
JavaOne℠
Sun's Worldwide Java Developer Conference

Remote Objects for Java™

Peter B. Kessler
Roger Riggs

Java IDL and Java RMI

Java™ IDL and Java RMI are complementary technologies

- Java IDL is a heterogeneous solution
  - Uses a standard, language-neutral, interface description language
  - Uses open, standard, wire protocols to interact with services written in many languages
- Java RMI is a Java-only solution
  - Uses Java interfaces and data types to describe remote interfaces
  - Uses a specialized protocol to transmit parameters and class data

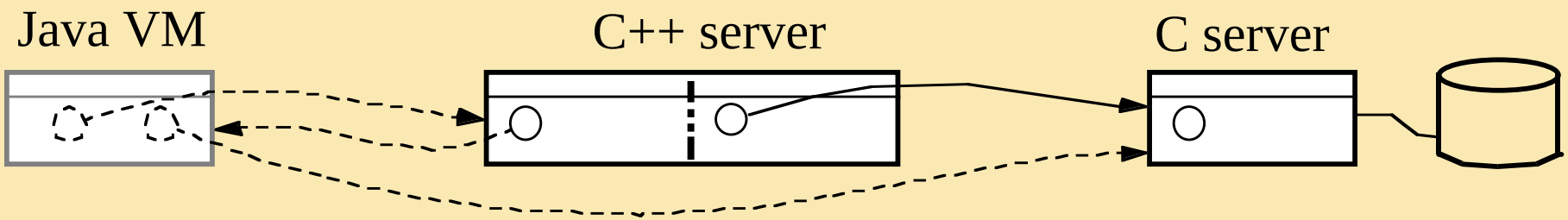
JavaOne℠

Sun's Worldwide Java Developer Conference

Java IDL and the Java ORB

Peter B. Kessler

The Problems



- Access to, and delivery of, network services
  - Written on heterogeneous platforms
  - Written in heterogeneous languages
- Java-compatible virtual machines in isolation
  - Applets talking to remote services
  - Applications talking to other applications
- Client-side system administration

What is IDL?

Language-neutral Interface Definition Language

- CORBA standard, specified by Object Management Group (600+ companies)
- Data, data structures, interfaces to objects (collections of methods), inheritance, exceptions
  - + data passed by-value, objects passed by reference
  - + parameter modes, standard exceptions
- Compilers map from IDL to various languages
- A client may be in a different language than the server

What is Java?

- Portable
- Ubiquitous
- Down-loadable
- Safe
- Statically-typed
- Full-featured programming language
- Run-time libraries

What is IDL for Java?

The mapping from IDL to the Java language

- The API's to which a client writes to use IDL objects from Java
- The API's to which a server writes to implement IDL objects in Java
- Third parties may add value above the basic mapping, or via server frameworks, etc.

Mapping IDL to Java

Goal was a simple, natural mapping for programmers using the Java language

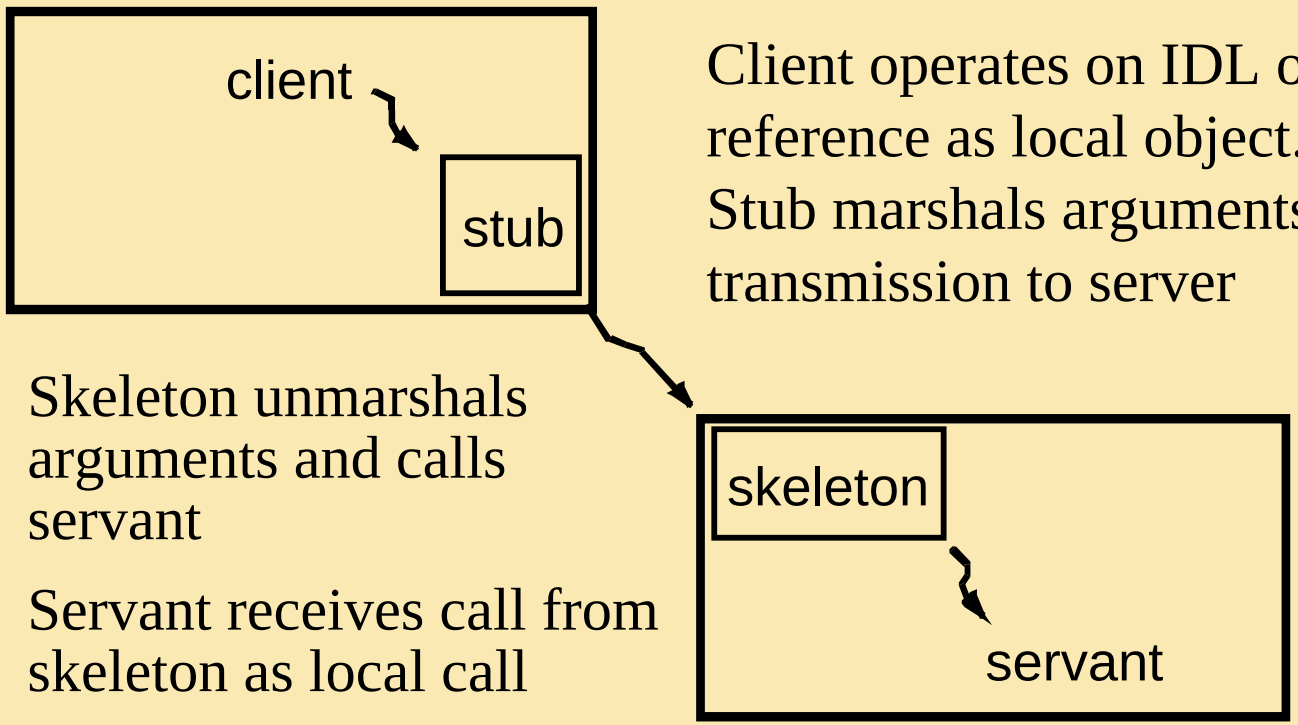
Example:

```
interface Ballot {
    enum Choice { favor, oppose, abstain };
    exception TimedOut { };
    void mark(in Choice selection) raises (TimedOut);
};
```

```
public void vote(BallotRef aBallot) {
   try {
       aBallot.mark(Ballot.Choice.favor);
   } catch (omg.corba.SystemException that) {
       ...
   }
}
```

What Are Stubs and Skeletons?

*Stubs and skeletons are **generated** by an IDL compiler from the description of an interface*



