CSci 5105

Introduction to Distributed Systems

Communication: RPC In Practice

Linux RPC

- Language-neutral RPC
- Can use Fortran, C, C++
- IDL compiler *rpgen* –N to generate all stubs, skeletons (server stub)
- Example:
  - RPC for simple arithmetic

Linux RPC - IDL

```
/*  Arith service interface
definition in file Arith.x */

program Arith {
 version VERSION {
        int add(int, int)=1;
    }=2;
 }=9999;
```

uniquely identifies this function

version # of the interface

uniquely identifies this program

Decision?

- You are coding both sides of an RPC
- Why would you choose to implement the procedure remotely?

Linux RPC

`rpcgen -N -a Arith.x`

Creates automatically: (do it)

Makefile.Arith, Arith.h,
Arith_clnt.c (stub)
Arith_svc.c (skeleton/stub)
Arith_xdr.c (conversion)
Arith_client.c
Arith_server.c

Linux RPC

```
make –f makefile.Arith =>
Arith_client
Arith_server
```

Linux RPC: Server-side

Before I build … need to add actual implementations, e.g.

`Arith_server.c:`

```
int * add_2(arg1, arg2, rqstp)
        int arg1;
        int arg2;
        struct svc_req *rqstp; {
        static int  result;
        /*
         * insert server code here
         */
        result = arg1 + arg2;
        return (&result);
}
```

Linux RPC: Client-side

Look at Arith_client.c

Linux RPC: Binding/Transparency

```
clnt_create(host, Arith,
    VERSION,“netpath");
```

Client must specify remote host and all version numbers

(although generated automatically)

Linux RPC: Plumbing

Look at Arith_clnt.c

Look at Arith_svc.c

Look at Arith_xdr.c

Whew! Glad I didn't have to write that!

Linux RPC: To Run

<remote host>**Arith_server&**

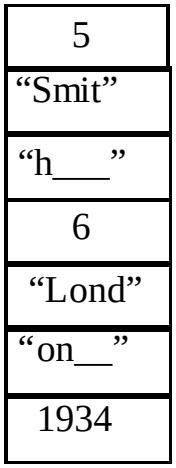
…

<client host>**Arith_client <remote host>**

XDR

- XDR “external data representation” used by Linux RPC
  - fixed size records: easy to process

“Smith” “London” 1934



length

sequence of records of equal size (4 bytes)

also provisions to indicate big/little endian

Thought Q

RPC to perform image manipulation:

image is a 2D array of 100x100 integers

```
image convolve (filter, image)
image scale (scale_factor, image)
```

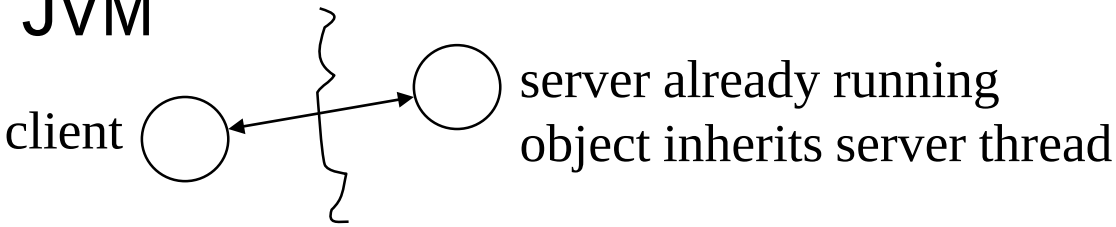
Client:
```
new_image =
        convolve (F, scale (S, input_image))
```

As a designer can you make this more efficient?

Java RMI

RMI: What is it?

- RMI allows clients on one computer to invoke methods on objects running on another computer in a different JVM



- Remote object is stored with a server
  - can be invoked locally on server machine or remotely via the server using RMI

RMI

Remote class – instances can be used remotely

- client refers to object using an *object handle or proxy*

Client need only know about remote interface to invoke the object

Registry

- A *Registry* is to used by clients to locate remote objects
- Remote objects must register themselves with Registry server, `rmiregistry`
- How to find the registry?
  - client must know which machine registry is on
  - server assumes it is the local machine

RMI example

- Hello World

```
import java.rmi.Remote;
import java.rmi.RemoteException;

public interface HelloInterface extends Remote {
    public String say() throws RemoteException;
}
```

required

RMI example (cont'd)

