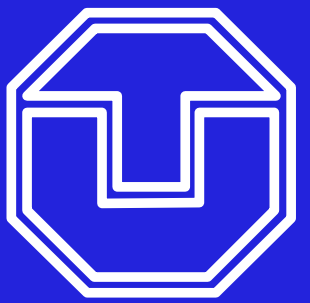
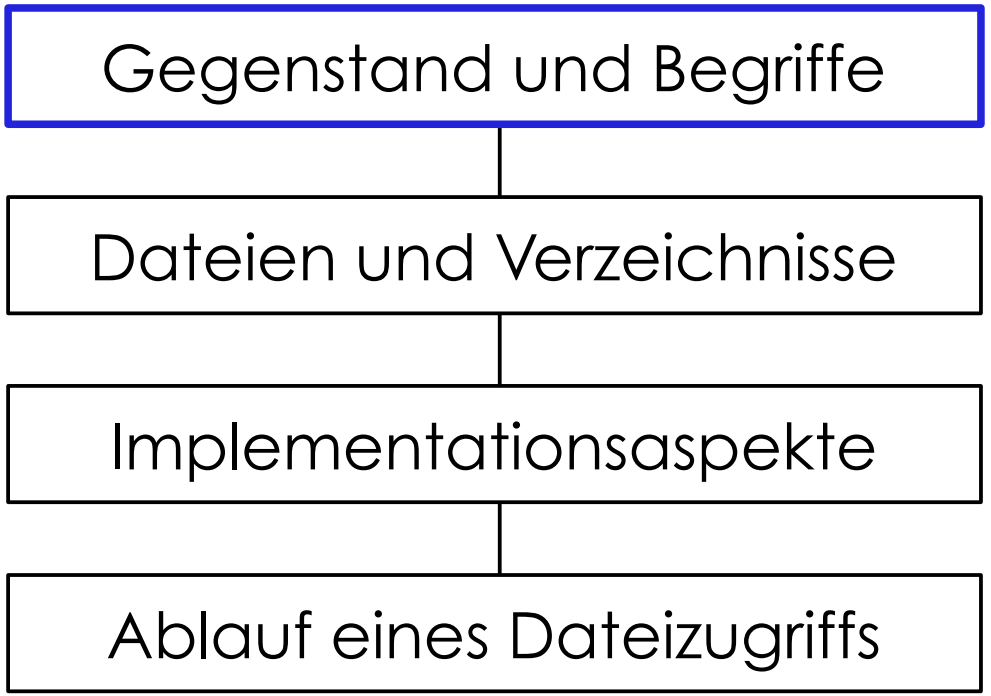
Ein einfaches Dateisystem

Betriebssysteme

Hermann Härtig
TU Dresden



Wegweiser


Gegenstand und Begriffe
Dateien und Verzeichnisse
Implementationsaspekte
Ablauf eines Dateizugriffs

Gegenstand

Aufgaben eines Dateisystems

- **Persistenz** von Daten, d. h. Daten sollen Prozess- und Systembeendigungen überstehen (auch Abstürze)
- Umgang mit großen Datenmengen (z.B. > 4GB = 2^{32} auf IA32)
- Schutz (separate Kapitel dieser Vorlesung)
- Basismechanismen für Datenbanken
- Organisation des Zugriffs paralleler Prozesse auf Dateien

Zentrale Herausforderungen

- Hardwareeigenschaften des persistenten Speichermediums
- Robustheit (→ separate Kapitel in dieser Vorlesung)

Begriffe (nach NEHMER)

Dateisystem

- BS-Komponente, die Anwendungsprogrammen einen effizienten Zugriff auf persistent gespeicherte Daten ermöglicht

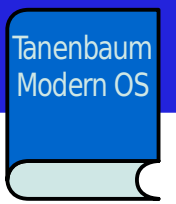
Datei

- „Behälter“ für die dauerhafte Speicherung von Informationen (gleicher oder ähnlicher Struktur) unter einem inhaltlichen Gesichtspunkt
- codiert in Bytes, durch einen „Bezeichner“ repräsentiert

Verzeichnis

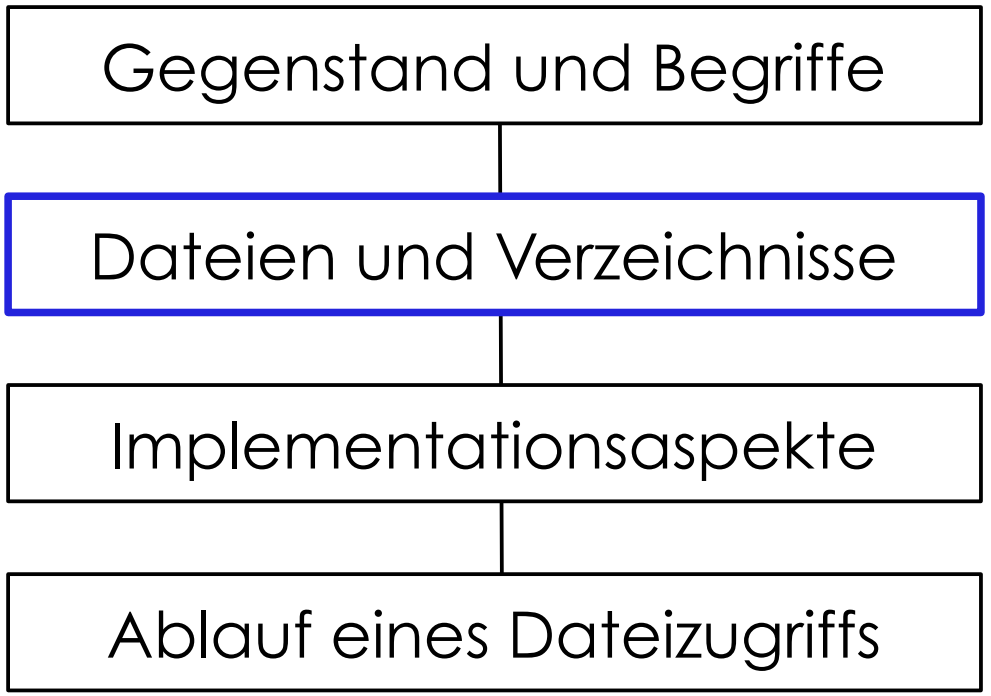
- besondere, vom Dateisystem verwaltete Datei zur Strukturierung der auf Externspeicher abgelegten Dateien

Datei-Operationen



Operation	typische Parameter
• Erzeugen (Create)	Name, Attribute, Zugriffsart → Handle
• Entfernen (Delete, Unlink)	Handle
• Öffnen (Open)	Name, Zugriffsart → Handle
• Schließen (Close)	Handle
• Lesen (Read)	Handle, Anzahl Zeichen, Puffer
• Schreiben (Write)	Handle, Anzahl Zeichen, Puffer
• Append	Handle, Anzahl Zeichen, Puffer
• Seek	Handle, Position
• Lesen/Setzen von Attributen	
• Umbenennen	
• „Einblenden“	

Wegweiser


Gegenstand und Begriffe
Dateien und Verzeichnisse
Implementationsaspekte
Ablauf eines Dateizugriffs

Benennung (von Dateien)

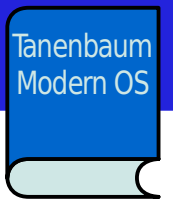
Aufgaben:

- Benennen: symbolisch, (auch) Mensch als Benutzer
- Identifizieren: Programme, Datenbanken als Benutzer
- (schnelles) Lokalisieren
- Zugreifen

Beispiele

- /home/hh7/tmp/Dateien02S.ppt
- 4, 19, 329 (I-Knoten-Nummer); 17 (Filedescriptor)
- canaletto.inf.tu-dresden.de; 141.76.20.10

Symbolische Dateinamen



Benutzer legen Dateinamen fest ...

- innerhalb bestimmter Grenzen, je nach System
- nach Konventionen:
  - Folgen lesbarer Zeichen
  - Groß-/Kleinschreibung signifikant (oder nicht)
  - kontextabhängig (oder nicht)
  - Extensionen, z. B. komprimiertes Quell-Programm: .c.gz
  - Längenbeschränkungen
  - mit Rechnernamen, z. B. erwin:/home/hh7/tmp/x
  - Hierarchiebildung
  - mehrere Namen für ein Objekt
  - inhaltsbezogene Benennung
  - Möglichkeiten zur Abkürzung

Datei-Attribute

Zugriffschutz

- Eigentümer, Besitzer, Rechte des Besitzers und anderer

Zeiten

- Erzeugung, letzte Modifikation, letzter Zugriff, ...

Organisatorisches

- aktuelle/maximale Größe
- Verwaltungsinformationen
- Strukturinformationen

Datei-Typen

- in Unix-Welt:
- ➔ alle (cum grano salis) Objekte werden auch als Dateien behandelt

Beispiele

- normale Dateien (regular file)
- Verzeichnisse (directories)
- E/A-Geräte (special files)
- Datenströme (pipes)
- Symbolic Links

Zugriffsstrukturen

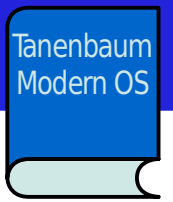
flach (Unix)

- ARRAY of BYTE
- Lesen/Schreiben in beliebigen Einheiten an beliebigen Stellen
- Aufprägung von Strukturen durch Anwendungen

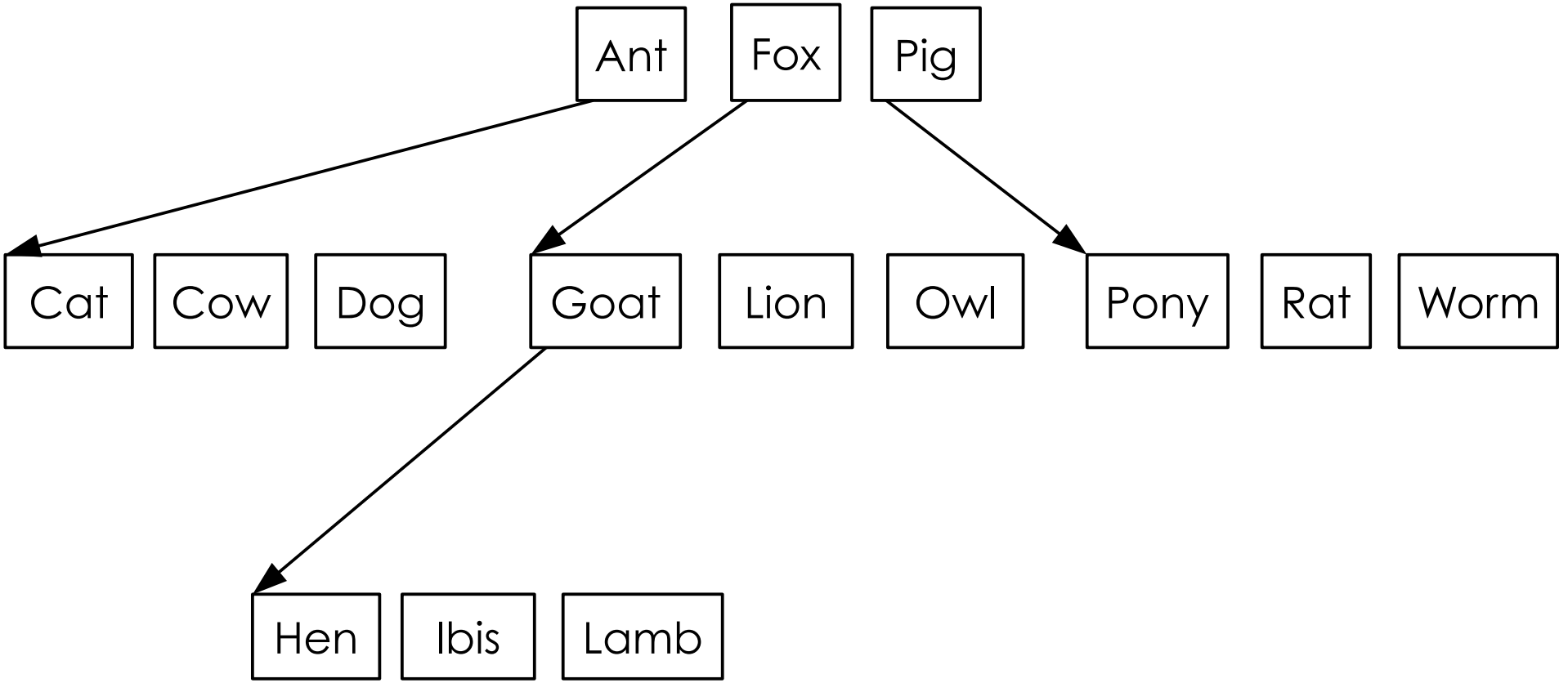
„Record“-Strukturen (satzstrukturierte Dateien)

- Länge konstant oder variabel
- ein Datensatz (record) pro Zugriff (Lesen/Schreiben)
  - sequentiell
  - direkt (B- bzw. B*-Bäume), Zugriff über Schlüssel
  - indexsequentiell

Indexsequentielle Dateien



- Datensätze beliebiger Länge mit Schlüssel
- Zugriffsreihenfolge: sequentiell oder über „Schlüssel“



Verzeichnisse und Pfadnamen

Problem:

- systemweite Eindeutigkeit von Dateinamen

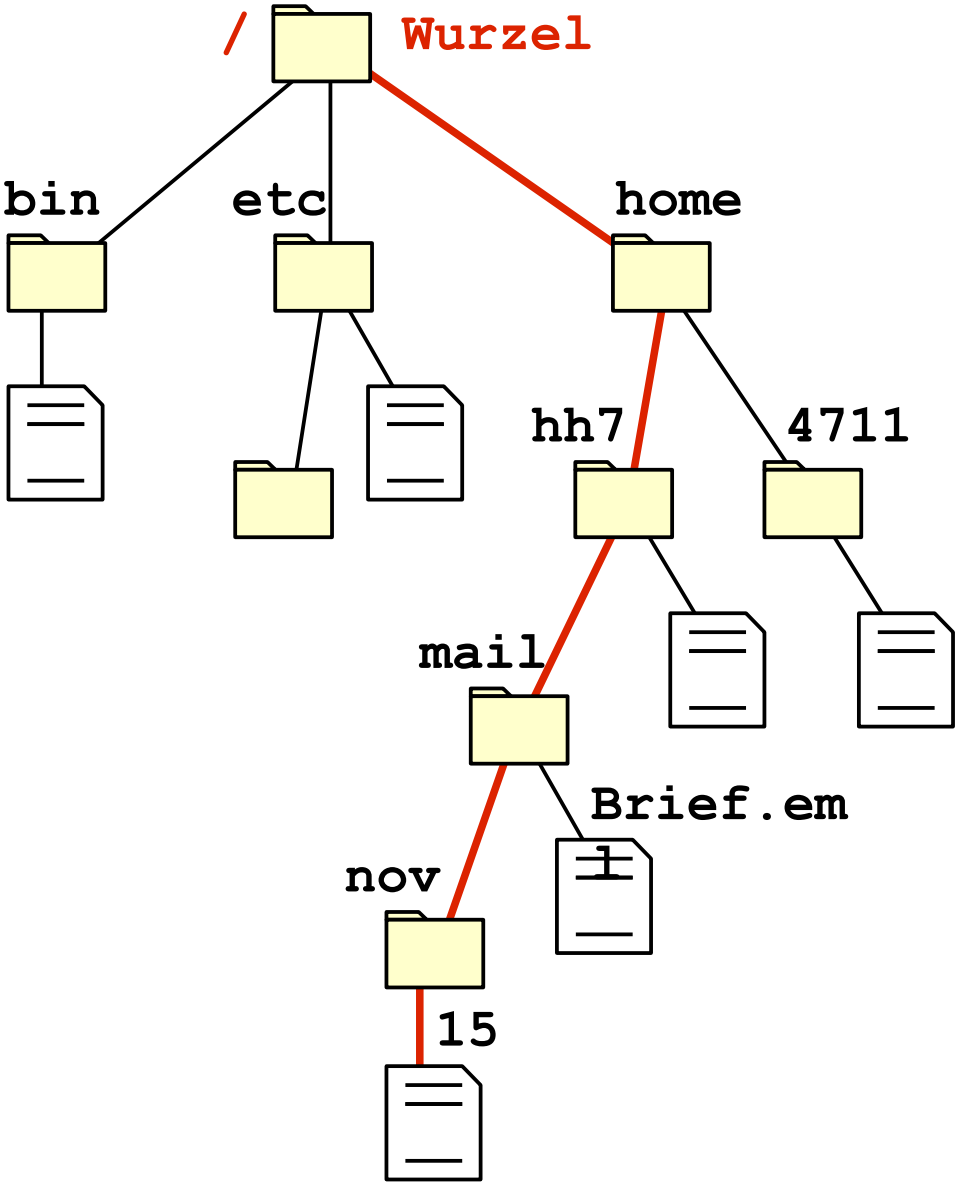
Lösung:

- Name eindeutig in einem „Kontext“
- auch Kontexte haben Namen

Pfadname:

- Folge von Teilnamen
- Namens-Auflösung:
  von einem Ausgangs-Kontext ausgehend wird jeder Teilname in seinem Kontext interpretiert

