

Struktur von Kapitel V – Beispiele bzw. Dienste

- I Einleitung
- II Grundlegende Kommunikationsdienste
- III Middleware
- IV Architekturen & Algorithmen
 - A Synchronisierung
 - B Konsistenz und Replication
 - C Fehlertoleranz
- V Beispiele bzw. Dienste
 - A Verteilte Dateisysteme
 - B Namensdienste
- VI Sicherheit & Sicherheitsdienste
- VII Zusammenfassung

1. Anforderungsanalyse, Schnittstelle & Architektur
2. Kommunikation
3. Synchronisierung, Konsistenz & Replikation
4. Sicherheit
5. Alternativen
6. Zusammenfassung



Anforderungsanalyse - Motivation

- Warum ein verteilten Dateisystem?

- **Kapazität:** Die Summe verschiedene lokale Dateisysteme = ein großes Dateisystem
- Unterstützung plattenlosen Geräte.
- Am Wichtigsten: **Gemeinsame Verwendung** – Komponenten eines verteilten Systems sollen ein Dateisystem *teilen* können.

Bzw. ein *entfernter Dateidienst*
(Remote File Service)

Anforderungsanalyse – Transparenz (1)

- **Zugriffstransparenz:**
Der Zugriff auf entfernte Dateien erfolgt für Klienten mit den gleichen Operationen, wie der Zugriff auf lokale Dateien.
- **Ortstransparenz:**
Es spielt keine Rolle, an welchem physikalischen Ort eine Datei gespeichert ist.
- **Nebenläufigkeitstransparenz:**
Greifen mehrere Klienten gleichzeitig auf eine Datei zu, soll kein Klient von den Zugriffen der anderen etwas mitbekommen.
Allerdings: nebenläufige *schreibende* Zugriffe sind problematisch.
- **Fehlertransparenz:**
Für Klienten, wie für Server, soll ein Fehlverhalten des jeweils anderen nicht zu Inkonsistenzen und Verklemmungen führen.
- **Lasttransparenz:**
Die Anfragelast auf Serverseite soll sich nicht auf dessen Antwortzeiten auswirken.
- **Hardware- und Betriebssystem-Transparenz:**
Das verteilte Dateisystem soll die *Heterogenität* unterschiedlicher Hardware und Betriebssysteme verbergen.

Anforderungsanalyse – Transparenz (2)

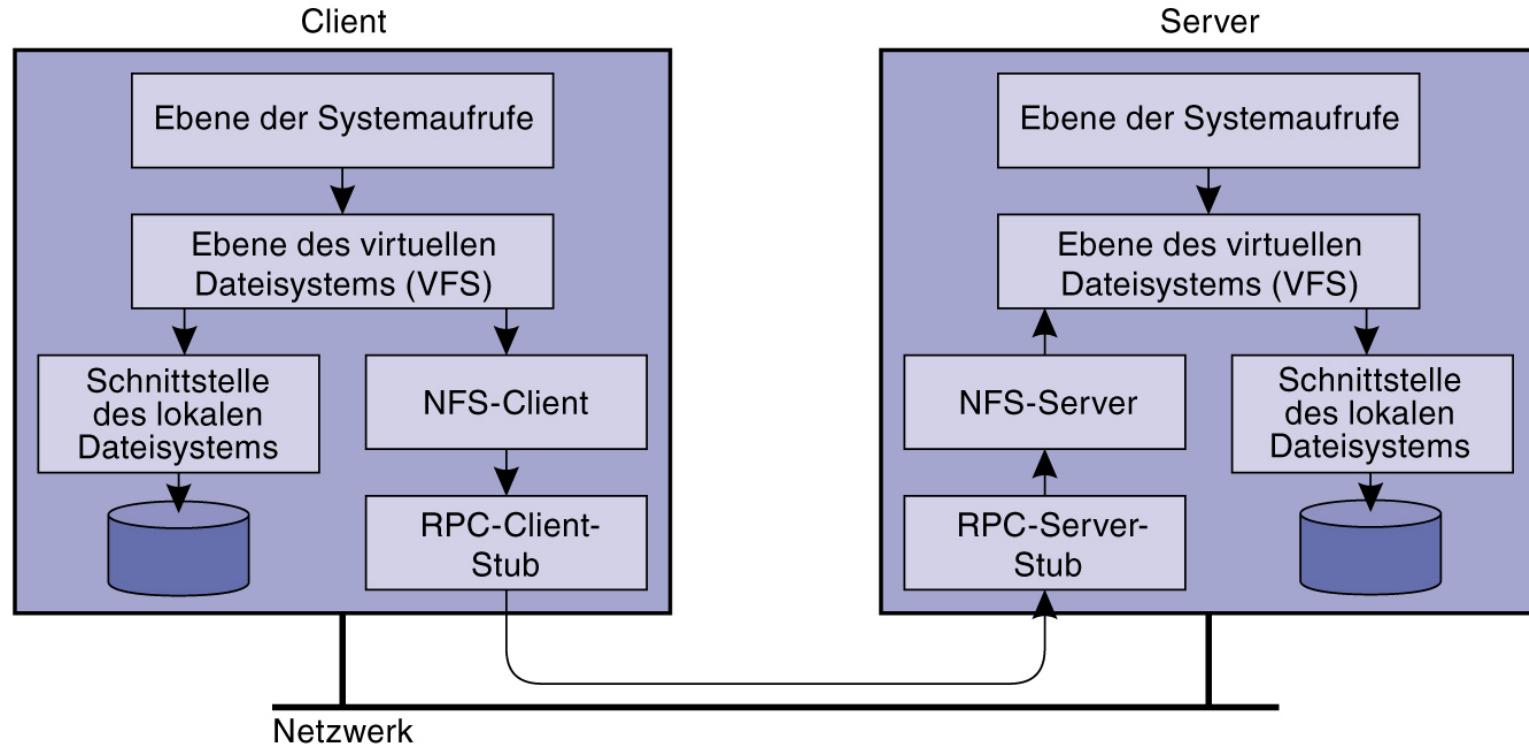
- **Replikationstransparenz:**
Der Klient darf immer nur eine logische Datei sehen, egal wie häufig und wo diese repliziert vorgehalten wird.
- **Migrationstransparenz:**
Verändert eine Datei ihren Speicherungsort, dann ist dies für den Klienten nicht sichtbar.
- **Partitionierungs- bzw. Erreichbarkeits-transparenz:**
Das Dateisystem kann partitioniert werden, d.h. bestimmte Dateiserver sind nicht mehr (direkt-) zugreifbar für bestimmte Klienten. Trotzdem stehen für alle Klienten noch alle benötigten Dateien zur Verfügung. Eine Vereinigung bisher getrennter Partitionen erzeugt dabei möglichst keine Inkonsistenzen.

Besonders für die Anbindung **mobiler** Rechner interessant.

Nicht alle diese Anforderungen werden von den verfügbaren Systemen erfüllt!

Anforderungsanalyse – Integration

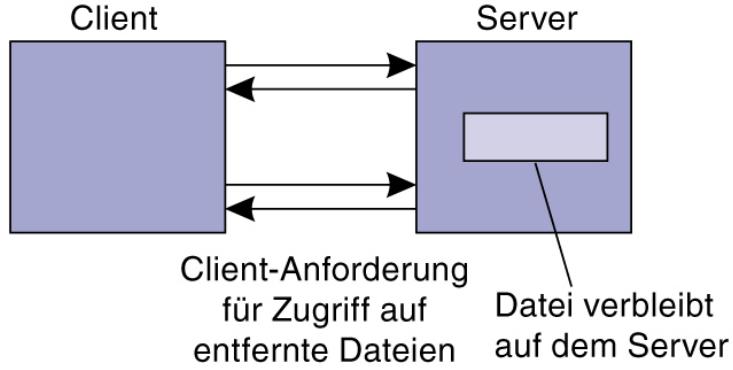
Integration eines verteilten Dateisystems in ein existierendes (lokalen) Dateisystem muß *nahtlos* erfolgen (können):



Beispiel: NFS (Network File System) in UNIX-Systemen

NFS wurde ursprünglich von Sun entwickelt, dient diesem Kapitel als Hauptbeispiel. Version 1 wurde 1983 entwickelt, Version 4 wurde in 2003 vereinbart (& 4.1 in 2010) als RFC.

Zugriffsmodelle

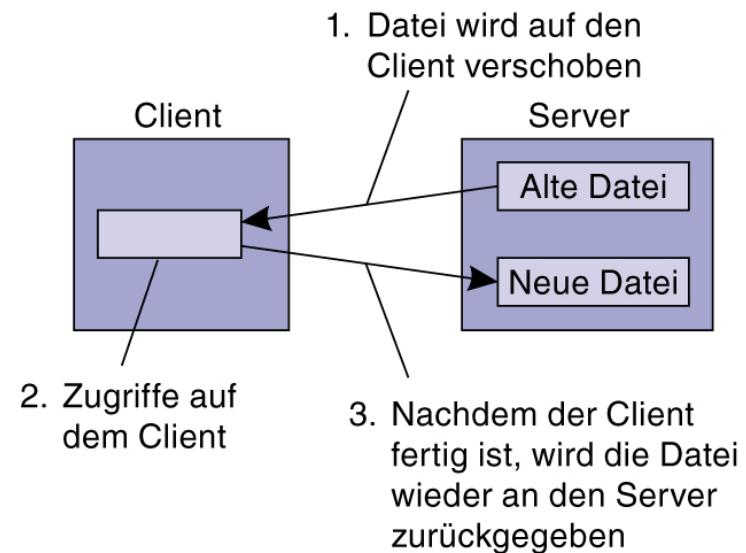


a

Remote-Access-Modell

- Alle Zugriffe eines Klienten auf eine Datei werden einzeln auf dem entfernten Server ausgeführt.

Von NFS verwendet.



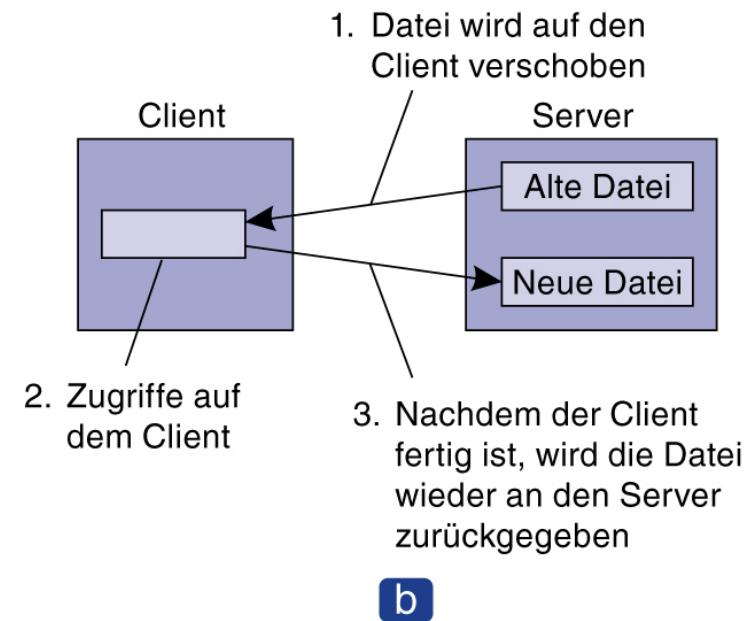
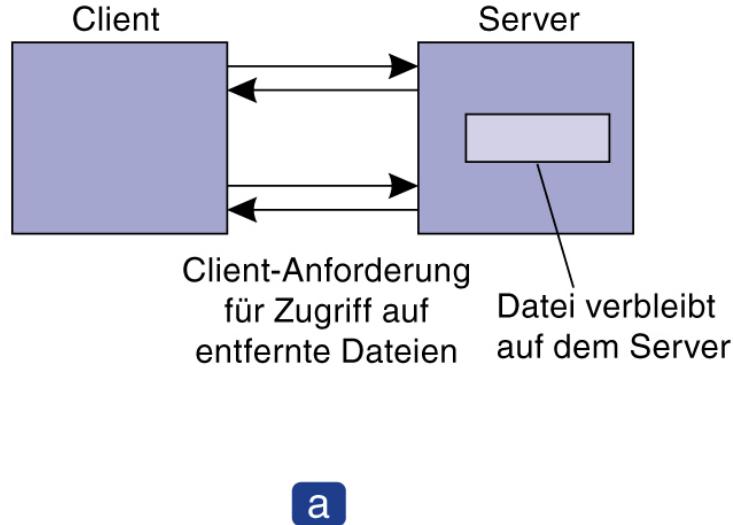
b

Upload/Download-Modell:

- Beim Öffnen einer Datei lädt der Server die Datei vollständig zum Klienten (download)
Benötigt der Klient die Datei nicht mehr, schließt er sie und sie wird zum Server zurück übertragen (upload).

Vgl. FTP

Zugriffsmodelle: Bewertung



Remote-Access-Modell

- Die meiste Funktionalität liegt auf der Serverseite.
- Die Schnittstelle zwischen Klient und Server benötigt viel Funktionalität.
- Das Netzwerk wird fortwährend belastet, allerdings mit jeweils kleinerem Volumen.

Upload/Download-Modell:

- sehr einfache Schnittstelle.
- Alle Dateioperationen müssen im Klient implementiert werden.
- Der Klient benötigt ausreichend Platz zum Zwischenspeichern .
- Netzwerk wird nur beim Öffnen und Schließen von Dateien belastet - dann aber mit großen Volumen..

h_da

HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES

fbi

FACHBEREICH INFORMATIK



Schnittstelle: NFS 3 & 4

Operation	v3	v4	Beschreibung
Create	Ja	Nein	Erstellen einer regulären Datei
Create	Nein	Ja	Erstellen einer irregulären Datei
Link	Ja	Ja	Erstellen einer direkten Verknüpfung zu einer Datei
Symlink	Ja	Nein	Erstellen einer symbolischen Verknüpfung zu einer Datei
Mkdir	Ja	Nein	Erstellen eines Unterverzeichnisses in einem gegebenen Verzeichnis
Mknod	Ja	Nein	Erstellen einer Spezialdatei
Rename	Ja	Ja	Ändern einer Dateibezeichnung
Remove	Ja	Ja	Entfernen einer Datei aus einem Dateisystem
Rmdir	Ja	Nein	Entfernen eines leeren Unterverzeichnisses aus einem Verzeichnis
Open	Nein	Ja	Öffnen einer Datei
Close	Nein	Ja	Schließen einer Datei
Lookup	Ja	Ja	Suchen einer Datei anhand ihrer Bezeichnung
Readdir	Ja	Ja	Lesen der Einträge eines Verzeichnisses
Readlink	Ja	Ja	Auslesen der in einer symbolischen Verknüpfung gespeicherten Pfadangabe
Getattr	Ja	Ja	Auslesen der Attributwerte einer Datei
Setattr	Ja	Ja	Setzen eines oder mehrerer Attributwerte für eine Datei
Read	Ja	Ja	Auslesen der in einer Datei enthaltenen Daten
Write	Ja	Ja	Schreiben von Daten in eine Datei

Die Liste ist *unvollständig*.

In NFS 3 folgte ein **read** (**write**) direkt auf ein **lookup**, ohne **open**.

In NFS 4 erzeugt **open** eine Datei wenn notwendig – also wird **create** nur für irreguläre Dateien (Verzeichnisse, Symlinks, Spezialdateien) verwendet.

Die Funktionalität von **rmdir** wird in NFS 4 von **remove** übernommen.

NFS 3 war **zusandslos**.
NFS 4 ist **zustandbehaftet**.



Zustandslos oder Zustandsbehaftet?

