Remote Method Invocation Part II

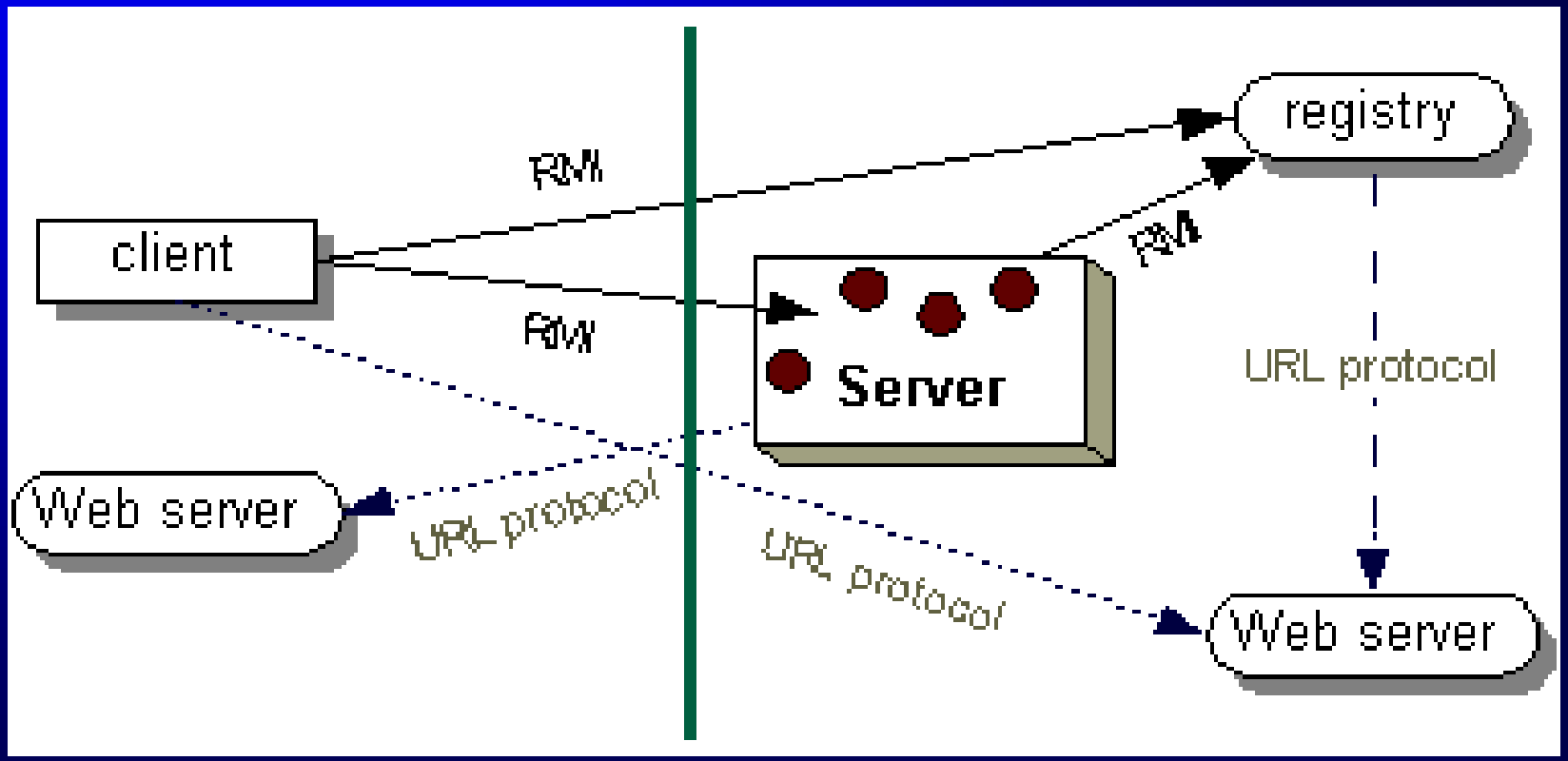
Based on Java Network Programming and Distributed Computing Chapter 11

