Parte: 4a – Modelli, Middleware, e Remote Call

Corso di: Sistemi Distribuiti
Lauree in: Ingegneria Informatica,
delle Telecomunicazioni ed Informatica di Scienze

Prof. Paolo Nesi

Department of Systems and Informatics, University of Florence
Via S. Marta 3, 50139, Firenze, Italy
tel: +39-055-2758515, fax: +39-055-2758570

DISIT Lab, Sistemi Distribuiti e Tecnologie Internet

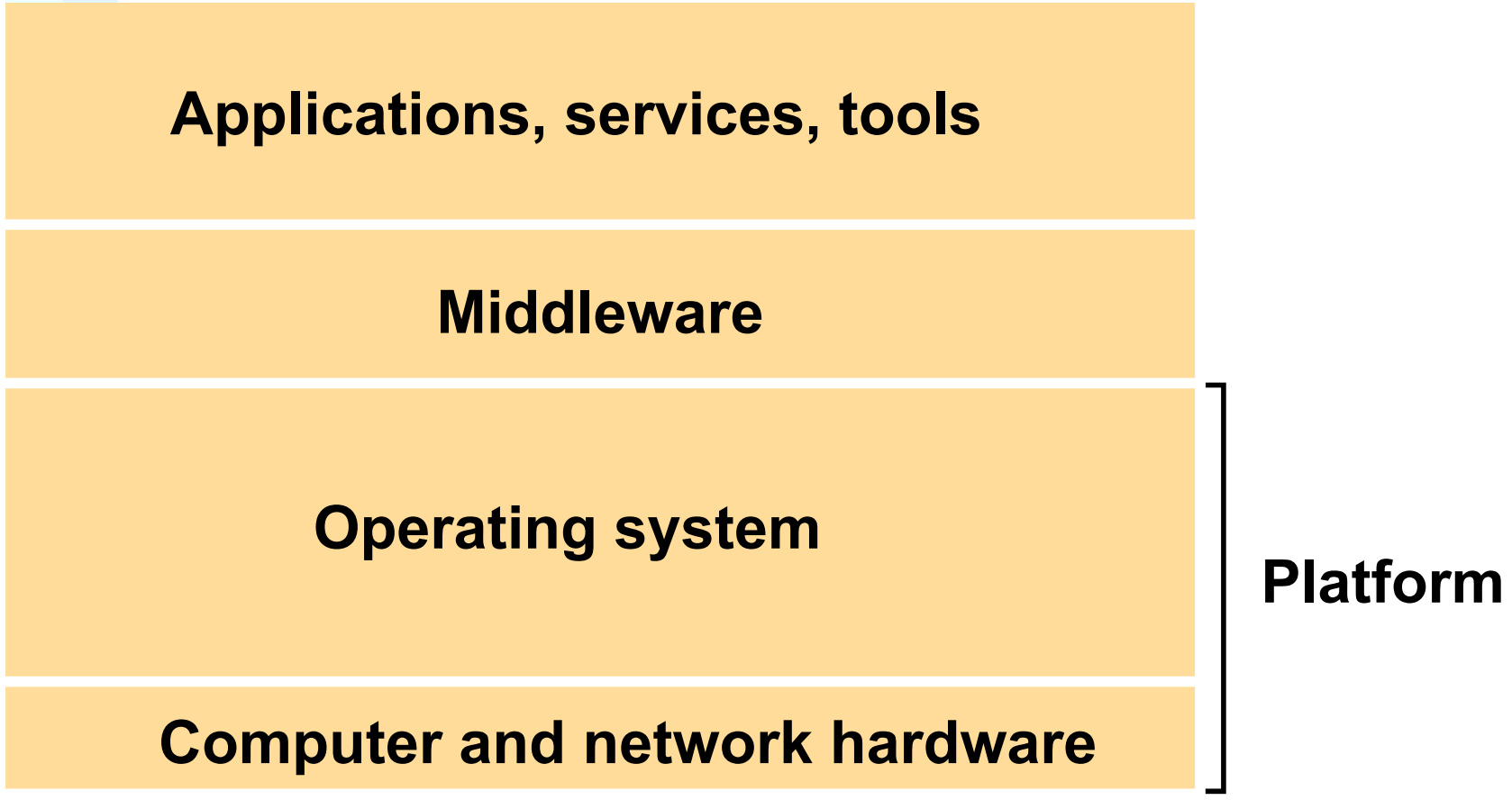
http://www.disit.dinfo.unifi.it/

paolo.nesi@unifi.it
http://www.disit.dinfo.unifi.it/nesi

Modelli ed Architetture

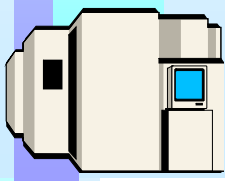
- Evoluzione delle architetture
- Client Server, Comunicazione fra processi
- Proxy, peer process, WEB applets, Thin clients
- Modelli di Sistemi Mobili
- Problemi di progettazione di Sistemi Distribuiti
- Modelli di Interazione sincroni ed asincroni
- Modelli di Sincronizzazione di eventi
- Modelli di Sicurezza e distribuzione contenuti

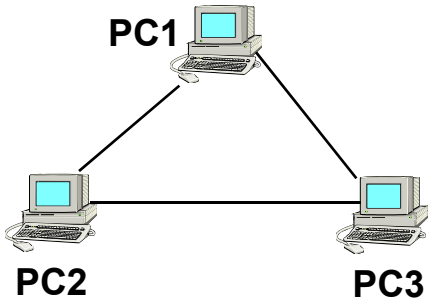
Software and hardware service layers in distributed systems

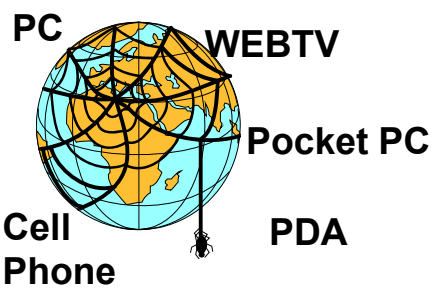


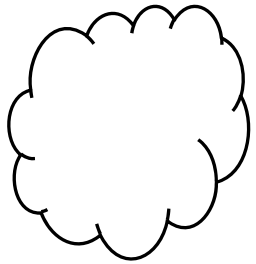
The Shifting Paradigm

Mainframes → *PC's* → *The Web* → *Users*







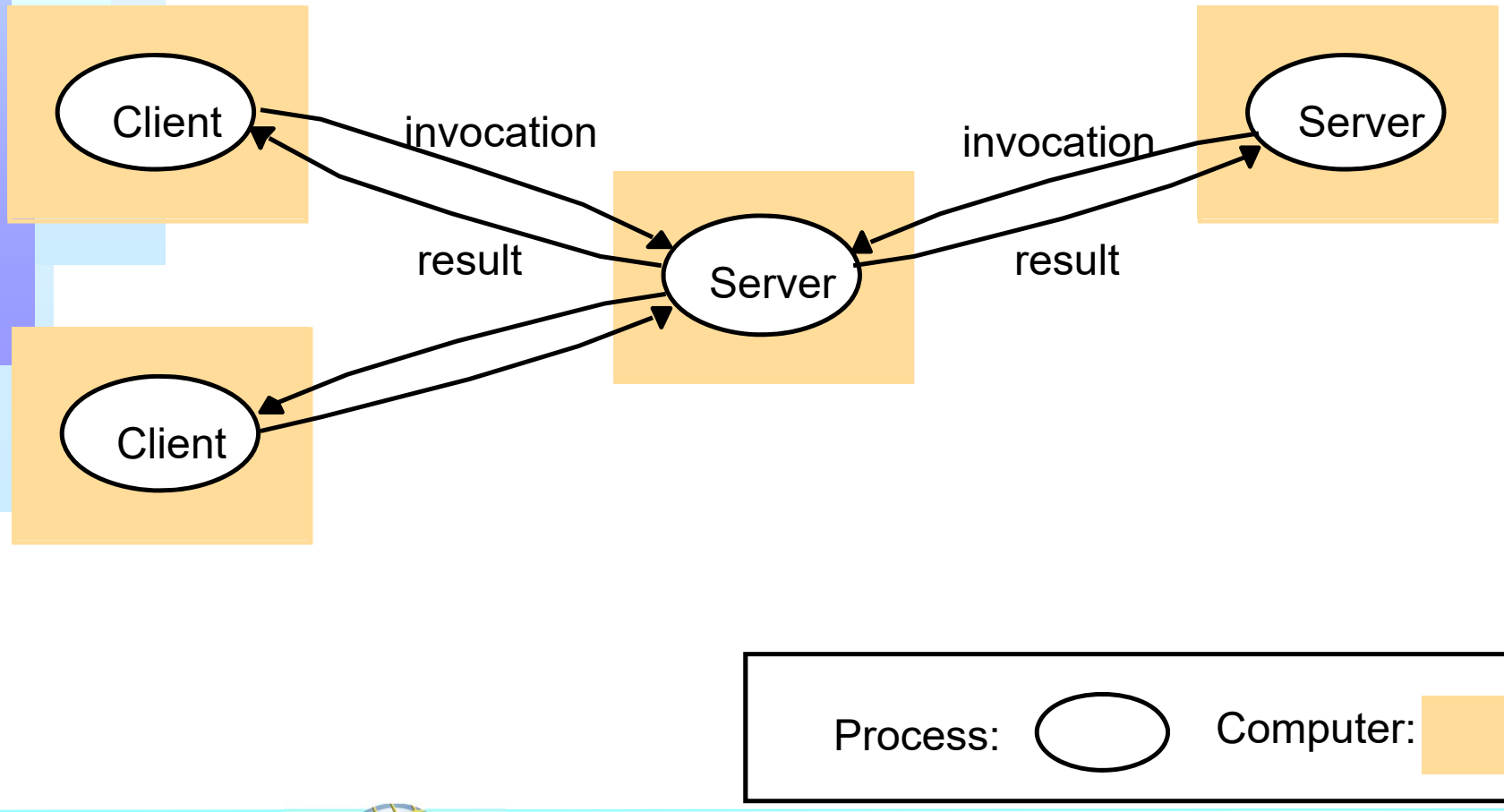


HARDWARE	**SOFTWARE**	**MIDDLEWARE**	**data/cloud**
IBM	**MICROSOFT**	**…….**	**W3C**
CLOSED PROPRIETARY	**CLOSED PROPRIETARY**	**OPEN STANDARDS**	**Multiple Formats**

Modelli e Architetture

- **Nelle prossime slide sono proposti**
  - i principali pattern/configurazioni e concetti alla base delle architetture distribuite
  - I principali problemi derivati dalle architetture e dei sistemi distribuiti in genere
- **Gli elementi di base sono**
  - **Server Processes**
    - Provide Services/Answers to the Clients
  - **Client Processes**
    - Produce Requests to the Servers
  - **Peer Processes**
    - Play the role of both server and client
      - Provide Services and Produce Requests
    - Sharing data and controls, Collaborating to processes

Clients invoke individual servers
Client Server Model



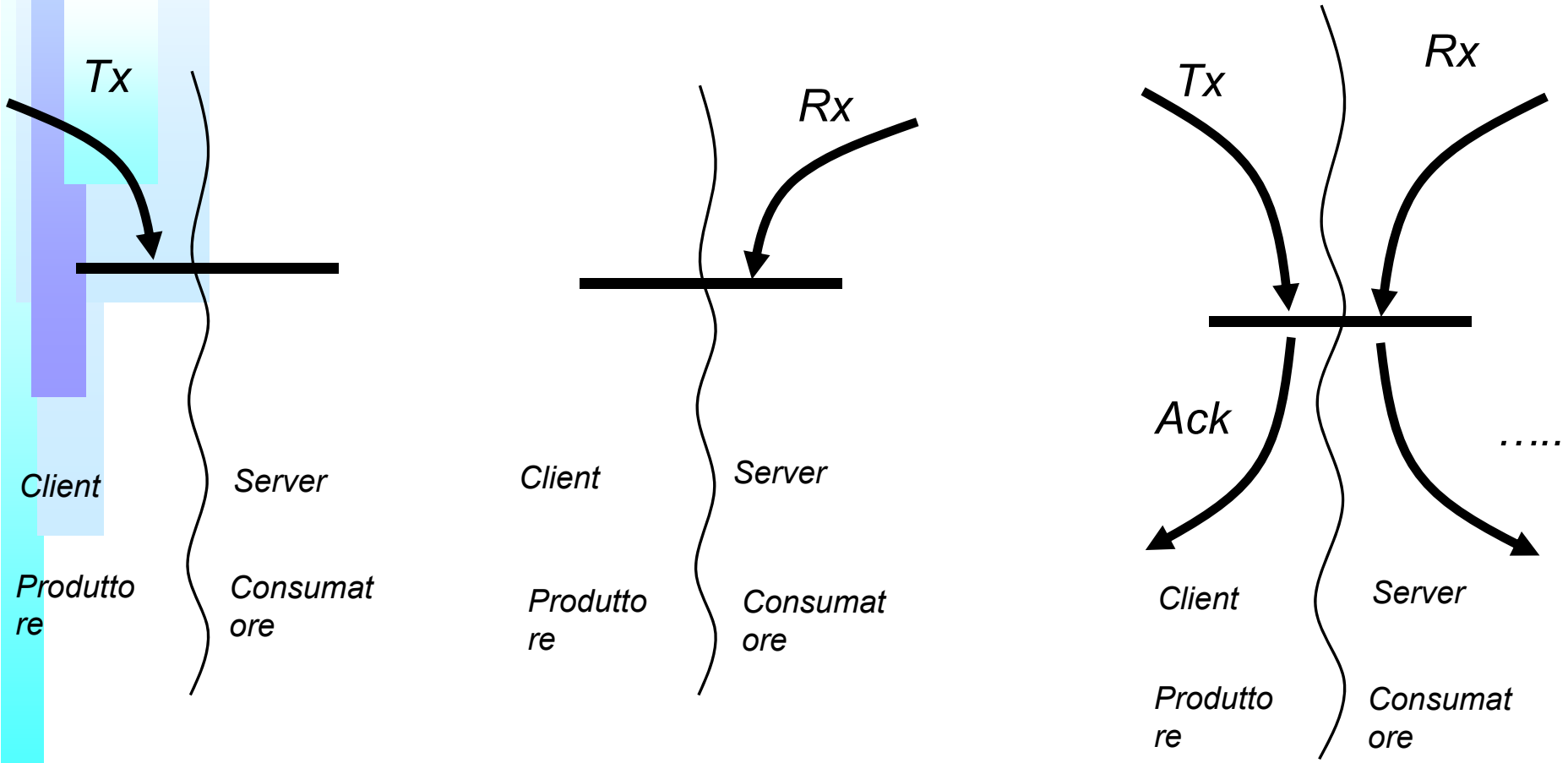
Comunicazioni fra processi

- Due operazioni: *send* and *receive*
  - un processo invia un comando o un dato ad un altro processo
  - il destinatario riceve il comando e lo processa,
    - al limite da conferma a chi gli ha inviato il messaggio/comando,
    - la conferma viene chiamata
      - Acknowledgement, ACK

- Comunicazione puo' essere
  - Sincrona, Synchronous
  - Asincrona, Asynchronous

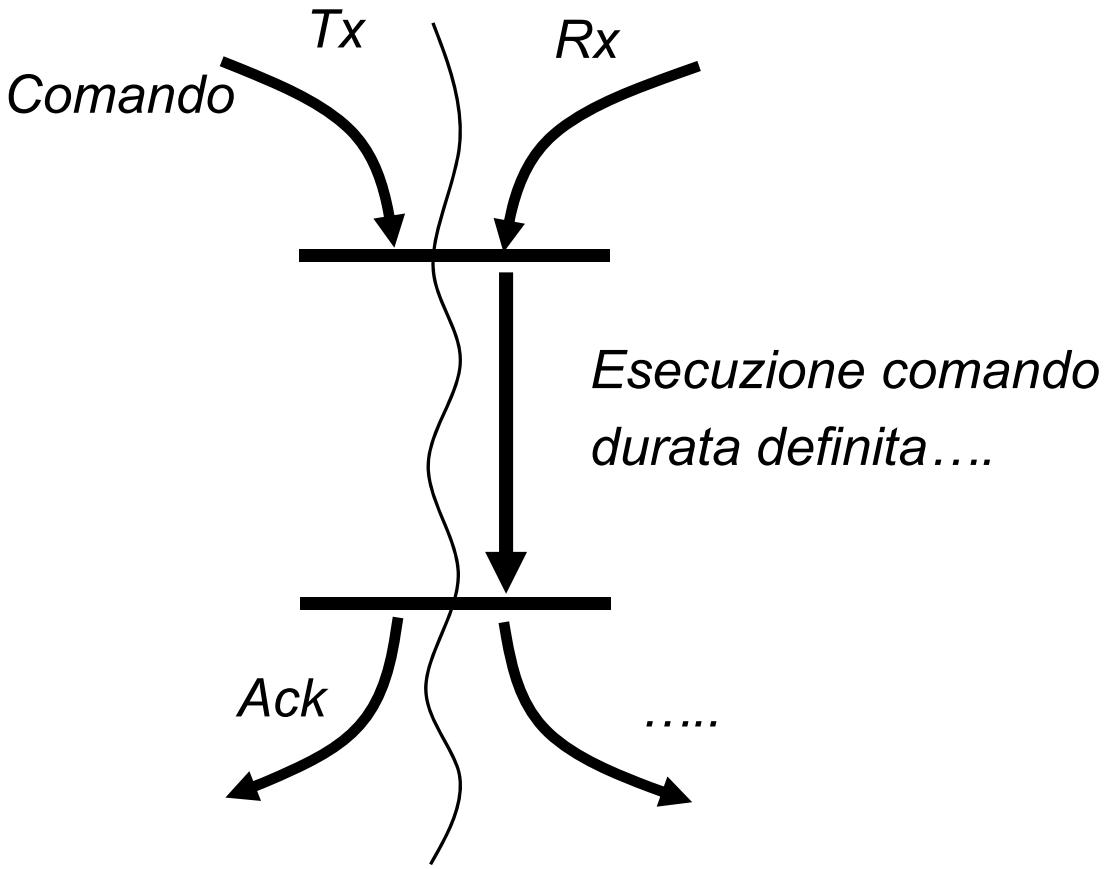
Comunicazione Sincrona

- Send and receive are blocking operations



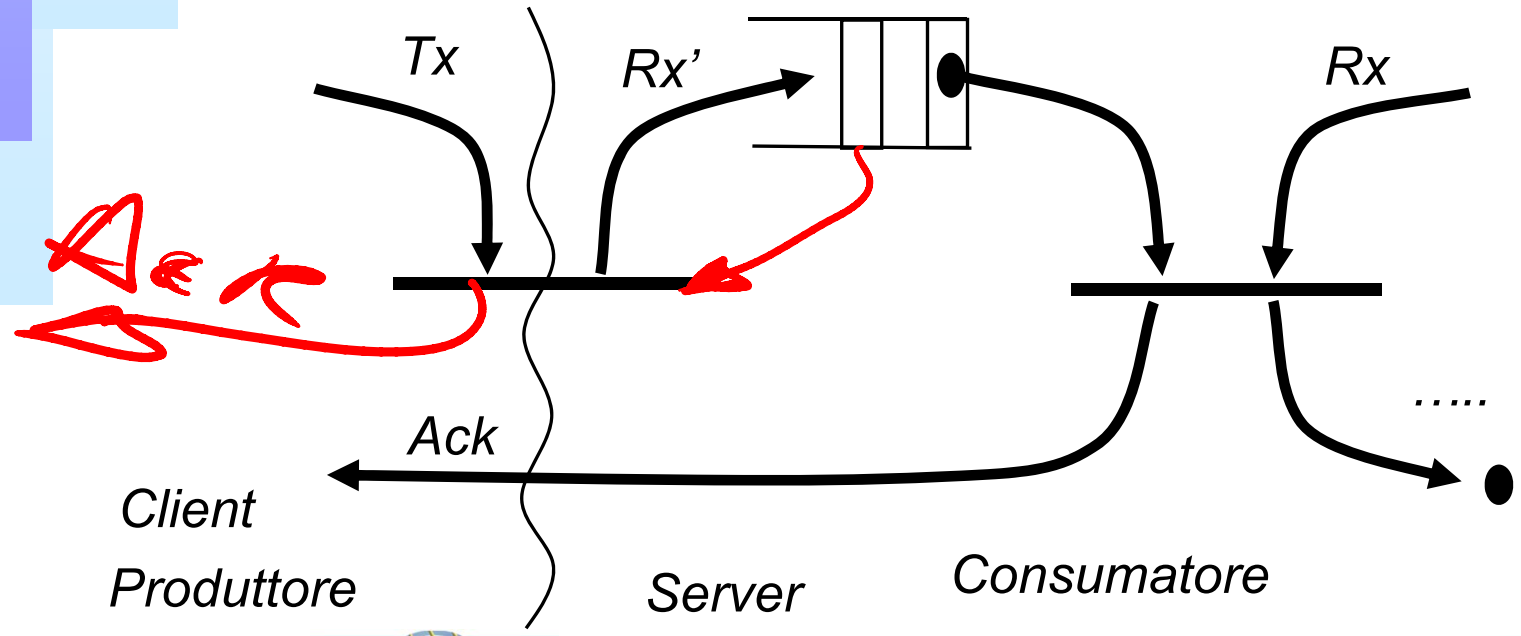
Comunicazione Sincrona

- Send and receive are blocking operations



Comunicazione Asincrona

- **Send** is non blocking
- **Receiving is blocking** (for the first msg), and non blocking
- A queue is associated with each message destination
  - Send => message added to remote queue
  - Receive => message removed from local queue



History of Distributed Object Models

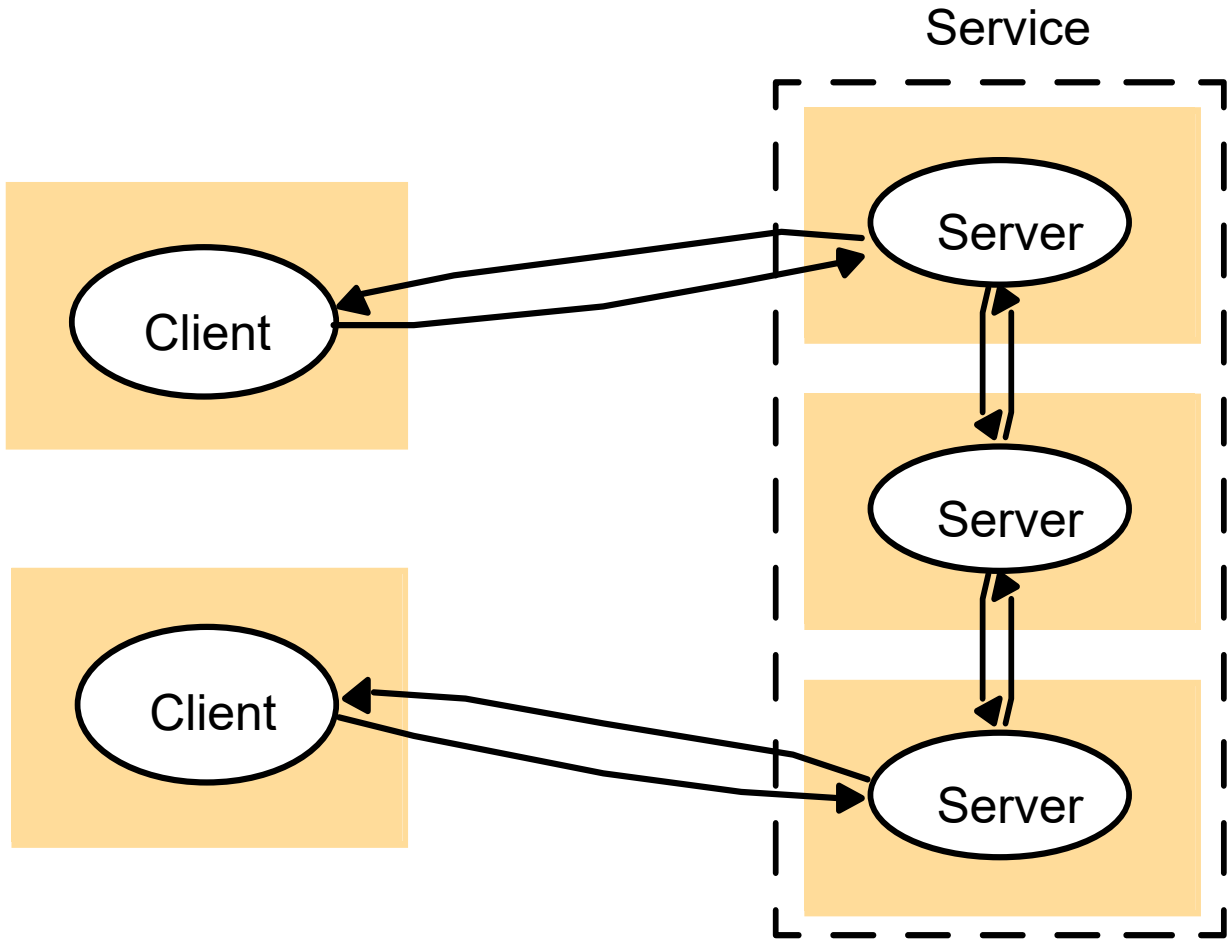
Communication Protocol Models:

- Message passing / queueing (DCE, Distrib. Comp. Environment)
- Request/response (RPC, remote procedure call)

- **1980: model based on network** layer
  - NFS (network file system)
  - DCE (Distributed Comp. Environment)
  - RPC (remote procedure call)

- **1990: object-oriented RPC**, to link objects

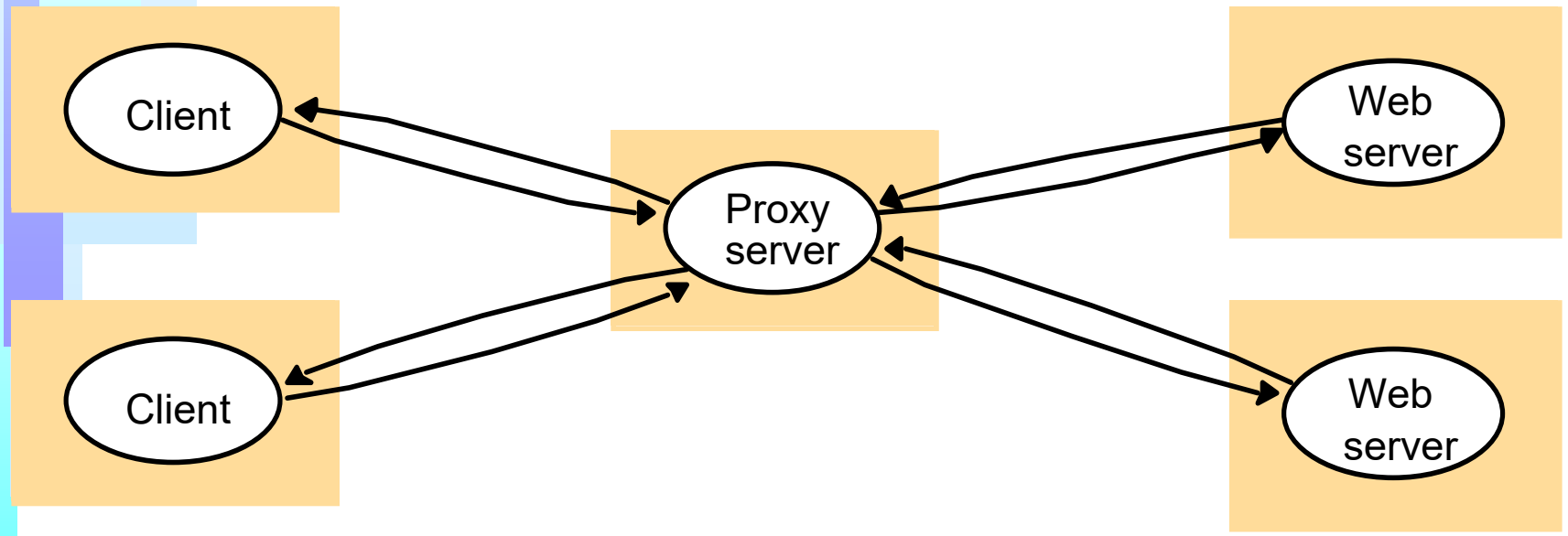
A service provided by multiple servers Clients of each other, CLUSTER



Il Cluster

- Questo pattern viene usato quando si devono realizzare servizi verso un elevato numero di client.
  - Dato tale elevato numero un solo server non e' sufficiente per soddisfarli, e si desidera mantenere verso l'esterno un solo IP, Domain
  - Si risolve, realizzando cluster di server che virtualizzano servizi, presentando il servizio come unico mentre nella realtà viene fornito da svariati server.
- **Il cluster puo'** sfruttare svariate soluzioni tecnologiche anche complesse, fra queste:
  - Bilanciamento del carico delle richieste sui server che le evadono
  - Duplicazione dei database, per avere degli accessi concorrenti e per recuperare situazioni di fallimento
  - Server in fail over, copia calda, per recuperare delle situazioni critiche
  - Etc.