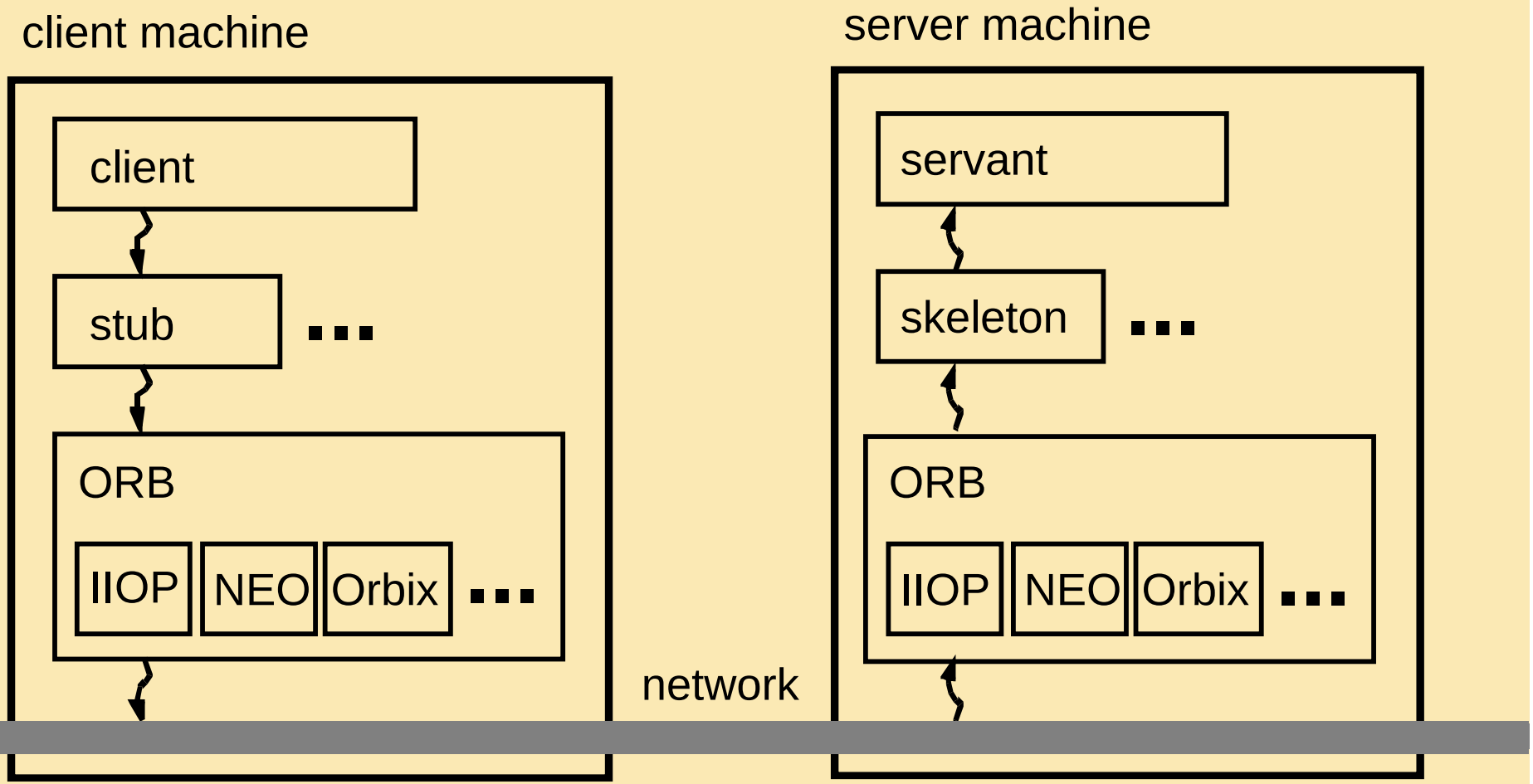
Client operates on IDL object reference as local object. Stub marshals arguments for transmission to server

Skeleton unmarshals arguments and calls servant

Servant receives call from skeleton as local call

And the process is reversed for returning results

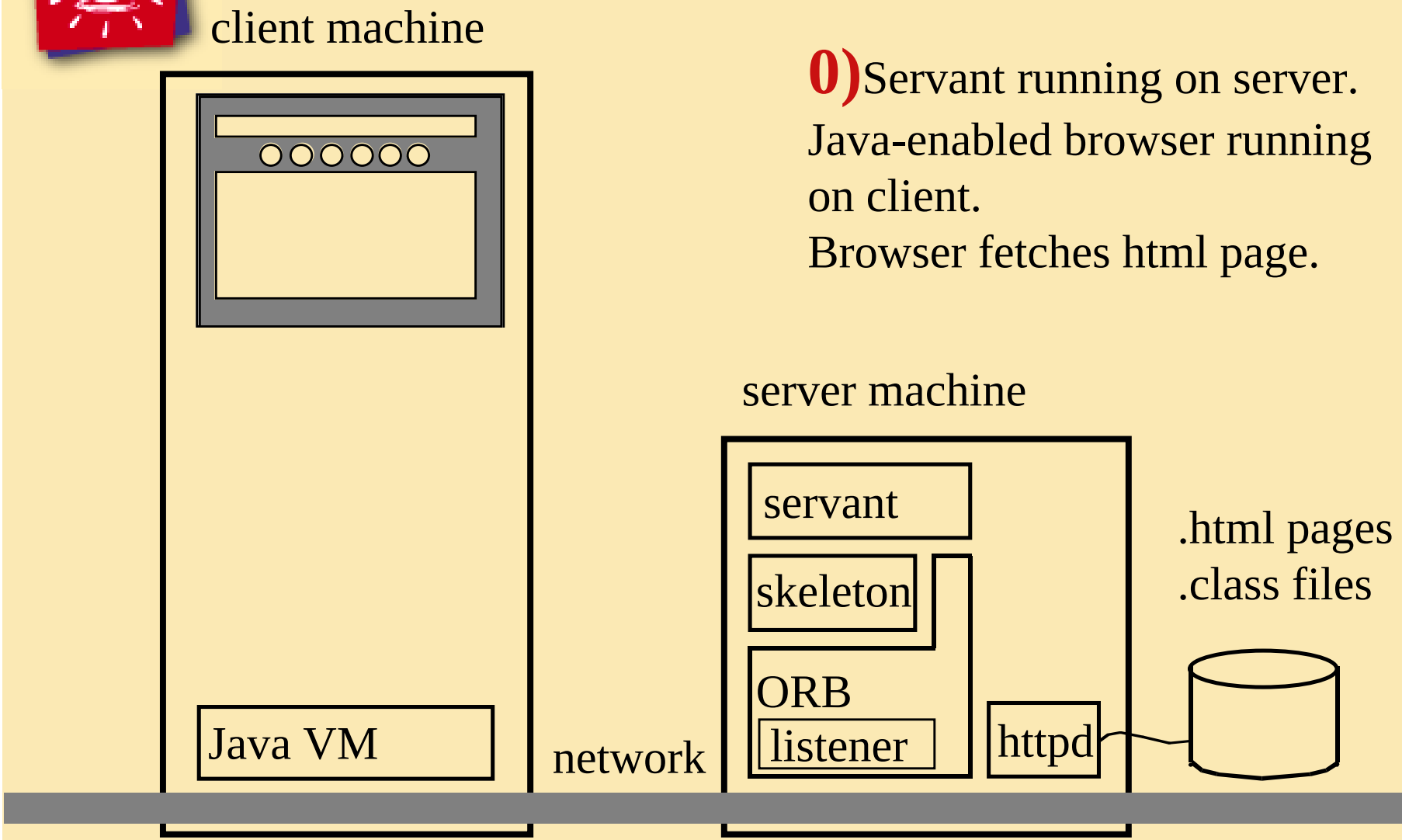
What Is an ORB?



What Is the Java ORB?

