Remote Procedure Calls

- Objective:allow development of client/server application, hiding session or transport layer.
- Commercial situation
  - Free from Sun
  - Netwise:major provider of RPC on PCs and UNIX.
  - Novell, has licensed it for use in Netware.

Sun RPC

- Client
- XDR
- Server
- Portmapper
- Language

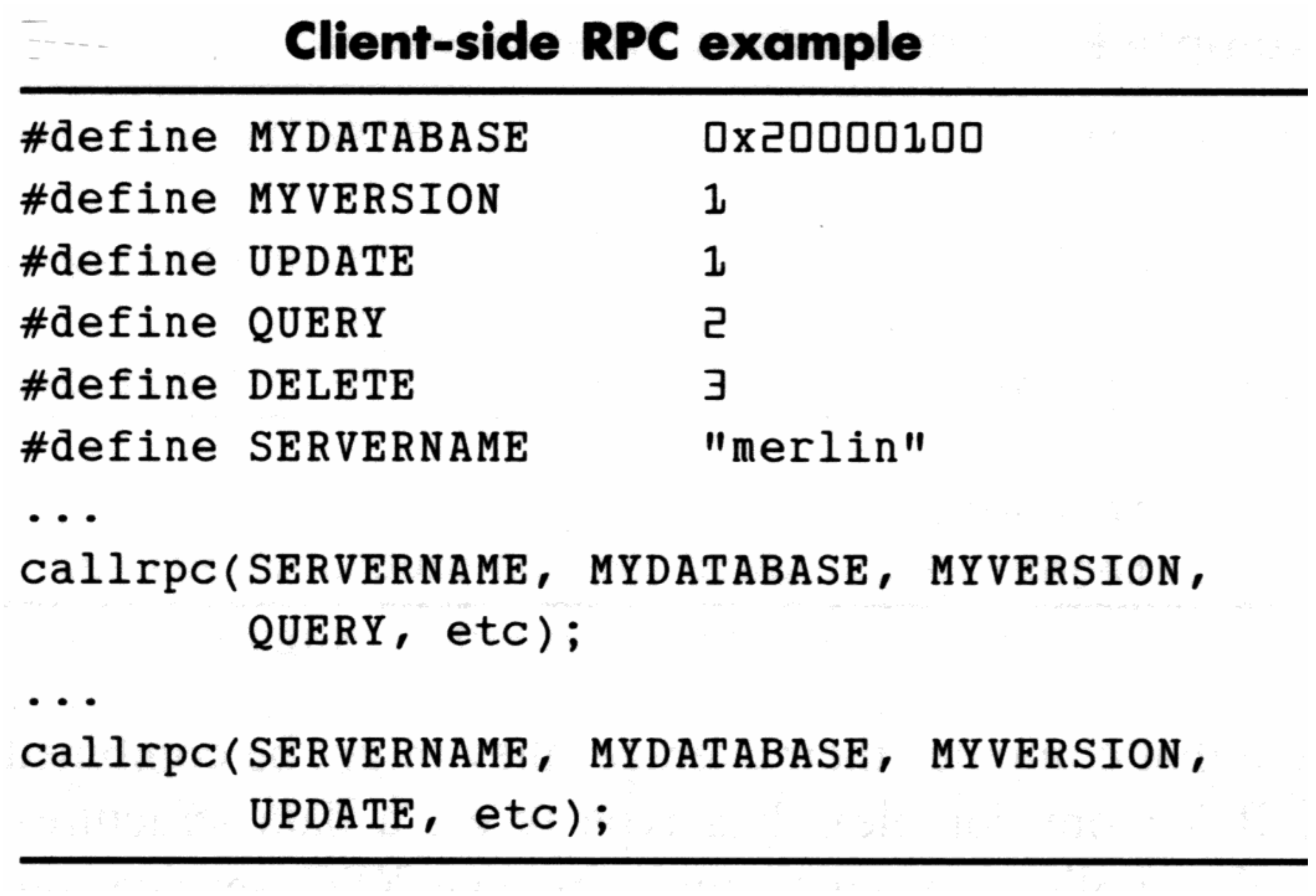
RPC Client

Simply call `callrpc (...):`

Parameters:

- The name of the remote node
- The name of the program called
  - In fact, this is a number known by client/server and assigned by administrator.
- The version number of the program called
  - So, new and old version can coexist.
- The procedure to be invoked
  - In fact, this is a number too, assigned by application programmer.
- The type of parameter (sent to server)
- The parameter
- The type of returning parameter
- The returning parameter

Example



Client-side RPC example

```
#define MYDATABASE      0x20000100
#define MYVERSION       1
#define UPDATE          1
#define QUERY           2
#define DELETE          3
#define SERVERNAME      "merlin"
...
callrpc(SERVERNAME, MYDATABASE, MYVERSION,
        QUERY, etc);
...
callrpc(SERVERNAME, MYDATABASE, MYVERSION,
        UPDATE, etc);
```

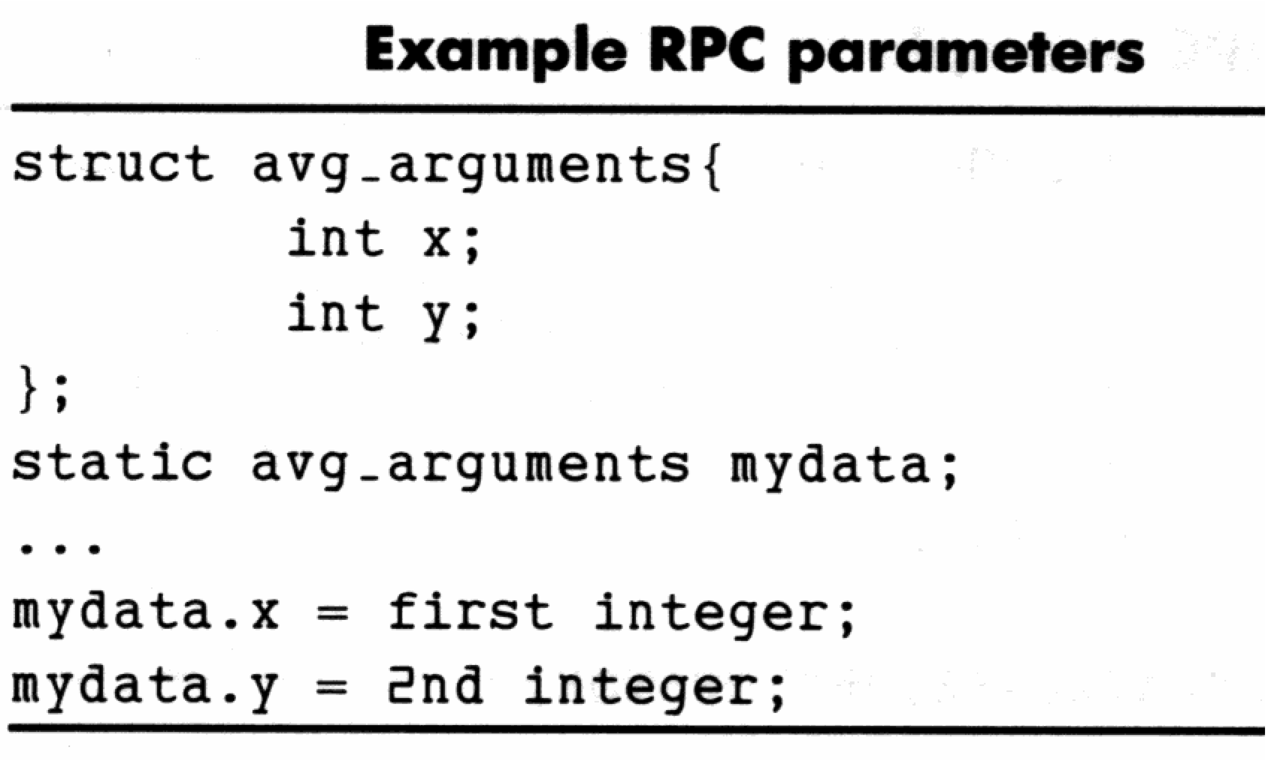
External Data Representation (XDR)

