ONC RPC

OPEN NETWORK COMPUTING REMOTE PROCEDURE CALL

OVERVIEW OF ONC RPC, AN RPC TECHNOLOGY FOR UNIX BASED SYSTEMS

Peter R. Egli
INDIGOO.COM

Contents

1. What is RPC?

RPC is an extension of conventional procedure calls:

RPC allows a client application to call procedures in a different address space on the same or on a remote machine (= transfer of control and data to a different address space and process).

RPC as middleware for traditional client – server programming:

RPC extends sockets with remote procedure call semantics. RPC allows an application to call a remote procedure just as it were a local procedure (location transparency).

Different flavors of RPC:

1. SUN-RPC (= ONC RPC):

ONC RPC = Open Network Computing RPC. Initially developed by Sun Microsystems.
First widely adopted RPC implementation.
Standards: RFC4506 XDR (Data presentation), RFC1057 RPC protocol specification.

2. DCE/RPC:

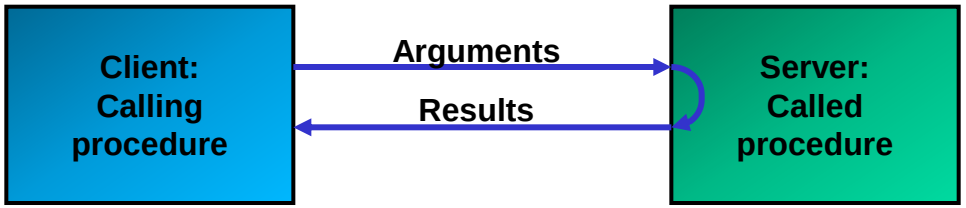
Distributed Computing Environment RPC by OSF (Open Software Foundation).
Not widely adopted by the IT industry.

3. MS RPC:

Microsoft's extension of DCE/RPC.

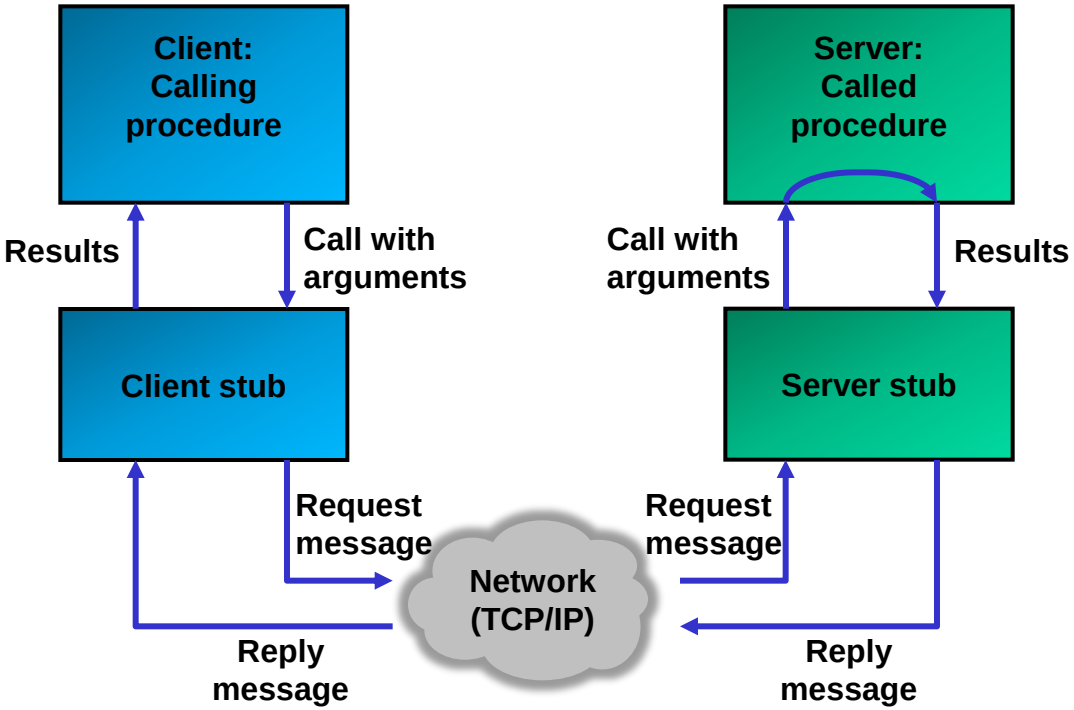
2. Local versus remote procedure call

Local procedure call:



The calling procedure executes the called procedure in its own address space and process.

