

ICS 362 Distributed Systems

Distributed Systems: Part 6

Lecturer: Toby Daniel

Communication

- Obviously for distributed systems to work, communication between devices, processors etc. is necessary.
 - we will look at:
 - Layered Communication Protocols
 - Remote Procedure Calls (RPC)
 - Remote Object Invocation (RMI – method)
 - Message Oriented Communication (MOM)
 - Streams

Layered Communication Protocols

- A protocol is an agreed set of rules
- Sender A, has to agree with Receiver B as to the nature of the communication, what format the encoding will take place in, etc.
- To simplify the way this communication can be presented, layered protocols break the problem into small sub-tasks.
- Communication protocol design principles include **effectiveness**, **reliability**, and **resiliency**.

Effectiveness

- A communications protocol needs to be specified in such a way, that engineers, designers, and software developers can implement and use it.
- Protocol layering accomplishes these objectives by dividing communication into a number of smaller parts, each of which performs closely related sub-tasks, and interacts in well-defined ways.





Reliability

ON OFF MAYBE

Handwritten signature

Reliability

- Assuring reliability of data transmission involves error detection and correction, or some means of requesting retransmission.
- The conventional measure of quality is the number of failed bits per bits transmitted.
 - This has the useful feature of being a dimensionless figure that can be compared across any speed or type of communication media.
- In telephony, links with bit error rates (BER) of 10^{-4} or more are regarded as faulty
- Data transmission often requires bit error rates below 10^{-12} .

Resiliency



- Resilience or strength against failure.
- What happens if a communication line fails?
 - Is the protocol strong enough to recover?
 - Will there be a total communication failure?
- Most modern communication protocols periodically send messages to test a link.
 - In phones, a framing bit is sent every 24 bits on T1 lines. In phone systems, when "sync is lost", fail-safe mechanisms reroute the signals around the failing equipment.

A number of major protocol stacks or families exist:

Open standards:

- Internet protocol suite (TCP/IP)
- Open Systems Interconnection (OSI)
- FTP
- UPnP (Universal Plug and Play)
- iSCSI
- NFS

