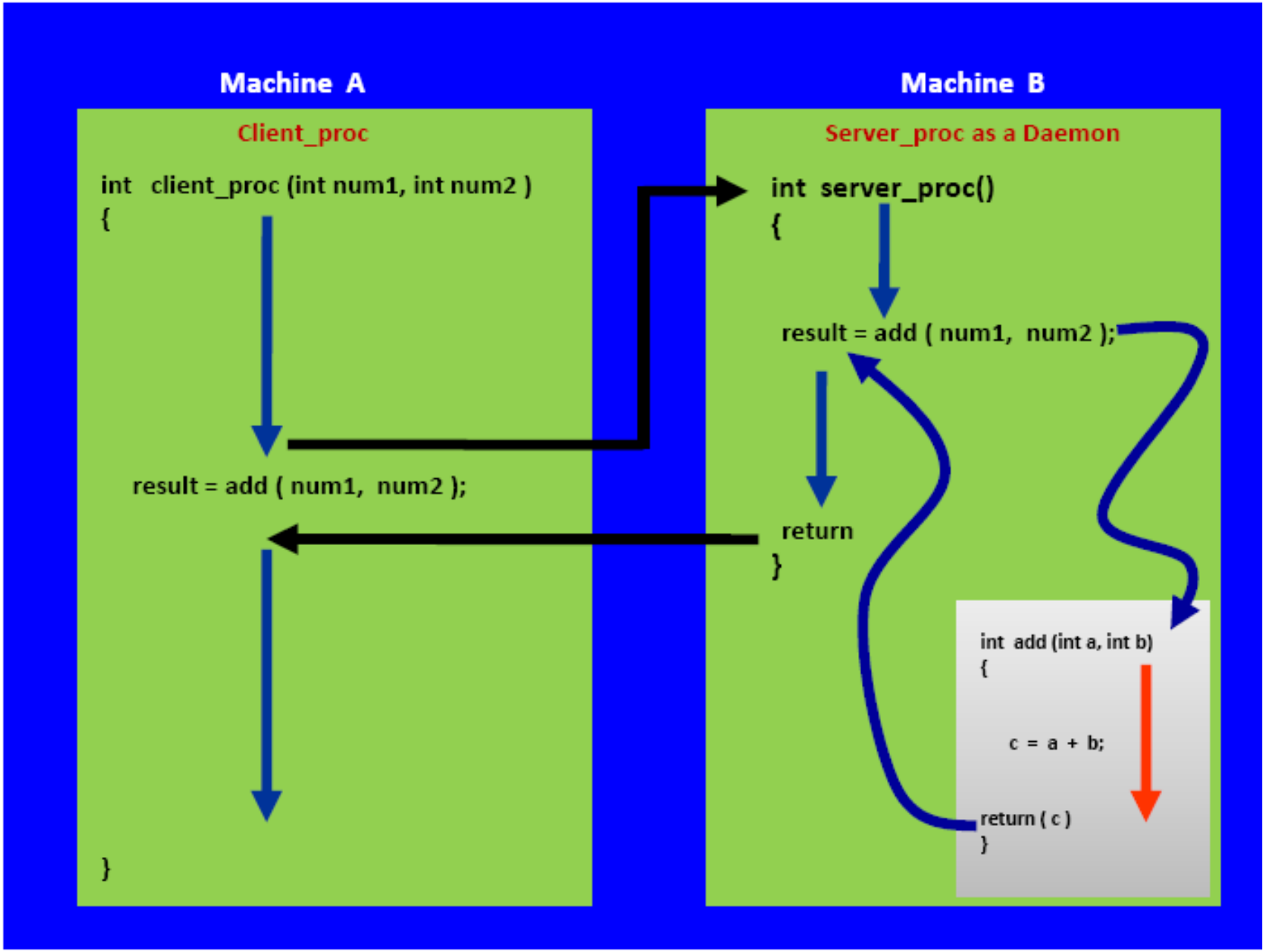
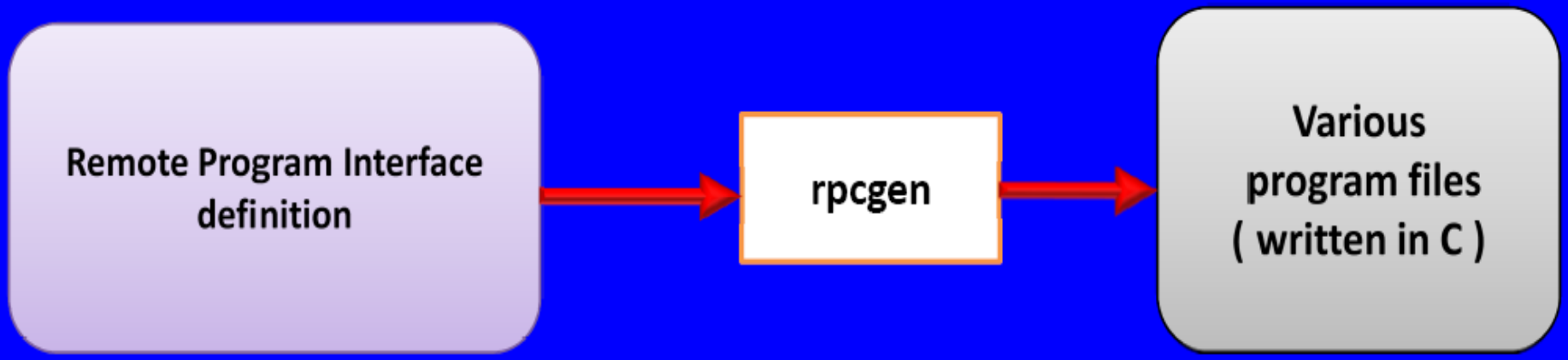
Title: Remote Procedure Call
Practical Programming using SUN RPC


Machine A
Machine B
Client_proc
int client_proc (int num1, int num2)
{
result = add (num1, num2);
}
Server_proc as a Daemon
int server_proc()
{
result = add (num1, num2);
return
}
int add (int a, int b)
{
c = a + b;
return (c)
}