Remote procedure call:



Client and server run as 2 separate processes (on the same or on different machines).

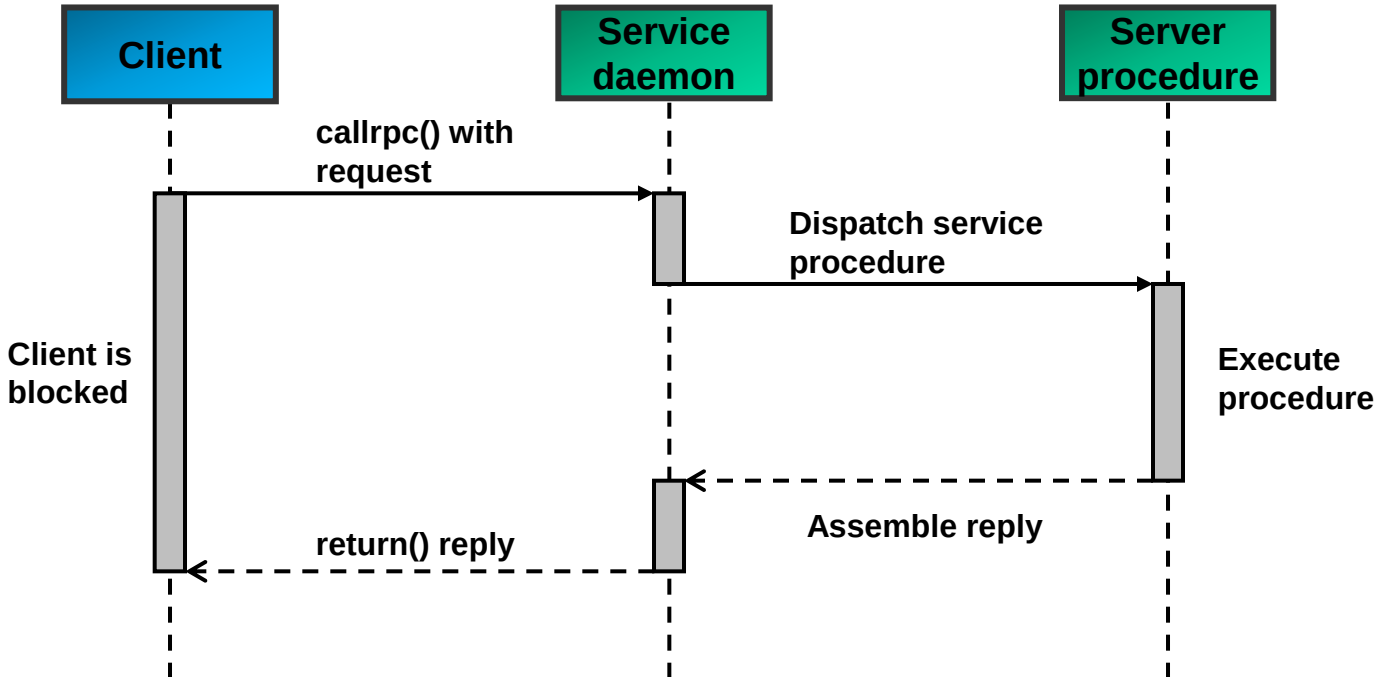
Stubs allow communication between client and server. These stubs map the local procedure call to a series of network RPC function calls.

3. RPC service daemon on the remote machine (portmapper)

How is the remote procedure called?

The remote procedure request message is received by a process (service daemon called portmapper). The daemon dispatches the call to the appropriate server stub.

