INTERPROCESS PRIMITIVES: REMOTE PROCEDURE CALL (=RPC)

REMOTE PROCEDURE CALL

- language-level abstraction of the request-reply communication based on message passing and procedure call where client sends a request to a server, and then blocks until the server sends a response,

- type-checked mechanism that permits a language-level call on one computer to be automatically turned into a corresponding language-level call on another computer,

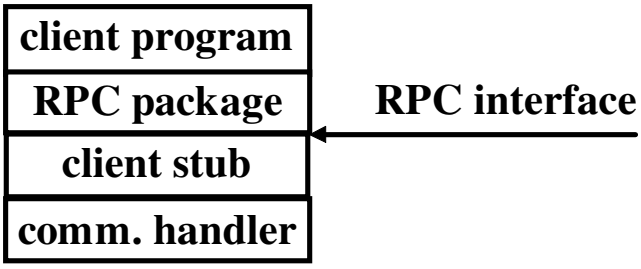
- its interface definition specifies those characteristics (names of procedures and types of their parameters) of the procedures provided by a server that are visible to the server's clients,

- library of conventional procedures that presents a convenient procedural interface for application programmers, because the remote procedure calls are embedded within the RPC package, and low-level system calls, data conversion and network communication are hidden from applications,

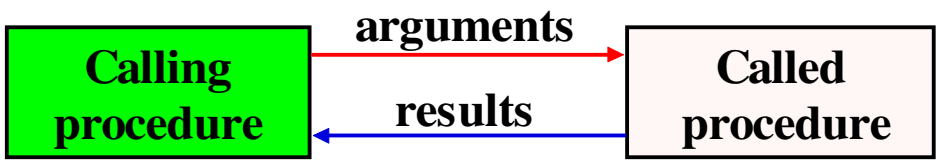
- allows distributed programs to be written in the same style as conventional programs for centralised systems, because RPC takes the form of a normal procedure call, with its associated input and output parameters,

- caller (client) and called procedure (server) are separate processes usually running on separate computers,

