Implementierung eines Dateisystems für den transparenten Zugriff auf ein Versionskontrollsystem

Präsentation zur Bachelorarbeit

Jens M. Nödler

Betreut durch Prof. Dr. Grabowski
Institut für Informatik
Georg-August-Universität Göttingen

20. Dezember 2005

Motivation

Motivation

Intuitiver Zugriff auf Versionskontrollsysteme mittels Dateisystem

Vorteile für Anwender:

- Produktivere Gruppenarbeit an gemeinsamen Daten,
- Versionierung der Daten,
- Zentrale Datenspeicherung,
- Netzwerkzugriff auf entfernte Daten.

Versionskontrolle
Eigenschaften der Versionskontrolle

- Gruppenarbeit mit gemeinsamen Daten ohne Konflikte,
- Änderungsverfolgung,
- Pflege mehrerer Versionszweige,
- Wiederherstellen und Vergleichen verschiedener Versionsstände,
- Hauptanwendung: Quelltextverwaltung bei Softwareentwicklung.

Subversion als Implementierung eines zentralen Versionskontrollsystems, welches als Basis für die Implementierung eines Dateisystems für den transparenten Zugriff verwendet wird.

WebDAV-Protokoll

WebDAV: Distributed Authoring and Versioning Protocol for the World Wide Web

WebDAV ist ein HTTP-kompatibles Internetprotokoll, welches das Lesen *und* Schreiben von entfernten Ressourcen ermöglicht.

Zusätzlich erlaubt WebDAV...

1. Eigenschaften (engl. *Properties*) zu Ressourcen hinzufügen und vorhandene Eigenschaften zu ändern und abzufragen,
2. Zusammengehörigkeiten in Kollektionen (engl. *Collections*) festzulegen und abzufragen,
3. Sperren (engl. *Locking*) von Ressourcen,
4. Versionierung von Ressourcen

Neue WebDAV-Methoden

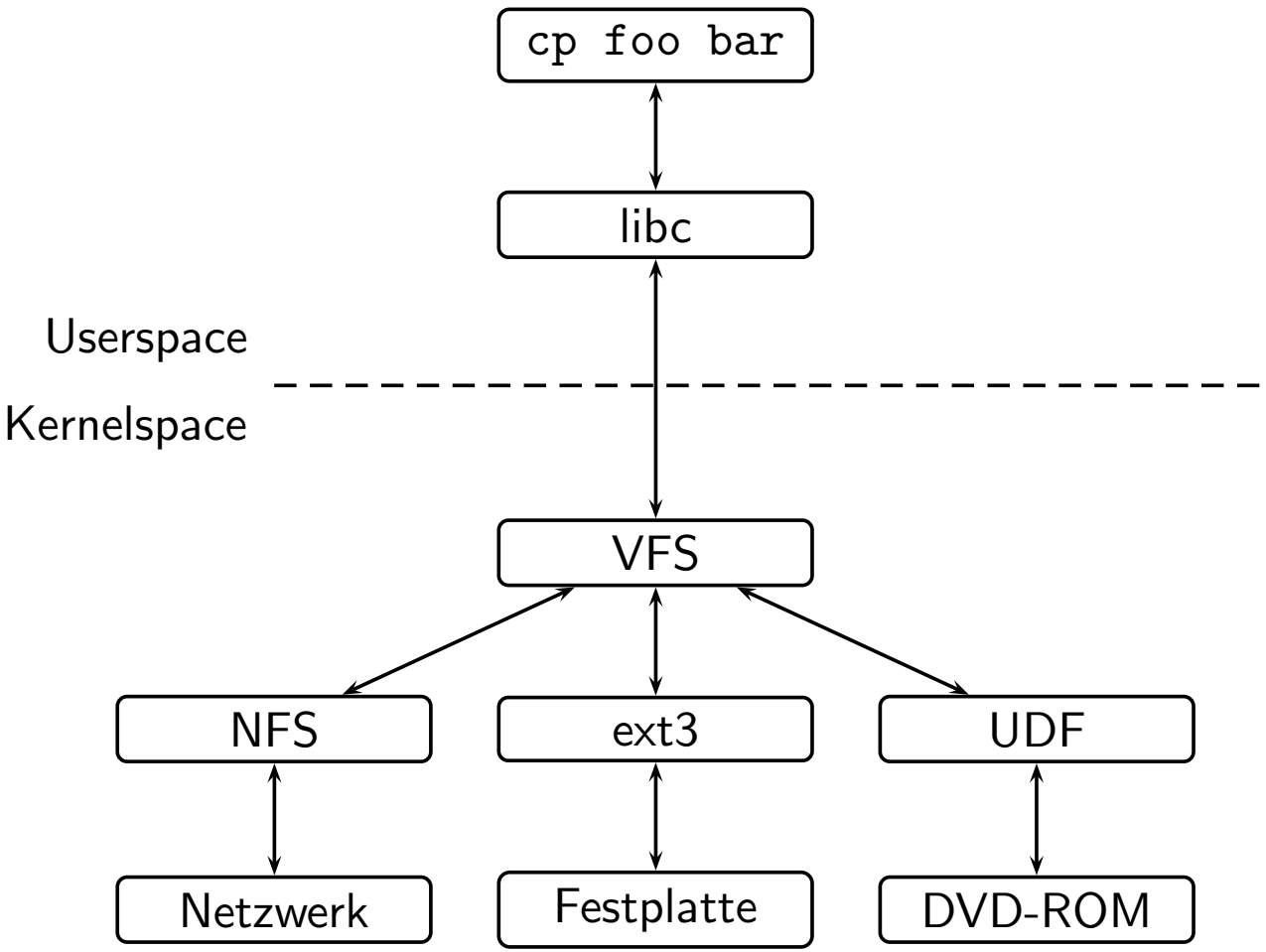
WebDAV ergänzt die bekannten HTTP-Methoden (GET, HEAD, POST, PUT, DELETE) um folgende Methoden:

- `PROPFIND` und `PROPPATCH` – Abfragen und Ändern von Eigenschaften (XML als Datenformat),
- `MKCOL` – Kollektionen anlegen,
- `COPY` und `MOVE` – Kopieren und Verschieben von Ressourcen,
- `LOCK` und `UNLOCK` – Sperren und Entsperren von Ressourcen, um *Schreib*zugriffe Dritter zu verhindern.

Eigenschaften von Dateisystemen unter Linux

- Einteilung in lokale Dateisysteme, Netzwerkdateisysteme und virtuelle Dateisysteme,
- Linux setzt auf monolithische Kernel, daher sind auch Dateisysteme Teil dieses Kernels. Konsequenzen:
  - Zwingende Verwendung von *GNU C* als Programmiersprache,
  - Kein Zugriff auf Bibliotheken (wie *libc*),
  - Keine Speicherschutzmechanismen.

Konzept des Virtual Filesystem (VFS)



Virtual Filesystem: Schnittstellen

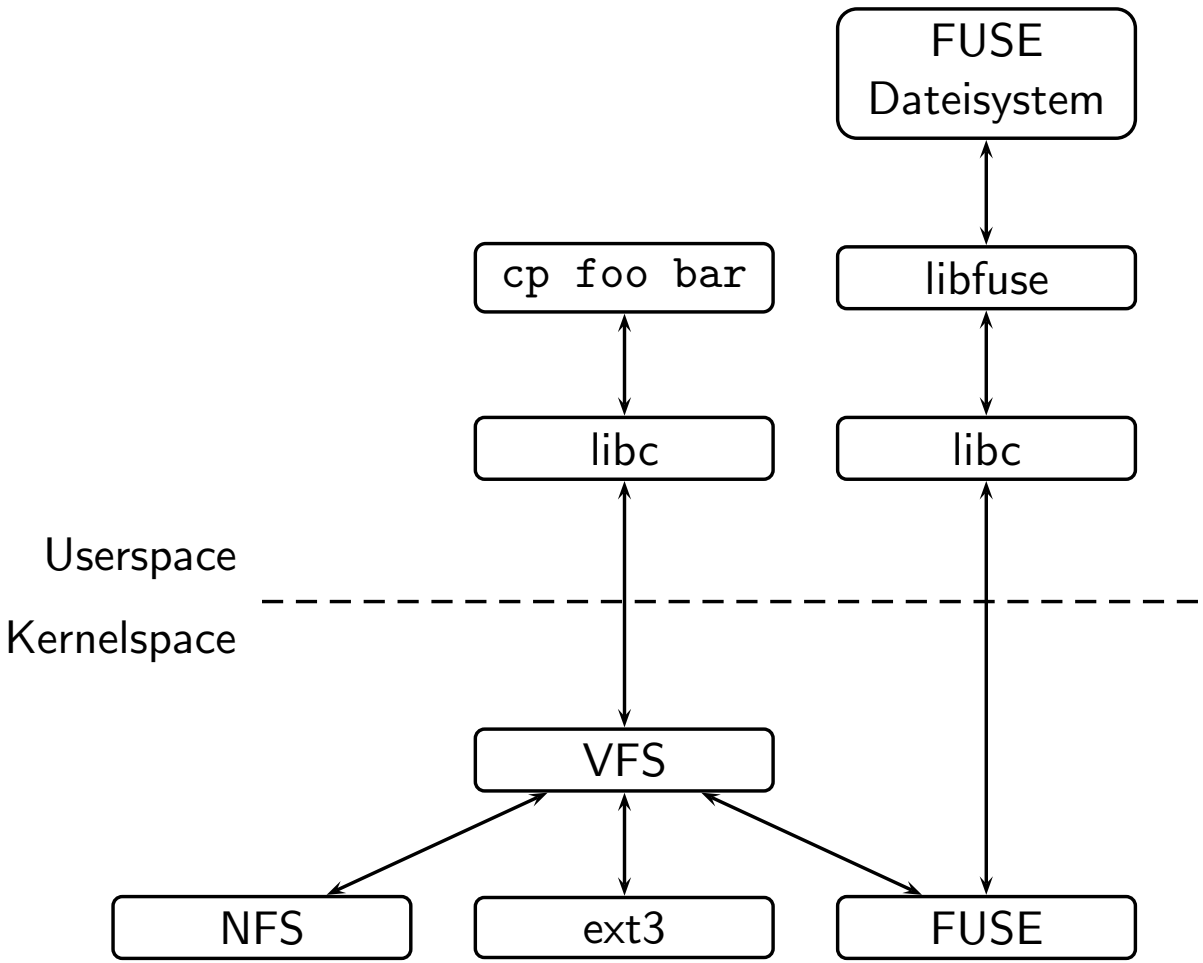
Das Virtual Filesystem bietet nicht nur für Applikationen eine einheitliche Schnittstelle, sondern auch die Dateisysteme des VFS müssen sich an Standards halten.

