36 ClassLoader

- How are classes loaded into the Java Virtual Machine (JVM)?
  - from the local file system (**`CLASSPATH`**).
  - by an instance of **`ClassLoader`**
- ... and when? - When they are needed the first time.

```
class Test {
    public String toString() {
        Hello hello = new Hello();
    }
}
```

```
ClassLoader cl = new ...;
Class c = cl.loadClass("Test");
Object t=c.newInstance();
t.toString();
```

(1) class Test is loaded by classloader **`cl`**

(2) toString is executed and class **`Hello`** is needed

(3) the classloader of **`Test`** (= **`cl`**) is asked to load **`Hello`**

Reproduktion jeder Art oder Verwendung dieser Unterlage bedarf der Zustimmung des Autors.

36.1 ClassLoader

- A classloader creates a name space.
  - There can exist classes with the same name in one JVM, provided that they are loaded by different classloader instances.
- **`java.lang.ClassLoader`**
  - can create a class from an array of bytes that conforms to the class file format (method **`defineClass()`**)
  - if this class uses other classes your class loader is asked to load them (method **`loadClass(String name)`**)
  - look for previously loaded classes with **`Class findLoadedClass(String name)`**
- multiple **`ClassLoader`**s can be running in one JVM

Reproduktion jeder Art oder Verwendung dieser Unterlage bedarf der Zustimmung des Autors.

36.2 Example ClassLoader

```
import java.io.*;

public class SimpleClassLoader extends ClassLoader {

    public synchronized Class loadClass(String name, boolean resolve) {

        Class c = findLoadedClass(name);
        if (c != null) return c;

        try {
            c = findSystemClass(name);
            if (c != null) return c;
        } catch(ClassNotFoundException e) {}

        try {
            RandomAccessFile file = new RandomAccessFile("test/" +
                                              name + ".class", "r");
            byte data[] = new byte[(int)file.length()];
            file.readFully(data);
            c = defineClass(name, data, 0, data.length);
        } catch(IOException e) {}

        if (resolve)
            resolveClass(c);

        return c;
    }
}
```


Reproduktion jeder Art oder Verwendung dieser Unterlage bedarf der Zustimmung des Autors.

36.3 Example Appletviewer

- **`appletviewer`** uses a special class loader (**`sun.applet.AppletClassLoader`**)
- What happens when the **`appetviewer`** is started?

1. String parameter is converted to **`URL`** object.
2. **`URL`** stream is parsed looking for the **`<applet>`** tag.
3. **`AppletClassLoader`** with URL specified in **`codebase`** is created.
4. Class specified in **`code`** parameter is loaded by the **`AppletClassLoader`**.

- **`AppletClassLoader`** is initialized with URL:

```
    AppletClassLoader(URL baseURL)
```

- the **`loadClass(String name)`** method creates an URL from **`baseURL`** and the class name **`name`** and calls **`loadClass(URL url)`**
- **`loadClass(URL url)`** gets the URL stream, reads the bytes of the class and calls **`defineClass()`**

Reproduktion jeder Art oder Verwendung dieser Unterlage bedarf der Zustimmung des Autors.

