Remote Procedure Call

RPC

- Design issues
- Implementation
- RPC programming

Introduction

- Remote Procedure Call (RPC) is a high-level model for client-sever communication.
- It provides the programmers with a familiar mechanism for building distributed systems.
- Examples: File service, Authentication service.

Introduction

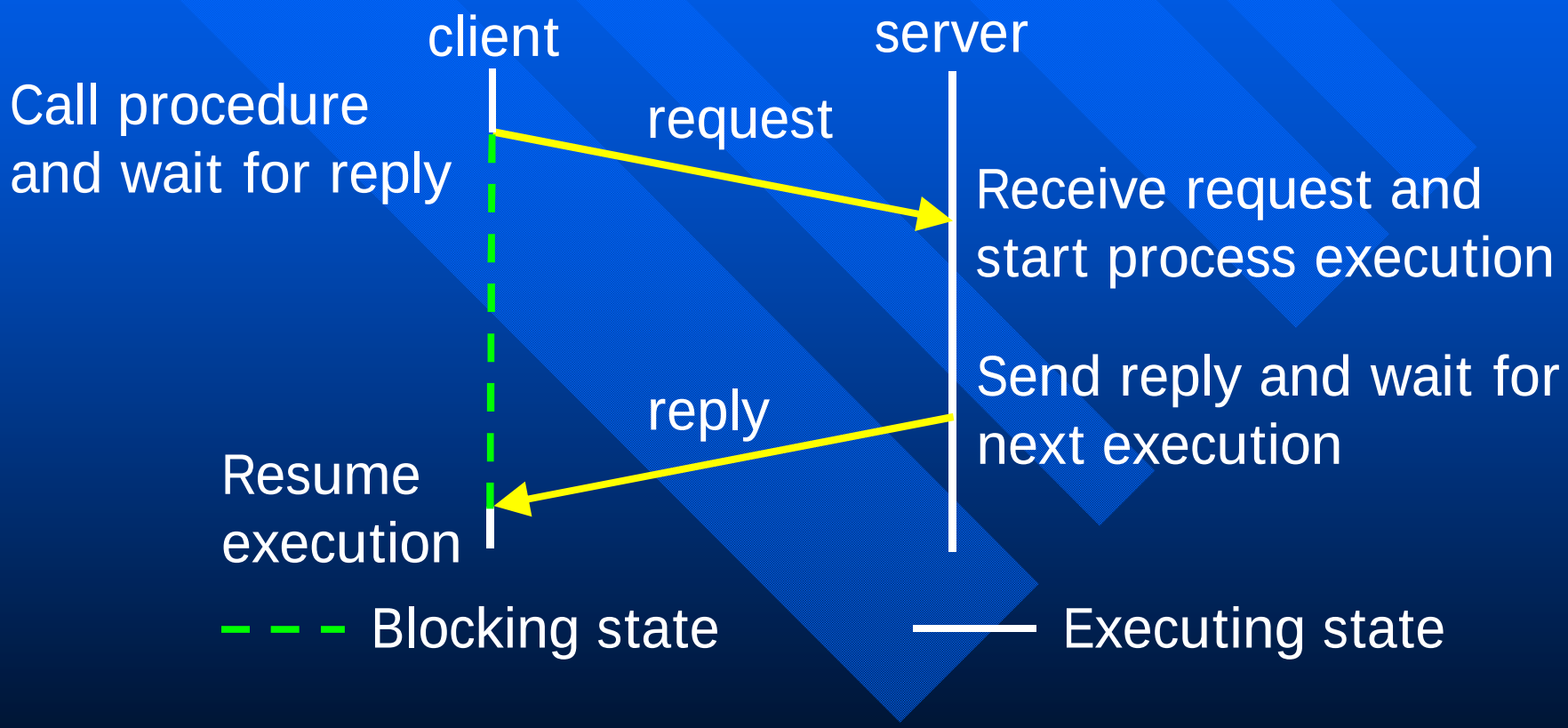
- Why we need Remote Procedure Call (RPC)?
  - The client needs a easy way to call the procedures of the server to get some services.
  - RPC enables clients to communicate with servers by *calling procedures in a similar way* to the conventional use of procedure calls in high-level languages.
  - RPC is modelled on the local procedure call, but the called procedure is executed in a different process and usually a different computer.

