Work in progress

Dateiverwaltung (1)

Ute Bormann, TI2

2023-10-13

Betriebssysteme

- Prozessverwaltung
- Speicherverwaltung
- ⇒ Dateiverwaltung
- Geräteverwaltung
- Prozessverwaltung
  ⇒ Nebenläufigkeit ⇒ Kommunikation

Inhalt

1. Aufbau eines Dateisystems
2. Klassisches Beispiel: Unix-V7-Dateisystem
3. Systemaufrufe zur Dateiverwaltung

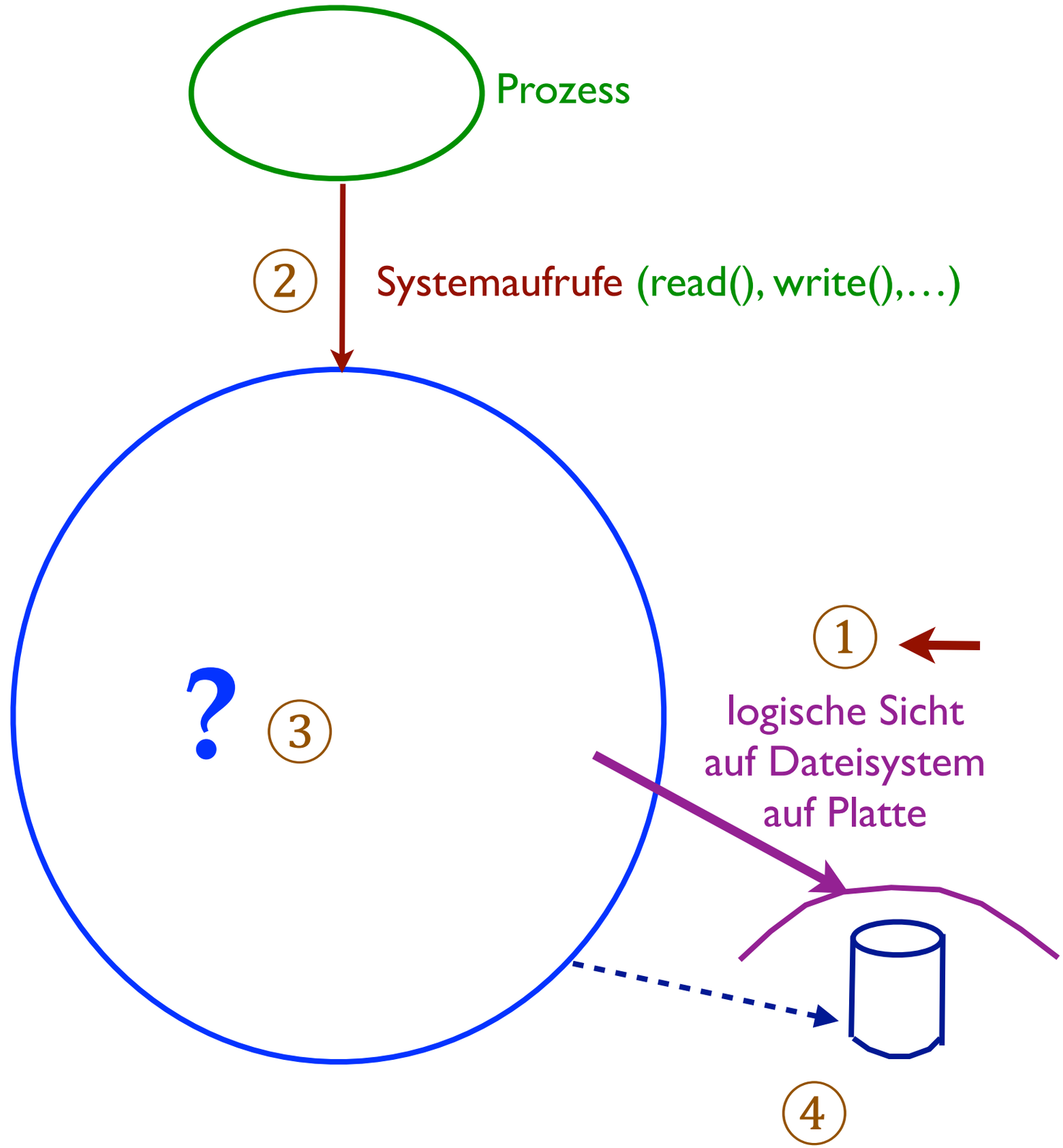
Teil 1:
Aufbau eines Dateisystems

Dateiverwaltung (Wdh.)

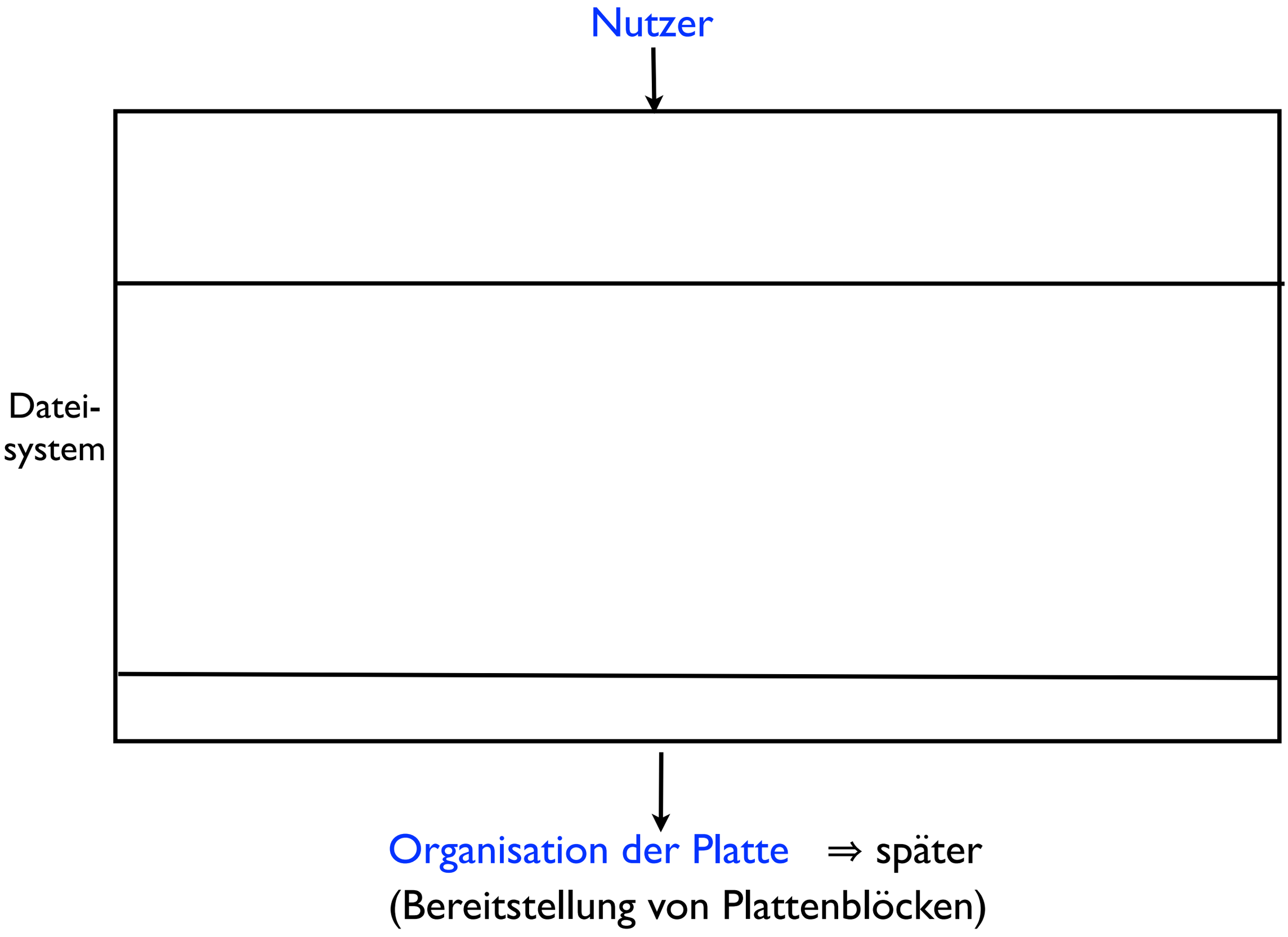
- Bisher: Benutzersicht in Unix
- Dateien sind langlebige Datenobjekte
  ⇒ Hintergrundspeicher (Platte,...)
- Dateistruktur: In Unix Folgen von Bytes
- Hierarchische Namensgebung über Directory-Struktur
  (⇒ spezielle Dateien)
- Hard links vs. symbolic links
- Schutzbits: R (lesen), W (schreiben), X (ausführen), ...
- Zugriffsoperationen: read(), write(), ... ⇒ Systemaufrufe
- Ähnliche Schnittstelle zu Geräten ⇒ Ein-/Ausgabeumlenkung

⇒ Bestandteile des Dateisystems

Überblick Dateiverwaltung



Ad 1) Logische Sicht auf ein Dateisystem



Organisation der Platte ⇒ später

(Bereitstellung von Plattenblöcken)

Ad 1) Logische Sicht auf ein Dateisystem

Nutzer

Datei-system

Verwaltung von (einzelnen) Dateien

a) Datei-Inhalt

⇒ Ablage auf der Platte

⇒ Verwaltung von Plattenblöcken

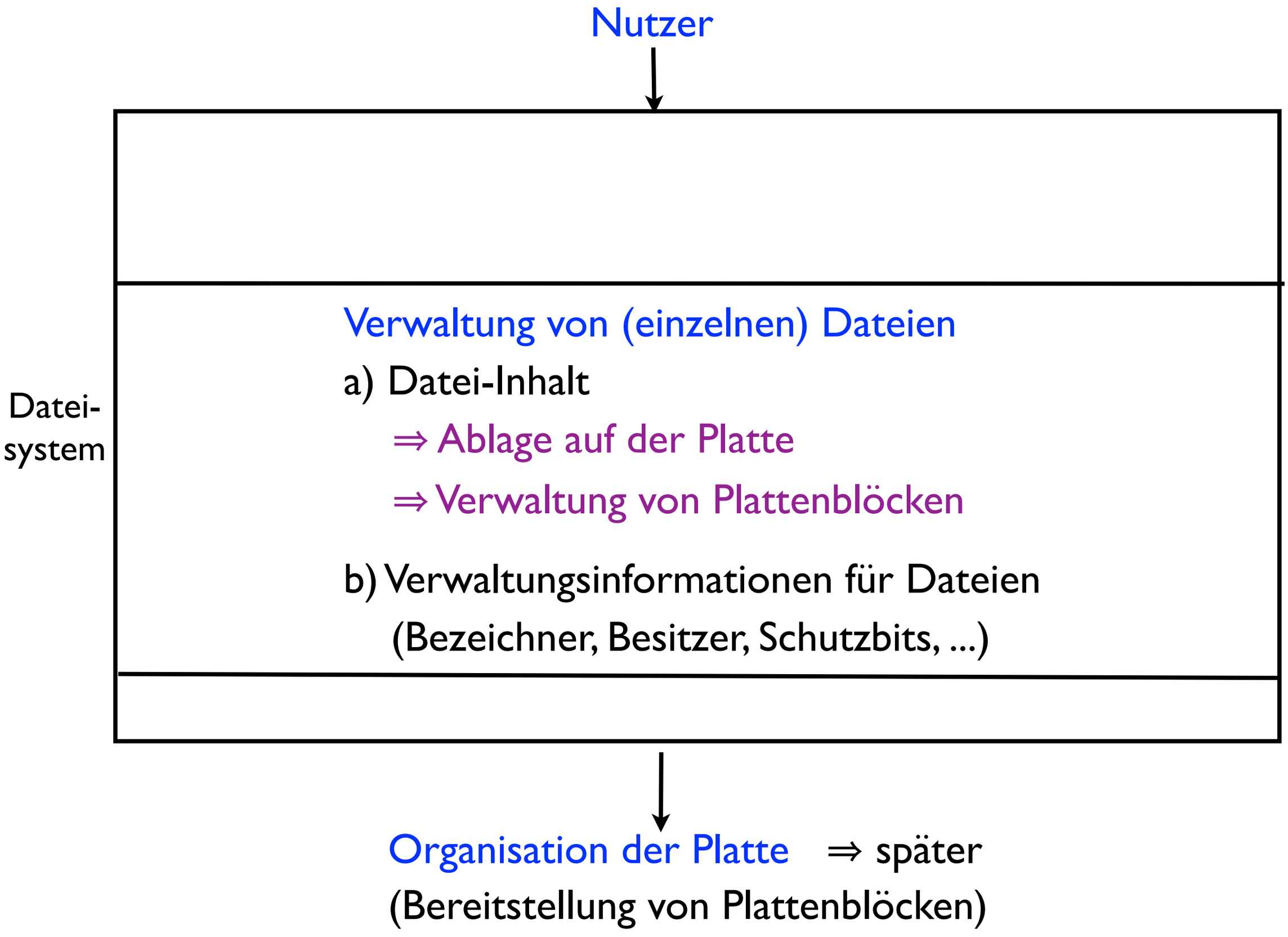
b) Verwaltungsinformationen für Dateien

(Bezeichner, Besitzer, Schutzbits, ...)

Organisation der Platte ⇒ später

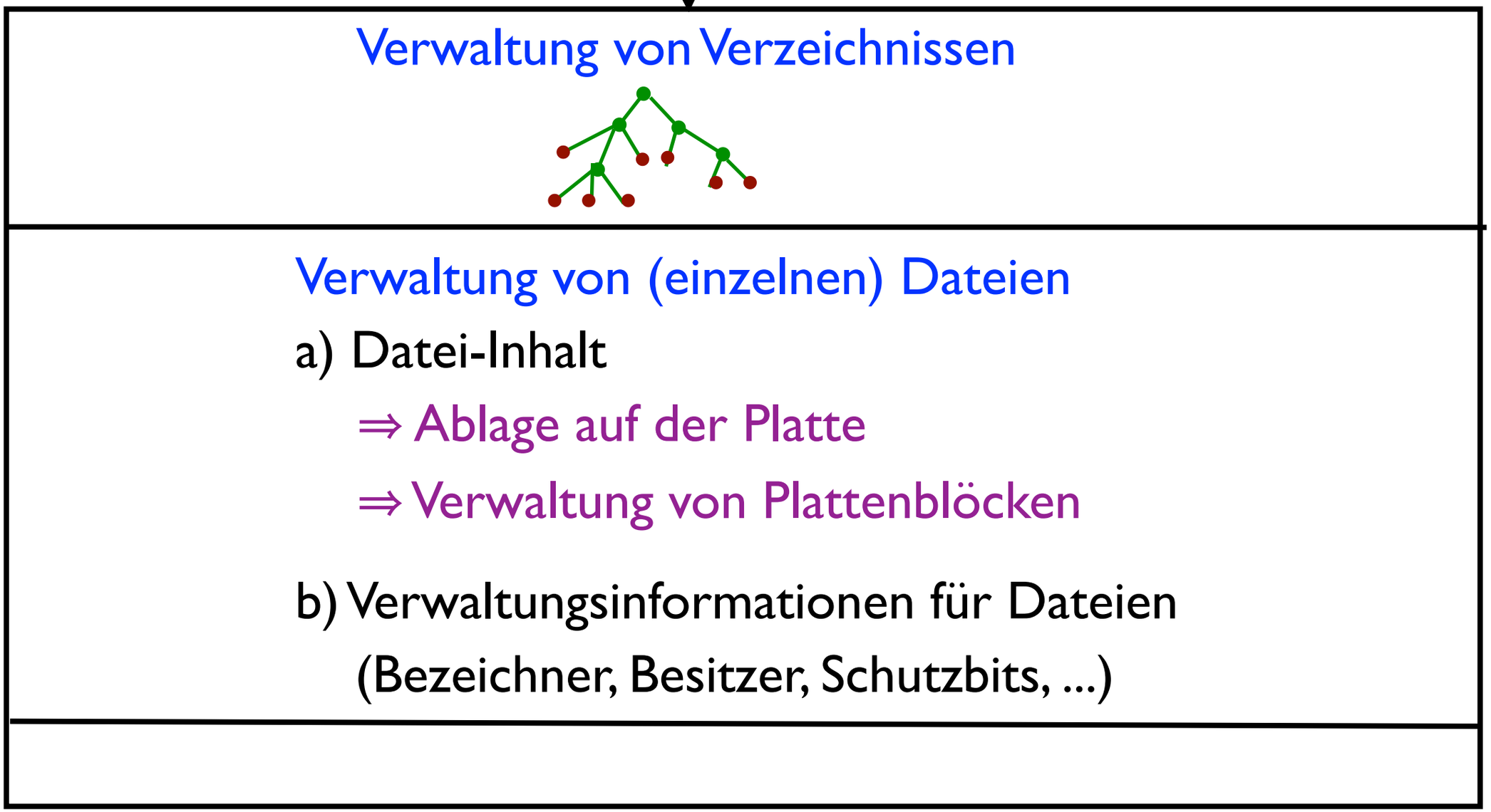
(Bereitstellung von Plattenblöcken)

