

## Vorlesung Netzbasierte Informationssysteme (WS 2004/05) Suchmaschinen

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

[1] © Robert Tolksdorf, Berlin

## Überblick

[2] © Robert Tolksdorf, Berlin

## Überblick

- Server- und Klientenseitige Verarbeitung
- Caching im Web

[3] © Robert Tolksdorf, Berlin

## Server- und Klientenseitige Verarbeitung Überblick

[4] © Robert Tolksdorf, Berlin

## Überblick

- Statische und Dynamische Inhalte
- Serverseitige dynamische Inhalte
- Klientenseitige dynamische Inhalte

## Dynamische Web Seiten

- Seiten können
  - *Statische Inhalte* liegen vorgefertigt auf dem Server und werden unverändert ausgeliefert und dargestellt
  - *Dynamische Inhalte* bewirken vor und bei der Auslieferung oder der Darstellung Effekte auf Inhalt oder Darstellung durch Ausführung von Programmen
- Aufgaben
  - Serverseitig
    - Vereinfachung der Inhaltserstellung (SSI)
    - Erzeugung/Konvertierung von Inhalt
    - Anpassung/Personalisierung von Inhalt
    - Ergebnis anderer serverseitiger Transaktionen
  - Clientseitig
    - Erzeugung einer Darstellung
    - Anpassung von Darstellung
    - GUI Interaktion mit Nutzer

## Serverseitige Verarbeitung

- Templatesprachen
  - SSI, PHP,...
  - Ähnl. SSI Anweisungen in HTML eingestreut, mächtigere Sprache
- CGI/Skriptsprachen
  - /bin/sh, Perl, Python,...
  - Imperativer Programmcode, oft mit Stärken in Stringverarbeitung
- CGI/Reguläre Programme
  - Binaries in beliebiger Programmiersprache
  - Geschwindigkeit
- Sprache/API in Server eingebunden
  - Servererweiterungen (NSAPI, ISAPI, Apache), Servlets/JSP, ASP,
  - Kein externer Interpreter sondern Teil des Servers

## Server Side Includes

- Server Side Includes (SSI) sind Anweisungen an den Web Server, die im HTML Code eingebunden sind
- Werden vom Server bei Auslieferung ausgeführt und zu HTML Code expandiert
- SSI Anweisung im statischen HTML-Code:

```
<div align="right">
  Letzte &Auml;nderung: <!--#flastmod file=""-->.
</div>
```

- Ergebnis nach Auslieferung

```
<div align="right">
  Letzte &Auml;nderung: wednesday, 08-Jan-2003
  09:30:15 CET.
</div>
```

## SSI Anweisungen

- Als HTML-Kommentar eingebettet:  
`<!--#direktive Attribut1="wert1,, Attribut2="wert2"!-->`
- Direktiven:
  - `flastmod`: Fügt Änderungsdatum der im Attribut `file` genannten Datei ein
  - `fsize`: Fügt Größe der im Attribut `file` genannten Datei ein
  - `include`: Fügt Datei ein. Bezeichnung je nach Attribut:
    - `virtual`: Pfad im Web-Verzeichnisbaum  
`<!--#include virtual="/inst/ag-nbi/include/nbiheader" -->`
    - `file`: Pfad im (nur „unter“ aktueller Datei)
  - `exec`: Führt Kommando oder
    - `cgi`: Pfad im Web-Verzeichnisbaum (Apache 2.0: `include virtual`)
    - `cmd`: Ausführung im Datei-Verzeichnisbaum  

```
<pre>  
<!--#exec cmd="/usr/bin/ls -l"-->  
</pre>
```

[9] © Robert Tolksdorf, Berlin

## SSI

- `echo`: Gibt Inhalt der im Attribut `var` bezeichneten Variablen aus
- SSI-Variablen
  - `DOCUMENT_NAME`: Dateiname des Dokuments im Datei-Verzeichnisbaum
  - `DOCUMENT_URI`: Dateiname des Dokuments im Web-Verzeichnisbaum
  - `QUERY_STRING_UNESCAPED`: Suchanfrage
  - `DATE_LOCAL`: Aktuelles lokales Datum
  - `DATE_GMT`: Aktuelles lokales Datum als GMT
  - `LAST_MODIFIED`: Veränderungsdatum
- Beispiel:  
Hier ist es `<!--#echo var="DATE_LOCAL"-->`.

