

Systeme II

6. Die Anwendungsschicht

Christian Schindelhauer

Technische Fakultät

Rechnernetze und Telematik

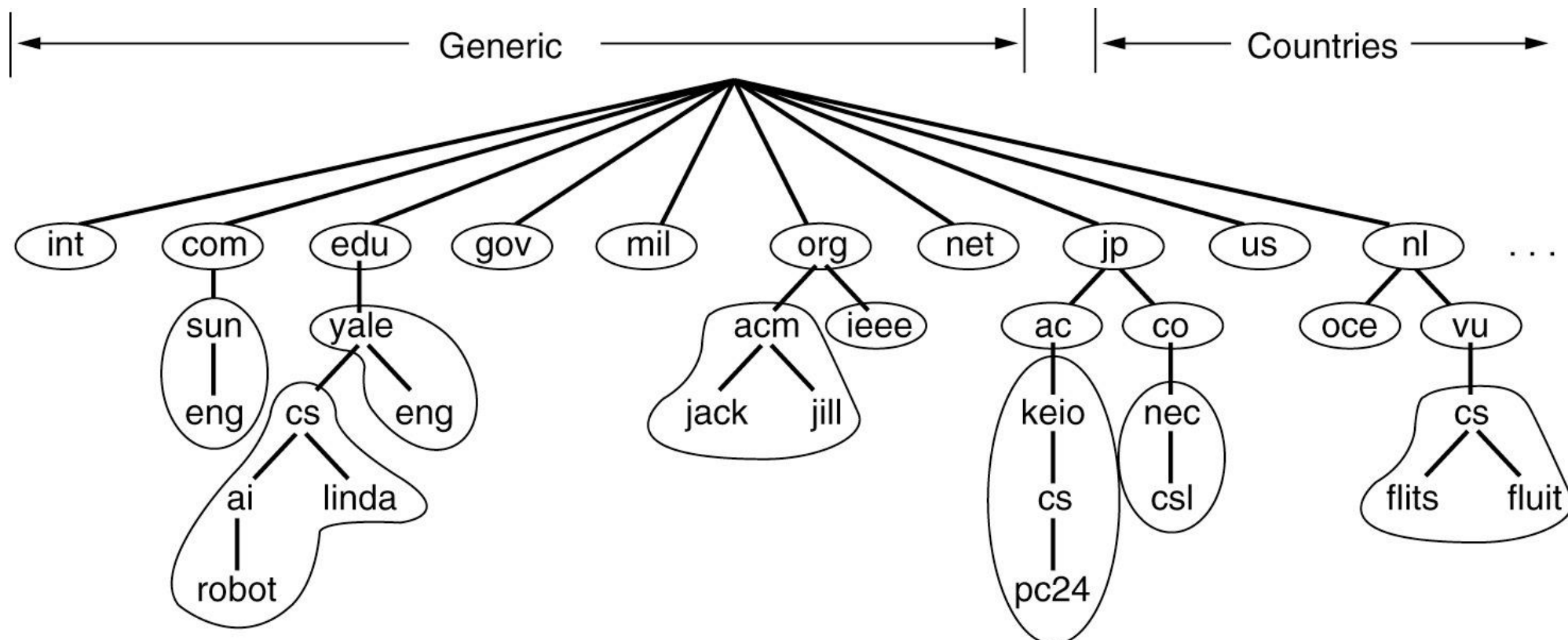
Albert-Ludwigs-Universität Freiburg

Version 06.07.2017

- Menschen kommen mit den 4-Byte IPv4-Adressen nicht zurecht:
 - 209.85.148.102 für Google
 - 132.230.2.100 für Uni Freiburg
 - Was bedeuten?
 - 77.87.229.75
 - 132.230.150.170
- Besser: Natürliche Wörter für IP-Adressen
 - Z.B. www.get-free-beer.de
 - oder www.uni-freiburg.de
- Das Domain Name System (DNS) übersetzt solche Adressen in IP-Adressen

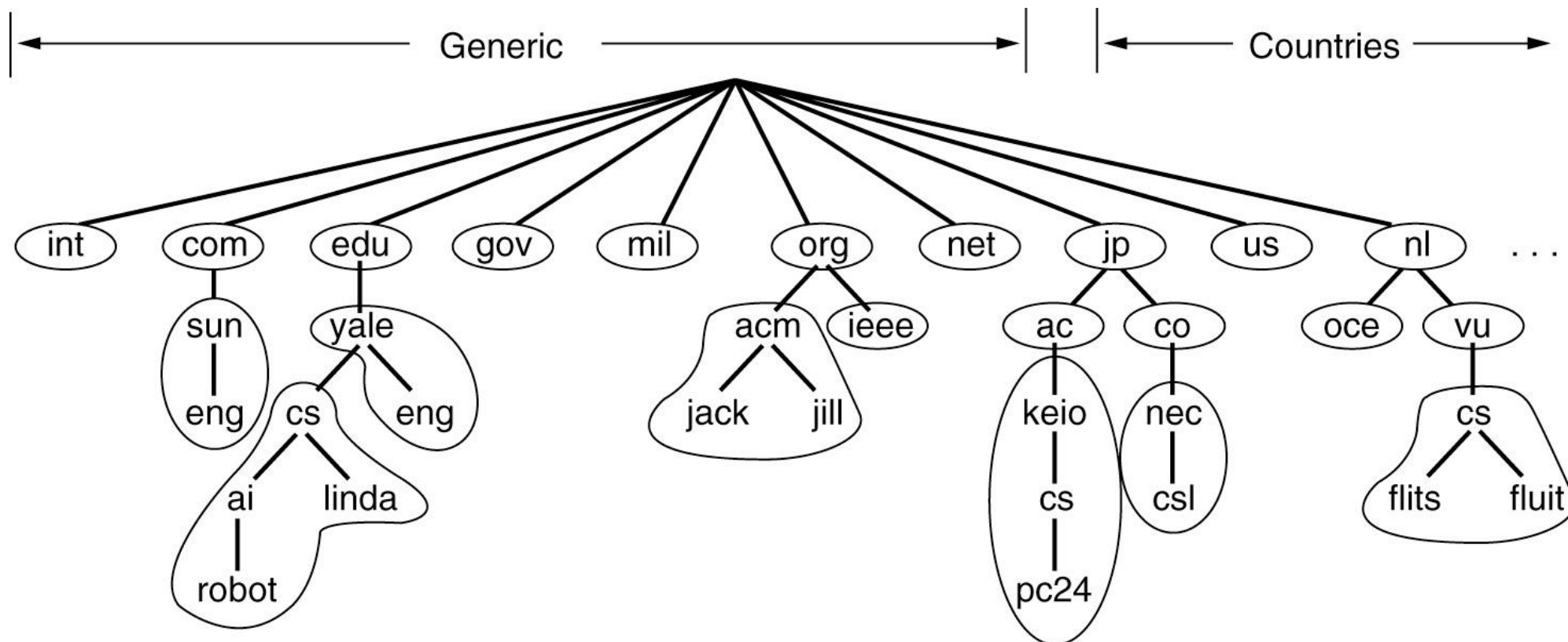
DNS – Architektur

- DNS bildet Namen auf Adressen ab
 - Eigentlich: Namen auf Ressourcen-Einträge
- Namen sind hierarchisch strukturiert in einen Namensraum
 - Max. 63 Zeichen pro Komponente, insgesamt 255 Zeichen
 - In jeder Domain kontrolliert der Domain-Besitzer den Namensraum darunter
- Die Abbildung geschieht durch Name-Server

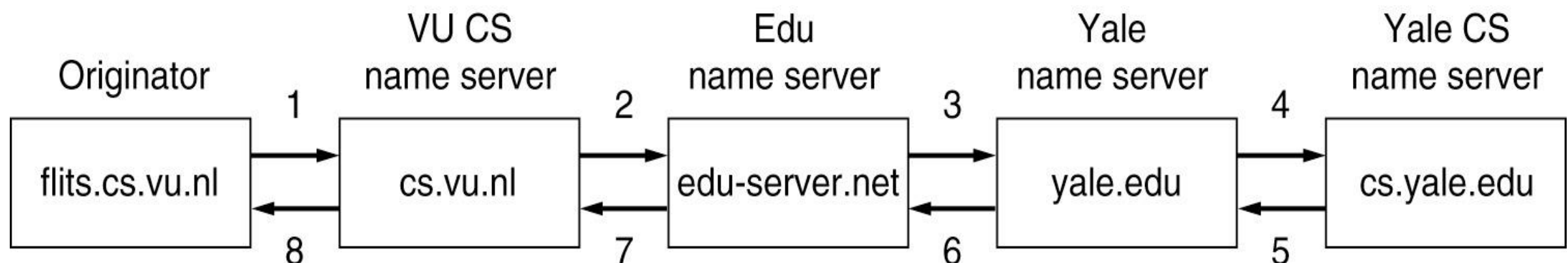


DNS Name Server

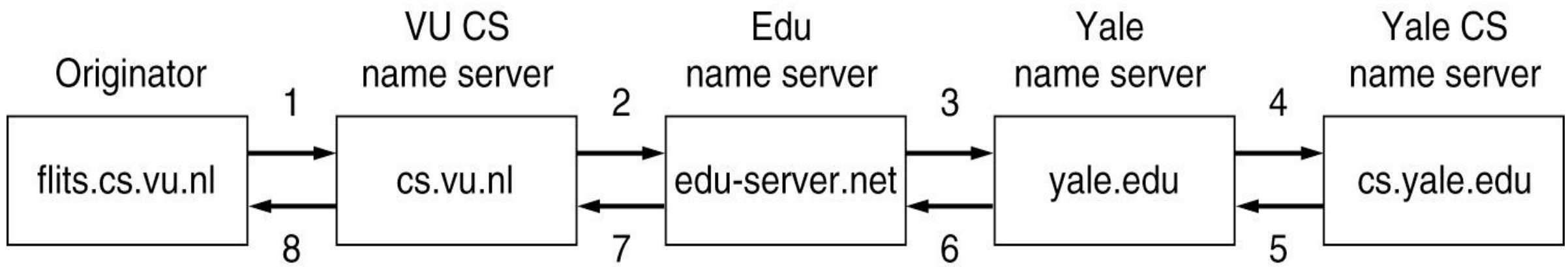
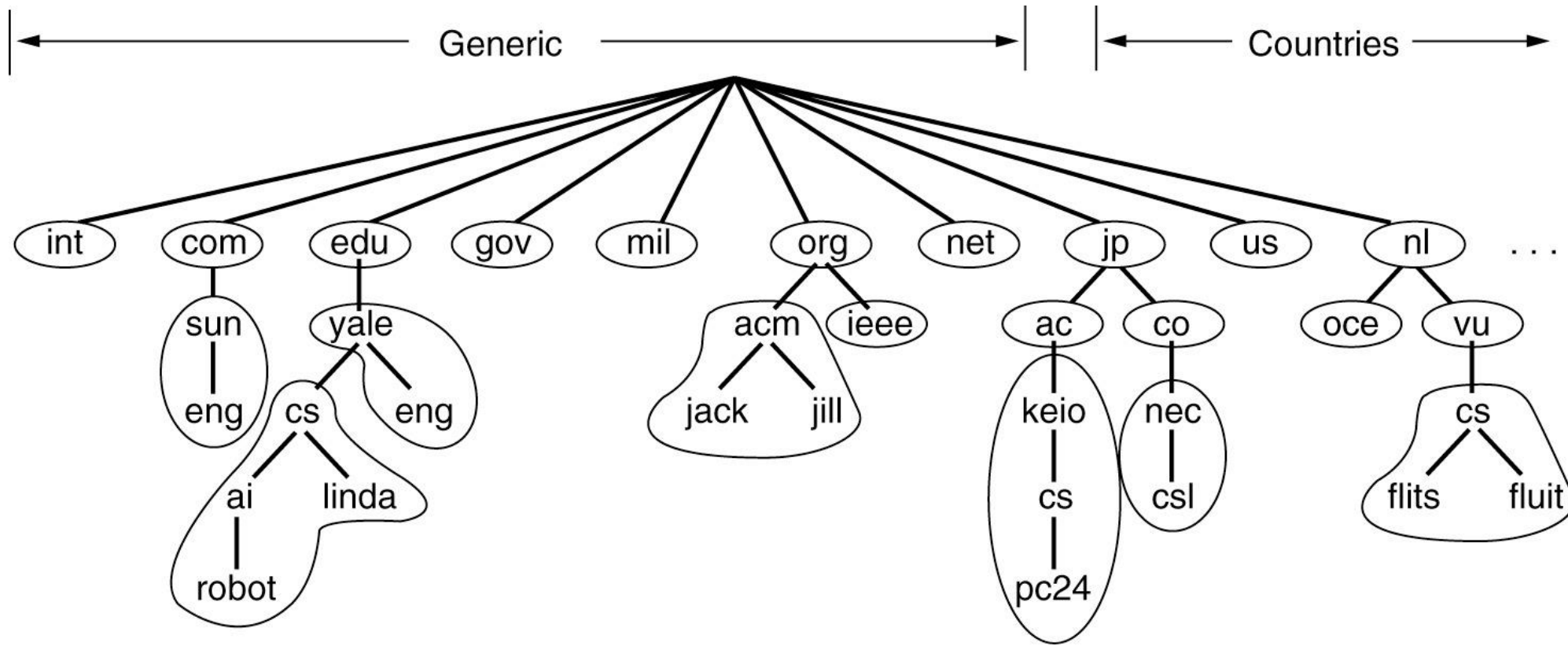
- Der Namensraum ist in Zonen aufgeteilt
- Jede Zone hat einen *Primary Name Server* mit maßgeblicher Information
 - Zusätzlich *Secondary Name Server* für Zuverlässigkeit
- Jeder Name Server kennt
 - seine eigene Zone
 - Name-Server der darunterliegenden Bereiche
 - Bruder-Name-Server oder zumindestens einen Server, der diese kennt



- Anfragen von einem End-System werden zu den vorkonfigurierten Name-Server geschickt
 - Soweit möglich, antwortet dieser Name-Server
 - Falls nicht, wird die Anfrage zu dem bestgeeigneten Name-Server weitergereicht
 - Die Antworten werden durch die Zwischen-Server zurückgeschickt
- Server darf Antworten speichern (cachen)
 - Aber nur für eine bestimmte Zeit

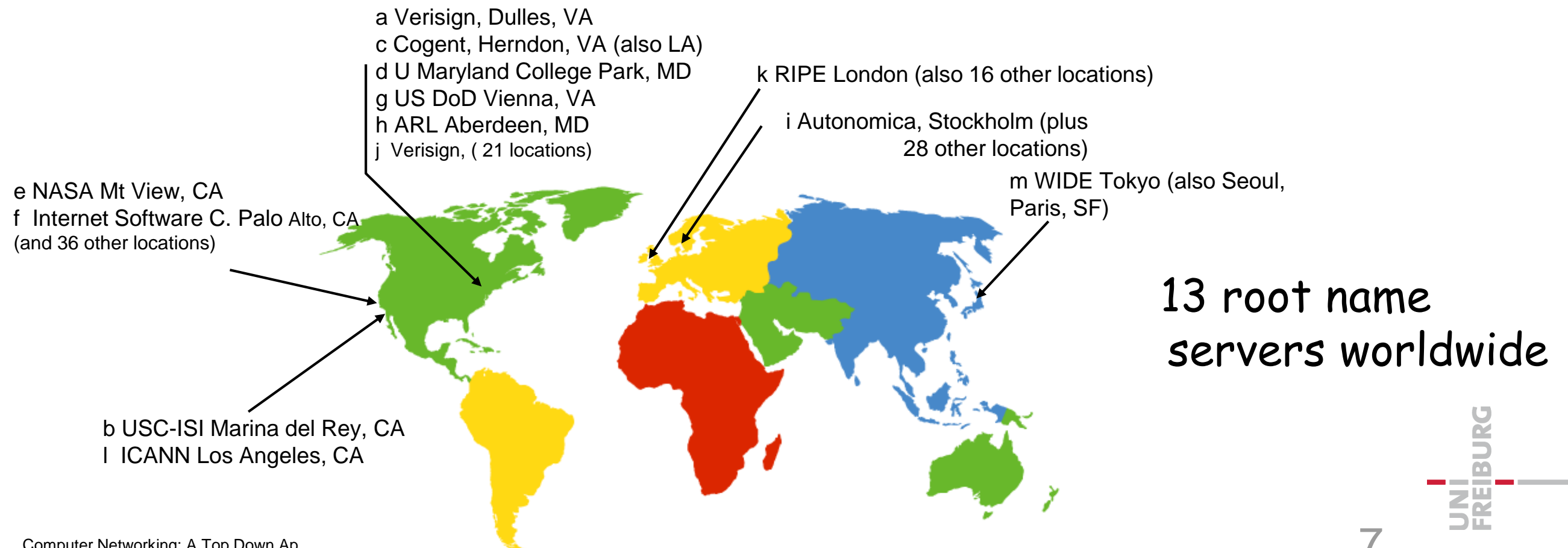


Beispiel



DNS Root Name Servers

- wird von lokalen Name-Server kontaktiert, wenn der Name nicht aufgelöst werden kann
- Root Name Server:
 - wird kontaktiert vom Name-Server falls die Zuordnung der Namen nicht bekannt ist.
 - erhält die Zuordnung
 - gibt die Zuordnung an den lokalen Name-Server weiter

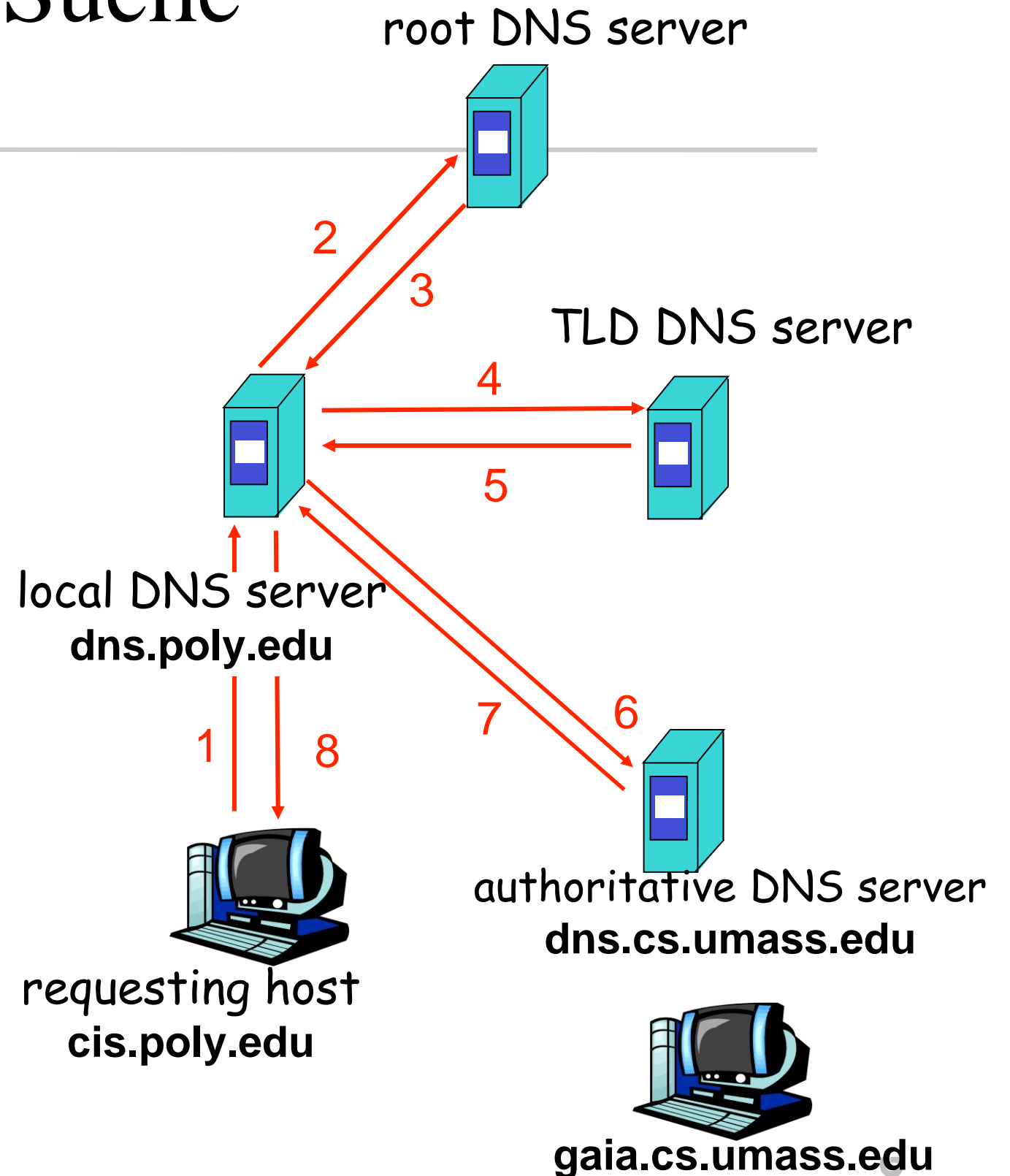


- Top-Level Domain (TLD) Server
 - verantwortlich für com, org, net, edu, etc, und alle Top-Level-Country-Domains uk, fr, ca, jp.
 - Network Solutions unterhält Server für *com* TLD
 - Educause für *edu* TLD
- Autorisierte DNS Servers:
 - DNS-Server von Organisationen
 - welche verantwortlich für die Zuordnung von IP-Adresse zu Hostnamen sind
 - können von den Organisationen oder Service-Provider unterhalten werden

