Verteilte Systeme:
Entfernte Methodenaufrufe

DHBW
Duale Hochschule
Baden-Württemberg
Karlsruhe
Wirtschaftsinformatik

- Funktionsweise entfernter Methodenaufrufe
- Entfernte Schnittstellen und verteilte Objektsysteme
- Entfernte Methodenaufrufe mit Java RMI
- Parameterübergabe Call-By-Value und Call-By-Reference

Lernziele dieses Kapitels

Unterschied von entfernten Aufrufen zu Streams und Sockets

Eigenschaften prozeduraler und objektorientierter Aufrufarten

Entfernte Referenzen, Schnittstellen und verteilte Objektsysteme

Technischer Ablauf eines entfernten Methodenabrufs

Funktionsumfang und Programmierung von Java RMI

Vor- und Nachteile von Java RMI

Was ist ein verteiltes System?

Ein verteiltes System besteht aus …

- Einem oder mehreren Computer und
- Mehreren parallel laufenden Anwendungsprozessen

Die Koordination erfolgt …

- Durch den Austausch einfacher Nachrichten
- Jedoch nicht durch direkte Methodenaufrufe etc.

Unter einer Nachricht versteht man …

- Eine Datenstruktur mit festem Aufbau,
- Welche die zu übermittelnden Informationen beinhaltet

Beispiel: Nachrichten

HTTP-Anfrage

Anfragezeile
Kopfdaten (optional)
Leerzeile
Anfragedaten (optional)

```
POST /service.php?client=55 HTTP/1.1
Host: dhbw-karlsruhe.de
Content-Type: text/xml
Content-Encoding: UTF-8
Content-Length: 542

<s:Envelope>
  <s:Body>
    <Vorlesung>
      Webentwicklung
    </Vorlesung>
  </s:Body>
</s:Envelope>
```

In der Übung

Kommando
Beliebig viele Parameter
Trennzeichen: /

```
WELCOME/Mark
```

```
CURRENT STATUS/16/23/41/Y
```

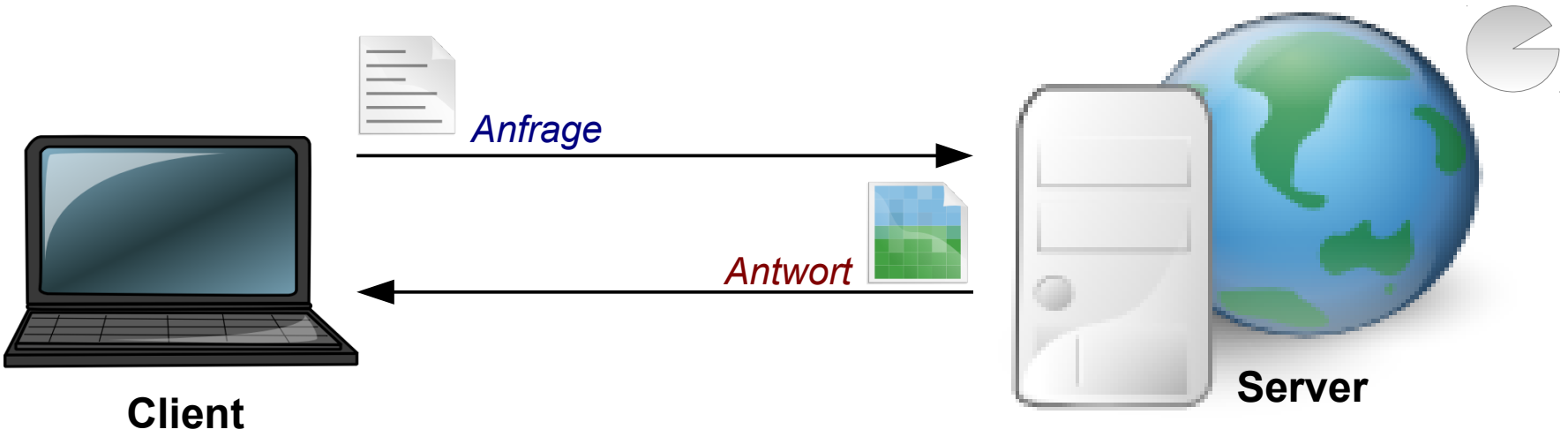
Grundsätzlich ist jede Datenstruktur, egal ob binär oder textbasiert, für den Austausch von Daten zwischen zwei Systemen geeignet. Bei Verwendung einer Middleware ist das Nachrichtenformat in der Regel vorgegeben. Sonst muss es selbst definiert werden.

Beispiel: Entfernter Aufruf

1. Der Client sendet eine Nachricht an den Server. Diese Nachricht wird **Anfrage** genannt, weil sie den Server auffordert, etwas zu tun.

2. Der Server verarbeitet die Anfrage und führt die gewünschte Aktion aus. Der Client wartet derweil.

3. Anschließend wird das Ergebnis der Verarbeitung als **Antwort** an den Client geschickt und die Verbindung getrennt.



Beispiel: Entfernter Aufruf

Server

```
public class CarService
implements CarRemote
extends UnicastRemoteObject {

  public String[] list() {
    // IDs aller Autos ermitteln
  }

  public Car getCar(String id) {
    // Details zu einem Auto
    // Rückgabe eines Objekts
  }

  public static void main() {
    // Objekt aufrufbar machen
    Naming.bind("MyCars",
      new CarService();
  }
}
```

Client

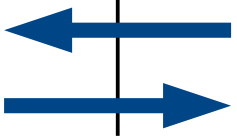
```
public class CarClient {

  public static void main() {
    // Entferntes Objekt suchen
    CarRemote cars = (CarRemote)
      Naming.lookup("MyCars");

    // Servermethoden aufrufen
    String[] ids = cars.list()
    Car kt = cars.getCar("Kit");

    ...
  }

}
```



Entfernte Aufrufarten

Prozedural

Remote Procedure Call / Webservices

Die Serveranwendung beinhaltet eine sogenannte Bibliothek von entfernt aufrufbaren Funktionen, welche Server-seitig ausgeführt werden können. Dabei besitzt jede Funktion einen Namen, Parameter sowie einen Rückgabewert.

int addCustomer(String)

void removeCustomer(int)

int placeOrder(int, float)

int cancelOrder(int)

Objektorientiert

Remote Method Invocation

Die Server-Anwendung verwaltet eine Menge von Objekten, deren Methoden über das Netzwerk aufgerufen werden können. Jedes Objekt besitzt seine eigenen Daten, wobei es natürlich auch Referenzen auf andere Objekte beinhalten kann.

CustomerService

Customer newCustomer()
Customer findCustomer(String)

Customer

Address getAddressData()
Order placeOrder(int, float)
List<Order> getAllOrders()

Address myAddress
List<Order> myOrders
TimeStamp lastAccess

RPC/RMI-Implementierungen

CORBA

Herstellerunabhängiger Standard, der für viele Betriebssysteme und Programmiersprachen zur Verfügung steht.

Beinhaltet eine eigene *Interface Definition Language*, um Schnittstellen zu beschreiben.

Java RMI

Im Lieferumfang des JDK enthalten. Schnittstellen werden durch einfache Java Interfaces definiert. Nicht mehr weiterentwickelt.

Apache Thrift, Pyro RPC, ...

Open-Source Bibliotheken für verschiedene Programmiersprachen.

XML-RPC Webservices

Leichtgewichtiger Standard für entfernte Prozeduraufrufe auf Basis von HTTP/XML.

