# CAT3053/N Distributed Computing

## Communication in Distributed Systems

# Objectives

- To identify different layers of communication from protocols to Application Programming Interfaces (APIs).
- To present the various ways and widely-used models of communication in distributed systems
- To specify different paradigms of communications
- To be able to program applications with distributed objects and Internet protocols

# References

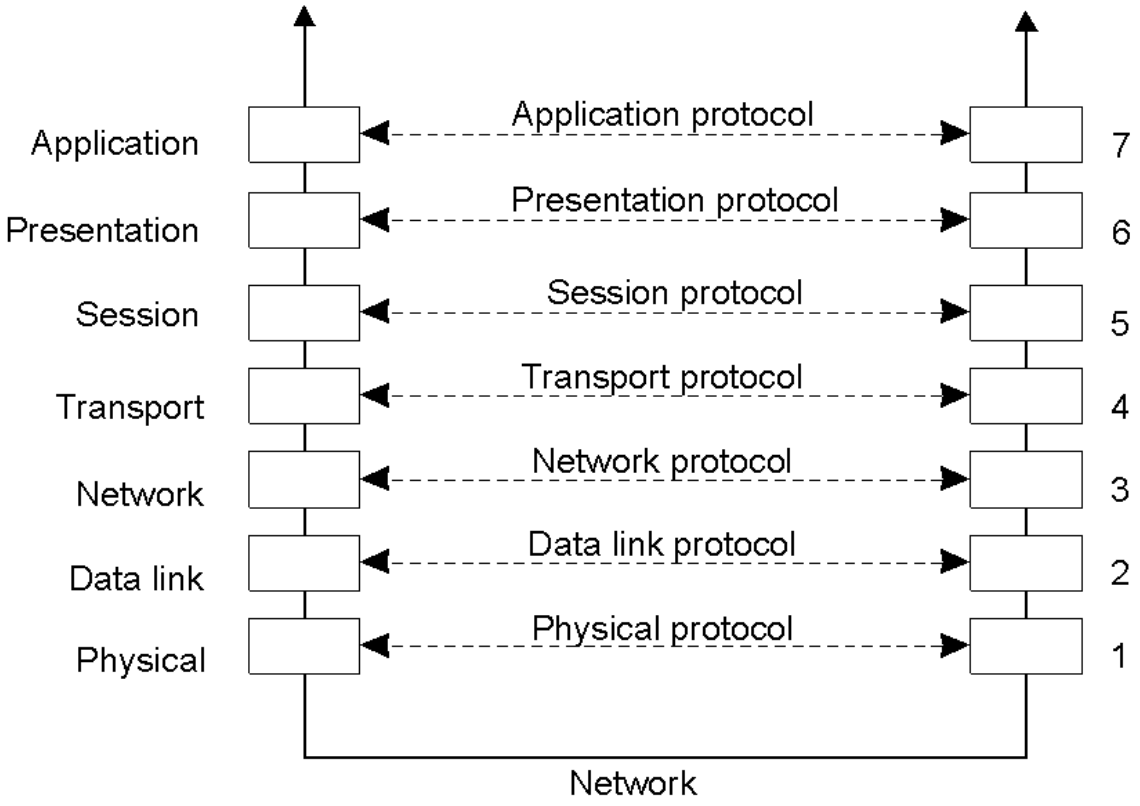

1. Distributed Systems: Principles and Paradigms by Tanenbaum (Chap 2, 3)
2. Distributed Systems: Concepts and Design by Coulouris (Chap 3, 4, 5, 6)
3. Distributed and Parallel Computing by El-Rewini and Lewis (10,11)

# Overview

- Protocols
- Models
  - Remote Procedure Call
  - Remote Method Invocation
  - Message Passing Interface Standard (MPI)
  - Streams
- Topology of communication
  - Client server
  - Peer to peer
  - Group

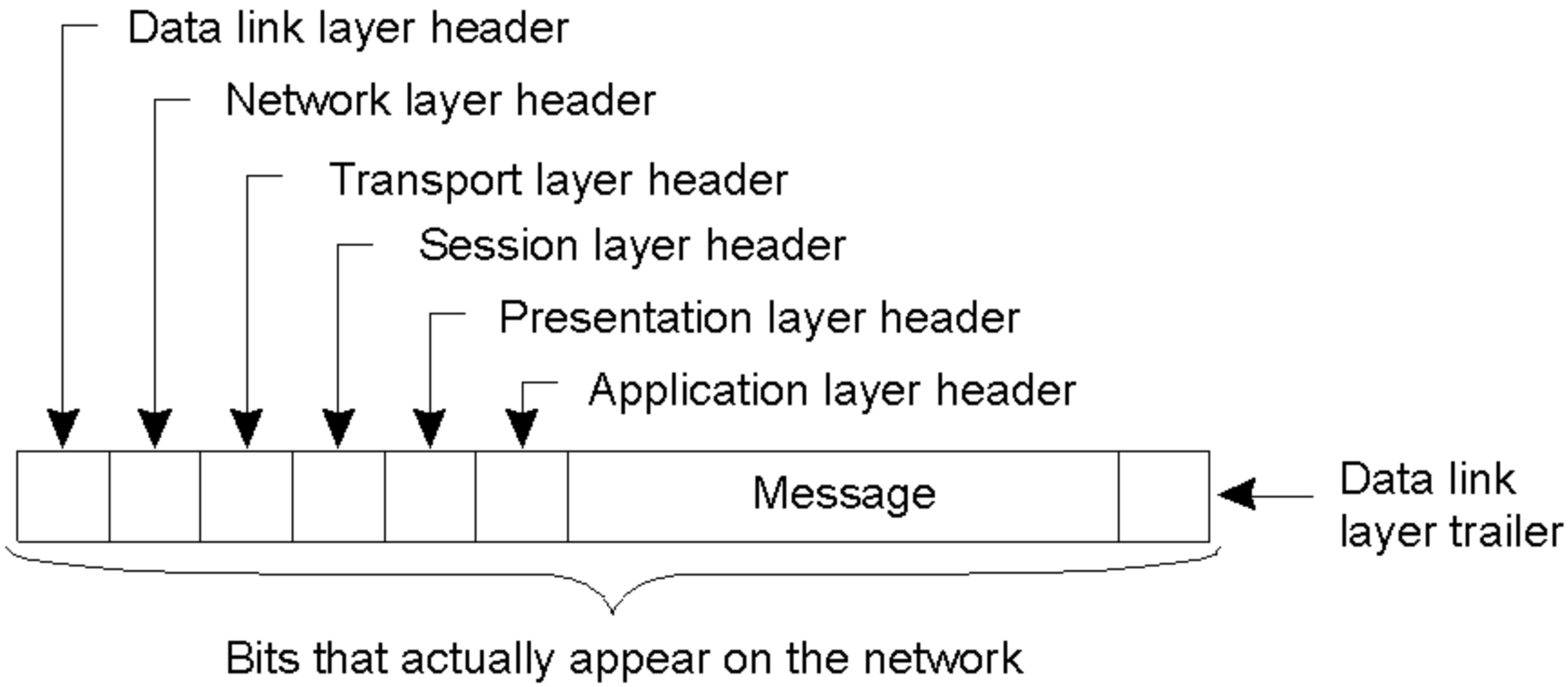
# Layered Protocols

Layers, interfaces, and protocols in the OSI model.



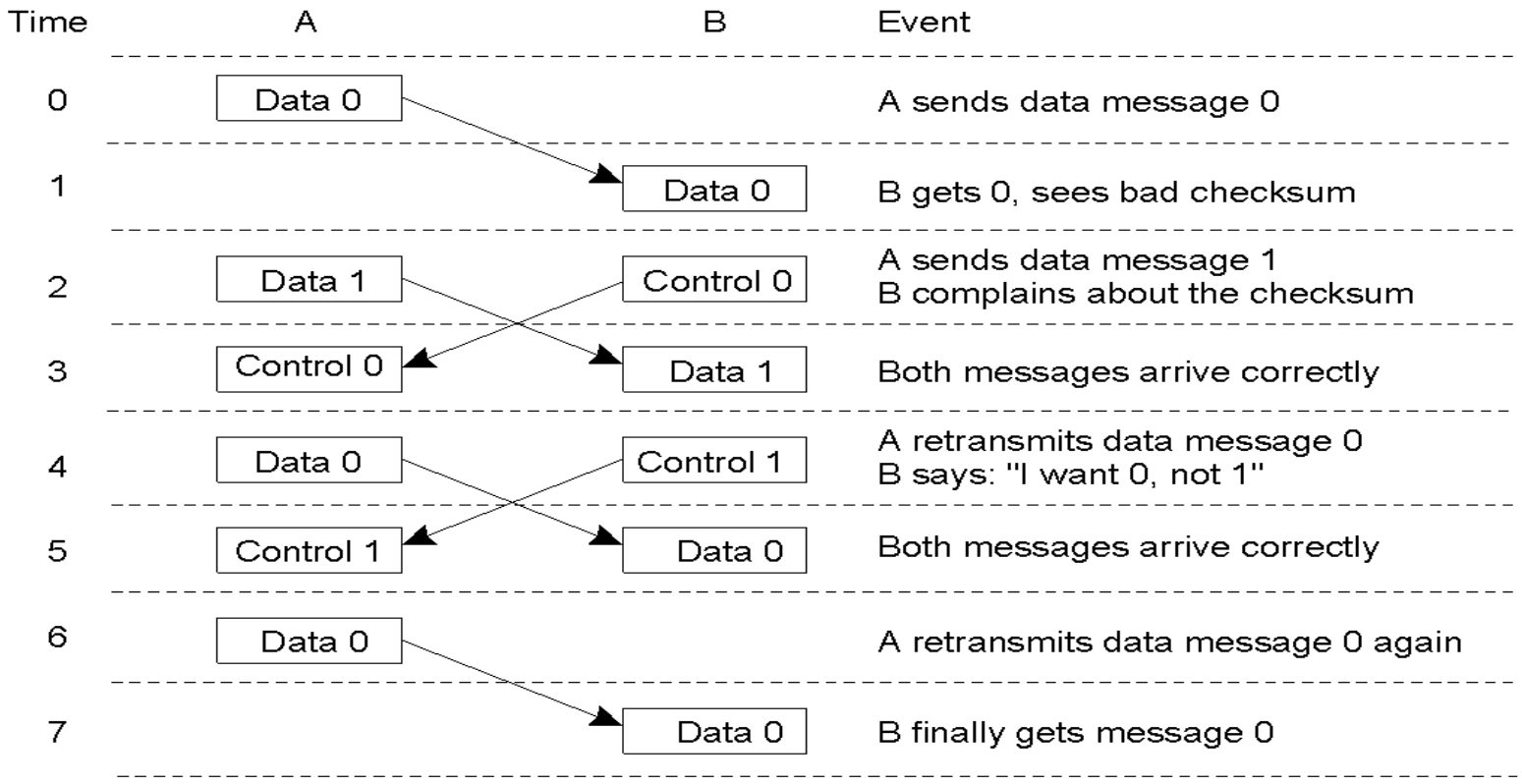
# Layered Protocols

A typical message as it appears on the network.