Introduction

- How to operate RPC?
  - When a process on machine A calls a procedure on machine B, the calling process on A is suspended, and the execution of the called procedure takes place on B.
  - Information can be transported from the caller to the callee in the parameters and can come back in the procedure result.
  - No message passing or I/O at all is visible to the programmer.

Introduction

- The RPC model



Characteristics

- The called procedure is in another process which may reside in another machine.
- The processes do not share address space.
  - Passing of parameters by reference and passing pointer values are not allowed.
  - Parameters are passed by values.
- The called remote procedure executes within the environment of the server process.
  - The called procedure does not have access to the calling procedure's environment.

Features

- Simple call syntax
- Familiar semantics
- Well defined interface
- Ease of use
- Efficient
- Can communicate between processes on the same machine or different machines

Limitations

- Parameters passed by values only and pointer values are not allowed.
- Speed: remote procedure calling (and return) time (i.e., overheads) can be significantly (1 - 3 orders of magnitude) slower than that for local procedure.
  - This may affect real-time design and the programmer should be aware of its impact.

Limitations

- Failure: RPC is more vulnerable to failure (since it involves communication system, another machine and another process).
  - The programmer should be aware of the call semantics, i.e. programs that make use of RPC must have the capability of handling errors that cannot occur in local procedure calls.

Design Issues

- Exception handling
  - Necessary because of possibility of network and nodes failures;
  - RPC uses return value to indicate errors;
- Transparency
  - Syntactic $\rightarrow$ achievable, exactly the same syntax as a local procedure call;
  - Semantic $\rightarrow$ impossible because of RPC limitation: failure (similar but not exactly the same);

Design Issues

- Delivery guarantees
  - Retry request message: whether to retransmit the request message until either a reply or the server is assumed to have failed;
  - Duplicate filtering : when retransmission are used, whether to filter out duplicates at the server;
  - Retransmission of replies: whether to keep a history of reply messages to enable lost replies to be retransmitted without re-executing the server operations.

Call Semantics

- Maybe call semantics
  - After a RPC time-out (or a client crashed and restarted), the client is not sure if the RP may or may not have been called.
  - This is the case when no fault tolerance is built into RPC mechanism.
  - Clearly, maybe semantics is not desirable.

Call Semantics

- At-least-once call semantics
  - With this call semantics, the client can assume that the RP is executed at least once (on return from the RP).
  - Can be implemented by retransmission of the (call) request message on time-out.
  - Acceptable only if the server's operations are idempotent. That is $f(x) = f(f(x))$.

Call Semantics

- At-most-once call semantics
  - When a RPC returns, it can assumed that the remote procedure (RP) has been called exactly once or not at all.
  - Implemented by the server's filtering of duplicate requests (which are caused by retransmissions due to IPC failure, slow or crashed server) and caching of replies (in reply history, refer to RRA protocol).

Call Semantics

- This ensure the RP is called exactly once if the server does not crash during execution of the RP.
- When the server crashes during the RP's execution, the partial execution may lead to erroneous results.
- In this case, we want the effect that the RP has not been executed at all.

Delivery Guarantee (Summary)

Retry request message	Duplicate filtering	Response	RPC call semantics
No	Not applicable	Not applicable	Maybe
Yes	No	Re-execute procedure	At-least-once
Yes	Yes	Retransmit reply	At-most-once

RPC Mechanism

- How does the client know the procedure (names) it can call and which parameters it should provide from the server?
- Server interface definition
  - RPC interface specifies those characteristics of the procedures provided by a server that are visible to the clients.
  - The characteristics includes: names of the procedures and type of parameters.
  - Each parameter is defined as input or output.

RPC Mechanism

- In summary, an interface contains a list of procedure signatures - the names and types of their I/O arguments (to be discussed later).
- This interface is made known to the clients through a server process binder (to be discussed later).