Also based on Sun's Online Java Tutorial

Topics

- RMI in Detail
  - Packages and classes (and exceptions!)
- The RMI Registry
- Implementing callbacks
- “Activating” remote objects
- Distributed garbage collection
- Deployment issues

RMI Architecture (Wollrath and Waldo)



RMI Packages Overview

- `java.rmi`
  - General RMI classes and exceptions.
- `java.rmi.server`
  - RMI server-specific classes and interfaces.
- `java.rmi.registry`
  - To access, launch, and locate RMI registries.
- `java.rmi.activation`
  - To start remote services on demand.
- `java.rmi.dgc`
  - To support distributed object garbage collection.

`java.rmi` Package

- `Remote` interface
  - To identify a service as remotely accessible.
- `RemoteException` class
  - `java.io.IOException` subclass, superclass of most RMI exceptions.
- `MarshalledObject` class
  - Includes the annotated codebase for dynamic class loading

java.rmi Package

- Naming class
  - Static methods to assign or retrieve object references of the RMI object registry (rmiregistry).
  - bind(String url, Remote obj)
    - Inserts a registry entry and binds it to given obj.
  - rebind(String url, Remote obj)
    - Does not throw AlreadyBoundException.
  - Remote lookup(String url)
    - Returns a reference for the remote object
  - Also unbind(url), list(url)

java.rmi Package

- `RMISecurityManager` class
  - Dynamic class loading requires a security manager to be registered with the JVM.
  - Default security manager protects rogue code from:
    - Initiating network connections
    - Masquerading as servers
    - Gaining file access
  - More restrictive than applets, but may be modified to grant additional privileges by using a security policy file.

`java.rmi` Exceptions

- `ServerError` class
  - An error in the RMI server was thrown (e.g. out of memory)
- `ServerException` class
  - When a method call to an RMI server throws a `RemoteException`, a `ServerException` is thrown.
- `UnexpectedException` class
  - Used by clients to represent an exception thrown by the remote method but not declared in the RMI interface.

`java.rmi` Exceptions

- `MarshalException` class
  - Exception while marshalling parameters of a remote method call, or when sending a return value.
  - At the client end, it is impossible to tell whether the method was invoked by the remote system --a subsequent invocation may cause the method to be invoked twice.
- `UnmarshalException` class
  - Exception while unmarshalling arguments of a remote method call, or when sending a return value.
- `NoSuchObjectException` class
  - A remote object no longer exists.
  - This indicates the method never reached the object, and may be re-transmitted at a later date, without duplicate invocations.

`java.rmi` Exceptions

- `AccessException` class
  - Thrown by naming to indicate that a registry operation cannot be performed.
- `AlreadyBoundException` class
  - A remote object is already bound to a registry entry.
- `ConnectException` class
  - Inability to connect to a remote service, such as a registry.
- `NotBoundException` class
  - Attempts to lookup or unbind a non-existent registry entry.

`java.rmi` Exceptions

- `UnknownHostException` class
  - A client making a remote method request can't resolve the hostname.
- `StubNotFoundException` class
  - Stub not in local file system or externally (if using dynamic class loading).
- `ConnectIOException` class
  - Inability to connect to a remote service to execute a remote method call.

`java.rmi.server` Package

- `RemoteRef` interface
  - A handle to a remote object. Used by stubs to issue method invocations on remote objects.
- `RMIClientSocketFactory` interface
- `RMIServerSocketFactory` interface

`java.rmi.server` Package

- `RMISocketFactory` class
  - Implements RMI client and server socket factory interfaces.
  - Enables customized sockets to be used by RMI, e.g., providing encryption, or communication through firewalls.
  - By default, three mechanisms are attempted:
    - A direct TCP connection
    - An HTTP connection using the port number of the service (e.g., http://server:1095/).
    - A modified HTTP connection using default port and a CGI script (e.g., http://server:80/cgi-bin/java-rmi.cgi)

`java.rmi.server` Package

- `RemoteObject` class
  - Implements the `Remote` interface.
  - Overrides Object methods making them “remote” aware, e.g., `equals, hashCode, toString`.
  - `RemoteRef getRef()`
    - returns a reference to the object.
  - `static Remote toStub(Remote obj)`
    - Returns a stub for the object. If invoked before the object is exported, throws a NoSuchObjectException.

