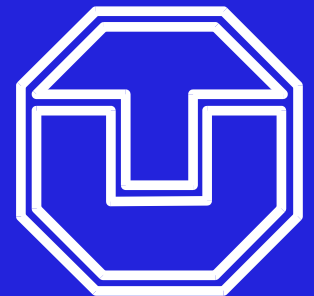


Ein einfaches Dateisystem

Betriebssysteme
WS 2016/17

Hermann Härtig
TU Dresden



Gegenstand

Aufgaben eines Dateisystems

- **Persistenz** von Daten, d. h. Daten sollen Prozess- und Systembeendigungen überstehen (auch Abstürze)
- Basismechanismen für Datenbanken
- manchmal: Schutz (separate Kapitel dieser Vorlesung)

Zentrale Herausforderungen

- Hardwareeigenschaften des persistenten Speichermediums
 - Robustheit
- separate Kapitel in dieser Vorlesung

In dieser Vorlesungs-DS

- ein ganz einfaches Dateisystem (ca Unix 1980)
- und seine Integration in Systemarchitektur

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) Name
- 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“

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 (unsichtbar)

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)

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

„Record“-Strukturen (satzstrukturierte Dateien)

- Zugriff per “key”
- Länge der Records konstant oder variabel
- ein Datensatz (record) pro Zugriff (Lesen/Schreiben)
 - sequentiell
 - direkter Zugriff über Schlüssel
 - Kombiniert: “indexsequentiell”
- “Key-Value Stores”

Verzeichnisse und Pfadnamen

Ziel:

- systemweite Eindeutigkeit von Dateinamen

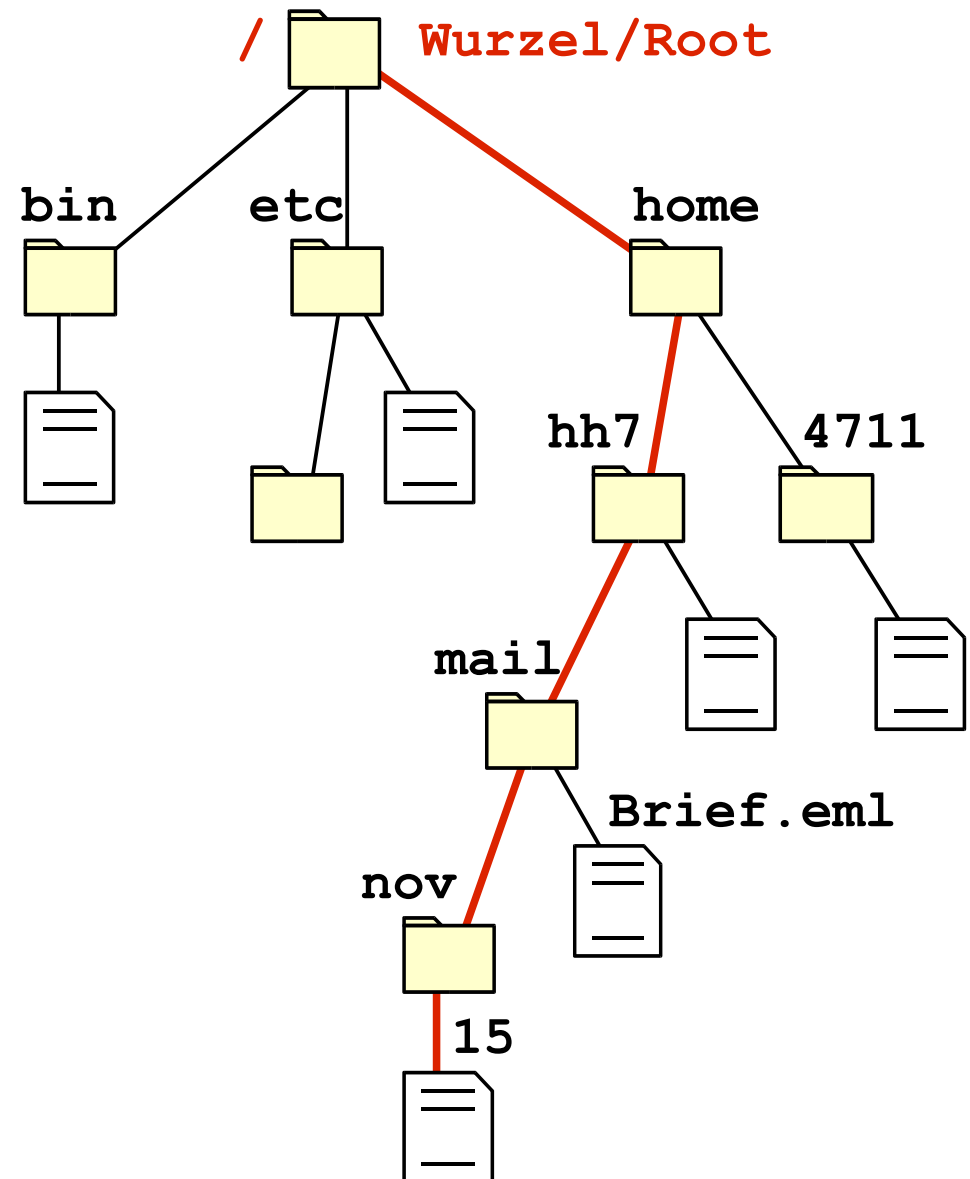
Lösung:

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

Pfadname:

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

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



Verzeichnis (Folder, Directory)

Abbildung von Teilnahmen auf Datei-Identifikatoren:

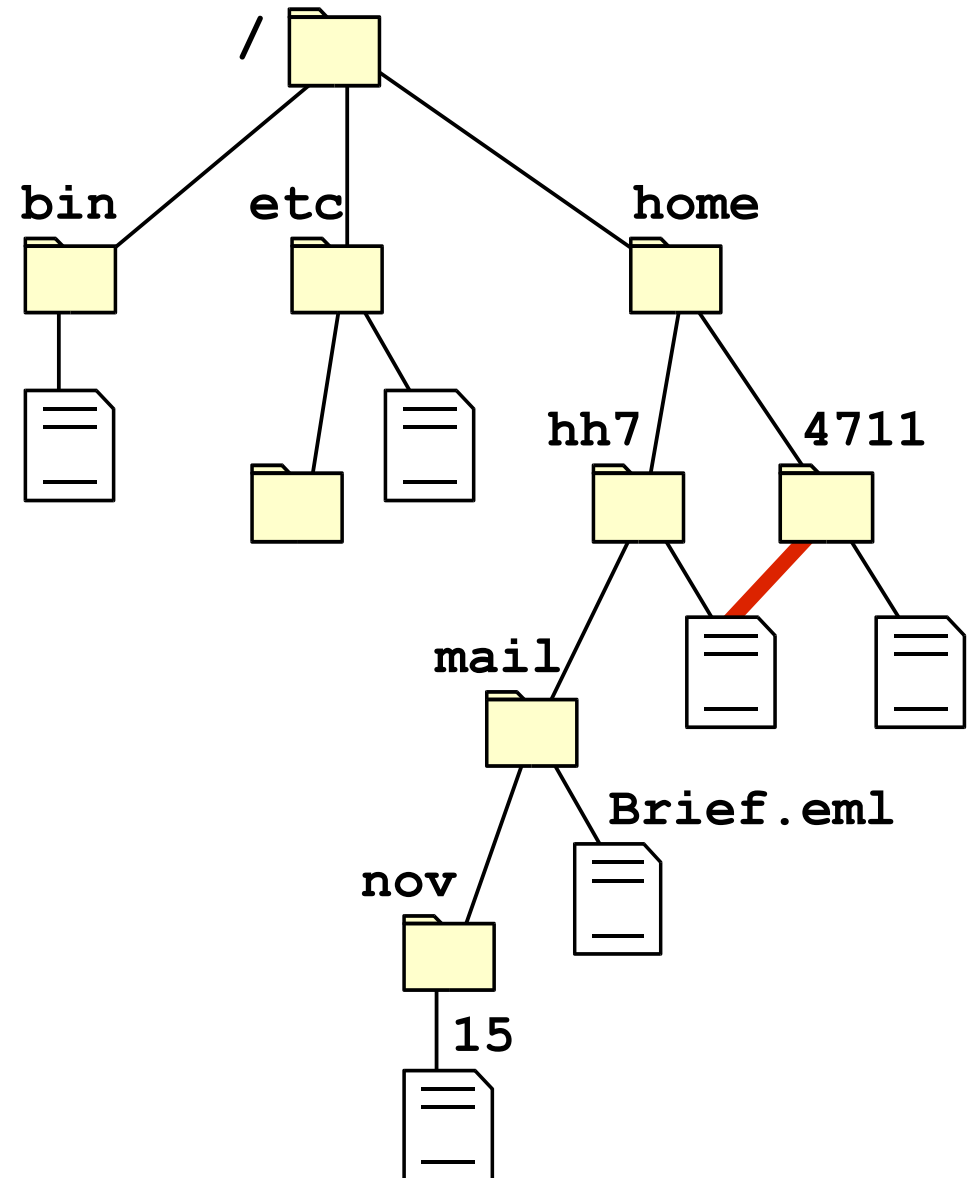
Beispiele:

- Wurzelverzeichnis:
bin → 5, etc → 7, home → 567
- “home”:
hh7 → 89, 4711 → 999

Mehrere Namen für eine Datei: Harte Links

„Harte Links“

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



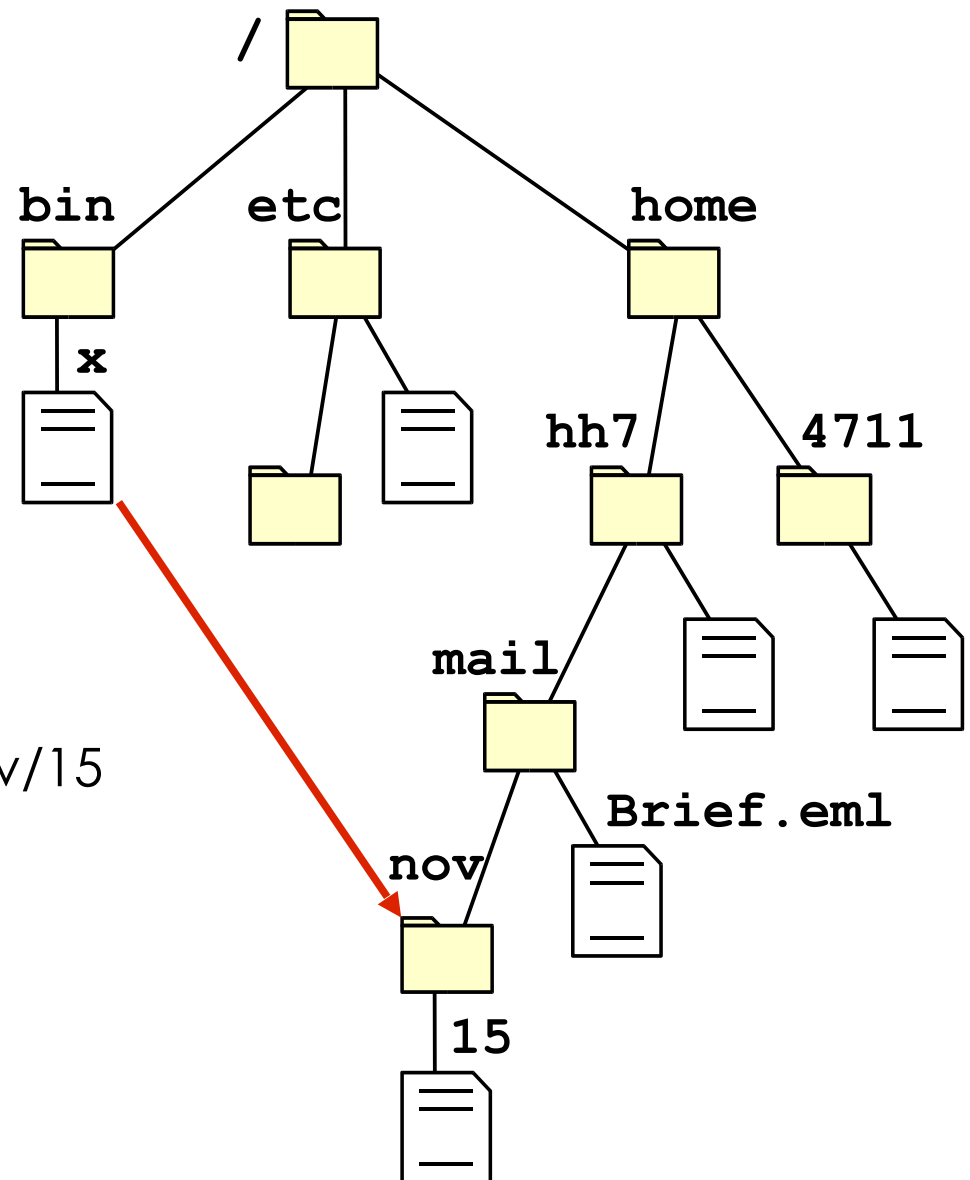
Mehrere Namen für ein Objekt: Weiche Links

„Symbolische Links“

- Namen als Objekte
- Zyklen !