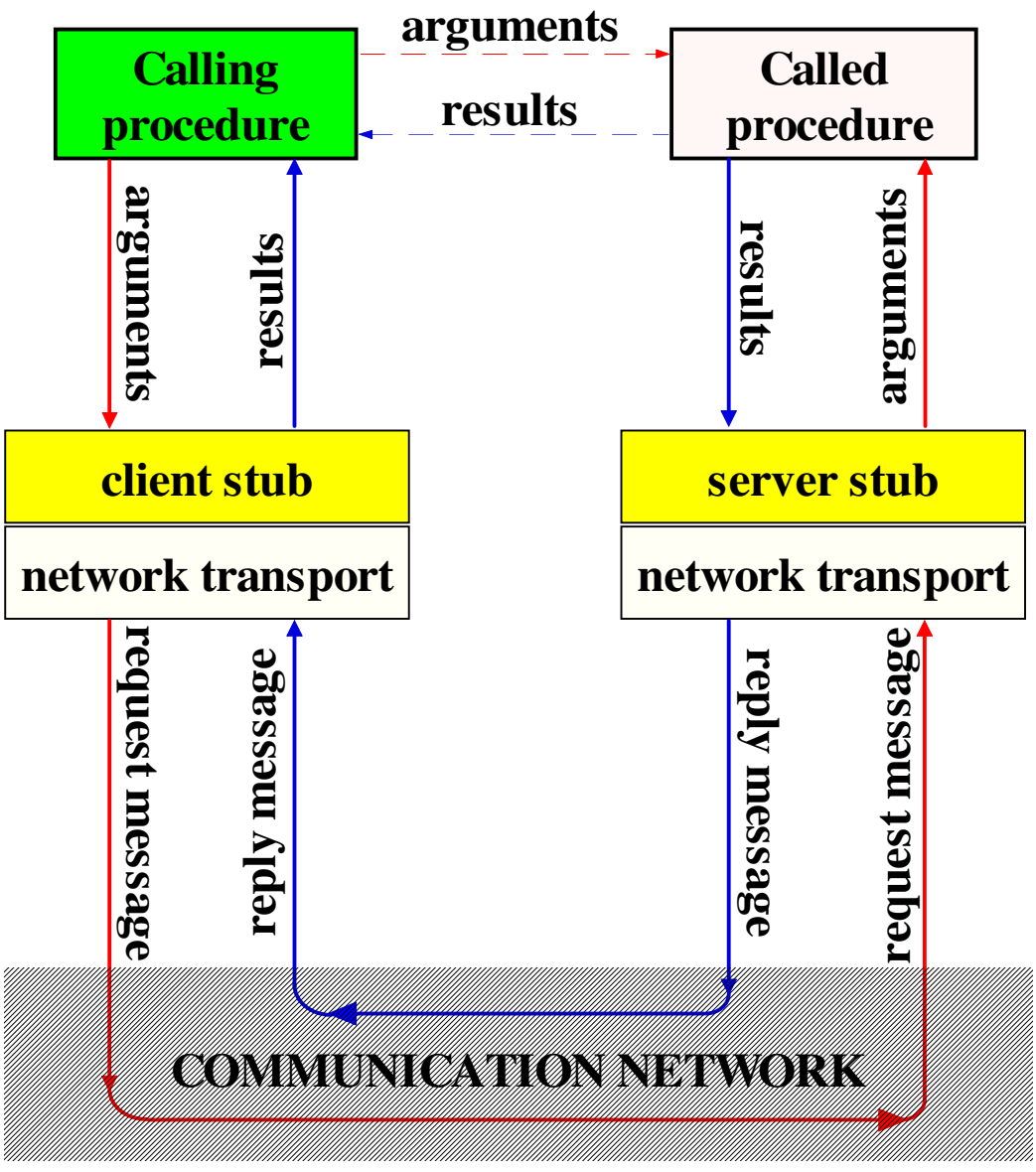
- processes in message passing systems act as partners whereas processes in remote procedure call systems are in master-slave relationship,



levels of the remote procedure call



LOCAL PROCEDURE CALL



REMOTE PROCEDURE CALL

FEATURES and PROPERTIES of RPC

goals of RPC:

- **to make remote procedure call transparent to users just like local procedure call,**

- **to perform a service by a module with an interface that exports a set of procedure appropriate for operating on some data abstraction or resource,**

types of RPC:

- **RPC mechanism is integrated with a particular programming language that includes a notation for defining interfaces,**

- **a special-purpose interface definition language is used for describing the interfaces between clients and servers,**

Features of RPC

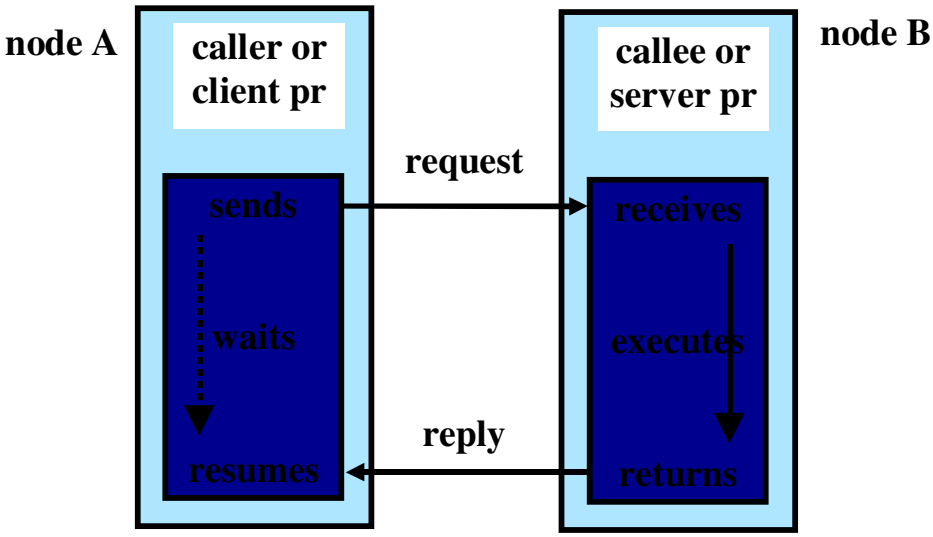
- in message passing all required values must be explicitly assigned to the message before transmission,

