Netzwerkprogrammierung mit HTTP/SMTP

PROG 2: Einführung in die Programmierung für Wirtschaftsinformatiker

---

Steffen Helke
Technische Universität Berlin
Fachgebiet Softwaretechnik

24. Juni 2013

Übersicht

- Client-Server-Konzept
- Webdienste mit HTTP
- Maildienst mit SMTP
- RMI ➜ nächste Vorlesung

Teil V der Vorlesung PROG 2

Netzwerkprogrammierung

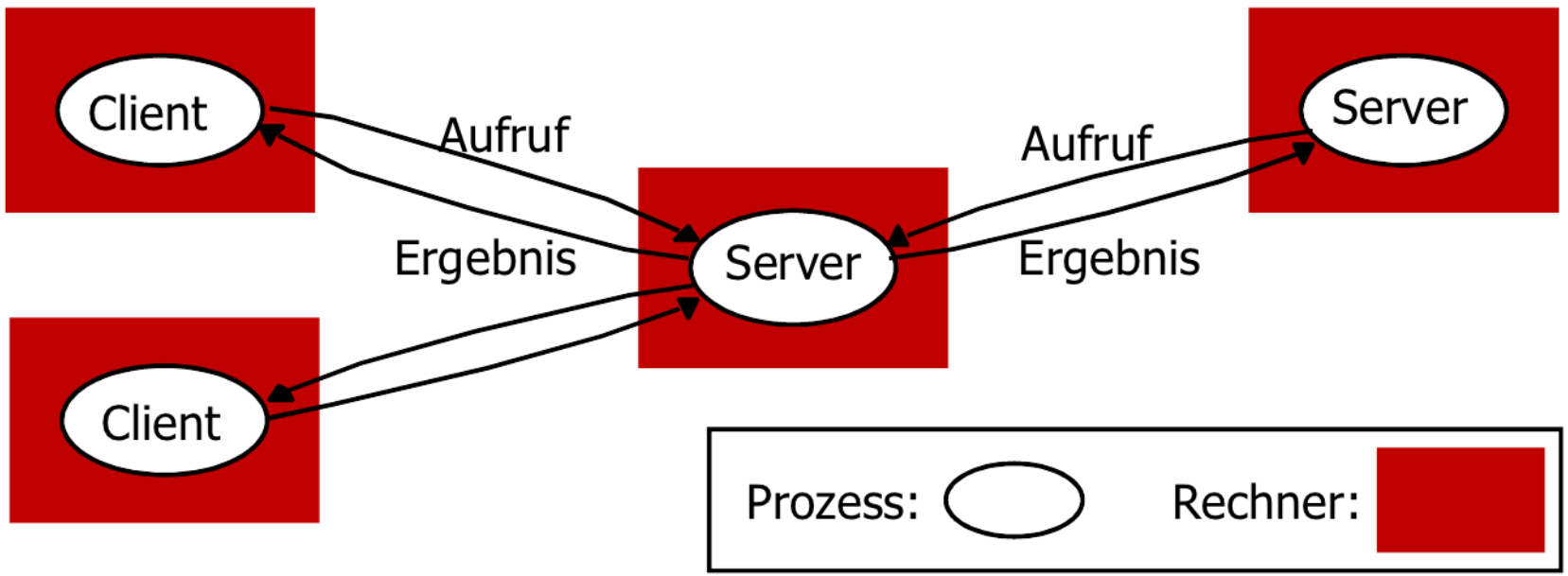
Client-Server-Konzept

Quelle: *Inhalt & Gestaltung nach Vorlesungsfolien von Peter Pepper und Odej Kao, TU Berlin*
Methodische- und Praktische Grundlagen der Informatik 4 (MPGI 4), WS 2010/11 bzw. WS 2011/12

Client-Server-Architektur

Grundprinzip

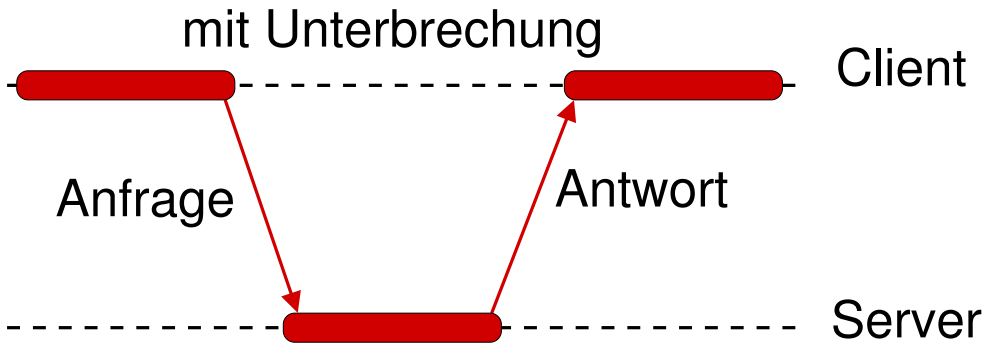
- mehrere Clients fordern bei einem Server Dienste an
- Server können Anfragen an andere Server weiterleiten
- auf Client und Server laufen unabhängige Prozesse



