Operating Systems
Remote Procedure Call

Andrey Shadrin

shavez@wjpserver.cs.uni-sb.de

Saarland University

Informatik, Saarbrücken

4. Oktober 2004

Content

- Introduction
- RPC
- RPC issues
- Sun RPC
- NFS

Introduction: Motivations

- One of the aim of distributed computation:
  - to perform transfer of data across network
  - to call a procedure on another machine
- Regards a network as another I/O device
  - Client and server are wholly responsible for message exchange
  - Primitives
    - [connect]
    - write(send) and read (recieve)
    - [disconnect]
- To make distributed computing more look centralized
  - Hand-coding of I/O is not way to go

Introduction: Birrell and Nelson 1984

- Process on machine M1 calls a procedure P on machine M2
  - M1 is suspended
  - M2 executes P
  - When M2 returns, then conrol backs to M1
- The call of a remote procedure should be look as much as possible like a local one
- Neither message passing nor I/O at all are visible to the programmer
- This method is known as **Remote Procedure Call** (RPC)

Introduction: LPC and RPC

- How works the local procedure call(LPC) ?
  - copies the input parameters and return address to the stack
  - performs the procedure
  - when returns, puts the result into register, transfers control back, removes return address and parameters from the stack
- Difference
  - Error handling
  - Global variables
  - Perfomance
  - Security

Content

- Introduction
- **RPC**
- RPC issues
- Sun RPC
- NFS

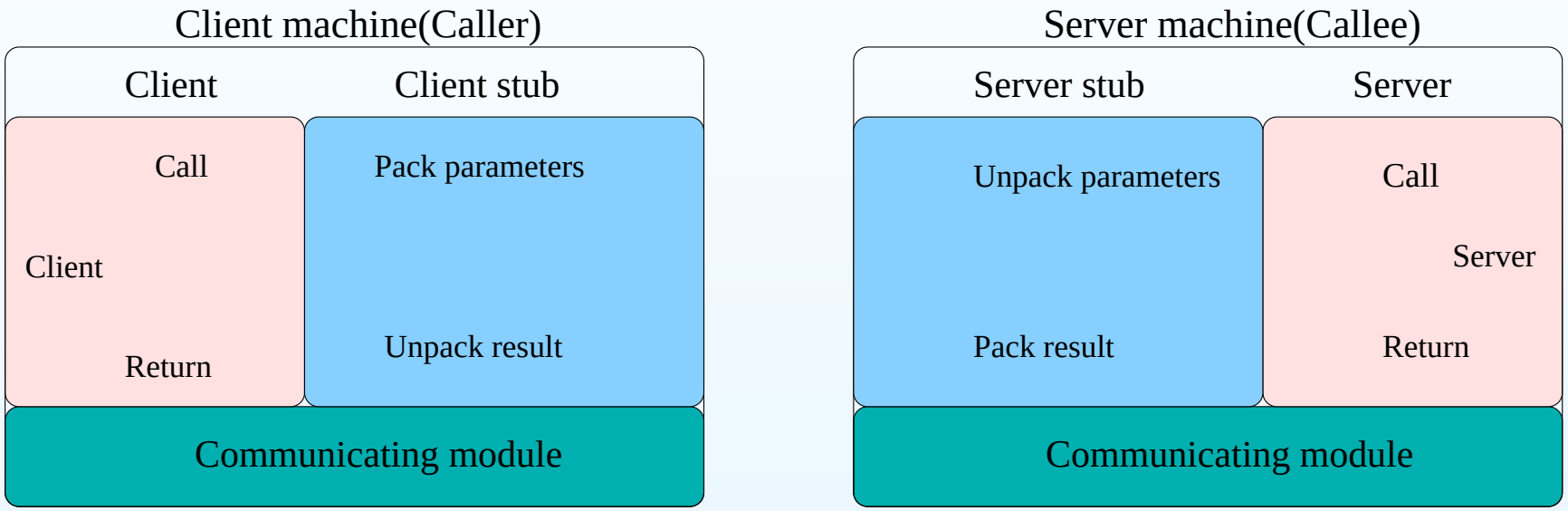
RPC: Classes of RPC systems

- the RPC mechanism is integrated with a particular programming language
- a special-purpose Interface Definition Language (IDL Hristo Pentchev) is used for describing the interfaces between clients and servers

RPC: Stub components

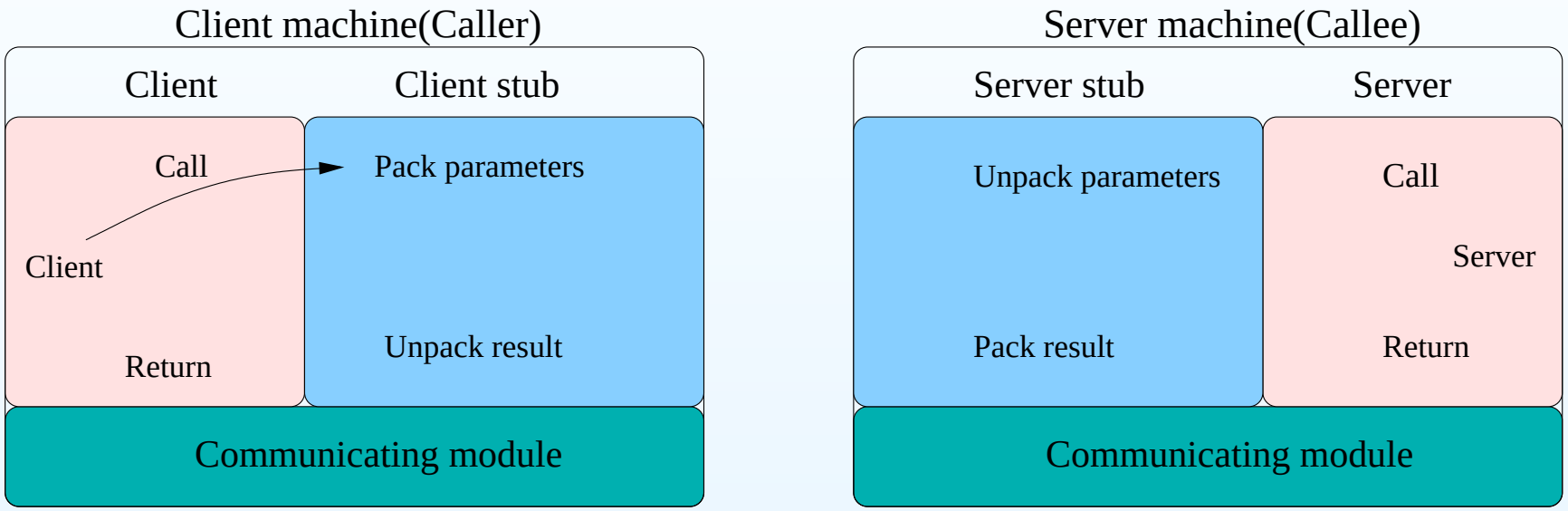
- No architectural supports for RPCs
  - *Simulate* it with tools we have (local procedure calls)
  - Simulation makes RPC a language-level construct
    - instead of an OS construct
- Creating stub components to make it appear to the user that the call is local
- The stubs are responsible for managing all details of the remote communication between client and server