Das VFS definiert Datenstrukturen, die ein Dateisystem implementieren muss, um mit dem VFS kompatibel zu sein:

1. *Superblock* – eingebundenes Dateisystem,
2. *Inode* (Index Node) – eine Datei des Dateisystems,
3. *Dentry* (Directory Entry) – Teil eines Pfades repräsentiert,
4. *File* – von einem Prozess geöffnete Datei.

Dateisysteme implementieren die Callback-Methoden der `*_operations`-Datenstrukturen.

Konzept des Filesystem in Userspace (FUSE)



FUSE: Aufbau und Vorteile

FUSE-Framework besteht aus drei Komponenten:

1. Kernel-VFS-Modul,
2. Userspace-Bibliothek *libfuse*,
3. Hilfsprogramm zum Ein-/Aushängen von Dateisystemen (*fusermount*).

Vorteile von FUSE-basierten Dateisystemen gegenüber innerhalb des Kernels implementierten Dateisystemen:

- Freie Wahl der Programmiersprache,
- Benutzung von Standard-C-Bibliotheken und anderen Benutzermodusbibliotheken,
- Speicherschutzmechanismus des Benutzermodus,
- Framework-Charakter vereinfacht die Implementierung.

Grundgerüst eines FUSE-Dateisystems in C

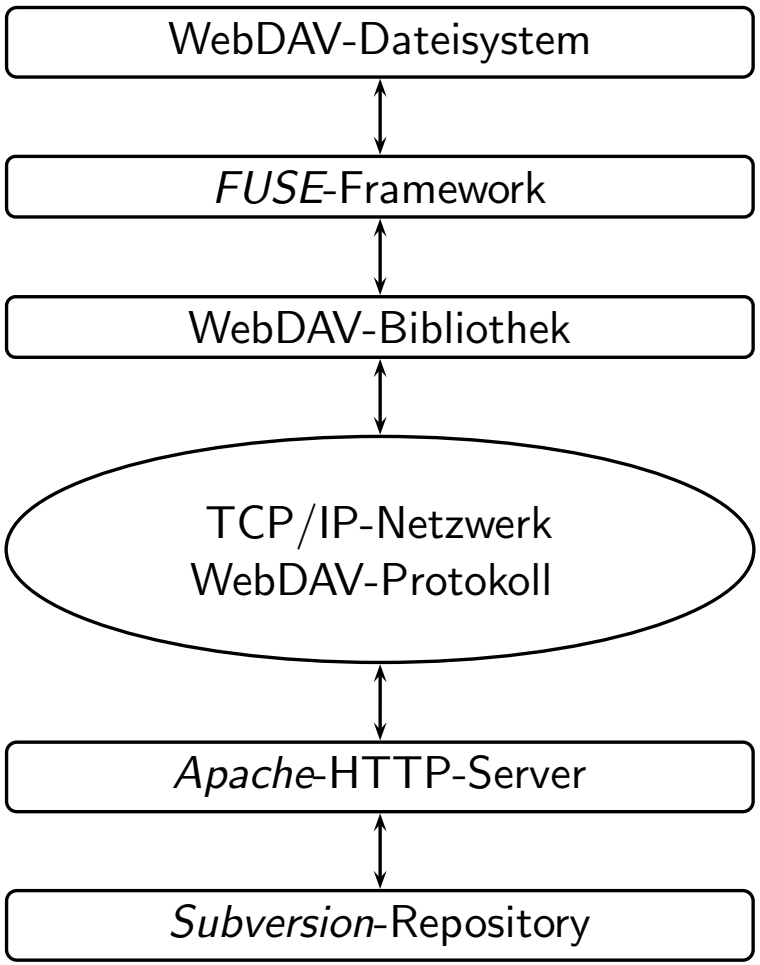
```
1  #define FUSE_USE_VERSION 22
2  #include <fuse.h>
3
4  static int myfilesystem_getattr(
5      const char *path, struct stat *stat) {...}
6  static int myfilesystem_open(
7      const char *path, struct fuse_file_info *fi) {...}
8
9  static struct fuse_operations myfs_operations = {
10     .getattr = myfilesystem_getattr,
11     .open    = myfilesystem_open,
12 };
13
14 int main(int argc, char *argv[]) {
15     return fuse_main(argc, argv, &myfs_operations);
16 }
```