Kommunikationsarten

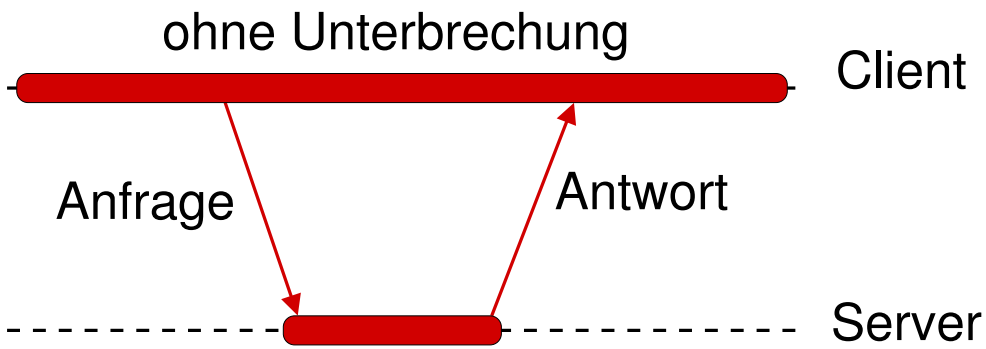
Synchrone Kommunikation

➜ z.B. Remote Method Invocation (RMI)



Asynchrone Kommunikation

➜ z.B. Email-Kommunikation (SMTP)



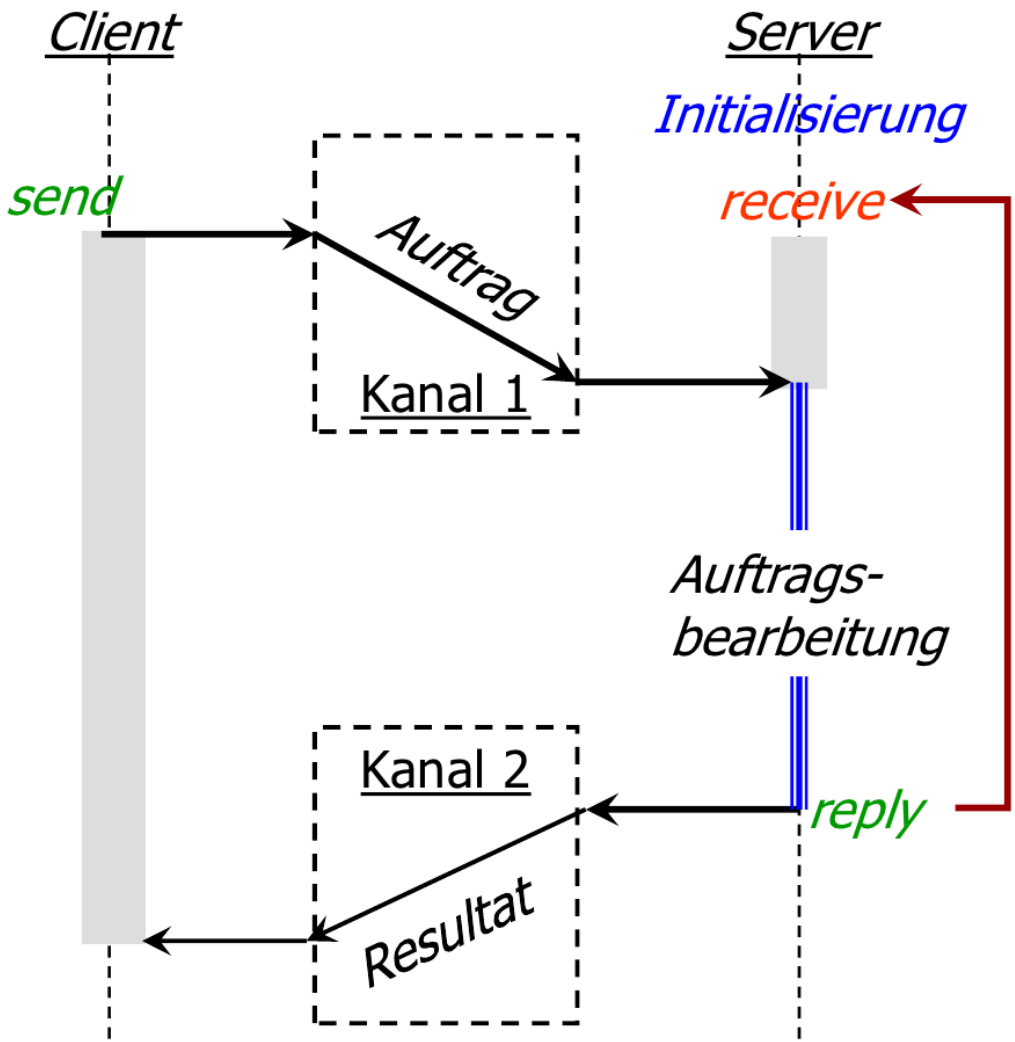
Nutzung von Server-Diensten

1. Auftragsversendung durch Client
2. Auftragsannahme, Bearbeitung und Antwort durch Server
3. Server wartet auf neue Aufträge