- Jeder ISP hat einen lokalen Name-Server
 - Default Name Server
- Jede DNS-Anfrage wird zum lokalen Name-Server geschickt
 - fungiert als Proxy und leitet Anfragen in die Hierarchie weiter

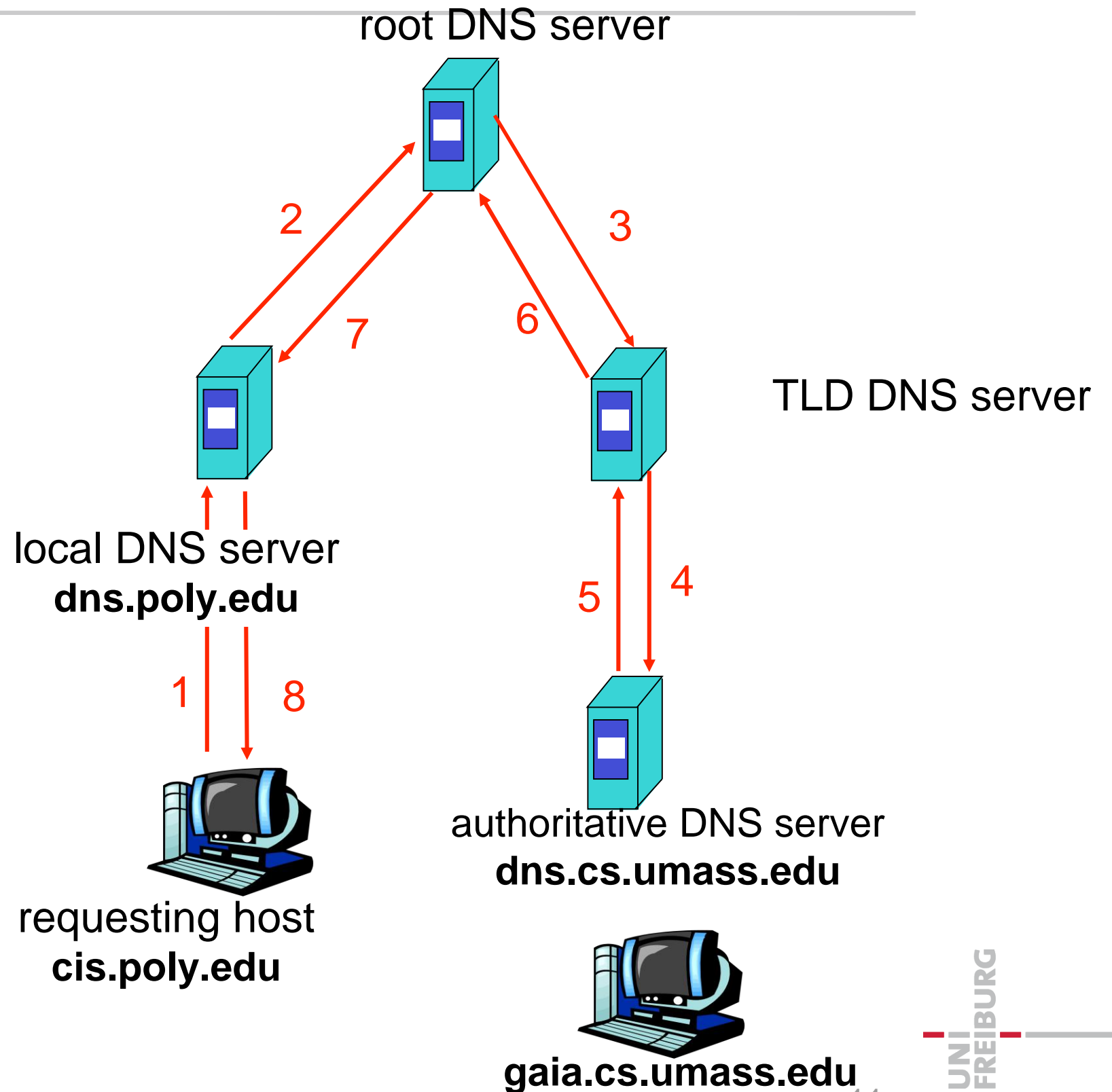
DNS Iterative Suche

- Rechner bei cis.poly.edu fragt nach IP address für gaia.cs.umass.edu
- Iterative Anfrage
 - Angefragte Server antworten
 - mit IP-Adresse
 - oder mit dem Namen des nächsten Servers
 - Lokaler DNS-Server ist selbst für Suche verantwortlich



DNS Rekursive Suche

- Jeder angefragte Server ist für die Namensauflösung zuständig
- Anfrage wird rekursiv weitergeleitet und dann zurück gegeben



- Sobald ein Name-Server einen Namen kennen lernt, speichert er die Zuordnung
 - Cache-Einträge haben einen Time-Out und werden nach einer gewissen Zeit gelöscht
 - TLD-Servers werden in lokalen Name-Servern gespeichert
 - Daher werden Root-Name-Server nicht oft besucht
- Update und Benachrichtungsmechanismus von IETF festgelegt
 - RFC 2136
 - <http://www.ietf.org/html.charters/dnsind-charter.html>

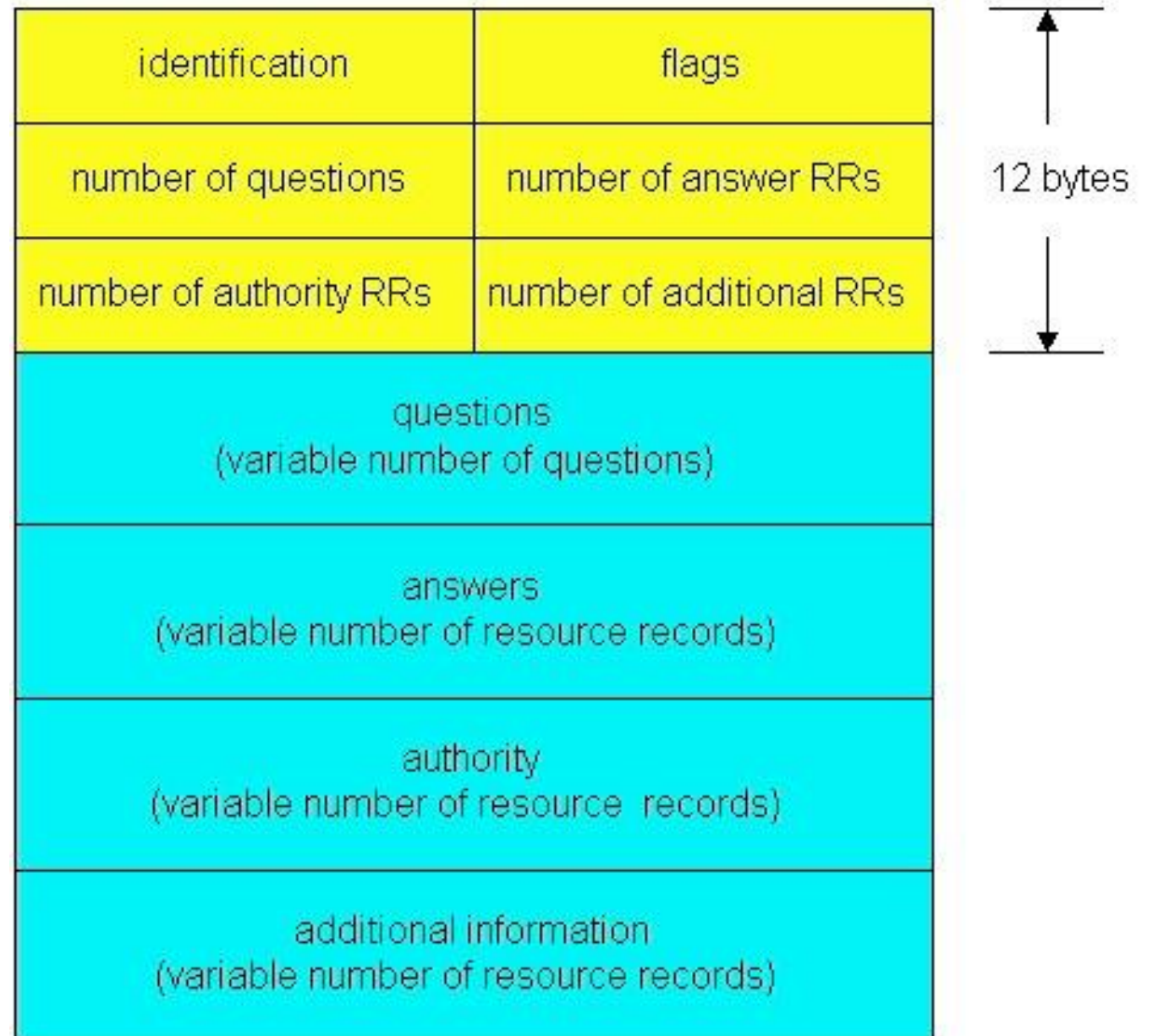
- DNS: verteilte Datenbank speichert Resource Records (RR)
- RR Format: (Name, Wert, Typ, TTL)
- Typ = A
 - Name = hostname
 - Wert = IP-Adresse
- Typ = NS
 - Name = domain (z.B. uni-freiburg.de)
 - Wert = hostname eines autorisierten Name-Servers für diese Domain
- Typ = CNAME
 - Name = Alias für einen „kanonischen“ (wirklichen) Namen
 - z.B. www.ibm.com ist in Wirklichkeit servereast.backup2.ibm.com
 - Wert ist kanonischer Name
- Typ = MX
 - Wert ist der Name des Mailservers

DNS Resource Record

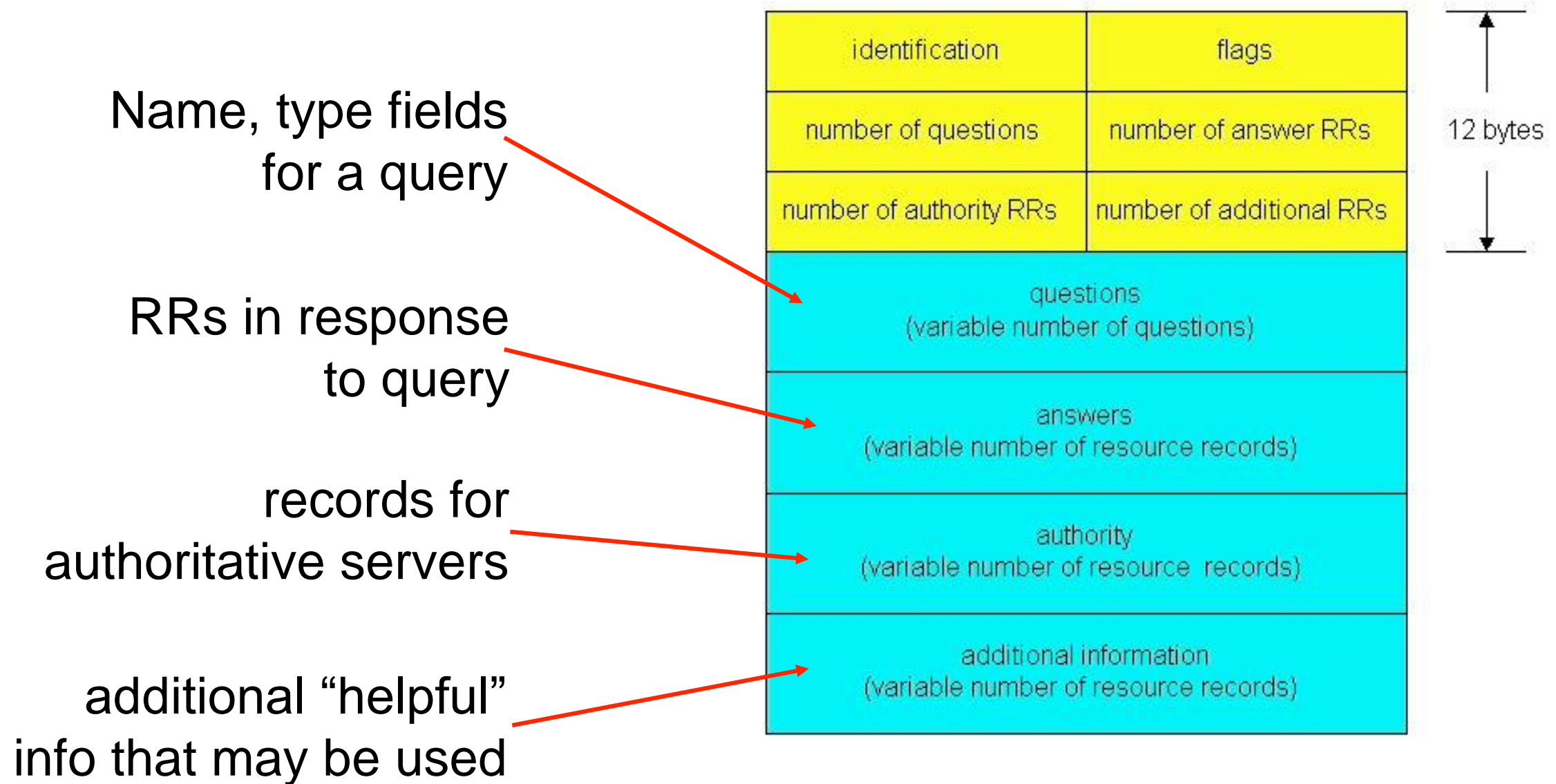
- Ressourcen-Einträge: Informationen über Domains, einzelne Hosts,...
- Inhalt:
 - Domain_name: Domain(s) des Eintrags
 - Time_to_live: Gültigkeit (in Sekunden)
 - Class: Im Internet immer "IN"
 - Type: Siehe Tabelle
 - Value: z.B. IP-Adresse

Type	Meaning	Value
SOA	Start of Authority	Parameters for this zone
A	IP address of a host	32-Bit integer
MX	Mail exchange	Priority, domain willing to accept e-mail
NS	Name Server	Name of a server for this domain
CNAME	Canonical name	Domain name
PTR	Pointer	Alias for an IP address
HINFO	Host description	CPU and OS in ASCII
TXT	Text	Uninterpreted ASCII text

- DNS-Protokoll
 - Anfrage und Antwort im selben Format
- Nachrichten-Header
 - ID, 16 Bit für Anzahl der Anfragen, Anzahl der Antworten, ...
- Flags:
 - Query oder Reply
 - Rekursion gewünscht
 - Rekursion verfügbar
 - Antwort ist autorisiert



DNS-Protokoll und Nachrichten



- Problem
 - Zeitlich zugewiesene IP-Adressen
 - z.B. durch DHCP
- Dynamisches DNS
 - Sobald ein Knoten eine neue IP-Adresse erhält, registriert dieser diese beim DNS-Server, der für diesen Namen zuständig ist
 - Kurze time-to-live-Einträge sorgen für eine zeitnahe Anpassung
 - da sonst bei Abwesenheit die Anfragen an falsche Rechner weitergeleitet werden
- Anwendung
 - Registrierung einer Domain für den Otto Normalverbraucher
 - Siehe www.dyndns.com

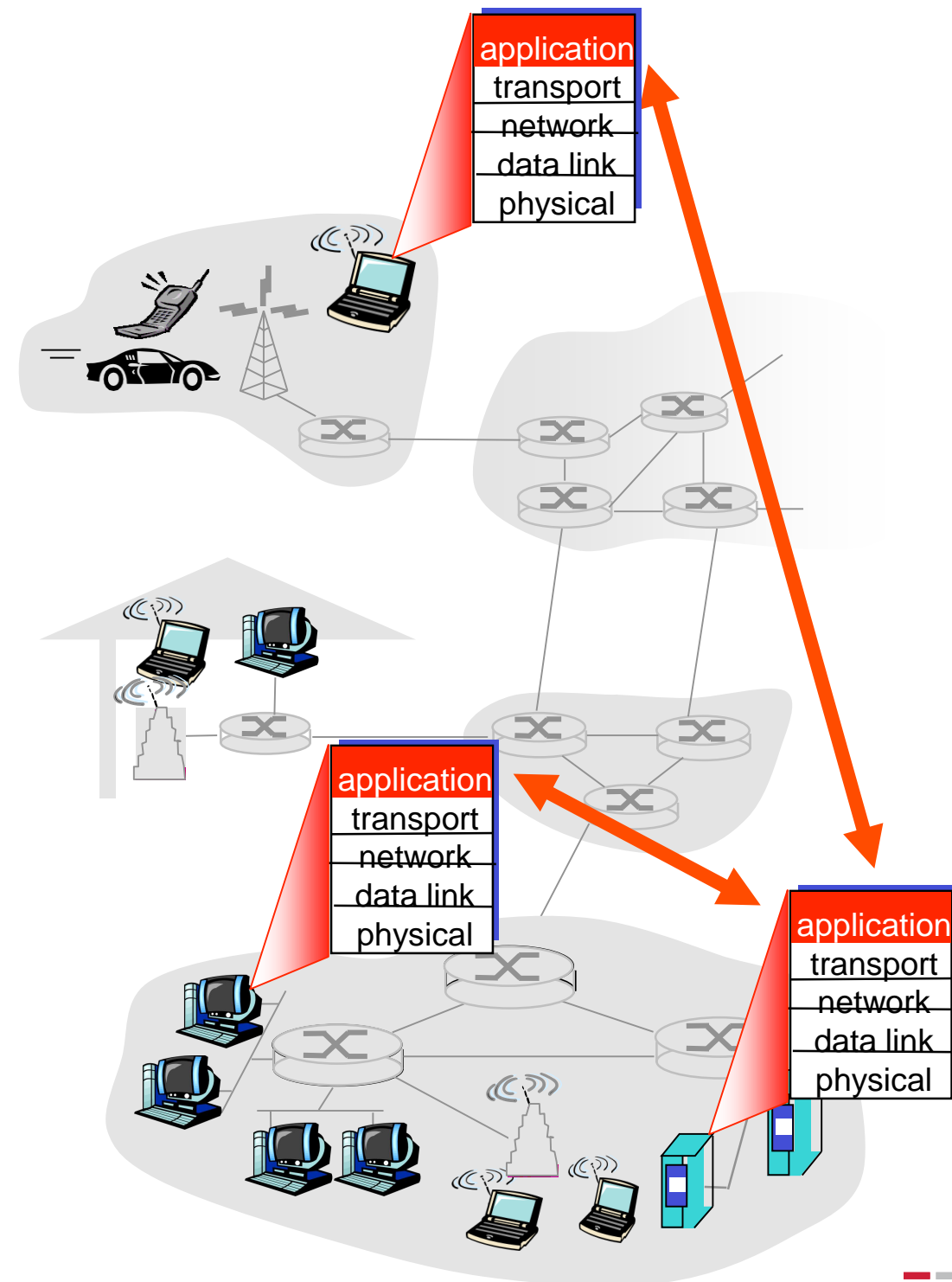
- Cache Poisoning
 - Falsche Einträge werden in DNS-Server eingebracht
 - weitergeleitete Einträge werden gecacht und können zu falschen Auskünften führen
- DNSSEC
 - implementiert seit 2010
 - Zuständiger Master-Server unterschreibt seine Einträge digital (mit Hilfe eines Public-Key-Kryptosystems)
 - Ursprüngliche Information bleibt unverschlüsselt
- Schlüsselmanagement
 - Gegenseitiges unterschreiben der Public-Keys
 - Aufwand wird gemildert durch „Chain of Trust“
 - Hierarchische Kette von unterschriebenen Schlüsseln
- Diskussion
 - aufwändigere DNS-Antworten
 - Sicherheitslücken bleiben bestehen

- Aspekte der Programmierung im Internet aus der Sicht der Anwendung
- Anforderungen an die Transportschicht
- Client-Server-Prinzip
- Peer-to-Peer-Prinzip
- Beispiel-Protokolle:
 - HTTP
 - SMTP / POP3 / IMAP
 - DNS
- Programmierung von Netzwerk-Anwendungen

- E-Mail
- Web
- Instant messaging
- Remote Login
- P2P File Sharing
- Multi-User Network Games
- Video Streaming
- Social Networks
- Voice over IP
- Real-time Video Konferenz
- Grid Computing

Erstellen einer Netzwerk-Anwendung

- Programme laufen auf den End-Punkten
 - kommunizieren über das Netzwerk
 - z.B. Web-Client kommuniziert durch Browser-Software
- Netzwerk-Router
 - werden nicht programmiert!
 - nicht für den Benutzer verfügbar
- Dadurch schnelle Programm-Entwicklung möglich
 - gleiche Umgebung
 - schnelle Verbreitung



- Client-server
 - beinhaltet auch Data Centers & Cloud Computing
- Peer-to-peer (P2P)
- Hybride Verbindung von Client-Server und P2P

