# 10. Sekundärspeicher

**Michael Schöttner**

Betriebssysteme und Systemprogrammierung

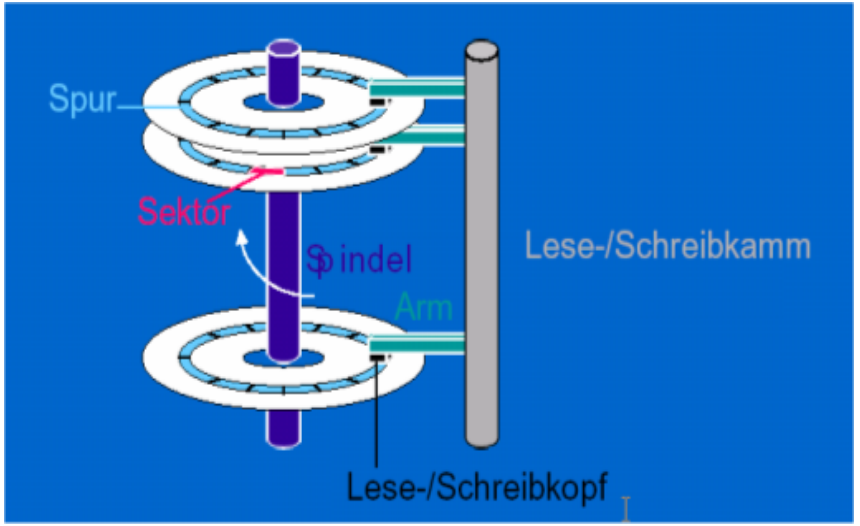
HEINRICH HEINE
UNIVERSITÄT DÜSSELDORF

# 10.1 Vorschau

- Festplatten & SSDs
- Disk-Scheduling
- Partitionen
- Freispeicherverwaltung
- Speicherallokation für Dateien

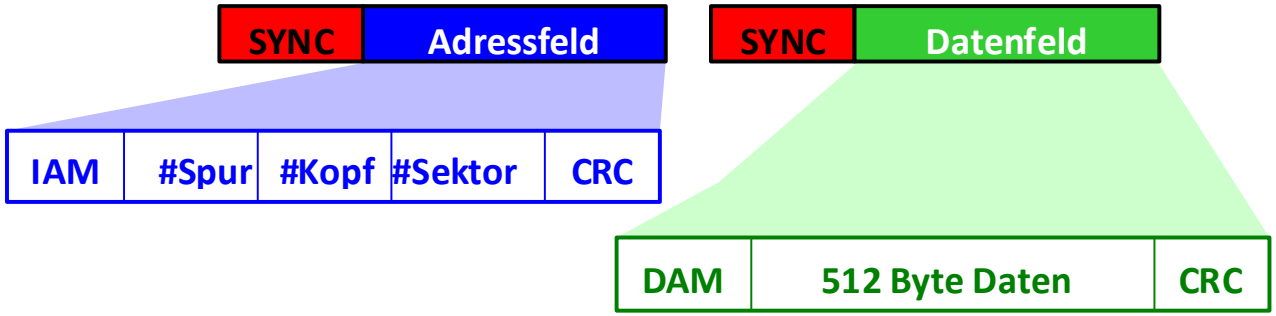
# 10.2 Festplatten

- Staubdicht versiegelt.
- Eine oder mehrere rotierende Platten:
  - 5.400 - 15.000 U/min., magnetisierbare Schicht
  - unterteilt in konzentrische Spuren (Tracks); Spuren wiederum in Sektoren unterteilt
  - Zylinder = Gruppe übereinander liegender Spuren
- Beweglicher Kamm/Arm:
  - Mit Schreib-/Leseköpfen
  - Schweben dicht über Magnetschicht
  - Positionierungszeit ca. 10 ms
- Sektoren à 512 Bytes



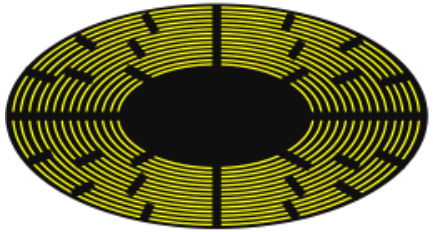
# Aufbau eines Sektors

- 512 Bytes pro Sektor sind üblich
- Entstehen durch physikalische bzw. Low-Level-Formatierung
- Sektor-Aufbau bei alten MFM-Disks:
  - IAM: Index Address Mark → markiert Adressfeld
  - DAM: Data Address Mark → markiert Datenfeld
  - SYNC: abhängig vom Aufzeichnungsverfahren (Lücke zwischen Index-Record & Datenrecord)



