

# Inhalt der Vorlesung „Rechnerkommunikation“

- ✓ Einführung
- **Anwendungsschicht**
- Transportschicht
- Netzwerkschicht
- Verbindungsschicht
- Physikalische Schicht

# Anwendungsschicht

## ■ Einführung

## ■ Verbreitete Anwendungen

- Hypertext Transfer Protocol (HTTP)
- File Transfer Protocol (FTP)
- E-Mail
- Netzwerkmanagement
- Domain Name System (DNS)
- Content Distribution Networks

## ■ Socket-Programmierung

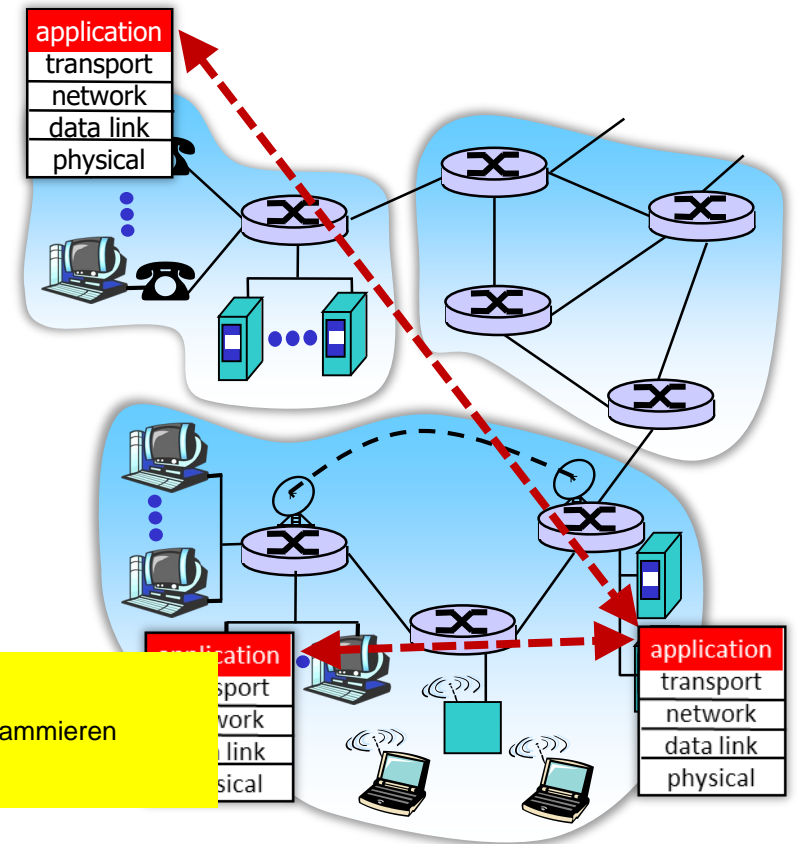
## ■ Peer-to-Peer-Systeme

# Einführung

## ■ Netzwerkanwendung

- **Anwendungsprozesse** auf verschiedenen Endsystemen (Hosts), die mittels **Nachrichten** über ein Netzwerk kommunizieren
- kann direkt unter Verwendung der **Dienste der Transportschicht** implementiert werden
- standardisierte Anwendungen benutzen ein **Anwendungsprotokoll**, das **das Format der Nachrichten und das Verhalten beim Empfang von Nachrichten festlegt**
- z.B: Web-Browser und Web-Server
- die unteren Schichten und der **Netzwerkkern benötigen keine Kenntnis der Anwendung**
- einfache Verbreitung, große Dynamik

u.U. auch ohne protokoll über socket-programmieren

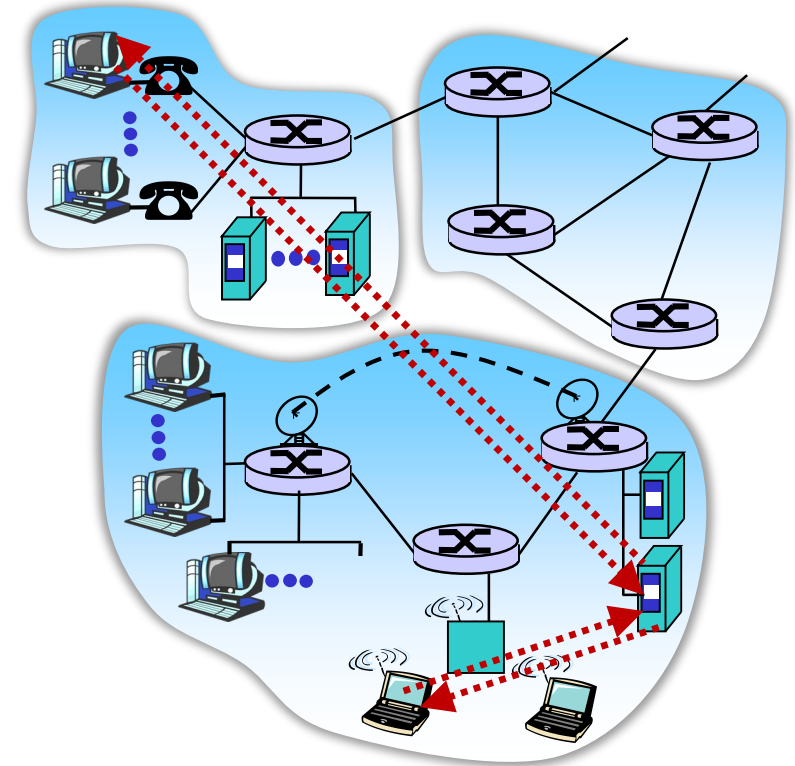


Quelle: Kurose, Ross.  
*Computer Networking: A Top-Down Approach*, 7th Ed.,  
Pearson Education, 2017.

# Einführung

## ■ Client-Server-Paradigma

- Server stellt Dienst zur Verfügung, der von Client angefordert wird
- übliches Paradigma von vielen traditionellen Anwendungen, wie z.B. Web-Browser und Web-Server
- typische Eigenschaften des Servers
  - leistungsfähig
  - immer verfügbar
- typische Eigenschaften der Clients
  - nur manchmal verbunden
  - kommunizieren mit Server, nicht untereinander



Quelle: Kurose, Ross.  
*Computer Networking: A Top-Down Approach*, 7th Ed.,  
Pearson Education, 2017.

# Einführung

- Client-Server-Paradigma ist **zentralisierte Architektur**
- Weitere Paradigmen
  - **wechselnde Rolle von Client und Server:** Hosts übernehmen mal die eine, mal die andere Rolle (z.B. bei Web-Caching oder **SMTP**)
  - **verteilte Anwendung:** besteht aus mehreren **unabhängigen Anwendungen**, die zusammen wie eine **einzelne Anwendung erscheinen** (z.B. WebShop mit **Web-Server, Applikations-Server und Datenbank**), Koordination ist zwar verteilt, findet aber für das Gesamtsystem statt
  - **Peer-to-Peer (P2P):** Vernetzung von Gleichen
    - **dezentrale Architektur:** autonome sich selbst organisierende Systeme ohne globale Steuerung (z.B. einige Peer-to-Peer-Anwendungen wie Gnutella, Chord, Bitcoin)
    - **Hybridarchitektur:** zur Initialisierung ist eine zentrale Architektur nötig, die Anwendung findet dann dezentral direkt zwischen Hosts statt (z.B. bei Session Initiation Protocol, SIP oder bei manchen Peer-to-Peer-Anwendungen wie BitTorrent)

MICROSERVICES!!!!

# Einführung

## ■ Dienste der Transportschicht

- im Internet gibt es zwei dominierende Transportprotokolle
  - **TCP**: verbindungsorientiert (abstrakte Sicht des Versendens eines Bytestroms), zuverlässig
  - **UDP**: verbindungslos (Versenden einzelner Datagramme), unzuverlässig
- werden meist im Betriebssystem realisiert
- die meisten Betriebssysteme bieten **Socket-Schnittstelle**, die durch Programmiersprachen als API angeboten wird
- mit Socket kann festgelegt werden
  - Transportprotokoll (TCP oder UDP)
  - IP-Adresse von Sende- und Zielhost
  - Portnummern (um Anwendungen auf Hosts zu unterscheiden)
- für Anwendungen bereitgestellte Dienste
- ggf. entsteht **QUIC** als Transportdienst auf Anwendungsschicht über UDP als Ersatz für TCP, s. nächstes Kapitel

# Einführung

## ■ Quantitative Anforderungen von Anwendungen

### ● Verlust

- nicht tolerierbar bei Dateitransfer, Online-Banking etc.
- teilweise tolerierbar bei Multimedia

### ● Bitrate

- traditionelle Anwendungen wie FTP, e-mail und HTTP benötigen keine feste Bitrate, sind aber „besser“, wenn sie viel Bitrate erhalten („Best-Effort-Verkehr“, „elastische Anwendungen“)
- Echtzeit-Multimedia benötigt Mindest-Bitrate

### ● Verzögerungszeit

- traditionelle Anwendungen benötigen keine maximale Verzögerungszeit, sind aber wieder „besser“ bei kurzen Zeiten
- Echtzeit-Multimedia und interaktive Spiele benötigen kurze Verzögerungszeit
- Steuerungen technischer Geräte benötigen oft Garantie einer maximalen Verzögerungszeit, insbesondere bei sicherheitskritischen Systemen

# Einführung

## ■ Quantitative Anforderungen von Anwendungen

Anwendung	Verlust	Bitrate	Verzögerungszeit
Dateitransfer	kein Verlust	elastisch	keine harte Grenze
E-Mail	kein Verlust	elastisch	keine harte Grenze
Web-Dokumente	kein Verlust	elastisch	keine harte Grenze
Echtzeit-Multimedia	Verlust tolerierbar	Audio: Kbps - Mbps Video: 10 Kbps – mehrere 100 Mbps	150 ms Einwegverzögerung unbemerkt
Streaming von Multimedia	Verlust tolerierbar	wie oben	einige s
Interaktive Spiele	Verlust tolerierbar	von Kbps an	einige 10 ms
Automatisierung	kein Verlust	Kbps	oft harte Grenzen, z.B. einige ms
Smartphone Messaging	kein Verlust	elastisch	keine harte Grenze

Quelle: Kurose, Ross.  
*Computer Networking:  
A Top-Down Approach*,  
7th Ed., Pearson  
Education, 2017.



# Einführung

## ■ Einige bekannte Anwendungsprotokolle und darunterliegende Transportprotokolle

Anwendung	Anwendungsprotokoll	verwendetes Transportprotokoll
E-Mail	SMTP [RFC 5321]	TCP
Remote Terminal Access	Telnet [RFC 854]	TCP
Web	HTTP [RFC 7320]	TCP
Dateitransfer	FTP [RFC 959]	TCP
Streaming Multimedia	HTTP, DASH variante von http	TCP
Internettelefonie	SIP [RFC 3261], RTP [RFC 3550], proprietär	UDP oder TCP

Quelle: Kurose, Ross.  
*Computer Networking:  
A Top-Down Approach*,  
7th Ed., Pearson  
Education, 2017.

# Anwendungsschicht

## ✓ Einführung

## ■ **Verbreitete Anwendungen**

- Hypertext Transfer Protocol (HTTP)
- File Transfer Protocol (FTP)
- E-Mail
- Netzwerkmanagement
- Domain Name System (DNS)
- Content Distribution Networks

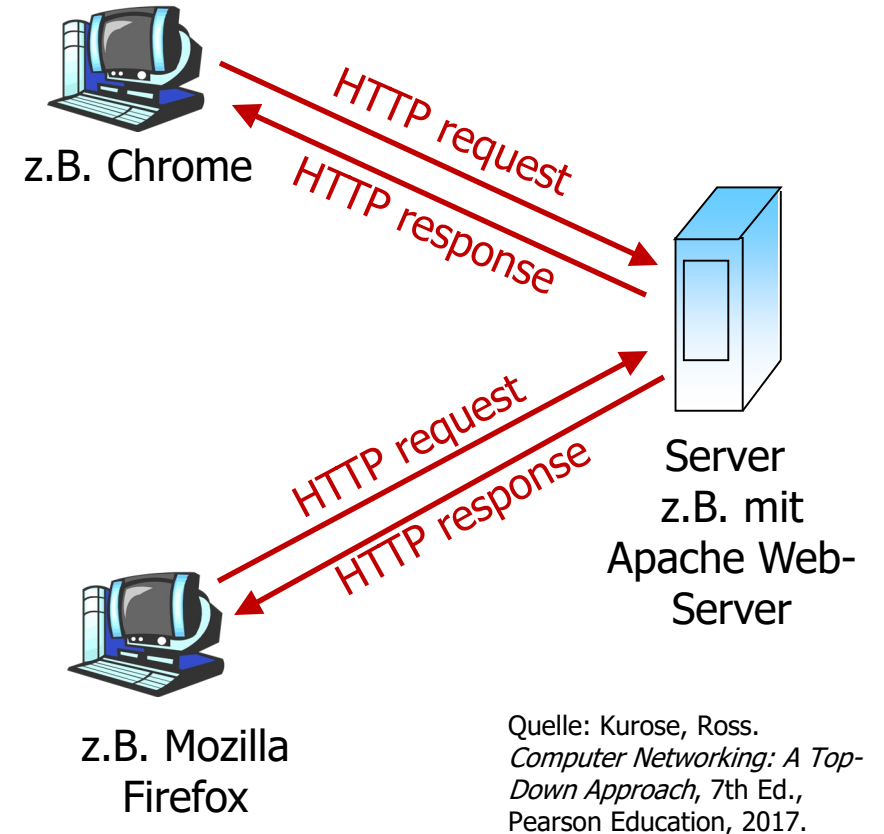
## ■ Socket-Programmierung

## ■ Peer-to-Peer-Systeme

# HTTP

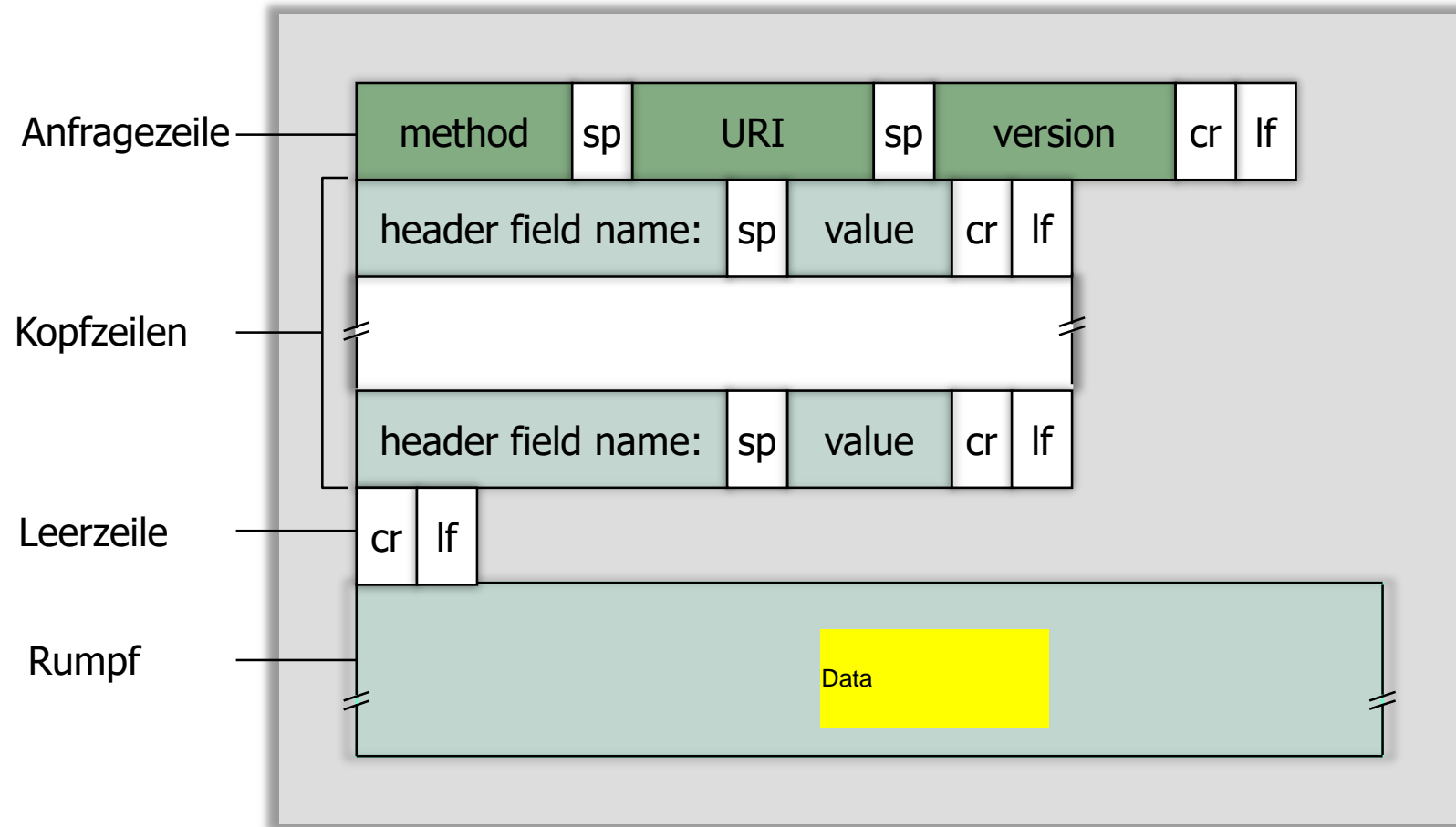
## ■ Ablauf

- Benutzer gibt **Uniform Resource Identifier (URI)** in Web-Browser ein
- URI enthält Host-Namen eines **Web-Servers** und den **Pfad zu einem Objekt** (Datei) dort
- Web-Browser stellt Anfrage an Web-Server für dieses Objekt
- Web-Server liefert Objekt an Web-Browser zurück
- Web-Browser stellt Objekt in für den Benutzer lesbarer Form dar



# HTTP: Anfragenachrichten

## ■ Format der Anfragenachrichten



Quelle: Kurose, Ross.  
*Computer Networking:  
A Top-Down Approach*,  
7th Ed., Pearson  
Education, 2017.

# HTTP: Anfragenachrichten

## ■ Methoden

- **GET**: Abruf eines Dokuments, besteht aus Methode, URI, Version
- **HEAD**: Abruf von Metainformationen eines Dokuments
- **POST**: Übergabe von Informationen an Server
- **PUT, DELETE**

## ■ Kopfzeilen

- Typ/Wert-Paare, Typen: Host, User-agent, ...

Key-value in normal people speak

## ■ Rumpf

- leer bei GET, kann bei POST Inhalt haben

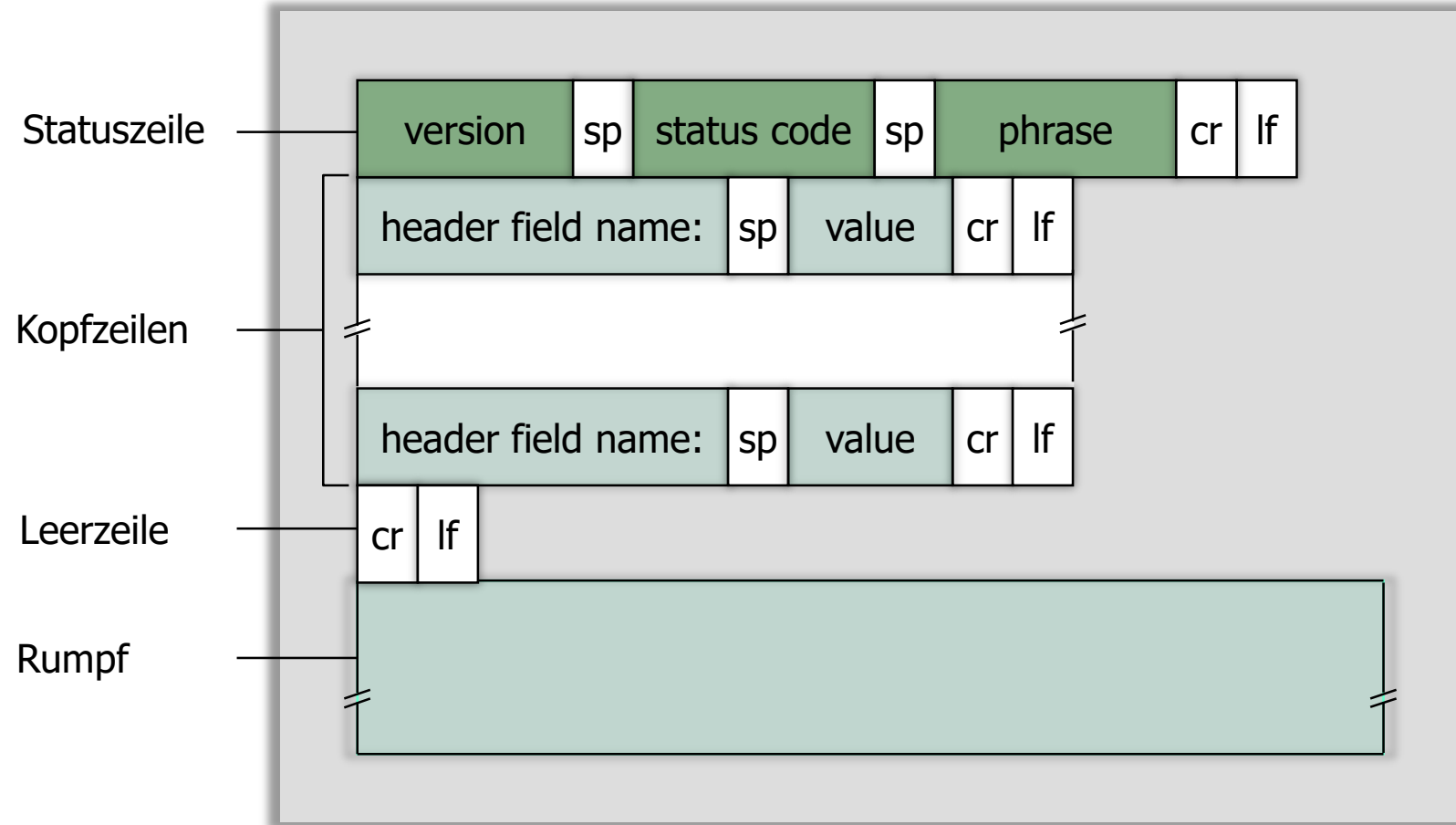
## ■ Beispiel Anfragenachricht:

```
GET /somedir/page.html HTTP/1.1
Host: www.someschool.edu
User-agent: Mozilla/4.0
Connection: close
Accept-language: de-de
```

Quelle: Kurose, Ross.  
*Computer Networking: A Top-Down Approach*, 7th Ed.,  
Pearson Education, 2017.

# HTTP: Antwortnachrichten

## ■ Format der Antwortnachrichten



Quelle: Kurose, Ross.  
*Computer Networking:  
A Top-Down Approach*,  
7th Ed., Pearson  
Education, 2017.

# HTTP: Antwortnachrichten

## ■ Mögliche Codes in der Statuszeile

- 200 OK (Anfrage erfolgreich, Objekt in Antwort)
- 301 Moved Permanently (Redirection: Objekt zu finden unter Location: ...)
- 400 Bad Request (Anfragenachricht nicht verstanden)
- 404 Not Found (Objekt nicht gefunden)
- 505 HTTP Version Not Supported

## ■ Beispiel- Antwortnachricht:

```
HTTP/1.1 200 OK
Connection: close
Date: Thu, 06 Aug 1998 12:00:15 GMT
Server: Apache/1.3.0 (Unix)
Last-Modified: Mon, 22 Jun 1998 ....
Content-Length: 6821
Content-Type: text/html

data data data data data ...
```

"connection closed" bei nicht persistentem HTTP

Quelle: Kurose, Ross.  
*Computer Networking: A Top-Down Approach*, 7th Ed.,  
Pearson Education, 2017.

