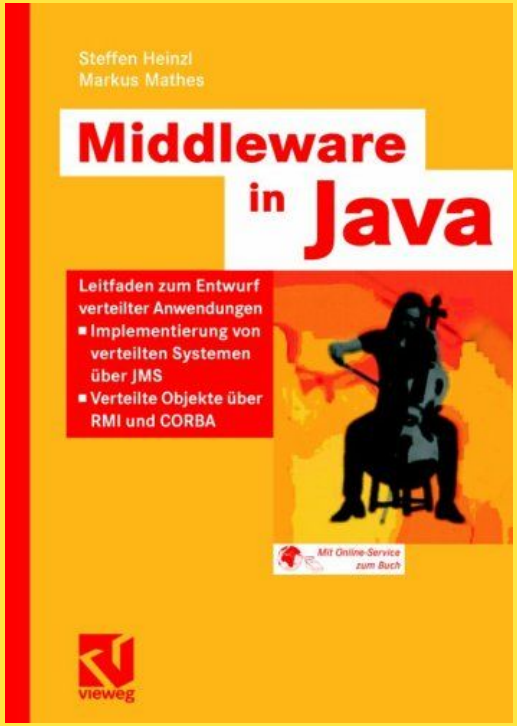
Kapitel 4:
Design von Client/Server-Software



Definition: Client und Server

Client

Ein Client ist ein ***Dienstnutzer***, der von einem Server ***aktiv*** einen Dienst anfordert und anschließend darauf wartet, dass der Server den angeforderten Dienst erbringt.

Server

Ein Server ist ein ***Dienstanbieter***, der für einen Client eine bestimmte Funktionalität in Form eines Dienstes erbringt und ***passiv*** darauf wartet, dass ein Client eine Anforderung an ihn stellt.

Aufgaben eines Servers

Server müssen neben der Kommunikation mit dem Client und der Erbringung des Dienstes noch weitere Aufgaben realisieren:

- ***Authentifizierung:*** Identifikation des Clients
- ***Autorisierung:*** Rechte eines Clients prüfen
- ***Datenschutz:*** Schutz personenbezogener Daten
- ***Datensicherheit:*** Schutz der Daten vor Manipulation
- ***Schutz des Betriebssystems***

$\Rightarrow$ Server sind komplexer als Clients

Vor- und Nachteile des Client/Server-Modells

Vorteile:

- in heterogenen Umgebungen einsetzbar
- Begriffe Client/Server sind in Theorie/Praxis gefestigt
- Interaktion zwischen Client/Server klar definiert
- ermöglicht auch Entwurf von Hardware

Nachteile:

- keinerlei Transparenz
- Anwendung kann sowohl Client- als auch Server-Funktionalität besitzen
- relativ altes Entwurfsmodell

Entwurf von Client und Server

Client

- parametrisierbar vs. nicht parametrisierbar
- iterativ vs. parallel
- verbindungslos vs. verbindungsorientiert

Server

- statuslos vs. statusbehaftet
- iterativ vs. parallel
- verbindungslos vs. verbindungsorientiert

Entwurfsaspekte eines Clients

- parametrisierbar
  - Benutzer kann Kommunikations-parameter vorgeben
- iterativ
  - genau ein Ausführungsfaden
- verbindungslose
  - UDP als Transport-protokoll
- nicht parametrisierbar
  - Client verwendet immer die selben Kom-munikationsparameter
- parallel
  - mehrere Ausführungs-fäden
- verbindungsorientiert
  - TCP als Transport-protokoll

Entwurfsaspekte eines Servers

- statuslos
  - Server speichert keinerlei Verlaufs-informationen über die Kommunikation
- iterativ
  - genau ein Ausführungsfaden
- verbindungslos
  - UDP als Transport-protokoll
- statusbehaftet
  - Server speichert Verlaufsinformationen über die Kommuni-kation
- parallel
  - mehrere Ausführungs-fäden
- verbindungsorientiert
  - TCP als Transport-protokoll

