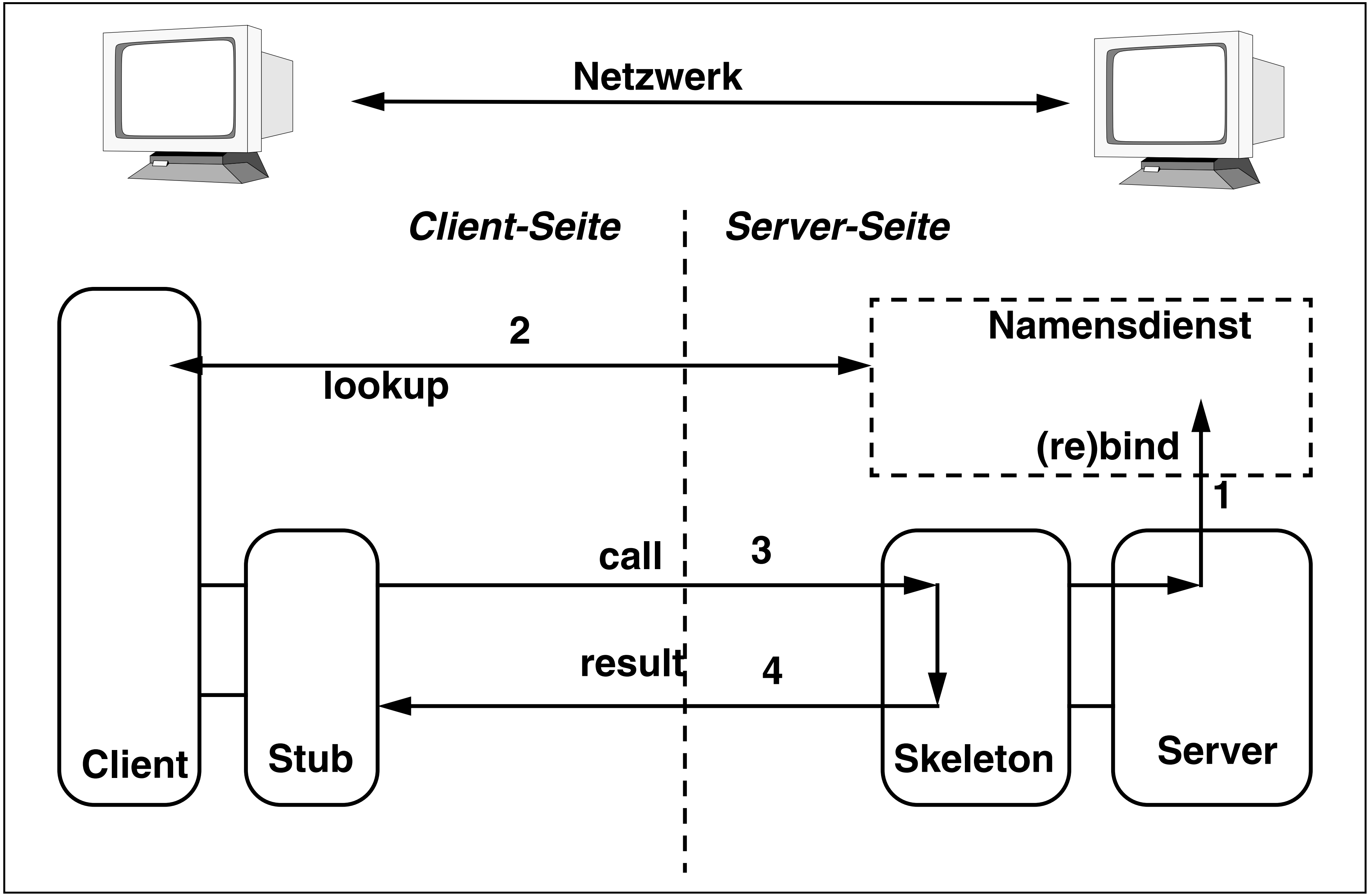
Remote Method Invocation

- spezielle Technik aus dem Java-Umfeld
- Ausführung der Methoden auf einem entfernten Rechner
- Analogon zum RPC (Remote Procedure Call)
- Zweck:
    - Objekte in verschiedenen Java-VM's
    - Aufruf einer Methode eines “remote”-Objektes
    - Programmieren wie mit lokalen Objekten
    - wenig Programmier-Overhead
    - verteilte Applikationen

RMI-Architektur (I)



RMI-Architektur (II)