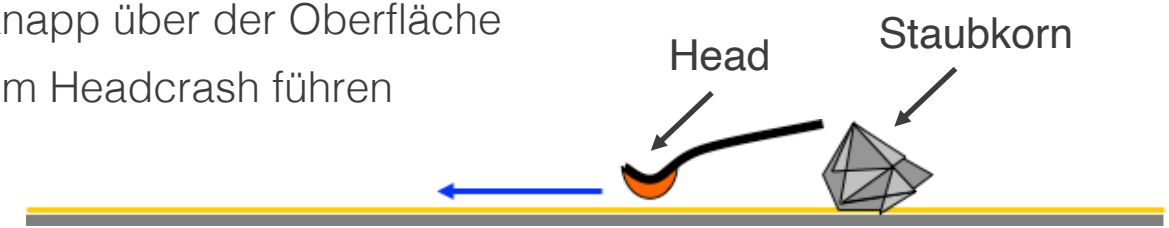
# Adressierung

- Früher CHS-Adressierung (**C**ylinder, **H**ead und **S**ector)
- Abgelöst durch Logical Block Addressing (LBA)
  - Hier sind die Sektoren = Blöcke fortlaufend durchnummeriert
  - Die LBA-Nummerierung wird im Disk-Controller umgesetzt
  - Lineare Zugriffe (fortlaufende Block-Nummern) sind am schnellsten
    - Die Nummerierung ist fortlaufend innerhalb eines Zylinders und dann geht es beim nächsten Zylinder weiter
    - Bei einem linearen Zugriff sind somit die Chancen hoch innerhalb eines Zylinders zu arbeiten, wodurch die teuren Kammbewegungen minimiert werden
- Äußere Spuren bieten Platz für mehr Sektoren →

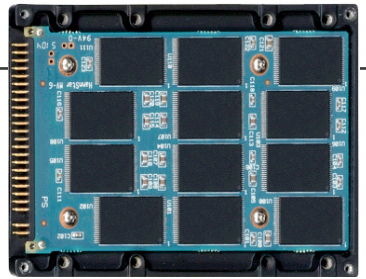


# Fehlerbehandlung

- Sector Forwarding:
  - Controller hat Liste „schlechter“ Blöcke und leitet Zugriffsversuch auf fehlerhaften Block (unsichtbar für Treiber) auf einen Reserveblock um
  - Reserveblöcke pro Zylinder oder in Reservezylinder
  - Lineare Zugriffe werden dadurch „unterbrochen“
- Prüfsumme und Forward Error Correction (FEC) für jeden Sektor.
- Disks sind fehleranfällig
  - teilweise fehlerhafte Blöcke bei Auslieferung
  - Köpfe schweben knapp über der Oberfläche
  - Staubkorn kann zum Headcrash führen



# 10.3 Solid State Drive (SSD)



- Schneller wahlfreier Zugriff
  - Sehr hohe Datenraten: bis ca. 2 GB/s lesen & schreiben
  - Geringe Latenz: 100-200µs
  - Teurer als Festplatten
- Schreiben etwas aufwändiger als Lesen:
  - Erst Erase-Block auslesen, dann löschen, dann schreiben
  - Erase-Block i.d.R. 256 - 512 KB
- Zellen nur beschränkt oft beschreibbar
  - Je nach Typ 10.000 - 100.000 Mal
  - Wear Leveling (Firmware oder OS) sorgt dafür, dass alle Zellen gleichmäßig oft beschrieben werden
- Vorteile gegenüber Festplatten: schneller, zuverlässiger (keine Mechanik)