# HTTP: Ablauf

## ■ HTTP-Ablauf

- nicht-persistentes HTTP

- für jedes Objekt wird einzelne TCP-Verbindung geöffnet, Server beendet sie sofort nach dem Senden eines Objekts
- entweder Basis-Seite und eingebettete Objekte sequentiell
- oder parallele einzelne Verbindungen für die eingebetteten Objekte

many connections: one per object/pipeline

- persistentes HTTP

- Server lässt Verbindung bestehen
- alle Objekte werden über eine TCP-Verbindung gesendet
- ohne Pipelining: nach jedem Objekt Anfrage für nächstes Objekt
- mit Pipelining: eine Anfrage für alle eingebetteten Objekte

Wie zugriff auf Ram: burst, vs single polling

- Was sind die Vor- und Nachteile?
- Standardport des Web-Servers: 80

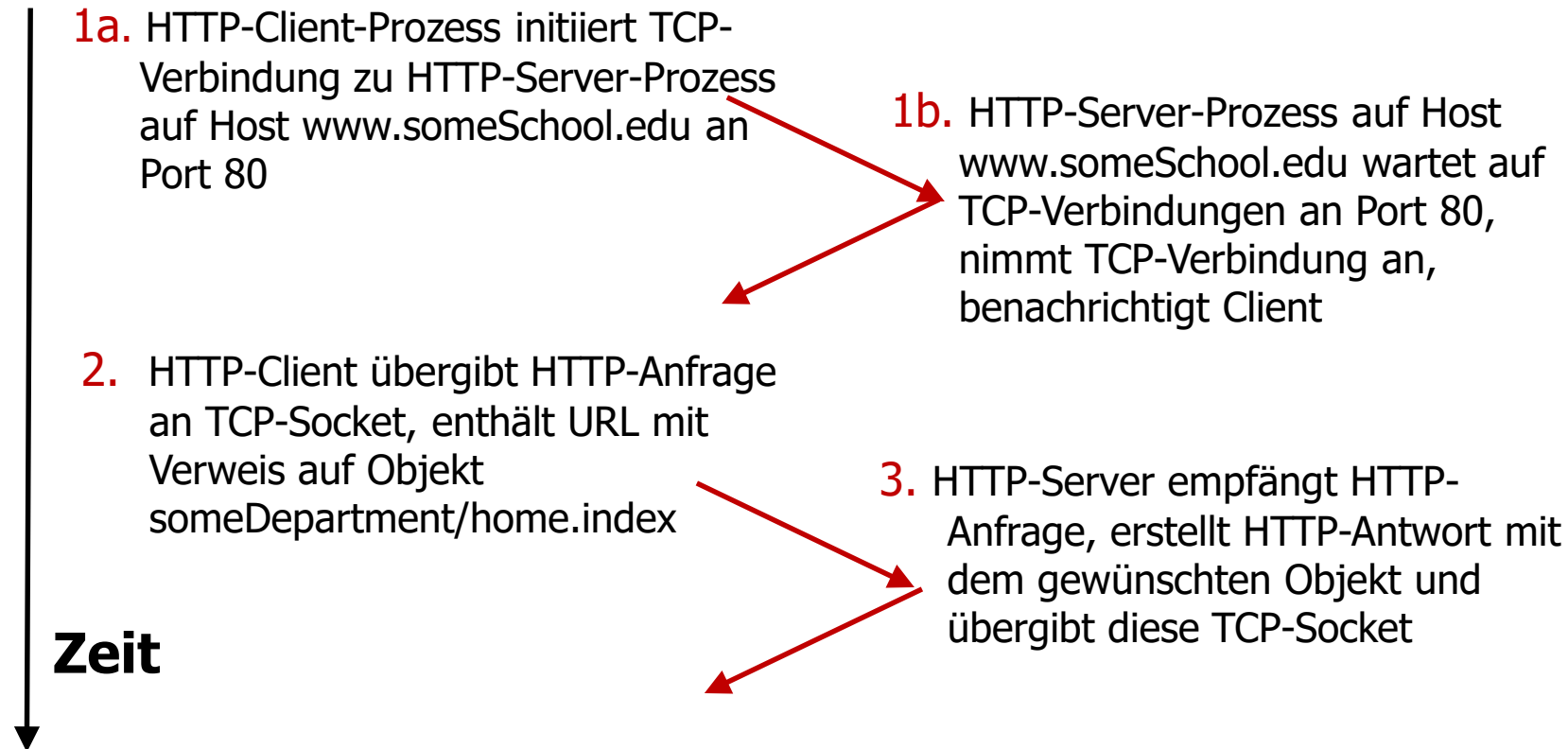


# HTTP: Ablauf

## ■ Beispiel-Ablauf von nicht-persistentem HTTP

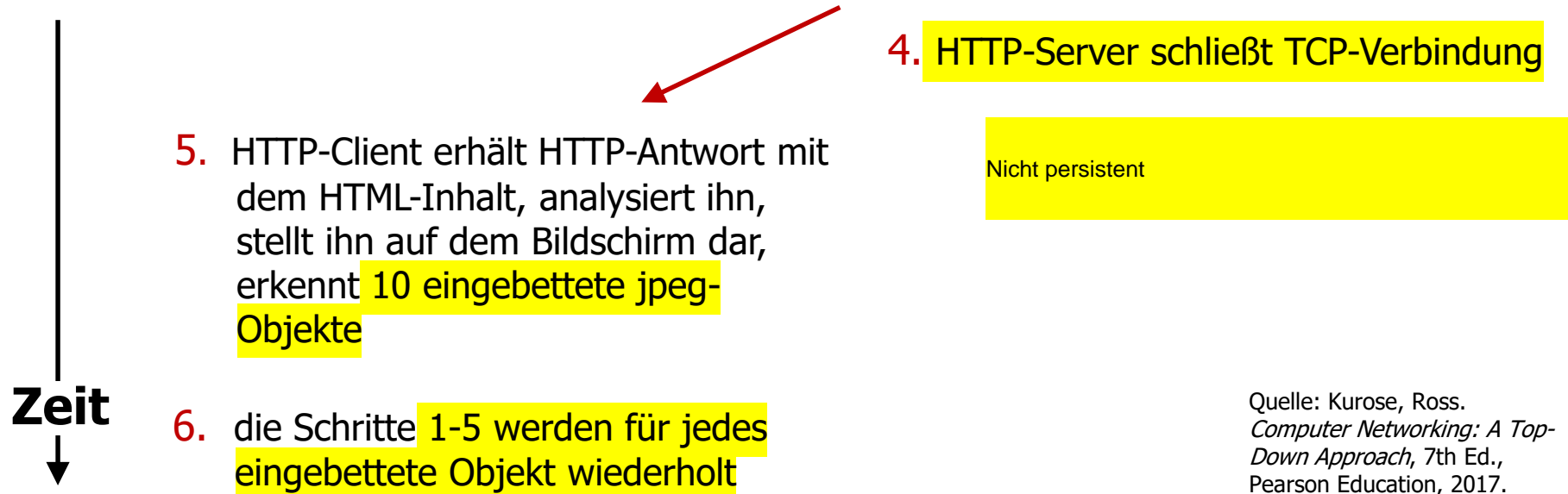
- URL: `www.someSchool.edu/someDepartment/home.index`
- Basis-Seite enthält 10 eingebettete Objekte (jpeg)

URL= spezial URI



Quelle: Kurose, Ross.  
*Computer Networking:  
A Top-Down Approach*,  
7th Ed., Pearson  
Education, 2017.

# HTTP: Ablauf



Quelle: Kurose, Ross.  
*Computer Networking: A Top-Down Approach*, 7th Ed.,  
Pearson Education, 2017.

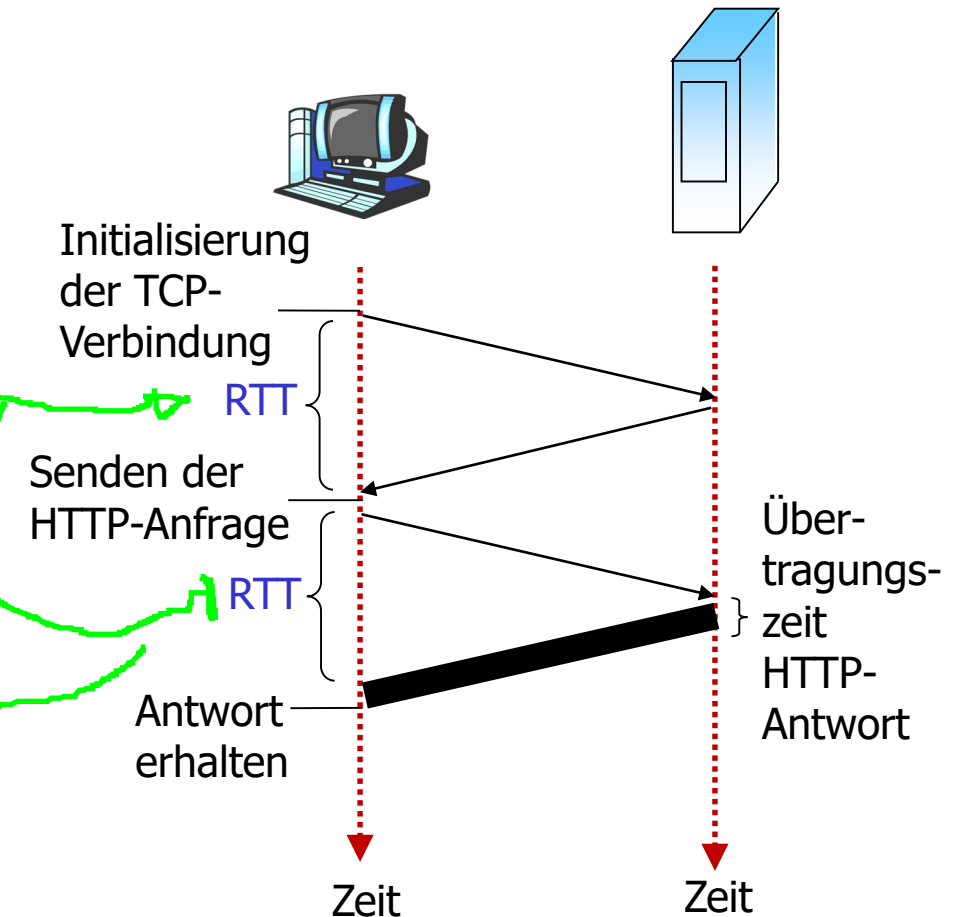
# HTTP: Ablauf

## ■ Antwortzeit

### ● Basis-Seite

- Aufbau der TCP-Verbindung erfordert eine **Round Trip Time (RTT)**
- Anfragenachricht hin, Antwortnachricht zurück, erfordert noch eine RTT
- insgesamt:
  - 2 RTT**
  - + Zeit zum Senden
  - + weitere Wartezeiten durch TCP

### ● wie ist es bei den anderen HTTP-Varianten?



Quelle: Kurose, Ross.  
*Computer Networking: A Top-Down Approach*, 7th Ed.,  
Pearson Education, 2017.

# HTTP: Dynamische Inhalte

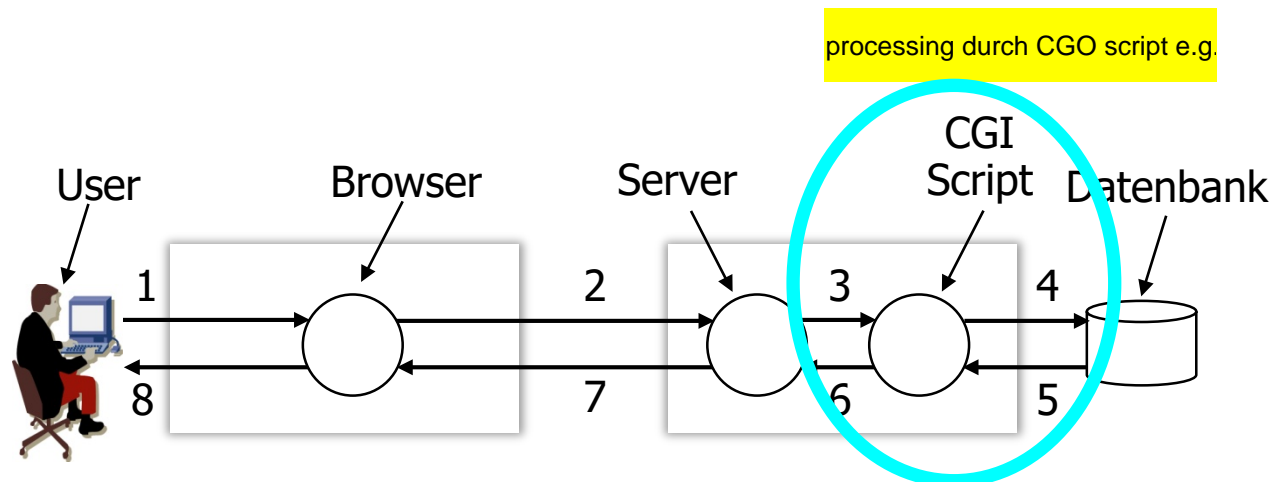
## ■ Senden von Information vom Browser zum Server

- in Rumpf von Anfragenachricht mit POST
- häufig: als Typ/Wert-Paare angehängt an die URL in einer Anfragenachricht mit GET

mit z.B. `http://.....com/ ?x="TEST12"`

## ■ Dynamische Inhalte mit CGI-Skripten

- **Common Gateway Interface (CGI)** verarbeitet als externer Prozess die Information und liefert neue HTML-Seite an Server



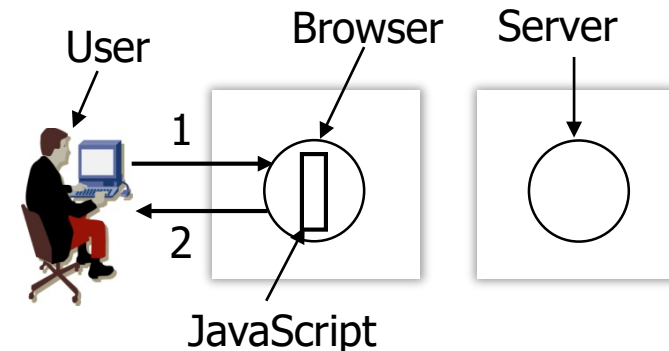
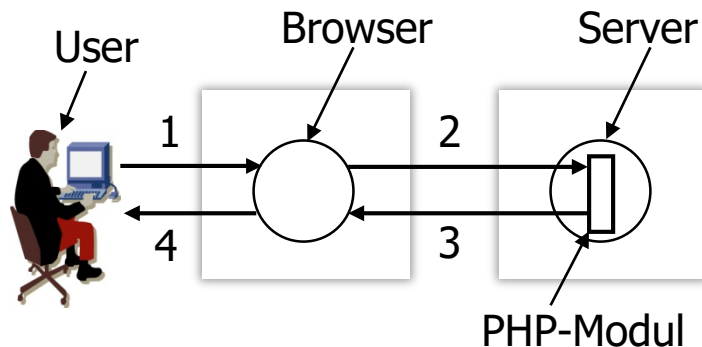
1. User füllt Formular aus
2. mit HTTP an Server
3. wird CGI übergeben
4. CGI fragt DB
5. DB-Eintrag gefunden
6. CGI erstellt HTML
7. mit HTTP an User
8. HTML darstellen

Quelle: Tanenbaum.  
Computer Networks. 5th  
Ed., Prentice Hall, 2011.

# HTTP: Dynamische Inhalte

## ■ Dynamische Inhalte durch Scripting

- durch Interpretation von eingebetteten Skripten können dynamische Inhalte erzeugt werden
- **Server-seitiges Scripting**: im HTML ist Code eingebettet, der vom **Server interpretiert wird und dabei HTML erzeugt**, z.B. **PHP**
- **Client-seitiges Scripting**: im HTML ist Code eingebettet, der vom **Client interpretiert wird**, z.B. **JavaScript**
- weitere Verbesserung durch **Ajax (Asynchrones JavaScript)**

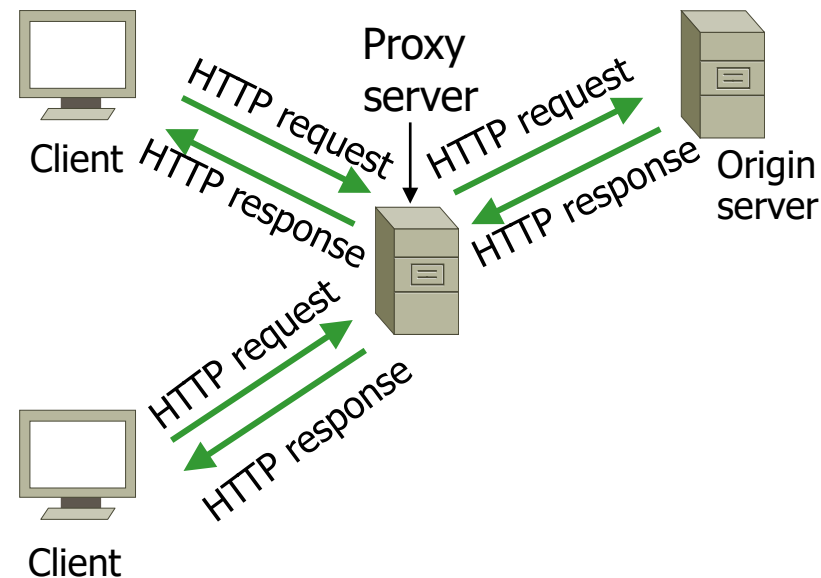


Quelle: Tanenbaum.  
Computer Networks. 5th  
Ed., Prentice Hall, 2011.

# HTTP: Caching

## ■ Web-Caching

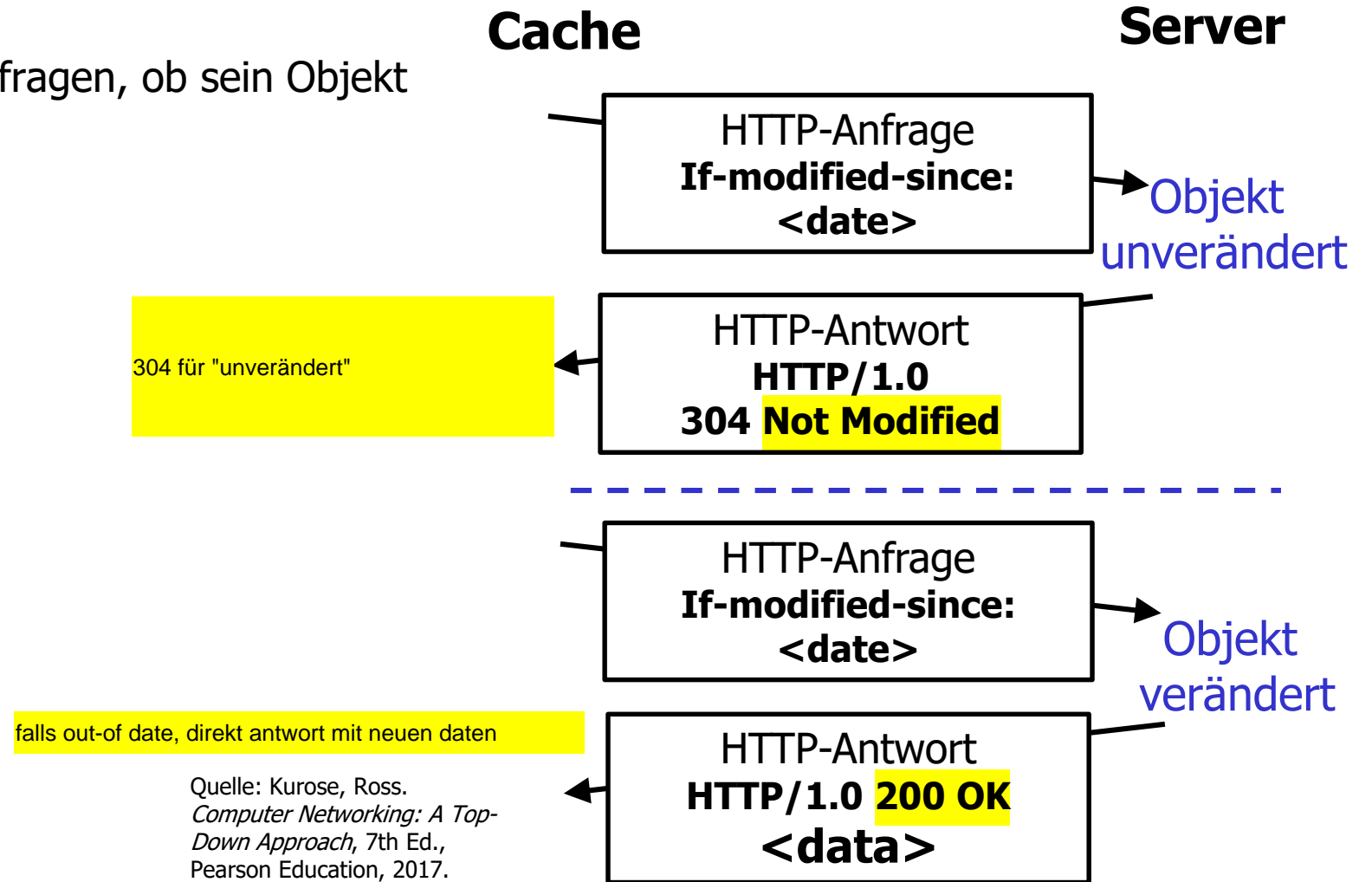
- Verringerung der Wartezeit des Benutzers und des Netzwerkverkehrs durch Zwischenspeicher
- Cache ist Server für Web-Browser und Client für Web-Server
- möglich an vielen Stellen: Browser, angeschlossenes LAN, ISP, ...



Quelle: Kurose, Ross.  
*Computer Networking: A Top-Down Approach*, 7th Ed.,  
Pearson Education, 2017.

# HTTP: Caching

- Cache kann bei Server erfragen, ob sein Objekt noch aktuell ist:



# HTTP: Caching

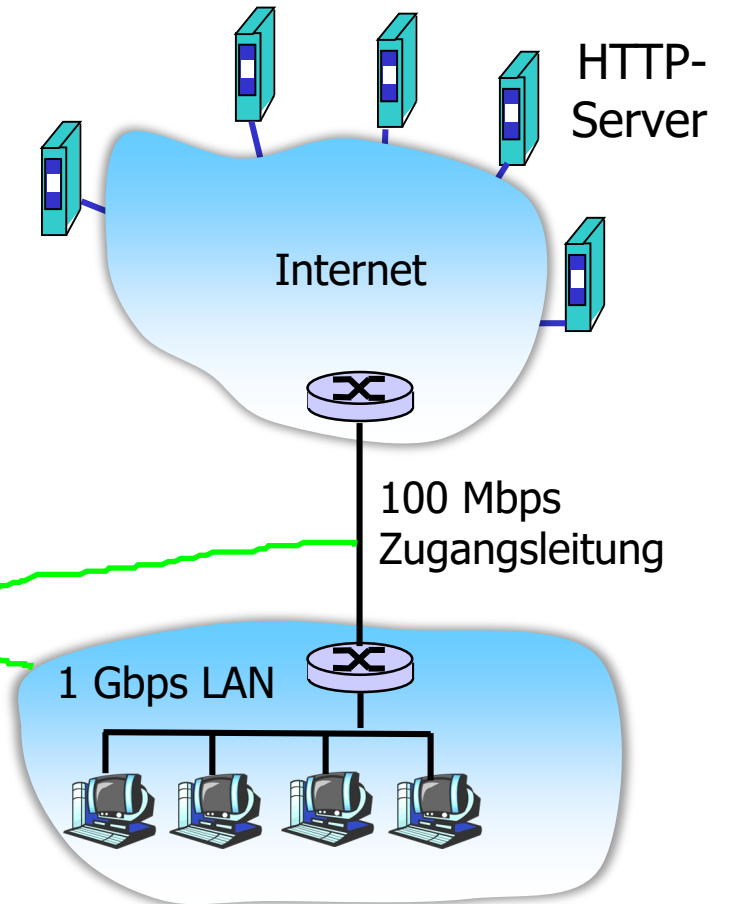
## ■ Beispiel für Nutzen eines Caches

### ● Annahmen

- mittlere Rate von HTTP-Anfragen der Clients im LAN = 8/s
- Objektgröße der HTTP-Anfrage wird vernachlässigt, mittlere Objektgröße der HTTP-Antwort = 1.500 KB = 12.288.000 bit,
- Latenz LAN, HTTP-Server und zurück = 2 s

### ● Folgen

- **Auslastung des LANs**  
 $8/s \cdot 12.288 \cdot 10^3 \text{ bit} / 10^9 \text{ bit/s}$   
 $\approx 0,098 \triangleq 10 \%$
- **Auslastung der Zugangsleitung**  
 $8/s \cdot 12.288 \cdot 10^3 \text{ bit} / 100 \cdot 10^6 \text{ bit/s}$   
 $\approx 0,98 \triangleq 98 \%$  schlecht: stochastisch überlastung wahrscheinlich
- **Gesamtverzögerung**  
Verzögerung im LAN + beim Zugang + im Internet  
 $\approx \text{ms} + \text{Minuten} + 2 \text{ s} \approx \text{Minuten}$



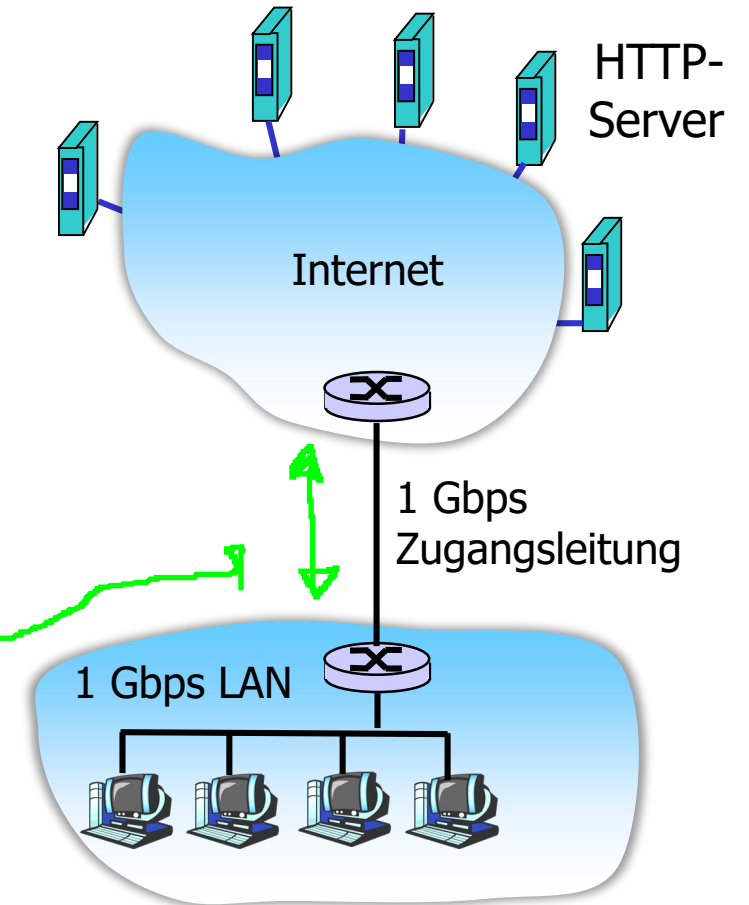
Quelle: Kurose, Ross.  
*Computer Networking: A Top-Down Approach*, 7th Ed.,  
Pearson Education, 2017.



# HTTP: Caching

## ■ 1. Lösung: Upgrade des Zugangs

- Zugangsleitung mit 1 Gbps
- möglich, aber mit Kosten verbunden
- **Folgen**
  - Auslastung des LANs = 10 %
  - Auslastung der Zugangsleitung mit gleicher Rechnung: 10 %
  - Gesamtverzögerung = Verzögerung im LAN + beim Zugang + im Internet  $\approx \text{ms} + \text{ms} + 2 \text{ s} \approx \text{Sekunden}$



Quelle: Kurose, Ross.  
*Computer Networking: A Top-Down Approach*, 7th Ed.,  
Pearson Education, 2017.