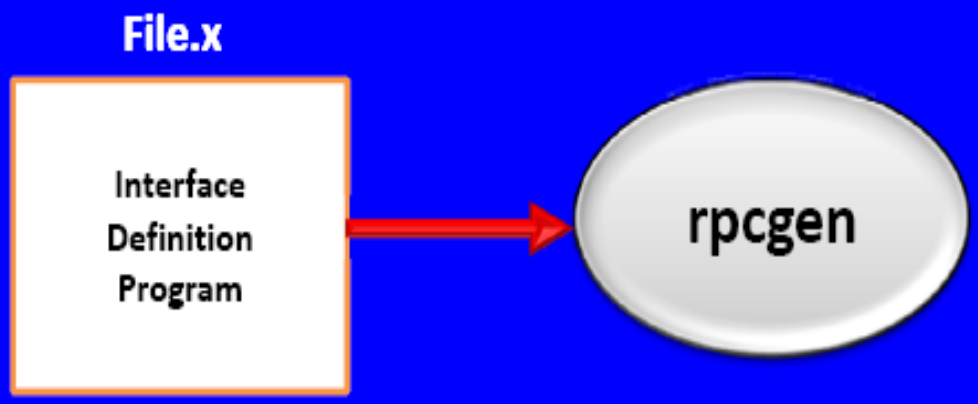
Sun RPC

- **rpcgen :** rpcgen is a compiler.



RPC language

- Provides function and data declaration facilities.



Example IDL file

P.x

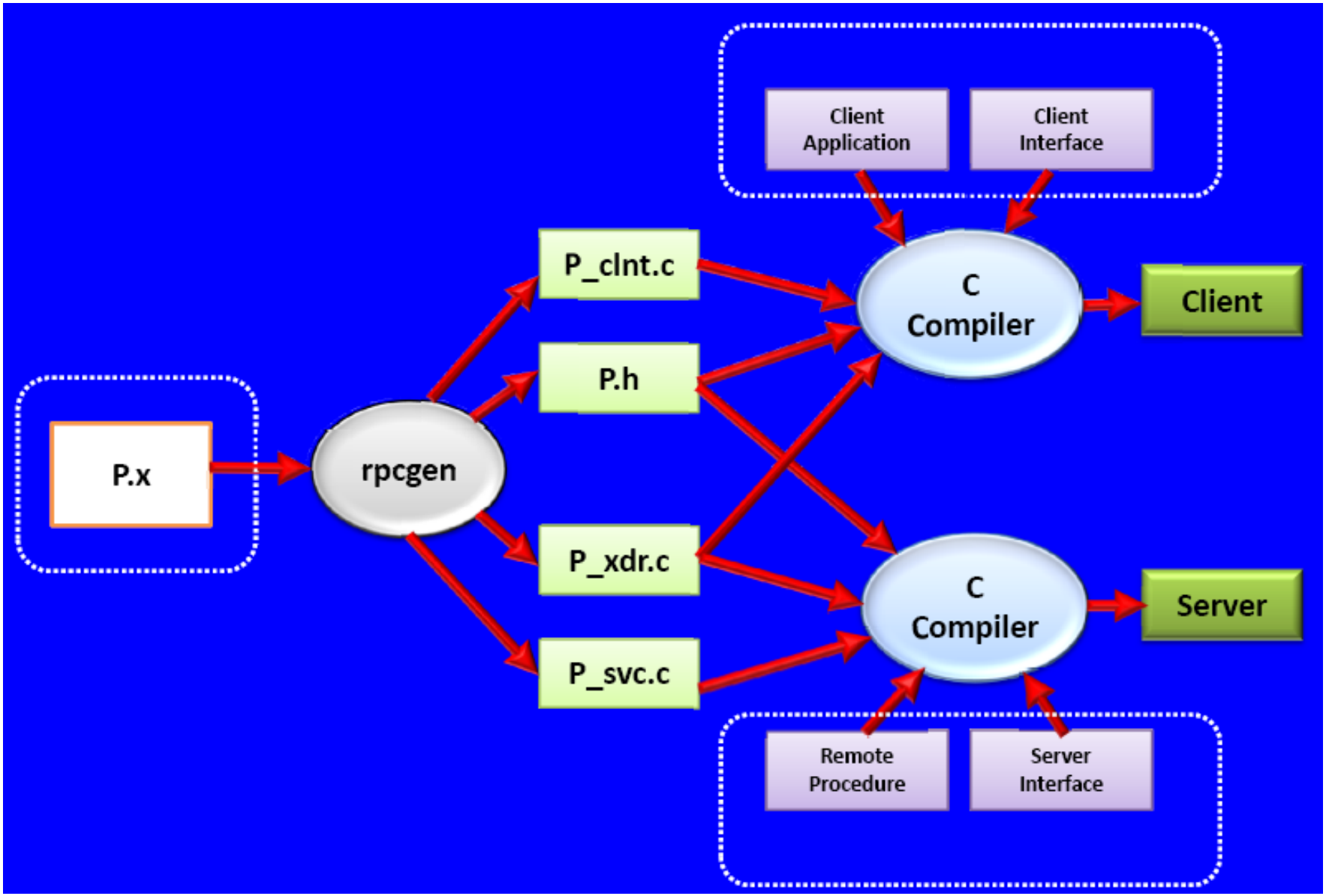
```
program   MYFIRSTRPCPROGRAM  {

        version  MYFIRSTPROGRAMVERSION {

                void procedure(void)=1;

        }=1;

}="32 bit hex number";
```


Client Application
Client Interface
P_clnt.c
C Compiler
Client
P.h
P.x
rpcgen
P_xdr.c
C Compiler
Server
P_svc.c
Remote Procedure
Server Interface

test program

- Specification file : test.x
- Client program : test_client.c
- Server program : test_server.c