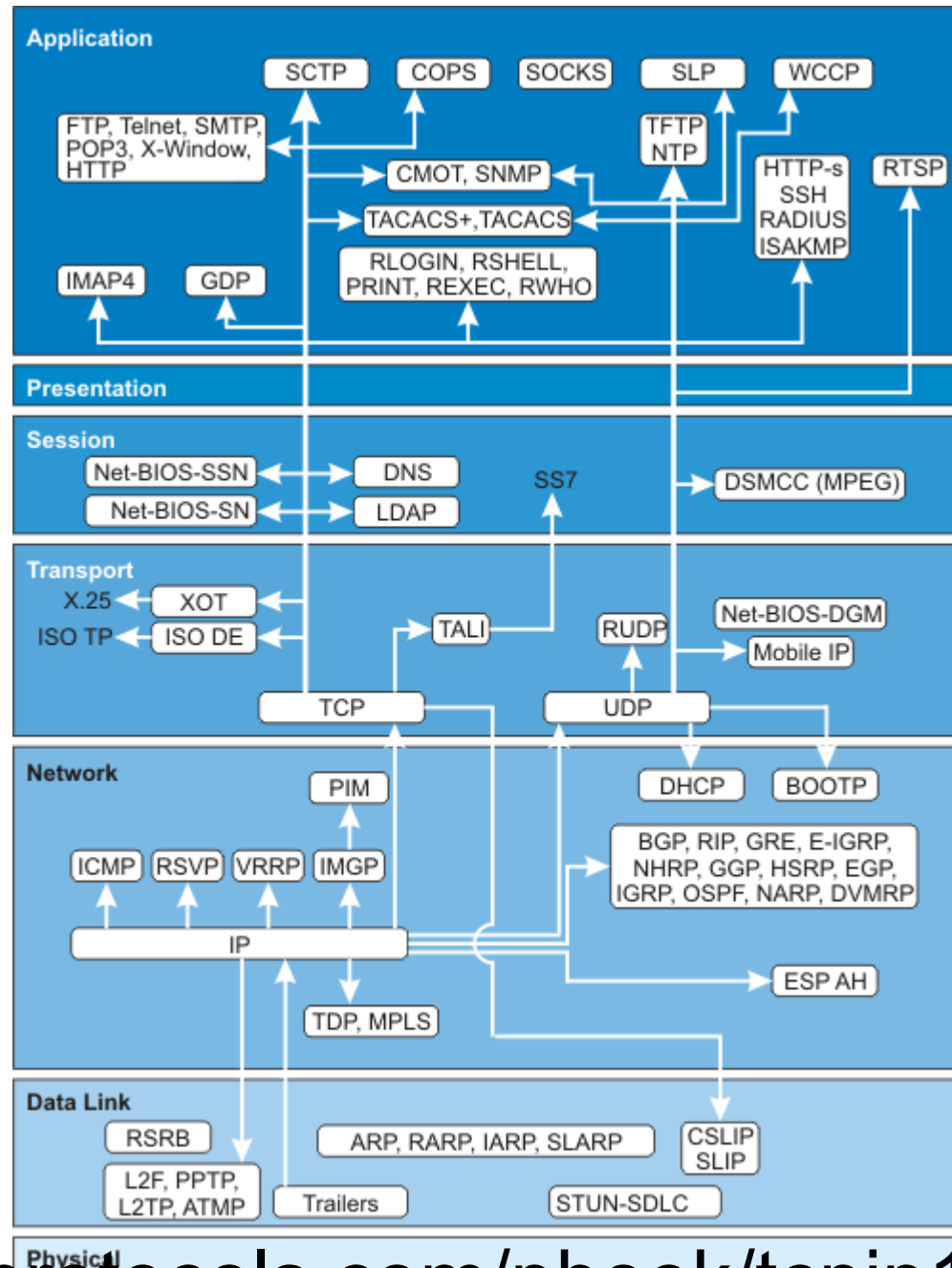
Proprietary standards:

- AppleTalk
- IPX/SPX
- Server Message Block (SMB)
- Systems Network Architecture (SNA)

OSI layered protocol model



Map of layered protocols



<http://www.protocols.com/pbook/tcpip1.htm#MAP>

Lower Layers

- Physical Layer
 - Responsible for sending bits, ensuring that when a 0 is sent, a 0 is received.
- Data Link Layer
 - Checksum added to remove errors, so the receiver can confirm they have received the correct sequence.
- Network Layer
 - Responsible for finding a route between sender and receiver, for instance IP.

Middle Layer (Transport)

- Layer responsible for ensuring a message is received.
 - The transport layer breaks a message into packets, and ensures the packets are received, so the application layer can simply pass a message to the transport layer and assume that it has arrived at the receiver.
- TCP is a de facto standard, while UDP is a limited version.

Upper Layers

- The upper layers are often grouped into the application layer, where any applications using the other communication layers exist, for instance email, file transfer etc.
- Middleware, therefore sits in the upper layer (application layer), so we will investigate further middleware communication services in the remainder of this section.

Connection vs Connectionless



Connectionless Communication

- Connectionless describes communication between two network end points in which a message can be sent from one end point to another without prior arrangement.
 - The device at one end of the communication transmits data to the other, without first ensuring that the recipient is available and ready to receive the data.
 - The device sending a message simply sends it addressed to the intended recipient.
 - Connectionless protocols are usually described as *stateless* because the endpoints have no way to remember where they are in a "conversation" of message exchanges.
- Examples: Internet Protocol (IP) and User Datagram Protocol (UDP)

Connection Orientated Comm.s

- A connection-oriented networking protocol is one which identifies traffic flows by some connection identifier rather than just listing the source and destination addresses.
 - Note that connection-oriented protocols are not necessarily reliable protocols. ATM and Frame Relay, for example, are both examples of a connection-oriented, unreliable protocol.
 - Connection-oriented protocols handle real-time traffic substantially more efficiently than connectionless protocols.
 - Described as stateful because they can keep track of a conversation
- Example: A Phone Call, TCP, ATM

Local Function Call

What is a local function call?