- Different machines may have different data formats, XDR is used to help interpret parameters.
- Example, `xdr_int, xdr_float`, etc.

```
callrpc(SERVNAME, SPROG, VERS, PROC, xdr_string,
  name, xdr_long, address)
```

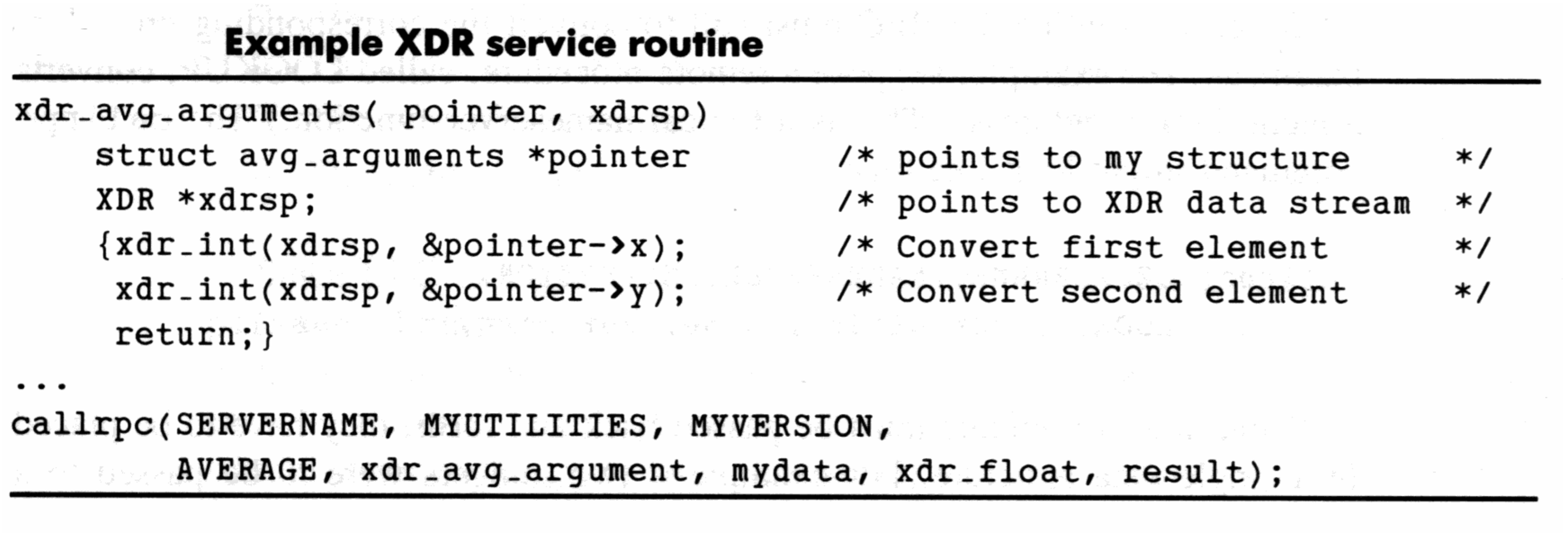
Complex Data Type

- What if we want to send more than two data?



Example RPC parameters

```
struct avg_arguments{
        int x;
        int y;
};
static avg_arguments mydata;
...
mydata.x = first integer;
mydata.y = 2nd integer;
```



Example XDR service routine

```
xdr_avg_arguments( pointer, xdrsp)
    struct avg_arguments *pointer       /* points to my structure    */
    XDR *xdrsp;                         /* points to XDR data stream */
    {xdr_int(xdrsp, &pointer->x);       /* Convert first element     */
     xdr_int(xdrsp, &pointer->y);       /* Convert second element    */
     return;}
...
callrpc(SERVERNAME, MYUTILITIES, MYVERSION,
        AVERAGE, xdr_avg_argument, mydata, xdr_float, result);
```

