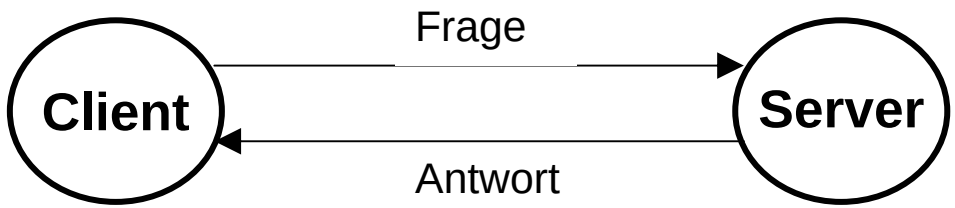
Verteilte Systeme

2. Die Client-Server-Beziehung und daraus resultierende Techniken

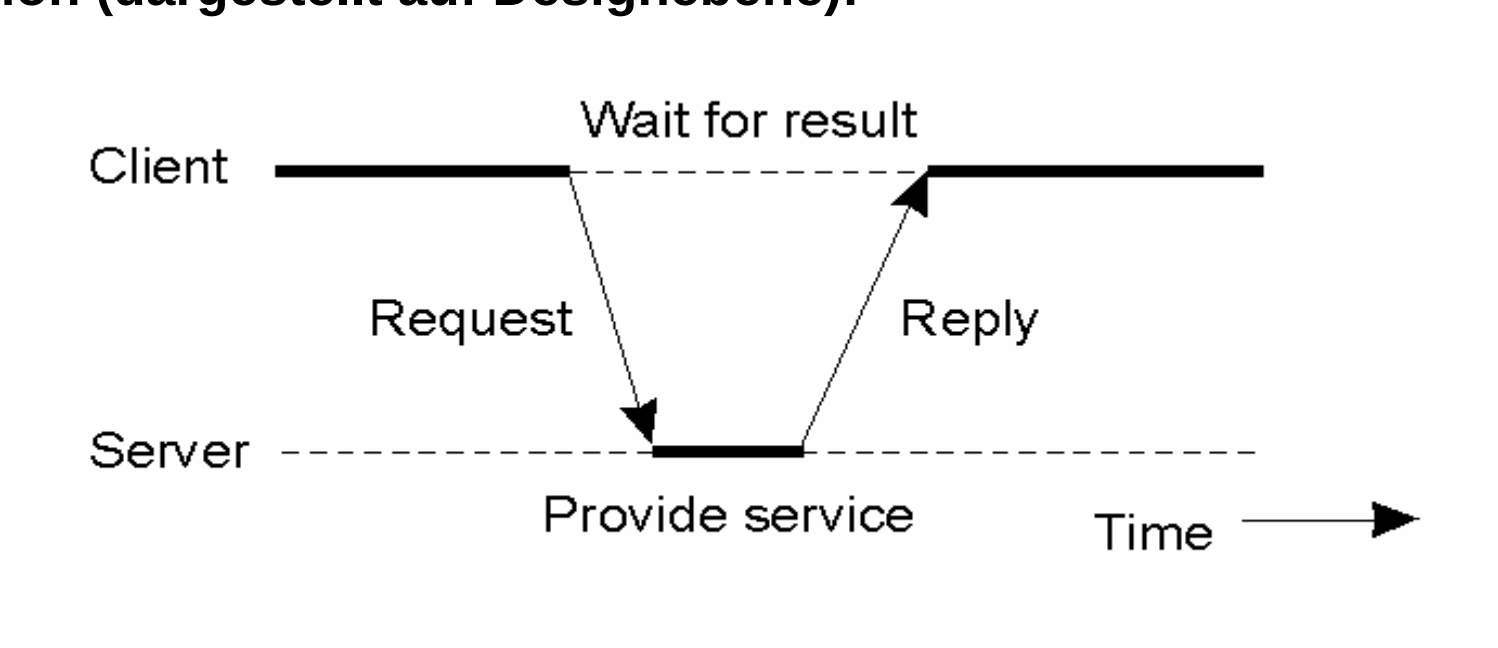
2.1 Grundlagen der Client-Server-Beziehung

Sebastian Iwanowski
FH Wedel

Client-Server-Architektur

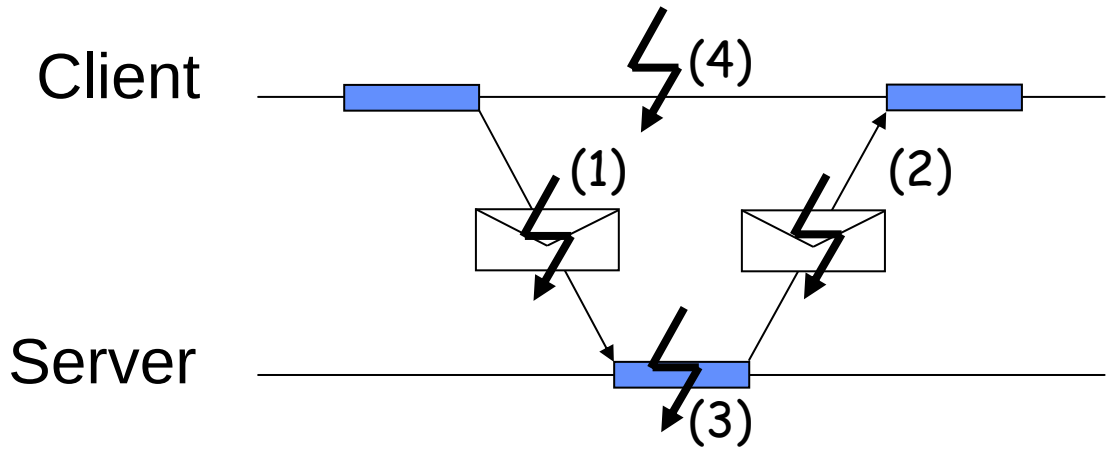


Situation (dargestellt auf Designebene):



Frage: Wie wird das auf Prozessebene realisiert ?