Anforderungen

...an ein Dateisystem für den transparenten Zugriff auf ein Versionskontrollsystem

- Dateisystemgleicher Zugriff auf das Versionskontrollsystem,
- Automatische Erzeugung einer neuen Version bei Änderungen,
- Effektiver Zugriff auf alle Versionen der Daten,
- Schutz vor gegenseitigem Überschreiben von Daten bei Gruppenarbeit,
- Netzwerkzugriff auf das Versionskontrollsystem.

Architektur



Implementierung: Architekturentscheidungen

- Benutzung von C als Programmiersprache
- C-Bibliothek *Neon* für den WebDAV-Zugriff
- GNOME C-Bibliothek *glib* für Standardaufgaben (zB Hash)
- Modularer Aufbau:
  - FUSE-Callback-Methoden (`wdfs-main.c`)
  - WebDAV-Verbindungsaufbau und Locking (`webdav.c`)
  - Dateiattributzwischenspeicher (`cache.c`)
  - Subversion-Funktionen (`svn.c`)

Abbildung Dateisystemoperationen auf WebDAV-Methoden

Dateisystemoperation	WebDAV-Methode
getattr()	PROPFIND
readdir()	PROPFIND
open()	GET
read()	--
write()	--
release()	PUT
truncate()	GET, PUT
mknod()	PUT
mkdir()	MKCOL
unlink()	DELETE
rmdir()	DELETE
rename()	MOVE

Strategien für das Sperren von Dateien

Ziele:

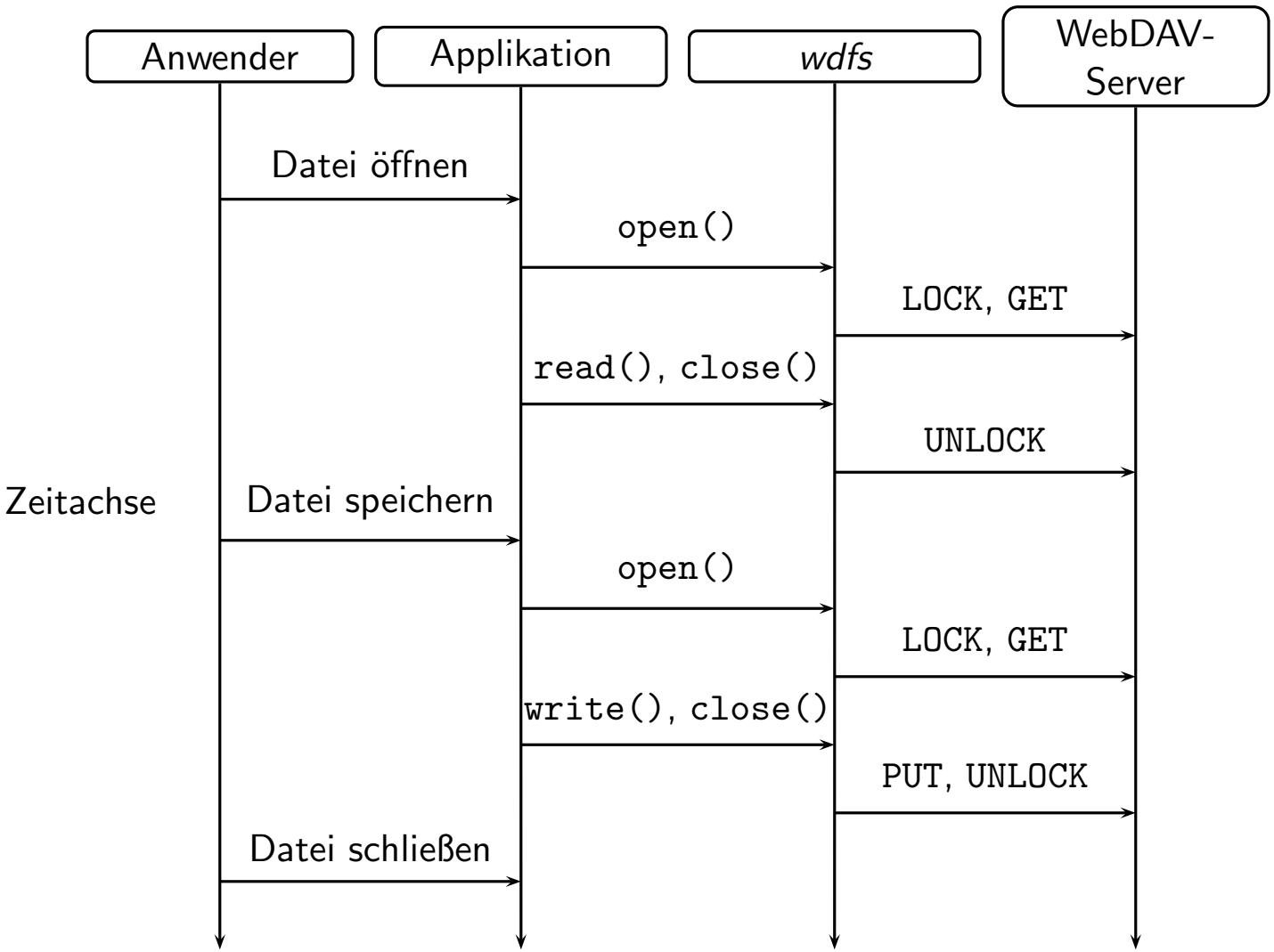
- Datenkonsistenz durch serialisierte Schreibzugriffe,
- Überschreiben von Daten durch Dritte verhindern.

