DISTRIBUTED SYSTEMS

Communication

Introduction

- In a distributed system, processes run on different machines.
- Processes can only exchange information through message passing.
  - harder to program than shared memory communication
- Successful distributed systems depend on communication models that hide or simplify message passing

Introduction

- IPC in distributed systems is always based on low-level message passing
- Remote procedure Call (RPC)

  Hiding most of the intricacies of message passing

  client-server applications
- Remote Method Invocation
- Message-Oriented Middleware (MOM)

  Not strict pattern of client-server interaction High-level message queuing
- Data Steaming for Multimedia

  Continuous flow subject timing constraints

Layered Protocols (1)

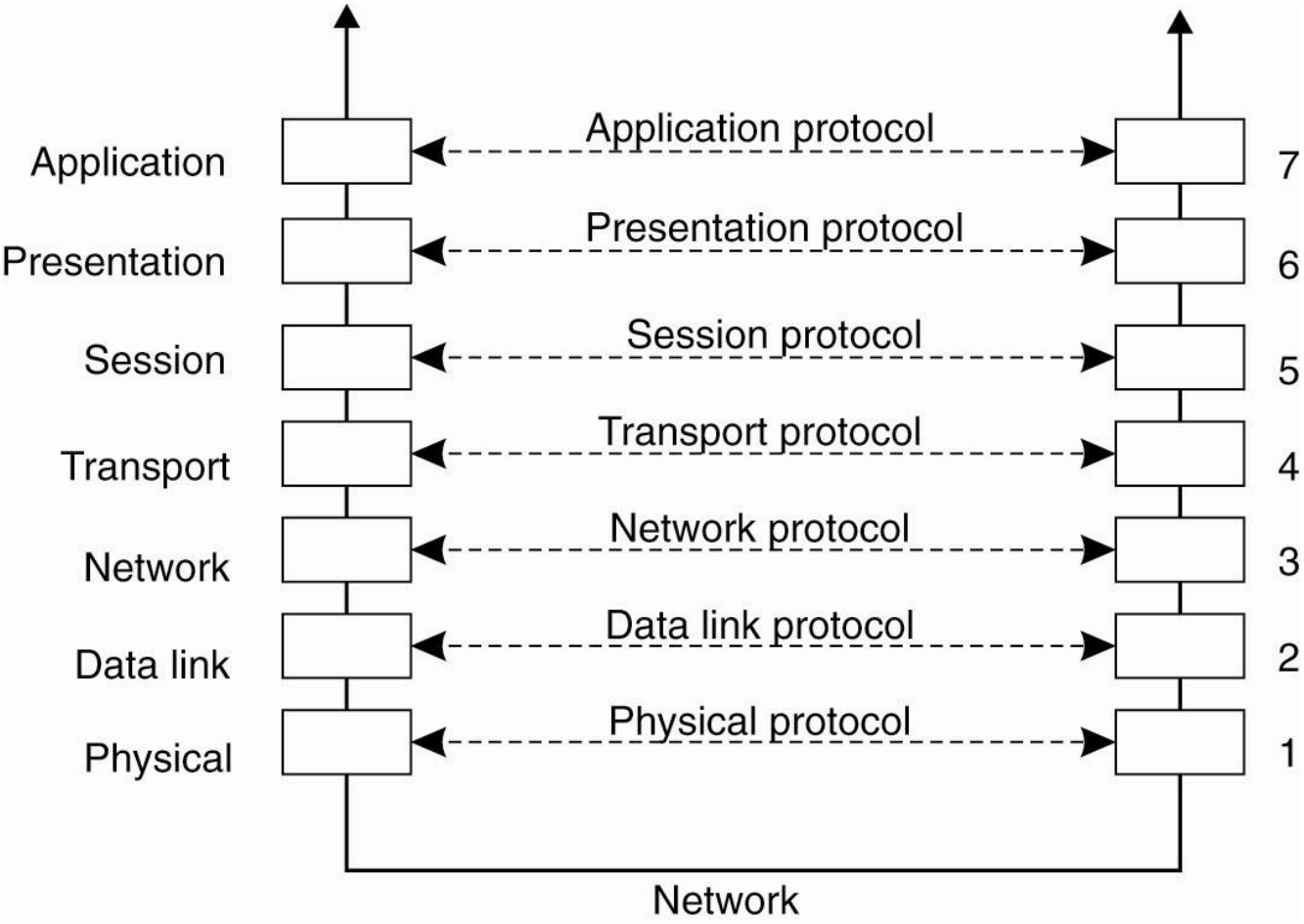


Figure. Layers, interfaces, and protocols in the OSI model.

Layered Protocols (2)