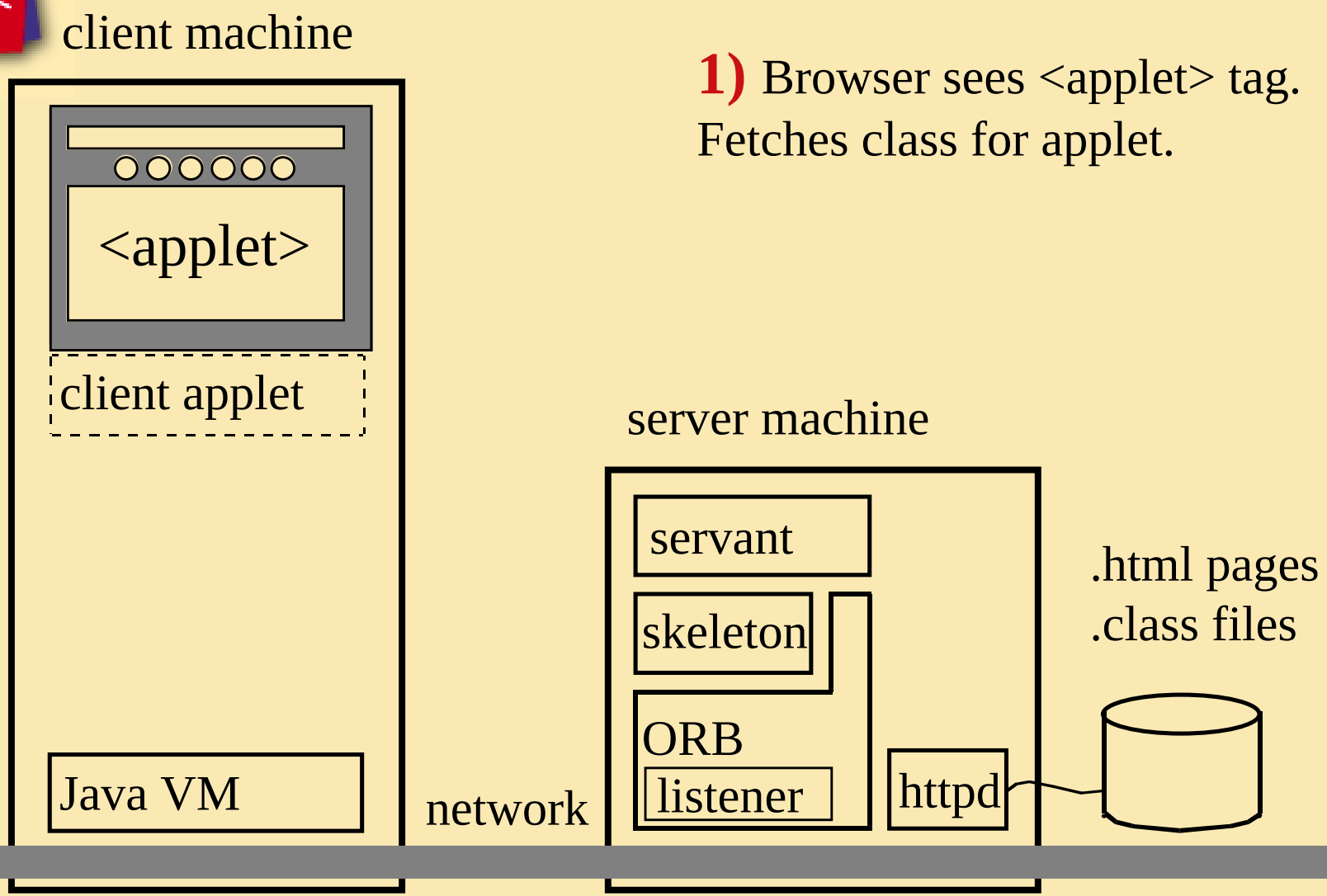
- API's the generated stubs use to talk to the ORB
- Additional functionality for ORB services
- Basic functionality mediating between stubs and the wire
- Java ORB defines API's for on-the-wire protocol modules
  - So third parties can plug in their own protocols
  - To support a multiplicity of protocols
  - To support protocol evolution

Zero-install Client



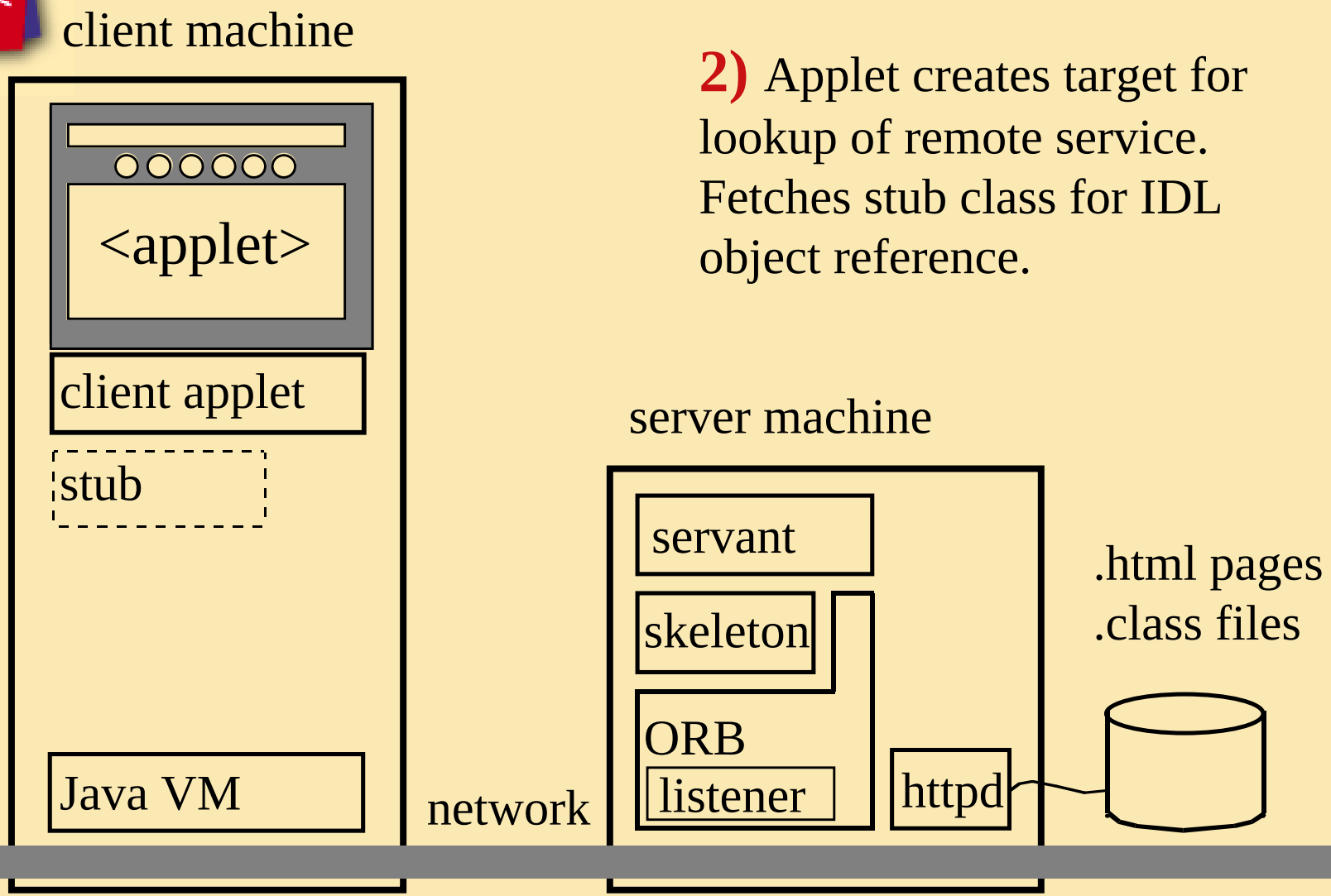
0) Servant running on server.
Java-enabled browser running on client.
Browser fetches html page.