SOAP-Webservices

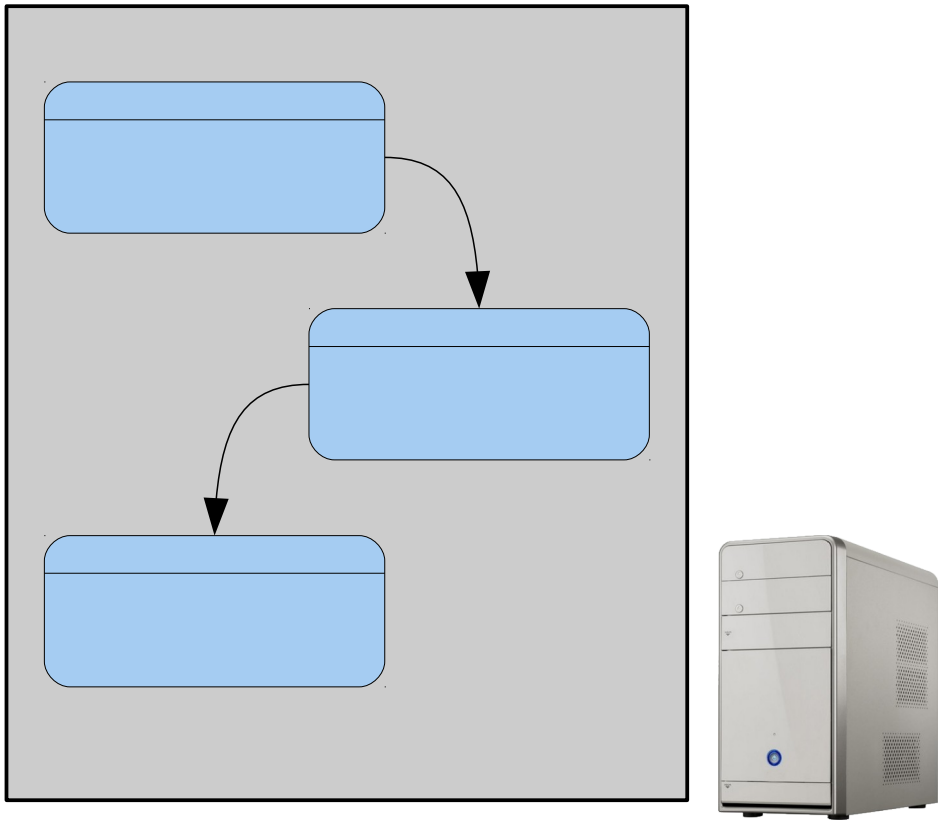
Nutzt die *Webservice Description Language*, um die ausgetauschten XML-Daten formal zu beschreiben.

Lauter neue Begriffe

Lokale und entfernte Objekte

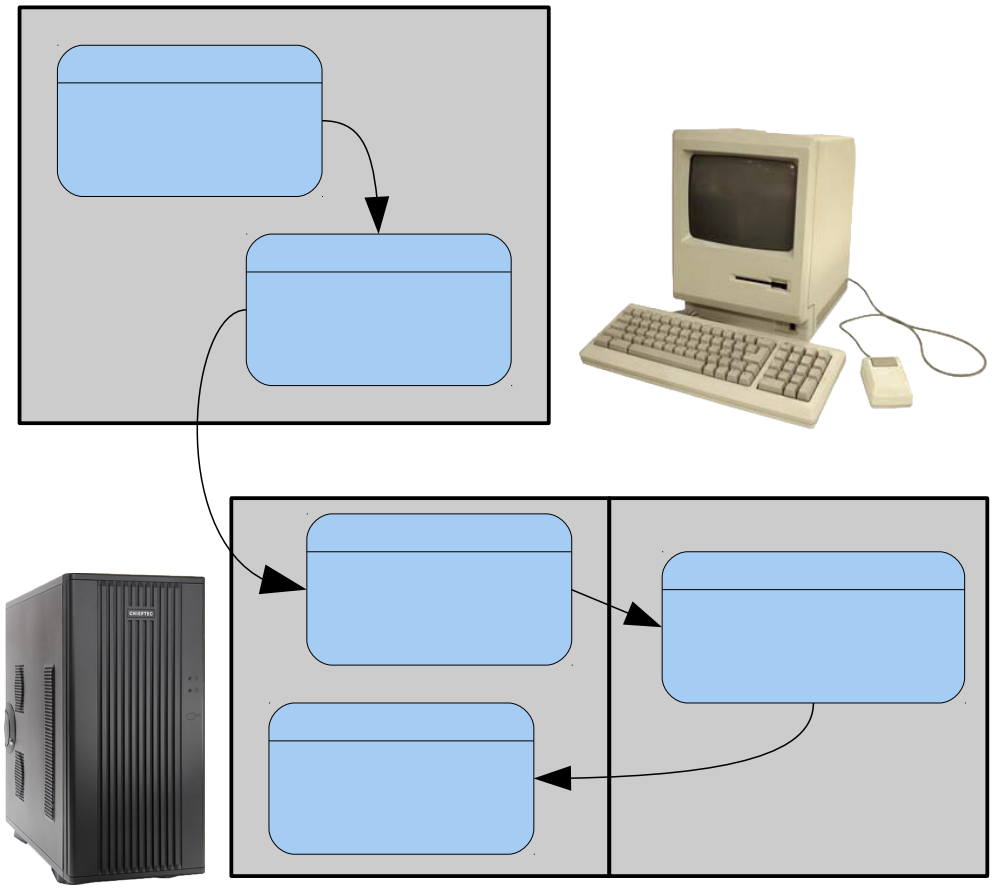
Lokale Objekte

Alle Objekte befinden sich im Speicher desselben Prozesses auf einem Computer.



Entfernte Objekte

Die Objekte befinden sich im Speicher verschiedener Prozesse, die teilweise auf unterschiedlichen Computern laufen.



Erinnerung: Objekte bisher

Objektsystem
Allgemeiner Begriff für die Objekte einer Anwendung. Das Objektsystem definiert, wie Objekte erzeugt, zerstört und benutzt werden können.

Objektreferenz
Eindeutige Identifikation eines Objekts innerhalb einer Anwendung

Schnittstelle
Definiert, welche Methoden eines Objekts aufgerufen werden können und welche Parameter diese besitzen

Ereignisse / Aktionen
Initiiert durch den Aufruf der Methoden eines Objekts. Häufig wird dabei der Zustand des Objekts verändert

Exceptions
Signalisieren ein fehlerhaftes Laufzeitverhalten und machen dieses behandelbar

Garbage Collection
Freigabe nicht mehr benutzten Speichers durch Zerstörung ungenutzter Objekte

Erweiterung um verteilte Objekte

Verteiltes Objektsystem

Das Objektsystem ist nicht mehr auf einen Prozess beschränkt. Die Objekte können in mehreren Prozessen auf mehreren Rechnern sein

Entfernte Objektreferenz

Eindeutige Identifikation eines Objekts über Prozess- und Rechnergrenzen hinweg

Entfernte Schnittstelle

Definiert die entfernt aufrufbaren Methoden. Dabei sind die verschiedenen Übergabesemantiken zu beachten!

Ereignisse / Aktionen

Werden sowohl durch lokale als auch durch entfernte Methodenaufrufe ausgelöst

Exceptions

Neue Fehlerursachen führen zu mehr Exceptions. Entfernte Aufrufe führen dazu, dass Ausnahmen in fremden Prozessen entstehen können

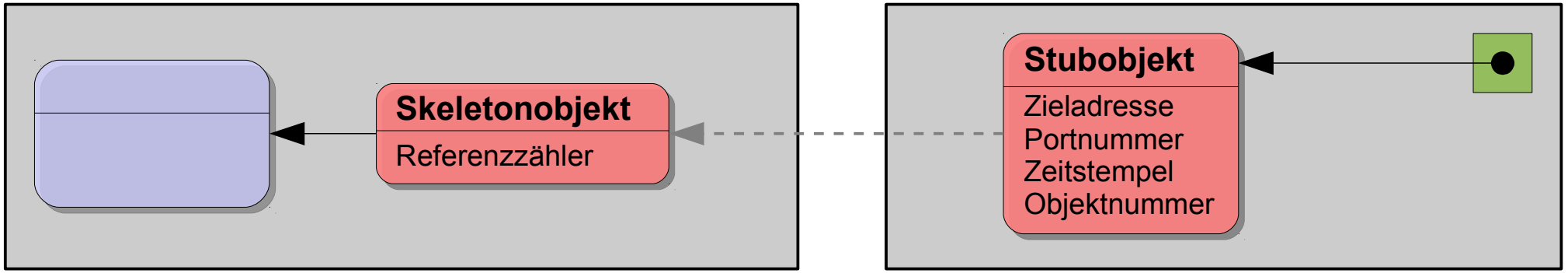
Garbage Collection

Muss über Prozess- und Rechnergrenzen hinweg auch entfernte Referenzen berücksichtigen können

Entfernte Aufrufe im Detail

1. Der Server erzeugt ein entferntes Objekt und meldet es bei der Middleware an. Es entsteht ein lokales Skeletonobjekt, um das sich der Server allerdings nicht kümmern muss.