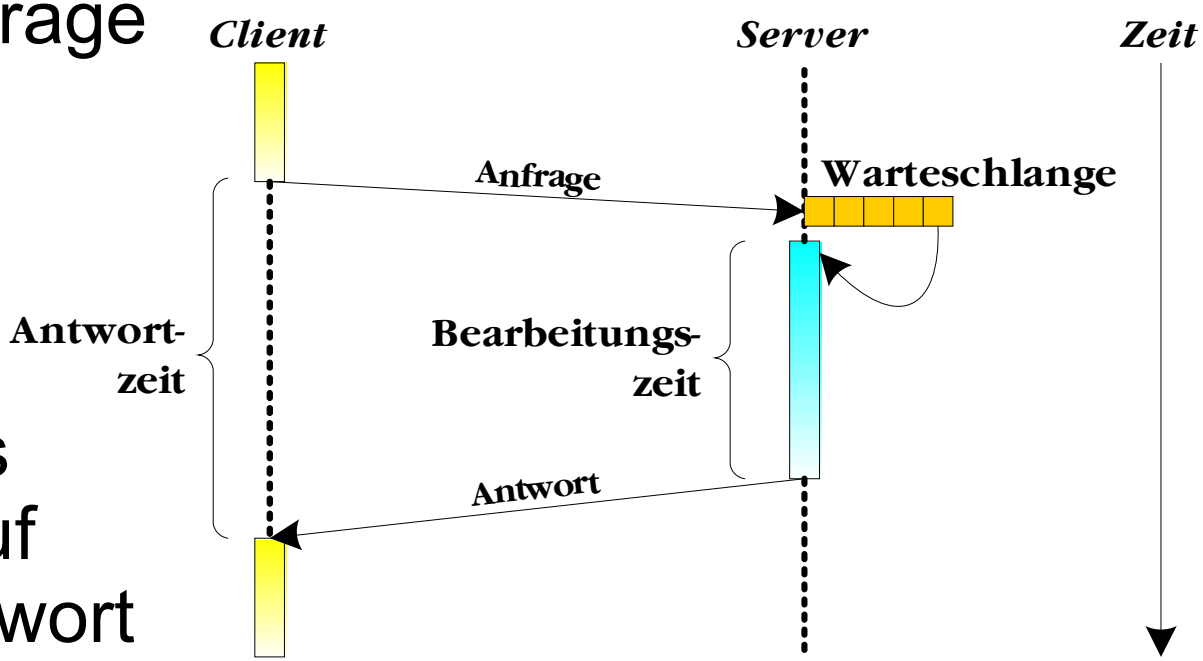
Antwort- und Bearbeitungszeit

Bearbeitungszeit:

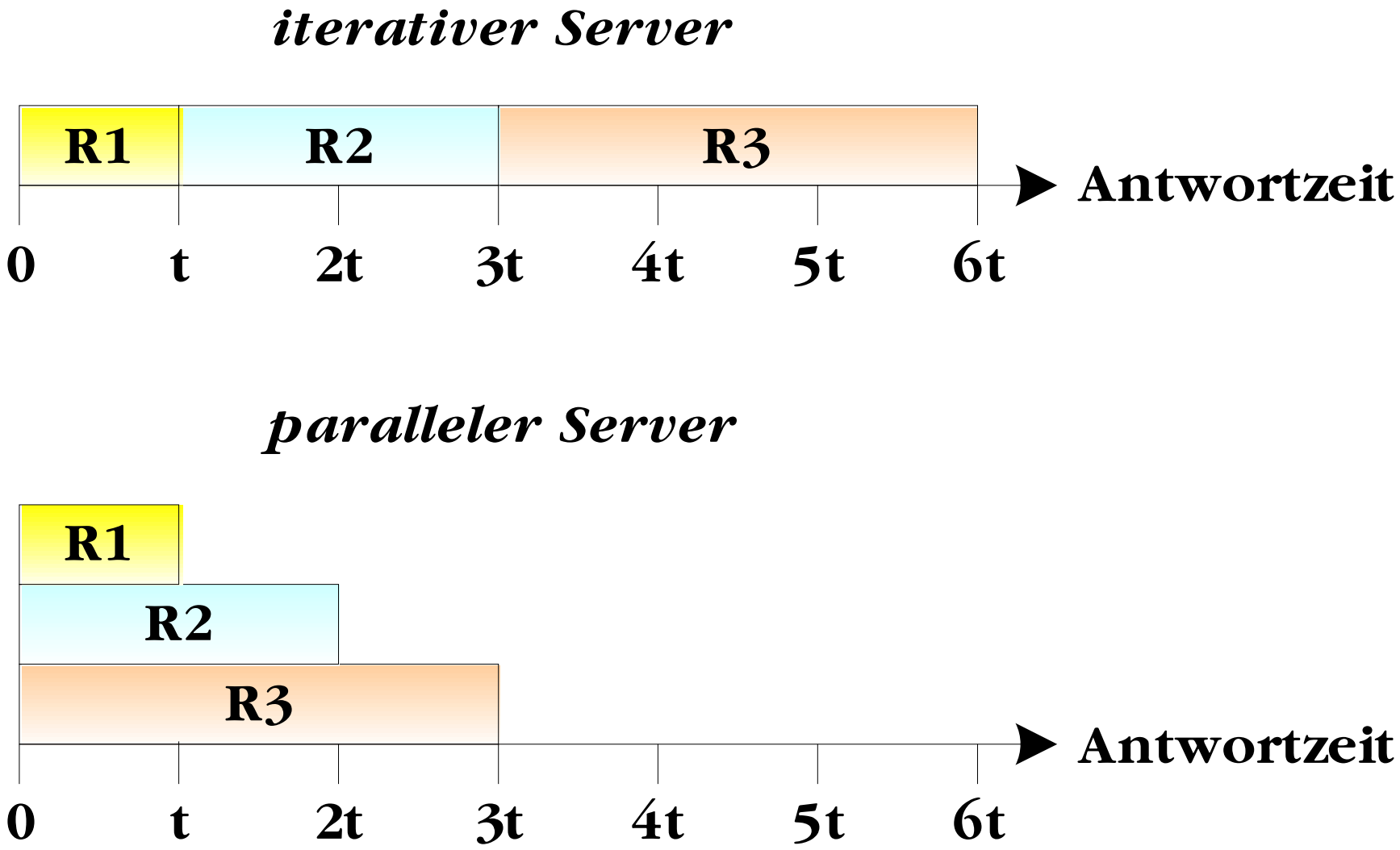
Zeit, die der Server zur Bearbeitung einer Anfrage benötigt