Zero-install Client



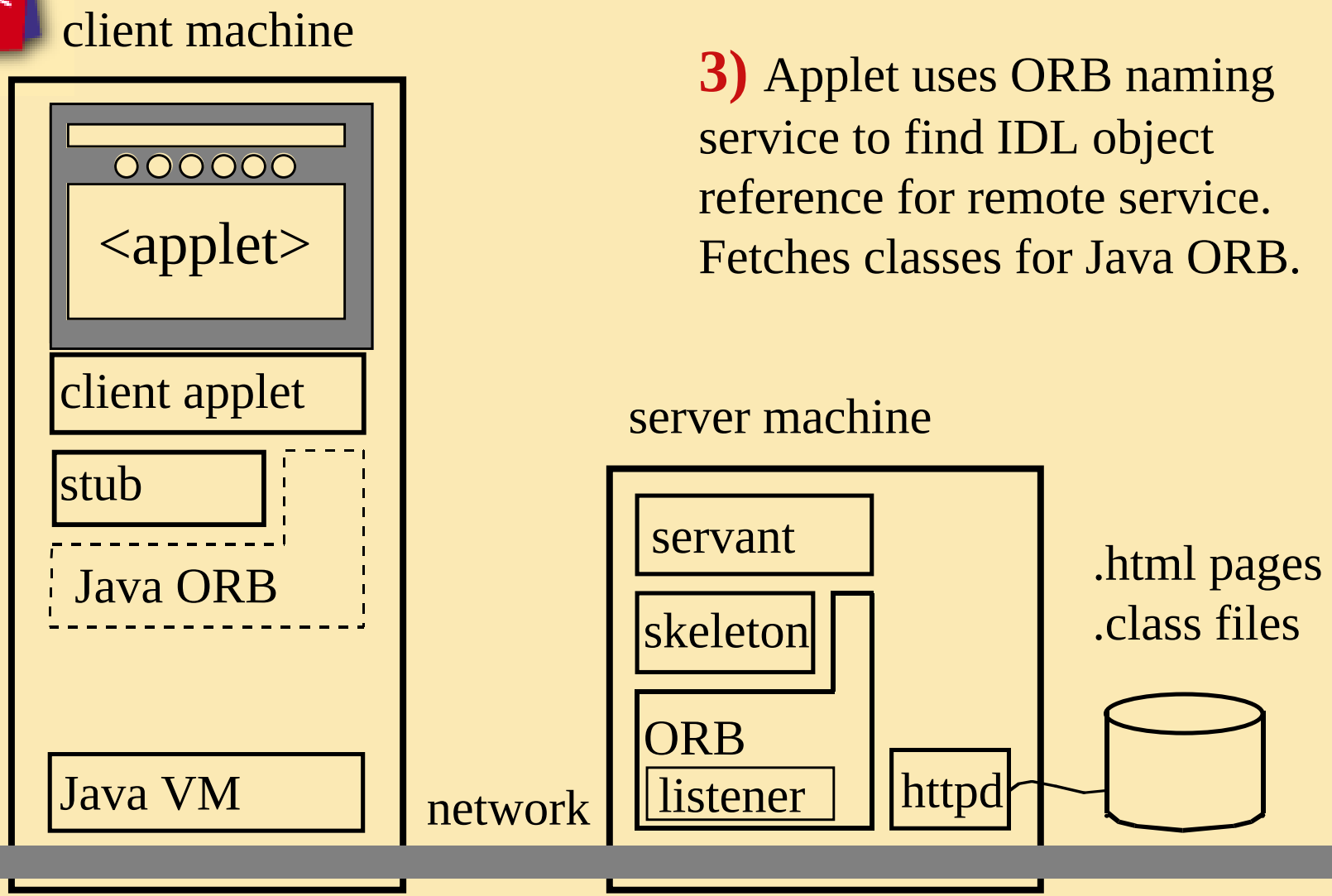
1) Browser sees <applet> tag.
Fetches class for applet.

Zero-install Client



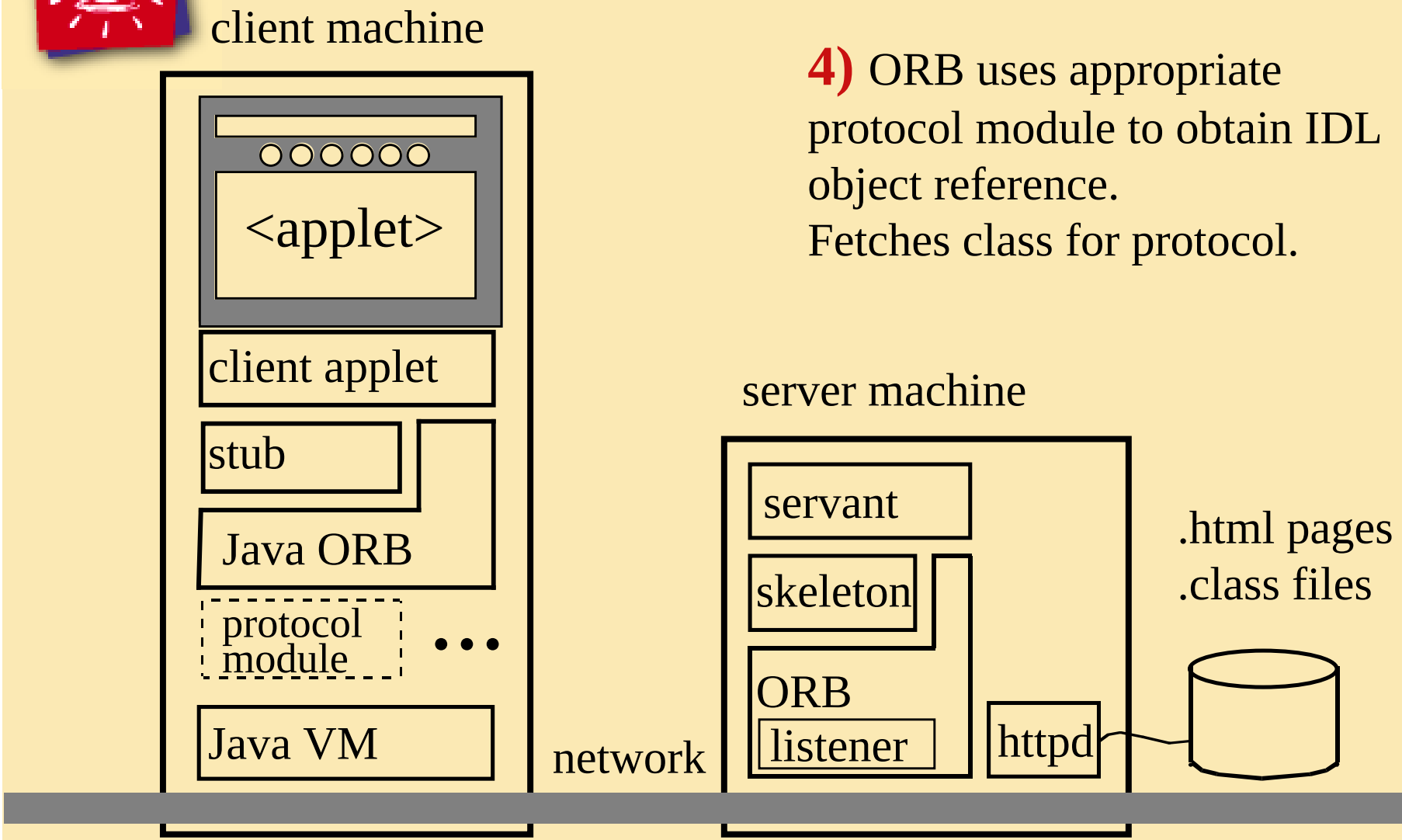
2) Applet creates target for lookup of remote service. Fetches stub class for IDL object reference.

