CC5212-1
Procesamiento Masivo de Datos
Otoño 2016

Lab 2: Mensaje

Aidan Hogan
aidhog@gmail.com

Step 1: Get Started

- Login:
  - Username: nombre/cc5212
  - Password on board

- Get code and import into Eclipse:
  - http://aidanhogan.com/teaching/cc5212-1-2016/02/mdp-lab-02.zip
  - File > Import > …

Step 2: Connect to Directory

- I start the directory!
  - Will put details on the board

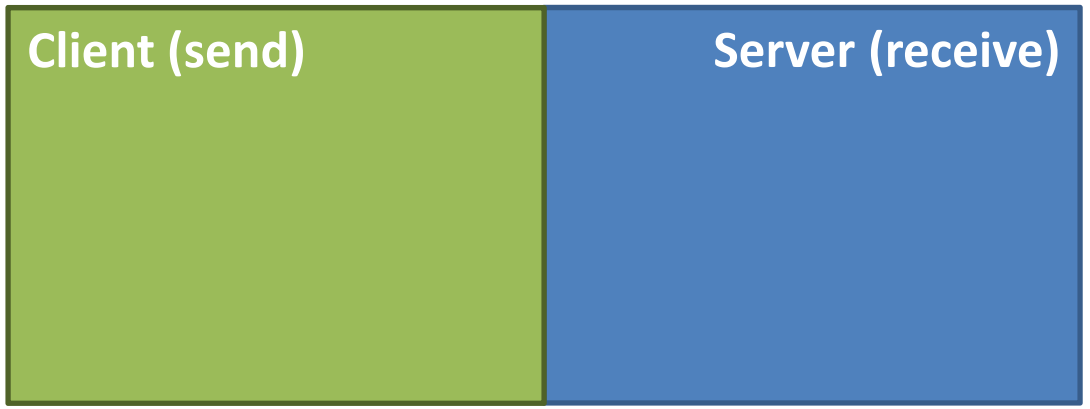
- Connect …
  - `org.mdp.cli.UserDirectoryClientApp`

Step 3: Implement Message Server

- `org.mdp.im.InstantMessagingServer`
  - Implement `InstantMessagingStub`
  - Write `message(.,.)` method
    - Print the message to the console and whom it's from

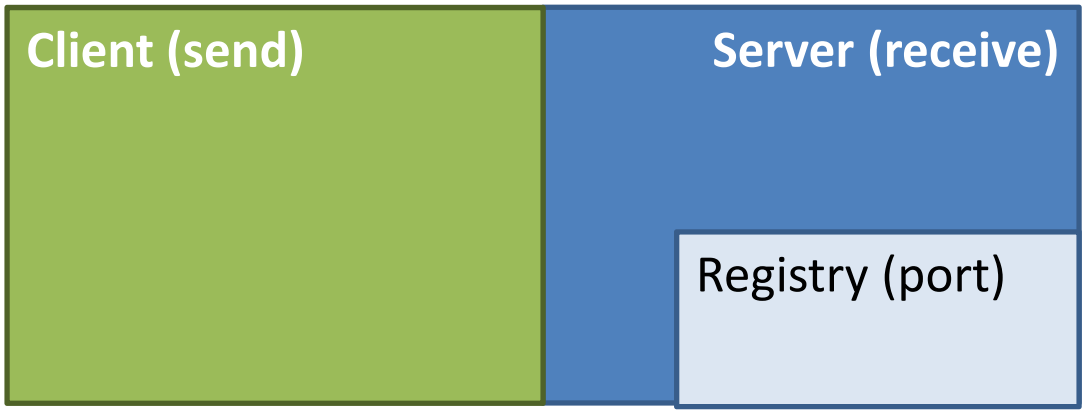
Step 4: Implement Message App.