Web proxy server

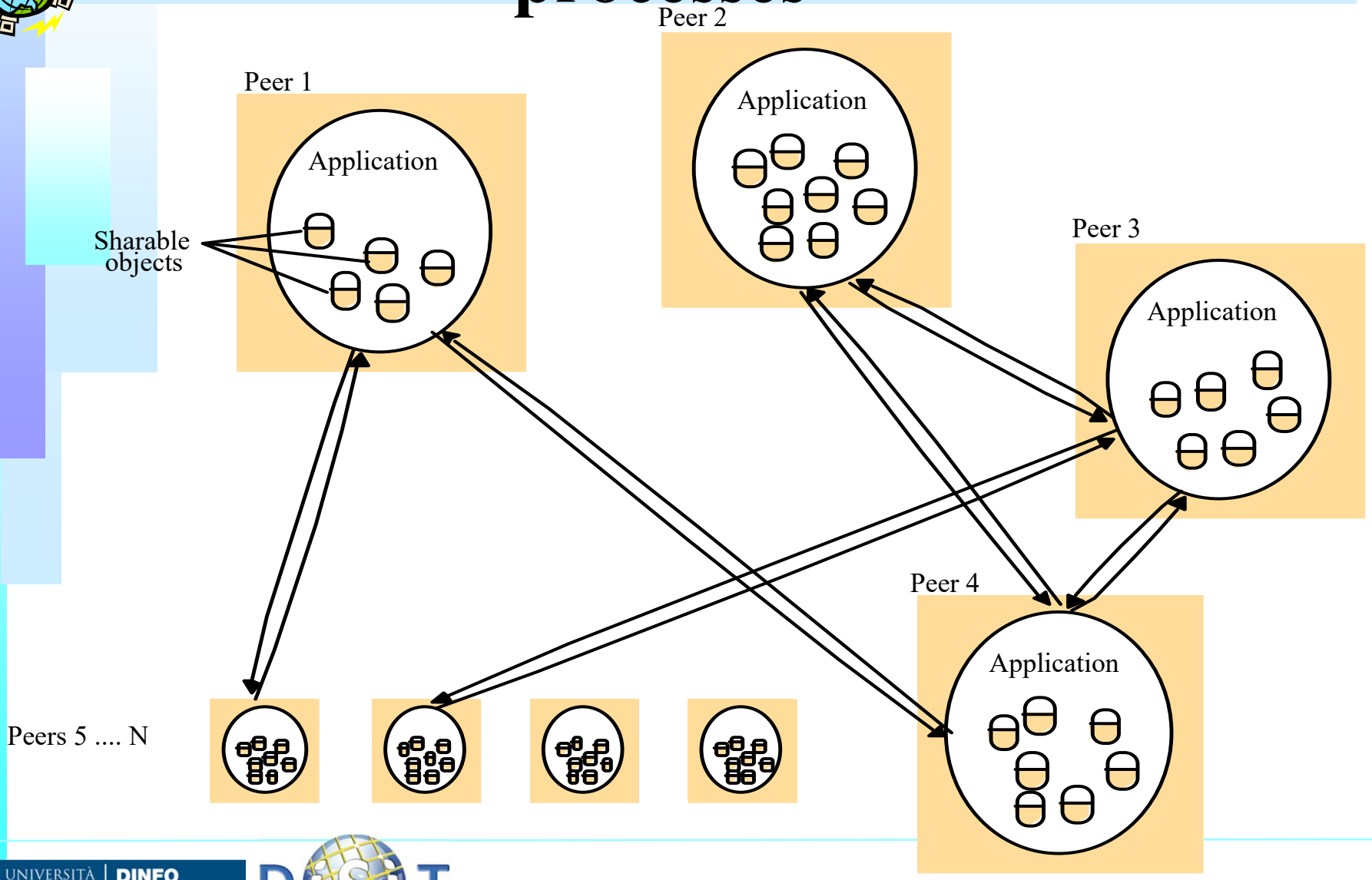


Proxy

- **Il concetto di Proxy** modella un pattern secondo il quale un server o agente intermedio viene utilizzato per accedere a servizi remoti anche molto complessi ed in numero elevato. Il client si riferisce al proxy per avere tale servizio e pertanto si presenta al servizio come se fosse un client diverso dall'originale.
  - Esempi sono i proxy per accesso HTTP
    - Proxy del browser,

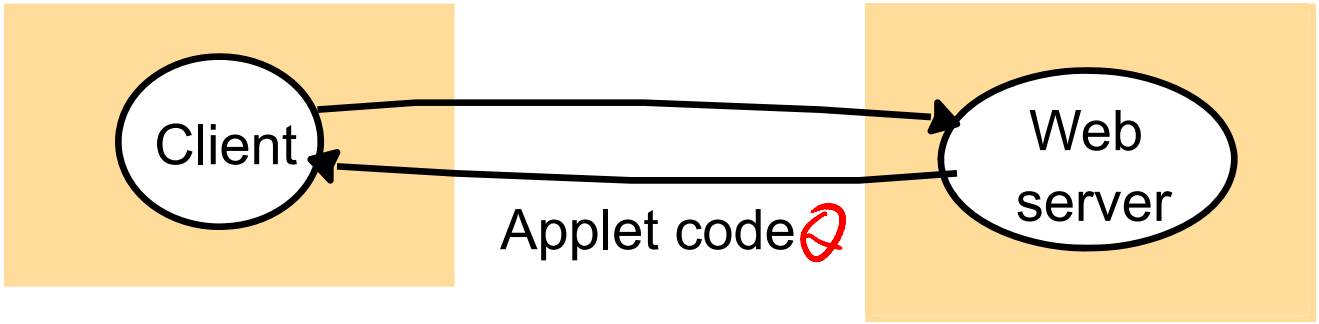
- **I server proxy** possono avere
  - Una memoria cache dei risultati delle richieste
  - Mascherano la complessità del sistema reale
    - Semplificano la gestione del server chiamante che non deve conoscere la complessità del sistema cluster server
  - non solo http protocol ma anche servizi complessi

A distributed application based on peer processes

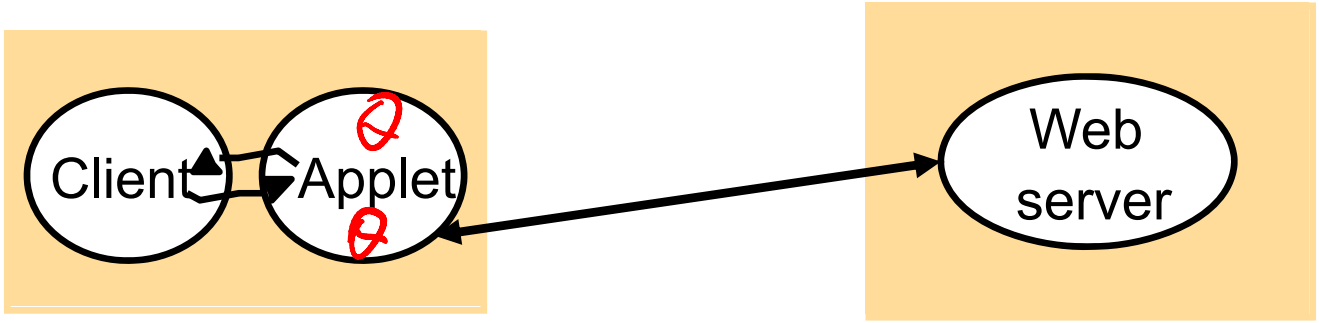


Web applets

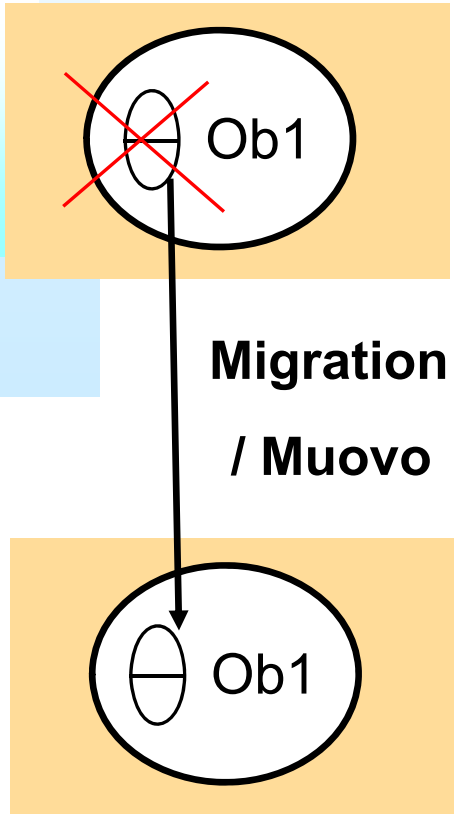
- a) client request results in the downloading of applet code

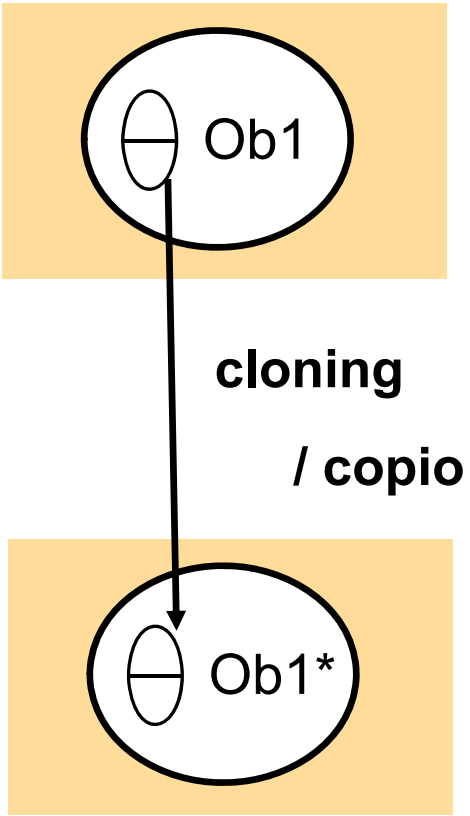


- b) client interacts with the applet



Migration and cloning

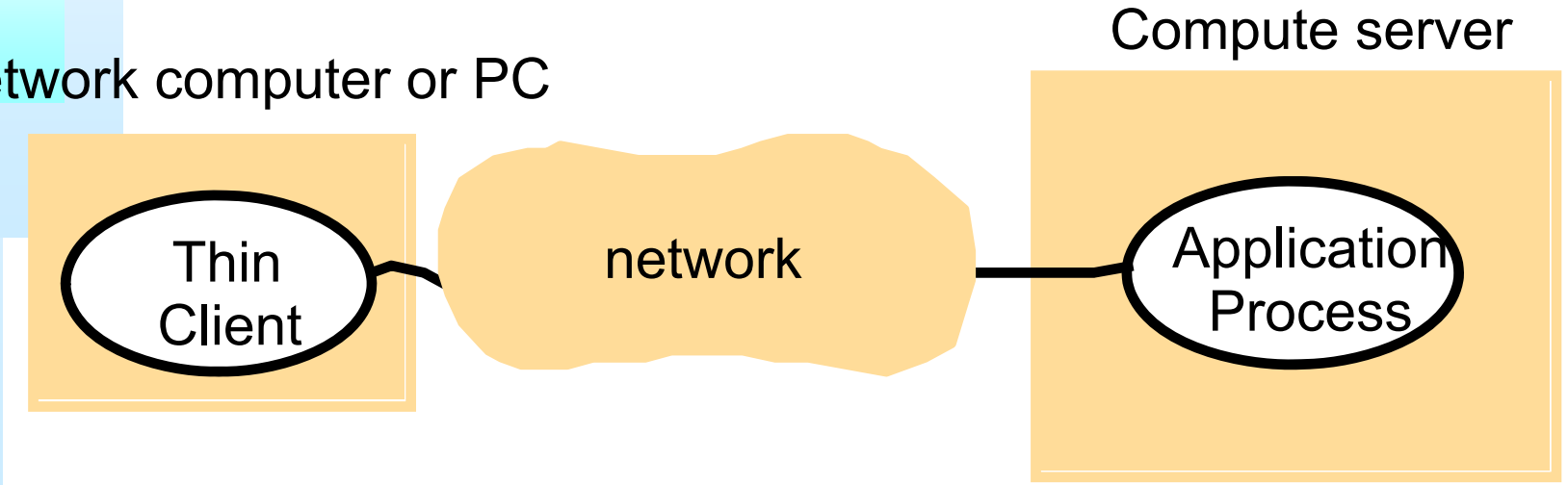




Mobile Agents

- **Un Agente e' running program**
  - può includere data e codice
- **Un Agente può** muoversi/migrare da un computer ad un altro nella rete per compiere un certo compito, per esempio:
  - Recuperare dei dati
  - Definire la struttura delle comunicazioni di rete
  - Fare dei conti
  - Etc.
- **Un Agente si può comportare in base ad un**
  - Approccio definito, basato su obiettivi prefissati:
    - ricerca dati, per esempio un crawler,
    - i worm sono degli agenti, etc.
  - Approccio dinamico in base a funzionali di costo che cerca di minimizzare, o strategie di vario tipo, …

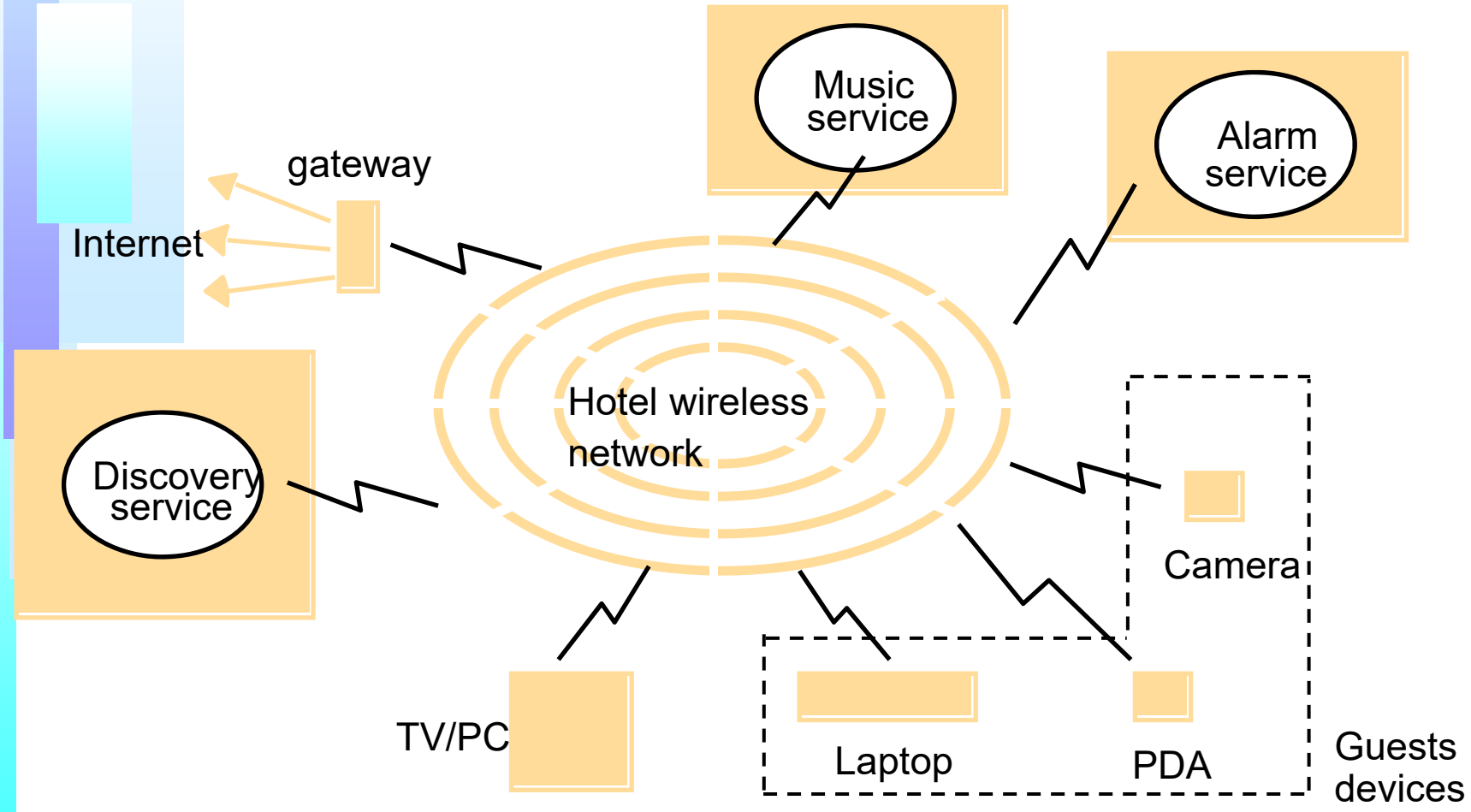
Thin clients and compute servers
X11, Citrix, Window Terminal Server



- VNC
- Radmin
- PCanyware

- Remote Desktop

Mobile networking, un esempio



Problemi progettuali dei Sistemi Distribuiti

- **Prestazioni, performance**
  - Responsiveness: reattività
  - Throughput: velocità di esecuzione, banda
    - Time critical application: e.g., streaming, WEB caching, proxy, etc.

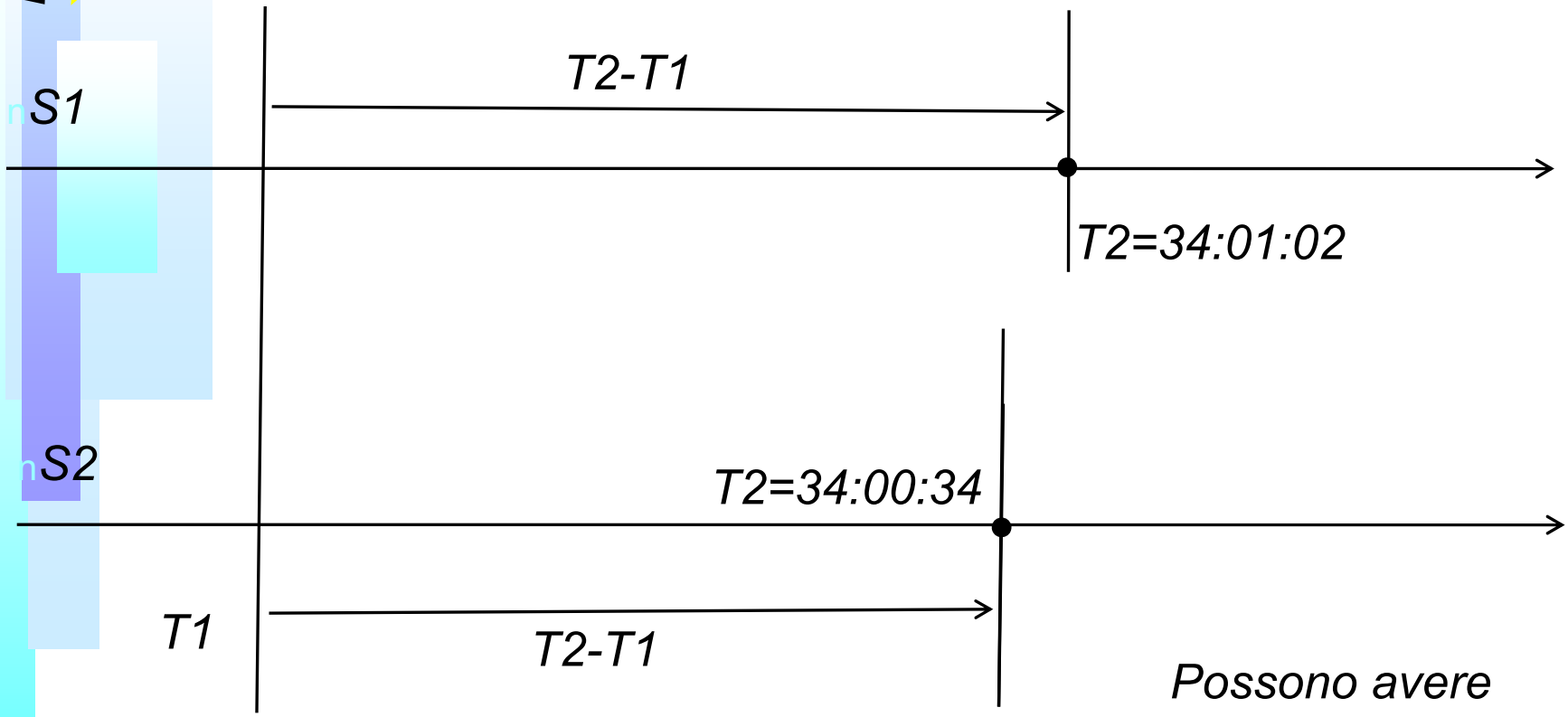
- **Quality of Service, QOS, qualità del servizio**
  - Reliability, affidabilità
  - Security, sicurezza delle informazioni
  - Performance, prestazioni
  - Adaptability, adattabilità a nuove funzioni e modelli

- **Dependability:**
  - Correctness+security+fault tolerance
  - Correttezza+sicurezza+tolleranza ai guasti

Modelli di Interazione per Sistemi Distribuiti

- **Sincroni**
  - i tempi di comunicazione sono vincolati da vincoli temporali
    - Temporal Constraint, TC,
  - Ogni messaggio/comando puo' avere un TC che definisce i tempi min e max di reazione
    - un TC=[Tmin, Tmax]
    - E.g.: esegui XX non prima di 3s ma entro 5s
  - Ogni processo ha un clock con una deriva nota (errore di sincronizzazione, drift, deriva)
- **Asincroni**
  - The execution time cannot be predicted,
    - da un ms a un anno ??. La comunicazione deve poter avvenire anche fra sistemi che hanno diverse velocità
    - il tempo di trasmissione da 0 a anni !!…
    - il Drift è arbitrario!

Differenze di tempo Assoluto



- *Due computer partono con time identico ma in evoluzione libera si trovano ad averlo diverso….time drift*

Possono avere

≠Start

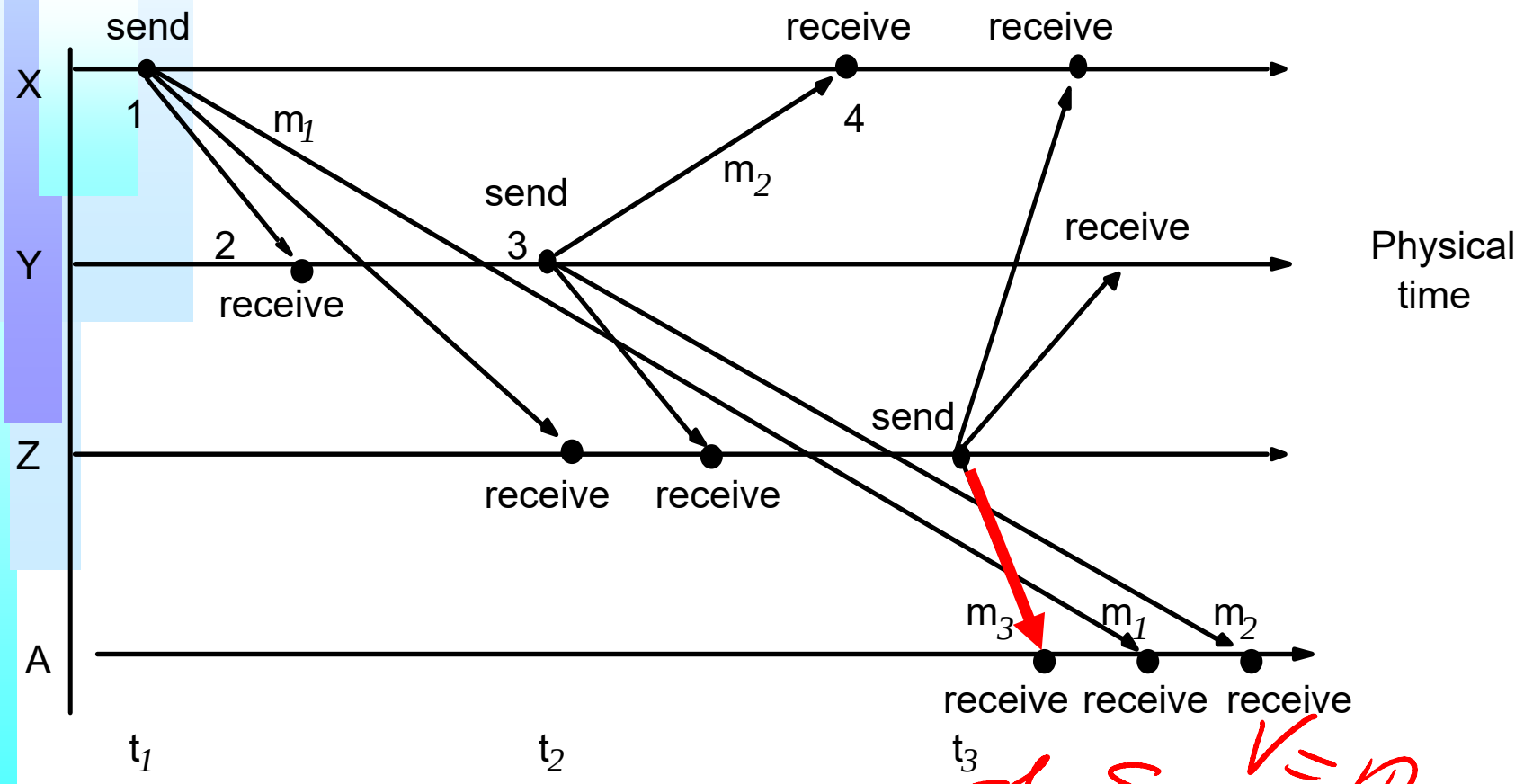
≠Stop

≠ Duration

Ordinamento di Eventi

- **Affidabilità**
  - garantire: ordinamento di eventi
  - garantire: tempi di consegna
  - produrre: protocolli robusti
- **Eventi ordinati**
  - Time stamp, time information associated with the event and information….
  - Implica avere un clock comune fra i vari sistemi inclusa la presenza di eventuali fusi orari
- **Problemi di Consistenza temporale**
  - causalita', convergenza, etc.

Problems of time ordering of events



Network performance

	Example	*Range*	*Bandwidth (Mbps)*	*Latency (ms)*
Wired:				
LAN	Ethernet	1-2 kms	10-1000	1-10
WAN	IP routing	worldwide	0.010-600	100-500
MAN	ATM	250 kms	1-150	10
Internetwork	Internet	worldwide	0.5-600	100-500
Wireless:				
WPAN	Bluetooth (802.15.1)	10 - 30m	0.5-2	5-20
WLAN	WiFi (IEEE 802.11)	0.15-1.5 km	2-54	5-20
WMAN	WiMAX (802.16)	550 km	1.5-20	5-20
WWAN	GSM, 3G phone nets	worldwide	0.01-02	100-500

IEEE 802 network standards

IEEE No.	*Name*	*Title*	*Reference*
802.3	Ethernet	CSMA/CD Networks (Ethernet)	[IEEE 1985a]
802.4		Token Bus Networks	[IEEE 1985b]
802.5		Token Ring Networks	[IEEE 1985c]
802.6		Metropolitan Area Networks	[IEEE 1994]
802.11	WiFi	Wireless Local Area Networks	[IEEE 1999]
802.15.1	Bluetooth	Wireless Personal Area Networks	[IEEE 2002]
802.15.4	ZigBee	Wireless Sensor Networks	[IEEE 2003]
802.16	WiMAX	Wireless Metropolitan Area Networks	[IEEE 2004a]

Ethernet ranges and speeds

			Cat.5	Cat.5e,6	Cat.7
	10Base5	*10BaseT*	*100BaseT*	*1000BaseT*	
Data rate	10 Mbps	10 Mbps	100 Mbps	1000 Mbps	10 Gbps
Max. segment lengths:					
Twisted wire (UTP)	100 m	100 m	100 m	25 m	10 m
Coaxial cable (STP)	500 m	500 m	500 m	25 m	
Multi-mode fibre	2000 m	2000 m	500 m	500 m	
Mono-mode fibre	25000 m	25000 m	20000 m	2000 m	

Failure Model

- Nei sistemi distribuiti la comunicazione fra processi può ovviamente fallire
  - Vi sono vari modelli di fallimento e varie cause

- I modelli sono principalmente:
  - **Omission Failures**:
    - il processo non effettua quello che ci sia aspetta da lui
    - Fail to start, fail to stop, etc..
    - Fail to send, fail to acknowledgment, etc.
  - **Bizantine Failure**:
    - fallimento arbitrario, set wrong value, return wrong values, etc.
  - **Timing Failure**:
    - too late, too early, too long durate, not expected at that time, etc.

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.

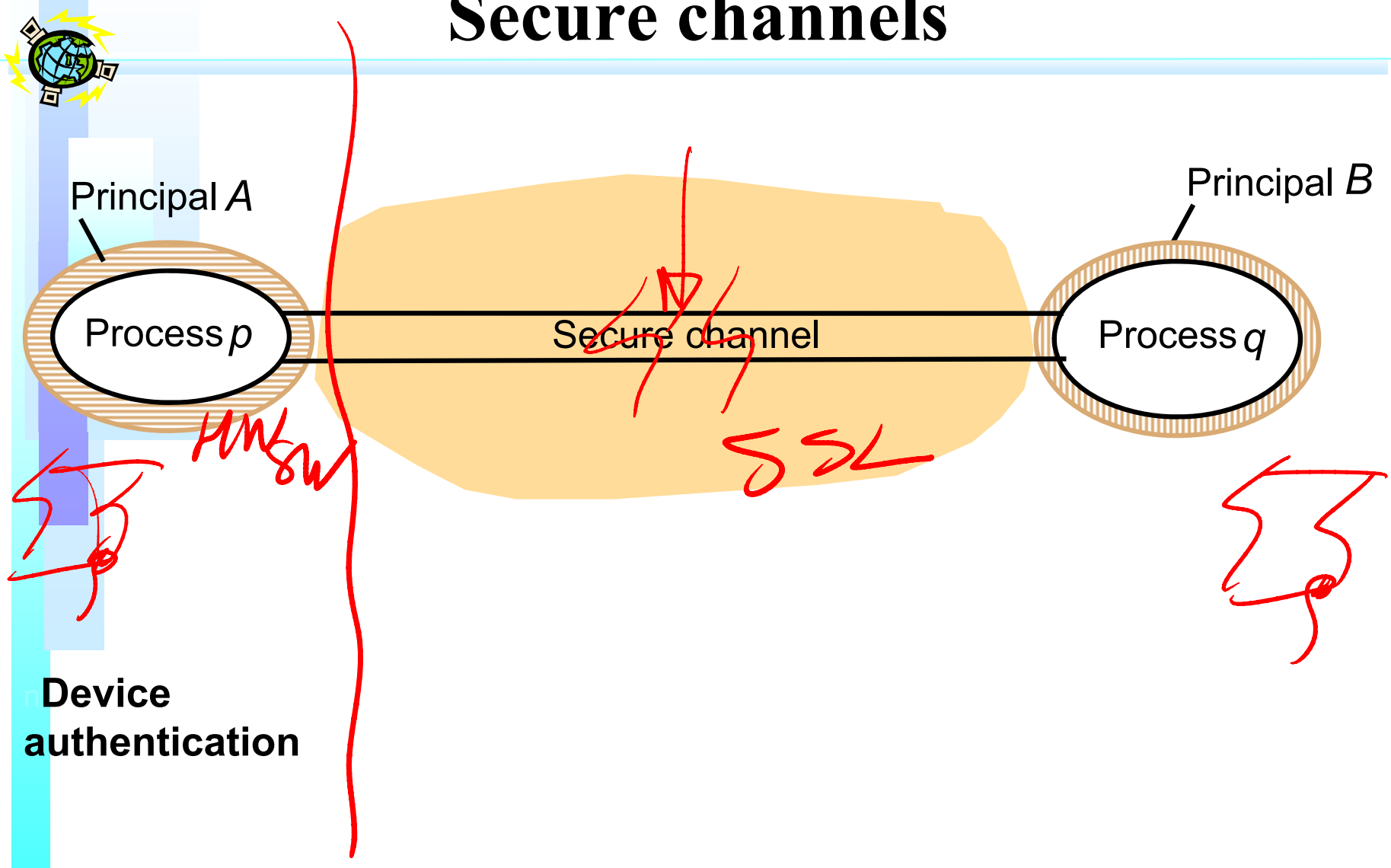
Security Model, controllo accesso

- **Authentication** of users
  - User ID and password, frequently saved as MD5 into database
  - User profiling with several information of the user
  - Production of certificates per la gestione delle chiavi

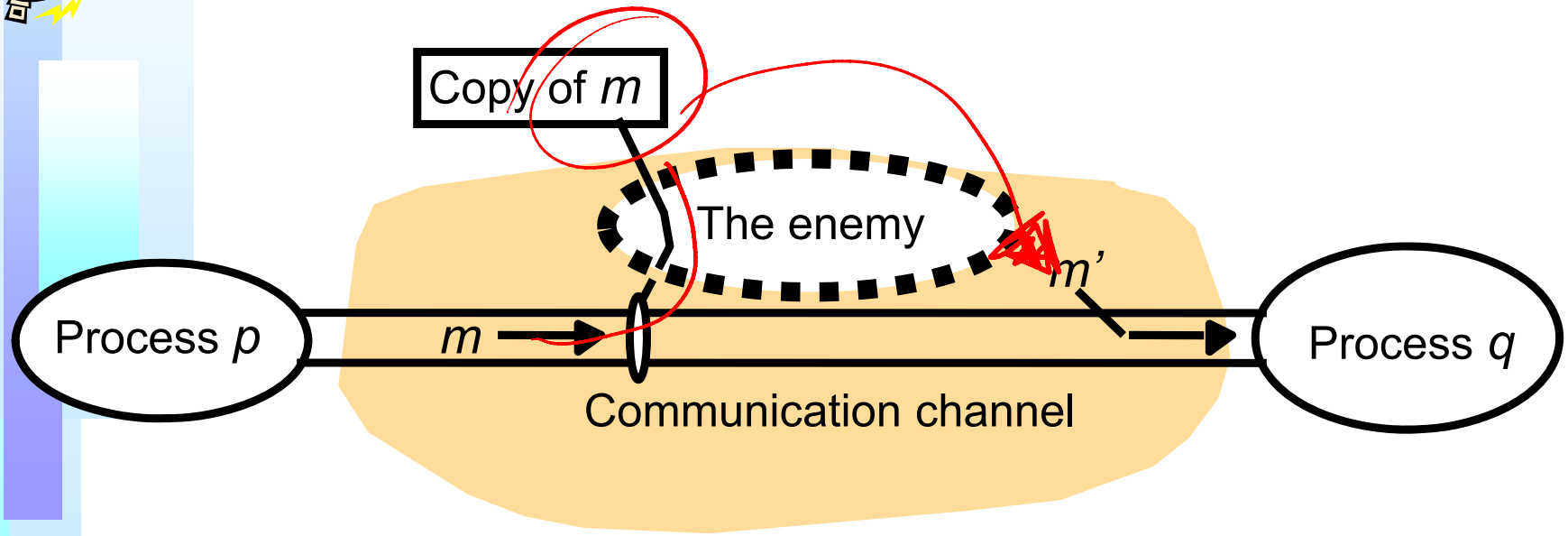
- **Certification** delle integrità di elementi come
  - devices (HW+SD), SW tools, etc.
  - data e content, dati e contenuti
  - Channel, canale

- **Protezione** di canale
  - Per esempio: SSL, HTTPS, SFTP
  - Certificati per la connessione

Secure channels



The Enemy, il nemico



- *Al momento della conncessione e creazione del canale protetto sia hanno i maggiori rischi*

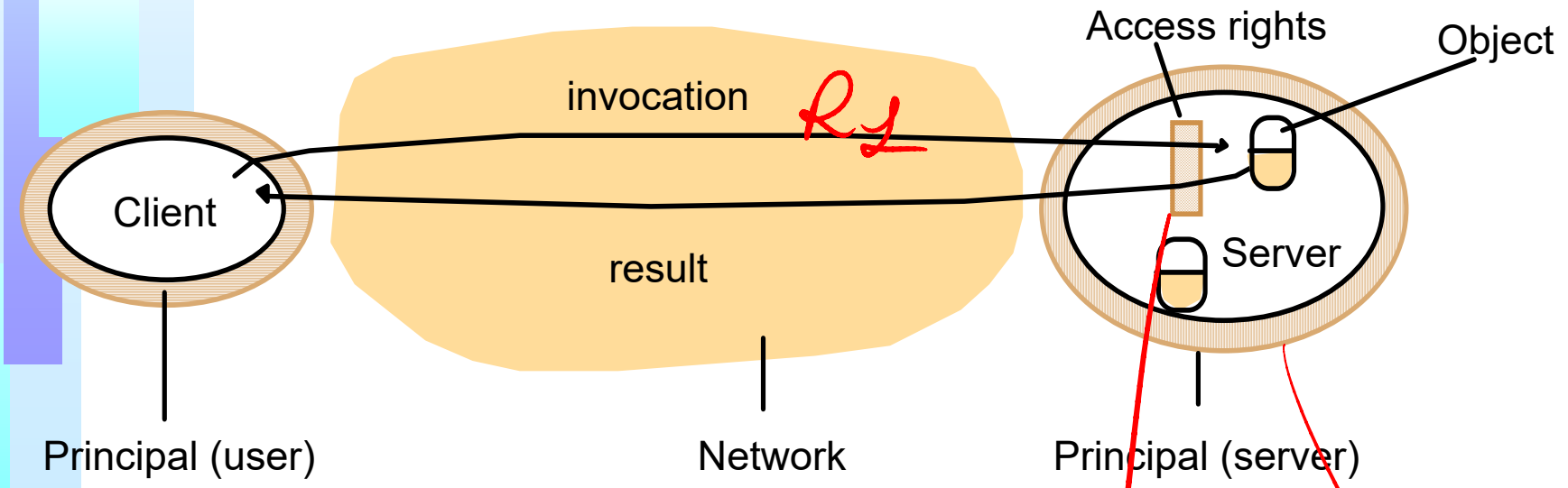
Security Model con diritti (rights)

- **Protezione** dei contenuti, degli oggetti:
  - Objects/components: software tools
  - Data: digital media, audiovisual, etc.