[10] © Robert Tolksdorf, Berlin

## SSI

- CGI-Variablen

<code>SERVER_SOFTWARE</code>	<code>SERVER_NAME</code>	
<code>GATEWAY_INTERFACE</code>		
<code>SERVER_PROTOCOL</code>	<code>SERVER_PORT</code>	<code>REQUEST_METHOD</code>
<code>PATH_INFO</code>	<code>PATH_TRANSLATED</code>	<code>SCRIPT_NAME</code>
<code>QUERY_STRING</code>	<code>REMOTE_HOST</code>	<code>REMOTE_ADDR</code>
<code>AUTH_TYPE</code>	<code>REMOTE_USER</code>	<code>REMOTE_IDENT</code>
<code>CONTENT_TYPE</code>	<code>CONTENT_LENGTH</code>	
- HTTP-Header der Form `HTTP_Headername`:
  - `HTTP_ACCEPT`, `HTTP_USER_AGENT`, ...
- Beispiel:  
wie geht es Ihnen bei  
`<!--#echo var="REMOTE_ADDR"-->` mit Ihrem  
`<!--#echo var="HTTP_USER_AGENT"-->`?

[11] © Robert Tolksdorf, Berlin

## SSI

- `set`: Setzt den Inhalt der im Attribut `var` bezeichneten Variablen auf den im Attribut `value` angegebenen Wert
- Einfachste Ausdrücke bildbar  

```
<!--#set var="salutation" value="Guten  
Tag"-->  
<!--#set var="remote"  
value="${REMOTE_ADDR} (${REMOTE_HOST})"--  
>  
  
<!--#echo var="salutation"--> at  
<!--#echo var="remote"-->!
```

[12] © Robert Tolksdorf, Berlin

## SSI

- Bedingte Abschnitte

```
<!--#if expr="Bedingung" -->
<!--#elif expr="Bedingung" -->
<!--#else -->
<!--#endif -->
```
- Bedingungen und Ausdrücke
  - String
  - $String_1 \neq String_2$
  - $String_1 < String_2$
  - $String_1 \leq String_2$
  - $String_1 > String_2$
  - $String_1 \geq String_2$
  - ( Bedingung )
  - ! Bedingung
  - $Bedingung_1 \ \&\& \ Bedingung_2$
  - $Bedingung_1 \ || \ Bedingung_2$