➜ *zyklischer Server-Prozess*

➜ Socket-Kommunikation

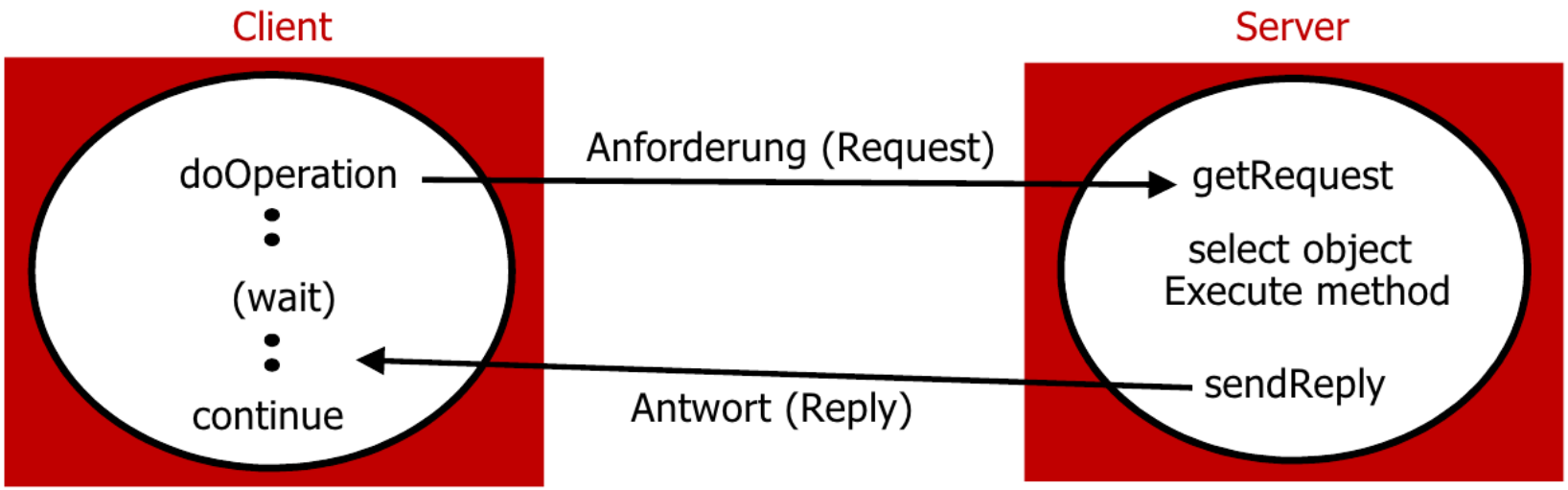
➜ verbindungsorientierte Kommunikation



Client-Server-Kommunikation

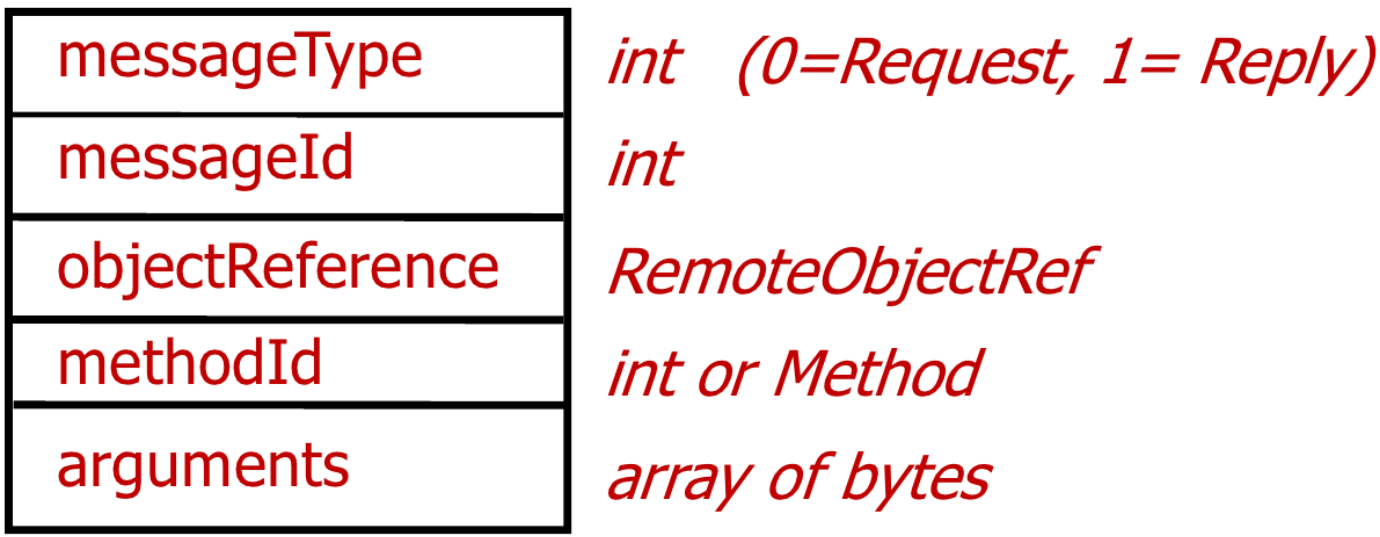
- Aufruf von *doOperation()* als Client-Anfrage
- Aufruf von *getRequest()* zur Anfrageannahme durch Server
- Versenden der Antwort durch Server mit *SendReply()*