- **Controllo e prevenzione**
  - Controllo accesso all'oggetto:
    - → Conditional Access Systems, CAS
  - Controllo dei diritti per l'uso dei servizi degli oggetti
    - → Digital Rights Management, DRM

- **Sfruttamento controllato** di ogni loro caratteristica/ feature/ servizio
  - Conto corrente: leggo, muovo soldi, cambio parametri, etc.
  - Processo: start, stop, uso F1, uso F2, etc.
  - Contenuto digitale: play, print, copy, etc.

Objects and principals



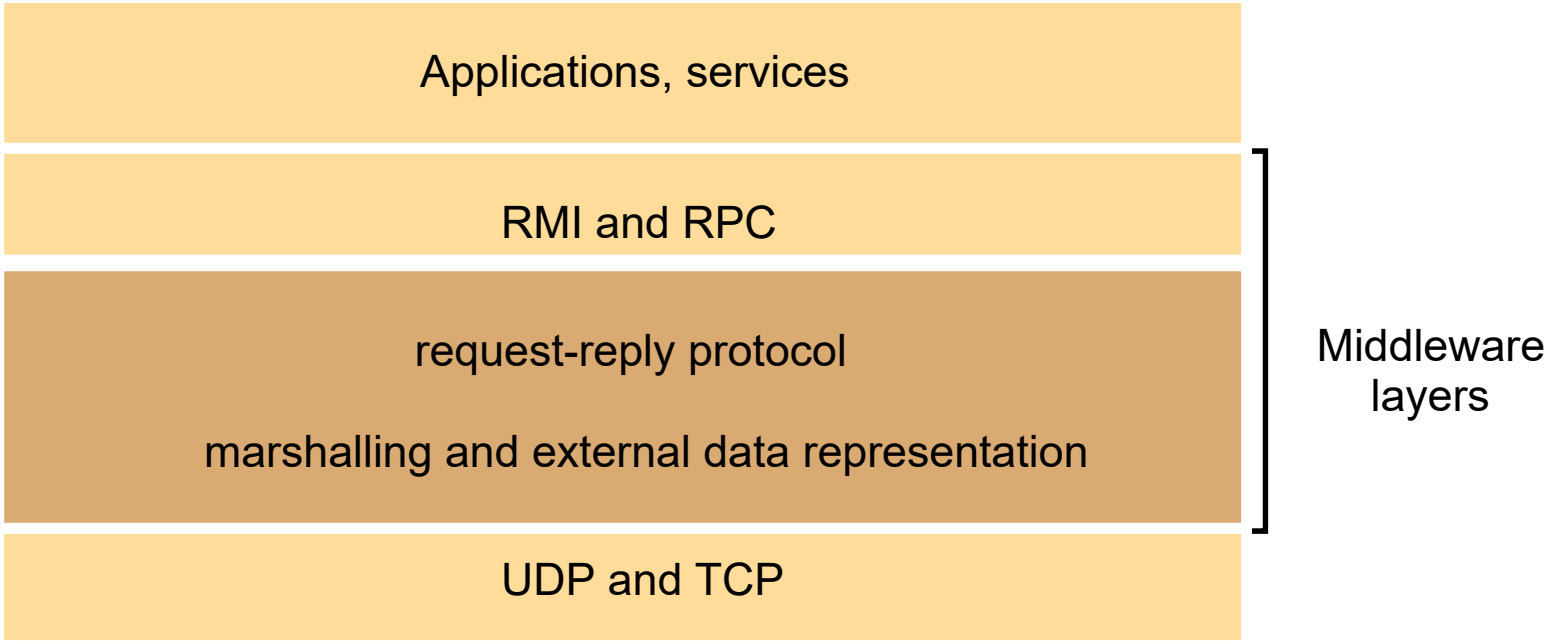
Controllo e supervisione dei diritti

- Nel corso di Sistemi Collaborativi e di Protezione, queste questioni saranno prese in considerazione in modo molto più dettagliato:
  - Modellli protezione
  - Controllo dei diritti
  - IPR: Intellectually Property Rights
  - DRM: Digital Rights Management
  - Distribuzione dei contenuti digitali
  - Distribuzione dei componenti software
  - DRM vari come: MS DRM, Apple iTune, MPEG-21, OMA, …
  - Etc.

Middleware

- Comunicazione fra processi, Livelli OSI
- Perché il Middleware
- Sockets and ports
- UDP e TCP
- Example: TCP communication
- RPC e RMI
- Data representation and coding for transmission
- MIME, Multipurpose Internet Mail Extensions

Comunicazione fra Processi
Middleware layers

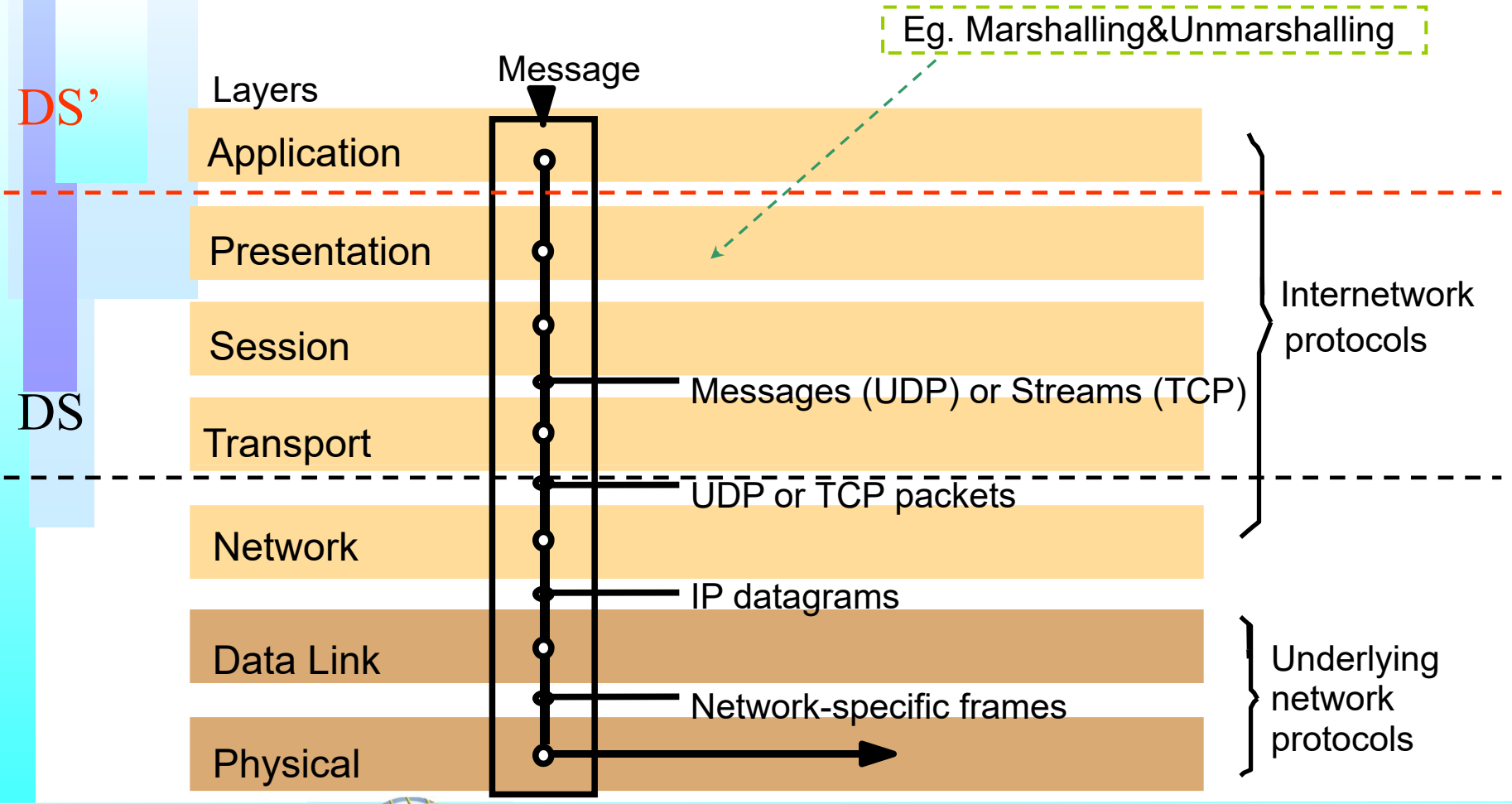


OSI protocol summary

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

Middleware Requirements

Network Communication



UDP or TCP

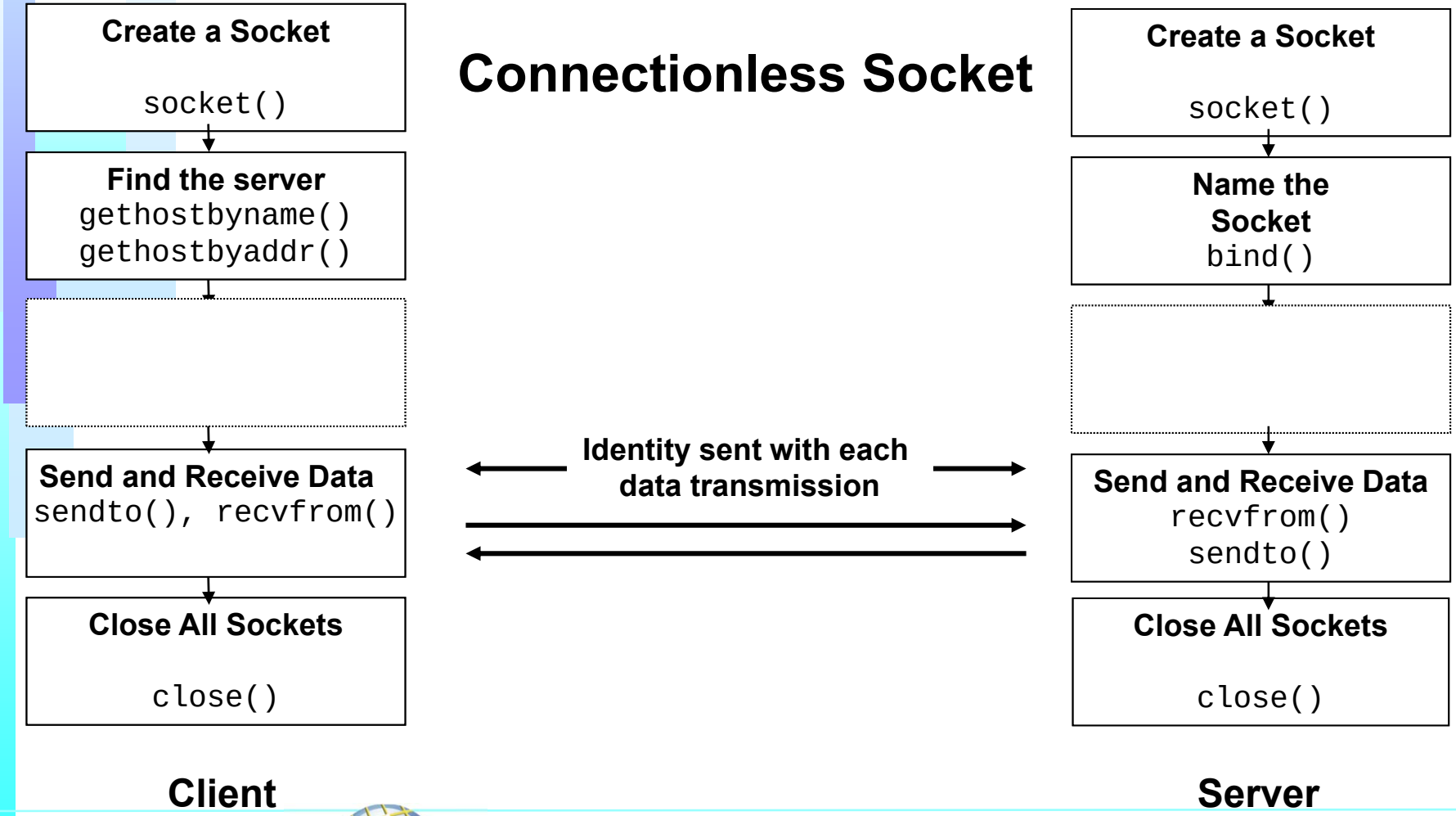
- **UDP messages**
  - Basata su messaggi fra processi
  - I singoli pacchetti vengono chiamati Datagram
  - Si identifica una Socket port

- **TCP Streams**
  - Si definisce una canale stream bidirezionale
  - I messaggi non hanno limitazioni di dimensioni, sono decomposti
  - Modello produttore-consumatore
  - Il consumatore deve aspettare
  - Si identifica due Socket port

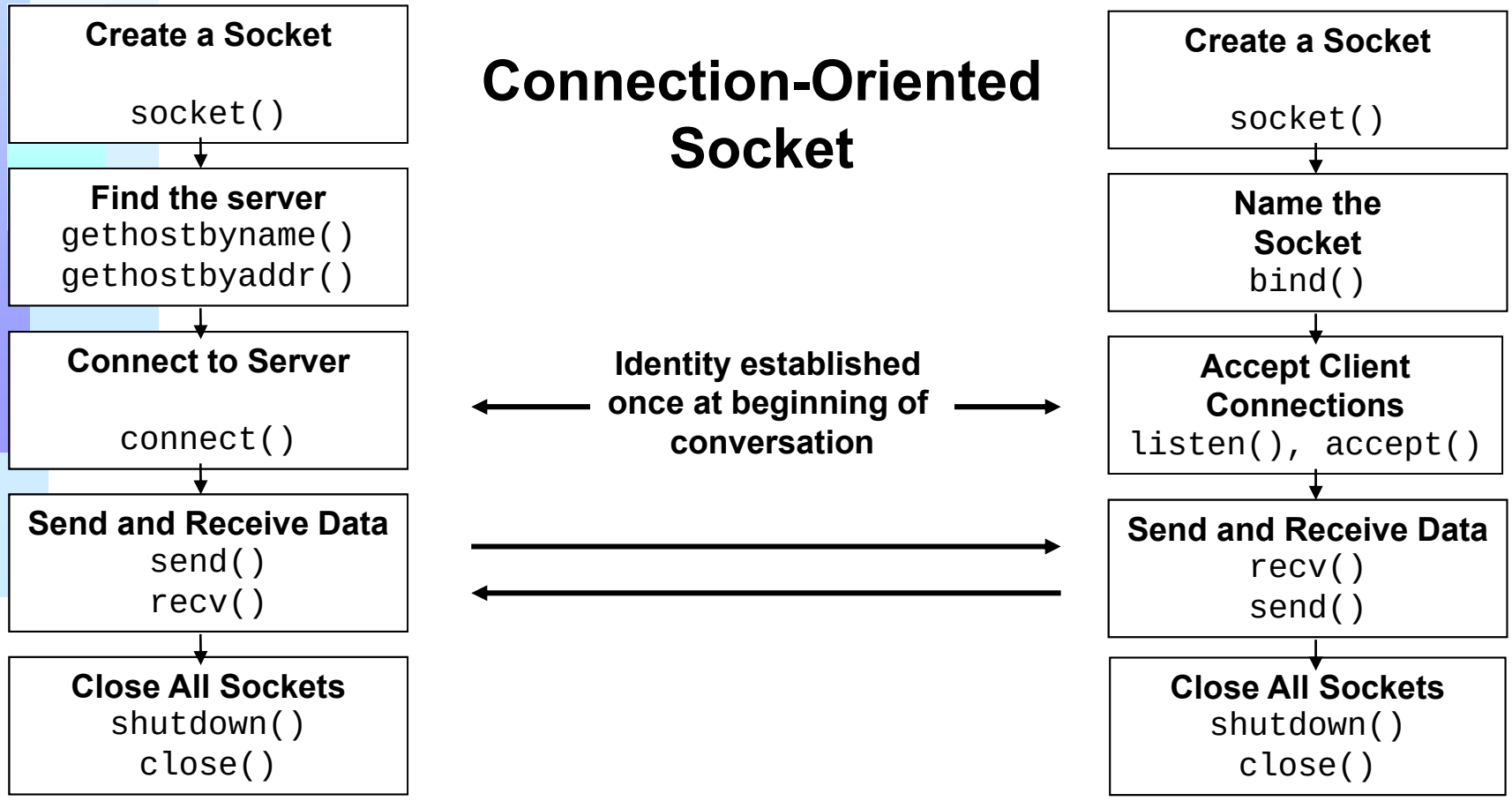
Introduction to sockets

- Two types of Internet domain sockets:
  - stream sockets (= TCP sockets)
    - provides communication in connected mode
    - guarantees reliable data transmission
    - costs time in connection set-up and error checking
  - datagram sockets (= UDP socket)
    - provides communication in connectionless mode
    - does not guarantee reliable data transmission
    - saves time from connection set-up and error checking

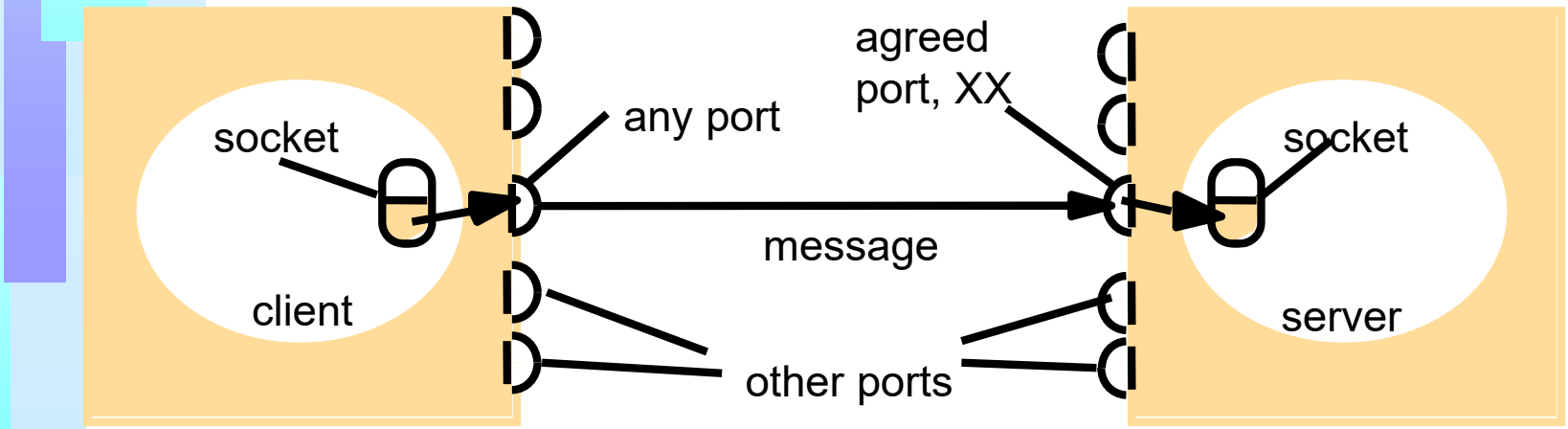
UDP


Connectionless Socket
Create a Socket
socket()
Find the server
gethostbyname()
gethostbyaddr()
Send and Receive Data
sendto(), recvfrom()
Close All Sockets
close()
Client
Identity sent with each data transmission
Create a Socket
socket()
Name the Socket
bind()
Send and Receive Data
recvfrom()
sendto()
Close All Sockets
close()
Server

TCP


Create a Socket
socket()
Find the server
gethostbyname()
gethostbyaddr()
Connect to Server
connect()
Send and Receive Data
send()
recv()
Close All Sockets
shutdown()
close()
Client
Connection-Oriented Socket
Identity established once at beginning of conversation
Create a Socket
socket()
Name the Socket
bind()
Accept Client Connections
listen(), accept()
Send and Receive Data
recv()
send()
Close All Sockets
shutdown()
close()
Server

Sockets and ports



TCP client makes connection to server, sends request and receives reply

```
import java.net.*;
import java.io.*;
public class TCPClient {
        public static void main (String args[]) {
        // arguments supply message and hostname of destination
        Socket s = null;
          try{
                int serverPort = 7896;
                s = new Socket(args[1], serverPort);
                DataInputStream in = new DataInputStream( s.getInputStream());
                DataOutputStream out =
                        new DataOutputStream( s.getOutputStream());
                out.writeUTF(args[0]);          // UTF is a string encoding see Sn 4.3
                String data = in.readUTF();
                System.out.println("Received: "+ data) ;
          }catch (UnknownHostException e){
                        System.out.println("Sock:"+e.getMessage());
          }catch (EOFException e){System.out.println("EOF:"+e.getMessage());
          }catch (IOException e){System.out.println("IO:"+e.getMessage());}
        }finally {if(s!=null) try {s.close();}catch (IOException
e){System.out.println("close:"+e.getMessage());}}
        }
}
```

TCP server makes a connection for each client and then echoes the client's request

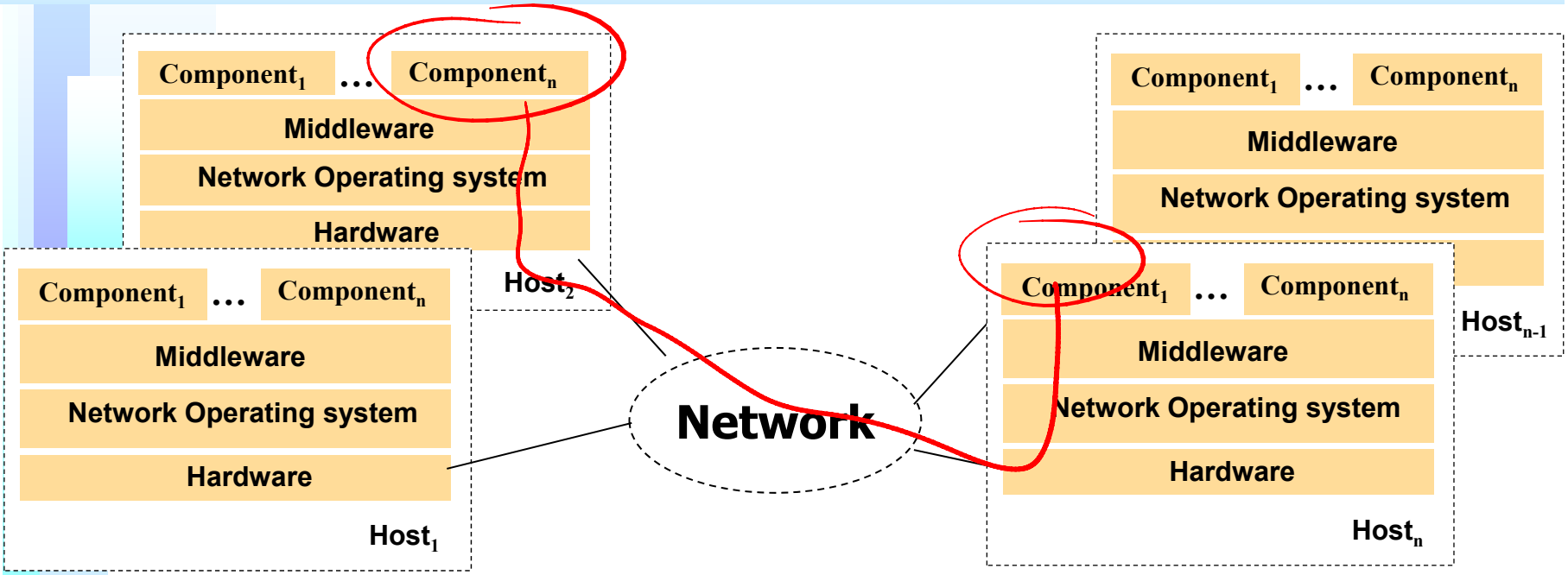
```
import java.net.*;
import java.io.*;
public class TCPServer {
    public static void main (String args[]) {
            try{
                    int serverPort = 7896;
                    ServerSocket listenSocket = new ServerSocket(serverPort);
                    while(true) {
                            Socket clientSocket = listenSocket.accept();
                            Connection c = new Connection(clientSocket);
                    }
            } catch(IOException e) {System.out.println("Listen :"+e.getMessage());}
    }
}

// this figure continues on the next slide
```

continued

```
class Connection extends Thread {
        DataInputStream in;
        DataOutputStream out;
        Socket clientSocket;
        public Connection (Socket aClientSocket) {
           try {
                clientSocket = aClientSocket;
                in = new DataInputStream( clientSocket.getInputStream());
                out =new DataOutputStream( clientSocket.getOutputStream());
                this.start();
            } catch(IOException e)  {System.out.println("Connection:"+e.getMessage());}
        }
        public void run(){
           try {                                        // an echo server
                String data = in.readUTF();
                out.writeUTF(data);
           } catch(EOFException e) {System.out.println("EOF:"+e.getMessage());
           } catch(IOException e) {System.out.println("IO:"+e.getMessage());}
           } finally{ try {clientSocket.close();}catch (IOException e){/*close failed*/}}
        }
}
```

Why use Middleware?



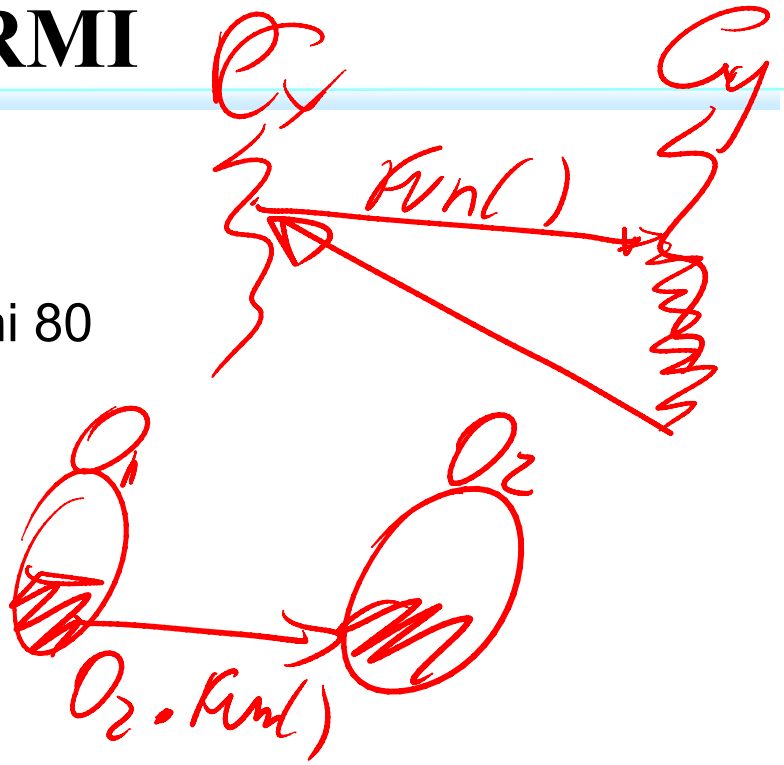
Middleware in Distributed System Construction

Think...

- Using Middleware to build DS is comparable to that of using DBMS when building IS.

