Freie Universität Berlin

Netzbasierte Informationssysteme
Caching und Datenaufkommen

Prof. Dr.-Ing. Robert Tolksdorf
Freie Universität Berlin
Institut für Informatik
Netzbasierte Informationssysteme
mailto: tolk@inf.fu-berlin.de
http://www.robert-tolksdorf.de

Caching im Web

Überblick

- Caches
- Caching Architekturen
- Cache Füllung
- Cache Ersetzung
- Cache Kohärenz

Internet Verkehr

- Zwei Messungen 10.1.05 beim ISP sprint in USA [http://ipmon.sprint.com/packstat/packet.php?050110]

Application Breakdown

Category	Packets (%)	Bytes (%)	Flows (%)
Web	51.20	58.08	34.70
File Sharing	5.89	6.19	7.50
FTP	0.65	1.62	0.41
Email	4.21	4.03	2.93
Streaming	1.82	0.69	4.03
DNS	1.36	0.55	4.96
Games	0.00	0.00	0.00
Other TCP	16.24	10.46	19.99
Other UDP	7.02	7.29	14.06
Not TCP/UDP	11.60	11.09	11.42

Application Breakdown

Category	Packets (%)	Bytes (%)	Flows (%)
Web	42.19	60.86	26.48
File Sharing	8.02	6.17	11.76
FTP	0.65	0.35	0.52
Email	3.46	4.09	1.92
Streaming	5.76	3.56	5.19
DNS	0.96	0.27	3.55
Games	0.00	0.01	0.00
Other TCP	21.51	11.66	34.90
Other UDP	6.44	4.97	6.79
Not TCP/UDP	11.00	8.06	8.89

Caching

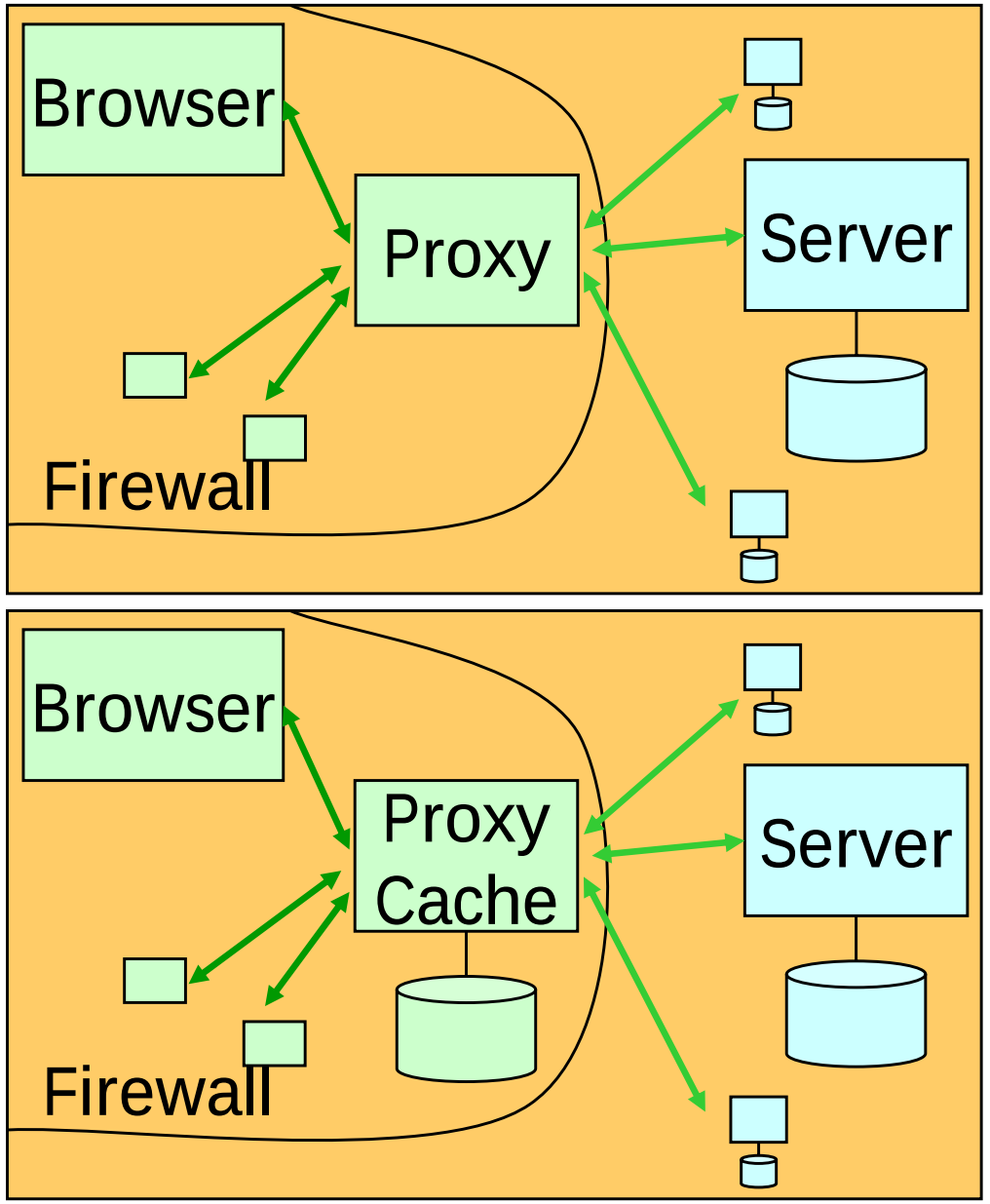
- Ursprünglich aus Rechnerarchitektur:
  - CPU schneller als Hauptspeicher
  - ➔ Daten in schnellem Zwischenspeicher, dem *Cache* halten
- Ziel im Web: Netzwerklatenz kaschieren
  - Klient schneller als Netz (+Server)
- Grundbegriffe:
  - Gesuchte Daten zwischengespeichert vorgefunden: *Hit / Treffer*
  - Gesuchte Daten nicht gefunden: *Miss / Fault / Fehler*
  - Bei Fehler nachgeladenes Originaldatum in Cache gespeichert
  - ➔ Annahme: Mehrere Zugriffe zeitlich gruppiert
  - Oft: Block um gesuchte Daten in Cache geholt
  - ➔ Annahme: Zugriffe örtlich gruppiert

