UNIVERSITÄT BERN

2405 Betriebssysteme
X. Implementierung von Dateisystemen

Thomas Staub, Markus Anwander
Universität Bern

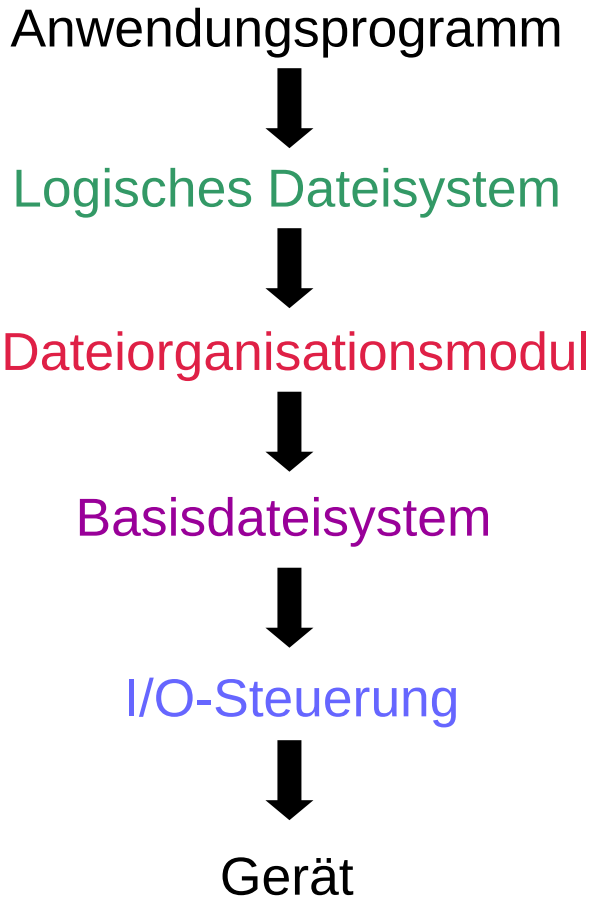
Inhalt

1. Implementierungsarchitektur von Dateisystemen
2. Implementierung von Dateisystemen
   1. Datenstrukturen zur Dateisystem-Implementierung
   2. File Control Block
   3. Tabellen im Hauptspeicher
      1. Öffnen einer Datei
      2. Lese- oder Schreibzugriff auf Dateien
   4. Virtuelles Dateisystem
   5. Network File System
   6. Implementierung von Verzeichnissen
      1. Verzeichnisimplementierung in UNIX
      2. B-Bäume
3. Allokation von Dateiblöcken
   1. Zusammenhängende Allokation
   2. Verkettete Allokation
   3. Indizierte Allokation
4. Freispeicherverwaltung
   1. Bitvektoren
   2. Verkettete Freispeicherliste
   3. Freispeicherliste mit Gruppieren
   4. Freispeicherliste mit Zählen
   5. Space Maps
5. Zuverlässigkeit
   1. Konsistenzprüfung
   2. Backup
   3. Log-Structured File Systems
   4. Journaling File Systems

1. Implementierungsarchitektur von Dateisystemen

UNIVERSITÄT BERN

- Logisches Dateisystem
  - Anbieten von Datei- und Verzeichnisoperationen
  - Verwalten von Verzeichnis- und Dateistrukturen
  - Schutzmechanismen
- Dateiorganisationsmodul
  - Übersetzung logischer Blockadressen in physikalische
  - Speicherallokation
  - Freispeicherverwaltung
  - Festplattenmanagement
- Basisdateisystem
  - Kommandoübergabe an I/O-Steuerung, z.B. „Lese, Disk1, Zylinder 73, Spur 2, Sektor 10“
  - Lesen und Schreiben von Blöcken
  - Festplatten-Scheduling
  - Caching
- I/O-Steuerung
  - Gerätetreiber und Interrupt-Handler



2.1 Datenstrukturen zur Dateisystem-Implementierung

UNIVERSITÄT BERN

auf der Disk

- Informationen zum Booten des Systems (Boot-Block)
- Informationen über die einzelnen Partitionen (Volume Control Block) oder superblock
- Verzeichnisstruktur
- File Control Blocks
  - enthalten Details über Datei, z.B. Zugriffsrechte, Daten, Grösse, Dateiblöcke
  - Beispiel: i-node

im Hauptspeicher

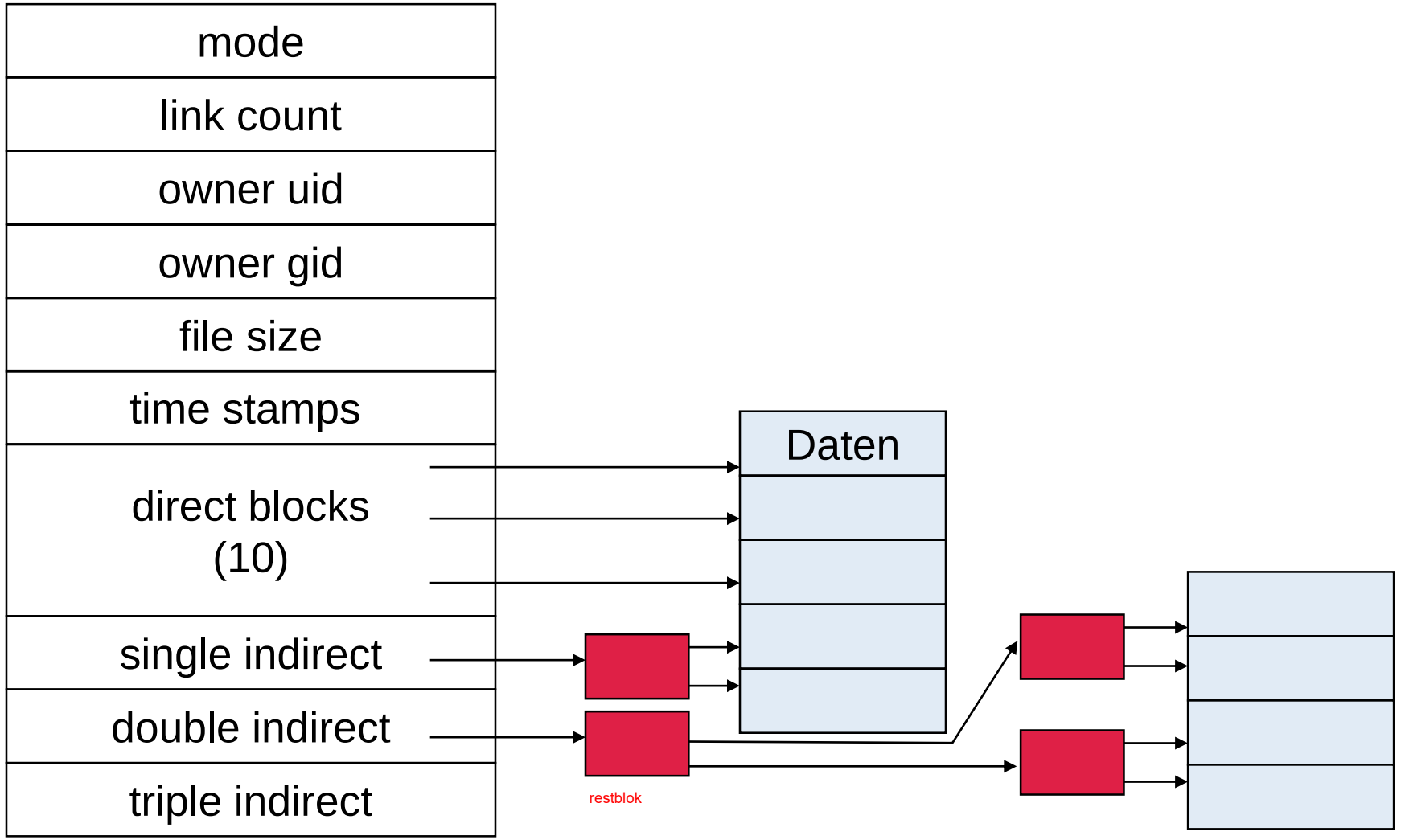
- Informationen über „gemountete“ Partitionen und im Cache vorhandene Verzeichnisse
- Tabellen
  - systemweite Tabelle offener Dateien
  - Tabelle offener Dateien für jeden Prozess
- Puffer zum Lesen / Schreiben von / zur Disk

2.2 File Control Block

- Zugriffsmodi
- Datum der Erzeugung, der Modifikation oder des letzten Zugriffs
- Eigentümer, Gruppe und Zugriffsrechte
- Grösse
- Datenblöcke oder Zeiger auf solche
- Referenzzähler

