Geräteverwaltung (1) / Dateiverwaltung (3)

Ute Bormann, TI2

2023-10-13

Betriebssysteme

- Prozessverwaltung
- Speicherverwaltung
- Dateiverwaltung

⇒ • Geräteverwaltung

- Prozessverwaltung
  ⇒ Nebenläufigkeit ⇒ Kommunikation

Inhalt

1. Organisation einer Platte
2. Zugriffsalgorithmen auf Plattenblöcke
3. Kleine Exkurse
4. Konsistenzprüfung eines Dateisystems
5. Moderne Dateisysteme

Teil 1:
Organisation einer Platte

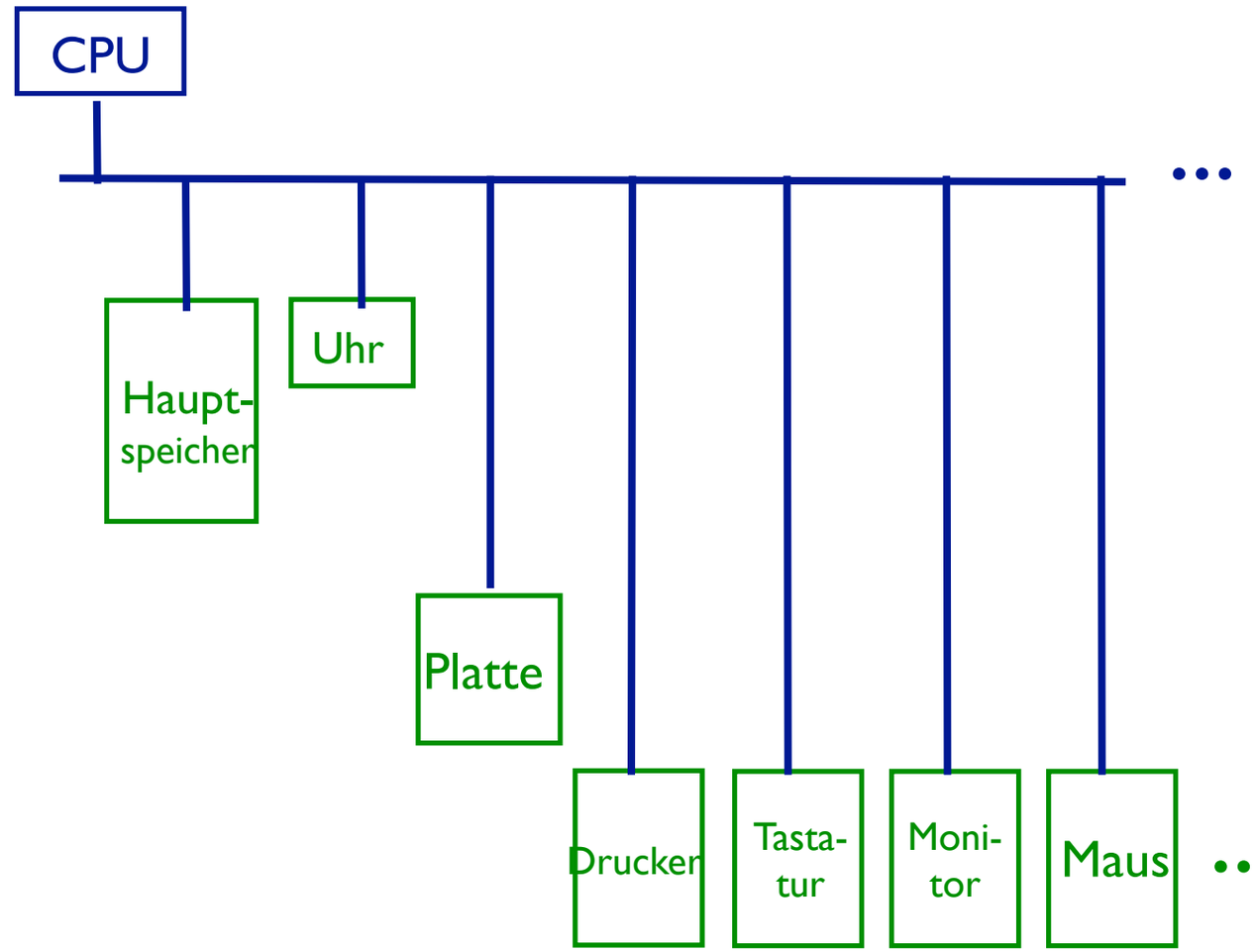
12 Geräteverwaltung (1), Dateiverwaltung (3), Ute Bormann

Anschluss von Geräten (vereinfacht)

- Klassischer Rechneraufbau:
  CPU über Bus mit Vielzahl von „Geräten“ verbunden

  ⇒ (in Grenzen) eigenständige Arbeit der Geräte

- Mehrere Geräte desselben Typs möglich
- U.U. mehrere Busabschnitte unterschiedlicher Länge
- Ansteuerung über Geräteadressen



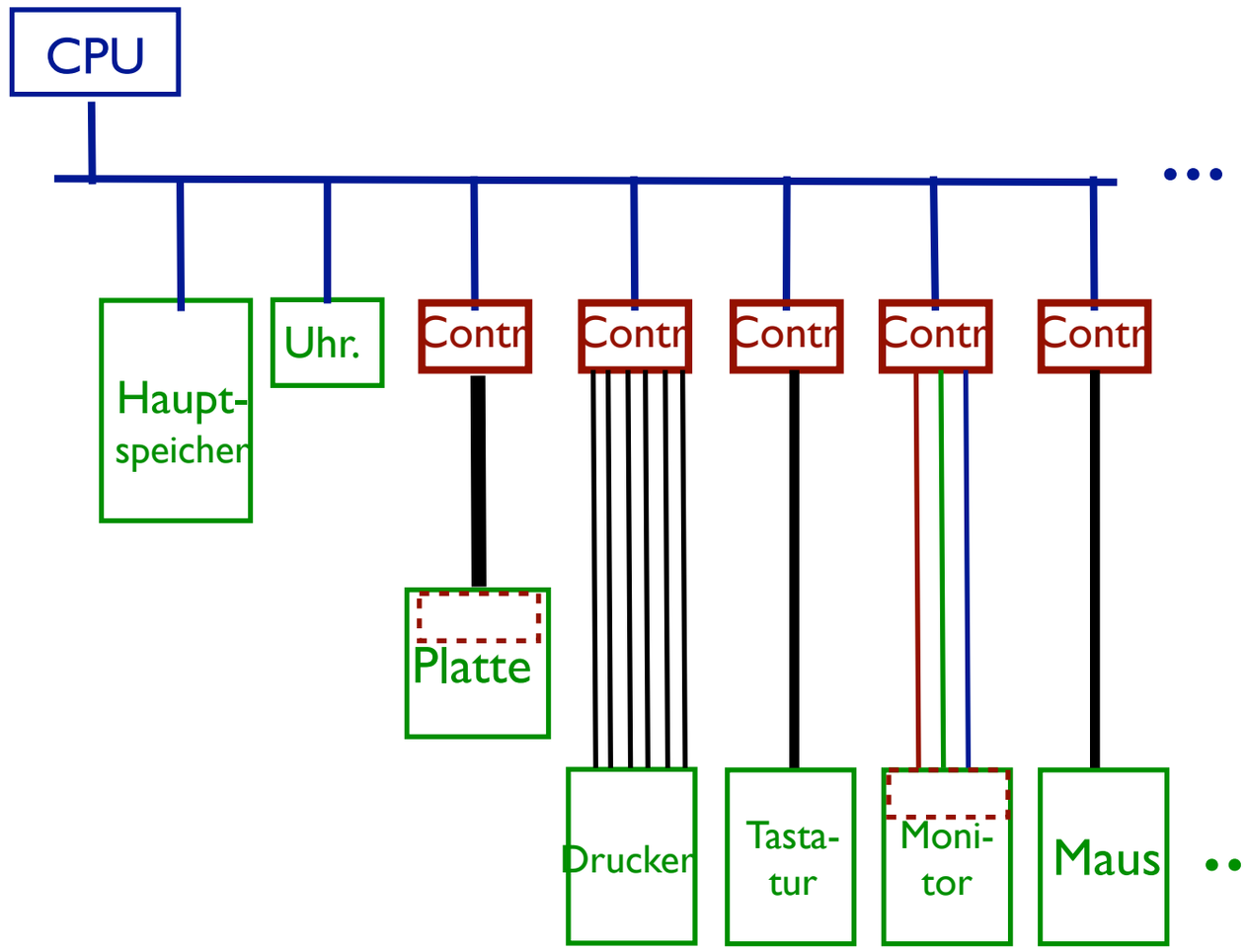
Anschluss von Geräten (vereinfacht)

- Klassischer Rechneraufbau:
  CPU über Bus mit Vielzahl von „Geräten“ verbunden

  ⇒ (in Grenzen) eigenständige Arbeit der Geräte

- Mehrere Geräte desselben Typs möglich
- U.U. mehrere Busabschnitte unterschiedlicher Länge
- Ansteuerung über Geräteadressen
- Anschlussleitungen
  - unterschiedlich „lang“
  - geräteabhängige Schnittstellen

  ⇒ Geräte-Controller:
  - Adapter zum Busanschluss
  - Ansteuerungshardware (zunehmend in Geräte integriert)



In Unix: Grobe Unterscheidung:

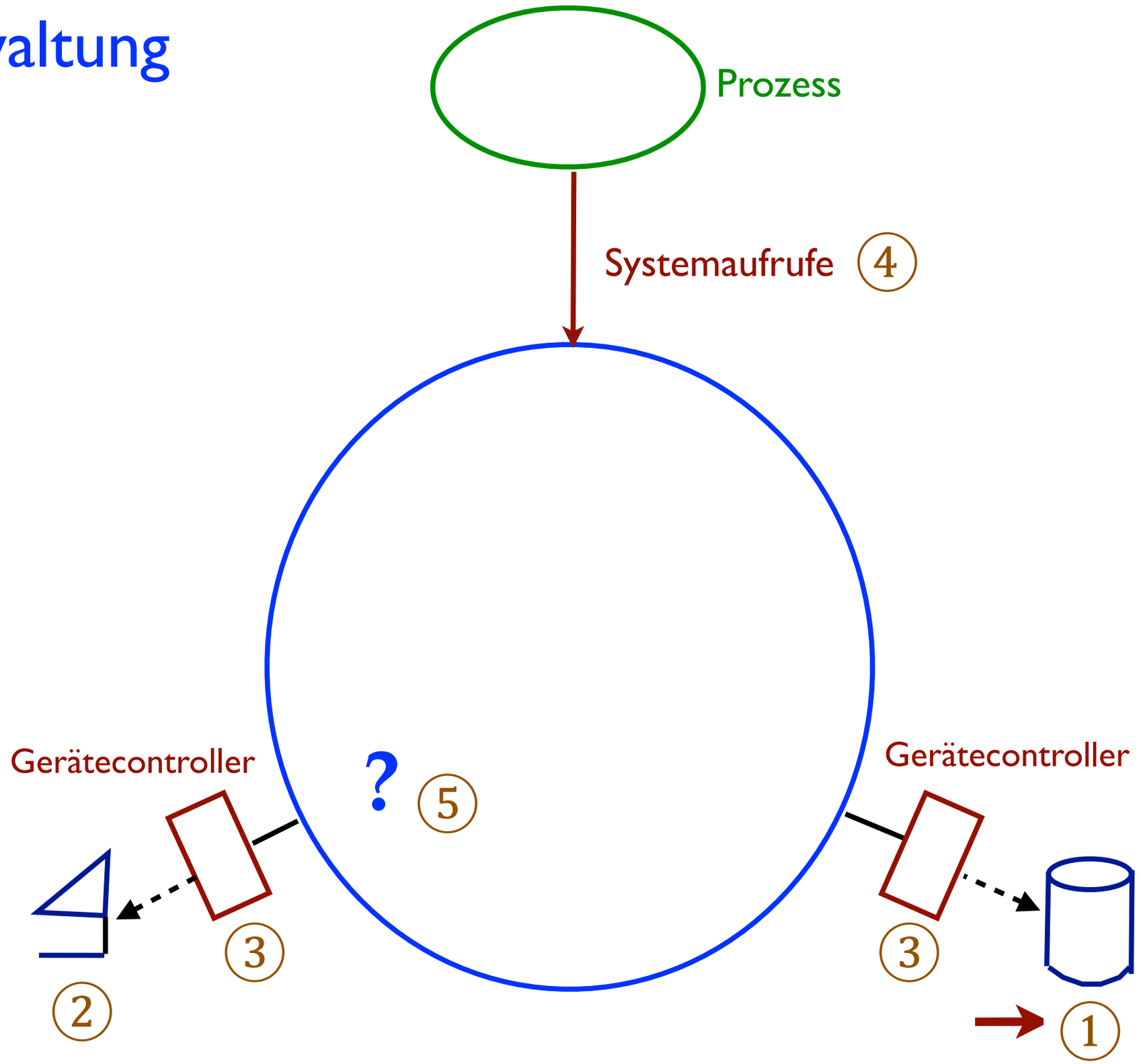
a) Blockgeräte:

- Zugriffseinheit sind Blöcke (z.B. 512 B ⇒ 4 KiB)
- I.d.R. wahlfrei adressierbar
- Beispiele: Platten, (früher auch Disketten)...

b) Charactergeräte:

- Zugriffseinheit sind Bytes
- I.d.R. nur sequentieller Zugriff (Bytestrom)
- Beispiele: Monitor, Drucker, Tastatur,…

Überblick Geräteverwaltung

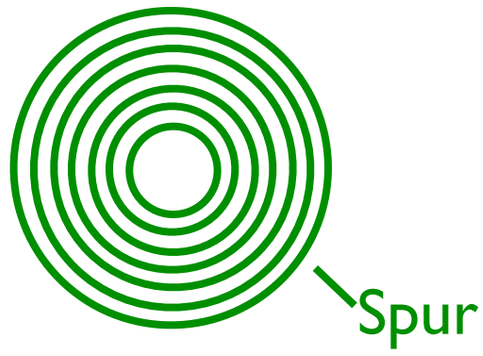


Organisation einer Platte (z.B. 500 GB)

- Plattenstapel: 3 Platten

  ⇒ 6 Oberflächen mit eigenem Schreib-/Lesekopf

- Jede Oberfläche unterteilt in Spuren (ca. 200000)

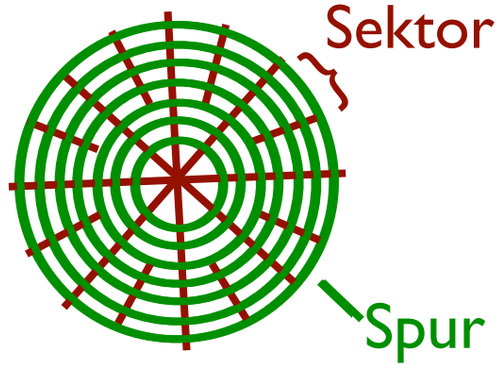


Organisation einer Platte (z.B. 500 GB)

- Plattenstapel: 3 Platten

  ⇒ 6 Oberflächen mit eigenem Schreib-/Lesekopf

- Jede Oberfläche unterteilt in Spuren (ca. 200000)
- Jede Spur unterteilt in Sektoren (ca. 500-1000)

