Remote Procedure Call (RPC)

History

- 1984: Birrell & Nelson
  - Mechanisms to call procedures on other machines
  - Processes on machine A can call procedures on machine B
    - A is suspended
    - Execution continues on B
    - When B returns, control passed back to A
- Goal: it appears to the programmer that a normal call is taking place

Procedure

- Same as *routine*, *subroutine,* and *function*. A procedure is a section of a program that performs a specific task.

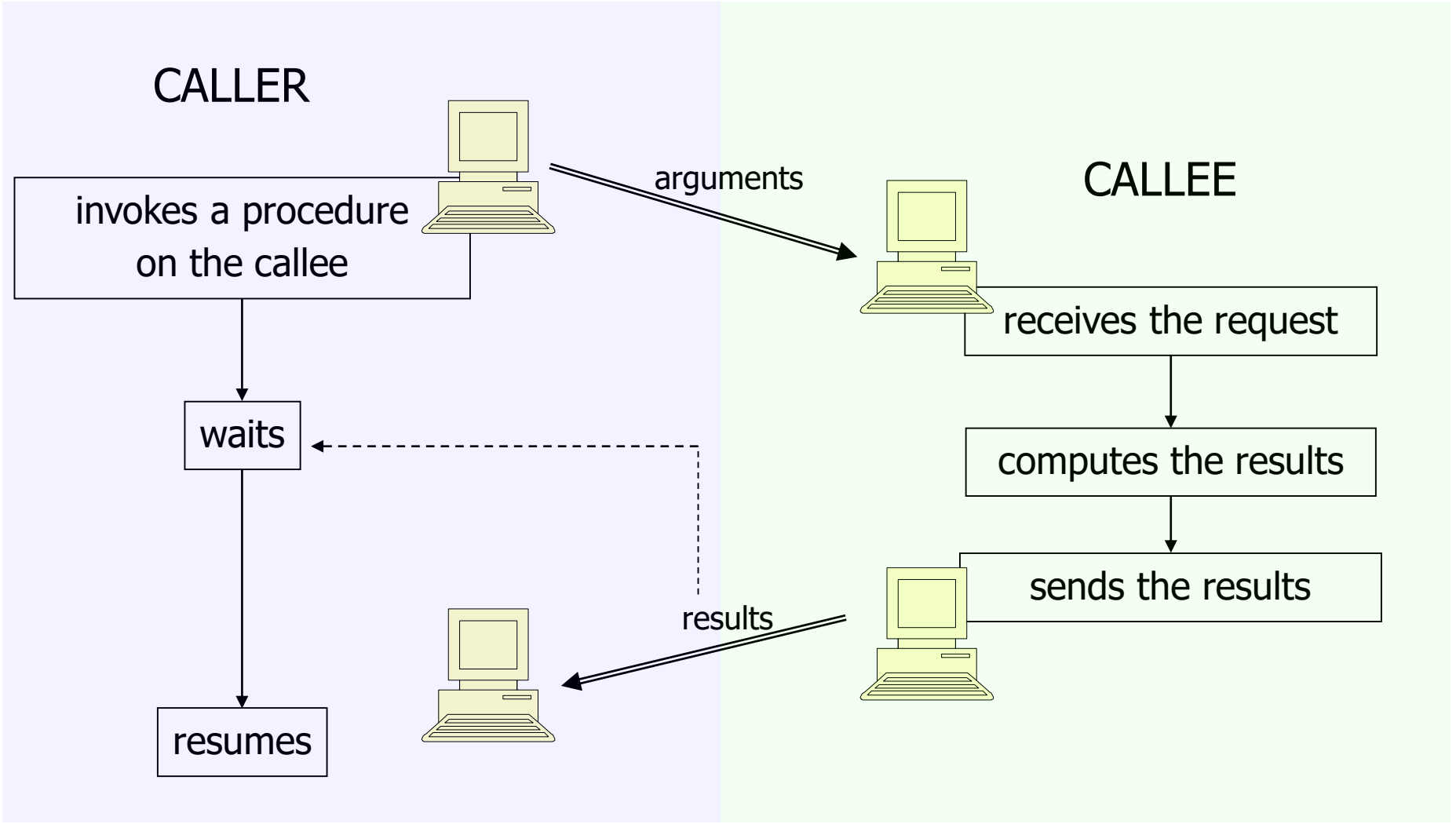
- An ordered set of tasks for performing some action.

What are Remote Procedure Calls (RPCs)

RPCs represent a set of communication paradigms that allow one procedure to call another procedure on a different machine.

1. one procedure (**caller**) calls another procedure (**callee**)
2. the **caller waits** for the result from the callee
3. the **callee receives the request**, computes the results, and then **send them to the caller**
4. the **caller resumes** upon receiving the results from the callee

What are Remote Procedure Calls (RPCs)



What are Remote Procedure Calls (RPCs)

PURPOSE: Make distributed computing easy!

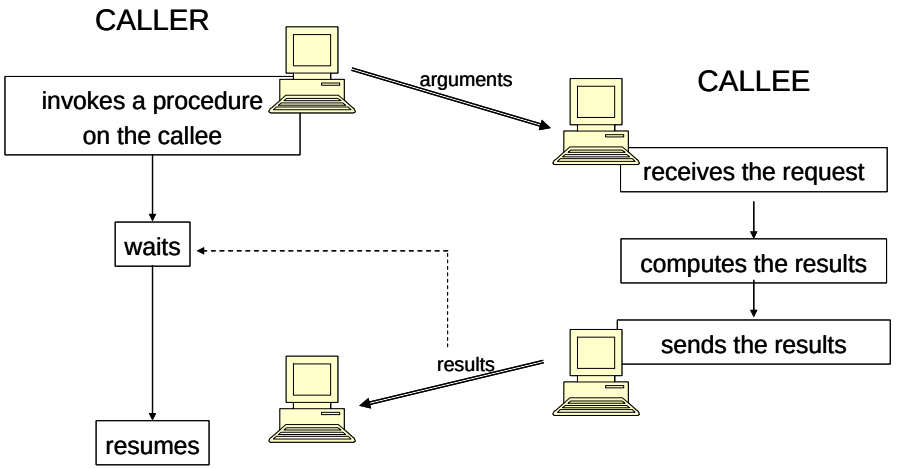
MAIN PRINCIPLE:

The procedure's communication patterns are transparent to the user.

