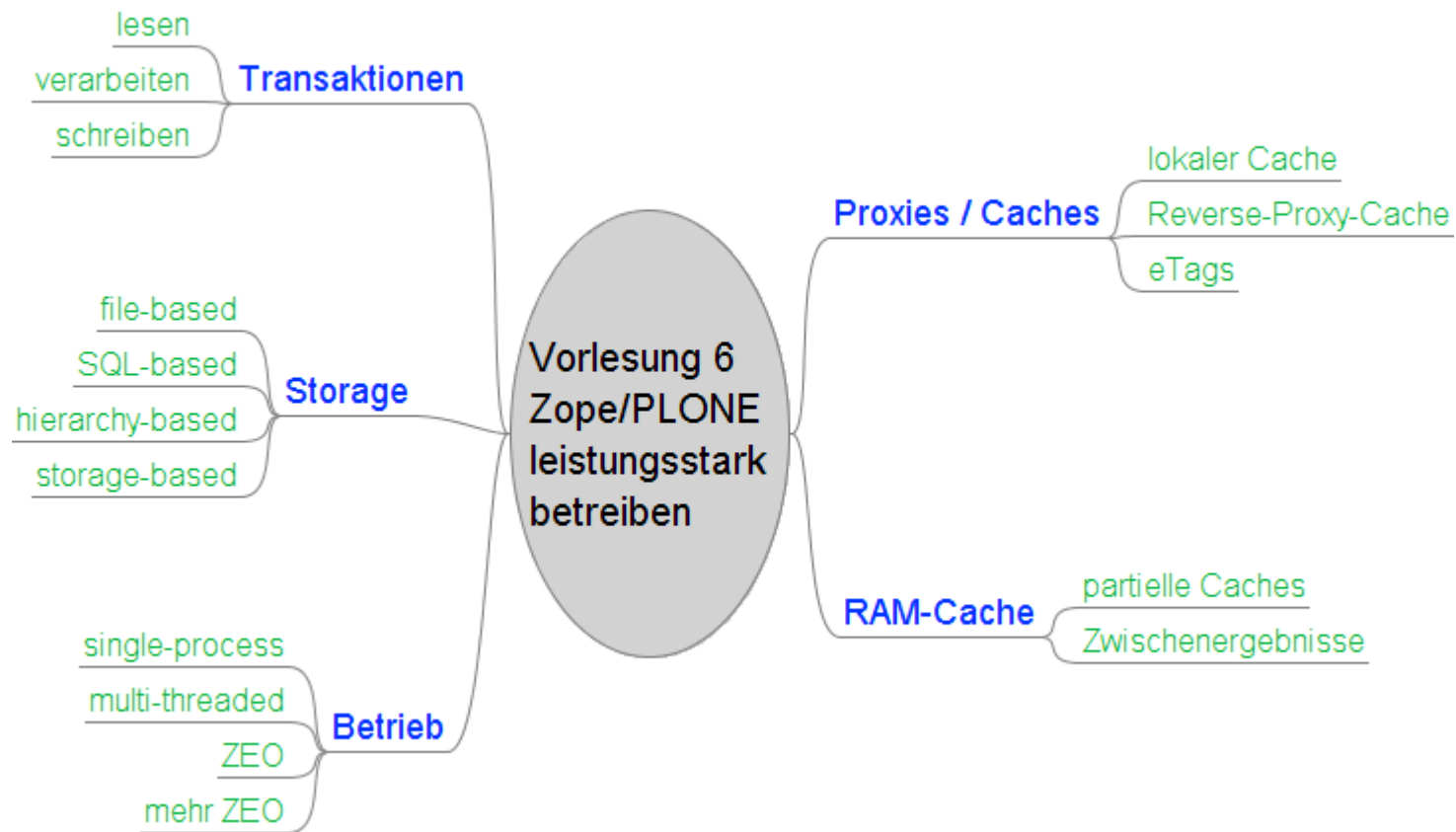


# Content-Management-Systeme

Dipl.-Inform. Roman Jansen-Winkeln

Vorlesung 6: PLONE leistungsstark betreiben

# Inhalt und Organisation



# Transaktionen

Transaktionen kapseln jede Anfrage

# Transaktionen - Prinzip

- Transaktion =  
eine feste Folge von Operationen, welche als eine logische Einheit betrachtet werden
- ACID-Prinzip
  - Atomarität (Atomicity)
  - Konsistenz (Consistency)
  - Isolation (Isolation)
  - Dauerhaftigkeit (Durability)

# Transaktionen - Zope

- Jede Anfrage ist in einer Transaktion gekapselt
  - HTTP-GET eröffnet Transaktion mit BEGIN
  - Persistente Objekte lesen aus Storage in RAM
  - Verarbeitung durch Methoden, Skript, Templates
  - Speichern veränderter/neuer Objekt im Storage
  - ROLLBACK bei Transaktionsfehlern
  - HTTP-PUT beendet die Transaktion mit COMMIT
- Datenintegrität sichern durch OPTIMISTIC LOCK

# Optimistic Lock

- Optimistic Lock  
Optimistic Concurrency Control (OCC)
- Verfahren
  - **Lesen**: Daten aus Storage lesen, private Kopie anlegen, Ausgangswerte zusätzlich speichern
  - <Verarbeiten der Daten in der Transaktion>
  - **Validieren**: prüfen, ob sich der Ausgangswerte eines geänderten Werts im Storage geändert hat --> Konflikt!!
  - Bei Konflikt: Konfliktauflösung versuchen.  
Falls Fehlschlag: ROLLBACK
  - **Write**: Änderungen in Storage schreiben, eigene Kopien löschen, COMMIT

# Optimistic Lock

- Beispiel:

- Class myTest(ZODBObject):  
    zahl = 2

- Def miniTrans(self):  
    wert = self.zahl  
    self.zahl = wert \* 2  
    return

- Gtest = myTest()  
    ....  
    Gtest.miniTrans()

- Bei Aufruf von miniTrans() sei self.zahl == 2  
Deshalb gilt wert == 2  
Andere Transaction schreibt self.zahl = 3  
Bei „return“ will miniTrans() schreiben self.zahl = 4  
KONFLIKT

- Def \_p\_resolveConflict(self, old, saved, new):  
    return new\*2

# Optimistic Lock

- Vorteile Optimistic Lock
  - Kompatibel mit ACID-Prinzip
  - Gut geeignet, wenn selten Konflikte auftreten
  - Sehr schnell, weil keine echten Datensperren erforderlich sind
- Nachteile Optimistic Lock
  - Bei vielen Konflikten ineffizient
  - Verhindert konflikthafte Transaktionen erst am Ende
  - Sagt nicht, wie auf Rollback reagiert werden soll
- Fazit
  - Web-Umgebung lesen viel, verändern wenig
  - Optimistic Lock für Zope geeignet

# Transaktionen

- Bei Rollback: Transaktion n=3-mal wiederholen
- Transaktionen synchronisieren RAM-Speicher mit persistentem Storage
- Zope unterstützt Subtransaktionen
  - Erlauben bessere Speichernutzung
  - Machen Speicherveränderungen früher für alle sichtbar
  - Verhindern ineffektives „weiterrechnen“ bei Konflikten
- Literatur
  - <http://www.zope.org/Documentation/ZODB2>

# Storage

# Storage

- Zope speichert Objekt in der ZODB = Zope-Object-Database
- ZODB stellt Objekte und Operationen ähnlich einem Dictionary zur Verfügung
  - `MyDB['key1'] = myTest()`
  - Neue Instanz von `myTest()` wird in ZODB unter dem Namen 'key1' gespeichert
- ZODB garantiert dauerhafte Speicherung
  - Dazu verwendet die ZODB sogenannten STORAGE
- Was ist als Storage für Python-Instanzen geeignet?