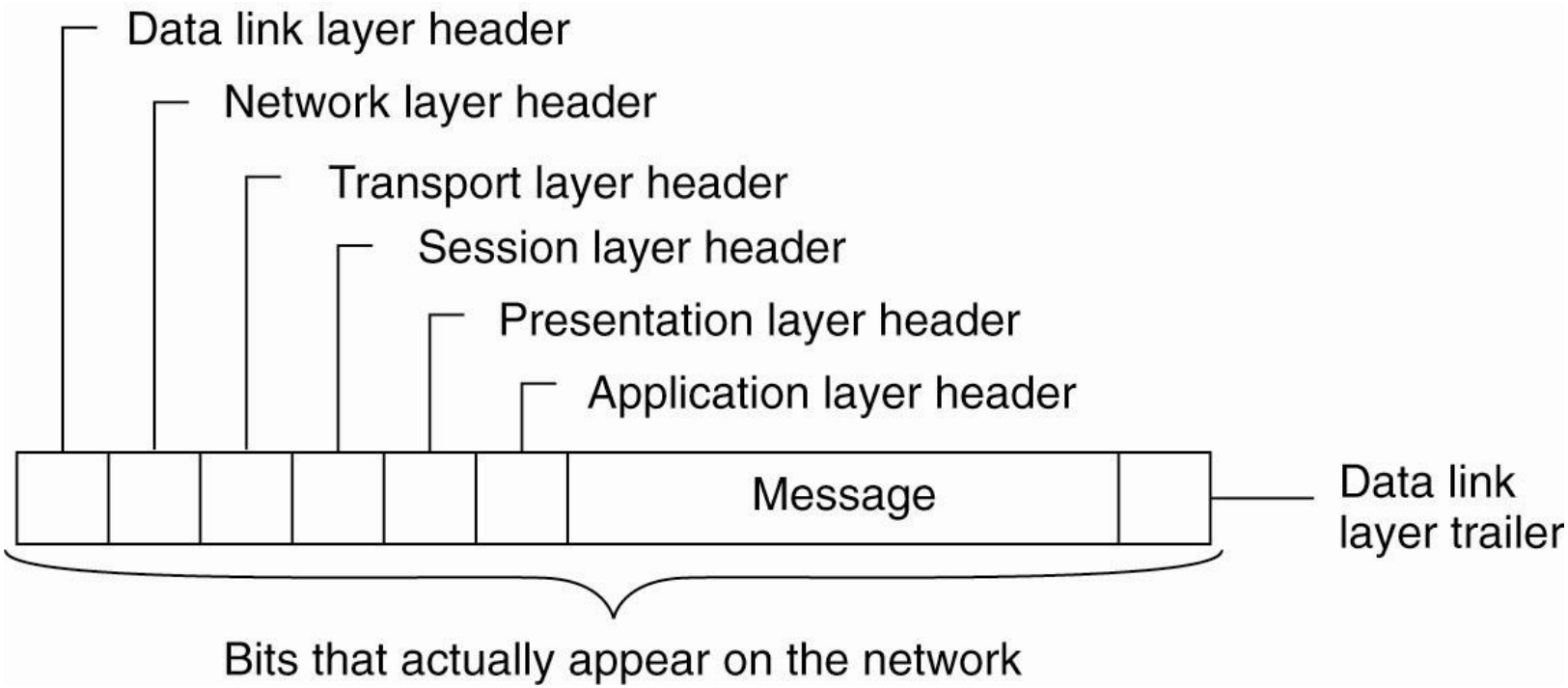


Figure. A typical message as it appears on the network.

Middleware Protocols

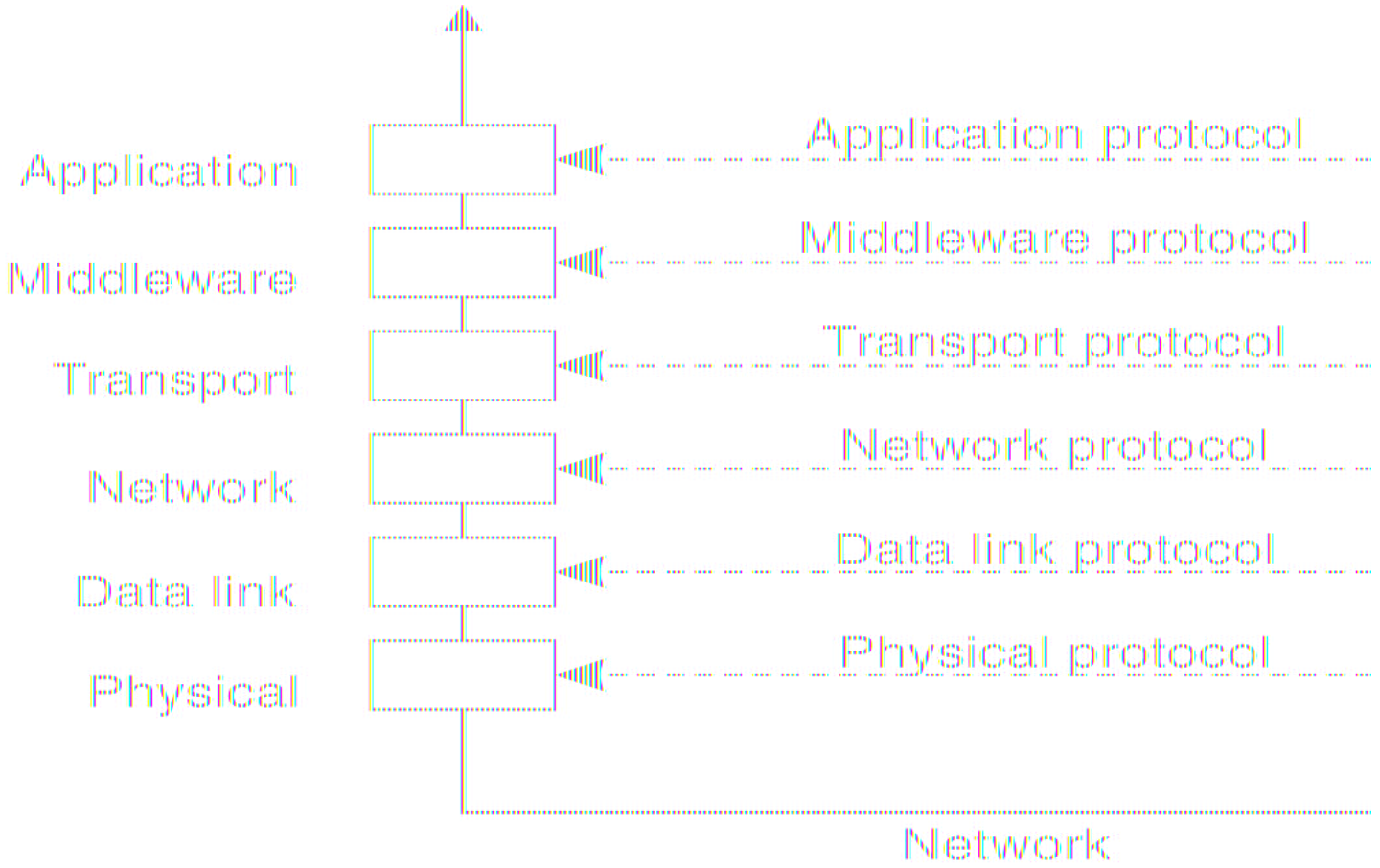
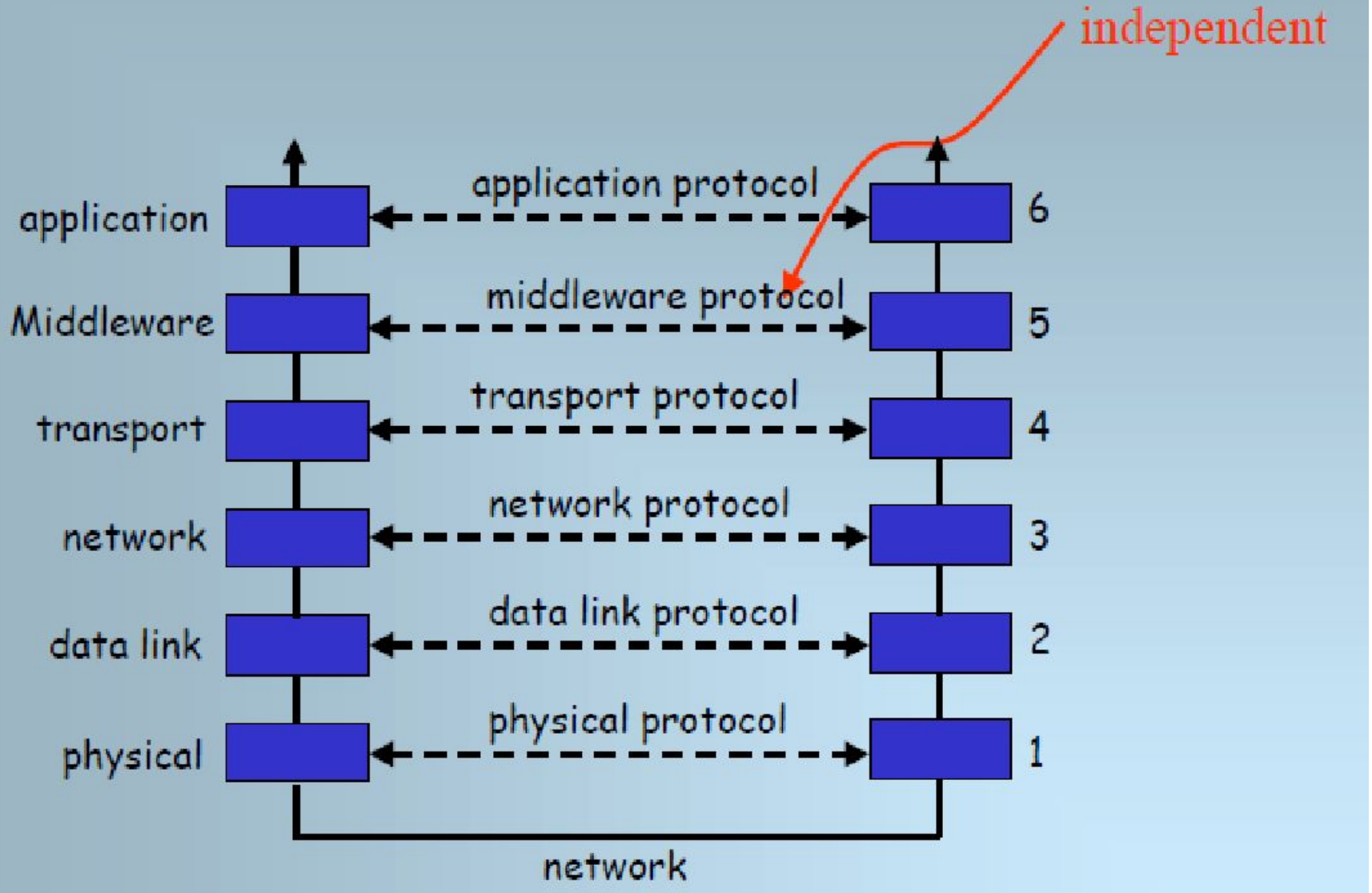