➜ meist synchrone Kommunikation



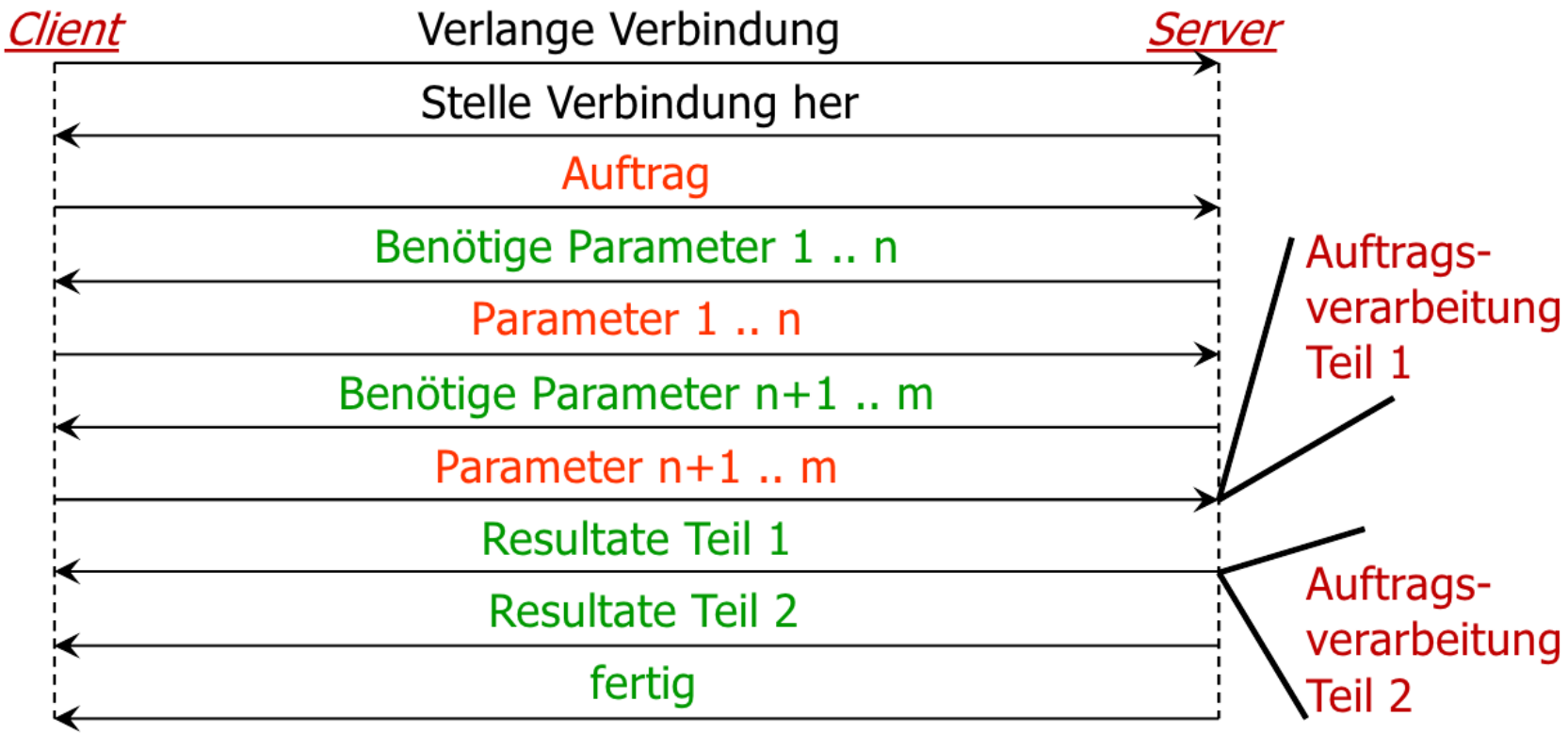
Nachrichtenstruktur von Anfragen/Antworten

