

Remote Procedure Call (RPC)

Complexity of the distributed applications

- The programming of distributed applications is difficult. In addition to the usual tasks, programmers who build clients and servers must deal with the complex issues of communication.
- Although many of the needed functions are supplied by a standard API such as the socket interface, the socket calls require the programmer to specify many low level details as names ,addresses,protocols and ports.
- Moreover, asynchronous communications models are complex to implement.
- Distributed implementations tend to use the same standard API (e.g.,the socket interface). As a consequence most of the detailed code found in one program is replicated in others

-
- For example, all client programs that use a **connection oriented transport** must create a socket, specify the server's endpoint address, open a connection to the server, send requests, receive responses and close the connection when interaction is complete.
 - Tools (software that generates all or part of a computer program) have been created to construct clients and servers.
 - Tools cannot eliminate all programming: a programmer must supply the code that perform the computation for the particular service. **However, a tool can handle the communication details.**
 - As a result the code contains fewer bugs.

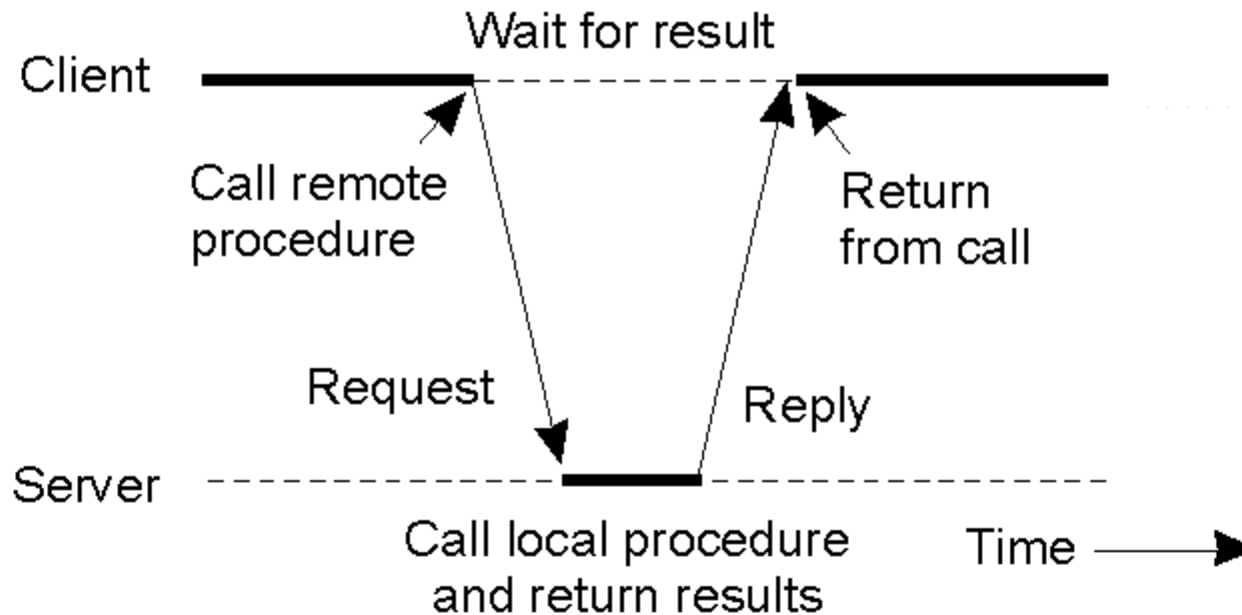
-
- **RPC = Remote Procedure Call**
 - The basic model has been proposed by **Birrell e Nelson** in 1984.
 - The essence of this technique is to allow programs on different machines to interact using **simple procedure call/return semantics**, just as if the two programs were in the same computer
 - The popularity of this approach is due to:
 - The procedure call is a widely accepted, used and understood abstraction.
 - The use of RPC enables remote interfaces to be specified as a set of named operations with designed types
 - **Because a standardized and precisely defined interface is specified**, the communication code for a an application can be generated automatically

Local Procedure Call

- **Semantic of a procedure call.** When a program calls a procedure:
 - the parameters of the call are made available to the procedure -
 - the control is transferred to the procedure;
 - when the procedure completes its execution, the results are made available to the program and the control of the CPU is returned to it.
- When programmers build a program, they begin by dividing the program into major pieces . A procedure is associated with each of them.
- If the resulting tasks require a large amount of code, the programmer subdivides the task, and uses a subprocedure to perform each of the subtasks.
- The overall form of a program consist of a hierarchy of procedure calls.