Probleme in der Client-Server-Kommunikation



Fehlermöglichkeiten:

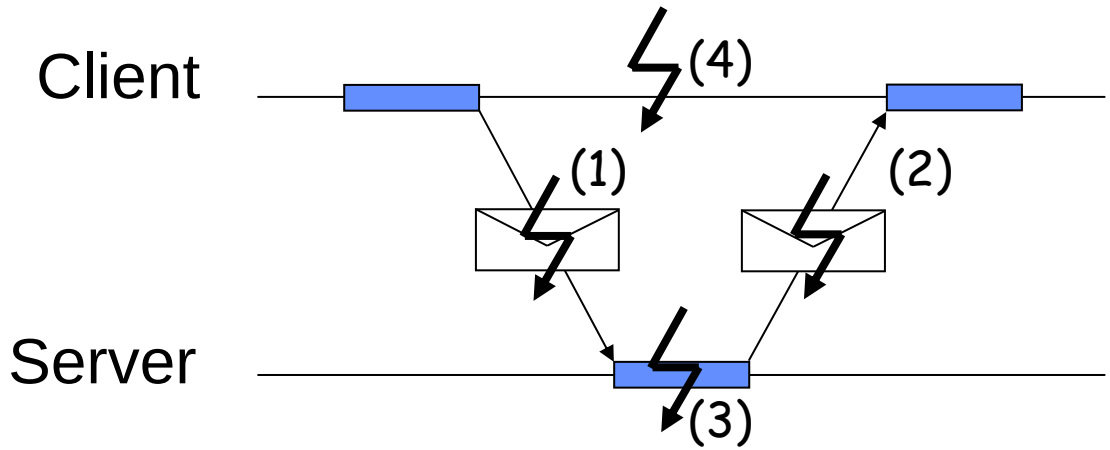
Verlust der Auftragsnachricht (1)

Verlust der Ergebnisnachricht (2)

Ausfall des Servers (3)

Ausfall des Klienten (4)

Probleme in der Client-Server-Kommunikation



Client wartet und versucht...

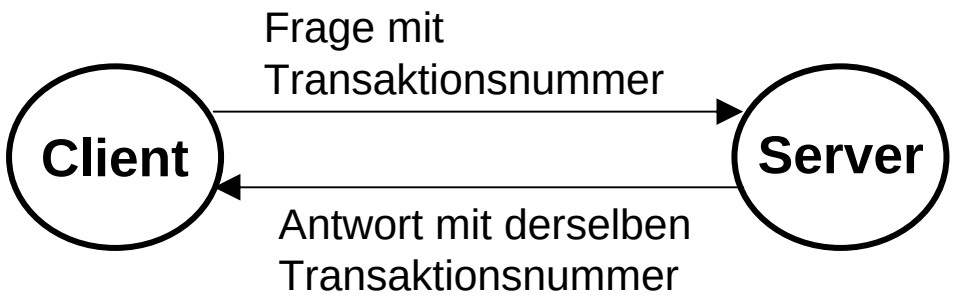
... nach Timeout ein erneutes Senden,

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 !**

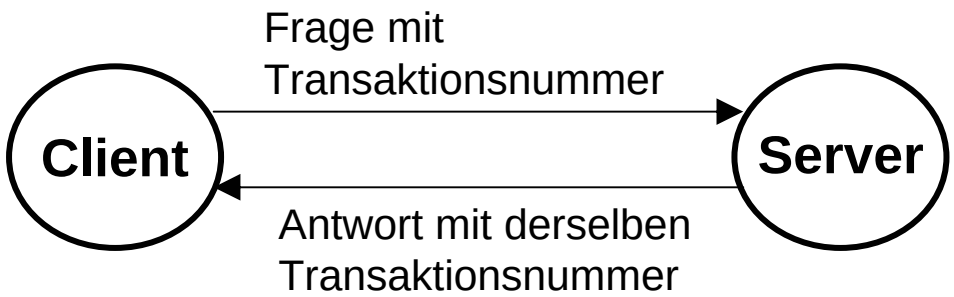
Was ist eine Transaktion ?

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



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.

Lösungsansatz: Protokolle / Transaktionskonzept

Warum brauchen wir Transaktionen und Protokolle?

Bsp. Geldautomat: Abheben eines Barbetrags vom Konto

Störfälle:
1) Aktion wird vorzeitig abgebrochen.
2) Eine Aktion wird doppelt ausgeführt.

Protokolle sind auch für die Kommunikation auf Anwendungsebene notwendig!

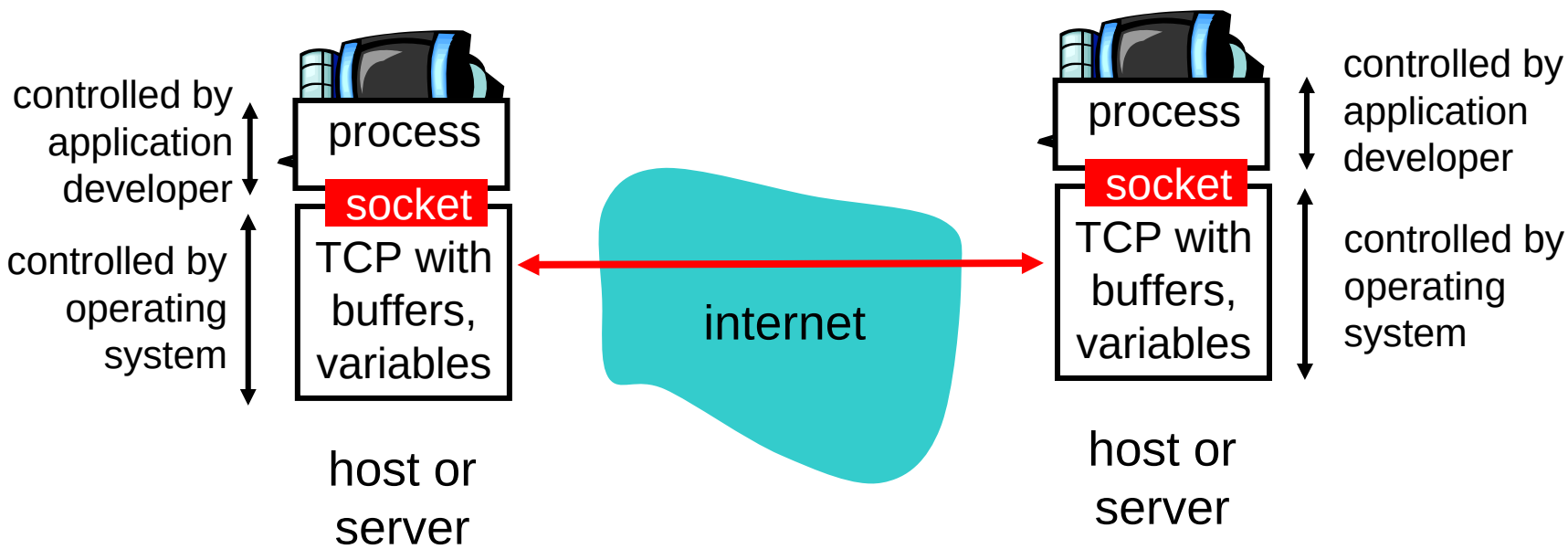
Beispielanwendungen:
1) Auktionsprotokolle
2) Verhandlungsprotokolle
3) Fehlerprotokolle
...