2. Der Client fordert eine Referenz auf das entfernte Objekt an und erhält daraufhin ein lokales Stubobjekt, dessen Methoden er aufrufen kann.



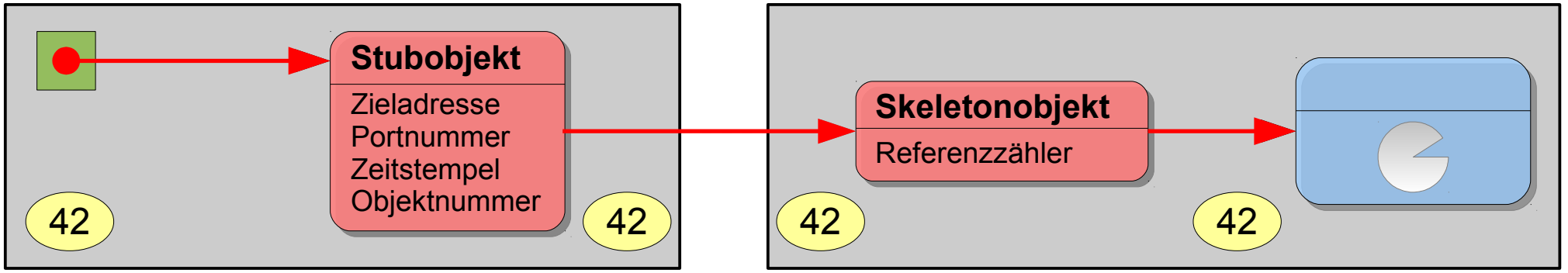
Entfernte Aufrufe im Detail

3. Der Client ruft eine Methode des erhaltenen Stubobjekts auf.

4. Der Stub versendet die Anfrage über das Netzwerk an das Skeleton.

5. Das Skeletonobjekt empfängt die Anfrage und ruft die echte Methode auf.

6. Der Rückgabewert der Methode wird zurückgesendet und an den Client übergeben.



Entfernte Objekte suchen

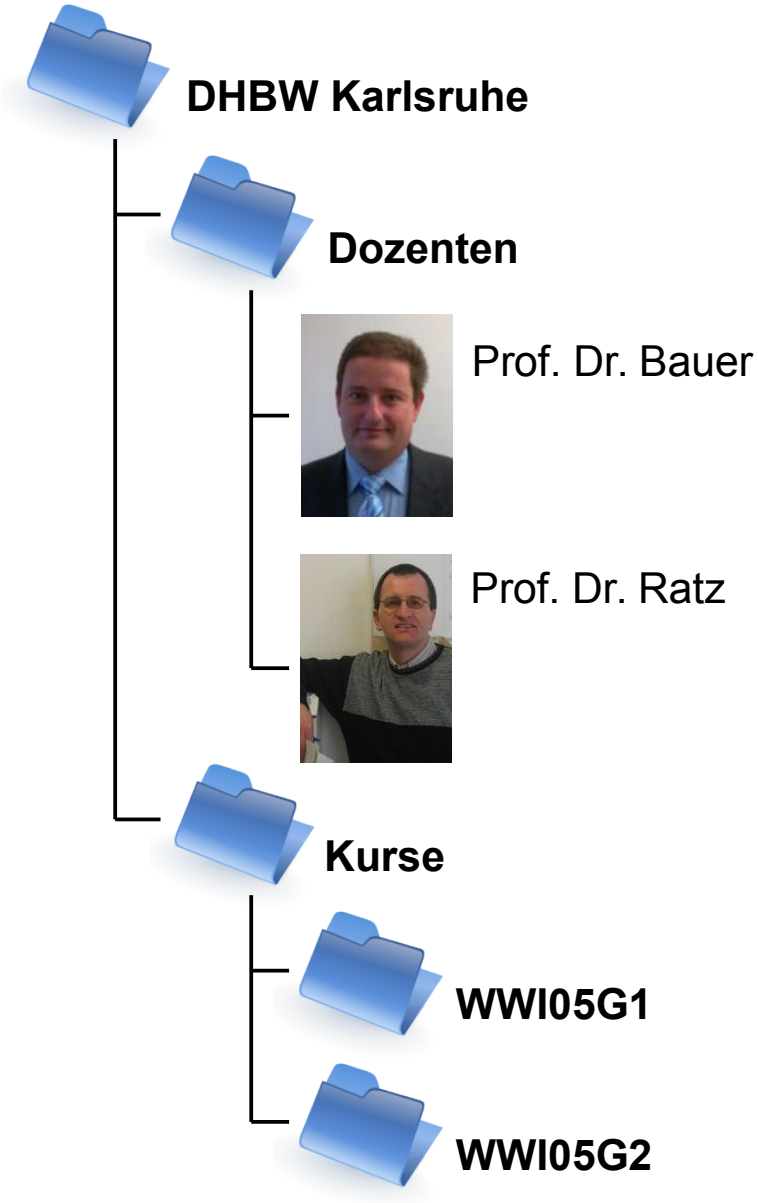
Die Middleware verwaltet alle entfernten Objekte mithilfe eines Namens- oder Verzeichnisdienstes

Namensdienste

- Verwalten eine Menge von Objekten
- Jedes Objekt erhält einen Namen
- Verwaltung als flache Liste oder hierarchische Baumstruktur (je nach Dienst)

Verzeichnisdienste

- Erweiterung der Namensdienste
- Ordnen den Objekten zusätzliche Metadaten zu (z.B. Zugriffsrechte)



Orts- und Mobilitätstransparenz

Clients kennen die Position der entfernten Objekte nicht. Denn Jedes Objekt wird nur über seinen Namen gesucht.

Ändert sich die Position des Objekts, muss nur sein Eintrag im Namensdienst angepasst werden

Objekte können darum beliebig verschoben werden, ohne dass negative Auswirkungen entstehen.

Clients müssen allerdings ihre Referenzen auffrischen, wenn ein Objekt verschoben wurde.

Vorgehen bei der Entwicklung

Auf Serverseite

Remote Interface definieren

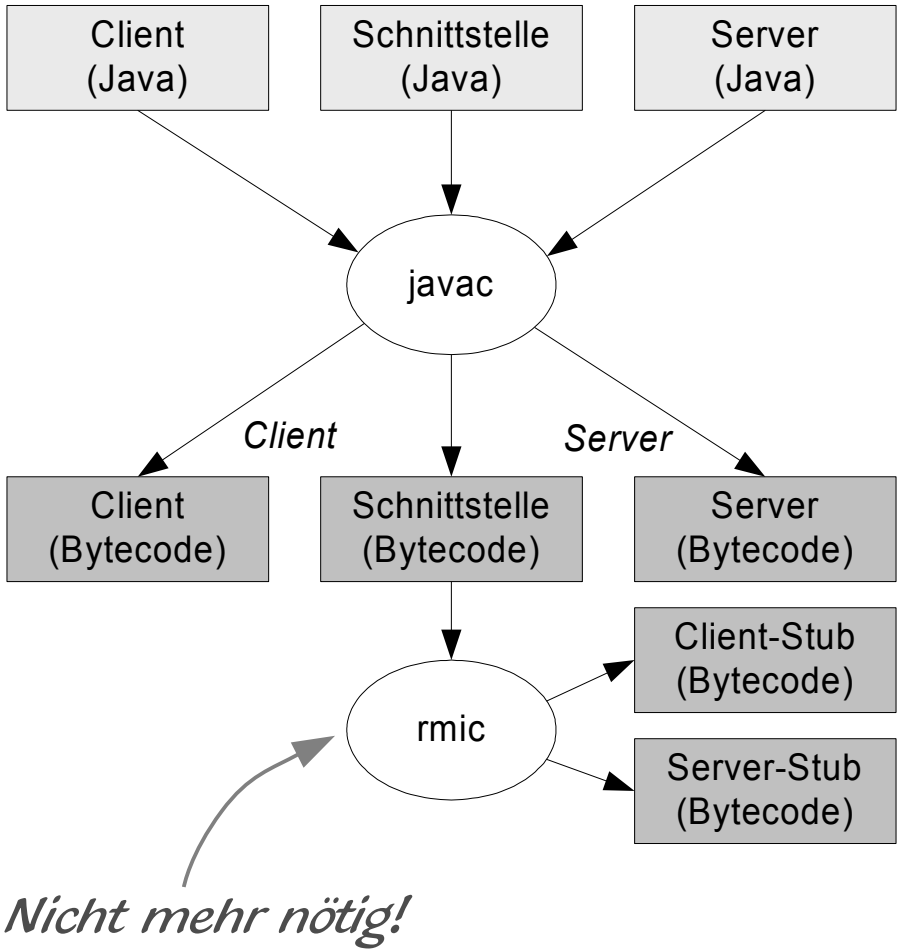
Ausprogrammieren mindestens einer Serviceklasse, die das Remote Interface implementiert