Client-Server-Architektur

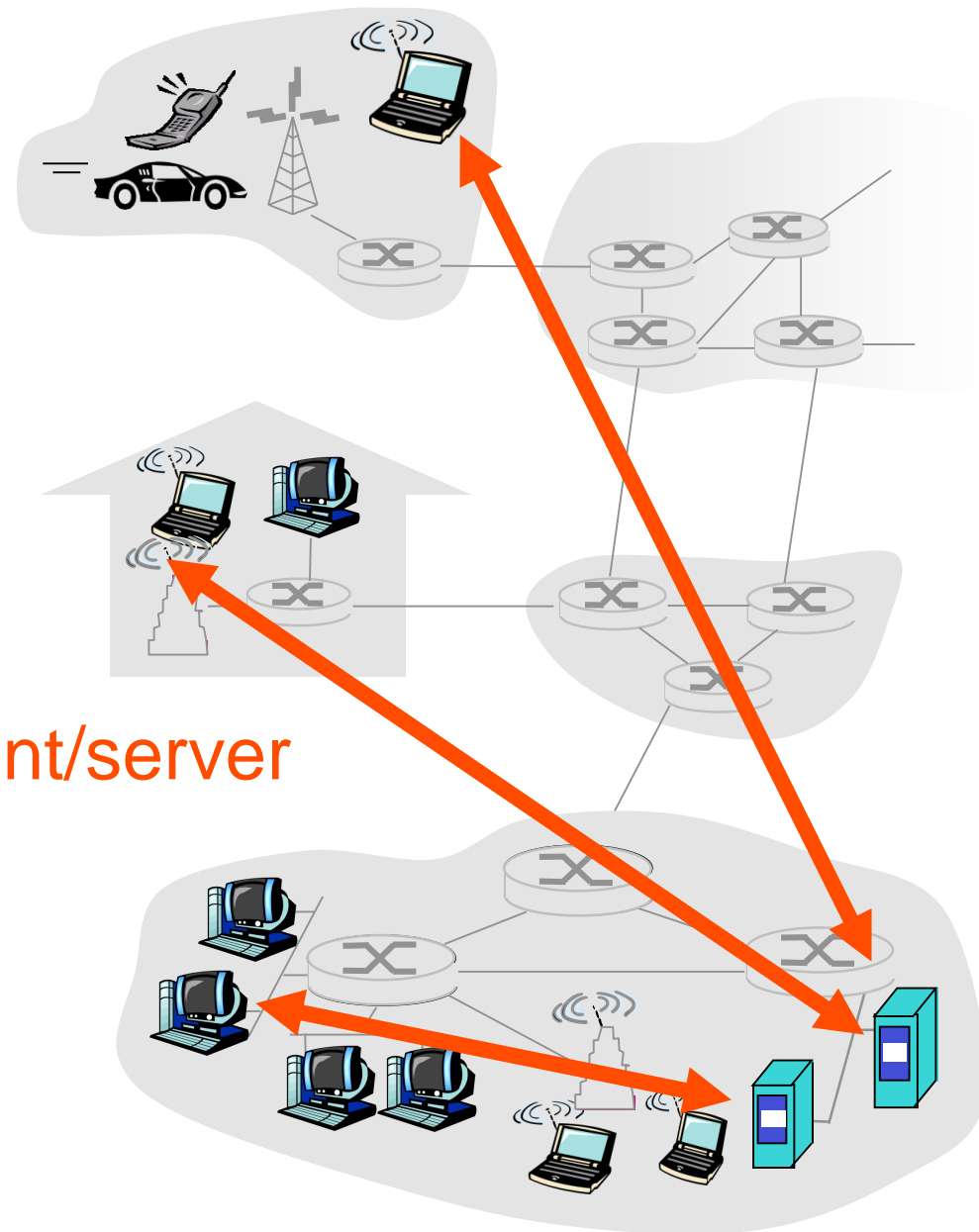
■ Server

- allzeit verfügbarer Host
- permanente IP-Address
 - oder per DNS ansprechbar
- Server-Farms wegen Skalierung

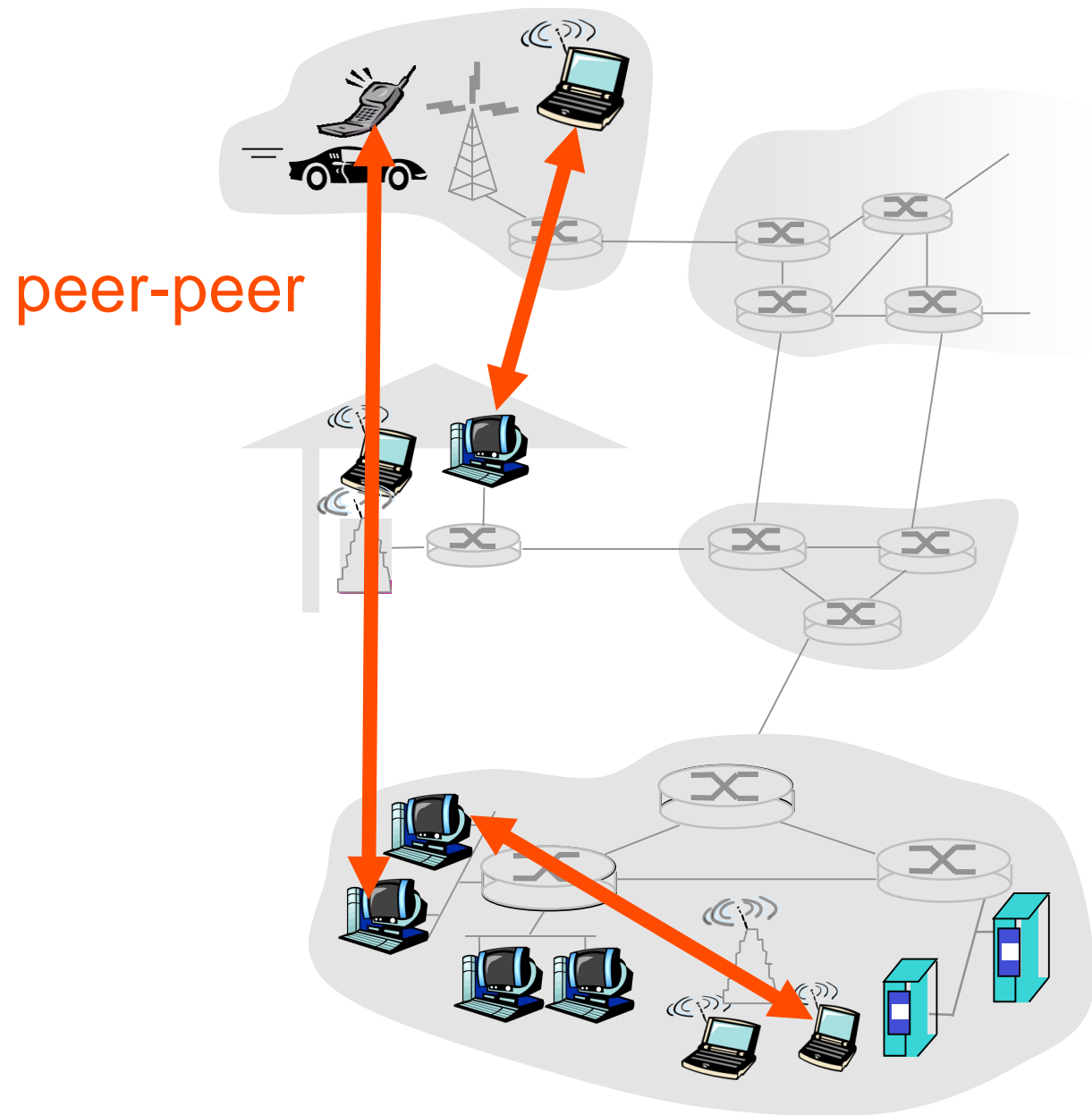
■ Client

- kommuniziert mit dem Server
- möglicherweise nicht durchgängig verbunden
- evtl. dynamische IP-Adresse
- Clients kommunizieren **nicht** miteinander

client/server



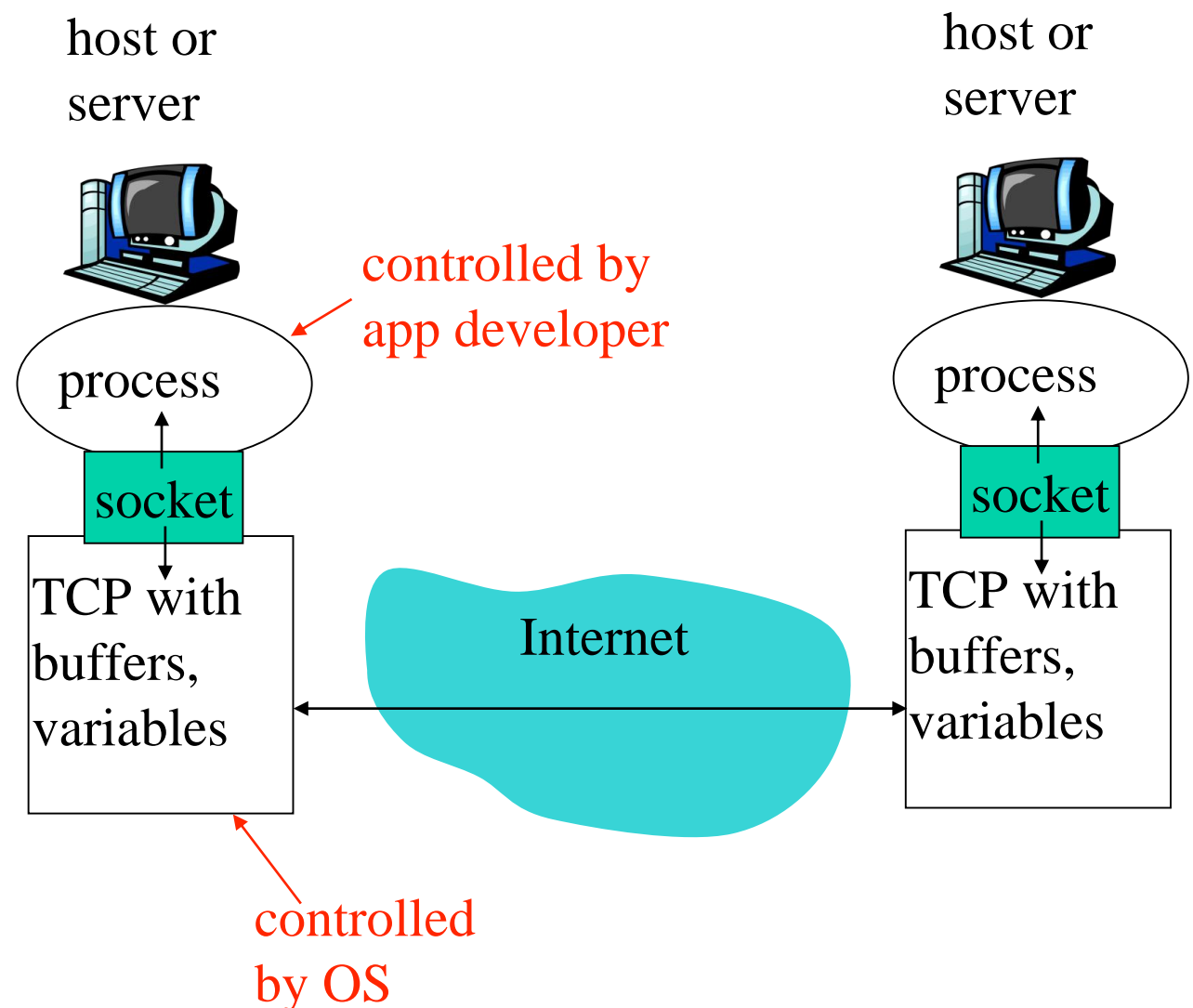
- Ohne Server
- End-Systeme kommunizieren direkt
- Peers
 - sind nur zeitlich begrenzt online
 - verändern von Zeit zu Zeit ihre IP-Adresse
- Hochskalierbar, aber schwer zu handhaben



- z.B. Skype
 - Voice-over-IP P2P
 - Server für Anmeldung und Verzeichnis
 - Telefonie und Video-Verbindung Direktverbindung
- Instant Messaging
 - Chat zwischen zwei Benutzern ist P2P
 - Zentraler Service:
 - Client-Anwesenheit
 - Suche und Zuordnung der IP-Adresse
 - Benutzer registrieren die IP-Adresse, sobald online
 - Benutzer fragen beim Server nach IP-Adresse der Partner

- Prozess: Programm auf einem Rechner (Host)
 - innerhalb des selben Rechners kommunizieren Prozesse durch Inter-Prozess-Kommunikation
 - über OS
- Prozesse in verschiedenen Rechnern
 - kommunizieren durch Nachrichten
- Client-Prozess
 - Initiiert die Kommunikation
- Server-Prozess
 - wartet auf Client-Kontakt
- P2P
 - haben Client und Server-Prozesse

- Prozesse senden und empfangen Nachrichten über Sockets (Steckdosen)
- Sockets mit Türen vergleichbar
- Sender-Prozess
 - schiebt die Nachricht zur Tür hinaus
 - vertraut auf die Transport-Infrastruktur, dass die eine Seite der Tür mit der anderen verbindet
- API
 - Wahl des Transport-Protokolls
 - kann bestimmte Parameter wählen



- Nachrichtentyp
 - z.B. Request, Response
- Nachrichten-Syntax
 - Nachrichtfelder und Zuordnung
- Nachrichten-Semantik
 - Bedeutung der Felder
- Regeln für das Senden und Empfangen von Nachrichten
- Public-domain Protokolle
 - definiert in RFC
 - für Kompatibilität
 - z.B. HTTP, SMTP, BitTorrent
- Proprietäre Protokolle
 - z.B. Skype, ppstream

Welchen Transport-Service braucht eine Anwendung?

- **Datenverlust**
 - einige Anwendungen (z.B. Audio) tolerieren gewissen Verlust
 - andere (z.B. Dateitransfer, Telnet) benötigen 100% verlässlichen Datentransport
- **Timing**
 - einige Anwendungen (z.B. Internet Telefonie, Spiele) brauchen geringen Delay
- **Durchsatz (throughput)**
 - einige Anwendungen (z.B. Multimedia) brauchen Mindestdurchsatz
 - andere (“elastische Anwendungen”) passen sich dem Durchsatz an
- **Sicherheit**
- **Verschlüsselung, Datenintegrität**

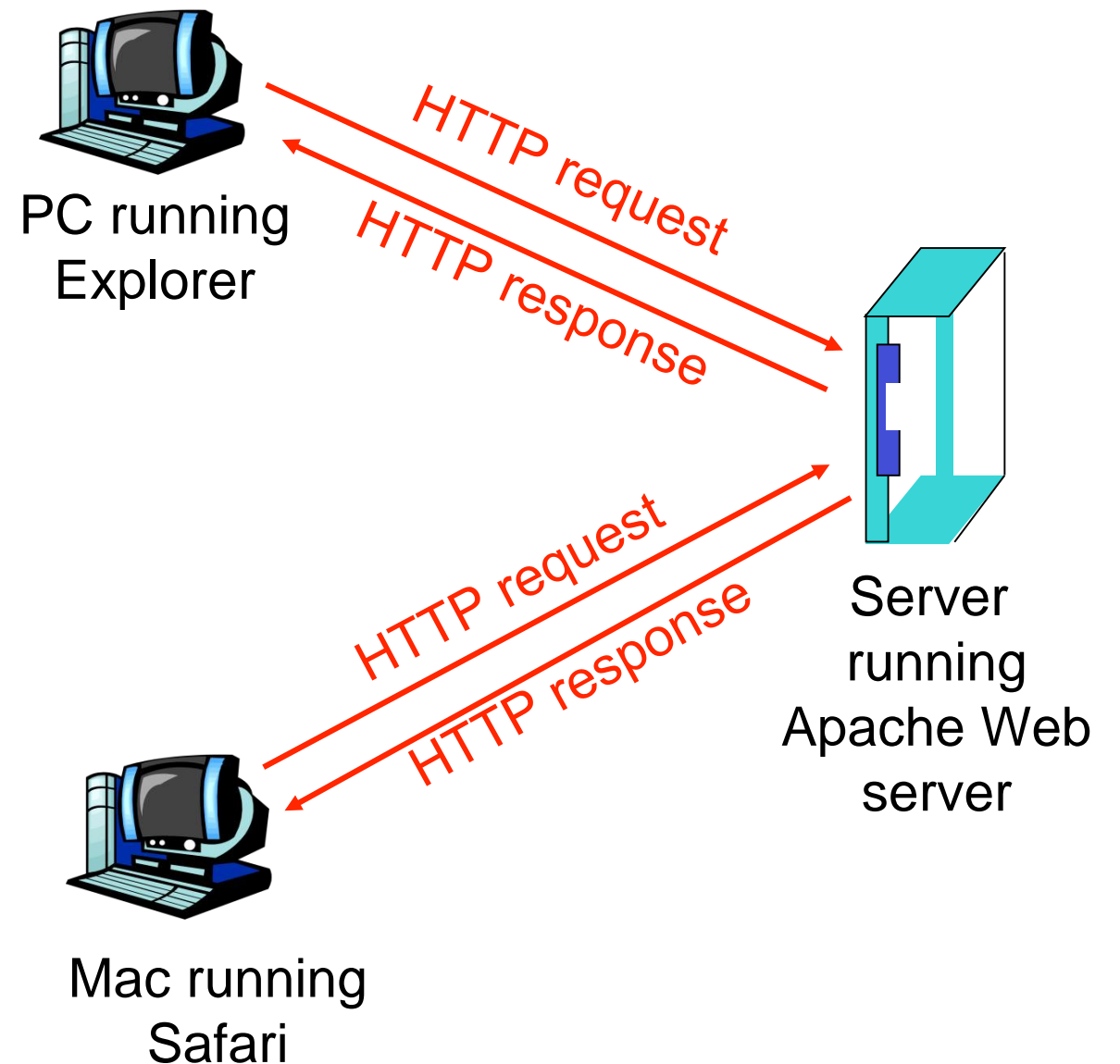
- Web-Seiten (web page) besteht aus Objekten
- Objekte sind HTML-Datei, JPEG-Bild, Java-Applet, Audio-Datei,...
- Web-Seite besteht aus Base HTML-Datei mit einigen referenzierten Objekten
- Jedes Objekt wird durch eine URL adressiert
 - Beispiel URL:

www.someschool.edu/someDept/pic.gif

host name

path name

- HTTP: Hypertext Transfer Protocol
 - Anwendungsschicht-Protokoll des Webs
- Client/Server-Modell
 - Client
 - Browser fragt an
 - erhält und zeigt Web-Objekte an
 - Server
 - Web-Server sendet Objekte als Antwort der Anfrage

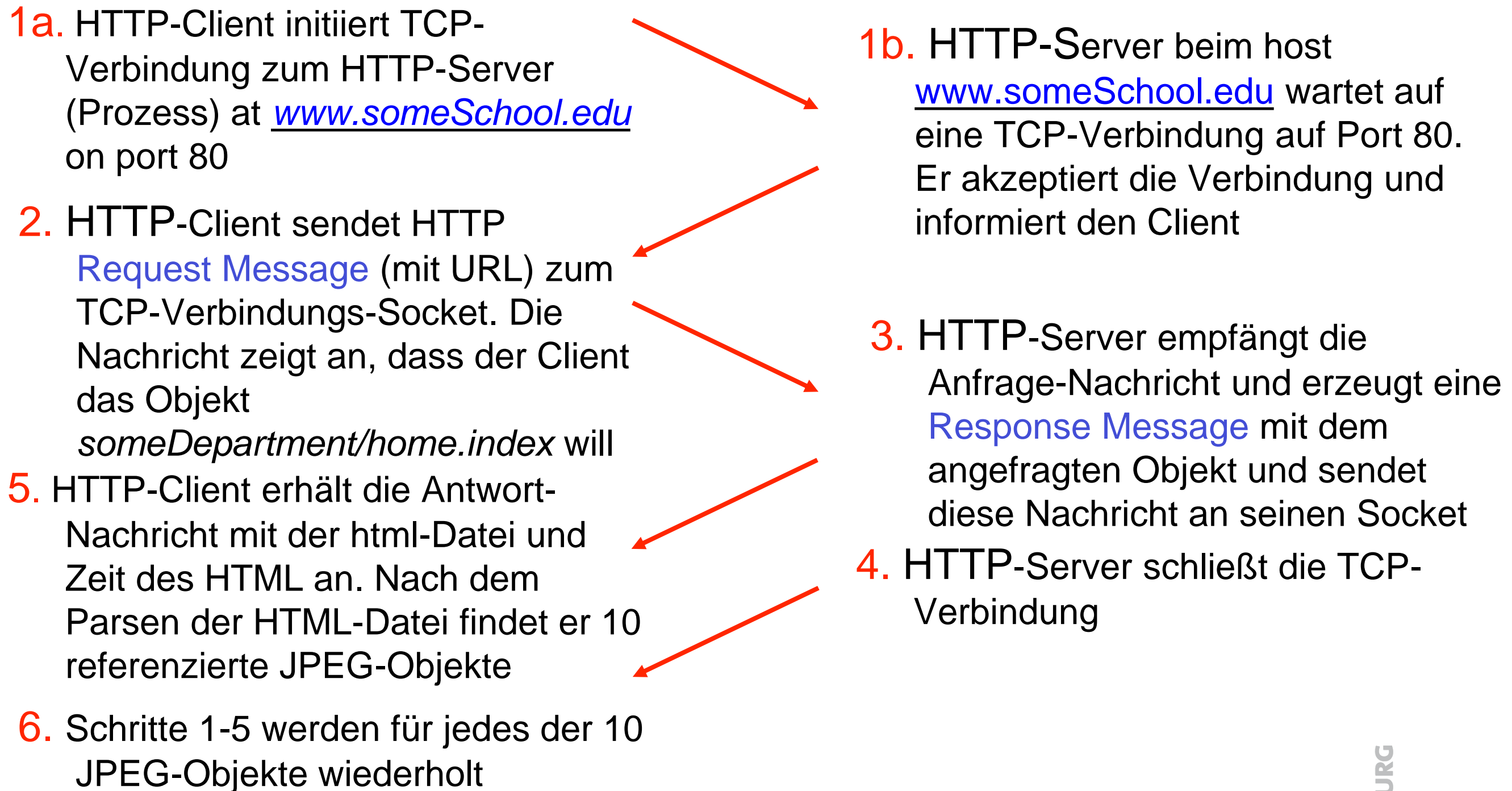


- Verwendet TCP
- Client initiiert TCP-Verbindung
 - erzeugt Socket zum Server auf Port 80
- Server akzeptiert TCP-Verbindung vom Client
- HTTP-Nachrichten
 - zwischen HTTP-Client und HTTP-Server
 - Anwendungsschicht-Protokoll-Nachrichten
- TCP-Verbindung wird geschlossen