Antwortzeit:

Zeit, die aus Sicht des Clients vergeht, bis auf eine Anfrage eine Antwort eintrifft



Antwortzeit bei iterativem und parallelem Server



Multiprotokoll-Server

- Ein Multiprotokoll-Server unterstützt gleichzeitig ***mehrere Transportprotokolle***.
- Oftmals werden die Transportprotokolle so gewählt, dass eine ***verbindungslose*** und ***verbindungsorientierte*** Kommunikation möglich wird.
- ***Vorteile:***
  - Einsparung von Systemressourcen
  - Codewiederverwendung

Multiservice-Server

- Ein Multiservice-Server bietet ***mehrere Dienste*** gleichzeitig für ein bestimmtes Transportprotokoll an.
- Beispielsweise kann ein Server gleichzeitig einen TIME- und DAYTIME-Dienst für UDP oder TCP anbieten.
- ***Vorteile:***
  - Einsparung von Systemressourcen
  - Codewiederverwendung

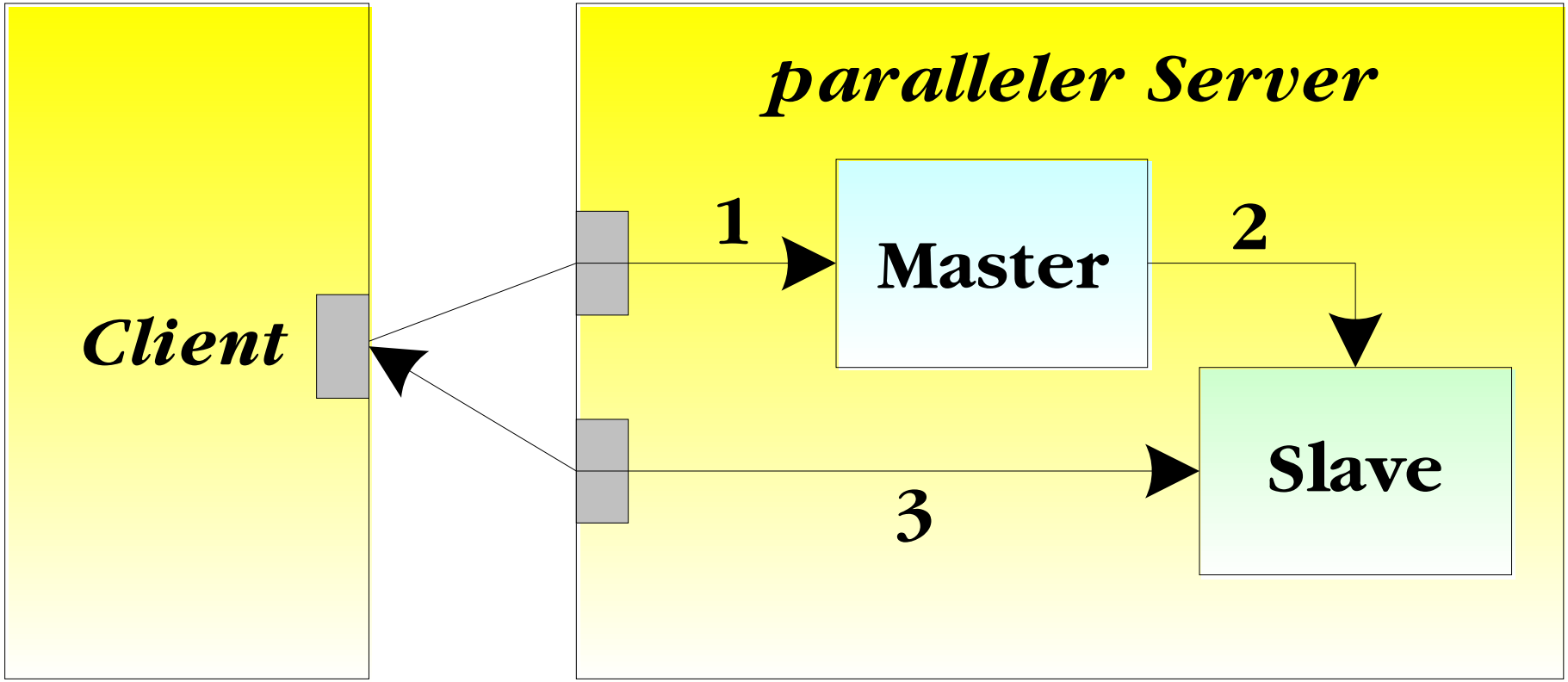
Super-Server

- Ein Super-Server bietet ***mehrere Dienste*** gleichzeitig für ***mehrere Transportprotokolle*** an.
- Dadurch kommt es zu einer ***starken*** Einsparung von Systemressourcen.
- Viele Linux-Systeme liefern beispielsweise den Super-Server ***inetd*** mit.
- ***Vorteile:***
  - starke Einsparung von Systemressourcen
  - Codewiederverwendung

Master/Slave-Prinzip (1)

- Viele parallele Server werden nach dem ***Master/Slave-Prinzip*** implementiert.
- Der Master wartet in einer Endlosschleife auf das Eintreffen von Anforderungen an einem bestimmten Port.
- Trifft eine Anforderung ein, erzeugt der Master einen Slave und delegiert die Bearbeitung der Anfrage an diesen. Anschließend wartet der Master wieder auf neue Anforderungen.
- Der Slave bearbeitet die erhaltene Anforderung und kommuniziert dazu über einen eigenen Port mit dem Client.

Master/Slave-Prinzip (2)



Socket-API (1)

Die Socket-API wurde erstmals ***1982*** durch die University of California in Berkeley als Teil von BSD UNIX 4.1c eingeführt.

Durch sie sollte ein einfacher Zugriff auf die Funktionen des Protokoll-Stacks zum Senden und Empfangen von Daten ermöglicht werden.

Socket-Primitive:

- `socket`
- `bind`
- `listen`
- `accept`
- `connect`
- `read/write`
- `sendto/recvfrom`
- `close`