Entwicklung einer Anwendung, die ein Objekt der Serviceklasse erzeugt und beim Namensdienst registriert

Auf Clientseite

Das kompilierte Remote Interface in die Clientanwendung kopieren

Ausprogrammieren des Clients



Remote Interface definieren

Definition der Schnittstelle durch ein Java Interface:

- Das Interface muss von **`java.rmi.Remote`** abgeleitet werden
- Jede Methode muss eine **`RemoteException`** deklarieren
- Davon abgesehen sind keine Einschränkungen vorhanden

```
import java.rmi.Remote;
import java.rmi.RemoteException;

public interface TimeService extends Remote {
    public String getDate() throws RemoteException;
    public String getTime() throws RemoteException;

    public void setTimezone(Timezone tz)
        throws InvalidTimeZoneException, RemoteException;
}
```

Serviceklasse entwickeln

Ausprogrammieren des eigentlichen Diensts:

- Die Klasse muss das Remote Interface implementieren
- Alle Methoden des Interfaces müssen ausprogrammiert werden
- Zusätzlich kann die Klasse von **`UnicastRemoteObject`** erben

```
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;

public class ExampleTimeService
extends UnicastRemoteObject implements TimeService {
    public String getDate() throws RemoteException { ... }
    public String getTime() throws RemoteException { ... }
    public void setTimezone(Timezone tz) throws ...
    ...
}
```

Serveranwendung schreiben

Anmelden des Objekts beim Namensdienst:

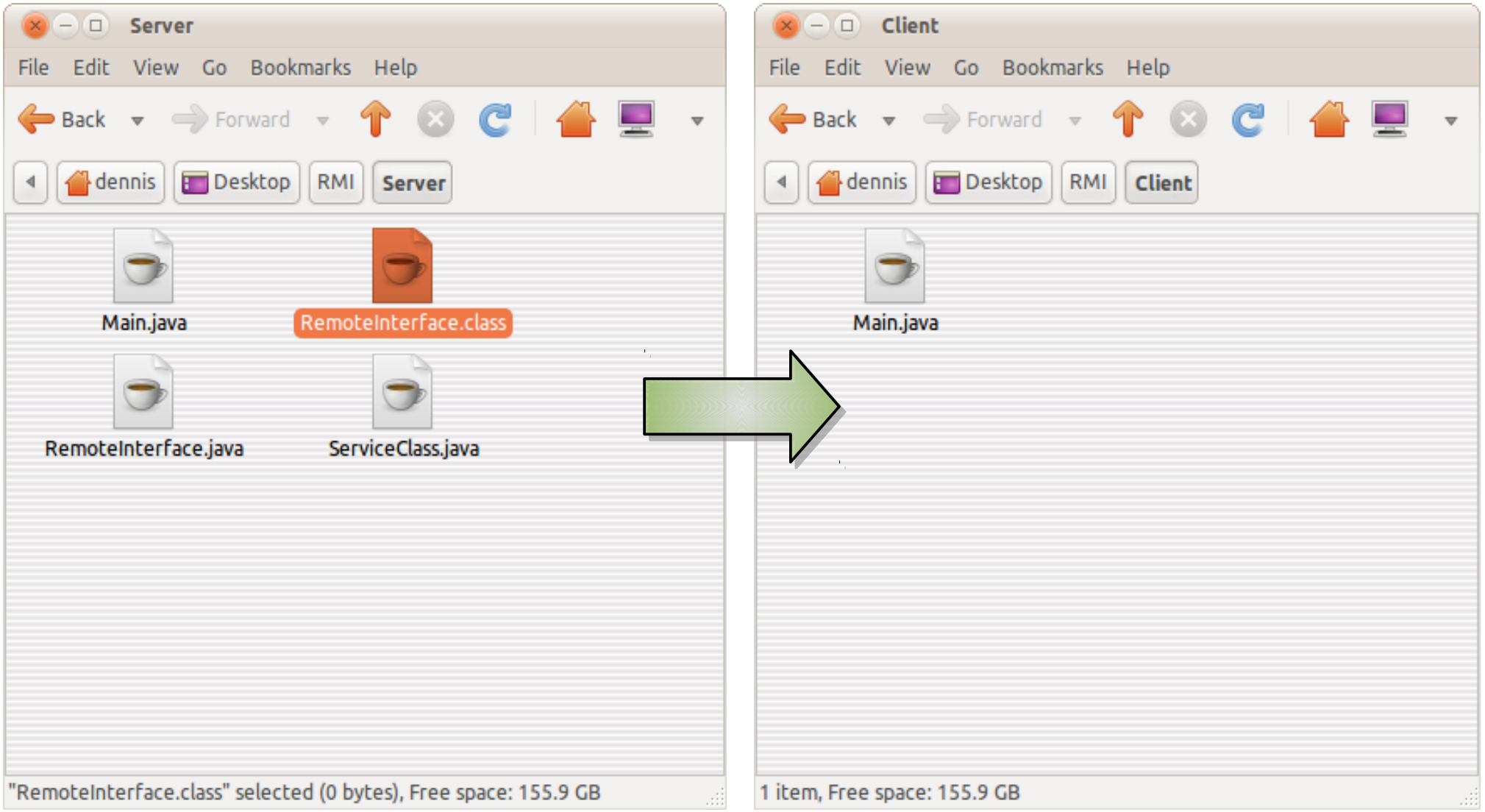
- Erst muss das Objekt vom Serverprogramm erzeugt werden
- ~~Zusätzlich muss noch ein Stub(!)objekt erzeugt werden~~
- Das ~~Stub~~objekt wird in der **`rmiregistry`** abgelegt

```
import java.rmi.AlreadyBoundException;
import java.rmi.Naming;
import java.rmi.server.UnicastRemoteObject;

public class Main {
    public static void main(String[] args) {
        LocateRegistry.createRegistry(Registry.REGISTRY_PORT);

        TimeService timeService = new ExampleTimeService();
        TimeService timeServiceStub = (TimeService)
                    UnicastRemoteObject.exportObject(timeService, 0);
        try {
            Naming.bind("TimeService", timeServiceStub);
        } catch (AlreadyBoundException ex) { ... }
    }
}
```

Remote Interface kopieren



Clientanwendung schreiben

Demo

Entfernte Referenz des Objekts anfordern:

- Stub der entfernten Referenz im Namensdienst nachschlagen
- Aufruf der entfernten Methoden mit dem Stubobjekt durchführen
- Dabei auftretende **`RemoteExceptions`** behandeln

```
import java.rmi.Naming;
import java.rmi.RemoteException;

public class Main {
    public static void main(String[] args) {
        try {
            TimeService ts = (TimeService)
                Naming.lookup("rmi://localhost:1099/TimeService");
            System.out.println(ts.getDate());
        } catch (RemoteExcetpion ex) { ... }
    }
}
```

Nachteile von Java RMI

Client und Server müssen in Java programmiert sein

Java RMI wird nicht mehr aktiv weiterentwickelt, Technologie ist im Prinzip veraltet

Kann nicht in Android-Apps genutzt werden, obwohl diese in Java geschrieben werden

Jedes Serverobjekt belegt einen eigenen TCP-Port auf dem Server!

Kommunikation kann nur sehr umständlich verschlüsselt werden

1st Law of Distributed Objekts: Don‘t Distribute Them!

- Idee der verteilten Objekte hat sich nicht bewahrheitet
- Grundidee ist einfach, die Details sind aber komplex
  (z.B. müssen die Objekte thread-safe sein, wenn mehrere Clients auf sie zugreifen)

Übergabe-Semantiken

Frage: Wie werden Aufrufparameter und Rückgabewerte zwischen den kommunizierenden Prozessen übertragen?