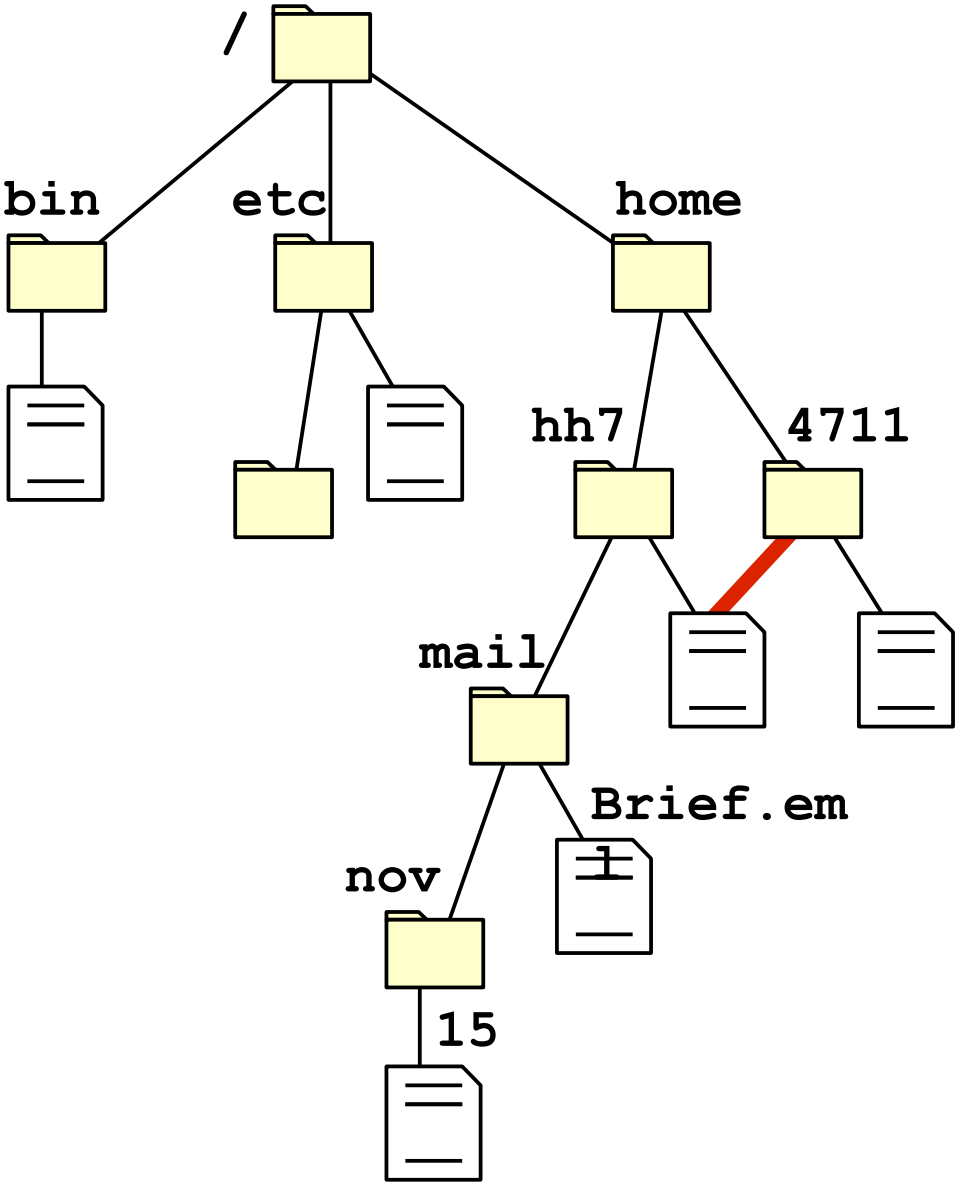
Beispiel: /home/hh7/mail/nov/15



Mehrere Namen für eine Datei: Harte Links

„Harte Links“

- Identifikatoren in mehreren Verzeichnissen
- Limitation:
  Gültigkeit Identifikatoren über Rechnergrenzen hinweg ?



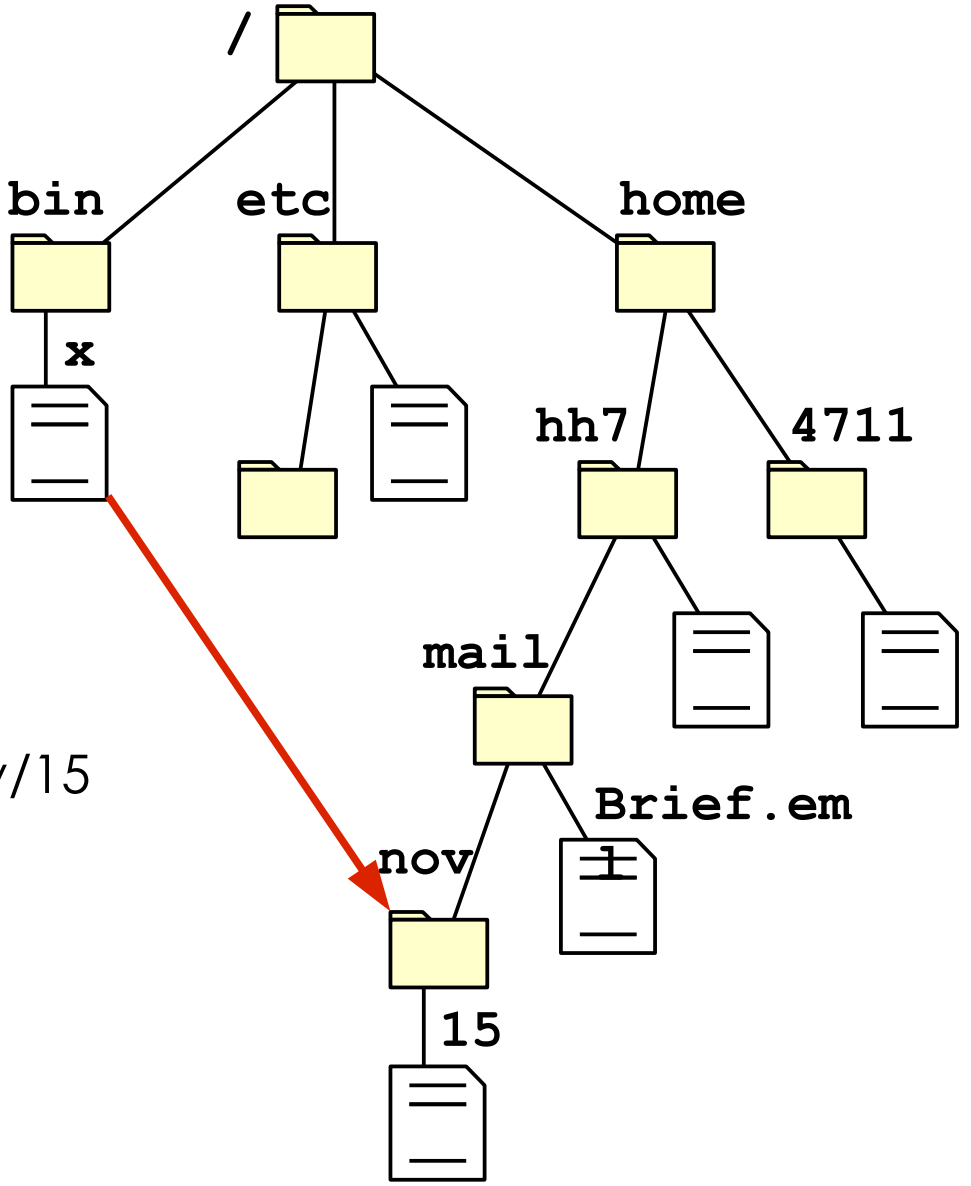
Mehrere Namen für ein Objekt: Weiche Links

„Symbolische Links“

- Namen als Objekte
- Zyklen !

Beispiel

- x ⟶ /home/hh7/mail/nov
- → /bin/x/15 = /home/hh7/mail/nov/15



Beispiel für Zyklen

/etc/bin/XYUtils → /bin/XYUtils

/bin/XYUtils → /usr/local/bin/XYUtils

/usr/local/bin/XYUtils → /etc/bin/XYUtils

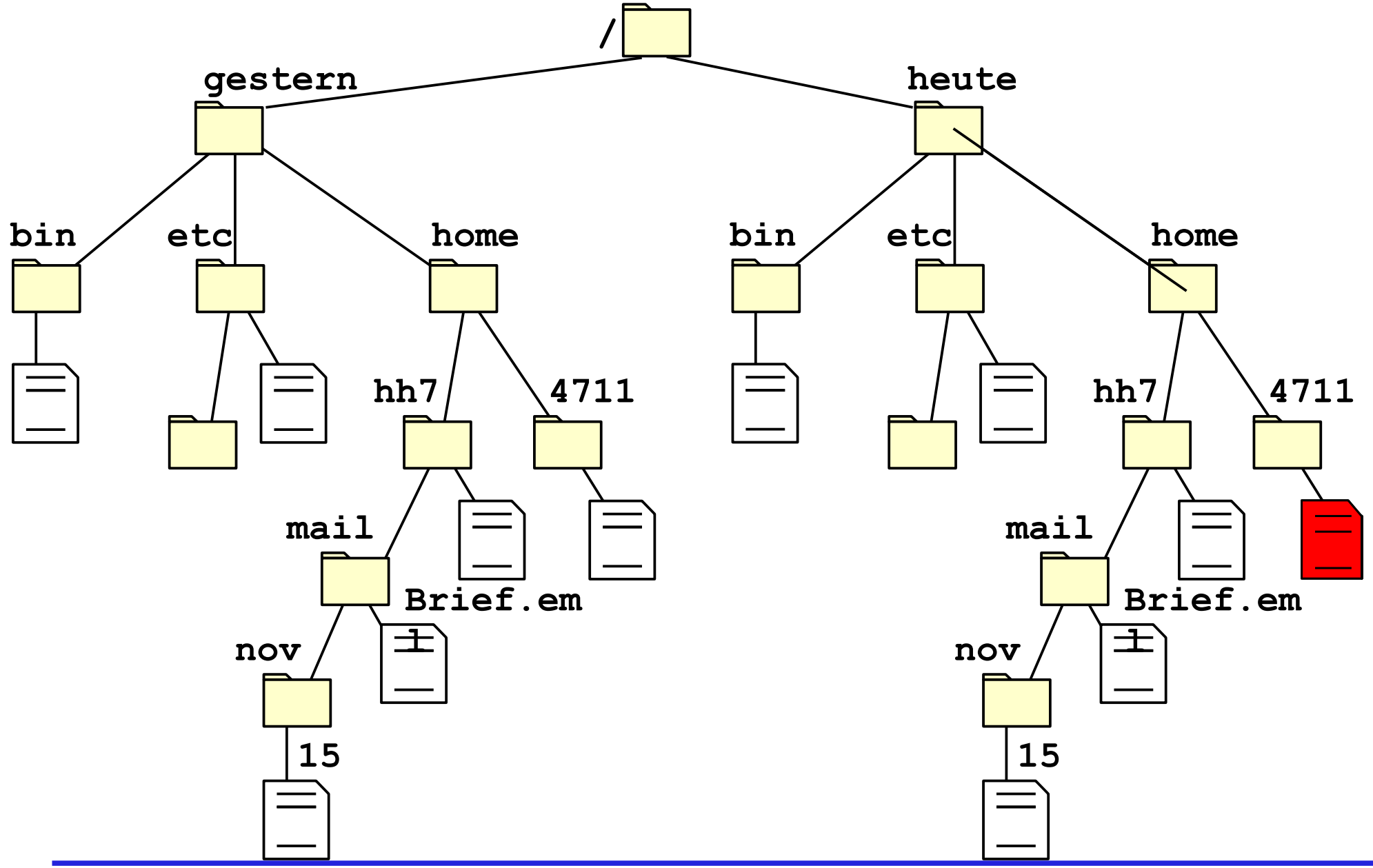
Auflösung von /bin/XYUtils/example

- /usr/local/bin/XYUtils/example
- /etc/bin/XYUtils/example
- /bin/XYUtils/example
- /usr/local/bin/XYUtils/example
- /etc/bin/XYUtils/example

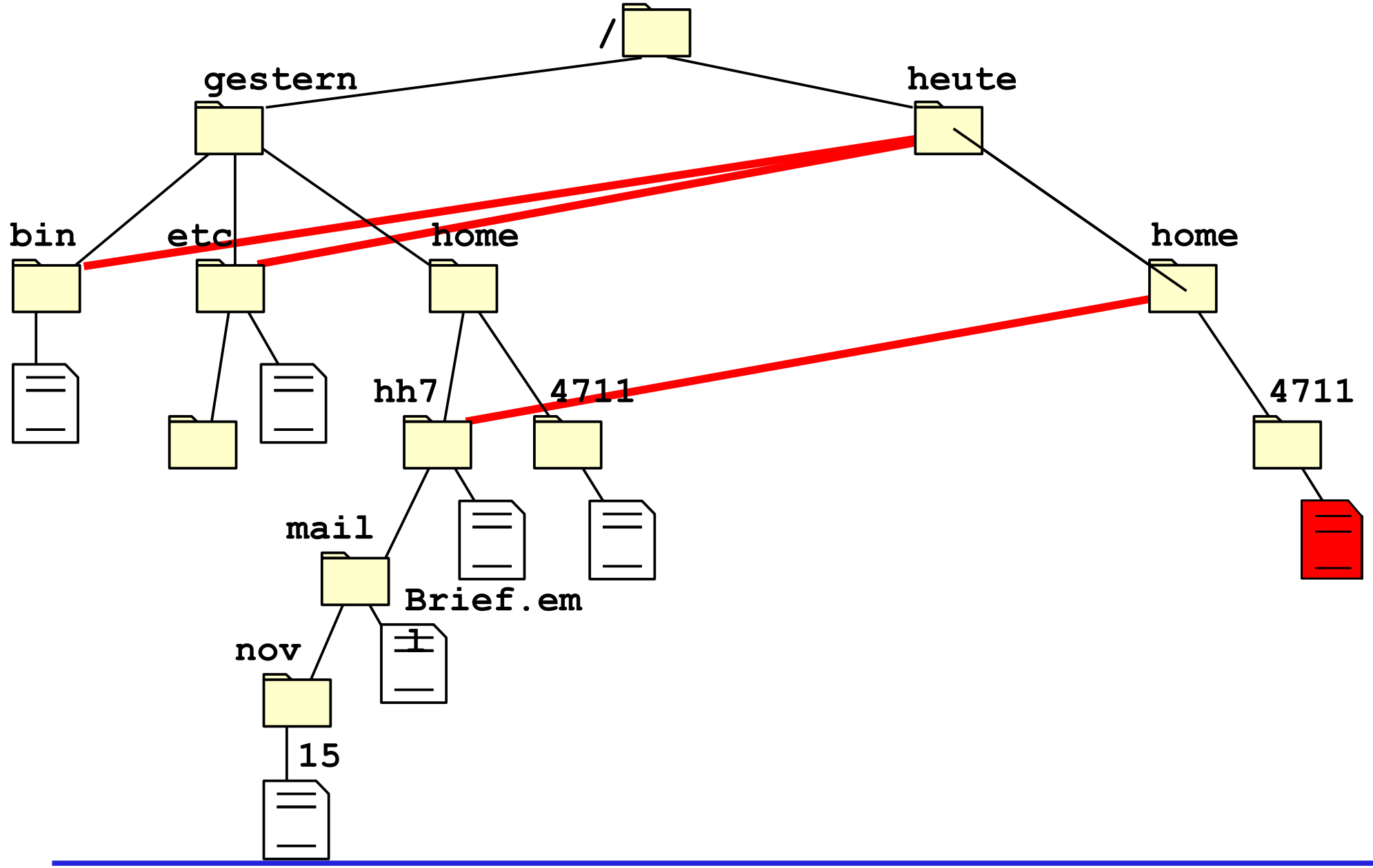
Beispiel: Backup

- Alle X Stunden vollständige Kopie des Dateisystems
- Nur das Nötige Kopieren
- Mehrfache harte Links auf alles, was sich nicht änderte

Beispiel: Backup



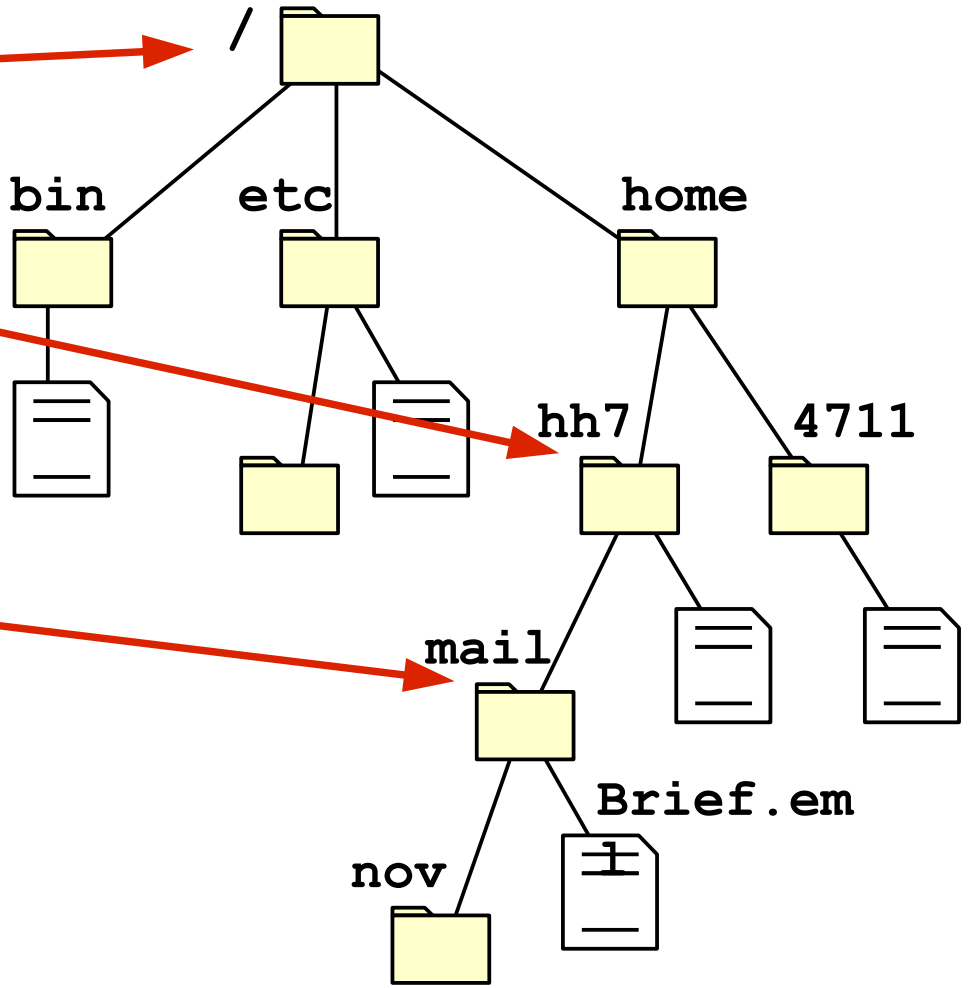
Beispiel: Backup



Kontexte für Namen

Ausgangs-Kontexte durch Prozesse festgelegt, z. B. in Unix :