- **Namensdienst wird ganz am Anfang gestartet**
- **Server-Seite:**

```
//entfernte Objekt anmelden
rmi.Naming.bind(“ObjectName“, RemoteObject)

//entfernte Objekt abmelden
rmi.Naming.unbind(“ObjectName“, RemoteObject)
```

- **Client-Seite:**

```
//Referenz (Stub) des entfernten Objektes bekommen
rmi.Naming.lookup(“rmi://some.server.com/ObjectName“)
```

RMI Probleme

Server:

- **muß Methoden nach außen verfügbar machen (exportieren)**
- **besitzt auch nicht remote-aufrufbare Methoden (nicht exportiert)**
- **ist mit einem Ort assoziiert**

Client:

- **muß Ort des Servers kennen**
- **muß Klasse des Servers kennen**
- **muß exportierte Methoden kennen**

RMI-Benutzung

- **Schritt 1: Implementierung der Server-Seite**
  - Basis Interface
  - Server Implementation
- **Schritt 2: Stub und Skeleton generieren**
  - automatische Generierung mittels *rmic*
- **Schritt 3: Client-Seite Implementieren**
  - Anfrage im Repository nach dem entfernten Server
  - Methoden-Aufruf wie auf dem lokalen Objekt
- **Schritt 4: Namensdienst aktivieren**
  - Starten der Namensdienst mit *rmiregistry*
- **Schritt 6: Server starten**
- **Schritt 7: Client starten**

RMI-Beispiel (I) - Der Server

Deklaration der entfernt zugreifbaren Methoden in einem Basis-Interface

```
interface Server extends Remote {
    public void do_something() throws java.rmi.RemoteException
}
```

ermöglicht das Weiterleiten der eventuellen Fehlermeldungen

Implementation des Server-Objektes

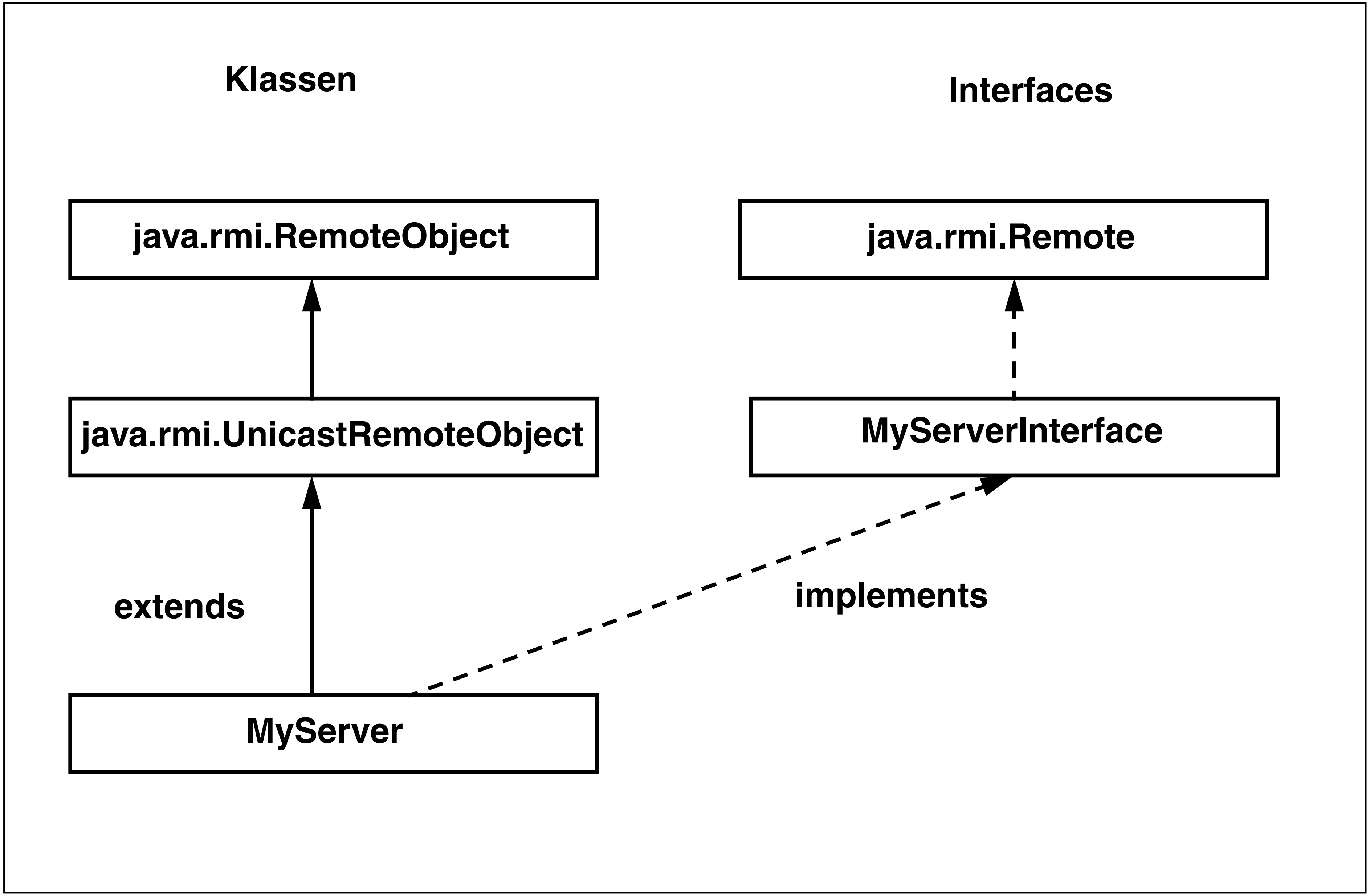
```
import java.rmi.*;
import java.rmi.server.*;

public class ServerImpl extends UnicastRemoteObject implements Server {

public ServerImpl throws RemoteException {
    super();
}
public do_something() { ... }
public static void main (String args[]){
    try {
        ServerImpl server=new ServerImpl();
        Naming.rebind(“MyServer“, server);
    } catch (Exception e){
        e.printStackTrace;
        }
    }
}
```