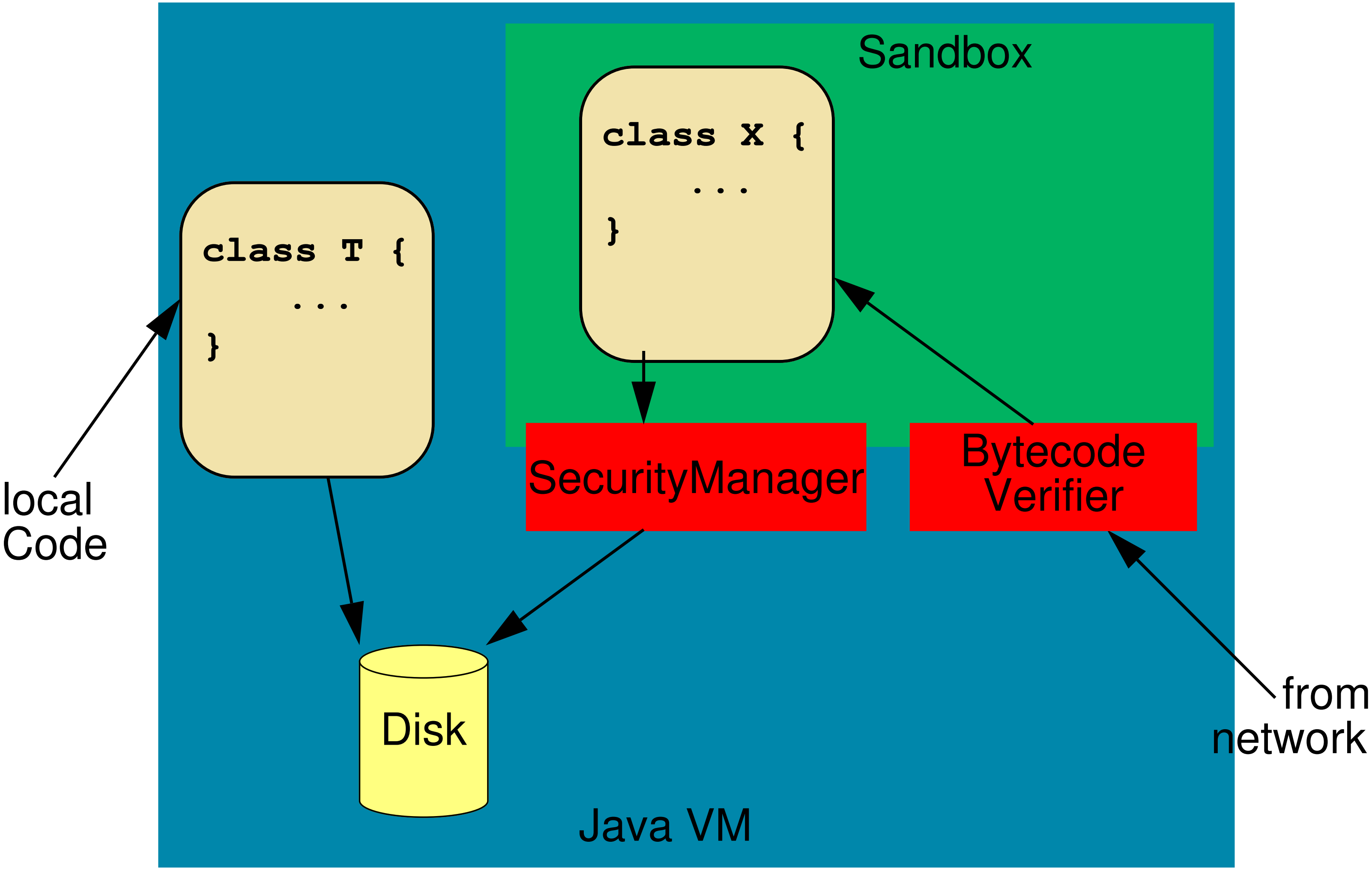
37 Java Security

- Sandboxing
- Bytecode Verifier
- Security Manager
- Implementation of the Java libraries

Reproduktion jeder Art oder Verwendung dieser Unterlage bedarf der Zustimmung des Autors.

37.1 Sandboxing

- Classes loaded by ClassLoader execute within a *Sandbox*



Reproduktion jeder Art oder Verwendung dieser Unterlage bedarf der Zustimmung des Autors.

37.2 SecurityManager

- Checks if a class is allowed to perform an operation
- **`System.getSecurityManager()`** returns the global security manager
- checks must e done by the protected class itself

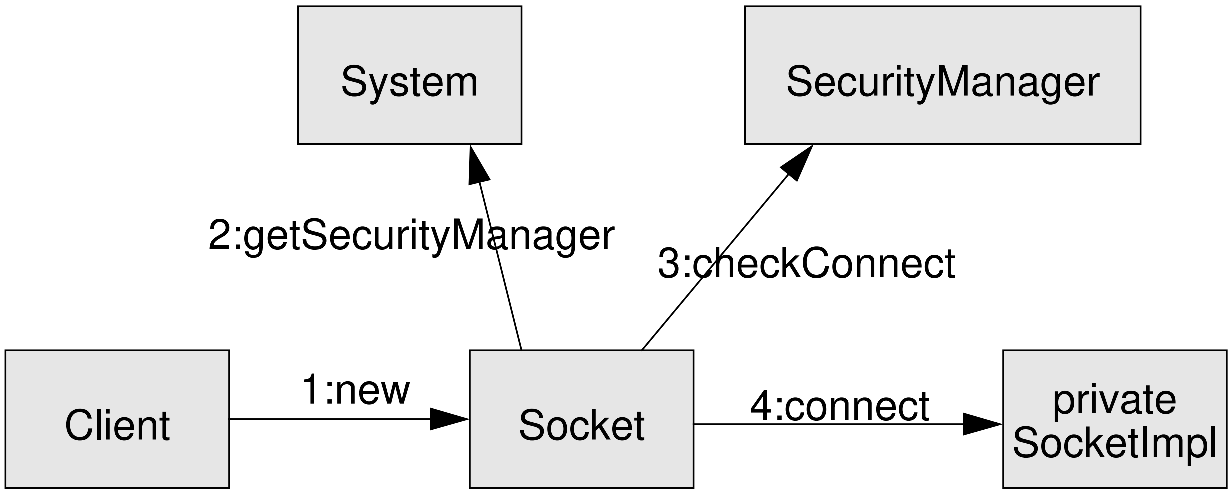
```
SecurityManager security = System.getSecurityManager();
if (security != null) {
    security.checkConnect(address.getHostAddress(), port);
}
```

- **`System.setSecurityManager()`** installs a global security manager
  - ◆ this method can be called *only once*!

Reproduktion jeder Art oder Verwendung dieser Unterlage bedarf der Zustimmung des Autors.

37.3 SecurityManager und Sockets

- Example network communication: creating a new Socket



Reproduktion jeder Art oder Verwendung dieser Unterlage bedarf der Zustimmung des Autors.

37.4 Example SecurityManager

```
public class SimpleSecurityManager extends SecurityManager
{
    public void checkRead(String s) {
        if (...) {
            throw new SecurityException("checkread");
        }
    }
}
```


Reproduktion jeder Art oder Verwendung dieser Unterlage bedarf der Zustimmung des Autors.

37.4 What is Protected by a Security Manager?

- Access to the local file system
- Access to operating system
  - executing programs
  - reading system information
- Access to the network
- Thread manipulation
- Creation of factory objects (socket implementation)
- JVM: Linking native code, Exiting the interpreter, creating classloaders,
- Creating windows
- ...

Reproduktion jeder Art oder Verwendung dieser Unterlage bedarf der Zustimmung des Autors.

37.5 AppletSecurityManager

Security Checks	AppletSecurityManager
checkCreateClassLoader	not allowed
checkConnect	allowed, if URL of the classes ClassLoader contains the target host
checkExit	not allowed
checkExec	not allowed

Reproduktion jeder Art oder Verwendung dieser Unterlage bedarf der Zustimmung des Autors.

