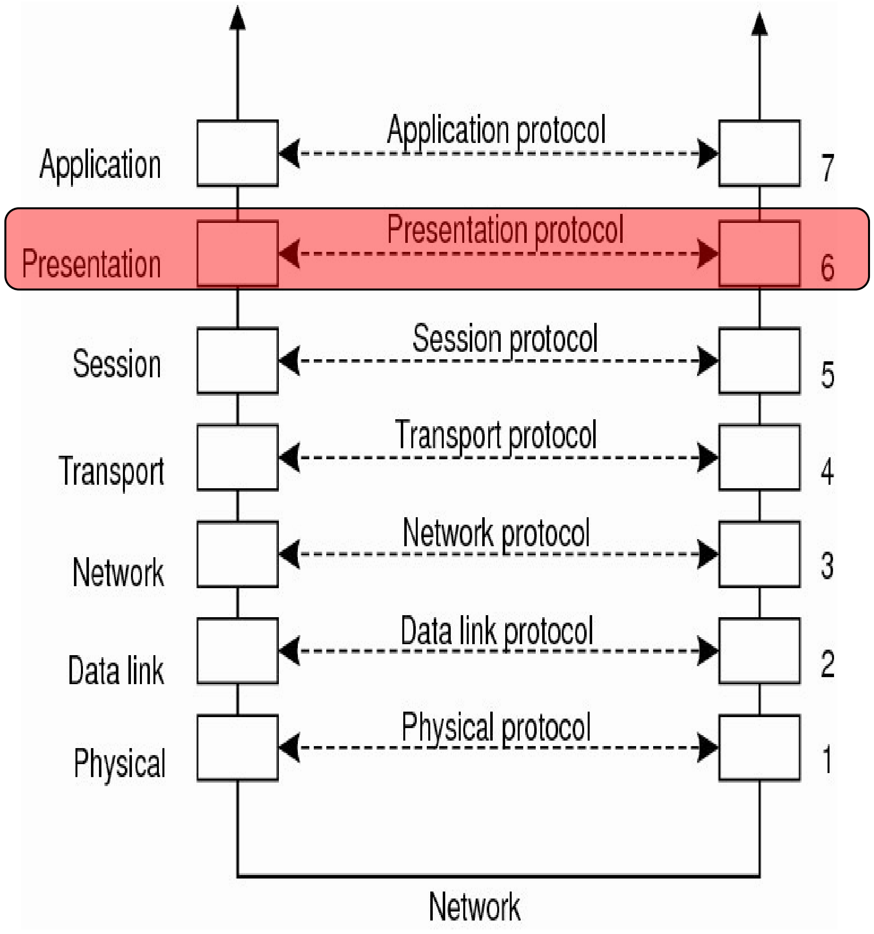
- Provide end-to-end functionality
- Most known protocols
  - **TCP (Transmission Control Protocol)**
  - **UDP (User Datagram Protocol)**
    - Often called **Unreliable Datagram Protocol**
- Message fragmentation
  - An application's msg is split in packets
- Reliable delivery
  - Normally packets may get lost or damaged
  - The transport layer arranges their retransmission *transparently*
- In-order delivery
  - Packets may follow different routes, and arriving in wrong order

Layer 5: **Session**



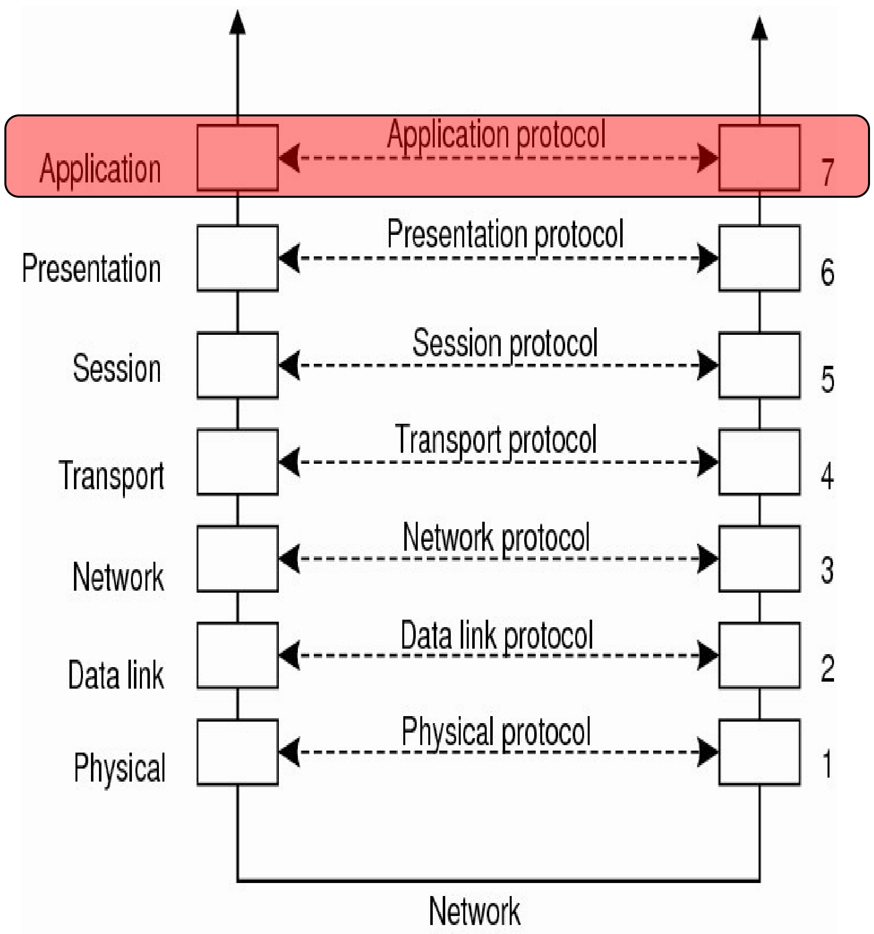
- Checkpoints in communcation
- Synchronization facility
- Not used in practice!
  - Not even implemented in the Internet protocol stack

