DISTRIBUTED OPERATING SYSTEMS

REMOTE PROCEDURE CALLS

Overview

- The RPC Model
- Transparency
- Implementation
- Stub
- Messages
- Marshaling
- Server Management
- Parameter Passing Semantics – Call Semantics
- Communication protocols
- Client server Binding
- Heterogeneous

RPC

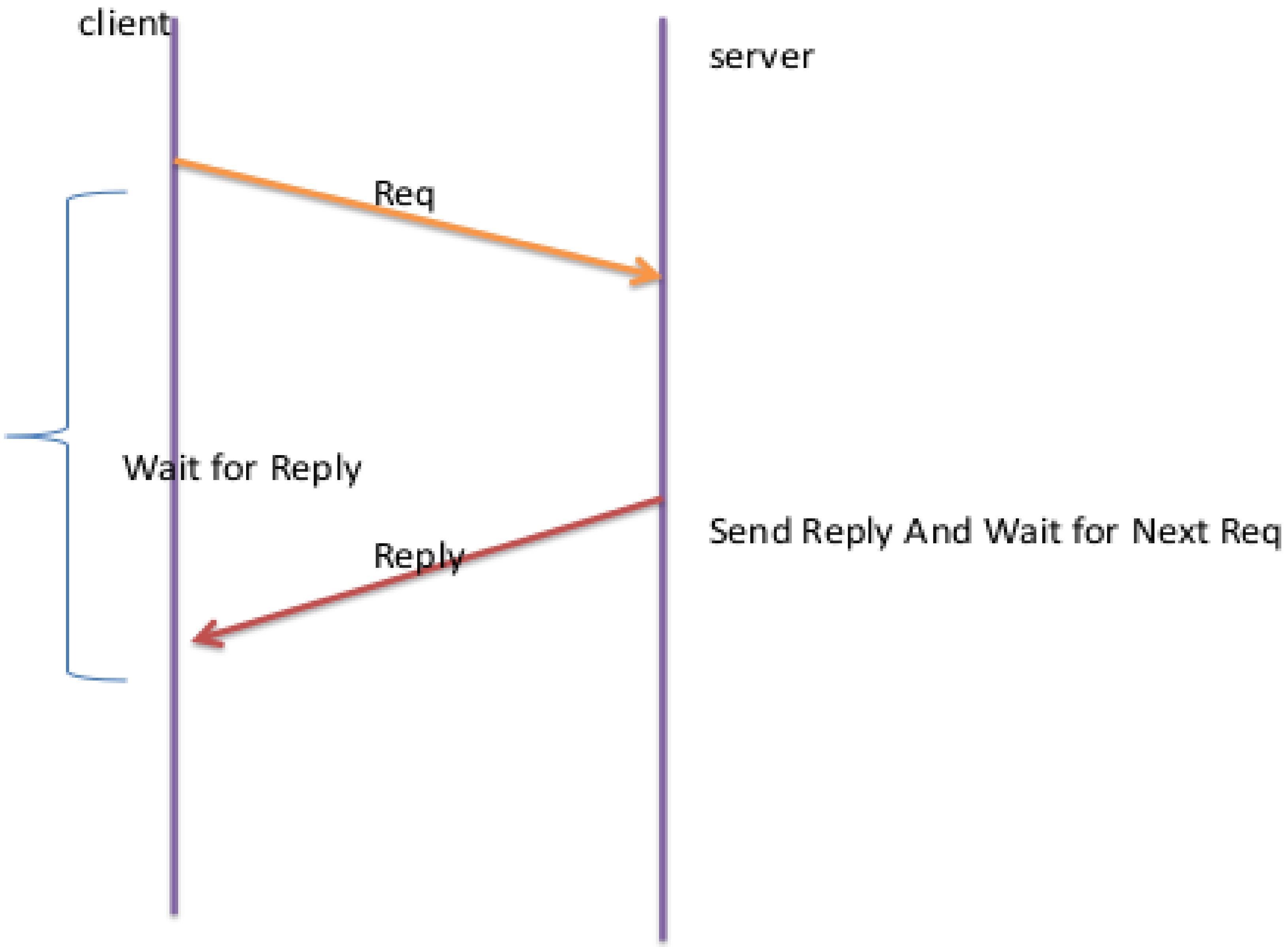
- The RPC is an accepted IPC mechanism in distributed systems.

RPC

- Simple call syntax.
- Familiar semantics - similar to local procedure calls
- A well-defined interface.
- Compile-time type checking and automated interface generation.
- Its ease of use.
- Its generality
- Its efficiency.
- Facilitate to communicate between all processes

RPC MODEL

- The caller places arguments to the procedure in some well-specified location.
- Control is then transferred to the sequence of instructions that constitutes the body of the procedure.
- The procedure body is executed in a newly created execution environment
- After the execution is over, control returns to the calling point, With result.

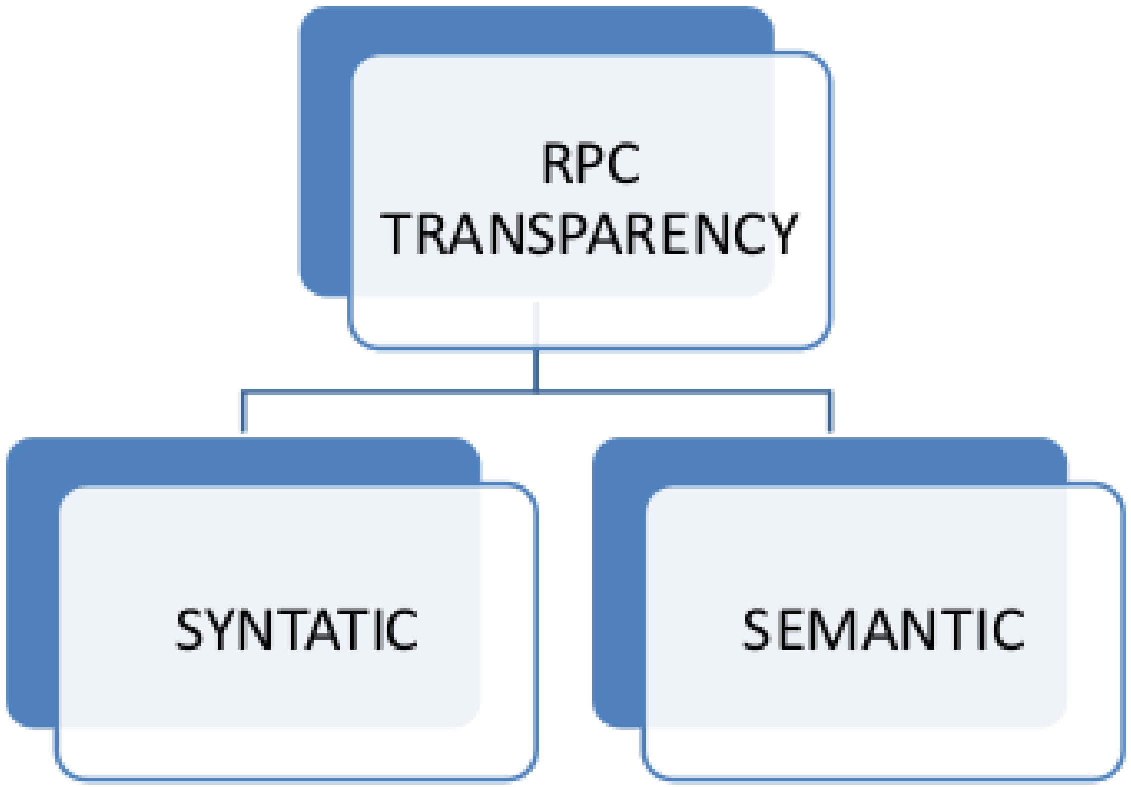

client
server
Req
Wait for Reply
Send Reply And Wait for Next Req
Reply

RPC

>Transparency of RPC
>Local procedures and remote procedures are indistinguishable to programmers.

