D Überblick über die 3. Übung

- RPC und ORB
  - Aufrufsemantiken
- Hinweise Aufgabe 2

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

D.1 Organisatorisches

- Präsentation der Übungsaufgaben 1 und 2
  - Live in der Rechnerübung am 15. oder 22. Nov.
  - Ein Gruppenmitglied ausreichend, Mitglieder müssen sich abwechseln
  - Alternativer Termin nach Rücksprache per eMail (mit Begründung)

MW - Übung

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

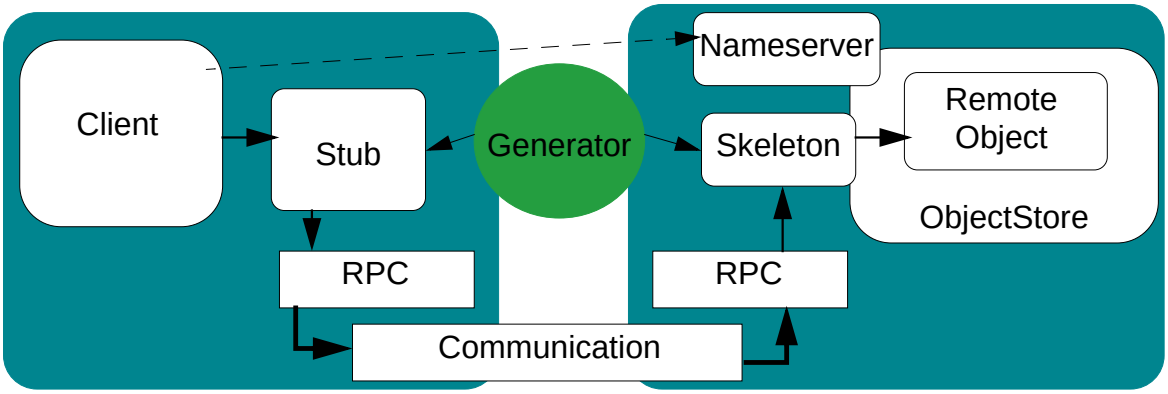
D.2 Object Request Brokers

- ermöglichen Methodenaufrufe an entfernten Objekten (Objekte in anderen JVM)
- Beispiel-ORBs: RMI, JavaIDL

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

1 Komponenten eines ORBs

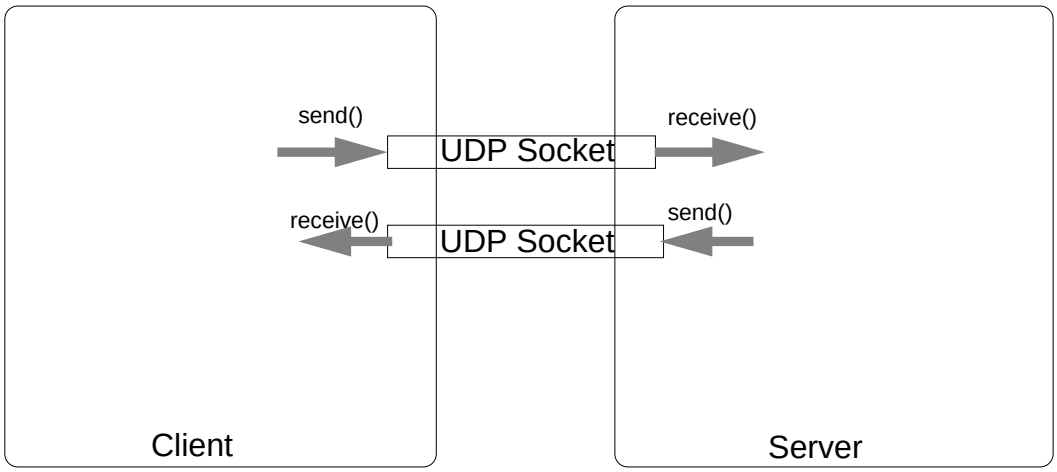
- *Kommunikationsschicht*: tauscht Daten zwischen zwei Rechnern aus
- *RPC Schicht*: definiert die Aufrufsemantik und das Marshalling
- *Object Store*: verwaltet den Lebenszyklus der Objekte
- *Stub / Skeleton Generator*: erzeugt Code für die Stubs und die Skeletons
- *Nameserver*: findet Objekte anhand deren Namen



Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

2 Kommunikationsschicht

- zuständig für den Datenaustausch zwischen zwei Rechnern
- verwendet **`DatagramSocket`** (UDP)



MW - Übung

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

3 RPC-Schicht

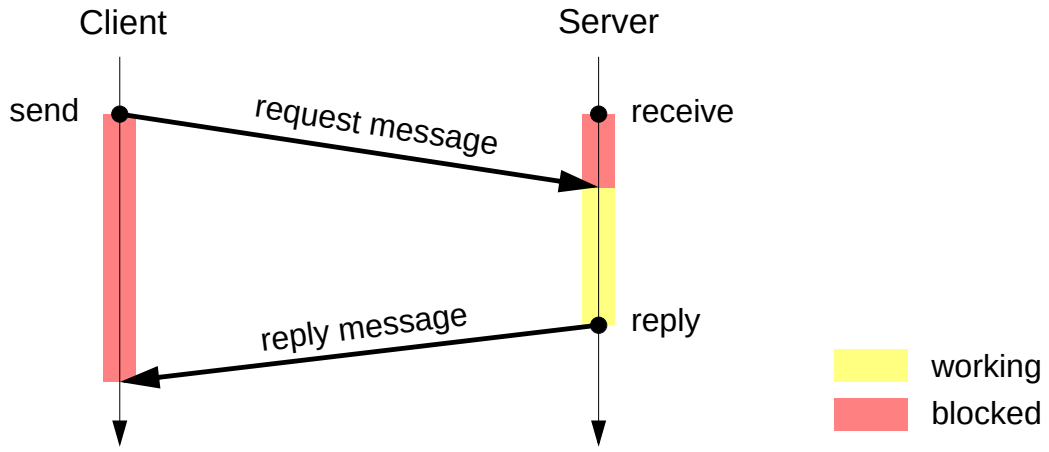
- erledigt die Weiterleitung von Methodenaufrufen
- wird von den Stubs und den Skeletons verwendet
- verwendet die Kommunikationsschicht um Bytes zu versenden

MW - Übung

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

3 RPC

- einfachster RPC ist ein primitives (synchrones) request/reply-Protokoll:



Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

3 Marshalling

- RPC-Schicht verpackt die Parameter in ein Anfragepaket und den Rückgabewert in ein Antwortpaket == Marshalling
- kopiere alle Parameter zum Server
  (besser wäre es, eine Referenz zu verschicken wenn ein Parameter ein *remote Interface* implementiert)
- verwende ObjectStreams/ByteArrayStreams/Datagramme um Objekte zu verschicken:

```
ByteArrayOutputStream stream = new ByteArrayOutputStream();
ObjectOutputStream out = new ObjectOutputStream(stream);
out.writeObject(...);
byte[] buf = stream.toByteArray();
DatagramPacket packet = new DatagramPacket(buf, ... );
```