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

TCP als Beispiel für ein Kommunikationsprotokoll zwischen Client und Server

Socket: eine Tür zwischen Programmanwendung und Protokoll der Transportschicht (UDP oder TCP)

TCP-Dienst: zuverlässiger Transport von **bytes** zwischen zwei verschiedenen Prozessen

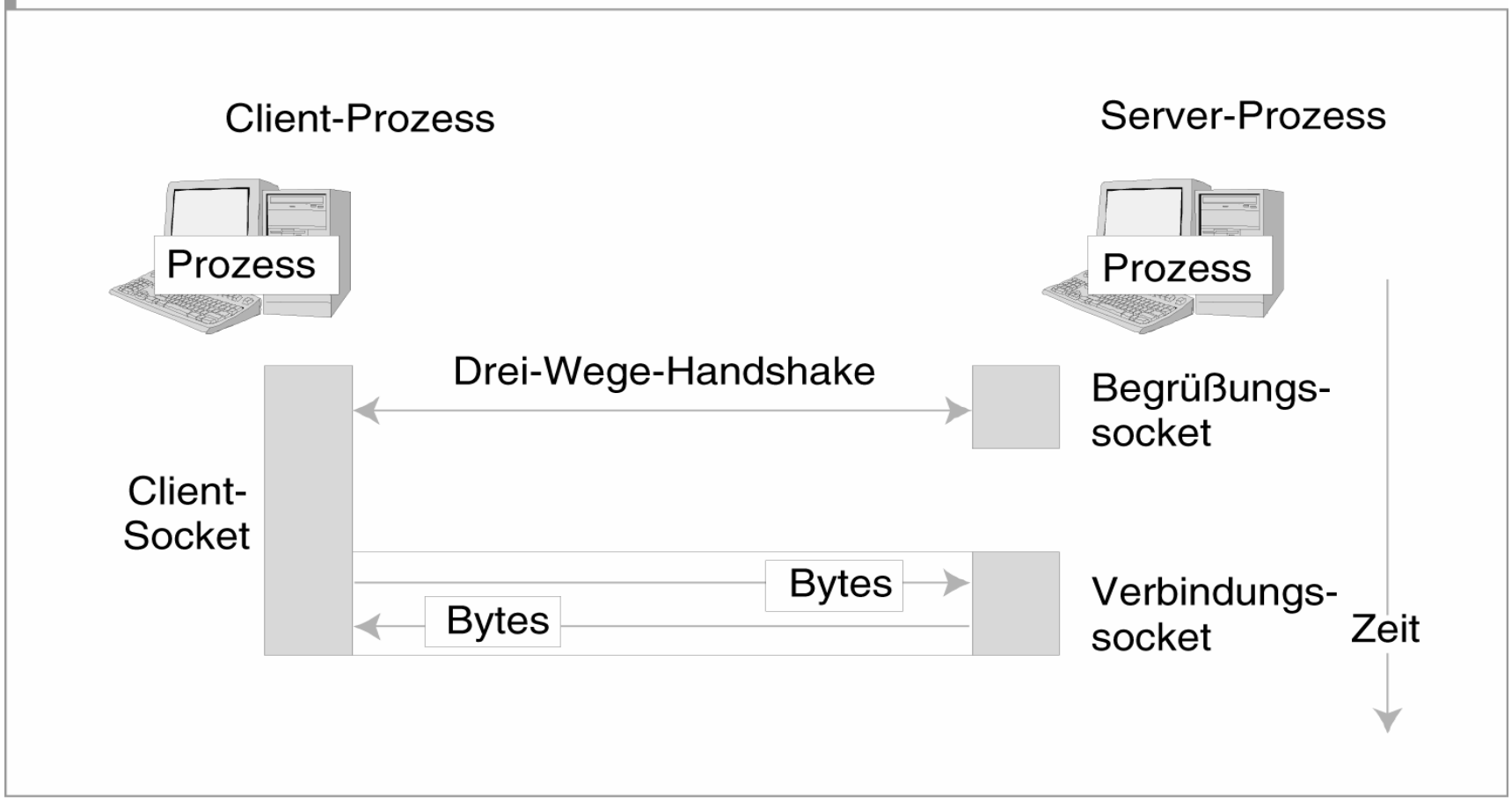


aus: Kurose / Ross: Computer Networking, Kap. 2

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

Kommunikations**mittel** von TCP

Abbildung 2.24 Client-, Begrüßungs- und Verbindungssocket



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

Kommunikations**ablauf** von TCP

1) Client muss Server kontaktieren

Dazu muss der Serverprozess bereits laufen.

Server muss Begrüßungssocket bereits eingerichtet haben.