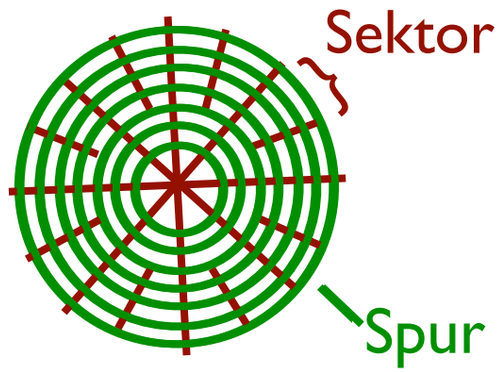


Organisation einer Platte (z.B. 500 GB)

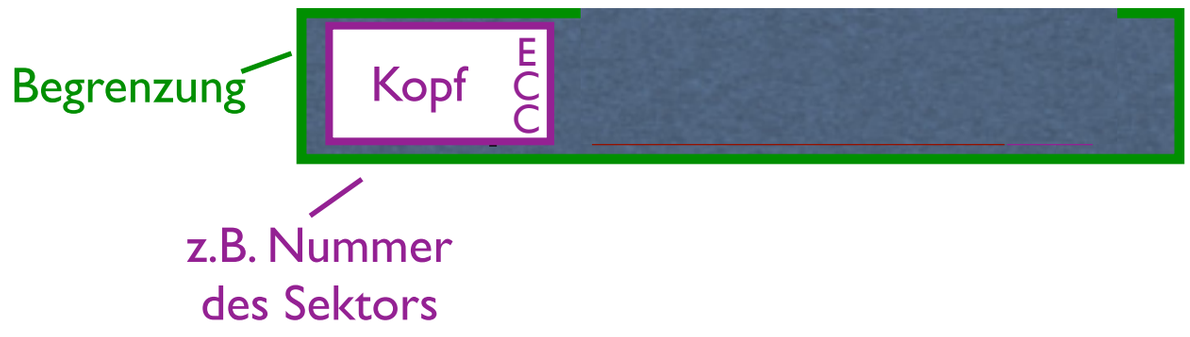
- Plattenstapel: 3 Platten

  ⇒ 6 Oberflächen mit eigenem Schreib-/Lesekopf

- Jede Oberfläche unterteilt in Spuren (ca. 200000)
- Jede Spur unterteilt in Sektoren (ca. 500-1000)



- Jeder Sektor umfasst einen physischen Plattenblock

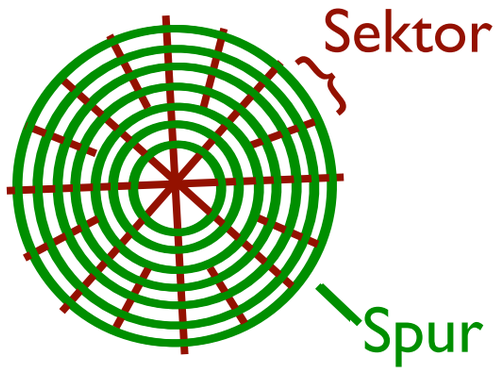


- Sektorerzeugung/-numerierung bei Plattenformatierung

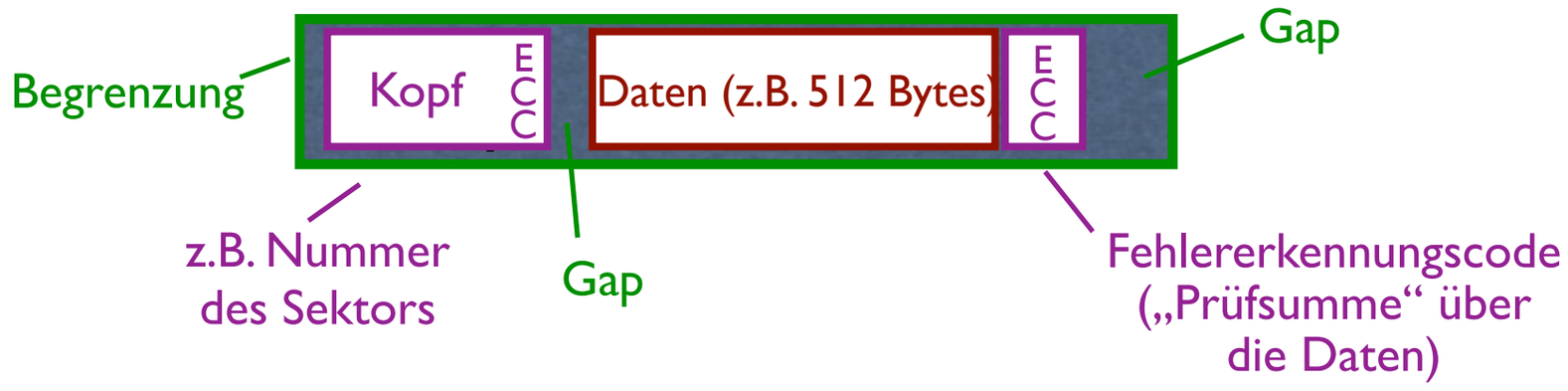
Organisation einer Platte (z.B. 500 GB)

- Plattenstapel: 3 Platten

  ⇒ 6 Oberflächen mit eigenem Schreib-/Lesekopf



- Jede Oberfläche unterteilt in Spuren (ca. 200000)
- Jede Spur unterteilt in Sektoren (ca. 500-1000)
- Jeder Sektor umfasst einen physischen Plattenblock



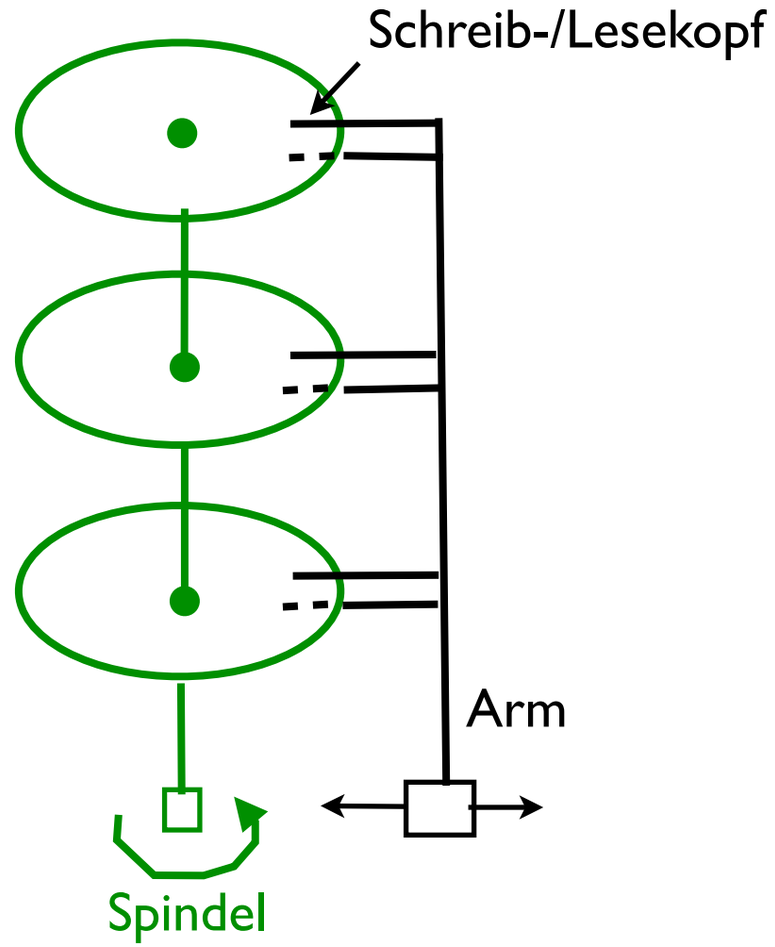
- Sektorerzeugung/-numerierung bei Plattenformatierung
- Datenblock nur als ganzes lesbar/schreibbar
- Gaps, da keine genaue Positionierung möglich („Puffer“)

Zurück zum Plattenstapel:

- Zugriff auf Datenblock x

  ⇒ Oberfläche i, Spur j, Sektor k

  - Positionieren des Arms auf Spur j
  - Warten bis Platte auf Sektor k gedreht wurde
  - Aktivieren des Schreib-/Lesekopfs von Oberfläche i

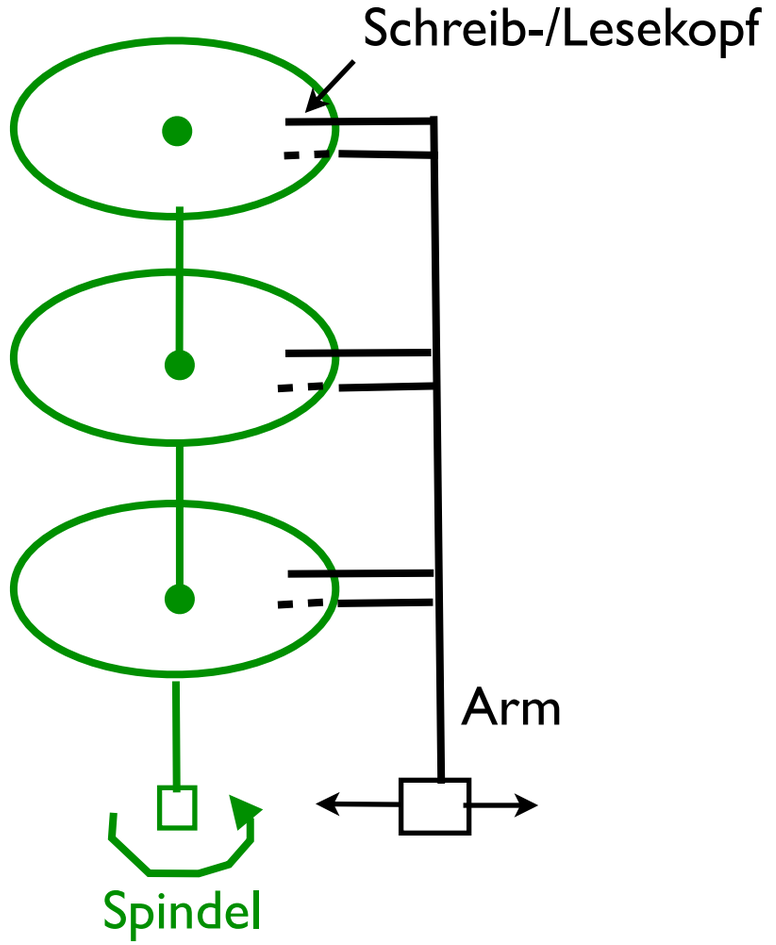


Zurück zum Plattenstapel:

- Zugriff auf Datenblock x

  ⇒ Oberfläche i, Spur j, Sektor k

  - Positionieren des Arms auf Spur j
  - Warten bis Platte auf Sektor k gedreht wurde
  - Aktivieren des Schreib-/Lesekopfs von Oberfläche i



- Verfahren zur Positionierung des Arms
  (Historische Entwicklung):

  a) Schrittmotor: langsam, veraltet

  b) Servo-Oberfläche: Grobpositionierung durch Impuls
  und eigene Oberfläche mit Infos zur Nachregulierung

  ⇒ ungerade Anzahl von „Nutz“-Oberflächen

  c) Nachregulierungsinfos auf jeder Oberfläche
  (heute üblich)

  ⇒ Schreib-/Leseköpfe unabhängig regulierbar

- Umdrehungsgeschwindigkeit:

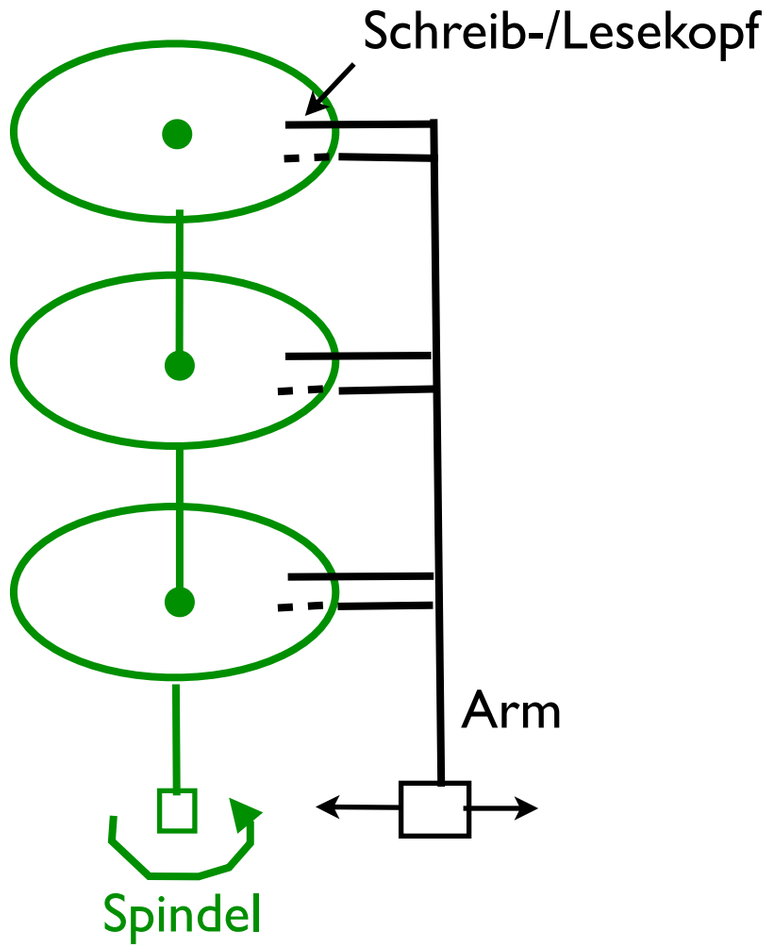
  7200 Umdrehungen/min (= 120 Umdr./s)

  ⇒ Umdrehung: 8,33ms

  ⇒ Wartezeit auf Sektor: im Mittel 4,17ms

- Armpositionierung:
  - ca. 2ms zur nächsten Spur
  - ca. 15ms von erster zu letzter Spur

  ⇒ Positionieren des Arms: im Mittel 8.5ms



- Folge: Armverschiebungen/Umdrehungen reduzieren, da teuer

- Umdrehungsgeschwindigkeit:

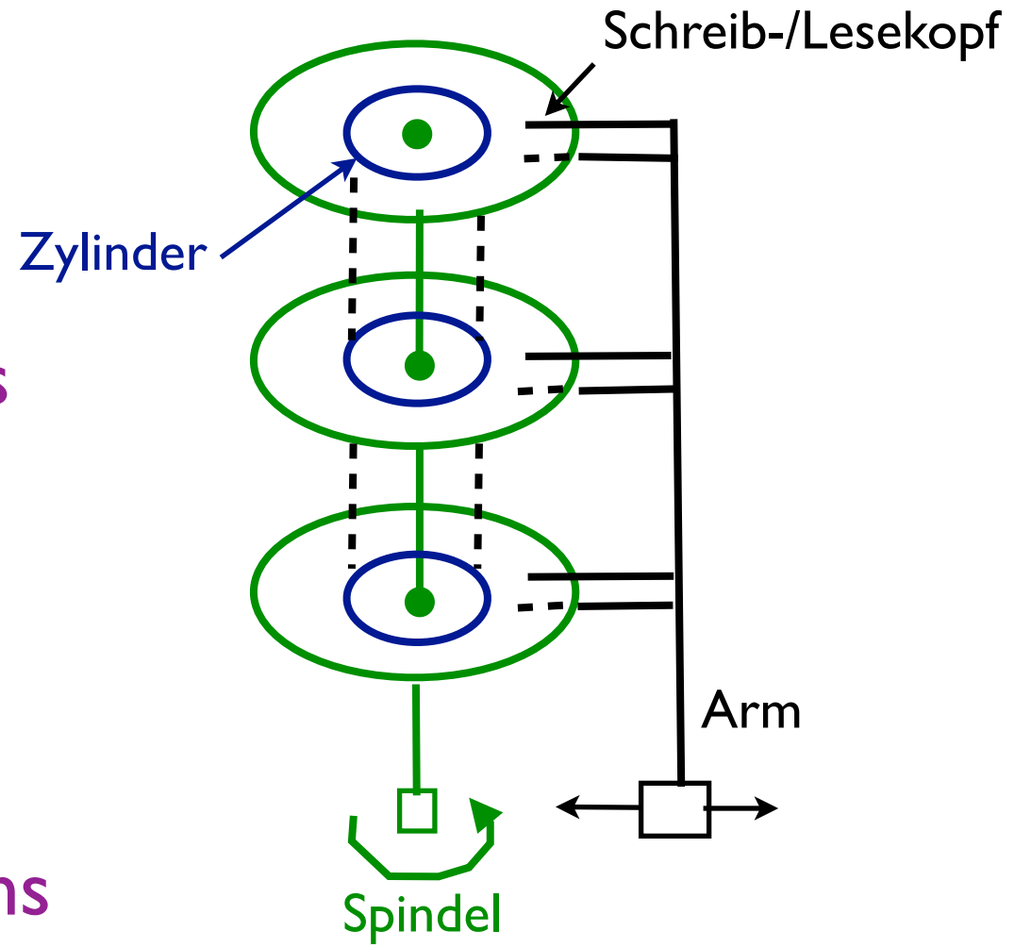
  7200 Umdrehungen/min (= 120 Umdr./s)