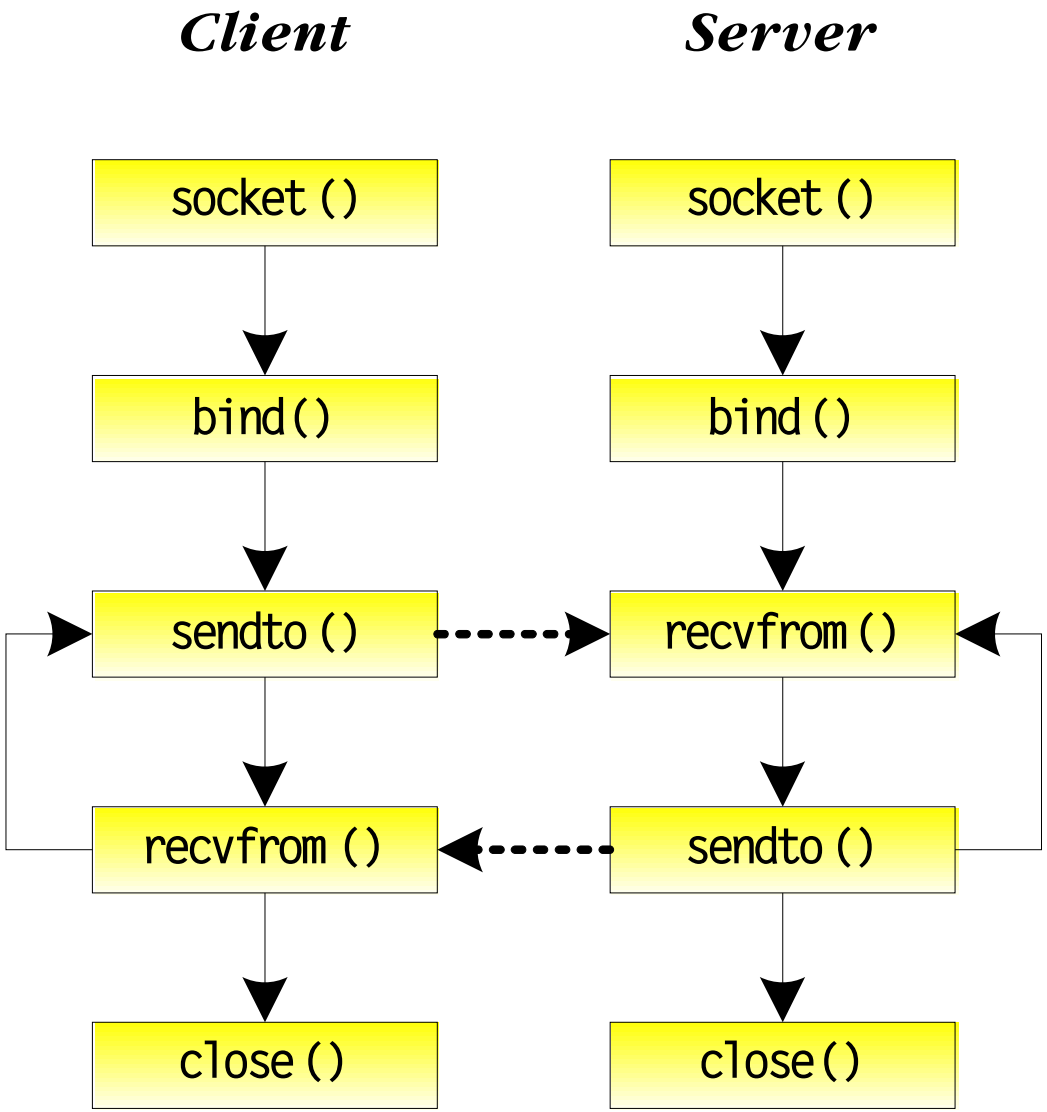
Socket-API (2)

- `socket`
  - neuen Kommunikationsendpunkt für UDP oder TCP erzeugen
- `bind`
  - Socket lokale Adresse zuordnen (Socket an Adresse binden)
- `listen`
  - Socket in passiven Zustand versetzen
- `accept`
  - blockierend auf einen Verbindungswunsch warten