2) Client veranlasst Server:

Einrichtung eines client-spezifischen Verbindungssockets

Kommunikation von IP-Adresse und Portnummer des Serverprozesses an Client

3) Client richtet selbst ein:

Port zum Verbindungssocket des Servers

Server generiert neuen Prozess für jeden Client:

Erfüllung der Anforderungen des Clients

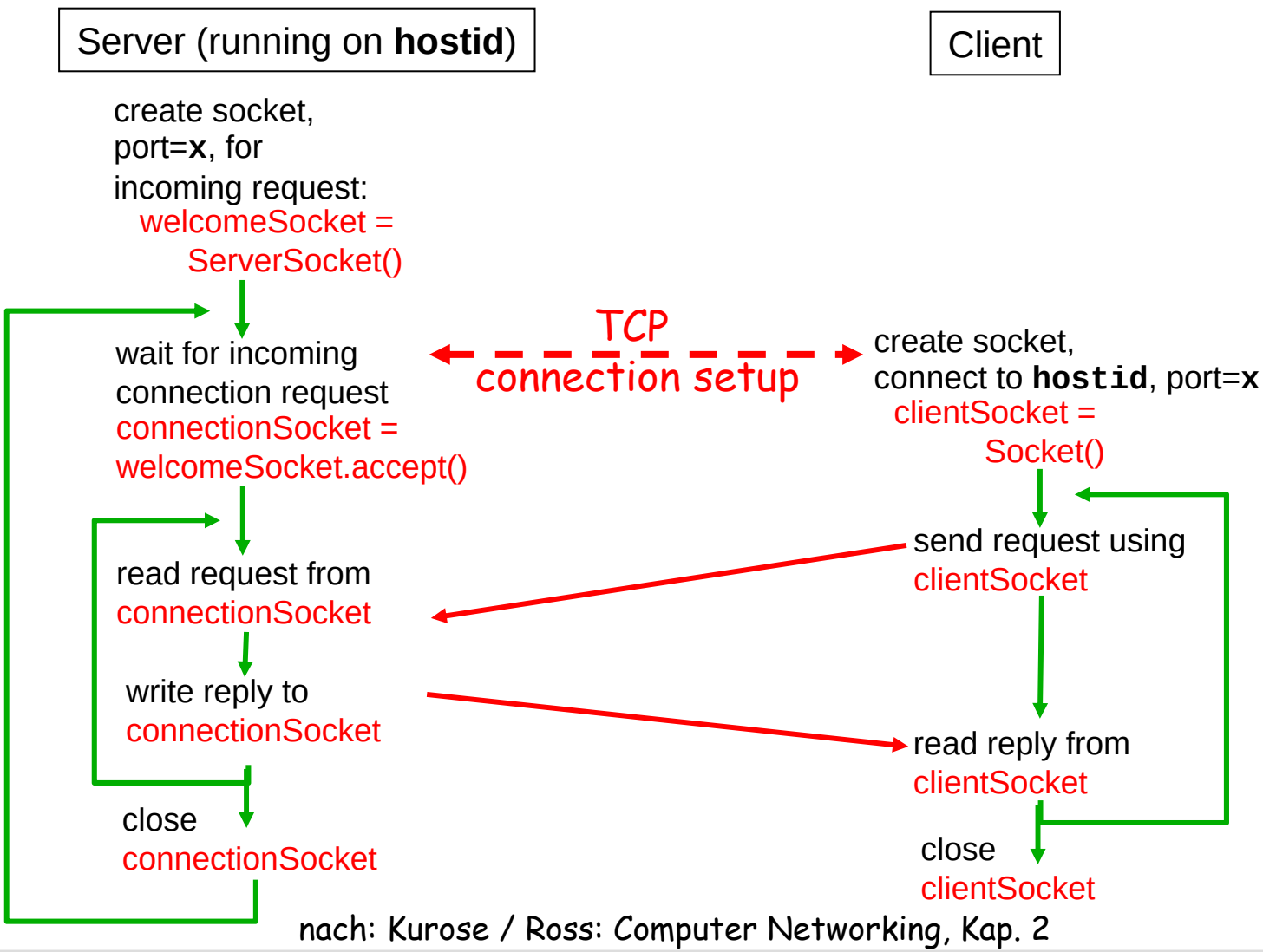
Vorteile:

- Server kann “gleichzeitig” mit mehreren Clients reden.
- Die Clients werden über die eindeutigen Portnummern der Verbindungssockets eindeutig unterschieden.

nach: Kurose / Ross: Computer Networking, Kap. 2

Die Benutzung einer TCP / IP – Verbindung

Kommunikations**ablauf** von TCP



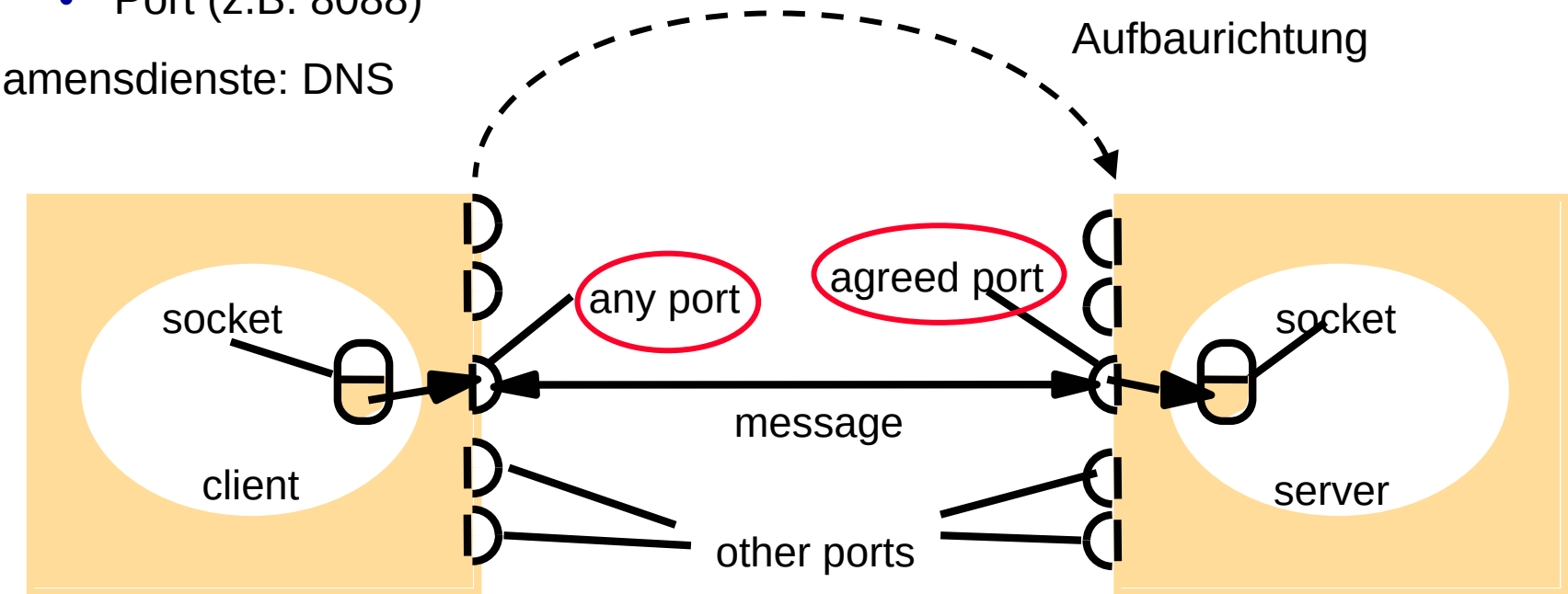
nach: Kurose / Ross: Computer Networking, Kap. 2