- Annahme für Aufgabe 2: alle Parameter passen in ein **`DatagramPacket`**

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

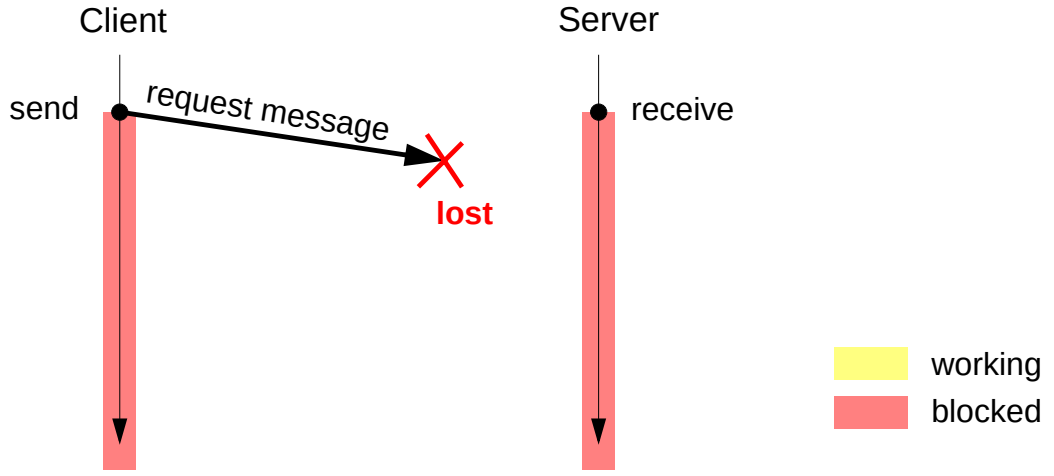
3 Fehlerbehandlung

- Implementierung einer bestimmten Aufrufsemantik
  - exactly once
  - at least once
  - at most once
  - last of many
  - ...
- abhängig von der Servicequalität der Kommunikationsschicht
- muss mit Kommunikationsfehlern umgehen können:
  - verlorene Pakete
  - veränderte Reihenfolge (non-FIFO)
  - duplizierte Pakete
  - veränderte Pakete

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

3 Fehlerbehandlung - Problem 1

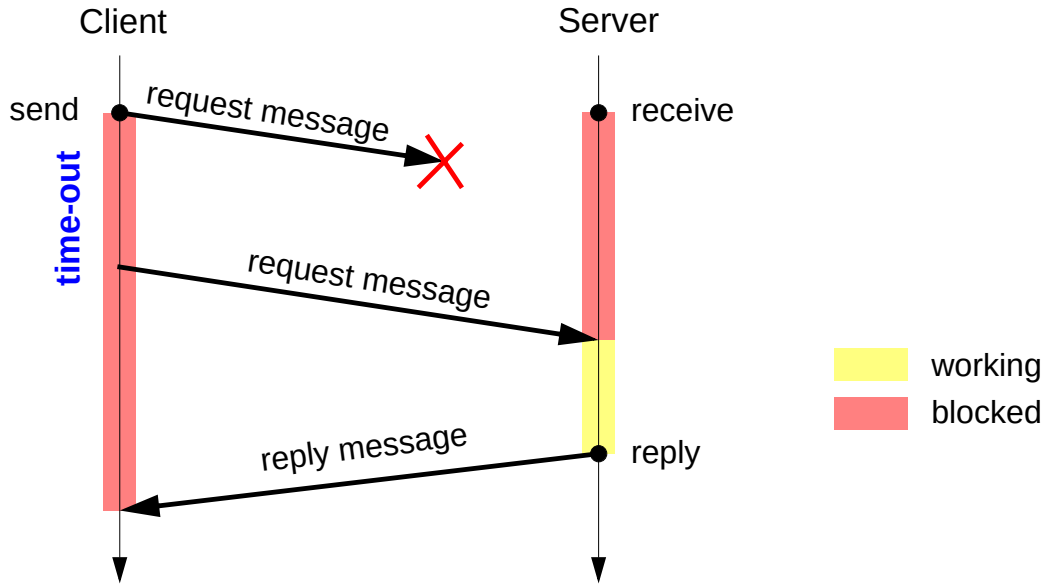
- Kommunikationsfehler: Verlust einer Anfragenachricht



Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

3 Fehlerbehandlung - Lösung 1

- Verlust einer Anfragenachricht:
  - mögliche Lösung: Anfrage erneut schicken

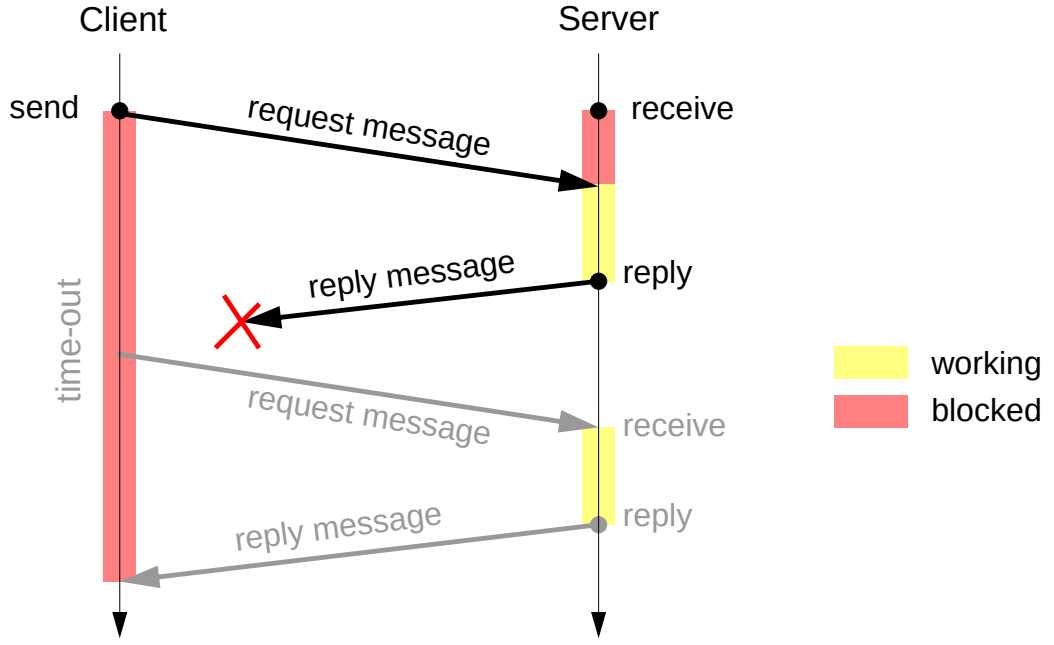


MW - Übung

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

3 Fehlerbehandlung - Problem 2

- Kommunikationsfehler: Verlust einer Antwortnachricht
  - bei Wiederholung wird die Methode mehrmals ausgeführt