- HTTP ist zustandslos (stateless)
 - Server merkt sich nichts über vorige Anfragen
- Warum?
 - Protokolle mit Zuständen sind komplex
 - Zustände müssen gemerkt und zugeordnet werden
 - falls Server oder Client abstürzen, müssen die möglicherweise inkonsistenten Zustände wieder angepasst werden

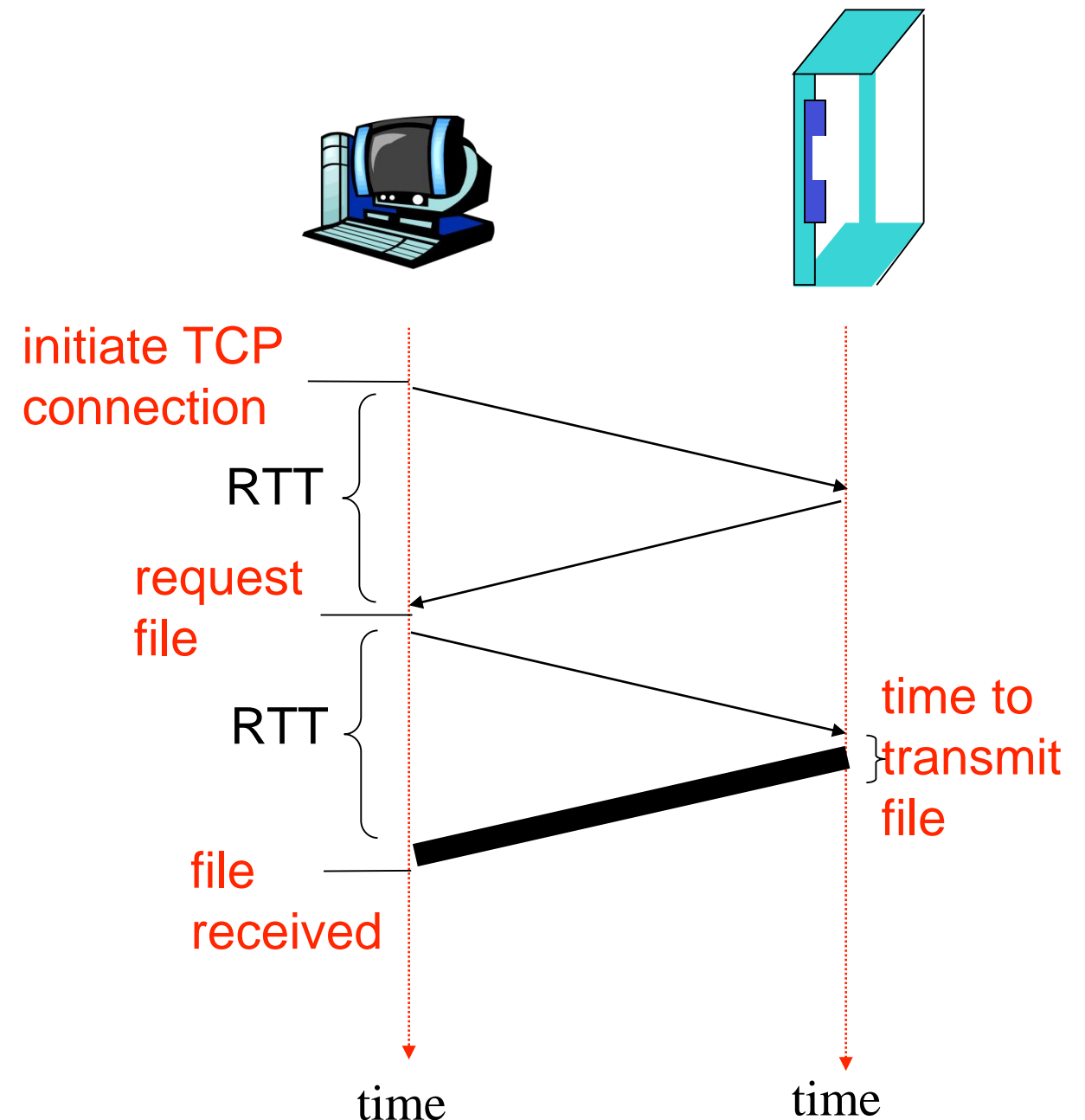
- Abbrechende (nicht persistente) HTTP-Verbindung
 - Höchstens ein Objekt wird über eine TCP-Verbindung gesendet
- Weiter bestehende (persistente) HTTP
 - Verschiedene Objekte können über eine bestehende TCP-Verbindung zwischen Client und Server gesendet werden

Nicht-Persistente HTTP-Verbindung

- 
- The diagram illustrates the sequence of steps for a non-persistent HTTP connection. It consists of two columns of text, each containing a numbered step. Red arrows connect the steps between the two columns to show the flow of the process. The steps are as follows:
- 1a.** HTTP-Client initiiert TCP-Verbindung zum HTTP-Server (Prozess) at www.someSchool.edu on port 80
 - 1b.** HTTP-Server beim host www.someSchool.edu wartet auf eine TCP-Verbindung auf Port 80. Er akzeptiert die Verbindung und informiert den Client
 - 2.** HTTP-Client sendet HTTP **Request Message** (mit URL) zum TCP-Verbindungs-Socket. Die Nachricht zeigt an, dass der Client das Objekt *someDepartment/home.index* will
 - 3.** HTTP-Server empfängt die Anfrage-Nachricht und erzeugt eine **Response Message** mit dem angefragten Objekt und sendet diese Nachricht an seinen Socket
 - 5.** HTTP-Client erhält die Antwort-Nachricht mit der html-Datei und Zeit des HTML an. Nach dem Parsen der HTML-Datei findet er 10 referenzierte JPEG-Objekte
 - 4.** HTTP-Server schließt die TCP-Verbindung
 - 6.** Schritte 1-5 werden für jedes der 10 JPEG-Objekte wiederholt

Nicht-persistentes HTTP: Antwortzeit

- Umlaufzeit (RTT – Round Trip Time)
 - Zeit für ein Packet von Client zum Server und wieder zurück
- Antwortzeit (Response Time)
 - eine RTT um TCP-Verbindung zu initiieren
 - eine RTT für HTTP Anfrage und die ersten Bytes des HTTP-Pakets
 - Transmit Time: Zeit für Dateiübertragung
- $\text{Zeit} = 2 \text{ RTT} + \text{transmit time}$



- Nicht-persistentes HTTP
 - benötigt 2 RTTs pro Objekt
 - Betriebssystem-Overhead für jede TCP-Verbindung
 - Browser öffnet oft TCP-Verbindungen parallel um referenzierte Objekte zu laden
- Persistentes HTTP
 - Server lässt die Verbindung nach der Antwortnachricht offen
 - Folgende HTTP-Nachrichten zwischen den gleichen Client/Server werden über die geöffnete Verbindung versandt
 - Client sendet Anfragen, sobald es ein referiertes Objekt findet
 - höchstens eine Umlaufzeit (RTT) für alle referenzierten Objekte

HTTP-Request Nachricht

- Zwei Typen der HTTP-Nachricht: request, response
- HTTP-Request Nachricht:
 - ASCII (human-readable format)

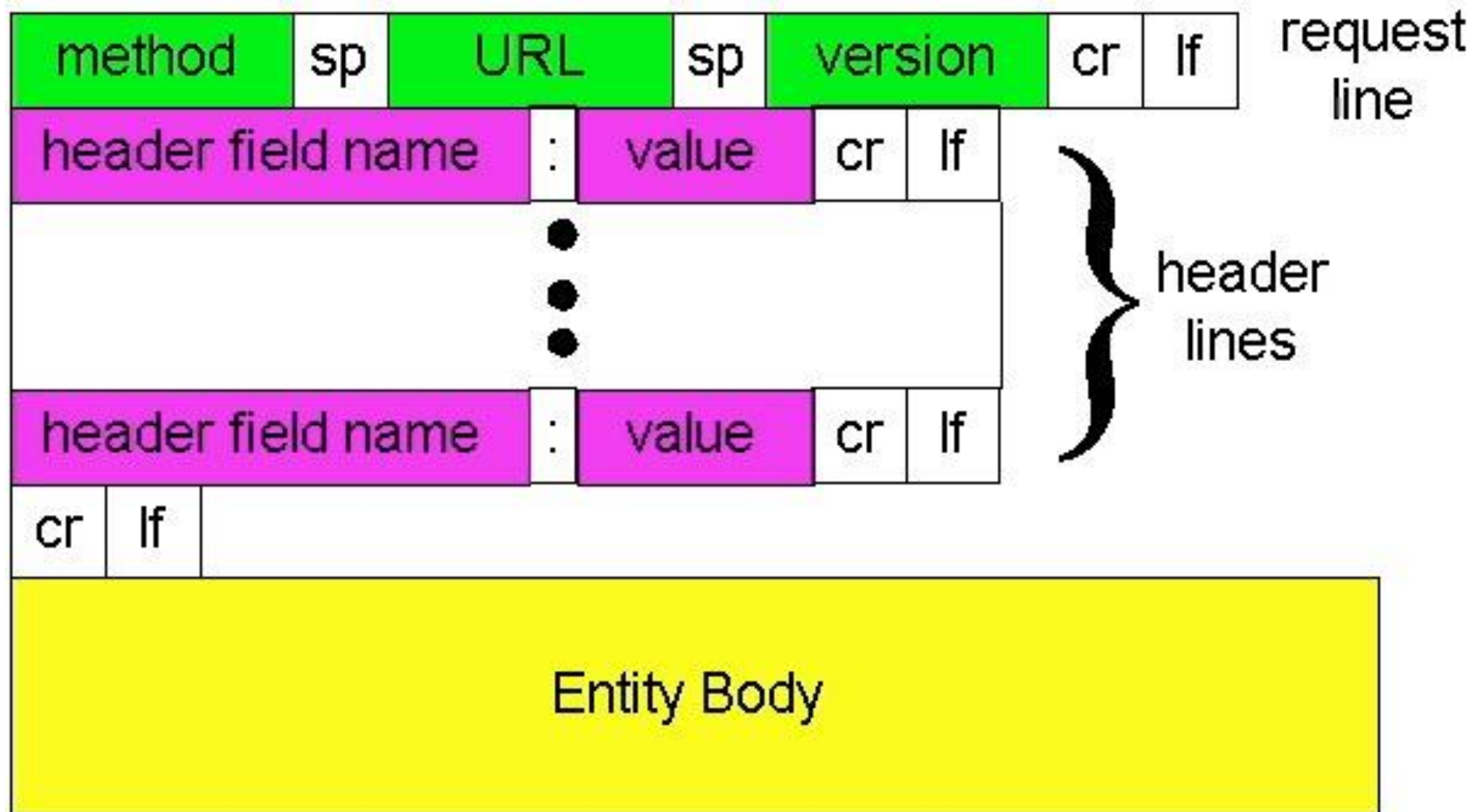
Request Zeile
(GET, POST,
HEAD Befehle)

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

Extra Zeilenschaltung
zeigt das Ende der
Nachricht an

(extra carriage return, line feed)

HTTP-Request Nachricht: Allgemeines Format

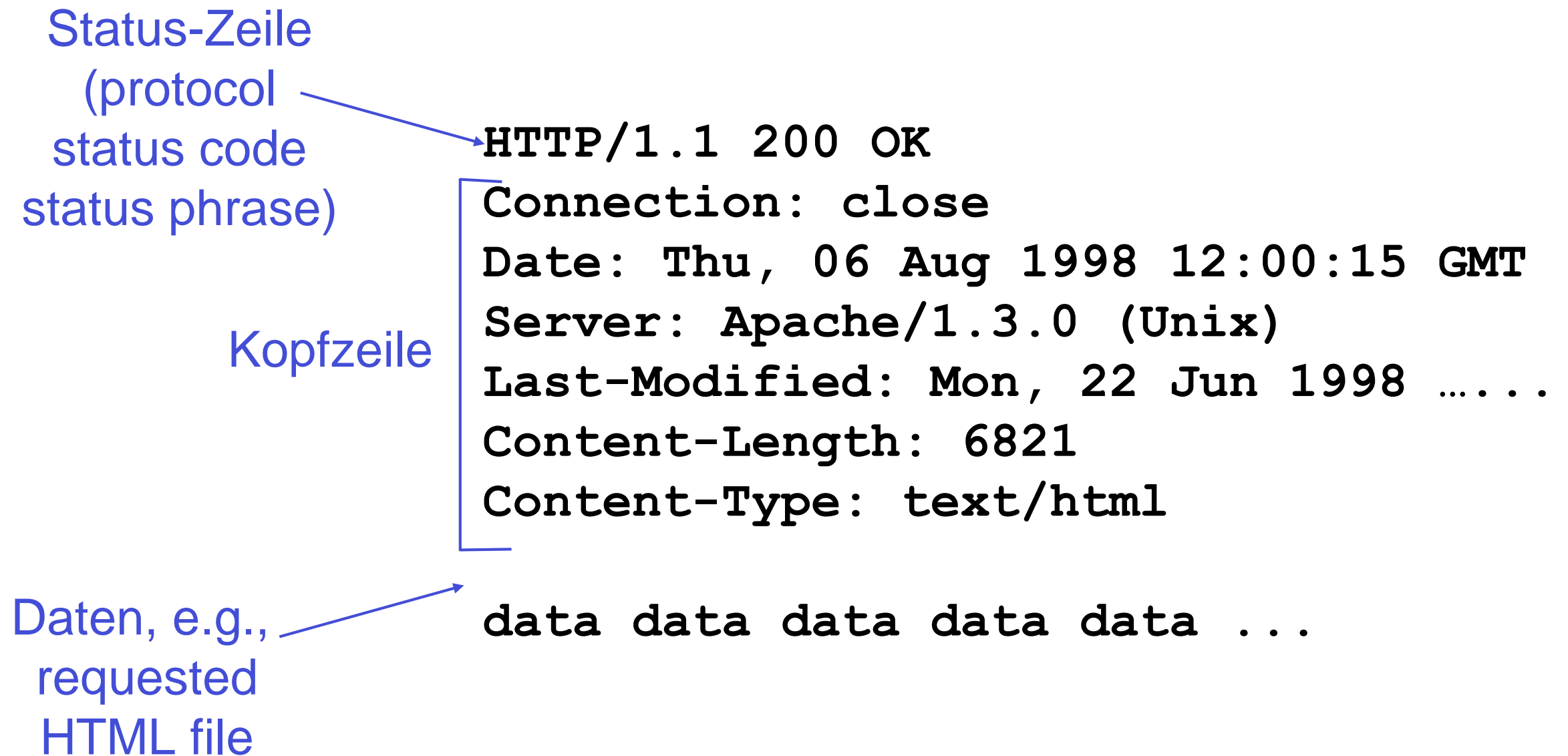


- Post
 - Web-Seiten haben öfters Leerfelder für Eingaben
 - Eingabe wird im *Body* zum Server hochgeladen
- URL-Methode
 - Verwendet GET-Methode
 - Input wird im URL-Feld der Anfrage-Nachricht gesendet:

`www.somesite.com/animalsearch?monkeys&banana`

- HTTP/1.0
 - GET
 - POST
 - HEAD
 - fragt den Server nur nach dem Head, nicht nach dem Inhalt (*body*)
- HTTP/1.1
 - GET, POST, HEAD
 - PUT
 - lädt eine Datei im *body*-Feld zum Pfad hoch, der im URL-Feld spezifiziert wurde
 - DELETE
 - löscht Datei, die im URL-Feld angegeben wurde

HTTP-Antwort Nachricht



1. Telnet zum Web-Server

```
telnet cis.poly.edu 80
```

Öffnet TCP Verbindung auf Port 80
(default HTTP Server-Port) von cis.poly.edu.

2. Eingabe einer GET HTTP Anfrage:

```
GET /~ross/ HTTP/1.1  
Host: cis.poly.edu
```

Erzeugt einen minimalen
und vollständigen GET-Request
zu einem HTTP-Server

3. Was kommt als Antwort vom HTTP server?

- In der ersten Zeile der Client-Antwort-Nachricht (client response)
- Beispiele:
 - 200 OK
 - Anfrage wird beantwortet in dieser Nachricht
 - 301 Moved Permanently
 - neue Adresse für Objekt
 - Adresse folgt in der Nachricht
 - 400 Bad Request
 - Anfrage wird nicht verstanden
 - 404 Not Found
 - Angefragtes Dokument nicht vorhanden
 - 505 HTTP Version Not Supported

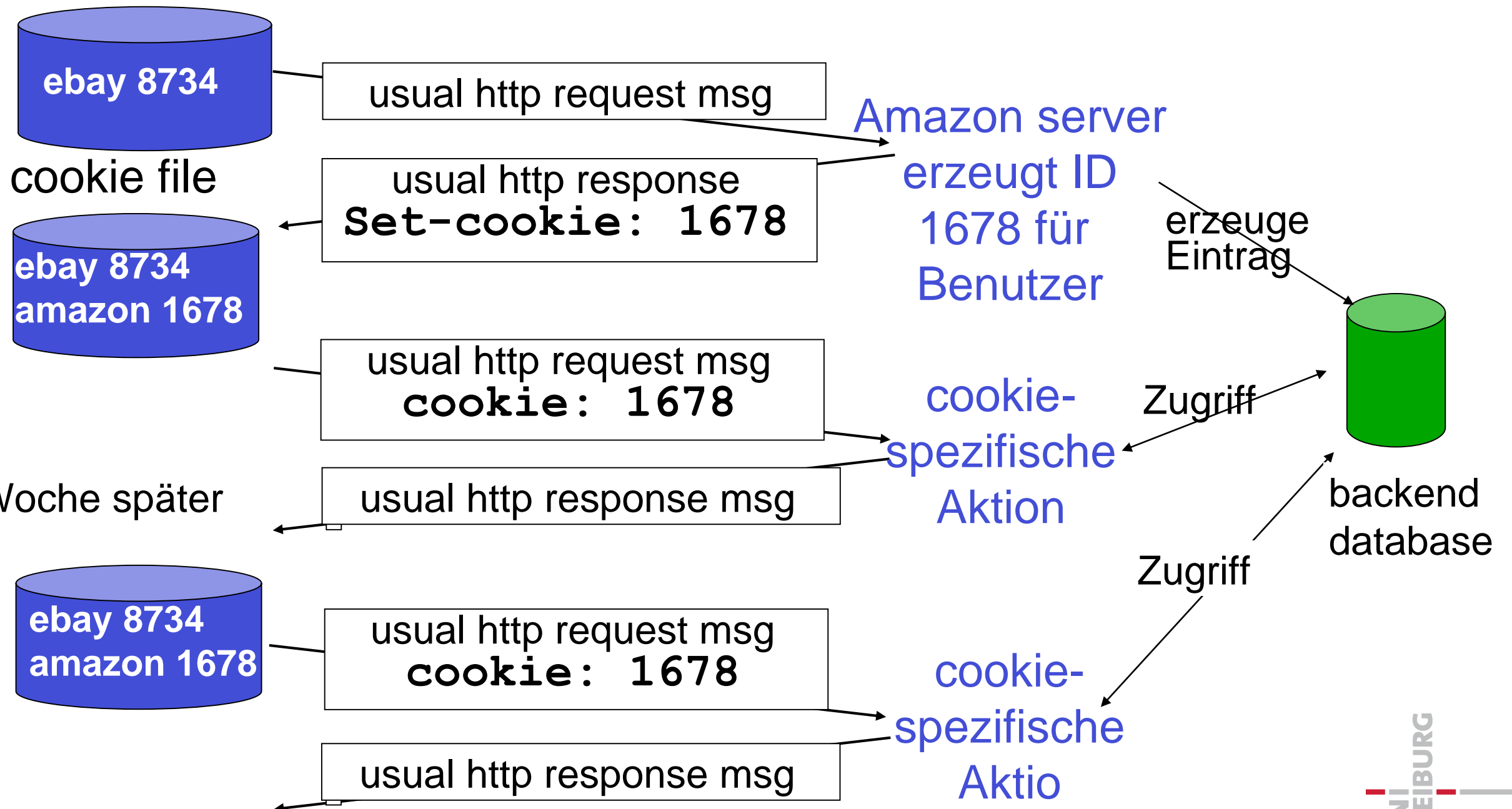
- Viele Web-Sites verwenden Cookies
- Vier Komponenten
 - 1) Cookie Kopf-Zeile der HTTP-Antwort-Nachricht (Response Message)
 - 2) Cookie-Kopf-Zeile in HTTP-Anfrage-Nachricht (Request Message)
 - 3) Cookie-Datei auf dem Benutzer-Rechner
 - wird vom Web-Browser des Benutzers unterhalten
 - 4) Datenbank auf der Web-Site (des Servers)

- Beispiel:
- Susan
 - surft das Web vom PC
 - besucht E-Commerce-Site *Amazon* zum ersten Mal
 - wenn die HTTP-Anfrage die Site erreicht, erzeugt die Web-Site
 - eindeutige ID
 - Eintrag in der Datenbank des Web-Servers

