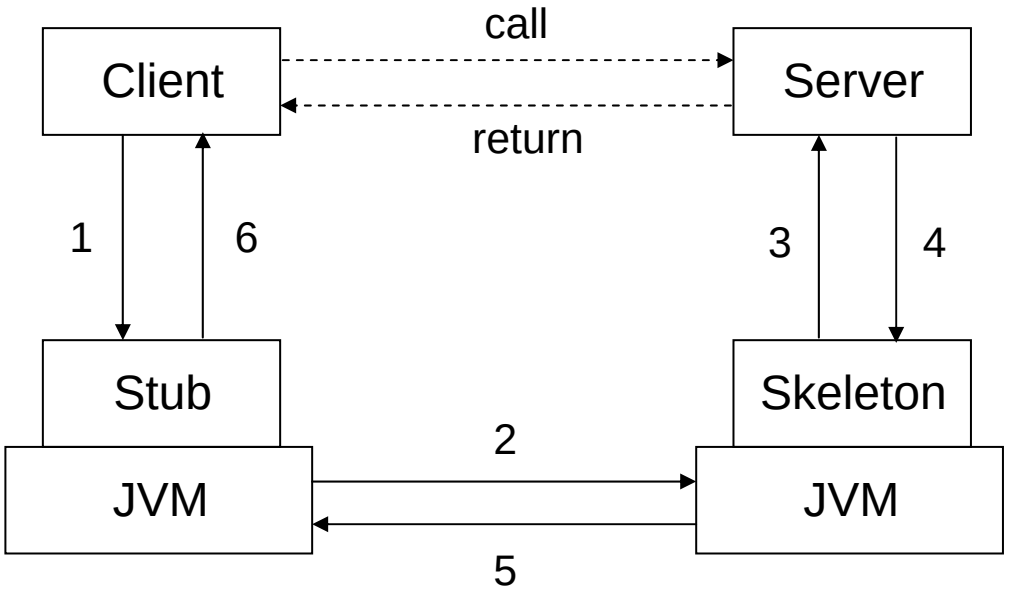
Java RMI
Remote Method Invocation

Remote Method Invocation

- Why RMI?
  - In socket programming, programmers have to make explicit connections between clients and servers and manage data transmission.
  - Thus, it's hard and error-prone to write socket programs.
  - Can the connection and data transmission be managed by JVM?

What's RMI?

