Remote Procedure Calls

By
CHITRAKANT BANCHHOR
IT, SCOE, Pune

Outline

- Remote Procedure Call (Sun RPC)
  - Theoretical Introductions
  - Practical programming using Sun RPC
- Required practical RPC programs (CLP – II)

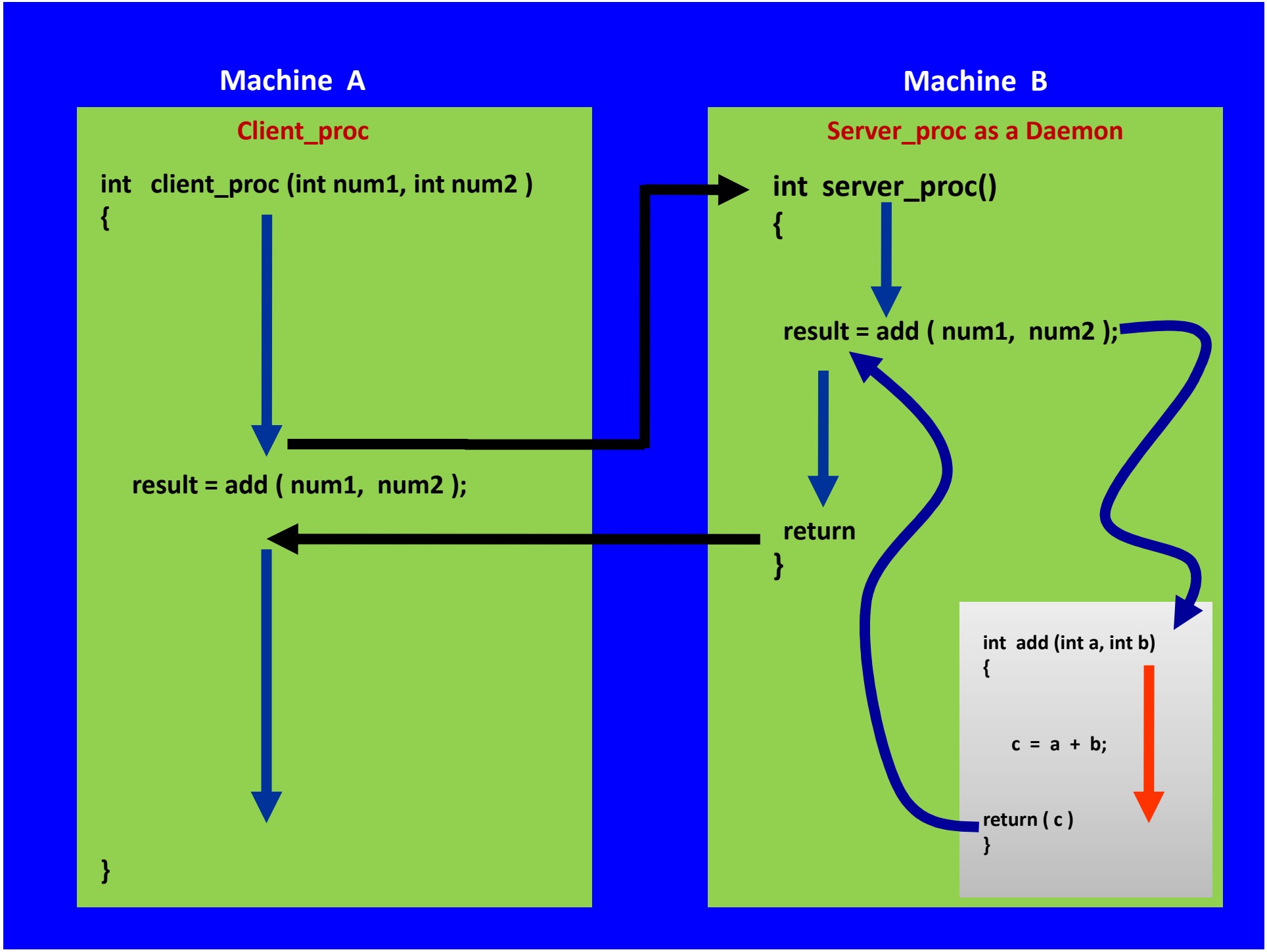
References

1. W. Richard Stevens, “UNIX Network Programming, Vol – 2”, PHI
2. Douglas E. Comer, “Internetworking With TCP/IP, Vol – 3”, PHI
3. Andrew S. Tanenbaum, “ Distributed Systems, 2nd Edition”, PHI

Remote Procedure Call

Remote Procedure Call

RPC is a call to a procedure/function located on another machine


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

RPC programming: First step

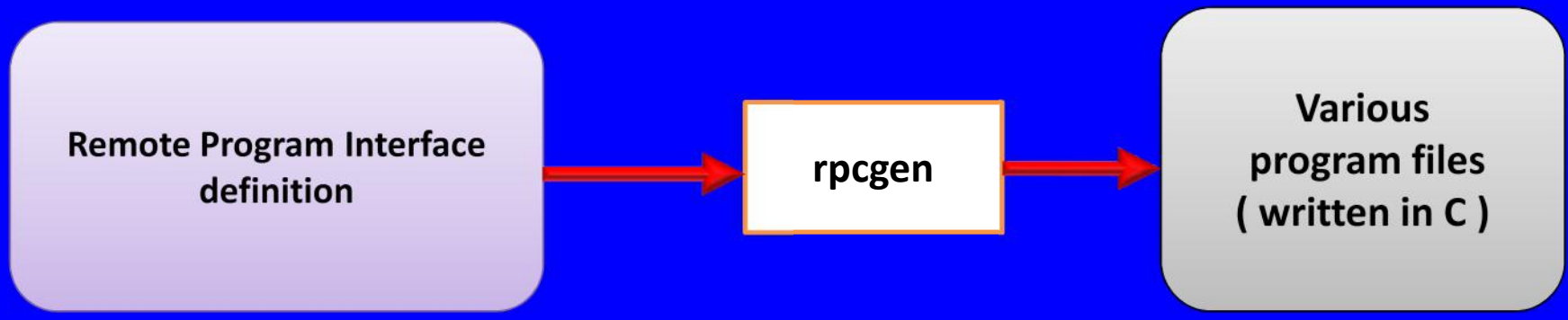
awareness about technology

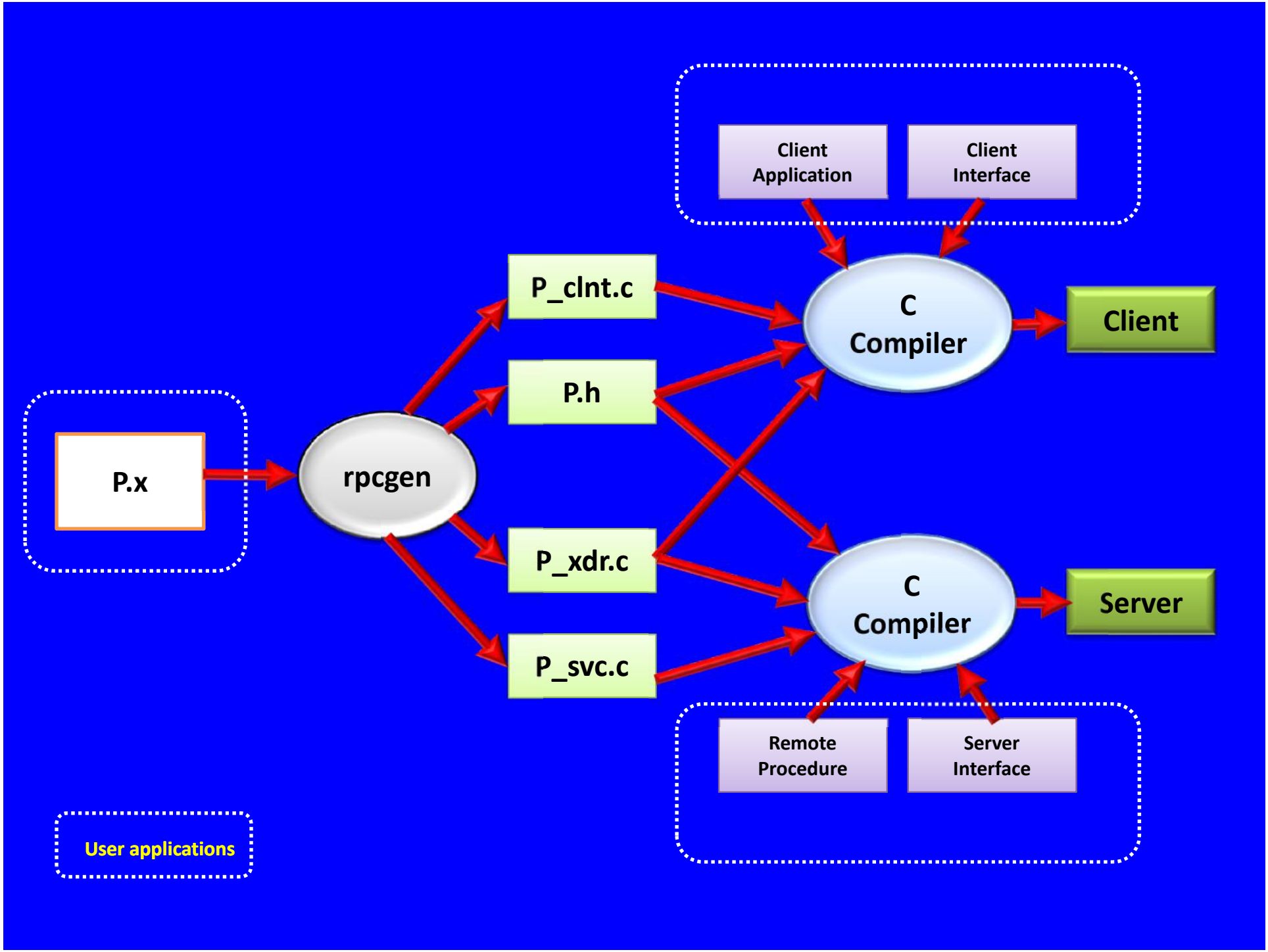
RPC implementations

1. Sun RPC (ONC RPC)
2. DCE RPC
3. ISO RPC

Sun RPC

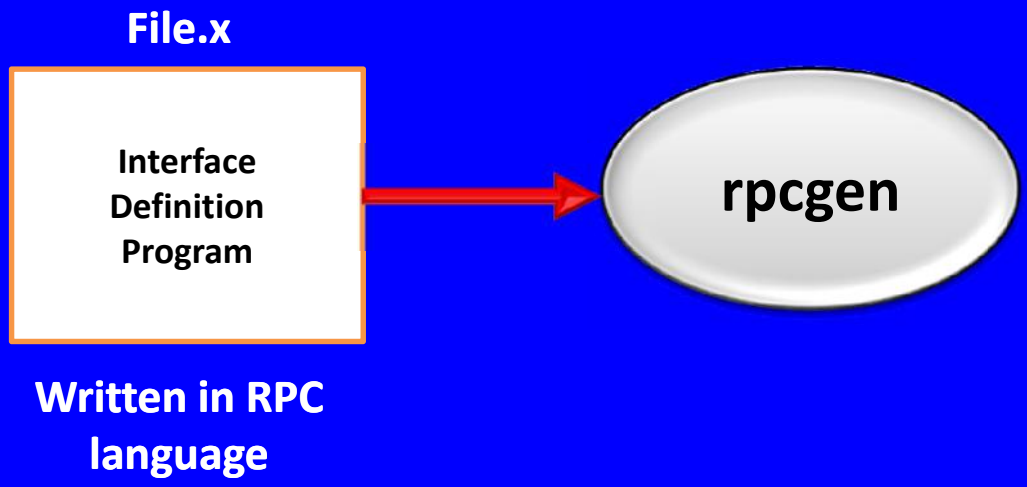
❖ **rpcgen :** rpcgen is a compiler.




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
User applications

RPC language

- **Provides function and data declaration facilities.**



The RPC Language Data types

typedefs

- typedefs have the same syntax as C typedefs:

"typedef" declaration

Constants

"const" const-ident "=" integer

const PI = 3.14;

Converted into C language as

#define PI 3.14

Fixed array in RPC

```
typedef  int  a[10];

program IARRAY_PROGRAM {

        version IARRAY_VERSION {

                int IARRAYADD(a)=1;

        }=1;
}=22222222;
```

Variable-Length Array Declarations

- type-ident variable-ident "<" value ">“
- type-ident variable-ident "<" ">"

1. int array1<12>; /* at most 12 items */
2. int array2<>; /* any number of items */

Strings

- **strings are declared using the string keyword and compiled into char *s in the output header file.**

❑ **string name< >;**

Converted into :

char* name;

voids

- The variable is not named as void.

- void declarations can only occur in following places:
  1. union definitions
  2. program definitions as the argument
  3. Return type of the result of a remote procedure.

```
program    DUMMY_PROGRAM  {

            void  PROCEDURE_ADD(  void  );
};
```

Structures

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

Enumerations

- same syntax as C enumerations

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

Booleans (bool_t)

- The boolean type called bool_t is either TRUE or FALSE.
- Things declared as type bool are compiled into bool_t in the output header file.

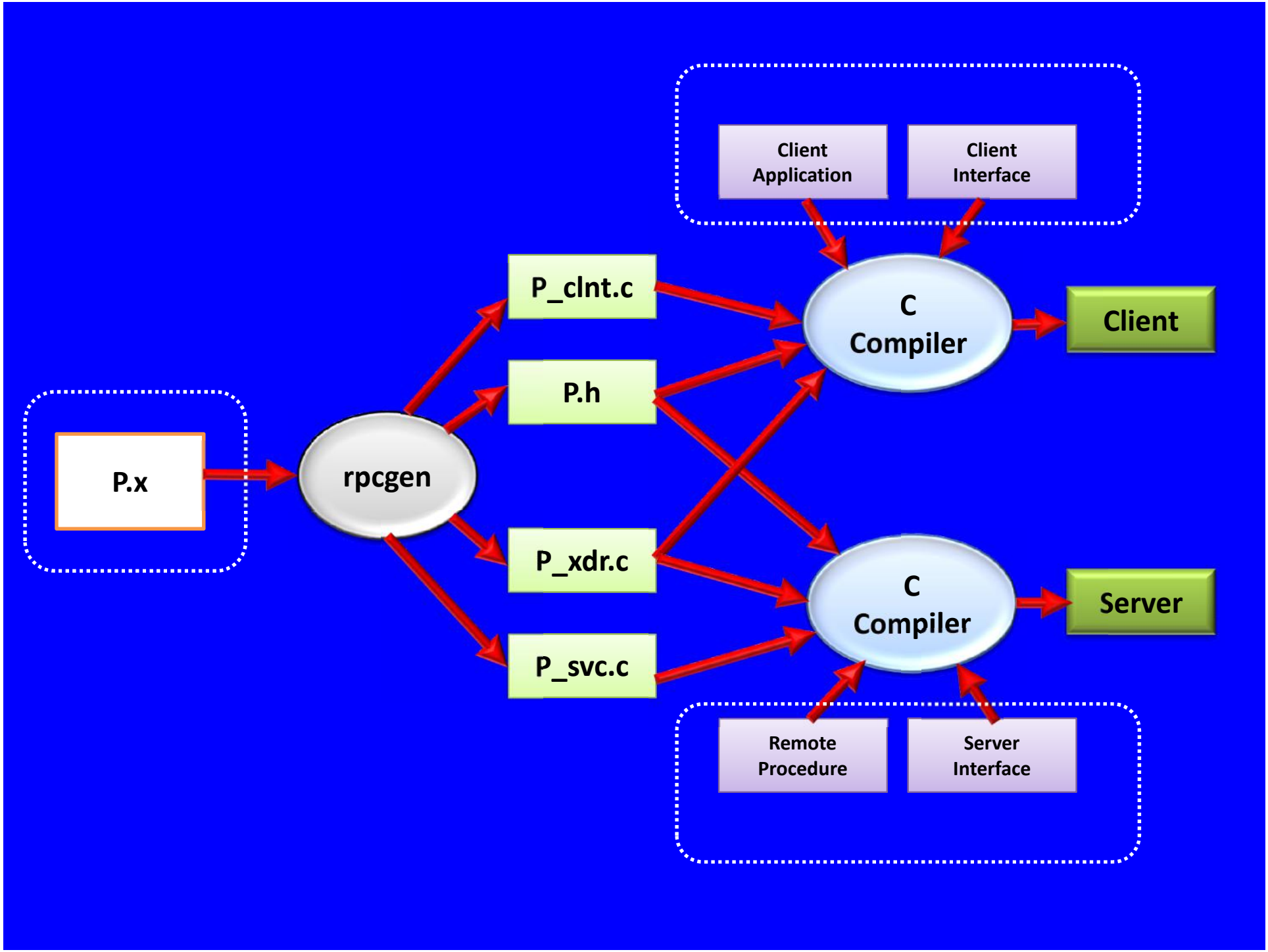
```
bool   AreYouMarried;
```

becomes bool_t AreYouMarried;

Example IDL file

P.x

```
program    MY_FIRST_RPC_PROGRAM  {

        version  MY_FIRST_PROGRAM_VERSION {

                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

Note: test_client.c

- A client template for an interface.

- Contains:
  - Declaration of function parameters.
  - Return values for each of the functions.

- **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

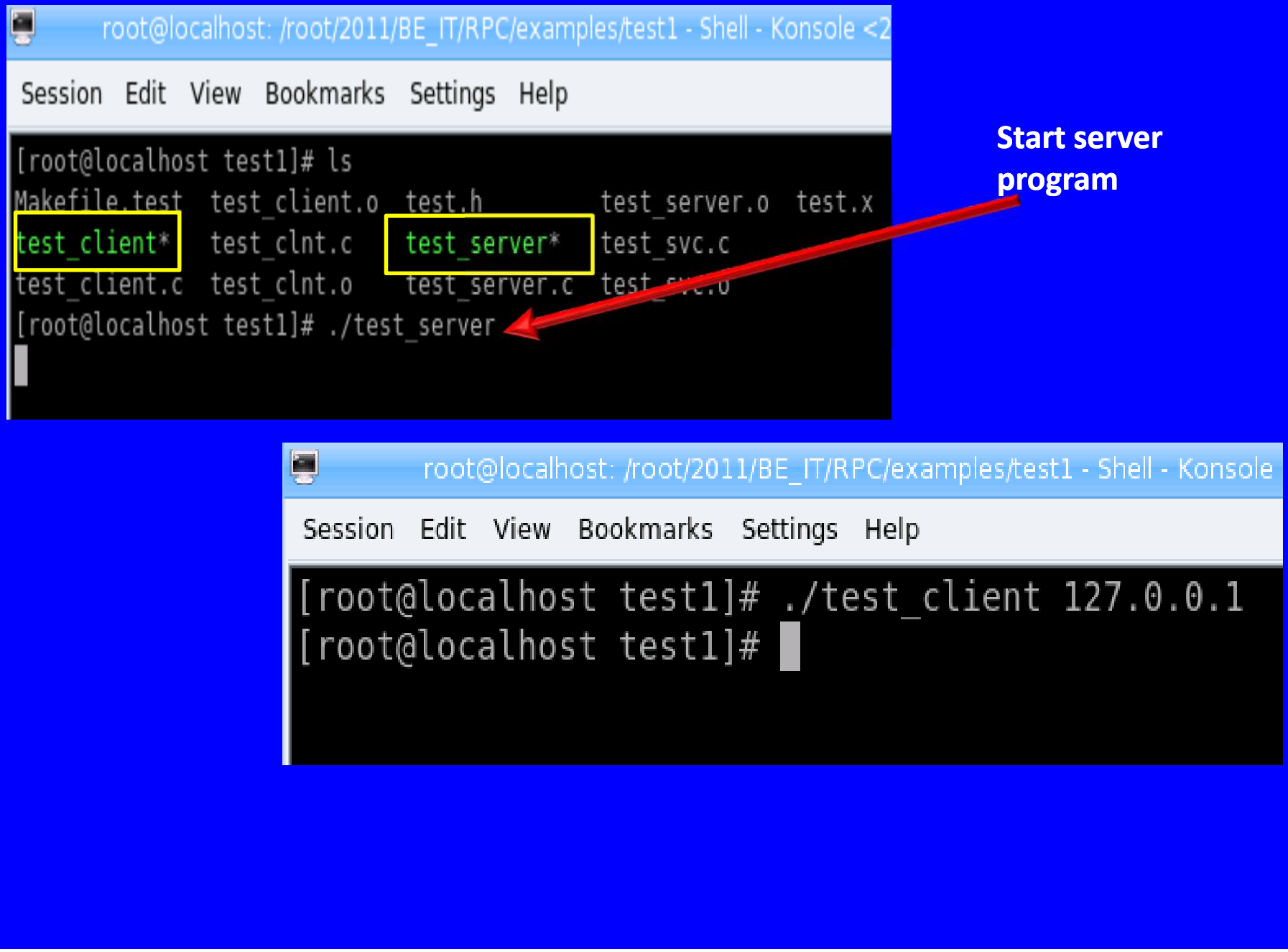
```c
int
main (int argc, char *argv[])
{
        char *host;