- Root directory:
  Kontext für globale Namen
- home directory:
  current directory bei login und durch parameterloses „cd“
- current directory:
  Ausgangskontext für relative Pfadnamen



Prozesse und Pfadnamen

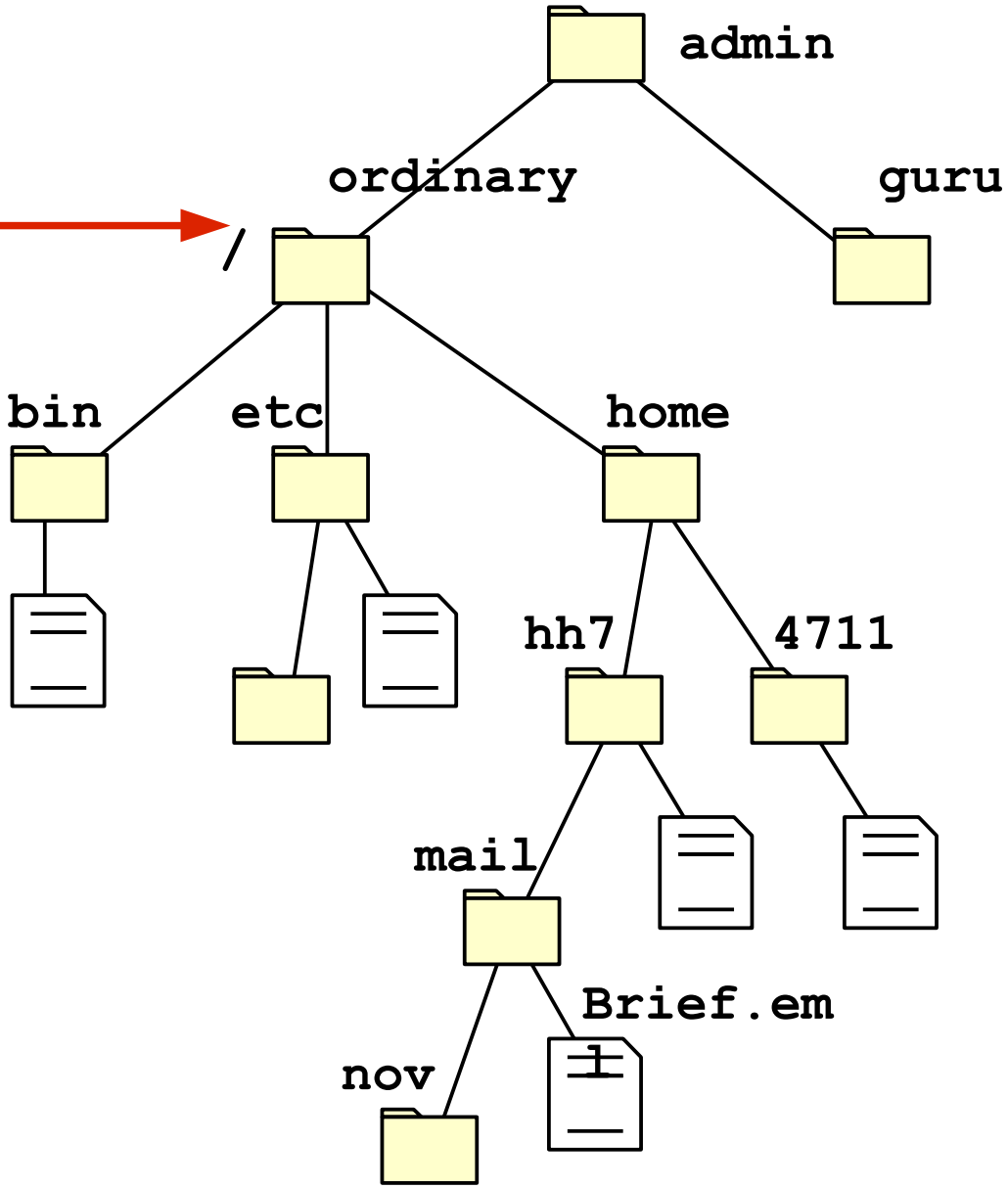
Ausgangs-Kontexte durch Prozesse festgelegt, z. B. in Unix :

- current root:
  Ausgangskontext für absolute Pfadnamen

  „chroot“ Kommando

  mehrere „Globale“ Namensräume

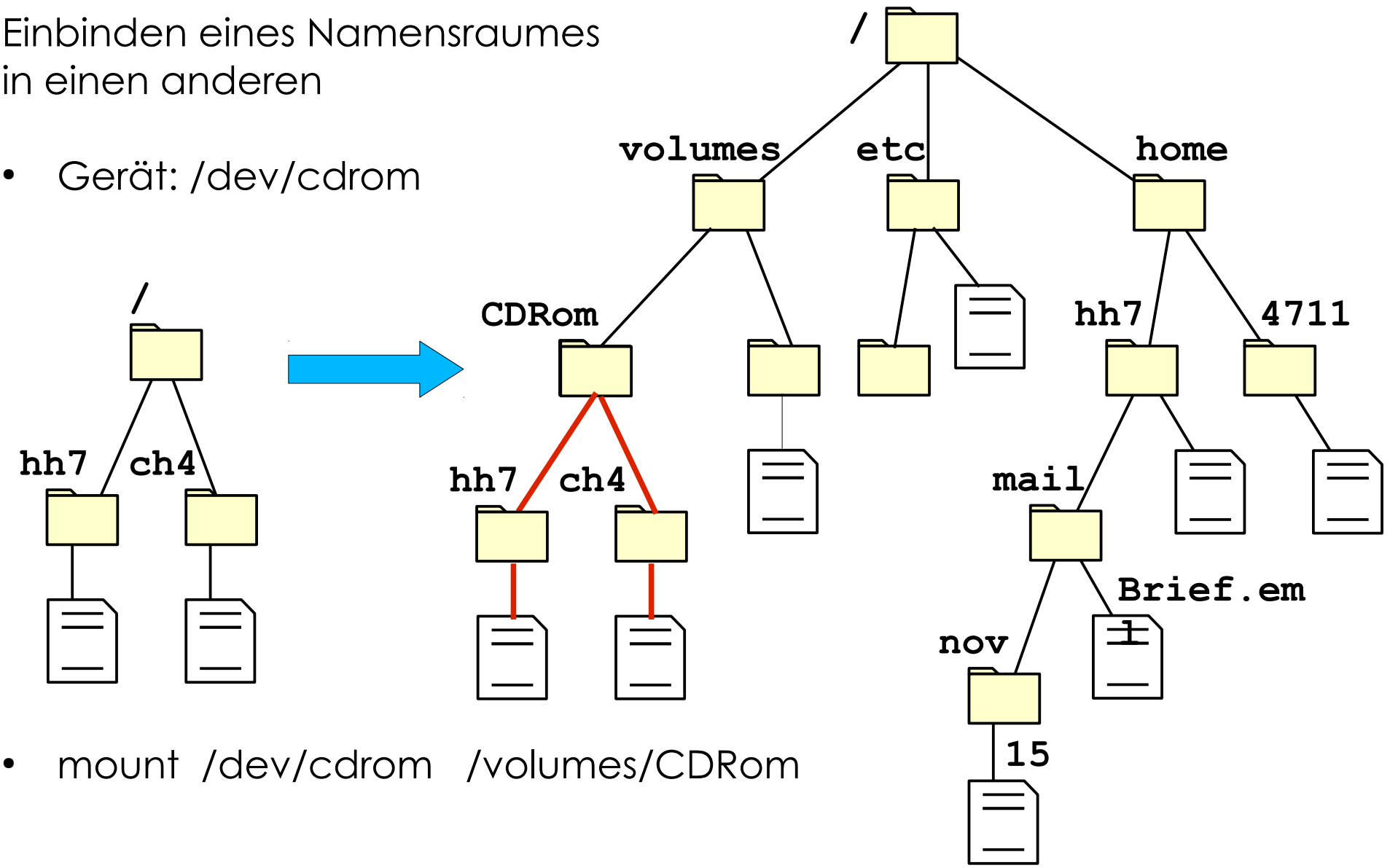
  OS Virtualisierung



Verbinden von Namensräumen: Mounting

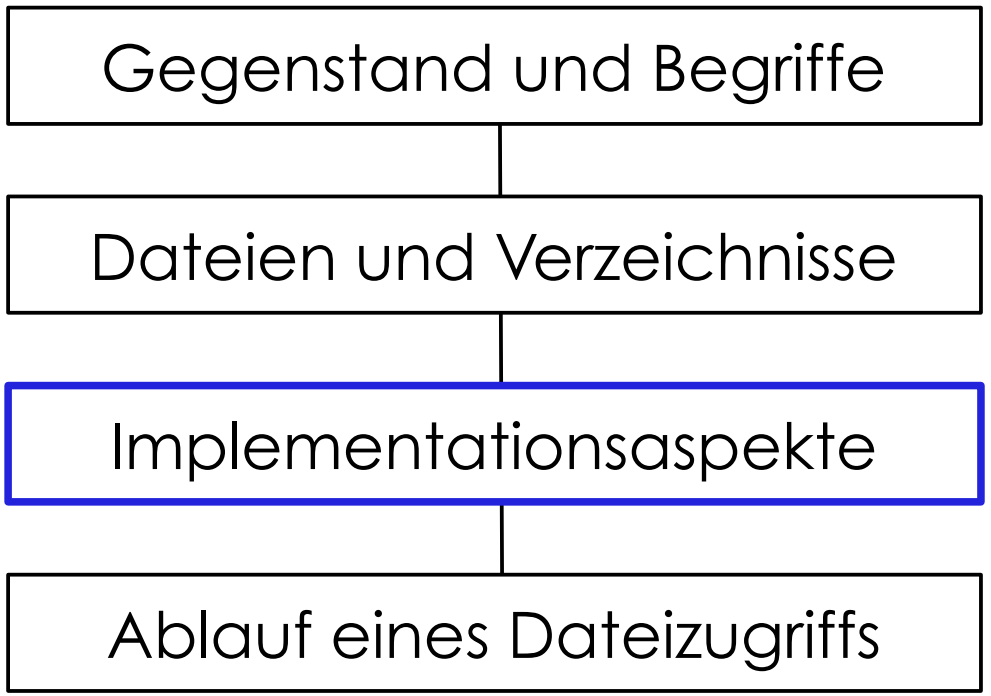
Einbinden eines Namensraumes
in einen anderen

- Gerät: /dev/cdrom



- mount /dev/cdrom /volumes/CDRom

Wegweiser

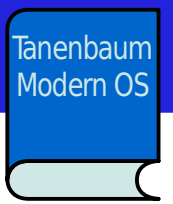

Gegenstand und Begriffe
Dateien und Verzeichnisse
Implementationsaspekte
Ablauf eines Dateizugriffs

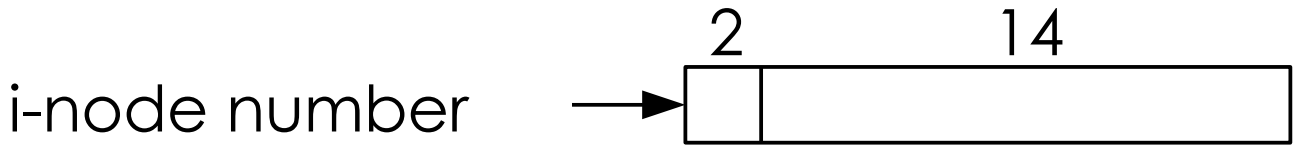
Verzeichnisse

Implementierung und Zugriff analog normale Datei mit:

- interner Struktur
- speziellen Operationen
  - opendir / closedir
  - readdir
  - lookup
  - rename

Eintrag im Unix-Verzeichnis





Zugriff auf /home/hh7/mbox

Root-Directory

1	.
1	..
4	bin
7	dev
14	lib
9	etc
6	home
8	tmp

looking up home yields i-node 6

i-node 6 is for /home

mode
size
times
132

i-node 6 says that /home is in block 132

block 132 is /home directory

6	.
1	..
19	sdsd
30	erik
51	jim
26	hh7
45	hal
8	tmp

/home/hh7 is i-node 26

i-node 26 is for /home/hh7

mode
size
times
406

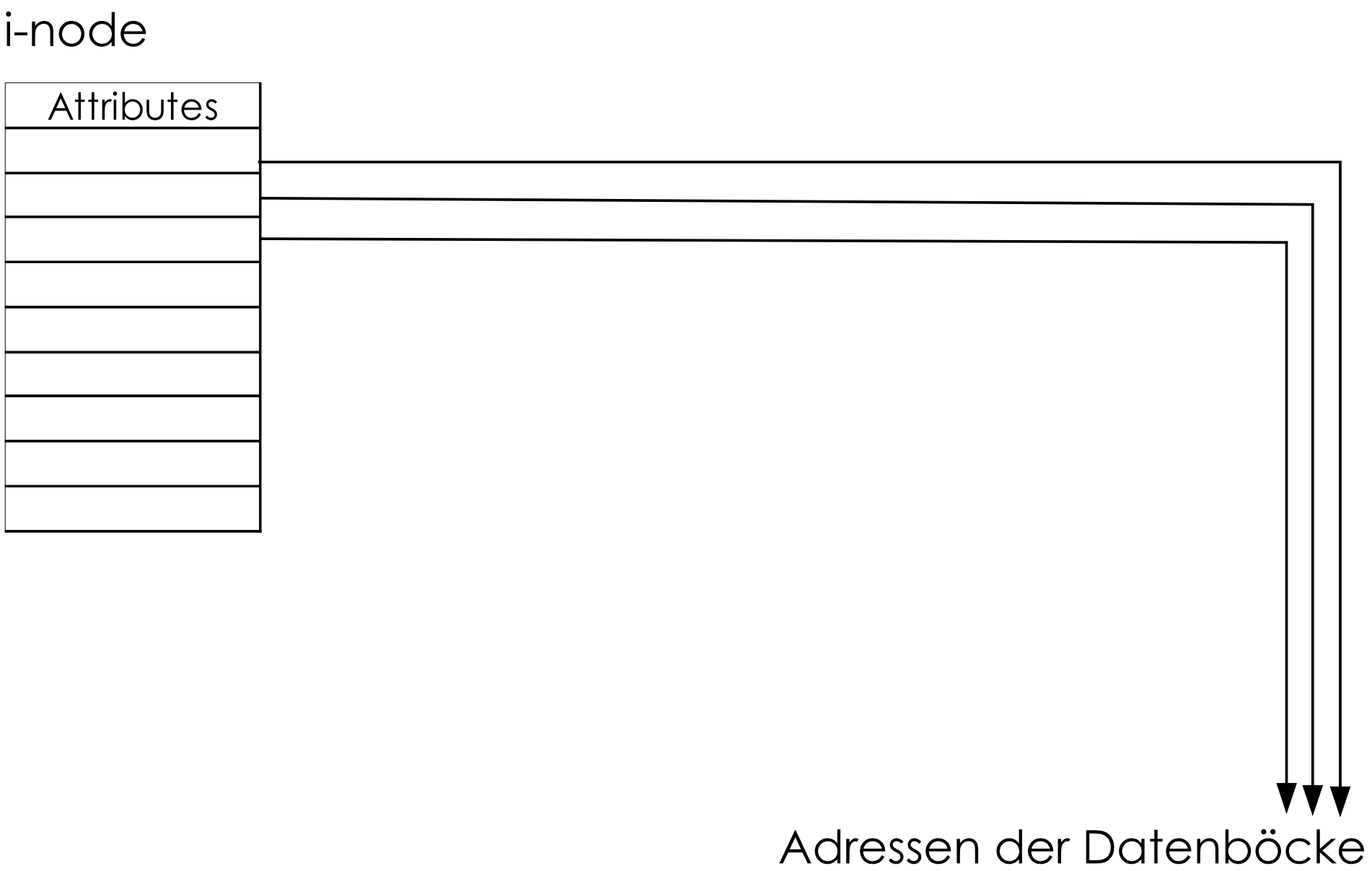
i-node 26 says that /home/hh7 is in block 406