Ad 1) Logische Sicht auf ein Dateisystem



Ad 1) Logische Sicht auf ein Dateisystem

Nutzer

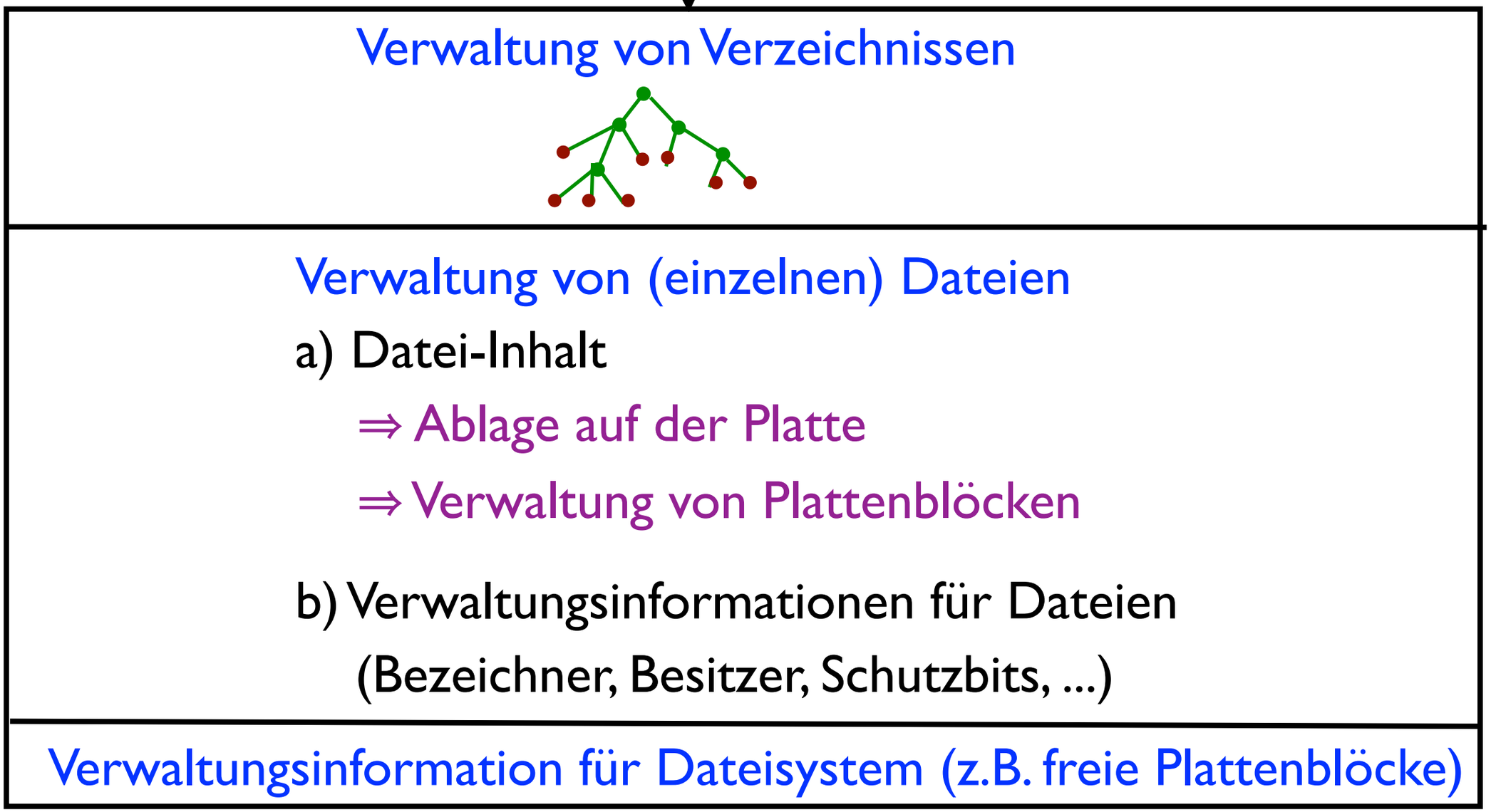


Datei-system

Organisation der Platte ⇒ später
(Bereitstellung von Plattenblöcken)

Ad 1) Logische Sicht auf ein Dateisystem

Nutzer



Datei-system

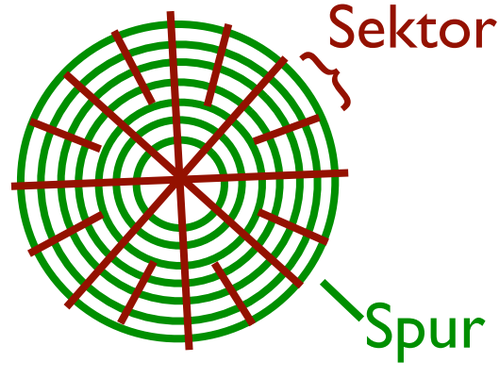
Organisation der Platte ⇒ später
(Bereitstellung von Plattenblöcken)

Ablage des Datei-Inhalts

- Datei-Inhalt: Folge von Bytes

- Organisation der Platte: Block-orientiert

Mehrere Oberflächen der Form:



⇒ Datei-Inhalt muss auf Platte verteilt werden

⇒ Spezielle Form von „Speicherverwaltung“

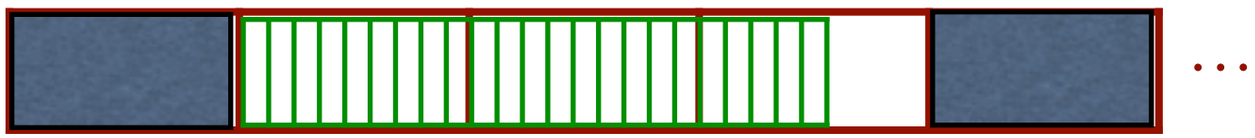
- Verschiedene Verfahren denkbar

1. Versuch: Sequentielle Ablage

- Ablage in einen „zusammenhängenden“ freien Bereich auf der Platte (z.B. nach First-Fit-Prinzip)

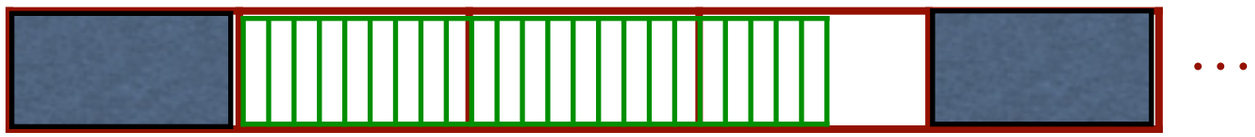
1. Versuch: Sequentielle Ablage

- Ablage in einen „zusammenhängenden“ freien Bereich auf der Platte (z.B. nach First-Fit-Prinzip)



1. Versuch: Sequentielle Ablage

- Ablage in einen „zusammenhängenden“ freien Bereich auf der Platte (z.B. nach First-Fit-Prinzip)



- Einfacher Zugriff auf jedes Byte (ab Anfangsadresse zählen)
- Aber:
  - Verschnittprobleme
  - Dynamisches Wachsen schwierig

  ⇒ Block-Struktur der Platte nicht ausgenutzt

2. Versuch: Blockweise Ablage

- Bytestrom in Dateiblöcke aufteilen

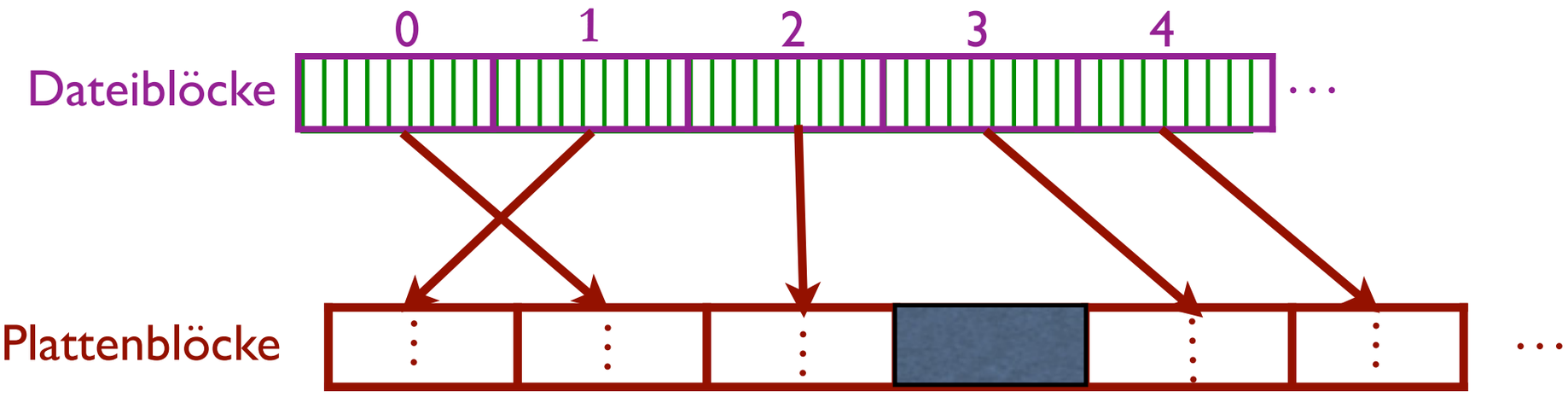
2. Versuch: Blockweise Ablage

- Bytestrom in Dateiblöcke aufteilen

2. Versuch: Blockweise Ablage

- Bytestrom in Dateiblöcke aufteilen
- Dateiblöcke in Plattenblöcke ablegen
  (im Prinzip beliebig über Platte verstreut)

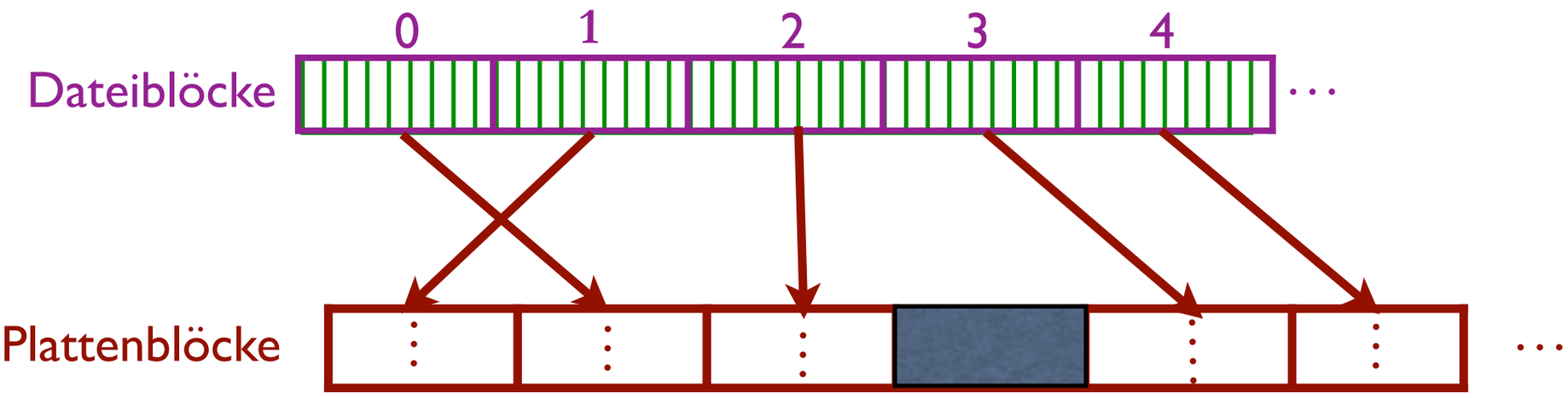
  ⇒ Bytenummer in Datei = Blocknummer + Adresse in Block



2. Versuch: Blockweise Ablage

- Bytestrom in Dateiblöcke aufteilen
- Dateiblöcke in Plattenblöcke ablegen
  (im Prinzip beliebig über Platte verstreut)

  ⇒ Bytenummer in Datei = Blocknummer + Adresse in Block

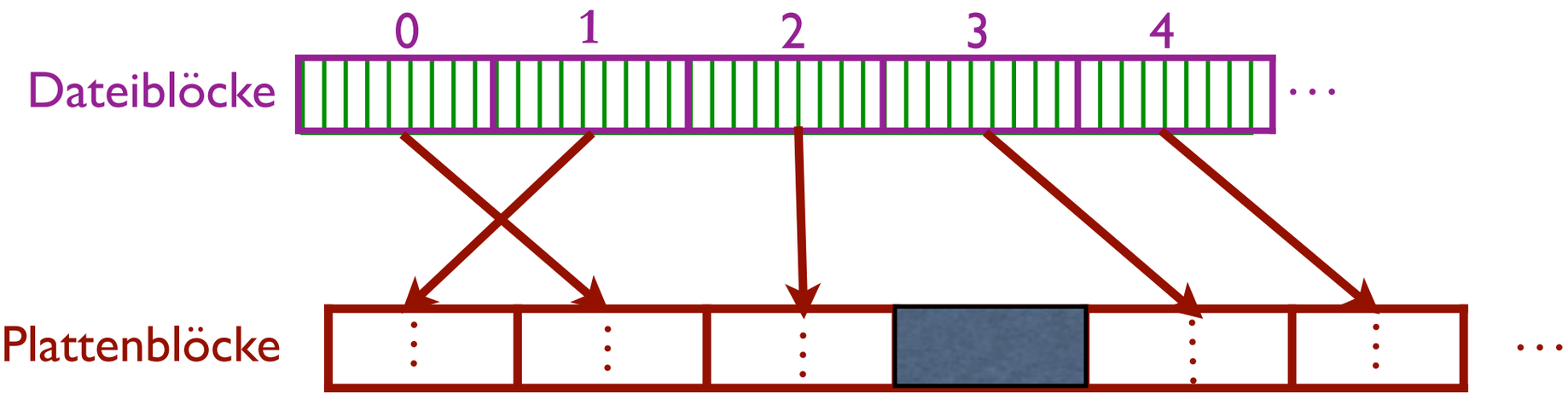


- Keine Verschnittprobleme (jeder Plattenblock nutzbar)
- Wachsen der Datei: Neue Plattenblöcke zuweisen
- Schrumpfen der Datei: Plattenblöcke freigeben

2. Versuch: Blockweise Ablage

- Bytestrom in Dateiblöcke aufteilen
- Dateiblöcke in Plattenblöcke ablegen
  (im Prinzip beliebig über Platte verstreut)

  ⇒ Bytenummer in Datei = Blocknummer + Adresse in Block

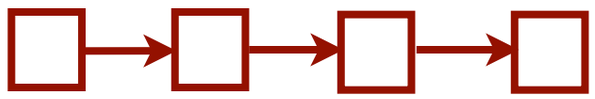


- Keine Verschnittprobleme (jeder Plattenblock nutzbar)
- Wachsen der Datei: Neue Plattenblöcke zuweisen
- Schrumpfen der Datei: Plattenblöcke freigeben
- Aber: Auffinden der Bytes komplizierter

Möglichkeiten zum Auffinden:

a) Verkettete Liste

- Verweis von einem Block auf den nächsten

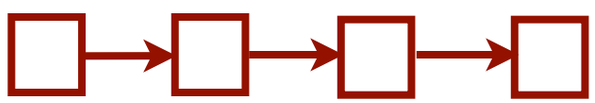


- Dynamisches Wachsen/Schrumpfen einfach
- Aber: nur sequentieller Zugriff direkt unterstützt
  ⇒ wahlfreier Zugriff dauert lange