Figure. An adapted reference model for networked communication.

Middleware Protocols



An adapted reference model for networked communication.

Remote Procedure Call (RPC)

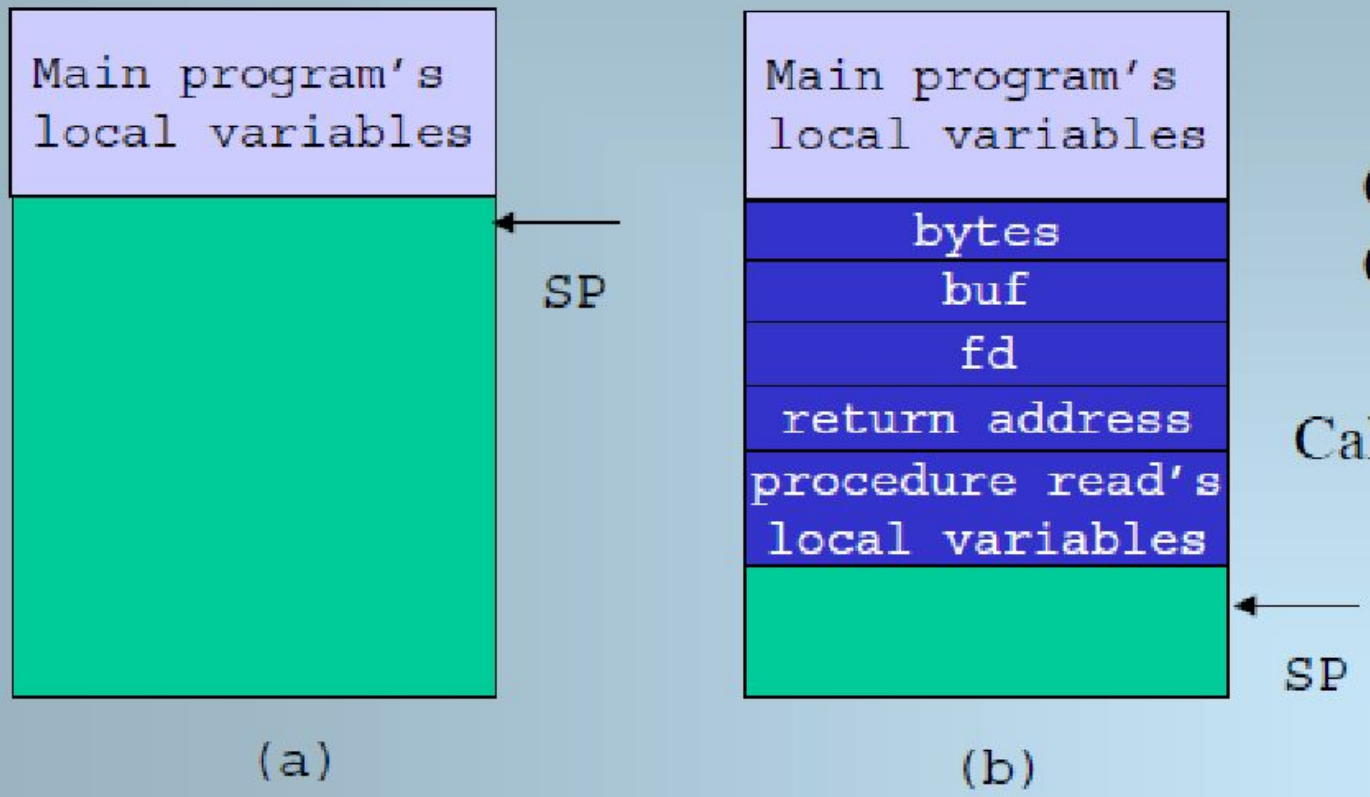
The procedures such as `send` and `receive` do not conceal the communication

More sophisticated is allowing programs to call procedures located on other machines.