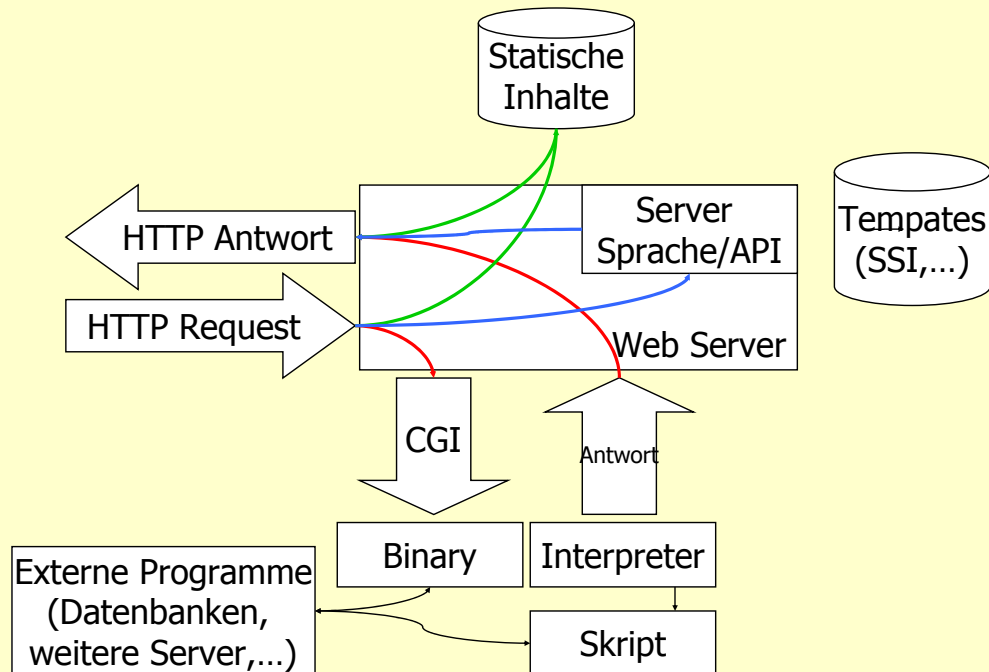
[13] © Robert Tolksdorf, Berlin

## CGI

- CGI ist eine Schnittstelle zwischen Web-Server und Programm
  - URL bezeichnet CGI-Programm
  - Server startet Programm als Prozess
  - Server übergibt Parameter der Anfrage und anderes in Umgebungsvariablen
  - Programm verarbeitet Anfrage
  - Kann Seiteneffekte haben, teilweise sehr große
  - Programm erzeugt eine HTTP-Antwort (mit HTML-Seite oder anderem)
  - Server leitet Antwort an Klienten weiter
- CGI-Programm ist etwas Ausführbares
  - Interpreter (der Skript interpretiert)
  - Compiliertes Programm

[14] © Robert Tolksdorf, Berlin

## Im Überblick



[15] © Robert Tolksdorf, Berlin

## Klientenseitige Verarbeitung

- Vom Server gelieferte Seiten enthalten Dinge, die beim Klienten Ausführung bewirken
  - In Seiten referenzierte Objekte
    - andere Medienarten als HTML
    - Applets
  - Programme
    - Eingebettete Scripte

[16] © Robert Tolksdorf, Berlin

## Referenzierte Objekte

- Browser verstehen bestimmte Medienarten
  - text/html
  - image/jpg
  - ...
- Bei unbekanntem Medienarten wie `application/x-shockwave-flash` oder `application/pdf` kann
  - gespeichert werden
  - durch Plugin ausgeführt werden
- *Plugin*
  - Browsererweiterung
  - Klientenseitig kompiliert und vorinstalliert
  - Herstellerabhängig / Browserabhängig / Plattformabhängig
  - wird als Teil des Klientenprozesses ausgeführt
  - kann Verbindungen aufbauen

[17] © Robert Tolksdorf, Berlin

## Referenzierte Objekte

- *Applets*
  - enthalten Programmcode und werden nicht dargestellt sondern ausgeführt
  - Ausführungsbasis ist nicht der Prozessor des Rechners sondern eine Java Virtual Machine, die direkt oder als Plugin im Browser installiert ist
  - Serverseitig kompiliert und vorgehalten
  - Herstellerunabhängig / Browserunabhängig / Plattformunabhängig (?)
  - werden als Teil des Klientenprozesses in der JVM ausgeführt
  - können (konfigurierbar) eingeschränkt Verbindungen aufbauen
  - laden Programmcode nach
- *ActiveX*
  - ähnlich, aber Browser- und Plattformabhängig