Cookies: Erzeugen einer Status-Information

Client

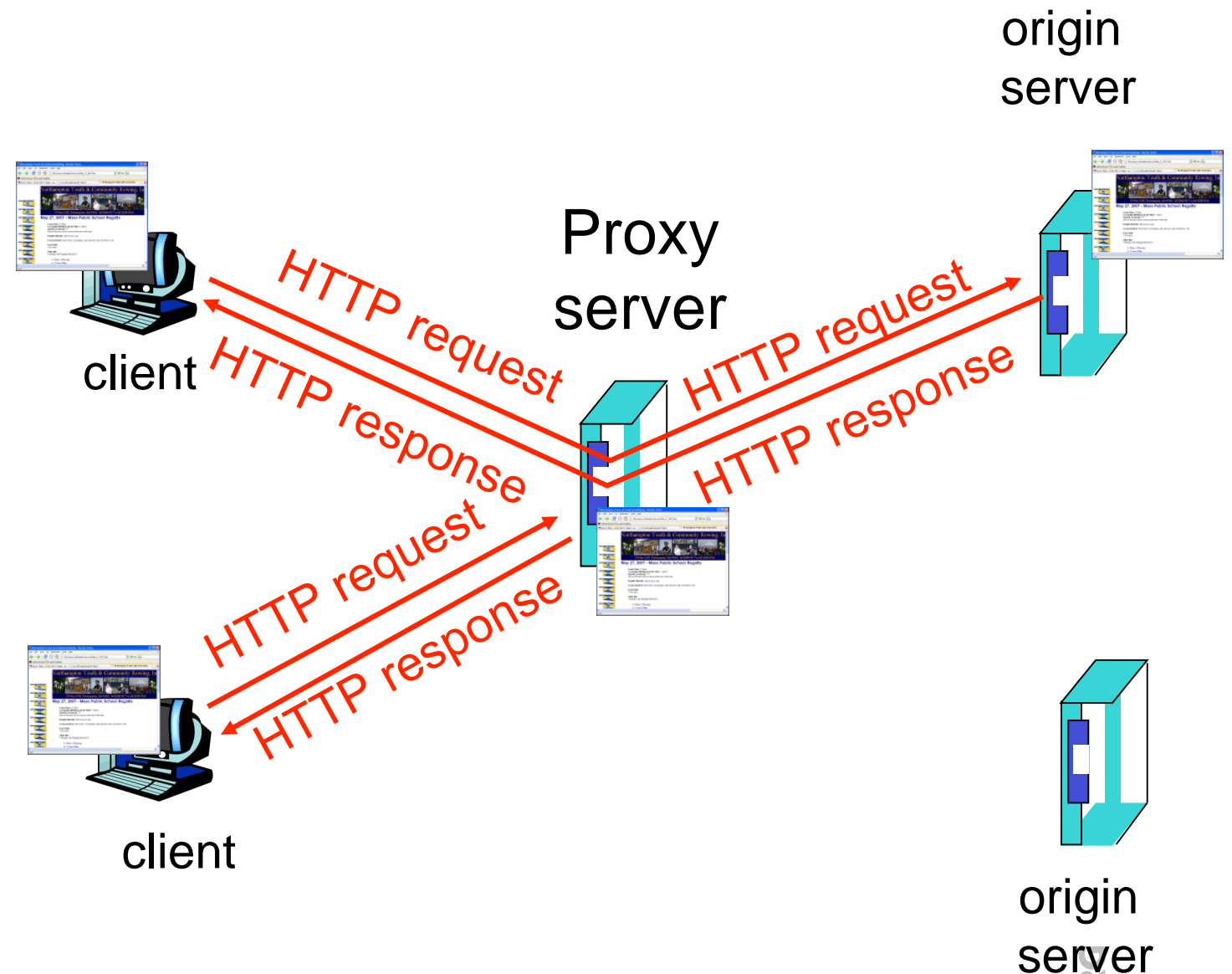
Server



- Cookies erlauben
 - Authentifikation
 - Einkaufswagen
 - Empfehlungen
 - Sitzungs-Status des Benutzers (Web Mail)
- Wie man den Status unterhält
 - speichert Zustand zwischen verschiedenen Transaktionen
 - Cookies: HTTP Nachrichten transportieren den Status
- Cookies und Privatsphäre
 - Cookies übergeben der Web-Site eine Menge von Informationen
 - z.B. Name, E-Mail, Kaufverhalten, etc.

Web Caches (Proxy Server)

- Ziel:
 - Client-Anfragen erfüllen ohne den Original-Server zu verwenden
- Benutzer greift auf das Web per Cache zu
 - Hierfür wird Browser konfiguriert
- Browser sendet alle HTTP-Anfragen zum Cache
 - Ist das Objekt im Cache, dann wird das Objekt geliefert
 - ansonsten liefert der Original-Server an den Proxy-Server
 - dieser liefert dann das Objekt an den Client



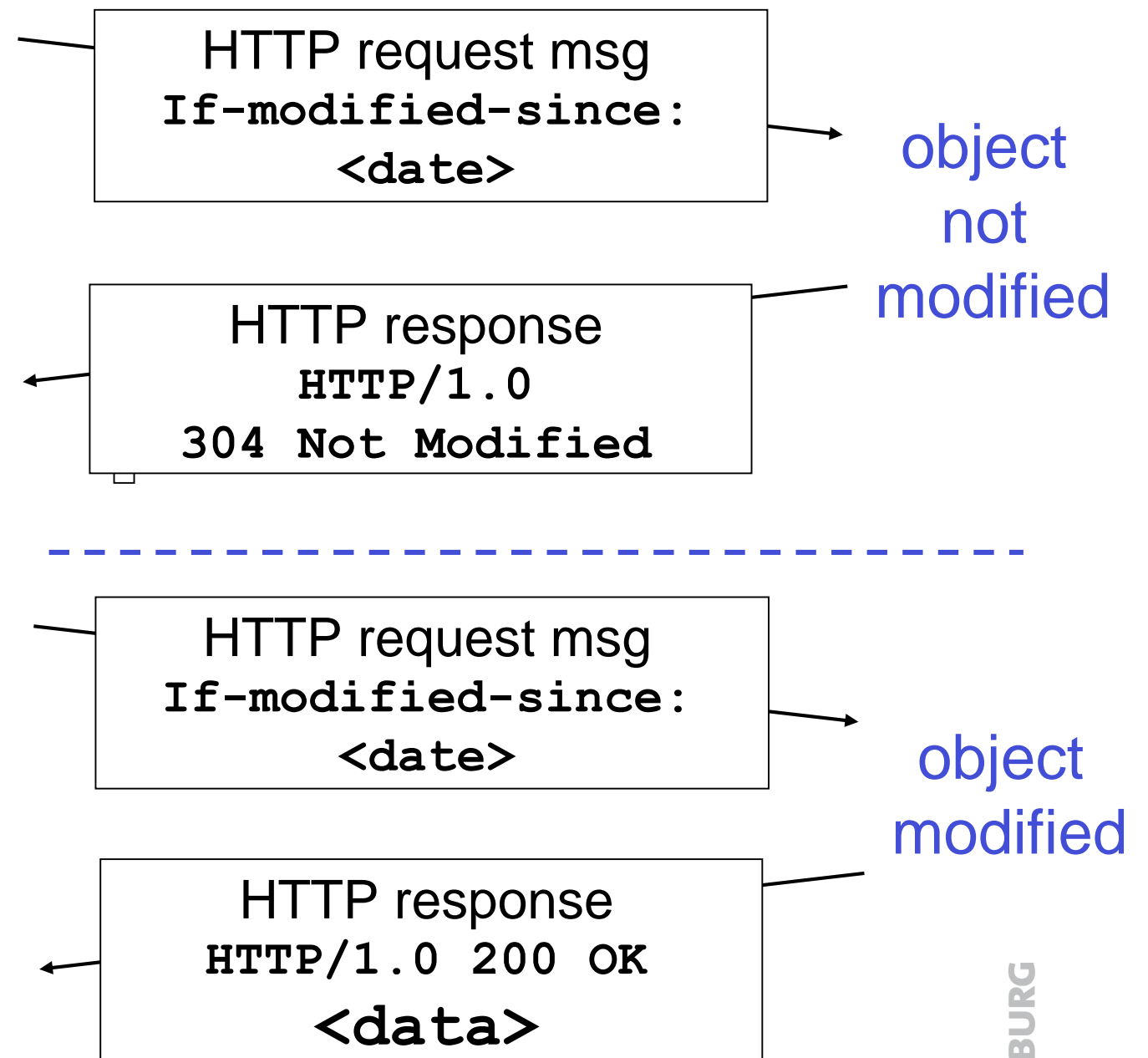
- Cache fungiert als Client und Server
 - typisch wird der Cache vom ISP (Internet Service Provider) bereit gestellt
- Warum
 - reduziert Antwortzeit für Client-Anfragen
 - reduziert den Verkehr über die Leitungen zu anderen ISPs
 - ermöglicht „kleinen“ Web-Servern effizient Inhalte zu verteilen

Conditional GET

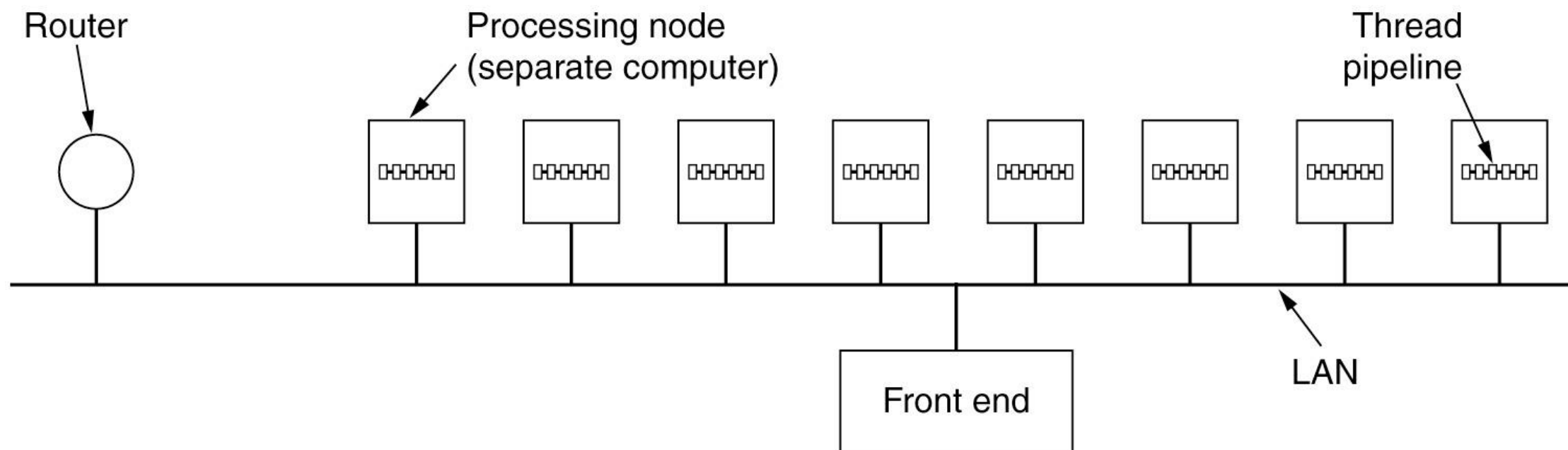
- Ziel: Objekt soll nicht gesendet werden, falls der Cache die aktuelle Version hat
- Cache: gibt den Zeitsempel der gecachten Kopie einer HTTP-Anfrage
 - If-modified-since: <date>
- Server: Antwort enthält kein Objekt, falls, die gecachte Kopie aktuell ist
 - HTTP/1.0 304 Not Modified

Cache

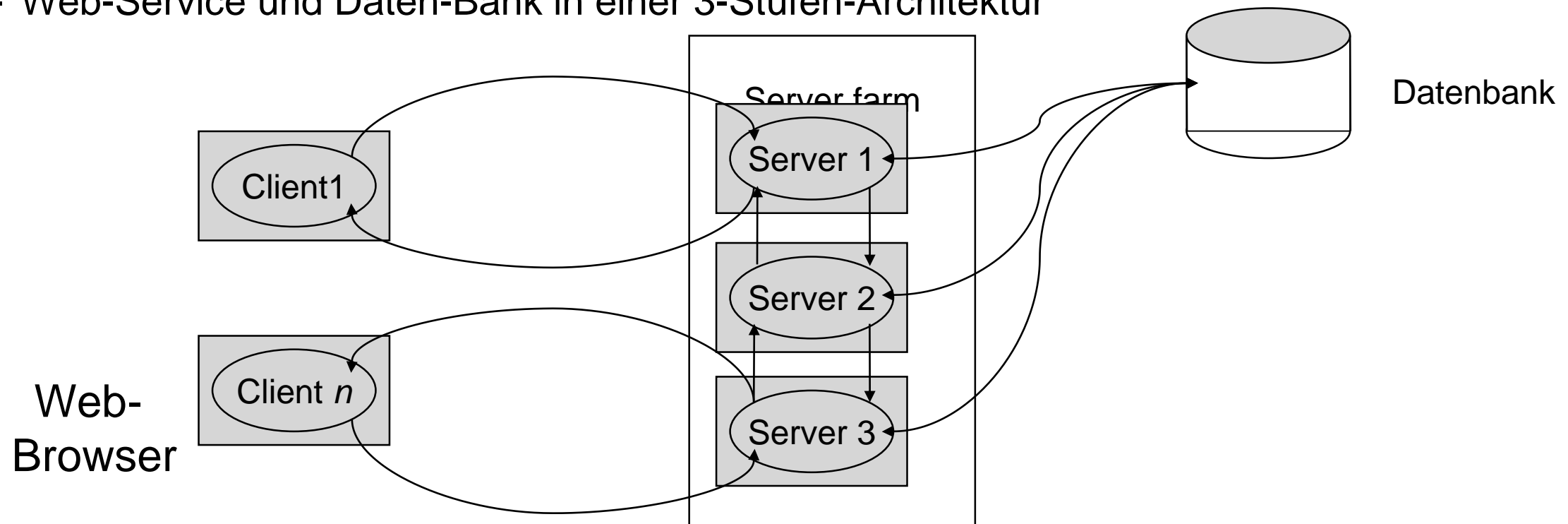
Server



- Um die Leistungsfähigkeit auf der Server-Seite zu erhöhen
 - wird eine Reihe von Web-Servern eingesetzt
- Front end
 - nimmt Anfragen an
 - reicht sie an separaten Host zur Weiterbearbeitung weiter



- Web-Server stellen nicht nur statische Web-Seiten zur Verfügung
 - Web-Seiten werden auch automatisch erzeugt
 - Hierzu wird auf eine Datenbank zurückgegriffen
 - Diese ist nicht statisch und kann durch Interaktionen verändert werden
- Problem:
 - Konsistenz
- Lösung
 - Web-Service und Daten-Bank in einer 3-Stufen-Architektur



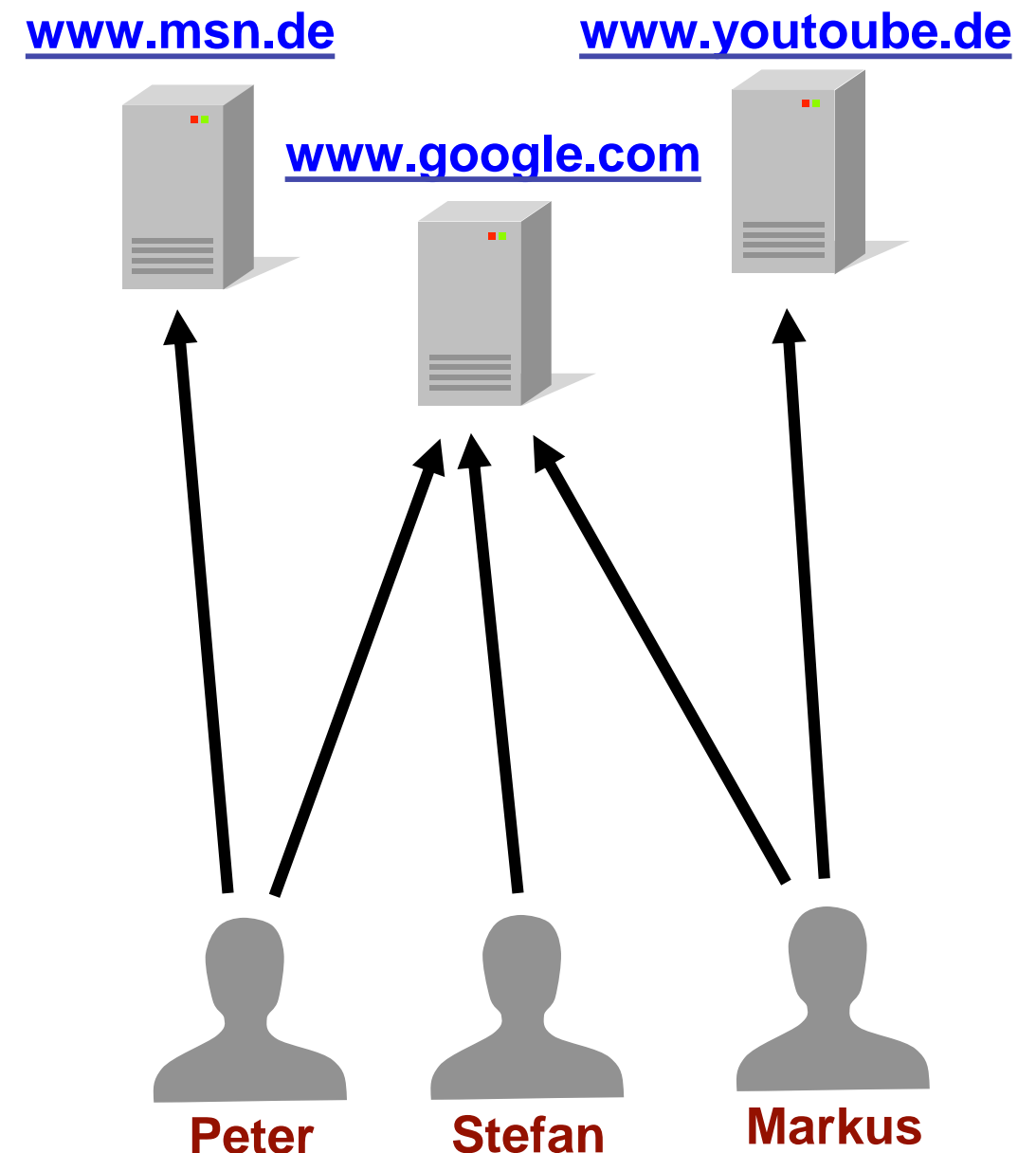
Beispiel: Google Data Centers

- Kosten eines Daten-Centers: 600 Mio US\$
- Google investierte 2007 2,4 Mrd. US\$ in Daten-Center
- Jedes Daten-Center verbraucht 50-100 MW

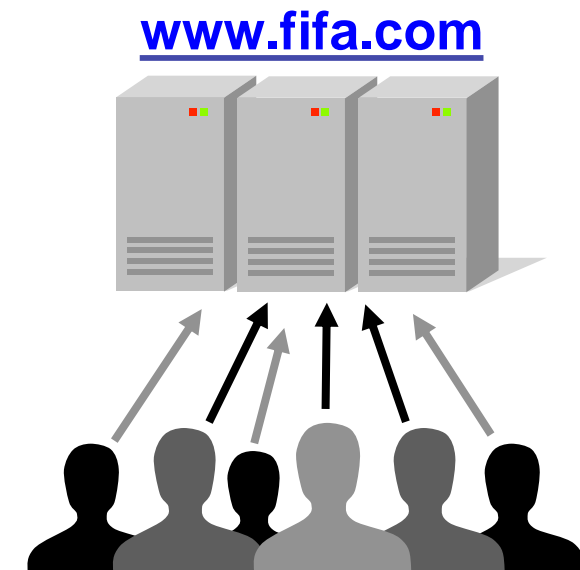


- Eine koordinierte Menge von Caches
 - Die Last großer Web-Sites wird verteilt auf global verteilte Server-Farmen
 - Diese übernehmen Web-Seiten möglichst verschiedener Organisationen
 - z.B. News, Software-Hersteller, Regierungen
 - Beispiele: Akamai, Digital Island
 - Cache-Anfragen werden auf die regional und lastmäßig bestgeeigneten Server umgeleitet
- Beispiel Akamai:
 - Durch verteilte Hash-Tabellen ist die Verteilung effizient und lokal möglich

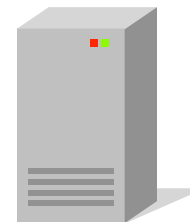
- Für Surfen im Web typisch:
 - Web-Server bieten Web-Seiten an
 - Web-Clients fordern Web-Seiten an
- In der Regel sind diese Mengen disjunkt
- Eingehende Anforderungen belasten Web-Server hinsichtlich:
 - Übertragungsbandbreite
 - Rechenaufwand (Zeit, Speicher)



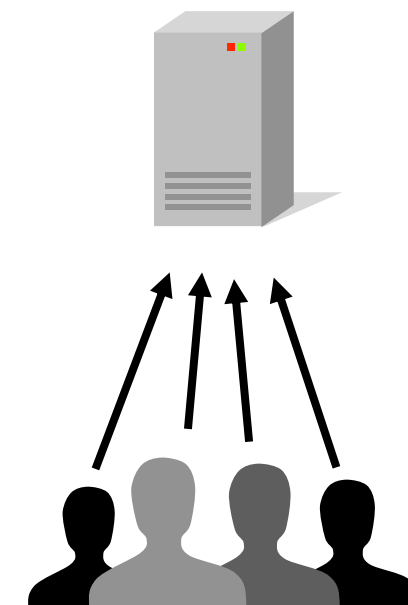
- Einige Web-Server haben immer hohe Lastanforderungen
 - Z.B. Nachrichten-Sites, Suchmaschinen, Web-verzeichnisse
 - Für permanente Anforderungen müssen Server entsprechen ausgelegt werden
- Andere leiden unter hohen Fluktuationen
 - z. B. bei besonderen Ereignissen:
 - fifa.com (Fussball-EM)
 - t-mobile.de (iPhone 6 Einführung)
 - Server-Erweiterung nicht sinnvoll
 - Bedienung der Anfragen aber erwünscht