Gütemaße

- Hit Rate:
  Anteil der aus dem Cache geholten Objekte an Gesamtzahl der angefragten Objekte
- Weighted Hit Rate:
  Anteil der Gesamtgröße in Bytes der aus dem Cache geholten Objekte an Gesamtgröße der angefragten Objekte
- Time:
  Mittlere Antwortzeit beim Nutzer

Proxies und Caching

- *Proxy/Stellvertreter* anstelle vom Klienten
  - Leitet HTTP Anfrage von Klienten an Server / andere Proxies weiter
  - Tritt für den Server als Klient auf
  - Leitet Antwort an Klienten weiter
- *Proxy Cache*
  - Agiert auch als Cache
  - ➔ Annahme: Zugriffe organisatorisch gruppiert



Vorteile

- Vorteile von (Proxy-)Caching
  - Netzlast kann effektiv gesenkt werden
  - Netzlatenz für Nutzer sinkt
    - Übertragungszeit sinkt, wenn Objekte im (netztopologisch) nahen Cache gefunden werden
    - Nicht gefundene Objekte werden schneller geholt wegen
      - geringere Netzlast auf dem Weg zum Server
      - geringere Last beim Server

      [http://www.heise.de/newsticker/meldung/80812]
    - "~~eight~~ second rule": Web Seite muss innerhalb von ~~8~~ 4 Sekunden angezeigt werden oder Nutzer verlieren Interesse an Site (bzw. Kauf...)
  - Serverlast sinkt wegen geringerer Zugriffszahl
  - Verfügbarkeit von Objekten steigt
  - Proxies lassen Nutzungsanalysen zu

Nachteile

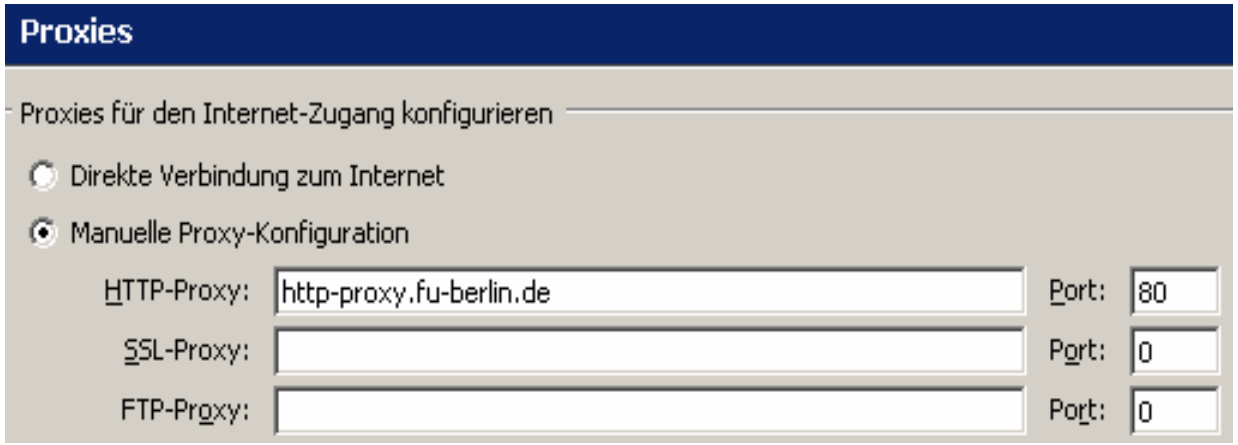
- Nachteile von (Proxy-)Caching
  - Cache-Inhalt muss nicht konsistent mit Originaldaten sein
  - Bei einem Cache-Fehler steigt die Netzlatenz
    (➔ Hit-Rate maximieren, Miss-Kosten minimieren)
  - Proxy wird zum Engpass für Klienten
  - Proxy wird zum Single-point-of-failure
  - Proxy-Cache senkt Hit-Raten (und anderen Maße) beim Server
    (➔ Server können versuchen, Caching zu verhindern)
  - Unnötig bei Seiten die nur einmalig geladen werden