RPC

- Transparency of RPC
- Local procedures and remote procedures are indistinguishable to programmers.



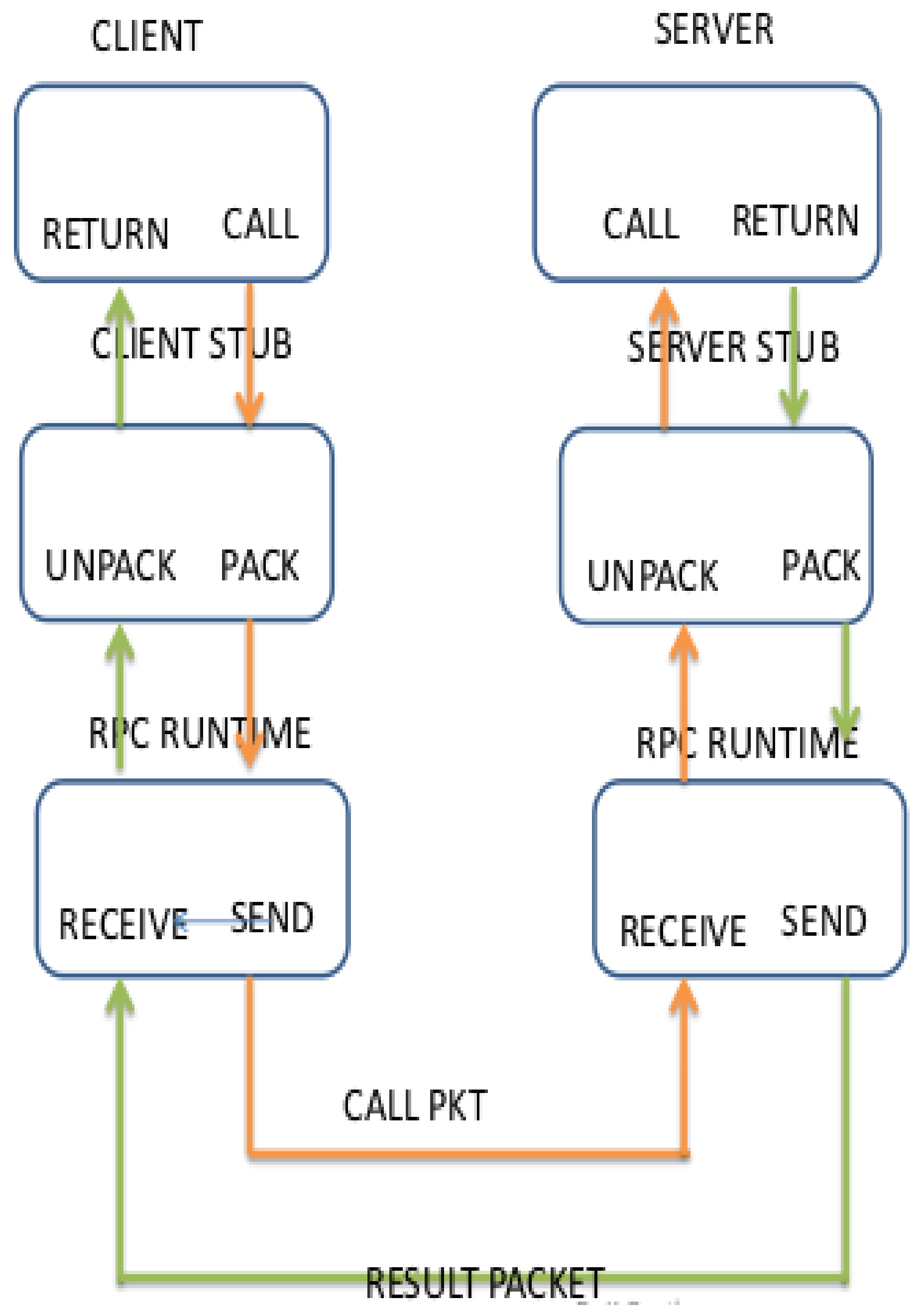
RPC

- The calling process is suspended until the called procedure returns.
- The caller can pass arguments to the called procedure (Remote procedure).
- The called procedure (remote procedure) can return results to the caller.

IMPLEMENTING RPC MECHANISM

Five elements of program with RPC

- 1. The client
- 2. The client stub
- 3. The RPC Runtime
- 4. The server stub
- 5. The server


CLIENT
SERVER
RETURN
CALL
CALL
RETURN
CLIENT STUB
SERVER STUB
UNPACK
PACK
UNPACK
PACK
RPC RUNTIME
RPC RUNTIME
RECEIVE
SEND
RECEIVE
SEND
CALL PKT
RESULT PACKET

IMPLEMENTING RPC MECHANISM

Client

- A user process that initiates a remote procedure call.
- Makes a normal local call that invokes a procedure in the client stub.

Client Stub

- A stub is a piece of code that converts parameters during a remote procedure call (RPC)
- Responsible for conversion (marshalling) of parameters and de conversion of results .
- Packs a procedure and the arguments into a message for local RPC Runtime to send it to the server stub.
- Unpacks the result and passes it to the client.

IMPLEMENTING RPC MECHANISM

RPC Runtime

- Handles transmission of messages between client and server.
- Responsible for retransmissions, acknowledgments, packet routing, and encryption.
- on the client machine receives the call request message from the client stub. Receives the result of procedure execution
- on the server machine receives the message containing the result of procedure from the server stub and receives the call request.

IMPLEMENTING RPC MECHANISM

Server Stub

- On receipt of the call request message from the local RPCRuntime, the server stub unpacks it and makes a perfectly normal call to invoke the appropriate procedure in the server.
- On receipt of the result of procedure execution from the server, the server stub packs the result into a message and then asks the local RPCRuntime to send it to the client stub.