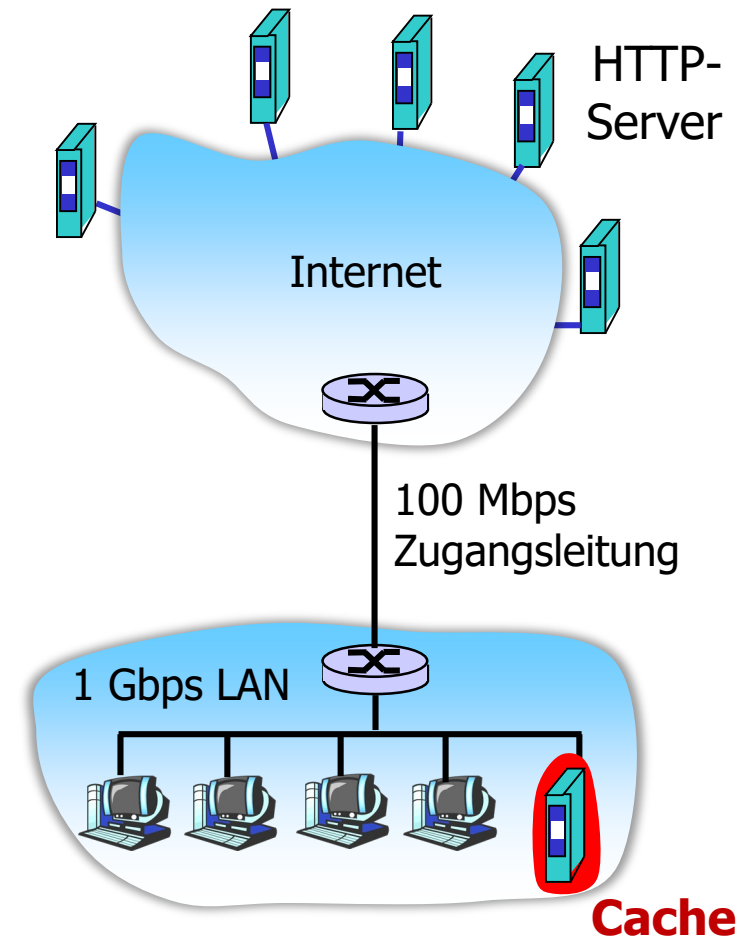
# HTTP: Caching

## ■ 2. Lösung: Verwendung eines Caches

- **Annahme:** Cache-Hitrate ist 0,4
- **realistisch:** 40 % der abgefragten Seiten befinden sich langfristig im Cache, 60% müssen bei HTTP-Servern angefordert werden

- **Folgen**

- **Auslastung des LANs**  $\triangleq 10 \%$
- **Auslastung der Zugangsleitung**  
 $0,6 \cdot 8/s \cdot 12.288 \cdot 10^3 \text{ bit} / 100 \cdot 10^6 \text{ bit/s} \approx 0,59 \triangleq 59 \%$
- **Gesamtverzögerung** = Verzögerung im LAN + beim Zugang + im Internet  $\approx \text{ms} + \text{ms} + 0,6 \cdot 2 \text{ s} < 2 \text{ s}$



Quelle: Kurose, Ross.  
*Computer Networking: A Top-Down Approach*, 7th Ed.,  
Pearson Education, 2017.

# HTTP/2

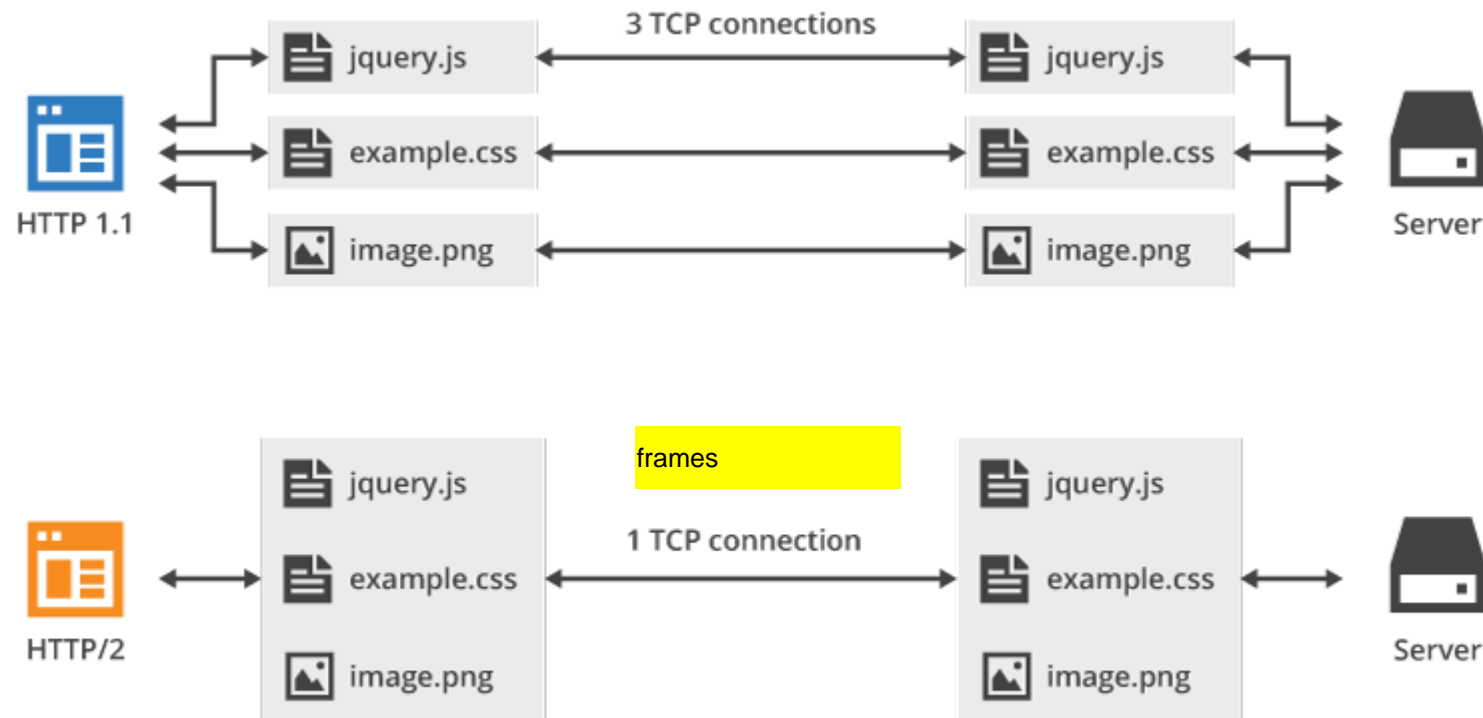
## ■ Weiterentwicklung von HTTP

- Version HTTP/1.1 aus dem Jahr 1999 [RFC2616]
- Latenzprobleme bei heutigen Web-Seiten
- proprietäre Weiterentwicklung: SPDY („speedy“) von Google
- HTTP/2: hierauf aufsetzend Standardisierung der IETF seit 2007, RFC 7540 im Mai 2015, Hypertext Transfer Protocol Version 2
- wesentliche Elemente
  - grundsätzlich gleiche Methoden
  - binäres statt textbasiertes Format
  - Multiplexing verschiedener Ströme über eine TCP-Verbindung, Vermeidung von Head-of-Line (HOL) Blockierung durch große Objekte durch Aufteilung in kleinere Frames und Interleaving
  - Header-Kompression
  - Server-Push
- Unterstützung durch die bekannten Browser seit Ende 2015 (mit TLS Verschlüsselung als „de facto“ Standard)
- HTTP/3: basierend auf QUIC, s. nächstes Kapitel

interleaving verhindert, dass ein video das laden von html/css vollkommen blockiert

# HTTP/2

## ■ Multiplexing



Quelle: <https://blog.cloudflare.com/http-2-for-web-developers/>

# Vergleich HTTP/1.1 (links) und HTTP/2 (rechts)


Google Chrome → Rechtsklick in Hauptfenster → Untersuchen → Network

https://http1.golang.org/gopher/ x +

http1.golang.org/gophertiles?latency=0

A grid of 180 tiled images is below. Compare:

[\[HTTP/2, 0s latency\]](#) [\[HTTP/1, 0s latency\]](#)  
[\[HTTP/2, 30ms latency\]](#) [\[HTTP/1, 30ms latency\]](#)  
[\[HTTP/2, 200ms latency\]](#) [\[HTTP/1, 200ms latency\]](#)  
[\[HTTP/2, 1s latency\]](#) [\[HTTP/1, 1s latency\]](#)



Times from connection start:  
DOM loaded: 349ms  
DOM complete (images loaded): 4516ms

[Back to Go HTTP/2 demo server](#)

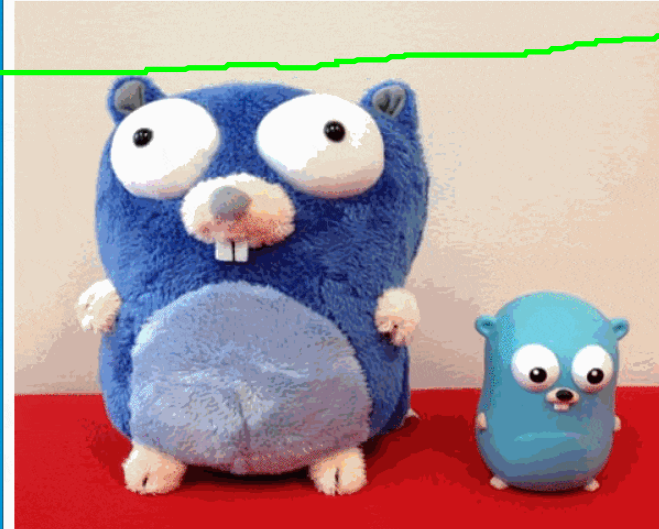
Network tab: 183 requests | 184 KB transferred | 161 KB resources | Finish

https://http2.golang.org/gopher/ x +

http2.golang.org/gophertiles?latency=0

A grid of 180 tiled images is below. Compare:

[\[HTTP/2, 0s latency\]](#) [\[HTTP/1, 0s latency\]](#)  
[\[HTTP/2, 30ms latency\]](#) [\[HTTP/1, 30ms latency\]](#)  
[\[HTTP/2, 200ms latency\]](#) [\[HTTP/1, 200ms latency\]](#)  
[\[HTTP/2, 1s latency\]](#) [\[HTTP/1, 1s latency\]](#)



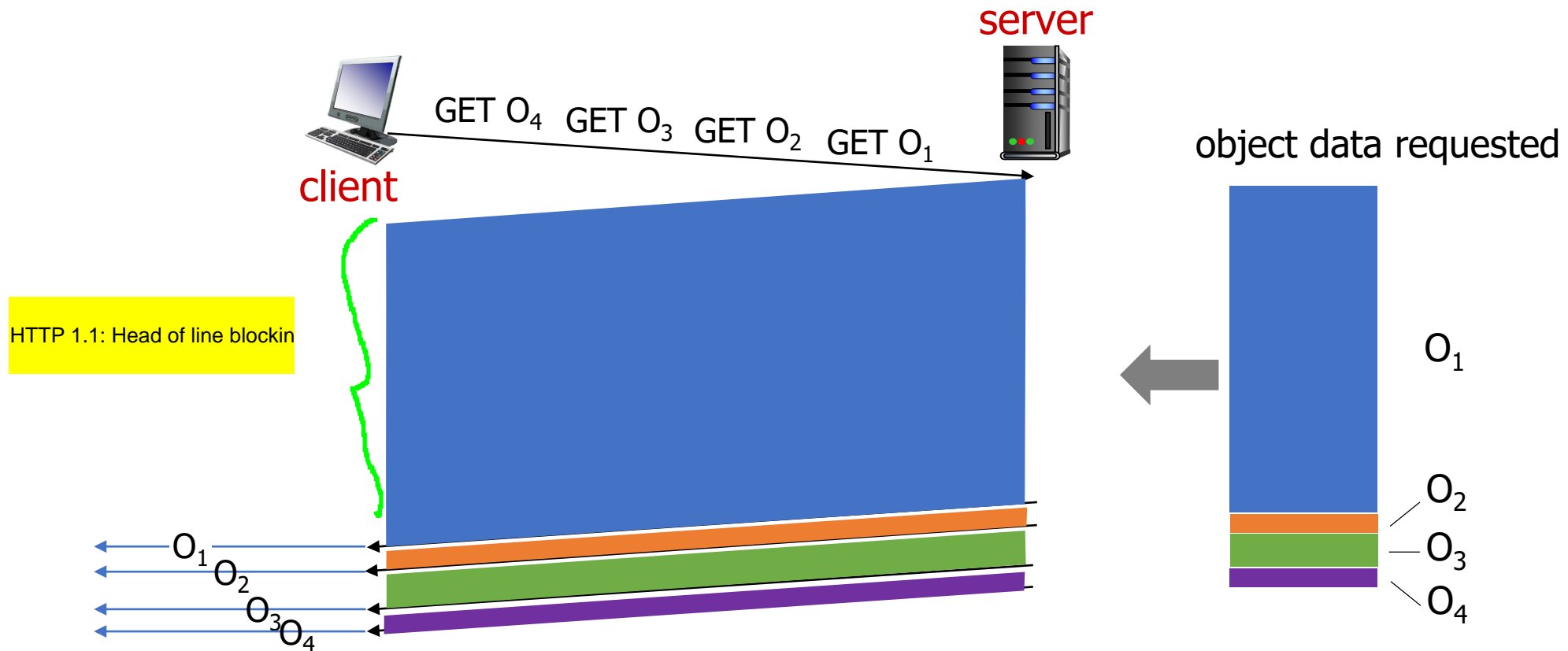
Times from connection start:  
DOM loaded: 376ms  
DOM complete (images loaded): 853ms

[Back to Go HTTP/2 demo server](#)

Network tab: 181 requests | 166 KB transferred | 161 KB resources | Finish

## HTTP/2

- HTTP 1.1: Pipelined Request für 1 großes (z.B. Video) und 3 kleine Objekte

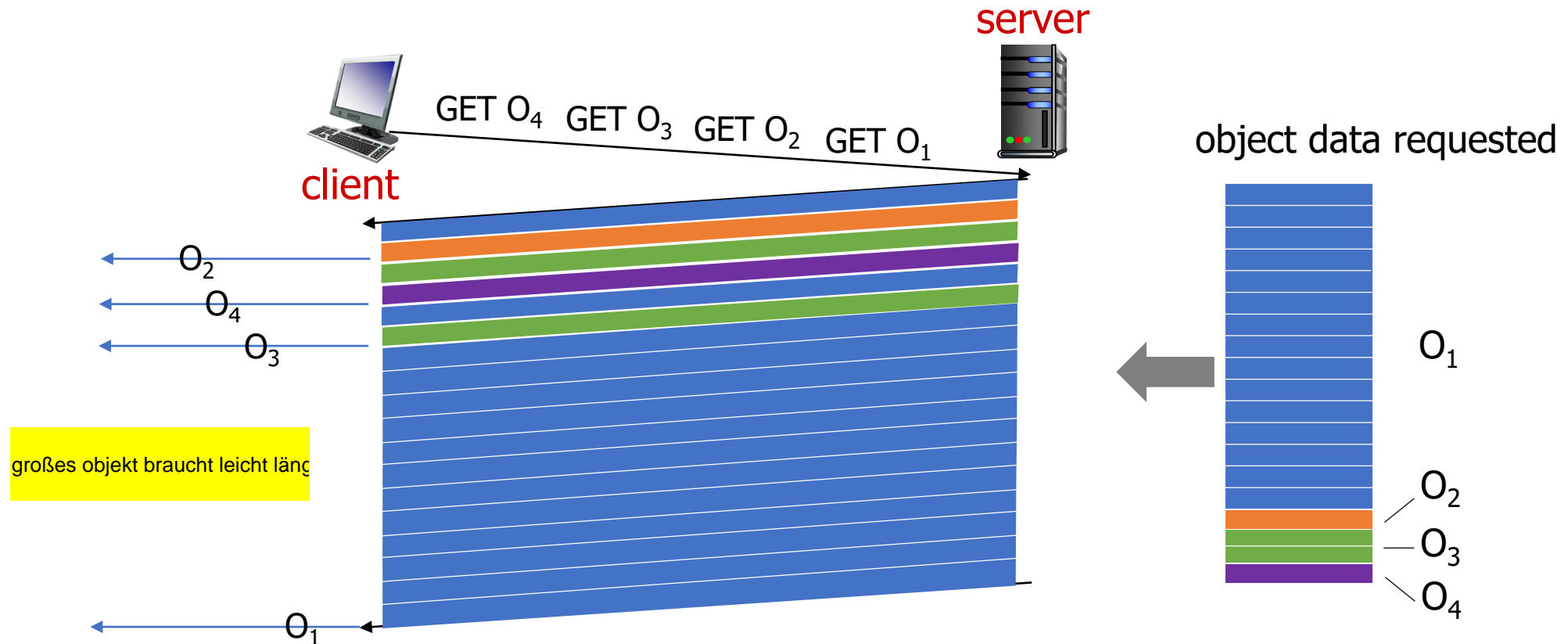


Quelle: Kurose, Ross.  
*Computer Networking:  
A Top-Down Approach*,  
7th Ed., Pearson  
Education, 2020.

objects delivered in order requested: O<sub>2</sub>, O<sub>3</sub>, O<sub>4</sub> wait behind O<sub>1</sub> (HOL Blocking)

# HTTP/2

## ■ HTTP/2: Zerlegung in Frames, Interleaving



Quelle: Kurose, Ross.  
*Computer Networking:  
A Top-Down Approach*,  
7th Ed., Pearson  
Education, 2020.

# Anwendungsschicht

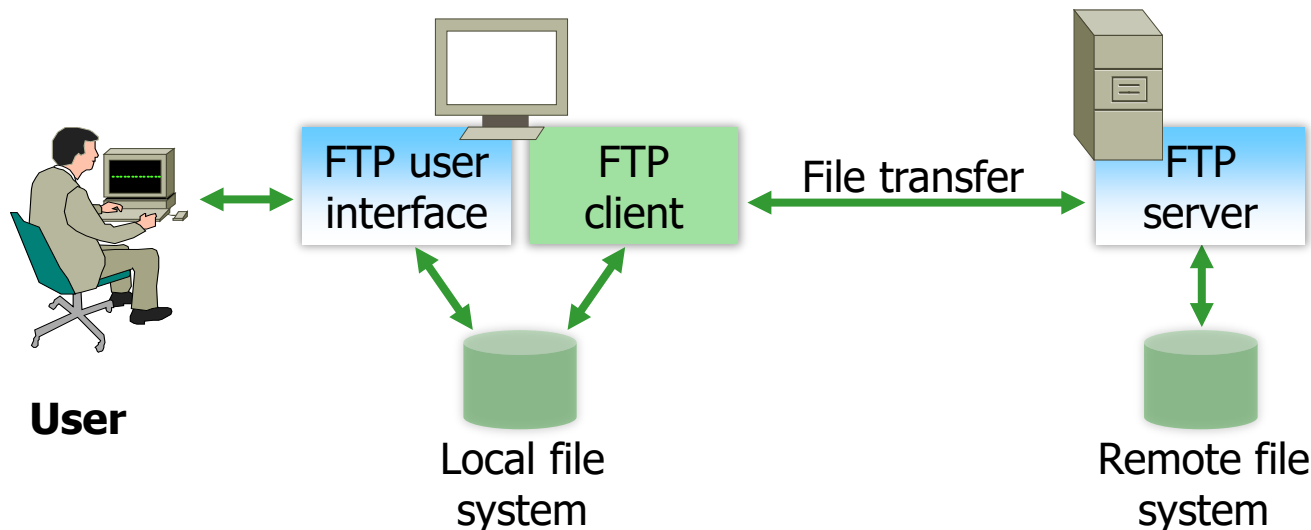
- ✓ Einführung
- ✓ Verbreitete Anwendungen
  - ✓ Hypertext Transfer Protocol (HTTP)
  - File Transfer Protocol (FTP)
  - E-Mail
  - Netzwerkmanagement
  - Domain Name System (DNS)
  - Content Distribution Networks
- Socket-Programmierung
- Peer-to-Peer-Systeme



# FTP

## ■ File Transfer Protocol [RFC 959, 1985]

- Übertragung von Dateien zwischen Hosts
- eine TCP-Verbindung (Port 21) zur Steuerung
- lesbare Kommandos: USER username, PASS password, LIST, RETR filename, STOR filename, ...
- jeweils eine TCP-Verbindung (Port 20) zur Übertragung einer Datei
- „out-of-band-control“
- mehr Einzelheiten in der Übung

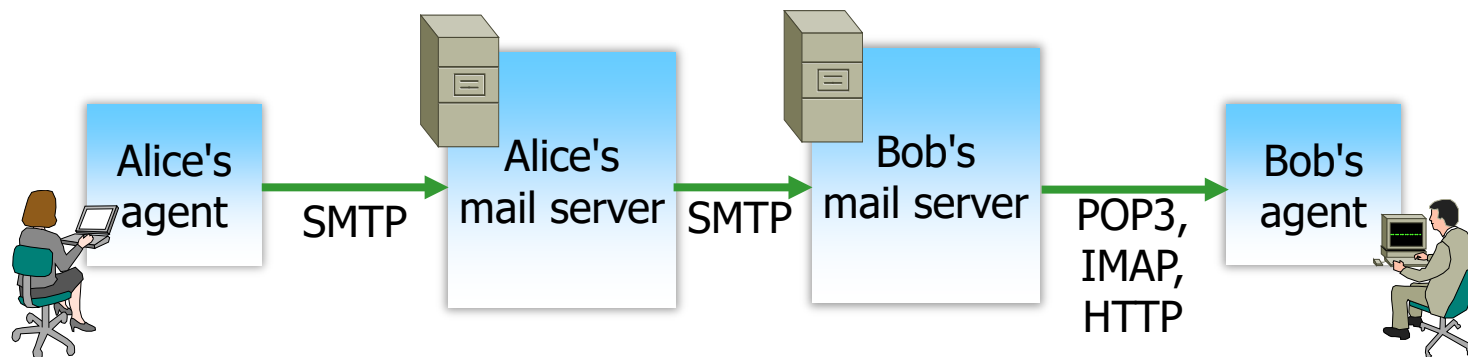


Quelle: Kurose, Ross.  
*Computer Networking: A Top-Down Approach*, 7th Ed.,  
Pearson Education, 2017.

# E-Mail

## ■ Simple Mail Transfer Protocol (SMTP)

- original RFC 821, 1982
- Nachrichten im ASCII-Format, Kopf, Rumpf
- andere Daten (Word-Dateien u.ä.) werden in ASCII umgewandelt angehängt: multimedia mail extension (MIME) das hat noch nie probleme gemacht...
- Versenden mit SMTP über TCP (lesbar)
- Abholen mit POP3, IMAP, HTTP (lesbar)
- mehr Einzelheiten in der Übung gmail



Quelle: Kurose, Ross.  
*Computer Networking: A Top-Down Approach*, 7th Ed.,  
Pearson Education, 2017.

## E-Mail: SMTP [RFC 5321]

- nutzt TCP zur zuverlässigen Übertragung der Nachrichten vom Client zum Server, dazu wird Port 25 verwendet
- direkte Übertragung: vom sendenden Server zu empfangendem Server
- drei Phasen der Übertragung
  - Handshaking (Begrüßung)
  - Nachrichtenübertragung
  - Abschlussphase
- Interaktion mittels Befehlen und Antworten
  - **Befehle:** ASCII-Text
  - **Antworten:** Statuscode und Text
- Nachrichten müssen 7-bit ASCII-Text sein

## Beispiel für einen SMTP-Dialog

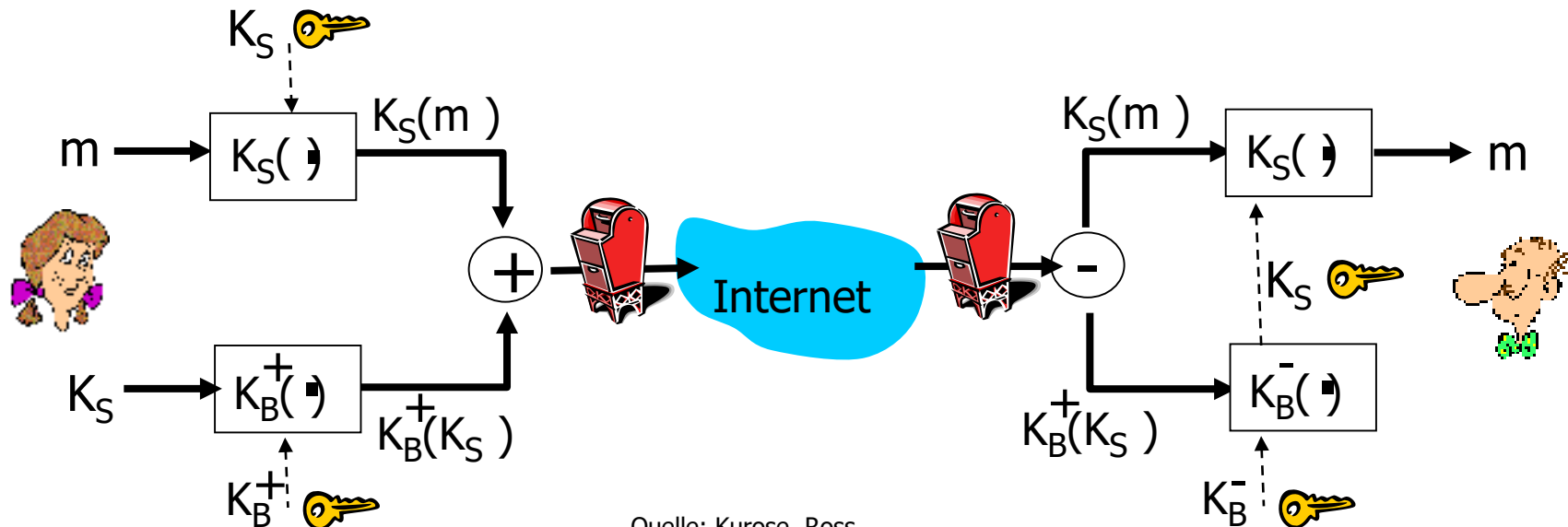
```
S: 220 hamburger.edu           # Server öffnet Session mit Domain-Namen
C: HELO crepes.fr               # Client antwortet mit eigenem Domain-Namen
S: 250 Hello crepes.fr, pleased to meet you # Server quittiert
C: MAIL FROM: <alice@crepes.fr>
S: 250 alice@crepes.fr... Sender ok
C: RCPT TO: <bob@hamburger.edu>
S: 250 bob@hamburger.edu ... Recipient ok
C: DATA
S: 354 Enter mail, end with "." on a line by itself
C: Do you like ketchup?
C: How about pickles?
C: .                             # Client beendet Mail-Text mit einem einzelnen Punkt
S: 250 Message accepted for delivery
C: QUIT
S: 221 hamburger.edu closing connection # Server schließt Sitzung
```

Quelle: Kurose, Ross.  
*Computer Networking:  
A Top-Down Approach*,  
7th Ed., Pearson  
Education, 2017.

