CoNe Freiburg

IIF INSTITUT FÜR INFORMATIK FREIBURG

Systeme II

4. Die Anwendungsschicht (2. Teil)

Thomas Janson°, Kristof Van Laerhoven*, Christian Ortolf°

Folien: Christian Schindelhauer°

Technische Fakultät

°: Rechnernetze und Telematik, *: Eingebettete Systeme

Albert-Ludwigs-Universität Freiburg

Version 20.05.2015

UNI FREIBURG

CoNe Freiburg

Anwendungsschicht
Ziele

- Aspekte der Programmierung im Internet aus der Sicht der Anwendung
- Anforderungen an die Transportschicht
  - z.B. TCP übermittelt Daten zuverlässig
- Prinzipien:
  - Client-Server
  - Peer-to-Peer
- Beispiel-Protokolle:
  - HTTP
  - SMTP / POP3 / IMAP
  - DNS
- Programmierung von Netzwerk-Anwendungen

CoNe Freiburg

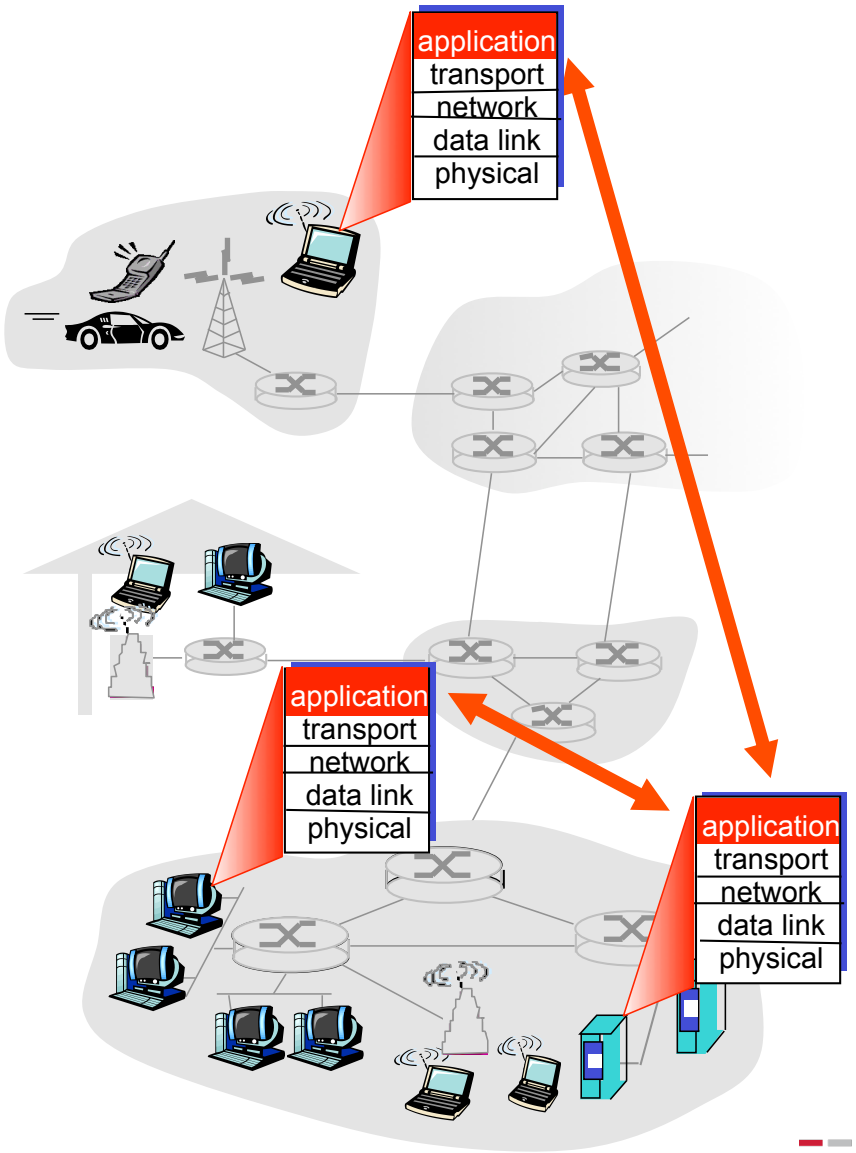
Beispiele Netzwerk-Anwendungen

- E-Mail
- Web
- Instant messaging
- Remote Login (z.B. ssh)
- P2P File Sharing (z.B. BitTorrent)
- Multi-User Network Games
- Video Streaming
- Social Networks
- Voice over IP (Internet-Telefonie, z.B. Skype)
- Real-time Video Konferenz
- Grid Computing (verteiltes Rechnen)

CoNe Freiburg

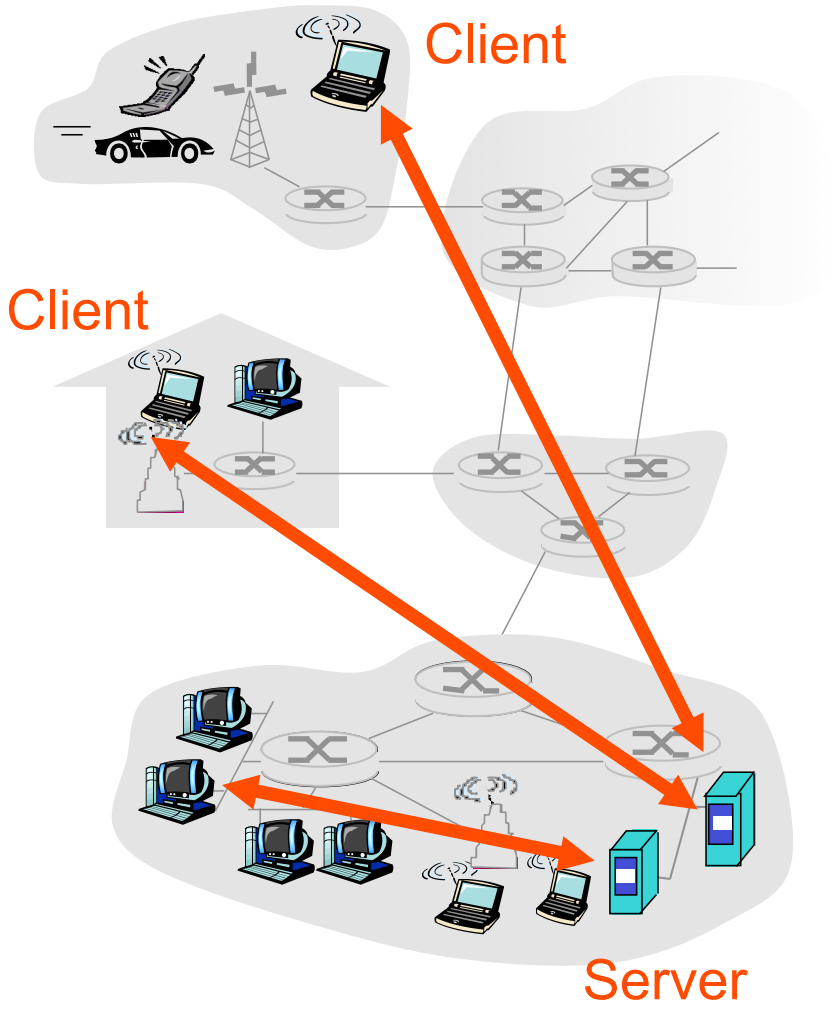
Erstellen einer Netzwerk-Anwendung

- Programme laufen auf den End-Punkten
  - kommunizieren über das Netzwerk, z.B Web-Browser mit Web-Server
  - Programmierung nur in der Applikationsschicht
  - Sockets bieten plattformunabhängige Schnittstelle zu Transportschicht
- Netzwerk-Router
  - werden nicht programmiert!
  - nicht für den Benutzer verfügbar
- dadurch schnelle Programm-Entwicklung möglich
  - gleiche Umgebung
  - schnelle Verbreitung



Client-Server-Architektur

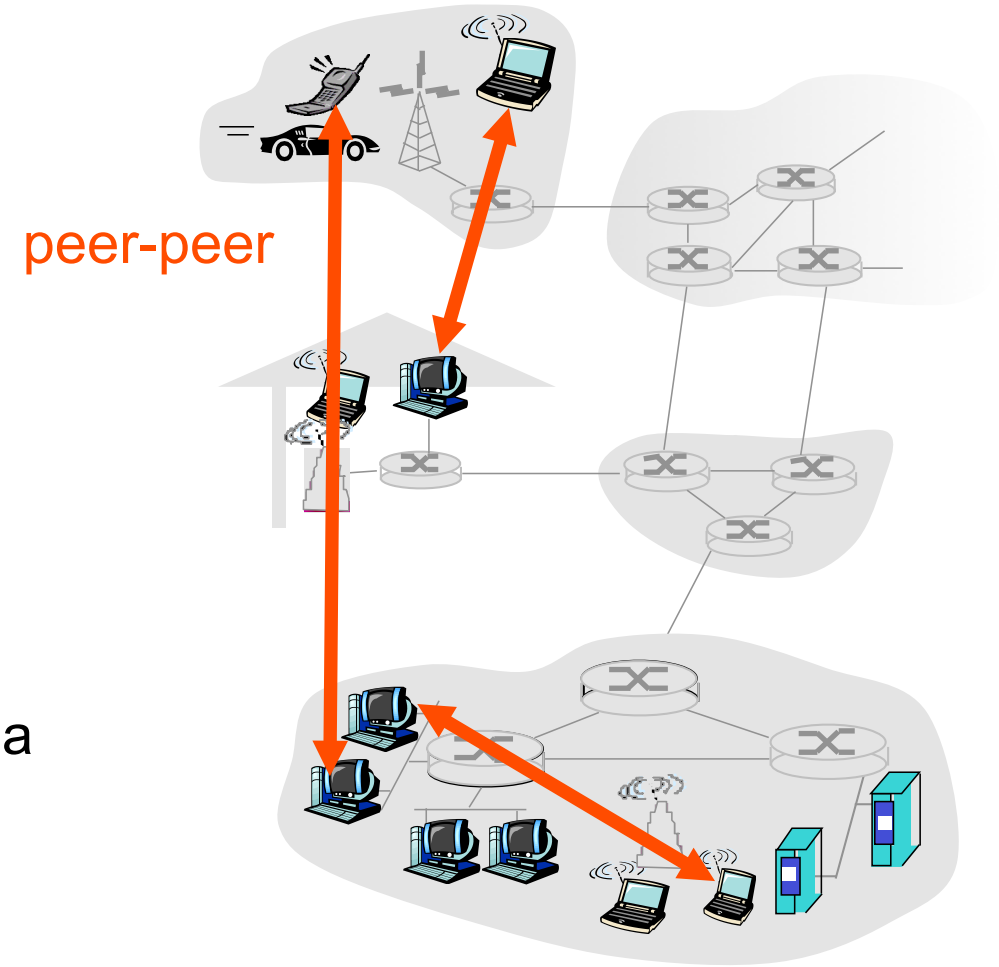
- Server
  - allzeit verfügbarer Host
  - feste IP-Adresse oder per DNS ansprechbar
  - stellt Daten/Dienst bereit
- Client
  - kommuniziert mit dem Server
  - evtl. nicht durchgängig verbunden oder dynamische IP-Adresse
  - Clients kommunizieren **nicht** miteinander
- Server-Farm statt Server
  - wegen Skalierung mehrere Server
  - benötigt eventuell Datenabgleich
- Server ist „Single Point of Failure“