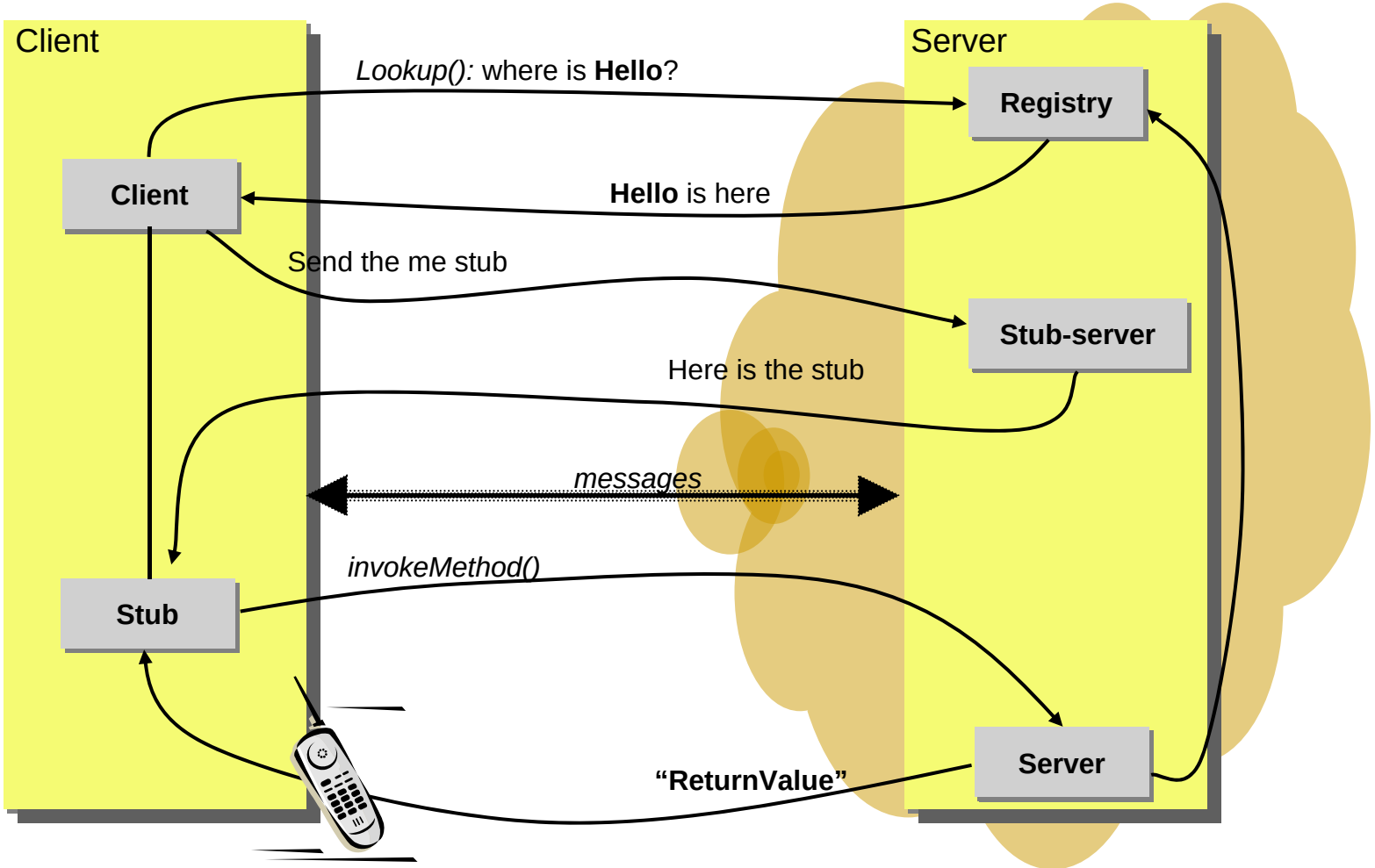
- Distributed programming model
  - to allow objects residing on different hosts (*remote objects*) to be manipulated as if they were all on the same host (*local objects*)
- RMI architecture



Local vs. Remote Objects

- Local objects
  - Objects accessible only within the local hosts
- Remote objects
  - Objects accessible from remote hosts
  - Instances of classes that implements a marker interface java.rmi.Remote
- Property of remote objects
  - Similar to local objects (arguments, downcasting, instanceof, etc)
  - Clients of remote objects interact with stubs
  - Passing arguments and results for RMI calls
    - Call by value for local objects (through serialization and deserialization)
    - Call by reference for remote objects

Java RMI in a Nutshell



Locating Remote Objects

- RMI registry
  - Directory service mapping RMI servers (or objects) to their names
  - Server: register itself to make it available to remote clients
  - Client: locate a server by looking up an RMI registry with a URL protocol rmi, e.g.,

    rmi://host:port/name

  - The programming interface by the class java.rmi.Naming

Method	Description
bind(name, obj)	Bind obj to name
rebind(name, obj)	Bind obj to name even if already bound
unbind(name)	Remove the binding
lookup(url)	Return object bound to url
list(url)	Return a list of all bindings

Writing RMI Programs

1. Define a remote interface, e.g.,

```
public interface Service extends java.rmi.Remote {
    public void doSomething(...) throws java.rmi.RemoteException;
    // ...
}
```

2. Define a service implementation class, e.g.,

```
public class ServiceProvider extends
    java.rmi.server.UniCastRemoteObject
    implements Service {
    public void doSomething(...) throws java.rmi.RemoteException {
       // ...
    }
    // ...
}
```

Writing RMI Programs (Cont.)

3. Create a server instance and register to an RMI registry, e.g.,

```
Service server = new ServiceProvider(...);
java.rmi.Naming.bind(name, server);
```

4. Generate the stub and skeleton classes by using the RMI compiler (rmic), e.g.,

```
% rmic ServiceProvider
```

The command produces:
ServiceProvider_Stub.class and ServiceProvider_Skel.class

Writing RMI Programs (Cont.)

