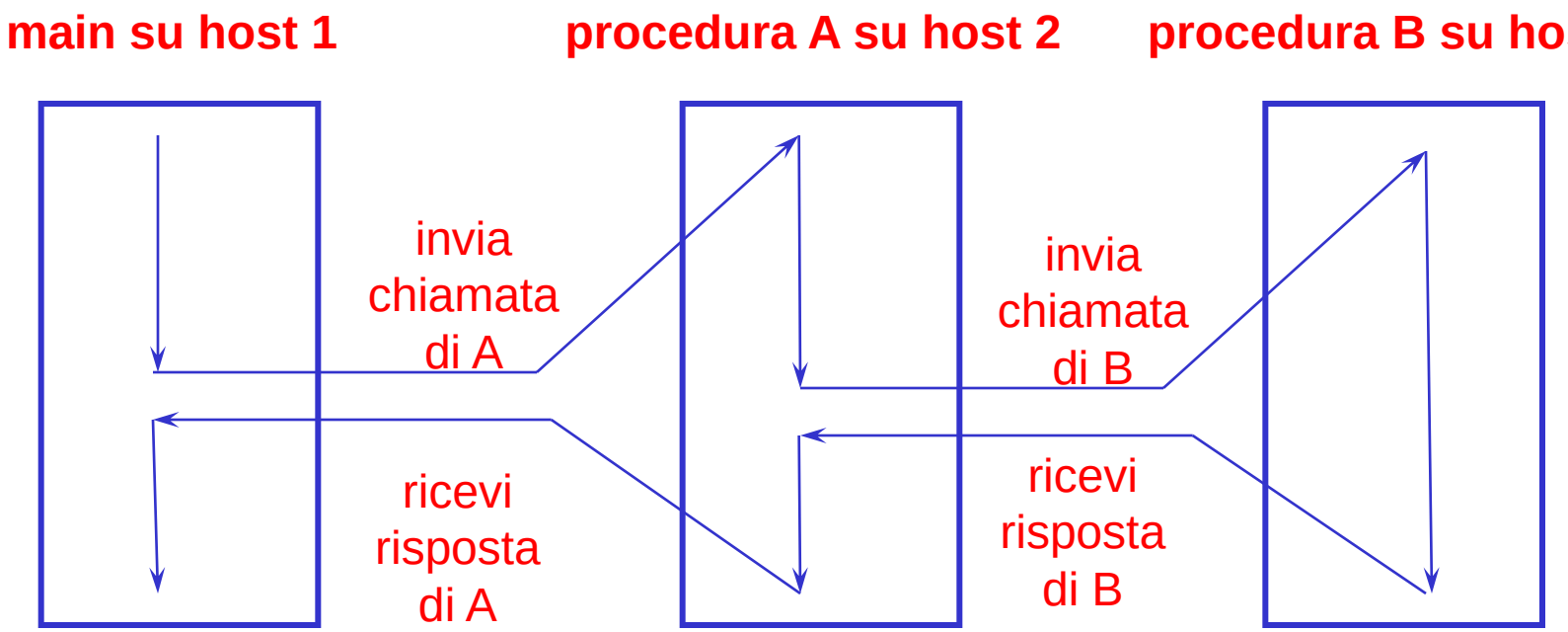
Remote Procedure Call (RPC)

- Concetto generale
- È la trasposizione del meccanismo di chiamata di procedura (locale) ad un ambiente **distribuito:**



Particolarità delle procedure remote

- Chiamante e chiamato girano su **host diversi**
  - servono meccanismi di *localizzazione* del server, prima della chiamata (linking dinamico)
  - occorre *sincronizzare* chiamate multiple in arrivo sul server
  - in caso di loop infinito o crash parziale, occorrono meccanismi di *timeout e recovery*
  - Chiamante e chiamato hanno spazi di indirizzi disgiunti
    - Il passaggio dei parametri per riferimento non viene normalmente consentito

Particolarità delle procedure remote

- Chiamante e chiamato sono **eterogenei** (diversi linguaggi, piattaforme hw/sw, ecc.)
  - parametri e valore di ritorno devono essere convertiti (*Marshaling – ordinamento/smistamento*)
- Devono essere trattati i problemi di *security* (in particolare quelli di autenticazione)
- Una chiamata remota è più lenta di una chiamata locale (di ordini di grandezza)