RPC: The calls and messages in an RPC



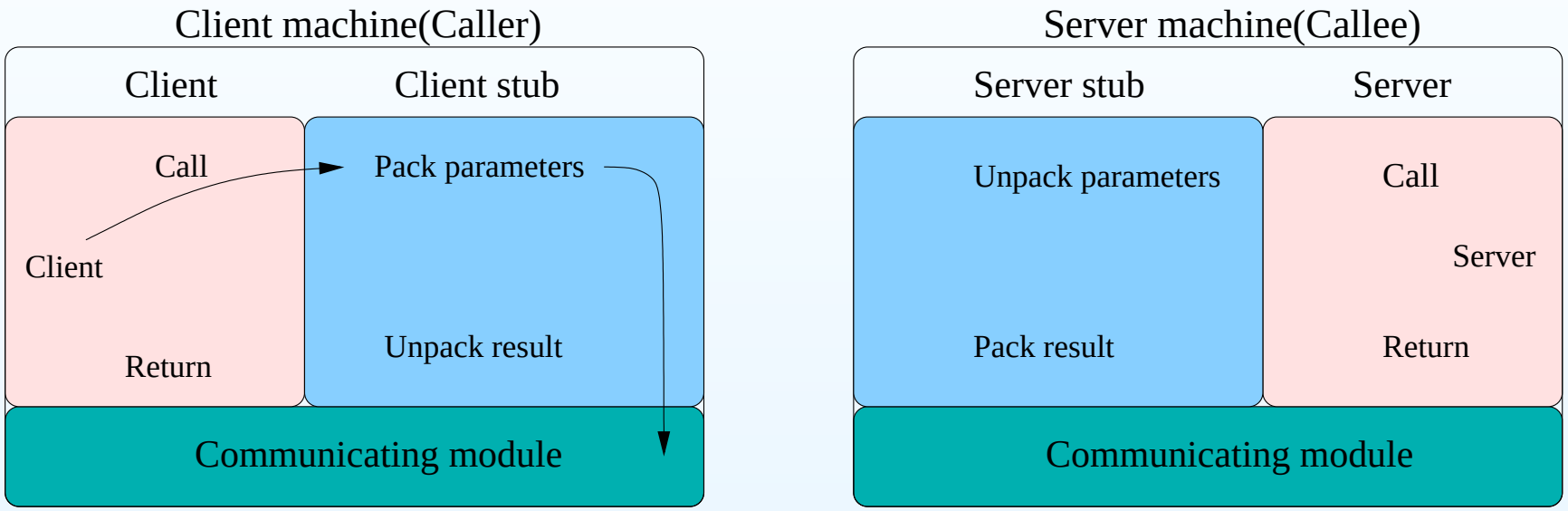
Initial situation

RPC: The calls and messages in an RPC



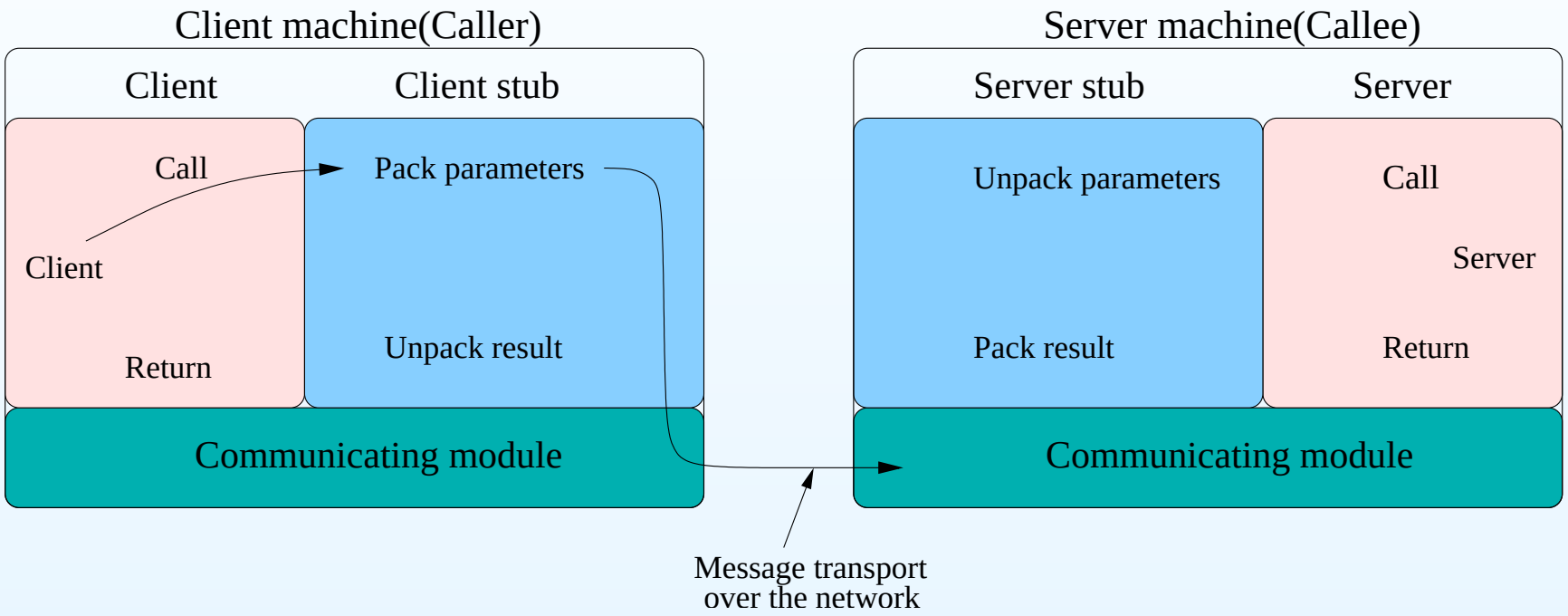
The client procedure calls the client stub

RPC: The calls and messages in an RPC



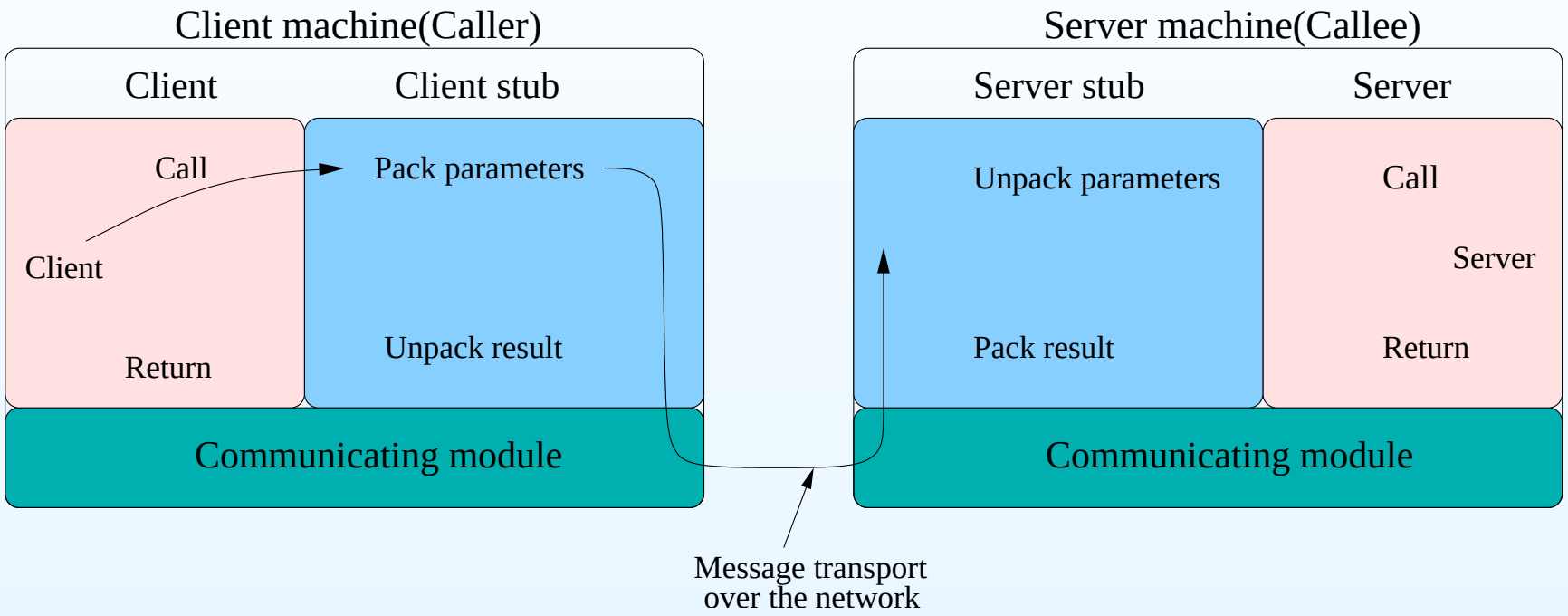
The client stub builds a message
and traps to the Communicating Module (CM)

RPC: The calls and messages in an RPC



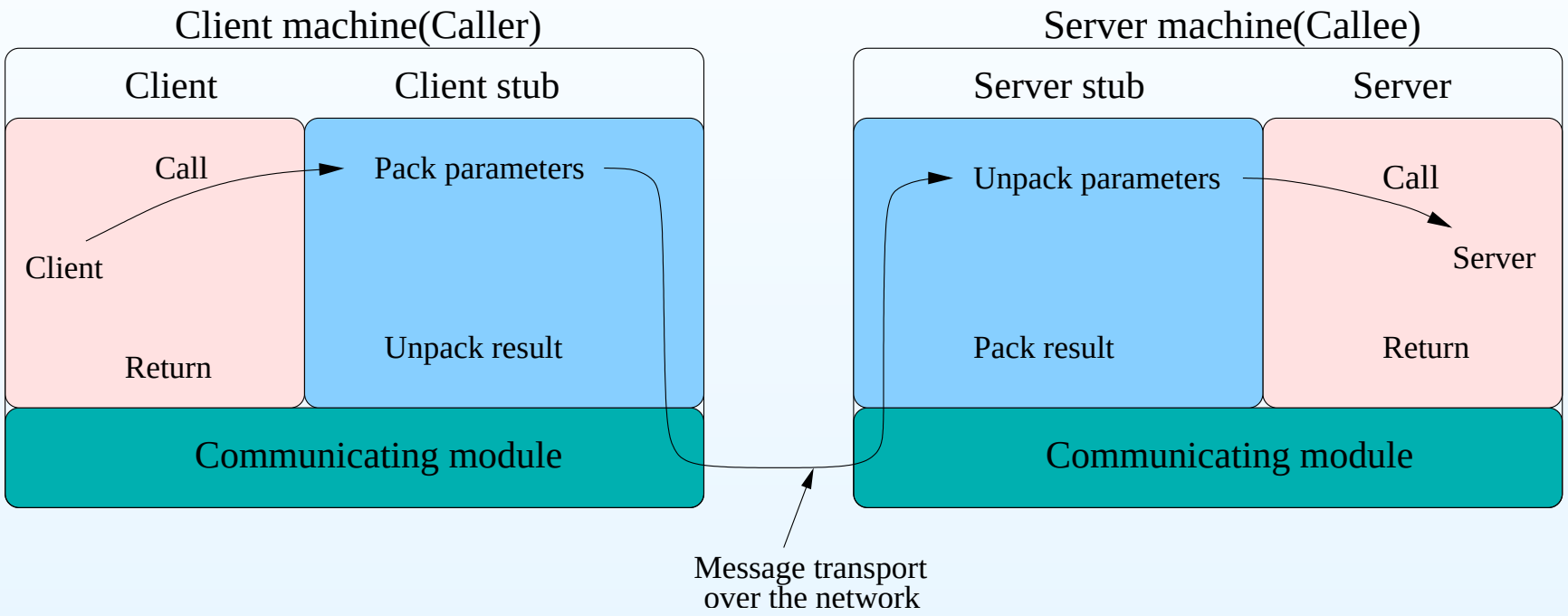
The CM sends the message to the remote's CM