Computer Networking: A Top Down Approach
Jim Kurose, Keith Ross

Peer-to-Peer-Architektur

- Ohne Server
- End-Systeme kommunizieren direkt
- Daten/Dienst verteilt auf End-Systemen
  - kein „Single Point of Failure“
  - hochskalierbar, da gesamte Bandbreite/Rechenkapazität mit jedem Peer steigt
  - aber schwer zu handhaben, da Peers nur zeitlich begrenzt online sind



CoNe Freiburg

Kommunikationsformen in der Anwendungsschicht

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

Hybrid aus Client-Server und Peer-to-Peer

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

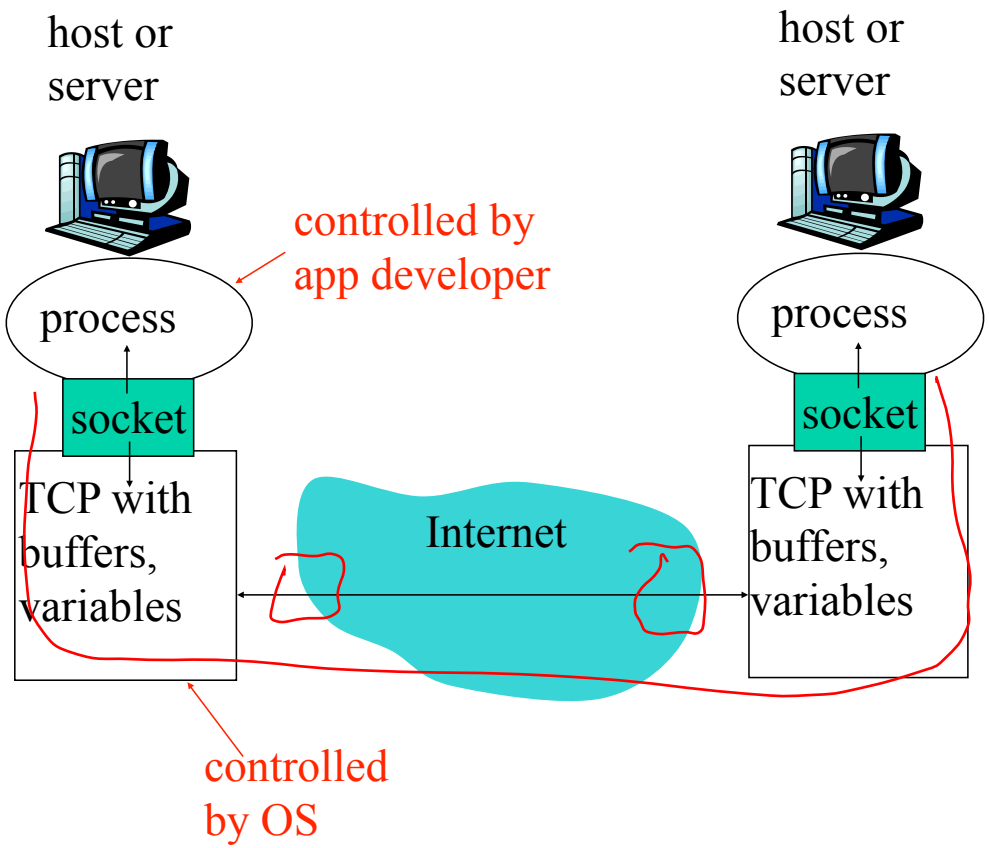
CoNe Freiburg

Kommunizierende Prozesse

- Prozess: Programm auf einem Rechner (Host)
- Prozesse auf demselben Rechner
  - kommunizieren durch Inter-Prozess-Kommunikation über OS
    - gemeinsamer Speicherbereich (z.B. C++ Boost-Interprocess)
    - Datenstrom mit Pipes (in Puffer schreiben/lesen)
- Prozesse auf verschiedenen Rechnern in verteiltem System
  - kommunizieren mit Nachrichten übers Netzwerk
  - Client-Server:
    - Client-Prozess initiiert die Kommunikation
    - Server-Prozess wartet auf Client-Kontakt
  - P2P:
    - haben Client und Server-Prozesse

Sockets

- Sockets (Steckdosen) sind Schnittstelle zwischen Prozess und Netzwerk-Implementierung
- Prozesse senden und empfangen Nachrichten über Sockets
- Sockets sind im Netzwerk über Socket-Adresse (IP und Port) verbunden
  - Prozess vertraut auf Transport-Infrastruktur
- API (Programmierschnittstelle)
  - Wahl des Transport-Protokolls (TCP, UDP)
  - kann bestimmte Parameter wählen



CoNe Freiburg

Anwendungschicht-Programm beschreibt

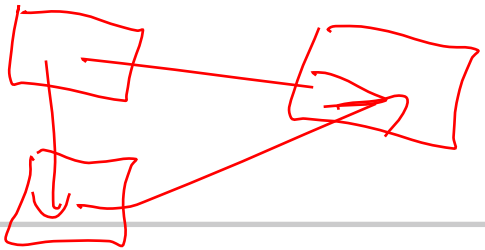
- beschreibt Übertragungsprotokoll mit Nachrichten
  - Nachrichtentyp: z.B. Request, Response
  - Nachrichten-Syntax: Nachrichtenfelder und Zuordnung
  - Nachrichten-Semantik: Bedeutung der Felder
  - Regeln für das Senden und Empfangen von Nachrichten (z.B. auf Request folgt Response)
- Public-domain Protokolle
  - definiert in RFC
  - Standard für Kompatibilität
  - z.B. HTTP, SMTP, BitTorrent
- Proprietäre (unveröffentlichte) Protokolle
  - z.B. Skype, PPStream (Peer-to-Peer-TV)
  - Nutzer muss Software von Hersteller nutzen

Welchen Transport-Service braucht eine Anwendung?

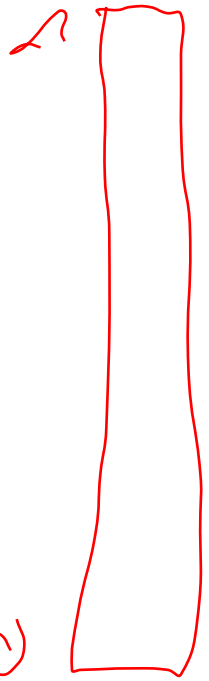
- Transport-Services: TCP, UDP
- Datenverlust
  - einige Anwendungen (z.B. Audio) tolerieren gewissen Verlust
  - andere (z.B. Dateitransfer, Telnet) benötigen 100% verlässlichen Datentransport
- Timing
  - einige Anwendungen (z.B. Internet Telefonie, Spiele) brauchen geringen Delay
- Durchsatz (throughput)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

CoNe Freiburg

Web und HTTP



- Hypertext Transfer Protokoll zur Übertragung von Daten, z.B. Web-Seiten
- Web-Seiten (web page) besteht aus Objekten
- Objekte sind HTML-Datei, JPEG-Bild, Java-Applet, Audio-Datei, Video-Datei, …
- Web-Seite besteht aus Base HTML-Datei mit einigen referenzierten Objekten
- Jedes Objekt wird durch eine URL adressiert
  - Beispiel URL:



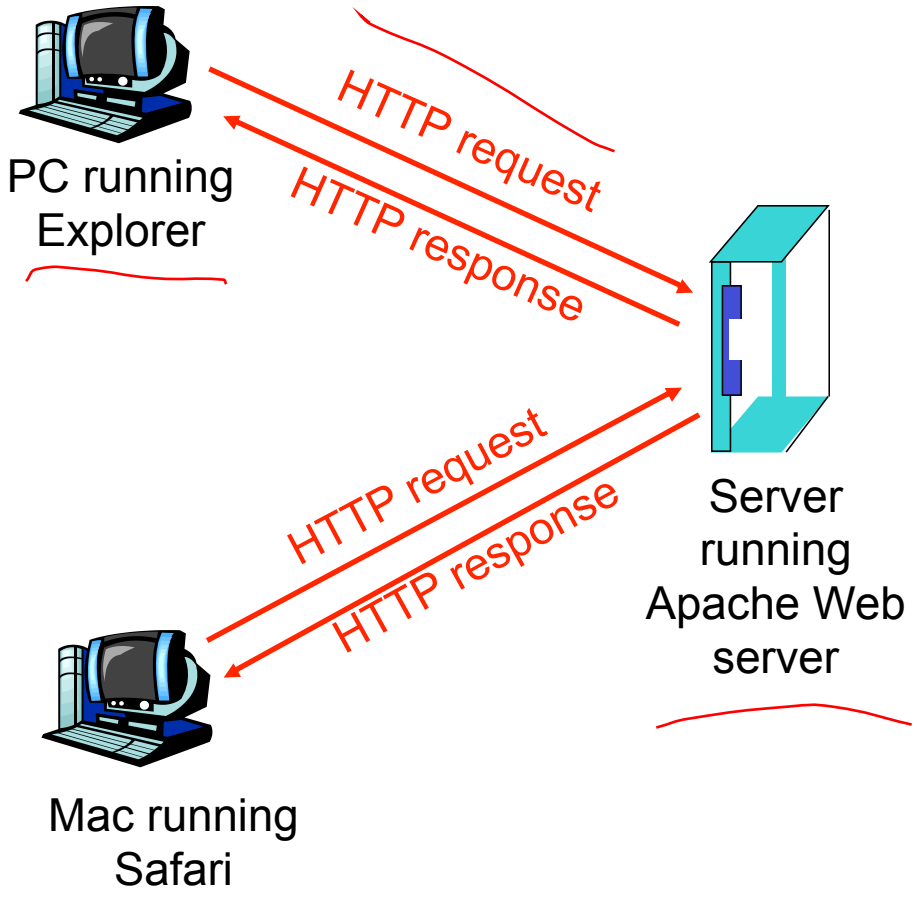
`www.someschool.edu/someDept/pic.gif`

host name | path name

CoNe Freiburg

HTTP-Überblick

- 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



CoNe Freiburg

HTTP-Überblick

- Verwendet TCP (zuverlässiges Protokoll)
- 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