Die Benutzung einer TCP / IP – Verbindung

Adressierung eines Knotens (Computer)

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

Namensdienste: DNS



IP-Adresse = 138.37.94.248

IP-Adresse = 138.37.88.249

Netz-Name = „Simulator“

Verbindungssockets 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

Welches Datenformat sollte gewählt werden ?

Antwort abhängig von Homogenität der Partner !

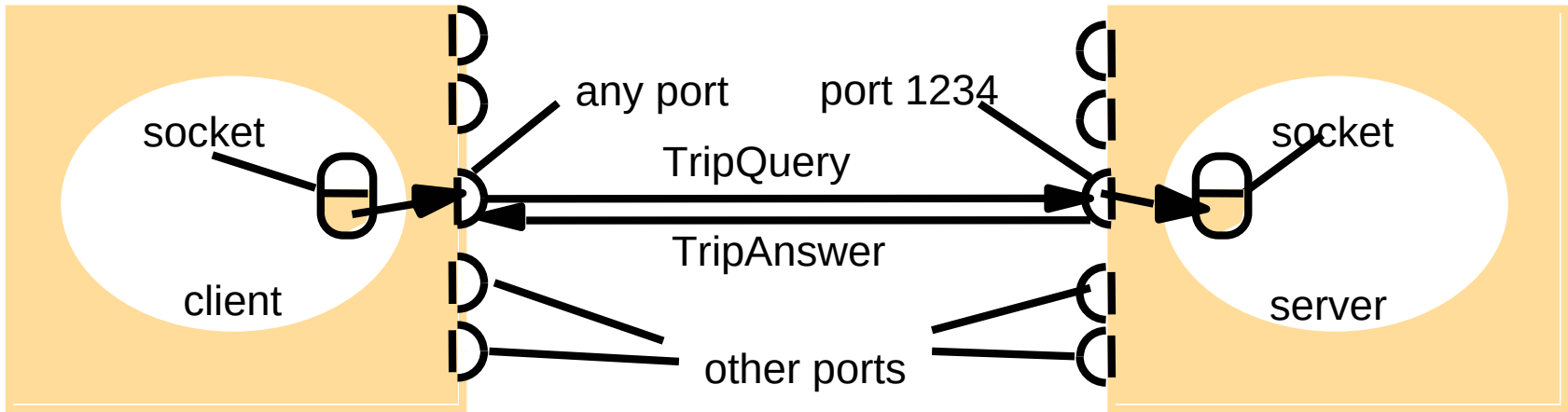
Alle Beteiligten sollten das gleiche Verständnis des Datenformats haben !

Bei beliebigen Partnern: ASCII-Zeichenketten

Für Partner aus derselben 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

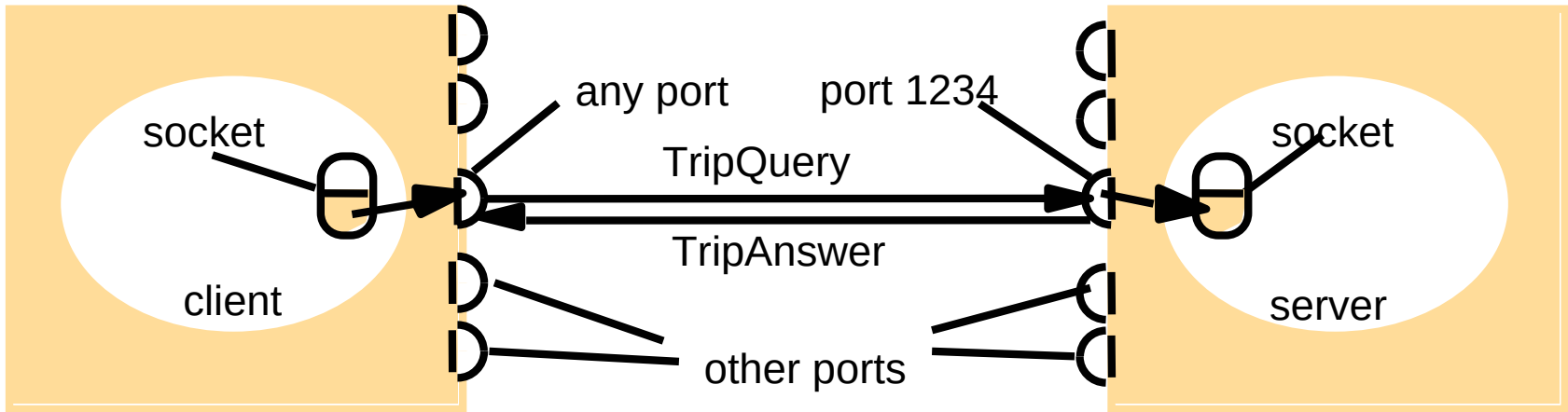
Wiederholte Datenübertragung in beiden Richtungen möglich