# E-Mail

## ■ Sichere E-Mail: Vertraulichkeit

- Erzeugung eines symmetrischen Schlüssels  $K_S$
- Verschlüsselung der E-Mail mit  $K_S$
- asymmetrische Verschlüsselung von  $K_S$  mit dem öffentlichen Schlüssel  $K_B^+$  von Bob



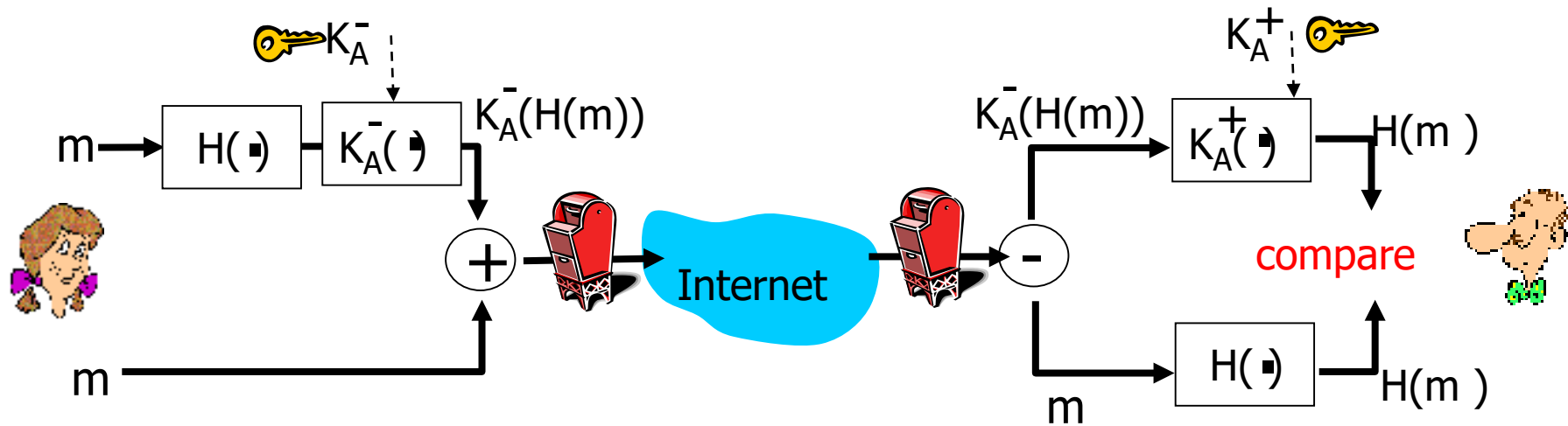
Quelle: Kurose, Ross.  
*Computer Networking: A Top-Down Approach*, 7th Ed.,

schneller mit synchronen, dann synchronen verschlüsseln für übertragung des synchronen schlüssels

# E-Mail

## ■ Sichere E-Mail: Datenintegrität

- Erzeugung eines Hashwerts der E-Mail
- Signierung mit dem privaten Schlüssel  $K_A^-$  von Alice
- keine Verschlüsselung der E-Mail



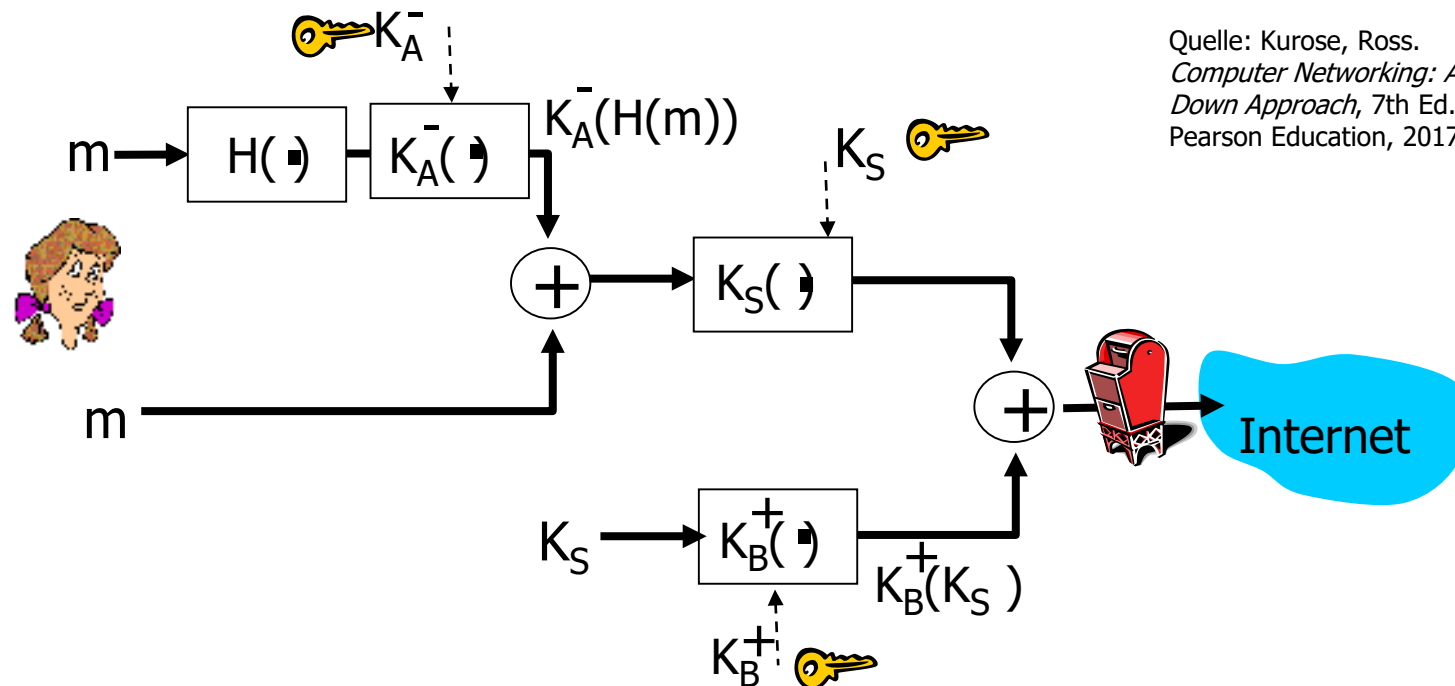
Quelle: Kurose, Ross.  
*Computer Networking: A Top-Down Approach*, 7th Ed.,  
Pearson Education, 2017.

# E-Mail

## ■ Sichere E-Mail: Vertraulichkeit und Datenintegrität

- Erzeugung eines Hashwerts der E-Mail
- Signierung mit dem privaten Schlüssel  $K_A^-$  von Alice
- Verschlüsselung der E-Mail und der Signatur mit  $K_S$
- asymmetrische Verschlüsselung von  $K_S$  mit dem öffentlichen Schlüssel  $K_B^+$  von Bob

also einfach die letzten 2 Folien gemeinsam



Quelle: Kurose, Ross.  
*Computer Networking: A Top-Down Approach*, 7th Ed.,  
Pearson Education, 2017.

# E-Mail

## ■ Sichere E-Mail: Standards

- S/MIME, RFC 2630, 2632, 2633

- Erweiterung des Multipurpose Internet Mail Extension (MIME) Standards, bei der eine E-Mail aus verschiedenen Teilen mit Kopffeldern bestehen kann
- Erweiterungen für die meisten Mail-Clients
- erfordert X.509-Zertifikate, mehrere CAs möglich

Zertifikation hierarchy

- Pretty Good Privay (PGP)

- bekannte Public Domain Software
- Web of Trust: Vertrauensnetz mit transitiven Vertrauensbeziehungen und Algorithmus zur Bestimmung der Vertrauenswürdigkeit (Trust-Level)

also, wie sicher ist die email nicht modifiziert, CA freie version

- Sicherheitsprobleme im operativen Betrieb seit 2018 bekannt („Efail“),  
Abhilfe durch S/MIME 4.0, RFC 8551

Bug in S/Mime gefixt in S/Mime 4.0



# Anwendungsschicht

- ✓ Einführung
- ✓ Verbreitete Anwendungen
  - ✓ Hypertext Transfer Protocol (HTTP)
  - ✓ File Transfer Protocol (FTP)
  - ✓ E-Mail
  - **Netzwerkmanagement**
  - Domain Name System (DNS)
  - Content Distribution Networks
- Socket-Programmierung
- Peer-to-Peer-Systeme

# Netzwerkmanagement

## ■ Aufgaben des Netzwerkmanagements

- Überwachung und Verwaltung eines Netzwerks = komplexes HW/SW-Gebilde (zahlreiche Geräte, Leitungen, Datenstrukturen, ...)
- FCAPS-Modell nach ISO
  - Fault Management/Fehlermanagement: Monitoring, Dokumentation, Reaktionsmaßnahmen
  - Configuration Management/Konfigurationsmanagement: Übersicht über Geräte und deren HW/SW-Konfigurationen
  - Accounting Management/Abrechnungsmanagement: Verwendung von Netzressourcen, Festlegung, Kontrolle, Dokumentation des Zugangs von Benutzern und Geräten
  - Performance Management/Leistungsmanagement: Monitoring von Auslastung, Durchsatz, Antwortzeiten, Dokumentation (z.B. für die Überwachung von Service Level Agreements), Reaktionsmaßnahmen
  - Security Management/Sicherheitsmanagement: Monitoring und Kontrolle des Zugangs, Schlüsselverwaltung, z.B. Filterregeln für Firewalls, Intrusion Detection
- z.B. Telecommunications-Management-Network (TMN), Standards der ITU-T, komplex

# Netzwerkmanagement

## ■ Simple Network Management Protocol (SNMP)

- SNMPv2 [RFC 1441, 1993], SNMPv3 [RFC 3410, 2002]
- einfach und verbreitet
- SNMP für Unternehmen mit großer Anzahl von Geräten wichtig
- z.B. Hunderte von Servern, Routern, Druckern ...
- ermöglicht Administrator proaktive Überwachung der Geräte im Netzwerk, um Probleme zu beheben
- SNMP wird von Management-SW verwendet
- z.B. nutzen Spiceworks, Getif, und LANView SNMP, um mit den Geräten im Netzwerk zu kommunizieren und Informationen von ihnen zu sammeln
- Geräte benötigen SW-Agenten, die gemäß SNMP kommunizieren
- durch SNMP ist es möglich, eine Vielzahl von Informationen von den Geräten in Echtzeit zu sammeln
- z.B.: Wie viel ist von einem Festplattenspeicher verwendet worden? Wie viele UDP-Pakete wurden empfangen?

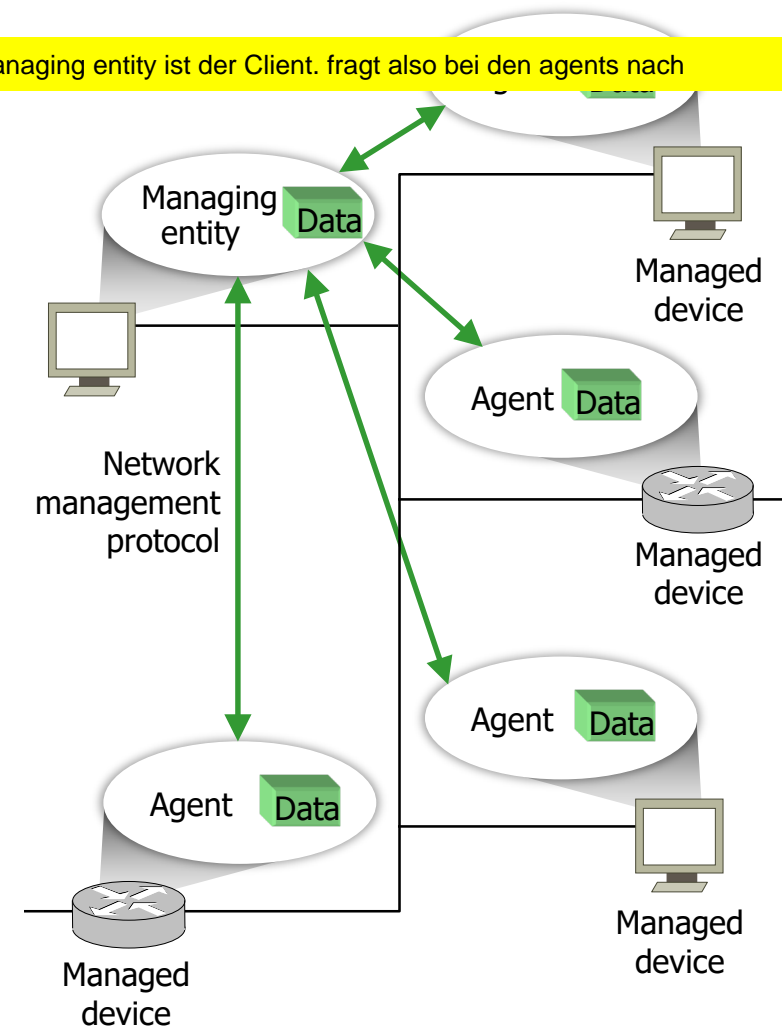
also parallel agent modell, a'la erlang

# Netzwerkmanagement

## ■ Organisationsmodell von SNMP

- **Managing Entity** bzw. Manager, Prozess auf zentraler Management Station,  $\triangleq$  Client
- **Managed Device**, Gerät im Netz
- **Managed Object**, HW oder SW im Managed Device, z.B. Routing-Tabelle
- **Management Agent**, Prozess auf Managed Device, kann lokale Aktionen ausführen,  $\triangleq$  Server
- Anfrage/Antwort-Protokoll zwischen Manager und Agent über UDP
- Manager entscheidet basierend auf Regeln, was zu tun ist (z.B. Warnmeldung auf Bildschirm, E-Mail senden)

Die managing entity ist der Client. fragt also bei den agents nach



Quelle: Kurose, Ross.  
*Computer Networking: A Top-Down Approach*, 7th Ed.,  
Pearson Education, 2017.

# Netzwerkmanagement

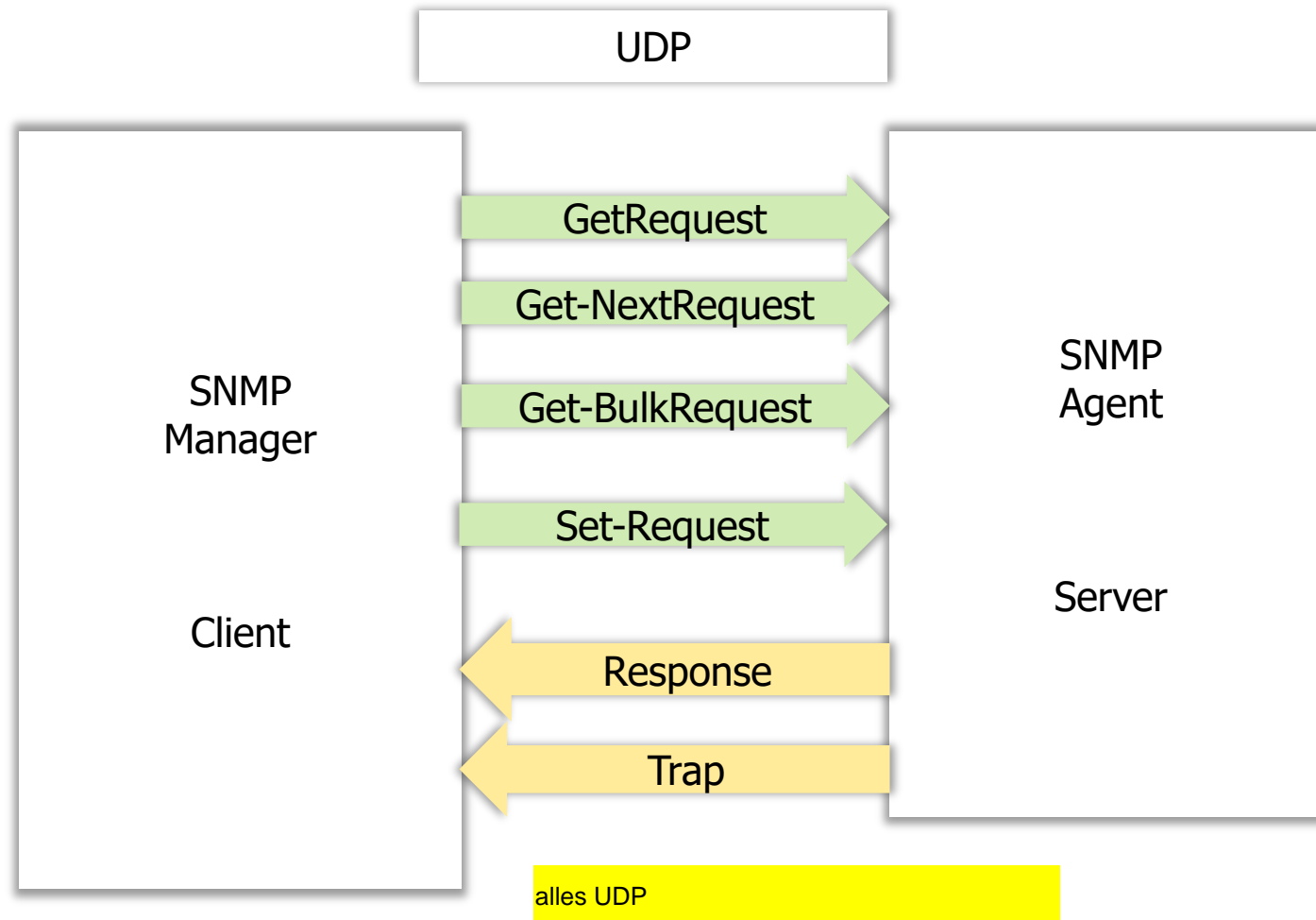
## ■ SNMP Messages

- Get-Request
  - Manager an Agent, um Daten von Agent zu erhalten
- Get-Next-Request
  - Manager an Agent, für nächsten Datensatz, Zugriff auf sequentielle Datensätze
- Get-Bulk-Request
  - Manager an Agent, für mehrere Datensätze auf einmal
- Set-Request
  - Manager an Agent, initialisiert oder ändert den Wert eines Datensatzes
- Response
  - Agent an Manager, Antwort auf Get und Set-Nachrichten
- Trap
  - Agent an Manager, unaufgeforderte Nachricht über Fehlersituation
  - z.B. 80% der Festplatte belegt

Fault, oder ressource probleme (entspricht eher dem interrupt aus

# Netzwerkmanagement

## ■ SNMP Messages zwischen dem Manager und dem Agenten

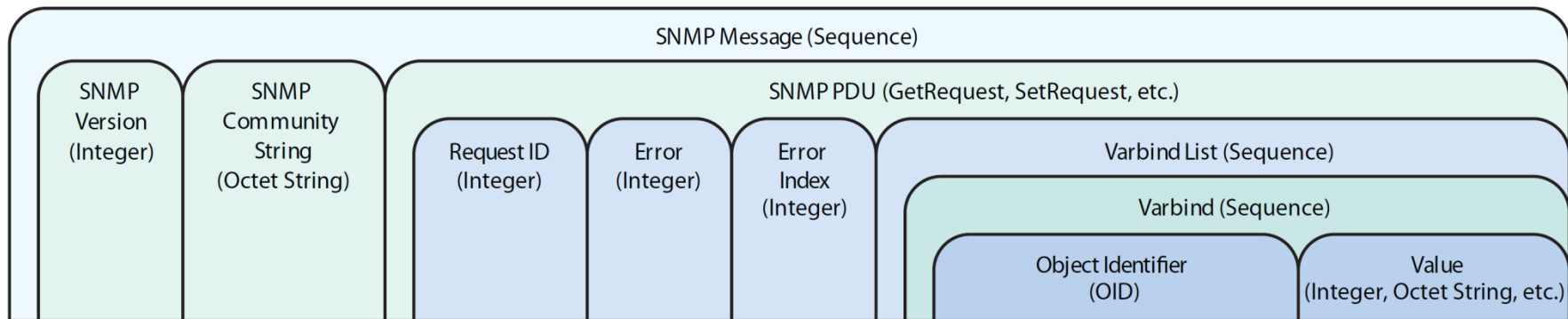


# Netzwerkmanagement

## ■ Format von SNMP Nachrichten

- z.B. Get- und Set-Requests:

Rein binär



Quelle: RaneNote: "SNMP: Simple? Network Management Protocol"

key-value

# Netzwerkmanagement

## ■ Management Information Base (MIB)

- MIB-Module enthalten Datenstrukturen für die Managed Objects, von der IETF genormt
- Syntax wird in Structure of Management Information (SMI) der IETF festgelegt, die wiederum die Abstract Syntax Notation One (ASN.1) der ISO benutzt ähnlich einer programmiersprache
  - Basisdatentypen: INTEGER, OCTET STRING, OBJECT IDENTIFIER, COUNTER64, ...
  - Sequenzen: SEQUENCE OF (andere MIB Objekte)
- ASN.1 besitzt auch ein Numerierungsschema zur eindeutigen Objekt-Identifizierung (OID), damit wird jedes MIB-Modul eindeutig bezeichnet
- mit den Basic Encoding Rules (BER) wird noch das genaue binäre Format für die Übertragung festgelegt

MSB first oder LSB first



# Netzwerkmanagement

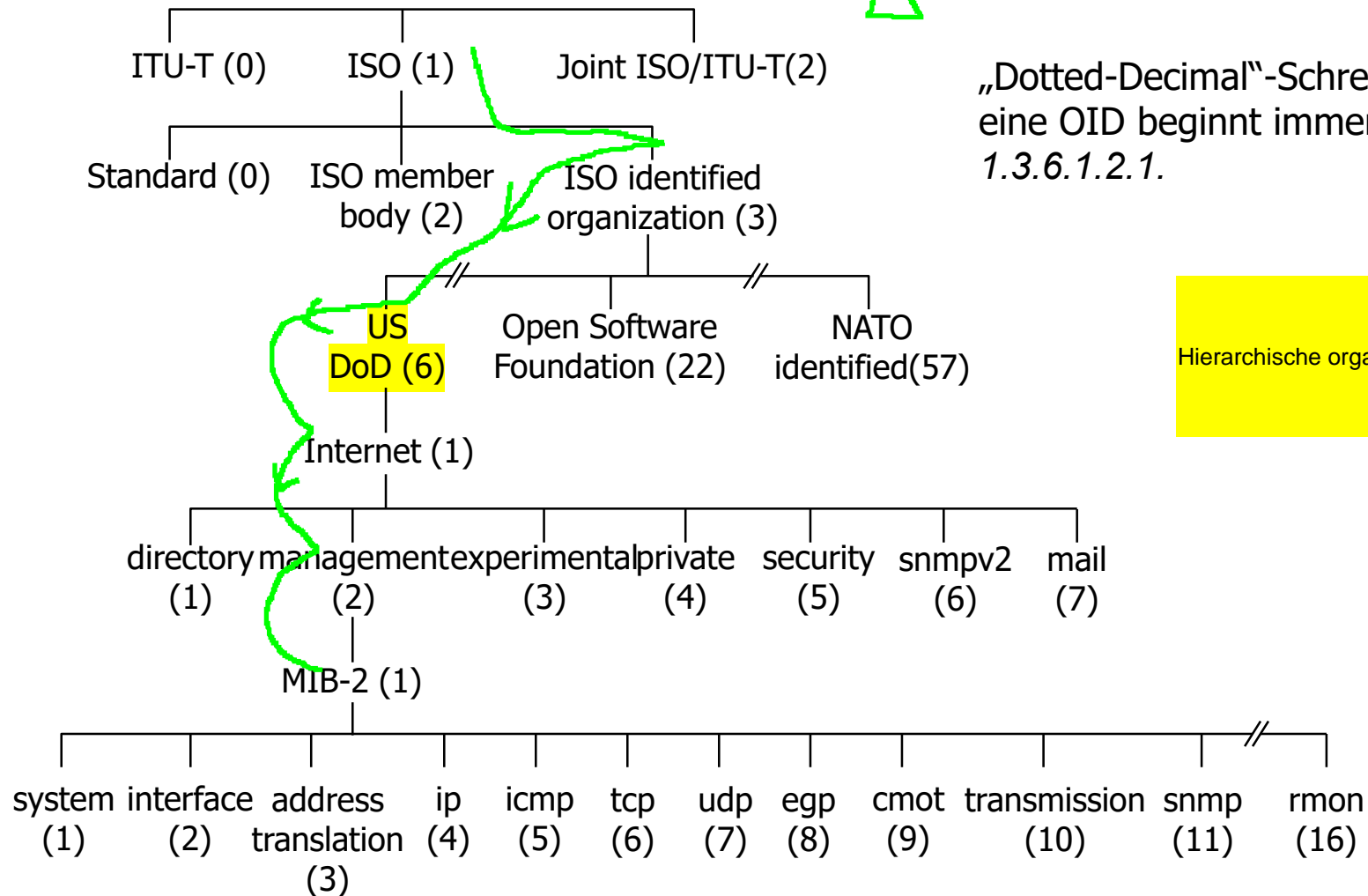
## ■ Bsp. für ein MIB-Modul:

```
UDP-MIB DEFINITIONS ::= BEGIN
...
udp      OBJECT IDENTIFIER ::= { mib-2 7 }
...
udpInDatagrams OBJECT-TYPE
    SYNTAX      Counter32
    MAX-ACCESS   read-only
    STATUS       current
    DESCRIPTION
        "The total number of UDP datagrams delivered to UDP users."
    ::= { udp 1 }
...
udpGroup OBJECT-GROUP
    OBJECTS      { udpInDatagrams, udpNoPorts,
                  udpInErrors, udpOutDatagrams,
                  udpLocalAddress, udpLocalPort }
    STATUS       current
    DESCRIPTION
        "The udp group of objects providing for management of UDP
        entities."
    ::= { udpMIBGroups 1 }
END
```

also UDP (vgl nächste Seite Baum)

# Netzwerkmanagement

## ■ Objekt-Identifizierung (OID)



„Dotted-Decimal“-Schreibweise,  
eine OID beginnt immer mit  
1.3.6.1.2.1.

Hierarchische organisation von typen

Quelle: Kurose, Ross.  
*Computer Networking: A Top-Down Approach*, 7th Ed.,  
Pearson Education, 2017.

