Kapitel WT:II

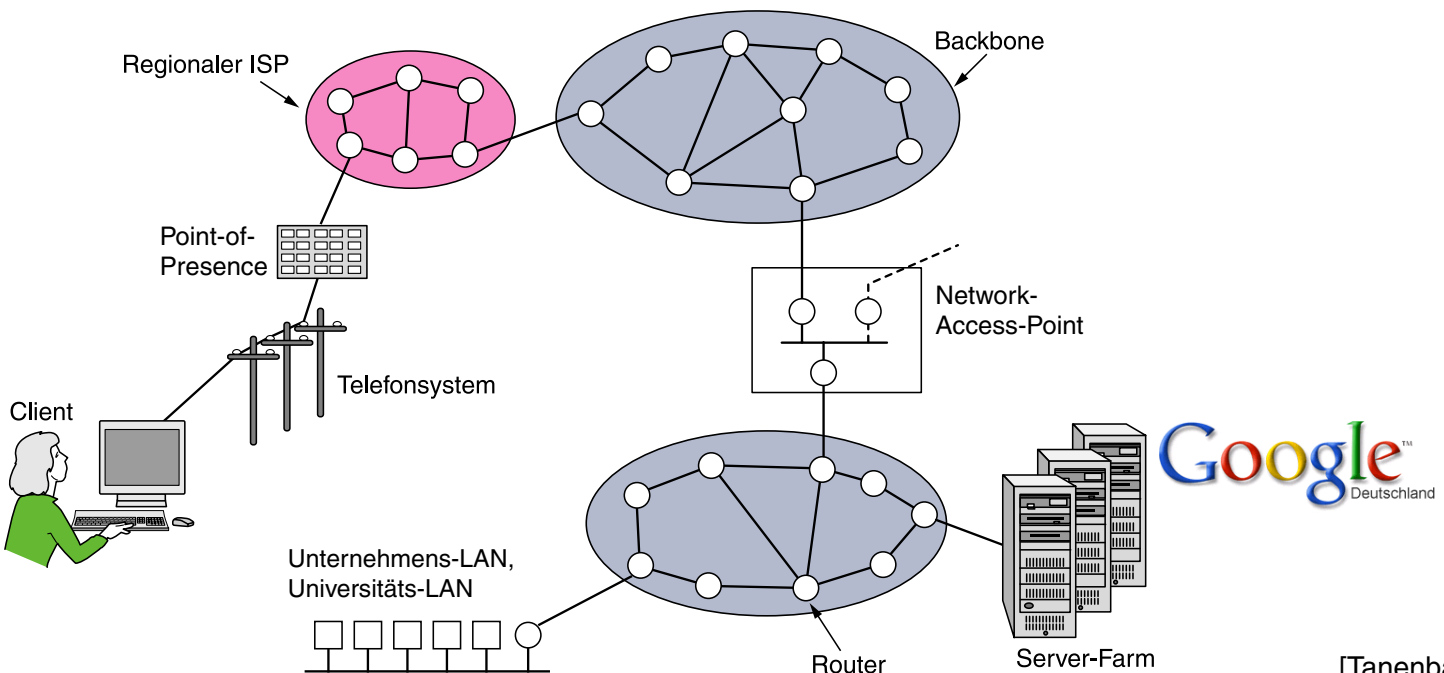
II. Rechnerkommunikation und Protokolle

- Rechnernetze
- Prinzipien des Datenaustauschs
- Netzsoftware und Kommunikationsprotokolle
- Internetworking
- Client-Server-Interaktionsmodell
- Uniform Resource Locator
- Hypertext-Transfer-Protokoll HTTP
- Fortgeschrittene HTTP-Konzepte

Rechnernetze [Tanenbaum]

Eigenschaften von Rechnernetzen

- Rechner arbeiten quasi autonom
- Rechner sind miteinander verbunden und können Informationen austauschen
- Probleme durch Verzögerungen und Fehler des Kommunikationskanals werden weitestgehend eliminiert



[Tanenbaum]

Broadcasting:

- *ein* Übertragungskanal, der von allen Netzkomponenten genutzt wird
- Nachrichten (Pakete) werden von einer Station an alle anderen Stationen gesendet; Stationen senden abwechselnd
- je nach Adressierung wird die Nachricht von nur einer Station (*unicast*), mehreren Stationen (*multicast*) oder allen Stationen (*broadcast*) verarbeitet

Rechnernetze [Tanenbaum]

Übertragungstechnik

Broadcasting:

- *ein* Übertragungskanal, der von allen Netzkomponenten genutzt wird
- Nachrichten (Pakete) werden von einer Station an alle anderen Stationen gesendet; Stationen senden abwechselnd
- je nach Adressierung wird die Nachricht von nur einer Station (*unicast*), mehreren Stationen (*multicast*) oder allen Stationen (*broadcast*) verarbeitet

Punkt-zu-Punkt:

- zwei miteinander verbundene Stationen: eigener Übertragungskanal
- zwei nicht benachbarte Stationen: verschiedene Routen möglich
  ➜ Wegfindung (*routing*) wichtig
- ein Paket wird in der Regel für eine bestimmte Station adressiert

Rechnernetze [Tanenbaum]

Klassifikation

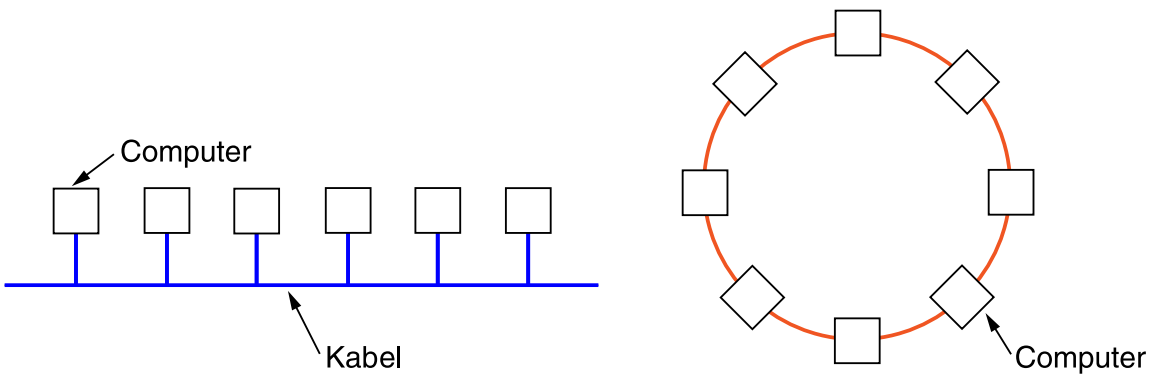
Klassifikation anhand der räumlichen Ausdehnung:

Entfernung	Organisation	Beispiel	Abkürzung
1m	nächste Umgebung	persönliches Netz	PAN
10m	Raum	lokales Netz	LAN
100m	Gebäude		
1km	Liegenschaft		
10km	Stadt	Stadtnetz	MAN
100km	Land	Fernnetz	WAN
1000km	Kontinent		
10.000km	Planet	Internet	

Rechnernetze [Tanenbaum]

Klassifikation

Klassifikation anhand der Topologie (hier LAN):

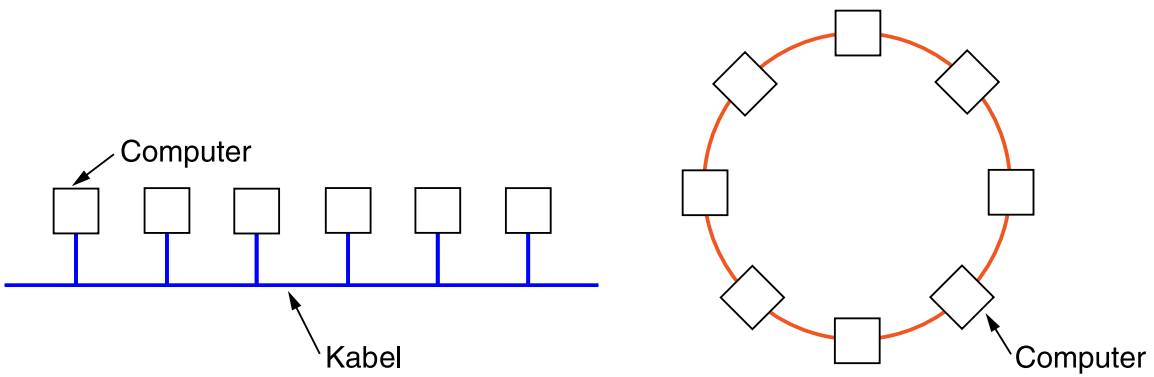


[Tanenbaum]

Rechnernetze [Tanenbaum]

Klassifikation

Klassifikation anhand der Topologie (hier LAN):

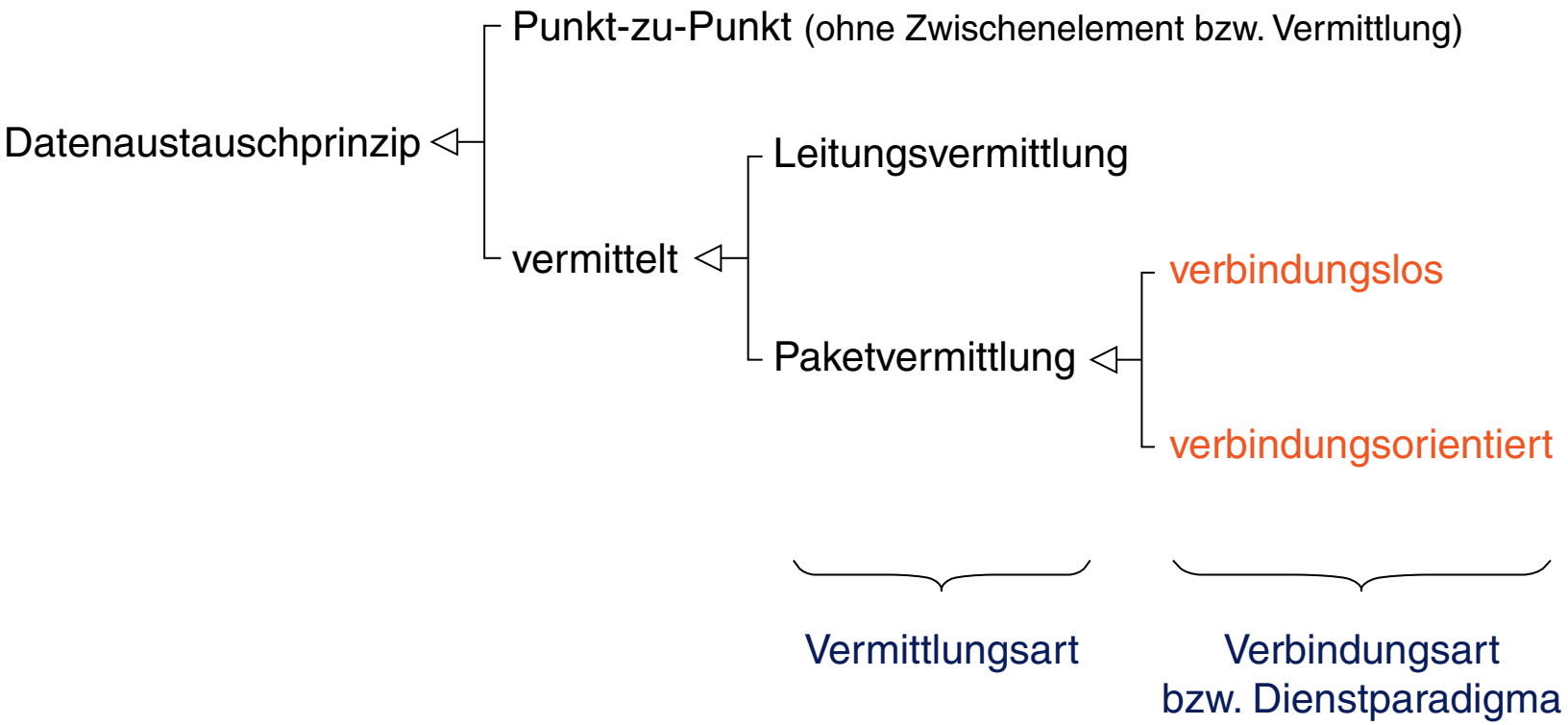


[Tanenbaum]

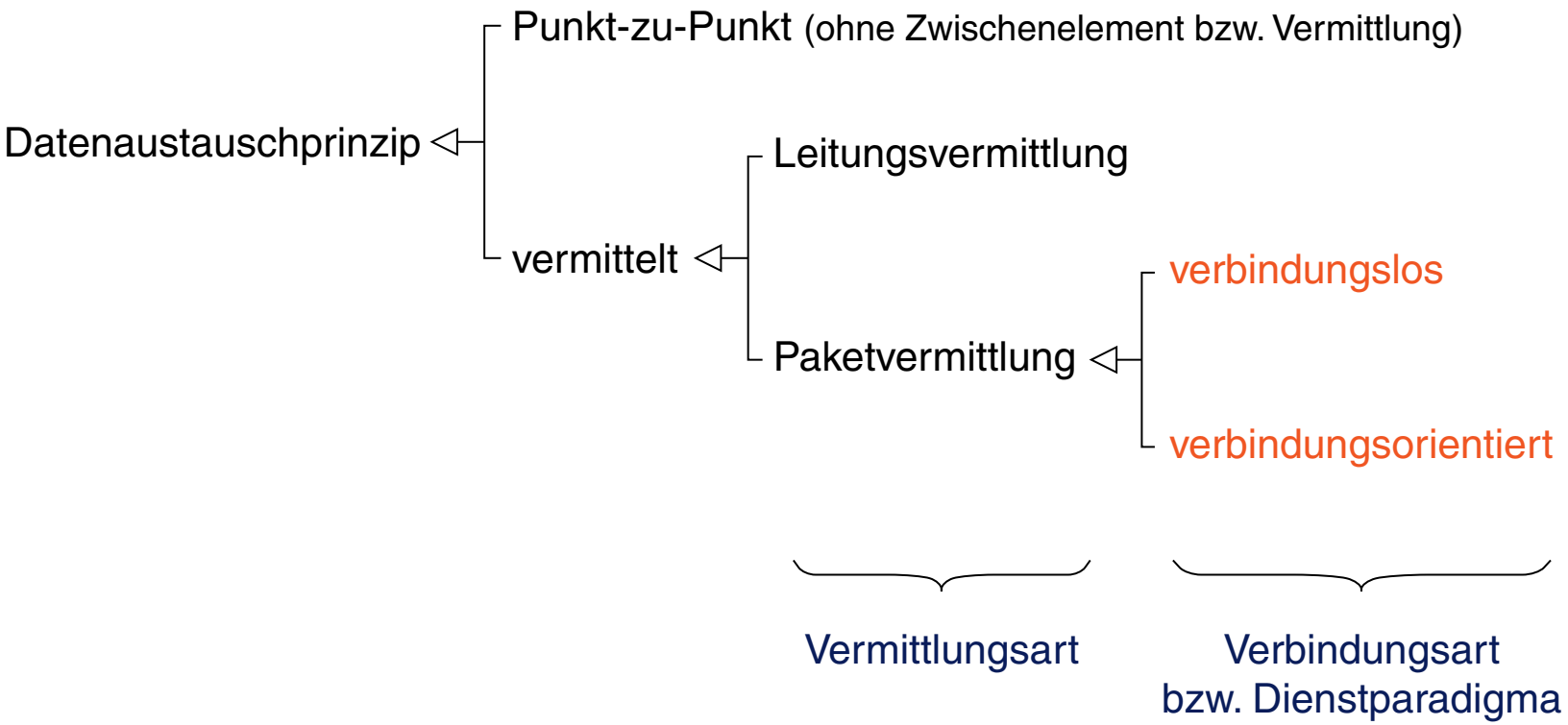
Weitere Klassifikationsmöglichkeiten:

- anhand des Übertragungsmediums: Twisted-Pair, Glasfaser, Infrarot, etc.
- anhand des Übertragungsprotokolls: Ethernet, Tokenring, FDDI, ATM, etc.
- anhand der Trägerschaft: öffentlich, privat
- anhand der Einsatzcharakteristik: Funktionsverbund, Lastverbund, Nachrichtenverbund, Sicherheitsverbund

Prinzipien des Datenaustauschs



Prinzipien des Datenaustauschs



Definition 8 (Verbindung)

Eine Verbindung ist eine Beziehung zwischen zwei kommunizierenden Stationen über einen bestimmten Zeitraum.

Prinzipien des Datenaustauschs [Tanenbaum]

Vermittlungsart

Punkt-zu-Punkt-Verbindung ohne Vermittlung:

- ❑ je zwei Rechner permanent miteinander verbunden
- \+ Kommunikation einfach
- – Verkabelungsaufwand wächst quadratisch in der Rechneranzahl

Leitungsvermittlung (*Switching Network*):

- ❑ Schaltung einer festen Verbindung durch Vermittlungsstellen
- ❑ Beispiel: analoges Telefonnetz
- \+ Teilnehmer erhalten feste Bandbreite zur alleinigen Nutzung
- \+ Kommunikation einfach
- – ungenutzte Übertragungskapazitäten
- – Aufbau von Verbindungen ist zeitintensiv
- – Ausfall von Vermittlungsstellen legt Teile des Netzes lahm

Prinzipien des Datenaustauschs [Tanenbaum]

Vermittlungsart

Paketvermittlung:

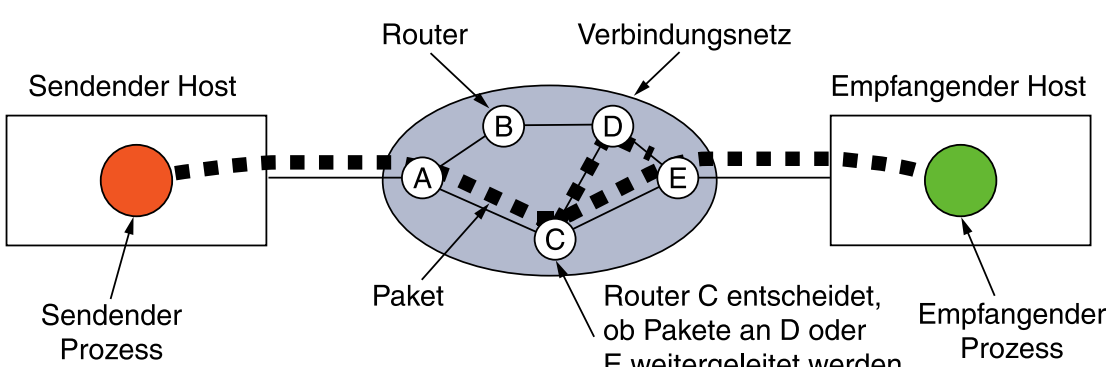
- ❑ Zerlegung einer Nachricht in individuell adressierte Pakete
- ❑ Datenpakete werden in Netzknoten zwischengespeichert (*store and forward*)
  ➜ Verzögerungen möglich, aber bessere Ausnutzung der Übertragungskanäle
- ❑ für jedes korrekt empfangene Paket kann Quittung an den Sender geschickt werden; keine Quittung bei Fehlern oder Paketverlust ➜ Paket wird erneut gesendet

\+ faire Ressourcenzuteilung wird möglich

\+ deutlich erhöhte Ausfallsicherheit

– aufwändiges Kommunikationsprotokoll

– keine (unmittelbar) garantierte Dienstgüte



[Tanenbaum]

Prinzipien des Datenaustauschs [Tanenbaum]

Verbindungsart bzw. Dienstparadigma

Verbindungslose Kommunikation:

- Daten werden ohne Vorankündigung zur Übertragung übergeben und von Netzwerkknoten zu Netzwerkknoten übertragen
- kein initialer Kontakt zwischen Sender und Empfänger
- keine Garantie, dass die gesendeten Daten den Empfänger erreichen
- Analogie: Briefzustellung durch die Post

Prinzipien des Datenaustauschs [Tanenbaum]

Verbindungsart bzw. Dienstparadigma

Verbindungslose Kommunikation:

- Daten werden ohne Vorankündigung zur Übertragung übergeben und von Netzwerkknoten zu Netzwerkknoten übertragen
- kein initialer Kontakt zwischen Sender und Empfänger
- keine Garantie, dass die gesendeten Daten den Empfänger erreichen
- Analogie: Briefzustellung durch die Post

+/– keine Reservierung von Ressourcen
➜ Variationen der Zustellungsgeschwindigkeit und -qualität möglich

\+ kein Verwaltungsaufwand durch Verbindungsaufbau

– Adressierung der Daten komplizierter, da Pakete unabhängig voneinander durch das Netz befördert werden

Prinzipien des Datenaustauschs [Tanenbaum]

Verbindungsart bzw. Dienstparadigma

Verbindungsorientierte Kommunikation:

- Aufstellung einer definierten Verbindung zwischen Stationen notwendig
- Datenaustausch in drei Phasen:
  1. Verbindungsaufbau (*connect, set-up*). Sender spricht den Empfänger an, sendet Authentifizierungsdaten und verlangt den Verbindungsaufbau
  2. Uni- oder bidirektionalerDatenaustausch (*data transfer*).
  3. Verbindungsabbau (*disconnect*).
- Analogie Telefonieren:
  Anrufen = connect, Abnehmen = Kommunikationserlaubnis, Stimme = Authentifizierung,
  Gespräch = Datenaustausch,
  Auflegen = disconnect

Prinzipien des Datenaustauschs [Tanenbaum]

Verbindungsart bzw. Dienstparadigma

Verbindungsorientierte Kommunikation:

- Aufstellung einer definierten Verbindung zwischen Stationen notwendig
- Datenaustausch in drei Phasen:
  1. Verbindungsaufbau (*connect, set-up*). Sender spricht den Empfänger an, sendet Authentifizierungsdaten und verlangt den Verbindungsaufbau
  2. Uni- oder bidirektionalerDatenaustausch (*data transfer*).
  3. Verbindungsabbau (*disconnect*).
- Analogie Telefonieren:
  Anrufen = connect, Abnehmen = Kommunikationserlaubnis, Stimme = Authentifizierung,
  Gespräch = Datenaustausch,
  Auflegen = disconnect

\+ steht eine Verbindung, ist der Datenaustausch einfach: Empfänger gefunden, Reihenfolge bleibt erhalten, Ressourcen reserviert, etc.

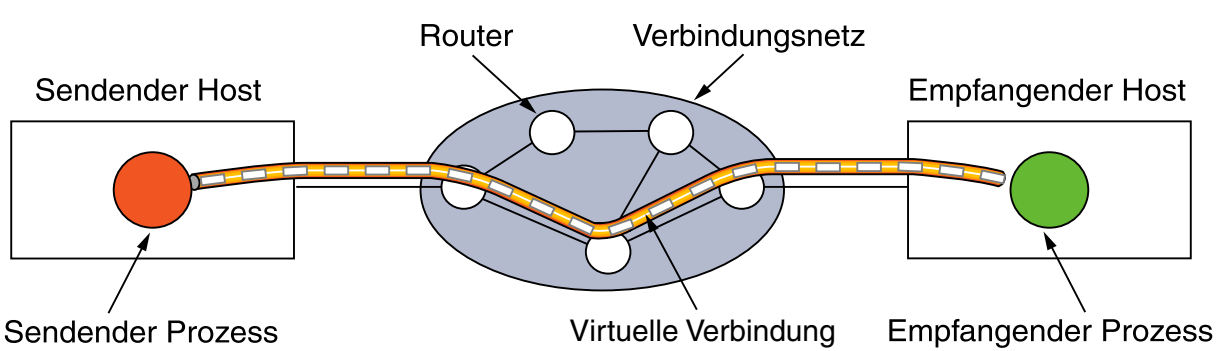
– der Aufbau einer Verbindung ist komplex und zeitintensiv, insbesondere wenn viele Stationen involviert sind

Prinzipien des Datenaustauschs [Tanenbaum]

Verbindungsart bzw. Dienstparadigma

Verbindungsorientierte Kommunikation: (Fortsetzung)

- die Einrichtung einer Verbindung stellt sicher, dass alle Daten den Empfänger erreichen – und in der richtigen Reihenfolge
- Herausforderung: verbindungsorientierte Dienste über paketvermittelte Netzwerke müssen auf den dort verfügbaren Diensten aufsetzen
  ➜ *die Verbindung ist nur virtuell*
- für die Dauer einer Verbindung werden benötigte Ressourcen im Netz reserviert (Speicher in Zwischenknoten, Übertragungskapazität, etc.)
  ➜ garantierte Dienstgüte, z.B. zur Auslieferung von Videos



[Tanenbaum]

Prinzipien des Datenaustauschs [Tanenbaum]

Dienste

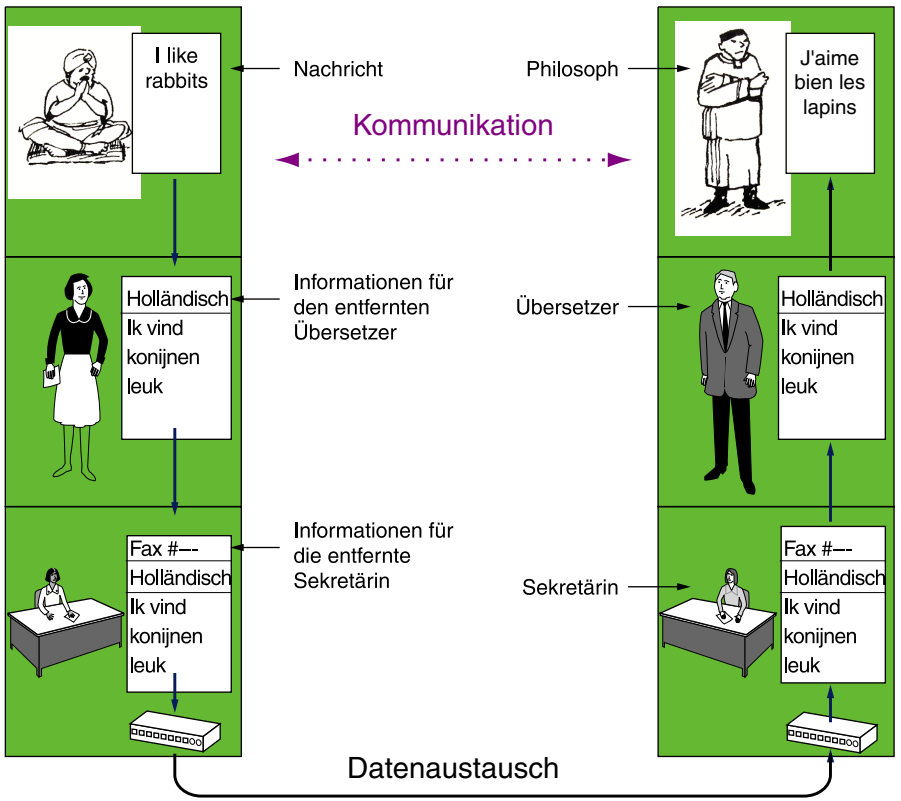
Verbindungsart und Zuverlässigkeit sind orthogonal (= bedingen sich nicht):

Dienst	**Verbindungsart**	**Beispiel**
zuverlässiger Nachrichtenstrom	verbindungsorientiert	Folge von Seiten
zuverlässiger Bytestrom		Remote-Terminal
unzuverlässige Verbindung		digitalisierte Sprache
unzuverlässiges Datagramm	verbindungslos	Junk-E-Mail
bestätigtes Datagramm		registrierte E-Mail
Anforderung/Antwort		Datenbankanfrage

Netzsoftware und Kommunikationsprotokolle

Herstellung einer Verbindung und Datenaustausch $\neq$ Kommunikation.