Zero-install Client



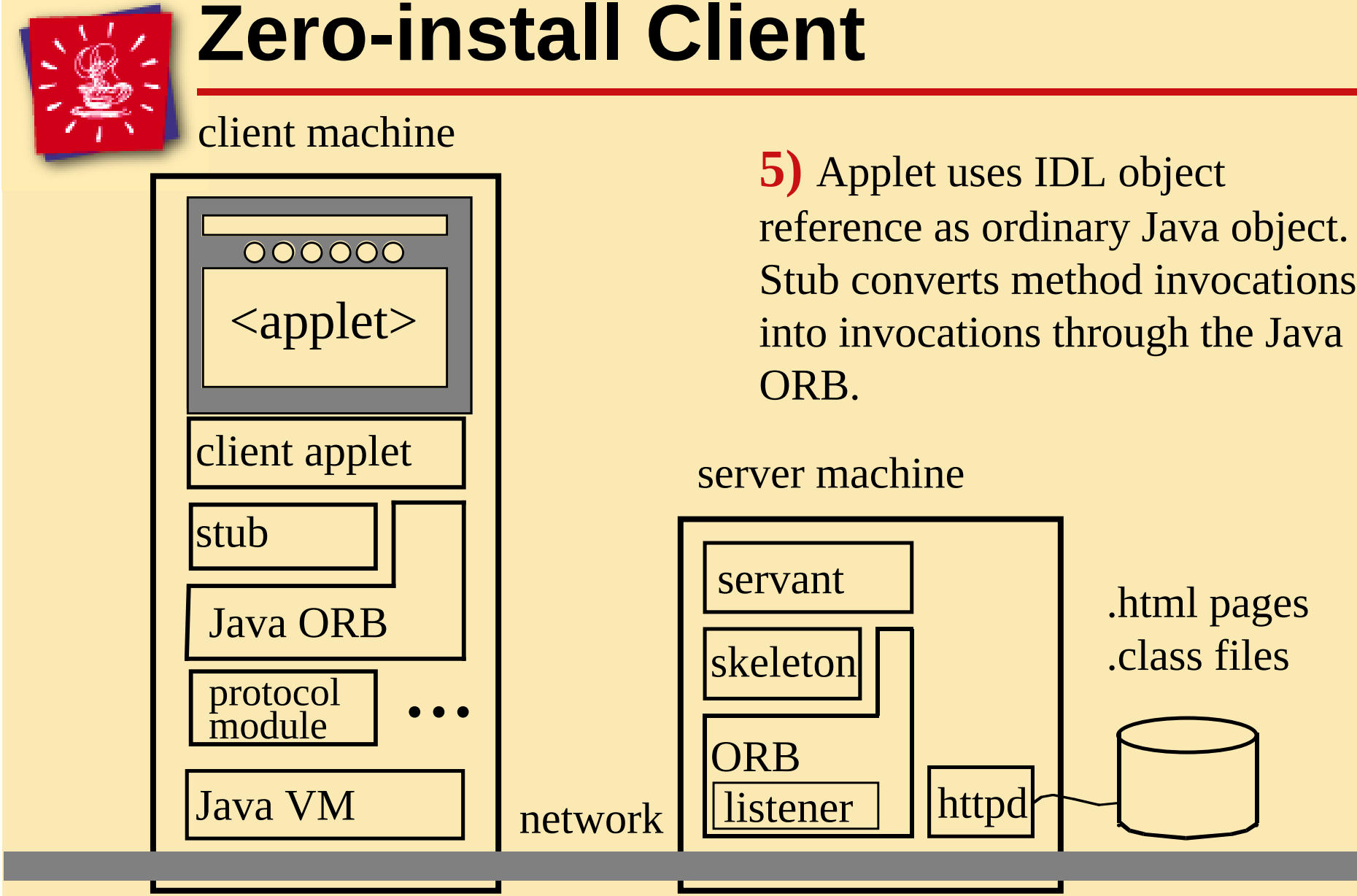
3) Applet uses ORB naming service to find IDL object reference for remote service. Fetches classes for Java ORB.

Zero-install Client



4) ORB uses appropriate protocol module to obtain IDL object reference.
Fetches class for protocol.