Local Function Call

- An expression that moves the path of execution from the current function to a specified function and evaluates to the return value provided by the called function. A function call contains the name of the function to which control moves and a parenthesized list of values.

For example:

```
stub()  
overdue(account, date, amount)  
notify(name, date + 5)  
report(error, time, date, ++num)
```

This happens “locally” within the one computer

Remote Procedure Call

- Remote Procedure Call, or RPC, refers to a middleware technique where a client can simply call a function located on a server. (not local)
- Ideally this works in the same way as a local function call, parameters are sent and results returned.
- The user need have no knowledge that the function hasn't actually been performed locally. But how does this work exactly?

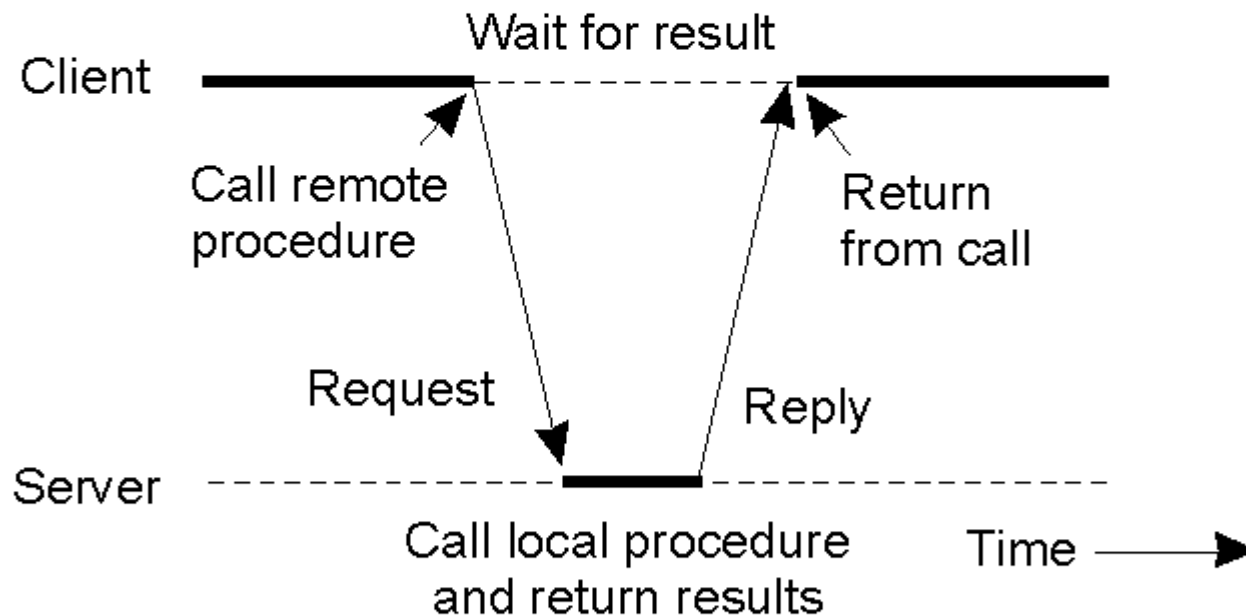
RPC

- Consider this function call;
word = read(start, length);
- When running locally, the two variables are sent to the read function, and the result returned to 'word'.
- When running using RPC, the parameters are sent to the server, and the server returns data to be stored in word.

Client and server stubs

Principle of RPC between a client and server program.

A “stub” provides a communication point, it is a way of packaging and sending the procedure call.