- `org.mdp.cli.InstantMessagingApp`



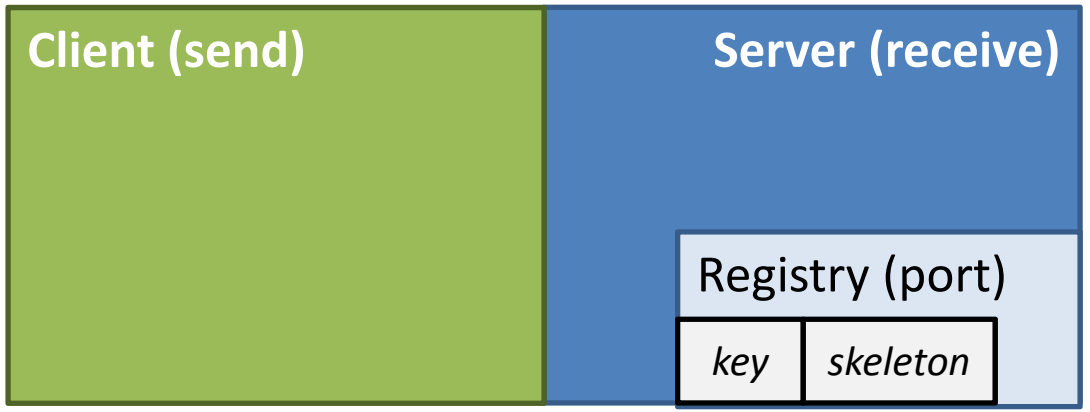
Step 4a: Start Registry

- `org.mdp.cli.InstantMessagingApp`
  - Implement `startRegistry(.)`



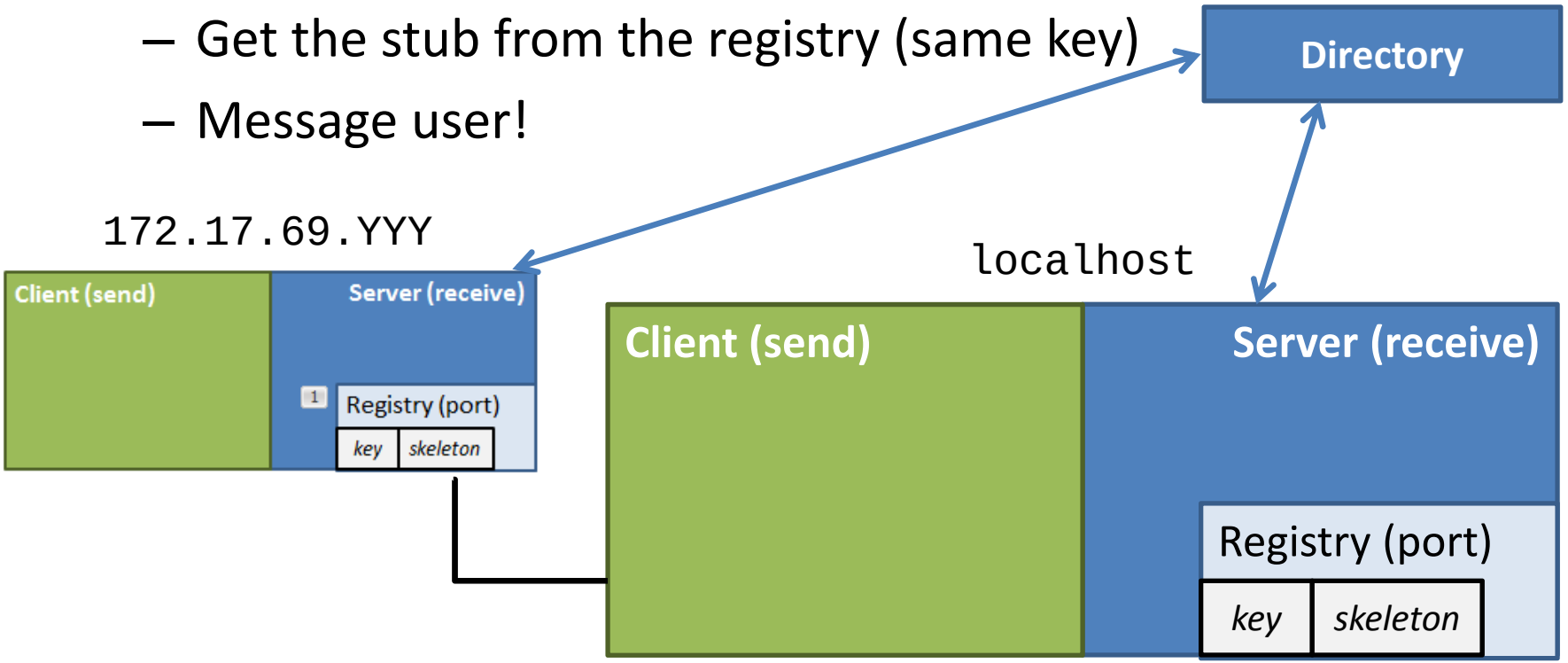
Step 4b: Register Skeleton

- `org.mdp.cli.InstantMessagingApp`
  - Implement `bindSkeleton(.)`
  - *key* = "`InstantMessagingServer`"
    - *Aka.* `InstantMessagingServer.class.getSimpleName();`
  - *skeleton* = `new InstantMessagingServer()`