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“

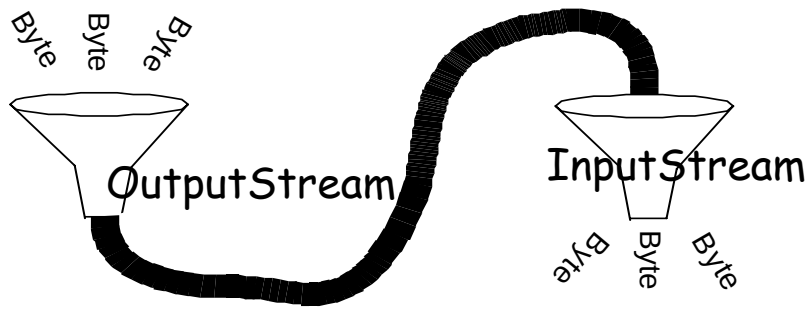
Im Client:

```
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



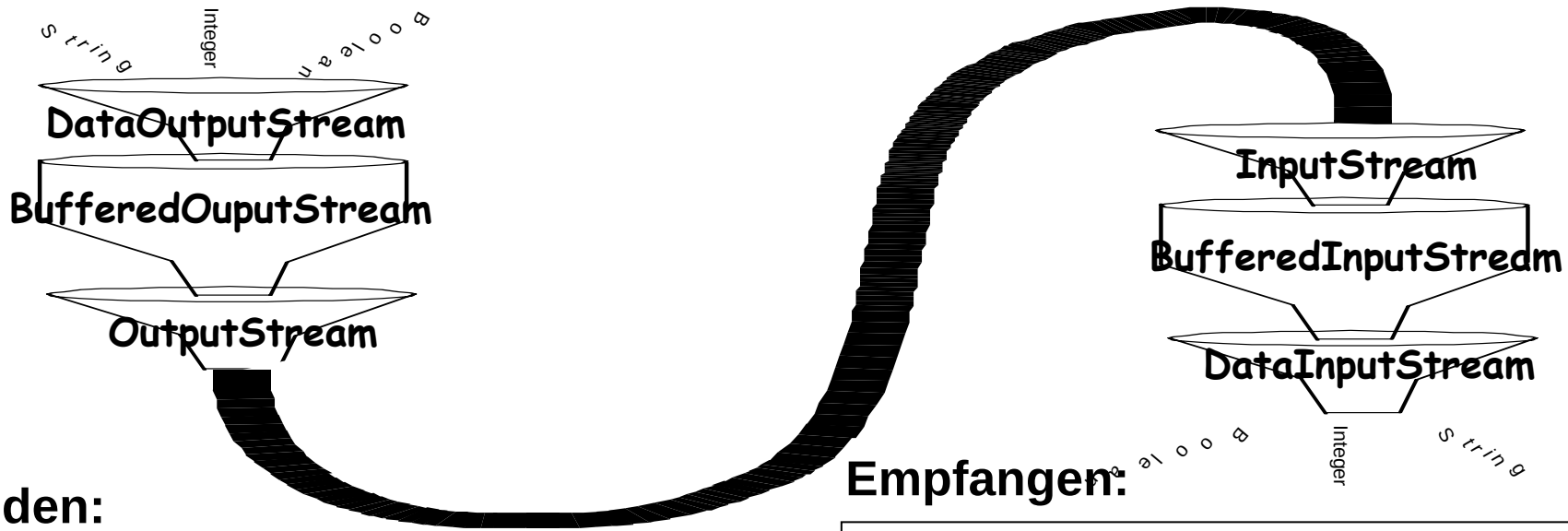
Senden:

```
Socket socket;
   // muss inititialisiert 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 inititialisiert 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



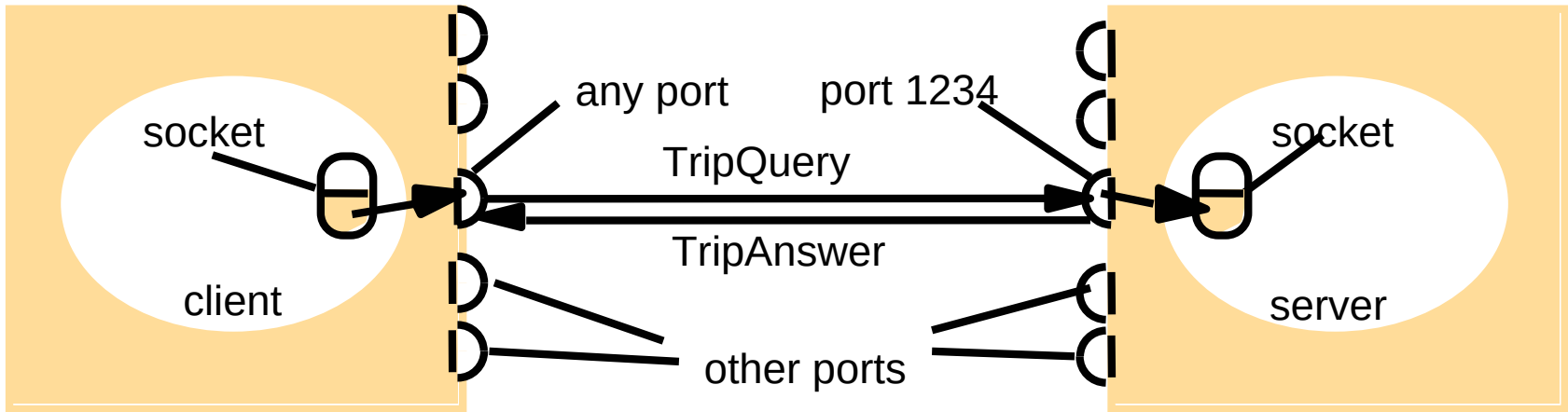
Senden:

```
Socket socket;

   // muss inititialisiert werden

DataOutputStream out = new
   DataOutputStream(new
   BufferedOutputStream(
   socket.getOutputStream()));

Object obj = new Object;

out.writeUTF(obj.toString());

out.flush();
```

