TU Braunschweig
Institut für Betriebssysteme
und Rechnerverbund

Verteilte Systeme

Prof. Dr. Stefan Fischer

Kapitel 7: Verteilte Verzeichnis- und Dateidienste

Überblick

- Namens- und Verzeichnisdienste
  - Motivation
  - Namen/Attribute
  - Generelle Anforderungen an Verzeichnisdienste
  - Generelle Funktionalität eines Verzeichnisdienstes
  - Beispiele
- Verteilte Dateidienste
  - Überblick
  - Beispiel: NFS

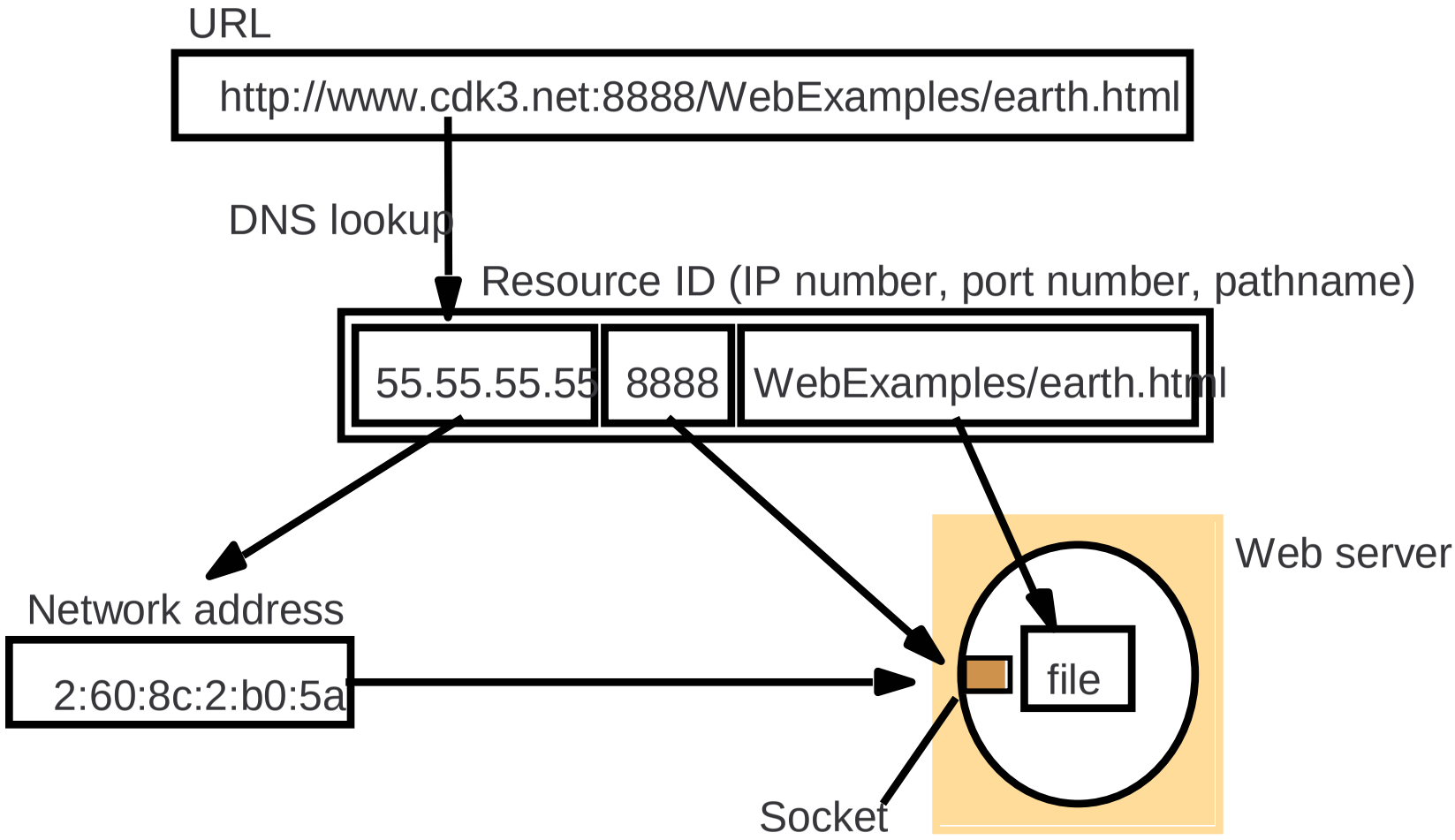
Motivation

- Verzeichnisdienste (*directory services*) sind fundamentale Werkzeuge in verteilten Systemen.
- Ihre Aufgabe besteht in der Zuordnung von Objektnamen zu einer Menge von Eigenschaften und Werten.
- Objekte können bei Verzeichnisdiensten registriert, ihre Namen und Eigenschaften können anschließend abgefragt werden.
- Ein spezieller Verzeichnisdienst ist der *Name Service*, der den Namen eines Objekts seiner Adresse zuordnet.

Motivation (Forts.)

- Namensdienste bieten damit die grundlegendste Funktionalität.
- Menschen bevorzugen „sinnvolle“ Namen für Ressourcen statt „Adressen“
  - in Anwendungen verwendet man www.cs.tu-bs.de.
- Computer bevorzugen Zahlen
  - Netzwerkprotokolle wie IP verwenden numerische Adressen: 134.169.34.18.
- Aufgabe eines Name Service
  - Übersetze Namen in Adressen und lasse Anwendungen so Netzdienste nutzen: www.cs.tu-bs.de $\leftrightarrow$ 134.169.34.18.

Beispiel: URL-Abbildung



Generelle Verzeichnisdienste

- Verzeichnisdienste werden generell genutzt, um einem Namen Attribut-/Wert-Paare zuzuordnen.
- Beispiele:
  - Telefonnummern- und Adressenverzeichnis in einer Firma
  - Namen und Verfügbarkeit von Ressourcen (Drucker, Platten, Modems)
  - Verfügbare Anwendungsdienste
- Dabei sind nicht nur Abfragen von Name zu Attribut, sondern auch umgekehrt möglich („Yellow Pages“).
- Die Verknüpfung zwischen einem Namen und seinen Attributen wird als *Bindung* („binding“) bezeichnet.

Anforderungen an Verzeichnisdienste

- Die Möglichkeit, eine im wesentlichen unbeschränkte Zahl von Namen zu verwalten
- Lange Lebensdauer des Dienstes (Investitionssicherung)
- Hohe Verfügbarkeit
- Isolierung von Fehlern (um nicht den ganzen Service zu beeinflussen)
- Tolerierung von Misstrauen (in einem großen System sind nicht alle Komponenten gleich vertrauenswürdig)

Der Namensraum (Name Space)

- Ein Namensraum ist die Menge aller gültigen Namen, die von einem bestimmten Namensdienst erkannt werden.
- “gültig” bedeutet: Der Dienst versucht, den Namen aufzulösen. Der Name muss aber nicht gebunden sein.
- Gültige Namen werden durch Syntaxregeln festgelegt.
- Namensräume können flach oder hierarchisch organisiert sein.

Hierarchische Namensräume

- Zeichnen sich durch eine interne Struktur aus
- Beispiel: DNS-Namensraum
- Vorteile
  - Jeder Namensteil hat seine Gültigkeit nur in einem bestimmten Kontext, so dass Namen in verschiedenen Kontexten doppelt verwendet werden können.
  - Dadurch wird eine dezentrale Verwaltung des Namensraums möglich.

Definition des DNS-Namensraums

- Ein DNS Domain-Name besteht aus einem oder mehreren Strings (*name components*, die durch „.“ getrennt sind.
- Es gibt kein Trennzeichen am Anfang und am Ende eines Domain-Names.
- Die Name Components sind druckbare, nicht-leer Zeichenketten, die kein „.“ enthalten.