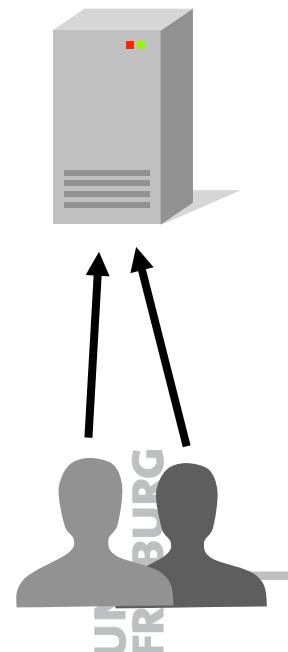
Montag



Dienstag

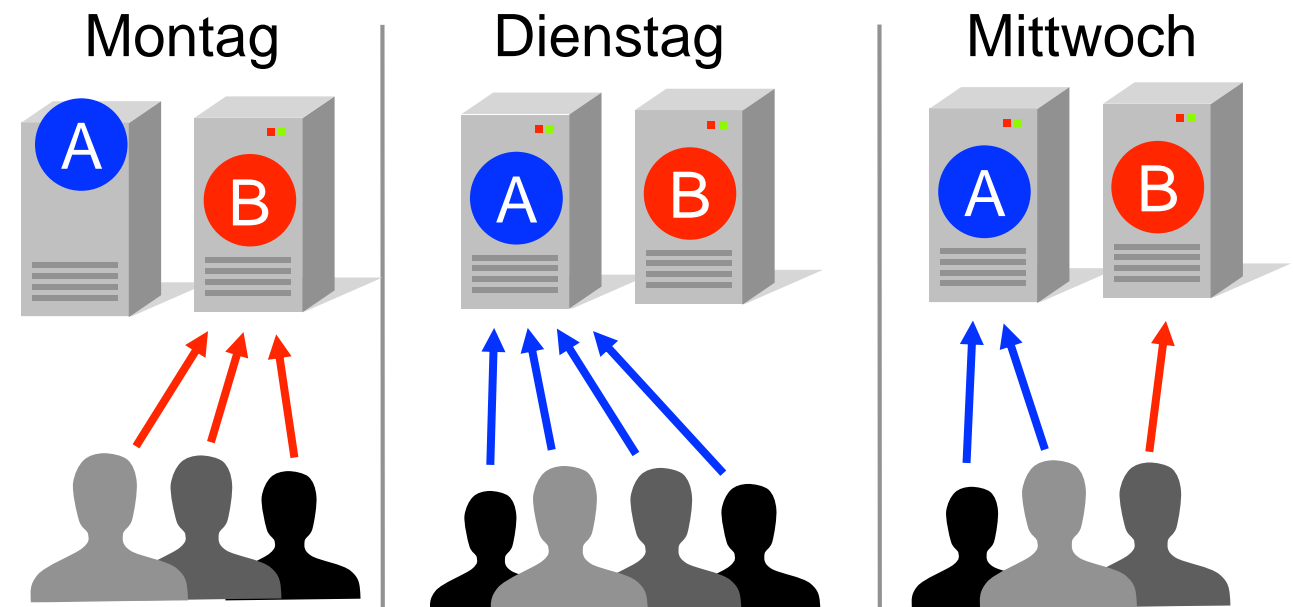


Mittwoch

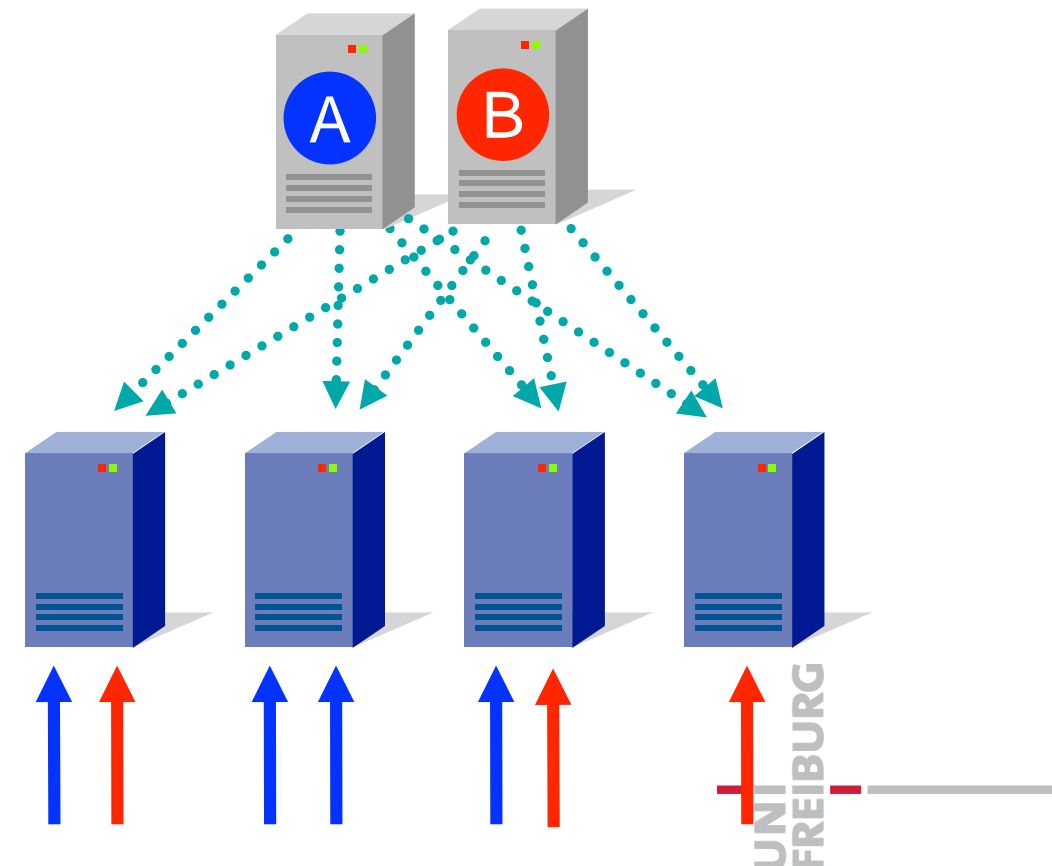


Lastbalancierung im WWW

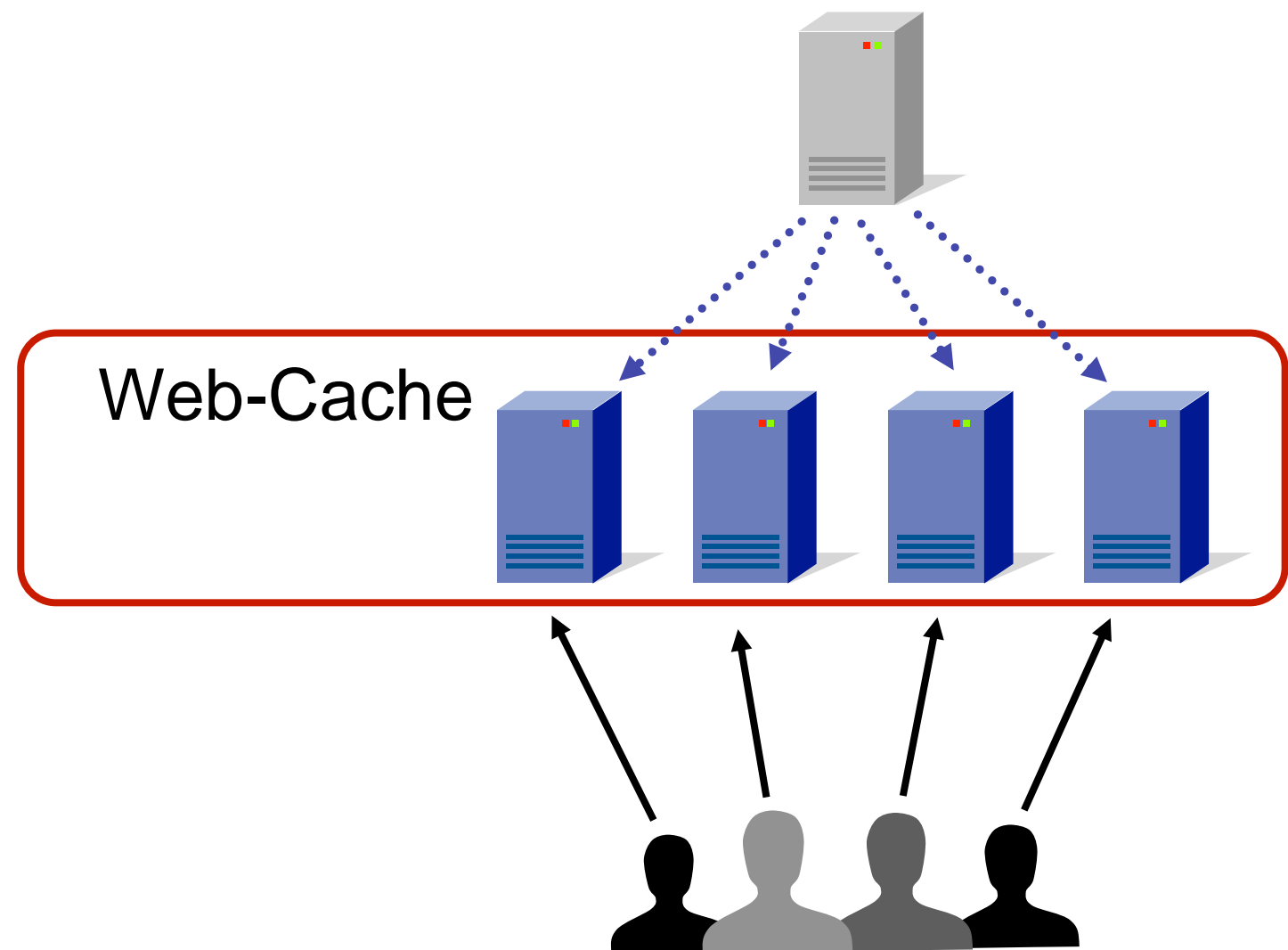
- Fluktuationen betreffen meistens einzelne Server



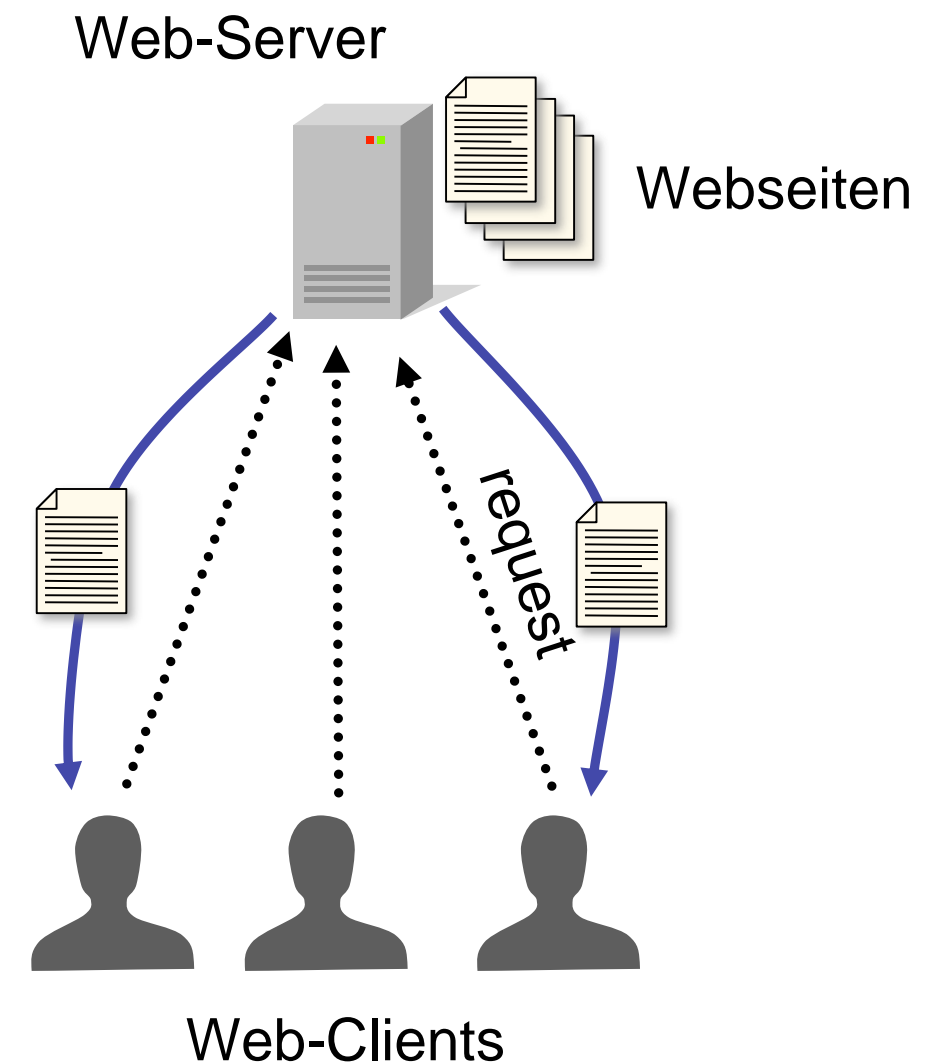
- (Kommerzielle) Lösung
 - Dienstleister bieten Ausweich-(Cache-)Server an
 - Viele Anforderungen werden auf diese Server verteilt
- Aber wie?



- Leighton, Lewin, et al.
STOC 97
 - *Consistent Hashing and Random Trees: Distributed Caching Protocols for Relieving Hot Spots on the World Wide Web*
 - Passen bestehende Verfahren für dynamische Hash-Funktionen an WWW-Anforderungen an
- Leighton und Lewin (MIT) gründen Akamai 1997

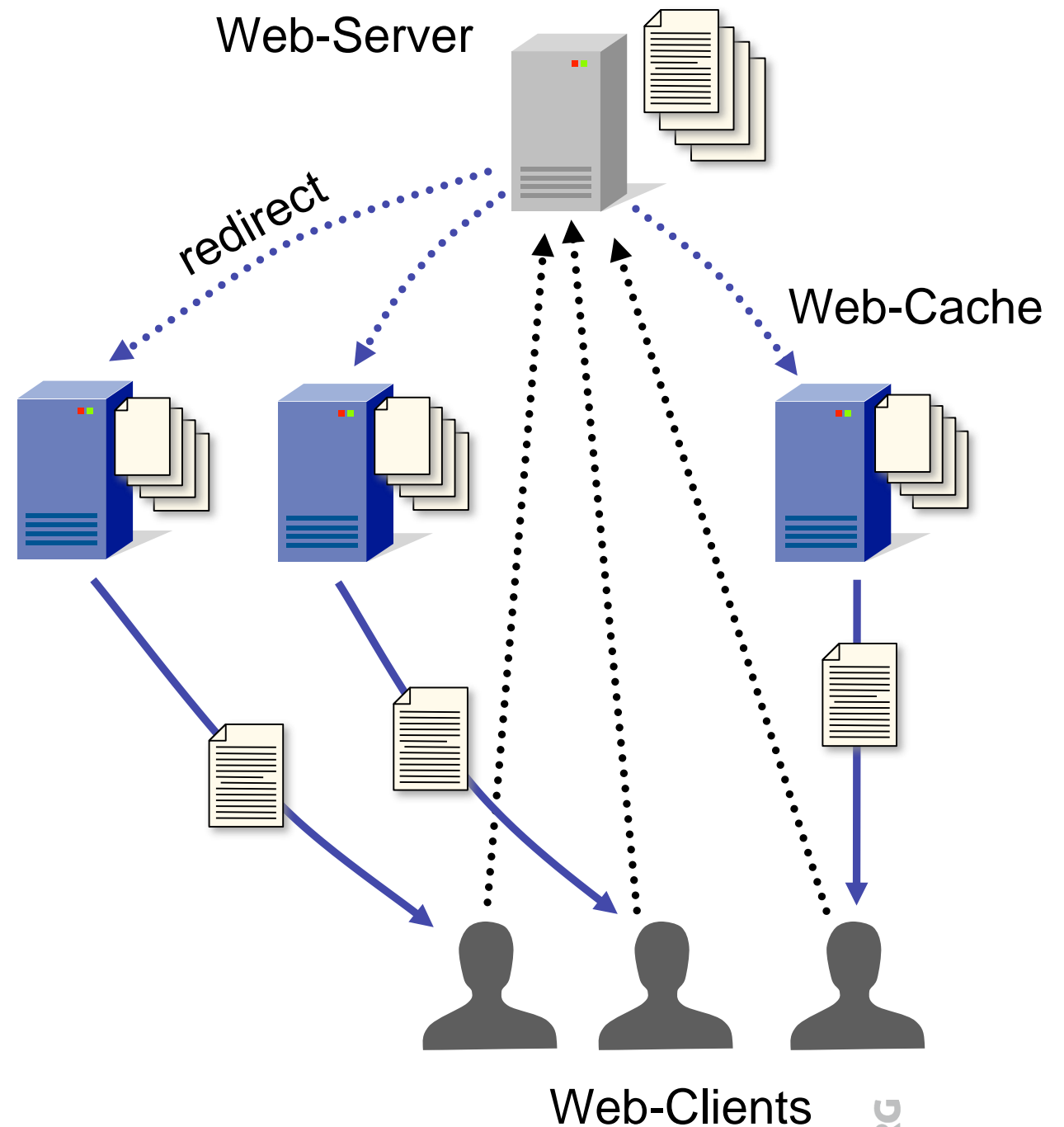


- Ohne Lastbalancierung:
 - Jeder Browser (Web-Client) belegt einen Web-Server für eine Web-Site
- Vorteil:
 - Einfach
- Nachteil:
 - Der Server muss immer für den Worst-Case ausgelegt werden



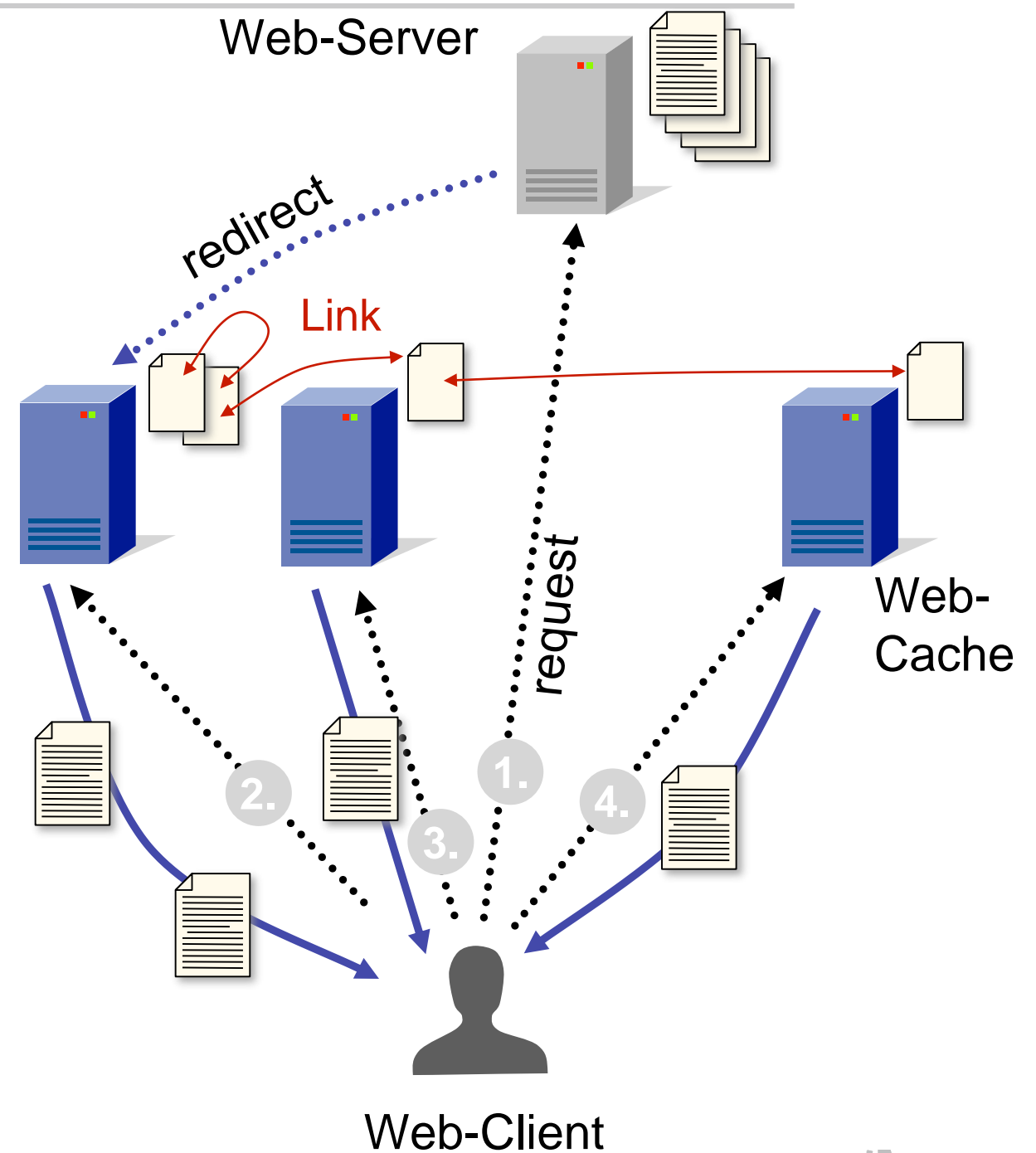
Site Caching

- Ganze Web-Site wird auf verschiedene Web-Caches kopiert
- Browser fragt bei Web-Server nach Seite
- Web-Server leitet Anfrage auf Web-Cache um (redirect)
- Web-Cache liefert Web-Seite aus
- Vorteil:
 - Gute Lastbalancierung für Seitenverteilung
- Nachteil:
 - Bottleneck: Redirect
 - Großer Overhead durch vollständige Web-Site-Replikationen



