

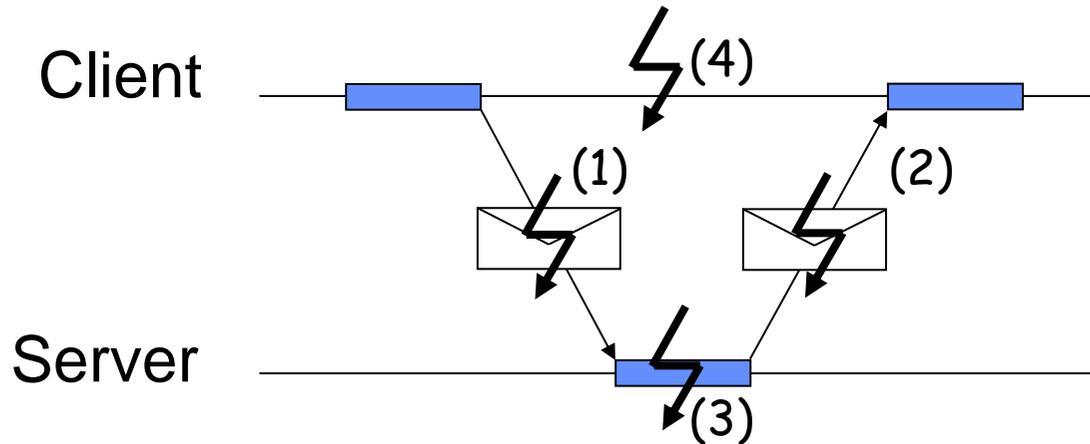
Verteilte Systeme

Vorlesung 3
Dr. Sebastian Iwanowski
FH Wedel

Verteilte Systeme

1. Innovative Beispiele aus der Praxis
2. Allgemeine Anforderungen und Techniken verteilter Systeme
- ➔ 3. Die Client-Server-Beziehung und daraus entstehende Fragestellungen
4. Nebenläufigkeitstechniken in Java
5. Entfernte Aufrufe
6. Objektmigration
7. Agententechnologie
8. Dienstevermittlung
9. Synchronisation von Daten
10. Konzepte zur Erzielung von Fehlertoleranz
11. Web Services

Probleme in der Client-Server-Kommunikation



Client wartet und versucht...

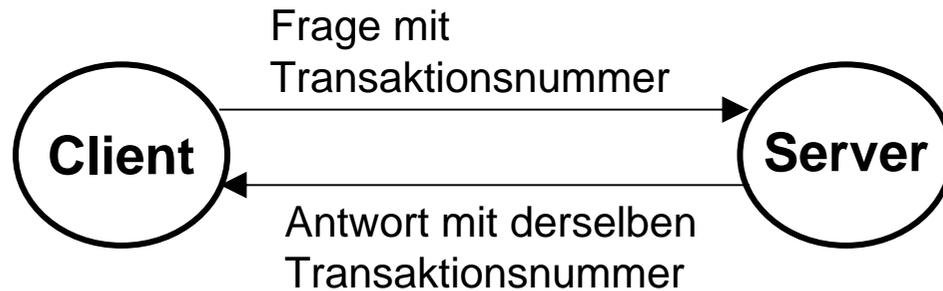
... nach Timeout ein erneutes Senden,

kann aber nicht zwischen verschiedenen Fehlersituationen unterscheiden.

Erneutes Senden führt zur erneuten Ausführung.

Problem: Wie erkennt der Server, dass der Client dieselbe Anfrage noch einmal gestellt hat und nicht eine neue gestellt hat ?

Lösungsansatz: Protokolle / Transaktionskonzept



wichtig:

- Transaktionsnummer ist im ganzen Netzwerk eindeutig !

Was ist eine Transaktion ?

Eine **Transaktion** ist eine Folge von Operationen, die entweder alle vollständig oder alle überhaupt nicht durchgeführt werden sollen.

Was ist ein Protokoll ?

Ein **Protokoll** ist ein Regelwerk für Kommunikationsaktionen. Es legt die Abfolge der Aktionen und die zu benutzenden Formate fest, unterscheidet die Rollen der beteiligten Kommunikationspartner und legt eventuell weitere logische Zusammenhänge fest.

Lösungsansatz: Protokolle / Transaktionskonzept

Warum brauchen wir Transaktionen ?

Bsp. Bankkonto: Umbuchung eines Betrages von Konto A nach Konto B

geplant:

Umbuchung

```
read (A, a1)
a1 := a1 - 300
write (A, a1)
read (B, b1)
b1 := b1 + 300
write (B, b1)
```

tatsächlicher Verlauf:

Umbuchung

```
read (A, a1)
a1 := a1 - 300
write (A, a1)
read (B, b1)
```

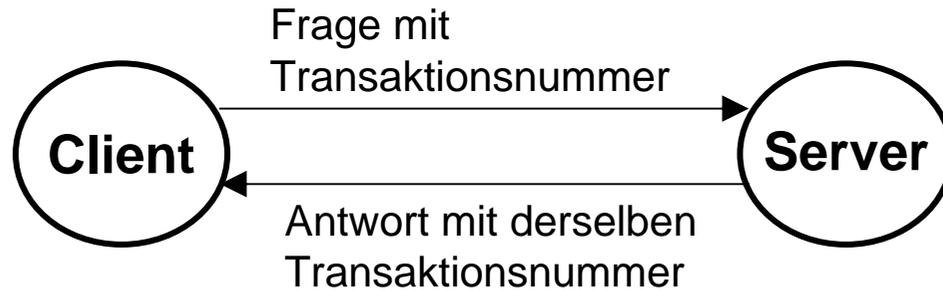
Störung 

Wo sind die 300 € geblieben?

