Datei (File)

Menge von logisch zusammengehörigen und auf einem Medium permanent gespeicherten Daten, die über einen Bezeichner eindeutig identifizierbar ist.

Beispiele unter Unix/Linux:
reguläre Datei, Verzeichnis, symbolischer Link, Pipe, Socket, Gerätedatei.

Dateiverzeichnis (Verzeichnis, Directory)

Spezielle Datei, in der in Einträgen (Entries) Informationen über Dateien zu dem Zweck gespeichert werden, sie effizient finden und verwalten zu können.

Unter Unix/Liniux kann ein Verzeichnis Einträge erhalten auf: reguläre Dateien, Verzeichnisse, symbolische Links, Pipes, Gerätedateien.

Dateisystem (File System)

1.Def.

Software-System zur Unterstützung der Verwaltung und Nutzung von in Dateien gespeicherten Daten.

Beispiele unter Unix/Linux:
ext2, ext3, ext4, XFS, ZFS, JFS, ReiserFS

Dateisystem (File System)

2.Def.

Hierarchische Organisation, z.B. ein Baum, aus Verzeichnissen und Dateien.

Dämon (Demon)
Im Hintergrund laufender Prozess, der einem Client Dienste zur Verfügung stellt.

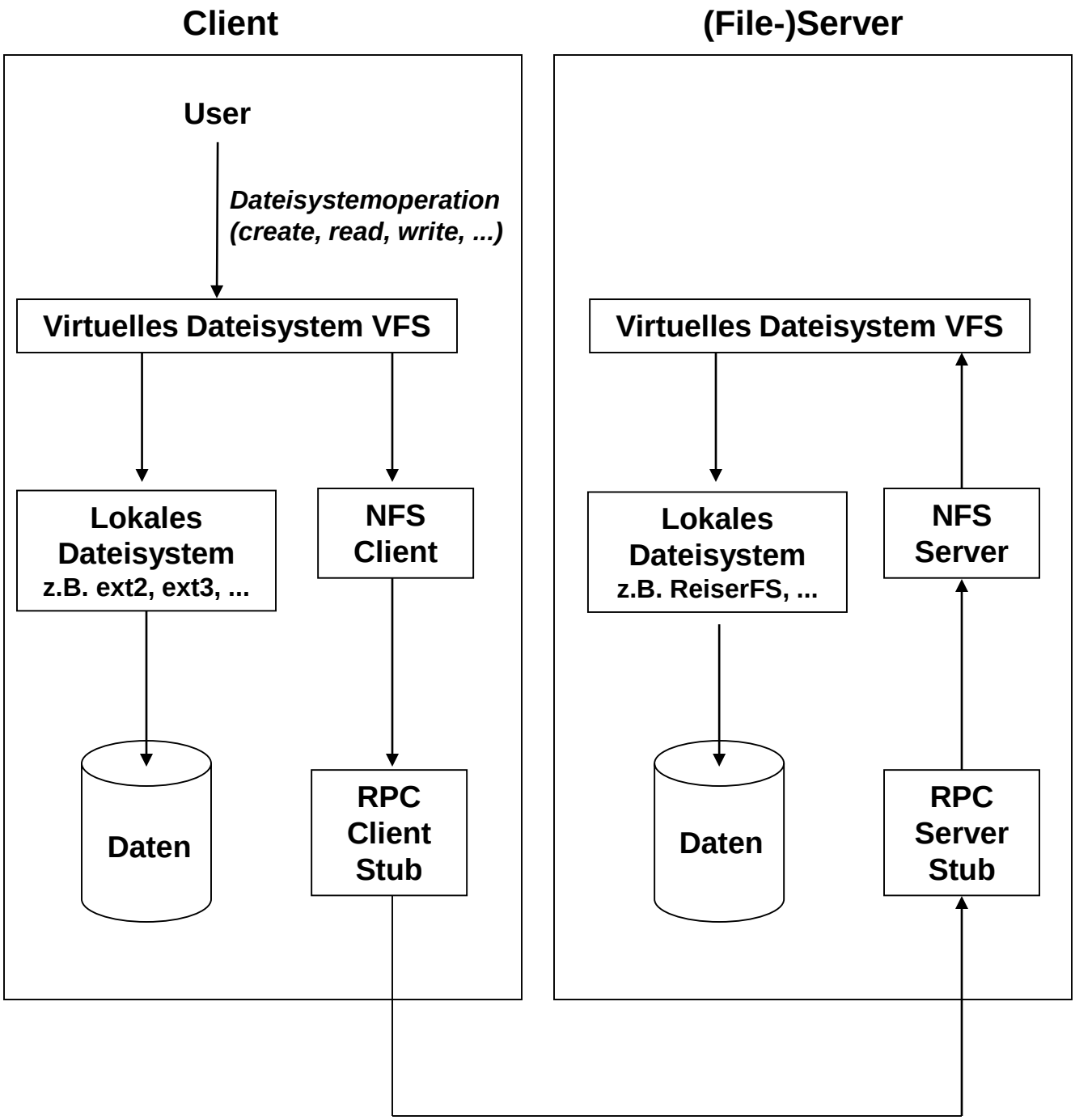
Harter Link (hard link)
Eintrag in ein Verzeichnis zum Verweis auf eine Datei.