`java.rmi.server` Package

- `RemoteServer` class
  - Extends `RemoteObject`. Superclass of `Activatable` and `UnicastRemoteObject`.
  - `String getClientHost()`
    - Returns the location of the RMI client.
    - Allows to handle requests differently based on the IP address of the client.

Beware of IP spoofing!

  - `setLog(OutputStream out)` logs RMI calls including time, date, IP address, and method.
  - `PrintStream getLog()` returns the RMI logging stream; writing to it automatically includes the date and time.

`java.rmi.server` Package

- `UnicastRemoteObject` class
  - Extends `RemoteServer`. Base class for most RMI service implementations.
  - Provides specialized constructors to export a service on a specific port, or to use a specialized socket factory.
    - UnicastRemoteObject(port)
    - UnicastRemoteObject(port, csf, ssf);

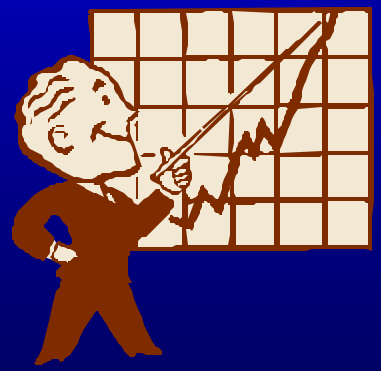
java.rmi.registry

- `Registry` interface
  - For accessing a registry service.
- `LocateRegistry` class
  - To create a new RMI registry, or locate an existing one.
  - A registry can be launched by a server (rather than separately using `rmiregistry`).
    - createRegistry([[port][,csf,ssf]])
    - getRegistry([host][,port])

Default host is localhost and default port is 1099

Implementing callbacks

- “Mr. Broker, whenever the stock price for MyDot.com gets out of the $5-$100 range, give me a phone call!”



Defining a Listener (client) interface

- This `Remote` interface defines the method(s) to be invoked from the server to the client, when an event happens.

Defining a Service (server) interface

- This is the same as the normal RMI `Remote` interface to export a given service, except that methods for adding and removing a `Listener` remote object are included.

Implementing the Listener interface

- The code is the same as a traditional RMI client, except that a Listener object is registered with the remote service.
- How?
  - Invoking a remote method on the server (`register(Listener)`) and passing the Listener object as an argument to it.

Recall that Remote parameter passing is by reference!

Implementing the Service interface

- This is your normal remote service implementation. It needs to:
  - Keep a list of event listeners
  - Provide methods to add and remove listeners
  - Implement the remote service
  - Detect relevant state changes and notify listeners as appropriate.

BankAccountMonitor Example

- The goal is to notify a bank account monitor whenever the balance becomes less than $100.
- See:
  - `BankAccountMonitor` interface
  - `BankAccount` interface
  - `BankAccountImpl` class
  - `BankAccountMonitorImpl` class
  - `Deposit` class

Remote Object Activation

- Why?
  - To free resources from servers with seldom-used services.
  - To enable devices with limited resources to activate multiple kinds of services.

java.rmi.activation

- `Activatable` class
- `ActivationDesc` class
- `ActivationID` class
- `ActivationGroup` class
- `ActivationGroupDesc` class
- `ActivationGroupID` class
- `ActivationSystem` interface

Remote Object Activation

Transparent to RMI clients.

- Remote interface/client code is the same.

Server code needs modifications:

- Extends Activatable class
- Constructor receives ActivationID, MarshalledObject.
- Main method steps:
  - Create ActivationGroupDesc
  - Register activation group descriptor with ActivationSystem
  - Create an ActivationGroup
  - Create an ActivationDesc with class name, codebase
  - Register the activation descriptor with ActivationSystem
  - Register the stub (Returned in previous step) in registry.

Remote Object Activation

For an example and more documentation, please see:

http://java.sun.com/j2se/1.4/docs/guide/rmi/activation.html

Also, JNPDC textbook pp.365-376.