Vorteile eines zustandslosen Dateiservers

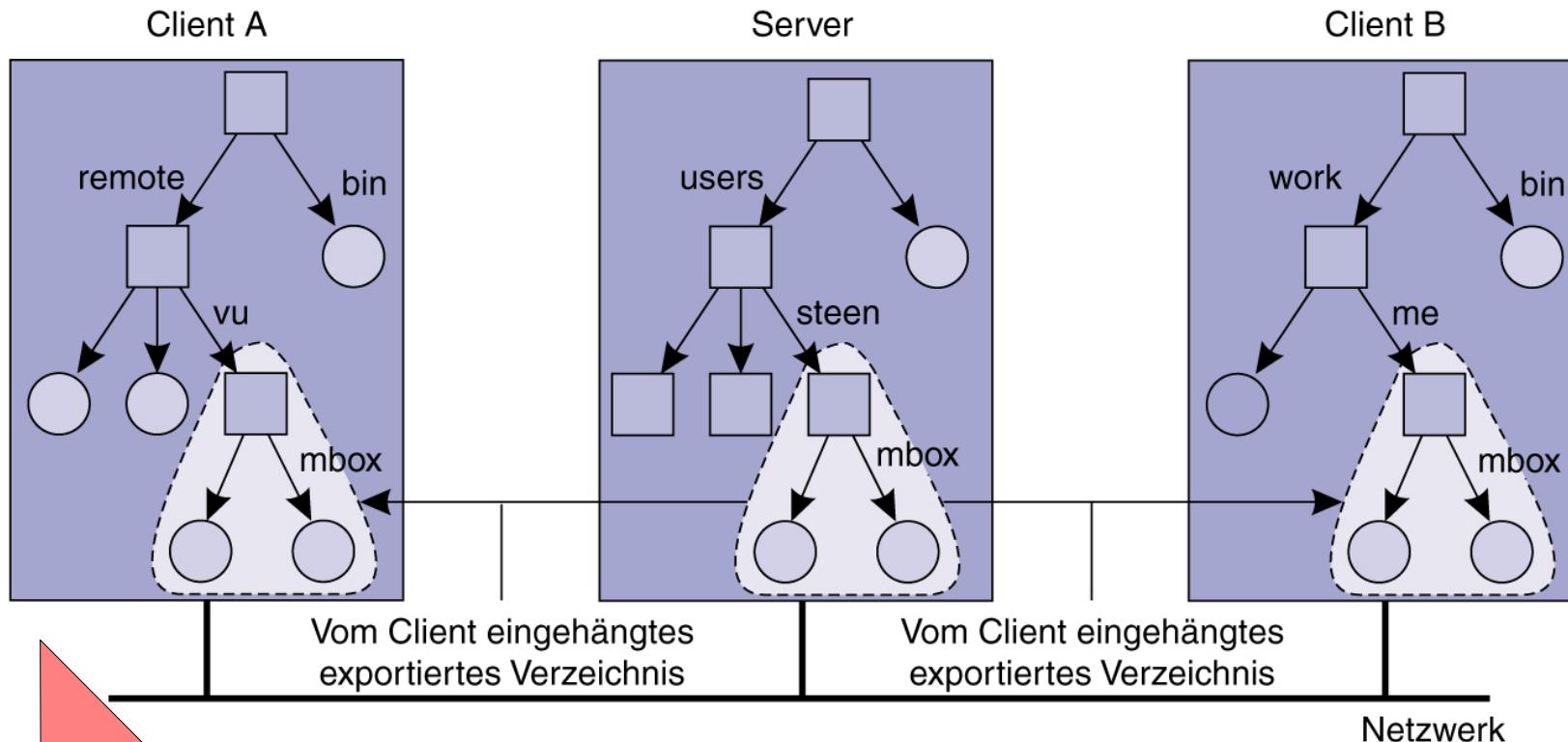
- Benötigt keinen Speicher für **Klienteninformation**.
- Operationen zum **Öffnen** oder **Schließen** von Dateien sind unnötig.
- Das System kann **leichter fehlertolerant** realisiert werden. Ohne Austausch von Zustandsinformationen können Replikations- und Backupserver einfacher realisiert werden.
- Beim **Absturz eines Klienten** entstehen für den Server keine Probleme, da keine verwaiste Information vorhanden ist.

Durch Precaching holt der Server bereits Daten der zugegriffenen Datei von seiner Platte, die der Klient noch gar nicht angefordert hat

Vorteile eines zustandsbehafteten Dateiservers

- Kürzere **Nachrichten genügen**, da nicht mehr in jedem Zugriff alle benötigten Klienteninformationen übertragen werden müssen.
- **Schreib- und Lesezugriffe sind effizienter**, da z.B. die Positionszeiger einer im Zugriff befindlichen Datei bereits am richtigen Platz stehen.
- Es ist **Precaching** möglich, da alle Dateien bekannt sind, die sich im Zugriff befinden.
- Durch Kenntnis des Benutzungsprofils ist **intelligentes Precaching** realisierbar. Z.B. benötigt ein Compiler immer seine Bibliotheksdateien.
- **Dateisperren** können vom Server unterstützt werden.

Schnittstelle: Namen (in NFS)

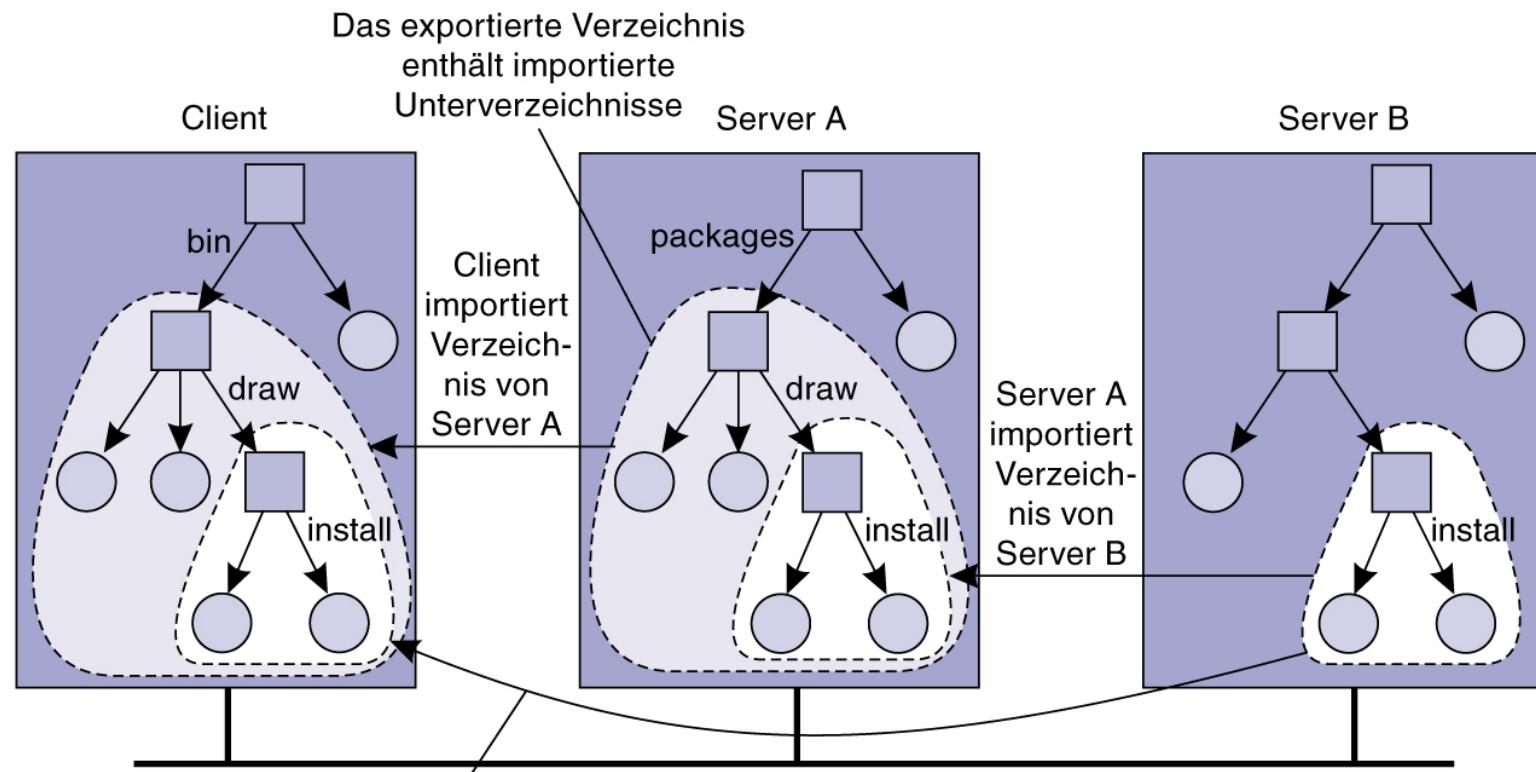


/remote/vu & /work/me
sind *Mount Points*

Frage: Ist das transparent? Wie könnten wir es transparenter machen?

Frage: Was ist wenn ein Server auch ein Client ist
- wenn ein Server ein Verzeichnis importiert?

Schnittstelle: Namen (fortgesetzt)

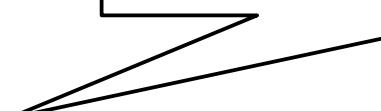


Regel (in NFS): Server A darf die Dateien von Server B nicht einfach weiterleiten.

Wie soll **lookup** hier erfolgen?
(Die Antwort kommt später...)



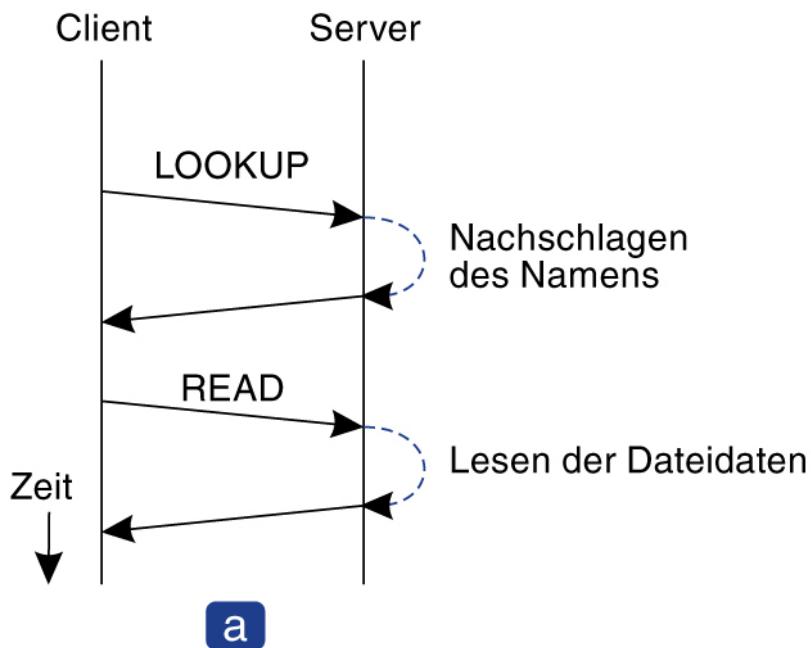
Struktur von Kapitel V – Beispiele bzw. Dienste

- I Einleitung
 - II Grundlegende Kommunikationsdienste
 - III Middleware
 - IV Architekturen & Algorithmen
 - A Synchronisierung
 - B Konsistenz und Replication
 - C Fehlertoleranz
 - V Beispiele bzw. Dienste
 - A Verteilte Dateisysteme
 - B Namensdienste
 - VI Sicherheit & Sicherheitsdienste
 - VII Zusammenfassung
1. Anforderungsanalyse, Schnittstelle & Architektur
 2. Kommunikation
 3. Synchronisierung, Konsistenz & Replikation
 4. Sicherheit
 5. Alternativen
 6. Zusammenfassung
- 

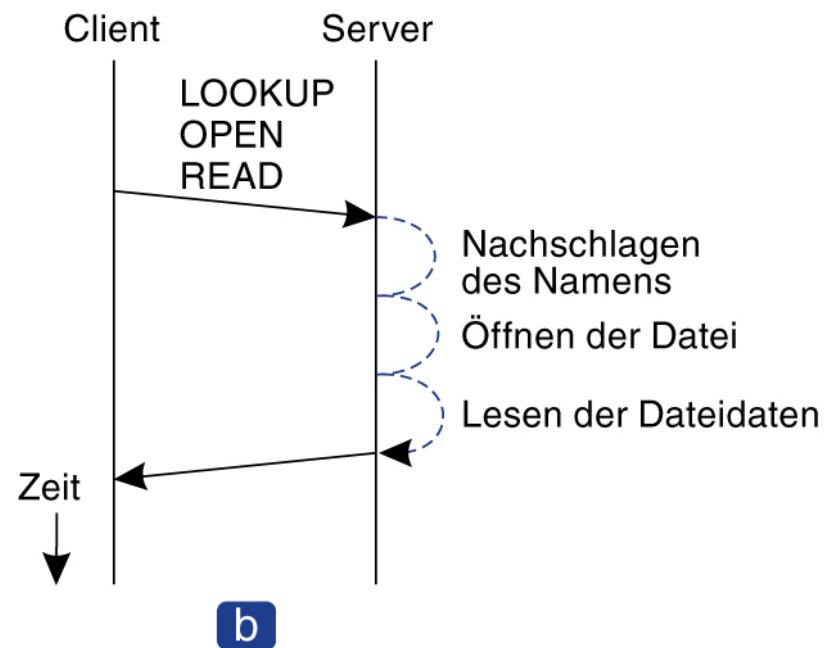
Kommunikation mittels RPC

NFS wird mit RPC realisiert.

Erweiterung: Seit NFS v4 können verschiedene *Remote Procedures* gleichzeitig aufgerufen werden.



(a) Lesen von Daten aus einer Datei in NFS v3: 2 RPCs



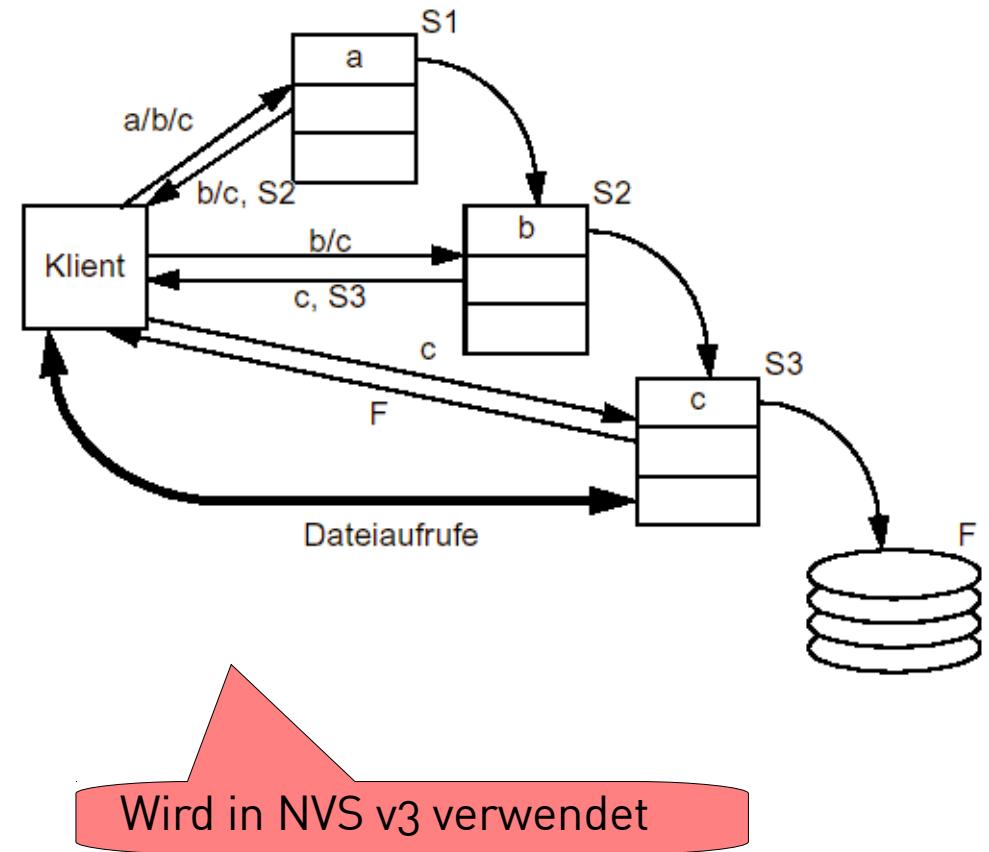
(b) Lesen von Daten aus einer Datei in NFS v4: 1 RPC

Auflösen von Dateinamen: Iterativ

Iteratives Durchlaufen:

- Der Klient fragt beim Server, für den der erste Teil des Pfades passt, nach der Datei.
- Kann dieser den Zeiger auf die eigentliche Datei nicht liefern, weil er nicht den gesamten benötigten Verzeichnisbaum bei sich hält, gibt er dem Klienten den Pfad und Server zurück, bei dem der Klient weiter nachfragen kann.
- Dieses Vorgehen wiederholt der Klient, bis er den Zeiger auf die Datei erhält.
- Dann kann der Klient direkt über den verwaltenden Server auf die Datei zugreifen.