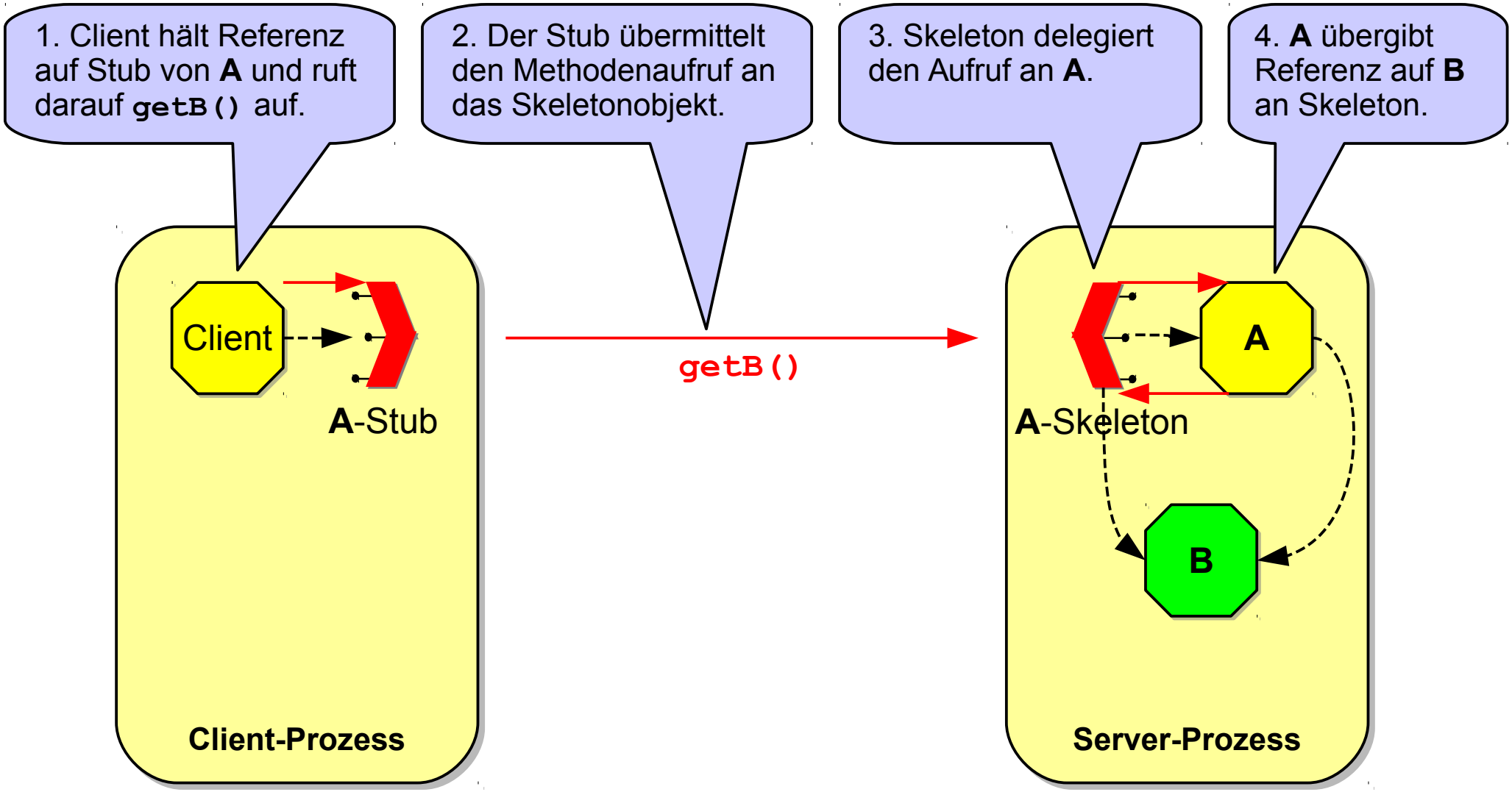
Übertragung einer Kopie (Call By Value):

- Normale Objekte und elementare Werte werden einfach kopiert
- Übertragung einer serialisierten Version der Objekte
- **Achtung:** Alle referenzierten Objekte werden ebenfalls kopiert
- Lokale Kopie der Objekte bei Client und Server vorhanden

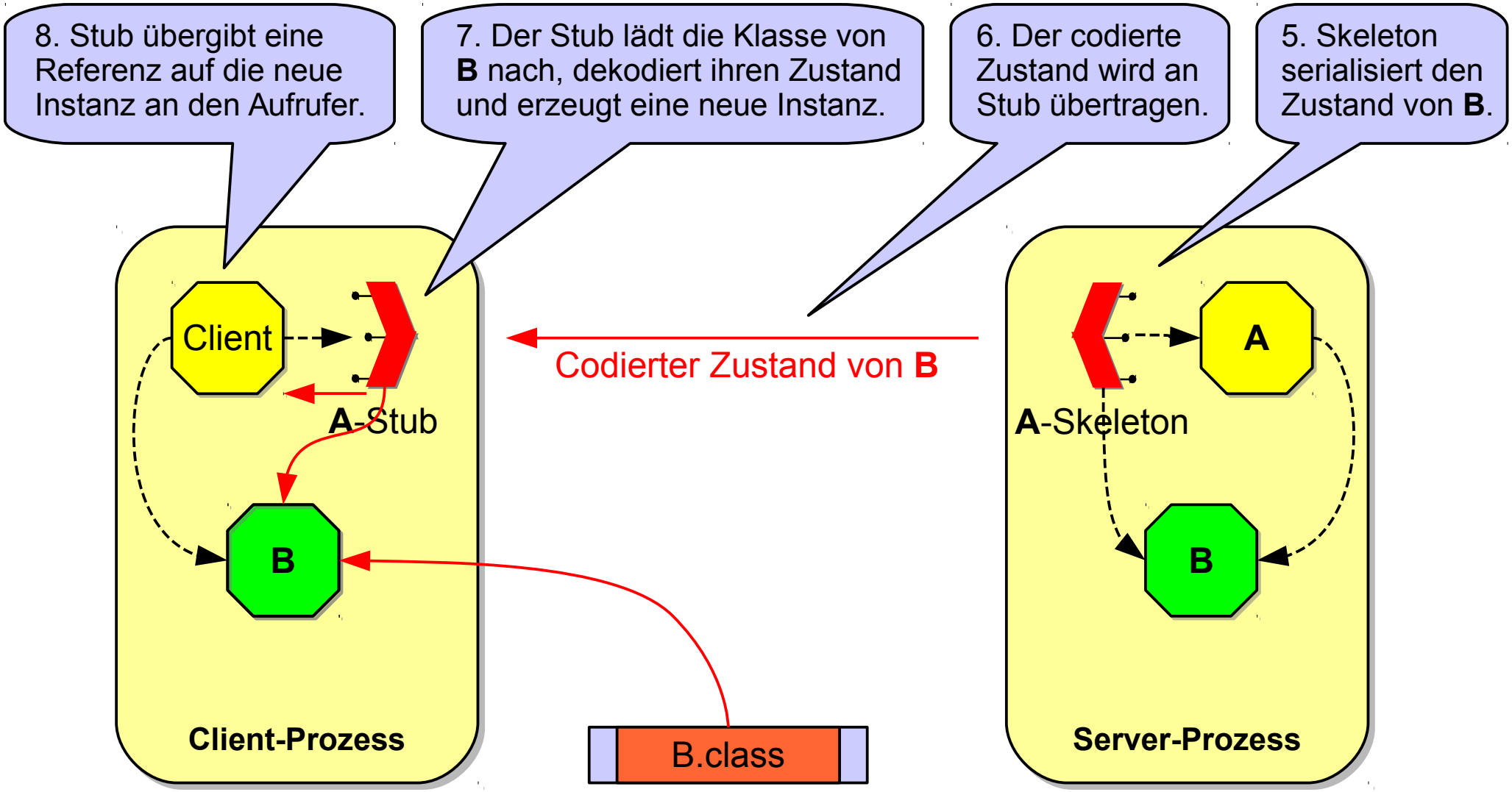
Übertragung einer Referenz (Call By Reference):