Möglichkeiten zum Auffinden:

a) Verkettete Liste

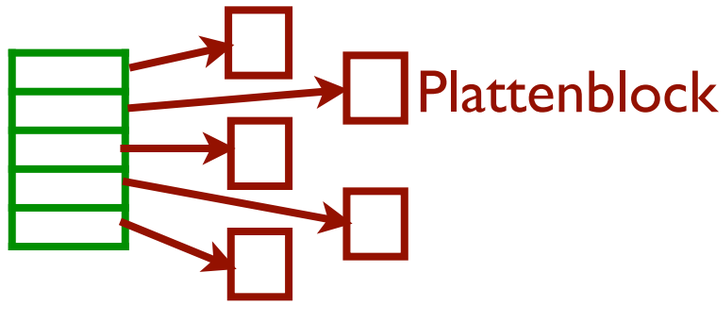
- Verweis von einem Block auf den nächsten



- Dynamisches Wachsen/Schrumpfen einfach
- Aber: nur sequentieller Zugriff direkt unterstützt
  ⇒ wahlfreier Zugriff dauert lange

b) Block-Index

- Zuordnungstabelle, über Dateiblocknummer indiziert

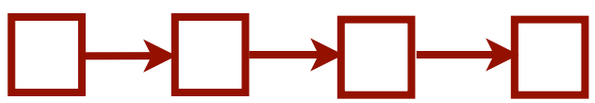


- Wahlfreier Zugriff einfach
- Aber: Größe der Tabelle variabel (große vs. kleine Dateien)

Möglichkeiten zum Auffinden:

a) Verkettete Liste

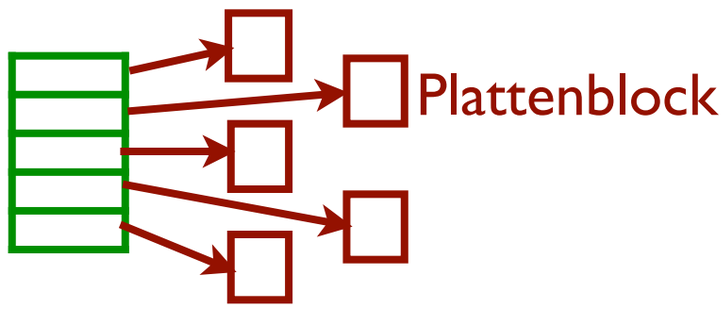
- Verweis von einem Block auf den nächsten



- Dynamisches Wachsen/Schrumpfen einfach
- Aber: nur sequentieller Zugriff direkt unterstützt
  ⇒ wahlfreier Zugriff dauert lange

b) Block-Index

- Zuordnungstabelle, über Dateiblocknummer indiziert



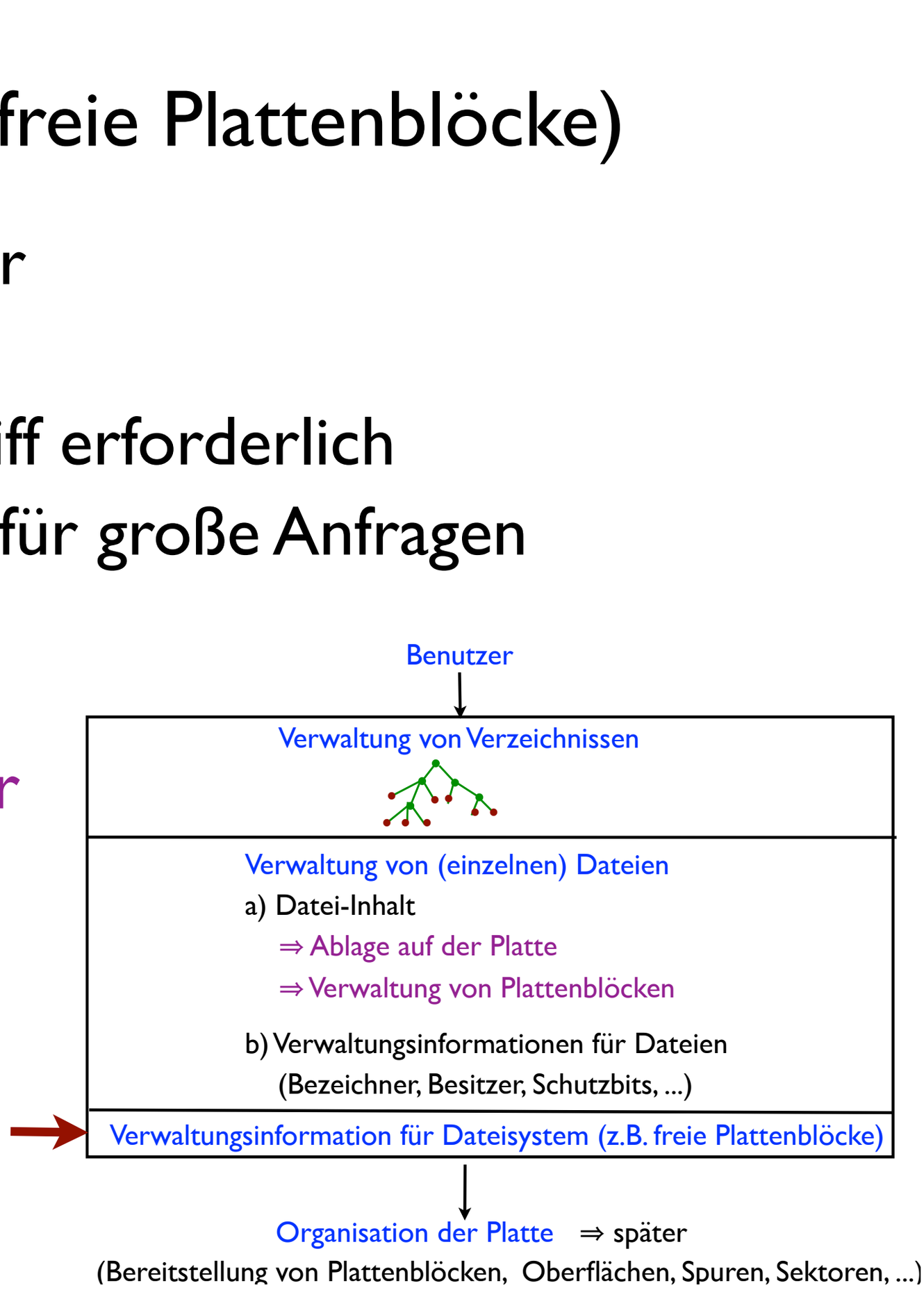
- Wahlfreier Zugriff einfach
- Aber: Größe der Tabelle variabel (große vs. kleine Dateien)

Resümee: Oft Block-Index-Varianten in Nutzung

Freispeicherverwaltung (freie Plattenblöcke)

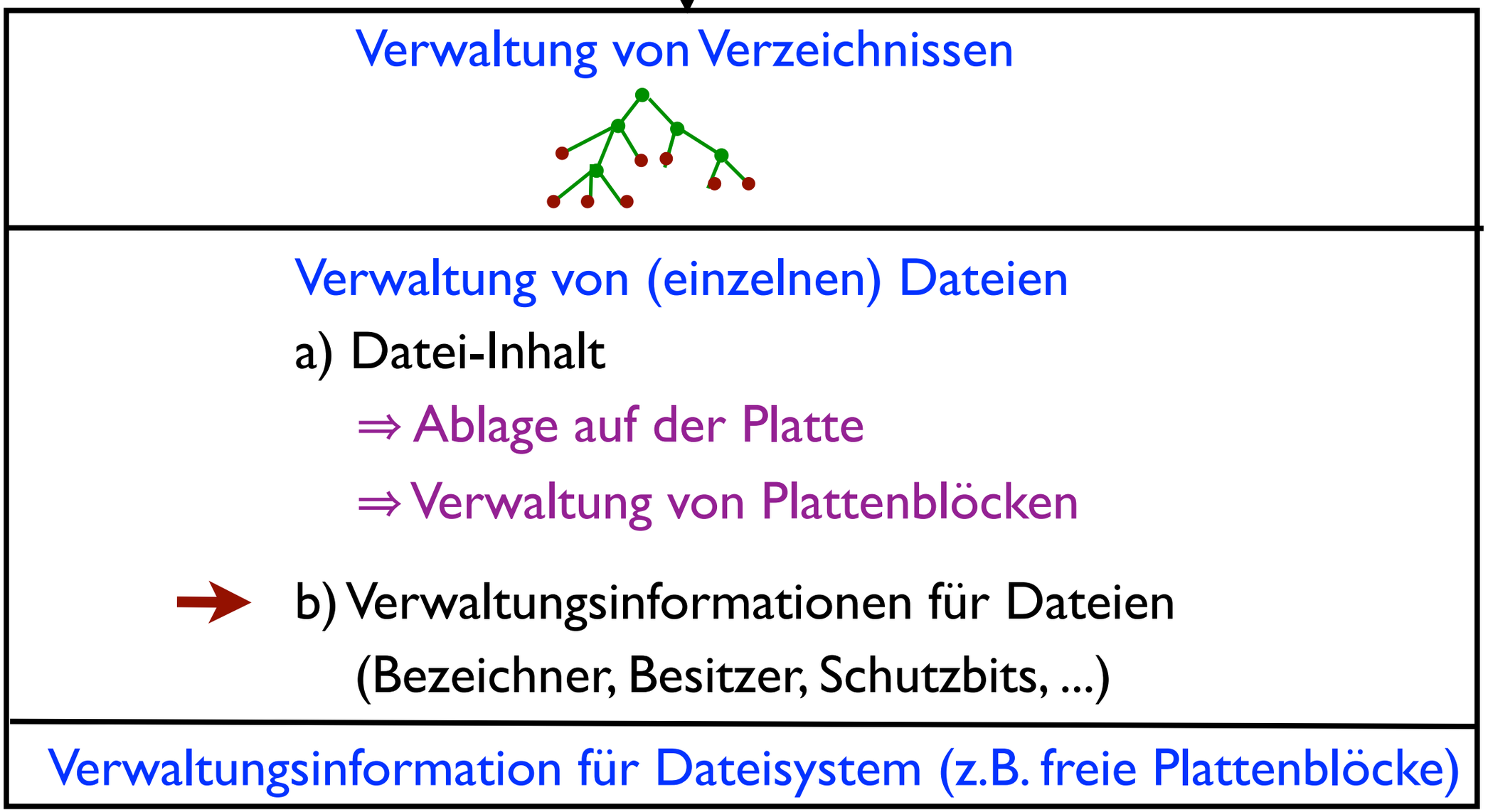
- Ähnliche Verfahren denkbar
  - Verkettete Liste
    - kein wahlfreier Zugriff erforderlich
    - allerdings langwierig für große Anfragen
  - Block-Index-Varianten

⇒ auch Mischformen denkbar



Ad 1) Logische Sicht auf ein Dateisystem

Nutzer



Organisation der Platte ⇒ später

(Bereitstellung von Plattenblöcken, Oberflächen, Spuren, Sektoren, ...)

Fragen – Teil 1

- Aus welchen grundlegenden Komponenten besteht ein Dateisystem?

Teil 2:
Klassisches Beispiel:
Unix-V7-Dateisystem

Beispiel: Das Unix-V7-Dateisystem

1) Verwaltung von Dateien

a) Datei-Inhalt: Block-orientiert mit speziellem Block-Index (s. unten)

b) Inodes: Ablage der Verwaltungsinformationen:

- Jede Datei hat genau einen Inode
- Enthält Verwaltungsinformationen für Datei:
  - eindeutiger Bezeichner (Inode-Nummer)
    ⇒ Position innerhalb der Tabelle

Beispiel: Das Unix-V7-Dateisystem

1) Verwaltung von Dateien

a) Datei-Inhalt: Block-orientiert mit speziellem Block-Index (s. unten)

b) Inodes: Ablage der Verwaltungsinformationen:

- Jede Datei hat genau einen Inode
- Enthält Verwaltungsinformationen für Datei:
  - eindeutiger Bezeichner (Inode-Nummer)
    ⇒ Position innerhalb der Tabelle
  - Anzahl der Hard Links (mehrere Namen für Datei)
    ⇒ verweisen auf denselben Inode
  - Dateityp
  - Besitzer (uid)
  - Gruppe (gid)
  - Zugriffsrechte (RWX-Bits...)
  - Anzahl der Bytes
  - Zeitpunkt des letzten Zugriffs, der letzten Änderung, ...

Beispiel: Das Unix-V7-Dateisystem

1) Verwaltung von Dateien

a) Datei-Inhalt: Block-orientiert mit speziellem Block-Index (s. unten)

b) Inodes: Ablage der Verwaltungsinformationen:

- Jede Datei hat genau einen Inode
- Enthält Verwaltungsinformationen für Datei:
  - eindeutiger Bezeichner (Inode-Nummer)
    ⇒ Position innerhalb der Tabelle
  - Anzahl der Hard Links (mehrere Namen für Datei)
    ⇒ verweisen auf denselben Inode
  - Dateityp
  - Besitzer (uid)
  - Gruppe (gid)
  - Zugriffsrechte (RWX-Bits...)
  - Anzahl der Bytes
  - Zeitpunkt des letzten Zugriffs, der letzten Änderung, ...

Aber nicht der/die Dateiname/n !

Beispiel: Das Unix-V7-Dateisystem

1) Verwaltung von Dateien

a) Datei-Inhalt: Block-orientiert mit speziellem Block-Index (s. unten)

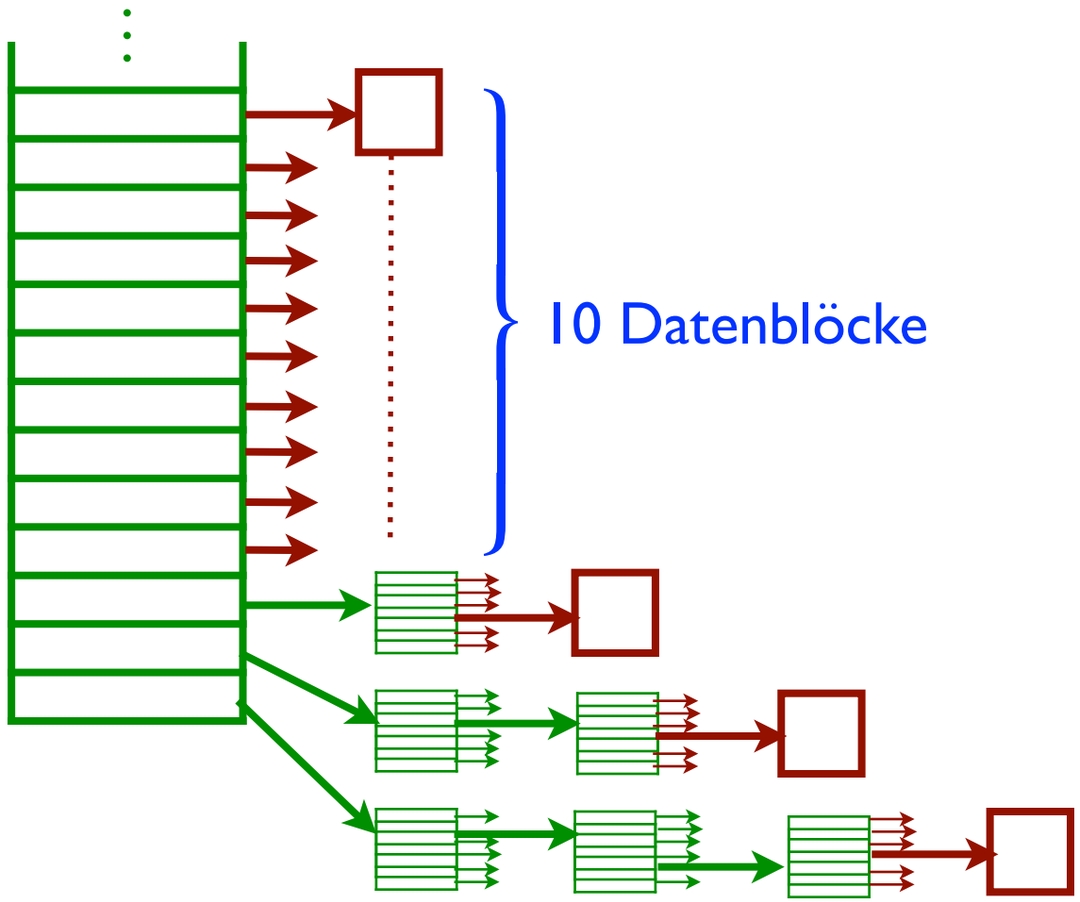
b) Inodes: Ablage der Verwaltungsinformationen:

- Jede Datei hat genau einen Inode
- Enthält Verwaltungsinformationen für Datei:
  - eindeutiger Bezeichner (Inode-Nummer)
    ⇒ Position innerhalb der Tabelle
  - Anzahl der Hard Links (mehrere Namen für Datei)
    ⇒ verweisen auf denselben Inode
  - Dateityp
  - Besitzer (uid)
  - Gruppe (gid)
  - Zugriffsrechte (RWX-Bits...)
  - Anzahl der Bytes
  - Zeitpunkt des letzten Zugriffs, der letzten Änderung, ...
  - Verweise auf Datenblöcke

Aber nicht der/die Dateiname/n !