# Storage – File Based

- Datei-basierter Storage
- Alle Objekte stehen in einer Datei  
per Konvention: `var/Data.fs`
- Python-Instanz wird als String-kodiert (serialisiert):  
Methoden: `pickle` / `unpickle`
- Kodierung als XML-Datei oder Attribute-Wert-Paare.
- Objekt-Zugriff:
  - Neue leere Instanz erzeugen
  - Daten aus ZODB-suchen und String lesen
  - Instanz durch „unpickle“ füllen

# Storage – Relational Storage

- Relationaler Storage – Speicherung in SQL-DB
- Objekt-Relationales Mapping erforderlich
- Intelligentes Mapping:
  - 1 Klasse == 1 Tabelle
  - 1 Objekt == 1 Tabellen-Record
  - 1 Field == 1 Tabellenattribut
- Generisches Mapping
  - Tabelle für alle Objekte
  - Tabelle für alle Attribute
  - Verknüpfung über Foreign Keys
- Storages für MySQL, PostGRES, (Oracle, MS SQL-Server)

# Storage – Hierarchical Storage

- Directory Storage – Speicherung in Verzeichnis und Datei
- Hierarchische Objekt-Struktur direkt auf Datei-System abbilden
- Jedes Objekt ist eine Datei
- Jedes Folder-/Container-Objekt ist ein Verzeichnis
  
- Leicht lesbare Datenstruktur
- Einfache Implementierung
- 30% größer und langsamer als File-based

# Storage – Aggregated Storage, Storage as Storage

- Aggregated Storage: Komplexe Datenverwaltungssysteme  
Beispiel: LDAP
- Nutzt die komplexen Speichermöglichkeiten um Python-Instanzen persistent auszulagern.
  
- Storage as Storage:  
Verwendet eine andere ZODB als Storage
- Beispiele: Demo-Storage, ZEO-Storage

# Storage – Transaktionen

- ZODB-Storages unterstützen Transaktionen
  - ZODB implementiert Optimistic Lock
  - History mit Änderungen == Transaktionsprotokoll == Transaction Log
  - Frühere Objekt-Zustände werden als sog. *Revision* aufbewahrt
- Transaktionsprotokoll:
  - Storage implementiert optional Transaktionsprotokoll
  - Rollback über Transaktionsprotokoll möglich  
== Undo-Operation
  - Speicherung von Revisionen erfordert viel Speicherplatz
  - Storage benötigt „Packing“-Operation
  - Einfacher Copy-Pack-Algorithmus für File-Storage

# Zope-Storages

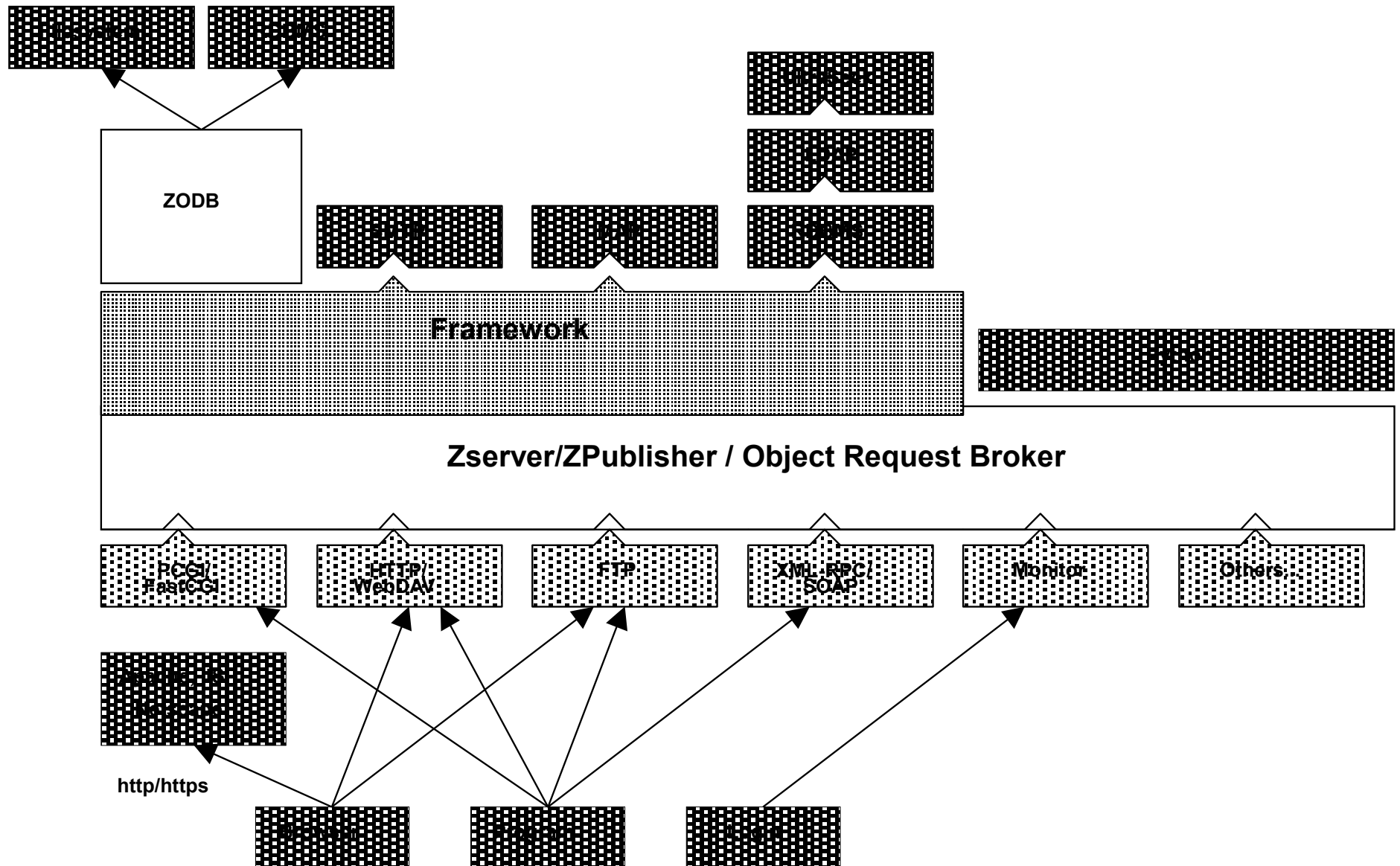
- Zope konfiguriert, welcher Storage verwendet wird
- Mehrere Mount-Points in einem Zope-Server
  - Unterstützt mehrere unabhängige Storages
  - Schwierig beim Kopieren und bei Transaktionen

# Zope-Betrieb

# Betrieb – Single Prozess

- Einfachster Fall: Zope als normales Python-Programm
  - Python -setup.py -.....
  - Gestartet als Shell-Skript bzw. BAT-File
  - Alternativ als Demon bzw. Service ausgeführt
- In einem Prozess laufen:
  - Integrierter Web-Server
  - Zope-Verarbeitung (Application-Server)
  - ZODB mit Zugriff auf File-based Storage Data.fs
- Schwachpunkt
  - Parallel Anfrage werden innerhalb des Prozesses geschedult
  - Internes Zope-Scheduling ist nicht gut