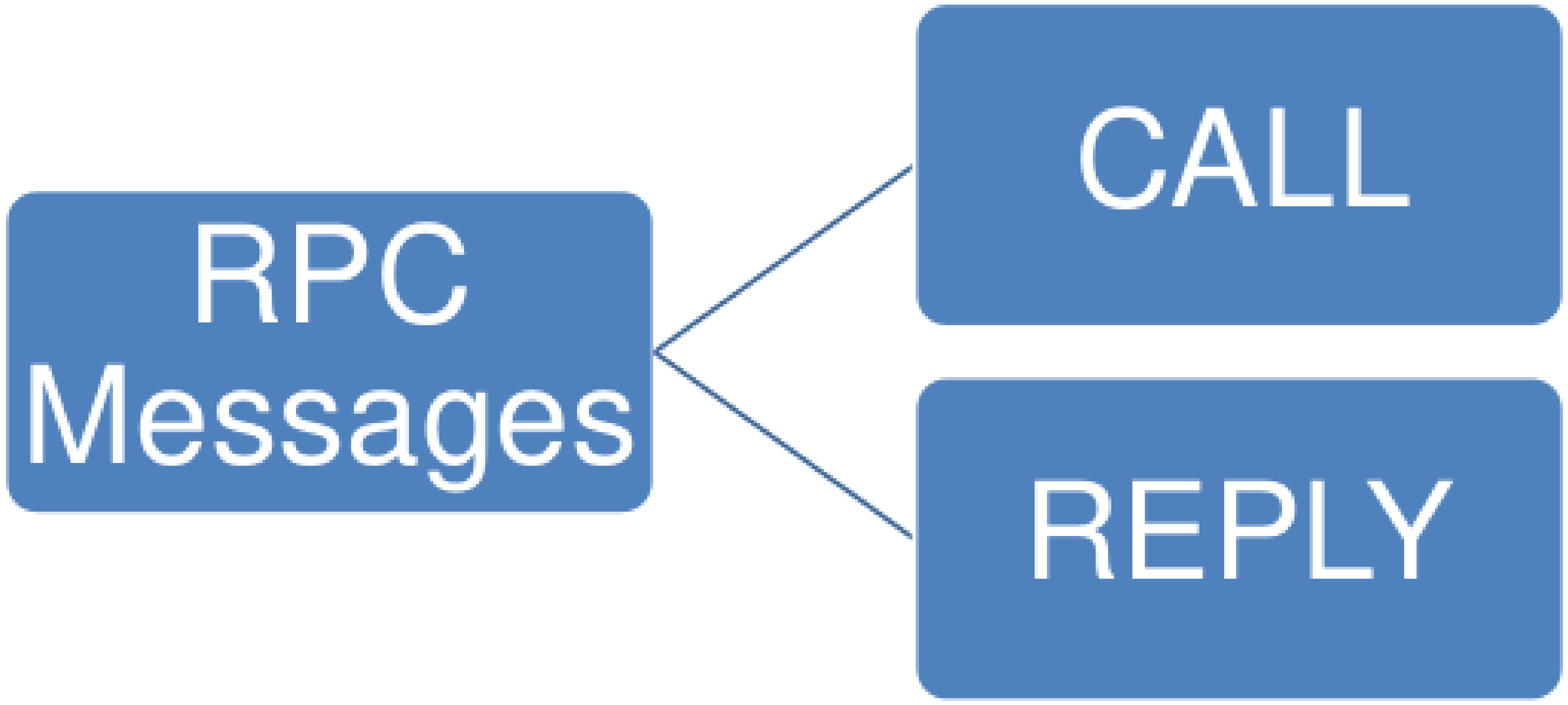
IMPLEMENTING RPC MECHANISM

- **Server**
  - On receiving a call request from the server stub,
    - Executes the appropriate procedure and
    - Returns the result of procedure execution to the server stub

IMPLEMENTING RPC MECHANISM

- **STUB GENERATION**
- • *Manually.*
  - RPC implement or provides a set of translation functions
  - User can construct his or her own stubs.
  - Simple to implement and can handle very complex parameter types.
- • *Automatically.*
  - Commonly used method for stub generation.
  - Uses Interface Definition Language (JDL) for defining the interface between a client and a server.

RPC MESSAGES



RPC MESSAGES

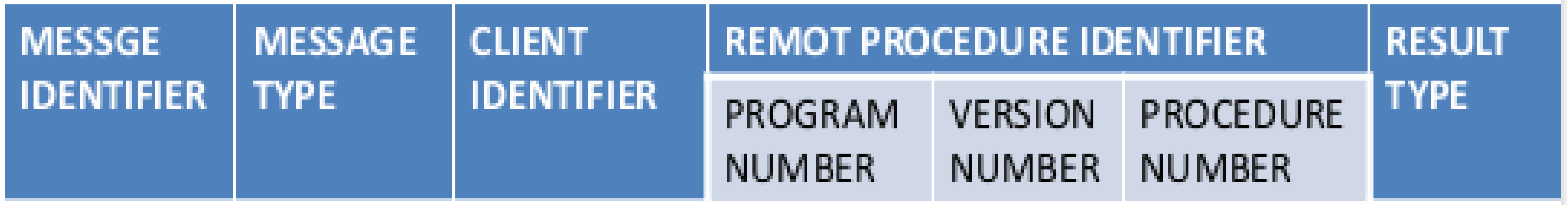
- 1. *Call Messages:* that are sent by the client to the server for requesting execution of a particular remote procedure
- 2. *Reply Messages:* that are sent by the server to the client for returning the result of Remote Procedure Execution

RPC MESSAGES

- The FIVE basic components necessary in a call message are as follows:
  - The identification information of the remote procedure to be executed
  - The arguments necessary for the execution of the procedure
  - A message identification field that consists of a sequence number.
  - A message type field that is used to distinguish call messages from reply messages.
  - A client identification field for executing the concerned procedure

RPC MESSAGES

- CALL MESSAGE



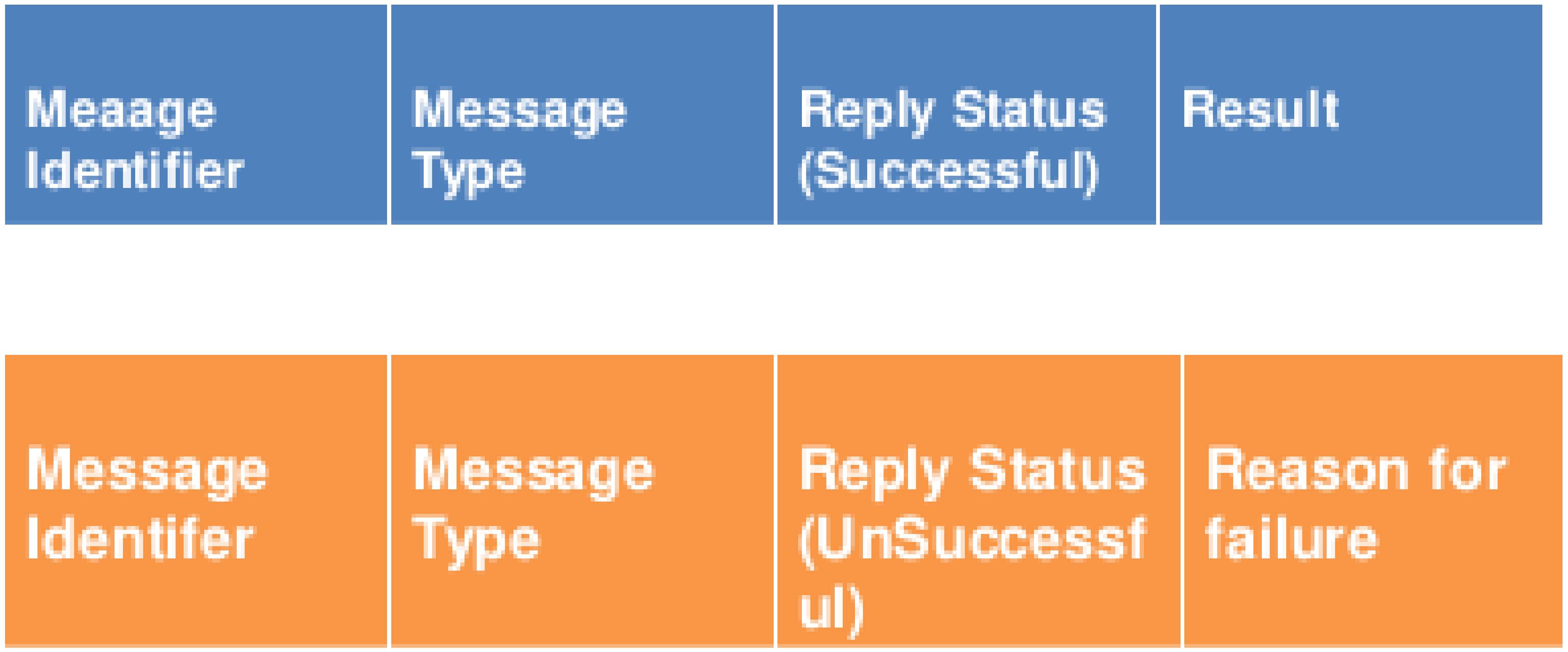
MESSGE IDENTIFIER	MESSAGE TYPE	CLIENT IDENTIFIER	REMOT PROCEDURE IDENTIFIER			RESULT TYPE
			PROGRAM NUMBER	VERSION NUMBER	PROCEDURE NUMBER	

RPC MESSAGES

- REPLY MESSAGES
- 1, Not intelligible to it. The server rejects it.
- 2. Not authorized to use the service. The server will return an unsuccessful reply
- 3. The remote program, version, or procedure number not available with it. Will return an unsuccessful reply
- 4. The remote procedure is not able to decode the supplied arguments.
- 5. An exception condition (such as division by zero) occurs while executing the specified remote procedure.