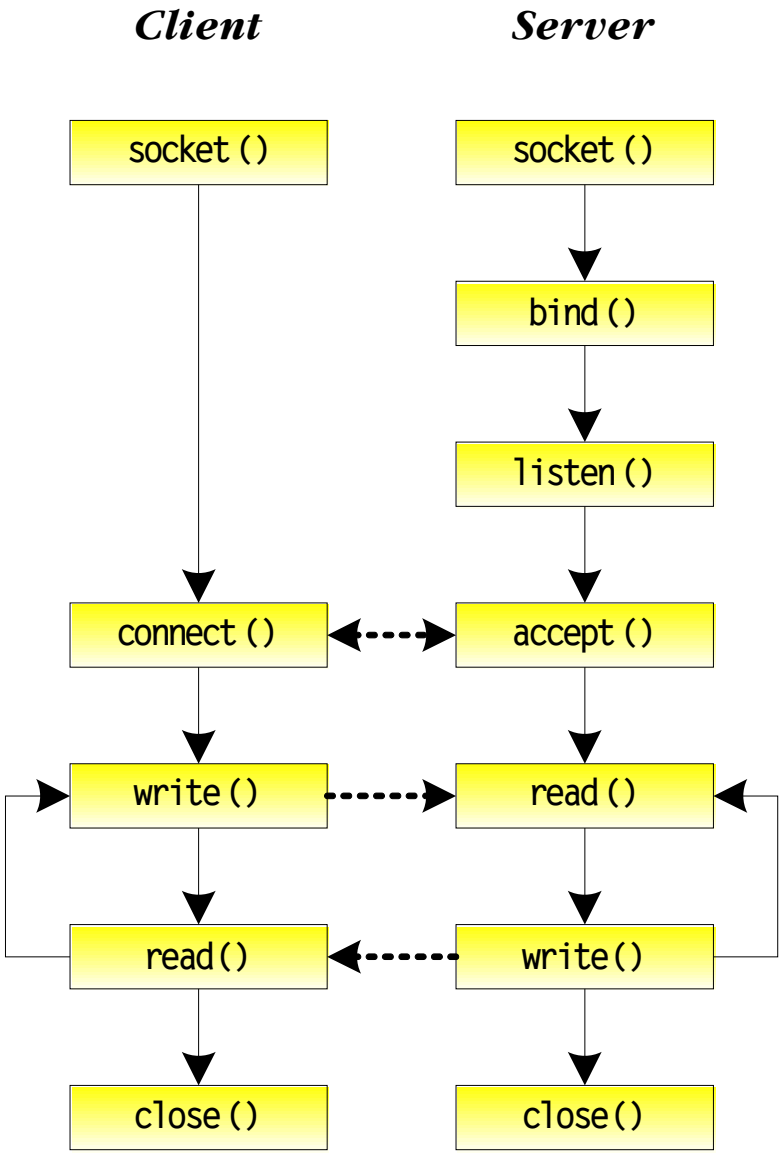
Socket-API (3)

- `connect`
  - Aufbau einer Verbindung zum Server
- `read/write`
  - Empfangen und Senden von Daten
- `sendto/recvfrom`
  - Datagramm senden und empfangen
