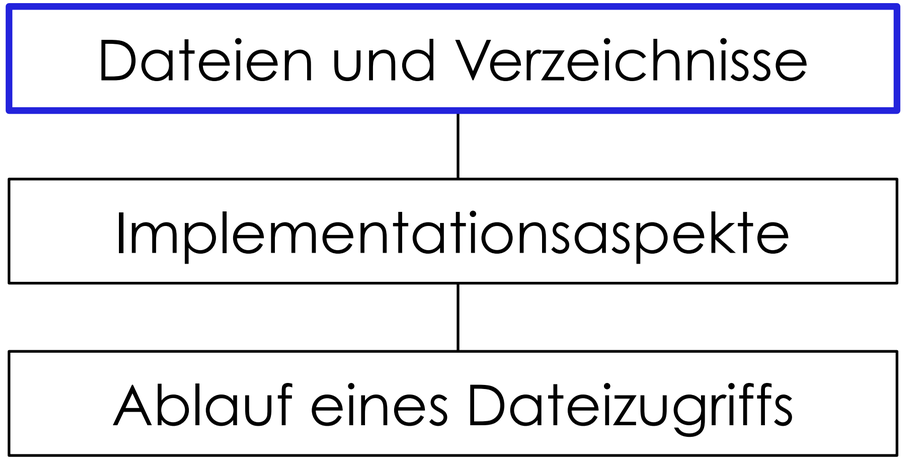
TECHNISCHE UNIVERSITÄT DRESDEN

Faculty of Computer Science Institute of Systems Architecture, Operating Systems Group

DATEISYSTEME

MICHAEL ROITZSCH

Wegweiser


Dateien und Verzeichnisse
Implementationsaspekte
Ablauf eines Dateizugriffs

Gegenstand

Aufgaben eines Dateisystems

- Persistenz von Daten: Daten sollen Prozess- und Systembeendigung überstehen (auch Abstürze)
- Verwaltung von Speicherobjekten auf Speichermedium
- Namensdienst für Datenzugriff: Abbildung von Dateinamen auf Speicherobjekte
- Schutz (ACLs)

Zentrale Herausforderungen

- Hardwareeigenschaften des Speichermediums
- Robustheit (nächste 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

kodiert in Bytes, repräsentiert durch einen „Bezeichner“

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
Seek	Handle, Position
Lesen/Setzen von Attributen	
Umbenennen	

Zugriffsstrukturen

Flach

- Byte-Array, Lesen/Schreiben an beliebigen Stellen
- Interpretation von Strukturen durch Anwendungen

Satzstrukturierte Daten: Records

- Länge der Records konstant oder variabel
- ein Datensatz (record) pro Zugriff (Lesen/Schreiben)
  - sequenziell
  - direkter Zugriff über Schlüssel
  - kombiniert: indexsequenziell
- heute: Key-Value-Store

Datei-Typen

Unix

zahlreiche Objekte werden als Dateien behandelt

Beispiele

- normale Dateien (regular file)
- Verzeichnisse (directories)
- E/A-Geräte (special files)
- Datenströme (pipes)
- Endpunkte für Dienste (sockets)
- symbolische Links

Datei-Attribute

Zugriffsschutz

- Besitzer, Rechte des Besitzers und anderer

Zeiten

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

Organisatorisches

- aktuelle/maximale Größe
- Verwaltungsinformationen

Dateinamen

Aufgaben

- Benennen: symbolisch – Mensch als Benutzer
- Identifizieren: Programme als Benutzer
- (schnelles) Lokalisieren

Benutzerdefinierte Dateinamen

- innerhalb bestimmter Regeln, je nach System
- Längenbeschränkung
- Hierarchiebildung: Trennzeichen für Verzeichnisnamen
- Groß-/Kleinschreibung: signifikant oder nicht
- Konvention: Dateiendung

Verzeichnisse und Pfadnamen

Ziel

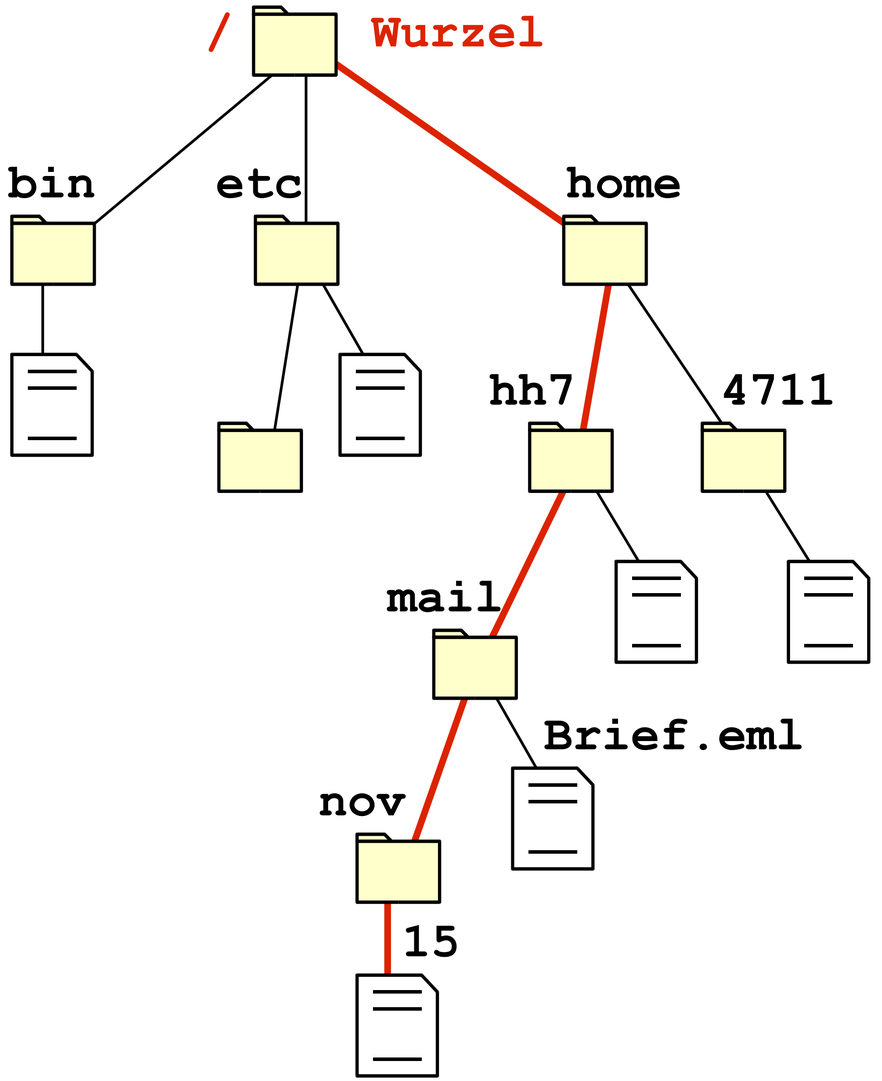
- systemweit eindeutige Dateinamen

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



Kontexte für Namen

Ausgangs-Kontexte durch Prozesse festgelegt

Root Directory

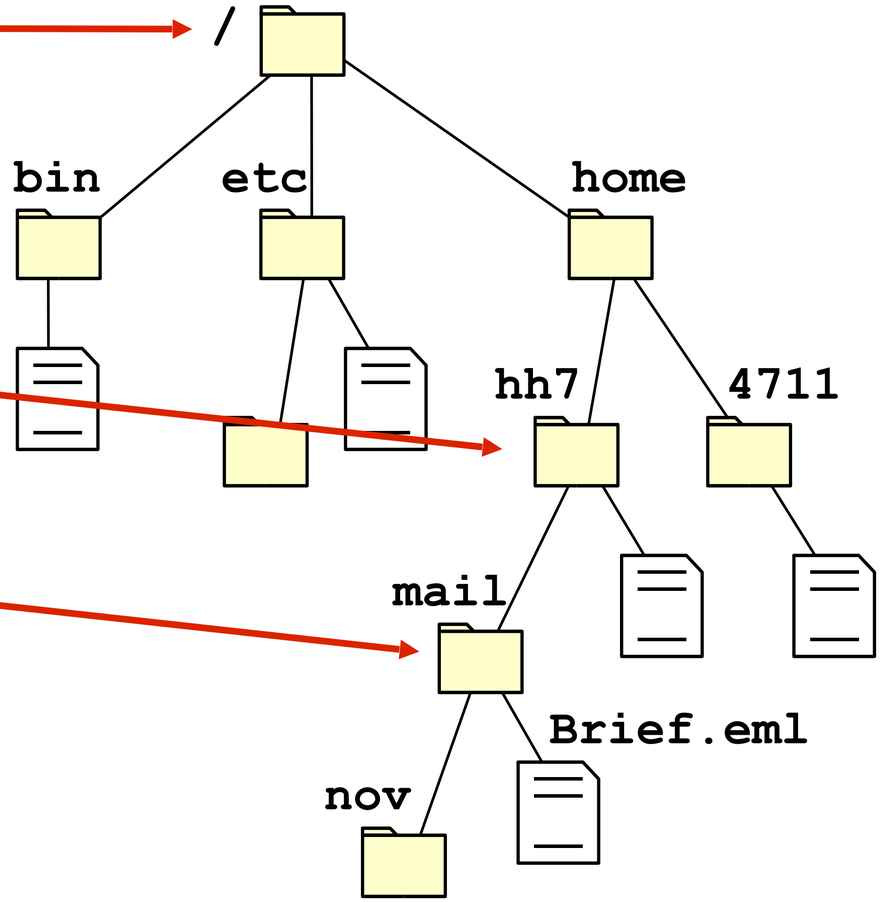
Kontext für absolute Namen,
durch **`chroot()`** änderbar

Home Directory

Startpunkt für Benutzer

Current Directory

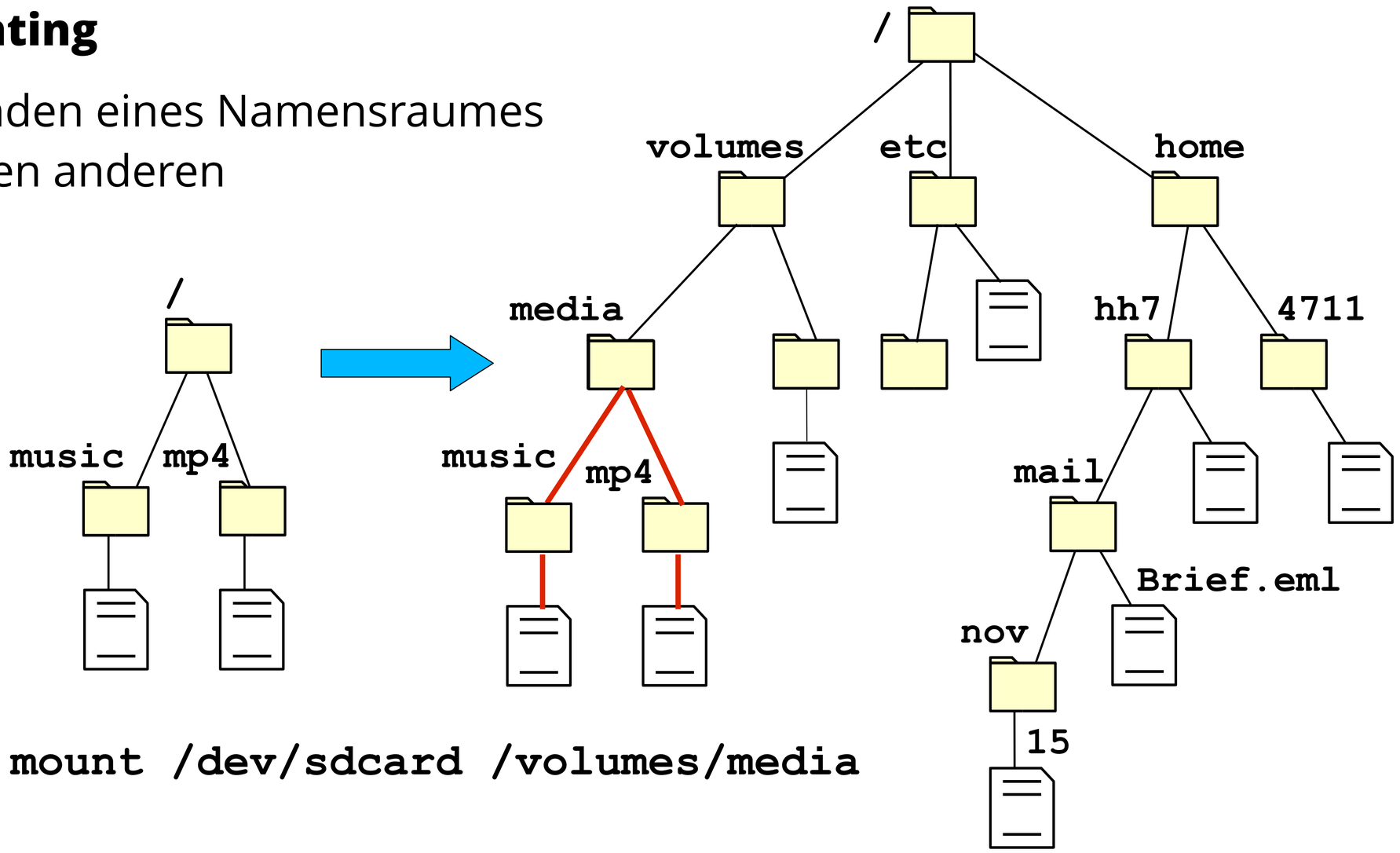
Kontext für relative Pfadnamen,
durch **`chdir()`** änderbar



