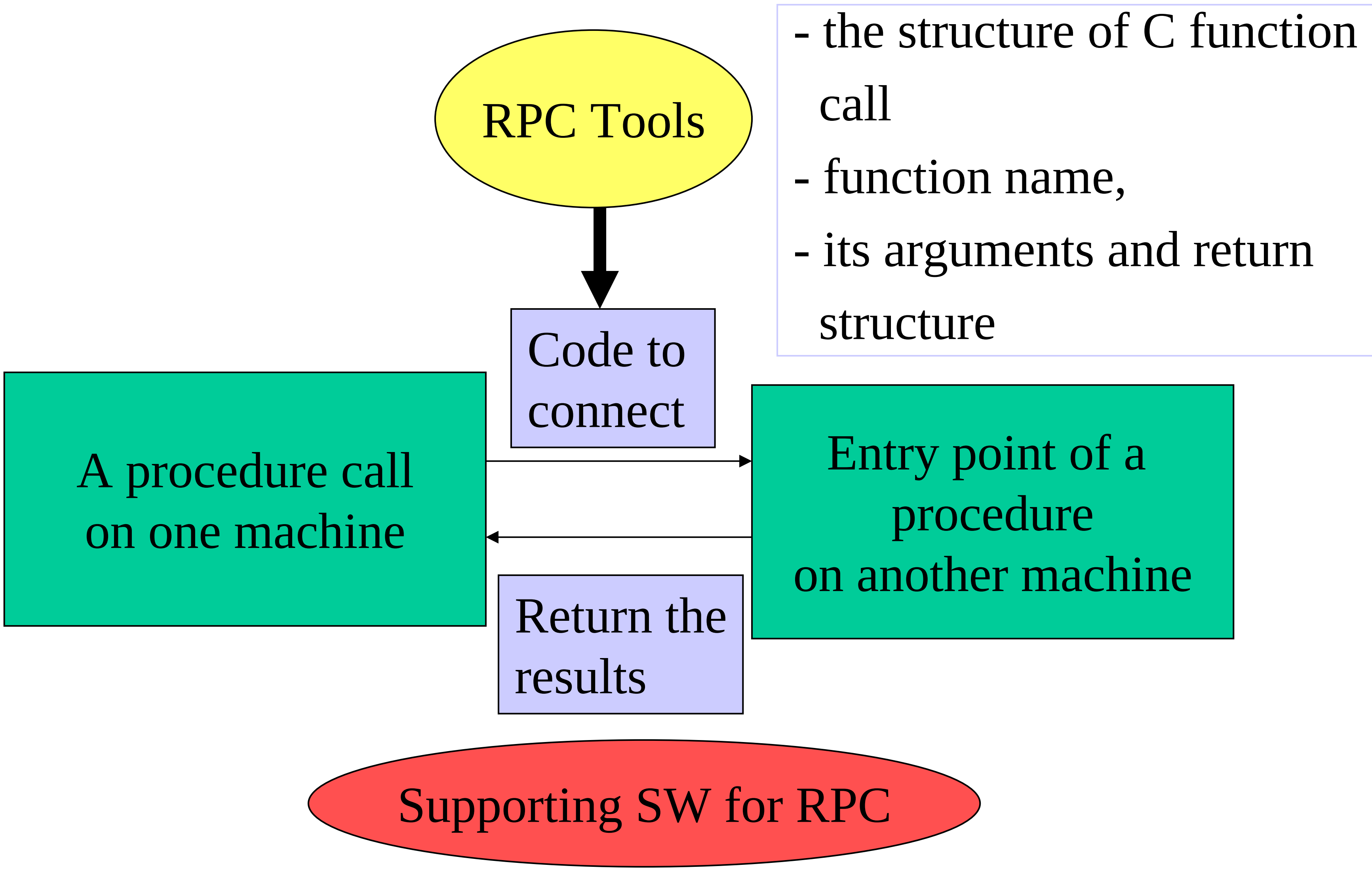
Remote Procedure Call

Introduction of RPC

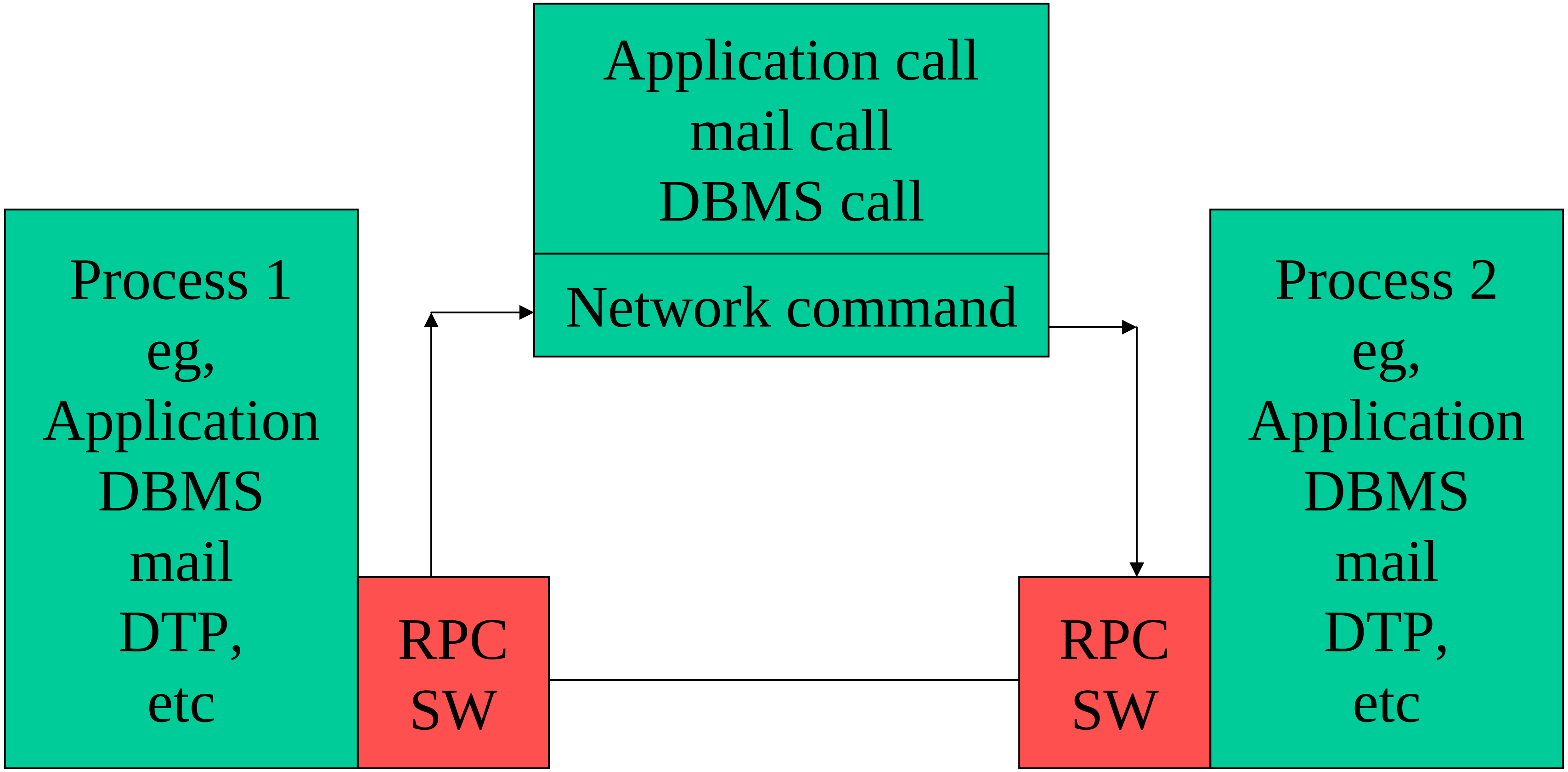
- Support the communication layer
  - provide an easy-to-use form of IPC and a reliable technology
    - high performance connectivity
- Main disadvantage
  - lack the flexibility and functionality of MOM
    - no-wait capability
    - broadcasting
    - queuing and deferred synchronous processing
    - asynchronous processing
- Advanced RPCs
  - extend these functionality of
  - offer the management services, greater network transparency, and traslation services

What are RPCs (1)



What are RPCs (2)