Ziel ist es, sich zu verstehen, zu kommunizieren . . .

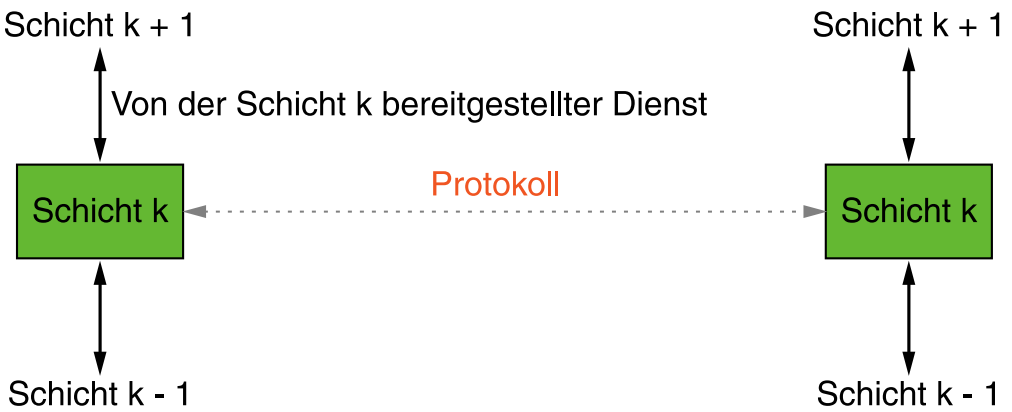


[Tanenbaum]

Für Rechner definieren Übertragungsprotokolle die Regeln der Kommunikation.

Netzsoftware und Kommunikationsprotokolle

Die Netzsoftware ist eine in Schichten organisierte, komplexe Software, die die gesamte Umsetzung der Kommunikation übernimmt: von der Anwendung über die Protokolle bis zu Steuerung der Netz-Hardware.



[Tanenbaum]

Netzsoftware und Kommunikationsprotokolle

Die Netzsoftware ist eine in Schichten organisierte, komplexe Software, die die gesamte Umsetzung der Kommunikation übernimmt: von der Anwendung über die Protokolle bis zu Steuerung der Netz-Hardware.

[Tanenbaum]

Definition 9 (Netzsoftware-Dienst)

Ein Dienst fungiert als Schnittstelle zwischen zwei Schichten und bezeichnet eine Gruppe von Operationen, die eine Schicht der ihr überliegenden Schicht anbietet.

Definition 10 (Netzsoftware-Protokoll)

Ein Protokoll ist eine Menge von Regeln, die Format (Syntax) und Bedeutung (Semantik) der von gleichgestellten Schichten ausgetauschten Pakete festlegen.

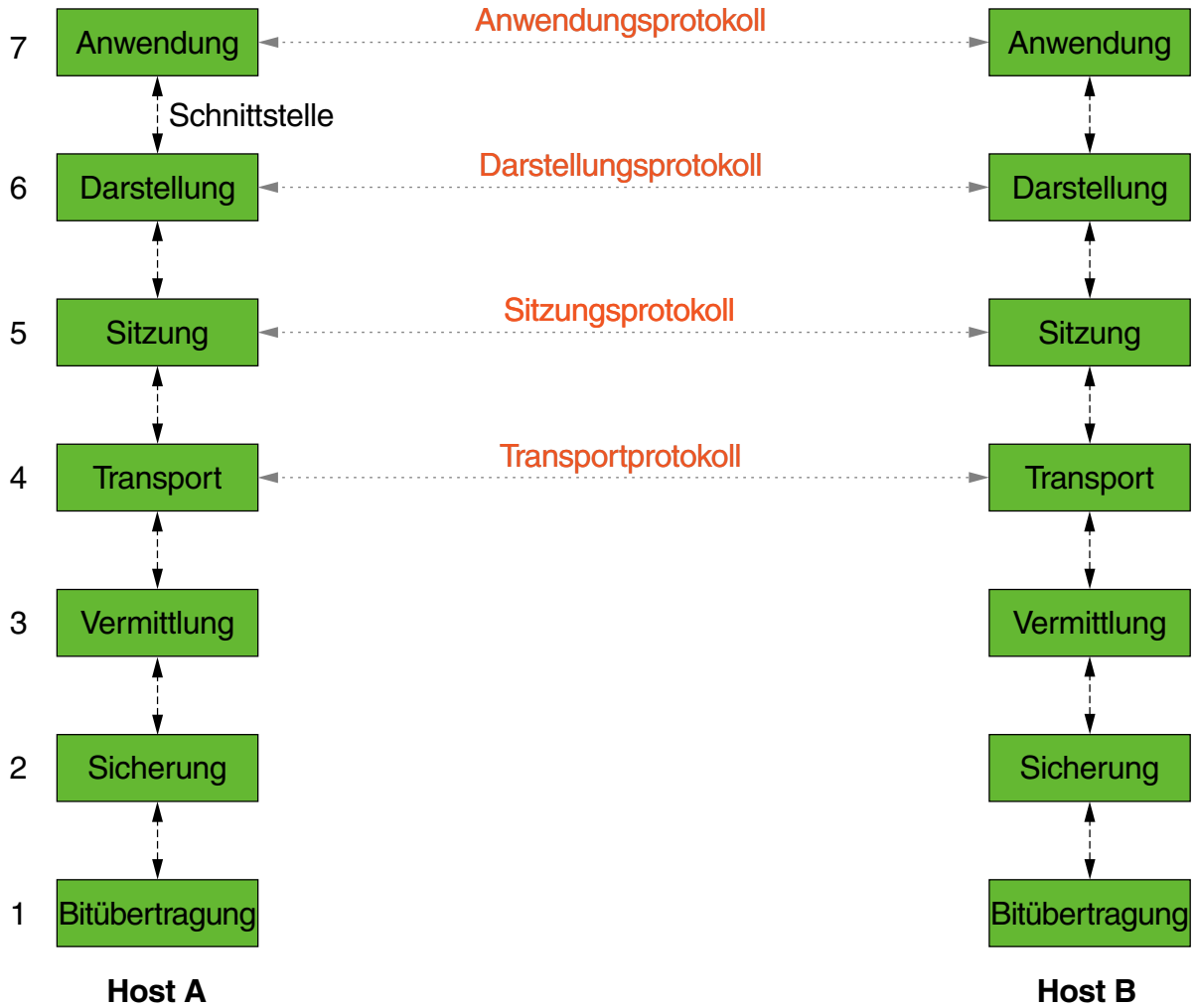
Netzsoftware und Kommunikationsprotokolle

ISO-OSI-Modell [related]

[Tanenbaum]

Netzsoftware und Kommunikationsprotokolle

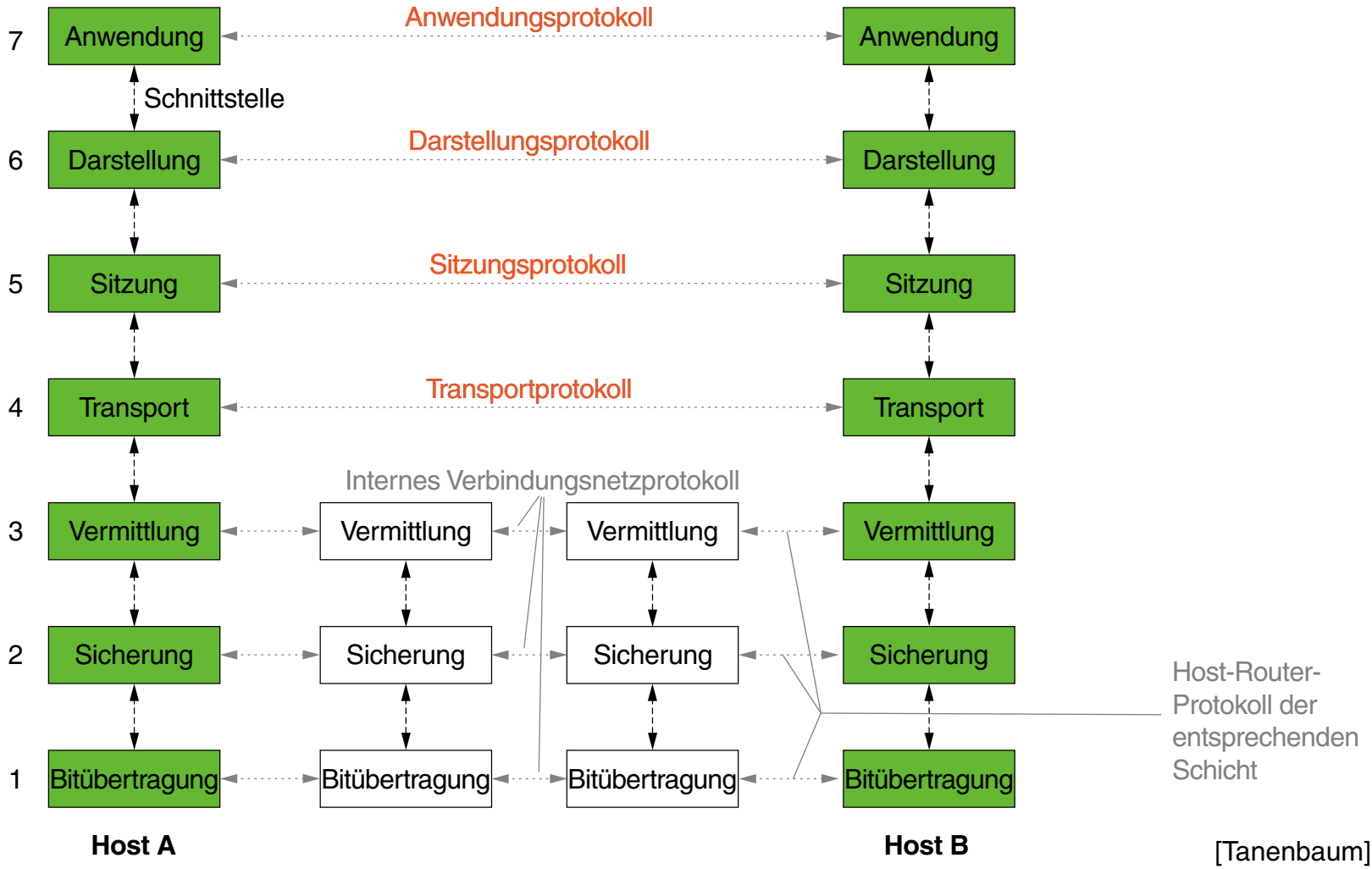
ISO-OSI-Modell [related]



[Tanenbaum]

Netzsoftware und Kommunikationsprotokolle

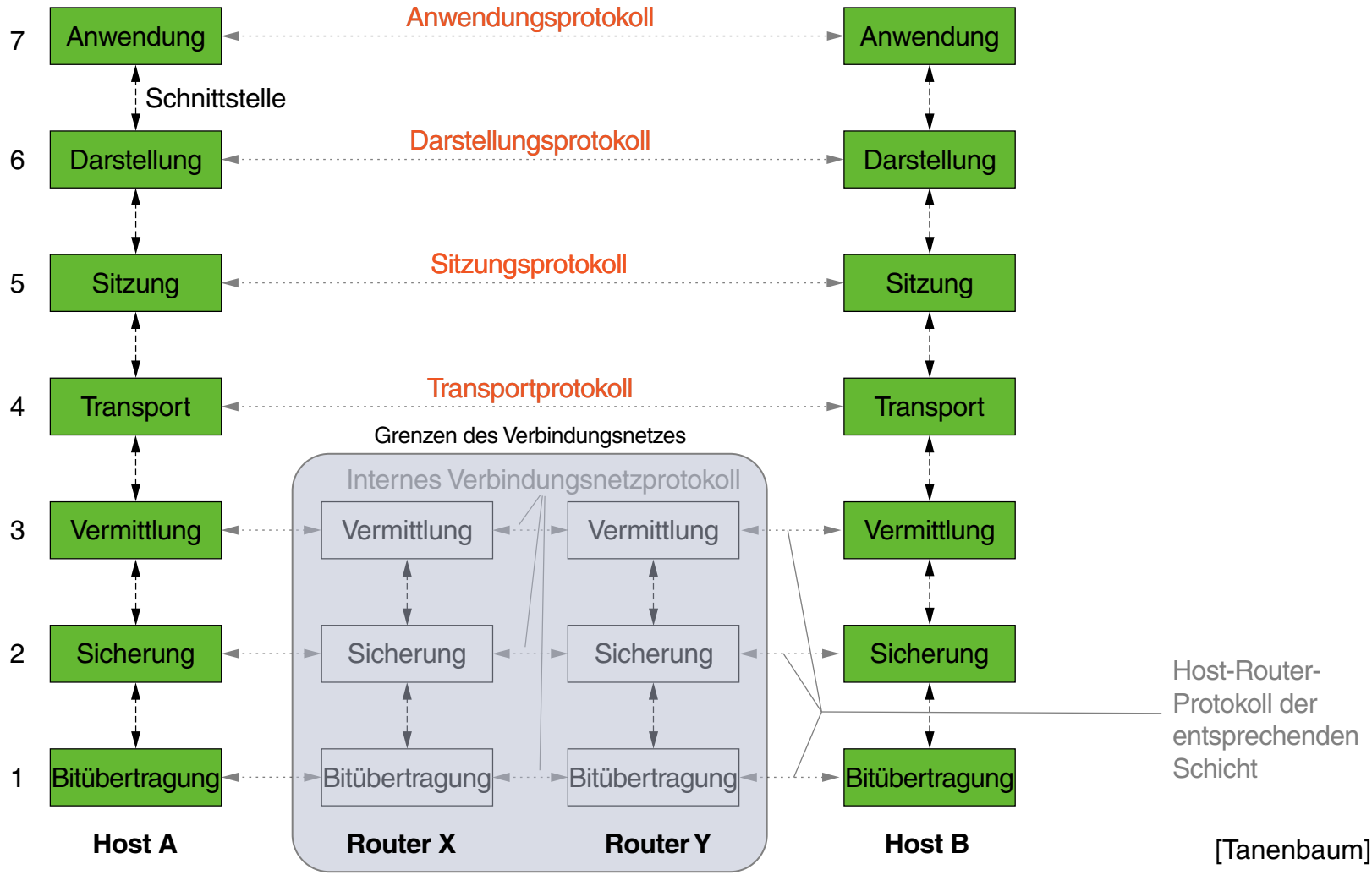
ISO-OSI-Modell [related]



[Tanenbaum]

Netzsoftware und Kommunikationsprotokolle

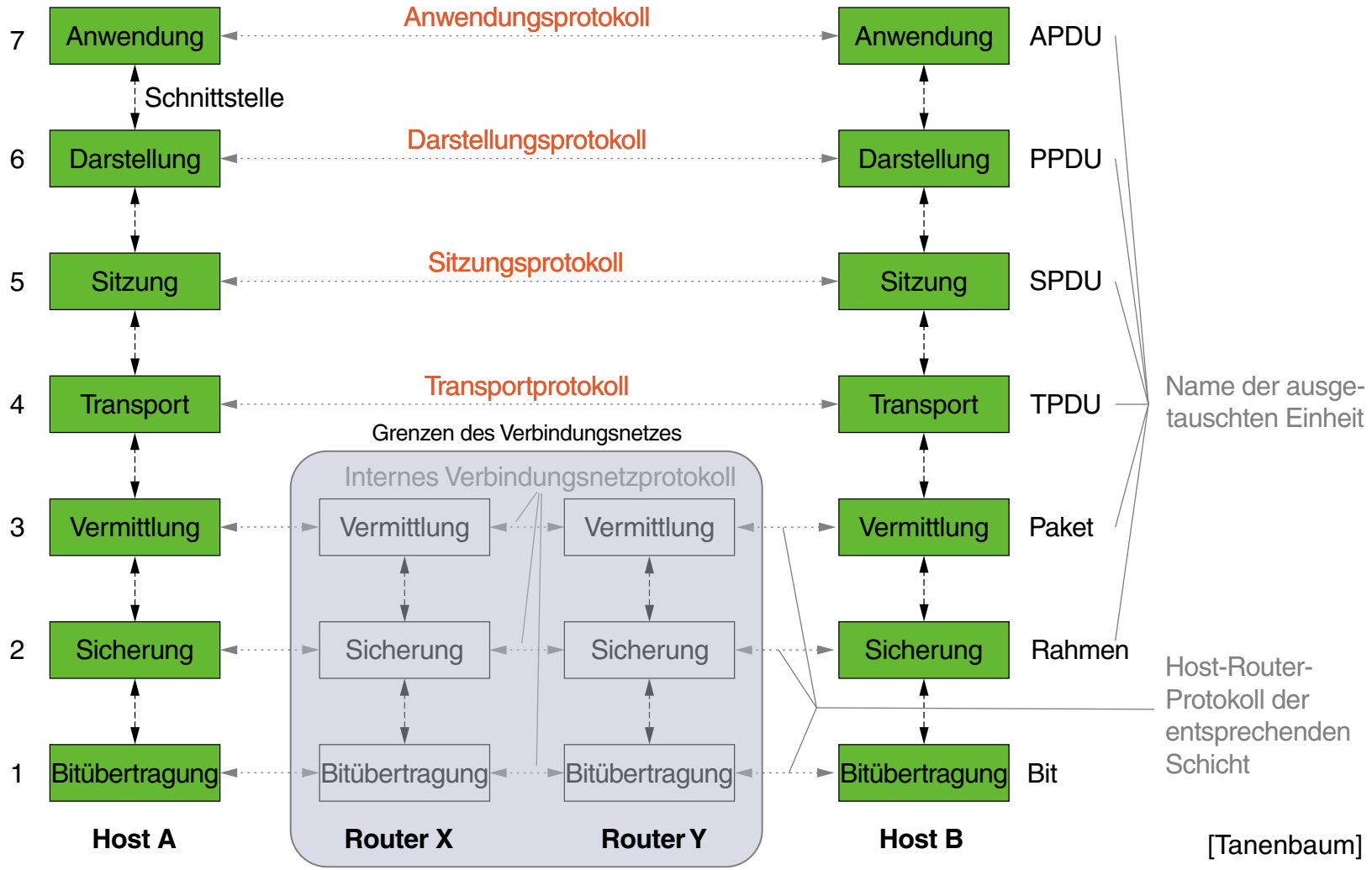
ISO-OSI-Modell [related]



[Tanenbaum]

Netzsoftware und Kommunikationsprotokolle

ISO-OSI-Modell [related]



[Tanenbaum]

Bemerkungen:

- Dienste und Protokolle sind unabhängig voneinander: gleichgestellte Schichten können ihre Protokolle nach Belieben ändern, solange die für den Dienstnutzer sichtbaren Dienste unverändert bleiben.
- Abkürzungen für die Datenpakettypen:
  - APDU = Application Protocol Data Unit
  - PPDU = Presentation Protocol Data Unit
  - SPDU = Session Protocol Data Unit
  - TPDU = Transport Protocol Data Unit
- ISO = International Organization for Standardization, OSI = Open Systems Interconnection.

Netzsoftware und Kommunikationsprotokolle

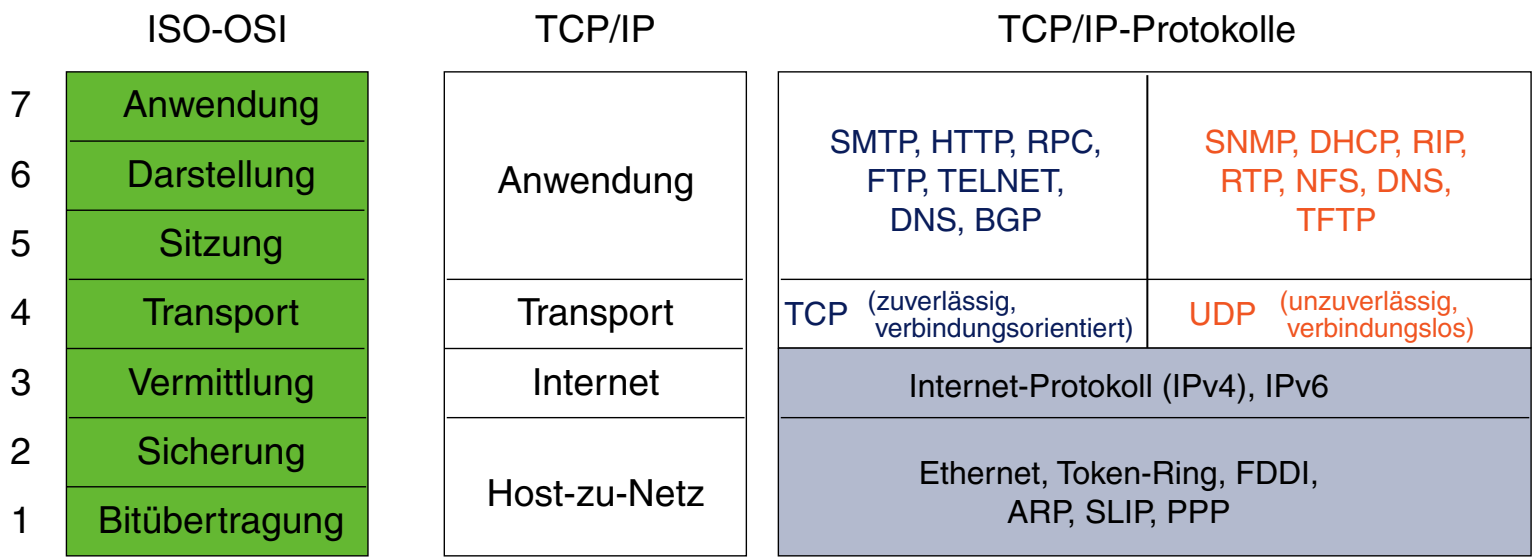
ISO-OSI versus TCP/IP Protokoll-Stack

Aufgaben der Netzsoftware im ISO-OSI-Modell:

- Schicht 4. Fehlerfreie Endpunkt-zu-Endpunkt-Übertragung.
- Schicht 3. Routing, Flusskontrolle, Qualitätssicherung.
- Schicht 2. Sicherungsmaßnahmen, Fehlerbehandlung, Medienzuteilung.
- Schicht 1. Übertragung der Bits über einen Kommunikationskanal.

Netzsoftware und Kommunikationsprotokolle

ISO-OSI versus TCP/IP Protokoll-Stack



Aufgaben der Netzsoftware im TCP/IP-Modell:

- Schicht 4. Fehlerfreie Endpunkt-zu-Endpunkt-Übertragung.
- Schicht 3. Netzübergreifende Zustellung von IP-Paketen.
- Schicht 1+2. Verbindung zum Netz und Versenden von IP-Paketen.