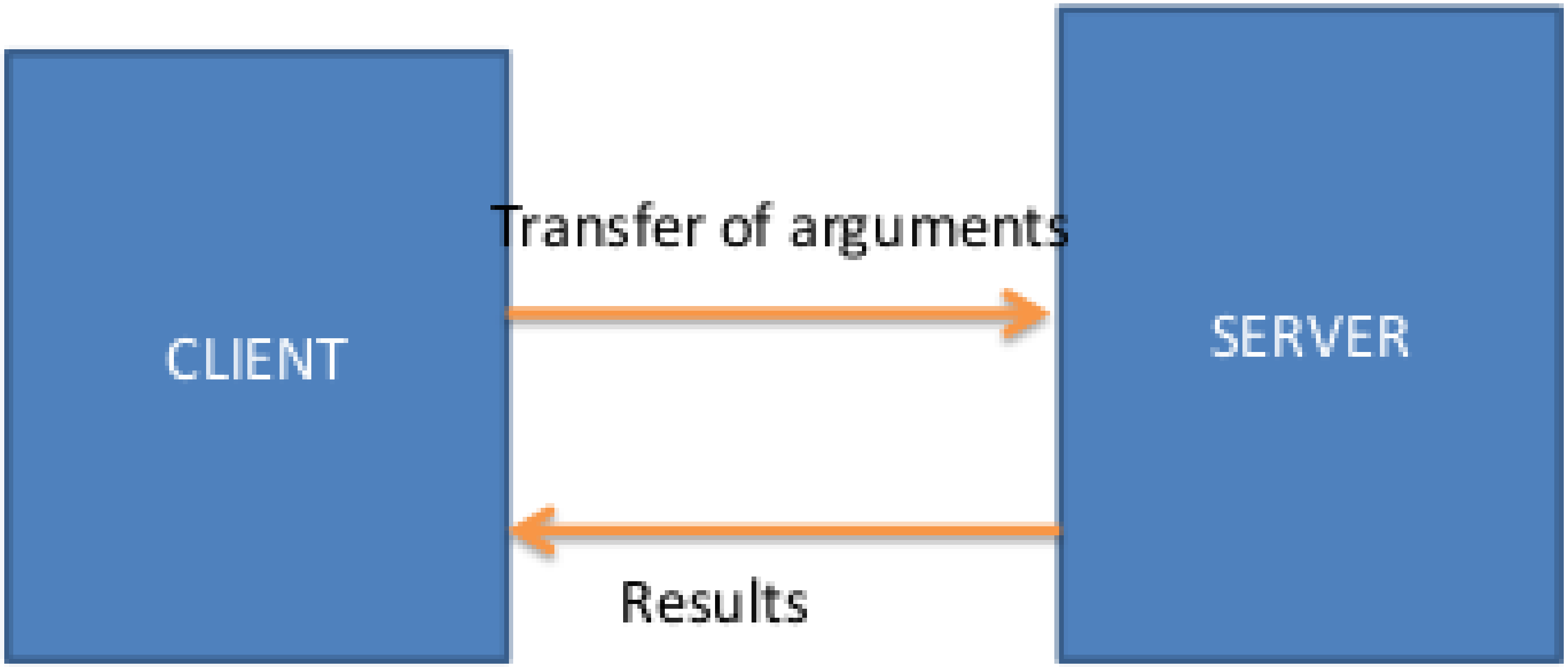
RPC MESSAGES

- SUCCESSFUL AND UNSUCCESFUL REPLY



Marshalling Arguments

- Implementation of remote procedure calls



Marshalling Arguments

- Transfer of message data requires encoding and decoding of the message data.
- For RPCs this operation is known as *Marshaling* and involves the following Actions:
  - 1. Taking the arguments of a client process or the result of a server
  - 2.Encoding the message data of step 1 above on the sender's computer.
  - Placing them into a message buffer.
  - 3. Decoding of the message data on the receiver's computer.
  - Then reconstruction of program objects from the message data that was received in stream form.

Marshalling Arguments

- Marshalling may be
  - 1. Provided as a part of the RPC software.
  - 2. Those that are defined by the users of the RPC system.
- A good RPC system
  - generate in-line marshaling code for every remote
  - it is difficult to achieve this goal because of the large amounts of code

SERVER MANAGEMENT

- **Stateful Servers**
  - Open (*filename, mode*): This operation is used to open a file identified by filename in the specified mode.
  - Read (*byt id, n, buffer*): This operation is used to get *n* bytes of data from the file
  - Write (*fid, n, buffer*): On execution of this operation, the server takes *n* bytes of data
  - Seek (*fid, position*): causes the server to change the value of the *read write pointer*
  - Close (*fid*): This statement causes the server to delete from its *file-table* the file state

SERVER MANAGEMENT....

- Stateless Servers
  - Read (filename, *position,n, buffer*): This operation is used to get *n* bytes of data from the file, the position within the file from where to begin reading is specified as the *position* parameter.
  - Write (*filename, position, n, buffer*): On execution of this operation, the server takes *n* bytes of data, the position within the file from where to begin writing is specified as the *position* parameter.

Server Creation Semantics

- The remote procedure to be executed is totally independent of the client process.
- Based on the time duration for which RPC servers survive, they may be classified as
  - Instance-per-call servers,
  - Instance-per-transaction or Session servers
  - Persistent servers.

Server Creation Semantics

- **Instance-per-Call Servers**
  - Servers belonging to this category exist only for the duration of a single call.
  - A server of this type is created by RPC Runtime on the server machine only when a call message arrives.
  - The server is deleted after the call has been executed.

Server Creation Semantics

- The servers of this type are stateless because they are killed as soon as they have serviced.
- The inter call state information will make the remote procedure calls expensive.
- If it is maintained by the client process, the state information must be passed to and from the server with each call.
- Will lead to the loss of data abstraction across the client-server