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

test_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;
}
```


root@localhost: /root/2011/BE_IT/RPC/examples/test1 - Shell - Konsole <2
Session Edit View Bookmarks Settings Help
[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: /root/2011/BE_IT/RPC/examples/test1 - Shell - Konsole
Session Edit View Bookmarks Settings Help
[root@localhost test1]# ./test_client 127.0.0.1
[root@localhost test1]#

Step - I

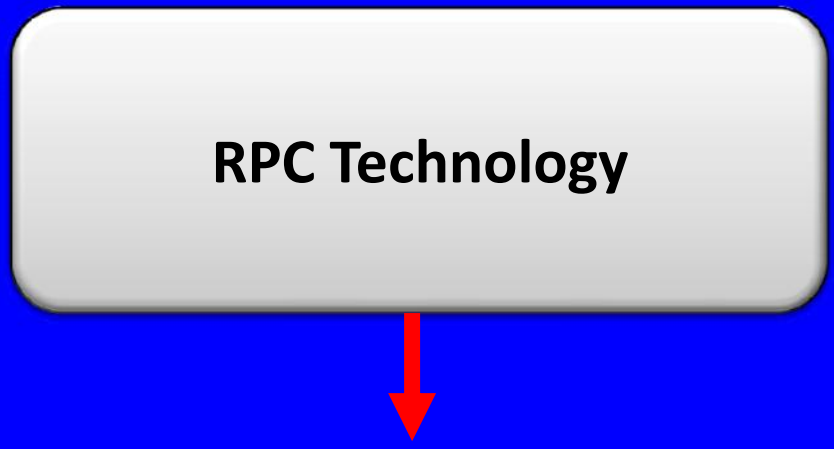
Remote Procedure Call: Second step

Theoretical Background

RPC: Introduction

What Is RPC ?

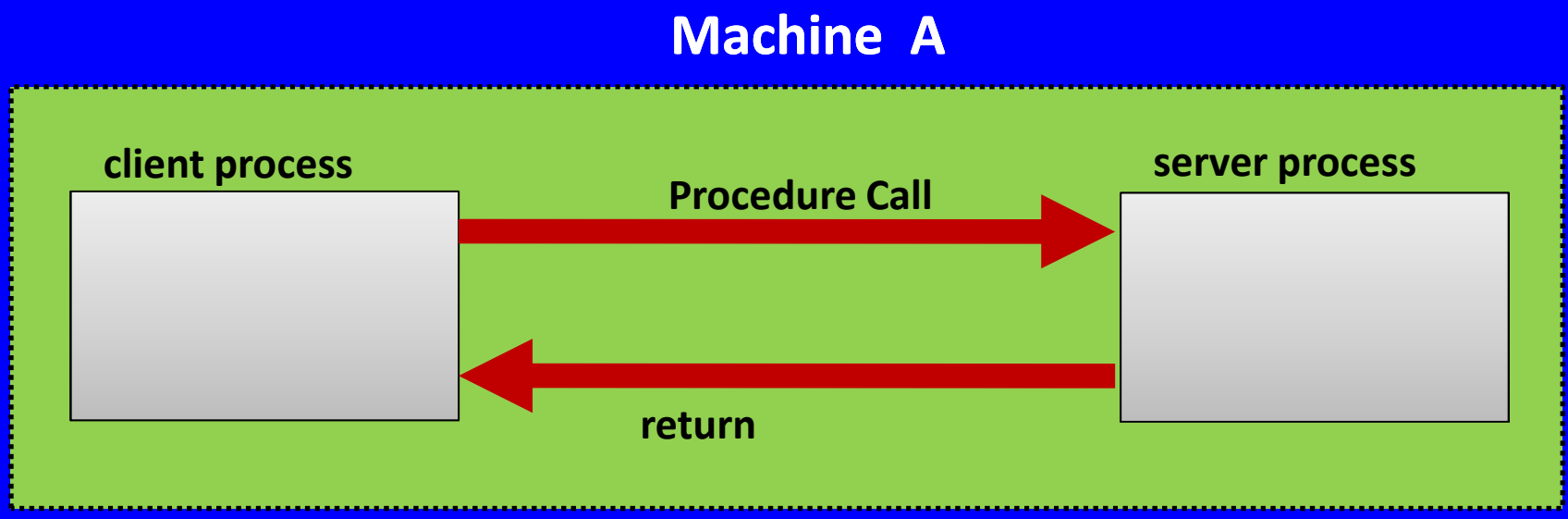
- RPC is a technique for constructing distributed, client-server based applications.



- Extending LPC: the calling procedure and called procedure need not exist in the same address space.

RPC on a single host

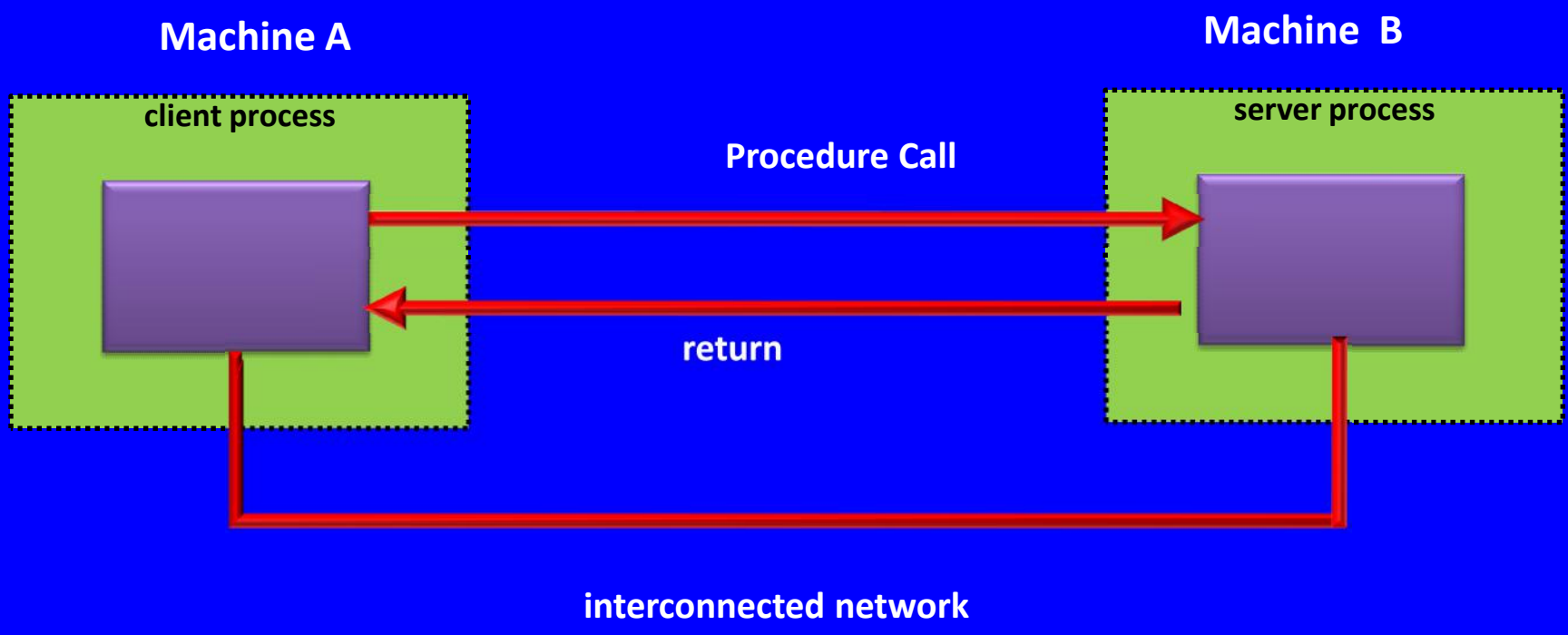
- Client and server processes are in separate address spaces.



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

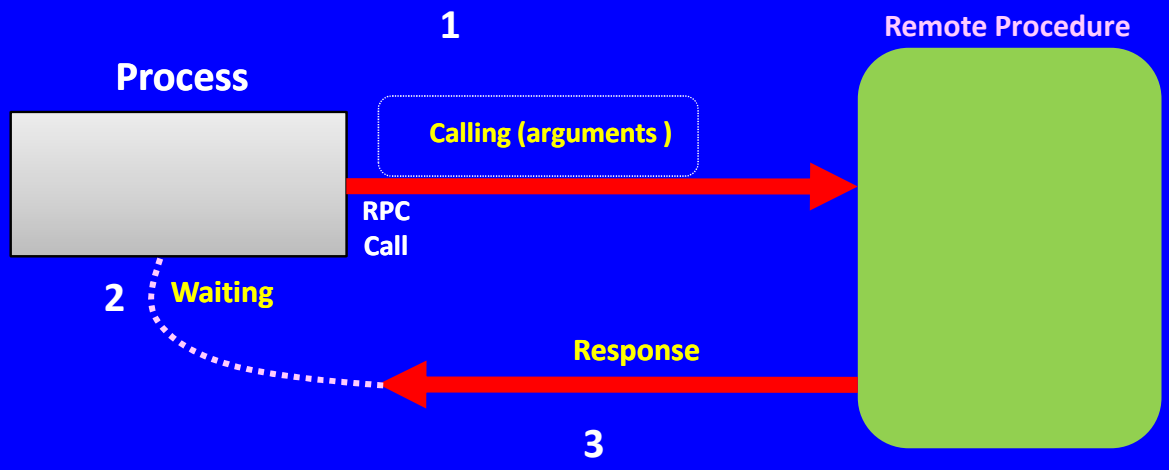
RPC between hosts

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

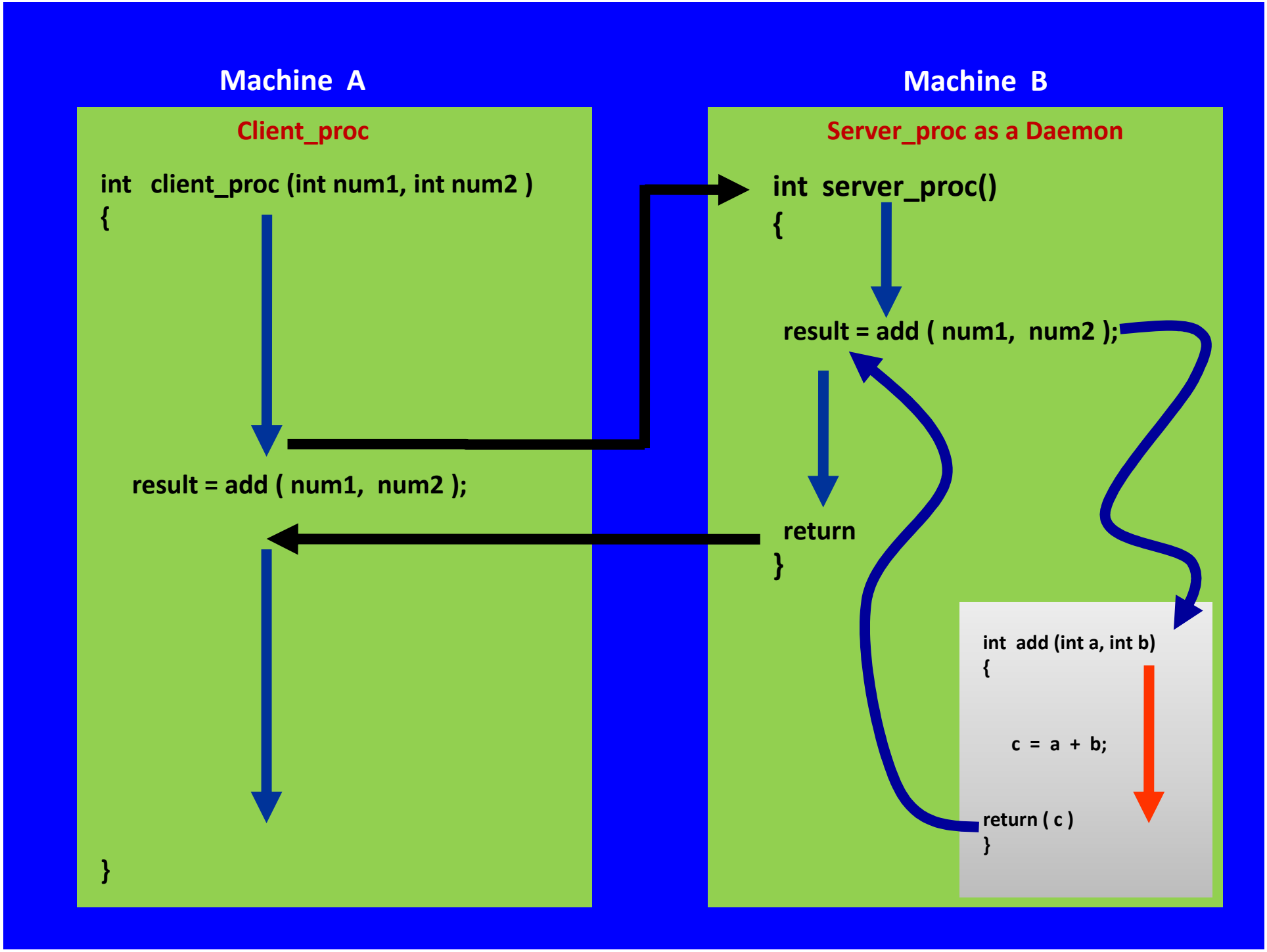


How RPC works ?

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

The RPC Model

- **Similar to that of the local model, which works as follows:**

Caller process

1. sends a call message to the server process and blocks for a reply message.

2. When the caller receives the reply message, it gets the results of the procedure.

3. The caller process then continues executing.

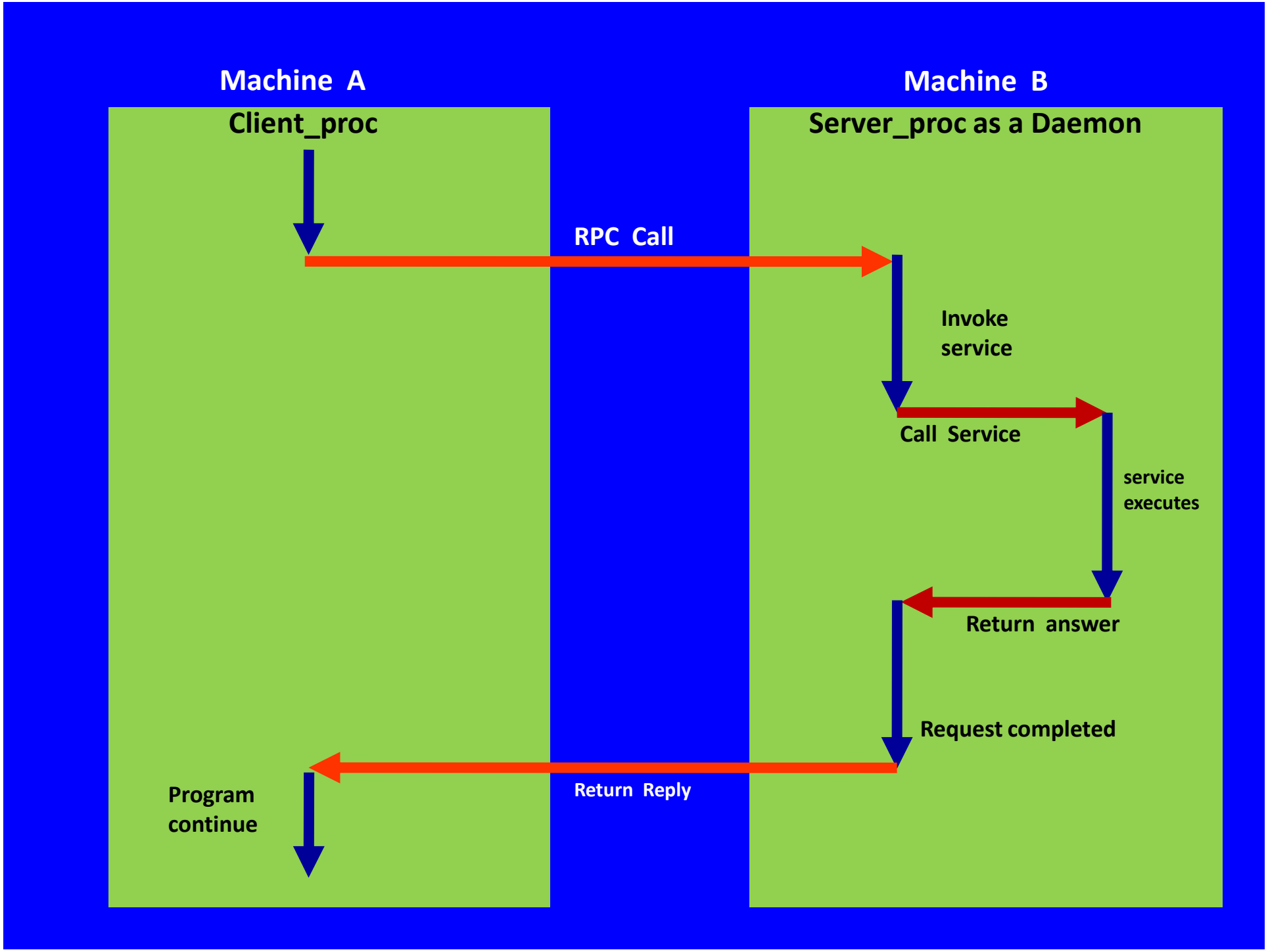
Server process

1. a process is dormant -- awaiting the arrival of a call message.

2. The server process computes a reply against call from the client.

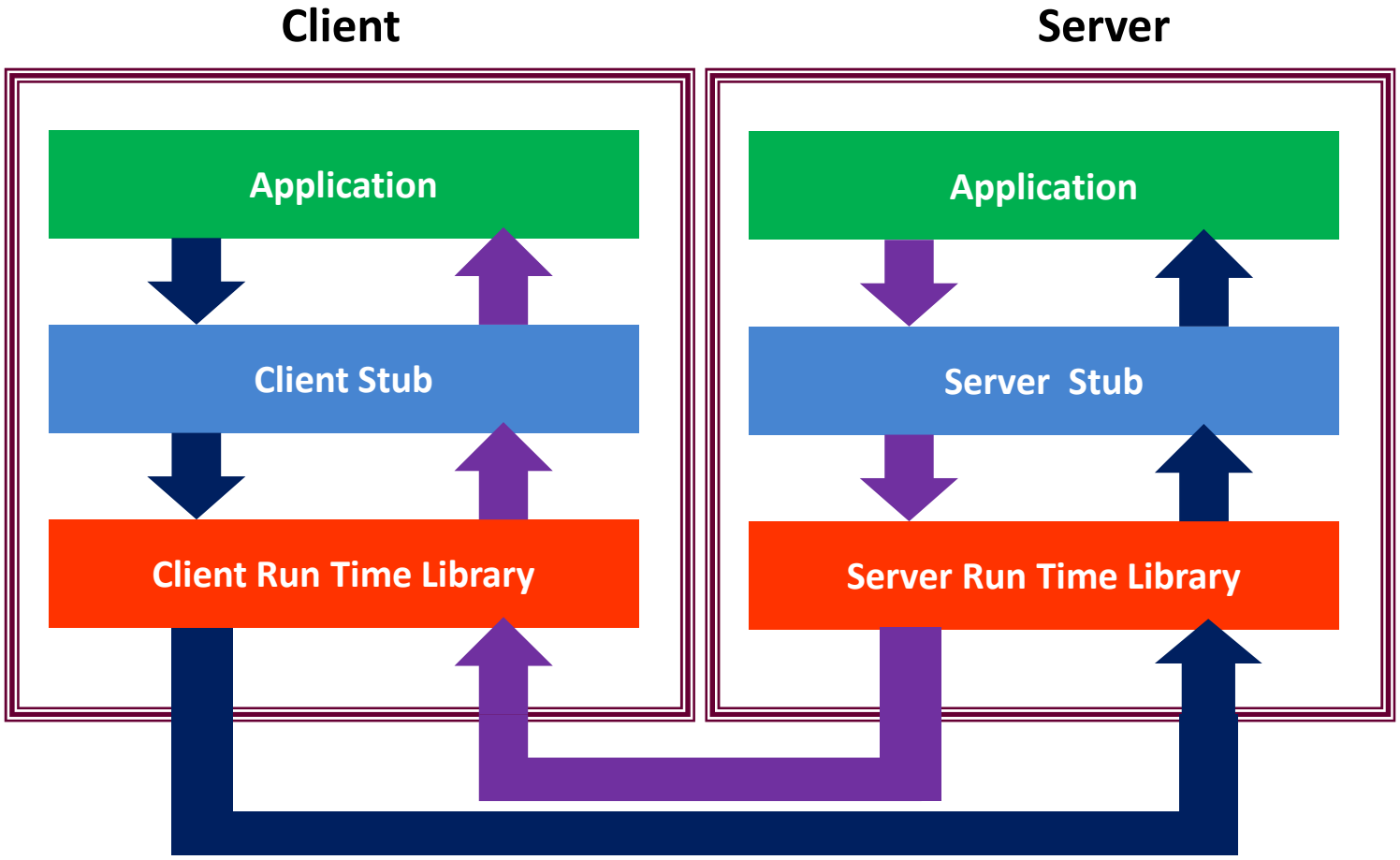