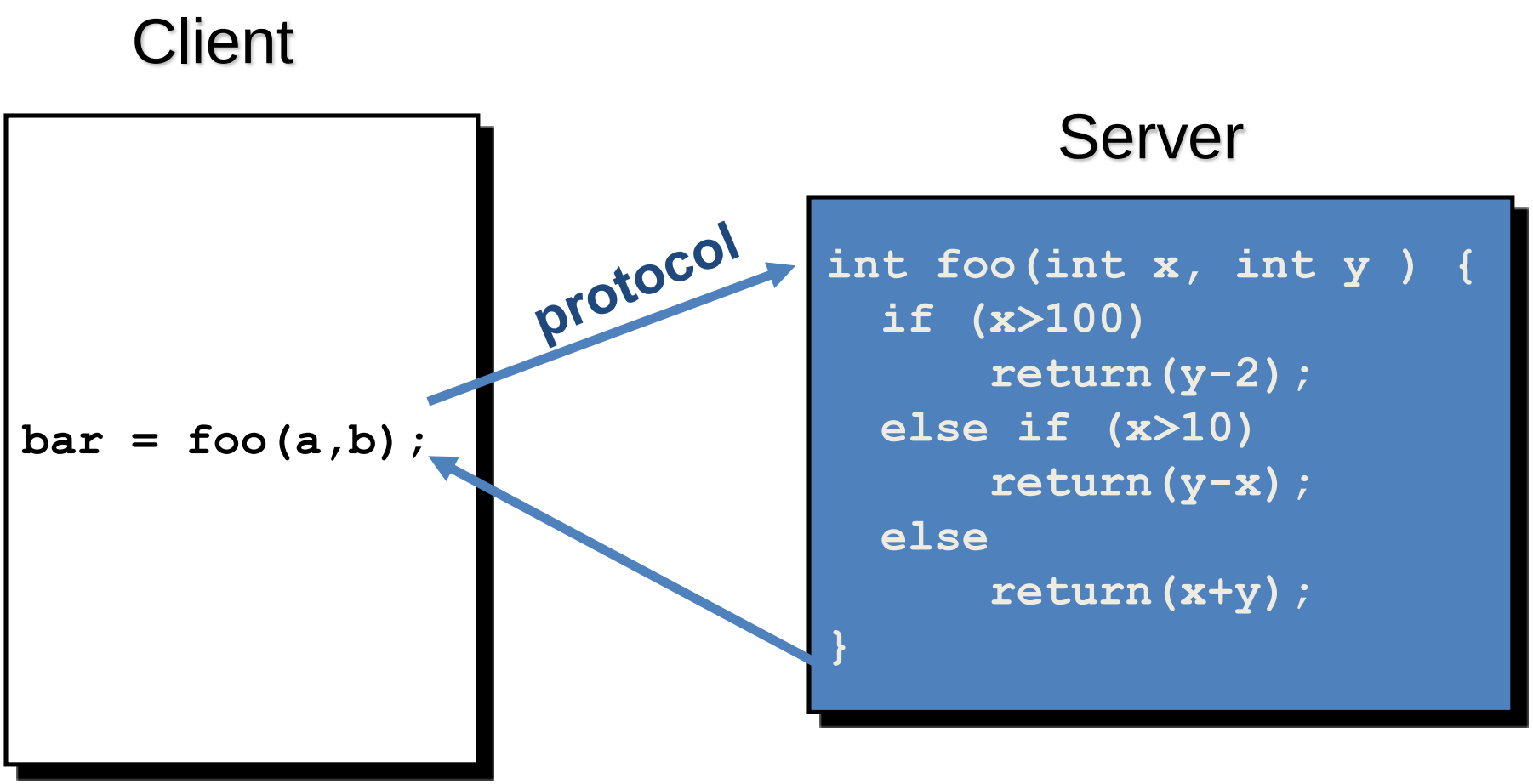
The procedure invocation looks just like a local procedure call.

Attractive aspects:

- simple semantics
- efficiency
- generality



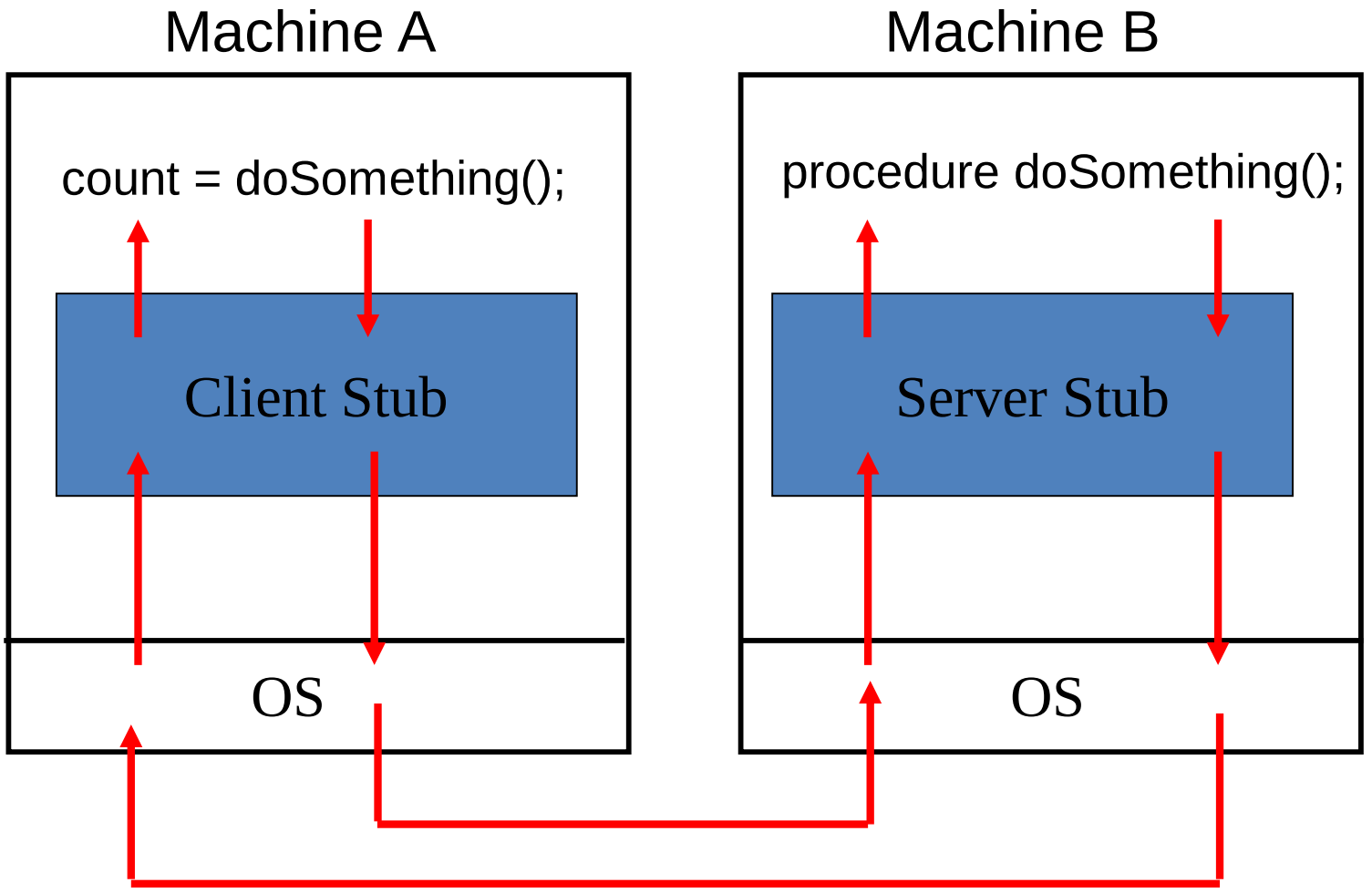
Remote Subroutine



Client-server architecture

- Client sends a request, server replies ,a response
  - Interaction fits many applications
  - Naturally extends to distributed computing
- Why do people like client/server architecture?
  - Scalable performance (multiple servers)
  - Provides fault isolation between modules
  - Central server:
    - Easy to manage
    - Easy to program

Client and Server



Remote procedure call

- A remote procedure call makes a call to a remote service look like a local call
  - RPC makes transparent whether server is local or remote
  - RPC allows applications to become distributed transparently
  - RPC makes architecture of remote machine transparent

Remote Procedure Call (RPC)

- The most common framework for newer protocols and for middleware
- Used both by operating systems and by applications
  - NFS is implemented as a set of RPCs
  - DCOM (Distributed Component Object Model) is a proprietary Microsoft technology for communication among software components distributed across networked computers.
  - CORBA (Common Object Request Broker Architecture) is a standard defined by the Object Management Group (OMG) that enables software components written in multiple computer languages and running on multiple computers to work together (i.e., it supports multiple platforms).
  - Java RMI (Java Remote Method Invocation) enables the programmer to create distributed Java technology-based to Java technology-based applications, in which the methods of remote Java objects can be invoked from other Java virtual machines, possibly on different hosts. etc., are just RPC systems

Remote Procedure Call (RPC)

- Fundamental idea: –
  - Server process exports an *interface* of procedures that can be called by client programs
- Clients make local procedure/function calls
  - Under the covers, procedure/function call is converted into a message exchange with remote server process

Developing with RPC

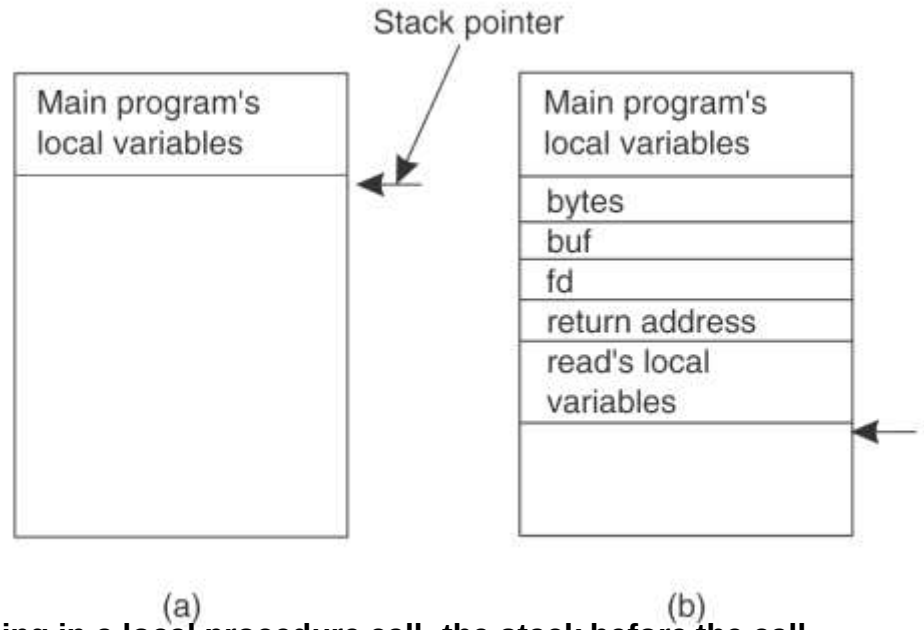
1. Define APIs between modules
   - Split application based on function, ease of development, and ease of maintenance
   - Don’t worry whether modules run locally or remotely
2. Decide what runs locally and remotely
   - Decision may even be at run-time
3. Make APIs bullet proof
   - Deal with partial failures

Ordinary (Conventional) procedure/function call

- **Consider a call in C like…..**

- **count = read(fd, buf, nbytes)**

- If the call is made from the main program…the stack will be as in fig(a) before the call
- The caller pushes the parameters onto the stack in order, last one first, as shown in fig(b)
- After read has finished, it puts the return value in a register, removes the return address, and transfer control back to the caller
- The caller then removes the parameter from the stack, returning to the original state