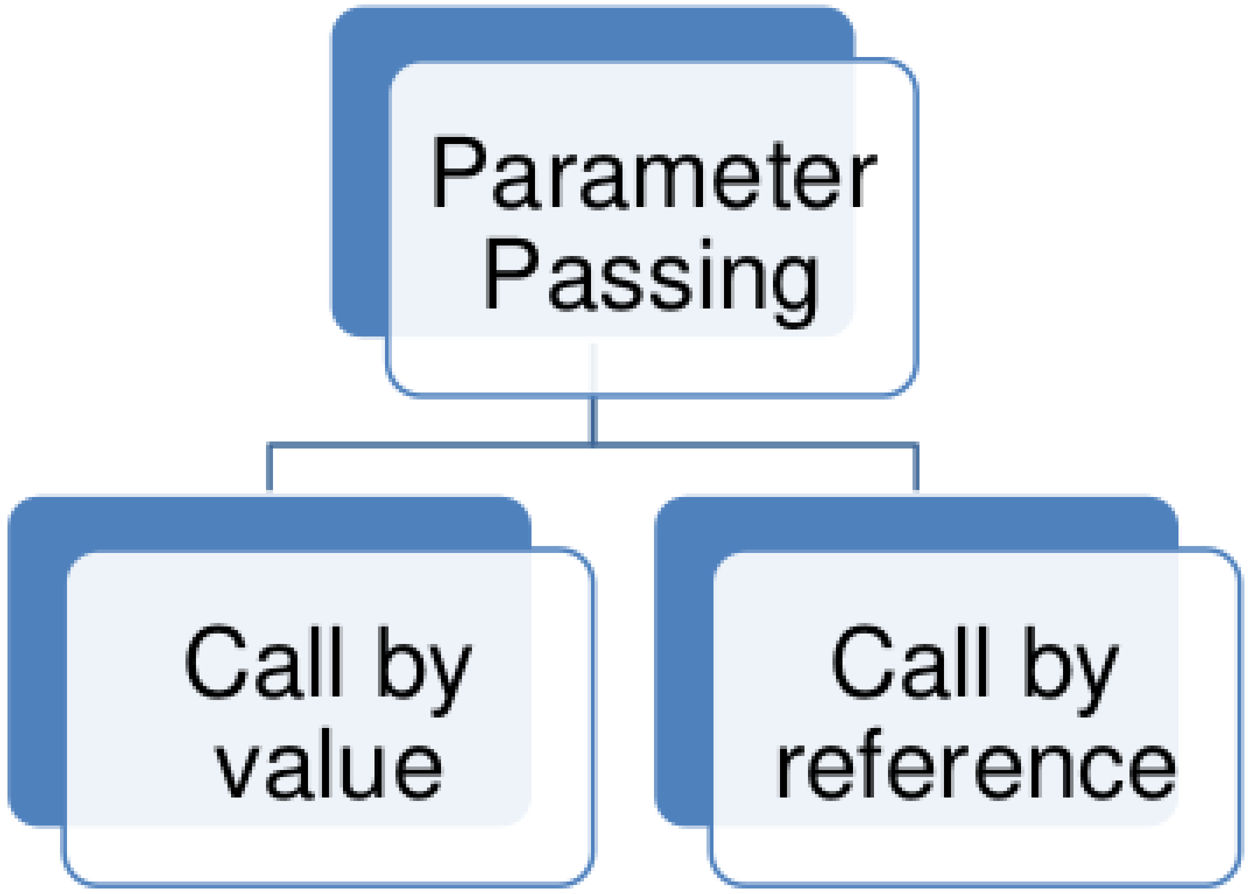
Server Creation Semantics

- **Instance-per-Session Servers**
  - Servers exist for the entire session
  - can maintain inter-call state information
  - The overhead for a client-server session that involves a large number of calls is also minimized.
  - There is a server manager for each type of service.

Server Creation Semantics

- **Persistent Servers**
  - Servers are usually created and installed before the clients that use them.
  - Minimum number of clients currently bound to it and returns the address of the selected server to the client.
  - The client then directly interacts with that server.
  - Manage several sets of state information.

PARAMETER PASSING SEMANTICS



PARAMETER PASSING SEMANTICS

- Call by value
  - In the Call-by- value method, all parameters are copied into a message
  - Passing larger data types such as multidimensional arrays, trees, and so on, can consume much time for transmission of data that may not be used.

PARAMETER PASSING SEMANTICS

- Most RPC mechanisms use the call-by-value semantics for parameter passing
- The client and the server exist in different address spaces, possibly even on different types of machines, so that passing pointers or passing parameters *by reference* is meaningless.

PARAMETER-PASSING SEMANTICS

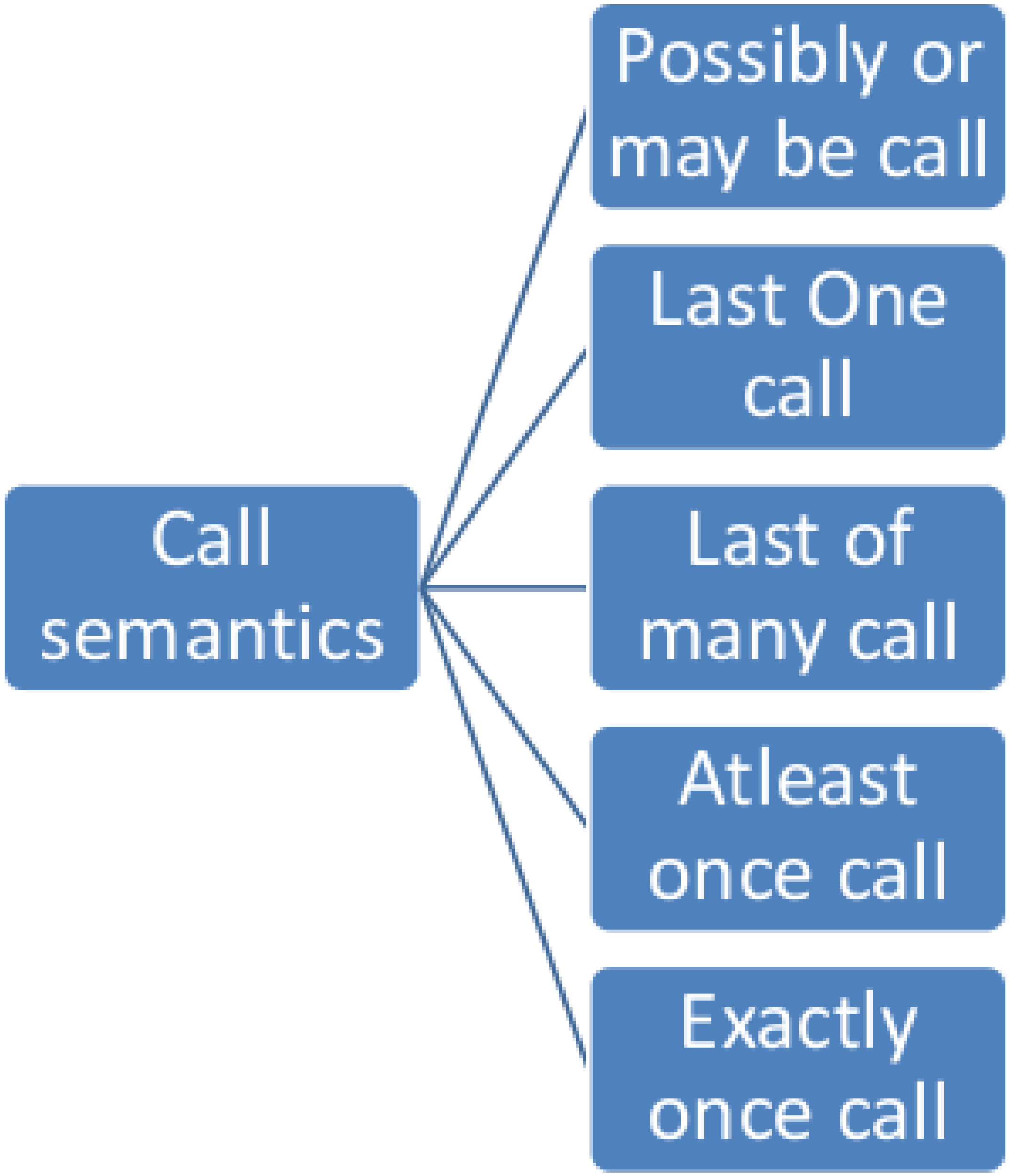
- In an object-based system that uses the RPC mechanism for object invocation, the call-by-reference semantics is known as call-by-object-reference.

PARAMETER·PASSING SEMANTICS

- Call By-Move
  - A parameter is passed by reference, as in the method of call-by-object-reference
  - but at the time of the call, the parameter object is moved to the destination node (site of the callee). –**Call By Visit**
  - The argument object may either return to the caller's node or remain at the callee' s node - **Call-By-Move**

Call SEMANTICS

- Normal functioning of an RPC may get disrupted due to
- – The call message gets lost.
- – The response message gets lost.
- – The callee node crashes and is restarted.
- – The caller node crashes and is restarted.