Start Server and client program

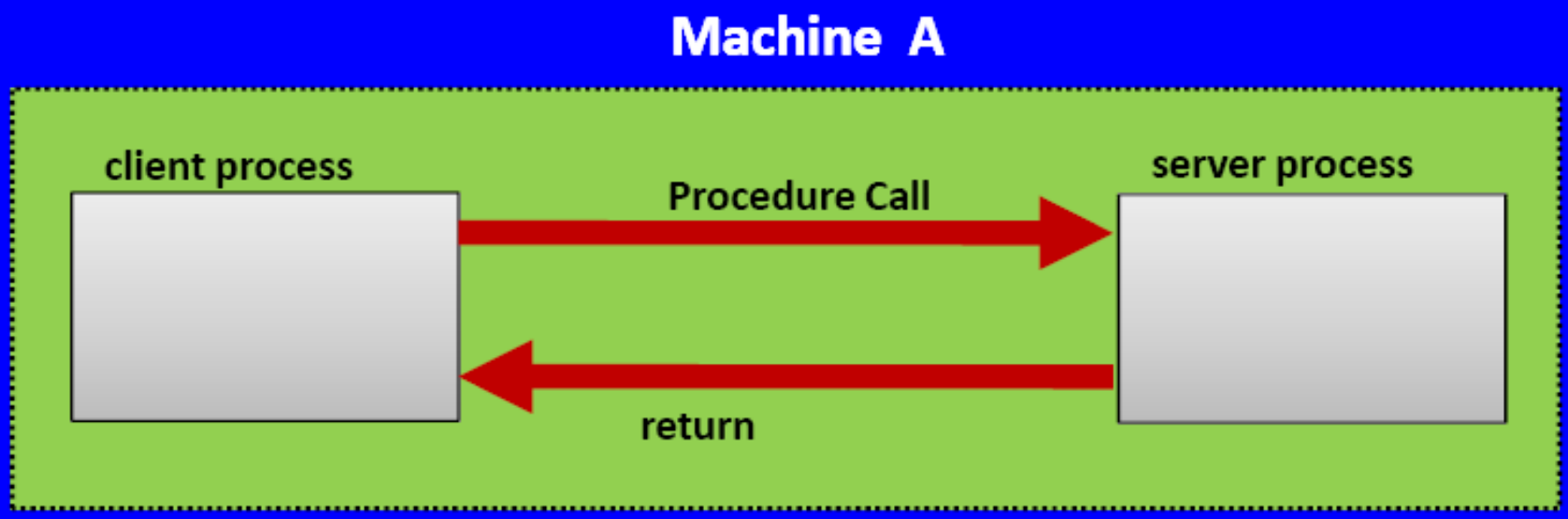
```
[root@localhost test1]# ls
Makefile.test  test_client.o  test.h        test_server.o  test.x
test_client*   test_clnt.c    test_server*   test_svc.c
test_client.c  test_clnt.o    test_server.c  test_svc.o
[root@localhost test1]# ./test_server
```

Start server program

```
[root@localhost test1]# ./test_client 127.0.0.1
[root@localhost test1]#
```

RPC on a single host

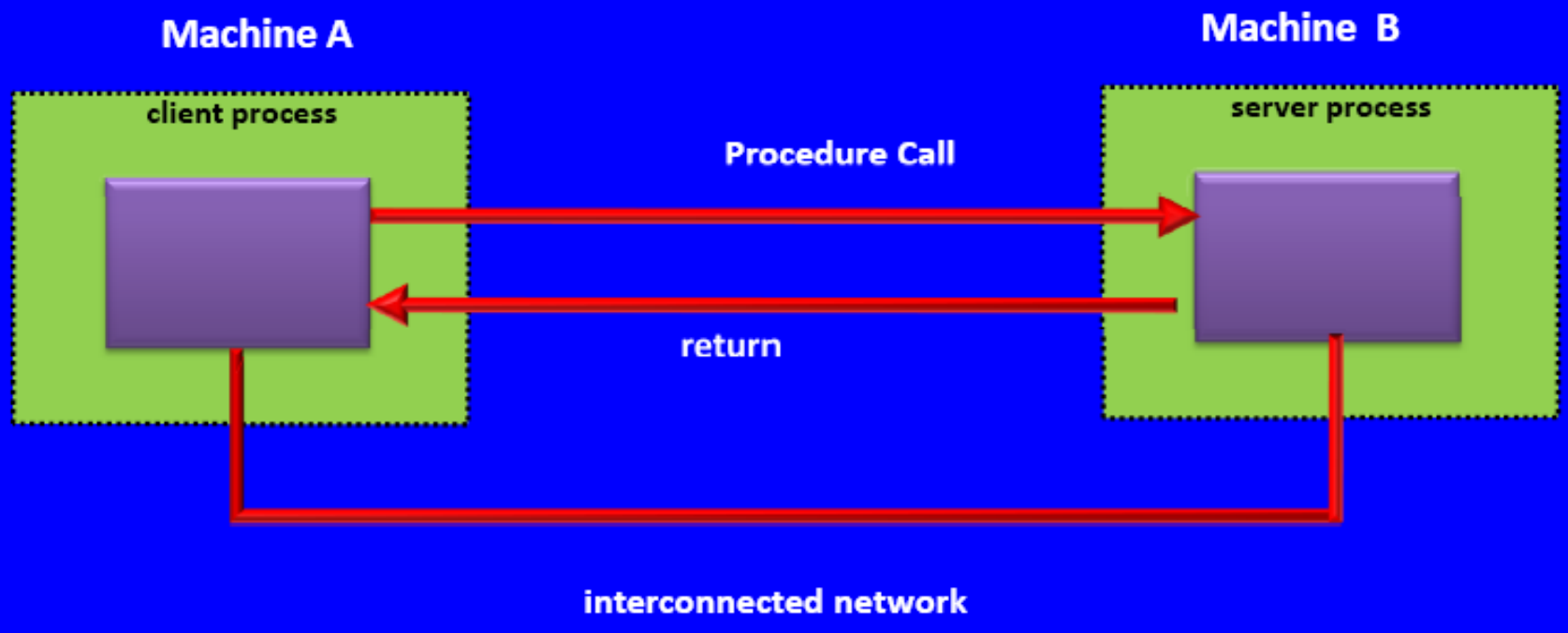
- Client and server processes are in separate address spaces.



❖ A process call a procedure in another process on the same host.