Verbinden von Namensräumen: Mounting

Mounting

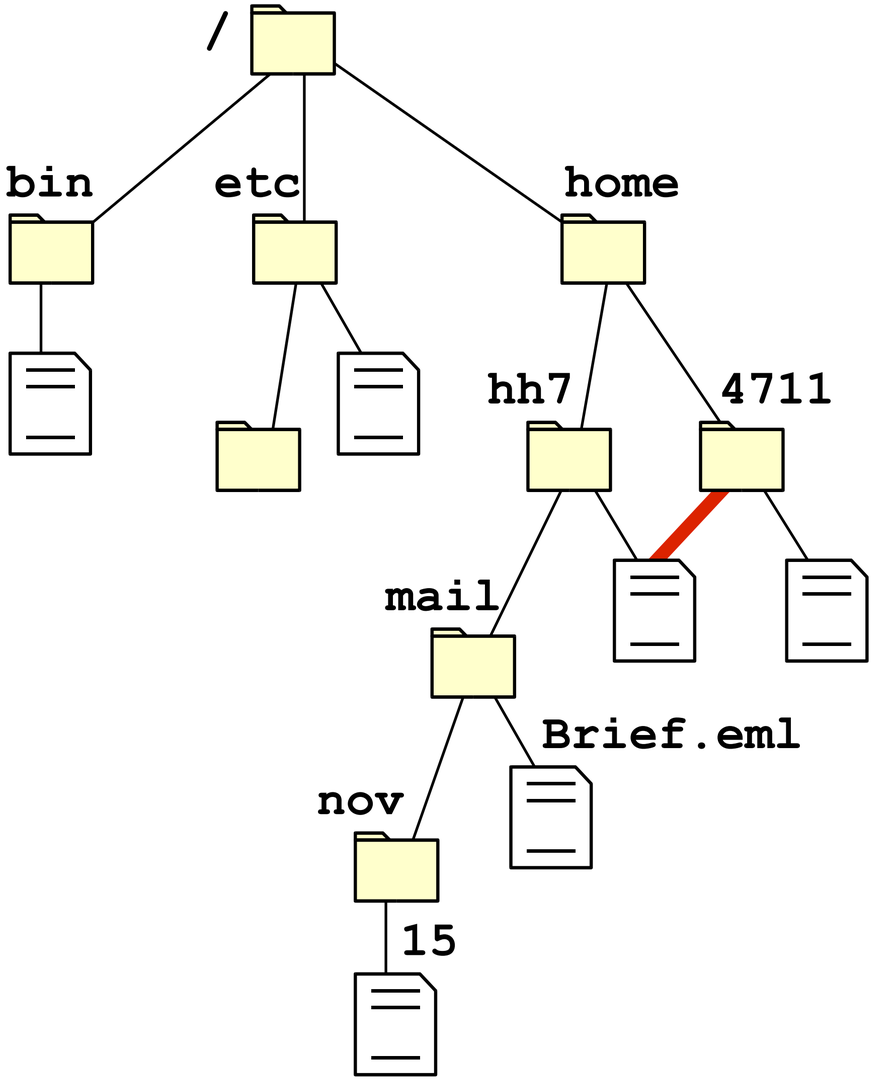
Einbinden eines Namensraumes
in einen anderen



Mehrere Namen für eine Datei: Hard Links

Harte Links / Hard Links

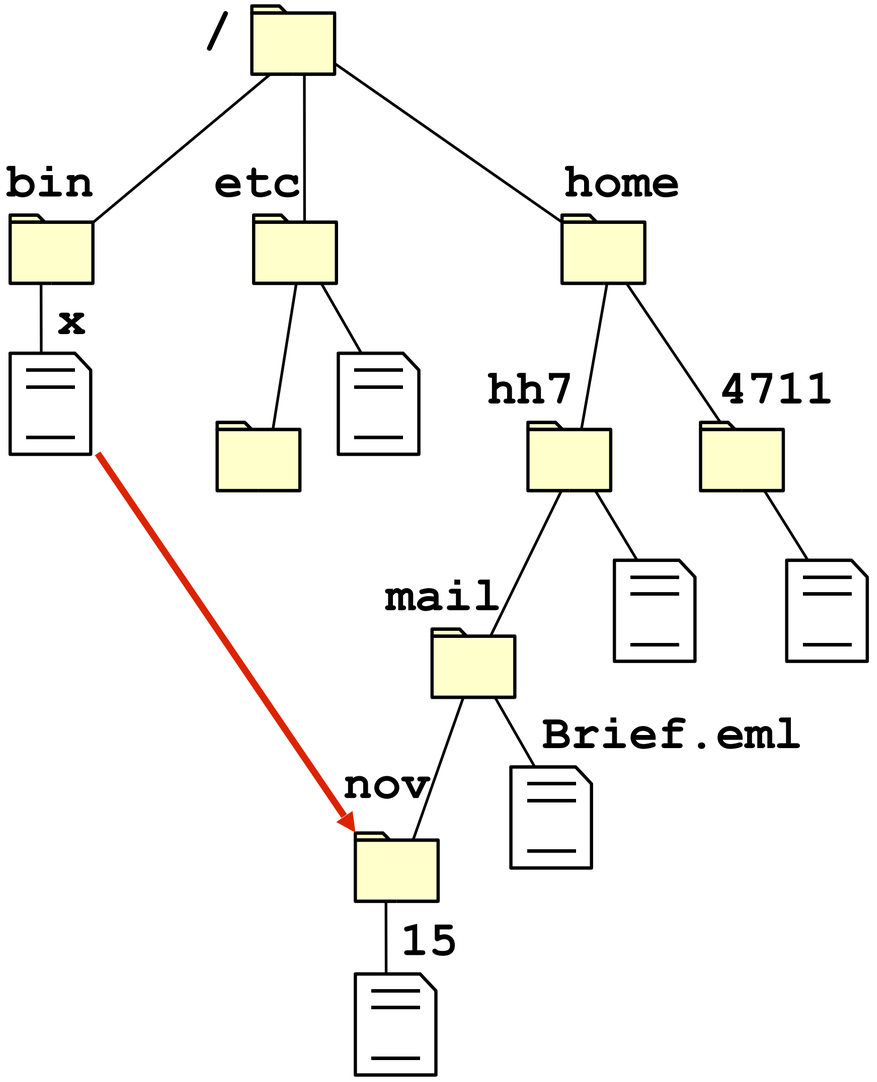
- mehrere Namen für dieselbe Datei
- potenziell (aber nicht notwendig) in verschiedenen Verzeichnissen
- Limitation:
  nur für Dateien im selben Dateisystem möglich



Verweise auf Dateien: Symbolische Links

Symbolische Links

- Verweise als spezieller Dateityp
- Inhalt: Pfadname
- auch zwischen verschiedenen Dateisystemen möglich
- Achtung: Zyklenbildung
- Limitation:
  können auf ungültiges Ziel verweisen



Beispiel: Backup

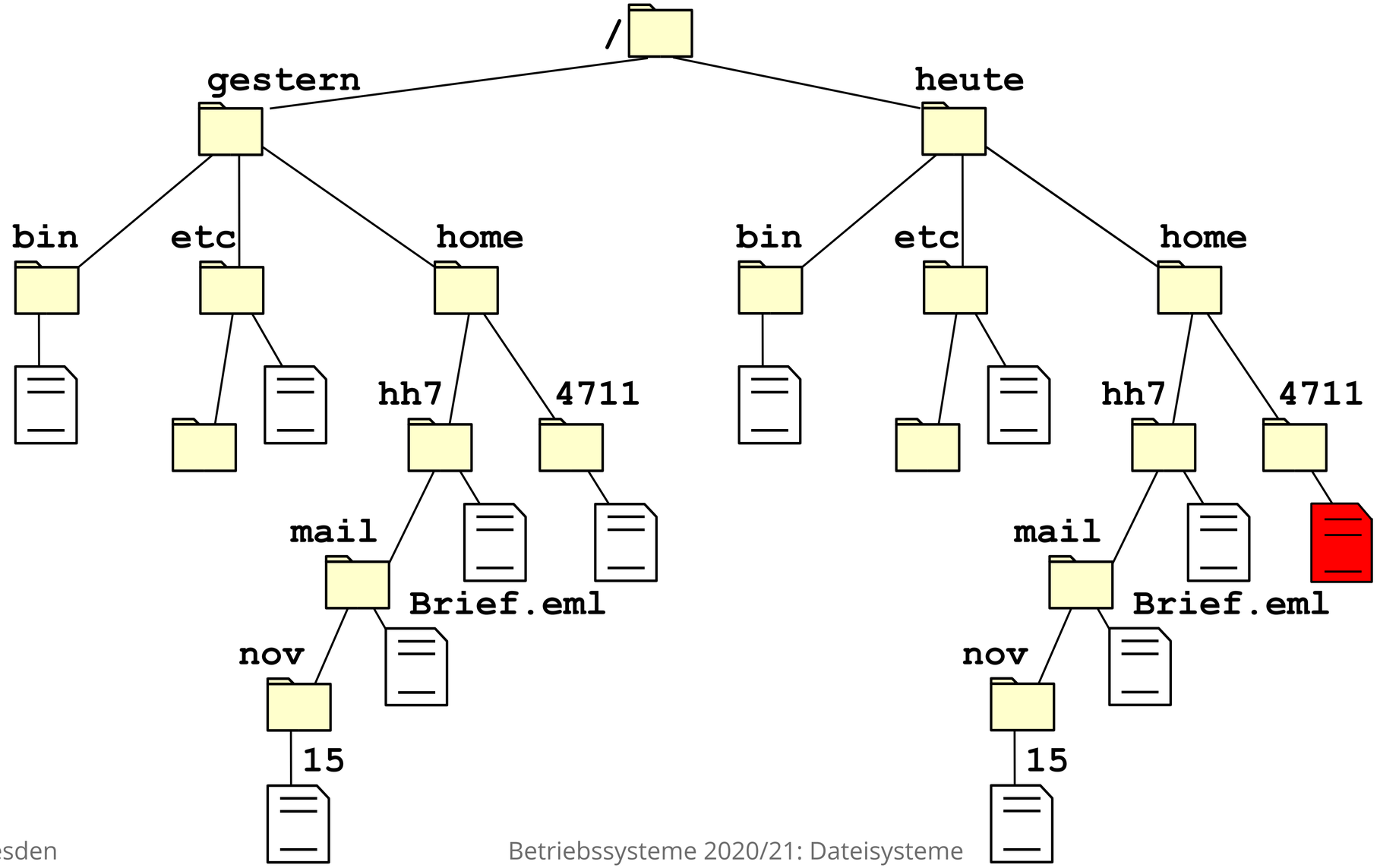
Ziele

- vollständige Kopie des Dateisystems in regelmäßigen Abständen (zum Beispiel stündlich)
- platzsparend: nur das Nötige kopieren
- Ausdünnen: einfaches Löschen alter Sicherungen