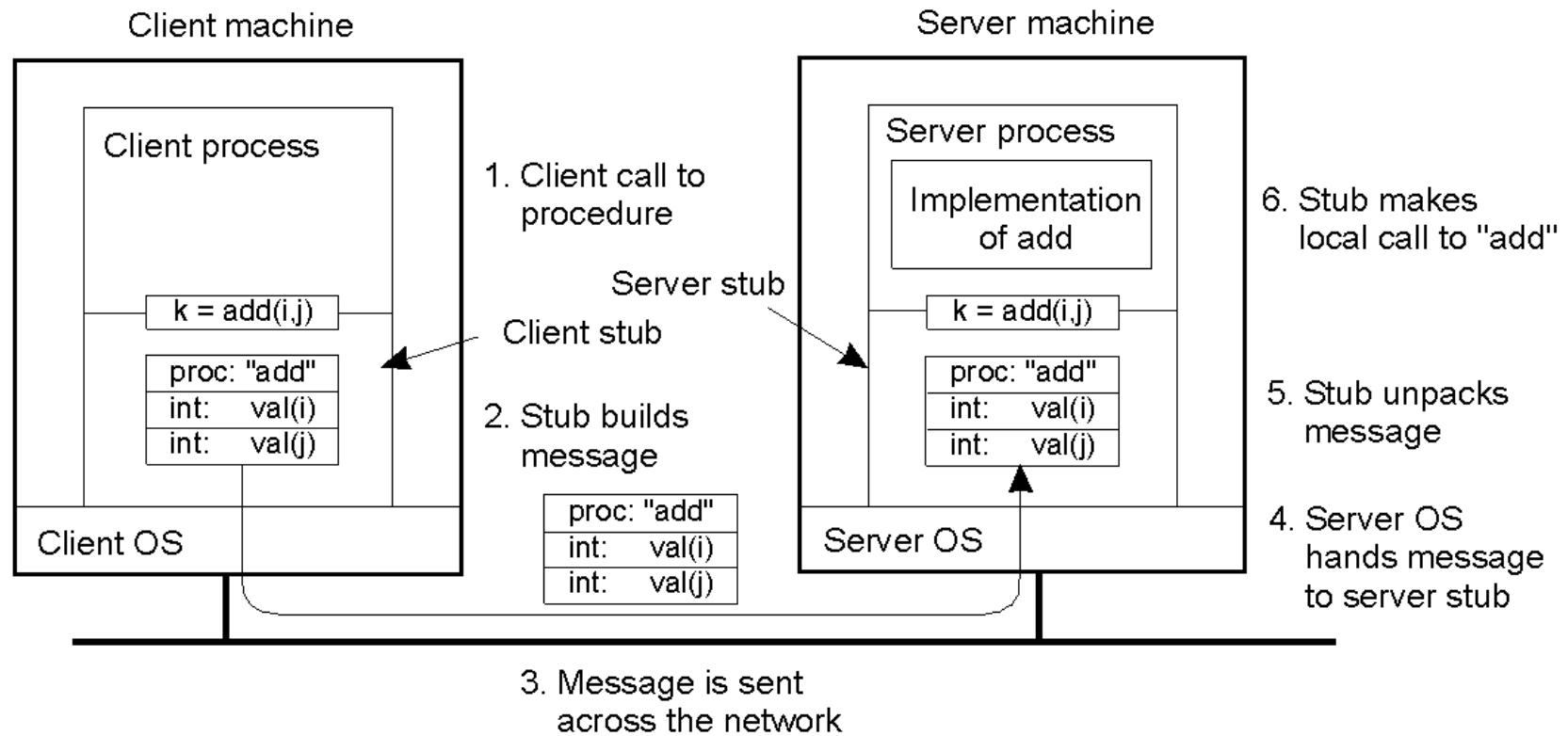


Steps of a Remote Procedure Call

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

Passing Value Parameters

Steps involved in doing remote computation through RPC

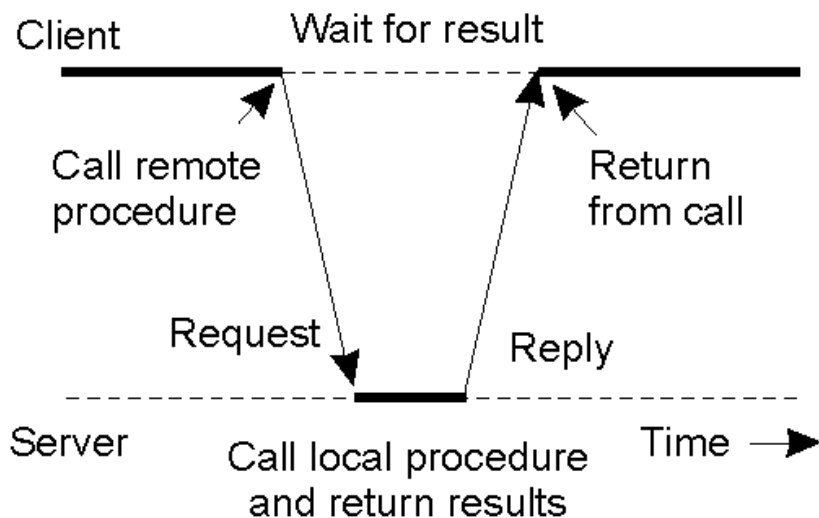


Passing Value Parameters

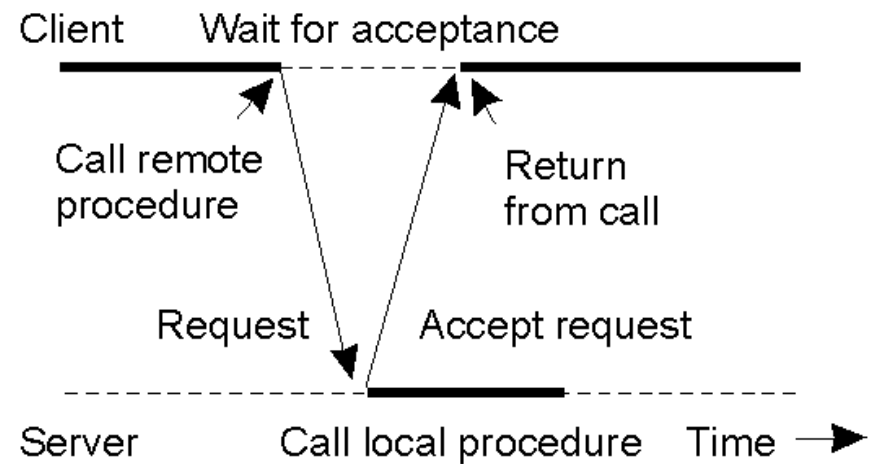
- Obviously for this method to work, the sender and receiver need to encode data in the same format.
- This is why we need communication protocols!

Asynchronous RPC

- Sometimes it isn't convenient for the client to wait inactive for the result to be returned from the server;
 - Consider if we requested the server to transfer money from our account, then we only need wait for confirmation that the server has received our call, not that it has completed processing it.



(a)