- in RPC input parameters are passed to the server by sending the values of arguments in the request message,

- input parameters of RPC are equivalent to parameters passed by local procedure calls,

- in RPC output parameters are returned to the client in the reply message and they are used to replace the values of the corresponding variables in the calling environment,

- the remote procedure is executed in a different execution environment than its caller thus, it cannot access variables in the calling environment, such as the global variables declared by the caller,



- the caller (client process) sends an RPC in the form of the call (request) message to the remote process - callee - (server process),

- the callee executes the procedure and sends back a response message, but the remote procedure has no access to data and variables of the caller's environment

Properties of the RPC

1. uniform call semantics

- **RPC must maintain the same semantics as that used for local procedure calls,**

3. full parameter functionality

- **all basic data types should be allowed as parameters to RPC**

3. type checking

- **the same static type checking is applied to both local and remote procedure calls**

4. type translation

5. atomic transactions

- **one process announces that it wants to begin a transaction with one or more other processes,**

- **they can negotiate create and delete objects, and perform operations,**

- **the caller announces that it wants all others to commit themselves to the action performed so far,**

- **if all of them agree, the results are made permanent, if one or more processes refuse, the situation reverts to the state it had before the transaction began,**

6. concurrency control and exception handling

- **the programming language that supports RPC should provide these services,**

7. distributed binding

- **the programming language that supports RPC must have some means of compiling, binding and loading distributed programs onto the network**

8. orphan computations

- **provide reliability of an RPC i.e. recovery from RPC that fails, there are two basic approaches:**

  a. extermination - to find and abort orphan computations that result from a crash

  b. expiration - to determine when a computation has existed beyond its expected lifetime (it is analogous to time-out)

9. respect for autonomy

10. transparency

- **the client calls remote procedures in the normal way for the language in use without knowing anything about the remote procedure call,**

11. remote debugging

12. good performance

RPC USER PACKAGE

- the remote procedure call is a refinement of the send, and receive primitives,

- to use services by application programs an RPC user package is applied to,

- the user package is a library of conventional procedures that presents a convenient procedural interface for use by application programs,

- tasks of the user package:

- **interface processing (marshalling and unmarshalling)**
- **binding**
- **communication between clients and servers**
- **exception and fault recovery**

Interface definition language

- services provided by a server process are specified by a server interface using interface definition language,

- the RPC interface specifies those characteristics of the procedures provided by a server that are visible to the server's clients for example: names of procedures and types of their parameters,

- the interface contains a list of procedure signatures, i.e. their names, types of their input and output arguments,

Caller and callee

caller

- **the caller (or client) would like to access to a service or services provided by one or several servers located on another computers,**

callee

- **there are two types of callee processes:**

  a. the callee (or server) process exists before the call and runs continuously, this process is created for each caller,

  ♦ this process listens for request messages, executes the appropriate procedure, and sends back response messages,

  ♦ this process may be called by any number of remote processes and executed sequentially,

  b. the callee process is created for each caller of for each procedure call

Parameters of RPC

- parameter passing in RPC is performed by stub procedures using either call-by value or call-by-copy/restore semantics:

call-by-value

♦ a value passed to a procedure is copied to a local variable upon the entry to the procedure,

♦ in RPC the stub copies parameters into a message and transmit it,

call-by-copy/restore

♦ it is a call-by-value semantics at the entry of a procedure call, and call-by-reference is restricted to the exit of the call,

♦ results are copied back to the calling procedure at the completion of the called procedure without making any memory references during the execution of the procedure,

