Work in progress

Das Unix-Dateisystem aus Nutzersicht

Ute Bormann, TI2

2023-10-13

Inhalt

1. Struktur des Dateisystems
2. Arten von Dateien
3. Zugriffsschutz für Dateien

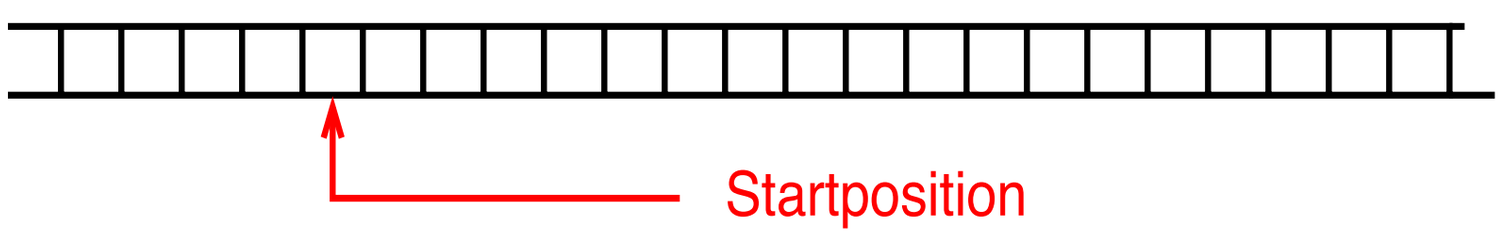
Teil 1:
Struktur des Dateisystems

Das Unix-Dateisystem aus Nutzersicht

- Datei (File) ist langlebiges Datenobjekt
  ⇒ muss vom System verwaltet werden
- Nutzer können darauf zugreifen ⇒ z.B. Shell
- Prozesse können darauf zugreifen ⇒ Systemaufrufe

Das Unix-Dateisystem aus Nutzersicht

- Datei (File) ist langlebiges Datenobjekt
  ⇒ muss vom System verwaltet werden
- Nutzer können darauf zugreifen ⇒ z.B. Shell
- Prozesse können darauf zugreifen ⇒ Systemaufrufe
- Inhalt:
  - C-Programm, ASCII-Text, Datensätze, Foto,...
    ⇒ viele denkbare interne Strukturen
    ⇒ für Unix lediglich Folge von Bytes
  - sequentiell zugreifbar (Startposition angebbar)



- Dateinamen
  - Muss Datei eindeutig benennen
  - Im Prinzip beliebige Zeichenfolge
  - Außer: / und Nullbyte haben Sonderbedeutung

    `Dies ist ein Dateiname` aber SP Sonderbedeutung in Shell !

- Dateinamen
  - Muss Datei eindeutig benennen
  - Im Prinzip beliebige Zeichenfolge
  - Außer: / und Nullbyte haben Sonderbedeutung

    `Dies-ist-ein-Dateiname` Besser!

- Dateinamen
  - Muss Datei eindeutig benennen
  - Im Prinzip beliebige Zeichenfolge
  - Außer: / und Nullbyte haben Sonderbedeutung

    `Dies-ist-ein-Dateiname` Besser!
  - Konvention: Typ des Inhalts wird „angehängt“

    `bla.c  bla.o  bla.tex  bla.pdf`

- Dateinamen
  - Muss Datei eindeutig benennen
  - Im Prinzip beliebige Zeichenfolge
  - Außer: / und Nullbyte haben Sonderbedeutung

    `Dies-ist-ein-Dateiname` Besser!
  - Konvention: Typ des Inhalts wird „angehängt“

    `bla.c  bla.o  bla.tex  bla.pdf`

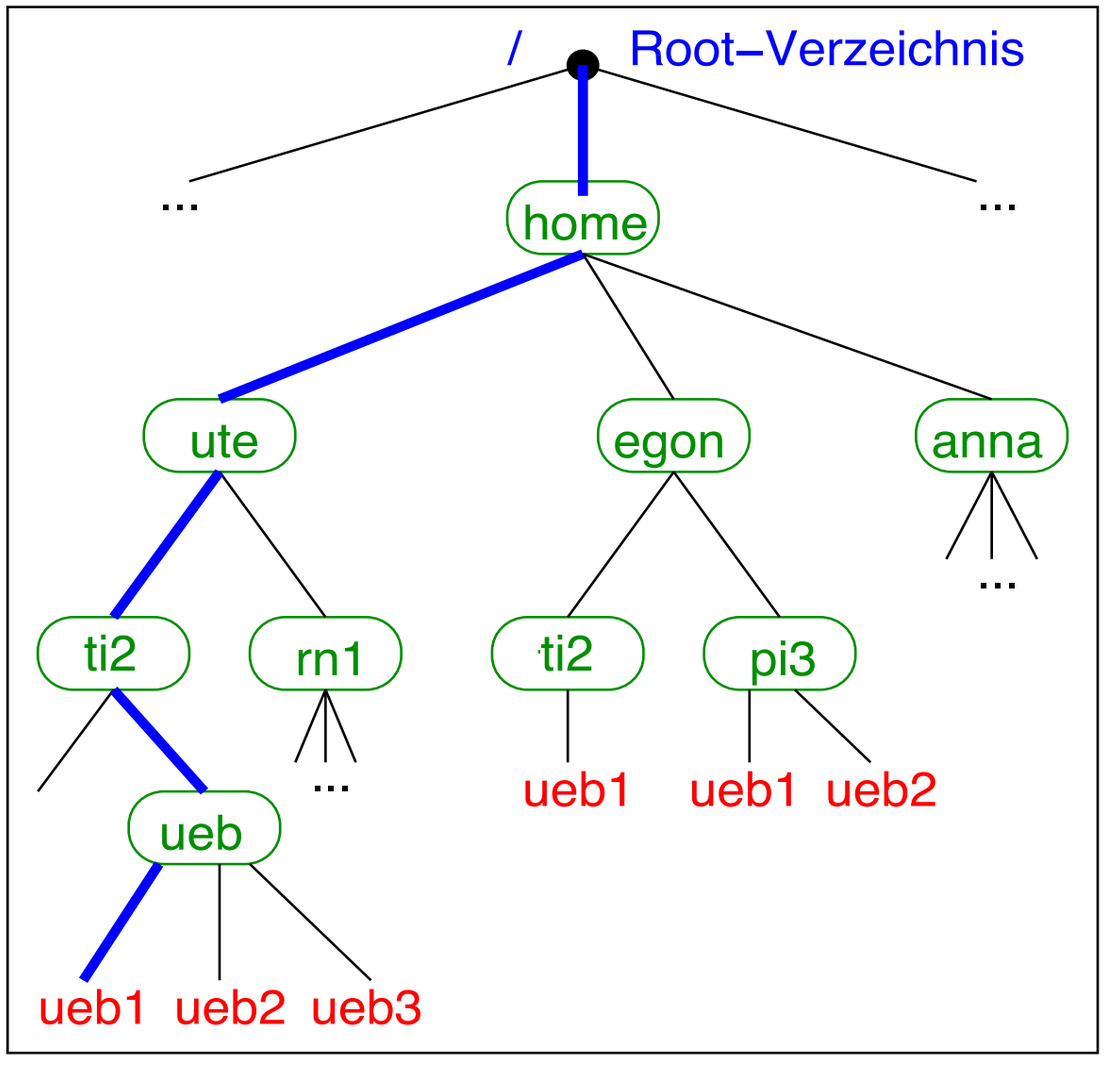
- Arbeiten mit Dateien (siehe „man-Pages“)
  - Erzeugen: (typischerweise mit Editor)
  - Löschen: `rm fn` (`fn` = Filename)
  - Umbenennen: `mv old-fn new-fn`
  - Kopieren: `cp old-fn new-fn`
  - Anzeigen: `cat fn`
  - Anzeigen, seitenweise: `more fn` (bzw. `less fn`)

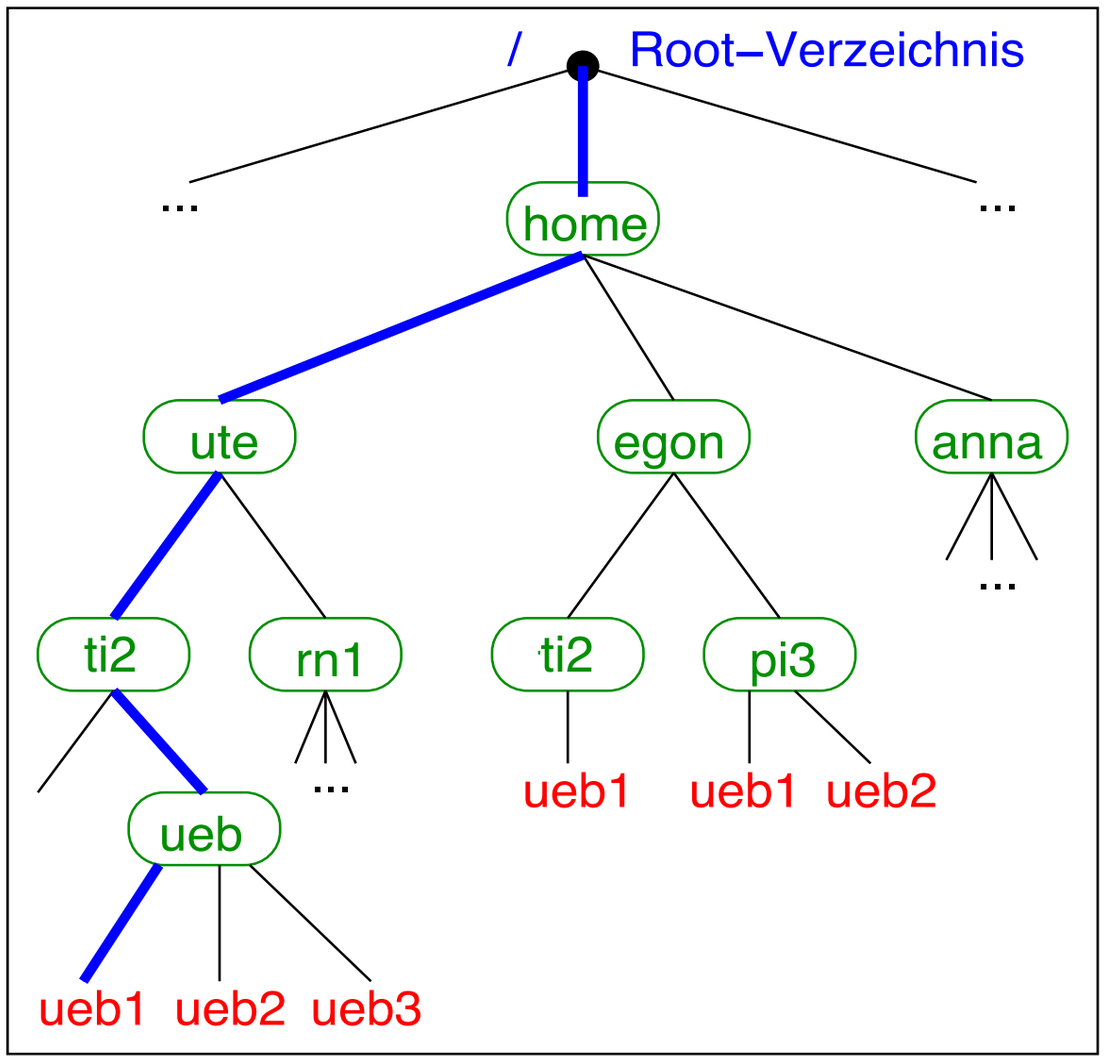
- Strukturierung von Dateisammlungen
  - Dateisammlungen können sehr groß werden

    ⇒ sollten strukturiert werden
  - Dateien werden in Verzeichnisse (Kataloge, Directories) eingeordnet

    ⇒ „thematische“ Gliederung möglich
  - Verzeichnisse werden hierarchisch angeordnet

    ⇒ weitergehende Gliederung


/
Root–Verzeichnis
...
home
...
ute
egon
anna
...
ti2
rn1
ti2
pi3
...
ueb1
ueb1
ueb2
ueb
ueb1
ueb2
ueb3



- ueb1 mehrdeutig

  ⇒ nur im Kontext des Verzeichnisses eindeutig

- Vollständige Dateinamen sind Pfadnamen von der Wurzel abwärts

  `/home/ute/ti2/ueb/ueb1`

  `/home/egon/pi3/ueb1`

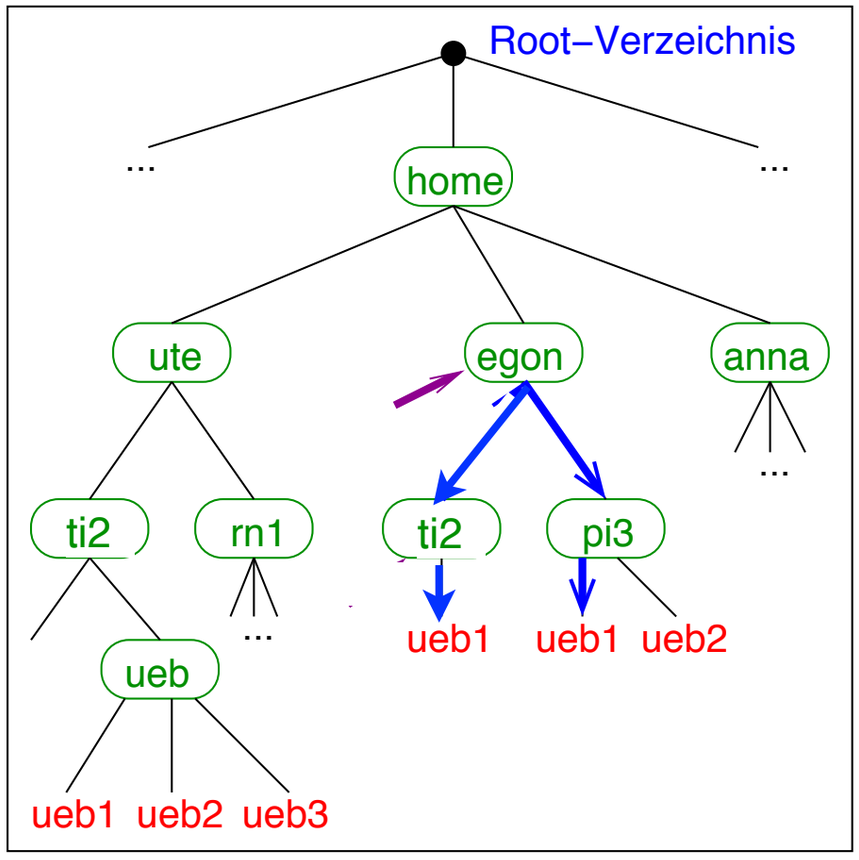
Dateinamen müssen nicht vollständig angegeben werden:

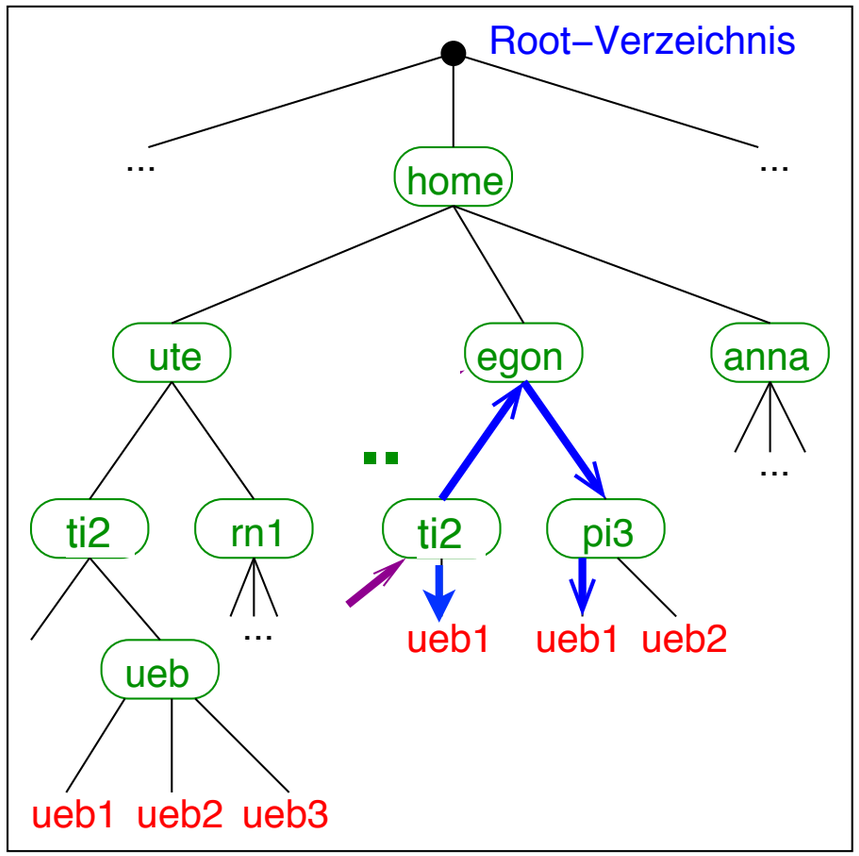
- Im Kontext eines Arbeitsverzeichnisses (Working Directory):

  (/home/egon:)

  `ti2/ueb1`

  `pi3/ueb1`





Dateinamen müssen nicht vollständig angegeben werden:

- Im Kontext eines Arbeitsverzeichnisses (Working Directory):

  (/home/egon:)

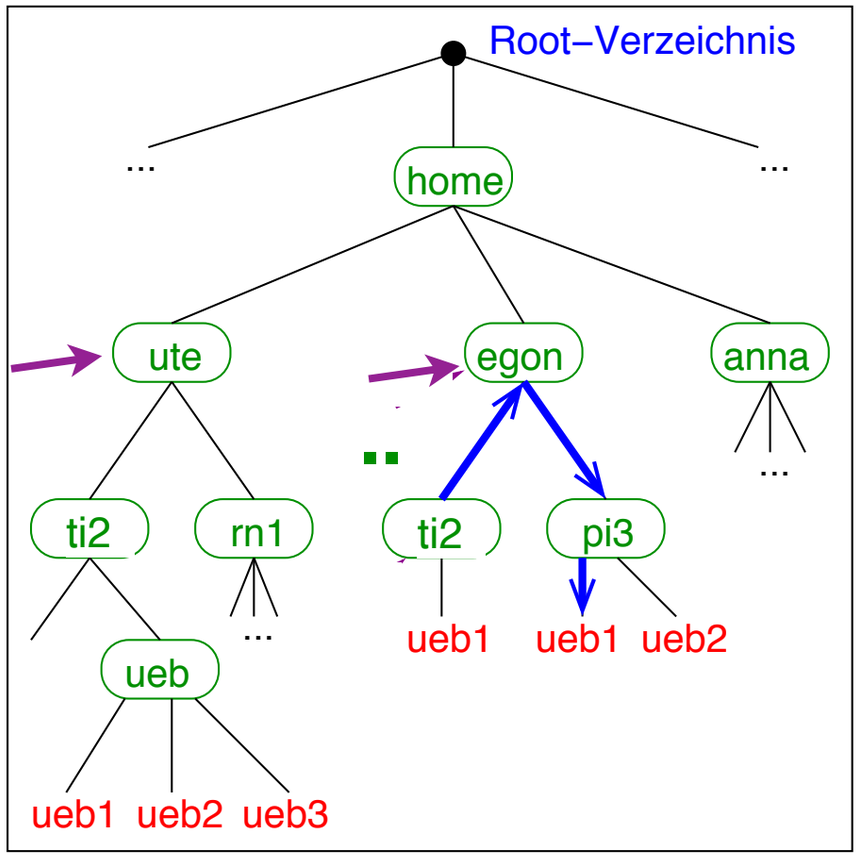
  `ti2/ueb1`

  `pi3/ueb1`

  (/home/egon/ti3:)

  `ueb1`

  `../pi3/ueb1`



Dateinamen müssen nicht vollständig angegeben werden:

- Im Kontext eines Arbeitsverzeichnisses (Working Directory):

  (/home/egon:)

  `ti2/ueb1`

  `pi3/ueb1`

  (/home/egon/ti3:)

  `ueb1`

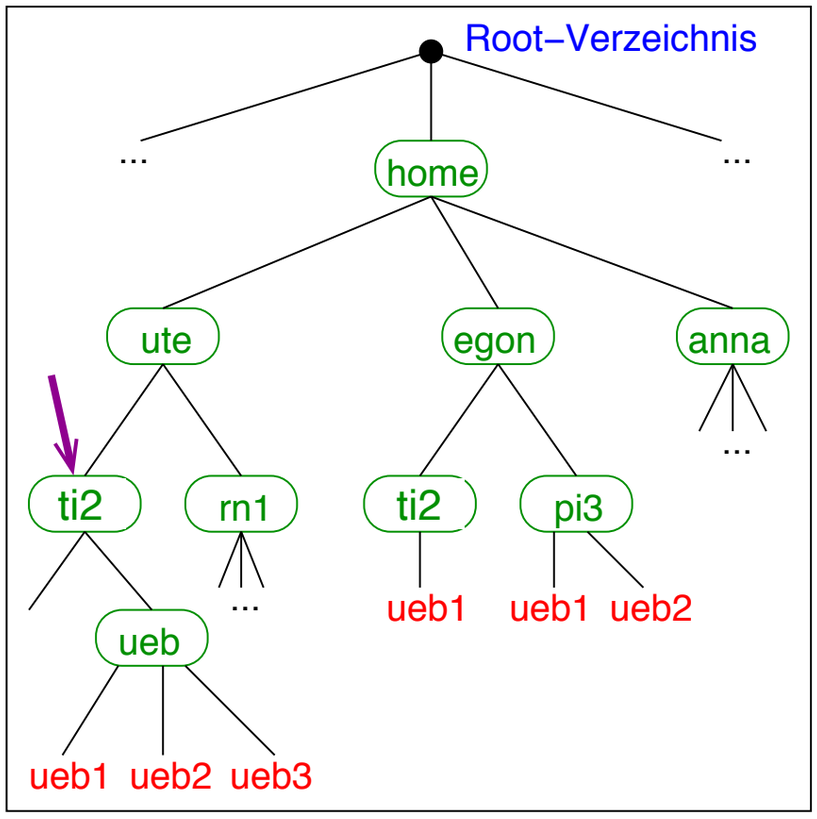
  `../pi3/ueb1`

- Im Kontext eines Heimverzeichnisses (Home Directory) ⇒ Shell-Konvention:

  `~/ti2/ueb/ueb1`

- Im Kontext des Heimverzeichnisses eines anderen:

  `~egon/ti2/ueb1`



- Ausgeben des Working Directory:

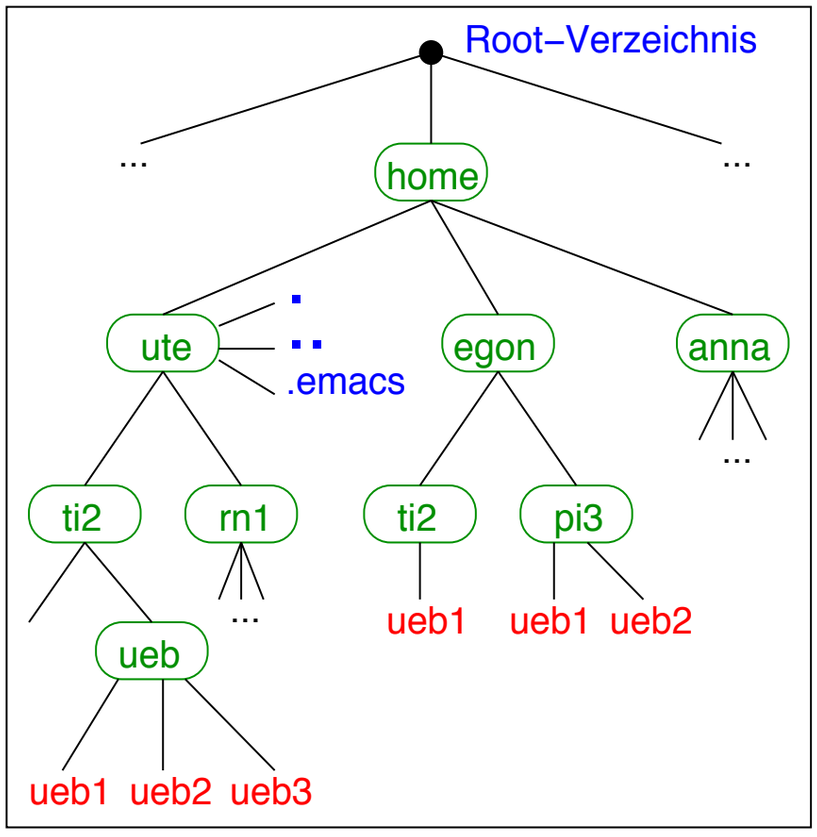
  `pwd` ⇒ `/home/ute/ti2`

- Ändern des Working Directory:

  `cd dir`

- Beispiele:

cd ueb	(/home/ute/ti2/ueb)
cd .	(/home/ute/ti2/ueb)
cd ../..	(/home/ute)
cd ~egon/ti2	(/home/egon/ti2)
cd ~	(/home/ute)
cd /home/ute/ti2	(/home/ute/ti2)



- Ausgeben der Einträge eines Verzeichnisses

  `ls dir`

- Beispiele:

  `ls /home/ute` ⇒ `ti2/ rn1/`

  `ls ~` ⇒ `ti2/ rn1/`

  `ls -a ~` ⇒ `. .. .emacs ti2/ rn1/`

  `ls` (Ausgabe des Working Directory)

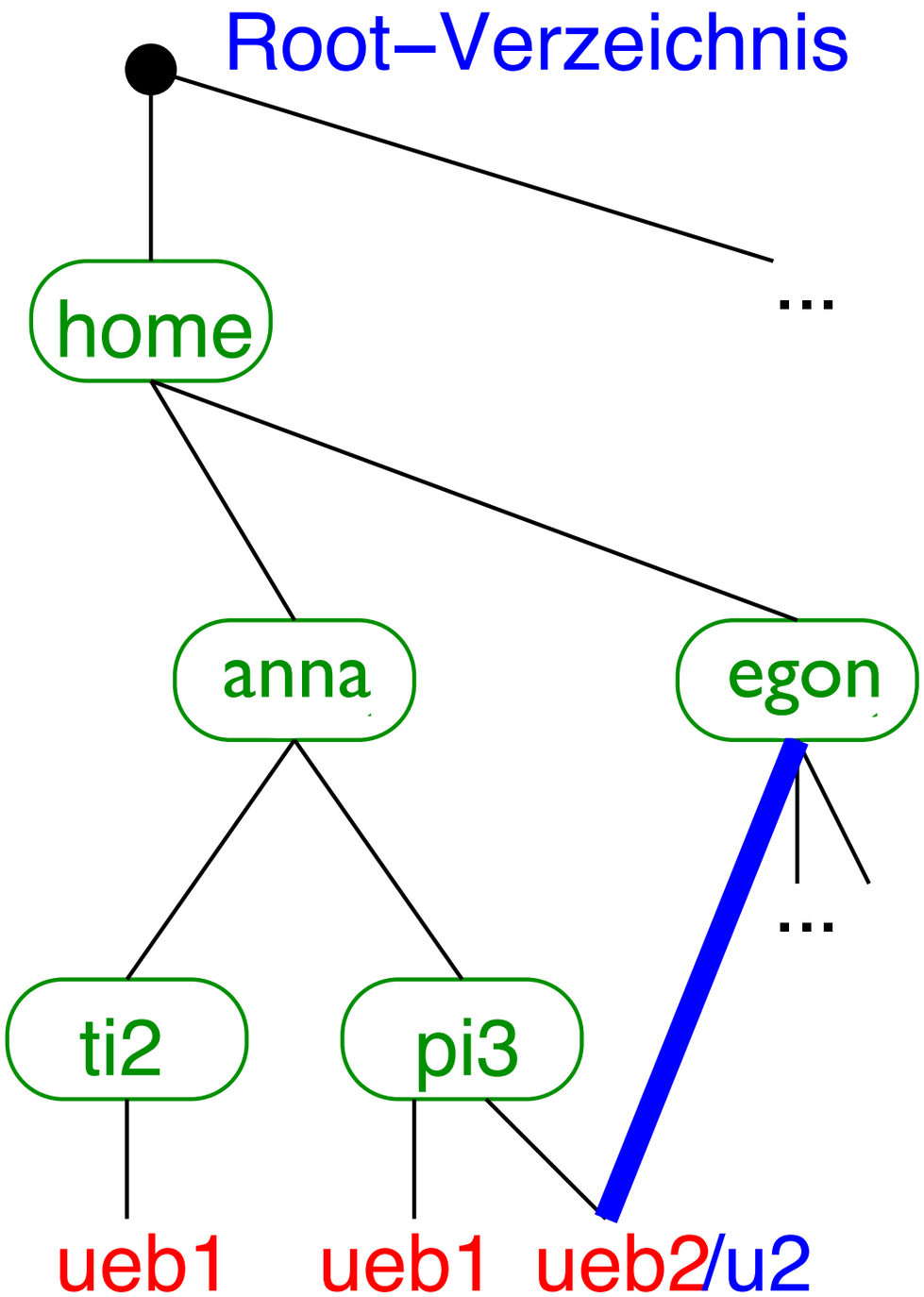
- Erzeugen eines Unterverzeichnisses

  `mkdir dir` (darf noch nicht existieren)