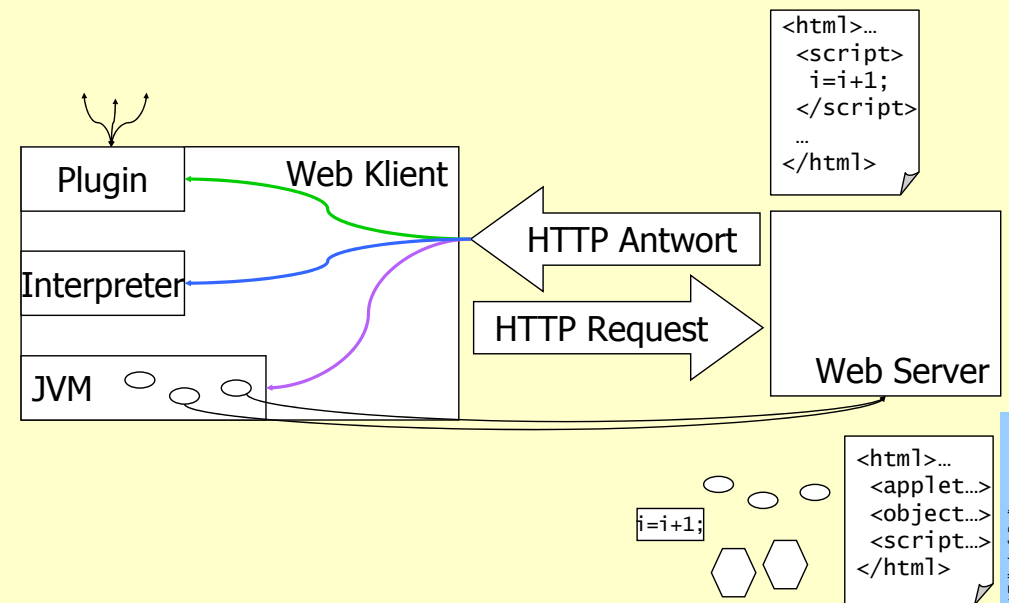
[18] © Robert Tolksdorf, Berlin

## Eingebettete (oder referenzierte) Skripte

- *Skripte*
  - Per `<script>` Tag eingebetteter oder referenzierter Programmcode
  - enthalten Programmcodequelltext und werden nicht dargestellt sondern ausgeführt bei Eintreffen von Ereignissen
  - Ausführungsbasis ist nicht der Prozessor des Rechners sondern ein Skriptinterpreter der direkt oder als Plugin im Browser installiert ist
  - Skriptsprachen sind einfache imperative Sprachen
  - Skripte werden serverseitig vorgehalten
  - Herstellerunabhängig / Browserabhängig / Plattformunabhängig (?)
  - Haben Zugriff auf Browserkontext (z.B. DOM des aktuellen Dokuments), unterschiedlich auf Rechner und Netz
  - JavaScript, ECMAScript, JScript, weitere...

[19] © Robert Tolksdorf, Berlin

## Im Überblick



[20] © Robert Tolksdorf, Berlin

## Zusammenfassung

- Dynamische Seiten
  - Serverseitig
    - Templates (SSI)
    - CGI
    - Servererweiterungen
  - Klientenseitig
    - Plugins / Medientypen
    - Applets etc.
    - Skripte

## Literatur

- NCSA HTTPd Development Team. *Server Side Includes (SSI)*.  
<http://hoohoo.ncsa.uiuc.edu/docs/tutorials/includes.html>
- Apache HTTP Server Documentation Project. *Apache Tutorial: Introduction to Server Side Includes*.  
<http://httpd.apache.org/docs/howto/ssi.html>  
<http://httpd.apache.org/docs-2.0/howto/ssi.html>
- Sun Microsystems, Inc. *JavaServer Pages*.  
<http://java.sun.com/products/jsp/>
- Netscape Communications Corporation. *Plug-in Guide*.  
<http://developer.netscape.com/docs/manuals/communicator/plugin/index.htm>

## Caching im Web

## Überblick

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

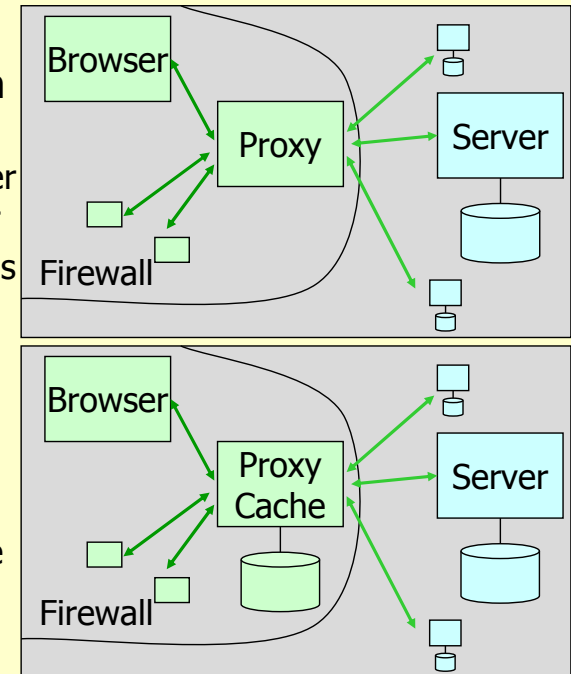
## Caching

- 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

[25] © Robert Tolksdorf, Berlin

## 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



[26] © Robert Tolksdorf, Berlin

## 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
    - "eight second rule": Web Seite muss innerhalb von 8 Sekunden angezeigt werden oder Nutzer verlieren Interesse an Site (bzw. Kauf...)
  - Serverlast sinkt wegen geringerer Zugriffszahl
  - Verfügbarkeit von Objekten steigt
  - Proxies lassen Nutzungsanalysen zu