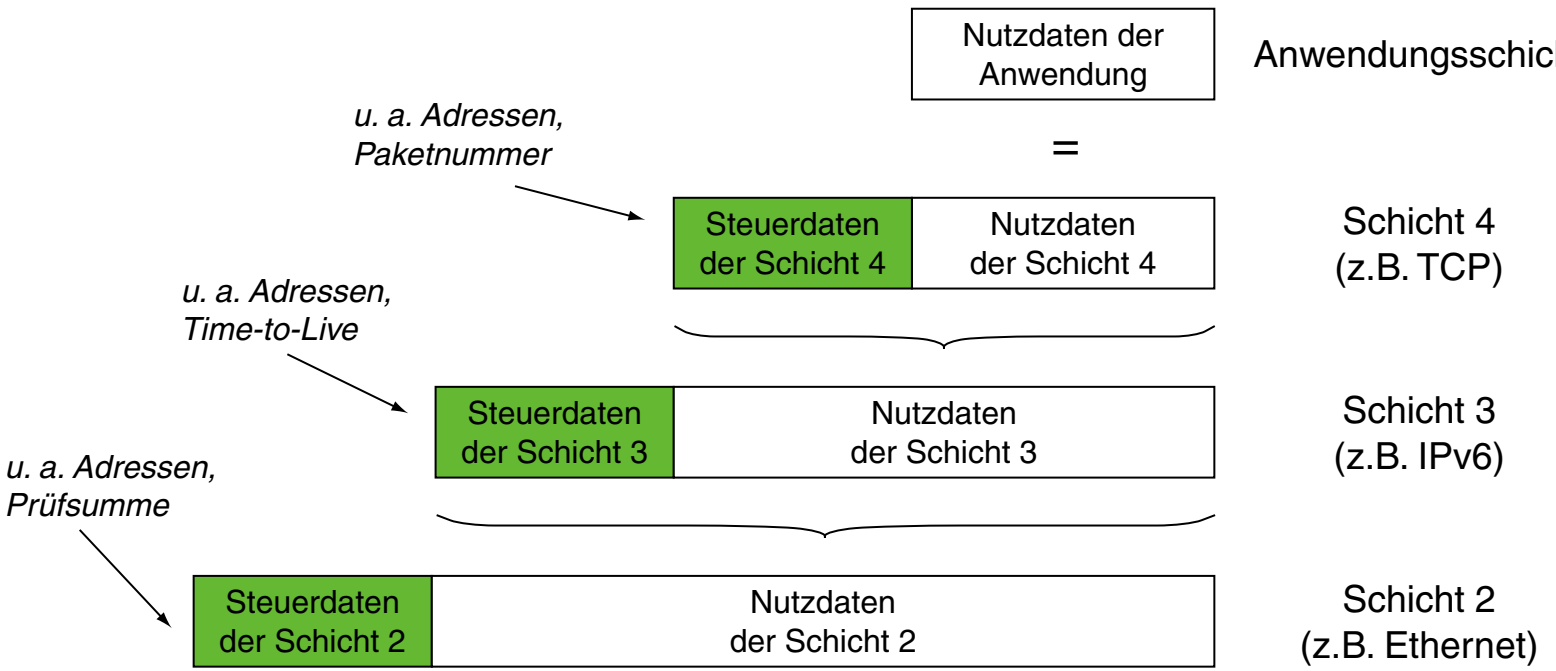
Bemerkungen:

- TCP/IP ist die Abkürzung für Transmission Control Protocol/Internet Protocol.
- Ein grundsätzliches Designprinzip des TCP/IP-Modells ist die Verlagerung der Verantwortung für die Zuverlässigkeit vom physischen Netzwerk auf die einzelnen Hostrechner. Die radikale Reduktion der Aufgaben des physischen Netzwerks vereinfacht das Zusammenschließen von unterschiedlichen Netzwerken erheblich.
- Das TCP/IP-Modell hat in der Vermittlungsschicht (Schicht 3) nur einen verbindungslosen Kommunikationsmodus, unterstützt in der Transportschicht (Schicht 4) aber sowohl verbindungslose als auch verbindungsorientierte Kommunikation.
- Die Protokolle der Transportschicht (Schicht 4) gehören zu den kompliziertesten.
- Siehe die Übersichtsbox des TCP/IP-Modells in der deutschen Wikipedia: IP, TCP, FTP

Netzsoftware und Kommunikationsprotokolle

Datenfragmentierung und Kapselung

Prinzip: Nachrichten aus höheren Schichten werden als Nutzdaten für die unteren Schichten eingesetzt.



Internetworking

- Zahlreiche Netzwerke mit unterschiedlichen Technologien sind im Internet zu einem homogen erscheinenden Netzwerk zusammengeschaltet.
- Internetworking = Kommunikation über unterschiedliche Rechnernetze
- Internetworking wird durch ein einheitliches Protokoll oberhalb der technologiegebundenen Schicht realisiert.

„Das Internet ist ein reines Software-Produkt.“

[Meinel/Sack 2004]

Internetworking [Meinel/Sack 2004]

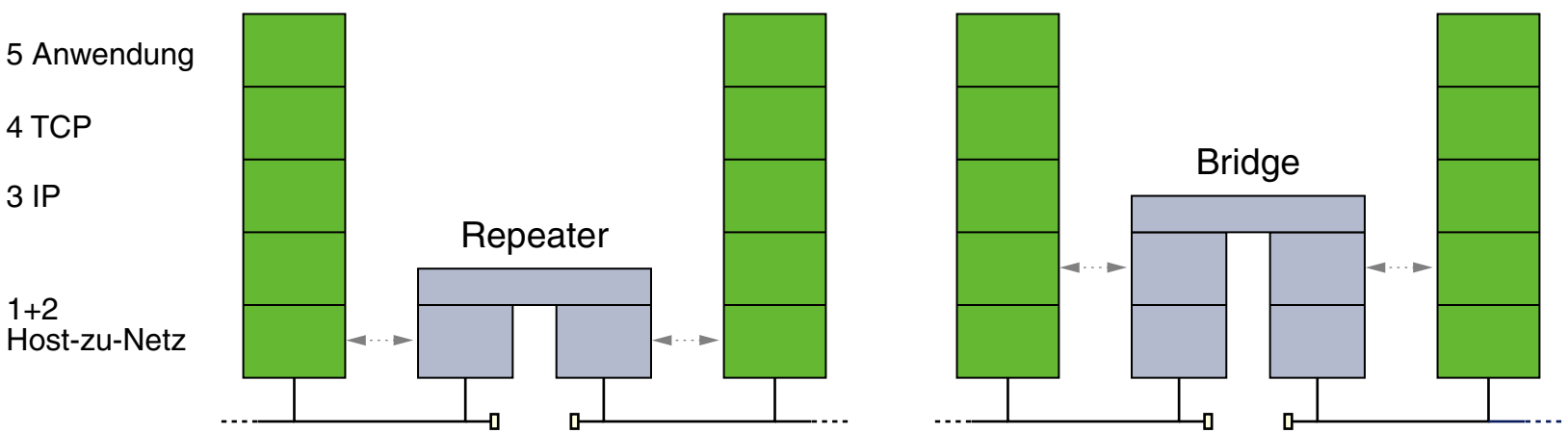
Vermittlungssysteme im Internet

- Repeater

  Arbeitet auf der physikalischen Schicht (Schicht 1); bewirkt reine Signalverstärkung für größere Distanzen.

- Bridge

  Verbindet Netzsegmente auf der Sicherungs- bzw. Bitübertragungsschicht (Schicht 2); dient zur Erweiterung von LANs; leistet Verkehrsmanagement.



Internetworking [Meinel/Sack 2004]

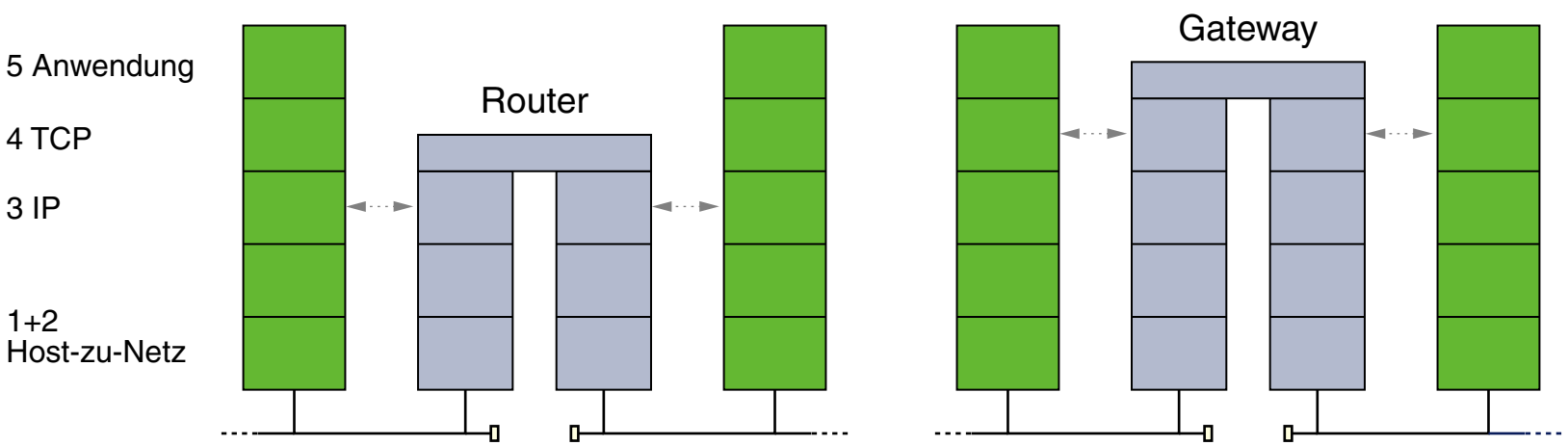
Vermittlungssysteme im Internet

- Router

  Verbindet einzelne LANs miteinander, die von verschiedenem Typ sein können; sind vom Netzprotokoll abhängig.

- Gateway

  Verbindet Netzwerke; ermöglicht Kommunikation zwischen Anwendungsprogrammen.



Internetworking
MAC-Adressierung

- MAC-Adresse = Medium Access Layer Address = Hardware-Adresse eines Netzwerkgeräts (Netzwerkkarte, Switch, etc.)
- dient zur eindeutigen Identifikation des Netzwerkgeräts im Netzwerk
- wird beim Einschalten gesetzt und kann danach in der Regel nicht mehr verändert werden
- Das Internet-Protokoll (IPv4) verwendet eine dynamische Zuordnung von MAC-Adressen zu Internet-Adressen. Basis ist das *Address Resolution Protocol* ARP.

Bemerkungen:

- Aufbau einer MAC-Adresse bei der Ethernet-Technologie:
  - Länge 48 Bit
  - Darstellung hexadezimal, Beispiel: 08-00-20-ae-fd-7e
  - Die Bits 1-24 enthalten die von der IEEE vergebene Herstellerkennung, die Bits 25-48 sind herstellerintern verwendbar.
- Statische MAC-Adressen sind weltweit eindeutig und dienen zur automatischen Gerätekonfiguration und als Basis für Protokolle wie DHCP.
- Unter IPv6 ermöglicht die Erzeugung des Interface Identifiers aus der MAC-Adresse die Identifizierung von Benutzern. Deshalb wurden in RFC 4941 sogenannte Privacy Extensions spezifiziert.

Internetworking

IP-Adressierung mit IPv4

141.54.1.11

- IPv4-Adressen bestehen aus 32 Bit bzw. 4 Bytes, angegeben als Folge von 4 ganzzahligen, durch Dezimalpunkte getrennte Dezimalzahlen.
- IPv4-Adressen sind in zwei Teile gegliedert: Adresspräfix und Adresssuffix.
- Adresspräfix (Netzwerk-ID) identifiziert das physikalische Netzwerk.
- Adresssuffix (Host-ID) identifiziert Rechner im Netzwerk der Netzwerk-ID.

Internetworking

IP-Adressierung mit IPv4

`141.54.1.11`

- IPv4-Adressen bestehen aus 32 Bit bzw. 4 Bytes, angegeben als Folge von 4 ganzzahligen, durch Dezimalpunkte getrennte Dezimalzahlen.
- IPv4-Adressen sind in zwei Teile gegliedert: Adresspräfix und Adresssuffix.
- Adresspräfix (Netzwerk-ID) identifiziert das physikalische Netzwerk.
- Adresssuffix (Host-ID) identifiziert Rechner im Netzwerk der Netzwerk-ID.
- Subnetzmaske: 32 Bit lang; kennzeichnet den Netzwerk-ID-Teil durch 1-Bits und den Host-ID-Teil durch 0-Bits. [Wikipedia]
  - Dotted-Decimal-Notation: `141.54.1.11/255.255.0.0`
  - Suffix-Notation: `141.54.1.11/16`

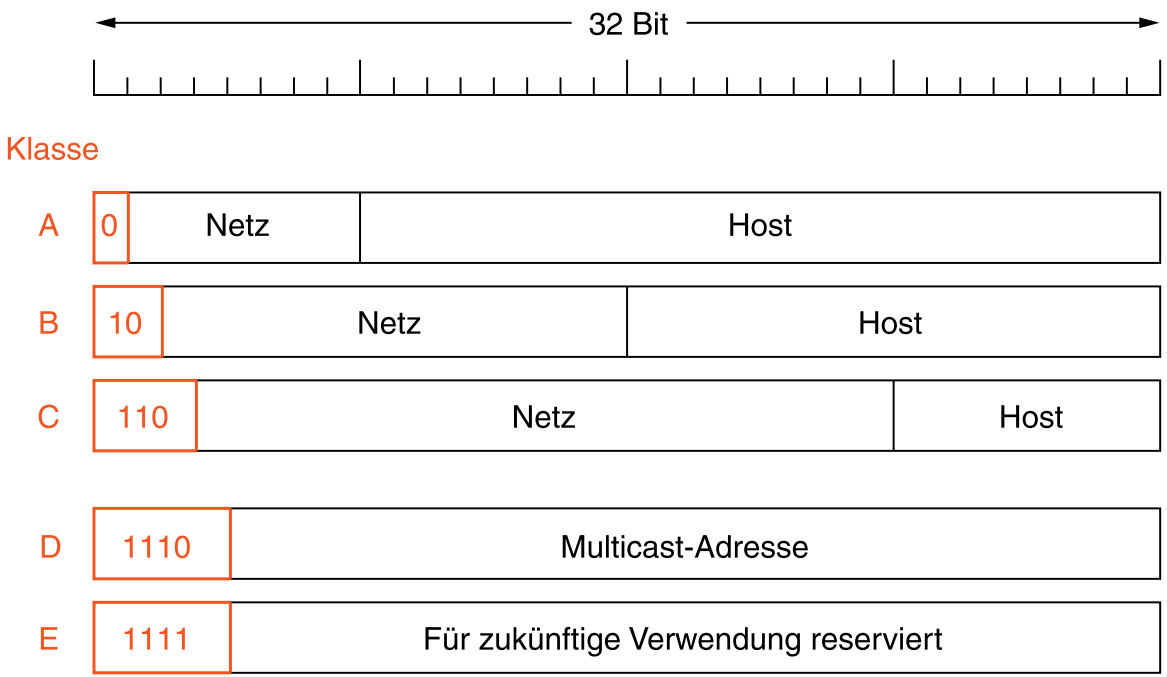
Spezielle IP-Adressen:

- Broadcast-Adresse. Alle Bits der Host-ID sind auf 1 gesetzt.
- Loop-Back-Adresse. 127.0.0.1, sendender Rechner erhält Paket zurück.

Internetworking

IP-Adressierung mit IPv4: Netzklassen (veraltet)

1981-1993. Netzklassen zur Einteilung des IPv4-Adressbereiches. [Wikipedia]



[Tanenbaum]

Bemerkungen:

- Netzklassen waren eine von 1981 bis 1993 verwendete Unterteilung des IPv4-Adressbereiches in Teilnetze für verschiedene Nutzer. Von der Netzklasse konnte die Größe eines Netzes abgeleitet werden. Dies ist beim Routing im Internet wichtig, um zu unterscheiden, ob eine Ziel-IP-Adresse im eigenen oder einem fremden Netz zu finden ist.
- Da Netzklassen sich als zu unflexibel und wenig sparsam im Umgang mit der knappen Ressource IP-Adressen herausgestellt haben, wurden sie 1985 zunächst durch Subnetting und 1992 mit Supernetting ergänzt und 1993 schließlich mit der Einführung des *Classless Inter-Domain Routing* CIDR ersetzt. [Wikipedia]

`2001:0db8:85a3:08d3:1319:8a2e:0370:7344/64`

- IPv6-Adressen bestehen aus aus 128 Bit bzw. 16 Bytes, angegeben als Folge von 8 durch Doppelpunkt getrennte Hexadezimalzahlen.
- IPv6-Adressen sind wie IPv4-Adressen in zwei Teile gegliedert: Adresspräfix und Adresssuffix, auch Interface Identifier genannt.
- IPv6-Netzwerke werden gemäß CIDR notiert, durch Anhängen der Präfixlänge in Bits mit “/” an die Adresse.
- In einer URL werden IPv6-Adressen in eckige Klammern eingeschlossen.

Internetworking

IP-Adressierung mit IPv6

`2001:0db8:85a3:08d3:1319:8a2e:0370:7344/64`

- IPv6-Adressen bestehen aus aus 128 Bit bzw. 16 Bytes, angegeben als Folge von 8 durch Doppelpunkt getrennte Hexadezimalzahlen.
- IPv6-Adressen sind wie IPv4-Adressen in zwei Teile gegliedert: Adresspräfix und Adresssuffix, auch Interface Identifier genannt.
- IPv6-Netzwerke werden gemäß CIDR notiert, durch Anhängen der Präfixlänge in Bits mit "/" an die Adresse.
- In einer URL werden IPv6-Adressen in eckige Klammern eingeschlossen.

- IPv6 ermöglicht 2^{128} Adressen ($3.4 \cdot 10^{38}$ bzw. 340 Sextillionen) gegenüber 2^{32} (3.4 Milliarden) bei IPv4. Zum Vergleich: die Erde hat 10^{51} Atome.
- IPv6 führt Verbesserungen u.a. im Protokollaufbau ein.
- IPv6 ist als RFC 2460 spezifiziert.

Internetworking

Domain Name System, DNS

Auflösung von Hostnamen und Umwandlung in die zugehörigen IP-Adressen.

➜ DNS-Server als verteilte Datenbank

erste Realisierung:

- Alle Namen und Adressen befanden sich in einer zentralen Masterdatei, die per FTP auf jeden Rechner geladen wurde.
- nicht skalierbar, keine lokale Organisation möglich

aktuelle Realisierung:

- hierarchische Organisation durch organisatorische Partitionierung (.com, .edu, .gov, .mil, etc.) als auch geografische Partionierung (.de, .uk, .fr, etc.)
- Der Suffix nach dem letzten Punkt wird als Top-Level-Domain, TLD, bezeichnet. Liste aktueller und neu zugelassener TLDs.

Bemerkungen:

- Die geografischen Endungen sind unabhängig von der physischen Position der Ressourcen. Aber: die Betreiber der Domains unterliegen der Rechtsprechung des bezeichneten Landes.

Internetworking

Quellen zum Nachlernen und Nachschlagen im Web

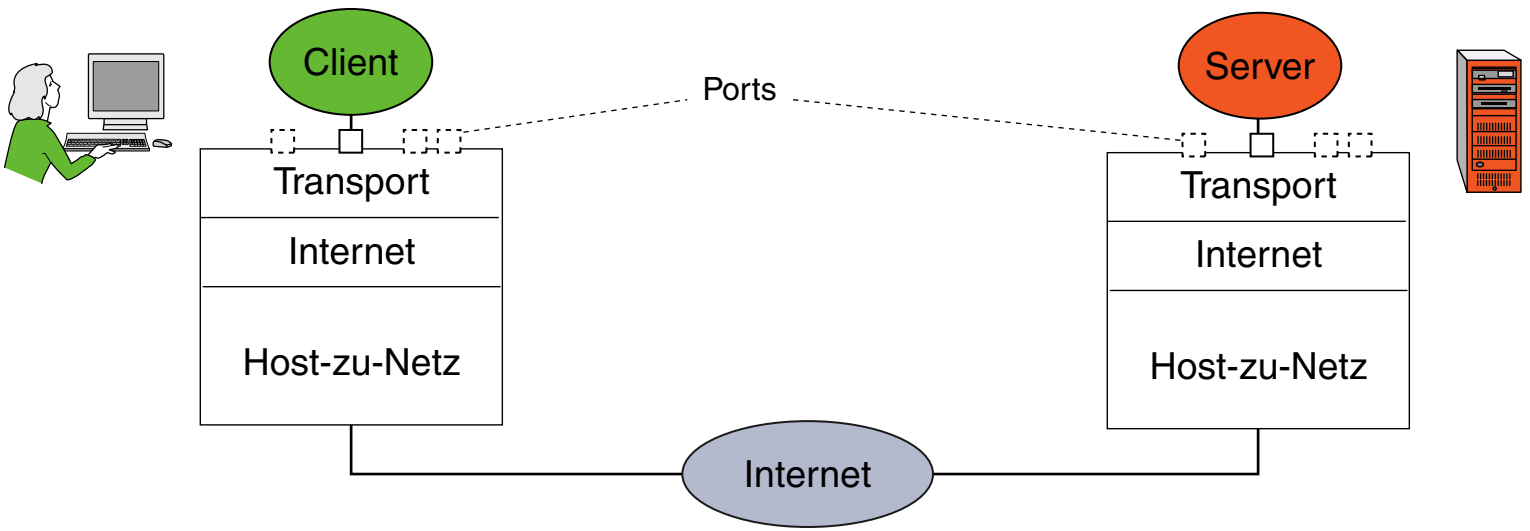
- Wikipedia. *Top-Level-Domain.*
  de.wikipedia.org/wiki/Top-Level-Domain
- Wikipedia. *Domain Name System.*
  de.wikipedia.org/wiki/Domain_Name_System
- Wikipedia. *Root-Nameserver.*
  de.wikipedia.org/wiki/Root-Nameserver
- Wikipedia. *Whois.*
  https://de.wikipedia.org/wiki/Whois

Kapitel WT:II (Fortsetzung)

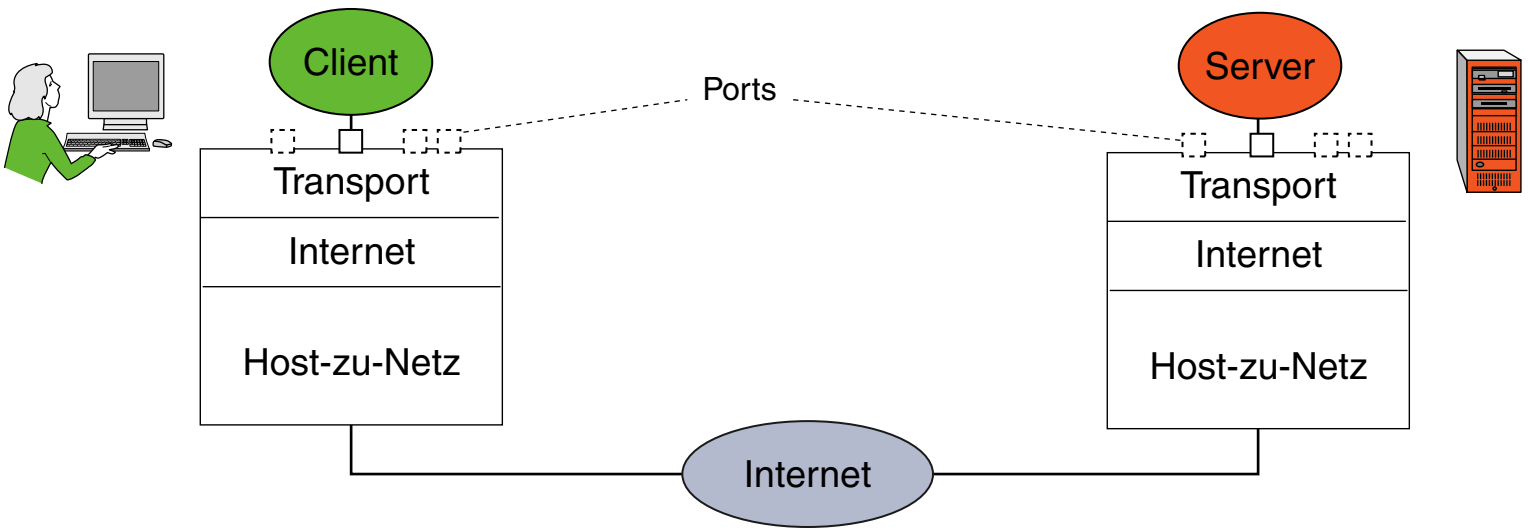
II. Rechnerkommunikation und Protokolle

- Rechnernetze
- Prinzipien des Datenaustauschs
- Netzsoftware und Kommunikationsprotokolle
- Internetworking
- Client-Server-Interaktionsmodell
- Uniform Resource Locator
- Hypertext-Transfer-Protokoll HTTP
- Fortgeschrittene HTTP-Konzepte

Client-Server-Interaktionsmodell



Client-Server-Interaktionsmodell



Rollenverteilung in einem Web-basierten Informationssystem:

- Dienstgeber (*Server*), die einen bestimmten Dienst (*Service*) erbringen.
  Ein Dienst besteht aus einer oder mehreren Funktionen (Operationen, Methoden), die aufgerufen werden können.
  Ein Server ist ein Prozess oder eine Prozessgruppe auf einem Rechner.
- Dienstnehmer (*Clients*), die Serverdienste von anderen Prozessen in Anspruch nehmen.

Bemerkungen:

- Das Client-Server-Interaktionsmodell wird auch als Client-Server-Paradigma bezeichnet.
- Ein Prozess ist häufig sowohl Server (d.h., er bietet einen Dienst an) als auch Client (d.h., er benutzt andere Dienste). Bezeichnung in diesem Zusammenhang auch „Servant“.
- Das Gegenstück zum Client-Server-Paradigma ist das Peer-to-Peer-Paradigma: die Kommunikation unter Gleichgestellten.

Client-Server-Interaktionsmodell

Portkonzept

- Dienste (auf Anwendungsebene) werden über eine Endpunkt-zu-Endpunkt-Verbindung auf Basis der Transportschicht abgewickelt.
- Ein Port ist ein Dienstzugriffspunkt (*Service Access Point*) der Transportschicht des TCP/IP-Protokolls. Ports sind als 16Bit-Zahl codiert.
- Zusammen definieren die IP-Adresse (Internetschicht, Schicht 3) und die Portnummer (Transportschicht, Schicht 4) einen Kommunikationskanal.

Endpunkt-zu-Endpunkt: $\underbrace{\text{IP-Adresse + Portnr.}}_{\text{Source}} \longleftrightarrow \underbrace{\text{IP-Adresse + Portnr.}}_{\text{Destination}}$

Client-Server-Interaktionsmodell

Portkonzept

- Dienste (auf Anwendungsebene) werden über eine Endpunkt-zu-Endpunkt-Verbindung auf Basis der Transportschicht abgewickelt.
- Ein Port ist ein Dienstzugriffspunkt (*Service Access Point*) der Transportschicht des TCP/IP-Protokolls. Ports sind als 16Bit-Zahl codiert.
- Zusammen definieren die IP-Adresse (Internetschicht, Schicht 3) und die Portnummer (Transportschicht, Schicht 4) einen Kommunikationskanal.