  ⇒ Umdrehung: 8,33ms

  ⇒ Wartezeit auf Sektor: im Mittel 4,17ms

- Armpositionierung:
  - ca. 2ms zur nächsten Spur
  - ca. 15ms von erster zu letzter Spur

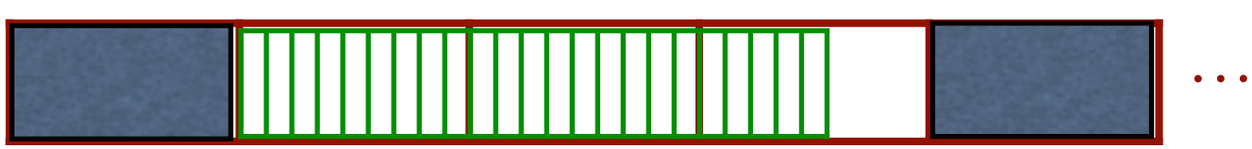
  ⇒ Positionieren des Arms: im Mittel 8.5ms



- Folge: Armverschiebungen/Umdrehungen reduzieren, da teuer

  ⇒ „Optimal“ ist sequentielles Lesen der Sektoren einer Spur

  ⇒ Zylinder: Gruppierung der Spur i aller Oberflächen



- Umdrehungsgeschwindigkeit:

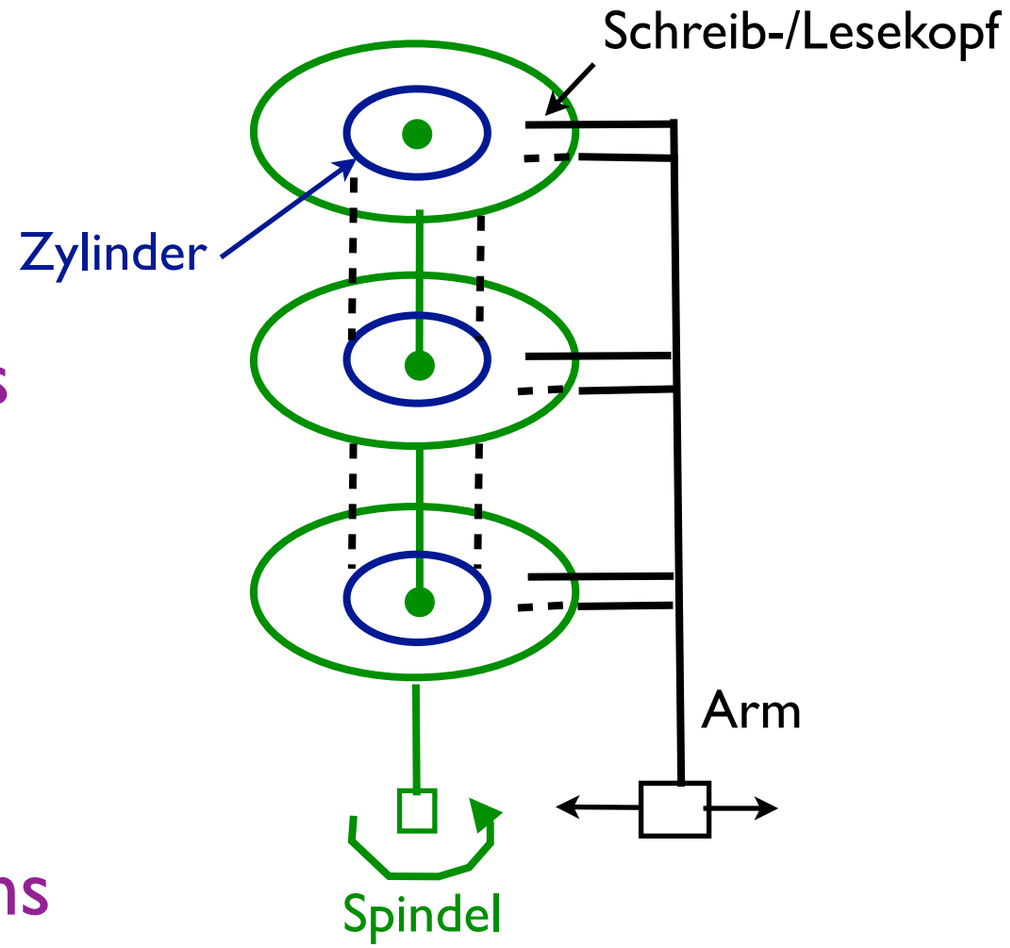
  7200 Umdrehungen/min (= 120 Umdr./s)

  ⇒ Umdrehung: 8,33ms

  ⇒ Wartezeit auf Sektor: im Mittel 4,17ms

- Armpositionierung:
  - ca. 2ms zur nächsten Spur
  - ca. 15ms von erster zu letzter Spur

  ⇒ Positionieren des Arms: im Mittel 8.5ms



- Folge: Armverschiebungen/Umdrehungen reduzieren, da teuer

  ⇒ „Optimal“ ist sequentielles Lesen der Sektoren einer Spur

  ⇒ Zylinder: Gruppierung der Spur i aller Oberflächen

- Umdrehungsgeschwindigkeit:

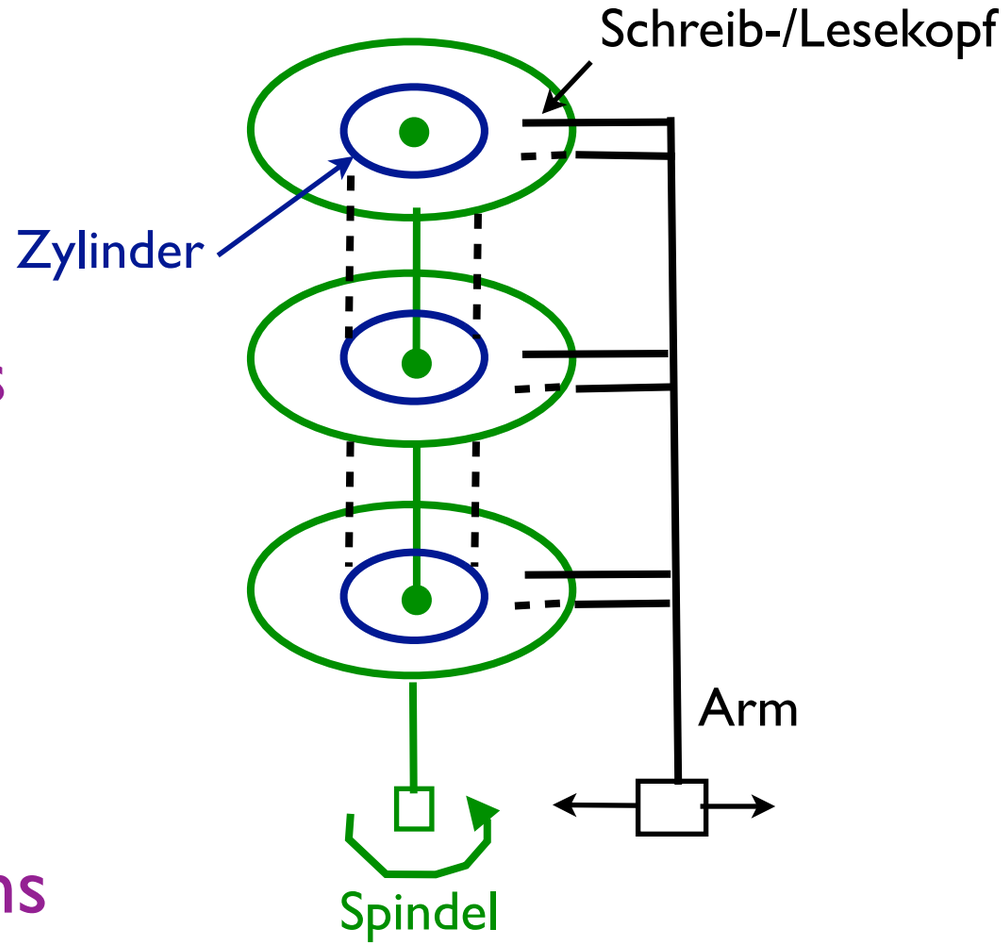
  7200 Umdrehungen/min (= 120 Umdr./s)

  ⇒ Umdrehung: 8,33ms

  ⇒ Wartezeit auf Sektor: im Mittel 4,17ms

- Armpositionierung:
  - ca. 2ms zur nächsten Spur
  - ca. 15ms von erster zu letzter Spur

  ⇒ Positionieren des Arms: im Mittel 8.5ms



- Folge: Armverschiebungen/Umdrehungen reduzieren, da teuer

  ⇒ „Optimal“ ist sequentielles Lesen der Sektoren einer Spur

  ⇒ Zylinder: Gruppierung der Spur i aller Oberflächen

- Heute durch zusätzlichen Datenpuffer entschärft (ca. 1024 Blöcke)

Exkurs: Auswirkung auf Dateisysteme

- „Sequentielle“ Plattenblöcke (physische Blöcke) zu Dateiblöcken (logischen Blöcken) zusammenfassen

  ⇒ erhöhter Durchsatz (mehr Infos auf einmal transferieren)

- Erhöhte Fragmentierung durch (in Grenzen) variable Blockgröße abfangen

- Datenblöcke einer Datei nicht beliebig über Platte verteilen

- Inodes und Verzeichnisinformationen nicht beliebig von zugehörigen Dateien trennen

  ⇒ Reduzierung der nötigen Armbewegungen

Exkurs: Auswirkung auf Dateisysteme

- „Sequentielle“ Plattenblöcke (physische Blöcke) zu Dateiblöcken (logischen Blöcken) zusammenfassen

  ⇒ erhöhter Durchsatz (mehr Infos auf einmal transferieren)

- Erhöhte Fragmentierung durch (in Grenzen) variable Blockgröße abfangen
- Datenblöcke einer Datei nicht beliebig über Platte verteilen
- Inodes und Verzeichnisinformationen nicht beliebig von zugehörigen Dateien trennen

  ⇒ Reduzierung der nötigen Armbewegungen

Realisiert z.B. in Unix Fast File System (UFS)

- 4 bzw. 8 KiB Datenblöcke
- letzter Block kann aus 512-Byte-Fragmenten bestehen

  ⇒ Folgen für Buffer-Cache

- Platte wird in Zylindergruppen unterteilt
- Jede Zylindergruppe hat Inode- und Datenbereich:
  - Dateiinhalt und Inode möglichst in derselben Zylindergruppe
  - Verzeichnis und enthaltene Dateien möglichst in derselben Zylindergruppe
  - nicht jedoch Unterverzeichnisse

Fragen – Teil 1

- Wie ist eine Platte intern organisiert?
- Wie wirkt sich dies auf den Informationszugriff aus?
- Wie geht das Unix *Fast File System* damit um?

Teil 2:
Zugriffsalgorithmen auf Plattenblöcke

Zugriffsalgorithmen auf Plattenblöcke

- Auch geschickt angeordnete Dateien erfordern Armbewegungen

- Zugriff auf verschiedene Dateien
  (u.U. von verschiedenen Prozessen aus)

⇒ Zugriffsstrategie erforderlich

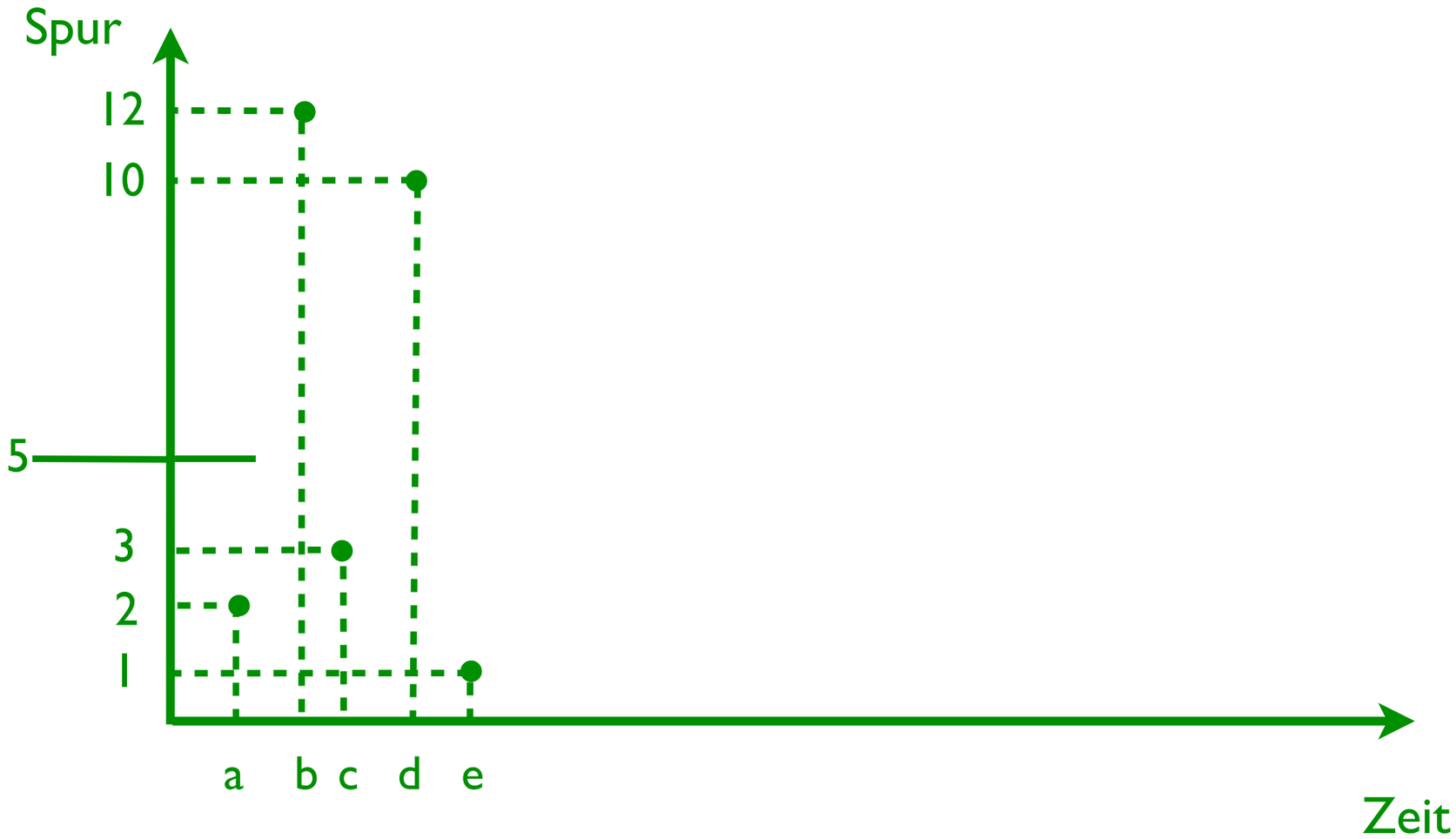
- Auftragswarteschlange

  ⇒ wie abarbeiten?