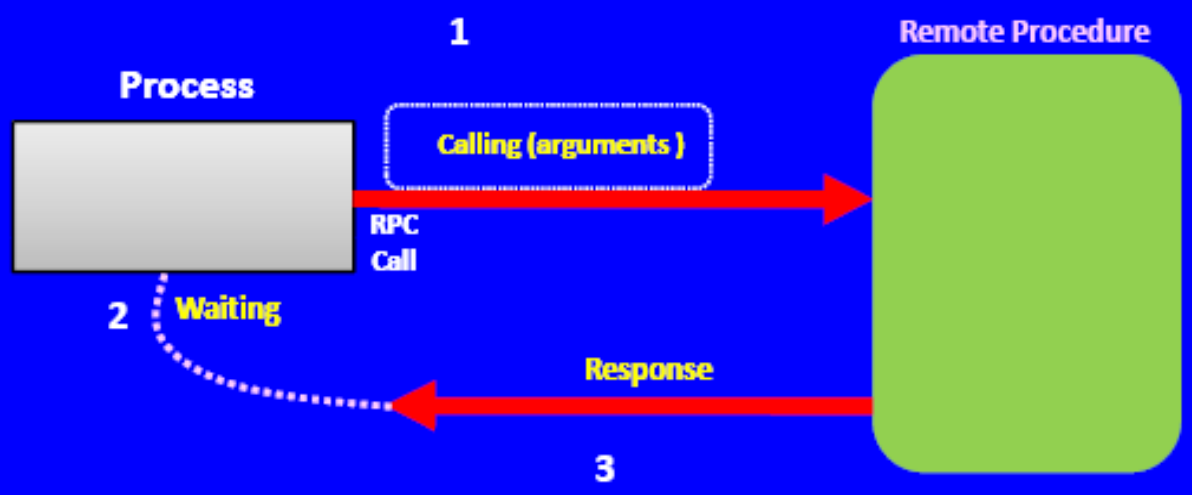
RPC between hosts

- RPC in general allows a client on one host to call a server procedure on another host.