- Löschen eines Unterverzeichnisses

  `rmdir dir` (muss leer sein)

Verweise (Links)

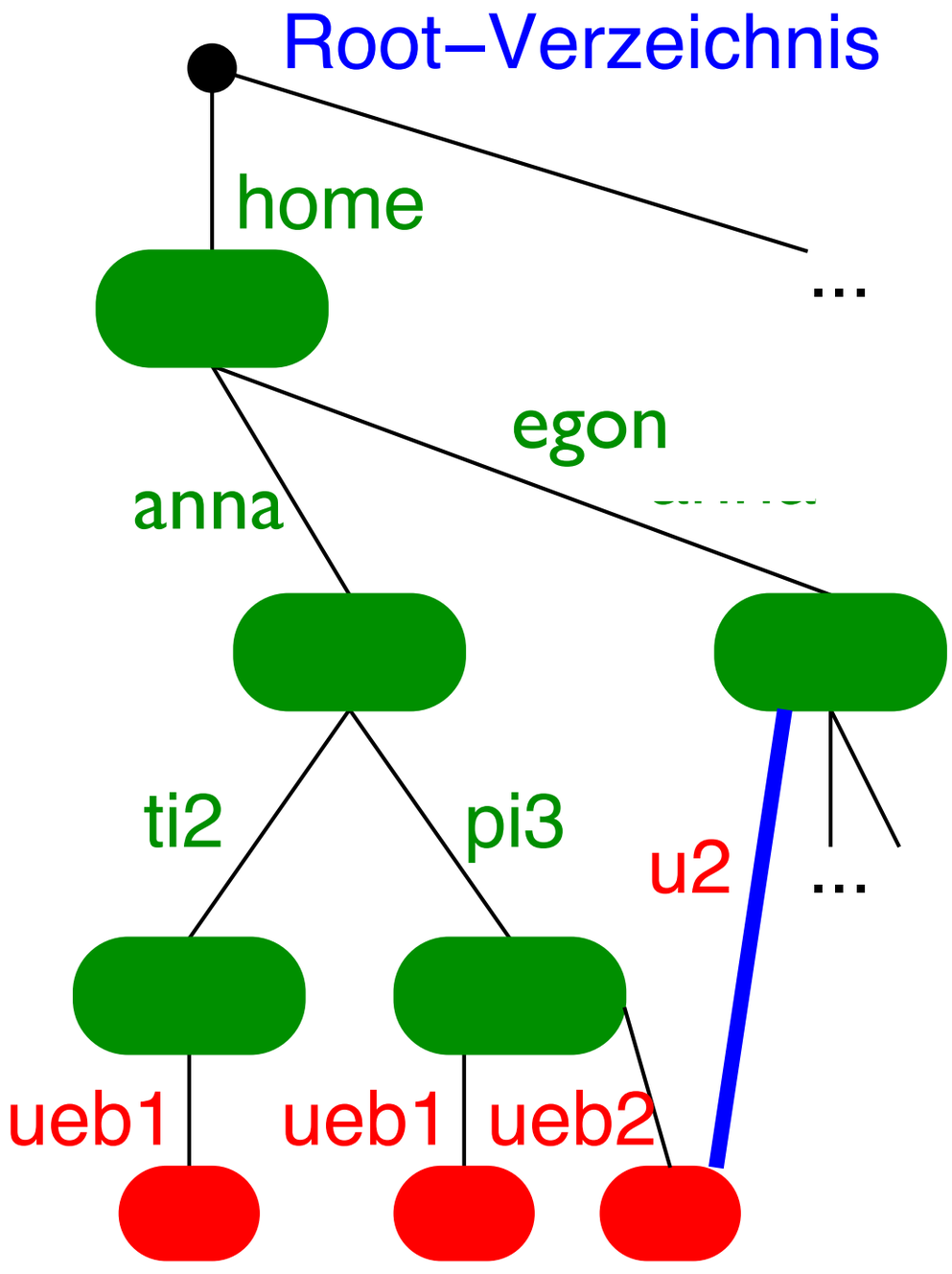


- Datei soll unter mehreren Namen bekannt sein
- Anna und Egon wollen dieselbe Datei im eigenen Kontext benutzen

a) Hard Links

```
ln fn addfn
```

```
ln /home/anna/pi3/ueb2 /home/egon/u2
```



- Zwei Namen für dasselbe Objekt

  ⇒ kein Baum

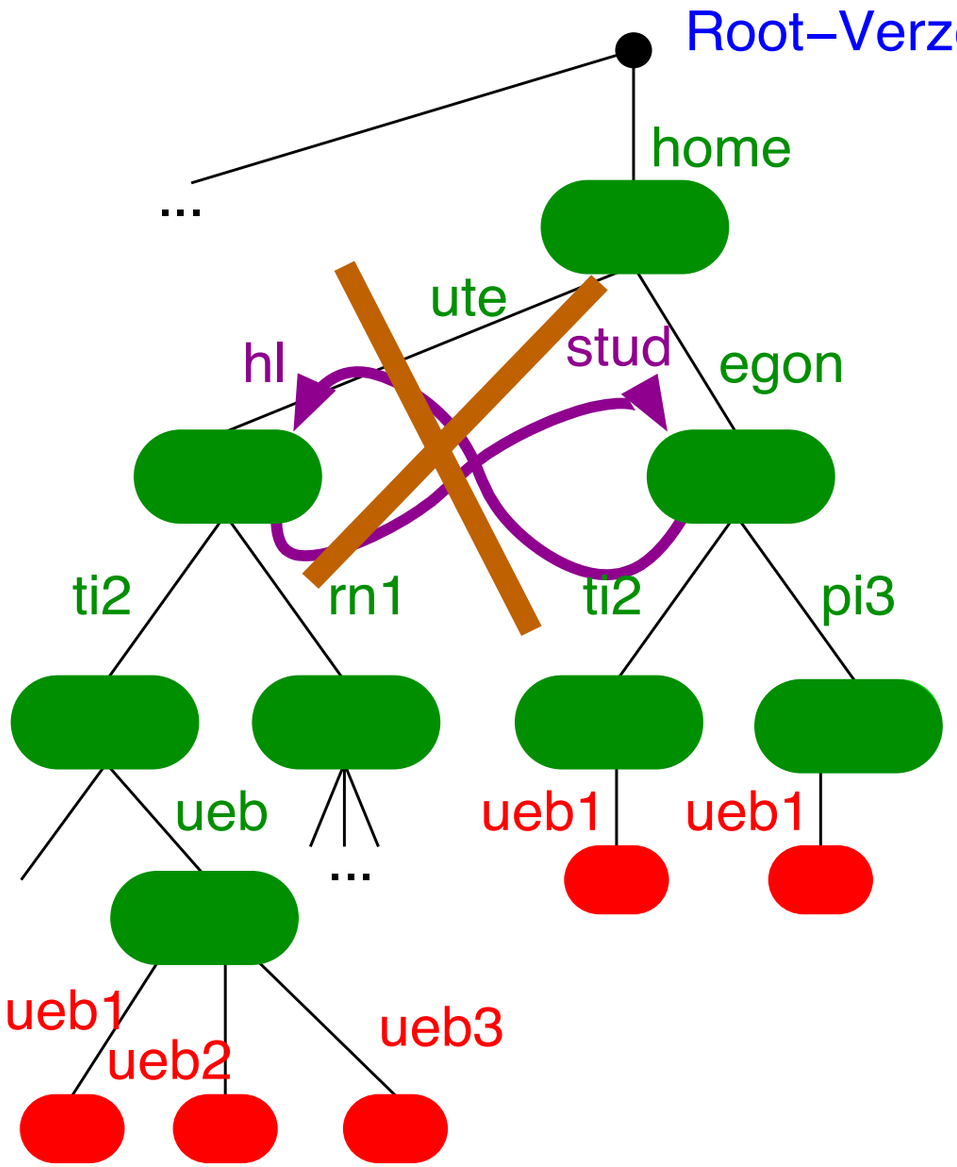
  ⇒ Kanten statt Knoten benannt

- Löschen mit `rm fn`

  ⇒ anderer Name noch da

- Alle Namen gelöscht

  ⇒ Datei nicht mehr zugreifbar



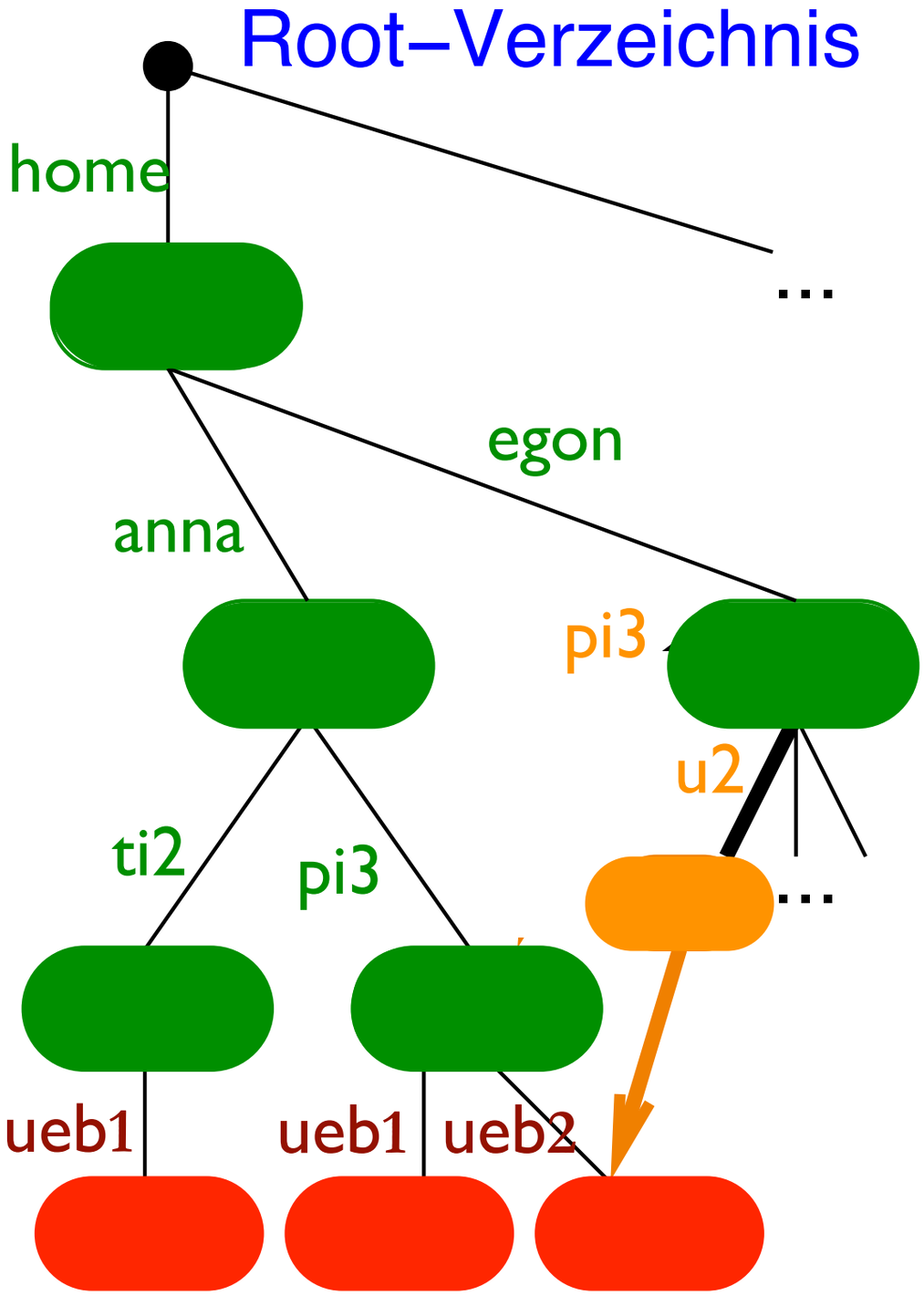
Achtung:
Keine Hard Links auf Verzeichnisse