# Data Link Layer

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



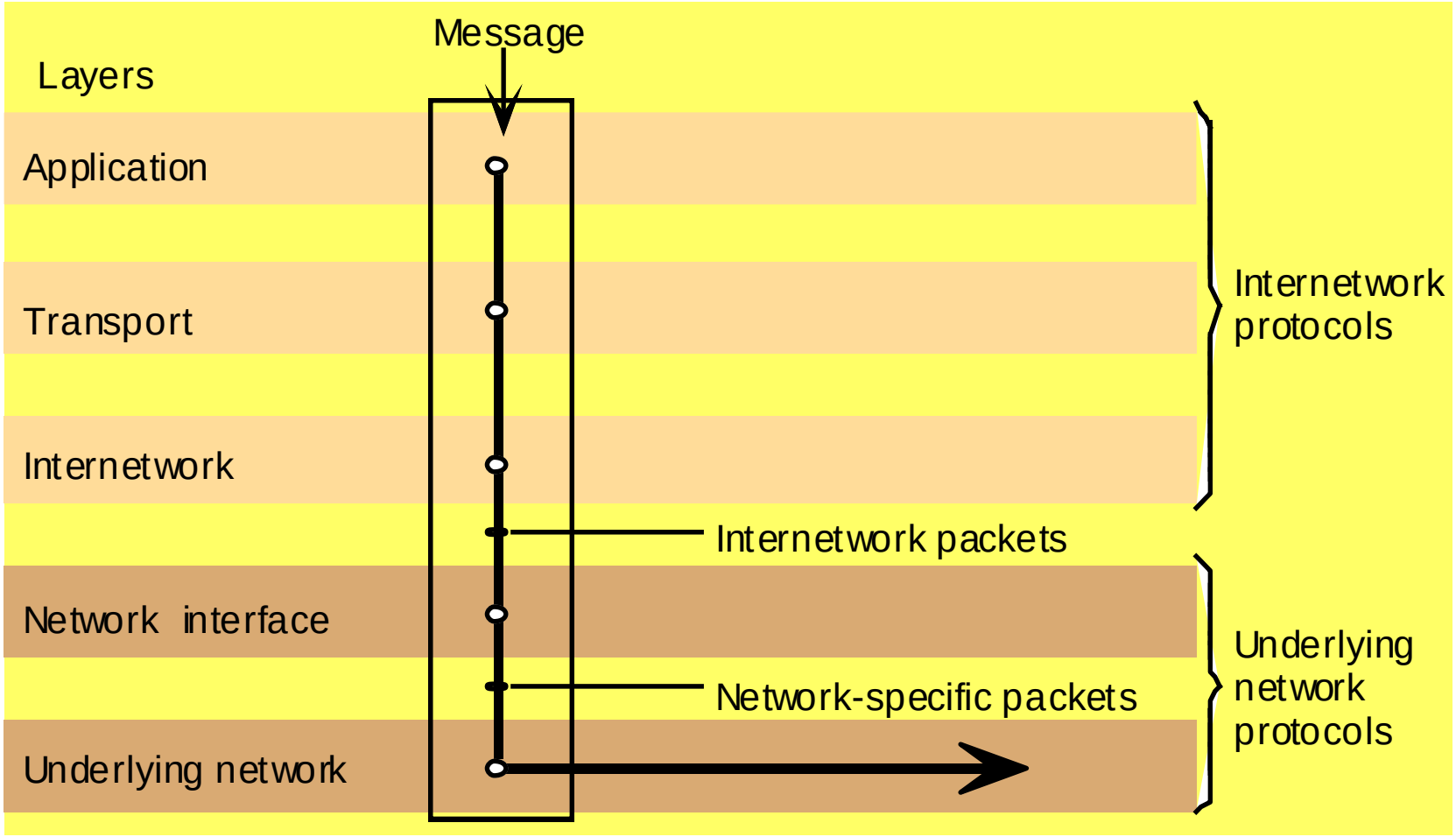
# OSI Protocol Summary

| *Layer* | *Description* | *Examples* |
|---|---|---|
| Application | Protocols that are designed to meet the communication requirements of specific applications, often defining the interface to a service. | HTTP, FTP , SMTP, CORBA IIOP |
| Presentation | Protocols at this level transmit data in a network representation that is independent of the representations used in individual computers, which may differ. Encryption is also performed in this layer, if required. | Secure Sockets (SSL),CORBA Data Rep. |
| Session | At this level reliability and adaptation are performed, such as detection of failures and automatic recovery. | |
| Transport | This is the lowest level at which messages (rather than packets) are handled. Messages are addressed to communication ports attached to processes, Protocols in this layer may be connection-oriented or connectionless. | TCP, UDP |
| Network | Transfers data packets between computers in a specific network. In a WAN or an internetwork this involves the generation of a route passing through routers. In a single LAN no routing is required. | IP, ATM virtual circuits |
| Data link | Responsible for transmission of packets between nodes that are directly connected by a physical link. In a WAN transmission is between pairs of routers or between routers and hosts. In a LAN it is between any pair of hosts. | Ethernet MAC, ATM cell transfer, PPP |
| Physical | The circuits and hardware that drive the network. It transmits sequences of binary data by analogue signalling, using amplitude or frequency modulation of electrical signals (on cable circuits), light signals (on fibre optic circuits) or other electromagnetic signals (on radio and microwave circuits). | Ethernet base- band signalling, ISDN |

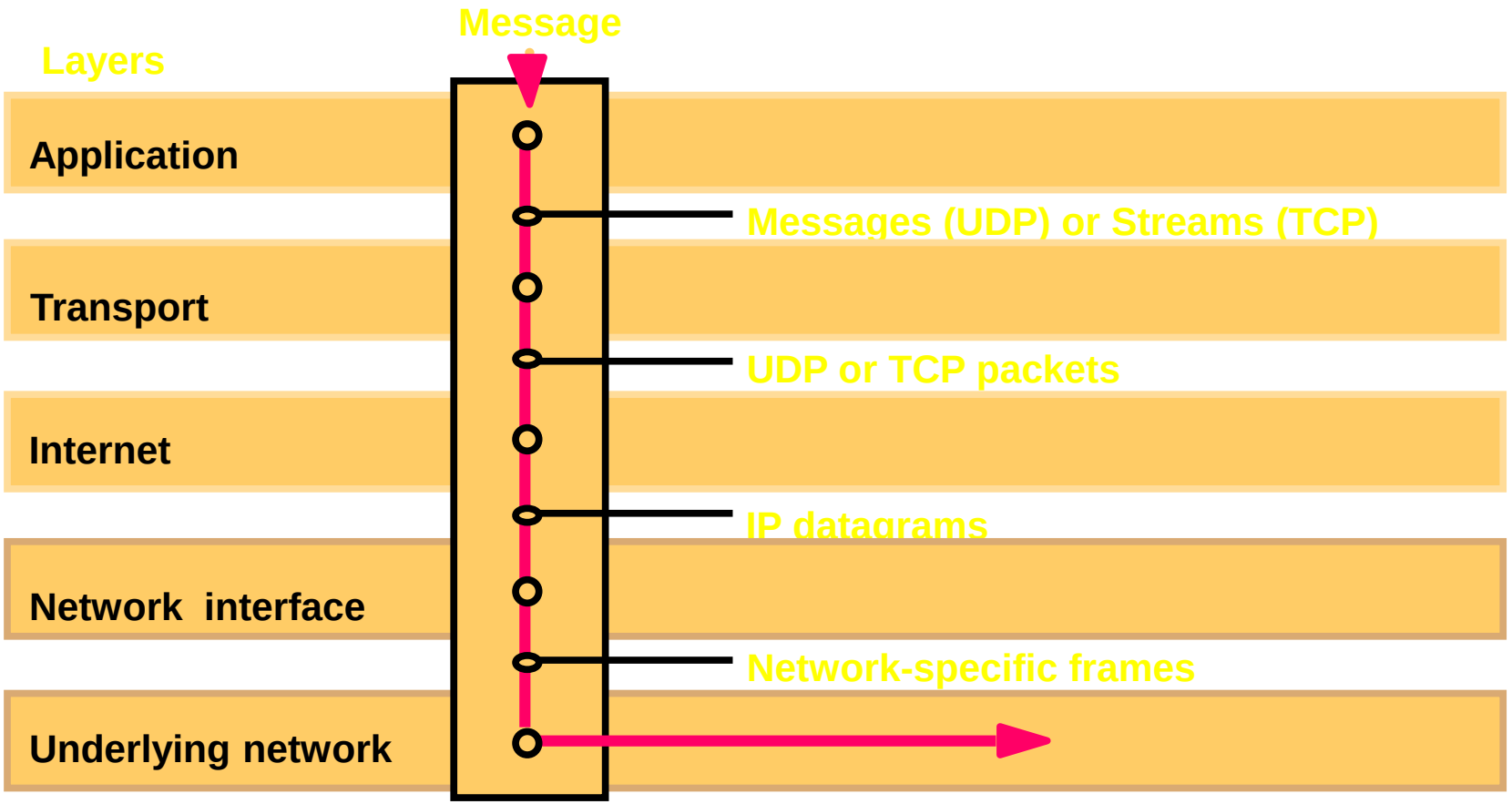
# Internet (a huge distributed system)

- Implementation does not follow OSI
- The first three layers are not distinguished
  - Presentation integrated with application (CORBA – in a single middleware, http and SSL separate)
  - Session and transport
- Transport layer – network independent message transported between pairs of network ports
  - Port – software definable destination points for communication (attached to process)

# Internetwork layers



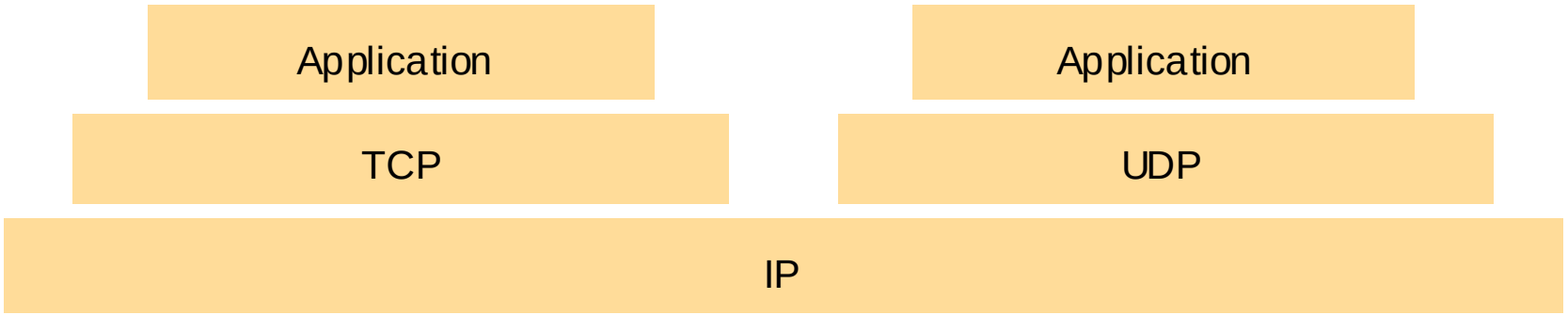
# TCP/IP layers



# TCP/IP Layers

- TCP/IP is independent of underlying transmission technology ->> success
- Build up internetwork base on heterogeneous networks
- Users – single virtual network supporting UDP & TCP while developers of UDP & TCP see a single virtual IP network