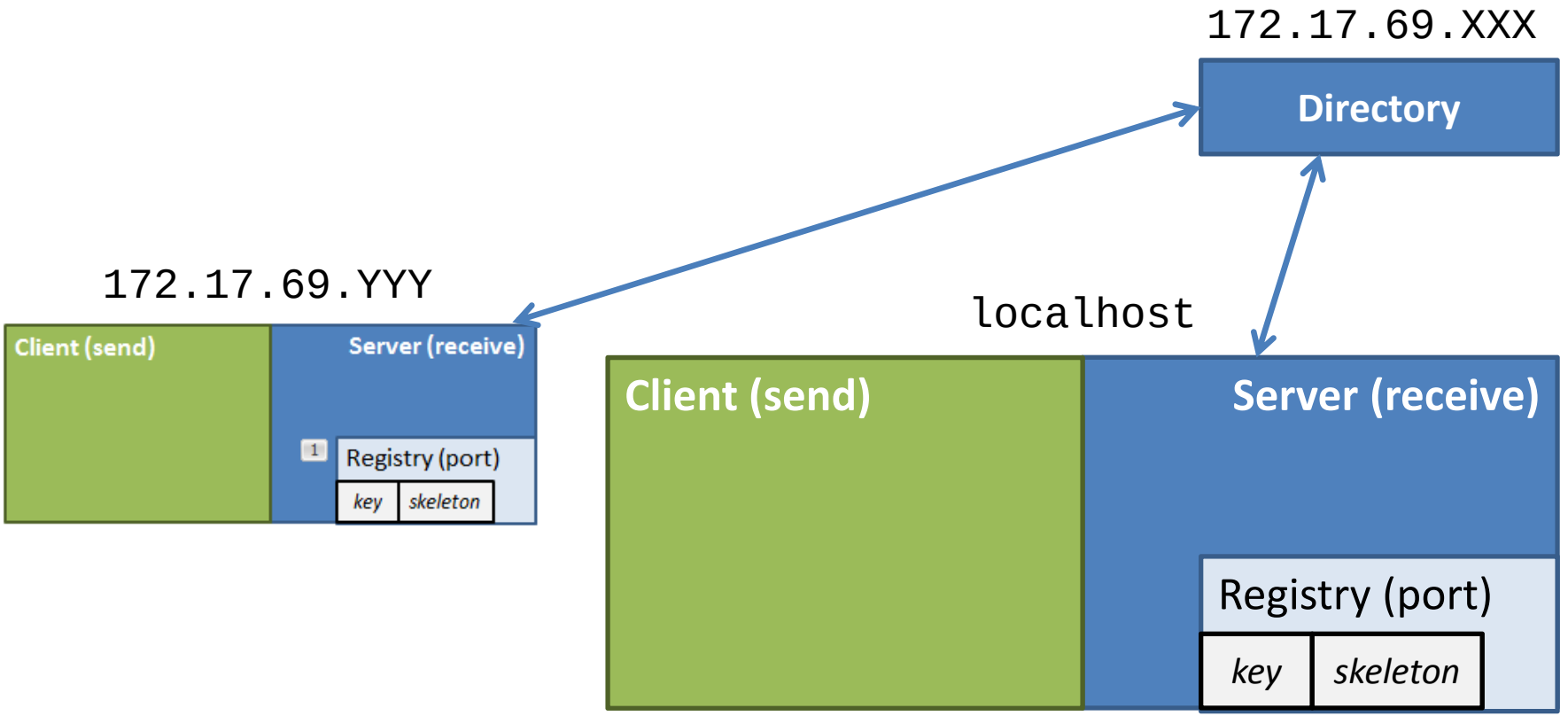
Step 4c: Implement Client

- `org.mdp.cli.InstantMessagingApp`
  - `messageUser(.,.,.)`
  - Connect to remote registry
  - Get the stub from the registry (same key)
  - Message user!