RPC Server

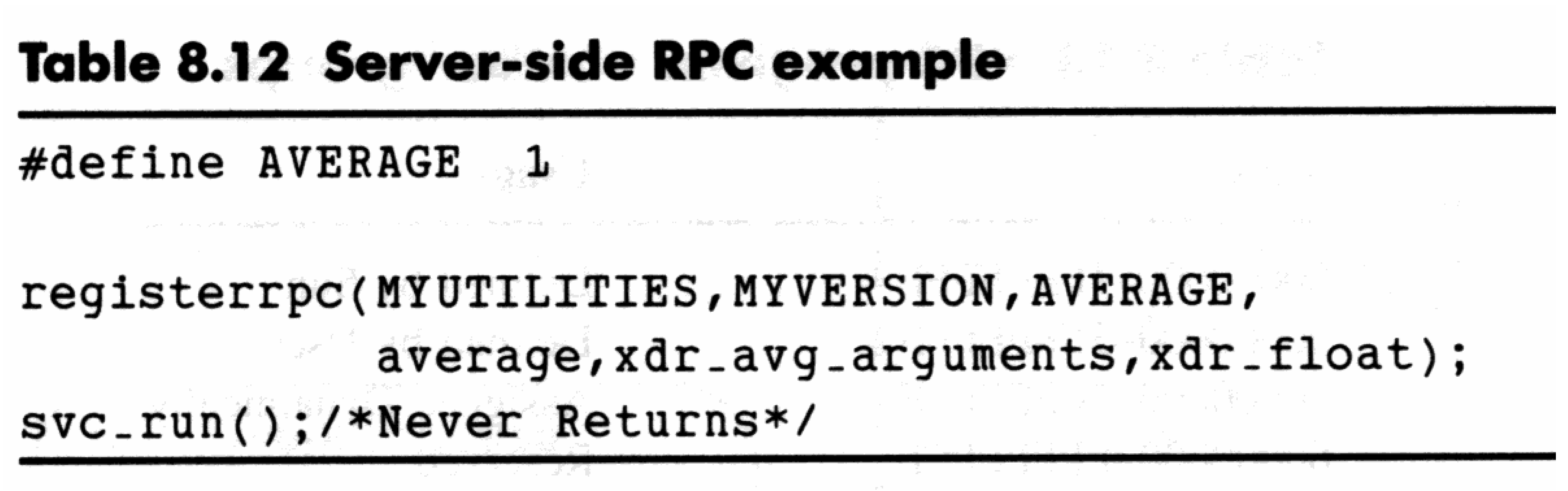


Table 8.12 Server-side RPC example

```
#define AVERAGE  1

registerrpc(MYUTILITIES,MYVERSION,AVERAGE,
            average,xdr_avg_arguments,xdr_float);
svc_run();/*Never Returns*/
```

Register and SVC_run

- Register each remote procedure using `resigterrpc(...).` Parameters are:
  - The program number to register as
  - A version number
  - The procedure number
  - The name of the procedure to call
  - The XDR service routine for parameters
  - The XDR service routine for returning data.
- Then invoke `svc_run(),`
  - Wait for RPC request.
  - Call the appropriate procedure when one arrives.