- Spezifikation ob Antwort oder Anfrage mit messageType
- eindeutige IDs mit messageId, vom Client erzeugt, enthält zusätzlich requestID, Informationen über Port/IP
- Referenz auf entferntes Objekt mit objectReference und aufzurufende Methode methodID
- Argumente zur Methodenausführung in arguments



Kommunikation zwischen Client und Server

- ➜ flexibler Austausch von Parametern zwischen Client und Server
- ➜ versetzte Auftragsbearbeitung durch Server möglich



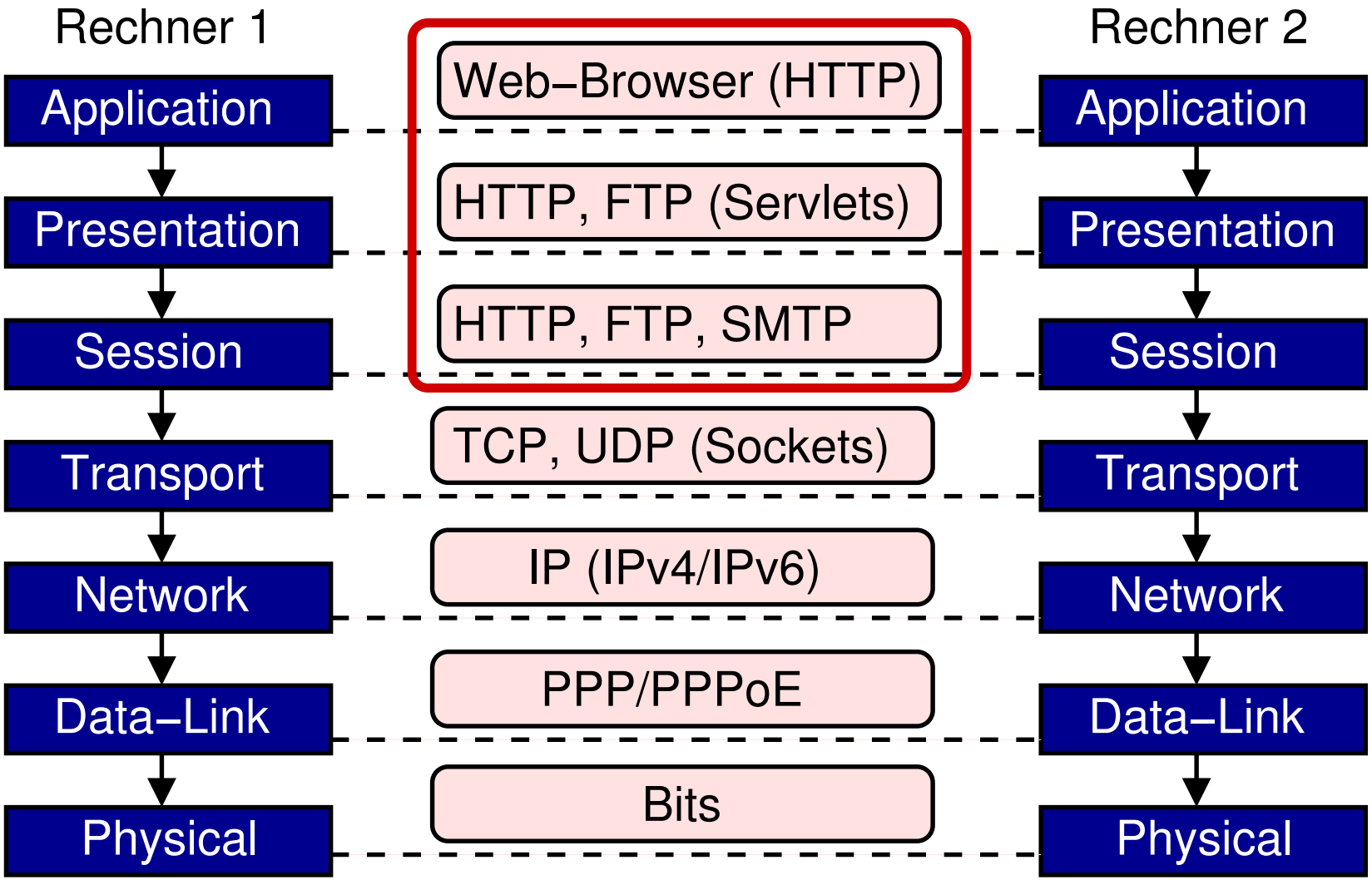
Teil V der Vorlesung PROG 2

Netzwerkprogrammierung

Webdienste mit HTTP

Quelle: *Inhalt & Gestaltung nach Vorlesungsfolien von Peter Pepper und Odej Kao, TU Berlin*
Methodische- und Praktische Grundlagen der Informatik 4 (MPGI 4), WS 2010/11 bzw. WS 2011/12

HTTP-Protokolle im OSI-Modell



Hypertext Transfer Protocol - HTTP

HTTP-Spezifikation unterstützt ...