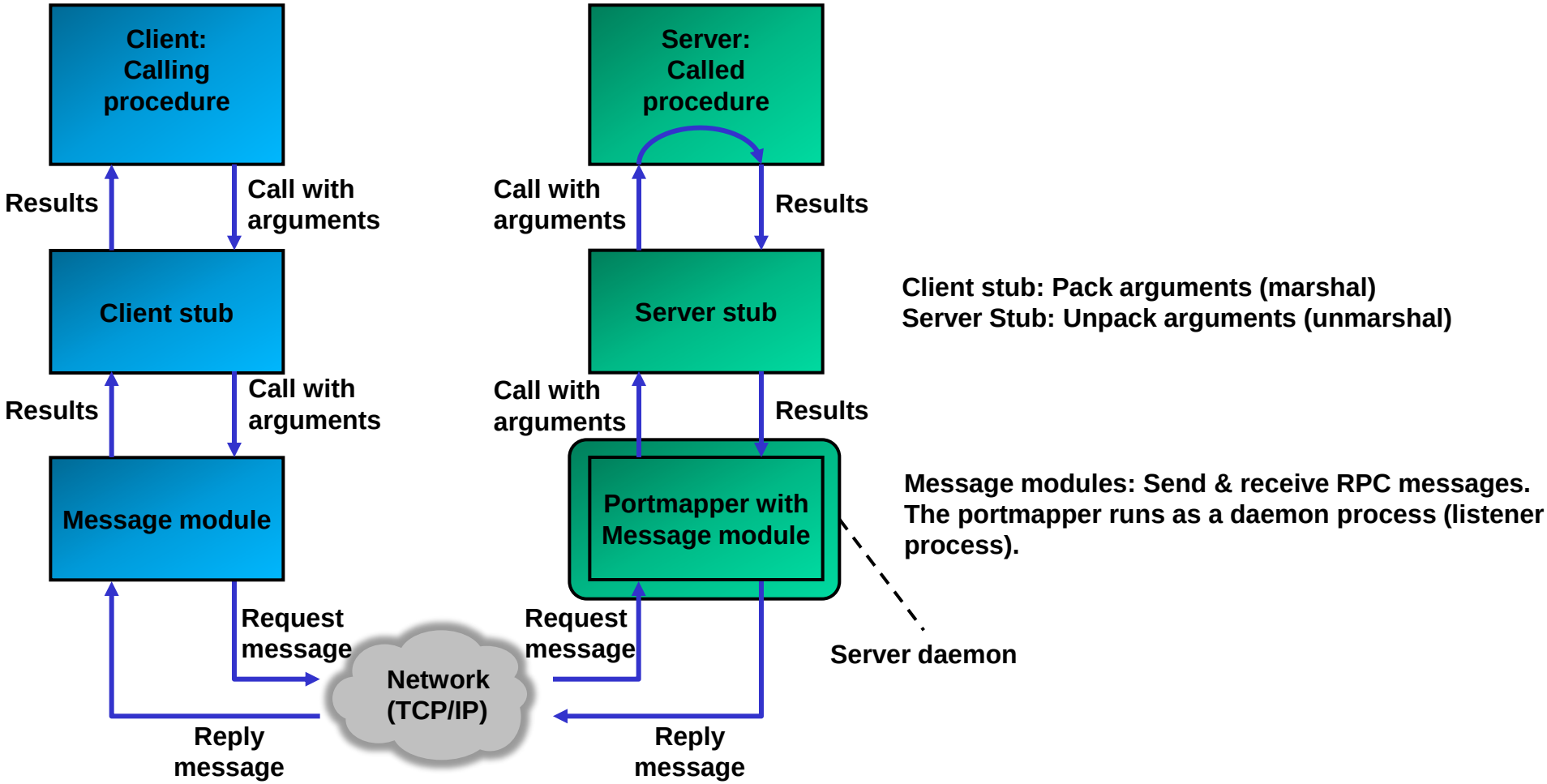
The service daemon listens on incoming requests.



4. RPC system components

Main RPC components:

The client stub is a proxy object that passes calls to the server stub. The message modules pack and unpack the marshaled arguments in UDP packets.



5. RPC parameter passing (1/2)

Local procedures:

The call of a local procedure allows pass-by-value and pass-by-reference.

Call by value example (plain old C):

```
int sqrt(int arg0)
{
  return arg0 * arg0;
}
...
int number = 9;
int result;
result = sqrt(number);
```

Call-by-reference example (plain old C):

```
struct Arguments {
  int number0;
  int number1;
} args;
struct Arguments args;
...
int product(Arguments* args)
{
  return args->number0 * args->number1;
}
...
int result = product(&args);
```

5. RPC parameter passing (2/2)

Remote procedures / RPC:

RPC only allows pass-by-value because pointers are meaningless in the address space of the remote process.

XDR (the RPC Interface Description Language, see below) allows declaring pointers to data structures. But RPC does actually copy-in/copy-out parameter passing (passes full copies of referenced data structures to the remote procedure).