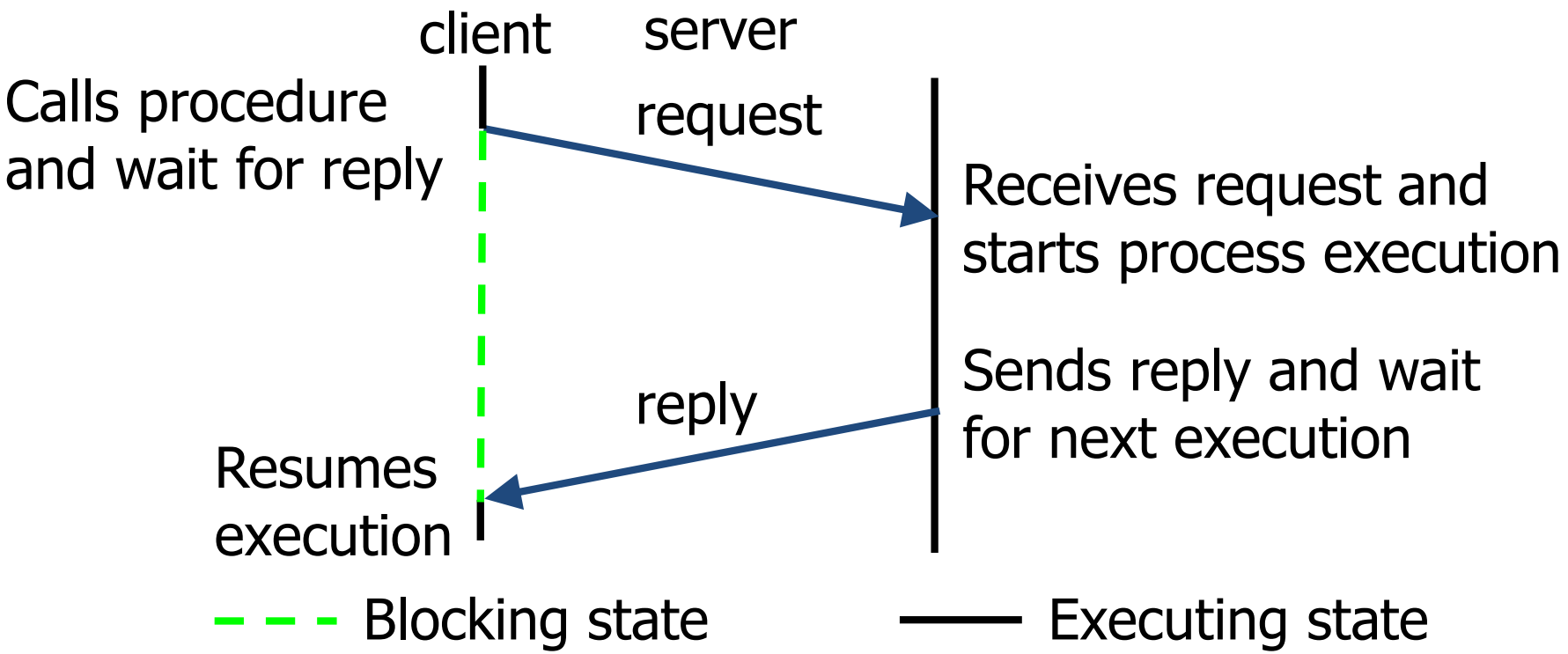
(a) Parameter passing in a local procedure call: the stack before the call
(b) The stack while the called procedure is active

Issues: call-by-value/reference

Conventional Procedure Call

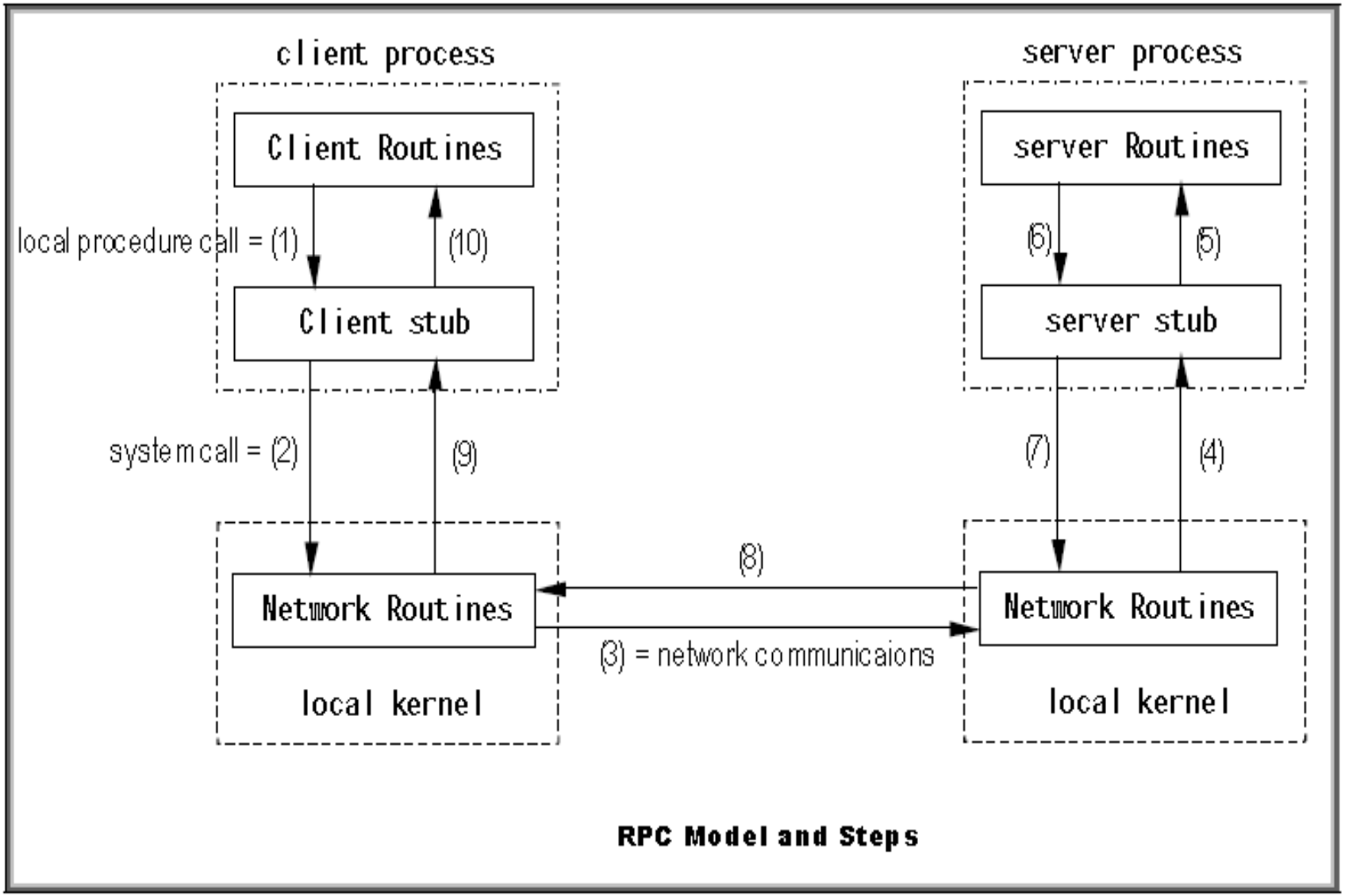
- Passing parameters by reference or value
- Value (fd, bytes): copy value to stack
- Reference (buf): copy address to stack