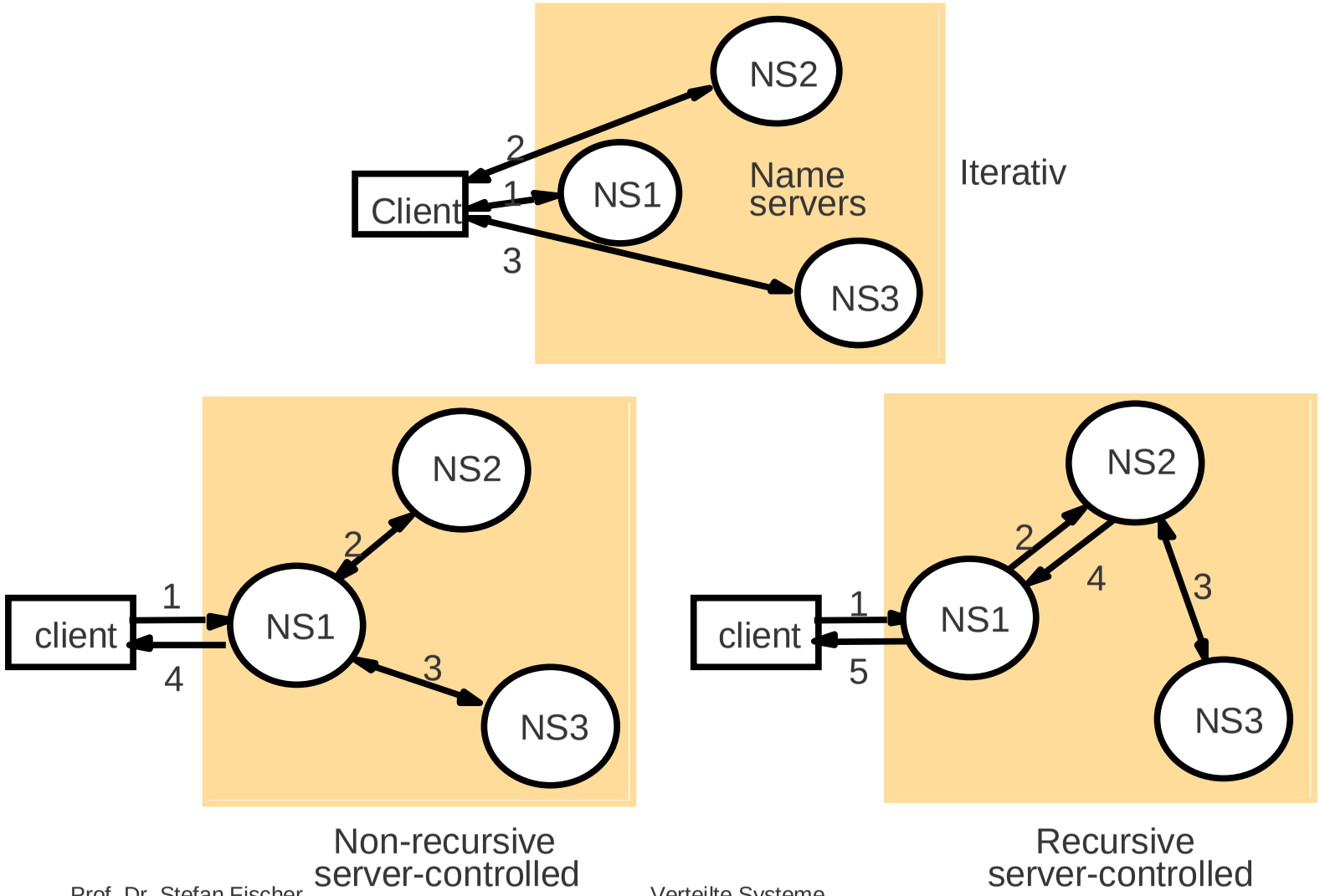
Verwaltung des Namensraums

- Ist in hierarchischen Namensräumen dezentralisierbar.
- Für jede Naming Domain (ein Ausschnitt des Namensraums) gibt es eine *administrative authority*, die
  - die Kontrolle darüber hat, welche Namen in der Domain gebunden werden können,
  - diese Kontrolle delegieren kann (für *sub-domains*)
  - die Datenbank der Bindings administriert.

Namensauflösung

- Namensauflösung: Prozess des Findens der Bindung eines Namens
- Ein Name wird einem Naming Context präsentiert, dann kann folgendes passieren
  - Die Bindung wird gefunden.
  - Die Namensauflösung wird an einen weiteren Kontext weitergegeben, in dem der Prozess erneut abläuft.
  - Der Name wird als nicht gebunden gewertet.

Strategien zur Namensauflösung



Funktionen eines Directory Servers

- Hauptkomponente: Datenbank von Bindungen
- Hauptfunktion: Namensauflösung
- Ansonsten:
  - Erzeugung von Bindungen
  - Löschen von Bindungen
  - Auflistung gebundener Namen
  - Suche von Namen anhand von Attributen

Beispielcode für den Name-Server-Zugriff

- Name Binding (Server):

```
ServerImpl server = new ServerImpl();
orb.connect(server);
org.omg.CORBA.Object nameservice =
     orb.resolve_initial_references("NameService");
NamingContext namingcontext =
NamingContextHelper.narrow(nameservice);
NameComponent name = new NameComponent("Datum", "");
NameComponent path[] = {name};
namingcontext.rebind(path, server);
```

- Name Resolution (Client):

```
Server server;
org.omg.CORBA.Object nameservice =
     orb.resolve_initial_references("NameService");
NamingContext namingcontext = NamingContextHelper.narrow(nameservice);
NameComponent name = new NameComponent("Datum", "");
NameComponent path[] = {name};
server = ServerHelper.narrow(namingcontext.resolve(path));
```

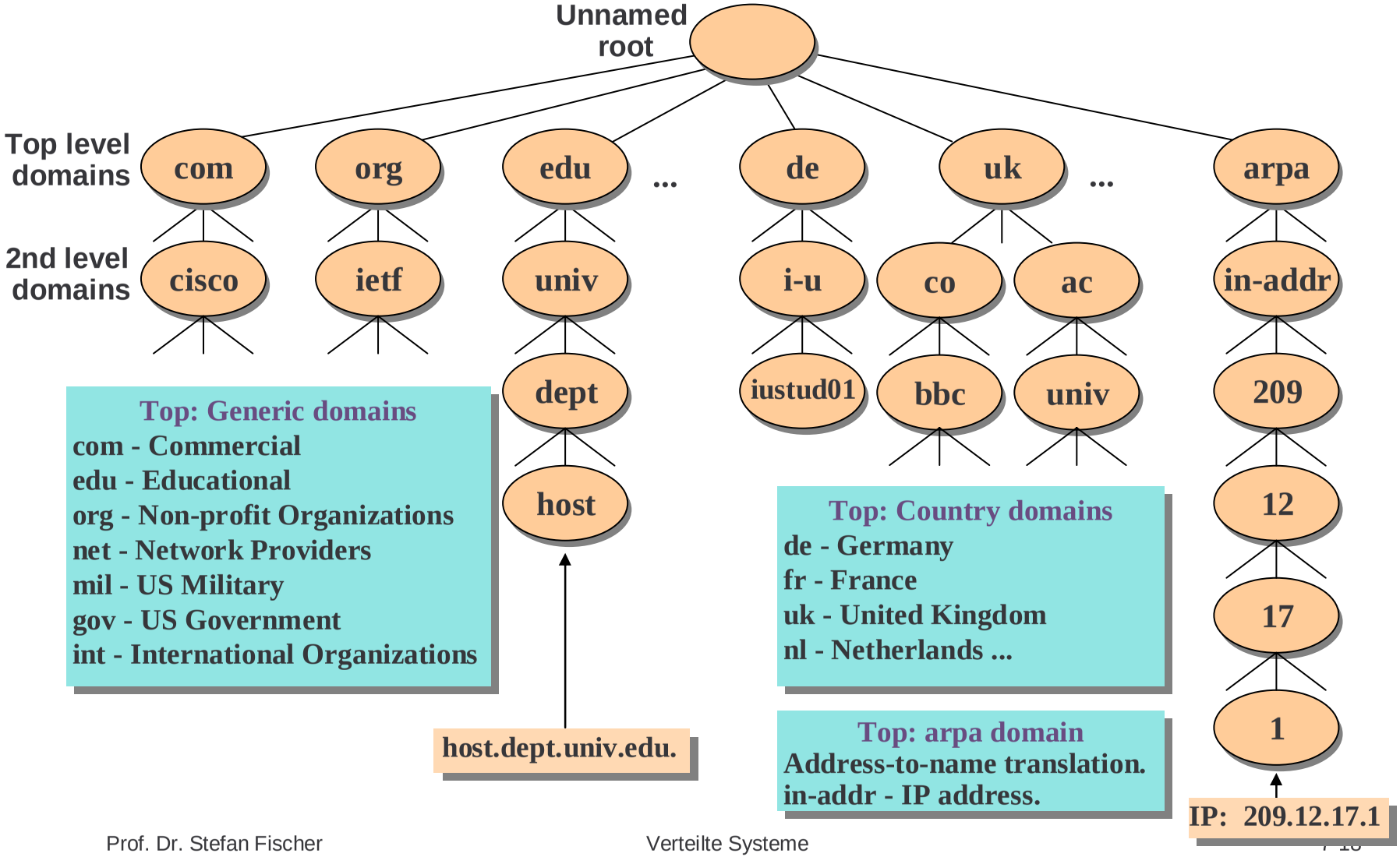
Beispiele für Namens-/Verzeichnisdienste

- Das Telefonbuch
- Die „Gelben Seiten“
- **Internet DNS**
- **Jini lookup service**
- OSI X.500
- CORBA Name Service (`tnameserv` in Suns Implementierung)
- **Java JNDI und LDAP als Integratoren**
- **UDDI**
- ...