38 Remote Method Invocation (RMI)

- Abstraction in a distributed system

Socket Communication	no abstraction
↓	
Remote Procedure Call	procedural abstraction
↓	
Remote Method Invocation	object abstraction

- Remote Method Invocation: calling methods at objects on other hosts
- Transparent object references to remote objects

© 1997- 2000 Michael Golm

Reproduktion jeder Art oder Verwendung dieser Unterlage bedarf der Zustimmung des Autors.

38.1 RMI Introduction

- *Remote Object*: object that can be used from a different JVM
- *Remote Interface*: Declares Methods of the remote object.
- Remote Interface must inherit **`java.rmi.Remote`** (marker interface)
- Accessing remote objects only via remote interfaces
- remote class must implement remote interface
- remote objects can implement multiple remote interfaces
- remote objects can use other remote objects
- Parameter passing:
  - by value: default (non-remote objects) -- Copy to remote host
  - by reference: if parameter implements **`java.rmi.Remote`**

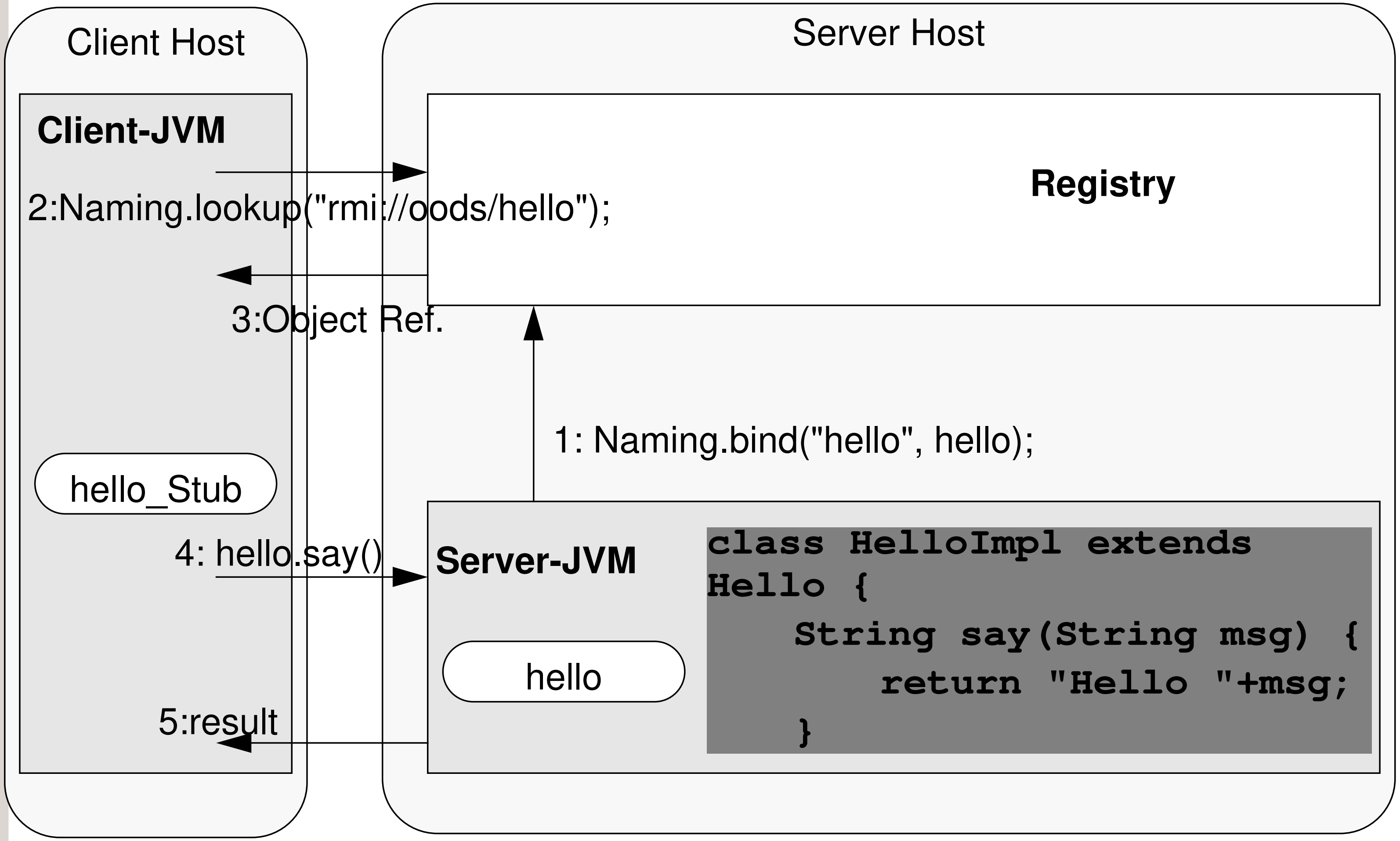
Reproduktion jeder Art oder Verwendung dieser Unterlage bedarf der Zustimmung des Autors.

38.2 Local vs. Remote Method Call

- every invocation can throw a **`RemoteException`**,
  - client does not know if mehod was executed completely, partly, or not at all
- remote objects can only be used via interfaces
- local objects are passed by value

Reproduktion jeder Art oder Verwendung dieser Unterlage bedarf der Zustimmung des Autors.

38.3 RMI Operation



Reproduktion jeder Art oder Verwendung dieser Unterlage bedarf der Zustimmung des Autors.