RPC and RMI

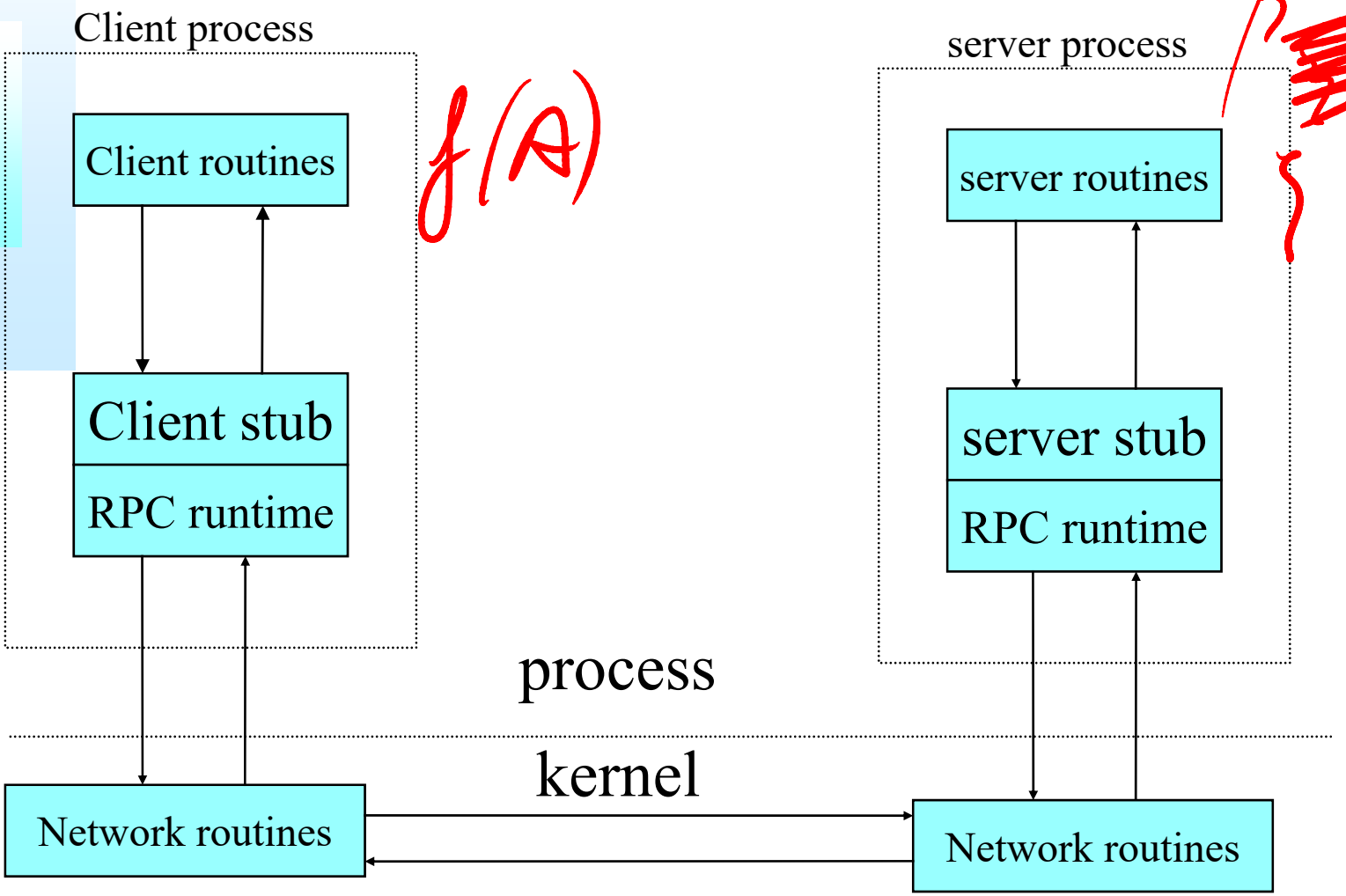
- **Remote procedure call, RPC**
  - Procedural Models
  - Introdotto da Sun agli inizi degli anni 80
- **Remote Method Invocation, RMI**
  - Object Oriented Context
  - Corba or Java RMI
- **Necessario garantire**
  - Affidabilità
  - Ordinamento dei messaggi



Elements of RPC

- Client process
- Server process
- Gli Stub:
  - **Client**: packages the data input to the remote procedures and un-packages the data output from the remote procedures
  - **Server**: un-packages the data received from client, and packages the return value before sending to client
- RPC Runtime
  - library routines
- Network communication:
  - Via data marshalling and un-marshalling
  - Via RPC language

Overview of Sun RPC



XDR: eXternal Data Representation

- **Standard for eXternal Data Representation**
  - Sun RPC servers transmit data in this representation,
  - it can be in XML, or ASCII.. etc..
- **Marshalling**:
  - Any internal data structure is packaged into XDR before sending, this is done by the client stub
- **Un-marshalling**:
  - Any received external data structure in XDR is unpackaged into local representation by the server stub

Data Representation and coding for Communication

- **Micro processori, CPU, calcolatori hanno modelli diversi per la rappresentazione dei dati nella memoria fisica: per esempio riguardo all'ordinamento dei nibble:**
  - Intel – Motorola ← → Little endian --- big endian
  - Rappresentazioni in virgola mobile
  - Numero dei bit per rappresentare un integer
  - Etc.
- **I meccanismi di marshalling e unmarshalling**
  - Prendono i dati e li trasformano avanti / indietro per essere inviati in uno o piu' pacchetti
  - Questo lavoro viene effettuato dal layer middleware / stubs
  - Portare tutto a livello ASCII
  - Necessitano di linguaggi standard per la formalizzazione dei dati

CORBA CDR (Common Data Representation) for complex types

Type	*Representation*
sequence	length (unsigned long) followed by elements in order
string	length (unsigned long) followed by characters in order (can also can have wide characters)
array	array elements in order (no length specified because it is fixed)
struct	in the order of declaration of the components
enumerated	unsigned long (the values are specified by the order declared)
union	type tag followed by the selected member

- *15 basic types: short, long, float, etc…*
- *Plus the above components*
- *Common Object Request Broker Architecture*
- *Marshalling performed into the CORBA Support*

CORBA CDR message

index in sequence of bytes	← 4 bytes →	*notes on representation*
0–3	5	*length of string*
4–7	"Smit"	*'Smith'*
8–11	"h "	
12–15	6	*length of string*
16–19	"Lond"	*'London'*
20-23	"on "	
24–27	1934	*unsigned long*

The flattened form represents a *Person* struct with value: {'Smith', 'London', 1934}

Indication of Java serialized form

Serialized values				*Explanation*
Person	8-byte version number		h0	*class name, version number*
3	int year	java.lang.String name:	java.lang.String place:	*number, type and name of instance variables*
1934	5 Smith	6 London	h1	*values of instance variables*

The true **serialized** form contains additional type markers; h0 and h1 are handles

Recently: Person structure

```
<person id="123456789">
        <name>Smith</name>
        <place>London</place>
        <year>1934</year>
        <!-- a comment -->
</person >
```

Per delle slide sull'XML si consulti le pagine web del corso di sistemi distribuiti

Illustration of the use of a namespace in the *Person* structure

```
<person pers:id="123456789"
     xmlns:pers = "http://www.cdk4.net/person">
    <pers:name> Smith </pers:name>
    <pers:place> London </pers:place >
    <pers:year> 1934 </pers:year>
</person>
```

An XML schema for the Person structure

```
<xsd:schema  xmlns:xsd = URL of XML schema definitions   >
   <xsd:element name= "person" type ="personType" />
      <xsd:complexType name="personType">
         <xsd:sequence>
            <xsd:element name = "name"  type="xs:string"/>
            <xsd:element name = "place"  type="xs:string"/>
            <xsd:element name = "year"  type="xs:positiveInteger"/>
         </xsd:sequence>
         <xsd:attribute name= "id"   type = "xs:positiveInteger"/>
      </xsd:complexType>
</xsd:schema>
```

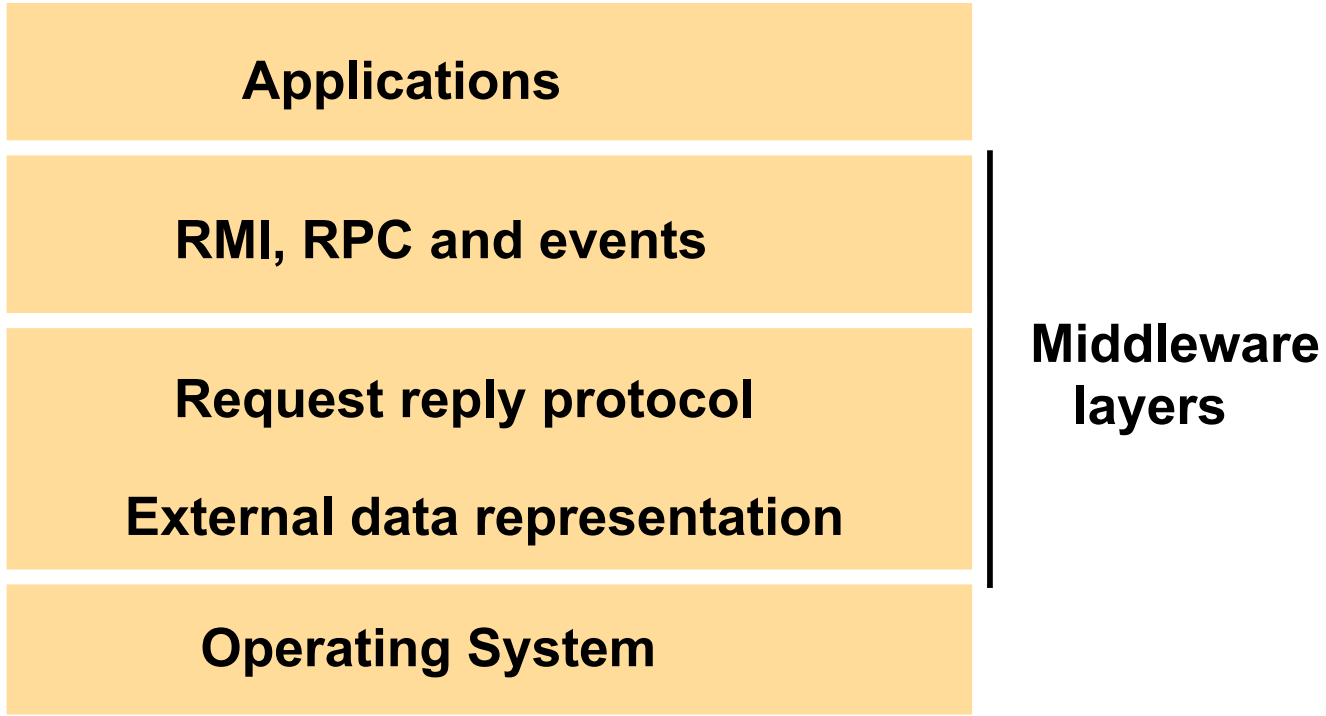
MIME type for file types

- **Multipurpose Internet Mail Extension**
  - Use for sending binary and files in the emails
  - Classificazione delle estensioni
- **MIME Type**
  - Struttura: Type/subtype
  - E.g.: text/doc, text/html, image/gif, image/jpeg
  - il sistema operativo presenta API per la loro gestione
  - Ad ogni tipo vi sono associate delle funzionalità e dei programmi eseguibili, opzioni di Explorer
    - ➔Le operazioni che il SO puo' chiedere a tali programmi per quei tipi di dati: Open, Print, copy, etc.
  - Informazioni/tabelle contenute nel Registry

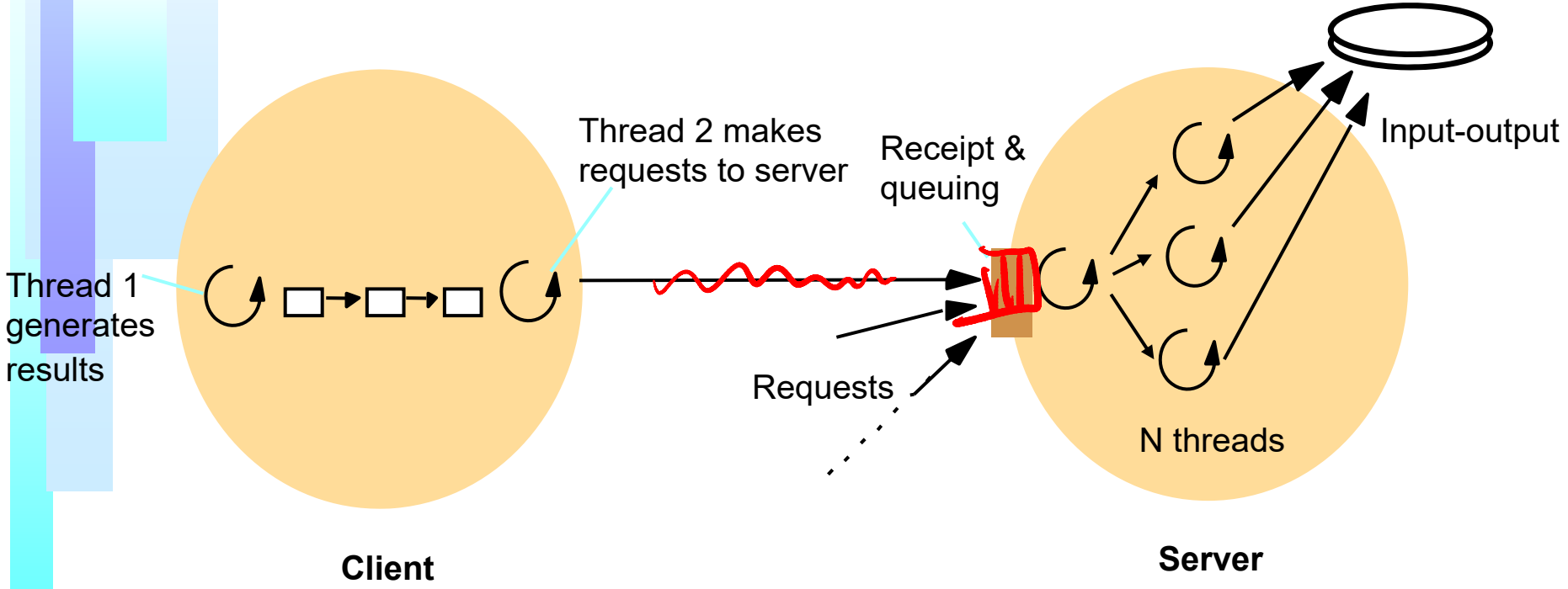
Call Remote

- Invocazioni Remote
- Interfacce, IDL
- Remote Procedure Call
- CORBA IDL
- Modello ad oggetti di sistemi distribuiti
- Oggetti remoti ed interfacce
- Comunicazione fra oggetti, RMI

Oggetti Distribuiti e Invocazioni Remote

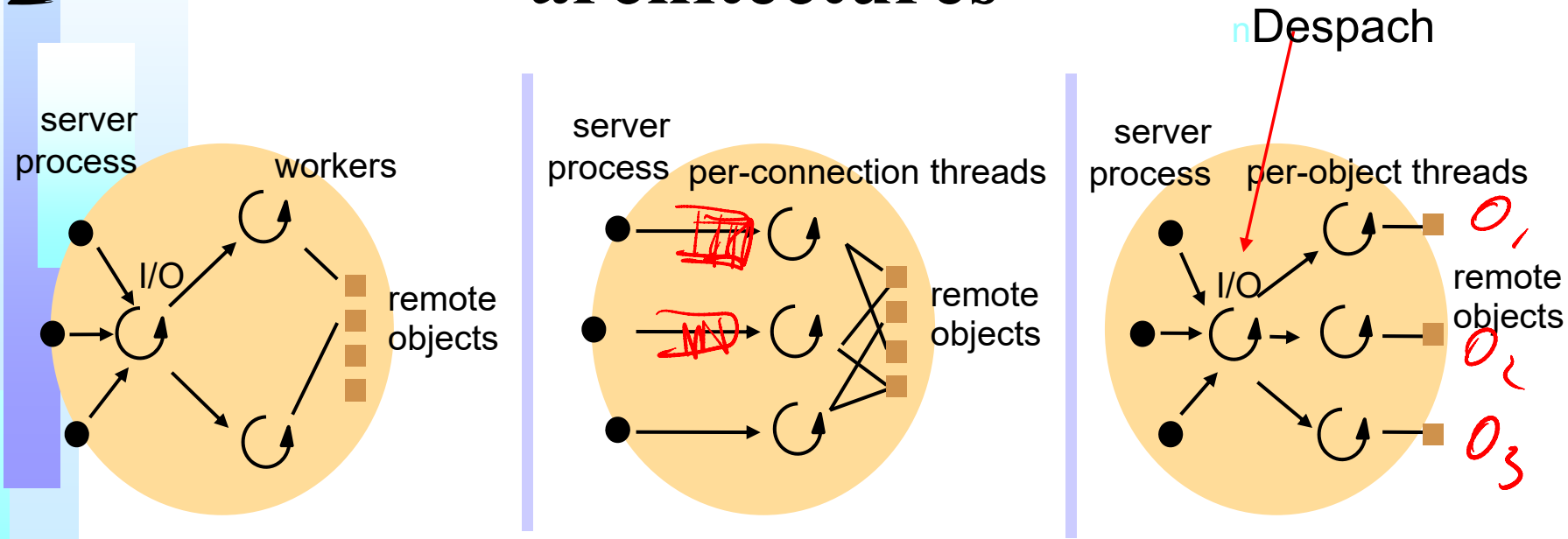


Client and server with threads



The 'worker pool' architecture

Alternative server threading architectures



a. Thread-per-request b. Thread-per-connection c. Thread-per-object

- Implemented by the server-side ORB in CORBA
- (a) would be useful for UDP-based service, e.g. NTP
- (b) is the most commonly used - matches the TCP connection model
- (c) is used where the service is encapsulated as an object. E.g. could have multiple shared whiteboards with one thread each. Each object has only one thread, avoiding the need for thread synchronization within objects.

Java thread constructor and management methods

Methods of objects that inherit from class Thread

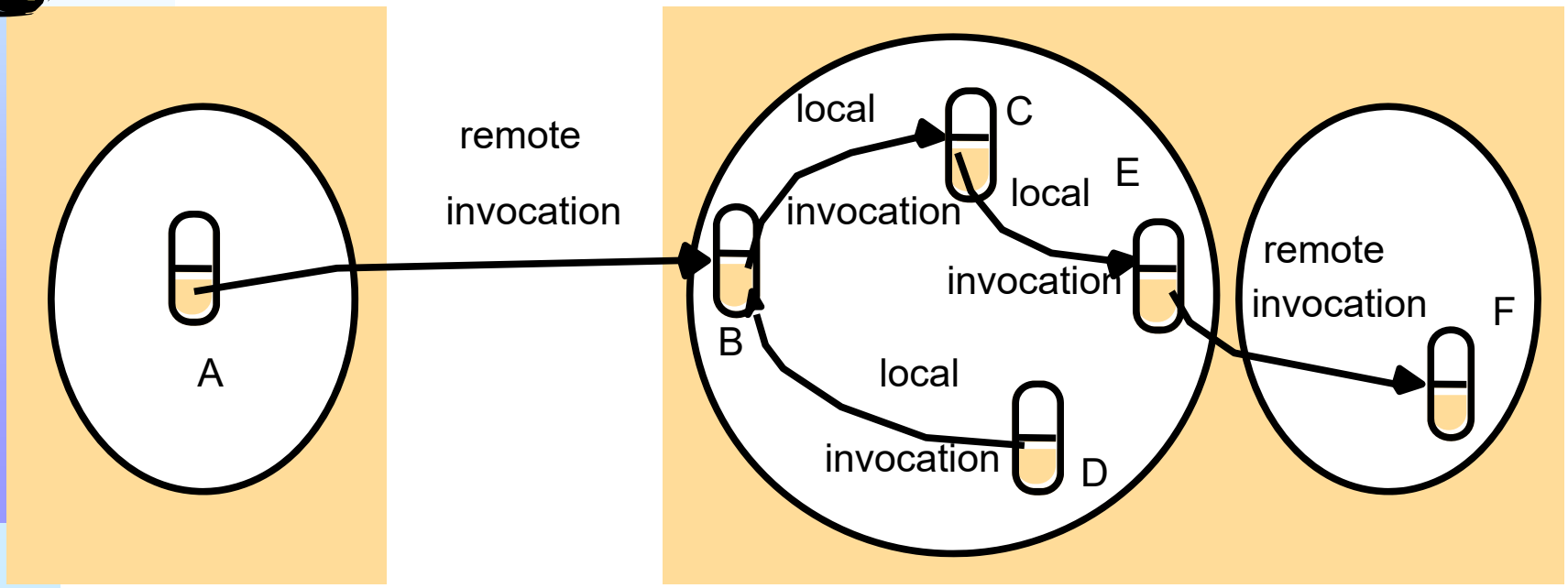
- ***Thread(ThreadGroup group, Runnable target, String name)***
  - Creates a new thread in the *SUSPENDED* state, which will belong to *group* and be identified as *name*; the thread will execute the *run()* method of *target*.
- ***setPriority(int newPriority), getPriority()***
  - Set and return the thread's priority.
- ***run()***
  - A thread executes the *run()* method of its target object, if it has one, and otherwise its own *run()* method (*Thread* implements *Runnable*).
- ***start()***
  - Change the state of the thread from *SUSPENDED* to *RUNNABLE*.
- ***sleep(int millisecs)***
  - Cause the thread to enter the *SUSPENDED* state for the specified time.
- ***yield()***
  - Enter the *READY* state and invoke the scheduler.
- ***destroy()***
  - Destroy the thread.

The Middleware

- **Middleware is a software layer that provides**
  - a programming model above the basic building blocks of processes and message passing
  - location transparency
  - independence from the details of communication protocols
  - independence on the underlying transport protocols
  - independent on the low level coding of types

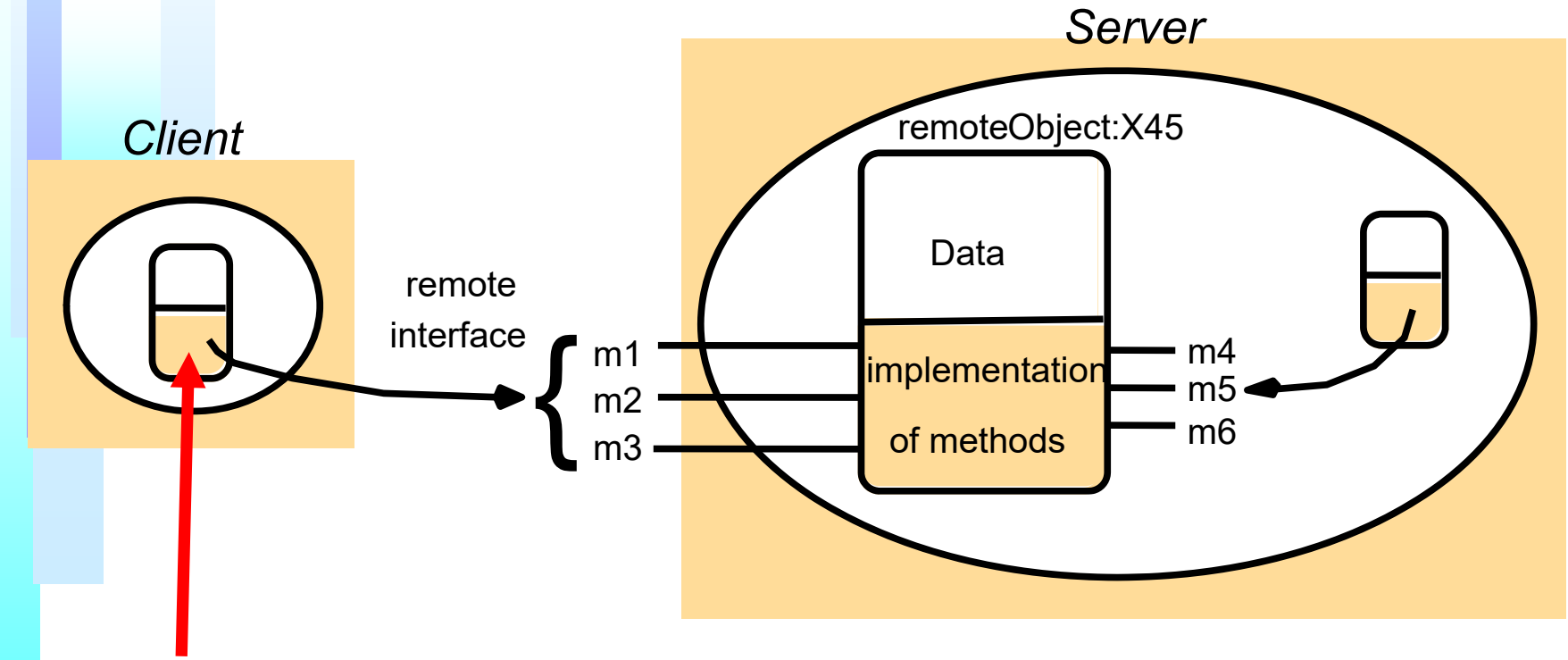
- Gestione statica e dinamica di
  - Remote Object Reference
  - Remote Objects
  - Remote Interfaces

Remote and local method invocations



- each process contains objects, some of which can receive remote invocations, others only local invocations
- those that can receive remote invocations are called *remote objects*
- objects need to know the ***remote object reference*** of an object in another process in order to invoke its methods. How do they get it?
- the ***remote interface*** specifies which methods can be invoked remotely

A remote object and its remote interface



- *Conosce il Remote object reference X45*

Descrizione delle Interfacce

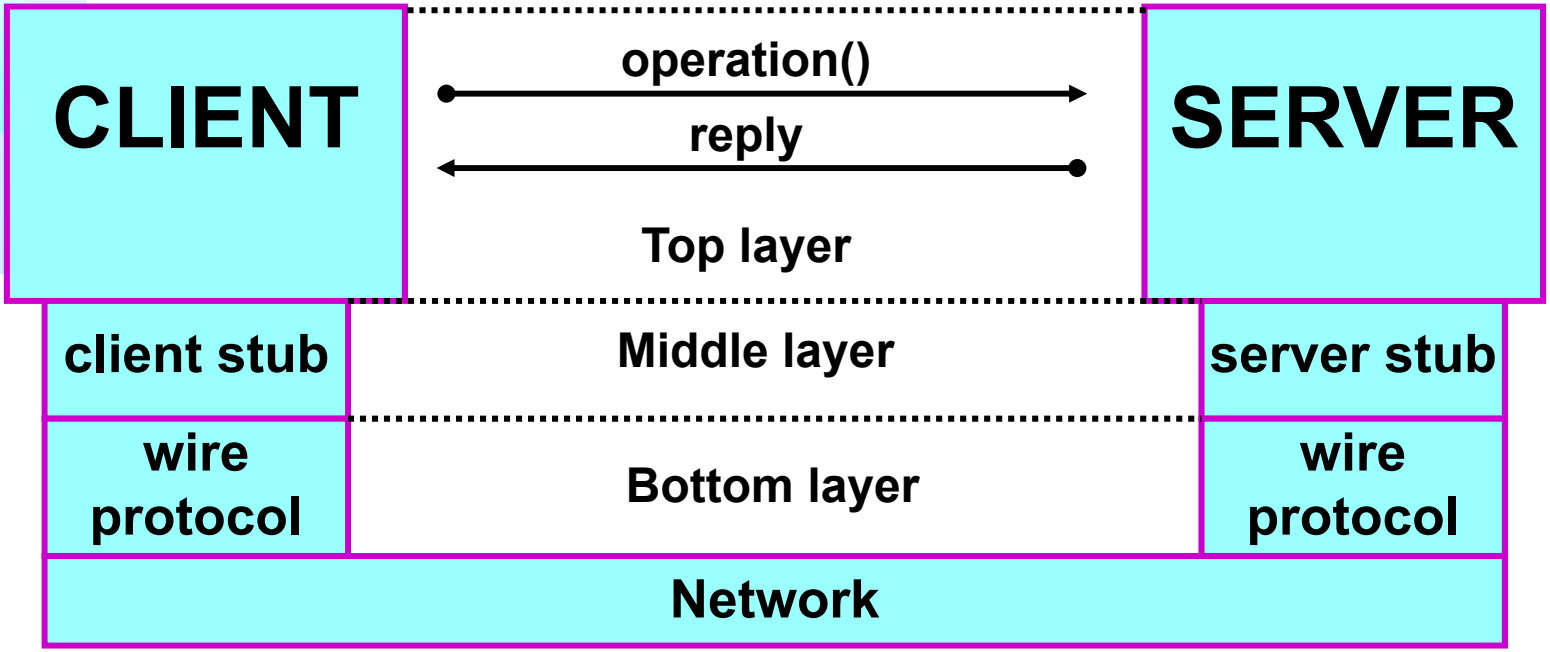
- RMI: Remote Method Invocation
- RCP: Remote Procedure Call

- L'interfaccia di comunicazione deve essere pubblica:
  - Nome dei metodi, tipo, ordine e numero di parametri della RPC/RMI, in/out parameter
- La descrizione dell'interfaccia puo' essere utile per la generazione automatica del contesto di Marshalling
- Esistono linguaggi per la descrizione delle interfacce,
  - Per esempio: IDL, Interface Description Language
- Lo stesso oggetto puo' avere diverse interfacce, una pubblica, una per l'interno…
- Oggetti che permettono RMI vengono detti oggetti distribuiti

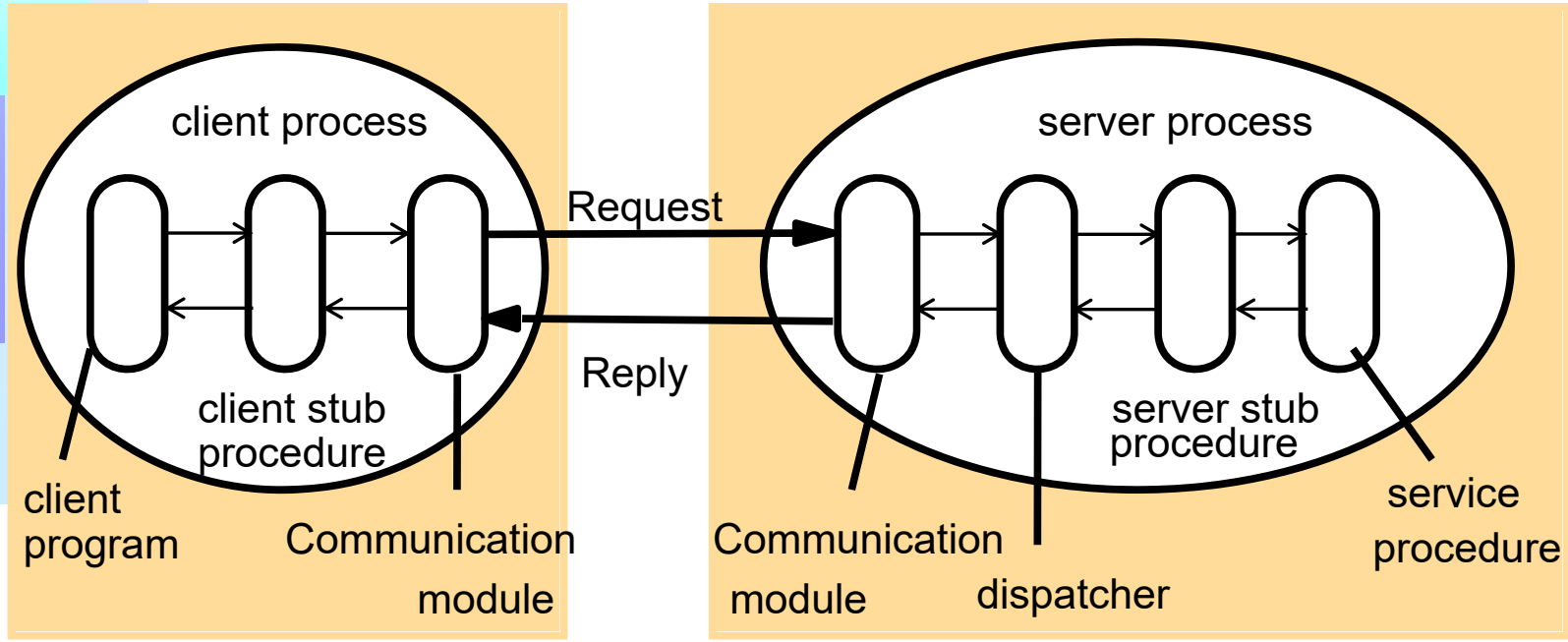
IDL, Interface Description/Definition Language

- Sun XDR has IDL for RPC
  - Obsolete
- DCOM (Distributed Common Object Model)
  - IDL based on DCE (Distributed Computing Environment) for RPC
  - Microsoft COM and OLE, alla base degli ActiveX…..
- CORBA IDL, RMI
- Java RMI, RMI
- .NET, etc…

Remote Procedure Call



Role of Client & Server stub procedures in RPC



Files interface in Sun XDR, RPC

```
const MAX = 1000;
typedef int FileIdentifier;
typedef int FilePointer;
typedef int Length;
struct Data {
    int length;
    char buffer[MAX];
};
struct writeargs {
    FileIdentifier f;
    FilePointer position;
    Data data;
};
```

```
struct readargs {
    FileIdentifier f;
    FilePointer position;
    Length length;
};

program FILEREADWRITE {
  version VERSION {
    void WRITE(writeargs)=1;
    Data READ(readargs)=2;
  }=2;
} = 9999;
```