3. Sends reply back to the requesting client.

4. After this, the server process becomes dormant again.


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

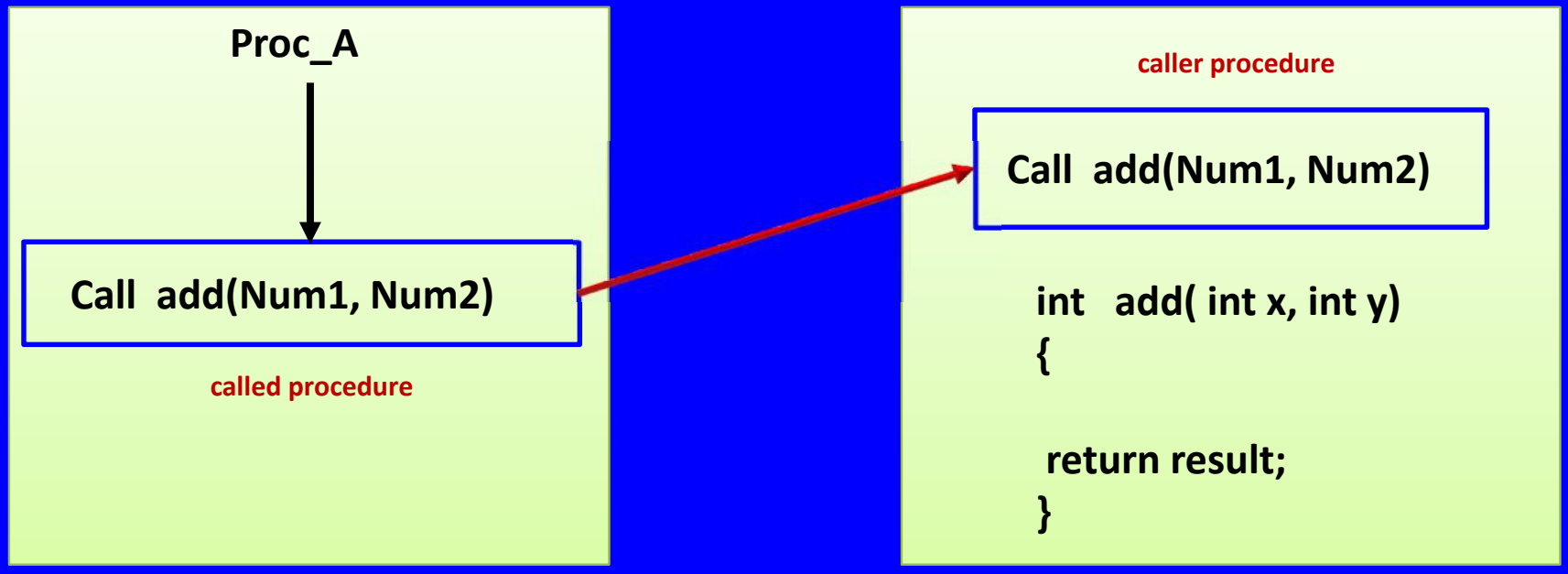
RPC Architecture

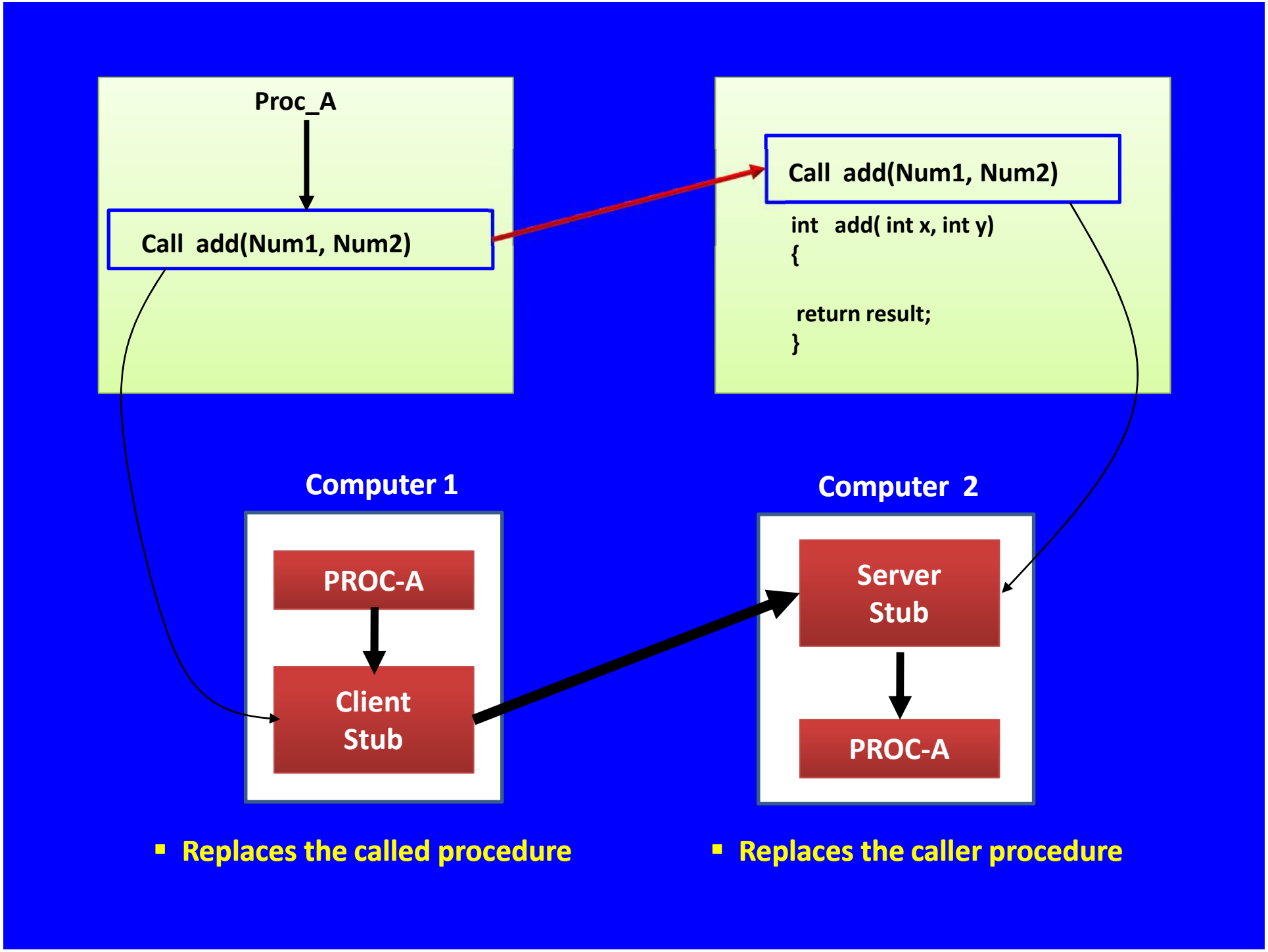
❖ **The client application calls a local stub procedure instead of the actual code implementing the procedure.**



Stubs are compiled and linked with the client application

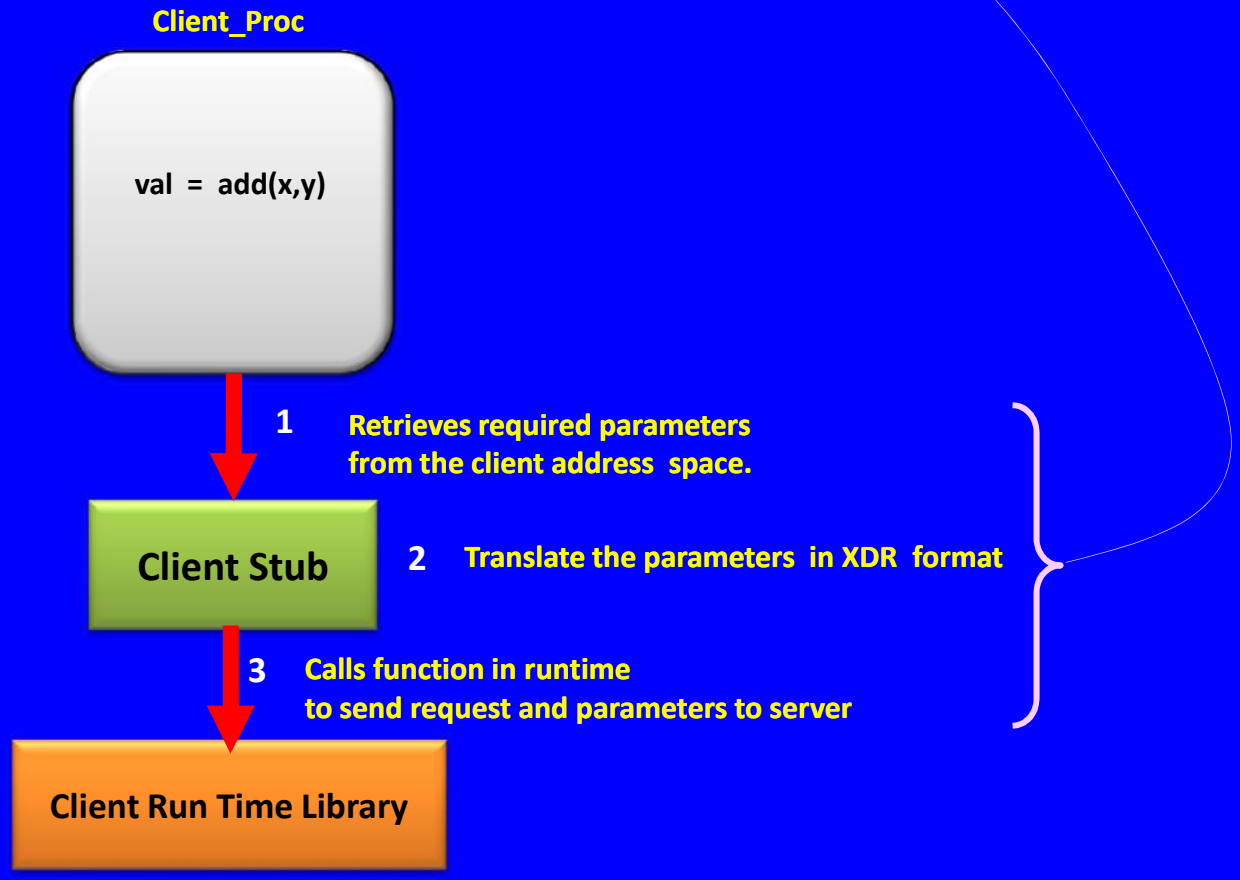
Stub procedures




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

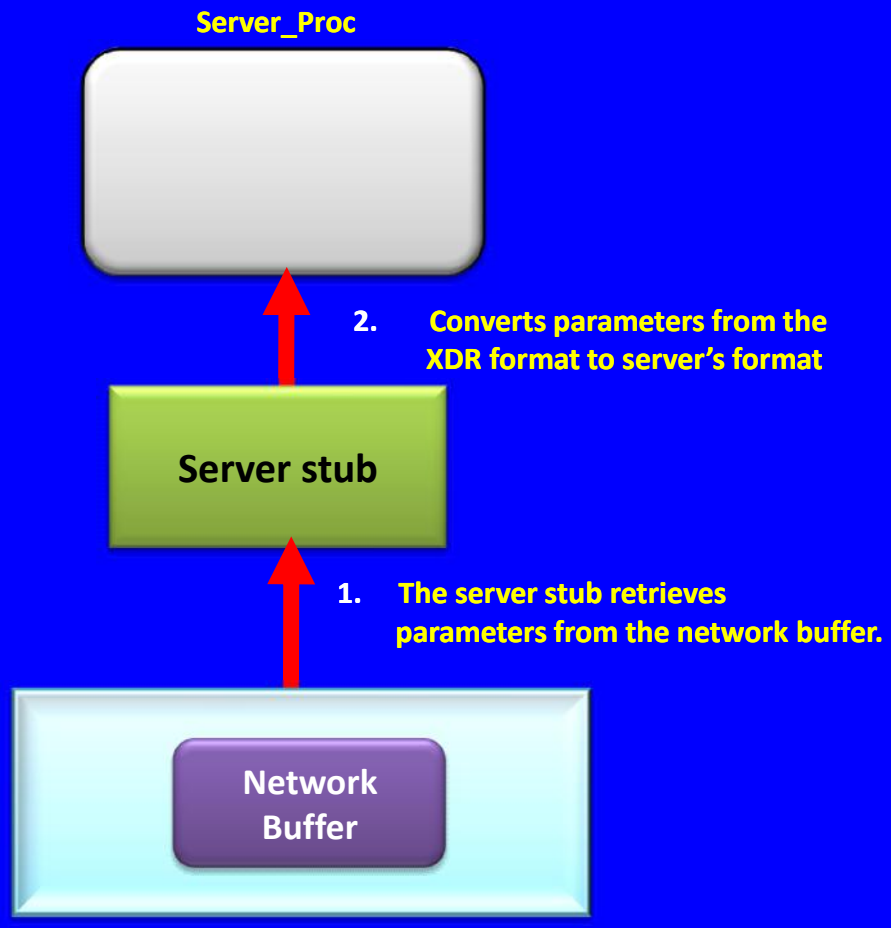
Client Stub

- The client stub code contains (instead of actual procedure code)

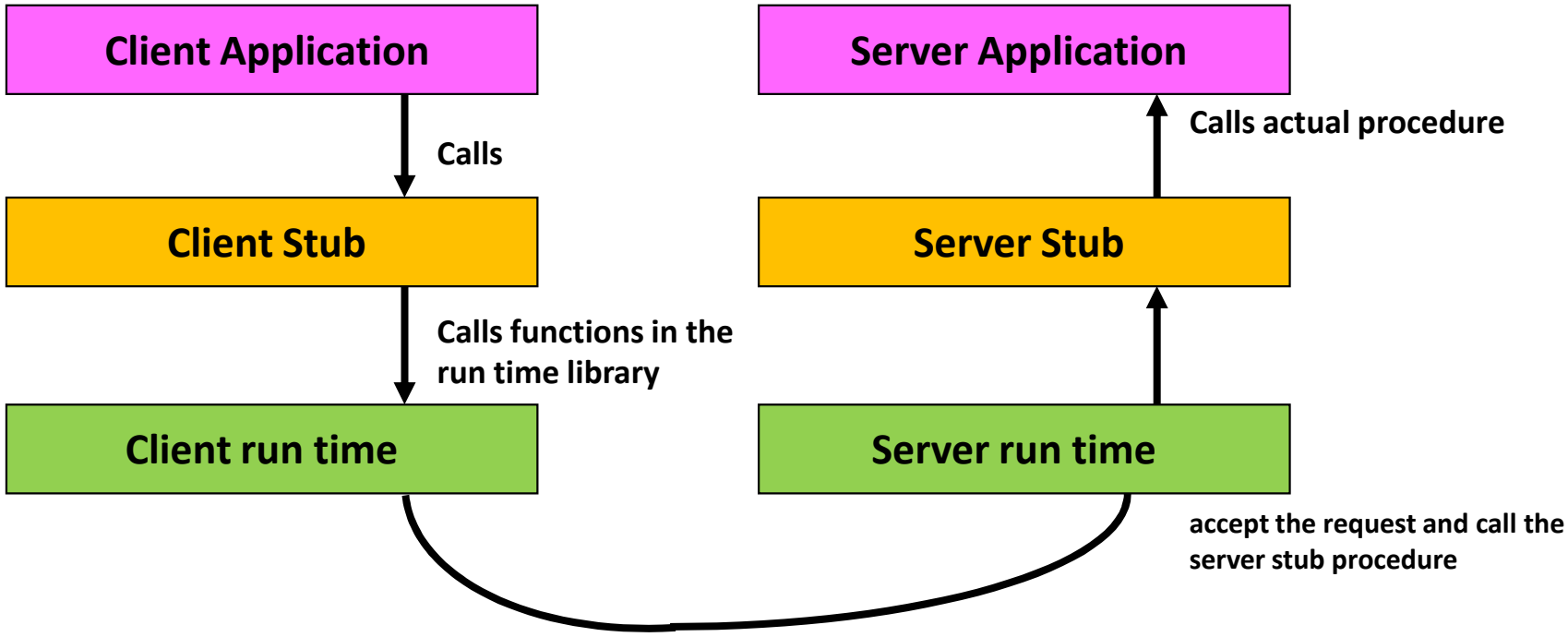


Server Stub

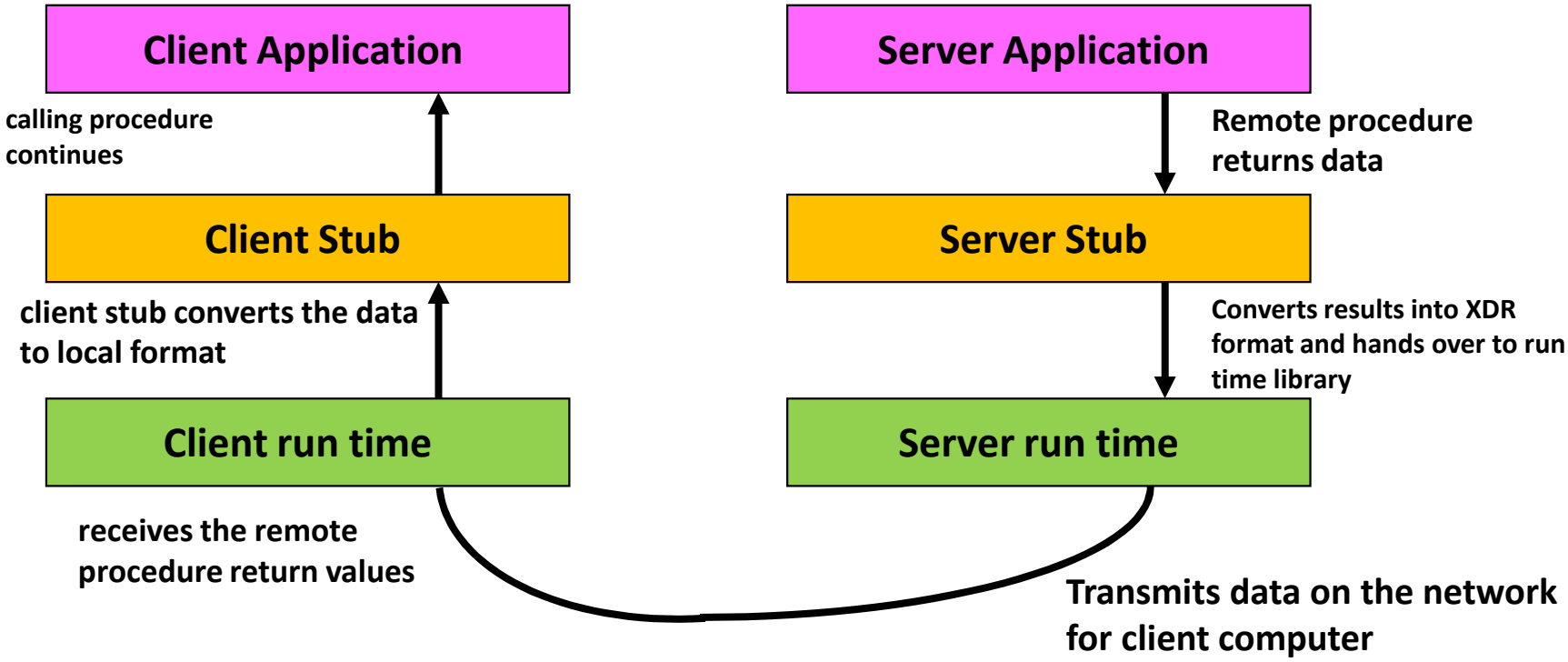
- Server stub performs



RPC calling sequence



RPC return sequence

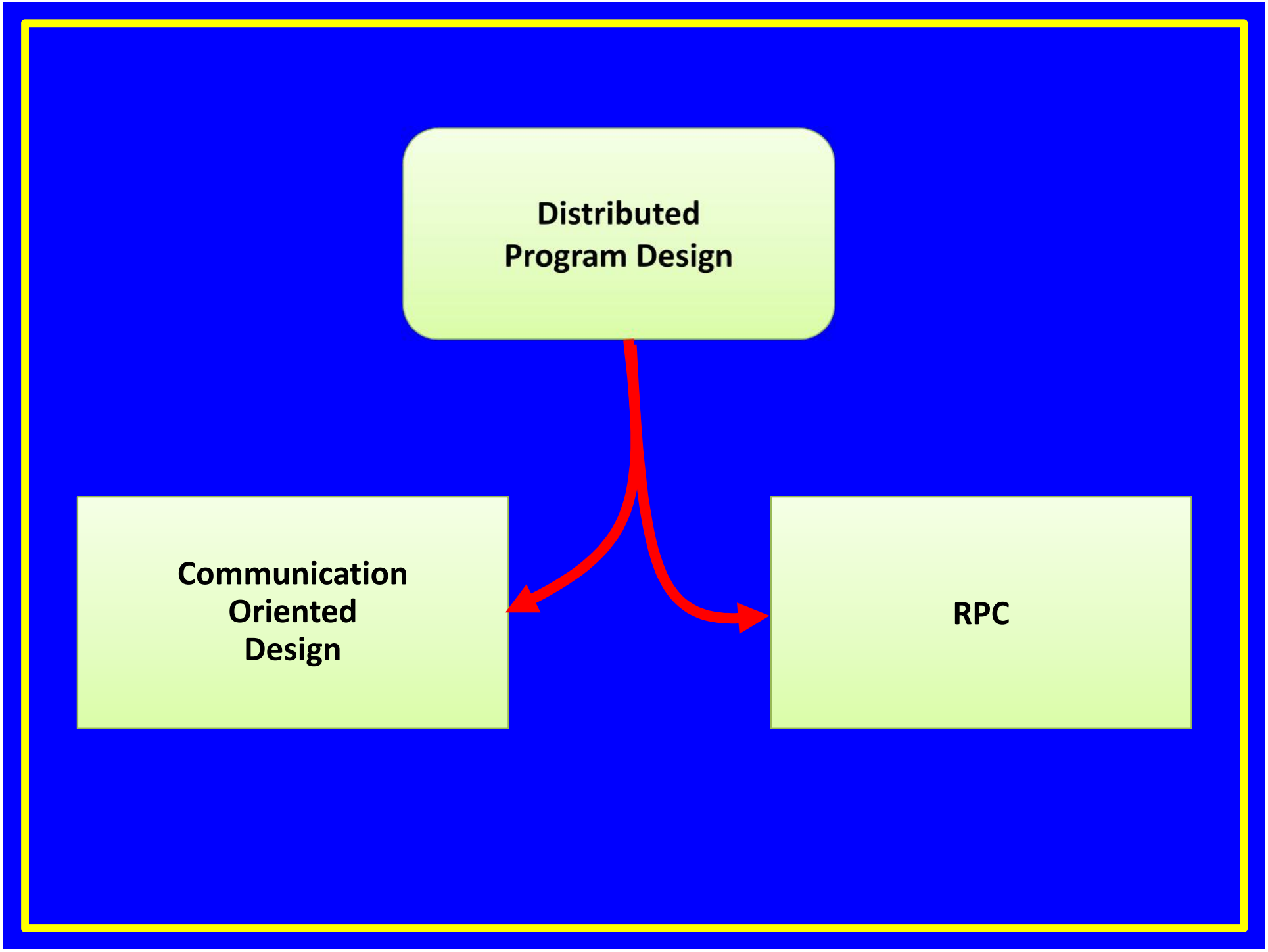


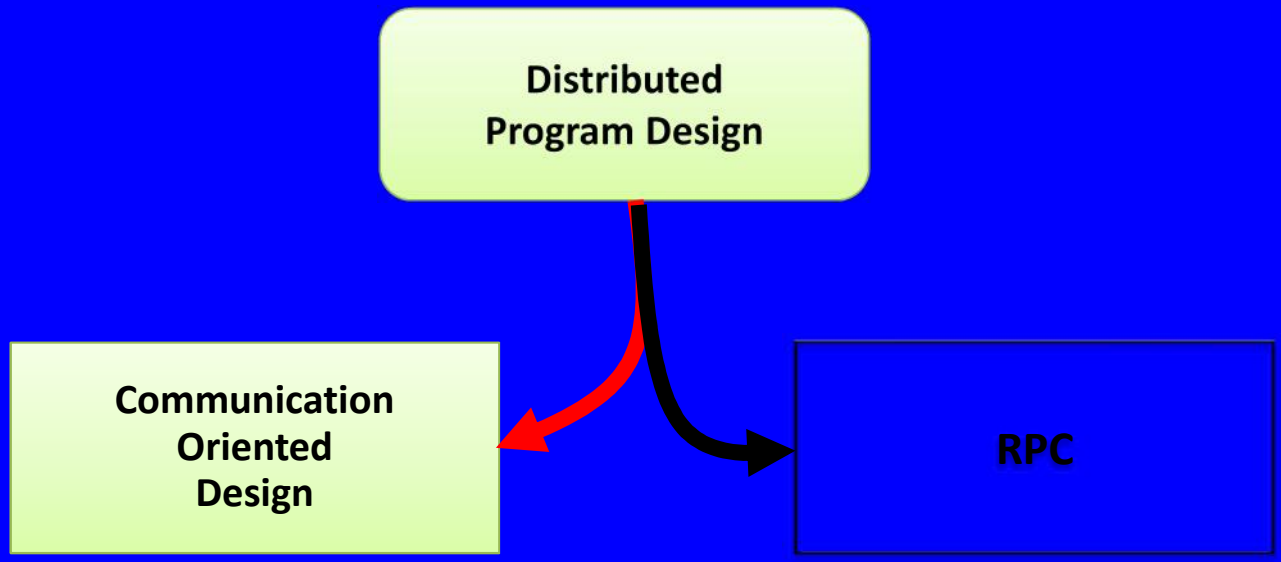
Step - II

Remote Procedure Call: Third step

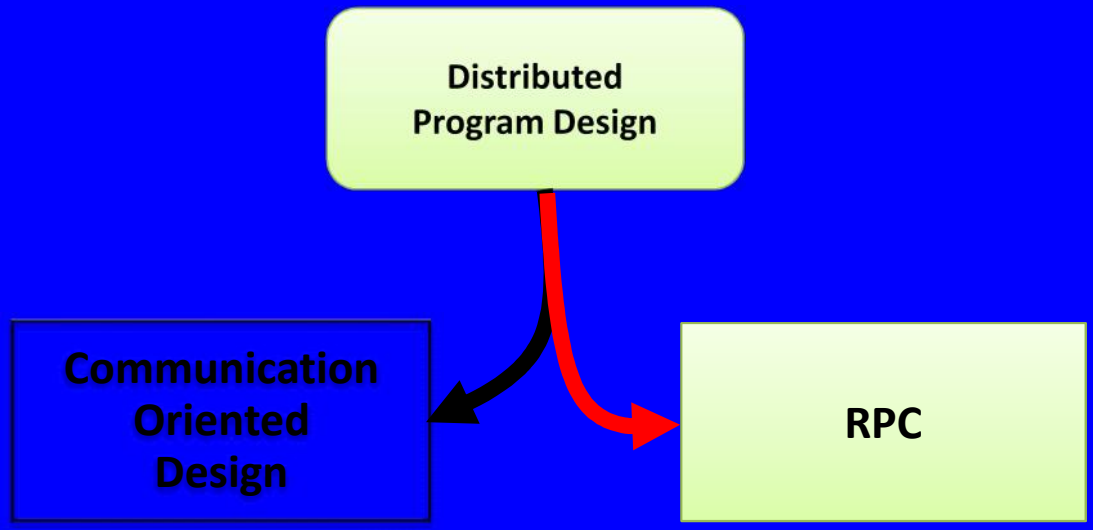
Sun RPC

Sun RPC (ONC RPC)


Distributed
Program Design
Communication
Oriented
Design
RPC



- **Typical socket approach:**
  - **Design protocol first.**
  - **Build programs based upon the protocol.**



- **RPC**
  - **Build application (write stand alone application)**
  - **Divide programs up and add communication protocols.**

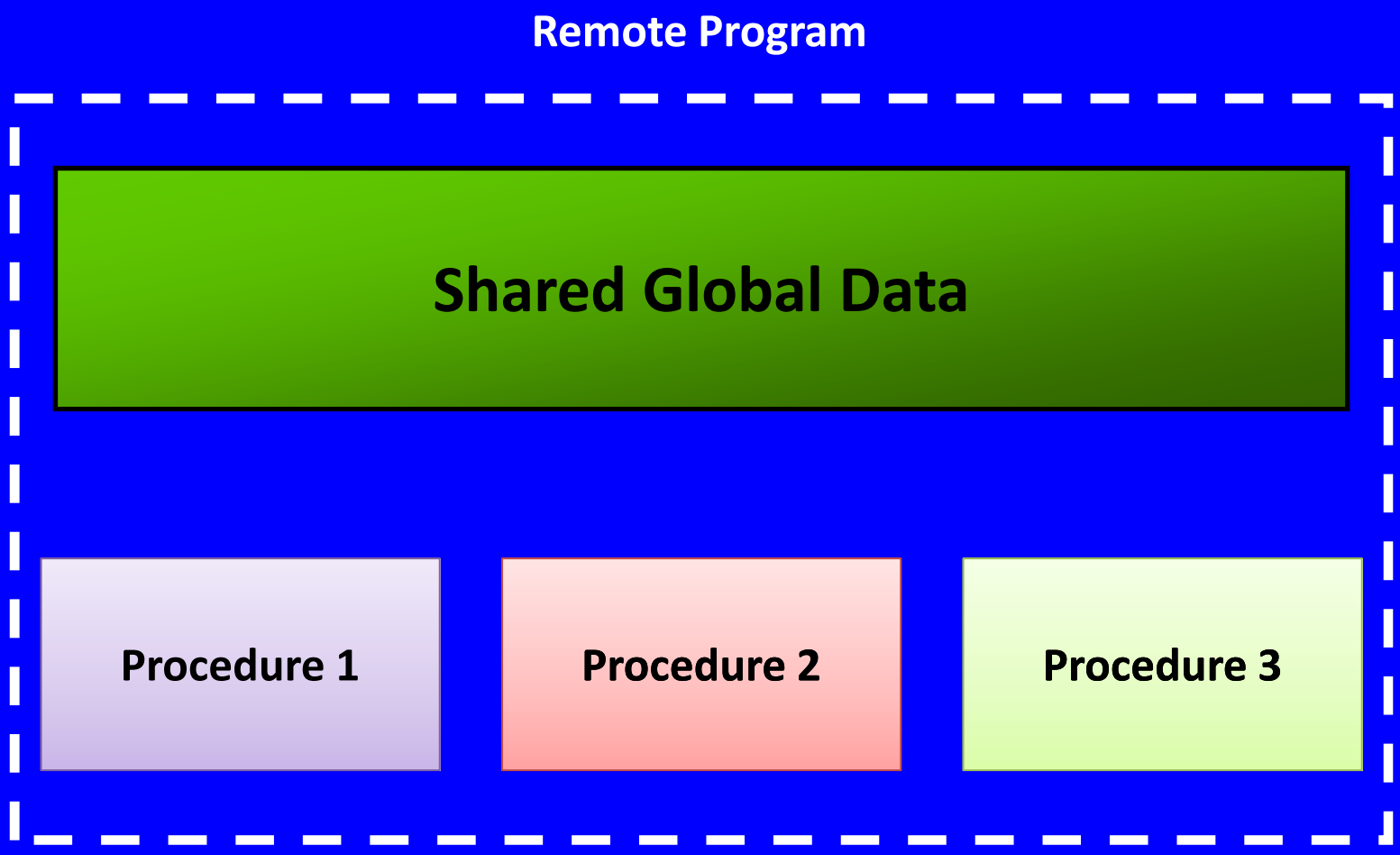
RPC issues

- 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

Sun RPC Organization



Procedure Arguments

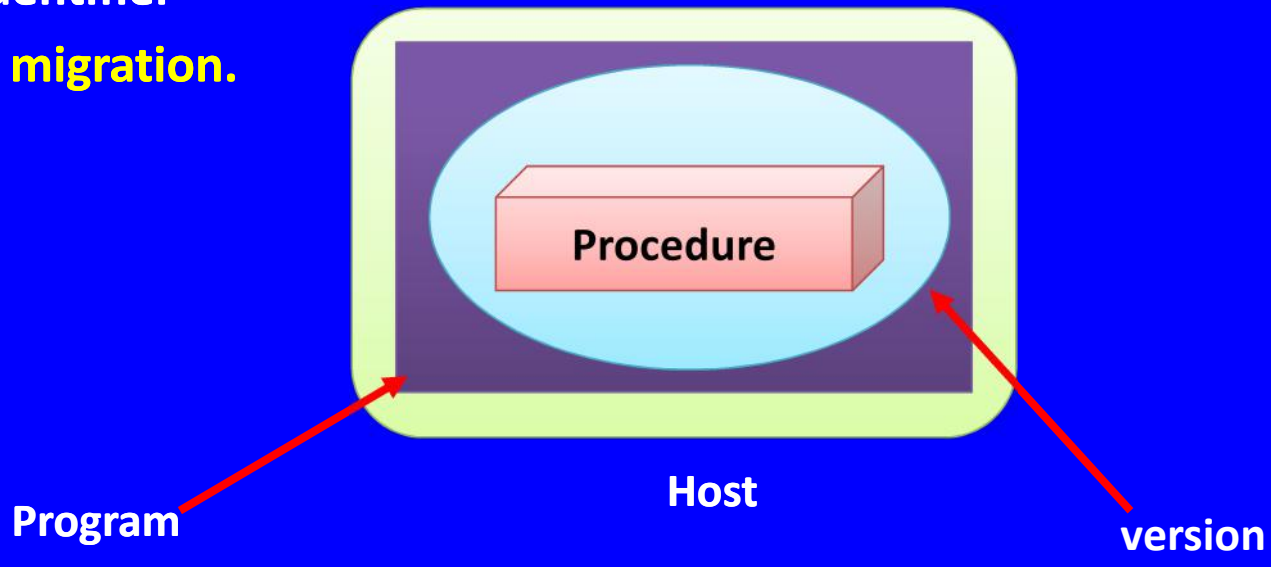
- **Single argument:** Sun RPC includes support for a single argument to a remote procedure.
- Typically the single argument is a structure that contains a number of values.

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

Call proc_add( struct Num);
```