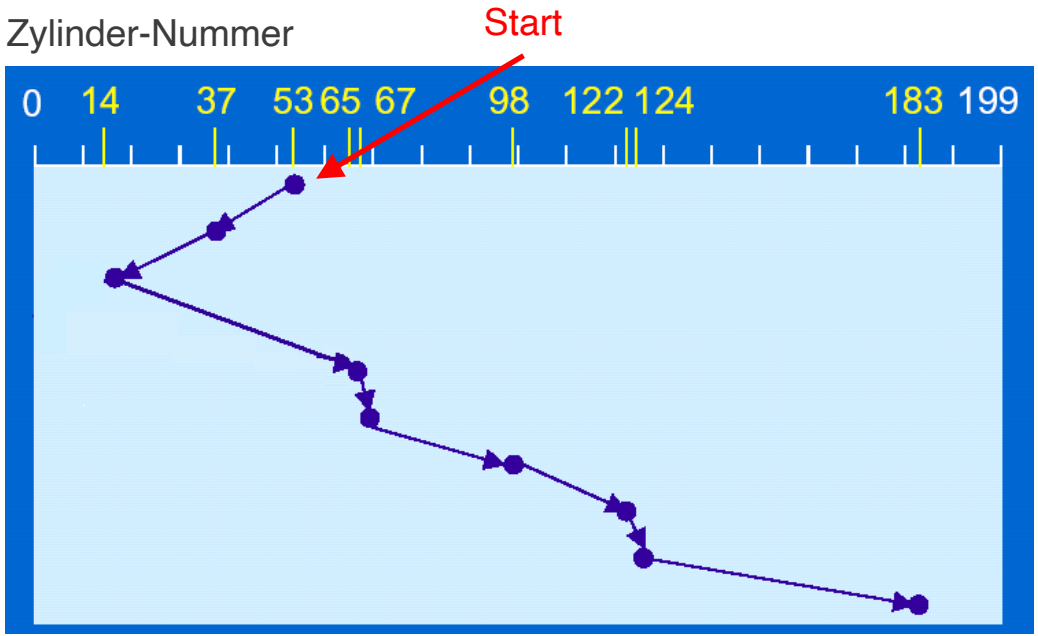
# 10.4 Festplatten-Scheduling

- Armpositionierung ist bei Festplatten langsam
  - Der Arm muss beschleunigt, bewegt und abgebremst werden
  - Dann muss gewartet werden bis der richtige Block/Sektor unter dem Kopf liegt (dies hängt von der Rotationsgeschwindigkeit ab)

- Betriebssystem verwendet für die Festplattenzugriffe einen Disk-Scheduler:
  - Dieser sammelt die Block-Zugriffe aller Prozesse in einem definierten Zeitintervall in einer Queue, re-organisiert dann die Reihenfolge (sofern keine Daten-Abhängigkeiten vorliegen) und legt somit die Reihenfolge der Block-Zugriffe fest
  - Evt. auch für Power-Management (Festplatte erst einschalten, wenn mehrere Zugriffsaufträge vorhanden sind)
  - Festplatten-Scheduling kann der Treiber abgeschalten und selbst übernehmen

# Strategie: SCAN

- Auch Fahrstuhlstrategie (engl. elevator seek) genannt.
- Beispiel: Block-Sequenz: 98, 183, 37, 122, 14, 124, 65, 67
  - Start bei Zylinder 53
  - Resultiert in Kopfbewegung über 208 Zylinder
- Diese Strategie ist vor allem bei starker Last vorteilhaft

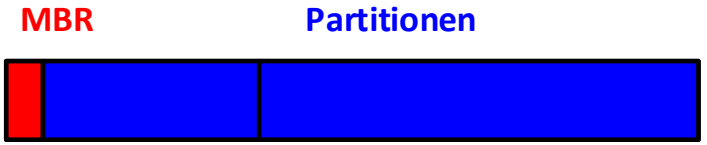


# Bemerkungen

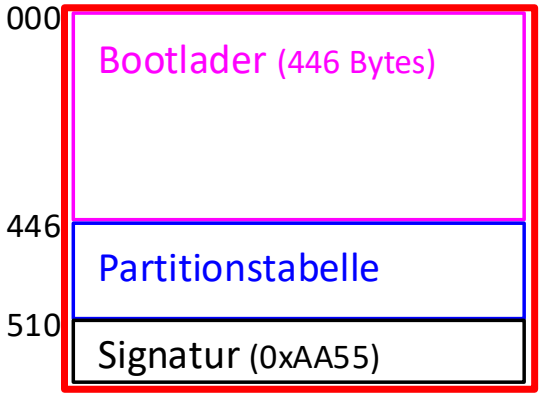
- Es gibt noch weitere Strategien für Festplatten-Scheduling
- Für SSDs sind diese nicht relevant
- Festplatten sind aber im Backend-Storage in Rechenzentren inkl. Cloud weiterhin wichtig, wegen der sehr großen und günstigen Speicherkapazität
- Hierfür ist Festplatten-Scheduling wichtig

# 10.5 Partitionen bei PCs

- Festplatten werden oft in eine/mehrere Partitionen (Bereiche) unterteilt:
  - für verschiedene Dateisysteme auf einer Festplatte
  - mehrere Betriebssysteme auf einer Disk
  - Trennung von System & Benutzerdaten
  - Besonderheit in Linux: Swap-Partition




- Master Boot Record (MBR):
  - Nur ein Mal pro Disk immer in Block 0
  - Wird durch BIOS beim Einschalten an Adr. 0x7C00 geladen und der Bootlader-Code angesprungen
  - Bootlader lädt BS oder Boot-Manager aus aktiven Partition (siehe später)



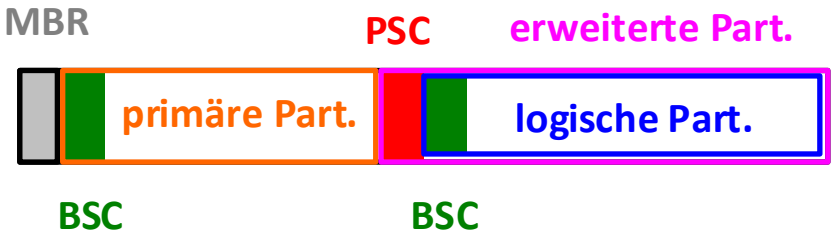
# Partitionen

- Eine **Partitionstabelle** beschreibt max. vier Partitionen, die die Disk in unabhängig voneinander nutzbare Bereiche unterteilen.
- Primäre Partitionen:
  - Nur in der Partitionstabelle des Master Boot Records
  - max. vier Stück pro Disk bei PCs
  - beginnen mit Bootsektor (BSC)
- Erweiterte Partitionen:
  - Container für logische Unterpartitionen
  - beginnen mit Partitionssektor (PSC)
  - PSC ist ähnlich wie MBR aufgebaut, aber nur Partitionstabelle genutzt (enthält keinen Bootcode)