# Netzwerkmanagement

## ■ MIB-Modul für UDP

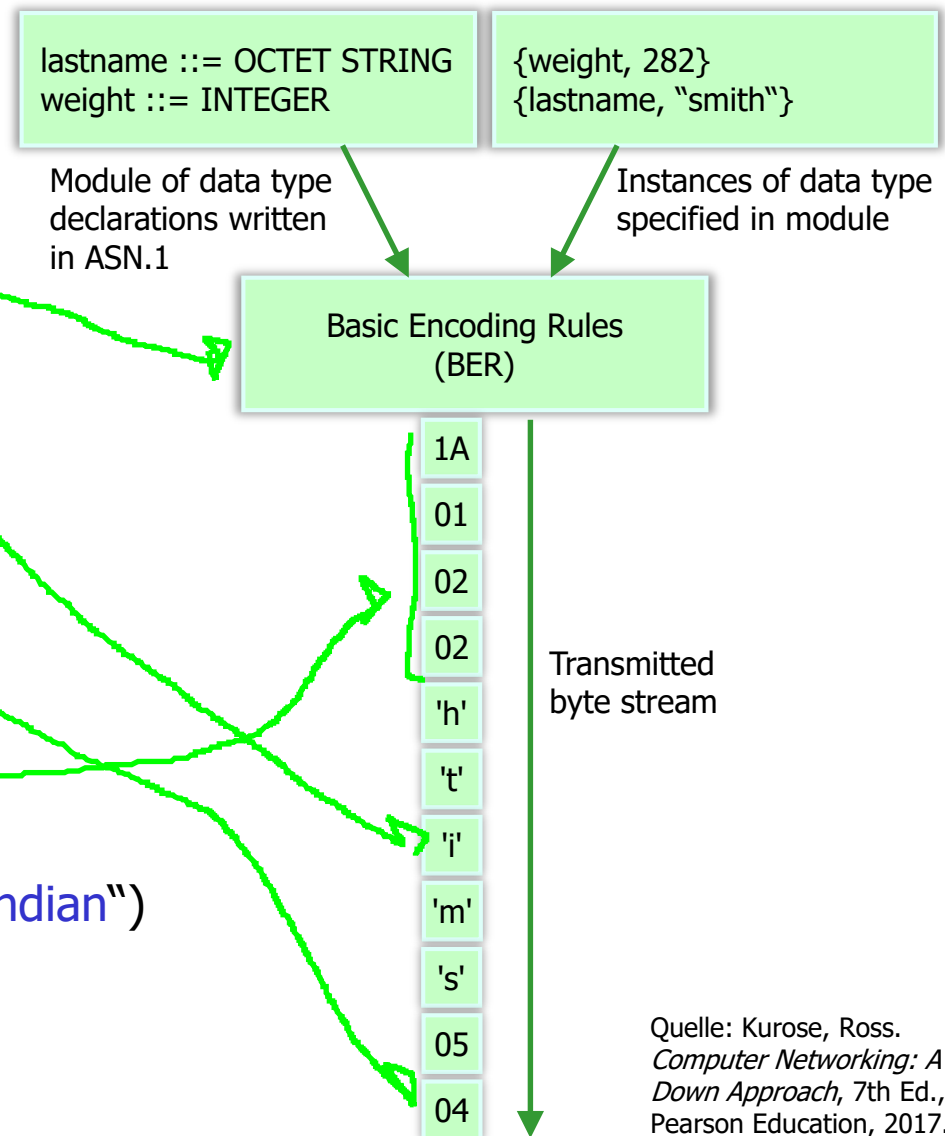
Object Identifier	Name	Type	Description (from RFC 2013)
1.3.6.1.2.1.7.1	udpInDatagrams	Counter32	“total number of UDP datagrams delivered to UDP users”
1.3.6.1.2.1.7.2	udpNoPorts	Counter32	“total number of received UDP datagrams for which there was no application at the destination port”
1.3.6.1.2.1.7.3	udpInErrors	Counter32	“number of received UDP datagrams that could not be delivered for reasons other than the lack of an application at the destination port”
1.3.6.1.2.1.7.4	udpOutDatagrams	Counter32	“total number of UDP datagrams sent from this entity”
1.3.6.1.2.1.7.5	udpTable	SEQUENCE of UdpEntry	“a sequence of UdpEntry objects, one for each port that is currently open by an application, giving the IP address and the port number used by application”

Quelle: Kurose, Ross.  
*Computer Networking:  
A Top-Down Approach*,  
7th Ed., Pearson  
Education, 2017.

# Netzwerkmanagement

## ■ Basic Encoding Rules (BER)

- Repräsentation zur Übertragung
- Tag, Length, Value (TLV)
  - Tag = Nummer für Typ
  - Length = Länge in Bytes
- Übertragung von „smith“
  - Tag 4 für OCTET STRING
  - Length 5
  - ASCII-Werte der Zeichen
- Übertragung von 282
  - Tag 2 für INTEGER
  - Length 2
  - 011A<sub>16</sub>, höherwertiges Byte zuerst („Big Endian“)



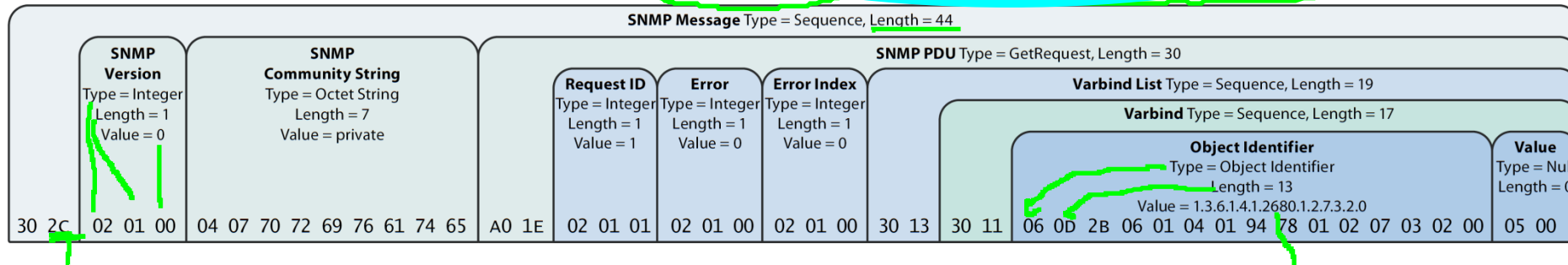
Quelle: Kurose, Ross.  
*Computer Networking: A Top-Down Approach*, 7th Ed.,  
Pearson Education, 2017.

# Netzwerkmanagement

## ■ Format von SNMP Nachrichten

- Get-Request mit OID und BER:

ALLES TYP TAG|LENGTH|VALUE



=44<sub>10</sub>

Quelle: RaneNote: "SNMP: Simple? Network Management Protocol"

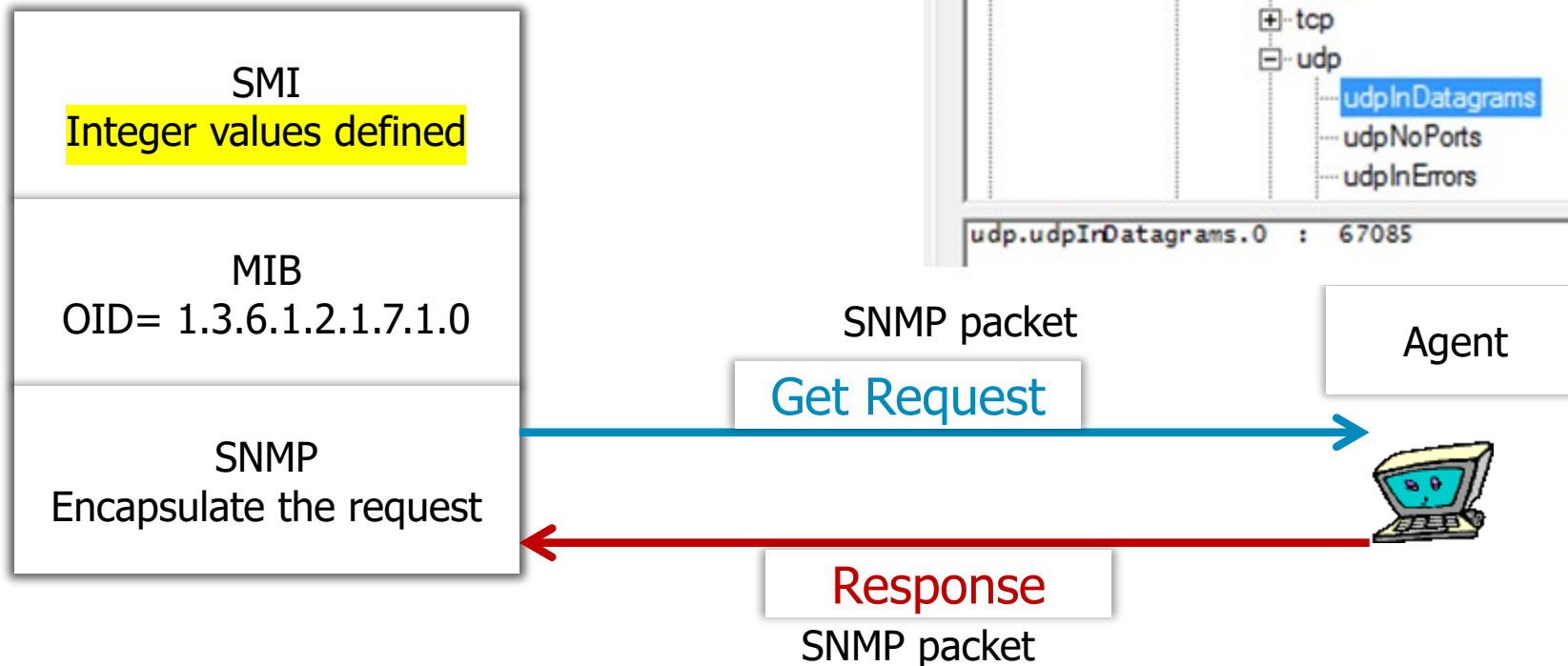
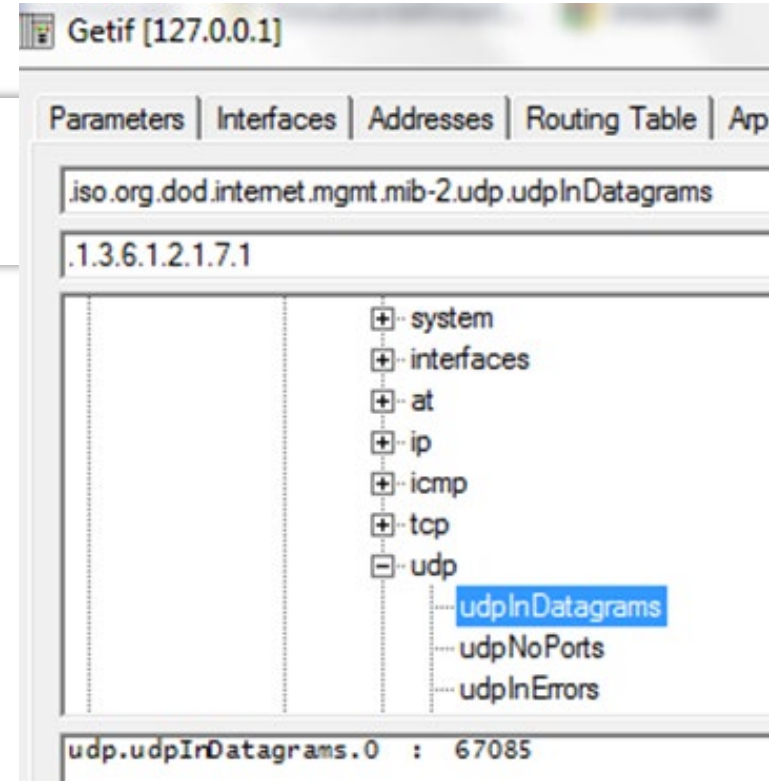
- zum Verständnis, besondere Kodierungsregeln für OIDs:

- TLV-Tripel beginnt mit Tag-Wert 06<sub>16</sub>
- die beiden ersten Werte werden in ein Byte kodiert, indem der erste Wert mit 40 multipliziert wird und der zweite addiert wird,  
z.B. für 1.3. ergibt sich  $1_{10} * 40_{10} + 3_{10} = 43_{10} = 2B_{16}$
- Short Form: Werte ≤ 127 werden in ein Byte kodiert, Bit 7 wird auf Null gesetzt
- Long Form: Werte > 127 werden in mehrere Bytes kodiert: nur Bits 0 bis 6 tragen kodierte Werte, Bit 7 aller Bytes bis auf das letzte wird auf Eins gesetzt,  
z.B.:  $9478_{16} = 10010100\ 01111000_2$ , Bit 7 in jedem Byte eliminieren, Bits 0 bis 6 führen zu  $00001010\ 01111000_2 = 10_{10} * 256_{10} + 7_{10} * 16_{10} + 8_{10} * 1_{10} = 2680_{10}$

# Netzwerkmanagement

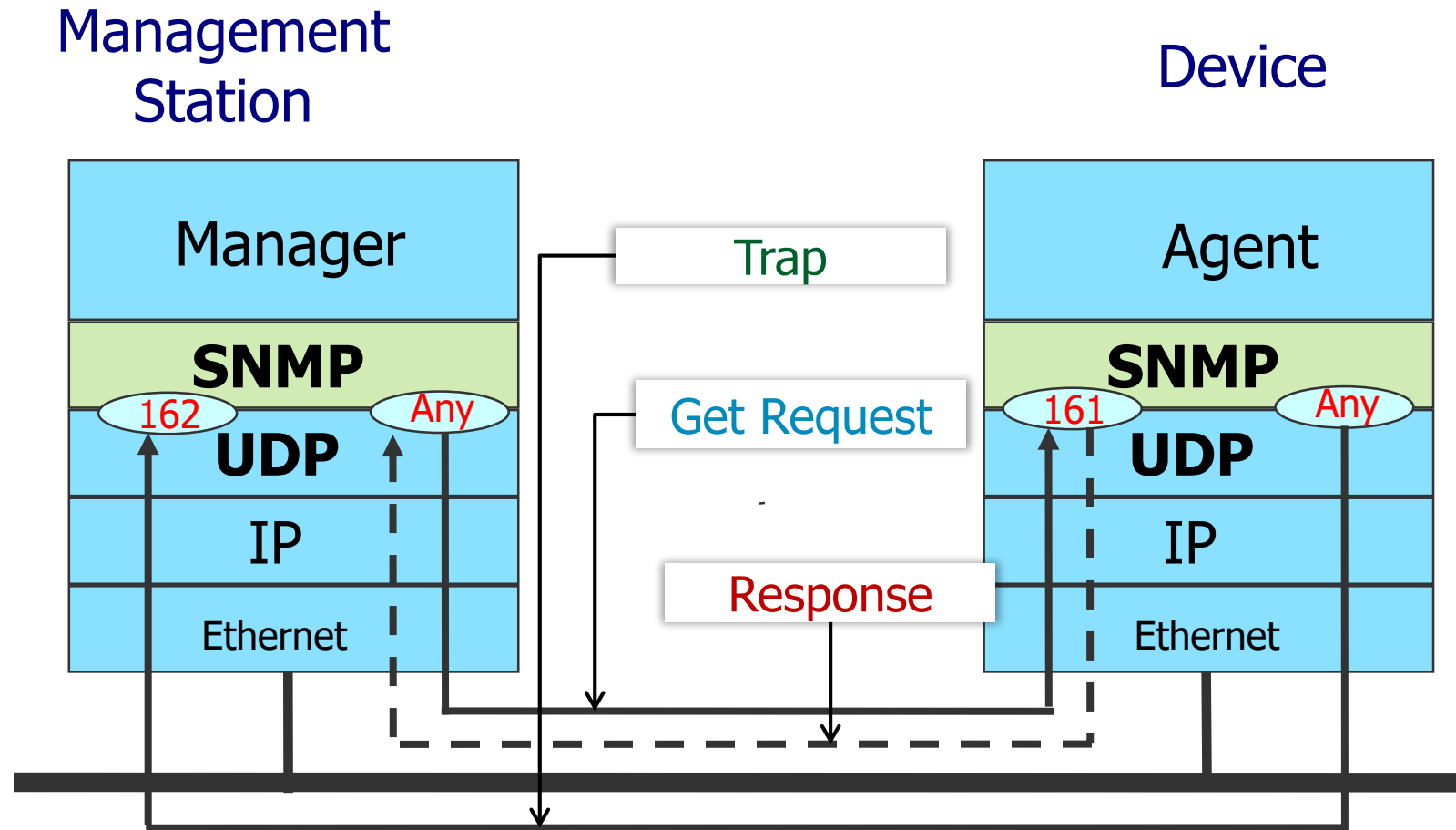


What is the number of UDP user datagrams received?



# Netzwerkmanagement

## ■ Default UDP Ports für SNMP:



# Anwendungsschicht

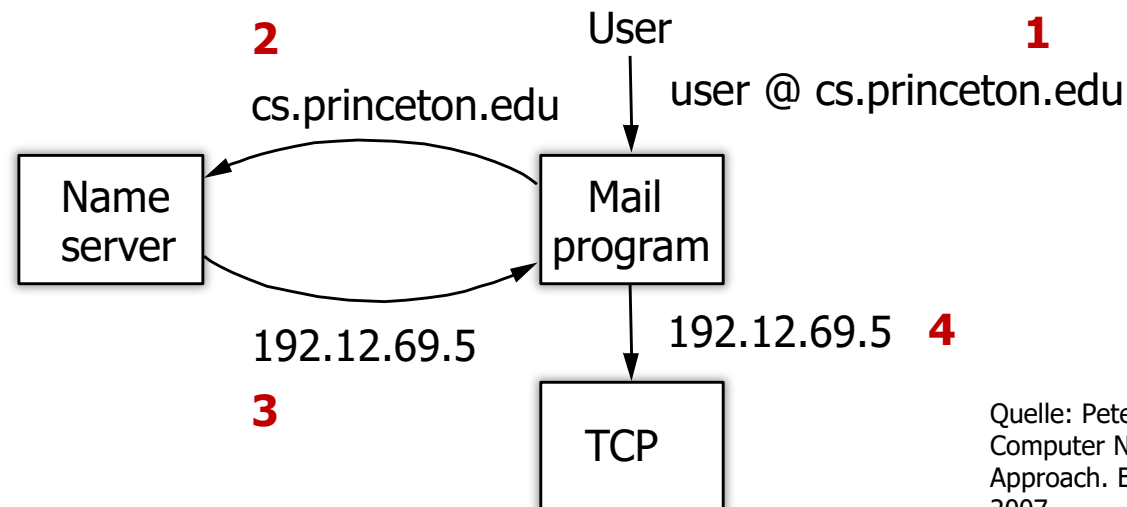
- ✓ Einführung
- ✓ Verbreitete Anwendungen
  - ✓ Hypertext Transfer Protocol (HTTP)
  - ✓ File Transfer Protocol (FTP)
  - ✓ E-Mail
  - ✓ Netzwerkmanagement
    - Domain Name System (DNS)
    - Content Distribution Networks
- Socket-Programmierung
- Peer-to-Peer-Systeme



# DNS

## ■ Domain Name System (DNS)

- Host-Namen bzw. Domain-Namen lesbar
- DNS bildet Domain-Namen auf Werte ab
- diese Werte sind u.a. IP-Adressen
- DNS ist verteilte Datenbank, besteht aus vielen Namen-Servern, die über ein Anwendungsprotokoll kommunizieren
- eine wesentliche Aufgabe, um die Infrastruktur zu nutzen
- z.B. Namens-Auflösung beim Versenden einer E-Mail:

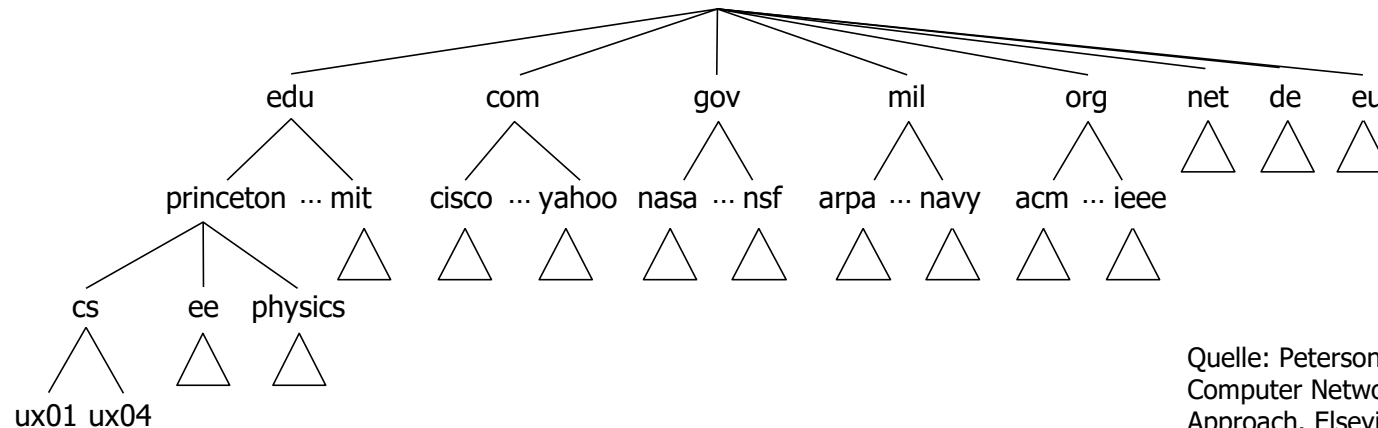


Quelle: Peterson, Davie.  
Computer Networks: A Systems  
Approach. Elsevier, 4th Ed.,  
2007.

# DNS

## ■ Domain-Struktur

- DNS implementiert hierarchischen Namensraum für Internet-Objekte
- von links nach rechts lesen, von rechts nach links verarbeiten
- eine **Zone** wird von einem Name-Server verwaltet
- die Hierarchie wird durch die Name-Server implementiert



Quelle: Peterson, Davie.  
Computer Networks: A Systems  
Approach. Elsevier, 4th Ed.,  
2007.

# DNS

## ■ Hierarchie von Name-Servern

- Root Name Server

weltweit

Wo ist der .com Domain server?

- Top-level Domain-Server

für com, org, net, edu, uk, de, eu, ...

- ...

Wo ist fau.de

- autoritativer Name-Server

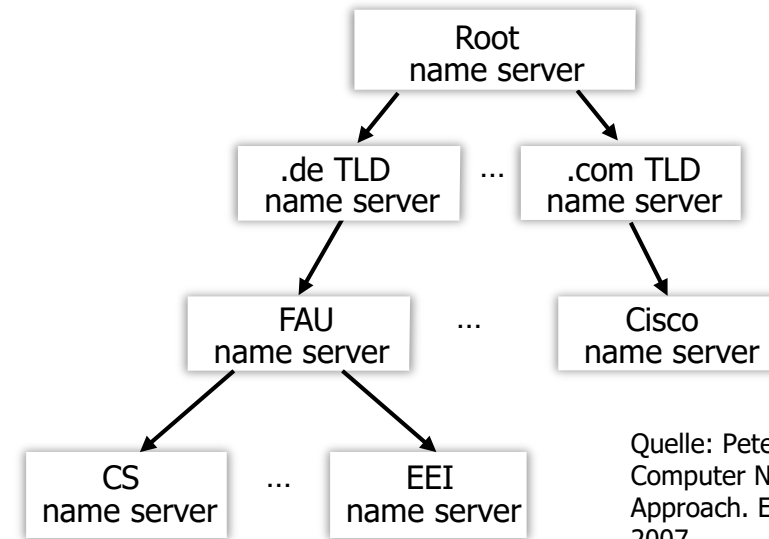
wo ist fau.cs.de

unterste Ebene, für einzelne Organisation

## ■ 13 Root Name Server

- Lastverteilung mittels Anycast, die meisten werden durch viele Server realisiert

- insgesamt mehrere hundert Server



Quelle: Peterson, Davie.  
Computer Networks: A Systems  
Approach. Elsevier, 4th Ed.,  
2007.



# DNS: Resource Records

## ■ Resource Records

- Datensätze der Namenserver (Domainname, Wert, Typ, TTL)
- **TTL:** Time to Live, Dauer der Gültigkeit
- **Typ = A**
  - Wert = IPv4-Adresse
  - Bsp.: (ns.cisco.com, 128.96.32.20, A, TTL)
  - AAAA-Einträge für IPv6-Adressen ipv6 ist auch 4 mal so lang...
- **Typ = NS**
  - Wert = Domainname eines Hosts, auf dem ein Namen-Server läuft, der Namen in der Domain auflösen kann also die DNS vom fau.de nameserver
  - Bsp.: (princeton.edu, cit.princeton.edu, NS, TTL)
- **Typ = CNAME** (Canonical Name)
  - Wert = kanonischer Name eines Hosts, ermöglicht Aliasnamen also 2 namen, die die selbe seite referenzieren
  - Bsp.: (cic.cs.princeton.edu, cicada.cs.princeton.edu, CNAME, TTL)
- **Typ = MX** (Mail Exchange)
  - Wert = Domain-Name des Hosts, auf dem Mail-Server läuft
  - Bsp.: (cs.princeton.edu, optima.cs.princeton.edu, MX, TTL)

# DNS: Resource Records

## ■ Bsp: Resource Records

- Root Name Server

(princeton.edu, cit.princeton.edu, NS, TTL)

(cit.princeton.edu, 128.196.128.233, A, TTL)

(cisco.com, ns.cisco.com, NS, TTL)

(ns.cisco.com, 128.96.32.20, A, TTL)

...

also NS ist der name eines servers, auf dem ein DNS läuft und A ist die eigentliche IP eines serv

- enthält einen NS-Datensatz für jeden Server der nächsten Ebene und einen A-Datensatz mit der IP-Adresse
- diese bilden zusammen einen Verweis auf die Server der zweiten Ebene

Quelle: Peterson, Davie.  
Computer Networks: A Systems  
Approach. Elsevier, 4th Ed.,  
2007.

# DNS: Resource Records

- Server von princeton.edu

Quelle: Peterson, Davie.  
Computer Networks: A Systems  
Approach. Elsevier, 4th Ed.,  
2007.