# Betrieb – Single Prozess



# Betrieb – Multiple Threads

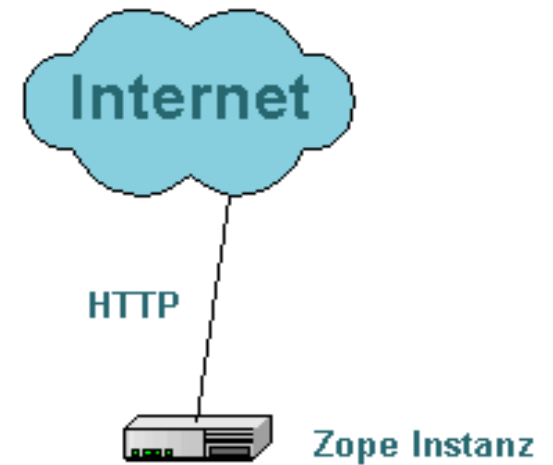
- Ein Prozess mit mehreren Threads
- In einem Prozess laufen:
  - Gemeinsamer, integrierter Web-Server
  - Mehrere Thread mit Zope-Verarbeitung (Application-Server)
  - ZODB mit Zugriff auf File-based Storage Data.fs
- Ablauf:
  - Alle Anfragen gehen an den gleichen Web-Server
  - ORB verteilt Anfragen auf feste Anzahl Verarbeitungs-Threads (4-10)
  - Bei mehr parallelen Anfragen: Zope-internes Scheduling
  - Alle Objekte werden aus der gleichen Datenbank gelesen

# Betrieb – Multiple Threads

- Verbesserungen:
  - Threads schedulen über das Betriebssystem besser
- Schwachpunkte:
  - Alle Threads laufen im gleichen Python-Executable
  - Python kann keinen Multi-Prozessor-Support
  - Deshalb nutzen Multi-Thread-Zopes auch nur einen Prozessor
- Gute Betriebsform bei mittlerer Last und starkem Prozessor

# Betrieb – Multiple Threads

## Standard Zope



# Betrieb – Zope Enterprise Objects – ZEO

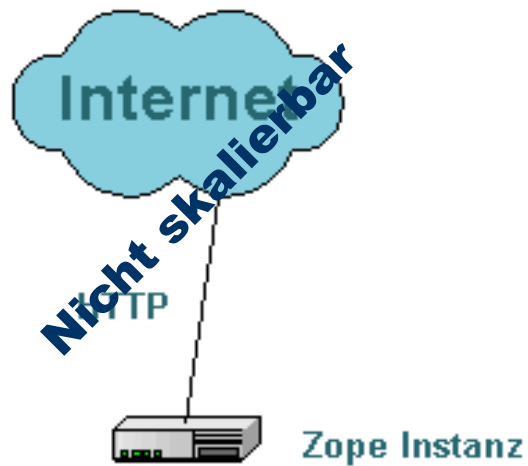
- Mehrere Prozesse mit je einem vollständigen Zope
  - Alle Zopes benutzen den gleichen Storage
  - Jedes Zope hat seinen eigenen Web-Server
  - Einzelne Zopes als Single-Process oder Multi-Threaded
- ZEO-Server und ZEO-Client-Storage
  - Spezieller ZEO-Client-Storage
  - Greift auf den Storage eines ausgewählte Zopes zu: ZEO-Server
  - ZEO-Server = Minimales Zope: ZODB+Storage + ZEO-Protokoll
  - ZEO-Client übernimmt Anfrage und Verarbeitung

# Betrieb – Zope Enterprise Objects – ZEO

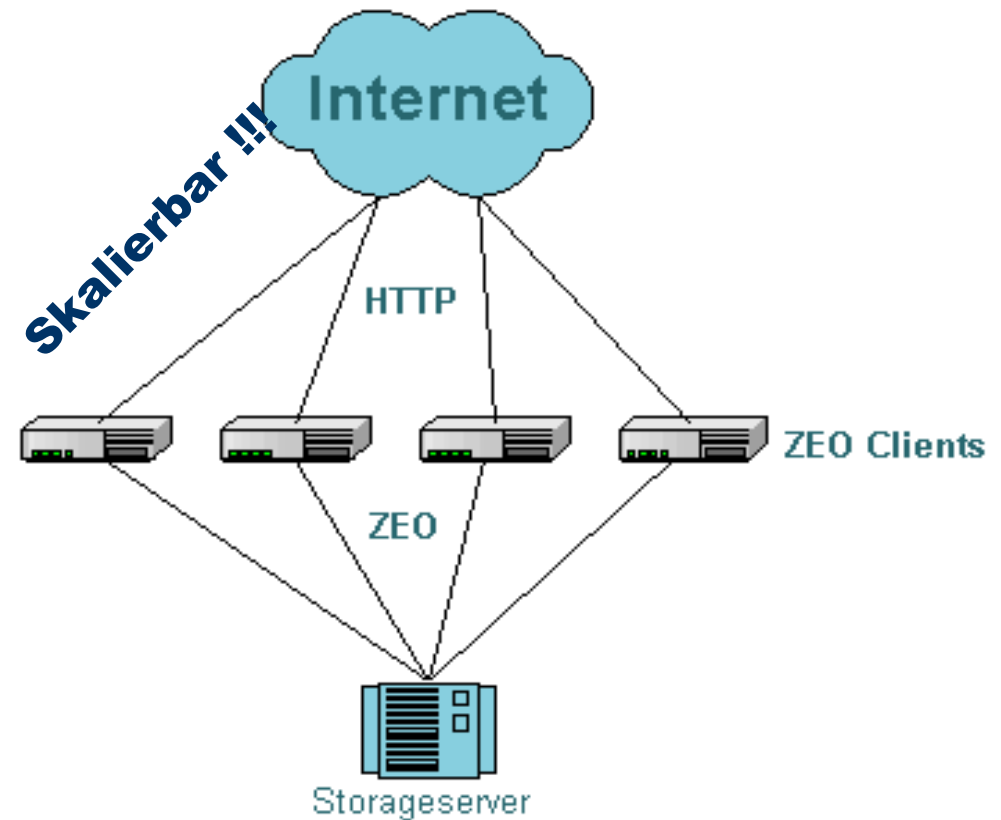
- Verbesserungen:
  - Mehrere ZEO-Clients auf einem Rechner nutzen Multi-Prozessoren
  - ZEO-Clients auf mehreren Rechnern möglich
  - Kaskaden mit ZEO-Server möglich
  - Dedizierte ZEO-Clients möglich: Read-Only, Long-Running-Jobs, ...
  - Synchronisation über gemeinsamen ZEO-Server-Storage
- Schwachpunkte:
  - Verteilung der Anfragen auf ZEO-Clients:  
Dedizierter Load-Balancer, Web-Server mit Load-Balancer,  
Manuelle Verteilung,
  - Ausfall des ZEO-Servers
- Robuste Betriebsform bei hoher Last und vielen Prozessoren