block 406 is /hh7 directory

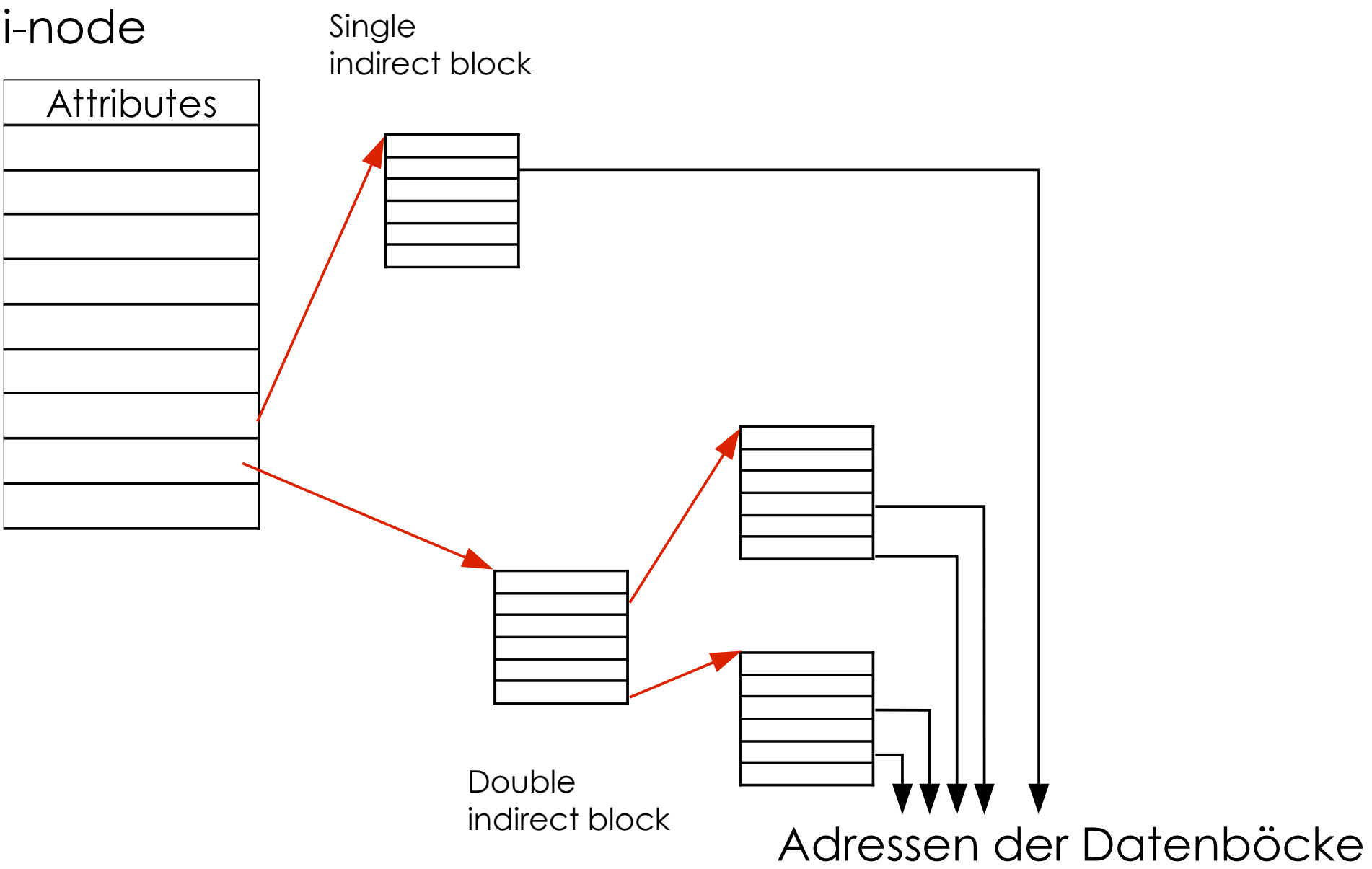
26	.
6	..
64	dick
92	books
60	mbox
81	minix
17	src

/home/hh7/mbox is i-node 60

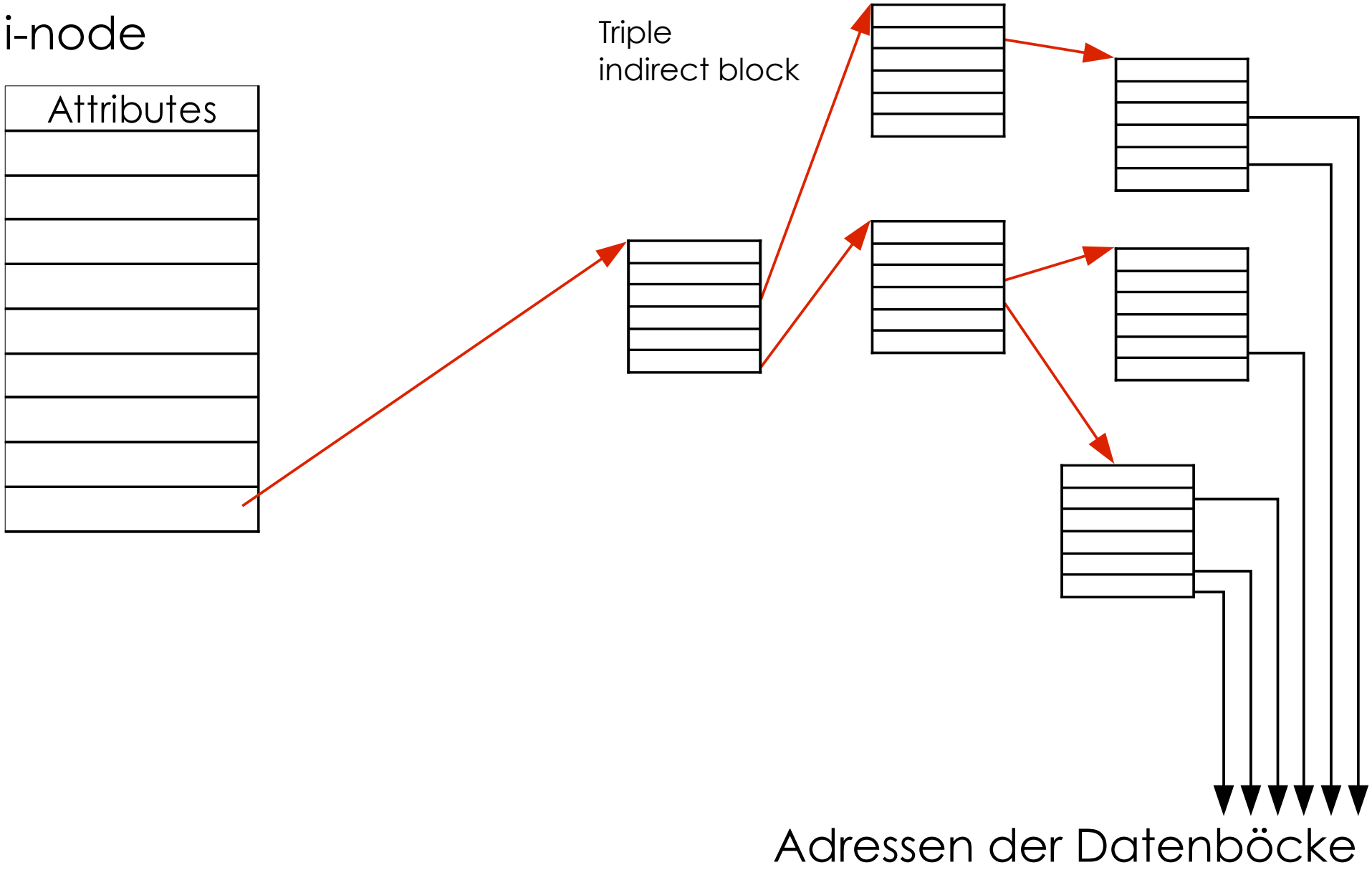
i-nodes (I-Knoten)



i-nodes (I-Knoten)

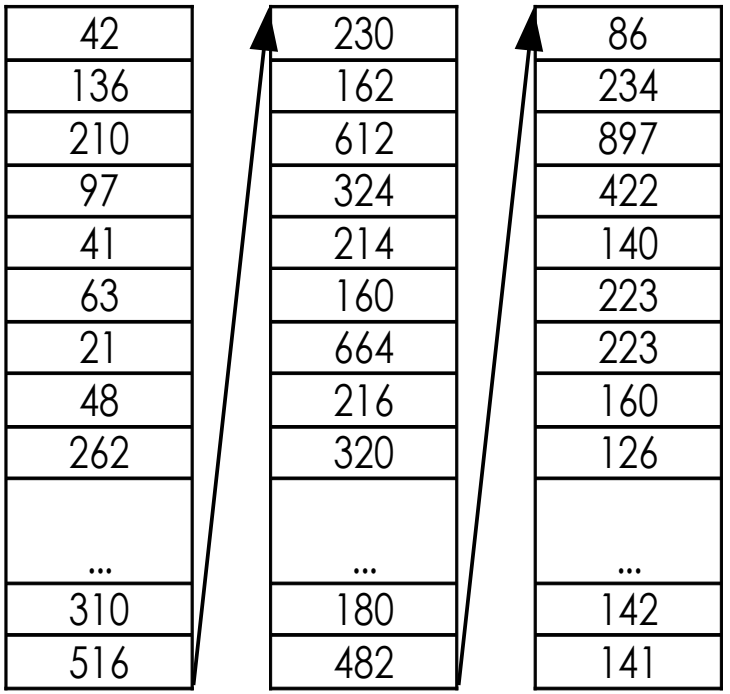


i-nodes (I-Knoten)

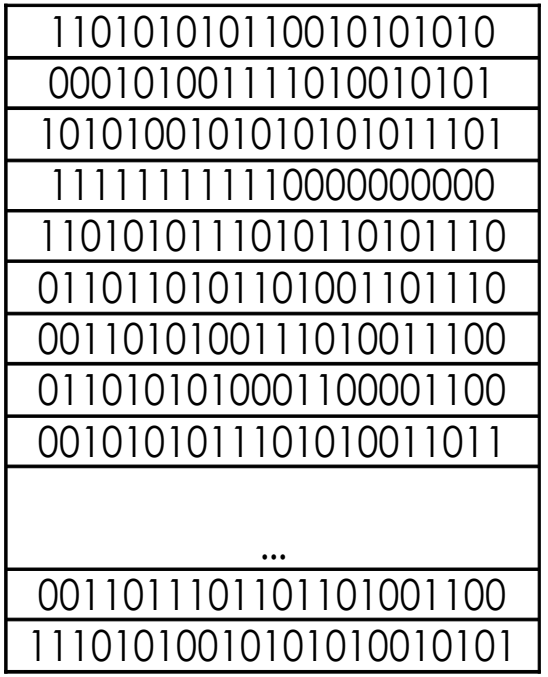


Verwaltung des freien Plattenspeichers

Verwendung von Listen



Verwendung von Bitmaps



Wegweiser


Gegenstand und Begriffe
Dateien und Verzeichnisse
Implementationsaspekte
Ablauf eines Dateizugriffs
Plattenspeicher
Persistenz bei Auftreten von Fehlern

Schnittstelle

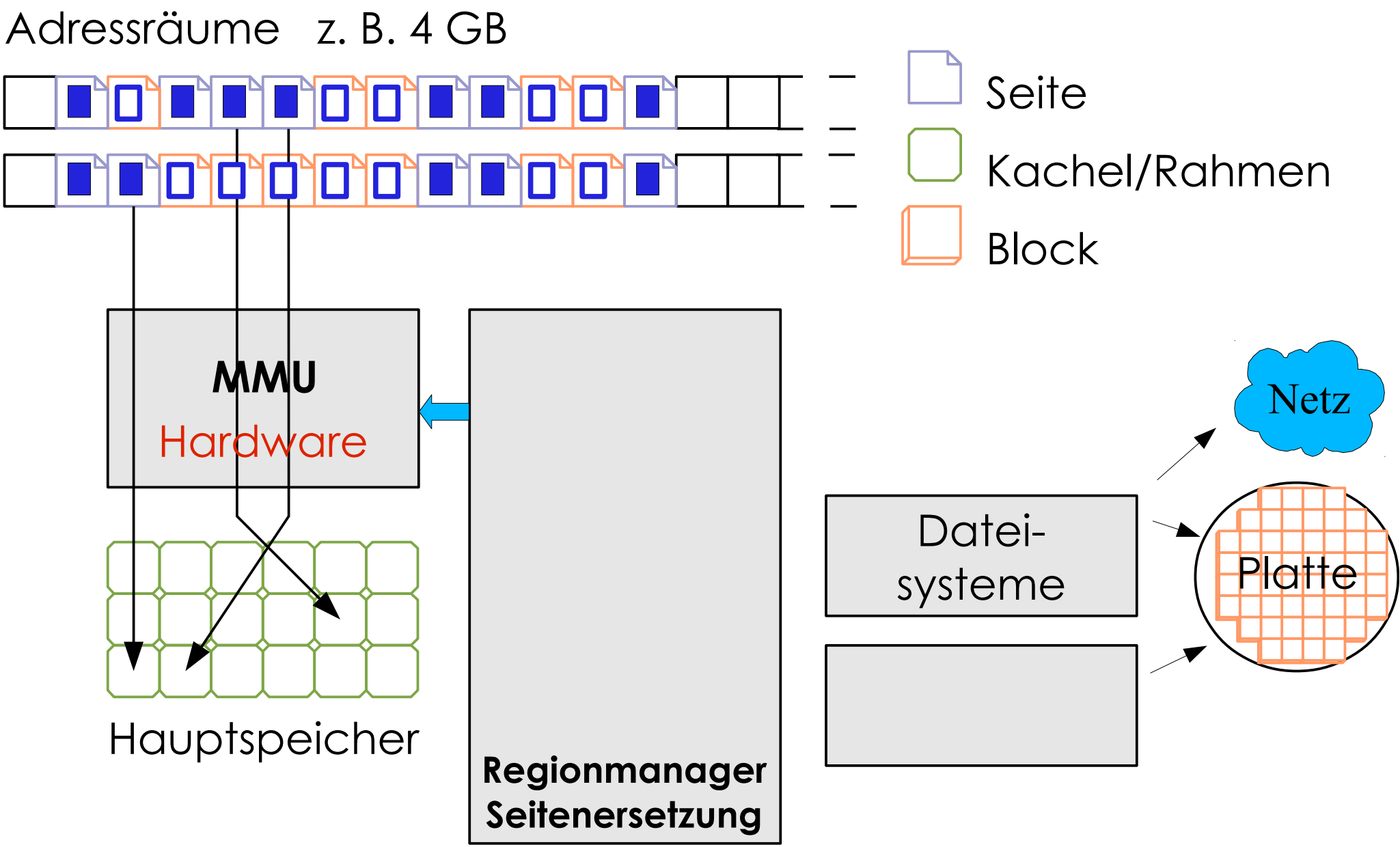
Namensdienste:
symbolische Namen → Identifikatoren

Zugriffe auf (Daten in) Dateien:

- häufig: **`open`** (Zugriff)* **`close`**
- Zugriff über
  - Kopieren aus Datei in Adressraum und zurück:
    `read/write` (Datei, Position in Datei, Länge, Adresse in Adressraum)
    keine Ausrichtung der Adressen notwendig
    `seek`-Operation ersetzt „Position in Datei“ als Parameter
  - Einblenden in Adressraum:
    `map` (Datei, Adresse in Adressraum, Offset, Länge)
    dann Zugriff über normale Maschinen-Instruktionen
    Offset und Adresse müssen an Seitengrenzen ausgerichtet sein

Zugriffe auf Dateiattribute ...