5. Write a client program, e.g.,

```
java.rmi.Remote obj = java.rmi.Naming.lookup(name);
Service server = (Service) obj;
...
server.doSomething(...); // RMI call
...
```

Example -- A Simple Time Server

- Remote interface, TimeService

```
public interface TimeService extends java.rmi.Remote {
    java.util.Date getTime() throws java.rmi.RemoteException;
}
```

- Server and client classes

A Server Class, TimeServer

```
import java.rmi.*;
improt java.util.*;

public class TimeServer extends java.rmi.server.UnicastRemoteObject
    implements TimeService {

    public TimeServer() throws RemoteException {}

    public Date getTime() {  return Calendar.getInstance().getTime(); }

    public static void main(String [] args) {
        try {
            TimeServer server = new TimeServer();
            Naming.rebind("TimeServer", server);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

A Client Class, TimeClient

```
import java.rmi.*;
improt java.util.*;

public class TimeClient {

    public static void main(String [] args) {
        try {
            TimeService server =
                (TimeService) Naming.lookup("rmi://localhost/TimeServer");
            System.out.println(server.getTime());
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Compiling and Running

1. Compile the server and client programs, e.g.,

   % javac TimeServer.java TimeClient.java TimeService.java

2. Generates the stubs and skeletons, e.g.,

   % rmic TimeServer

3. Start the RMI registry on the server host, e.g.,

   % rmiregistry &

4. Run the server on the server host, e.g.,

   % java TimeServer &

5. Runt the client on the client host, e.g.,

   % java TimeClient

Serialization

- What is it?
  - Process of transforming an object into a stream of bytes; the reverse process is called *deserialization*.
  - Allows objects to be saved to files or sent to remote hosts over a network (e.g., arguments to RMI calls)
- How to make objects serializable?
  - By implementing the marker interface java.io.Serializable
  - A default implementation for (de) serialization is automatically provided.
  - Can customize the process by implementing readObject() and writeObject() methods:

```
private void writeObject(java.io.ObjectOutputStream out)
  throws IOException;
private void readObject(java.io.ObjectInputStream in)
  throws IOException, ClassNotFoundException;
```

Example

- Make the following class Student serializable

```
public class Student {
    private String name;
    private int score;
    //@ private invariant 0 <= score && score <= 100;
    private char grade;
    //@ private invariant (* grade is one of 'A', ..., 'F' *);
    // ...
}
```

- Answer 1:

```
public class Student implements Serializable {
    // ...
}
```

Example (Cont.)

- Answer 2:

```
public class Student implements Serializable {
    // ...
    private void writeObject(java.io.ObjectOutputStream out) throws IOException {
       out.writeUTF(name);
       out.writeInt(score);
       out.writeChar(grade);
    }
   private void readObject(java.io.ObjectInputStream in)
      throws IOException, ClassNotFoundException {
       name = in.readUTF();
       score = in.readInt();
       grade = in.readChar();
   }
}
```

Example (Cont.)

- Answer 3:

```
public class Student implements Serializable {
    // ...
    private void writeObject(java.io.ObjectOutputStream out) throws IOException {
        out.writeUTF(name);
        out.writeInt(score);
    }
    private void readObject(java.io.ObjectInputStream in)
        throws IOException, ClassNotFoundException {
        name = in.readUTF();
        score = in.readInt();
        grade = calculateGrade(score);
    }
    private char calculateGrade(int score) { /* ... */ }
}
```

Using Serialization

- Serializaing objects

```
ObjectOutputStream out = new ObjectOutputStream(/* ... */);
Student s = new Student(/* ... */);
out.writeObject(s);
// ...
```

- Deserializing objects

```
ObjectInputStream in = new ObjectInputStream(/* ... */);
Object obj = in.readObject();
Student s = (Student) obj;
// ...
```

Remote Method Invocation

- RMI protocol interface lets Java objects on different hosts communicate with each other in a transparent way
- Clients can invoke methods of a remote object as if they were local methods
- Preserve the object oriented paradigm in distributed computing

Java RMI Optimization

- Protocol
  - Use of Mediators to minimize the exchange of data through the wireless link.
- Data Communication
  - Optimized Communication: Compress and Optimize data communication
- Stub&Class Loading
  - If possible, avoid to download stubs