Call semantics
Possibly or may be call
Last One call
Last of many call
Atleast once call
Exactly once call

Call SEMANTICS

- **Possibly or may be call**
- This is the weakest semantics
- In this method, to prevent the caller from waiting indefinitely for a response from the callee, a timeout mechanism is used.
- The caller waits until a pre-determined timeout period and then continues with its execution.
- Does not guarantee anything about the receipt of the call message.
- The response message is not important for the caller

Call SEMANTICS

- **Last one call**
  - suppose process PI of node N1 calls
  - procedure Fl on node N2, which in turn calls procedure F2 on node N3.
- **Node N crashes.**
  - Node N1 's processes will be restarted, and
  - PI's call to F1 will be repeated. The second invocation of FI will again call procedure F2 on node N3. Unfortunately, node N3 is totally unaware of node N crash.
  - Therefore procedure F2 will be executed twice on node N3 and N3 may return the results of the two executions of F2 in any order

Call SEMANTICS

- The basic difficulty in achieving last-one semantics is caused by orphan calls.
- • An orphan call is one whose parent (caller) has expired due to a node crash.
- • To achieve last-one semantics, these orphan calls must be terminated before restarting the crashed processes Killing by “orphan extermination”

Call SEMANTICS

- **Last of many calls**
  - A simple way to neglect orphan calls is to use call identifiers to uniquely identify each call. When a call is repeated, it is assigned a new call identifier.
  - Each response message has the corresponding call identifier associated with it.
  - A caller accepts a response only if the call identifier associated with it matches with the identifier of the
  - Most recently repeated call; otherwise it ignores the response message.

Call SEMANTICS

- **At least once call**
  - This is an even weaker call semantics than the last-of-many call semantics.
  - Guarantees that the call is executed one or more times but does not specify which results are returned to the caller.
  - can be implemented simply by using timeout- based retransmissions
  - if there are any orphan calls, it takes the result of the first response message and ignores the others, whether or not the accepted response is from an orphan.

Call SEMANTICS

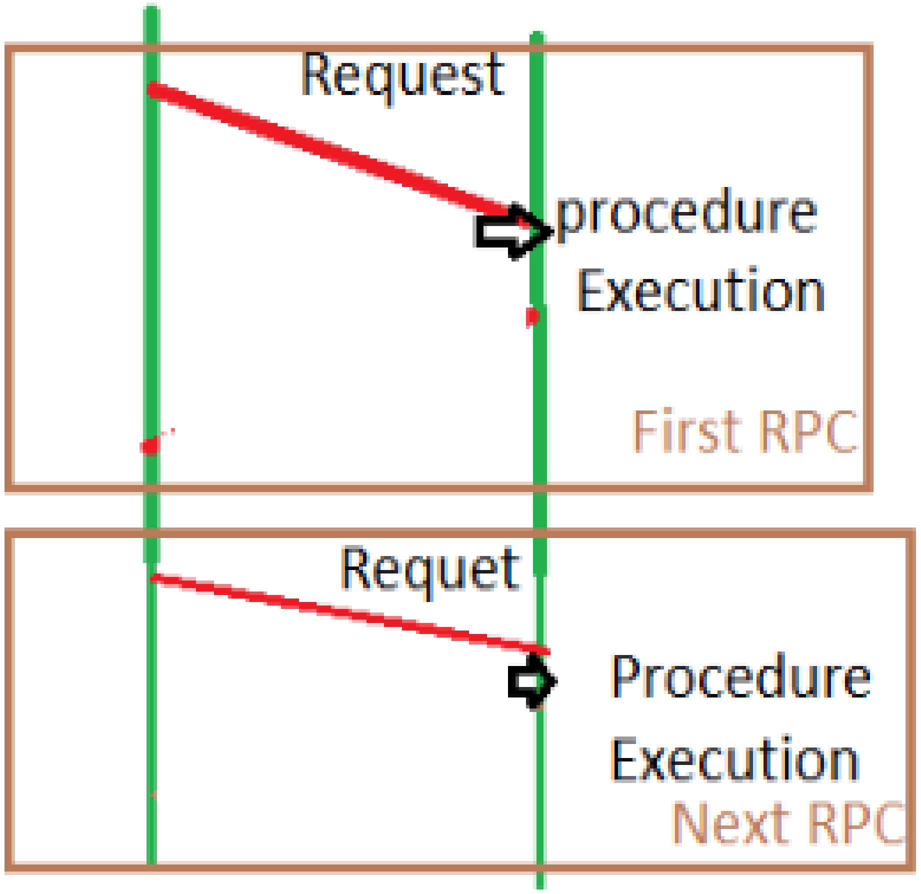
- **Exactly once call**
  - This is the strongest and the most desirable call semantics because it eliminates the
  - Possibility of a procedure being executed more than once
  - No matter how many times a call is retransmitted. The last-one, last-of-many, and at-least-once call semantics cannot guarantee this

Call SEMANTICS

- The main disadvantage of these cheap semantics is that, if a procedure is executed more than once with the same parameters, the same results and side effects will be produced
  - ReadNextRecord(Filename)
  - ReadRecordN(Filename, N)
  - AppendRecord(Filename, Record)
  - WriteRecordN(FiJename, Record, N)