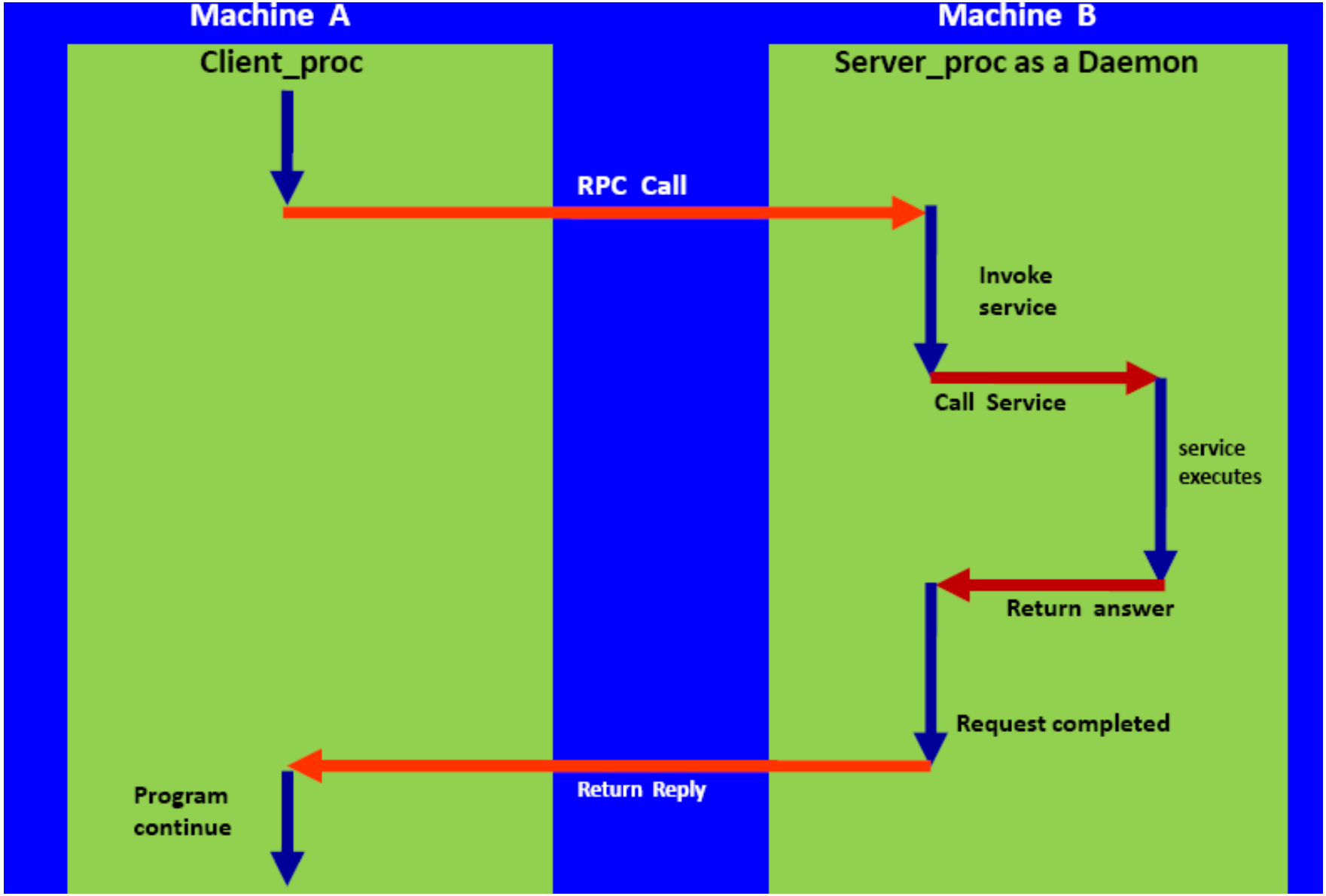


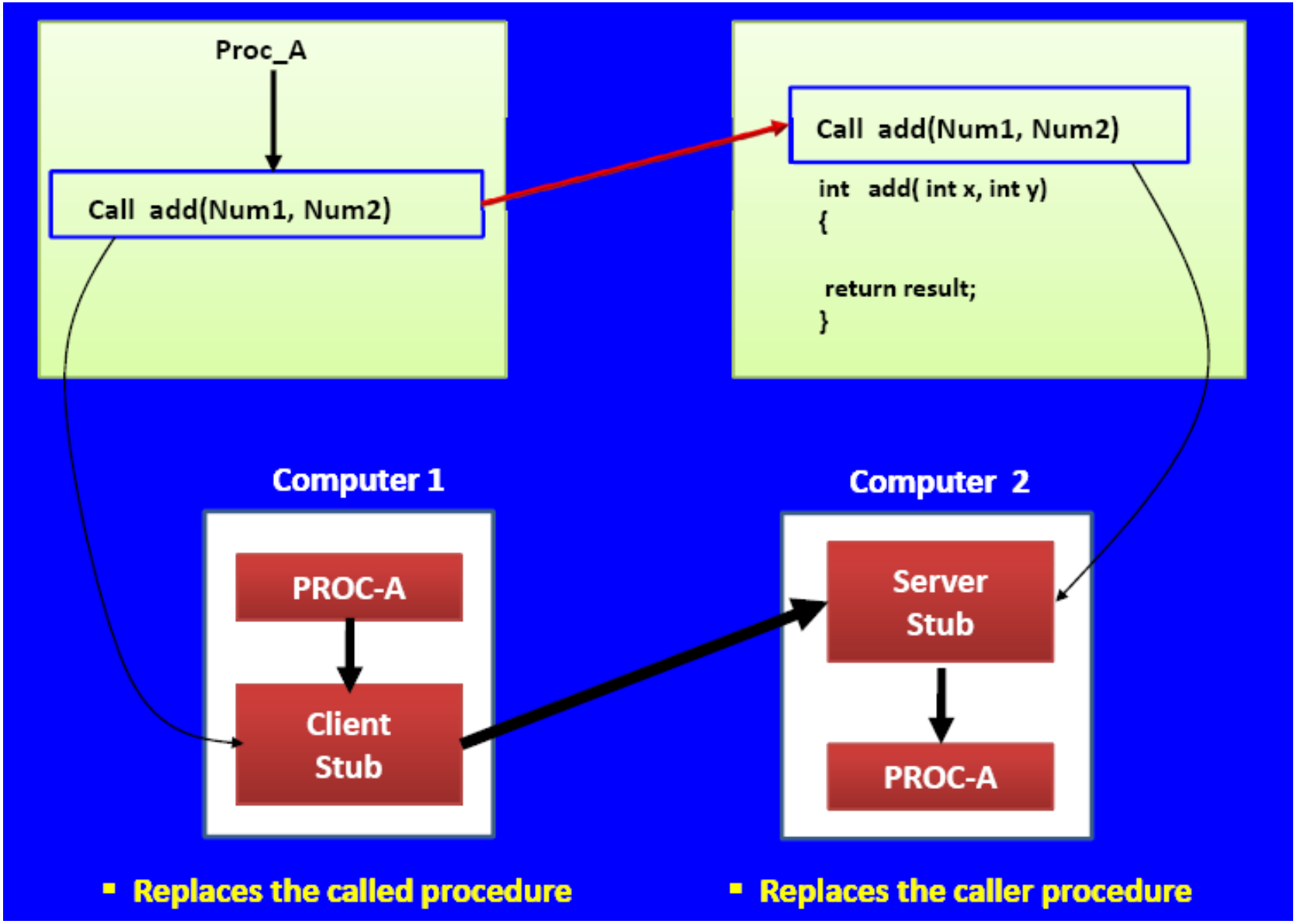
How RPC works ?

- An RPC is analogous to a function call.



- When an RPC is made, the calling arguments are passed to the remote procedure and the caller waits for a response to be returned from the remote procedure.


Machine A
Machine B
Client_proc
Server_proc as a Daemon
RPC Call
Invoke service
Call Service
service executes
Return answer
Request completed
Return Reply
Program continue


Proc_A
Call add(Num1, Num2)
Call add(Num1, Num2)
int add(int x, int y)
{
return result;
}
Computer 1
PROC-A
Client Stub
Computer 2
Server Stub
PROC-A
Replaces the called procedure
Replaces the caller procedure

SUN RPC

RPC issues

- RPC server can have many procedures

Basic issues

- Identify and access the remote procedure
- Parameters required to call a procedure
- Return value from the procedure

Procedure Arguments

- **Single argument: Sun RPC includes support for a single argument to a remote procedure.**
- **Typically the single argument is a structure that contains a number of values.**

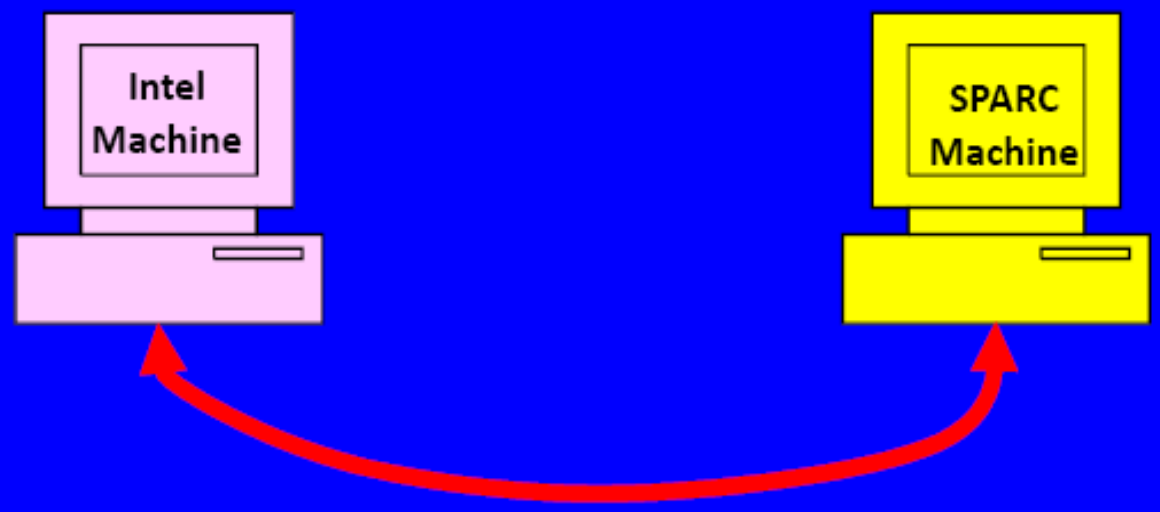
```
struct  Num  {
            int  a,  b;
};

Call proc_add( struct Num);
```

```
proc_add (int a, int b)
{
        ....................
}
```