⇒ unkontrollierte Schleifen vermeiden

~~`ln /home/ute /home/egon/hl`~~

~~`ln /home/egon /home/ute/stud`~~

Dadurch einfaches Traversieren und einfaches Löschen möglich



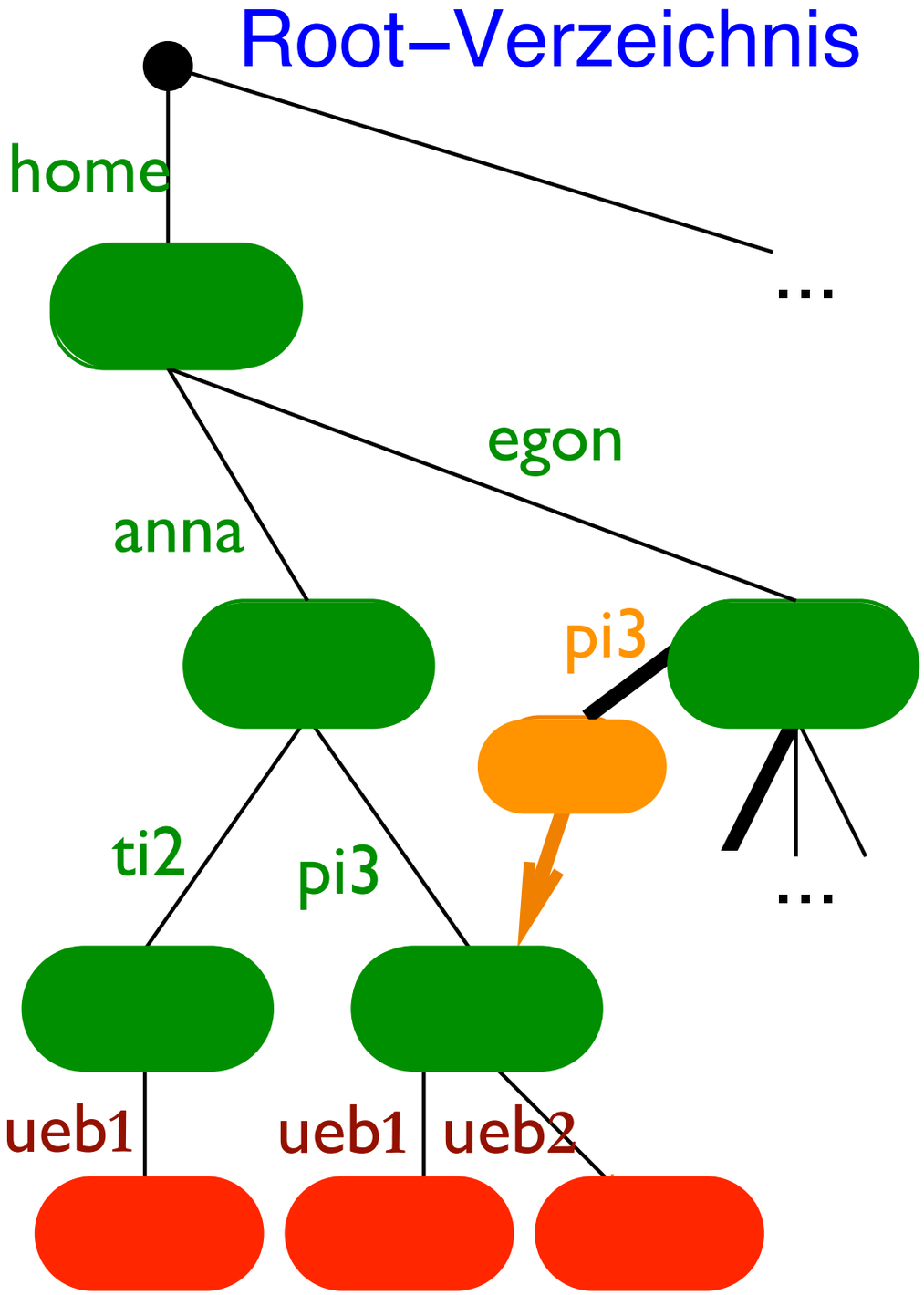
Stattdessen:

b) Symbolic Links

```
ln -s fn linkfn
```

```
ln -s /home/anna/pi2/ueb2 /home/egon/u2
```

- Linkfile anlegen, das Namen der Originaldatei enthält



Stattdessen:

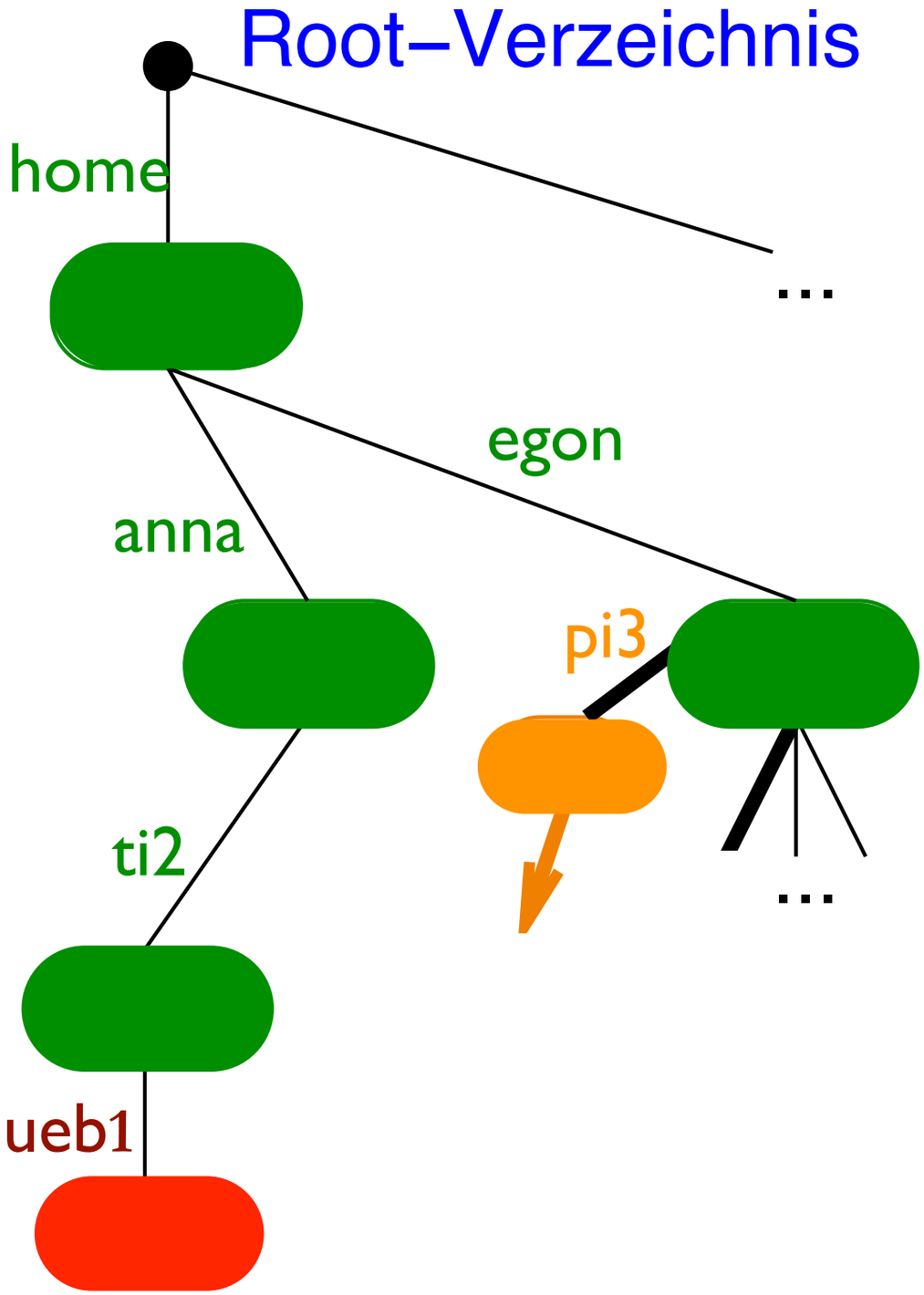
b) Symbolic Links

```
ln -s fn linkfn
```

```
ln -s /home/anna/pi2/ueb2 /home/egon/u2
```

- Linkfile anlegen, das Namen der Originaldatei enthält
- Auch für Verzeichnisse möglich

```
ln -s /home/anna/pi3 /home/egon/pi3
```



Stattdessen:

b) Symbolic Links

```
ln -s fn linkfn
```

```
ln -s /home/anna/pi2/ueb2 /home/egon/u2
```

- Linkfile anlegen, das Namen der Originaldatei enthält
- Auch für Verzeichnisse möglich

```
ln -s /home/anna/pi3 /home/egon/pi3
```

- Originaldatei löschen

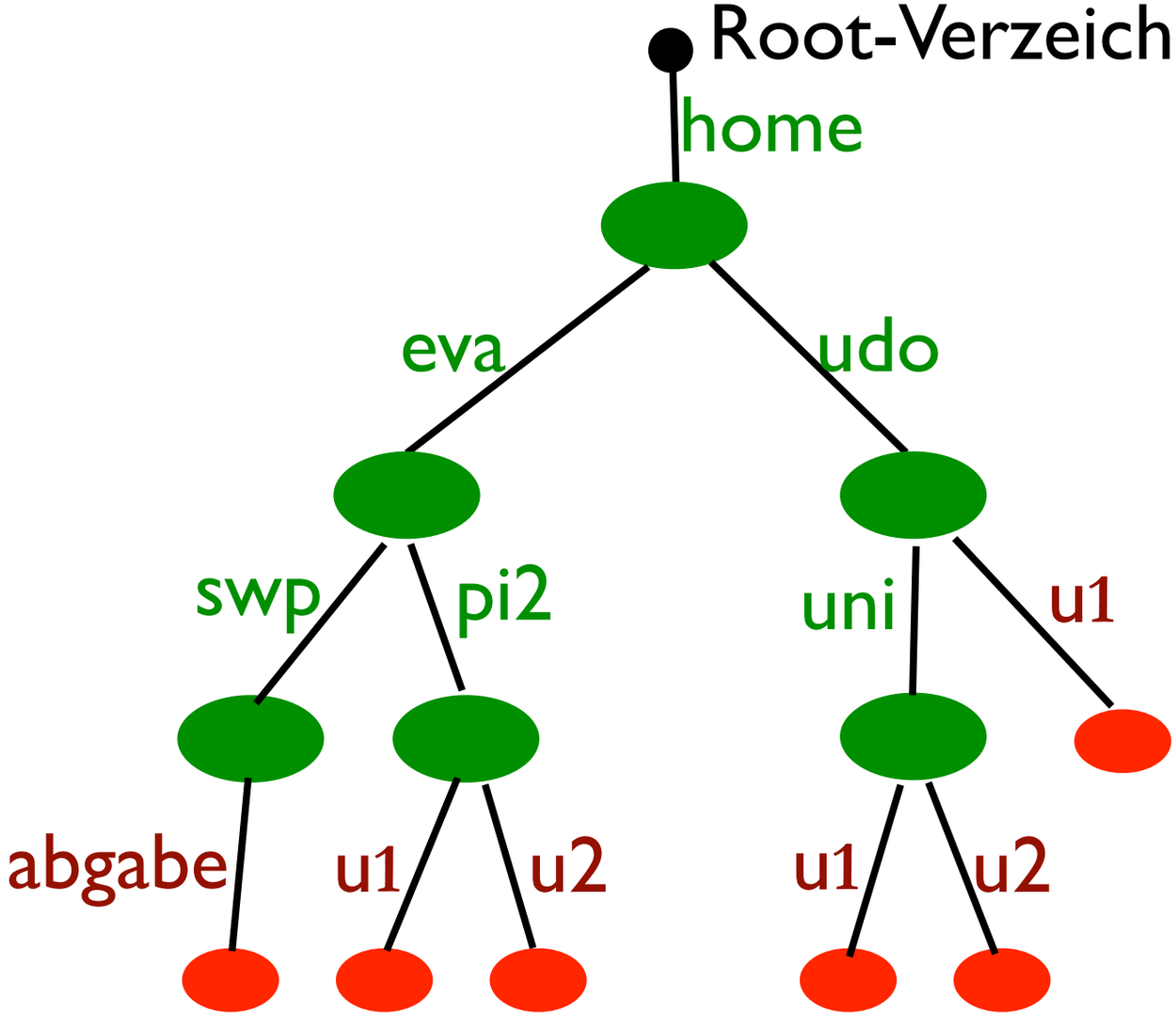
⇒ Link „hängt in der Luft“

Fragen – Teil 1

- Wie ist ein Unix-Dateisystem strukturiert?
- Wie können Dateien darin (eindeutig) aufgefunden werden?
- Was ist ein *Hard Link*? Was ist ein *Symbolic Link*?
- Ist das Unix-Dateisystem wirklich ein Baum? Begründung.