Step 4d: Find Peers w/ Directory

- `org.mdp.cli.InstantMessagingApp`
  - `connectToDirectory()`



Step 4e: Setup Command Line

- Run `InstantMessagingApp`
  - With argument: `-n [dir-location]`

- Add yourself
- Message classmates!

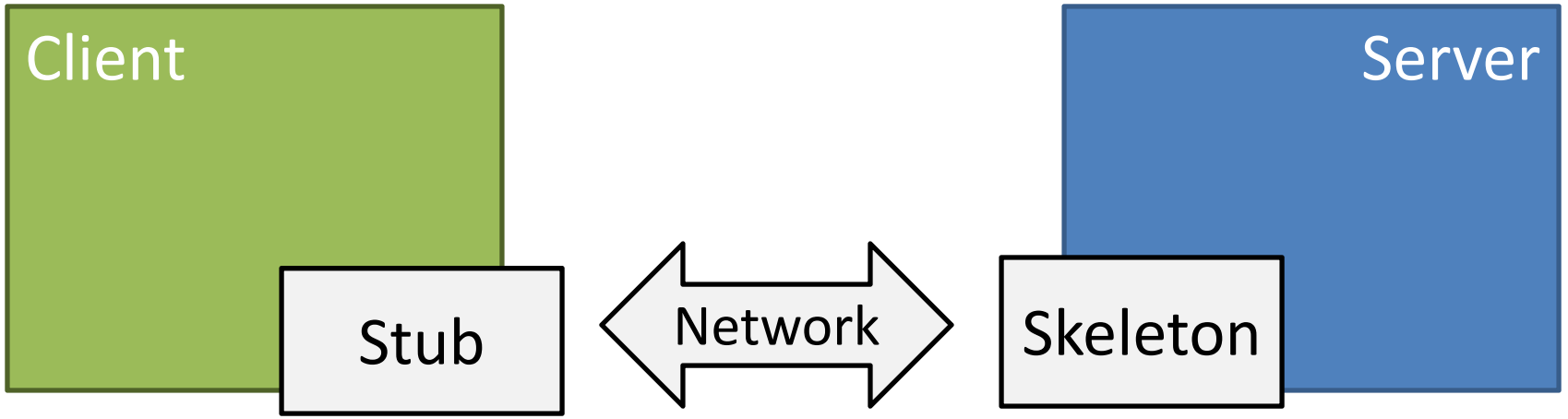
JAVA RMI OVERVIEW

Why is Java RMI Important?

We can use it to quickly build distributed systems using some standard Java skills.

What is Java RMI?

- RMI = Remote Method Invocation
- Remote Procedure Call (RPC) for Java
- Predecessor of CORBA (in Java)
- Stub / Skeleton model (TCP/IP)



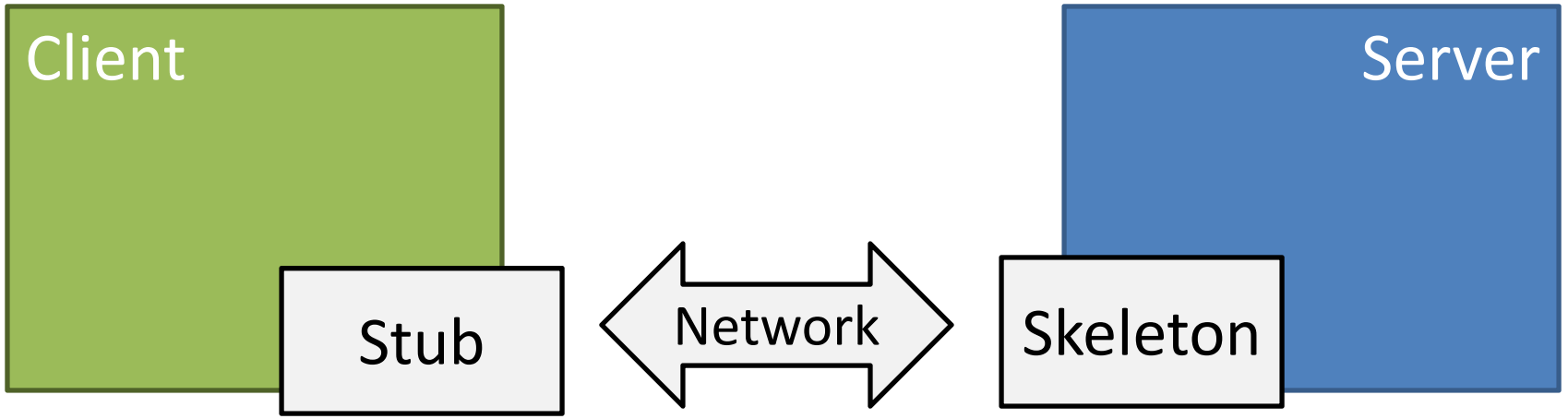
What is Java RMI?