Notwendigkeit

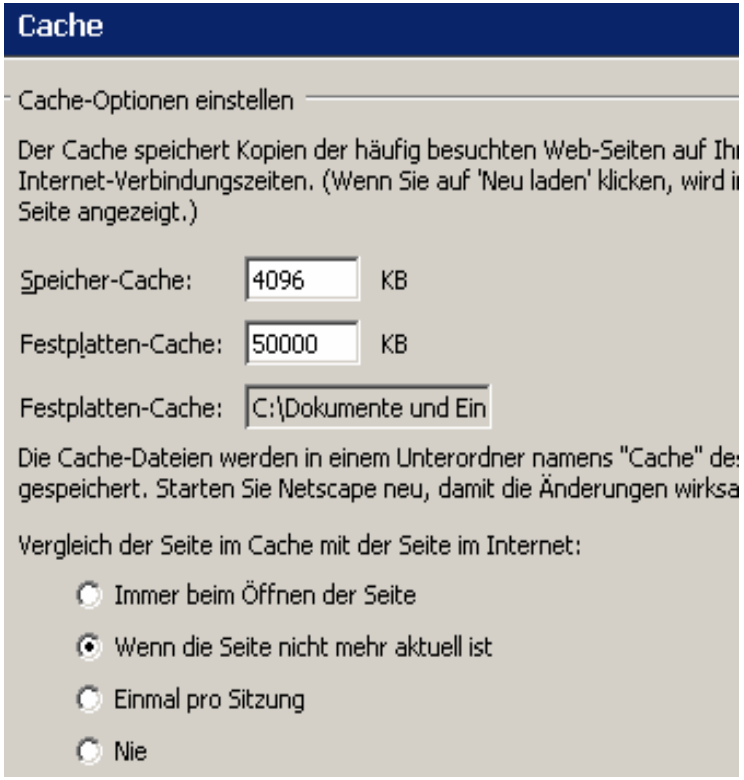
- Netznutzung verursacht immer Kosten
- Netzlatenzen werden immer schwanken
- Entfernungen im Netz werden durch mehr Geräte größer
- Bandbreite der Inhalte wird immer steigen
- Populäre Server sind immer überlastet
- Netzkosten sind größer als Rechenkosten