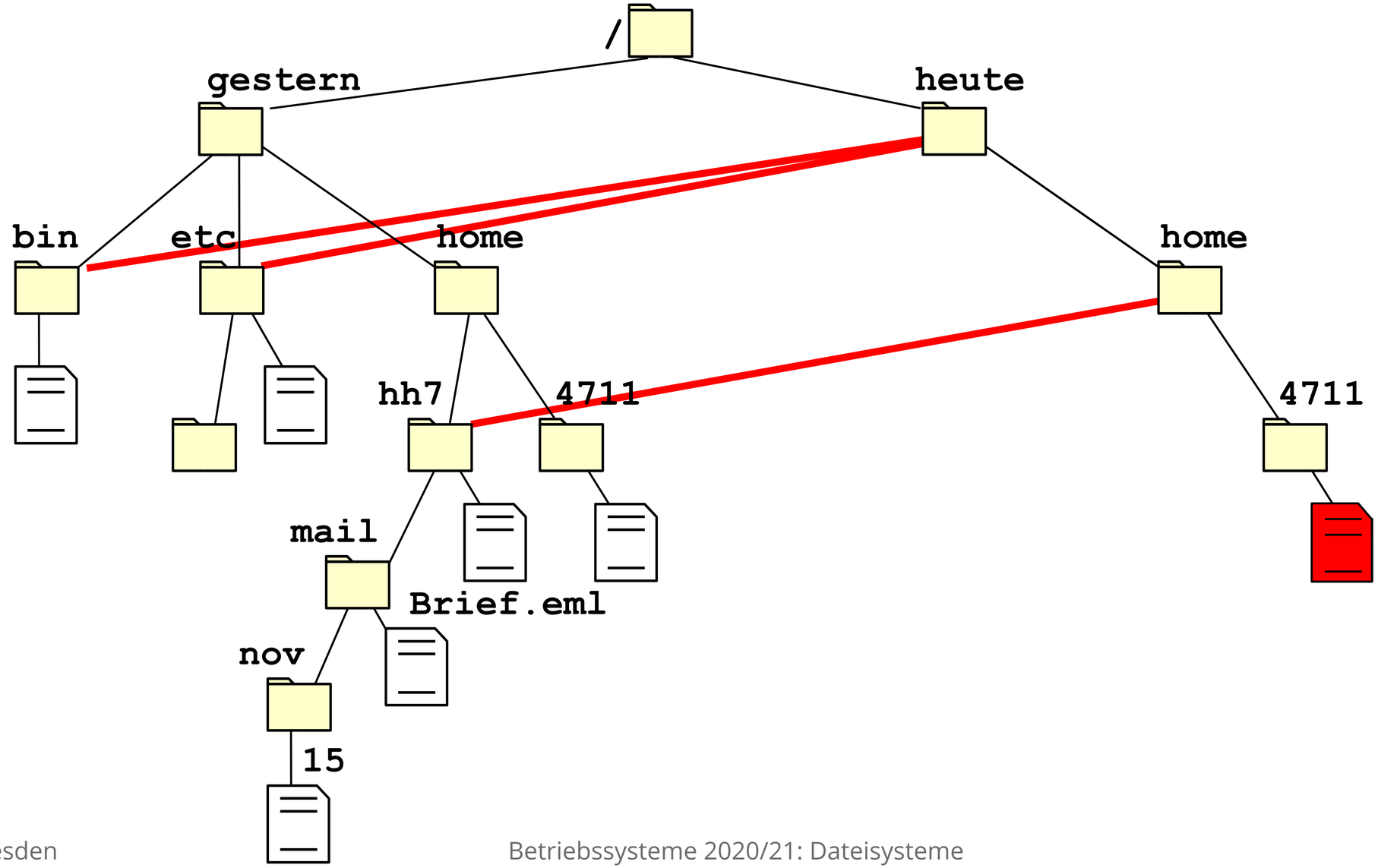
Lösung

- harte Links auf alles, was sich nicht ändert
- harte Links auf ganze Teilbäume, in denen sich nichts geändert hat (nicht in allen Dateisystemen erlaubt)
- bei symbolischen Links: Löschen alter Backups schwierig
- orthogonal: Erkennen der Änderungen