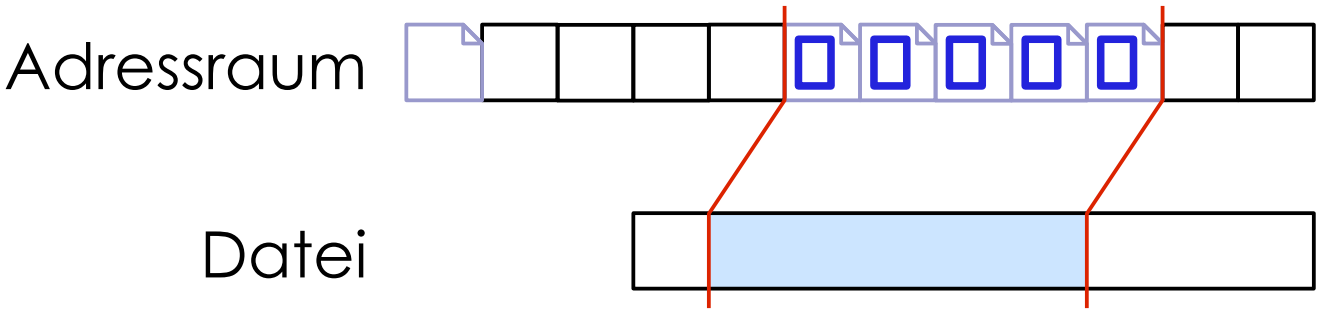
Integration in Systemarchitektur



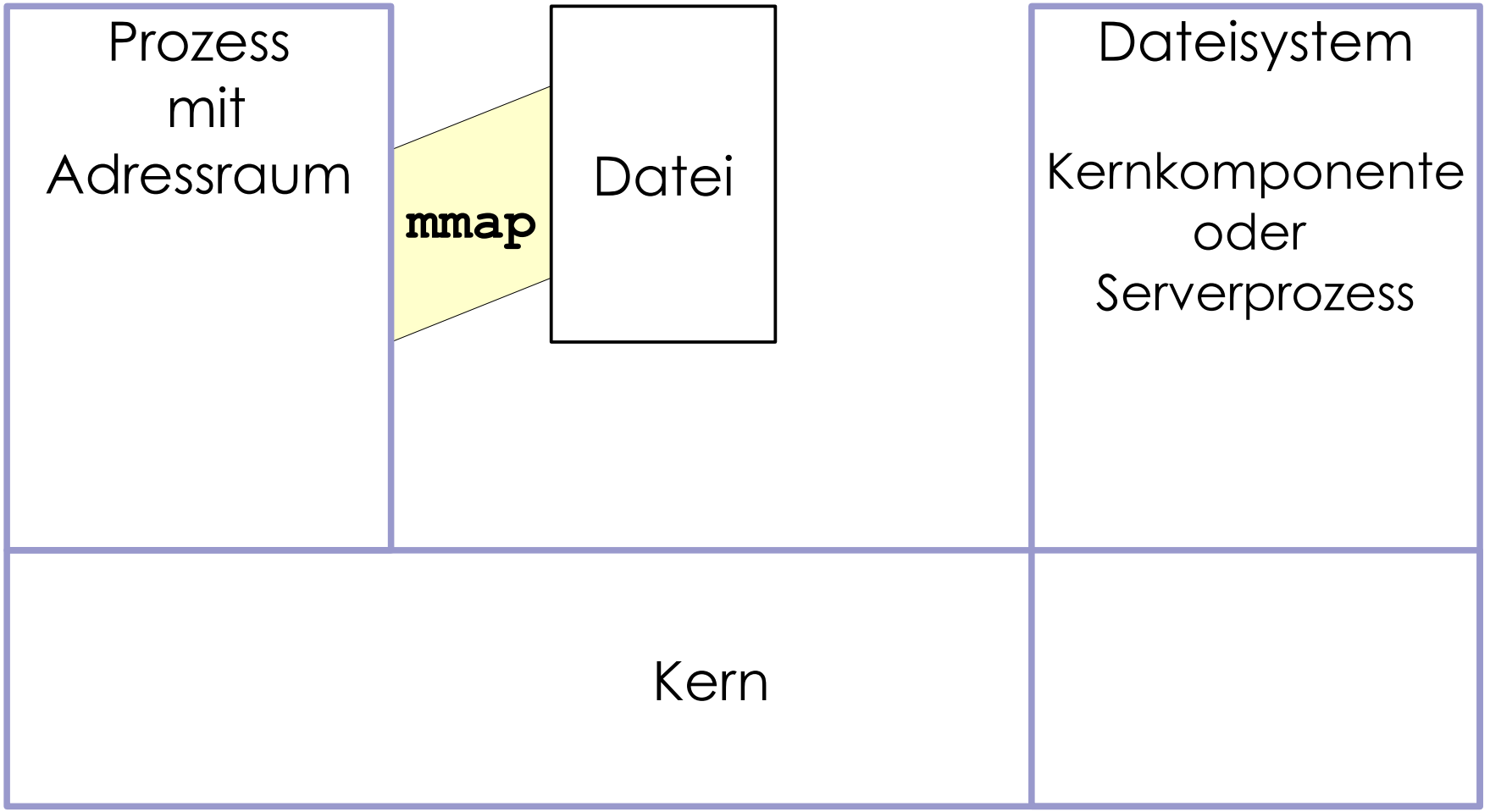
`mmap (Datei, Adresse, Länge, Offset)`

- bei Seitenfehler:
  - Zugriff auf Datei über Seitenfehlerbehandlung
  - dann Zugriff über normale HW-Instruktionen

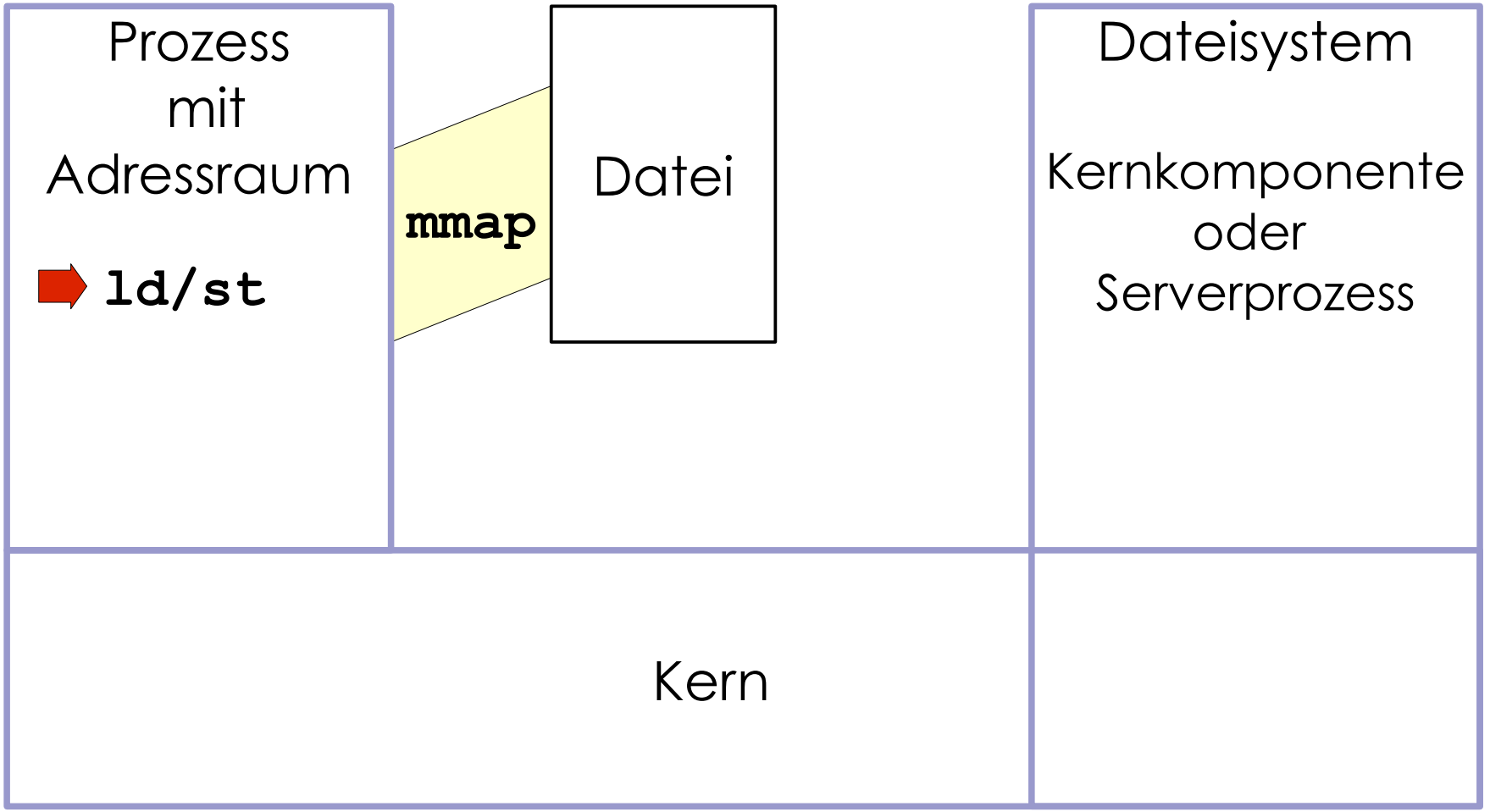
➔ „Pager“ muss Struktur der Dateiimplementierung kennen!