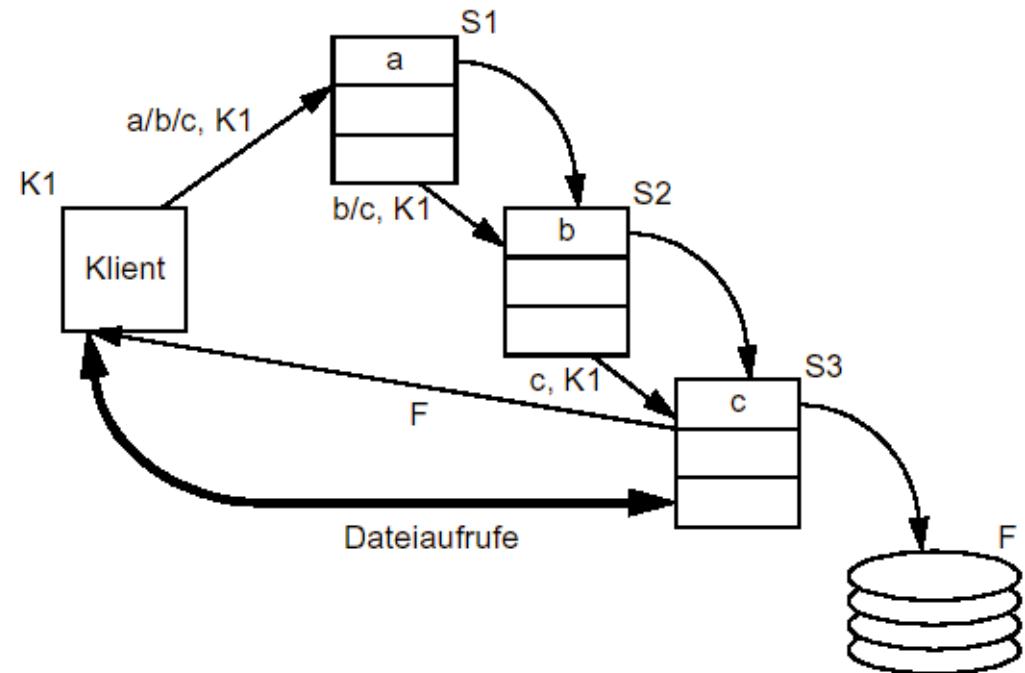
Der Navigationsalgorithmus ist vollständig auf Klientenseite implementiert.



Auflösen von Dateinamen: Rekursiv

Rekurses Durchlaufen:

- Der Klient stellt an den Server die Anfrage bezüglich einer Datei.
- Kann der Server die Anfrage nicht beantworten, so gibt er sie mit der Information, welcher Klient die Anfrage gestellt hat, an den nächsten Server weiter.
- Ist der gesamte Pfadname aufgelöst, schickt der letzte Server den gewünschten Dateizeiger zum Klienten zurück.
- Die Dateizugriffe können dann zwischen Klient und Server erfolgen.

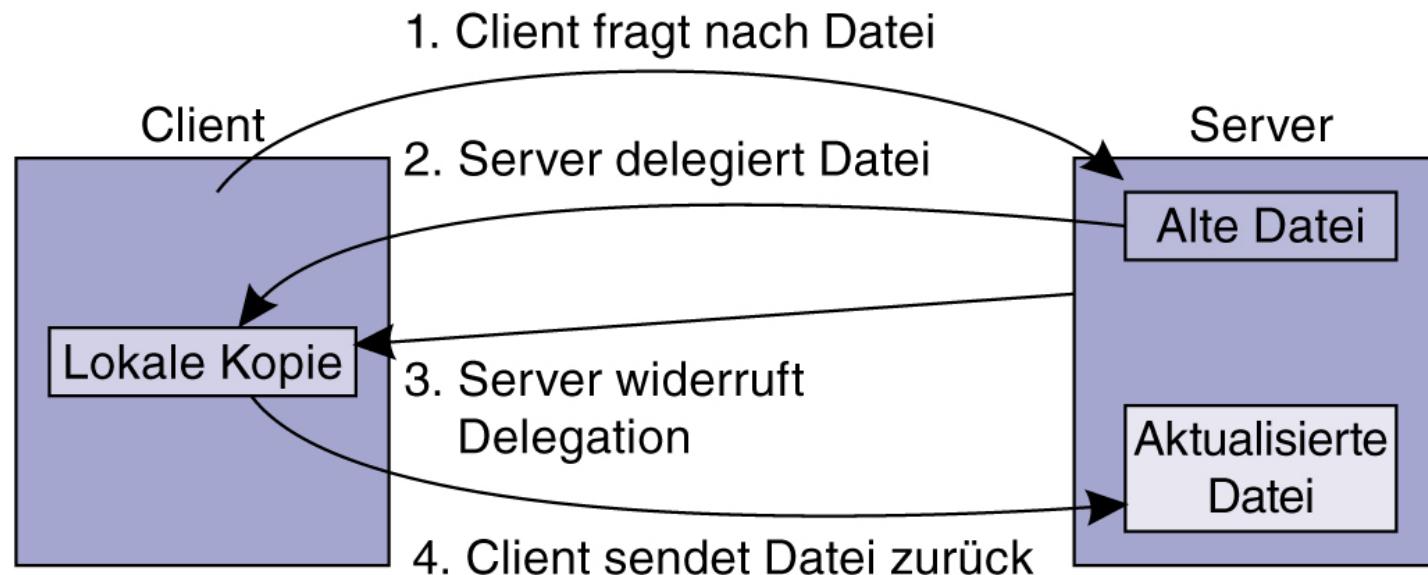


Wird in NVS v4 verwendet

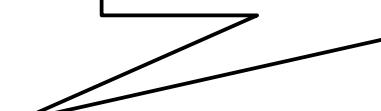
Delegieren des Öffnens von Dateien

Ein Client kann in NFSv4 eine Datei komplett übernehmen.

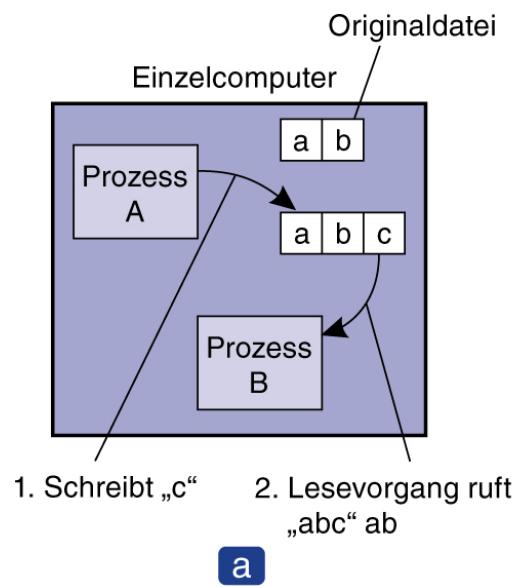
Allerdings muss der Server das Delegieren widerrufen können.



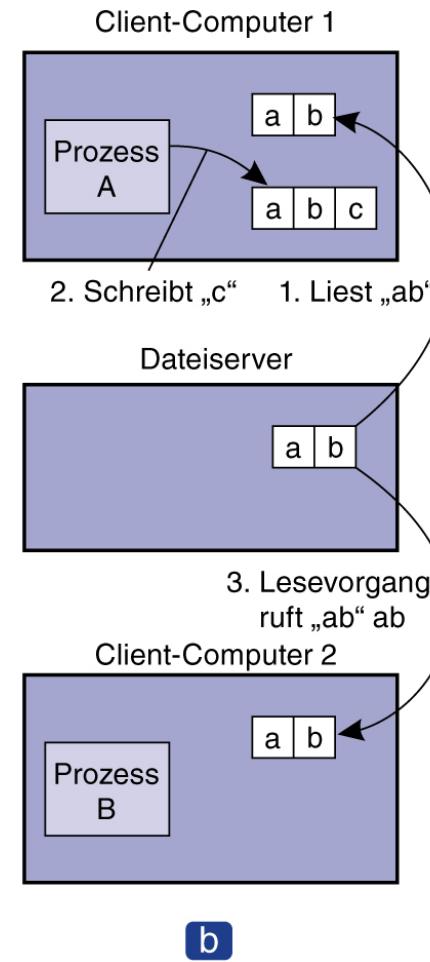
Struktur von Kapitel V – Beispiele bzw. Dienste

- I Einleitung
 - II Grundlegende Kommunikationsdienste
 - III Middleware
 - IV Architekturen & Algorithmen
 - A Synchronisierung
 - B Konsistenz und Replication
 - C Fehlertoleranz
 - V Beispiele bzw. Dienste
 - A Verteilte Dateisysteme
 - B Namensdienste
 - VI Sicherheit & Sicherheitsdienste
 - VII Zusammenfassung
- 1. Anforderungsanalyse, Schnittstelle & Architektur
 - 2. Kommunikation
 - 3. Synchronisierung, Konsistenz & Replikation
 - 4. Sicherheit
 - 5. Alternativen
 - 6. Zusammenfassung
- 

Synchronisierung – das Problem



(a) Einzelprozessor: Zuerst wird geschrieben, danach gelesen. Kein Problem.



(b) Verteiltem System mit Cache:
Client-Computer 2 bekommt eine veraltete Wert. Problem?

Semantik des Dateisharings

Methode	Kommentar
Auch „Single-Copy“ genannt.	
UNIX-Semantik	Jede Änderung an einer Datei ist sofort für alle Prozesse sichtbar.
Sitzungssemantik	Keine Veränderung ist sichtbar für andere Prozesse, bis die Datei geschlossen wird.
Unveränderbare Dateien	Aktualisierungen sind nicht möglich; vereinfacht die gemeinsame Verwendung und Replikation.
Transaktionen	Alle Veränderungen geschehen atomar.
Vgl. Datenbanken (s. Auch Kapitel IV).	Versionsführung Es kann verschiedene Versionen einer Datei geben, jede Version ist unveränderbar. Zusammenfügen ist ein Problem.

Dateisperren - Schnittstelle

In NFSv4 können Dateien gesperrt werden.

Es gibt Read- und Write- Sperren.

Sperren sind zeitlich beschränkt.

Es gibt nur 4 Operationen bzgl. Sperren (plus „Open“ mit „Freigabereservierung“).

Operation	Beschreibung
lock	Erstellen einer Sperre für eine Abfolge von Bytes
lockt	Testen, ob eine konkurrierende Sperre gewährt wurde
locku	Aufheben einer Sperre für eine Abfolge von Bytes
renew	Erneuern der Lease für eine angegebene Sperre

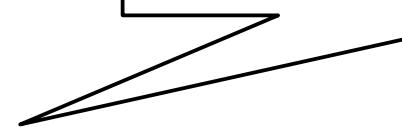
Dateisperren - Änderungen

Jede Datei kann mit einem „Verweigerungszustand“ geöffnet werden.

		a Aktueller Verweigerungszustand der Datei			
		NONE	READ	WRITE	BOTH
Verlangter Zugriff	READ	Gewährt	Verweigert	Gewährt	Verweigert
	WRITE	Gewährt	Gewährt	Verweigert	Verweigert
	BOTH	Gewährt	Verweigert	Verweigert	Verweigert
		b Verlangter Verweigerungszustand der Datei			
Aktueller Zugriffs-zustand	READ	Gewährt	Verweigert	Gewährt	Verweigert
	WRITE	Gewährt	Gewährt	Verweigert	Verweigert
	BOTH	Gewährt	Verweigert	Verweigert	Verweigert

Das Ergebnis einer `open`-Operation mit Reservierungen für den gemeinsamen Zugriff in NFS:
(a) Wenn der Client bei jeweils bestehendem Verweigerungszustand einen gemeinsamen Zugriff verlangt;
(b) wenn der Client beim derzeitigen Zugriffszustand einen Verweigerungszustand verlangt

Struktur von Kapitel V – Beispiele bzw. Dienste

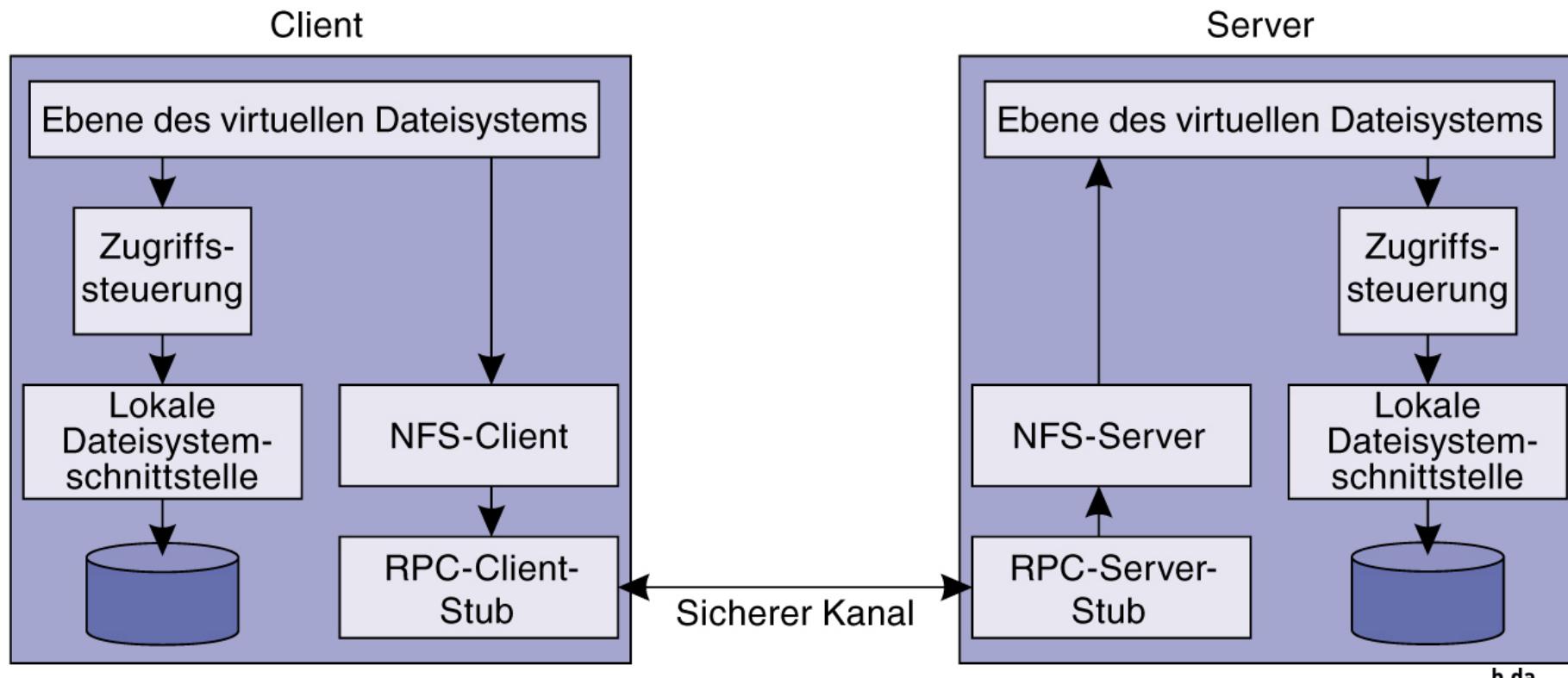
- I Einleitung
 - II Grundlegende Kommunikationsdienste
 - III Middleware
 - IV Architekturen & Algorithmen
 - A Synchronisierung
 - B Konsistenz und Replication
 - C Fehlertoleranz
 - V Beispiele bzw. Dienste
 - A Verteilte Dateisysteme
 - B Namensdienste
 - VI Sicherheit & Sicherheitsdienste
 - VII Zusammenfassung
- 1. Anforderungsanalyse, Schnittstelle & Architektur
 - 2. Kommunikation
 - 3. Synchronisierung, Konsistenz & Replikation
 - 4. Sicherheit
 - 5. Alternativen
 - 6. Zusammenfassung
- 

Sicherheit in NFS v < 4

Oft wird so gut wie *keine* Authentifizierung des Users gemacht: Der Client gibt dem Server eine Benutzer-ID und Gruppen-ID, und diese werden geprüft.