Warum brauchen wir Protokolle ?

- zur effizienten Beschreibung eines gewünschten Kommunikationsschemas
- weitere Beispiele später

Lösungsansatz: Protokolle / Transaktionskonzept



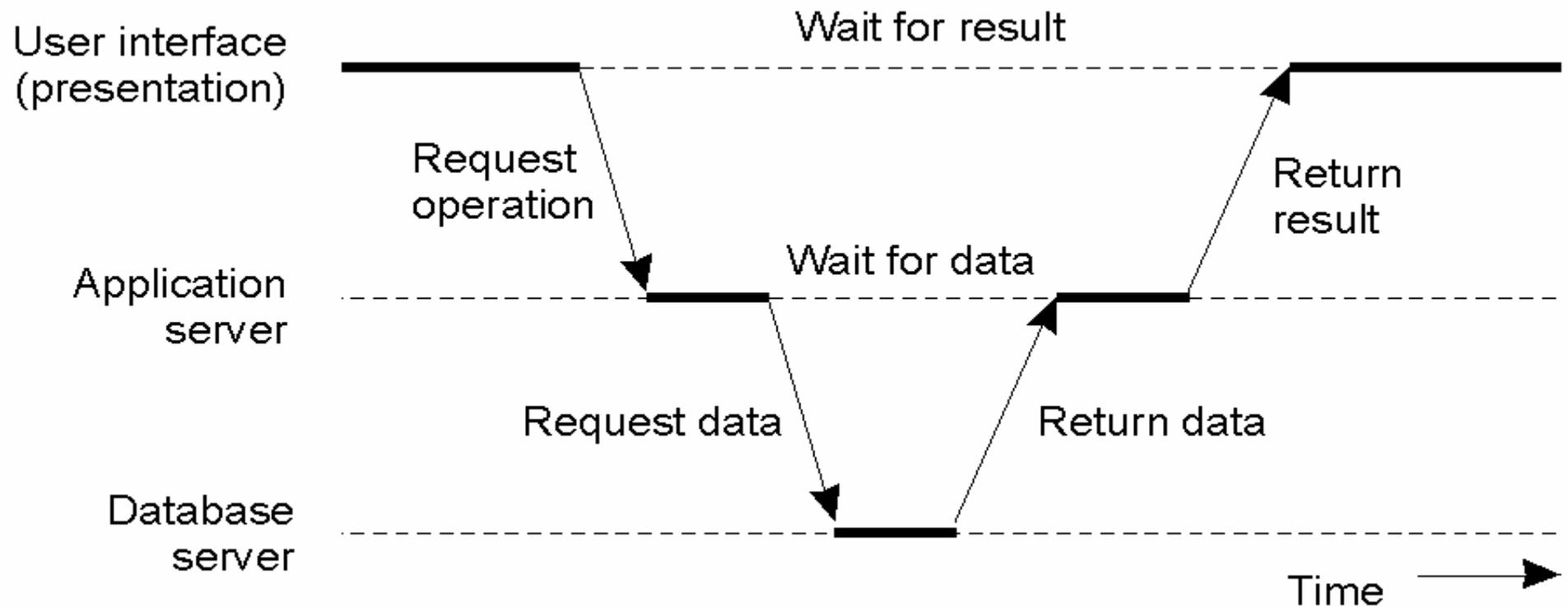
wichtig:

• Transaktionsnummer ist im ganzen Netzwerk eindeutig !