Example

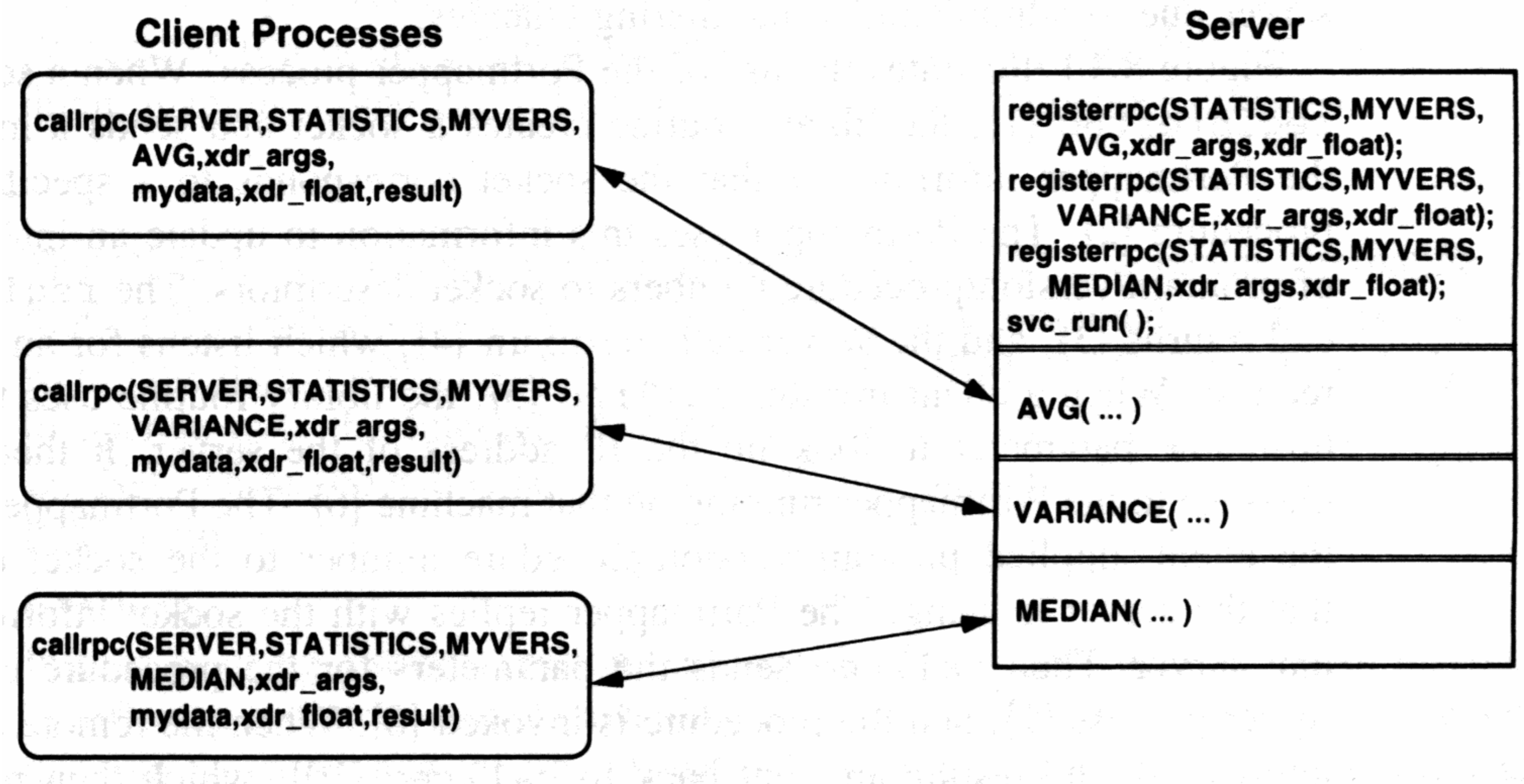
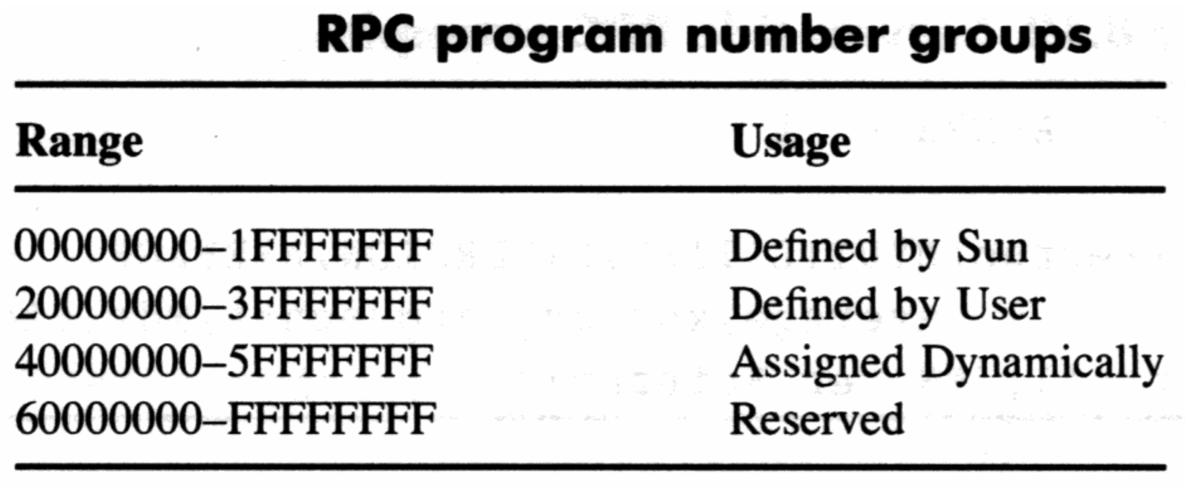