- Beispiel: UNIX i-node

2.2.1 UNIX i-node

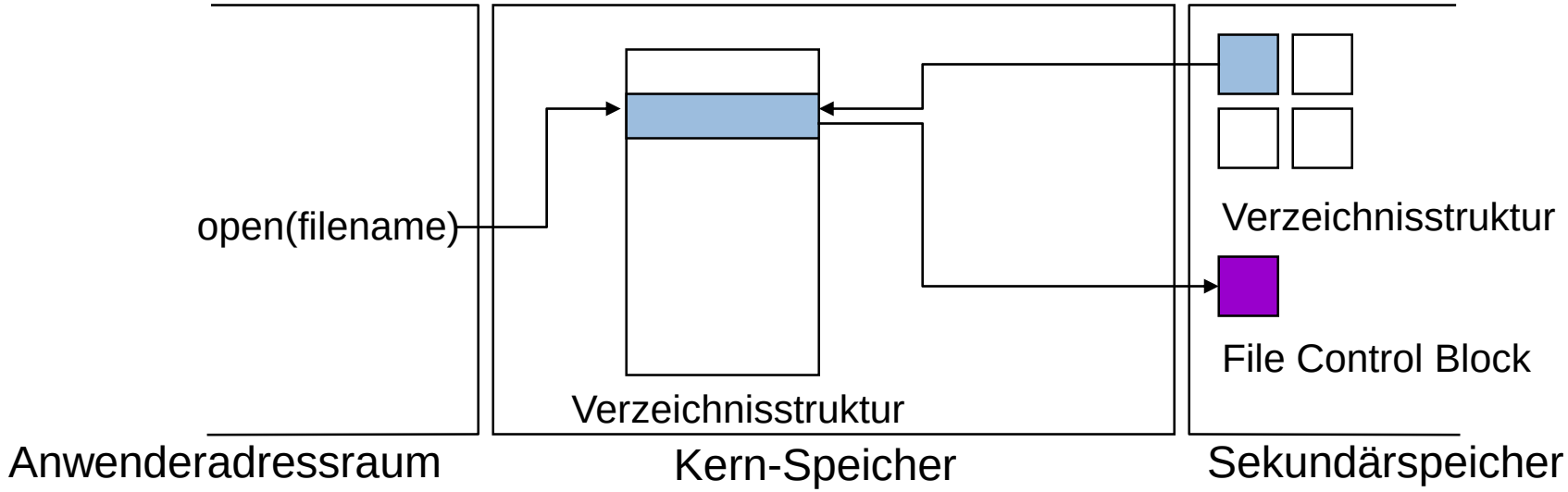


2.3 Tabellen im Hauptspeicher

- Systemweite Tabelle aller geöffneter Dateien
  - mit prozessunabhängigen Informationen, z.B.
    - Zähler, wie viele Prozesse eine Datei geöffnet haben
    - Lokationsinformation
    - Zugriffsinformation, z.B. Zugriffsdatum, Zugriffsrechte etc.
    - Länge der Datei
    - Eigentümer
  - zum schnellen Zugriff auf Informationen über eine Datei
- Tabelle pro Prozess mit den von diesem geöffneten Dateien
  - Dateideskriptor
  - Zusätzliche Informationen über Datei, z.B. aktuelle Position
  - Zeiger auf systemweite Tabelle

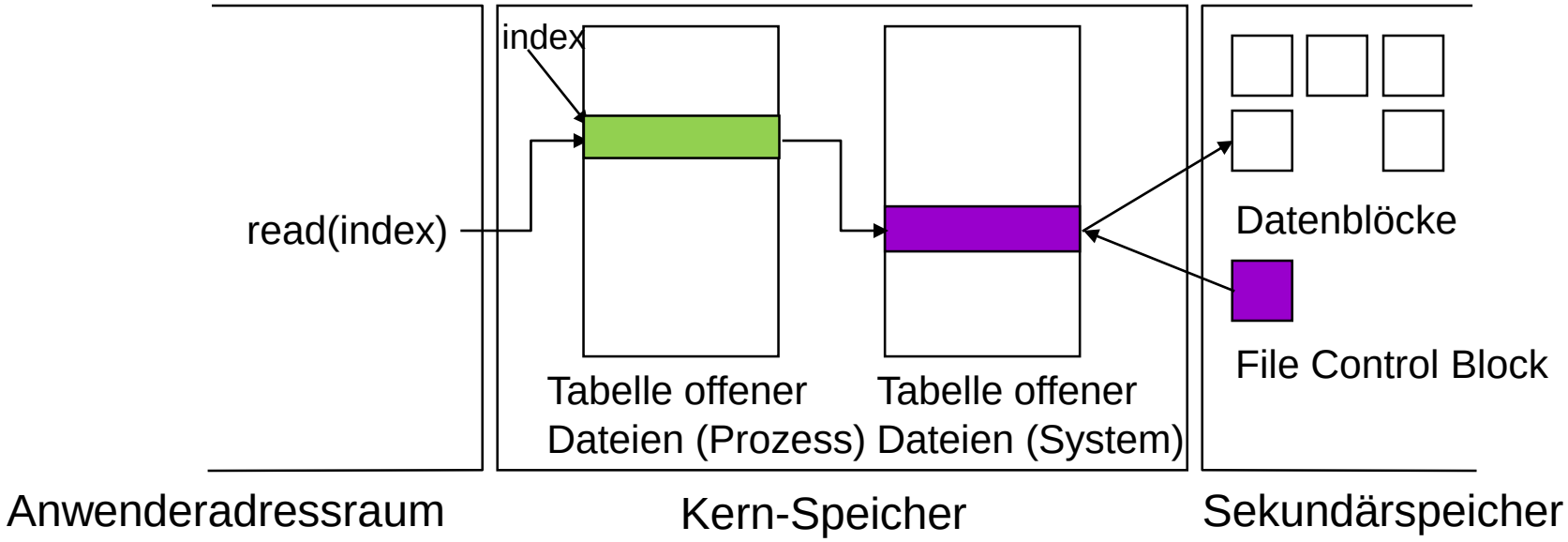
2.3.1 Öffnen einer Datei

- open-Aufruf leitet Dateiname an Dateisystem weiter.
- Suche der Verzeichnisstrukturen (Caching im Hauptspeicher)



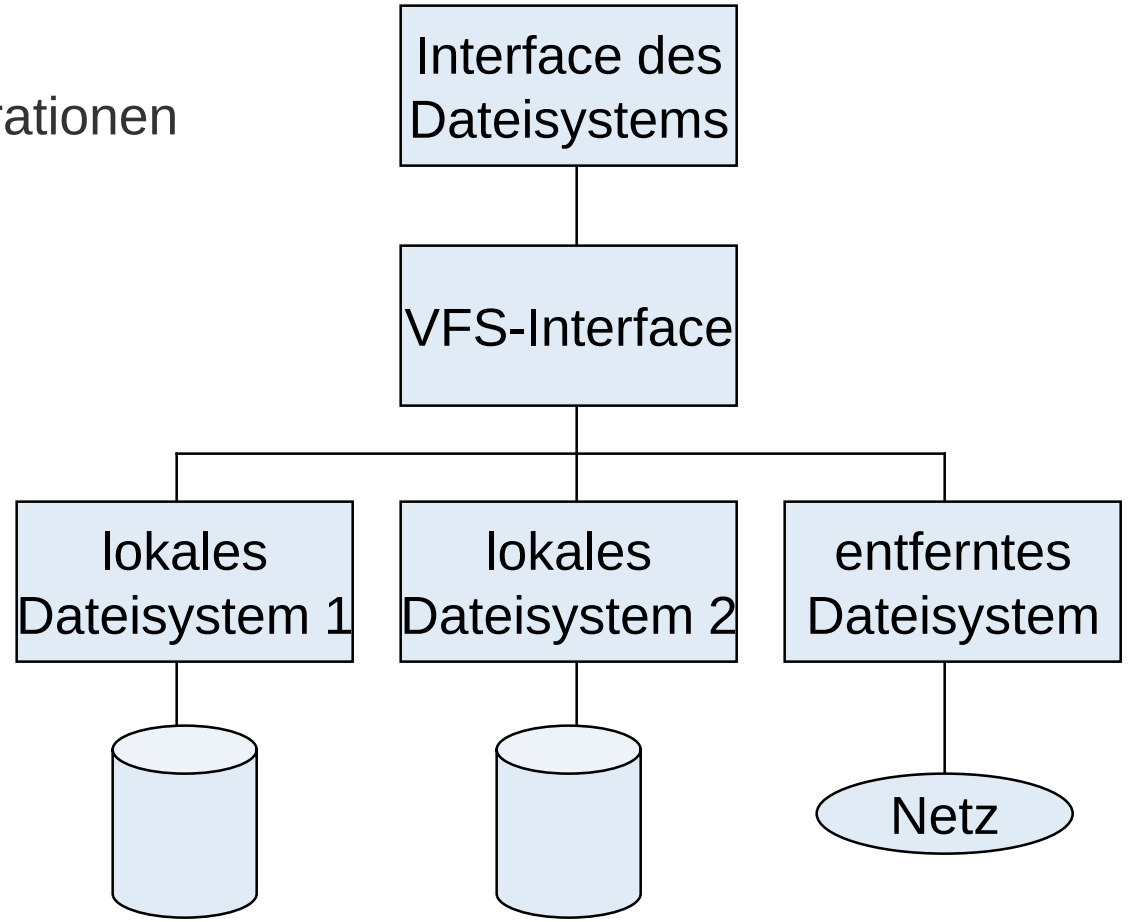
- Kopieren des File Control Block in systemweite Tabelle offener Dateien
- Anlegen eines Eintrags in Tabelle offener Dateien des Prozesses
- open-Aufruf liefert Verweis auf den Eintrag für weitere Dateizugriffe zurück (Dateideskriptor)