RPC: The calls and messages in an RPC



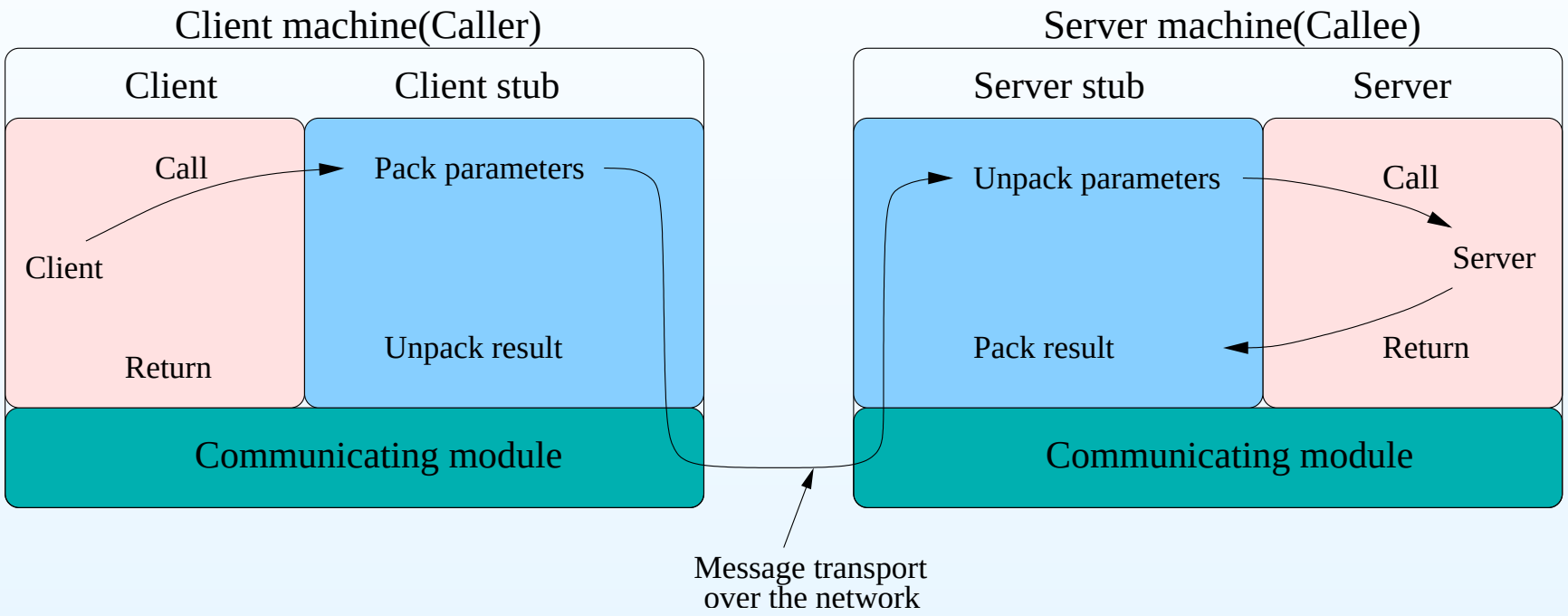
The remote's CM gives the message to the server stub

RPC: The calls and messages in an RPC



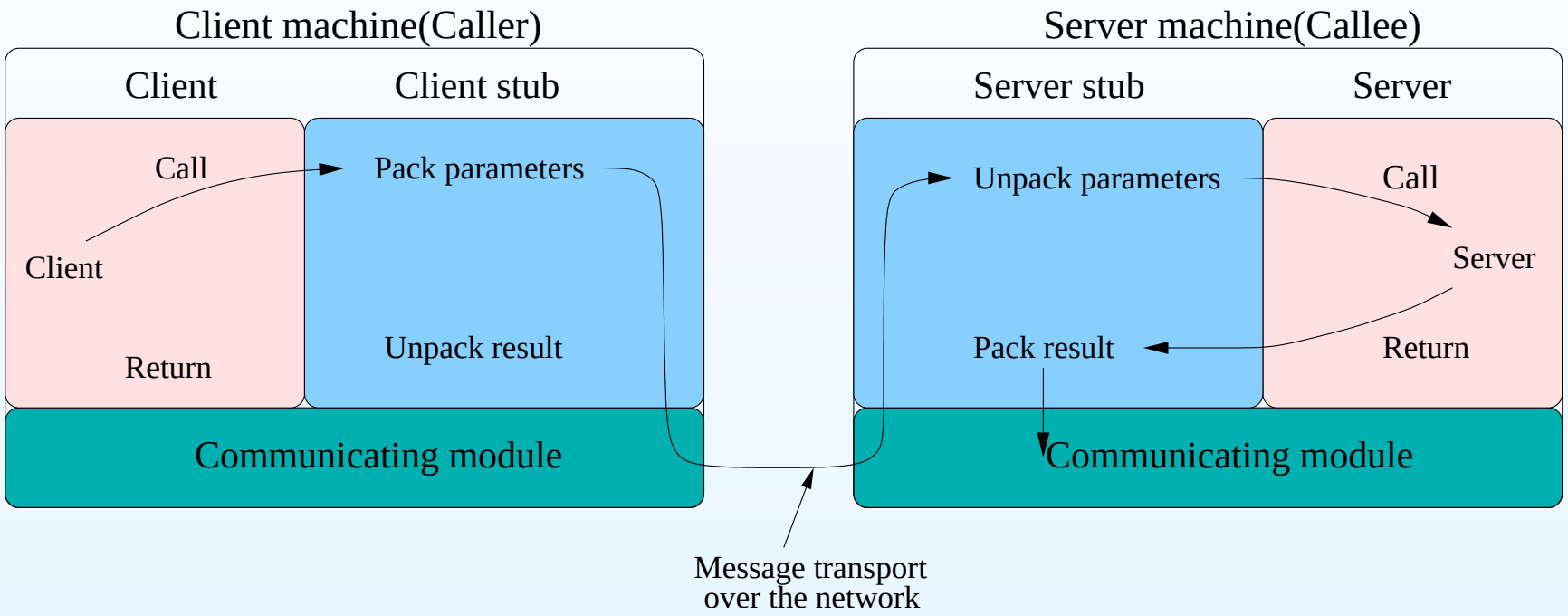
The server stub unpacks the parameters and calls the server

RPC: The calls and messages in an RPC



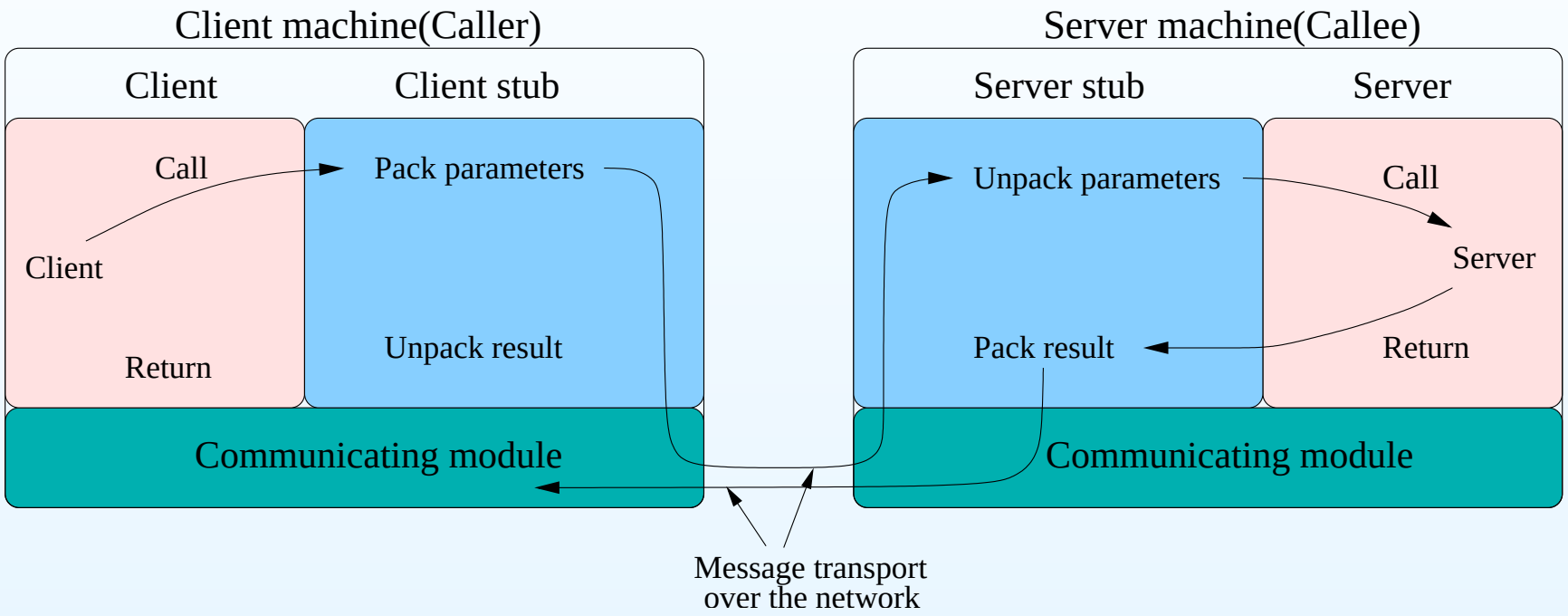
The server does the work and returns the result to the server stub