Punkt-zu-Punkt Kommunikation, Garbage-Collection

ermöglicht entfernter Zugriff auf die im Basis-Interface aufgelistete Methoden

Hier wird die entfernt aufrufbare Methode implementiert

RMI Klassen und Interfaces



RMI-Beispiel (II) - Stub und Skeleton

Klassen Stub und Skeleton werden automatisch wie folgt erzeugt:

- **ServerImpl.java compilieren**
- **Für automatische Erzeugung wird Tool *rmic* verwendet**

  >rmic ServerImpl

- **Dabei entstehen folgende Klassen:**

  ServerImpl_Stub.class
  - Stub für die Client-Seite

  ServerImpl_Skel.class
  - Skeleton für die Server-Seite

RMI-Beispiel (III) - Namensdienst

- **Verzeichnis der zur Verfügung gestellten entfernten Objekte**
- **spezielles Tool *rmiregistry*:**

  > rmiregistry <port> &

- **Namensdienst wird gestartet**
- **Portnummer ist ein optionaler Parameter (per Definition wird 1099 eingestellt).**
- **Fehlermeldung falls Port bereits von einem anderen Prozess verwendet wird**

RMI-Beispiel (IV) - Der Client

- **keine zusätzliche Vererbung von Interfaces und Objektklassen nötig**
- **Client findet den Server, indem er sich an den Namensdienst in dem Server-Rechner wendet**
- **bei Implementation assoziierter Name wird für die Nachfrage benutzt**

```
import java.rmi.*;

public class ClientImpl {

public static void main (String args[]) {

    try{
        Server server=(Server) Naming.lookup(“rmi://some.server.com/MyServer“);
        server.do_something();
    }
    catch (RemoteException e){
        System.out.println(e) }
}
}
```

RMI-Naming

Remote-Objekte müssen erreichbar sein:

- java.rmi.Naming - Service
- Katalog von verfügbaren Remote-Objekten
- enthält Referenzen auf Remote-Objekte
- Referenzen über Namen identifizierbar
- Namen sind wie URL's zusammengesetzt
- Naming-Methoden:
    - void bind(String url, Remote reference)
    - void rebind(String url, Remote reference)
    - void unbind(String url)
    - String[] list(String url)
    - Remote lookup(String url)

rmi://host.domain/objectname

Zusammenfassung

- RMI bietet Methode zum Verstecken von Verteilung
- vereinfacht das Programmieren
- aber: neue Fehlerquellen + Exceptions
- (fast) gleiche Behandlung von lokalen und Remote-Objekten
- Remote-Objekt fungiert als Server (exportiert Methoden)
- Client kann über ein Interface auf das Remote-Objekt zugreifen
- RMI-Paket bietet einfachen Naming-Service
- Parallele Ausführung von Methodenaufrufen im Server mittels Threads (erfordert Synchronisation!)