Domain Name Service DNS

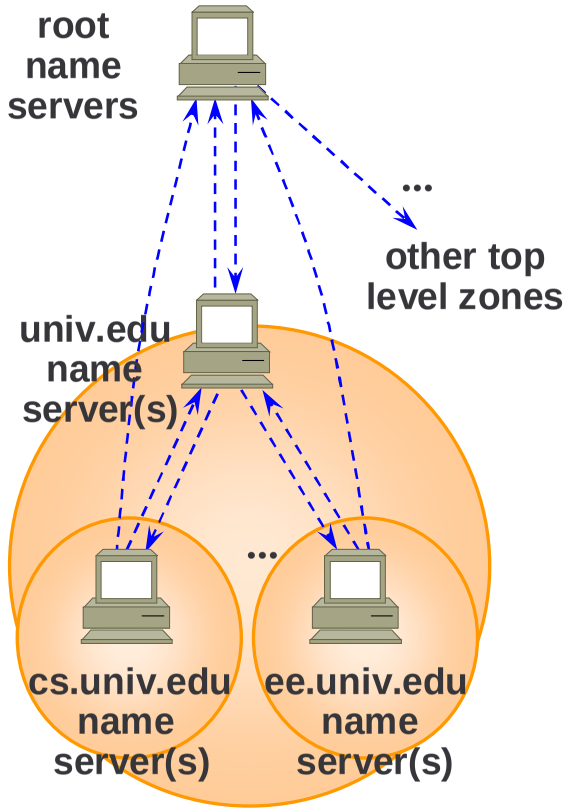
- Aufgabe von DNS
  - Übersetze Namen in Adressen und lasse Anwendungen so Netzdienste nutzen: www.cs.tu-bs.de $\leftrightarrow$ 134.169.34.18.
- Internet Name Service
  - DNS: Domain Name System. Frühe 80er Jahre.
  - Grundlegende Definition: RFC 1034 und 1035, 1987
  - Zahllose weitere RFCs für die Nutzung von DNS, Updates etc.
  - Hierarchisches Namensschema
  - Verteilte Namensdatenbank
  - DNS-Protokoll: query-response protocol.

Namenshierarchie



Server-Hierarchie

- „Zone of authority“
  - Eine Zweig des Namensraums, der getrennt verwaltet wird
  - Der Namensraum ist rekursiv in kleinere Zonen aufgeteilt.



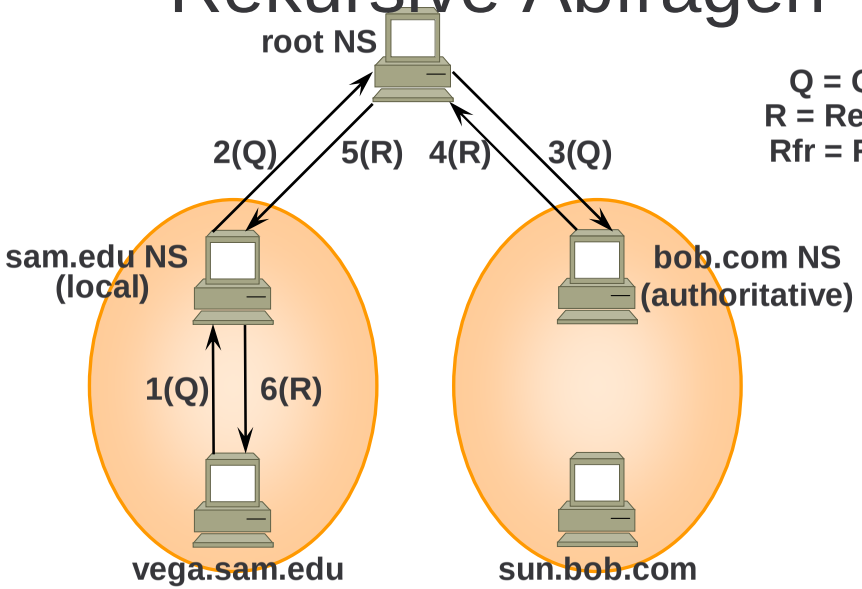
Delegation der Verantwortung

Der Administrator einer Zone verwaltet den/die Name Server (Primär-, Sekundär-)
Jeder Name Server verwaltet die Namensinformation für seine Zone und kennt die Name Server der Unterzonen.

Eine Gruppe von „root name servers“ stellen Name-Syerver-Adressen für die Top-Level-Domains zur Verfügung. Alle anderen Server kennen die Adresse der Root-Server und einiger Name Server der oberen Ebenen.

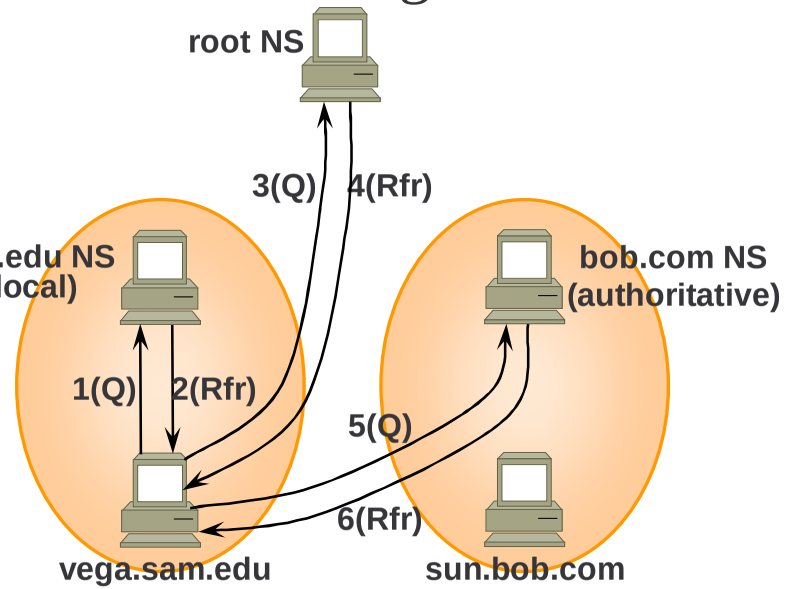
DNS-Abfragen

- Rekursive Abfragen

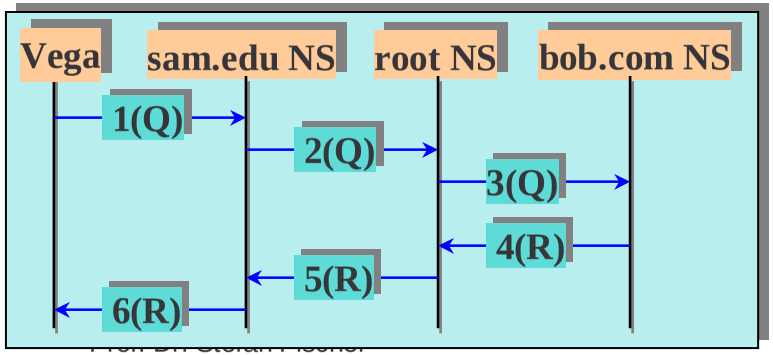


Q = Query
R = Response
Rfr = Referral

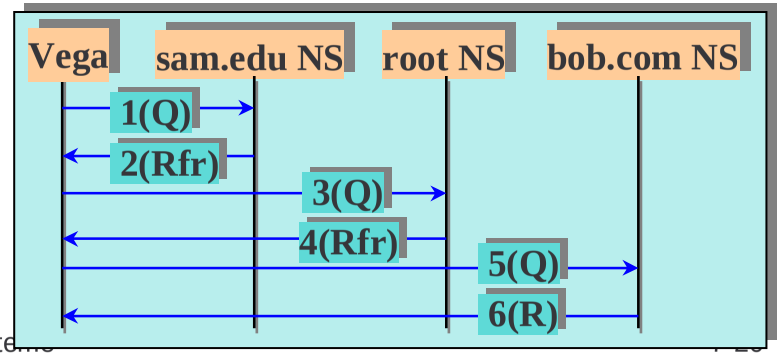
Iterative Abfragen



Query: "Address of sun.bob.com"



Query: "Address of sun.bob.com"