Kleine Aufgabe

Gegeben sei der folgende Auszug aus einem Unix-Dateisystem. Was ändert sich durch die angegebene Folge von Shell-Kommandos? Welches ist danach das aktuelle Arbeitsverzeichnis?




```
$ mkdir /home/fritz
$ cd /home/fritz
$ ln -s ~eva/pi2 pi
$ ln /home/udo/uni/u1 ueb
$ mv ueb ueb1
$ cd pi
$ cat u2
$ cd ../..
$ pwd
```

Teil 2:
Arten von Dateien

Nutzung des Betriebssystems (vereinfacht)

- Bisher: Dateisystem aus Benutzersicht

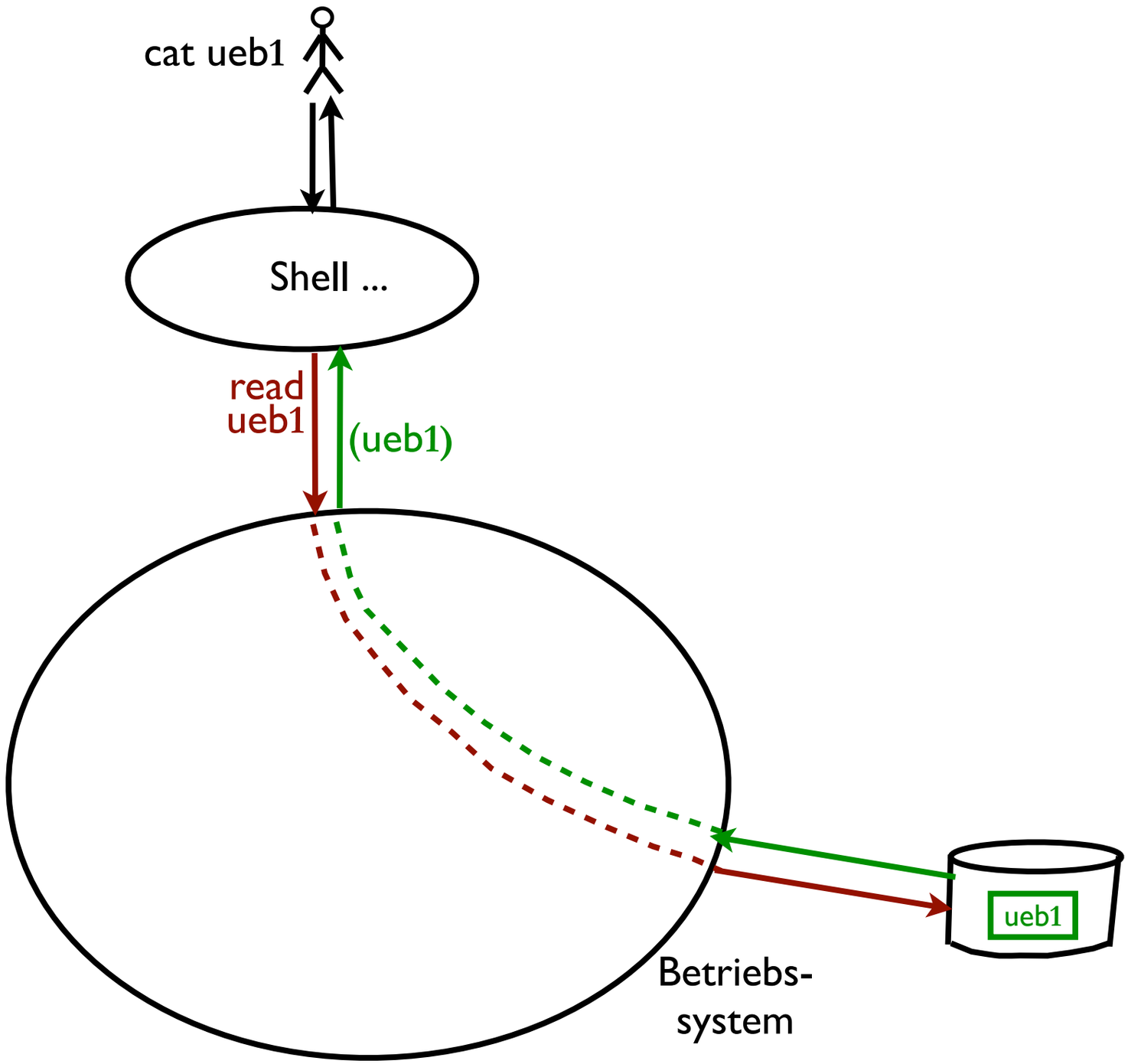
  ⇒ Shell-Kommandos zum Zugriff auf Dateien, Verzeichnisse...

- Verwaltung der Dateien wird jedoch vom Betriebssystem(kern) vorgenommen

  ⇒ Shell-Kommandos müssen auf Betriebssystemaufrufe abgebildet werden

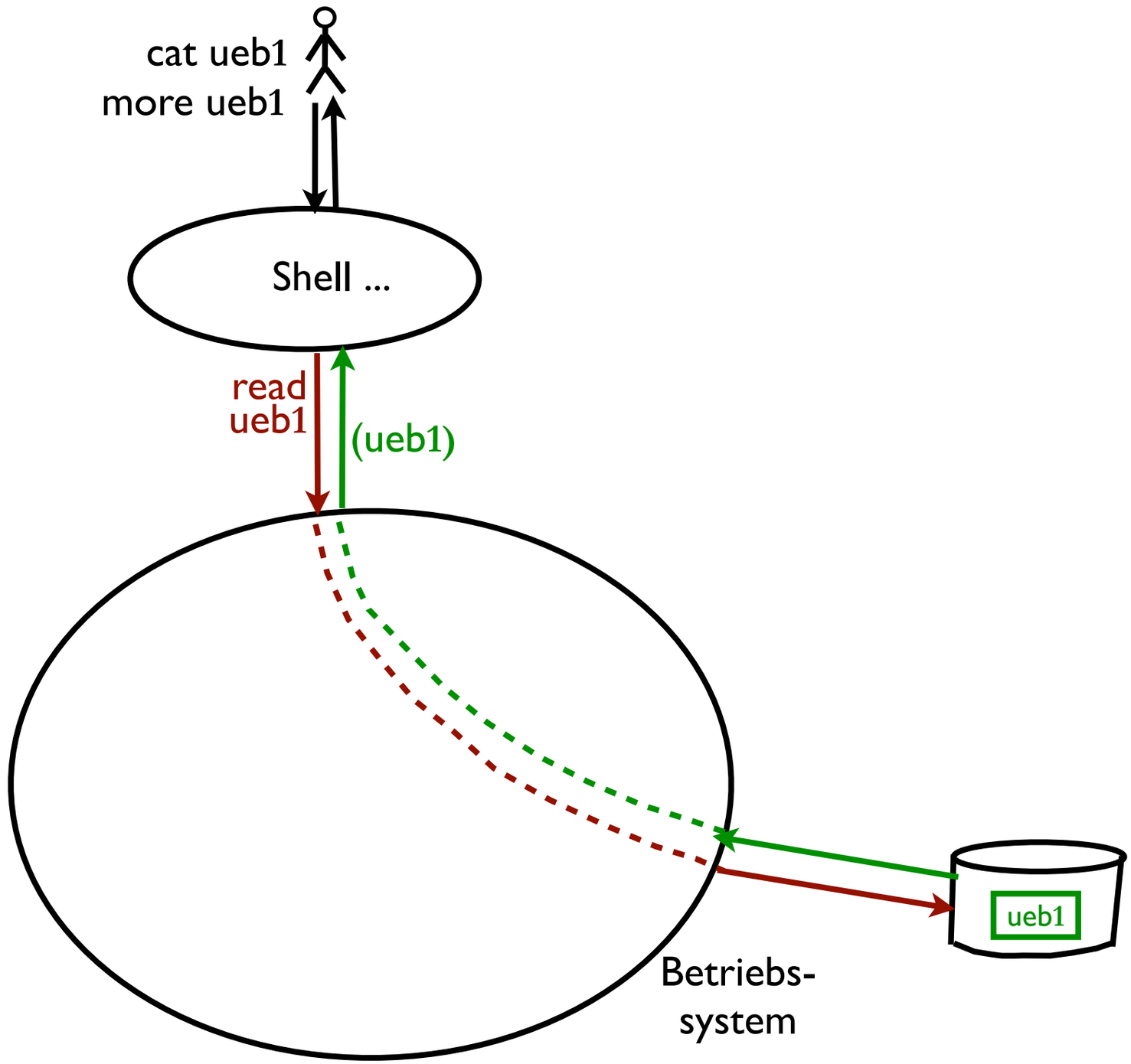
Nutzung des Betriebssystems (vereinfacht)

- Stark vereinfachtes Beispiel:



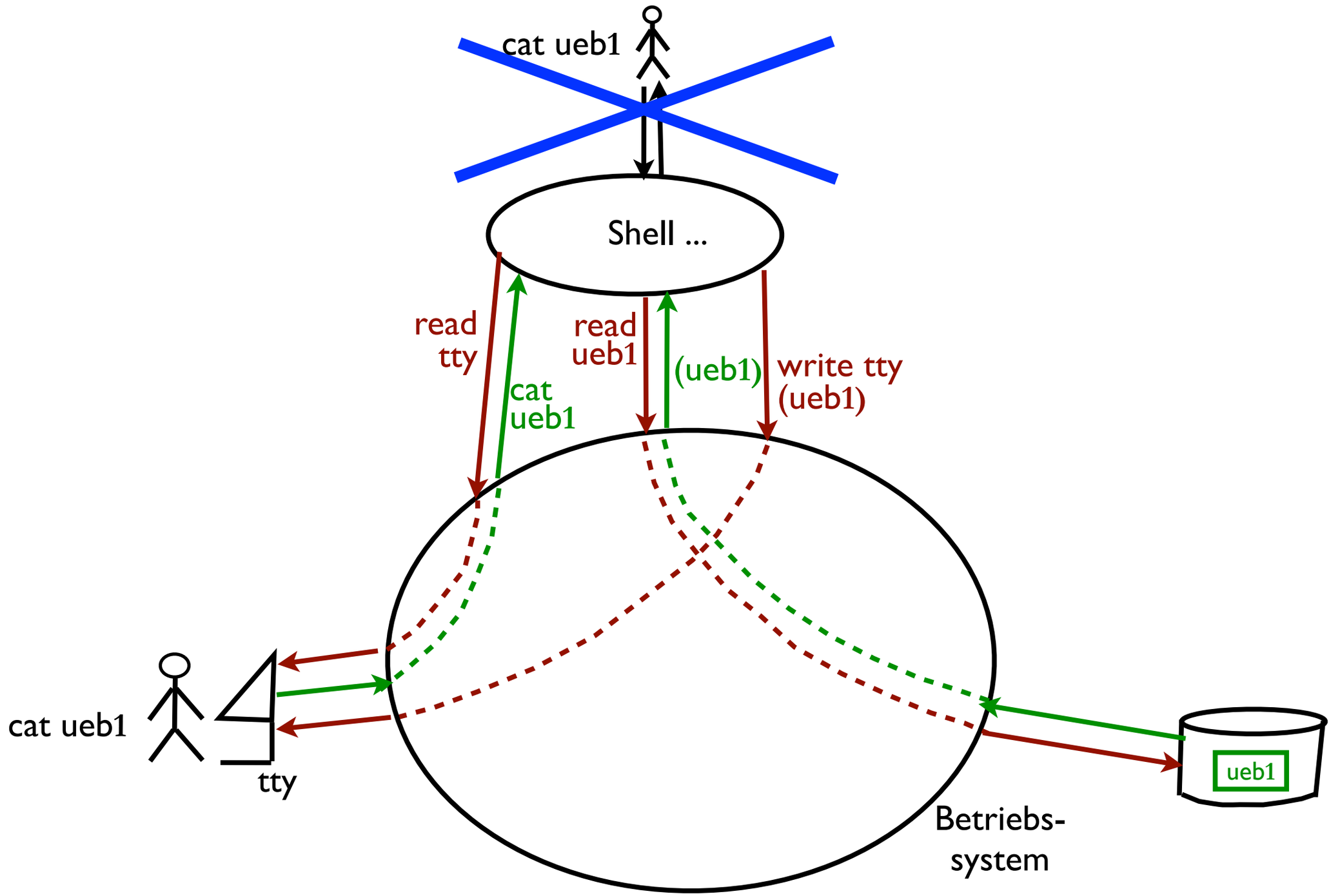
Nutzung des Betriebssystems (vereinfacht)

- Stark vereinfachtes Beispiel:



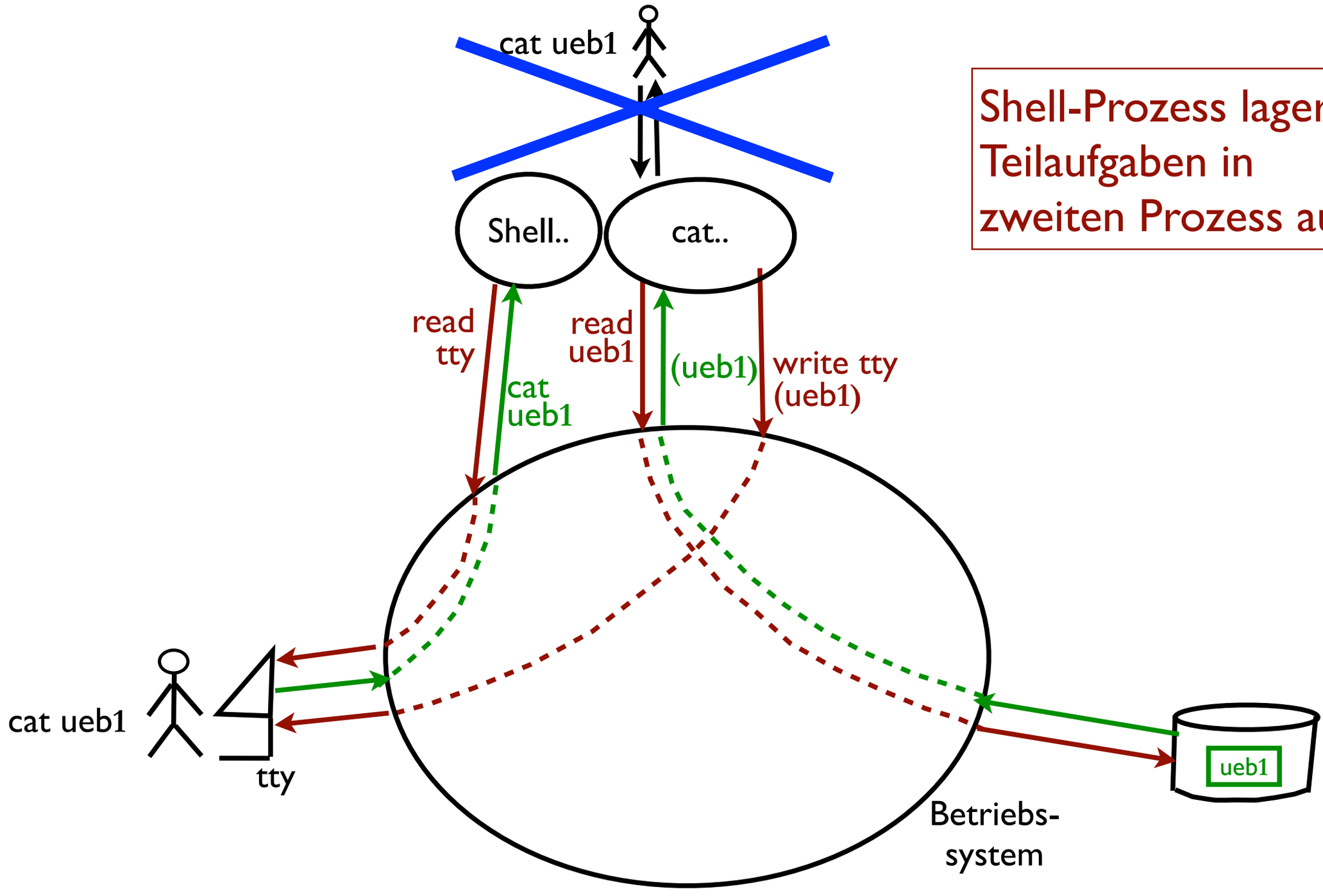
Nutzung des Betriebssystems (vereinfacht)

- Stark vereinfachtes Beispiel:



Nutzung des Betriebssystems (vereinfacht)

- Stark vereinfachtes Beispiel:



Shell-Prozess lagert Teilaufgaben in zweiten Prozess aus

Arten von Dateien

„Normale“ Dateien (Plain Files)

- Folge von Bytes (beliebige interne Struktur)
- Eindeutiger Name
- Schnittstelle zum Betriebssystem: read, write, ...

