Distributed Systems Development

Paulo Gandra de Sousa, PhD

psousa@dei.isep.ipp.pt

MSc in Computer Engineering

DEI/ISEP

Programação de Sistemas Distribuidos

Paulo Gandra de Sousa

psousa@dei.isep.ipp.pt

Mestrado em Engenharia Informática

DEI/ISEP

Disclaimer

- Parts of this presentation are from:
  - Tannembaum
  - Coulouris
  - Doug Terry (CS 294)
  - Miguel Losa (ARQSI)

Today's lesson

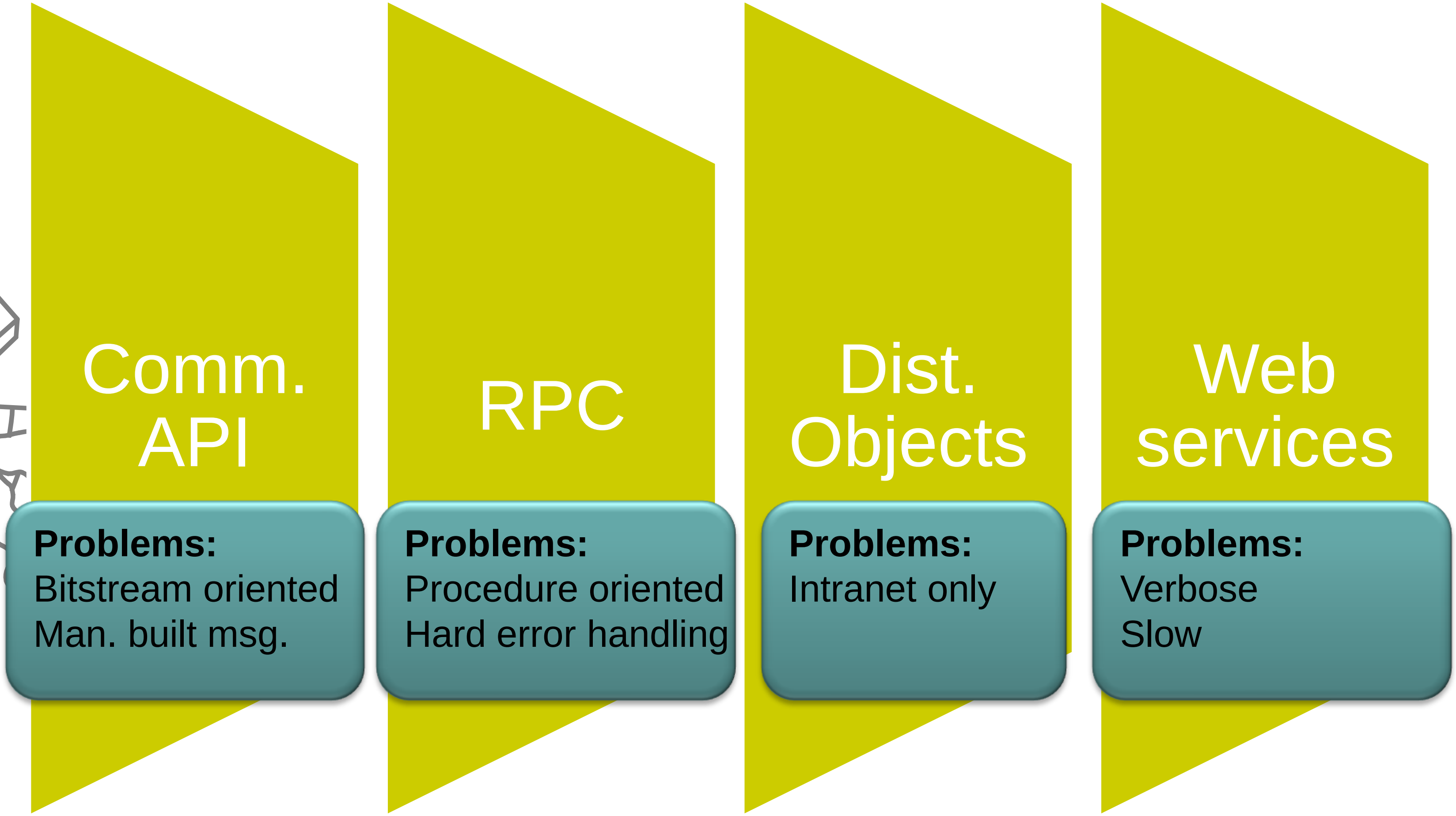
- Communication
  - APIs
  - Web services

COMMUNICATION

Communication APIs

- Sockets
- MPI
- RPC
- Remote objects
  - CORBA, DCOM
  - Java RMI, .net remoting
- SOAP and Web services

“History”



Comm. API

Problems:
Bitstream oriented
Man. built msg.