2.3.2 Lese- oder Schreibzugriff auf Dateien

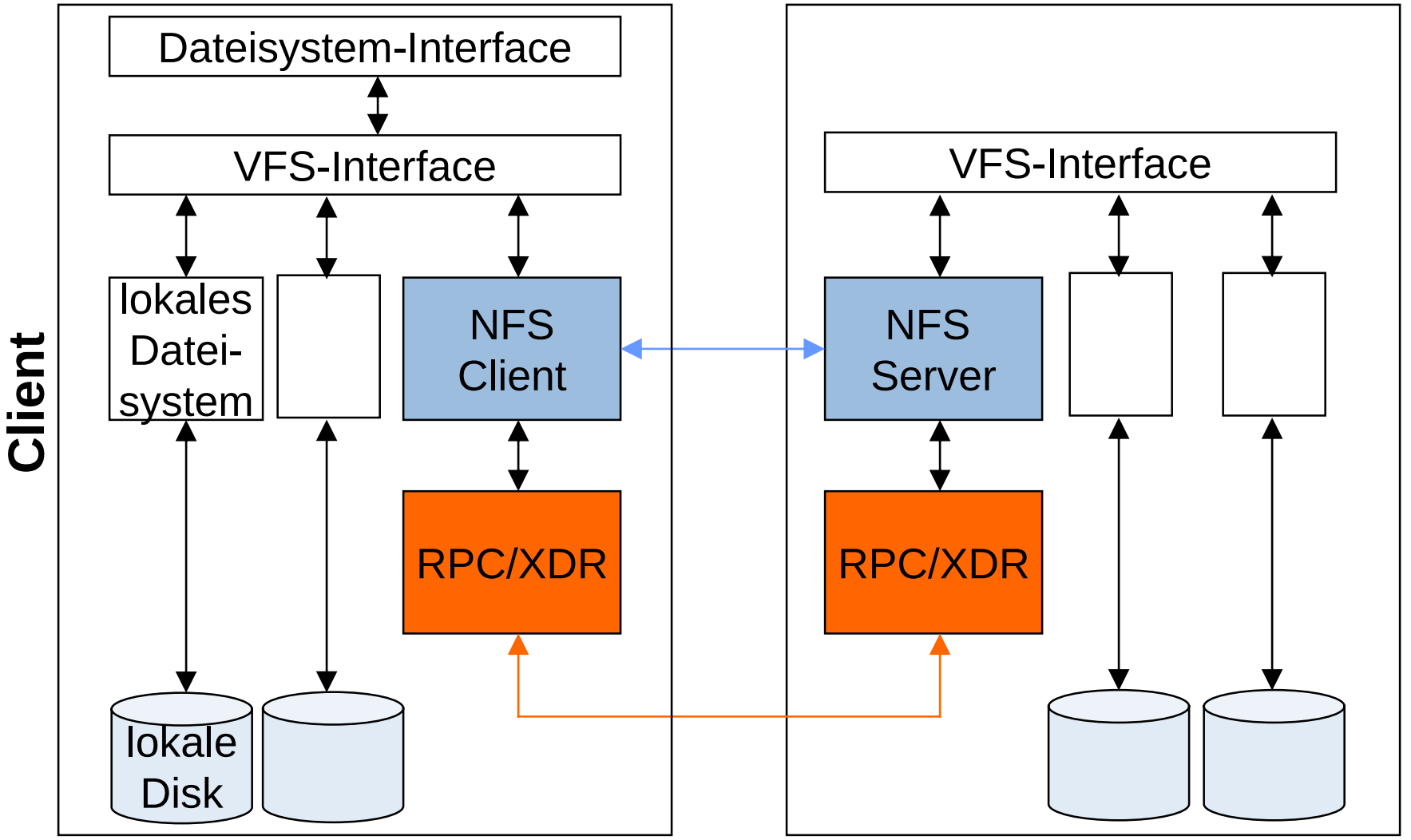


2.4 Virtuelles Dateisystem

- Integration verschiedener Dateisysteme in eine Verzeichnishierarchie
- Virtual File System (VFS)
  - Trennung von allgemeinen Dateisystem-Operationen von ihrer Implementierung
  - Netz-eindeutige Identifikation von Dateien



2.5 Network File System



- NFS-Protokolle
  - Mount-Operationen
  - Dateizugriff, z.B. Lesen/Schreiben von Dateien und Dateiattributen
- Remote Procedure Call und eXternal Data Representation

2.6 Implementierung von Verzeichnissen

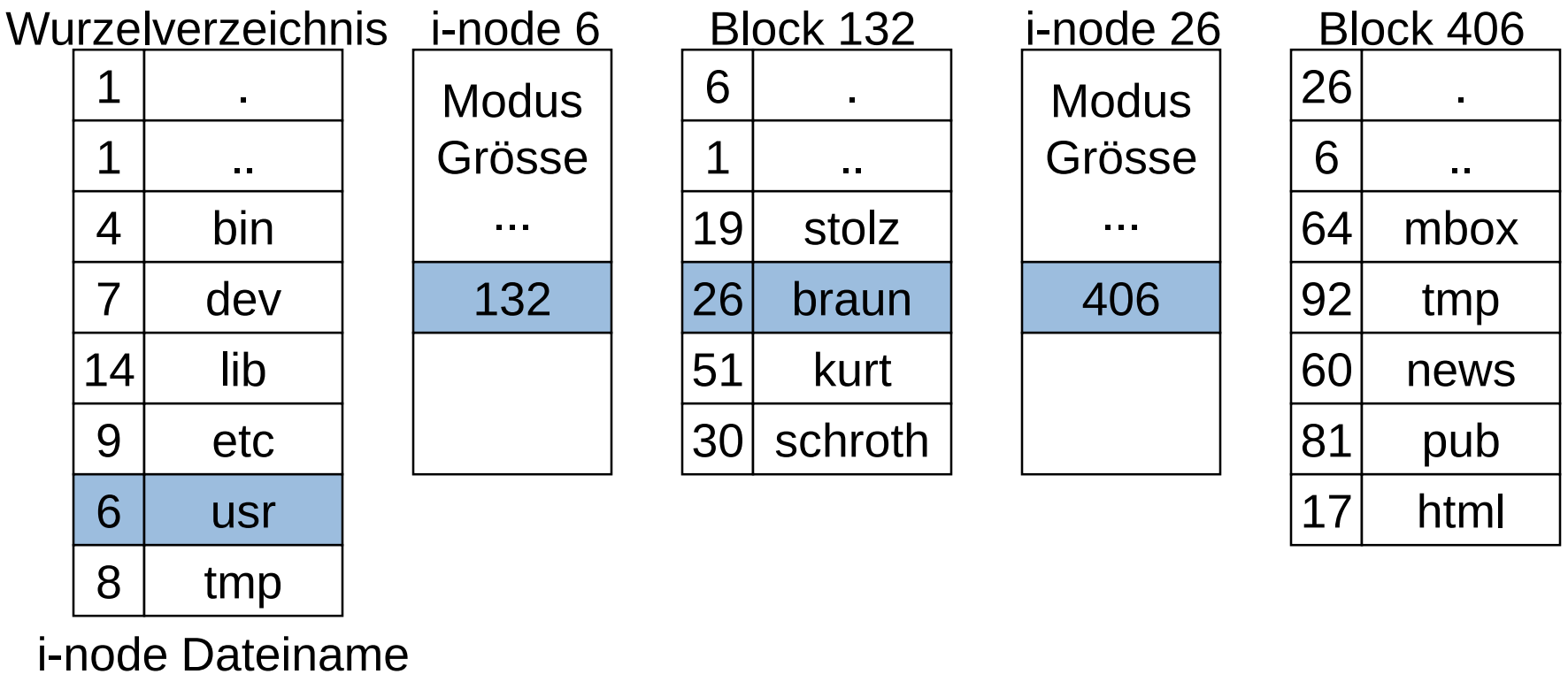
Lineare Liste mit Dateinamen und Zeigern auf Datenblöcke