- Basic RPC operation
- Parameter passing
- Variations

Conventional Procedure Call

```
int  read (int fd, char *buf, int bytes);
```



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

b) The stack while the called procedure is active

Conventional Procedure Call

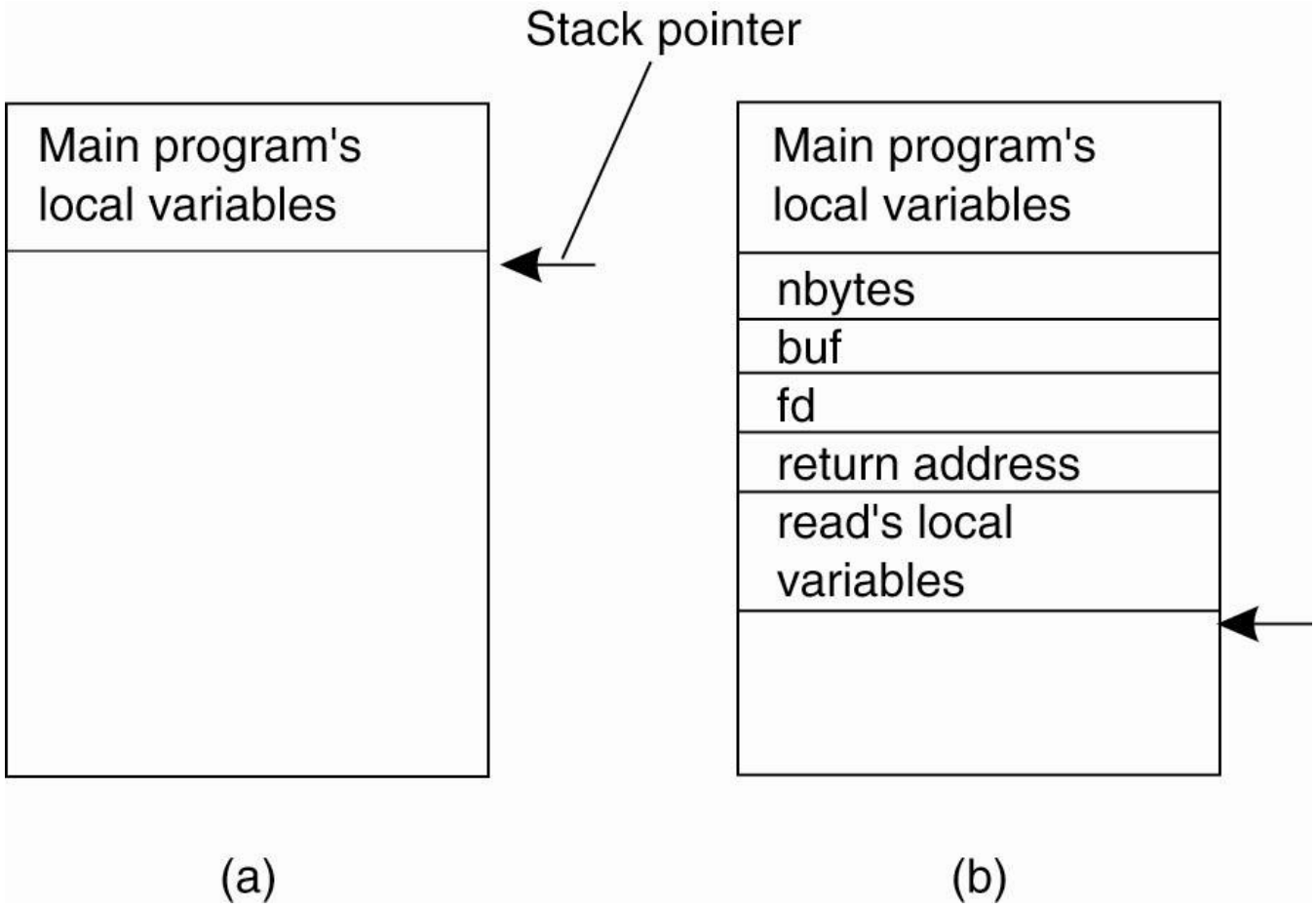


Figure : Parameter passing in a local procedure call: the stack before the call to read. (b) The stack while the called procedure is active.

Why RPC

Calling a function at a remote server:

Advanced form of communication

- Sockets and MPI: data transfer
- RPC: control transfer

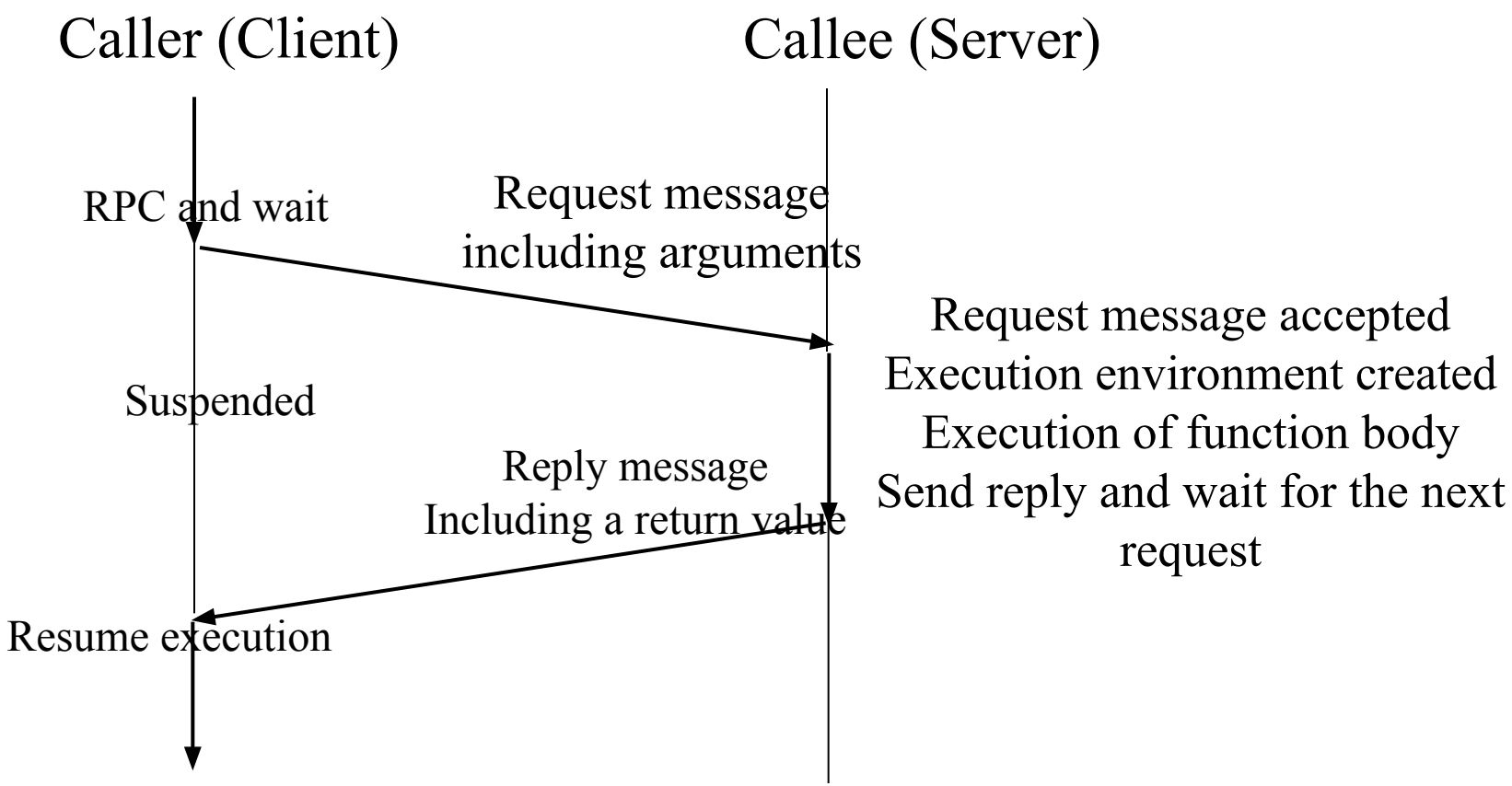
Transparency in function calls

- No distinguish between local and remote calls
- Also applied to IPC on the same machin

Ease of use

- Compiled-time argument type checking
- Automatic interface generation

RPC Model


Caller (Client)
Callee (Server)
RPC and wait
Request message
including arguments
Request message accepted
Execution environment created
Execution of function body
Send reply and wait for the next
request
Suspended
Reply message
Including a return value
Resume execution

*

Remote Procedure Calls (1)

A remote procedure call occurs in the following steps:

1. The client procedure calls the client stub in the normal way.
2. The client stub builds a message and calls the local operating system.
3. The client's OS sends the message to the remote OS.
4. The remote OS gives the message to the server stub.
5. The server stub unpacks the parameters and calls the server.

Continued …

Remote Procedure Calls (2)

A remote procedure call occurs in the following steps (continued):

6. The server does the work and returns the result to the stub.
7. The server stub packs it in a message and calls its local OS.
8. The server's OS sends the message to the client's OS.
9. The client's OS gives the message to the client stub.
10. The stub unpacks the result and returns to the client.

Passing Value Parameters (1)

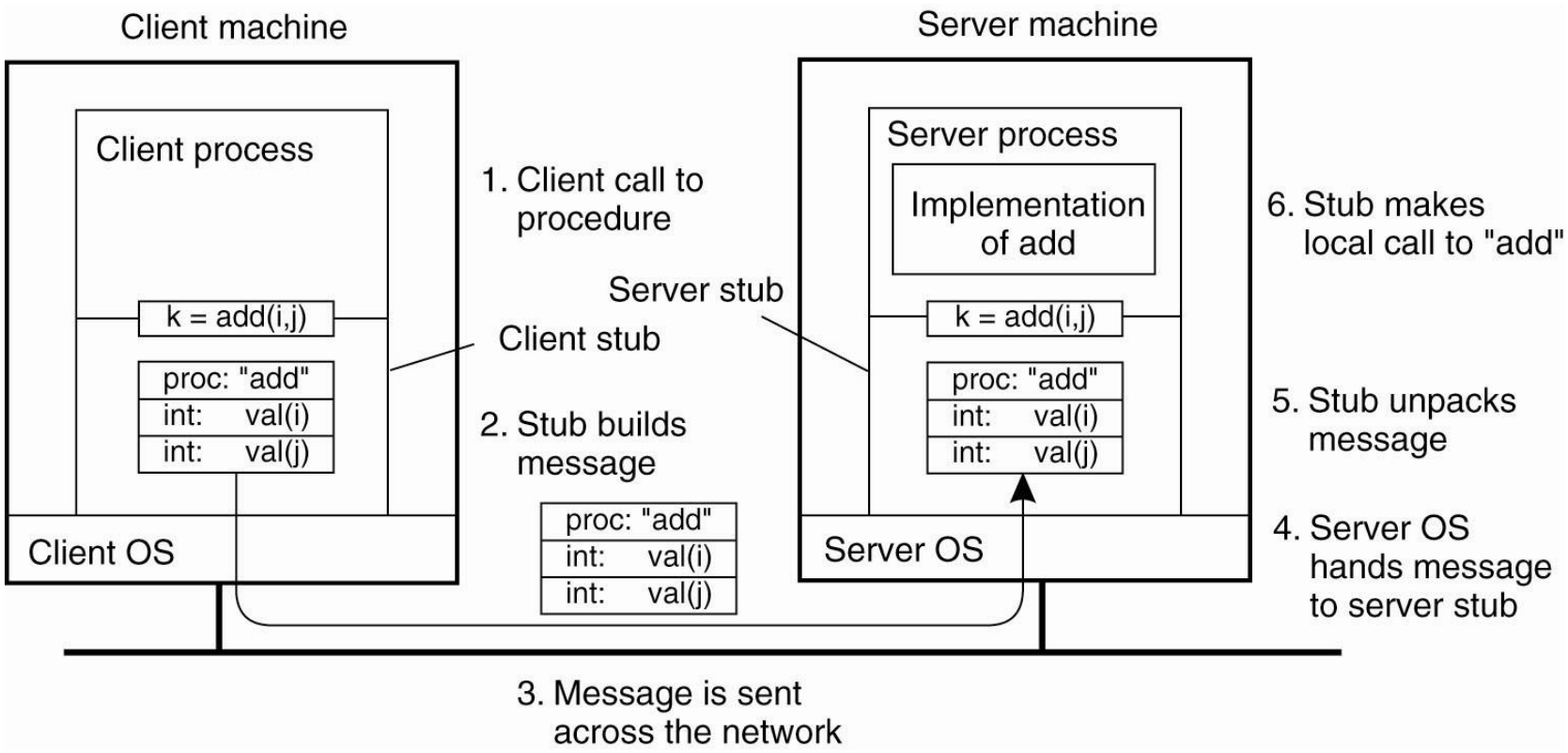


Figure . The steps involved in a doing a remote computation through RPC.