(cs.princeton.edu, optima.cs.princeton.edu, NS, TTL)

(optima.cs.princeton.edu, 192.12.69.5, A, TTL) Dieser hat noch einen DNS zwischengeschaltet

(ee.princeton.edu, helios.ee.princeton.edu, NS, TTL)

(helios.ee.princeton.edu, 128.196.28.166, A, TTL)

(jupiter.physics.princeton.edu, 128.196.4.1, A, TTL) Dies weisen direkt auf den zielserver

(saturn.physics.princeton.edu, 128.196.4.2, A, TTL)

(mars.physics.princeton.edu, 128.196.4.3, A, TTL)

(venus.physics.princeton.edu, 128.196.4.4, A, TTL)

- einige Datensätze sind Verweise auf die dritte Ebene, einige lösen die IP-Adressen direkt auf

# DNS: Resource Records

- Server der Domain cs.princeton.edu

(optima.cs.princeton.edu, 192.12.69.5, A, TTL)  
(cheltenham.cs.princeton.edu, 192.12.69.60, A, TTL)  
(baskerville.cs.princeton.edu, 192.12.69.35, A, TTL)  
(che.cs.princeton.edu, cheltenham.cs.princeton.edu, CNAME, TTL)  
(opt.cs.princeton.edu, optima.cs.princeton.edu, CNAME, TTL)  
(bas.cs.princeton.edu, baskerville.cs.princeton.edu, CNAME, TTL)  
(www.cs.princeton.edu, optima.cs.princeton.edu, CNAME, TTL)  
(cs.princeton.edu, optima.cs.princeton.edu, MX, TTL)

Quelle: Peterson, Davie.  
Computer Networks: A Systems  
Approach. Elsevier, 4th Ed.,  
2007.

hier werden die kurzformen auf die eigentlichen vollnamen

- enthält A-Datensätze für alle Hosts
- Aliasnamen: praktischere Namen, erlaubt Flexibilität, z.B. für Web-Server
- MX-Datensätze: gleicher Zweck speziell für Mail-Server

# DNS: Protokoll

## ■ DNS-Protokoll

- Anfrage- und Antwortnachrichten, gleiches Format:
- Kopf
  - Identification: Zuordnung Anfrage, Antwort
  - Flags: Art der Anfrage bzw. Antwort
- Rumpf
  - Questions: Domainnamen
  - Answers: Resource Records
  - Authority: Antworten von autoritativen Servern

Identification	Flags
Number of questions	Number of answers RRs
Number of authority RRs	Number of additional RRs
Questions (variable number of questions)	
Answers (variable number of resource records)	
Authority (variable number of resource records)	
Additional information (variable number of resource records)	

Quelle: Kurose, Ross.  
*Computer Networking: A Top-Down Approach*, 7th Ed.,  
Pearson Education, 2017.



# DNS: Protokoll

## ■ Anfragearten

### ● iterativ

- Antwort: anderer Server, der Namen evtl. auflösen kann (oder keine Antwort)
- NS- und A-Datensatz
- Antwort wird sofort geliefert, es muss keine Information gespeichert werden, gut für hochfrequentierte Server

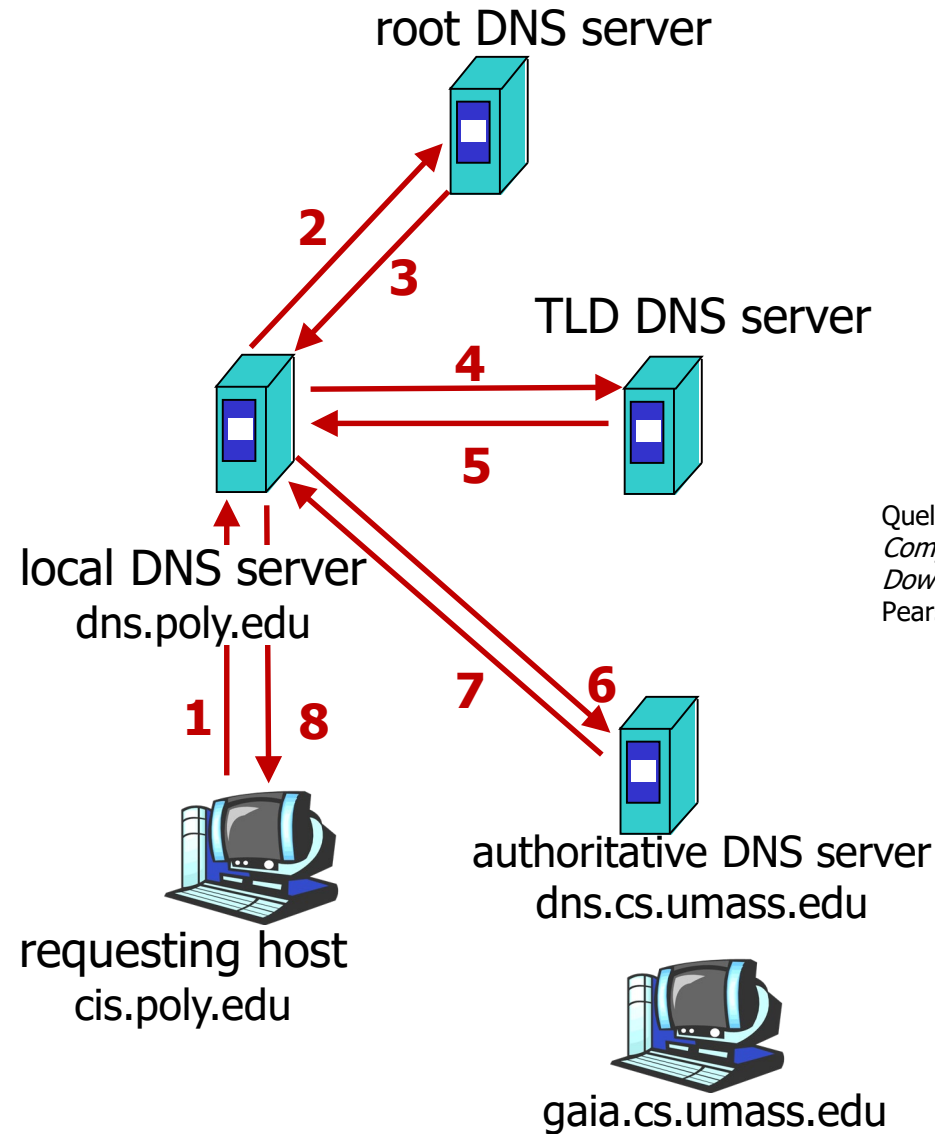
### ● rekursiv

- Antwort: Auflösung des Namens, die u.U. von anderen Servern geholt wird
- A-Datensatz
- bei Anfrage an einen anderen Server muss die Information gespeichert werden

Also wer macht die arbeit: iterativ: der angefragte muss sich um die richtige

# DNS: Protokoll

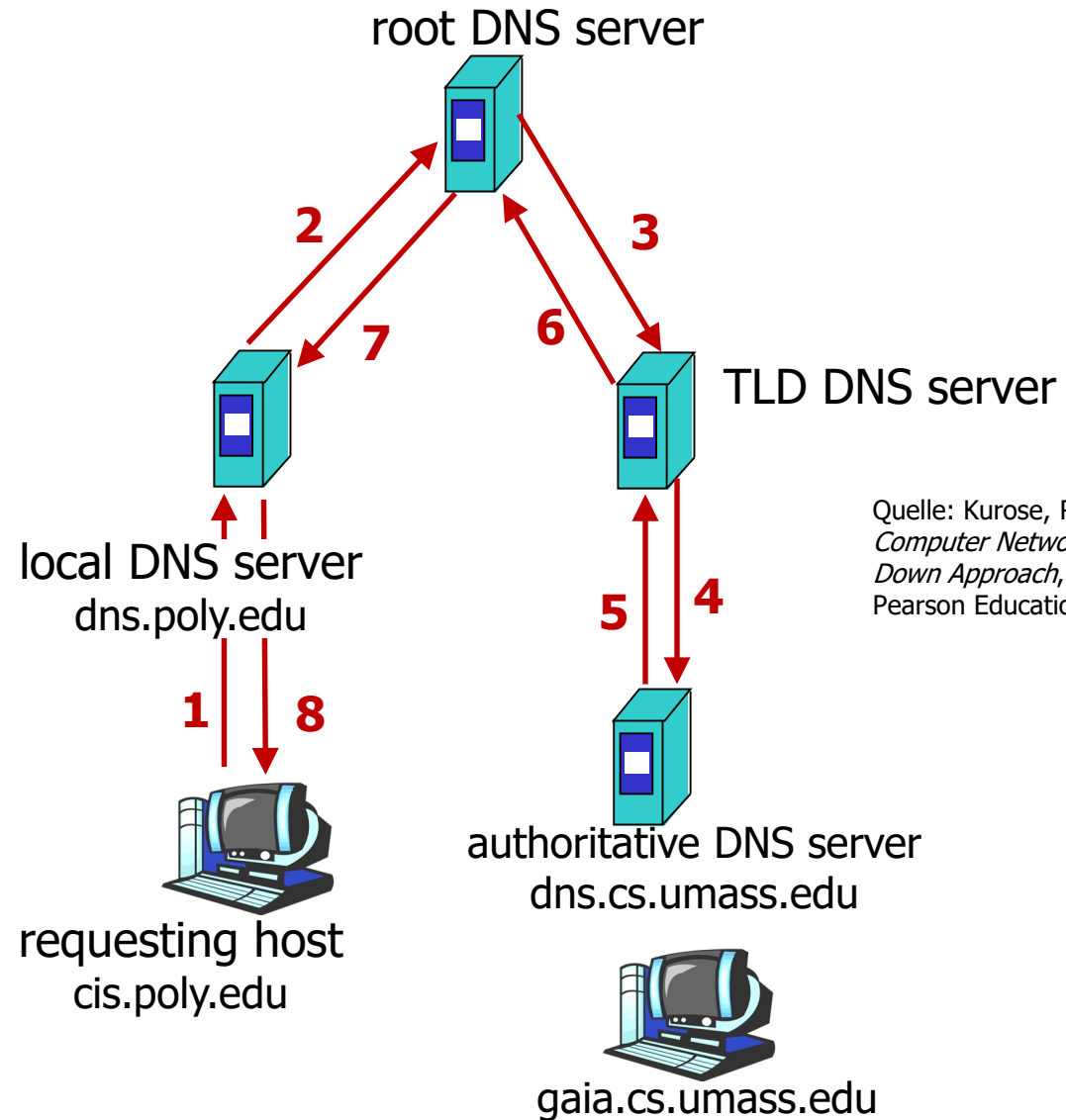
## ■ Beispiel für eine iterative Anfrage:



Quelle: Kurose, Ross.  
*Computer Networking: A Top-Down Approach*, 7th Ed.,  
Pearson Education, 2017.

# DNS: Protokoll

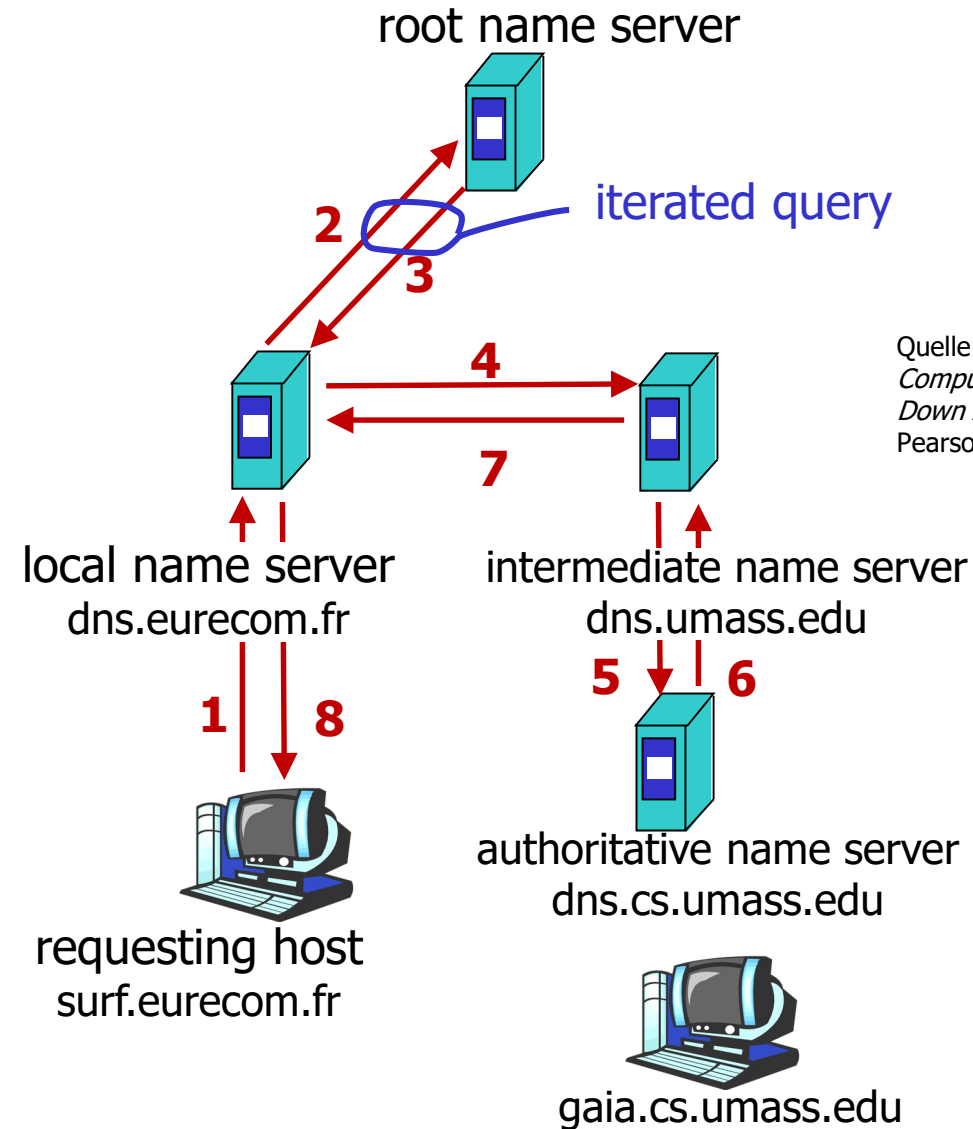
## ■ Beispiel für eine rekursive Anfrage:



Quelle: Kurose, Ross.  
*Computer Networking: A Top-Down Approach*, 7th Ed.,  
Pearson Education, 2017.

# DNS: Protokoll

- Kombination aus rekursiver und iterativer Anfrage:



Quelle: Kurose, Ross.  
*Computer Networking: A Top-Down Approach*, 7th Ed.,  
Pearson Education, 2017.

# Anwendungsschicht

- ✓ Einführung
- ✓ Verbreitete Anwendungen
  - ✓ Hypertext Transfer Protocol (HTTP)
  - ✓ File Transfer Protocol (FTP)
  - ✓ E-Mail
  - ✓ Netzwerkmanagement
  - ✓ Domain Name System (DNS)
  - Content Distribution Networks
- Socket-Programmierung
- Peer-to-Peer-Systeme

# Content Distribution Networks

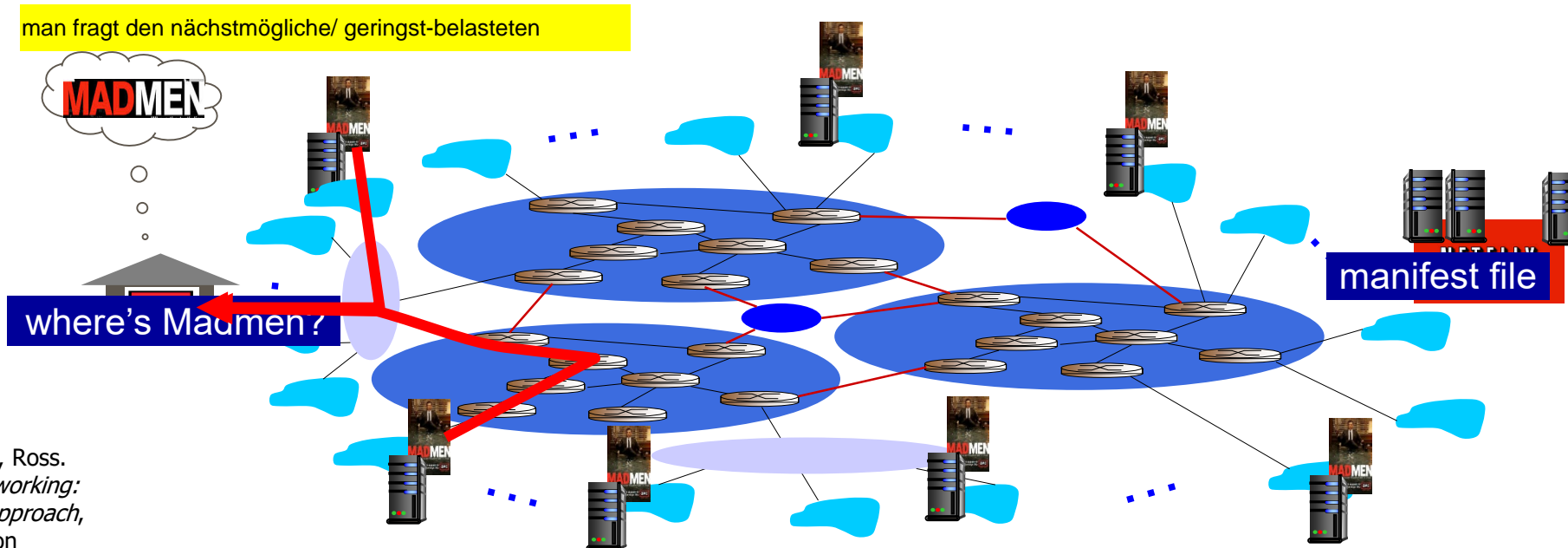
## ■ Content Distribution Networks (CDNs)

- Ziel:
  - Video-Streaming für Milliarde Nutzer weltweit mit je mehreren Mbit/s gleichzeitig
  - Reduktion von Wartezeiten beim Laden von Web-Seiten, auch bei Flash-Crowds (Millionen Benutzer greifen auf eine Seite zu)
- Idee: Statt singulärem, massivem Server → sehr viele (bis zu mehrere Hunderttausend) Spiegel-Server geografisch verteilen und näher am Nutzer platzieren
- 3 Probleme vermeiden:
  - Engpässe auf Strecke zum Nutzer minimieren: erste Meile, letzte Meile, Peering-Punkte (Übergänge zwischen ISPs)
  - Keine mehrfache Übertragung des selben Inhalts über eine Strecke
  - Vermeiden eines Single Point of Failure

# Content Distribution Networks

## ■ Content Distribution Networks (CDNs)



- Konzept ähnlich zu Web-Caches, der Inhalt wird aber abhängig von der Beliebtheit proaktiv repliziert
- bekannte CDNs: 3rd-party (Akamai, Limelight), privat (Google→YouTube)
- z.B. Netflix:



Quelle: Kurose, Ross.  
*Computer Networking:  
A Top-Down Approach*,  
7th Ed., Pearson  
Education, 2020.

# Content Distribution Networks

## ■ Möglichkeiten für die Verteilung der Anfragen

- **Server-basierte HTTP Redirection:** Server liefert aufgrund der IP-Adresse des Clients einen geeigneten anderen Server, erfordert zusätzliche RTT, Server-Betreiber muss Verteilung des Inhalts kennen, nicht bewährt  
 wie caches, aber schwierig zu bauen
- **Client-nahe HTTP-Redirection:** z.B. durch Web-Proxy, schwieriger zu verwirklichen, nicht bewährt
- **URL-Rewriting:** Server liefert Basisseite, die URLs der eingebetteten Objekte werden umgeschrieben, mit dem Domain-Namen eines geeigneten anderen Servers  
 Der content wird auf verschiedene
- **DNS-basierte Redirection:** DNS-Server bilden den Domain-Namen des Services auf Domain-Namen und zuletzt auf die IP-Adresse eines geeigneten Servers ab
- kommerzielle CDNs verwenden Kombination aus URL-Rewriting und DNS-basierter Redirection
- Wahl geeigneter Server geographisch nah, wenig ausgelastet, ...

Der DNS liefert

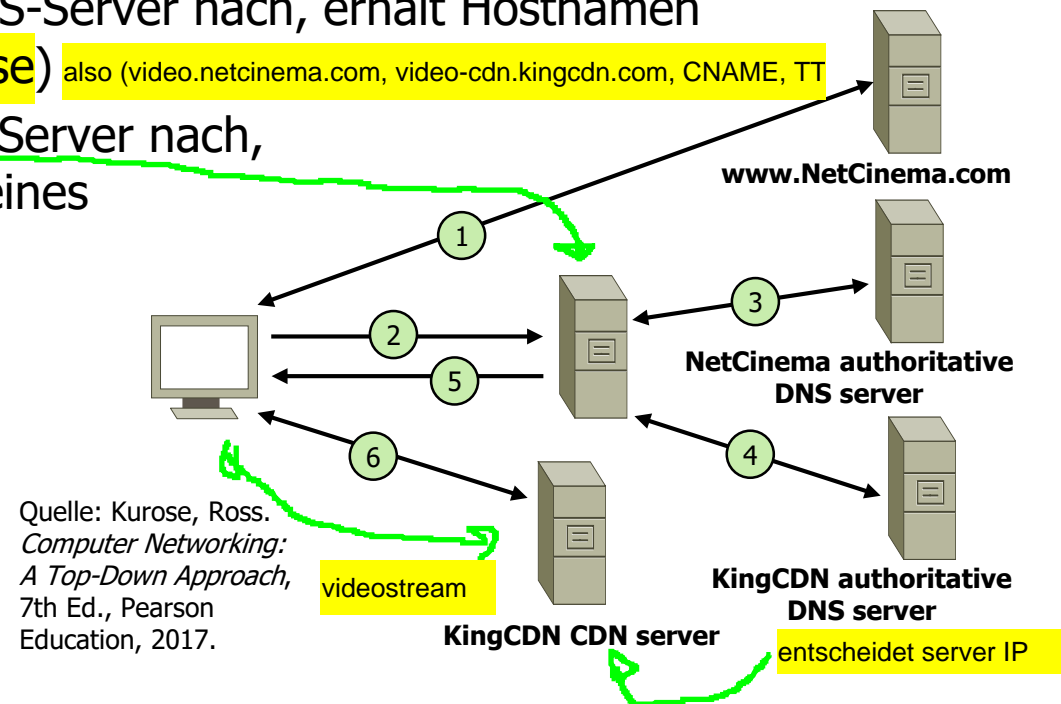




# Content Distribution Networks ✓

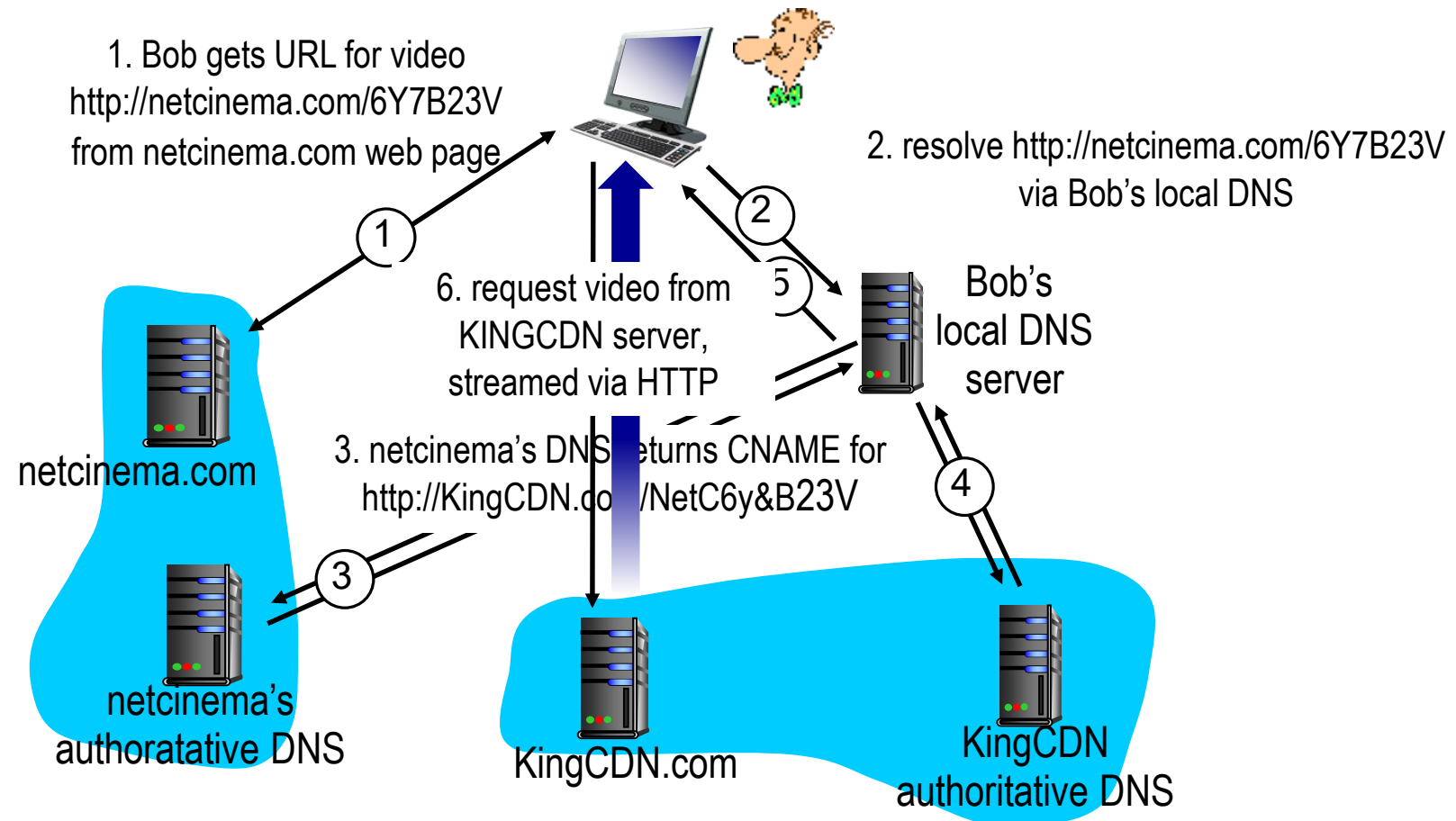
## ■ Bsp. für DNS-basierte Redirection bei 3rd-party-CDN

- www.NetCinema.com ist Video-Dienst, Video-Dateien werden auf den CDN-Servern von kingcdn.com verteilt
- 1. HTTP-Anfrage für NetCinema-Seite mit Links zu Videos
- 2. Nutzer klickt auf `http://video.netcinema.com/video1234`, DNS-Anfrage für IP-Adresse von `video.netcinema.com`
- 3. Lokaler DNS-Server fragt bei NetCinema-DNS-Server nach, erhält Hostnamen `video-cdn.kingcdn.com` (CNAME statt IP-Adresse) also (`video.netcinema.com`, `video-cdn.kingcdn.com`, CNAME, TT)
- 4. Lokaler DNS-Server fragt bei KingCDN-DNS-Server nach, erhält aufgrund seiner IP-Adresse IP-Adresse eines geographisch nahe gelegenen CDN-Servers
- 5. Lokaler DNS-Server gibt IP-Adresse des auserwählten CDN-Servers an Nutzer-PC weiter
- 6. HTTP-Anfrage an diesen Server



# Content Distribution Networks

## ■ Und nochmal:



Quelle: Kurose, Ross.  
*Computer Networking:  
A Top-Down Approach*,  
7th Ed., Pearson  
Education, 2020.