- einfach zu programmieren
- aufwändig zu durchsuchen
- Varianten
  - Bäume
  - sortierte Listen
  - Caching

Hash-Tabelle

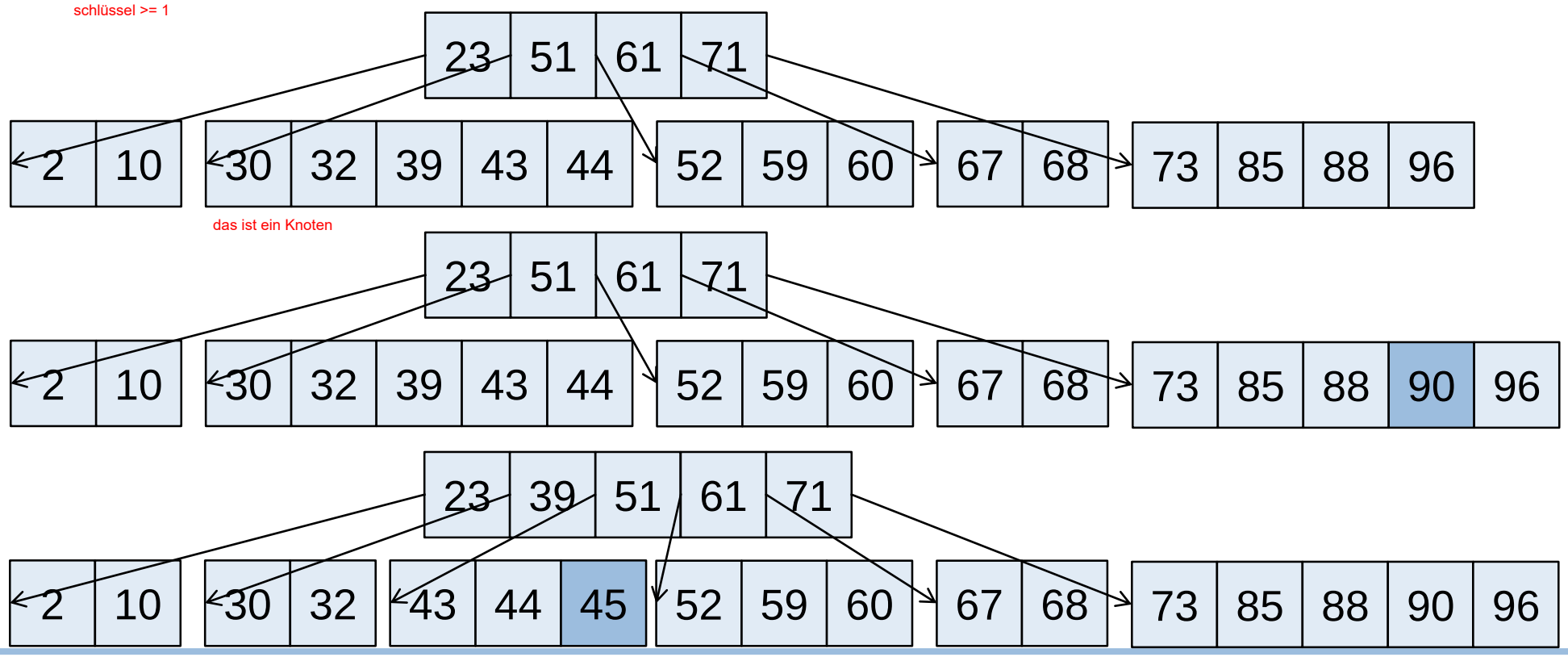
- Lineare Liste von Dateinamen + Hash-Tabelle
- Berechnung eines Hash-Werts aus Dateinamen und Rückgabe eines Zeigers auf Dateinamen in linearer Liste
- Kollisionsbehebung über verkettete Liste
- reduziert Suchzeit in einem Verzeichnis

2.6.1 Verzeichnisimplementierung in UNIX

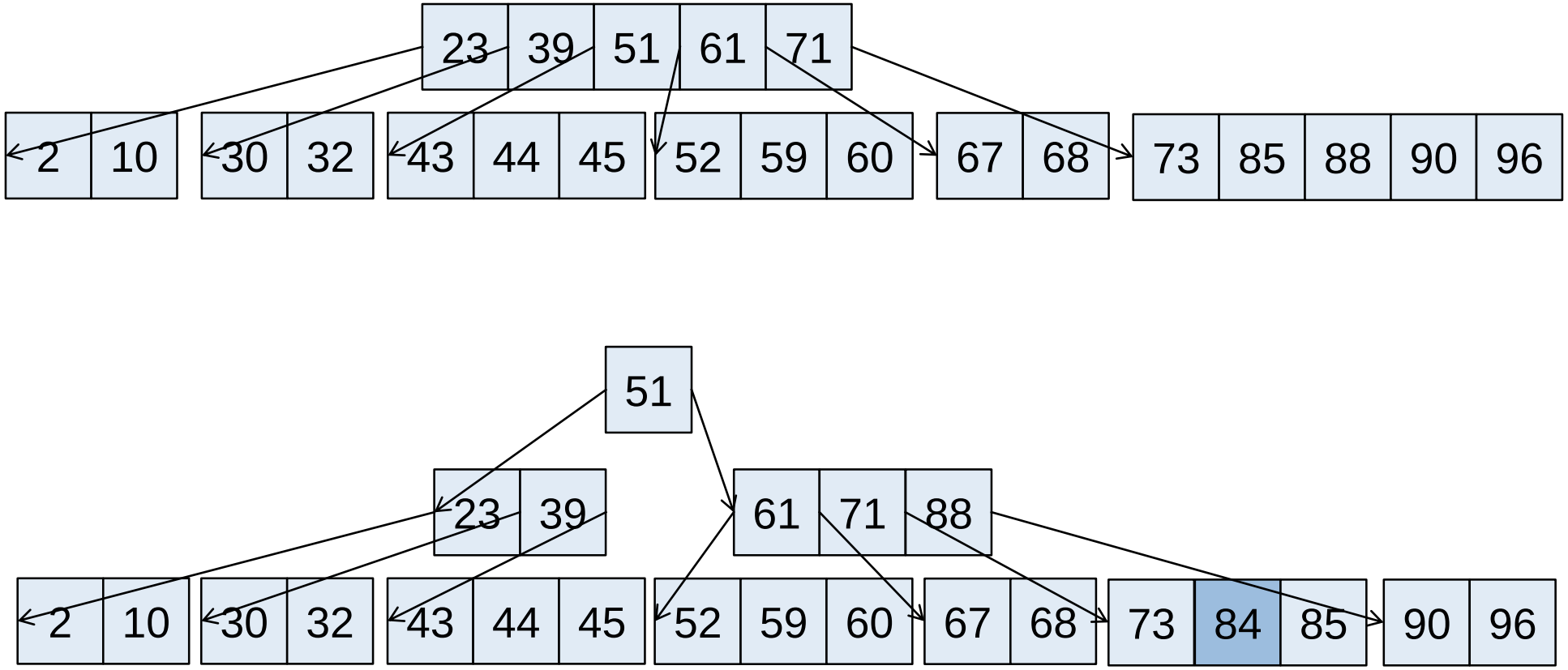


2.6.2.1 B-Bäume

- Verwendung in Datenbanken und Verzeichnisimplementierungen (NTFS)
- Verzeichnisse verwenden Bäume zur Organisation von Dateien
- d-1 ≤ Schlüssel ≤ 2d-1; d ≤ Zeiger ≤ 2d; hier: d = 3 d= tiefe