Client Processes
Server
callrpc(SERVER,STATISTICS,MYVERS, AVG,xdr_args, mydata,xdr_float,result)
callrpc(SERVER,STATISTICS,MYVERS, VARIANCE,xdr_args, mydata,xdr_float,result)
callrpc(SERVER,STATISTICS,MYVERS, MEDIAN,xdr_args, mydata,xdr_float,result)
registerrpc(STATISTICS,MYVERS, AVG,xdr_args,xdr_float);
registerrpc(STATISTICS,MYVERS, VARIANCE,xdr_args,xdr_float);
registerrpc(STATISTICS,MYVERS, MEDIAN,xdr_args,xdr_float);
svc_run( );
AVG(...)
VARIANCE(...)
MEDIAN(...)

Figure 8.13 Sun RPC example.

Portmapper

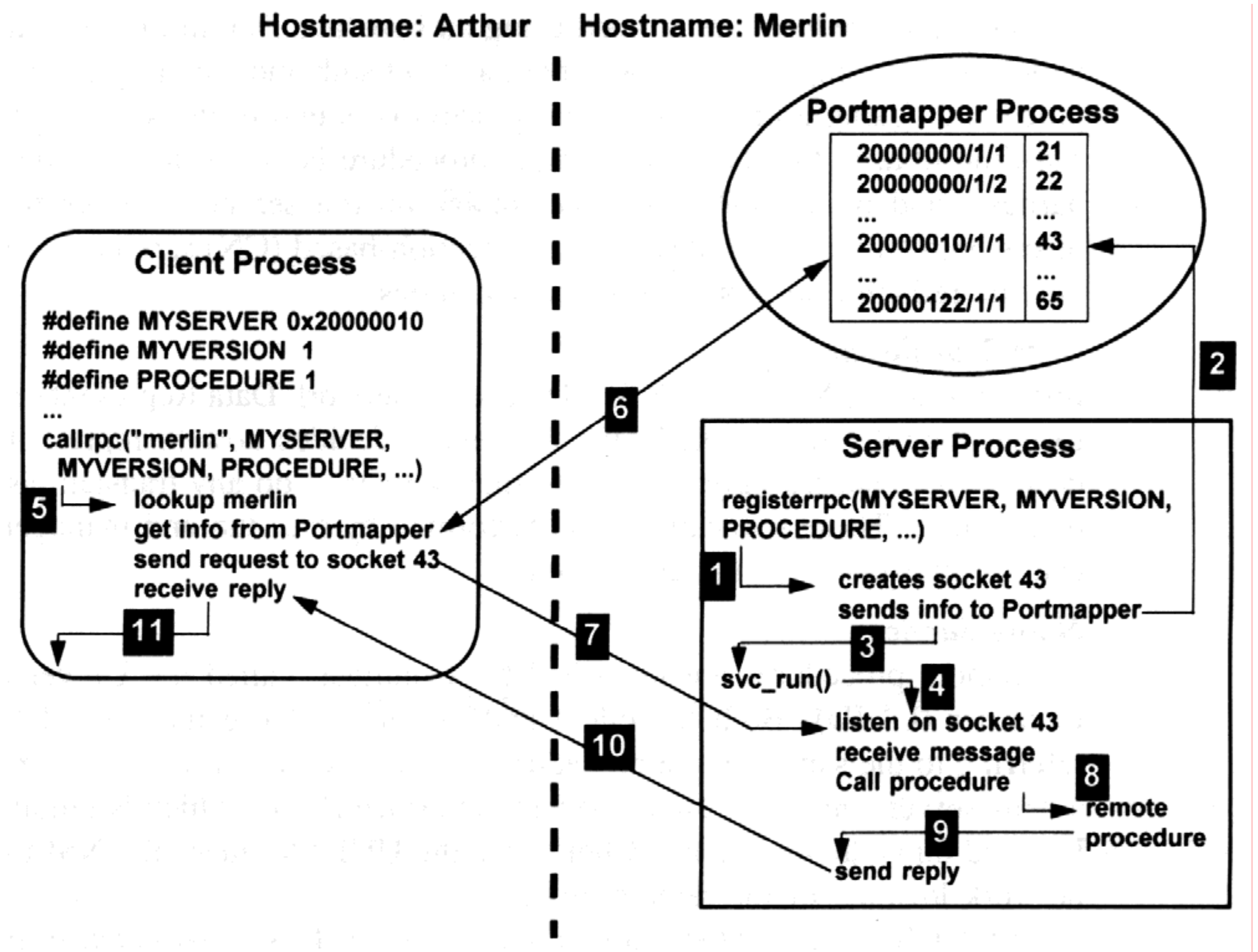


RPC program number groups

Range	Usage
00000000–1FFFFFFF	Defined by Sun
20000000–3FFFFFFF	Defined by User
40000000–5FFFFFFF	Assigned Dynamically
60000000–FFFFFFFF	Reserved

- Portmapper maintains
  - program numbers, partitioned into group of 20000000_{16}
  - version number
  - procedure number
  - socket id
- Portmapper itself runs on a fixed port defined in the `/etc/services(port 111).`