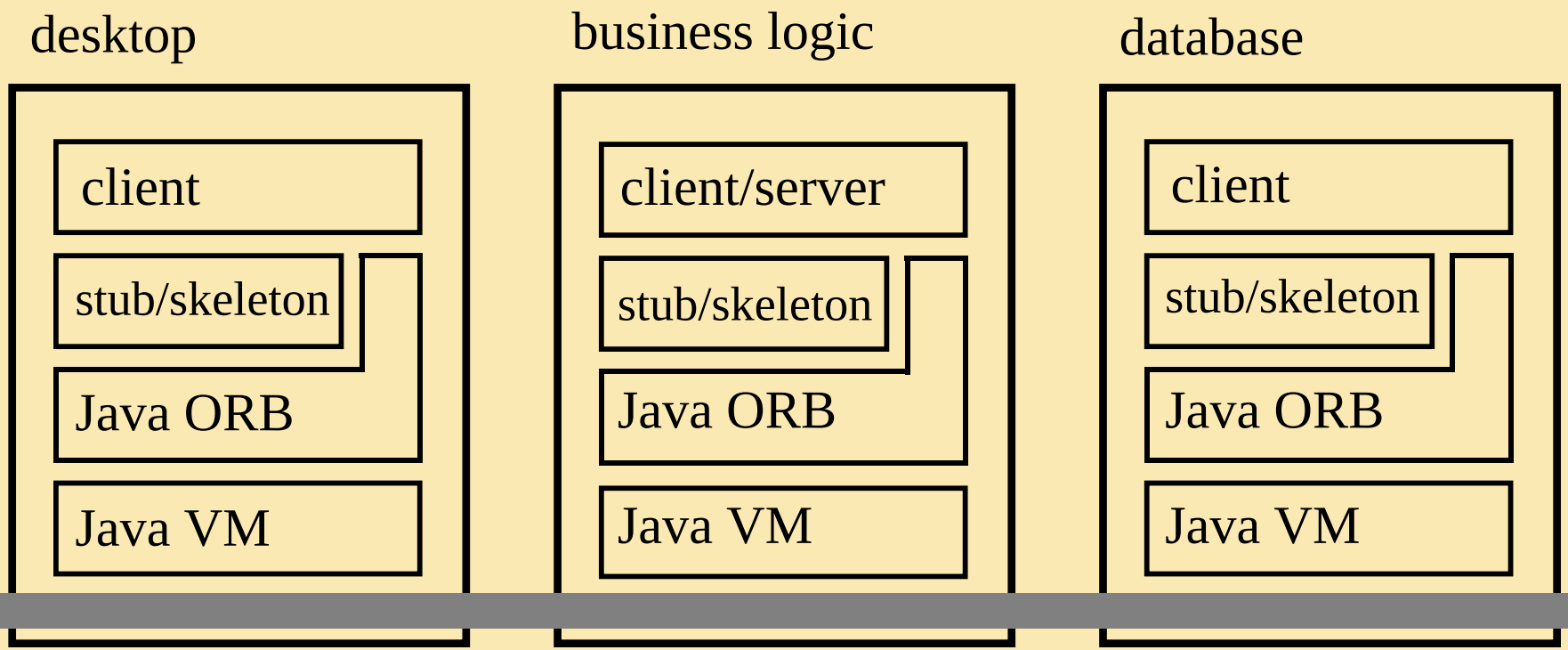
Zero-install Client



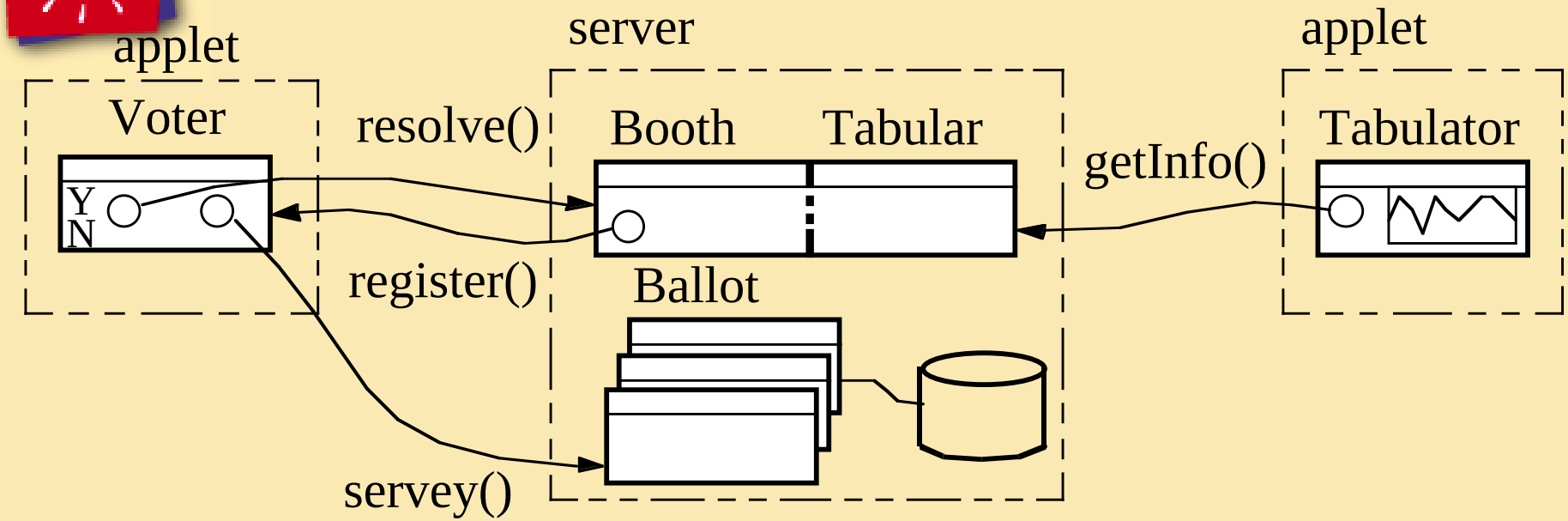
5) Applet uses IDL object reference as ordinary Java object. Stub converts method invocations into invocations through the Java ORB.

Java ORB Applications

- Using Java **applications** instead of **applets** allows more general communication among clients and servers



A Voting Application



Voter applet looks up Booth service
Voter creates call-back object and registers with Booth
Booth passes new Ballot to Voter for survey
Voter can make choice on Ballot or submit write-ins

Tabulator applet looks up Booth service as Tabular object
Tabulator periodically polls Booth for statistics

Status

- Currently in alpha release to the net
- Look under:
  - **http://splash.javasoft.com/pages/intro.html**
- Working with third parties for additional protocol modules for popular ORBs
- We will deliver Java ORB into JDK

Advantages of Java IDL and the Java ORB

- Standard, language-neutral, interface definition language
- Connects Java clients and servers to enterprise network services
- Clients and servers portable across architectures, operating systems
- Allows mixing of heterogeneous hardware platforms, operating environments, implementation languages, and ORB protocols
- Zero-install clients

Java Remote Method Invocation

Roger Riggs
Staff Engineer

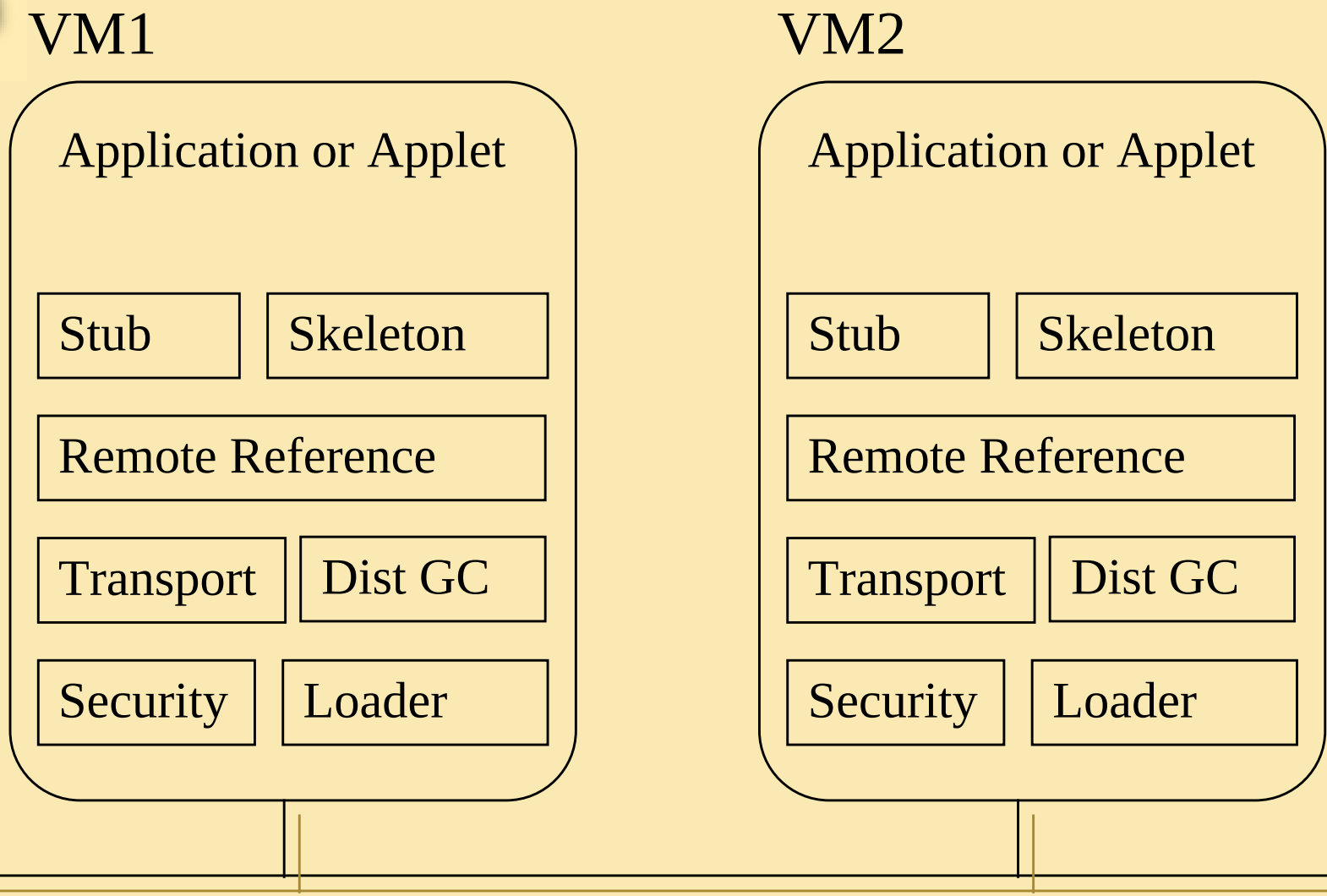
Java Remote Method Invocation

- Method invocation between objects in different Java Virtual Machines
- Pure Java interfaces
  No new interface definition language
- Pass and return any Java Object
- Dynamic loading of classes