CoNe Freiburg

HTTP-Überblick

- HTTP ist zustandslos (stateless)
  - Server merkt sich nichts über vorige Anfragen
  - jede Anfrage wird gleich behandelt
- 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

CoNe Freiburg

HTTP-Verbindungen

- 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

Nachrichten

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

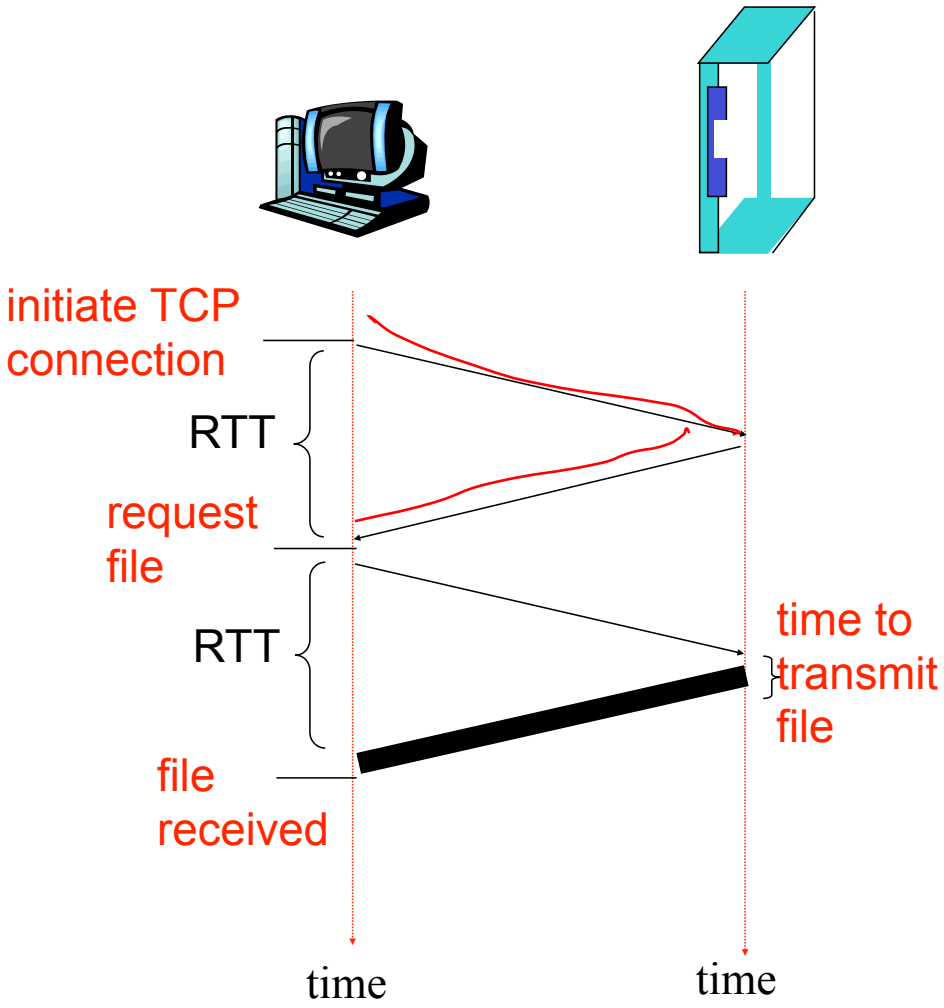
4. HTTP-Server schließt die TCP-Verbindung

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

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
- Zeit = 2 RTT+ transmit time



CoNe Freiburg

Persistentes HTTP

- 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 (mehrere Ports)
- 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 verlinktes Objekt findet
  - höchstens eine Umlaufzeit (RTT) für alle referenzierten Objekte

1 + 11 = 12

2 · 11 = 22 RTT

CoNe Freiburg

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)

CoNe Freiburg

HTTP per Telnet

1. Telnet zum Web-Server

```
telnet www.uni-freiburg.de 80
```

Öffnet TCP Verbindung auf Port 80 (default HTTP Server-Port) von www.uni-freiburg.de.
(alternativ mit PuTTY und Typ Raw)

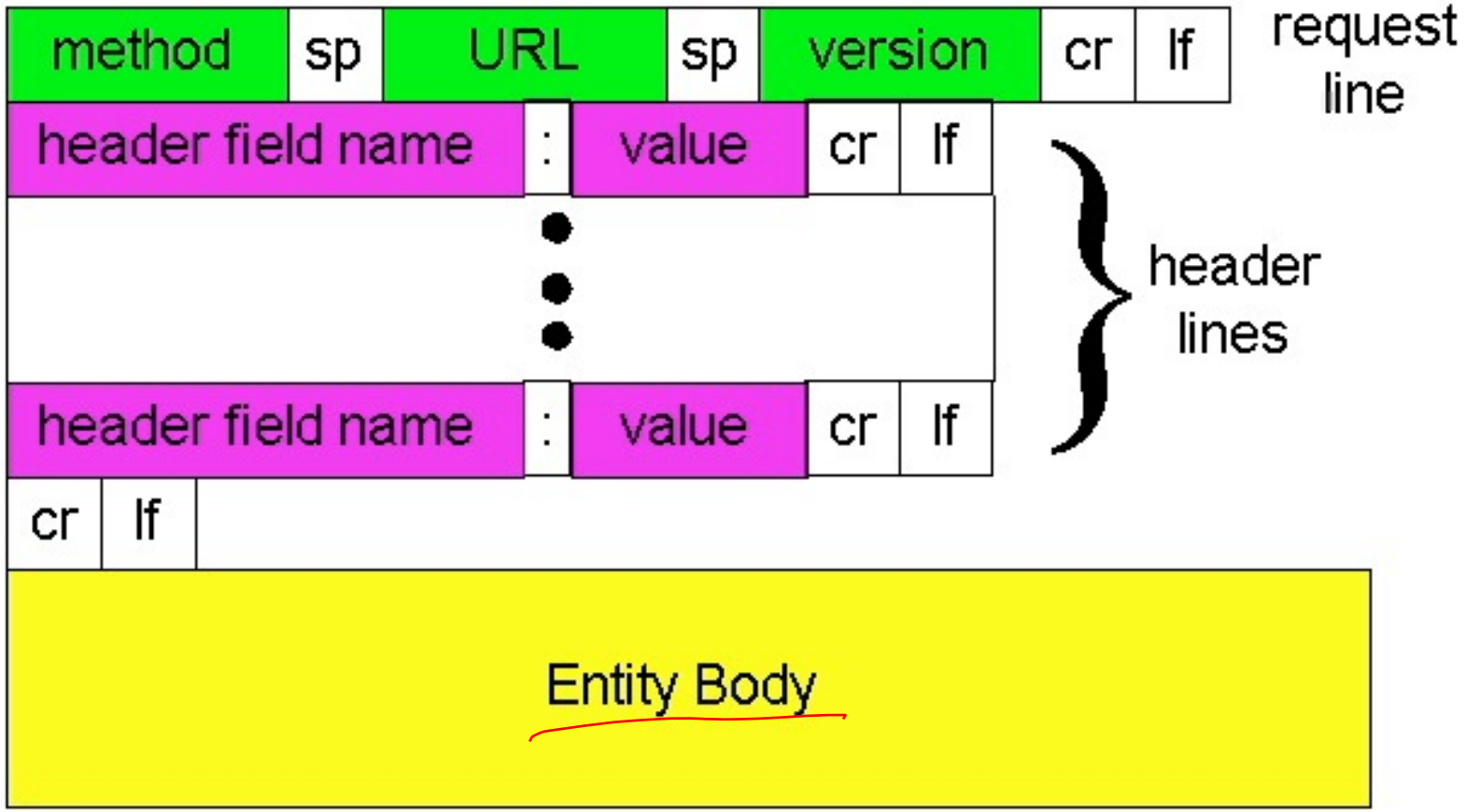
2. Eingabe einer GET HTTP Anfrage:

```
GET /index.html HTTP/1.1
Host: uni-freiburg.de
```

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

3. Was kommt als Antwort vom HTTP server?

HTTP-Request Nachricht: Allgemeines Format



CoNe Freiburg

Upload

- Web-Seiten haben Leerfelder für Eingaben
  - Eingaben zum Server hochladen
  - mit POST-Befehl
    - Eingabe wird im *Body* zum Server hochgeladen
  - URL-Methode mit GET-Befehl
    - Input wird im URL-Feld der Anfrage-Nachricht gesendet:

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

- POST kann auch große Datenmengen wie Bilder im *Body* zum Server hochladen

CoNe Freiburg

Methoden

- 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

WebDAV

HTTP-Antwort Nachricht

Status-Zeile (protocol status code status phrase)

Kopfzeile

Daten, e.g., requested HTML file

```
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 ...
```