Mittel:

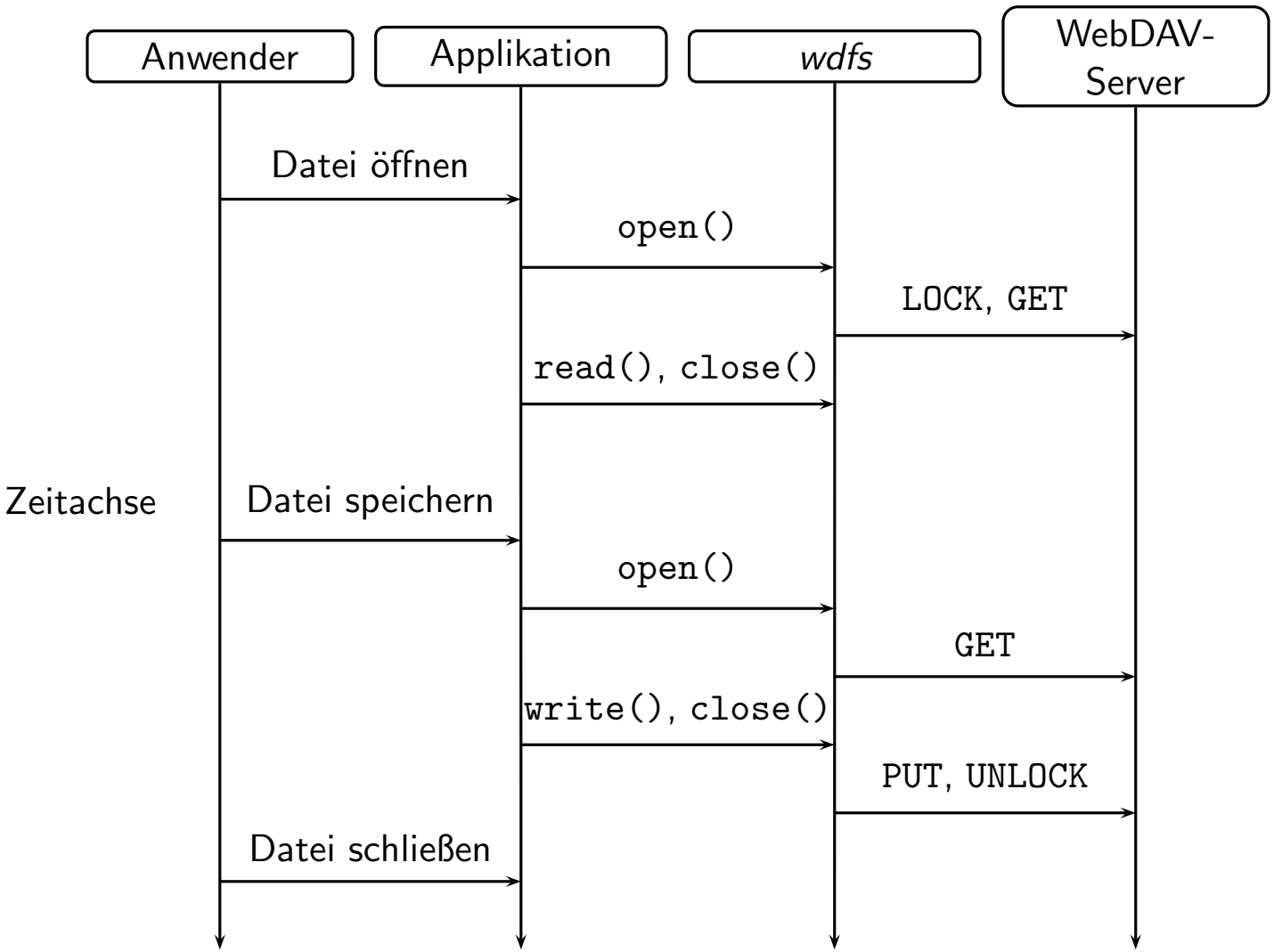
- WebDAV-Locks bieten Schutz vor dem *schreibenden* Zugriff Dritter,
- Sperren besitzen eine bestimmte Lebensdauer (n Sekunden oder unendlich),
- Stehlen von Sperren durch Dritte ist möglich.