Stub (Client):

- Sends request to skeleton: marshalls/serialises and transfers arguments
- Demarshalls/deserialises response and ends call

Skeleton (Server):

- Passes call from stub onto the server implementation
- Passes the response back to the stub



Stub/Skeleton Same Interface!

```java
package org.mdp.dir;

import java.io.Serializable;

/**
 * This is the interface that will be registered in the server.
 * In RMI, a remote interface is called a stub (on the client-side)
 * or a skeleton (on the server-side).
 *
 * An implementation is created and registered on the server.
 *
 * Remote machines can then call the methods of the interface.
 *
 * Note: every method *must* throw RemoteException!
 *
 * Note: every object passed or returned *must* be Serializable!
 *
 * @author Aidan
 *
 */
public interface UserDirectoryStub extends Remote, Serializable{
    public boolean createUser(User u) throws RemoteException;

    public Map<String,User> getDirectory() throws RemoteException;

    public User removeUserWithName(String un) throws RemoteException;
}
```

Client

Server

Server Implements Skeleton

```java
package org.mdp.dir;

import java.util.HashMap;

 * This is the implementation of UserDirectoryStub.
public class UserDirectoryServer implements UserDirectoryStub {

    private static final long serialVersionUID = -6025896167995177840L;
    private Map<String,User> directory;

    public UserDirectoryServer(){
        directory = new HashMap<String,User>();
    }

     * Return true if successful, false otherwise.
    public boolean createUser(User u) {
        if(u.getUsername()==null)
            return false;

        directory.put(u.getUsername(), u);

        System.out.println("New user registered! Bienvendio a ...\n\t"+u);
        return true;
    }

     * Returns the current directory of users.
    public Map<String, User> getDirectory() {
        return directory;
    }

     * Just an option to clean up if necessary!
    public User removeUserWithName(String un) {
        System.out.println("Removing username '"+un+"'. Chao!");
        return directory.remove(un);
    }
}
```

Problem?

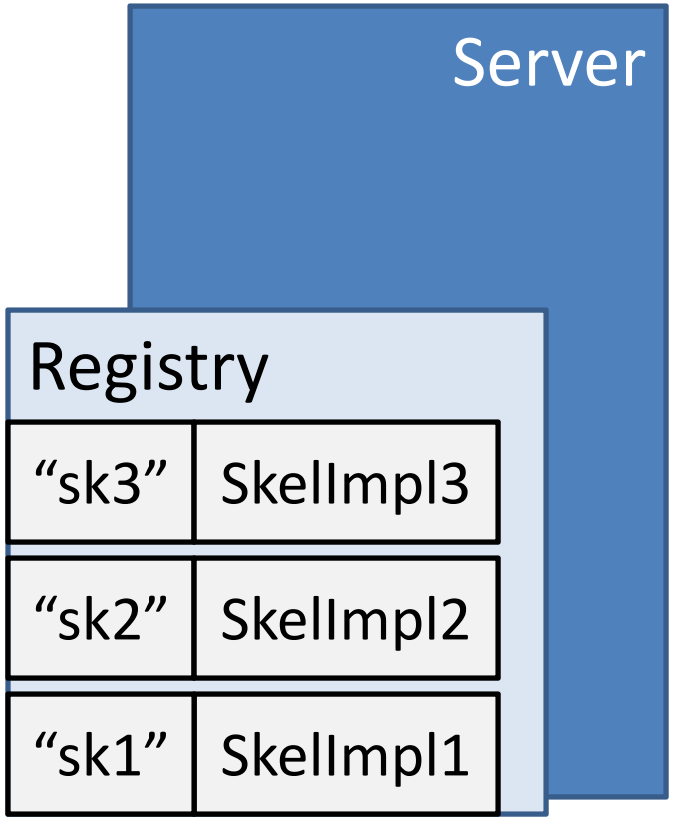
Synchronisation:
(e.g., should use ConcurrentHashMap)

[Thanks to Tomas Vera ☺]