RPC: Inter-process communication between application processes

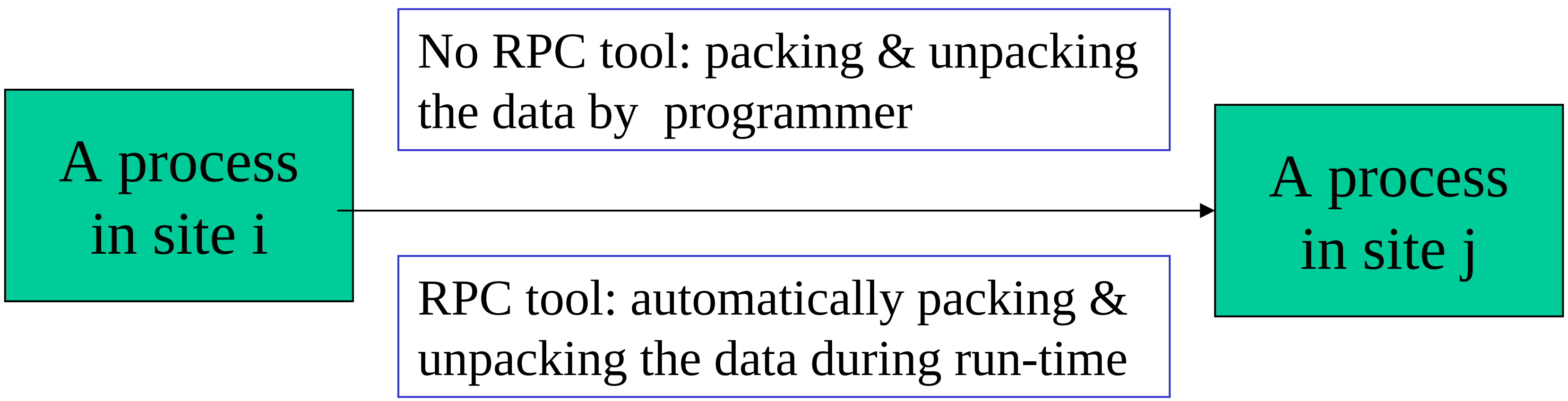


Background of RPCs

- First version
    - Xerox's Palo Alto Research Center in the late 1970s
    - named 'Courier'
- In the early 1980s
    - Sun Microsystemsadopted RPCs for its network
    - created the XDR (eXternal Data Representation) RPC format
    - used it to develop NFS
    - developed 'rpcgen', a RPC compiler
- ONC
    - developed ONC rpcgen
    - bundled with Unix System
- SUN
    - developed TI-RPC, bridged the Unix aand PC systems
- DCE
    - developed new RPC compiler, named IDL(Interface Definition Lang)

How do RPCs work?

- RPC: distributed processing
  - how to pass data between difference processes
  - data must actually be packaged and moved across the network

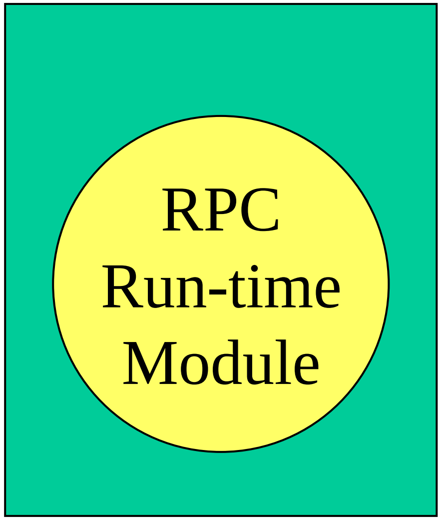


- No RPC tool
  - the programmer has to ensure the application-specific data structures are converted into a sending form
  - the programmer has to program any translation between data formats
  - the programmer has to code the network transport protocols

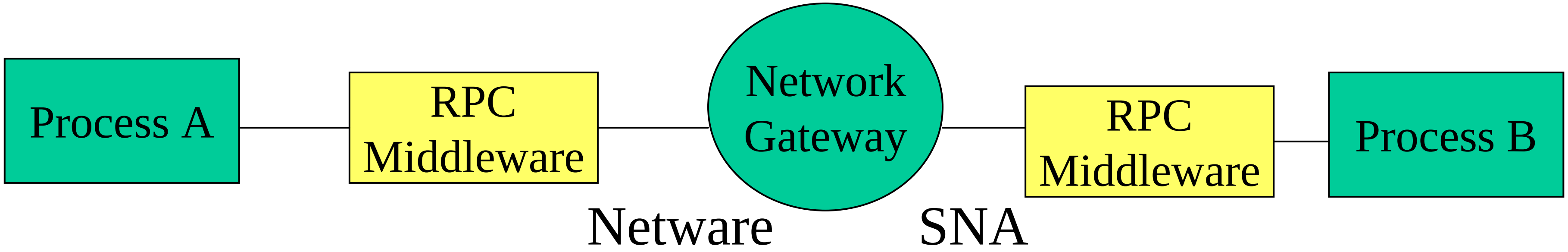
RPC run-time environment (1)