RPC: The calls and messages in an RPC



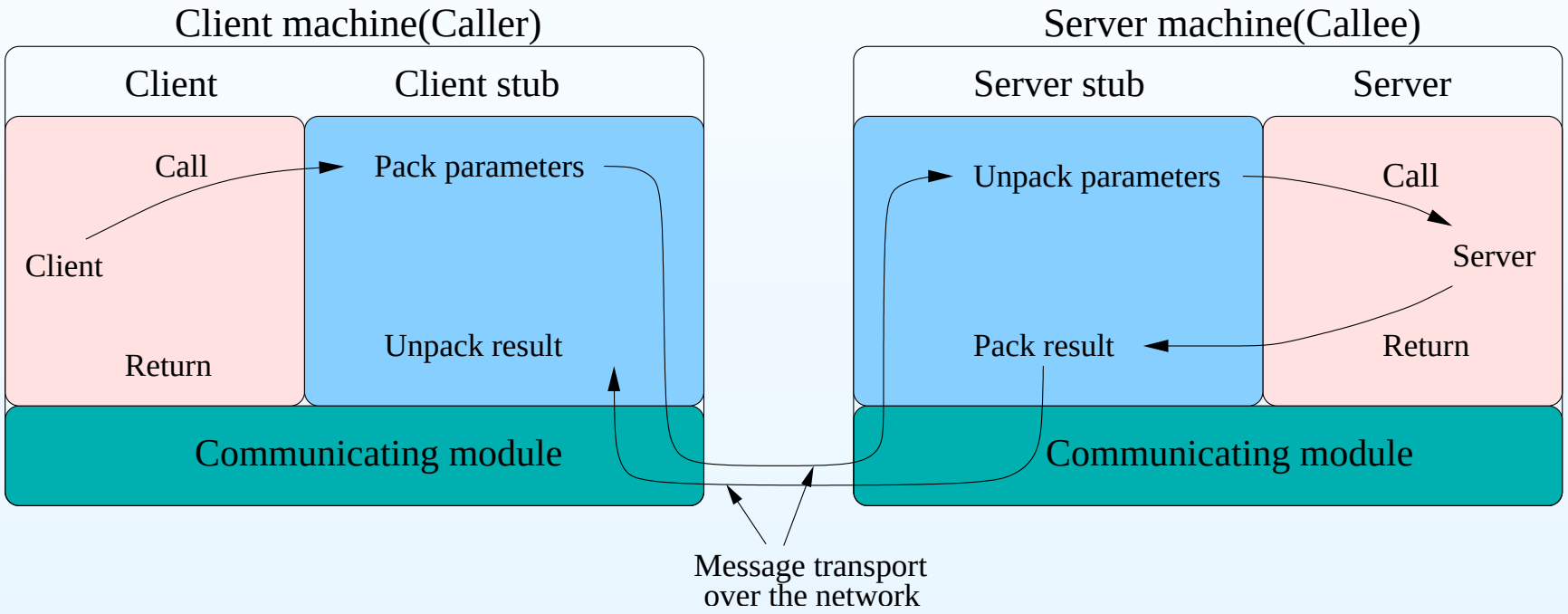
The server stub packs it in a message and traps to the CM

RPC: The calls and messages in an RPC



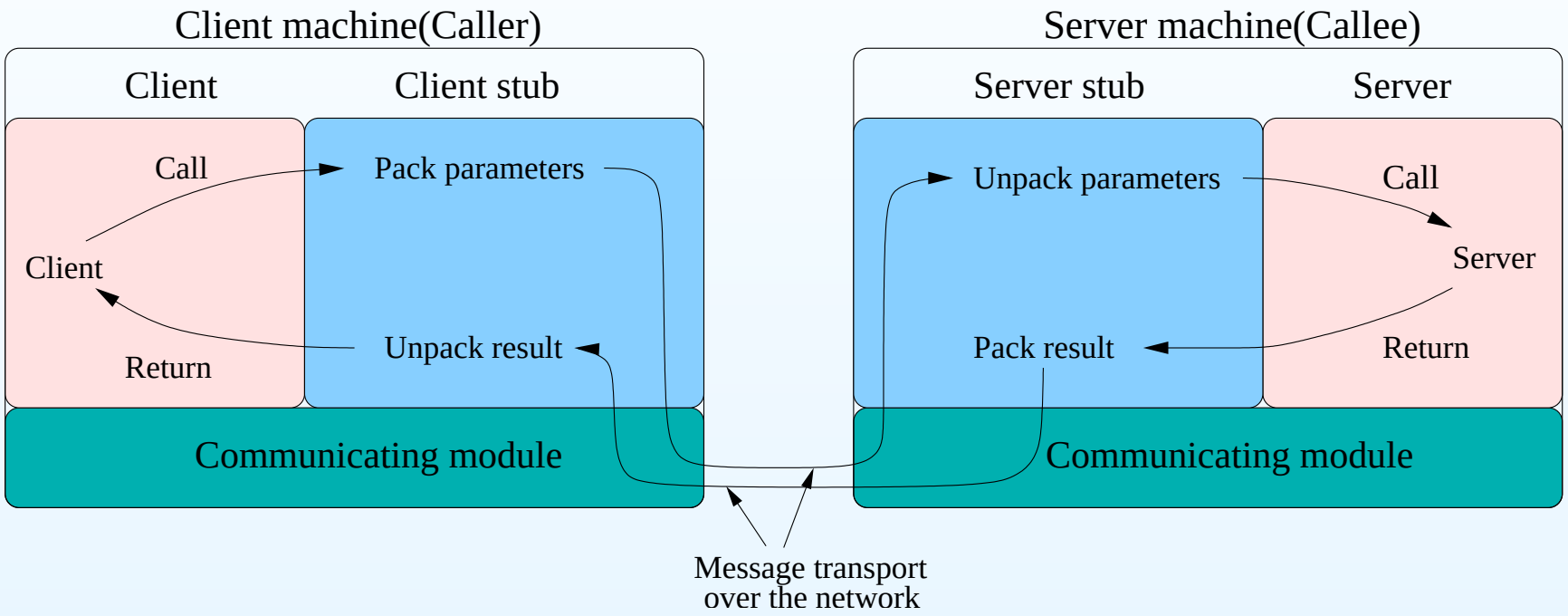
The remote's CM sends the message to the client's CM

RPC: The calls and messages in an RPC



The client's CM gives the message to the client stub

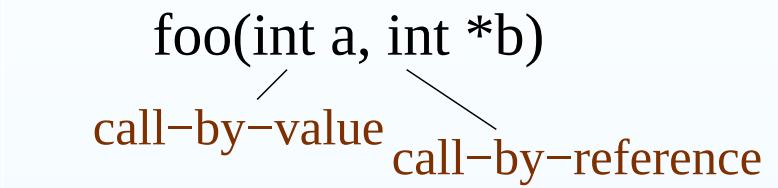
RPC: The calls and messages in an RPC



The stub unpacks the result and returns to the client

RPC: Parameter passing

- call-by-value
  - just copy data to the message
- call-by-reference
  - makes no sense if there is not shared memory
  - use a call-by-value instead
  - how to support complex data structures?
- call-by-copy/restore



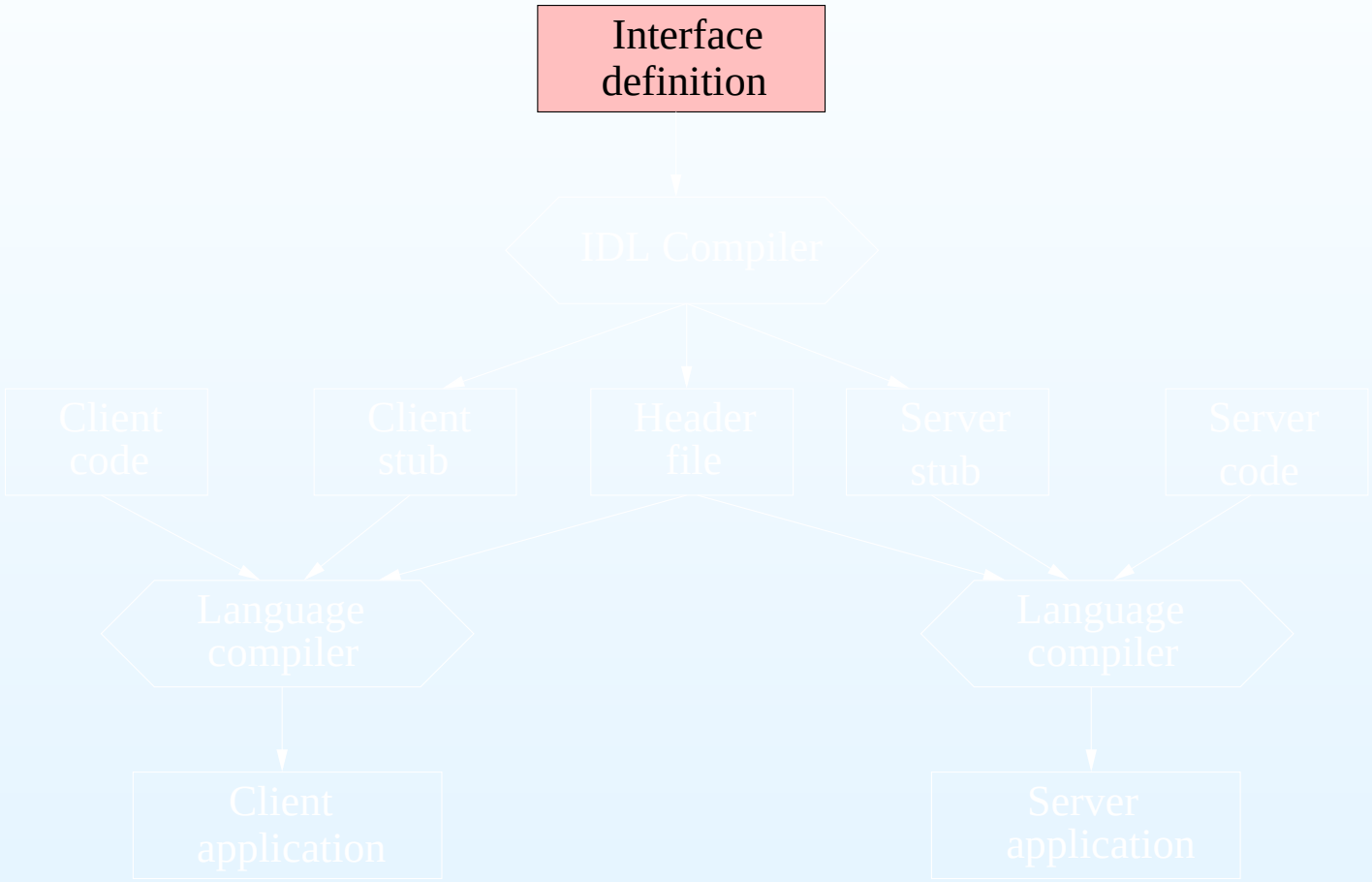
RPC: Interface definition language

- An RPC interface definition specifies those characteristics of the procedures provided by a server that are visible to the server's clients, i.e. interface contains a list of procedure *signatures*
  - name of procedures and the type of their parameters
  - each parameter defined as
    - input
    - output
    - both (i.e. input and output)