Beispiel

- $x \rightarrow 23$
in 23: `/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

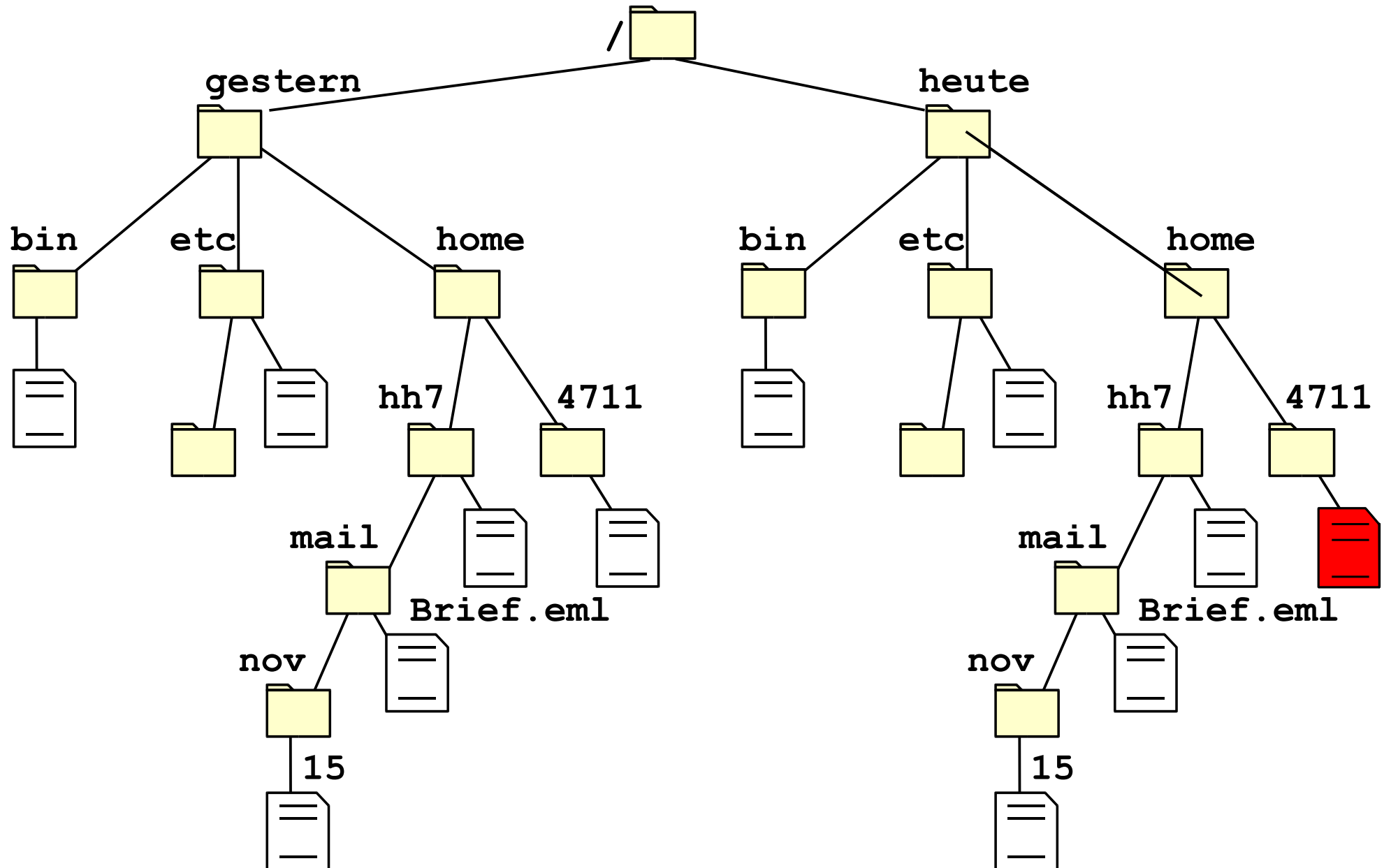
Auflösung von /bin/XYUtils/example

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

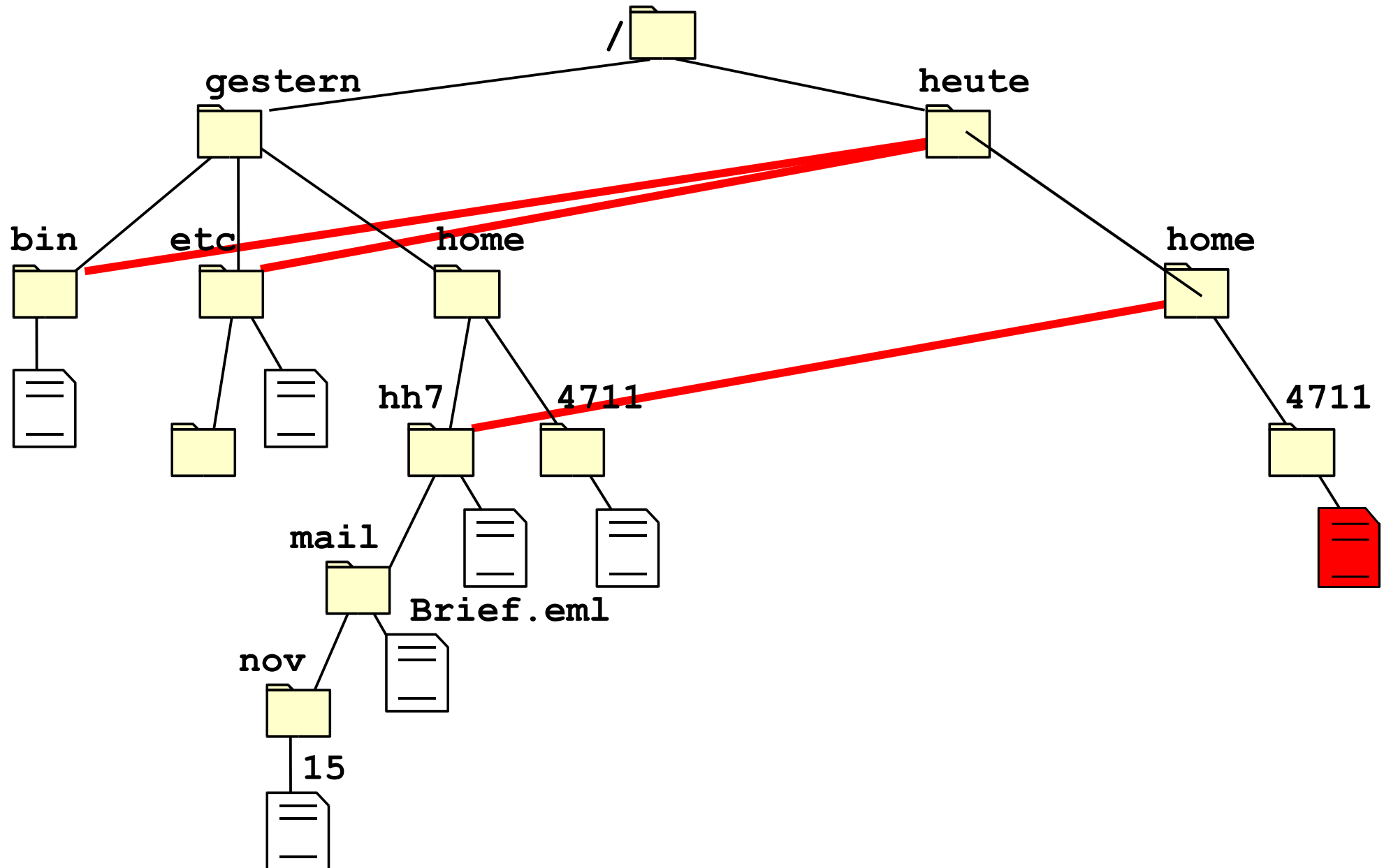
Beispiel harter links: Backup

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

Beispiel: Backup



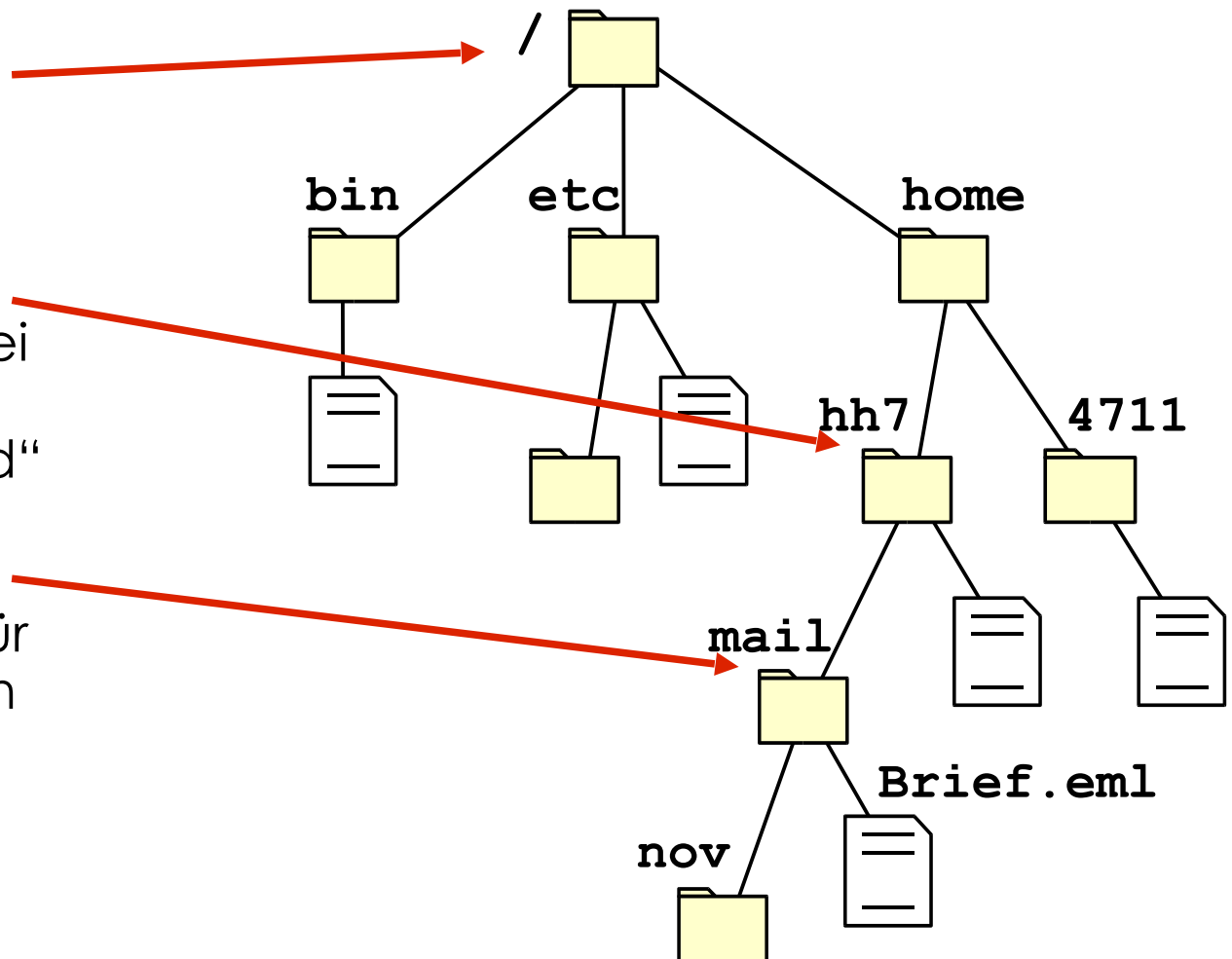
Beispiel: Backup



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

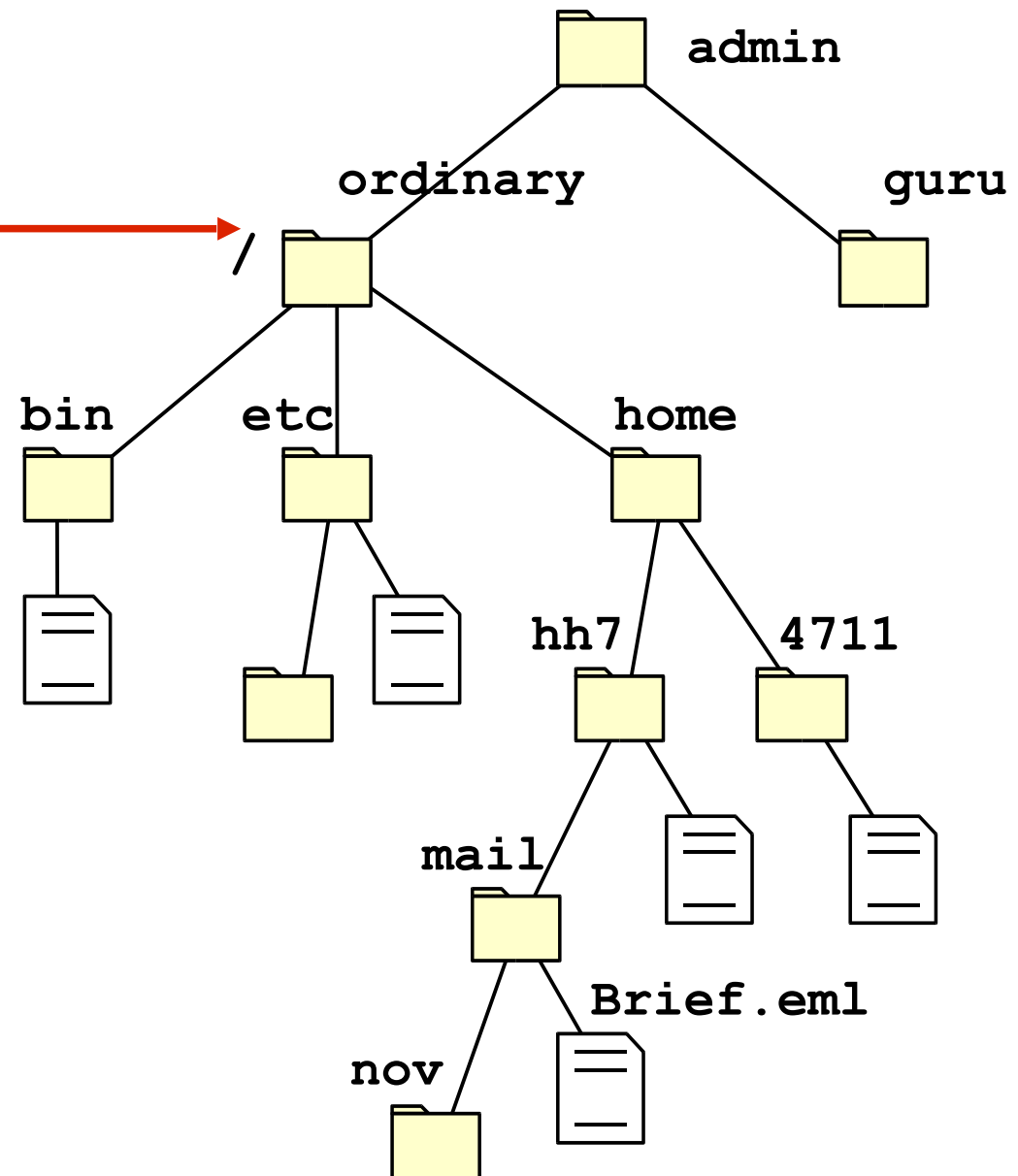
Ausgangs-Kontexte durch Prozesse festgelegt, z. B. in Unix :