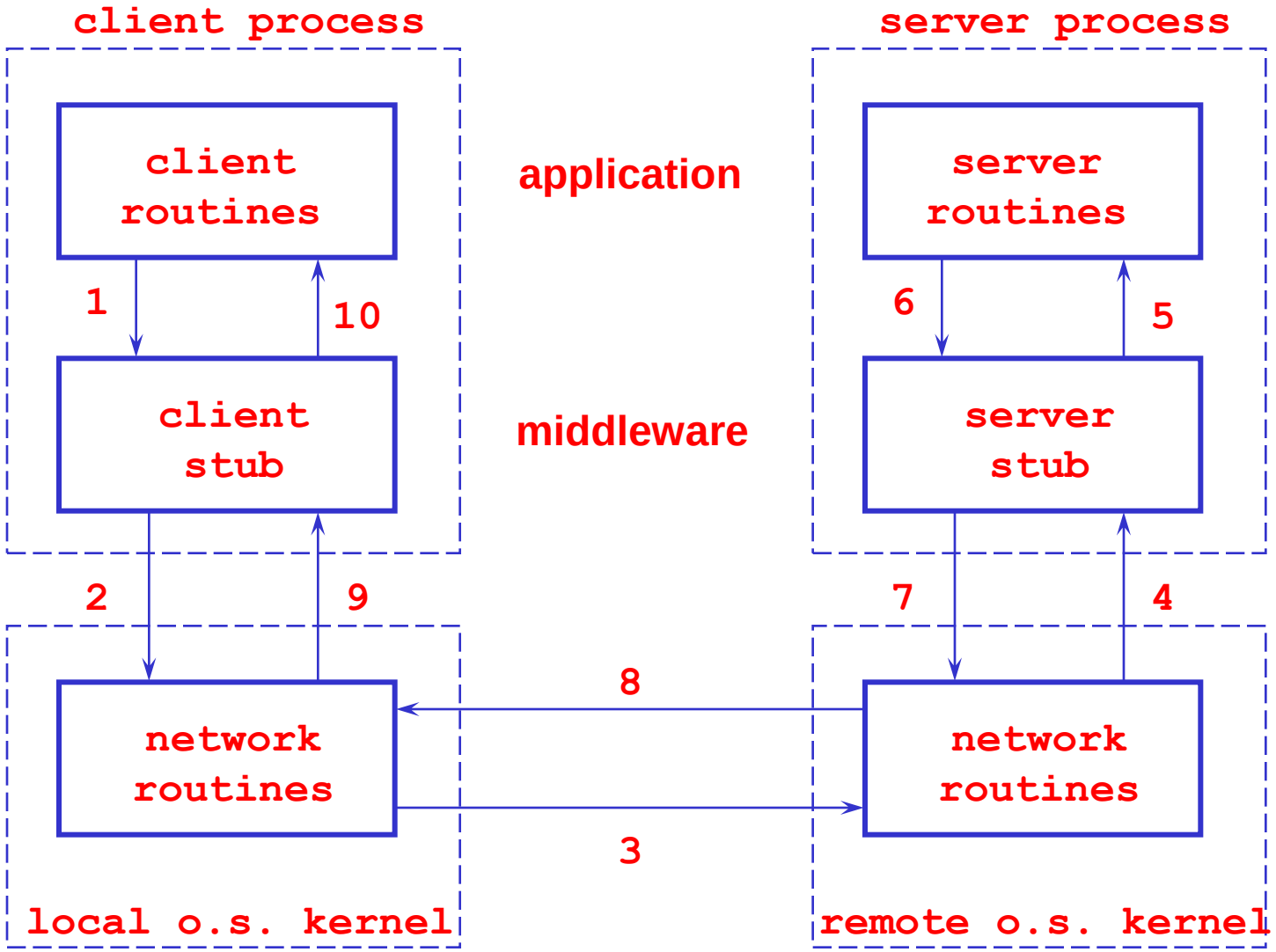
Fallimenti e semantica delle chiamate remote

- Una chiamata remota può fallire (crash del server, crash o inaffidabilità della rete) o essere eseguita più volte (duplicazione dei messaggi)

  ⇒ il **middleware** deve gestire queste situazioni

- A seconda del protocollo RPC usato esistono diverse semantiche (per ritorno «successo» e/o «failure»):
  - **exactly once** : la procedura è stata eseguita esattamente una volta (protocollo sofisticato, incide sui tempi di esecuzione).
  - **at most once** : la procedura non è stata eseguita più di una volta (numeri di sequenza delle chiamate).
  - **at least once** : la procedura è stata eseguita una o più volte (timeout e ritrasmissioni).