Endpunkt-zu-Endpunkt: $\underbrace{\text{IP-Adresse + Portnr.}}_{\text{Source}} \longleftrightarrow \underbrace{\text{IP-Adresse + Portnr.}}_{\text{Destination}}$

Unterscheidung nach Verbindungsart bzw. Zuverlässigkeit:

- TCP-Port.
  Einrichtung von verbindungsorientiertem, zuverlässigem Transportdienst.
- UDP-Port.
  Einrichtung von verbindungslosem, unzuverlässigem Transportdienst.

Client-Server-Interaktionsmodell

Klassifikation von Ports

1. Standardisierte Ports.
   Weltweit eindeutig für Standarddienste reserviert: 0 - 255 für TCP/IP-Anwendungen, 256 – 1.023 für besondere Unix-Anwendungen.
2. Registrierte Ports.
   1.024 – 49.151, werden von der IANA verwaltet.
3. Private, dynamische Ports.
   Restliche Nummern von 49.152 – 65.535.

Client-Server-Interaktionsmodell

Klassifikation von Ports

1. Standardisierte Ports.
   Weltweit eindeutig für Standarddienste reserviert: 0 - 255 für TCP/IP-Anwendungen, 256 – 1.023 für besondere Unix-Anwendungen.
2. Registrierte Ports.
   1.024 – 49.151, werden von der IANA verwaltet.
3. Private, dynamische Ports.
   Restliche Nummern von 49.152 – 65.535.

Portnummern nach RFC 1700 [related] :

Port	Beschreibung
20	File Transfer (Daten)
21	File Transfer (Steuerung)
23	Telnet, Network Virtual Terminal
25	Simple Mail Transfer (Mail-Weiterleitung)
53	Domain Name Service
80	World Wide Web, Hyper Text Transfer

Bemerkungen:

- Die Ports von 0 bis 1023 werden auch als „well-known Ports“ bezeichnet. Sie sind in der RFC 1700 spezifiziert (Seite 15ff.).
- Auf Unix-Systemen sind in der Datei `/etc/services` die Ports mit ihren Service-Zuordnungen beschrieben.

Client-Server-Interaktionsmodell

Socket-API

- Die (Programmier-)Schnittstelle einer Anwendung zur Transportschicht wird als API (*Application Program Interface*) bezeichnet.
- Protokollstandards wie z.B. TCP/IP definieren keine API, sondern abstrakte Dienste bzw. Operationen.
- Der de-Fakto-Standard der API zur Transportschicht ist die Socket-API – kurz: Sockets. Sie stellt einen Dienst von Schicht 4 für Schicht 5 bereit.
- Ursprung: Teil von BSD-Unix, University of California at Berkeley.

Client-Server-Interaktionsmodell

Socket-API

- Die (Programmier-)Schnittstelle einer Anwendung zur Transportschicht wird als API (*Application Program Interface*) bezeichnet.
- Protokollstandards wie z.B. TCP/IP definieren keine API, sondern abstrakte Dienste bzw. Operationen.
- Der de-Fakto-Standard der API zur Transportschicht ist die Socket-API – kurz: Sockets. Sie stellt einen Dienst von Schicht 4 für Schicht 5 bereit.
- Ursprung: Teil von BSD-Unix, University of California at Berkeley.

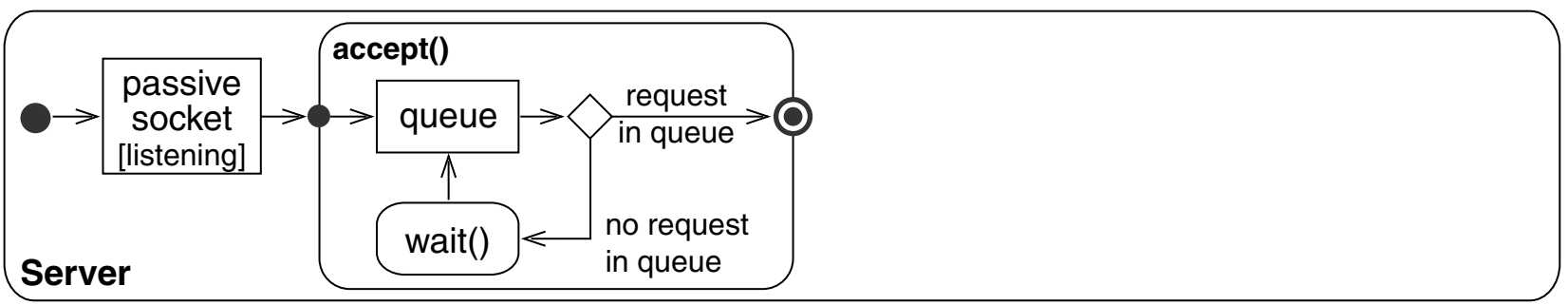
Socket-Datenstruktur:

- realisiert bei Client und Server einen Kommunikationsendpunkt
- ist vom Typ Stream, `SOCK_STREAM`, oder Datagramm, `SOCK_DGRAM`
- macht die Abwicklung des TCP/IP-Protokolls (IP-Adressen, Ports, Pufferung) transparent
- stellt Funktionen zum Schreiben, Lesen, Lauschen, etc. zur Verfügung

Client-Server-Interaktionsmodell

Verbindungsherstellung mit Sockets

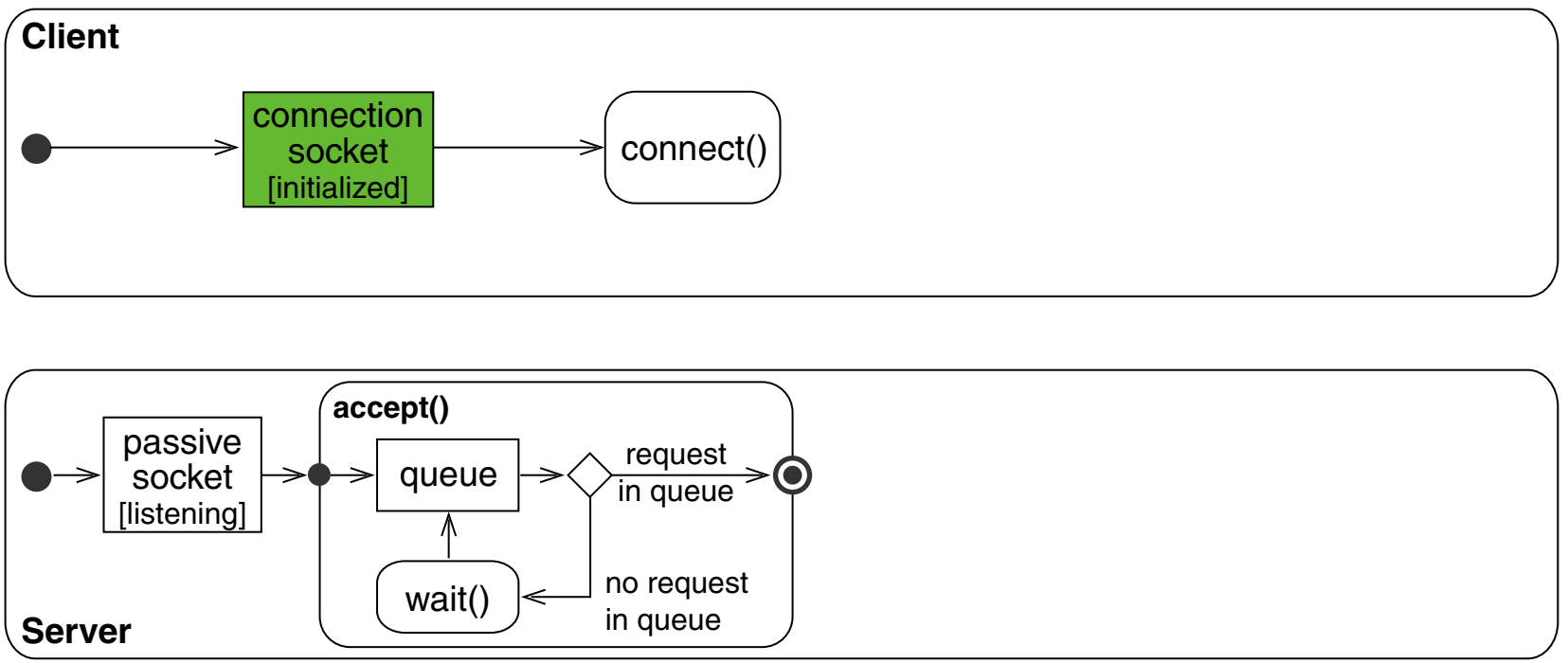
- Server fordert vom Betriebssystem eine Server-Socket-Datenstruktur an (eigene IP-Adresse, Service-Port).



Client-Server-Interaktionsmodell

Verbindungsherstellung mit Sockets

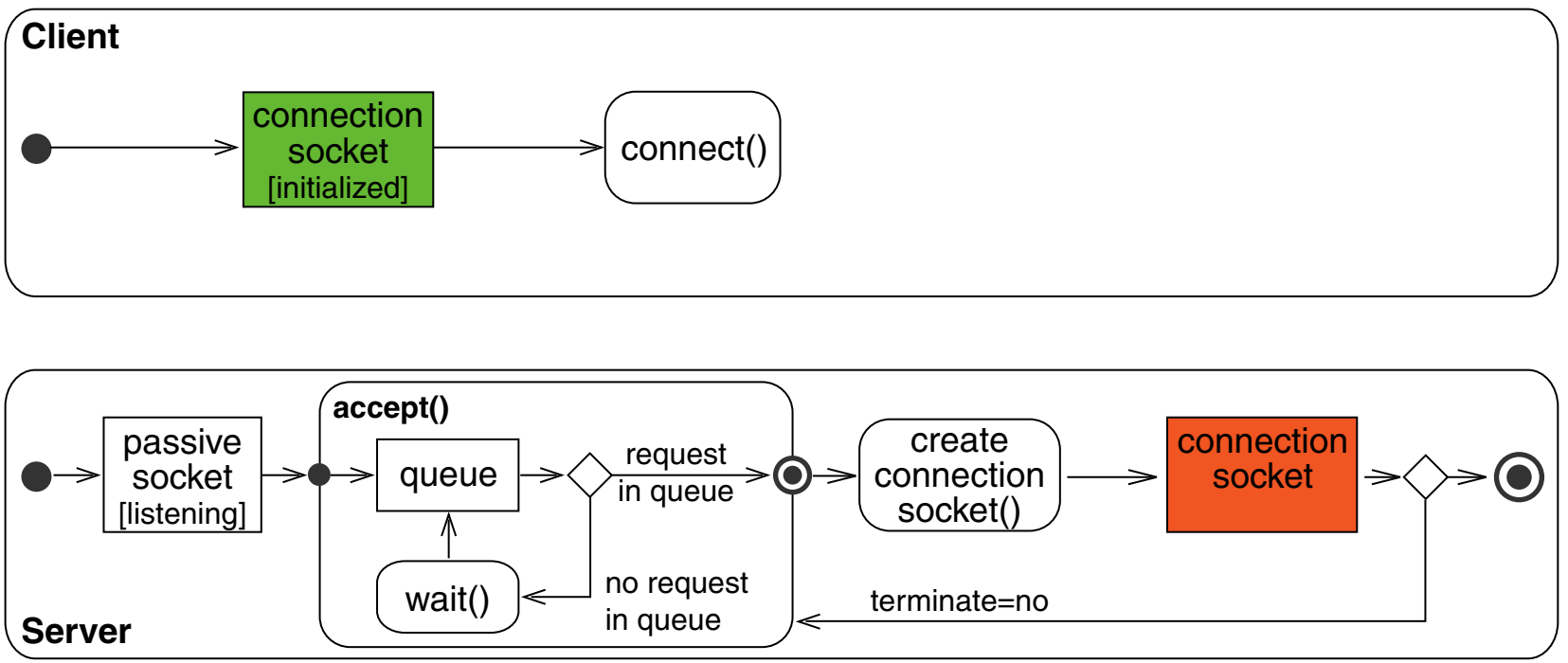
- Server fordert vom Betriebssystem eine Server-Socket-Datenstruktur an (eigene IP-Adresse, Service-Port).
- Client fordert vom Betriebssystem eine Client-Socket-Datenstruktur an (Server-IP-Adresse, Server-Service-Port); dabei wird auch ein lokaler Port beim Client reserviert.



Client-Server-Interaktionsmodell

Verbindungsherstellung mit Sockets

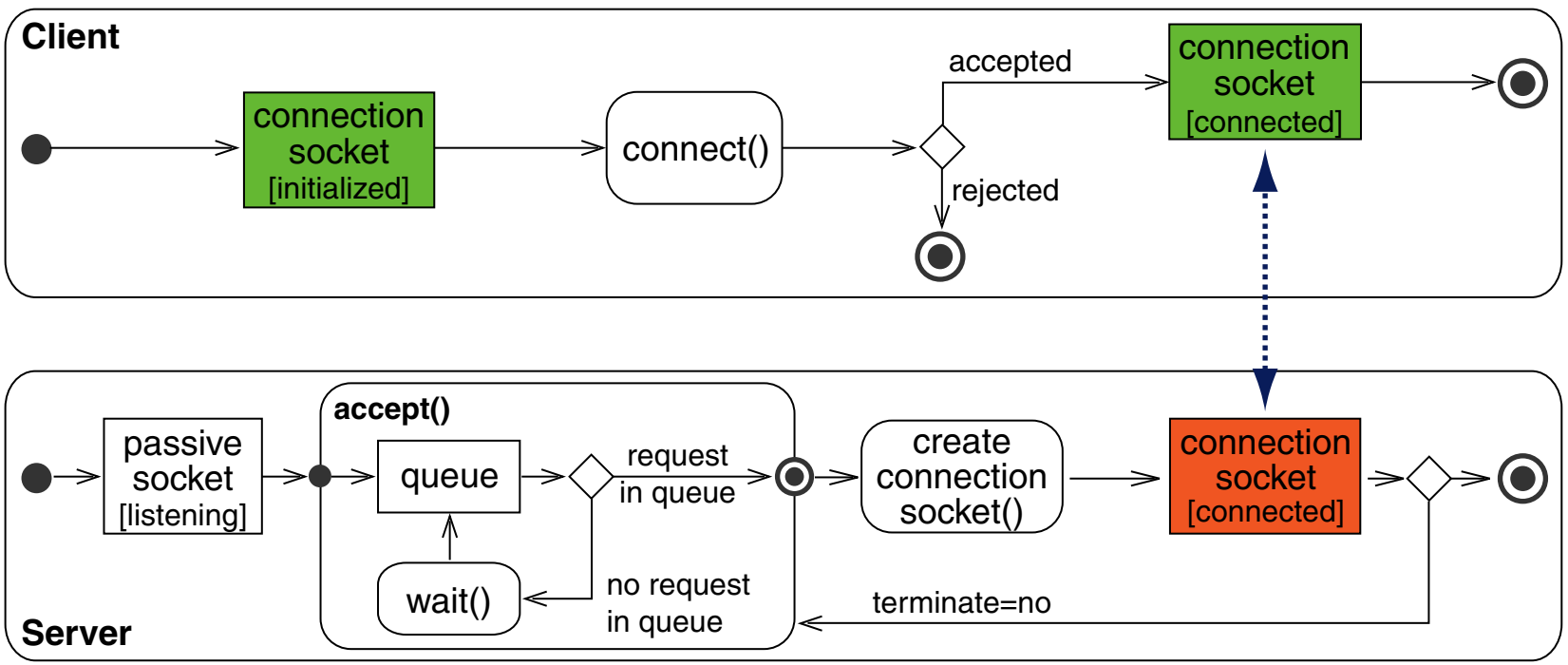
- Server fordert vom Betriebssystem eine Server-Socket-Datenstruktur an (eigene IP-Adresse, Service-Port).
- Client fordert vom Betriebssystem eine Client-Socket-Datenstruktur an (Server-IP-Adresse, Server-Service-Port); dabei wird auch ein lokaler Port beim Client reserviert.



Client-Server-Interaktionsmodell

Verbindungsherstellung mit Sockets

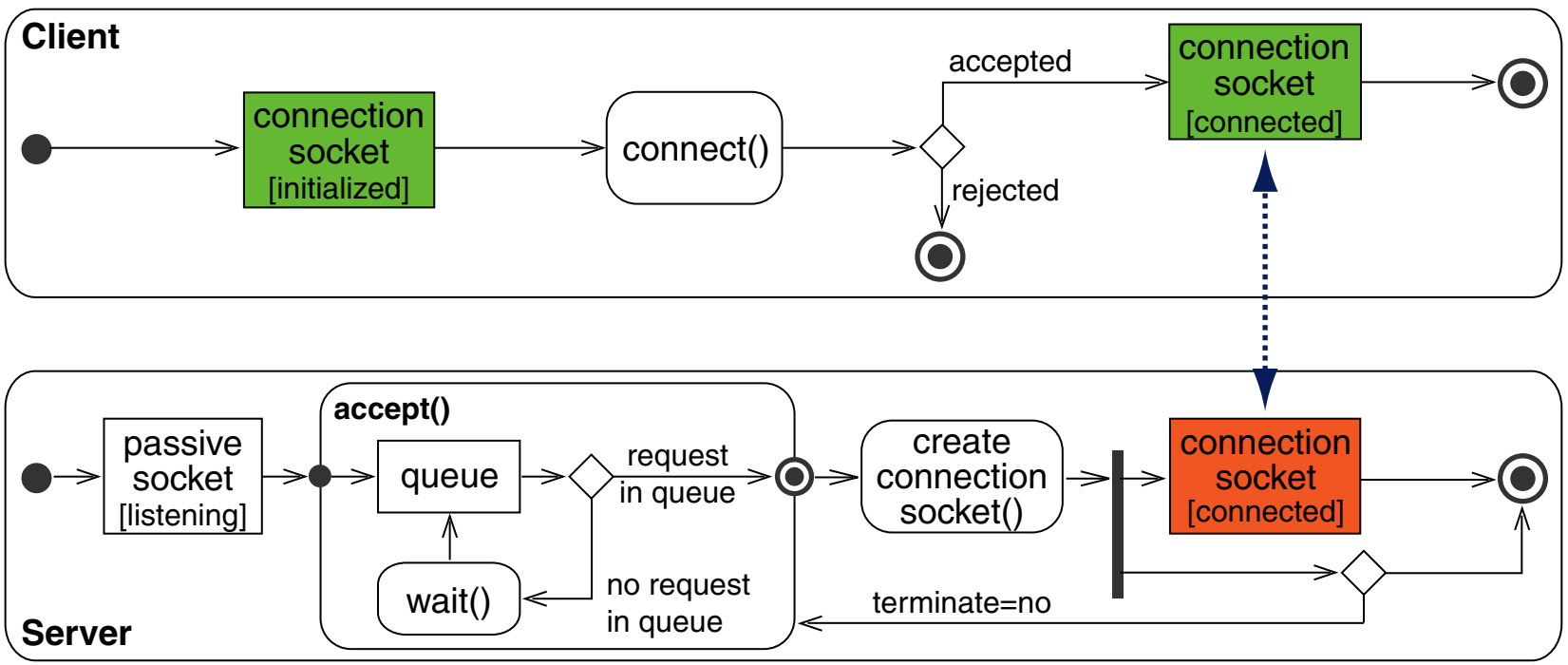
- Server fordert vom Betriebssystem eine Server-Socket-Datenstruktur an (eigene IP-Adresse, Service-Port).
- Client fordert vom Betriebssystem eine Client-Socket-Datenstruktur an (Server-IP-Adresse, Server-Service-Port); dabei wird auch ein lokaler Port beim Client reserviert.



Client-Server-Interaktionsmodell

Verbindungsherstellung mit Sockets

- Server fordert vom Betriebssystem eine Server-Socket-Datenstruktur an (eigene IP-Adresse, Service-Port).
- Client fordert vom Betriebssystem eine Client-Socket-Datenstruktur an (Server-IP-Adresse, Server-Service-Port); dabei wird auch ein lokaler Port beim Client reserviert.

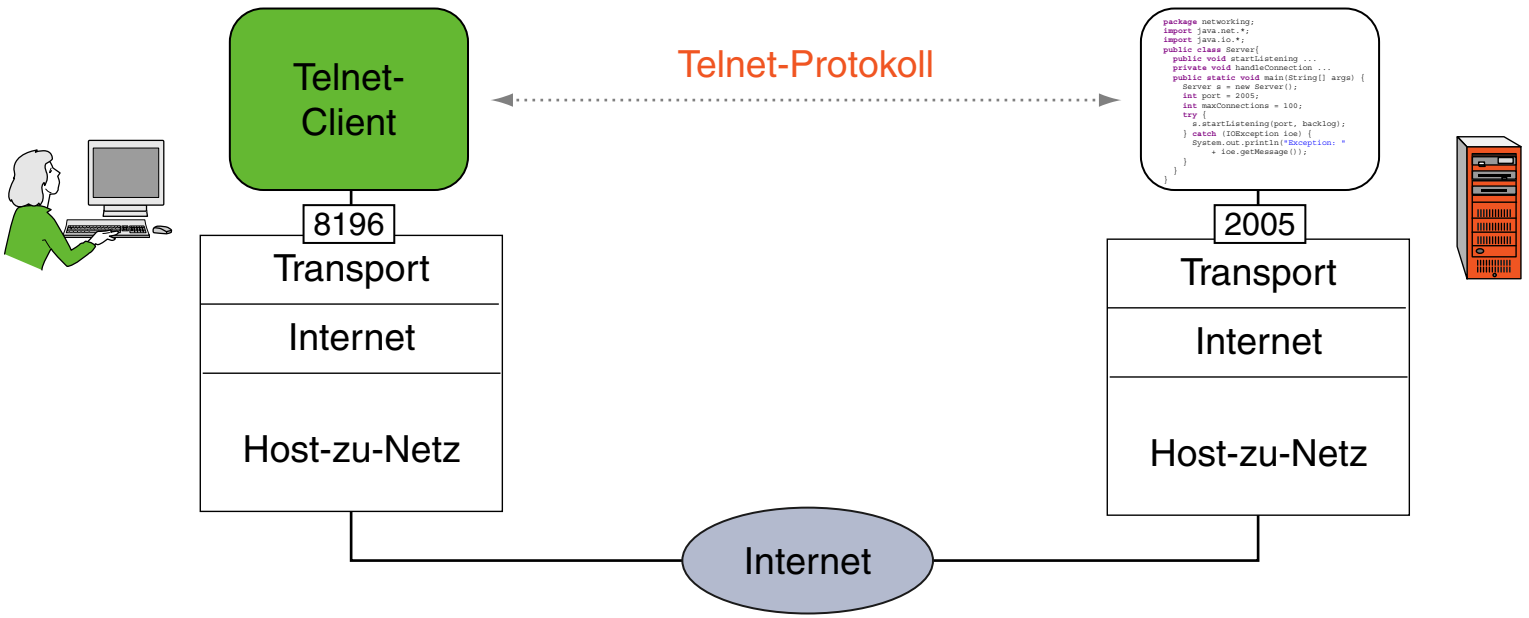


Bemerkungen:

- Schritte bei der Verbindungsherstellung:
  1. a) Server: Passiven Socket instanziieren (listening, asynchron).
     b) Server: Wartet auf Anfrage (accept(), synchron).
  2. a) Client: Instanziiert neuen Verbindungs-Socket (initialized, synchron).
     b) Client: Setzt Verbindungswunsch ab (connect(), synchron).
  3. Server: Der accept()-Aufruf liefert einen neuen Verbindungs-Socket bzgl. des Clients.
  4. Server, Client: Die beiden Verbindungs-Sockets bilden einen gemeinsamen Kommunikationskanal. Im Falle eines Stream-Sockets ist das ein bidirektionaler Stream.
- Der passive Socket ist in Java durch die Klasse `ServerSocket` implementiert. [Javadoc]
- Der Verbindungs-Socket ist in Java durch die Klasse `Socket` implementiert. [Javadoc]

Client-Server-Interaktionsmodell

Socket-Verbindung mit Java [Client-side]



Auf der Client-Seite bildet ein Telnet-Client das Gegenstück einer Verbindung zu dem Beispiel-Server. Der Telnet-Client lässt sich auf diese Art einsetzen, weil das Telnet-Protokoll lediglich Tastatureingaben vom Client zum Server bzw. Textausgaben vom Server zum Client überträgt. [RFC 854]

Client-Server-Interaktionsmodell

Socket-Verbindung mit Java

```
public void startListening(int port, int backlog) throws IOException {
     // Start a passive socket, step (1a).
     ServerSocket passiveSocket = new ServerSocket(port, backlog);
     // Now the passive socket is listening.
     System.out.println("Server started on port " + port + ".");
     boolean listen = true;
     while(listen) {
          // Blocking wait for a connection.
          // If a client connects, a connection socket is returned
          // by the passive socket.
          Socket connectionSocket = passiveSocket.accept(); // step (1b+3)
          handleConnection(connectionSocket); // step (4)
          // Eventually, set boolean variable terminate.
     }
}
```

Client-Server-Interaktionsmodell

Socket-Verbindung mit Java (Fortsetzung)

```
private void handleConnection(Socket connectionSocket) throws IOException {
     // Send a string to the client, step (4).
     OutputStream os = connectionSocket.getOutputStream();
     PrintWriter pw = new PrintWriter(os);
     pw.println("Tell me your name, please.");
     pw.flush(); // "Flushing" empties the send buffer.

     // Receive a string from the client, step (4).
     InputStream is = connectionSocket.getInputStream();
     BufferedReader br = new BufferedReader(new InputStreamReader(is));
     String name = br.readLine();
     if (name != null) {
          pw.println("Welcome " + name + "!");
          pw.flush();
     }

     // Close the connections.
     br.close();
     pw.close();
}
```

Client-Server-Interaktionsmodell

Socket-Verbindung mit Java (Fortsetzung)

```
package networkprotocol;

import java.net.*;
import java.io.*;

public class Server {

     public void startListening(int port, int backlog) ...
     private void handleConnection(Socket connectionSocket) ...

     public static void main(String[] args) {
          Server s = new Server();
          int port = 2005;
          int backlog = 100;
          try {
               s.startListening(port, backlog);
          } catch (IOException ioe) {
               System.out.println("There was an exception, the message is: "
                         + ioe.getMessage());
          }
     }
}
```

Client-Server-Interaktionsmodell

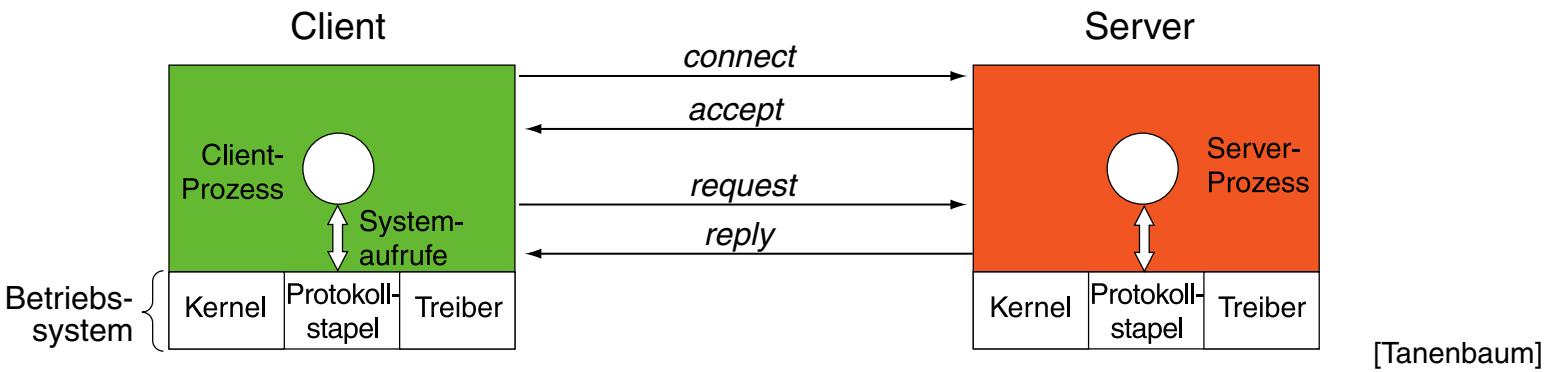
Socket-Verbindung mit Java (Fortsetzung)

```
[stein@webis stein]$
[stein@webis stein]$ telnet localhost 2005
Trying 127.0.0.1...
Connected to localhost (127.0.0.1).
Escape character is '^]'.
Tell me your name, please.
Benno
Welcome Benno!

Connection closed by foreign host.

[stein@webis stein]$
```

Client-Server-Interaktionsmodell

Dienstabwicklung durch Anforderungs-/Antwortprotokoll

Zur Abwicklung eines Dienstes ist neben dem Kommunikationskanal noch ein Protokoll notwendig. Vorherrschend im Web: Anforderungs-/Antwortprotokoll.