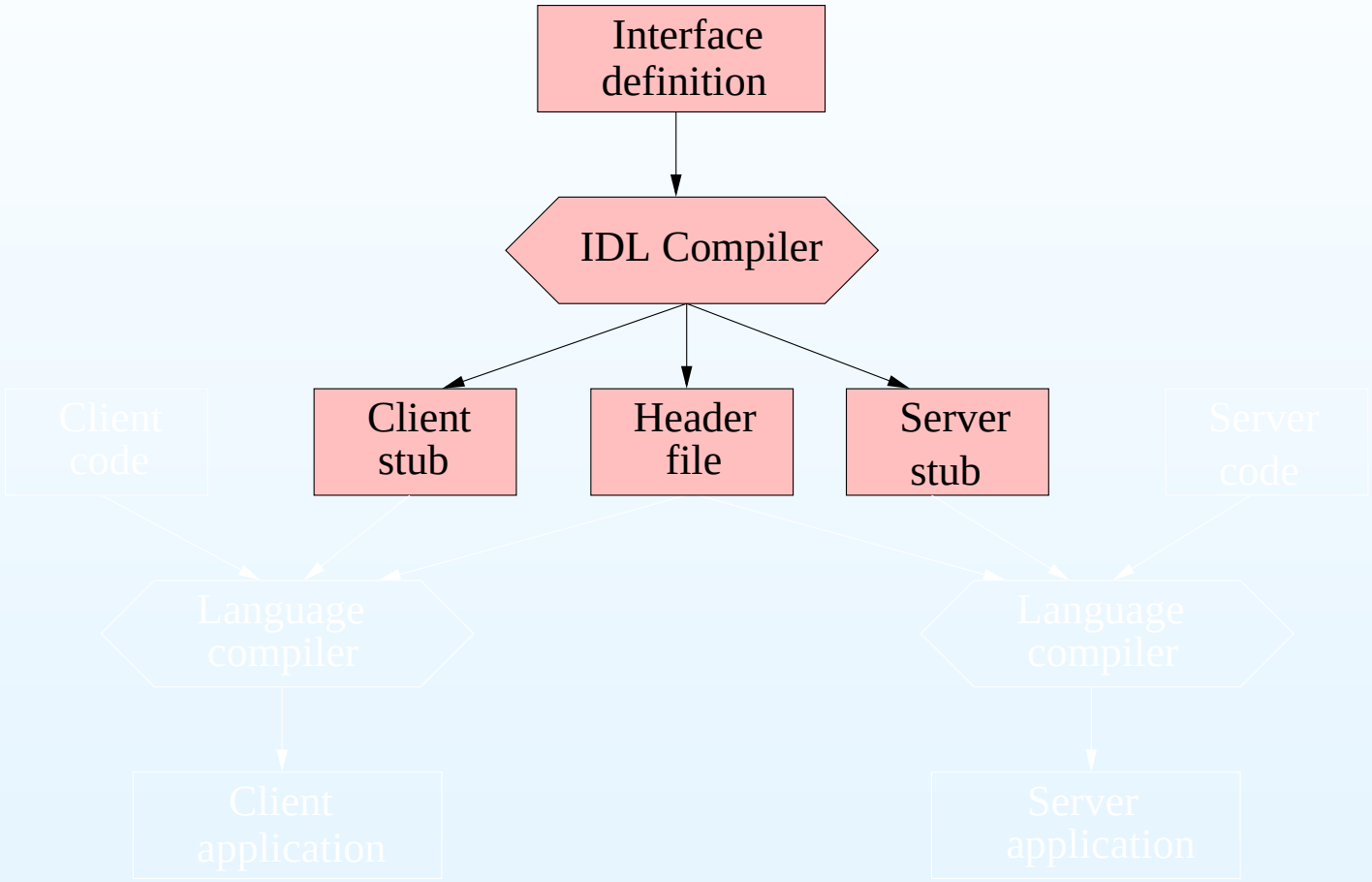
RPC: Interface processing

- Most programming languages have no concept of RPC
- Integrating the RPC mechanism with client and server programs in conventional programming languages
- Interface compiler
  - is designed to produce components that can be combined with client and server programs
  - sources are Interface definition files
- It generates
  - Client stub procedures
  - Server stub procedures and dispatcher
  - Packing and unpacking operations for each stub procedure
  - Header files