Advantages

- Capitalizes on the Java Object Model
- Minimizes complexity
- Preserves safety of the Java runtime
- Recognizes distribution differences
  - Partial failure
  - Latency
  - No global knowledge

RMI Architecture



System Features

- Garbage collection of remote objects, distributed reference-counting
- Security manager and class loader
- Designed to support :
  - Replication
  - Persistent References
  - Activation
  - Fully multi-threaded

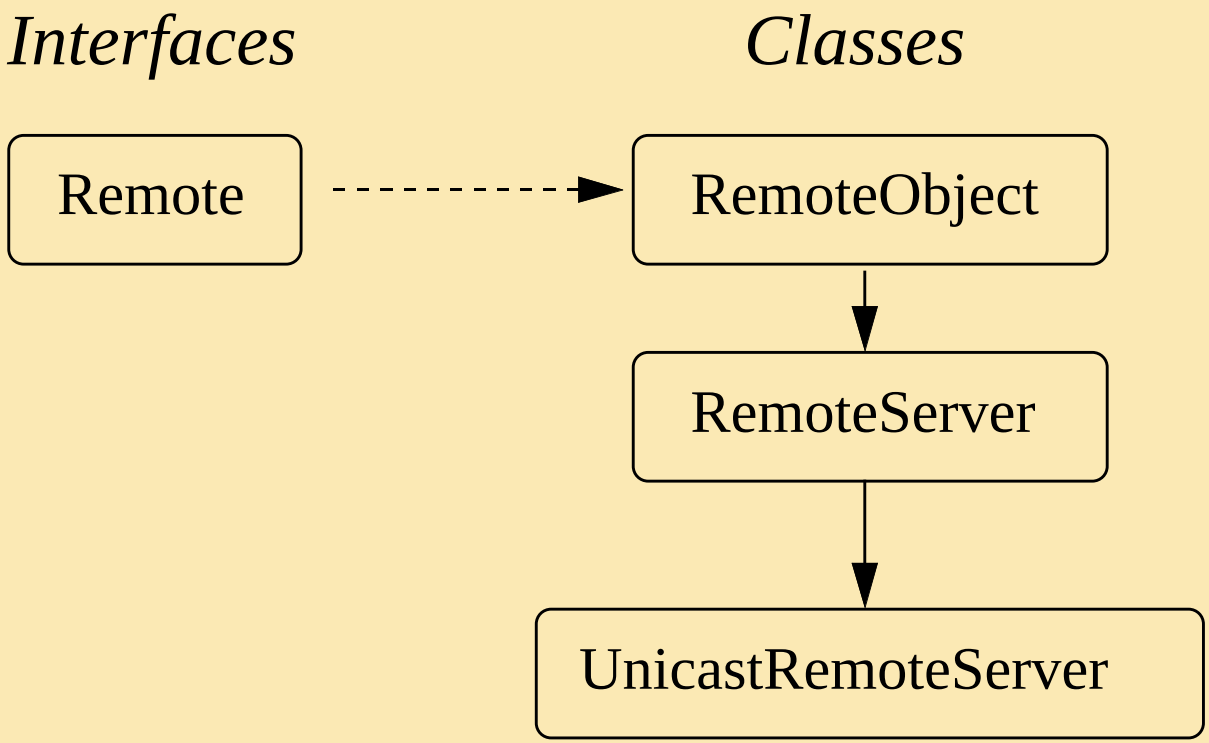
Definitions

- Remote object
  - Object whose methods can be invoked from another Java VM
- Remote interface
  - Java interface that declares the methods of the remote object

Remote Objects are Java Objects

- Referenced via Remote interfaces
- All types as arguments and returns
- Stub objects have the same remote interfaces as the remote object.
  - Casting to other Remote interfaces
  - Instance of to check type of interfaces
- Exceptions report communication failures

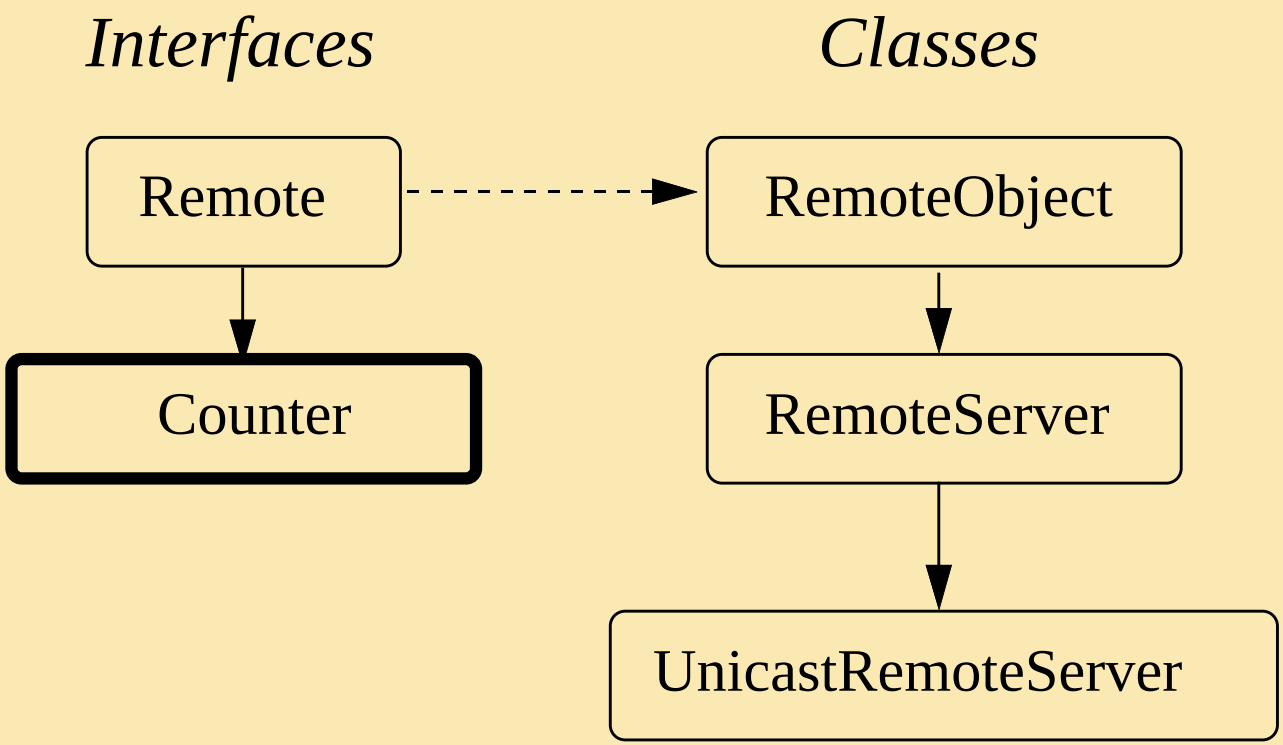
RMI Interfaces and Classes



The Remote Interface

```
package java.rmi;
interface Remote {}
```

RMI Interfaces and Classes



Remote Interface Example

```
public interface Counter
                extends java.rmi.Remote
  {
      public void deposit(float amt)
          throws java.rmi.RemoteException;
      public void withdraw(float amt)
          throws OverdrawnException,
                 java.rmi.RemoteException;
      public float balance()
          throws java.rmi.RemoteException;
  }
```