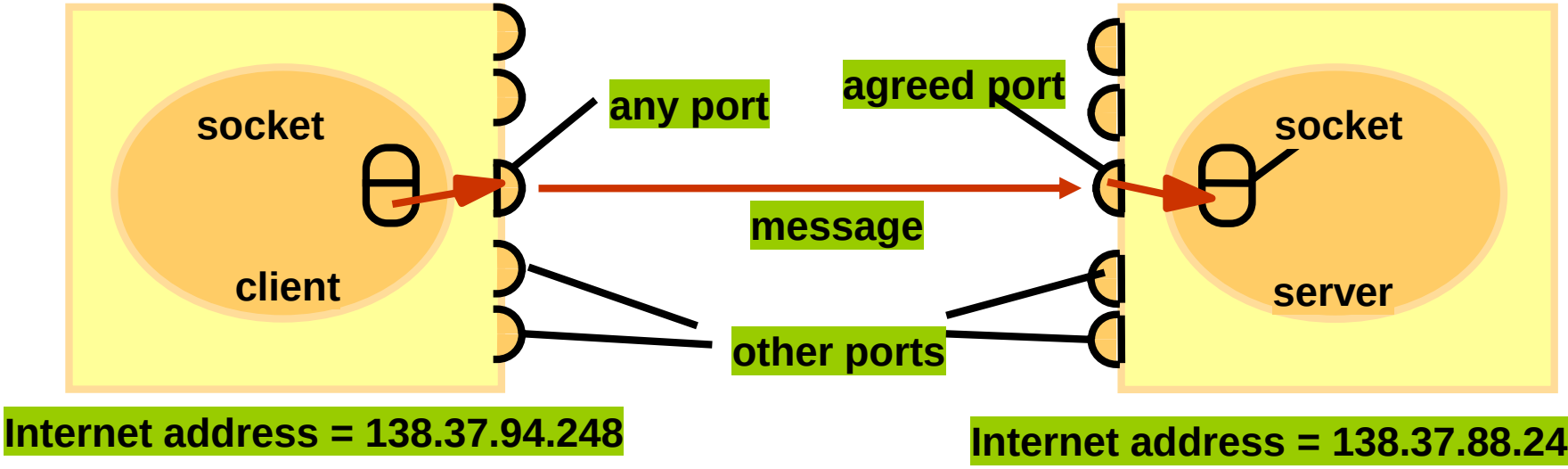
# The programmers view of a TCP/IP Internet



# Sockets

- Socket – endpoint for communication between processes
- Inter Process Communication (IPC) – transmitting message between a socket from one process and a socket in another
- Process use send/receive
  - Create and bind socket to internet address and local port
  - Use the same port for sending and receiving
  - Processes do not share ports (exception to IP multicast)
- Both UDP and TCP use socket

# Sockets and ports

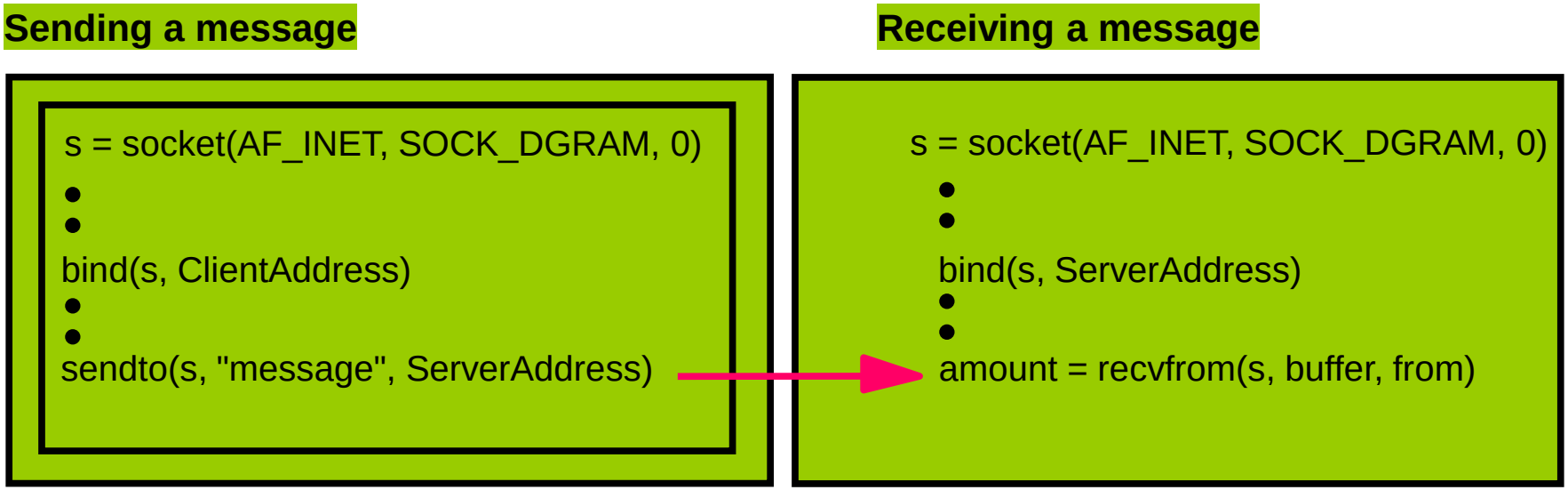


# UDP datagram and TCP stream

- UDP & TCP - transport level protocol
- User Datagram Protocol
  - API to UDP provides message passing abstraction
  - Sender process transmit a single message
  - Independent packets containing the message - datagram
- Transport Control Protocol
  - API to TCP provides the abstraction of a two-way stream between pair of processes
  - Information transferred – stream of data items with no message boundaries (e.g. producer-consumer)

# Socket Used for Datagram

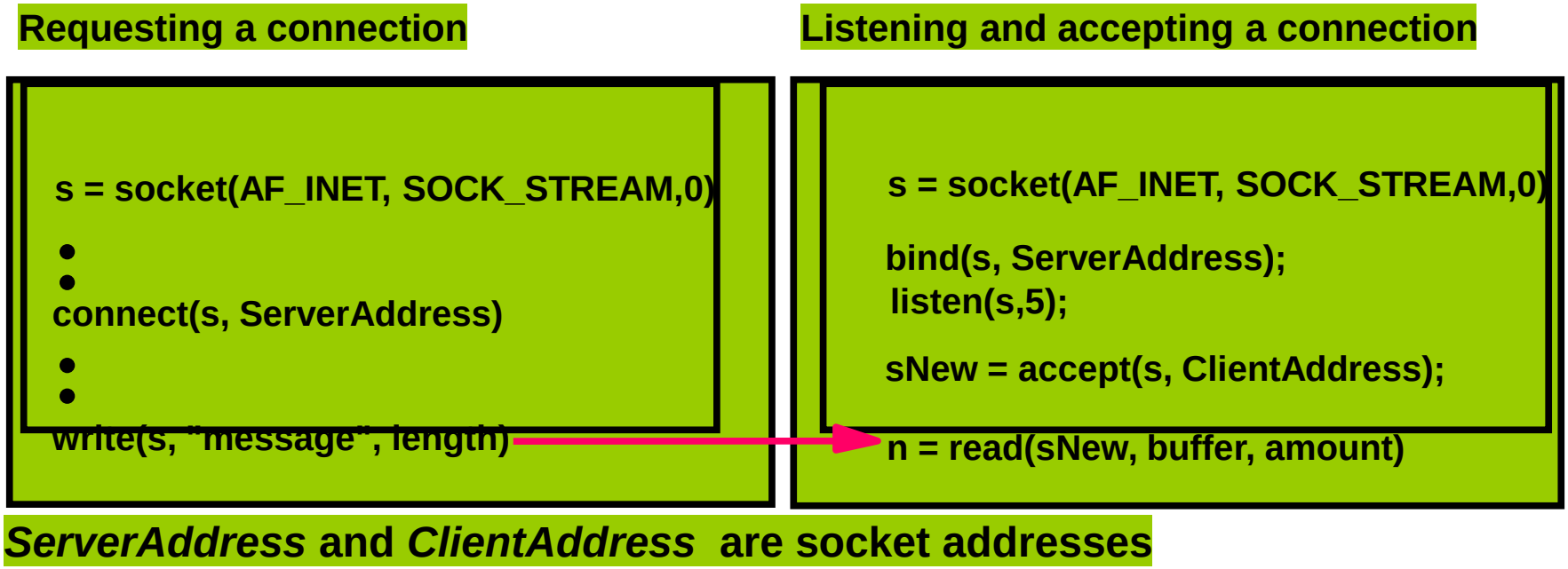
- Both processes create a socket and get a descriptor
- Bind their sockets to socket addresses then communicate




***ServerAddress* and *ClientAddress* are socket addresses**

# Sockets used for Streams

- Listening process (server) create a stream socket and binds to the server's socket address and wait for request
- Server accepts connection and obtain a new socket for communication with client



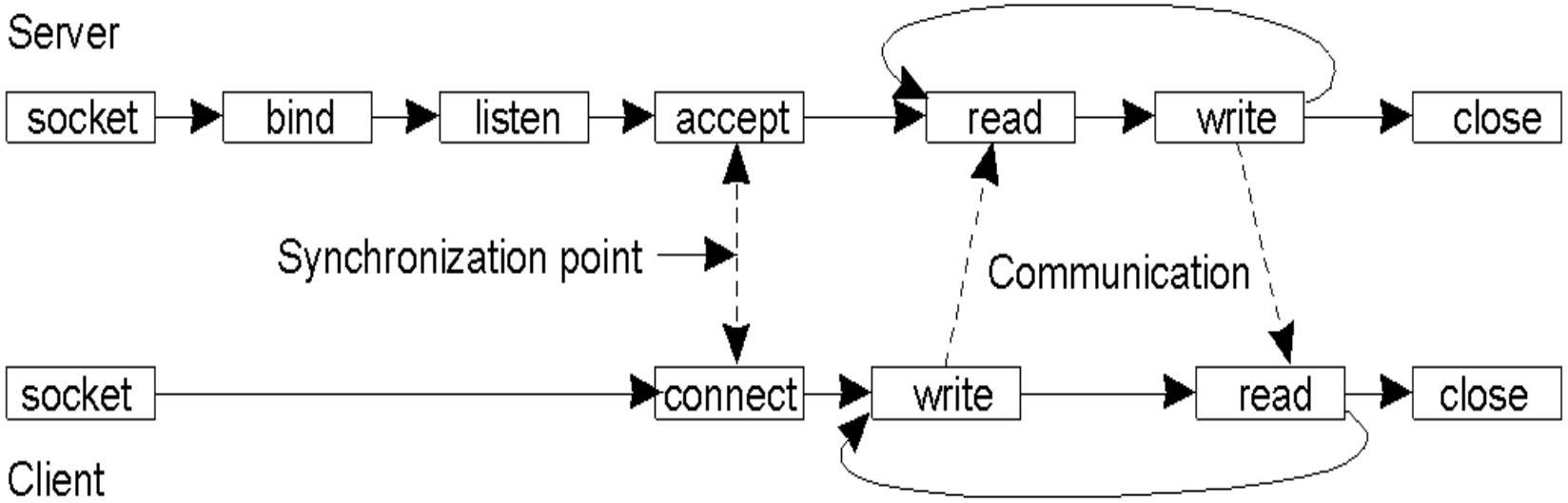

***ServerAddress* and *ClientAddress* are socket addresses**

# Berkeley Sockets (1)

- Socket primitives for TCP/IP.