Example:

XDR file with pointer argument:

```
/* msg.x: Remote msg printing protocol */
  struct PrintArgs {
    char*  name;
    int    number;
  };
program MSGPROG {
  version PRINTMSGVERS {
    int PRINTMSG(PrintArgs*) = 1;
  } = 1;
} = 0x20000001;
```

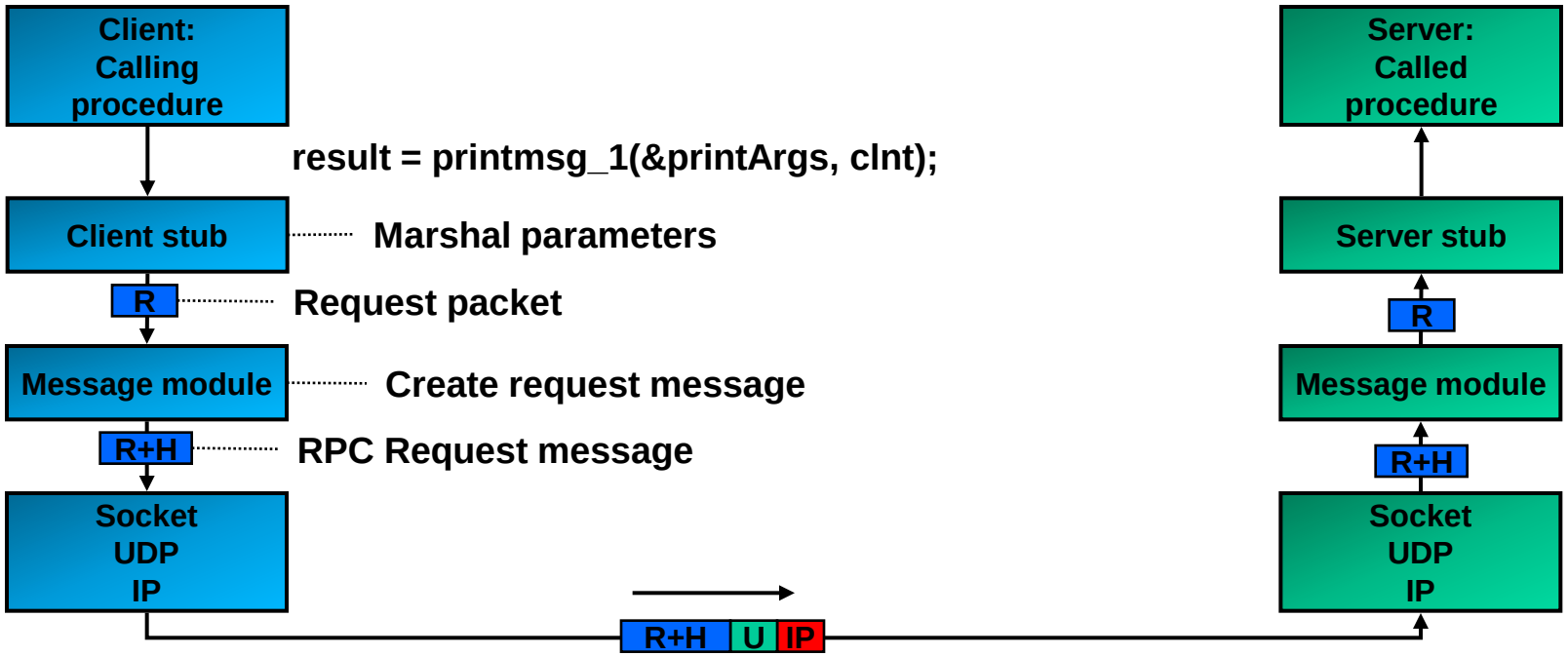
Client application:

```
struct PrintArgs printArgs;
int* result;
result = printmsg_1(&printArgs, clnt);
```

6. Network transport protocol for RPC

RPC uses the underlying network just as any other network application.
RPC may use connection oriented (TCP, SCTP) or connectionless transport protocols (UDP).

Usually RPC uses UDP because UDP (connectionless transport service) better supports the RPC protocol because:
1. RPC interactions are generally short so connections are not really required.
2. Servers generally serve a large number of clients and maintaining state information on connections means additional overhead for the server.
3. LANs are generally reliable so the reliable service of TCP is not needed.



7. RPC interface description and code generation (1/2)

Interface definition:

The server procedure(s) are defined as an interface in a formal language (XDR) and then compiled to source code (plain old C).

Elements of XDR (External Data Representation):

1. **Interface definition language (IDL).**
   XDR defines procedure signatures (procedure names, parameters with types) and user defined types.
2. **Data encoding standard (how to encode data for transmission over the network).**
   XDR defines how to pack simple types like int and string into a byte stream.