# Partitionen

- Logische Partitionen:
  - einer erweiterten Partition zugeordnet
  - beginnen mit Bootsektor (BSC)
  - beinhalten keine Partitionstabelle

- Beispiel: 1 primäre und 1 logische Partition



# Aufbau eines Partitionseintrags in der Partitionstabelle

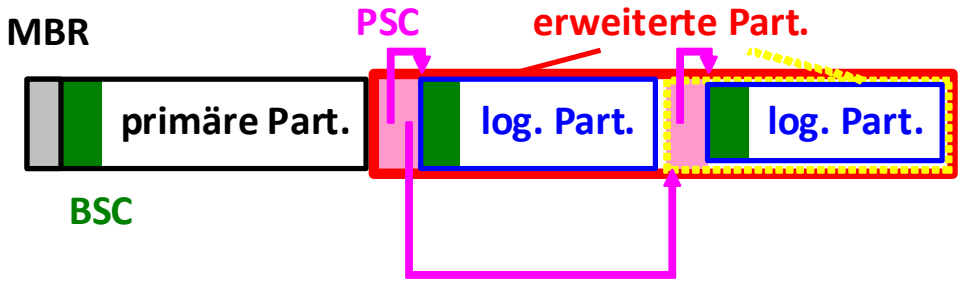
| Offset | Größe | Inhalt |
|---|---|---|
| 0 | 1 | Boot – Flag (80h aktiv; 00h inaktiv) |
| 1 | 3 | ~~Beginn der Partition (CHS)~~ // veraltet |
| 4 | 1 | System Kennung:<br><br>z.B. 0: Eintrag frei, 1: DOS 12-Bit FAT, 4: DOS 16-Bit FAT …<br>5: erweiterte Partition, F: erweiterte Partition<br>... |
| 5 | 3 | ~~Ende der Partition (CHS)~~ // veraltet |
| 8 | 4 | **Entfernung des ersten Blocks der Partition (Anzahl Blöcke):**<br>- vom MBR bei primären Partitionen<br>- vom PSC bei erweiterten Partitionen |
| 12 | 4 | **Größe der Partition in Blöcke** |

- Bem.: Nur eine Partition pro Festplatte kann als aktiv markiert werden.

# Partitionen

- Eine erweiterte Partition nutz max. zwei Einträge in der Partitionstabelle:
  - 1. Eintrag beschreibt ein logisches Laufwerk
  - 2. Eintrag verweist auf eine erweiterte Partition
  - Somit entsteht bei Bedarf eine Kette erweiterter Partitionen
    - Die 1. Partition ist so groß, sodass sie alle geschachtelten Partitionen einschließt.

- Beispiel: 1 primäre und zwei logische Partitionen



# Aufbau eines Partitionseintrags in der Partitionstabelle

- Ergänzung / Besonderheit: „Entfernung des ersten Blocks der Partition“
  - in primären Partitionen immer bezogen auf physischen Anfang der Disk (MBR)
  - alle erweiterten Partitionen beziehen sich auf die physische Adresse des Ankers der PSC-Kette
  - alle logischen Partitionen beziehen sich auf ihren zugehörigen PSC

- Bemerkung zu Partitionierungswerkzeugen:
  - zeigen Schachtelung von erweiterter Partition nicht an