| **Primitive** | **Meaning** |
|---|---|
| Socket | Create a new communication endpoint |
| 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 |

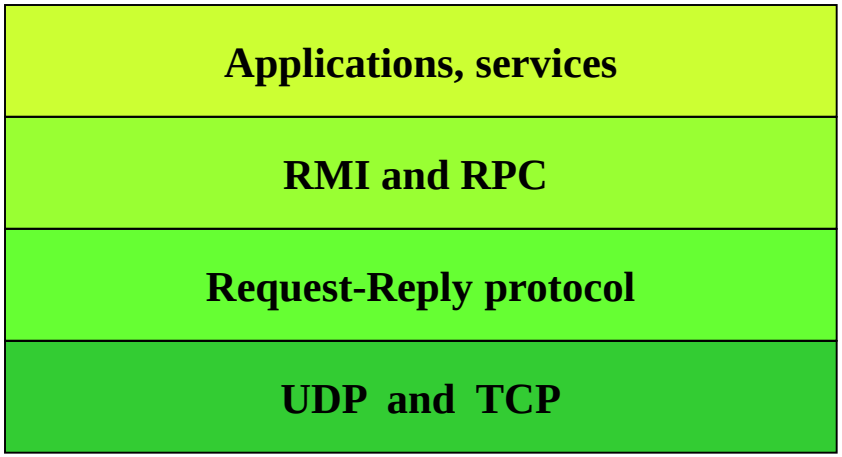
# Berkeley Sockets (2)




- Connection-oriented communication pattern using sockets.

# Request-reply protocol

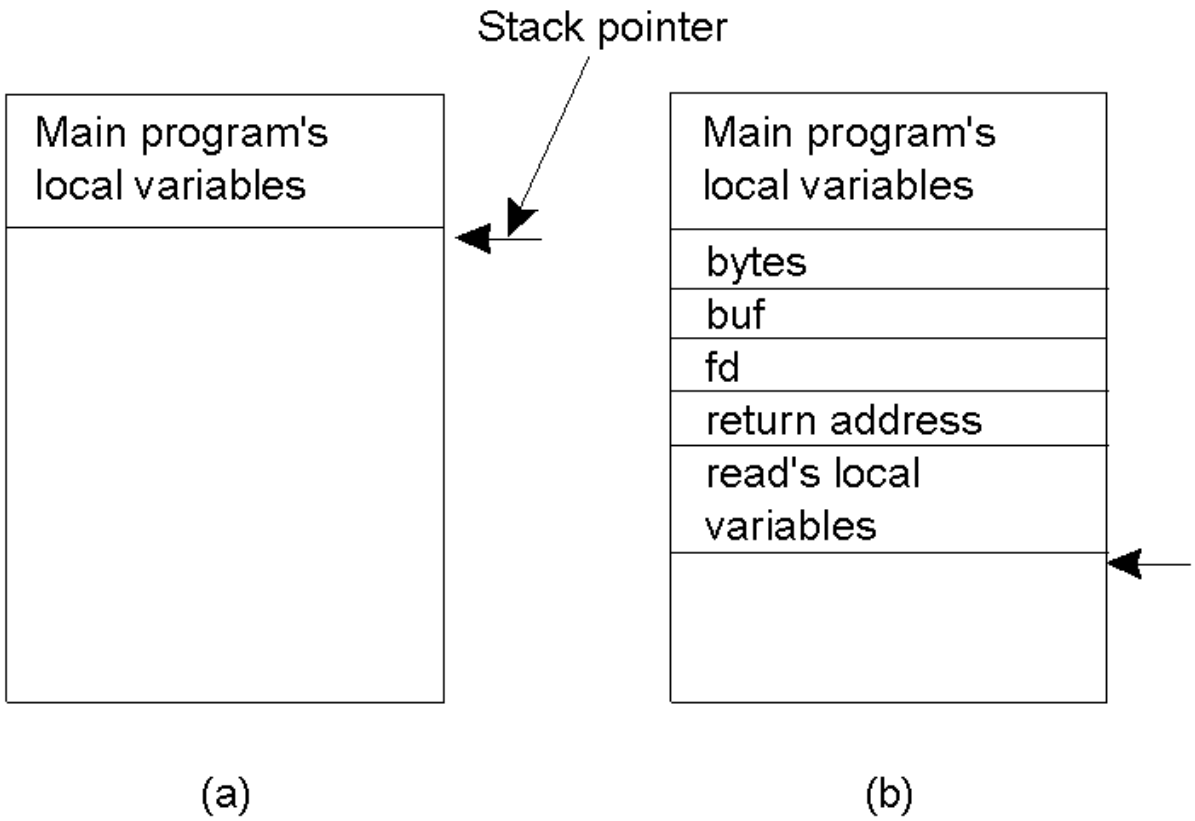
- Were designed to support client-server communication either in the form of RPC or RMI



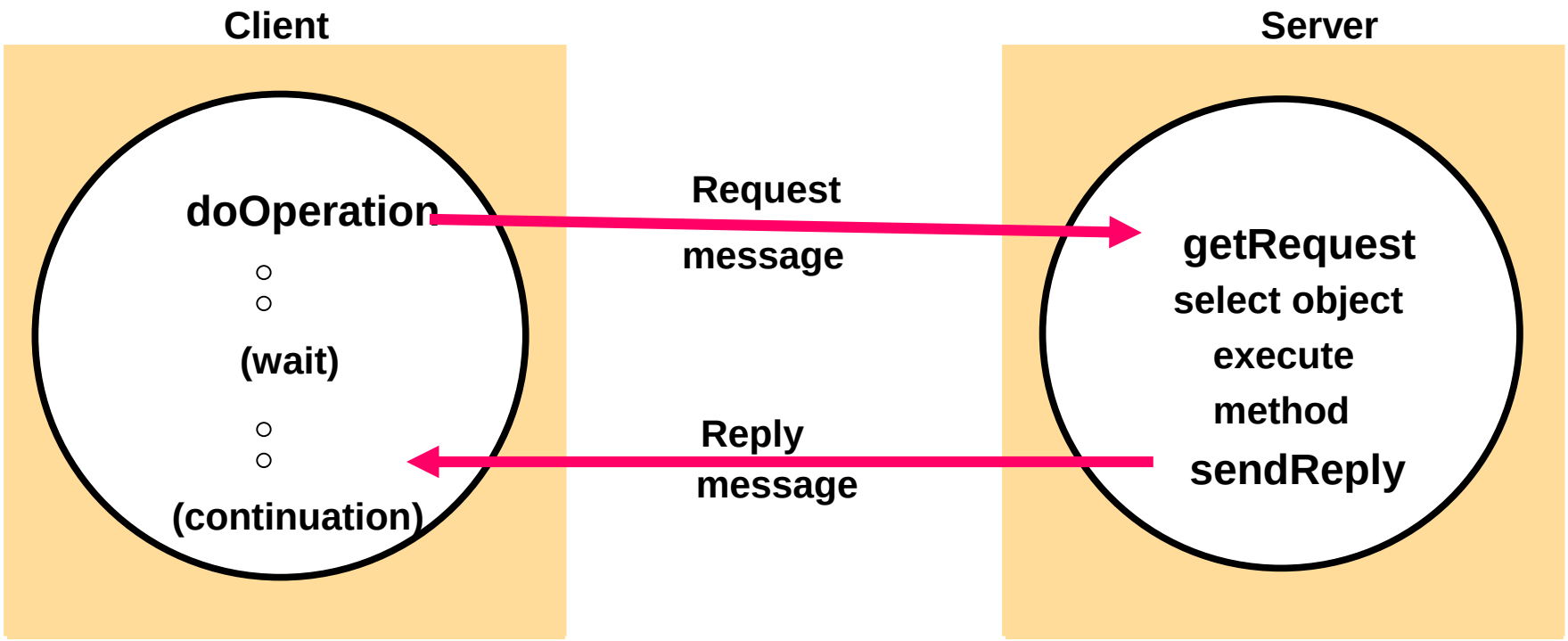
# Conventional Procedure Call

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

b) The stack while the called procedure is active

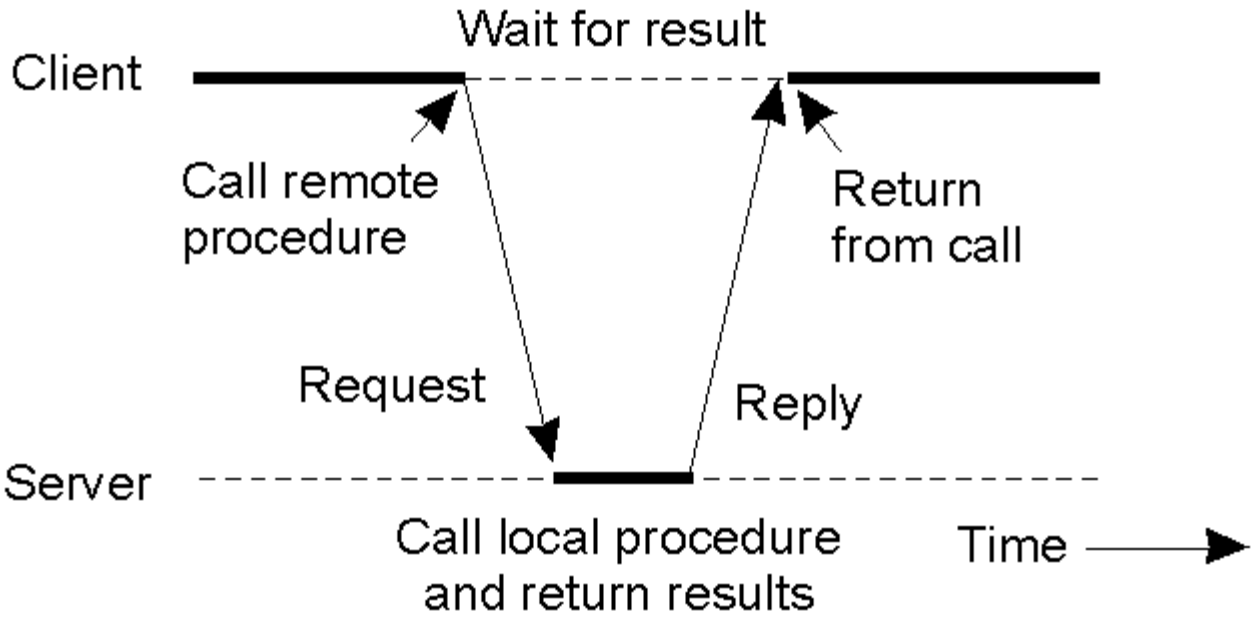


# Request-reply communication



# Client and Server Stubs

- Principle of RPC between a client and server program.