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

Procedure Identification

- Each procedure is identified by:
  - Hostname (IP Address)
  - Program identifier
  - Procedure identifier
- Program Version identifier
  - for testing and migration.



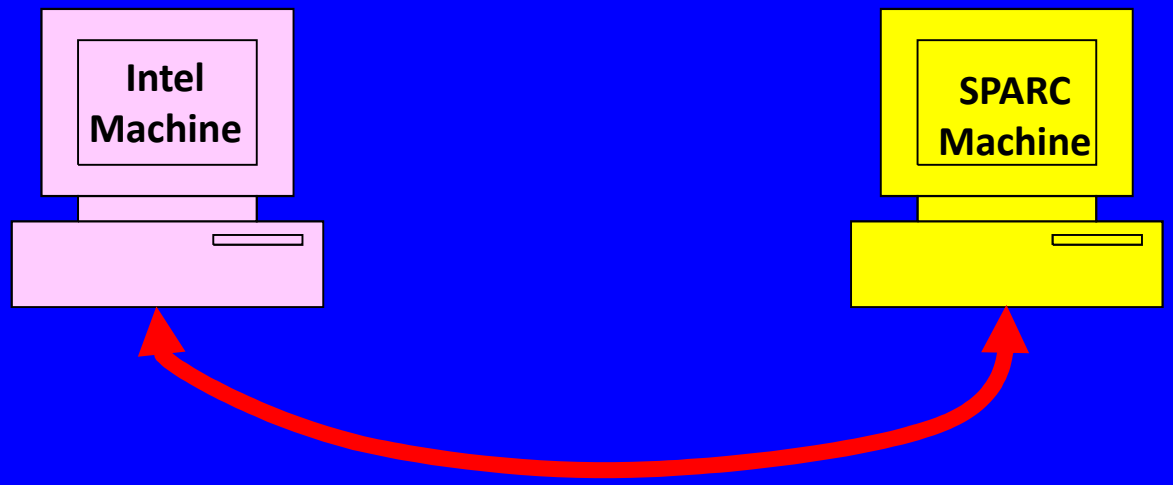
Assigning Program Numbers

- **Program numbers are assigned in groups of 0x20000000 according to the following chart:**

0x0 - 0x1fffffff	Defined by Sun
0x20000000 - 0x3fffffff	Defined by user
0x40000000 - 0x5fffffff	Transient
0x60000000 - 0x7fffffff	Reserved
0x80000000 - 0x9fffffff	Reserved
0xa0000000 - 0xbfffffff	Reserved
0xc0000000 - 0xdfffffff	Reserved
0xe0000000 - 0xffffffff	Reserved

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.

Finding a Remote Procedure

- 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.

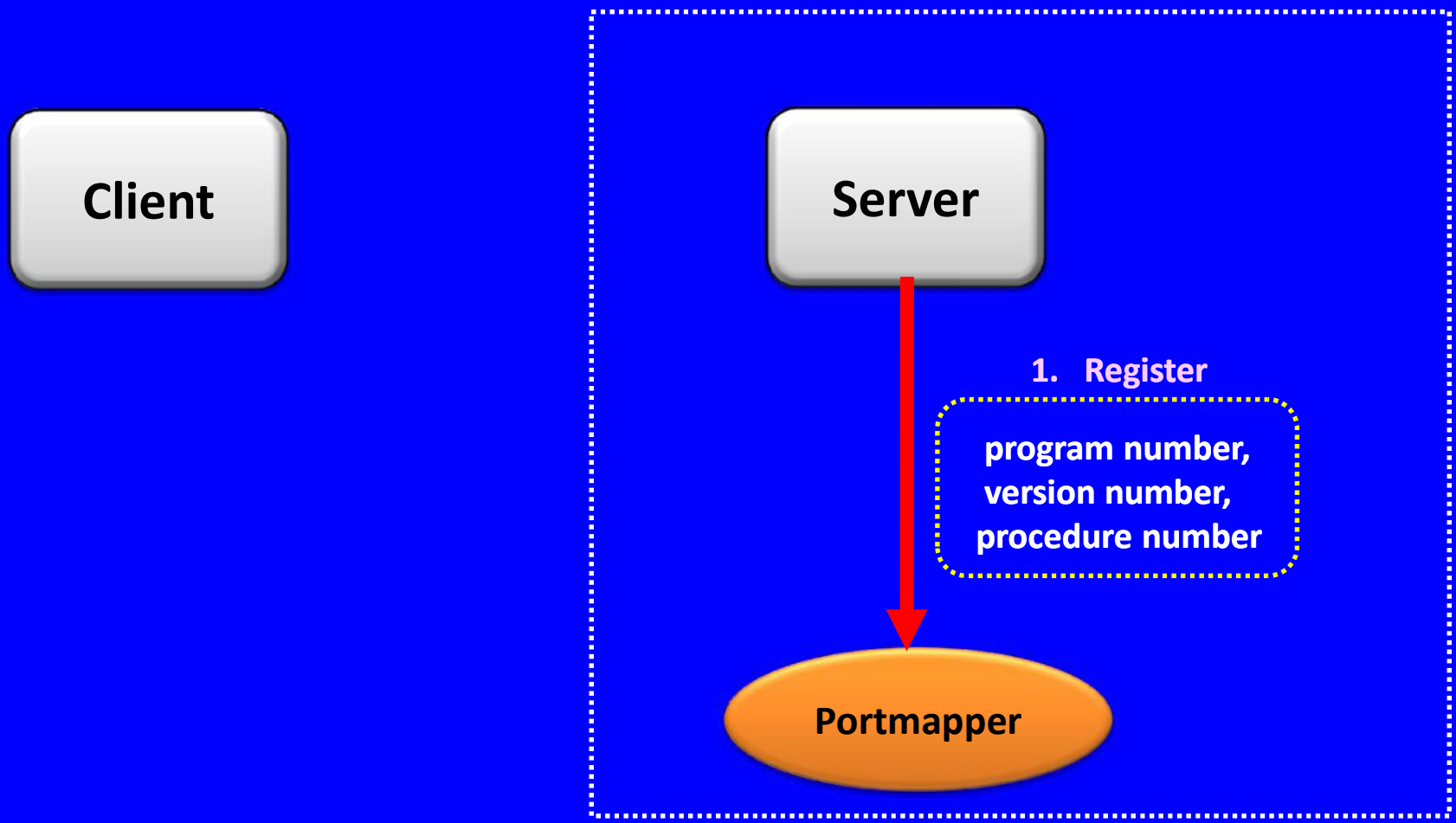
  - ❑ RPC: the user only supplies a host name

Portmapper (rpcbind)

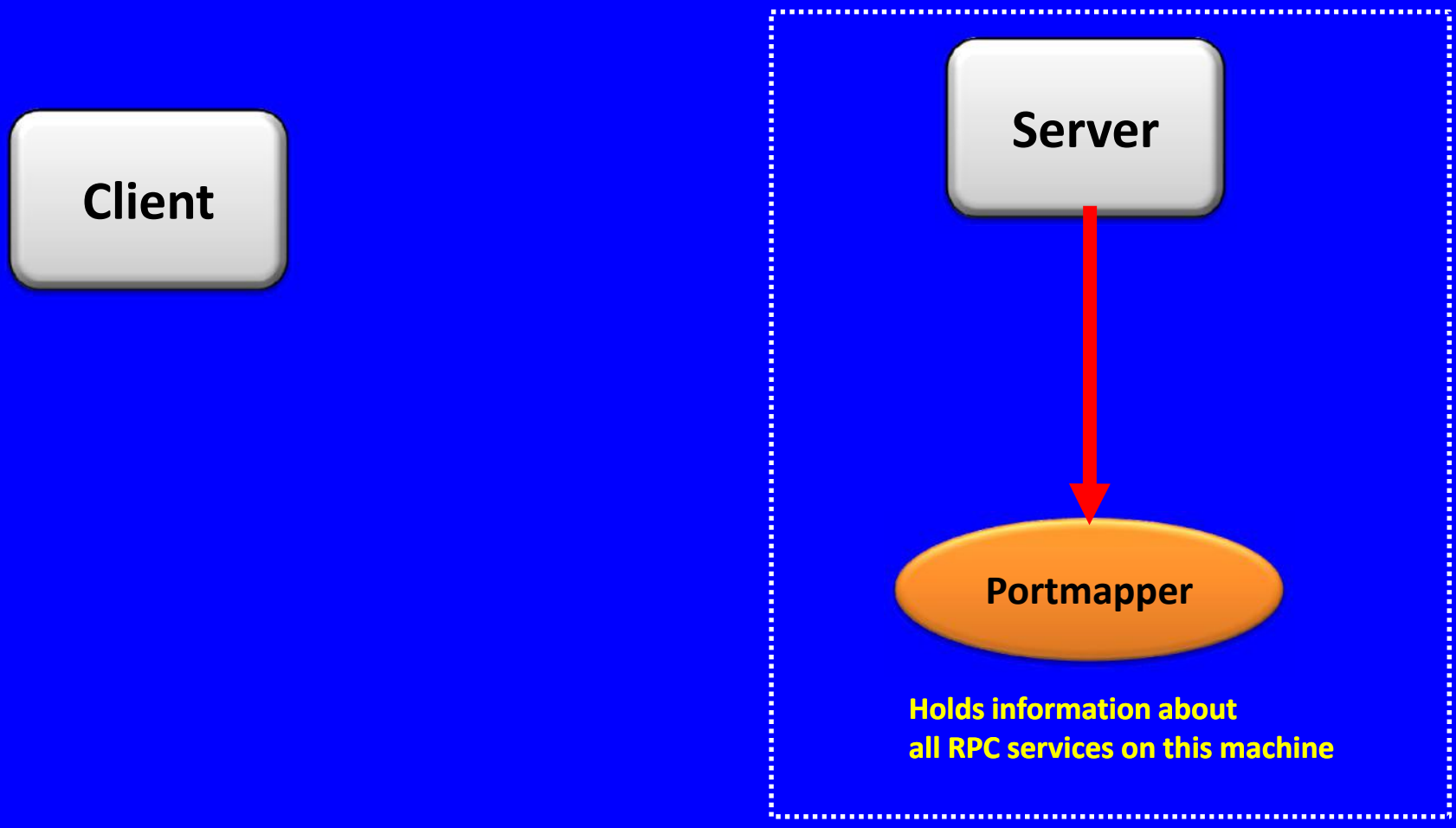
- Each system (RPC servers) runs a port mapper server.
- Portmapper provides a central registry for RPC services.
- Servers tell the port mapper what services they offer.