Besteht unter Unix/Linux aus Name und i-Node-Nummer der Datei, die i-Node-Nummer repräsentiert eine Datei.

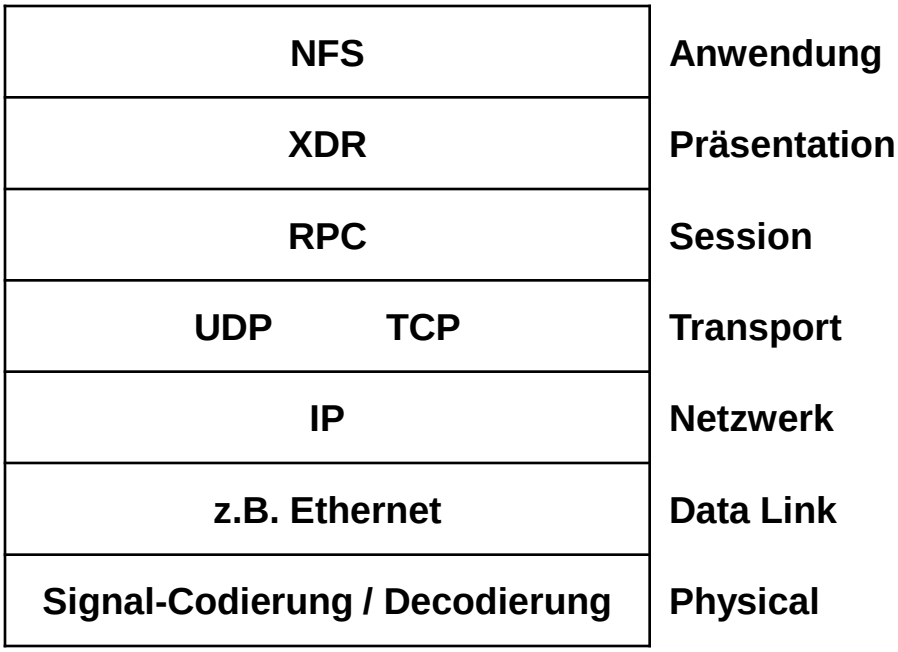
Eigenschaft:
Ein harter Link verweist nur auf Dateien oder Verzeichnisse im *gleichen* Dateisystem.

Symbolischer Link (soft link)
Spezielle Datei, die den Pfadnamen einer (Ziel-)Datei enthält.

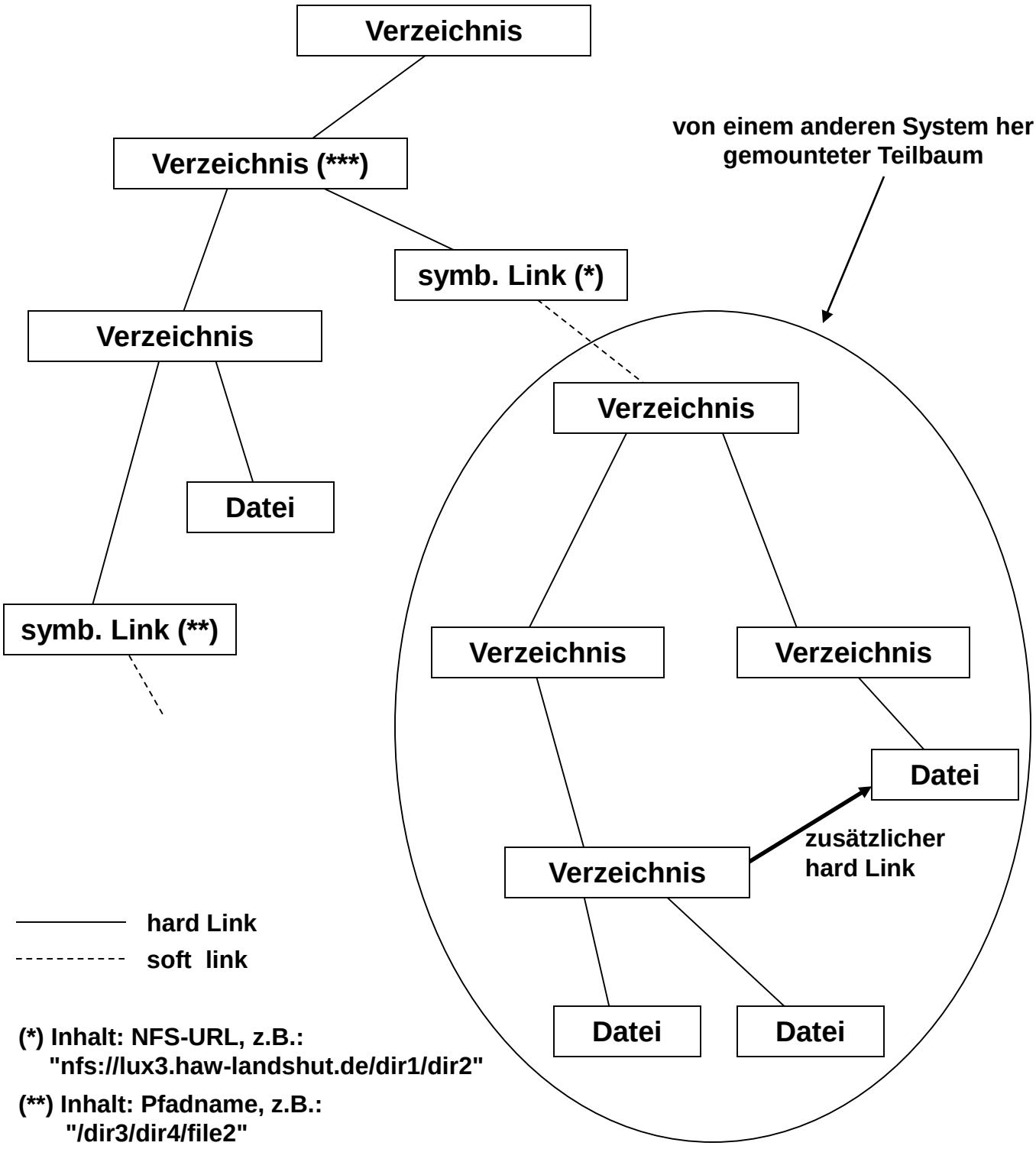
Eigenschaft:
Erlaubt auch einen Verweis auf ein *anderes* Dateisystem, im Gegensatz zu einen Hard Link.