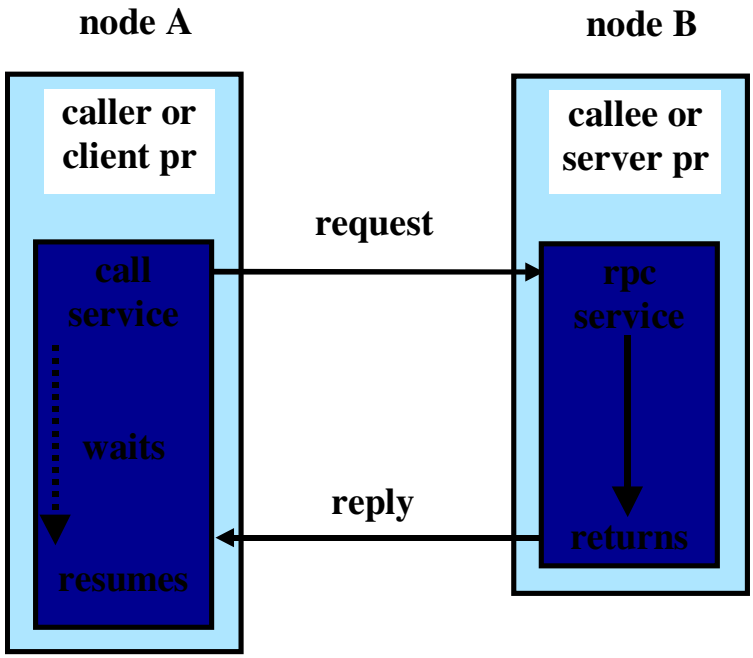
Primitives of RPC

1. call service (value, result)

- **the primitive provides interaction of a client and a server using remote procedure call,**
- **service is a name of a communication channel containing both source the destination id, for example in direct communication it designates the corresponding server process,**
- **the "value" arguments are sent to the server and the calling process delays until the service has been completed and the results have been returned and assigned to the "result" arguments,**

2. remote service (value, result)

- **the callee process awaits receipt of a message containing value arguments from a caller process, and assigns them to its value parameters,**
- **next, executes the primitive, and then returns a response message containing the values of the result parameters,**



remote procedure call

Activities of RPC

STEP 1 **a client makes a normal local call which invokes a corresponding procedure in the client stub,**

STEP 2 **the client stub prepares the message buffer, and places a message, which includes the target procedure and arguments, (marshalling) in the message buffer, and the client stub traps to the kernel,**

STEP 3 **the kernel copies the message into its address space and defines the destination address and puts it in the message header,**

the RPC routine in the transport layer sends a message to the callee computer which also runs the corresponding RPC routine,

STEP 4 **the RPC routine in the callee computer receives the message from the RPC routine in the caller computer**

the RPC checks the message for validity and decides which stub to give it, the RPC routine in the transport layer passes the message to the corresponding server stub and switches the context to the server stub

STEP 5 **the server stub unpacks the received message (unmarshalling) and makes a normal call to invoke the appropriate procedure in the server**