Browser Konfiguration

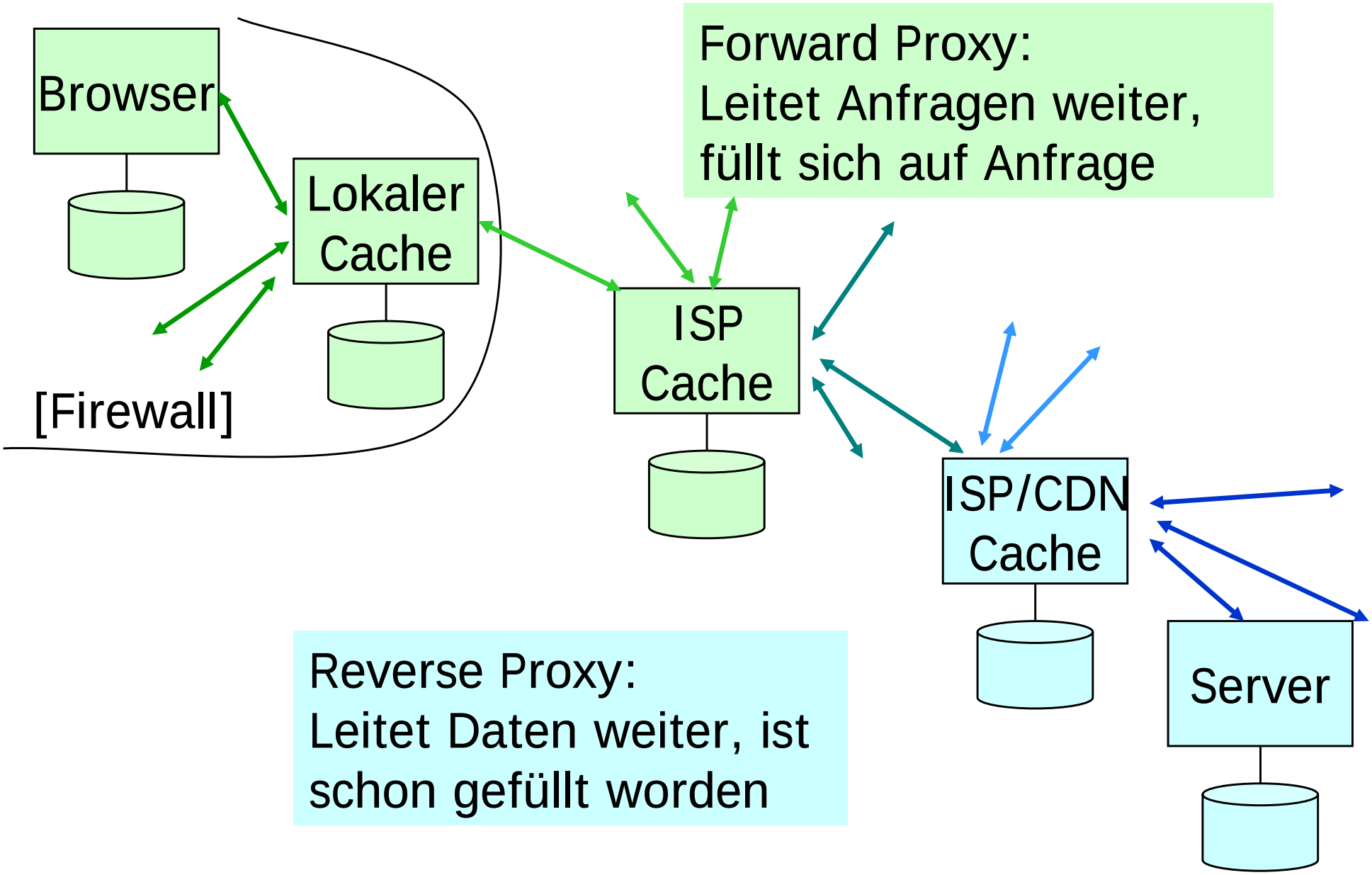
- Browser kann zu externem Proxy-Cache gerichtet werden



- Browser hat aber auch selber schon einen lokalen Cache (Speicher / Disk)