Il modello RPC



Confronto sviluppo socket / RPC

- I socket forniscono solo servizi a livello 4 (trasporto), tutta l'implementazione del protocollo applicativo è lasciata al programmatore
  - Codifica dei dati, procedure di interazione
  - Enfasi sulla comunicazione più che sulla logica applicativa
  - Generalmente utilizzati per scrivere un proprio protocollo applicativo
- RPC permette di sviluppare come se il codice fosse centralizzato (ma necessario considerare che l'interazione sarà distribuita e le funzioni possono fallire)
  - Enfasi sulla logica applicativa piuttosto che sulle comunicazioni

Esempi di implementazioni RPC

- SUN Microsystems ONC (Open Network Computing) (molto diffuso in ambiente Unix, linguaggio C)
  - Molto spesso riferito anche semplicemente come RPC
- JAVA RMI (linguaggio Java)
- CORBA (indipendente da linguaggio)
- Web services (indipendente da linguaggio)

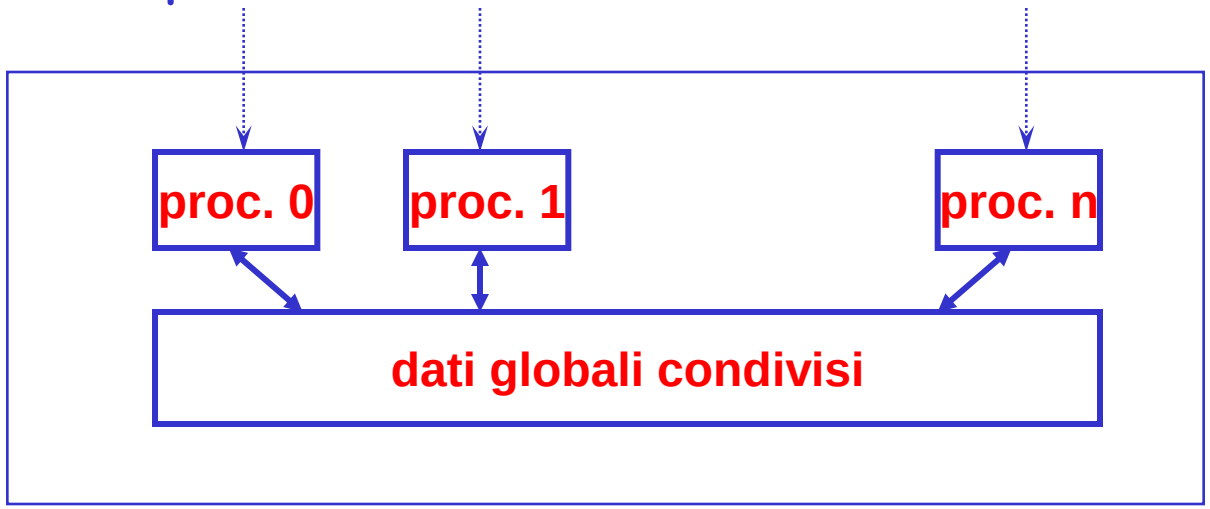
SUN RPC (ONC)

- E' il meccanismo RPC più noto e diffuso
- *trasporto*: rete TCP/IP (può utilizzare TCP oppure UDP)
- *rappresentazione dei dati*: **XDR**
- *generazione automatica degli stub* (**rpcgen**)

- Documentazione:
  - Rfc 1057 (Protocollo RPC)
  - Rfc 1014 (linguaggio XDR)

Programmi RPC

- Un **Programma RPC** è
  - un insieme di procedure accessibili tramite un server
  - che operano su dati condivisi



Identificazione di Programmi e Procedure

- Ogni programma SUN RPC viene identificato *univocamente* tramite un intero su 4 byte:
  - **0x00000000 - 0x1fffffff** assegnati dalla SUN
  - **0x20000000 - 0x3fffffff** definibili dall'utente
  - **0x40000000 - 0x5fffffff** per uso temporaneo
  - **0x60000000 - 0xffffffff** riservati per usi futuri
- Un programma può essere disponibile in più **versioni** (numerate a partire da 1)
- Per ogni versione è disponibile un diverso insieme di **procedure** (anche queste numerate)
- procedura 0 : è sempre la *"procedura di test"*

Identificazione di Programmi e Procedure

- La procedura da chiamare viene dunque identificata tramite la terna:

  (programma, versione, procedura)

- È inoltre necessario identificare, tra tutti i server che rendono disponibile la procedura richiesta, quello su cui la si vuole eseguire.