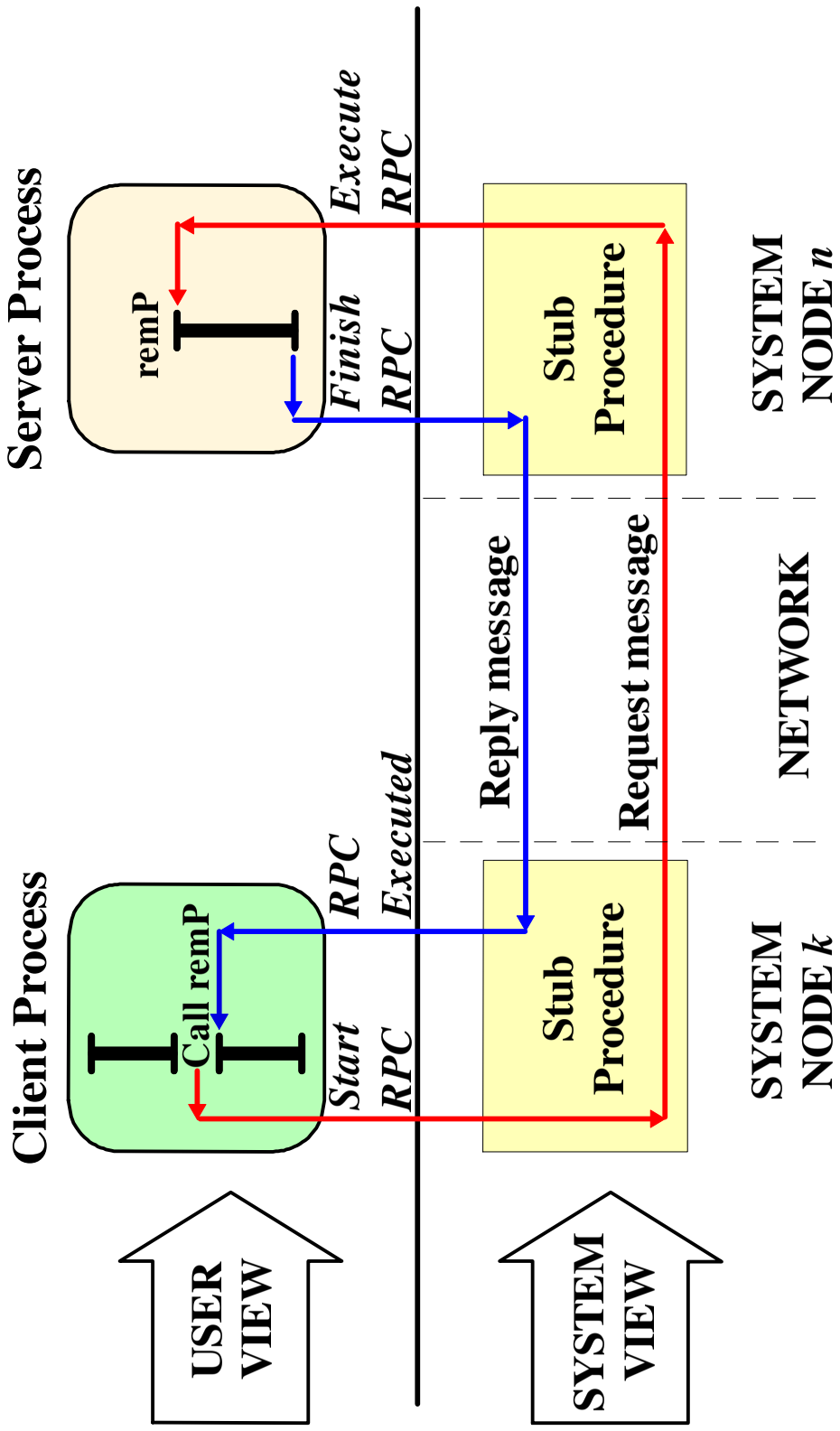
STEP 6 **when the call in the server completes, it returns results to the server stub**

STEP 7 **the server stub places results into a message and asks RPC routine in the transport layer to transmit to the caller computer**

STEP 8 **the RPC routine transmits the message to the caller computer**

STEP 9 **the RPC routine in the caller computer receives the message, and passes it to the client stub**

STEP 10 **the client stub unpacks results, and returns to the user and assigns to the result arguments**


Server Process
remP
Execute RPC
Finish RPC
Stub Procedure
SYSTEM NODE n
Reply message
Request message
NETWORK
Client Process
Call remP
RPC Executed
Start RPC
Stub Procedure
SYSTEM NODE k
USER VIEW
SYSTEM VIEW

Interface processing (parameter passing and data conversion)

- differences among heterogeneous system components make necessary management of different data types, data representation forms and data transfer syntax,

- marshalling is the translation of the language-level data structure between two computers by packing the data into a network buffer on one computer before transmission and doing inverse operations on the other computer,

- a client stub procedure converts the local procedure call to a remote procedure call to the server,

- at the client site before sending a message the client stub procedure marshals the arguments and packs them up with the procedure identifier in the request message to select one of the server stub procedures and sends the message to the server stub,