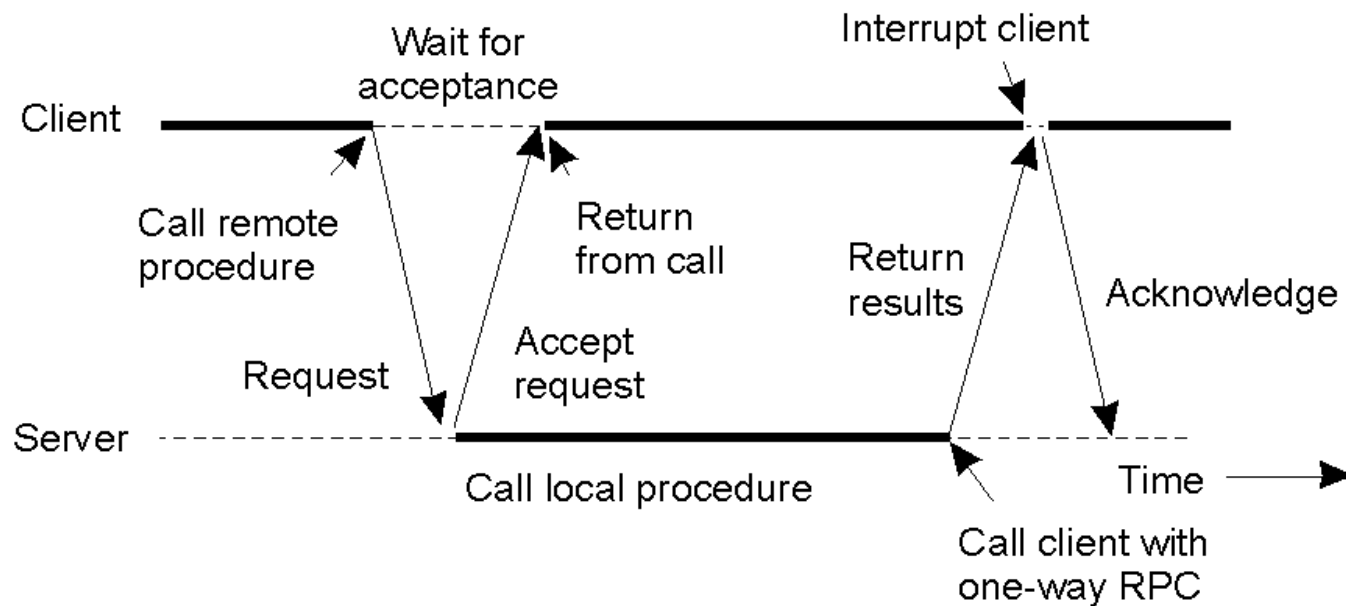


(b)

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

Deferred Synchronous RPC

- In other cases we require results when the server completes processing, but can continue in the meantime;
 - Consider if we requested the addresses of various resources that we expected to need in the future.