Steps in RPC Communication

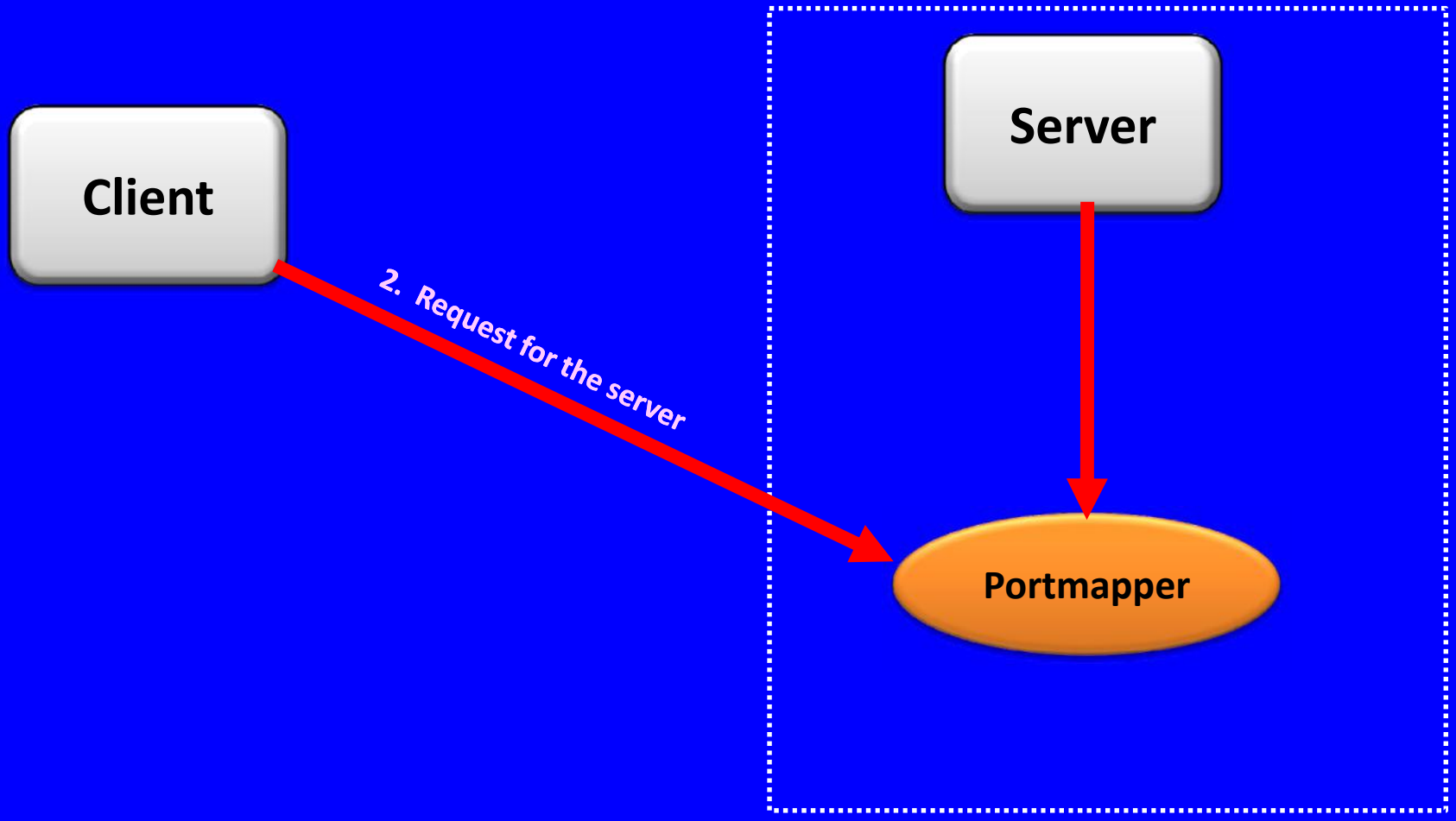
❖ Each server registers itself with the portmapper when it first starts.



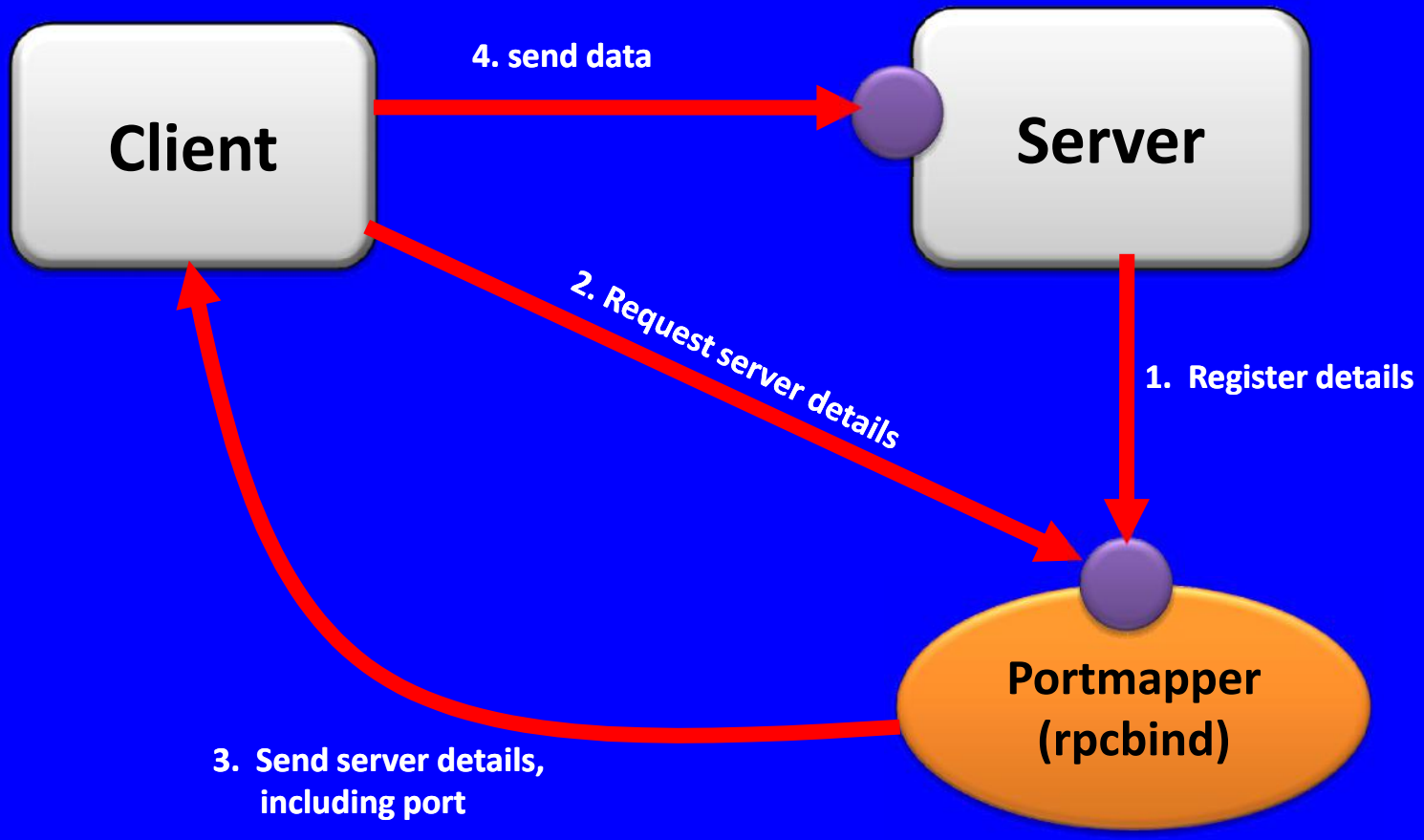
- portmapper holds a database of RPC services



❖ **The client asks to the portmapper for the port of the server.**



Steps in RPC Communication



Introduction to ONC RPC programming

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

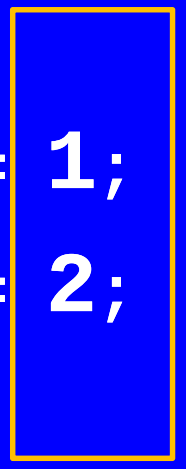
- Each remote program executing on a computer must be assigned a unique 32 – bit integer that the caller uses to identify it.

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

0x20000000 - 0x3fffffff

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

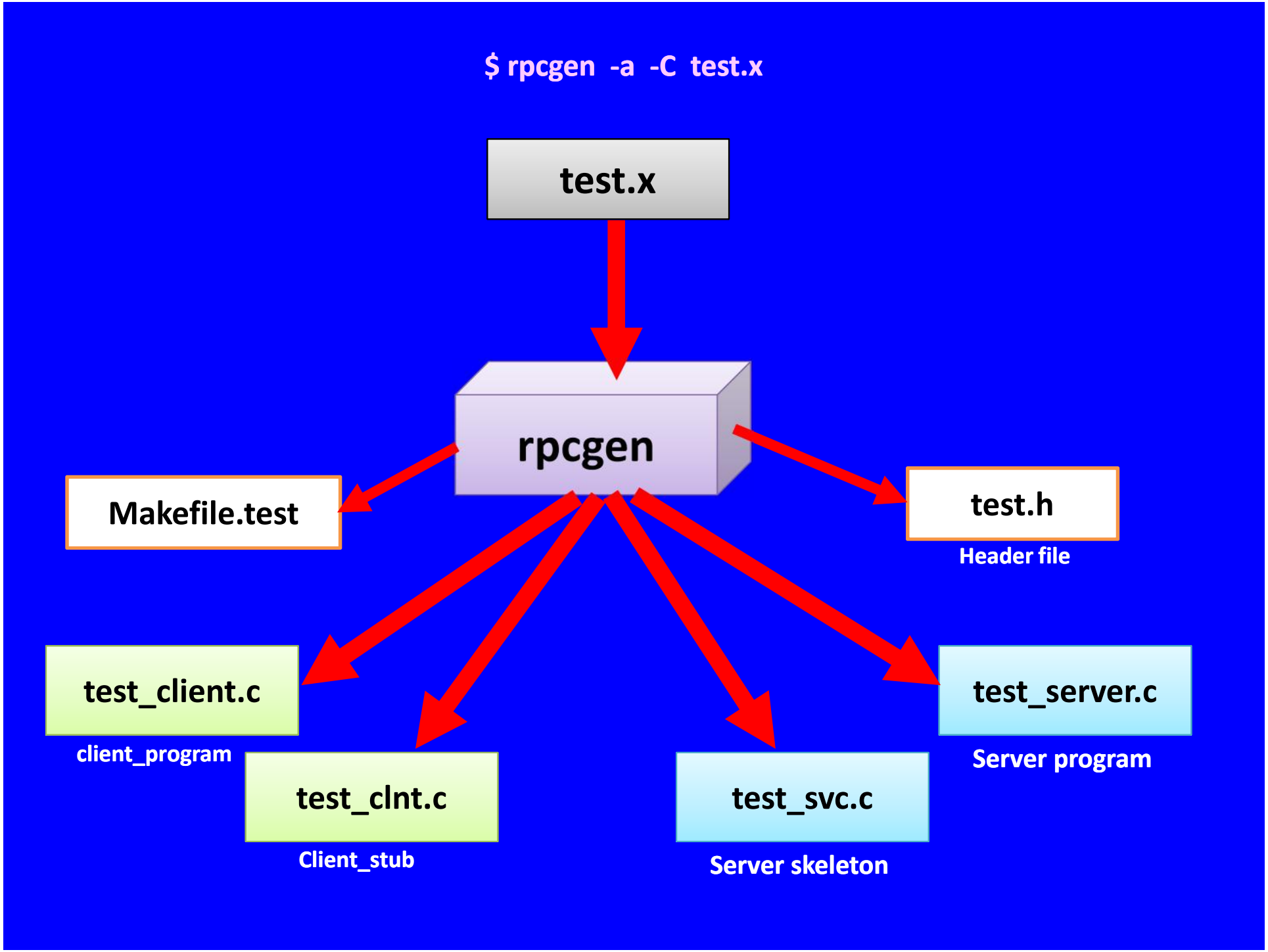
```
Program DEMO_PROG {

      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");
```

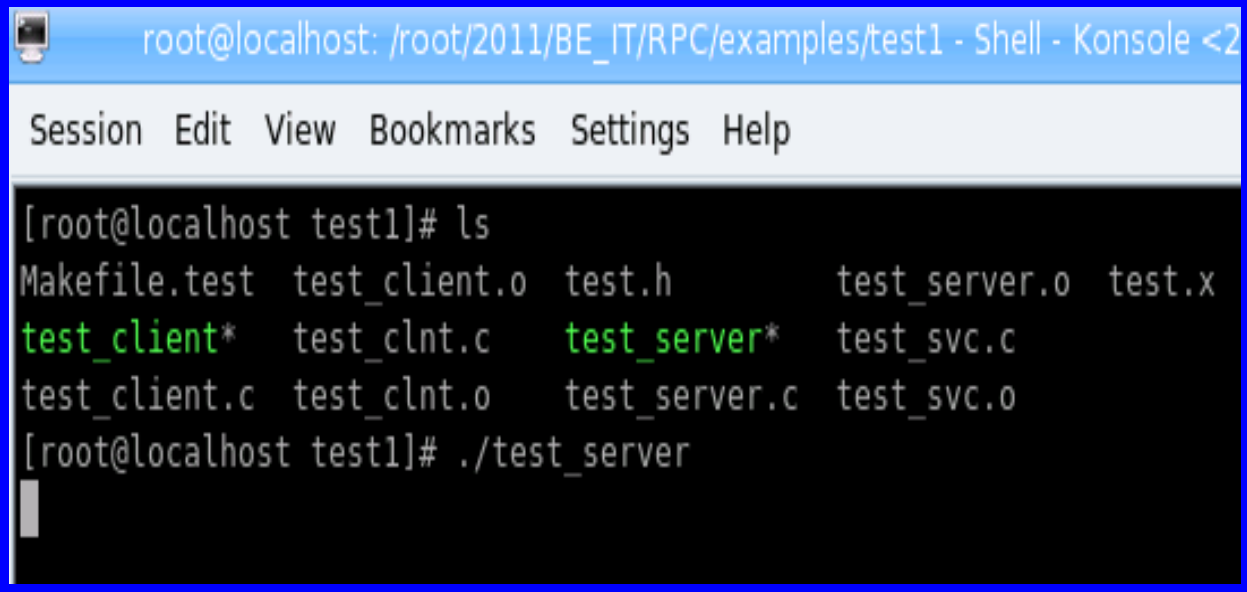
test_server.c

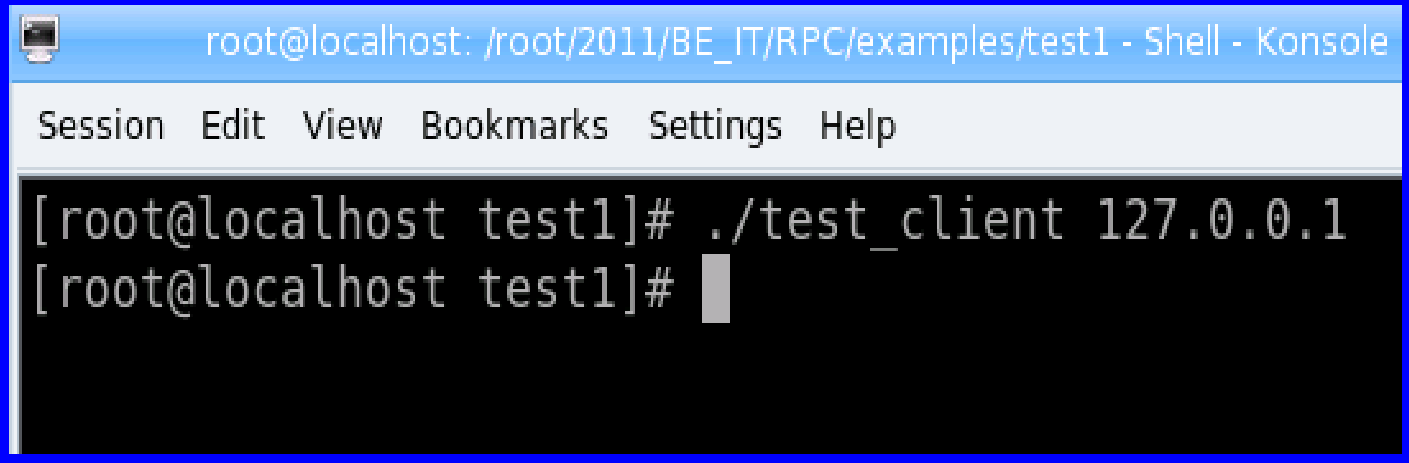
```c
#include "test.h"

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

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

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

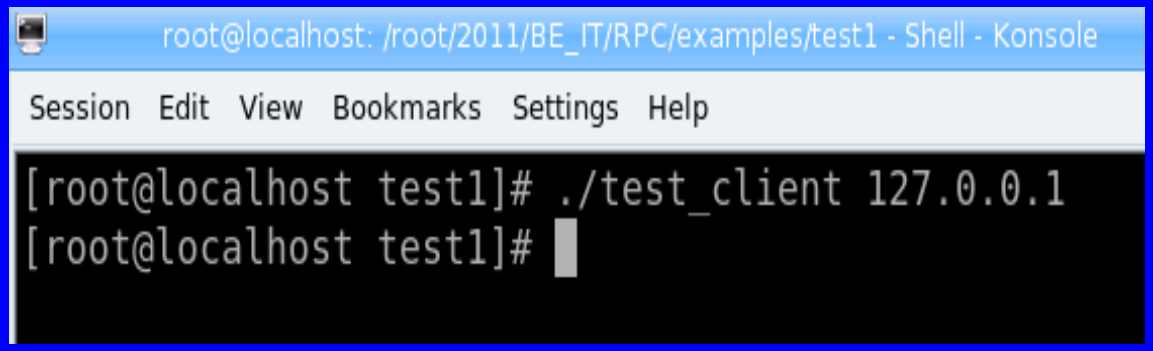
Step 5. Run programs

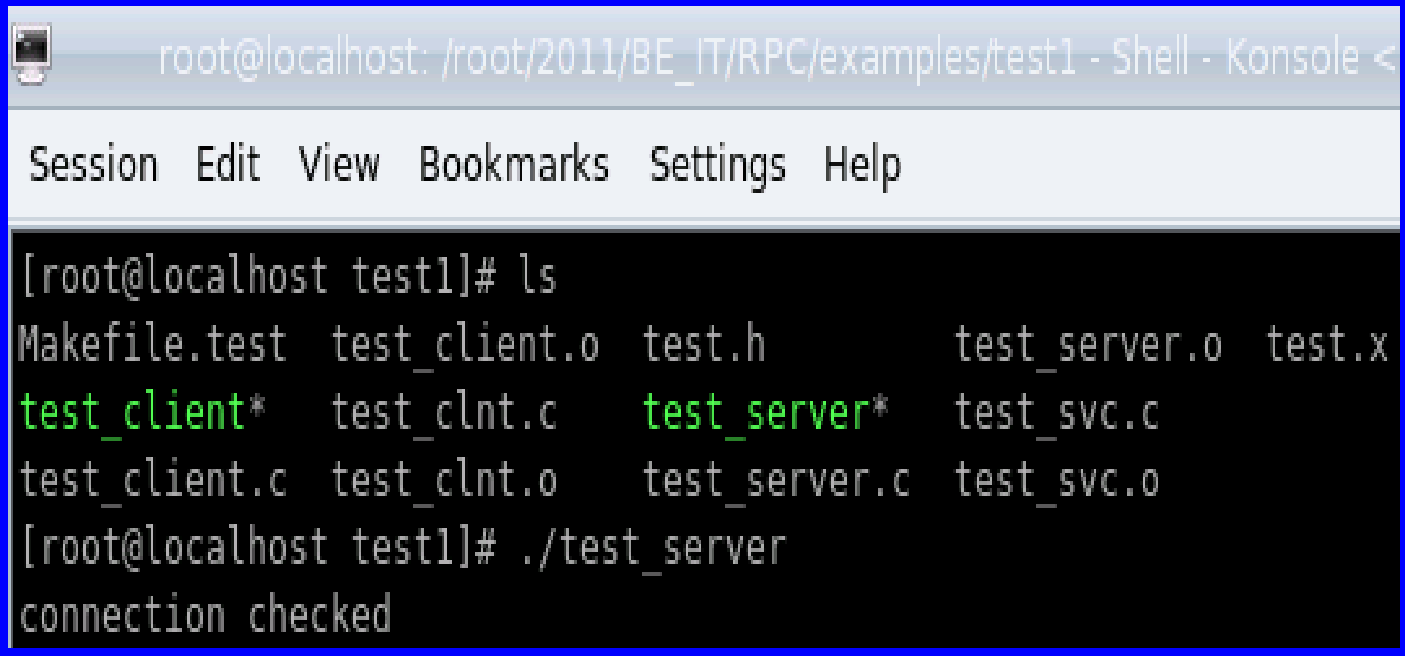
- Build using make