External Data Representation (XDR)

- XDR is a machine-independent description and encoding of data that can communicate between diverse machines



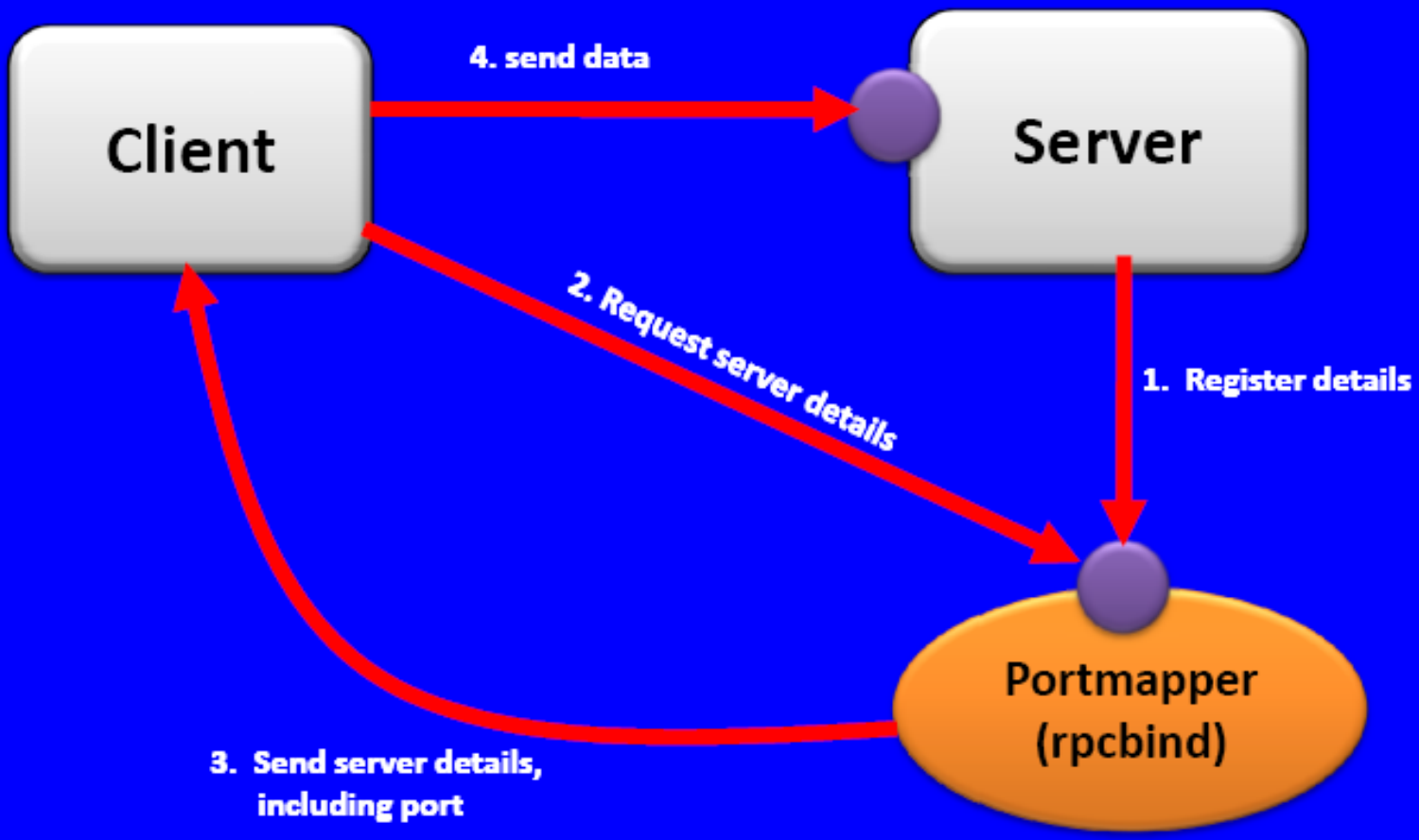
- Serialization: Converting from a particular machine representation to XDR format is called serializing; the reverse process is deserializing.

- How does a client find the right server over the network?

  - Ordinary client-server code: the user must supply a host name and a port number.

  - RPC: the user only supplies a host name

Steps in RPC Communication



Steps to handle

- Step 1. Create the IDL
- Step 2. Generate sample client and server code
- Step 3. First test of the client and server
- Step 4. Getting the server to do some work
- Step 5. Making the client functional

Step 1. Create the IDL (Defining the interface)

1. Declarations for constants used in the client or server.

2. Declarations of the data types used (especially in arguments to remote procedures).

3. Declarations of remote programs, the procedures contain in each program, and the types of their parameters.

File with .x extension

```
program   program_name {

        version  program_version {

                        procedure_name_1()= 'procedure_number';
                        procedure_name_2()= 'procedure_number';
                        ..............................................
                        procedure_name_N()= 'procedure_number';

        } = 'version_number';

} = '32-bits Hex number';
```

Example: demo1.x

```
Program DEMO_PROG {
    version DEMO_VERSION {
        type1 PROC1(operands1) = 1;
        type2 PROC2(operands2) = 2;
    } = 1;
} = 40000000;
```

Color Code:

Keywords

Generated Symbolic Constants

Used to generate stub and procedure names

Program Numbers

- Each remote program executing on a computer must be assigned a unique 32 – bit integer that the caller uses to identify it.

```
Program DEMO_PROG {
    version DEMO_VERSION {
            type1 PROC1(operands1) = 1;
            type2 PROC2(operands2) = 2;
    } = 1;
} = 40000000;
```

Procedure Numbers

```
Program DEMO_PROG {
    version DEMO_VERSION {
            type1 PROC1(operands1) = 1;
            type2 PROC2(operands2) = 2;
    } = 1;
} = 40000000;
```

- SUN RPC assigns an integer identifier to each remote procedure inside a given remote program.
- The procedures are numbered sequentially: 1,2,3…N.