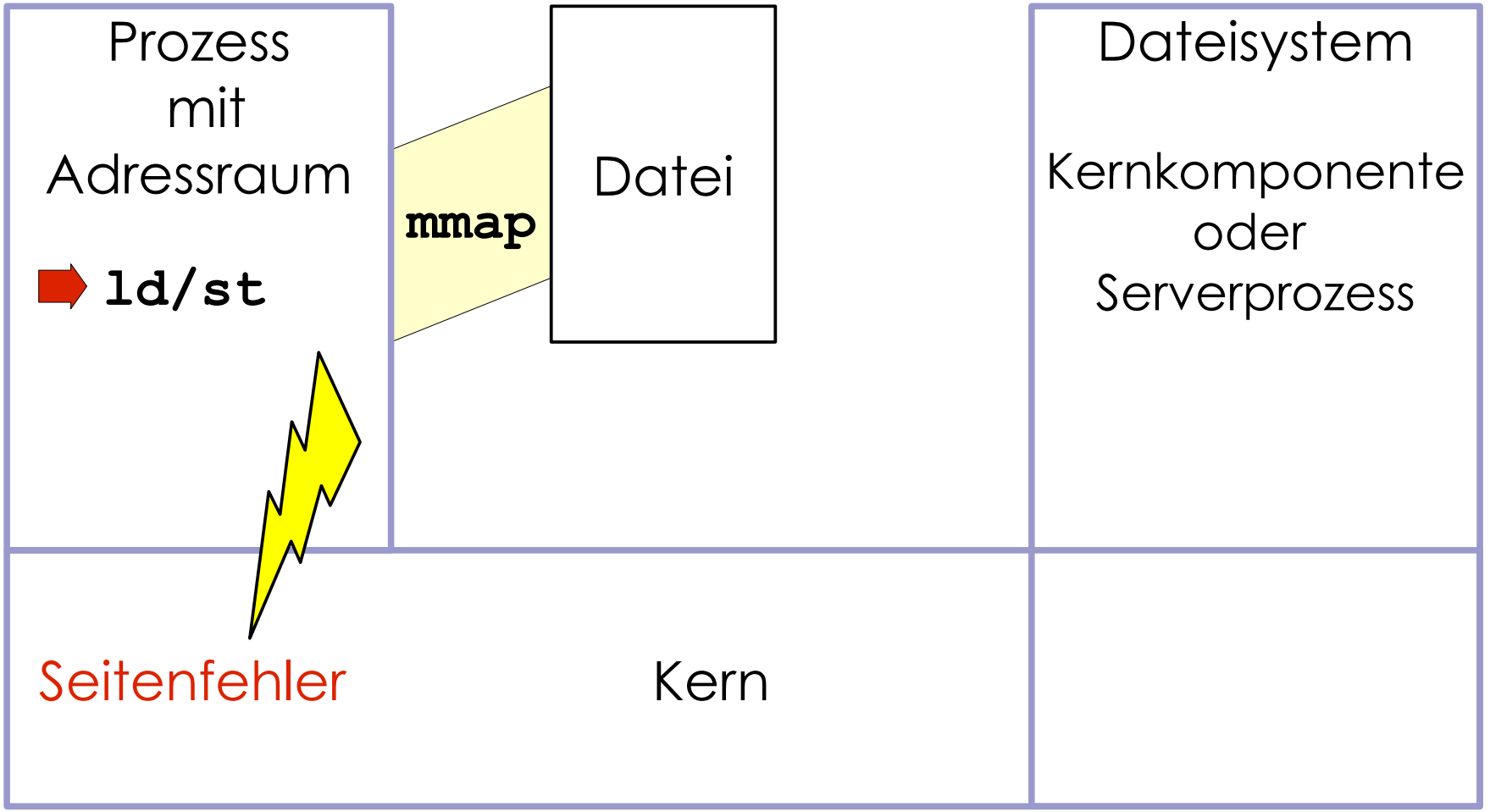
Zugriff mittels Einblendung



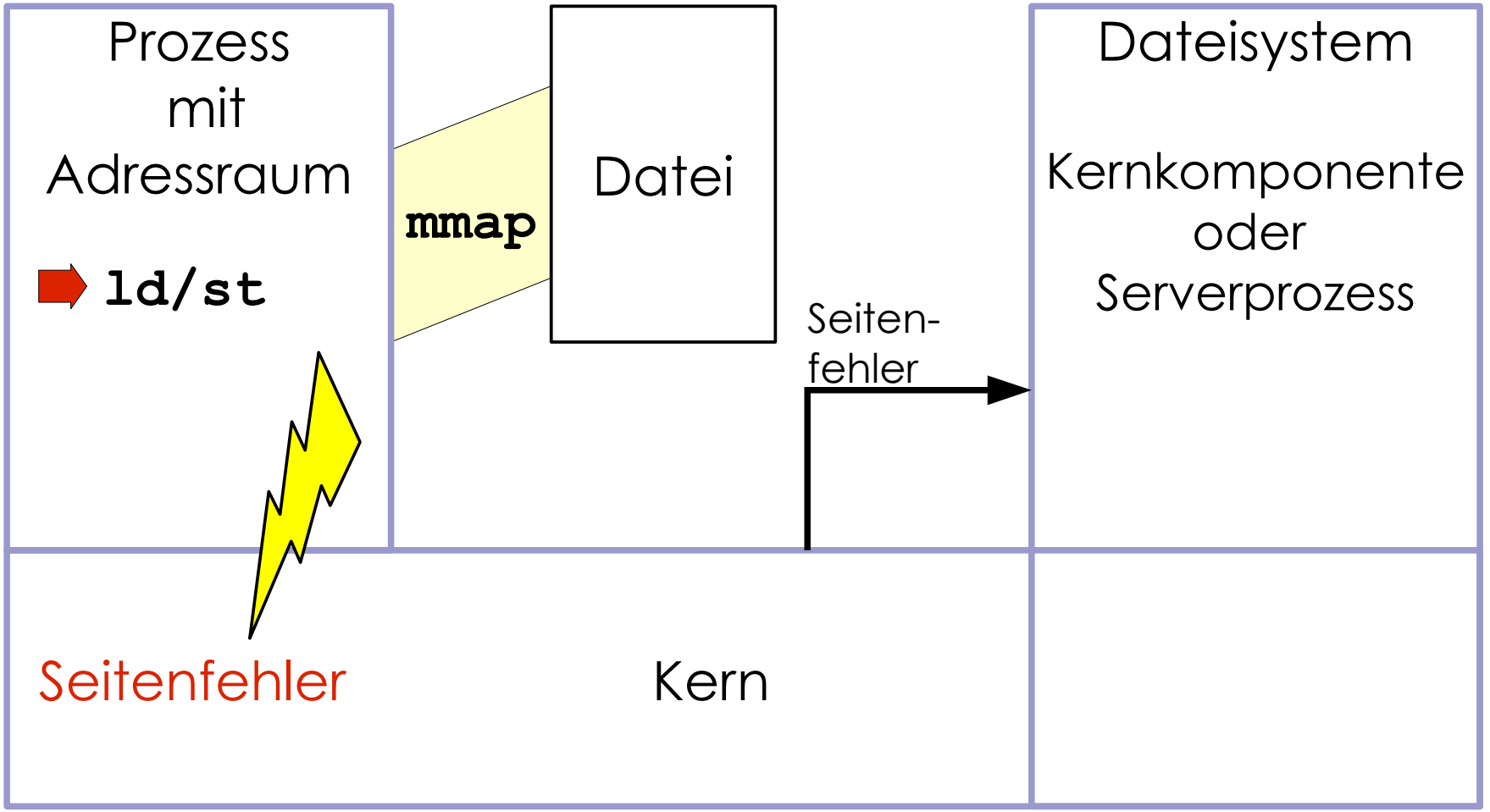
Zugriff mittels Einblendung



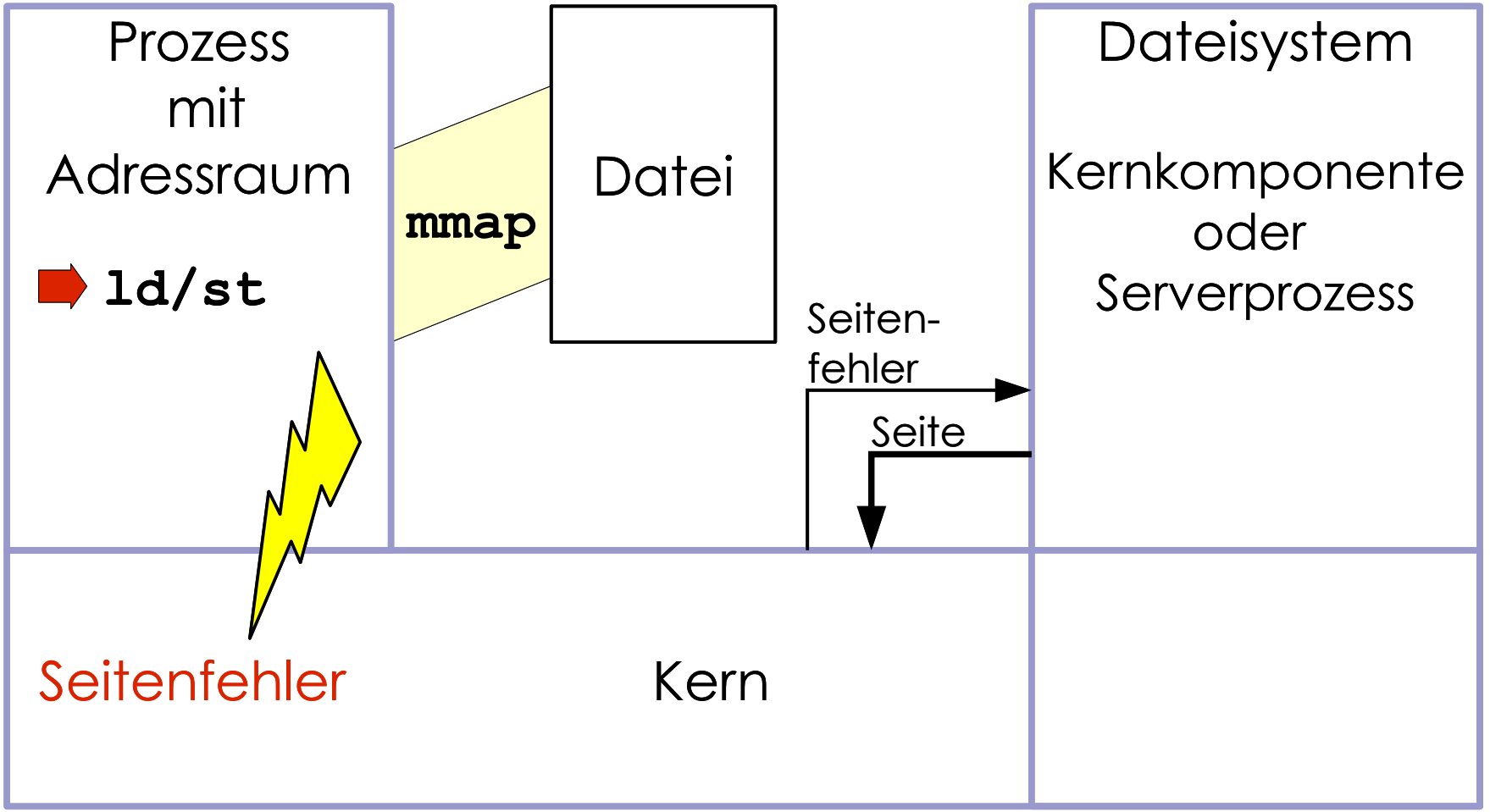
Zugriff mittels Einblendung



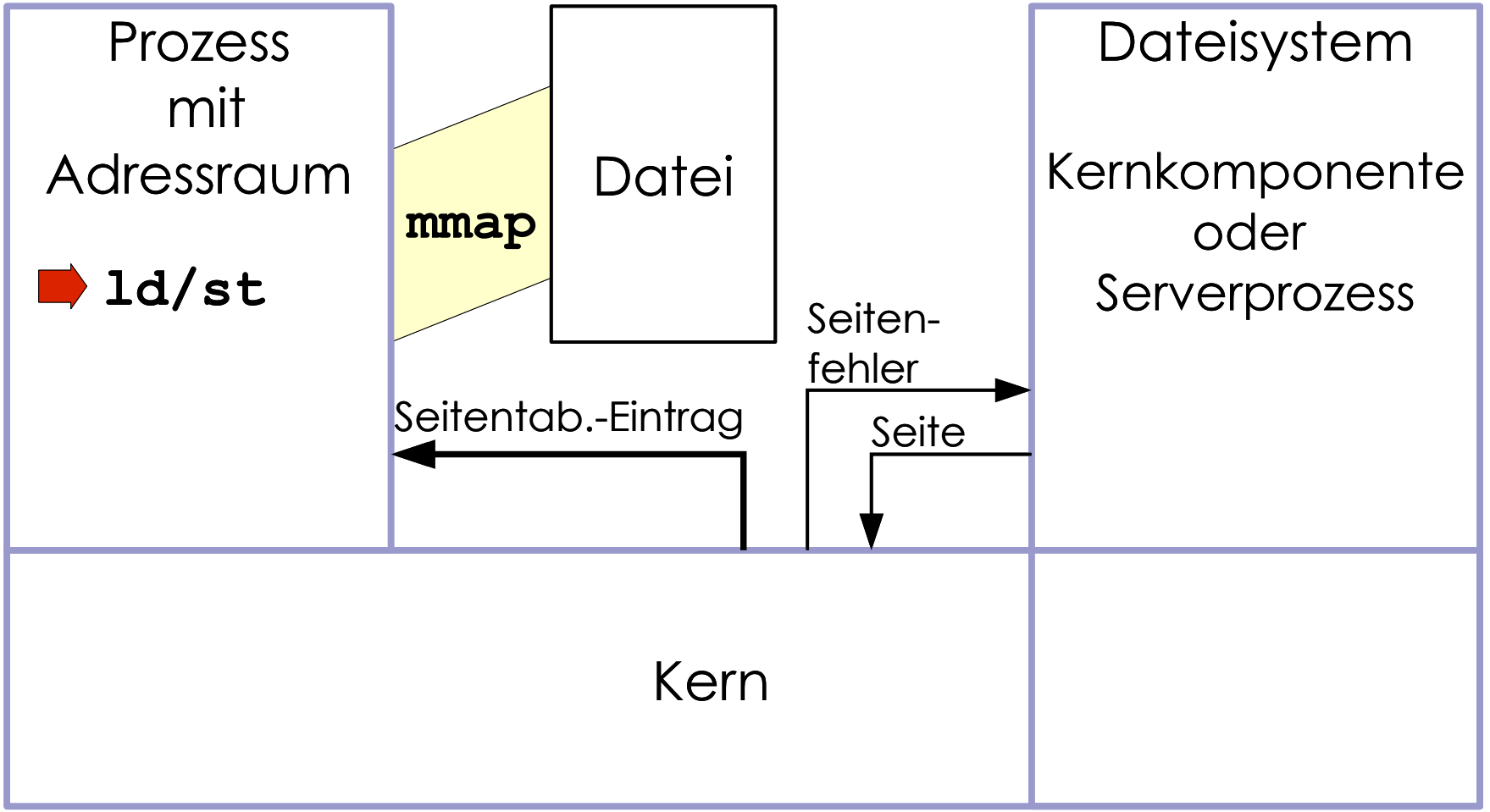
Zugriff mittels Einblendung



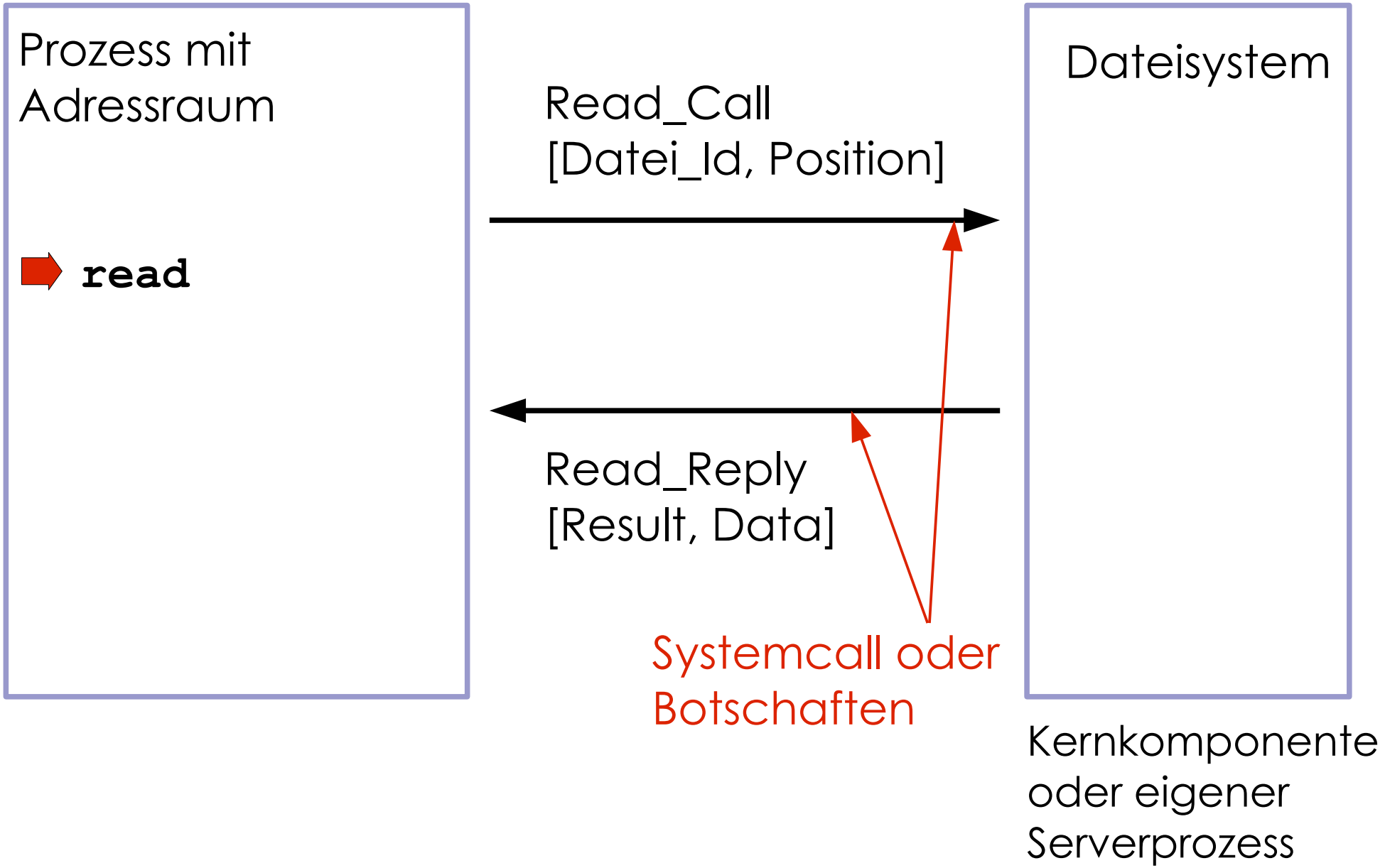
Zugriff mittels Einblendung



Zugriff mittels Einblendung



Zugriff mittels Kopieren (read/write)



Ablauf eines „write“ Dateizugriffs (Unix)

`fd = open(„drops/papers/xy.tex“, RW, ...)`

Pfadname → i-node number (Namensauflösung)

i-node number → fd

`seek (fd, position)`

`write (fd,address, length)`

Im folgenden

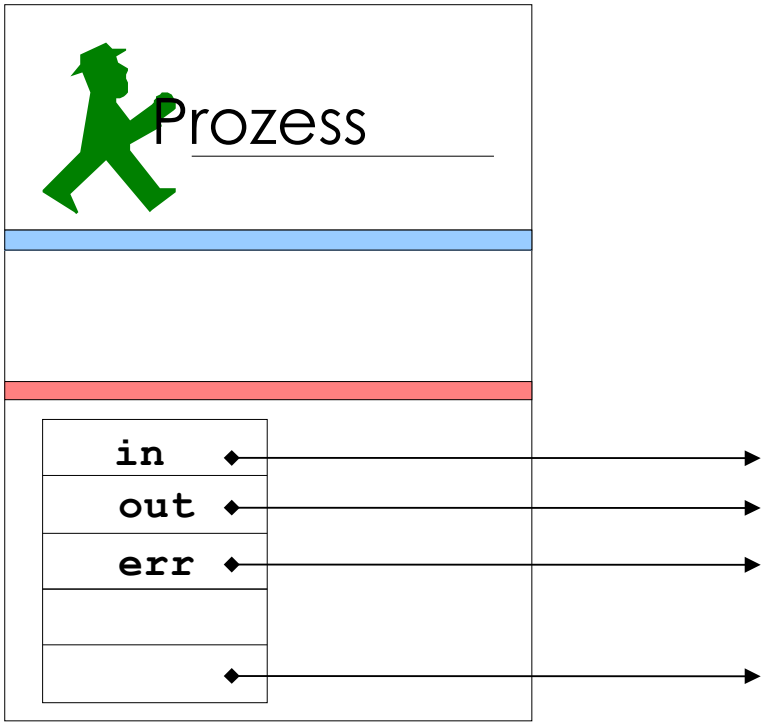
- beteiligte Datenstrukturen
- Ablauf

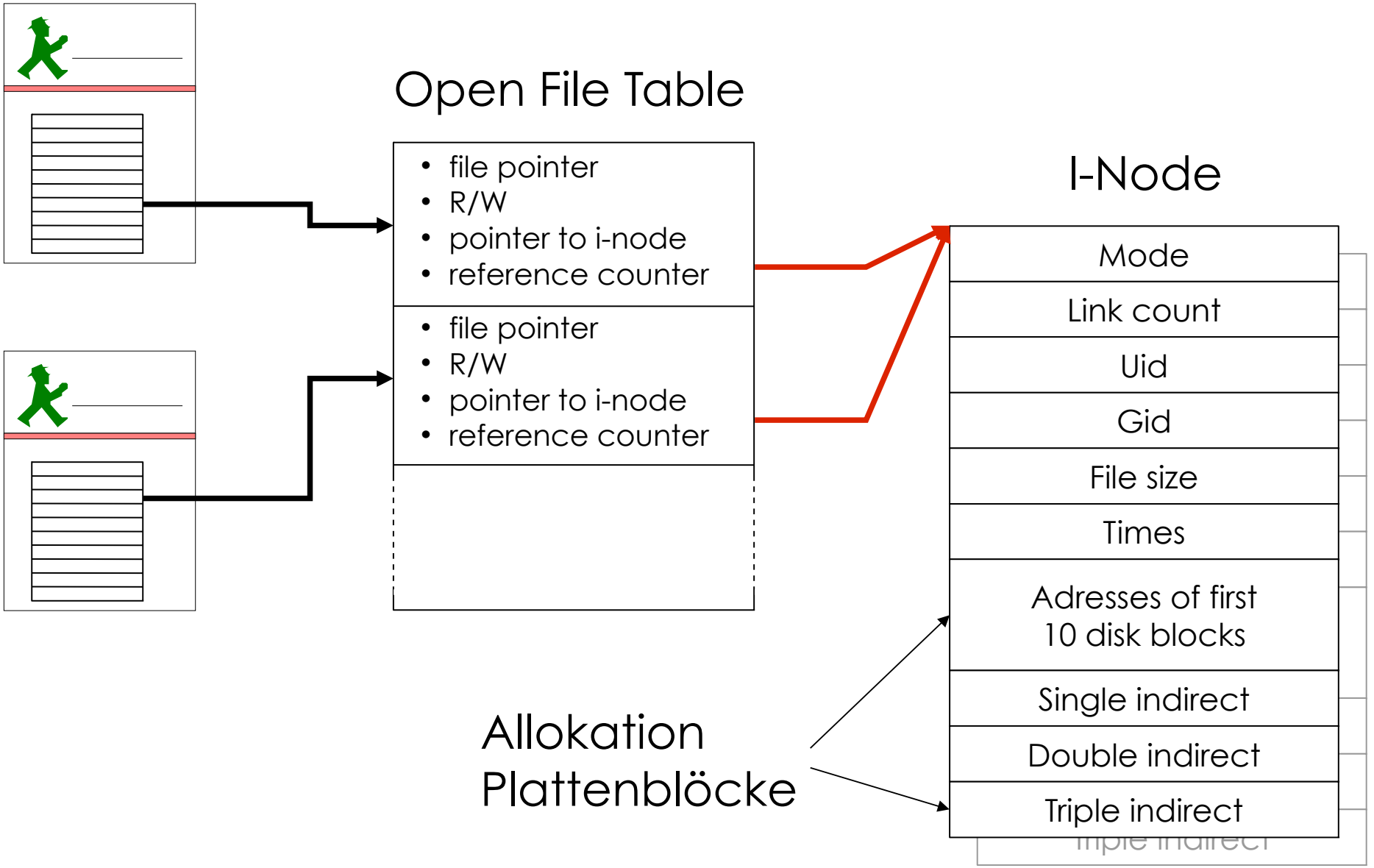
vereinfacht: ignoriert werden

- Fehlerbehandlung
- Attribute, Rechte
- Vorhandensein mehrerer Dateisysteme pro Rechner
- Plattentreiberdetails

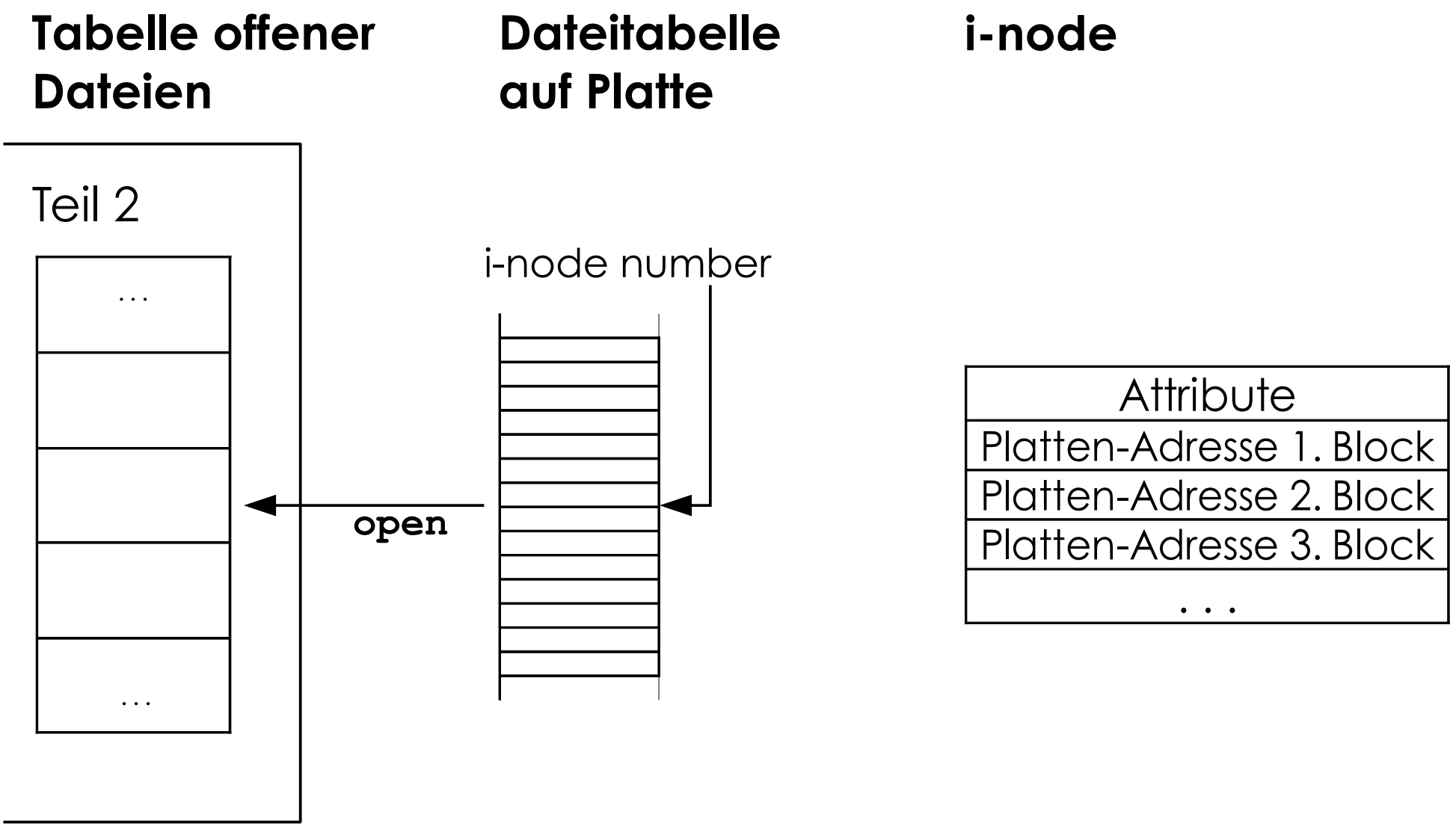
Beteiligte Datenstrukturen (des Dateisystems)

