Communication in Distributed Systems

REK's adaptation of

Tanenbaum's

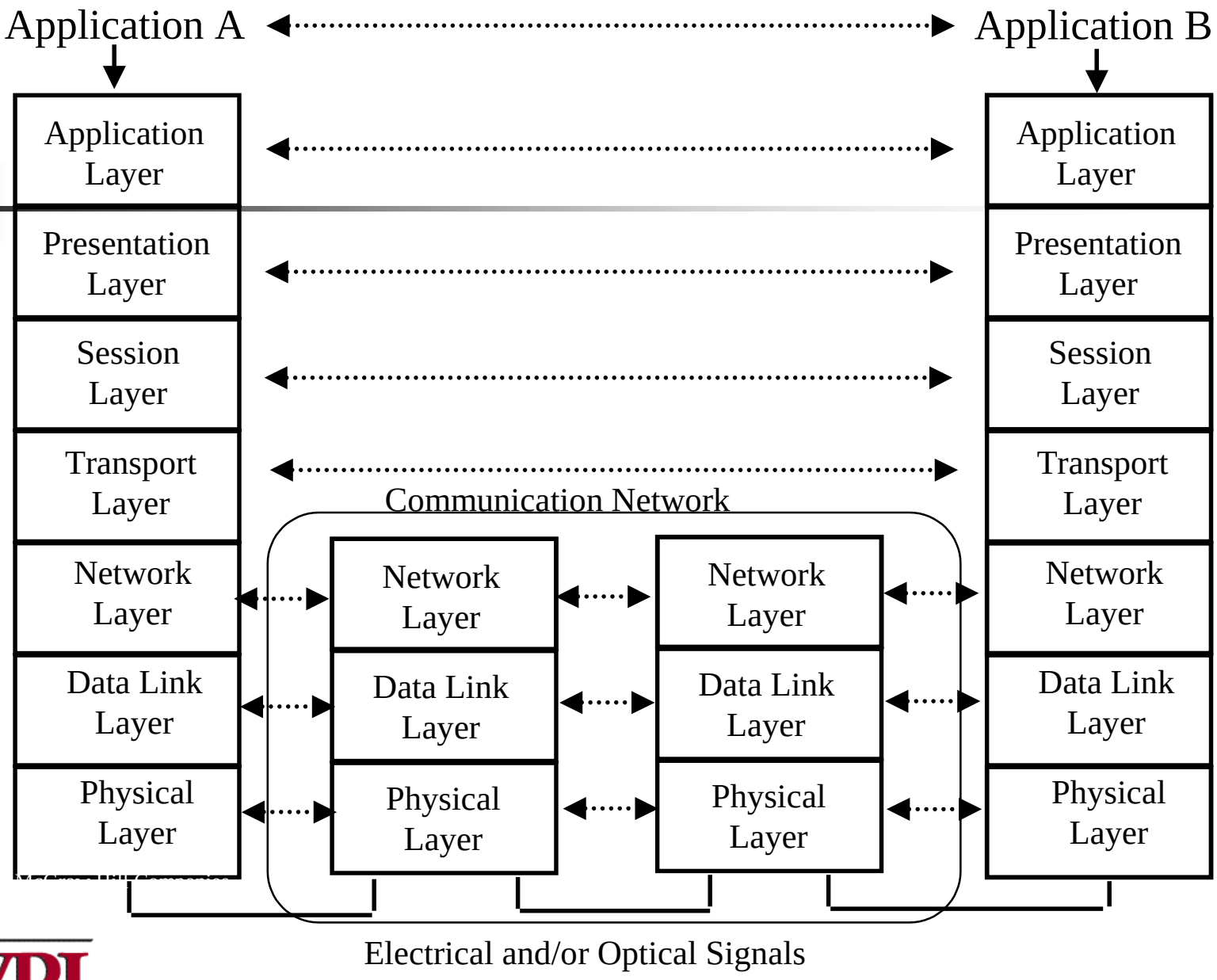
Distributed Systems

Chapter 2

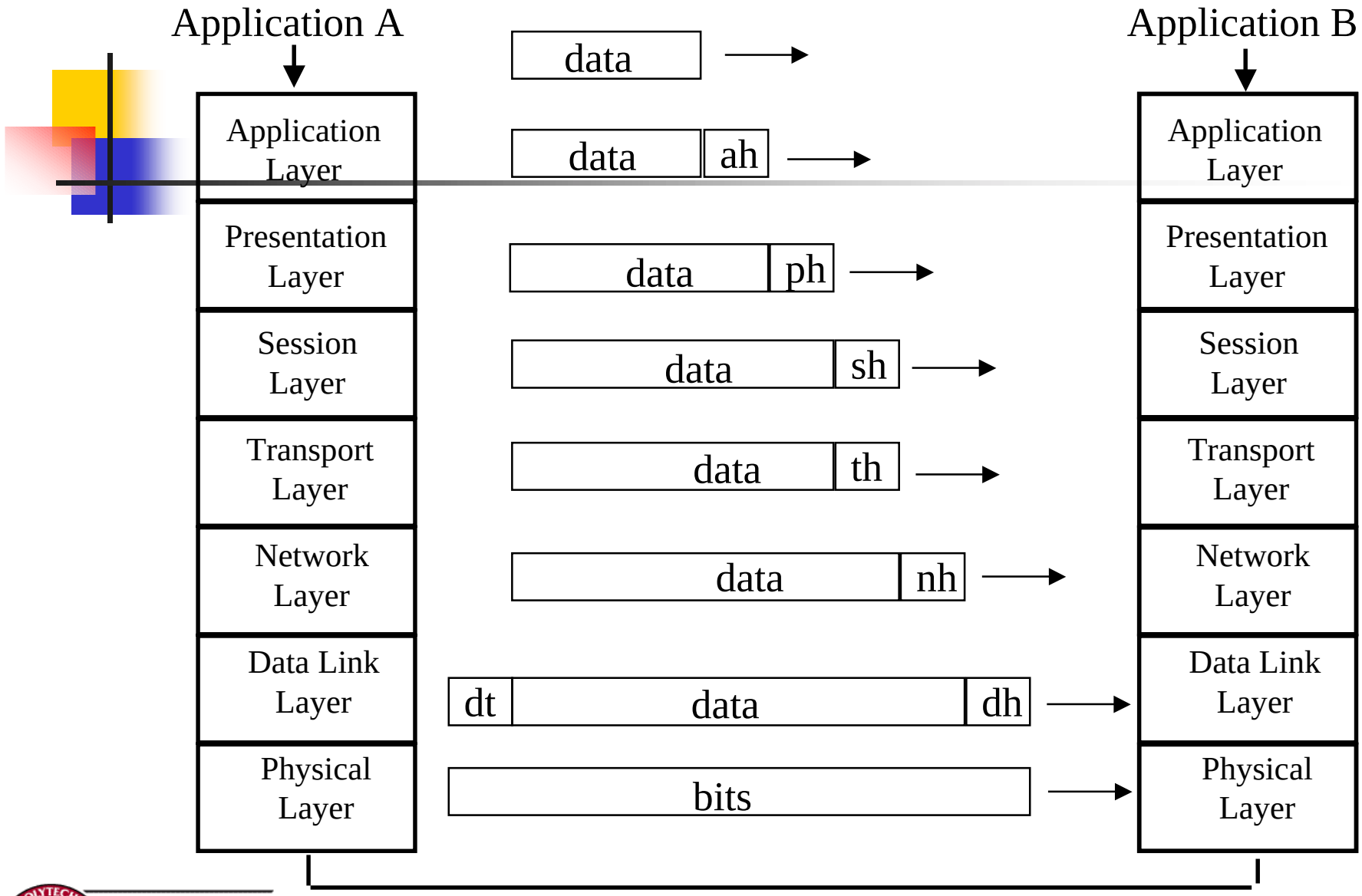
Communication Paradigms

- Using the Network Protocol Stack
- Remote Procedure Call - RPC
- Remote Object Invocation - Java Remote Method Invocation
- Message Queuing Services - Sockets
- Stream-oriented Services