# Betrieb – Multiple Threads

## Standard Zope



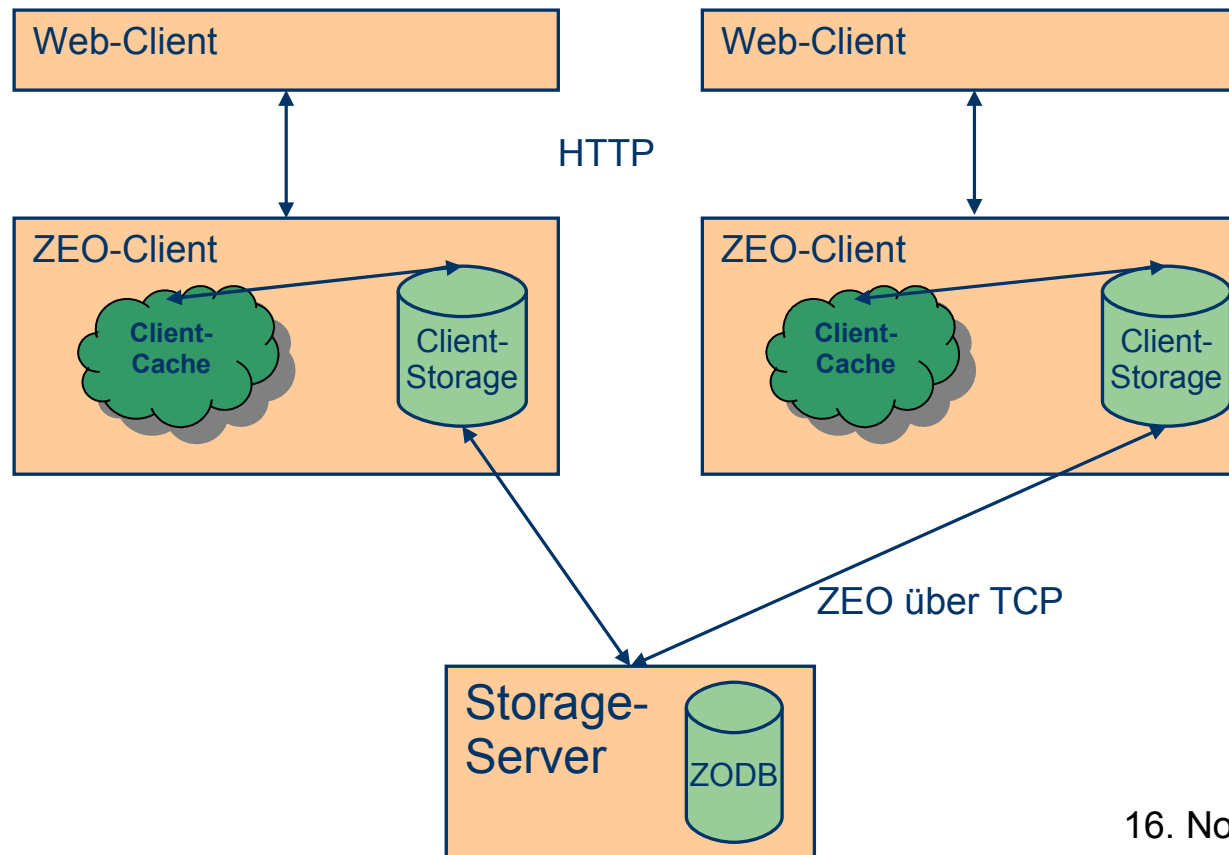
## ZEO (Zope Enterprise Objects)



# Betrieb – ZEO-Storage

- ZEO-Client-Storage: Storage as Storage
- Beinhaltet alle Funktionen eines echten Storage
- Eigenes IP-Protokoll zur Kommunikation mit ZEO-Server:  
Z3-Handshake-Protokoll
- ZEO-Server: abgespeckter Zope-Server
- ZEO-Client hat lokalen Cache um alle abgerufenen Objekte zu speichern