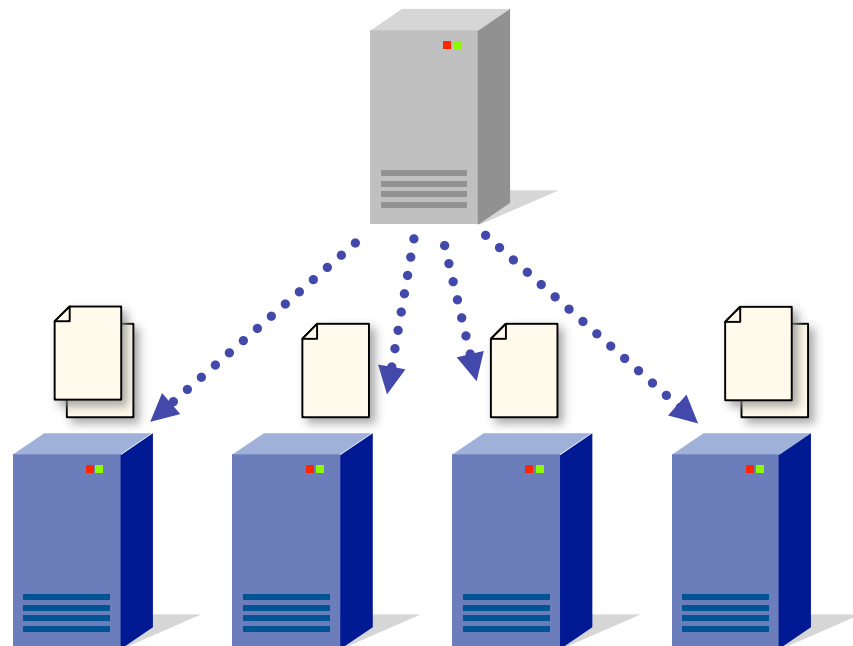
Proxy Caching

- Jede Web-Seite wird auf einige (wenige) Web-Caches verteilt
- Nur Startanfrage erreicht Web-Server
- Links referenzieren auf Seiten im Web-Cache
- Dann surft der Web-Client nur noch auf den Web-Cache
- Vorteil:
 - Kein Bottleneck
- Nachteil:
 - Lastbalancierung nur implizit möglich
 - Hohe Anforderung an Caching-Algorithmus

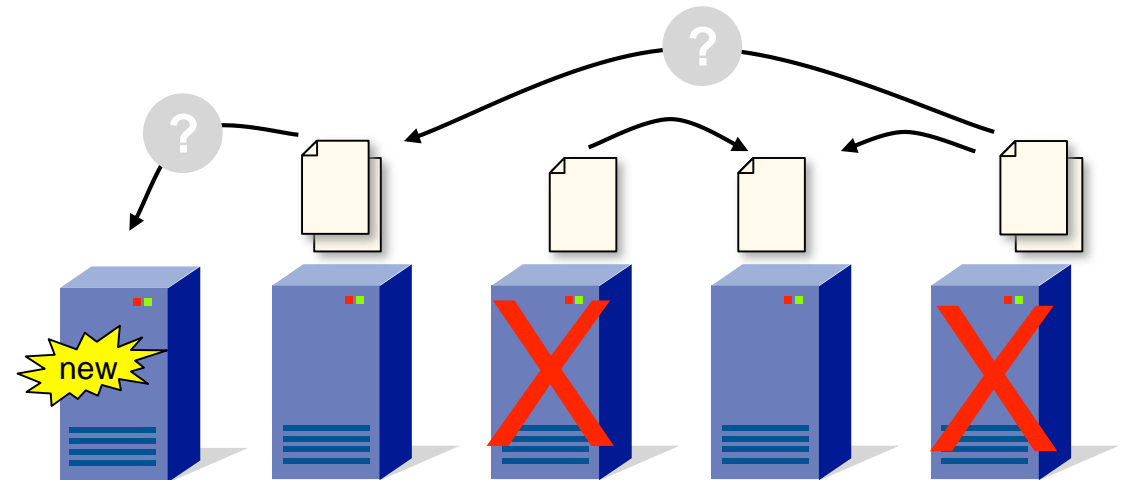


Anforderungen an Caching-Algorithmus

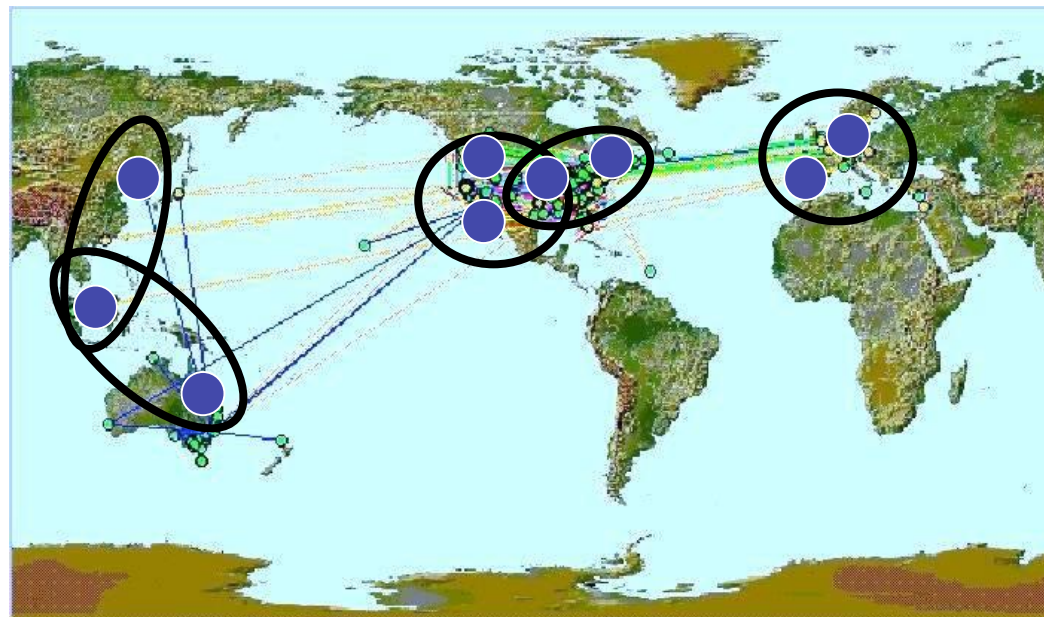
Balance
Gleichmäßige Verteilung der Seiten



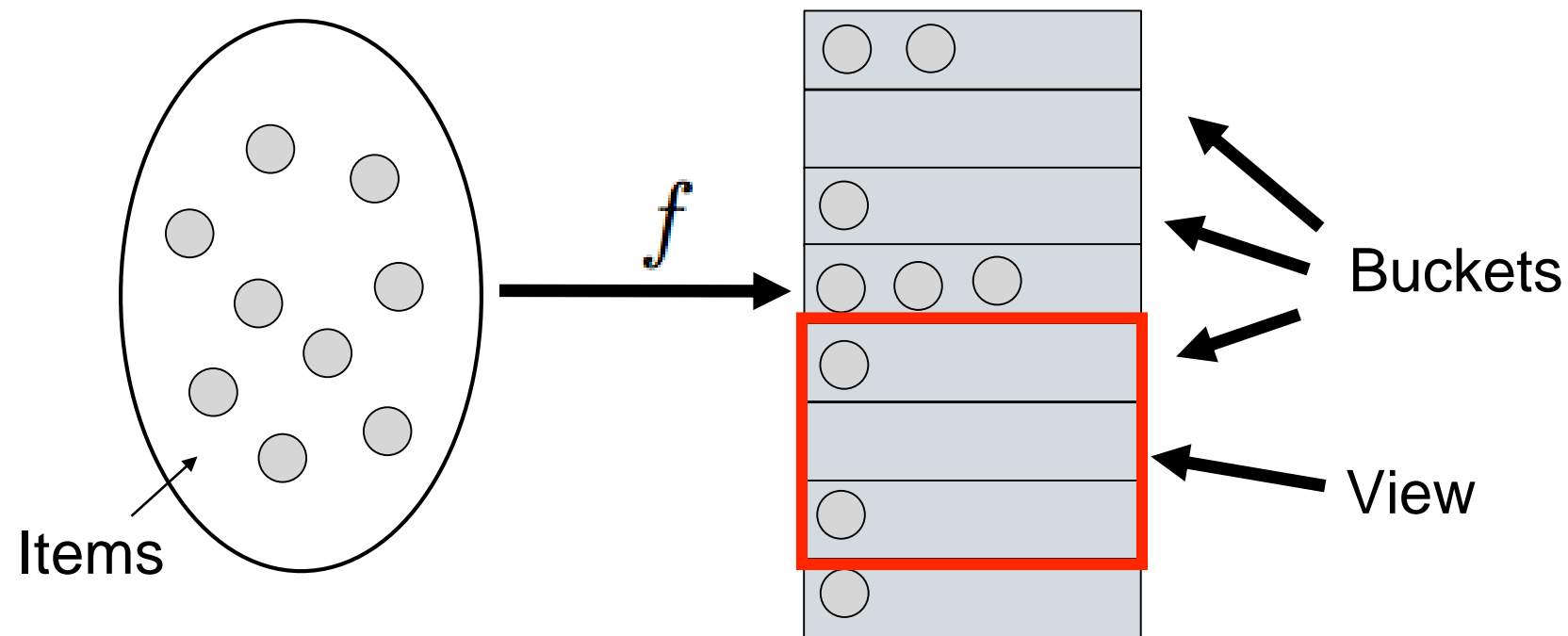
Dynamik
Effizientes Einfügen/Löschen von neuen
Web-Cache-Servern



Views
Web-Clients „sehen“
unterschiedliche Menge
von Web-Caches

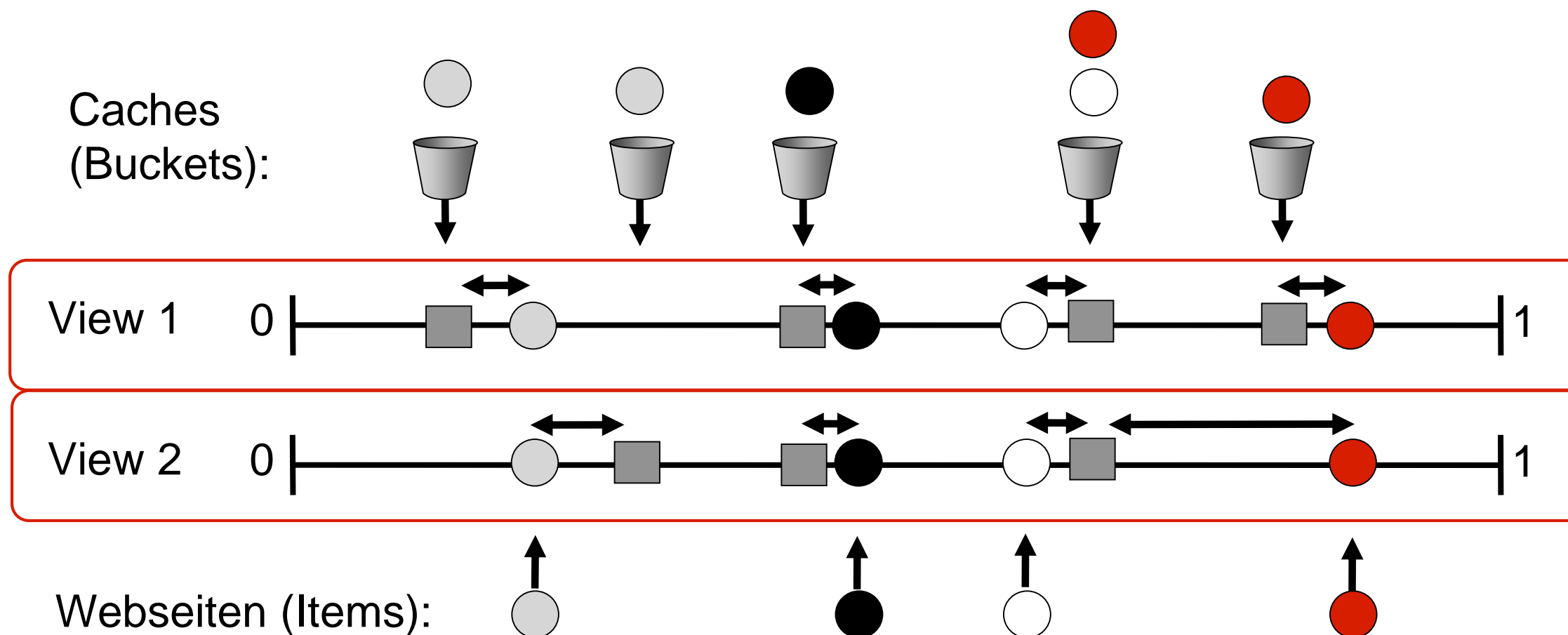


- Gegeben:
 - Elemente (Items)
 - Caches (Buckets)
 - Views: Menge von Caches
- Ranged Hash-Funktion:
 - Zuordnung eines Elements zu einem Cache in einem View

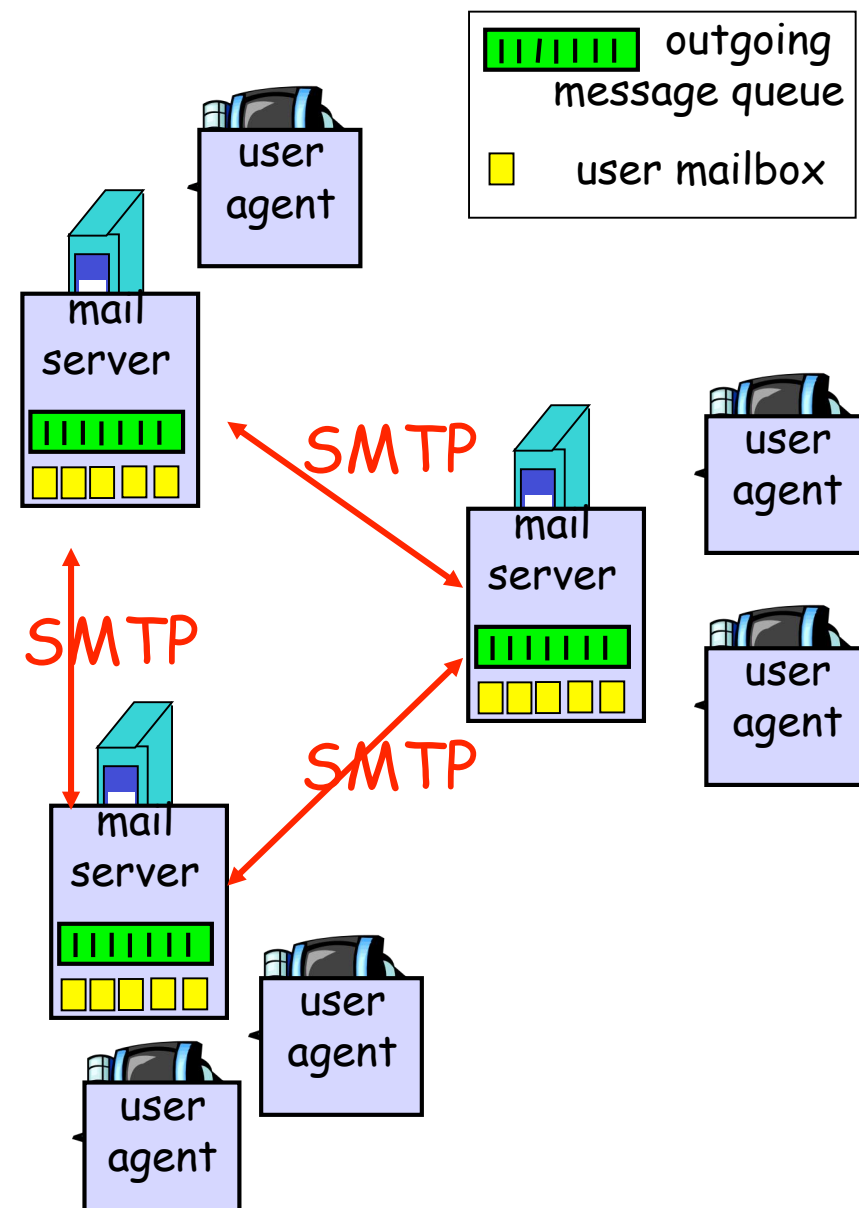


- **Monotonie**
 - nach dem Hinzufügen neuer Caches (Buckets) sollten keine Seiten (Items) zwischen alten Caches verschoben werden
- **Balance**
 - Alle Caches sollten gleichmäßig ausgelastet werden
- **Spread (Verbreitung, Streuung)**
 - Eine Seite sollte auf eine beschränkte Anzahl von Caches verteilt werden
- **Load**
 - Kein Cache sollte wesentlich mehr als die durchschnittliche Anzahl von Seiten enthalten

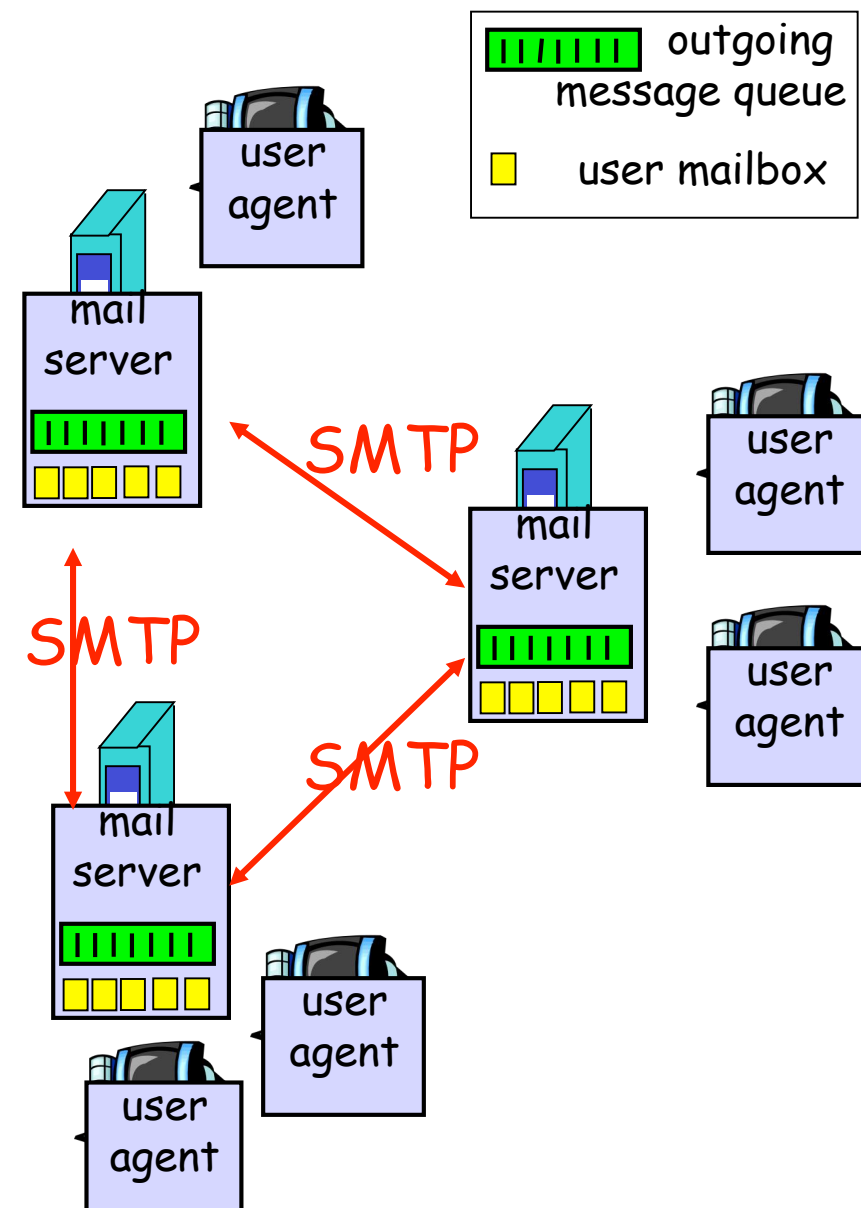
Distributed Hash Tables als Lösung



- Hauptkomponenten
 - user agents
 - mail servers
 - simple mail transfer protocol: SMTP
- User Agent
 - Mail Client
 - Erstellen, ändern und lesen von E-Mail-Nachrichten
 - z.B. Eudora, Outlook, pine, Mozilla Thunderbird
 - abgehende und ankommende Nachrichten werden auf dem Server gespeichert