- Austausch von Nachrichten, Methoden und Ergebnissen
- Satz von HTTP-Methoden, wie z.B. *GET*, *PUT* oder *POST*
- Marshalling: Festlegung eines gemeinsamen Zeichensatzes und gemeinsamer Datenformate
- *Authentifizierung:* Berechtigungen (Credentials) und/oder Challenges (die der Server an den Client stellt), Antwort könnte Passwort-Eingabe des Benutzers am Client sein

Kodierungsformate sind ...

- ASCII-Texte für Anfragen bzw. Antworten
- Byte-Darstellungen für Ressourcen mit Header

  ➜ MIME-Format (*Multipurpose Internet Mail Extension*)

Kommunikationsverlauf bei HTTP

Anfragen stellen

➜ http://servername[:port][/PfadNameAufServer][?Argumente]

Ablauf der Kommunikation vor HTTP 1.1

1. Client fordert Verbindung, Server akzeptiert und baut Verbindung auf
2. Client sendet Anfrage, erhält Antwort vom Server
3. Verbindung wird wieder geschlossen

Änderungen ab HTTP 1.1

- Verbindungsaufbau für jeden Auftrag ist zu teuer
  ➜ Verbindung bleibt für mehrere Aufträge bestehen
- Verbindung kann von Client/Server flexibel geschlossen werden; Server schließt automatisch bei Anfrageinaktivität

HTTP-Request-Methoden

- **GET**:
  Anforderung einer Ressource vom Server incl. Header
- **POST:**
  Senden von Daten an den Server (z.B. Name-Wert-Paare)
- **HEAD:**
  Anforderung des Headers einer Ressource
- **PUT:**
  Hochladen einer Ressource auf den Server
- **TRACE:**
  Rückgabe der vom Server empfangenen Client-Anfrage (Test)
- **OPTIONS:**
  Liste der vom Server unterstützen Methoden und Features
- **CONNECT:**
  Aufbau von SSL-Tunneln (*Transport Layer Security*)

HTTP-Server gibt Antwort als Status-Code

Varianten von Status-Codes

1. Anfrage wurde erfolgreich empfangen, Antwort wird bearbeitet, d.h. keine erneute Anfrage nötig
2. Bearbeitung wurde erfolgreich abgeschlossen, Antwort wird zurückgeschickt
3. angeforderte Seite ist umgezogen, Rückantwort enthält neue Adresse
4. fehlerhafte Anfrage, z.B. Seite existiert nicht (404)
5. ein anderer Fehler beim Server ist aufgetreten

➜ zusätzlich zum Status wird eine (Fehler-)beschreibung geliefert

Wie wird der HTTP-Client in Java umgesetzt?

Varianten

1. Stream-Socket-Verbindung mit der Klasse *Stream* und selbst zusammengebauten HTTP-Anfrage-String, z.B.

```
String anfrage = Methode +" "+ Datei + " HTTP/1.1\r\n" +
                 "Host: "+ host +"\r\n\r\n"
```

2. Benutzung von Methoden der Klasse *URL*

```
// Adresserzeugung
public URL(String protocol,String host,
           int port,String file)
// Verbindungsaufbau
public URLConnection openConnection()

// Input-Stream aufstellen
public InputStream getInputStream()

// Verbinden und Daten lesen mit einer Methode
public final InputStream openStream()
```

Variante 1: Zugriff auf HTTP-Webserver

```
public class HTTPClient {
 public static void main(String[] args) {
  String host = "www.user.tu-berlin.de";
  String method = "GET";
  String file = "/helke/index.html";
  Socket httpSocket = null;
  OutputStream ausgabeStream = null;
  InputStream eingabeStream = null;
  try {
    httpSocket = new Socket(host,80);
    ausgabeStream = httpSocket.getOutputStream();
    eingabeStream = httpSocket.getInputStream();
    String httpBefehl = method + " " + file + " HTTP/1.1" +
                        "\r\n"+ "Host: " + host +"\r\n\r\n";
    ausgabeStream.write(httpBefehl.getBytes());
    // Rueckgabe-String auslesen
    byte[] bytearray = new byte[1000];
    while ((length = eingabeStream.read(bytearray)) != -1)
       System.out.write(bytearray, 0, length);
    eingabeStream.close(); ausgabeStream.close(); ...
    } catch { ... }}}
```

Variante 2: Zugriff auf HTTP-Webserver

➜ deutlich weniger Schreibaufwand bei Verwendung der URL-Klasse

```
public class HTTPClient {
 public static void main(String[] args) {
  String host = "www.user.tu-berlin.de";
  String method = "GET";
  String file = "/helke/index.html";
  Socket httpSocket = null;
  OutputStream ausgabeStream = null;
  InputStream eingabeStream = null;
  try {
    URL url = new URL("http", host, 80, file)
    eingabeStream = url.openStream();
    // Rueckgabe-String auslesen
    byte[] bytearray = new byte[1000];
    while ((length = eingabeStream.read(bytearray)) != -1)
       System.out.write(bytearray, 0, length);
    eingabeStream.close(); ausgabeStream.close(); ...
    } catch { ... }}}
```