Arten von Dateien

„Normale“ Dateien (Plain Files)

- Folge von Bytes (beliebige interne Struktur)
- Eindeutiger Name
- Schnittstelle zum Betriebssystem: read, write, ...

⇒ ähnliche Mechanismen auch für andere Objekte nutzbar:

- Geräte (tty)
- Verzeichnisse
- ...

⇒ Datei: Universelles Konzept in Unix

Spezielle Dateien haben Besonderheiten ⇒ folgende Folien

Verzeichnisse

- enthalten Namen von anderen Dateien
  (spezielle Struktur)
- lesbar mit ls
- nicht direkt schreibbar
  - mv, rm, etc. wird intern auf Schreiben abgebildet
    - z.B. Emacs-Modus zum Bearbeiten von Verzeichnissen
- mkdir, rmdir

Verzeichnisse

- enthalten Namen von anderen Dateien (spezielle Struktur)
- lesbar mit ls
- nicht direkt schreibbar
  - mv, rm, etc. wird intern auf Schreiben abgebildet
    - z.B. Emacs-Modus zum Bearbeiten von Verzeichnissen
- mkdir, rmdir

Symbolic Links

- enthalten Namen der Datei, auf die verwiesen werden soll

Geräte-Dateien

- Zugriff auf Geräte
  (Terminals, Drucker, Platten, Hauptspeicher, ...)
- Unterscheidung:
  - block special files
    (blockorientierte Schnittstelle)
  - character special files
    (byteorientierte Schnittstelle)

Sockets und Named Pipes

- Zugriff auf „Kommunikationskanäle“
  (zwischen Prozessen, ggf. auch Systemen)

- Dateityp wird bei ls -l angezeigt

Plain	`-`
Directory	`d`
Link	`l`
Block.spec.	`b`
Char.spec.	`c`
Socket	`s`
Named Pipe	`p`

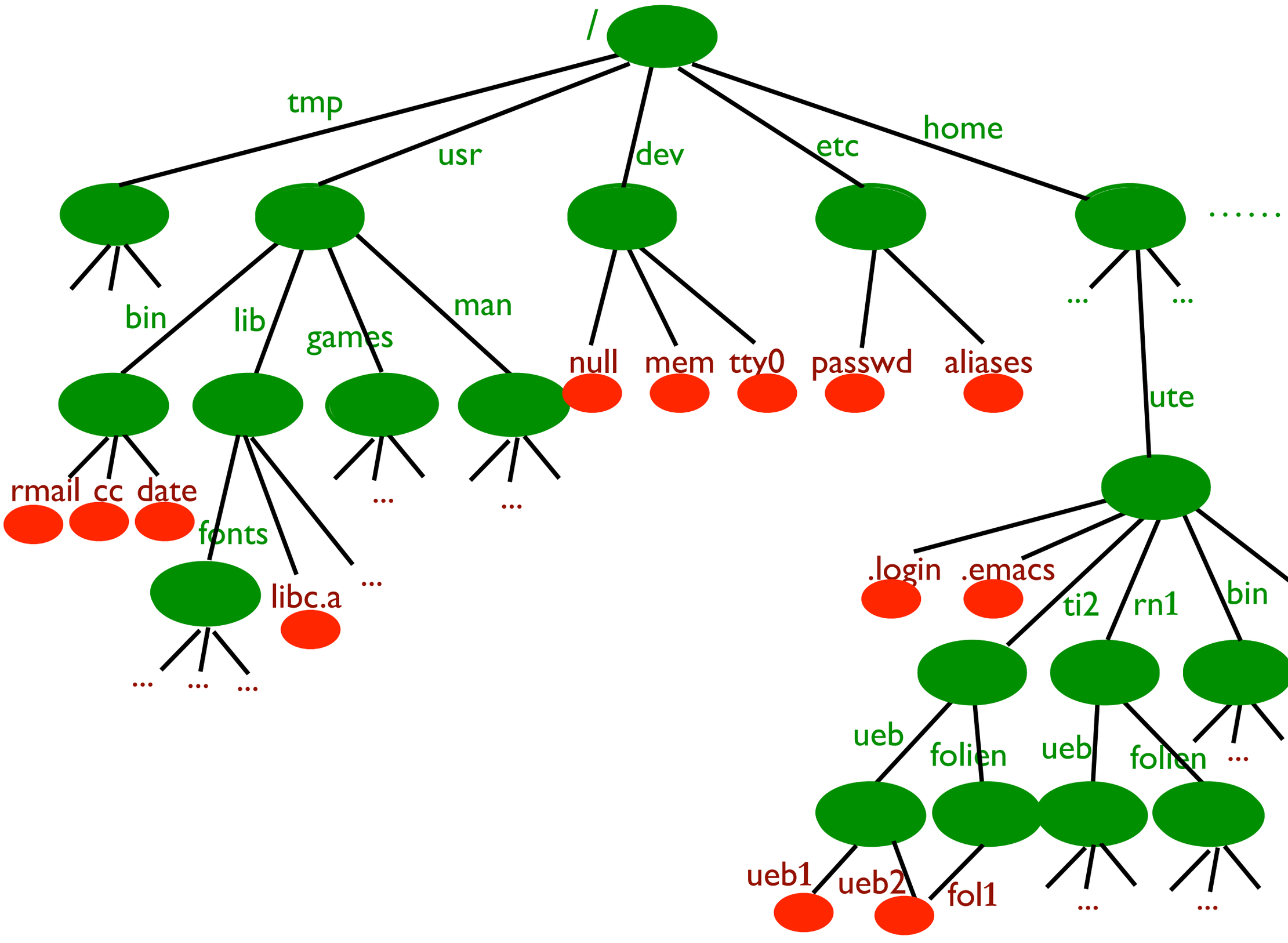
- Dateityp wird bei ls -l angezeigt

Plain	-	rw-r--r-- 2 root 30 Nov 10 14:11 README
Directory	d	rwxr--r-- 1 root 512 Nov 3 07:15 etc/
Link	l	rwxrwxrwx 1 root 7 May 14 10:27 bin@ ⇒ /usr/bin
Block.spec.	b	rw-r----- 1 root 7,41 May 16 1990 sd5
Char.spec.	c	rw--w--w- 1 root 20,0 Nov 8 1990 tty0
Socket	s	...
Named Pipe	p	...

major number (Geräteklasse)

minor number (spezifisches Gerät dieser Klasse)

Ein klassisches Unix-Dateisystem (vereinfacht)



Ein-/Ausgabeumlenkung

Analogie Datei $\longleftrightarrow$ Gerät hat weitere Vorteile:

- Benutzer kann sich entscheiden, wo Kommando-Ausgabe hingehen soll

```
date
Tue Oct 19 14:42:40 MET 1993
```

  $\Rightarrow$ date schreibt seine Ausgabe in eine „Standard-Ausgabedatei“ (stdout)

  - Defaultmäßig: „aktives“ Terminal(fenster)

Ein-/Ausgabeumlenkung

Analogie Datei $\longleftrightarrow$ Gerät hat weitere Vorteile:

- Benutzer kann sich entscheiden, wo Kommando-Ausgabe hingehen soll

```
date
Tue Oct 19 14:42:40 MET 1993
```

⇒ date schreibt seine Ausgabe in eine „Standard-Ausgabedatei“ (stdout)

  - Defaultmäßig: „aktives“ Terminal(fenster)
  - Umlenkung in andere Datei möglich:

```
date > bla
cat bla
Tue Oct 19 14:42:40 MET 1993
```

```
date >> bla
```
(hängt hinten an)

Ein-/Ausgabeumlenkung

Analogie Datei ⟷ Gerät hat weitere Vorteile:

- Benutzer kann sich entscheiden, wo Kommando-Ausgabe hingehen soll

```
date
Tue Oct 19 14:42:40 MET 1993
```

⇒ date schreibt seine Ausgabe in eine „Standard-Ausgabedatei“ (stdout)

  - Defaultmäßig: „aktives“ Terminal(fenster)
  - Umlenkung in andere Datei möglich:

```
date > bla
cat bla
Tue Oct 19 14:42:40 MET 1993
```

```
date >> bla
```
(hängt hinten an)

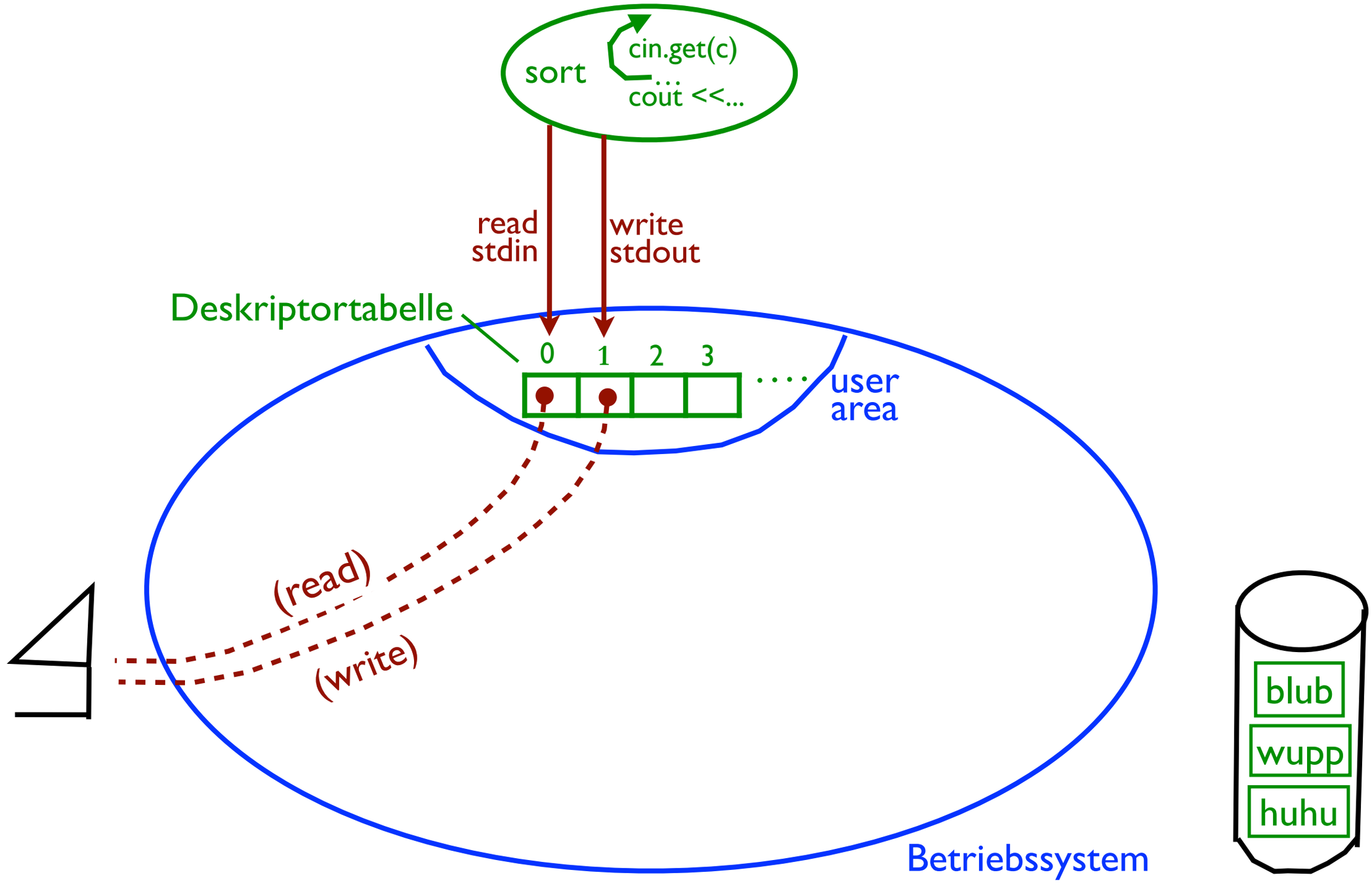
- Entsprechend auch für Eingabe

```
sort < huhu
```
(Standard Input, stdin)