CORBA IDL example

- *Interface Definition Language*
- *Data hiding, utilizzato in OO*
- *L'interfaccia è il solo modo di accedere all'informazione*
- *Formalizzazione della chiamata con parametri di In/out*
- *L'interfaccia trova la sua implementazione nella classe*

```
// In file Person.idl
struct Person {
        string name;
        string place;
        long year;
    } ;

interface PersonList {
        readonly attribute string listname;
        void addPerson(in Person p) ;
        void getPerson(in string name, out Person p);
        long getyear();
    };
```

- *Parametri per valore nelle due direzioni*
- *puntatori non hanno senso, i processi sono diversi, la memoria è diversa*
- *Reference: in alcuni casi esistono reference assoluti del sistema distribuito che si possono usare come OBJECT-ID*

Object Model

- **Object References**
  - Java and C++, possono essere passati
- **Interfaces**
  - Insieme di metodi con la loro signature (tipo e ordine dei parametri incluso quello di ritorno)
  - una classe puo' implementare diverse interfacce
  - L'interfaccia è considerata anche un tipo
- **Actions**
  - Per cambiare lo stato
  - Per attivare altre azioni, chiamare altri metodi
- **Exceptions**
  - Throw-catches
- **Garbage Collection**
  - Singolo o distribuito

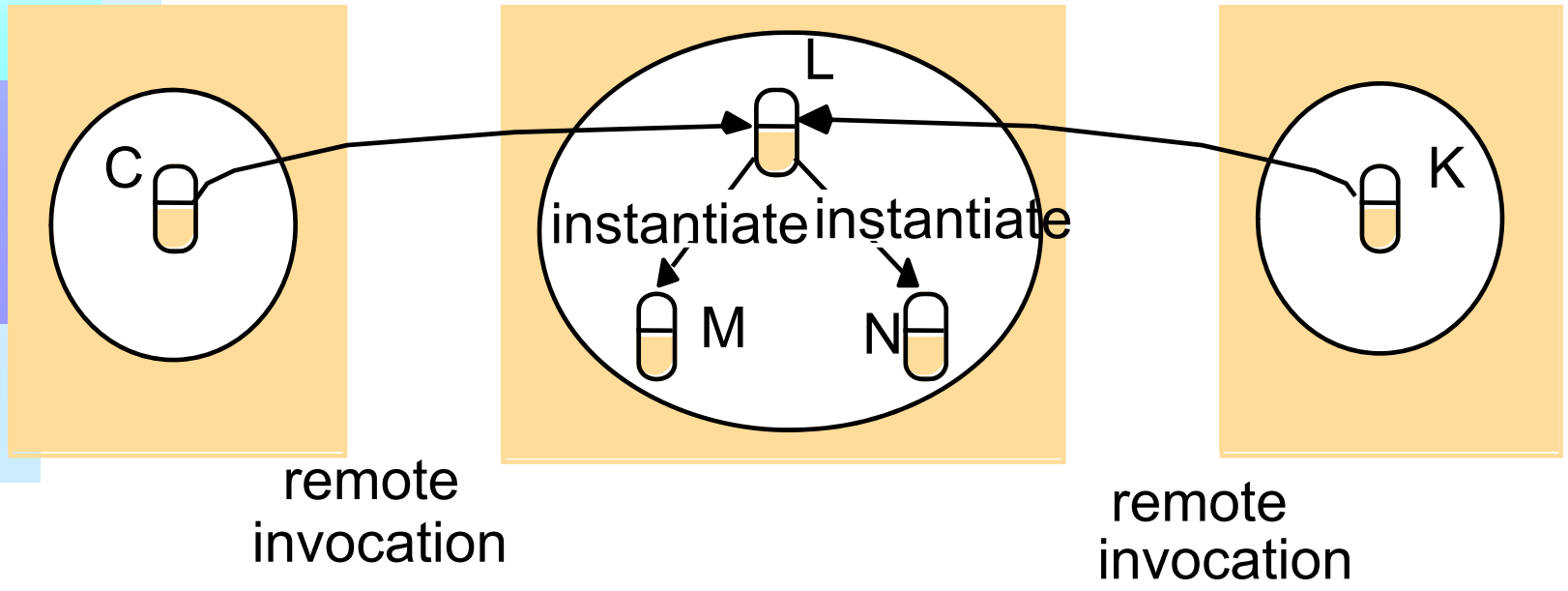
Distributed Objects

- Comunicazione tramite RMI
- Instanziazione di oggetti in locazioni diverse
- Migrazione degli oggetti con il loro codice (classe)
- Realizzazione di soluzioni fault-tolerant

- Remote object reference, OBJID, OBJ unique ID
- Remote Interface

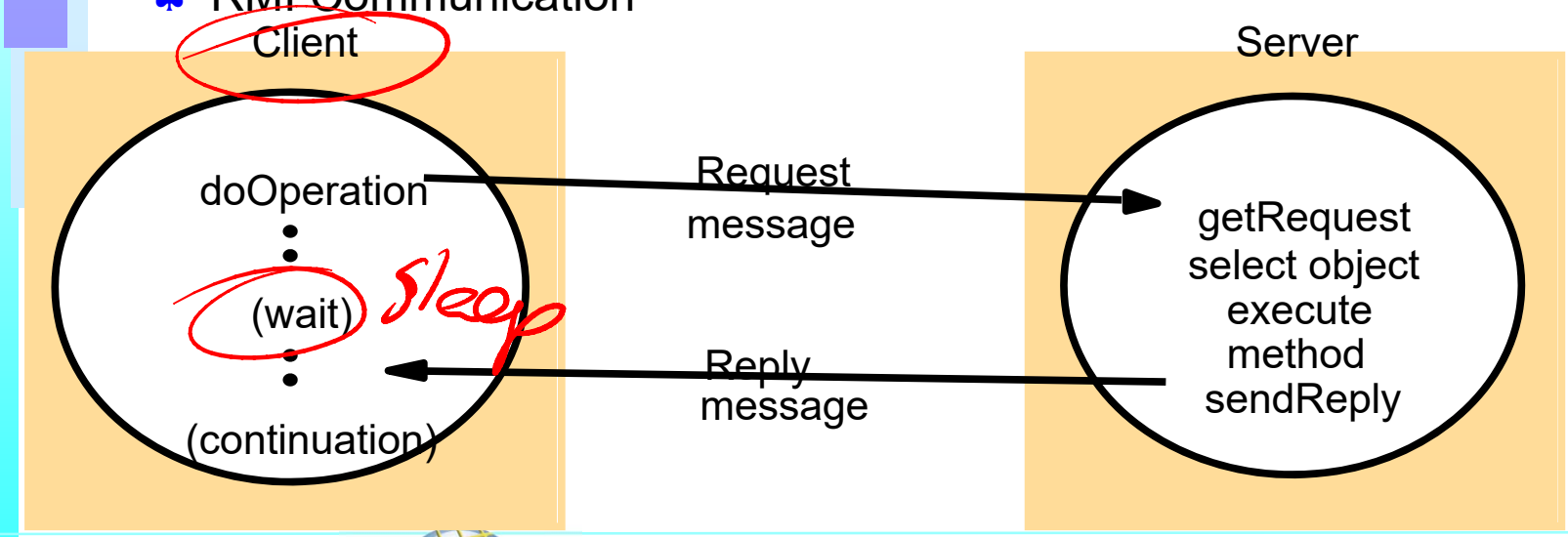
- Accesso ai descrittori dell'interfaccia degli oggetti
  - per definire in modo dinamico come accedere e quali accedere

Instantiation of remote objects



Object and Component Middleware

- Remote Method Invocation (RMI)
  - Main Idea: Working with an object on a remote machine is *made to look like calling a procedure on the remote site*, i.e.
    - the application/client sends a message to the remote object
    - the remote object receives message, does processing and sends back message with results - the server side;
    - the client receives message and uses/prints result
  - RMI Communication

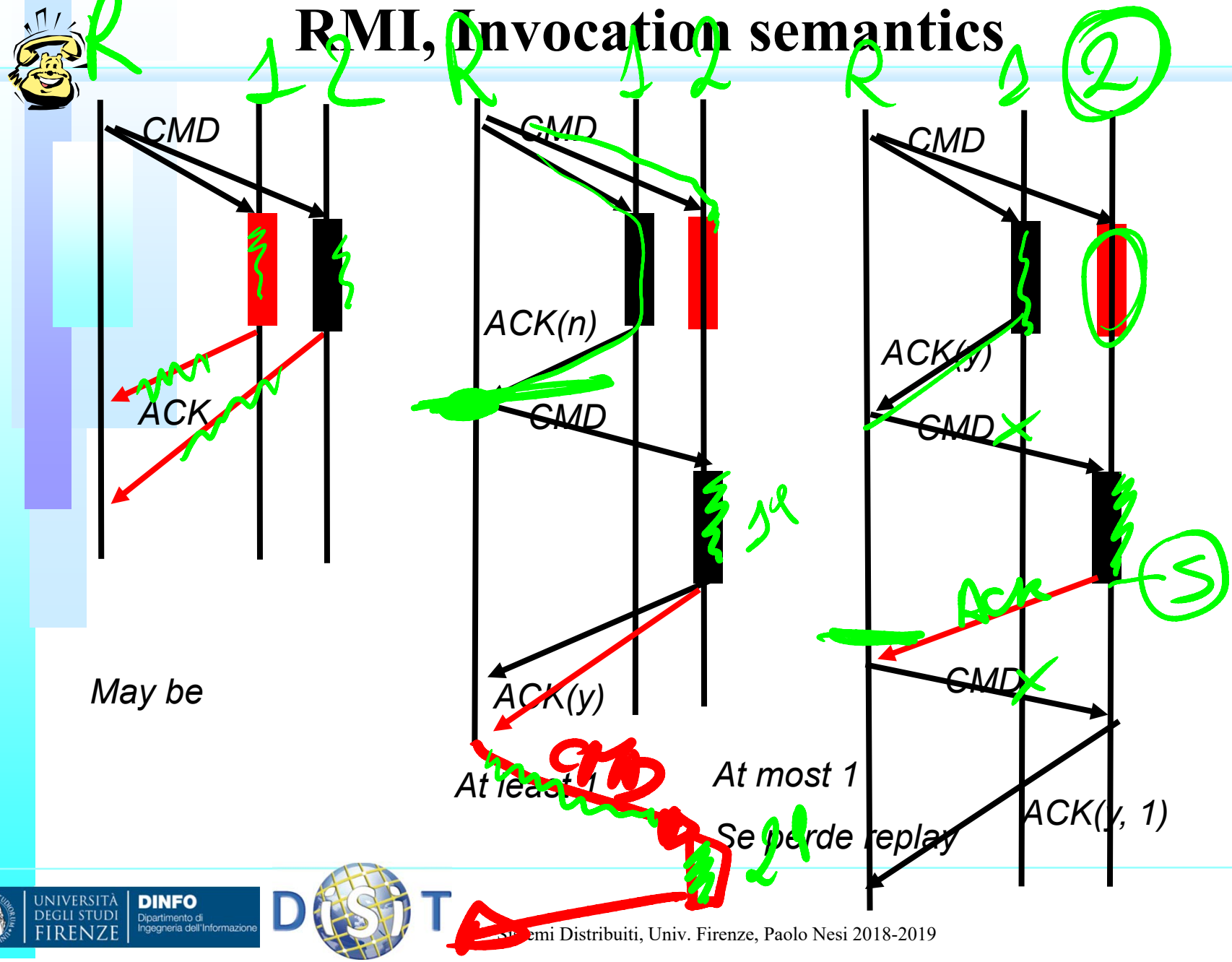


RMI, Invocation semantics

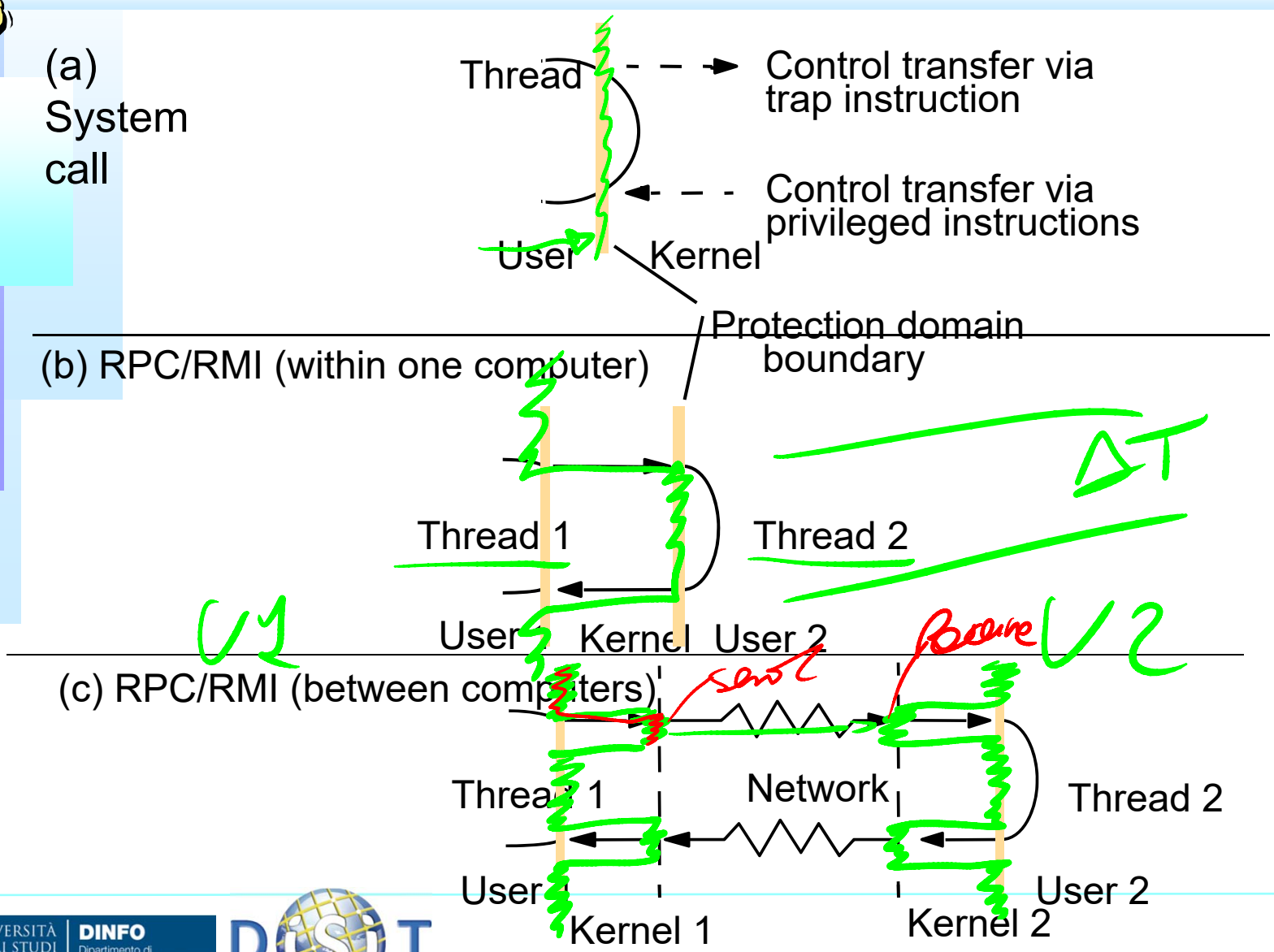
Fault tolerance measures			*Invocation semantics*
Retransmit request message	*Duplicate filtering*	*Re-execute procedure or retransmit reply*	
No	Not applicable	Not applicable	*Maybe*
Yes	No	Re-execute procedure	*At-least-once*
Yes	Yes	Retransmit reply	*At-most-once*

- *Invocation Semantics:*
  Esattamente una esecuzione
  solo un modello per local call

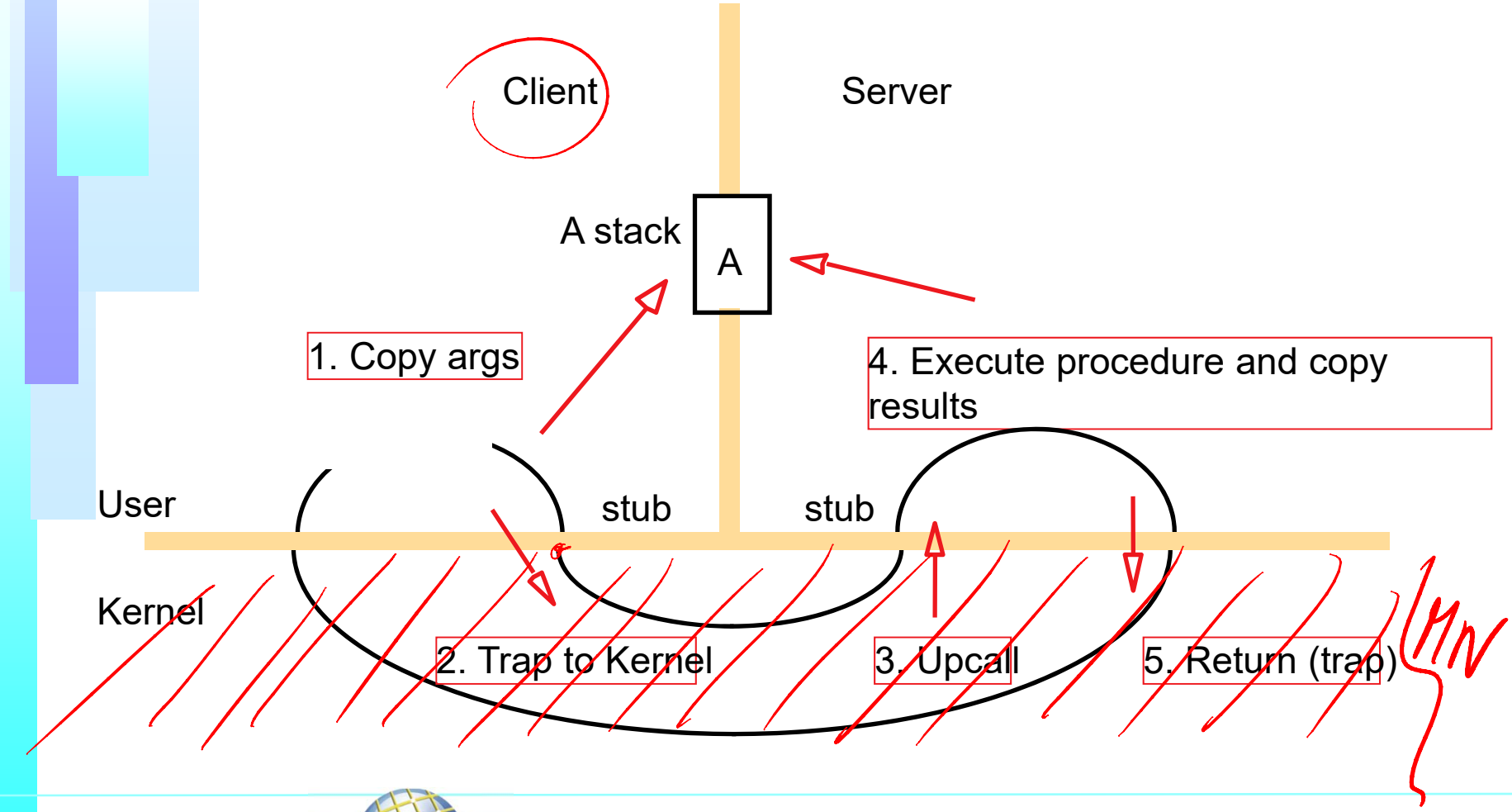
RMI, Invocation semantics



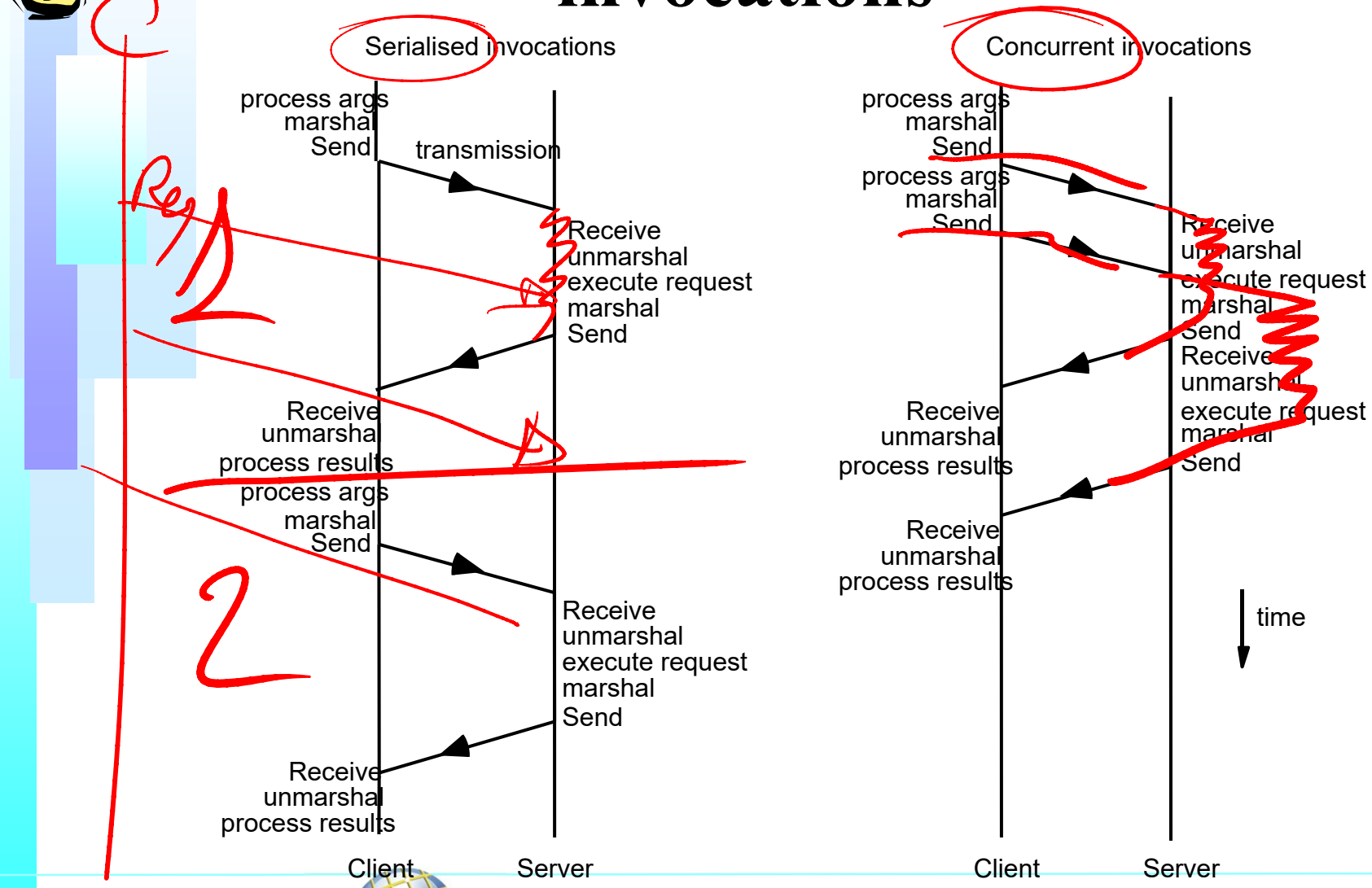
Invocations between address spaces



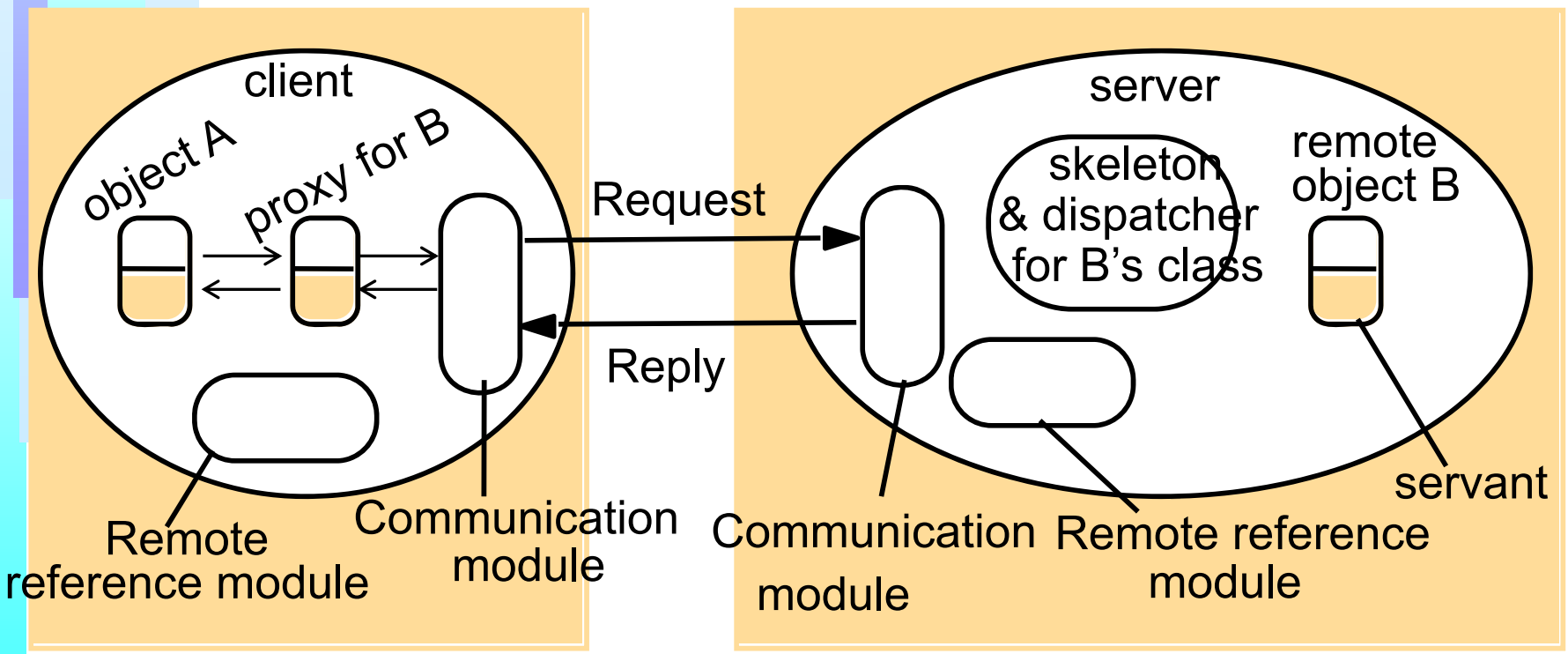
A lightweight remote procedure call



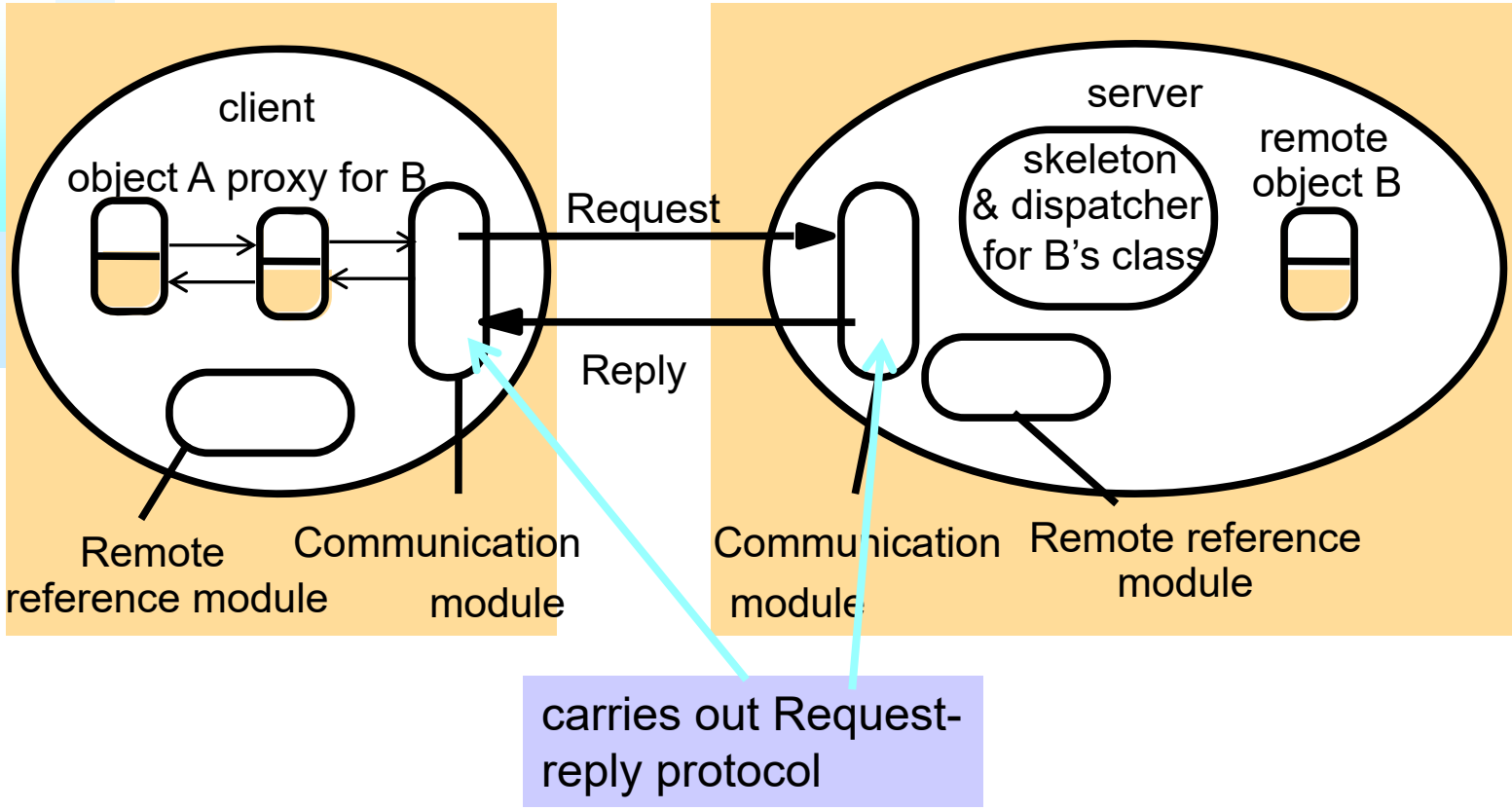
Times for serialized vs concurrent invocations