Weitere Methoden für Zugriff auf Webserver

➜ zusätzliche Funktionalität in den Klassen *URLConnection* und *HttpURLConnection*

```
// Verbindungsaufbau bei gegebenem URL-Objekt
protected URLConnection(URL url)

// Kodierungsformat des Headers
String getContentEncoding();
// Laenge des Headers
int getContentLength();
// Ablaufangabe fuer Webseite
public long getExpiration()
// Auslesen notwendiger Zugriffsrechte
public Permission getPermission()
// Auslesen falls seit Datum modifiziert
public long getIfModifiedSince()
// Verbindung für schreibenden Stream erzeugen
public OutputStream getOutputStream()
```

Teil V der Vorlesung PROG 2

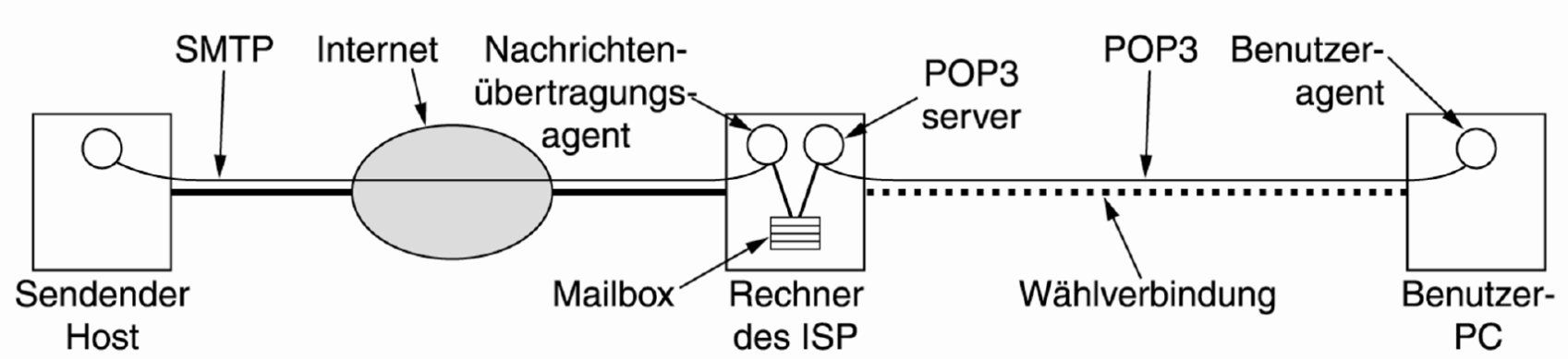
Netzwerkprogrammierung

Maildienste mit SMTP

Quelle: *Inhalt & Gestaltung nach Vorlesungsfolien von Peter Pepper und Odej Kao, TU Berlin*
Methodische- und Praktische Grundlagen der Informatik 4 (MPGI 4), WS 2010/11 bzw. WS 2011/12

Simple Mail Transfer Protocol - SMTP

- E-Mail-Übertragung basiert auf TCP-Verbindung
- speziell SMTP ist einfaches ASCII-Protokoll
- Mail-Server hört auf Port 25
- Client baut Verbindung auf und wartet, bis Server mit Kommunikation beginnt
- SMTP erfordert permanente Internetverbindung, anders als bei IMAP (Internet Message Access Protocol)



Aufbau von E-Mails

Basis-Definition

- Nachrichtenformat RFC 822
- Header enthält Felder, wie *TO*, *CC*, *Bcc*, *FROM* oder *RETURN-PATH*
- Nachrichtentexte wurden *früher nur auf Englisch* verfasst

  ➜ ASCII-basiert, macht Probleme in anderen Sprachen

Erweiterung

- Benutzung von MIME (*Multipurpose Internet Mail Extension*)
- Akzente, nichtlateinische Sprachen usw. jetzt darstellbar
- Kombination von Text und multimedialen Inhalten erlaubt
- Art des Formats wird zu Beginn der Nachricht spezifiziert

Ablauf einer Kommunikation via SMTP

1. Client spricht Mail-Server auf Port 25 an

```
telnet xy.tu-berlin.de 25
```

2. Server gibt Antwort, falls SMTP-Dienst verfügbar

```
220 xy.tu-berlin.de ESMTP Sendmail 8.11.6/8.9.3;
Mon, 25 Jun 2012 7:55:20 + 0200
```

3. Fortsetzung der Verbindung durch Client mit Befehl *HELO*

```
HELO z.tu-berlin.de
```

4. Falls Server akzeptiert, gibt es eine Antwort

```
250 xy.tu-berlin.de Hello z.tu-berlin.de[139.235.66.186]
```

5. Client erfragt verfügbare Befehle mit *HELP*
6. Antwort des Servers

```
214-Commands supported:
214 AUTH STARTTLS HELO EHLO MAIL RCPT DATA ...
```