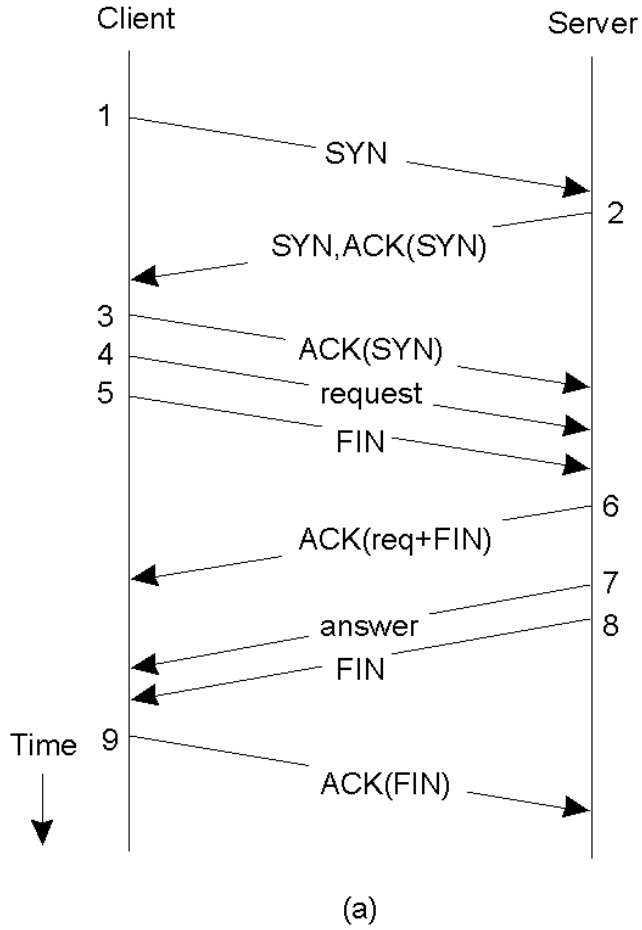
Network Stack - OSI Reference Model

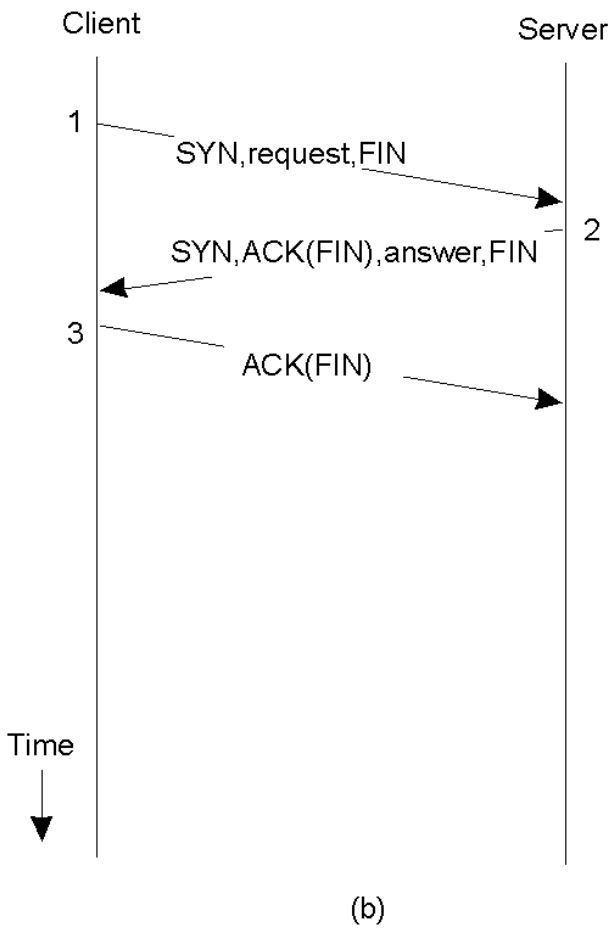


OSI Reference Model



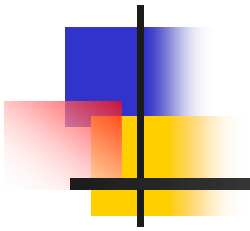
TCP Connection Overhead





Normal operation of TCP

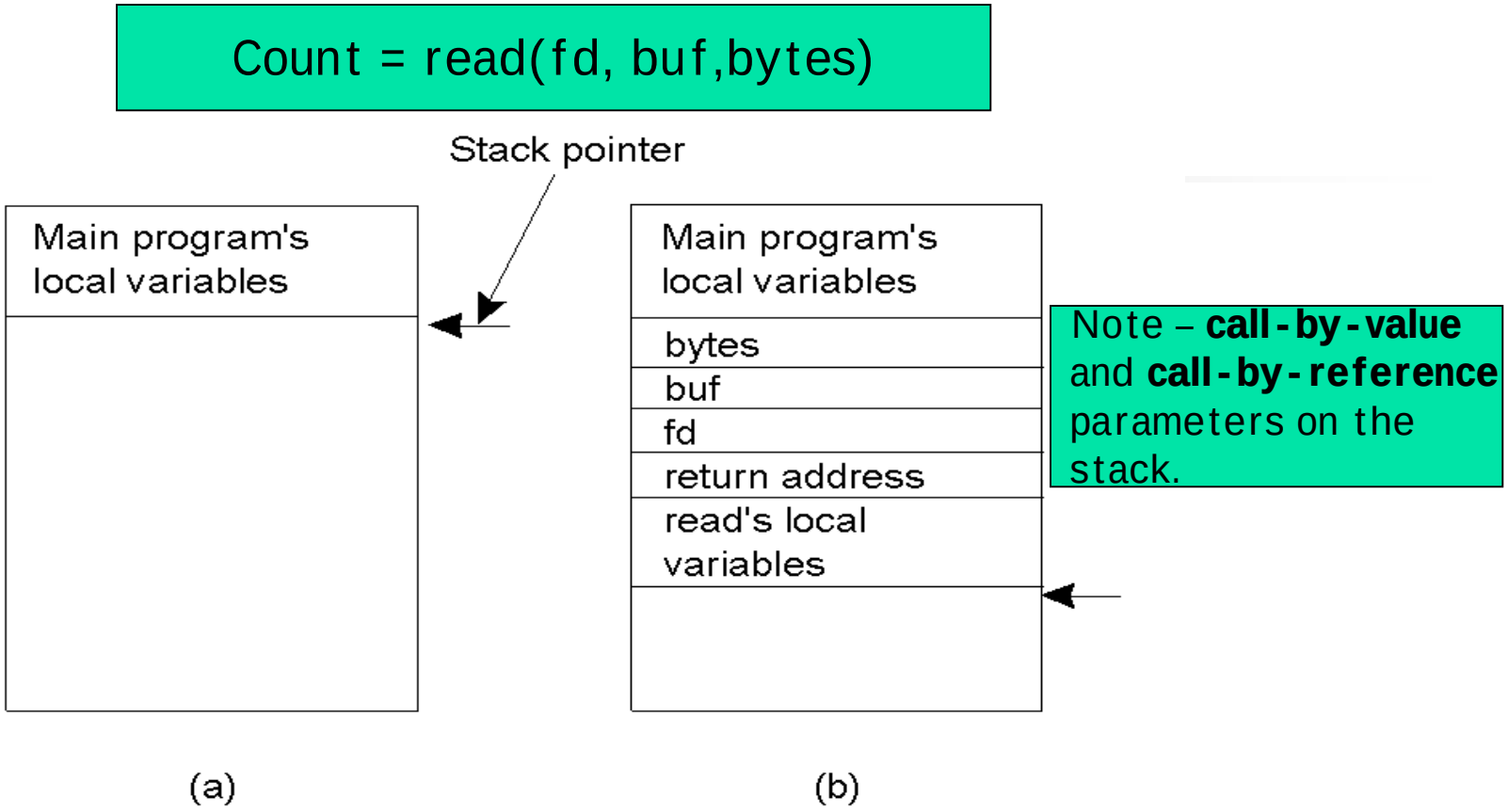
Transactional TCP



Remote Procedure Calls

RPC

Conventional Procedure Call



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

b) *The stack while the called procedure is active.*

Remote Procedure Call

- *RPC concept :: to make a remote procedure call appear like a local procedure call.*