Caches im Web allgemein



Caches

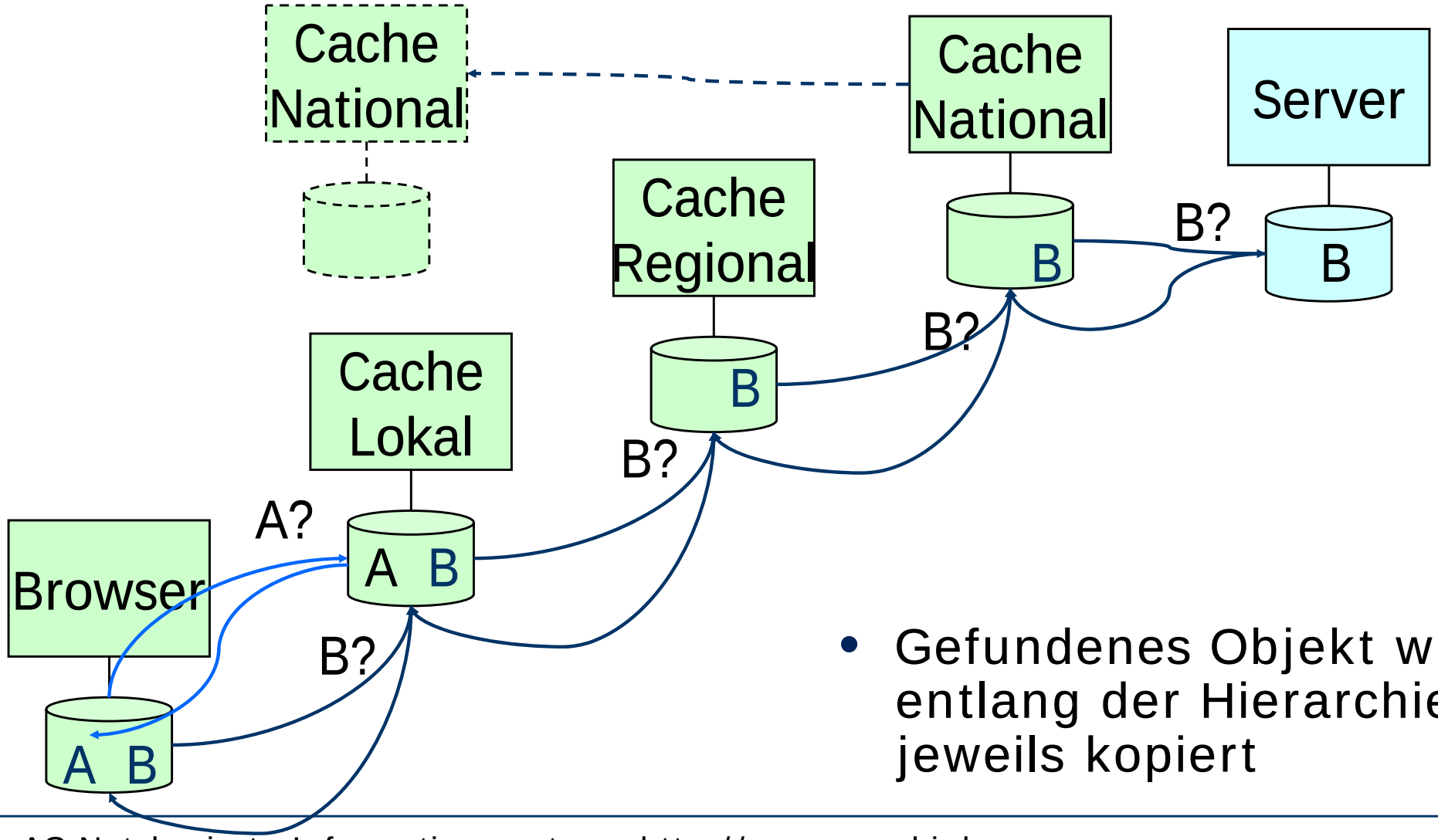
- Forward Proxy Caches
  - Stellvertreter für Klienten
  - Vermeidung der Netznutzung
  - Browser-Cache: Lokal, nutzerspezifisch
  - Proxy-Cache: Organisationsweit, gruppenspezifisch
  - ISP-Cache: Teilnetzweit
- Reverse Proxy Caches („HTTP Accelerators“)
  - Stellvertreter für Server
  - Vermeidung der Servernutzung
  - Content Delivery Network-Cache: Anbieterweit, sitespezifisch
  - Server Cache: Serverweit, sitespezifisch

Design-Ziele

- *Senkung der Netzlatenz* für Nutzer
- *Robustheit* gegenüber Fehlern
  - Toleranz zu einzelnen Ausfällen
  - Kontrollierter Leistungsrückgang
  - Einfaches Wiederherstellung nach Ausfällen
- *Transparenz* des Cachings für den Nutzer
- *Skalierbarkeit* mit Web-Wachstum
- *Effizienz* der Ressourcennutzung
  - Minimale zusätzliche Netzlast
  - Erhaltung optimaler Ausnutzung von Ressourcen
- *Adaptivität* zum Nutzerverhalten und Netzzustand
- *Stabilitätserhaltung* der Gesamtnetzes
- *Lastverteilung* entlang den beteiligten Komponenten
- *Einfachheit* als Voraussetzung weiter Verbreitung

Cache Architekturen – Hierarchisches Caching

- Hierarchisches Caching: Anfrage wird bei einem Cache-Miss über mehrere Hierarchiestufen weitergereicht