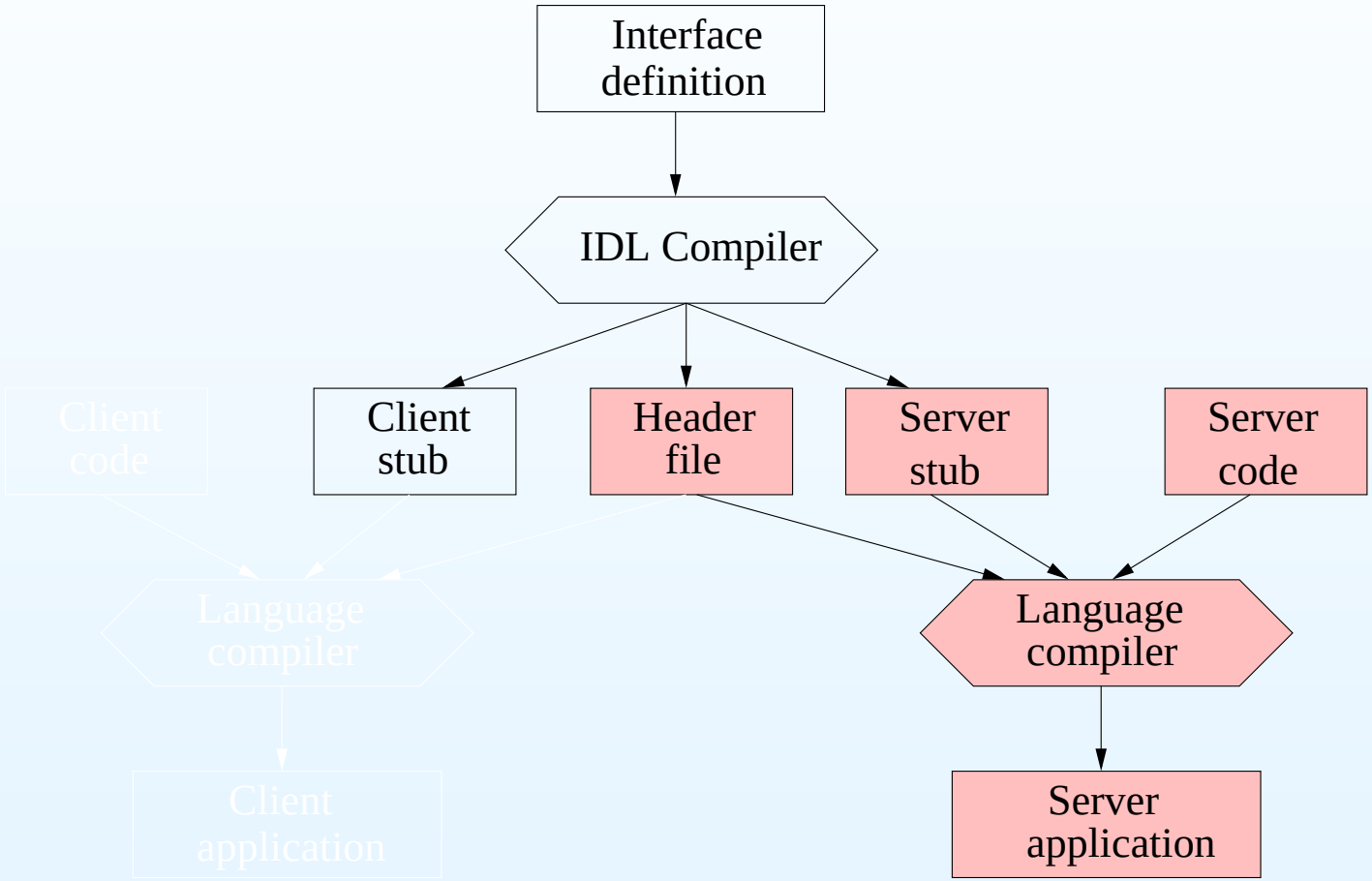
RPC: Compilation example



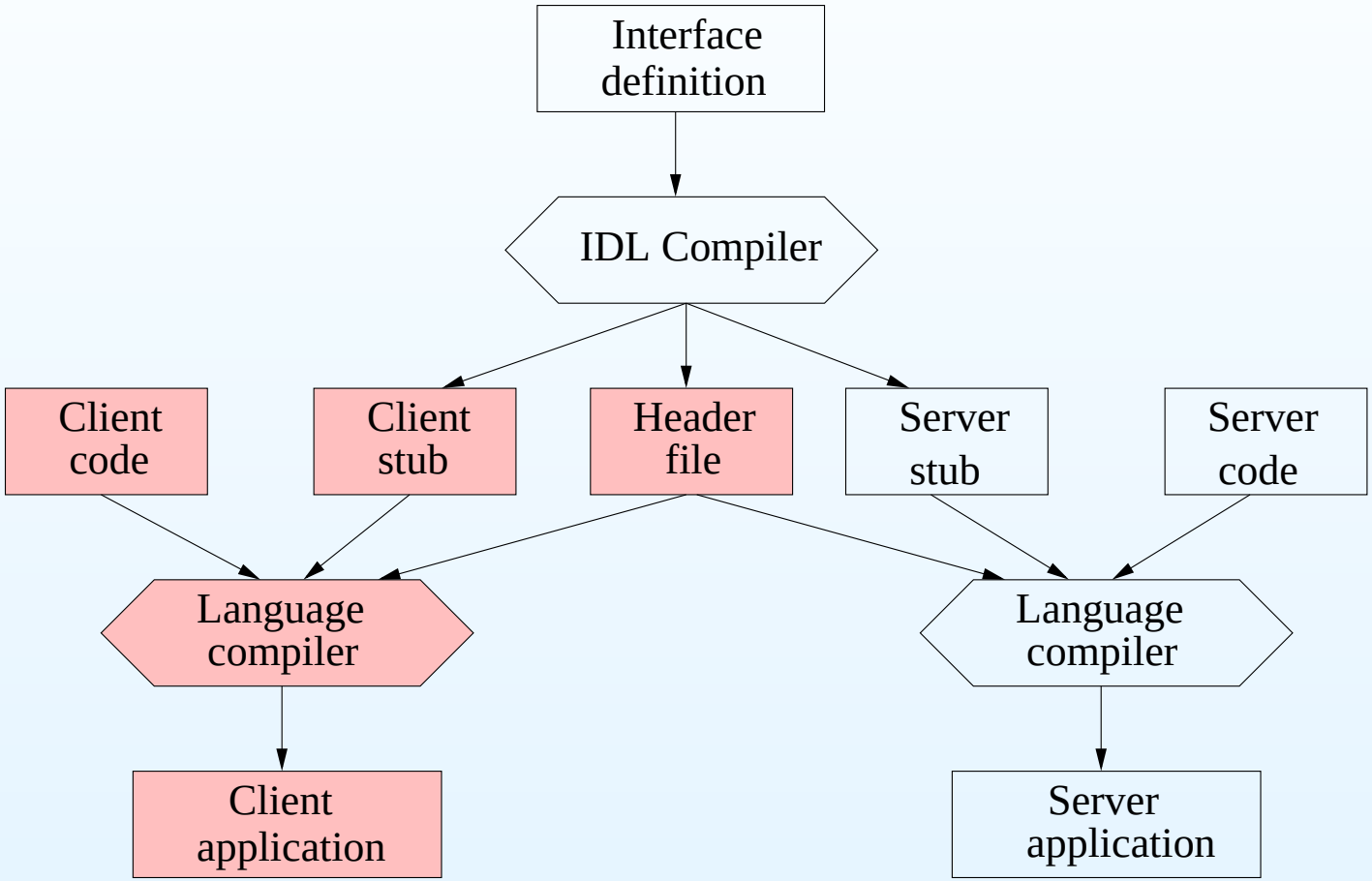
RPC: Compilation example



RPC: Compilation example



RPC: Compilation example



RPC: Binding

- How to know the server location?
- Binding means mapping from a service name to the server address
- Binder - is a service that provides the binding (e.g. *portmapper* in Sun RPC)
  - Has three functions
    1. *Register*
    2. *Unregister*
    3. *LookUp*

Call	Input	Output
Register	name, version, address	
Unregister	name, version, address	
Lookup	name, version	address

The binder interface

RPC: Locating the Binder

- The binder address compiles into all clients
- Supplying the binder address at run time
  - environment variable
- Broadcast messages

Content

- Introduction
- RPC
- **RPC issues**
- Sun RPC
- NFS

RPC issues: Protocols requirements

- Unique specification of a procedure to be called
- Provisions for matching response messages to request messages
- Provisions for authenticating the caller to service and vice-versa

RPC issues: Communication handling

- RPC can use any network protocol to deal the client and server
  - connectionless protocol (e.g. UDP)
  - connection-oriented protocol (e.g. TCP)
  - create new protocol

RPC issues: Security

- Authentication
  - establishes whether service requestors are who they say they are
- Access control
  - establishes the requestor’s rights to perform some operation
- Data protection
  - guarantees the secrecy and integrity of data exchanged between clients and servers

RPC issues: Calls semantics

- Local call
  - *exactly-once*
- Remote call
  - *maybe*
  - *at-least-once*: retry request
  - *at-most-once*: retry request, duplicates filtering and retransmit reply

RPC issues: RPC with Failures

- Client cannot locate the server
- Lost request messages
- Lost reply messages
- Server crashes
- Client crashes

Content

- Introduction
- RPC
- RPC issues
- **Sun RPC**
- NFS

Sun RPC: Briefly about Sun RPC

- Unix RPC system
- Designed for Sun NFS [Sun 1990]
- Calls over either TCP or UDP
- Uses al-least-once call semantics

Sun RPC: XDR and RPC language

- XDR - eXternal Data Representation language
  - Use to describe data formats
  - Transfer data between different computer architectures
  - It is not a programming language
  - XDR data types, some like in C are
- The RPC Language
  - identical to XDR, only added program-definition
  - Parameters passing
    - A single argument and a single result are allowed
    - Multiple arguments and resutl values should be packaged into a single structure

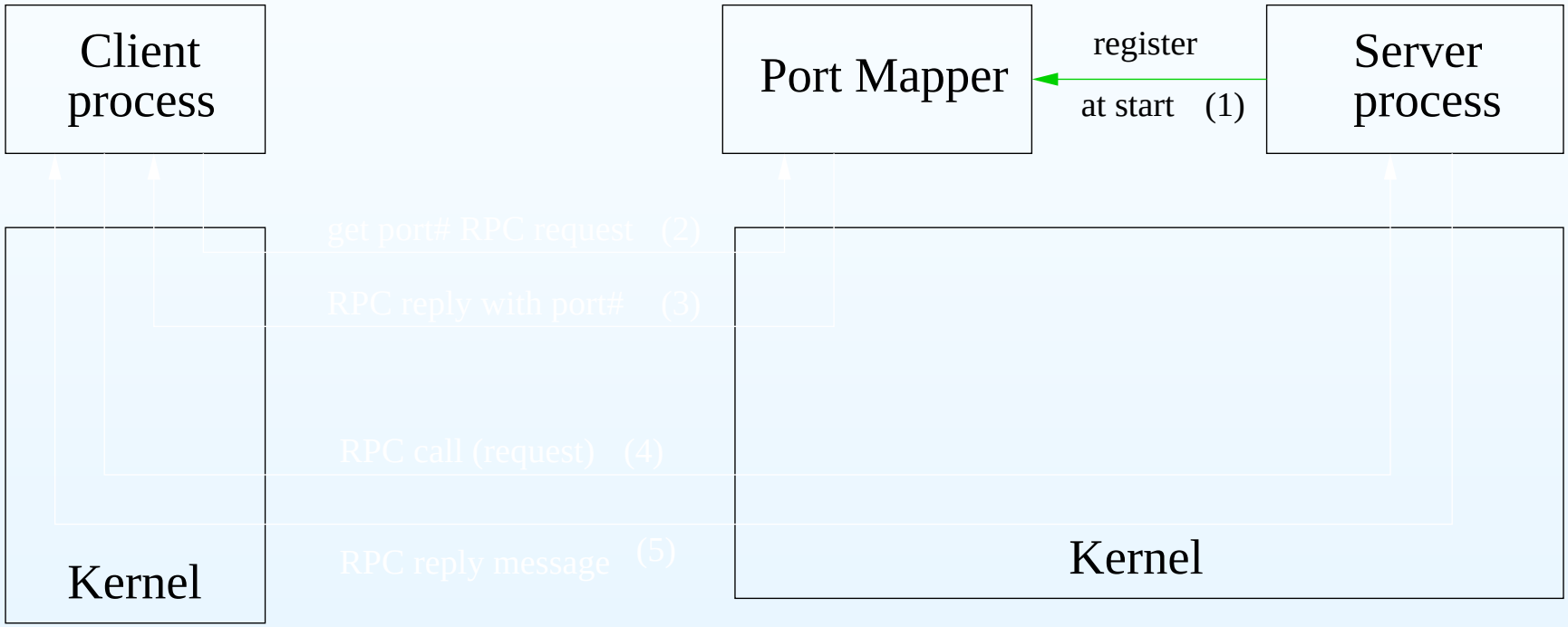
Sun RPC: Interface compiler

- Use the interface compiler *rpcgen* to generate
  - Client stub procedures
  - Server *main* procedure, dispatcher and server stub procedures
  - XDR packing and unpacking procedures
  - Header files

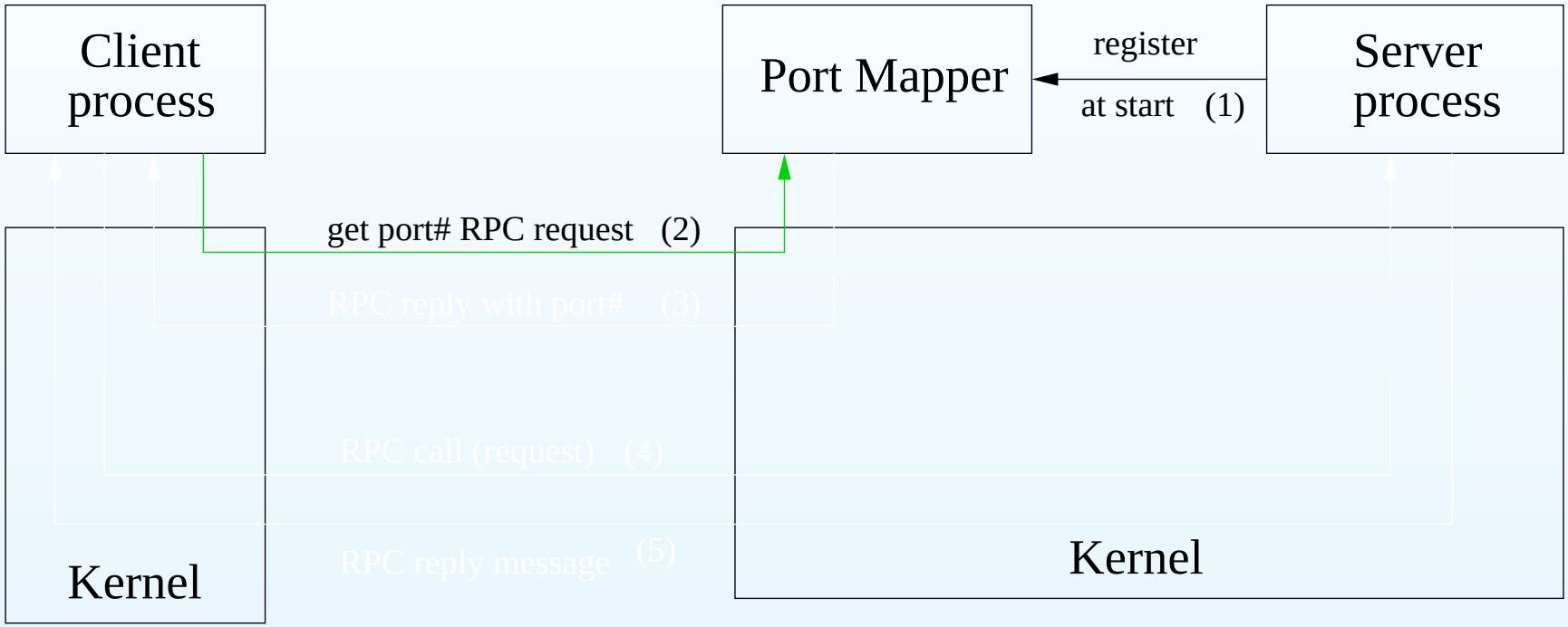
Sun RPC: Binder - Portmapper

- Sun RPC doesn’t have a network-wide binding service
  - local binding service - portmapper
- Runs on every computer
- Has the fixed port number (111)
- Each instance of the portmapper records
  - The port in use by each service running locally
    - port the same for different versions of program
    - not the same for the same versions of programs with different protocols
    - not the same for other cases

