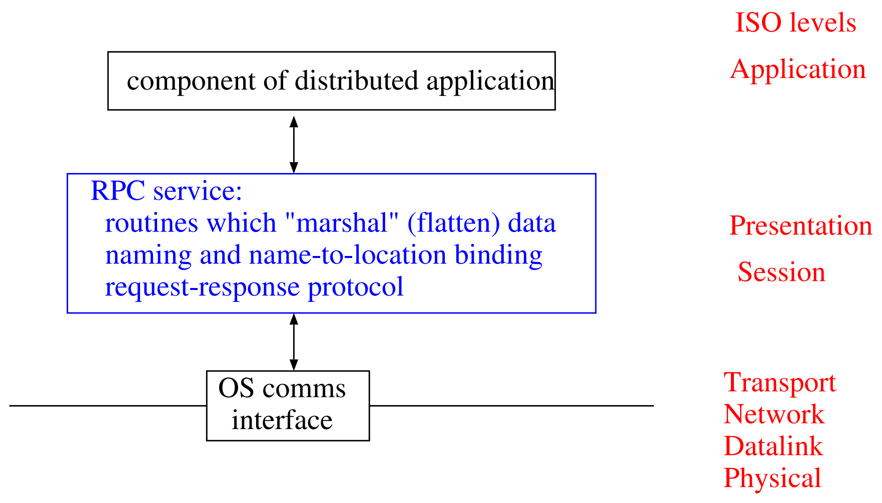
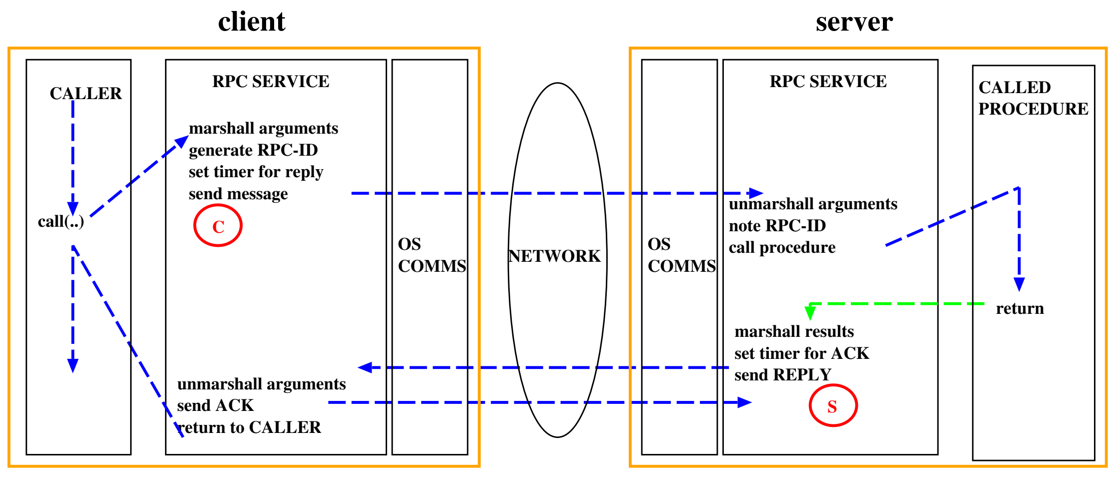
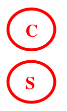
Remote Procedure Call (RPC)


ISO levels
Application
component of distributed application
RPC service:
routines which "marshal" (flatten) data
naming and name-to-location binding
request-response protocol
Presentation
Session
OS comms interface
Transport
Network
Datalink
Physical

examples: Mayflower/CCLU RPC, SUN RPC, ANSA RPC, MSRPC
Xerox Courier over XNS (SPP, Ethernet)
ISO-ODP, OSF DCE

RPC Request-Reply Acknowledge (RRA) protocol





client timer
server times

RPC semantics

recall that client, server and network may be congested or may fail independently of each other
(fundamental property of Distributed Systems)

RPC systems may offer AT MOST ONCE or EXACTLY ONCE semantics

C if the client timer expires:

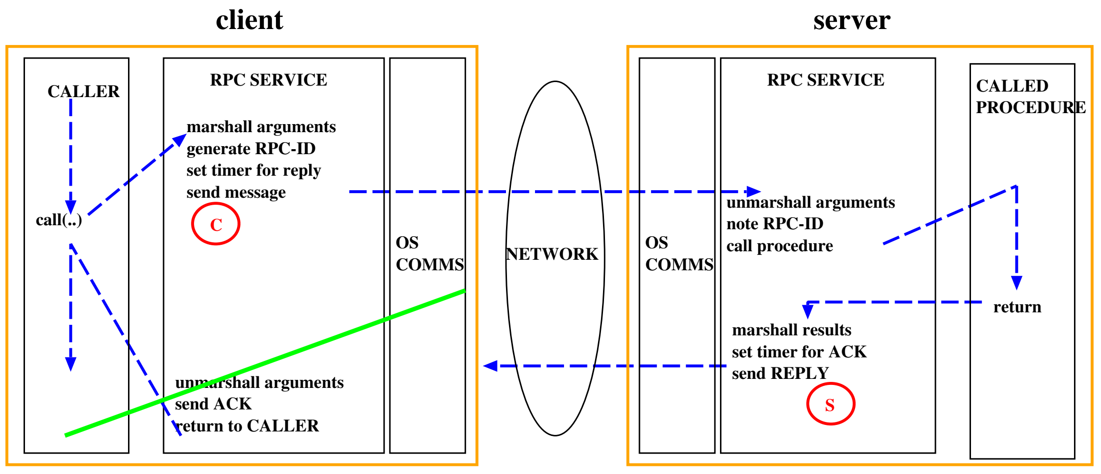
AT MOST ONCE semantics:
exception return to the application
it is likely to repeat the call but this is not detectable
i.e. it will have a new RPC-ID