RPC

Problems:
Procedure oriented
Hard error handling

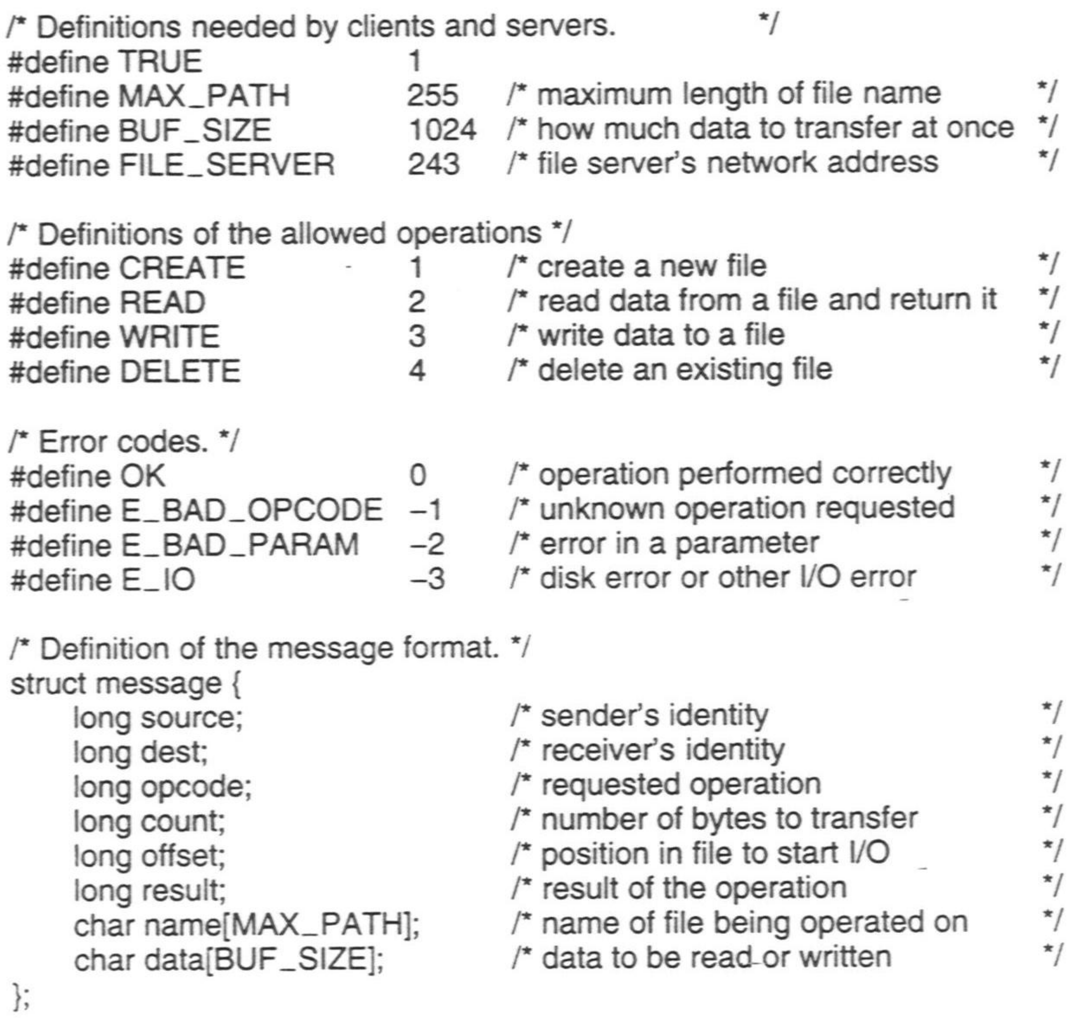
Dist. Objects

Problems:
Intranet only

Web services

Problems:
Verbose
Slow

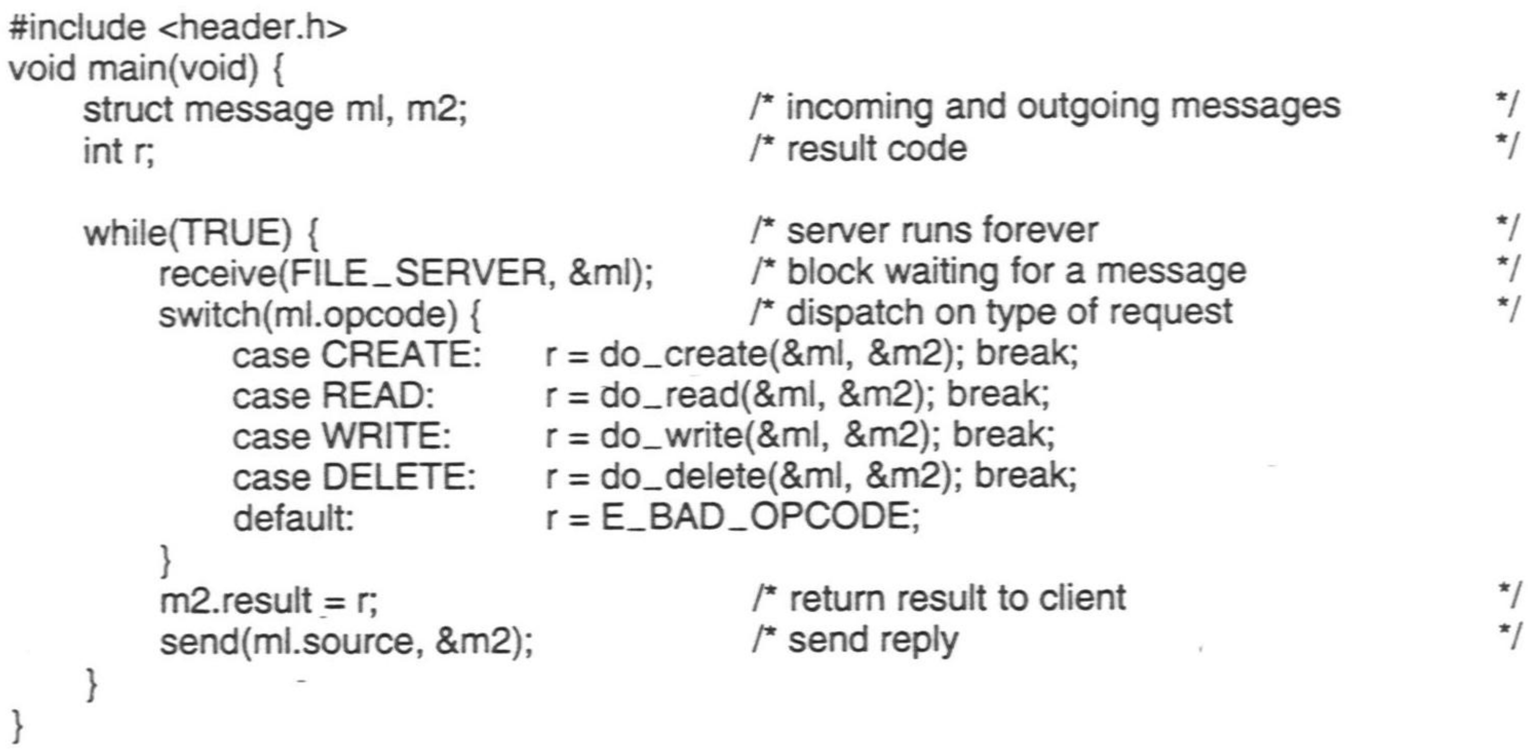
An Example Client and Server (1)



```
/* Definitions needed by clients and servers.                  */
#define TRUE                1
#define MAX_PATH            255    /* maximum length of file name          */
#define BUF_SIZE            1024   /* how much data to transfer at once    */
#define FILE_SERVER         243    /* file server's network address        */

/* Definitions of the allowed operations */
#define CREATE              1      /* create a new file                    */
#define READ                2      /* read data from a file and return it  */
#define WRITE               3      /* write data to a file                 */
#define DELETE              4      /* delete an existing file              */

/* Error codes. */
#define OK                  0      /* operation performed correctly        */
#define E_BAD_OPCODE        –1     /* unknown operation requested          */
#define E_BAD_PARAM         –2     /* error in a parameter                 */
#define E_IO                –3     /* disk error or other I/O error        */

/* Definition of the message format. */
struct message {
      long source;                 /* sender's identity                    */
      long dest;                   /* receiver's identity                  */
      long opcode;                 /* requested operation                  */
      long count;                  /* number of bytes to transfer          */
      long offset;                 /* position in file to start I/O        */
      long result;                 /* result of the operation              */
      char name[MAX_PATH];         /* name of file being operated on       */
      char data[BUF_SIZE];         /* data to be read or written           */
};
```

Header file used both by server and client

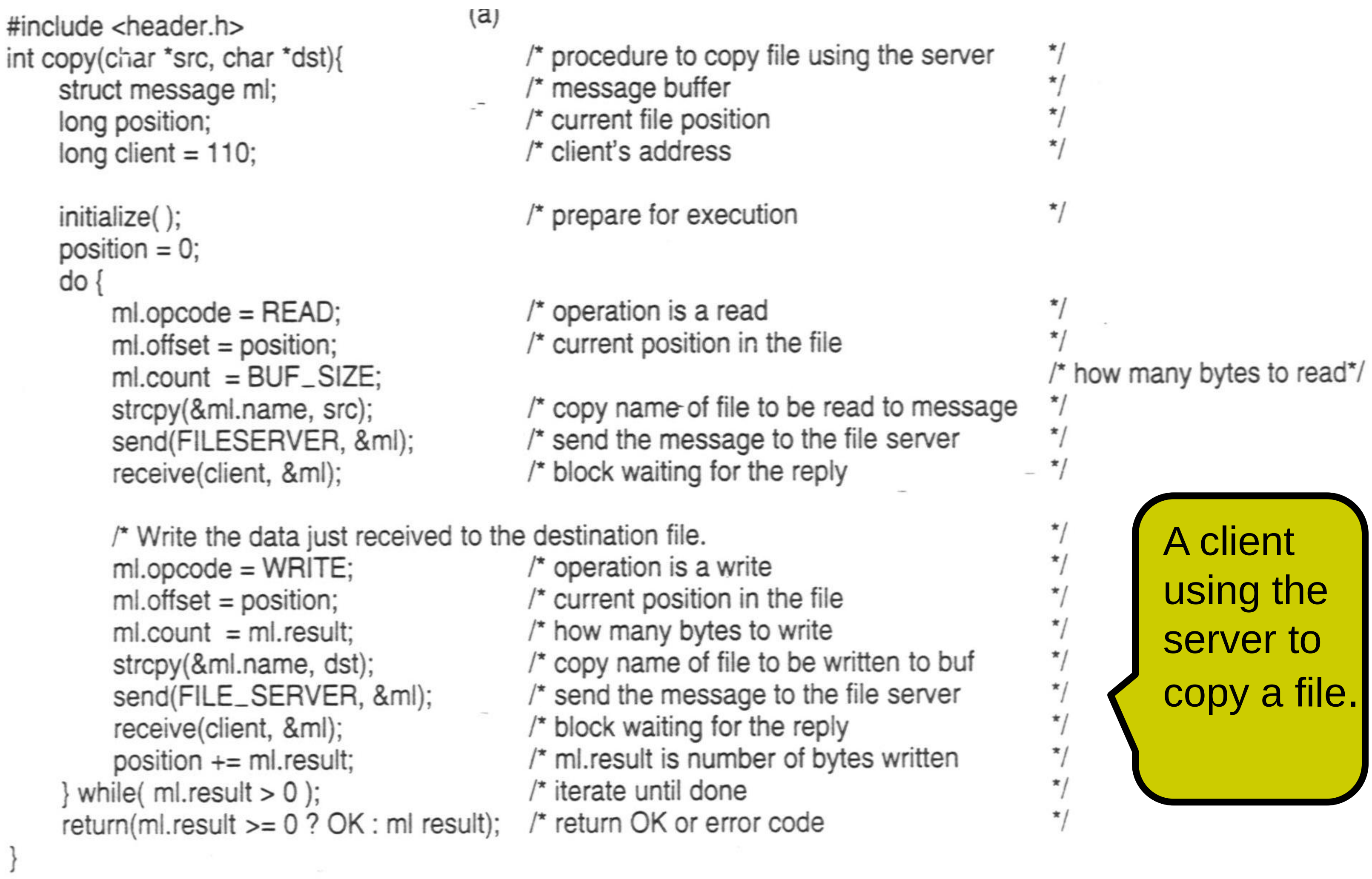
An Example Client and Server (2)



```
#include <header.h>
void main(void) {
      struct message ml, m2;                      /* incoming and outgoing messages        */
      int r;                                      /* result code                           */

      while(TRUE) {                               /* server runs forever                   */
            receive(FILE_SERVER, &ml);            /* block waiting for a message           */
            switch(ml.opcode) {                   /* dispatch on type of request           */
                  case CREATE:   r = do_create(&ml, &m2); break;
                  case READ:     r = do_read(&ml, &m2); break;
                  case WRITE:    r = do_write(&ml, &m2); break;
                  case DELETE:   r = do_delete(&ml, &m2); break;
                  default:       r = E_BAD_OPCODE;
            }
            m2.result = r;                        /* return result to client               */
            send(ml.source, &m2);                 /* send reply                            */
      }
}
```