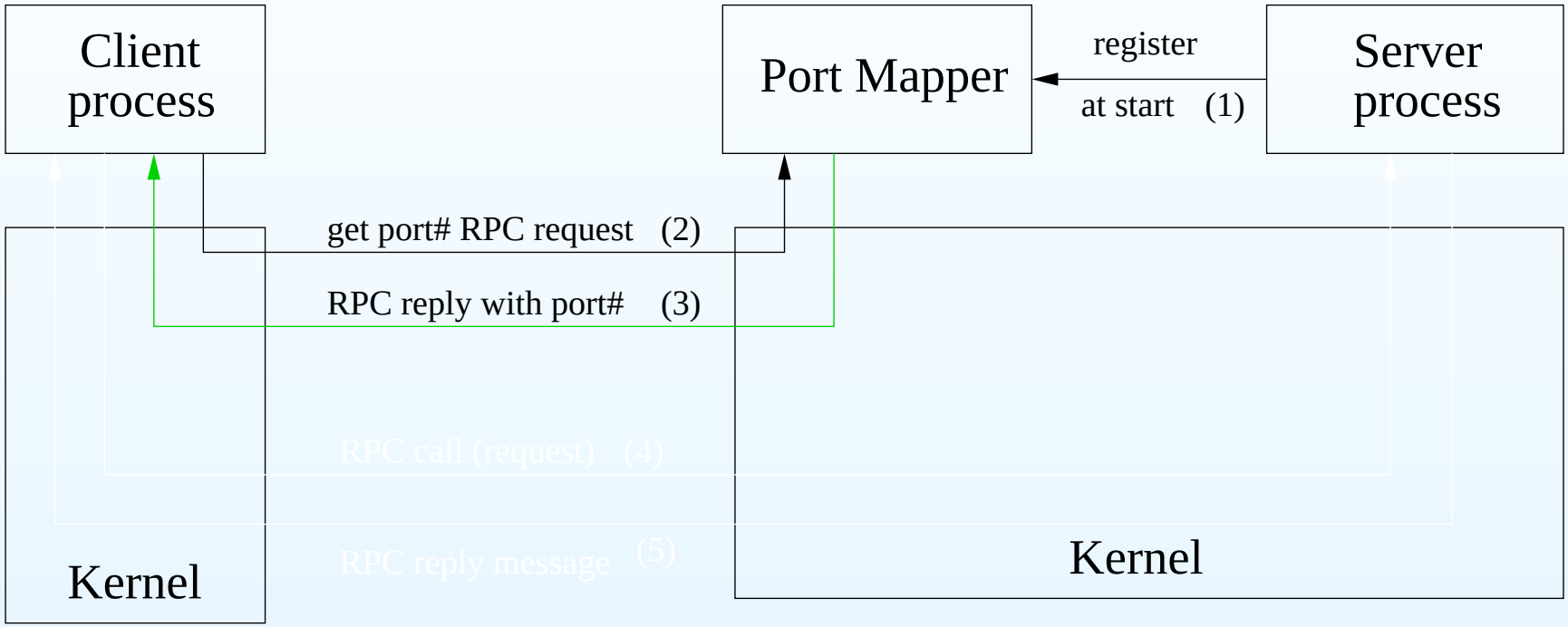
Sun RPC: RPC binding example



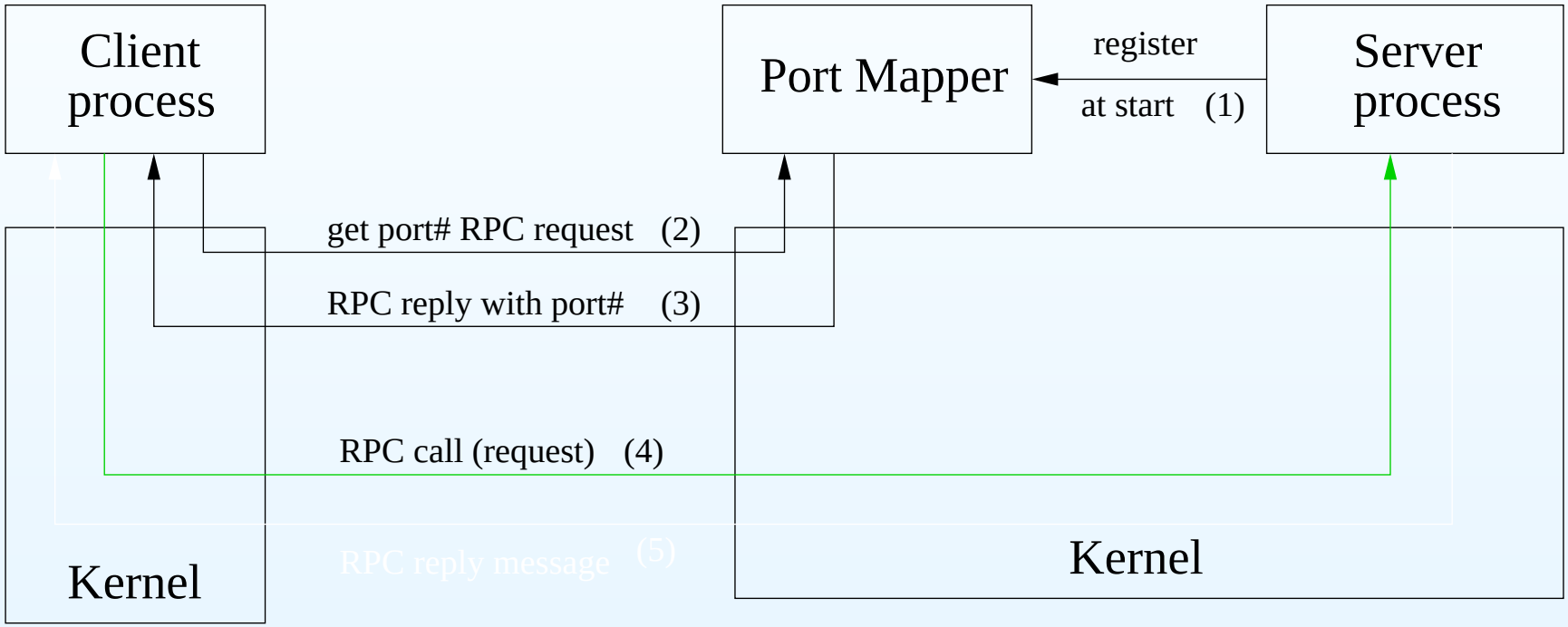
Sun RPC: RPC binding example



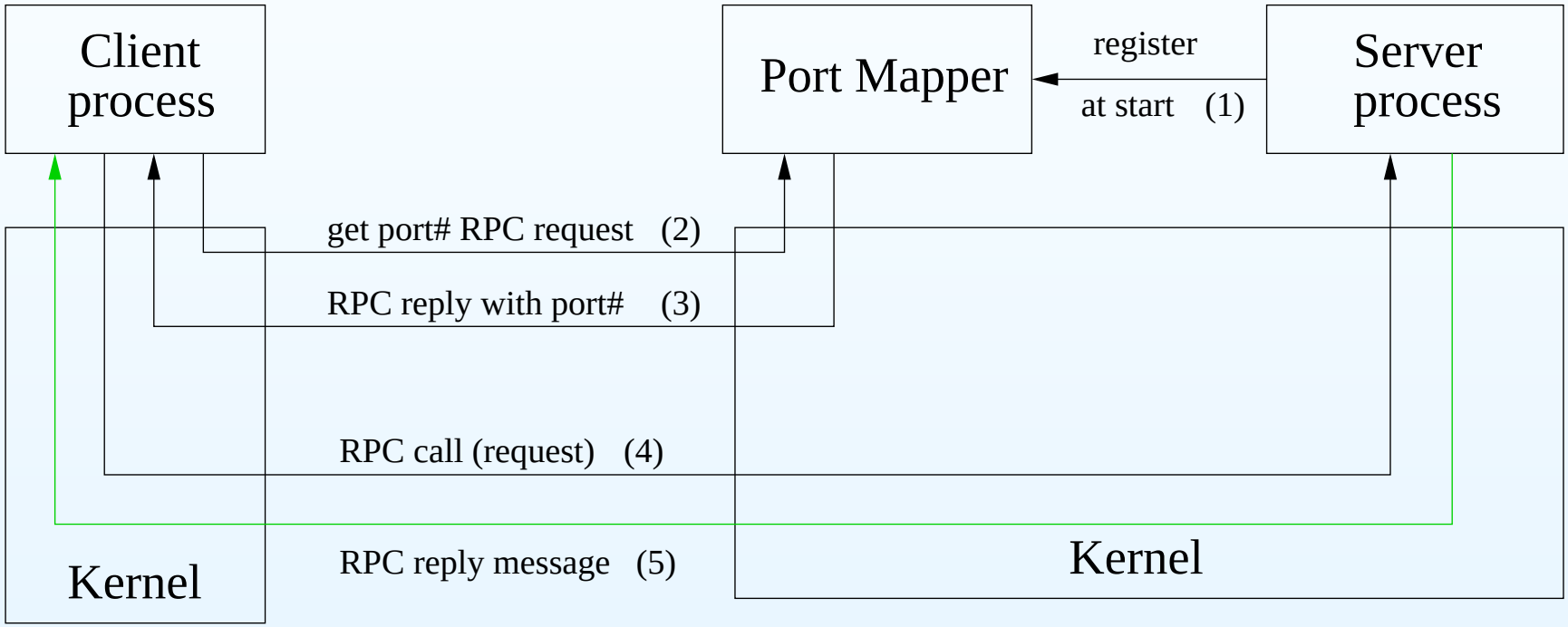
Sun RPC: RPC binding example



Sun RPC: RPC binding example



Sun RPC: RPC binding example



Content

- Introduction
- RPC
- RPC issues
- Sun RPC
- NFS

NFS: Network File System

- Designed and implemented by Sun Microsystems (Walash et al,1985; Sandberg et al,1985)
- The most commercially successful and widely available
- Reasons for success
  - The NFS protocol is public domain [Sun 1989]
  - Sun sells that implementation (under license)