# ZEO: Architektur

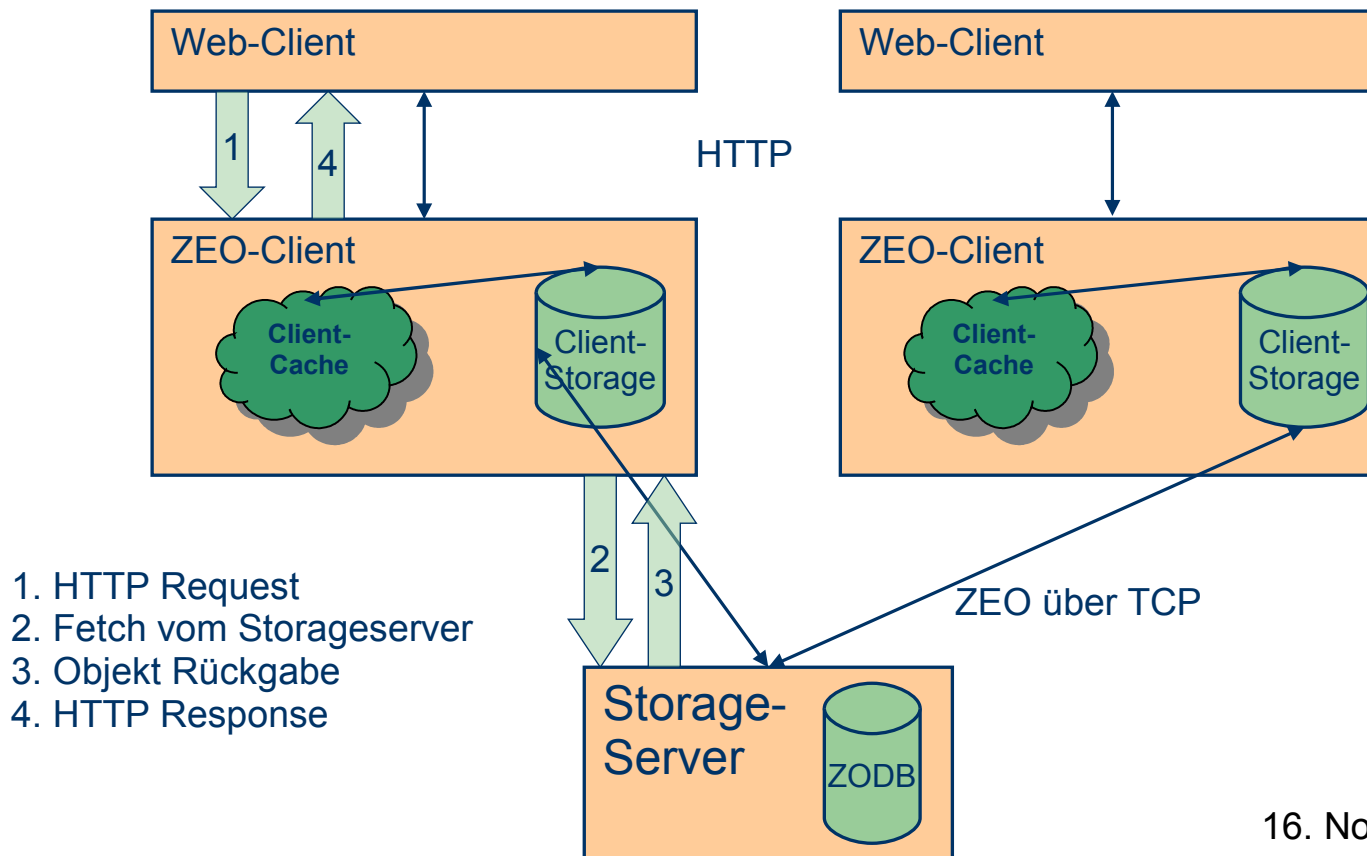


16. November 2009

# Betrieb – ZEO-Storage

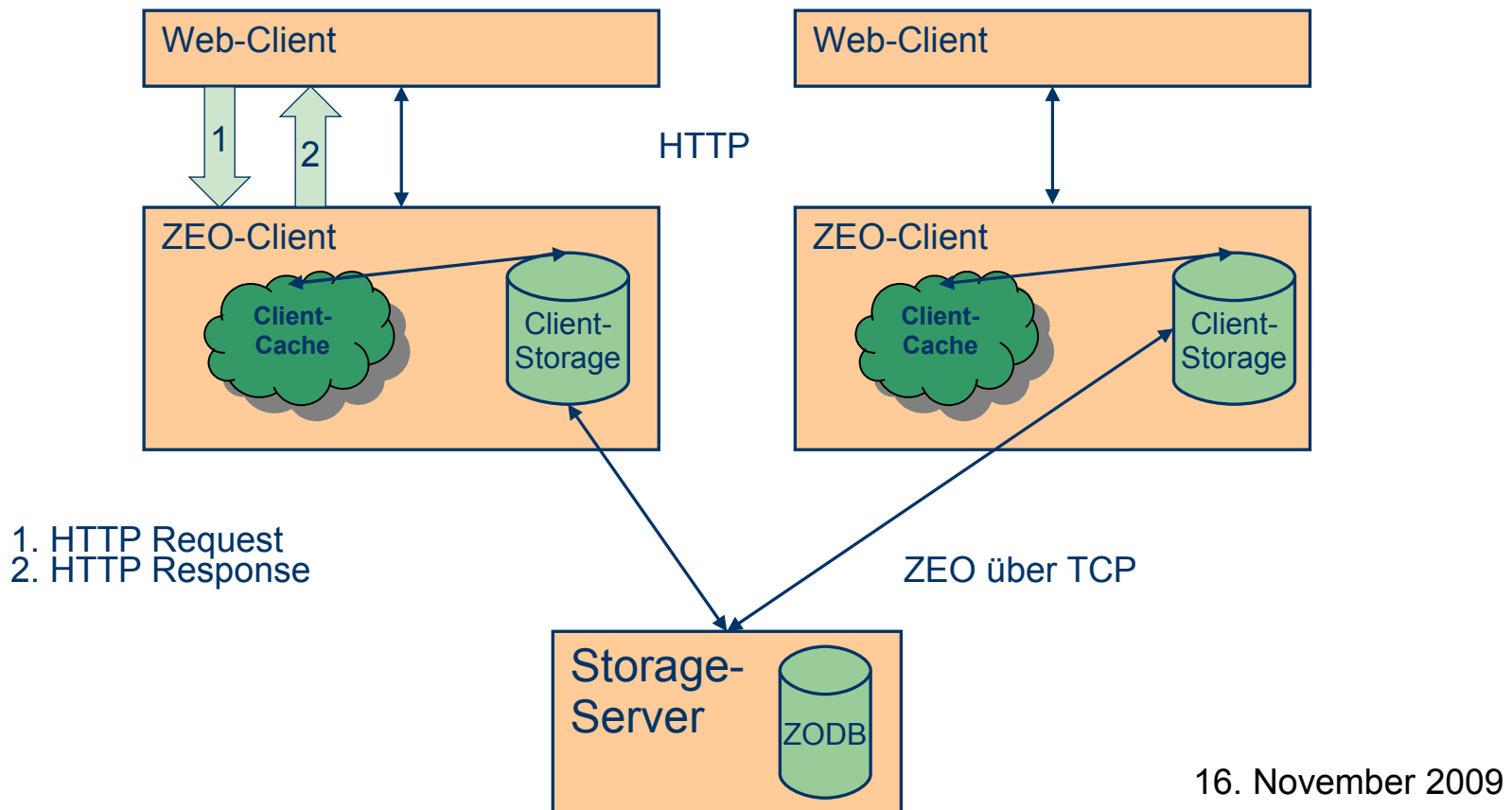
- Lesende Anfragen in ZEO
  1. Lesende Anfrage aus einer Transaktion an den ZEO-Storage
  2. Der ZEO-Storage sucht im lokalen Cache nach angefragten Objekten
  3. Falls nicht lokal gecacht fragt der ZEO-Client-Storage den ZEO-Server nach dem Objekt an
  4. Zurückgeliefertes Objekt wird im lokalen Cache gespeichert.
  5. Objekt wird an die Transaktion geliefert.
- ZEO-Storage wird dank Cache immer schneller
- Vollständige Kopie des Server-Storage denkbar

# ZEO: Laden (nicht gecacht)



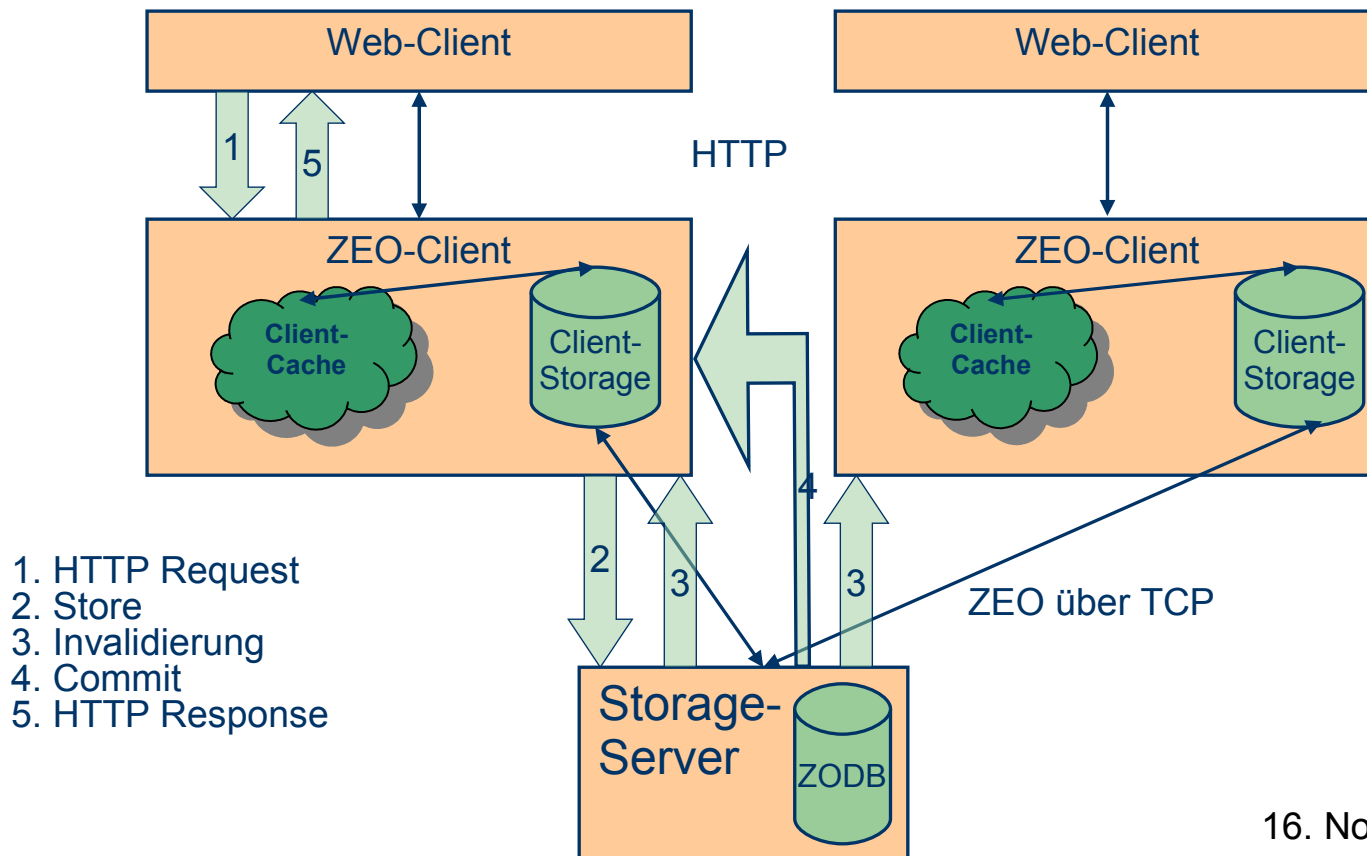
16. November 2009

# ZEO: Laden (aus Client-Cache)



- Schreibende Anfragen in ZEO
  1. Schreibauftrag für ein Objekt aus der Transaktion.
  2. ZEO-Storage überprüft lokal Optimistic-Lock und sendet den Schreibauftrag an den Server.
  3. Der ZEO-Server speichert das Objekt in seinem Storage
  4. Der ZEO-Server sendet **allen** ZEO-Clients ein „invalidate“-Signal für dieses Objekt.
  5. Alle ZEO-Clients markieren daraufhin dieses Objekt in ihrem lokalen Cache als nicht mehr aktuell bzw. löschen es.
  6. ZEO-Server bestätigt Commit an auslösenden ZEO-Client
  7. Storage bestätigt Commit an die ausführende Transaktion