COMMUNICATION PROTOCOLS FOR RPCs

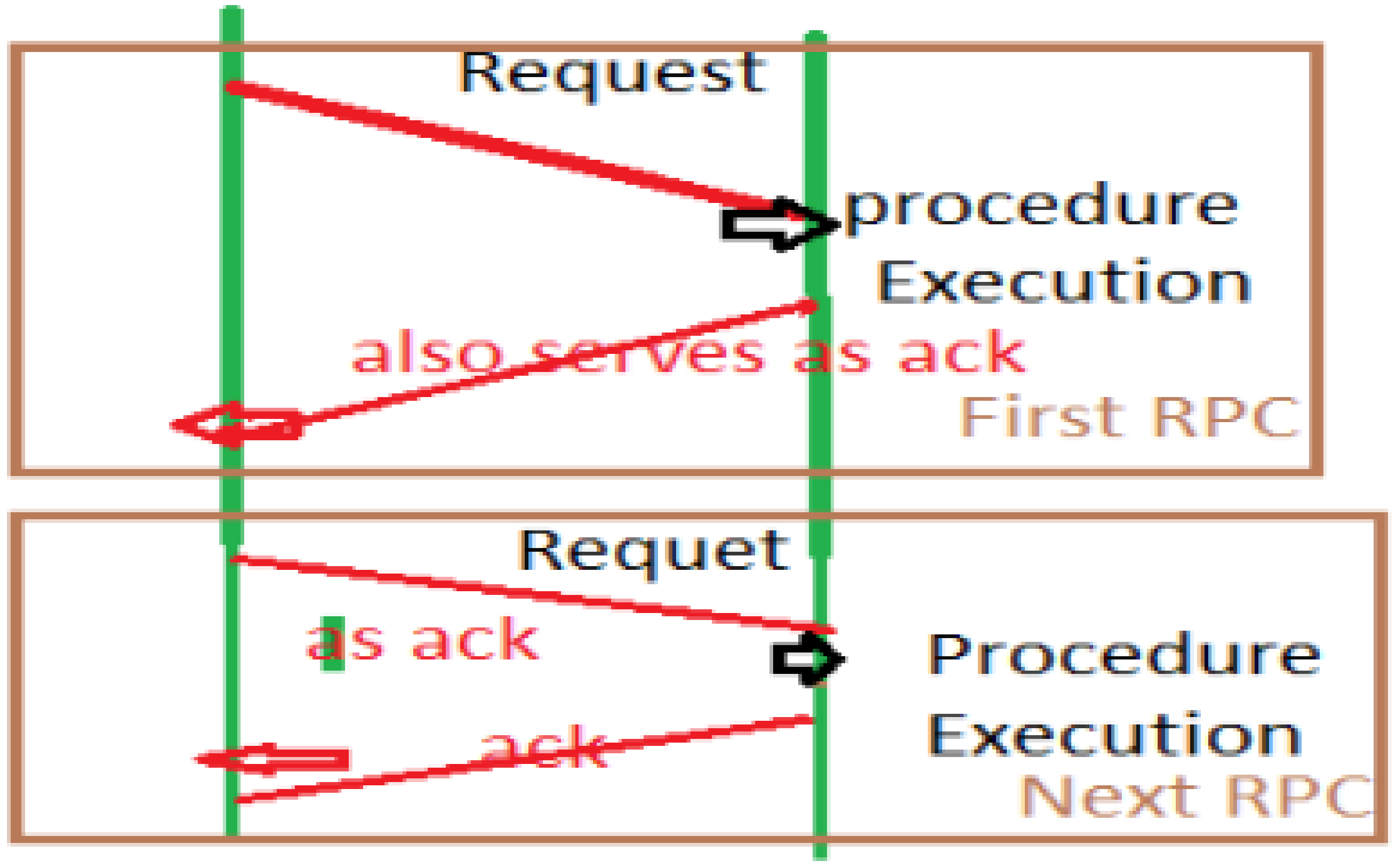
- The Request Prtocol -R Protocol



COMMUNICATION PROTOCOLS FOR RPCs

- The Request/Reply Protocol/The RR Protocol
  - using implicit acknowledgment to eliminate explicit acknowledgment messages.
  - A server's reply message is regarded as an acknowledgment of the client's request message.
  - A subsequent call packet from a client is regarded as an acknowledgment of the
  - server's reply message of the previous call made by that client.

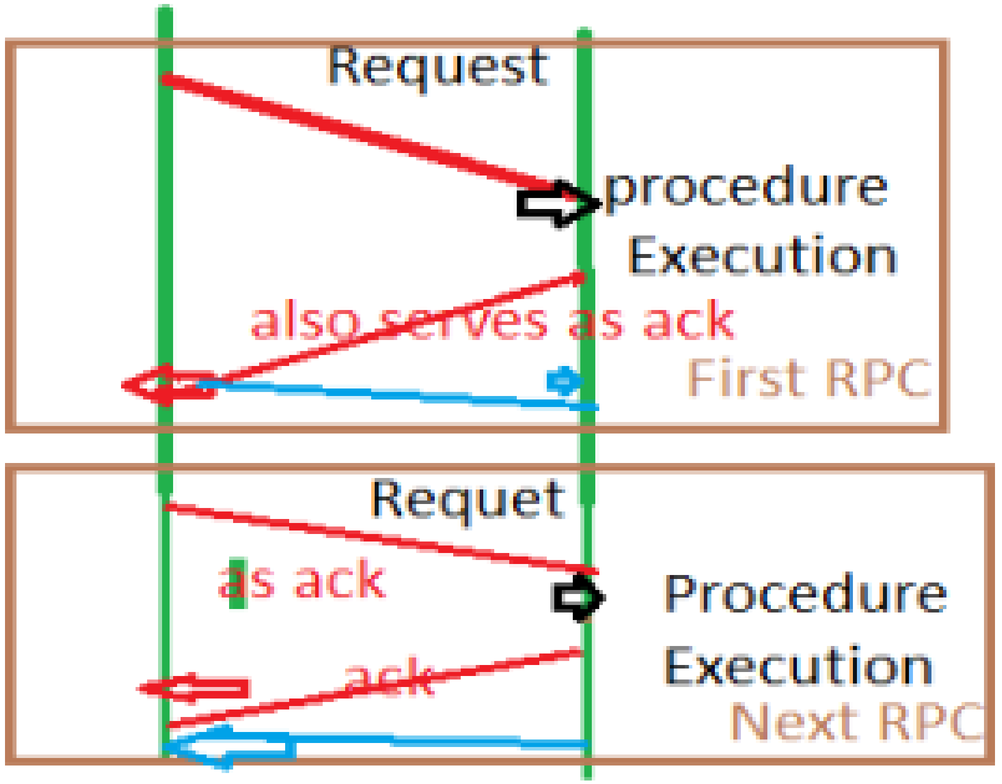
COMMUNICATION PROTOCOLS FOR RPCs



COMMUNICATION PROTOCOLS FOR RPCs

- The Request/Reply/Acknowledge.Reply Protocol

- The Request/Reply/Acknowledge.Reply Protocol



CLIENT-SERVER BINDING

- A client should know the location of a server before a remote procedure call and also know
- 1. To specify a server to which it wants to get bound
- 2. The binding process locate the specified server
- 3. Time to bind a client to a server
- 4. To change a binding during execution
- 5. Simultaneously bound to multiple servers that provide the same

CLIENT-SERVER BINDING

- **Server Naming**
- – the use of interface names
- – interface name has two parts
- – a type and an instance
- – Type specifies the interface itself and instance specifies a server providing the services within that interface.

CLIENT-SERVER BINDING

- **Server Locating**
- • The two most commonly used methods
- **Broadcasting.**
- • message to locate the desired server is broadcast to all the nodes from the client node.
- • The nodes on which the desired server is located return a response message.
- • Desired server may be replicated on several nodes so the client node will receive a response from all these nodes.
- • The first response that is received at the client's node is given to the client process and all subsequent responses are discarded.
- • This method is easy to implement suitable for use for small networks only

CLIENT-SERVER BINDING

- Binding Agent.
  - A name server used to bind a client to a server by providing the client with the location information of the desired server.
  - Maintains a binding table, which is a mapping of a server's interface name to its locations.
  - All servers register themselves with the binding agent as a part of their initialization process.,

CLIENT-SERVER BINDING

- To register with the binding agent, a server gives
  - Binder its identification information and a handle used to locate it.
  - A server can also deregister with the binding agent when it is no longer prepared to offer service.
  - The binding agent can also poll the servers periodically, automatically deregistering any server that fails to respond.