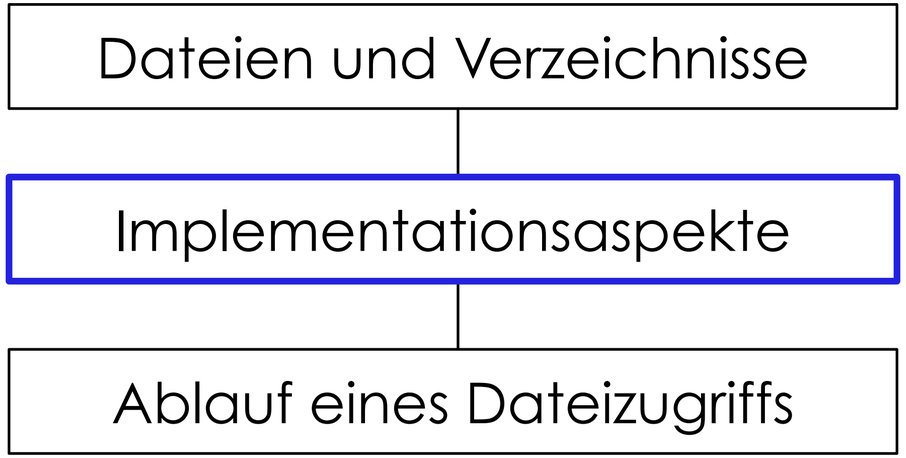
Beispiel: Backup



Beispiel: Backup

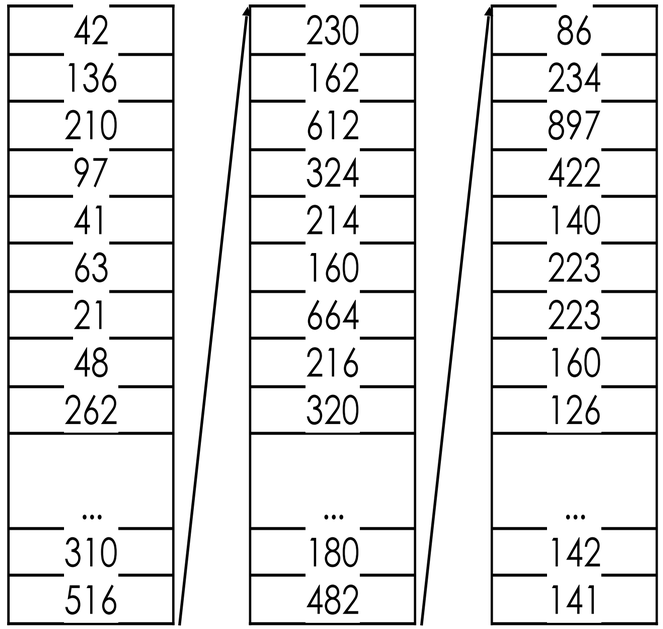


Wegweiser

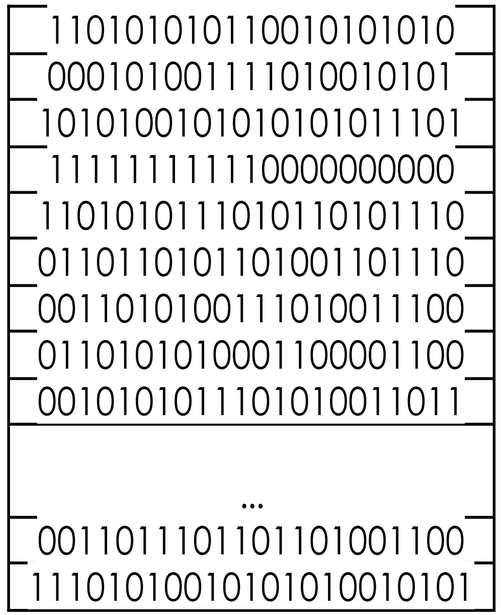

Dateien und Verzeichnisse
Implementationsaspekte
Ablauf eines Dateizugriffs

Verwaltung des freien Plattenspeichers

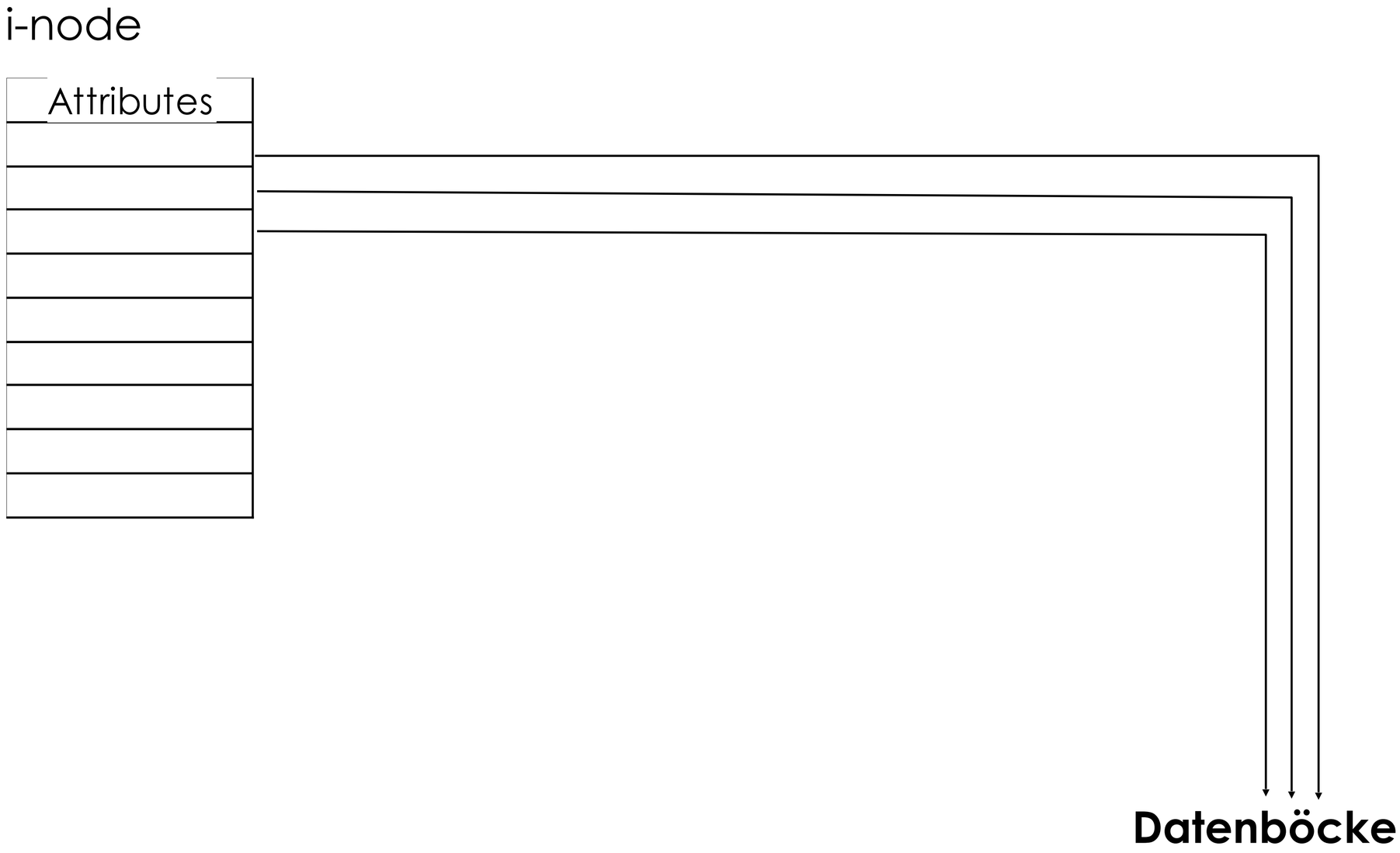
Verwendung von Listen



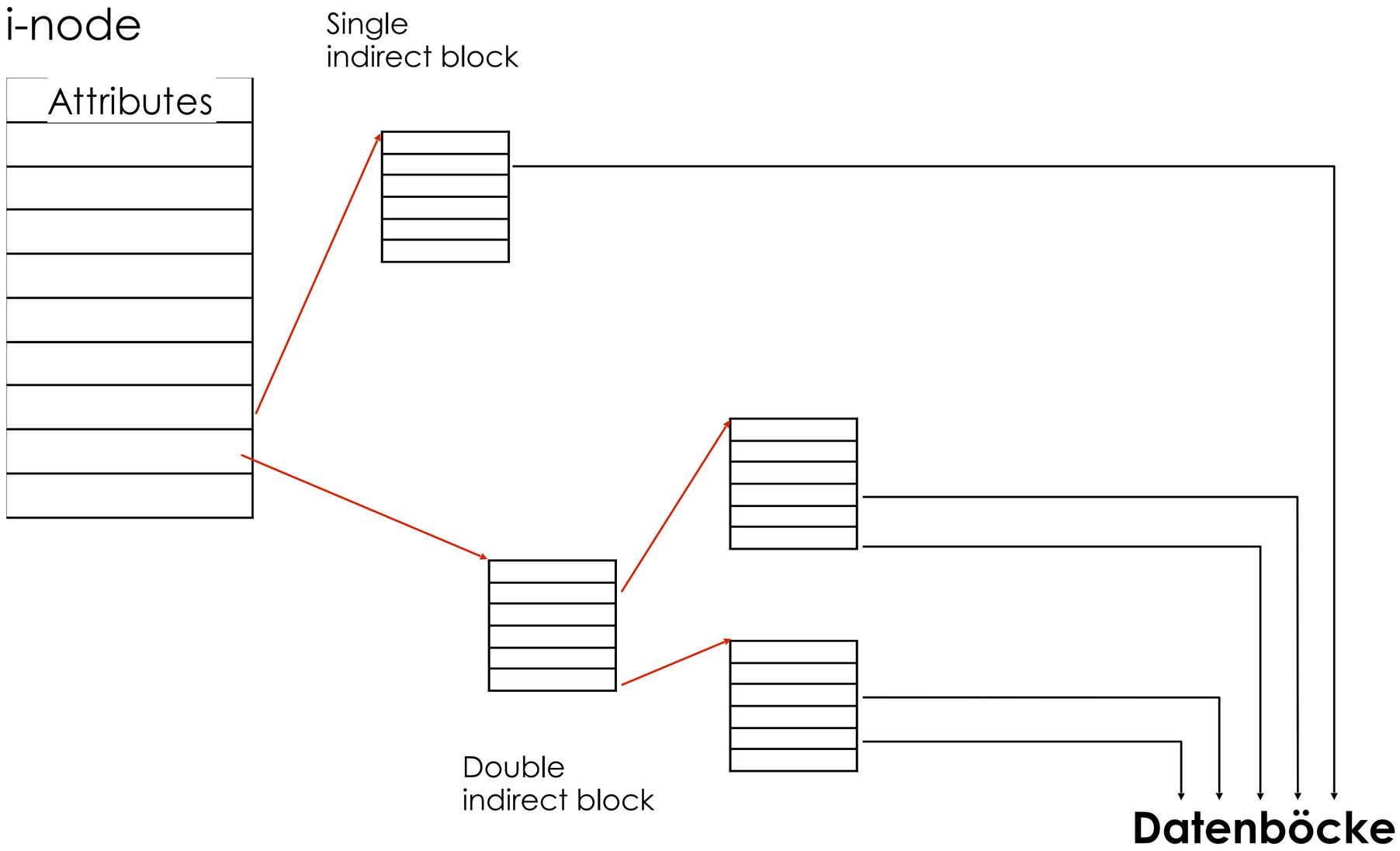
Verwendung von Bitmaps



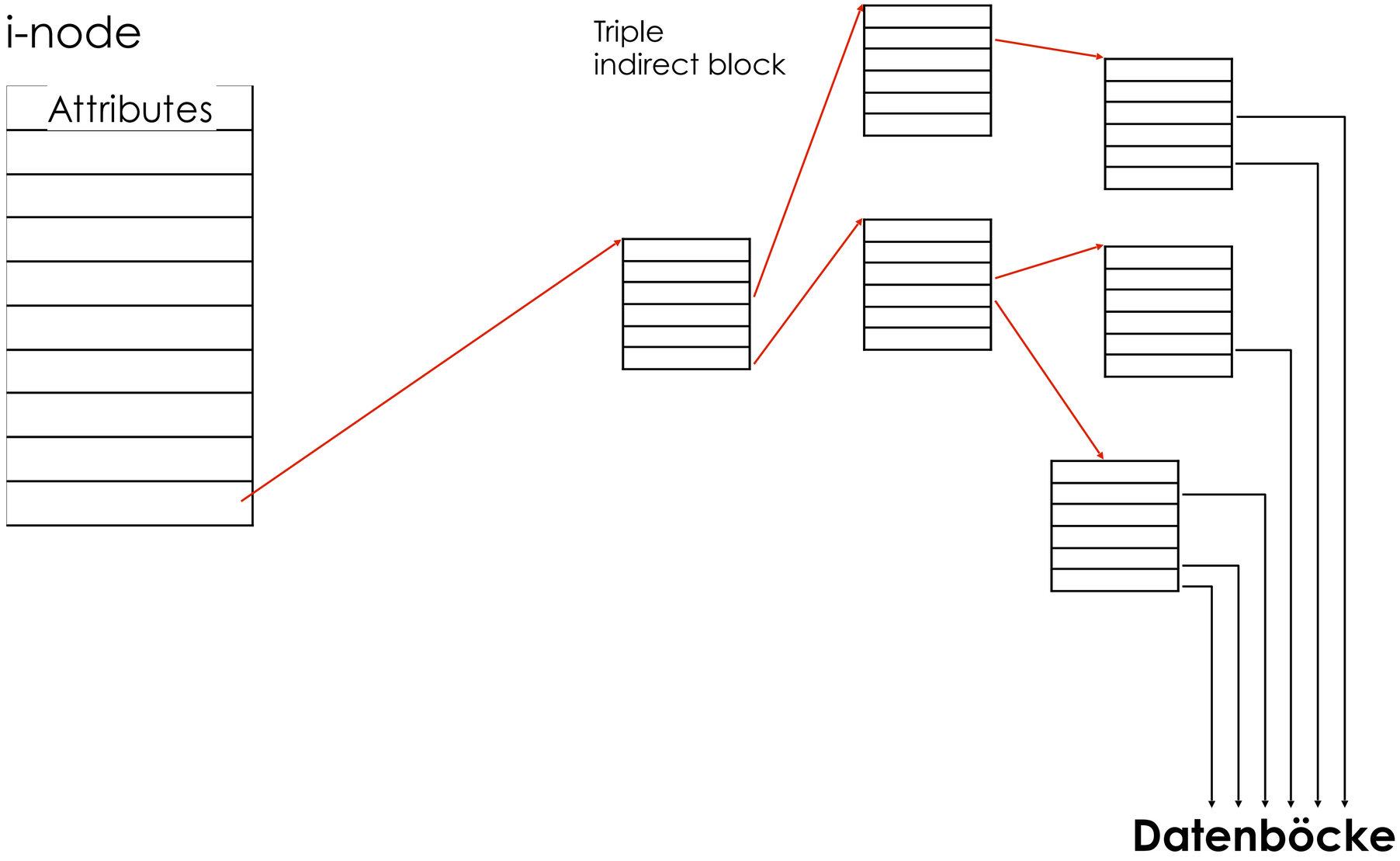
I-nodes (I-Knoten)



I-nodes (I-Knoten)



I-nodes (I-Knoten)



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

Dateideskriptoren

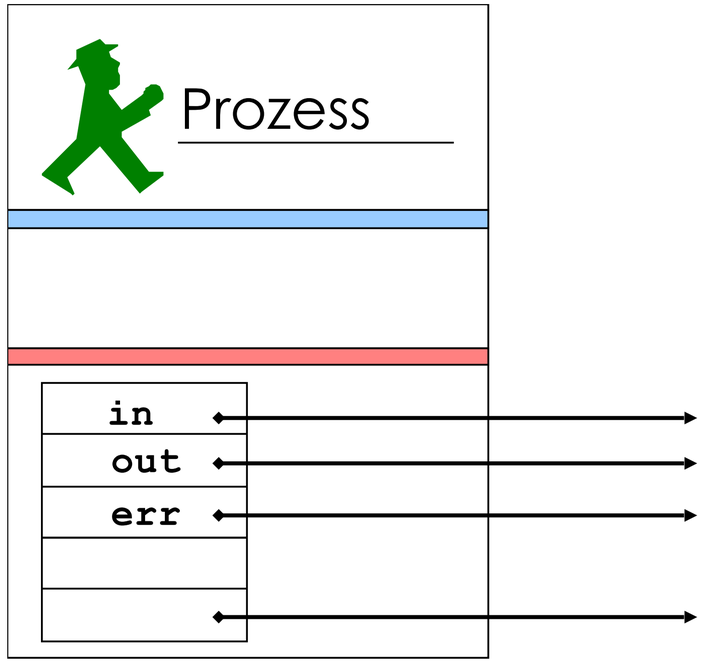
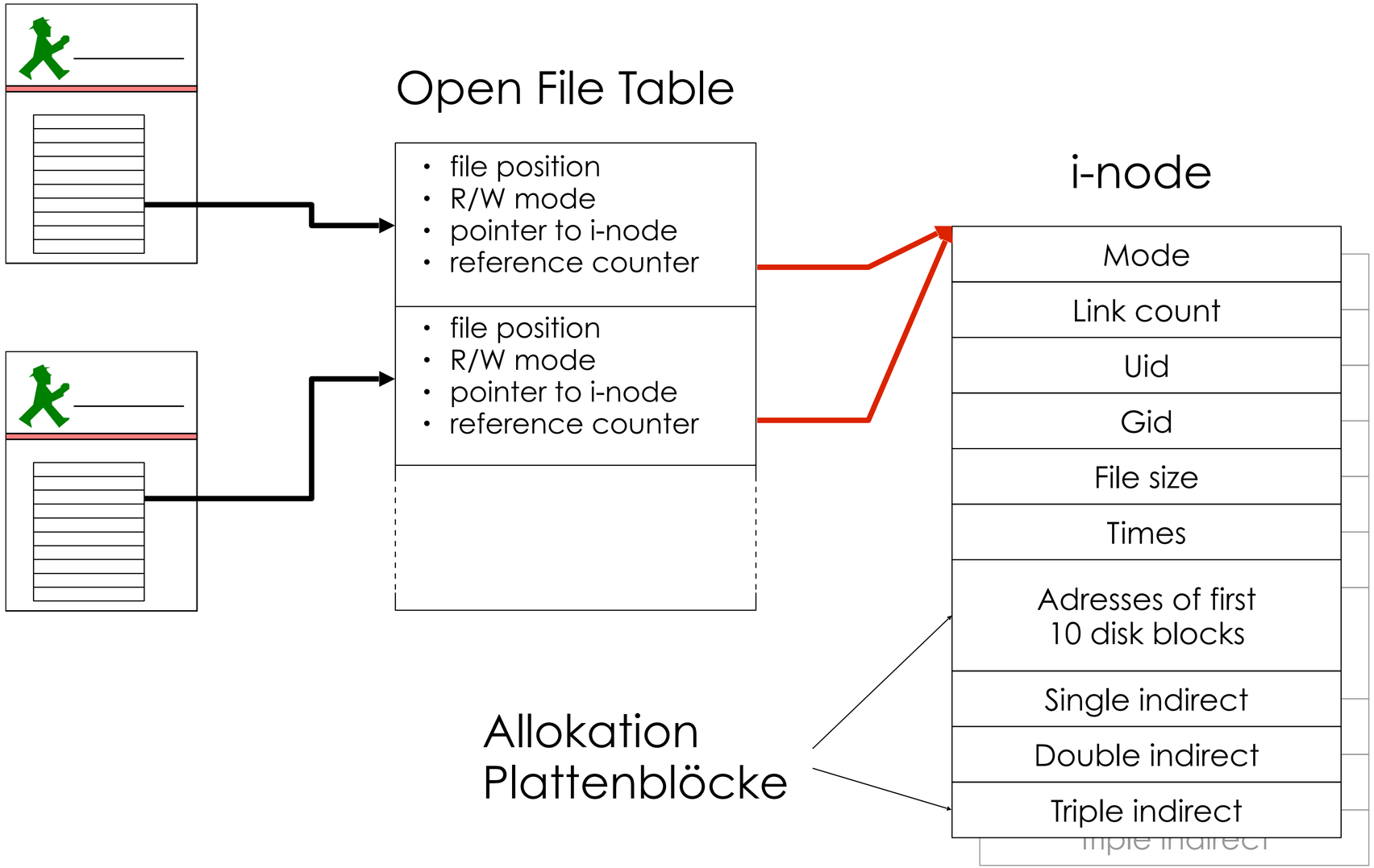


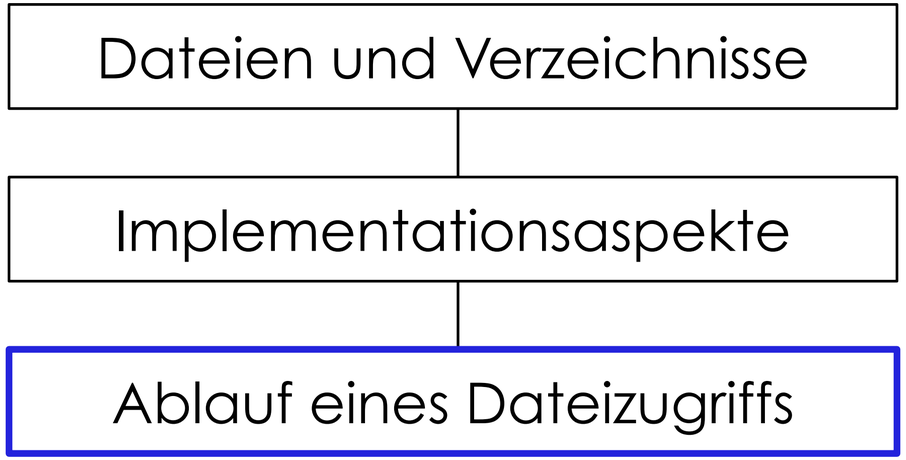
Tabelle geöffneter Dateien



Beteiligte Datenstrukturen

- Prozess: aktuelles Verzeichnis, Dateideskriptoren
- Verzeichnisse: zum Auffinden der i-nodes zu Dateinamen
- Tabelle der geöffneten Dateien, enthält pro Datei:
  - gegenwärtige Schreib-/Leseposition, ...
  - i-node-Nummer
- Dateiverwaltung auf Platte: Datenstruktur der i-nodes
  - i-node-Nummer identifiziert Datei
- für alle (auch nicht geöffnete) Dateien: i-node
  - Allokation: Zuordnung von Plattenblöcken
  - Attribute

Wegweiser


Dateien und Verzeichnisse
Implementationsaspekte
Ablauf eines Dateizugriffs

Schnittstelle

Namensdienst / Lookup

- Pfadname → i-node-Nummer → Dateideskriptor
- **`open()`** ... Zugriff ... **`close()`**

Zugriff auf Daten

- Kopieren aus Datei in Adressraum und zurück:
  `read`/**`write`**`(`Datei, Länge, Adresse im Adressraum`)`
- Position in Datei wird implizit geändert oder explizit über **`seek()`**
- Einblenden in Adressraum:
  `mmap``(`Datei, Adresse im Adressraum, Offset, Länge`)`
- Zugriff auf Dateiattribute: **`stat()`**, ...