- Verschiedene Varianten
  (vereinfachend Schrittmotor angenommen)

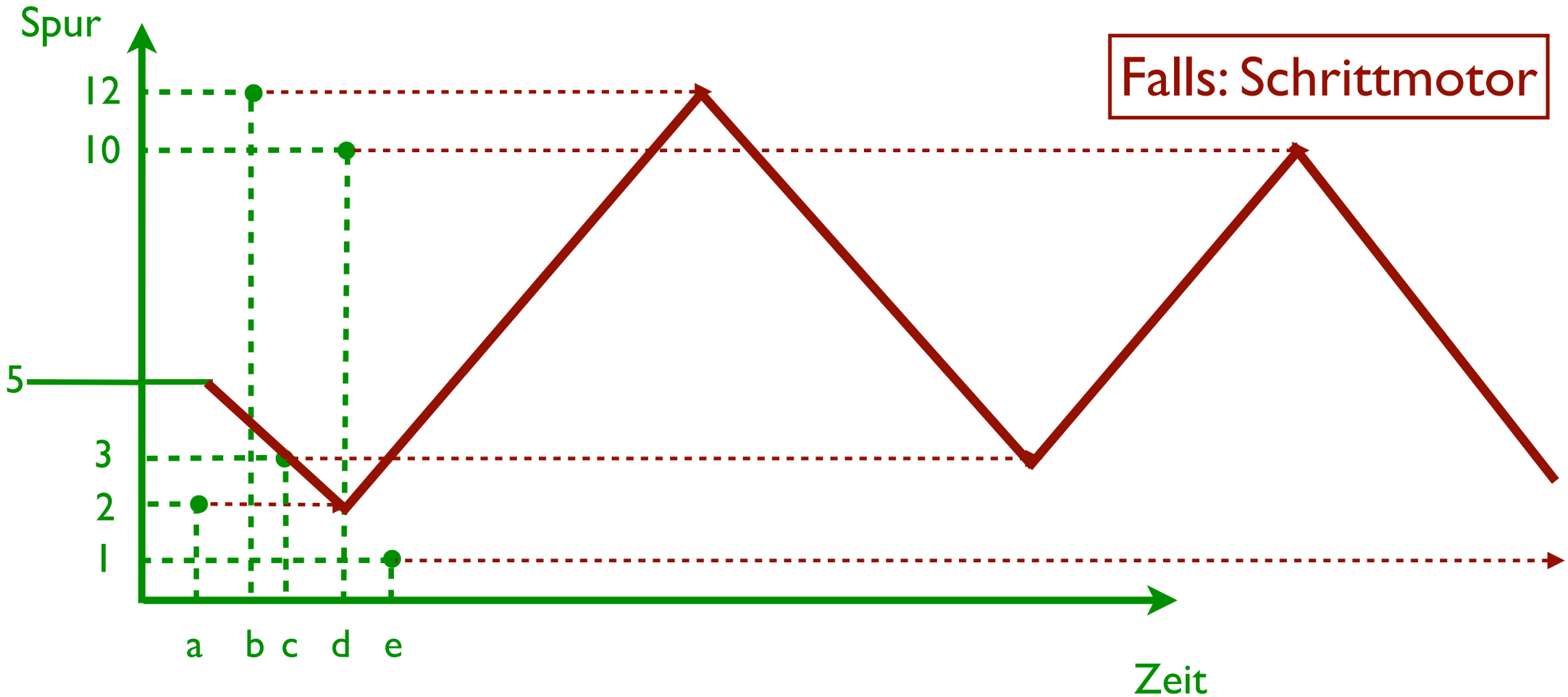
Betrachtetes Beispiel:

- Abarbeitung in der Reihenfolge des Eingangs



1. Versuch: First-Come-First-Served (FCFS)

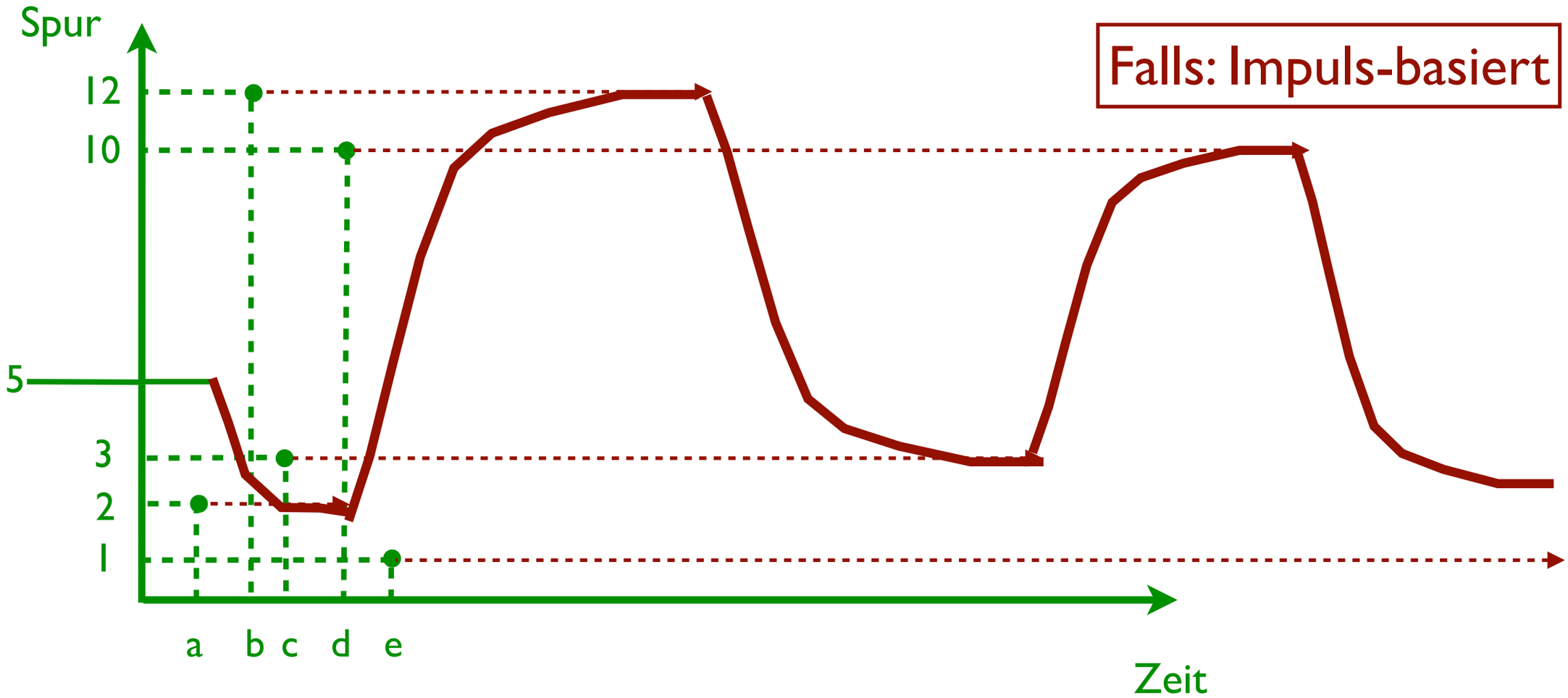
- Abarbeitung in der Reihenfolge des Eingangs



⇒ i.d.R. schlechte Auslastung, da zuviele Armbewegungen

1. Versuch: First-Come-First-Served (FCFS)

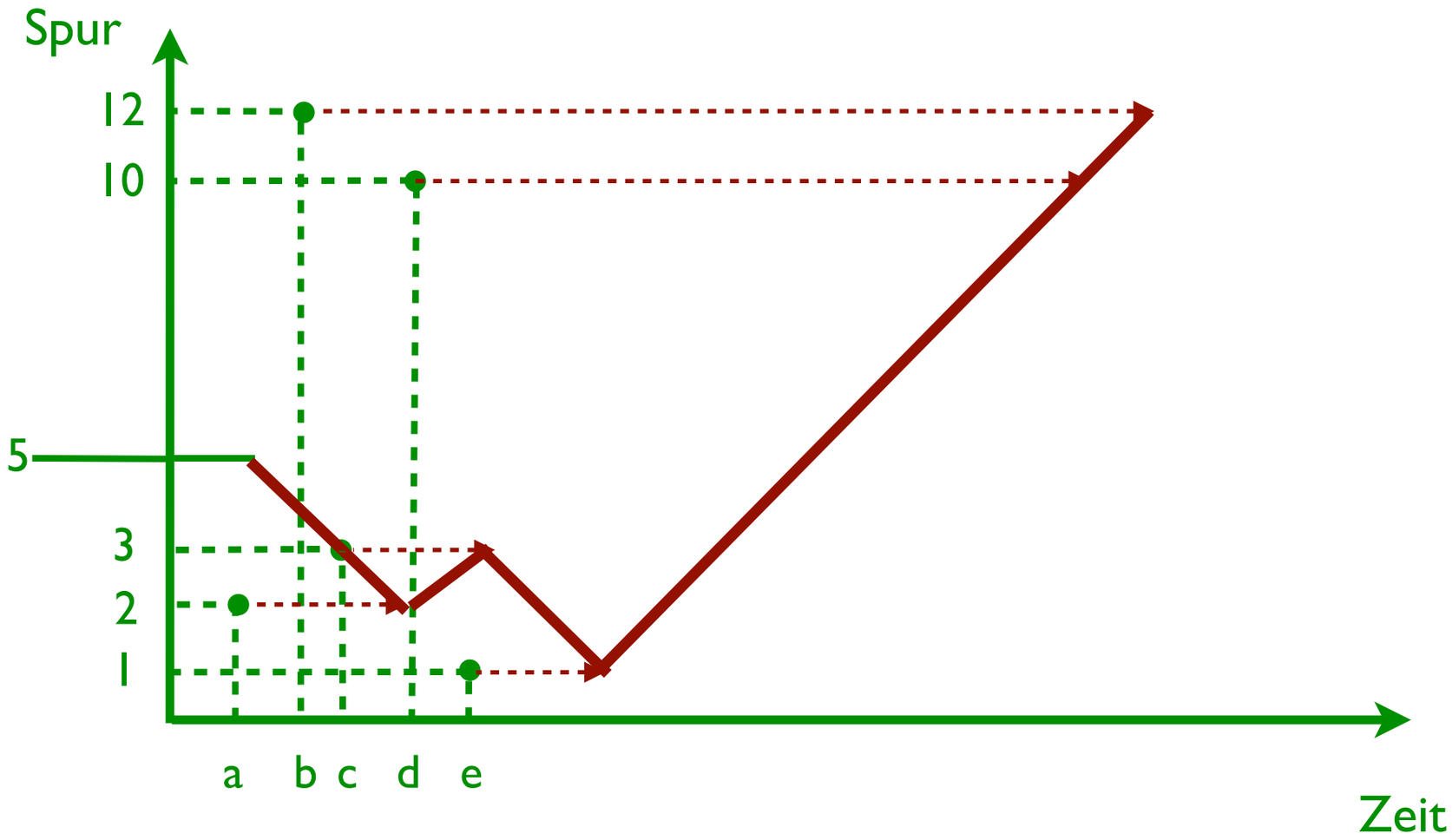
- Abarbeitung in der Reihenfolge des Eingangs



⇒ i.d.R. schlechte Auslastung, da zuviele Armbewegungen

2. Versuch: Shortest-Seek-First (SSF)

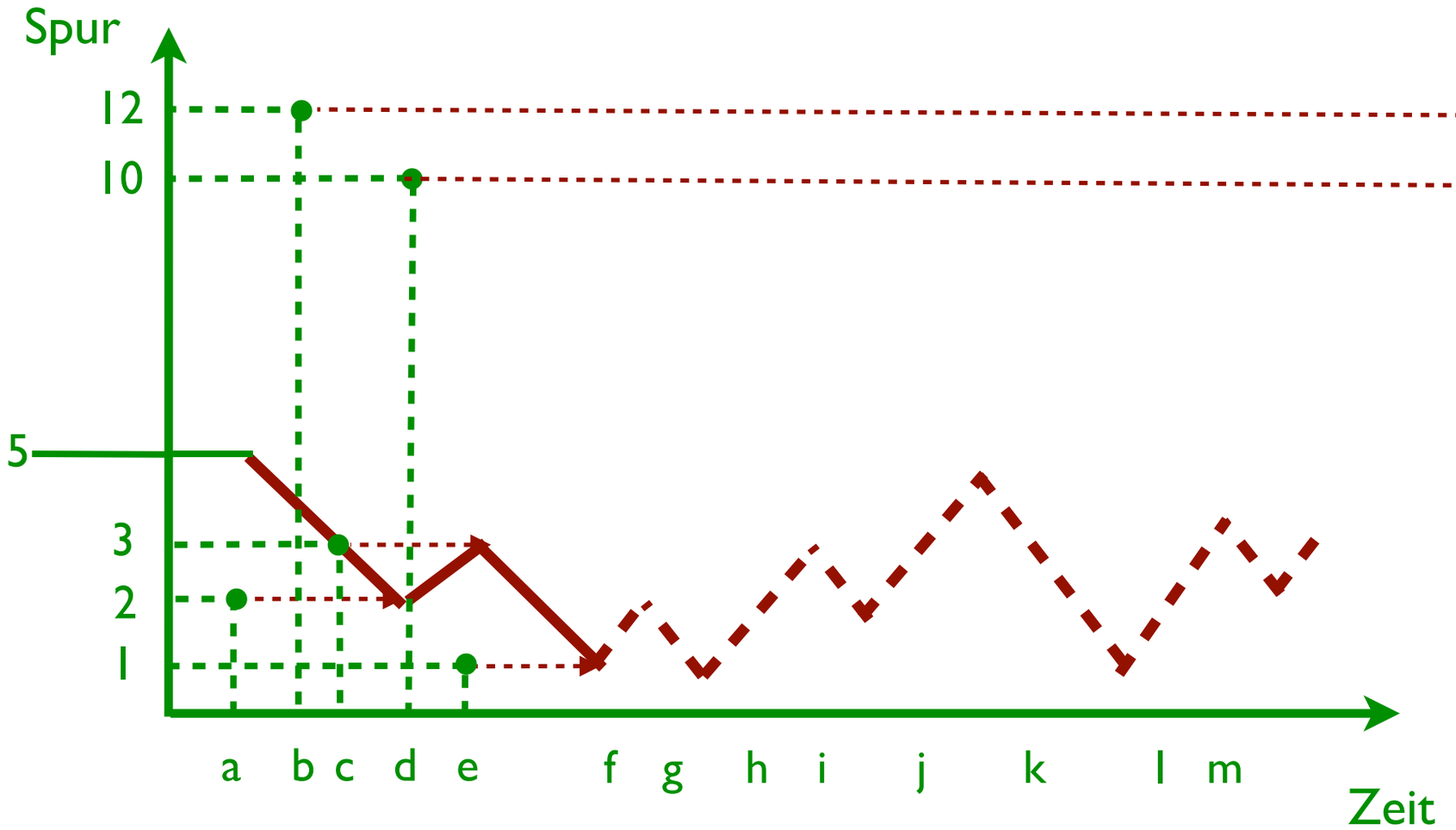
- Auftrag mit geringster Armbewegung als nächsten abarbeiten



⇒ gute Auslastung ...