Empfangen:

```
Socket socket;

   // muss inititialisiert 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 verhindert man die Blockade von Client und Server ?

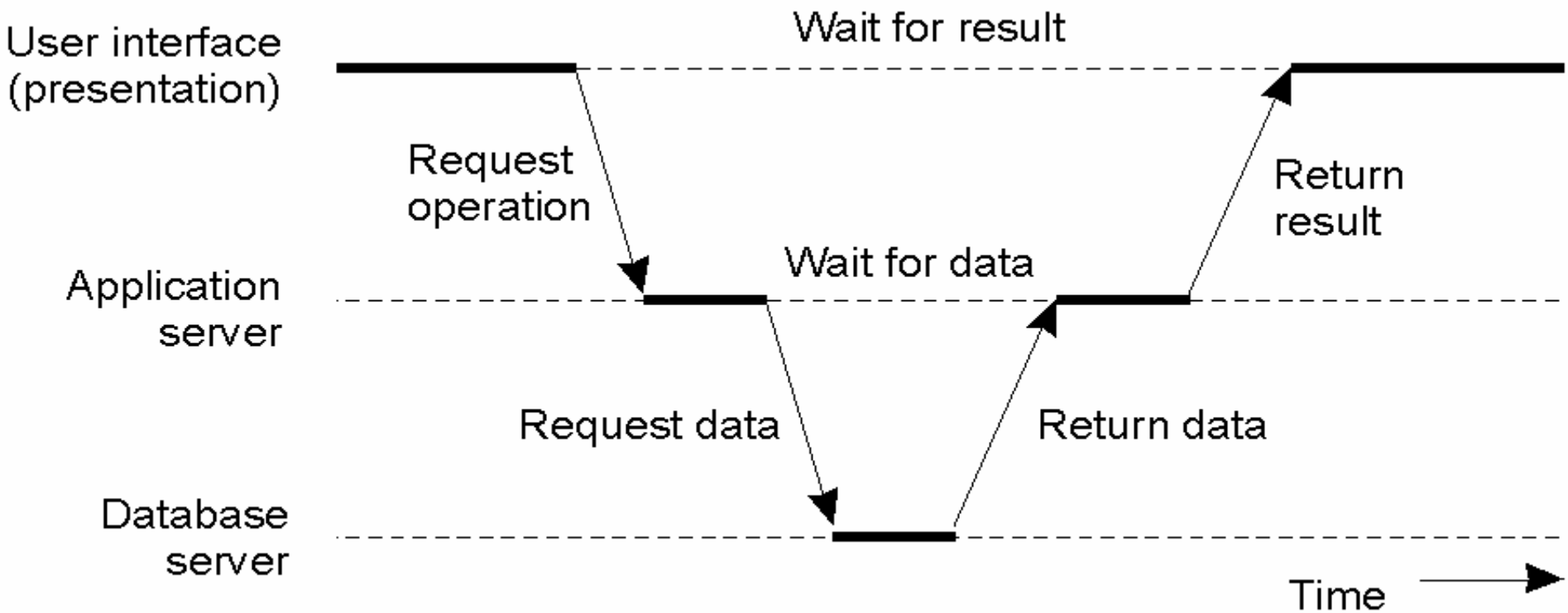
➔ durch Nebenläufigkeitstechniken (Details in Kapitel 2.2)

Wie verknüpft man auf Anwendungsebene Fragen und Antworten ?

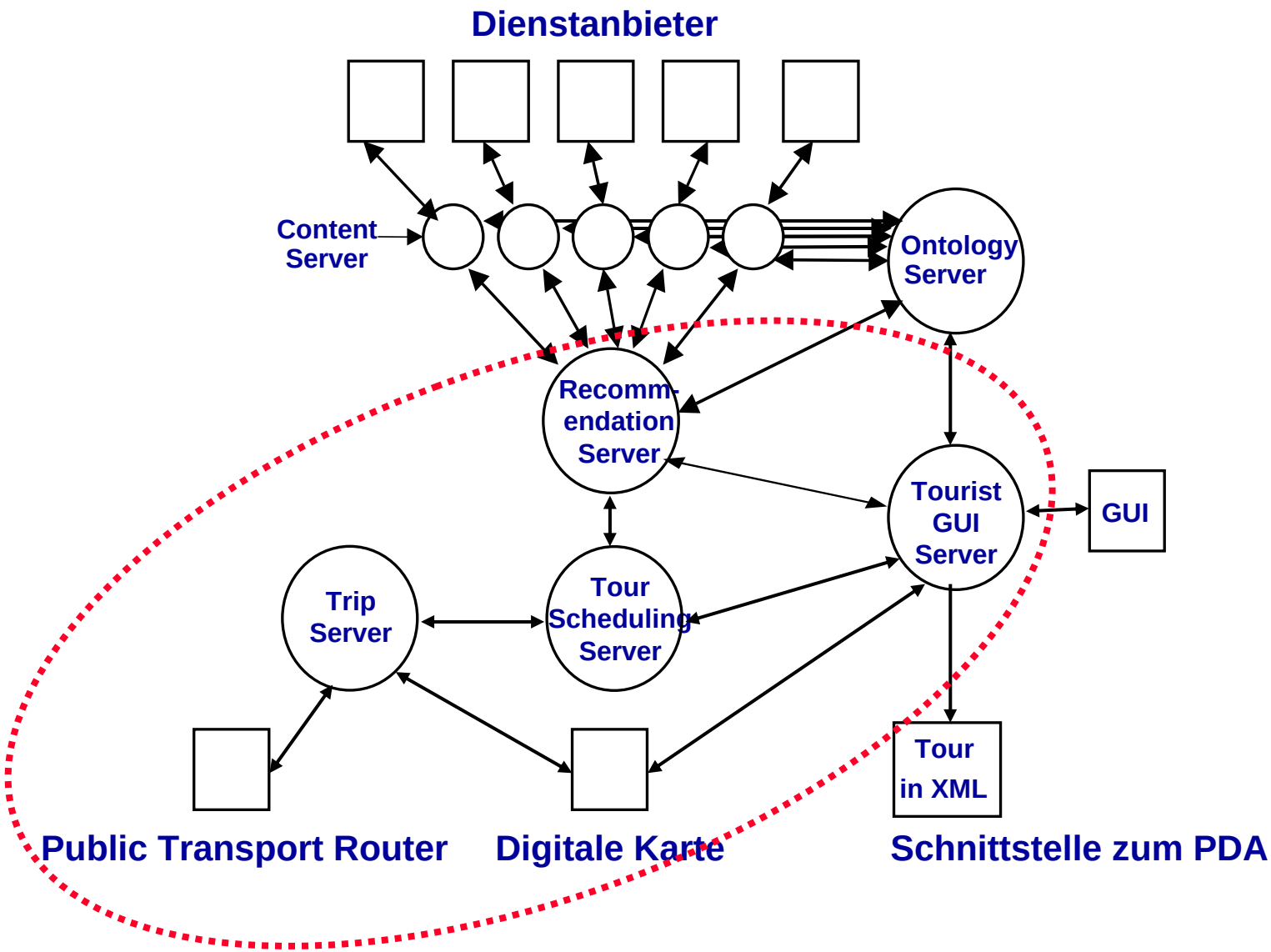
➔ kann durch verschiedene Techniken geregelt werden (2.2, 2.3, 2.4, Kapitel 3)