Example XDR file:

```
/* msg.x: Remote msg printing protocol */
program MESSAGEPROG
{
   version PRINTMESSAGEVERS
   {
      int PRINTMESSAGE(string) = 1;
   } = 1;
} = 0x20000001;
```

Name of the (remote) program

Version of (remote) program (=1 in the example)

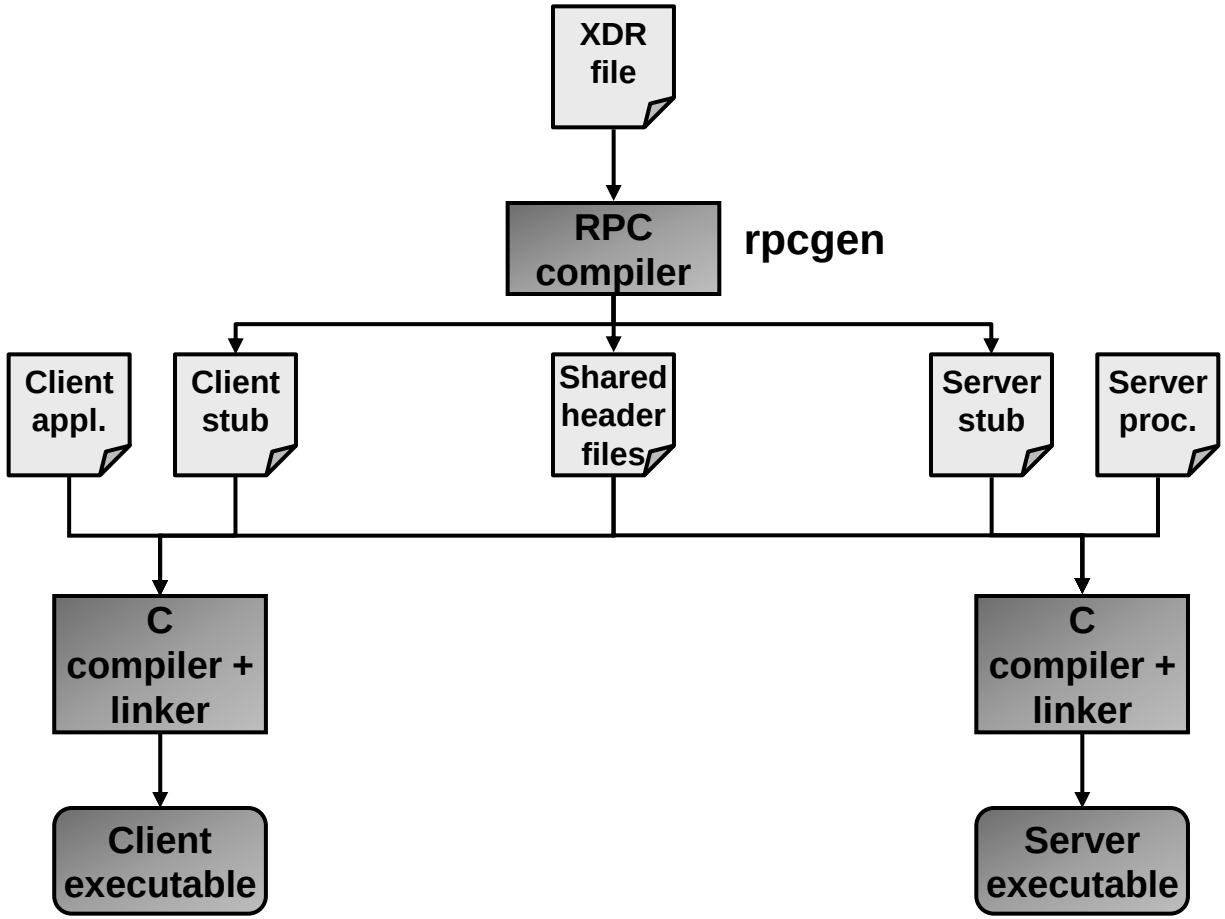
Procedure declaration (procedure number 1)

Program number

7. RPC interface description and code generation (2/2)

Code generation from XDR file:

The RPC compiler (ONC RPC: rpcgen) generates all the necessary C source files from the XDR interface specification file (header files, stubs).