-
- Because conventional programming languages do not allow procedure calls to pass from a program on one computer across a network to a program on another, **tools are used to help programmers build client-server software using the procedure call abstraction.**
 - After deciding which procedures to place in the server (**remote procedures**) and which to place in the client (**local procedures**), **a RPC tool can be used.**
 - If a programmer follows the same procedure call paradigm used to build conventional programs when building client and server software, the programmer will find the task easier and will make fewer mistakes

RPC model

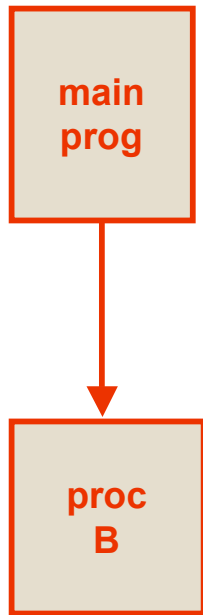


- As in conventional procedure calls, when a client calls a remote procedure, the **client will block** until a reply is returned

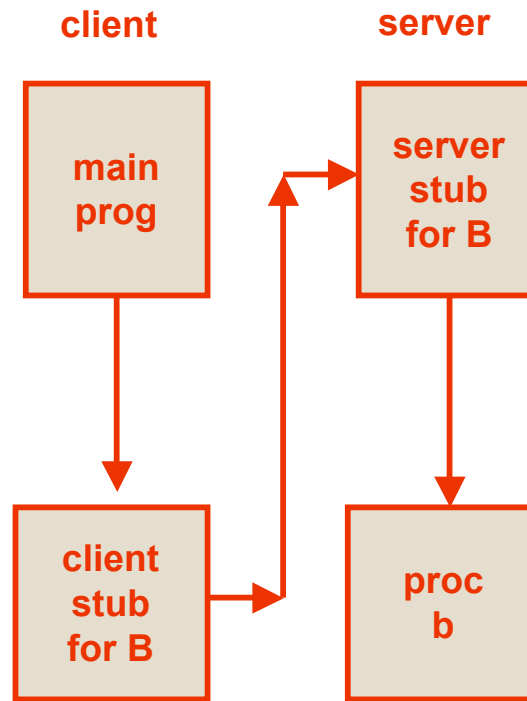
Communications Stubs

- **Extra software (RPC Tools)** must be added to each part of the program to implement the interaction.
- In the client side the extra software handles the details of sending a message across the network and waiting for a response. On the server side, receiving the message, calling the specified procedure and sending a response.
- The extra software is called as a **communication stub or proxy**.
- If a call is made to a procedure that is not local, a **communication stub intercepts** the procedure call, gathers values for the arguments (marshaling) and send a message across the network to the **communication stub on the server**

-
- The communication stub on the server site uses the conventional procedure call mechanism to invoke the specified procedure and sends the result back to the client stub.
 - When the client stub receives the response it returns the result to its caller exactly as if a local procedure was returning.
 - The following figure (a) shows the procedure call in the original program before RPC stubs are added.
 - When the main program call procedure B, the arguments it passes must agree exactly with the formal parameters of B.
 - Figure (b) shows the communication stubs that must be added to the program when it is divided into client and server pieces













(a)



(b)

-
- It is important to note that the procedural interfaces in part b) use the same number and type of arguments as the original interface in a).
 - The call from main to the client stub and the call from the server stub to procedure B use exactly the same interface as the conventional call from main to procedure B.
 - Each client stub acts as an exact replacement for one of the procedures in the original program.
 - Client stubs can be given the same name as the procedure they replace
 - As a result , code from the original program need not be changed