- current root:
Ausgangskontext für absolute Pfadnamen

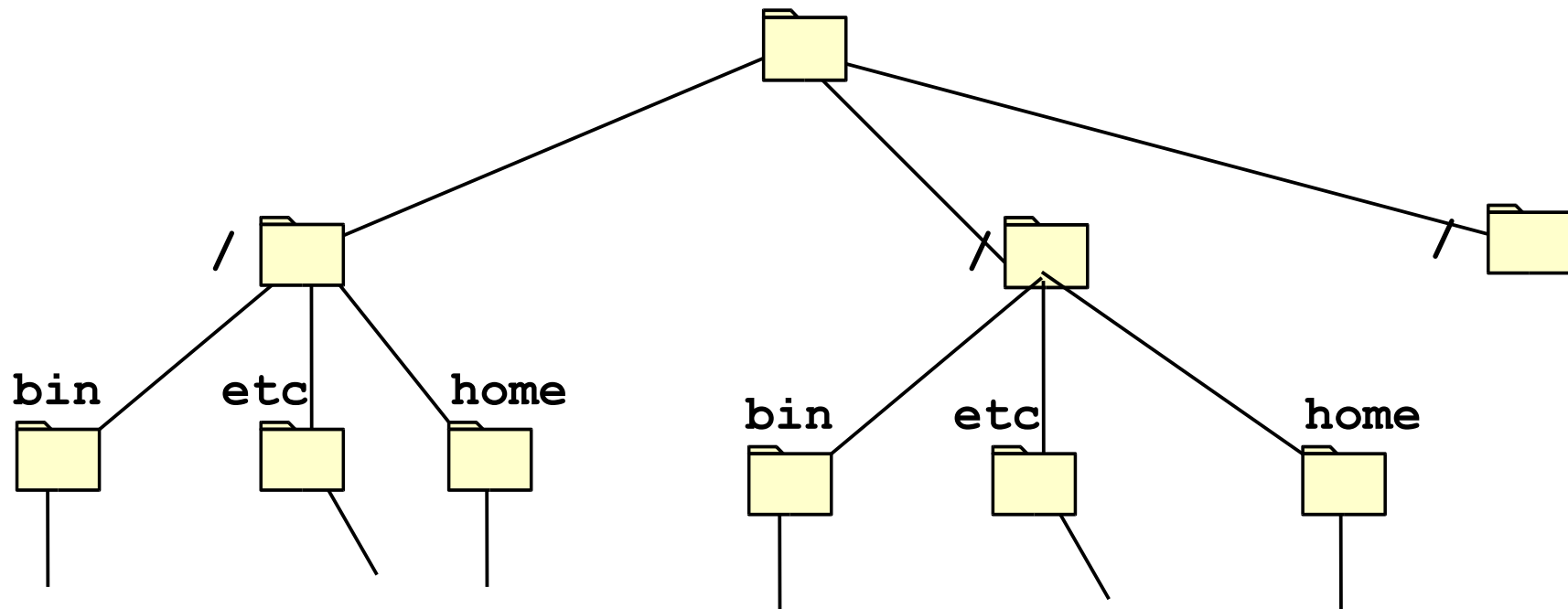
„chroot“ Kommando

mehrere „Globale“ Namensräume

OS Virtualisierung



Schritt in Richtung Virtuelle Maschinen



Rudimentäre Form von
“Container”

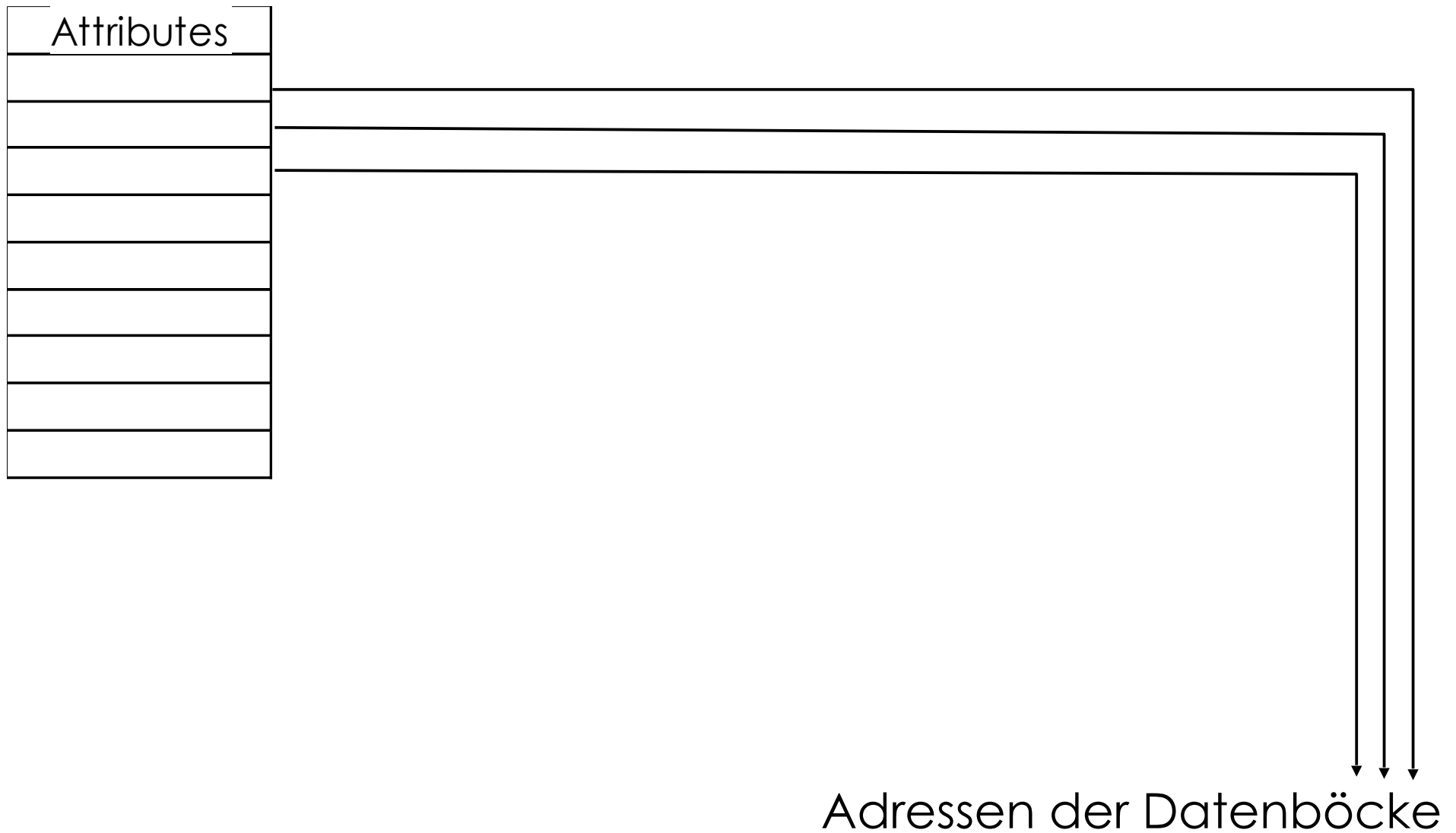
Verzeichnisse

Implementierung und Zugriff analog normale Datei mit:

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

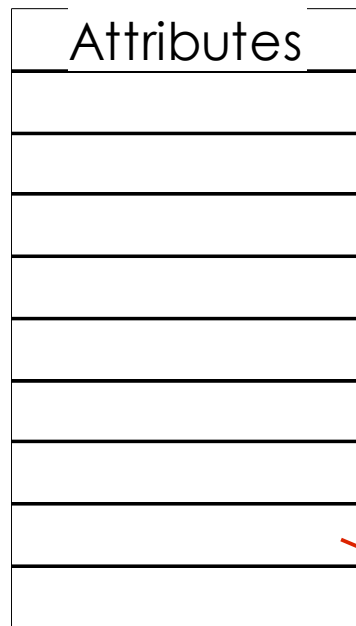
i-nodes (I-Knoten)

i-node



i-nodes (I-Knoten)

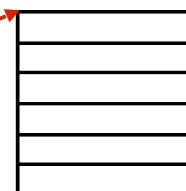
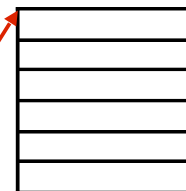
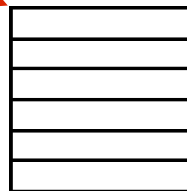
i-node



Single
indirect block



Double
indirect block



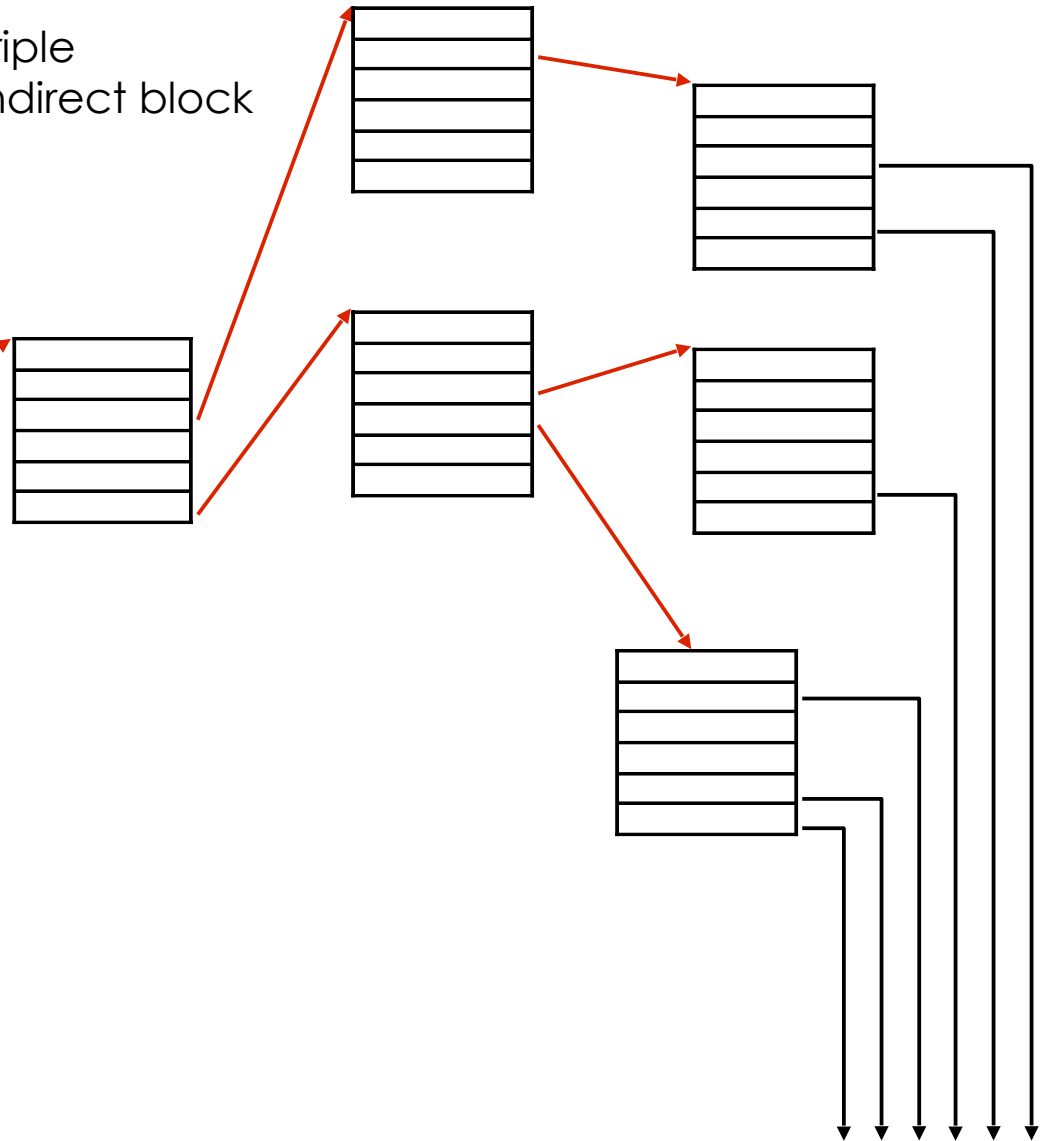
Adressen der Datenblöcke

i-nodes (I-Knoten)

i-node



Triple
indirect block



Adressen der Datenblöcke

Verwaltung des freien Plattenspeichers

Verwendung von Listen

42	230	86
136	162	234
210	612	897
97	324	422
41	214	140
63	160	223
21	664	223
48	216	160
262	320	126
...
310	180	142
516	482	141