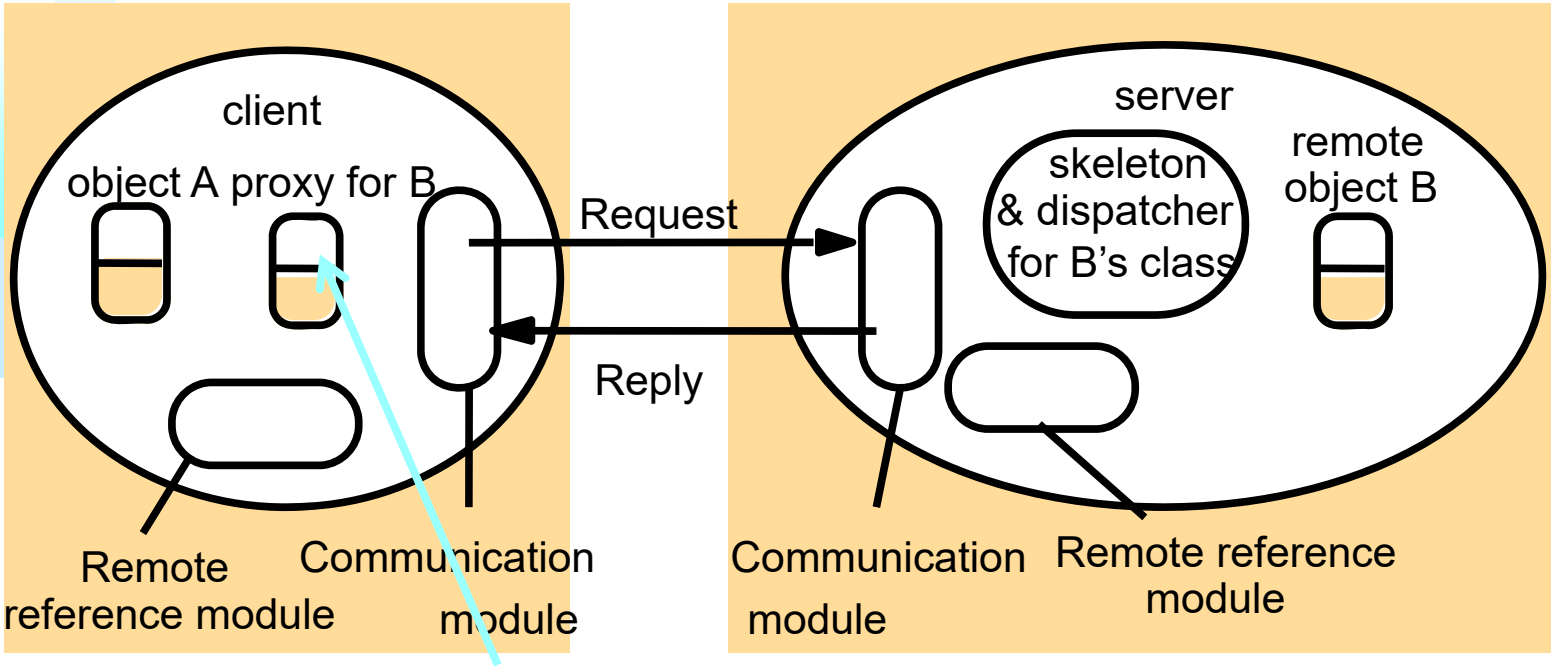
The role of proxy and skeleton in remote method invocation



The architecture of remote method invocation

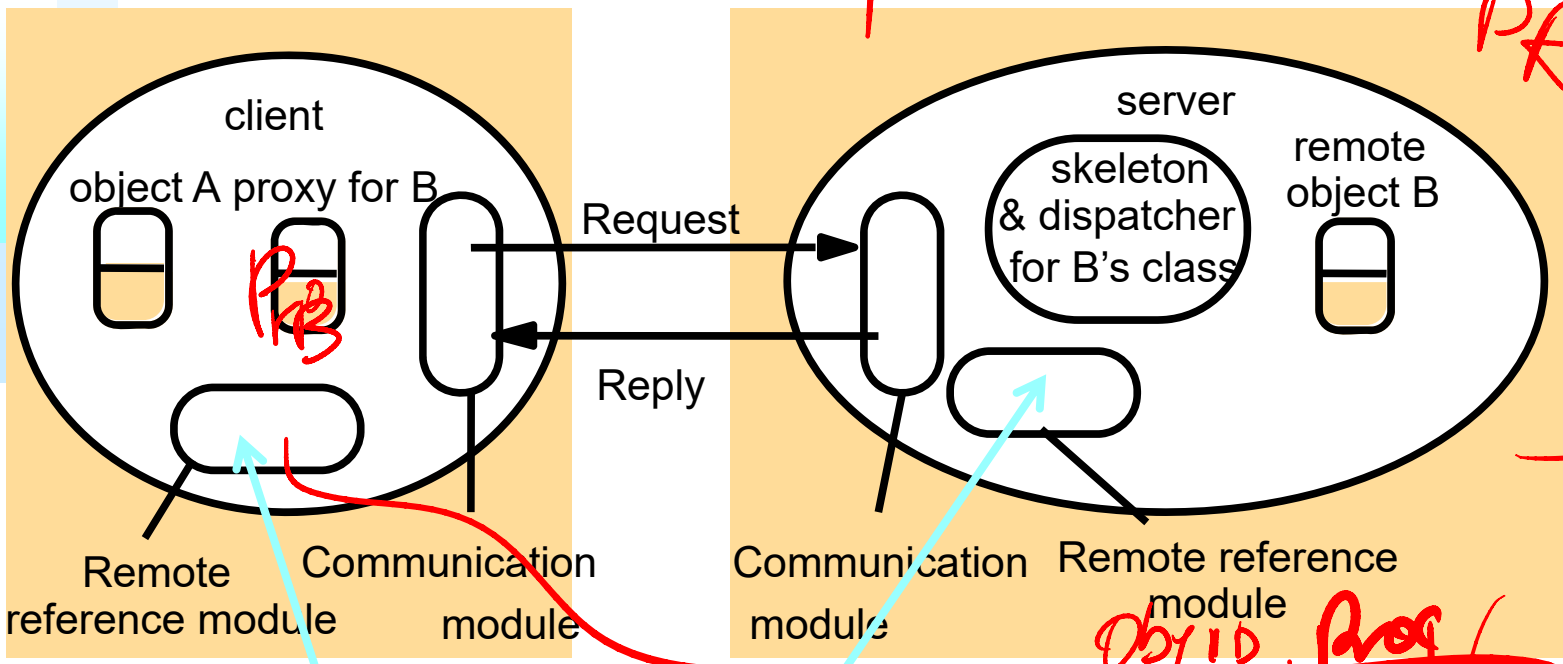


The architecture of remote method invocation



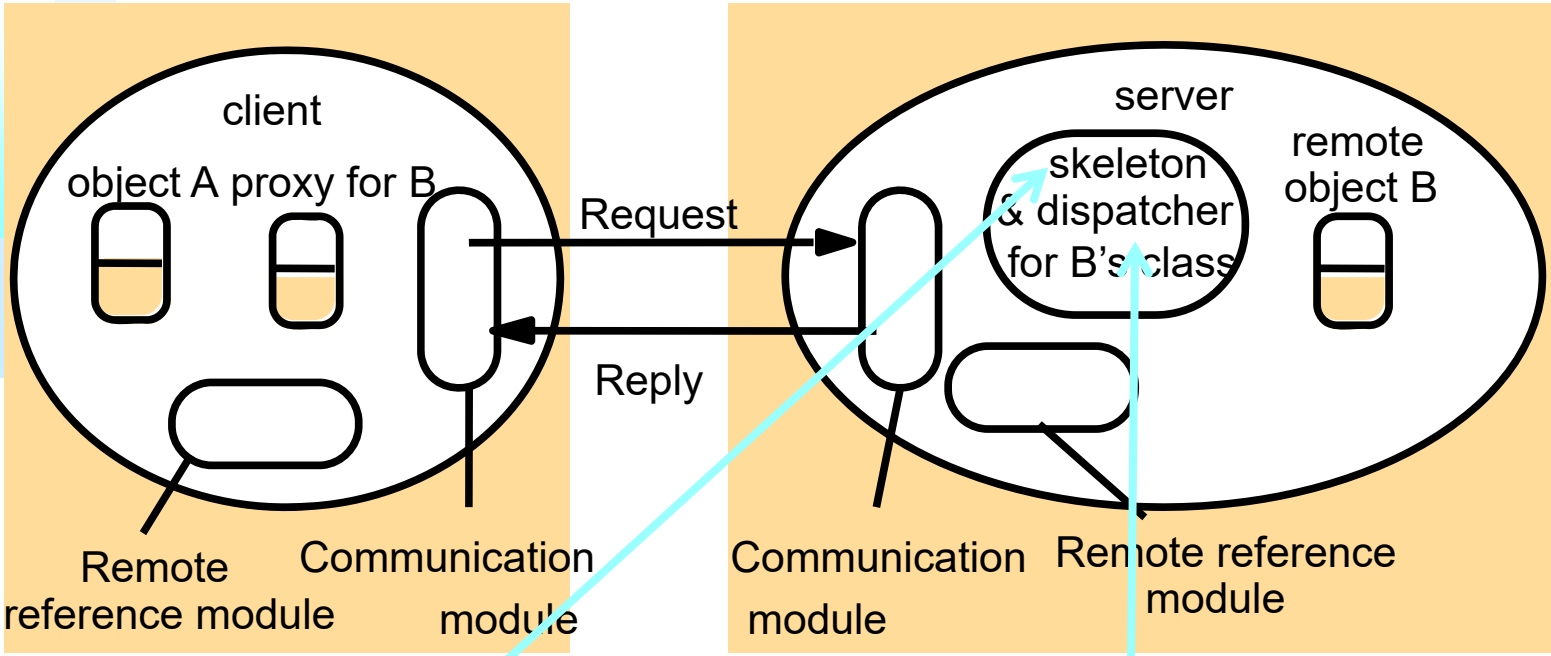
Proxy - makes RMI transparent to client. Class implements remote interface. Marshals requests and unmarshals results. Forwards request.

The architecture of remote method invocation



translates between local and remote object references and creates remote object references. Uses remote object table

The architecture of remote method invocation

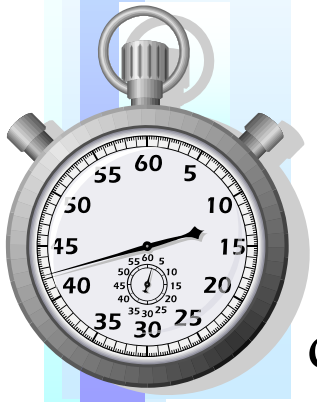


Skeleton - implements methods in remote interface. Unmarshals requests and marshals results. Invokes method in remote object.

Dispatcher - gets request from communication module and invokes method in skeleton (using *methodID* in message).

History of Distributed Object Models

- Communication Protocol Models:
  - Message passing/queuing (DCE, Distrib. Comp. Environment),
  - Request/response, (RPC, Remote Procedure Code)
  - Virtual processes: PVM (Parallel Virtual Machine), etc.
- 1980 model based on network layer (NFS, DCE, RPC)
- 1990 object-oriented RPC, to link objects
- 1993 COM (Comp. Object Model), (from OLE, Active X)
- 1996 Java
- 1997 Mary Kirtland's articles in Microsoft System Journal first sketch (COM+)
- 1997 Sun vs Microsoft over Java licensing
- 1999 Java 1.2
- 2000 Microsoft announces .net, C#



Parte: 4b – Clock e Ordinamenti Temporali

Corso di: Sistemi Distribuiti
Lauree in: Ingegneria Informatica,
delle Telecomunicazioni ed Informatica di Scienze

Prof. Paolo Nesi

Department of Systems and Informatics, University of Florence
Via S. Marta 3, 50139, Firenze, Italy
tel: +39-055-2758515, fax: +39-055-2758570

DISIT Lab, Sistemi Distribuiti e Tecnologie Internet

http://www.disit.dinfo.unifi.it/

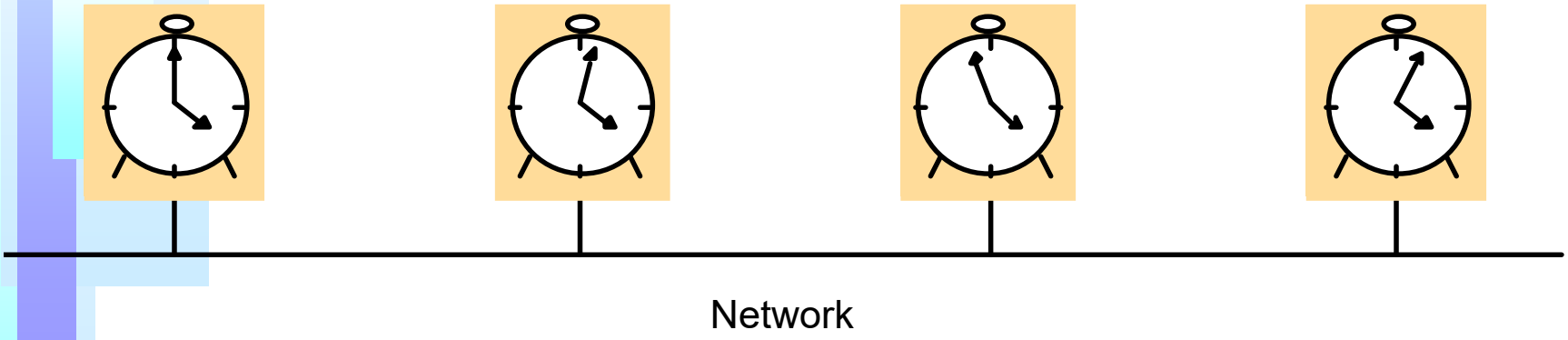
paolo.nesi@unifi.it
http://www.disit.dinfo.unifi.it/nesi


Clock e ordinamenti

- Motivazioni
- Problemi di sincronizzazione fra nodi
- Algoritmi di sincronizzazione
- Sincronizzazione di tempo assoluto fra nodi
- Ordinamento di eventi sui nodi

- Per riferimenti si veda il libro.

Problemi di Sincronizzazione fra Nodi



- Differenze di tempo fra Clock/Orologi (valore assoluto del tempo) nei vari nodi/processi sulla rete

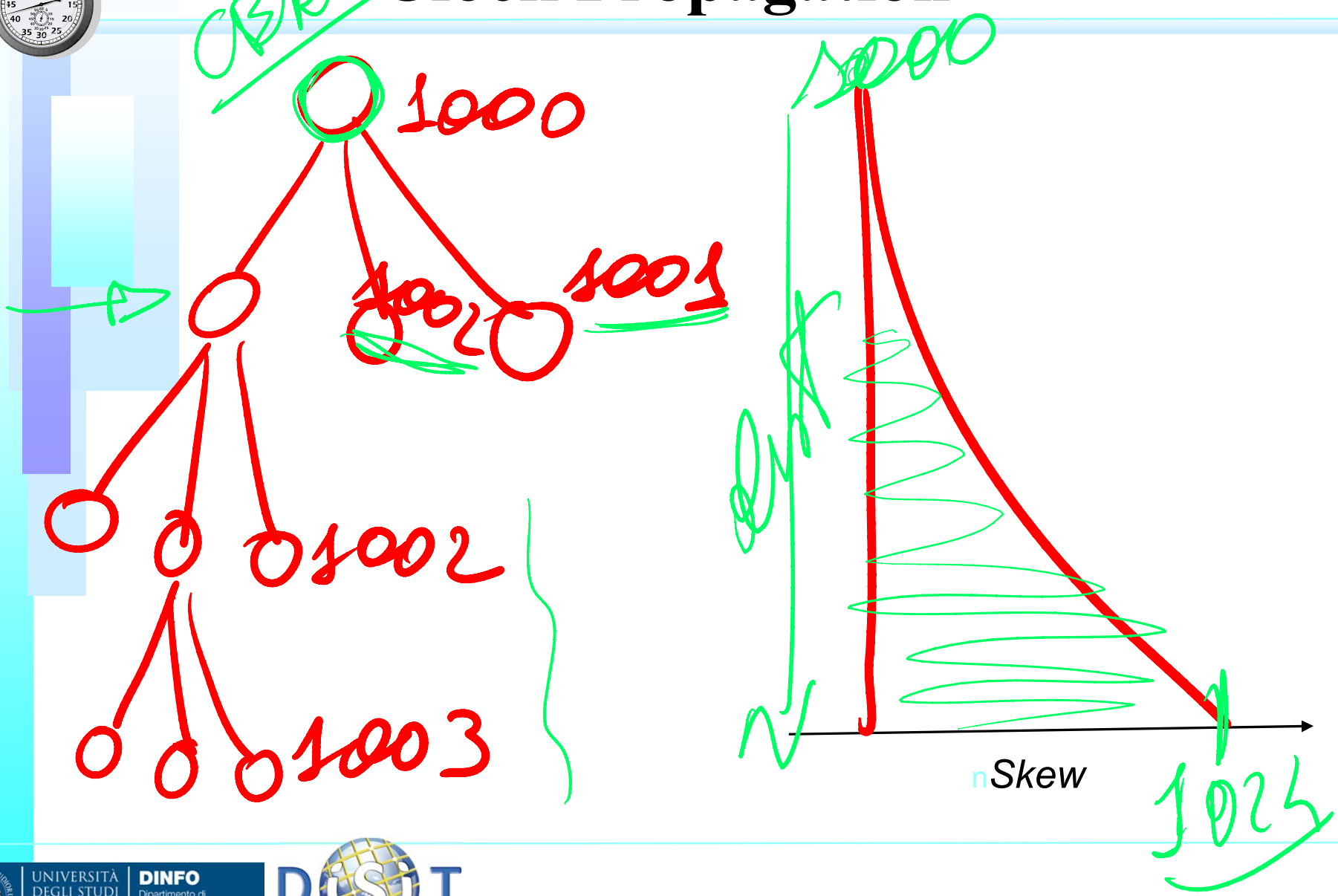
Motivazioni

- I nodi di un sistema distribuito possono avere **necessità** di effettuare azioni **sincronizzate** rispetto allo stesso tempo assoluto e non a fronte di un semplice comando d sinc.
  - Il comando su TCP/IP può arrivare con un ritardo (anche non predicibile) e questo in molti casi non e' accettabile
  - Per questo fine si possono inviare comandi su canali I/O specifici o tramite protocolli deterministici…….. ma se si ha solo la rete….
- In sistemi distribuiti i messaggi arrivano con dei **TimeStamp** in modo che si possa sapere in che ordine devono essere eseguiti (sono stati inviati)
  - I TimeStamp provenienti da nodi diversi devono riferirsi allo stesso tempo assoluto altrimenti il loro ordinamento (assoluto) non e' possibile, non ha senso, puo' essere errato.
  - L'ordinamento serve per ordinare in modo Causale i Comandi
- In sistemi distribuiti si effettuano anche misure del tempo per programmare delle durate, dei ritardi
  - Nelle stima delle durate (delta t) gli errori sono minori

Problemi di Sincronizzazione fra Nodi

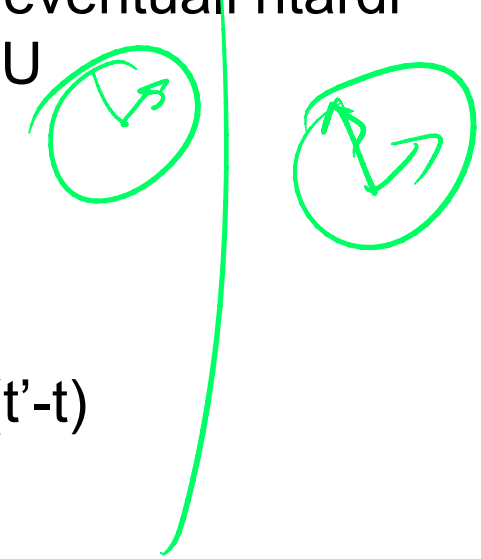
- La precisione degli orologi al quarzo e' dell'ordine di 10^{-6} , quelli molto precisi sono anche 10^{-7} o 10^{-8}
  - ***Drift o deriva***
- Un quarzo che ha precisione 10^{-6} puo' produrre uno scarto di un secondo ogni 11.6 giorni circa
  - Pertanto si ha circa un decimo di secondo ogni giorno
- In certe applicazioni real time una differenza di un decimo di secondo non e' accettabile,
  - per esempio quando devo far partire due audio insieme, un grande stadio, una stazione, etc.
  - un umano non addestrato sente differenze di 10 ms
- Questi problemi producono uno SKEW / deriva fra clock
  - differenza di tempo assoluto o comunque di tempo fra clock sincronizzati

Clock Propagation



Problemi di Sincronizzazione fra Nodi

- I singoli processi possono avere un proprio clock
- certamente usare quello del sistema che e' HW e' meglio poiche' qualsiasi orologio SW dipende da eventuali ritardi di **accesso al processo** del clock alla CPU



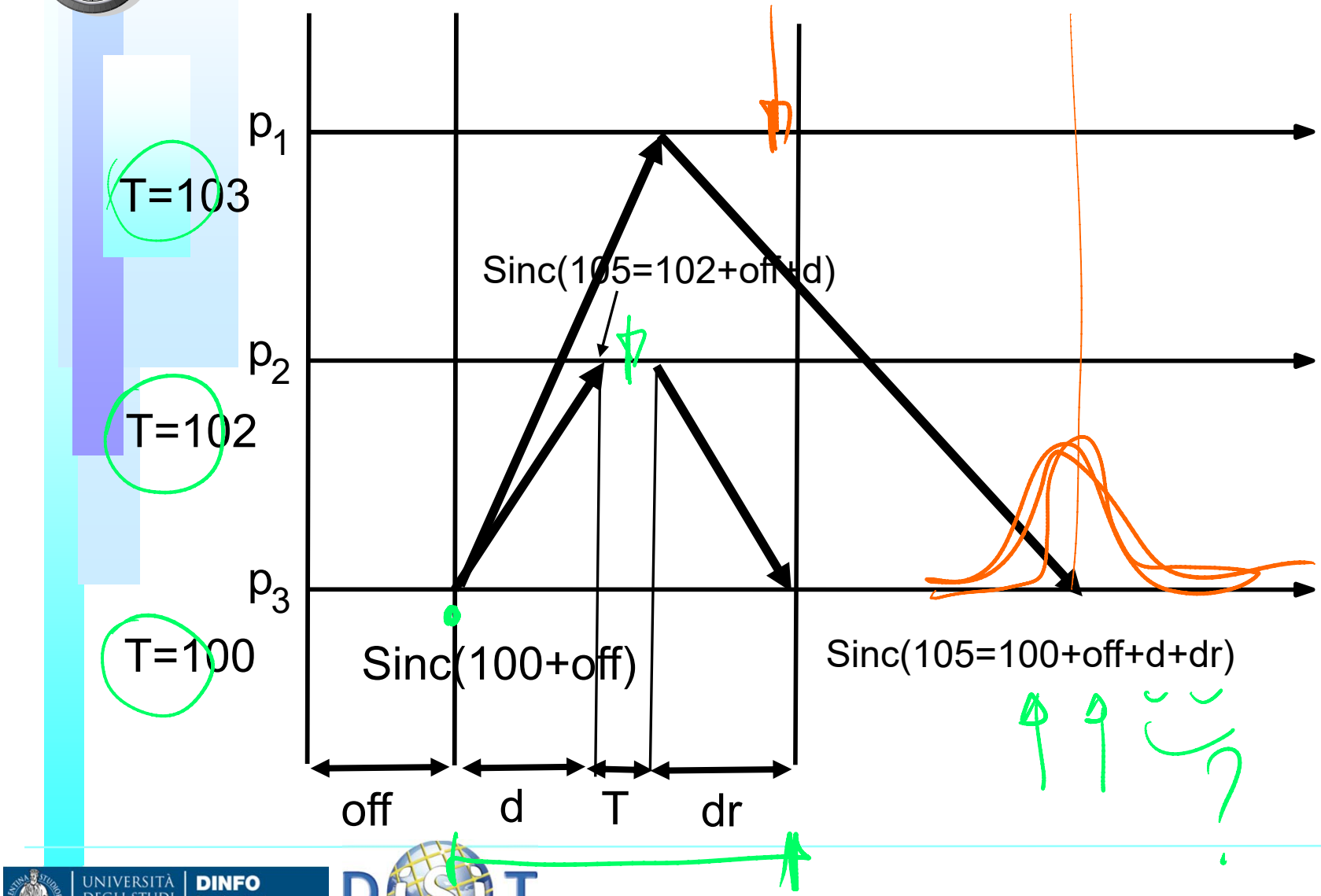
- Se due HW clock hanno una deriva:

$$(1-d)(t'-t) \;<=\; H(t')-H(t) \;<=\; (1+d)(t'-t)$$

Dove:

- H(t) e' il valore del clock del computer H al tempo t.
- d e' il valore della deriva per esempio 10^{-6}

Sincronizzazione fra Nodi, MODO A



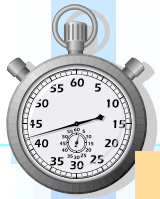
- d: delay di andata
- dr: delay di ritorno
- T: tempo di esecuzione, reazione
- off: offset, scarto di sincronizzazione reale

- Di Seguito vediamo modelli di correzione:
  - MODO A
  - MODO B
  - Modello C

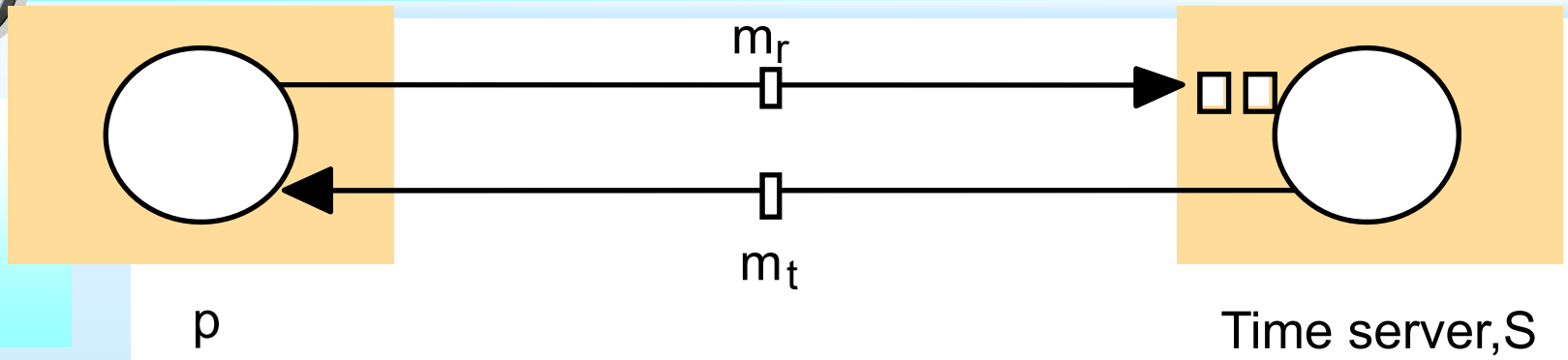
Sincronizzazione fra Nodi, MODO A

- Da un nodo Server di sincronizza gli altri nodi con dei messaggi
- I messaggi arrivano ai nodi con
  tempi non predicibili: d(i), dove i e' il nodo
- Le risposte arrivano al Server dai nodi con
  tempi non predicibili: dr(i), dove i e' il nodo

- Tramite varie iterazioni e' possibile stimare per ogni nodo
  - valori medi, minimi e massimi per: d(i) e dr(i), certamente per d(i)+T+dr(i). Dove T e' il tempo di risposta del nodo, non noto.
  - Alla fine e' possibile considerare anche:
    - ritardo(i) = [mean(d(i)+T+dr(i))] / 2
  - Come valore medio di ritardo, che può essere comunicato al momento della sincronizzazione dal Server:
    - Sinc(i,ritardo(i))
- Quando si comunica Sinc() non e' detto che si abbia un tempo effettivo simile a quello medio, da questo nascono degli errori.

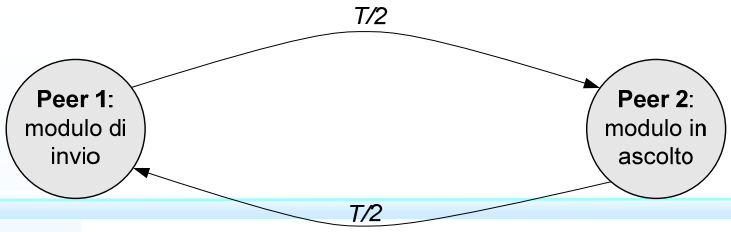


MODO B, Cristian (1989)

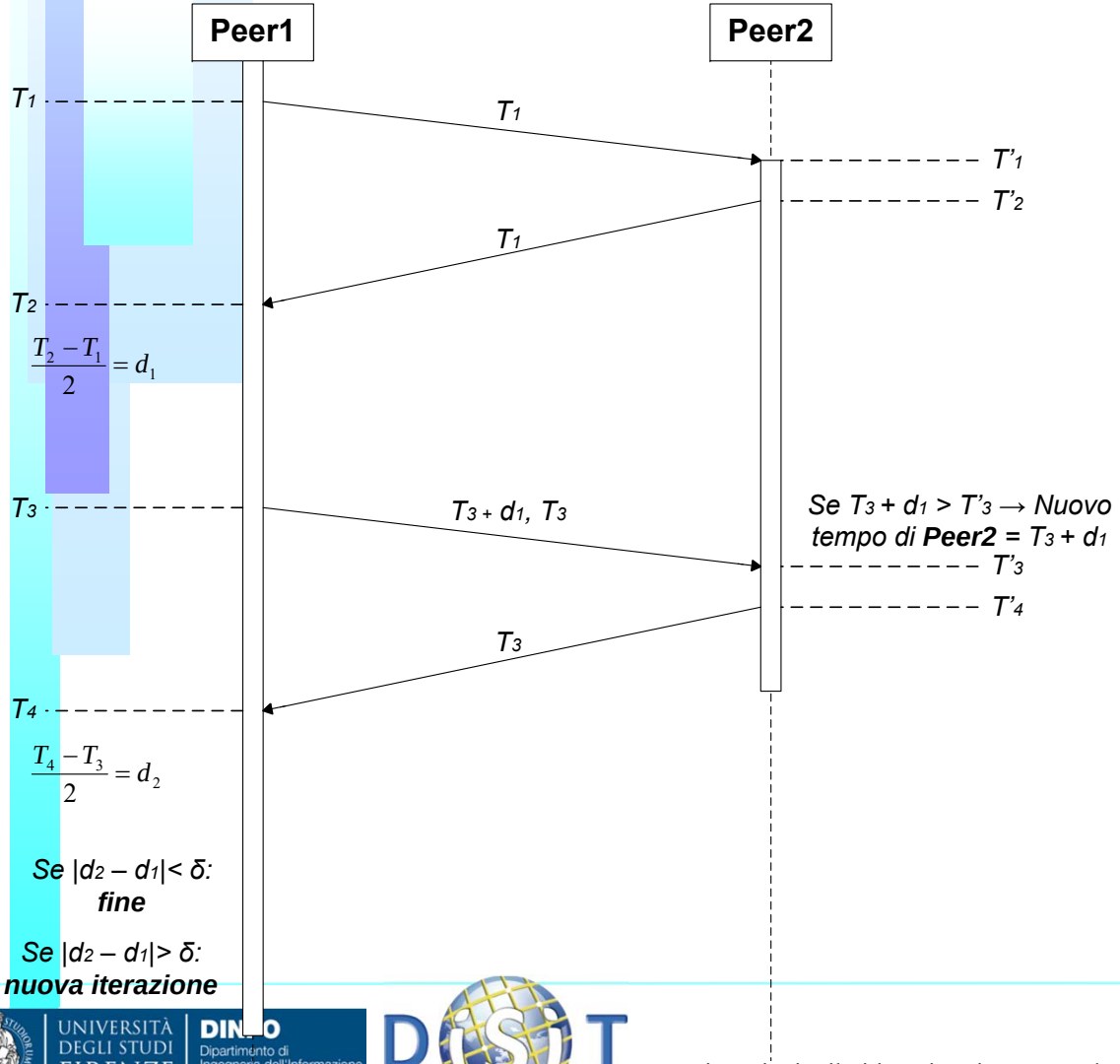


- E' il client P che chiede al *Server S* il tempo assoluto T
- P misura il tempo che passa da quando si manda *mr* a quando si riceve *mt*:
  - detto *Tround* (tempo per fare il giro)
- P impone/setta il suo tempo assoluto pari a (T+Tround/2)
- Errore pari a ±*(Tround/2 - Tmin)*
  - Tmin e' il minimo dei Tround possibili
  - Pertanto in questo modo e' possibile capire se l'errore e' accettabile per il tipo di applicazioni in cui si vuole utilizzare tale metodo

Sincronizzazione



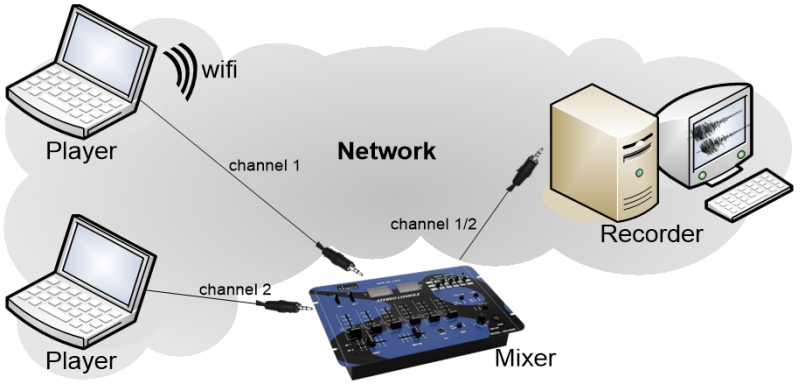
roundtrip-time = T → ritardo di rete = T/2



- Il peer con *Clock Logico* più elevato impone il tempo agli altri peer.
- Il nuovo tempo impostato non può mai essere inferiore al precedente (funzione monotona crescente).
- Alla connessione ogni peer imposta a 0 il suo clock logico e viene 'contattato' dagli altri peer attivi per essere sincronizzato.
- Si tenta di minimizzare l'errore nella comunicazione del ritardo di rete (d_1).
- δ si rilassa per raggiungere la convergenza.
- Necessaria periodica ri-sincronizzazione a causa della deriva dei clock fisici.