- at the server site after receiving a message the server stub procedure unmarshals the arguments, calls the appropriate service procedure,

Binding

- determination of the computer address of the remote procedure (callee or server) and the specification of the procedure to be invoked,

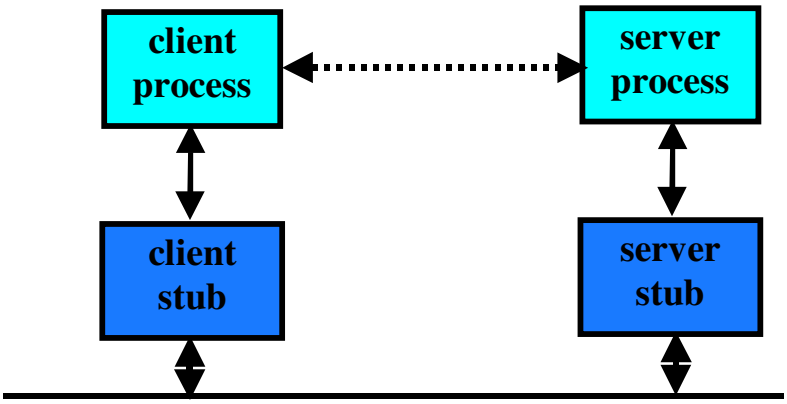
- it may also check the compatibility of the parameters passed and the procedure type called by the RPC,

- mapping from a name to a particular object, usually identified by a communication identifier, i.e. from a service name to a server port,

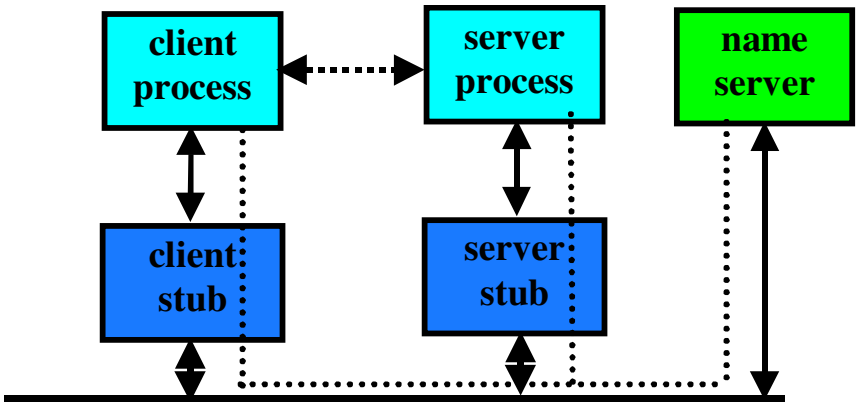
- binder is used by servers to make their port identifiers known to potential clients,

- binder maintains a table containing mappings from service names to server ports,

- clients can locate the server by either broadcasting a request of the service or using a dedicated binding name server,



binding based on broadcasting



binding based on name server

- when a client calls a remote procedure for the first time, the client stub checks whether it is not yet assigned to a server by sending a message to the binder asking to import the corresponding version of the interface,

- the binder checks to see if one or more servers have already exported the requested interface,

- if the server exists, the binder assigns its handle and unique identifier to the client stub, that uses the handle as the address to send the request message to,

- procedures of binding:

1. **register – the binder records the service name, the server port and its version if the server process starts execution,**

2. **lookup – when a client process starts, it sends a message to the binder requesting it to look up the requested service and to return its address if the version is OK,**

3. **withdraw – the binder removes the service from its table if the server process terminates,**

Communication

- communication module uses request-reply communication to send and receive messages among client and server processes,

RPC compilation and linking

- *compilation* of RPC programs require the following components in the RPC package:

- **an interface specification file,**
- **an RPC generator,**
- **a run-time library,**

- the RPC generator creates the stub procedure and a header file for the independent compilation of the client and server programs using the interface specification file as input,

- server stub registers and initialises a server in order to provide the requested service,