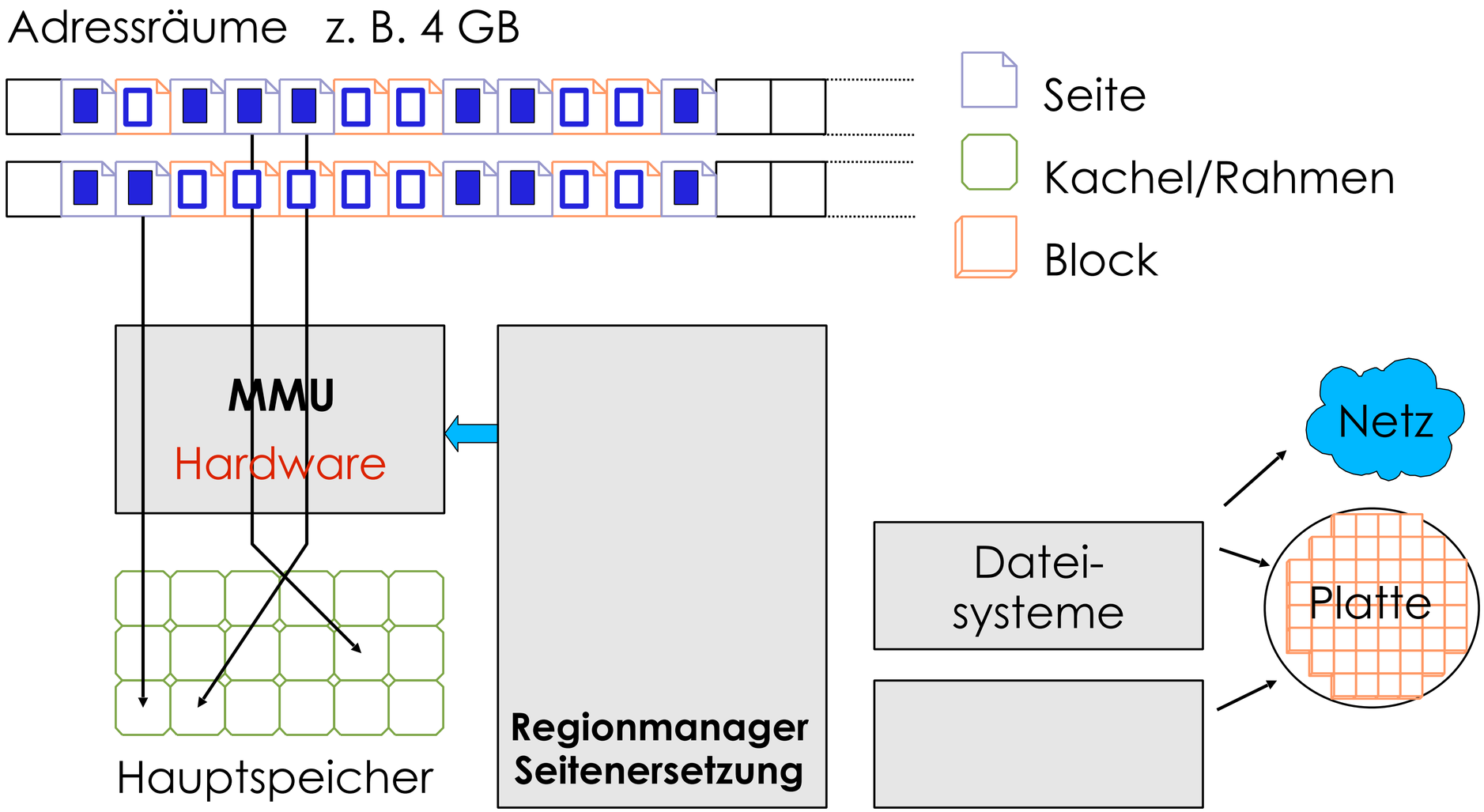
Beispiel: Konto

```
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;
```

Integration in Systemarchitektur



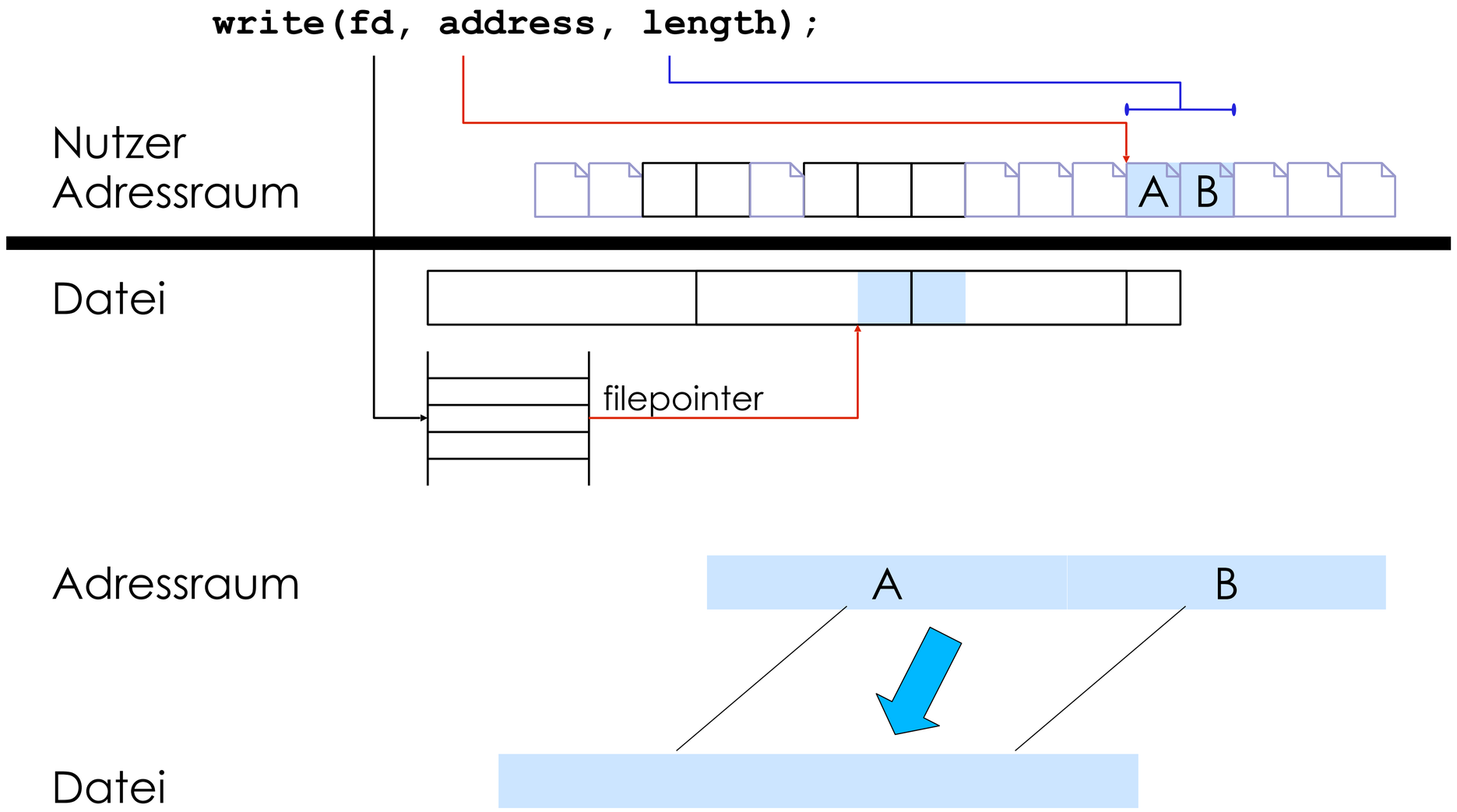
Pufferverwaltung beim Dateizugriff mit „write“

```
fd = open("papers/xy.tex", O_RDWR)
seek(fd, position)
write(fd, address, length)
```

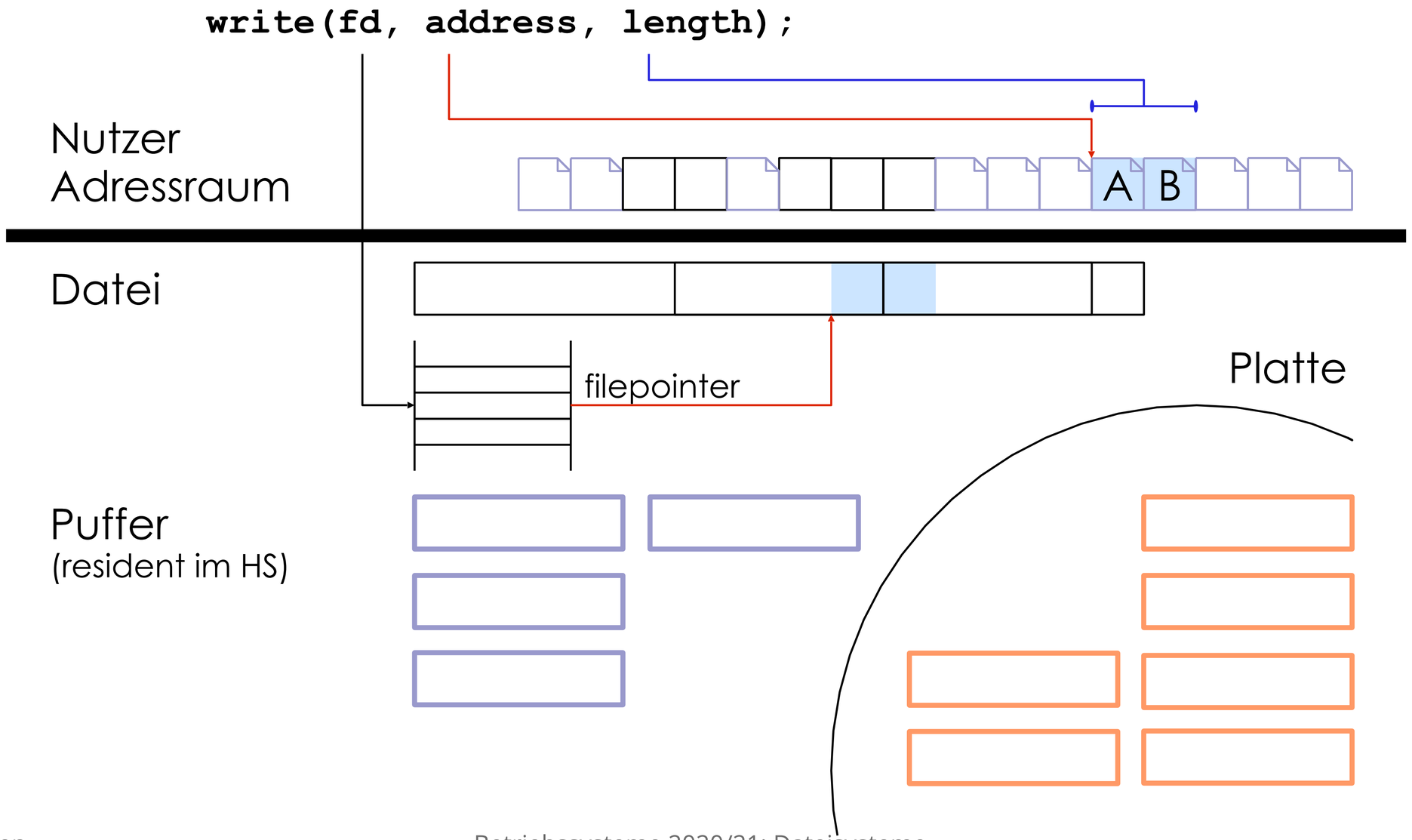
Buffer Cache

- Struktur im Kern-Adressraum (Hauptspeicher)
- zum schnellen Bedienen von Lese-/Schreibzugriffen
- Zuordnung: Plattenadresse ↔ Puffer
- Inhalte von Platte laden, geänderte Inhalte zurückschreiben