# Remote Procedure Call (RPC)

- Implemented on top of Request-Reply protocol
- Allows a client to call a procedure in a remote process
- Server can call another server, allowing chain of RPCs
- Similar to RMI

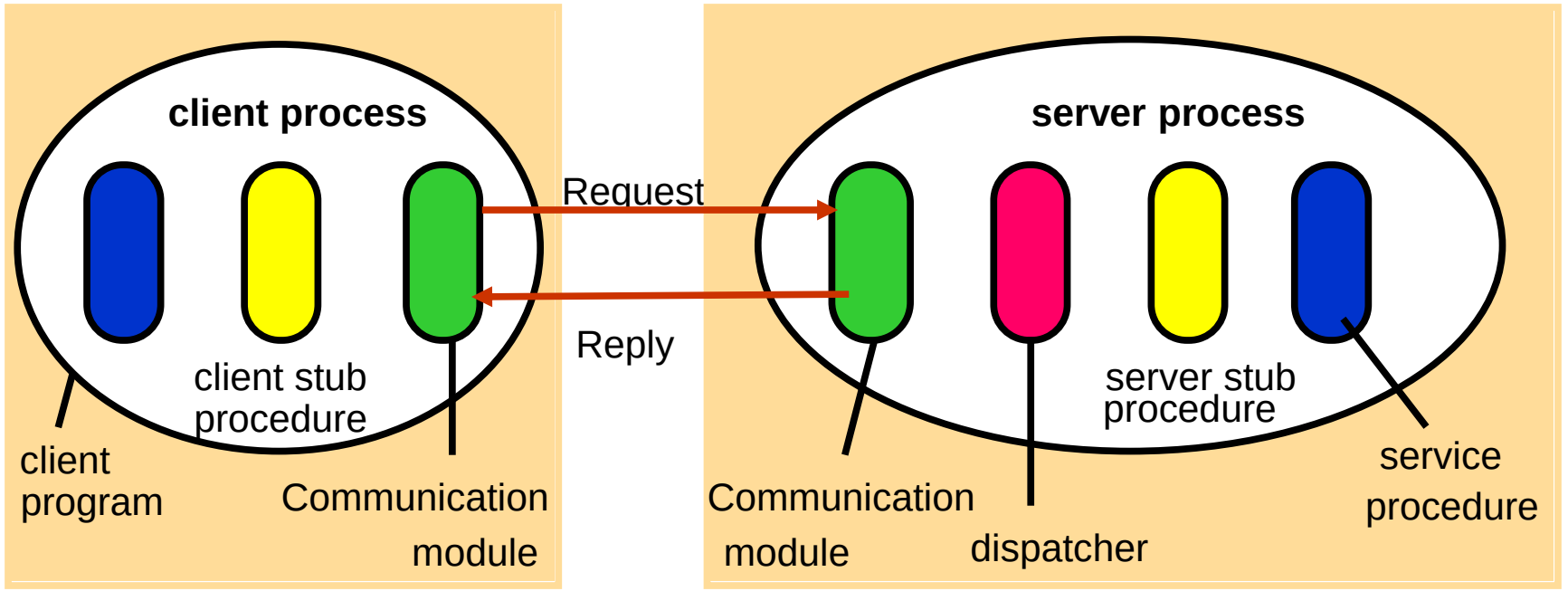
# Remote Procedure Call (cont.)

- Server defines interface of procedure that can call remotely
- Function of the stub:
  - Marshal the procedure id and the argument into a request message which is sent via a communication module to the server
  - Reply - un-marshal the results
- Stub – behave like a local procedure to client
- In the server, Dispatcher selects one of the server stub procedure

# 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

# Client and Server in RPC

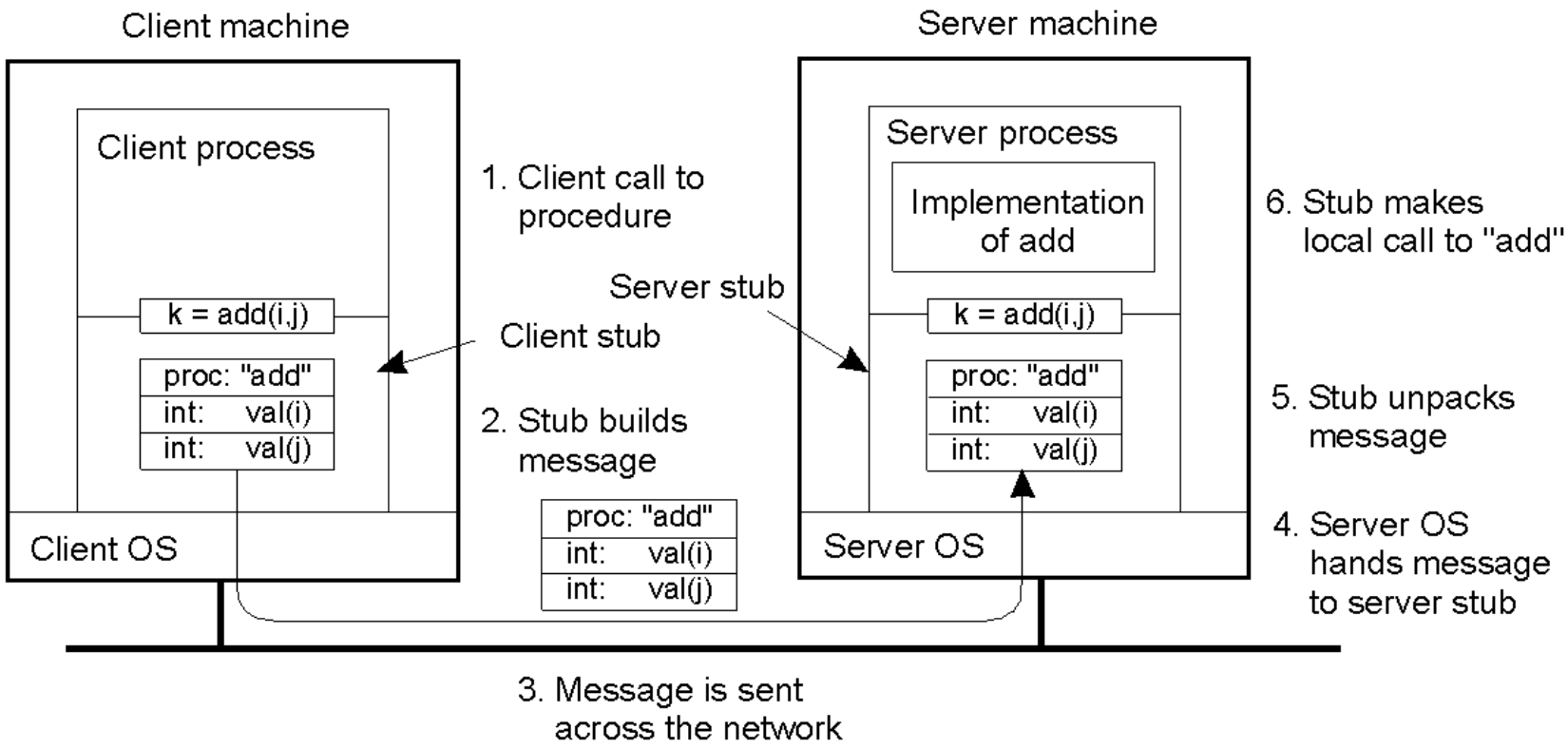


# RPC Exchange Protocol

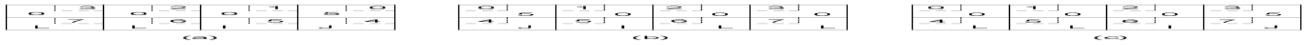
| *Name* | *Messages sent by* | | |
|---|---|---|---|
| | *Client* | *Server* | *Client* |
| R | *Request* | | |
| RR | *Request* | *Reply* | |
| RRA | *Request* | *Reply* | *Acknowledge reply* |

# Passing Value Parameters (1)

- Steps involved in doing remote computation through RPC



# Passing Value Parameters (2)



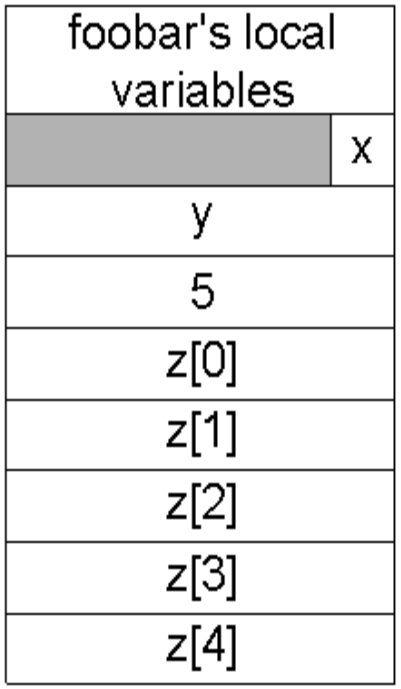
a) Original message on the Pentium
b) The message after receipt on the SPARC
c) The message after being inverted. The little numbers in boxes indicate the address of each byte

# Parameter Specification and Stub Generation

a) A procedure
b) The corresponding message.

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

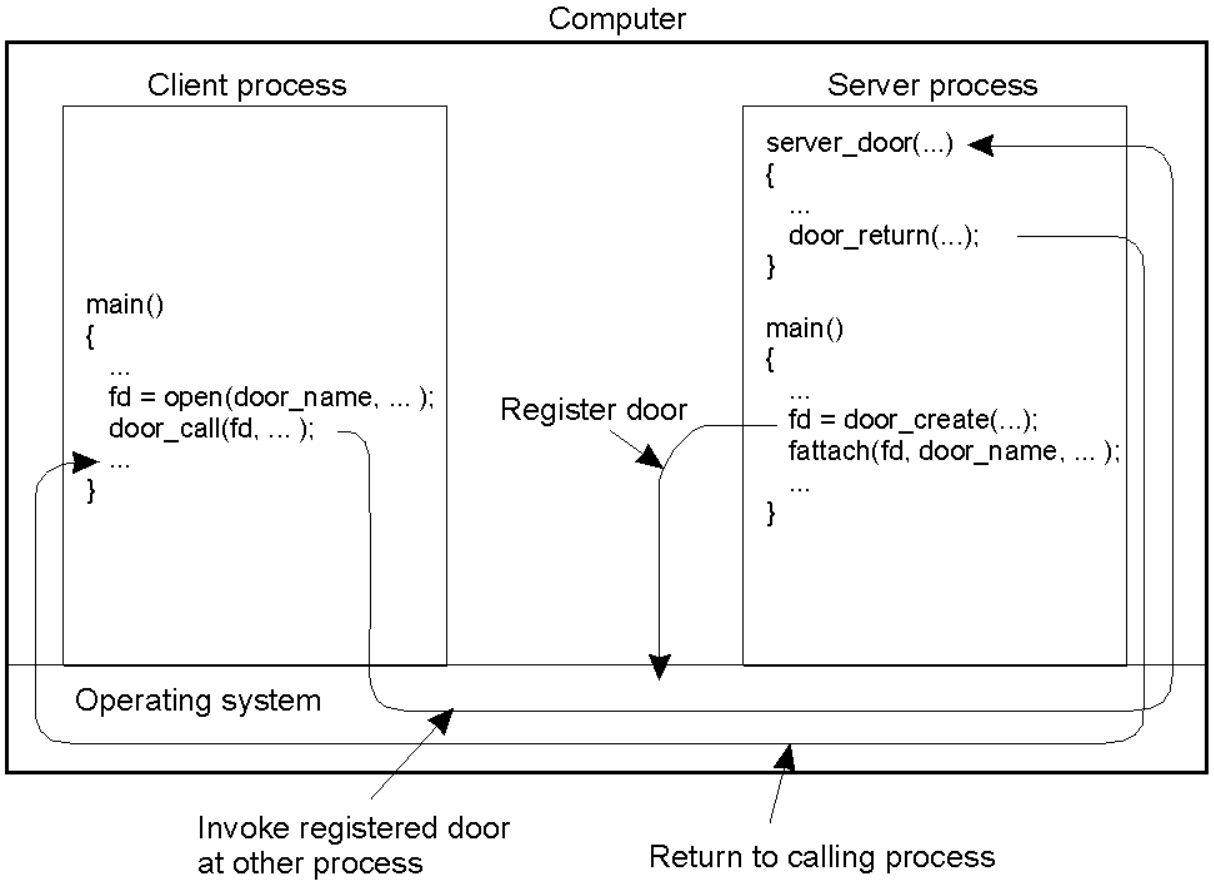
(a)