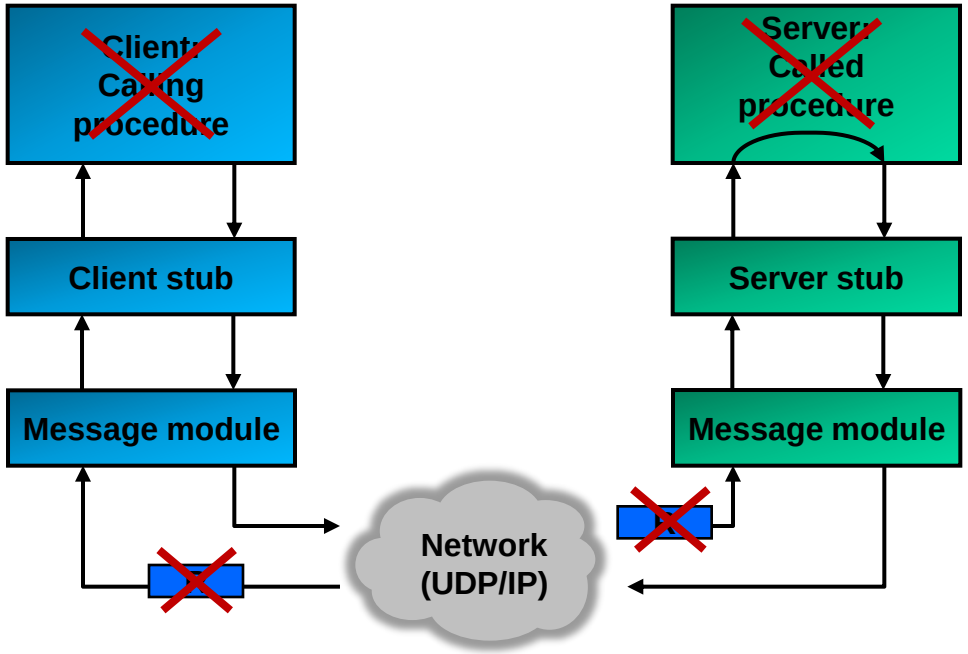


8. RPC procedure call semantics (1/4)

Potential problems with RPC calls:

a. The request message may be lost (in the network, in the message module).
b. The reply message may be lost.
c. The server or client may crash during an RPC call.

In these cases, it is not clear to the client if the remote procedure has been called or not. RPC incorporates different strategies for handling these situations.



8. RPC procedure call semantics (2/4)

RPC message delivery strategies:

1. Retry request message:

The client retransmits the request message until either a reply is received or the server is assumed to have failed (timeout).

2. Duplicate filtering:

The server filters out duplicate request messages when retransmissions are used.

3. Retransmission of replies:

The server maintains a history of reply messages.
In case of a lost reply message, the server retransmit the reply without re-executing the server operation.

8. RPC procedure call semantics (3/4)

RPC supports 3 different "call semantics" that define the level of guarantee of requests:

1. **maybe call semantics.**
2. **at-least-once call semantics.**
3. **at-most-once call semantics.**

RPC mechanisms usually include timeouts to prevent clients waiting indefinitely for reply messages.

1. maybe call semantics:

- **No retransmission of request messages.**
- **Client is not certain whether the procedure has been executed or not.**
- **No fault-tolerance measures (RPC call may or may not be successful).**
- **Generally not acceptable (low level of guarantee).**

2. at-least-once call semantics:

- **The message module repeatedly resends request messages after timeouts occur until it either gets a reply message or a maximum number of retries have been made.**
- **No duplicate request message filtering.**
- **The remote procedure is called at least once if the server is not down.**
- **The client does not know how many times the remote procedure has been called. This could produce undesirable results (e.g., money transfer) unless the called procedure is "idempotent" (= repeatable with the same effect as a single call).**

8. RPC procedure call semantics (4/4)

3. at-most-once call semantics:

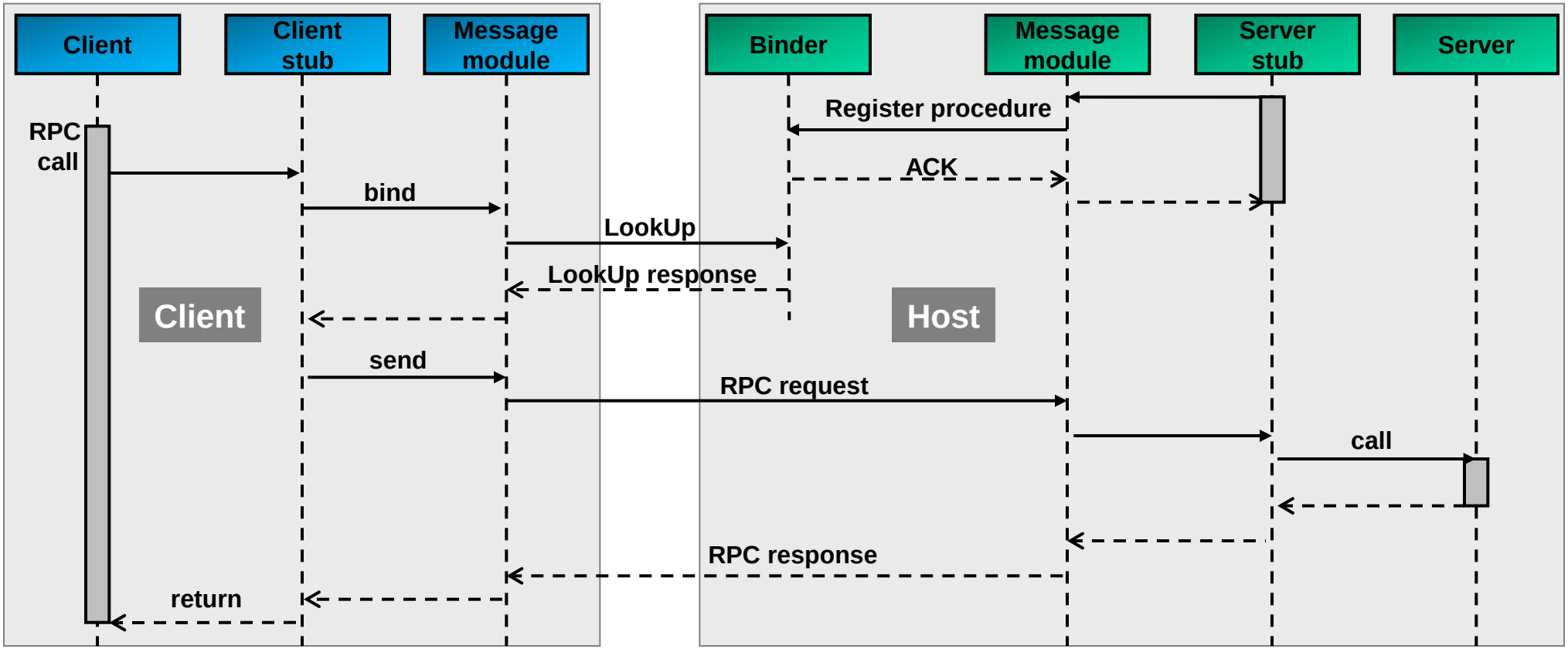
- Retransmission of request messages.
- Duplicate request message filtering.
- If the server does not crash and the client receives the result of a call, then the procedure has been called exactly once. Otherwise, an exception is reported and the procedure will have been called either once or not at all.
- Works for both idempotent and non-idempotent operations.
- More complex support required due to the need for request message filtering and for keeping track of replies.

Comparison of call semantics:

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

9. RPC remote procedure location and discovery (1/2)

How does the client find the IP address and port of the host hosting the remote procedure?
In RPC, *binding* refers to determining the location and identity of the called procedure.
The binder (RPCBIND, formerly portmapper) is a daemon that:
- is a registry for registering server procedures on the server,
- is a lookup service for the client to get the address of the server procedure host,
- runs on the host and listens on a well-known address (port number 111).



9. RPC remote procedure location and discovery (2/2)

Static versus dynamic binding:

Static binding:

- Binds the host address of a server into the client program at compilation time.
- Undesirable because the client and server programs are compiled separately and often at different times and the server may be moved from one host to another.

Dynamic binding:

- Dynamic lookup of the server address (see previous slide).
- Allows servers to register and remove their exporting services.
- Allows clients to lookup the named service.

Binding API:

```
PROCEDURE Register (serviceName:String; serverPort:Port; version:integer)
```

Binder records the service name and server port of a service in its table, together with a version number.

```
PROCEDURE Withdraw (serviceName:String; serverPort:Port; version:integer)
```

Binder removes the service from its table.

```
PROCEDURE LookUp (serviceName:String; version:integer): Port
```