- the run-time library supports the execution of RPC by supporting data conversion, binding and communication,

- *linking* a server announces itself by exporting itself through the RPC support mechanism,

- a client binds itself to a specific server by making an import call to this mechanism,

- *calling* can take place after completing the binding process by employing some transport layer protocol to transmit parameters between clients and servere,

RPC for MANY CLIENTS and MANY SERVERS

- **a client kernel can provide services to more than one process, making a remote procedure call at the same time,**

- **there are client channels that connect client processes to a server process, they serialise the RPCs from various client processes,**

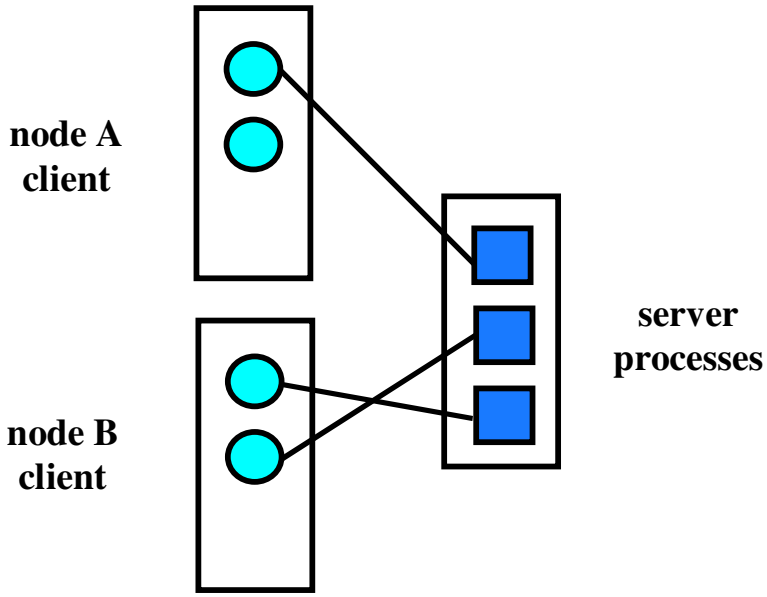
- **there are few client channels in each client computer, and processes compete them, i.e. a client process can use a channel if it is free,**

- **the server processes are multiplexed between the client channels that use the server, the server process keeps information about the client channels,**

- **the kernel process assigns a server process to a particular client channel for a series of RPCs,**

- **the server process is released after receiving an acknowledgement from a client channel ending the RPC,**

- **the server computer has several server processes, thus it can handle more than one request**



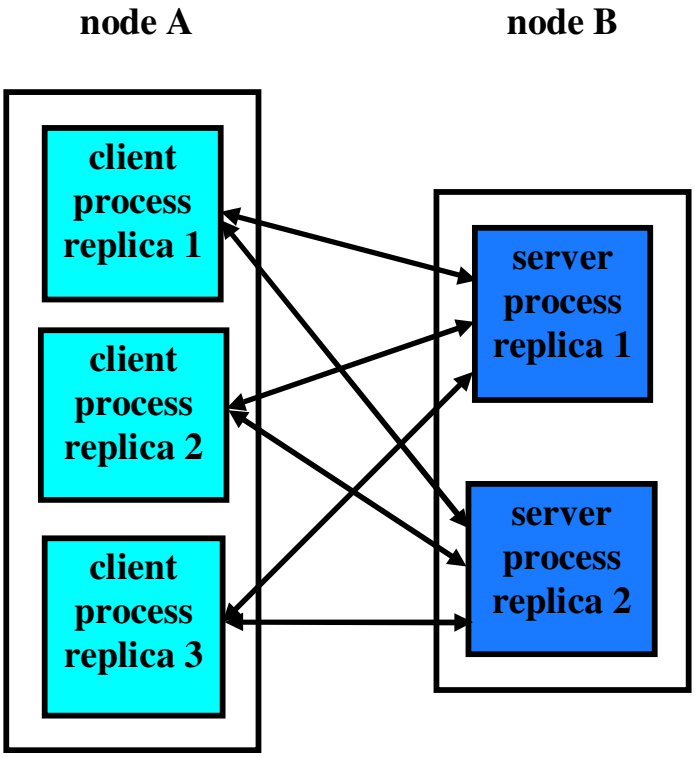
client channel and server processes

troupe = **set of replicas of module executing on several computers,**

replicated RPC = **the extension of the basic RPC for communicating troupes**

- the replicated RPC provides many-to-many communication with exactly once execution at all troupe members,