`Interface java.rmi.Remote`

methods can be called from any JVM;

any object that implements this interface becomes a remote object

`Remote Interface methods`

the `say` method is a remote method

RMI Example : Remote Class

```
import java.rmi.*;
import java.rmi.server.*;

public class Hello extends UnicastRemoteObject
                    implements HelloInterface {
   private String message;
   // this method is not remote!
   public Hello (String msg) throws RemoteException {
      message = msg;
   }
   public String say() throws RemoteException {
      return message;
   }
}
```

RMI Example : Server

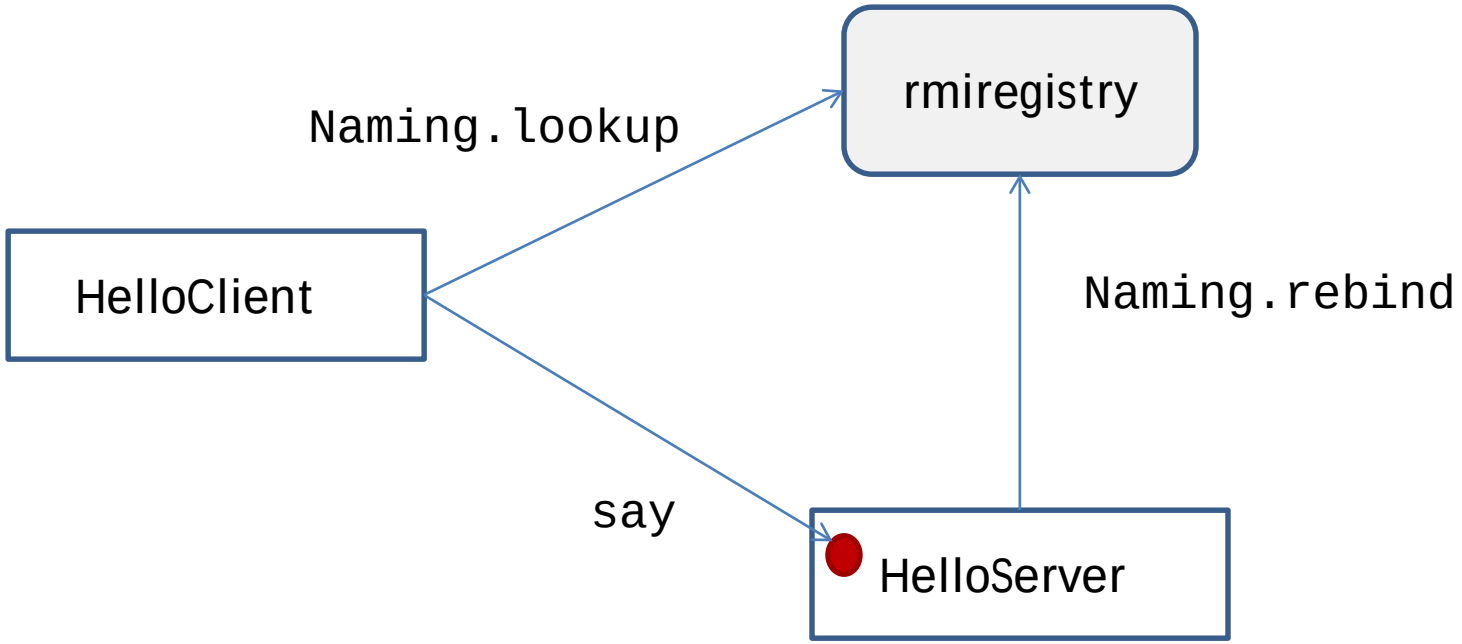
```
import java.rmi.Naming;

public class HelloServer
{
    public static void main (String[] argv)
    {
     try {
      Naming.rebind // assumes registry on this machine
       ("Hello", new Hello ("Hello from Minnesota!"));
      System.out.println
      ("Server is connected and ready for operation.");
     }
     catch (Exception e) {
      System.out.println ("Server not connected: " + e);
     }
    }
}
```