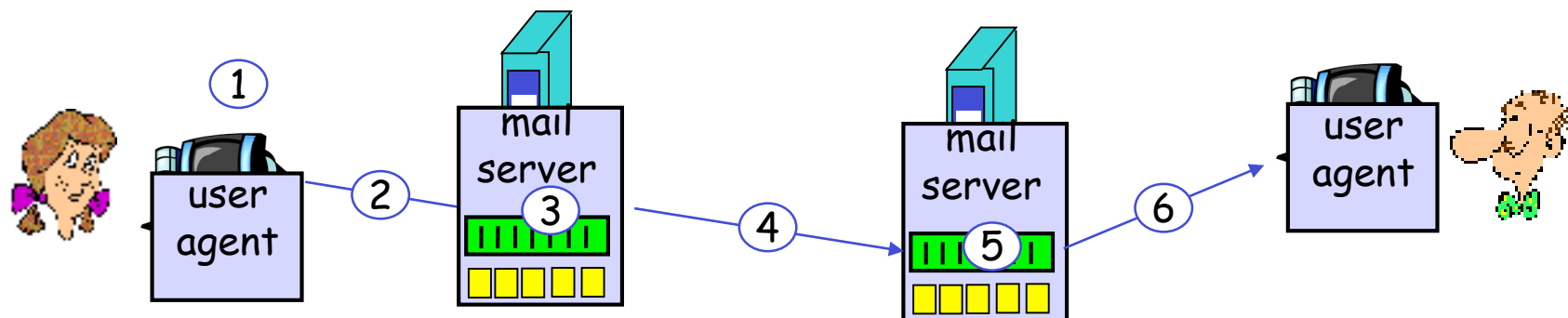
- Mailbox speichert eingehende Nachrichten für den User
- Nachrichten-Warteschlange (queue) der zu versendenden Nachrichten
- SMTP-Protocol zwischen Mail-Servern um E-Mail-Nachrichten zu schicken



- verwendet TCP um zuverlässig E-Mail-Nachrichten vom Client auf Port 25 zu verschicken
- Direkte Übertragung von Absender-Server zum Empfangs-Server
- 3 Phasen in der Übertragung
 - Handshake
 - Transfer der Nachricht
 - Abschluss
- Befehle und Antwort
 - Befehle als ASCII text
 - Antwort: Status-Code und Kurzbeschreibung
- Nachrichten sind in 7-bit ASCII

Beispiel: Alice sendet eine Nachricht an Bob

- 1) Alice verwendet UA um die Nachricht zu erzeugen mit Eintrag “to” bob@some school.edu
- 2) Alice UA sendet die Nachricht zu ihren Mail-Server
 - Nachricht wird in der Nachrichtenwartenschlange platziert
- 3) Client-Seite des SMTP öffnet TCP-Verbindung mit Bobs Mail-Server
- 4) SMTP Client sendet Alice Nachricht über die TCP-Verbindung
- 5) Bobs Mail-Server schreibt die Nachricht in Bobs Mailbox
- 6) Bob ruft seinen User Agent auf, um die Nachricht zu lesen

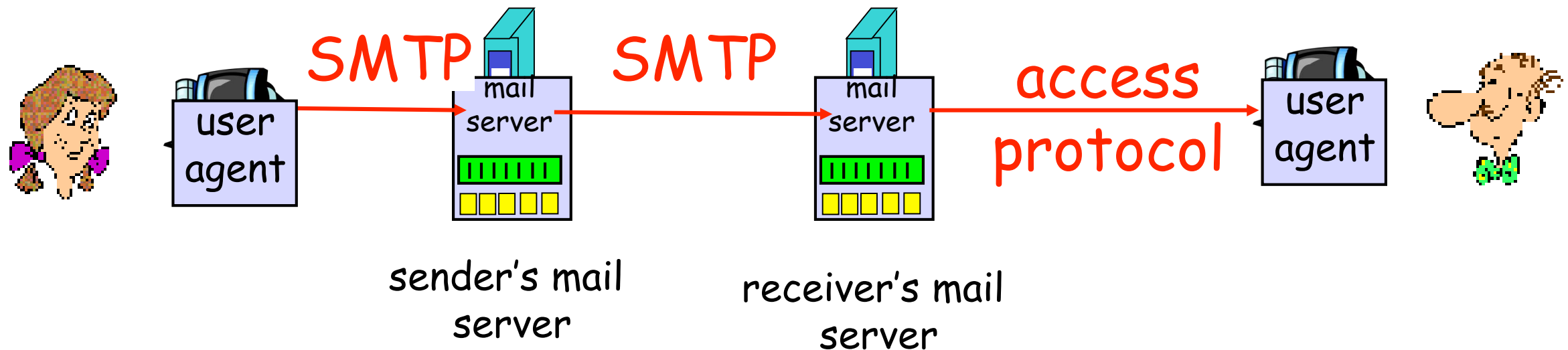


Beispiel SMTP Interaktion

```
S: 220 hamburger.edu
C: HELO crepes.fr
S: 250 Hello crepes.fr, pleased to meet you
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: .
S: 250 Message accepted for delivery
C: QUIT
S: 221 hamburger.edu closing connection
```

- SMTP
 - verwendet persistente Verbindungen
 - verlangt Nachrichten (header & body) in 7-bit ASCII
 - SMTP-Server verwenden „CRLF.CRLF“ um das Ende einer Nachricht zu beschreiben
- Vergleich mit HTTP:
 - HTTP: pull
 - SMTP: push
 - beide haben ASCII Befehls- und Antwort-Interaktion und Status-Codes
- HTTP
 - jedes Objekt wird in eigener Nachricht verpackt
- SMTP
 - verschiedene Objekte werden in einer Multipart-Nachricht verschickt

Mail-Zugriffsprotokolle



- SMTP: Auslieferung und Speicher zum Server des Empfängers
- Mail-Zugriffsprotokolle: E-Mail-Abruf vom Server
 - POP: Post Office Protocol [RFC 1939]
 - Authentifizierung (zwischen Agent und Server) und Download
 - IMAP: Internet Mail Access Protocol [RFC 1730]
 - mehr Features und komplexer
 - Bearbeitung von gespeicherten Nachrichten **auf** dem Server
 - HTTP: gmail, Hotmail, Yahoo! Mail, web.de, etc.

- POP3 (Post-Office-Protocol)
 - User kann im “download and delete” Modus E-Mails einmalig herunterladen
 - User kann E-Mails noch einmal lesen, wenn er den Client wechselt:
 - “Download-and-keep”: Kopien der Nachricht auf verschiedenen Clients
 - POP3 ist zustandslos (stateless) von einer Sitzung zur nächsten
- IMAP (Internet Message Access Protocol)
 - hält alle Nachrichten an einem Ort: dem Server
 - erlaubt dem User die Nachrichten in Ordnern zu organisieren
 - IMAP speichert den Benutzer-Status zwischen Sitzungen
 - Namen der Ordner und Zuordnung zwischen Nachrichten-ID und Ordnernamen

- Napster 1999-2000
 - Filesharing, nur rudimentäres P2P
- Gnutella 2000
 - 1. echtes P2P-Netzwerk
- Edonkey 2000
 - Mehr Filesharing als P2P
- FreeNet 2000
 - Anonymisiertes P2P-Netzwerk
- FastTrack 2001
 - KaZaa, Morpheus, Grokster
- Bittorrent 2001
- Skype 2003
 - VoIP (voice over IP), Chat, Video

- Distributed Hash-Tables (DHT) (1997)
 - Ziel: Lastbalancierung für Web-Server
- CAN (2001)
 - DHT-Netzwerk-Struktur
- Chord (2001)
 - Erstes effiziente P2P-Netzwerk
 - Logarithmische Suchzeit
- Pastry/Tapestry (2001)
 - Effizientes verteiltes P2P-Netzwerk unter Verwendung des Plaxton-Routing
- Und viele andere Ansätze
 - Viceroy, Distance-Halving, Koorde, Skip-Net, P-Grid, ...
- In den letzten fünf Jahren:
 - Network Coding for P2P
 - Game theory in P2P
 - Anonymity, Security

- Was ist P2P **NICHT**?
 - Ein Client-Server network
- Etymologie: peer
 - lateinisch: par = gleich
 - Standesgleich
 - P2P, Peer-to-Peer: Beziehung zwischen gleichwertigen Partnern
- Definition
 - Ein Peer-to-Peer Network ist ein Kommunikationsnetzwerk im Internet
 - ohne zentrale Kontrolle
 - mit gleichwertigen, unzuverlässigen Partnern

Distributed Hash-Table (DHT)

■ Hash-Tabellen

- nicht praktikabel in P2P

■ Verteilte Hash-Tabellen

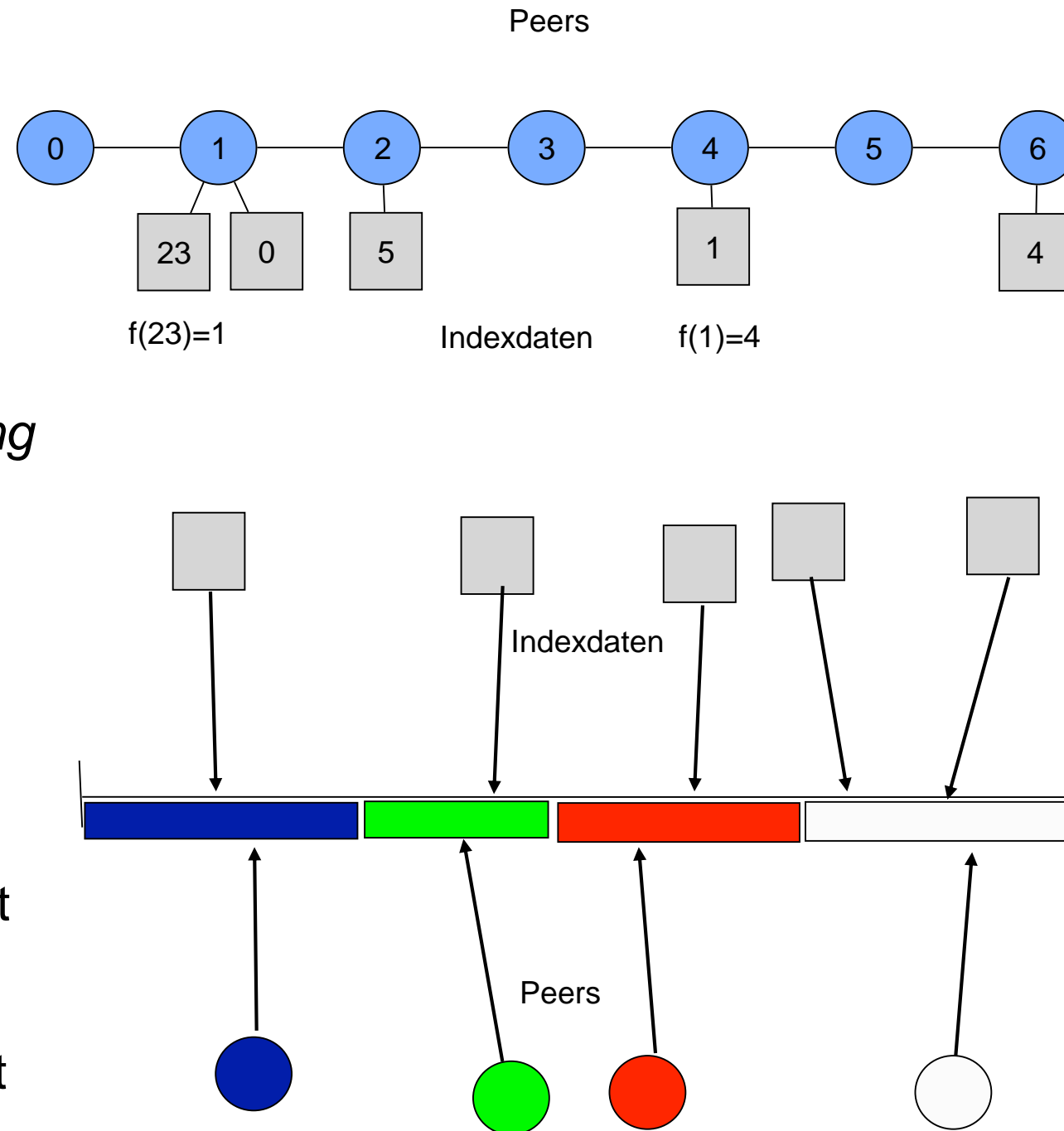
- *Consistent Hashing and Random Trees: Distributed Caching Protocols for Relieving Hot Spots on the World Wide Web*, Karger, Lehman, Leighton, Levine, Lewin, Panigrahy, STOC 1997

■ Daten

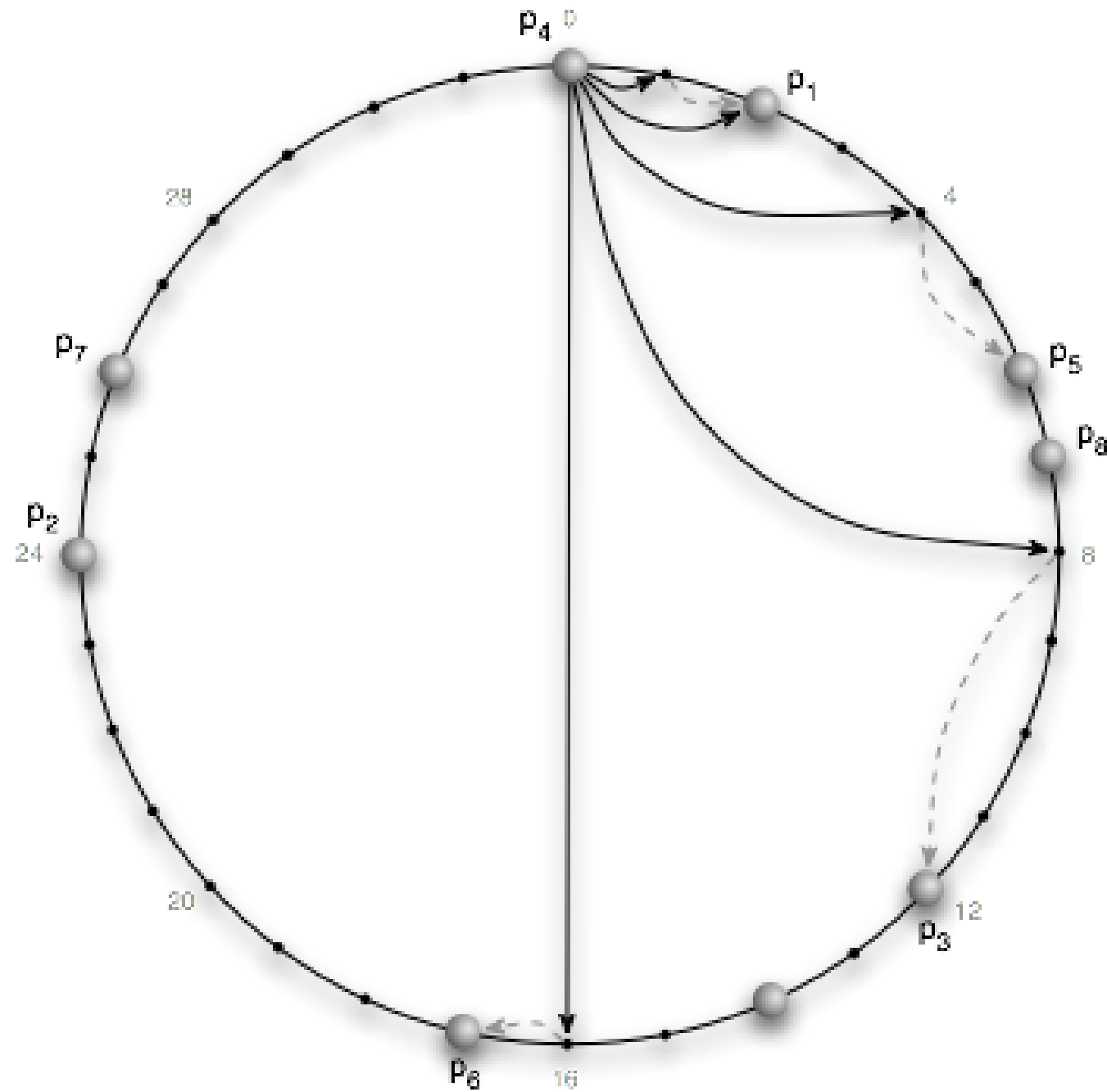
- werden *gehasht* und nach Bereich den Peers zugeordnet

■ Peers

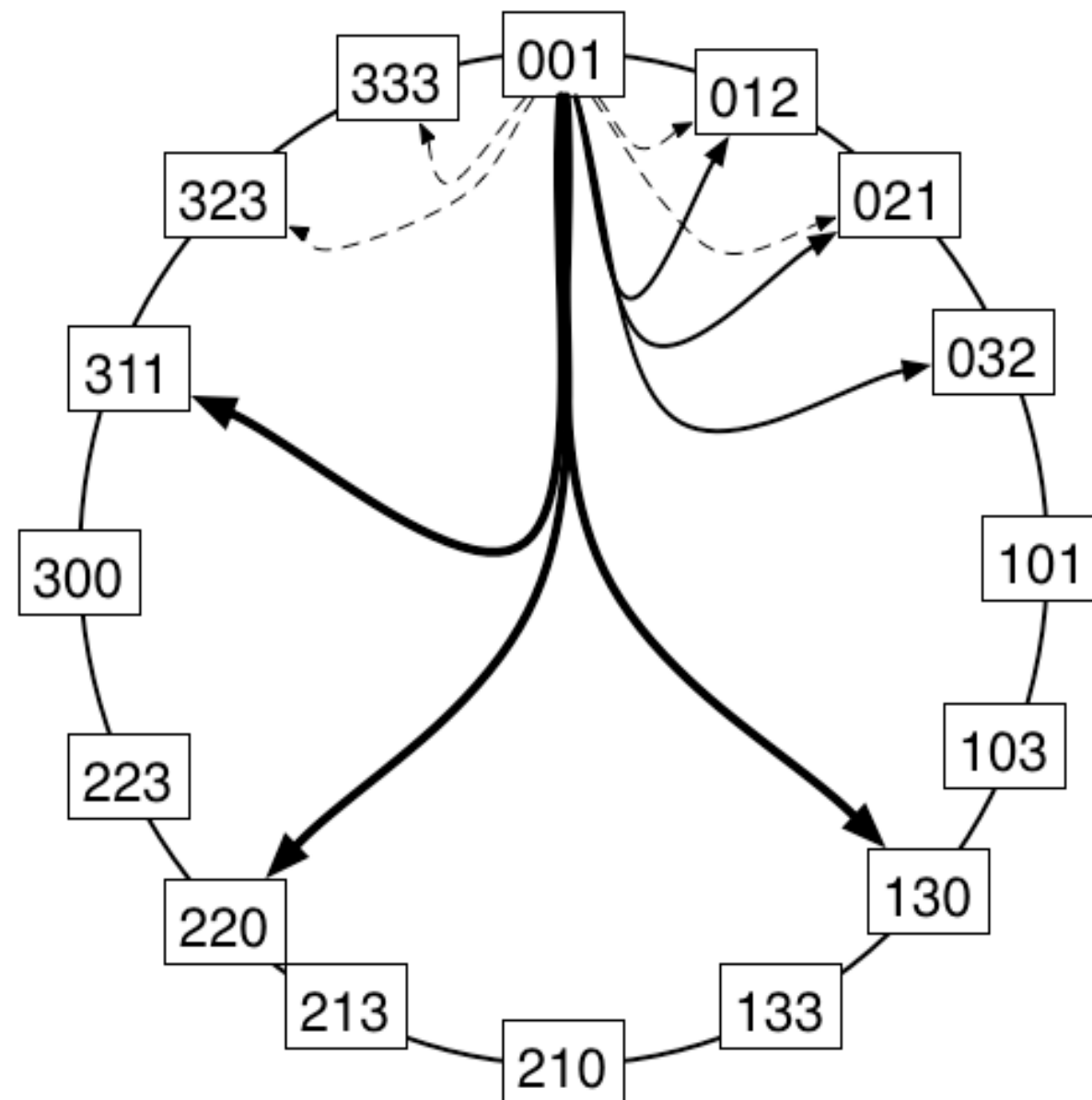
- werden an eine Stelle *gehasht* und erhalten Bereiche des Wertebereichs der Hashfunktion zugeteilt



Zeiger-Struktur in Chord



- Peter Druschel
 - jetzt Direktor des Max-Planck-Instituts für Informatik, Saarbrücken/Kaiserslautern
- Antony Rowstron
 - Microsoft Research, Cambridge, GB
- Pastry
 - *Scalable, decentralized object location and routing for large scale peer-to-peer-network*
 - Chord-ähnliches Netzwerk, welches das Routing von Plaxton, Rajamaran, Richa (1997) verwendet



- Bram Cohen

- BitTorrent ist ein P2P-Netzwerk für den Download von Dateien
- Dateien werden in Blöcke aufgeteilt
- verwendet implizit Multicast-Bäume für die Verteilung von Blöcken

- Ziele

- schneller Download einer Datei unter Verwendung des Uploads vieler Peers
 - Upload ist der Flaschenhals
 - z.B. wegen asymmetrischen Aufbau von ISDN oder DSL
- Fairness
 - seeders against leeches
- Gleichzeitige Verwendung vieler Peers

Systeme II

5. Die Anwendungsschicht

Christian Schindelhauer

Technische Fakultät

Rechnernetze und Telematik

Albert-Ludwigs-Universität Freiburg