A sample server

An Example Client and Server (3)



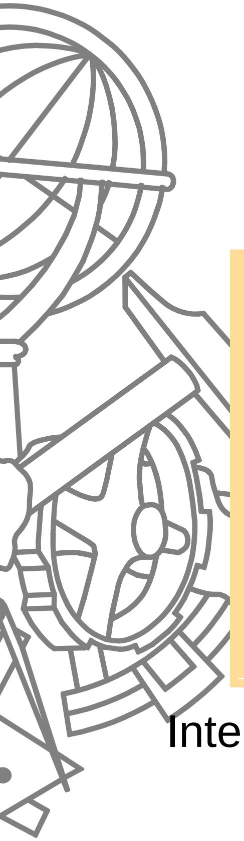
```
                                  (a)
#include <header.h>
int copy(char *src, char *dst){                /* procedure to copy file using the server      */
     struct message ml;                        /* message buffer                               */
     long position;                            /* current file position                        */
     long client = 110;                        /* client's address                             */

     initialize( );                            /* prepare for execution                        */
     position = 0;
     do {
          ml.opcode = READ;                    /* operation is a read                          */
          ml.offset = position;                /* current position in the file                 */
          ml.count  = BUF_SIZE;                                                               /* how many bytes to read*/
          strcpy(&ml.name, src);               /* copy name of file to be read to message      */
          send(FILESERVER, &ml);               /* send the message to the file server          */
          receive(client, &ml);                /* block waiting for the reply                  */

          /* Write the data just received to the destination file.                             */
          ml.opcode = WRITE;                   /* operation is a write                         */
          ml.offset = position;                /* current position in the file                 */
          ml.count  = ml.result;               /* how many bytes to write                      */
          strcpy(&ml.name, dst);               /* copy name of file to be written to buf       */
          send(FILE_SERVER, &ml);              /* send the message to the file server          */
          receive(client, &ml);                /* block waiting for the reply                  */
          position += ml.result;               /* ml.result is number of bytes written         */
     } while( ml.result > 0 );                 /* iterate until done                           */
     return(ml.result >= 0 ? OK : ml result);  /* return OK or error code                      */
}
```

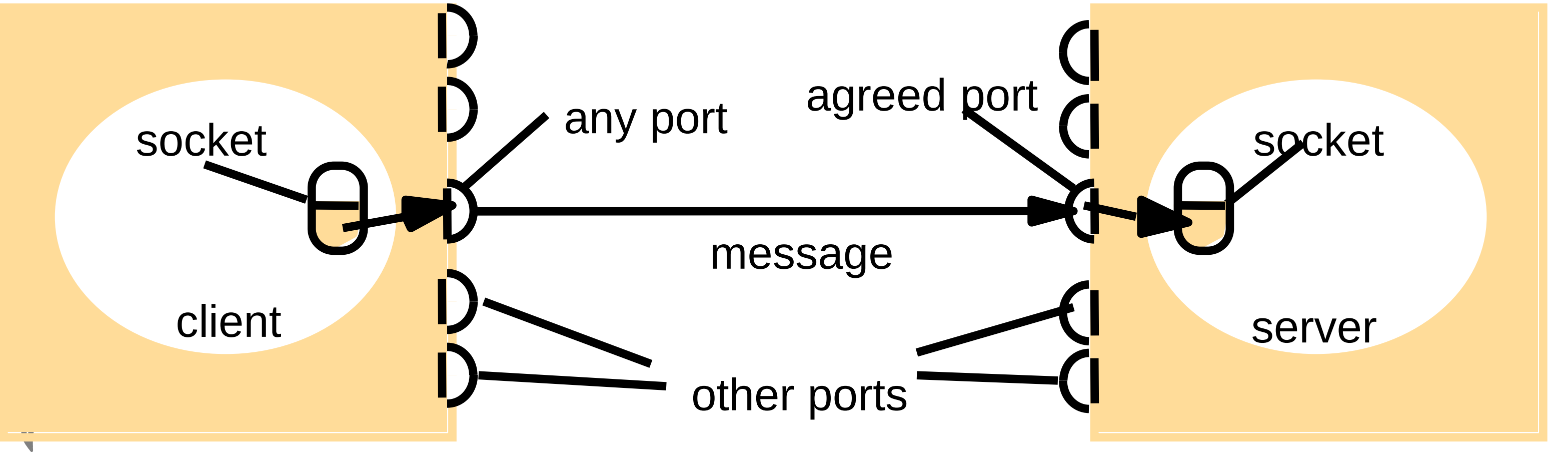
A client using the server to copy a file.

sockets

- Originally in BSD unix (1983)
- Adopted as *de facto* standard for TCP/IP communications
  - Windows
  - Several unix OS
  - IBM OS/400

Sockets and ports





socket

client

any port

agreed port

message

other ports

socket

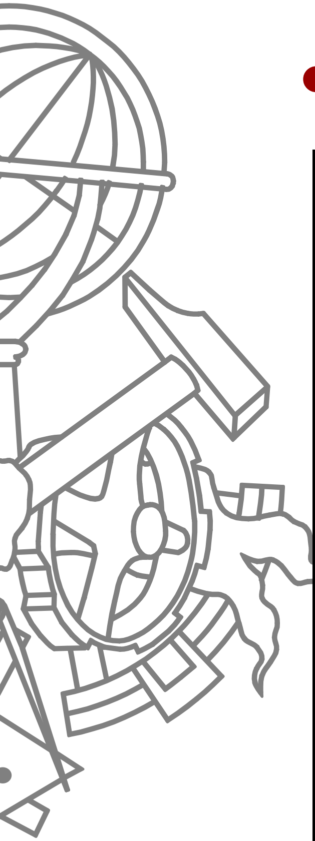
server

Internet address = 138.37.94.248

Internet address = 138.37.88.249

Berkeley Sockets (1)