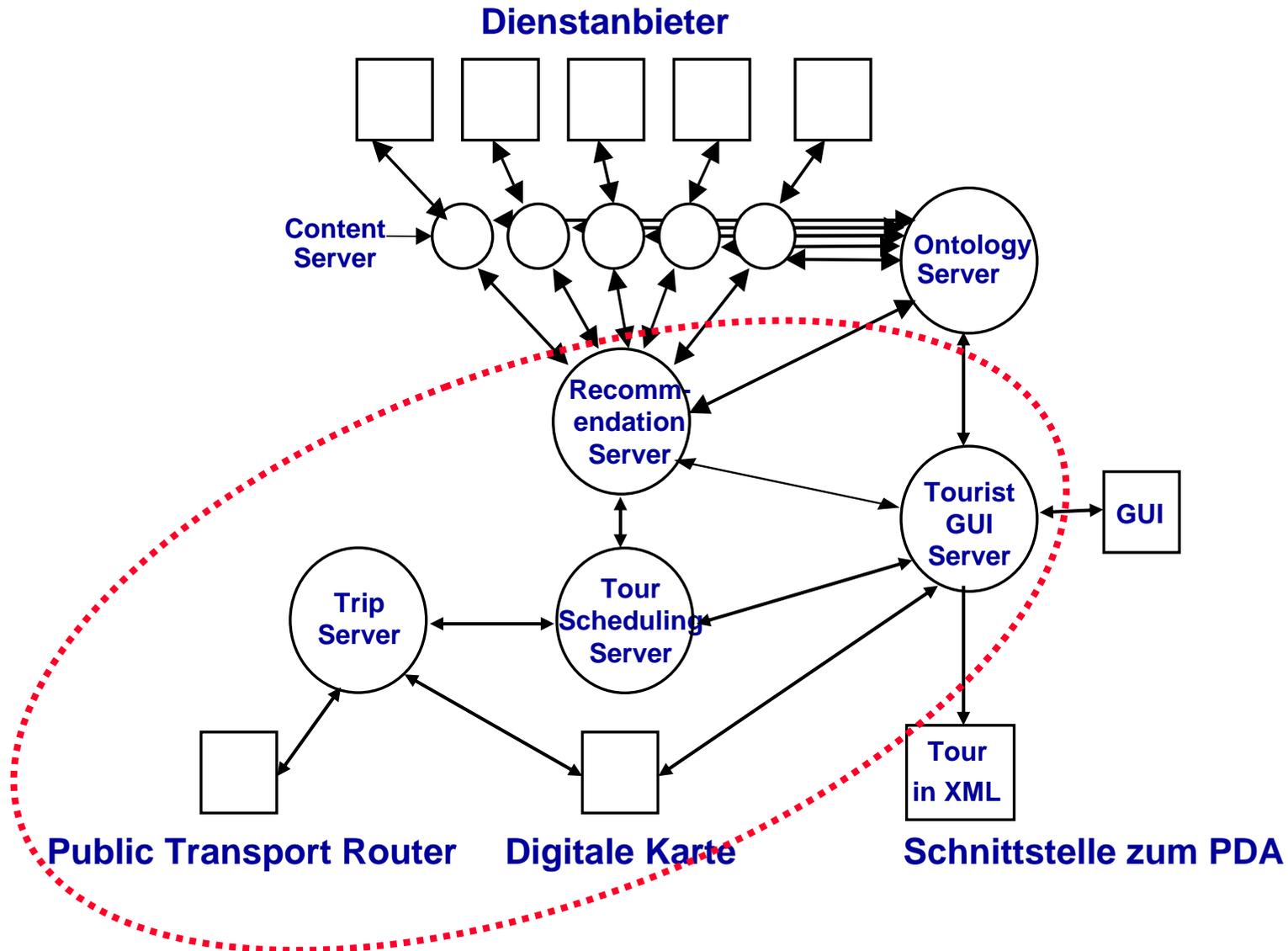
- Die Transaktionen werden innerhalb eines Protokolls ausgeführt
- Jede Operation derselben Transaktion hat dieselbe Transaktionsnummer und eine Ausführungsnummer, die seine Stellung innerhalb des Protokolls beschreibt
- Wenn eine Transaktion nicht ordnungsgemäß zu Ende geführt wurde, werden alle Operationen dieser Transaktion rückgängig gemacht.

Mehrschichten-Architektur (Multitiered architecture)

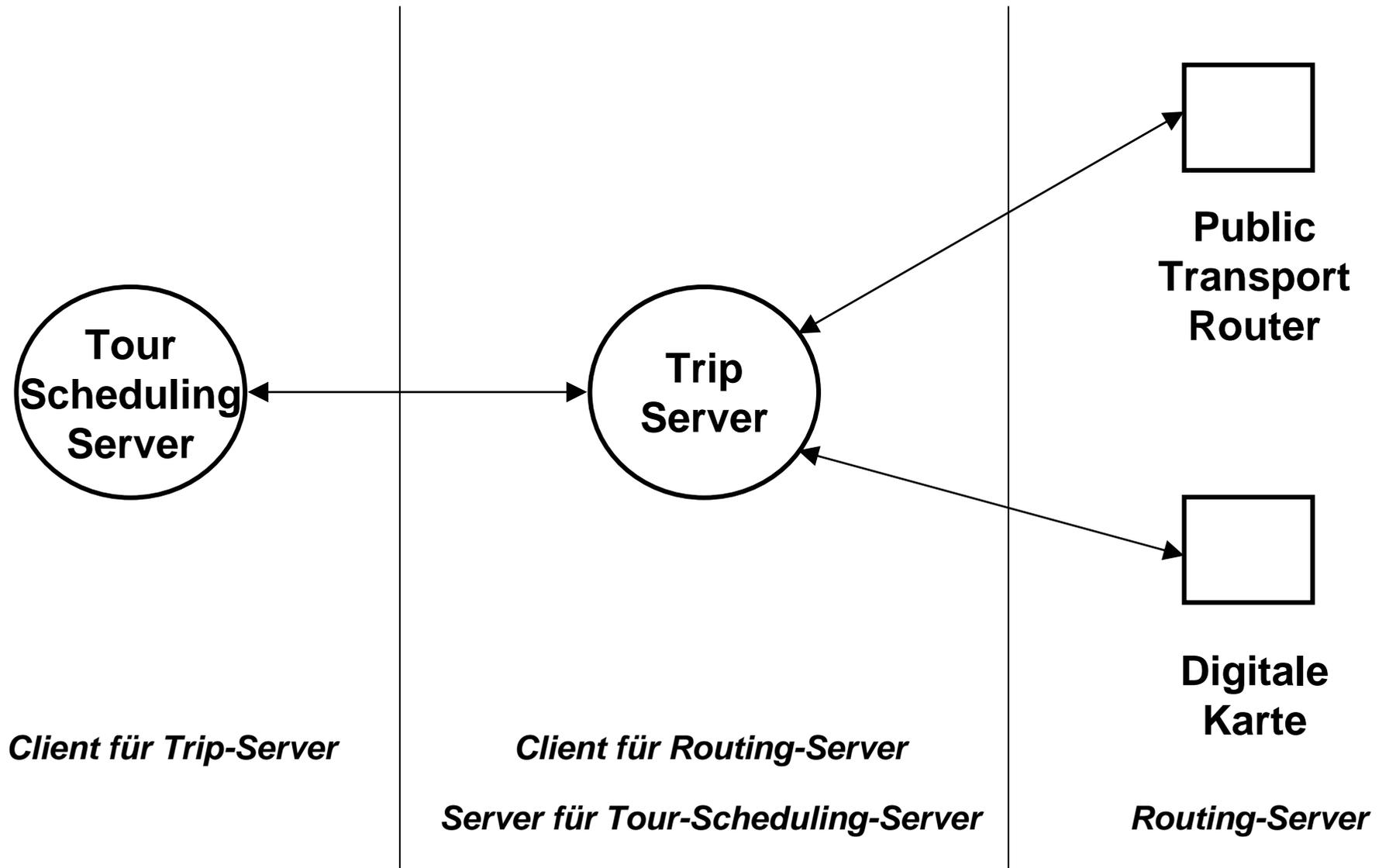
Beispiel für 3 Schichten:



Beispiel: Touristeninformationssystem



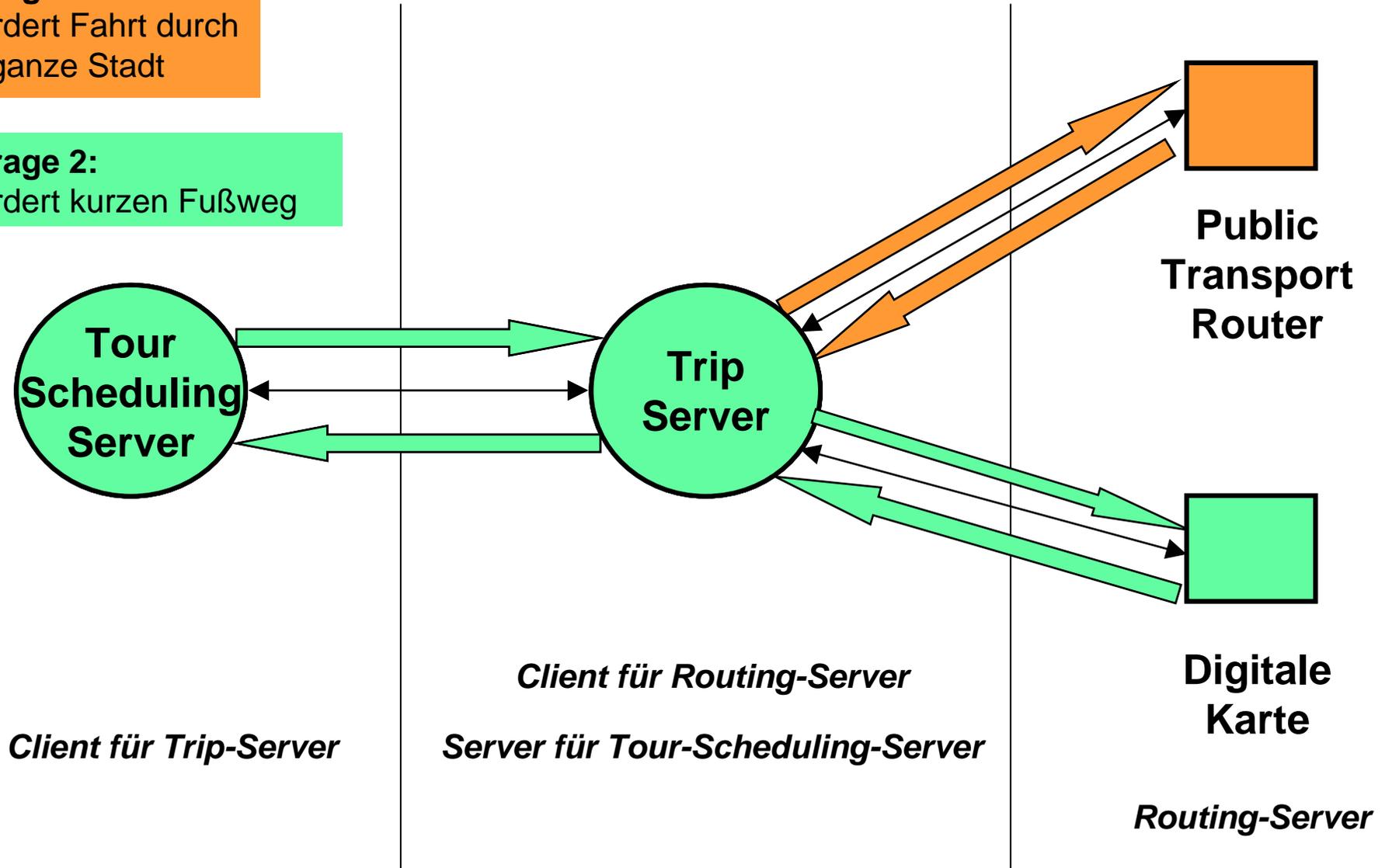
Bsp. für mehrere Schichten im Touristeninformationssystem