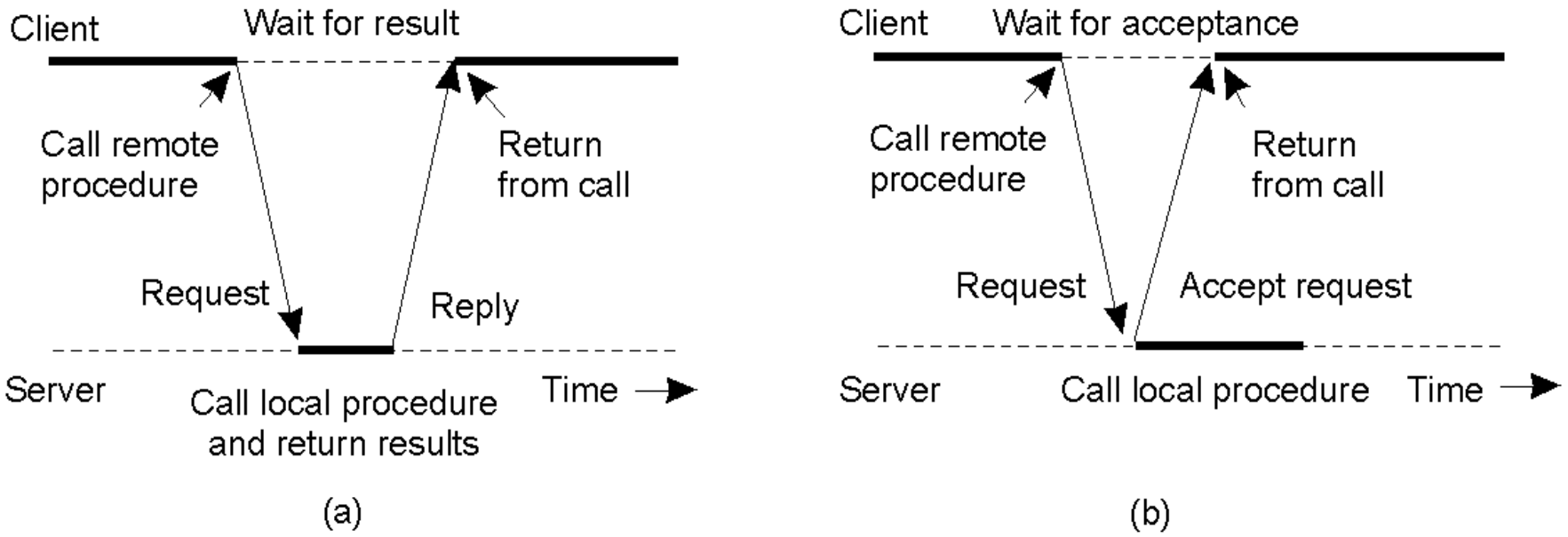

(b)

# Doors

- The principle of using doors as IPC mechanism.



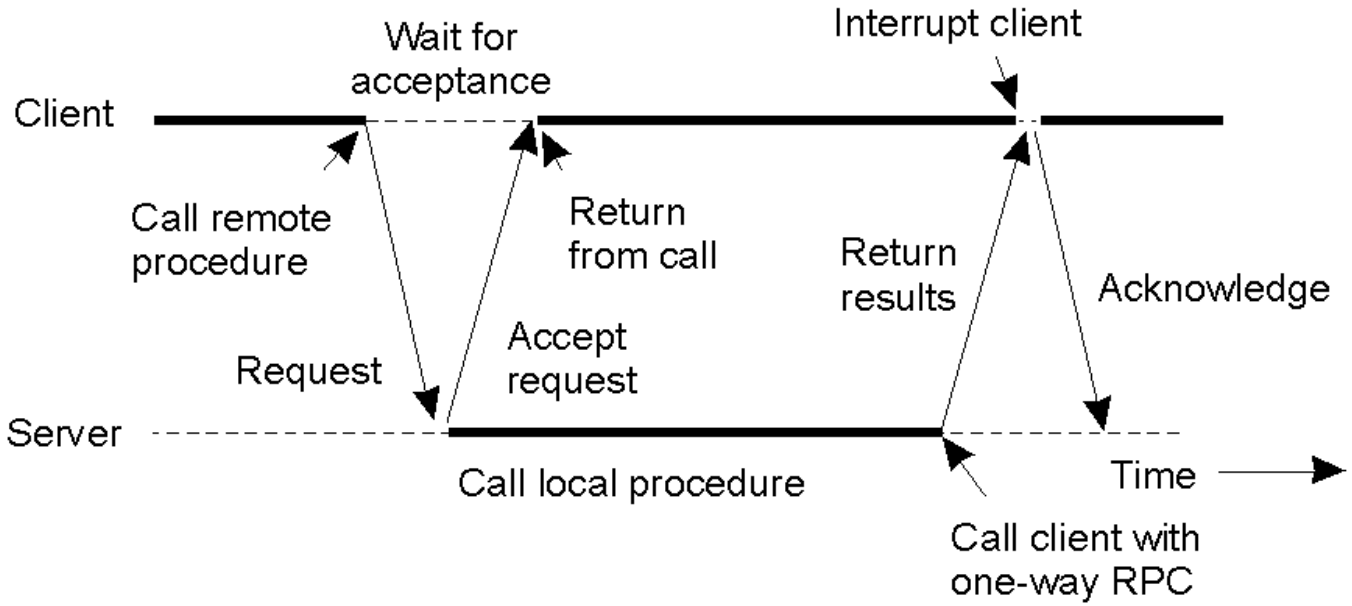
# Asynchronous RPC (1)




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

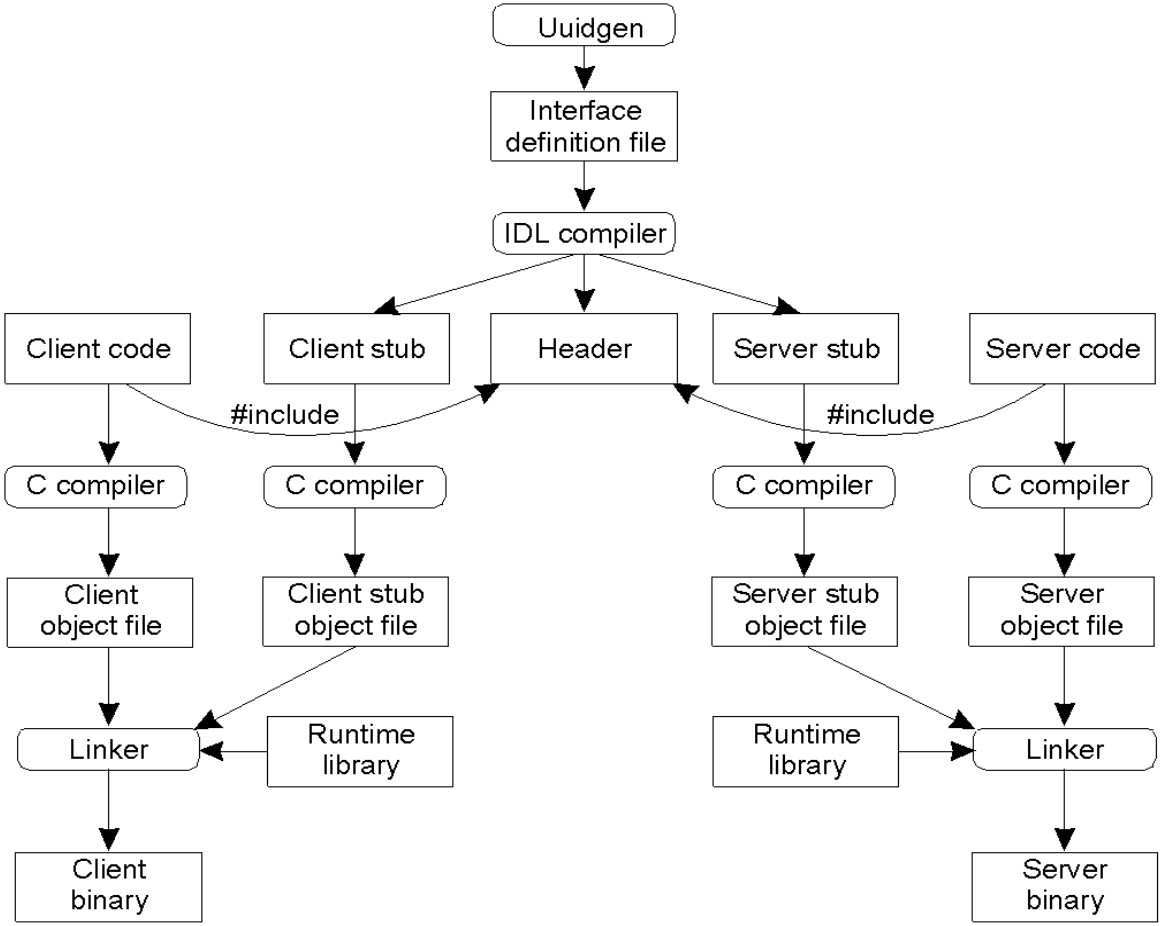
# Asynchronous RPC (2)