Layer 6: **Presentation**



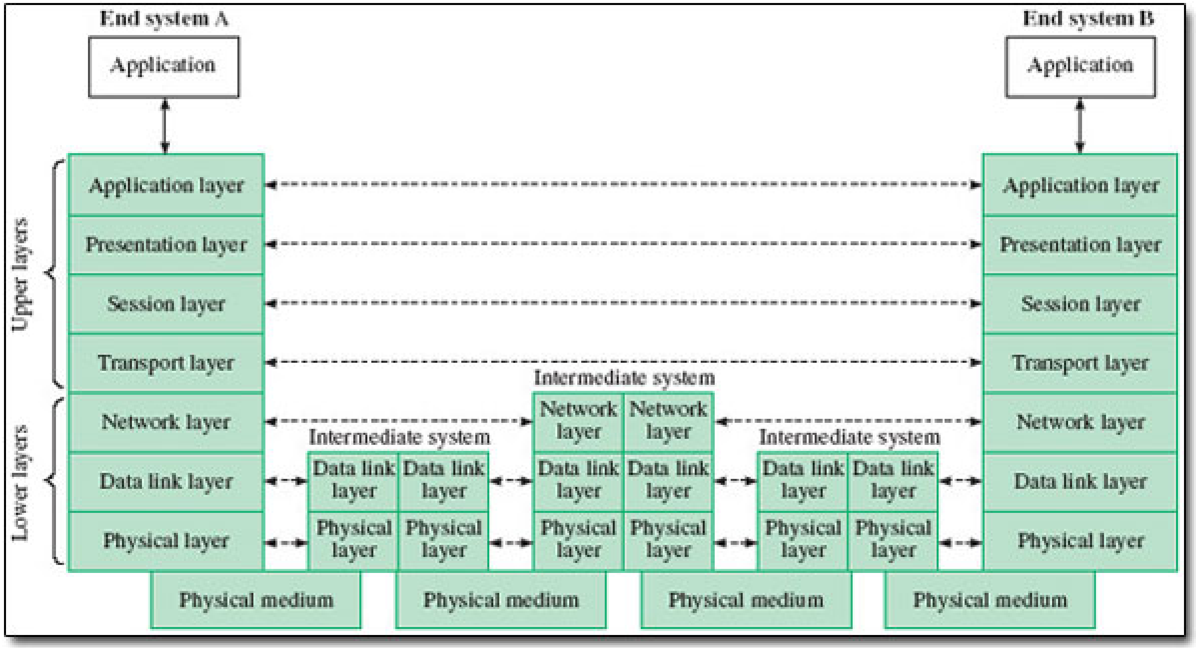
- Deals with the meaning of bits
- E.g., adjusts the representation of numbers

Layer 7: **Application**



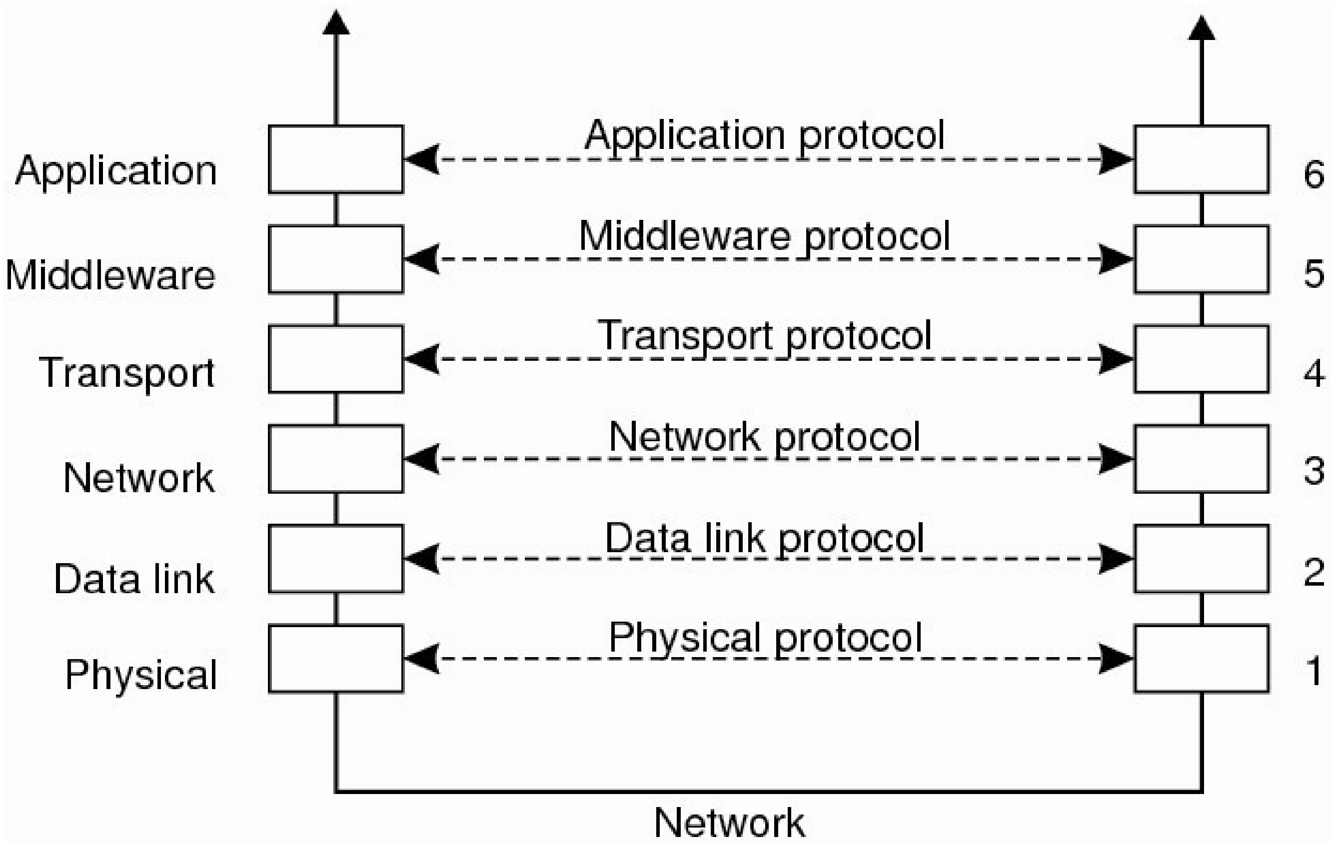
- Applications & Protocols
  - File Transfer Protocl (FTP)
  - HyperText Transfer Protocol (HTTP)
  - Simple Mail Transfer Protocol (SMTP)
  - Telnet Protocol
  - Secure Shell (SSH)
  - …
- All distributed systems are here!