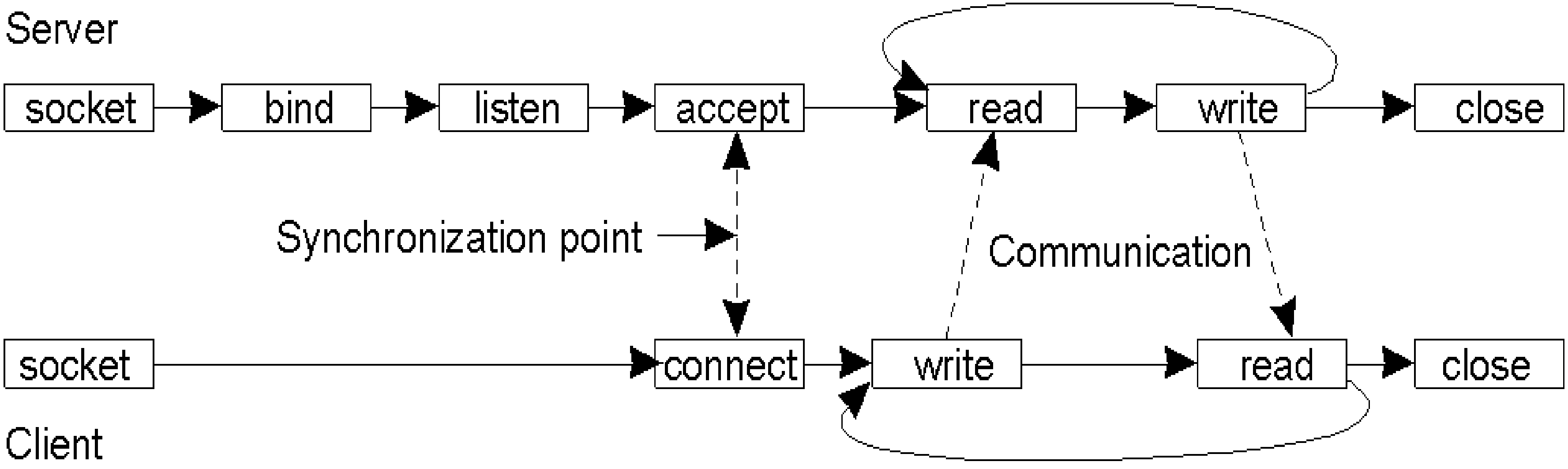
- Socket primitives for TCP/IP.



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.

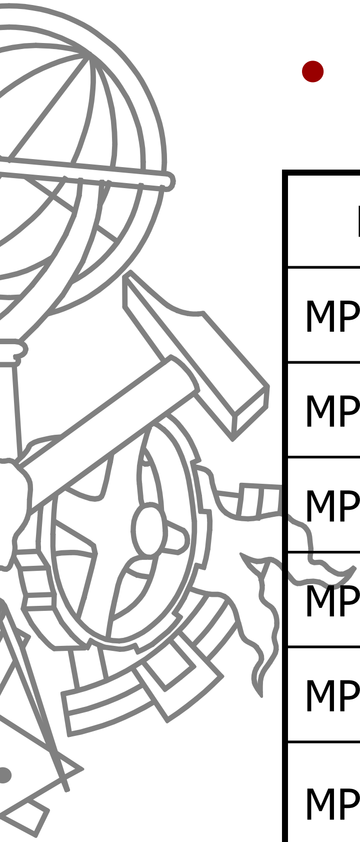


Bit stream oriented

- ssize_t send(int *socket*, const void **buffer*, size_t *length*, int *flags);*
- ssize_t recv(int *socket*, void **buffer*, size_t *length*, int *flags*);

- Must care for
  - Buffer handling (overflow, memory allocation, …)
  - Internal representation of data when connecting two different hardware nodes

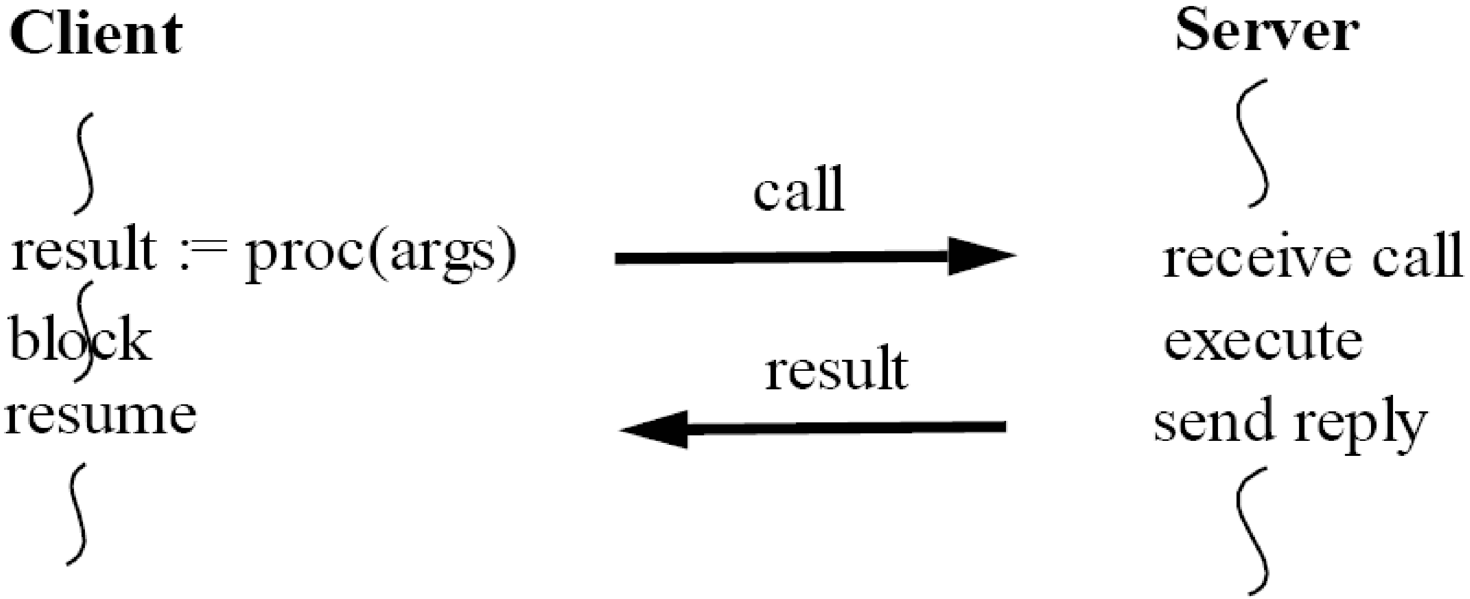
The Message-Passing Interface (MPI)



- Some of the most intuitive message-passing primitives of MPI.

Primitive	Meaning
MPI_bsend	Append outgoing message to a local send buffer
MPI_send	Send a message and wait until copied to local or remote buffer
MPI_ssend	Send a message and wait until receipt starts
MPI_sendrecv	Send a message and wait for reply
MPI_isend	Pass reference to outgoing message, and continue
MPI_issend	Pass reference to outgoing message, and wait until receipt starts
MPI_recv	Receive a message; block if there are none
MPI_irecv	Check if there is an incoming message, but do not block

RPC



Client

result := proc(args)