- i.e. when a client troupe makes a call to a server troupe, each process of the server troupe performs the requested procedure exactly once, and each process in the client troupe receives all the results,



replicated procedure call

RPC and FAILURE HANDLING

Exception handling in RPC

- any RPC may fail thus, the RPC requires an effective exception handling mechanism for reporting such failures to the client process (caller),

- when an error occurs in a procedure, an exception is raised and the appropriate handling procedure is automatically executed in the caller's environment,

- on the other side, a client or stub procedure may have to stop the execution of the corresponding server procedure

- problems of exception handling:

1. **How does the server inform clients about exception(s) – status information ?**

2. **How does the client inform the server about exception(s) – control information ?**

solution:

- control and status information is exchanged via data channels for example separate channel to have a dedicated status channel,

Failures in RPC

1. client cannot locate the server

possible faults:

- **server might be down or does not exist,**
- **mismatch of the client and server version,**

solution: raise an exception informing about the failure,

2. request message from the client is lost

possible faults:

- **failure in the communication network**
- **the server might be down or does not exist**

solution: kernel starting a timer when sending the message (time-out),

if the timer expires before a reply message, the kernel re-send the request message,

3. reply message from the server is lost

possible faults:

- **failure in the communication network**
- **the server might be down or does not exist**

solution: if there is no reply from the server then the client sends the request one more time,

there is a problem, the client does not know why there is no answer (reply get lost or the server is down)

it should be told server whether the request is idempotent or non-idempotent, (whether the request could be executed more than one time without harm being done)

4. the server crashes after receiving a request

possible faults:

- **a request arrives, but the server crashes before it can even be started or performed,**
- **a request arrives from a client and is executed, but the server crashes before it can send the reply,**
- **the client sends the second request to the server after time-out of the first request, the server checks the cache table to define whether or**

not the request is a duplicate,

<u>solution:</u> 1. ***at least once approach*** **= to keep trying until a reply has been received, then give it to the client**

2. ***at most once approach*** **= to give up immediately and reports back the failure, then the RPC may be performed at mots one time**

3. ***no guarantee approach*** **= when a server crashes, the client gets no help and no promises, the RPC may have been carried out anywhere**

4. ***exactly once approach*** **= it would be an optimal solution, but it cannot be achieved**

5. the client crashes after sending a request

<u>possible faults:</u>

- **orphan computation (unwanted computation) = client sends a request to a server and crashes before the server replies,**
- **orphan computation consume server resources and may also confuse the client after rebooting,**
- **it is difficult for the server to detect the disappearance of clients (solution = using time-out, or waiting for rebooting of the client),**

<u>solution:</u> 1. ***extermination*** **= before a client stub sends a request message, it makes a log entry telling what it is about to do,**

the log is kept in order to survive the crash, after a reboot, the log is checked and the orphan is explicitly killed off

2. ***reincarnation*** **= it divides time up into sequential time slots, when a client reboots, it broadcasts a message to all nodes declaring the start of a new time slot, when such a broadcast comes in, all remote computations are killed**

3. ***gentle reincarnation*** **= when a time slot broadcast comes in, each machine checks to see if it has any remote computations, and if so it tries to locate their owner**

4. *expiration* = each RPC has a constant amount of time to do a task, if it cannot finish, it must explicitly ask for another quantum,

if after a crash the server waits a time before rebooting, all orphans are sure to be gone

5. reversing = the server occasionally tries to locate the owners of its remote operations and abort those whose owners cannot be found,