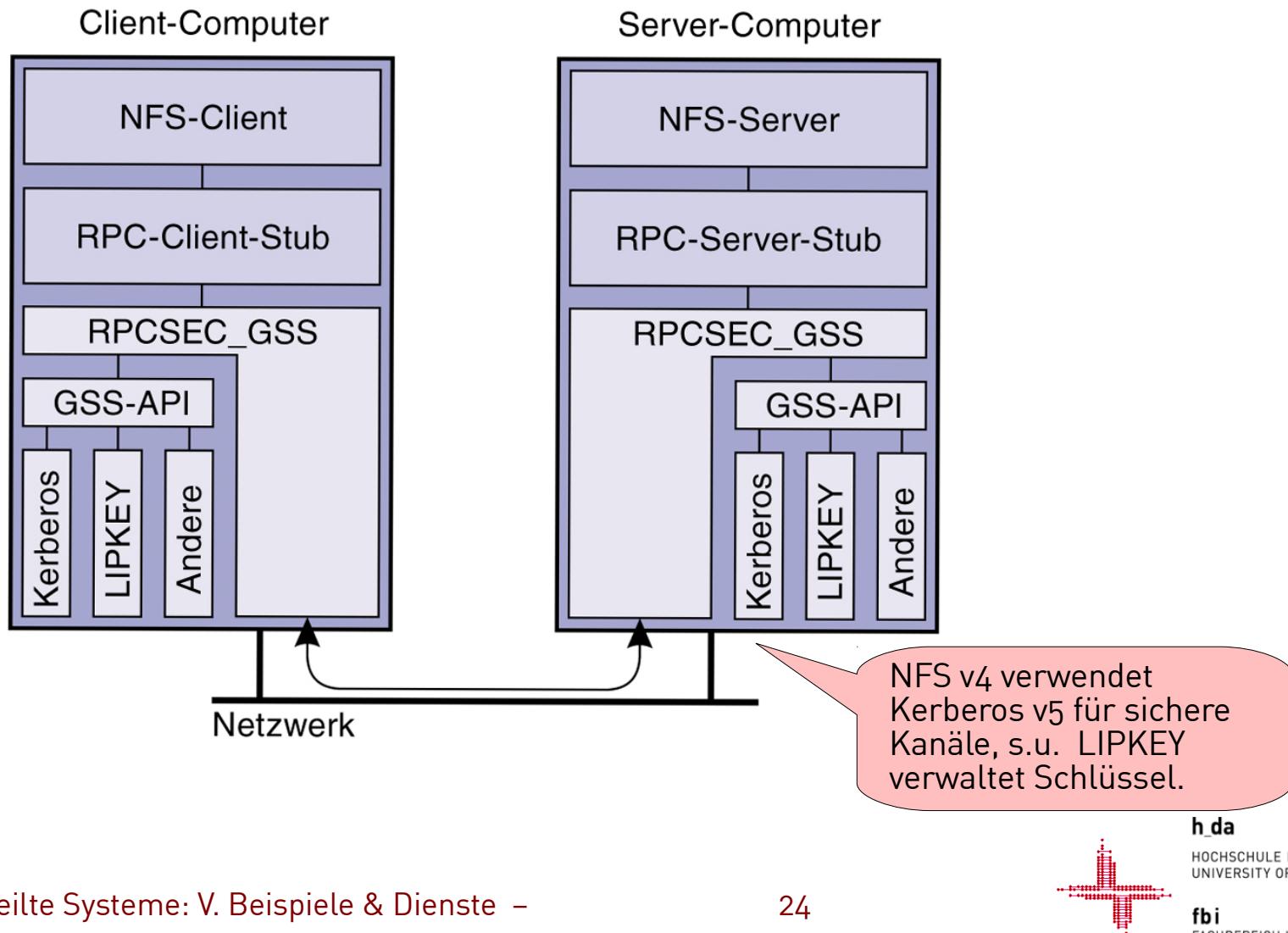
Danach erfolgt Zugriffssteuerung (bzw. Prüfung) zweimal – in Client und in Server.

NFS v3 (und ältere Versionen) kannten auch verschlüsselte Verbindungen – allerdings mit festgelegte Methoden & Schlüssel-länge (bald veraltet!).



Sicherheit in NFS v4

NFS v4 baut auf RPCSEC_GSS, ein *Plug-and-Play* Framework für *verschiedene* Sicherheitsmechanismen – flexibler! Aufgaben besser getrennt.



Arten von Benutzern in NFS

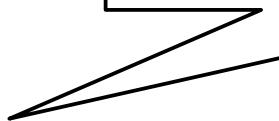
Rechtevergabe ist standardisiert. Z.B. kennt NFS nicht nur *User*, *Group*, *Everyone* bzgl. Rechtevergabe (vgl. Unix- bzw. Windows-Rechte).

Benutzertyp	Beschreibung
Owner	Der Besitzer einer Datei
Group	Die mit einer Datei verknüpfte Benutzergruppe
Everyone	Jeder beliebige Benutzer oder Prozess
Interactive	Jeder beliebige Prozess, der von einem interaktiven Endgerät aus auf die Datei zugreift
Network	Jeder beliebige Prozess, der vom Netzwerk aus auf die Datei zugreift
Dialup	Jeder beliebige Prozess, der über eine Einwählverbindung zum Server auf die Datei zugreift
Batch	Jeder beliebige Prozess, der im Rahmen eines Batch-Auftrages auf die Datei zugreift
Anonymous	Jeder, der ohne Authentifizierung auf die Datei zugreift
Authenticated	Jeder authentifizierte Benutzer oder Prozess
Service	Jeder systemdefinierte Dienstprozess

Struktur von Kapitel V – Beispiele bzw. Dienste

- I Einleitung
- II Grundlegende Kommunikationsdienste
- III Middleware
- IV Architekturen & Algorithmen
 - A Synchronisierung
 - B Konsistenz und Replication
 - C Fehlertoleranz
- V Beispiele bzw. Dienste
 - A Verteilte Dateisysteme
 - B Namensdienste
- VI Sicherheit & Sicherheitsdienste
- VII Zusammenfassung

1. Anforderungsanalyse, Schnittstelle & Architektur
2. Kommunikation
3. Synchronisierung, Konsistenz & Replikation
4. Sicherheit
5. Alternativen
6. Zusammenfassung



Alternativ-Architektur 0: AFS

AFS = Andrew File System,
nach Andrew Carnegie & Andrew Mellon benannt,
wurde an der Carnegie-Mellon University entwickelt.

Ziel: Ein verteilte Datei-System für eine ganze Universität. Infolgedessen...

Haupt-Designkriterium: Skalierbarkeit.

War sehr einflussreich, besonders auf NFS 4.

Wird heute noch verwendet z.B. in LinuxAFS, oder als DFS (von der Open Software Foundation).

AFS – Strategie (1)

Design basiert auf folgende Beobachtungen:

- Die meiste Dateien sind **klein** (unter 10 Kbyte);
- **Lesezugriffe** finden viel öfter statt als **Schreibzugriffe**
- Sequentielle Zugriffe finden viel öfter statt als wahlfreie Zugriffe.
- Die meisten Dateien werden nicht geteilt (sondern von nur einem Benutzer verwendet). Wenn geteilt, gibt es meistens nur ein Schreiber.
- Dateien werden in **Bursts** verwendet (lange nicht, dann öfter, dann wieder nicht....). Wenn eine Datei verwendet wird, wird es wahrscheinlich gleich wieder verwendet.

AFS – Strategie (2)

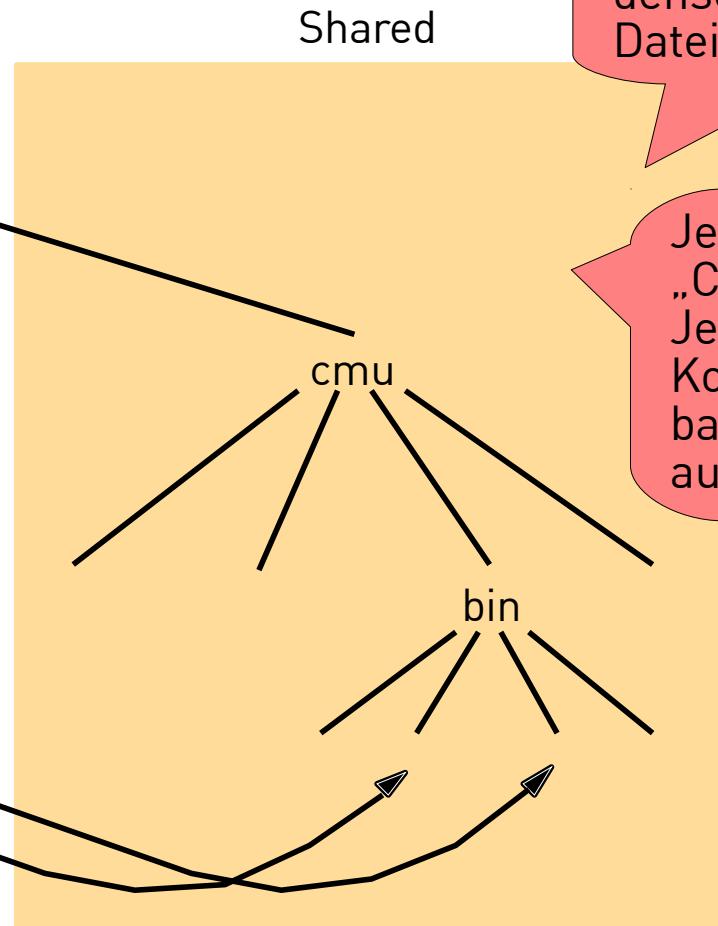
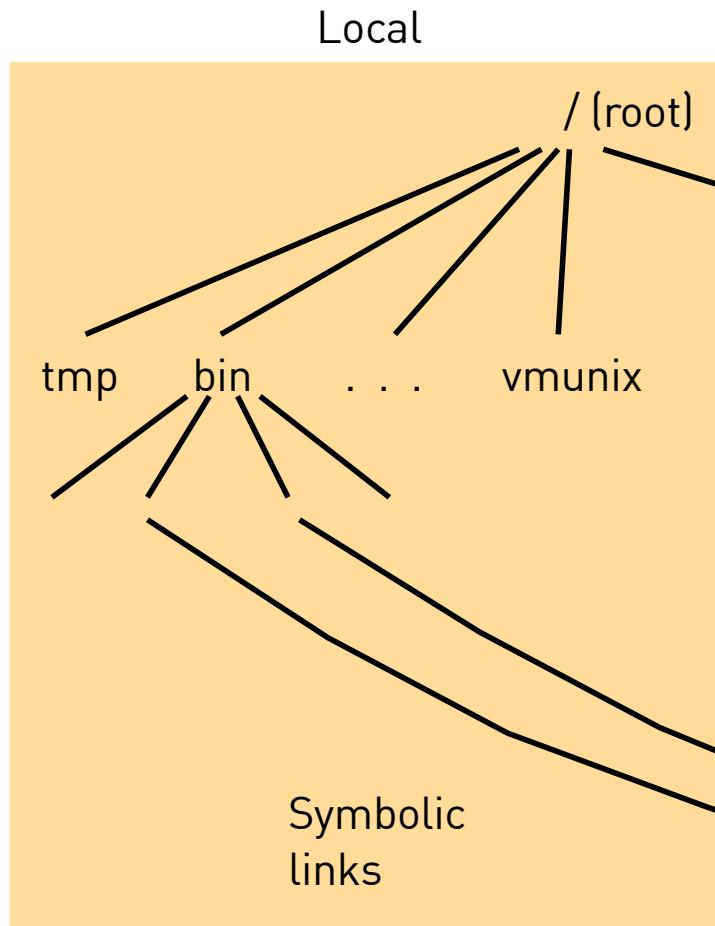
Also – zwei Strategien:

- **Whole-File Sharing:** Server schicken ganze Dateien bzw. Verzeichnisse zu Klienten (bzw. seit AFS-3, 64-KByte Blöcke, wenn Dateigröße > 64K)
- **Whole-File Caching:** Lokale Kopien werden beibehalten (auch nach Client-Reboot) und verwendet für zukünftige Dateizugriffe. An *close* wird die Datei zurück zu Server geschickt *und weiter lokal gehalten*.

AFS – Namen

Verteilte Dateien (und Verzeichnisse) werden von nicht-verteilte Dateien (und Verzeichnisse) getrennt.

Wie der Klient das Dateisystem sieht:



Jeder Klient sieht denselben „shared“ Dateibaum (vgl. NFS).