NFS-Architektur



NFS-Kommunikationsstack



NFS-Dateisystem

type	**Dateityp (regulär, Verzeichnis, symbolischer Link)**
size	**Dateigröße (in Bytes)**
change	**Info, ob/wann die Datei geändert wurde**
fsid	**ID des Dateisystems, auf der die Datei liegt**
link_support	**wahr, falls das Dateisystem "harte" Links unterstützt**
symlink_support	**wahr, falls das Dateisystem symbolische Links unterstützt**
filehandle	**Datei-Handle der Datei (ID der Datei auf dem Server)**
	(weitere Attribute .....)

Zwingend erforderliche Dateiattribute

ACL	**Access Control List der Datei (*)**
fileid	**ID der Datei im Dateisystem**
mimetype	**MIME Typ/Subtyp der Datei (text/html, image/gif, ...)**
mode	**Unix-artige Mode und Permission Bits der Datei**
owner	**Owner der Datei (String)**
time_access	**Zeitpunkt des letzten Zugriffs auf die Datei**
time_create	**Zeitpunkt der Erzeugung der Datei**
time_modify	**Zeitpunkt der letzten Änderung der Datei**
	(weitere Attribute .....)

Empfohlene Dateiattribute

(*) Erlaubt / nicht erlaubt werden können:
READ_DATA, WRITE_DATA, APPEND_DATA, EXECUTE, ... für Dateien,
LIST_DIRECTORY, ADD_SUBDIRECTORY, ADD_FILE, DELETE_CHILD, ...
für Verzeichnisse.

(NFS Version 4)

NFS: Datei-Attribute

GETATTR	**Lesen der Attributwerte einer Datei**
SETATTR	**Setzen von Attributwerten einer Datei**
LOOKUP	**Aufsuchen (Suchen und Finden) des Datei-Handles einer Datei anhand ihres Namens und des Datei-Handles des zu durchsuchenden Verzeichnisses (Signatur s.u.)**
READLINK	**Lesen des in einem symbolischen Link gespeicherten Pfadnamens**
READ	**Lesen aus einer Datei**
WRITE	**Schreiben in eine Datei**
CREATE	**Erzeugen einer regulären Datei**
MKDIR	**Erzeugen eines Verzeichnisses**
SYMLINK	**Erzeugen eines symbolischen Links auf eine Datei**
REMOVE	**Löschen einer Datei**
RMDIR	**Löschen eines Verzeichnisses**
LINK	**Erzeugen eines ("harten") Links auf eine Datei**
	(weitere Operationen ...)

```
// LOOKUP-Operation aus RFC 1813 (NFSv3):

LOOKUP3res NFSPROC3_LOOKUP (LOOKUP3args);

struct LOOKUP3args {
        diropargs3      what;           // what: object to look up
};

struct diropargs3 {
        nfs_fh3         dir;            // file handle for the directory to search
        filename3       name;           // file name of object to be searched for
};

struct nfs_fh3 { opaque data<NFS3_FHSIZE>; };       // file handle size = 64

typedef string filename3<>;                         // <> = variable, but no max length

union LOOKUP3res switch  (...) {case NFS3_OK: LOOKUPresok resok, ...};

struct LOOKUPresok {nfs_fh3 object, ... };          // file handle of found object
```