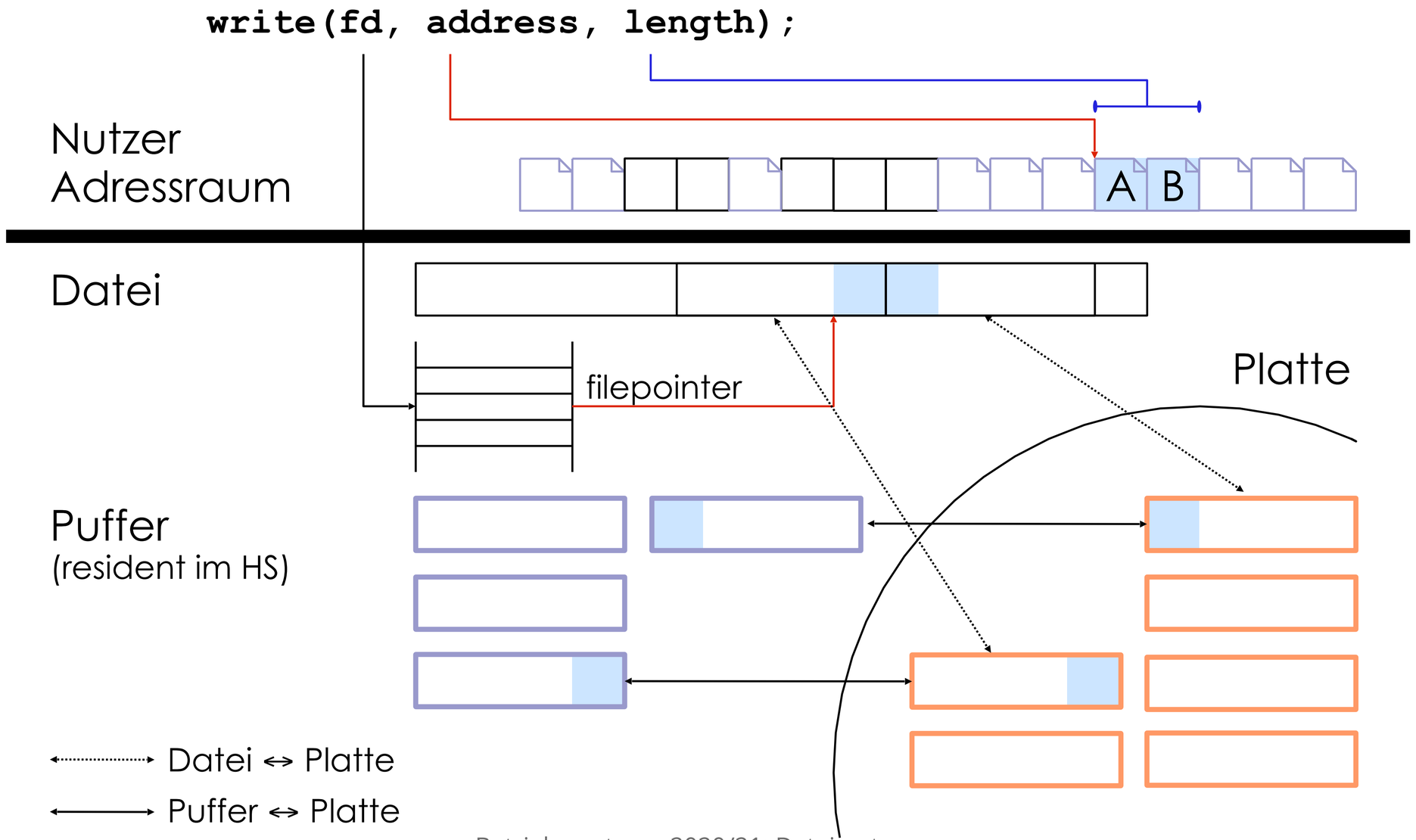
Ablauf einer write()-Operation



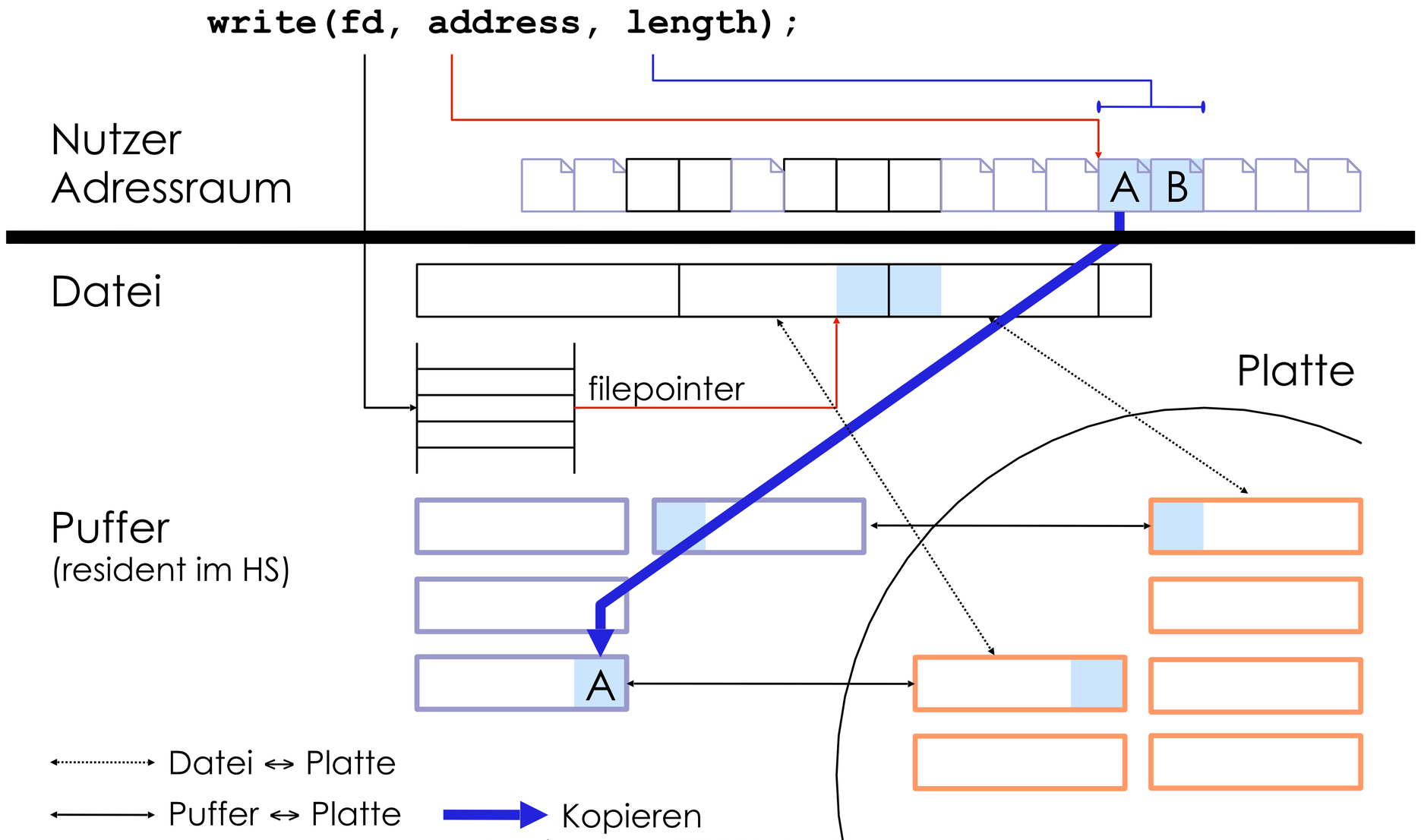
Ablauf einer write()-Operation



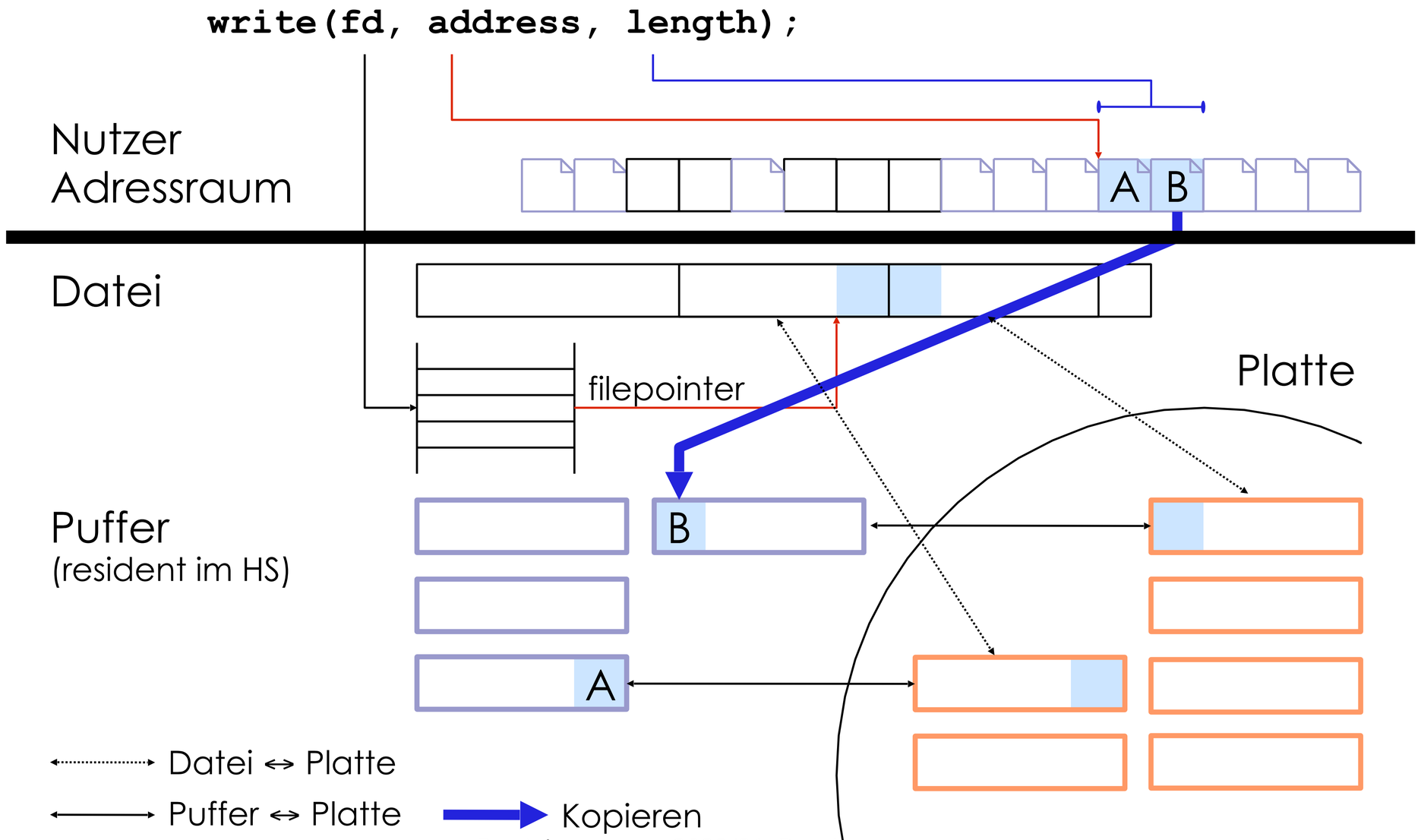
Ablauf einer write()-Operation



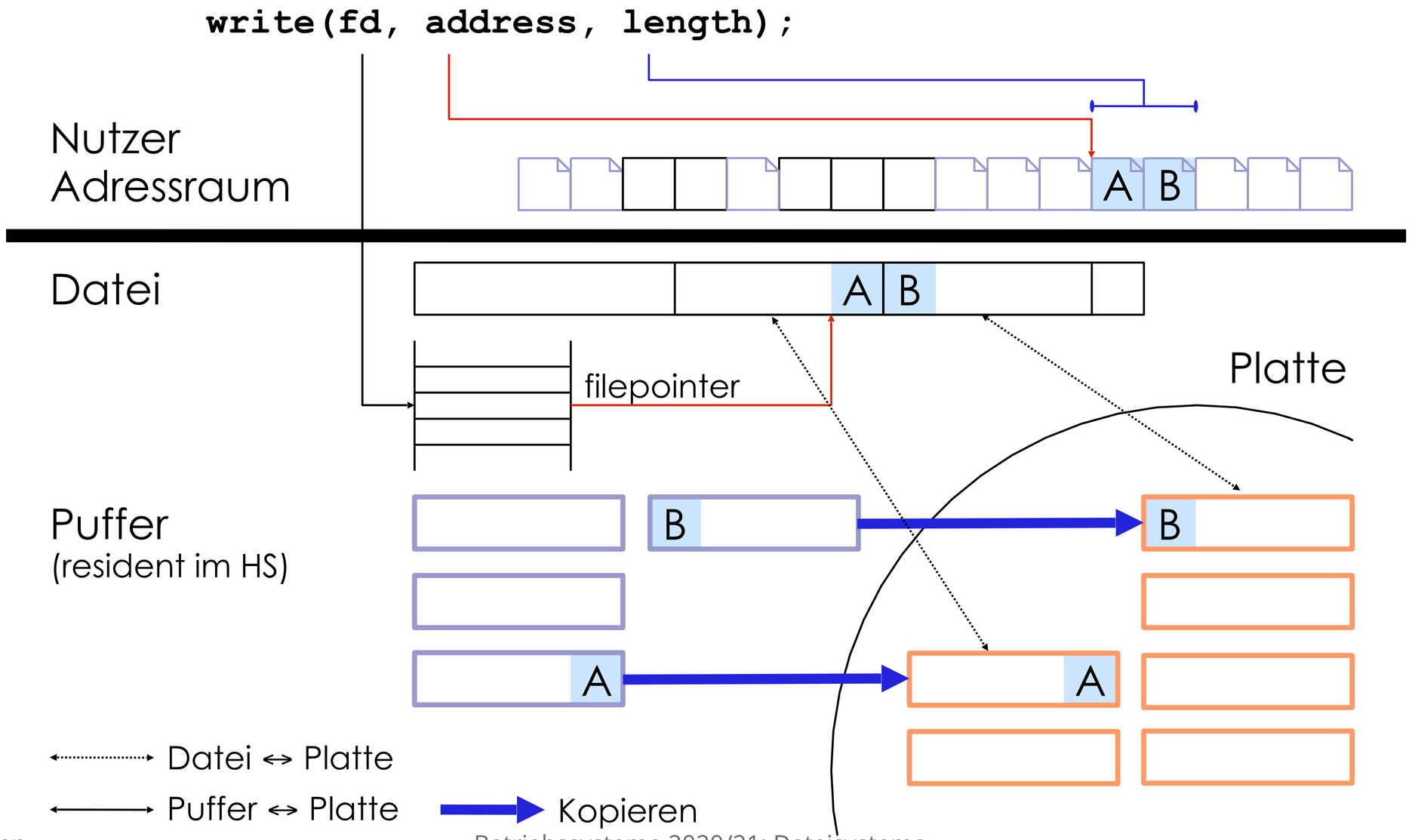
Ablauf einer write()-Operation



Ablauf einer write()-Operation