Das Dateisystem implementiert drei Locking-Strategien.

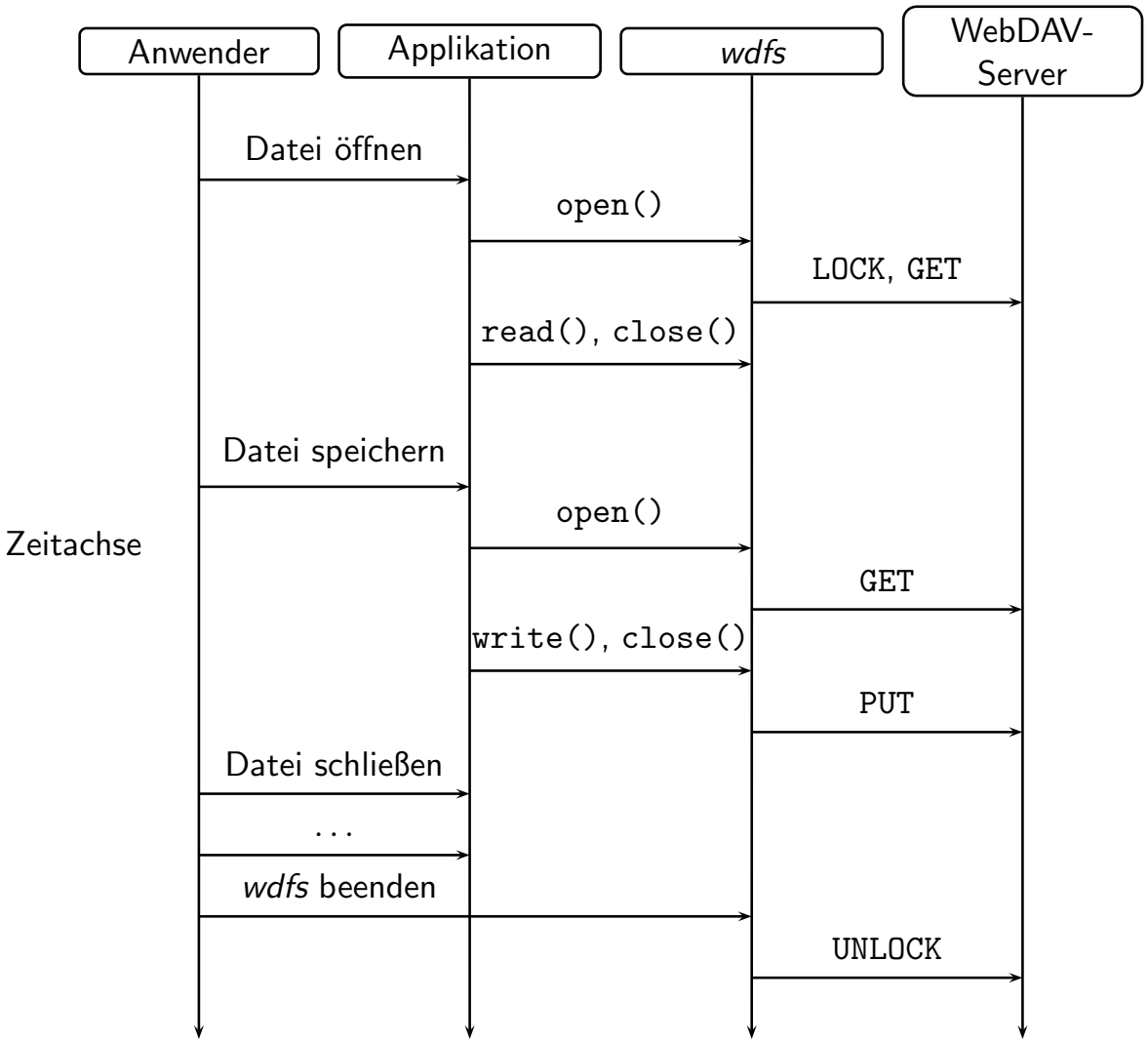
Strategie 1 für das Sperren von Dateien

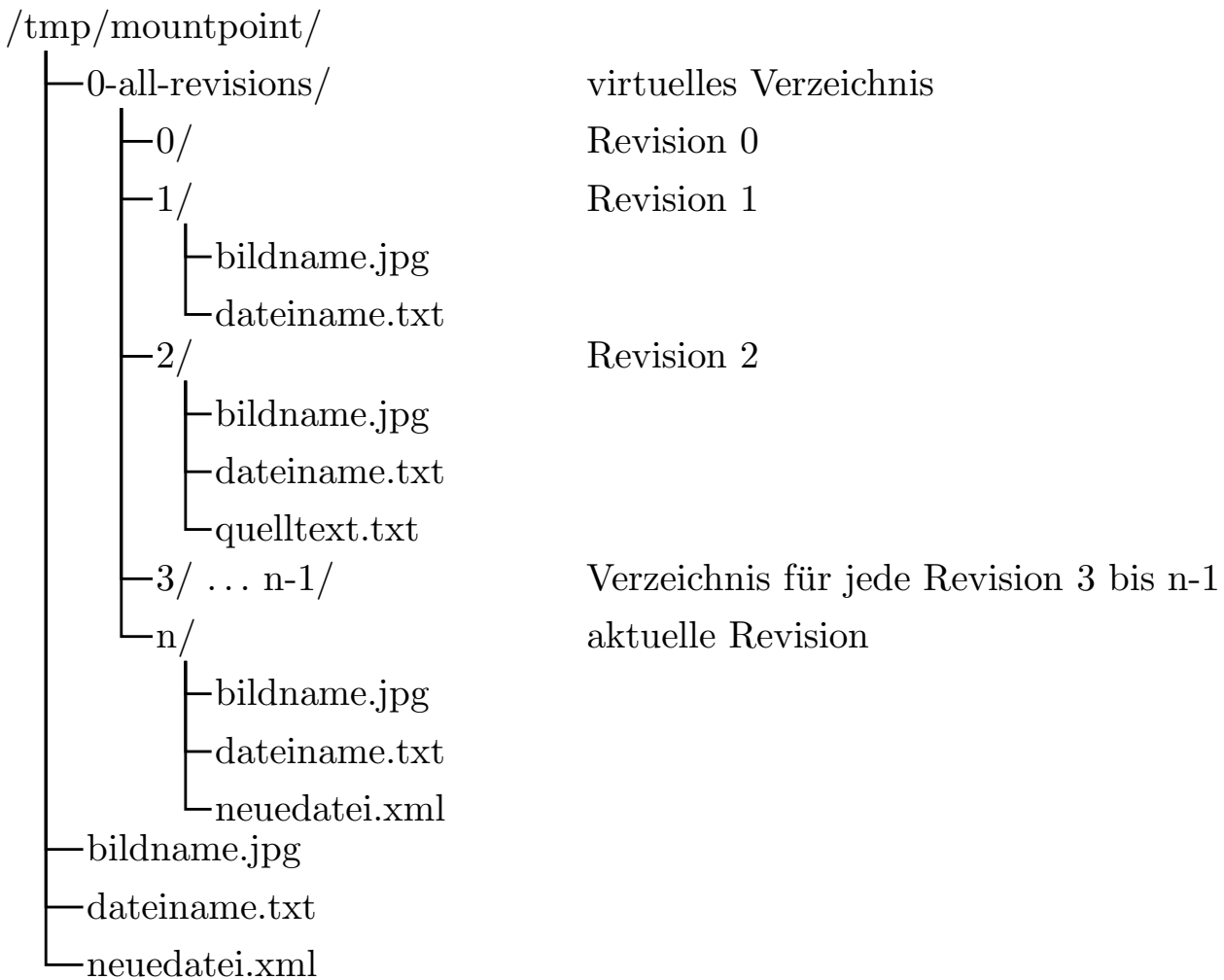


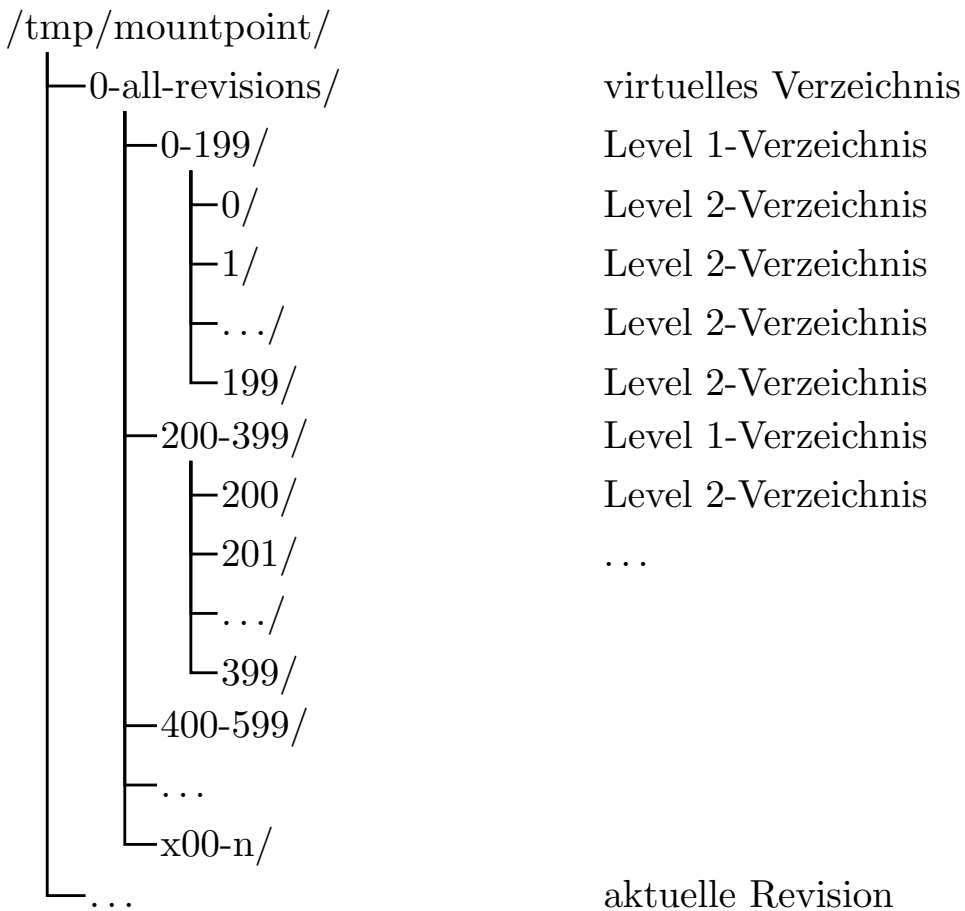
Strategie 2 für das Sperren von Dateien



Strategie 3 für das Sperren von Dateien




/tmp/mountpoint/
0-all-revisions/
virtuelles Verzeichnis
0/
Revision 0
1/
Revision 1
bildname.jpg
dateiname.txt
2/
Revision 2
bildname.jpg
dateiname.txt
quelltext.txt
3/ ... n-1/
Verzeichnis für jede Revision 3 bis n-1
n/
aktuelle Revision
bildname.jpg
dateiname.txt