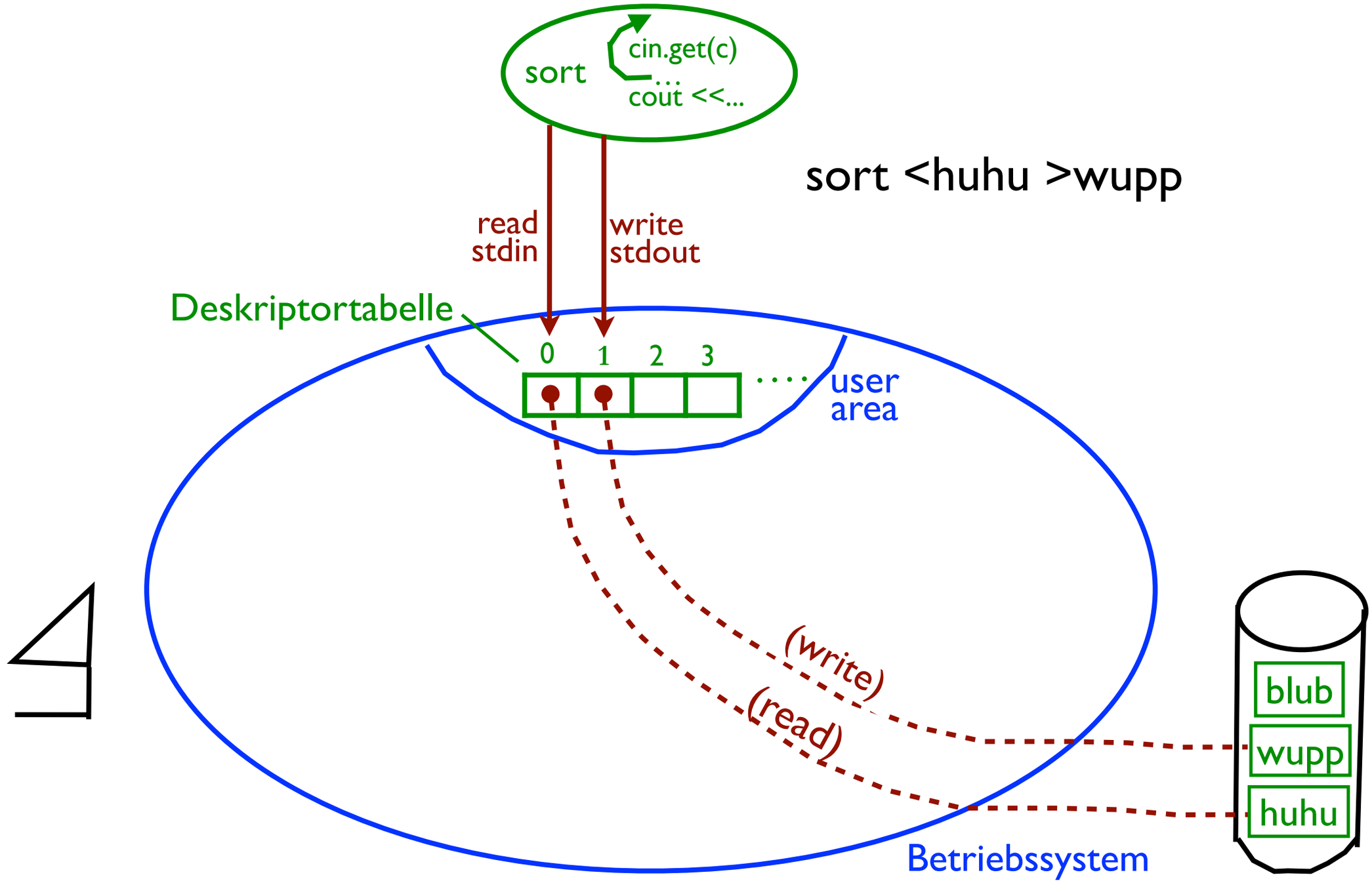
- Umlenkung von Fehlermeldungen

```
sort 2> error
```
(Standard Error, stderr)

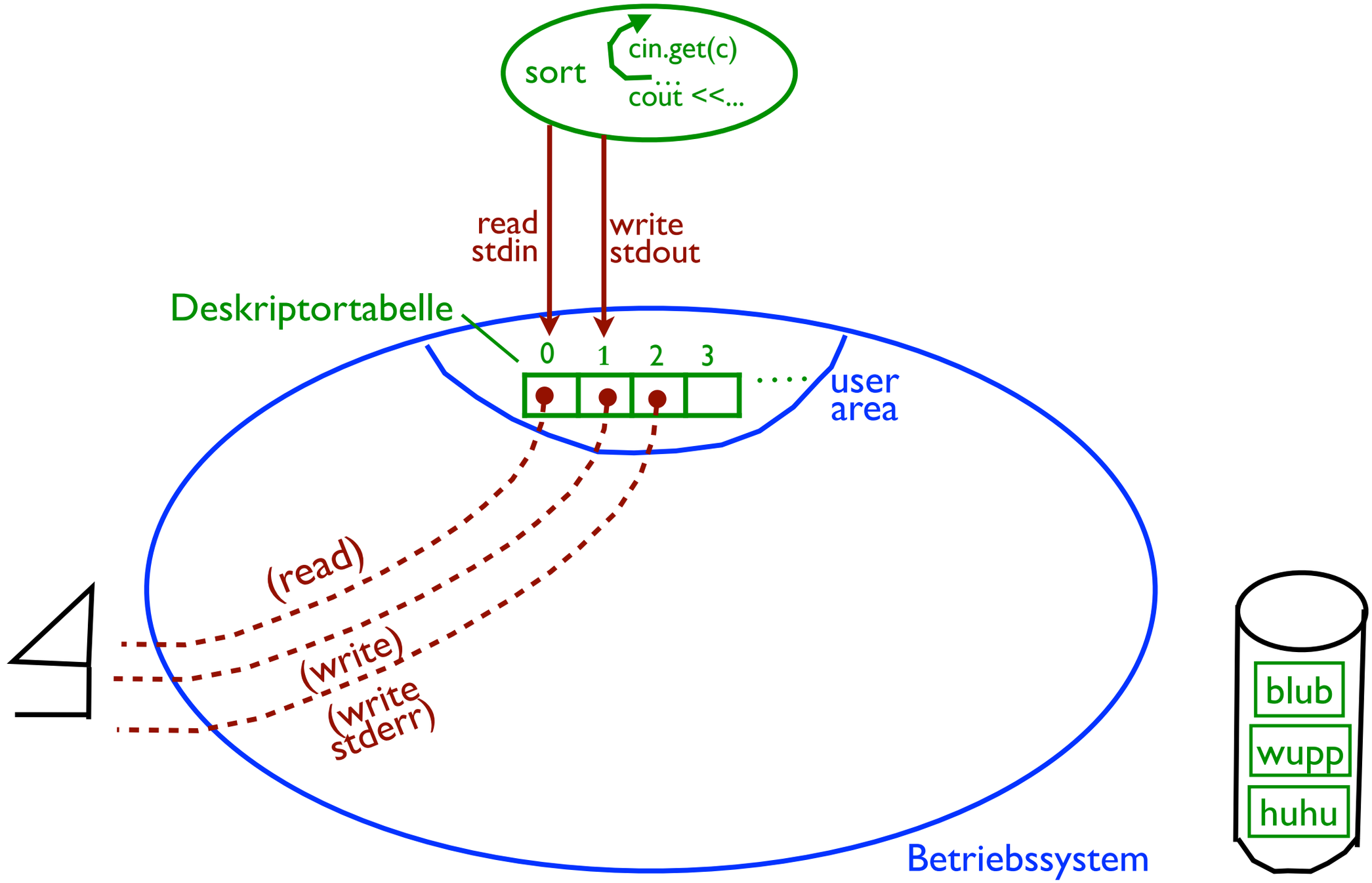
Verwaltung im Betriebssystemkern (vereinfacht)



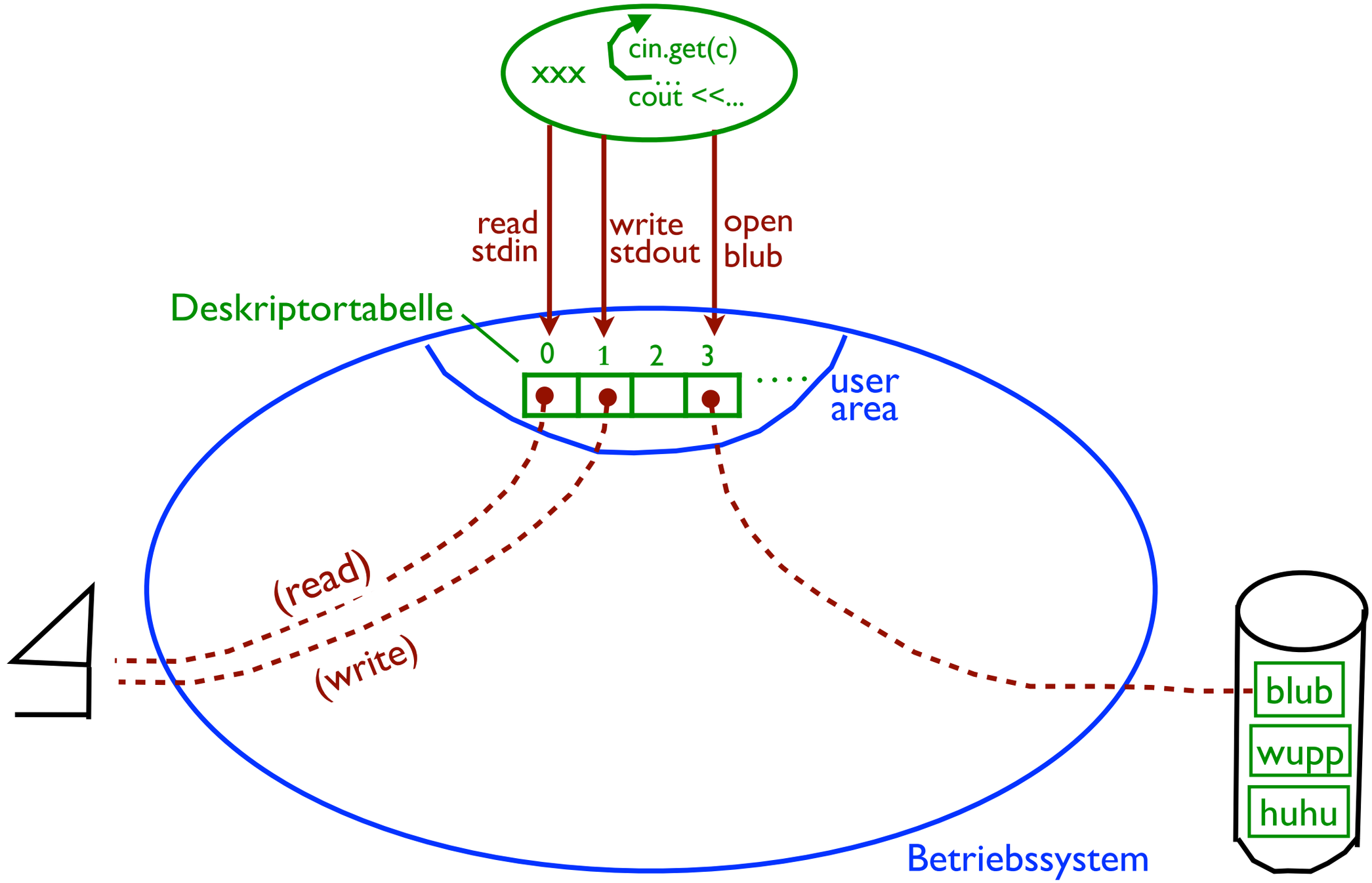
Verwaltung im Betriebssystemkern (vereinfacht)



Verwaltung im Betriebssystemkern (vereinfacht)



Verwaltung im Betriebssystemkern (vereinfacht)



Fragen – Teil 2

- Welche Vorteile bietet es, auf Geräte in Unix wie auf Dateien zuzugreifen?
- Was versteht man unter *Ein-/Ausgabeumlenkung*?

Teil 3:
Zugriffsschutz für Dateien

Zugriffsschutz für Dateien

- In Unix eigentlich „offene“ Zugriffsphilosophie
- Grundsätzlicher Zugang über Passwort

  ⇒ Dennoch: Dateien müssen vor unberechtigtem Zugriff geschützt werden

- unabsichtliche/absichtliche Angriffe:
  - unberechtigtes Lesen
  - unberechtigtes Schreiben
  - unberechtigtes Ausführen (von Programmen)

- Jede Datei hat einen Besitzer (Owner, uid)

  ⇒ hat Kontrolle über mögliche Zugriffsrechte

  a) für ihn selbst

  b) für „den Rest der Welt“ (= andere Nutzer dieses Systems)

  ⇒ zu grobe Unterscheidung

- Jede Datei hat einen Besitzer (Owner, uid)

  ⇒ hat Kontrolle über mögliche Zugriffsrechte

  a) für ihn selbst

  b) für „den Rest der Welt“ (= andere Nutzer dieses Systems)

  ⇒ zu grobe Unterscheidung

- Nutzer kann mehreren Gruppen angehören (ti2-Gruppe, pi3-Gruppe, ...)

  ⇒ 1 primäre Gruppe (DEFAULT), bis zu 16 sekundären Gruppen

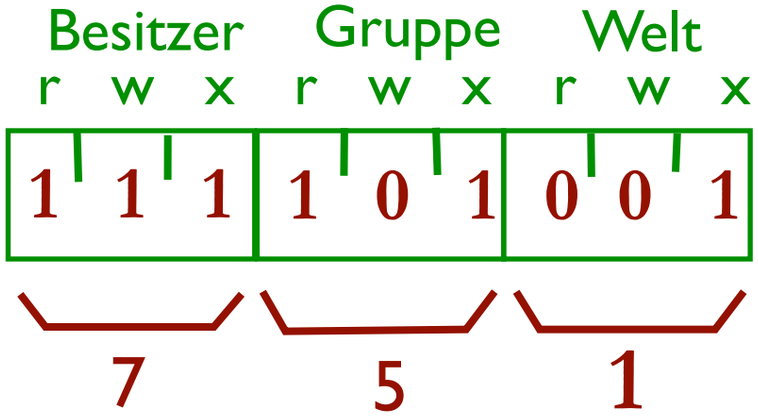
  c) für eine Gruppe

- Besitzer kann folgende Zugriffsrechte für Dateien vergeben

Beispiel

	Lesen (r)	Schreiben (w)	Ausführen (x)
Besitzer	X	X	X
Gruppe	X		X
„Welt“			X

- „Repräsentation“ über entsprechende Attribute der Dateien



	r	w	x
Besitzer	X	X	X
Gruppe	X		X
„Welt“			X

- Setzen von Zugriffsrechten:

  `chmod 751 fn`

  - Es gibt Default-Einstellung: oft 644 oder 755
    - Wird durch umask (oft 22) gesteuert

- Setzen der Gruppe:

  `chgrp gname fn`

  - Besitzer muss Mitglied der Gruppe sein

- Ändern des Besitzers („Verschenken“ von Dateien)

  `chown newowner fn`

  - nicht immer zulässig

Einrichtung von Gruppen: Tool grp

`grp -setup gname`

(Erzeugen der Gruppe)

`grp -invite gname uname`

(Nutzer in Gruppe einladen)

⇒ Mail an Nutzer

`grp -join gname`

(Gruppe beitreten; max. 16 Gruppen pro Nutzer)

(Achtung: Wird nicht sofort wirksam!)