2.6.2.2 B-Bäume

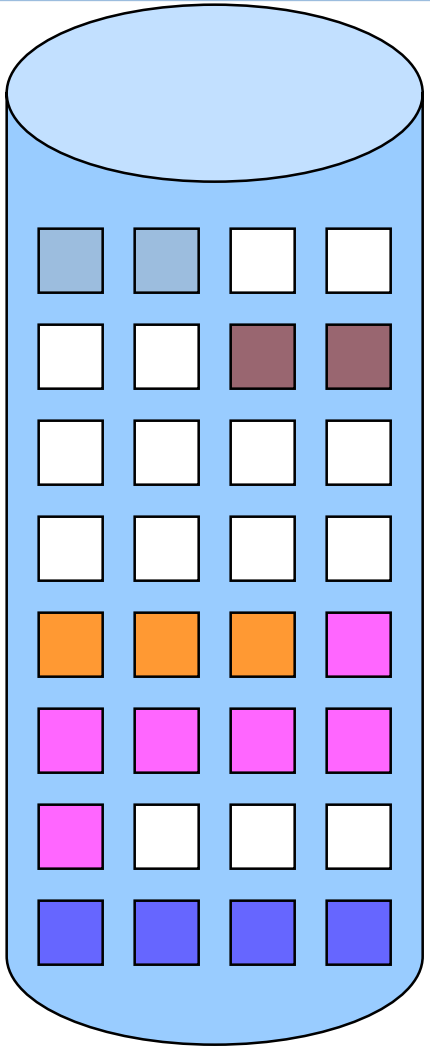

23 39 51 61 71
2 10 30 32 43 44 45 52 59 60 67 68 73 85 88 90 96
51
23 39 61 71 88
2 10 30 32 43 44 45 52 59 60 67 68 73 84 85 90 96

3. Allokation von Dateiblöcken

- Dateien benötigen Speicherplatz auf Disks
- Allokations-Mechanismen
  - zusammenhängend
  - verkettet
  - indiziert
- Ziele
  - effektive Ausnutzung der Festplatte (Disk)
  - schneller Dateizugriff

3.1 Zusammenhängende Allokation

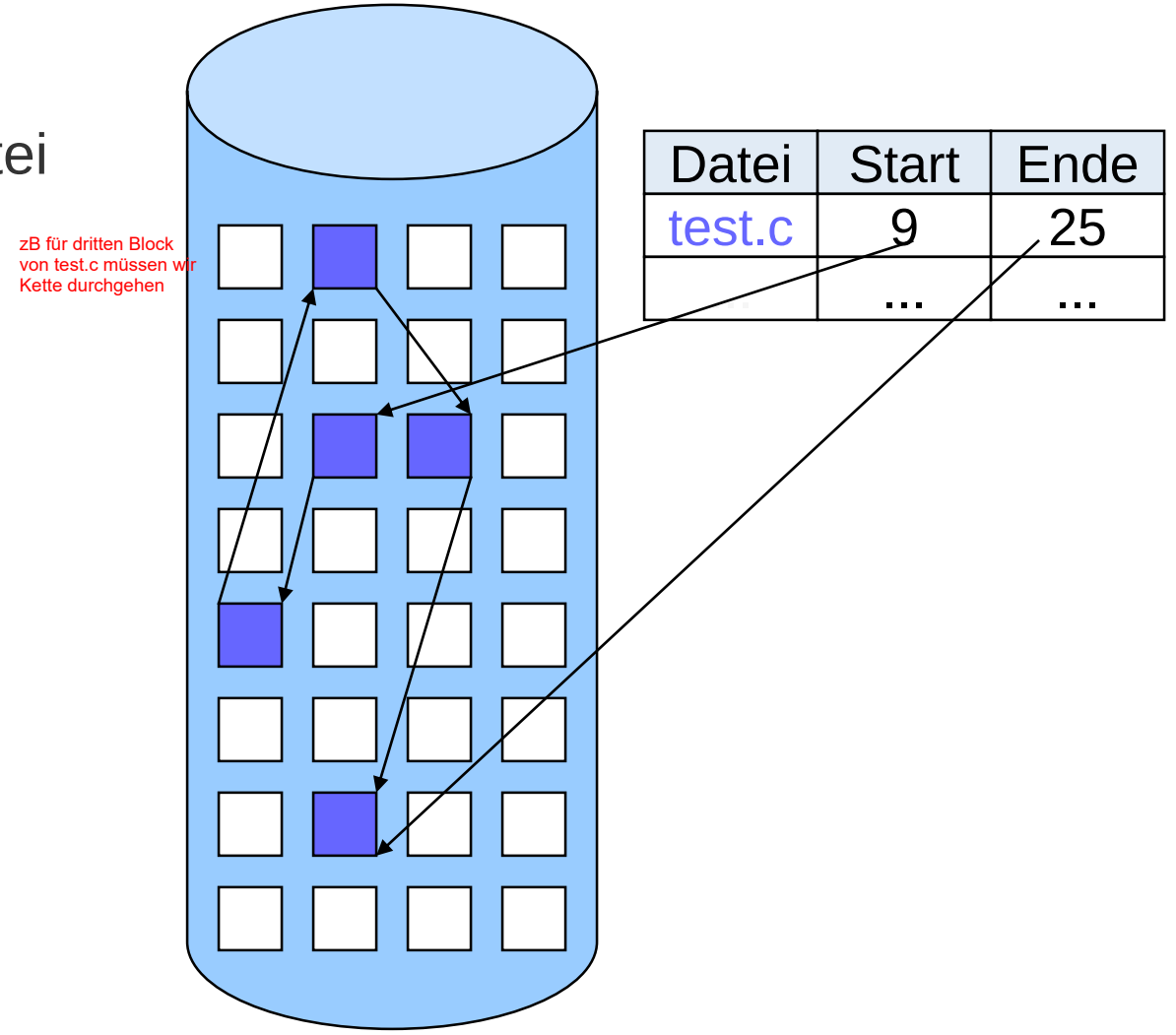
- Jede Datei belegt zusammenhängende Blöcke.
- einfache Implementierung und Abbildung (Start-Block, Länge)
- wahlfreier Zugriff
- Dateien können nicht wachsen.
- externer Verschnitt
- Platzverschwendung
- Allokation, z.B. best-, worst-, first-fit



Datei	Start	Länge
test.c	0	2
.profile	6	2
.plan	16	3
mail	19	6
news	28	4

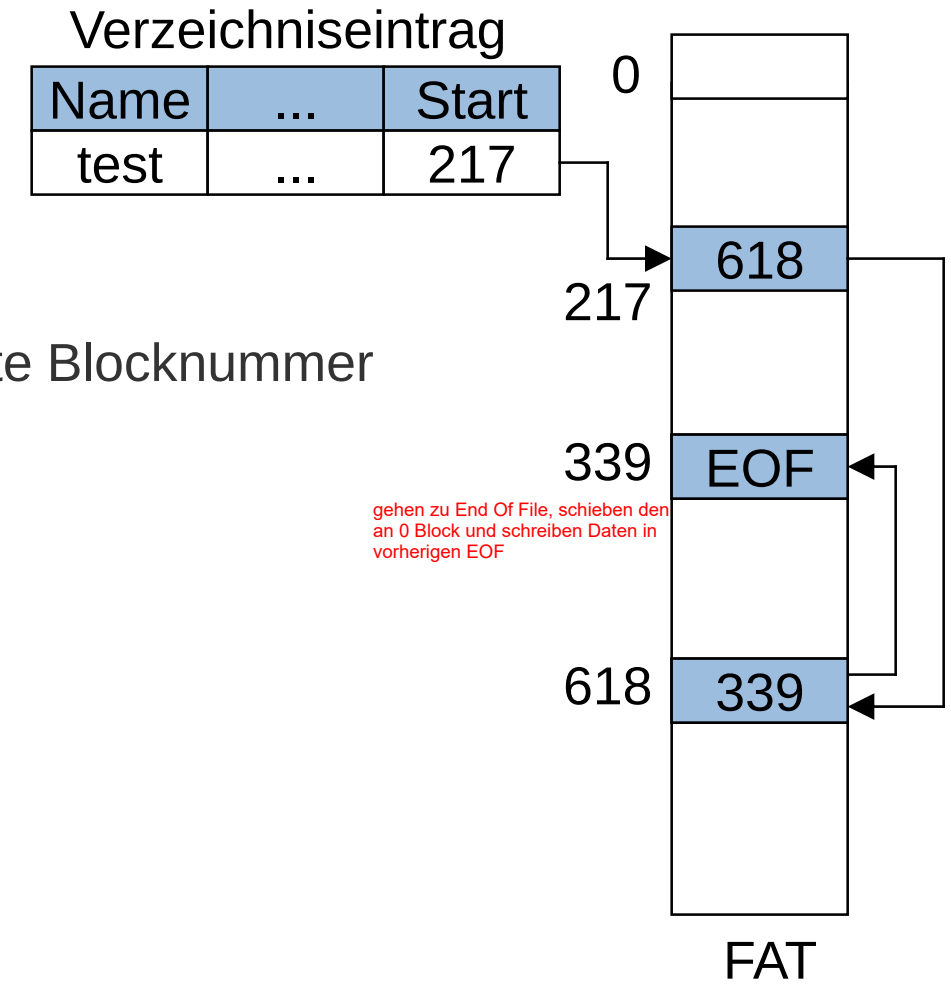
3.2 Verkettete Allokation

- Datei als verkettete Liste von Blöcken
- beliebige Anordnung der Blöcke einer Datei
- sequenzieller, aber kein wahlfreier Zugriff
- keine Platzverschwendung
- Speichern von Zeigern in Blöcken
- Bei beschädigtem Block geht ganze Datei verloren.



