Distributed Systems

Java RMI

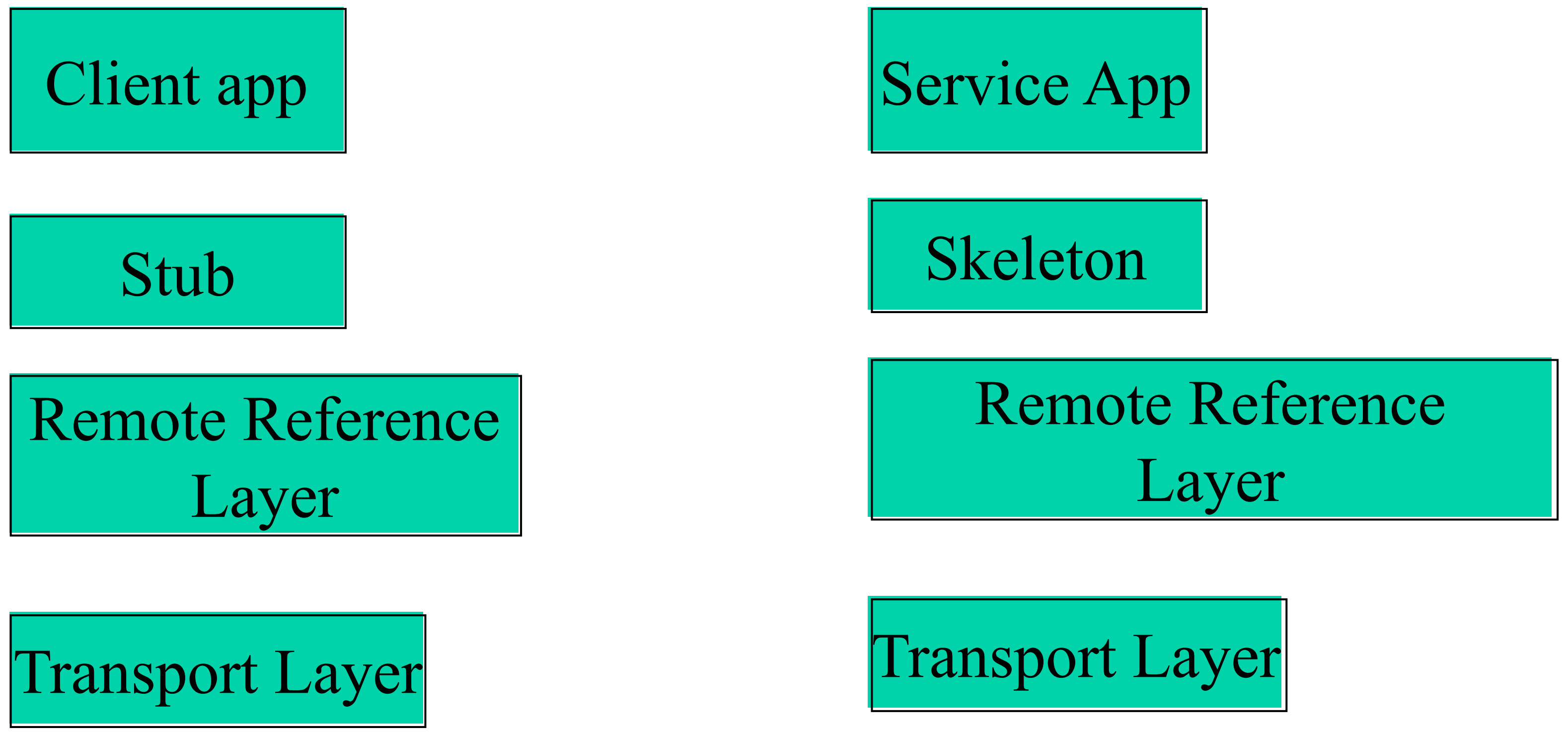
Goals/Principles Of RMI

- Distributed Java
- Almost the same syntax and semantics used by non-distributed applications
- Allow code that defines behavior and code that implements behavior to remain separate and to run on separate JVMs
- The transport layer is TCP/IP

Goals/Principles Of RMI

- On top of TCP/IP, RMI originally used a protocol called Java Remote Method Protocol (JRMP). JRMP is proprietary.
- For increased interoperability RMI now uses the Internet Inter-ORB Protocol (IIOP). This protocol is language neutral and runs on TCP/IP providing a standard way to make method calls to remote objects.
- RMI is all about remote calls at runtime. It's not about compilation against a remote class.