- Gefundenes Objekt wird entlang der Hierarchie jeweils kopiert

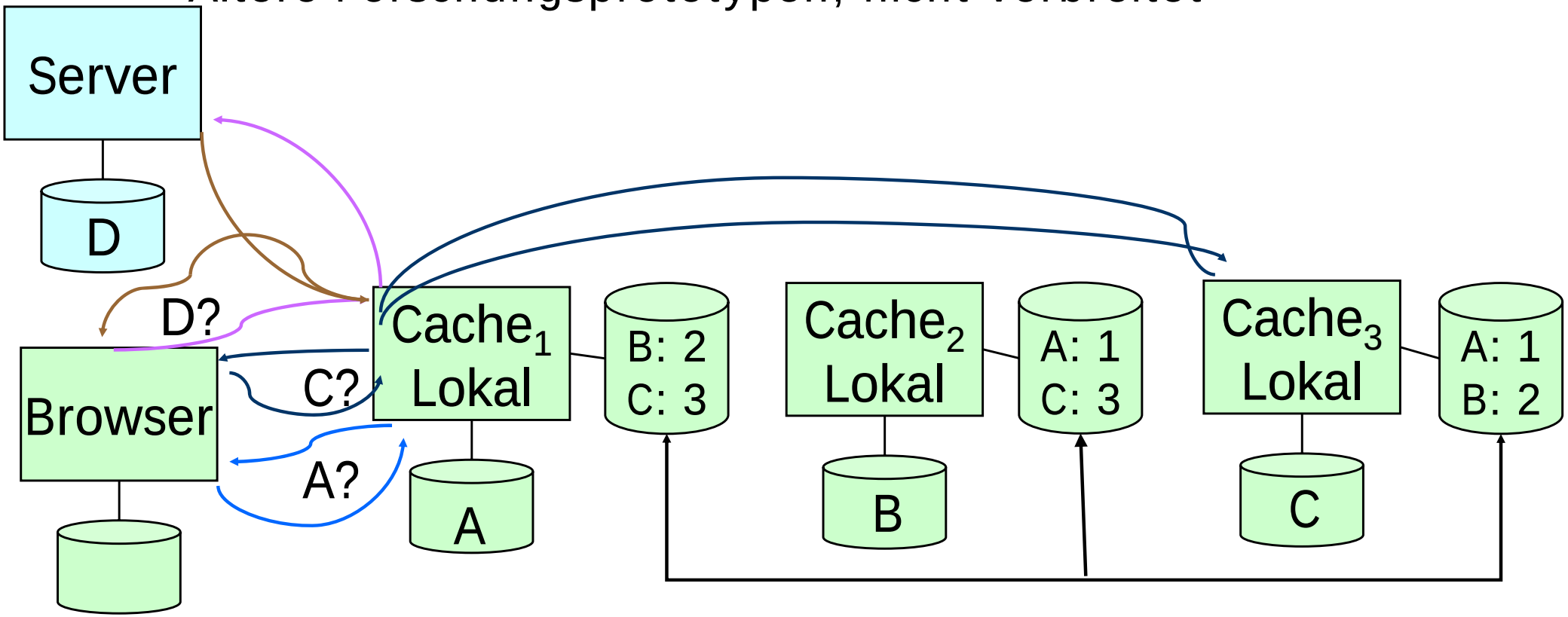
Cache Architekturen – Hierarchisches Caching

Freie Universität Berlin

- Vorteile
  - In der Hierarchie niedrig stehende Caches profitieren von besserer Bandbreite der hoch stehende Caches
  - Daten werden in Richtung der Nachfrage repliziert
  - Senkt Latenz für kleine Dokumente relativ mehr
- Nachteile
  - Platzierung der Caches an zentralen Netzknoten ist kritisch
  - Je Cache-Level erneut zusätzliche Latenz möglich
  - Je höher ein Cache in der Hierarchie steht umso eher wird er zum Engpass
  - Daten werden an sehr vielen Stellen entlang der Cache-Hierarchie in Kopien gehalten

Cache Architekturen – Verteiltes Caching

- Caches kooperieren
  - wissen, welcher Cache welche Objekt hält (Tabelle, Hashing,...)
  - fragen nach (➔ Harvest ICP)
  - Ältere Forschungsprototypen, nicht verbreitet



Cache Füllung

- Caches kommen auf Hit-Raten von 40-50%
  ➔ Wahrscheinliche Zugriffe vorwegnehmen
- Cache wird gefüllt durch
  - Kopieren nachgefragter Objekte
  - Aktives holen benachbarter Objekte
    - „Nachbarschaft“ ergibt sich aus Einbettung von Objekten auf Web-Seiten und Links
    - Aktives Füllen eines Web-Caches: *Prefetching*

- Weitere Beobachtung:
  Die meisten Objekte im Cache werden nur einmalig genutzt

Prefetching

- Auswirkung
  - zwischen Browser und Server
    - 45% weniger Latenz beim Browser
    - 200% Netzlast
  - zwischen Proxy und Server
    - 60% weniger Latenz beim Browser
  - zwischen Klient und Proxy
    - 23% Weniger Latenz
    - Großer Browser-Cache notwendig
- Push-Ansätze
  - Server/Proxy verteilt Dokumente zu Klienten
  - nicht im HTTP Modell
- Prefetching nicht verbreitet / nicht als sinnvoll angesehen

Cache Ersetzung (Placement/Replacement)