Bsp. für die Blockierung eines Prozesses

Anfrage 1:
erfordert Fahrt durch
die ganze Stadt

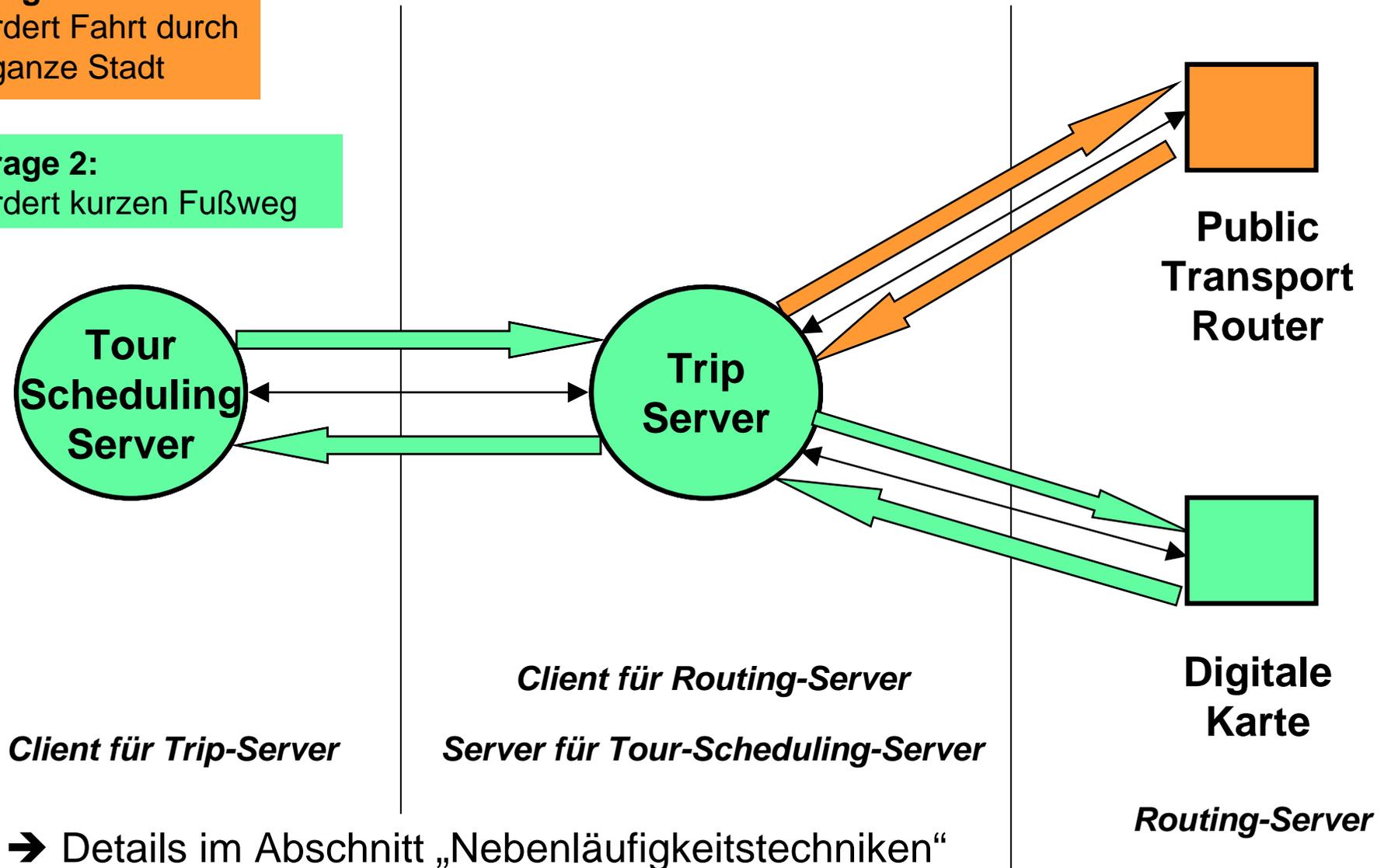
Anfrage 2:
erfordert kurzen Fußweg



Bsp. für die Aufhebung der Blockade durch Multithreading

Anfrage 1:
erfordert Fahrt durch
die ganze Stadt

Anfrage 2:
erfordert kurzen Fußweg



Weitere Schwierigkeiten beim Touristeninformationssystem

Client und Server befinden sich an unterschiedlichen Orten

→ Lösungen im Abschnitt „Entfernte Aufrufe“

Heterogene Datenwelt in unterschiedlichen Servern

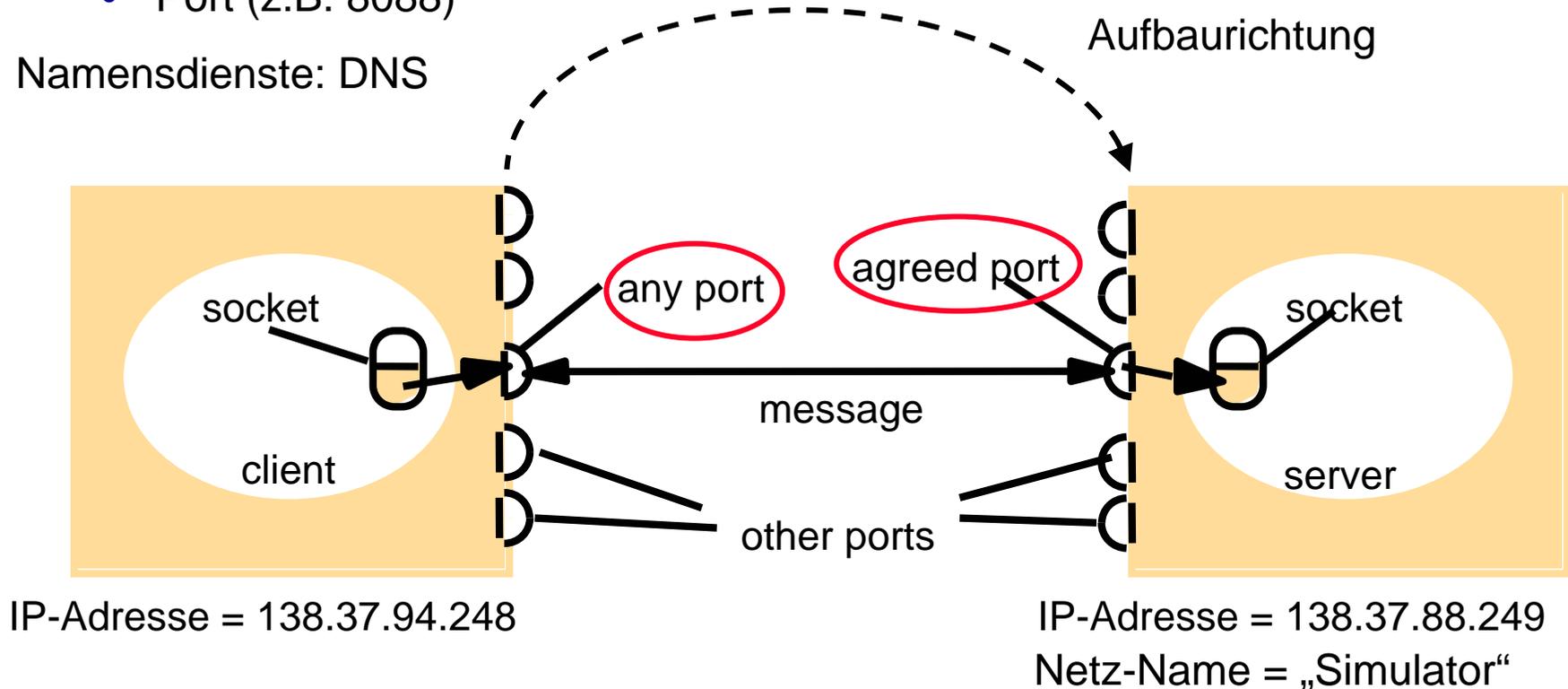
→ Lösungen im Abschnitt „Dienstevermittlung“

Die Benutzung einer TCP / IP – Verbindung („Socket-Schnittstelle“)