A remote procedure call occurs in the following steps:

-  The client procedure calls the client stub in the normal way (**local call**).
-  **The client stub builds a message** and traps to the local O.S.
-  The local O.S. sends the message to the remote O.S.
-  The remote O.S. gives the message to the server stub.
-  **The server stub unpacks** the parameters and calls the server.
-  The server does the work and returns the result to the stub.
-  The server stub packs it in a message and traps to the O.S.
-  The remote O.S. sends the message to the client O.S.
-  The client' O.S. gives the message to the client stub.
-  The stub unpacks the result and returns to the client

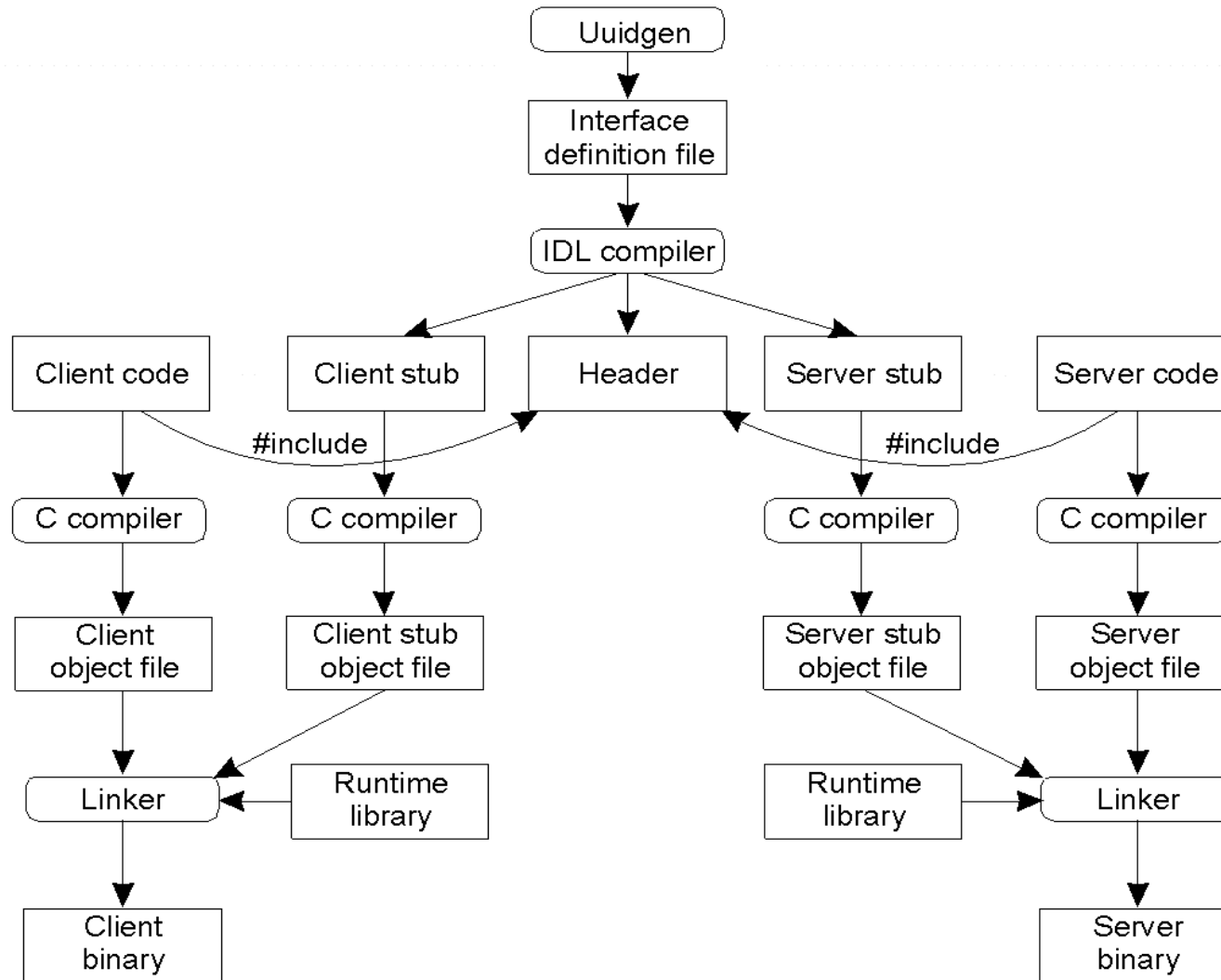
I problemi di RPC

- As long the client and the server machines are identical and all the parameters and results are scalar types (integer, characters and boolean) this model works fine.
- However, in a large distributed system, it is common that multiple machine type are present. Each machine has its own representation for numbers, characters and others data items.