block

resume

call

result

Server

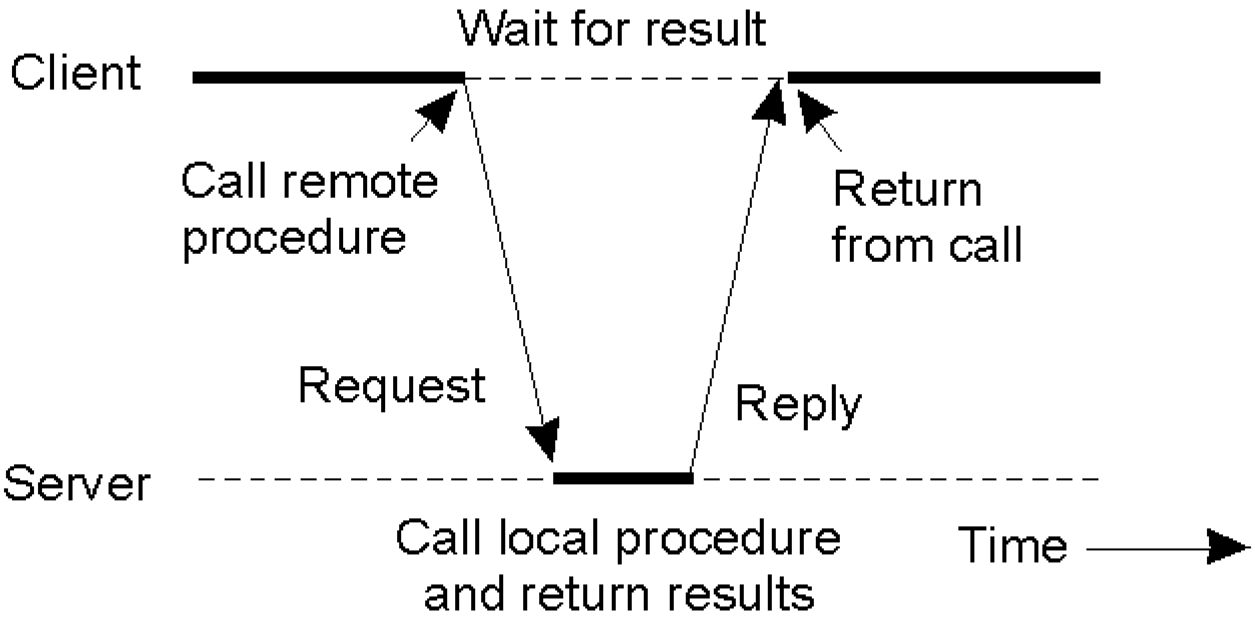
receive call

execute

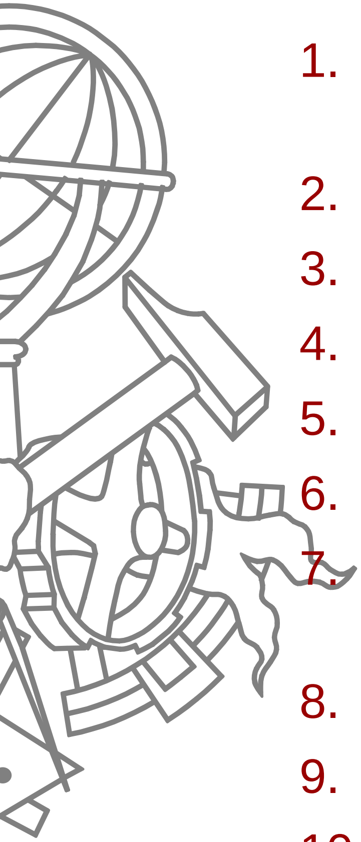
send reply

Client and Server Stubs

- Principle of RPC between a client and server program.