[Tanenbaum]

- Client initiiert Kommunikation, sendet Auftrag an Server.
- Server nimmt Auftrag entgegen, bearbeitet ihn und schickt das Ergebnis an den Client zurück.
- Stichwort „synchrone Kommunikation“: Client blockiert, bis Antwort eintrifft.
- Server wartet auf nächsten Auftrag.

Client-Server-Interaktionsmodell

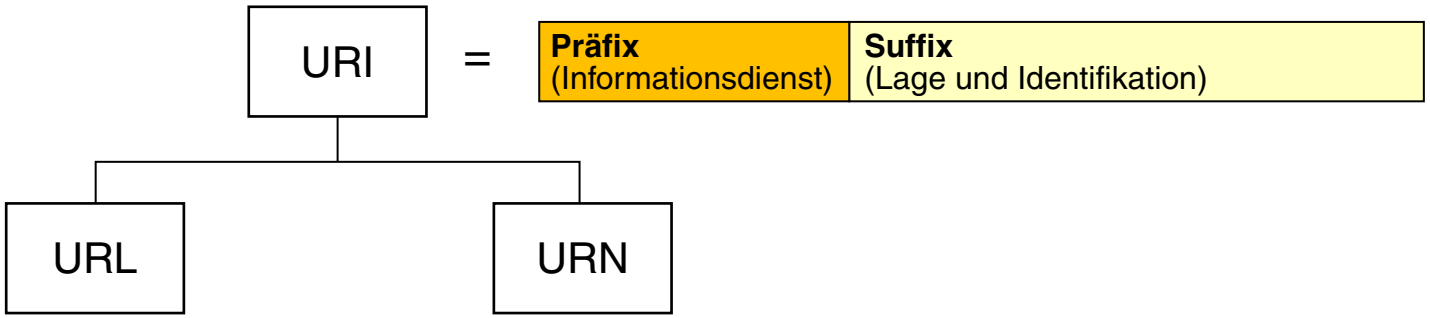
Dienstabwicklung durch Anforderungs-/Antwortprotokoll

Ein Protokoll spezifiziert die zugelassen Anfragen (*Message Types*), die Art der Parameter-Codierung, Verhalten im Fehlerfall, Time-Outs, etc.

Bekannte Anforderungs-/Antwortprotokolle [related] :

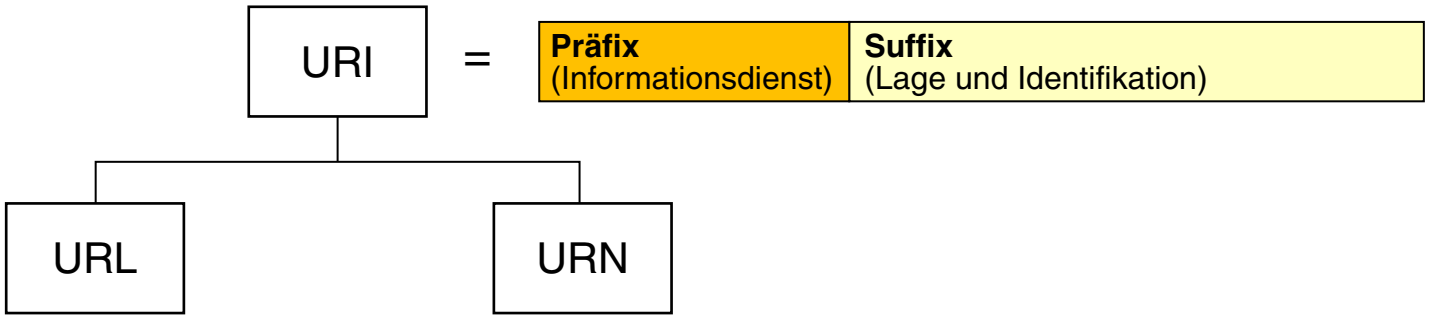
Dienst	**Port**	**Beschreibung**
ftp-data	20	File Transfer (Daten)
ftp	21	File Transfer (Steuerung)
ssh	22	Secure Shell
smtp	25	Simple Mail Transfer (Mail-Vermittlung)
dns	53	Domain Name Service
www-http	80	World Wide Web, Hyper Text Transfer
pop3	110	Post Office Protocol v3 (Mail ausliefern und abholen)
nntp	119	Network News Transfer
imap	110	Interactive Mail Access (Mail ausliefern und abholen)

Uniform Resource Locator



- URI [RFC 3986]
- URL [RFC 1738]
- URN [RFC 2141]

Uniform Resource Locator



- URI [RFC 3986]
  Identifiziert eindeutig eine Informationsressource im WWW, unabhängig davon, ob es sich um ein Hypermedia-Dokument handelt.
- URL [RFC 1738]
  Identifiziert über eine eindeutige Adresse den Ort (*Location*) einer Informationsressource im WWW.
- URN [RFC 2141]
  Identifiziert über einen eindeutigen Namen eine Informationsressource im WWW. Auf Basis der URN soll (in Zukunft) der Zugriff auf die Ressource sowie die Abfrage ihrer Eigenschaften möglich sein.

Uniform Resource Locator

URL versus URI [RFC 3986]

The term “Uniform Resource Locator” (URL) refers to the subset of URIs that, in addition to identifying a resource, provide a means of locating the resource by describing its primary access mechanism (e.g., its network “location”).

. . .

A common misunderstanding of URIs is that they are only used to refer to accessible resources. The URI itself only provides identification; access to the resource is neither guaranteed nor implied by the presence of a URI. Instead, any operation associated with a URI reference is defined by the protocol element, data format attribute, or natural language text in which it appears.

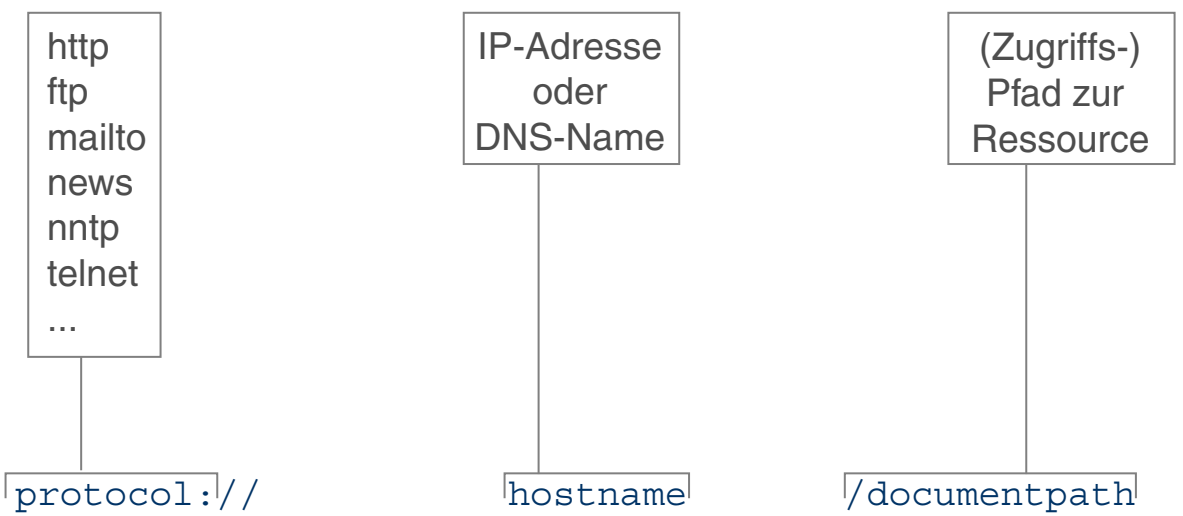
Uniform Resource Locator

URL versus URI [RFC 3986]

The term "Uniform Resource Locator" (URL) refers to the subset of URIs that, in addition to identifying a resource, provide a means of locating the resource by describing its primary access mechanism (e.g., its network "location").

. . .

A common misunderstanding of URIs is that they are only used to refer to accessible resources. The URI itself only provides identification; access to the resource is neither guaranteed nor implied by the presence of a URI. Instead, any operation associated with a URI reference is defined by the protocol element, data format attribute, or natural language text in which it appears.



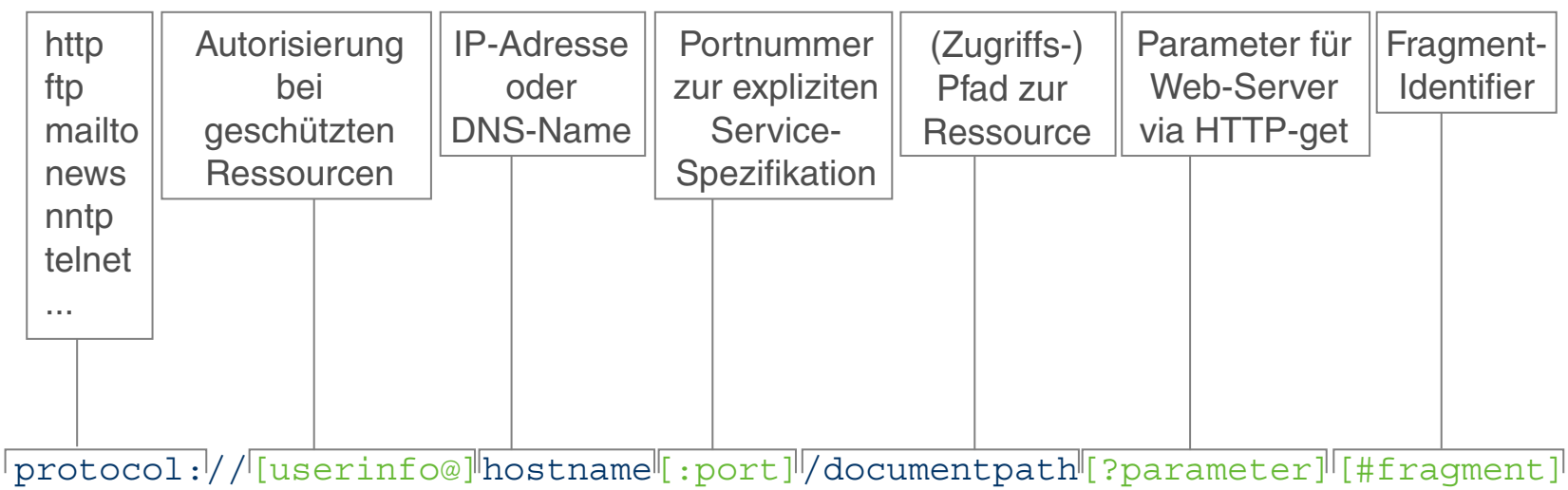
Uniform Resource Locator

URL versus URI [RFC 3986]

The term “Uniform Resource Locator” (URL) refers to the subset of URIs that, in addition to identifying a resource, provide a means of locating the resource by describing its primary access mechanism (e.g., its network “location”).

. . .

A common misunderstanding of URIs is that they are only used to refer to accessible resources. The URI itself only provides identification; access to the resource is neither guaranteed nor implied by the presence of a URI. Instead, any operation associated with a URI reference is defined by the protocol element, data format attribute, or natural language text in which it appears.



Uniform Resource Locator

BNF-Syntax der HTTP-URL

```
URI ::=             schema ':' hier-part [ '?' query ] [ '#' fragment ]

schema ::=          'http' | ...
hier-part ::=       '//' authority path-rooted
authority ::=       [ userinfo '@' ] host [ ':' port ]
host ::=            hostname | IPv4address | IPv6address

hostname ::=        { domainlabel '.' }* toplabel
domainlabel ::=     alphadigit { { alphadigit | '-' }* alphadigit }?
toplabel ::=        alpha { { alphadigit | '-' }* alphadigit }?

path-rooted ::=     { '/' segment }*
path ::=            { segment-r }? { '/' segment }*
segment ::=         { uchar | ':' | '@' }*
segment-r ::=       { uchar | '@' }+

query ::=           { uchar | ':' | '@' | '/' | '?' }*
fragment ::=        { uchar | ':' | '@' | '/' | '?' }*

userinfo ::=        { uchar | ':' }*
IPv4address ::=     digits . digits . digits . digits
port ::=            digits
```

Bemerkungen:

- Die angegebene BNF basiert auf der BNF für URIs [RFC 3986:A] und für URLs [RFC 1738:5]. In der vorliegenden Vereinfachung erlaubt die BNF die Bildung typischer URLs, nicht nur im HTTP-Kontext.
- `alpha`, `alphadigit`, `alphadigit` und `uchar` sind Zeichenmengen, die Buchstaben, Ziffern und ggf. zusätzliche Sonderzeichen (ohne Sonderbedeutung) enthalten und unterschiedliche Repräsentationen erlauben.
- URI Planning Interest Group, W3C/IETF.
  URIs, URLs, and URNs: Clarifications and Recommendations 1.0. [W3C]

Uniform Resource Locator

Namensauflösung URN → URL

- URN-Syntax [RFC 2141] :

  ```
  URN ::= 'urn:' namespace-id ':' namespace-specific-string
  ```

- Beispiel: `urn:ISBN:0-262-01210-3`

Uniform Resource Locator

Namensauflösung URN → URL

- URN-Syntax [RFC 2141] :

  ```
  URN ::= 'urn:' namespace-id ':' namespace-specific-string
  ```

- Beispiel: `urn:ISBN:0-262-01210-3`

Uniform Resource Locator

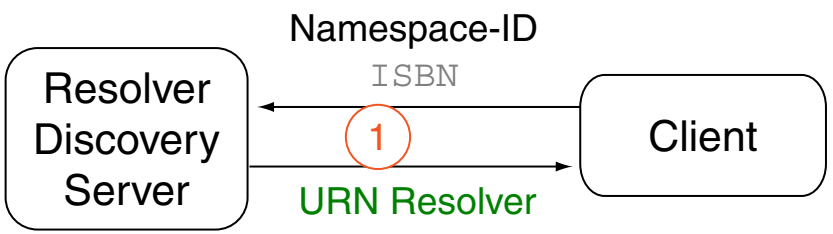
Namensauflösung URN → URL

- URN-Syntax [RFC 2141] :

  ```
  URN ::= 'urn:' namespace-id ':' namespace-specific-string
  ```

- Beispiel: `urn:ISBN:0-262-01210-3`

Namensauflösung URN → URL mittels Resolver Discovery Service [RFC 2276] :



Uniform Resource Locator

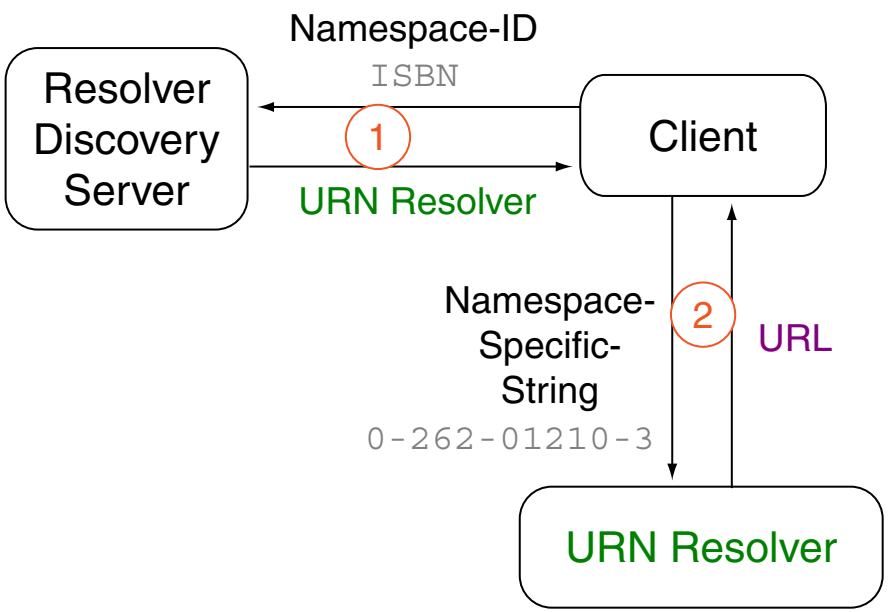
Namensauflösung URN → URL

- URN-Syntax [RFC 2141] :

  ```
  URN ::= 'urn:' namespace-id ':' namespace-specific-string
  ```

- Beispiel: `urn:ISBN:0-262-01210-3`

Namensauflösung URN → URL mittels Resolver Discovery Service [RFC 2276] :



Uniform Resource Locator

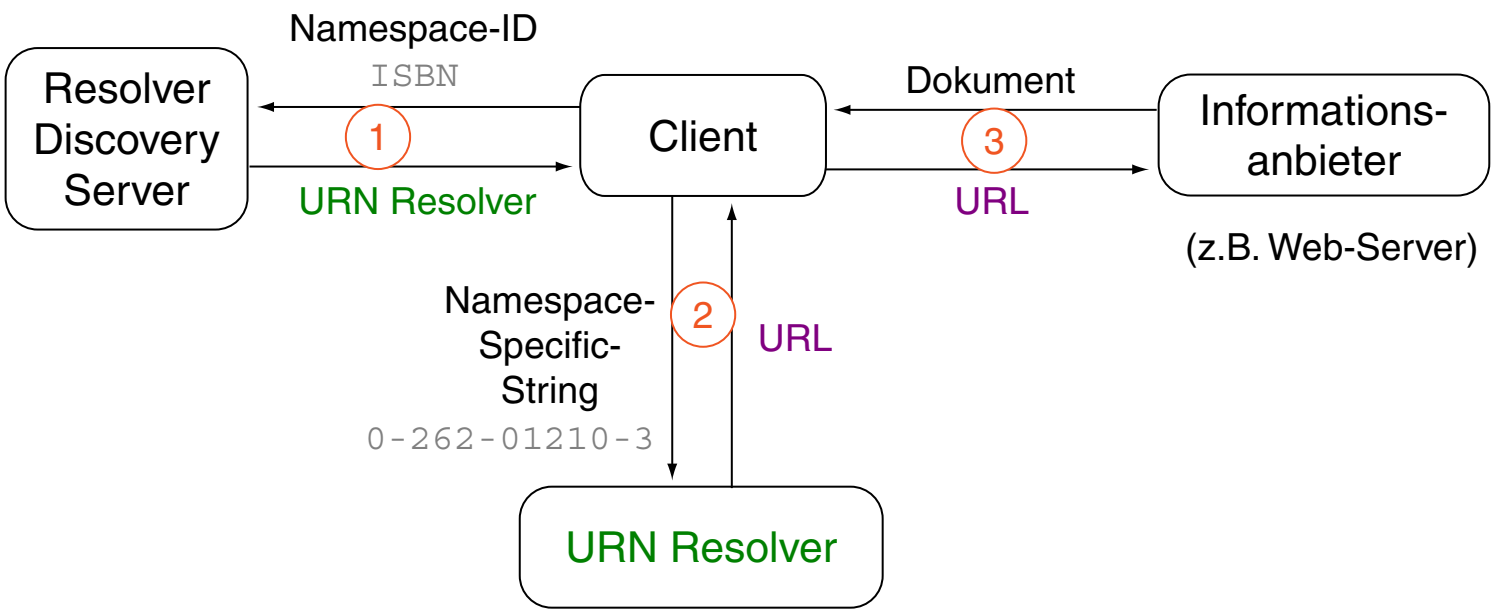
Namensauflösung URN → URL

- URN-Syntax [RFC 2141] :

  ```
  URN ::= 'urn:' namespace-id ':' namespace-specific-string
  ```

- Beispiel: `urn:ISBN:0-262-01210-3`

Namensauflösung URN → URL mittels Resolver Discovery Service [RFC 2276] :



Bemerkungen:

- Ein Ort (URL) einer Ressource kann sich ändern, ihr Name (URN) nicht. Ziel ist es, Änderungen des Ortes einer Ressource automatisch nachzuvollziehen. Hierfür ist ein Dienst erforderlich, der in der Lage ist, aus dem Namen einer Ressource deren Standort zu ermitteln.

- Zurzeit werden im WWW fast ausschließlich URLs zur Identifikation von Ressourcen verwendet; hinsichtlich der Standardisierung eines Dienstes zur Abbildung URN $\rightarrow$ URL konnte sich noch nicht geeinigt werden: Die Category von RFC 2276 (URN Resolution) ist „Informational“, die von RFC 2141 (URN Syntax) ist „Standards Track“.

- Namensauflösung URN $\rightarrow$ URL bei *Digital Object Identifiern*, DOI, [www.doi.org] :
  1. Für die Namespace-ID „DOI“ wird der URN-Resolver dx.doi.org herausgesucht.
  2. Der URN-Resolver ermittelt für den Namespace-Specific-String `10.1007/s10579-010-9115-y` die URL `https://link.springer.com/article/10.1007%2Fs10579-010-9115-y`.
  3. Die URL verweist auf das Dokument „Stein/Lipka/Prettenhofer: Intrinsic Plagiarism Analysis“, das durch den in der URL spezifizierten Web-Server ausgeliefert wird.

  Die Schritte 2 und 3 werden auf der Web-Seite webis.de > publications durch Klicken des [doi]-Links transparent ausgeführt.

Kapitel WT:II (Fortsetzung)

II. Rechnerkommunikation und Protokolle

- Rechnernetze
- Prinzipien des Datenaustauschs
- Netzsoftware und Kommunikationsprotokolle
- Internetworking
- Client-Server-Interaktionsmodell
- Uniform Resource Locator
- Hypertext-Transfer-Protokoll HTTP
- Fortgeschrittene HTTP-Konzepte

Hypertext-Transfer-Protokoll HTTP

Der HTTP-Standard sieht das Client-Server-Prinzip mit folgenden Funktionseinheiten vor [RFC 2616] :

1. WWW-Client bzw. User-Agent
   Initiiert Verbindungen zu WWW-Servern; der WWW-Client ist in der Regel ein Web-Browser.

2. WWW-Server
   Wartet auf Verbindungswünsche von WWW-Clients und antwortet auf die gestellten Anfragen; liefert gewünschte Ressource oder Statusinformation.

3. Proxy-Server
   System zwischen WWW-Client und WWW-Server; arbeitet sowohl als WWW-Server (hat aufgrund früherer Kommunikation Antworten im Cache) als auch als WWW-Client gegenüber dem sogenannten *Origin-Server*.

4. Gateway
   Vergleichbar dem Proxy-Server mit dem Unterschied, dass der WWW-Client keine Kenntnis über die Existenz des Gateways besitzt.

Hypertext-Transfer-Protokoll HTTP

Der HTTP-Standard sieht das Client-Server-Prinzip mit folgenden Funktionseinheiten vor [RFC 2616] :

1. WWW-Client bzw. User-Agent
   Initiiert Verbindungen zu WWW-Servern; der WWW-Client ist in der Regel ein Web-Browser.

2. WWW-Server
   Wartet auf Verbindungswünsche von WWW-Clients und antwortet auf die gestellten Anfragen; liefert gewünschte Ressource oder Statusinformation.

3. Proxy-Server
   System zwischen WWW-Client und WWW-Server; arbeitet sowohl als WWW-Server (hat aufgrund früherer Kommunikation Antworten im Cache) als auch als WWW-Client gegenüber dem sogenannten *Origin-Server*.