Esempi di numeri di programma pre-assegnati da SUN

port mapper	100000
rstat	100001
remote users	100002
nfs	100003
yp (NIS)	100004
mount	100005
DBX	100006
yp binder	100007
rwall	100008
yppasswd	100009
ethernet statistics	100010

Parametri e valore di ritorno

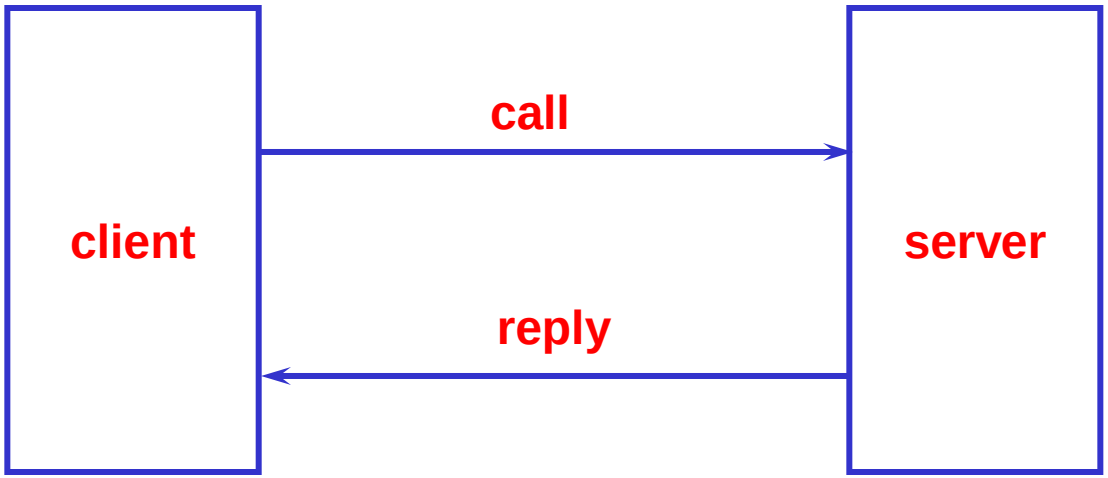
- Vengono rappresentati e trasferiti in formato XDR
- Per semplicità viene ammesso un unico parametro.
  - se si vogliono più parametri occorre definirli come campi di una struttura

Autenticazione

- Il protocollo prevede la possibilità di scegliere tra diversi meccanismi di autenticazione:
  - nessuna autenticazione
  - autenticazione UNIX-like
  - autenticazione basata su protocollo crittografico (DES, Kerberos, ecc.)

I Messaggi del Protocollo RPC

- Il protocollo prevede una chiamata (call) ed una risposta (reply):



Semantica delle chiamate

- Il protocollo SUN RPC non prevede meccanismi per garantire l'affidabilità: essa dipende dal trasporto usato
- Il middleware SUN RPC implementa una semplice politica di ritrasmissione
  - con timeout e numero di ritrasmissioni non adattativi (l'utente può impostarli per una data applicazione)
  - raggiunto il numero massimo di ritrasmissioni, la chiamata viene abortita (fallisce)

NB: il fallimento **non** significa che la procedura non sia stata eseguita

Semantica delle chiamate

- In pratica, l'applicazione
  - deve essere consapevole del trasporto usato
  - può trarre solo le conclusioni più conservative compatibili con le caratteristiche del trasporto
- Esempio: usando il trasporto UDP:
  - se la chiamata ha successo: *at least once*
  - se la chiamata fallisce: *zero or more*
- Una strategia comunemente usata è quella di rendere le procedure **indempotent**. Esempio:

  appendi X al file Y => scrivi X in posizione k nel file Y

Concorrenza e Sincronizzazione

- La sincronizzazione viene ottenuta serializzando le chiamate:
  - ogni server non può eseguire più di una chiamata per volta
- La serializzazione impatta negativamente sulla scalabilità
  - Soprattutto se il rate di richieste è elevato rispetto al tempo di esecuzione della procedura