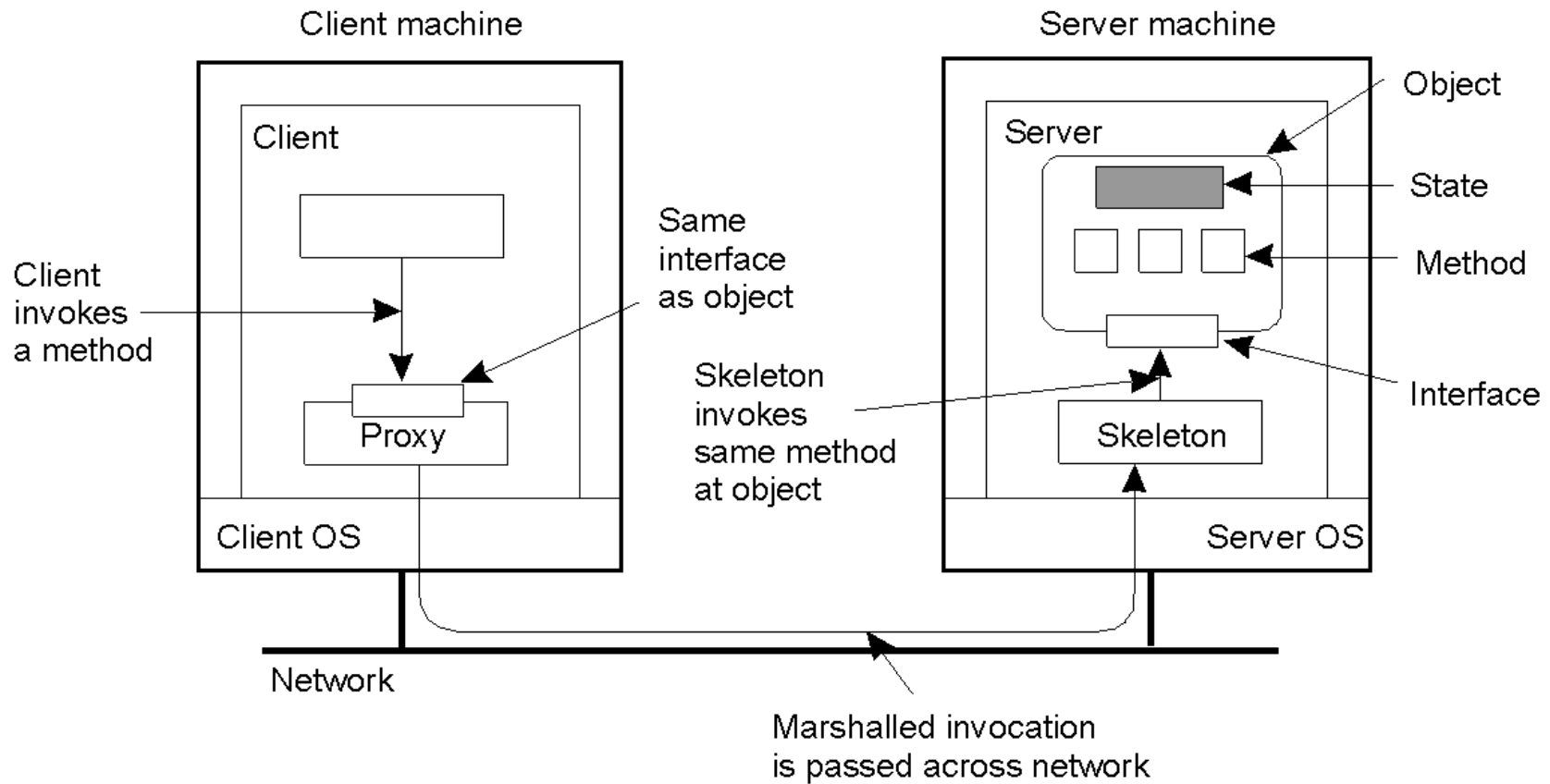
O-O Programming

- Some RPC's, such as the DCE (Distributed Computer Environment), were developed concurrently with emerging O-O practice.
- If we can call a remote procedure, why not invoke a remote object?
 - Objects encapsulate data and operations together (state and methods).
 - To manipulate the state of an object we use methods available in the objects interface.
 - As the interface is separate from the object, is convenient for distributed systems.

Remote Object Invocation

- Remote Object Invocation (ROI) is an approach allowing interfaces to be placed on different machines to the object, creating a ***distributed object***.
- Note that the object's state isn't distributed, this remains on the same machine, however, the interface is distributed to allow remote calls.

Distributed Objects



Binding

- For a client to use a server's distributed object, first it must bind to the object.
 - Binding creates a proxy in the client's address space, the proxy can then 'marshal' requests to the server.
- Binding can be implicit, perhaps simply using a pointer to the distributed object.
- Or Explicit, where initially a call to a special function 'bind()' creates the local proxy.

Remote Method Invocation

- Remote Method Invocation (RMI) is similar to RPC.
 - Methods included in the interface are defined in a Interface Definition Language (IDL).
- The IDL could be an existing O-O language, in which case invocation is static, dependent on pre-compilation.
- Alternatively different interfaces could be composed at runtime allowing dynamic invocation.

RPC and RMI

- What are the advantages and disadvantages?



RPC and RMI

- Both methods improve access transparency.
- However, both are inherently synchronous
- Even asynchronous RPC's 'wait' for confirmation.
- If the server was unavailable, or network unreliable, the system is ineffective.
- Messaging Oriented Middleware (MOM), a.k.a. Message Oriented Communication (MOC), is an alternative allowing asynchronous communication.

Persistence and Synchronicity