Argument and Return Values

- Any Java language type
- Remote objects are replaced by stubs
  - Stub has embedded remote reference
- Objects are passed by copy using Java Object Serialization
- Classes loaded on demand
  - Stubs, arguments, and return values

Object Serialization

- Objects and graphs of objects
- Write to and read from streams
- Preserves cycles
- Per class methods only to customize
- Objects can refuse to be serialized
  - Mark Fields as transient
  - Throw Exception
  - Default is NOT to Serialize

Serialization Example

```
// Write today's date to a file
   OutputStream out = new FileOutputStream("t");
   ObjectOutput w = ObjectOutputStream(out);
   w.writeObject("Today");
   w.writeObject(new Date());
   w.flush();
// Read string and date from file
   InputStream in = new FileInputStream("t");
   ObjectInput r = new ObjectInputStream(in);
   String today = (String)r.readObject();
   Date date = (Date)r.readObject();
```

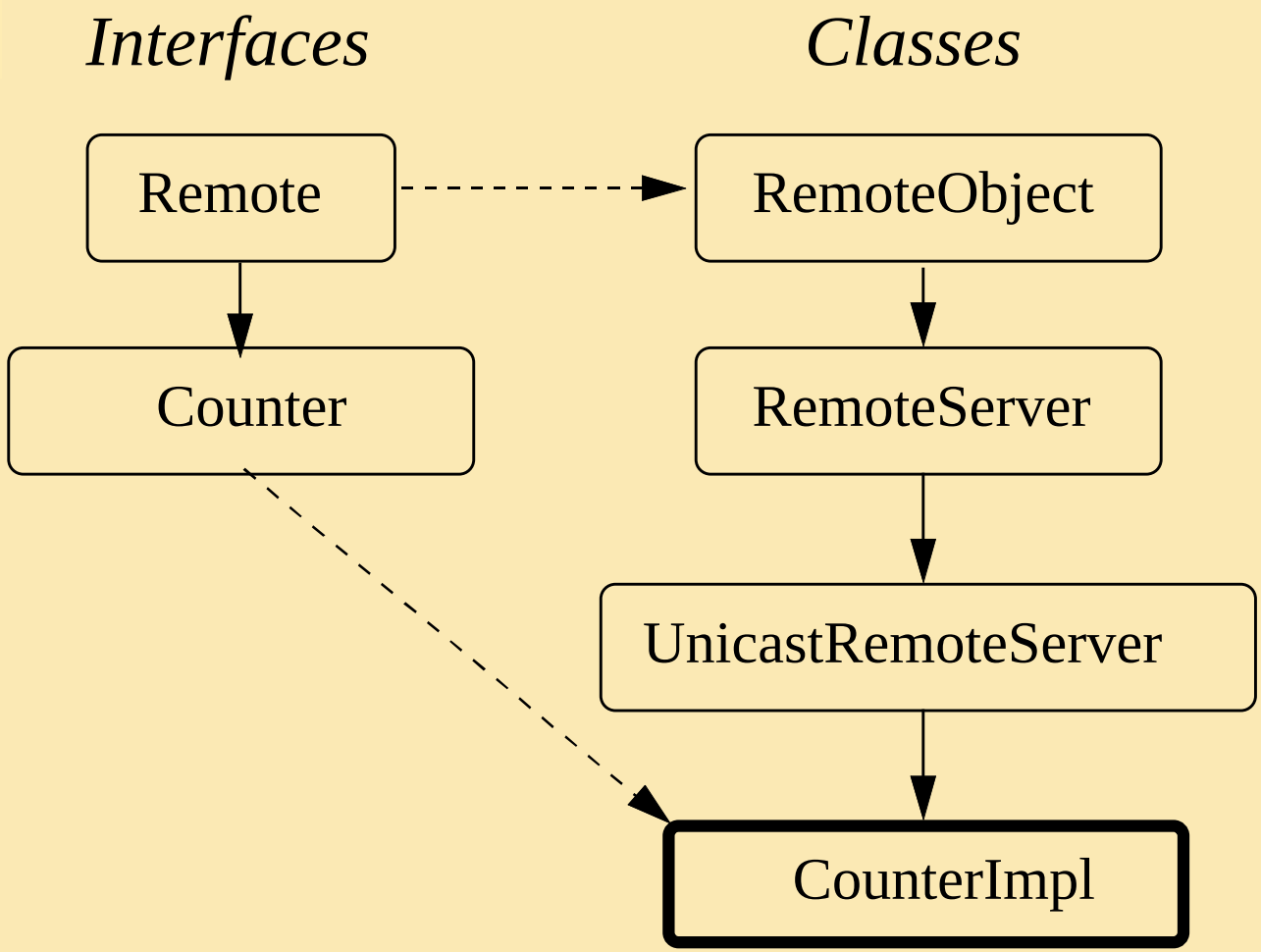
Class RemoteException

- No distributed system can mask communication failures:
  - Each remote method must declare RemoteException
  - Thrown when an method invocation fails
- Failures must be handled

Classes

- Interface Remote
  - Identifies interface to remote objects
- Class RemoteObject
  - Object methods specialized for remote
- Class RemoteServer
  - Methods create and export remote objects
- Class UnicastRemoteServer
  - Non-replicated remote objects

RMI Interfaces and Classes



Implementing a Remote Object

```
import java.rmi.*;
import java.rmi.server.*;
public class CounterImpl
    extends UnicastRemoteServer
    implements Counter
{
    private value = 0;
    public CounterImpl()
          throws RemoteException {...};
    public synchronized void increment(int amt)
          throws RemoteException {
        return ++value;
    };
    ...
}
```

Locating Remote Objects

- Uniform Resource Locator (URL)
- Defining the name

```
Counter acct = new CounterImpl();
String name = “rmi://java.Sun.COM/account”;
java.rmi.Naming.bind(name, acct);
```

- Lookup by name

```
Counter acct =
 (Counter)java.rmi.Naming.lookup(name);
acct.withdraw(1000000.00);
```

Dynamic Stub Loading

- Classes dynamically loaded
  stubs, arguments and return classes
  - No type truncation
- Classes subject to a security manager
- Must protect against stub misbehavior

  Not always complete
  - Applications configurable to disable
    stub loading

RMI Security

- Leverages Java Security mechanisms
- For Applets
  - AppletClassLoader loads and AppletSecurityManager protects
- For Applications
  - StubClassLoader loads and StubSecurityManager protects
- Can define own security manager

Status

- Development team
- Alpha release available now in the Developers Corner at
  - http://java.sun.com/devcorner.html
- Customer ship available late summer or early fall

Summary

- Pure Java Object Model
- Pass arbitrary objects and graphs
- Dynamic stub to class loading
- Distribution models
  - Applet and Application
  - Application to Application