```
[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
```


root@localhost: /root/2011/BE_IT/RPC/examples/test1 - Shell - Konsole <2
Session Edit View Bookmarks Settings Help
[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


root@localhost: /root/2011/BE_IT/RPC/examples/test1 - Shell - Konsole
Session Edit View Bookmarks Settings Help
[root@localhost test1]# ./test_client 127.0.0.1
[root@localhost test1]#


root@localhost: /root/2011/BE_IT/RPC/examples/test1 - Shell - Konsole
Session Edit View Bookmarks Settings Help
[root@localhost test1]# ./test_client 127.0.0.1
[root@localhost test1]#


root@localhost: /root/2011/BE_IT/RPC/examples/test1 - Shell - Konsole
Session Edit View Bookmarks Settings Help
[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
connection checked

Step - III

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

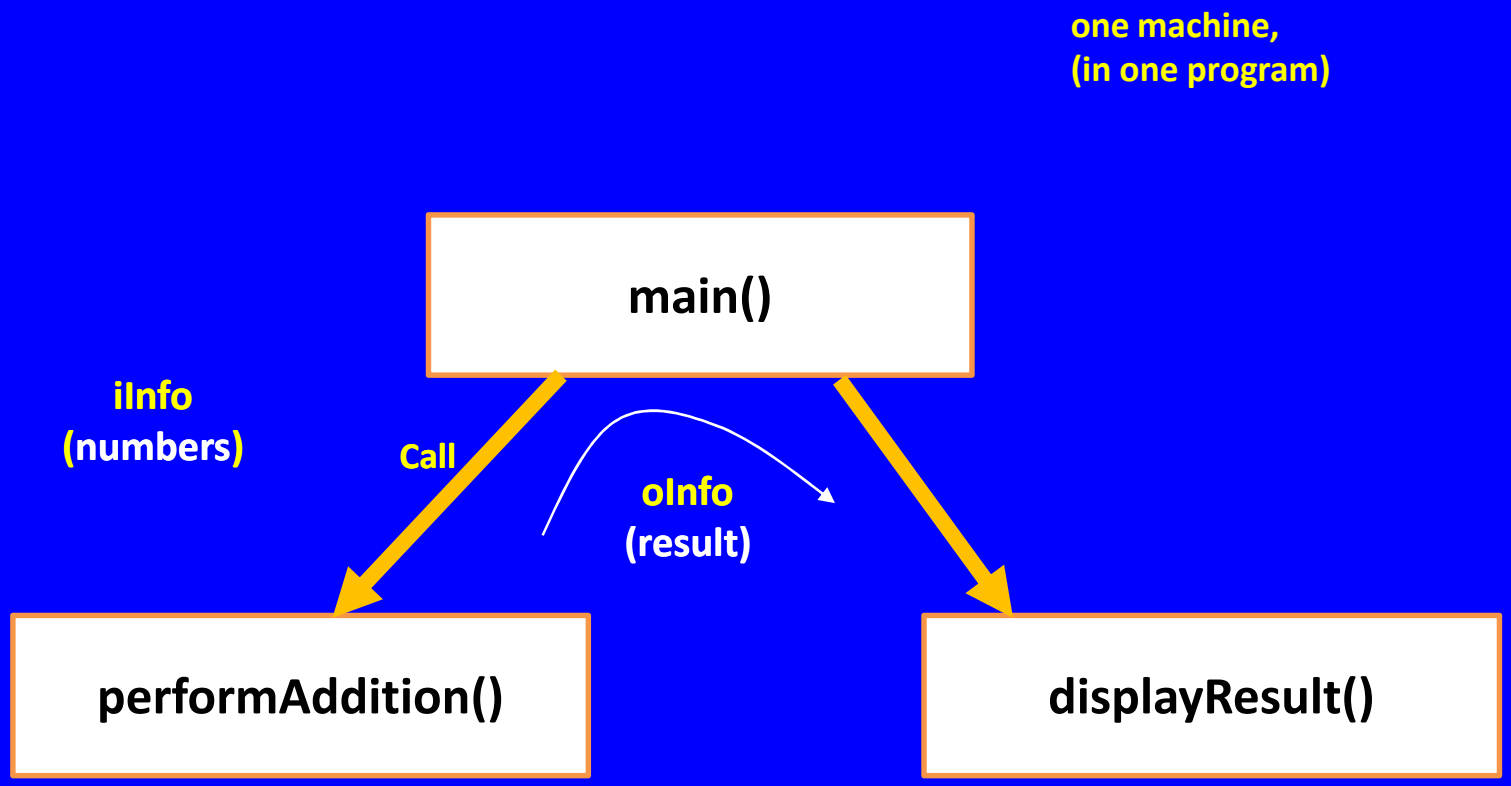
```
struct InputInfo {
        int num1;
        int num2;
};
```

oInfo

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

```
void displayResult(struct OutputInfo oInfo);

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

Flow of Functions



add.c

```
#include<stdio.h>

struct InputInfo {
        int num1;
        int num2;
};

struct OutputInfo {
        int result;
};


void displayResult(struct OutputInfo oInfo);
struct OutputInfo performAddition(struct InputInfo iInfo);
```

```
int main(int argc, char* argv[])
{
        int n1, n2;
        struct InputInfo iInfo;
        struct OutputInfo oInfo;

        if(argc != 3){
                printf("usage: ./add num1 num2\n");
                exit(-1);
        }

        n1 = atoi(argv[1]);
        n2 = atoi(argv[2]);
        iInfo.num1 = n1;
        iInfo.num2 = n2;
        oInfo = performAddition(iInfo);
        displayResult(oInfo);
return 0;
}
```

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

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

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

```
struct   OutputInfo   performAddition( struct  InputInfo  iInfo )
{
        struct   OutputInfo oInfo;
        int       sum;

        sum = iInfo.num1 + iInfo.num2;

        oInfo.result = sum;

return oInfo;
}
```

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

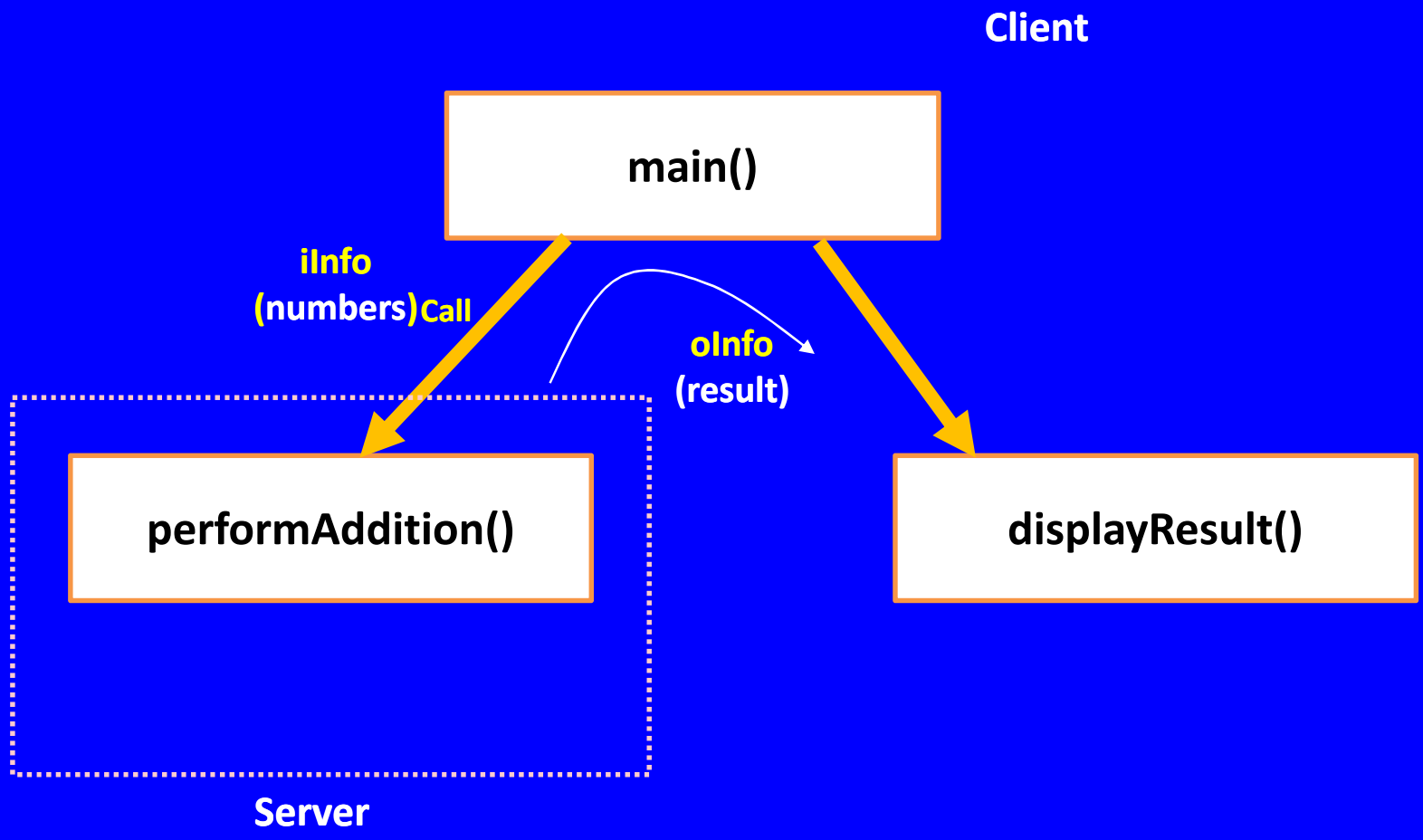
Note about add.c

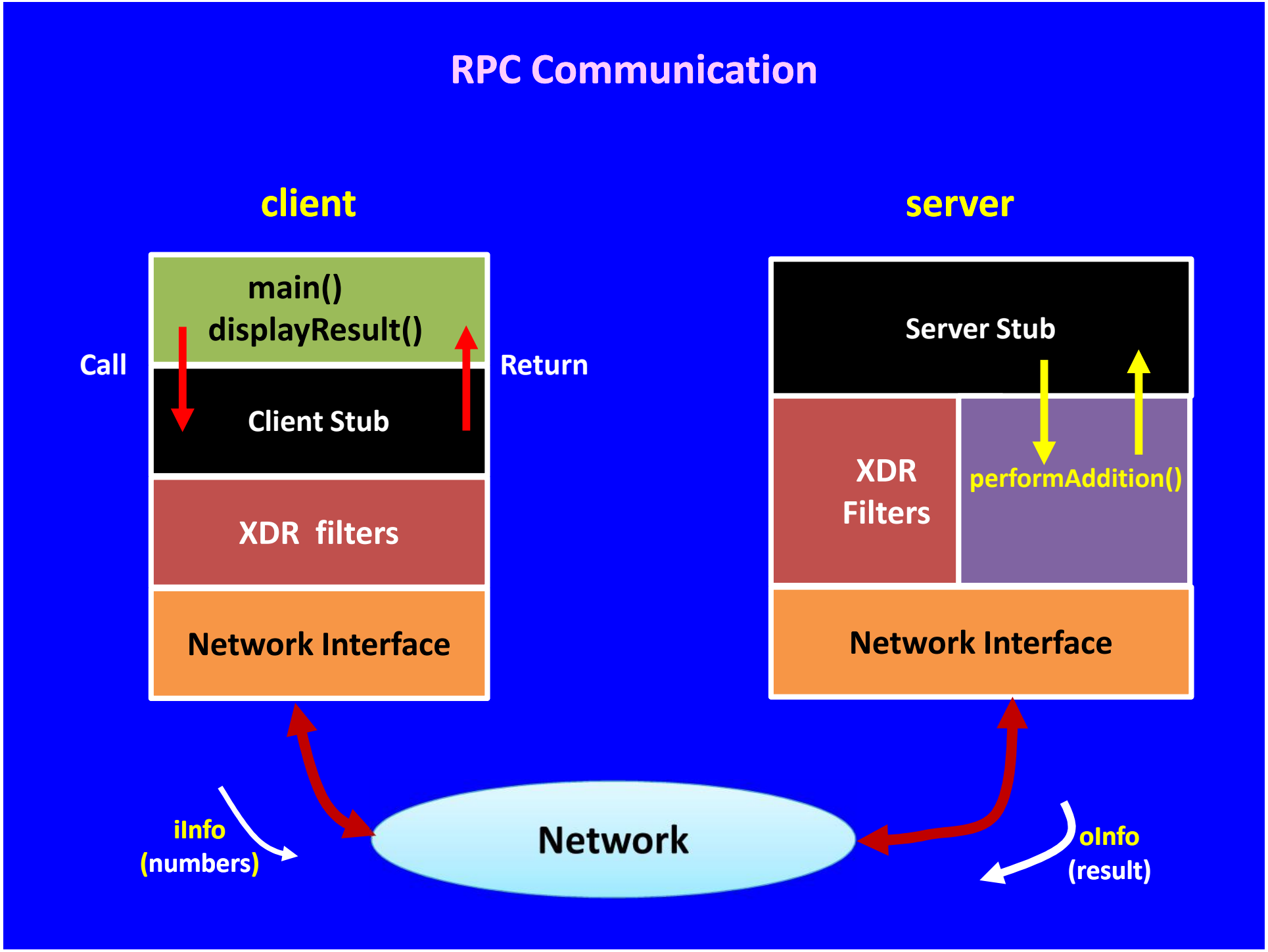
- **Complex data structures:** standalone program to networked RPC conversion becomes easier.
- **Coding strategy:**
  - Write the standalone application first (perform addition);
  - then convert into the network code (RPC)

add.c program (rpc based)

- Convert standalone add.c into a network application using RPC.
- performAddition() will be the remote procedure.

Flow of Functions

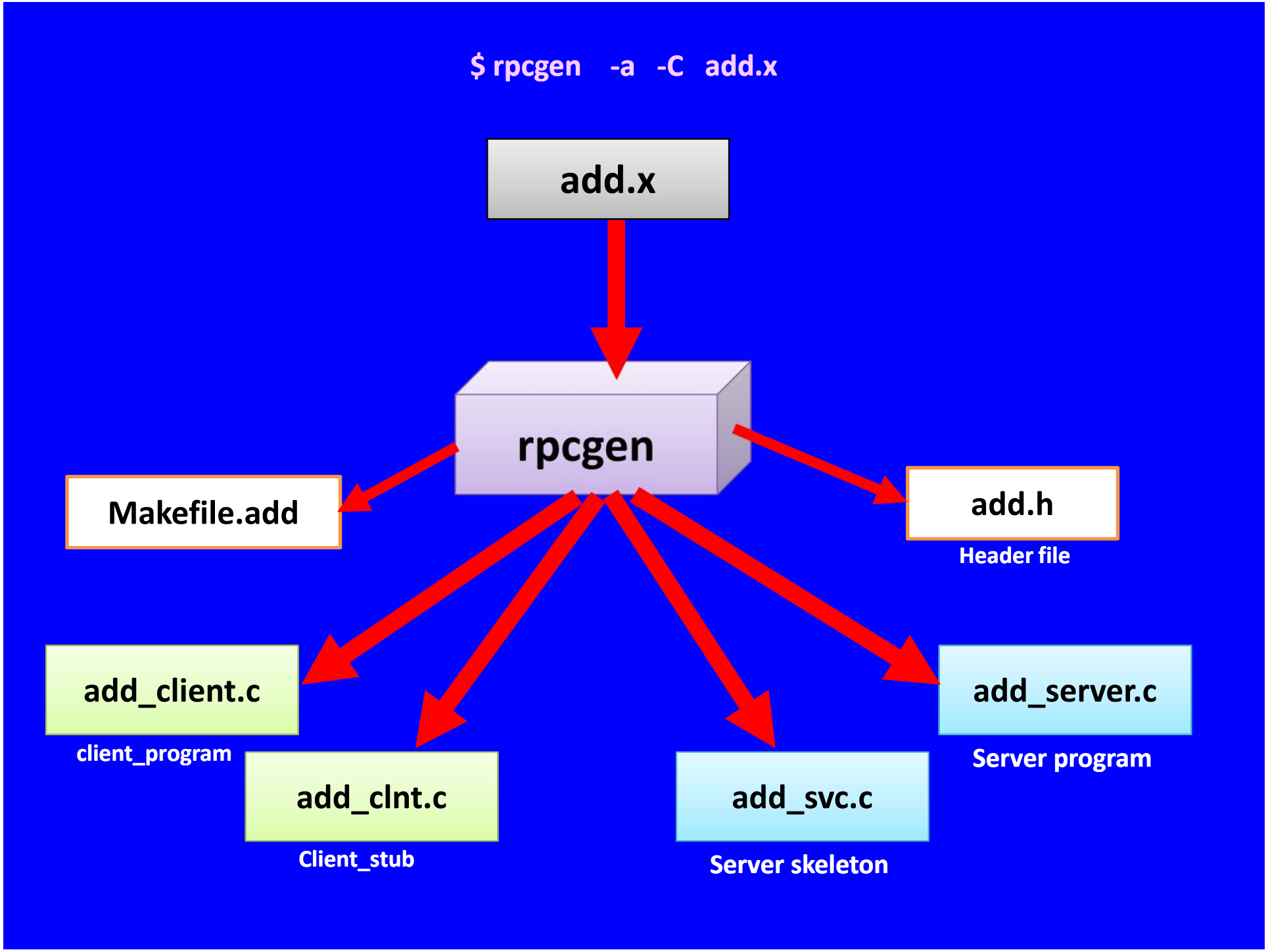



RPC Communication
client
server
main()
displayResult()
Call
Return
Client Stub
XDR filters
Network Interface
Server Stub
XDR
Filters
performAddition()
Network Interface
iInfo
(numbers)
Network
oInfo
(result)

$ rpcgen -C add.x

- add.h → header file for C datatypes involved in network commn
- add_xdr.c → XDR filters
- add_clnt.c → client stub
- add_svc.c → server skeleton