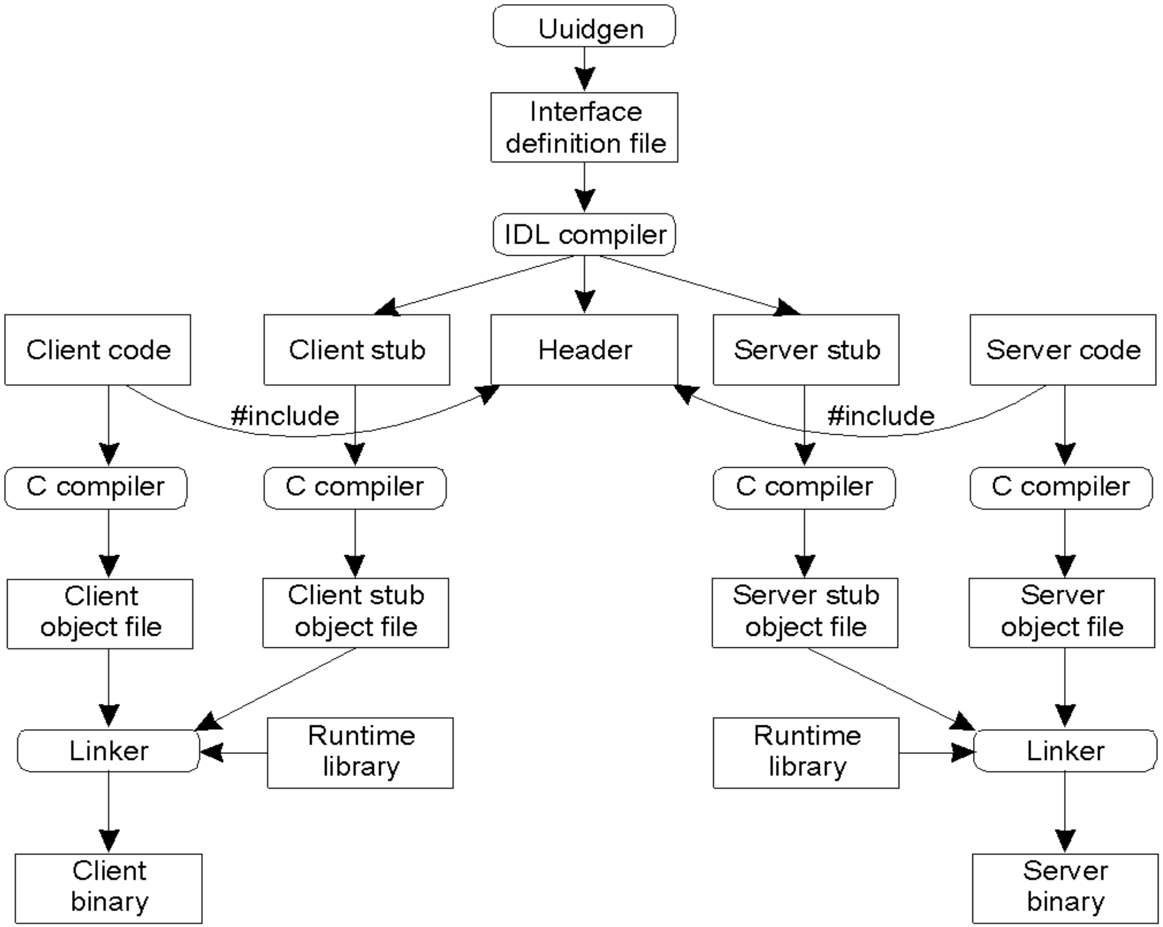
Steps of a Remote Procedure Call



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

Writing a Client and a Server

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

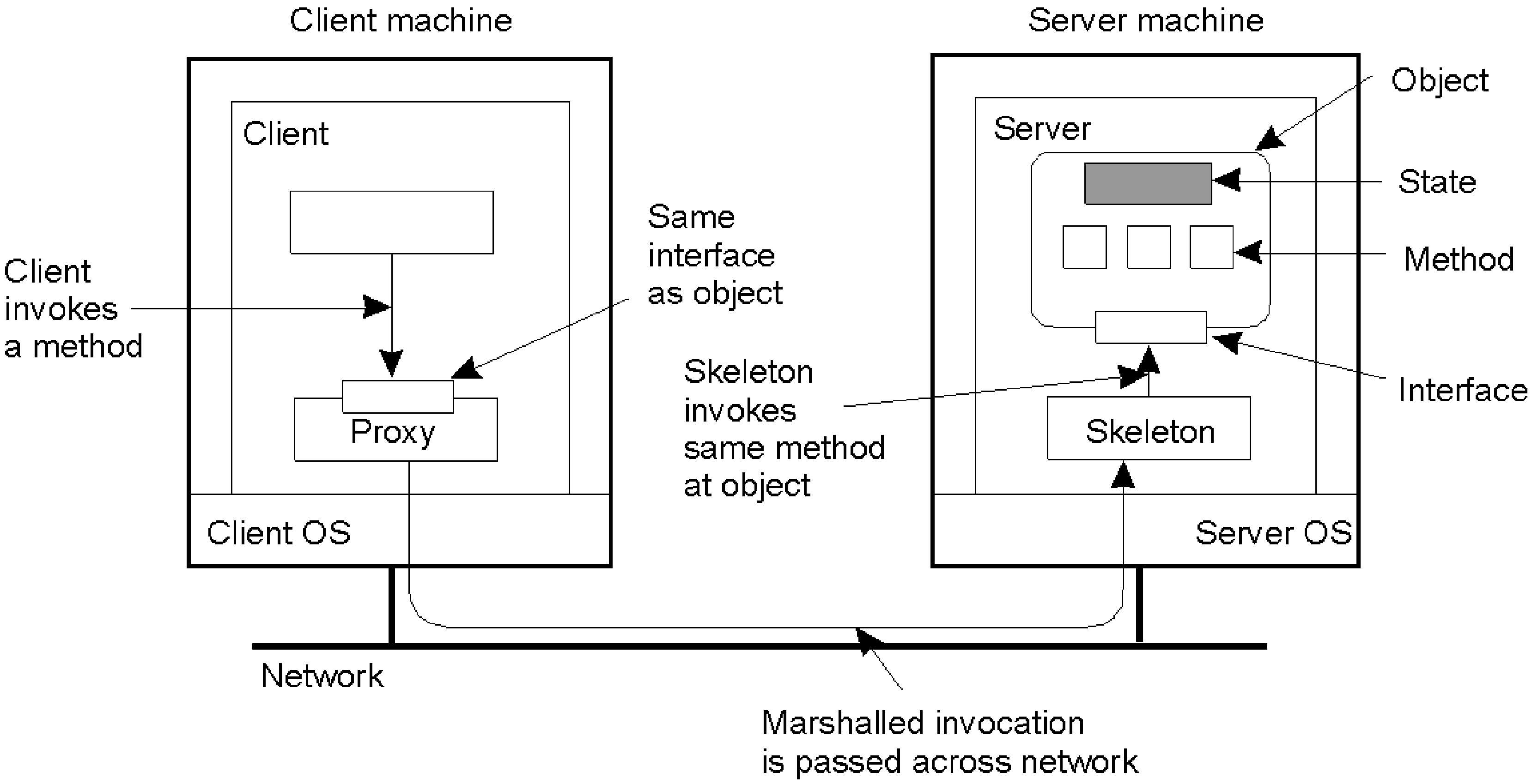


Distributed objects

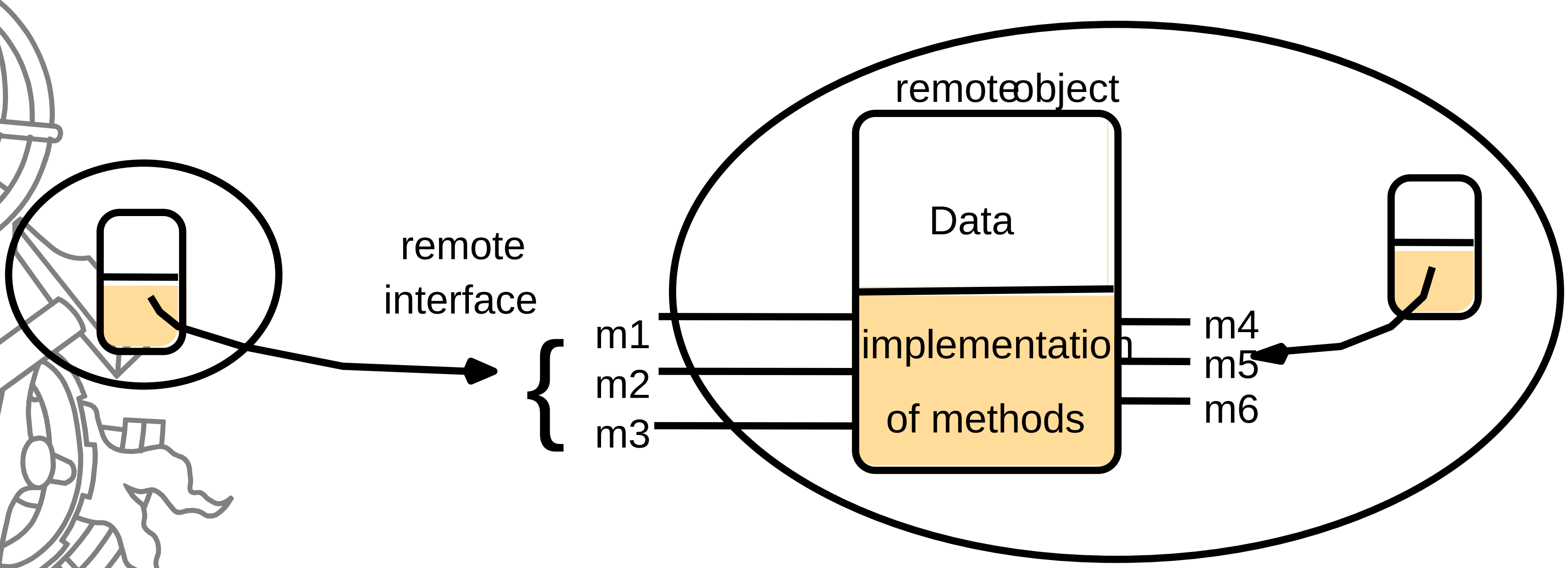
- Component based / Object Oriented on the network
- Handles object activation and access transparently
  - Mascarades error handling
  - Hides latency issues

Distributed Objects

- Common organization of a remote object with client-side proxy.

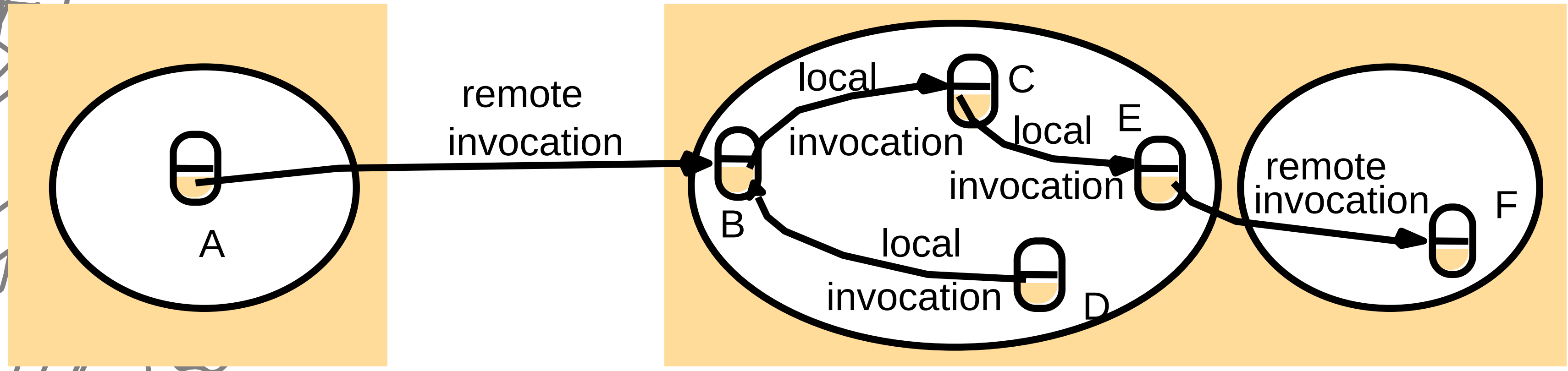


A remote object and its remote interface

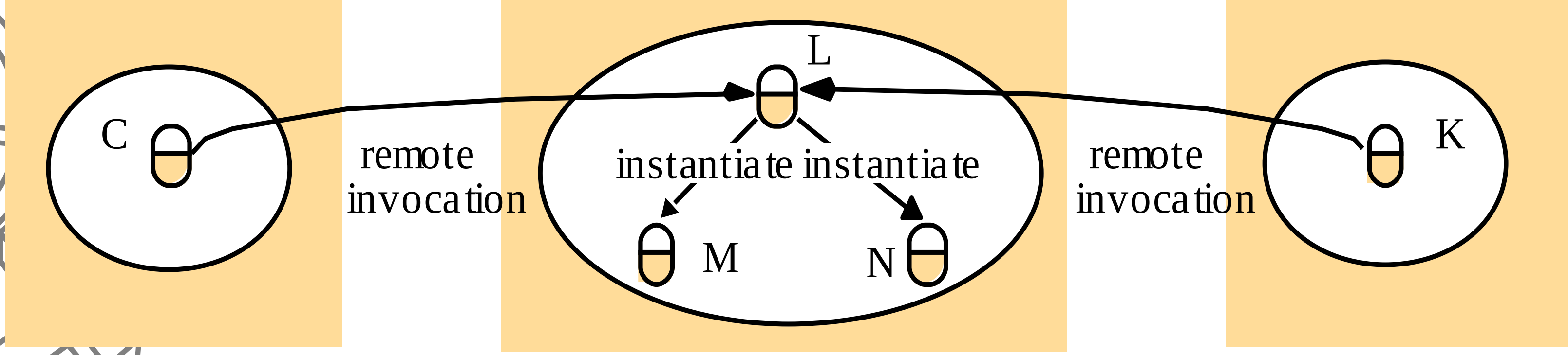


Instructor's Guide for Coulouris, Dollimore
and Kindberg Distributed Systems:
Concepts and Design Edn. 4