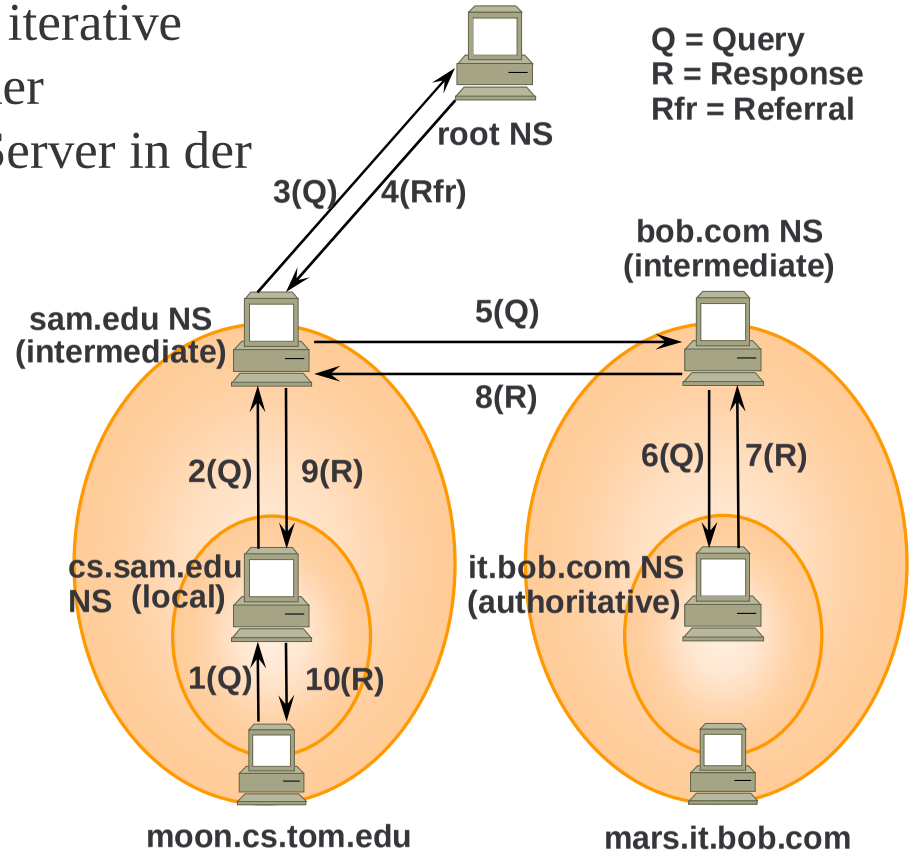
DNS-Abfragen (Forts.)

Iterativ und rekursiv können kombiniert werden

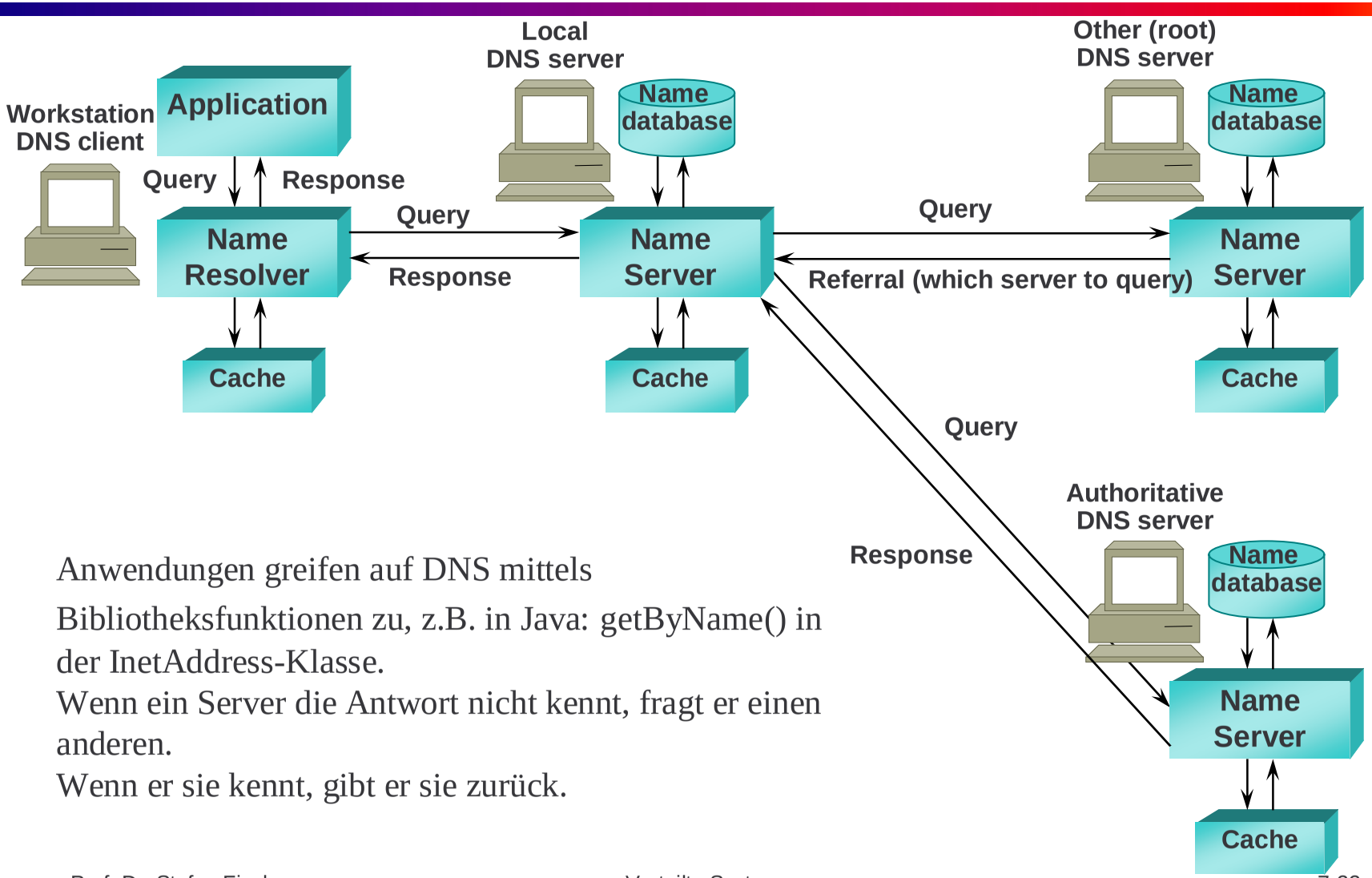
- Client startet rekursive Abfrage.
- Ein Zwischen-Server startet eine iterative Abfrage an Root, gefolgt von einer rekursiven Query an den ersten Server in der Ziel-Domain.

DNS caching

- Name Server speichern erhaltene Antworten (für begrenzte Zeit).
- Eine Anfrage wird nur weitergeleitet, wenn die Antwort nicht im Cache liegt.
- Ziel?



DNS-Implementierung



Anwendungen greifen auf DNS mittels Bibliotheksfunktionen zu, z.B. in Java: getByName() in der InetAddress-Klasse.

Wenn ein Server die Antwort nicht kennt, fragt er einen anderen.

Wenn er sie kennt, gibt er sie zurück.

DNS Datensatztypen

Record type	*Meaning*	*Main contents*
A	A computer address	IP number
NS	An authoritative name server	Domain name for server
CNAME	The canonical name for an alias	Domain name for alias
SOA	Marks the start of data for a zone	Parameters governing the zone
WKS	A well-known service description	List of service names and protocols
PTR	Domain name pointer (reverse lookups)	Domain name
HINFO	Host information	Machine architecture and operating system
MX	Mail exchange	List of ⟨*preference*, *host*⟩ pairs
TXT	Text string	Arbitrary text