[27] © Robert Tolksdorf, Berlin

## 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

[28] © Robert Tolksdorf, Berlin

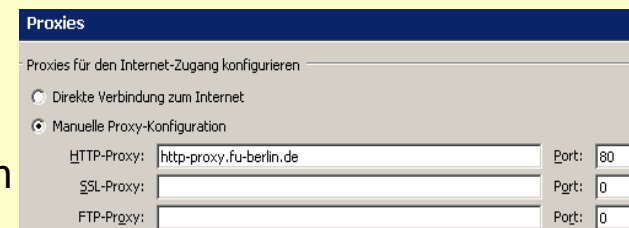
## Notwendigkeit

- 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

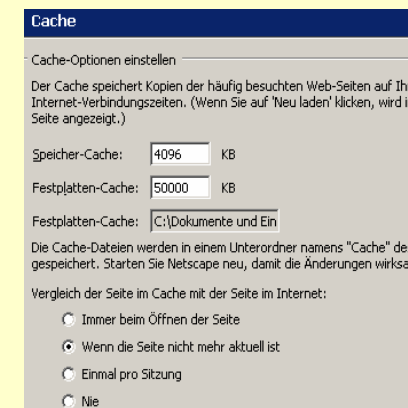
[29] © Robert Tolksdorf, Berlin

## Browser Konfiguration

- Browser kann zu externem Proxy-Cache gerichtet werden

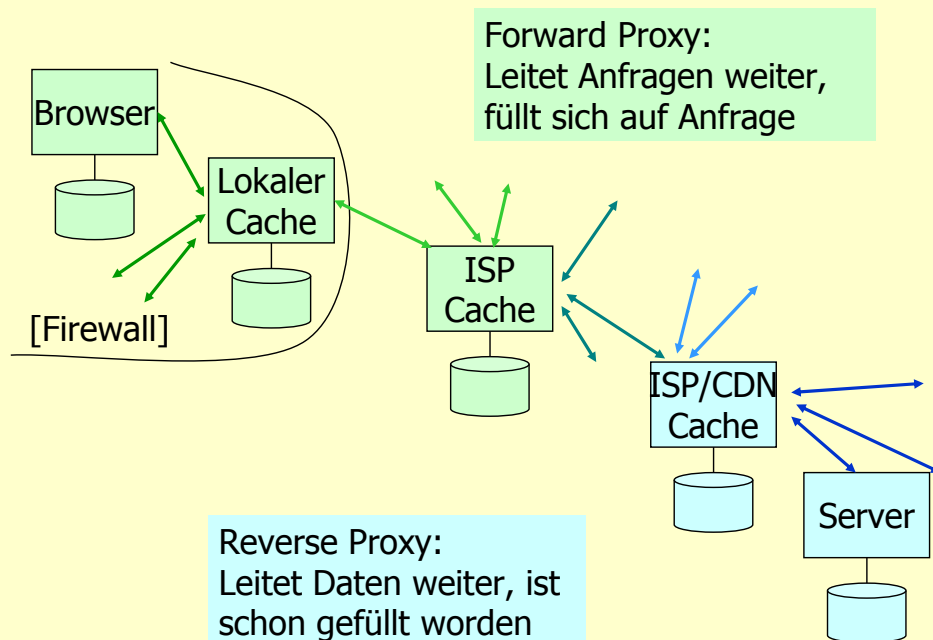


- Browser hat aber auch selber schon einen lokalen Cache (Speicher / Disk)