# ZEO: Speichern



# Betrieb – ZEO-Storage

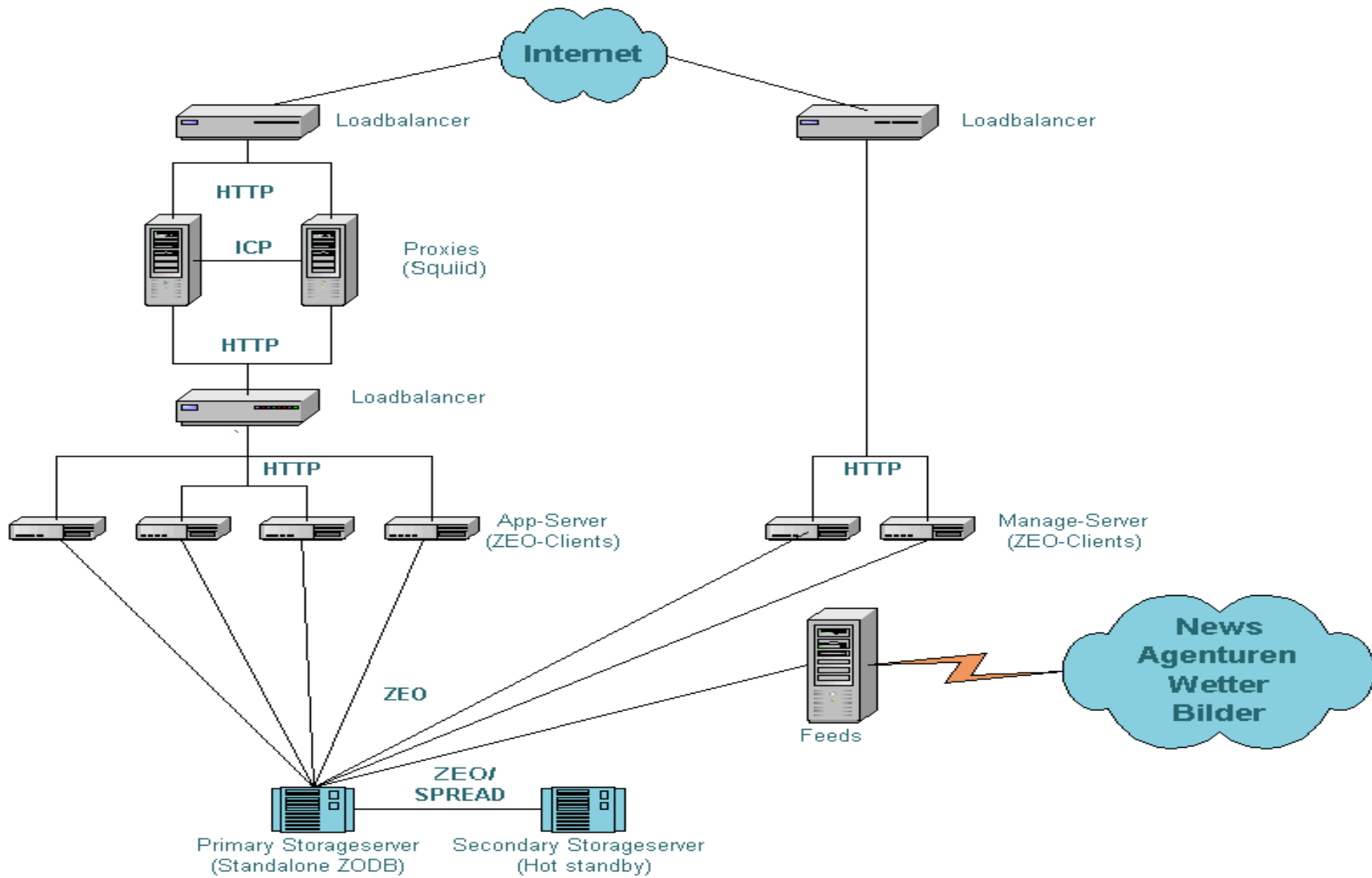
- Schreibende Anfragen in ZEO sind langsamer
  - In Web-Server-Umgebungen akzeptabel, da überwiegend Datenabruf
- Schreiben wird langsamer je mehr ZEO-Clients aktiv sind
  - Typischerweise 2-20 ZEO-Clients
- Achilles-Ferse ist der ZEO-Server
  - Redundant auslegen
  - Stand-by-Server mit kurzen Anlaufzeiten verwenden
  - Bei File-Based ZEO-Storage ggf. NAS für Data.fs verwenden
  - Variante: Relational-Storage mit ausfallsicherem RDBMS verwenden

# ZEO-Betrieb in Multi-Server-Szenarien

- Die Verteilung der ZEO Prozesse auf Hardware ist ein heuristischer Prozess.
- Bei der Ermittlung der optimalen Verteilung helfen Tests und Erfahrung.
- Sinnvoll ist die Nutzung von Partitionierungsmöglichkeiten
  - Anwendungs-Cluster zu bilden
  - einfach und ohne Zope-Kenntnisse installieren und vervielfältigen
  - Beispiel: Blade-Server, SUN-Zones oder Linux-VMs.

# ZEO-Betrieb in Multi-Server-Szenarien

- Typische Konstellation in Blade-Architekturen:
  - Master-Blade mit Apache, Load-Balancer und ZEO-Server
  - Redundantes Master-Blade als Stand-By
  - Arbeits-Blade, je nach Leistungsfähigkeit mit 2-4 ZEO-Clients
  - NAS als gemeinsames Speichersystem für alle Blades
- SUN-Umgebungen unter SOLARIS
  - an Stelle von Blades kann man Zones definieren
  - Die beiden Master-Zones auf unterschiedlicher Hardware verteilen
- Arbeits-Blades bzw. -Zones werden sooft vervielfältigt wie die Benutzerlast es erfordert und die Hardware es erlaubt.