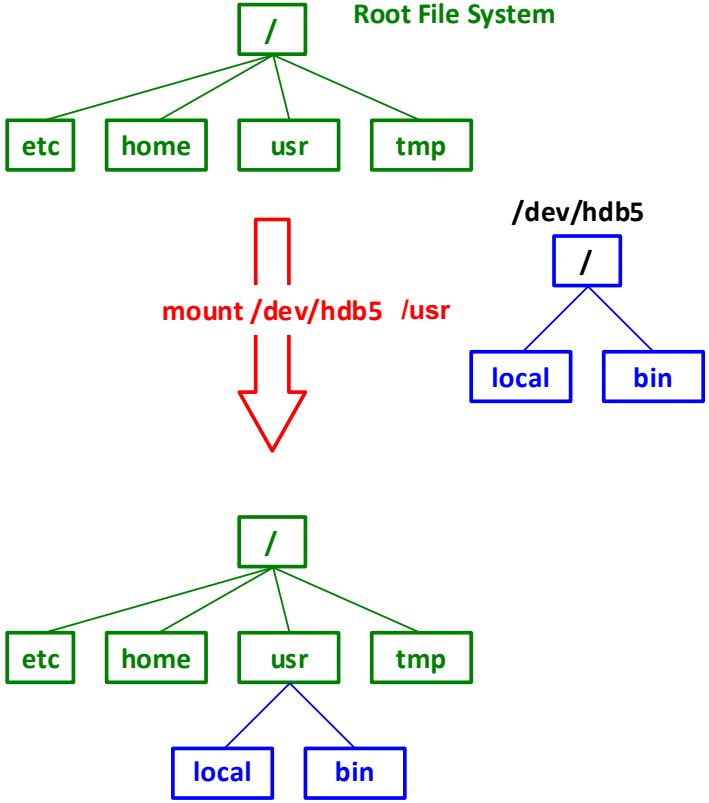
# Namensgebung von Partitionen

- UNIX: interner Name: z.B. **`/dev/hda2`**
  - Disk-Volumes: a, b, c, d
  - Partitionen: primäre = 1 - 4; log.: >= 5

- Microsoft Windows: Laufwerksbuchstaben **`c:`**, **`d:`**, ...
  - zuerst primäre Partitionen von allen Disks
  - dann logische Partitionen aller Disks
  - ab NT frei umbenennbar

# Mounting von Partitionen

- „Montieren“ einer Partition in einem Verzeichnis des Dateisystems.
- Auch unterschiedliche (verteilte) Dateisysteme können in einem Dateibaum vereint werden.
- UNIX: `mount` und `umount` Befehle.



# 10.6 UEFI

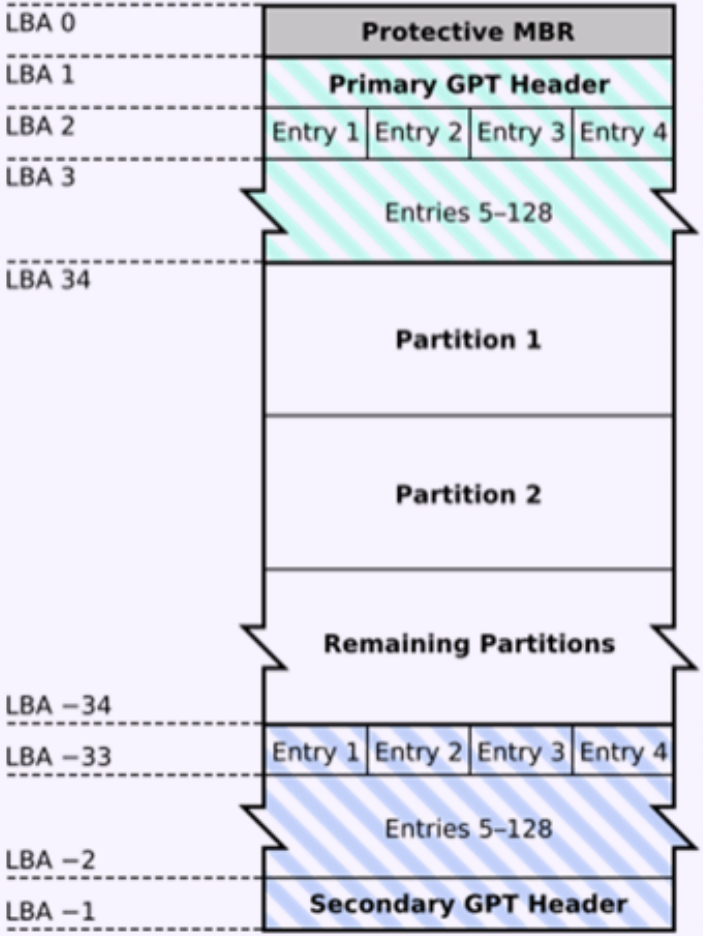
- = Unified Extensible Firmware Interface (2005) → ersetzt BIOS
- Software Interface zwischen Betriebssystem und Hardware
  - 32-Bit oder 64-Bit Code statt 16-Bit Real-Mode des BIOS
  - Device Drivers für Pre-boot Environment (auch Netzwerkstack)
  - Bootmanager, Disk Support: Unterstützung der GUID Partition Tabelle,
  - Dateisystem Support (FAT32), textuelle und graphische Konsole
  - Erweiterungen: können von persistenten Speichern geladen und installiert werden
  - Unterstützung von Pre-Boot Applikationen
- Weitere Infos: http://software.intel.com/en-us/articles/about-uefi/

# 10.6 UEFI

- Intel unterstützt ab 2020 nur noch Mainboards mit UEFI
- Das alte PC-BIOS bietet sehr viele Funktionen für den HW-Zugriff (Disk/SSD, Uhr, Textbildschirm, Tastatur, Maus sowie VGA-Grafik).
- Diese Funktionen sind jedoch alle im Real-Mode (16-Bit Code) programmiert
- Diese werden bisher allenfalls für den Boot-Vorgang genutzt und sind danach hinfällig