⇒ Danach Mitglied der Gruppe

⇒ Setzen von Gruppenrechten für Dateien möglich

Zugriffsrechte (genauer)

		Besitzer			Gruppe	Welt
		r	w	x	r w x	r w x
Plain Files		lesen	schreiben	ausführen	(dito)	(dito)

Zugriffsrechte (genauer)

	Besitzer			Gruppe			Welt		
	r	w	x	r	w	x	r	w	x
Plain Files	lesen	schreiben	ausführen	(dito)			(dito)		
Directory	Einträge lesen (ls)	Einträge (indirekt) ändern (mv, rm)	auf Dateien darin zugreifen	(dito)			(dito)		

Zugriffsrechte (genauer)

	SetUID	SetGID	Sticky	Besitzer r	Besitzer w	Besitzer x	Gruppe r w x	Welt r w x
Plain Files	Ausführen Benutzer vs. Besitzer der Datei	Ausführen Benutzer vs. Datei-gruppe	(Bleibt in Swap Space „kleben“)	lesen	schreiben	ausführen	(dito)	(dito)
Directory		Dateien darin erhalten Gruppe der Direct.	Nur eigene Dateien in Direct. löschbar (/tmp/…)	Einträge lesen (ls)	Einträge (indirekt) ändern (mv, rm)	auf Dateien darin zugreifen	(dito)	(dito)

bei x=on für plain files
x=off ➔ mandatory file locking

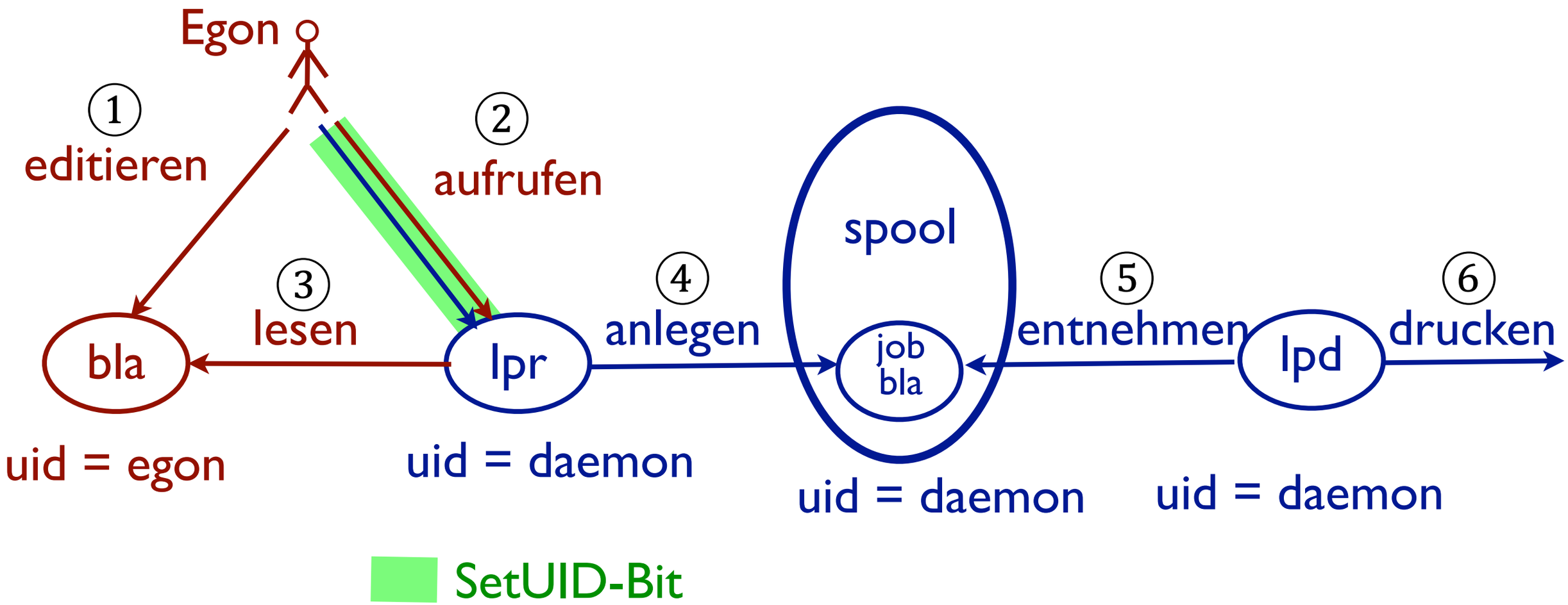
Zugriffsrechte (genauer)

	SetUID	SetGID	Sticky	Besitzer r	Besitzer w	Besitzer x	Gruppe r w x	Welt r w x
Plain Files	Ausführen Benutzer vs. Besitzer der Datei	Ausführen Benutzer vs. Datei-gruppe	(Bleibt in Swap Space „kleben“)	lesen	schreiben	ausführen	(dito)	(dito)
Directory		Dateien darin erhalten Gruppe der Direct.	Nur eigene Dateien in Direct. löschbar (/tmp/…)	Einträge lesen (ls)	Einträge (indirekt) ändern (mv, rm)	auf Dateien darin zugreifen	(dito)	(dito)

bei x=on für plain files
x=off ➔ mandatory file locking

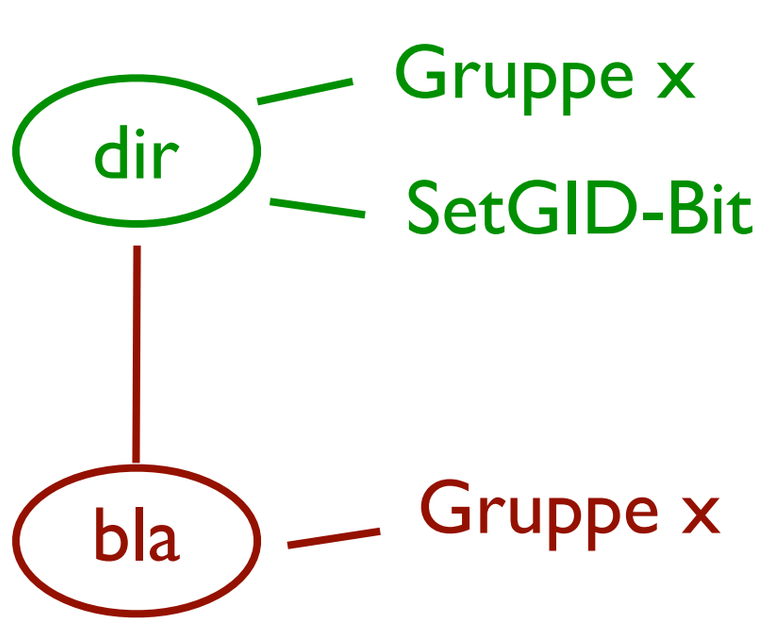
SetUID-Bit bei Plain Files (mit x=on)

- Ausstatten einer Datei mit Rechten von Besitzer und Aufrufer
- Beispiel: Druckerverwaltung

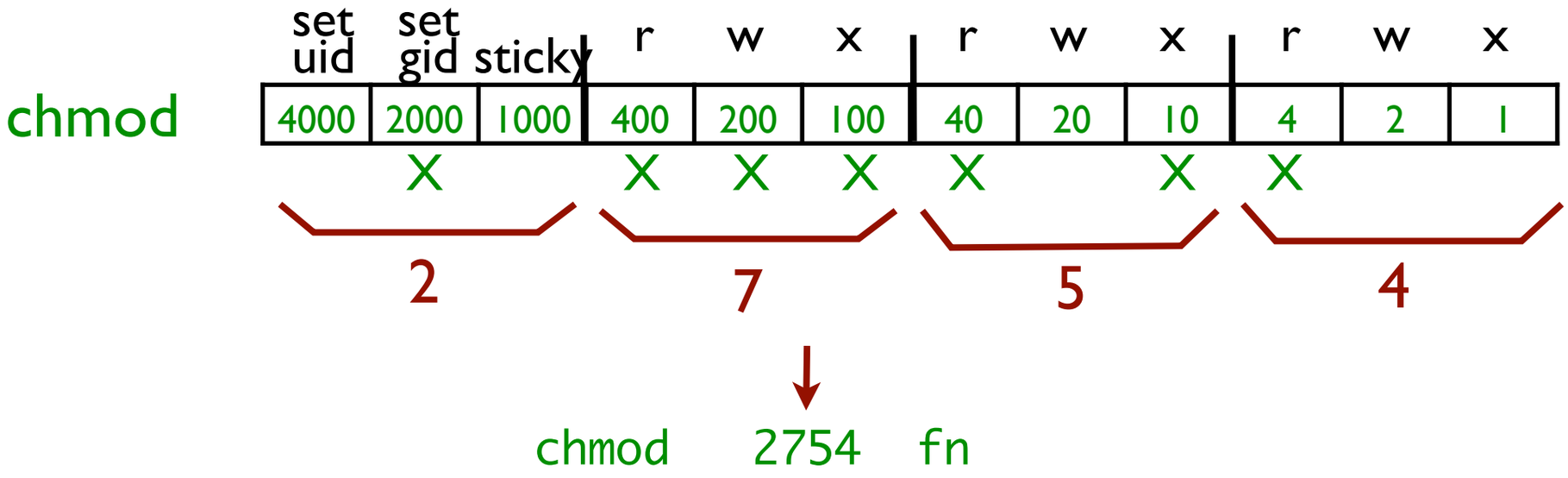


SetGID-Bit bei Verzeichnissen

- Dateien werden defaultmäßig mit der primären Gruppe ihres Besitzers angelegt
- Ausnahme:



Setzen von Zugriffsrechten (genauer)



- Achtung: 2000 (setgid) in manchen Unix-Varianten nicht verwendbar
- Besser alternative Syntax verwenden:

```
chmod  [ugoa] [+-=] [rwxs...]  fn
```

```
chmod  g=r   Gruppe nur Lesen
chmod  o-r   Lesen aus für Others
chmod  a+w   Schreiben an für alle
chmod  +x    Ausführen an für alle (mod. umask)
chmod  g+wx  Schreiben und Ausführen an für Gruppe
chmod  g+s   Setgid an
```

- Anzeigen der Zugriffsrechte über `ls -l`

```
- rwxr-xr-x   1   ute  prof  437  Oct 5 8:30  bla
- rw-------   2   ute  prof   17  May 8 3:05  geheim
```

Art Zugriffsrechte RefCount Owner Gruppe Size Datum Name

Owner Group Others

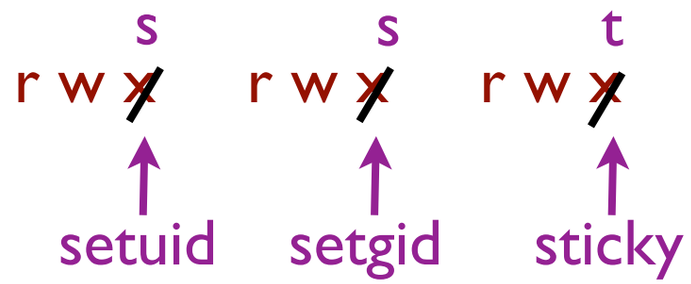
- Anzeigen der Zugriffsrechte über `ls -l`

```
- rwxr-xr-x   1   ute  prof  437  Oct 5 8:30  bla
- rw-------   2   ute  prof   17  May 8 3:05  geheim
```

Art Zugriffsrechte RefCount Owner Gruppe Size Datum Name

Owner Group Others

- Anzeigen von Setuid/Setgid/Sticky-Bits bei `ls`:



bei x=1

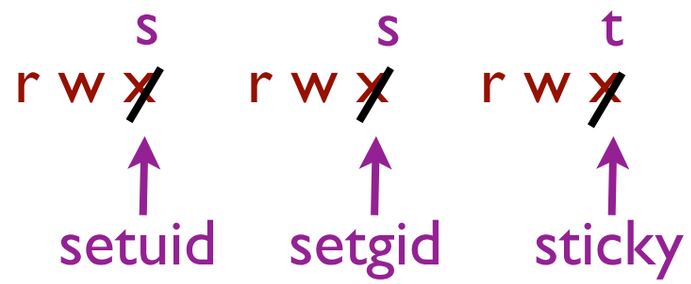
- Anzeigen der Zugriffsrechte über `ls -l`

```
- rwxr-xr-x   1   ute  prof  437  Oct 5 8:30  bla
- rw-------   2   ute  prof   17  May 8 3:05  geheim
```

Art Zugriffsrechte RefCount Owner Gruppe Size Datum Name

Owner Group Others

- Anzeigen von Setuid/Setgid/Sticky-Bits bei `ls`:



bei x=1

- `ls` sortiert Ausgabe alphabetisch. Alternativen:
  `ls -t` sortiert nach dem Datum „rückwärts“
  `ls -rt` sortiert nach dem Datum „vorwärts“
  `ls -lrt` lange Ausgabe + Vorwärts-Sortierung

- `ls -a` erwähnt auch .files
  (mit den anderen Optionen kombinierbar)

- `ls -lL` dereferenziert Symbolic-Link-Dateien

Fragen – Teil 3

- Welche Zugriffsrechte kann man auf eine Unix-Datei haben?
- Wer darf was bei folgenden Zugriffsrechten auf eine normale Datei?
  - `chmod 640 bla`
  - `chmod 511 blub`

Zusammenfassung

- Eigenschaften von Unix-Dateien
- Shell-Kommandos zur Unix-Dateiverwaltung
- Arten von Dateien
- Ein-/Ausgabeumlenkung
- Zugriffsrechte

Das Unix-Dateisystem aus Nutzersicht – Fragen

1. Wie ist ein Unix-Dateisystem strukturiert?
2. Wie können Dateien darin (eindeutig) aufgefunden werden?
3. Was ist ein *Hard Link*? Was ist ein *Symbolic Link*?
4. Ist das Unix-Dateisystem wirklich ein Baum? Begründung.
5. Welche Zugriffsrechte kann man auf eine Unix-Datei haben?
6. Wer darf was bei folgenden Zugriffsrechten auf eine normale Datei?
   - `chmod 640 bla`
   - `chmod 511 blub`
7. Welche Vorteile bietet es, auf Geräte in Unix wie auf Dateien zuzugreifen?
8. Was versteht man unter *Ein-/Ausgabeumlenkung*?