- RPC run-time modules in Calling
  - reduce the data structure to be passed
  - convert the individual data elements to a format understood by the called process
  - make calls to the network interface to co-ordinate the sending of the packaged data to the called process
  - support multiple network protocol

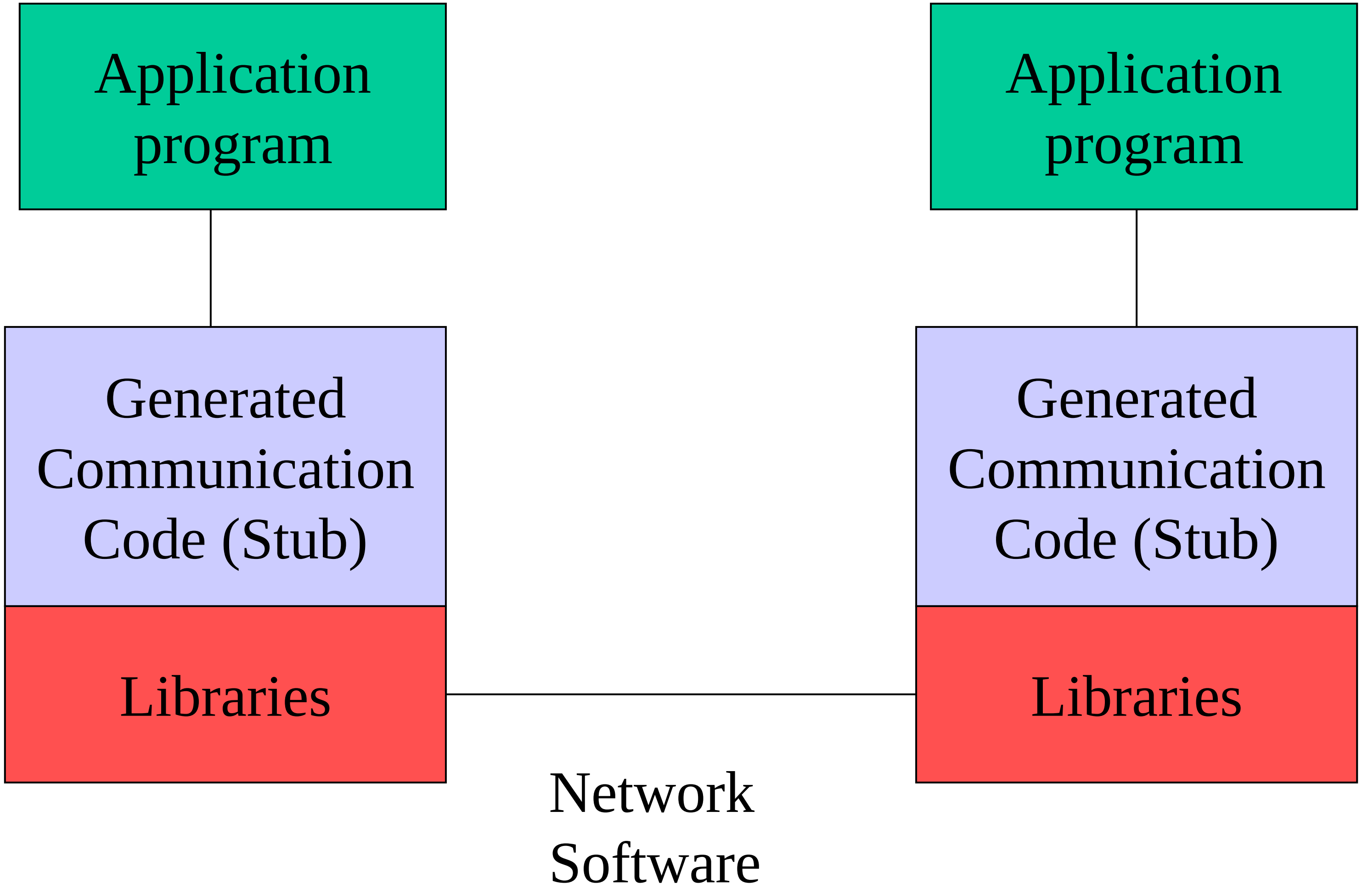
Calling process/
Called process



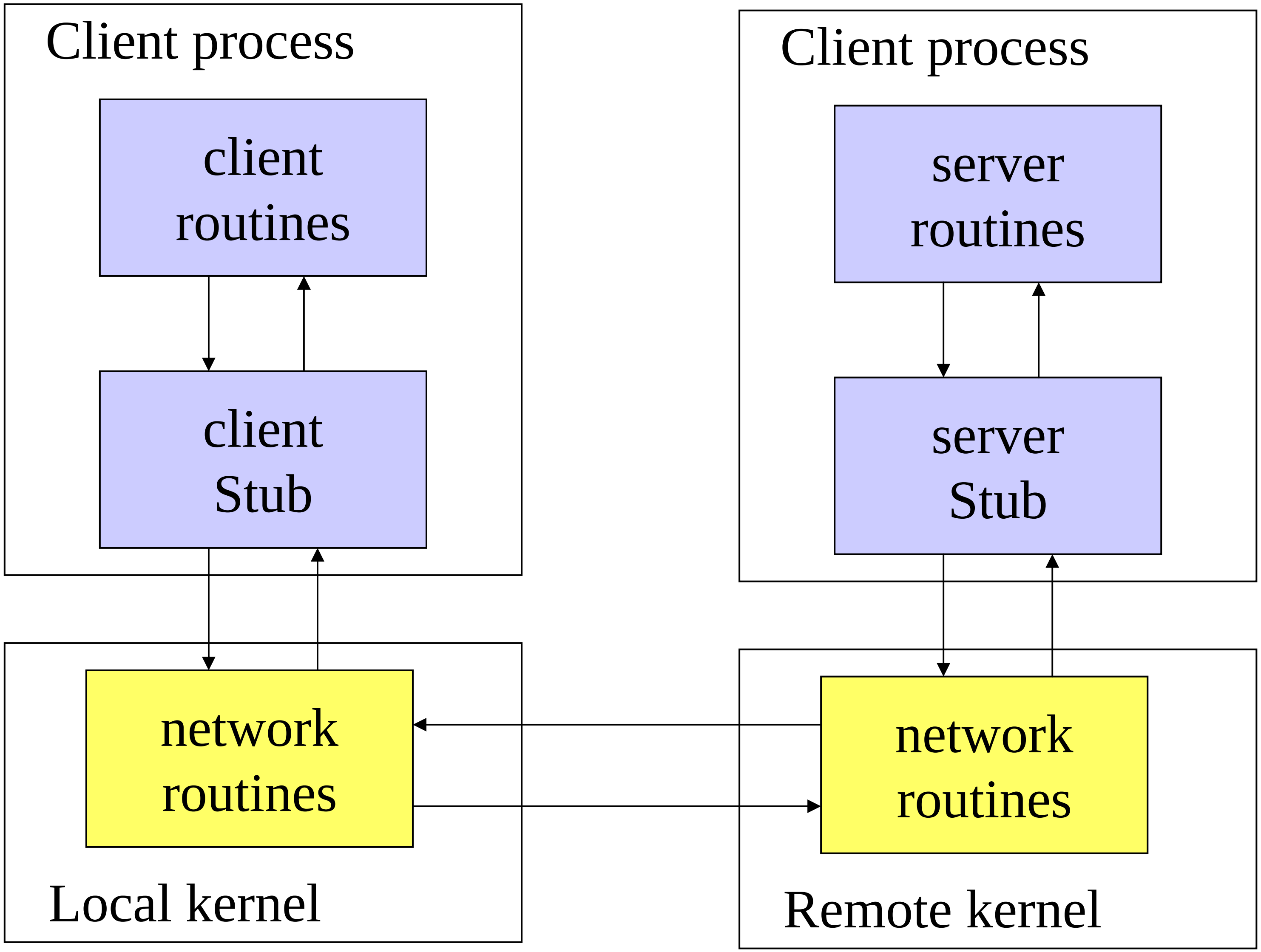
- Multi-network environments
  - need to have a network gateway to transfer RPCs



RPC libraries and stubs



Remote Procedure Call model



RPC run-time environment (2)