Verwendung von Bitmaps

110101010110010101010
000101001111010010101
1010100101010101011101
111111111110000000000
1101010111010110101110
0110110101101001101110
0011010100111010011100
0110101010001100001100
0010101011101010011011
...
0011011101101101001100
11101010010101010010101

Ablauf eines Dateizugriffs

Namensdienste:

symbolische Namen → Identifikatoren

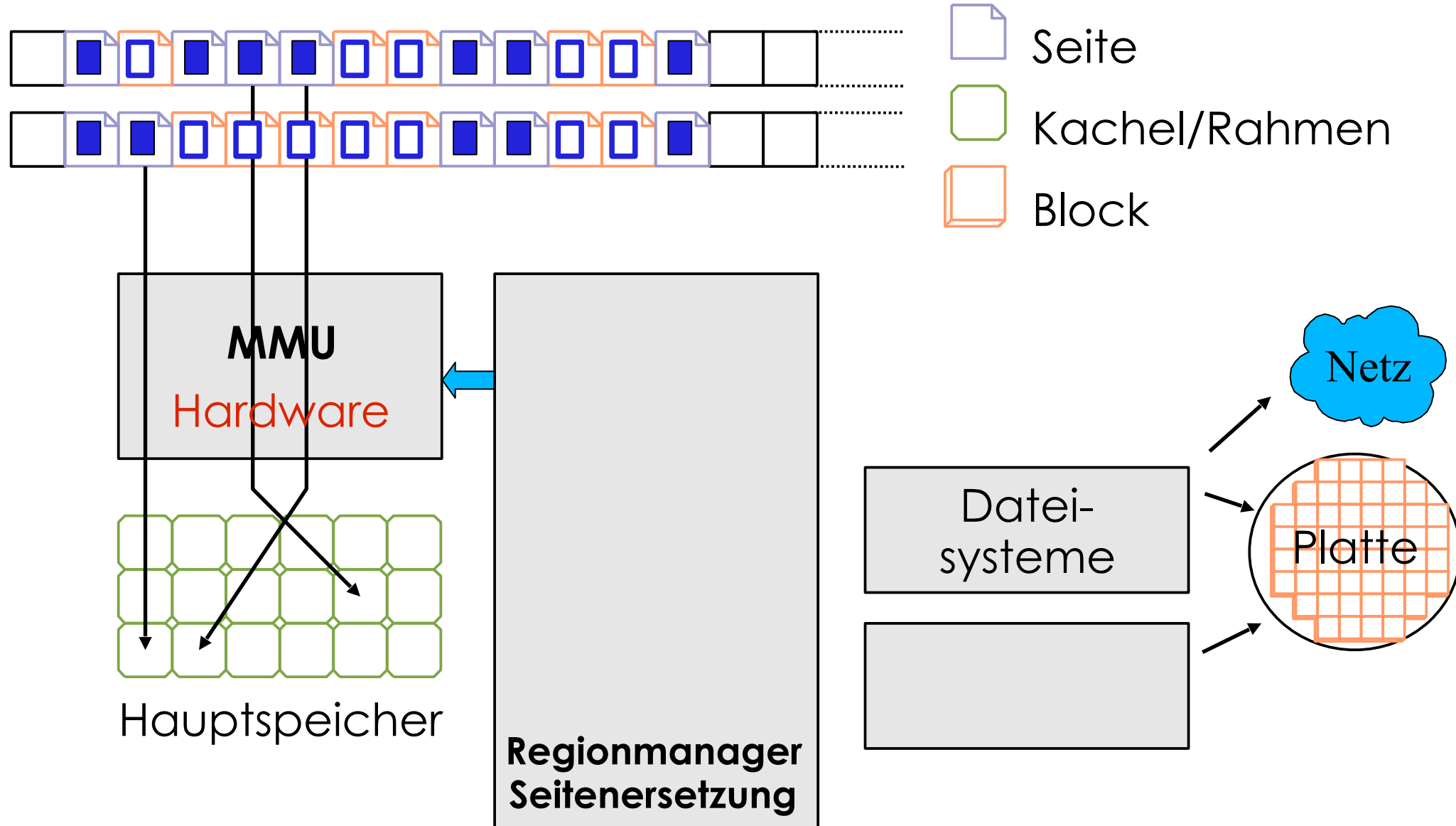
Zugriffe auf (Daten in) Dateien:

- häufig: **open** (Zugriff)* **close**
- Zugriff über
 - Kopieren aus Datei in Adressraum und zurück:
read/write (Datei, Position in Datei, Länge, Adresse im 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 ...