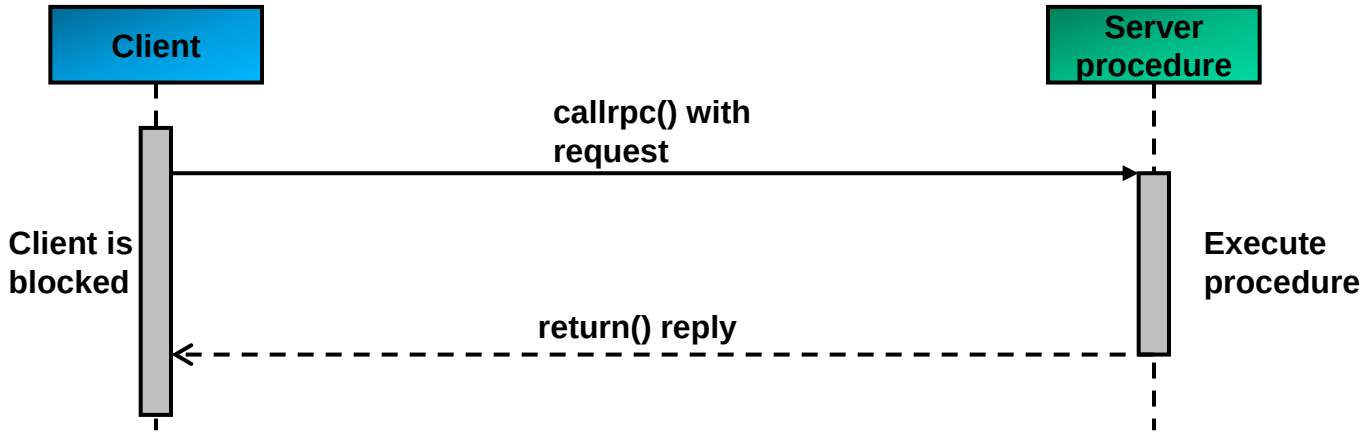
The binder looks up the named service and returns its address (or set of addresses) if the version number agrees with the one stored in its table.

10. RPC interaction (1/3)

RPC allows synchronous and asynchronous interaction.

Synchronous RPC:

In synchronous RPC, the client is blocked until it received the server reply.
Synchronous RPC is the default mode of interaction.
🙂 Easier to program (no synchronization required on client).
☹ Client is blocked during the entire RPC operation (waste of execution time, performance negatively affected).



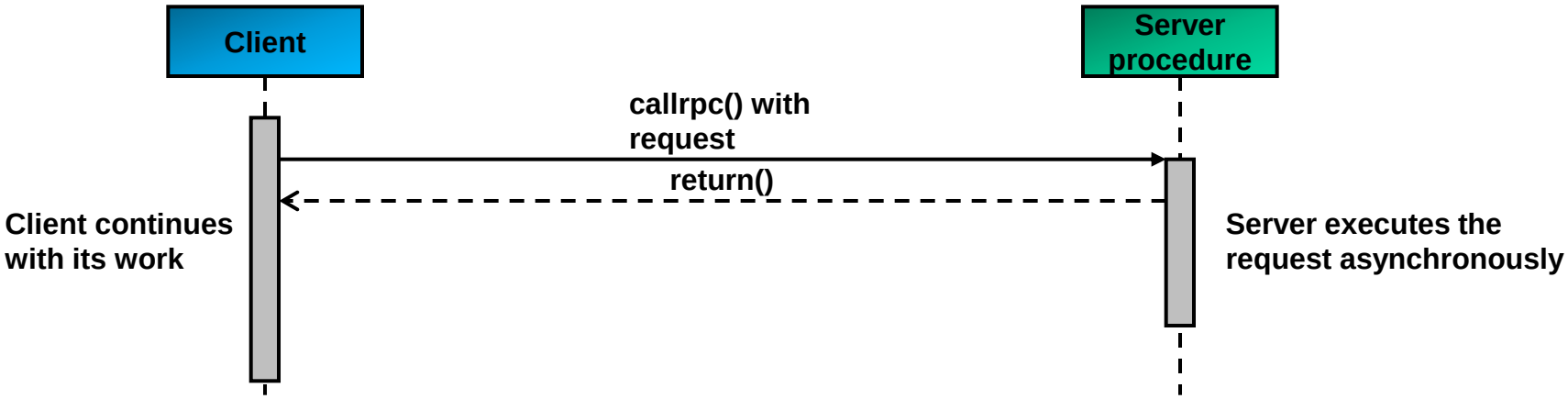
10. RPC interaction (2/3)

Asynchronous RPC:

If a remote procedure does not require a response, the client can send a request to the server without waiting for a reply and continue with its work.

The client is not blocked, can continue with its work.

Only possible for remote procedures that do not return a result (e.g. send a log message to a log server).



10. RPC interaction (3/3)

Deferred synchronous RPC:

With deferred synchronous RPC, the client sends a request to the server and only waits for the acceptance of the request. The server executes the request and sends an answer back to the client.

This mode is useful in cases where the server procedure requires lengthy computations.

Client not blocked, can continue with its work.

More complicated synchronization required on the client.