CoNe Freiburg

HTTP Antwort-Status

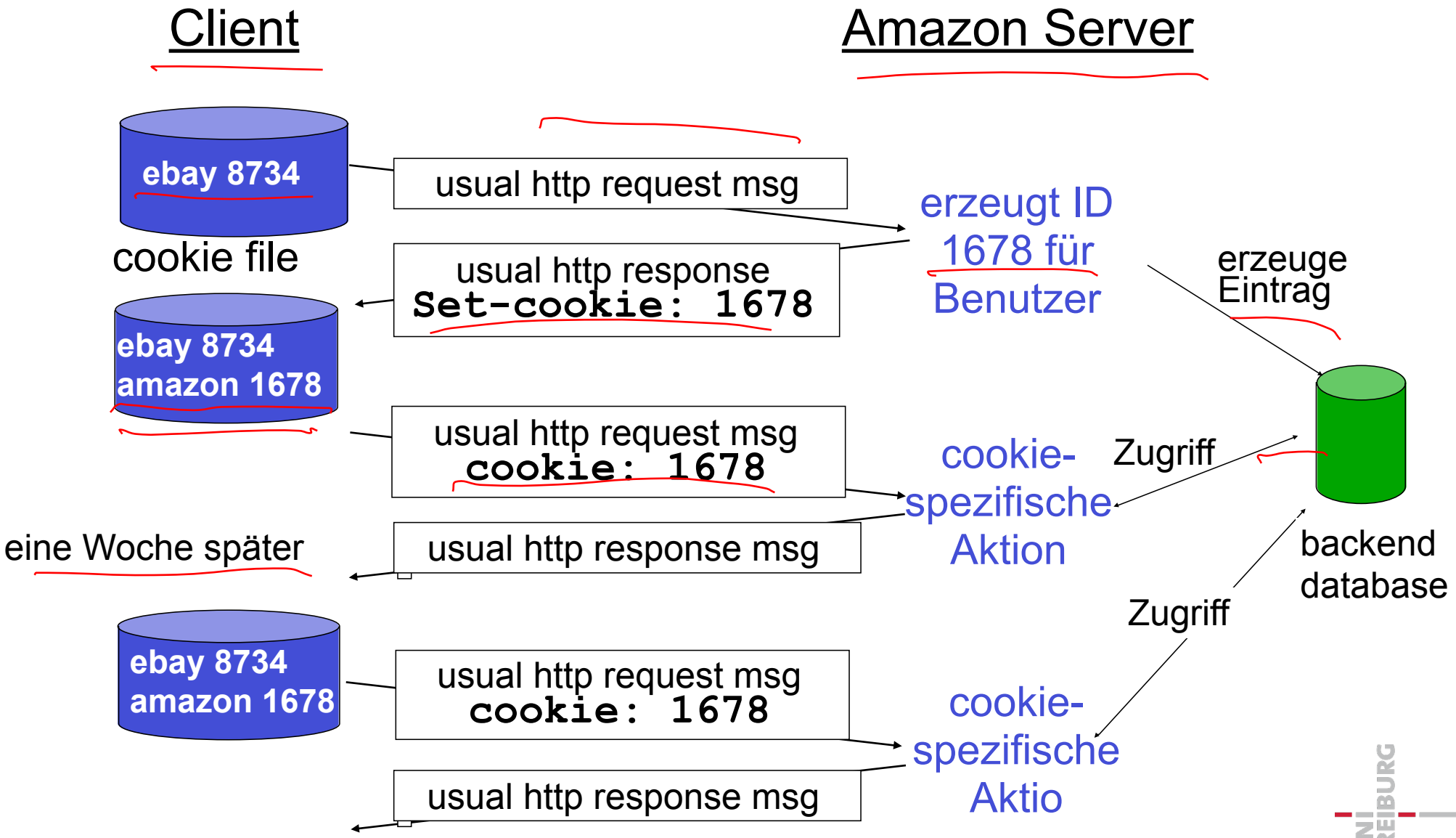
- In der ersten Zeile der Client-Antwort-Nachricht (client response)
- Beispiele:
  - 200 OK
    - Anfrage wird beantwortet in dieser Nachricht
  - 301 Moved Permanently
    - neue Adresse für Objekt
    - Adresse folgt in der Nachricht
  - 400 Bad Request
    - Anfrage wird nicht verstanden
  - 404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

CoNe Freiburg

Benutzerstatus: Cookies

- Zustände für zustandsloses Web-Protokoll HTTP
- Viele Web-Sites verwenden Cookies
- Vier Komponenten
  - (1) Cookie-Datei auf dem Benutzer-Rechner
    - wird vom Web-Browser des Benutzers unterhalten
  - (2) Cookie Eintrag in Datenbank des Servers der Web-Site
  - (3) Cookie Kopf-Zeile in HTTP Antwortnachricht
  - (4) Cookie-Kopf-Zeile in HTTP Anfragenachricht

Cookies: Erzeugen einer Status-Information



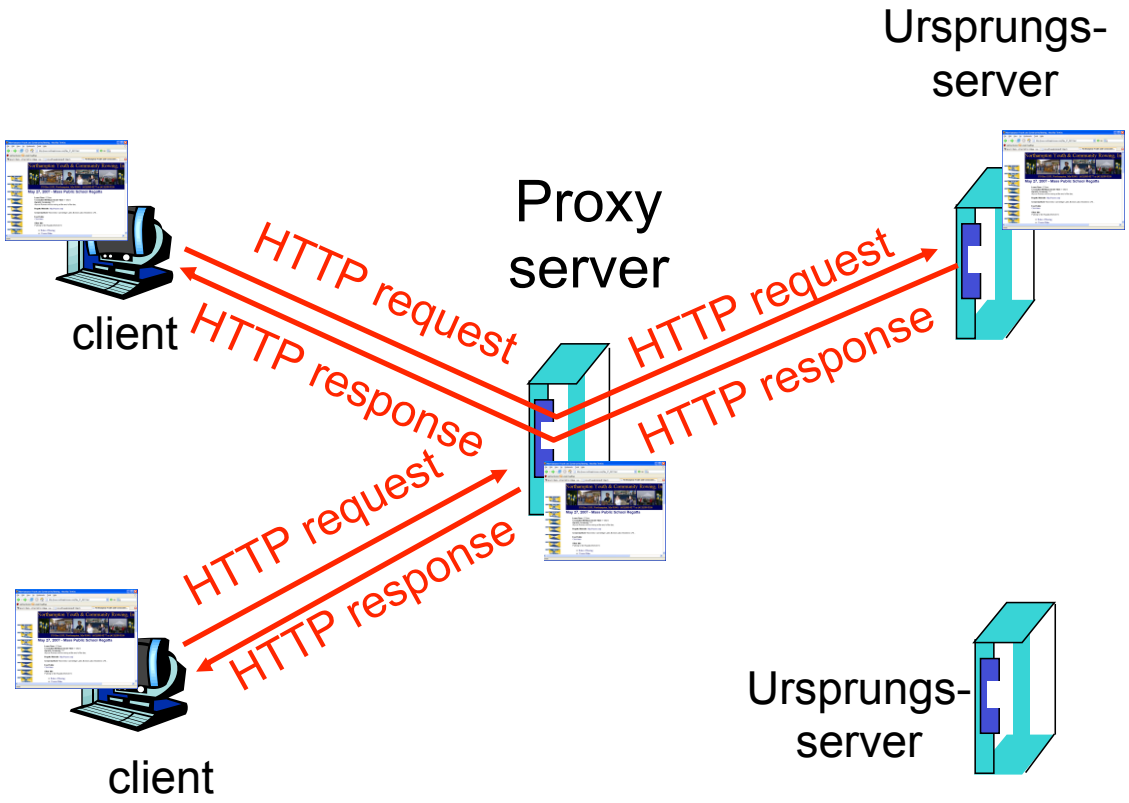
CoNe Freiburg

Cookies

- 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 verknüpfen Informationen verschiedener Anfragen
  - z.B. Name, E-Mail, Kaufverhalten, etc.

Web Caches (Proxy Server)

- Web Cache speichert Web-Seiten von Ursprungsserver zwischen
- Benutzer konfiguriert Browser für Cache
- 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