NFSv3: Dateisystem-Operationen

CLOSE	**Schließen einer regulären Datei**
CREATE	**Erzeugen einer nicht-regulären Datei (Verzeichnis, symb. Link)**
GETATTR	**Lesen der Attributwerte einer Datei**
LINK	**Erzeugen eines ("harten") Links auf eine Datei**
LOOKUP	**Aufsuchen (Suchen und Finden) des Datei-Handles einer Datei anhand ihres Namens im Verzeichnis mit dem aktuellen Filehandle**
OPEN	**Erzeugen (wenn noch nicht vorhanden) und Öffnen einer regulären Datei**
READ	**Lesen aus einer Datei**
READLINK	**Lesen des in einem symbolischen Link gespeicherten Pfadnamens**
REMOVE	**Löschen einer Datei**
SETATTR	**Setzen von Attributwerten einer Datei**
WRITE	**Schreiben in eine Datei**
	(weitere Operationen .....)

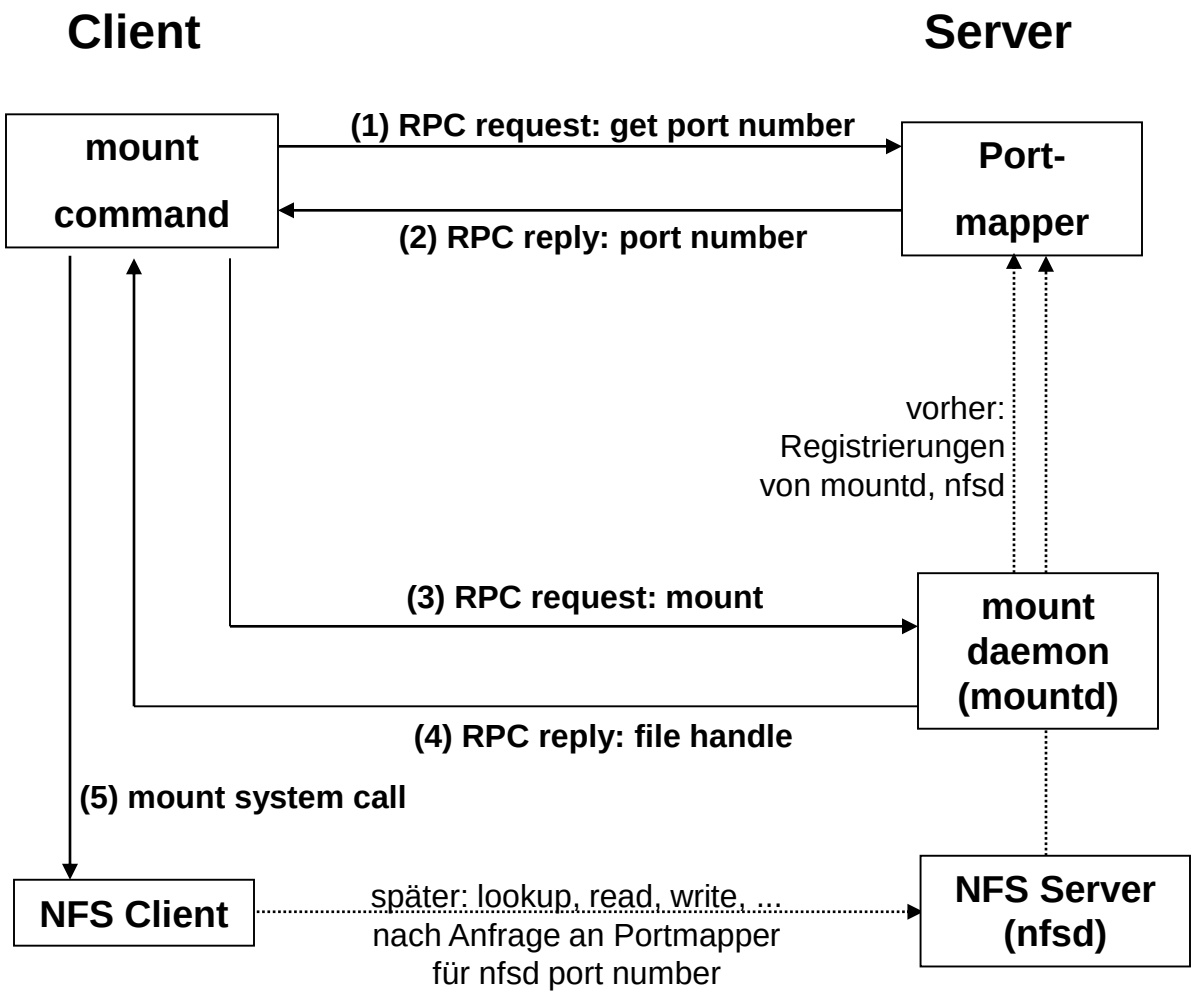
// LOOKUP-Operation aus RFC 3530 (NFSv4):

Synopsis: (current file handle), component → (current file handle)