Distributed Garbage Collection

- Remote service developers don’t need to track remote object clients to detect termination.
- RMI uses a reference-counting garbage collection algorithm similar to Modula-3's Network Objects. (See "Network Objects" by Birrell, Nelson, and Owicki, *Digital Equipment Corporation Systems Research Center Technical Report 115*, 1994.)

java.rmi.dgc

- `Lease` class
  - A remote object is offered to a client for a short duration of time (called a *lease*). When the lease expires, the object can be safely garbage-collected.
- `VMID` class
  - To uniquely identify a Java virtual machine.
  - `boolean isUnique()` represents whether the generated VMID is truly unique. If and only if an IP address can be determined for the host machine.

Distributed Garbage Collection

- When a reference to a remote object is created in a JVM, a `referenced` message is sent to the object server.
- A reference count keeps track of how many local references there are.
- When the last reference is discarded, an `unreferenced` message is sent to the server.

Distributed Garbage Collection

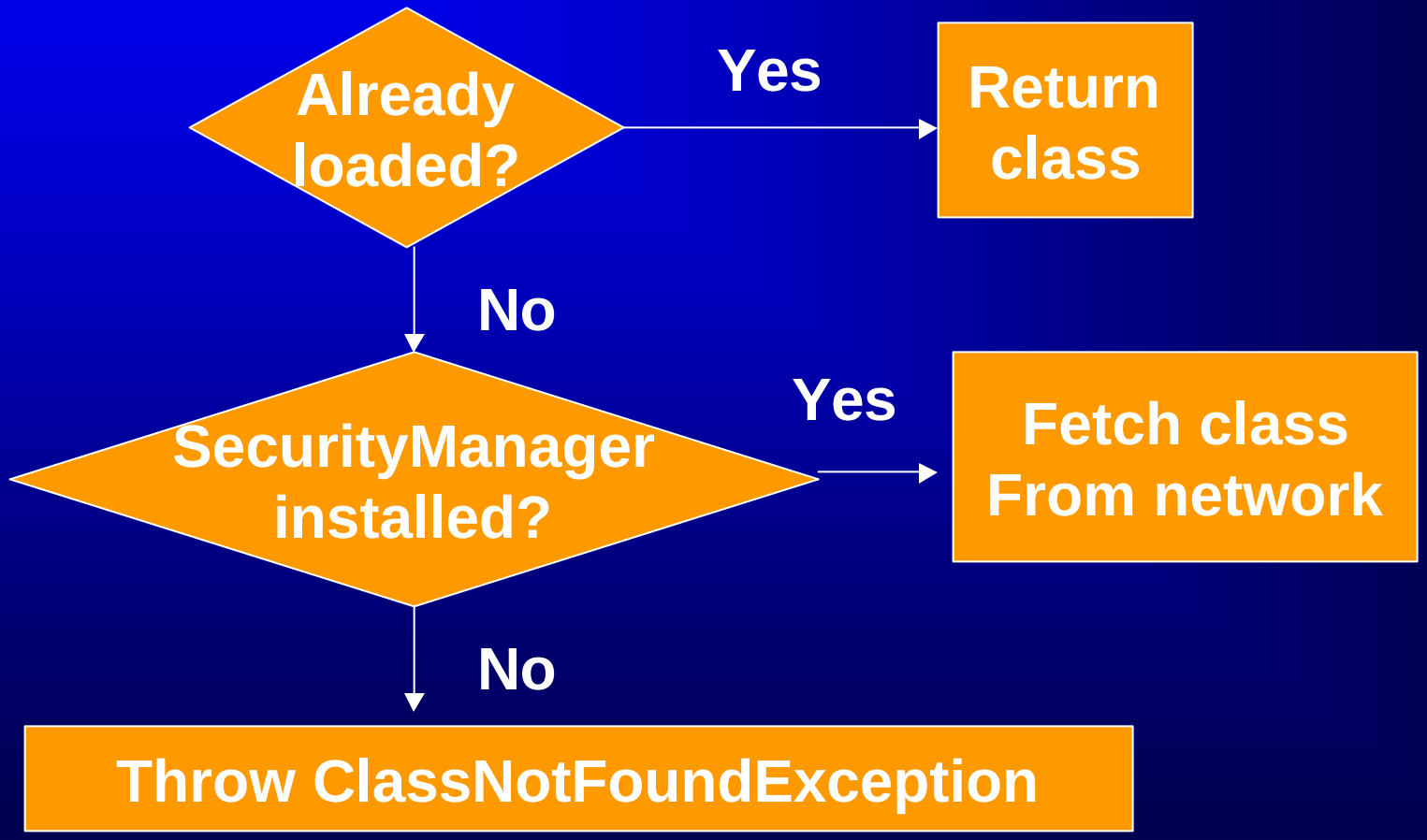
- When a `Remote` object is not referenced by any client, the run-time refers to it as a *weak* reference.
- The weak reference allows the JVM's garbage collector to discard the object if no other local references exist.