Protocol Layers



Goals/Principles Of RMI

- RMI uses the proxy design pattern. An object in one context is represented by another (the stub) in a separate context. The stub knows how to forward method calls between the participating objects.

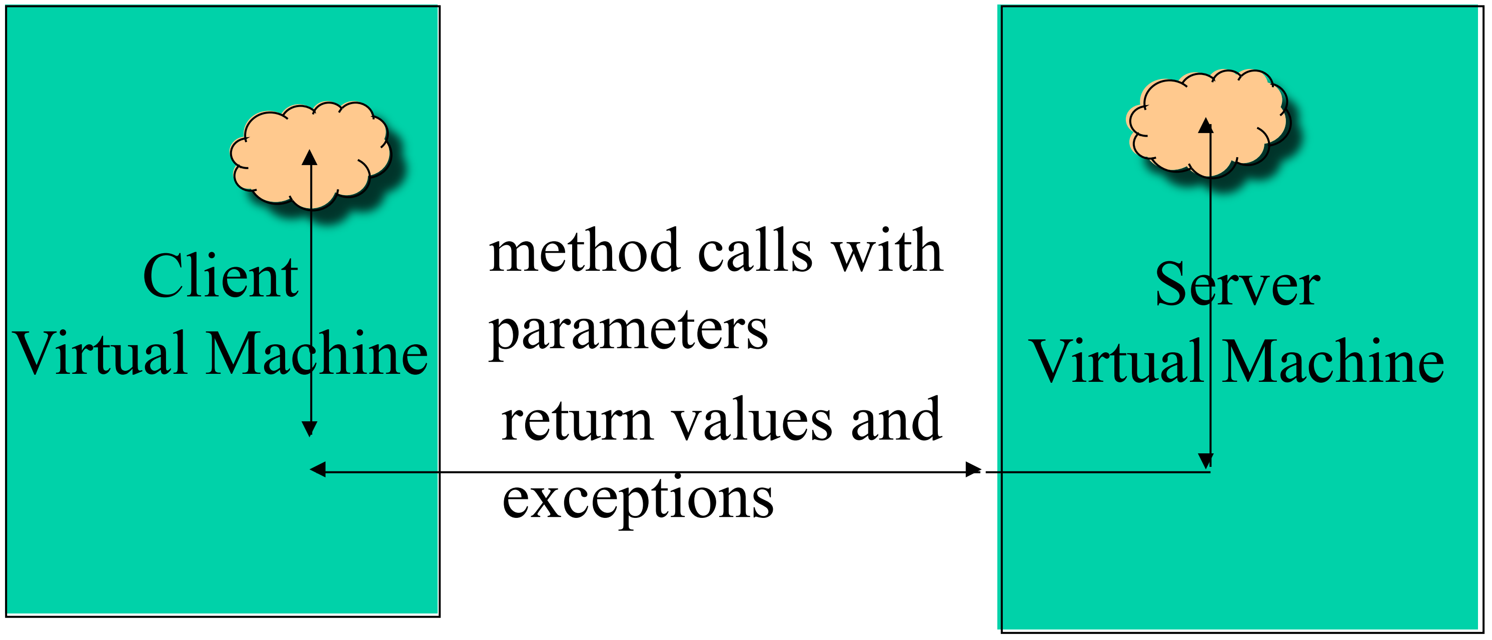
Goals/Principles Of RMI

- A naming or directory service is run on a well-known host and port number
- Usually a DNS name is used instead of an IP address
- RMI itself includes a simple service called the RMI Registry, rmiregistry. The RMI Registry runs on each machine that hosts remote service objects and accepts queries for services, by default on port 1099