/* LOOKUP evaluates the component and if the object exists the current file handle is replaced with the component's file handle */

```
Argument:  struct LOOKUP4args {
               /* CURRENT_FH: directory */      // cfh represents directory
               component4   objname;
           };
           typedef   utf8str_cs   component4;   // represents path name comp's
           typedef   utf8string   utf8str_cs;   // case-sensitive UTF-8 string
Result:    struct LOOKUP4res {
               /* CURRENT_FH: object */         // cfh represents found object
               nfsstat4         status;
           };
           enum nfsstat4 { NFS4_OK = 0, ..... };
```

NFSv4: Dateisystem-Operationen



Ausführung des mount Kommandos auf dem Client:

(1) RPC- Anfrage an den Portmapper auf dem Server nach der Portnummer des Mount Daemons *mountd auf dem Server*

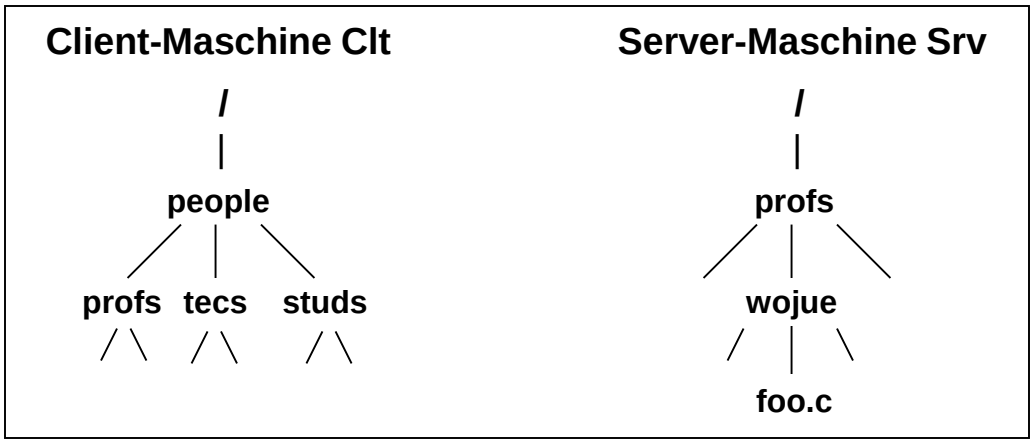
(2) Rückgabe der Portnummer von *mountd*

(3) RPC-Anfrage an *mountd* nach dem Datei-Handle des einzuhängenden entfernten Dateisystems (*mount*)

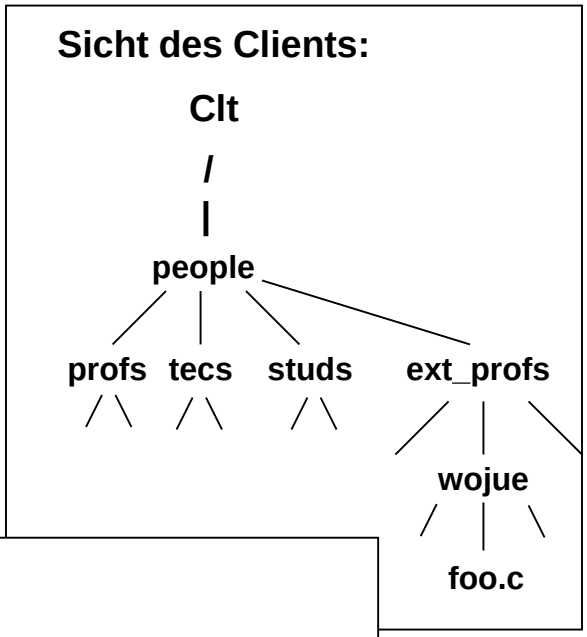
(4) Rückgabe des Datei-Handles des einzuhängenden Dateisystems

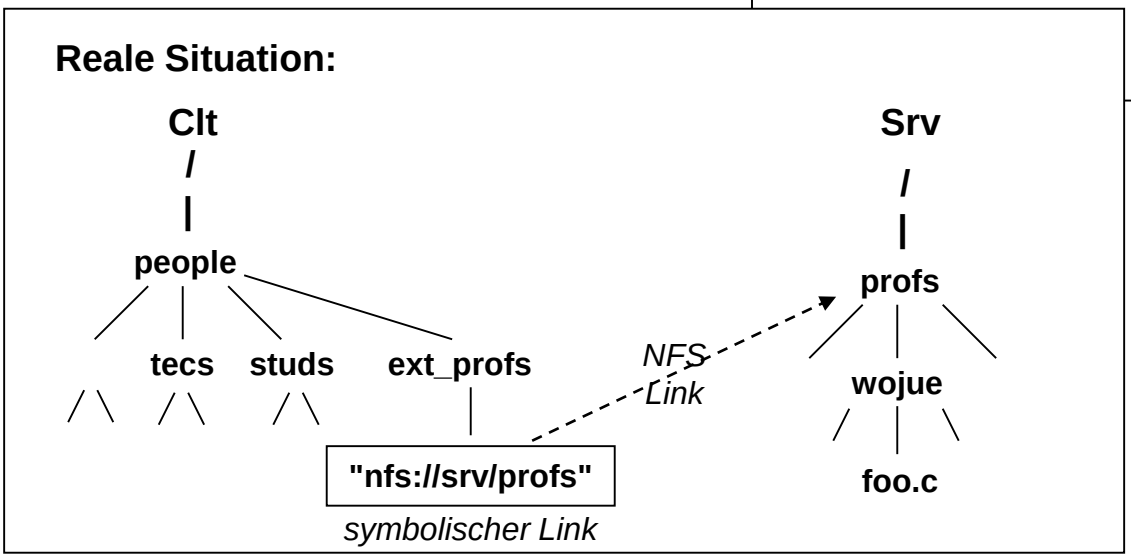
***(5)* *mount* System Call an den NFS Client, um den erhaltenen Datei-Handle mit einem Mount Point (Verzeichnis) im lokalen Dateisystem zu assoziieren, d.h. das entfernte Dateisystem einzuhängen. Dieses Datei-Handle wird dann bei allen vom Benutzer ausgehenden Dateisystem-Operationen verwendet, um auf Dateien im gemounteten Dateisystem zuzugreifen, ohne dass der Benutzer selbst es kennen muss.**