CoNe Freiburg

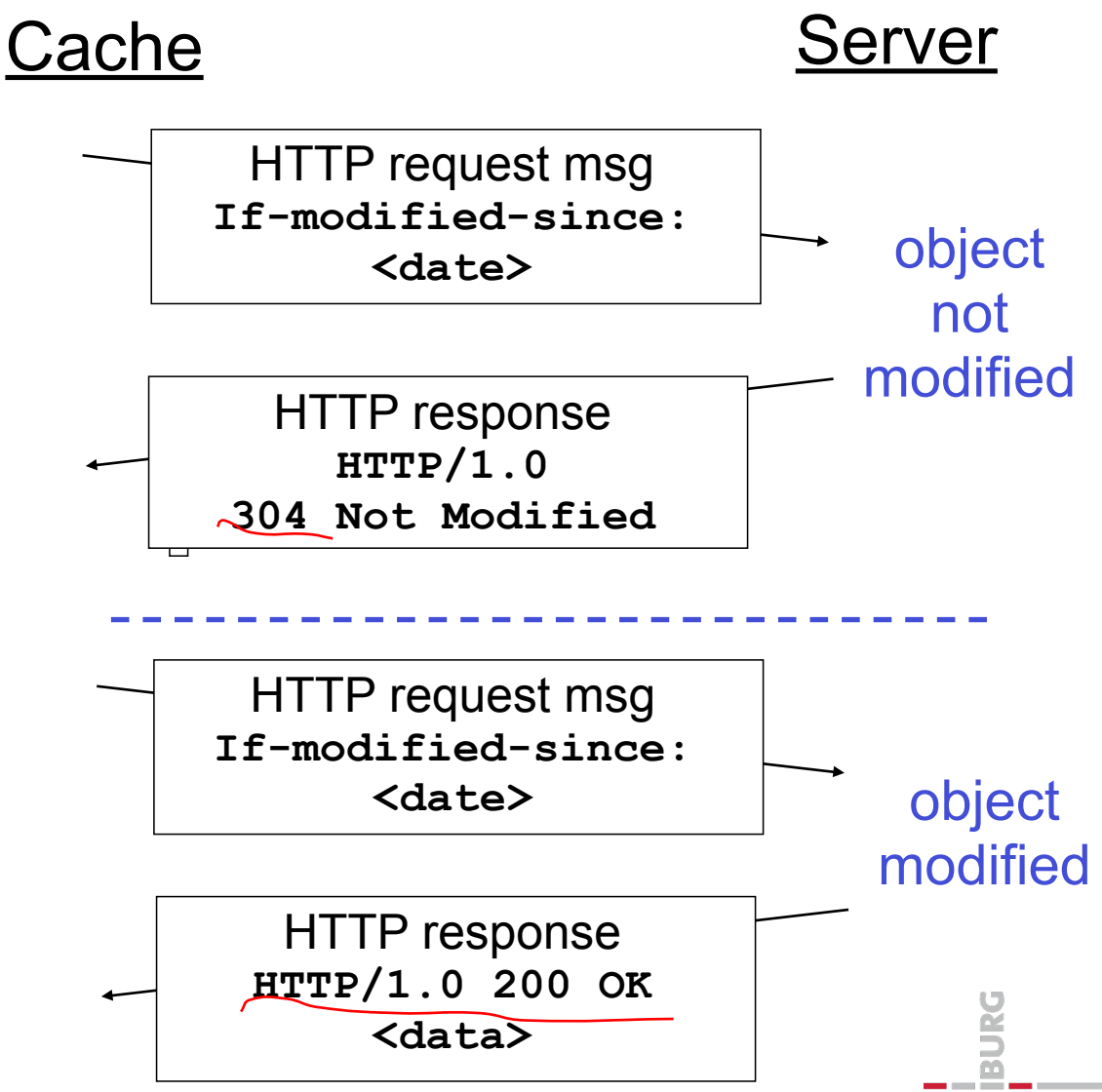
Web-Caching

- 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

CoNe Freiburg

Conditional GET

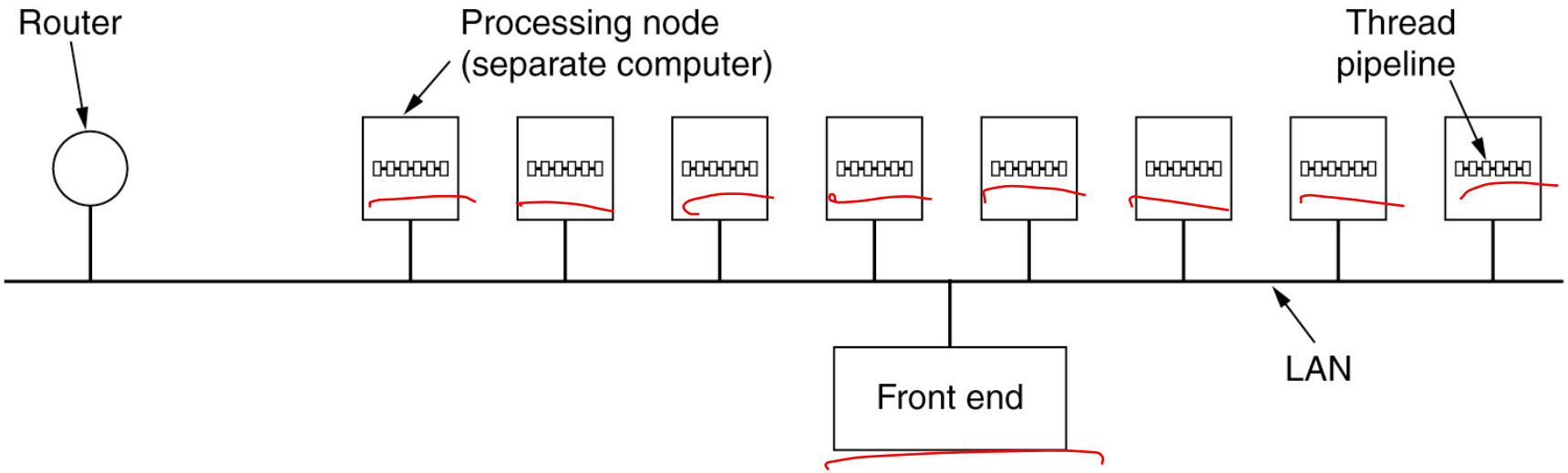
- Ziel: Objekt soll nicht gesendet werden, falls der Cache die aktuelle Version hat
- Cache: gibt den Zeitsempel der gecachten Kopie einer HTTP-Anfrage
  - If-modified-since: <date>
- Server: Antwort enthält kein Objekt, falls, die gecachte Kopie aktuell ist
  - HTTP/1.0 304 Not Modified



CoNe Freiburg

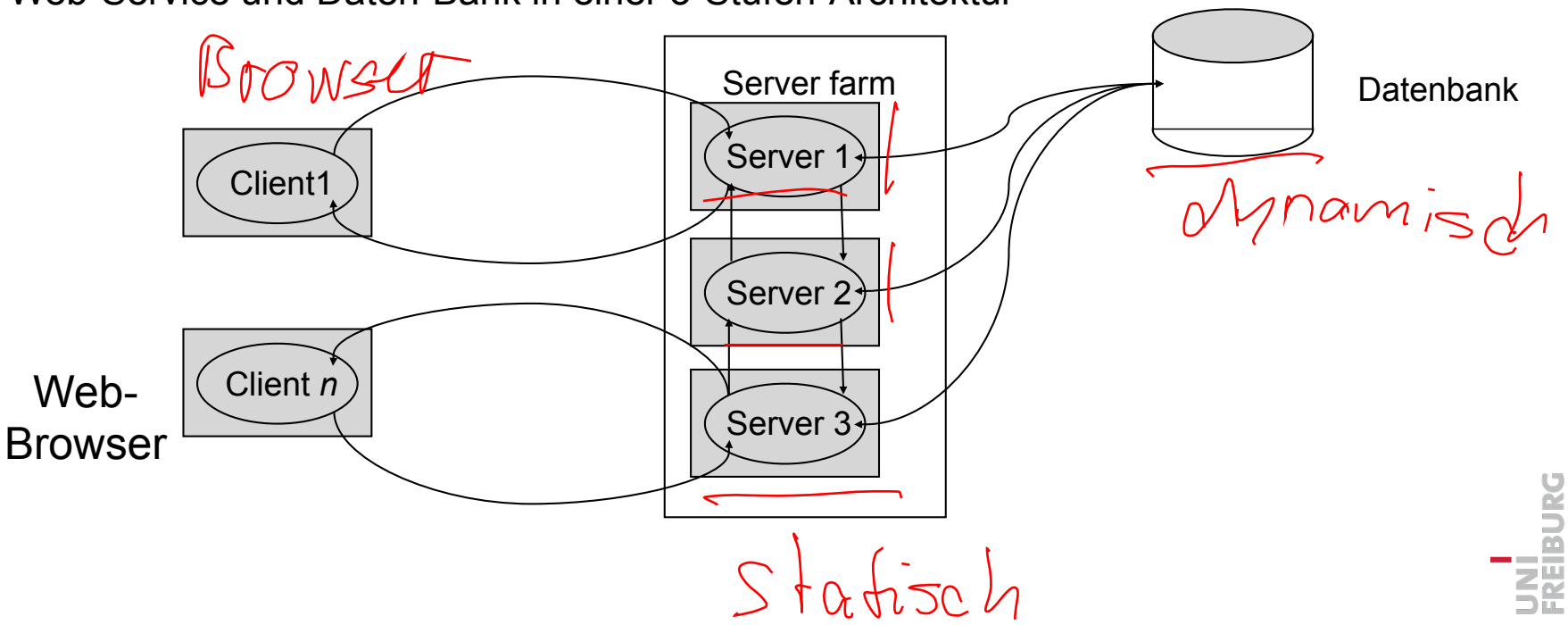
Server-Farm

- 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-Servers und Datenbanken

- Web-Server stellen nicht nur statische Web-Seiten zur Verfügung (z.B. Forum)
  - 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



CoNe Freiburg

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
- Google-Suche verbraucht 0,3 Wattstunden

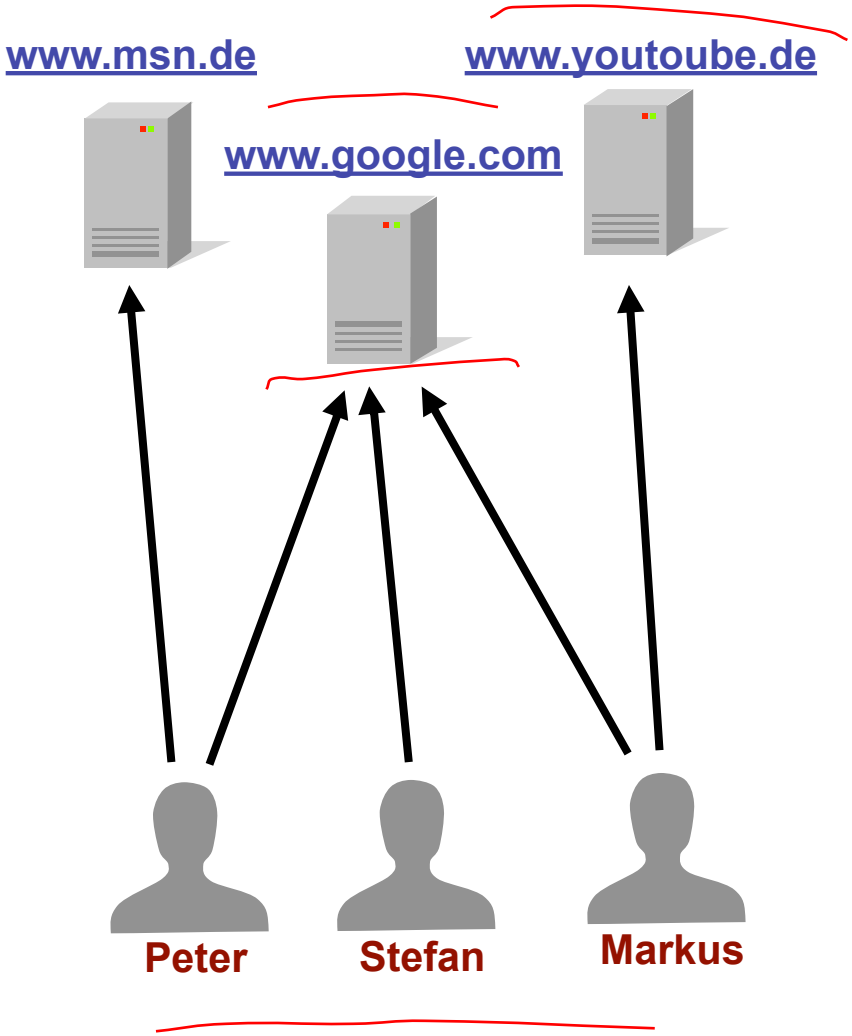
CoNe Freiburg

Content Distribution Networks (CDN)