Implementation Issues - RPC

- Parameter Passing
  - Call by Value – no major issues
  - Call by reference – different address spaces
- Transparency property
  - Syntactic transparency
  - Semantic transparency
    - Analogy in semantics b/w local and remote procedure calls
      - Caller capable of passing arguments
      - Caller suspended till a return from a function
      - Callee capable of returning a value to caller
    - Difference in semantics b/w local and remote procedure calls
      - No call by reference and no pointer-involved arguments
      - Error handling required for communication
      - Performance much slower than local calls.

Marshalling Arguments

- *Marshalling* is the packing of function parameters into a message packet

– the RPC stubs call type-specific functions to marshal or unmarshal the parameters of an RPC
  - Client stub marshals the arguments into a message
  - Server stub unmarshals the arguments and uses them to invoke the service function

– on return:
  - the server stub marshals return values
  - the client stub unmarshals return values, and returns to the client program

*

Passing Value Parameters (2)

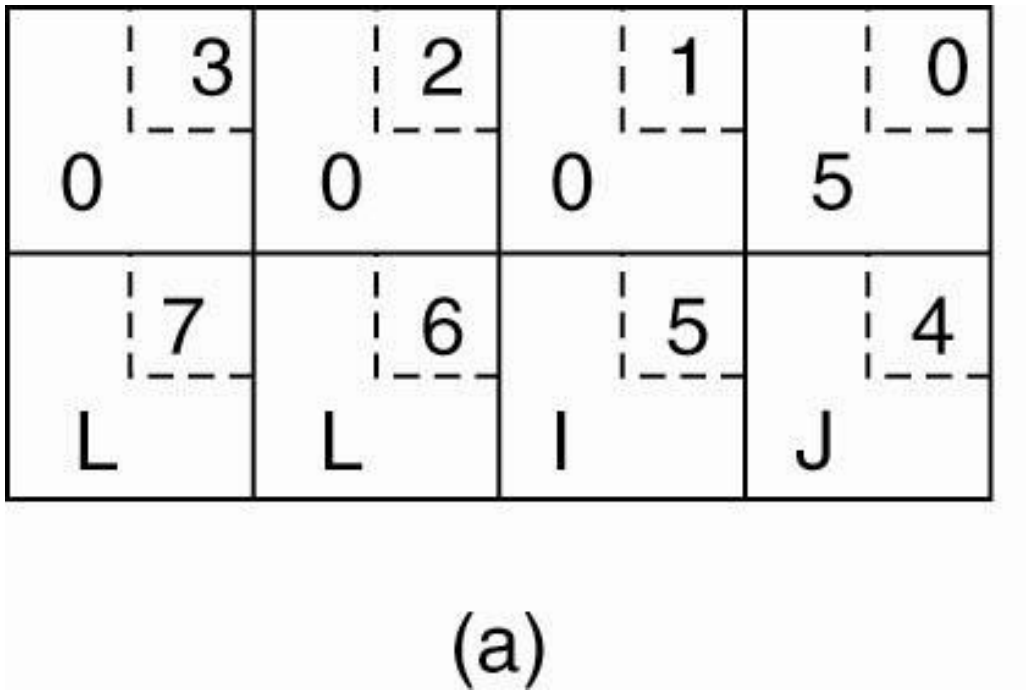


Figure. (a) The original message on the Pentium.

Passing Value Parameters (3)

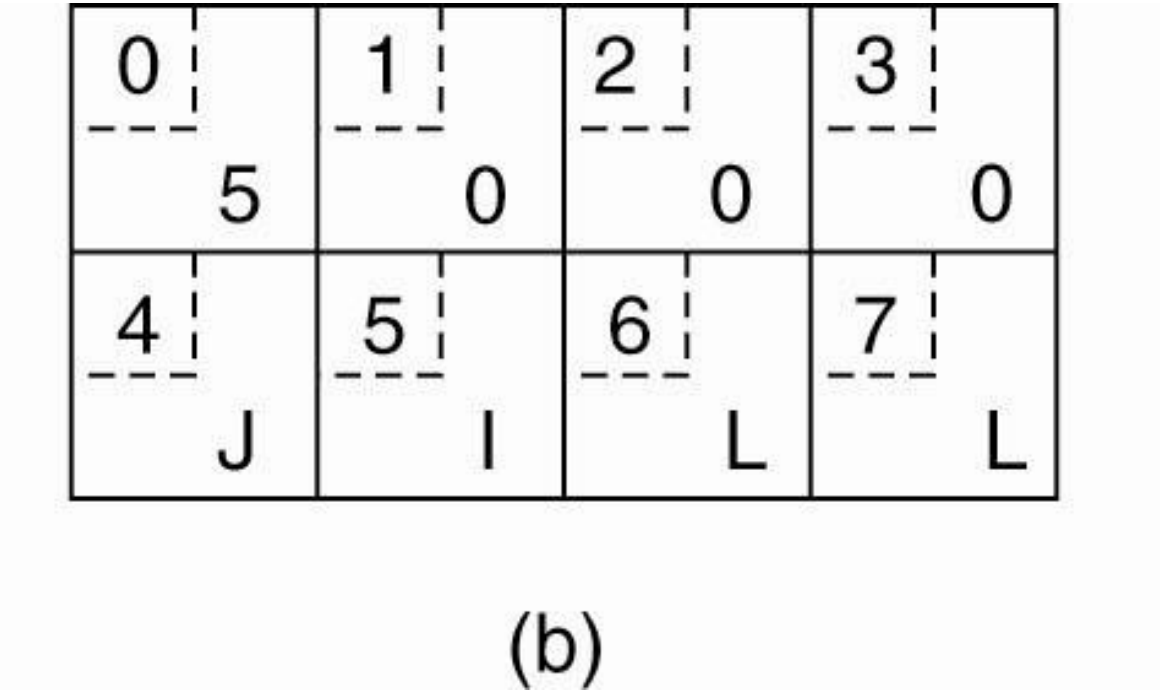


Figure. (b) The message after receipt on the SPARC

Passing Value Parameters (4)