- Problems can occur with the representation of integers (1s complement versus 2s complement) and especially with floating points numbers.
- Some machines, such as the INTEL, number their bytes from right to left (little endian format), whereas others, such as the Sun SPARC, number them the other way (big endian format).
- A widely accepted method defines a standard external representation and requires each side to translate between the external and local representations.

Stubs creation

- The programmer specifies a set of procedures that will be remote by giving the interface details(i.e., number and type of arguments).
- To do so, the programmer uses the tool's IDL (Interface Definition Languages).
- A IDL compiler is used to generate the client and server stubs in a high level language. (for example, C language)
- The programmer then compiles and links two separate programs. The server stubs are combined with the remote procedures to form the server program. The client stubs are combined with the main program and local procedures to form the client program.

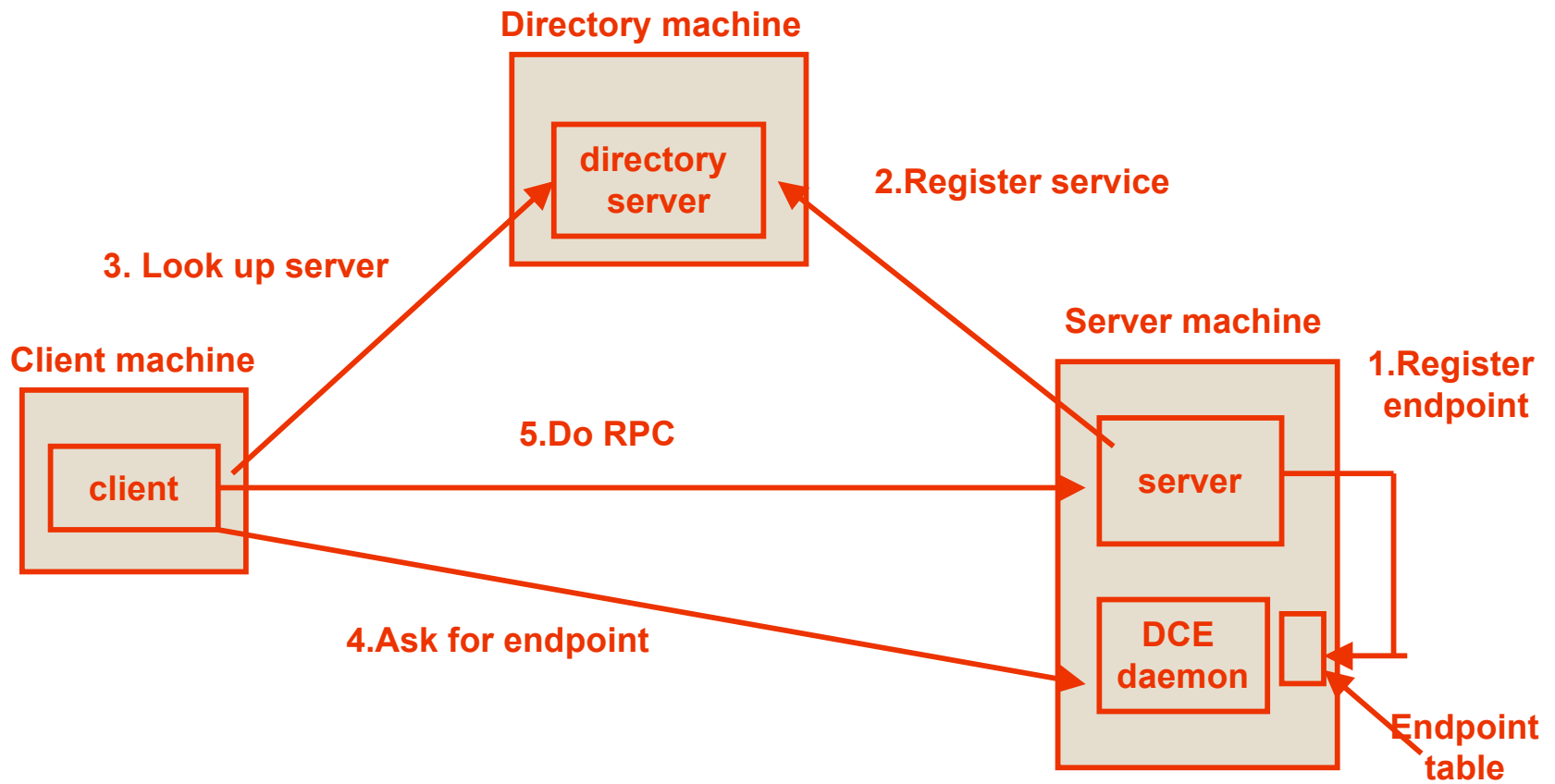
IDL (DCE)