The RPC model



RPC to be transparent- the calling procedure should not be aware that called procedure is executing on a different machine

RPC Execution Steps



Cont…

(1) The **client** calls the procedure. It calls the client stub, which packages the arguments into network messages. Packaging is called **marshaling**.

(2) The **client stub** executes a system call (usually write or sendto) into the local kernel to send the message.

(3) The **kernel** uses the network routines (TCP or UDP) to send the network message to the remote host. Note that connection-oriented or connectionless protocols can be used.

(4) A **server stub** receives the messages from the kernel and **unmarshals** the arguments.

(5) The **server stub** executes a local procedure call to start the function and pass the parameters.

(6) The **server process** executes the procedure and returns the result to the server stub.

(7) The **server stub** marshals the results into network messages and passes them to the kernel.

(8) The **messages** are sent across the network.

(9) The **client stub** receives the network messages containing the results from the kernel using *read or recv.*

(10) The **client stub** unmarshals the results and passes them to the client routine.

Task of Stubs

- Client stub
  - Marshals arguments into machine-independent format
  - Sends request to server
  - Waits for response
  - Unmarshals result and returns to caller
- Server stub
  - Unmarshals arguments and builds stack frame
  - Calls procedure
  - Server stub marshalls results and sends reply

Marshalling Arguments

- *Marshalling* is the packing of function parameters into a message packet
  - The RPC stubs call 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

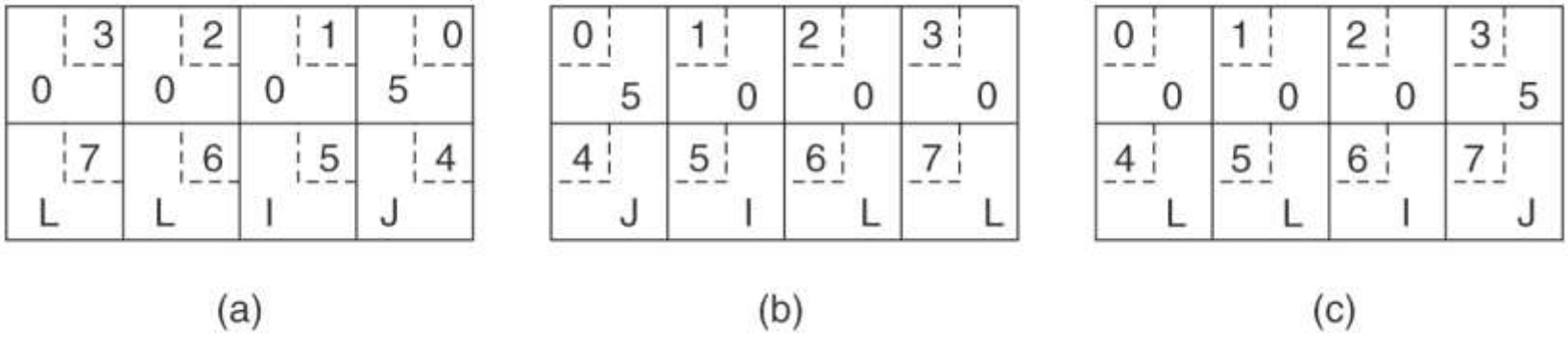
Marshalling

- Argument encoding sometimes called marshalling
- Decoding -> unmarshalling
- 3 issues
  - Data types
  - Conversion
  - Tagging

Design issues

Representation of data

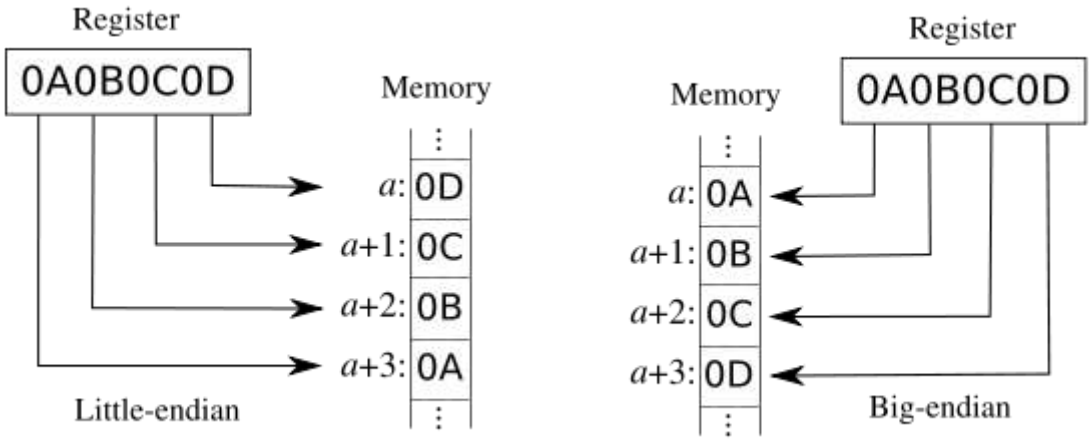
- Big endian *vs.* little endian



Sent by Intel Pentium

Rec'd by SPARC

After inversion


Register
0A0B0C0D
Memory
a: 0D
a+1: 0C
a+2: 0B
a+3: 0A
Little-endian
Register
0A0B0C0D
Memory
a: 0A
a+1: 0B
a+2: 0C
a+3: 0D
Big-endian

Representation of Data (continued)

- IDL – Interface Definition Language (enabling communication between software components that do not share a language – for example, between components written in C++ and components written in Java.) must also define representation of data on network
  - It is a language for specifying operations (procedures or functions), parameters to these operations, and data types.
- Each stub converts machine representation to/from network representation

Parameter Specification and Stub Generation

RPC protocol agreements include questions like:

How many bytes for each data type?