Layer Interaction



- Lower layers (1-3): interaction between consecutive nodes in the Internet infrastructure (bridges, routers)
- Upper layers (4-7): end-to-end interaction

Layers in practice



- Note the Middleware layer

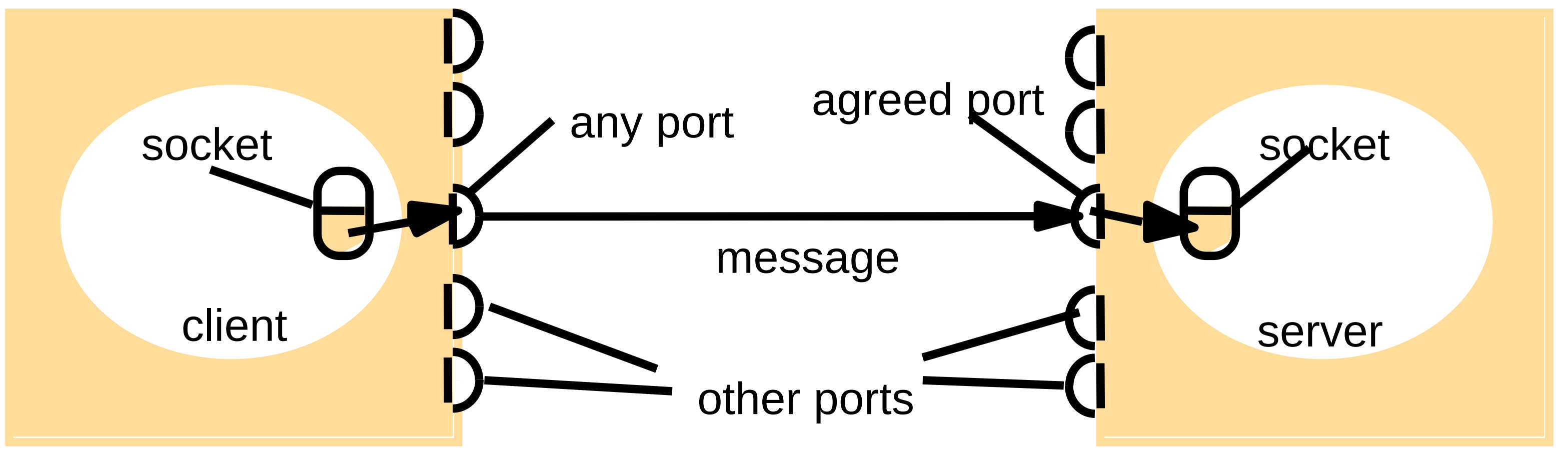
Type of Connections

- Connection-oriented
  - Before communication, sender & receiver negotiate what protocols and parameters will be used
  - When done, terminate the connection
  - The sender sends a stream of bytes, that transparently get grouped in packets and delivered to the receiver.
  - Analogous to making a phone call
- Connectionless
  - No setup
  - No termination
  - The sender explicitly sends individual packets to the receiver
  - Analogous to sending letters by post

Working with Sockets

Dynamic and Distributed
Information Systems

Sockets and ports



Internet address = 138.37.94.248

Internet address = 138.37.88.249

Figure 4.2

UDP vs. TCP message

UDP (connectionless)

- Send and forget
- Size <2^{16} bytes (16K)
- Blocking
  - Send – none
  - Receive – yes (timeout)
- Failure model: message may be
  - Lost
  - Received out of order

TCP (connection-oriented)