- 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
  - Cache-Anfragen werden auf die regional und lastmäßig bestgeeigneten Server umgeleitet
- Beispiel Akamai:
  - Seit 1998 mit 150.000 Server in 92 Ländern mit Kunden u.a. Apple, Facebook, Microsoft, eBay
  - Durch verteilte Hash-Tabellen ist die Verteilung effizient und lokal möglich

WWW-Lastbalancierung

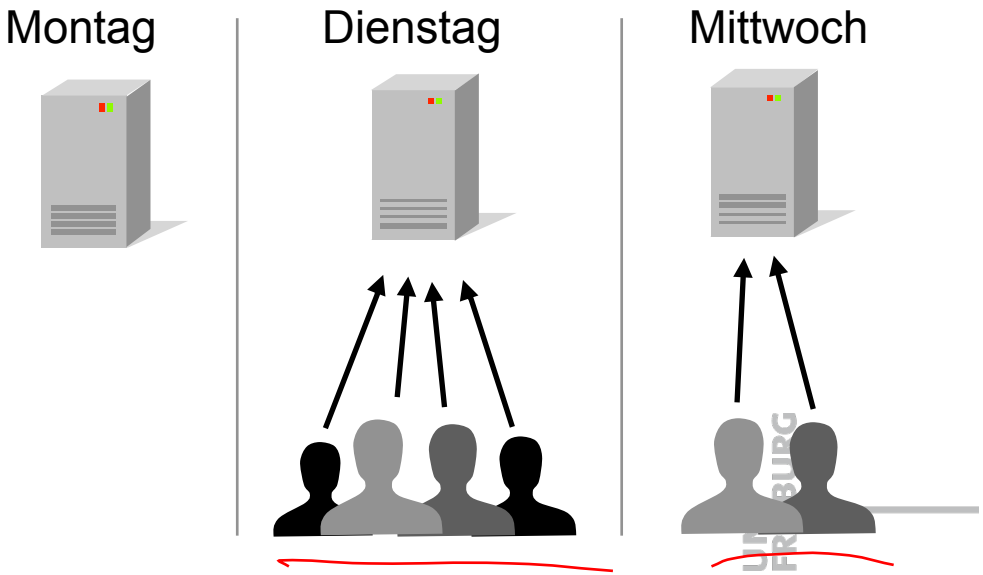
- Für Surfen im Web typisch:
  - Web-Server bieten Web-Seiten an
  - Web-Clients fordern Web-Seiten an
- Benutzer-Zugriffe belasten Web-Server hinsichtlich:
  - Übertragungsbandbreite
  - Rechenaufwand (Zeit, Speicher)



Lastanforderungen

- Einige Web-Server haben immer hohe Lastanforderungen
  - Z.B. Suchmaschinen, Nachrichten-Seiten
  - Für permanente Anforderungen müssen Server entsprechen ausgelegt werden

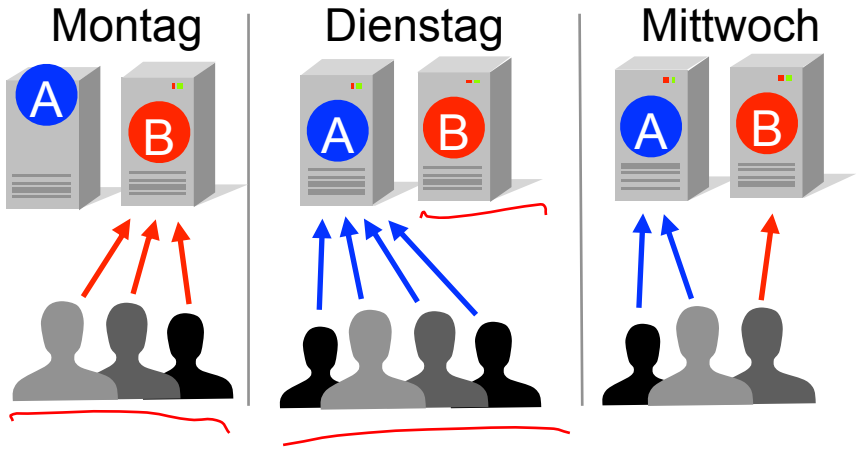
- Andere leiden unter hohen Fluktuationen
  - z. B. bei besonderen Ereignissen:
    - fifa.com (Fussball-EM)
    - t-mobile.de (iPhone 6 Einführung)
  - Server-Erweiterung nicht sinnvoll
  - Bedienung der Anfragen aber erwünscht



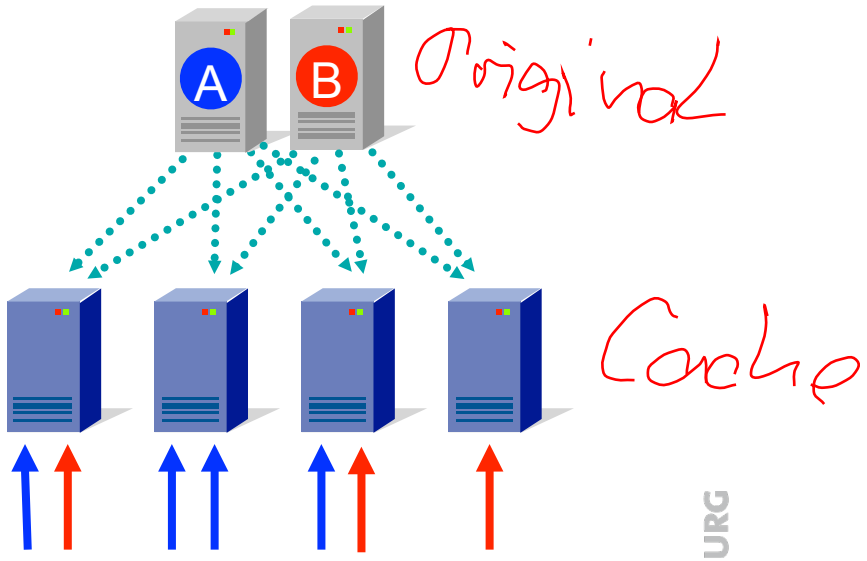
CoNe Freiburg

Lastbalancierung im WWW

- Zeitliche Fluktuationen betreffen meistens einzelne Server



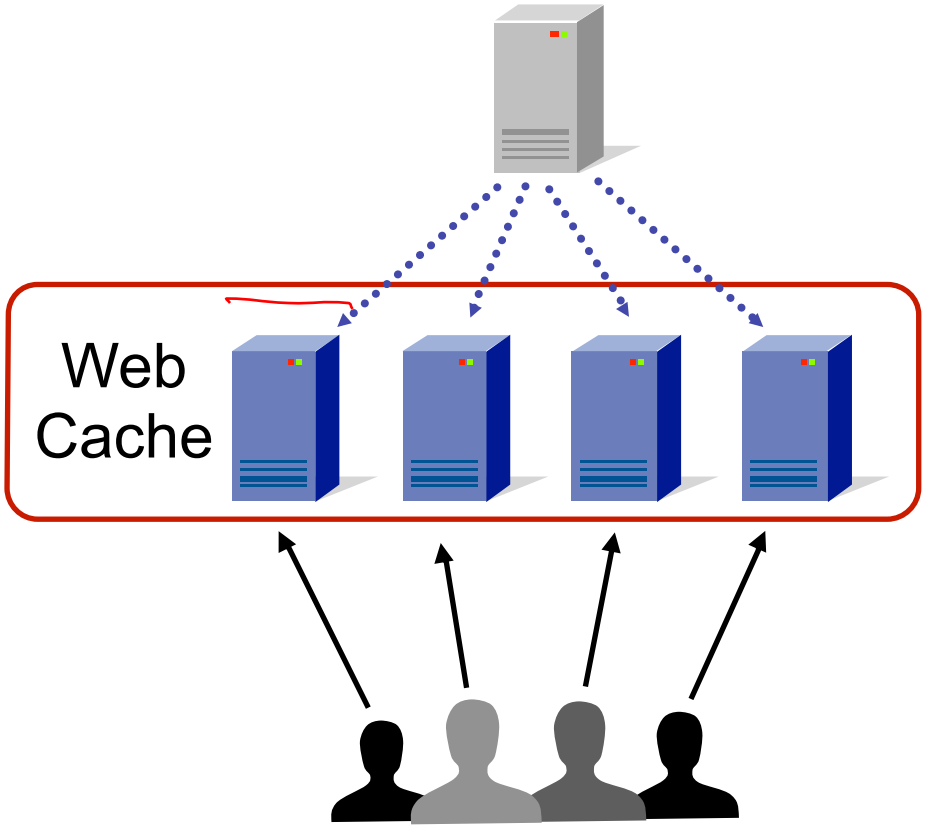
- (Kommerzielle) Lösung
  - Dienstleister bieten Ausweich-(Cache-)Server an
  - Viele Anforderungen werden auf diese Server verteilt
- Aber wie?