Big or little endian?

Data representation? (e.g. 1's or 2's complement, IEEE 754)

Transport service: connectionless or connection-oriented?

Stub Generation:

Using and IDL compiler

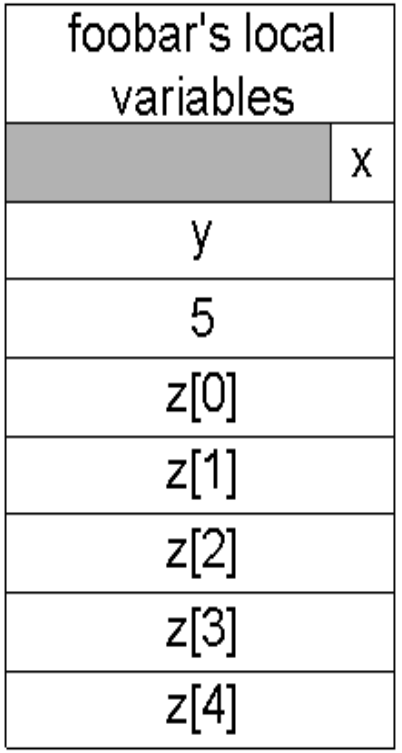
PURPOSE:

•Client stub for foobar knows that it must use the format of fig(b) &

•The server stub knows that incoming message for foobar will have the format of fig(b)