16. November 2009

# Reverse-Proxy + Cache

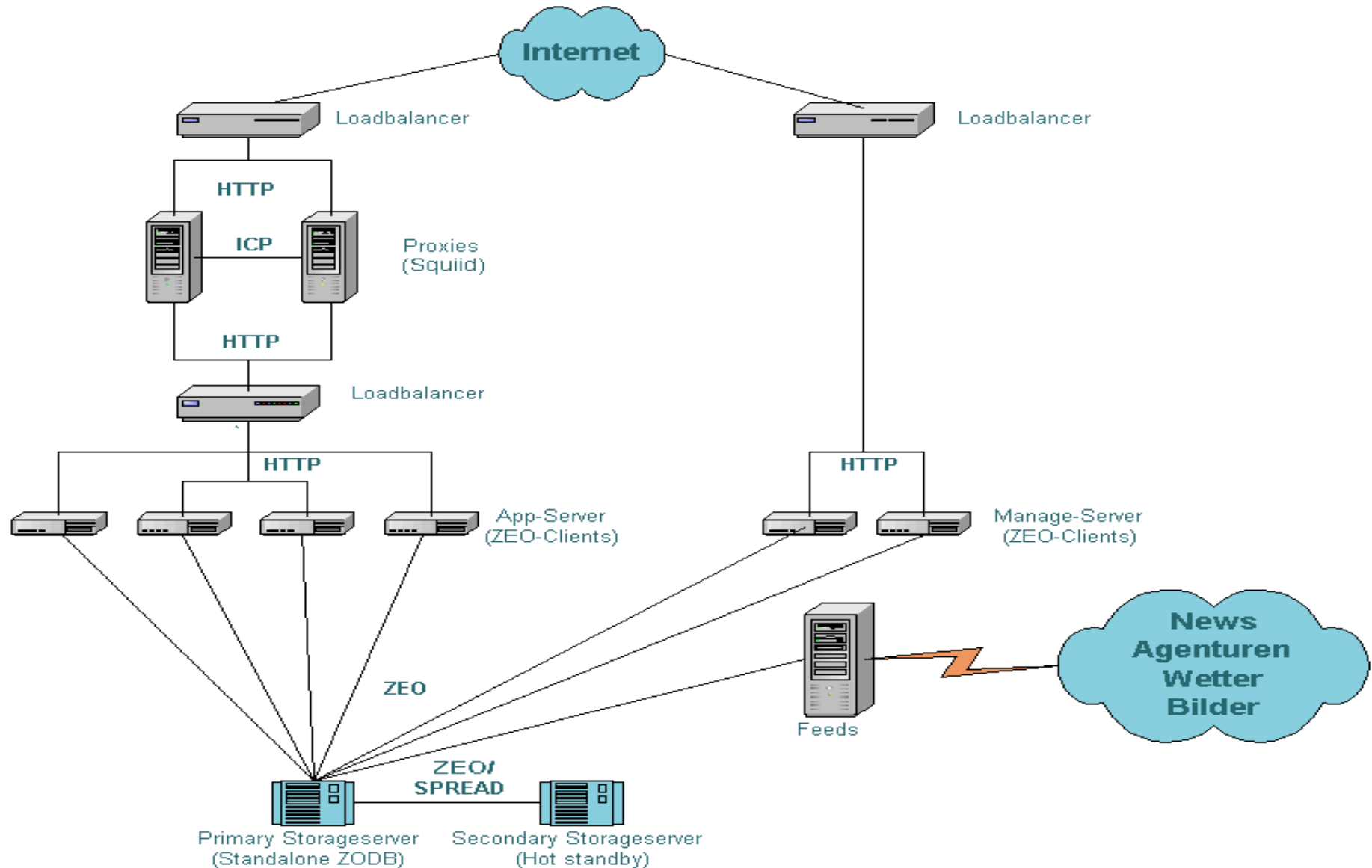
# Web-Server als Reverse-Proxy/Cache-Frontend

- Normaler Web-Server leitet Anfragen an Zope weiter.
- Vorteile:
  - Robuster Vor-Filter für Anfragen (Abwehr von Angriffen)
  - Nutzt integrierten Cache zur Beschleunigung
  - Hostet rein statischen Content (Apache Faktor 1.000 schneller)
  - Unterstützt SSL / HTTPS – Protokolle
  - Integriert Load-Balancer für ZEO-Verteilung
- Beispiele:
  - Web-Server: Apache, Microsoft IIS,
  - Balancer: LH-Proxy, (Apache, Varnish)
  - Cache: Squid, Varnish, (Apache)

# Statischer + Dynamischer Server

- Aufteilen des Inhalts in statischen und in dynamischen Content
  - Statisch: CSS, JS, Bilder
  - Dynamisch: Texte, Main-Content, Portlets
- Eigenen Bereich für statischen Inhalt
  - Eigene URL, z.B. <http://static.meinweb.de>  
dynamischer Teil unter <http://www.meinweb.de>
  - Eigener Pfad in der Domain, z.B. <http://www.meinweb.de/static>  
Reverse-Proxy unterscheidet, ob er weiterleitet
  - Eigene URL: lädt schneller wg. Session per Server  
Eigener Pfad: weniger Sicherheitsbedenken
- Vor- und Nachteile:
  - +: schnelleres Laden; entlastet CMS mit Trivial-Anfragen
  - -: komplexe Konfiguration; Konsistenz der Applikation

# Revers-Proxy/Cache vor Zoep



# Fakten – Fakten - Fakten

- Performanz:
  - 10 Millionen Hits/Tag
  - ca. 900 Hits/Sekunde (Peak)
- Squid-Cache Hitrate: ~95 Prozent
- Situation am 11. September 2001:
  - Zope Cluster waren ausnahmslos erreichbar
  - Konkurrenzsites waren stundenlang nicht erreichbar

16. November 2009

# Zope-Support für Reverse-Proxy/Cache

- Virtual Host Monster
  - Zope-Product für URL-Rewriting
- Caching
  - Zope setzt Cache-Hinweise in HTTP-Header  
Cache ja/nein/dauer
  - Cache-Hinweise je nach Objekt-Typ, z.B. für Bilder
  - Cache registriert sich bei Zope  
Zope sendet Invalidate an Cache bei Datenänderungen

# Reverse Proxy-Konfiguration

- Einfache Konfiguration

```
<VirtualHost 85.214.56.233:80>
    ServerName ard.lexisnexus.at
    #Rewrite-Rules
    ProxyPass /
    http://localhost:10080/VirtualHostBase/http/ard.lexisnexus.at:80/Plone/ard/VirtualHostRoot/
</VirtualHost>
```

# Reverse Proxy-Konfiguration

- Konfiguration mit Load-Balancer

```
<VirtualHost 85.214.56.233:80>

    ServerName www.fonstdok.com

    <Location /balancer-manager>
        SetHandler balancer-manager
        Order Deny,Allow
        Allow from all
    </Location>

    <Proxy balancer://lb>
        #/home/fonstdok/clients/2
        BalancerMember http://localhost:8084 route=8084"
        #/home/fonstdok/clients/3
        BalancerMember http://localhost:8086 route=8086"
    </Proxy>

    ProxyPass /fdImages !
    ProxyPass /fdStyles !
    ProxyPass /fdJavascript !
    ProxyPass /balancer-manager !
    ProxyPass /
        balancer://lb/VirtualHostBase/http/www.fonstdok.com:80/FondsDok/VirtualHostRoot/
        stickysession=STICKY_ROUTE
```

# Reverse Proxy-Konfiguration

- Noch ein paar Extras und Alternativen

```
<IfModule mod_expires.c> # HTTP-Header auf Cache-im-Client setzen
ExpiresActive On
ExpiresByType image/gif "access plus 14 days"
ExpiresByType image/png "access plus 14 days"
ExpiresByType image/jpeg "access plus 14 days"
.....
ExpiresByType application/x-javascript "access plus 14 days"
</IfModule>
```

```
<IfModule mod_disk_cache.c> # Seiten lokal statisch cachen
CacheEnable disk /
CacheForceCompletion 100
CacheLastModifiedFactor 0.1
CacheDefaultExpire 1

CacheRoot /var/cache/apache2/nordlb
CacheSize 10000
CacheGcInterval 24
CacheExpiryCheck On
CacheDirLength 2
</IfModule>
```

```
<IfModule mod_deflate.c> # Alle Daten „zippen“
SetOutputFilter DEFLATE
DeflateCompressionLevel 5
</IfModule>
```

```
</VirtualHost>
```