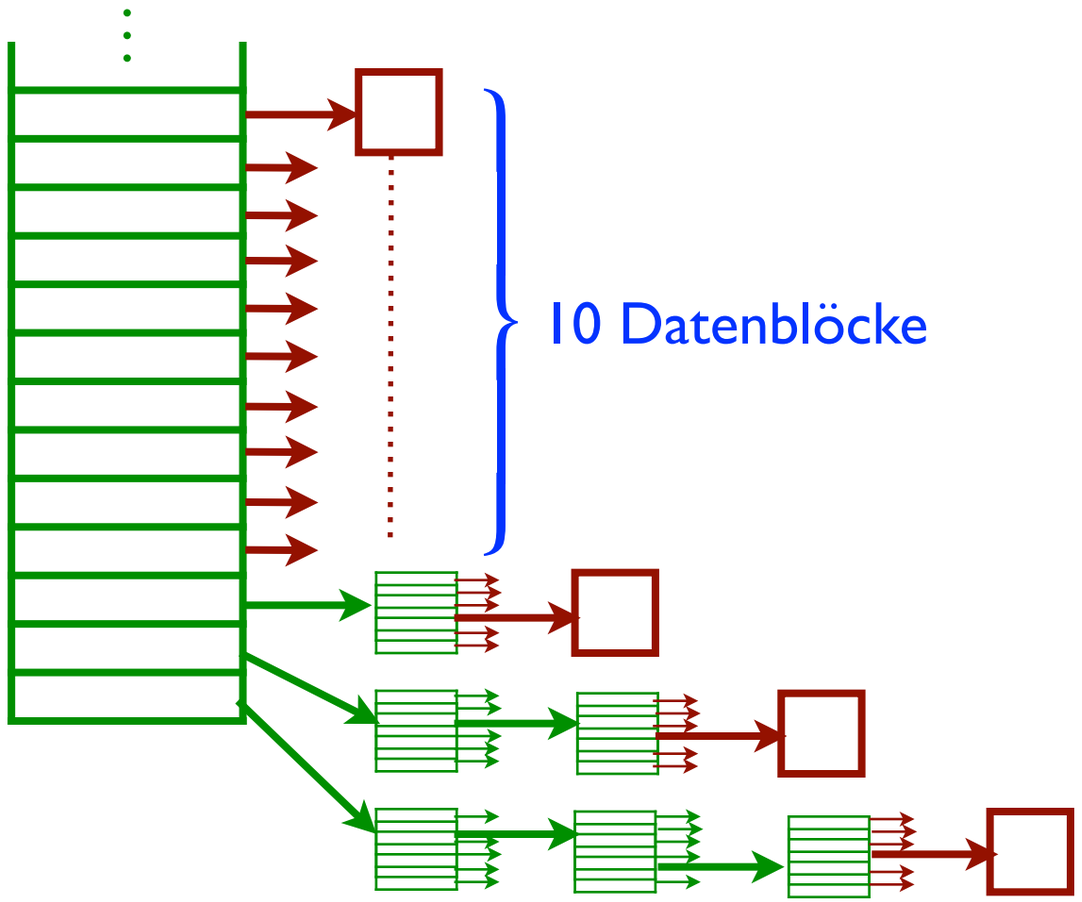
Verweise auf Datenblöcke

- 10 direkte Verweise
- 1 indirekter Verweis
- 1 doppelt indirekter Verweis
- 1 dreifach indirekter Verweis



Verweise auf Datenblöcke

- 10 direkte Verweise
- 1 indirekter Verweis
- 1 doppelt indirekter Verweis
- 1 dreifach indirekter Verweis



- Annahme: z.B. 256 Einträge pro Indirektblock

  $\Rightarrow 10 + 256 + 256^2 + 256^3$ Datenblöcke

- Direkter (schneller) Zugriff auf Blöcke kleiner Dateien
- Sehr große Dateien realisierbar
- Maximal vier Zugriffe für einen Dateiblock

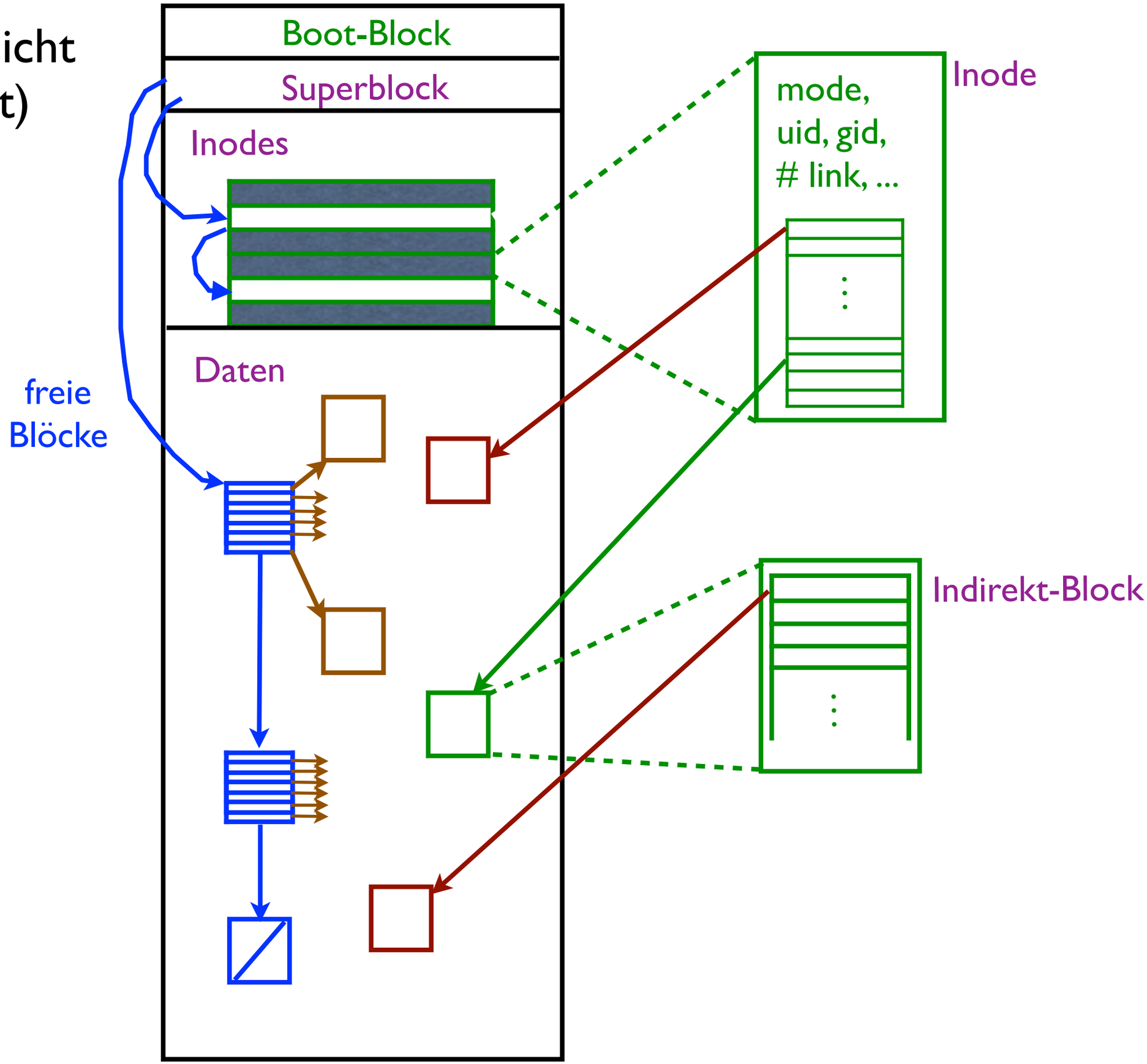
Boot-Block

- Block 0 des „Root“-Dateisystems
- Wird beim Booten geladen

Superblock

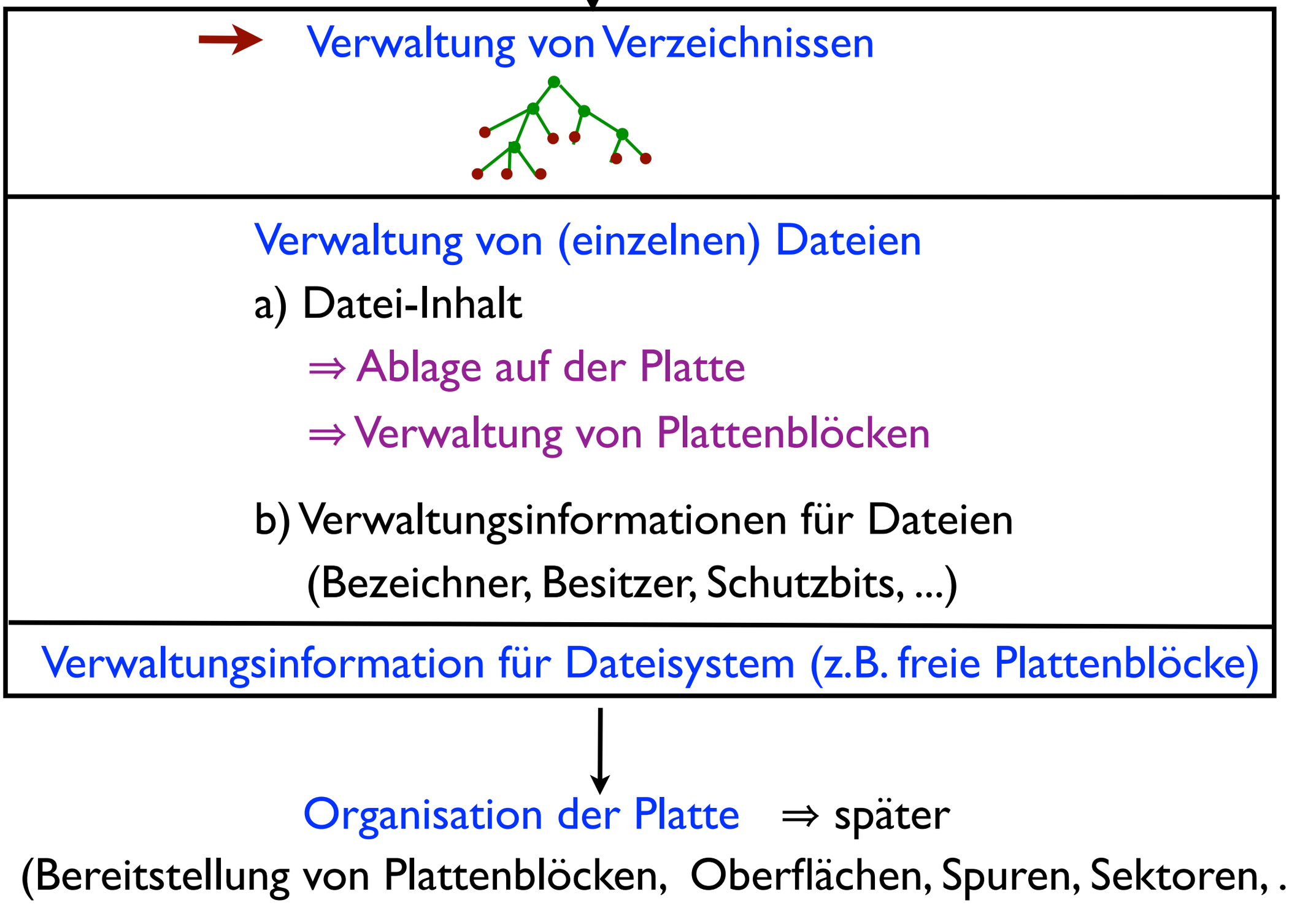
- Verwaltungsinformationen des Dateisystems:
  - Größe
  - Verwaltung der freien Inodes
  - Verwaltung der freien Blöcke
    (Verkettete Liste von Blöcken mit freien Blocknummern)

Gesamtübersicht (vereinfacht)



Ad 1) Logische Sicht auf ein Dateisystem

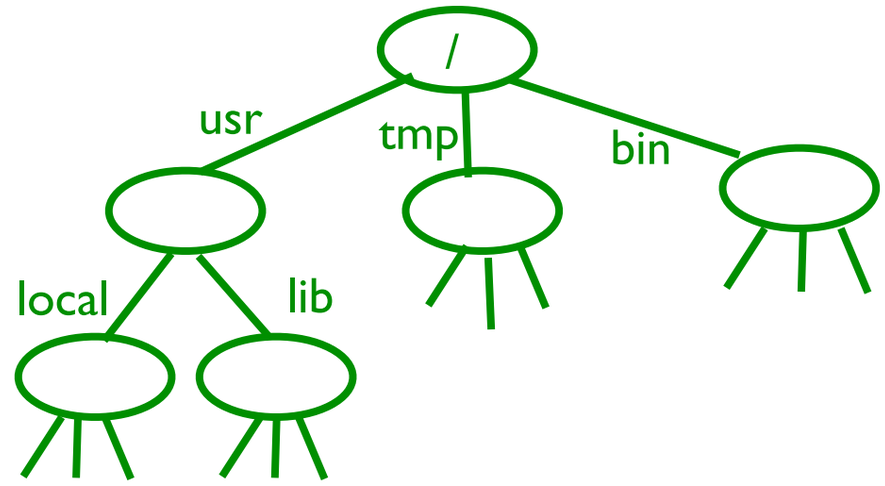
Nutzer



Organisation der Platte ⇒ später

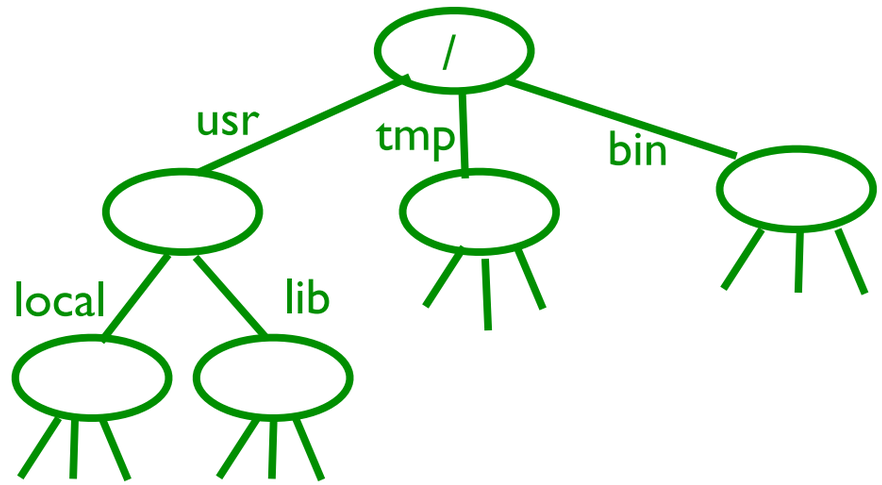
(Bereitstellung von Plattenblöcken, Oberflächen, Spuren, Sektoren, ...)