- Wenn Cache voll ist, muss Platz durch Löschung von Objekten geschaffen werden
- Drei Ansätze
  - Traditionelle Ansätze, basiert auf Objektnutzung
    - Least Recent Used LRU:
      Objekt mit ältester Nachfrage wird gelöscht
      - Populärste Strategie
      - Großes neues Objekt verdrängt viele kleine alte
    - Least Frequently Used LFU:
      Objekt mit wenigsten Nachfragen wird gelöscht
      - Größe von Objekten nicht berücksichtigt
      - Da die meisten Objekte gleich oft genutzt werden (1) ist Auswahl eigentlich zufällig
    - Pitkow/Recker:
      Wie LRU, aber: falls alle Objekte am selben Tag nachgefragt wurden, wird größtes gelöscht

Cache Ersetzung (Placement/Replacement)

- Basiert auf Ordnung von Objekteigenschaften
  - Size:
    Jeweils Größtes Objekt wird gelöscht bis Platz ist
  - LRU-MIN:
    Wendet LRU auf Objekte größer als s an, falls nur kleinere Objekte, LRU auf Objekte größer als s/2 anwenden usw.
  - LRU-Threshold:
    Wie LRU, aber Objekte über einer Größe s werden nicht zwischengespeichert
  - Hyper-G:
    Wie LFU, aber gleicher Rang über LRU und Size aufgelöst
  - Lowest-Latency-First (Latency Access Time, LAT):
    Objekt mit geringster Ladedauer wird gelöscht
  - Hybrid:
    Gewichtetes Maß aus Zugriffzeit, Zugriffsfrequenz und Größe

Einige Vergleichsmessungen
[Menaud, Issarny, Banatre 2000]

- Simulation mit Web-Traces der Univ. Berkeley
  - 1.11.96-18.11.96, 10000 Personen, 2500000 Zugriffe
- Algorithmus „Perfect“: Cache mit unbegrenzter Größe
-

Algorithms	Hr	Whr	Time
Perfect	30.2	28.7	8.97
LRU	24.6	21.4	10.54
LFU	25.3	21.8	10.62
SIZE	26.0	18.4	9.43
LAT	24.7	20.4	9.51
LRU-MIN	25.9	20.2	10.20
Hybrid	25.4	22.0	9.31

  - SIZE erhöht die Anzahl der Objekte im Cache
  - LRU/LFU profitieren von zeitlicher Lokalität der Zugriffe
  - LAT profitiert von hohen Latenzen der gehaltenen Objekte
  - LRU-MIN und Hybrid haben erhöhten Aufwand

Cache Ersetzung (Placement/Replacement)

Freie Universität Berlin

- Basiert auf Nutzungskosten
  - Greedy Dual-Size:
    Kosten werden mit Objekt verbunden, Objekt mit geringstem Kosten/Größe Verhältnis wird gelöscht
  - Lowest Relative Value:
    Nutzen wird mit Objekt verbunden, Objekt mit geringstem Nutzen wird gelöscht
  - Least Normalized Cost Replacement:
    Funktion Zugriffshäufigkeit $\times$ Übertragungskosten $\times$ Größe
  - ...
- Performanz der Ersetzung ist stark von Nutzungscharakteristik abhängig
- Kein Verfahren ist für alle Nutzungscharakteristika überlegen

Cache Kohärenz

- Nutzer können veraltete Seiten vom einem Cache erhalten
- Ähnlich Cache-Kohärenz in Verteilten Systemen, aber
  - andere Zugriffsmuster
  - andere Dimensionen
  - Web-Objekte werden nur an einem Ort geändert
- HTTP Unterstützung
  - Header `Expires:` *Datum* liefert Ungültigkeitsdatum
  - GET mit `If-Modified-Since:` *Datum* Header liefert Seite nur bei Änderungen nach einem Datum
  - Header `Pragma: no-cache` verhindert Caching
  - Header `Last-Modified:` *Datum* liefert Änderungsdatum
  - Header `Date:` *Datum* enthält Datum des letzten Tests auf Aktualität
  - Header `ETag:` *Signatur* liefert eine Quersumme des Objekts

Cache Kohärenz Mechanismen

- Starke Cache-Konsistenz: Immer aktuelle Objekte halten
  - Klient validiert
    - Annahme: Ressourcen im Cache sind veraltet
    - Vorgehen: Bei jeder Nutzung validieren
    - Implementierung: GET mit If-Modified-Since: Header
      - 200 – Keine Änderung
      - 304 – Not modified Antwort bei keiner Änderung (RFC: „should“)
  - Server invalidiert
    - Annahme: Ressourcen im Cache sind aktuell
    - Vorgehen: Server sendet Mitteilung bei Änderung
    - Implementierung: Listen über Cache-Klienten führen
      - Wie skalieren?
      - Wie Listen aktuell halten?