RMI Example: Client

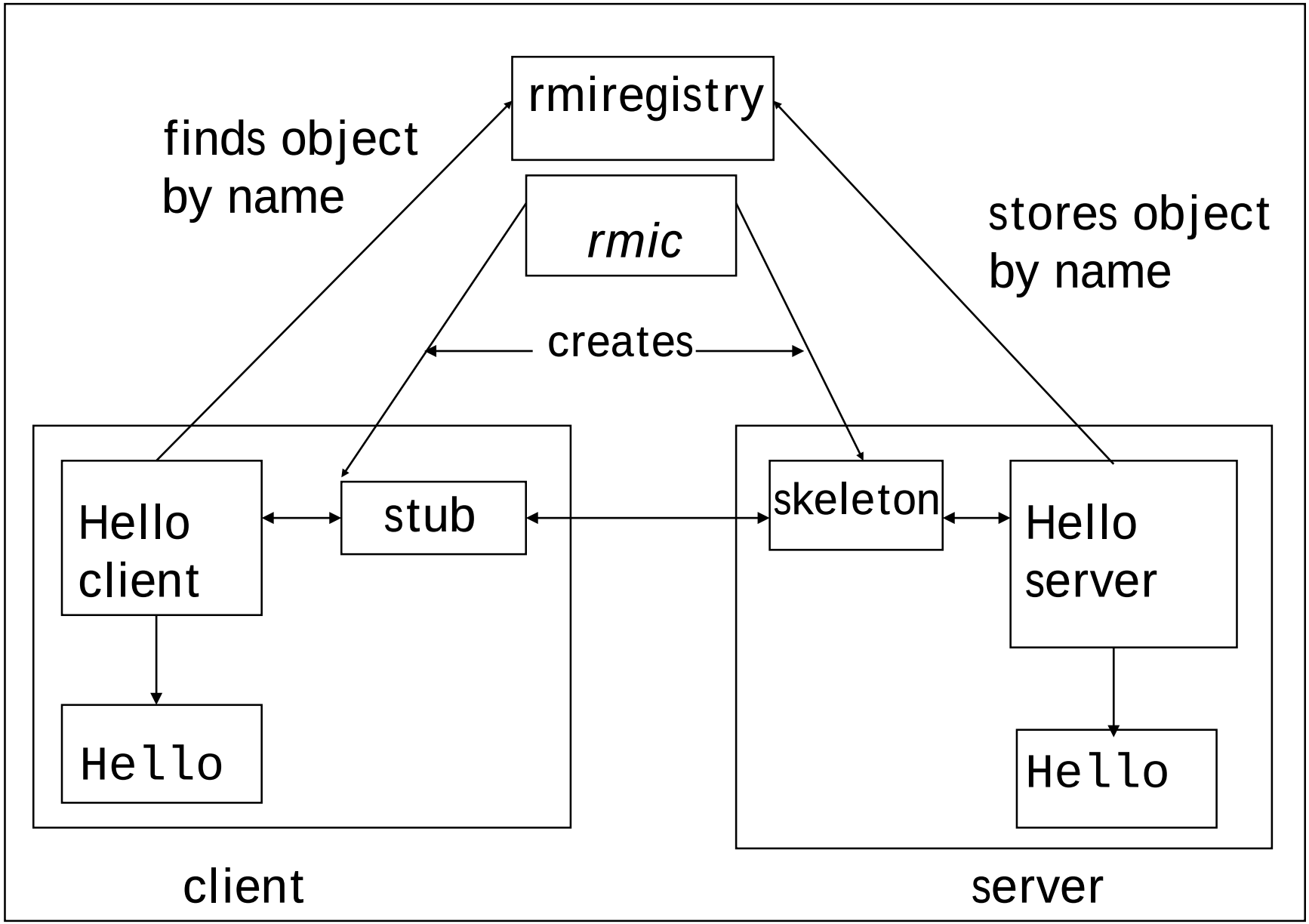
```
import java.rmi.Naming;

public class HelloClient
{
   public static void main (String[] args) {
       try {
         // host/args[0] is the registry location
         HelloInterface hello =(HelloInterface)
           Naming.lookup ("//” + args[0] + “/Hello");
         System.out.println (hello.say());
       }
       catch (Exception e){
        System.out.println ("HelloClient bug: " + e);}
    }
}
```

RMI Example: Client (cont'd)

Message flow in the client



Inside RMI: Hello example



RMI Example: Building it

javac Hello.java => Hello,HelloInterface.class

rmic Hello

produces Hello_Stub.class, ~~Hello_Skel.class~~

Server-side:

javac HelloServer.java

`HelloInterface` must be on classpath

Client-side:

javac HelloClient.java

RMI Example: Running

First run the registry

<caesar> rmiregistry &

Start the server:

<caesar> java HelloServer&

Run the client:

<tera> java HelloClient caesar