- *The goal is to hide the details of the network communication (namely, the sending and receiving of messages).*
- *The calling procedure should not be aware that the called procedure is executing on a different machine.*

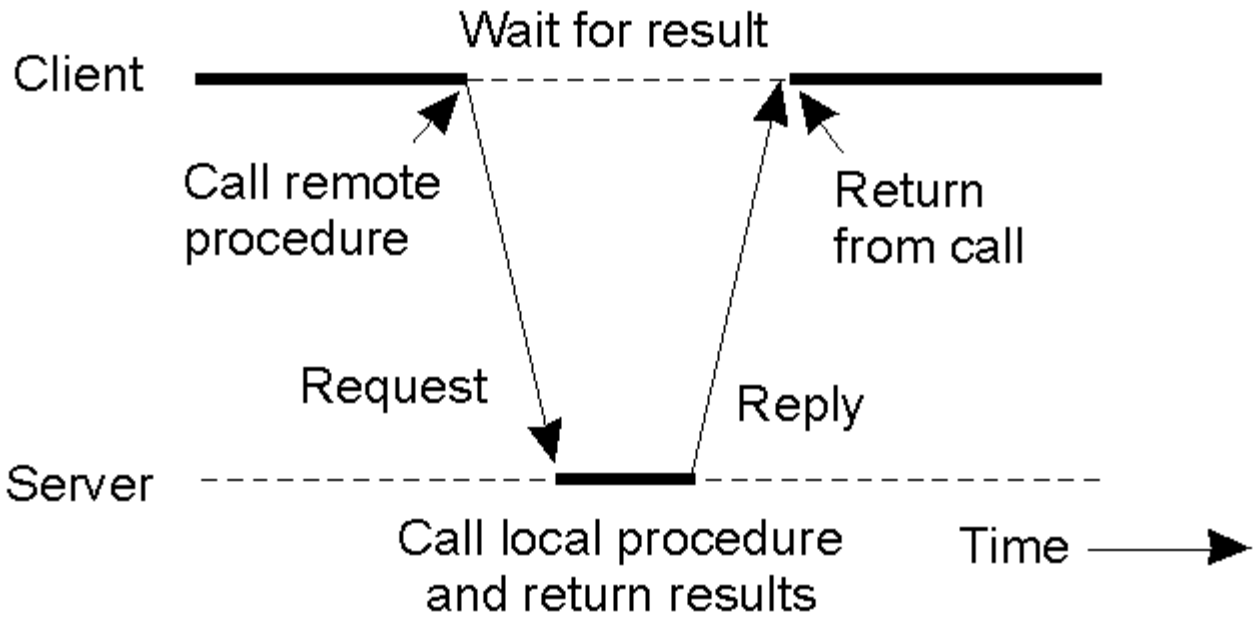
Remote Procedure Call

- When making a RPC:
  - *The calling environment is suspended.*
  - *Procedure parameters are transferred across the network to the environment where the procedure is to execute.*
  - *The procedure is executed **there.***
  - *When the procedure finishes, the results are transferred back to the calling environment.*
  - *Execution resumes as if returning from a regular procedure call.*