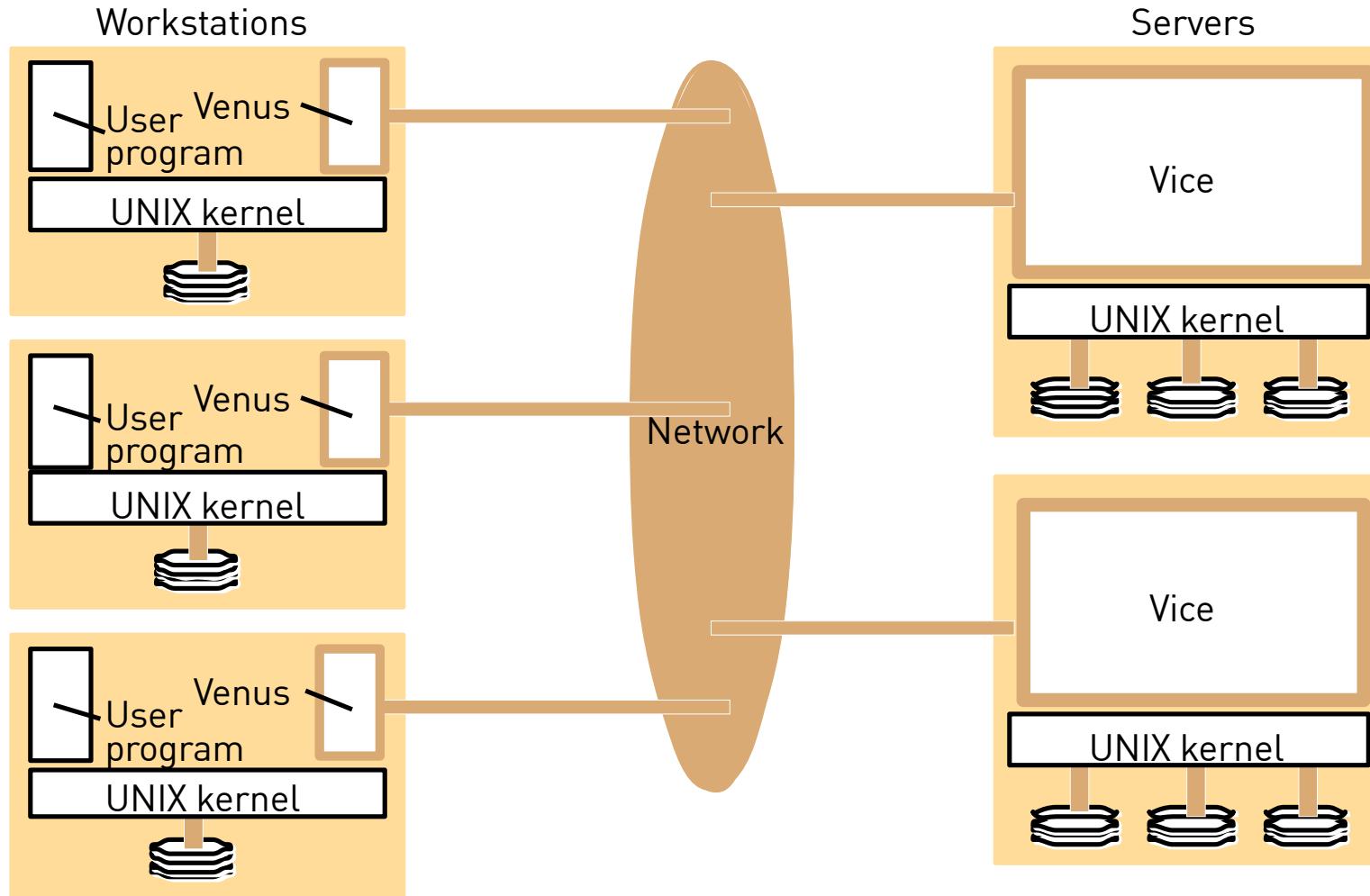
Jede Datei hat einen „Custodian“ Server. Jeder Server hat eine Kopie einer Datenbank, um die Namen auf zu lösen.

Quelle: Coulouris, Dollimore & Kindberg
Distributed Systems:
Concepts and Design
4th Edition, Addison-Wesley Publishers

AFS – Architektur (1)

AFS verwendet zwei User-Space Daemonen: Venus & Vice

das lateinische Wort für „an Stelle“. Vgl „Vice Versa“.

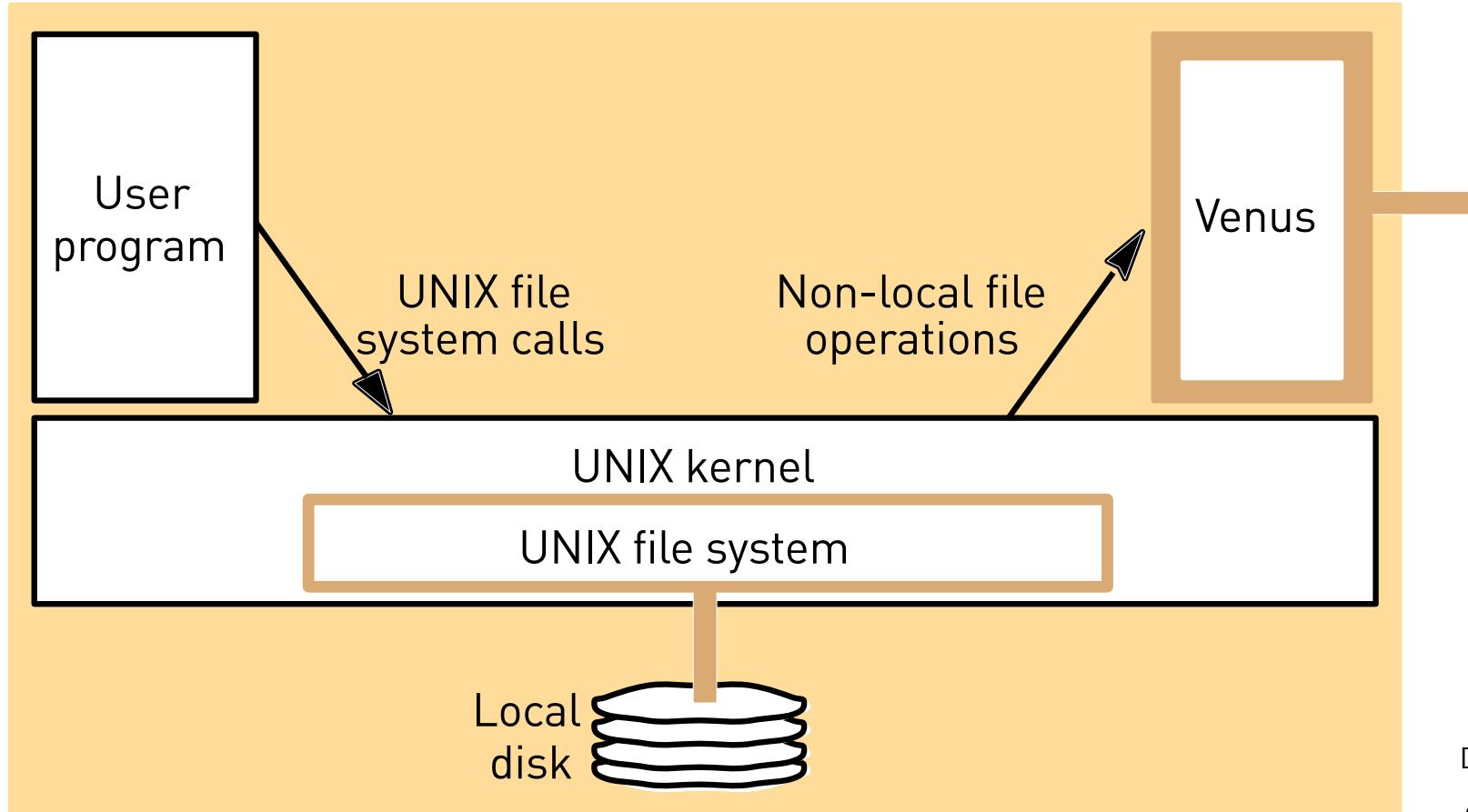


Quelle: Coulouris,
Dollimore & Kindberg
Distributed Systems:
Concepts and Design
4th Edition, Addison-
Wesley Publishers

AFS – Architektur (2)

AFS verwendet zwei User-Space Daemonen: Ein Treiber ist notwendig, um Klienten-Zugriffe abzufangen und weiterzuleiten.

Workstation



Quelle: Coulouris,
Dollimore & Kindberg
Distributed Systems:
Concepts and Design
4th Edition, Addison-
Wesley Publishers

AFS – Venus/Vice Schnittstelle

<i>Fetch(fid) -> attr, data</i>	Returns the attributes (status) and, optionally, the contents of file identified by the <i>fid</i> and records a callback promise on it.
<i>Store(fid, attr, data)</i>	Updates the attributes and (optionally) the contents of a specified file.
<i>Create() -> fid</i>	Creates a new file and records a callback promise on it.
<i>Remove(fid)</i>	Deletes the specified file.
<i>SetLock(fid, mode)</i>	Sets a lock on the specified file or directory. The mode of the lock may be shared or exclusive. Locks that are not removed expire after 30 minutes.
<i>ReleaseLock(fid)</i>	Unlocks the specified file or directory.
<i>RemoveCallback(fid)</i>	Informs server that a Venus process has flushed a file from its cache.
<i>BreakCallback(fid)</i>	This call is made by a Vice server to a Venus process. It cancels the callback promise on the relevant file.

Unvollständig. Verzeichnis- und Verwaltung-Funktionen kommen auch dazu.

Frage: Wie kann man die Role der *Callbacks* zusammenfassen?

Quelle: Coulouris, Dollimore & Kindberg
Distributed Systems:
Concepts and Design
4th Edition, Addison-Wesley Publishers
HOCHSCHULE GARMSTADT
fbi
FACHBEREICH INFORMATIK
SCiences

AFS – Ablauf

User process	UNIX kernel	Venus	Net	Vice
<code>open(FileName, mode)</code>	If <i>FileName</i> refers to a file in shared file space, pass the request to Venus.	Check list of files in local cache. If not present or there is no valid <i>callback promise</i> , send a request for the file to the Vice server that is custodian of the volume containing the file. Place the copy of the file in the local file system, enter its local name in the local cache list and return the local name to UNIX.		Transfer a copy of the file and a <i>callback promise</i> to the workstation. Log the callback promise.
<code>read(FileDescriptor, Buffer, length)</code>	Perform a normal UNIX read operation on the local copy.			
<code>write(FileDescriptor, Buffer, length)</code>	Perform a normal UNIX write operation on the local copy.			
<code>close(FileDescriptor)</code>	Close the local copy and notify Venus that the file has been closed.	If the local copy has been changed, send a copy to the Vice server that is the custodian of the file.		Replace the file contents and send a <i>callback</i> to all other clients holding <i>callback promises</i> on the file.

Quelle: Coulouris, Dollimore & Kindberg, Distributed Systems:
Concepts and Design
4th Edition, Addison-Wesley
Publishers

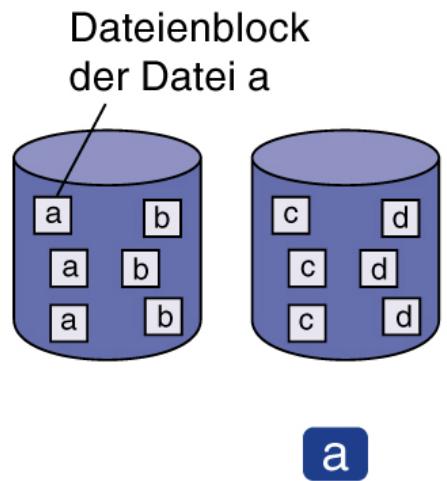
Alternativ-Architektur 1: Datei-Striping

NFS speichert eine Datei auf einem Server.

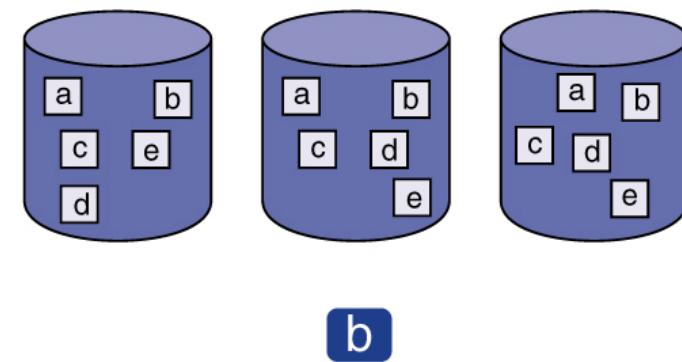
Eine Datei kann auch verteilt werden.

Statische Verteilung = „Datei-Striping“:

Wird in RAIDs verwendet,
und in manchen neuen
verteilten Dateisysteme z.B.
Lustre.



(a) Ohne Striping (ganze Dateien werden verteilt).



(b) Mit Striping (die Blöcke der Dateien werden verteilt).

Alternativ-Architektur 2: GFS - Hintergrund

- ▶ Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung:
"The Google File System", ACM Symposium on Operating
Systems Principles, 2003
 - ▶ Keywords: "Fault tolerance, scalability, data storage, clustered
storage"
- ▶ Annahmen
 - ▶ "First, component failures are the norm rather than the exception.
The file system consists of hundreds or even thousands of storage
machines built from inexpensive commodity parts."
 - ▶ "Second, files are huge by traditional standards. Multi-GB files are
common."
 - ▶ "Third, most files are mutated by appending new data rather than
overwriting existing data. Random writes within a file are practically
non-existent."
 - ▶ "Fourth, co-designing the applications and the file system API
benefits the overall system by increasing our flexibility. For
example, we have relaxed GFS's consistency model"

5-36

h_da

HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES

fbi

FACHBEREICH INFORMATIK



Alternativ-Architektur 2: GFS - Design

- ▶ Weitere Annahmen
 - ▶ Unterstützung von Millionen von Dateien
 - ▶ Dateien ab 100 MB Größe
 - ▶ Optimierung für sehr große Dateien
 - ▶ Unterstützung, aber keine Optimierung für kleine Dateien
 - ▶ Bandbreite wichtiger als Latenz
 - ▶ Lesen: 2 Operationen
 - ▶ Viele Daten werden sequentiell gelesen
 - ▶ Wenige Daten an "zufälligen" Positionen
 - ▶ Schreiben
 - ▶ Gleichzeitiges Anhängen an eine Datei
 - ▶ Selten: Schreiben an zufälligen Positionen
 - ▶ Frage: Welche „Use-Cases“ benötigen so etwas?

5-37



h_da

HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES

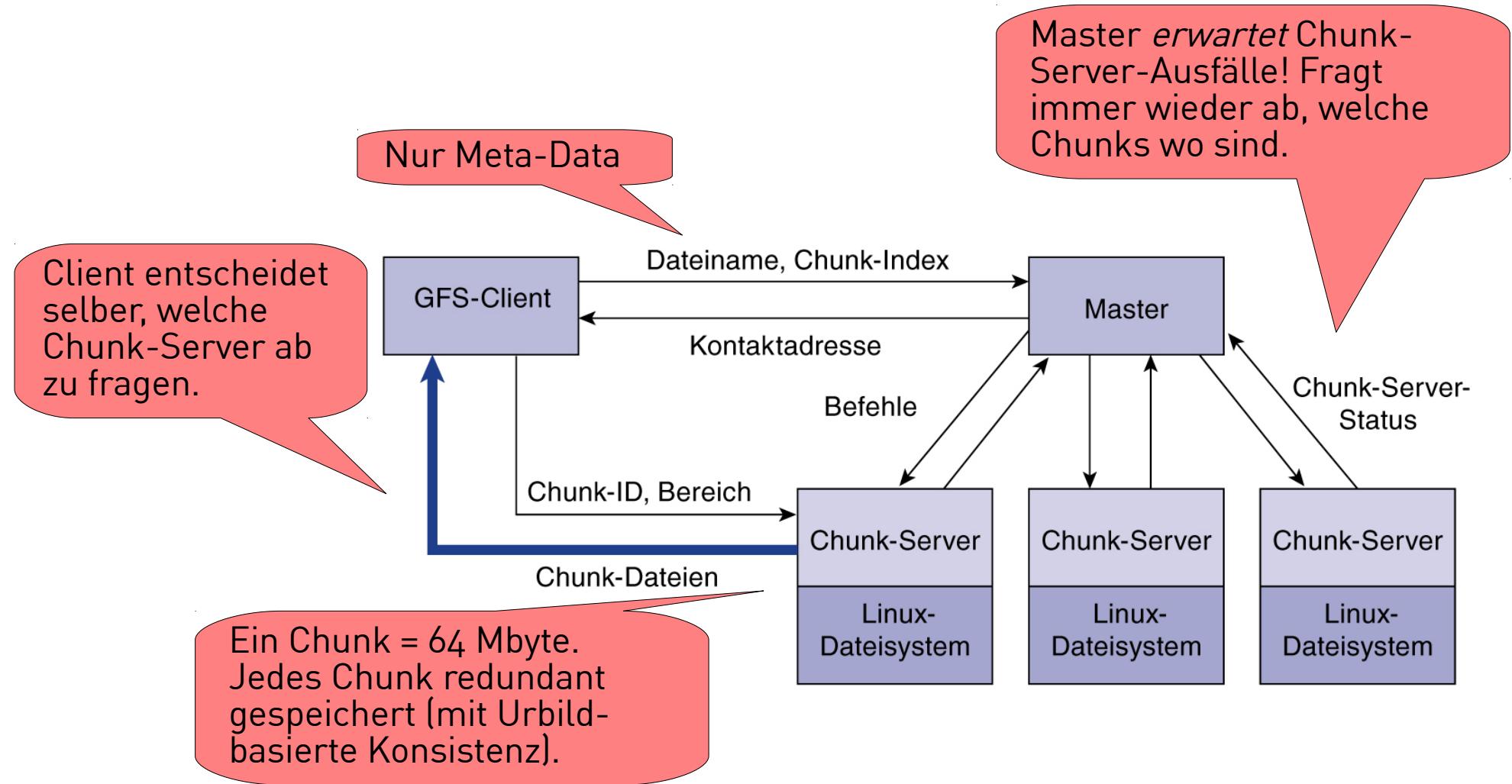
fbi

FACHBEREICH INFORMATIK

Alternativ-Architektur 2: GFS - Design (fortgesetzt)