Adressierung eines Knotens (Computer)

- IP-Adresse (z.B. 134.100.12.135) oder Name
- Port (z.B. 8088)

Namensdienste: DNS



Die Benutzung einer TCP / IP – Verbindung („Socket-Schnittstelle“)

Sockets sind prozessspezifisch:

Gleichzeitig kann nur ein Prozess ein Socket benutzen

**Jeder Prozess darf mit mehreren Sockets in
Verbindung stehen**

Die Benutzung einer TCP / IP – Verbindung („Socket-Schnittstelle“)

Welches Datenformat sollte gewählt werden ?

Antwort abhängig von Homogenität der Partner !

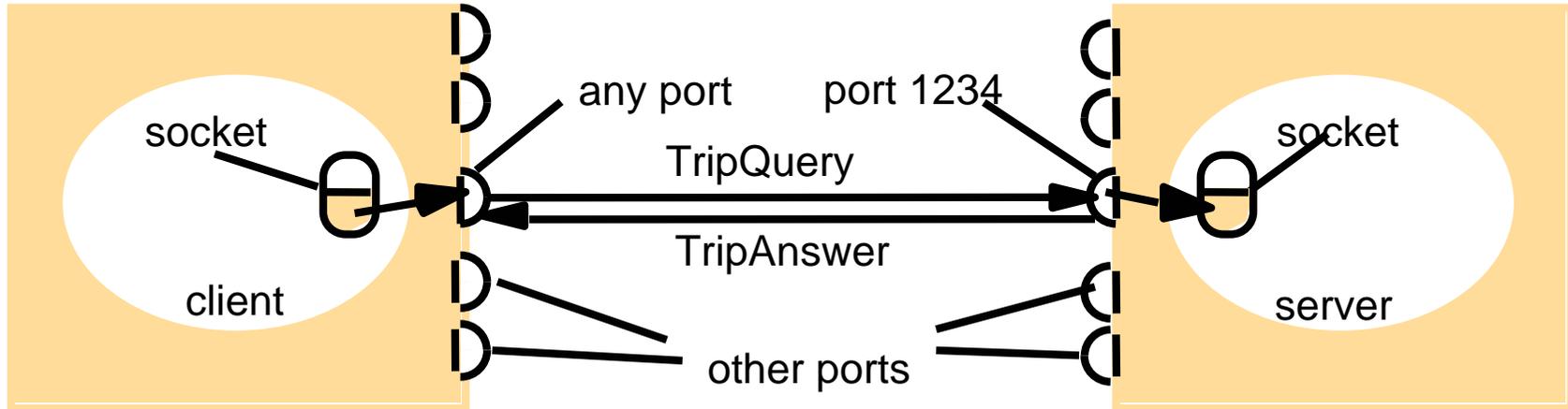
Alle Beteiligten sollten das gleiche Verständnis des Datenformats haben !