Verwaltung von Verzeichnissen

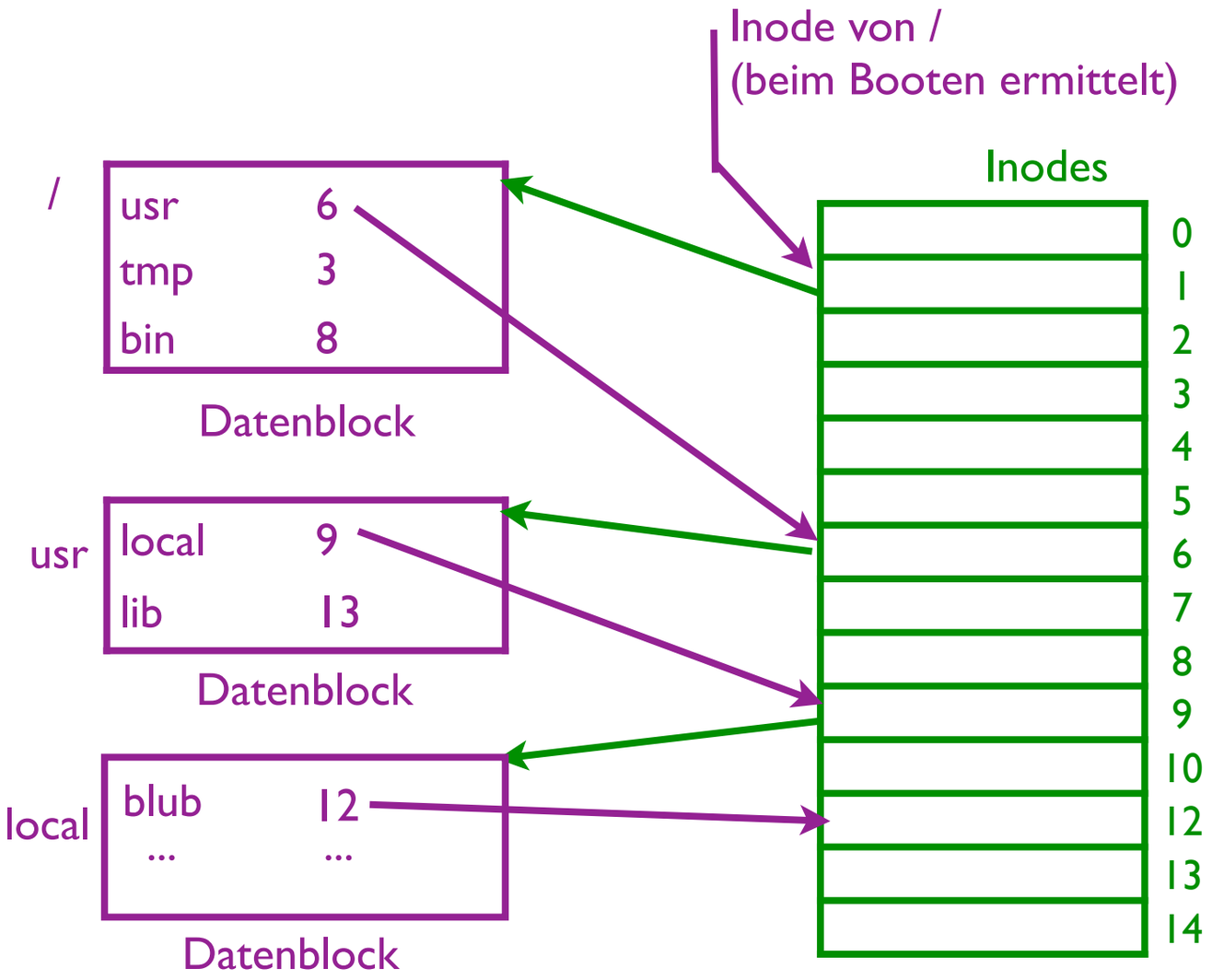


- Verzeichnisse sind Dateien (Inode, Datenblöcke,...)
- Folgen von Einträgen
- Jeder Eintrag enthält:
  - Dateiname („nächste“ Pfadkomponente)
  - Inode-Nummer

Verwaltung von Verzeichnissen



- Verzeichnisse sind Dateien (Inode, Datenblöcke,...)
- Folgen von Einträgen
- Jeder Eintrag enthält:
  - Dateiname („nächste“ Pfadkomponente)
  - Inode-Nummer
- Beispiel:
  open /usr/local/blub



Kleine Aufgabe

Mit dem Shell-Kommando `ls -l` wird der folgende Inhalt eines Beispielverzeichnisses angezeigt:

```
-rw-------  1 anna  stud  227243 Oct 25 20:29 bild.jpg
-rw-r-----  1 anna  stud   50110 Nov  3 10:30 notes.txt
drwsr-s---  2 anna  pi3-15    68 Feb  2  2017 pi3
drwxr-xr--  5 anna  stud     512 Oct 23 07:55 ti2-uebungen
-rw-rw-rw-  1 anna  stud   77168 Sep 17 17:23 vortrag.tex
```

a) Wie ändert sich diese Anzeige nach Ausführung der folgenden Shell-Kommandos:

- `chmod 600 vortrag.tex`
- `ln bild.jpg pi3/ueb3-bild.jpg`
- `chgrp ti2 ti2-uebungen`
- `cp notes.txt notizen.txt`

b) Wo verwaltet Unix diese Informationen?

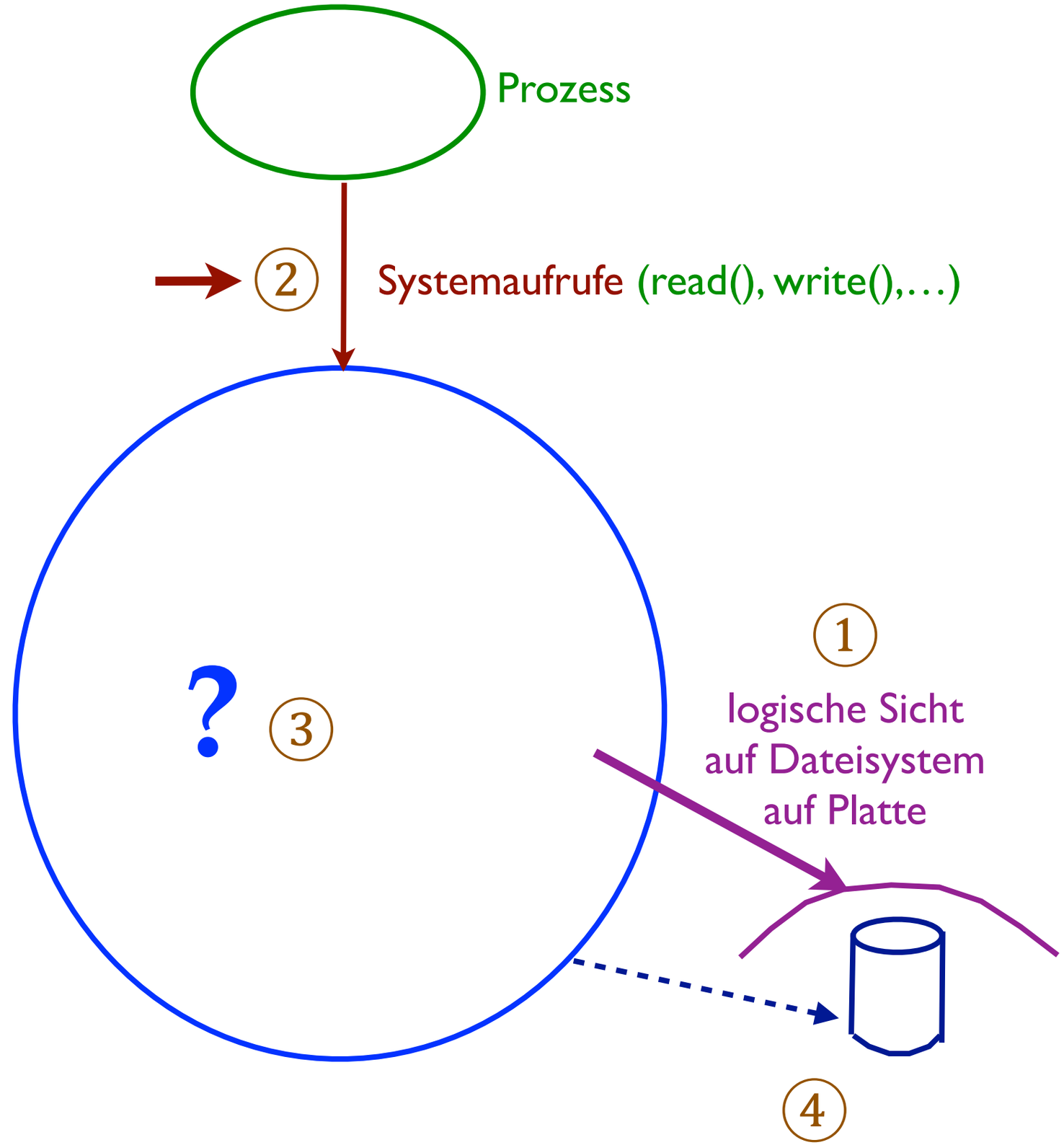
Fragen – Teil 2

- Wie sieht die Struktur des Unix-V7-Dateisystems auf der Platte in etwa aus? Warum erfolgt die Verwaltung der Freispeicherliste über Indirekt-Blöcke?
- Welche Angaben enthält ein *Inode*? Welche Angaben enthält eine *Verzeichnis-Datei*?

Teil 3:
Systemaufrufe zur Dateiverwaltung

10 Dateiverwaltung (1), Ute Bormann

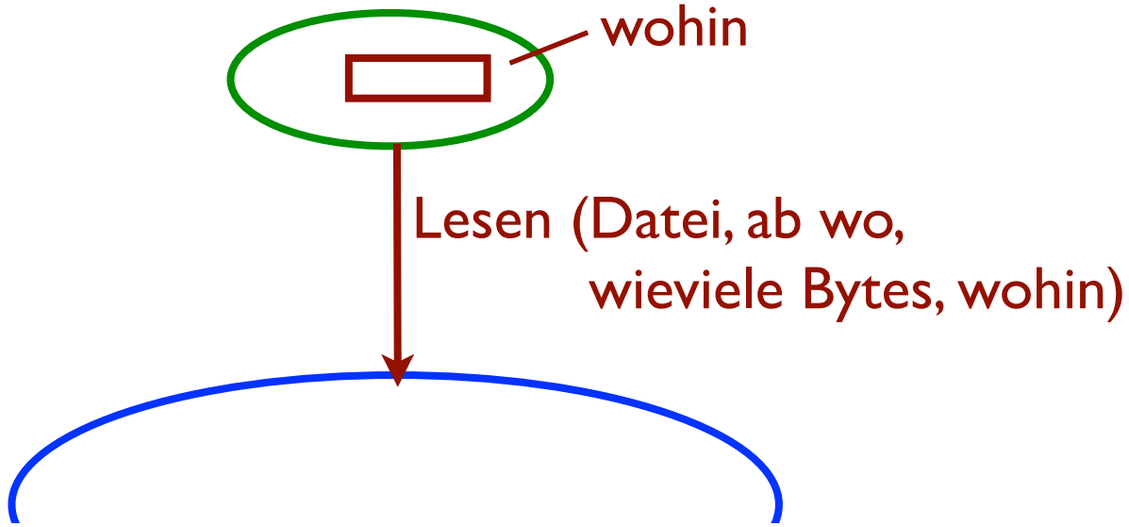
Überblick Dateiverwaltung



Ad 2) Zugriffsoperationen auf Dateien

⇒ Systemaufrufschnittstelle in Unix

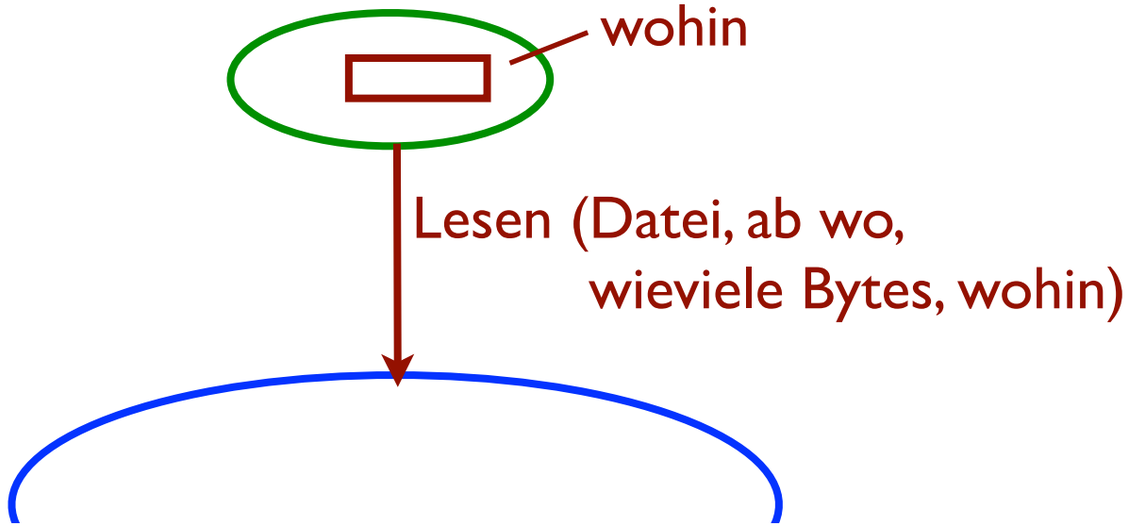
Beispiel: „Lesen aus Datei“
⇒ naive Lösung



Ad 2) Zugriffsoperationen auf Dateien

⇒ Systemaufrufschnittstelle in Unix

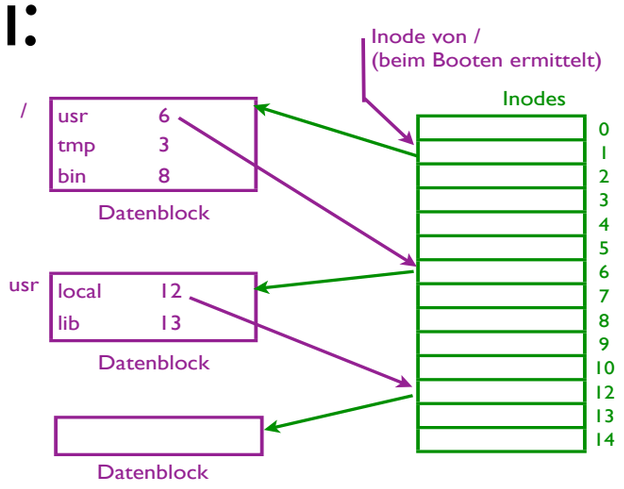
Beispiel: „Lesen aus Datei“
⇒ naive Lösung



Eine solche Schnittstelle würde nicht berücksichtigen:

a) Häufig mehrere Lesevorgänge auf dieselbe Datei:

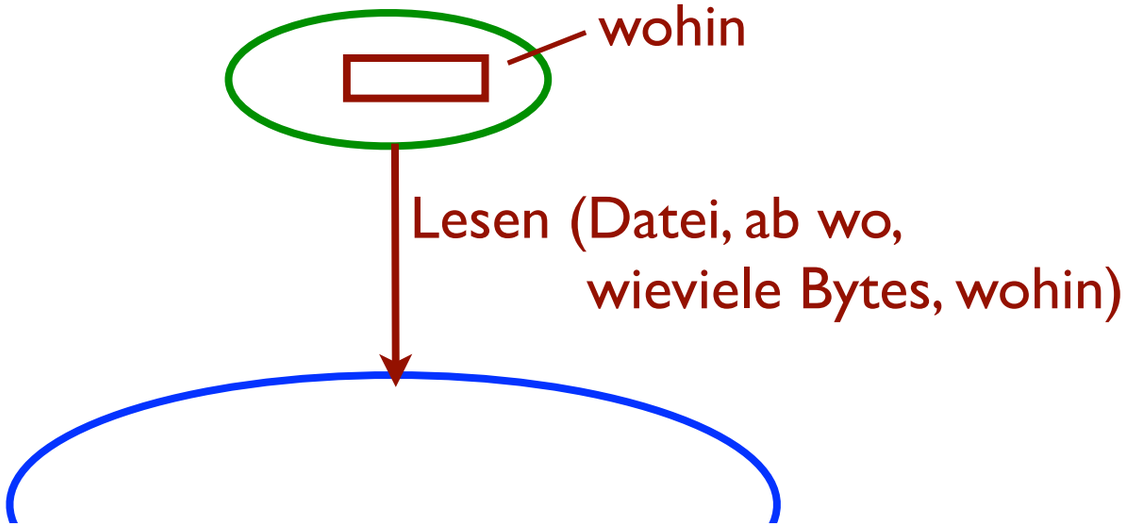
⇒ Im Kern: Zugriff optimieren
⇒ Zustandsinformationen verwalten