4. Gateway
   Vergleichbar dem Proxy-Server mit dem Unterschied, dass der WWW-Client keine Kenntnis über die Existenz des Gateways besitzt.

Hypertext-Transfer-Protokoll HTTP

Historie

1992 HTTP/0.9

1996 HTTP/1.0 [RFC 1945]

1997 HTTP/1.1 [RFC 2616]

2015 HTTP/2 [RFC 7540, HTTP/2 Home, W3C]

Das HTTP-Protokoll spezifiziert Nachrichtentypen, Datentransfer, Darstellungsregeln (Zeichensatz, Datenformate), Inhaltsabstimmung, Authentisierung, etc. zwischen den genannten Funktionseinheiten.

Kommunikationsablauf aus Client-Sicht:

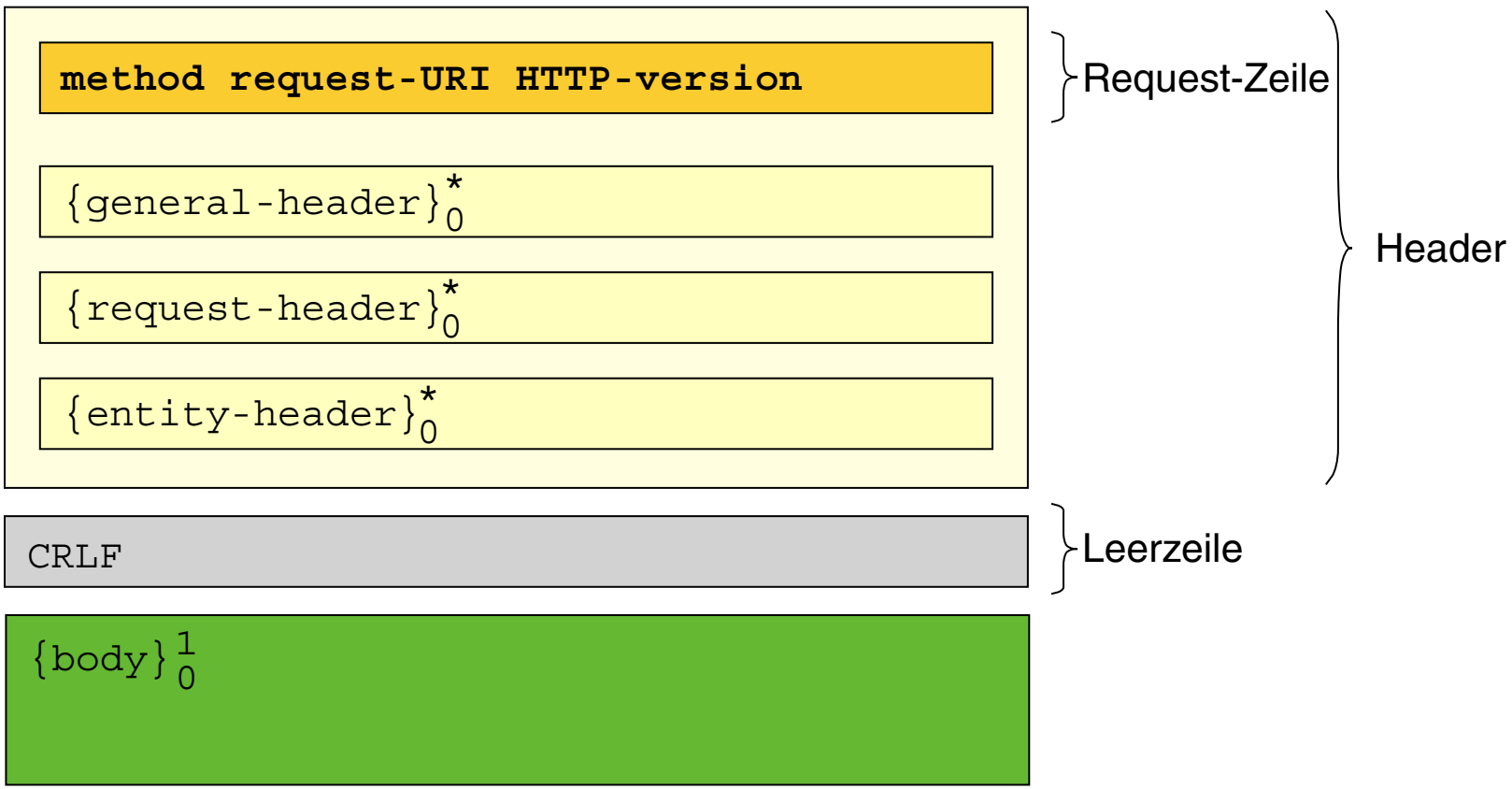
1. Öffnen einer TCP/IP Verbindung zum WWW-Server
2. Request. Senden der Anforderung an WWW-Server
3. Response. Empfangen der Antwort vom WWW-Server
4. Schließen der Verbindung

Bemerkungen:

- HTTP/0.9 versteht nur die GET-Methode, keine Statusinformation und auch keine Information über Medientypen.
- HTTP/1.1 ermöglicht unter anderem Pipelining: mehrere HTTP-Anfragen werden über *eine* TCP/IP-Verbindung abgewickelt.
- HTTP/2 ist deutlich verbessert hinsichtlich Performanz, abwärtskompatibel zu HTTP/1.1 und unterstützt das Zusammenfassen von mehreren HTTP-Anfragen via Multiplexing. Die Entwicklung wurde von Google (SPDY) und Microsoft forciert.

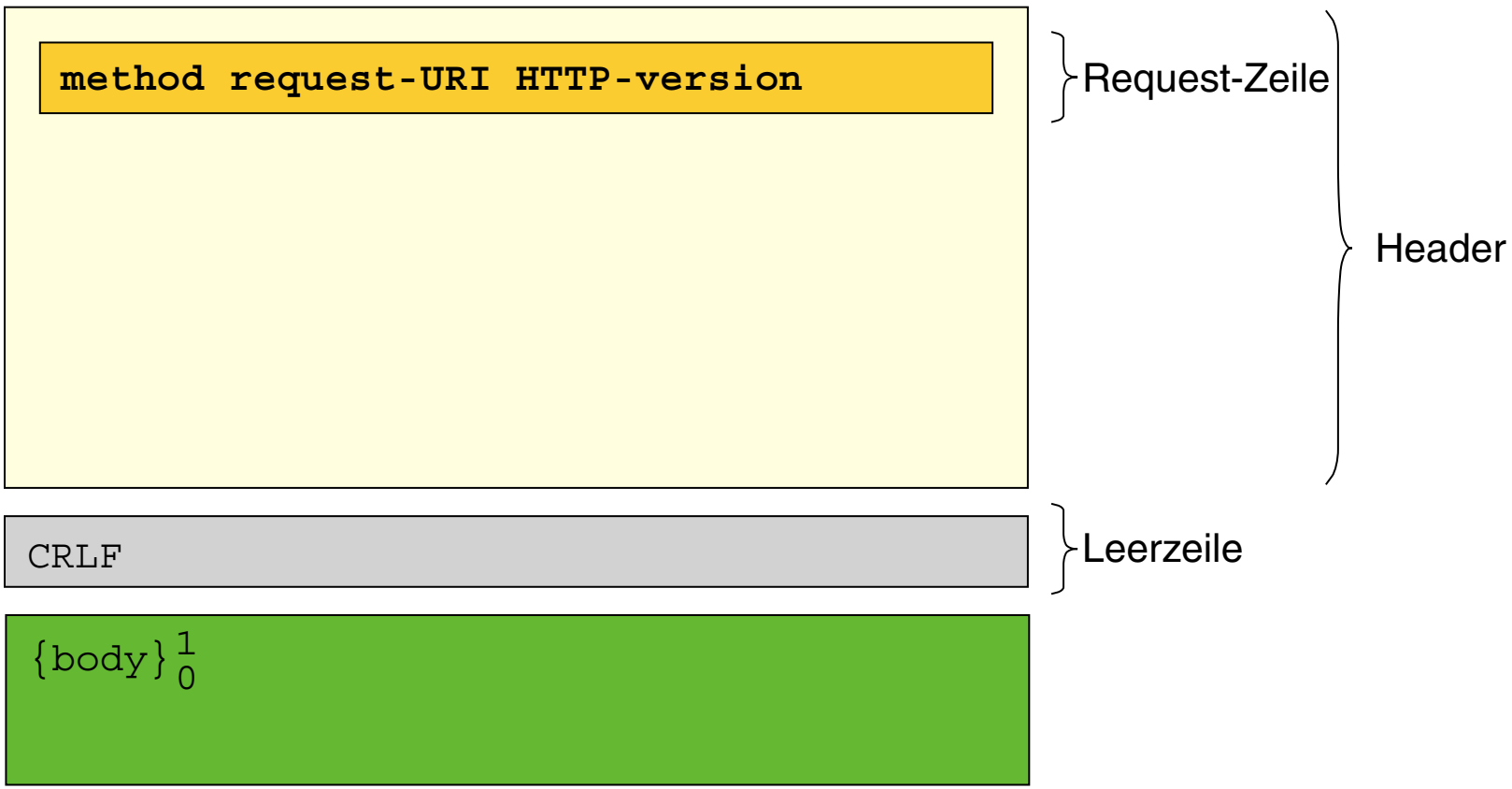
Hypertext-Transfer-Protokoll HTTP

HTTP-Request-Message [Message-Header]



Hypertext-Transfer-Protokoll HTTP

HTTP-Request-Message



Beispiel für Request-Zeile: `GET www.uni-weimar.de/index.html/ HTTP/1.0`

Hypertext-Transfer-Protokoll HTTP

HTTP-Request-Message: Methoden

Methode	
GET	Anfrage der im Request-URI angegebenen Ressource. Client-Daten wie z.B. HTML-Feldwerte werden als Bestandteil der URI der Ressource übergeben.

Hypertext-Transfer-Protokoll HTTP

HTTP-Request-Message: Methoden

Methode	
GET	Anfrage der im Request-URI angegebenen Ressource. Client-Daten wie z.B. HTML-Feldwerte werden als Bestandteil der URI der Ressource übergeben.
POST	Wie GET, jedoch werden Client-Daten nicht an die URI angehängt, sondern im Message-Body untergebracht.
HEAD	Wie GET, jedoch darf der Server keinen Message-Body zurücksenden. Wird u.a. zur Cache-Validierung verwendet.

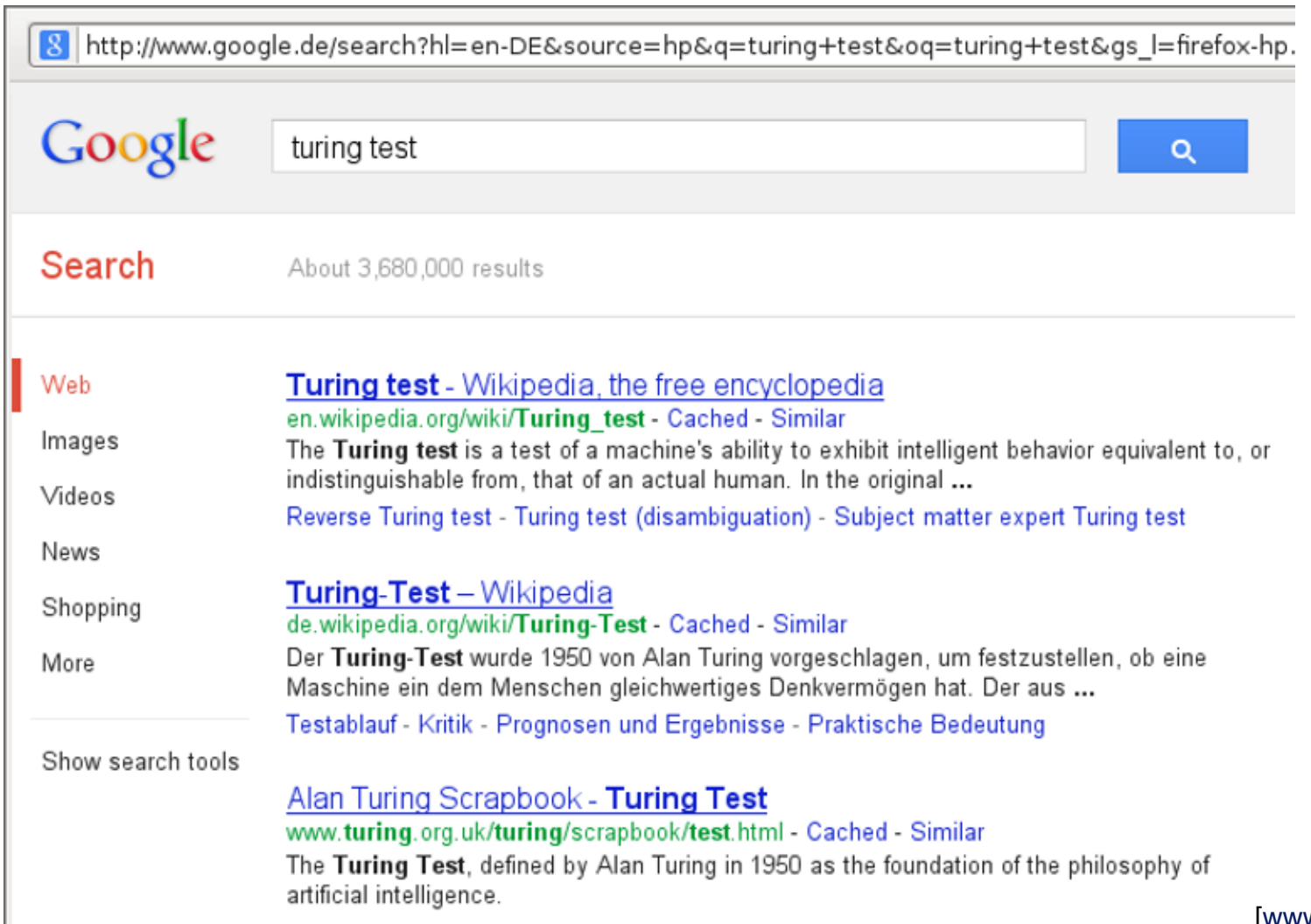
Hypertext-Transfer-Protokoll HTTP

HTTP-Request-Message: Methoden

Methode	
GET	Anfrage der im Request-URI angegebenen Ressource. Client-Daten wie z.B. HTML-Feldwerte werden als Bestandteil der URI der Ressource übergeben.
POST	Wie GET, jedoch werden Client-Daten nicht an die URI angehängt, sondern im Message-Body untergebracht.
HEAD	Wie GET, jedoch darf der Server keinen Message-Body zurücksenden. Wird u.a. zur Cache-Validierung verwendet.
PUT	Client erzeugt mit Daten des Message-Body auf dem Server eine neue Ressource an der Stelle der angegebenen Request-URI.
DELETE	Löschen der im Request-URI angegebenen Ressource auf dem Server.
OPTIONS	Abfrage der vorhandenenen Kommunikationsmöglichkeiten entlang der Verbindungsstrecke zum Server.
TRACE	Verfolgen eines Requests auf dem Weg zum Server durch die Proxies.
CONNECT	Verbindungsherstellung zum Proxy-Server, um Tunnelbetrieb einzurichten. Anwendung: Einrichtung einer SSL-Verbindung (*Secure Socket Layer*)

Hypertext-Transfer-Protokoll HTTP

HTTP-Request-Message: Beispielanfrage mittels GET-Methode (1)



[www.google.de]

Hypertext-Transfer-Protokoll HTTP

HTTP-Request-Message: Beispielanfrage mittels GET-Methode (2)

```
[stein@webis stein]$ telnet www.uni-weimar.de 80
```

Hypertext-Transfer-Protokoll HTTP

HTTP-Request-Message: Beispielanfrage mittels GET-Methode (2)

```
[stein@webis stein]$ telnet www.uni-weimar.de 80
Trying 141.54.1.34...
Connected to www.uni-weimar.de.
Escape character is '^]'.
```

Hypertext-Transfer-Protokoll HTTP

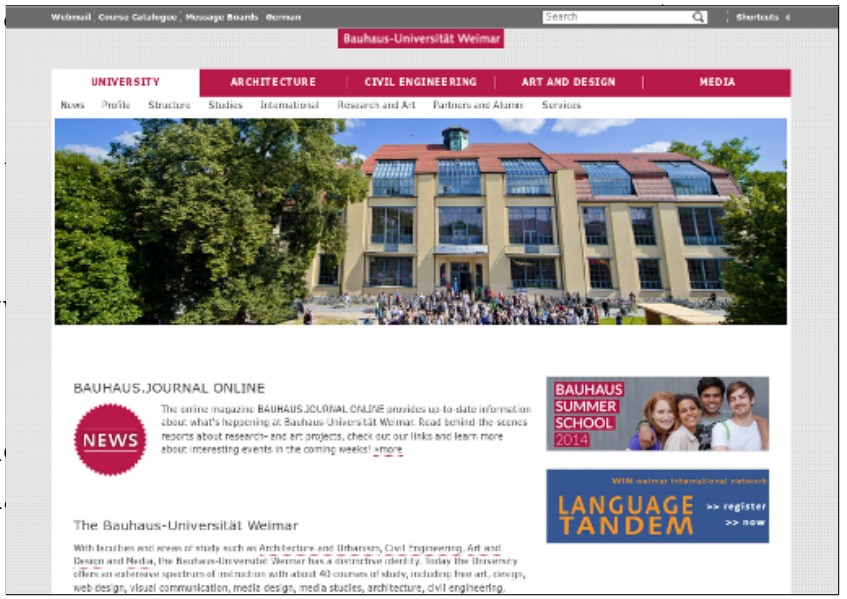
HTTP-Request-Message: Beispielanfrage mittels GET-Methode (2)

```
[stein@webis stein]$ telnet www.uni-weimar.de 80
Trying 141.54.1.34...
Connected to www.uni-weimar.de.
Escape character is '^]'.
GET /de/universitaet/start/ HTTP/1.0
HOST: www.uni-weimar.de
```

Hypertext-Transfer-Protokoll HTTP

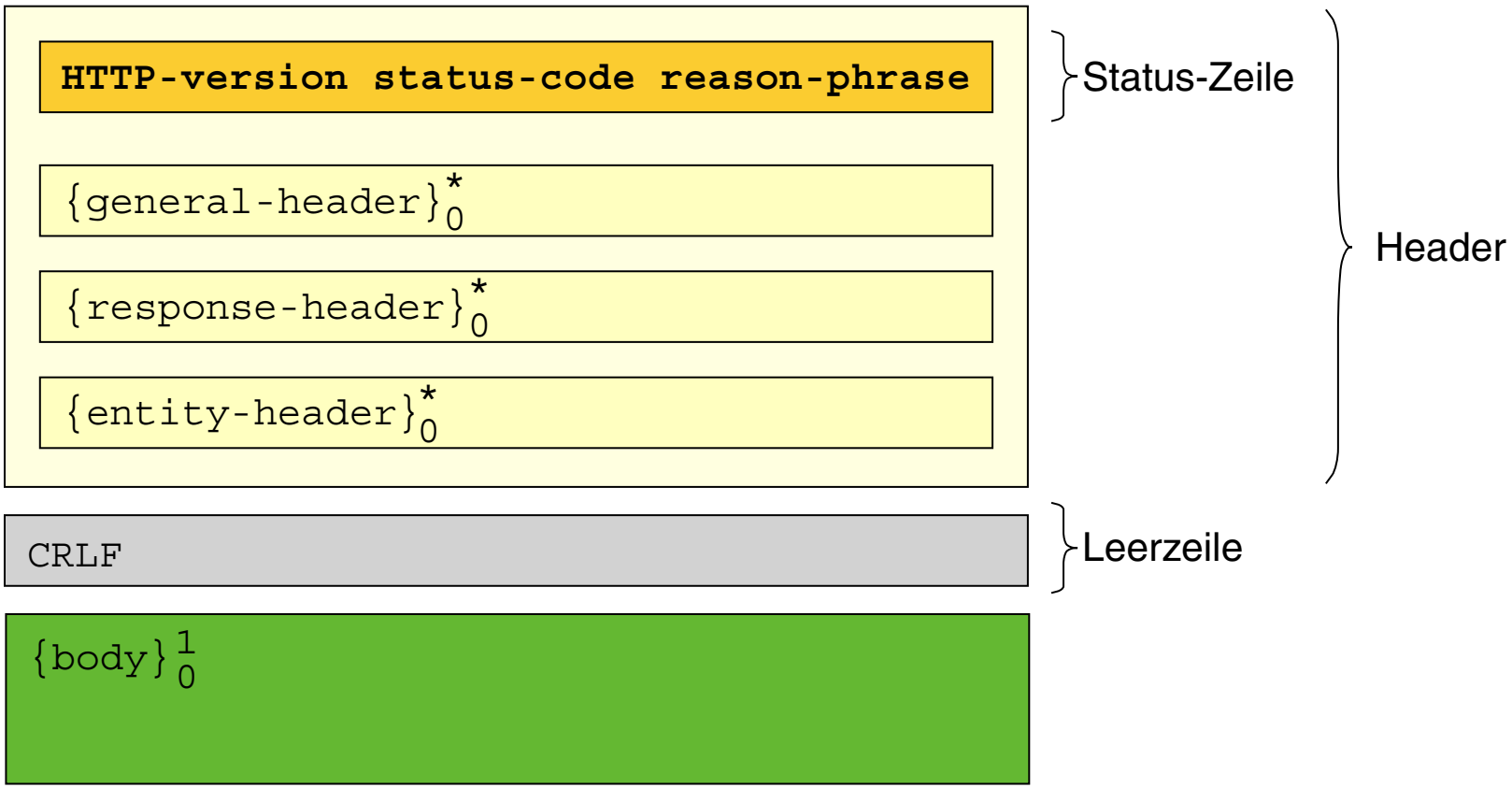
HTTP-Request-Message: Beispielanfrage mittels GET-Methode (2)

```
[stein@webis stein]$ telnet www.uni-weimar.de 80
Trying 141.54.1.34...
Connected to www.uni-weimar.de.
Escape character is '^]'.
GET /de/universitaet/start/ HTTP/1.0
HOST: www.uni-weimar.de

HTTP/1.1 200 OK
Date: Tue, 12 Apr 2016 22:53:05 GMT
Server: Apache
Set-Cookie: fe_typo_user=a2af6e53654
Cache-Control: private
Connection: close
Content-Type: text/html; charset=utf

<!DOCTYPE html>
<html lang="de-DE" xmlns="https://ww
<head>
...
<base href="https://www.uni-weimar.d
<title>Bauhaus-Universit&auml;t Weim
...
```



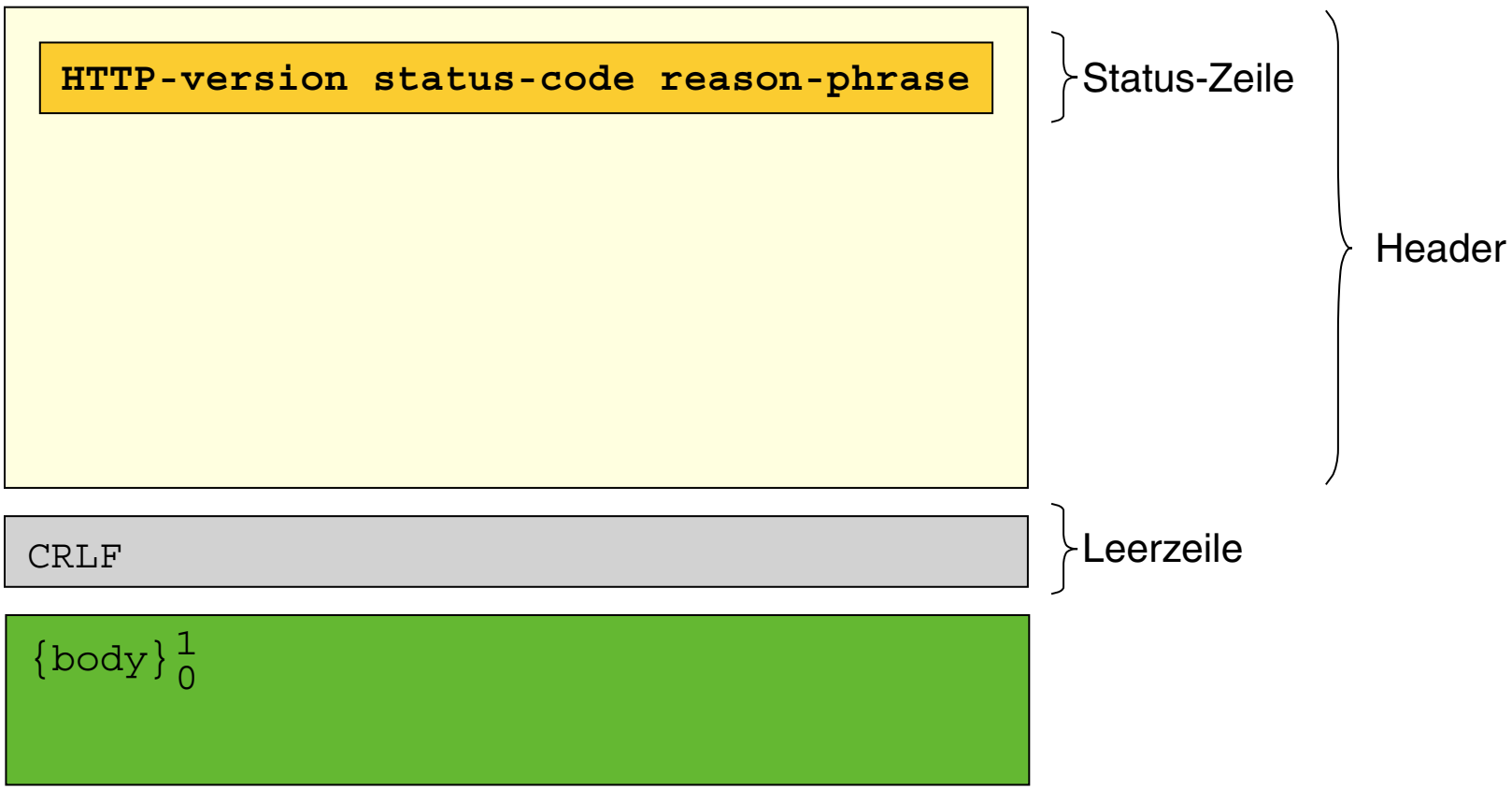
Hypertext-Transfer-Protokoll HTTP

HTTP-Response-Message [Message-Header]



Hypertext-Transfer-Protokoll HTTP

HTTP-Response-Message



Beispiel für Status-Zeile: `HTTP/1.1 200 OK`

Hypertext-Transfer-Protokoll HTTP

HTTP-Response-Message: Status-Codes

Der Status-Code besteht aus 3 Ziffern und gibt an, ob eine Anfrage erfüllt wurde bzw. welcher Fehler aufgetreten ist.