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

(a) A procedure

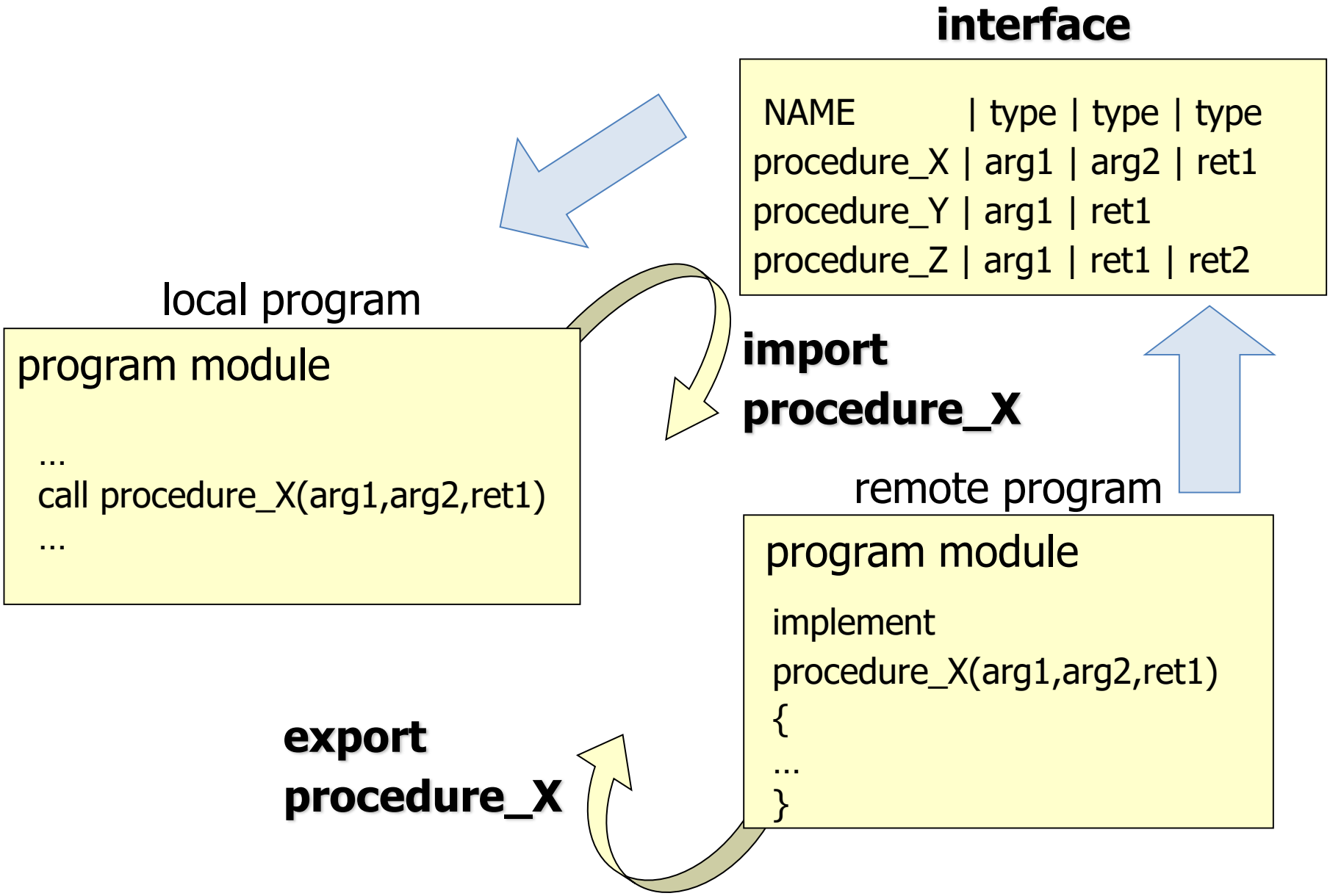


(b) The corresponding message

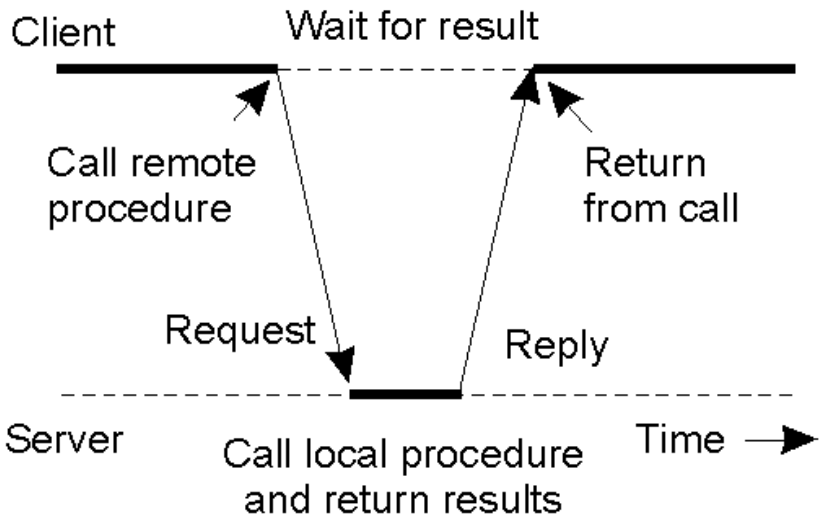
RPC Binding

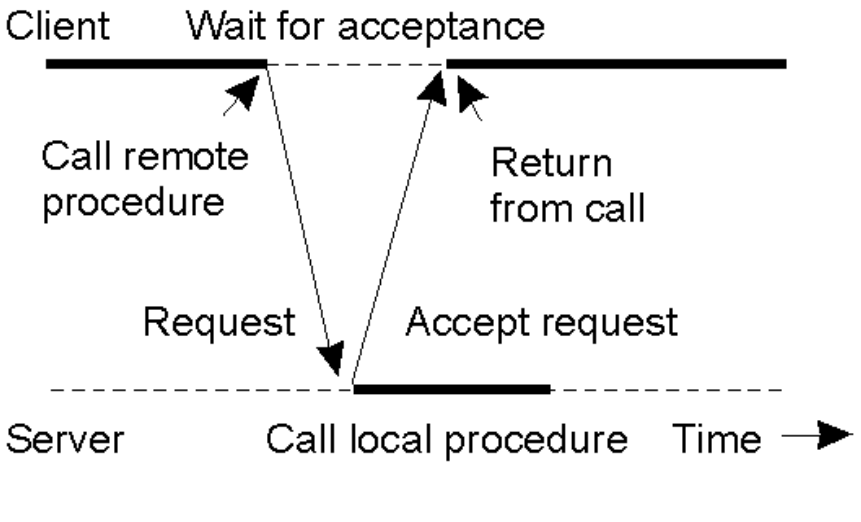
- Binding is the process of connecting the client to the server
  - the server, when it starts up, exports its interface
    - identifies itself to a *network name server*
    - tells *RPC runtime* that it is alive and ready to accept calls
  - the client, before issuing any calls, imports the server
    - RPC runtime uses the name server to find the location of the server and establish a connection
- The import and export operations are explicit in the server and client programs

Basic Concepts



Asynchronous RPC





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

b) The interaction using asynchronous RPC

Cont …

- Asynchronous RPC separates a remote procedure call from its return value, which resolves the following limitations of traditional, synchronous RPC

- Multiple outstanding calls from a single-threaded client
- Slow or delayed clients
- Slow or delayed servers
- Transfer of large amounts of data

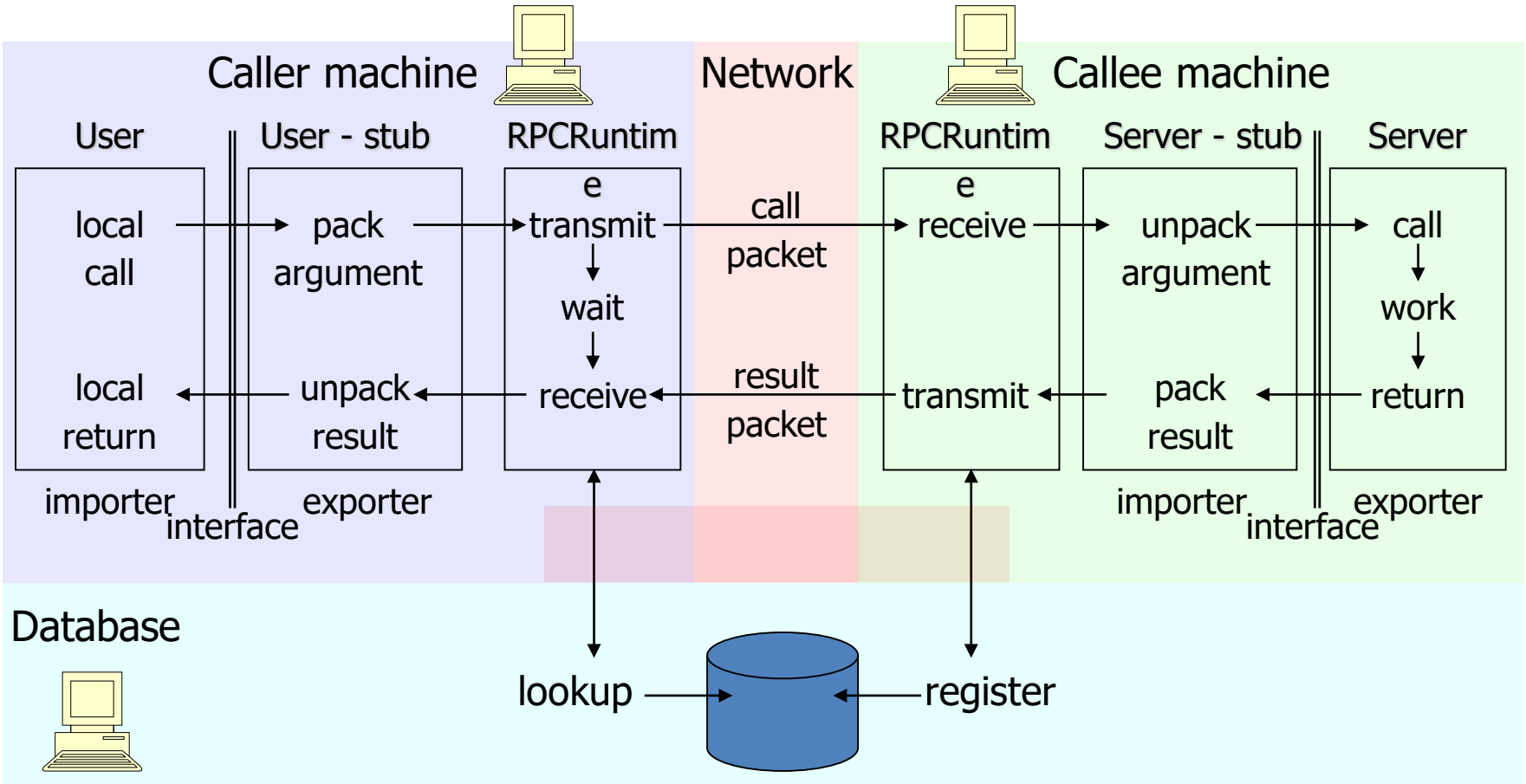
RPC Issues (1)