- ▶ Dateien
 - ▶ Aufteilung in *chunks* fester Größe: 64 MB
 - ▶ 64 Bit *handle* für *chunks*
 - ▶ Replikation: 3 Kopien pro *chunk*
- ▶ Kein Caching
 - ▶ Pragmatischer Grund: Dateien sind zu groß zum performanten Caching
 - ▶ Dateien werden meistens sequentiell gelesen, daher ist Caching sinnlos

Alternativ-Architektur 2: GFS - Überblick

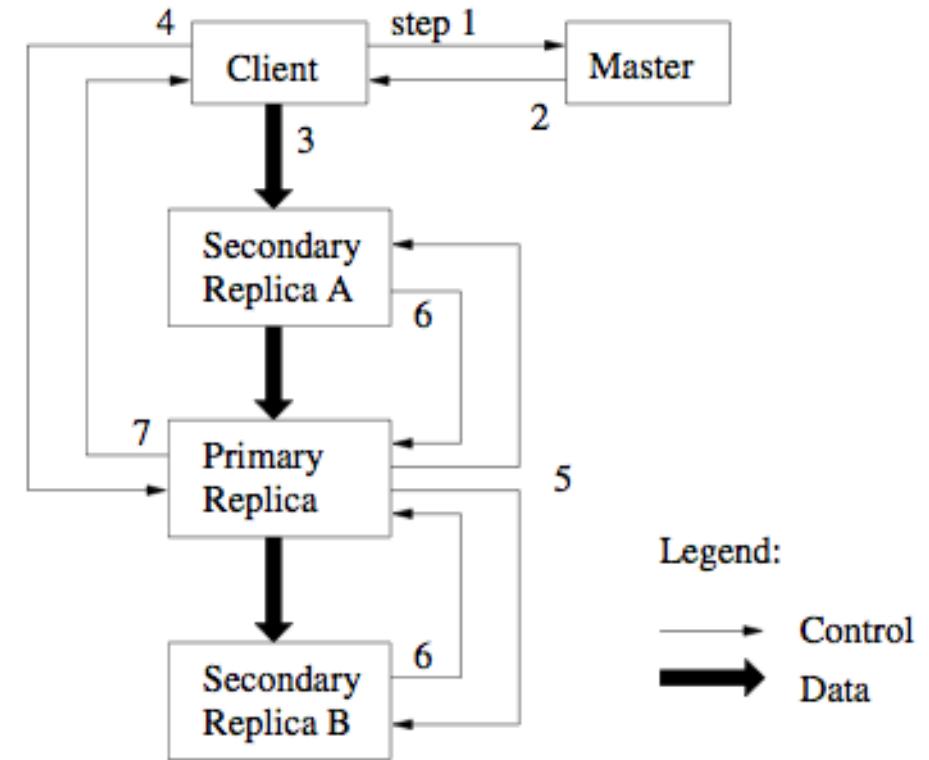


Alternativ-Architektur 2: GFS - Master

- ▶ Master
 - ▶ Hält ungefähre Informationen über alle Replikas
 - ▶ *polling* aller *chunkserver* beim Start
 - ▶ Periodisches ping aller *chunkserver*: *HeartBeat*
 - ▶ Liefert ebenfalls Informationen über Replikas
 - ▶ Master bestimmt *primary replica* als Server, der alle Schreiboperationen koordiniert
 - ▶ Frage: Warum wird nicht versucht, im Master eine konsistente Sicht zu halten?
 - ▶ Antwort: Master kennt genaue Situation der *chunkserver* nicht

Alternativ-Architektur 2: GFS - Master

- ▶ Variante der master/worker Architektur
- ▶ Master bestimmt *primary replica* (Schritt 2)
- ▶ Client schickt erst die Daten (Schritt 3) an alle Replikas und sendet dann die Schreiboperation an *primary replica* (Schritt 4)
- ▶ *Primary replica* koordiniert dann Schreiboperationen auf allen *replicas* (2 Phasen Commit!)



Alternativ-Architektur 2: GFS - Operation Log

- ▶ Enthält alle Operationen der Vergangenheit
- ▶ Geschrieben vom Master, auf mehrere externe Rechner gesichert
- ▶ Nach Crash führt Master alle enthaltenen und nicht beendeten Operation noch einmal aus
 - ▶ Regelmäßiges *checkpointing* zur Sicherung des Status
 - ▶ Nur letzter *checkpoint* wird zur Wiederherstellung gebraucht

5-42

h_da

HOCHSCHULE DARMSTADT

UNIVERSITY OF APPLIED SCIENCES

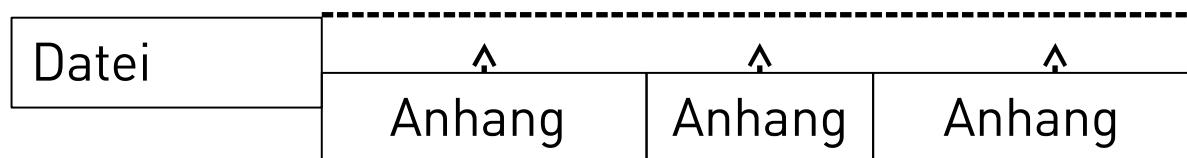
fbi

FACHBEREICH INFORMATIK



Alternativ-Architektur 2: GFS - Konsistenz

- ▶ Namen von Dateien:
 - ▶ werden in Transaktionen (atomar) vom Master geändert
- ▶ Daten:
 - ▶ Inkonsistenzen treten nur bei Fehlern auf, z.B. beim Ausfall einer *secondary replica* oder Nachrichtenverlust
 - ▶ Atomares *record append*: Client gibt keine Position an, nur die Daten die angehängt werden sollen
 - ▶ Daten werden mit ***at-least-once*** Semantik geschrieben
 - ▶ *Primary replica* wählt die Position aus, d.h. es können mehrere Operationen gleichzeitig durchgeführt werden
 - ▶ Ein Client erhält keine Garantie für die Position



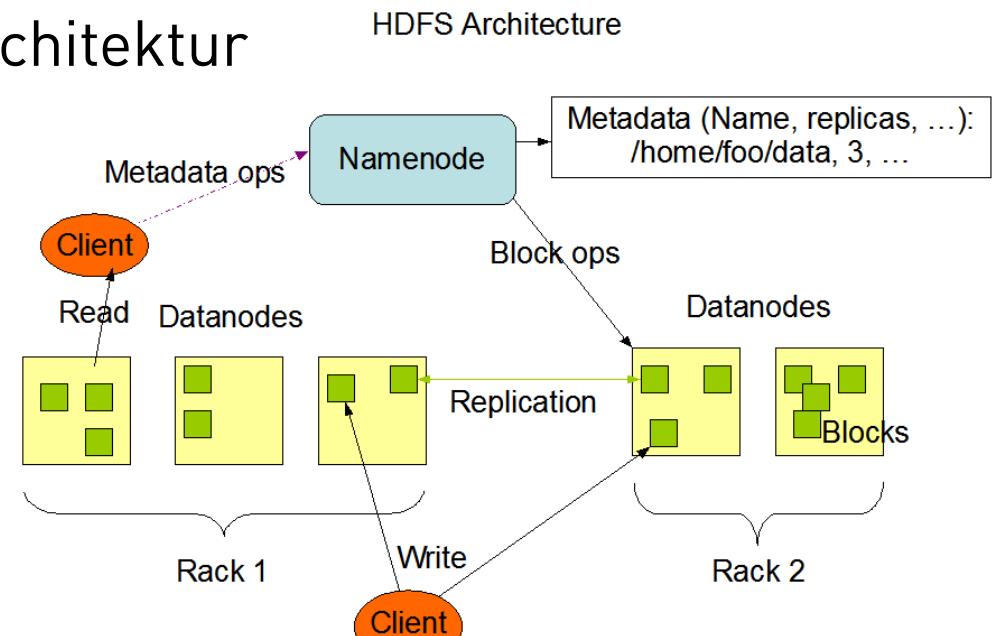
Parallel in
irgendeiner
Reihenfolge

Alternativ-Architektur 2: GFS - Konsistenz

- ▶ *chunk replication* sorgt für Replikation der Daten
- ▶ Der Master ist ebenfalls repliziert
 - ▶ *operation log* und *checkpoints* liegen auf mehreren Servern
 - ▶ *shadow masters* bieten Lesezugriff falls Master ausgefallen: durch fortgesetztes Lesen des *operation logs*

Alternativ-Architektur 3: Hadoop

- ▶ Teil von Apache hadoop
- ▶ Ähnlicher (nicht gleicher) Aufbau wie GFS
- ▶ Optimiert für map-reduce (klar!)
- ▶ Ebenfalls Master/worker Architektur
- ▶ Blocks statt *chunks*
- ▶ Eigene RPC
- Implementierung auf Java und TCP/IP
- ▶ *rack awareness*: beeinflusst Platzierung der Replikate



Quelle: http://hadoop.apache.org/docs/stable1/hdfs_design.html

Struktur von Kapitel V – Beispiele bzw. Dienste

- I Einleitung
- II Grundlegende Kommunikationsdienste
- III Middleware
- IV Architekturen & Algorithmen
 - A Synchronisierung
 - B Konsistenz und Replication
 - C Fehlertoleranz
- V Beispiele bzw. Dienste
 - A Verteilte Dateisysteme
 - B Namensdienste
- VI Sicherheit & Sicherheitsdienste
- VII Zusammenfassung

1. Anforderungsanalyse, Schnittstelle & Architektur
2. Kommunikation
3. Synchronisierung, Konsistenz & Replikation
4. Sicherheit
5. Alternativen
6. Zusammenfassung



Anforderungsanalyse – Review/Übung

Welches System ist besser
bzgl. jede Kriterium?

- **Zugriffstransparenz:**
Der Zugriff auf entfernte Dateien erfolgt für Klienten mit den gleichen Operationen, wie der Zugriff auf lokale Dateien.
- **Ortstransparenz:**
Es spielt keine Rolle, an welchem physikalischen Ort eine Datei gespeichert ist.
- **Nebenläufigkeitstransparenz:**
Greifen mehrere Klienten gleichzeitig auf eine Datei zu, soll kein Klient von den Zugriffen der anderen etwas mitbekommen.
Allerdings: nebenläufige *schreibende* Zugriffe sind problematisch.
- **Fehlertransparenz:**
Für Klienten, wie für Server, soll ein Fehlverhalten des jeweils anderen nicht zu Inkonsistenzen und Verklemmungen führen.
- **Lasttransparenz:**
Die Anfragelast auf Serverseite soll sich nicht auf dessen Antwortzeiten auswirken.
- **Hardware- und Betriebssystem-Transparenz:**
Das verteilte Dateisystem soll die *Heterogenität* unterschiedlicher Hardware und Betriebssysteme verbergen.

NFS	GFS	AFS



Anforderungsanalyse – Review/Übung (2)

Welches System ist besser
bzgl. jede Kriterium?

- **Replikationstransparenz:**
Der Klient darf immer nur eine logische Datei sehen, egal wie häufig und wo diese repliziert vorgehalten wird.
- **Migrationstransparenz:**
Verändert eine Datei ihren Speicherungsort, dann ist dies für den Klienten nicht sichtbar.
- **Partitionierungs- bzw. Erreichbarkeits-transparenz:**
Das Dateisystem kann partitioniert werden, d.h. bestimmte Dateiserver sind nicht mehr (direkt-) zugreifbar für bestimmte Klienten. Trotzdem stehen für alle Klienten noch alle benötigten Dateien zur Verfügung. Eine Vereinigung bisher getrennter Partitionen erzeugt dabei möglichst keine Inkonsistenzen.

Besonders für die Anbindung **mobiler** Rechner interessant.

NFS	GFS	AFS

Struktur von Kapitel V – Beispiele bzw. Dienste

- I Einleitung
- II Grundlegende Kommunikationsdienste
- III Middleware
- IV Architekturen & Algorithmen
 - A Synchronisierung
 - B Konsistenz und Replication
 - C Fehlertoleranz
- V Beispiele bzw. Dienste
 - A Verteilte Dateisysteme
 - B Namensdienste
- VI Sicherheit & Sicherheitsdienste
- VII Zusammenfassung

- 1. Definitionen
- 2. DNS
- 3. Directory Services z.B. LDAP

Definitionen

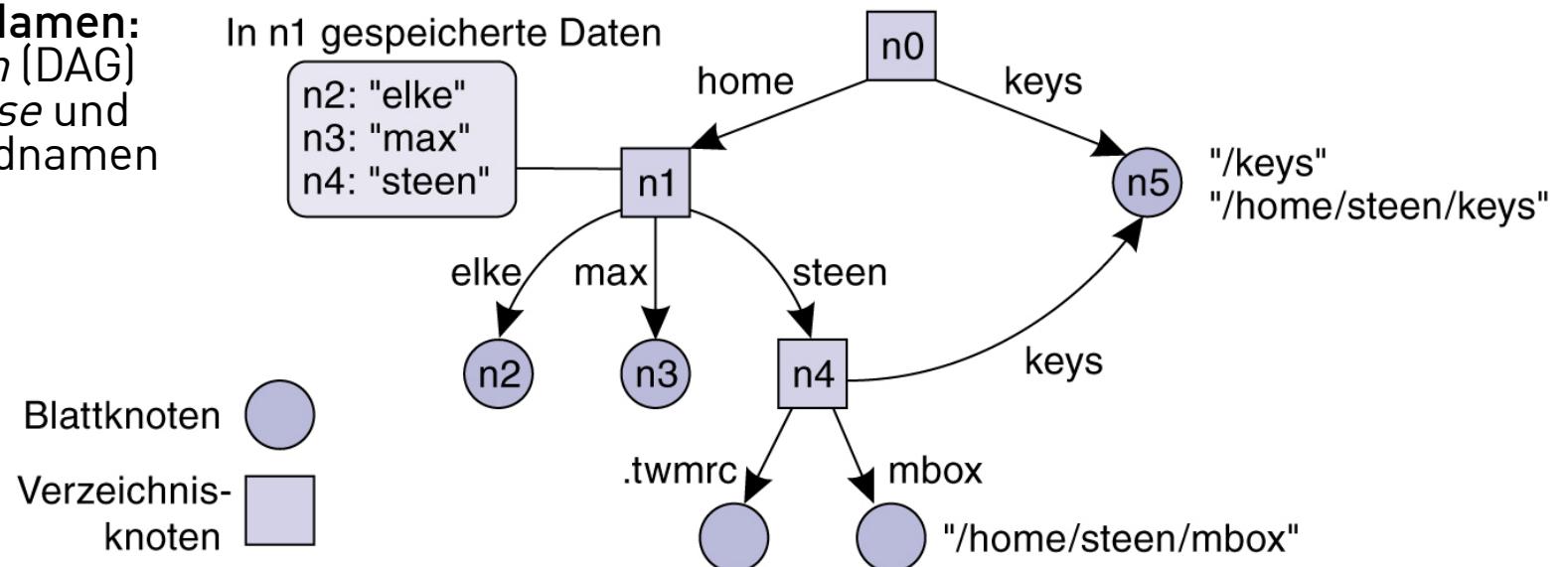
Namen beschreiben Entitäten.