Cache Kohärenz Mechanismen

- Schwache Cache-Konsistenz: Irgendwann aktuelle Objekte
  - Klient invalidiert: Adaptive Time-To-Live (TTL)
    - Ausgangspunkt: Lebensdauer von Objekten ist bimodal
      - Entweder sehr kurze Lebensdauer
      - oder sehr lange Lebensdauer
    - Vorgehen: TTL eines Objekts = Anteil seines Alters (Aktuelle Zeit – Last-Modified)
    - Implementierung: Harvest: Anteil = 50% (CERN httpd 10%)
    - Vorteil: Hält Anteil alter Dokumente unter 5%
    - Nachteile
      - Nur Heuristik: Nutzer muss eventuell unnötig warten
      - Nur Heuristik: Keine Aussage über tatsächliche Gültigkeit
      - Nutzer können Heuristik nicht beeinflussen
      - Was passiert bei abgebrochenen Ladevorgängen?

Cache Kohärenz Mechanismen

- Piggyback Invalidation (Piggyback = „Huckepack“)
  - Ausgangspunkt: Kommunikation mit Server nutzen um Gültigkeit zu erfragen
  - Vorgehen:
    - Piggyback Cache Validation (PCV):
      Mit einer Anfrage schickt Proxy ein Liste zu validierender Objekte
    - Piggyback Server Invalidation (PSI):
      Mit einer Antwort schickt Server eine Liste geänderter Objekte
    - Hybrid: PCV+PSI
      - Wenn letzter Kontakt lange her: PCV (Overhead bei langer PSI-Liste größer)
      - Wenn letzter Kontakt kurz her: PSI (Liste kurz)

Cache-Fähigkeit von Objekten

- Statischer Inhalt von Seiten sehr gut zwischenspeicherbar
  - „sehr statische“ Inhalte (Logos) mit sehr spätem Expires Header
- Dynamische Inhalte schlecht zwischenspeicherbar
  - „sehr dynamisch“ Inhalte (Börsendaten) nicht cachen
  - „wenig dynamische“ Inhalte (Nutzeranschrift) kurz cachen
  - Serverseitiger Cache als Alternative

Zusammenfassung

- Caches zur Latenzverkürzung beim Nutzer
- Mehrstufiges Caching im Web
- Cache Füllung durch Prefetching
- Cache Ersetzung nach verschiedenen Methoden
- Cache Kohärenz mit verschiedenen Methoden

Literatur

- Jia Wang. A Survey of Web Caching Schemes for the Internet. ACM Computer Communication Review, 25(9), pp. 36-46, October 1999. http://www.cs.cornell.edu/Info/People/jiawang/web-survey.ps
- Zona Research, Inc. The Economic Impacts of Unacceptable Web-Site Download Speeds. April1999 http://www.keynote.com/downloads/whitepapers/economic_impact_of_downloadspeed.pdf
- Brian D. Davison. A Web Caching Primer. IEEE Internet Computing, 5(4), pp. 38-45, July/August 2001. http://www.cs.rutgers.edu/~davison/pubs/2001/internetcomputing/pubprimer.ps.gz
- Michael Baentsch, Lothar Baum, Georg Molter, Steffen Rothkugel and Peter Sturm. Enhancing the Web's Infrastructure: From Caching to Replication. IEEE Internet Computing, 1(2), pp. 18-27, 1997. http://citeseer.nj.nec.com/baentsch97enhancing.html
- K. Chinen and S. Yamaguchi. An interactive prefetching proxy server for improvement of WWW latency. Proceedings of INET'97, June 1997. Siehe http://shika.aist-nara.ac.jp/products/wcol/wcol.html
- R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee. Hypertext Transfer Protocol - HTTP/1.1. RFC 2616. June 1999. http://www.ietf.org/rfc/rfc2616.txt
- I. Cooper, J. Dilley Akamai. Known HTTP Proxy/Caching. RFC 3143. June 2001. http://www.ietf.org/rfc/rfc3143.txt
- Christoph Lindemann, Oliver P. Waldhorst. Analysis of Web Caching in the Gigabit Research Network G-WiN. Abschlußbericht zum Projekt Analyse der Wirksamkeit von Web Caching im G-WiN. University of Dortmund. April 2001. http://webdoc.sub.gwdg.de/ebook/ah/dfn/Cache-Analysis.pdf
- Jean-Marc Menaud, Valérie Issarny, Michel Banatre. Improving Effectiveness of Web Caching. In Recent Advances in Distributed Systems. S. Krakowiak and S. Shrivastava editors. Springer Verlag, LNCS 1752. 2000. http://www-rocq.inria.fr/arles/doc/ps00/Caching.pdf