Binding a client to a server

- To allow a client to call a server, it is necessary that the server be registered and prepared to accept incoming calls.
- Registration of a server makes it possible for a client to actually locate the server and bind to it. Server location is made in two steps:
 - Locate the server machine
 - Locate the server(i.e.,the correct process) on that machine.

-
- To communicate with a server, the client needs to know an endpoint on the server machine to which to send messages.
 - An endpoint (port) is used by the server O.S. to distinguish incoming messages for different messages.
 - A table of (server-endpoint)-pairs is maintained on each server machine by a process called **the DCE-daemon**.
 - Before it becomes available for incoming requests, the server must ask the O.S. for an endpoint. It then registers this endpoint with the DCE daemon.
 - The server also registers with the directory service by providing it the network address of the server machine and a name under which the server can be looked up

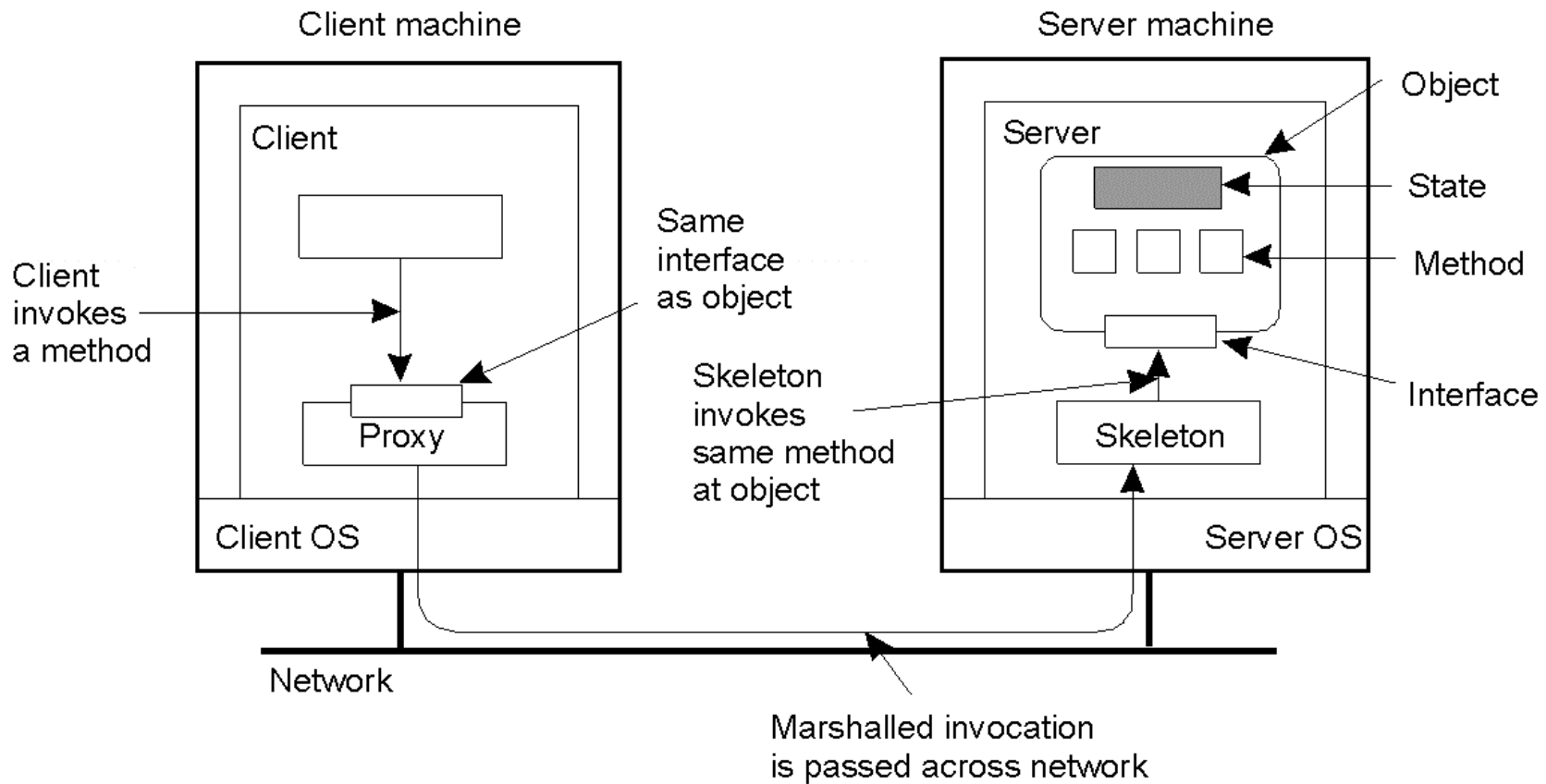


Semantic options

- DCE provides several semantic options. The default is **at-most-once operation**. No call is ever carried out more than once, even in face of system crashes.
- In practice, if a server crashes during an RPC and then recovers quickly, the client does not repeat the operation, for fear that it might already have been carried out once.
- Alternatively, it is possible to mark a remote procedure as **idempotent** (in the IDL file), in which case it can be repeated multiple times without harm. For example, reading a specified block from a file can be tried over and over until it succeeds.

RMI

- **RMI = Remote Method Invocation**
- The distributed object systems use RMI that may be considered as an object oriented extension of the RPC tool.
- A fundamental characteristic of the OO model is the separation among the method interface and its implementation.
- The execution of the methods causes object state modifications following the interface properties.
- In a distributed system the object interface belong to one machine and its implementation on the other.