2. Versuch: Shortest-Seek-First (SSF)

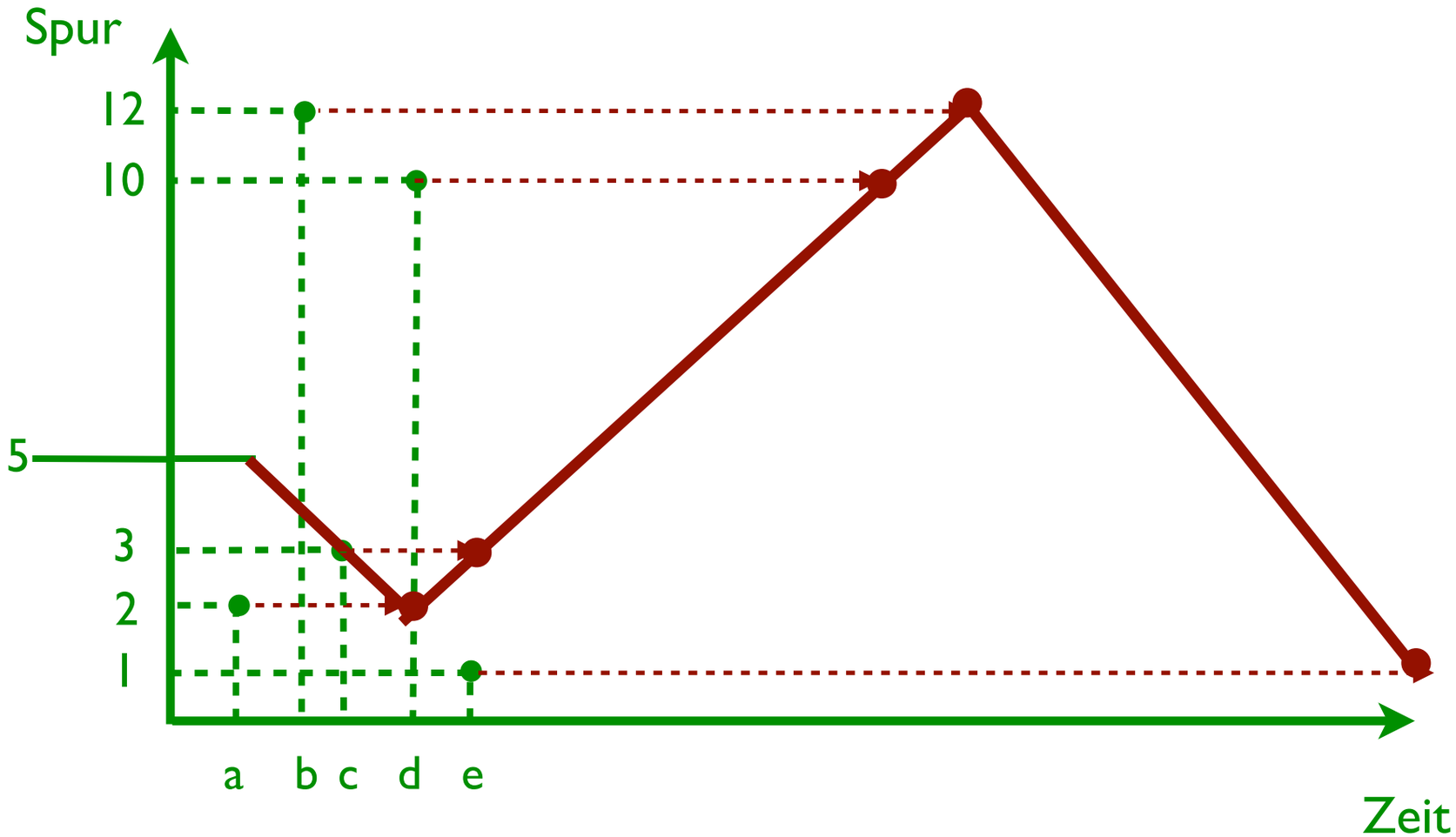
- Auftrag mit geringster Armbewegung als nächsten abarbeiten



⇒ gute Auslastung, aber alte Aufträge können „verhungern“

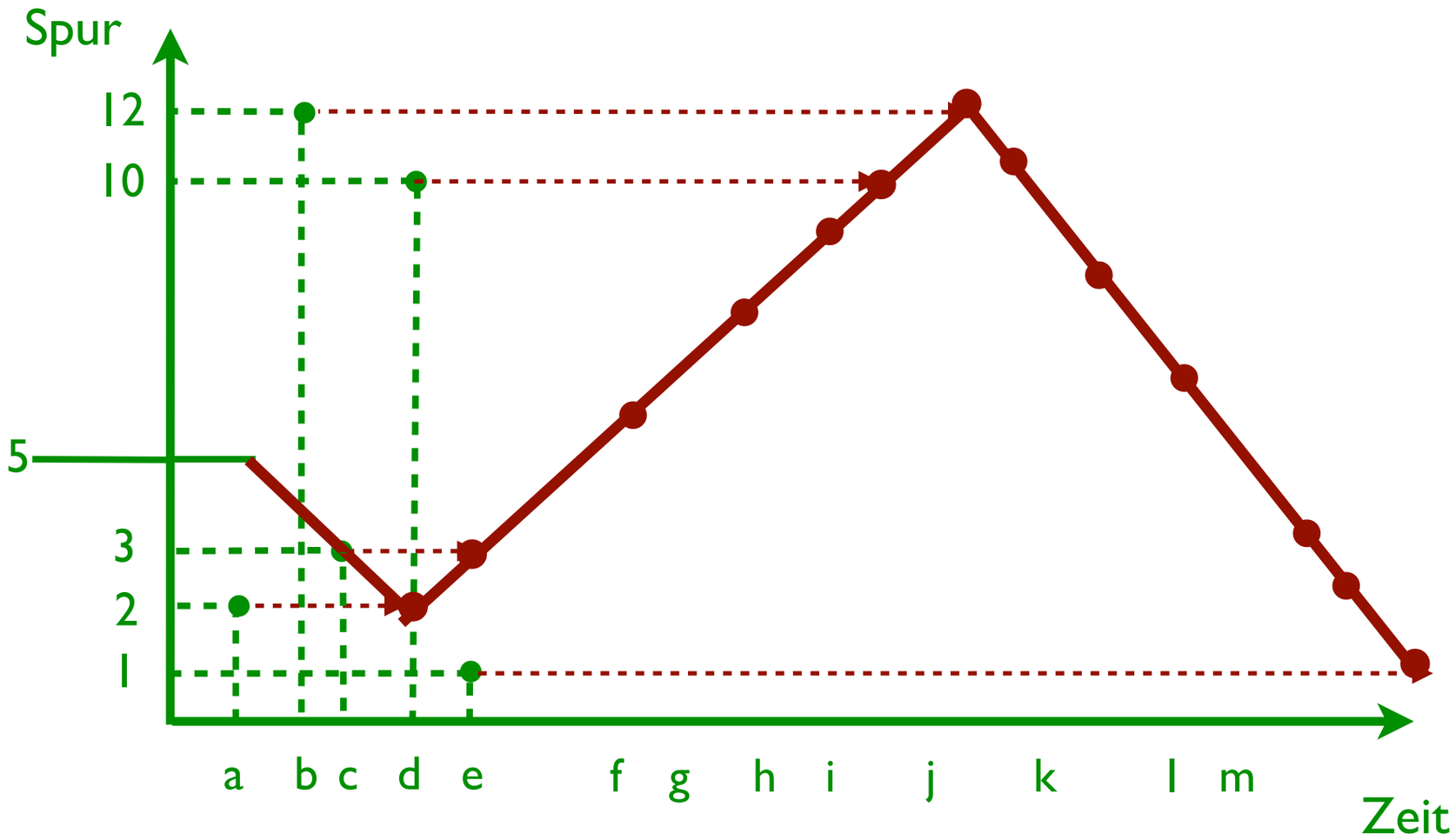
3. Versuch: Fahrstuhlalgorithmus

- Solange Richtung beibehalten, bis keine Aufträge in dieser Richtung mehr anstehen



3. Versuch: Fahrstuhlalgorithmus

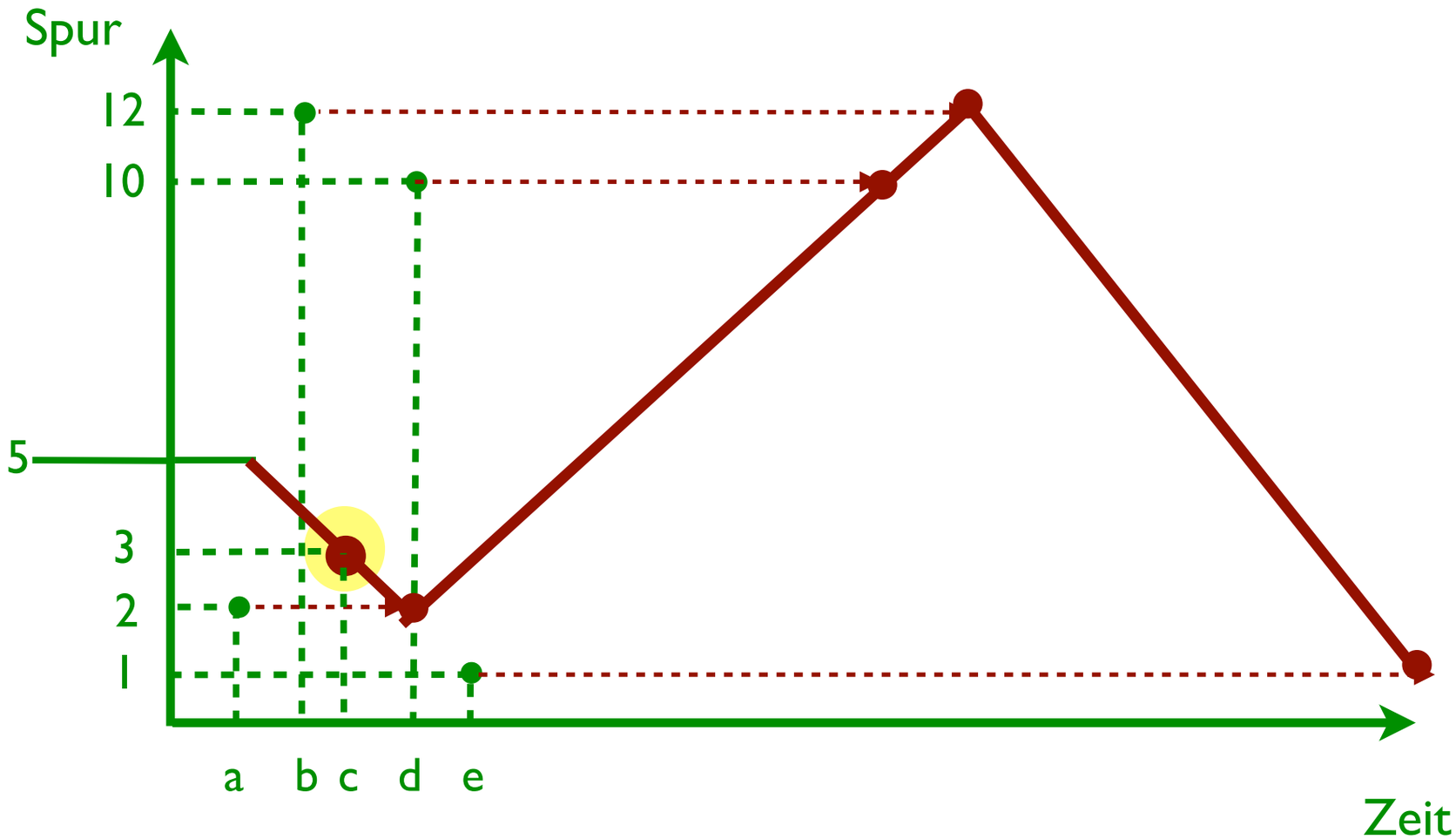
- Solange Richtung beibehalten, bis keine Aufträge in dieser Richtung mehr anstehen



- [Auch bei vielen Aufträgen noch akzeptable Abarbeitungszeiten möglich, kein „Verhungern“ von Aufträgen]

3. Versuch: Fahrstuhlalgorithmus

- Solange Richtung beibehalten, bis keine Aufträge in dieser Richtung mehr anstehen

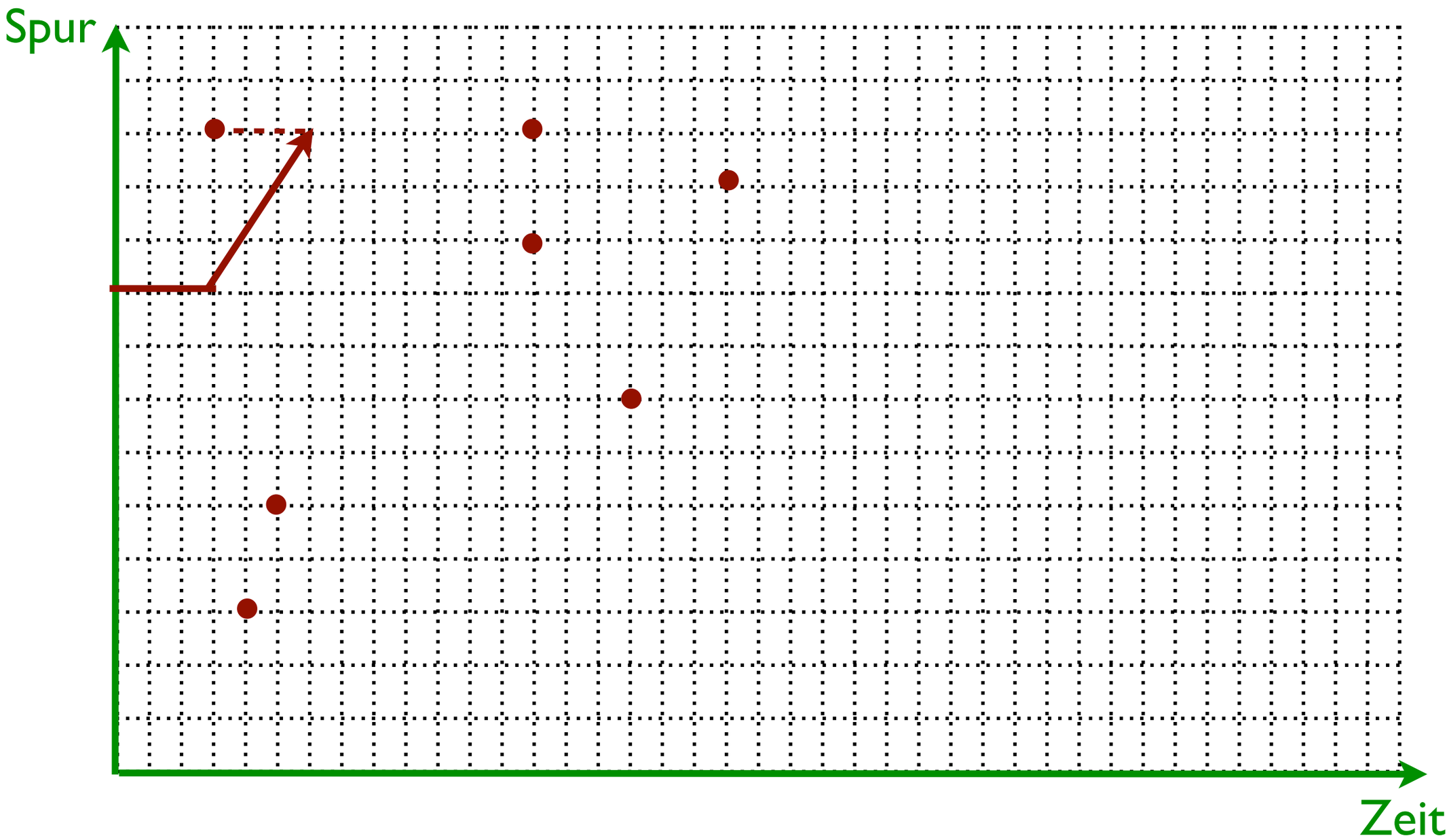


- [Alternative: Unterwegs noch neu anstehende Aufträge einsammeln
  ⇒ natürlich nur bei Schrittmotor denkbar]

⇒ wird häufig eingesetzt, gibt auch Optimierungen

Kleine Aufgabe

Wie würden die folgenden Plattenzugriffe auf Basis eines einfachen Fahrstuhlalgorithmus abgearbeitet?