RPC differs from OSI

- *User does not open connection, read, write, then close connection – client may not even know they are using the network.*
- *RPC may **omit** protocol layers for efficiency. (e.g. diskless Sun workstations will use RPC for every file access.)*
- *RPC is well-suited for client-server interaction where the flow of control alternates.*

RPC between Client and Server



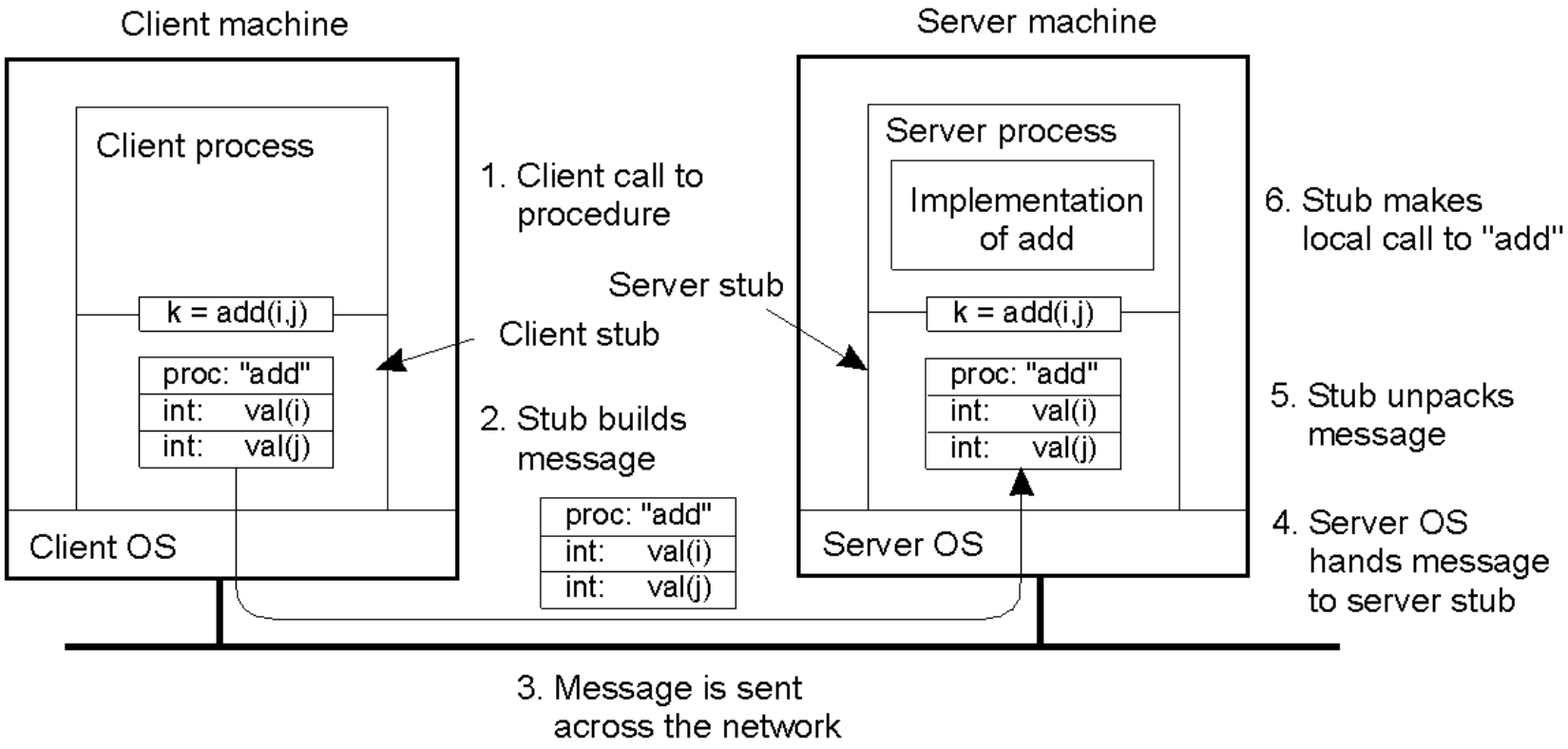
RPC Steps

1. The client procedure calls a client stub passing parameters in the normal way.
2. The client stub marshals the parameters, builds the message, and calls the local OS.
3. The client's OS sends the message (using the transport layer) to the remote OS.
4. The server remote OS gives transport layer message to a server stub.
5. The server stub demarshals the parameters and calls the desired server routine.