Procedure Names

```
Program DEMO_PROG {
    version DEMO_VERSION {
            type1  PROC1  (operands1) = 1;

            type2  PROC2  (operands2) = 2;
    } = 1;
} = 40000000;
```

Version numbers

```
Program DEMO_PROG {

    version DEMO_VERSION {

        type1 PROC1(operands1) = 1;
        type2 PROC2(operands2) = 2;

    } = 1;

} = 40000000;
```

Example specification file : test.x

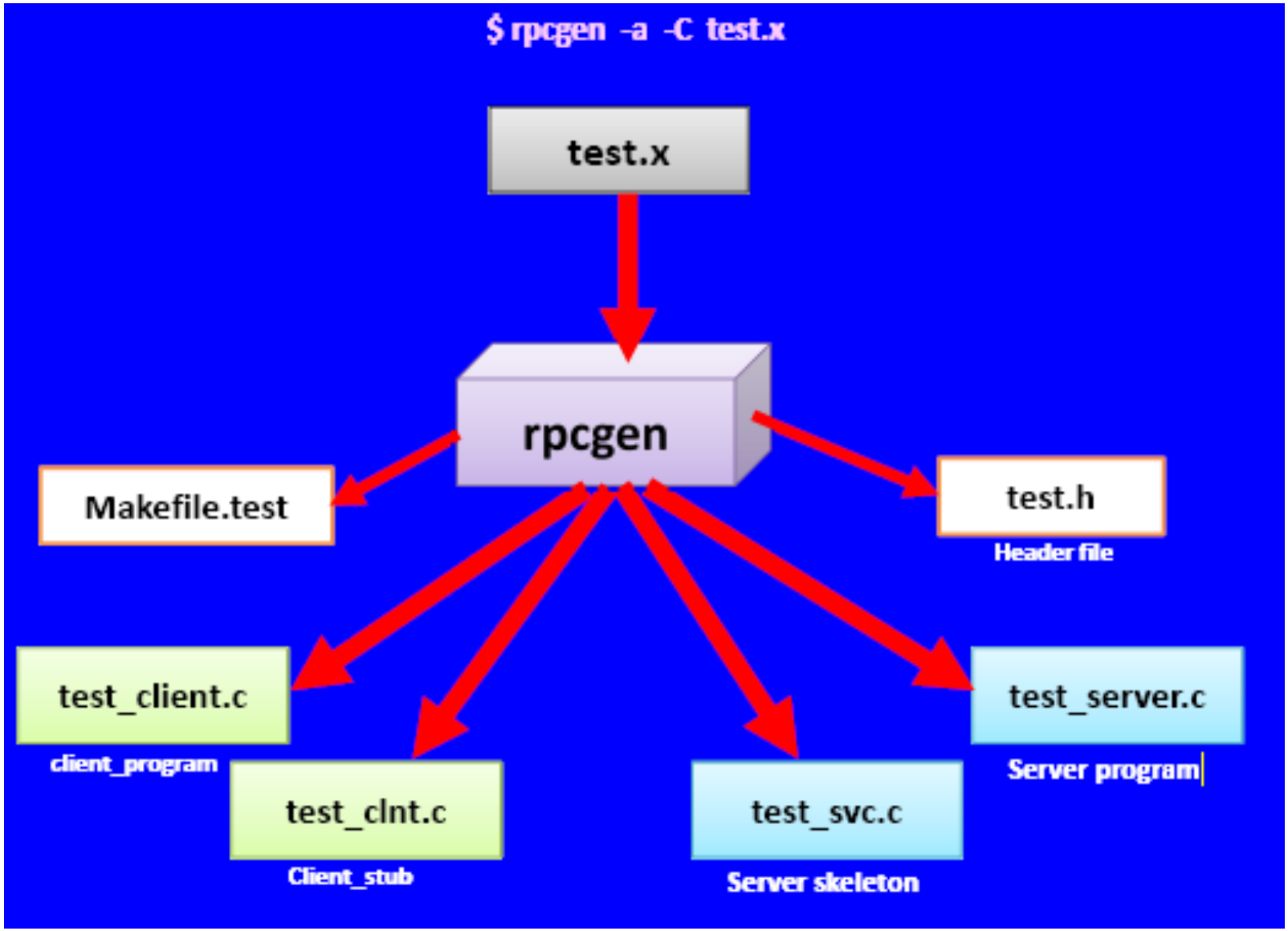
```
program   TEST_PROGRAM  {

          version   TEST_VERSION {

                    void   TEST_PROC(void)=1;
          }=1;
}   = 22222222;
```

Step 2. Generate sample client and server code

```
[root@localhost test1]# ls
test.x
[root@localhost test1]# rpcgen -a -C test.x
```


$ rpcgen -a -C test.x
test.x
rpcgen
Makefile.test
test.h
Header file
test_client.c
client_program
test_clnt.c
Client_stub
test_svc.c
Server skeleton
test_server.c
Server program

test.h

```
#ifndef _TEST_H_RPCGEN
#define _TEST_H_RPCGEN

#include <rpc/rpc.h>


#ifdef __cplusplus
extern "C" {
#endif


#define TEST_PROGRAM 22222222
#define TEST_VERSION 1

#if defined(__STDC__) || defined(__cplusplus)
#define TESTPROC 1
extern  void * testproc_1(void *, CLIENT *);
extern  void * testproc_1_svc(void *, struct svc_req *);
extern int test_program_1_freeresult (SVCXPRT *, xdrproc_t, caddr_t);
```

```
program   TEST_PROGRAM {

        version   TEST_VERSION {

                void   TEST_PROC(void)=1;
        }=1;
} = 22222222;
```

Step 3. First test of the client and server

Edit makefile

- **Edit the makefile and find the line that defines CFLAGS:**

1

```
CFLAGS += -g
and change it to:

CFLAGS += -g  -DRPC_SVC_FG
```

- **We will make sure that the server is compiled so that the symbol RPC_SVC_FG is defined.**

- **This will cause our server to run in the foreground.**

2

- Change RPCGENFLAGS = to
- RPCGENFLAGS = -C

- rpcgen generates code that conforms to ANSI C, add a –C parameter to the rpcgen command