RPC Mechanism

- How does the client transfer its call request (the procedure name) and the arguments to the server via network?
- Marshalling and communication with server:
  - For each remote procedure call, a (client) stub procedure is generated and attached to the (client) program.
  - Replace the remote procedure call to a (local) call to the stub procedure.

RPC Mechanism

- The (codes in the) stub procedure marshals (the input) arguments and places them into a message together with the procedure identifier (of the remote procedure).
- Use IPC primitive to send the (call request) message to the server and wait the reply (call return) message (DoOperation).

RPC Mechanism

- How does the server react the request of the client? From which port? How to select the procedure? How to interpret the arguments?
- Despatching, Unmarshalling, communication with client:
  - A despatcher is provided. It receives the call request message from the client and uses the procedure identifier in the message to select one of the server stub procedures and passes on the arguments.

RPC Mechanism

- For each procedure at the server which is declared (at the sever interface) as callable remotely, a (server) stub procedure is generated.
- The task of a server stub procedure is to unmarshal the arguments, call the corresponding (local) service procedure.

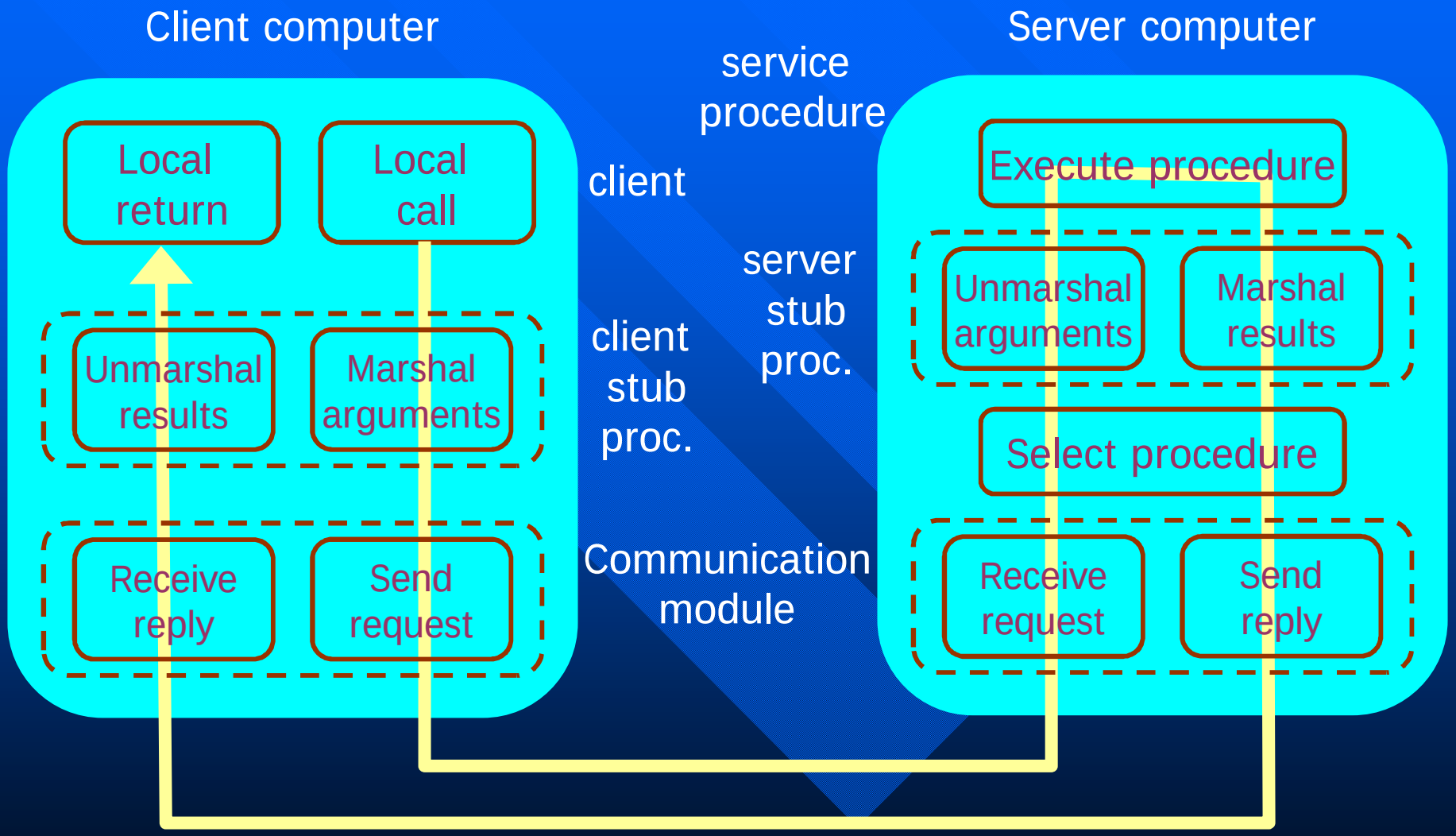
RPC Mechanism

- How does the server transmit the reply back?
- On return, the stub marshals the output arguments into a reply (call return) message and sends it back to the client.

RPC Mechanism

- How does the client receive the reply?
- The stub procedure of the client unmarshals the result arguments and returns (local call return). Note that the original remote procedure call was transformed into a (local) call to the stub procedure.

RPC Mechanism



RPC Mechanism (Summary)

1. The client provides the arguments and calls the client stub in the normal way.
2. The client stub builds (marshals) a message (call request) and traps to OS & network kernel.
3. The kernel sends the message to the remote kernel.
4. The remote kernel receives the message and gives it to the server dispatcher.
5. The dispatcher selects the appropriate server stub.
6. The server stub unpacks (unmarshals) the parameters and call the corresponding server procedure.

RPC Mechanism (Summary)

7. The server procedure does the work and returns the result to the server stub.
8. The server stub packs (marshals) it in a message (call return) and traps it to OS & network kernel.
9. The remote (receiver) kernel sends the message to the client kernel.