Ad 2) Zugriffsoperationen auf Dateien

⇒ Systemaufrufschnittstelle in Unix

Beispiel: „Lesen aus Datei“
⇒ naive Lösung



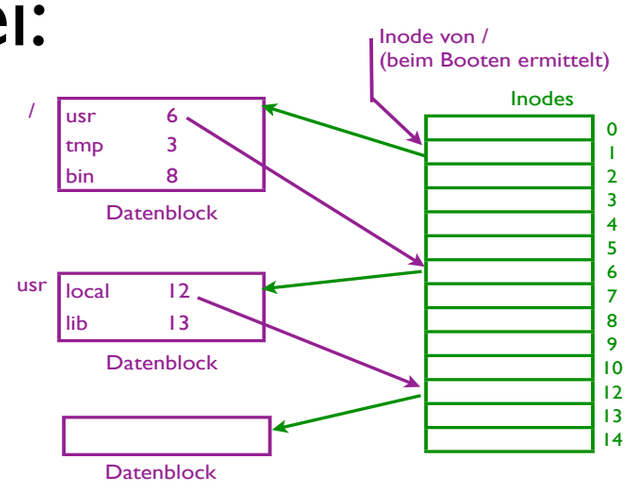
Eine solche Schnittstelle würde nicht berücksichtigen:

a) Häufig mehrere Lesevorgänge auf dieselbe Datei:

⇒ Im Kern: Zugriff optimieren
⇒ Zustandsinformationen verwalten

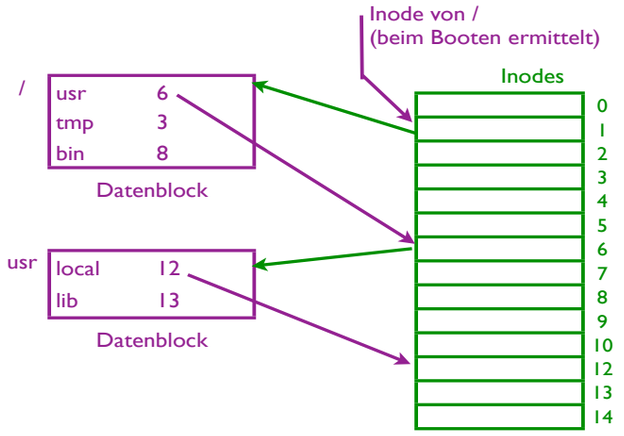
b) Häufig sequentielles Lesen:

⇒ Default: ab „aktueller“ Position weiterlesen
⇒ explizites Verschieben der Position



Öffnen von Dateien (für die weitere Arbeit)

open (path, flags, mode)



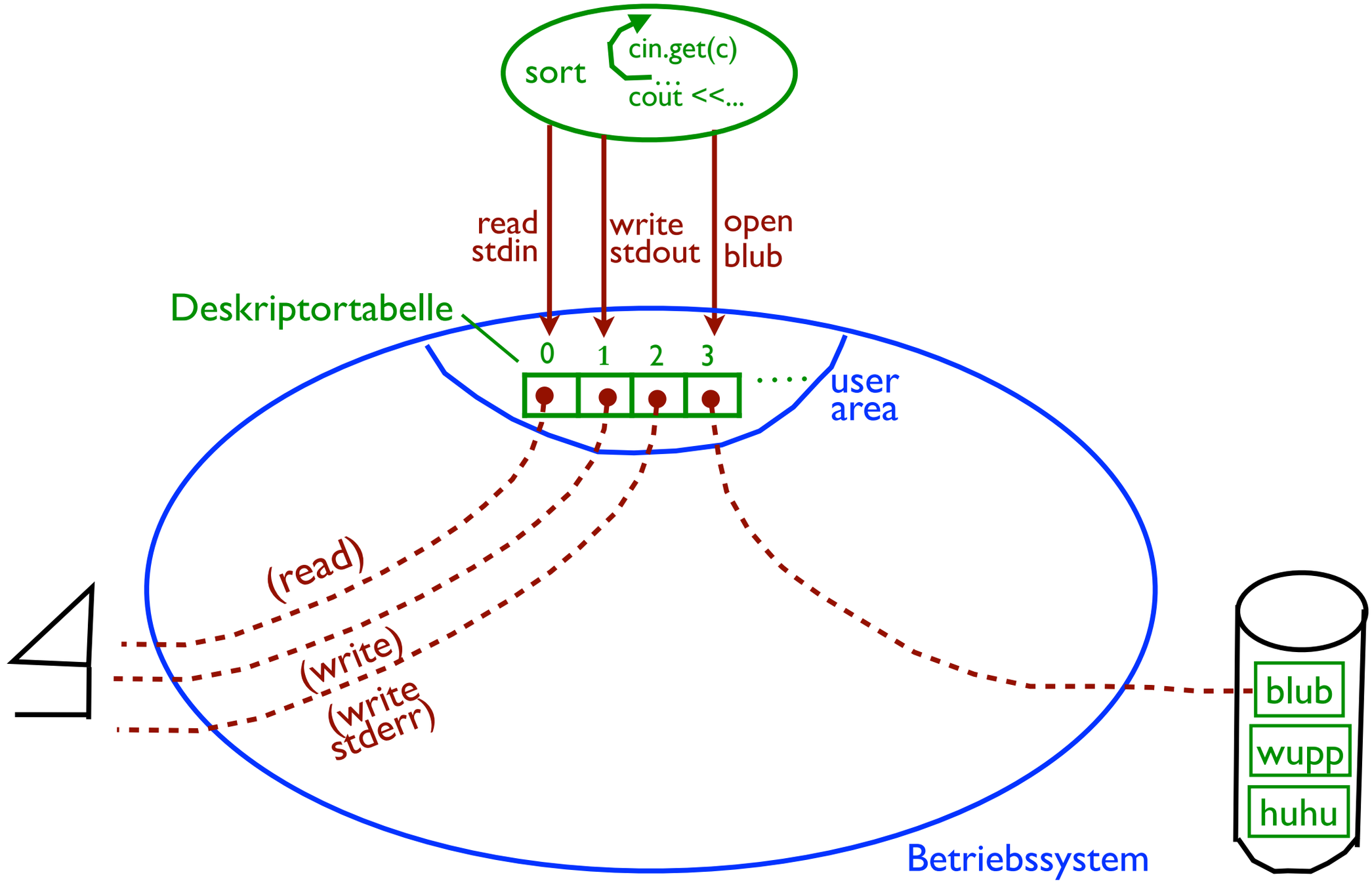
- path: Pfadname der Datei
  (⇒ komponentenweise abarbeiten)

altern.: relativer Pfadname

- flags: welche Folgeoperationen erlaubt:
  - lesen, schreiben, am Ende anfügen...
- mode: Zugriffsrechte bei neu erzeugter Datei:
  - RWX für User/Group/World...

⇒ liefert File Descriptor (fd):
Kurzbezeichnung für weiteren Zugriff

Wdh.: Verwaltung im Betriebssystemkern (vereinfacht)



Lesen von Dateien

read (fd, buf, len)

- Lesen von len Bytes „ab der aktuellen Position“ der Datei fd in den Puffer buf
- aktuelle Position wird um gelesene Bytes weitergeschoben (sequentielles Lesen)

⇒ liefert Anzahl der tatsächlich gelesenen Bytes

Lesen von Dateien

read (fd, buf, len)

- Lesen von len Bytes „ab der aktuellen Position“ der Datei fd in den Puffer buf
- aktuelle Position wird um gelesene Bytes weitergeschoben (sequentielles Lesen)

⇒ liefert Anzahl der tatsächlich gelesenen Bytes

Schreiben der Datei

write (fd, buf, len)

- analog zu Lesen

⇒ liefert Anzahl der geschriebenen Bytes

Positionieren in der Datei

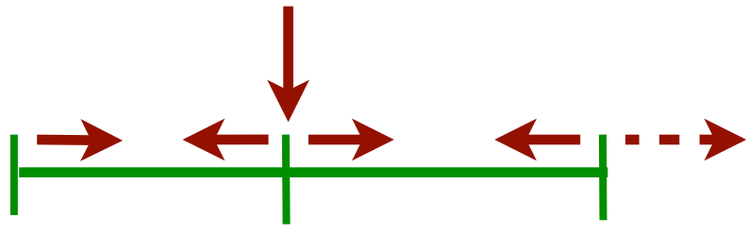
lseek (fd, offset, whence)

- Aktuelle Position in der Datei fd wird um offset Bytes verschoben gemäß whence:

  = SEEK_SET vom Anfang

  = SEEK_CUR von aktueller Position

  = SEEK_END vom Ende



⇒ liefert neue Position

(ggf. auch über das aktuelle Ende hinaus ⇒ „Löcher“)

Positionieren in der Datei

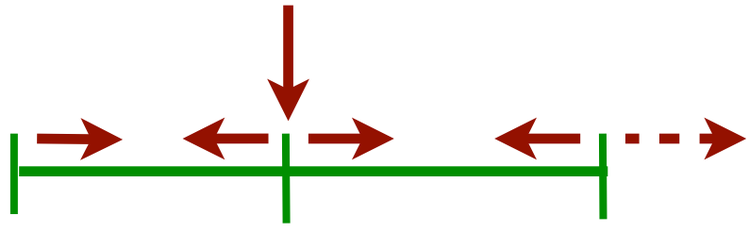
lseek (fd, offset, whence)

- Aktuelle Position in der Datei fd wird um offset Bytes verschoben gemäß whence:

  = SEEK_SET vom Anfang

  = SEEK_CUR von aktueller Position

  = SEEK_END vom Ende



⇒ liefert neue Position

(ggf. auch über das aktuelle Ende hinaus ⇒ „Löcher“)

Schließen der Datei (bis zum nächsten Öffnen)

close (fd)

Typische Abfolgen der Zugriffsoperationen (vereinfacht)

a) Sequentielles Schreiben

```
open ("/usr/bla",...);      // ⇒ fd
write (fd, buf, 100);       // 100 Bytes schreiben
write (fd, buf, 100);
...
close (fd);
```

Typische Abfolgen der Zugriffsoperationen (vereinfacht)

a) Sequentielles Schreiben

```
open ("/usr/bla",...);      // ⇒ fd
write (fd, buf, 100);       // 100 Bytes schreiben
write (fd, buf, 100);
...
close (fd);
```

b) Wahlfreies Lesen

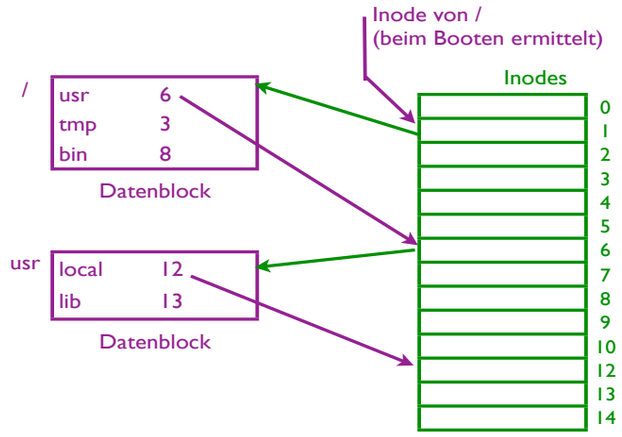
```
open ("/usr/bla",...);      //  ⇒ fd
lseek (fd, 500, SEEK_SET); // ab Anfang
read (fd, buf, 27);         // 27 Bytes lesen
lseek (fd, 1500, SEEK_SET);
read (fd, buf, 27);
...
close (fd);
```

Weitere Zugriffsoperationen:

Zustandsinformationen über Datei erfragen

stat (path, buf)

- Infos aus Inode der Datei path
  Inode-Nummer, mode, uid, gid, size, nlink,...
- Ergebnis wird in buf abgelegt
- wird z.B. für ls -l verwendet



Weitere Zugriffsoperationen:

Zustandsinformationen über Datei erfragen

stat (path, buf)

- Infos aus Inode der Datei path
  Inode-Nummer, mode, uid, gid, size, nlink,...
- Ergebnis wird in buf abgelegt
- wird z.B. für ls -l verwendet

Erzeugen/Löschen von Verzeichnissen

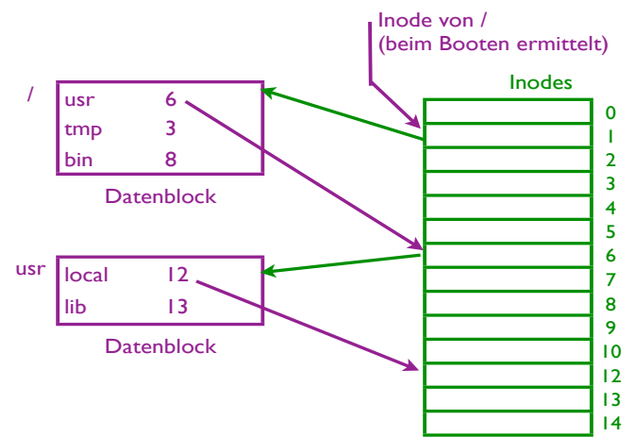
mkdir (path, mode)

rmdir (path)

Erzeugen/Löschen von Hard Links

link (path1, path2)

unlink (path)



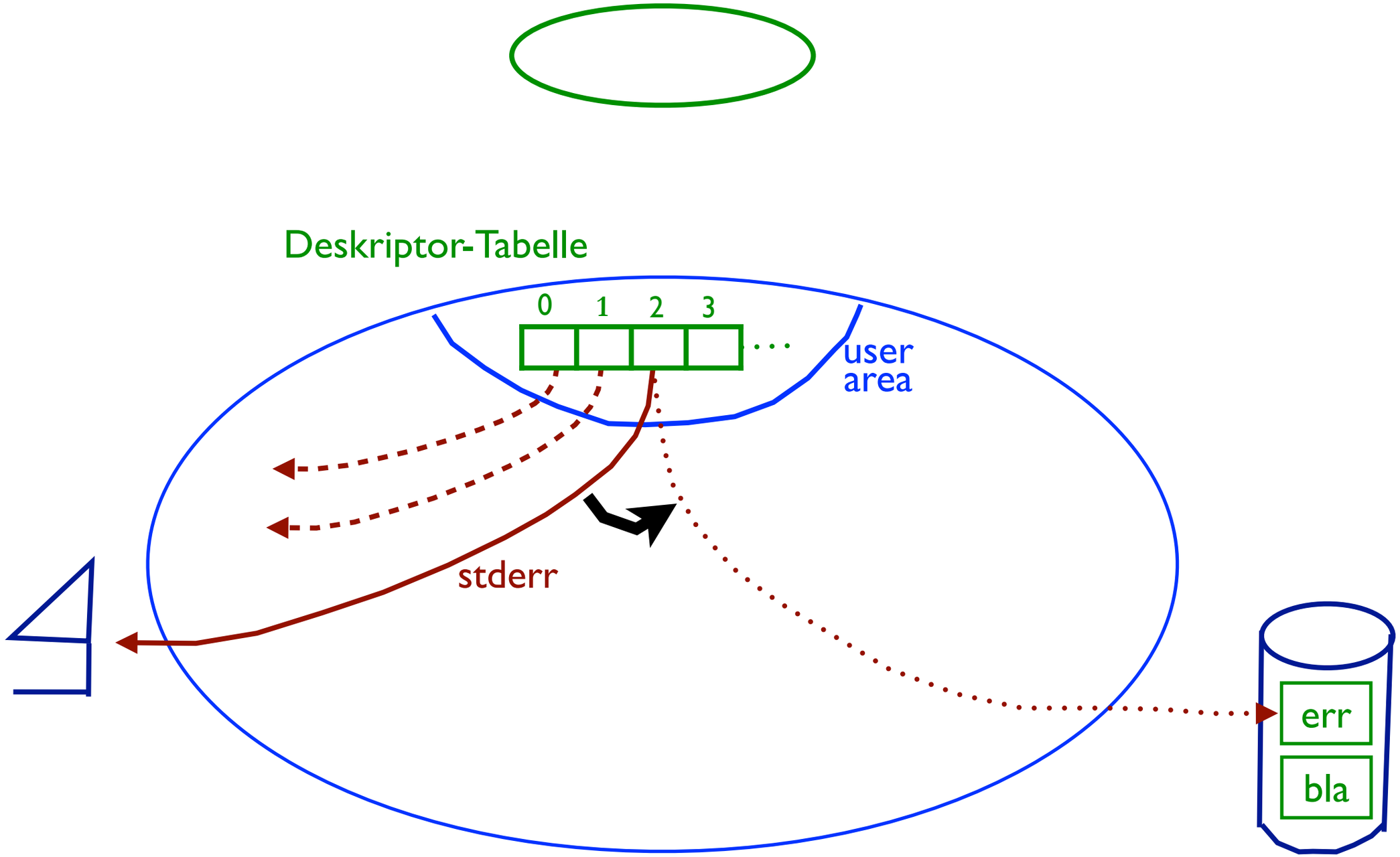
Duplizieren von Dateideskriptoren

newfd = dup (fd)