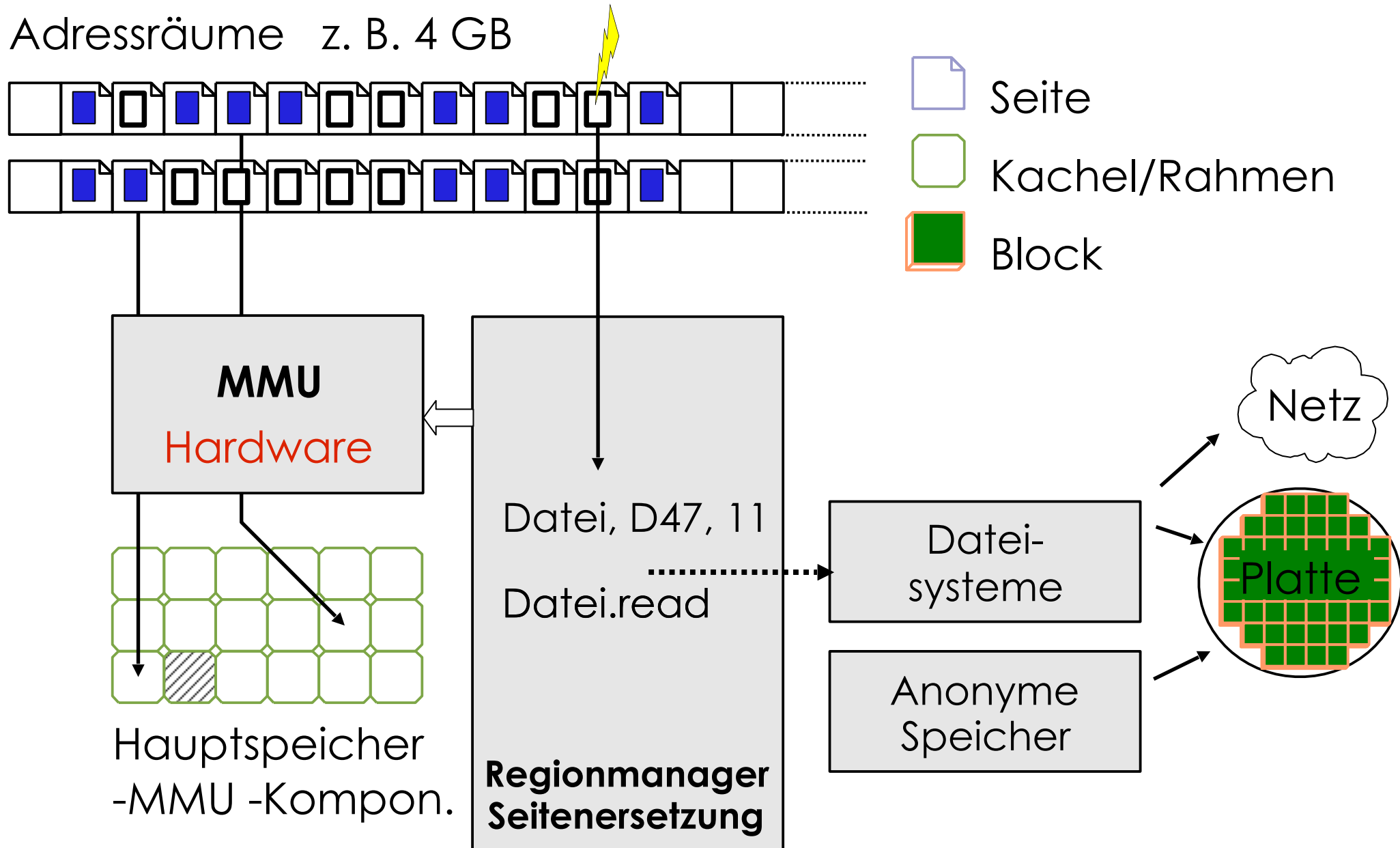
Integration in Systemarchitektur

Adressräume z. B. 4 GB



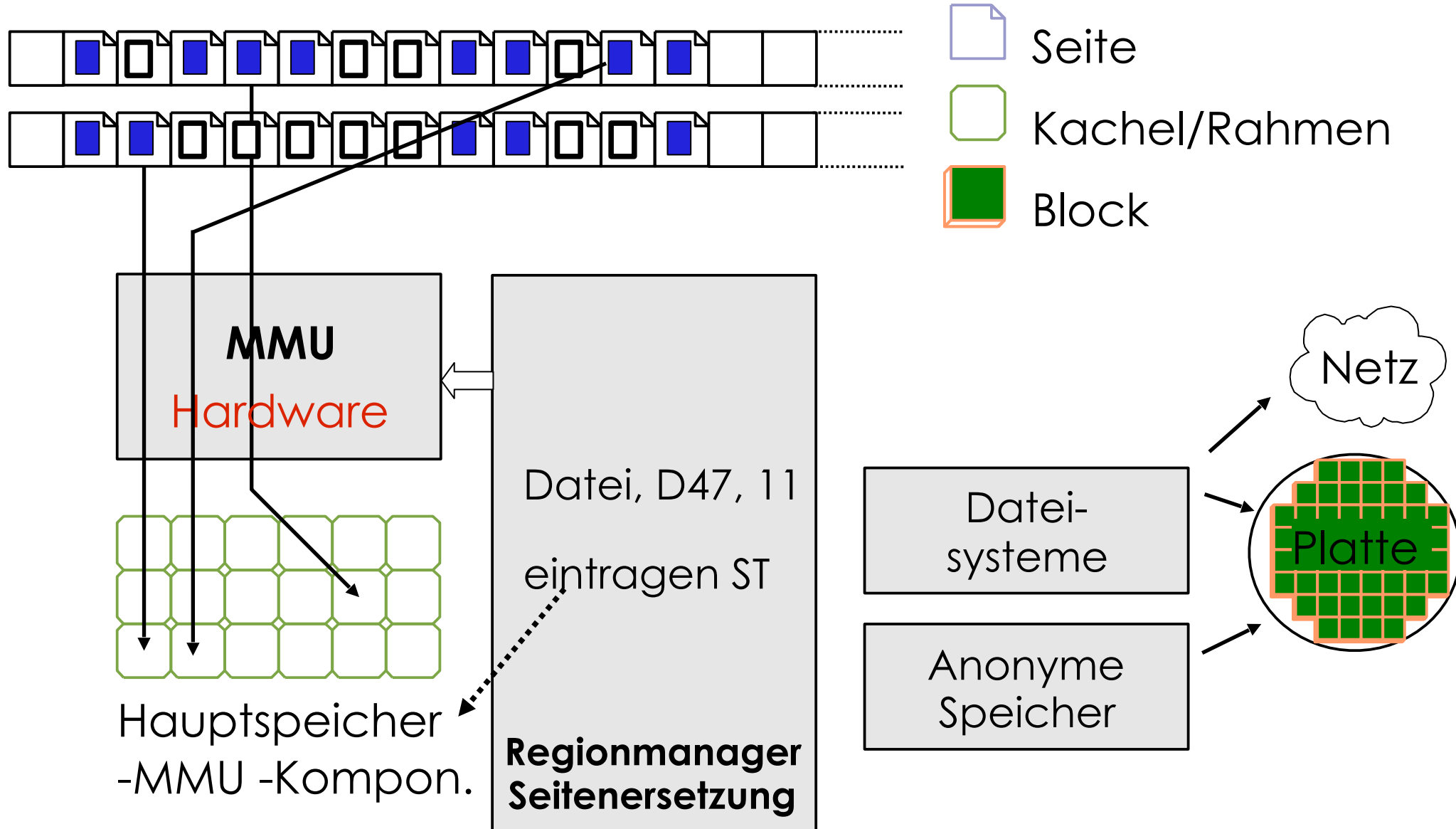
Das Zusammenspiel

Adressräume z. B. 4 GB



Das Zusammenspiel

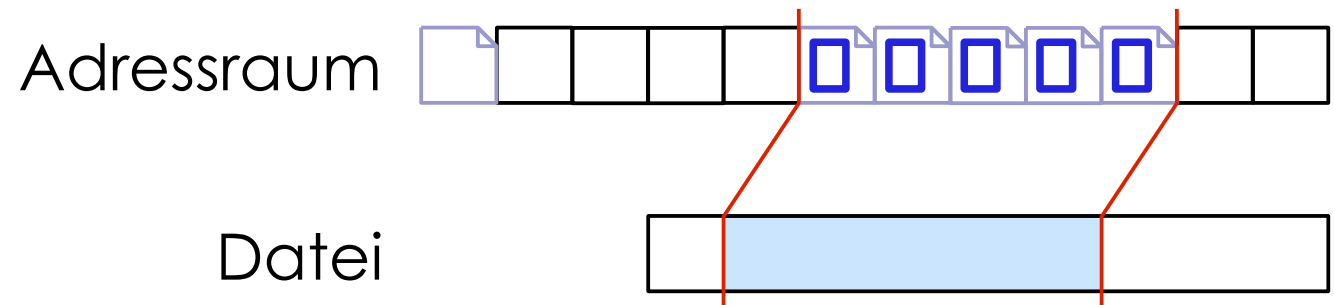
Adressräume z. B. 4 GB



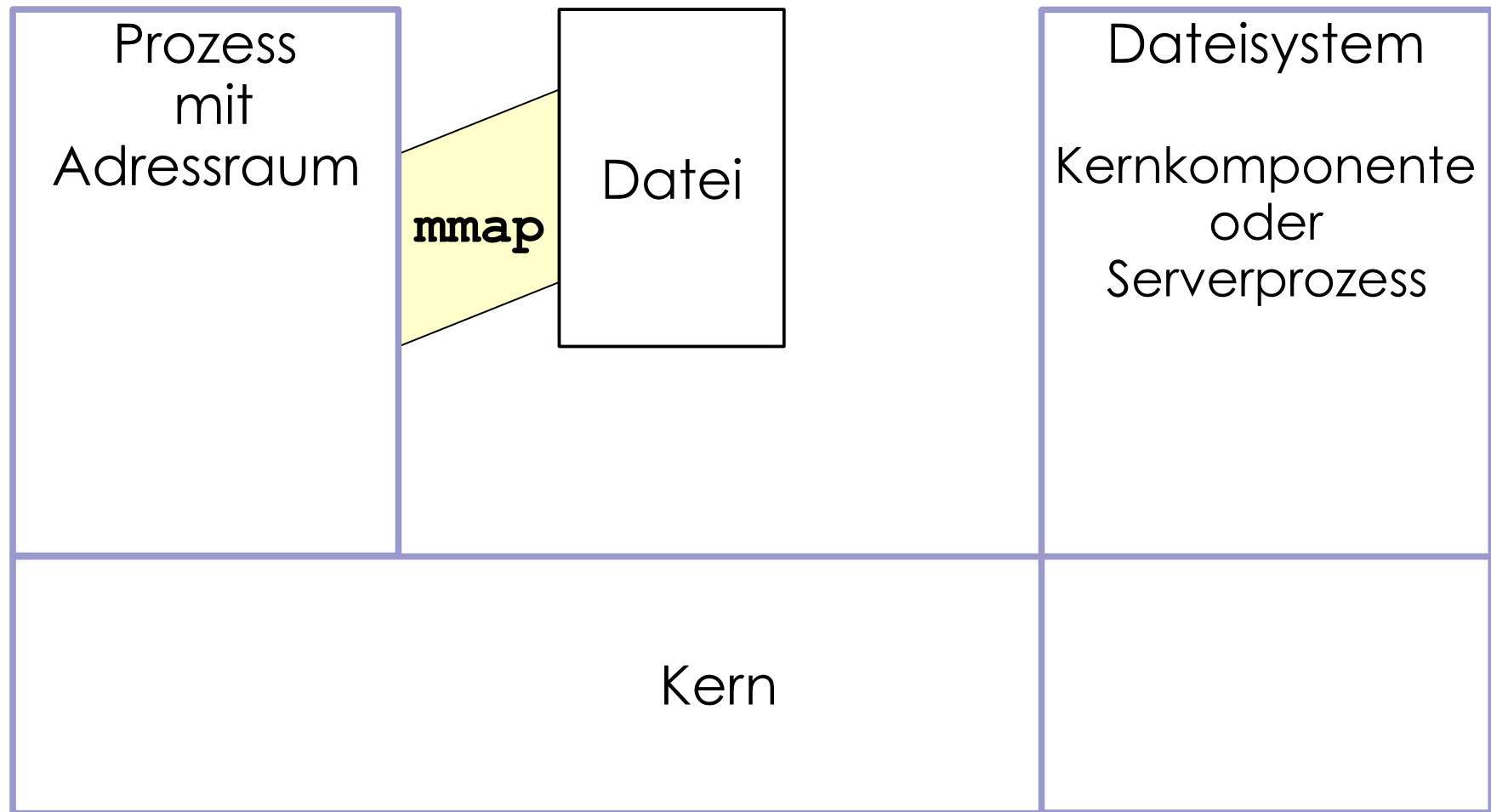
Zugriff mittels Einblendung

mmap (Datei, Adresse, Länge, Offset)

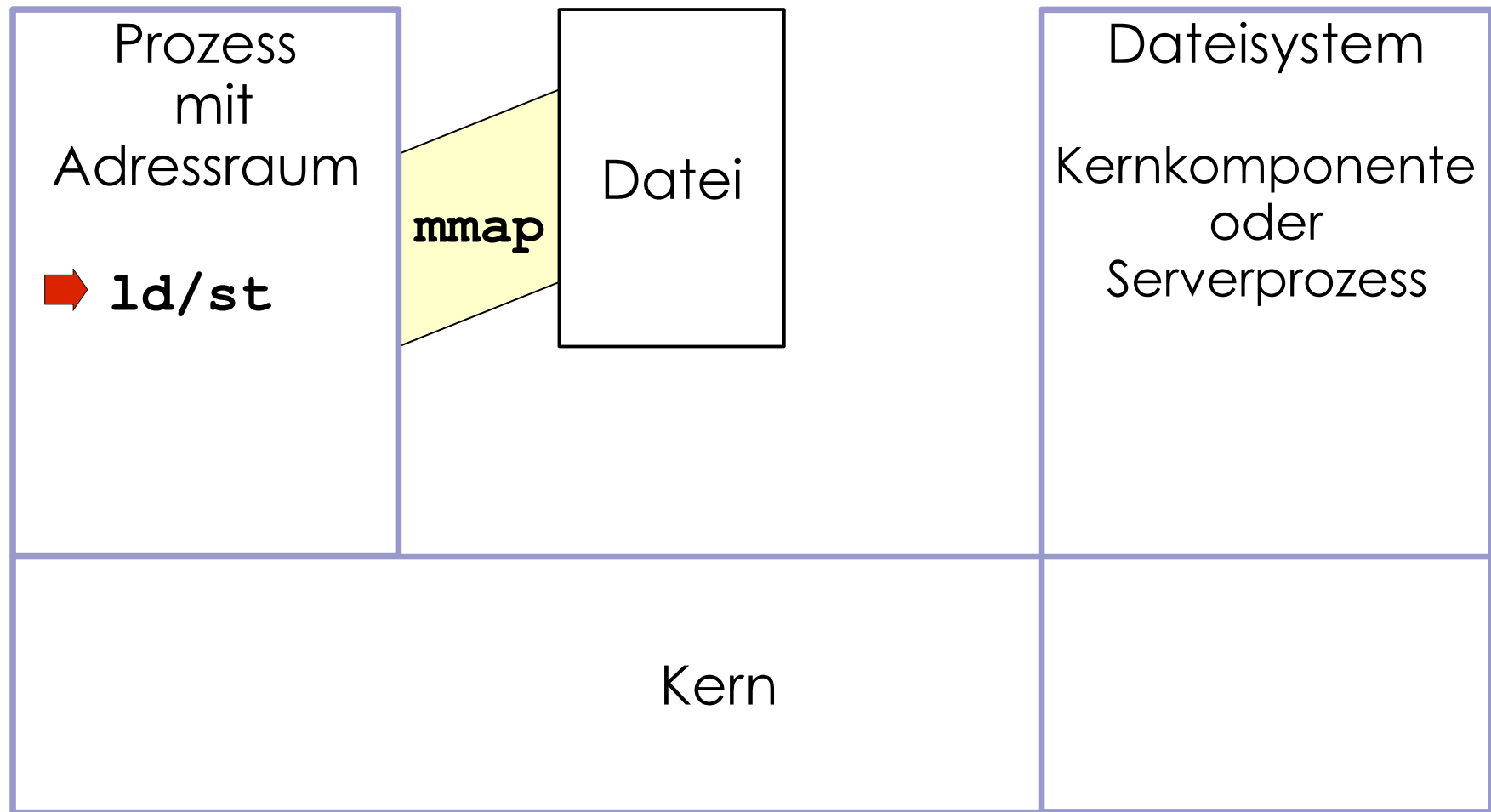
- bei Seitenfehler:
 - Zugriff auf Datei über Seitenfehlerbehandlung
 - dann Zugriff über normale HW-Instruktionen
- „Pager“ muss Struktur der Dateiimplementierung kennen!