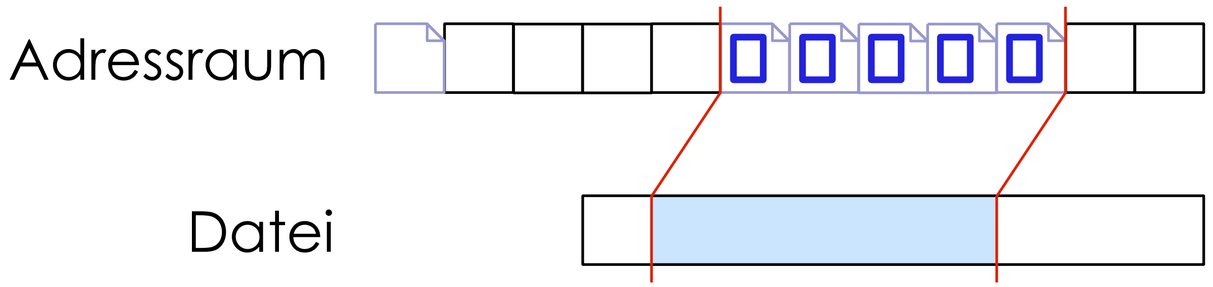
Ablauf einer write()-Operation



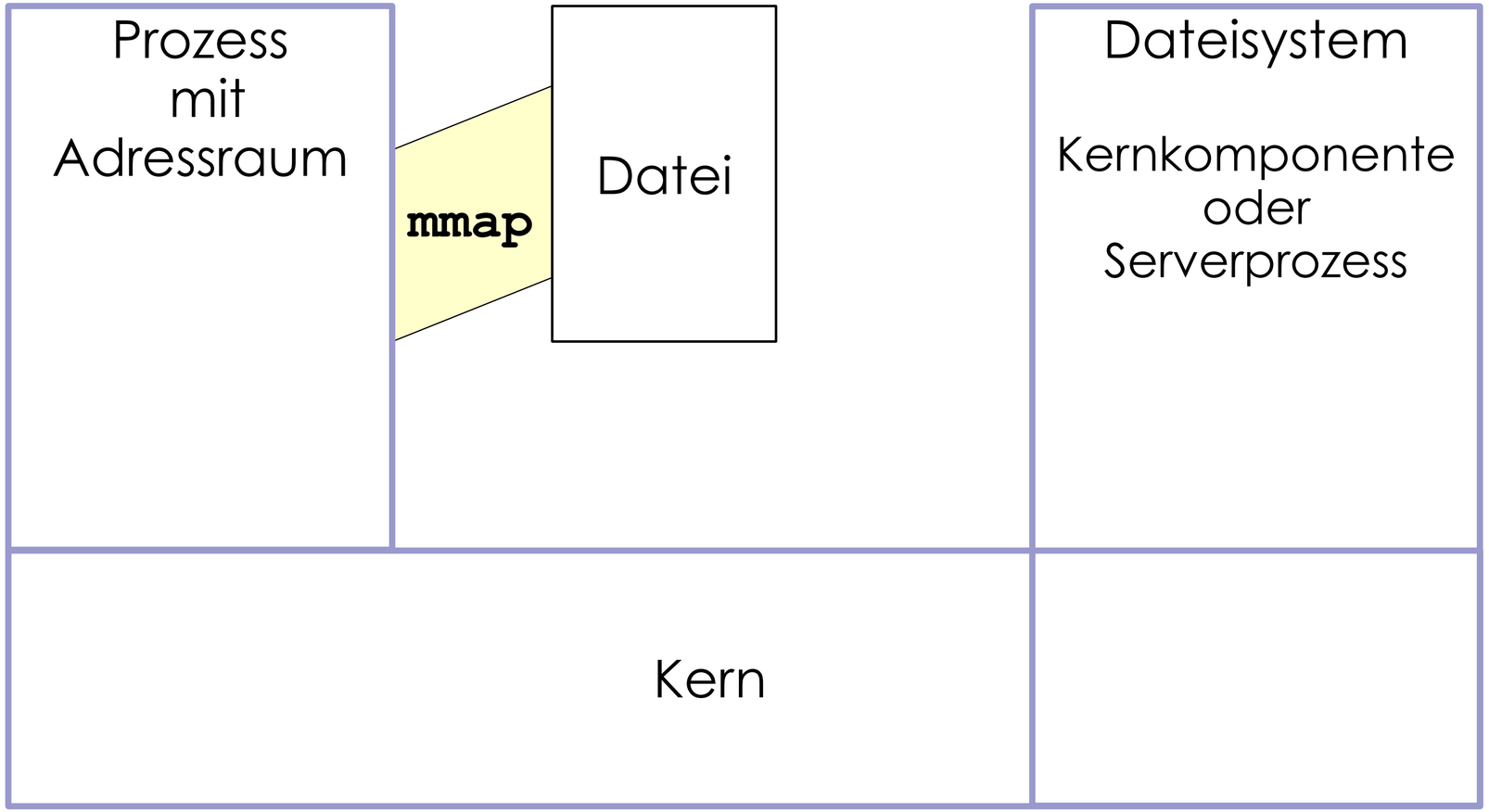
Zugriff mittels Einblendung

`mmap(Datei, Adresse, Länge, Offset)`

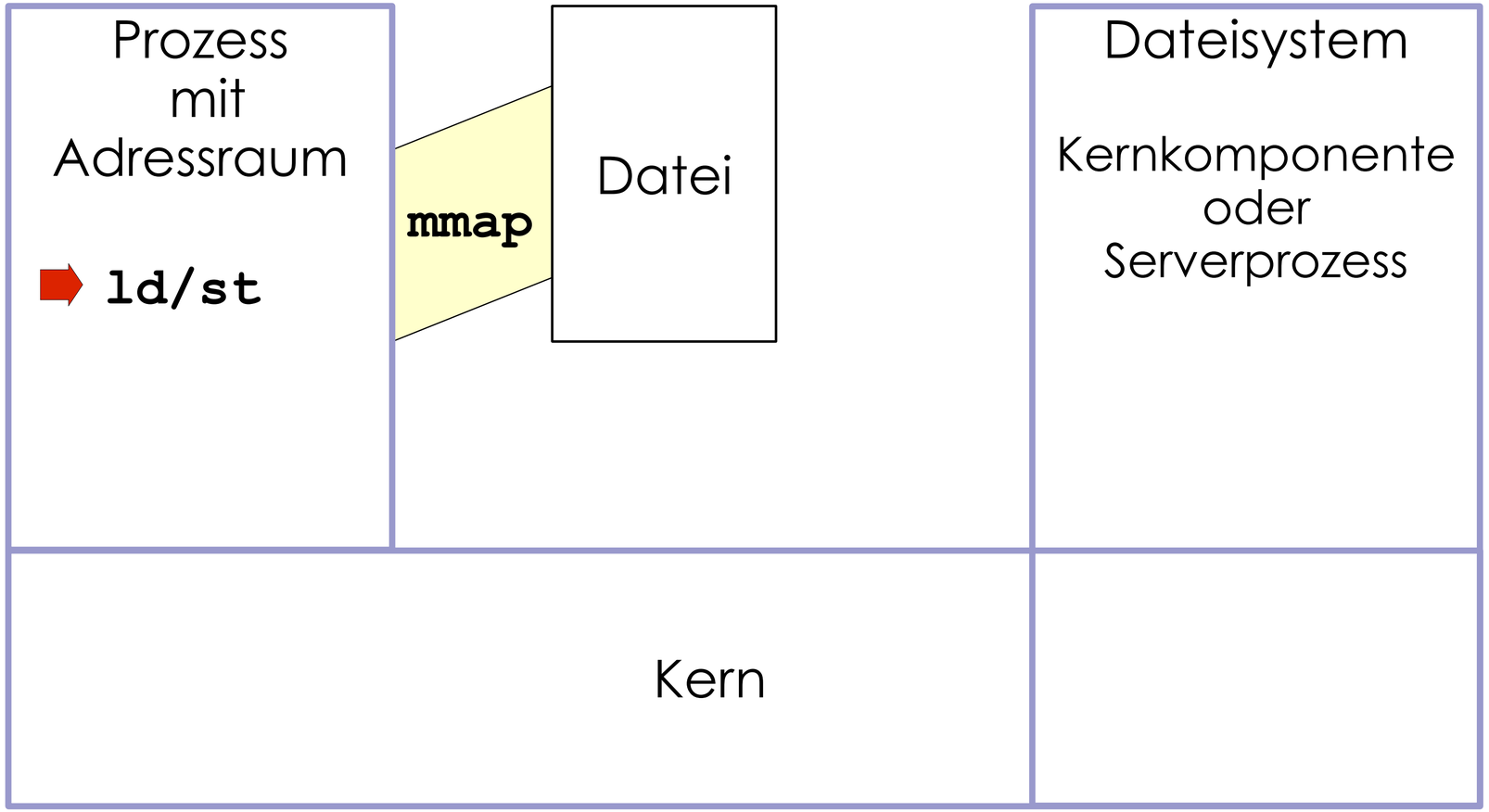
- Zugriff auf Datei über Seitenfehlerbehandlung
- dann Zugriff auf Inhalt über normale HW-Instruktionen