Inside of RPC

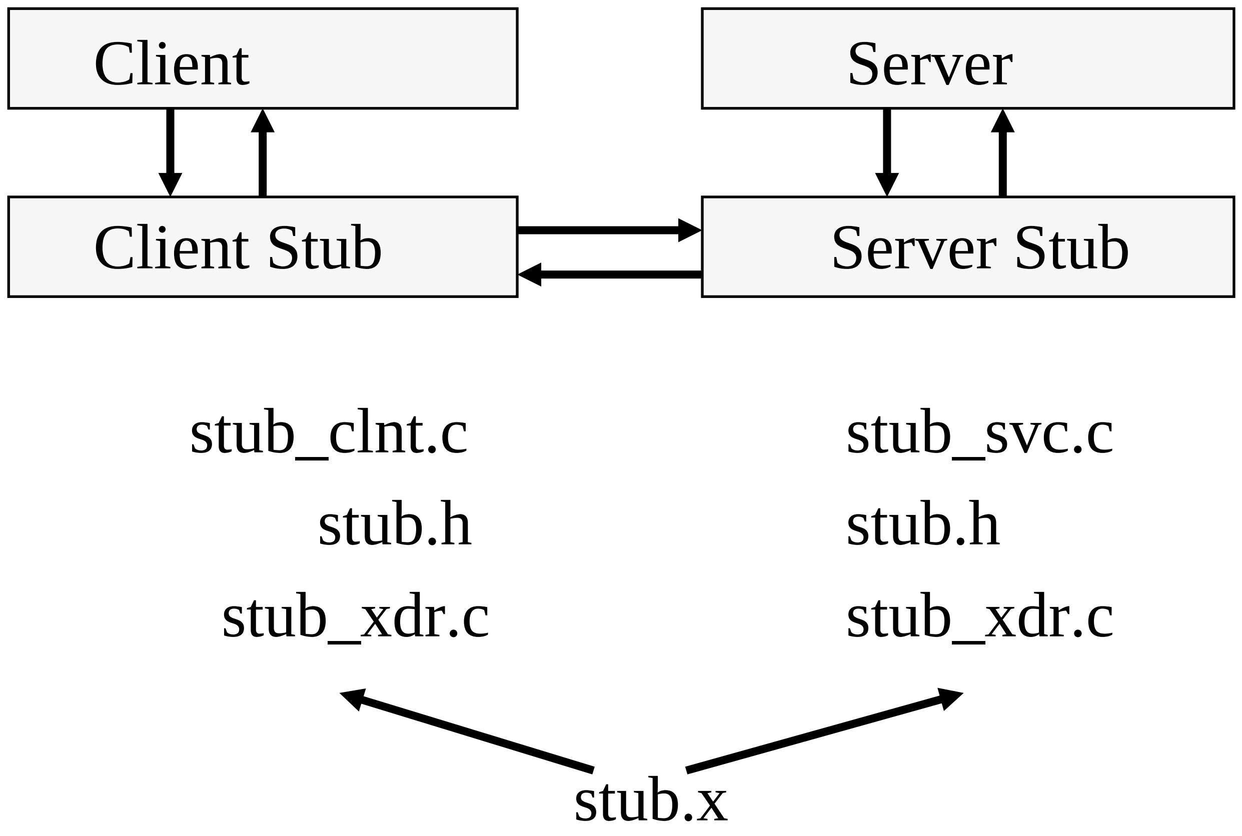


Behind the scenes with RPC.

XDR Language – Higher Level

- File Model
- Language definitions

File Model


Client
Server
Client Stub
Server Stub
stub_clnt.c
stub.h
stub_xdr.c
stub_svc.c
stub.h
stub_xdr.c
stub.x

RPC Language -- (1)

- XDR service routines are still hard to write.
- Solution: provide an RPC language so that a compiler can help automatically generate XDR routines.
- Example : in file stub.x.

```
struct avg{
     int x;
     int y;
};
```

- Compiler `rpcgen` generate stub.h.

```
typedef struct avg{
     int x;
     int y;
} avg;
```

RPC Language -- (2)

- Also generate stub_svc.c and stub_clnt.c, stub_xdr.c, including

```
xdr_avg(pointer, xdrsp)
avg *pointer;
XDR *xdrsp;{
      xdr_int (xdrsp, &pointer->x);
      xdr_int (xdrsp, &pointer->y)
}
```

RPC language -- (3)

Pointer Declarations (optional data)

```
listitem *next;
```

$\longrightarrow$

```
listitem *next;
```

Structures

```
struct coord{
     int x;
     int y;
};
```

$\longrightarrow$

```
struct coord{
    int x;
    int y;
}
typedef struct coord coord;
```

RPC language -- (4)

Unions

```
union read_result switch (int errno){
case 0:
    opaque data[1024];
default:
    void;
};
```

$\longrightarrow$

```
struct read_result{
    int errno;
    union{
        char data[1024];
    }read_result_u;
};
typedef struct read_result read_result;
```

RPC language -- (5)

Enumerations

```
enum colortype{
    RED = 0;
    GREEN=1;
    BLUE=2;
};                          };
```

```
enum colortype{
   RED=0;
   GREEN=1;