Fragen – Teil 2

- Was versteht man beim Zugriff auf Plattenblöcke unter dem *Fahrstuhlalgorithmus*?

Teil 3:
Kleine Exkurse

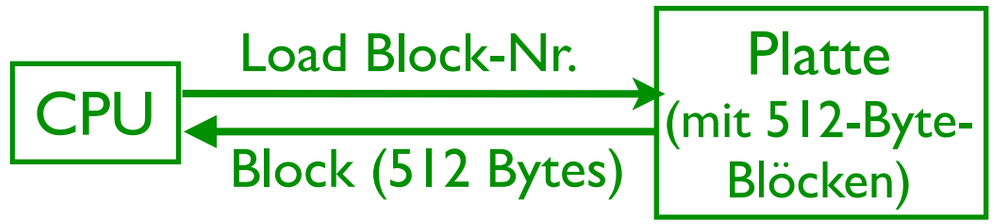
12 Geräteverwaltung (1), Dateiverwaltung (3), Ute Bormann

- Bisher betrachtet: Klassische Schnittstelle zu voll funktionsfähiger „klassischer“ Platte

- Kleine Exkurse:
  - Plattenblöcke: 512 Bytes → 4 KiB
  - Bad Blocks
  - SSD

Exkurs Blockgröße:

- Klassisch: 512-Byte-Plattenblöcke
- Allerdings:
  - Dateiblöcke: 1 KiB ⇒ 4 KiB
  - Pages: 4 KiB

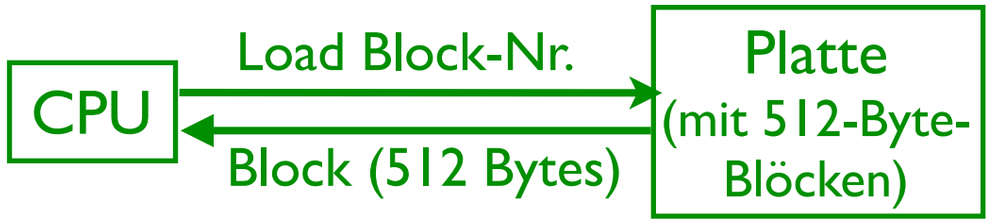


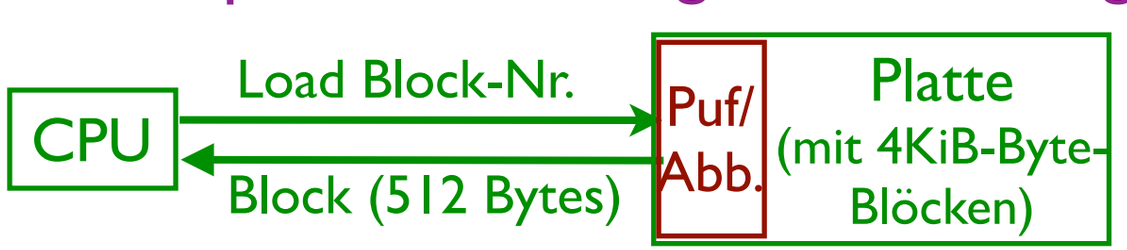
Exkurs Blockgröße:

- Klassisch: 512-Byte-Plattenblöcke
- Allerdings:
  - Dateiblöcke: 1 KiB ⇒ 4 KiB
  - Pages: 4 KiB
- Zunehmend: Übergang zu 4-KiB-Plattenblöcken

  ⇒ Jedoch Schnittstelle erstmal unverändert

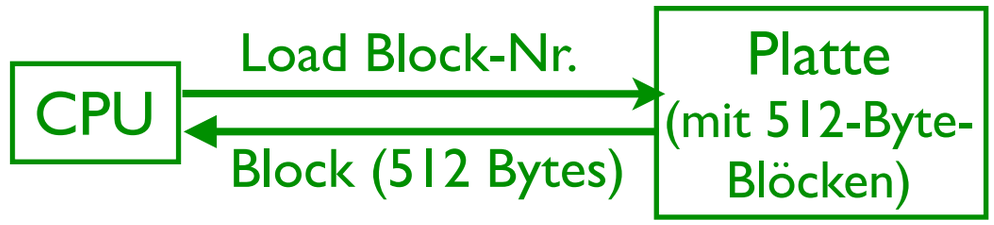
  ⇒ Zwischenpuffer/Abbildung in Platte nötig





Exkurs Blockgröße:

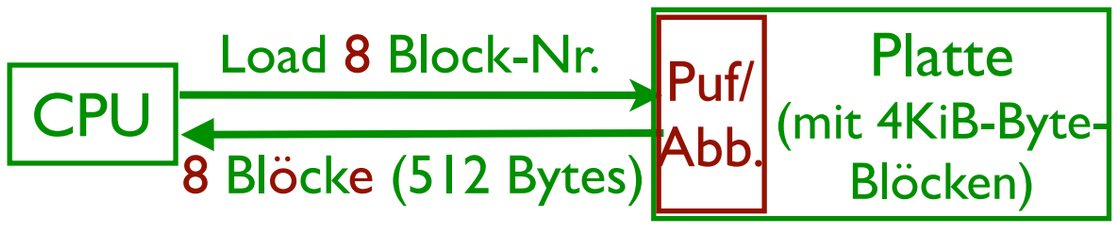
- Klassisch: 512-Byte-Plattenblöcke
- Allerdings:
  - Dateiblöcke: 1 KiB ⇒ 4 KiB
  - Pages: 4 KiB



- Zunehmend: Übergang zu 4-KiB-Plattenblöcken

  ⇒ Jedoch Schnittstelle erstmal unverändert

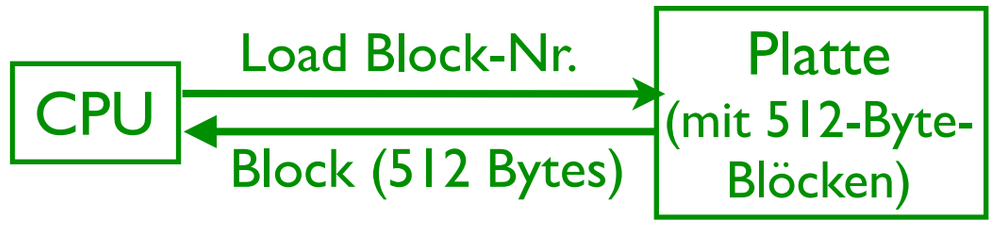
  ⇒ Zwischenpuffer/Abbildung in Platte nötig



- Allerdings ohnehin meist 4-KiB-Einheiten lesen/schreiben (Dateiblock, Page)

Exkurs Blockgröße:

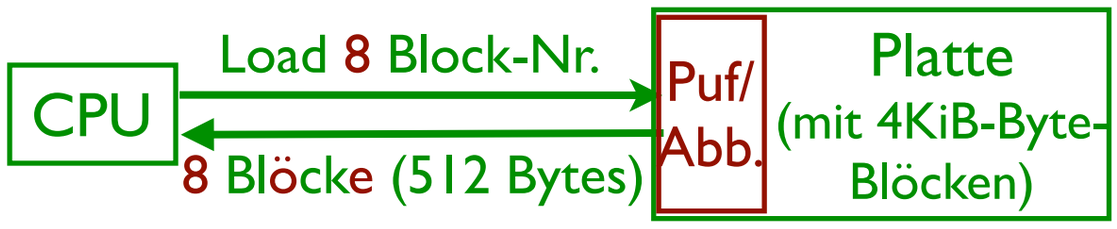
- Klassisch: 512-Byte-Plattenblöcke
- Allerdings:
  - Dateiblöcke: 1 KiB ⇒ 4 KiB
  - Pages: 4 KiB



- Zunehmend: Übergang zu 4-KiB-Plattenblöcken

  ⇒ Jedoch Schnittstelle erstmal unverändert

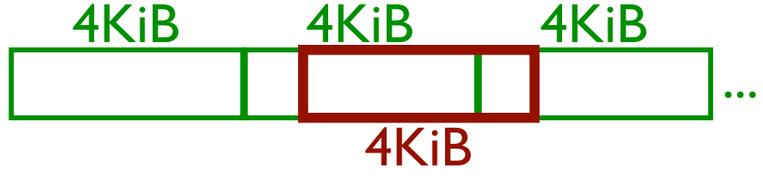
  ⇒ Zwischenpuffer/Abbildung in Platte nötig



- Allerdings ohnehin meist 4-KiB-Einheiten lesen/schreiben (Dateiblock, Page)

  ⇒ Platte so organisieren, dass alle Bereiche in 4-KiB-Einheiten strukturiert

- Andernfalls:



  ⇒ ggf. Platte umformatieren...

Exkurs: Fehlerhafte Plattenblöcke

- Platten nicht immer 100% in Ordnung (z.B. „Kratzer“, Verunreinigungen)
  ⇒ einzelne Blöcke nicht lesbar/schreibbar (permanente Prüfsummenfehler)

- Solche Bad Blocks nicht mehr an Dateien zuweisen. Möglichkeiten:
  a) Früher: Spezielle Datei mit Bad Blocks anlegen
     ⇒ im Dateisystem abhandeln
     ⇒ aber ggf. Probleme mit Sicherungskopie ganzer Platten

Exkurs: Fehlerhafte Plattenblöcke

- Platten nicht immer 100% in Ordnung (z.B. „Kratzer“, Verunreinigungen)
  ⇒ einzelne Blöcke nicht lesbar/schreibbar (permanente Prüfsummenfehler)

- Solche Bad Blocks nicht mehr an Dateien zuweisen. Möglichkeiten:
  a) Früher: Spezielle Datei mit Bad Blocks anlegen
     ⇒ im Dateisystem abhandeln
     ⇒ aber ggf. Probleme mit Sicherungskopie ganzer Platten

  b) Heute: „Platten-intern“ Ersatzblöcke zuweisen
     - z.B. spezielle Spuren/Sektoren reservieren
     - Abbildungstabellen führen

       ⇒ im Plattencontroller abfangen

     - Allerdings: Schlecht, wenn Auftragsabwicklung davon entkoppelt (u.U. schlechte Auslastung trotz Fahrstuhl-Algorithmus)

       ⇒ Fahrstuhlalgorithmus im Controller realisieren

Exkurs: Solid State Disk (SSD)

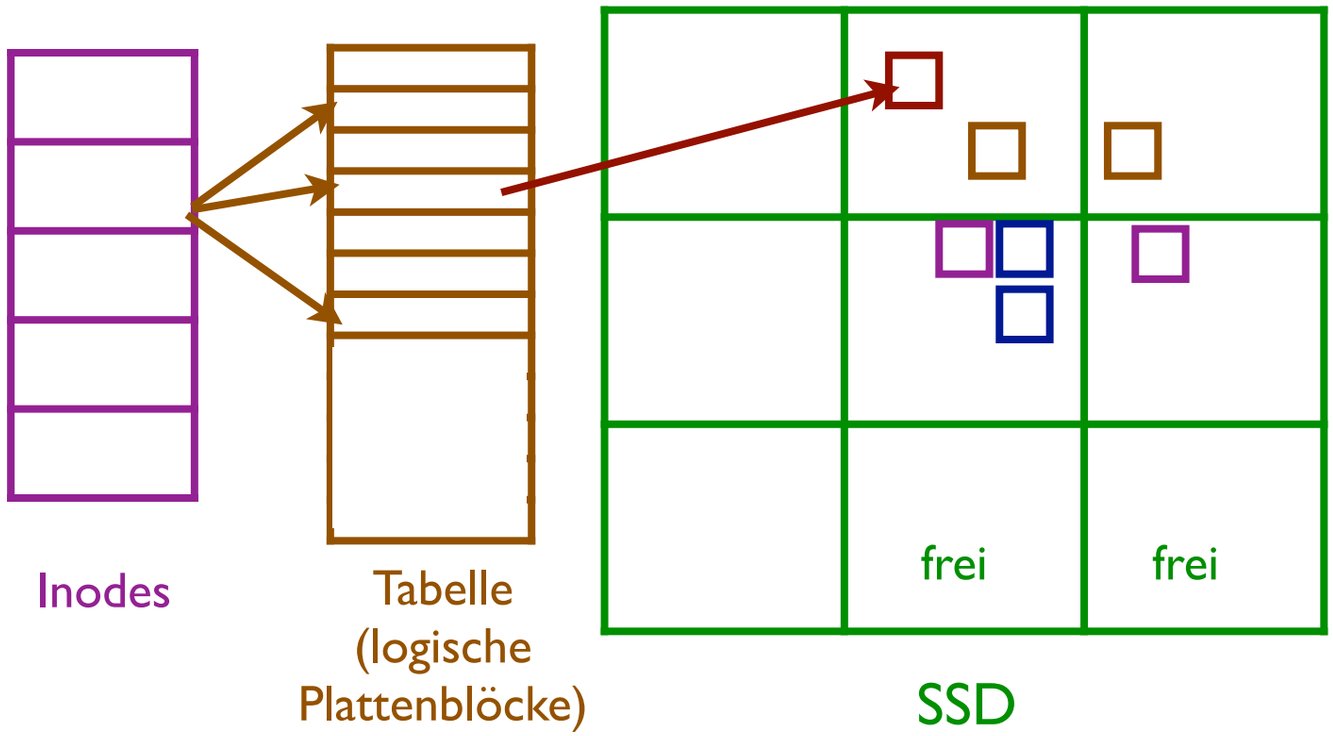
- Flash-Speicher
- Im Grundsatz wahlfreies Lesen/Schreiben

  ⇒ Fahrstuhlalgorithmus macht keinen Sinn

- Allerdings:
  - vor dem Schreiben löschen
  - nur größere Blöcke löschbar (ca. 512 KiB)
  - nicht beliebig oft löschbar
- Stattdessen:
  Beim Schreiben ggf. erstmal neuen Bereich nutzen

- Komplexe Abbildungstabelle verwalten (zwischen Dateisystem und Gerät)