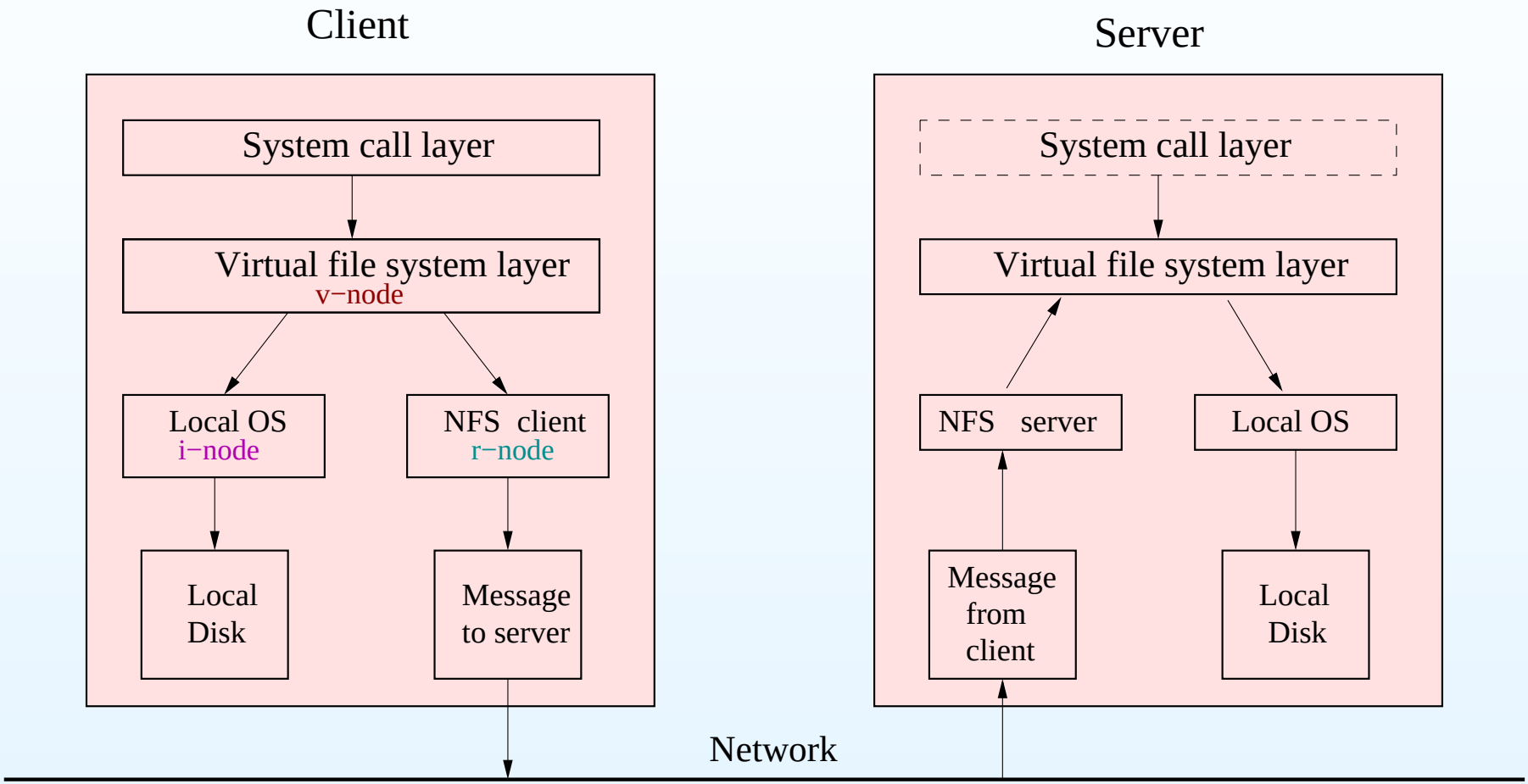
NFS: The NFS Architecture

- NFS server exports directories for access by remote clients
  - List of directories to export (e.g. UNIX "/etc/exports")
- Clients access exported directories by mounting them
- A machine may be both client and server

NFS: The NFS protocols

- Handles mounting
  - client sends request to server with path name of directory to be mounted in the clients directory
  - if the path name is legal and the directory is exported by the server, the server returns a file handle
  - Automounting also supported
- File access protocol
  - Files and directories manipulation

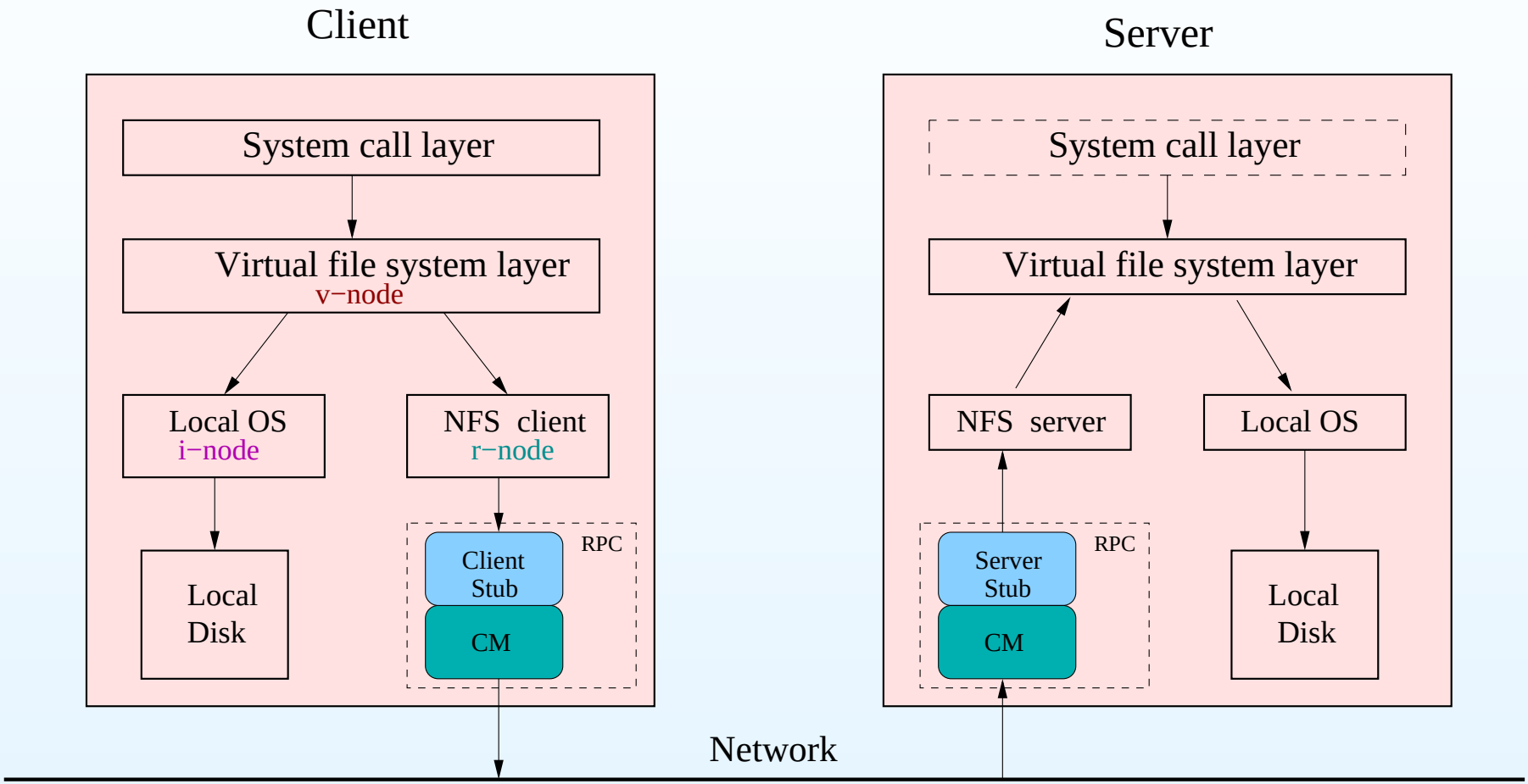
NFS: NFS layers structure



NFS: NFS layers

- System Call
  - Handles calls like READ, LOOKUP, . . .
- Virtual File System
  - Keeps a table of v-nodes (virtual i-node) for each open file
    - v-node points to either an i-node (internal) or r-node (remote)
    - i-node kept by Local OS
    - r-node kept by NFS client
    - from v-node see the location of the file (i.e. local or remote)
  - No table entries are made on the server side

NFS: NFS layers structure



Summary

- the RPC idea: a client running on one machine calls a procedure running on another machine
- the RPC transparency
- RPC problems
  - calls semantics
  - server has to be located
  - pointers and complex data structures are hard to pass
  - global variables are diffuclt
  - network failures
  - client and server can fail independently of another one
  - security
- Stub components and IDL are good approach to make an RPC mechanism like the LPC
- considered the part of an RPC implementation on the Sun RPC
- and also the application of an RPC in NFS

The End

Thanks

for Attention