Network partitions may cause premature `Remote` object collections.

RMI Deployment Issues

- Dynamic Class Loading
  - What happens if a new object is passed using RMI, and the defining class is not available to the remote system?
  - Recall that you can pass an object with an interface type (e.g., Runnable) which can have multiple implementations.
  - We need a way to download such code dynamically.

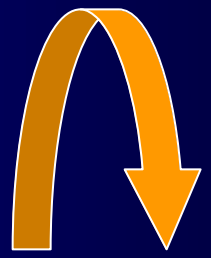
Dynamic Class Loading



Where to download code from?

- Setting the system property
  - `java.rmi.server.codebase`
- For example:

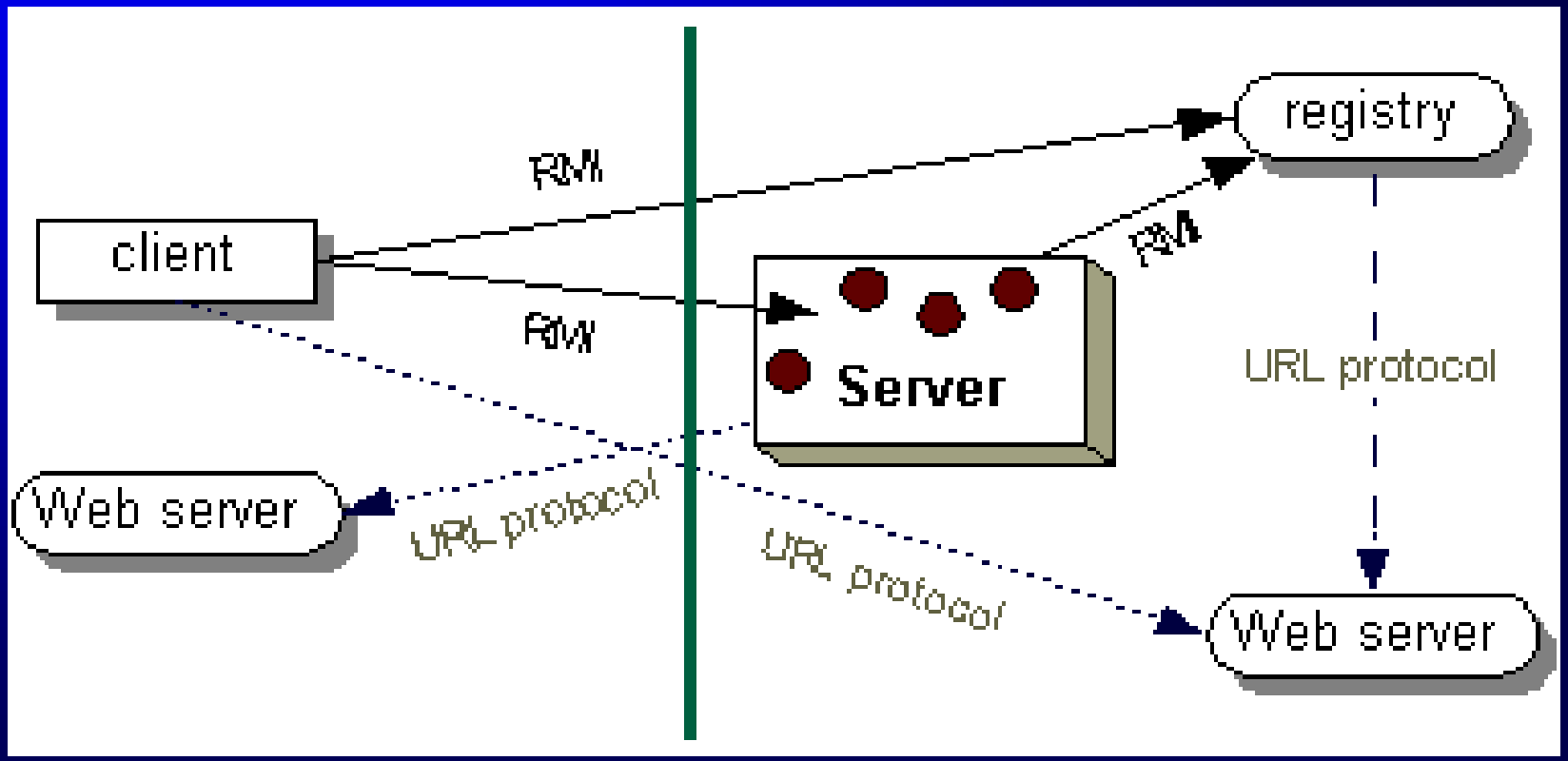
```
java -Djava.rmi.server.codebase
=http://www.cs.rpi.edu/~joe/classes/
  MyRemoteImpl
```