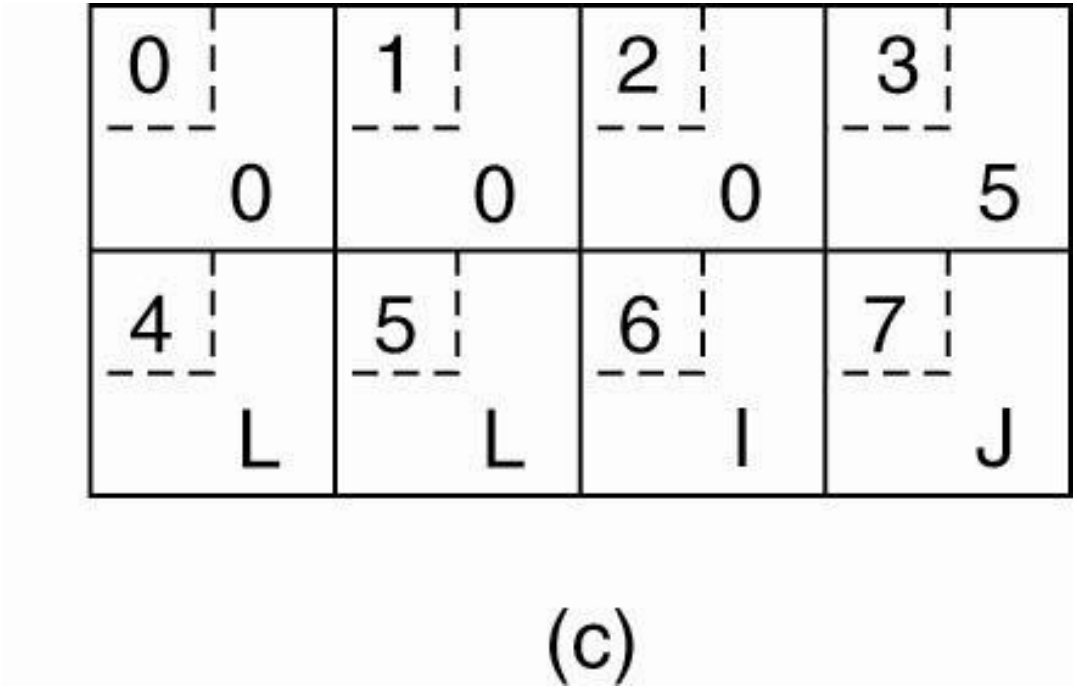


Figure (c) The message after being inverted. The little numbers in boxes indicate the address of each byte.

Issue #2 — Pointers and References

`read(int fd, char* buf, int nbytes)`

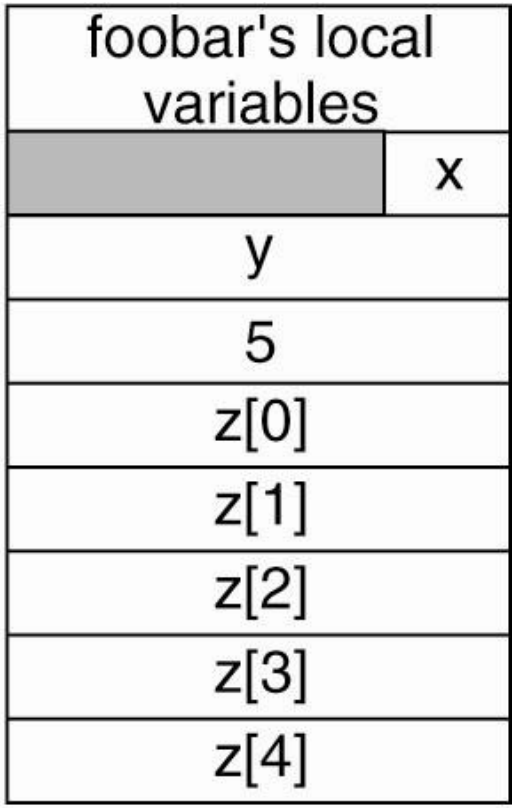
- Pointers are only valid within one address space
  - Cannot be interpreted by another process
  - Even on same machine!

- Pointers and references are ubiquitous in C, C++
  - Even in Java implementations!

Parameter Specification and Stub Generation

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

(a)



(b)

Figure. (a) A procedure. (b) The corresponding message.