- Anstelle eines remotefähigen Objekts wird der Stub übertragen
- Der Client erhält einfach eine entfernte Referenz auf das Objekt
- Entfernte Objekte werden also nicht serialisiert oder kopiert

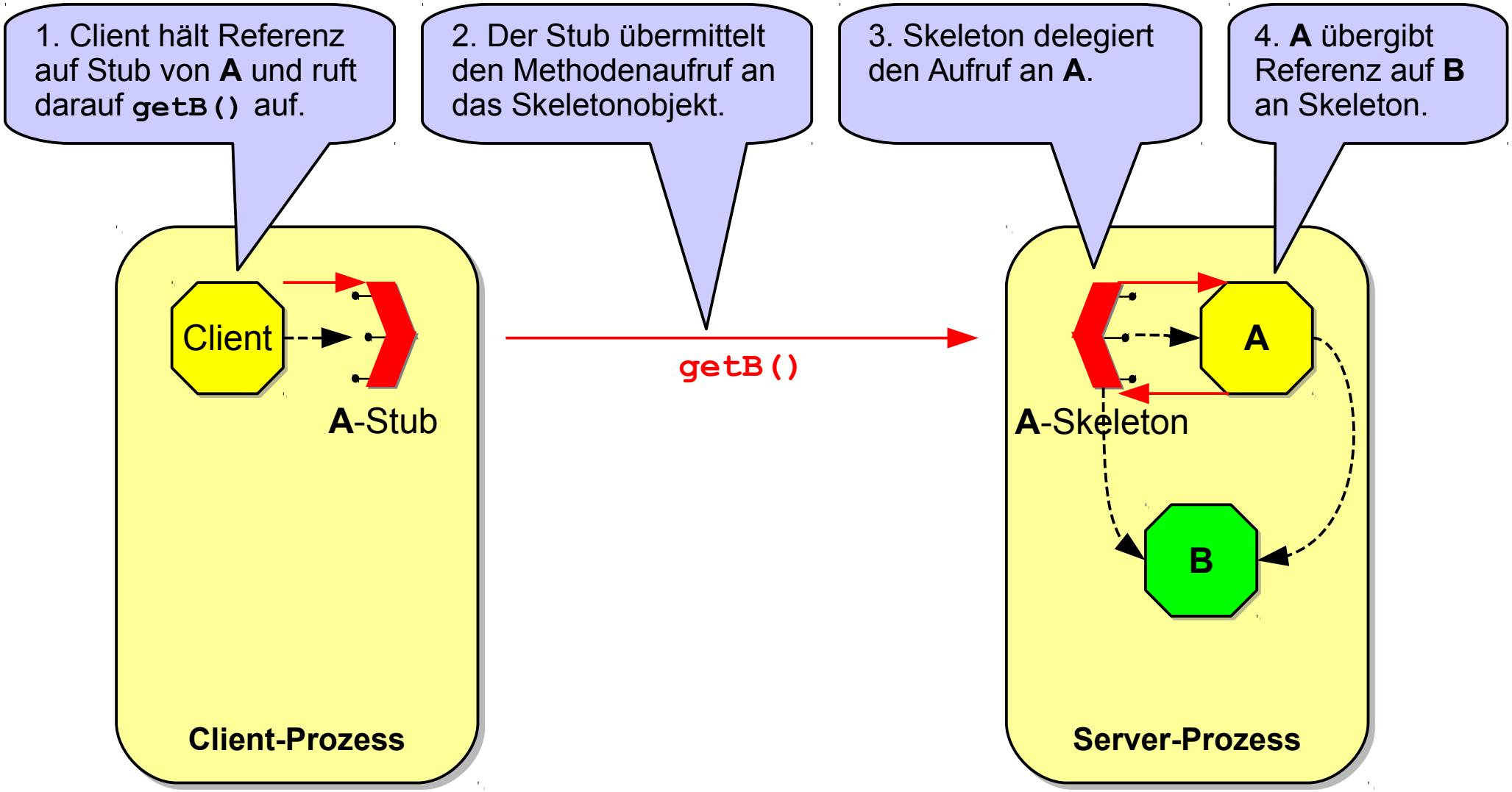
Beispiel: Kopier-Semantik



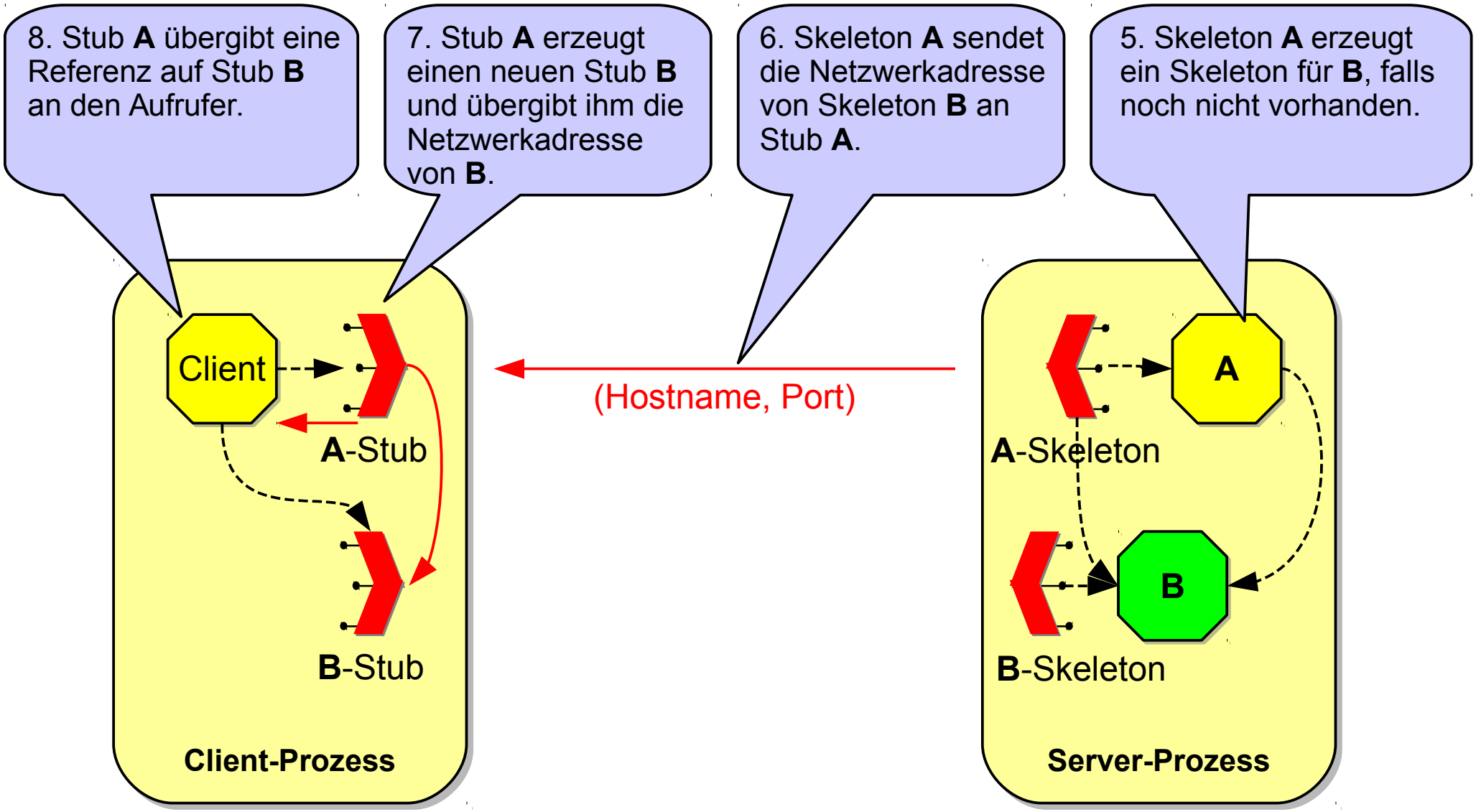
Beispiel: Kopier-Semantik



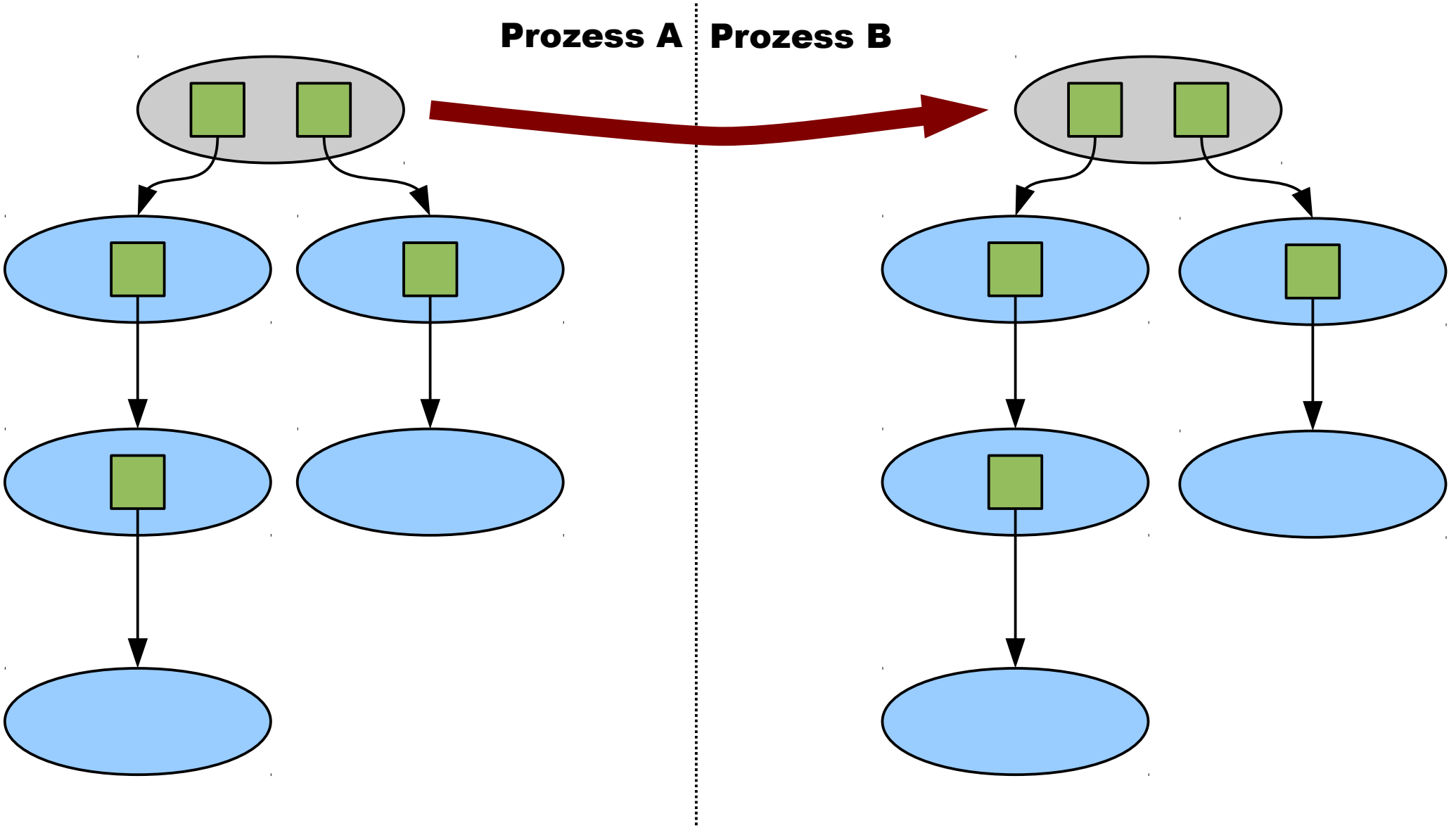
Beispiel: Referenz-Semantik



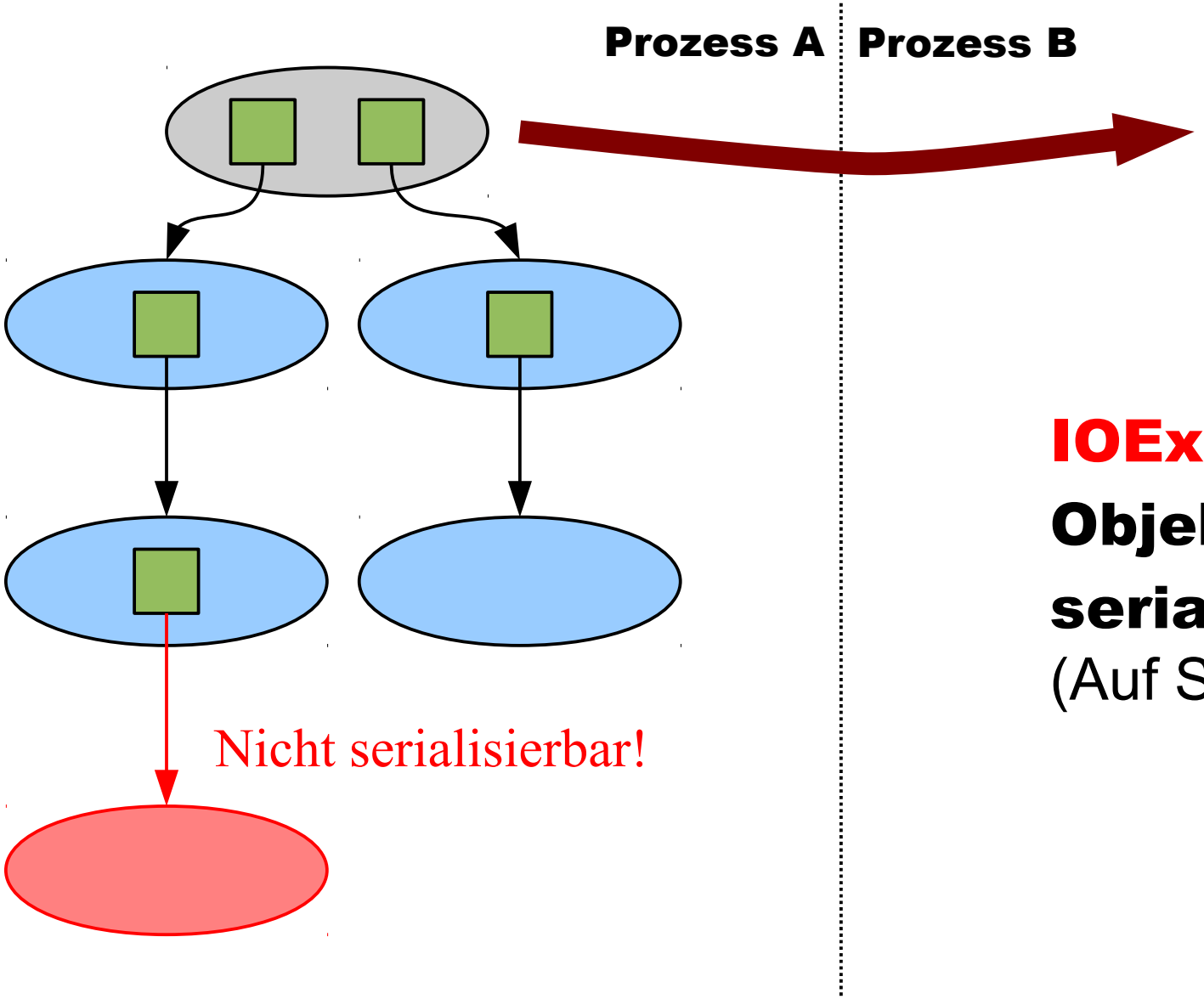
Beispiel: Referenz-Semantik



Probleme der Kopier-Semantik