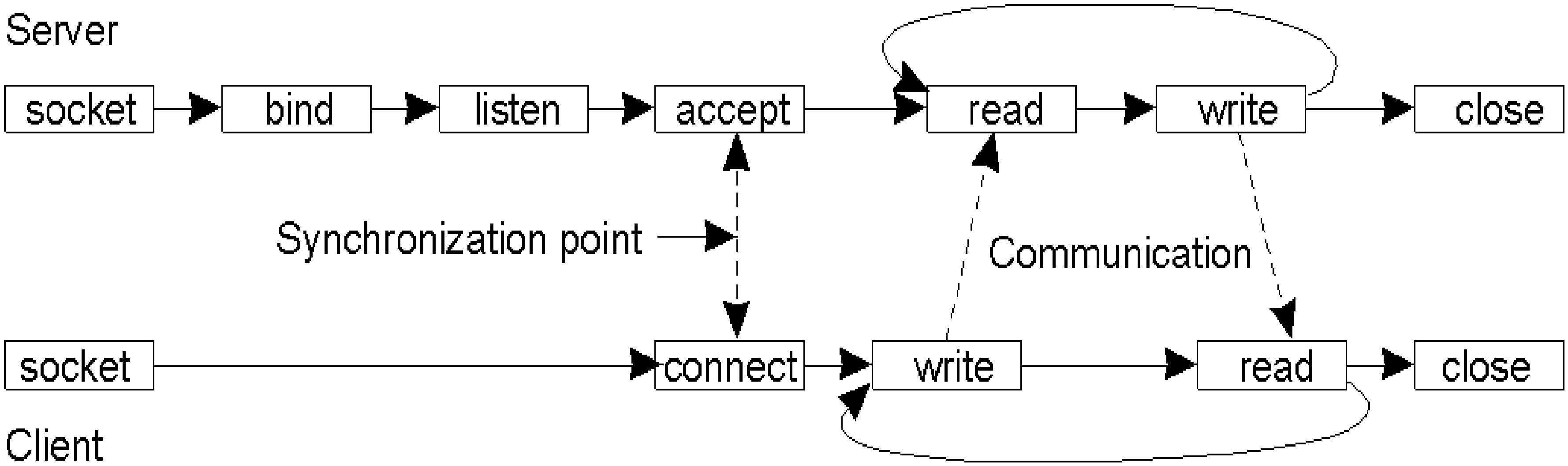
- Stream of bytes
- Size: repeated sends
- Blocking
  - Send – none
  - Receive – choice
- Failure model:
  Reliable comm.
  - Acknowledgement
  - Flow control (ordering)
  - checksums

System calls for TCP/IP sockets

System call	*Who calls it*	*Meaning*
Socket	Server / Client	Create a new communication endpoint
Bind	Server	Attach a local address and port to a socket
Listen	Server	Define how many clients can be queued
Accept	Server	Block until a connection request arrives
Connect	Client	Actively attempt to establish a connection
Write	Server / Client	Send some data over the connection
Read	Server / Client	Receive some data over the connection
Close	Server / Client	Release the connection