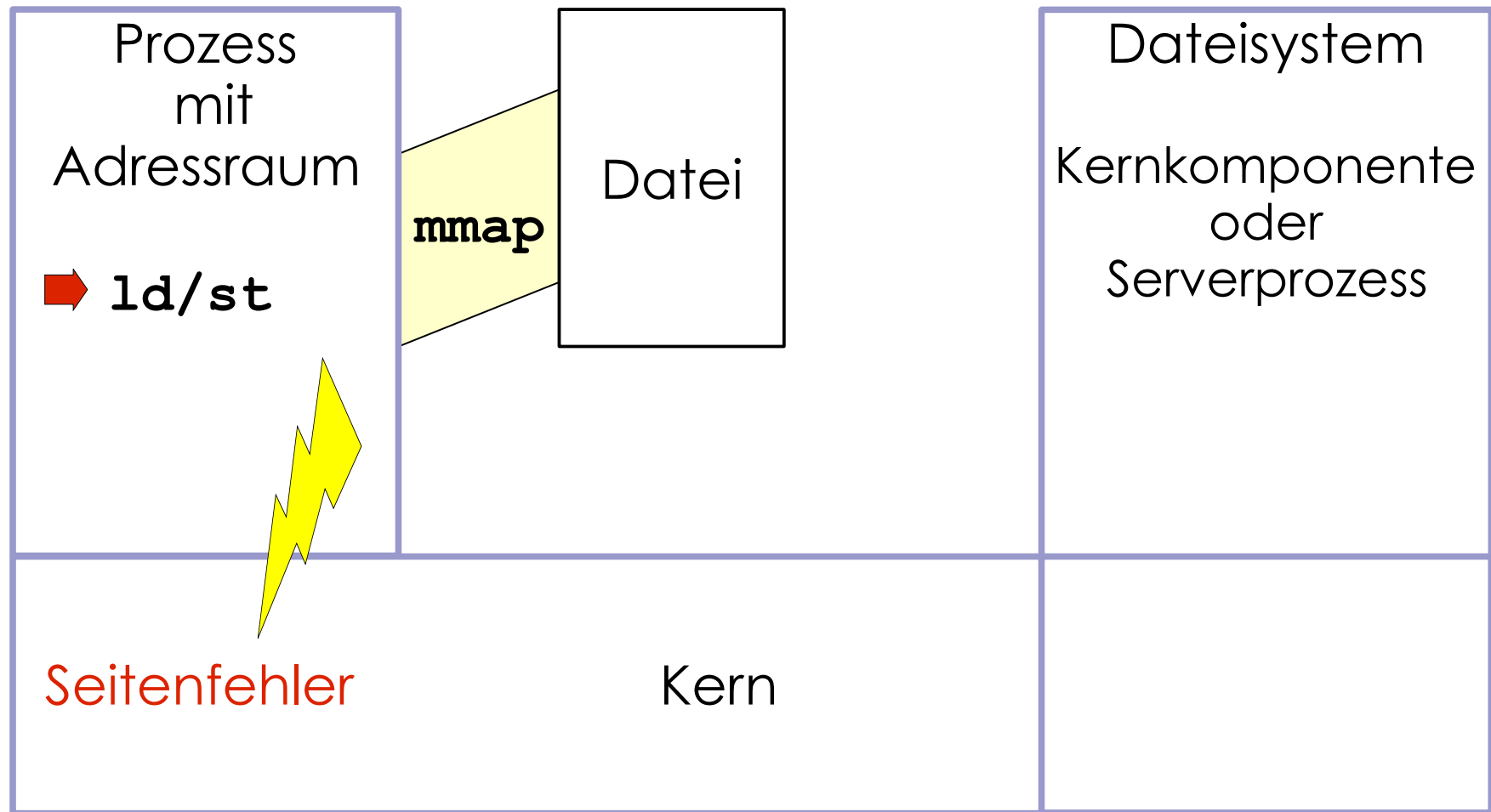
Zugriff mittels Einblendung



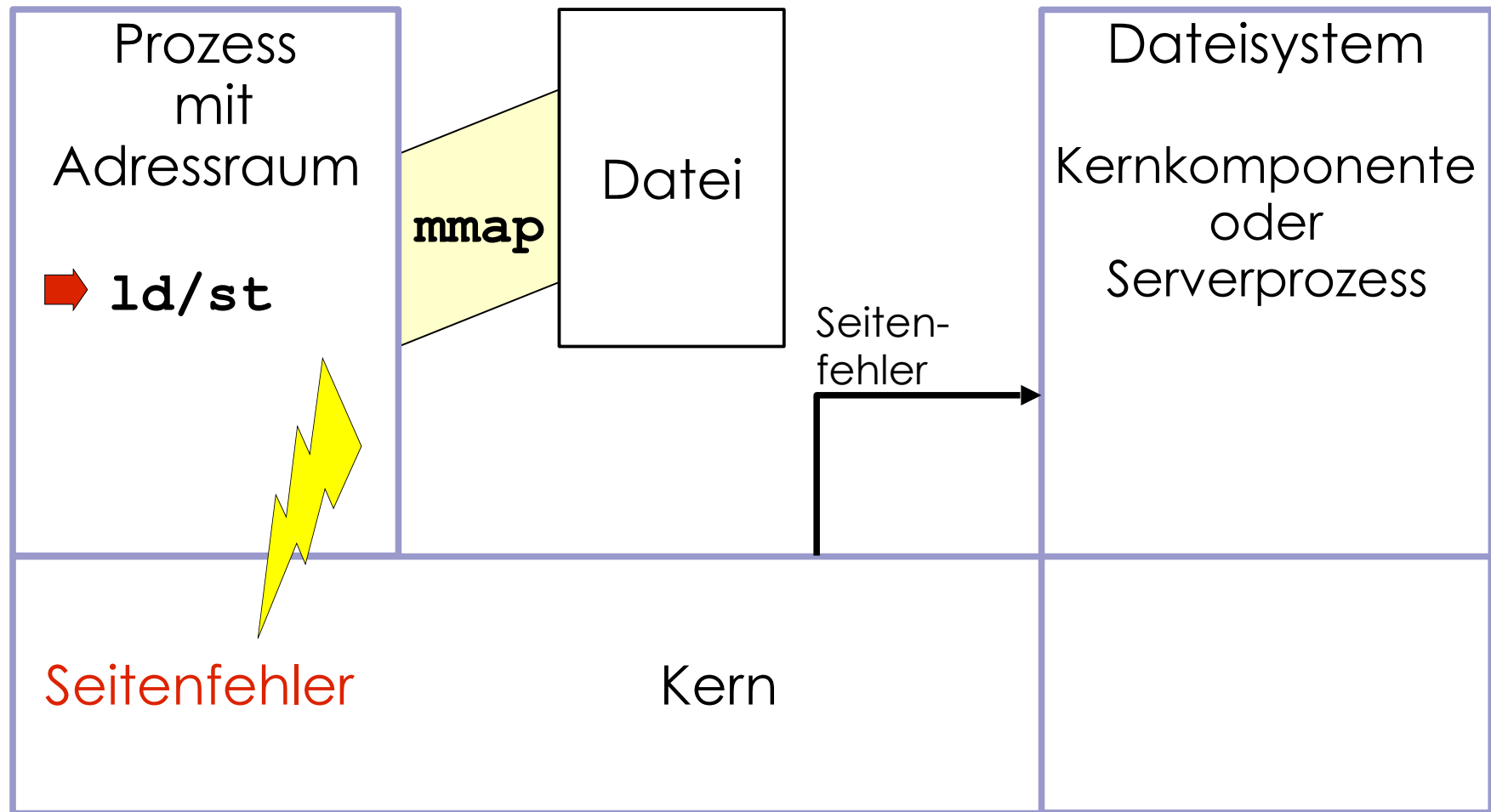
Zugriff mittels Einblendung



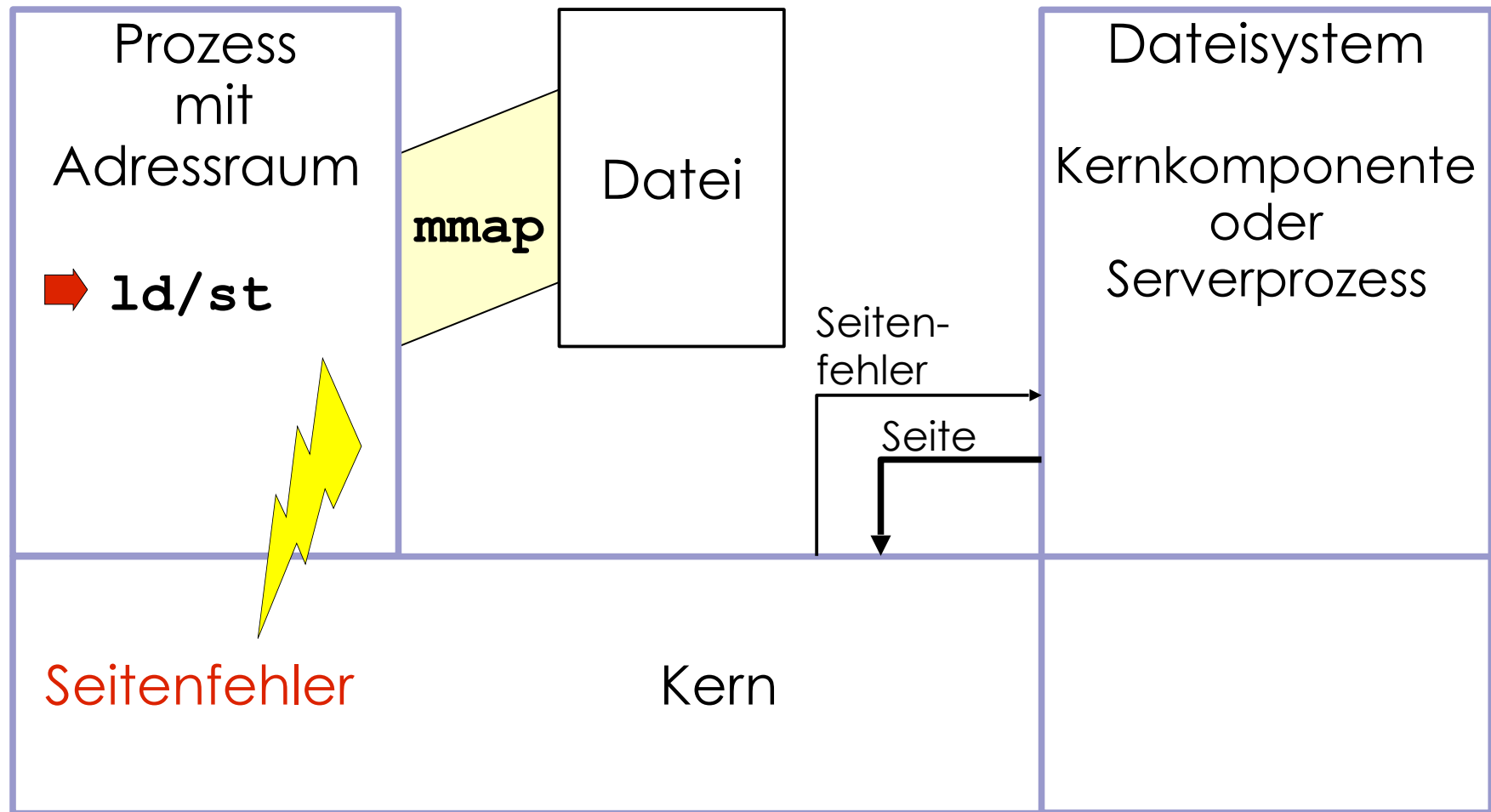
Zugriff mittels Einblendung



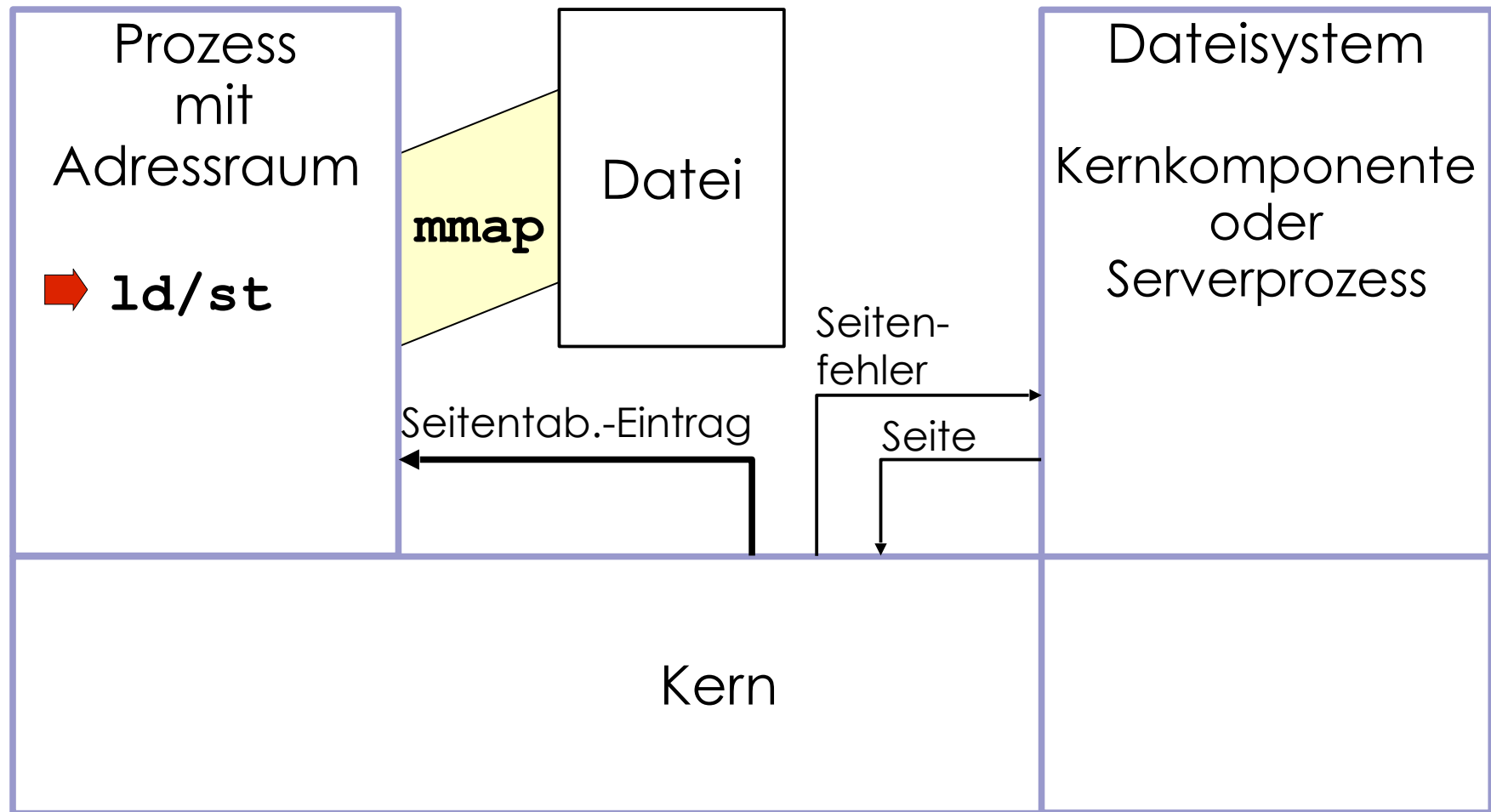
Zugriff mittels Einblendung



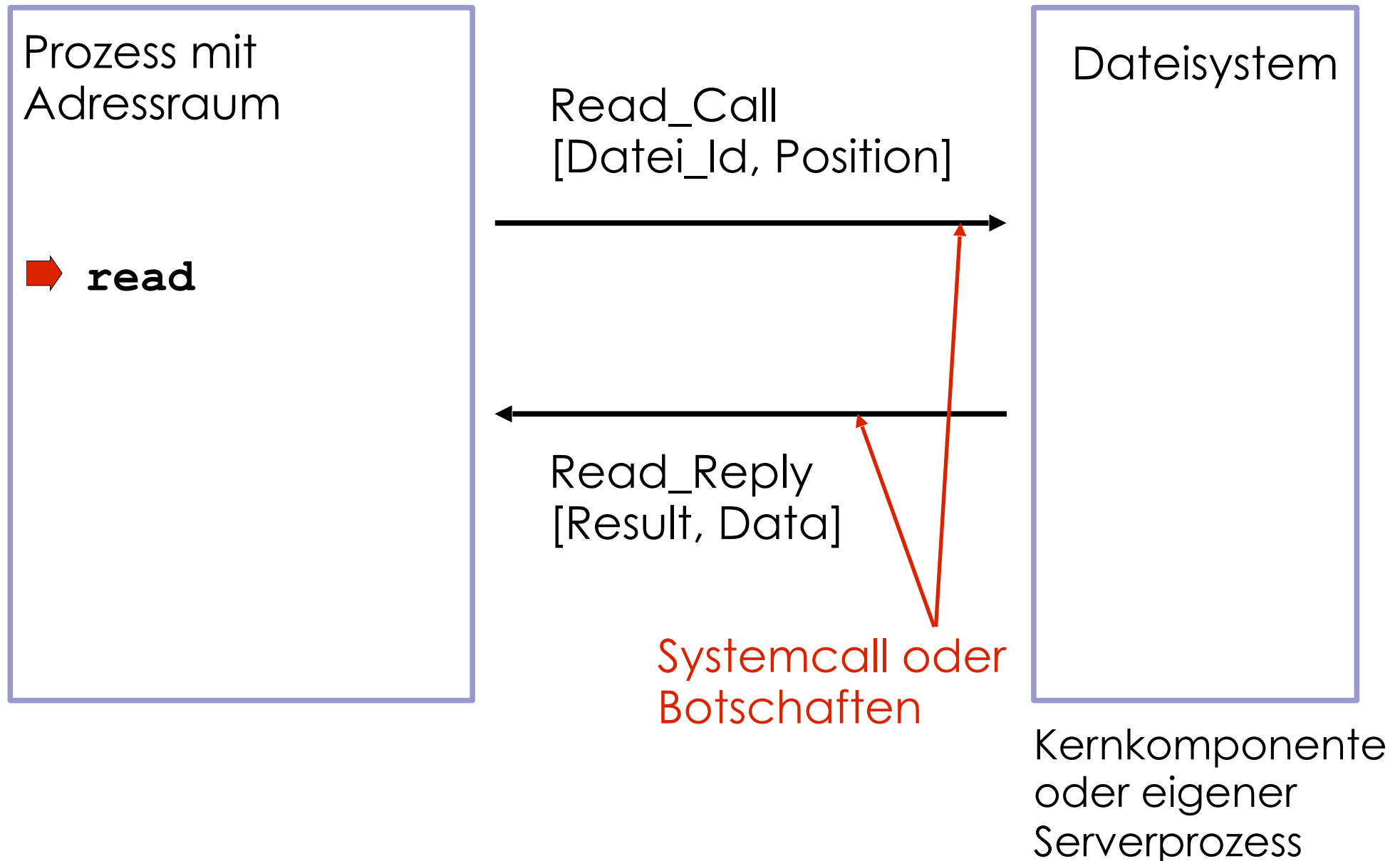
Zugriff mittels Einblendung



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

Dateideskriptoren

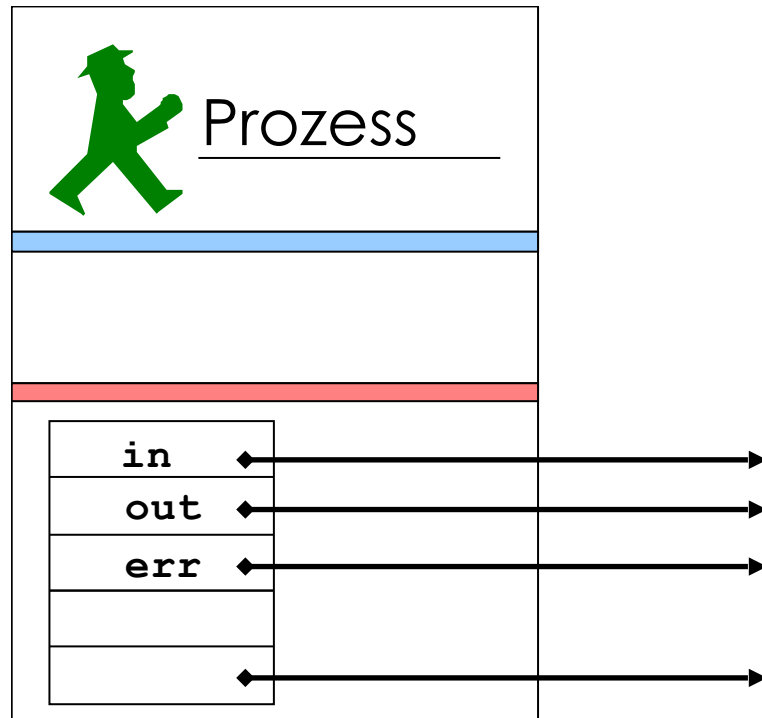
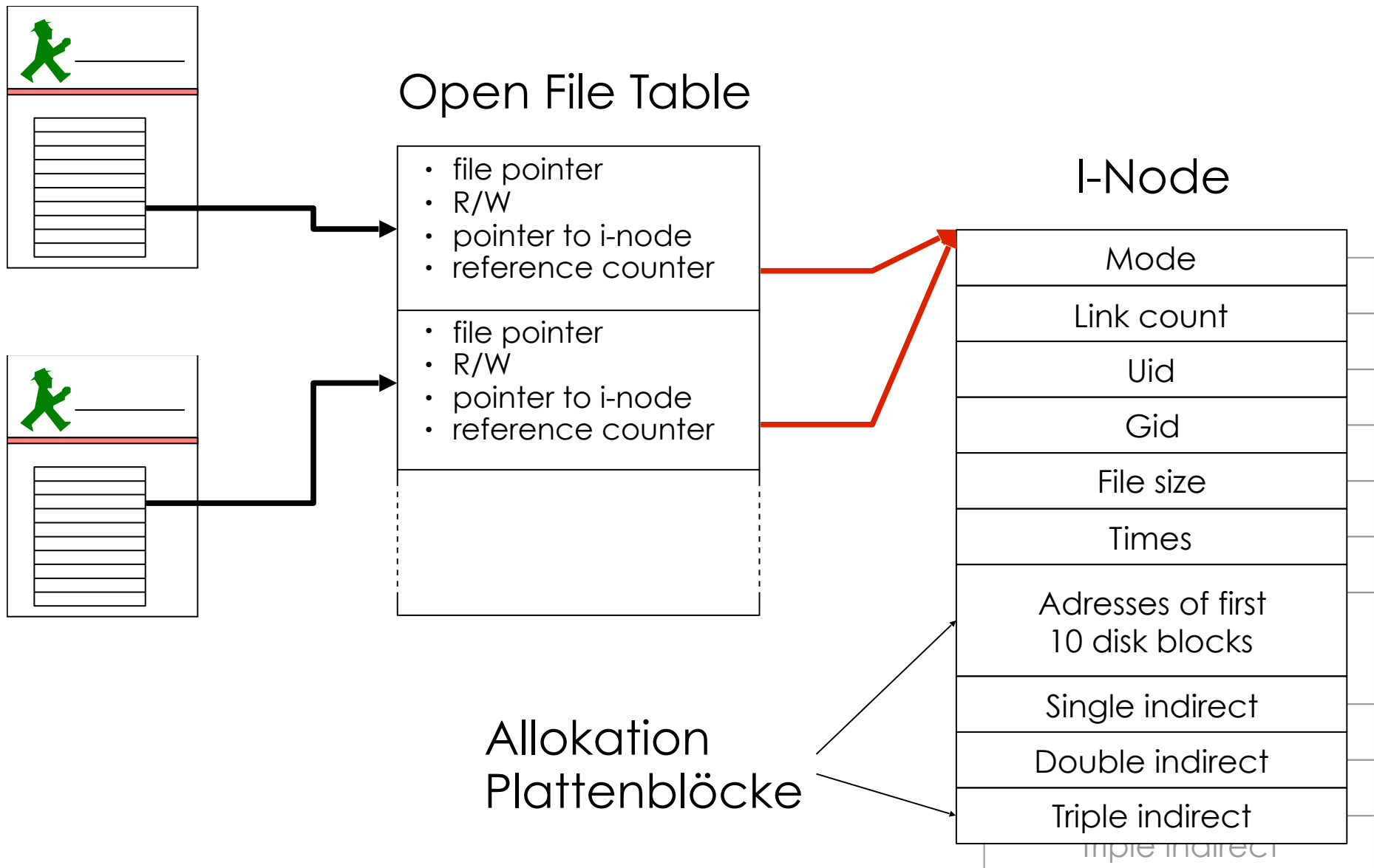


Tabelle geöffneter Dateien

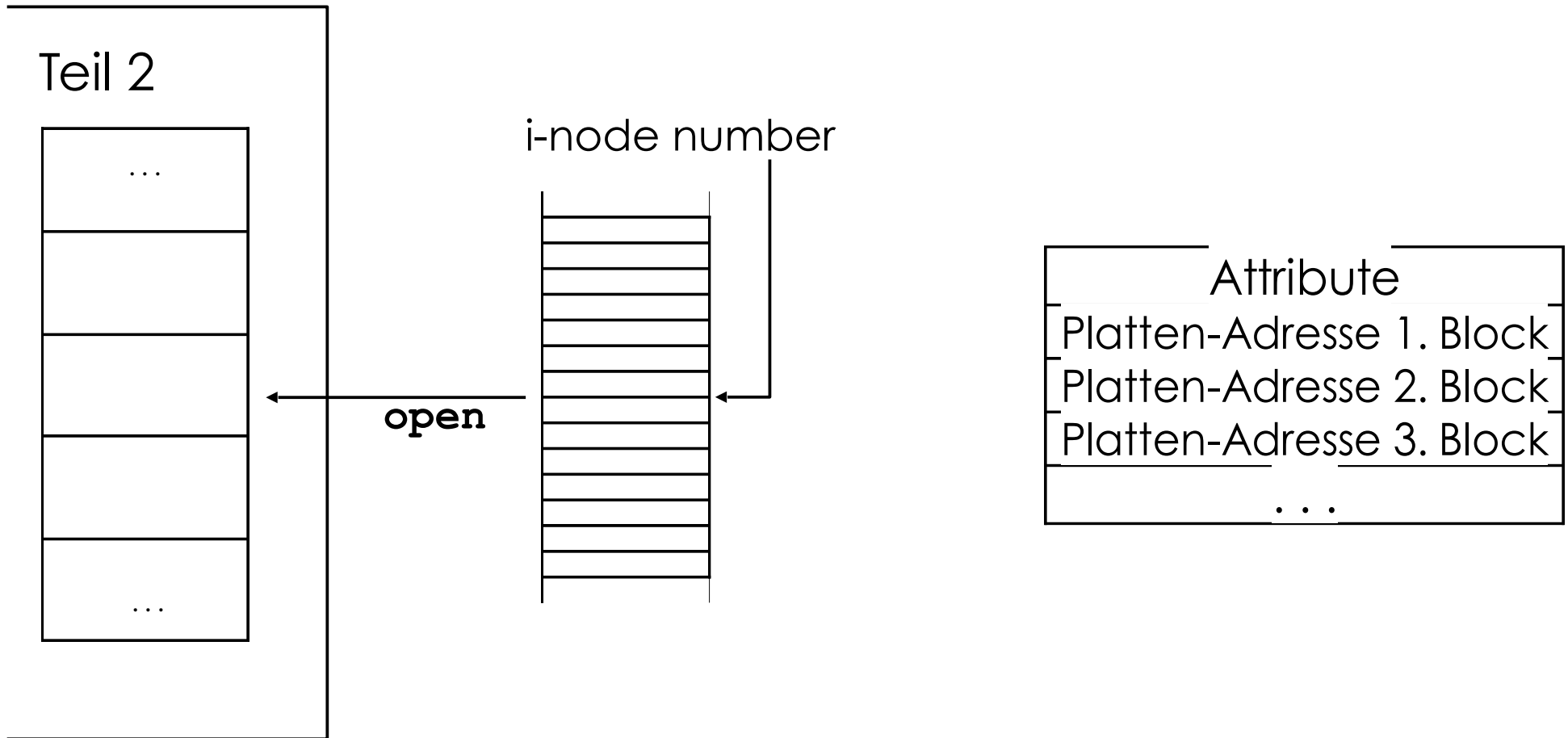


i-node (I-Knoten)

**Tabelle offener
Dateien**

**Dateitabelle
auf Platte**

i-node