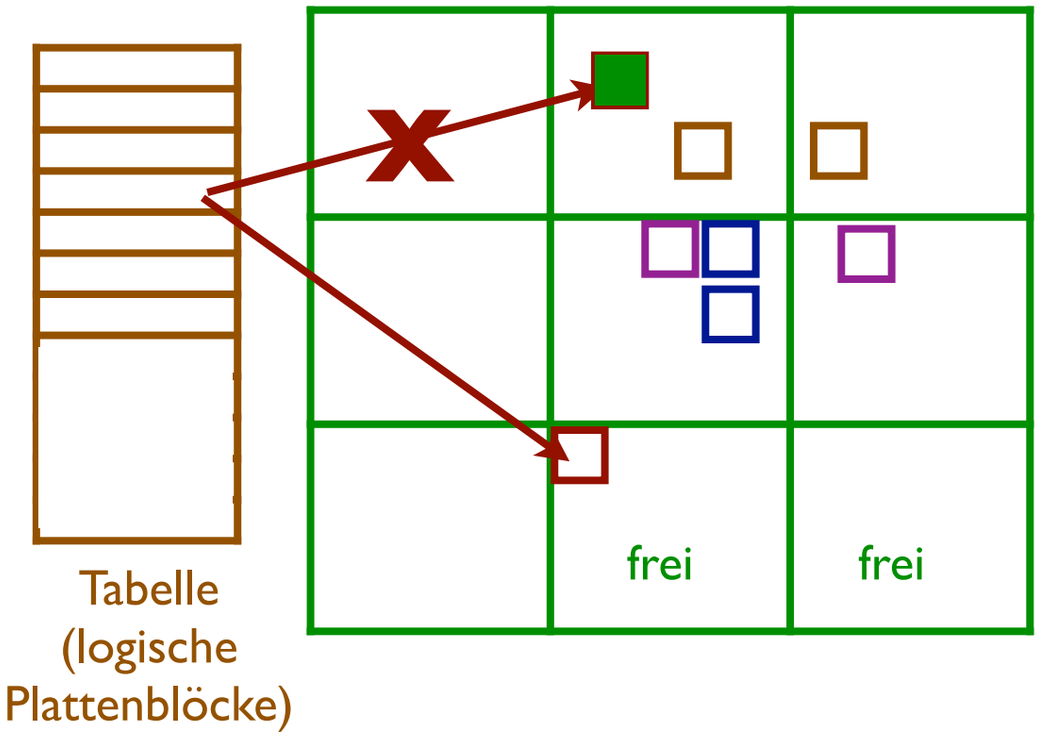
  ⇒ zunächst unveränderte Sicht auf das Dateisystem



- Komplexe Abbildungstabelle verwalten (zwischen Dateisystem und Gerät)

  ⇒ zunächst unveränderte Sicht auf das Dateisystem

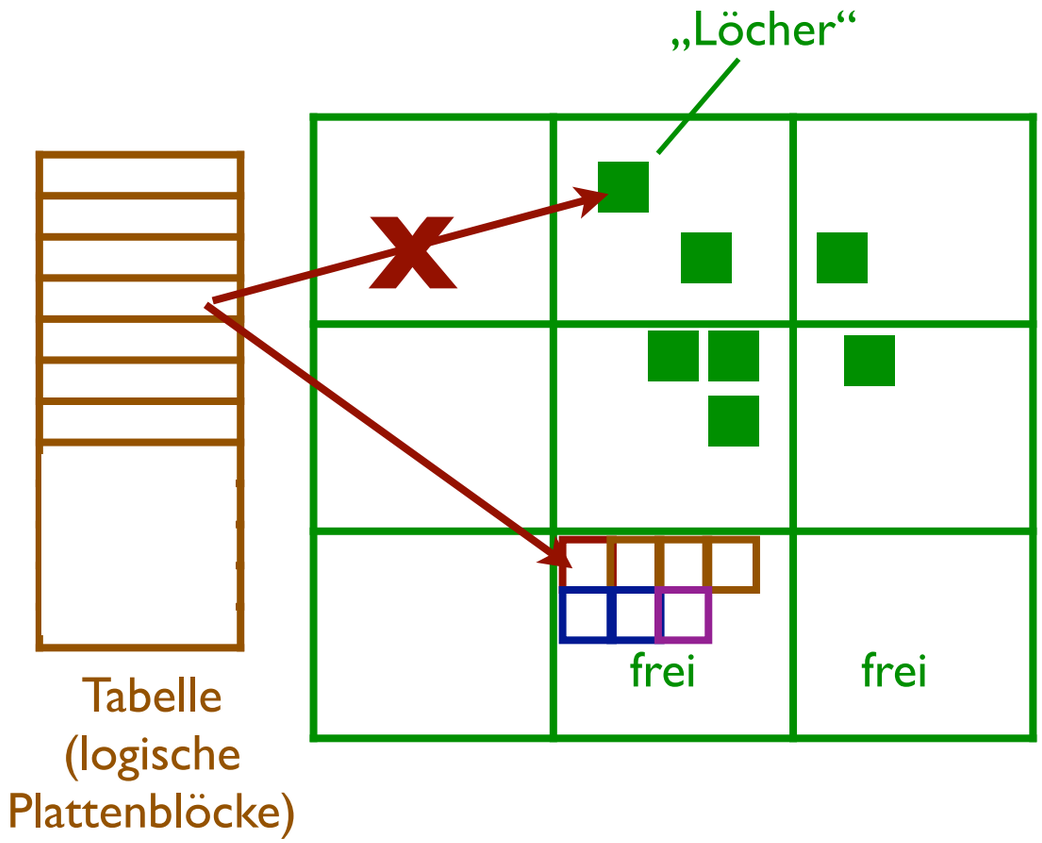
- Bei Änderungen neuen Block zuweisen



- Komplexe Abbildungstabelle verwalten (zwischen Dateisystem und Gerät)

  ⇒ zunächst unveränderte Sicht auf das Dateisystem

- Bei Änderungen neuen Block zuweisen



- Komplexe Abbildungstabelle verwalten (zwischen Dateisystem und Gerät)

  ⇒ zunächst unveränderte Sicht auf das Dateisystem

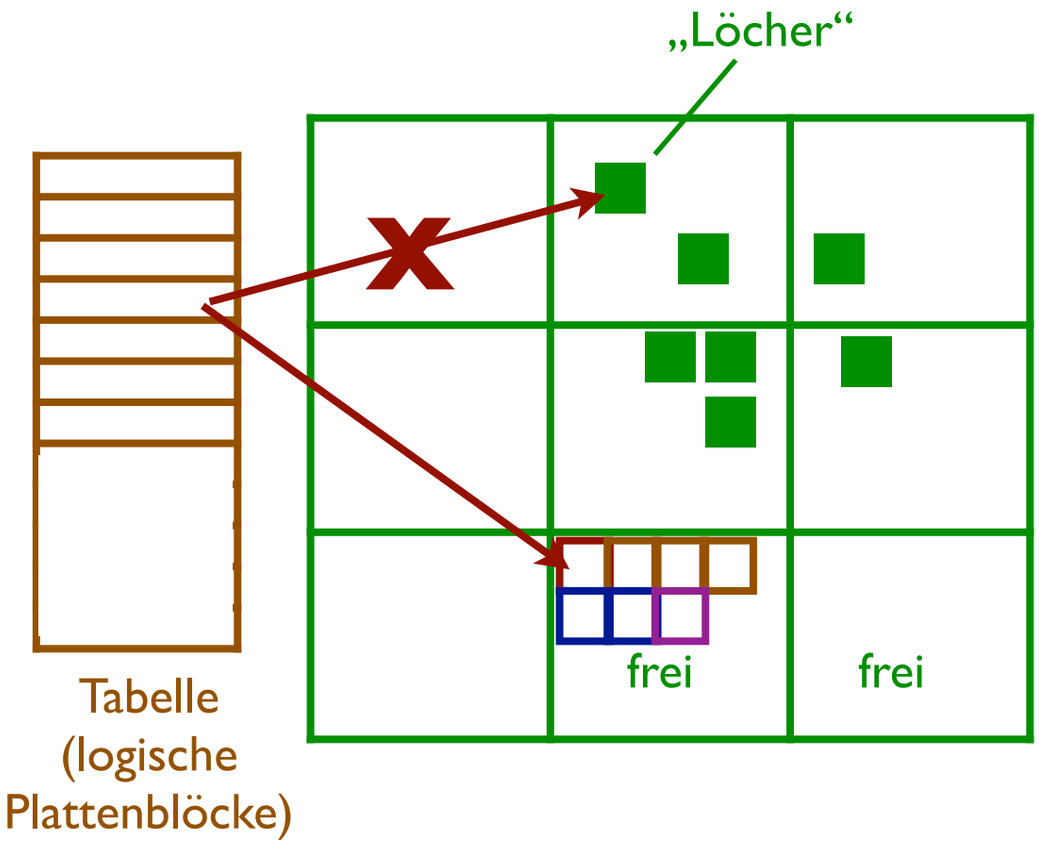
- Bei Änderungen neuen Block zuweisen

- entstehende Fragmentierung regelmäßig bereinigen

  ⇒ beim Umkopieren zusammenschieben

  (dazu Teil des Speichers ungenutzt lassen; leere Blöcke nicht mitkopieren)

  ⇒ trim()-Funktion, zusätzlich zu read() und write()



Fragen – Teil 3

- Welche wesentlichen Eigenschaften hat eine SSD *(Solid State Disk)*?

Teil 4:
Konsistenzprüfung eines Dateisystems

12 Geräteverwaltung (1), Dateiverwaltung (3), Ute Bormann

Konsistenzprüfung eines Dateisystems (Filesystem-Check)

- Systemzusammenbruch kann inkonsistentes Dateisystem hinterlassen:

  Beispiel: Block in Buffer-Cache gelesen, dort modifiziert, aber noch nicht auf Platte zurückgeschrieben (bdwrite())

- (Möglichst reduzieren: regelmäßig sync() aufrufen)

- Besonders kritisch bei Blöcken für Inodes, Verzeichnisse, Freispeicherverwaltung. Beispiele:
  - Datenblöcke nirgends oder mehrfach zugewiesen
  - Falscher Refcount in Inode
  - Dateien hängen in der Luft (in keinem Verzeichnis aufgeführt)

Darüber hinaus: Reihenfolge der Aktivitäten kann entscheidend sein.

Einige Beispiele:

1) Freigeben eines Datenblocks

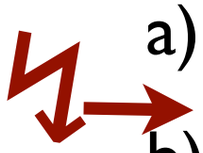
   a) Eintragen des Datenblocks in Freispeicherliste

   b) Austragen des Datenblocks aus Inode

Darüber hinaus: Reihenfolge der Aktivitäten kann entscheidend sein.

Einige Beispiele:

1) Freigeben eines Datenblocks



a) Eintragen des Datenblocks in Freispeicherliste

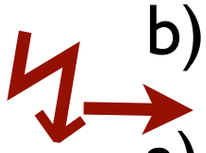
b) Austragen des Datenblocks aus Inode

⇒ potentielles Chaos !

Darüber hinaus: Reihenfolge der Aktivitäten kann entscheidend sein.

Einige Beispiele:

1) Freigeben eines Datenblocks



b) Austragen des Datenblocks aus Inode

a) Eintragen des Datenblocks in Freispeicherliste

⇒ besser !

Darüber hinaus: Reihenfolge der Aktivitäten kann entscheidend sein.

Einige Beispiele:

1) Freigeben eines Datenblocks

   b) Austragen des Datenblocks aus Inode

   a) Eintragen des Datenblocks in Freispeicherliste

2) Allozieren eines Datenblocks

   a) Eintragen des Datenblocks in Inode

   b) Füllen des Datenblocks mit Daten/Nullen

   c) Austragen des Datenblocks aus Freispeicherliste

⇒ potentielles Chaos !

Darüber hinaus: Reihenfolge der Aktivitäten kann entscheidend sein.

Einige Beispiele:

1) Freigeben eines Datenblocks

   b) Austragen des Datenblocks aus Inode

   a) Eintragen des Datenblocks in Freispeicherliste

2) Allozieren eines Datenblocks

   b) Füllen des Datenblocks mit Daten/Nullen

   c) Austragen des Datenblocks aus Freispeicherliste

   a) Eintragen des Datenblocks in Inode

⇒ besser !

Darüber hinaus: Reihenfolge der Aktivitäten kann entscheidend sein.

Einige Beispiele:

1) Freigeben eines Datenblocks

   b) Austragen des Datenblocks aus Inode

   a) Eintragen des Datenblocks in Freispeicherliste

2) Allozieren eines Datenblocks

   c) Austragen des Datenblocks aus Freispeicherliste

   a) Eintragen des Datenblocks in Inode

   b) Füllen des Datenblocks mit Daten/Nullen

⇒ bei bdwrite() auch üblich !

Darüber hinaus: Reihenfolge der Aktivitäten kann entscheidend sein.

Einige Beispiele:

1) Freigeben eines Datenblocks

   b) Austragen des Datenblocks aus Inode

   a) Eintragen des Datenblocks in Freispeicherliste

2) Allozieren eines Datenblocks

   c) Austragen des Datenblocks aus Freispeicherliste

   a) Eintragen des Datenblocks in Inode

   b) Füllen des Datenblocks mit Daten/Nullen

3) Anlegen/Löschen von Dateien

- Allozieren/Freigeben von Inodes...
- Eintragen/Austragen in Verzeichnis...
- Allozieren/Freigeben von Datenblöcken... (ggf. über Indirektblöcke)

- Probleme mit klassischen Dateisystemen nicht völlig vermeidbar

  ⇒ Daher Dienstprogramm zum Überprüfen von entstandenen Inkonsistenzen nach Systemzusammenbruch (fsck)

  - Zählen der Einträge in Verzeichnissen und Vergleich mit RefCount
  - Durchsuchen der Verweise auf Datenblöcke (je genau einer?)
  - ...

  ⇒ u.U. intelligentes Raten zur Überführung in konsistenten Zustand, z.B.

  - verlorengegangener Datenblock ⇒ Freispeicherliste
  - RefCount aktualisieren
  - ...

Fragen – Teil 4

- Nenne Beispiele für potentielle Inkonsistenzen in einem Dateisystem. Wie können sie entstehen? Wie kann man sie erkennen/beheben?