- Prozesssteuerblock: current directory, Filedescriptor fd
- Verzeichnisse
- Tabelle geöffneter Dateien, pro Datei:
  - file pointer (gegenwärtige Schreib-/Leseposition), ...
  - i-node number
- Dateitabelle auf Platte (Tabelle der I-Knoten)
  - Index in dieser Tabelle (i-node number) identifiziert Datei
- für alle (auch nicht geöffnete) Dateien: i-node
  - Allokation (Zuordnung von Dateiblöcken zu Plattenblöcken)
  - Attribute
- Puffer-Verwaltung


Prozess
in
out
err


Open File Table
file pointer
R/W
pointer to i-node
reference counter
file pointer
R/W
pointer to i-node
reference counter
I-Node
Mode
Link count
Uid
Gid
File size
Times
Adresses of first
10 disk blocks
Single indirect
Double indirect
Triple indirect
Allokation
Plattenblöcke

i-node (I-Knoten)



Pufferverwaltung und Ablauf read/write

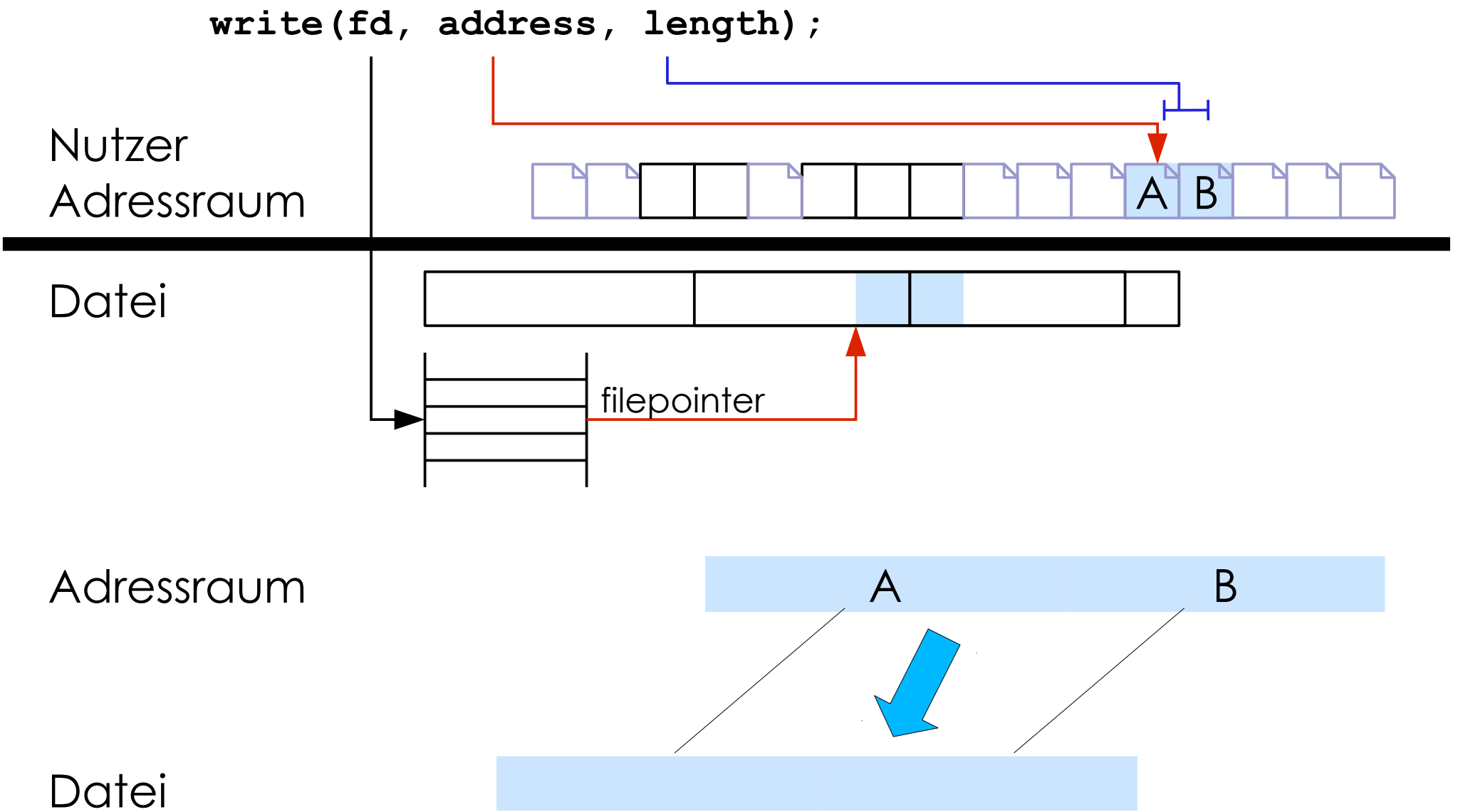
Block-Kachel-Tabelle
(„Buffer Cache“)

- Struktur im Kernadressraum
- Zuordnung:
  Plattenadresse <-> Puffer
- Puffer allokieren/laden
  (resident im Hauptspeicher)

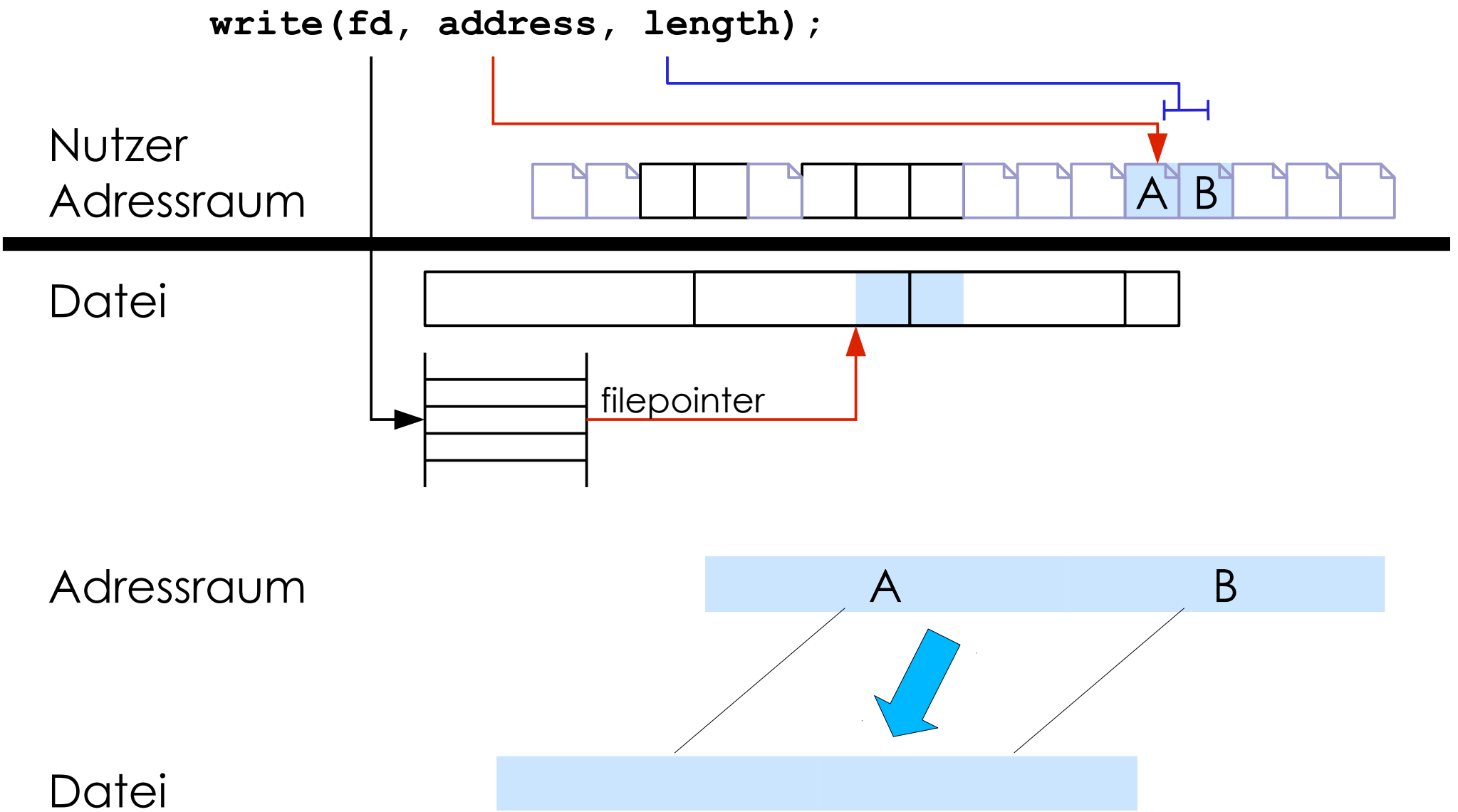
Ablauf (read/write)

- fd → file pointer, i-node
  → Plattenadresse
- Puffer
  - evtl. Puffer besorgen
  - evtl. Laden von Platte
- Kopieren in Adressraum