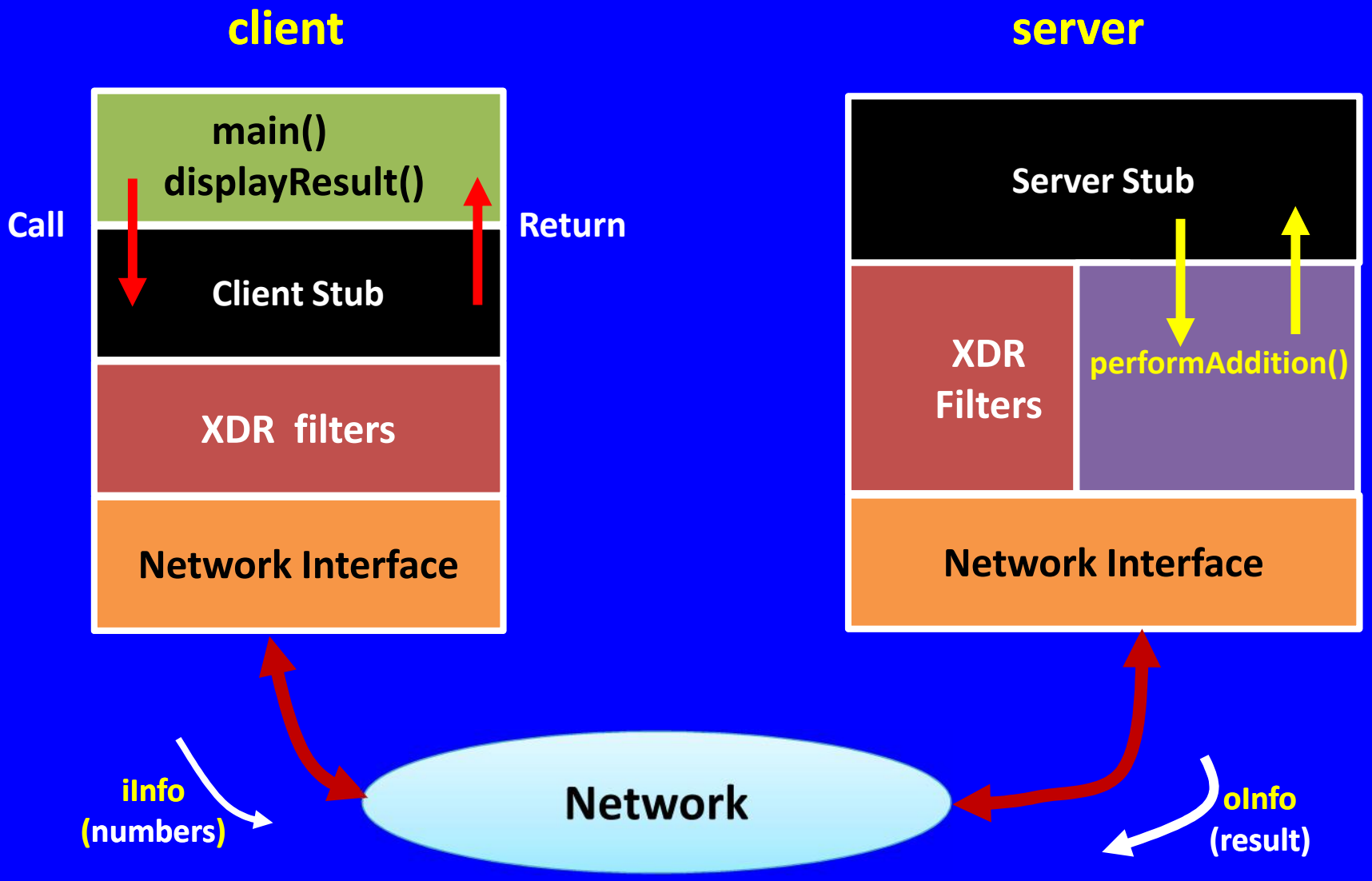
- We do not have to look at these '.c' files


$ rpcgen -a -C add.x
add.x
rpcgen
Makefile.add
add.h
Header file
add_client.c
client_program
add_clnt.c
Client_stub
add_svc.c
Server skeleton
add_server.c
Server program

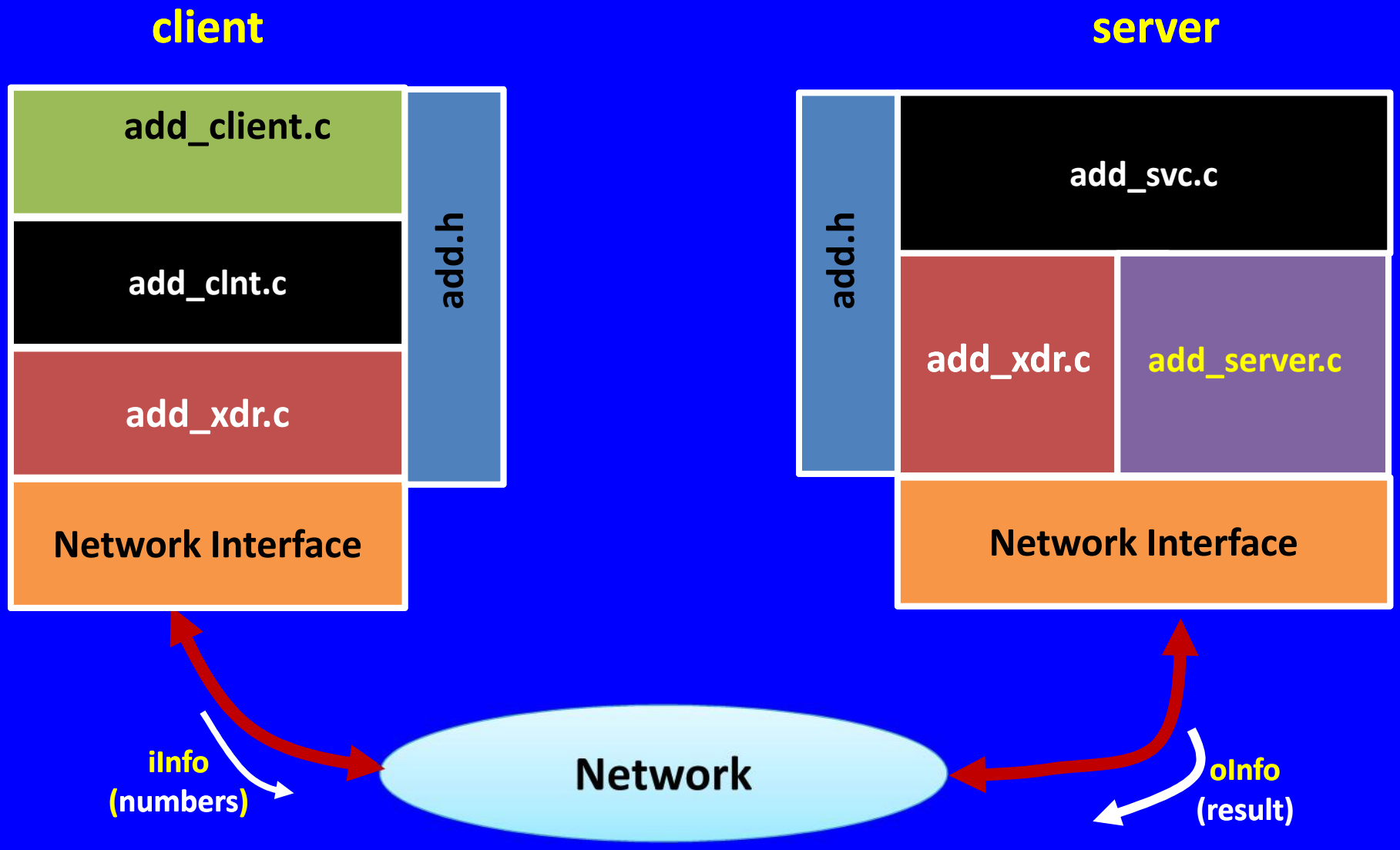
Note

- These files must not already exist
- The programmer must provide the functionalities in terms of codes.

Relationships among Files



Relationships among Files



add.x

Session Edit View Bookmarks Settings Help

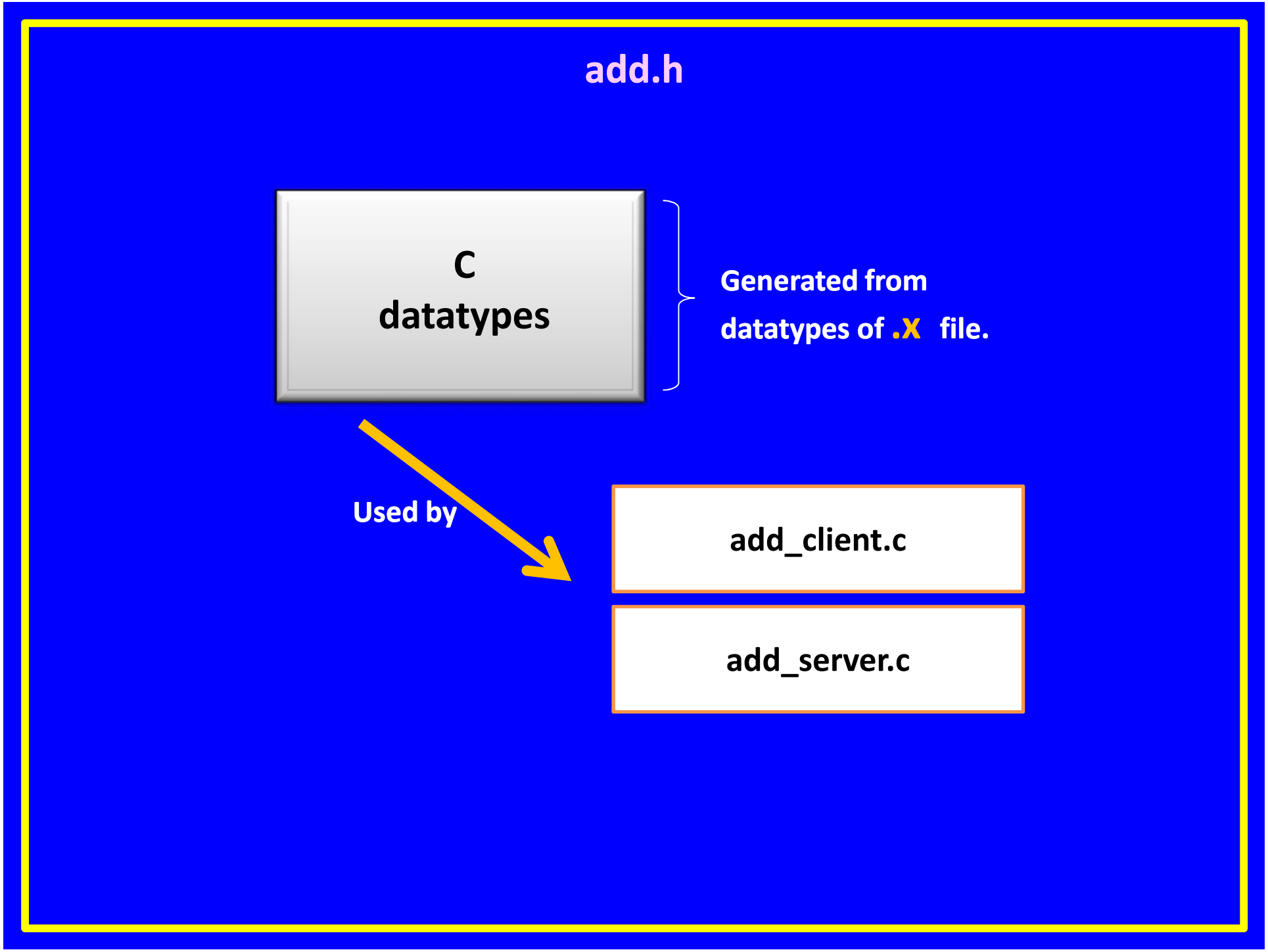
```
struct InputInfo {
        int num1;
        int num2;
};

struct OutputInfo {
        int result;
};

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

Notes

Standard RPC

The remote procedure can only take one input, and return one output.

- **Program, version, and function names must be in uppercase.**


add.h
C
datatypes
Generated from
datatypes of .X file.
Used by
add_client.c
add_server.c

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

struct OutputInfo {
        int result;
};
typedef struct OutputInfo OutputInfo;

#define ADDPROGRAM 22222222
#define ADDVERSION 1

#if defined(__STDC__) || defined(__cplusplus)
#define performAddition 1
extern  struct OutputInfo * performaddition_1(struct InputInfo *, CLIENT *);
extern  struct OutputInfo * performaddition_1_svc(struct InputInfo *, struct svc_req *);
extern int addprogram_1_freeresult (SVCXPRT *, xdrproc_t, caddr_t);

#else /* K&R C */
#define performAddition 1
extern  struct OutputInfo * performaddition_1();
```

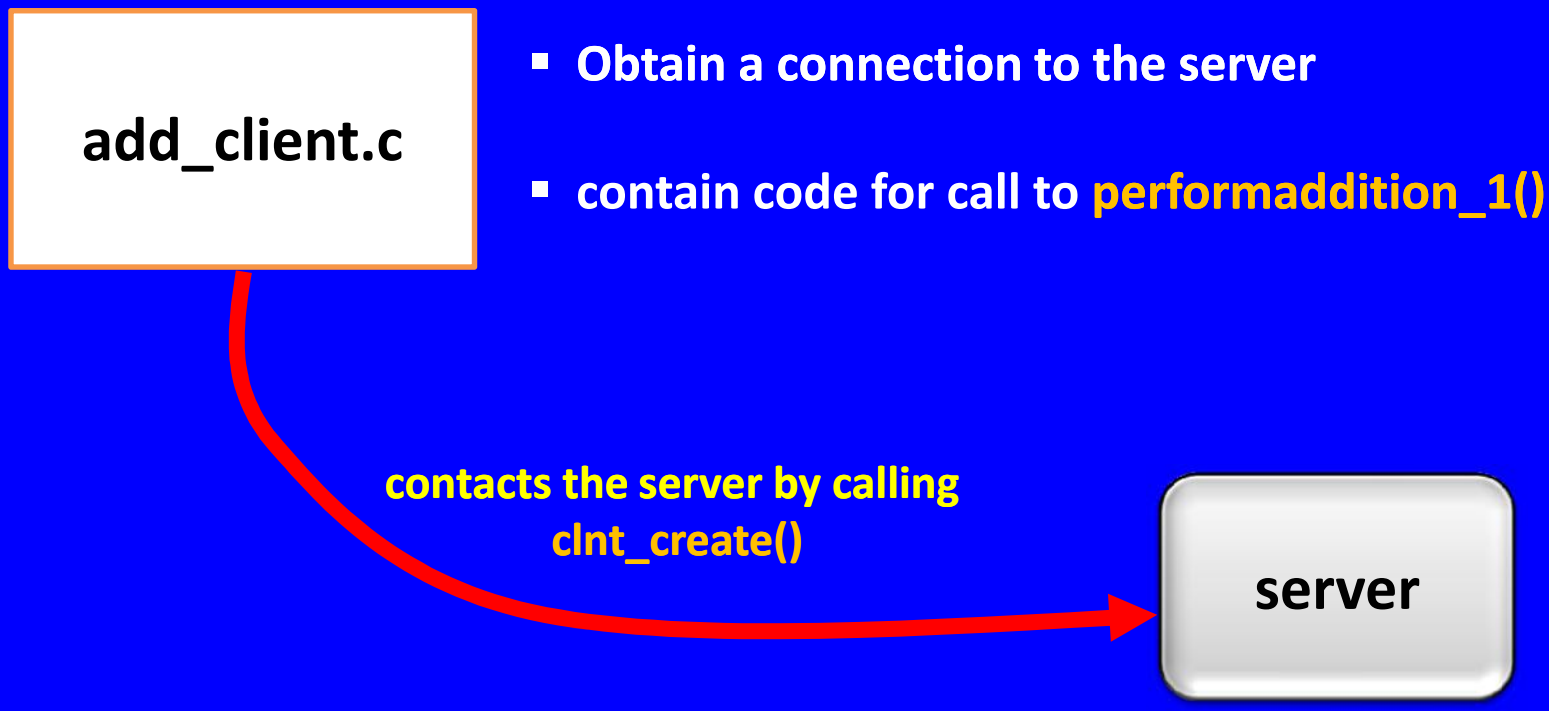
InputNumbers

Output result

Program number

Version number

add_client.c


add_client.c
Obtain a connection to the server
contain code for call to performaddition_1()
contacts the server by calling
clnt_create()
server

clnt_create()

- contact the portmapper on the specified host to get the server details

- failure : returns NULL
- Success: returns a client handle

client handle (used in other RPC library functions)

```
clnt = clnt_create(
        host,
        prog,
        version,
        protocol
    );
```

host name of (server program , portmapper)

Server program

Server program version

protocol (“udp” / “tcp”)

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

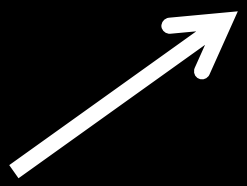
5 **Call display function**

add_clnt.c

- **add_clnt.c** is the client stub for network communication with the server

- It calls the XDR filters for argument passing
- It sends/receives network data to/from the server
- Sets a timeout for trying to contact the server
- all this is done by calling clnt_call()

```c
#include <memory.h> /* for memset */
#include "add.h"

/* Default timeout can be changed using clnt_control() */
static struct timeval TIMEOUT = { 25, 0 };

struct OutputInfo *
performaddition_1(struct InputInfo *argp, CLIENT *clnt)
{
        static struct OutputInfo clnt_res;

        memset((char *)&clnt_res, 0, sizeof(clnt_res));
        if (clnt_call (clnt, performAddition,
                (xdrproc_t) xdr_InputInfo, (caddr_t) argp,
                (xdrproc_t) xdr_OutputInfo, (caddr_t) &clnt_res,
                TIMEOUT) != RPC_SUCCESS) {
                return (NULL);
        }
        return (&clnt_res);
}
```



add_svc.c

- add_svc.c registers server's details with the portmapper

- **add_svc.c** handles incoming messages

- Converts XDR format data to C formats
- calls **performaddition_1_svc()**
  - this is the server-side version of **performAddition()**
- sends results back to the client
  - converts C data to XDR data formats

```c
int
main (int argc, char **argv)
{
        register SVCXPRT *transp;

        pmap_unset (ADDPROGRAM, ADDVERSION);

        transp = svcudp_create(RPC_ANYSOCK);
        if (transp == NULL) {
                fprintf (stderr, "%s", "cannot create udp service.");
                exit(1);
        }
        if (!svc_register(transp, ADDPROGRAM, ADDVERSION, addprogram_1, IPPROTO_UDP)) {
                fprintf (stderr, "%s", "unable to register (ADDPROGRAM, ADDVERSION, udp).");
                exit(1);
        }

        transp = svctcp_create(RPC_ANYSOCK, 0, 0);
        if (transp == NULL) {
                fprintf (stderr, "%s", "cannot create tcp service.");
                exit(1);
        }
        if (!svc_register(transp, ADDPROGRAM, ADDVERSION, addprogram_1, IPPROTO_TCP)) {
                fprintf (stderr, "%s", "unable to register (ADDPROGRAM, ADDVERSION, tcp).");
                exit(1);
        }

        svc_run ();
        fprintf (stderr, "%s", "svc_run returned");
        exit (1);
        /* NOTREACHED */
}
```

register

Call performaddition_1_svc()

```
static void
addprogram_1(struct svc_req *rqstp, register SVCXPRT *transp)
{
        union {
                struct InputInfo performaddition_1_arg;
        } argument;
        char *result;
        xdrproc_t _xdr_argument, _xdr_result;
        char *(*local)(char *, struct svc_req *);

        switch (rqstp->rq_proc) {
        case NULLPROC:
                (void) svc_sendreply (transp, (xdrproc_t) xdr_void, (char *)NULL);
                return;

        case performAddition:
                _xdr_argument = (xdrproc_t) xdr_InputInfo;
                _xdr_result = (xdrproc_t) xdr_OutputInfo;
                local = (char *(*)(char *, struct svc_req *)) performaddition_1_svc;
                break;
```

add_server.c

- This file contains performaddition_1_svc().
- The application code needs to be added.

stores details used for client authentication

```c
#include "add.h"

struct OutputInfo *
performaddition_1_svc(struct InputInfo *argp, struct svc_req *rqstp)
{
        static struct OutputInfo  result;

        /////////////////////////////////////////////
        int sum;
        sum = argp->num1 + argp->num2;
        result.result = sum;

        /////////////////////////////////////////////
        return &result;
}
```

Functionalities provided by user

Notes

- result must be static so that its memory is not deleted when the function returns.
- result is retained so that the top-level server can convert it to network form

add_xdr.c

```
#include "add.h"

bool_t
xdr_InputInfo (XDR *xdrs, InputInfo *objp)
{
        register int32_t *buf;

         if (!xdr_int (xdrs, &objp->num1))
                 return FALSE;
         if (!xdr_int (xdrs, &objp->num2))
                 return FALSE;
        return TRUE;
}

bool_t
xdr_OutputInfo (XDR *xdrs, OutputInfo *objp)
{
        register int32_t *buf;

         if (!xdr_int (xdrs, &objp->result))
                 return FALSE;
        return TRUE;
}
```