38.4 Registry (1)

- The registry maintains the mapping between object names and references.
- Accessed using the class **`java.rmi.Naming`**
  - ◆ **`void Naming.bind(String name, Remote obj)`**
    registers object with a specific name,
    exception is thrown if object already registered
  - ◆ **`void Naming.rebind(String name, Remote obj)`**
    registers object under name,
    if object already registered, old binding is destroyed
  - ◆ **`Remote Naming.lookup(String name)`**
    finds reference for a given object name
  - ◆ **`void Naming.unbind(String name)`**
    destroys binding between object and its name
- Binding only possible on registry at same host!

Reproduktion jeder Art oder Verwendung dieser Unterlage bedarf der Zustimmung des Autors.

38.5 Registry (2)

- Using a specific port:
  - Registry waits for TCP/IP connections at port 1099
  - other port can be specified as parameter, e.g. 10412:

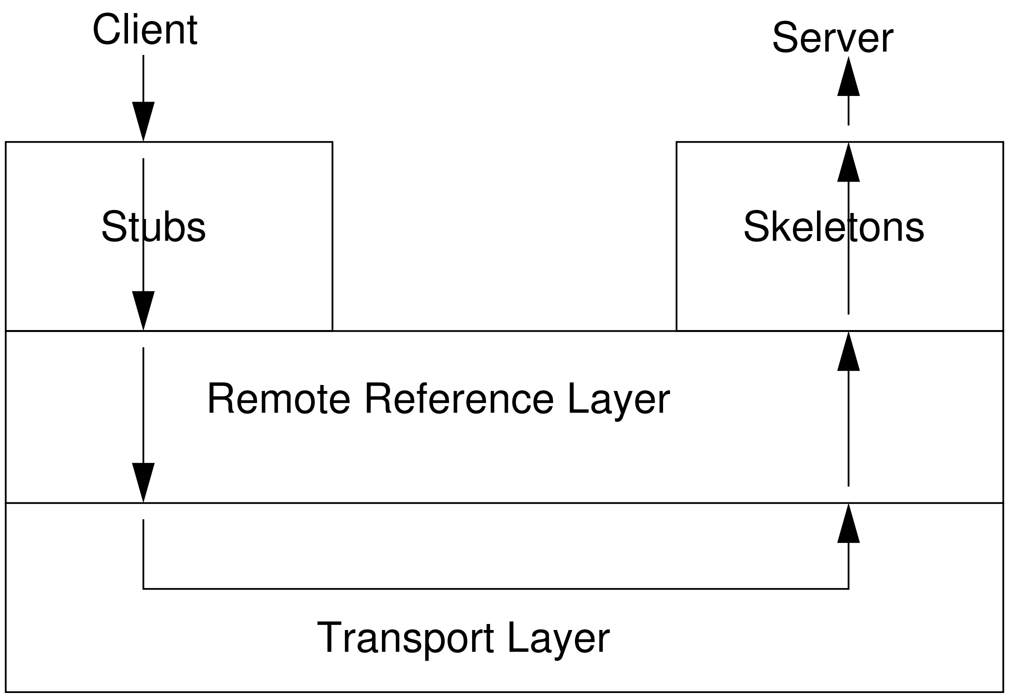
    `rmiregistry 10412`
  - if a registry at a specific port should be used, the URL passed to **`bind/rebind/unbind`** must contain this port:

    `Naming.rebind("//oobp.informatik:10412/hello", hello)`
  - ... and **`lookup`**

    `Naming.lookup("rmi://oobp.informatik:10412/hello")`

Reproduktion jeder Art oder Verwendung dieser Unterlage bedarf der Zustimmung des Autors.

38.6 System Architecture



Reproduktion jeder Art oder Verwendung dieser Unterlage bedarf der Zustimmung des Autors.

38.7 Stubs and Skeletons

- Stub (Client Side) - implements the remote interface

1. gets `ObjectOutputStream` from RemoteReference Layer
2. writes the parameters to this stream
3. message to remote reference layer to invoke method
4. gets `ObjectInputStream` from RemoteReference Layer
5. reads the return object from this stream
6. returns the return object to the caller

- Skeleton (Server Side)

1. gets `ObjectInputStream` from RemoteReference Layer
2. reads parameters from this stream
3. invokes the method at the implementation object
4. gets `ObjectOutputStream` from RemoteReference Layer
5. writes the return object to this stream