# GUID Partition Table (GPT)

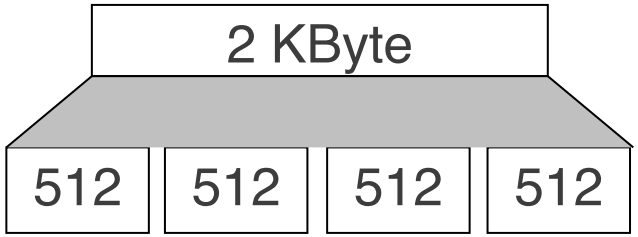
- Moderne Partitionierungstechnik, eingeführt mit UEFI
  - MBR Eintrag nur noch für Kompatibilität
- Primary GPT Header (LBA 1)
  - Partitionseinträge (max. 128)
  - GUID (Globally Unique Identifier) für Identifikation des Dateisystems
  - CRC32 Prüfsumme für GPT Header
- Secondary GPT Header:
  - Sicherheitskopie am Ende der Disk




https://de.wikipedia.org/wiki/GUID_Partition_Table#/media/File:GUID_Partition_Table_Scheme.svg

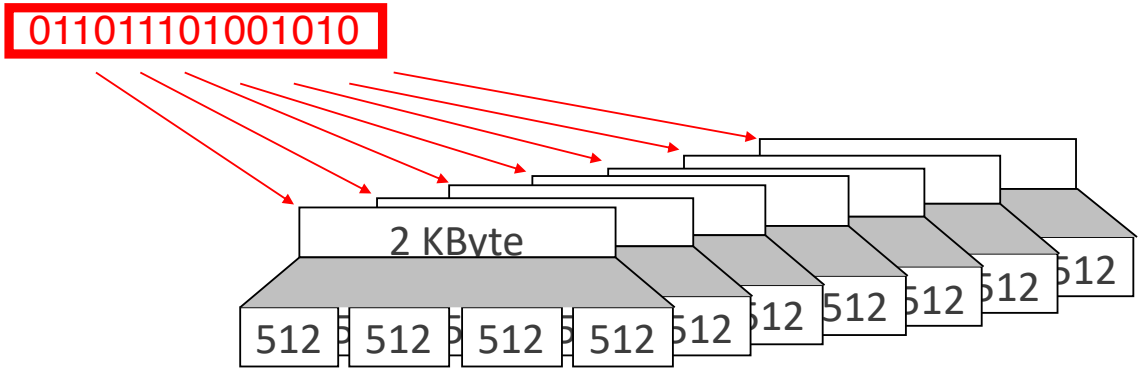
# 10.7 Freispeicherverwaltung

- Sequentielle Natur der Sekundärspeicher erfordert andere Verfahren als im RAM
- Vergabeeinheit:  Block (UNIX) oder Cluster (Microsoft Windows)
  - festgelegt für eine bestimmte Partition,
  - nicht zu groß wählen, da sonst interne Fragmentierung bei kleinen Dateien
  - nicht zu klein wählen, sonst zu viel Verwaltungsaufwand
  - typische Werte: 0,5 – 32 KB (je nach Disk-Partition)
- Die Vergabeeinheit ist ein Vielfaches der Sektorgröße (512 Byte), im Bild 2 KB



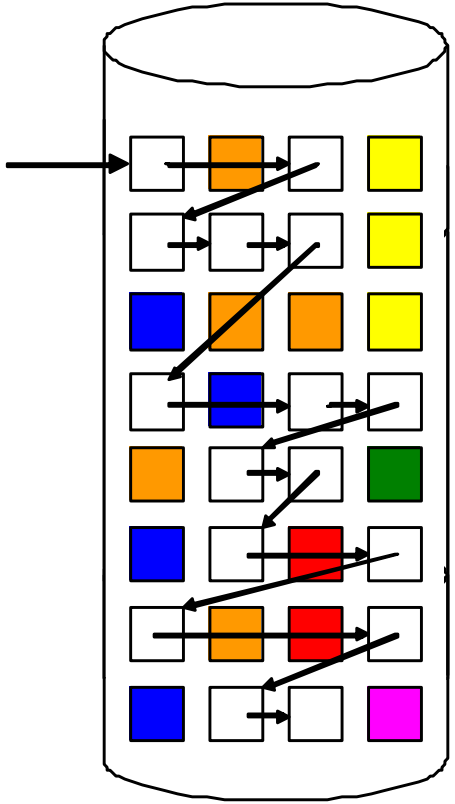
# Bitvektor / Bitmap

- Partition in Blöcke fester Größe unterteilen.
- Jeweils ein Bit zeigt an, ob ein Block frei ist od. nicht (0=belegt,1=frei).
- Beispiel: 512 GB Disk, 4 KB pro Block → 16 MB für Bitmap.
- Bewertung:
  - fortlaufende Blöcke (Vergabe-Einheiten) vgl. einfach zu finden (Bitmap i.d.R. im Hauptspeicher)