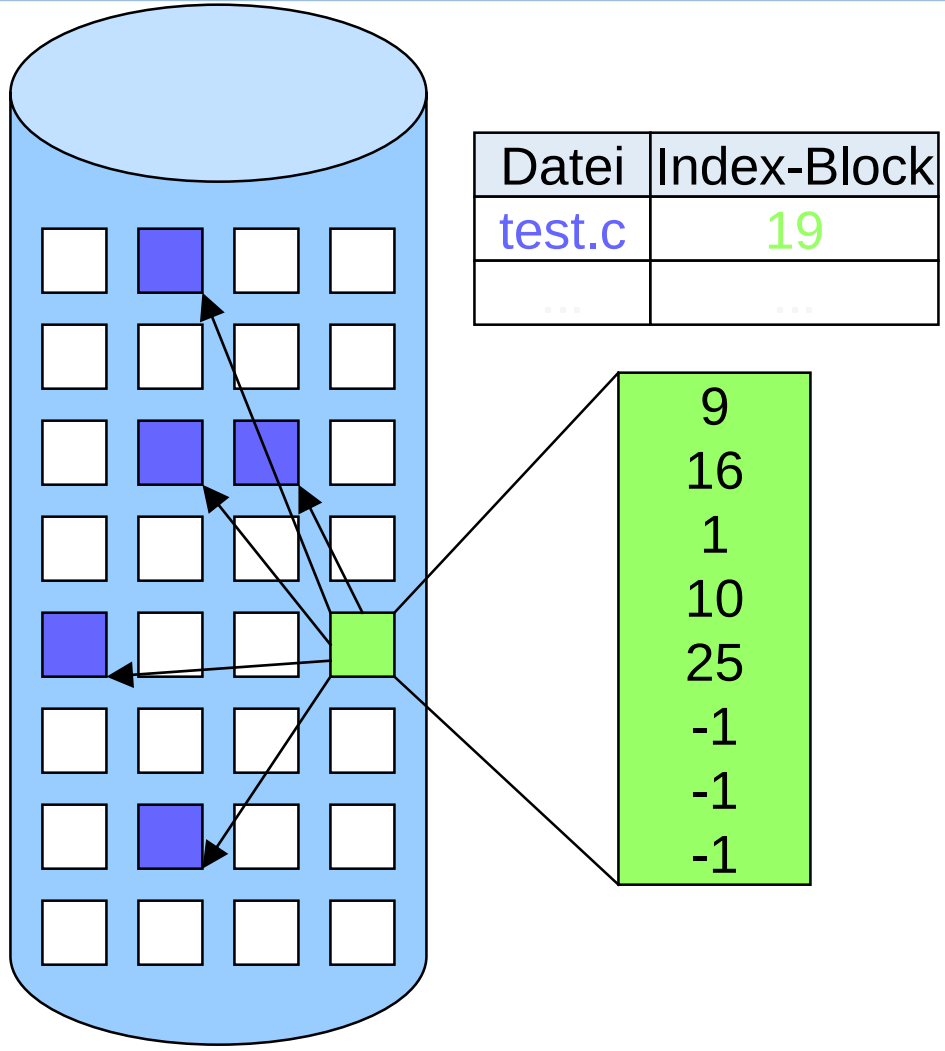
3.2.1 Beispiel: File Allocation Table (FAT)

- Variante der verketteten Allokation
- Unbenutzte Blöcke werden mit 0 markiert.
- Allokieren eines neuen Blocks:
  - Finden eines FAT-Eintrags mit Wert 0
  - Ersetzen des bisherigen Eintrags EOF durch allokierte Blocknummer
  - Neuer Eintrag wird mit EOF initialisiert.
- Caching der FAT
- Beispiel: MS-DOS, OS/2



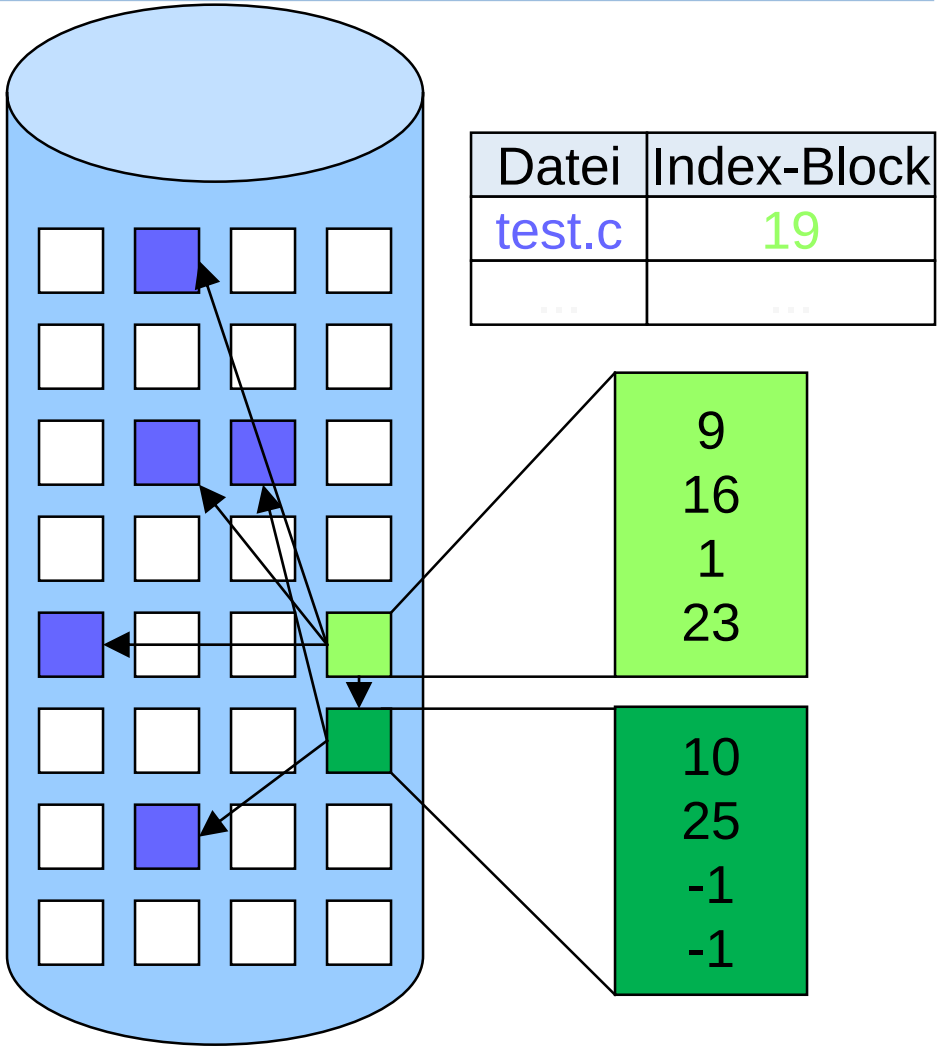
3.3 Indizierte Allokation

- alle Zeiger in einem Indexblock
- wahlfreier Zugriff
- kein externer Verschnitt
- Overhead durch Index-Block
- Problem: limitierte Dateigrösse bei 1 Index-Block
  - z.B. 1 Zeiger = 4 Bytes
  - 1 Block = 512 Bytes
  - 128 Zeiger
  - maximale Dateigrösse: 64 kB
- Lösungen:
  - Verketten von Index-Blöcken
  - Multilevel-Index
  - Kombination mehrerer Level
- häufig: Caching von Index-Blöcken