Teil 5:
Moderne Dateisysteme

12 Geräteverwaltung (1), Dateiverwaltung (3), Ute Bormann

Log-based File Systems (Journaling FS)

- Zunächst Log der geplanten Änderungen auf Platte schreiben
- Dann durchführen

  ⇒ Nach Systemzusammenbruch anhand des Logs rekonstruierbar

Log-based File Systems (Journaling FS)

- Zunächst Log der geplanten Änderungen auf Platte schreiben
- Dann durchführen

  ⇒ Nach Systemzusammenbruch anhand des Logs rekonstruierbar

- Dennoch ggf. Problem mit Inkonsistenzen bei Systemzusammenbruch während des Rausschreibens des Log-Files
- Außerdem: Heute oft RAID-Plattensysteme

Exkurs RAID: Redundant Array of Independent Disks

- Platten immer größer ⇒ komplexer ⇒ zu teuer
  ⇒ Stattdessen mehrere kleinere/günstigere Platten verwenden

- Allerdings auch Probleme mit Ausfällen
  ⇒ Redundanz vorsehen

- Beispiel RAID4 (vereinfacht):
  - Informationen auf *n* (z.B. 4) Platten verteilt

1. 1100
2. 1010
3. 0101
4. 1000

Exkurs RAID: Redundant Array of Independent Disks

- Platten immer größer ⇒ komplexer ⇒ zu teuer
  ⇒ Stattdessen mehrere kleinere/günstigere Platten verwenden

- Allerdings auch Probleme mit Ausfällen
  ⇒ Redundanz vorsehen

- Beispiel RAID4 (vereinfacht):
  - Informationen auf *n* (z.B. 4) Platten verteilt
  - *n*+*1*. Platte enthält Redundanz („Parity“) ⇒ XOR-Wert

1. 1100
2. 1010
3. 0101
4. 1000

5. 1011

Exkurs RAID: Redundant Array of Independent Disks

- Platten immer größer ⇒ komplexer ⇒ zu teuer
  ⇒ Stattdessen mehrere kleinere/günstigere Platten verwenden

- Allerdings auch Probleme mit Ausfällen
  ⇒ Redundanz vorsehen

- Beispiel RAID4 (vereinfacht):
  - Informationen auf *n* (z.B. 4) Platten verteilt
  - *n*+*1*. Platte enthält Redundanz („Parity“) ⇒ XOR-Wert

1. 1100
2. 1010

rekonstruierbar ⇒ ~~3. 0101~~

4. 1000

5. 1011

Exkurs RAID: Redundant Array of Independent Disks

- Platten immer größer ⇒ komplexer ⇒ zu teuer
  ⇒ Stattdessen mehrere kleinere/günstigere Platten verwenden

- Allerdings auch Probleme mit Ausfällen
  ⇒ Redundanz vorsehen

- Beispiel RAID4 (vereinfacht):
  - Informationen auf *n* (z.B. 4) Platten verteilt
  - *n*+*1*. Platte enthält Redundanz („Parity“) ⇒ XOR-Wert

1. 1100
2. 1010

rekonstruierbar ⇒ ~~3. 0101~~

4. 1000

5. 1011

- [Typischer: RAID5 ⇒ Redundanz verstreut über alle Platten]

A	B	C	D	E
A1	B1	C1	D1	XOR 1
A2	B2	C2	XOR 2	E2
A3	B3	XOR 3	D3	E3
A4	XOR 4	C4	D4	E4
XOR 5	B5	C5	D5	E5
⋮	⋮	⋮	⋮	⋮

Exkurs RAID: Redundant Array of Independent Disks

- Platten immer größer ⇒ komplexer ⇒ zu teuer
  ⇒ Stattdessen mehrere kleinere/günstigere Platten verwenden

- Allerdings auch Probleme mit Ausfällen
  ⇒ Redundanz vorsehen

- Beispiel RAID4 (vereinfacht):
  - Informationen auf *n* (z.B. 4) Platten verteilt
  - *n*+*1*. Platte enthält Redundanz („Parity“) ⇒ XOR-Wert

1. 1100
2. 1010

rekonstruierbar ⇒ ~~3. 0101~~

4. 1000

5. 1011

- [Typischer: RAID5 ⇒ Redundanz verstreut über alle Platten]

- Allerdings Problem mit Inkonsistenzen:
  Systemzusammenbruch zwischen Schreiben von Block auf Platte *i* und Aktualisierung der Redundanz auf Platte *j*

ZFS-Filesystem

- Abhilfe: Übergang auf ZFS-Filesystem (für verschiedene Unix-Varianten verfügbar)
- Mehrere Neuerungen, z.B.:
  - Copy-on-write-Transaktionen
  - Prüfsummen
  - 128-Bit-Dateiblocknummern ⇒ sehr viel größere Dateien möglich

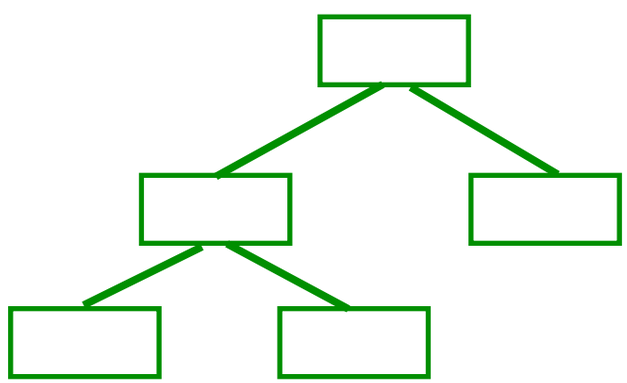
Copy-on-write-Transaktionen

- Bei jeder Änderung im Dateisystem auf der Platte wird neuer Block angelegt. Alter Block bleibt erstmal erhalten.

  ⇒ Änderung zieht sich bis zur Wurzel des Dateisystems durch

  ⇒ Wird erst aktuell, wenn Wurzelknoten geändert

- Vereinfacht:



ZFS-Filesystem

- Abhilfe: Übergang auf ZFS-Filesystem (für verschiedene Unix-Varianten verfügbar)
- Mehrere Neuerungen, z.B.:
  - Copy-on-write-Transaktionen
  - Prüfsummen
  - 128-Bit-Dateiblocknummern ⇒ sehr viel größere Dateien möglich

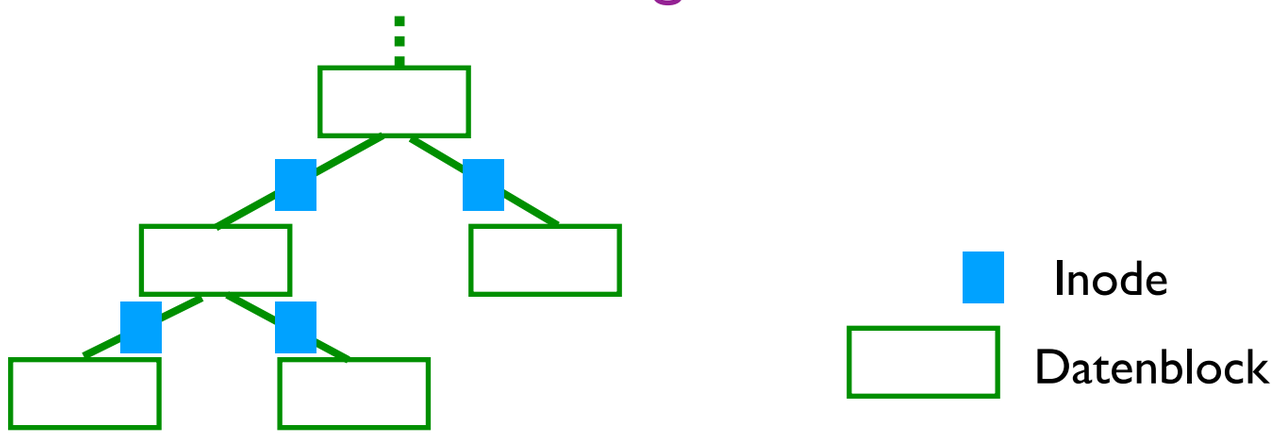
Copy-on-write-Transaktionen

- Bei jeder Änderung im Dateisystem auf der Platte wird neuer Block angelegt. Alter Block bleibt erstmal erhalten.

  ⇒ Änderung zieht sich bis zur Wurzel des Dateisystems durch

  ⇒ Wird erst aktuell, wenn Wurzelknoten geändert

- Vereinfacht:



ZFS-Filesystem

- Abhilfe: Übergang auf ZFS-Filesystem (für verschiedene Unix-Varianten verfügbar)
- Mehrere Neuerungen, z.B.:
  - Copy-on-write-Transaktionen
  - Prüfsummen
  - 128-Bit-Dateiblocknummern ⇒ sehr viel größere Dateien möglich

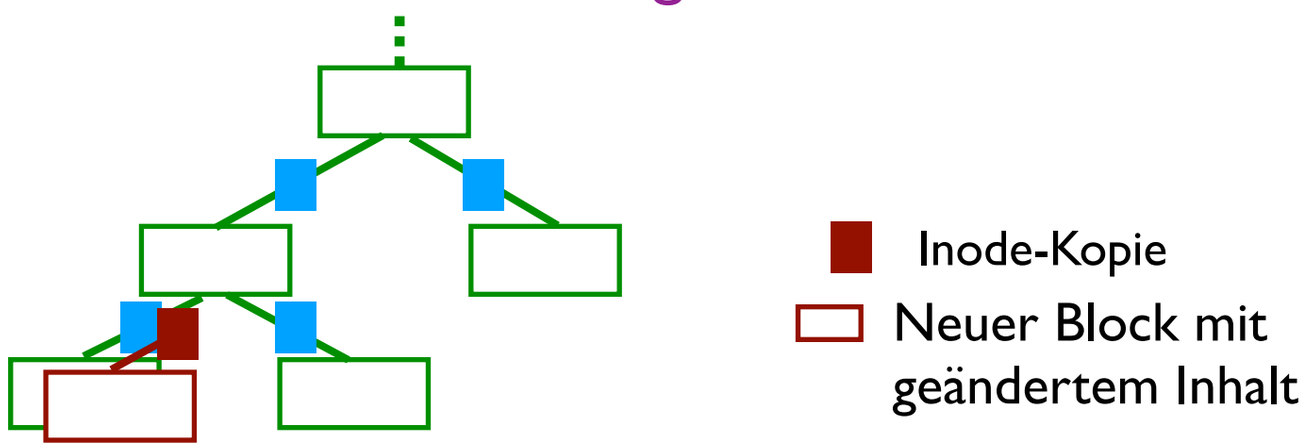
Copy-on-write-Transaktionen

- Bei jeder Änderung im Dateisystem auf der Platte wird neuer Block angelegt. Alter Block bleibt erstmal erhalten.

  ⇒ Änderung zieht sich bis zur Wurzel des Dateisystems durch

  ⇒ Wird erst aktuell, wenn Wurzelknoten geändert

- Vereinfacht:



ZFS-Filesystem

- Abhilfe: Übergang auf ZFS-Filesystem (für verschiedene Unix-Varianten verfügbar)
- Mehrere Neuerungen, z.B.:
  - Copy-on-write-Transaktionen
  - Prüfsummen
  - 128-Bit-Dateiblocknummern ⇒ sehr viel größere Dateien möglich

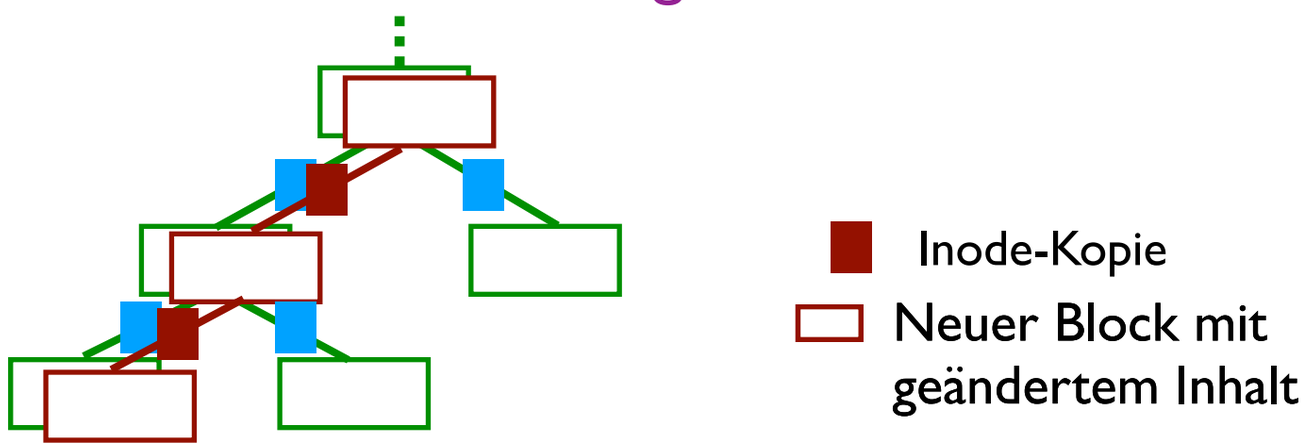
Copy-on-write-Transaktionen

- Bei jeder Änderung im Dateisystem auf der Platte wird neuer Block angelegt. Alter Block bleibt erstmal erhalten.

  ⇒ Änderung zieht sich bis zur Wurzel des Dateisystems durch

  ⇒ Wird erst aktuell, wenn Wurzelknoten geändert

- Vereinfacht:



ZFS-Filesystem

- Abhilfe: Übergang auf ZFS-Filesystem (für verschiedene Unix-Varianten verfügbar)
- Mehrere Neuerungen, z.B.:
  - Copy-on-write-Transaktionen
  - Prüfsummen
  - 128-Bit-Dateiblocknummern ⇒ sehr viel größere Dateien möglich

Copy-on-write-Transaktionen

- Bei jeder Änderung im Dateisystem auf der Platte wird neuer Block angelegt. Alter Block bleibt erstmal erhalten.

  ⇒ Änderung zieht sich bis zur Wurzel des Dateisystems durch

  ⇒ Wird erst aktuell, wenn Wurzelknoten geändert

- Bei Systemzusammenbruch alte Version noch verfügbar (Transaktionsorientierung)
- Bei Bedarf alte Version auch noch länger verfügbar (Snapshot)
- Nach einigen Änderungsrunden alte Versionen löschen

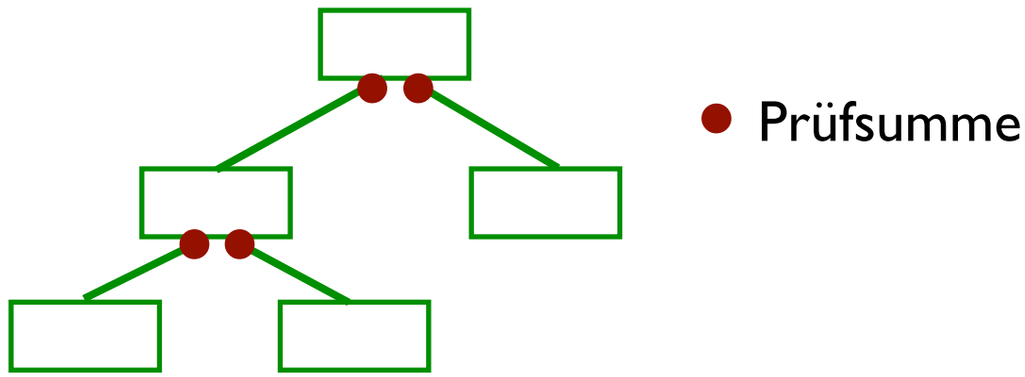
Prüfsummen

- Klassisch jeder Plattenblock durch Prüfsumme geschützt

  ⇒ Eigenschaft der Platte

  ⇒ Kann aus Dateisystem heraus nicht überprüft werden

- Nun: Zusätzlich Prüfsummen auf Dateisystemebene
- Vereinfacht:



- Bei Inkonsistenzen von gespiegelten Platten kann korrektes Exemplar ermittelt werden (selbst wenn Platte keinen Fehler meldet)

  ⇒ automatisch neue Kopie erstellen

- Ggf. regelmäßige automatische Checks des Dateisystems durchführen

Fragen – Teil 5

- Welche wesentlichen Eigenschaften hat das *ZFS-Filesystem*?

Zusammenfassung

- Blockgeräte vs. Charactergeräte
- Organisation einer Platte und Auswirkungen auf Dateisystem und Zugriffsstrategien
- Organisation einer SSD
- Konsistenzprüfung von Dateisystemen
- Log-based File Systems vs. ZFS-Filesystem

Geräteverwaltung 1 / Dateiverwaltung 3 – Fragen

1. Wie ist eine Platte intern organisiert?
2. Wie wirkt sich dies auf den Informationszugriff aus?
3. Wie geht das Unix *Fast File System* damit um?
4. Was versteht man beim Zugriff auf Plattenblöcke unter dem *Fahrstuhlalgorithmus*?
5. Welche wesentlichen Eigenschaften hat eine SSD *(Solid State Disk)*?
6. Nenne Beispiele für potentielle Inkonsistenzen in einem Dateisystem. Wie können sie entstehen? Wie kann man sie erkennen/beheben?
7. Welche wesentlichen Eigenschaften hat das *ZFS-Filesystem*?