- Wird z.B. zur Realisierung der Ein-/Ausgabeumlenkung verwendet

  Beispiel: bla 2> err

Verwaltung im Betriebssystemkern



Duplizieren von Dateideskriptoren

newfd = dup (fd)

- Wird z.B. zur Realisierung der Ein-/Ausgabeumlenkung verwendet

  Beispiel: bla 2> err

- Naive Lösung:

```
close (2);                    // stderr schließen
fd = open (“err”, ...);  // 1. freier Deskriptor ist 2
```

Duplizieren von Dateideskriptoren

newfd = dup (fd)

- Wird z.B. zur Realisierung der Ein-/Ausgabeumlenkung verwendet

  Beispiel: bla 2> err

- Naive Lösung:

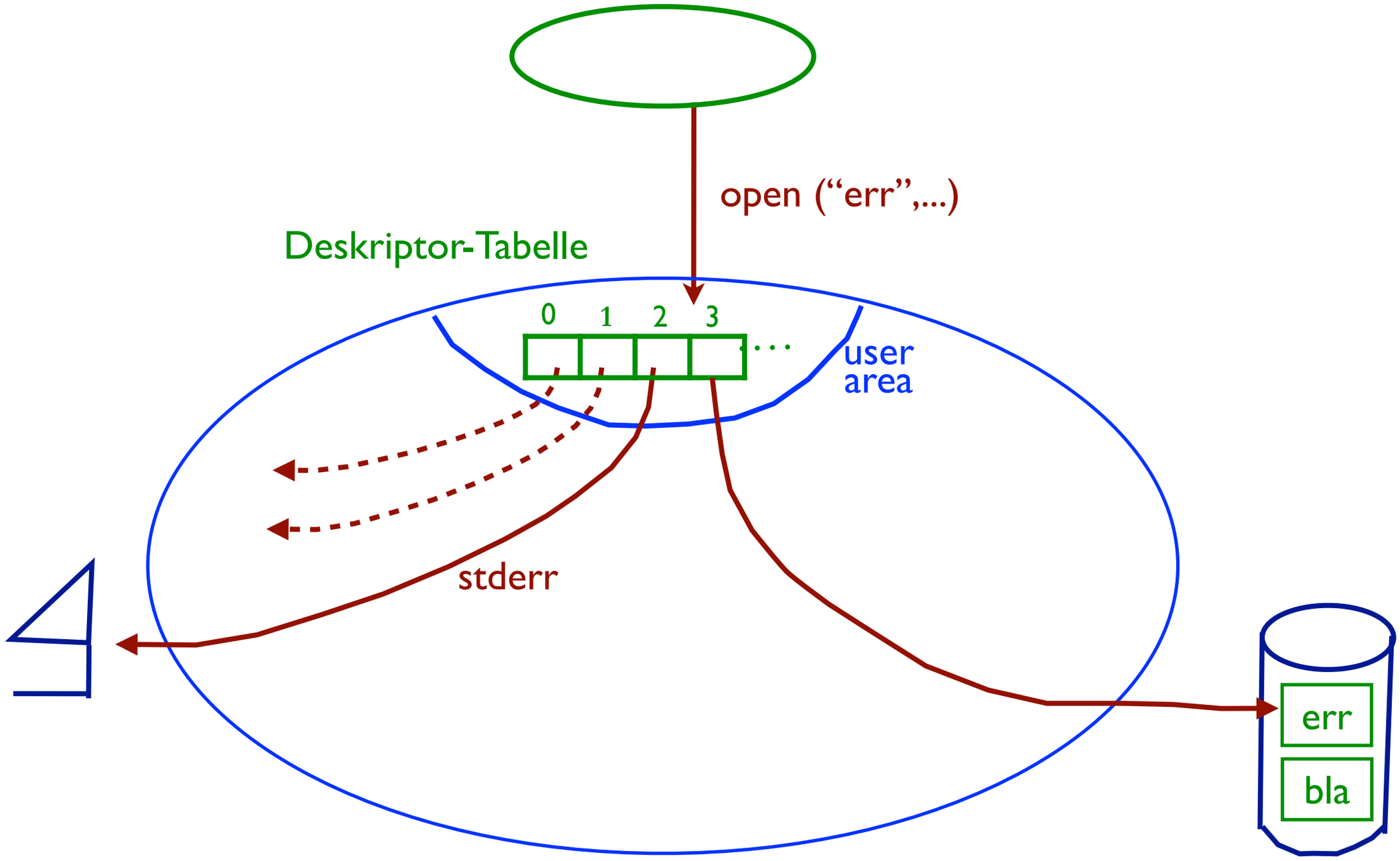
```
close (2);                    // stderr schließen
fd = open (“err”, ...);  // 1. freier Deskriptor ist 2
```

- Problem: Falls open() schiefgeht, kann Fehlermeldung nicht in stderr geschrieben werden

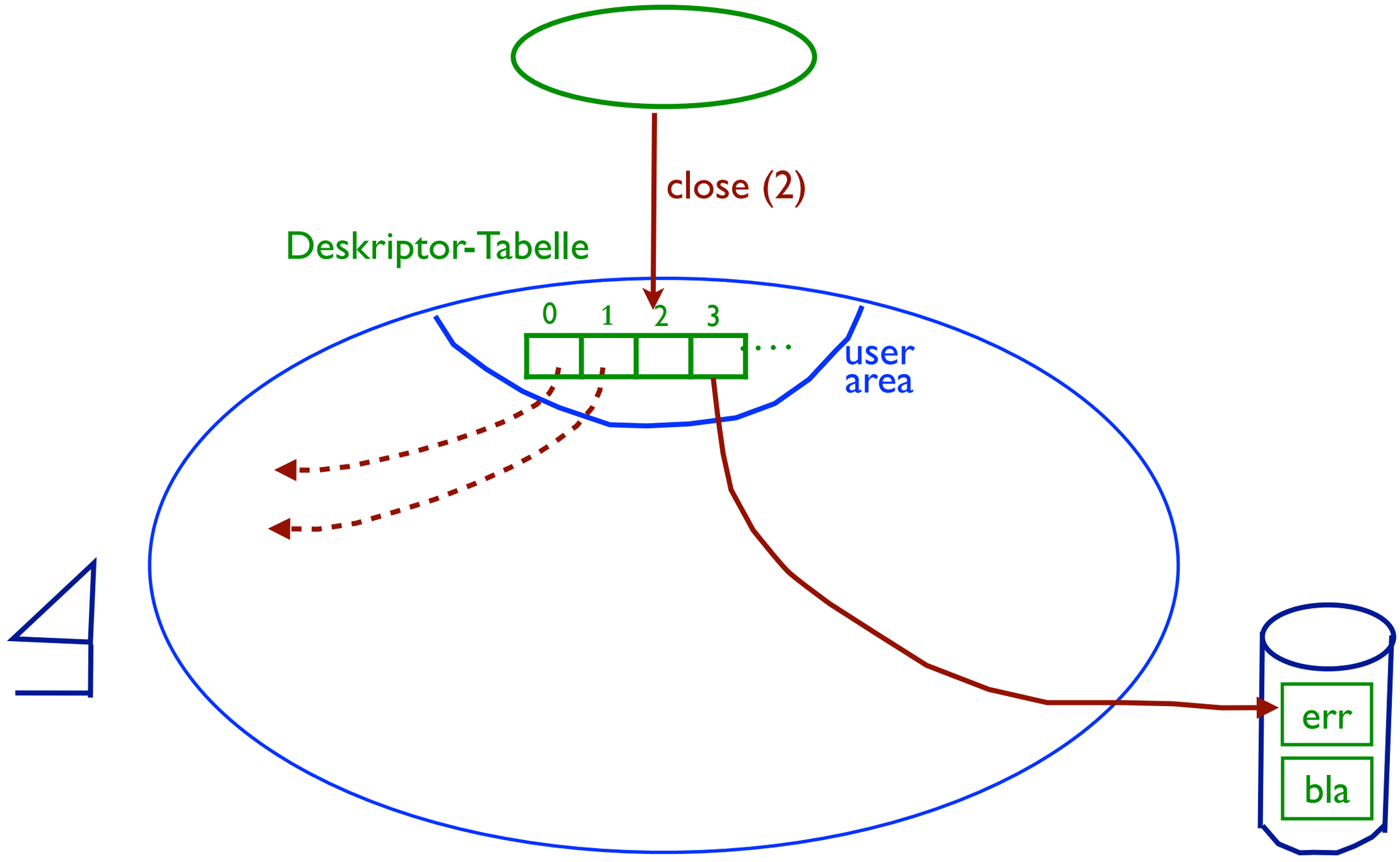
- Stattdessen:

```
fd = open (“err”, ...);
close (2);
fd2 = dup (fd);
close (fd);
```

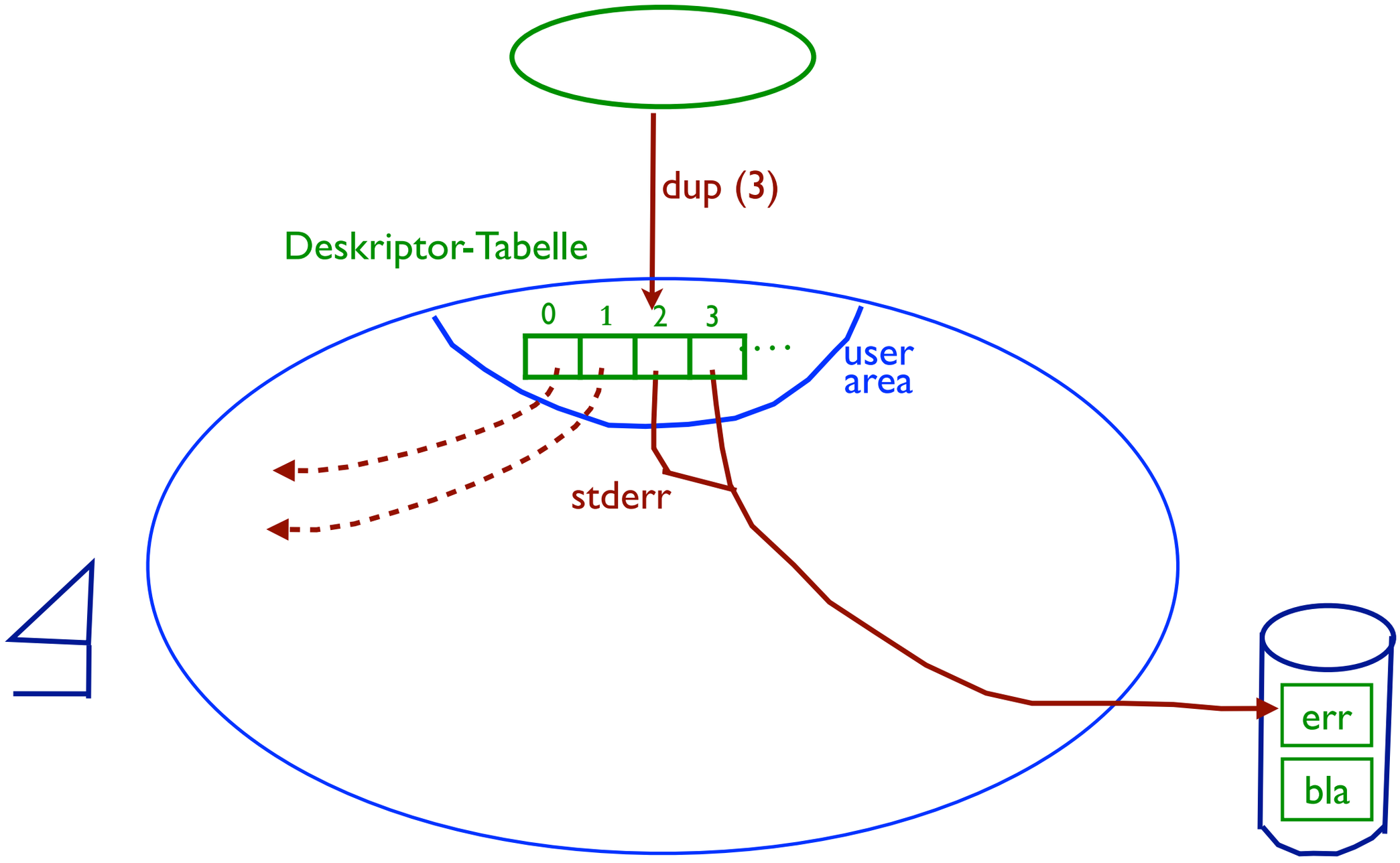
Verwaltung im Betriebssystemkern



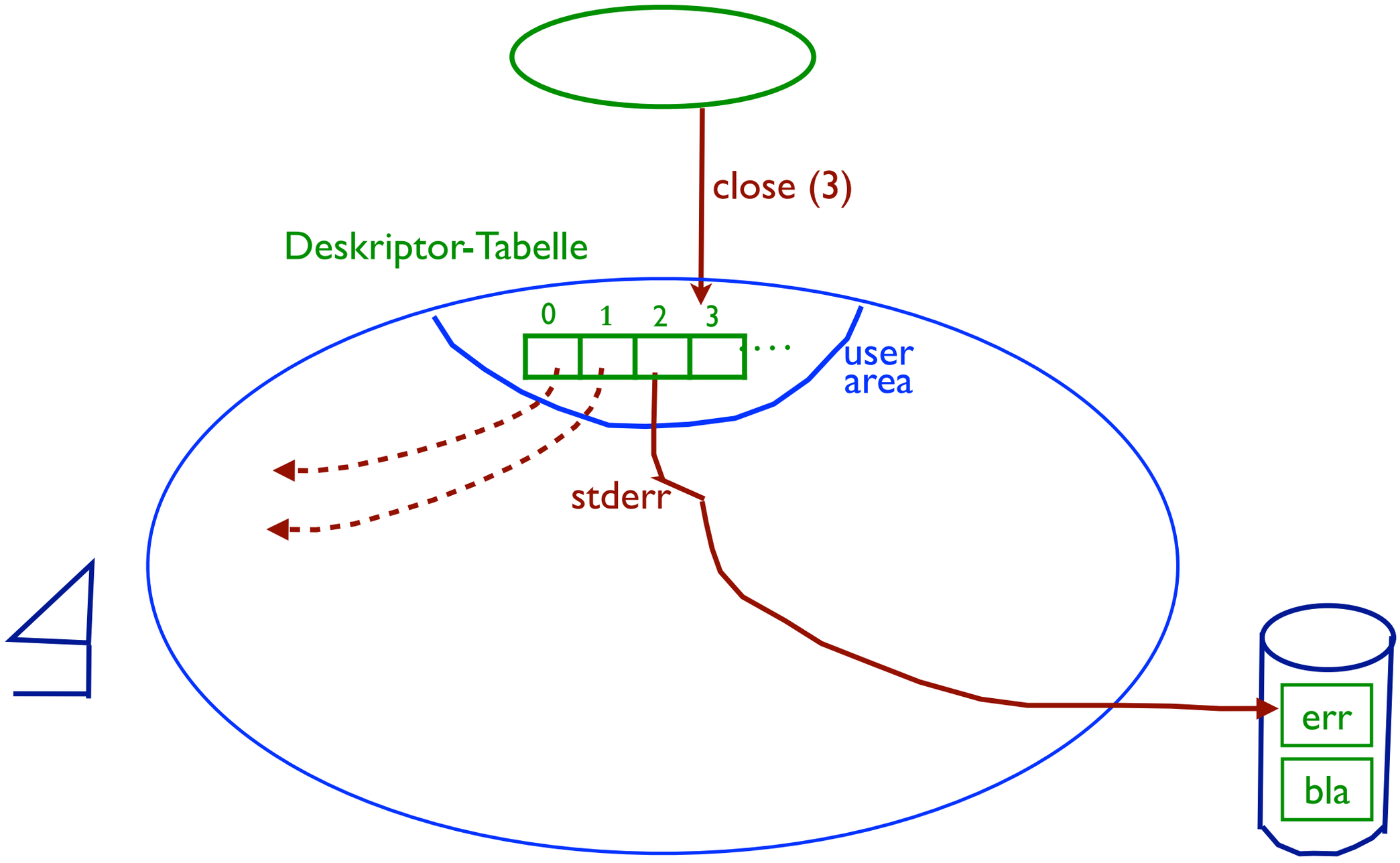
Verwaltung im Betriebssystemkern



Verwaltung im Betriebssystemkern



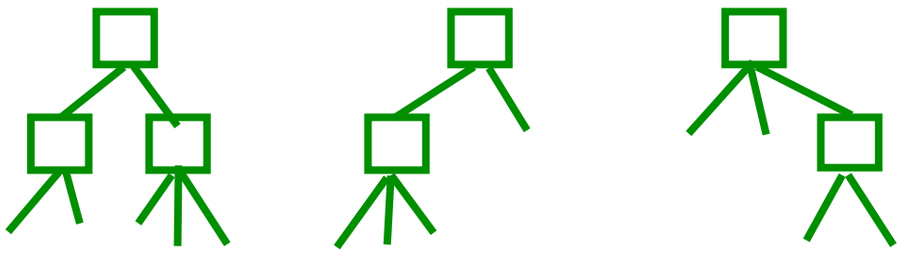
Verwaltung im Betriebssystemkern



„Zusammenbauen“ eines Unix-Dateisystems

- Rechner kann mehrere physische Platten besitzen
- Physische Platte kann in mehrere logische Platten unterteilt sein (Partitionen)
- Auf einer (logischen) Platte gespeicherter Datei„baum“