Bei beliebigen Partnern: ASCII-Zeichenketten

**Für Partner aus der selben Programmierwelt:
Spezifischere Objekte**

→ Java bietet vielfältige Möglichkeiten

Die Benutzung einer TCP / IP – Verbindung



IP-Adresse = 138.37.94.248
Netz-Name = „TripServer“

IP-Adresse = 138.37.88.249
Netz-Name = „HVVServer“

Gemeinsamkeiten aller TCP/IP-Realisierungen:

Einmalige Anmeldung an definierten Port erforderlich

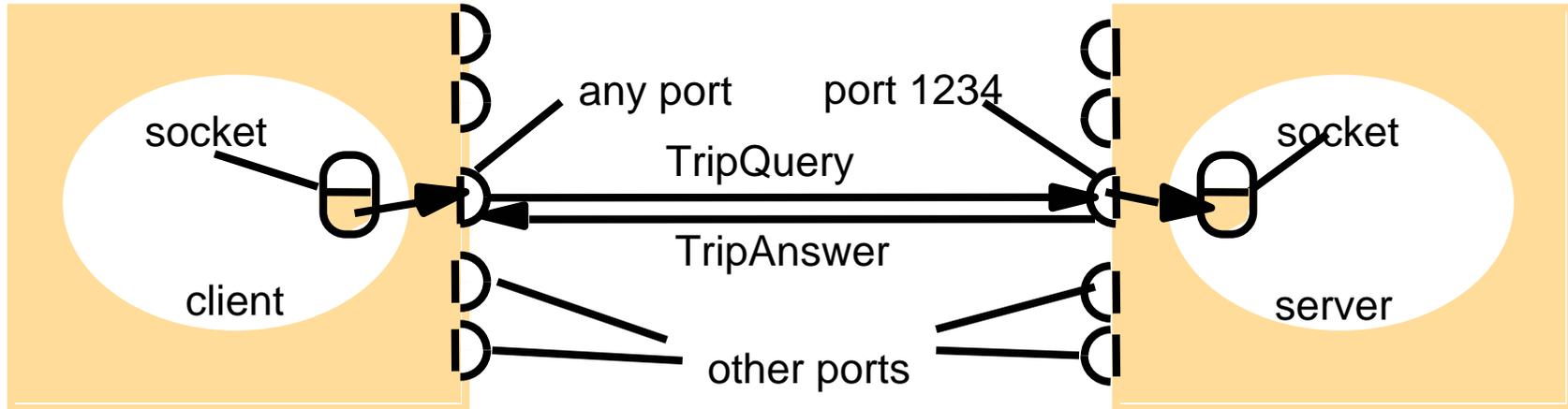
Wiederholte Datenübertragung in beiden Richtungen möglich

Realisierung in Java:

Socketeinrichtung über Classes Socket und ServerSocket

Übertragung von Daten über Streams

Java: Aufbau einer TCP/IP-Verbindung



IP-Adresse = 138.37.94.248
Netz-Name = „TripServer“

IP-Adresse = 138.37.88.249
Netz-Name = „HVVServer“

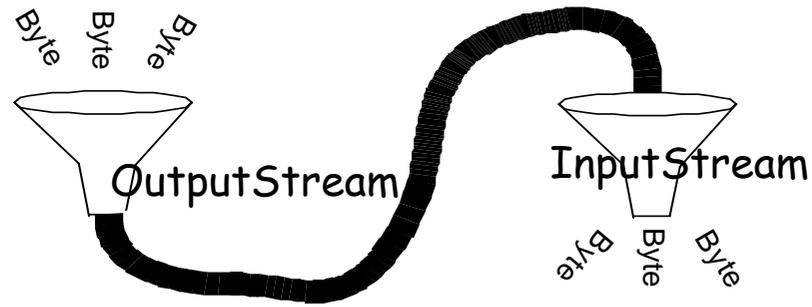
Im Client:

```
Socket server
    = new Socket („HVVServer“, 1234);
System.out.println
    ("Connected to " +
    server.getInetAddress());
```

Im Server:

```
int port = 1234;
ServerSocket server = new
    ServerSocket(port);
while (true) {
    Socket client = server.accept();
    System.out.println ("Client " +
        client.getInetAddress() +
        "connected."); }
```