- `close`
  - Socket schließen und Ressourcen freigeben

verbindungslose Socket-Kommunikation



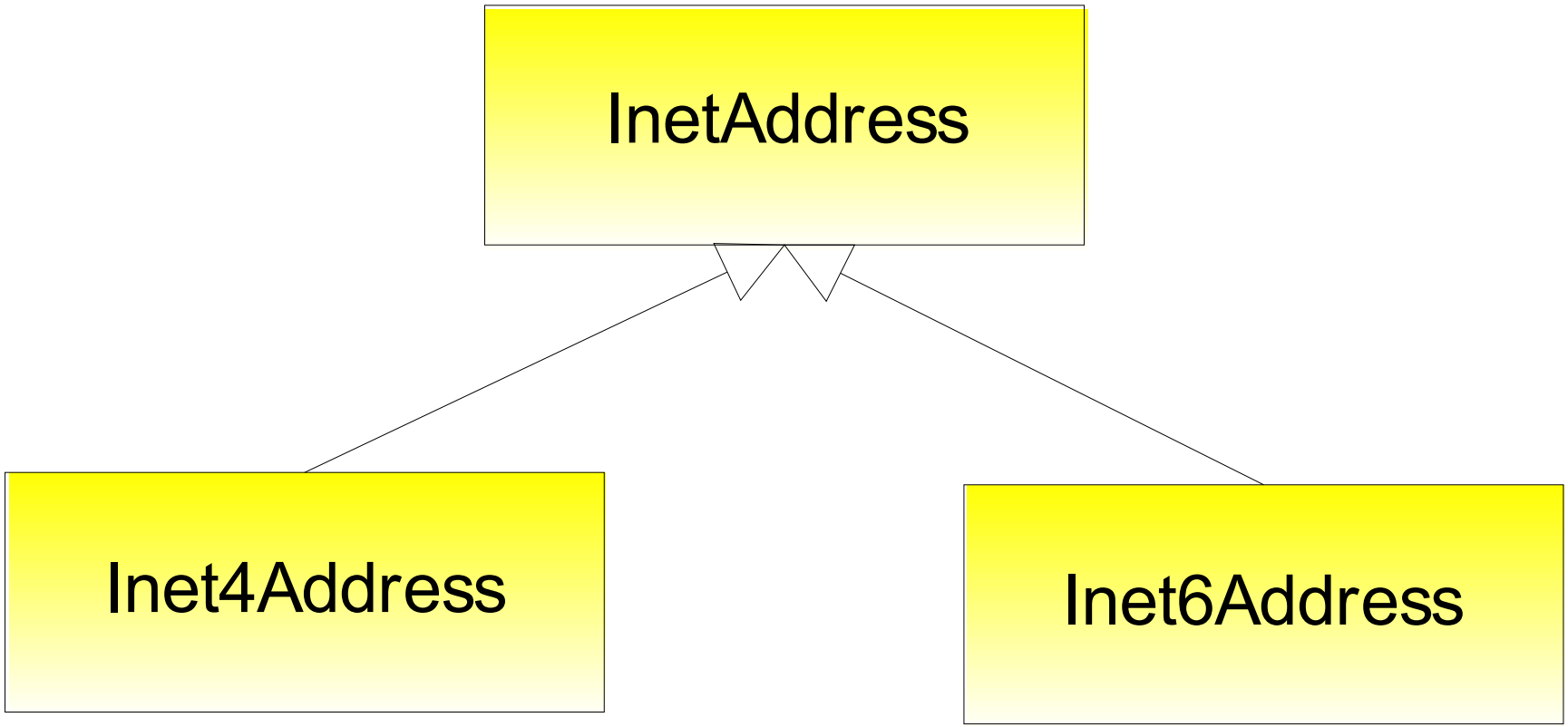
verbindungsorientierte Socket-Kommunikation