- transparency-Applications should be prepared to deal with RPC failure
- RPC is more vulnerable to failure
  - machine failure (Server crash/Client crash while server is still running code for it)
  - communication failure
- address based arguments
  - the address space is not shared
- programming integration
  - integration into programming environment
- data integrity

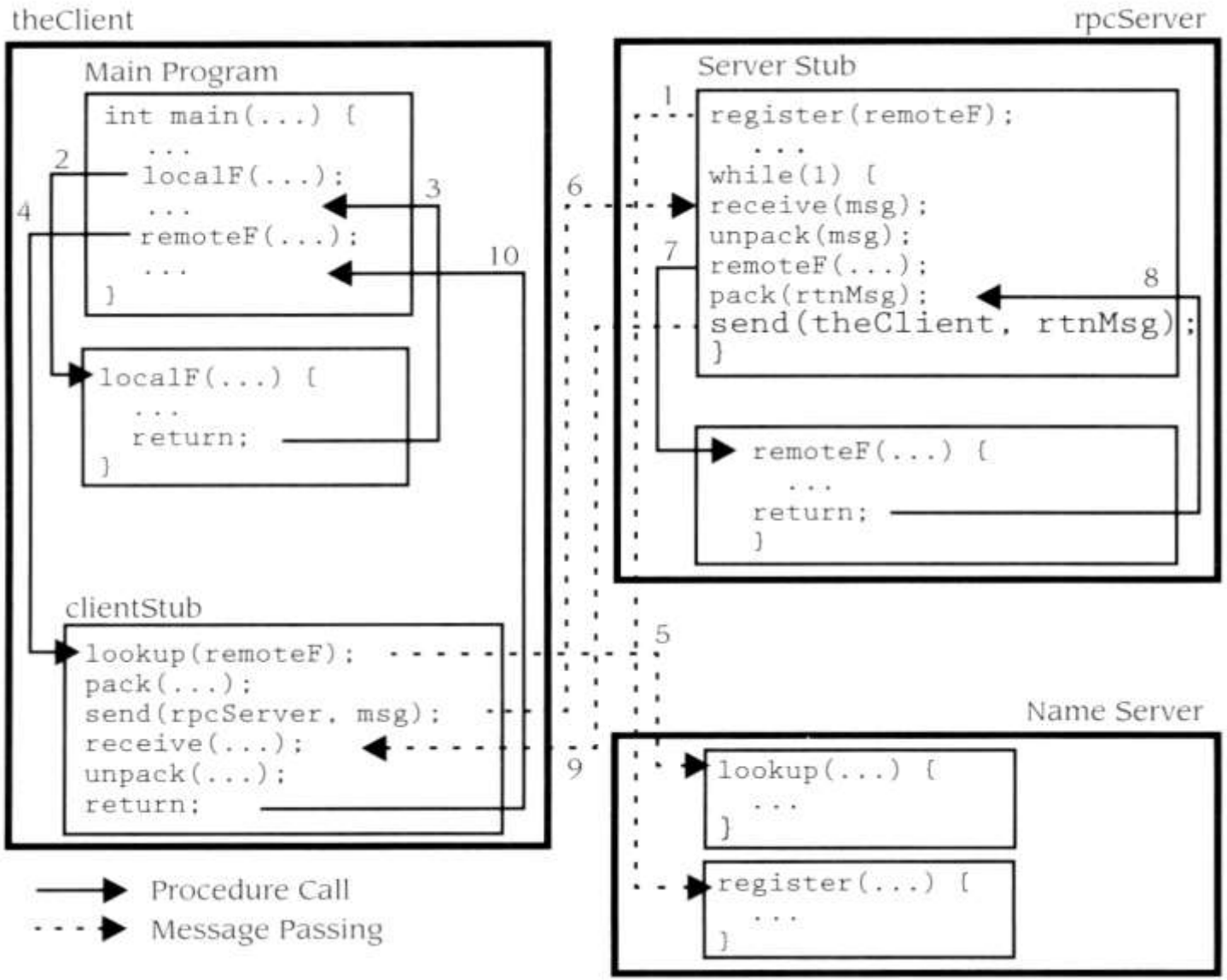
RPC Issues (2)

- data transfer protocols
  - network protocols
- binding
  - caller determines
    - the location
    - the identity
  - … of the callee
- security
  - open communication network
  - Authenticate client
  - Authenticate server

RPC Implementation – Overview



RPC Implementation – RPC flow



* *Garry Nutt*; **Operating systems**; 2nd Edition, Addison Wesley

RPC Implementation- Using rpcgen

- rpcgen is a compiler that generates C source code from a protocol specification file (normally with .x). rpcgen produces the following C files from a input file named proto.x
- **proto.h**
  Header file for data types and function prototypes.
- **proto_xdr.c**
  XDR routines to encode/decode arguments and return values, used by both the client and the server.
- **proto_svc.c**
  Server side stub. This contains mainly the code to start and register the server and the dispatch routine that receives client call requests and dispatch the calls to the server implementation functions.
- **proto_clnt.c**
  Client side stub.