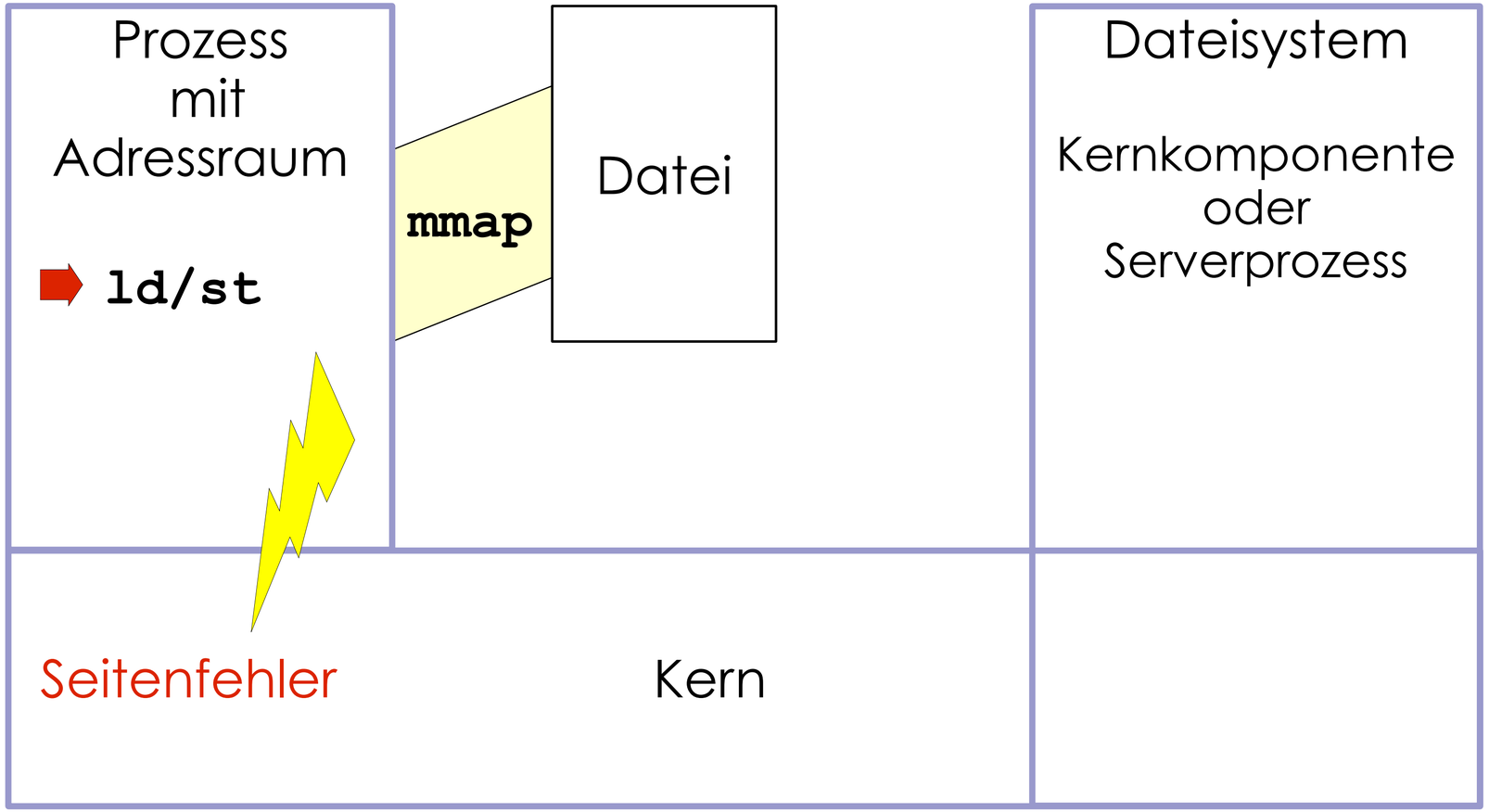
Zugriff mittels Einblendung



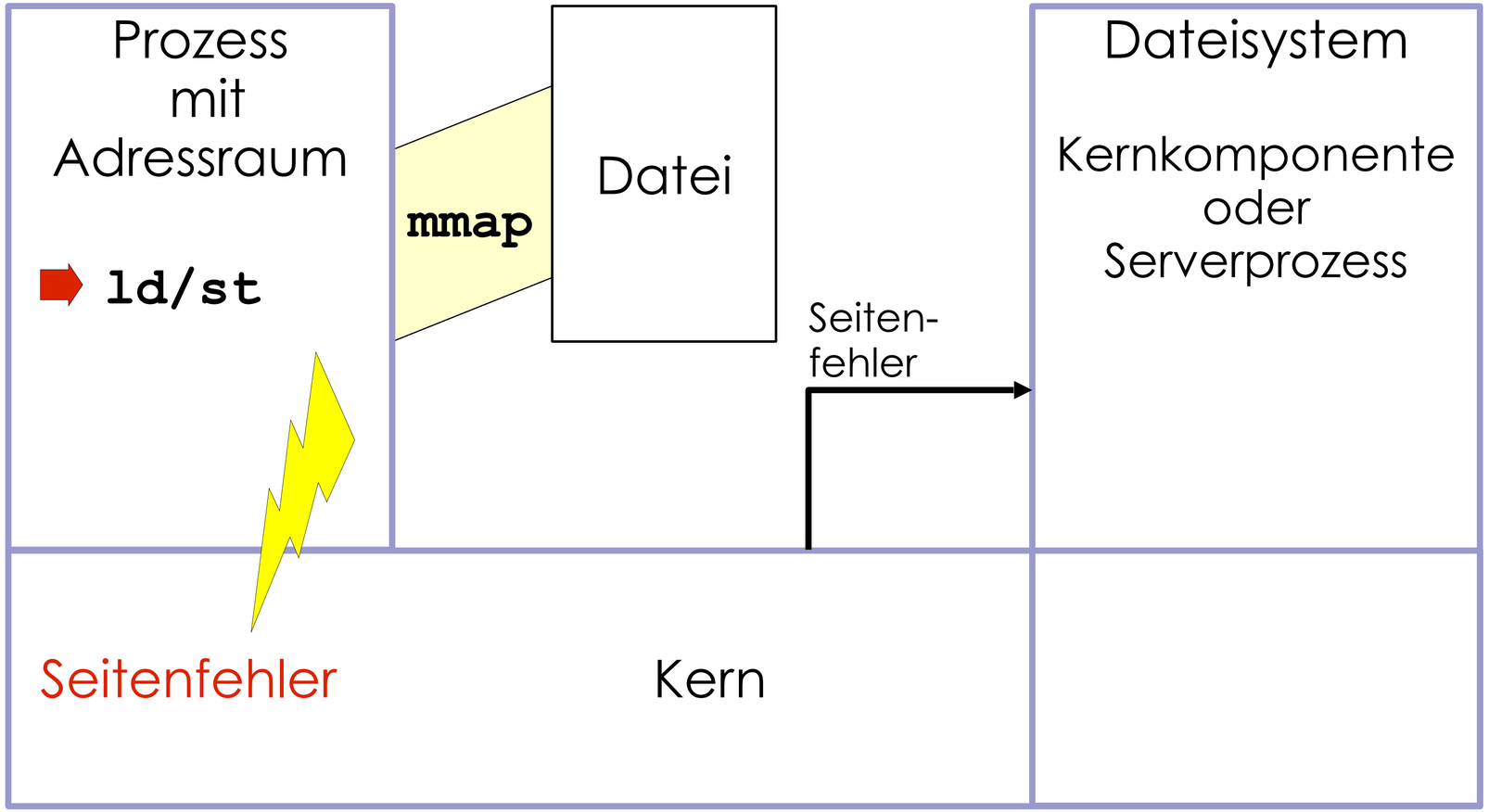
Zugriff mittels Einblendung



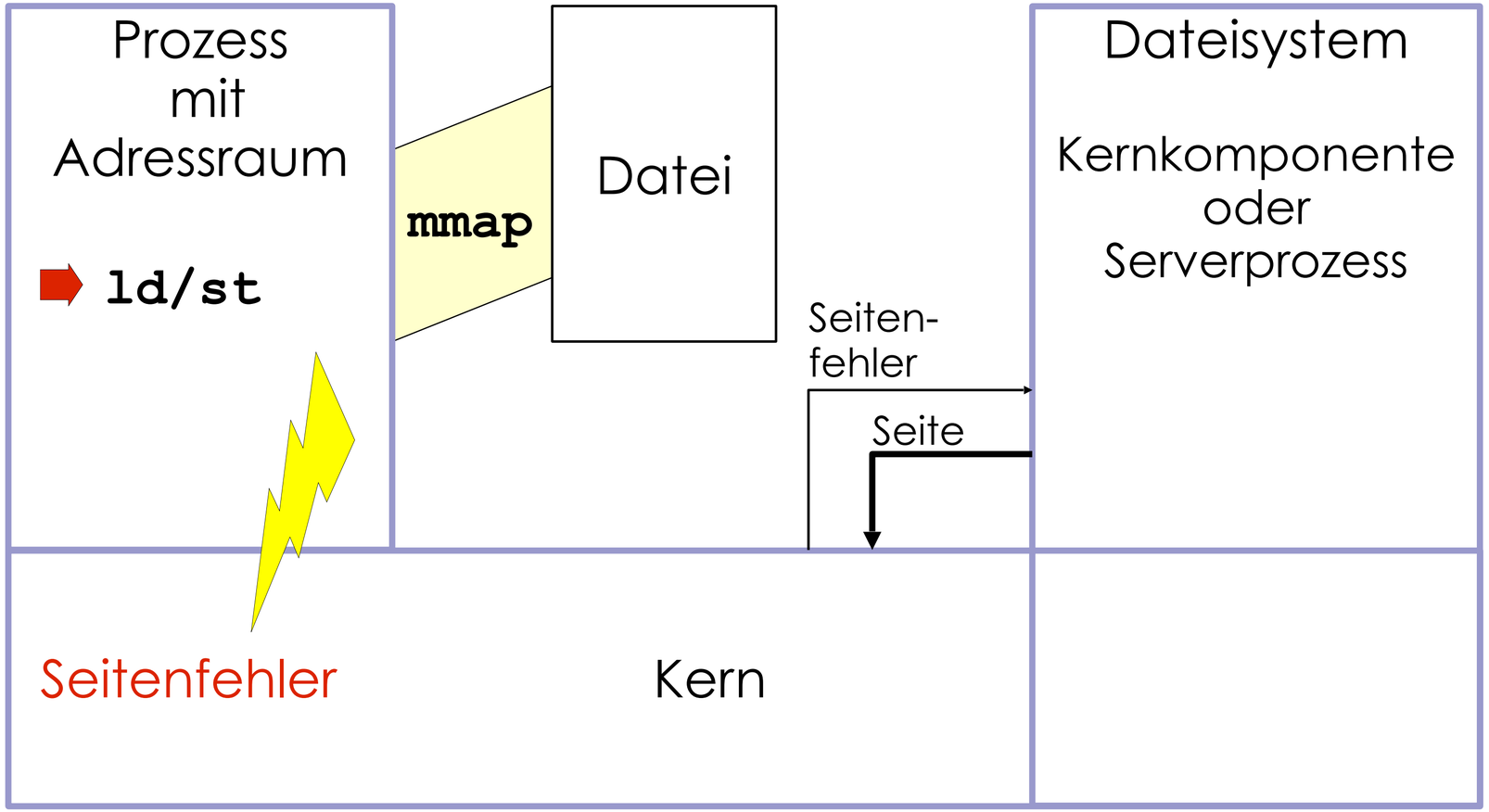
Zugriff mittels Einblendung



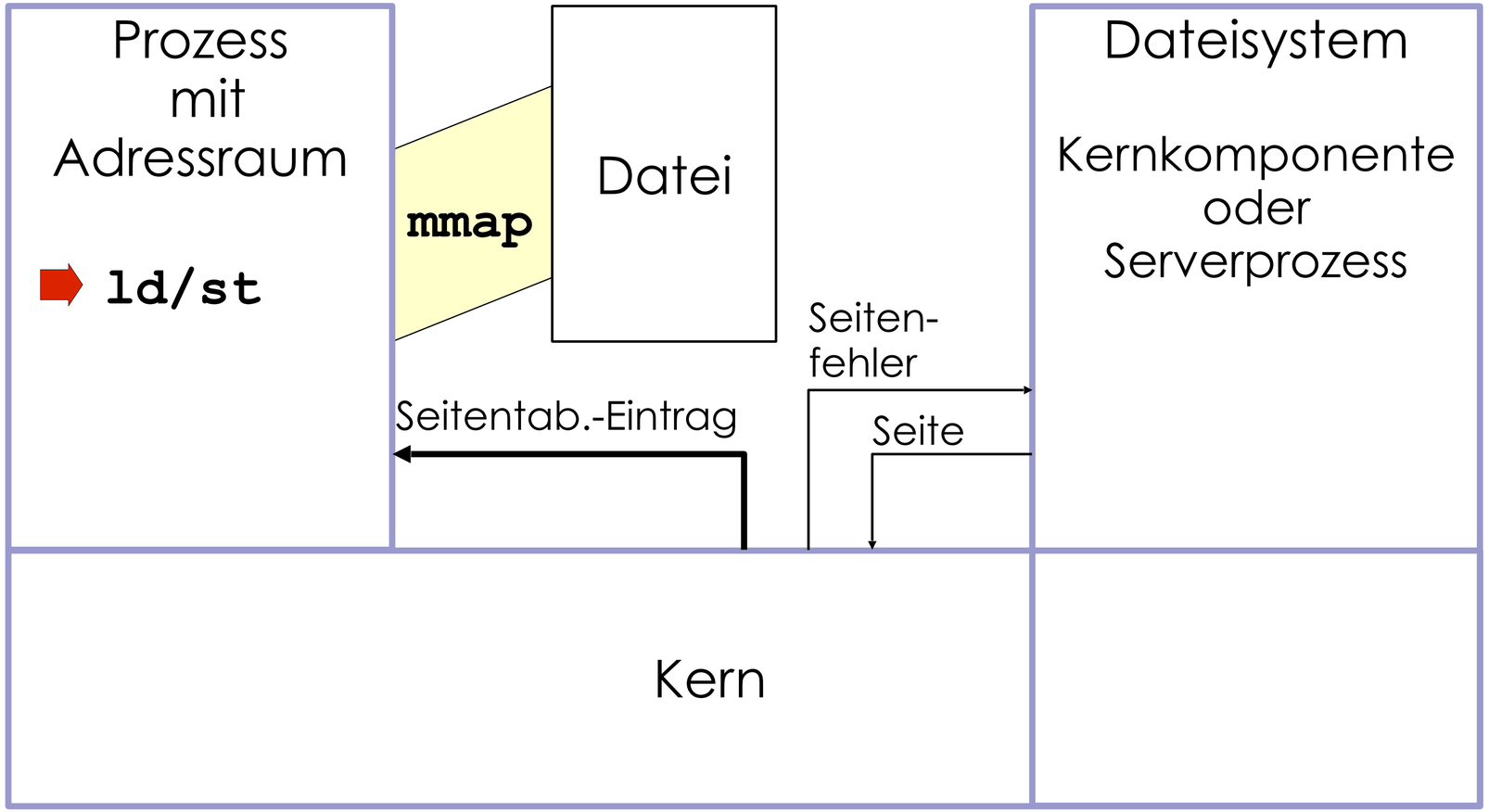
Zugriff mittels Einblendung



Zugriff mittels Einblendung



Zugriff mittels Einblendung



Gegenüberstellung: read/write vs. mmap

read/write

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

mmap

- Einsparen von Kopieroperationen
- Synchronisation wie bei gemeinsamem Speicher
- Bereich muss Vielfaches der Seitengröße sein
- Overhead zum Einrichten der Seitentabelle

Zusammenfassung

- Integration von Dateisystemen in Systemarchitektur
- Prinzipielle Aufgaben
- Beispiel-Operationen

Offene Probleme

- Absturz des Rechners → Verlust von Pufferinhalten, Inkonsistenzen
- Optimierung für schnelle Speichermedien: Flash
- Ausfall von Platten