[30] © Robert Tolksdorf, Berlin

## Caches im Web allgemein



[31] © Robert Tolksdorf, Berlin

## 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

[32] © Robert Tolksdorf, Berlin

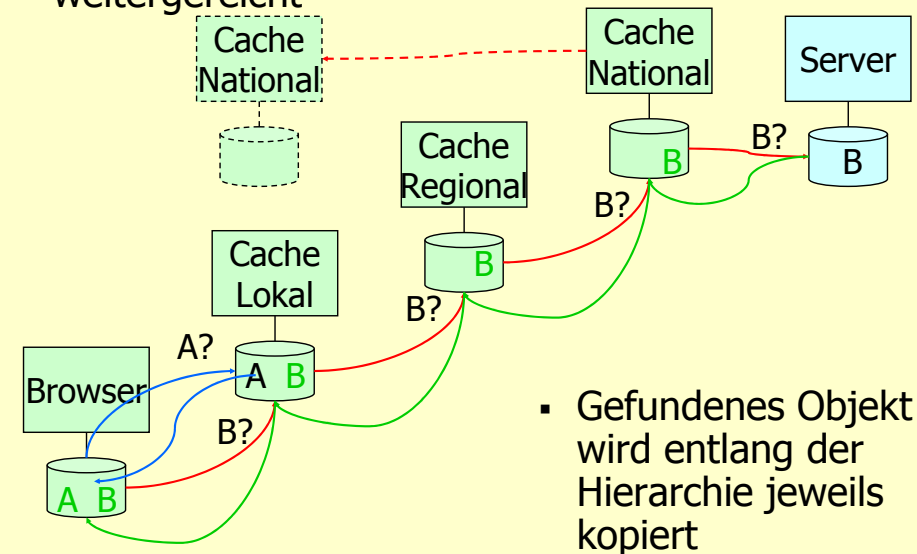
## Design-Ziele

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

[33] © Robert Tolksdorf, Berlin

## Cache Architekturen - Hierarchisches Caching

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



[34] © Robert Tolksdorf, Berlin

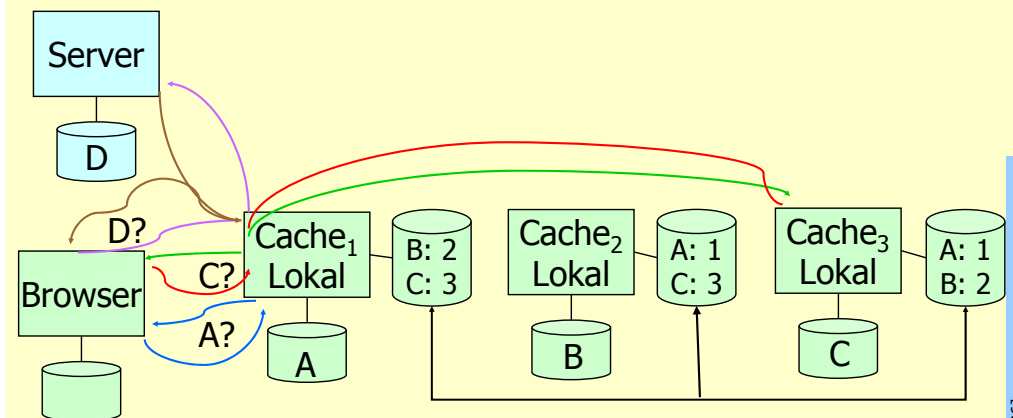
## Cache Architekturen - Hierarchisches Caching

- **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

[35] © Robert Tolksdorf, Berlin

## Cache Architekturen - Verteiltes Caching

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



[36] © Robert Tolksdorf, Berlin

## 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*

## 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
    - Least Frequently Used LFU:  
Objekt mit wenigsten Nachfragen wird gelöscht
    - 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:  
Größtes Objekt wird gelöscht
  - 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:  
Objekt mit geringster Ladedauer wird gelöscht

## Cache Ersetzung (Placement/Replacement)

- Basiert auf Nutzungskosten
  - GreedyDual-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  $\text{Zugriffshäufigkeit} \times \text{Übertragungskosten} \times \text{Größe}$
  - ...
- Performanz der Ersetzung ist stark von Nutzungscharakteristik abhängig
- Kein Verfahren ist für alle Nutzungscharakteristika am überlegen

[41] © Robert Tolksdorf, Berlin

## 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

[42] © Robert Tolksdorf, Berlin

## Cache Kohärenz Mechanismen

- Starke Cache-Konsistenz: Immer aktuelle Objekte halten
  - Klient validiert
    - Annahme: Ressource 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?

[43] © Robert Tolksdorf, Berlin

## 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?

[44] © Robert Tolksdorf, Berlin

## Cache Kohärenz Mechanismen

- Piggyback Invalidation (Piggyback = „Huckpack“)
  - 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)

[45] © Robert Tolksdorf, Berlin

## 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
  - Serverseitiges Cachen als Alternative

[46] © Robert Tolksdorf, Berlin

## Zusammenfassung

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

[47] © Robert Tolksdorf, Berlin

## 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*. April 1999. [http://www.keynote.com/downloads/whitepapers/economic\\_impact\\_of\\_downloadspeed.pdf](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>

[48] © Robert Tolksdorf, Berlin

## Zusammenfassung

## Zusammenfassung

- Server- und Klientenseitige Verarbeitung
- Caching im Web