Probleme der Kopier-Semantik

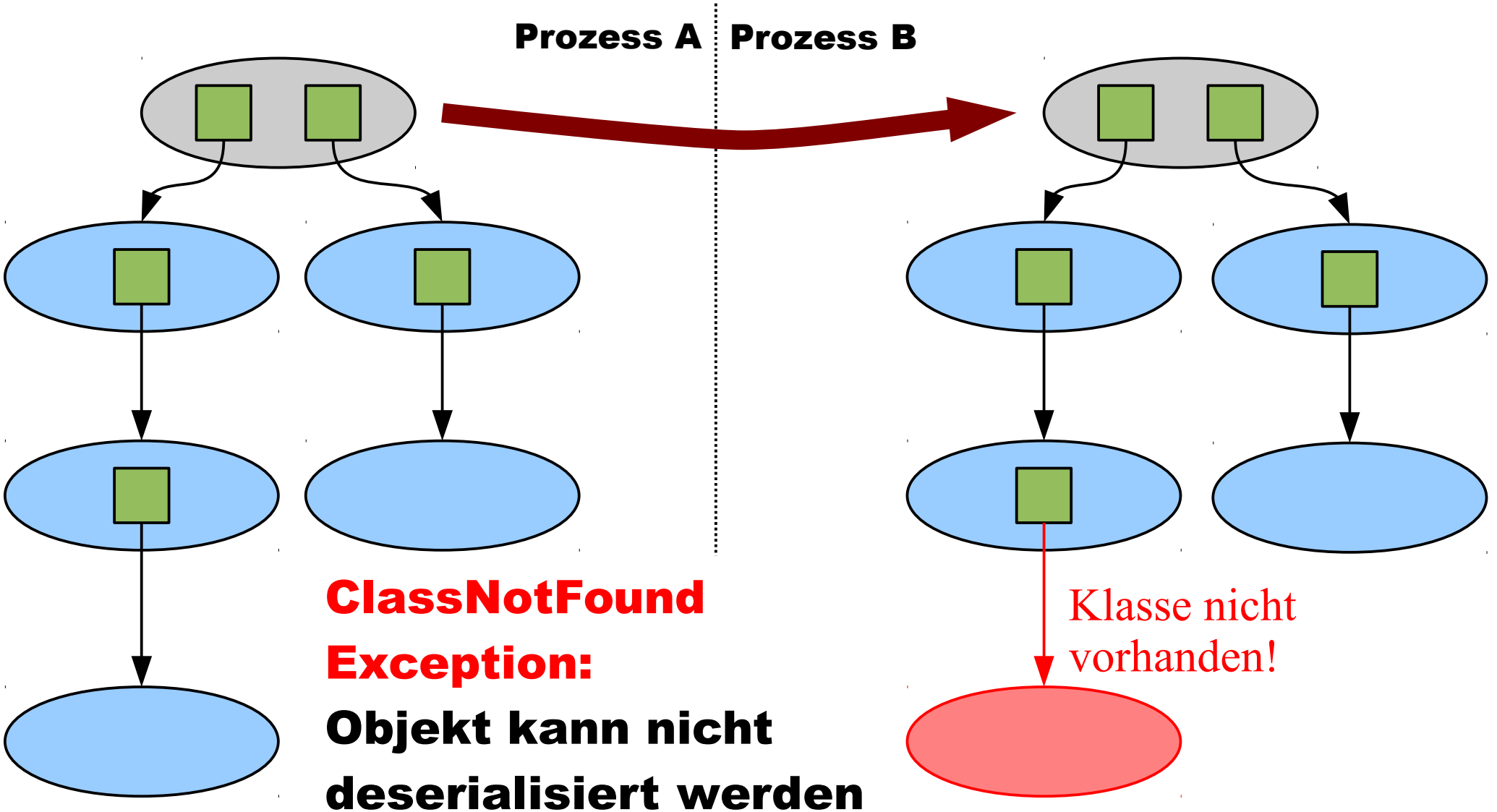


IOException:

Objekt kann nicht serialisiert werden

(Auf Senderseite)

Probleme der Kopier-Semantik

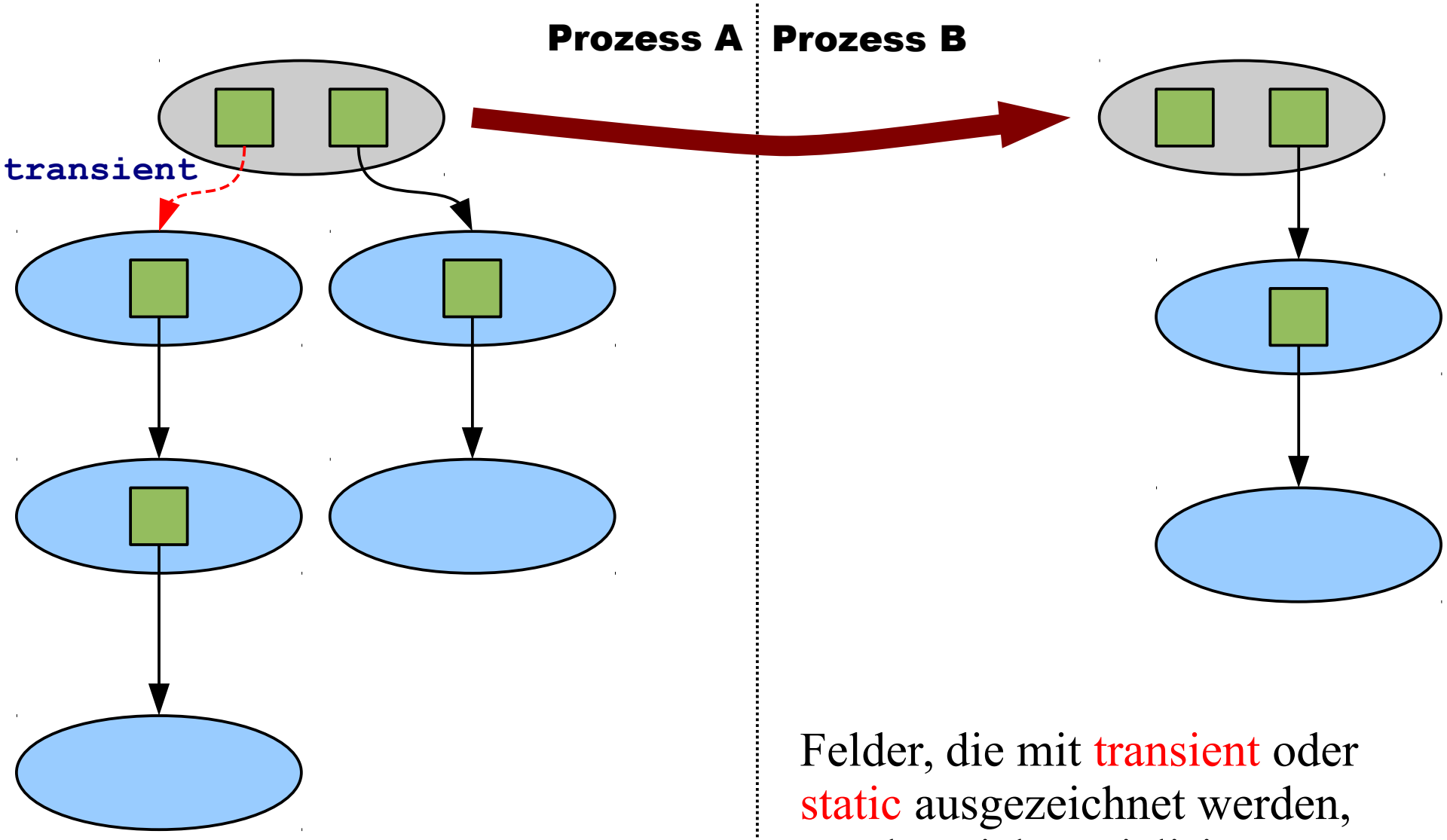


ClassNotFound Exception:

Objekt kann nicht deserialisiert werden

(Auf Empfängerseite)

Probleme der Kopier-Semantik



Felder, die mit transient oder static ausgezeichnet werden, werden nicht serialisiert.

Lernkontrolle

Kennen Sie den Unterschied zwischen Netzwerkprogrammierung mit Streams/Sockets und entfernter Aufrufe? Wo liegen die Vor- und Nachteile beider Techniken?

Wie unterscheiden sich prozedurale und objektorientierte Aufrufe?

Was passiert im Hintergrund, wenn Sie eine entfernte Methode aufrufen?

Was sind Stub- und Skeletonklassen? Wozu braucht man sie?

Was ist der Namens- und Verzeichnisdienst und wozu braucht man ihn?

Können Sie mit RMI entfernt aufrufbare Objekte erzeugen und verwenden?

Wissen Sie, wann ein Objekt *By Reference* übertragen wird?