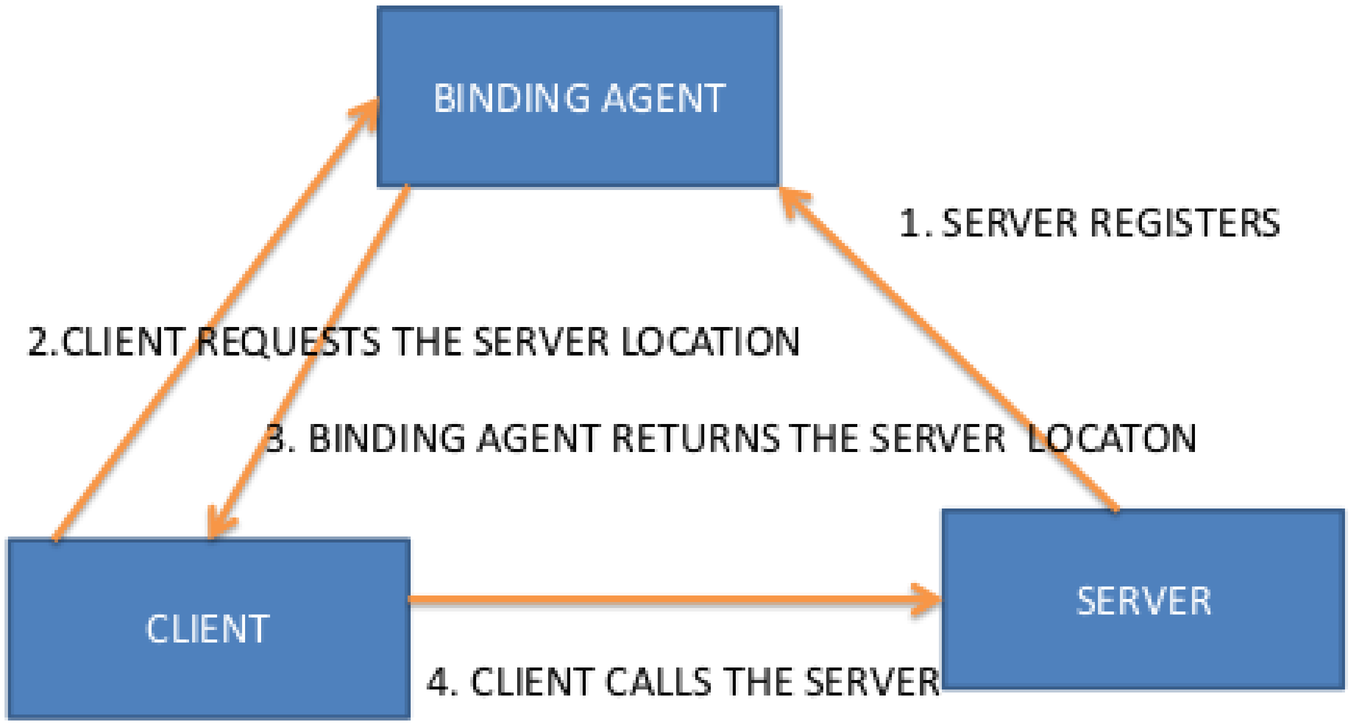
CLIENT-SERVER BINDING

- A binding agent interface has three primitives:
  - Register is used by a server to register itself with the binding agent,
  - Deregister is' used by a server to deregister itself with the binding agent, and ༀLookup is used by a client to locate a server.

CLIENT-SERVER BINDING

- The binding agent mechanism has several advantages.
- • This method can support multiple servers.
- • Higher fault tolerance.
- • The clients can be spread evenly over the servers to balance the load.
- • Servers specify a list of users who may use its service, in which case the binding agent would refuse to bind those clients servers who are not authorized to use its service.

CLIENT-SERVER BINDING


BINDING AGENT
1. SERVER REGISTERS
2.CLIENT REQUESTS THE SERVER LOCATION
3. BINDING AGENT RETURNS THE SERVER LOCATON
CLIENT
SERVER
4. CLIENT CALLS THE SERVER

CLIENT-SERVER BINDING

- Drawbacks
- – The overhead involved is large.
- – Replication involves extra overhead of keeping the multiple replicas consistent.

CLIENT-SERVER BINDING

- **Binding Time**
- – A client may be bound to a server at compile time, at link time, or at call time
- – **Binding at Compile Time**
- – it may be used in an application whose configuration is expected to remain static for a fairly long time.

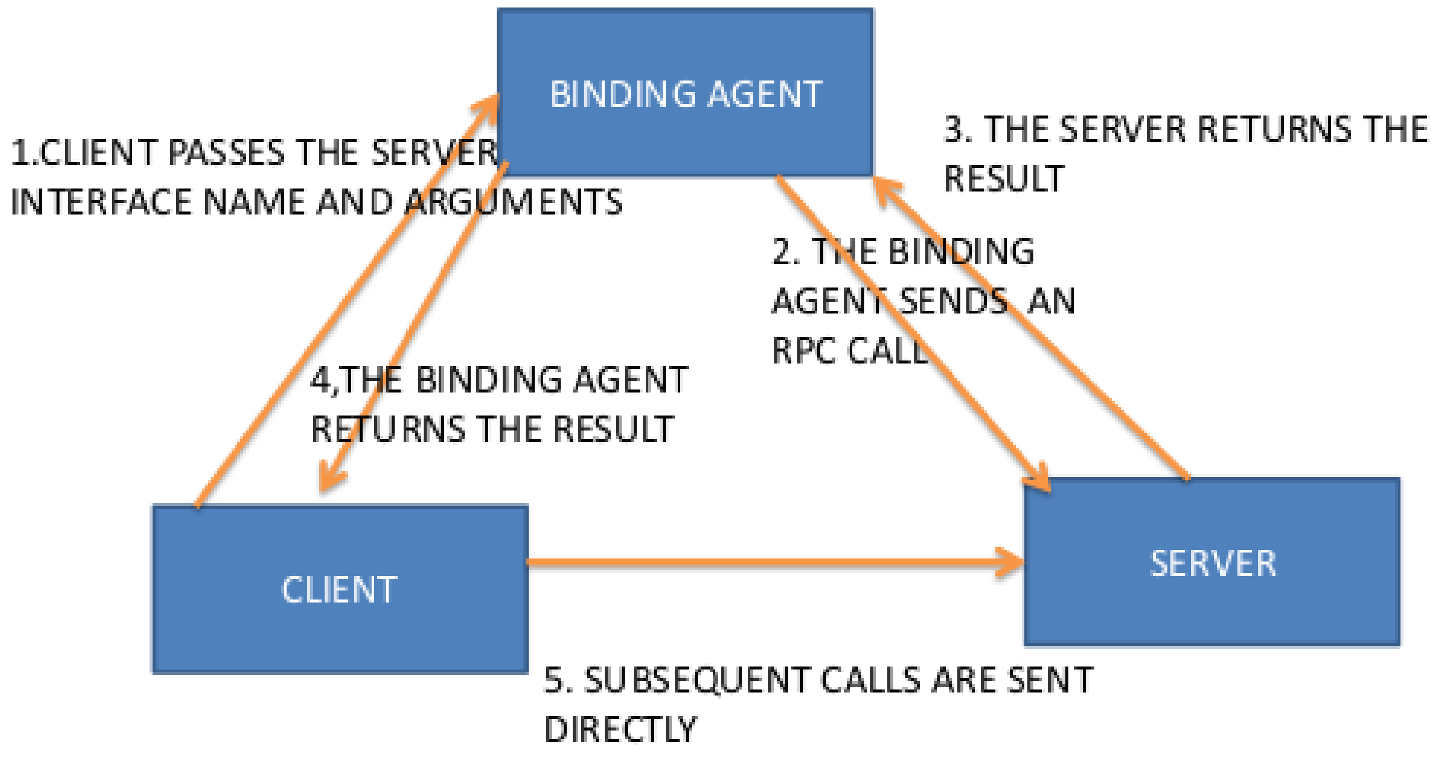
CLIENT-SERVER BINDING

- **Binding at Link Time**
- – A server process exports its service by registering itself with the binding agent as part of its initialization process.
- – A client then makes an import request to the binding agent for the service before making a call.
- – The binding agent binds the client and the server by returning to the client the server's handle

CLIENT-SERVER BINDING

- **Binding at Call Time**
- – A commonly used approach for binding at call time is the indirect call method,

CLIENT-SERVER BINDING


BINDING AGENT
1.CLIENT PASSES THE SERVER INTERFACE NAME AND ARGUMENTS
3. THE SERVER RETURNS THE RESULT
2. THE BINDING AGENT SENDS AN RPC CALL
4,THE BINDING AGENT RETURNS THE RESULT
CLIENT
SERVER
5. SUBSEQUENT CALLS ARE SENT DIRECTLY

CLIENT-SERVER BINDING

- **Changing Bindings**
- • when a file server has to be replaced with a new one, either it must be replaced when no files are open or the state of all the open files must be transferred from the old server to the new one
- • **Multilple Simultaneous Bindings**
- There may be situations when it is advantageous for a client to be bound

RPC IN HETEROGENEOUS ENVIRONMENTS

- The **three** common types of heterogeneity
- • Data representation. Machines having different architectures may use different data representations.
- • Transport protocol. For better portability of applications, an RPC system must be independent of the underlying network transport protocol
- • Control protocol. For better portability of applications, an RPC system must also be independent of control information in each transport packet to track the state of a call.