TCP/IP communication

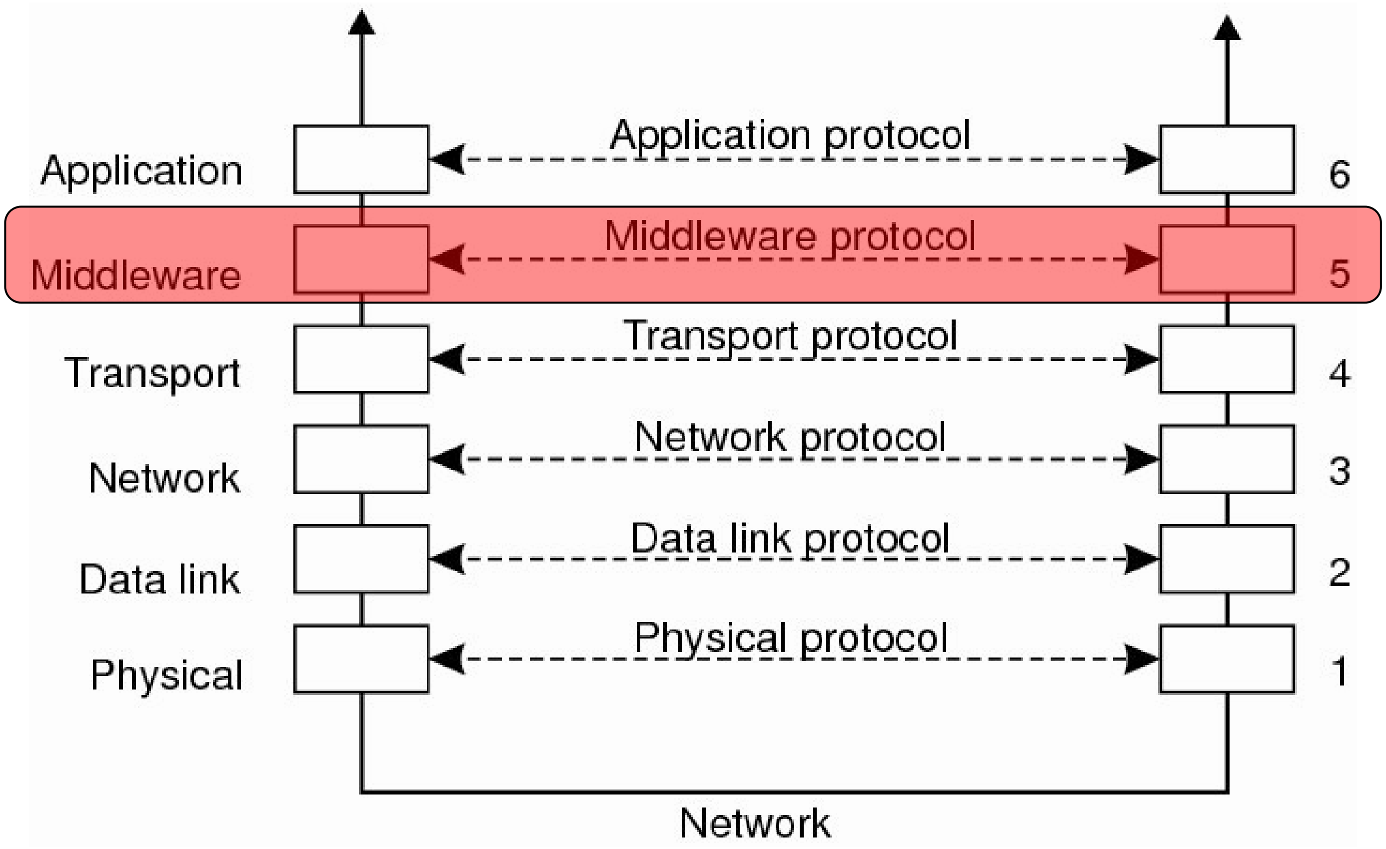
- Connection-oriented



Remote Procedure Calls

Dynamic and Distributed Information Systems

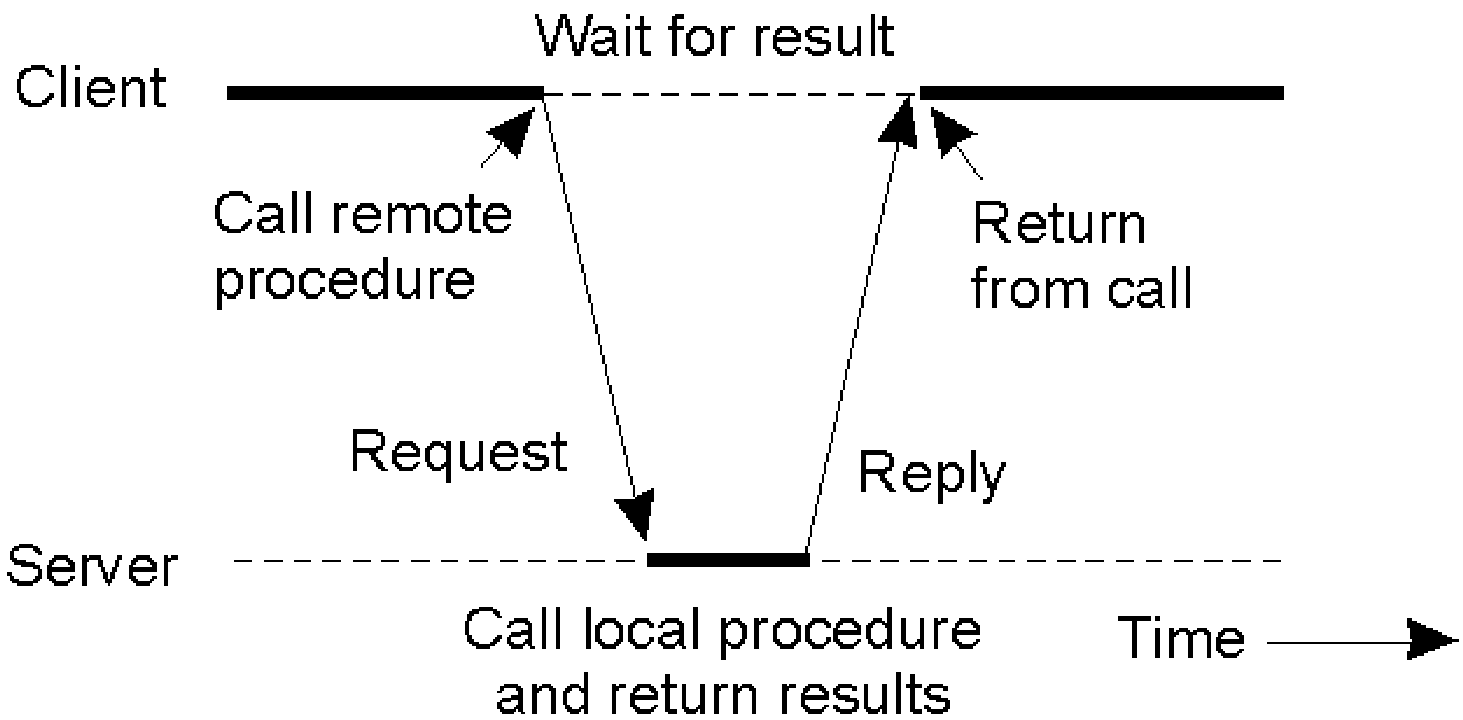
Context



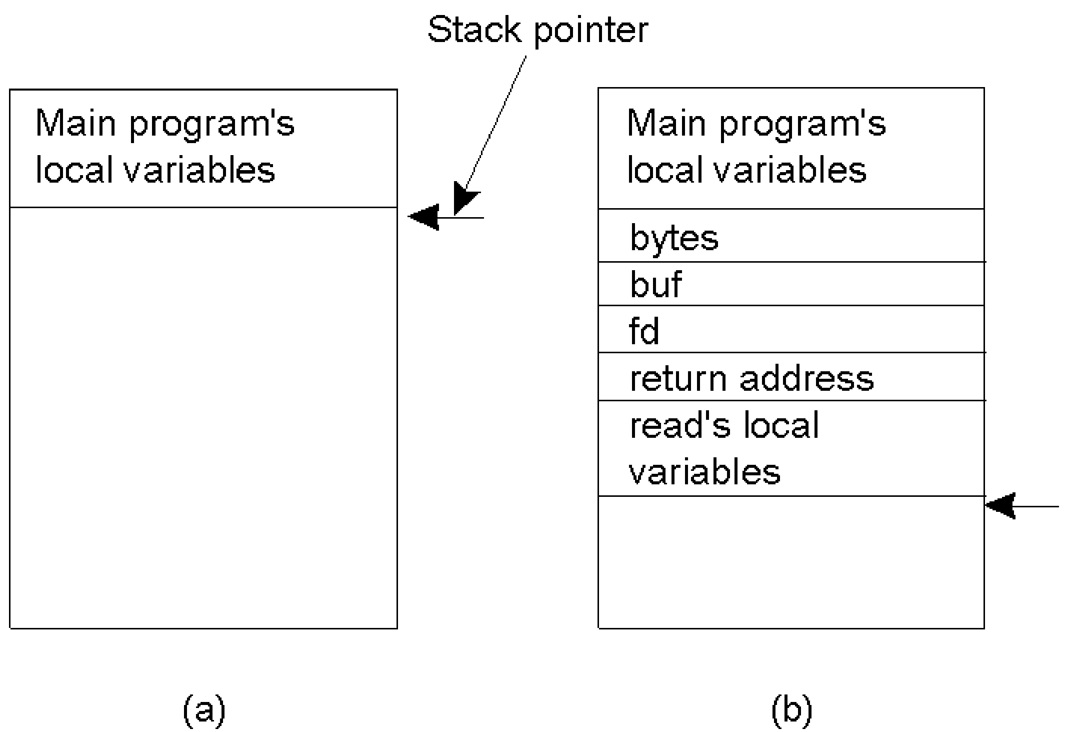
- Remote Procedure Calls constitute a Middleware-layer functionality