Sonderfälle:

- **Adressen:**
Namen, die den Zugriffspunkt einer Entität bezeichnen.
- **Bezeichner (Identifier):**
Namen, die höchstens eine Entität bezeichnen.
- **Hierarchische Namen:**
bilden ein *Baum* (DAG) aus *Verzeichnisse* und *Blätter*, vgl. Pfadnamen im Dateisystem.

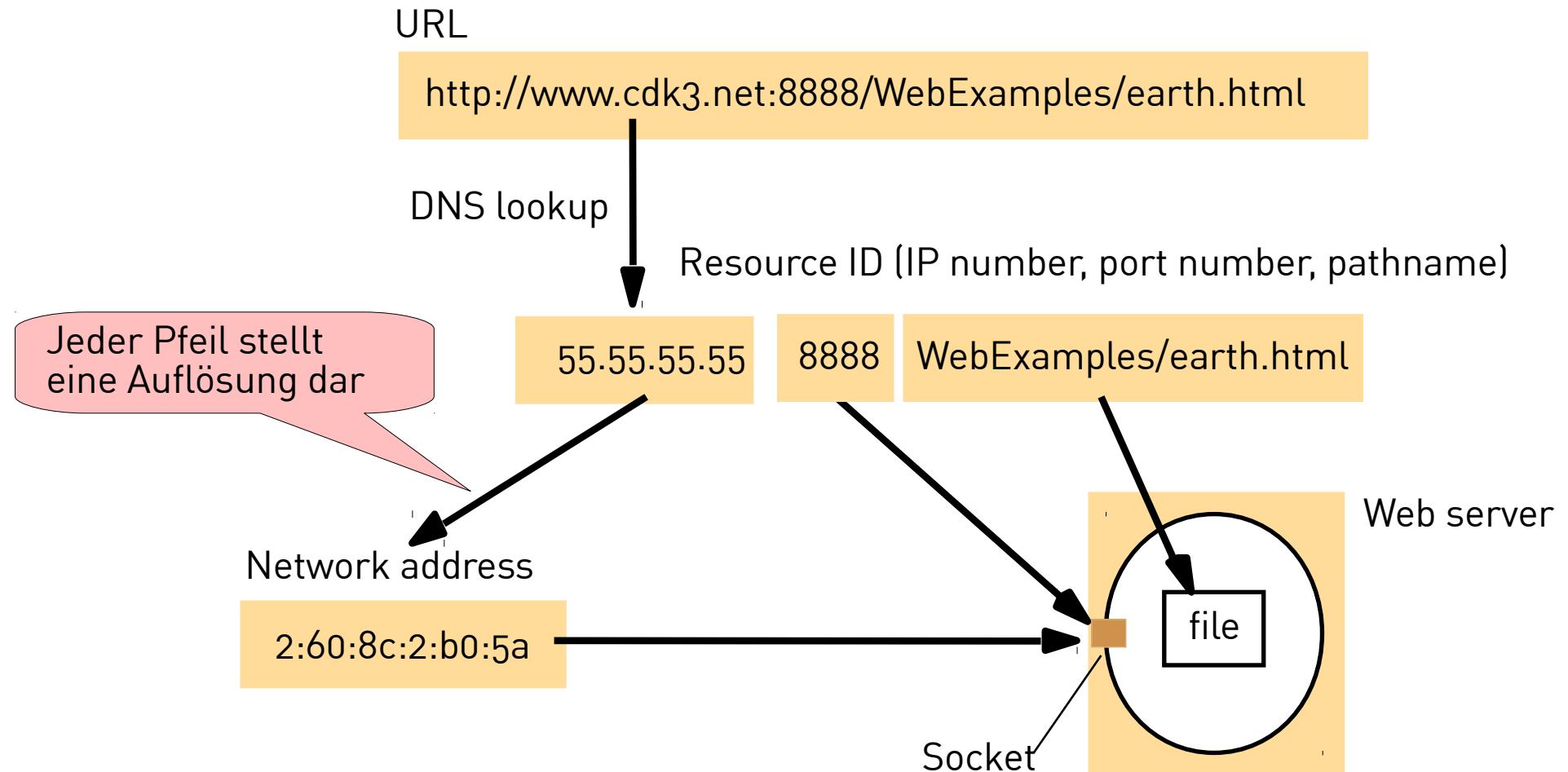
„Entitäten“ sind (unter anderem):

- **Komponenten**
z.B. Rechner,
- **Dateien**
S.O.,
- **Funktionen, Objekte bzw. Dienste**
vgl. Kapitel II, III
- **Benutzer** (vgl. Kapitel VI, u.a.).



Namensauflösung

Resolution: Nachschlagen eines Namens.
Übersetzen in eine Adresse.



Quelle: Coulouris, Dollimore and Kindberg
Distributed Systems: Concepts and Design
4th Edition, Addison-Wesley Publishers

Anforderungsanalyse (1)

Übung: Was sind die Anforderungen an ein *verteilten* Namens-Resolution-Dienst?

**Ein konkretes Beispiel –
DEC's Global Name Service (1986)**

- **Skalierbarkeit:**
Unbegrenzt viele Namen,
unbegrenzt viele Verwaltungs-Einheiten.
- **Robustheit:** Es wird Änderungen in der Namen und in dem Dienst erwartet.
- **Verfügbarkeit:** Klienten sind auf den Dienst abhängig!
- **Fehlertoleranz** bzw. Fehler-Isolation – lokale Fehler dürfen das ganze System nicht gefährden.
- **Toleranz von Misstrauen:** Nicht alle Klienten werden alle Komponenten vertrauen (können).

Vgl. Coulouris, Dollimore and Kindberg, §9.2.

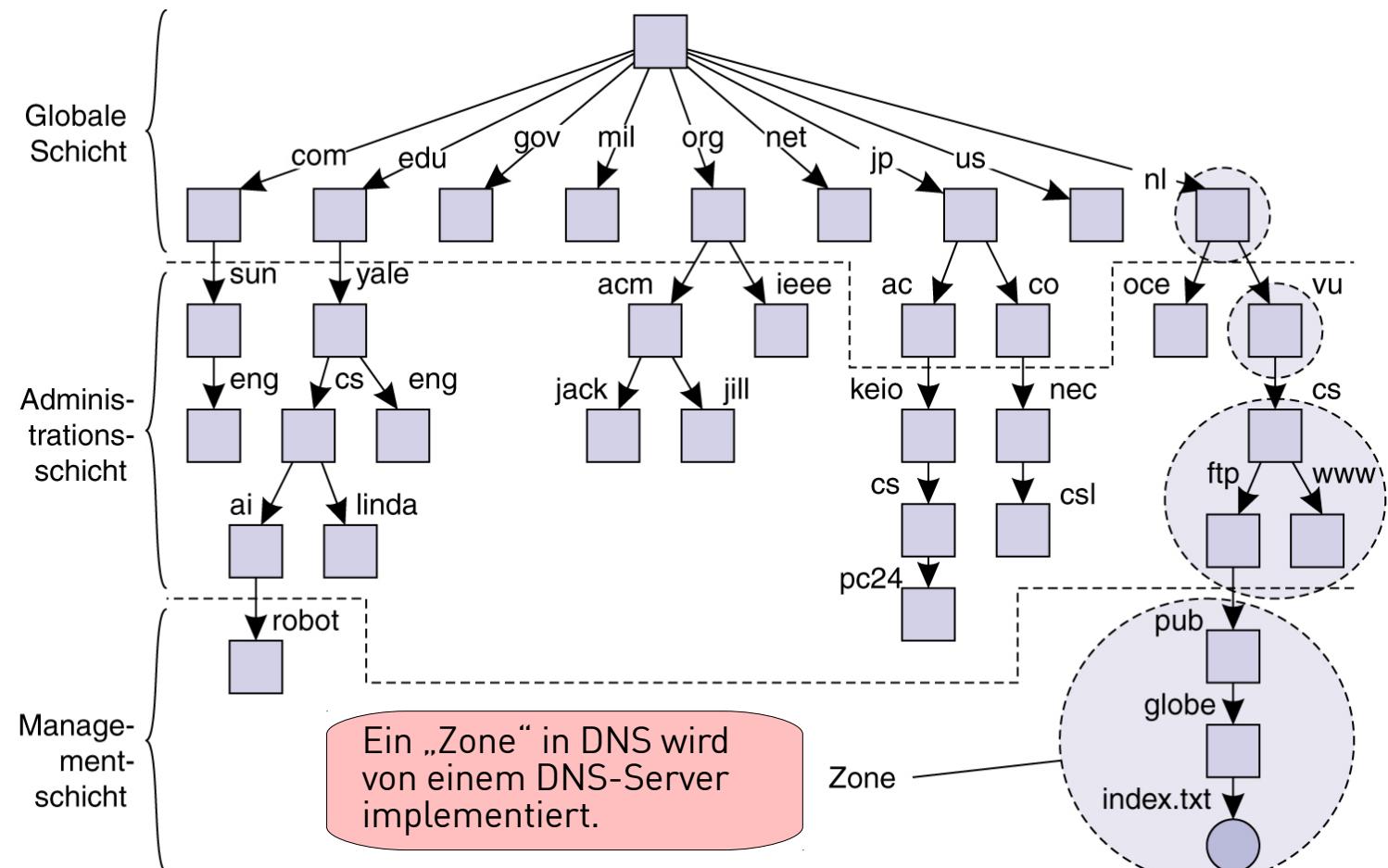
Anforderungsanalyse (2)

Verteilung des Namensraumes in 3 Schichten

- Globale Schicht – Organizationübergreifend
- Administrationsschicht – einer Organisation
- Managementschicht – oft geändert, eng gekoppelt.

Gelten alle Anforderungen in alle Schichten? Gleichermassen?

Beispiel:
Partitionierung
des WWW-
(DNS+...)
Namensraumes:



Anforderungsanalyse (3)

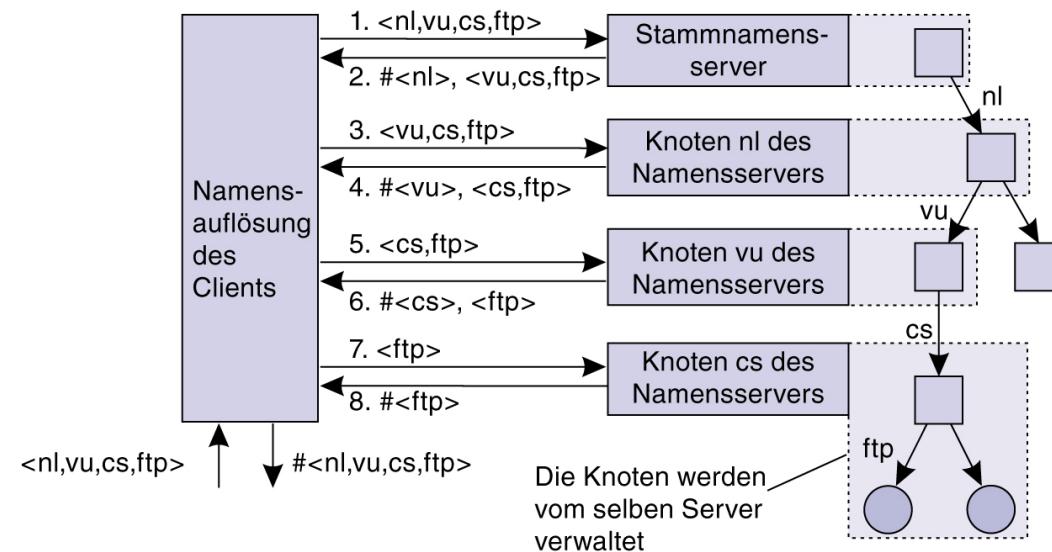
Die 3 Schichten sind *qualitativ unterschiedlich* und haben also eigene Anforderungen.

Aspekt	Globale Schicht	Administrations-schicht	Management-schicht
Geografische Ausdehnung des Netzwerkes	Weltweit	Organisation	Abteilung
Knotengesamtzahl	Wenige	Viele	Zahllose
Reaktionsfähigkeit auf Nachschlageanforderungen	Sekunden	Millisekunden	Sofort
Verbreitung von Aktualisierungen	Verzögert	Sofort	Sofort
Anzahl Replikate	Viele	Keine oder wenige	Keine
Caching durch Clients	Ja	Ja	Manchmal

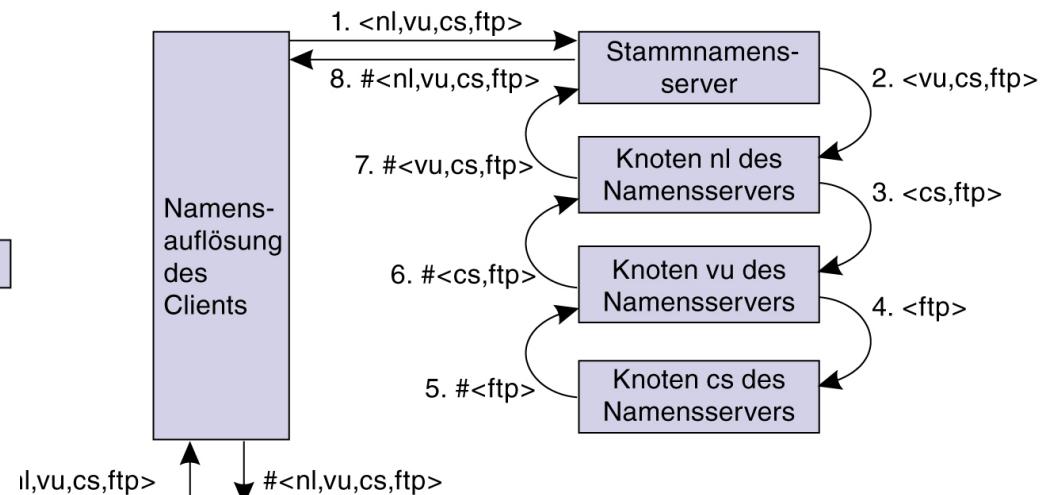
Anforderungsanalyse (4)

Strategie-Entscheidung: Iterative oder rekursive Auflösung?

Iterative



Rekursiv

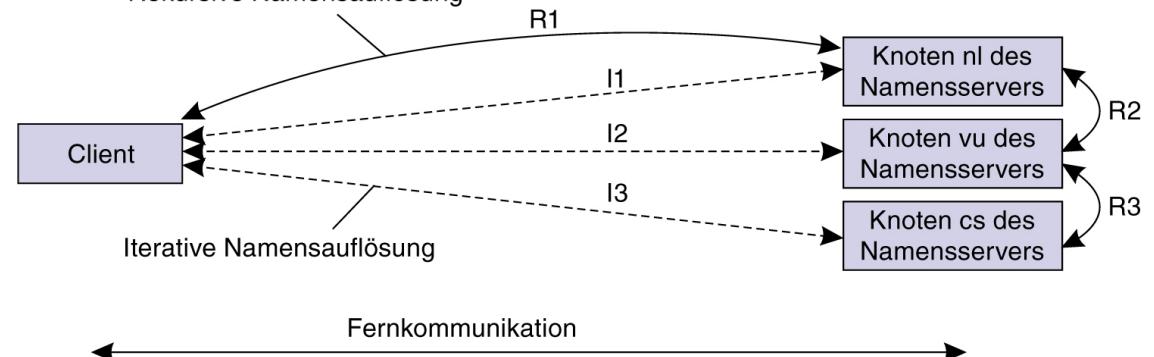


Analyse? Vorteile? Nachteile?

Vorteil von iterative Auflösung:
Zustandslose Server.