- A client and server interacting through two asynchronous RPCs



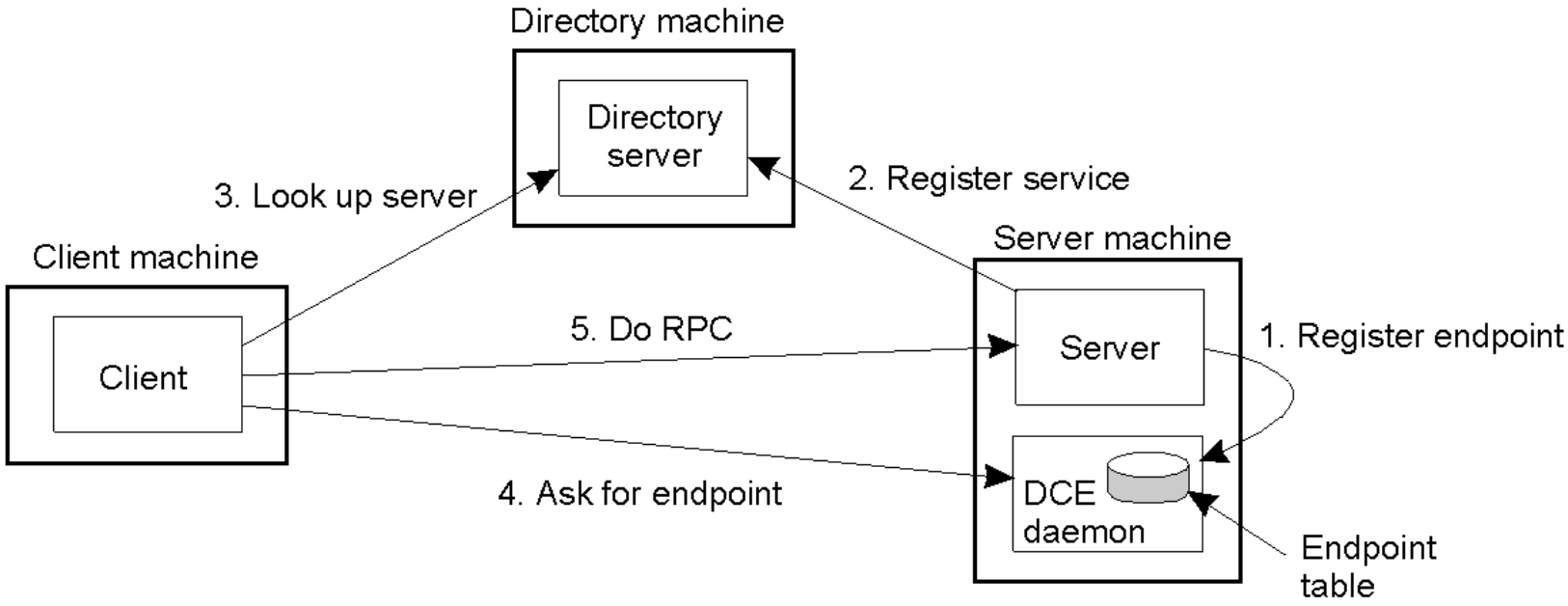
# Writing a Client and a Server

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

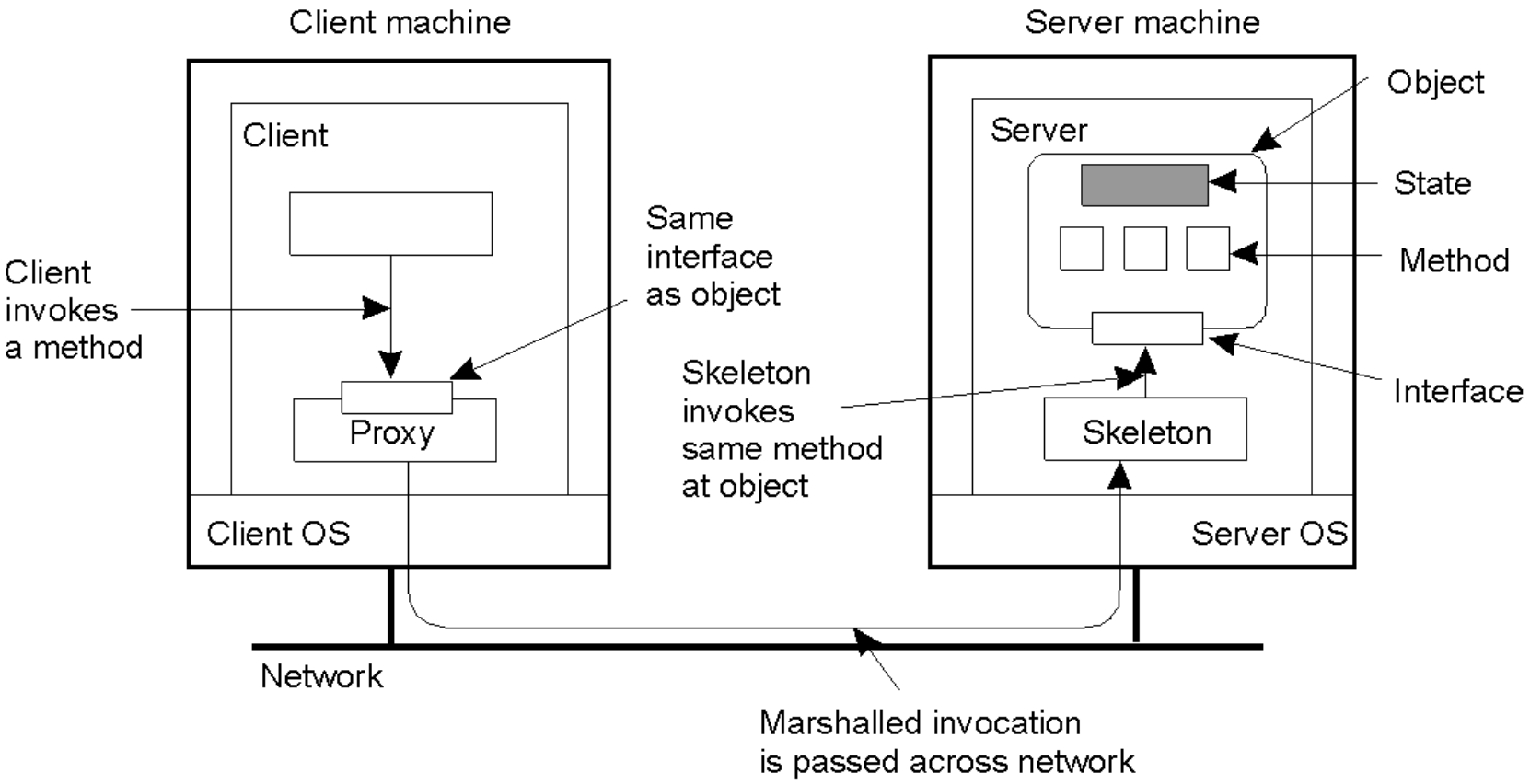


# Binding a Client to a Server

- Client-to-server binding in DCE.



# Distributed Objects



# Binding a Client to an Object

```
Distr_object* obj_ref;          //Declare a systemwide object reference
obj_ref = …;                    // Initialize the reference to a distributed object
obj_ref-> do_something();       // Implicitly bind and invoke a method
```

(a)

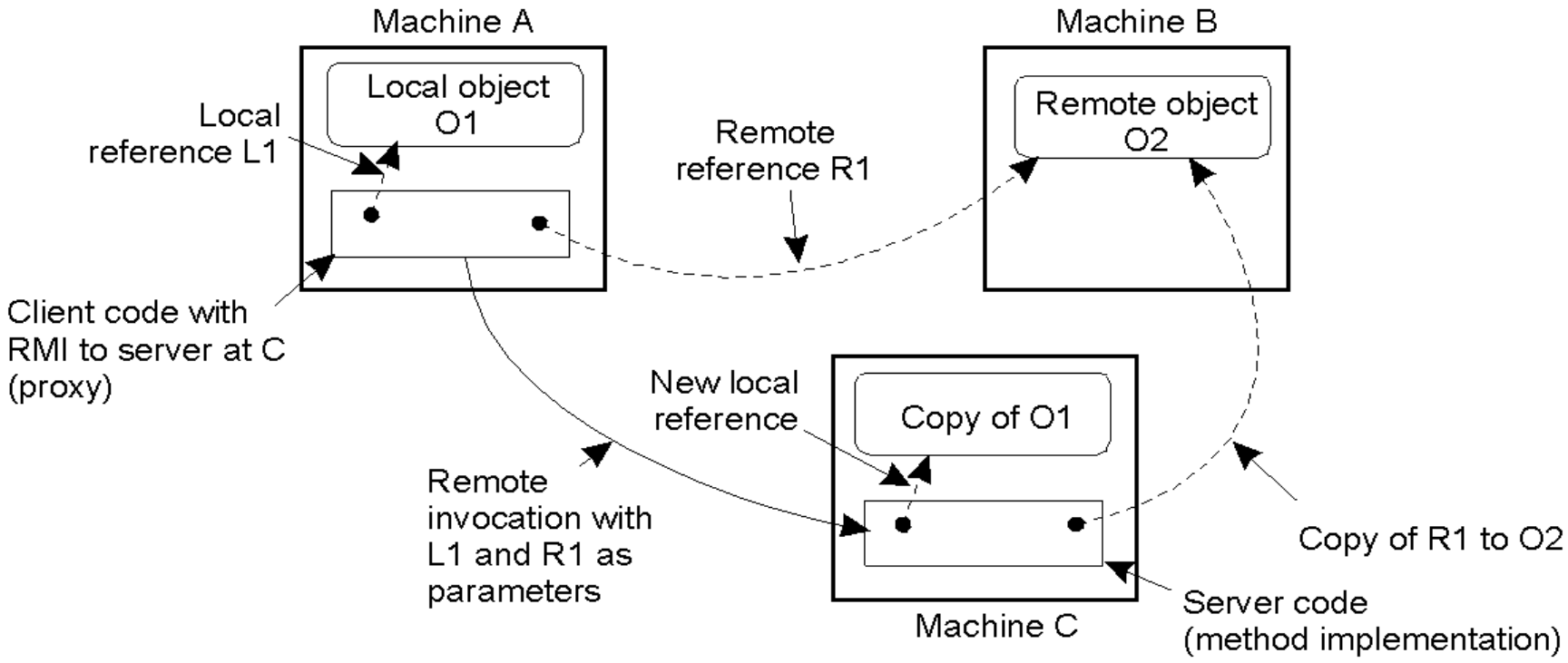
```
Distr_object objPref;           //Declare a systemwide object reference
Local_object* obj_ptr;          //Declare a pointer to local objects
obj_ref = …;                    //Initialize the reference to a distributed object
obj_ptr = bind(obj_ref);        //Explicitly bind and obtain a pointer to the local proxy
obj_ptr -> do_something();      //Invoke a method on the local proxy
```

(b)