Remote Procedure Call

- Principle of RPC between a client and server program.
- Request-reply communication



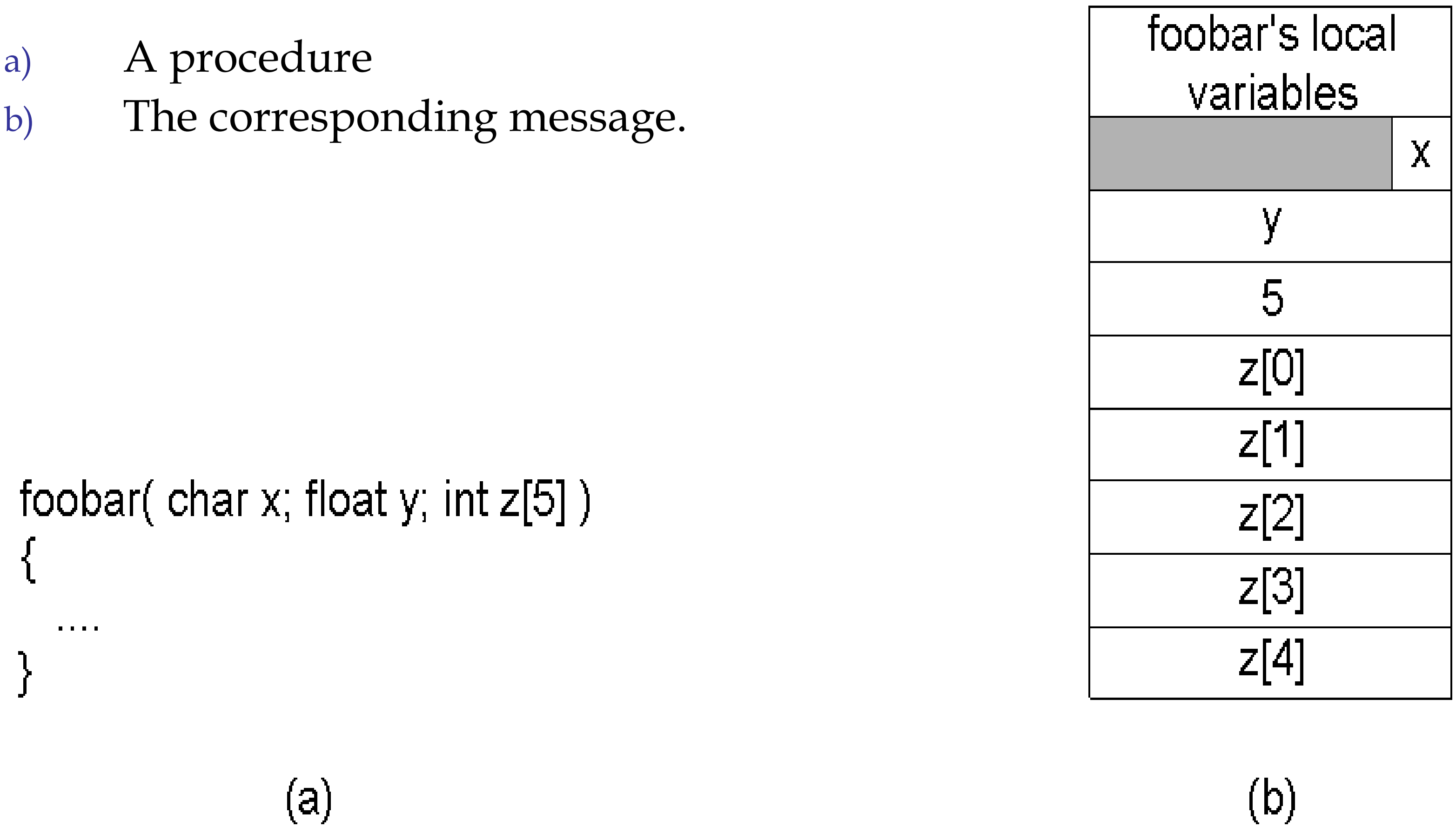
Conventional Procedure Call



- Parameter passing in a local procedure call: the stack before the call to read
- The stack while the called procedure is active

Stub Generation

a) A procedure
b) The corresponding message.