Status-Code-Kategorie		
1xx	Informational	Anforderung bekommen (selten verwendet).
2xx	Success	Anforderung bekommen, verstanden, akzeptiert und ausgeführt.
3xx	Redirection	Anweisung an den Client, an welcher Stelle die Seite zu suchen ist.
4xx	Client Error	Fehlerhafte Syntax oder unerfüllbar, da z.B. Seite nicht vorhanden.
5xx	Server Error	Server kann Anforderung nicht ausführen aufgrund eines Fehlers in der Systemsoftware oder wegen Überlastung.

Hypertext-Transfer-Protokoll HTTP

HTTP-Response-Message: Status-Codes (Fortsetzung)

100 Continue
101 Switching protocols

200 OK
201 Created
202 Accepted
203 Non-authoritative information
204 No content
205 Reset content
206 Partial content

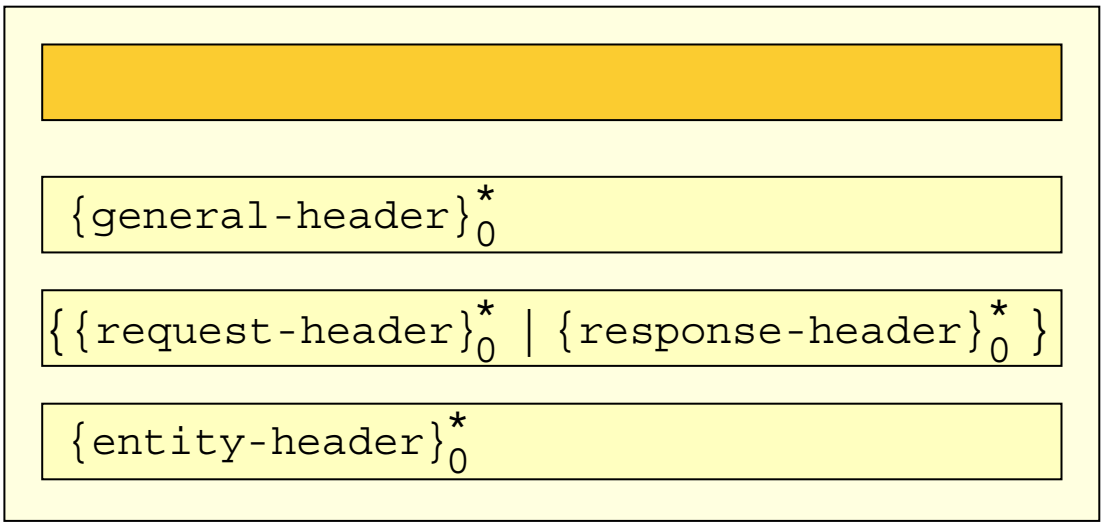
300 Multiple choices
301 Moved permanently
302 (veraltet) --> 307
303 See other
304 Not modified
305 Use proxy
307 Temporary redirect

400 Bad request
401 Unauthorized
402 Payment required
403 Forbidden
404 Not found
405 Method not allowed
406 Not acceptable
407 Proxy authentication required
408 Request timeout
409 Conflict
410 Gone
411 Length required
412 Precondition failed
413 Request entity too large
414 Request URI too large
415 Unsupported media type
416 Requested range not satisfiable
417 Expectation failed
424 Site too ugly ;-)

500 Internal server error
501 Not implemented
502 Bad gateway
503 Service unavailable
504 Gateway time out
505 HTTP version not supported

Hypertext-Transfer-Protokoll HTTP

HTTP-Message-Header [Request-Message] [Response-Message]



- General-Header
  Meta-Information zu Protokoll und Verbindung.
- Request-Header | Response-Header
  Informationen zur Anfrage oder zum Client | Antwort des Servers.
- Entity-Header
  Meta-Information über den Inhalt im Message-Body.
  Beispiele: Content-Encoding, Content-Language, Content-Type

Bemerkungen:

- BNF-Notation für den Aufbau von Header-Zeilen:

```
header   ::= keyword ':' value
keyword  ::= 'Date' | 'Server' | 'Last-Modified' | 'Set-Cookie' |
             'Content-Length' | 'Content-Type' | ...
```

- Analyse der Redirects einer URL mit HTTP-Status-Code-Checker.

Hypertext-Transfer-Protokoll HTTP

HTTP-Response-Message: Beispielantwort [Java]

```
HTTP/1.1 200 OK
Date: Tue, 15 Apr 2014 19:17:35 GMT
Server: Apache/2.2.12 (Unix) DAV/1.0.3 PHP/4.3.10
    mod_ssl/2.8.16 OpenSSL/0.9.7c
Last-Modified: Sat, 22 Mar 2014 14:11:21 GMT
ETag: "205e812-1479-42402789"
Accept-Ranges: bytes
Content-Length: 5241
Connection: close
Content-Type: text/html; charset=utf-8
```

Status line
General header
Response header
Entity header
Response header
Entity header
General header
Entity header

```
<!DOCTYPE html>
<html lang="de-DE" xmlns="http://www.w3.org/1999/...
<head>
<base href="http://www.uni-weimar.de">
<title>Bauhaus-Universit&auml;t Weimar</title>
...
```

Hypertext-Transfer-Protokoll HTTP

Content Type / MIME Type / Media Type

Medientypen bestehen aus einem Top-Level-Typ und einem Untertyp und können durch Parameter weiter spezifiziert werden. [Wikipedia]

Typ	Untertyp	Beschreibung
text	plain	unformatierter ASCII-Text
	enriched	ASCII-Text mit einfachen Formatierungen
image	gif	Standbild im GIF-Format
	jpeg	Standbild im JPEG-Format
audio	basic	Klangdaten
application	octet-stream	nicht-interpretierte Byte-Folge
	postscript	druckbares Dokument im PostScript-Format
...	...	...

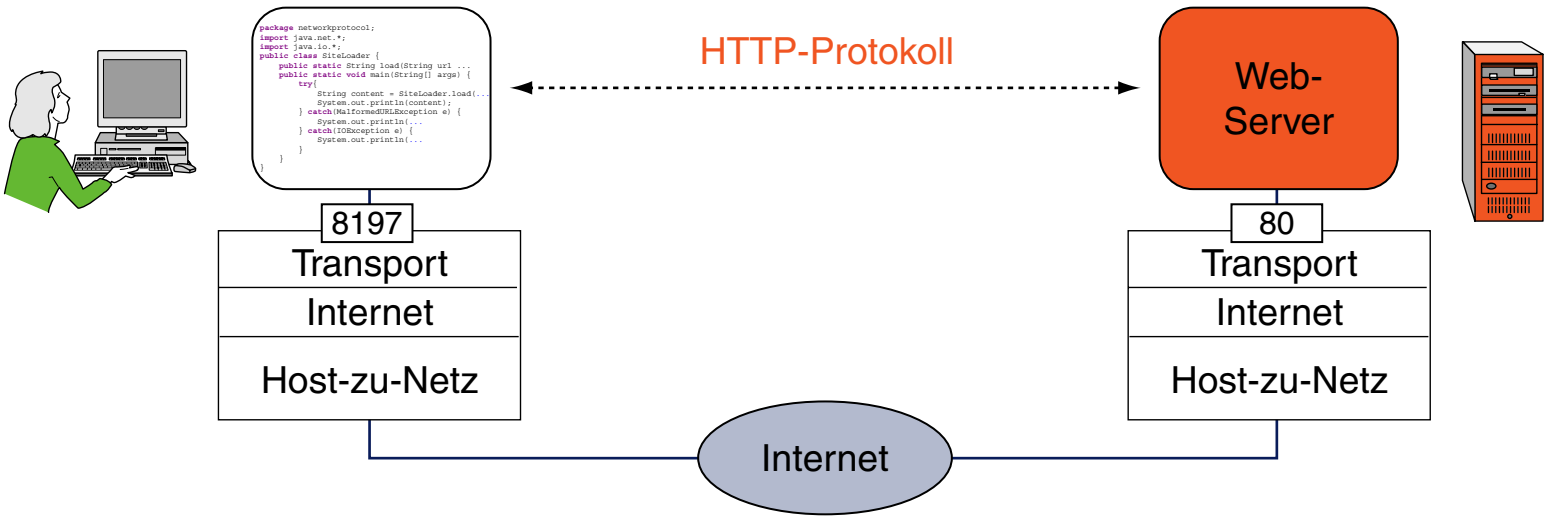
Eine Übersicht der erlaubten Medientypen findet sich bei der IANA.

Bemerkungen:

- Die MIME-Type-Spezifikation ist in RFC 2046 beschrieben.
- Die Abkürzung MIME steht für Multipurpose Internet Mail Extensions. Die zugehörigen Spezifikationen finden Anwendung bei der Deklaration von Inhalten in Internetprotokollen.

Hypertext-Transfer-Protokoll HTTP

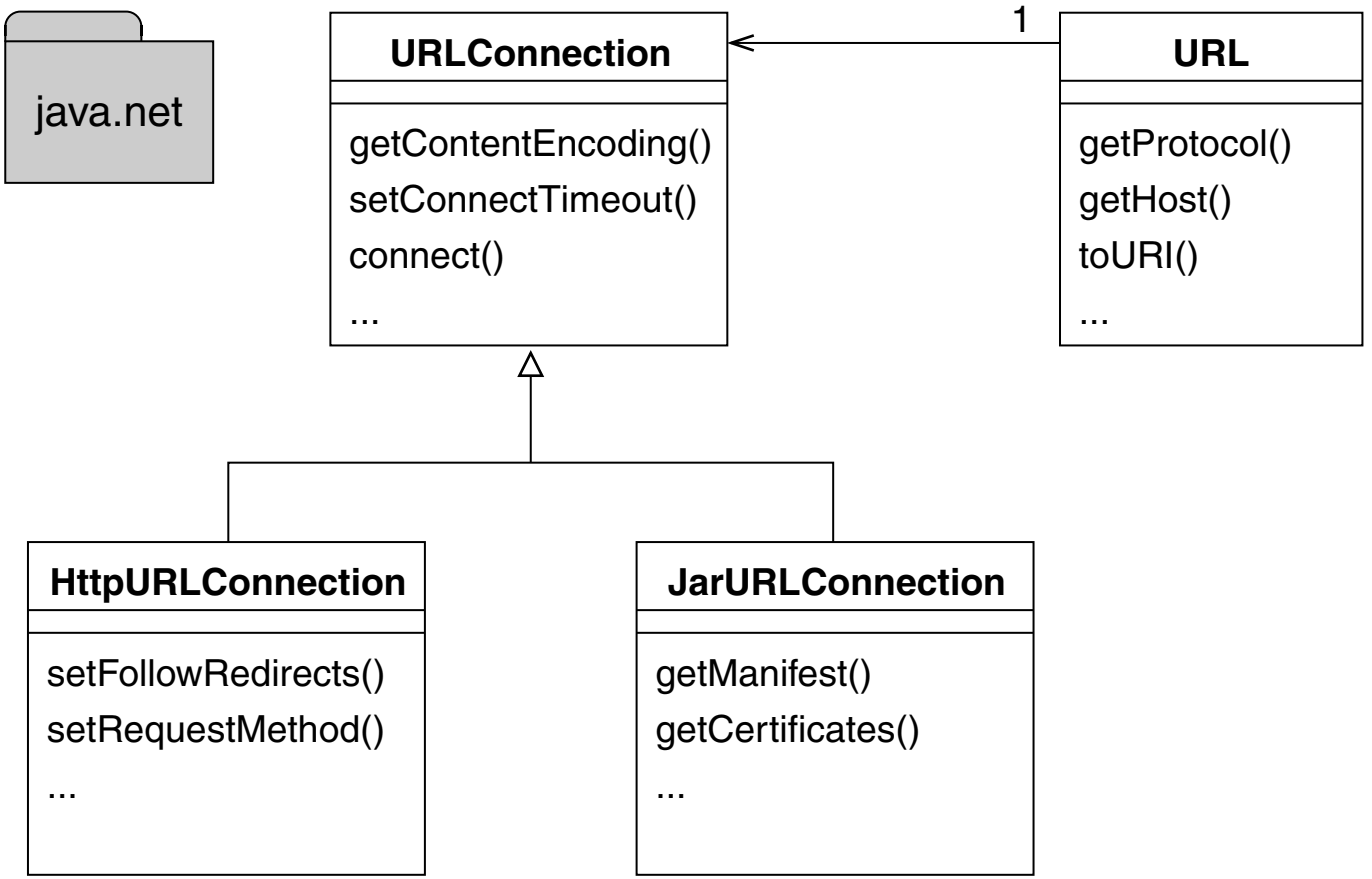
HTTP-Kommunikation mit Java [Server-side]



Auf der Server-Seite bildet ein WWW-Server das Gegenstück einer Verbindung zu dem Beispiel-Client. Der WWW-Server lässt sich auf diese Art anfragen, weil das HTTP-Protokoll vom Client korrekt abgewickelt wird. [RFC 2616]

Hypertext-Transfer-Protokoll HTTP

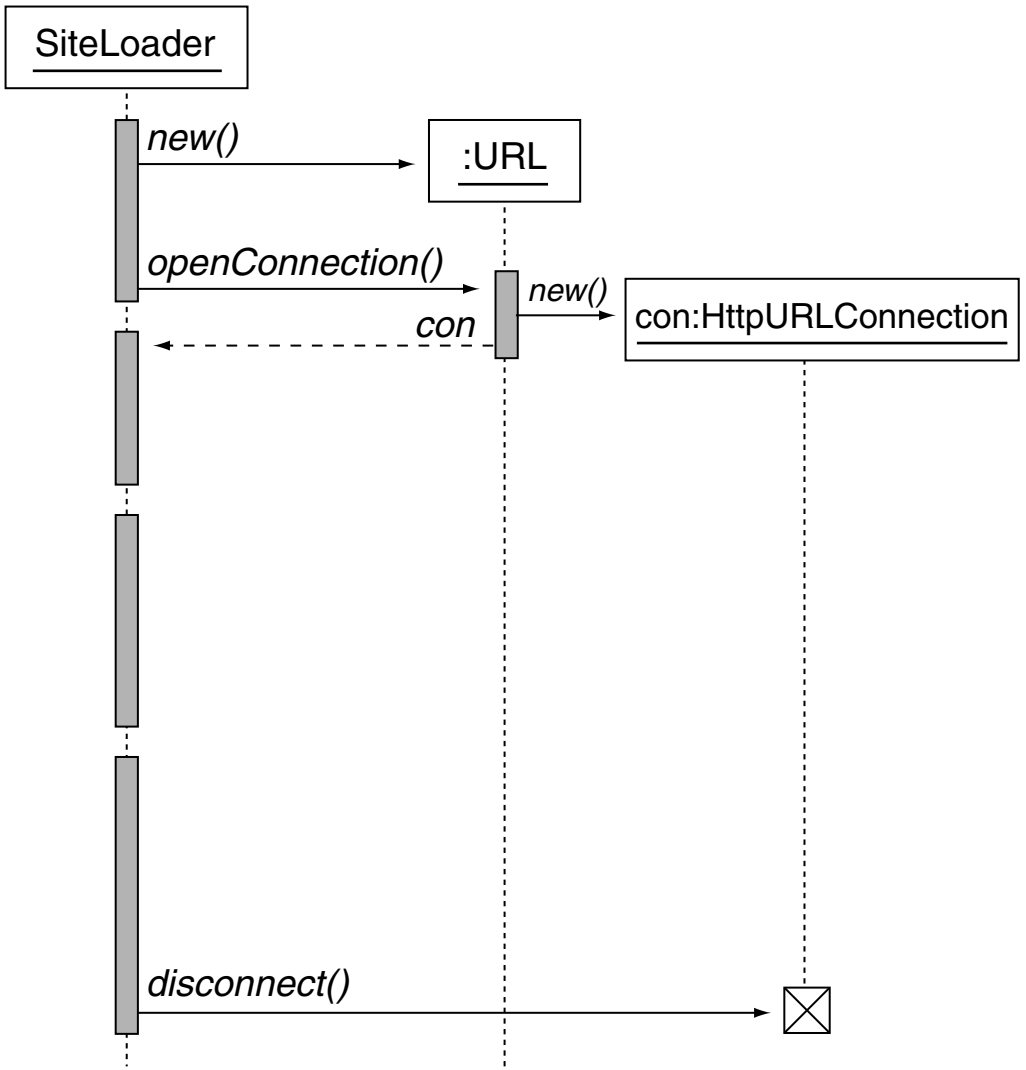
HTTP-Kommunikation mit Java



[Javadoc]

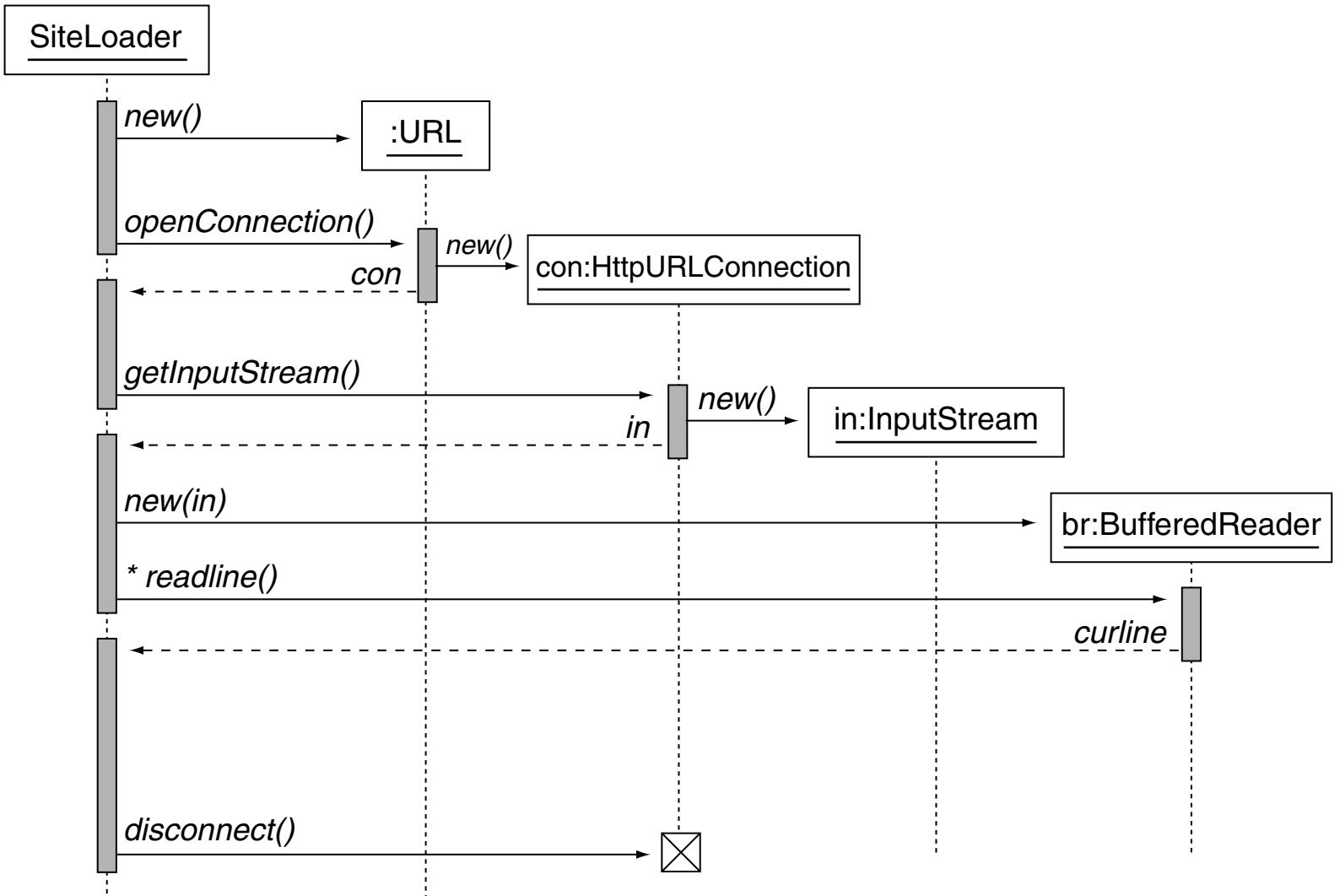
Hypertext-Transfer-Protokoll HTTP

HTTP-Kommunikation mit Java



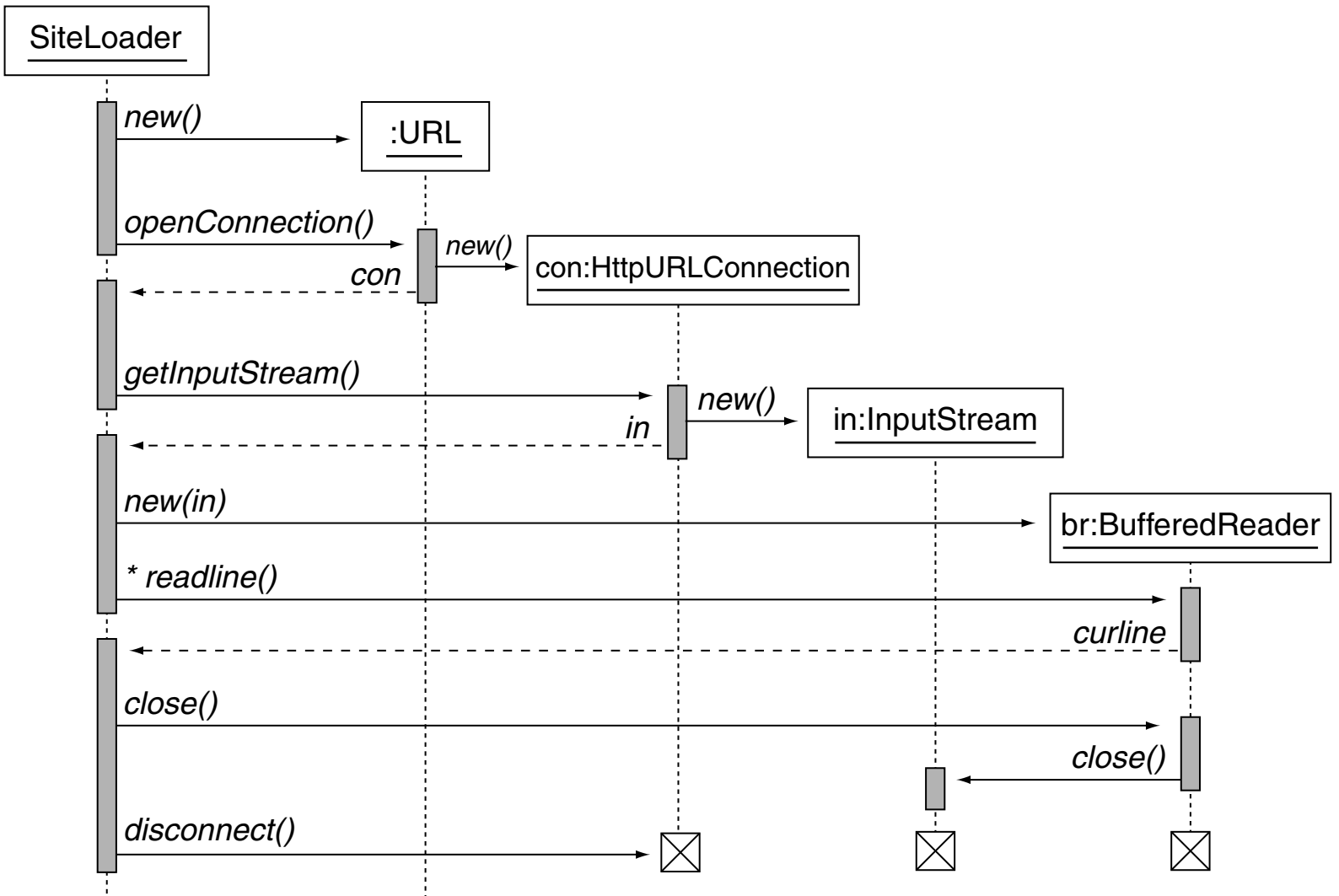
Hypertext-Transfer-Protokoll HTTP

HTTP-Kommunikation mit Java (Fortsetzung)



Hypertext-Transfer-Protokoll HTTP

HTTP-Kommunikation mit Java (Fortsetzung)



Hypertext-Transfer-Protokoll HTTP

HTTP-Kommunikation mit Java [Response-Message]

```
public static String load(String urlString) throws IOException {
    URL url = new URL(urlString);
    HttpURLConnection con = (HttpURLConnection) url.openConnection();

    System.out.println(con.getResponseCode() + " "
        + con.getResponseMessage()); // will print "200 OK"

    InputStream in = con.getInputStream();
    BufferedReader br = new BufferedReader(new InputStreamReader(in));
    String curline;
    StringBuilder content = new StringBuilder();

    while ((curline = br.readLine()) != null) {
        content.append(curline + '\n');
    }

    br.close();
    con.disconnect();
    return content.toString();
 }
```

Hypertext-Transfer-Protokoll HTTP

HTTP-Kommunikation mit Java (Fortsetzung)

```
package networkprotocol;
import java.net.*;
import java.io.*;

public class SiteLoader {

    public static String load(String urlString) ...

    public static void main(String[] args) {
        try{
            String content = SiteLoader.load("http://www.heise.de");
            System.out.println(content);
        }
        catch(MalformedURLException e) {
            System.out.println("MalformedURLException:" + e.getMessage());
        }
        catch(IOException e) {
            System.out.println("IOException:" + e.getMessage());
        }
    }
}
```

Hypertext-Transfer-Protokoll HTTP

HTTP-Kommunikation mit Java (Fortsetzung)

```
[stein@webis bin]$ java networkprotocol.SiteLoader | less
200 OK
<!DOCTYPE html>
<html lang="de">

<head>
    <title>heise online - IT-News, Nachrichten und Hintergründe
      </title>
        <meta name="description" content="News und Foren zu Computer, IT, ...
            <meta name="keywords" content="heise online, c't, iX, Technology ...
        <script type="text/javascript" src="/js/mobi/webtrekk_abtest.js"></script>

<meta charset="utf-8">
<meta name="publisher" content="Heise Medien" />
<meta name="viewport" content="width=1175" />
<link rel="alternate" media="only screen and...
<link rel="copyright" title="Copyright" href="/impressum.html" />

                <!--googleoff: all-->
    <meta property="fb:page_id"     content="333992367317" />
    <meta property="og:title"       content="IT-News, c&#39;t, iX, Technology ...
    <meta property="og:type"        content="website" />
    <meta property="og:locale"      content="de_DE" />
    <meta property="og:url"         content="http://www.heise.de/" />
    ...
```

Fortgeschrittene HTTP-Konzepte

Session-Management

HTTP ist ein zustandsloses Protokoll = ein Protokoll „ohne Gedächtnis“.