CoNe Freiburg

Web-Caching

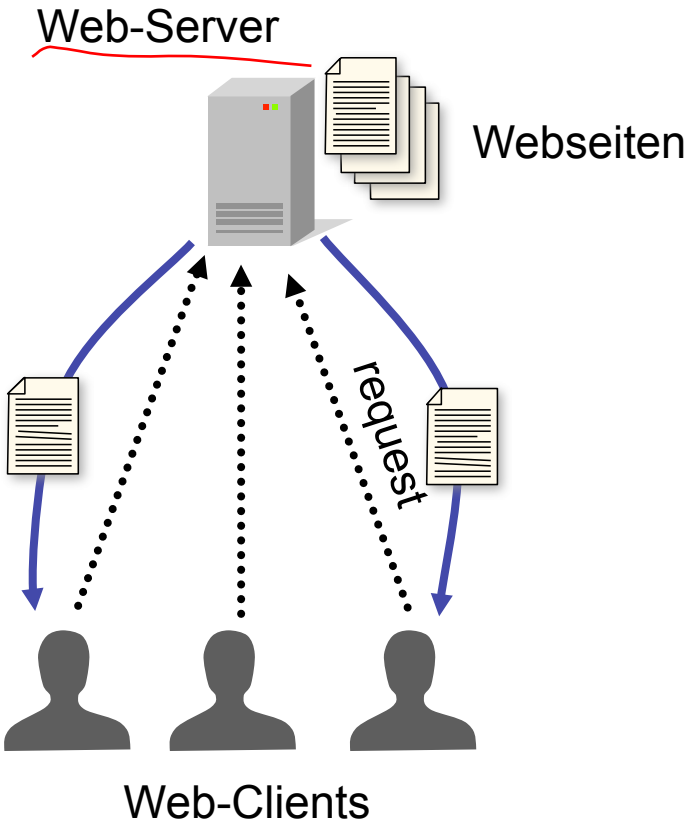
- wissenschaftliche Arbeit:
  - Autoren: Leighton, Lewin, et al.
    - Leighton und Lewin (MIT) gründeten Akamai 1997
  - *Titel: Consistent Hashing and Random Trees:*
    Distributed Caching Protocols for Relieving
    Hot Spots on the World Wide Web
  - Konferenz: STOC'97 (Symposium on Theory of Computing)
- Passen bestehende Verfahren für dynamische Hash-Funktionen an WWW-Anforderungen an
- Hash-Funktion bildet Daten gleichmäßig auf Server ab



CoNe Freiburg

Ausgangssituation

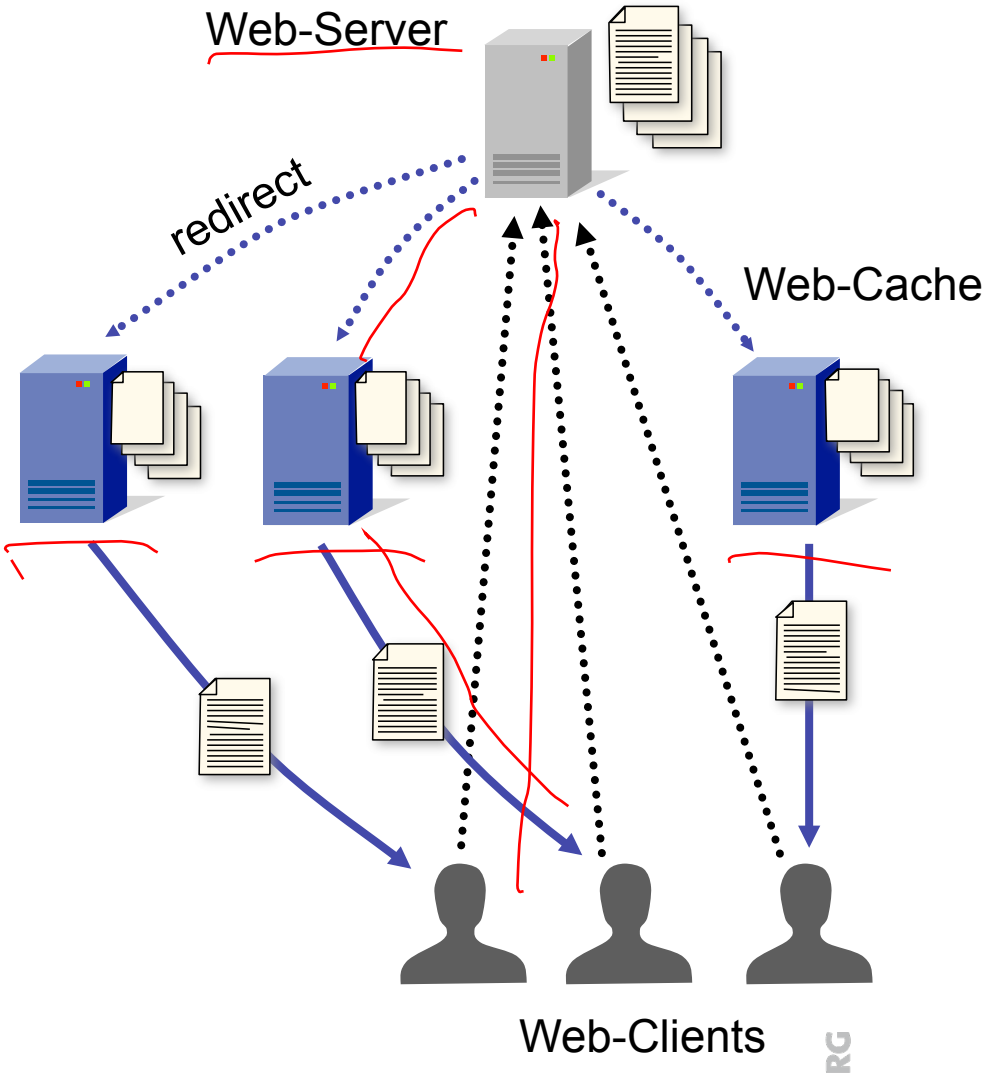
- Ohne Lastbalancierung:
  - Jeder Browser (Web-Client) belegt einen Web-Server für eine Web-Site
- Vorteil:
  - Einfach
- Nachteil:
  - Der Server muss immer für den Worst-Case ausgelegt werden



CoNe Freiburg

Site Caching

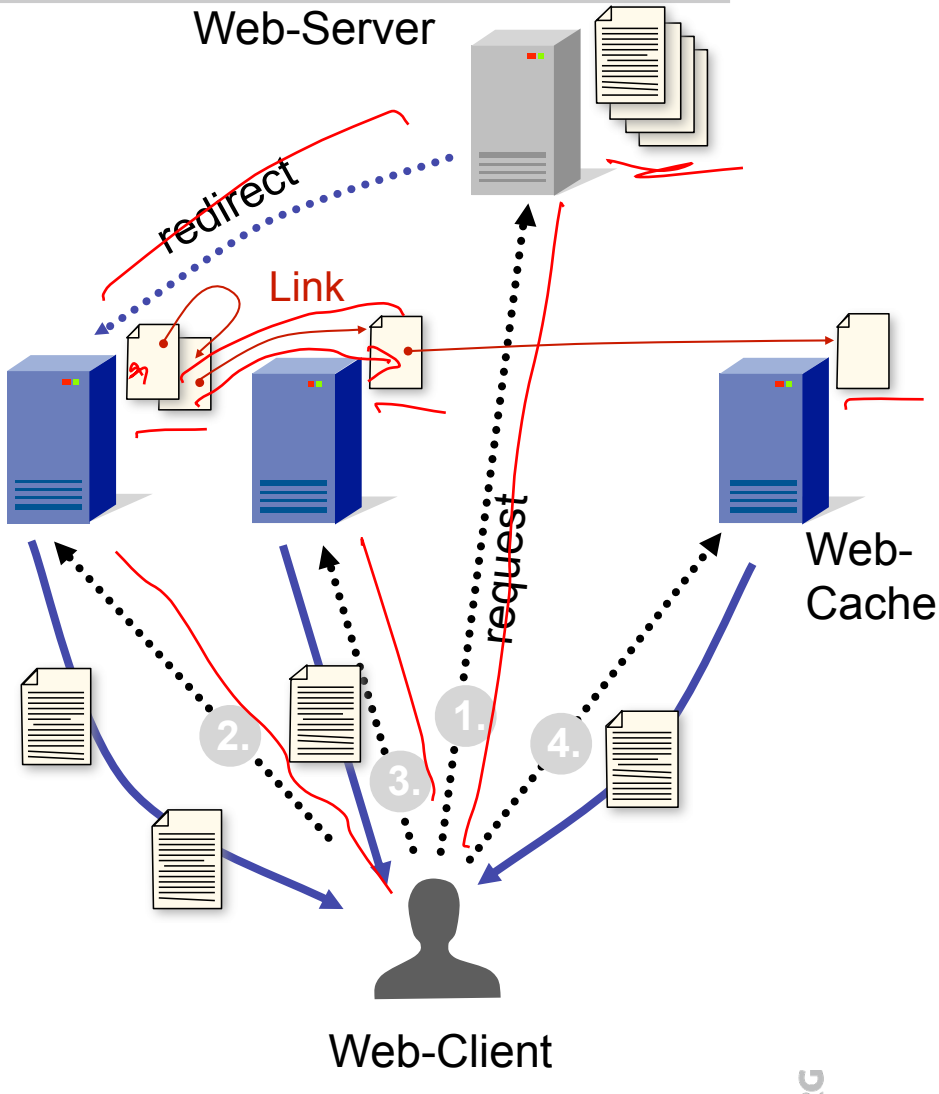
- Ganze Web-Site wird auf verschiedene Web-Caches kopiert
- Browser fragt bei Web-Server nach Seite
- Web-Server leitet Anfrage auf Web-Cache um (redirect)
- Web-Cache liefert Web-Seite aus
- Vorteil:
  - Gute Lastbalancierung für Seitenverteilung
- Nachteil:
  - Großer Overhead durch vollständige Web-Site-Replikationen



UNI FREIBURG

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:
  - weniger Daten-Replikation durch Verteilung auf Servern
- Nachteil:
  - Lastbalancierung nur implizit möglich
  - Hohe Anforderung an Caching-Algorithmus (etwa gleichmäßige Datenmenge, Anfragehäufigkeit)