Server

Server Registry

- Server (typically) has a Registry: a Map
- Adds skeleton *implementations* with key (a string)



Server Creates/Connects to Registry

```
// create registry
Registry registry = LocateRegistry.createRegistry(port);
```

OR

```
// connect to registry
Registry registry = LocateRegistry.getRegistry(hostname, port);
```

Server

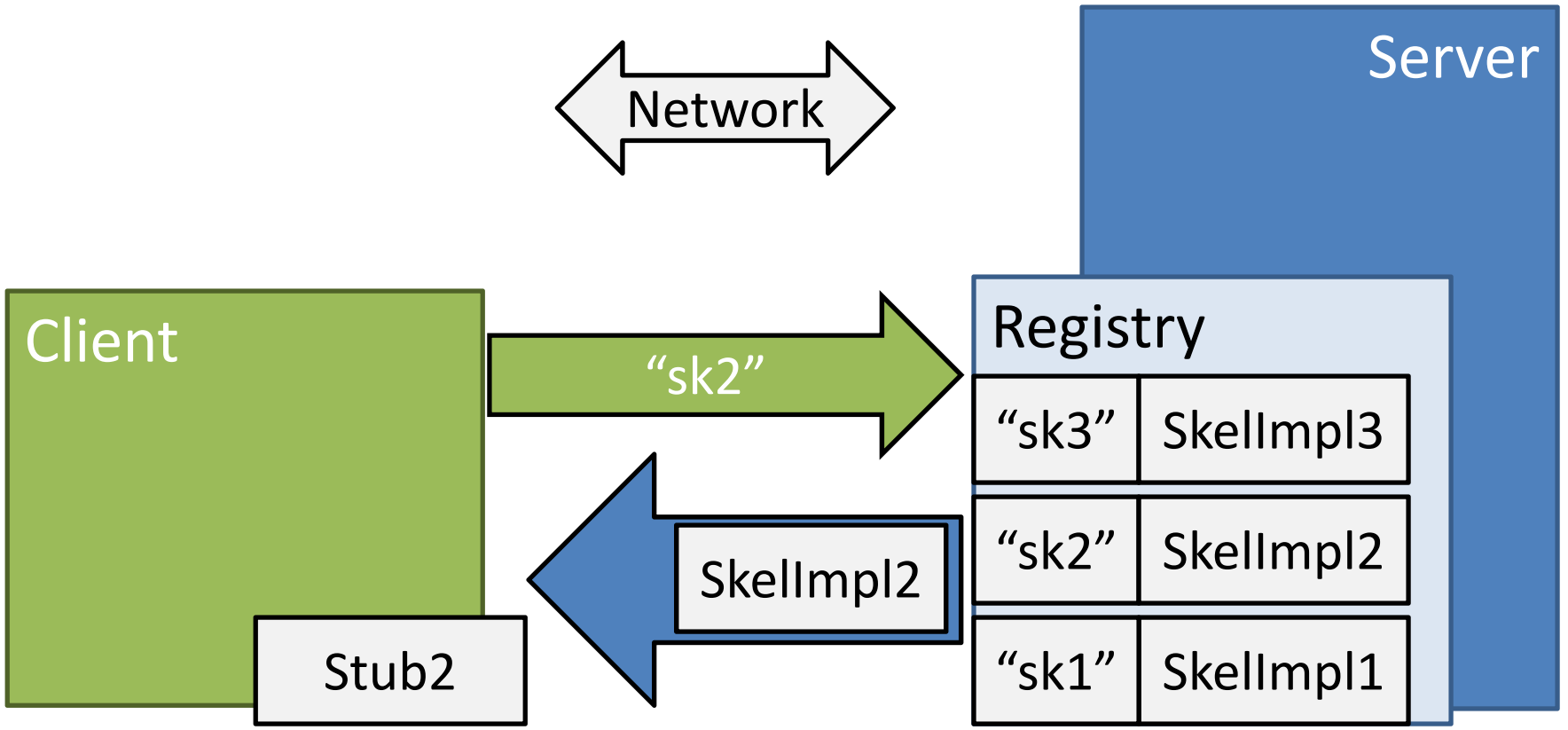
Server Registers Skeleton Implementation As a Stub

```
// create a remote stub to make it
// ready for incoming calls
Remote stub = UnicastRemoteObject.exportObject(new UserDirectoryServer(),0);

// register stub in registry under a key stub-name
String stubname = "mensaje";
registry.bind(stubname, stub);
```