mount-Protokoll



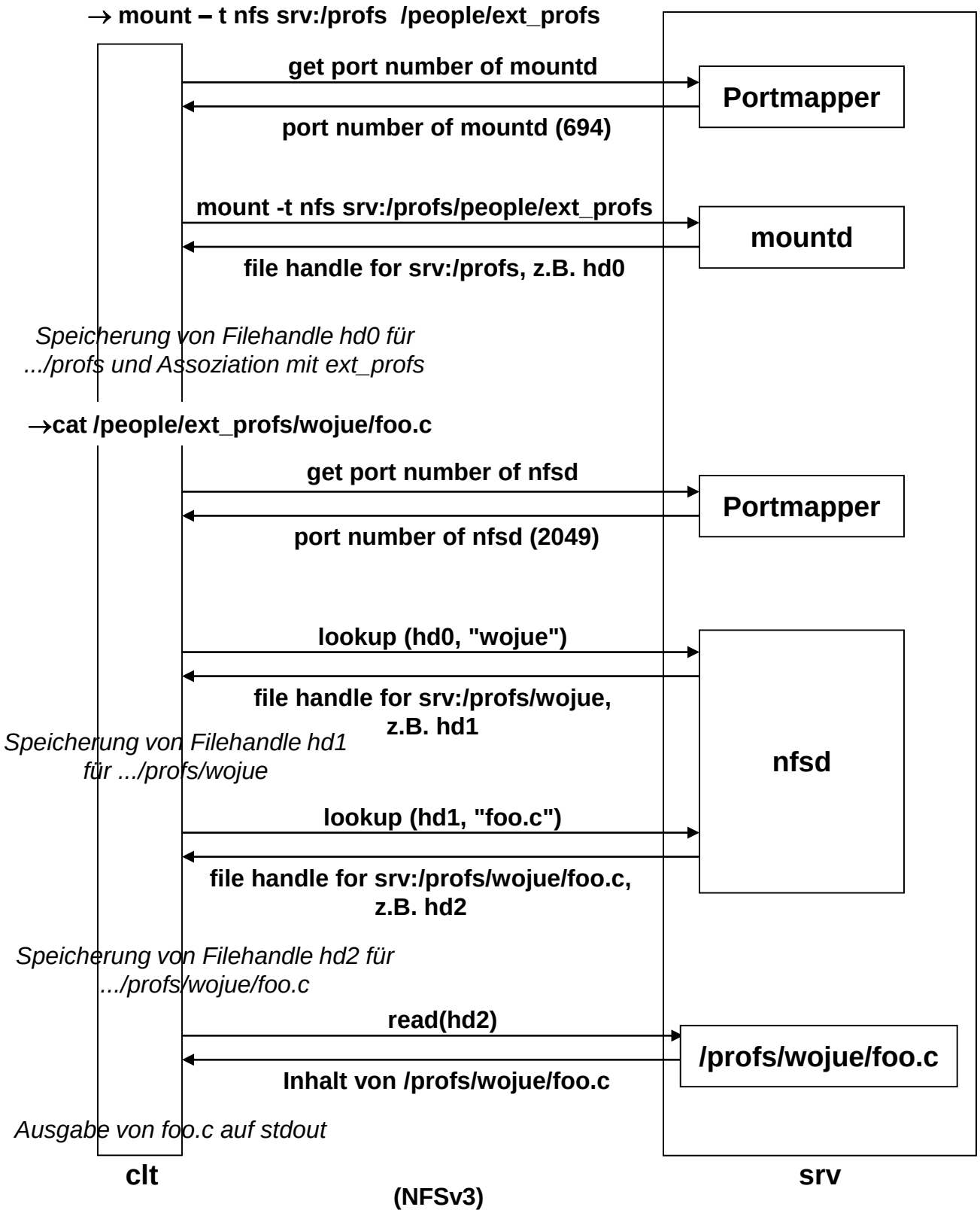
// in /people Mount Point erzeugen: ext_profs
mkdir ext_profs
// entferntes Dateisystem mounten, Format:
// mount –t nfs Hostname:Pfadname Mountpoint
mount –t nfs srv:/profs /people/ext_profs