# Caching und Benutzerrechte

- View hängt von der URL und mehr ab
  - Hängt ab von angemeldeten Account
  - Hängt ab von vorangegangenen Transaktionen, gespeichert im internen SESSION-Objekt
- Caches haben Mühe diese MEHR zu erkennen
  - Abhängigkeiten im HTTP-Header und in Cookies sichtbar machen
  - Cache so konfigurieren, dass er alle Abhängigkeiten berücksichtigt
  - RAM-Cache erkennt alle Abhängigkeiten.
- Einfache Cache-Strategien für Web-Server
  - Alle anonymen Anfragen cachen
  - Angemeldete Anfragen über andere, nicht-gecachte URL laufen lassen

# Zope-Support für Reverse-Proxy/Cache

- Etags: Cache nach Inhalt
  - Normales Cache-Kriterium: Name, Alter (TTL), Änderungsdatum
  - Portal-Rahmen: anderes Datum im Calendar-Portlet
  - Alternative: Alfa-numerischer Schlüssel  
Beispiel: Version oder MD5
  - Wird als ETAG im Header übermittelt
  - Dokumenttyp kann eigene ETAG-Methode implementieren  
--> inhaltliche Entscheidung, ob ETAG sich ändert
- Edge Side Includes ESI: partielle Web-Seiten Cachen
  - Spezielle XML-Tags (ESI-ML) in View-Methoden
  - Separates Caching z.B. der Portlets und des Contents
  - Support: Squid, Oracle AS Web Cache

# ESI - Beispiel

The screenshot shows the AT&T website interface. At the top left is the AT&T logo. To its right is a banner for "AT&T Online Billing - Simple, Secure, Convenient" with a small image of people. Below this is a search bar with the text "Enter Search Term or AT&T Keyword" and a "GO" button, followed by a link to "A-Z Index".

The main navigation area is divided into four columns: "consumer", "business", "wireless", and "broadband". Each column has a sub-header and a list of services. For example, under "consumer", there are links for "Switch to AT&T now!", "AT&T Unlimited Plan", "Internet Access", and "PrePaid Phone Cards".

On the left side, there is a vertical menu with links: "ABOUT AT&T", "ACCOUNT MANAGEMENT", "EMPLOYMENT", "INVESTOR RELATIONS", "WORLDWIDE AT&T", "POPULAR LINKS", "DIRECTORIES", and "CONTACT AT&T".

The "NEWSROOM" section is highlighted with a black box and an arrow labeled "News". It contains three news items, each with a red link: "AT&T Long Distance Customers in The Granite State Get the Message: Thanks for Your Loyalty", "AT&T Long Distance Customers in The Diamond State Get the Message: Thanks for Your Loyalty", and "AT&T Reaction To Minnesota A.L.J. Recommendation On Qwest Secret Deals".

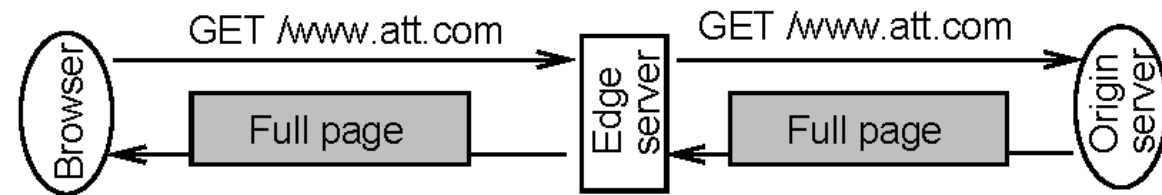
Below the news items is a "features" section with a heading "FREE Accelerating 3D Graphic Game - Take the Unlimited Challenge!" and a paragraph: "Navigate through the network and see if you have what it takes to make a connection." Below this is a link to "Take a quick video tour of AT&T e-Service for business".

At the bottom of the page, there is a "STOCKS" section with a black box and an arrow labeled "Stocks". It displays two stock price updates: "AT&T: 12.09 +0.17 Sep 26, 10:51 am ET" and "AWE: 4.51 +0.04 Sep 26, 10:51 am ET". Below the stock prices is a link "Stock Info Source".

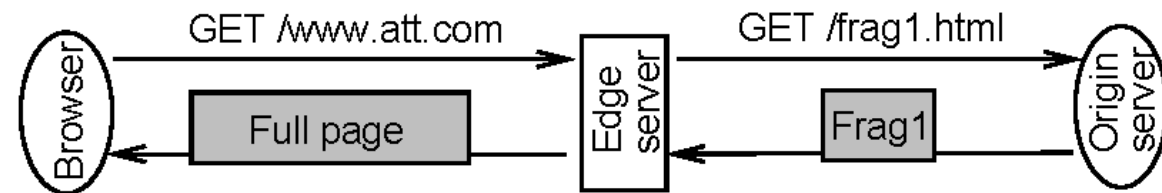
At the bottom left of the page, there is a small advertisement for "The Store @ AT&T" with the text "GREAT VALUE FROM A BRAND YOU CAN TRUST... Buy AT&T products like telephones, accessories & more."

# ESI - Beispiel

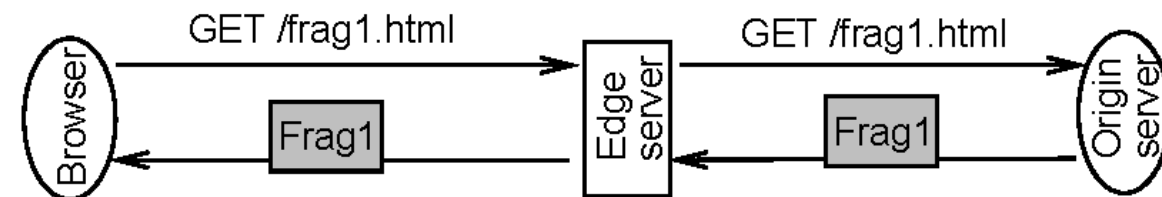
```
1. <HTML>
2. <!-- esi
3. <H3>Stock quote for ${QUERY_STRING}</H3>
4. <esi:try>
5. <esi:attempt>
6.   <esi:include src=/quote.html
7.     alt=/delayed_quote.html/>
8. <esi:choose>
9.   <esi:when
10.    test="$(HTTP_COOKIE{Type})=premium">
11.    <esi:include src=/market_news.html/>
12.  </esi:when>
13.  <esi:otherwise>
14.    To subscribe to premium services
15.    <A href=/subscribe.html> click here </A>
16.  </esi:otherwise>
17. </esi:choose>
18. </esi:attempt>
19. <esi:except>
20.   <esi:include src=/sorry.html />
21. </esi:except>
22. </esi:try>
23. -->
24. <esi:remove>
25.   Please click on a
26.   <A href=/non_esi.html> non-ESI version </A> of this site
27. </esi:remove>
</HTML>
```



(a) No ESI



(b) ESI with edge-side page assembly



# Zope-interner RAM-Cache

- Partielles Caching in Zope
  - Zwischenspeicher vor der View-Methode
  - Entscheidet pro Objekt, ob die view neu berechnet wird.
- Beispiel: Calendar-Portlet
  - Wird nur beim ersten Aufruf am Tag berechnet
  - Wird neu berechnet, wenn jemand einen Kalender-Eintrag ändert
  - Wird sonst aus dem Cache abgerufen

## Internationalisierung

## Aktive Suche