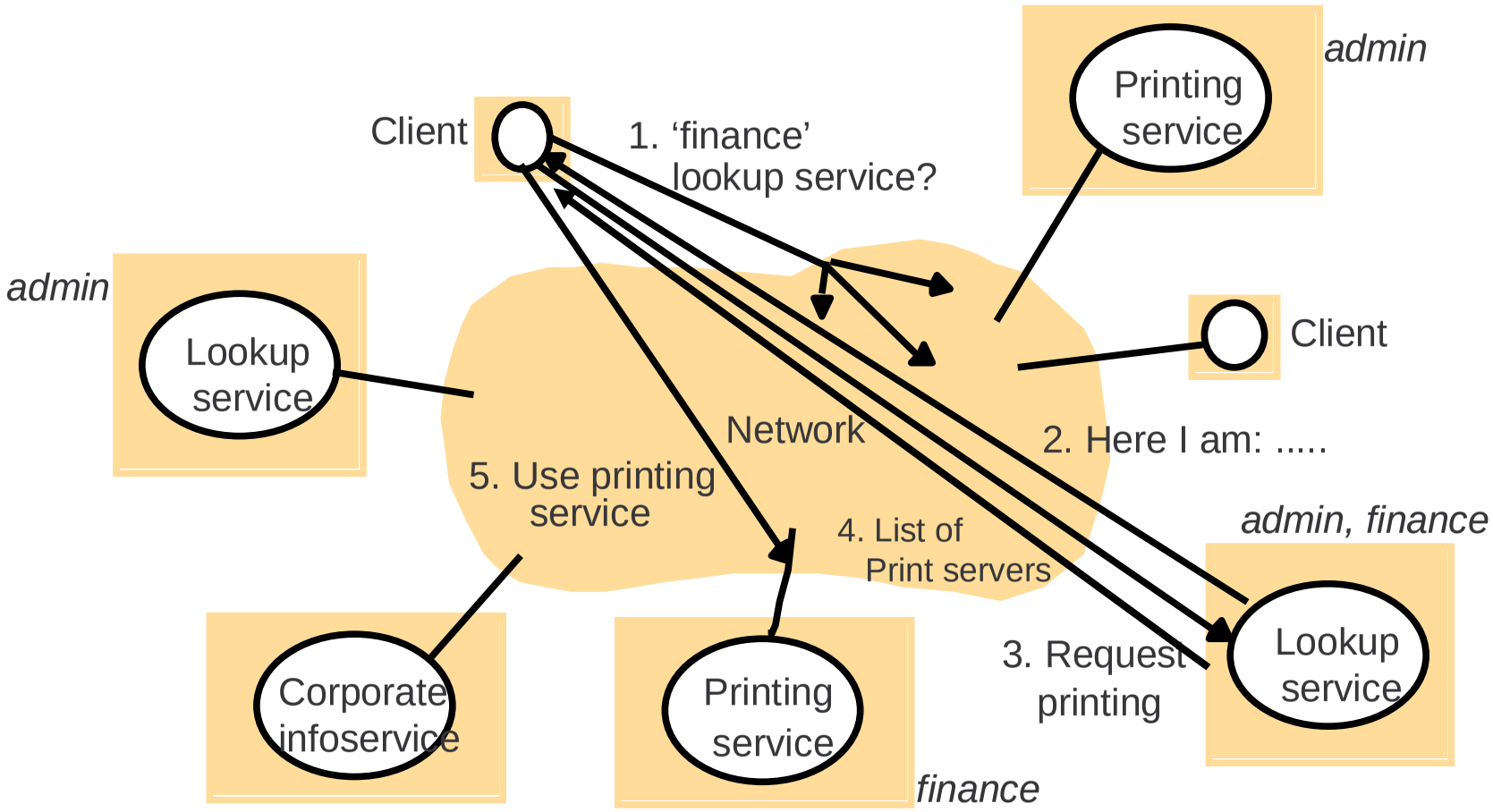
Beispiel: DNS am IBR

```
; Start of Authority:
;
@               IN      SOA     ibr.cs.tu-bs.de. root.ibr.cs.tu-bs.de. (
                                2001103001      ; Serial YYYYMMDDff
                                10800           ; Refresh       3H
                                3600            ; Retry         1H
                                604800          ; Expire 1W
                                86400 )         ; Minimum       1D
;
; Nameserver:
;
@               IN      NS      agitator.ibr.cs.tu-bs.de.

; Mail Exchanger fuer ibr.cs.tu-bs.de:
;
@               IN      MX      10 mumm.ibr.cs.tu-bs.de.

becks           IN      A       134.169.34.65
                IN      MX      10 mumm.ibr.cs.tu-bs.de.
koepi           IN      A       134.169.34.66
                IN      MX      10 mumm.ibr.cs.tu-bs.de.
tuborg          IN      A       134.169.34.67
                IN      MX      10 mumm.ibr.cs.tu-bs.de.
```

Discovery Services

- Ein Verzeichnisdienst, der Service-Registrierungen in Spontaneous-Networking-Umgebungen vornimmt.
- Discovery-Dienste besitzen Möglichkeiten zur automatischen Registrierung und De-Registrierung von Diensten. Dienstangebote sind häufig schwankend.
- Es müssen keine menschlichen Benutzer eingebunden sein!
- Beispiele
  - Jini Lookup Service
  - IETF Service Location Protocol (RFC 2608)

Service Discovery in Jini



LDAP

- Lightweight Directory Access Protocol
- Spezifiziert im RFC 2251
- Definiert ein vereinfachtes Internet-fähiges Protokoll zum Zugriff auf Verzeichnisdienste, die auf der Basis von OSI X.500 arbeiten
- Namensbeschreibung hierarchisch, in etwas anderer Form, Beispiel:
  `cn=Stefan Fischer, ou=cs, o=tu-bs, c=de`
- LDAP gestattet Authentisierung; wichtig für Updates von Bindungen

JNDI

- Java Naming and Directory Service
- Javas Schnittstelle für alle Arten von Namens- und Verzeichnisdiensten
- Mit JNDI können
  - Java-Objekt-basierte Verzeichnisdienste implementiert werden
  - vorhandene Dienste in Java-Anwendungen eingebunden werden
- Beispiel: LDAP, DNS, NIS
- s. http://java.sun.com/products/jndi

JNDI-API

Das API findet sich unter javax.naming.*.
Beispiel: die Klasse `Context`:

```
public interface Context {
        public Object lookup(Name name) throws NamingException;
        public void bind(Name name, Object obj) throws NamingException;
        public void rebind(Name name, Object obj) throws
                                                NamingException;
        public void unbind(Name name) throws NamingException;
        public void rename(Name old, Name new) throws NamingException;
        public NamingEnumeration listBindings(Name name) throws
                                                NamingException;
        ...
        public Context createSubcontext(Name name) throws
                                                NamingException;
        public void destroySubcontext(Name name) throws NamingException;
        ...
};
```

UDDI

- Universal Description, Discovery, and Integration
- Initiative der Wirtschaft, vor allem IBM, Microsoft, Ariba
- Ziel: erreiche Interoperabilität von Web Services
- Vorgehen:
  - Standardisierung von Schnittstellen des UDDI
  - Gemeinsamer Betrieb eines Verzeichnisdienstes für E-Commerce-Partner und – Anwendungen
- s. Kapitel 5 zu Web Services

Problem mit URL-Namen

- URLs mischen zwei Dinge miteinander:
  - Identifikation von *und*
  - Zugriff auf Ressourcen (Host-, Protokollangabe)
- Dies erzeugt Skalierbarkeitsprobleme, da beide Dienste unterschiedliche Anforderungen haben.
- Beispiel: Replikation von Dokumenten erfordert die Verwendung mehrerer verschiedener URLs.
- Dies ist ein für den Benutzer nicht-transparentes Vorgehen.

Lösung, Teil 1: URNs

- Verwendung von Uniform Resource Names (URNs), s. RFC 2141, 2648, 2276.
- Aufgabe: Bereitstellung eines einzigen Namens für eine Menge von URLs, die alle auf Replicas des gleichen Dokuments verweisen = Identifikation einer Ressource
- Der Zugriff auf die entsprechende Ressource erfolgt über die Auflösung des URN mittels eines Locator-Dienstes (Finden des „besten“ Ortes einer Ressource).
- Allerdings sind URNs nicht unbedingt immer verständlich für den menschl. Benutzer („must be human transcribable“, RFC 1737).
- Beispiel: s. RFC 2648
  - `urn:ietf:rfc:2141`
  - `urn:ietf:std:50`

Lösung Teil 2: Human-Friendly Names (HFN)

- Generelles Prinzip: Organisation der Namen nach dem DNS-Schema bzw. in der Form der Namen von Newsgroup-Namen.
- Beispiele:
  - `hfn:stable.src.linux.org` zur Identifikation der aktuellen stabilen Version der Linuxquellen
  - `hfn:aktuelle-ausgabe.der-spiegel.de`
- Mehrere HFNs können auf dieselbe Ressource verweisen (wie symbolische Links im Dateisystem)

HFN-Architektur

- HFN-to-URL ist eine M:N-Beziehung, da mehrere HFNs auf dieselbe Menge von URLs verweisen können
- Die Beziehung ist dynamisch, da
  - Ressourcen neue Namen bekommen und
  - neue Replicas erzeugt werden können.
- Zum effizienten Management wird deshalb die Namensauflösung in zwei Schritte aufgeteilt:
  - HFN-to-URN
  - URN-to-URL
- Dementsprechend werden zur üblichen Browser-Server-Architektur weitere Komponenten addiert, nämlich ein HFN-to-URL Proxy, ein Name Service und ein Location Service.