neuedatei.xml
bildname.jpg
dateiname.txt
neuedatei.xml


/tmp/mountpoint/
0-all-revisions/
virtuelles Verzeichnis
0-199/
Level 1-Verzeichnis
0/
Level 2-Verzeichnis
1/
Level 2-Verzeichnis
.../
Level 2-Verzeichnis
199/
Level 2-Verzeichnis
200-399/
Level 1-Verzeichnis
200/
Level 2-Verzeichnis
201/
...
.../
399/
400-599/
...
x00-n/
...
aktuelle Revision

Einsatzmöglichkeiten des Dateisystems

- Dateisystemzugriff auf Subversion (Anwendergruppen, die bisher nichts mit Versionskontrolle zu tun hatten),
- „Dateisystem mit Gedächtnis“ (Für Anwender, die alle Änderungen festhalten wollen. Beispiel: Home-Verzeichnis),
- SMB-Export des Dateisystems für den Zugriff anderer Betriebssysteme,
- Bearbeiten einzelner Dateien eines Subversions-Repositorys.

Zusammenfassung

- Grundlegende Dateisystemoperationen,
- Zugriff auf alle Revisionen eines Subversion-Repositorys,
- Sperren von Dateien,
- Performance ist akzeptabel,
- Nutzung der Versionskontrolle wird erleichtert,
- Neue Einsatzgebiete für Versionskontrollsysteme.

Vielen Dank

Vielen Dank für Ihre Aufmerksamkeit!

Diese Präsentation wurde mit der LaTeX-Beamer-Klasse erstellt.
`http://latex-beamer.sourceforge.net`