Writing a Client and a Server

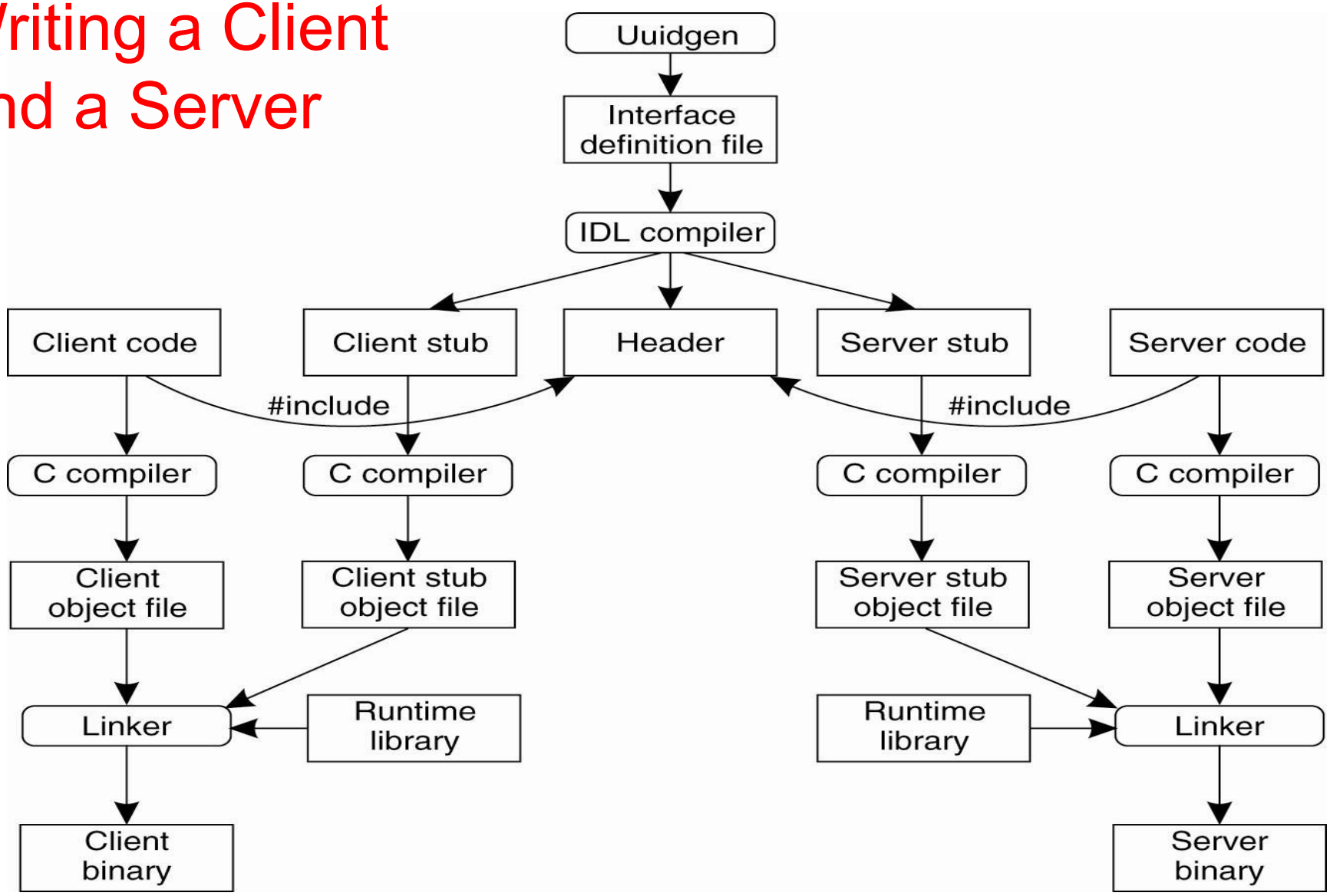


Figure. The steps in writing a client and a server in DCE RPC.

DCE RPC

- Distributed Computing Environment (DCE) developed by Open Software Foundation (OSF)
- Uuidgen generate a prototype IDL file
- Editing the IDL file filling remote procedures and parameters
- IDL compiler then generate
  Header file
  Client stub
  Server stub

Writing a Client and a Server (2)

Three files output by the IDL compiler:

- A header file (e.g., interface.h, in C terms).
- The client stub.
- The server stub.

Binding a Client to a Server (1)

- Registration of a server makes it possible for a client to locate the server and bind to it.

- Server location is done in two steps:
  1. Locate the server's machine.
  2. Locate the server on that machine.

Binding Client to Server (DCE RPC)

- Server asks operating system for an end point (port)
- Register the endpoint with DCE daemon that records in the end table
- Server register with directory server with its network address and its name (program name)
- Now Client pass the name to directory server and find the server network address
- Client goes to DCE daemon (well know port) on that machine and ask for look up the end point (port) of the server in its end point table
- RPC then takes place

Binding a Client to a Server (2)

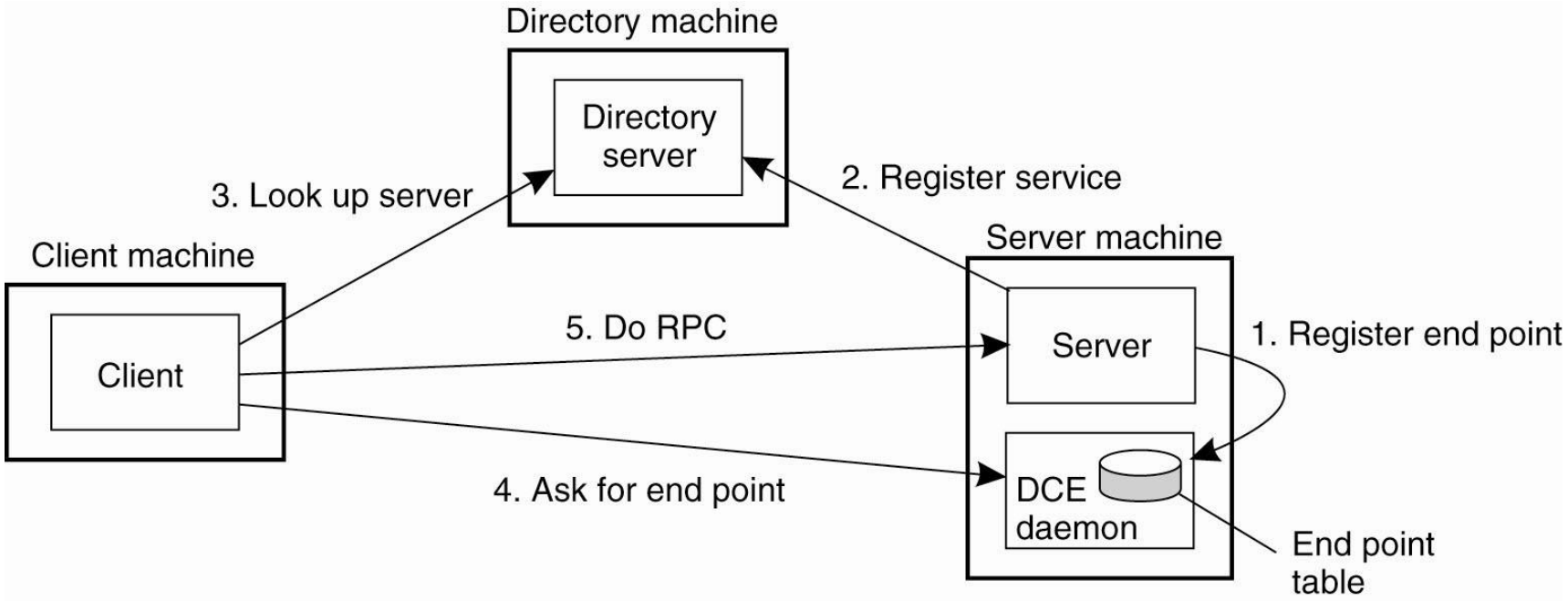


Figure. Client-to-server binding in DCE.

Questions

- Explain RPC model
- What are the issues in Parameter passing of RPC
- Describe a sample implementation of RPC/ Describe the working of DCE RPC
- Steps in binding