HFN-to-URL Proxy

- Dient als Front-End für die beiden anderen Dienste
- Jede Anfrage des Web Browsers nach einem HFN geht zuerst an diese Komponente.
- Aufgaben
  - Erkennen einer HFN
  - Anfrage an den Name Service, um die richtige URN zu finden
  - Anfrage an den Location Service, um die beste URL für die URN zu finden
- Implementierbar z.B. als separater Prozess oder als Browser-Plugin

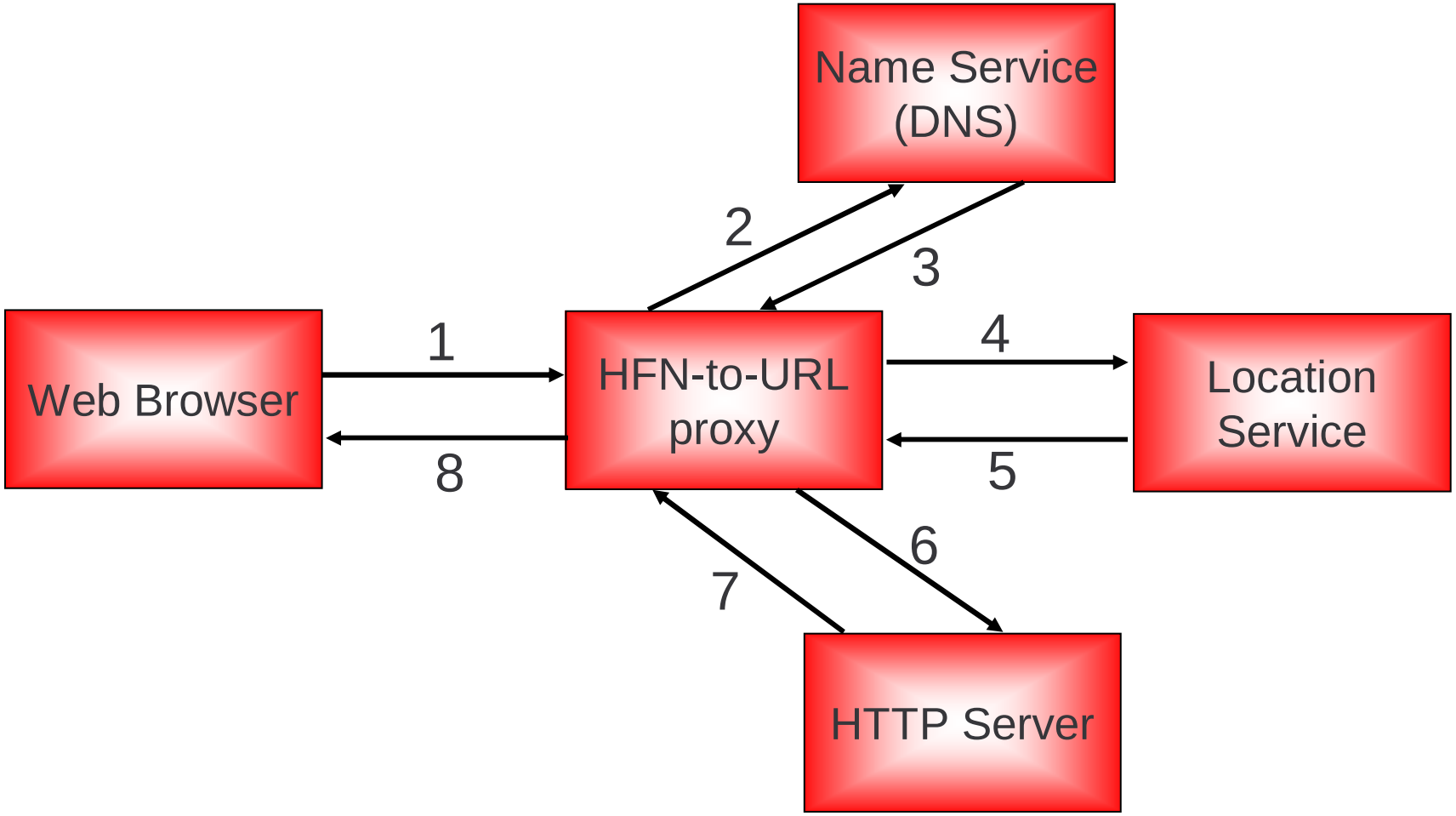
Name Service

- Als Name Service wird DNS eingesetzt.
- Wegen der Verwendung der DNS-Struktur bei den Namen bietet sich dieses an, da entsprechend auch die Infrastruktur von DNS genutzt werden kann.
- Es wird ein neuer Typ eines Resource Records eingeführt, der HFN-URN-Mappings beschreibt.

Location Service

- Aufgabe:
  - Abbildung der URN auf eine Menge von URLs
  - Finden der „besten“ URL für eine bestimmte Anfrage
- Das zugrundeliegende Netzwerk ist in Domains entlang geographischer Regionen aufgeteilt, so dass die jeweils nächstliegende URL für eine bestimmte Anfrage gefunden werden kann.

Zugriff auf Web-Ressourcen über HFNs



Verteilte Dateisysteme: Einführung

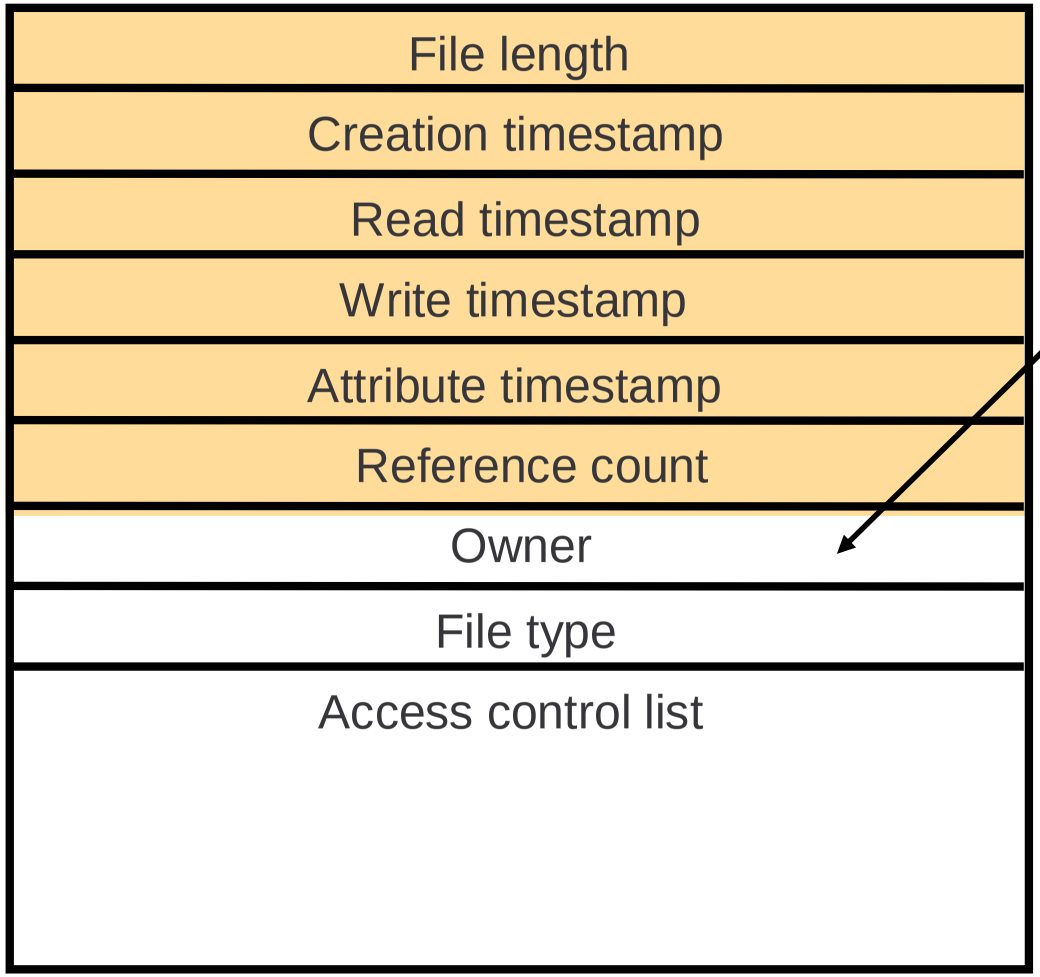
- Ziel verteilter Systeme: gemeinsame Nutzung von Ressourcen
- Wichtigste Ressource: Information
- Im Internet: durch WWW realisiert
- Im Intranet: verteilte Dateisysteme
  - Persistente Speicherung von Daten und Programmen im Auftrage von Clients und Zugriff auf die aktuellsten Versionen
- Wichtig: Transparenz, für den Client soll die Illusion eines lokalen Dateisystems bestehen