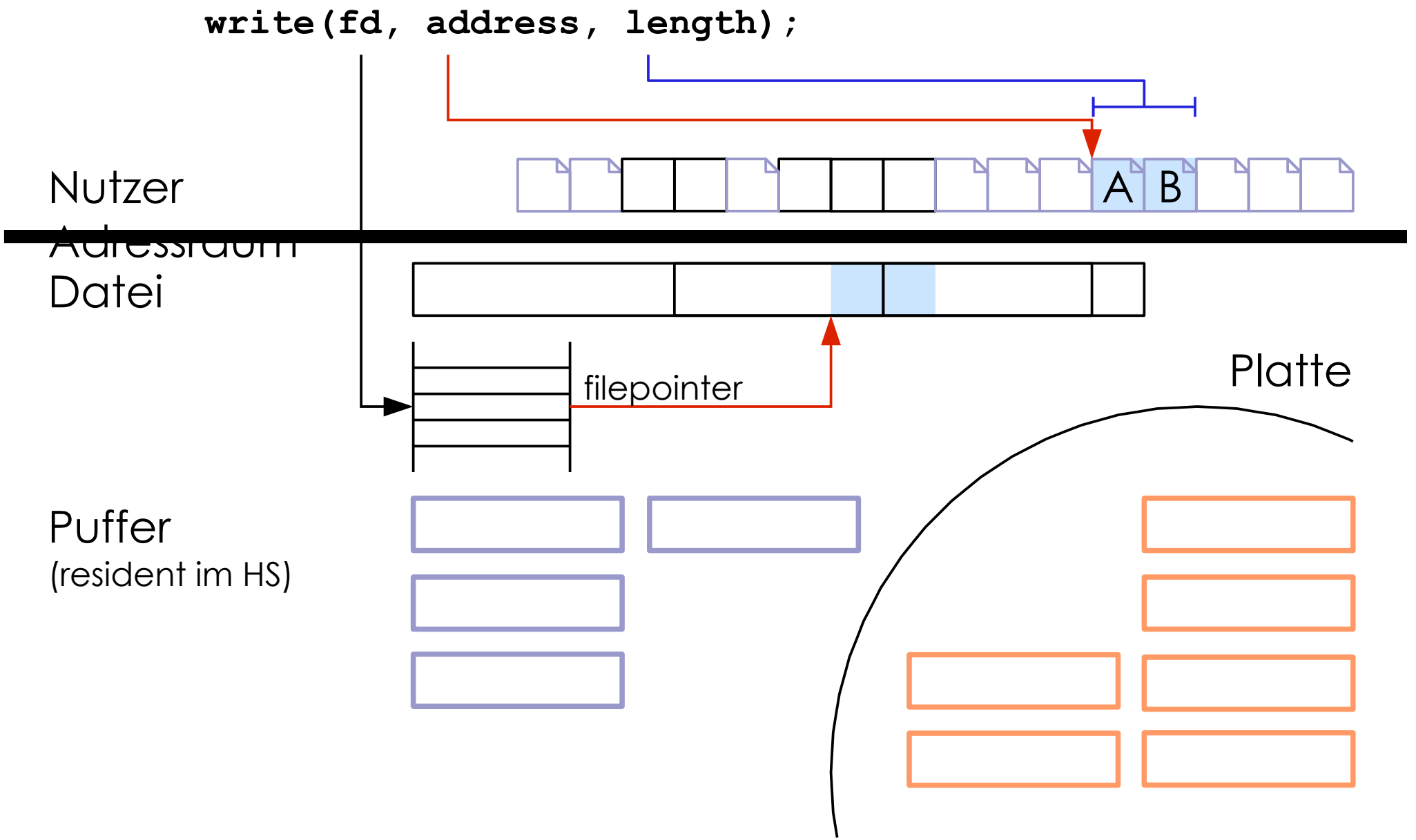
Ablauf write



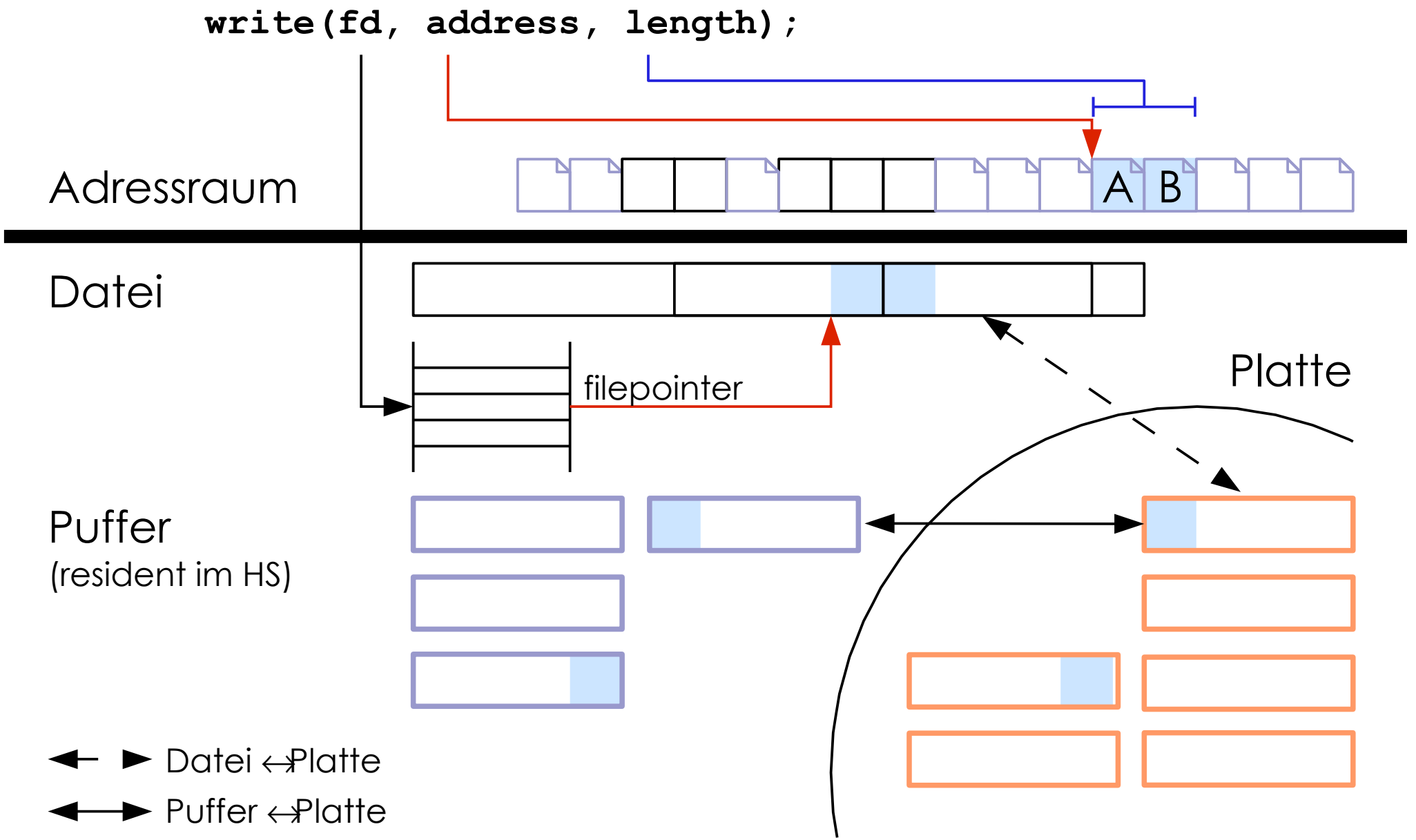
Ablauf write



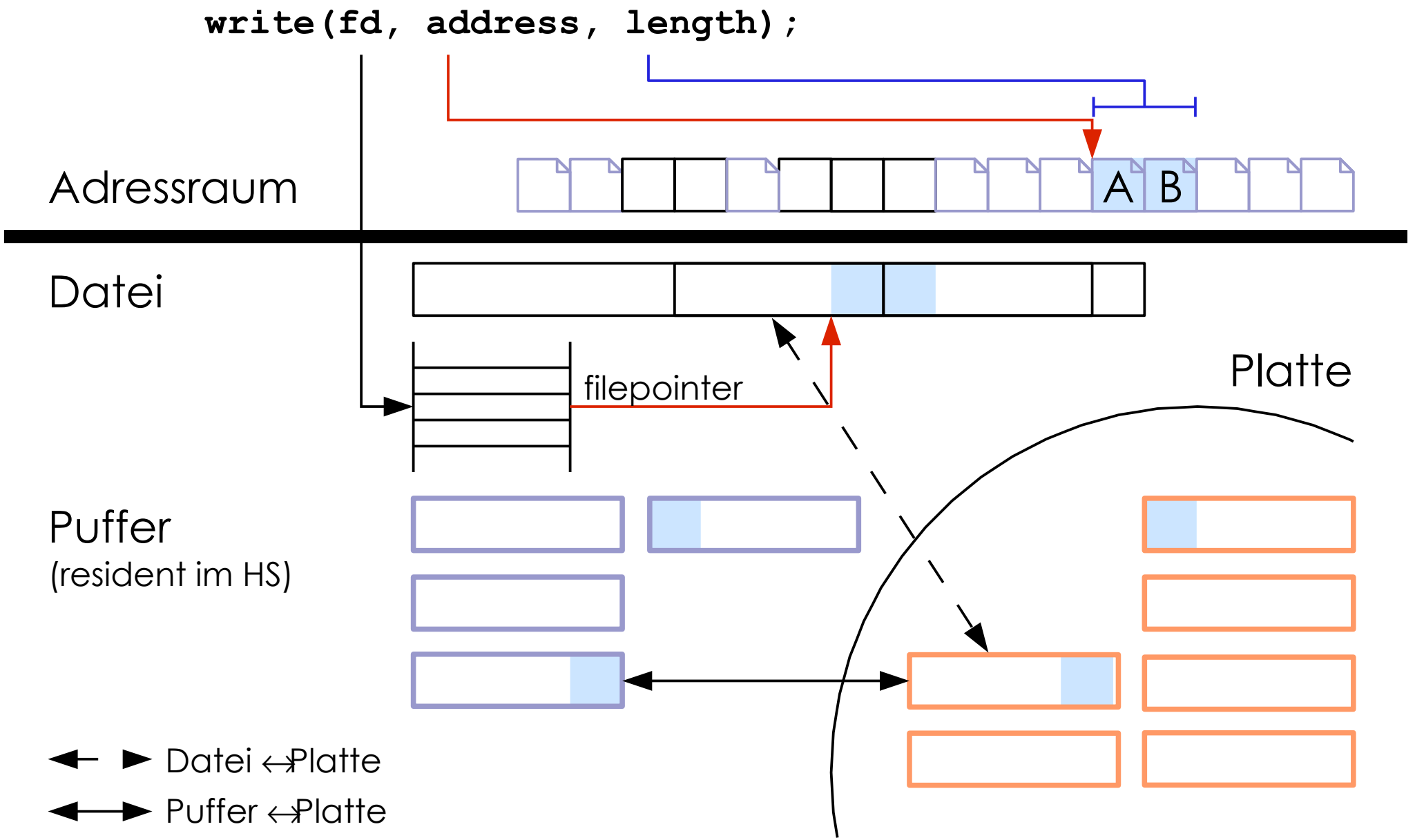
Ablauf write



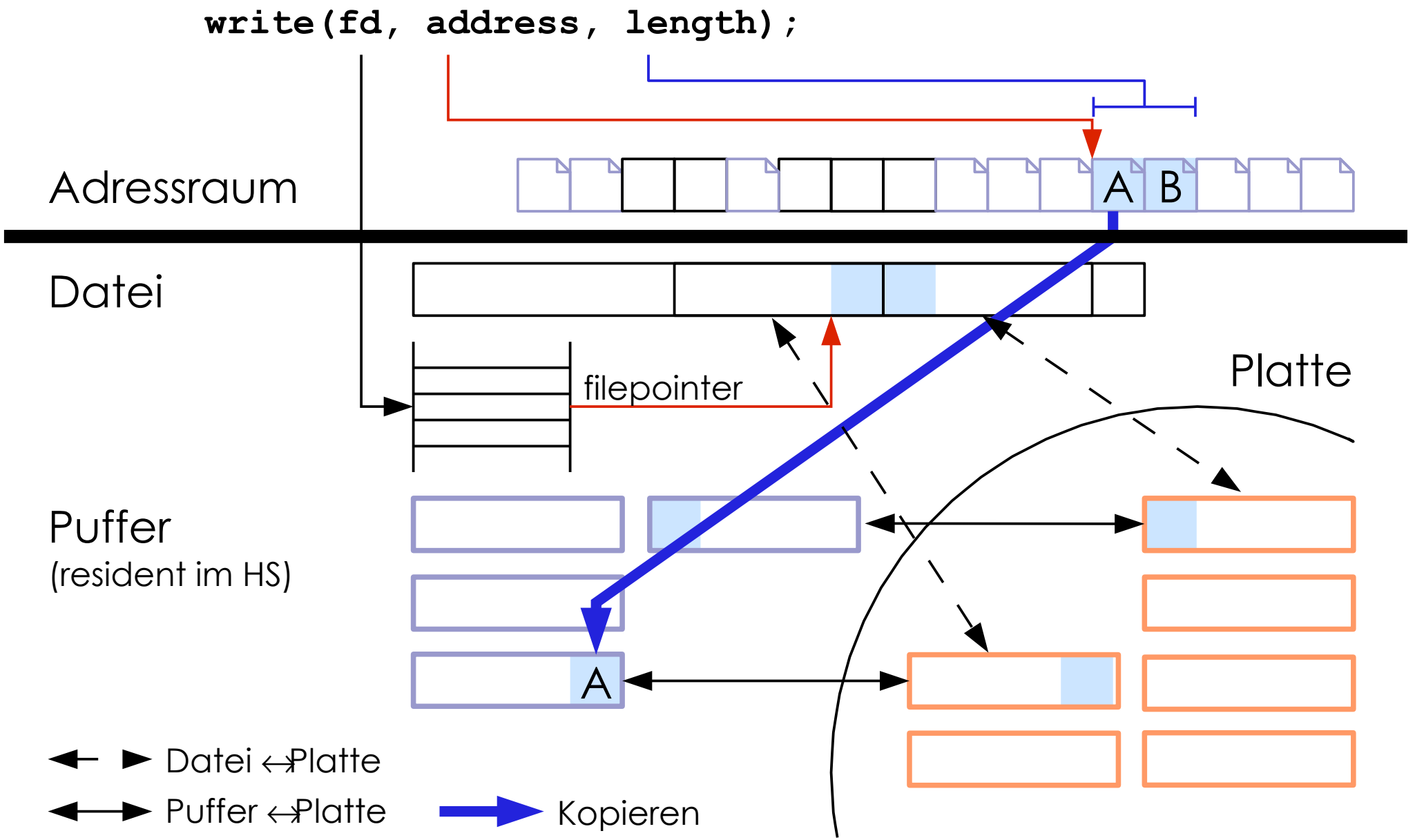
Ablauf write



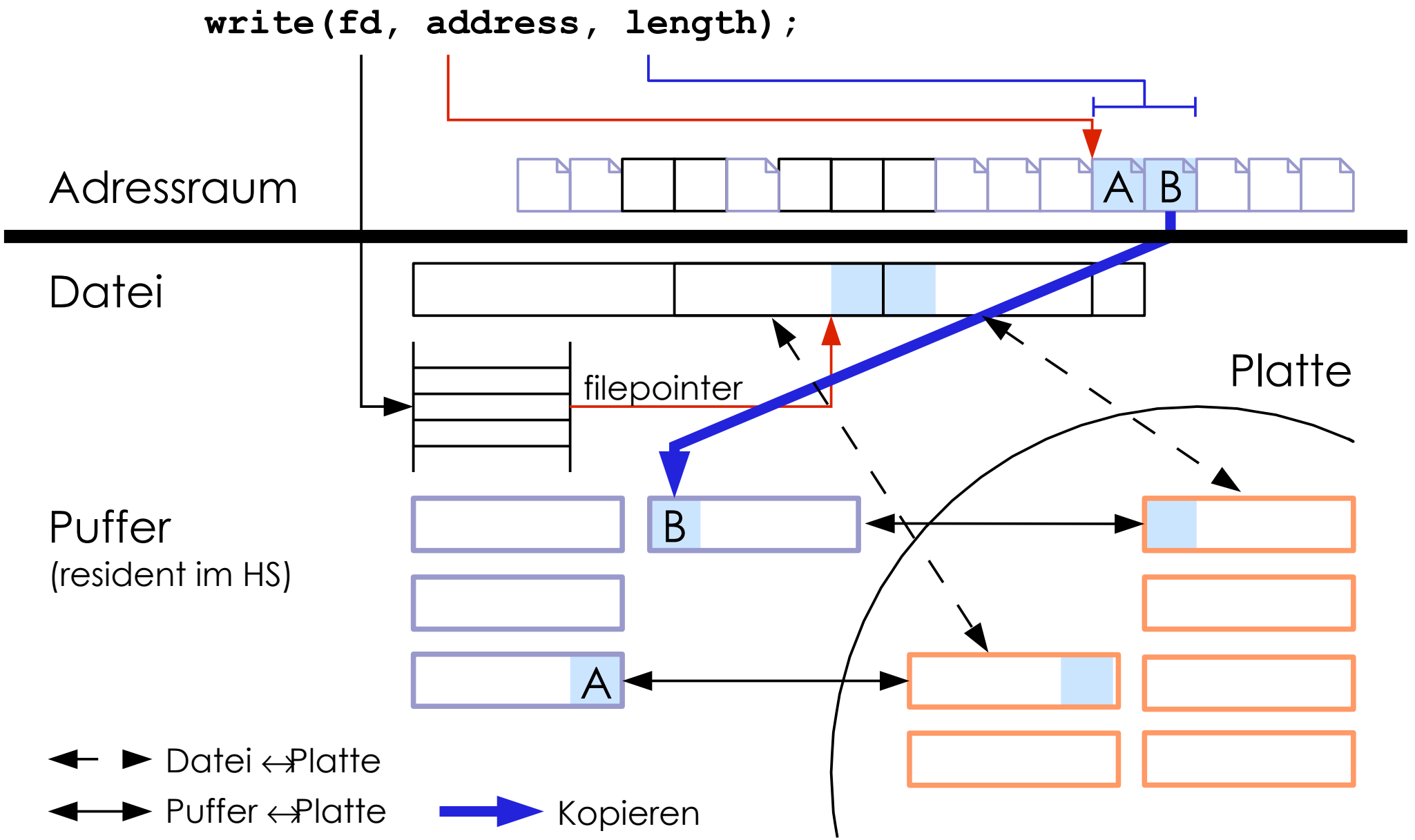
Ablauf write



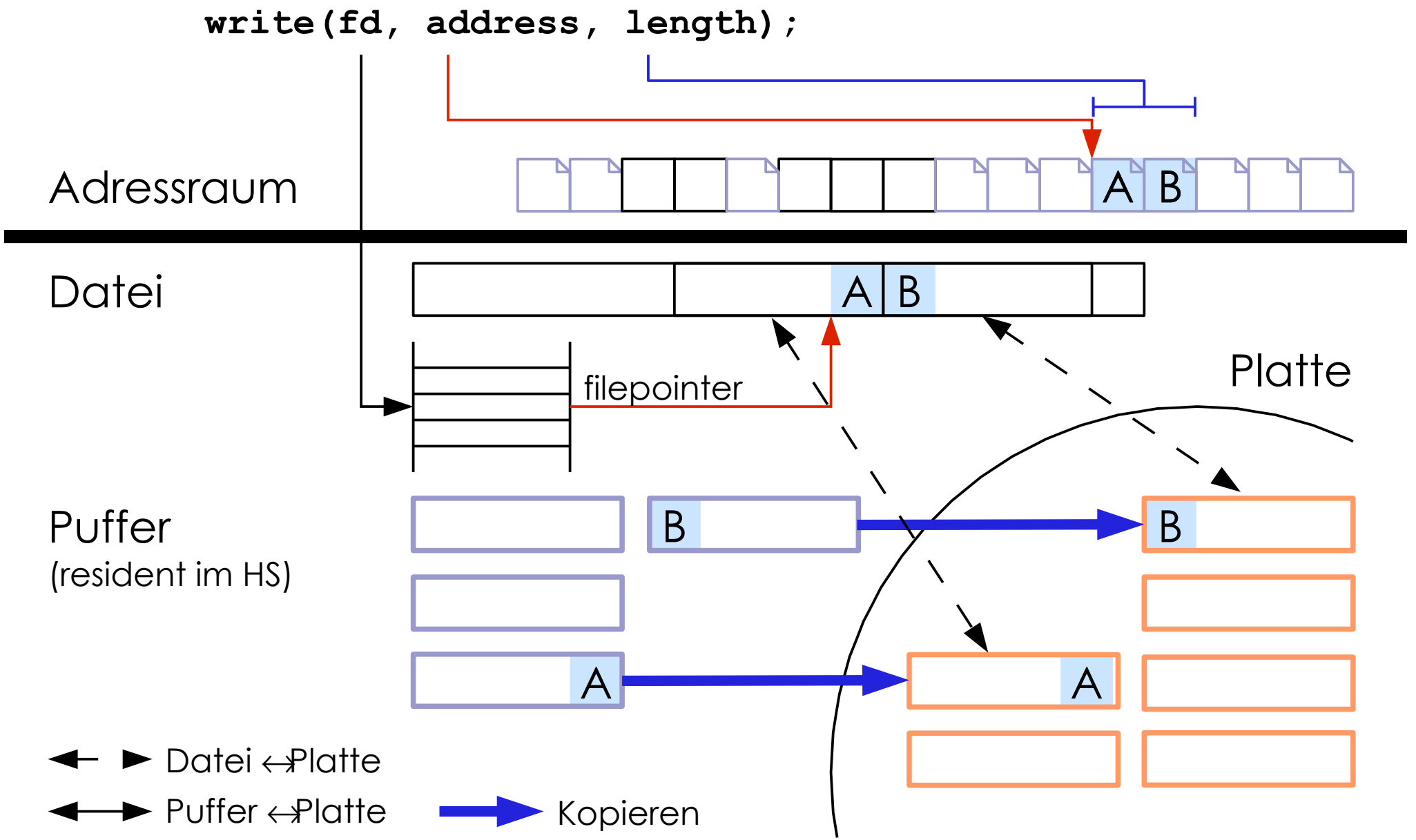
Ablauf write



Ablauf write



Ablauf write



Gegenüberstellung: read/write vs. map

read/write

- Kopie vor jeder Manipulation
- mehrere Prozesse können sich per read eine Kopie holen und dann den vom anderen Prozess vorher geschriebenen Wert überschreiben

mmap

- Einsparen von Kopieroperationen **!!!**
- Synchronisation wie bei gemeinsamem Speicher

Gegenüberstellung: read/write vs. map

read/write

- Kopie vor jeder Manipulation
- mehrere Prozesse können sich per read eine Kopie holen und dann den vom anderen Prozess vorher geschriebenen Wert überschreiben

mmap

- Einsparen von Kopieroperationen **!!!**
- Synchronisation wie bei gemeinsamem Speicher

Beispiel: Konto

```
open(Kontodatei, ... );
```

```
KontoAdrInDatei =
              Find_Konto;
read(Kontodatei,
    KontoAdrInDatei,
    &Konto, Länge);

Konto.Stand += Betrag;

write(Kontodatei,
    KontoAdrInDatei,
    &Konto, Länge);
```

```
mmap (Kontodatei, ...);

KontoAdr = Find_Konto;

KontoAdr->Stand += Betrag;
```

Beispiel: Konto

```
open(Kontodatei, ... );
```

```
KontoAdrInDatei =
              Find_Konto;
read(Kontodatei,
    KontoAdrInDatei,
    &Konto, Länge);

Konto.Stand += Betrag;

write(Kontodatei,
    KontoAdrInDatei,
    &Konto, Länge);
```

```
mmap (Kontodatei, ...);

KontoAdr = Find_Konto;

KontoAdr->Stand += Betrag;
```

Zusammenfassung

- Integration von Dateisystemen in Systemarchitektur
- Prinzipielle Teilaufgaben
- Ein einfaches altes Beispiel

ABER: Limitationen

Kleine Blöcke, riesige Dateien
Ausfall von Platten
Absturz eines BS → Verlust von Pufferinhalten, Inkonsistenzen
Platte ./. Flash-Speicher

→ spätere Kapitel in dieser Vorlesung