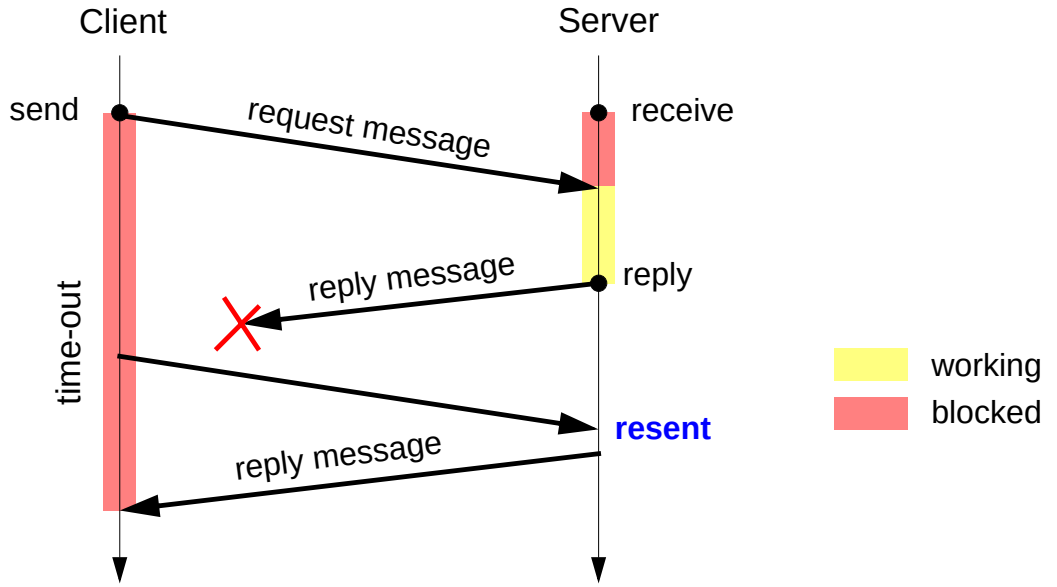


MW - Übung

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

3 Fehlerbehandlung - Lösung 2

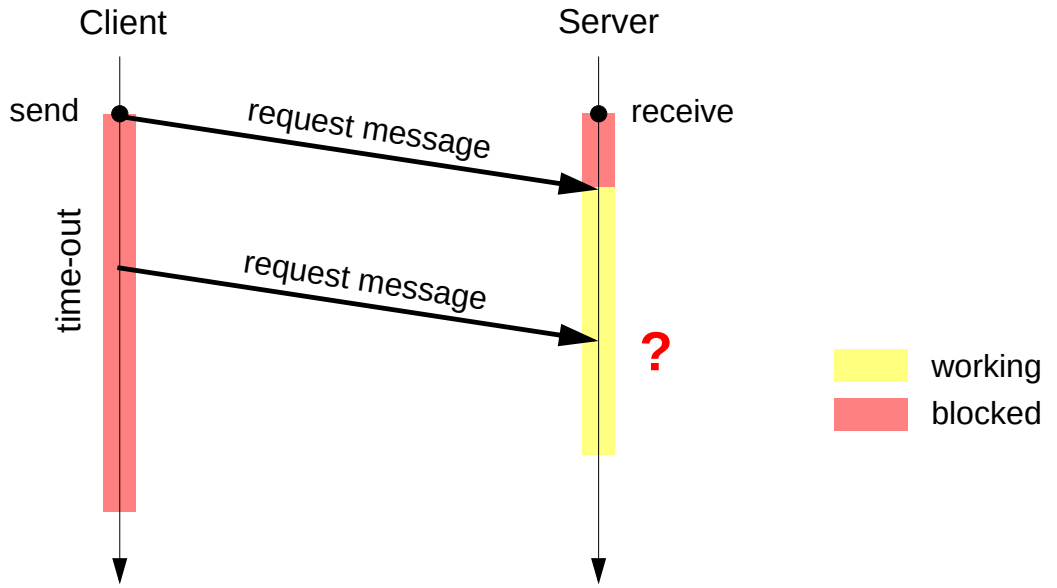
- Verlust einer Antwortnachricht
  - Server hebt Antwort auf und wiederholt sie



Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

3 Fehlerbehandlung - Problem 3

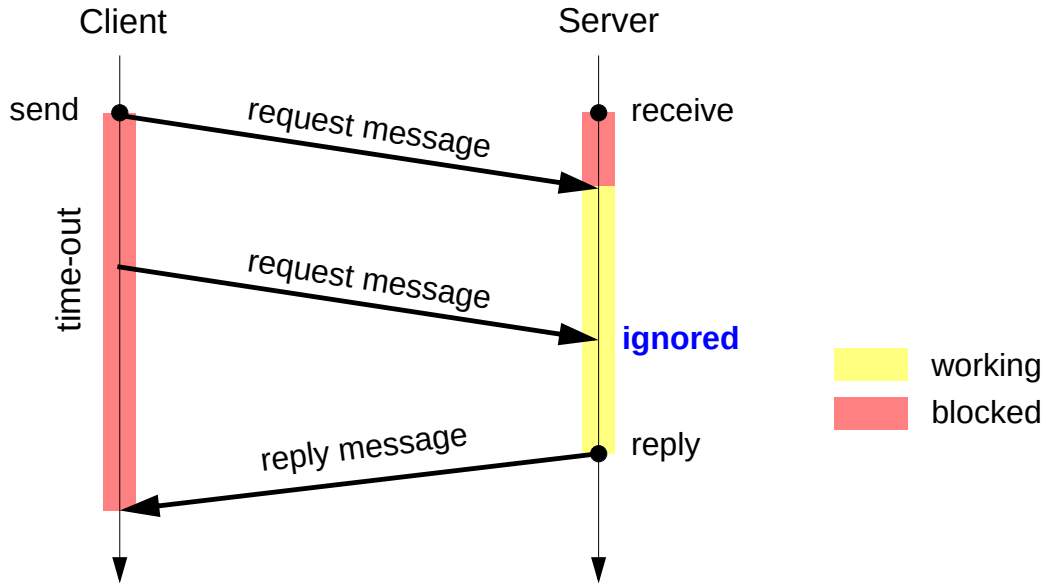
- Bearbeitung dauert länger als Timeout bei Client
  - Bearbeitung ist noch nicht fertig



Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

3 Fehlerbehandlung - Lösung 3

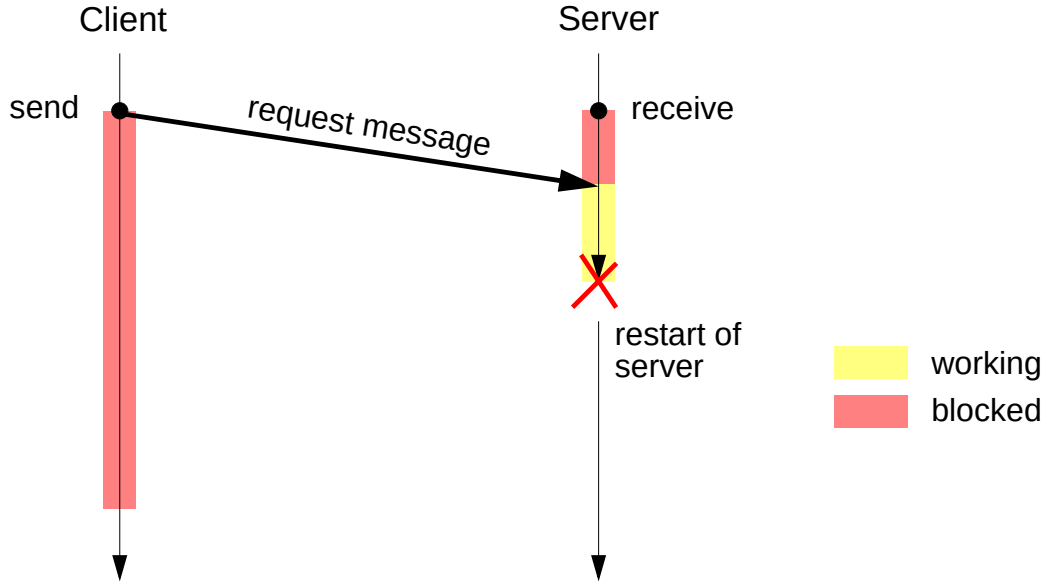
- Bearbeitung ist noch nicht fertig
  - wiederholte Anfrage wird ignoriert



Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

3 Fehlerbehandlung - Problem 4

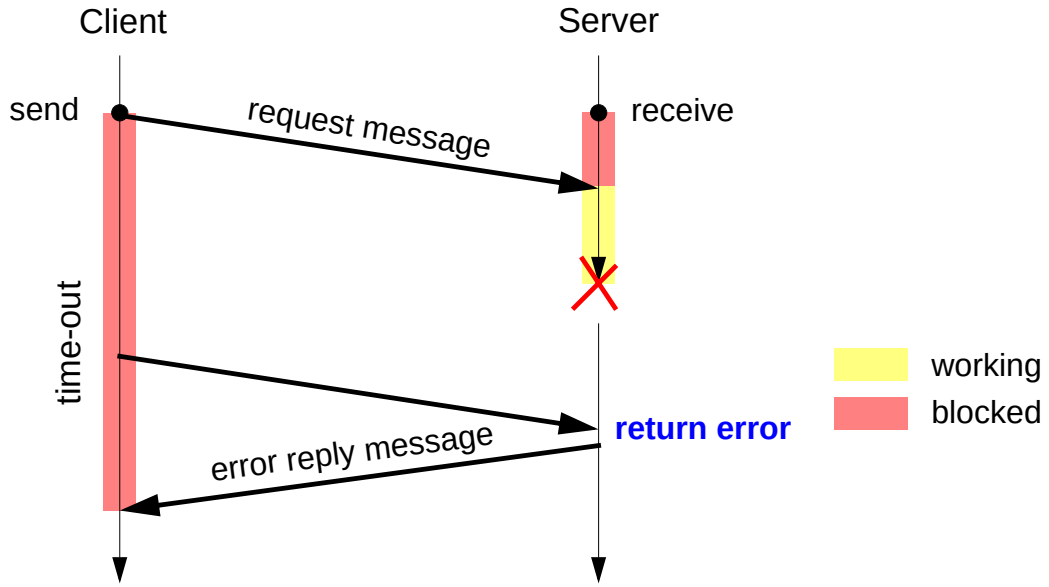
- Serverfehler
  - Fehler während des Ausführens der Methode (Server stürzt ab)



Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

3 Fehlerbehandlung - Lösung 4

- Server stürzt ab:
  - Server erkennt alte Anfragen (alte Generationsnummer) und liefert Fehler



Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

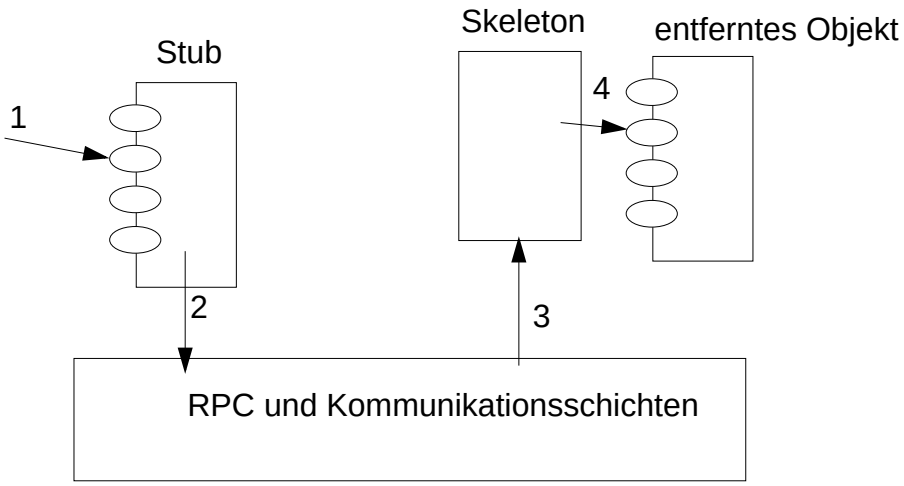
3 Fehlerbehandlung - mögliche Implementierung

- Annäherung an *exactly once*:
  - Um verlorene Anfragenachrichten entgegenzuwirken werden die Pakete nach einem Timeout erneut gesendet.
  - Zum Vermeiden von mehrfachen Ausführungen bei doppelt gesendeten Paketen werden IDs verwendet, welche bei einem erneuten Versenden der gleichen Anfrage nicht verändert werden.
  - Der Server soll abgesendete Antworten puffern, für den Fall, dass eine Antwort verloren geht und um eine erneute Ausführung der Methode zu vermeiden.
  - Den Absturz eines Servers zu verkraften ist schwieriger .....
    ..... daher ignorieren wir das Problem.

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

4 Stubs und Skeletons

- Stub: Stellvertreter (Proxy) des entfernten Objekts.
- Skeleton: Ruft die Methoden am entfernten Objekt auf



MW - Übung

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

4 Stub

- implementiert den gleichen Typen wie das entfernte Objekt (gleiches Interface)
- verpackt einen Methodenaufruf in ein Request-Objekt:
  - Objekt ID, Methoden ID, Parameter
- verwendet die RPC-Schicht um eine Anforderung zu versenden
- transformiert das Rückgabeobjekt in den entsprechenden Typ

MW - Übung

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

4 Stub Beispiel

```
package test;
import orb.*;
public class AccountImpl_Stub implements test.Account {
    int oid;
    ClientContext ctx;
    public AccountImpl_Stub(int oid, ClientContext ctx) {
        this.oid = oid;
        this.ctx = ctx;
    }

    // erzeugte Methoden (Bsp ohne Exceptionbehandlung)
    public int deposit(int param0) {
        Object[] parameters = new Object[1];
        parameters[0] = new Integer(param0);
        Request req = new Request(ctx, oid, 9, parameters);
        Object ret = req.send();
        return ((Integer)ret).intValue();
    }
    // weitere erzeugte Methoden ...
}
```


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

4 Skeleton

- ruft Methoden am "echten" Objekt auf
- notwendige Informationen:
  - Objektreferenz
  - Methoden ID
  - Parameter

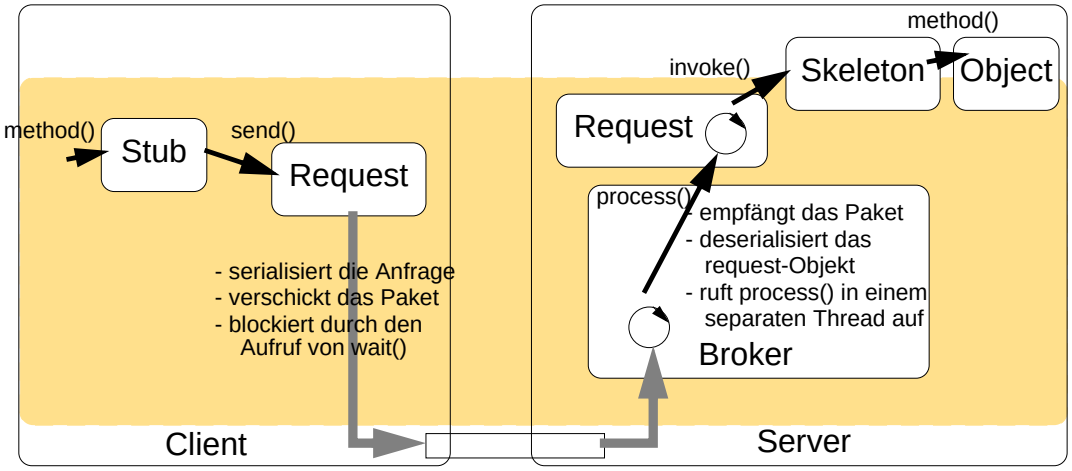
Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