Versenden von E-Mails via SMTP

7 Versenden einer E-Mail mit den Befehlen *MAIL FROM*, *RCPT TO*, *DATA* und *QUIT*

```
MAIL FROM: helke@tu-berlin.de
250 OK
```

```
RCPT TO: steffen.helke@tu-berlin.de
250 Accepted
```

```
DATA
354 Enter message, ending with "." on line by itself
```

```
Dies ist eine Testmail fuer die Vorlesung PROG 2
.
250 OK id=1Sj2Rw-0003Wl-Aj
```

```
QUIT
221 xy.tu-berlin.de closing connection
Connection closed by foreign host.
```

Wie wird ein SMTP-Client in Java umgesetzt?

Varianten

1. Stream-Socket-Verbindung mit der Klasse *Stream* und *selbst zusammengebauten SMTP-Anfrage-String*

```
"HELO " + InetAddress.getLocalHost().getHostName();

"MAIL from: " + sender

"RCPT to: " + receiver

"DATA " + Textnachricht + "."
```

2. Benutzung von *komfortablen Methoden* der *JavaMail*-API
   - ➜ schrittweises Erstellen von Nachrichten
   - ➜ Versenden per SMTP, POP, IMAP oder NNTP

Variante 1: Zugriff auf Mail-Server

```
public class SmtpMail {

  protected void send(String serverName,int port,
                      String sender,String receiver,
                      String subject,String data) { ... }

  public static void main(String[] args) {

  String serverName = "mail.tu-berlin.de";
  int serverPort = 25;
  String sender = "helke@tu-berlin.de";
  String receiver = "steffen.helke@tu-berlin.de";
  String subject = "Testmail VL PROG 2";
  String data = "Testmail aus der Vorlesung PROG 2."

  SmtpMail smtp = new SmtpMail();

  try {
    smtp.send(serverName,serverPort,sender,
              receiver,subject,data);
  } catch (...) { ... }}}
```

Variante 1: Mail mit SMTP selbst versenden

```
1  protected void send(String server,int port,
2                      String sender,String receiver,
3                      String subject,String data) ... {
4   Socket smtpSocket = null;
5   try {
6    InetAddress serverAddr = InetAddress.getByName(server);
7    String client = InetAddress.getLocalHost().getHostName();
8    smtpSocket = new Socket(serverAddr,port);
9    InputStream ins = smtpSocket.getInputStream();
10   InputStreamReader inr = new InputStreamReader(ins);
11   BufferedReader in = new BufferedReader(inr);
12   OutputStream outs = smtpSocket.getOutputStream();
13   PrintWriter out = new PrintWriter(outs);
14   // Abbildung Protokoll HELO, MAIL from, RCPT to und DATA
15   out.write("HELO " + client);out.write("\r\n");out.flush();
16   ..., out.write("DATA");out.write("\r\n");out.flush();
17   // Sende Header
18   out.write("From: " + sender + "\r\n");
19   out.write(data); out.write("\r\n.\r\n");out.flush();
20   out.write("QUIT");out.write("\r\n");out.flush();
21  } catch ( ... ) { ... }
```

Variante 2: Mail mit JavaMail verschicken

➜ zur Benutzung von JavaMail wird *mail.jar* benötigt

Vorgehen

1. Definition einer Session mit wichtigen Eigenschaften

```
Properties prop = new Properties();
props.put("mail.host","mail.tu-berlin.de");
// null ist der fehlende Authentikator
Session mailConnect = Session.getInstance(props,null);
Message message = new MimeMessage(mailConnect);
```

2. Nachricht (MIME) erstellen

```
message.setContent("Hello","text/plain");
// wenn nur Text einzufuegen ist
message.setText("Hello");
message.setSubject("First");
```

Variante 2: Mail mit JavaMail verschicken

Vorgehen (Fortsetzung)

3. Festlegen der Adresse

```
Address toAddr =
    new InternetAddress(steffen.helke@tu-berlin.de,
                        "Steffen Helke");
Address ccAddr =
    new InternetAddress(helke@cs.tu-berlin.de");
message.addRecipient(Message.RecipientType.TO,toAddr);
message.addRecipient(Message.RecipientType.CC,ccAddr);
```

4. Versenden der Nachricht (hier mit SMTP)

```
messages.saveChanges();
transport = session.getTransport("smtp");
transport.connect(host,username,password);
transport.sendMessage(message,
                      message.getAllRecipients());
transport.close();
```

Teil V der Vorlesung PROG 2

Netzwerkprogrammierung

RMI – Remote Method Invocation

➜ nächste Vorlesung

Quelle: *Inhalt & Gestaltung nach Vorlesungsfolien von Peter Pepper und Odej Kao, TU Berlin*
Methodische- und Praktische Grundlagen der Informatik 4 (MPGI 4), WS 2010/11 bzw. WS 2011/12