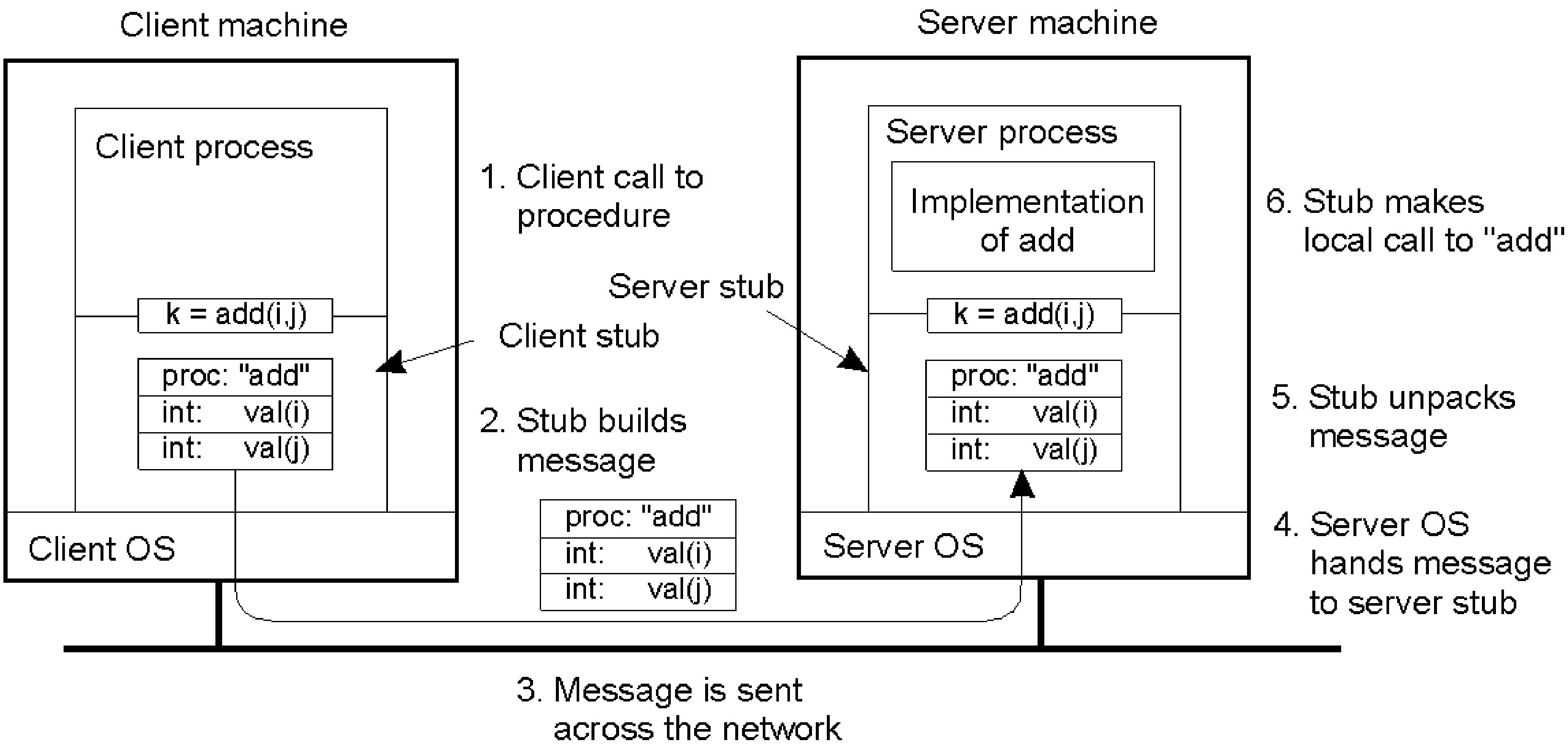


Steps of a Remote Procedure Call

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

Steps of a Remote Procedure Call

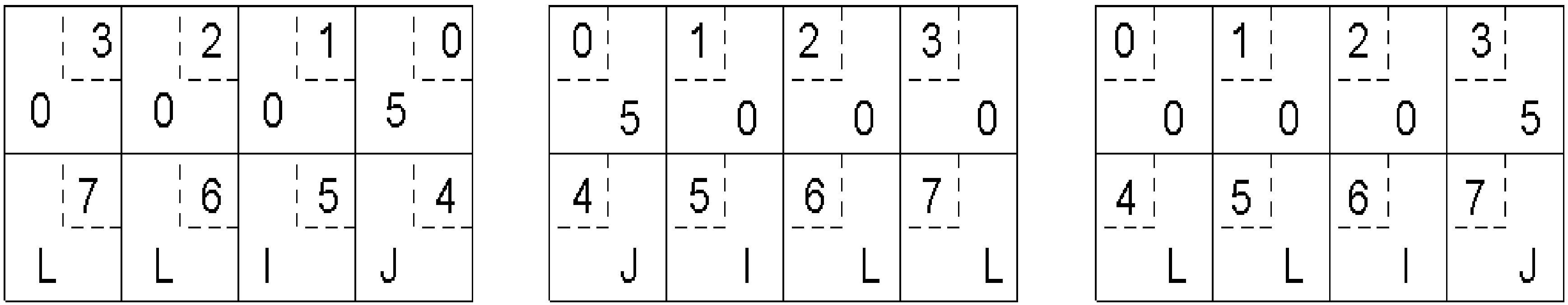
- Steps involved in doing remote computation through RPC



Problems with RPC

- Data representation
  - Different encodings
  - ASCII vs. EBCDIC (IBM)
  - Unicode vs. other encodings
  - little endian vs. big endian
- Passing arguments
  - "Pass-by-value" is ok
  - "Pass-by-reference" is problematic
  - "Pass-by-copy/restore"

Marshalling the values



(a) (b) (c)

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

Passing arguments

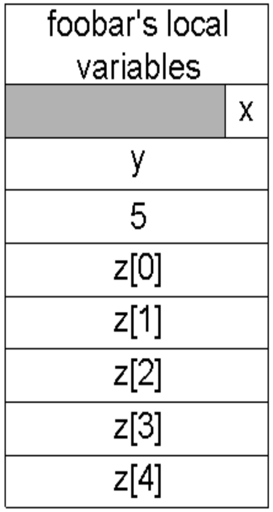
- Arguments passed by value (int, float, boolean, char, etc.) are simply passed by value.

- Arguments passed by reference (pointer to buffer, to string, etc.) are passed by copy/restore.