Remote and local method invocations

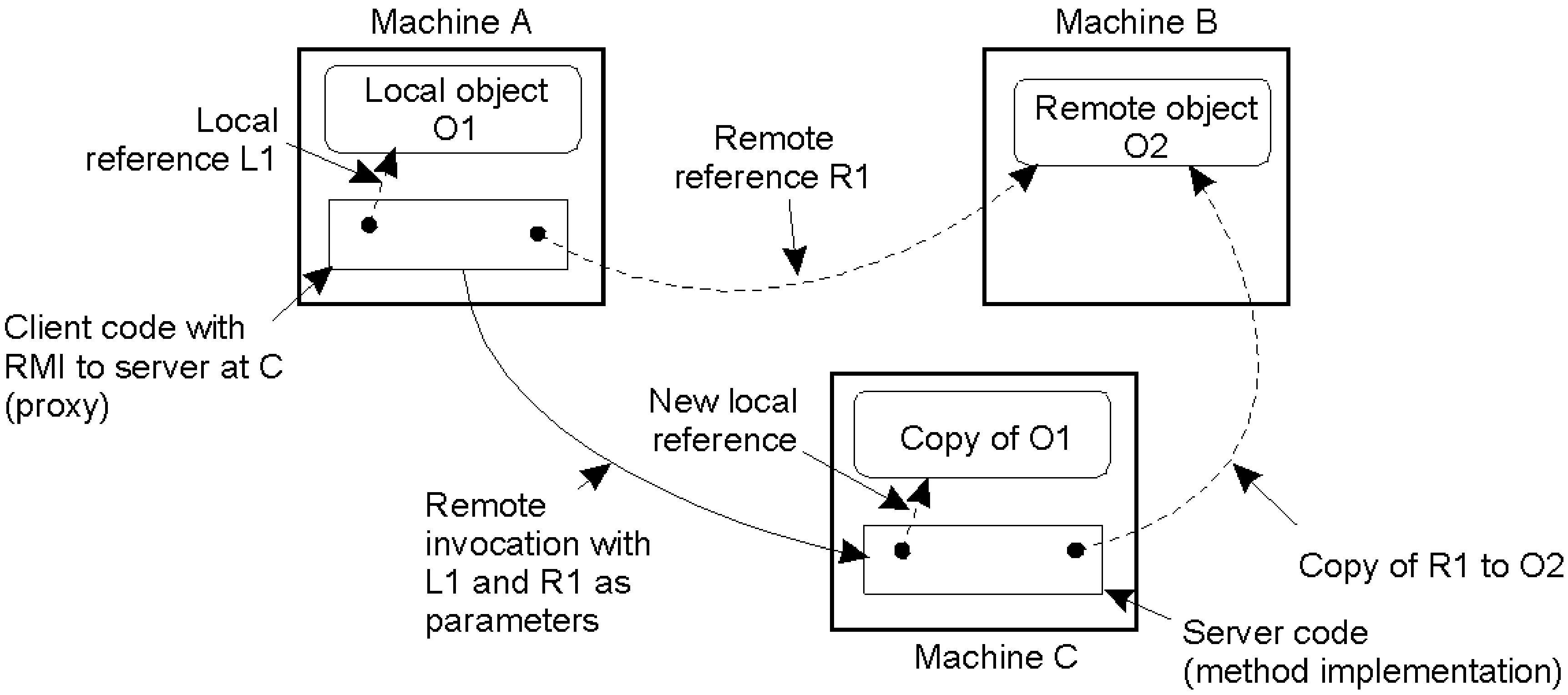


Instantiation of remote objects



Parameter Passing

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



Distributed objects

- CORBA
  - OMG
  - Based on the concept of Object Request Broker
- DCOM
  - Microsoft
  - Binary based interface compatibility
- Java RMI
  - Sun Java
  - Allows access to remote Java objects
- .Net Remoting
  - Microsoft
  - Allows access to remote .net objects
- EJB
  - Sun Java
  - Enterprise components for the Java platform
- .net enterprise services
  - Microsoft
  - Enterprise components for the .net platform

Web services

- What happened circa 1990?

The Internet

- Everybody wanted to make RPC over the internet. What were the problems?
  - Firewalls, …

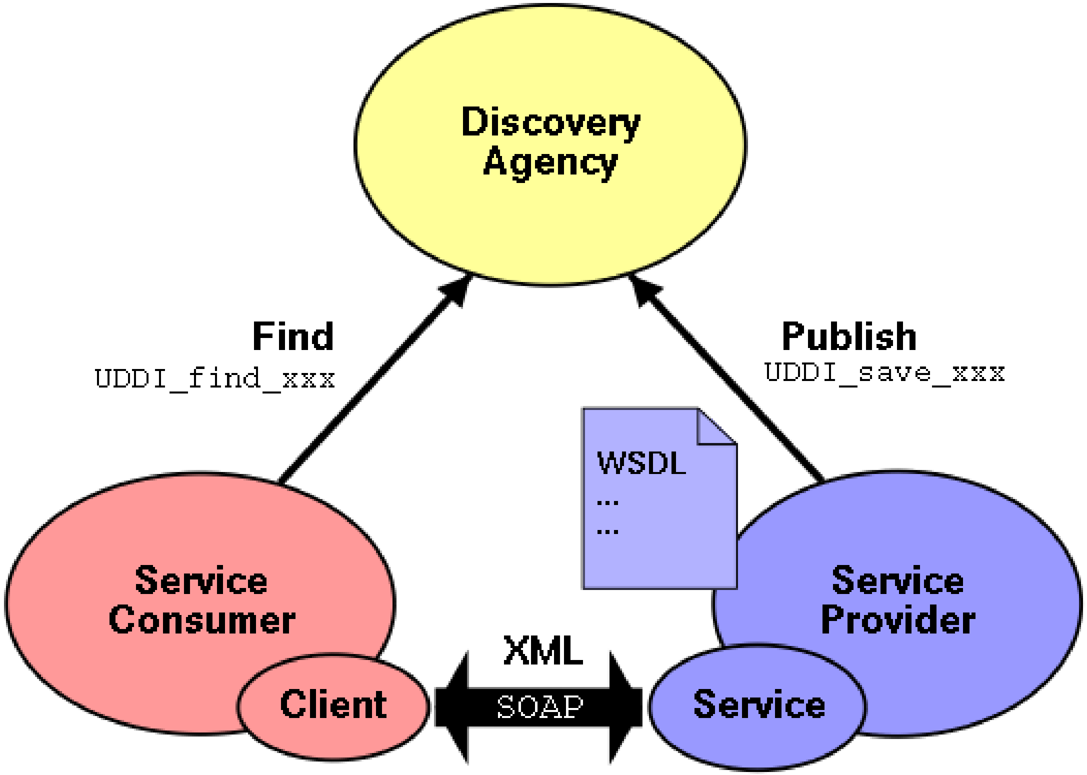
SOAP

- XML-RPC sucessor
- Independet of transport protocol (binding)
- Most common binding: HTTP
  - Thru firewalls
- Concept of Envelope
  - Header + payload

Web services

- Three key standards:
  - Universal Description, Discovery, and Integration (UDDI)
  - Web Services Description Language (WSDL)
  - Simple Object Access Protocol (SOAP)
    - RPC based on XML and HTTP
    - extended by other WS-standards
- Supported by IBM, SUN, Microsoft, …