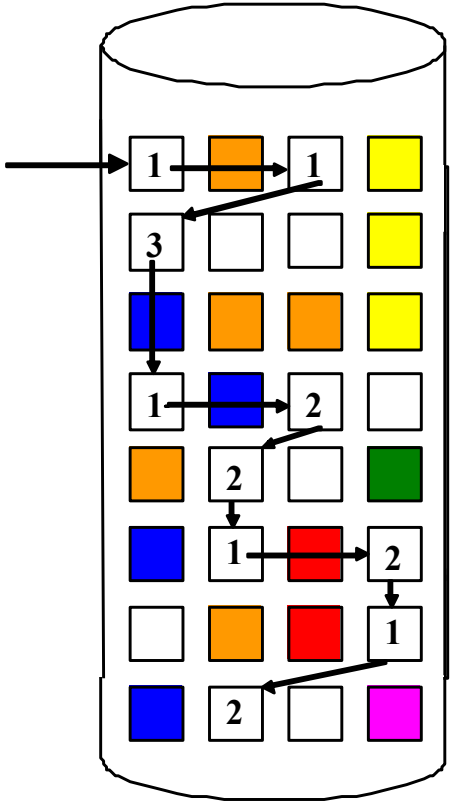
# Verkettete Freispeicherliste

- Finden von N zusammenhängenden Blöcken schwieriger.
- Ermitteln N aufeinanderfolgender Blöcke erfordert das Durchlaufen von mindestens N Blöcken.
- Gegebenenfalls speichereffizienter als Bitmap, da nur freie Blöcke verkettet werden.
- Bemerkung:
  - freie Blöcke → weiß
  - belegte Blöcke → andere Farben



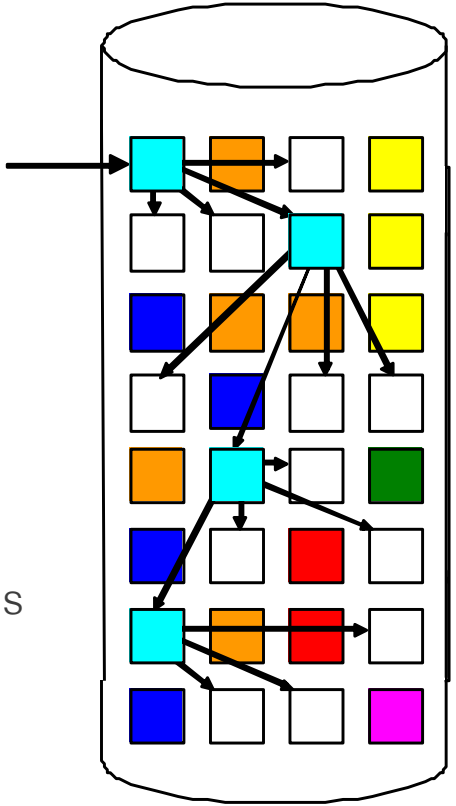
# Freispeicherverwaltung mit Zähler für „Run-Length“

- Verkettete Liste und zusätzlich Speichern von Zeiger und Anzahl unmittelbar nachfolgender freier Blöcke in einem freien Block.
- Vereinfacht die Suche nach N aufeinanderfolgenden Blöcken.



# Freispeicherverwaltung mit Gruppieren

- Speichern der ersten N freien Blöcke im ersten Block.
  - Im N-ten Block sind weitere N freie Blöcke gespeichert - usw.
- Beispiel: N=4.



- Bemerkung:
  - Freie Blöcke → weiß
  - Freie Blöcke mit Zeigern auf weitere freie Blöcke → türkis
  - Belegte Blöcke → andere Farben

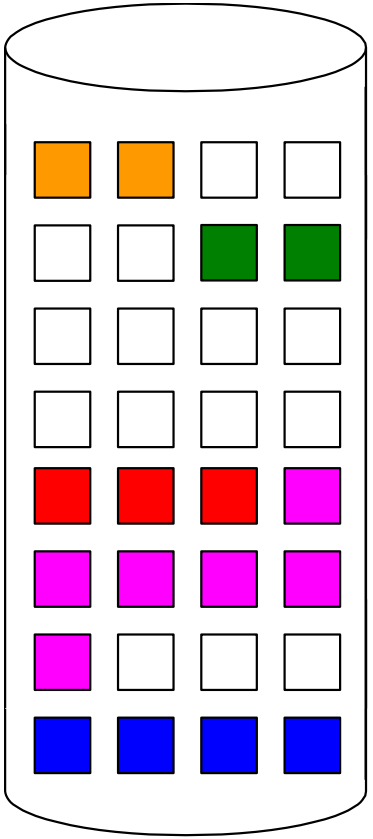
# 10.8 Speicherallokation für Dateien

- Ausgangssituation:
  - Blöcke werden Dateien meist schrittweise zugeordnet
  - I.d.R. viele offene Dateien auf einer Festplatte
- Ziele:
  - effektive Ausnutzung des Sekundärspeichers
  - kleine und sehr große Dateien müssen möglich sein
  - schneller Dateizugriff ist wichtig