Risultati: Sincronizzazione

- Test di sincronizzazione tramite impulsi audio.
- Delay di esecuzione di 1 secondo.
- Mixer collegato a macchina in registrazione.
- Ingressi al mixer: altoparlanti di 2 pc (uno connesso tramite wifi, l'altro tramite cavo).
- Misura del ritardo nell'esecuzione dell'impulso.
- **0,2ms < Ritardo < 2,5ms**

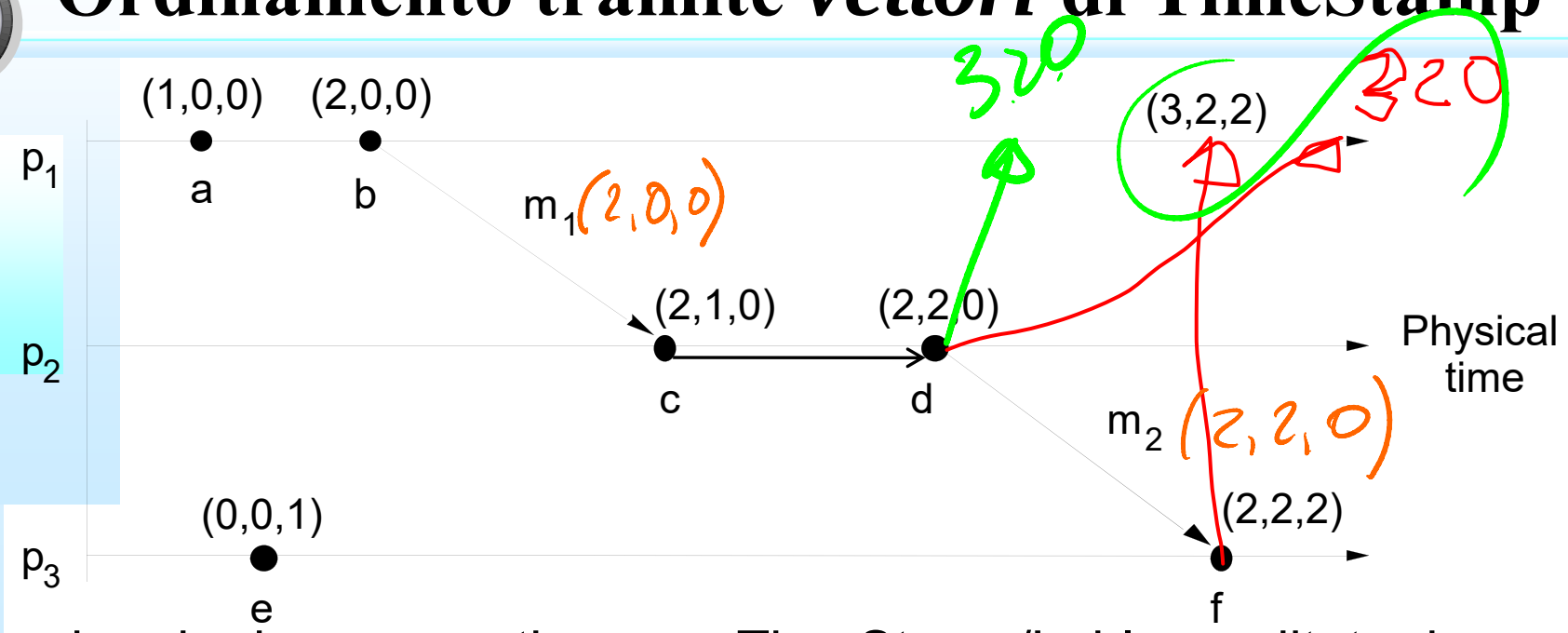


Altri Metodi

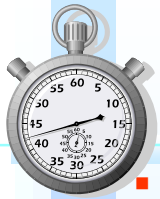
- **Modo B** non porta a risultati soddisfacenti quando i tempi di trasmissione sulla rete (Tround) sono elevati,
  - valido per Intranet
- In reti geografiche sono necessarie altre soluzioni
  - le soluzioni iterative permettono di ridurre l'errore tramite approssimazioni successive con l'incrocio di messaggi fra i vari nodi della rete

- In generale la precisione infinita/assoluta non e' possibile
- L'ordinamento di eventi e' però possibile per garantire la causalità fra comandi in sistemi di tempo reale stretto per CSCW o che altro
  - Queste cose sono impossibili da realizzare con una base dei tempi assoluta

Ordinamento tramite *vettori* di TimeStamp



- Ogni nodo riceve eventi con un TimeStamp/Lable ereditato da altri eventi (P1, P2, P3)
- Ogni nodo tiene conto di tutti i TimeStamp/lable degli altri N nodi
- Si crea una catena di eventi gollegati ed etichettati
- I comandi sono ordinabili (2,2,0) e' successivo a (2,1,0)
- E' possibile in questo modo creare degli ordinamenti



Sequenza di eventi

- **Eventi:**
  - (1,0,0): evento a T=1 di P1
  - (2,0,0): evento a T=2 di spedizione di P1, succ a (1,0,0)
  - (2,1,0): evento succ a (2,0,0) di P2, ricezione, a T=1 di P2
  - (2,2,0): evento succ a (2,1,0) di P2, invio, a T=2 di P2
  - (2,2,2): evento a T=2 di P3, ricezione di (2,2,0), succ a (0,0,1) T=1 di P3
  - ……
  - (3,2,2) potrebbe arrivare prima di (3,2,0) su P1, ma se questo accade P1 puo' riordinare tali eventi

- Da un eventuale non corretto ordinamento e' possibile riportarsi all'ordine corretto
  - creare degli ordinamenti
  - Complessità o(N)
  - Ha senso solo per comandi dipendenti e collegati, dovrebbe essere evitato per comandi indipendenti

Ordinamento di comandi/eventi

- **Un sistema gode della proprieta' di convergenza** se:
  - si accorge di condizioni di disordine temporale e riporta con delle operazioni di ordinamenti in uncerto tempo ad uno stato ordinato uguale per tutti
  - (importante per sistemi cooperativi, vi sono degli esempi nelle slide dei sistemi collaborativi)

- Non e' detto che il sistema possa rimettere semplicemente in ordine i comandi quando se ne accorge,
  - il danno potrebbe essere gia' stato fatto, per esempio in un sistema di controllo per macchine obliteratrici
    - ➔Muovi e poi buca e' diverso da buca e poi muovi
    - ➔Il buco si trovera' in posizioni diverse ☺
  - se si parla di dati si dovrebbe garantire l'UNDO di ogni comando, in certi casi fisici non e' possibile, il buco e' fatto ☹

Parte: 4c – CORBA
"un Middleware"

Corso di: Sistemi Distribuiti
Lauree in: Ingegneria Informatica,
delle Telecomunicazioni ed Informatica di Scienze

Prof. Paolo Nesi

Department of Systems and Informatics, University of Florence
Via S. Marta 3, 50139, Firenze, Italy
tel: +39-055-2758515, fax: +39-055-2758570

DISIT Lab, Sistemi Distribuiti e Tecnologie Internet

http://www.disit.dinfo.unifi.it/

paolo.nesi@unifi.it
http://www.disit.dinfo.unifi.it/nesi

CORBA

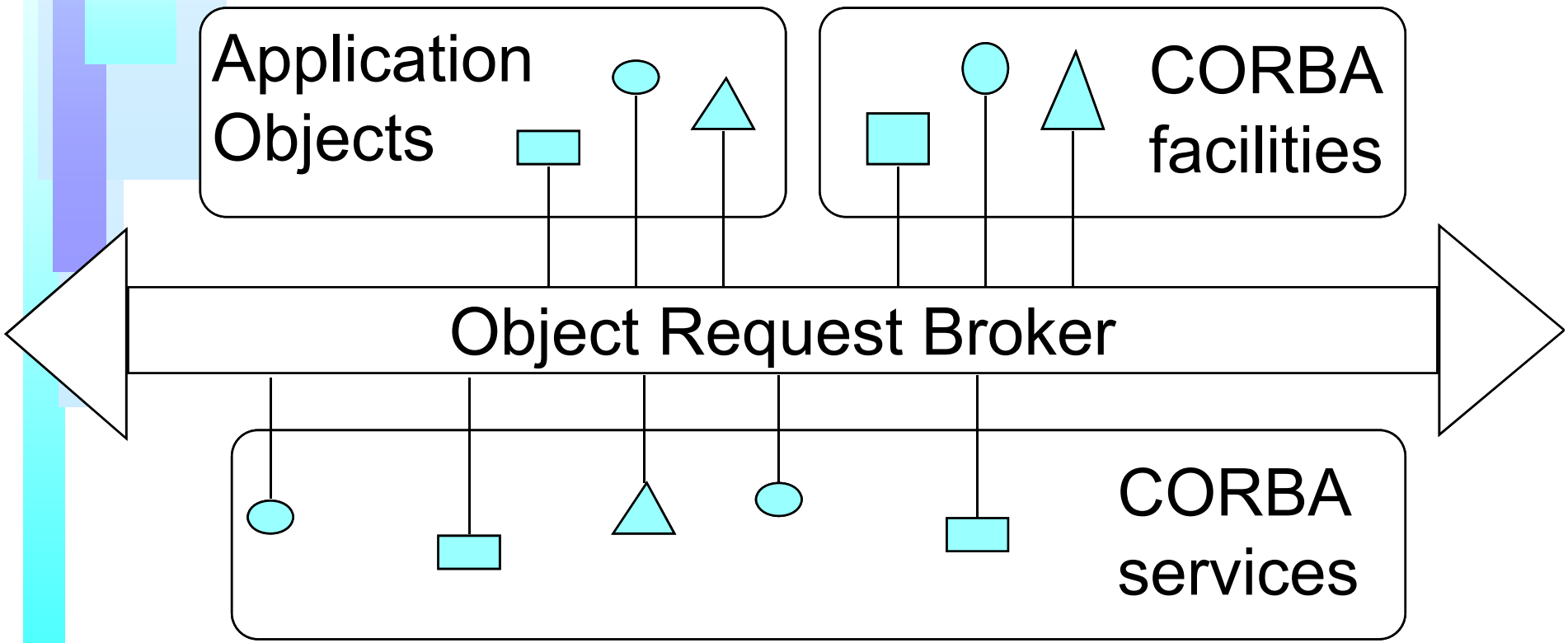
- CORBA Architecture
- General Concepts
- ORB Structure
- Client and Server in CORBA
- Object Adapter
- CORBA for WEB applications
- Usage of CORBA
- Single and Multithread CORBA

Common Object Request Broker Architecture

- **OMG**'s (Object Management Group) specification for interoperability between distributed computing nodes (1989)
- **ORB**: middleware that establishes requester-provider relationship
- **Goal**:
  - ♣ Usage of OO programming in Distributed Systems
  - ♣ Allow heterogeneous environments communicating at object level
  - ♣ regardless of implementation of the endpoints
    - ➔ Different languages in the applications
    - ➔ Different implementations of the ORB
- CORBA 1 (1990), CORBA 2 (1996)

CORBA, *Common Object Request Broker Architecture*

- Defined by the Object Management Group nel 1991
- Object Management Architecture

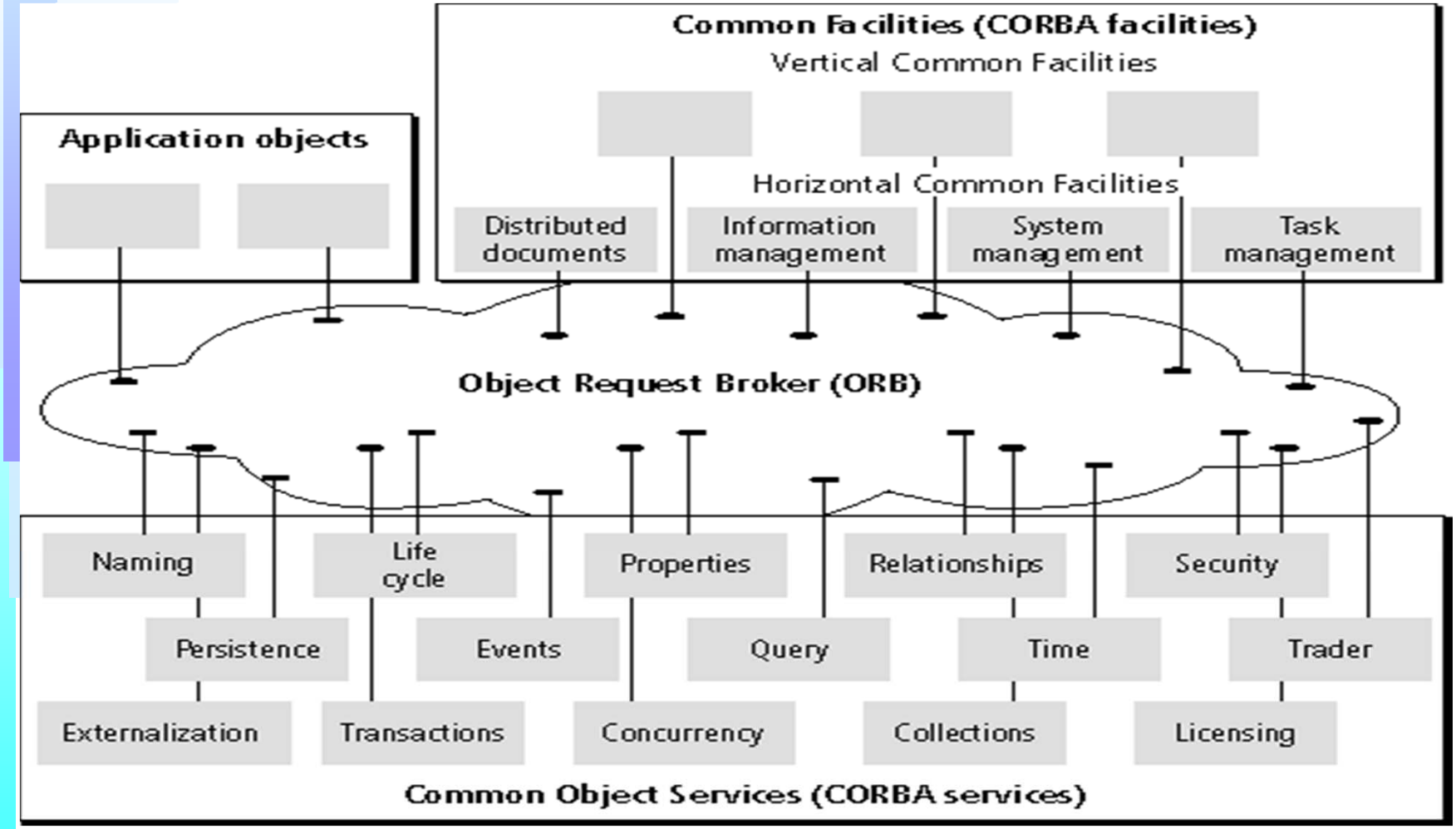


CORBA

- **Object Request Broker (ORB)**
  - The libraries, processes, and other infrastructure in a distributed environment that enable CORBA objects to communicate with each other.
  - The ORB connects objects requesting services to the objects providing them.

- **Naming service**
  - to allow CORBA objects to be named by binding a name to an object reference.
  - The name binding may be stored in the naming service, and
    - a client may supply the name to obtain the desired object reference.

CORBA detailed architecture

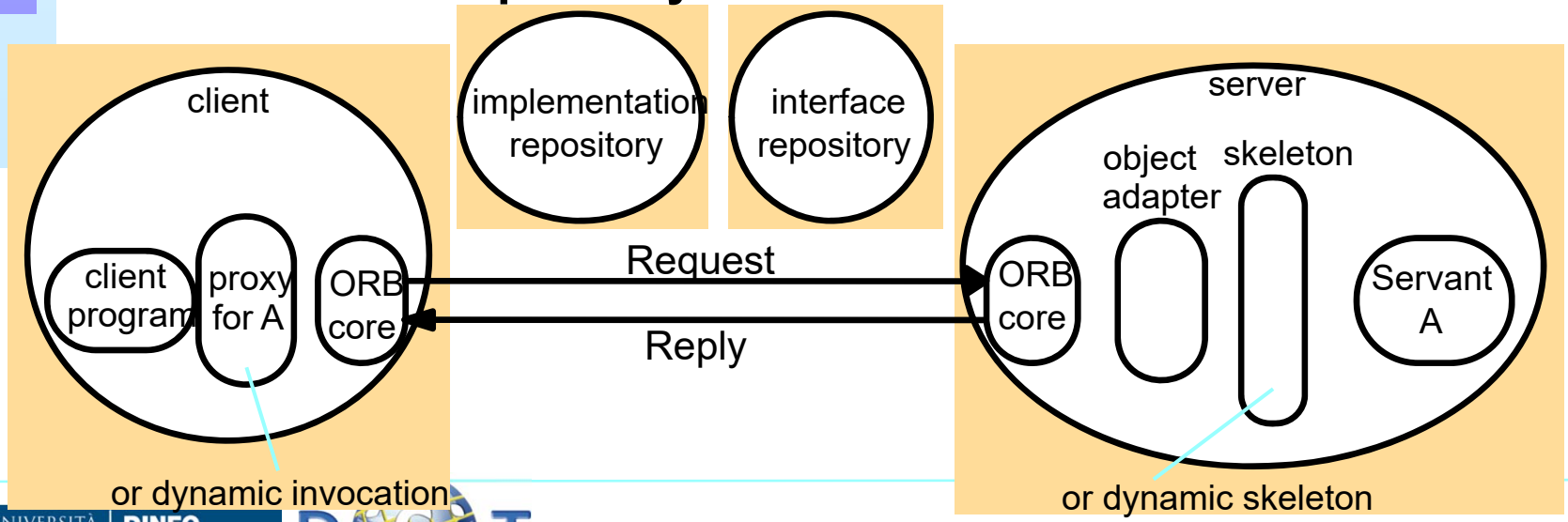


4 Componenti di CORBA

- **ORB, Object Request Broker**
  - Distributed application
  - rende trasparente la locazione fisica degli oggetti, naming
  - unmarshal-marshal, e invocazione dei metodi
- **CORBA Services**
  - Security, time, etc..
  - persistency, events, transactions, etc..
- **CORBA Facilities**
  - Servizi di base condivisi da molte applicazioni
  - Non vitali come i CORBA Services, OS esteso….
  - E.g.: amministrazione sistema, mail, etc.
- **Application Objects**
  - Objects basati su CORBA

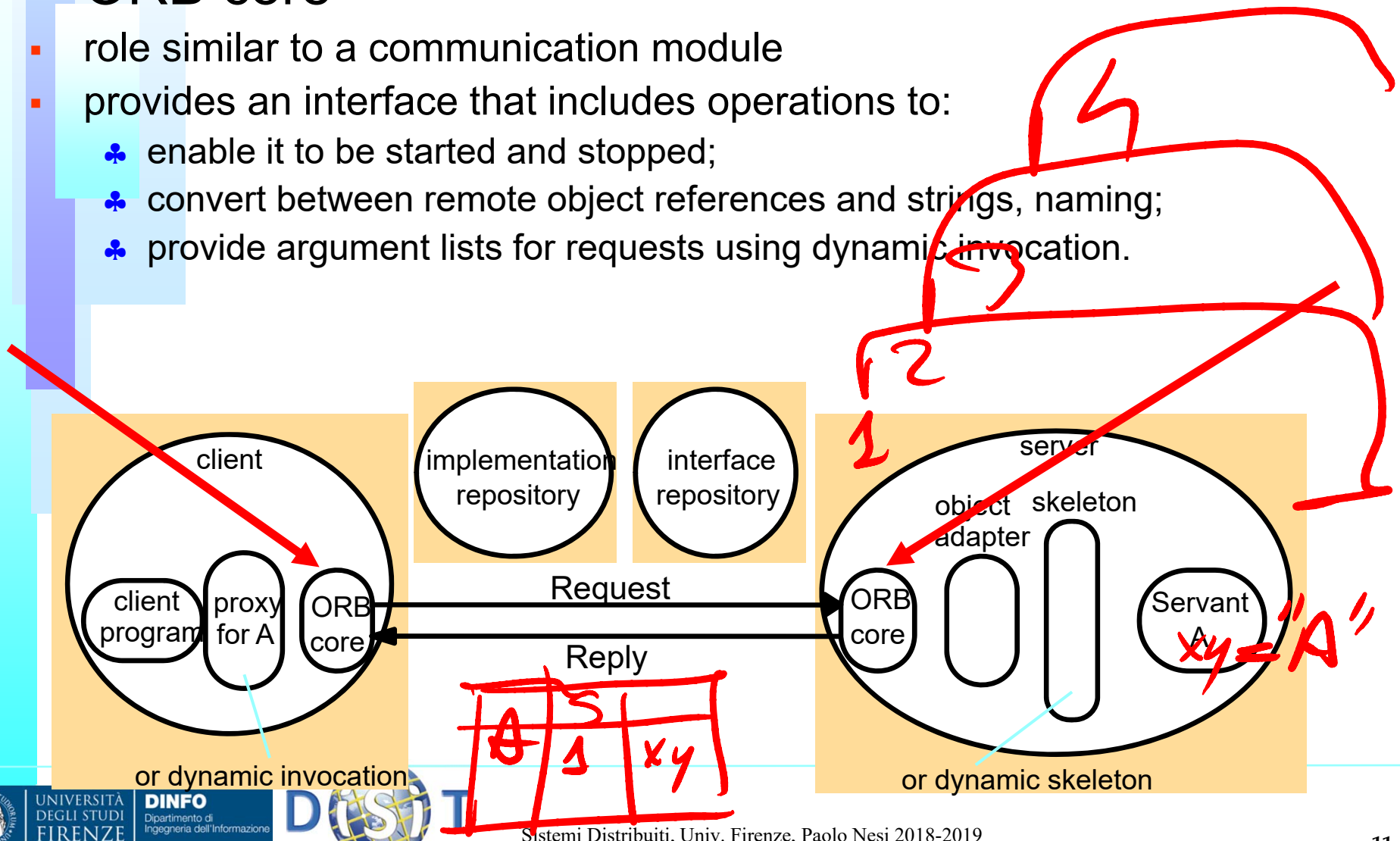
The main components of the CORBA architecture

- The CORBA architecture is designed to allow clients to invoke methods in CORBA objects
  - clients and objects can be implemented in a variety of programming languages
  - it has additional components with respect to a generic MiddleWare →
    - → **object adapter, implementation repository and interface repository**



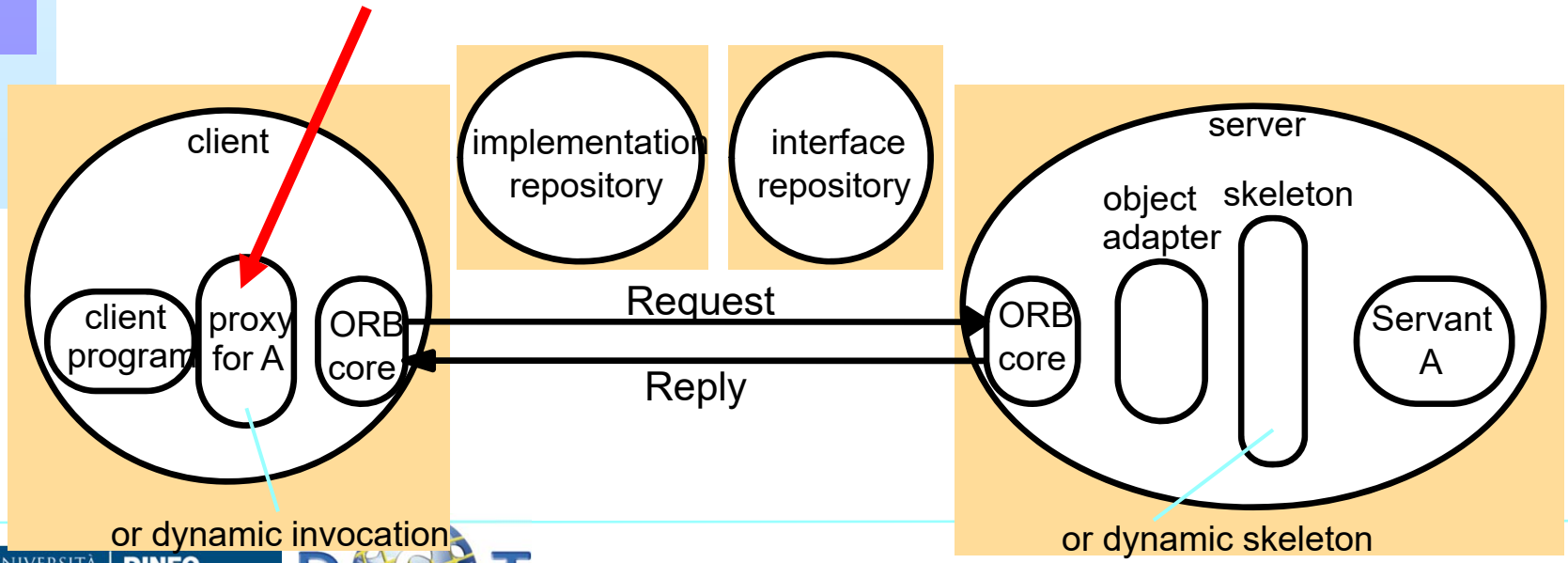
The main components of the CORBA architecture

- ORB core
- role similar to a communication module
- provides an interface that includes operations to:
  - enable it to be started and stopped;
  - convert between remote object references and strings, naming;
  - provide argument lists for requests using dynamic invocation.



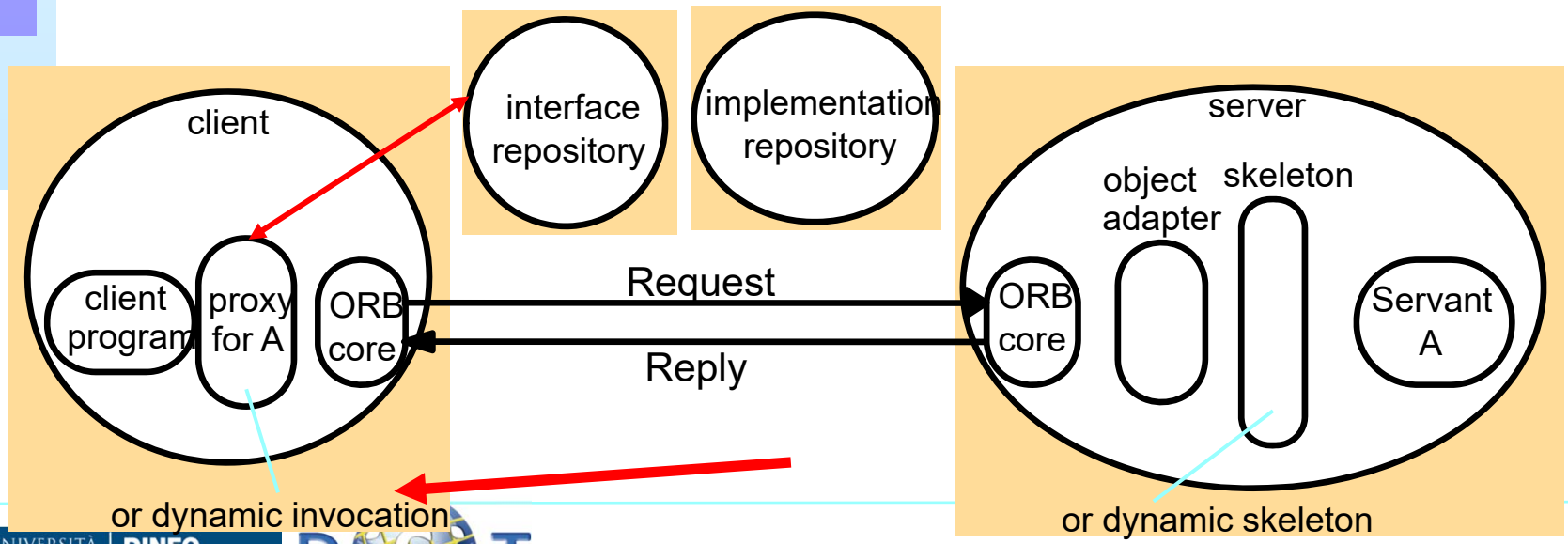
The main components of the CORBA architecture

- Client stubs/proxies
- Written in the **client language**.
- The IDL compiler for the client language uses an IDL interface to generate one of the following:
  - **for object-oriented languages** the class of a proxy
  - **for procedural languages** a set of stub procedures.
- the client stubs/proxies **marshal** the arguments in invocation requests and unmarshal exceptions and results in replies



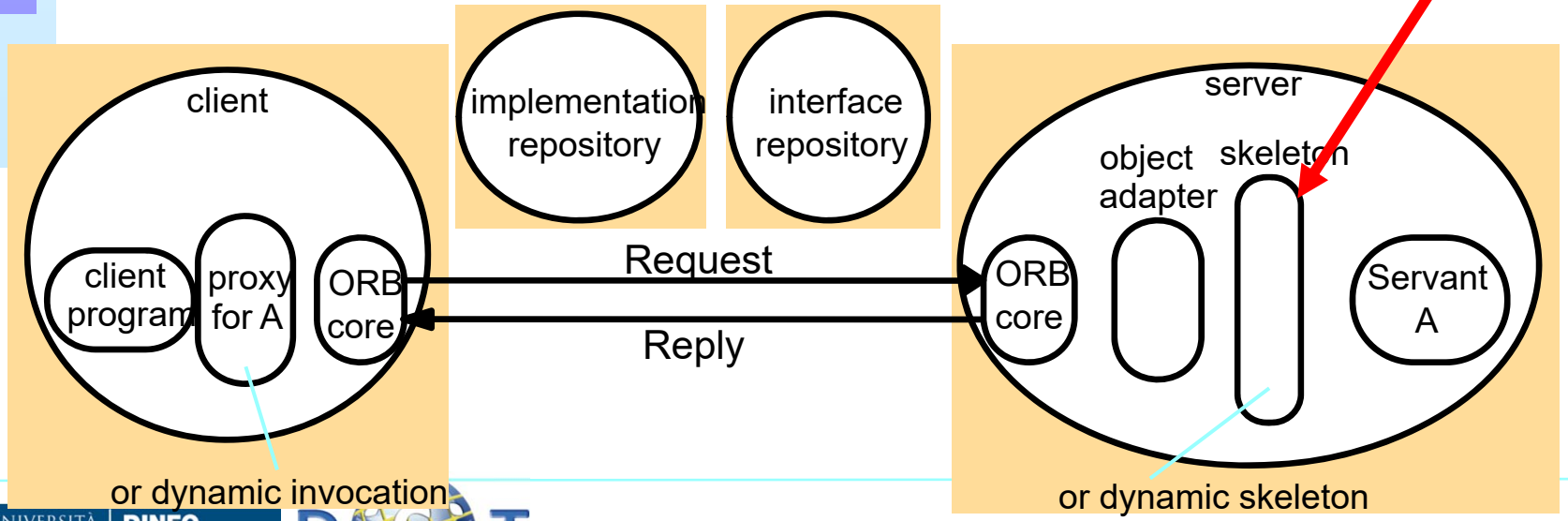
The main components of the CORBA architecture

- ***Dynamic invocation interface***
- In some applications (e.g., browsers), a client without the appropriate proxy class may need to invoke a method in a remote object.
- CORBA does not allow classes for proxies to be downloaded at run time as in Java RMI.
- The **dynamic invocation interface is CORBA's alternative**. (see the **Interface Repository**)



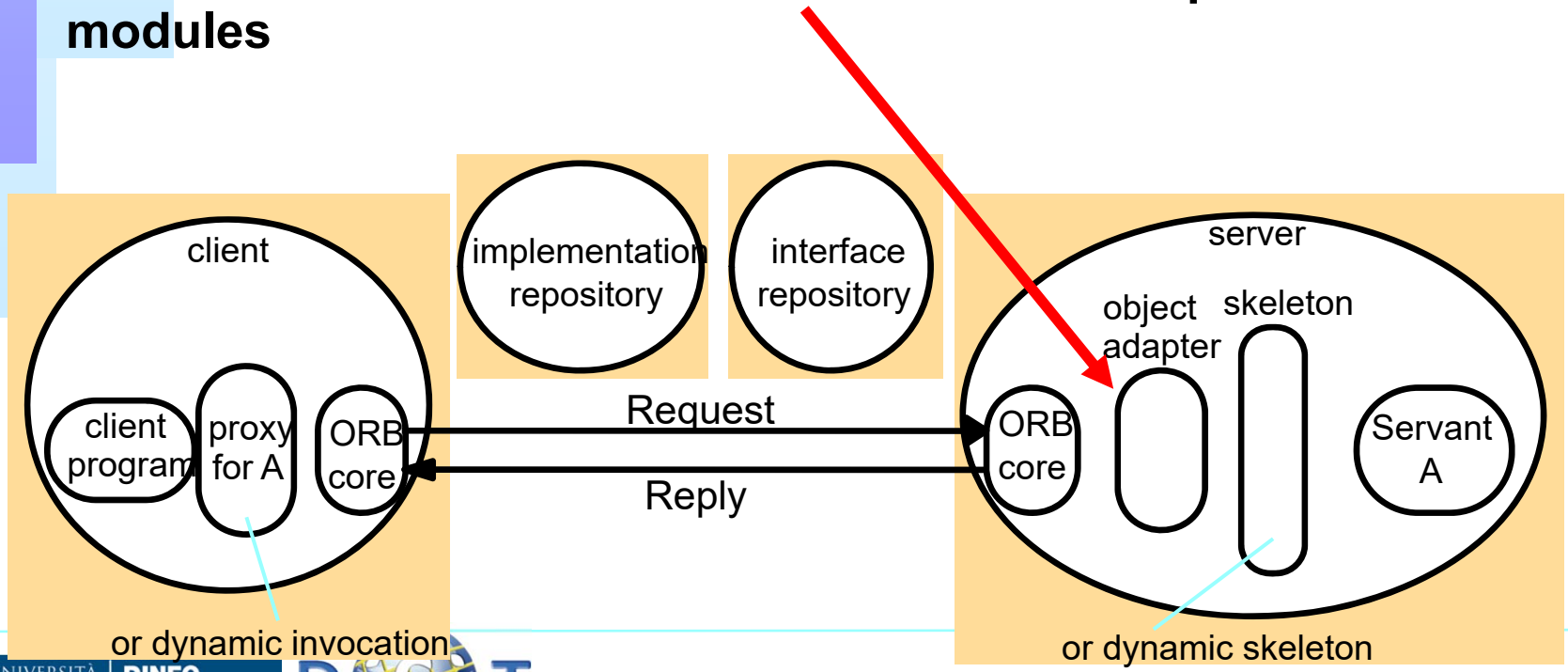
The main components of the CORBA architecture

- ***Skeletons***
  - skeleton classes (for OO languages) are **generated in the language of the server** by the IDL compiler.
  - remote method invocations are **dispatched** via the appropriate skeleton to a particular servant,
  - the **skeleton unmarshals** the arguments in request messages and marshals exceptions and results in reply messages.