Pufferverwaltung und Ablauf read/write

Block-Kachel-Tabelle

(„Buffer Cache“)

- Struktur im Kernadressraum
- Zuordnung:
Plattenadresse \leftrightarrow Puffer
- Puffer allokkieren/laden
(resident im Hauptspeicher)

Ablauf (read/write)

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

Ablauf write

```
write(fd, address, length);
```

Nutzer
Adressraum



Datei



filepointer

nächste Folie: vergrößerter Ausschnitt

Ablauf write

```
write(fd, address, length);
```

Nutzer
Adressraum



Datei



filepointer

Ablauf write: Puffern in Buffercache

`write(fd, address, length);`

Nutzer
Adressraum



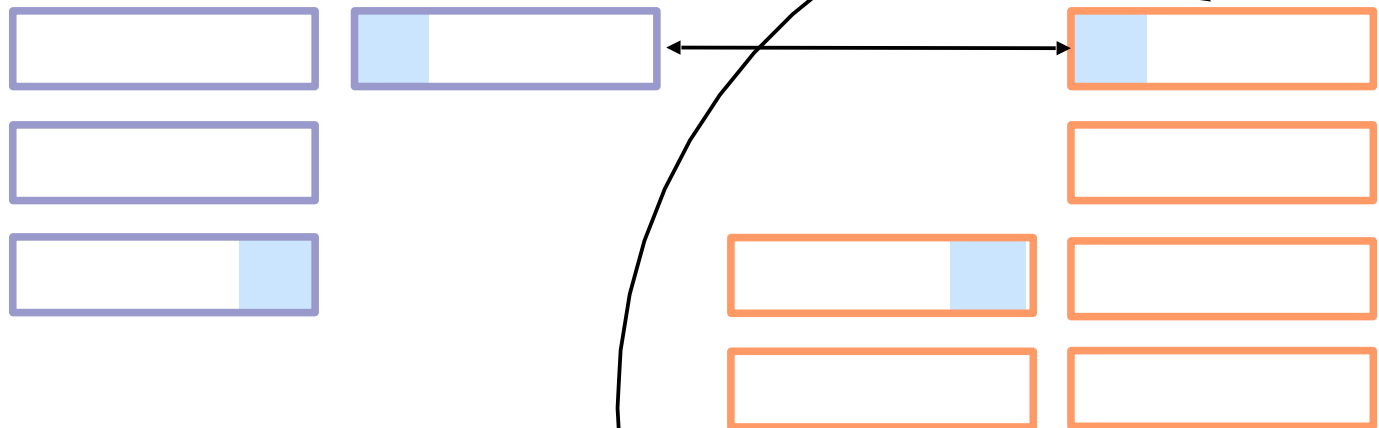
Datei



filepointer

Platte

Puffer
(resident im HS)