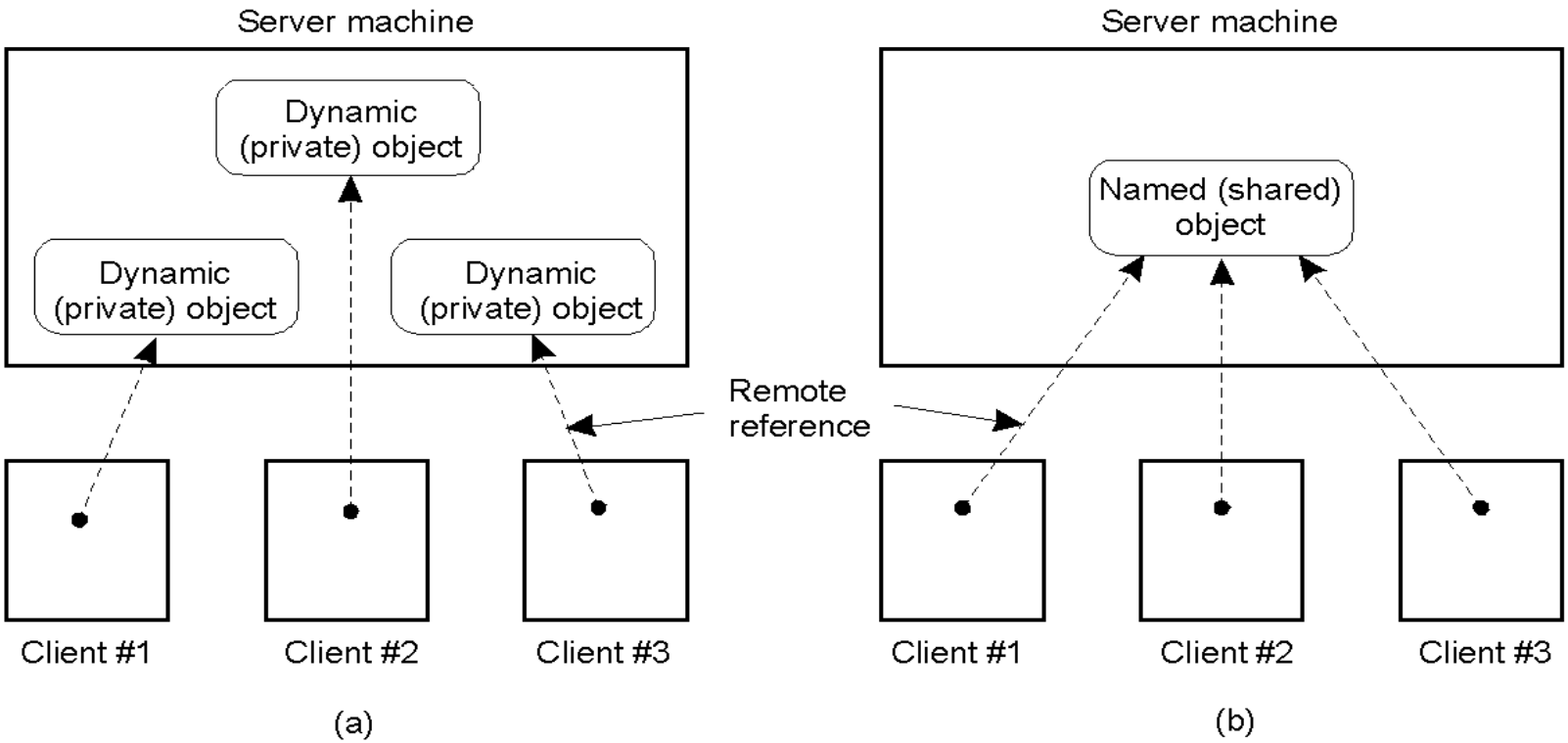
(a) Example with implicit binding using only global references
(b) Example with explicit binding using global and local references

# Parameter Passing

- The situation when passing an object by reference or by value.



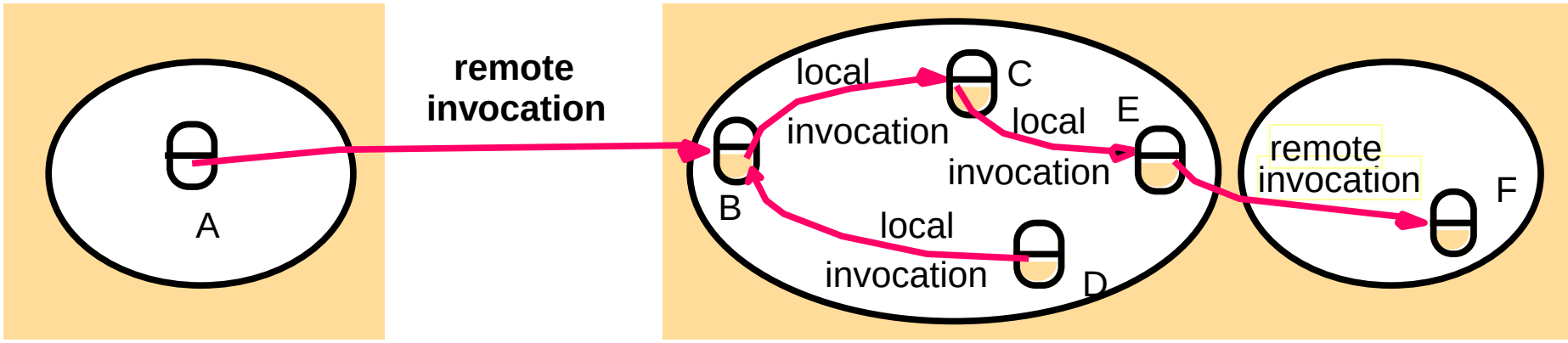
# The DCE Distributed-Object Model




a) Distributed dynamic objects in DCE.
b) Distributed named objects

# Remote Method Invocation (RMI)

- Method invocations between objects in different processes (within the same computer or not)

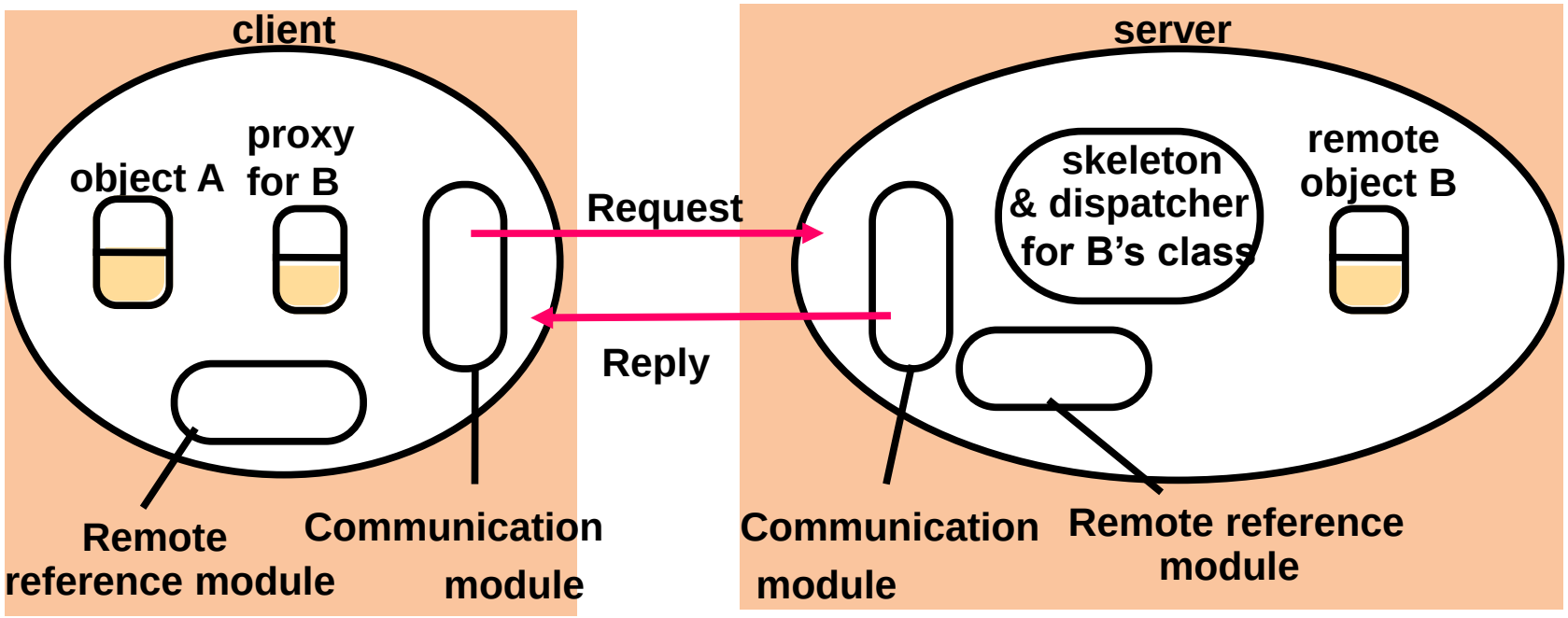



**Remote and local method invocations**

# RMI (Implementation)

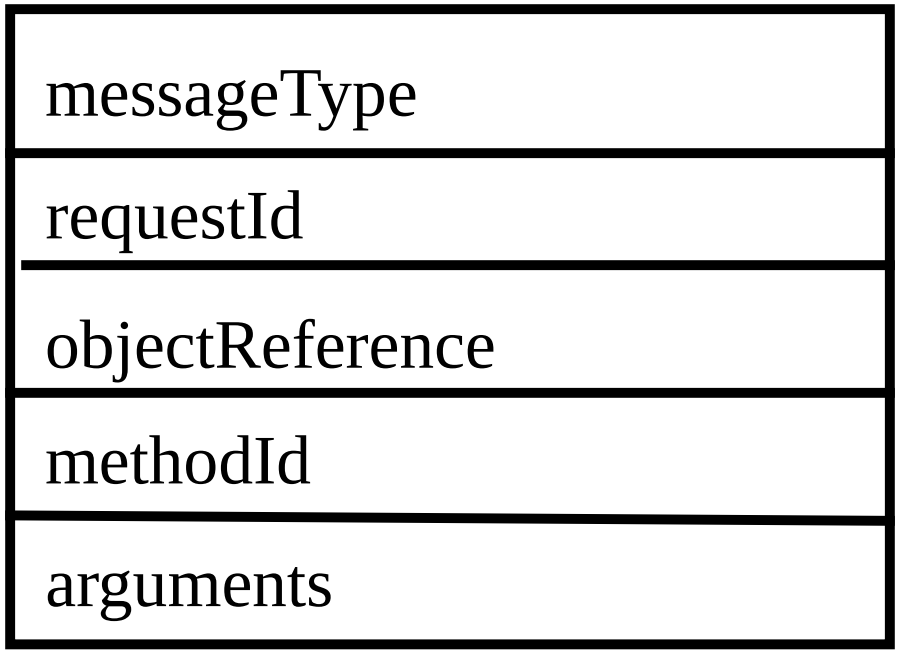
- Communication module carries out request-reply protocol
- Remote reference module translate between local and remote object references and create remote object references
- RMI software
  - Proxies – transparent by hiding the details of object reference, (un)marshalling arguments/results and send/receive message
  - dispatchers – select the appropriate method and pass to request message
  - Skeleton implements methods in remote interface

# The role of proxy and skeleton in remote method invocation



# Request-reply message structure

- Communication module uses the first three items




*int (0=Request, 1= Reply)*

*int*

*RemoteObjectRef*

*int or Method*

*array of bytes*

Thank you