3.3.1 Verketten von Index-Blöcken

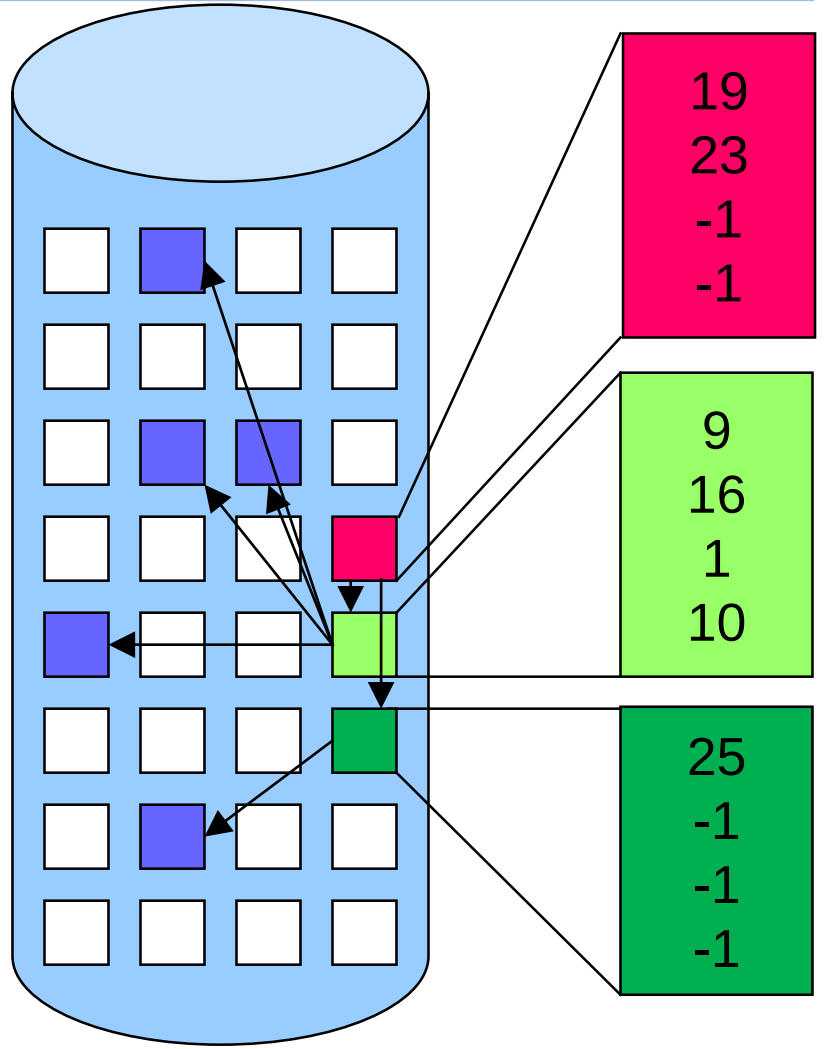
- Erste Einträge in einem Index-Block zeigen auf Datenblöcke.
- Letzter Eintrag in einem Index-Block zeigt auf nächsten Index-Block.
- Sequenzieller, aber kein wahlfreier Zugriff



3.3.2 Multilevel-Index

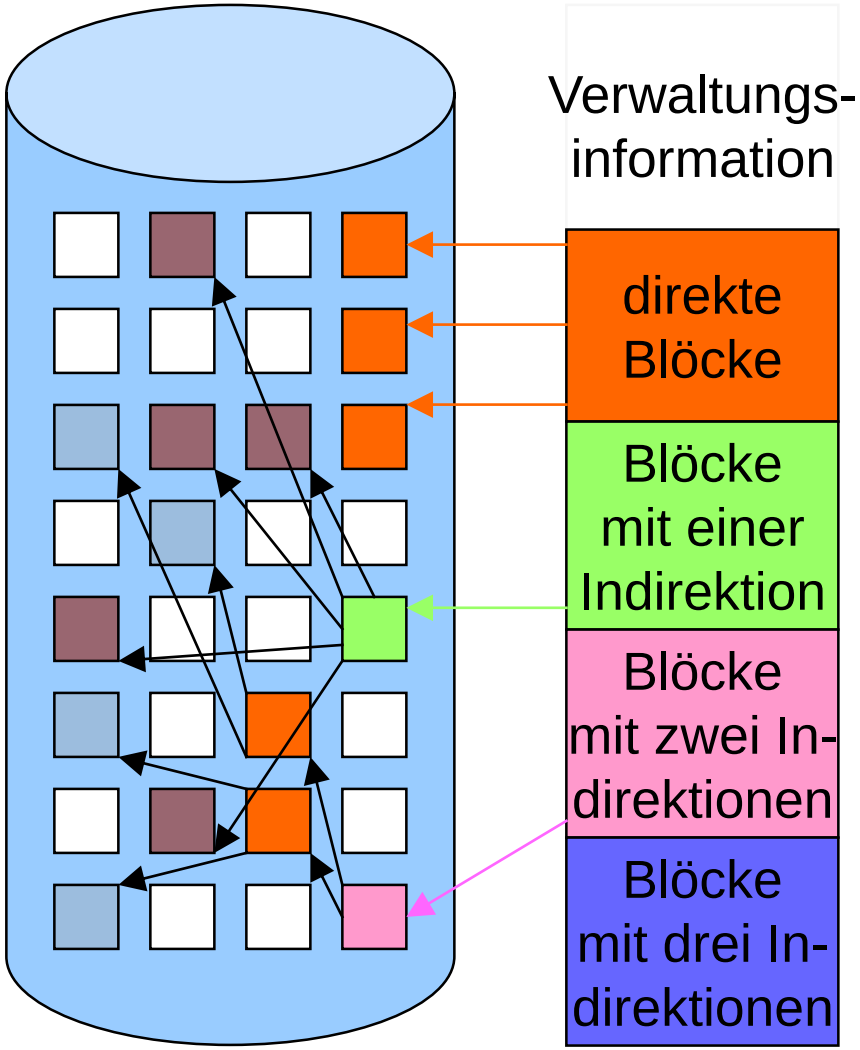
- Einführung mehrerer Stufen (Level)
- Zeiger im Index-Block der Stufe n auf Index-Blöcke der Stufe n+1
- Wahlfreier Zugriff, aber Durchlaufen mehrerer Stufen
- Beispiel: 2 Stufen
  - 128*128 = 16‘384 Zeiger
  - maximale Dateigrösse bei 512 Byte Blöcken: 8 MB

Datei	Index-Block
test.c	15
...	...



3.3.3 Kombination mehrerer Level

- Zugriff auf Datenblöcke über unterschiedliche Zahl von Stufen
- Direkter Zugriff für kleine Dateien
- Grosse Dateien erfordern mehrere Stufen
- Zeiger auf Stufen in Verwaltungsinformationen, z.B. Unix i-nodes



4. Freispeicherverwaltung

- Funktionen des Dateiorganisationsmoduls
  - Freispeicherverwaltung
  - Festplattenmanagement
- System muss Anforderungen nach Disk-Blöcken schnell erfüllen können, z.B. beim Erzeugen von Dateien.
- Ansätze
  - Freispeicherverwaltung mit Bitvektoren
  - Verkettete Freispeicherliste
  - Verkettete Freispeicherliste mit Zählen
  - Verkettete Freispeicherliste mit Gruppen

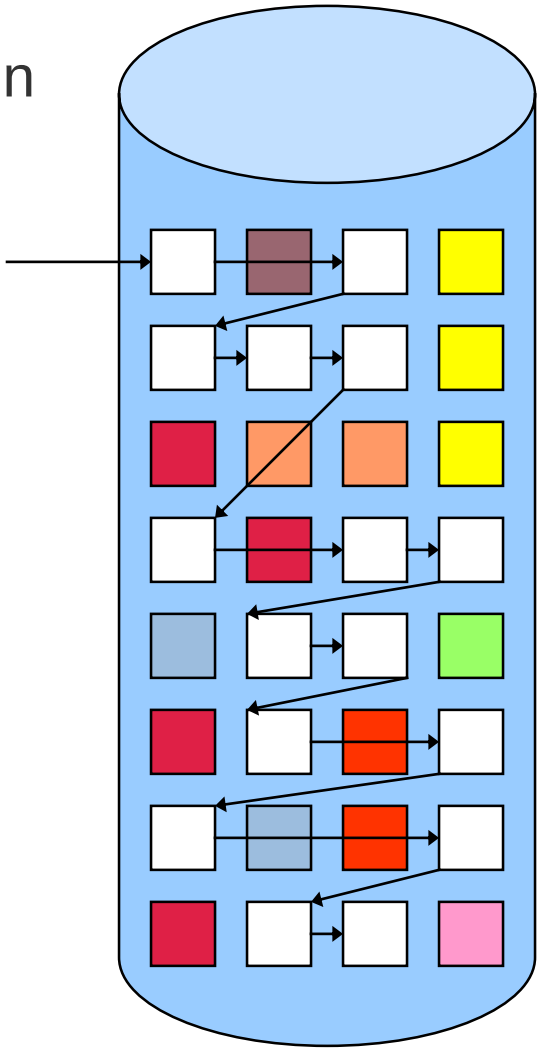
4.1 Bitvektoren

- bit[i] = 0: Block i frei
- bit[i] = 1: Block i belegt
- einfaches Schema um ersten freien Block und N zusammenhängende Blöcke zu finden
- Halten des gesamten Bitvektor im Speichers zur Leistungssteigerung, Zurückschreiben aus Robustheitsgründen
- Beispiel: 1 TB Festplatte, 4 kB pro Block $\rightarrow$ 256 MB für Freispeicherverwaltung

010110110100101
000111011010101
001011110101011
...