Ein zustandsloses Protokoll bezieht keine Information aus bereits stattgefundener Kommunikation ein.

Eine Session beschreibt einen Dialog, der sich über mehrere Anfrage/Antwort-Zyklen erstreckt. Innerhalb einer Session soll sich auf die vorangegangene Kommunikation bezogen werden können.

Fortgeschrittene HTTP-Konzepte

Session-Management

HTTP ist ein zustandsloses Protokoll = ein Protokoll „ohne Gedächtnis“.

Ein zustandsloses Protokoll bezieht keine Information aus bereits stattgefundener Kommunikation ein.

Eine Session beschreibt einen Dialog, der sich über mehrere Anfrage/Antwort-Zyklen erstreckt. Innerhalb einer Session soll sich auf die vorangegangene Kommunikation bezogen werden können.

Techniken zur Codierung von Session-Information:

1. URL Rewriting
   Session-Information wird in der URL codiert.
2. Cookies
   Session-Information wird beim WWW-Client gespeichert.
3. Hidden Fields
   Session-Information wird in unsichtbaren Formularfeldern untergebracht.

Fortgeschrittene HTTP-Konzepte

Session-Management

HTTP ist ein zustandsloses Protokoll = ein Protokoll „ohne Gedächtnis“.

Ein zustandsloses Protokoll bezieht keine Information aus bereits stattgefundener Kommunikation ein.

Eine Session beschreibt einen Dialog, der sich über mehrere Anfrage/Antwort-Zyklen erstreckt. Innerhalb einer Session soll sich auf die vorangegangene Kommunikation bezogen werden können.

Techniken zur Codierung von Session-Information:

1. URL Rewriting
   Session-Information wird in der URL codiert.
2. Cookies
   Session-Information wird beim WWW-Client gespeichert.
3. Hidden Fields
   Session-Information wird in unsichtbaren Formularfeldern untergebracht.

Fortgeschrittene HTTP-Konzepte

Session-Management: Cookies

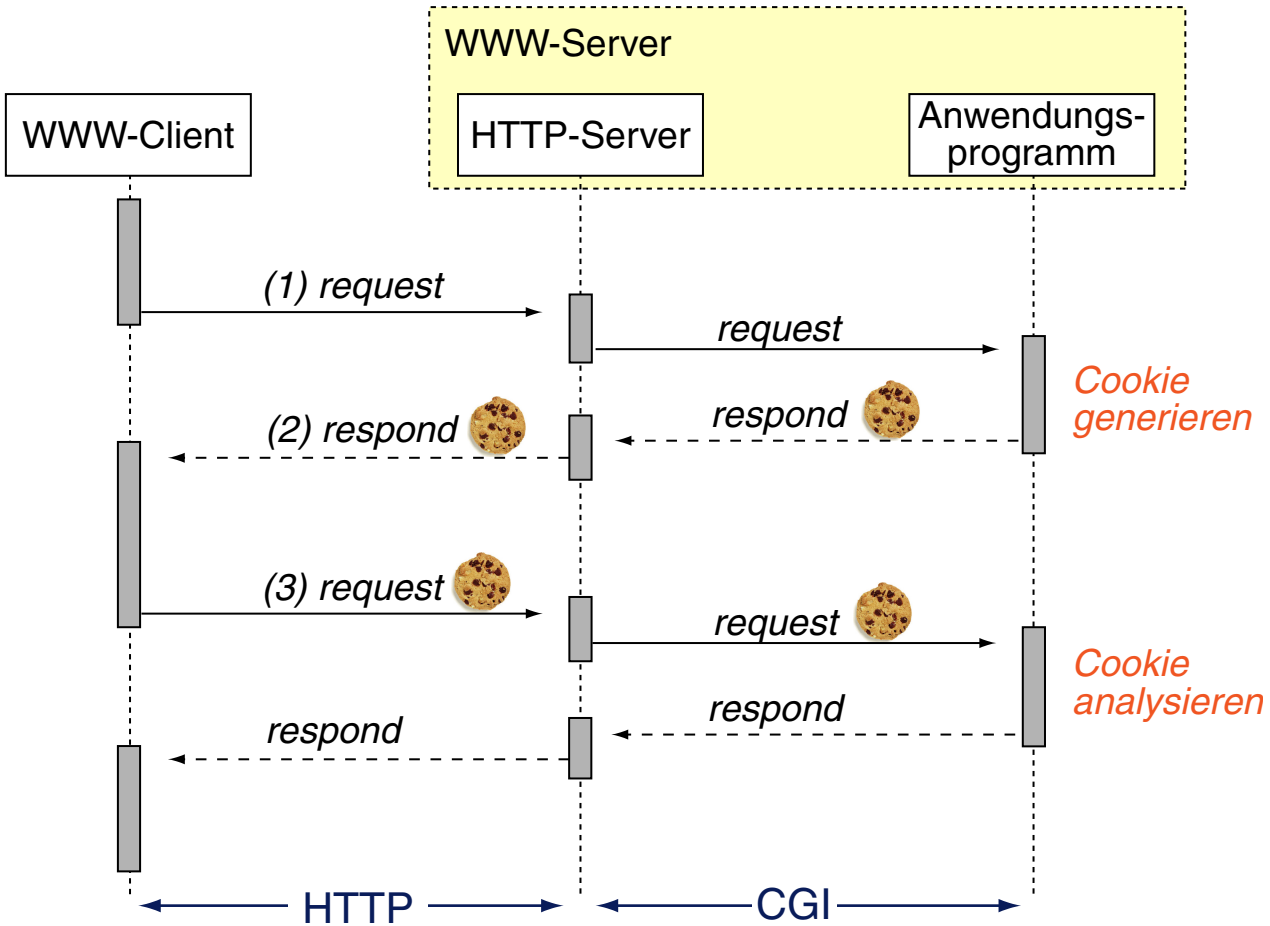
Ein Cookie (Keks) ist eine Zeichenkette (<4KB), die zwischen WWW-Client und WWW-Server ausgetauscht wird. Standardisiert in RFC 6265.

Verwendungsmöglichkeiten von Cookies:

- Session-Management
- Benutzer-Authentisierung
- Seitenabfolgesteuerung
- Erstellung von Nutzerprofilen
- Generierung nutzerspezifischer Seiten
- Informationsaustausch ergänzend zum HTTP-Protokoll

Fortgeschrittene HTTP-Konzepte

Session-Management: Cookies



[Meinel/Sack 2004]

Fortgeschrittene HTTP-Konzepte

Session-Management: Cookies

1. WWW-Client fragt Google-Startseite an:

```
Ethernet II, Src: 00:0c:f1:e8:fe:be, Dst: 00:00:0c:07:ac:01
Internet Protocol, Src Addr: 141.54.178.123, Dst Addr: 66.249.85.99
Transmission Control Protocol, Src Port: 1577, Dst Port: http (80), ...
Hypertext Transfer Protocol
    GET / HTTP/1.1
    Accept: */*
    Accept-Language: de
    Accept-Encoding: gzip, deflate
    User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0)
    Host: www.google.de
    Connection: Keep-Alive
```

Fortgeschrittene HTTP-Konzepte

Session-Management: Cookies

1. WWW-Client fragt Google-Startseite an:

```
Ethernet II, Src: 00:0c:f1:e8:fe:be, Dst: 00:00:0c:07:ac:01
Internet Protocol, Src Addr: 141.54.178.123, Dst Addr: 66.249.85.99
Transmission Control Protocol, Src Port: 1577, Dst Port: http (80), ...
Hypertext Transfer Protocol
    GET / HTTP/1.1
    Accept: */*
    Accept-Language: de
    Accept-Encoding: gzip, deflate
    User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0)
    Host: www.google.de
    Connection: Keep-Alive
```

5	Anwendung:	HTTP-Message
4	Transport:	Port
3	Internet:	IP-Adresse
1+2	Host-zu-Netz:	Mac-Adresse

Fortgeschrittene HTTP-Konzepte

Session-Management: Cookies (Fortsetzung)

2. WWW-Server antwortet:

```
Ethernet II, Src: 00:09:e9:a6:0b:fc, Dst: 00:0c:f1:e8:fe:be
Internet Protocol, Src Addr: 66.249.85.99, Dst Addr: 141.54.178.123
Transmission Control Protocol, Src Port: http (80), Dst Port: 1577, ...
```

```
    Transfer Protocol
    HTTP/1.1 200 OK
    Cache-Control: private
    Content-Type: text/html
    Set-Cookie: PREF=ID=8f6a3f6fad95a9ee:LD=de:[...]; domain=.google.de
    Content-Encoding: gzip
    Server: GWS/2.1
    Content-Length: 1487
    Date: Mon, 25 Apr 2005 11:26:57 GMT

    Content-encoded entity body (gzip)
    Line-based text data: text/html
    <html><head><meta http-equiv=content-typecontent=text/html; ...
```

5	Anwendung:	HTTP-Message
4	Transport:	Port
3	Internet:	IP-Adresse
1+2	Host-zu-Netz:	Mac-Adresse

Fortgeschrittene HTTP-Konzepte

Session-Management: Cookies (Fortsetzung)

3. WWW-Client sendet Google-Query „test“:

```
Ethernet II, Src: 00:0c:f1:e8:fe:be, Dst: 00:00:0c:07:ac:01
Internet Protocol, Src Addr: 141.54.178.123, Dst Addr: 66.249.85.99
Transmission Control Protocol, Src Port: 1577, Dst Port: http (80), ...
Hypertext Transfer Protocol
    GET /search?hl=de&q=test&meta= HTTP/1.1
    Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, ...
    Referer: https://www.google.de/
    Accept-Language: de
    Accept-Encoding: gzip, deflate
    User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0)
    Host: www.google.de
    Connection: Keep-Alive
    Cookie: PREF=ID=8f6a3f6fad95a9ee:LD=de: ...
```

Fortgeschrittene HTTP-Konzepte

Session-Management: Cookies (Fortsetzung)

3. WWW-Client sendet Google-Query „test“:

```
Ethernet II, Src: 00:0c:f1:e8:fe:be, Dst: 00:00:0c:07:ac:01
Internet Protocol, Src Addr: 141.54.178.123, Dst Addr: 66.249.85.99
Transmission Control Protocol, Src Port: 1577, Dst Port: http (80), ...
Hypertext Transfer Protocol
    GET /search?hl=de&q=test&meta= HTTP/1.1
    Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, ...
    Referer: https://www.google.de/
    Accept-Language: de
    Accept-Encoding: gzip, deflate
    User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.0)
    Host: www.google.de
    Connection: Keep-Alive
    Cookie: PREF=ID=8f6a3f6fad95a9ee:LD=de: ...
```

Auf Client-Seite gespeicherter Cookie:

```
PREF
ID=8f6a3f6fad95a9ee:LD=de:TM=1114428417:LM=1114428417:S=mDw2bedrst81ASKd
google.de/
1536 2618878336 32111634 3193845632 29706633 *
```

Bemerkungen:

- Nicht der Web-Server, sondern ein Anwendungsprogramm ist am Setzen von Cookies interessiert.
- Der Client kann Cookies mit einer Lebensdauer versehen und spezifisch für Web-Seiten erlauben oder sperren.
- Der Client wählt das passende Cookie anhand der URL der Informationsquelle aus und schickt es mit der Anfrage zum Web-Server. Dabei darf der Client nur Cookies an denjenigen Web-Server senden, von dem diese Cookies stammen.
- Der Client bringt Cookie-Information im Request-Header `Cookie` unter.
- Der Server bringt Cookie-Information im Reponse-Header `Set-Cookie` unter.
- Der Internet-Explorer unter Windows speichert Cookies im Verzeichnis `Dokumente und Einstellungen/Cookies`.
- Analyse des Protokollstapels mit dem Programm wireshark.

Fortgeschrittene HTTP-Konzepte

Content-Negotiation

Ressourcen auf dem WWW können in sprachspezifischen, qualitätspezifischen oder codierungsspezifischen Varianten vorliegen, besitzen jedoch dieselbe URI.

Seit HTTP/1.1 können WWW-Client und WWW-Server aushandeln, welche der angebotenen Varianten einer Informationsressource geliefert werden soll.

Arten der Content-Negotiation:

1. Server-driven. WWW-Server verantwortlich für Auswahl.
   Kriterien: Header des HTTP-Requests, Informationen über die Ressourcen

2. Agent-driven. WWW-Client verantwortlich für Auswahl.
   Client informiert sich zuerst, trifft dann Auswahlentscheidung.

3. Transparent. Proxy-Server verhandelt in der Agenten-Rolle.
   Vorteile: Lastverteilung, WWW-Client stellt nur eine Anfrage.

Fortgeschrittene HTTP-Konzepte

Content-Negotiation

Ressourcen auf dem WWW können in sprachspezifischen, qualitätspezifischen oder codierungsspezifischen Varianten vorliegen, besitzen jedoch dieselbe URI.

Seit HTTP/1.1 können WWW-Client und WWW-Server aushandeln, welche der angebotenen Varianten einer Informationsressource geliefert werden soll.

Arten der Content-Negotiation:

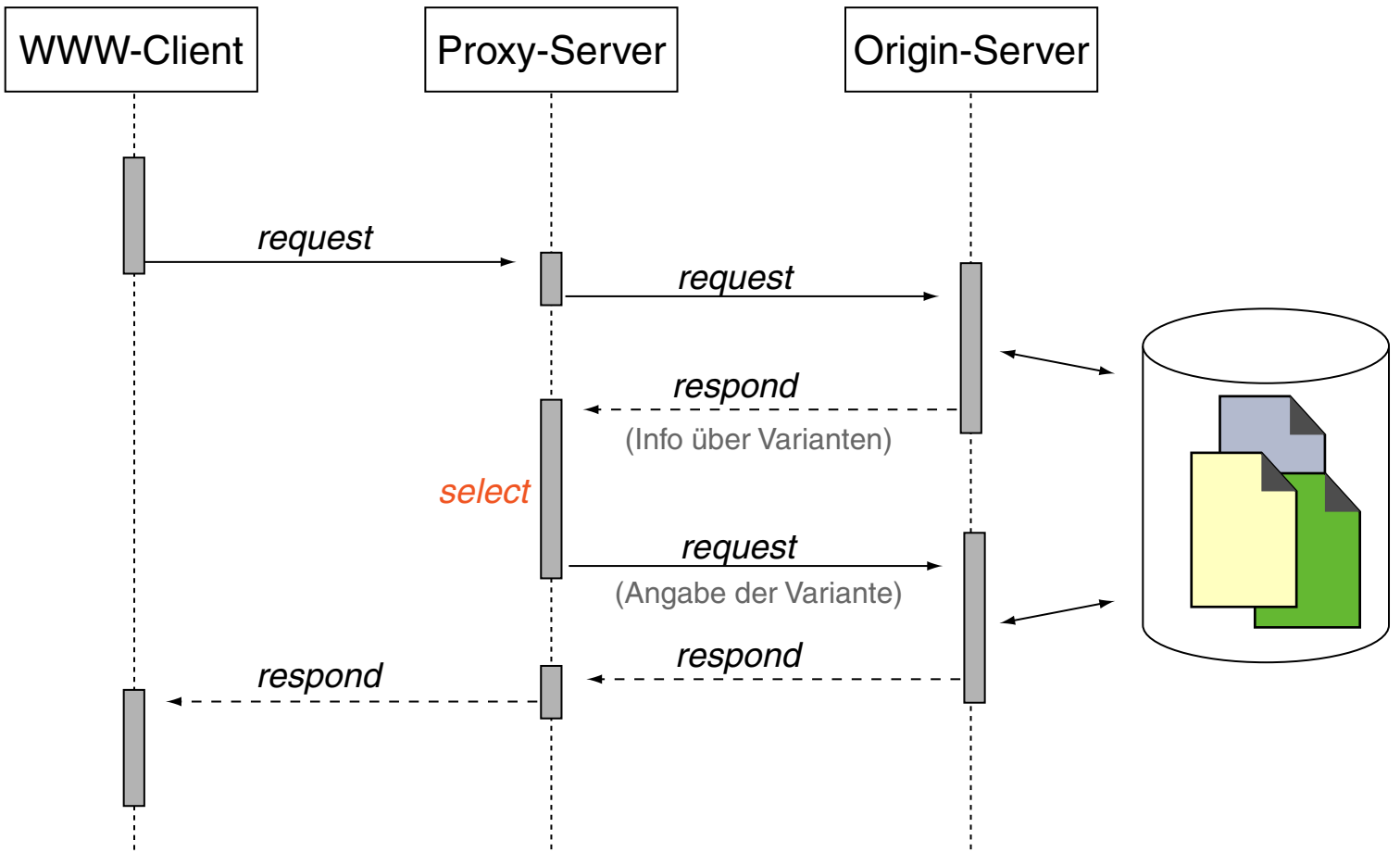
1. Server-driven. WWW-Server verantwortlich für Auswahl.
   Kriterien: Header des HTTP-Requests, Informationen über die Ressourcen

2. Agent-driven. WWW-Client verantwortlich für Auswahl.
   Client informiert sich zuerst, trifft dann Auswahlentscheidung.

3. Transparent. Proxy-Server verhandelt in der Agenten-Rolle.
   Vorteile: Lastverteilung, WWW-Client stellt nur eine Anfrage.

Fortgeschrittene HTTP-Konzepte

Content-Negotiation: transparent



Bemerkungen:

- Die drei Varianten der Content-Negotiation unterscheiden sich dahingehend, wer die select-Operation ausführt.
- Vorteil der Server-driven Content-Negotiation: der Server ist nah an den Ressourcen und hat den besten Überblick. Nachteile: der Server ist auf Information aus den Request-Headern des Clients angewiesen, ist nicht so performant.
- Vorteil der Agent-driven Content-Negotiation: der Client weiß genau, was er will. Nachteil: Overhead an Kommunikation. Erst nachfragen, was es alles gibt, dann Auswahl.

Fortgeschrittene HTTP-Konzepte

Persistente Verbindungen

HTTP/1.0 öffnet für jede Anfrage eine Verbindung, die nach jeder Antwort unmittelbar geschlossen wird:

- \+ Protokoll sehr einfach, leicht zu implementieren
- \+ Verbindungsabbruch signalisiert auch Abschluss der HTTP-Antwort
- – Verbindungsaufbau zeit- und ressourcenintensiv

HTTP/1.1 hat persistente Verbindungen, die Client-seitig mit `Connection: close` (General-Header) oder nach einem Timeout geschlossen werden:

- \+ effizientere Nutzung von Betriebssystemressourcen, weniger Pakete
- \+ *Pipelining*: Versenden von Anfragen ohne auf Antwort zu warten
- – aufwändigeres Protokoll, da *Chunked Encoding* notwendig

HTTP/2 hat Multiplexing (und andere Verbesserungen [HTTP/2 Home]).

Fortgeschrittene HTTP-Konzepte

Persistente Verbindungen

HTTP/1.0 öffnet für jede Anfrage eine Verbindung, die nach jeder Antwort unmittelbar geschlossen wird:

- \+ Protokoll sehr einfach, leicht zu implementieren
- \+ Verbindungsabbruch signalisiert auch Abschluss der HTTP-Antwort
- – Verbindungsaufbau zeit- und ressourcenintensiv

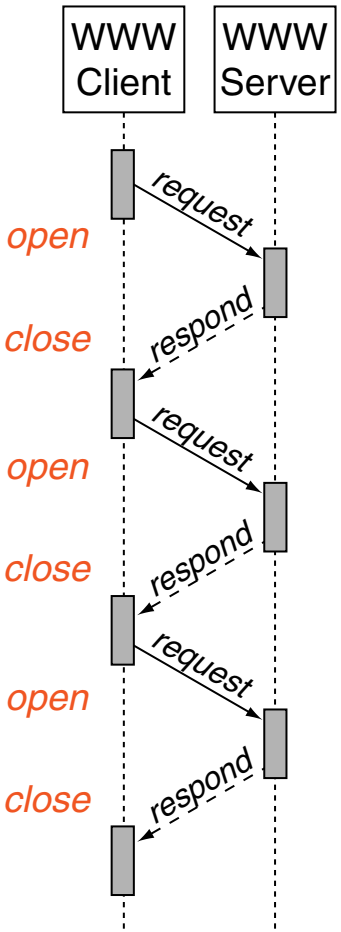
HTTP/1.1 hat persistente Verbindungen, die Client-seitig mit `Connection: close` (General-Header) oder nach einem Timeout geschlossen werden:

- \+ effizientere Nutzung von Betriebssystemressourcen, weniger Pakete
- \+ *Pipelining*: Versenden von Anfragen ohne auf Antwort zu warten
- – aufwändigeres Protokoll, da *Chunked Encoding* notwendig

HTTP/2 hat Multiplexing (und andere Verbesserungen [HTTP/2 Home]).

Fortgeschrittene HTTP-Konzepte

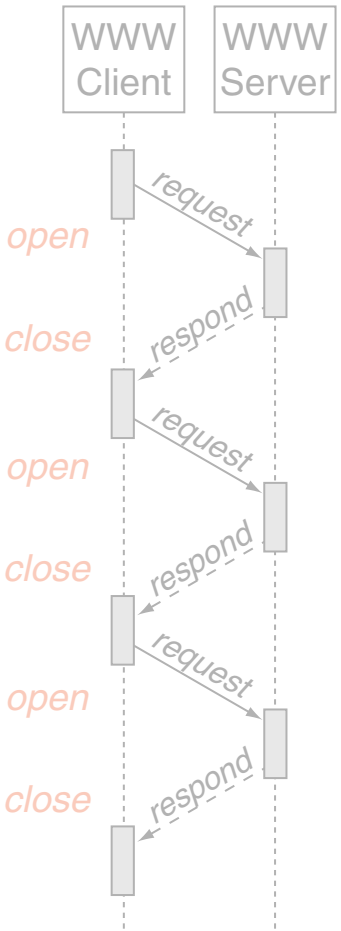
Persistente Verbindungen



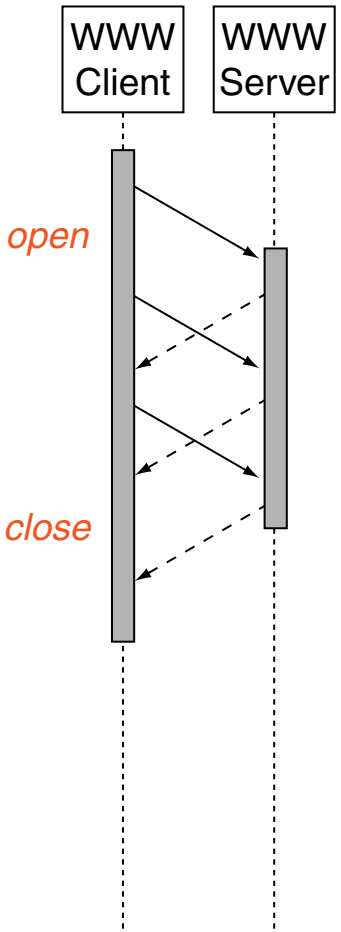
HTTP/1.0

Fortgeschrittene HTTP-Konzepte

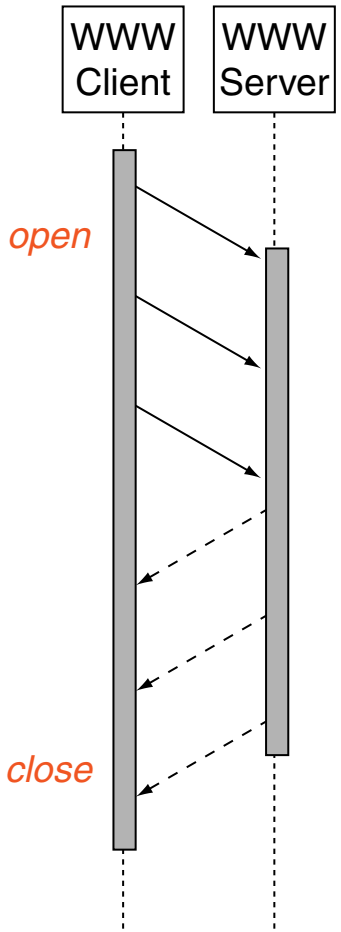
Persistente Verbindungen



HTTP/1.0



HTTP/1.1
(ideal)



HTTP/1.1
(practice)

Fortgeschrittene HTTP-Konzepte

Persistente Verbindungen