Socket-API unter Java

- Adress- und Namensauflösung
  - `InetAddress`
  - `Inet4Address`
  - `Inet6Address`
- UDP-Kommunikation
  - `DatagramPacket`
  - `DatagramSocket`
- TCP-Kommunikation
  - `ServerSocket`
  - `Socket`

Adressklassen



Auflösung eines Host-Namens

```
InetAddress ipAddr = null;
try
{
  // Namensauflösung
  ipAddr = InetAddress.getByName("www.fh-fulda.de");
}
catch(UnknownHostException e)
{
  e.printStackTrace();
  System.exit(-1);
}
```

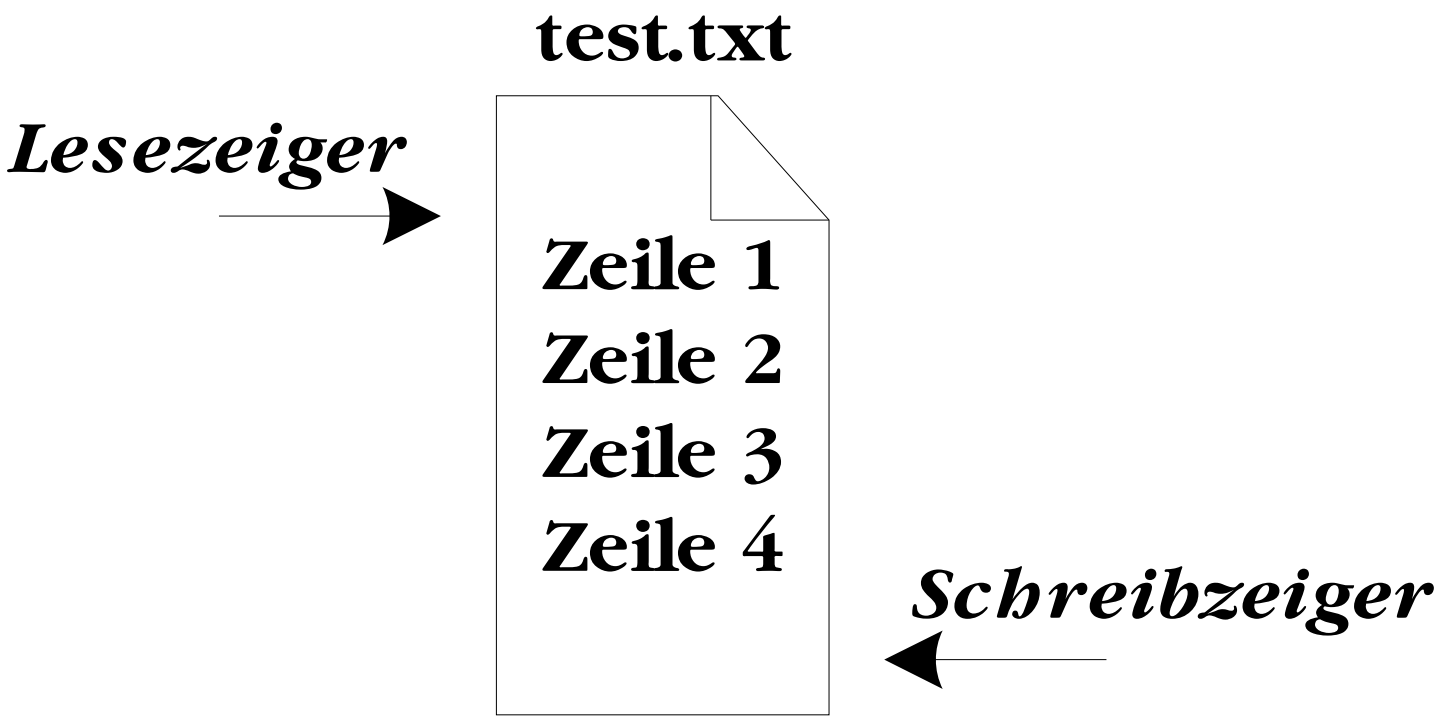
DatagramPacket und DatagramSocket

- Ein `DatagramSocket` repräsentiert einen ***UDP-Socket*** und besitzt die drei wichtigen Konstruktoren
  - `DatagramSocket()`
  - `DatagramSocket(int port)`
  - `DatagramSocket(int port, InetAddress laddr)`
- Ein `DatagramPacket` repräsentiert ein ***UDP-Paket*** und besitzt die Konstruktoren
  - `DatagramPacket(byte[] buf, int length)`
  - `DatagramPacket(byte[] buf, int length, InetAddress address, int port)`

Socket und ServerSocket

- Die Klasse `Socket` repräsentiert einen Kommunikationsendpunkt, über den Daten gesendet bzw. empfangen werden können. Die wichtigsten Konstruktoren sind:
  - `Socket()`
  - `Socket(String host, int port)`
- Ein `ServerSocket` ist ein passiver Socket und wird nur von Servern verwendet. Die beiden wichtigsten Konstruktoren von `ServerSocket` sind:
  - `ServerSocket(int port)`
  - `ServerSocket(int port, int backlog)`

Dateizugriff mit dem FILE-Protokoll (1)