Komponenten eines Dateisystems

Directory module:	relates file names to file IDs
File module:	relates file IDs to particular files
Access control module:	checks permission for operation requested
File access module:	reads or writes file data or attributes
Block module:	accesses and allocates disk blocks
Device module:	disk I/O and buffering

- Verteiltes Dateisystem zusätzlich:
  - Client-Server-Kommunikation
  - Verteilte Namensgebung und Auffinden von Dateien

Beschreibung von Dateien



- Dateien bestehen aus Daten und Attributen
- Nur wenige Attribute können vom Benutzer geändert werden

- Spezielle Dateien: Directories (Verzeichnisse)
  - Abbildung von textuellen Namen auf IDs des Dateisystems
  - Hierarchien von Verzeichnissen

API eines Datesystems: UNIX

filedes = open(name, mode) *filedes = creat(name, mode)*	Opens an existing file with the given *name*. Creates a new file with the given *name*. Both operations deliver a file descriptor referencing the open file. The *mode* is *read*, *write* or both.
status = close(filedes)	Closes the open file *filedes*.
count = read(filedes, buffer, n)	Transfers *n* bytes from the file referenced by *filedes* to *buffer*.
count = write(filedes, buffer, n)	Transfers *n* bytes to the file referenced by *filedes* from buffer. Both operations deliver the number of bytes actually transferred and advance the read-write pointer.
pos = lseek(filedes, offset, whence)	Moves the read-write pointer to offset (relative or absolute, depending on *whence*).
status = unlink(name)	Removes the file *name* from the directory structure. If the file has no other names, it is deleted.
status = link(name1, name2)	Adds a new name (*name2*) for a file (*name1*).
status = stat(name, buffer)	Gets the file attributes for file *name* into *buffer*.

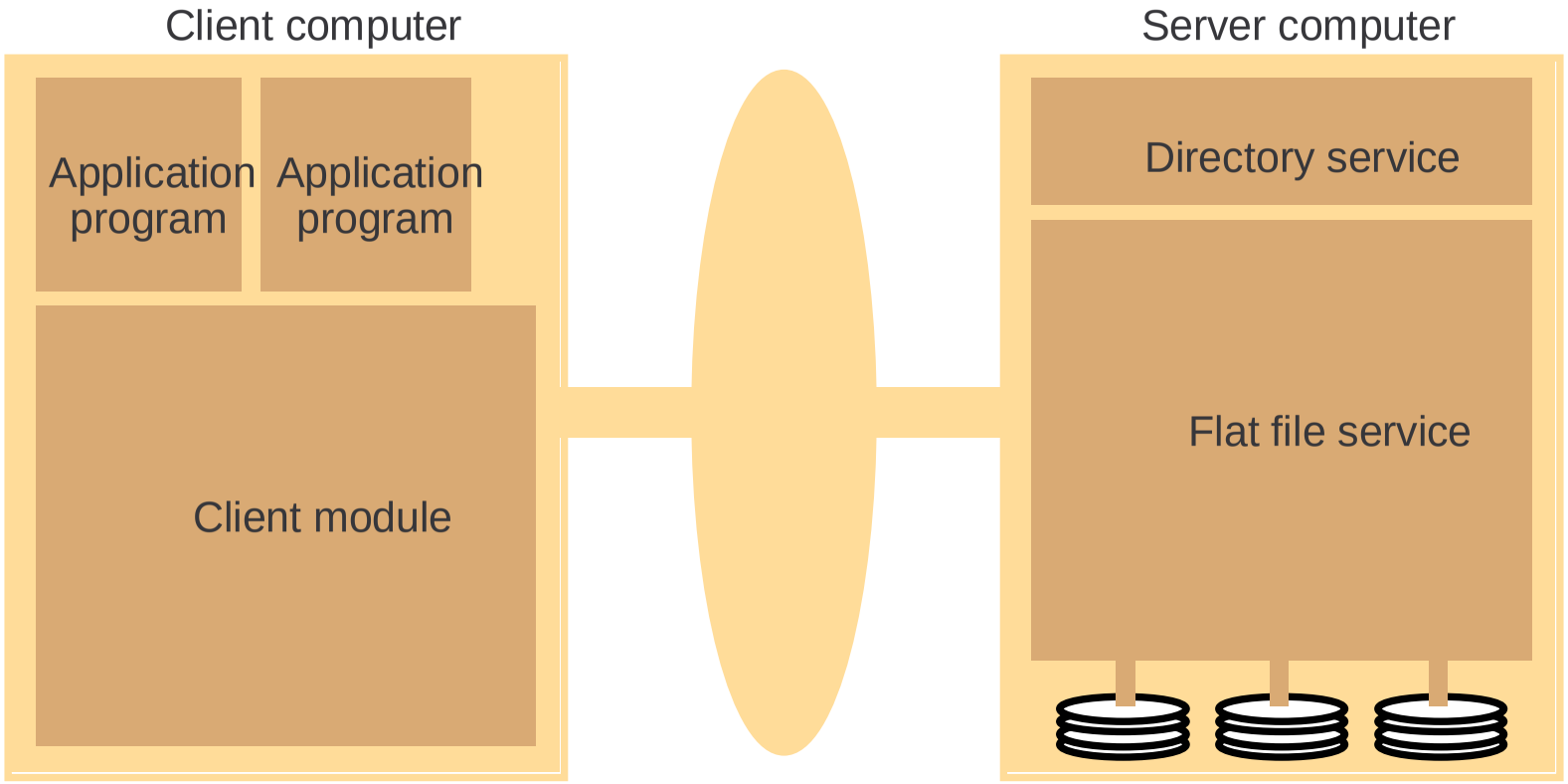
Anforderungen an ein vert. Dateisystem

- Transparenz
  - Zugriffstransparenz: Clients bemerken nichts von der Verteilung
  - Ortstransparenz und Mobilitätstransparenz : Dateien können physikalisch verschoben werden, ohne ihren Pfad zu ändern
  - Leistungstransparenz: auch unter veränderlicher Last sollte das System befriedigend funktionieren
  - Skalierungstransparenz: der Dienst kann erweitert werden, um auch in größeren Netzen zu funktionieren
- Nebenläufige Schreibzugriffe auf Dateien
- Dateienreplikation
- Unterstützung heterogener Hardware und Betriebssysteme

Anforderungen (II)

- Fehlertoleranz: relative leicht zu implementieren mit zustandslosen Servern
- Konsistenz: insbesondere bei Replikation
- Sicherheit: Zugriffskontrolle, Verschlüsselung, evtl. digitale Signaturen
- Effizienz: die Leistung des Systems sollte in etwa vergleichbar sein mit einem lokalen Dateisystem

Architektur eines Dateidienstes



- Wichtigstes Ziel: Aufteilung der Aufgaben, einfachere Implementierung der einzelnen Komponenten

API des Flat File Service

Read(FileId, i, n) -> Data — throws *BadPosition*	If $1 \leq i \leq Length(File)$: Reads a sequence of up to *n* items from a file starting at item *i* and returns it in *Data*.
Write(FileId, i, Data) — throws *BadPosition*	If $1 \leq i \leq Length(File)+1$: Writes a sequence of *Data* to a file, starting at item *i*, extending the file if necessary.
Create() -> FileId	Creates a new file of length 0 and delivers a UFID for it.
Delete(FileId)	Removes the file from the file store.
GetAttributes(FileId) -> Attr	Returns the file attributes for the file.
SetAttributes(FileId, Attr)	Sets the file attributes (only those attributes that are not shaded in).