Step - IV

Remote Procedure Call: Fifth Step

More Example Programs

Example 1: array in RPC

Variable array in RPC

```
typedef   int a<>;

program          IARRAY_PROGRAM {

                 version IARRAY_VERSION {

                          int IARRAYADD(a)=1;

                  }=1;
}=22222222;
```

vadd.h

```
typedef struct {
        u_int iarray_len;
        int *iarray_val;
} iarray;

#define VADD_PROG 555575555
#define VADD_VERSION 1

#if defined(__STDC__) || defined(__cplusplus)
#define VADD 1
extern  int * vadd_1(iarray *, CLIENT *);
extern  int * vadd_1_svc(iarray *, struct svc_req *);
extern int vadd_prog_1_freeresult (SVCXPRT *, xdrproc_t, caddr_t);

#else /* K&R C */
#define VADD 1
extern  int * vadd_1();
extern  int * vadd_1_svc();
extern int vadd_prog_1_freeresult ();
#endif /* K&R C */

/* the xdr functions */

#if defined(__STDC__) || defined(__cplusplus)
extern  bool_t xdr_iarray (XDR *, iarray*);

#else /* K&R C */
extern bool_t xdr_iarray ();

#endif /* K&R C */

#ifdef __cplusplus
}
#endif
```

vadd_client.c

```
#include "vadd.h"

void
vadd_prog_1(char *host, int* vArray, int sz)
{
        CLIENT *clnt;
        int  *result_1;
        iarray  vadd_1_arg;

////////////////////////////////////////////////////////////////
//created in *.h header file
/* typedef struct {
                u_int iarray_len;
                        int *iarray_val;
} iarray;
*/
////////////////////////////////////////////////////////////////
        ////////////////////////////////
    4   vadd_1_arg.iarray_len = sz; //assign no of elements in the array
        vadd_1_arg.iarray_val = vArray; // set up array for the server
////////////////////////////////////////////////////////////////////////////
#ifndef DEBUG
        clnt = clnt_create (host, VADD_PROG, VADD_VERSION, "udp");
        if (clnt == NULL) {
                clnt_pcreateerror (host);
                exit (1);
        }
#endif  /* DEBUG */

        result_1 = vadd_1(&vadd_1_arg, clnt);
        if (result_1 == (int *) NULL) {
                clnt_perror (clnt, "call failed");
```

3 Provide appropriate number of arguments and declare variables

```
        result_1 = vadd_1(&vadd_1_arg, clnt);
        if (result_1 == (int *) NULL) {
                clnt_perror (clnt, "call failed");
        }
        else {
                printf("sum oof array elements=%d\n",*result_1);
        }
#ifndef DEBUG
        clnt_destroy (clnt);
#endif   /* DEBUG */
}


int
main (int argc, char *argv[])
{
        char *host;
        //////////////////
        int *tArray, n, i;

        if (argc < 3) {
                printf ("usage: %s server_host\n", argv[0]);
                exit (1);
        }
        host = argv[1];
        ////////////////////////////
        n = argc-2; //size will be two less than ./ppgm localhost
        tArray = (int*)malloc(n*sizeof(int)); //create memory for elements
        for(i=2; i<argc; i++) { //assign values from command line iterate from second position ./pgm localhost 1 2 3 4 5
                tArray[i-2] = atoi(argv[i]); //index start from 0 -> i-2
        }

        vadd_prog_1 (host, tArray, n);
exit (0);
}
```

5 **Display result**

```
./add   127.0.0.1   12 45 5 7 8 9
```

1

2

vadd_server.c

```c
#include "vadd.h"

int *
vadd_1_svc(iarray *argp, struct svc_req *rqstp)
{
        static int  result;

        int i, sum;
        ////////////////////////
        //for reference
        ////////////////////////
/*      typedef struct {
                u_int iarray_len;
                int *iarray_val;
        } iarray;
*/
        ////////////////////////////
        printf("Got request from the client adding %d elements\n", argp->iarray_len);

        sum=0;
        for (i=0;i<argp->iarray_len;i++) {
                sum = sum + argp->iarray_val[i];
        }
        result = sum;
return &result;
}
~
```

string_lower_Upper

strOperations.x

```
struct InputString {

        string iWord<>;
        int len;
        int option;
};

struct OutputString{

        string oWord<>;
};

program STRINGPROGRAM {

        version STRINGPROGRAMVERSION {

                struct OutputString LOWERUPPER(struct InputString)=1;

        }=1;

}=0x20000234;
```

strOperations_client.c

```c
void
stringprogram_1(char *host, char* strInput, int len, int ch)
{
        CLIENT *clnt;
        struct OutputString  *result_1;
        struct InputString  lowerupper_1_arg;

        ////////////////////////////////////
        lowerupper_1_arg.iWord = (char*)malloc( len * sizeof(char));
        strcpy(lowerupper_1_arg.iWord, strInput);
        lowerupper_1_arg.len = len;
        lowerupper_1_arg.option = ch;
        ////////////////////////////////////

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

        result_1 = lowerupper_1(&lowerupper_1_arg, clnt);
        if (result_1 == (struct OutputString *) NULL) {
                clnt_perror (clnt, "call failed");
        }
        else {
                printf("result=%s\n", result_1->oWord);
        }

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

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

        if (argc < 4) {
                printf ("usage: %s server_host\n", argv[0]);
                exit (1);
        }
        host = argv[1];
        ////////////////////////////////////////
        len = strlen(argv[2]);
        ch = atoi(argv[3]);
        iString = (char*)malloc( len * sizeof(char));
        strcpy(iString, argv[2]);
        ////////////////////////////////////////
        stringprogram_1 (host, iString, len, ch);///
exit (0);
}
```

strOperations_server.c

```c
#include "strOperations.h"
#include <string.h>

void convert(char* iString);

struct OutputString *
lowerupper_1_svc(struct InputString *argp, struct svc_req *rqstp)
{
        static struct OutputString  result;
        int len;
        char *temp;

        printf("received request\n");
        len = strlen(argp->iWord);

        temp = (char*)malloc( len * sizeof(char));
        strcpy(temp, argp->iWord);
        result.oWord = (char*)malloc( len * sizeof(char));
        convert(temp);

        strcpy(result.oWord, temp);
        printf("result=%s\n",temp);

        return &result;
}
```

```
void convert(char* iString)
{
        while( *iString != '\0') {

                *iString = toupper(*iString);
                ++iString;
        }
}
```

arith_operations

arith.x

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

        int choice; /*choice*/
};

struct OutputInfo {
        int outputValue;
        float outputFloatValue;
};

program ARITHPROGRAM {
        version ARITHVERSION {
                struct OutputInfo performArithmatic(struct InputInfo iInfo)=1;
        }=1;
}=55555555;
```

arith_client.c

```c
//////////////////////
void  displayResult( struct InputInfo iInfo, struct  OutputInfo* oInfo );
//////////////////////
void arithprogram_1(char *host, int n1, int n2, int ch)
{
        CLIENT *clnt;
        struct OutputInfo  *result_1;
        struct InputInfo  performarithmatic_1_arg;

        ////////////////////////////////////////
        performarithmatic_1_arg.num1 = n1;
        performarithmatic_1_arg.num2 = n2;
        performarithmatic_1_arg.choice = ch;
        ////////////////////////////////////////
#ifndef DEBUG
        clnt = clnt_create (host, ARITHPROGRAM, ARITHVERSION, "udp");
        if (clnt == NULL) {
                clnt_pcreateerror (host);
                exit (1);
        }
#endif  /* DEBUG */
        result_1 = performarithmatic_1(&performarithmatic_1_arg, clnt);
        if (result_1 == (struct OutputInfo *) NULL) {
                clnt_perror (clnt, "call failed");
        }
        else { ////////////////////
                displayResult( performarithmatic_1_arg, result_1);
        }
#ifndef DEBUG
        clnt_destroy (clnt);
#endif   /* DEBUG */
}
```

```c
int main (int argc, char *argv[])
{
        char *host;
        int n1, n2, ch;//


        if (argc != 5) {
                printf ("usage: %s server_host num1 num2 choice\n", argv[0]);
                exit (1);
        }
        host = argv[1];
        ////////////////////////////////

        n1 = atoi(argv[2]);
        n2 = atoi(argv[3]);
        ch = atoi(argv[4]);

        ////////////////////////////////
        arithprogram_1 (host, n1, n2, ch);
exit (0);
}
```

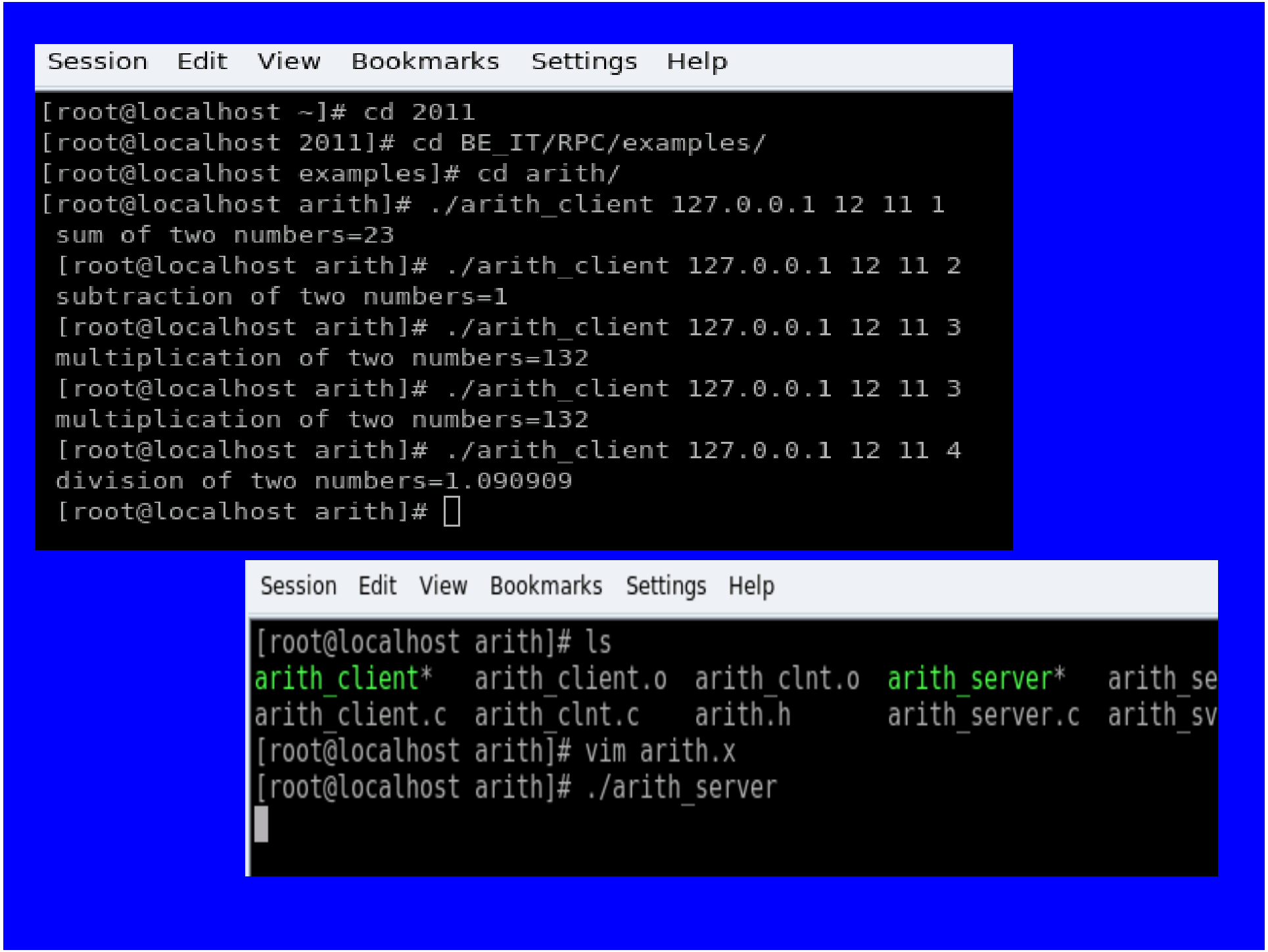
```
void  displayResult( struct InputInfo iInfo, struct  OutputInfo*  oInfo )
{
        if( iInfo.choice == 4) {
                printf(" division of two numbers=%f\n ", oInfo->outputFloatValue );
        }
        else if(iInfo.choice == 1) {
                printf(" sum of two numbers=%d\n ", oInfo->outputValue );
        }
        else if(iInfo.choice == 2) {
                printf(" subtraction of two numbers=%d\n ", oInfo->outputValue );
        }
        else if(iInfo.choice == 3) {
                printf(" multiplication of two numbers=%d\n ", oInfo->outputValue );
        }
}
```

arith_server.c

```c
#include "arith.h"

//////////////////
struct OutputInfo doAddition(struct InputInfo* iInfo);
struct OutputInfo doSubtraction(struct InputInfo* iInfo);
struct OutputInfo doMultiplication(struct InputInfo* iInfo);
struct OutputInfo doDivision(struct InputInfo* iInfo);
//////////////////
struct OutputInfo *
performarithmatic_1_svc(struct InputInfo *argp, struct svc_req *rqstp)
{
        static struct OutputInfo  result;

        //////////////
        switch(argp->choice)
        {
                case 1:
                        result = doAddition(argp);
                        break;
                case 2:
                        result = doSubtraction(argp);
                        break;
                case 3:
                        result = doMultiplication(argp);
                        break;
                case 4:
                        result = doDivision(argp);
                        break;
        }//switch

        //////////////
        return &result;
}
```

```c
struct OutputInfo doAddition(struct InputInfo* iInfo)
{
        struct OutputInfo oInfo;
        int a;
        a = iInfo->num1 + iInfo->num2;
        oInfo.outputValue = a;
return oInfo;
}

////////////////////
struct OutputInfo doSubtraction(struct InputInfo* iInfo)
{
        struct OutputInfo oInfo;
        int s;
        s = iInfo->num1 - iInfo->num2;
        oInfo.outputValue = s;
return oInfo;
}
```

```
//////////////////////
struct OutputInfo doMultiplication(struct InputInfo* iInfo)
{
        struct OutputInfo oInfo;
        int s;
        s = iInfo->num1 * iInfo->num2;
        oInfo.outputValue = s;
return oInfo;
}
//////////////////////
struct OutputInfo doDivision(struct InputInfo* iInfo)
{
        struct OutputInfo oInfo;
        float d;
        d = (float)iInfo->num1 / iInfo->num2;
        oInfo.outputFloatValue = d;
return oInfo;
}
```


Session Edit View Bookmarks Settings Help
[root@localhost ~]# cd 2011
[root@localhost 2011]# cd BE_IT/RPC/examples/
[root@localhost examples]# cd arith/
[root@localhost arith]# ./arith_client 127.0.0.1 12 11 1
sum of two numbers=23
[root@localhost arith]# ./arith_client 127.0.0.1 12 11 2
subtraction of two numbers=1
[root@localhost arith]# ./arith_client 127.0.0.1 12 11 3
multiplication of two numbers=132
[root@localhost arith]# ./arith_client 127.0.0.1 12 11 3
multiplication of two numbers=132
[root@localhost arith]# ./arith_client 127.0.0.1 12 11 4
division of two numbers=1.090909
[root@localhost arith]#
Session Edit View Bookmarks Settings Help
[root@localhost arith]# ls
arith_client* arith_client.o arith_clnt.o arith_server* arith_se
arith_client.c arith_clnt.c arith.h arith_server.c arith_sv
[root@localhost arith]# vim arith.x
[root@localhost arith]# ./arith_server

reverese_number

revNumber.x

```
struct InputNumber {
        int iNum;
};

struct OutputNumber {
        int oRev;
};

program REVNUMBERPROGRAM {
        version REVNUMBERVERSION {
                struct OutputNumber GETREVERSE(struct InputNumber)=1;
        }=1;
}=0x20000567;
```

revNumber_client.c

```c
#include "revNumber.h"


void
revnumberprogram_1(char *host, int num)
{
        CLIENT *clnt;
        struct OutputNumber  *result_1;
        struct InputNumber  getreverse_1_arg;

        ////////////////////////
        getreverse_1_arg.iNum = num;
        ////////////////////////
#ifndef DEBUG
        clnt = clnt_create (host, REVNUMBERPROGRAM, REVNUMBERVERSION, "udp");
        if (clnt == NULL) {
                clnt_pcreateerror (host);
                exit (1);
        }
#endif  /* DEBUG */

        result_1 = getreverse_1(&getreverse_1_arg, clnt);
        if (result_1 == (struct OutputNumber *) NULL) {
                clnt_perror (clnt, "call failed");
        }
        else {//////////
                printf("reverse of the number is=%d\n", result_1->oRev);
        }
#ifndef DEBUG
        clnt_destroy (clnt);
#endif   /* DEBUG */
}
```

```c
int
main (int argc, char *argv[])
{
        char *host;
        ////////////
        int num;
        if (argc < 3) {
                printf ("usage: %s server_host\n", argv[0]);
                exit (1);
        }
        host = argv[1];
        ////////////////////////////
        num = atoi(argv[2]);
        revnumberprogram_1 (host,num);
exit (0);
}
```

revNumber_server.c

```c
#include "revNumber.h"

struct OutputNumber getReverse(struct InputNumber* iN);

struct OutputNumber *
getreverse_1_svc(struct InputNumber *argp, struct svc_req *rqstp)
{
        static struct OutputNumber  result;

        ////////////////////////////
        printf("got request\n");
        result = getReverse(argp);

        return &result;
}

struct OutputNumber getReverse(struct InputNumber* iN)
{
        struct OutputNumber oN;
        int r, reverse=0;

        while(iN->iNum){
                r = iN->iNum % 10;
                reverse = reverse*10+r;
                iN->iNum = iN->iNum/10;
        }
        oN.oRev = reverse;
return oN;
}
```

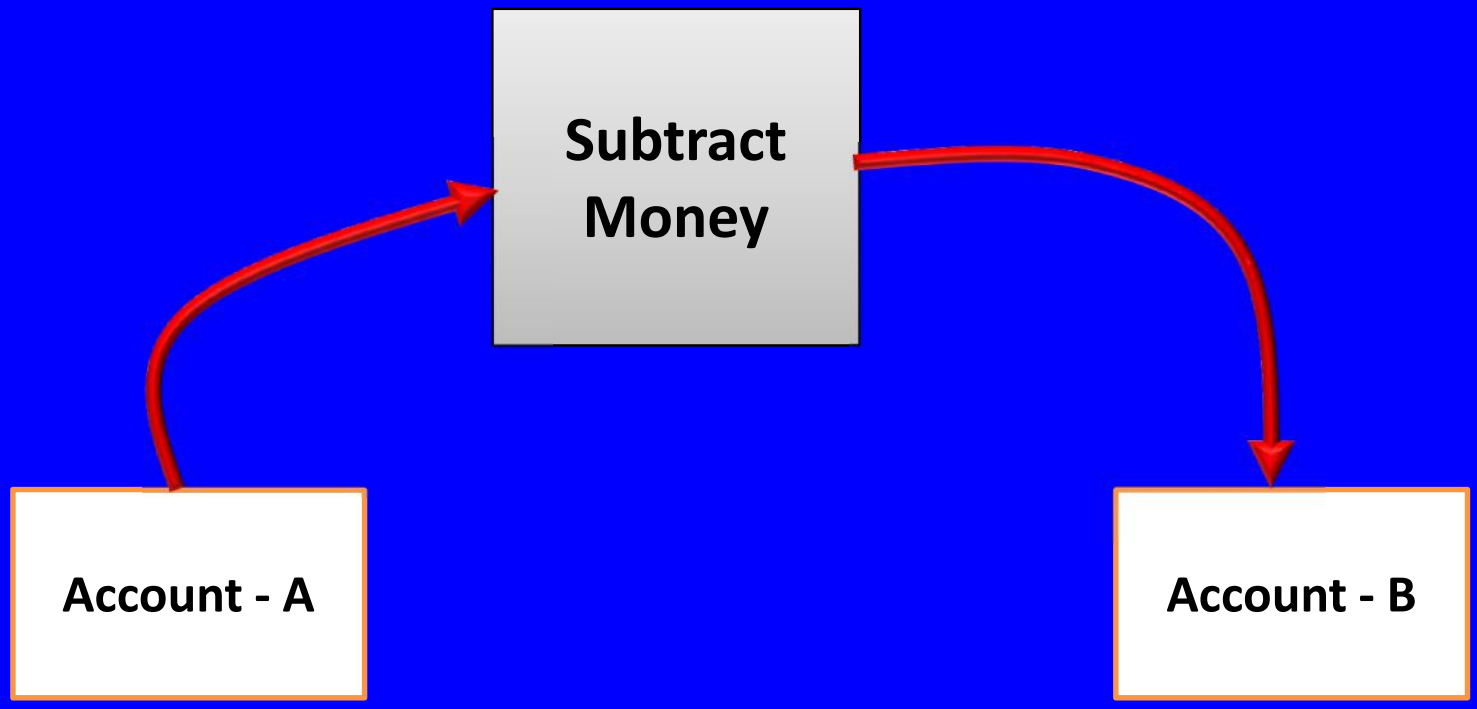
Idempotent Vs Non-Idempotent Procedures

- **Idempotent procedure: means the procedure can be called any number of times without harm.**

add_1(Num1, Num2)	❖ Returns correct result whether call it once or twice
Return_time()	❖ This procedure may return different information each time , still OK (correct output)

- **Nonidempotent procedure example:**

Called only once
otherwise end result will be wrong



Step - V

Thank You!