Server

Client Connecting to Registry

- Client connects to registry (port, hostname/IP)!
- Retrieves skeleton/stub with key



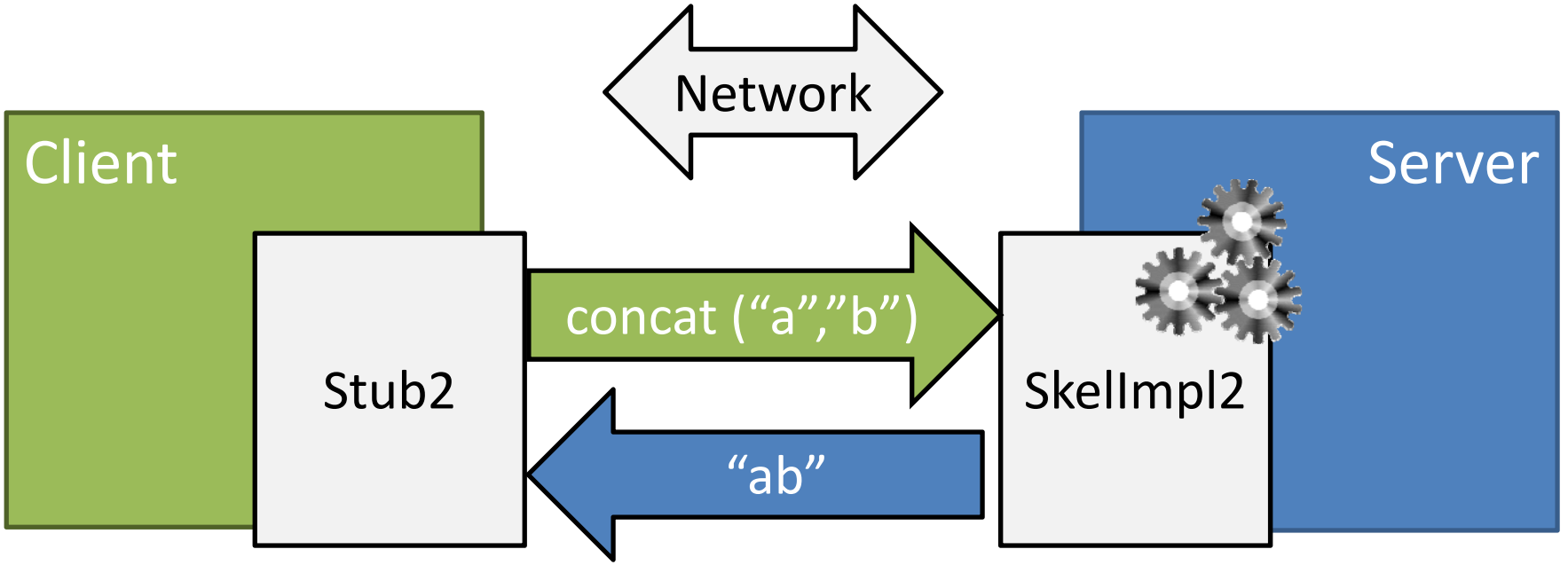
Client Connecting to Registry

```
String hostname = "server.com";
int port = 1985;
String stubname = "mensaje";

// first need to connect to the remote registry on the given
// IP and port
Registry registry = LocateRegistry.getRegistry(hostname, port);

// then need to find the interface we're looking for
UserDirectoryStub stub = (UserDirectoryStub) registry.lookup(stubname);
```

Client

Client Calls Remote Methods

- Client has stub, calls method, serialises arguments
- Server does processing
- Server returns answer; client deserialises result



Client Calls Remote Methods

```
// now we can use the stub to call remote methods!!
Map<String,User> users = stub.getDirectory();
System.err.println(users.toString());

User u = new User("aidhog", "Aidan Hogan", "10.0.114.59", 1509);
stub.createUser(u);

users = stub.getDirectory();
System.err.println(users.toString());

stub.removeUserWithName("aidhog");

users = stub.getDirectory();
System.err.println(users.toString());
```

Client

Java RMI: Remember …

1. Remote calls are pass-by-value, not pass-by-reference (objects not modified directly)
2. Everything passed and returned must be Serialisable (`implement Serializable`)
3. Every stub/skel method *must* throw a remote exception (`throws RemoteException`)
4. Server implementation can only throw `RemoteException`