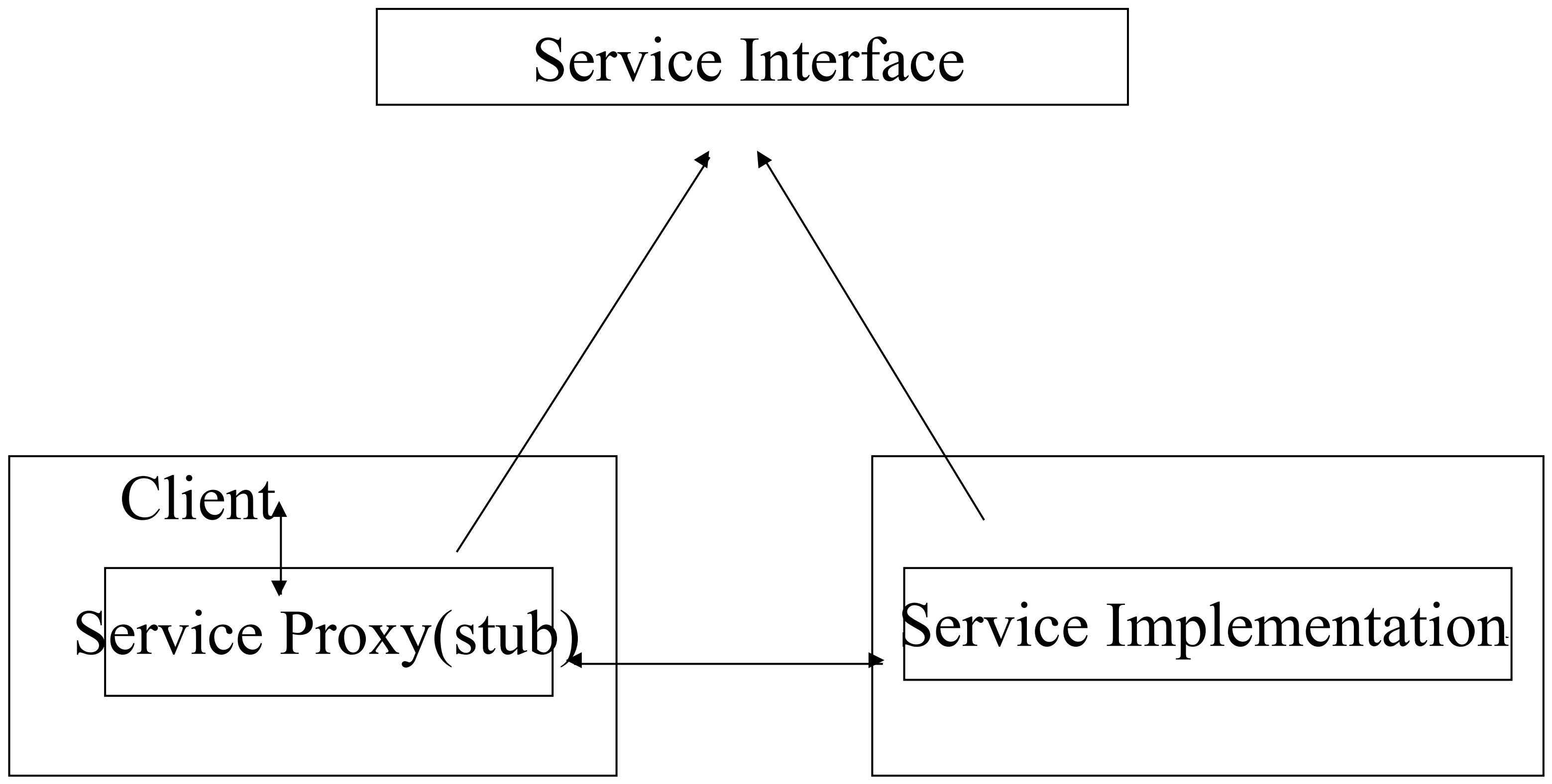
Goals/Principles Of RMI

- On the client side, the RMI Registry is accessed through the static class Naming. It provides the method lookup() that a client uses to query a registry.
- The registry is not the only source of remote object references. A remote method may return a remote reference.
- The registry returns references when given a registered name. It may also return stubs to the client.

Java RMI


Client
Virtual Machine
method calls with
parameters
return values and
exceptions
Server
Virtual Machine

The Proxy Design Pattern



Summary of Activities

1. Compile the java files:

   ```
   javac *.java
   ```

2. Run rmic on the ProductImpl.class producing the file ProductImpl_Stub.class

   ```
   rmic –v1.2 ProductImpl
   ```

3. Start the RMI registry

   ```
   start rmiregistry
   ```

4. Start the server

   ```
   start java ProductServer
   ```

5. Run the client

   ```
   java –Djava.security.policy=client.policy ProductClient
   ```

Parameter Passing in Remote Methods

When a remote object is passed from the server, the client receives a stub (or already has one locally):

```
        Product c1 = (Product)Naming.lookup(url + "toaster");
```

Using the stub, it can manipulate the server object by invoking remote methods. The object, however, remains on the server.

Parameter Passing in Remote Methods

It is also possible to pass and return *any* objects via a remote method call, not just those that implement the remote interface.

The method call

```
c1.getDescription()
```

returned a full blown String object to the client. This then became the client's String object. It has been copied via java serialization.