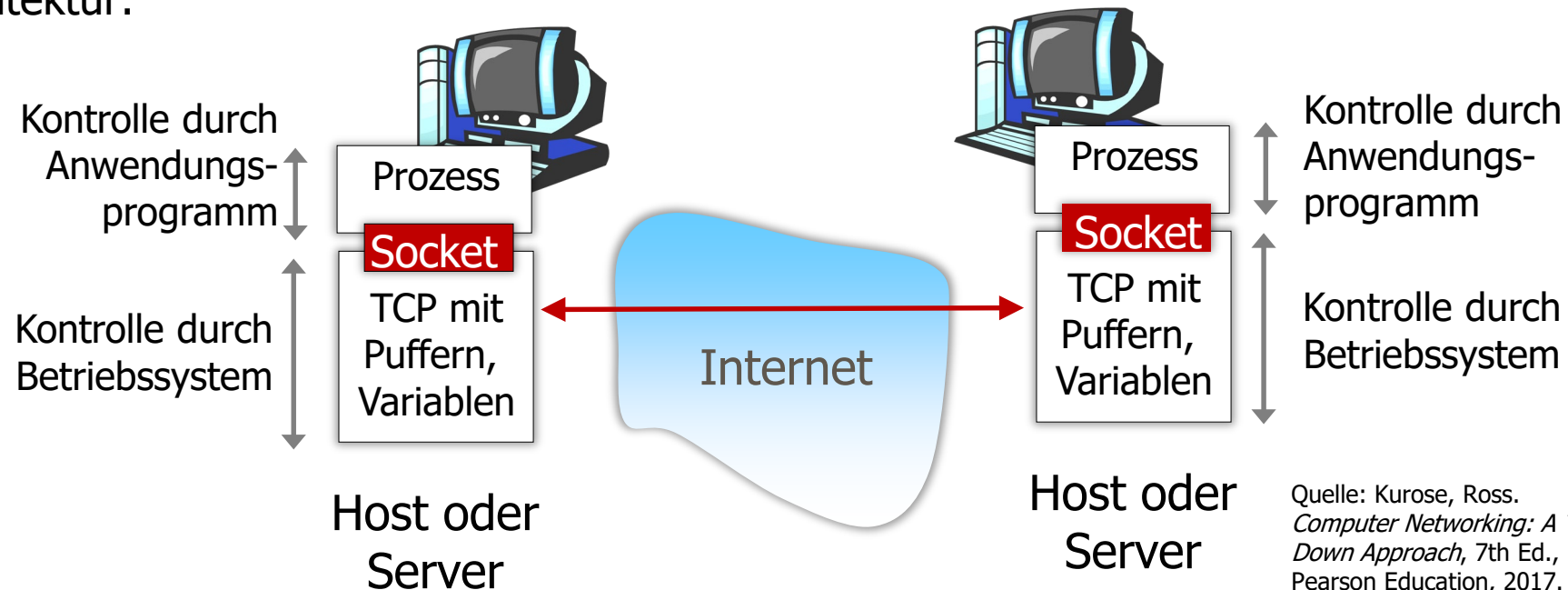
# Anwendungsschicht

- ✓ Einführung
- ✓ Verbreitete Anwendungen
  - ✓ Hypertext Transfer Protocol (HTTP)
  - ✓ File Transfer Protocol (FTP)
  - ✓ E-Mail
  - ✓ Netzwerkmanagement
  - ✓ Domain Name System (DNS)
  - ✓ Content Distribution Networks
- **Socket-Programmierung**
- **Peer-to-Peer-Systeme**

# Socket-Programmierung

## ■ Socket-Schnittstelle

- verbreitetes API für Transportdienste
- Festlegung von TCP/UDP, IP-Adressen, Portnummern
- im folgenden Java-Programm mit TCP (UDP in Übung)
- Java erlaubt flexible Nutzung der Strom-Abstraktion
- Architektur:

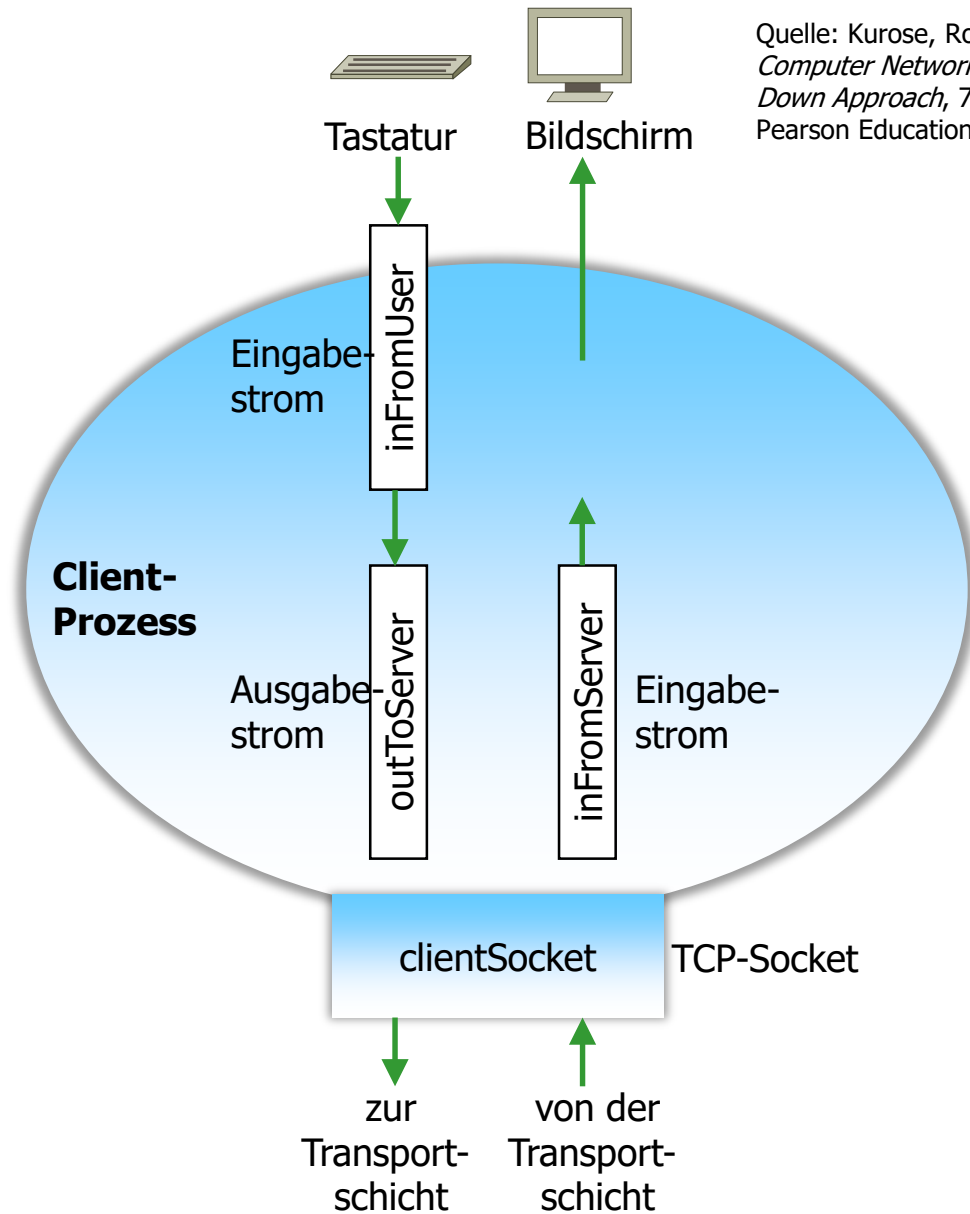


Quelle: Kurose, Ross.  
*Computer Networking: A Top-Down Approach*, 7th Ed.,  
Pearson Education, 2017.

# Socket-Programmierung

## ■ Beispiel für einfache Client-Server-Anwendung

- Client liest Zeile aus Standardeingabe (inFromUser stream) und sendet ihn über ein TCP-Socket zum Server
- Server liest Zeile aus TCP-Socket
- Server konvertiert in Großbuchstaben (seine Dienstleistung) und sendet diese Zeichenkette über TCP-Socket zurück an Client
- Client liest Zeichenkette aus TCP-Socket und gibt diese auf Standardausgabe aus



Quelle: Kurose, Ross.  
*Computer Networking: A Top-Down Approach*, 7th Ed.,  
Pearson Education, 2017.

# Socket-Programmierung: Übersicht

## Server-Prozess auf hostid

erzeuge Socket für eingehende Verbindungswünsche an Port x:

`welcomeSocket =  
ServerSocket(x)`

warte auf Verbindungswünsche:  
`connectionSocket =  
welcomeSocket.accept()`

lese Zeile aus  
`connectionSocket`

schreibe Antwort in  
`connectionSocket`

schließe  
`connectionSocket`

## Client-Prozess

erzeuge Socket, Verbindung zu hostid, port=x  
`clientSocket =  
new Socket(hostid, x)`

schreibe Zeile in  
`clientSocket`

lese Antwort aus  
`clientSocket`

schließe  
`clientSocket`

TCP-  
Verbindungsaufbau

TCP-  
Verbindungsabbau

Quelle: Kurose, Ross.  
*Computer Networking: A Top-Down Approach*, 7th Ed.,  
Pearson Education, 2017.

## ➤ Praktische Vorgehensweise und Programmierung in der Übung

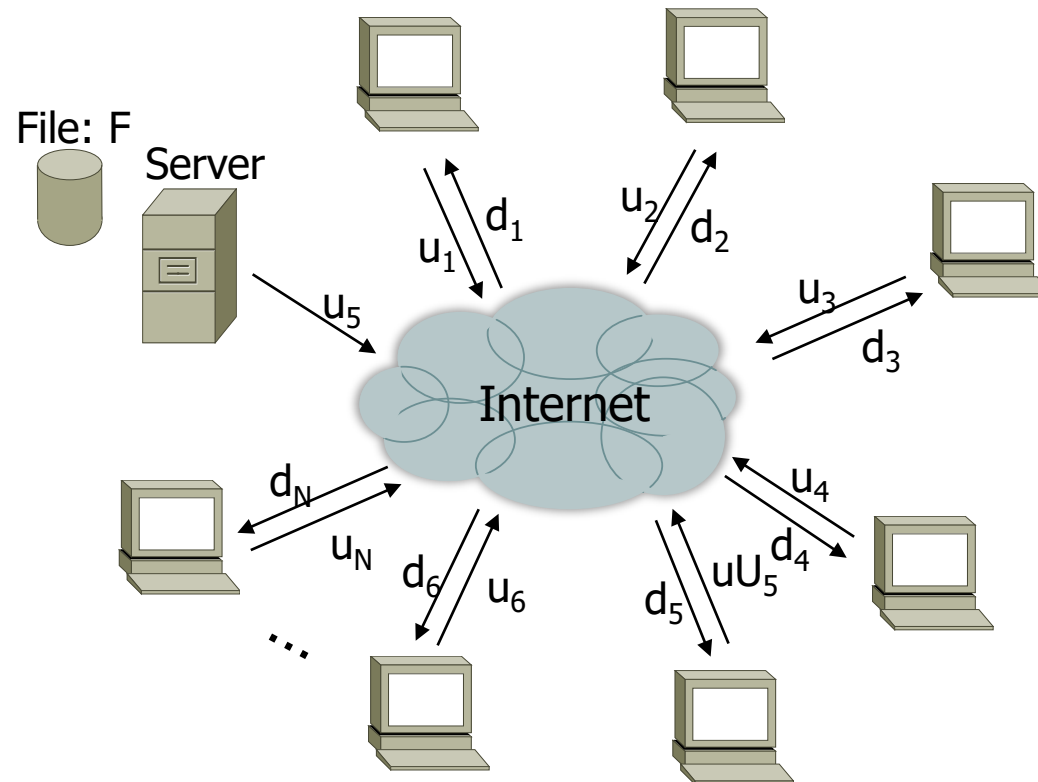
# Anwendungsschicht

- ✓ Einführung
- ✓ Verbreitete Anwendungen
  - ✓ Hypertext Transfer Protocol (HTTP)
  - ✓ File Transfer Protocol (FTP)
  - ✓ E-Mail
  - ✓ Netzwerkmanagement
  - ✓ Domain Name System (DNS)
  - ✓ Content Distribution Networks
- ✓ Socket-Programmierung
- **Peer-to-Peer-Systeme**

# P2P

## ■ Peer-to-Peer

- bekannt von Anwendungen zum Filesharing
- Grundidee: Inhalte nicht nur von zentralem Server, sondern auch von anderen Peers
- Upload-Bitrate der Peers wird mitgenutzt



Quelle: Kurose, Ross.  
*Computer Networking: A Top-Down Approach*, 7th Ed.,  
Pearson Education, 2017.



# P2P

## ■ Peer-to-Peer

- Anwendungen wie Napster zum direkten Austausch von Musikdateien (MP3) haben P2P populär gemacht
- Napster aus juristischen Gründen stillgelegt, diverse Nachfolger (Gnutella, ...)
- Peers kommunizieren direkt mittels TCP oder UDP
- P2P-Netze bilden **Overlay-Netz**: logisches Netz aus Peers über dem physikalischen Netz

## P2P: Architekturen

### ■ Unstrukturiert (1st/2nd Generation)

- zentralisierte P2P Netze (z.B. Napster)
  - Datenaustausch P2P, Verzeichnis aber zentral
- reine P2P Netze (z.B. Gnutella 0.4)
  - keine Zentrale, vollständige Vermaschung der Peers (Skalierbarkeit?!)
- hybride P2P Netze (z.B. Gnutella 0.6)
  - Inseln mit Serverstruktur, Vermaschung der Inseln

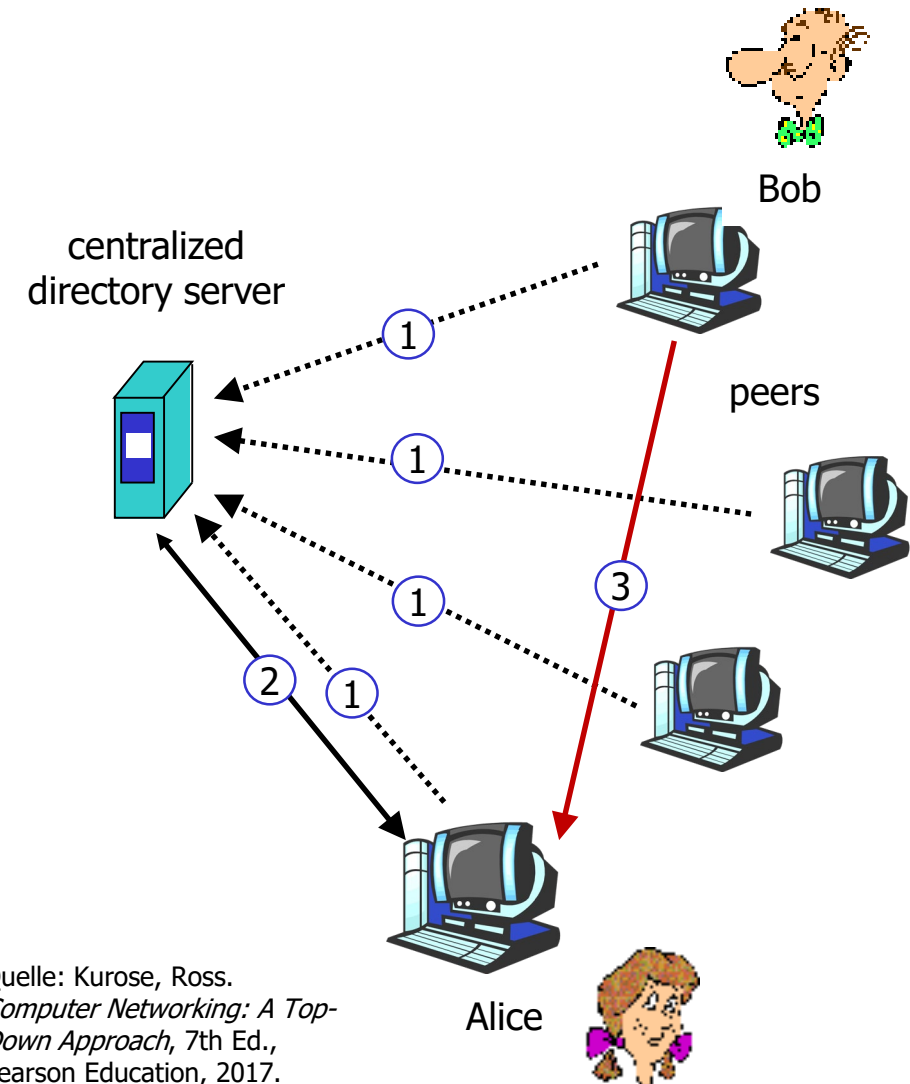
### ■ Strukturiert

- DHT (Distributed Hash Table) basiert
- z.B. Chord, CAN
- keine zentralen Server, hochgradig skalierbar

# P2P: Unstrukturiert (Zentralisiert)

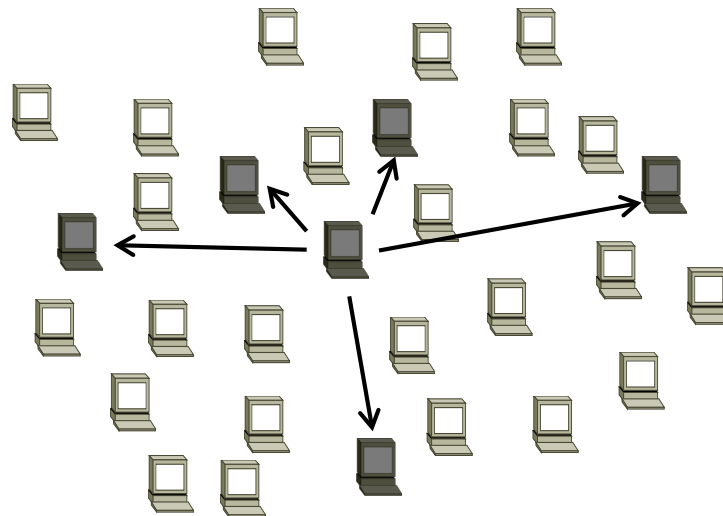
## ■ Zentralisiertes Verzeichnis

- Architektur von Napster
- Eintritt eines Peers:  
informiert zentralen Server über seine IP-Adresse  
und seine Inhalte
- Suche nach Inhalt:  
über zentralen Server
- Dateiübertragung: direkt zwischen Peers
- zentraler Server
  - juristischer „Schwachpunkt“
  - Leistungs- und Zuverlässigkeitsproblem



## P2P: Unstrukturiert (Rein P2P)

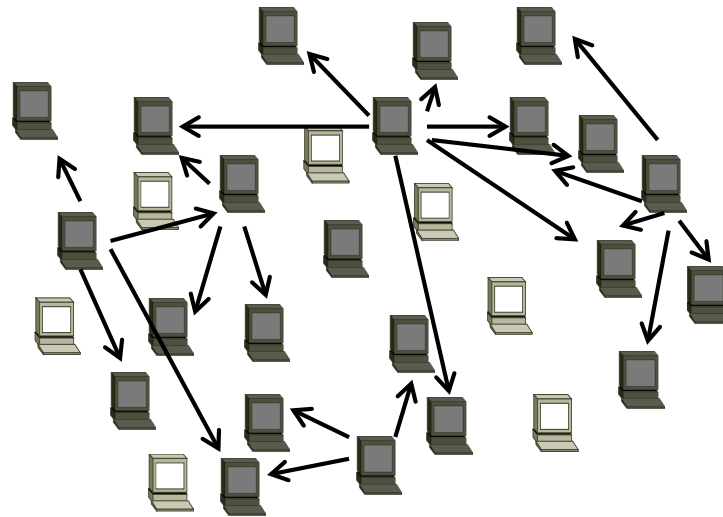
- Dezentralität durch Fluten von Anfragen
  - Peers bilden Overlay-Netzwerk über TCP-Verbindungen



- anfragender Peer sendet Anfrage an alle seine Nachbarn im Overlay-Netzwerk
- diese vergleichen Anfrage mit den von ihnen angebotenen Inhalten

## P2P: Unstrukturiert (Rein P2P)

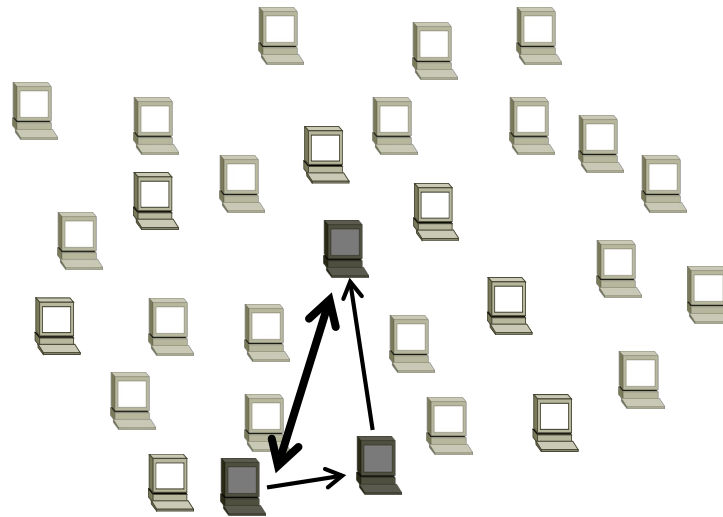
- Dezentralität durch Fluten von Anfragen (Fortsetzung)
  - Fluten: wenn sie die Anfrage nicht beantworten können, wird sie an mehrere Nachbarn weitergeleitet (aber nicht an den Peer, von dem die Anfrage kommt)



- das Fluten wird durch einen maximalen Hopcount begrenzt

## P2P: Unstrukturiert (Rein P2P)

- Dezentralität durch Fluten von Anfragen (Fortsetzung)
  - wenn ein Peer den Inhalt anbieten kann, antwortet er dem anfragenden Peer, dieser leitet wiederum zurück (die Identität des ursprünglich anfragenden Peers bleibt so unbekannt)



- die Antwort findet zur Quelle zurück, diese kontaktiert direkt einen der Peers, der die Anfrage beantworten kann, die Übertragung erfolgt z.B. mittels HTTP

der antworter ruft irgendwo ins netz. die nachricht kommt über umwege an

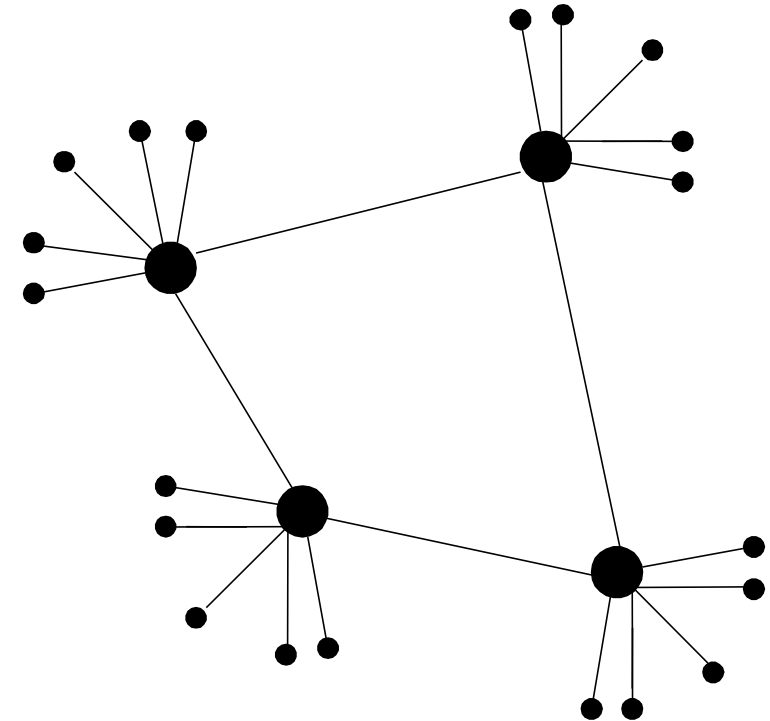
## P2P: Unstrukturiert (Rein P2P)

- Dezentralität durch Fluten von Anfragen (Fortsetzung)
  - Architektur von Gnutella
  - kein zentraler Server benötigt
  - Eintritt in das Overlay-Netzwerk: Nachricht an eine veröffentlichte Liste von möglichen Peers schicken
  - Skalierbarkeit ist wegen des Flutens problematisch

## P2P: Unstrukturiert (Hybrid)

### ■ Hierarchie

- Peers bilden Gruppen, einer ist Group Leader
- Group Leader kennt Inhalte aller Peers aus Gruppe (Gruppe  $\approx$  „Mini-Napster“)
- Overlay-Netzwerk zwischen Group Leadern
- Austausch zwischen Group Leadern ähnlich wie bei Gnutella
- bessere Skalierbarkeit, ebenfalls keine zentrale Kontrolle



Quelle: Kurose, Ross.  
*Computer Networking: A Top-Down Approach*, 7th Ed.,  
Pearson Education, 2017.