..... Datei ↔ Platte

———— Puffer ↔ Platte

Ablauf write

`write(fd, address, length);`

Nutzer
Adressraum



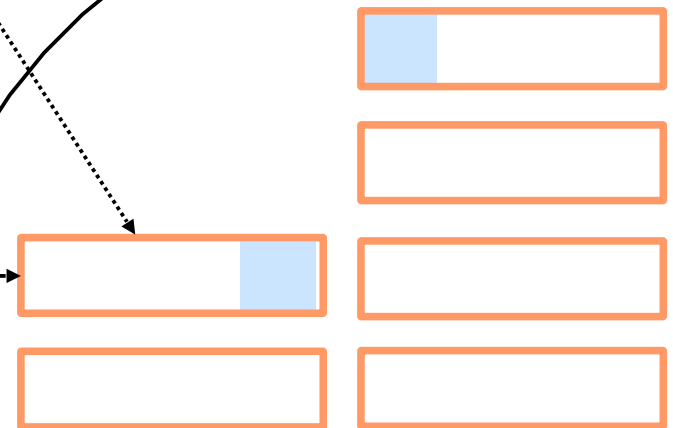
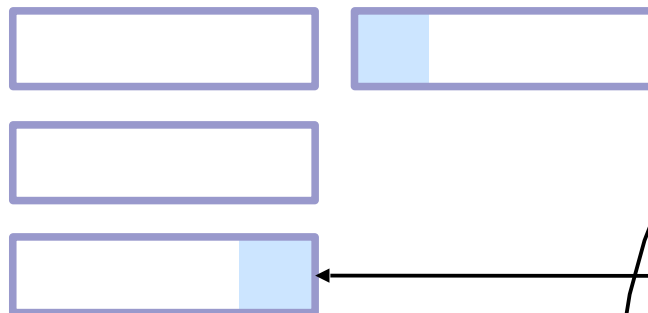
Datei



filepointer

Platte

Puffer
(resident im HS)

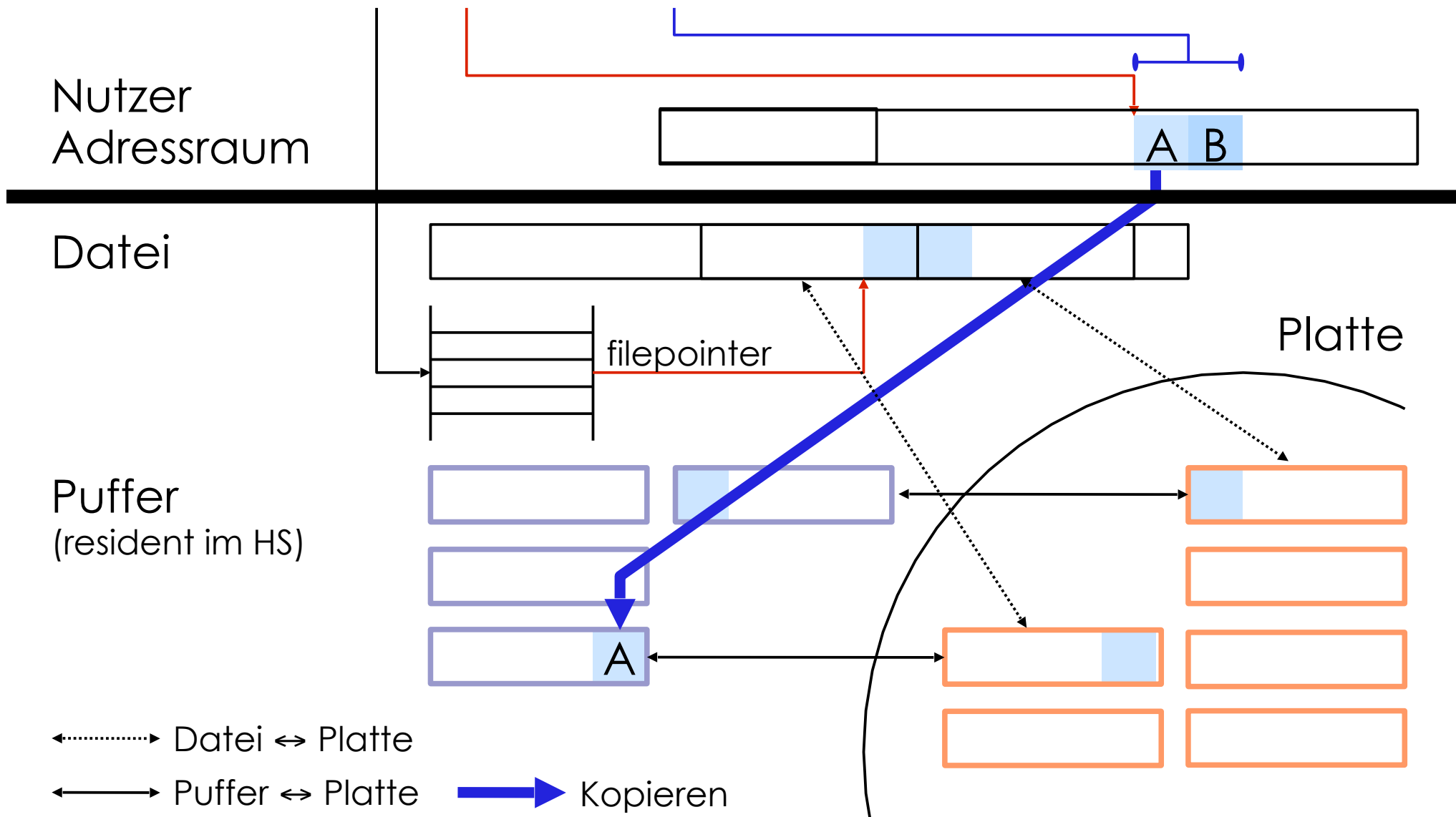


..... Datei ↔ Platte

↔ Puffer ↔ Platte

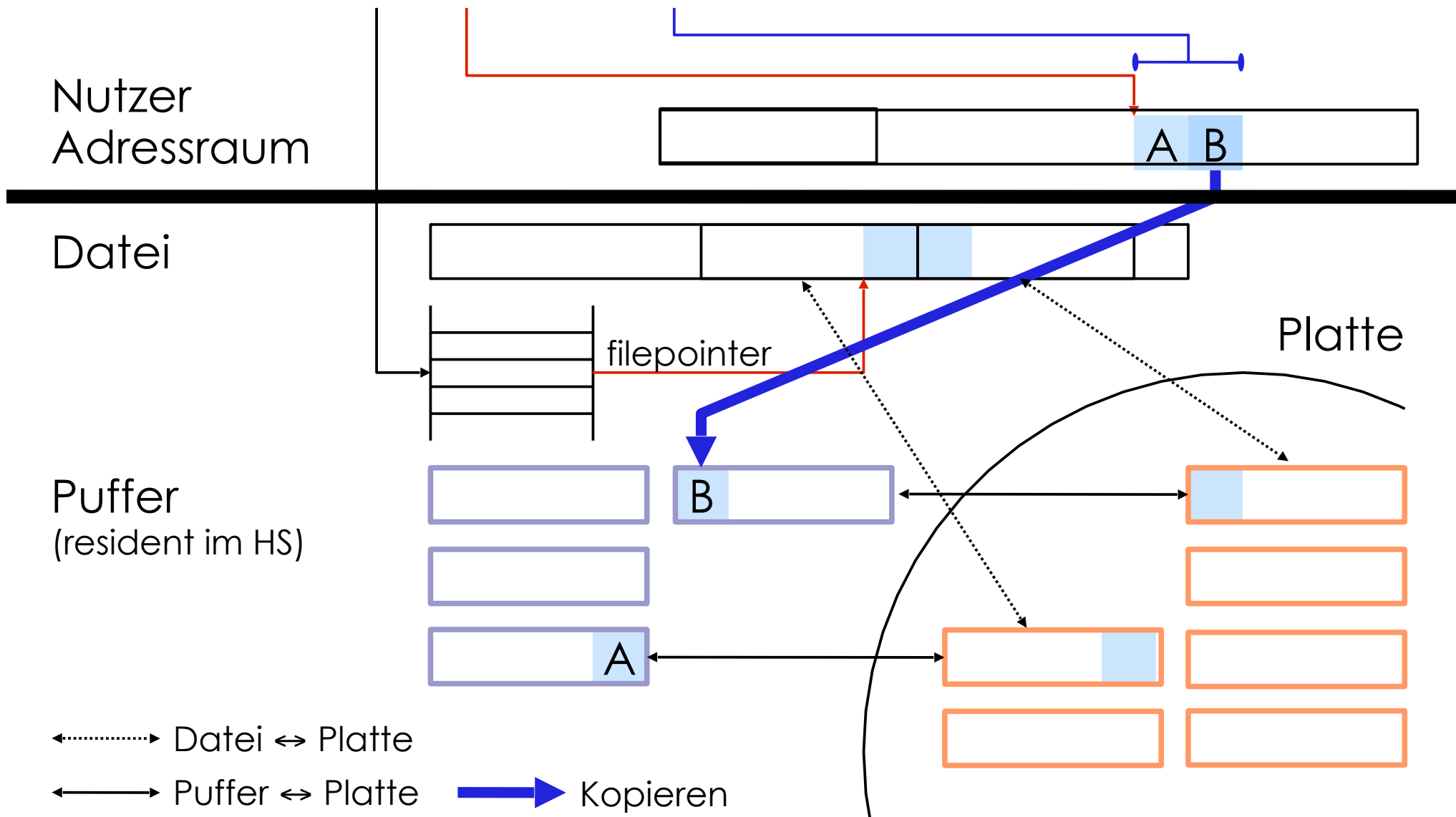
Ablauf write: Kopieren in Puffer

`write(fd, address, length);`



Ablauf write: Kopieren in Puffer

`write(fd, address, length);`



Ablauf write: Asynchrones Schreiben auf Platte

`write(fd, address, length);`

Nutzer
Adressraum



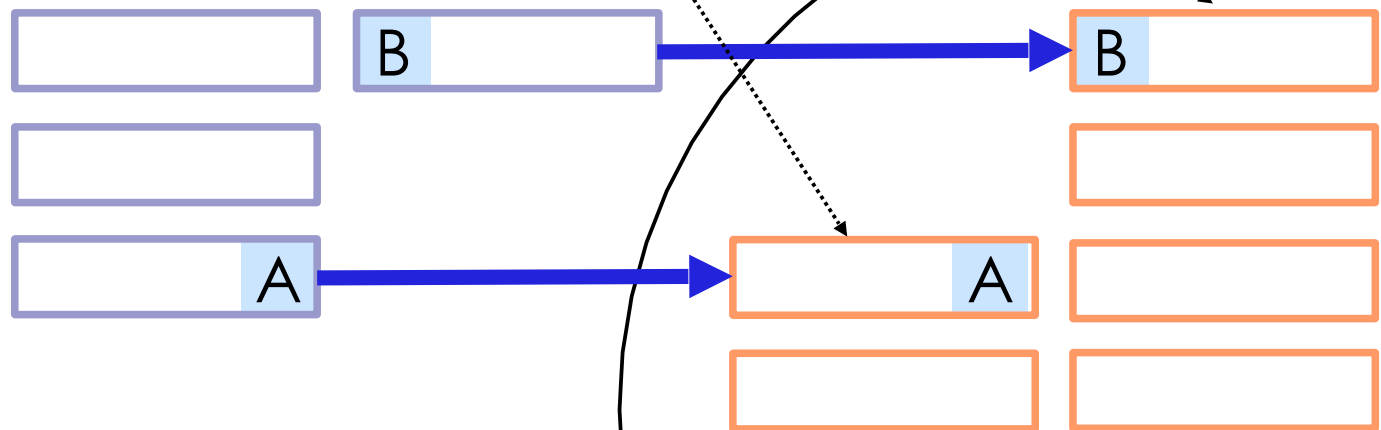
Datei



filepointer

Platte

Puffer
(resident im HS)



..... Datei ↔ Platte

↔ Puffer ↔ Platte ➡ Kopieren

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);
```

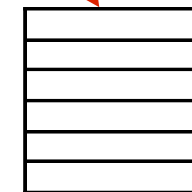
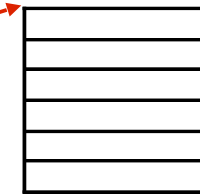
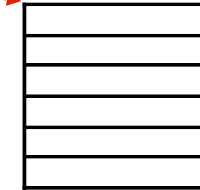
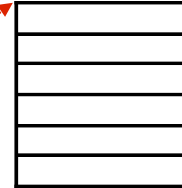
```
mmap (Kontodatei, ...);  
  
KontoAdr = Find_Konto;  
  
KontoAdr->Stand += Betrag;
```

Inkonsistente Datenstrukturen

i-node



Triple
indirect block



Adressen der Datenblöcke

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