Dateizugriff mit dem FILE-Protokoll (2)

- `OPEN Dateiname`
  - Öffnet die angegebene Datei.
- `READ`
  - Liest die nächste Zeile aus der zuvor geöffneten Datei.
- `WRITE Zeile`
  - Schreibt eine neue Zeile an das Ende der Datei.
- `CLOSE`
  - Schließt die zuvor geöffnete Datei.
- `SHUTDOWN`
  - Der Client beendet die Kommunikation mit dem Server.

Literatur

- Anatol Badach, Erwin Hoffmann: *Technik der IP-Netze – TCP/IP incl. IPv6*; Hanser 2001; http://www.fehcom.de /tipn
- Douglas E. Comer, David L. Stevens: *Internetworking with TCP/IP – Volume 3: Client-Server Programming And Applications*; Prentice Hall 2001; http://www.cs.purdue.edu/homes/comer/netbooks.html
- Guido Krüger: *Handbuch der Java-Programmierung (4. Auflage)*; Addison Wesley 2004; http://www.javabuch.de
- Martin Pollakowski: *Grundkurs Socketprogrammierung mit C unter Linux*; vieweg 2004; http://www.fh-gelsenkirchen.de/fb01/homepages/pollakowski/socket/index.html
- Sun Microsystems Inc.: *JDK 5.0 Documentation*; http://java.sun.com/j2se/1.5.0/docs/index.html, http://java.sun.com/j2se/1.5.0/docs/api/index.html
- Christian Ullenboom: *Java ist auch eine Insel*; Galileo Computing 2004; http://www.galileocomputing.de/openbook/javainsel4/

Aufgaben

In „***Middleware in Java***“ finden Sie

- Wiederholungs-,
- Vertiefungs-,
- Programmieraufgaben

zu den vorgestellten Themen.

Zur Festigung und Vertiefung des Erlernten wird eine Bearbeitung der Aufgaben empfohlen.