Makefile.test

```
# This is a template Makefile generated by rpcgen

# Parameters

CLIENT = test_client
SERVER = test_server

SOURCES_CLNT.c =
SOURCES_CLNT.h =
SOURCES_SVC.c =
SOURCES_SVC.h =
SOURCES.x = test.x

TARGETS_SVC.c = test_svc.c test_server.c
TARGETS_CLNT.c = test_clnt.c test_client.c
TARGETS = test.h   test_clnt.c test_svc.c test_client.c test_server.c

OBJECTS_CLNT = $(SOURCES_CLNT.c:%.c=%.o) $(TARGETS_CLNT.c:%.c=%.o)
OBJECTS_SVC = $(SOURCES_SVC.c:%.c=%.o) $(TARGETS_SVC.c:%.c=%.o)
# Compiler flags

CFLAGS += -g
LDLIBS += -lnsl
RPCGENFLAGS =
```

```
# Compiler flags

CFLAGS += -g -DRPC_SVC_FG
LDLIBS += -lnsl
RPCGENFLAGS = -C
```

- **The template code written by rpcgen.** test_client.c

```
void
test_program_1(char *host)
{
        CLIENT *clnt;
        void  *result_1;
        char *testproc_1_arg;

#ifndef DEBUG
        clnt = clnt_create (host, TEST_PROGRAM, TEST_VERSION, "udp");
        if (clnt == NULL) {
                clnt_pcreateerror (host);
                exit (1);
        }
#endif  /* DEBUG */

        result_1 = testproc_1((void*)&testproc_1_arg, clnt);
        if (result_1 == (void *) NULL) {
                clnt_perror (clnt, "call failed");
        }
#ifndef DEBUG
        clnt_destroy (clnt);
#endif  /* DEBUG */
```

Return value of a Function

Function parameter

```c
int
main (int argc, char *argv[])
{
        char *host;

        if (argc < 2) {
                printf ("usage: %s server_host\n", argv[0]);
                exit (1);
        }
        host = argv[1];
        test_program_1 (host);
exit (0);
}
```

test_client.c:

- A client template for an interface created by rpcgen.
- Contains:
  - Declaration of function parameters.
  - Return values for each of the functions.

test_server.c

- The server function: in test_server.c file
- It does nothing.
- It contains only comments:

test_server.c

```c
/*
 * This is sample code generated by rpcgen.
 * These are only templates and you can use them
 * as a guideline for developing your own functions.
 */

#include "test.h"

void *
testproc_1_svc(void *argp, struct svc_req *rqstp)
{
        static char * result;

        /*
         * insert server code here
         */

        return (void *) &result;
}
```

Step 4. Getting the server to do some work

- Replace comments with a single print statement:

```
printf("connection checked  \ n");
```

test_server.c

```c
#include "test.h"

void *
testproc_1_svc(void *argp, struct svc_req *rqstp)
{
        static char * result;

        /*
         * inserted test code here
         *
        */
        printf("connection checked\n");

        return (void *) &result;
}
```

Step 5. Run programs

- **Build using make**

```
[root@localhost test1]# make -f Makefile.test
cc -g -DRPC_SVC_FG   -c -o test_clnt.o test_clnt.c
cc -g -DRPC_SVC_FG   -c -o test_client.o test_client.c
cc -g -DRPC_SVC_FG    -o test_client  test_clnt.o test_client.o -lnsl
cc -g -DRPC_SVC_FG   -c -o test_svc.o test_svc.c
cc -g -DRPC_SVC_FG   -c -o test_server.o test_server.c
cc -g -DRPC_SVC_FG    -o test_server  test_svc.o test_server.o -lnsl
```

Remote Procedure Call: Fourth Step

Write meaningful RPC programs

add.c

- This program prints out the addition of two numbers provided by the user on the command line.

```
$./add   5  6

    11
$
```

❖ first get the stand-alone application working.

Data and functions in the program

iInfo

```
struct InputInfo {
        int num1;
        int num2;
};
```

oInfo

```
struct OutputInfo {
        int result;
};
```

```
void displayResult(struct OutputInfo oInfo);

struct OutputInfo performAddition(struct InputInfo iInfo);
```

add.x

Session Edit View Bookmarks Settings Help

```
struct InputInfo {
        int num1;
        int num2;
};

struct OutputInfo {
        int result;
};

program ADDPROGRAM {
        version ADDVERSION {
                struct OutputInfo performAddition(struct InputInfo iInfo)=1;
        }=1;
}=22222222;
```

add_client.c

```
int
main (int argc, char *argv[])
{
        char *host;
        int m, n;
        if (argc < 4) {
                printf ("usage: %s server_host/IP  num1 num2 \n", argv[0]);
                exit (1);
        }
        host = argv[1];
        m = atoi(argv[2]);
        n = atoi(argv[3]);
        addprogram_1 (host,m,n);
exit (0);
}
```

```
$./add   5  6
```

1

```
void  displayResult( struct  OutputInfo*  oInfo )
{
        6   printf(" sum of two numbers=%d\n ", oInfo->result );
}
```

```
addprogram_1(char *host, int n1, int n2)
{
        CLIENT *clnt;
        struct OutputInfo  *result_1;
        struct InputInfo  performaddition_1_arg;

        performaddition_1_arg.num1 = n1;
        performaddition_1_arg.num2 = n2;
#ifndef DEBUG
        clnt = clnt_create (host, ADDPROGRAM, ADDVERSION, "udp");
        if (clnt == NULL) {
                clnt_pcreateerror (host);
                exit (1);
        }
#endif  /* DEBUG */

        result_1 = performaddition_1(&performaddition_1_arg, clnt);
        if (result_1 == (struct OutputInfo *) NULL) {
                clnt_perror (clnt, "call failed");
        }
        else { /* added part */

                displayResult(result_1);
        }
#ifndef DEBUG
        clnt_destroy (clnt);
#endif   /* DEBUG */
```

2 Provide appropriate no of arguments

Assign numbers 3

Call server function 4

5 Call display function

Steps to handle

- Step 1. Create the IDL
- Step 2. Generate sample client and server code
- Step 3. First test of the client and server
- Step 4. Getting the server to do some work
- Step 5. Making the client functional

Write up Points

1. What is RPC?
2. Explain rpcgen utility
3. Explain working of RPC.
4. Write steps in detail for SUN RPC program.

- Take printout of All files for server and client