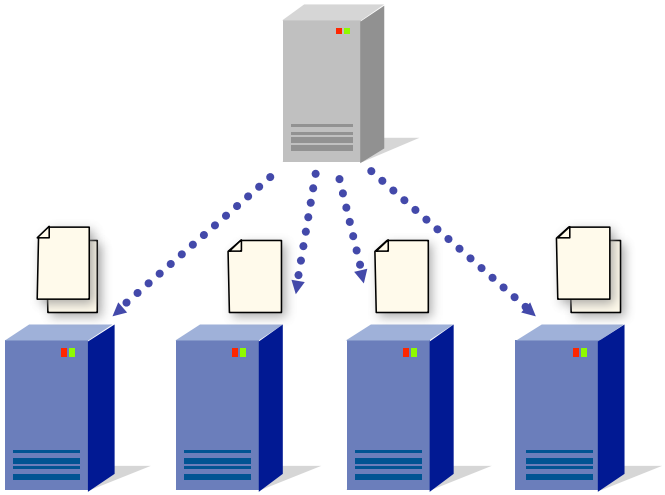


Anforderungen an Caching-Algorithmus

CoNe Freiburg

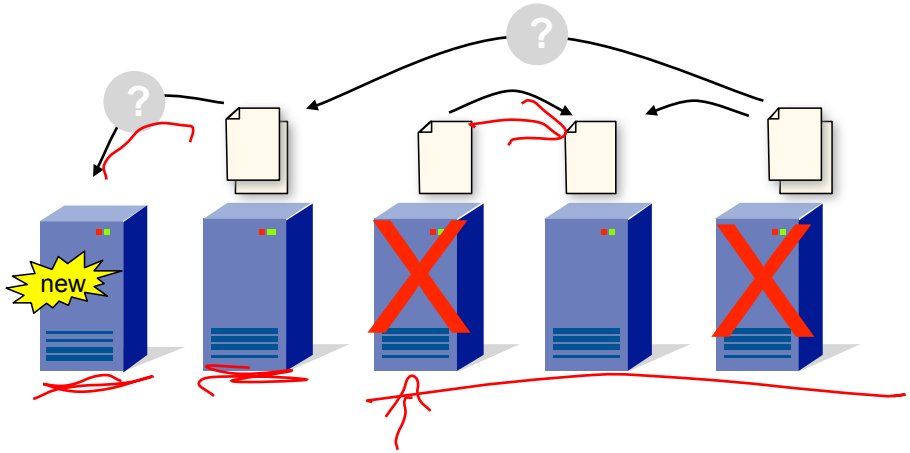
Balance

Gleichmäßige Verteilung der Seiten



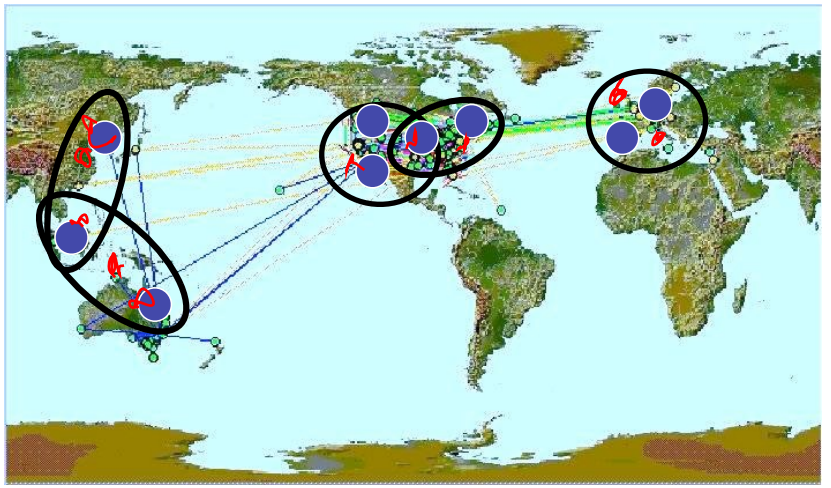
Dynamik

Effizientes Einfügen/Löschen von Web-Cache-Servern



Views

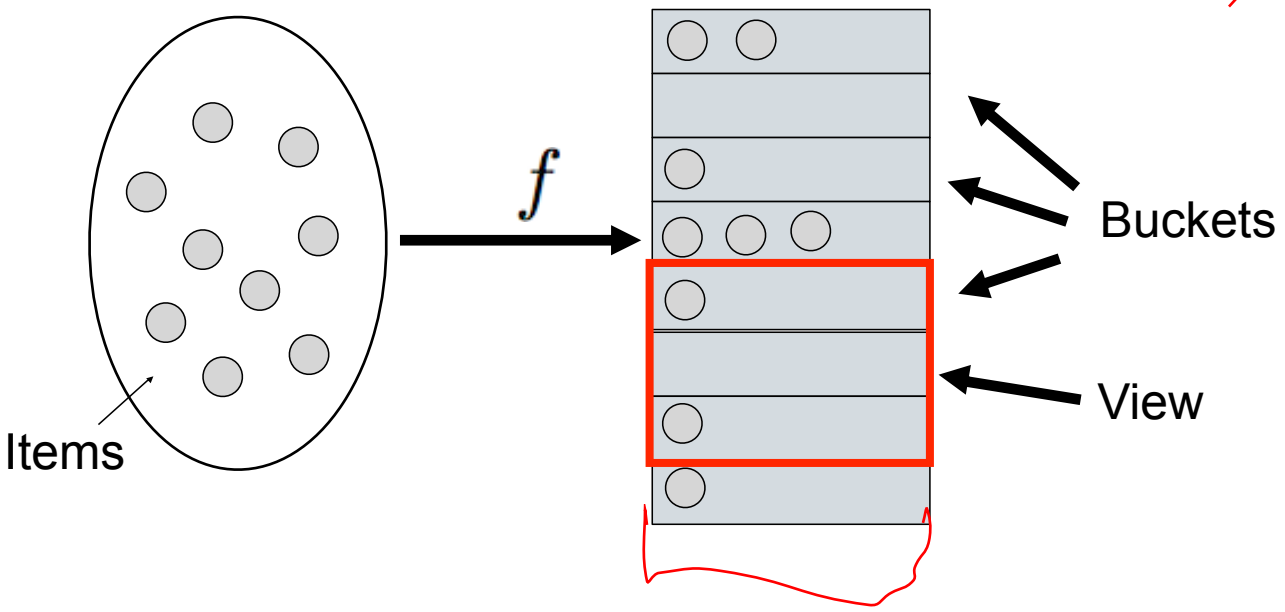
Web-Clients „sehen“ unterschiedliche Menge von Web-Caches



UNI FREIBURG

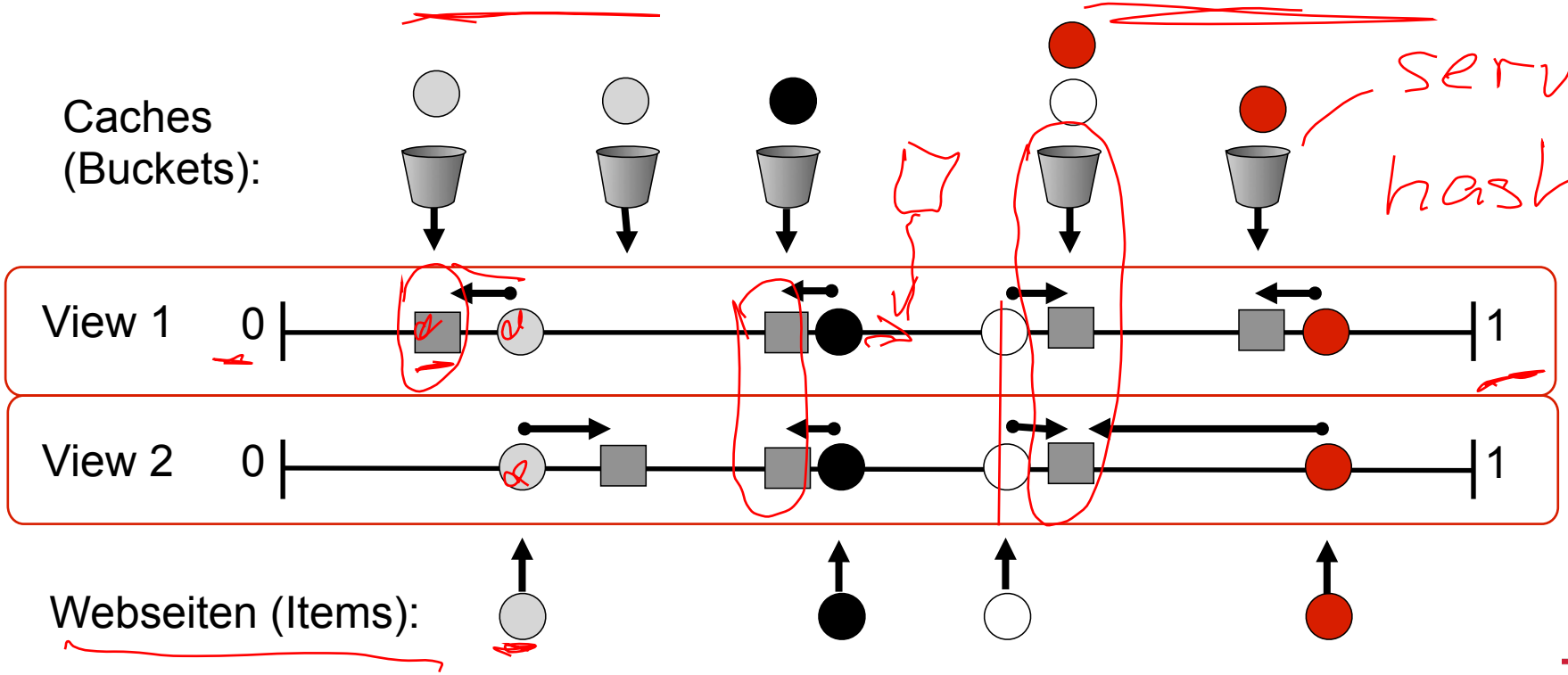
Ranged Hash-Funktionen oder Verteilte Hash-Funktionen

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



Distributed Hash Tables als Lösung

- jeder View hat einen Intervall [0,1)
- Caches werden auf Intervall [0,1) gehasht und Teil der Views zugeordnet
- Webseiten werden auf Intervall [0,1) gehasht und nächstliegendem Server in jedem View zugeordnet



CoNe Freiburg

Anforderungen an Ranged Hash-Funktionen

- Monotonie
  - nach dem Hinzufügen neuer Caches (Buckets) sollten keine Seiten (Items) zwischen alten Caches verschoben werden
- Balance
  - Alle Caches eines Views 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

UNI FREIBURG

CoNe Freiburg

IIF INSTITUT FÜR INFORMATIK FREIBURG

Systeme II

4. Die Anwendungsschicht

Thomas Janson°, Kristof Van Laerhoven*, Christian Ortolf°

Folien: Christian Schindelhauer°

Technische Fakultät

°: Rechnernetze und Telematik, *: Eingebettete Systeme

Albert-Ludwigs-Universität Freiburg

Version 13.05.2015

UNI FREIBURG