© 1997- 2000 Michael Golm

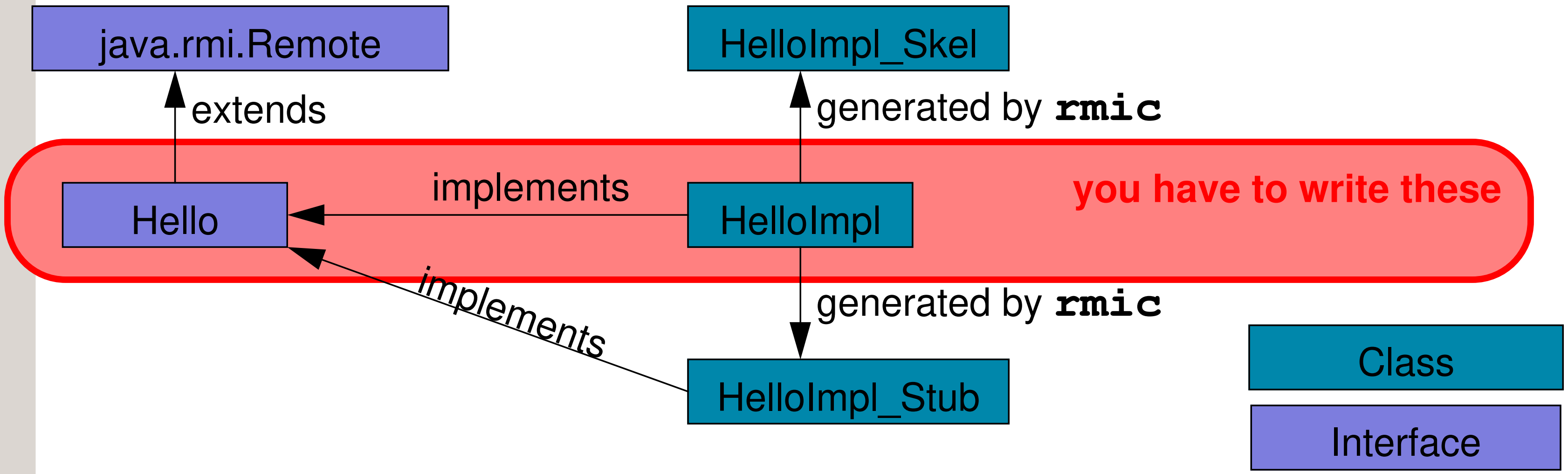
Reproduktion jeder Art oder Verwendung dieser Unterlage bedarf der Zustimmung des Autors.

38.8 Remote Reference Layer / Transport Layer

- Remote Reference Layer
  - ◆ responsible for garbage collection of remote objects
  - ◆ implements invocation semantics, e.g.
    - unicast point-to-point
    - invocation at replicated object
    - persistent reference to remote object
    - strategies to reconnect after connection termination
- Transport Layer
  - ◆ responsible for data transport between hosts
  - ◆ current implementation uses TCP/IP sockets
  - ◆ different transport mechanisms possible

Reproduktion jeder Art oder Verwendung dieser Unterlage bedarf der Zustimmung des Autors.

38.9 Interface and Implementation



- Programmer writes the remote interface **`Hello`** and the implementation **`HelloImpl`**.
- The stub **`HelloImpl_Stub`** and the skeleton **`HelloImpl_Skel`** are generated by **`rmic`** from the implementation class **`HelloImpl`**.
- **`RMIClassLoader`** loads classes of parameters and return values.

Reproduktion jeder Art oder Verwendung dieser Unterlage bedarf der Zustimmung des Autors.

38.10 Example

0

1. write the remote interface
2. write the server
3. register the remote object
4. write the client
5. start the system

© 1997- 2000 Michael Golm

Reproduktion jeder Art oder Verwendung dieser Unterlage bedarf der Zustimmung des Autors.

38.10.1 Remote Interface

- every remote object must implement an interface that contains all remotely callable methods
- interface must inherit from **java.rmi.Remote**
- all methods must throw a **java.rmi.RemoteException**
- all parameters and return values must be serializable (implement **java.io.Serializable**) or remote objects
- Exampe:

```
import java.rmi.*; import java.io.*;
public interface Bank extends Remote {
    void deposit(Money amount, Account account) throws RemoteException;
}

public class Money implements Serializable {
    private float value;
    public Money(float value) { this.value = value; }
}

public interface Account extends Remote { ... }
```


Reproduktion jeder Art oder Verwendung dieser Unterlage bedarf der Zustimmung des Autors.

38.10.2 Server

- The server must implement the remote interface.
- Methods has not to throw **RemoteException**.
- Two ways to create a remote object:
  - ◆ Server subclasses **rmi.server.UnicastRemoteObject**

```
public class BankImpl extends UnicastRemoteObject
                                        implements Bank {
    public void deposit(Money amount, Account account){
        account.deposit(amount);
    }
}
BankImpl bank = new BankImpl();
```

  - ◆ using **exportObject**

```
public class BankImpl implements Bank {... }

Remote bank = UnicastRemoteObject.exportObject(new BankImpl())
```


© 1997- 2000 Michael Golm

Reproduktion jeder Art oder Verwendung dieser Unterlage bedarf der Zustimmung des Autors.

Initialize the Server

- Register the remote object (**`bind`** or **`rebind`**).

```
Naming.bind("rmi://oods/bank", bank);
```

protocol hostname object name

- Install a security manager.
  - RMI refuses to load classes from the network without a security manager.
  - Server JVM sets a security manager with

```
System.setSecurityManager(new RMISecurityManager());
```

  - **`RMISecurityManager`** similar to **`AppletSecurityManager`**

© 1997- 2000 Michael Golm

Reproduktion jeder Art oder Verwendung dieser Unterlage bedarf der Zustimmung des Autors.

38.10.4 Client

- Get a remote reference using **`lookup`**.
- Example:

```
Bank bank = (Bank)Naming.lookup("rmi://oods/bank");
Account account = new AccountImpl();
bank.deposit(new Money(10), account));
```


© 1997- 2000 Michael Golm

Reproduktion jeder Art oder Verwendung dieser Unterlage bedarf der Zustimmung des Autors.