EXACTLY ONCE semantics:
retry a few times
RPC-ID means that the server can detect repeats
if no reply, exeption return to client

S if the server timer expires:

resend results
RPC-ID means that the client can detect repeats

RPC client crash

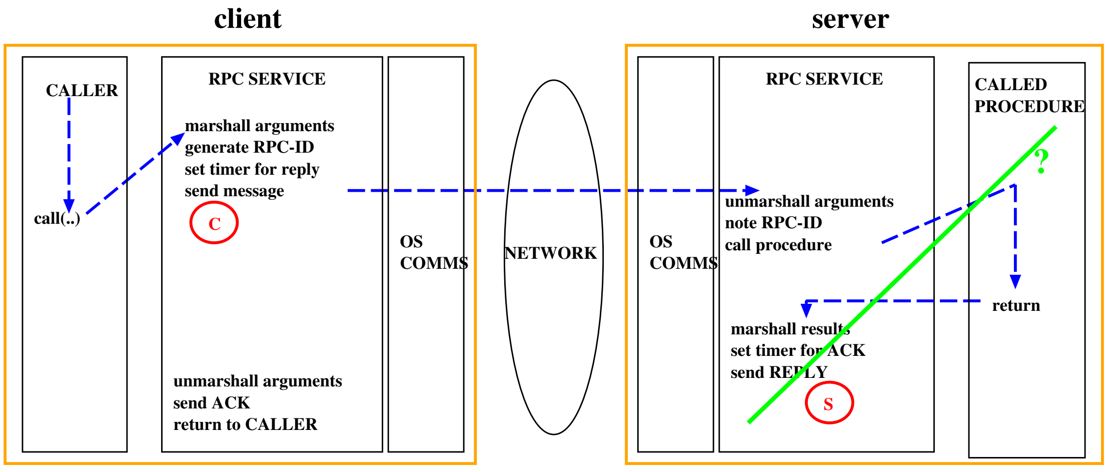


results are sent to crashed machine, are not acknowledged, and server timer S expires repeatedly on resend

persistent state may have been changed by the procedure call - should this be handled by RPC service?

NO - application-level transaction semantics (commit/abort) should be used.

RPC server crash



The server fails at some stage during the call. Results are not sent and the client timer C expires repeatedly

persistent state may or may not have been changed by the procedure call - should this be handled by RPC service?

NO - application-level transaction semantics (commit/abort) should be used.

Integration of Programming Languages and RPC (1)

* some early RPC systems aimed for complete distribution transparency
e.g. Xeroc PARC, Mesa language, Courier RPC
a preprocessor detects which calls are not to local procedures
and replaces them by calls to RPC support

problem of incorrect procedure names that don't exist anywhere
problem of call semantics for some arguments

* Cambridge Mayflower system, CCLU RPC - made distribution explicit
the compiler was changed
different syntax for definition and call of procedures that can be called remotely
BUT - this was still for a single language, CCLU

some RPC systems restricted the argument types
e.g. SUN RPC: C base-types only

CCLU RPC: most types including procedure names defined since developer supplies
marshalling and unmarshalling routines for constructed types (recursive descent)

Integration of Programming Languages and RPC (2)

* ANSA RPC, was initially developed for C
but later also supported C++ and Modula3 - a very early heterogeneous system

- defined a Distributed Programming Language (DPL)
- DPL statements are embedded in the programming language, and tagged
- a preprocessor detects these statement, replaces them with calls to RPC service

All RPC systems automatically generate marshalling and unmarshalling routines to flatten
call and return arguments into packet format suitable for transmission, and unpack them on receipt.
These routines are programming-language-specific.

Now assume that we wish to support a number of different programming languages,
i.e. components written in different languages can interoperate

* the standard approach (ANSA, ISO-ODP, OSF-DCE), O-O platforms

- define an Interface Definition Language (IDL)
- provide mappings for programming language’s type systems onto IDL
- (internally) define the transfer syntax for IDL types
- IDL compilers generate marshalling and unmarshalling routines
  appropriate for the programming languages involved.

(CORBA calls the invoker’s marshalling routine a STUB
and the invoked object’s unmarshalling routine a SKELETON)

Integration of Programming Languages and Middleware

* how do platforms that support objects and object invocation differ from the RPC schemes described above?

(as above for IDL and STUB/SKELETON generation)

RPC systems name and identify interfaces and procedures

e.g. ANSA IDL has base and constructed data types and the InterfaceRef type,
an instance of which is a reference to a loaded and running instance of a service’s interface

O-O systems name and invoke objects

Externally invocable objects must be registered with the platform,
an object-ID is returned (and may be recorded in a name service)
The object becomes known globally and may be invoked remotely
Object-IDs are first-class values which may be passed as arguments