A single line!!

Don’t forget to install a Security Manager

RMI Architecture Revisited



RMI Deployment: Differences in Java Virtual Machines

- Microsoft JVMs do not generally support RMI – even though RMI is part of the “core” Java API.
  - Solution: A patch to IE is available.

- JDK1.02 and JDK1.1 are not RMI-compatible.
  - `UnicastRemoteServer` replaced by `UnicastRemoteObject`.
  - Solution: Upgrade!

RMI Deployment: Differences in Java Virtual Machines

- JDK1.1 and Java 2 are not RMI-compatible.
  - New `RMISecurityManager` is more strict.
  - Solutions:
    - Remove the `RMISecurityManager` entirely (which disables dynamic class loading).
    - Replace the `RMISecurityManager` with a custom one, enabling restricted access to the network and file system.
    - Specify a security policy file, which allows network access and (optionally) file access.

Best option!

Sample Security Policy File

```
grant {
     permission java.net.SocketPermission
"*:1024-65535", "connect,accept";
     permission java.net.SocketPermission
"*:80", "connect";
};
```

Another Security Policy File

```
grant {
     permission java.net.SocketPermission
"*:1024-65535", "connect,accept";
     permission java.io.FilePermission
"c:\\home\\ann\\public_html\\classes\\-",
"read";
     permission java.io.FilePermission
"c:\\home\\chu\\public_html\\classes\\-",
"read";
};
```

Yet Another Security Policy File

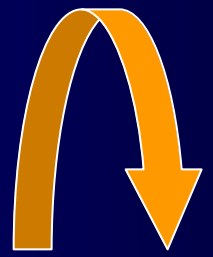
```
grant {
     permission java.security.AllPermission;
};
```

Not recommended in combination with dynamic class loading!

Where does RMI read the security policy from?

- Setting the system property
  - `java.security.policy`
- For example:

```
java -Djava.rmi.server.codebase
=http://www.cs.rpi.edu/~joe/classes/
  -Djava.security.policy=my.policy
  MyRemoteImpl
```



A single line!!

Deployment Issues: RMI, Applets, and Firewalls

- Applets cannot bind to TCP ports
  - ➡ an RMI service cannot run inside an applet.
- Applets cannot connect to arbitrary hosts
  - ➡ An applet can only be an RMI client to services hosted by the HTTP server serving the applet.
- Firewalls restrict connections to arbitrary ports.
  - ➡ A solution is to tunnel RMI requests through HTTP (a CGI script is available from Sun's Java RMI page).

An order of magnitude slower!