- RPC run-time modules in Called
  - receives the inbound request containing the packaged/converted data
  - unpacks the received buffer, re-creating the data structures
  - converts the individual data into the local format
  - calls the required process
  - coverts and packages the results for transmission to caller

- Error-handling service of RPC SW
  - terminate the session
  - close all files
  - clean up the tables
  - notify the application process
  - retry the application, if defined
  - invisible each process, called and caller
  - are achieved using network and run-time libraries

RPC run-time environment (3)

- The Library functions
  - packing and unpacking data types
    - converting HW-specific data before or after transmission
  - handling network-communication errors and recovering
  - co-ordinating the request/reply messages between caller and called
  - management of server scheduling

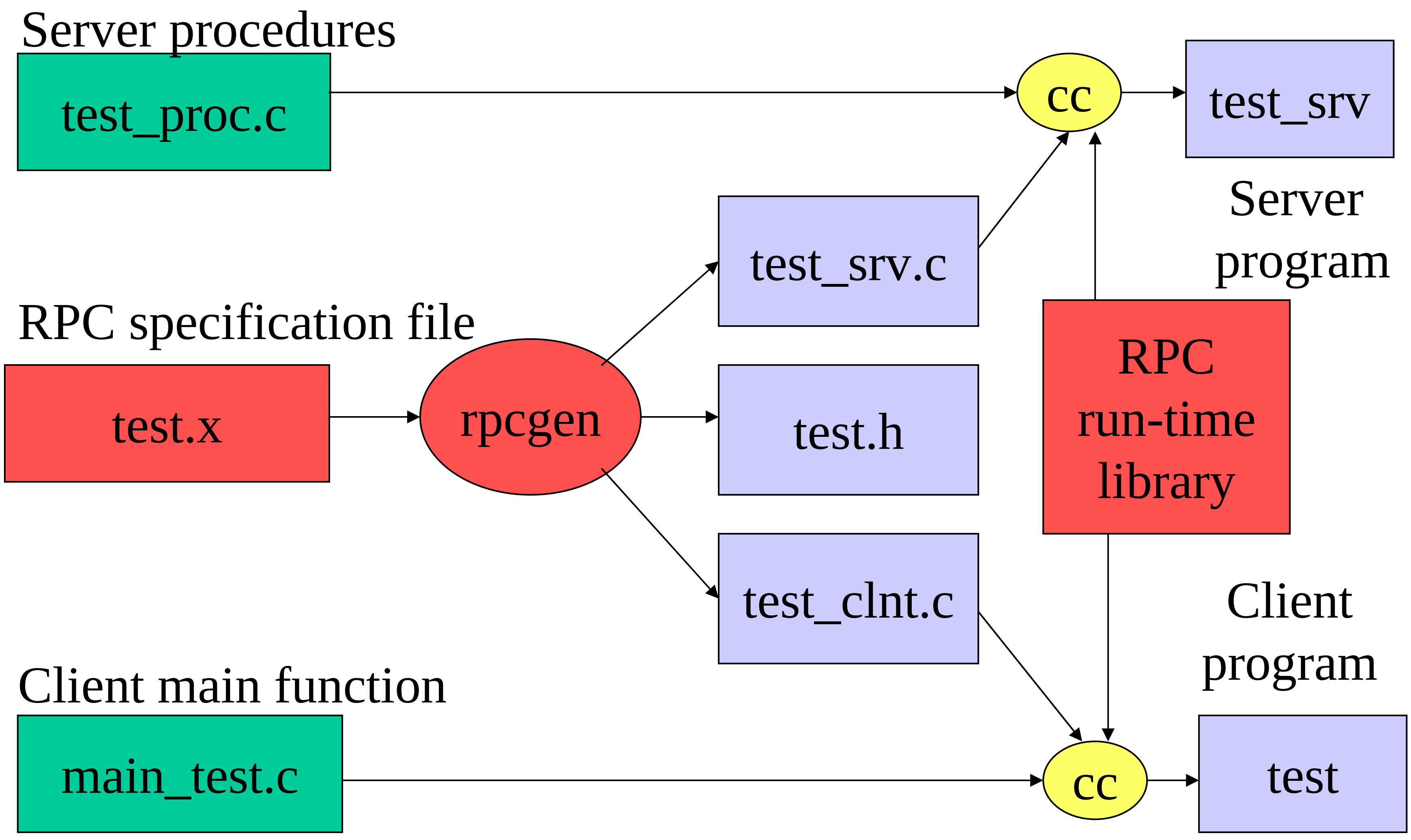
- The stub code
  - packing and unpacking data structures
  - allocating and freeing memory
  - packing and unpacking data types
    - an intermediate common protocol has been used
    - ASN (Abstract Syntax Notation) Basic Encoding Rules