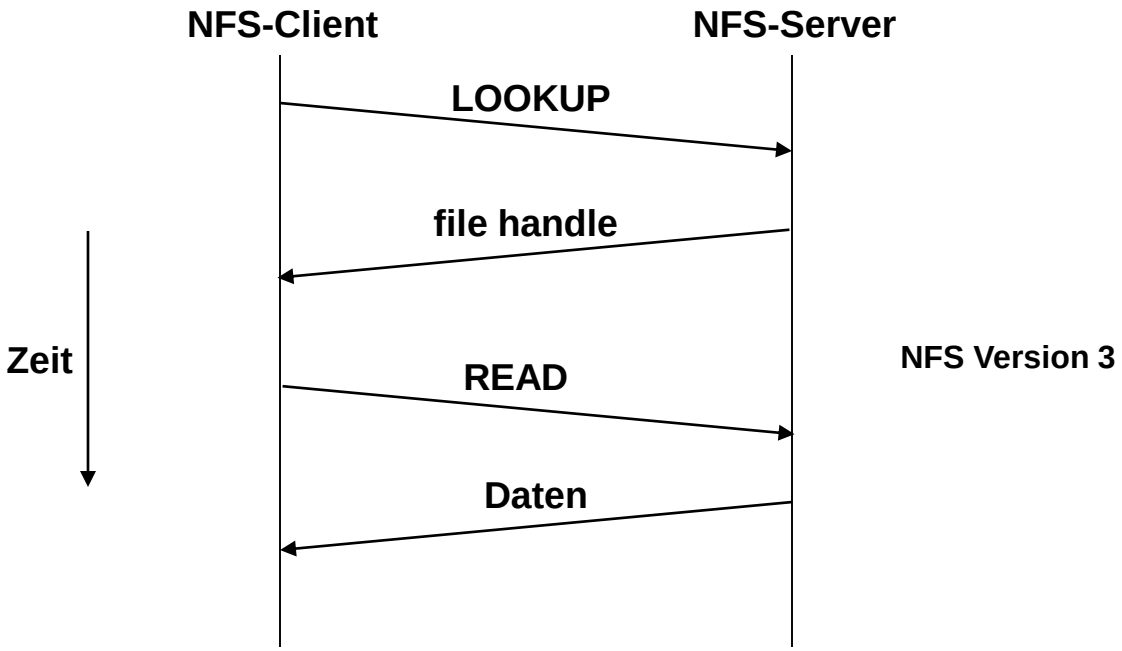


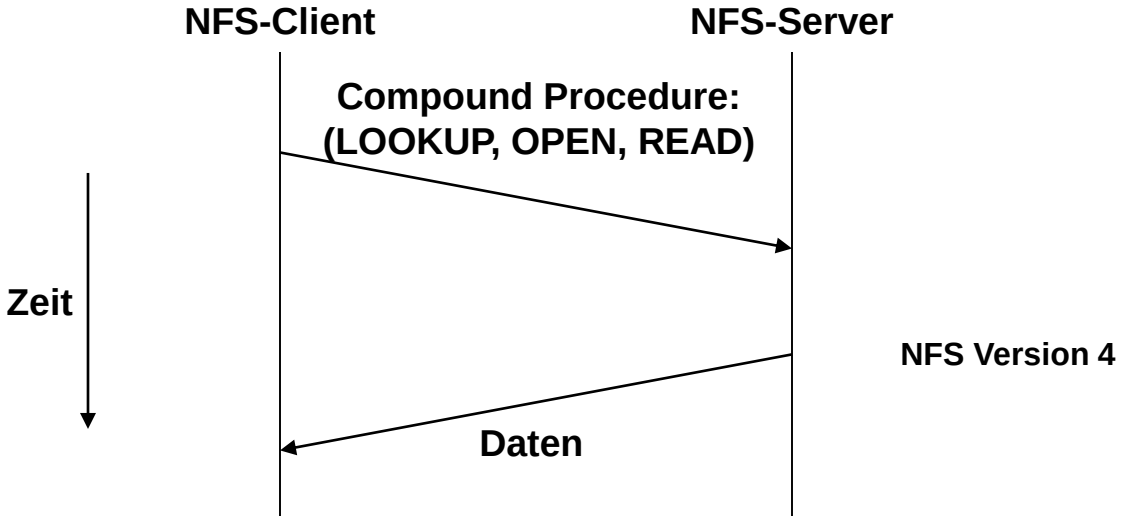
// Zugriff auf foo.c vom Clt aus
cat /people/ext_profs/wojue/foo.c