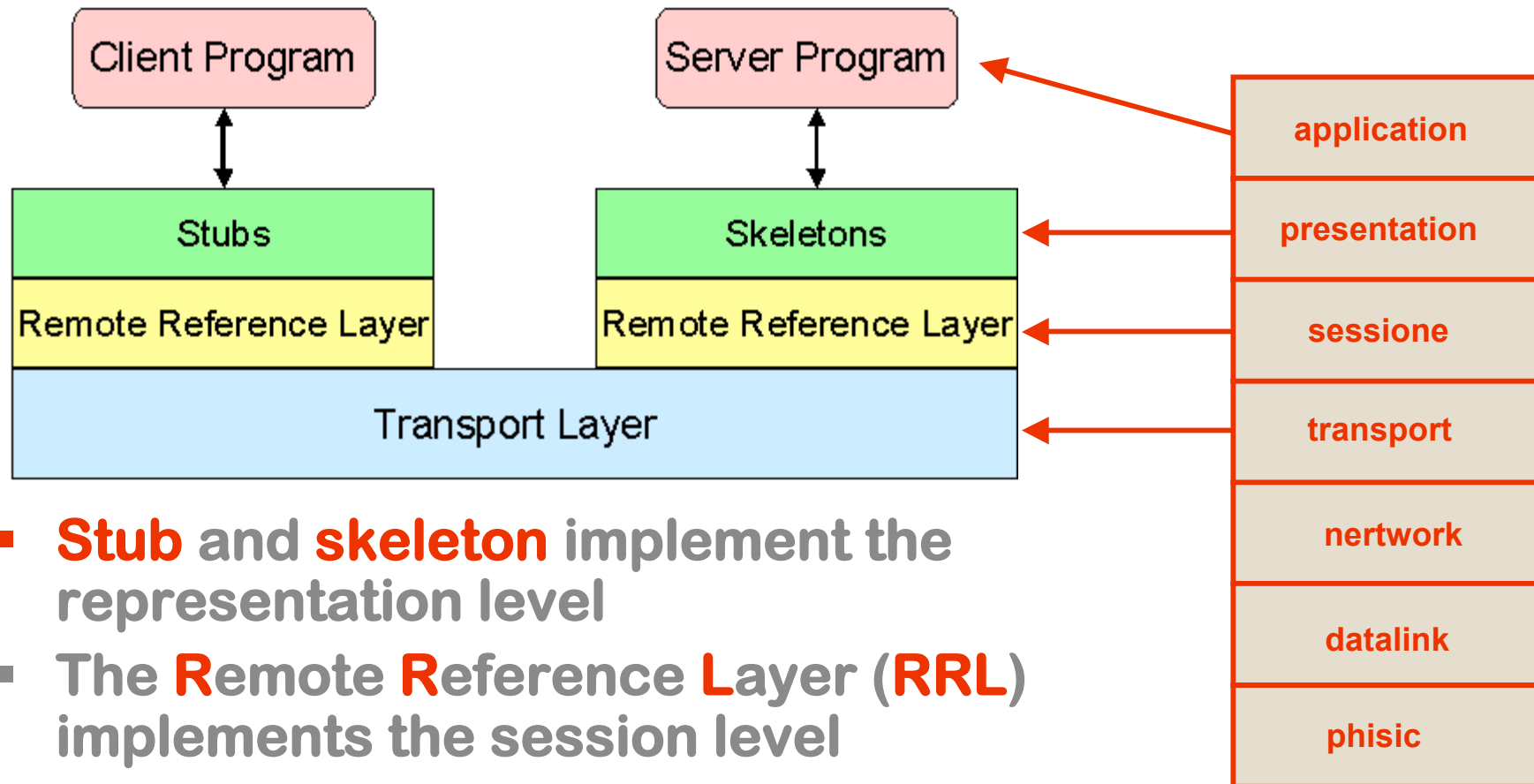


- **Proxy (o stub)** = client stub
- **Skeleton** = server stub

RMI

- On the client is create only a local proxy of the object that it is active on a server.
- The client calls the object methods by using the proxy.
- The widely-used object oriented technologies are:
 - **CORBA** (**C**ommon **O**bject **R**equest **B**roker **A**rchitecture): standard aperto, multilinguaggio
 - **DCOM** (**D**istributed **C**omponent **O**bject **M**odel)
 - **Java RMI**:

Java RMI and OSI



- **Stub** and **skeleton** implement the representation level
- The **Remote Reference Layer (RRL)** implements the session level
- **The application protocol**, based on TCP/IP, is named **IIOP** (Internet Inter Orb Protocol) and derive from CORBA

RMI and IDL

- Interface language description and automatic creation of stub e skeleton are utilized in RMI
- IDL have been extended with object oriented concepts (interfaces, subtyping, polimorphirsm ecc.)
- IDL example: an object that provides date and hour

```
module DateTimeApp
{
    interface DateTime
    {
        string getDate();
        string getTime();
        long getMsecs();
    };
};
```