Vorteile von rekursiver Auflösung:
- Caching
- Netzwerknähe-Optimierung

Rekursive Namensauflösung



Struktur von Kapitel V – Beispiele bzw. Dienste

- I Einleitung
- II Grundlegende Kommunikationsdienste
- III Middleware
- IV Architekturen & Algorithmen
 - A Synchronisierung
 - B Konsistenz und Replication
 - C Fehlertoleranz
- V Beispiele bzw. Dienste
 - A Verteilte Dateisysteme
 - B Namensdienste
- VI Sicherheit & Sicherheitsdienste
- VII Zusammenfassung

- 1. Definitionen
- 2. DNS
- 3. Directory Services z.B. LDAP

DNS - Zonen

DNS-Zonen beinhalten folgende Daten:

- **Attributen von Namen** für ein *Domain*, minus *Sub-domains*. Z.B. für *.h-da.de, nicht für *.fbi.h-da.de
- Namen und Adressen von mindestens zwei **autoritative Name-Servers** für die Zone
- Namen und Adressen von **autoritative Name-Servers** für die Sub-Domains
- Verwaltungs-Daten...

Vgl. Coulouris, Dollimore and Kindberg, §9.2.3

DNS - Attribute

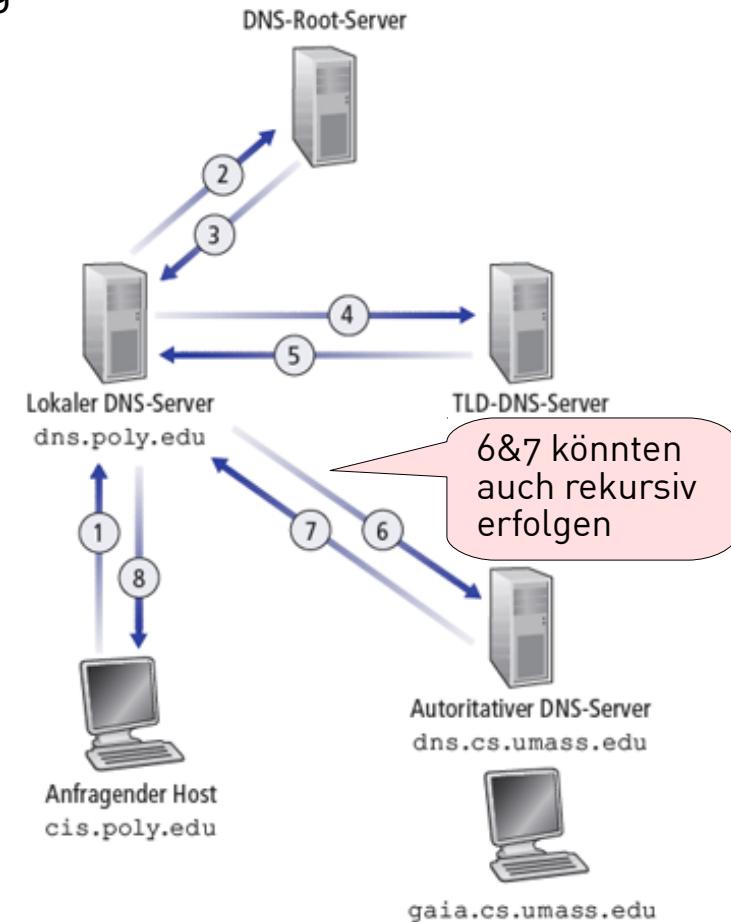
Fallbeispiel – DNS

RR-Typ	Zugehörige Einheit	Beschreibung	
SOA	Zone	Enthält Informationen zur dargestellten Zone.	SOA = State of Authority
A	Host	Enthält eine IP-Adresse des Hosts, die dieser Knoten darstellt.	Es kann mehrere A-Records geben
MX	Domäne	Verweist auf einen Mailserver, der an diesen Knoten gerichtete E-Mail verarbeitet.	Also kann Email an eine Domain, statt ein Server adressiert werden.
SRV	Domäne	Verweist auf einen Server, der einen besonderen Dienst bewältigt.	Ähnliche Fähigkeit für andere Dienste
NS	Zone	Verweist auf einen Namensserver, der die dargestellte Zone implementiert.	
CNAME	Knoten	Symbolischer Link auf den Primärnamen des dargestellten Knotens.	Ein Server kann Aliases haben, aber nur eine <i>canonical Name</i>
PTR	Host	Enthält den kanonischen Namen eines Hosts.	
HINFO	Host	Enthält Informationen zu dem Host, den dieser Knoten darstellt.	Für Reverse-Lookup.
TXT	Beliebig	Enthält beliebige Informationen zu einer Entität, die für nützlich erachtet werden.	Selten verwendet...

DNS – Protokoll (1)

In DNS gibt es Name-Server und Resolver (Klienten)

- DNS erlaubt **sowohl iterative als auch rekursive** Auflösung
- Der Resolver (Klient) gibt an, ob er iterative oder rekursive Auflösung haben möchte...
- Der Server kann aber rekursive Auflösung verweigern.
- **Fehlertoleranz:** Der Resolver verwendet (meistens) UDP
 - muss also u.U. Anfrage wiederholen.
 - Sollte mindestens zwei Server kennen, falls einer nicht erreichbar ist.
 - Der Resolver kann mehrere Anfragen zusammenpacken; der Server kann mehrere Antworten zusammenpacken.
- **Replikation/Konsistenz:** Jede gelieferte Information kommt mit ein *Time-to-Live*, d.h. eine Cache-Zeit (Gültigkeitsdauer).



Vgl. Coulouris, Dollimore and Kindberg, §9.2.3

Quelle: Kurose & Ross, Kapitel 2

h_da

HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES

fbi

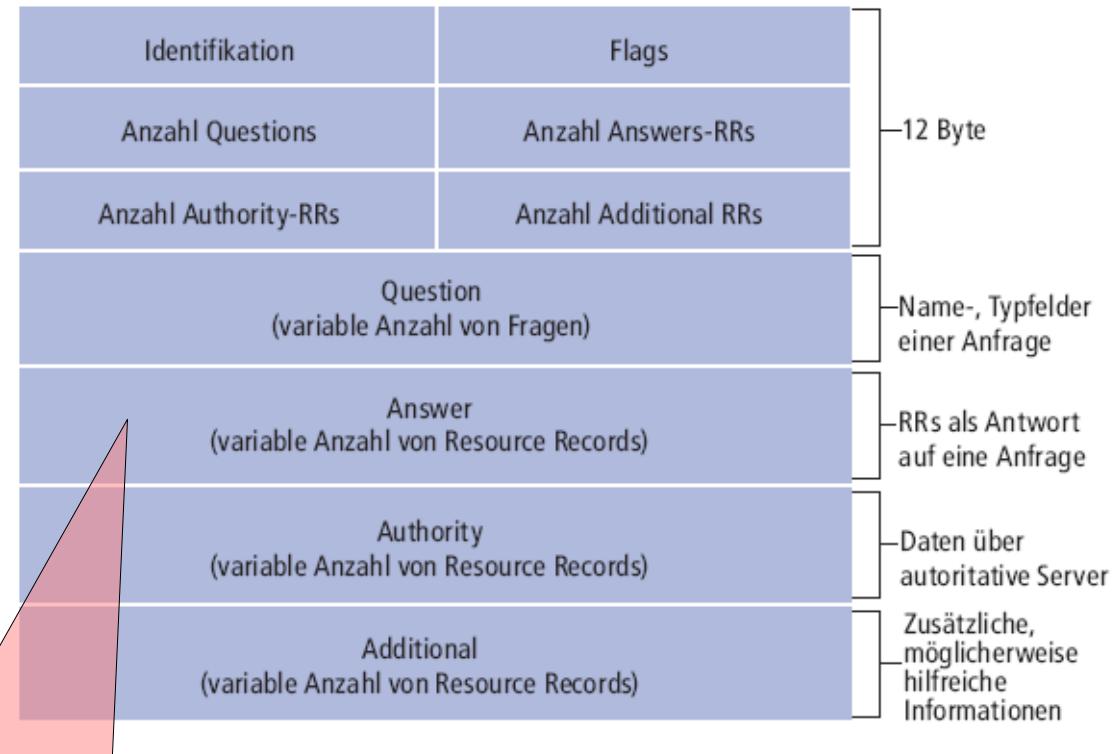
FACHBEREICH INFORMATIK

DNS – Protokoll (2)

Query- und *Reply*-Nachrichten, beide mit demselben Nachrichtenformat

Header-Felder

- **identification:** 16-Bit-ID, wird für die Query-Nachricht vergeben, die Reply-Nachricht verwendet dieselbe ID
- **flags:**
 - query/reply
 - recursion desired
 - recursion available
 - reply is authoritative



Frage: Warum gibt es *sowohl* eine Frage als *auch* eine Antwort?

Quelle: Kurose & Ross, Kapitel 2 bzw. Companion Website

Struktur von Kapitel V – Beispiele bzw. Dienste

- I Einleitung
- II Grundlegende Kommunikationsdienste
- III Middleware
- IV Architekturen & Algorithmen
 - A Synchronisierung
 - B Konsistenz und Replication
 - C Fehlertoleranz
- V Beispiele bzw. Dienste
 - A Verteilte Dateisysteme
 - B Namensdienste
- VI Sicherheit & Sicherheitsdienste
- VII Zusammenfassung

- 1. Definitionen
- 2. DNS
- 3. Directory Services z.B. LDAP

Directory Services

Ein Verzeichnisdienst beschreibt Entitäten an Hand deren **Eigenschaften (Attributes)**.

- Zu jede **Eigenschaft** wird einen **Wert** definiert.
- Der Wert kann einzeln oder vielfach sein (singular/multiple).
- Die Attributen sind in einer Reihenfolge; Damit lassen sich Verzeichnisse darstellen.
- Wegweisend war X.500 von der OSI (vgl. Corba, s.o.).
- LDAP (Lightweight Directory Access Protocol) ist eine Vereinvachung von X.500
- LDAP-Implementierung sind u.a.
 - Active Directory von Microsoft
 - Open Directory von Apple
 - OpenLDAP – Open Source

Ein LDAP Eintrag

Directory Server Eintrag

Distinguished Name

Attribute: OU („Organizational Unit“)
Value: webdesign

```
dn: uid=juser,ou=People, ou=webdesign, c=de, o=acme
objectclass: top
objectclass: person
objectclass: organizationalPerson
objectclass: inetorgPerson
cn: Joe User
givenname: Joe
sn: User
cn: User Joe
telephonenumber: +49 123 12345
mail: joe.user@acme.de
userpassword: {SHA}fdowskjdap123hdknfc
```

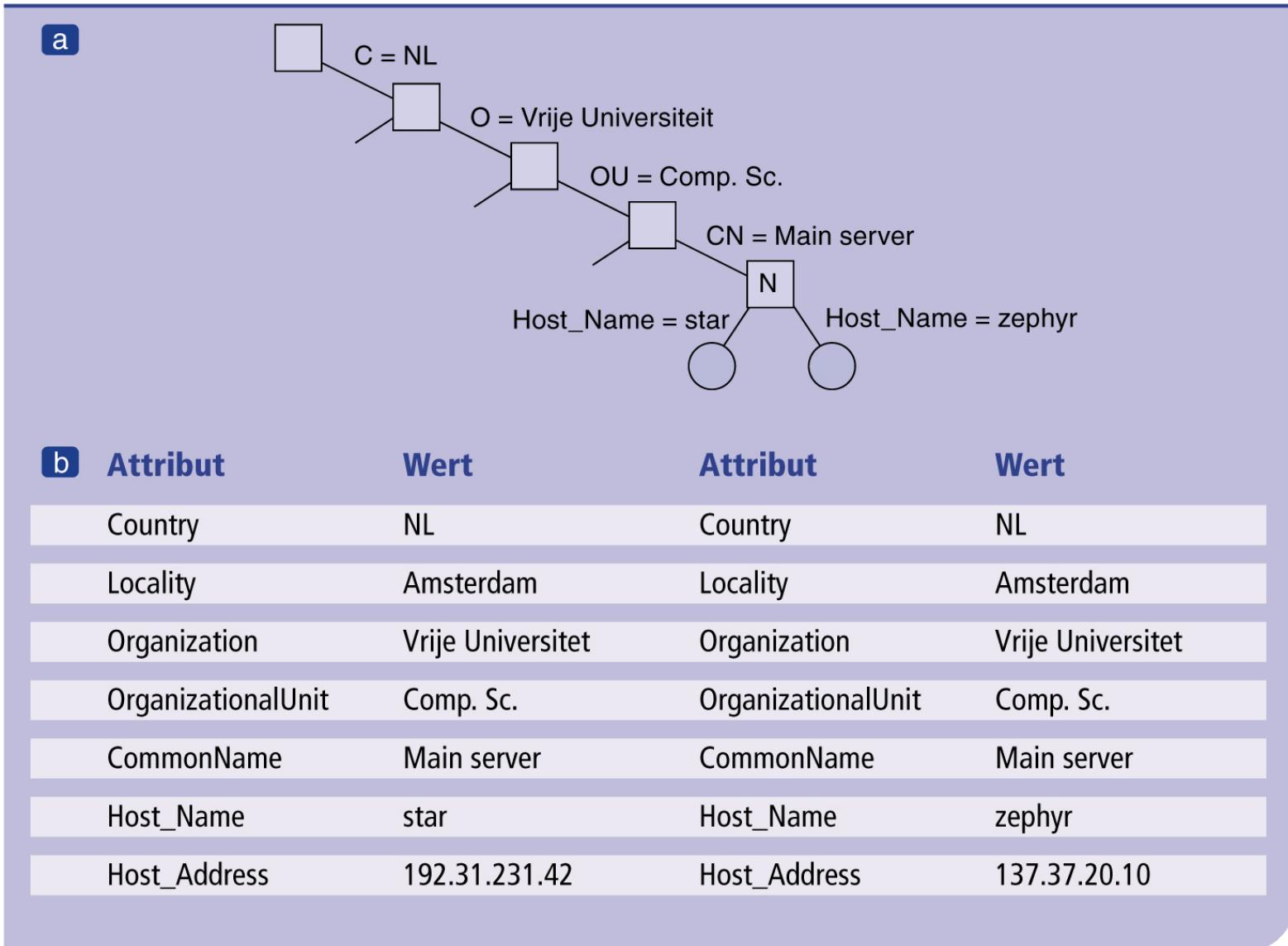
User-Nutzdaten

Schema-Definition des
Eintrags

Quelle: <http://de.wikipedia.org/w/index.php?oldid=39158084>

Ein Verzeichnisinformationsbaum

Bzw. ein DIT = Directory Information Tree



n_da

HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES

fbi

FACHBEREICH INFORMATIK



Struktur von Kapitel V – Beispiele bzw. Dienste

- I Einleitung
- II Grundlegende Kommunikationsdienste
- III Middleware
- IV Architekturen & Algorithmen
 - A Synchronisierung
 - B Konsistenz und Replication
 - C Fehlertoleranz
- V Beispiele bzw. Dienste
 - A Verteilte Dateisysteme
 - B Namensdienste
 - C Zusammenfassung
- VI Sicherheit & Sicherheitsdienste

Beispiele & Dienste - Zusammenfassung

Verteilte Dateisysteme

A) Anforderungsanalyse, Schnittstelle & Architektur

- Transparenz?
- Zugriffs-Operationen oder ganze Dateien?
- Zustands-behaftete oder Zustandslose Server?

B) Kommunikation

- Iterativ oder Rekursiv oder ganze Dateien?

C) Synchronisierung, Konsistenz & Replikation

- Welche Semantik? Welche „Benutzer“ (-Arten)?

D) Sicherheit

- Minimal oder Zukunftsfähig?

E) Alternativen

- AFS
- GFS bzw. Hadoop

Namensdienste

A) Definitionen

- Anforderungen
- Globale, Administrations- und Managementschicht
- Iterativ oder Rekursiv – nochmal!

B) DNS

C) Directory Services wie zB LDAP