Mehrschichten-Architektur (Multitiered architecture)

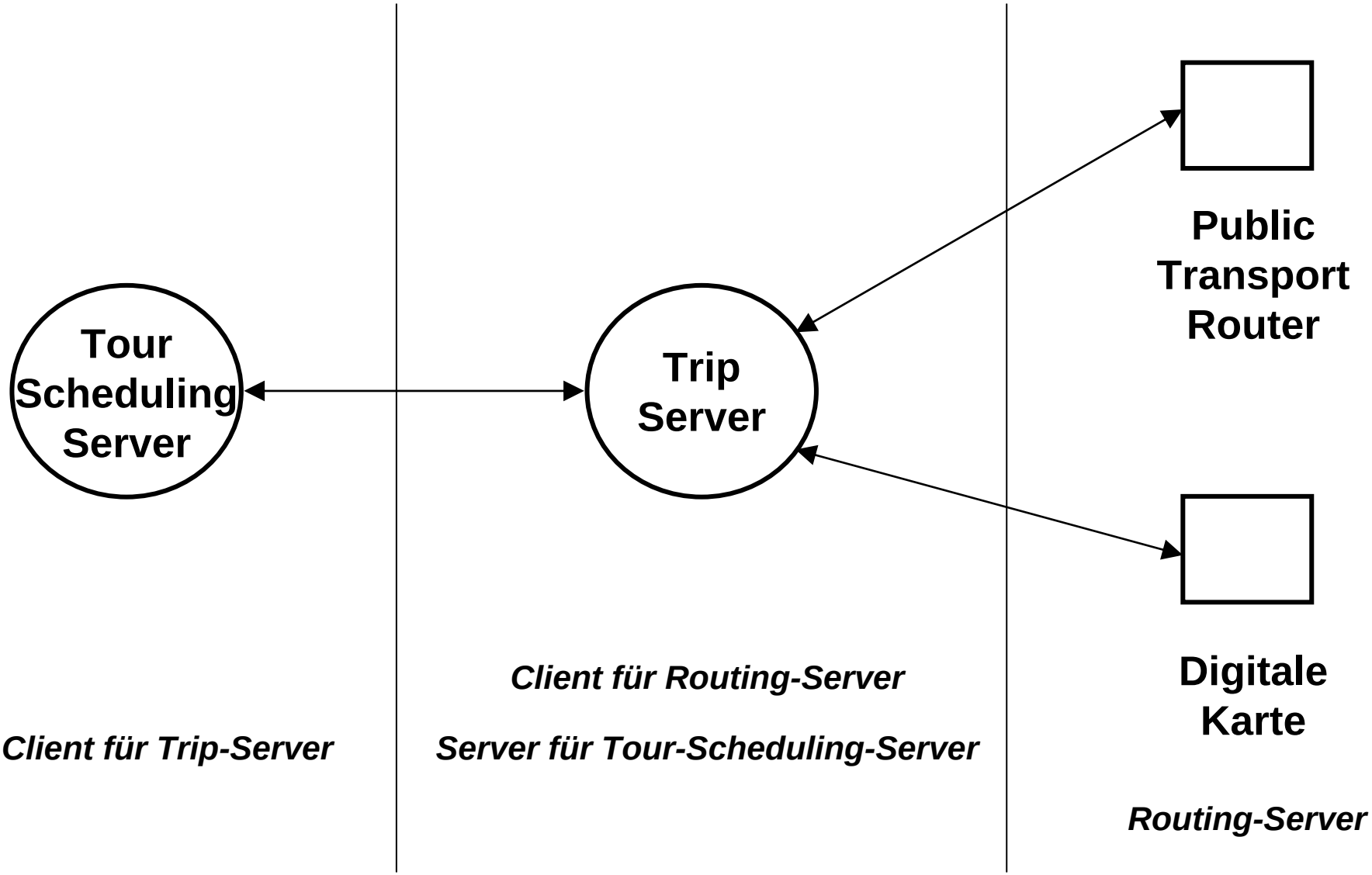
Beispiel für 3 Schichten:



Beispiel: Touristeninformationssystem



Bsp. für mehrere Schichten im Touristeninformationssystem



Weitere Schwierigkeiten beim Touristeninformationssystem

Client und Server befinden sich an unterschiedlichen Orten

➔ Lösungen in Kapitel 2.3 „Entfernte Aufrufe“

Heterogene Datenwelt in unterschiedlichen Servern

➔ Lösungen in Kapitel 3 „Dienstevermittlung“