The main components of the CORBA architecture

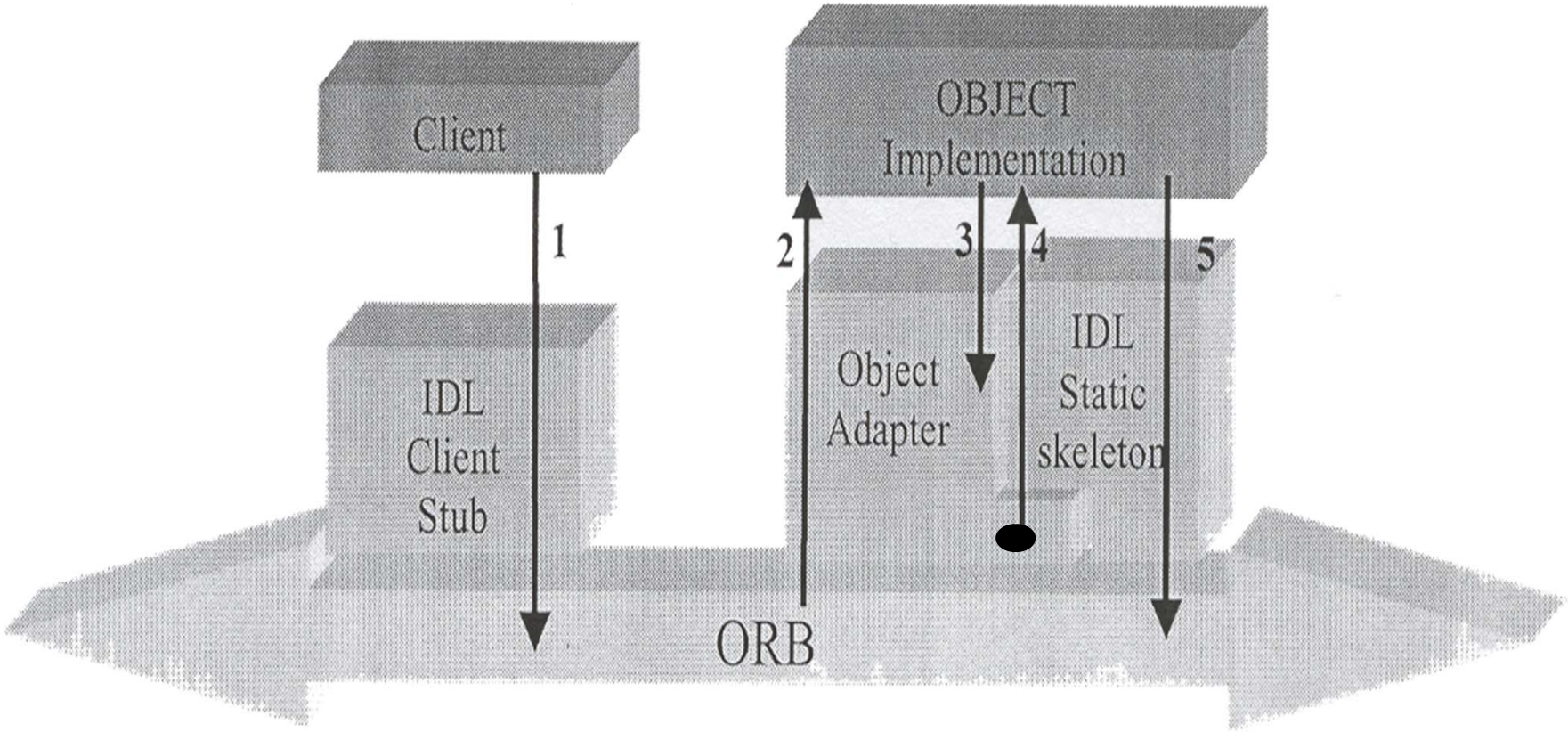
- ***Object adapter***
- bridges the gap between
  - ♣ CORBA objects with IDL interfaces and
  - ♣ the **programming language interfaces** of the corresponding servant classes.
- it does the **work of the remote reference and dispatcher modules**



Object Adapter

- **Object Adapter (Portable object adapter)**
  - provides ORB services to particular groups of object implementations
- **Services and duties/activities**
  - generation and interpretation of object references, mapping object references to
    - Specific implementations, and
    - registration of implementations.
  - method invocation (dispatching)
    - via a skeleton, object and implementation activation and deactivation
  - security of interactions (method access control, etc.)

OA, Object Adapter



OA, Object Adapter

1. The client via the stub calls the method at the ORB.
2. The ORB notifies the calls the OA which activates in turn the correct implementation
3. L'implementazione si registra e si dichiara pronta.
4. L'OA passa l'invocazione allo skeleton che spacchetta i parametri e li fornisce all'implementazione.
5. L'implementazione esegue il metodo e ritorna parametri di ritorno al client.

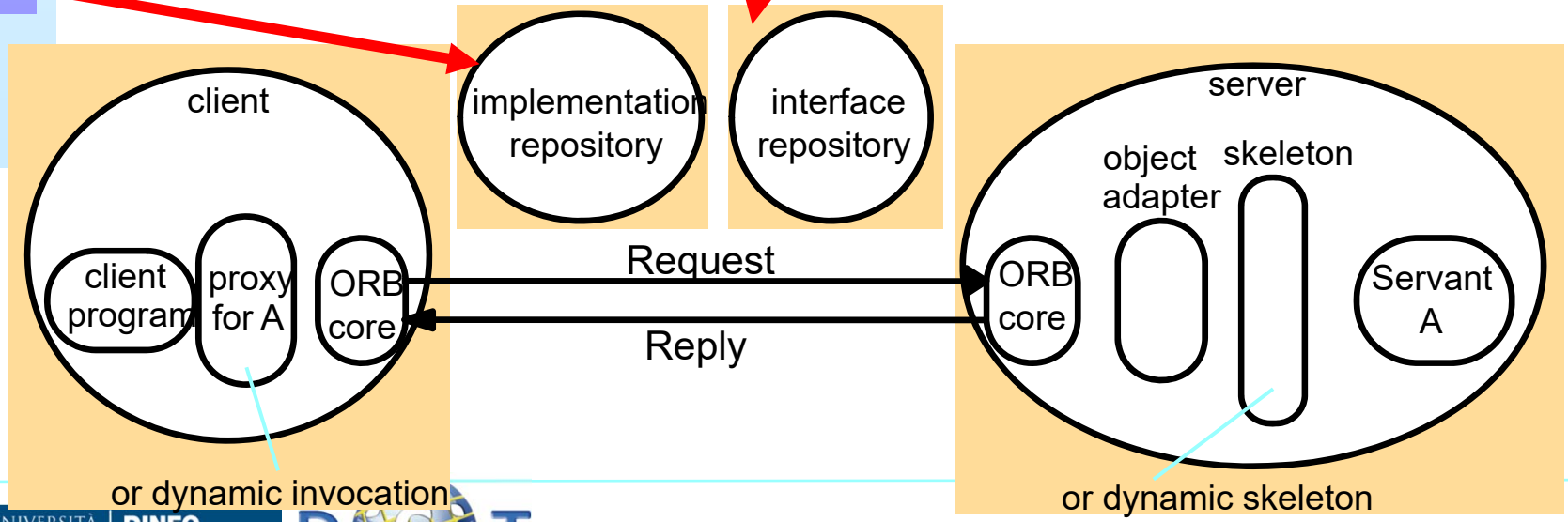
6. *Si noti che il ritorno dei parametri passa sempre dallo skeleton che gestisce anche le eventuali condizioni di eccezione e il loro marshalling verso il client.*

OA, Object Adapter

- **The Object Adapter (OA)**
  - to interface an object's implementation with its ORB.
- Three **Object Adapters**.
  - **Basic Object Adapter (BOA)**
    - Provides CORBA objects with a **common set of methods**.
      - CORBA object's interface to the ORB
      - available in every ORB implementation
    - Includes user authentication, object activation, object persistence ,etc...
  - **Library Object Adapter (LOA)**
  - **Object-Oriented Database Adapter (OODA)**
    - Both LOA and OODA are useful for accessing objects in persistent storage

The main components of the CORBA architecture

- **Interface repository**
  - the interface repository provides information about registered IDL interfaces to clients and servers that require it.
- **Implementation repository**
  - activates registered servers on demand and locates running servers
  - uses the object adapter name to register and activate servers.
  - more about this later



Interface repository

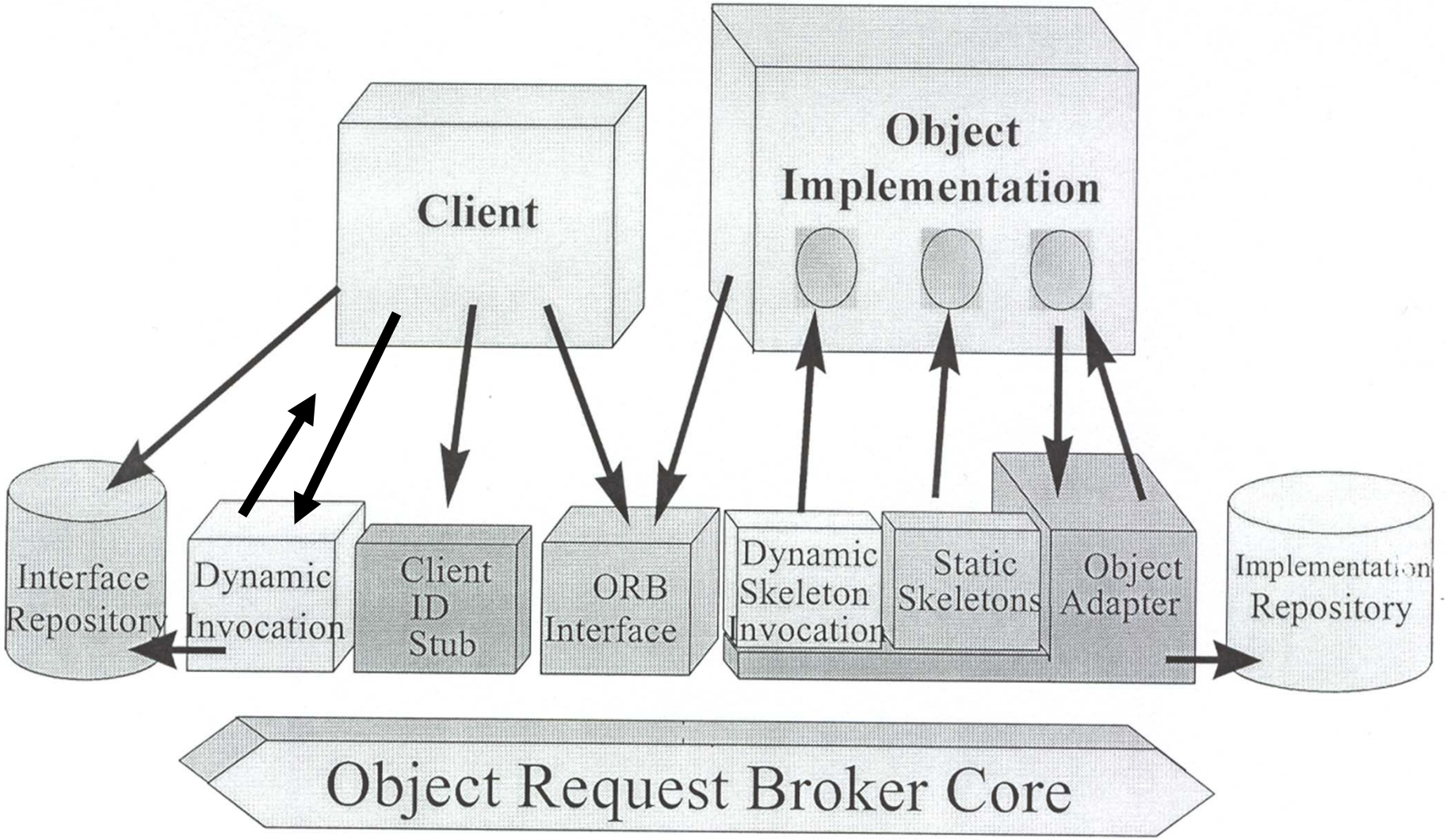
- provides information about registered IDL interfaces
  - **for each interface gives:**
    - ➔method names and the names and types of the arguments and exceptions. (a short version of “Manifesto”)
  - It is a facility for **reflection** in CORBA
    - ➔Having a remote reference to a CORBA object, it is possible to ask at the interface repository about its methods and their parameter types
    - ➔the client can use the dynamic invocation interface to dynamically construct an invocation with suitable arguments and send it to the server
- the IDL compiler gives a Type ID to each IDL type, which is
  - included in remote object references
  - used also as a Type repository ID

- Applications that use static invocation with client proxies and IDL skeletons do not require an interface repository.
  - Not all ORBs provide an interface repository.

Concetto di Manifesto

- Descrizione della classe/“component software”
- Dovrebbe includere:
  - Nome, descrizione, creatore, produttore, data, versione, sistema operativo, etc.
  - Interfaccia di uso, o interfacce per l’uso, metodi e loro signature, etc.
  - Informazioni di trading: costo, DRM, location per il download di aggiornamenti, scadenze, etc.
  - dipendenze da altri componenti, lib, dll, etc.
- In CORBA in concetto di manifesto e’ limitato:
  - L’ORB o chi per lui non e’ in grado di prendere decisioni su quali diverse implementazioni scegliere, etc.. Sulla base di informazioni disponibili
  - Le interfacce/implementazioni devono essere note agli ORB non possono arrivare dall’esterno del sistema, non possono essere caricate dinamicamente nel middleware.

ORB-Structure



ORB-invocation, Client

- **ORB Interface**
  - Identification of the objects
  - String to objects and viceversa (Marshalling/unmarshalling)
- **Client IDL Stubs**
  - Static interface to object services, precompiled stubs
- **Dynamic Invocation Interface, DII**
  - Permette di identificare i metodi che possono essere chiamati a run-time.
  - CORBA permette di identificare i metadati e l'interfaccia dei servizi degli oggetti non noti al client ma noti al MW
- **Interface Repository**
  - Database con tutte le interfacce possibili e registrate in base agli oggetti disponibili

ORB-provider part, Server

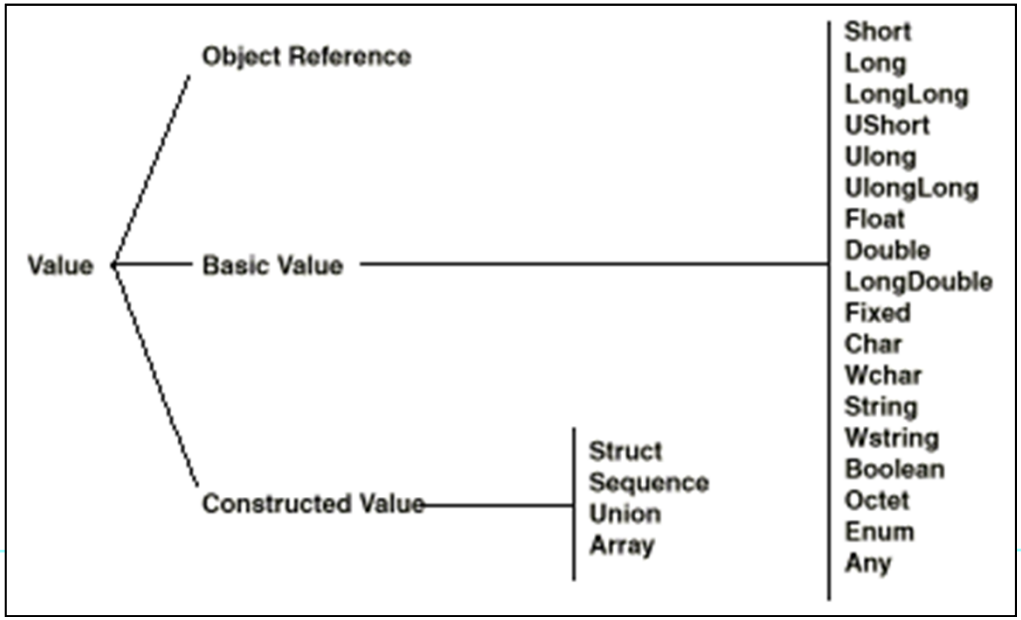
- **ORB Interface verso il Server**
  - Come quella da lato client
- **Static Skeleton equivalente al Server IDL Stubs**
  - Interfaccia statica dei servizi esportati dal server con IDL
- **Dynamic Skeleton Interface, DSI**
  - Interfaccia dynamica per la pubblicazione di servizi dinamici
  - Interface repository
- **Object Adapter / Dispatcher**
  - Accetta richieste di servizio per il Server
  - Istanzia oggetti distribuiti e gli assegna richieste
  - Fa uso del Implementation Repository
- **IR, Implementation Repository**
  - Tabelle di classi e loro ID

CORBA Characteristics

- Object-Oriented Programming
- Support multiple languages
  - Official: JAVA, C, C++, Smalltalk, COBOL
  - Also: eiffel, modula, perl, TCL, Python, etc.

What are Objects in CORBA !!

- **Objects are abstract**: not realized by any particular technology
  - An object system is a collection of objects that isolates the requestor of services (clients) from the providers of services by a well-defined **encapsulating interface**
- **Objects "talk" through requests**: operation, target object, zero or more parameters, optional request context
- **Objects are described** with interfaces
  - operations (methods)
  - attributes (properties)
  - Standard data types are supported
    - object references
    - Any



CORBA and IDL

- **Interface Definition Language (IDL)**
  - The OMG-standard language for defining the interfaces for all CORBA objects.
  - An IDL interface declares a set of operations, exceptions, and attributes.
  - Each operation has a signature, which defines its name, parameters, result and exceptions.
  - Format of messages, external data representation in CDR

- **Below there is the**
  - **Internet InterORB Protocol (IIOP)**
    The OMG-specified network protocol for communicating between ORBs fo different vendors. Based on TCP.

Interface Definition Language

- Language neutral specification

```
interface Polynomial : MathObject {
   sequence<Monomial> monomials;
   int rank;
   Polynomial add(in Polynomial p);
};
```

- Mappings to several languages
- Tools (compilers) generate stubs and skeletons in various languages

Note. No way to know at run-time which interfaces an objects provides: IDL is compiled away

Dynamic taking of an interface, but it has to be created in advance

Interface Definition Language (IDL)

- General Properties of IDL
  - Case sensitive
  - Definition syntax is the same as C++ definition syntax.
  - Assumes the existence of a C processor to process constructs such as macro definitions and conditional complication
  - An example; The Module:

```
module Bank {
        interface Customer {
        ….
        };
        interface Account {
        ….
        };
        ….
};
```

Interface Definition Language (IDL)

- Primitive types: void, Boolean, char, wchar,
- Floating point types: float, double and long double
- Integer types: long, long long, unsigned long etc.
- Constructed types: enum, struct, union, etc.
- The interface type: in, out, inout
- Attributes: readonly..
- Other IDL constructs
  - typedef
  - forward declaration
- Container types: sequence, array.
- The Exception type
- The Any type
- The TypeCode Pseudotype

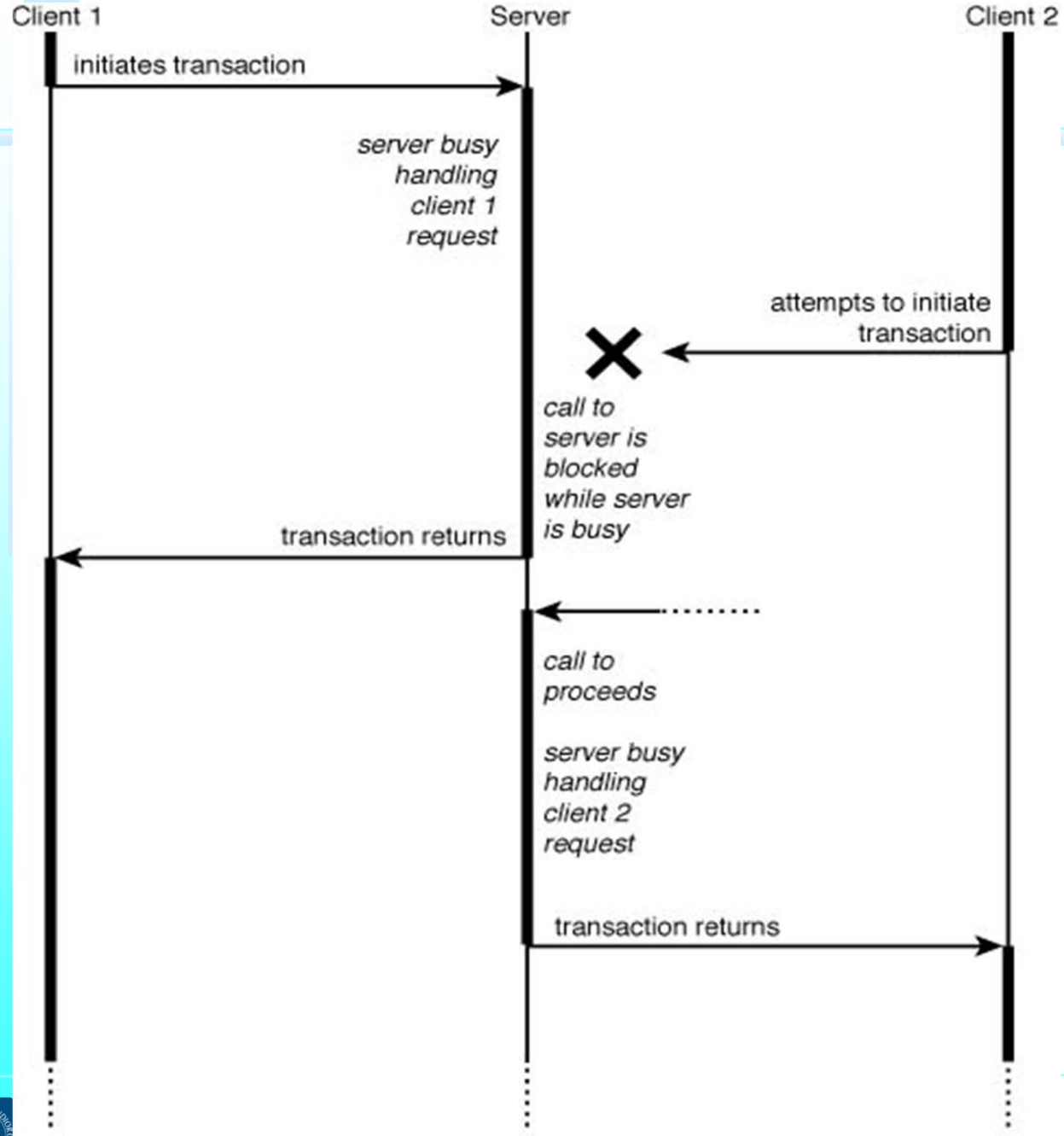
Building a CORBA Application

- Step1: Write IDL interfaces for Server: Server.idl
- Step2: Compile IDL file and generate Server_c.cpp and Server_s.ccp
- Step3: Write server implementation in C++: ServerMain.ccp
- Step4: Compile the ServerMain.cpp with the files created by IDL
- Step5: Write IDL interface for Client: Client.idl (se diversa da Server.idl)
- Step6: Compile client.idl and generate associated java files such as ServerSymbolHelper.java and ServerSymbolListHelper.java etc.
- Step7: Write client implementation in Java
- Step8: Compile client implementation and helper files together
- Step9: Run server and client programs together

CORBA Design Issues

- **Single-Threaded** Applications:
  - Common and Easy.
- **Multi-Threaded** Applications: Limited
  - Not all the Operating systems supports it.
  - Not all the developers are using the later versions of OSs.
  - Introduces new issues: the need to manage concurrent access to objects.

- Server Applications
- Client Applications
- Mixed Server/Client Applications
- Object Life Time

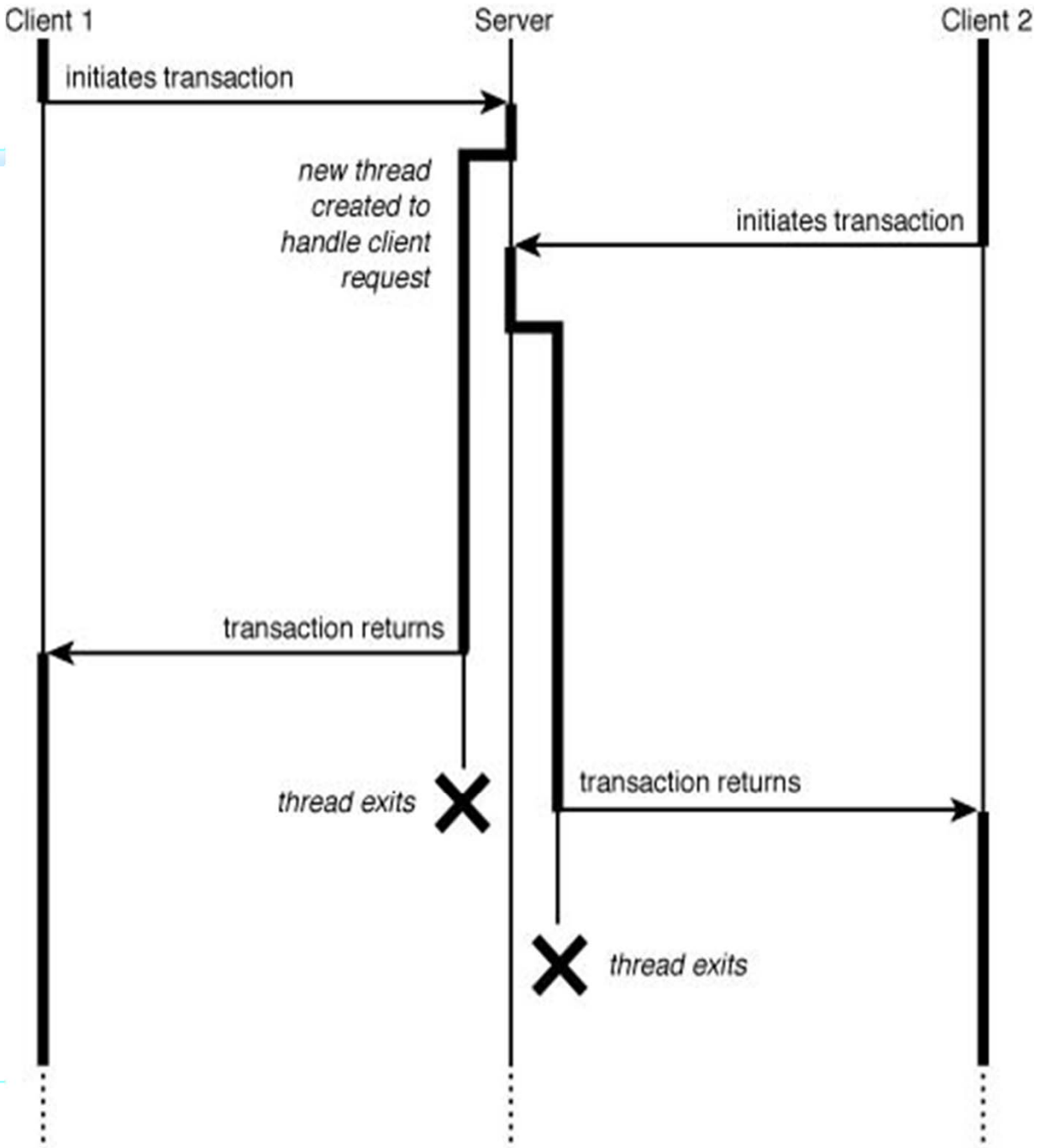


Single Thread

Serialize

Multi Thread

Concurrent



Reference

- **Visibroker, from Visigenic**
  - http://www.cse.cuhk.edu.hk/~csc5340/material/vbj40java-reference.pdf
- **Orbix from Iona**
  - http://www.cse.cuhk.edu.hk/~csc5340/material/OrbixProgrammersGuide.pdf
- **CORBA FAQ**
  - http://www.omg.org/gettingstarted/corbafaq.htm