10. The client kernel gives the message to the client stub.
11. The client stub unpacks (unmarshals) the result and returns to client.

RPC Implementation

- Three main tasks:
  - Interface processing: integrate the RPC mechanism with client and server programs in conventional programming languages.
  - Communication handling: transmitting and receiving request and reply messages.
  - Binding: locating an appropriate server for a particular service.

RPC Implementation

- Interface Processing
  - Marshalling and unmarshalling of arguments;
  - Dispatching of request messages to the appropriate procedure in the server;
  - Interface compiler processes interface definitions written in an interface definition language (msg.x);
  - Generate a client stub procedure (msg_clnt.c);
  - Generate a server stub procedure (msg_svc.c);

RPC Implementation

- Use the signatures of the procedures in the interface to generate marshalling and unmarshalling operations (msg_xdr.c);
- Generate procedure headings for each procedure in the service from the interface definition (msg.h);

- Communication handling
  - TCP, UDP communication
  - Socket programming

RPC Implementation

- Binding
  - It specifies a mapping from a name to a particular object usually identified by a communication ID.
  - Binding is important because
    - » An interface definition specifies a textual service name for use by clients and servers.
    - » Clients that request message must be addressed to a server port.

RPC Implementation

- Binder: a separate service that maintains a table containing mappings from service names to server ports.
- All other services depend on the binder service.
- Binder interface used by server
  - » Register (String serviceName, Port serverPort, int version)
  - » Withdraw (String serviceName, Port serverPort, int version)
- Binder interface used by client
  - » PortLookUp (String serviceName, int version)

Case Studies: SUN RPC

- Interface definition language: XDR
  - a standard way of encoding data in a portable fashion between different systems;
- Interface compiler: rpcgen
  - A compiler that takes the definition of a remote procedure interface, and generates the client stubs and the server stubs;
- Communication handling: TCP or UDP
- Version: RPCSRC 3.9 (4.3BSD UNIX)
  - A run-time library to handle all the details.

Application

- Remote Procedure Call (RPC) is a powerful technique for constructing distributed, client/server based applications.

Scope of Research

- Open Network Computing Using Remote Procedure Call
- Remote Procedure Call Using OSI