38.10.5 Start the System

1. Set CLASSPATH (for **`rmic`** and **`java`** (server))

```
setenv CLASSPATH /proj/i4oods/www/pub/rmi
```

2. Create stubs and skeletons

```
rmic -d ${CLASSPATH} bank.BankImp
```

3. Start the registry on the server host

```
rmiregistry 10412&
```

4. Start the server object

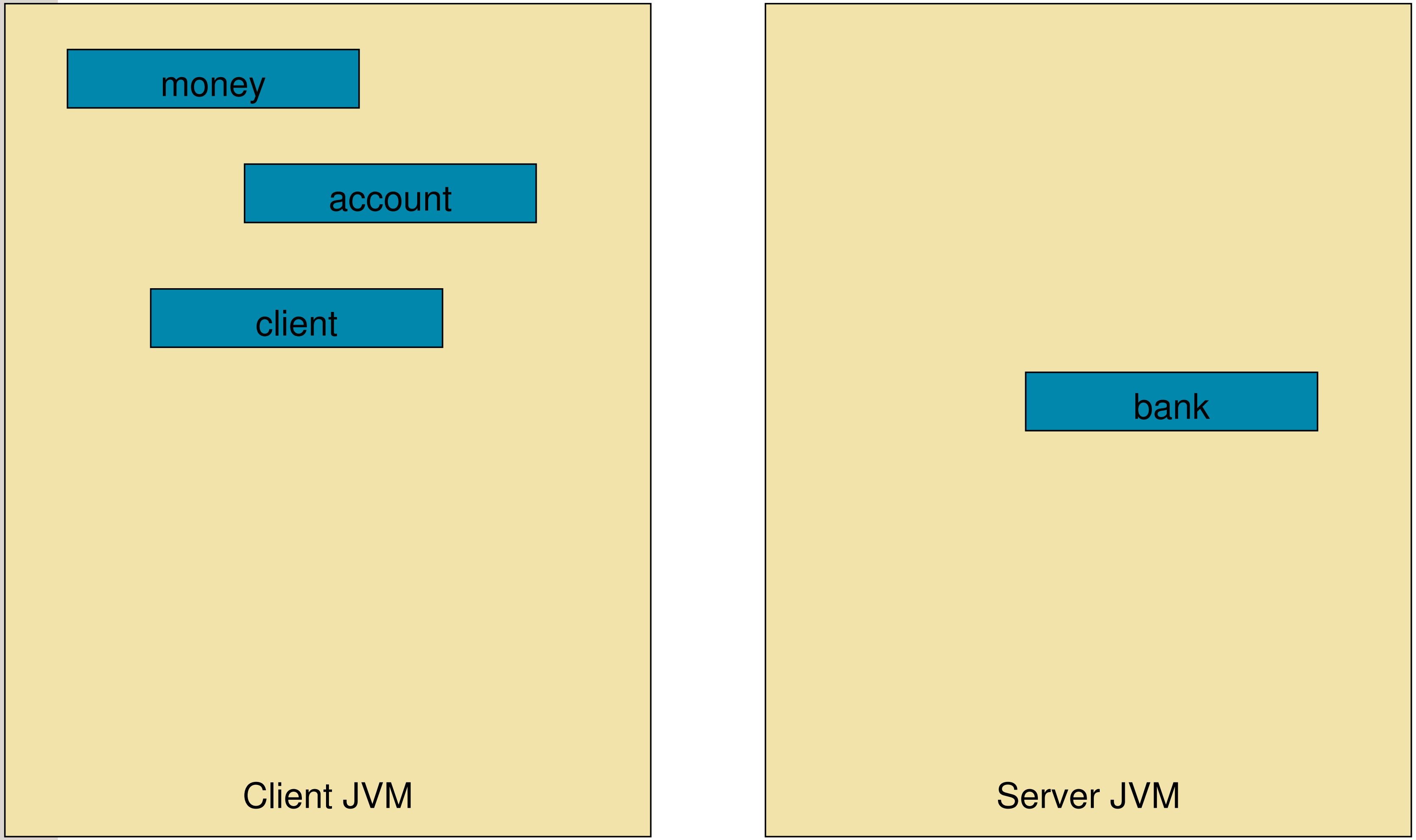
```
java -Djava.rmi.server.codebase=http://oods/pub/rmi/ bank.BankImpl
```