4.2 Verkettete Liste

- Ermitteln von N aufeinander folgenden Blöcken erfordert Durchlaufen von N Blöcken
- effizienter als Bitvektoren, da nur freie Blöcke gespeichert werden

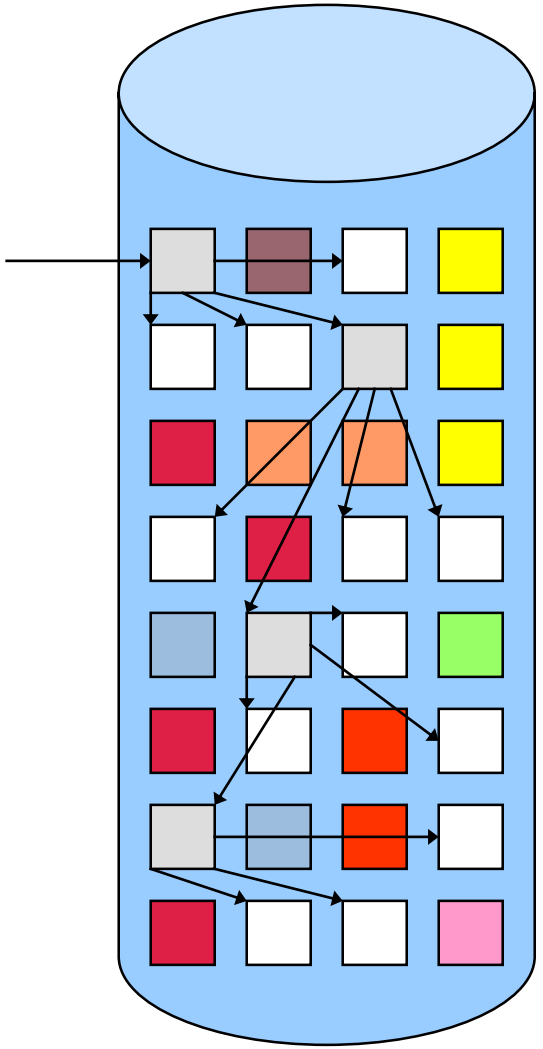


4.3 Gruppieren

- Speichern der ersten N freien Blöcke im ersten Block
- N-1 dieser Blöcke sind frei.
- Im N. Block sind weitere N freie Blöcke gespeichert
- usw.

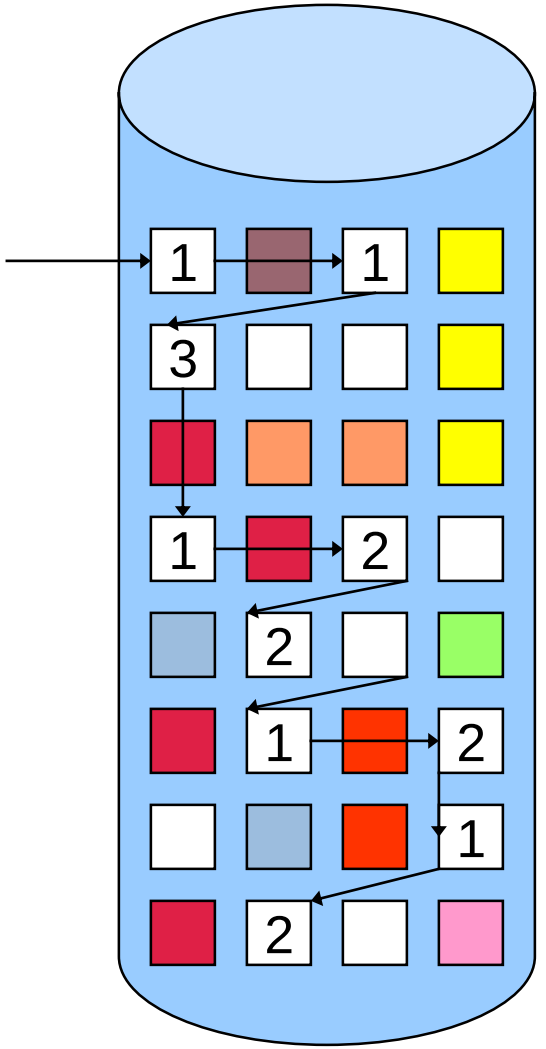
- schnelles Auffinden grosser Mengen von Blöcken

- Beispiel: N=4



4.4 Zählen

- verkettete Liste
- Speichern von Zeiger und Anzahl unmittelbar nachfolgender freier Blöcke



4.5 Space Maps

- Ziel: Unterstützung grosser Dateien mit vielen Blöcken
- Unterteilen von Partitionen in viele (mehrere 100) Metaslabs
- 1 Metaslab besitzt 1 Space Map.
- Space Map = Log (Zeit, Blockanzahl) von allen Blockaktivitäten (Allokationen und Freigaben, Benutzung von Zählen)
  → Log-Structured File Systems
- Laden der Space Map in den Hauptspeicher bei Allokationen oder Freigaben; Aufbau einer effizienten Datenstruktur, z.B. Baum
- Komprimieren der Space Map und Zurückspeichern nach Aktivität
- Beispiel: ZFS

5. Zuverlässigkeit

UNIVERSITÄT BERN

> Inkonsistenzen durch Systemabstürze oder Festplattenfehler

→ Konsistenzprüfung

> Datenverlust durch defekte Festplatten

→ Backup

> Systemabstürze

→ Journaling File Systems

5.1 Konsistenzprüfung

- Verzeichnisinformationen im Hauptspeicher
  → Inkonsistenzen bei Systemabsturz
- Konsistenzprüfung
  - Vergleich der Verzeichnisinformationen mit den auf der Disk gespeicherten Dateien, z.B.:
    - Auffinden von Blöcken, die weder in der Freispeicherliste noch in Dateien enthalten sind
    - Auffinden von Blöcken, die in einer Datei und in der Freispeicherliste enthalten sind
  - Versuch Inkonsistenzen zu beheben
  - Beispiel-Werkzeuge: scandisk, fsck

5.2 Backup

- Backup-Programme zum Speichern und Laden von Dateien auf anderen Speichermedien (z.B. Bänder, Disketten)
- Typische Backup-Strategie
  1. Vollständiges Backup
     - Kopieren aller Daten auf die Disk
  2. Inkrementelles Backup relativ zu 1.
     - Kopieren der Daten mit Änderungen seit 1.
  3. Inkrementelles Backup relativ zu 2.

  ...

  N. Inkrementelles Backup relativ zu N-1.

  N+1. Gehe zu 1.
- Dumps
  - physikalisch: einfach, schnell ineffizient (ungenutzte und fehlerhafte Blöcke)
  - logisch: Beginn mit spezifiziertem Verzeichnis, rekursiv

5.3 Log-Structured File Systems (LFS)

- Motivation
  - immer schnellere Prozessoren und grösserer Hauptspeicher (und damit Disk-Caches), bei eher gleichbleibender Geschwindigkeit und Kapazität von Festplatten
  - Lesezugriffe können meist durch Caching unterstützt werden, so dass die meisten Disk-Zugriffe durch Schreiboperationen entstehen.
  - Geringer Leistungsgewinn, falls Schreiboperationen auf Disk zurück geschrieben werden müssen
  - Geringe Leistung bei Schreiboperationen mit geringer Datenmenge (z.B. i-nodes)
- Konzept LFS
  - Struktur der Disk als Log
  - Sammeln von ausstehenden Schreiboperationen in Hauptspeicherpuffer
  - Periodisches Zurückschreiben in einem zusammenhängenden Segment (enthält i-nodes, Verzeichnis- und Datenblöcke)
  - i-nodes map zu deren Auffinden (auf Disk und im Cache)
  - Cleaner zum Auffinden und Löschen veralteter Daten

5.4 Journaling File Systems (JFS)

- Problem: LFS sind mit existierenden Systemen nicht kompatibel.
- Journaling File Systems (JFS, z.B. NTFS) verwenden Logs, um durchzuführende Aktionen zu beschreiben und bei Absturz diese zu wiederholen bzw. Inkonsistenzen zu vermeiden.
- Beispiel: Schritte beim Löschen einer Datei
  1. Lösche Datei aus Verzeichnis
  2. Freigabe des dazugehörenden i-nodes
  3. Freigabe der Diskblöcke der Datei
- JFS
  - schreibt Aktionen in Log-File auf Disk
  - führt Aktionen aus
  - löscht Aktionen aus Log-File
- Bei einem Absturz kann das System pendente Aktionen erkennen und ggf. wiederholen um Inkonsistenzen aufzuheben.