4 Skeleton Beispiel

```
package test;
import orb.*;
public class AccountImpl_Skel  implements Skeleton {
    AccountImpl  object;
    public AccountImpl_Skel() { /* ... */ }
    public void init(int oid, Object object) {
        this.oid = oid;
        this.object = (AccountImpl) object;
    }
    public Object invoke(int mid, Object[] parameters)
                                              throws Exception {
        switch(mid) {
            ...
            case 9: {
                int p1 = ((Integer)parameters[0]).intValue();
                int result = object.deposit(p1);
                return new Integer(result);
            }
    }
}
```

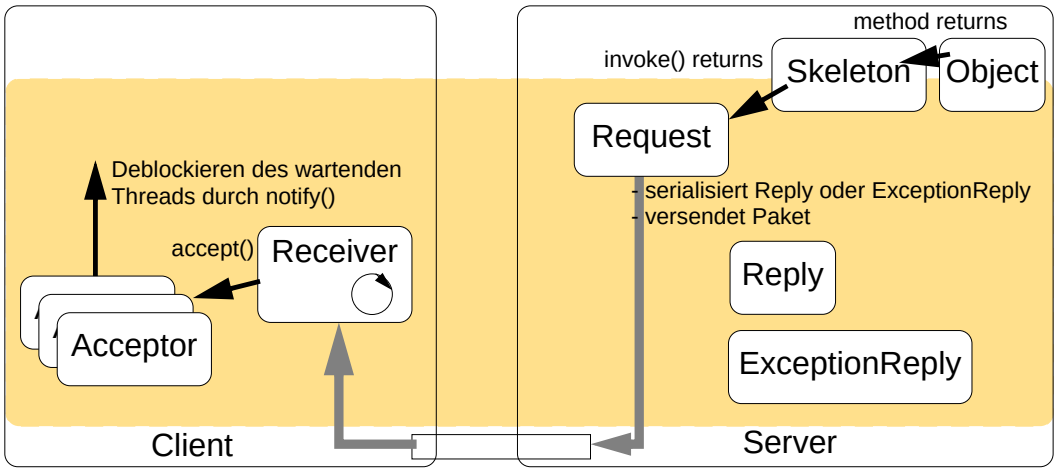

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

4 Request



Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

4 Reply



Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

5 Object Store

- Interface:
  - **`int registerObject(Object obj)`**
    gibt eine eindeutige Objekt-ID zurück.
  - **`Object lookupByID(int oid)`**
    liefert das Objekt zur gegebenen Objekt-ID
- Implementationstechnik:
  - **`java.util.Hashtable`** kann verwendet werden um die Abbildung zwischen Objekt-IDs und Objekten vorzunehmen (Hinweis: die Hashtabelle kann nur Objekte speichern, die *OID* sollte daher in ein **`Integer`**-Objekt gekapselt werden.)

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

6 Nameserver

- ist aufgeteilt in einen ID-Finder und einen Proxy-Erzeuger
- ID Finder
  - bildet Namen auf OIDs ab:
    - `void bind(String name, int oid)`
    - `int lookupID(String name)`
  - verwendet `Hashtable`
  - ist selbst als entferntes Objekt mit der OID 1 implementiert
- ein realistischer Nameserver würde Referenzen auf Stub-Objekte zurückliefern anstatt OIDs
- das erfordert, dass zusammen mit der OID der Klassenname des Stubs zurückgegeben wird.

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

7 Nameserver

- Stub-Erzeuger
  - sucht die ID (und den Klassennamen); erzeugt daraufhin ein Stub-Objekt
    - `Object lookup(String name)`
  - der Nameserver lädt die Klasse und erzeugt eine Instanz der Klasse

Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.