## P2P: Strukturiert

### ■ Verteilte Hash-Tabellen (Distributed Hash Tables, DHT)

- dezentrales Verfahren für Speicherung von Datenelementen, bekannt z.B. aus Chord-System
- Peers bilden ringförmiges Overlay-Netz
- jedem **Peer** wird zufälliger **Bezeichner**  $p$  ( $0 \leq p \leq 2^m - 1$ ) aus ringförmigen Bezeichnerraum mit  $m$  Bits zugewiesen
- jedem **Datenelement** wird mittels Hash-Funktion ein **Schlüssel**  $k$  ebenfalls aus diesem Raum zugewiesen
- **Nachfolger von  $k$** 
  - Datenelement mit Schlüssel  $k$  wird auf „Nachfolger“  $p$  von  $k$  gespeichert (Nachfolger darf Knoten  $p$  selbst sein)
  - der nächste Peer im Ring, formal: Peer  $p$  mit dem kleinsten  $a \geq 0$ , so dass  $p = \text{succ}(k) = (k+a) \bmod 2^m$  existiert (also ringförmig über  $2^m - 1$  hinaus)
  - Auslesen eines Datenelements mit Schlüssel  $k$  erfolgt auf Peer  $\text{succ}(k)$

## P2P: Strukturiert (Beispiel Chord)

### ■ Routing

- Finden des gesuchten Eintrags in der DHT
- Chord verwaltet Ringstruktur über alle Einträge
  - jeweils Vorgänger und Nachfolger
- Routing entlang des Rings (linearer Aufwand, ineffizient) oder durch Sprungtabellen ([Finger-Tabelle](#), logarithmischer Aufwand)
- es gibt auch Verweis auf Vorgänger (wird hier vernachlässigt)

### ■ Neuen Knoten einfügen

- Voraussetzung: ein bekannter Knoten p in der DHT
- neue ID wählen zwischen p und Nachfolger von p, Aktualisieren der Finger-Tabellen

### ■ Knoten entfernen

- Daten migrieren, ID entfernen und Finger-Tabellen aktualisieren

### ■ Selbststabilisierung

- kontinuierliche Überprüfung aller Knoten, evtl. Finger-Tabellen reparieren

## P2P: Strukturiert (Beispiel Chord)

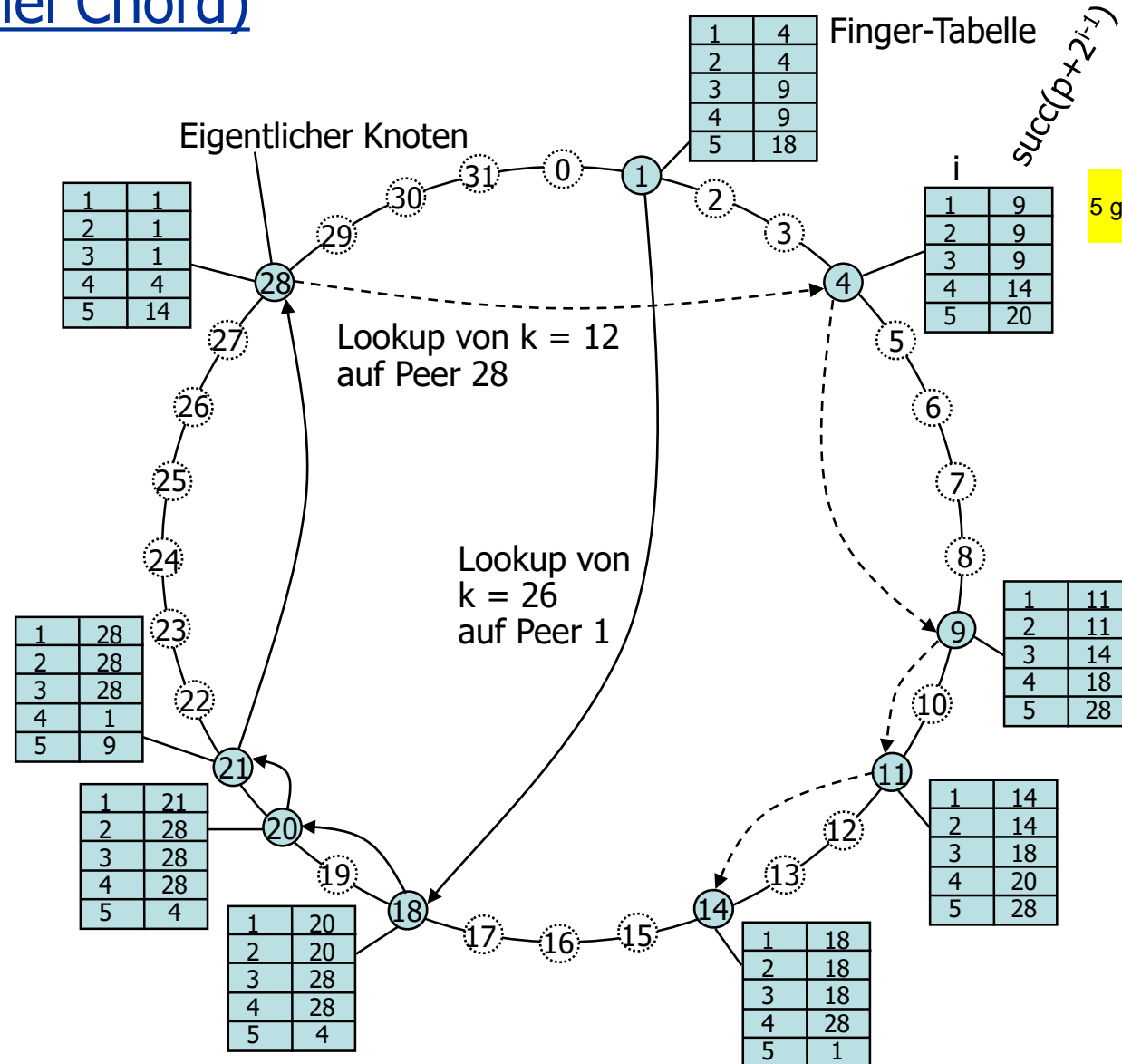
### ■ Finger-Tabelle

- jeder Peer  $p$  unterhält **Finger-Tabelle**  $FT_p$  mit  $m$  Einträgen
- $i$ -ter Eintrag der Finger-Tabelle:  $FT_p[i] = \text{succ}(p+2^{i-1})$   
enthält Nachfolger mit **exponentiell steigender Distanz**
- **Lookup** für Datenelement mit Schlüssel  $k$  kann auf beliebigem Peer  $p$  starten
- Algorithmus  $\text{lookup}(k,p)$ :
  - wenn  **$k = p$ , dann gib Peer  $p$**  als Ergebnis aus
  - wenn  $k$  zwischen  $p$  (exklusive) und  $FT_p[1]$  (inklusive) liegt,  
dann gib Peer  $q = FT_p[1]$  als Ergebnis aus
  - ansonsten wird **der Peer in der Finger-Tabelle** gesucht, der am nächsten vor  $k$  liegt und **dort ein erneutes Lookup** durchgeführt:
    - wenn  $k$  zwischen  $FT_p[i]$  (inklusive) und  $FT_p[i+1]$  (exklusive) liegt mit  $1 < i < m$   
dann  $q = FT_p[i]$   
außer bei 1 immer den nächstbesten kleinsten peer in der liste finden.
    - wenn  $k = FT_p[m]$  ist oder hinter  $FT_p[m]$  liegt,  
dann  $q = FT_p[m]$
    - gib von  $\text{lookup}(k,q)$  gefundenen Peer als Ergebnis aus

# P2P: Strukturiert (Beispiel Chord)

## ■ Beispiel

- $m = 5$ ,  
Bezeichnungsraum  
 $0 \leq p \leq 2^m - 1 = 31$
- farbige Peers  
gehören zum  
Overlay



5 gibts nicht, also step zum nächst best

Quelle: Tanenbaum.  
Computer Networks. 5th  
Ed., Prentice Hall, 2011.

## P2P: Strukturiert (Beispiel Chord)

### ■ Lookup von $k = 26$ auf Peer 1

- Lookup auf Peer 1 ergibt  $p = 18 = FT_1[5] \leq 26$
- Lookup auf Peer 18 ergibt  $p = 20 = FT_{18}[2] \leq 26 < FT_{18}[3]$
- Lookup auf Peer 20 ergibt  $p = 21 = FT_{20}[1] \leq 26 < FT_{20}[2]$
- Lookup auf Peer 21 ergibt  $p = 28 = FT_{21}[1] \geq 26$
- Ergebnis:  $\text{succ}(26) = 28$

### ■ Lookup von $k = 12$ auf Peer 28

- Lookup auf Peer 28 ergibt  $p = 4 = FT_{28}[4] \leq 12 < FT_{28}[5]$
- Lookup auf Peer 4 ergibt  $p = 9 = FT_4[3] \leq 12 < FT_4[4]$
- Lookup auf Peer 9 ergibt  $p = 11 = FT_9[2] \leq 12 < FT_9[3]$
- Lookup auf Peer 11 ergibt  $p = 14 = FT_{11}[1] \geq 12$
- Ergebnis:  $\text{succ}(12) = 14$

Quelle: Tanenbaum.  
Computer Networks. 5th  
Ed., Prentice Hall, 2011.

# P2P: Bittorrent

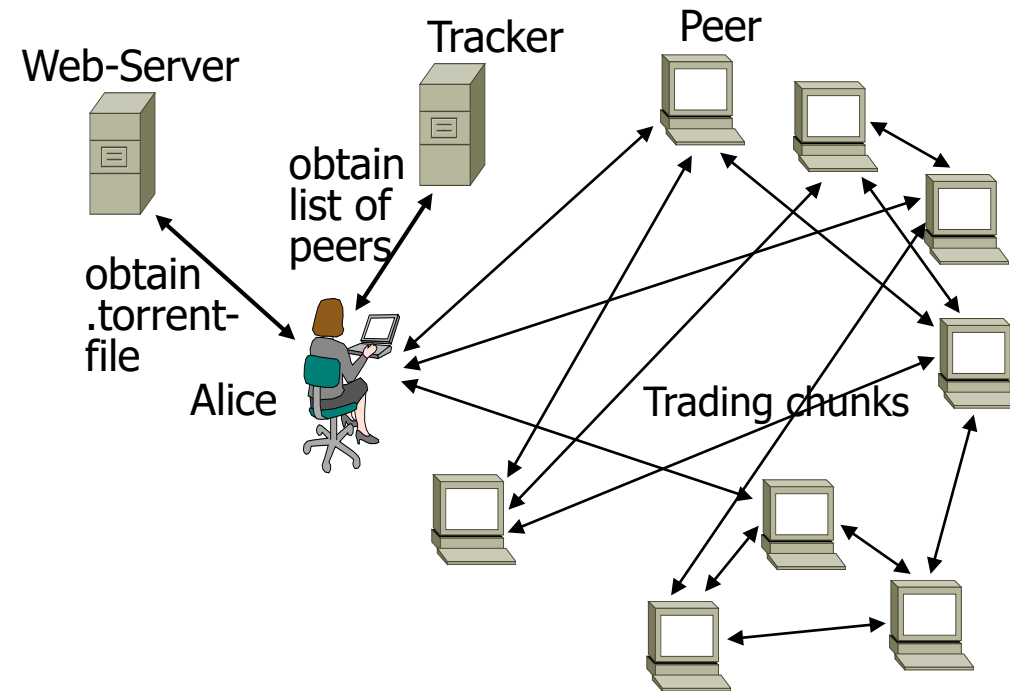
## ■ Bittorrent

- **Torrent**: Schwarm von Peers für gleiche Datei, z.B. Tausende
- **Chunks**: Teile der zu verteilenden Datei, z.B. 256 KB
- **Tracker**: Server, bei dem sich Peers registrieren
- .torrent-Datei mit Meta-Daten über zu verteilende Datei und Tracker
- neuer Peer A tritt Schwarm bei
  - A registriert sich bei Tracker und erhält IP-Adressen zufälliger anderer Peers (z.B. 50) des Schwarms
  - A baut TCP-Verbindung zu einigen dieser Peers auf, fragt Liste der Chunks in ihrem Besitz nach und sendet Anfragen für Chunks
  - **Rarest First**: A fragt die seltensten Chunks der Peers zuerst nach, dadurch gleichmäßige Verteilung  
A kann diese dann selbst verteilen
  - **Incentive Mechanismus (Tit-for-Tat)**: A misst Antwortrate der Peers und antwortet an diese in entsprechendem Anteil der Upload-Rate
  - neue Nachbarn werden zufällig dazu genommen
  - und mehr: Pipelining, Random First Piece, Endgame Model, Anti-Snubbing, ...

# P2P: Bittorrent

## ■ Bittorrent (Fortsetzung)

- Hybridarchitektur: Tracker ist zentraler Infrastrukturknoten, Rest P2P



Quelle: Kurose, Ross.  
*Computer Networking: A Top-Down Approach*, 7th Ed.,  
Pearson Education, 2017.

- trackerloser Betrieb: Trackerinformation wird über **DHT verteilt**, Torrent durch eine ID beschrieben

## P2P: Bitcoin



### ■ Bitcoin (BTC)

- „Kryptowährung“: P2P-Zahlungssystem für virtuelle Geldeinheiten ohne vertrauenswürdige zentrale Instanz (wie z.B. einer Bank)
- vorgeschlagen durch anonymen Autor Satoshi Nakamoto: “Bitcoin: A Peer-to-Peer Electronic Cash System,” 2008. Open-Source Implementierung Bitcoin Core, 2010
- **Transaktionen** von Geldbeträgen an öffentliche Bitcoin-Adressen, digital signiert mit privaten Schlüsseln, Versendung in unstrukturiertem P2P-Netz von Bitcoin-Clients (d.h. Fluten)
- **Miner** (Schürfer) sammeln Transaktionen in Blöcken und führen **Proof-of-Work** durch: Wettbewerb zur Erstellung eines Hashs des Blocks, bei Erfolg Fluten, Verkettung zu einer öffentlichen Blockchain
- **Incentive** (Belohnung) beim erfolgreichen Schürfen eines Blocks
- **Blockchain** ist verteilte Datenbank, auch verteiltes Hauptbuch (**Distributed Ledger**) genannt, mit allen Transaktionen seit Beginn
- erlaubt die Vermeidung von **Double-Spending**



# P2P: Bitcoin

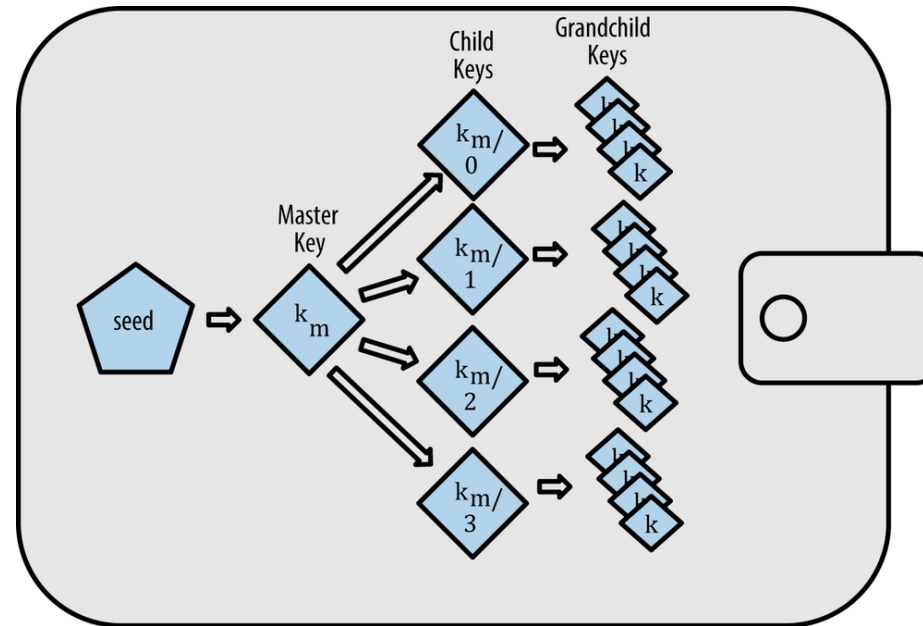
## ■ Adressen

- abgeleitet aus privaten und öffentlichen Schlüsseln basierend auf elliptischer Kurven-Kryptographie
- Schlüsselpaar
  - **Private Key** (Zufallszahl, 256 Bits), z.B. (hexadezimal):  
1E99423A4ED27608A15A2616A2B0E9E52CED330AC530EDCC32C8FFC6A526AEDD
  - ermöglicht die Berechnung des **Public Key** (Paar von 256 Bits), z.B. 1. Wert:  
F028892BAD7ED57D2FB57BF33081D5CFCF6F9ED3D3D7F159C2E2FFF579DC341A
- Abbildung Public Key auf **Bitcoin-Adresse**
  - zweifache Hashfunktion (SHA-256, RIPEMD160) anwenden
  - mit Base58 kodieren (Base64 außer 00II\+/, also Alphabet  
123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz)
  - z.B. Public Key oben: 1J7mdg5rbQyUHENYdx39WVWK7fsLpEoXZy
- Pseudonymität durch Verwendung von Bitcoin-Adressen statt Identitäten, können häufig gewechselt werden
- eingeschränkte Anonymität, da Verknüpfung der öffentlichen Transaktionsdaten mit identifizierenden Informationen ggfs. möglich ist

# P2P: Bitcoin

## ■ Adressen (Fortsetzung)

- **Wallet** (Brieftasche): Bitcoin-Client zur Verwaltung von Schlüsseln
- z.B. Wallet mit hierarchischem Schlüsselbaum, Ableitung aus einem **Seed**:



Quelle: A. M. Antonopoulos,  
Mastering Bitcoin, O'Reilly, 2014.

## P2P: Bitcoin

### ■ Transaktionen

- Transaktion = nicht ausgegebene Gutschriften (**Unspent Transaction Outputs, UTXO**) aus früheren Transaktionen werden zu neuen Bitcoin-Adressen transferiert, in Satoshi =  $10^{-8}$  BTC
  - **Input**: UTXO + Unlocking-Skript (z.B. Signierung durch zu Bitcoin-Adresse passenden privaten Schlüssel), erlaubt das Spending (Ausgeben)
  - **Output**: Transfer zu neuer Bitcoin-Adresse + Locking-Skript (z.B. Bitcoin-Adresse)
  - mehrere Inputs und Outputs möglich, dadurch Zusammenführung und Aufteilung
  - **Transaction-Fee**: Differenz aus Inputs und Outputs
  - eingeschränkte Skriptsprache erlaubt weitere Transaktionsarten
- Fluten: An Peers versenden, diese verifizieren Transaktion, überprüfen auf Double-Spending in Blockchain und senden weiter
- kein Konzept eines Kontos mit Geldbetrag, Wallet kann Summe von UTXOs bilden, Wallet stellt auch Transaktionen zusammen

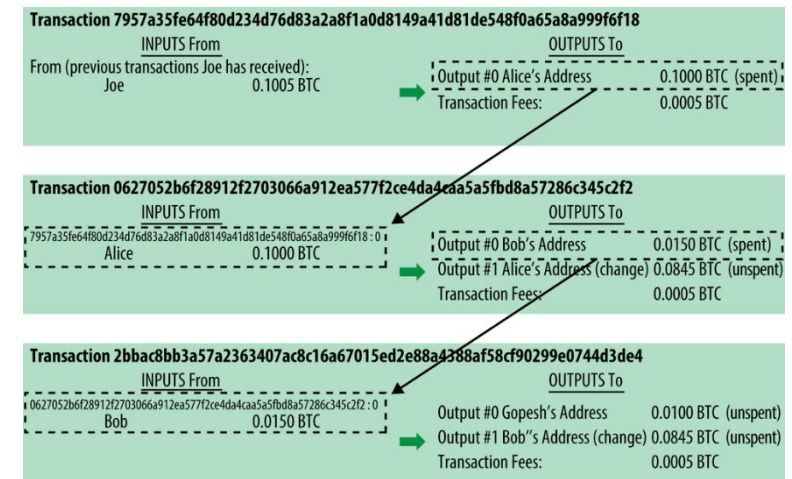
# P2P: Bitcoin

## ■ Transaktionen (Fortsetzung)

- Bsp. für Transaktion mit mehreren Inputs und Outputs:

Transaction as Double-Entry Bookkeeping			
Inputs	Value	Outputs	Value
Input 1	0.10 BTC	Output 1	0.10 BTC
Input 2	0.20 BTC	Output 2	0.20 BTC
Input 3	0.10 BTC	Output 3	0.20 BTC
Input 4	0.15 BTC		
Total Inputs:		Total Outputs:	0.50 BTC
	<i>Inputs</i> 0.55 BTC		
	<i>Outputs</i> 0.50 BTC		
	<i>Difference</i> 0.05 BTC (implied transaction fee)		

- Bsp. für Kette von Transaktionen:
  - in 2. Transaktion wird UTXO aus 1. Transaktion von Alice signiert (Unlocked, Spent)
  - dann wird Guthaben an Bitcoin-Adresse von Bob transferiert (Locked)



Quelle: A. M. Antonopoulos,  
Mastering Bitcoin, O'Reilly, 2014.

## P2P: Bitcoin

### ■ Blockchain

- Miner erstellen Blöcke mit Transaktionen (1 MB)
- Block-Header (80 Bytes) bestehen aus: Hash des Headers des letzten Blocks, Wurzel des Merkle-Baums mit Hash über alle Transaktionen im Block, Zeitstempel, **Difficulty** (Schwierigkeit), **Nonce** (Zufallszahl)
- **Proof-of-Work**: Berechnung von Hash (SHA-256) kleiner als Difficulty (führende Zahl von Nullen), Lösung durch Ausprobieren
- Wettbewerb der Miner
- wenn Block gefunden wird: Fluten, Peers verifizieren den Block und berechnen bei Akzeptanz nächsten Block, dadurch wächst die Blockchain, dies geschieht im Mittel alle 10 Minuten
- ergibt ca. 7 Transaktionen pro Sekunde
- mögliche Verzweigungen werden weitergeführt, bis sich eine längste Teilkette bildet, diese wird weiterverwendet, dadurch entscheidet Mehrheit der Rechenleistung über Fortsetzung (Konsensbildung)

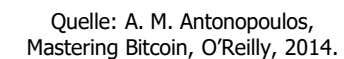
## ■ Blockchain (Fortsetzung)

## ■ Blockchain (Fortsetzung)

## Verkettung von Blockheadern, konzeptionell:



## Rechnerkommunikation, Anwendungsschicht



## P2P: Bitcoin

### ■ Blockchain (Fortsetzung)

- Rückverfolgung bis zum 1. Block ([Genesisblock](#)) vom 3.1.2009 möglich
- [Immutability](#) (Unveränderlichkeit): nachträgliche Änderung früherer Blöcke fast unmöglich, da in kurzer Zeit gültige nachfolgende Blöcke berechnet werden müssten
- [Irreversibilität](#) von in der Blockchain bestätigten Transaktionen
- nach 5 Blöcken gilt Transaktion als sicher
- Difficulty wird abhängig von Geschwindigkeit beim Mining angepasst
- [Incentive](#) durch 1. Transaktion im Block und Transaction Fees (mind. 1000 Satoshis)
- [Geldschöpfung](#) durch 1. Transaktion im Block, initial 50 BTC pro Block, Halbierung alle 210.000 Blöcke, ggw. (5.5.20) 12.5 BTC, nächste Halbierung ca. 12.5.20, begrenzt Bitcoin-Menge auf 21 Mio, wird voraussichtlich 2140 erreicht (i.w. 2030)
- Stand 5.5.20 (blockchain.com): Höhe der Blockchain: 629091, Größe der Blockchain: ca. 280 GB, Bitcoins im Umlauf: ca. 18,3 Mio

## P2P: Bitcoin

### ■ Kontroversen u.a.

- Angriffe: 51%, Quantencomputer gegen SHA-256 und elliptische Kurven-Kryptografie, ...
- Mining-Pools mit spezialisierter HW nur für Mining
- Strombedarf in Größenordnung eines kleinen Staats
- Skalierbarkeit (Visa mehrere Zehntausend Transaktionen pro Sekunde)
- Verwendung für illegale Zwecke

### ■ Weiterentwicklung der Blockchain-Technologie u.a.

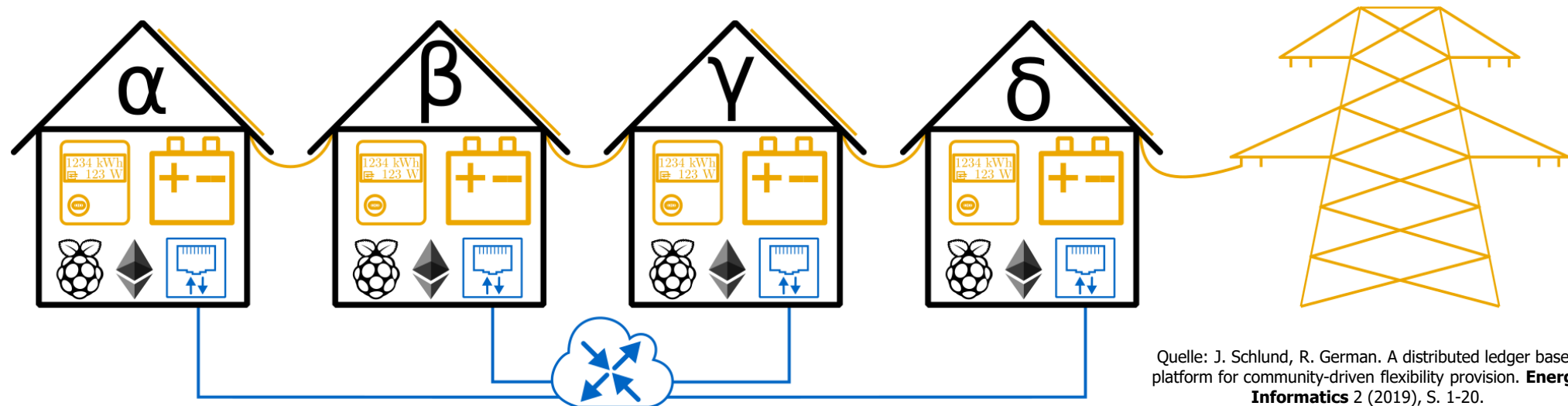
- Alt Coins: weitere Kryptowährungen basierend auf Forks der Bitcoin-Blockchain und alternativen Blockchains
- weitere Anwendungen außerhalb des Finanzbereichs
- Turing-mächtige Skriptsprache, Smart Contracts, z.B. Ethereum
- Proof-of-Stake: Konsensbildung basierend auf Vermögen
- Proof-of-Authority: Validierung durch vertrauenswürdige Instanzen
- Permissioned Blockchain: Zugangsbeschränkung, z.B. Hyperledger



## P2P: Blockchain bei Informatik 7

### ■ Blockchain-Technologie für Energie-Communities

- nachbarschaftliche Community mit Photovoltaik und Batteriespeichern
- Energieaustausch ermöglicht Steigerung von Eigenverbrauch, Autarkie und Senkung von externem Strombezug
- ETHome: Dezentrale Steuerung mit Smart Contracts über private Instanz einer Ethereum-Blockchain
- Versuchsaufbau (Blockchain auf Raspberry Pis, Kommunikation über TCP/IP, Häuser und energetische Aspekte simuliert):



Quelle: J. Schlund, R. German. A distributed ledger based platform for community-driven flexibility provision. **Energy Informatics** 2 (2019), S. 1-20.