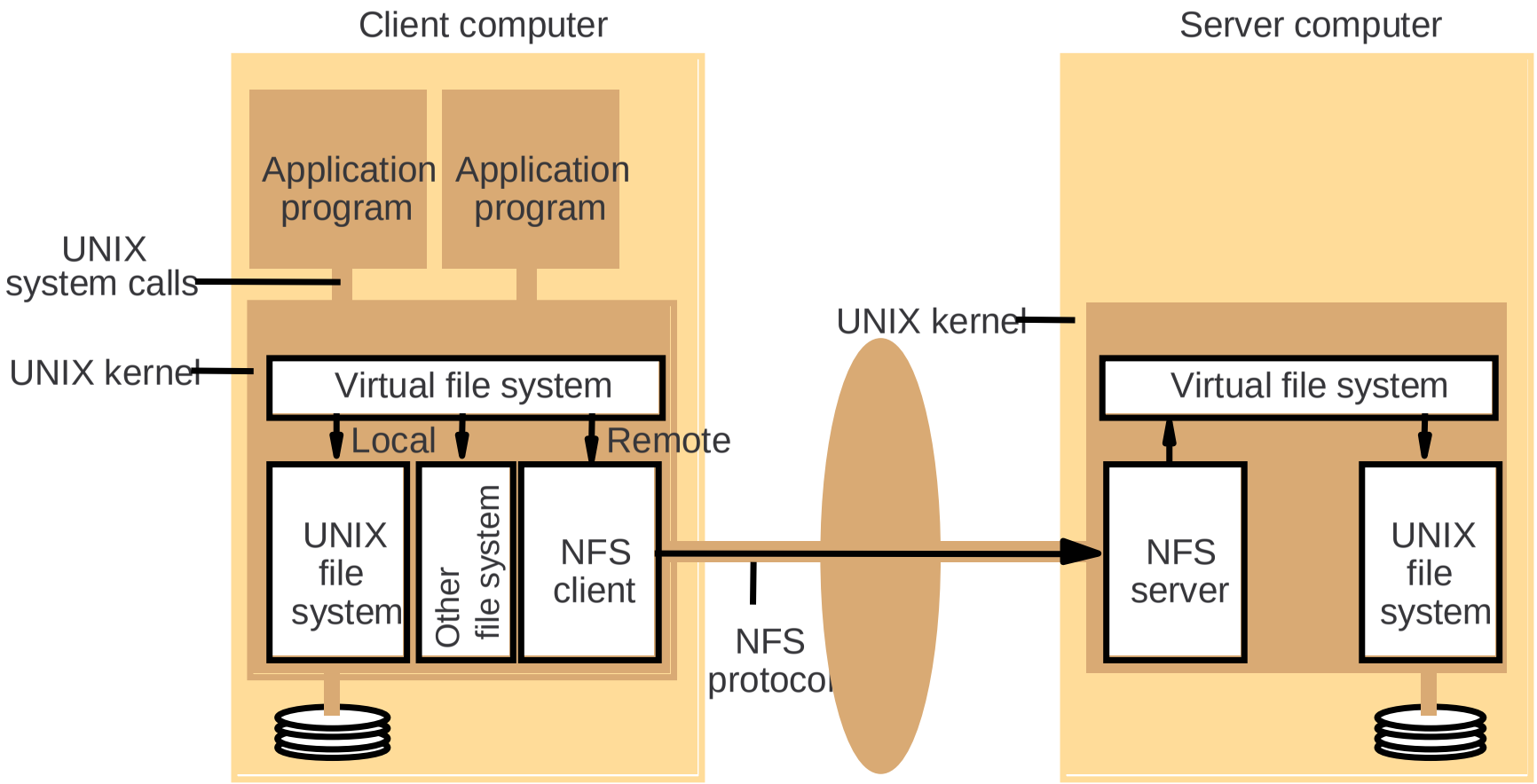
API des Directory Service

Lookup(Dir, Name) -> FileId — throws*NotFound*	Locates the text name in the directory and returns the relevant UFID. If *Name* is not in the directory, throws an exception.
AddName(Dir, Name, File) — throws*NameDuplicate*	If *Name* is not in the directory, adds (*Name*, *File*) to the directory and updates the file's attribute record. If *Name* is already in the directory: throws an exception.
UnName(Dir, Name) — throws*NotFound*	If *Name* is in the directory: the entry containing *Name* is removed from the directory. If *Name* is not in the directory: throws an exception.
GetNames(Dir, Pattern) -> NameSeq	Returns all the text names in the directory that match the regular expression *Pattern*.

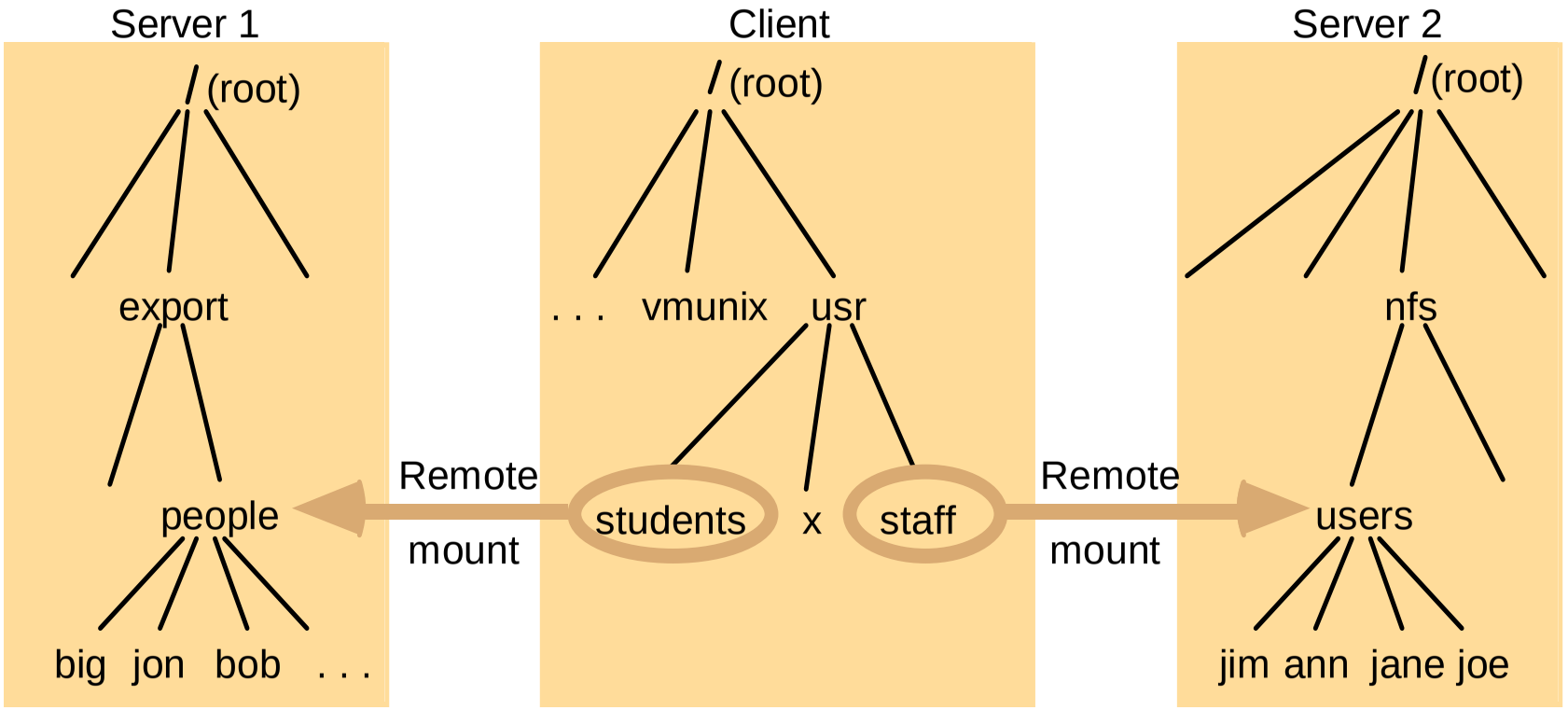
Network File System (NFS)

- Eingeführt 1985 von Sun Microsystems
- Weite Verbreitung in Industrie und Universitäten
- Public Domain, von jedem implementierbar, Protokoll beschrieben in RFC 1813
- NFS Client und Server Teil des UNIX-Kernels
- Aufgabe: transparenter Zugriff auf Dateien über Rechnergrenzen hinweg

Architektur von NFS



“Mounten” von Dateisystemen



Note: The file system mounted at */usr/students* in the client is actually the sub-tree located at */export/people* in Server 1; the file system mounted at */usr/staff* in the client is actually the sub-tree located at */nfs/users* in Server 2.

Realisierung der Anforderungen?

- Zugriffstransparenz ☺
- Ortstransparenz: Benutzer bemerken nichts von Ortsverschiebungen ☺
- Mobilitätstransparenz: bei Verschiebung von Dateien müssen sämtliche Tabellen in den Clients modifiziert werden ☹
- Skalierung: 😐 könnte besser sein mit replizierten Dateisystemen
- Replikation: nur mit Read, keine Updates ☹
- Heterogenität: ☺

Anforderungen? (II)

- Fehlertoleranz: ☺
- Konsistenz: 😐 eingeschränkt: sehr gut im lokalen Bereich, ansonsten kritisch
- Sicherheit: ☺ Integration von Kerberos
- Effizienz: ☺ praktische Erfahrungen zeigen exzellentes Leistungsverhalten

Weitere Literatur

- D. Larisch: *Verzeichnisdienste im Netzwerk*, Carl-Hanser-Verlag, 2000.
- G. Ballentijn et al.: *Scalable Human-Friendly Resource Names*, IEEE Internet Computing, Vol.5, No.5, Sept./Oct. 2001.

- Hal Stern: *Verwaltung von UNIX-Netzwerken mit NFS und NIS*, O‘Reilly, 1995.