# Zusammenhängende Allozierung

- Jede Datei belegt zusammenhängende Blöcke
  → schneller Dateizugriff
- Externe Fragmentierung der Festplatte
- Expansion der Dateien schwierig

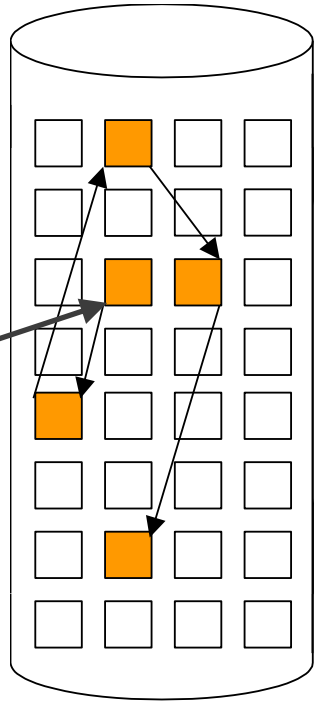
| Datei | Start | Länge |
|---|---|---|
| Test.java | 0 | 2 |
| .profile | 6 | 2 |
| Plan | 16 | 3 |
| News | 19 | 6 |
| Mail | 28 | 4 |



# Verkettete Allokation

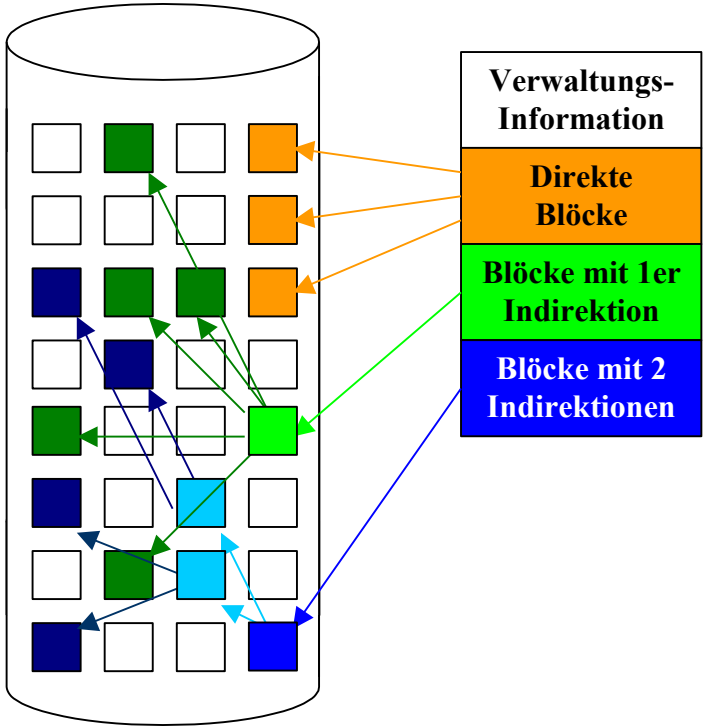
- Datei als verkettete Liste von Blöcken.
  - Beliebige Anordnung der Blöcke einer Datei
    - Expansion von Dateien einfach
    - Keine Platzverschwendung
  - Aber Zugriffsgeschwindigkeit abhängig von Zersplitterung
- Speichern von Zeigern in Blöcken.
  - Bei einem beschädigten Block geht evt. ganze Datei verloren
    → Alternativ Verkettung auslagern in separate Metadaten

| Datei | Start | Länge |
|---|---|---|
| Test.java | 9 | 250 |
| … | … | … |



# Multilevel-Index Allokation

- Feste Größe für Verwaltungsdatenblock
  → einfach zu implementieren
  → keine Probleme mit ext. Fragmentierung
- Realisierung
  - Feste Anzahl direkter Zeiger auf Datenblöcke (reicht für kleine Dateien)
  - Zusätzlich feste Anzahl von Zeigern auf Zeigerblöcke
    - Normale Datenblöcke die erst bei Bedarf alloziert werden, um mehr Zeiger speichern zu können
  - Metadaten wachsen somit erst bei Bedarf
    → speichereffizient



# Multilevel-Index Allokation

- Rechenbeispiel: maximale Dateigröße bei gegebenen Parametern
  - Blockgröße = 1 KB
  - Zeigergröße = 4 Bytes
  - Im Verwaltungsblock
    - 12 direkte Zeiger
    - 1 einfach indirekter Zeiger
    - 1 zweifach indirekter Zeiger
- Maximale Dateigröße
  = (12 * 1 KB) + (256 * 1 KB) + ($256^2$ * 1 KB) = = 64 MB

# 10.9 Caching

- Betriebssystem puffert übertragene Blocks im Block-/Page-Cache im Hauptspeicher → falls möglich, bei Bedarf gesamten Hauptspeicher nutzten
- Sekundärspeicher haben in der Regel noch einen eingebauten Cache (auch SSDs)
  - Neben dem gesuchten Block werden mehrere Blöcke sequentiell gelesen