5. Use the applet:

```
appletviewer http://oods/pub/rmi/bank.html
```

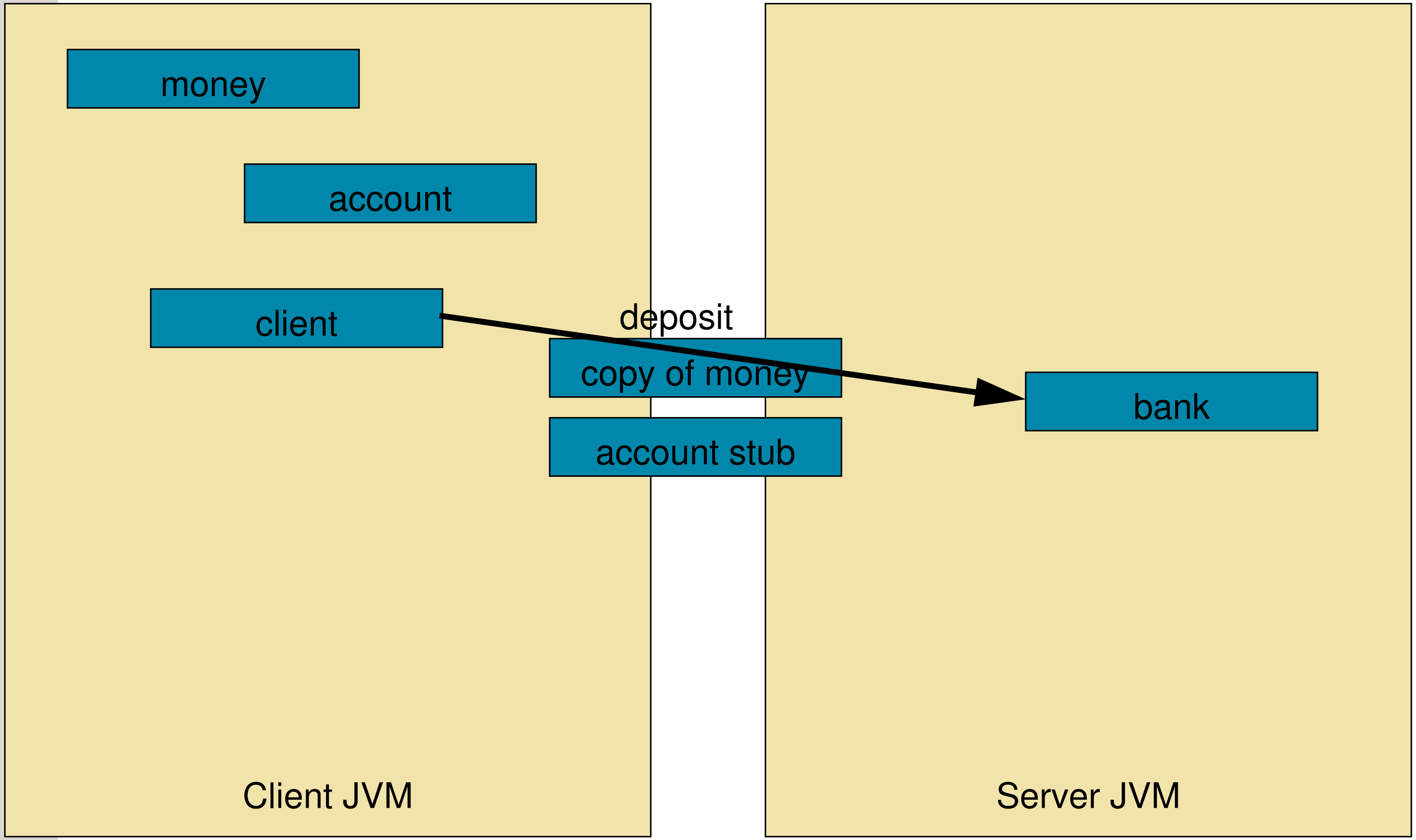

Reproduktion jeder Art oder Verwendung dieser Unterlage bedarf der Zustimmung des Autors.

38.10.6 Interaction during the remote call



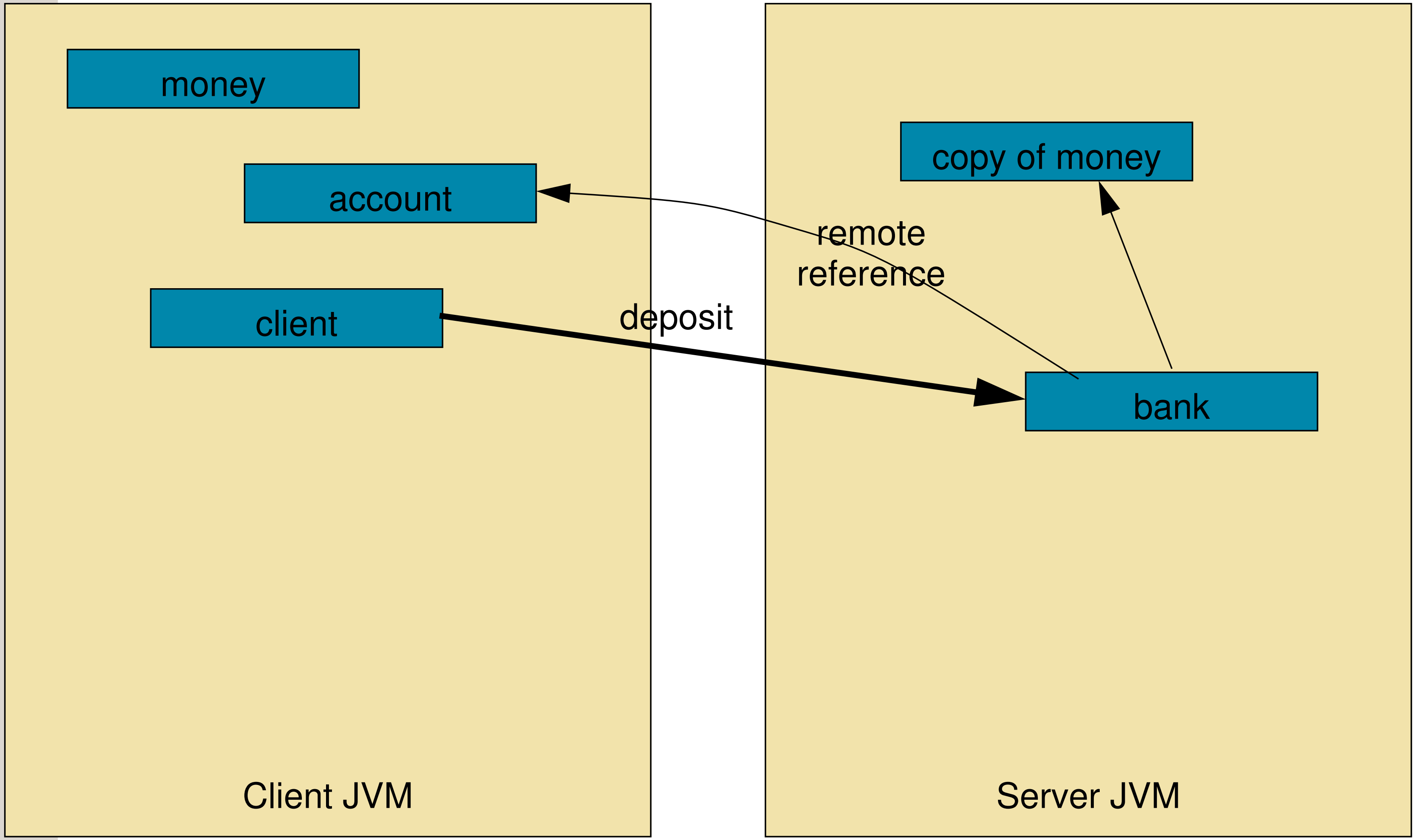
Reproduktion jeder Art oder Verwendung dieser Unterlage bedarf der Zustimmung des Autors.