Web Services

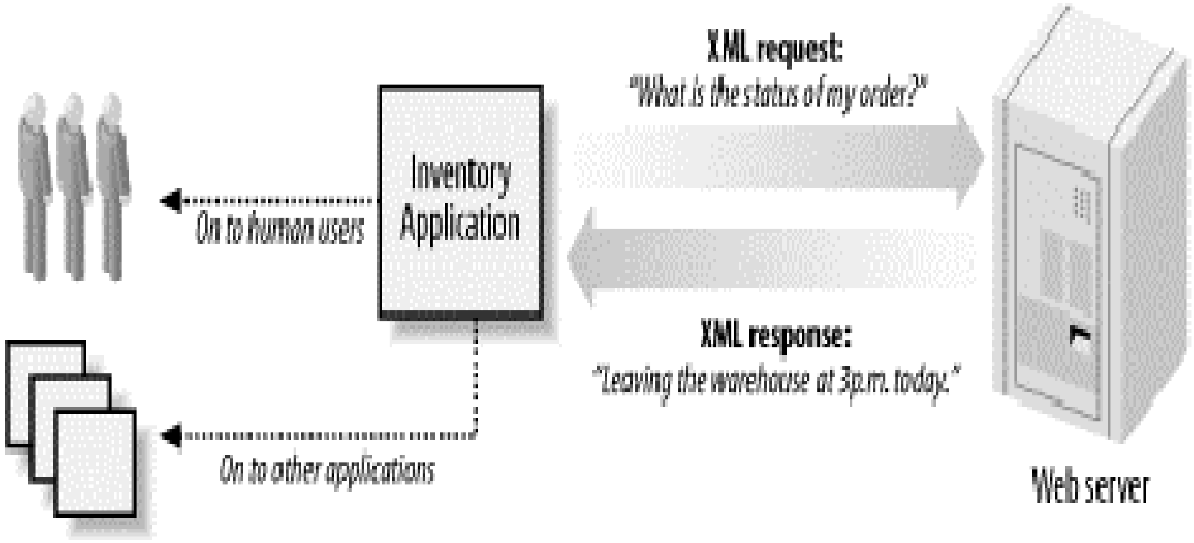


Web Services

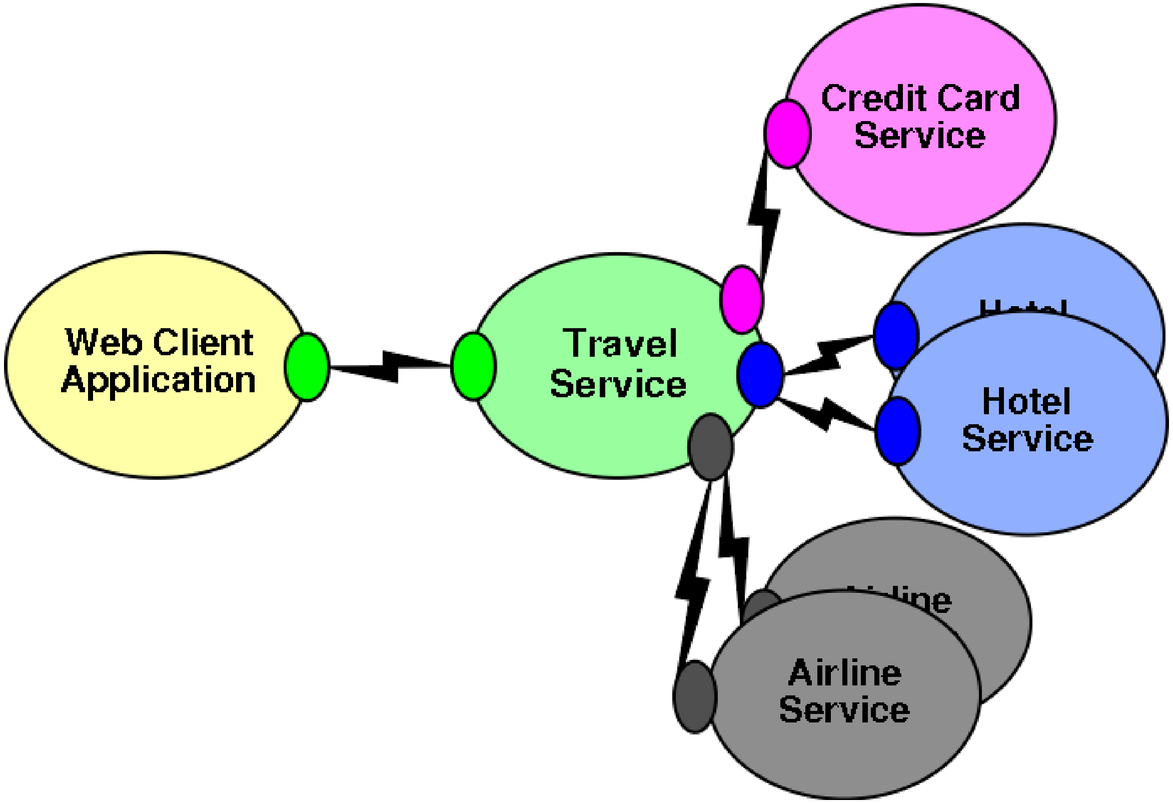
- A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards.

  - http://www.w3.org/TR/2004/NOTE-ws-arch-20040211

Web Services



Sample scenario



Issues

- Interoperability
  - Diferent implementation technologies/vendors
  - Diferent data types
- Aditional funcionalities
  - Security
  - Reliability
  - …

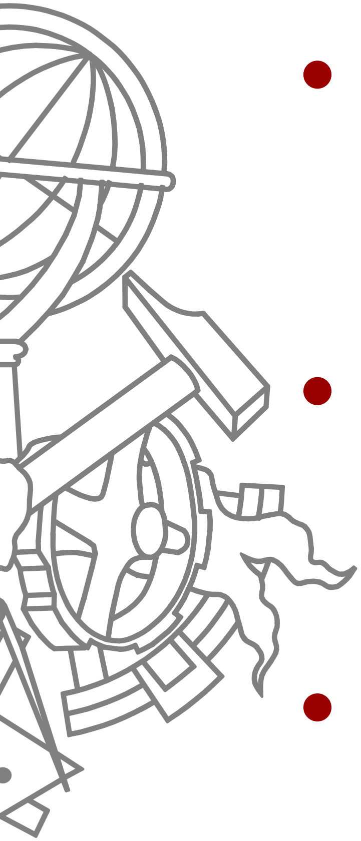
WS-*

- WS-Addressing (W3C)
  - Allows routing of messages based on header metadata and not TCP/IP endpoints
- WS-Security (OASIS)
  - Cyphered messages and headers, signed messages
    - WS-Trust
- WS-ReliableMessaging (OASIS)
  - Message delivery guarantee
  - WS-Reliability
- WS-Policy (W3C)
- WS-Coordination
  - WS-Transaction, WS-AtomicTransaction
- …

WS-I

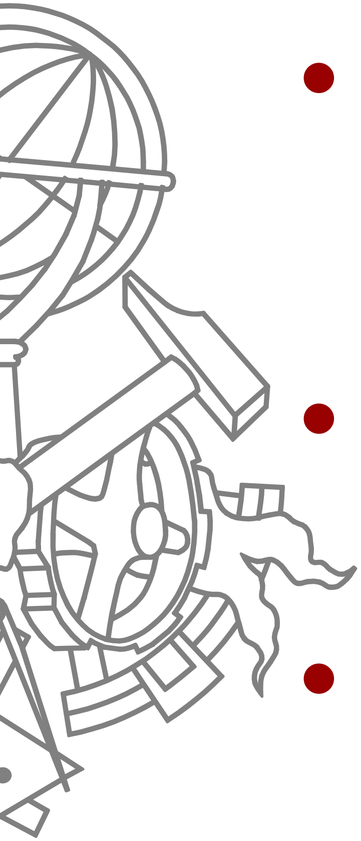
- Web Services Interoperability
- www.ws-i.org
- Suported by major vendors
- Uses open standards
- Defines profiles for interoprability
  - Basic
    - SOAP, WSDL, UDDI, attachments, WS-Addressing
  - Security
  - Reliable secure

Exercise



- Remember the example DS you provided in the last session.
- What kind of communication API (do you think) it uses?
- Would there be advantages in using another kind?

Exercise



- Can you imagine a scenario with mixed communication API?
- What would be the advantages opf such scenario?
- What kind of problems would rise?

Bibliography

- Chapter 2 Tanenbaum
- Chapter 2 & 4 Coulouris
- http://en.wikipedia.org/wiki/Inter-process_communication
- http://en.wikipedia.org/wiki/Distributed_object
- http://en.wikipedia.org/wiki/Web_service
- http://en.wikipedia.org/wiki/Enterprise_service_bus
- http://en.wikipedia.org/wiki/Loose_coupling

Suggested readings

- http://en.wikipedia.org/wiki/SOAP
- http://en.wikipedia.org/wiki/Distributed_Component_Object_Model
- http://en.wikipedia.org/wiki/CORBA
- http://en.wikipedia.org/wiki/.NET_Remoting
- http://en.wikipedia.org/wiki/Java_RMI
- http://en.wikipedia.org/wiki/XML-RPC