Localizzazione del Server

- Per localizzare un server occorre conoscere IP address, protocollo (TCP/UDP) e numero di porta.
- Problema: i numeri di porta sono limitati
  - non è possibile un'assegnazione statica porta-programma
  - viene usato un mapping dinamico ed un **port-mapper** (server che gestisce il database delle corrispondenze porta-programma su un dato host)

Struttura Messaggio (XDR)

```
struct rpc_msg {
        unsigned int xid;
        union switch (msg_type mtype) {
        case CALL:
                call_body cbody;
        case REPLY:
                reply_body rbody;
        } body;
};

enum msg_type {
        CALL = 0,
        REPLY = 1
};
```

Corpo del messaggio

```
struct call_body {
        unsigned int rpcvers;/* must be equal to 2 */
        unsigned int prog;
        unsigned int vers;
        unsigned int proc;
        opaque_auth cred;
        opaque_auth verf;
        /* procedure specific parameters start here */
};
```

Campi di Autenticazione

```
enum auth_flavor {
        AUTH_NULL = 0,
        AUTH_UNIX = 1,
        AUTH_SHORT = 2,
        AUTH_DES = 3
        /* and more to be defined */
};

struct opaque_auth {
        auth_flavor flavor;
        opaque body<400>;
};
```

Corpo di Risposta

```
union reply_body switch (reply_stat stat) {
        case MSG_ACCEPTED:
                accepted_reply areply;
        case MSG_DENIED:
                rejected_reply rreply;
} reply;

enum reply_stat {
        MSG_ACCEPTED = 0,
        MSG_DENIED = 1
};
```

Risposta Accettata

```
struct accepted_reply {
        opaque_auth verf;
        union switch (accept_stat stat) {
        case SUCCESS:
                opaque results[0];
        /*procedure-specific results start here*/
        case PROG_MISMATCH:
                struct {
                        unsigned int low;
                        unsigned int high;
                } mismatch_info;
        default:
        /* Void. Cases include PROG_UNAVAIL,
PROC_UNAVAIL, and GARBAGE_ARGS.
*/
                void;
        } reply_data;
};
```

Stati di Accettazione

```
enum accept_stat {
        SUCCESS = 0, /* RPC executed successfully */
        PROG_UNAVAIL = 1, /* remote hasn't exported program */
        PROG_MISMATCH = 2, /* remote can't support version */
        PROC_UNAVAIL = 3, /* program can't support procedure */
        GARBAGE_ARGS = 4 /* procedure can't decode params */
};
```

Risposta Rifiutata

```
union rejected_reply switch (reject_stat stat) {
        case RPC_MISMATCH:
                struct {
                        unsigned int low;
                        unsigned int high;
                } mismatch_info;
        case AUTH_ERROR:
                auth_stat stat;
};

enum reject_stat {
        RPC_MISMATCH=0, /* RPC vers. numb. != 2 */
        AUTH_ERROR=1 /* remote can't authenticate caller */
};
```

Port Mapper

- È un server SUN RPC, presente su ogni host, che usa la porta riservata 111 (assegnata staticamente)
- Un server SUN RPC che rende disponibile un programma deve *registrarsi* presso il port mapper locale, ottenendo da esso un numero di porta
- Un client SUN RPC che voglia attivare una procedura remota,
  - interroga il port mapper dell'host per conoscere il numero di porta su cui è disponibile il programma desiderato
  - contatta il server al numero di porta così conosciuto

Generazione degli Stub

- Gli stub possono essere generati
  - manualmente, facendo uso di una libreria in linguaggio C
  - automaticamente dal programma **rpcgen**
- L'input di rpcgen è un file di specifica, che descrive un programma SUN RPC (la sua interfaccia) in un apposito linguaggio basato su XDR

Libreria RPC

- Fornisce una serie di funzioni per:
  - eseguire le chiamate RPC
  - registrare servizi sul port mapper
- Gli stub si appoggiano a queste funzioni per realizzare il meccanismo di chiamata in modo trasparente.
- È possibile intervenire sugli stub generati automaticamente per modificare certi parametri (p. es. numero massimo di tentativi).