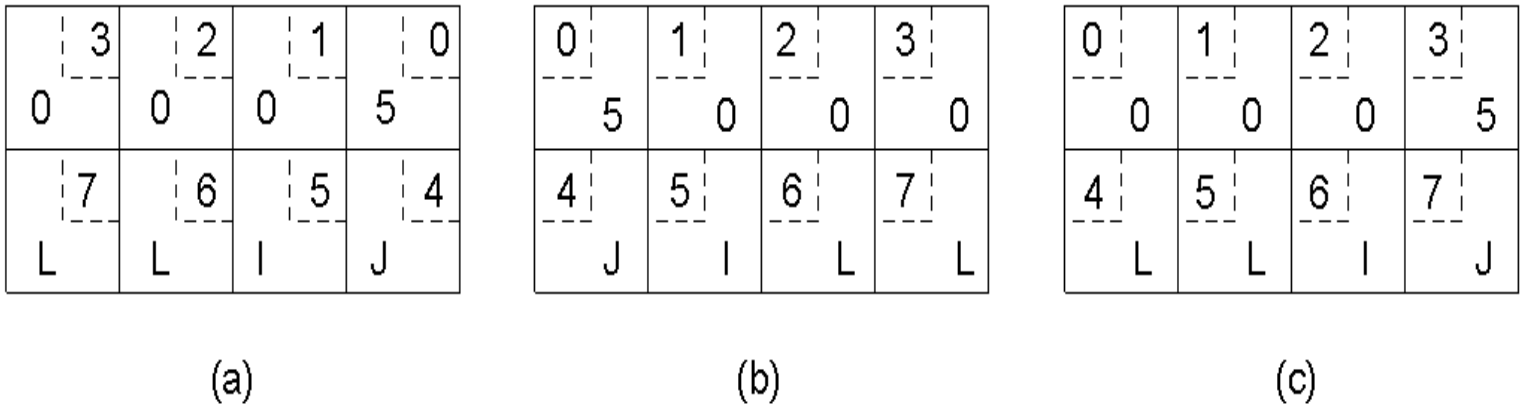
RPC Steps

6. The server routine does work and returns result to the server stub via normal procedures.
7. The server stub marshals the return values into the message and calls local OS.
8. The server OS (using the transport layer) sends the message to the client's OS.
9. The client's OS gives the message to the client stub
10. The client stub demarshals the result, and execution returns to the client.

RPC Steps


Client machine
Server machine
Client process
Server process
Implementation of add
k = add(i,j)
proc: "add"
int: val(i)
int: val(j)
1. Client call to procedure
Server stub
Client stub
2. Stub builds message
6. Stub makes local call to "add"
5. Stub unpacks message
4. Server OS hands message to server stub
Client OS
Server OS
3. Message is sent across the network

Passing Value Parameters



(a) (b) (c)

a) *Original message on the Pentium*

b) *The message after receipt on the SPARC*

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

Marshaling Parameters

- *Parameters must be marshaled into a standard representation.*
- *Parameters consist of simple types (e.g. integers) and compound types (e.g., C structures).*
- *The type of each parameter must be known to the modules doing the conversion into standard representation.*

Marshaling Parameters

- *Call-by-reference is not possible in parameter passing.*
- *It can be "simulated" by copy-restore. A copy of the referenced data structure is sent to the server, and upon return to the client stub the client's copy of the structure is replaced with the structure modified by the server.*

Marshaling Parameters

- *However, in general marshaling cannot handle the case of a pointer to an arbitrary data structure such as a complex graph.*

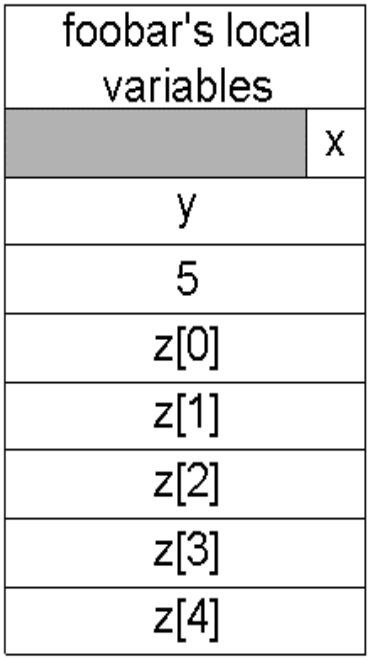
Parameter Specification and Stub Generation

The caller and the callee must agree on the format of the message they exchange, and they must follow the same steps when it comes to passing complex data structures.