38.10.7 Interaction during the remote call



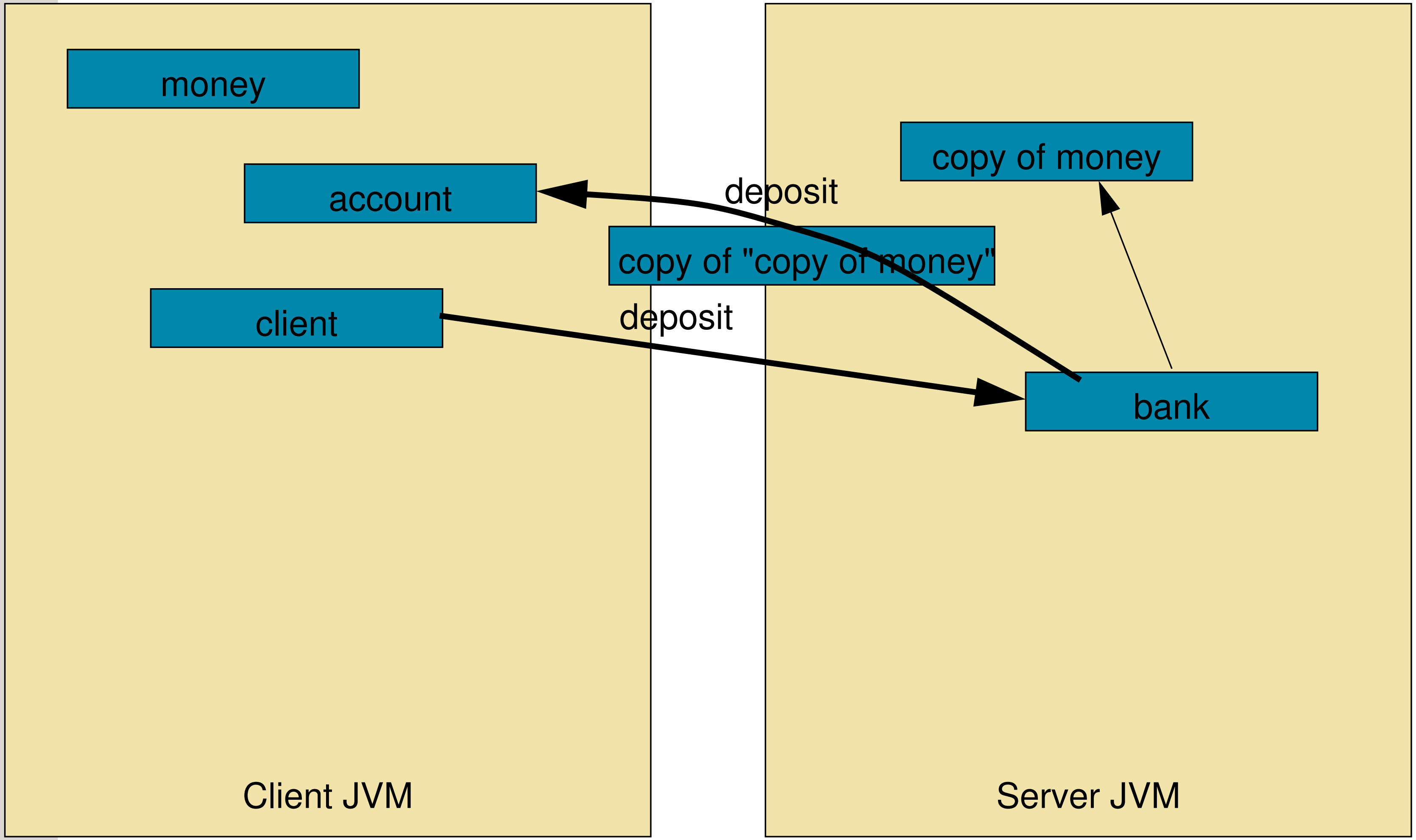
Reproduktion jeder Art oder Verwendung dieser Unterlage bedarf der Zustimmung des Autors.

38.10.8 Interaction during the remote call



Reproduktion jeder Art oder Verwendung dieser Unterlage bedarf der Zustimmung des Autors.

38.10.9 Interaction during the remote call



© 1997- 2000 Michael Golm

Reproduktion jeder Art oder Verwendung dieser Unterlage bedarf der Zustimmung des Autors.

38.11 Summary

- A remote object must implement a remote interface.
- All methods of the remote interface must throw RemoteException.
- Use/extend UnicastRemoteObject to create remote objects.
- Use Naming to bind / lookup the remote object at a registry.
- Clients only use the remote interface.

Reproduktion jeder Art oder Verwendung dieser Unterlage bedarf der Zustimmung des Autors.