   BLUE=2;
 }
typedef enum colortype
                colortype;
```

Typedefs

```
typedef string fname_type<255>;
```

```
typedef char *frame_type
```

RPC language -- (6)

Constants

```
int name<12>  /* at most 12 items */
int name<>                  /* any number of items */
struct{
      u_int name_len;
      int *name_val;
}name;
```

RPC language -- (7)

Programs

```
program TIMEPROG{
    version TIMEVERSA{
        unsigned int TIMEGATA(void) =1 ;
        void TIMESETA(unsigned) = 2;
    } = 1:
    version TIMEVERSB{
        unsigned int TIMEGETB(void) = 1;
        void TIMESETB(unsigned) = 2;
    } = 2:
}=44;
```

RPC language -- (8)

```
#define     TIMEPROG        44
#define     TIMEVERSA       1
#define     TIMEGETA        1
#define     TIMESETA        2
#define     TIMEVERSB       2
#define     TIMEGETB        1
#define     TIMESETB        2
```

Opaque

```
opaque diskblock[512];
opaque filedata<1024>;
```

Features – Lower Level

- Use the UDP protocol by default.
- Use lower-level RPC calls to change the underlying protocol
  - set the number of retries
  - send more than 8KB of data
  - perform special authentication
  - handle callbacks
  - batch several RPC requests into one call
  - broadcast requests

DCE RPC

- Promoted by OSF. Members includes. IBM, DEC, HP, etc.
- Interface:
  - Uses threads for RPC.

    DCE threads are user-level subprocesses.
  - Define an Interface Definition Language (IDL)for the remote procedure.
  - Use Network Data Representation (NDR) format for data interchange.
- Outline
  - Name Service
  - Security
  - Threads