mount-Kommando: Beispiel



NFS: Lesen einer entfernten Datei

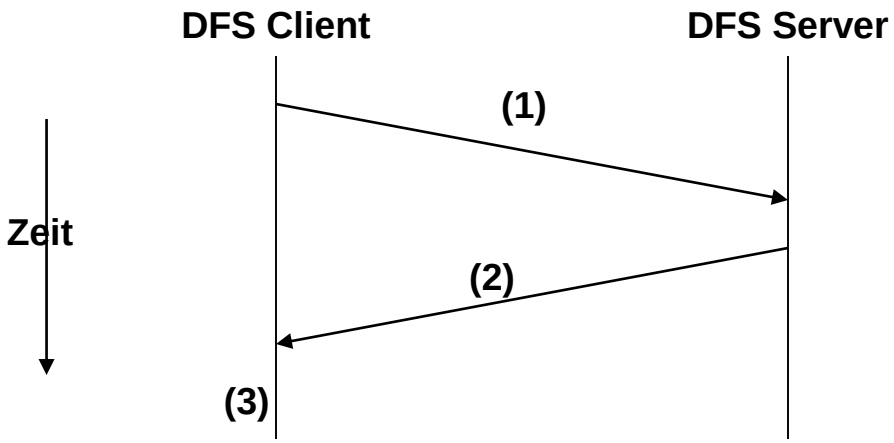




(zusammengesetzte Prozedur)

(Aufsuchen und Lesen einer entfernten Datei)

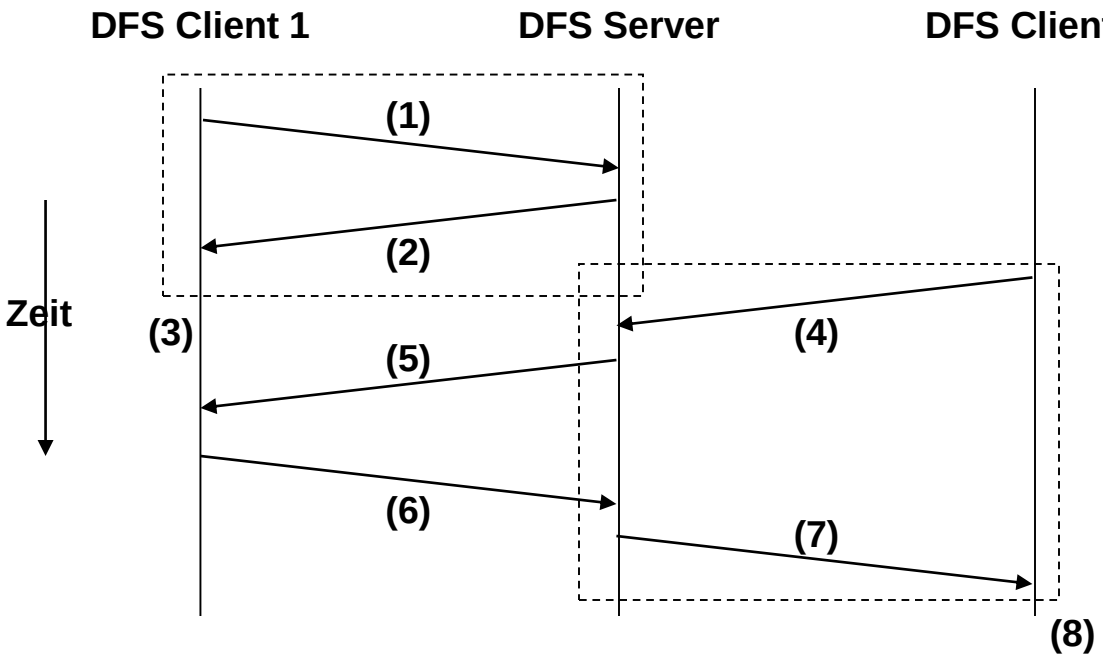
NFS-Kommunikation



(1) **Der Cache Manager des DFS Clients schickt ein File Access Request an den File Exporter des DFS Servers (über ein DCE RPC Request).**

(2) **Der File Exporter schickt die angeforderte Datei über ein DCE RPC Reply an den Client.**

(3) **Der Client bearbeitet die erhaltene Datei. Die geänderte Datei wird *nicht* spontan an den Server zurückgeschickt.**



(1) Client 1 schickt ein File Access Request auf eine Datei an den Server.

(2) Der Server schickt die angefordete Datei zusammen mit einem Token (exklusiver Besitz) an Client 1.

(3) Client 1 bearbeitet die Datei.

(4) Client 2 schickt ein File Access Request auf dieselbe Datei an den Server. Dieser stellt fest, dass die Datei von Client 1 bearbeitet wird.

(5) Der Server schickt Client 1 eine Token-Entzugsmeldung für diese Datei und entzieht ihm dadurch die Zugriffsberechtigung.

(6) Client 1 schickt daraufhin die geänderte Datei und das Token an den Server zurück, spätestens nach einer sog. Verfallszeit im Minutenbereich. Der Server ändert nötigenfalls seine lokale Kopie der Datei.

(7) Der Server schickt die angeforderte (durch Client 1 geänderte) Datei zusammen mit einem Token an Client 2.

(8) Client 2 bearbeitet die Datei.

DFS File Access Model

DFS File Sharing Model