RPC development environment

- A toolkit
  - enables the developer to specify the interactions between the processes
  - support a high-level scripting language
    - generate a standard language such as C

- The specification in a description file
  - the parameters to the remote calls and how they are used
  - user-defined data structures passed as parameters
  - names of the processes
  - mechanisms for assigning the network name used by each server process
  - communication options

Example of a Sun RPC program



RPC Library Routines (1)

- RPC client
  - callrpc call remote procedure, given [prognum, versnum, procnum]
  - clnt_broacast() broadcast remote procedure call everywhere
  - clntraw_create() create toy RPC client for simulation
  - clnttcp_create() create RPC client using TCP transport
  - clntudp_create() create RPC client using UDP transport
  - rpc_createerr global variable indicating reason why client creation failed
  - clnt_pcreateerror() print message to stderr about why client handle creation failed
  - clnt_call call remote procedure associated with client handle
  - clnt_geterr() copy error information from client handle to error structure
  - clnt_perrno() print message to stderr corresponing to condition given
  - clnt_freeres() free data allocated by RPC/XDR system when decoding results
  - clnt_destroy() destroy client's RPC handle

RPC Library Routines (2)

- RPC Server
  - regiserrpc() register procedure with RPC service pakage
  - src_run() wait for RPC requests to arrive and call appropriate service
  - svcraw_create() creates a toy RPC service transport for testing
  - svctcp_create() creates an RPC service based on TCP transport
  - svcudp_create() creates an RPC service based on UDP transport
  - svc_fds global variable with RPC service file descriptor mask
  - svc_register() associates prognum and versnum with service dispatch procedure
  - svc_getreq() returns when all associated sockets have been serviced
  - svc_getargs() decodes the arguments of an RPC request
  - svc_getcaller() get the network address of the caller of a procedure
  - svc_sendreply() send back results of a remote procedure call
  - svc_freeards() free data allocated by RPC/XDR system when decoding arguemnts

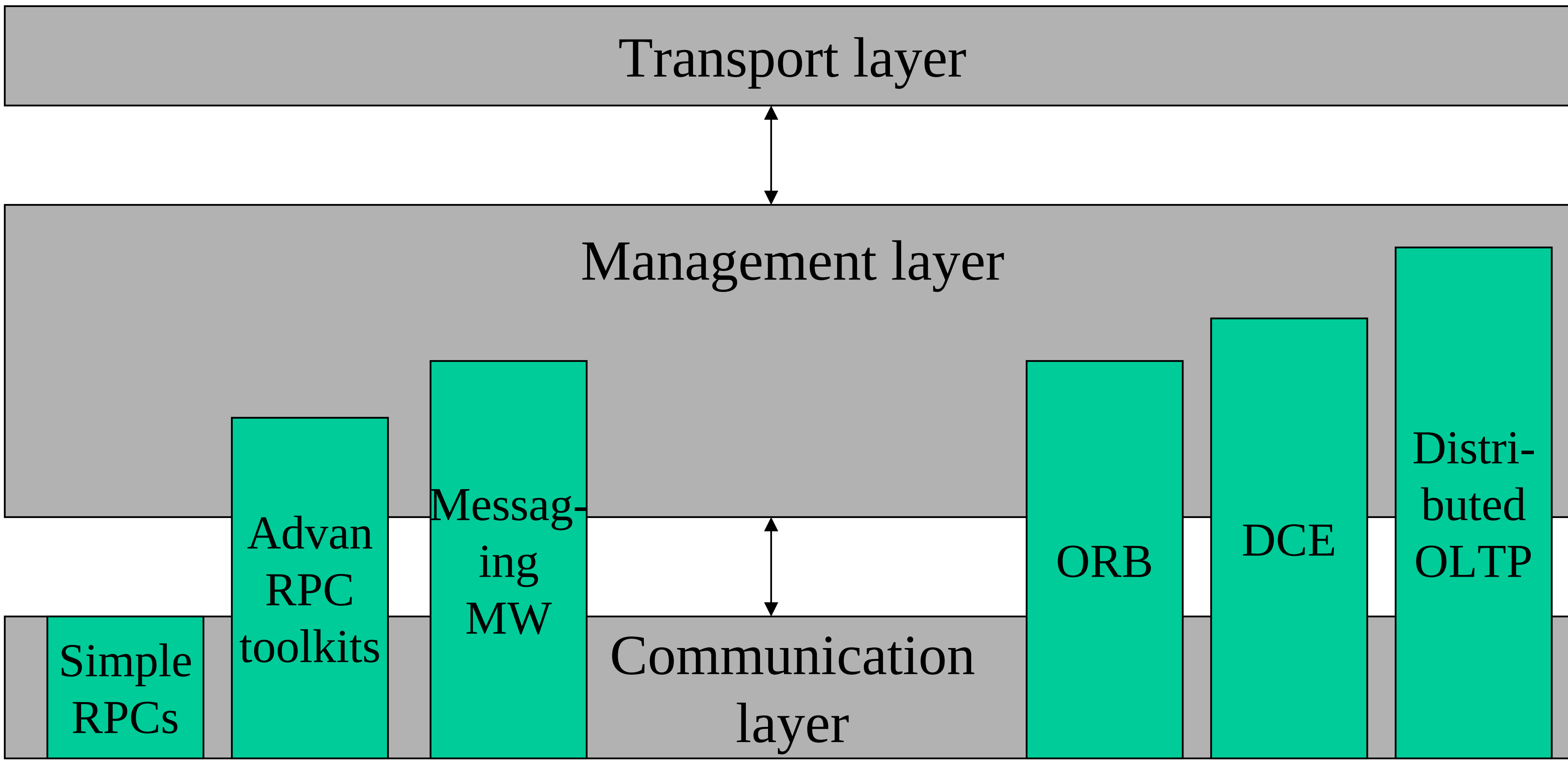
RPC Library Routines (3)