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

(a)

A procedure



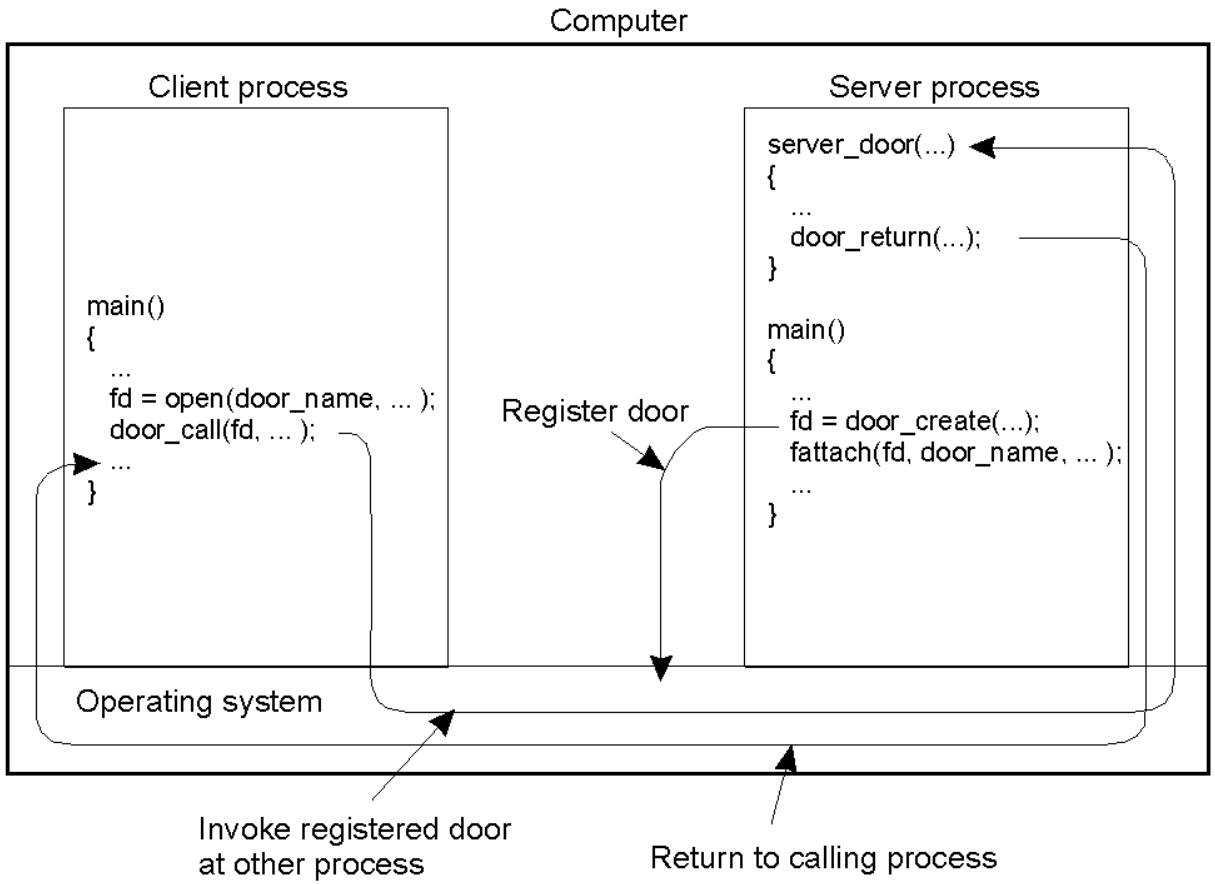
(b)

The corresponding message

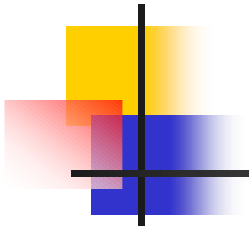
RPC Details

- An *Interface Definition Language (IDL)* is used to specific the interface that can be called by a client and implemented by the server.
- All RPC-based middleware systems offer an IDL to support application development.