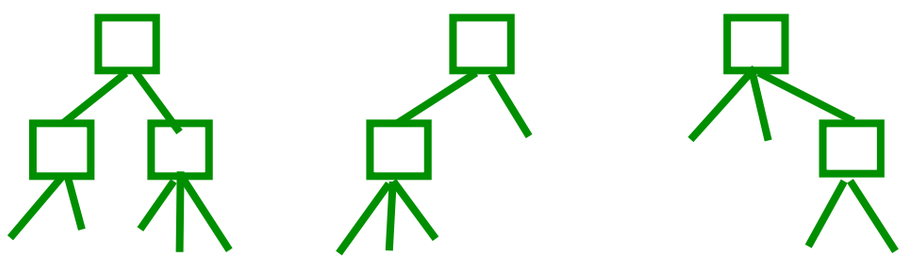
  ⇒ einzelnes Dateisystem



„Zusammenbauen“ eines Unix-Dateisystems

- Rechner kann mehrere physische Platten besitzen
- Physische Platte kann in mehrere logische Platten unterteilt sein (Partitionen)
- Auf einer (logischen) Platte gespeicherter Datei„baum“

  ⇒ einzelnes Dateisystem

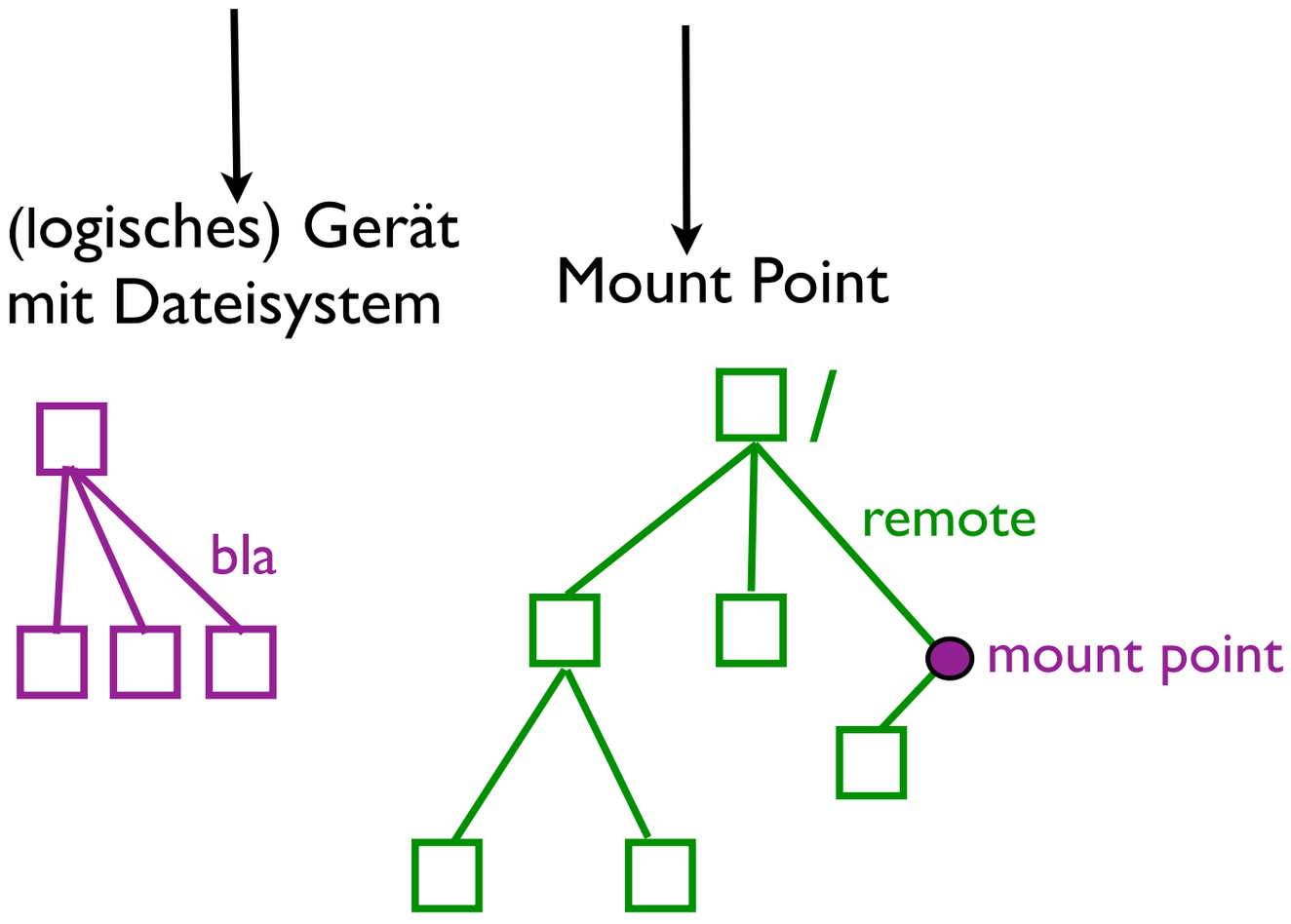


⇒ Gesamt-Dateisystem eines Unix-Systems kann aus mehreren Einzel-Dateisystemen zusammengesetzt werden

- Root-Dateisystem bei Systemstart bestimmt

- Weitere Dateisysteme können an speziellen Blättern (Mount Points) eingehängt werden

mount (spec-path, dir-path, option)

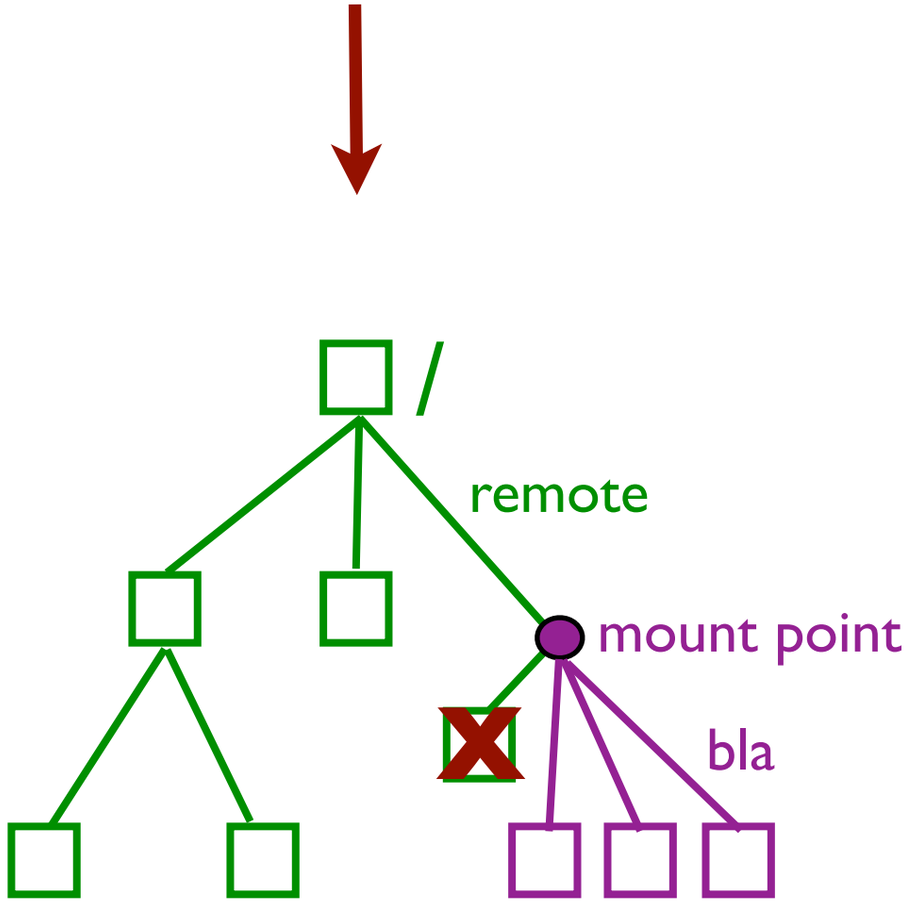


- Weitere Dateisysteme können an speziellen Blättern (Mount Points) eingehängt werden

mount (spec-path, dir-path, option)

(logisches) Gerät mit Dateisystem

Mount Point



- Weitere Dateisysteme können an speziellen Blättern (Mount Points) eingehängt werden

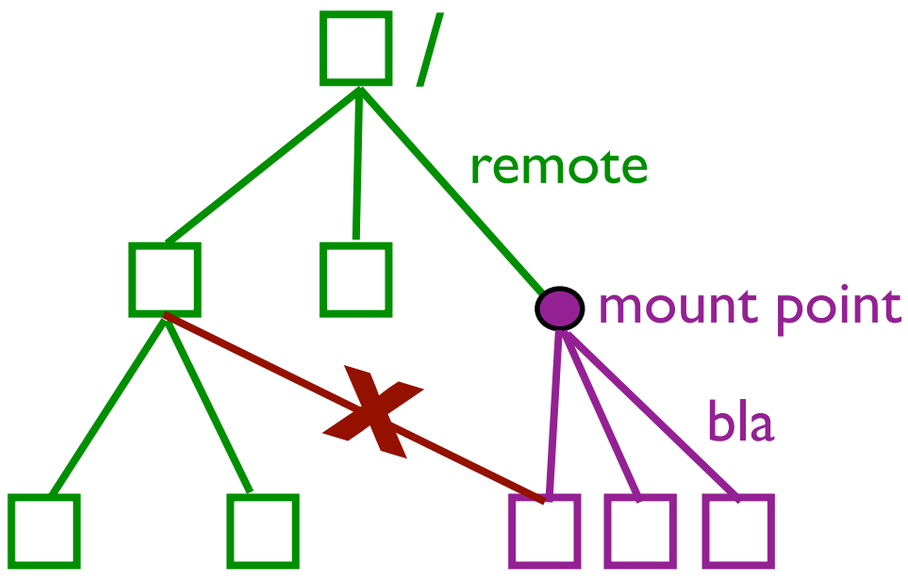
mount (spec-path, dir-path, option)

(logisches) Gerät mit Dateisystem

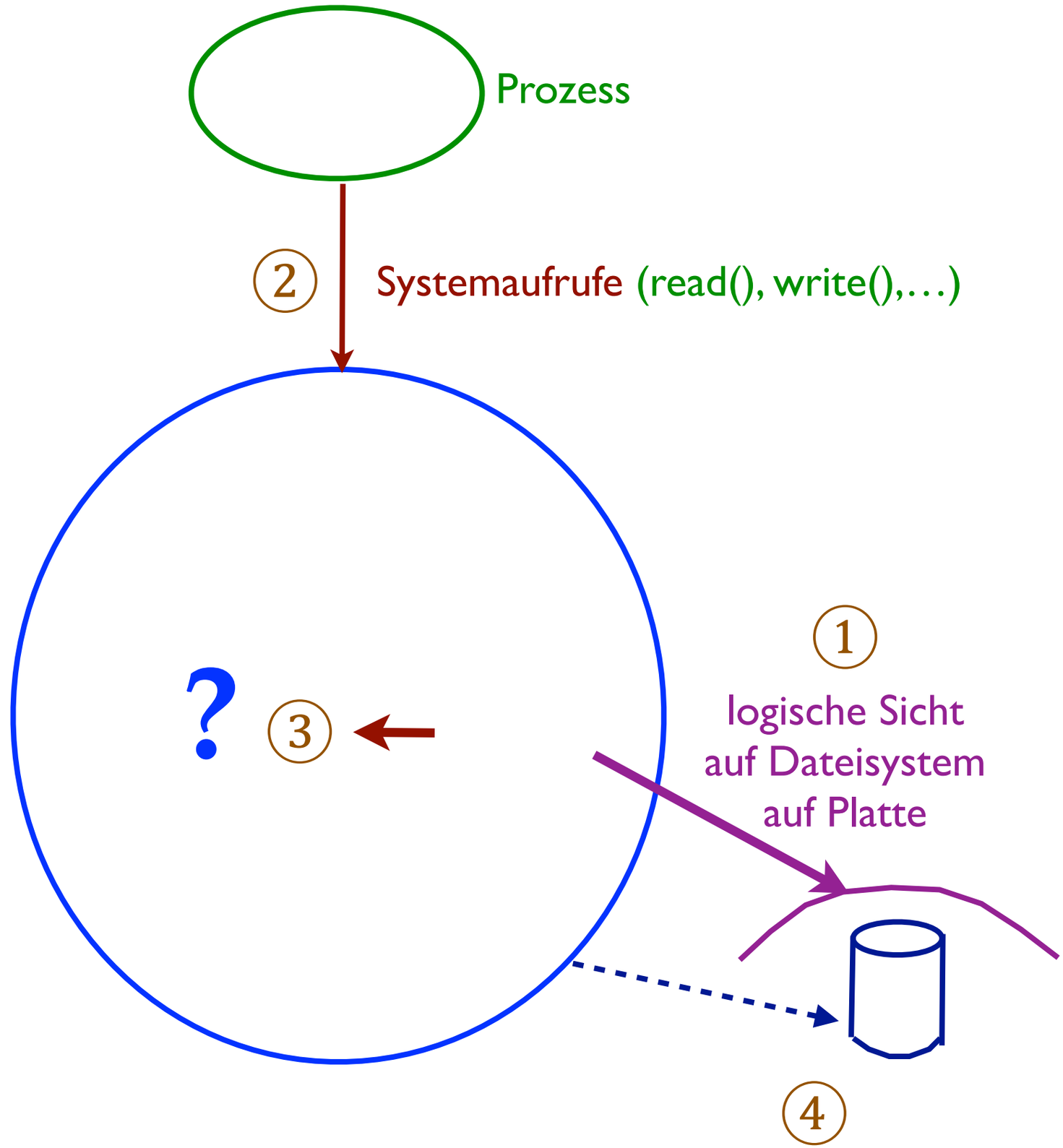
Mount Point

- Inode-Nummern sind Dateisystem-lokal

⇒ Keine übergreifenden Hard Links möglich



Überblick Dateiverwaltung



Fragen – Teil 3

- Beschreibe kurz die Zugriffsoperationen `open()`, `close()`, `lseek()`, `read()` und `write()` auf ein Unix-Filesystem. Welche Rolle spielt der *Filedeskriptor* dabei?
- Was geschieht durch einen `mount()`-Systemaufruf in etwa?

Zusammenfassung

- Ablage von Dateiblöcken auf der Platte
- Inodes
- Realisierung der Verzeichnis-Struktur
- Erstes Beispiel: Unix-V7-Dateisystem
- Systemaufrufe: open()/close(), read()/write()/lseek()...

Dateiverwaltung 1 – Fragen

1. Aus welchen grundlegenden Komponenten besteht ein Dateisystem?

2. Wie sieht die Struktur des Unix-V7-Dateisystems auf der Platte in etwa aus? Warum erfolgt die Verwaltung der Freispeicherliste über Indirekt-Blöcke?

3. Welche Angaben enthält ein *Inode*? Welche Angaben enthält eine *Verzeichnis-Datei*?

4. Beschreibe kurz die Zugriffsoperationen `open()`, `close()`, `lseek()`, `read()` und `write()` auf ein Unix-Filesystem. Welche Rolle spielt der *Filedeskriptor* dabei?

5. Was geschieht durch einen `mount()`-Systemaufruf in etwa?