- RPC Server (cont.)
  - svcerr_auth()	called when refusing service because of authentication error
  - svcerr_decode()	called when service cannot decode its parameters
  - svcerr_noproc()	called when service hasn't implemented the desired procedure
  - svcerr_noprog()	called when program is not registered with RPC package
  - svcerr_progvers()	called when version is not registered with RPC package
  - svcerr_systemerr()	called when service detects system error
  - svcerr_weakauth()	called when refusing service because of insufficient authentication
  - ssvc_unregister()	remove mapping of [prognum, vernum] to dispathc routines
  - svc_destroy()	destroy RPC service transport handle

RPC Standards

- The main standard for RPCs
  - embedded within the Open Software Foundation’s DCE standard
  - based for Windows RPC
  - based for CORBA
  - based for RMI

Converage of RPC products



- Advanced RPC: no-wait capability, broadcating, deferred synchronous
- Messaging MW: queueing

Simple VS Advanced RPC toolkits

- Parameter passing
- data types supported
- specification support
- large message handling
- multiple transport protocol
- memory management
- broadcasting
- no wait
- server search capability
- multi-threading
- multi-tasking
- multiple binds
- compression
- queuing, routing and prioritization

Advantages and Disadvantages of RPCs

- Advantages
  - Simple in concept
  - Useful for converting legacy applications
  - Good performance
  - Error checking is easier
  - Well tried and tested
  - Advanced toolkits save effort

- Disadvantages
  - One-to-one communication
  - Resilience
  - Complexity
  - Difficulty of change

When to use RPCs

- Require high performance
- Are tightly integrated as opposed to loosely coupled
- Are to be built by people familiar with C and procedural languages rather than OOPL
- Are likely to be constrained by memory or are to be run on low-end platforms
- Are not complex in design and do not require multiple many-to-many calls between processes
- Do not require asynchronous communication
- Do not require resilient services such as store-and-forward, but can be written to use exception routines to handle network errors
- Are unlikely to be subject to frequent changes of process residence once implemented

Research Directions

- Management services
  - queuing, security, guaranteed delivery,
  - advanced synchronization services, advanced threads services,
  - load balancing, rollback and recovery services,
  - performance monitoring, monitoring and logging for debugging purposes
  - advanced time handling services

- Network transparency
  - directory services: increase network and platform independence
  - establish the best routes for packets at run-time
  - add the network client and server names of processes at run-time
  - remove the need for this

- Translation capability
  - support database access and translation of DB commands from a standard