Java: Datenübertragung über Bytes



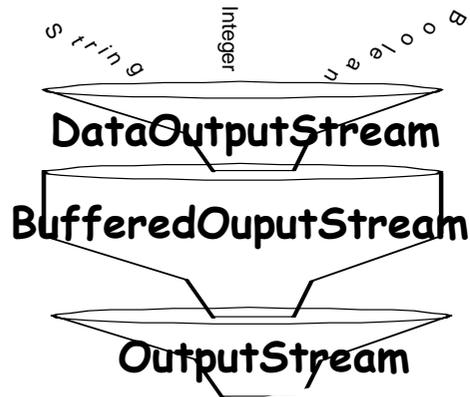
Senden:

```
Socket socket;  
    // muss initialisiert werden  
OutputStream out =  
    socket.getOutputStream();  
Object obj = new Object ();  
byte b[] = obj.getBytes();  
    /* getBytes() muss vom Objekt  
    implementiert werden */  
out.write(b);
```

Empfangen:

```
Socket socket;  
    // muss initialisiert werden  
InputStream in =  
    socket.getInputStream();  
byte b[] = new byte[100];  
    // 100 muss groß genug sein  
int num = in.read(b);  
Object obj = new Object(b);  
    /* new Object(byte[]) muss vom  
    Objekt implementiert werden */
```

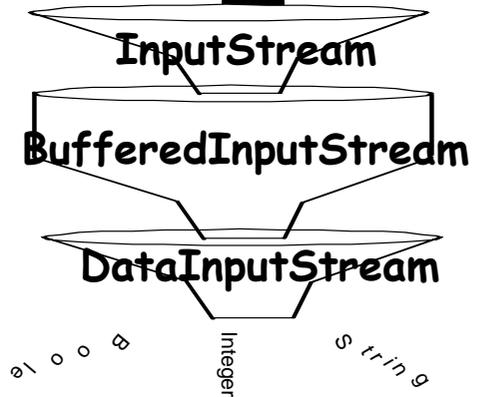
Java: Datenübertragung über String-Filter



Senden:

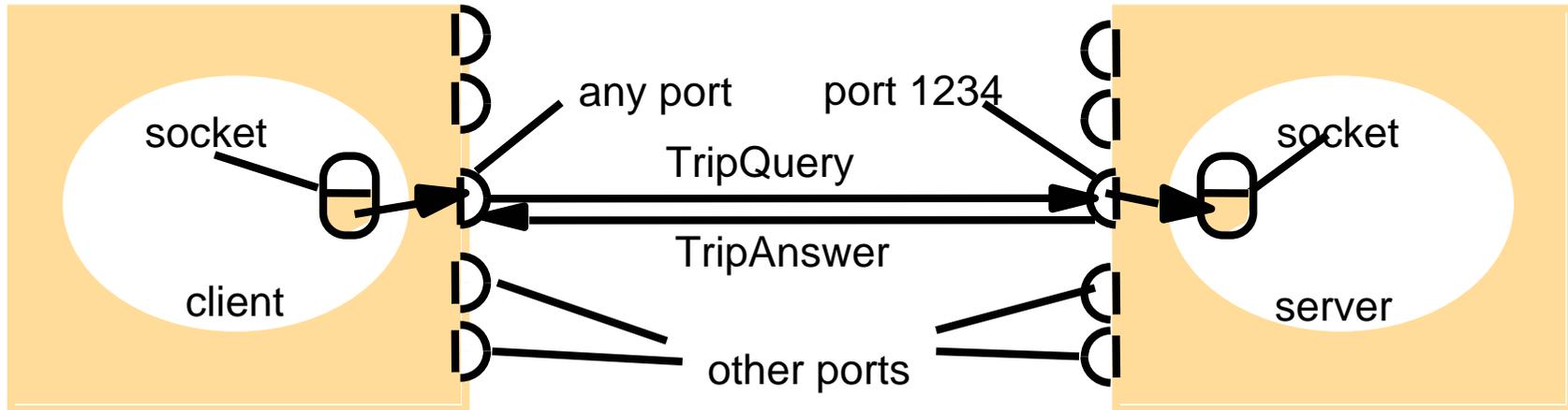
```
Socket socket;  
    // muss initialisiert werden  
DataOutputStream out = new  
    DataOutputStream(new  
    BufferedOutputStream(  
    socket.getOutputStream()));  
Object obj = new Object;  
out.writeUTF(obj.toString());  
out.flush();
```

Empfangen:



```
Socket socket;  
    // muss initialisiert werden  
DataInputStream in = new  
    DataInputStream(new  
    BufferedInputStream(  
    socket.getInputStream()));  
String str = in.readUTF();  
Object obj = new Object (str)  
    /* new Object(String) muss vom  
    Objekt implementiert werden */
```

Die Benutzung einer TCP / IP – Verbindung



IP-Adresse = 138.37.94.248
Netz-Name = „TripServer“

IP-Adresse = 138.37.88.249
Netz-Name = „HVVServer“

Offene Fragen:

Wie verknüpft man in Client und Server Frage und Antwort ?

→ durch Entfernte Aufrufe

Wie verhindert man die Blockade von Client und Server ?

→ durch Nebenläufigkeitstechniken

**Beim nächsten Mal:
Nebenläufigkeitstechniken in Java**