- What happens with more intricate structures? (e.g., graphs, trees, linked lists)

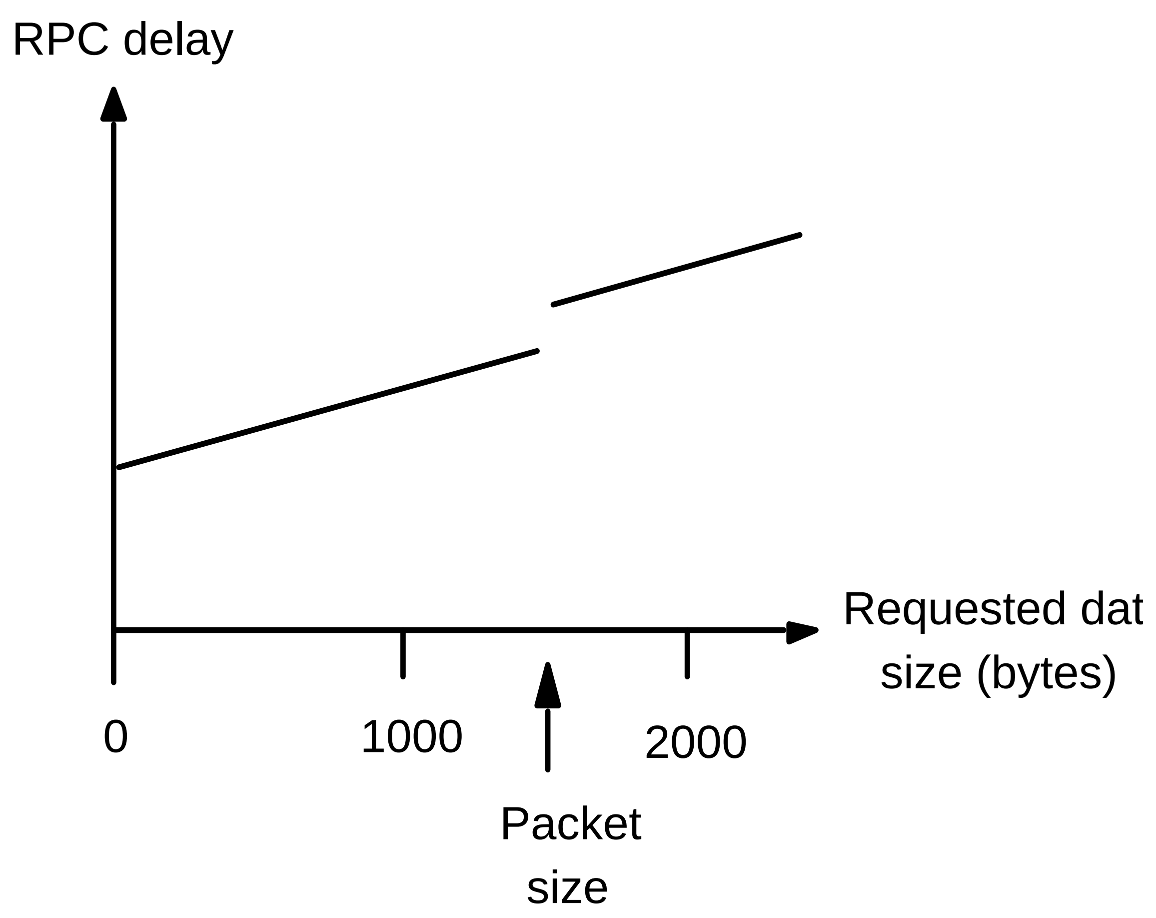
```
foobar( char x; float y; int z[5] )
{
  ....
}
```

(a)

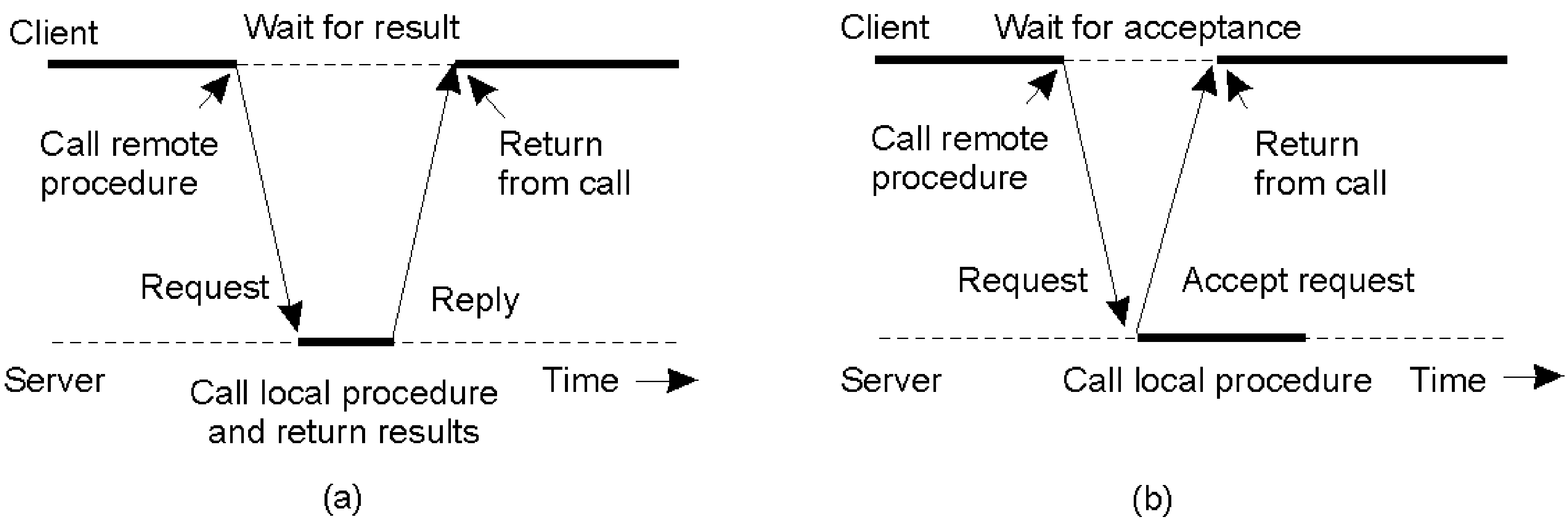


(b)

RPC delay against parameter size



Asynchronous RPC (1)



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

Asynchronous RPC (2)