Doors



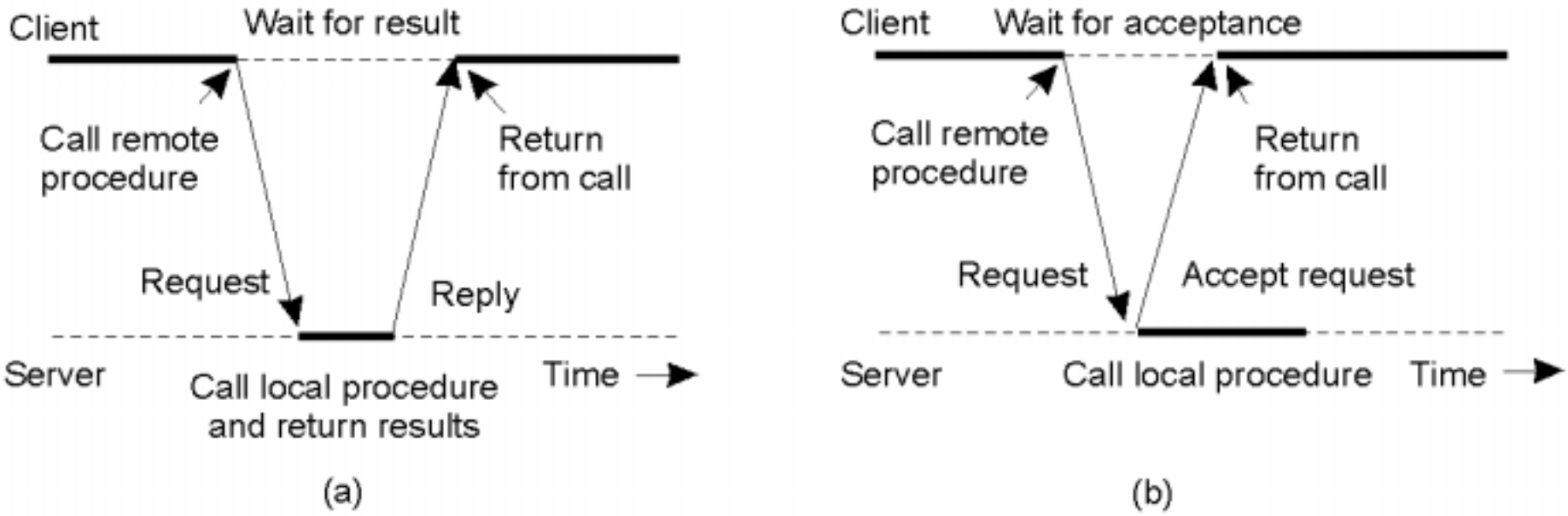
The principle of using doors as IPC mechanism.



Doors

- *A Door is a generic name for a procedure in the address space of a server process that can be called by processes colocated with the server.*
- *Doors require local OS support.*

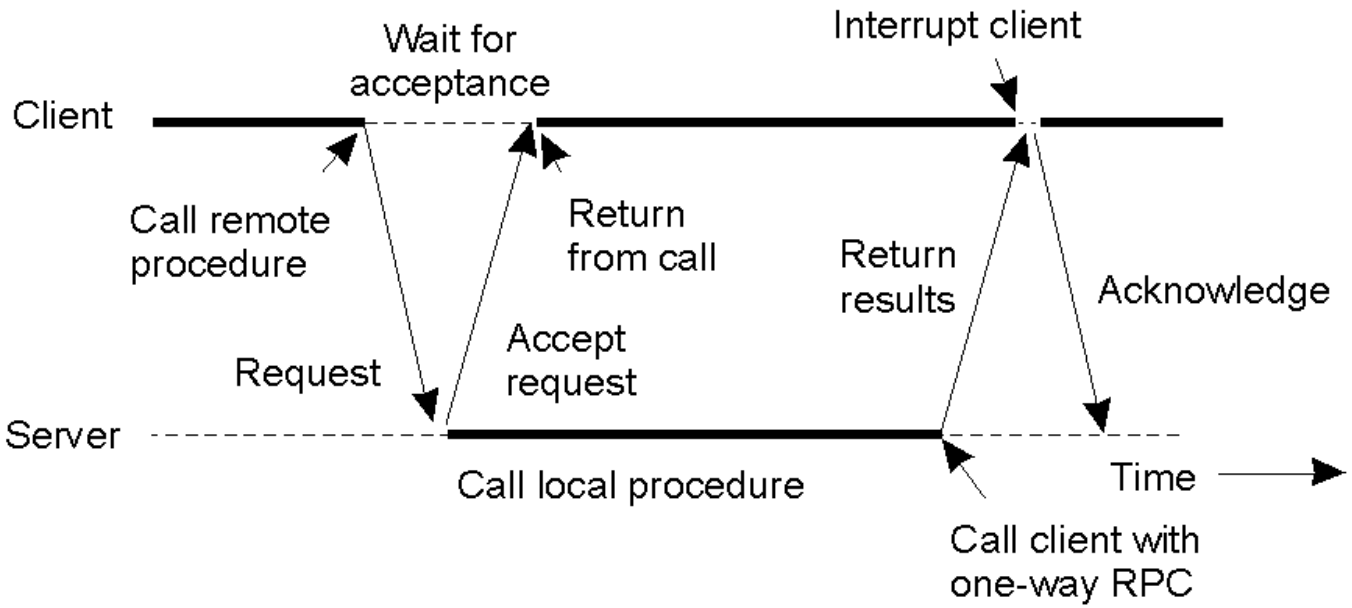
Asynchronous RPC



a) *The interconnection between client and server in a traditional RPC*

b) *The interaction using asynchronous RPC*

Asynchronous RPC's



A client and server interacting through two asynchronous RPCs