

PVK Technische Informatik 2

(Teil 2)



Michael Wolf

21.06.2016



Montag

- Kommunikation
 - Link Layer (+ Security)
 - Network Layer
 - Transport Layer
 - Application Layer
- Markov-Ketten

Dienstag

- Speicher
 - Storage
 - Dictionaries & Hashing
 - Datenbanken & SQL
- Concurrency & Locks

Grafisches Material, welches nicht selber erstellt wurde und keine weitere Referenzangabe besitzt, stammt aus dem Vorlesungsmaterial von Prof. Dr. R. Wattenhofer.



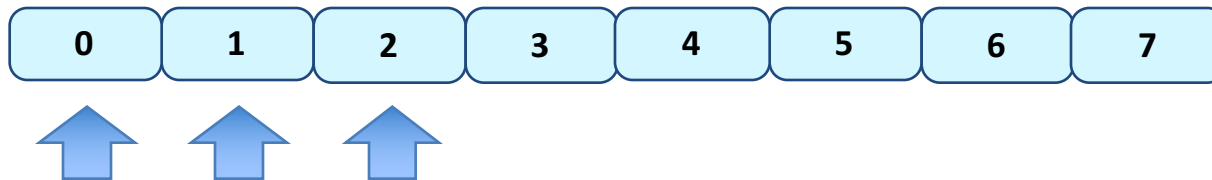
Inhalt

- Hard Disk Drives
- Disk Scheduling
- Flash-Based Solid-State Drives
- Files und Directories



Hard Disk Drives

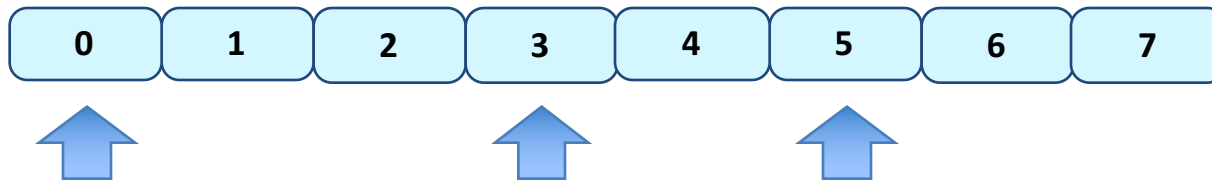
- Disk mit grosser Anzahl Sektoren
- **Sektor** → 512-Byte Block
 - Kann beschrieben und gelesen werden
 - Durchnummeriert von 0 bis $n - 1$ (Adressraum)
- **Zugriff**
 - Sequentiell (schnell)
 - Random Access (langsamer)





Hard Disk Drives

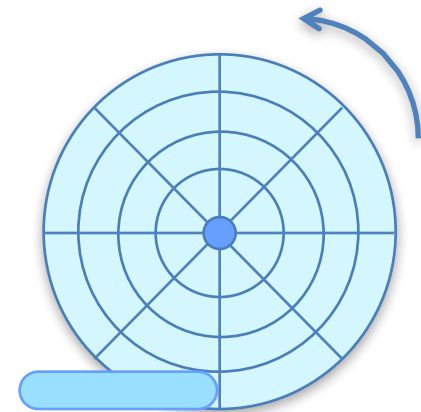
- Disk mit grosser Anzahl Sektoren
- **Sektor** → 512-Byte Block
 - Kann beschrieben und gelesen werden
 - Durchnummeriert von 0 bis $n - 1$ (Adressraum)
- **Zugriff**
 - Sequentiell (schnell)
 - Random Access (langsamer)





Hard Disk Drives

- **Platten** → Eigentlicher Träger der Daten
- **Surface** → Eine Seite des Platters
- **Rotation** → Rotations per minute (RPM)
 - Zeit für eine Umdrehung in ms: $60'000 / \text{RPM}$
- **Track** → Zyklische Anordnung der Sektoren auf Platte
- **Disk-Kopf** → Schreibt und liest





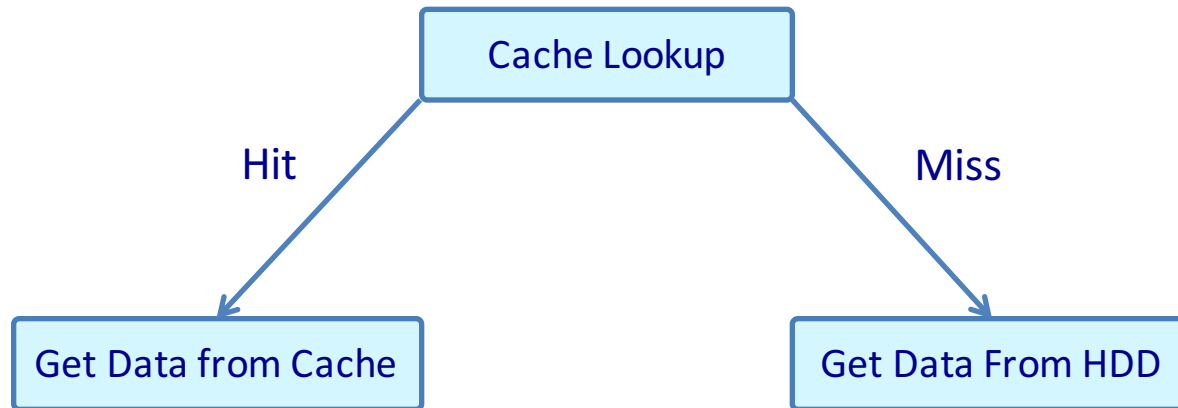
Hard Disk Drives

- **Seek Time** → Richtige Spur finden
 - Meist absolut angegeben
- **Rotational Delay** → Richtigen Sektor auf der Spure finden
 - Durchschnitt: $R/2$
 - RPM umrechnen ($60'000 : \text{RPM} = \text{Delay in ms}$)
- **Transfer Time** → Daten schreiben oder lesen
 - Datenmenge / Transferrate
- **Cache** → Schnellerer, kleinerer Speicher
 - Cache Hit / Miss
 - Zugriff auf HDD nur bei Cache Miss

$$T_{I/O} = T_{\text{seek}} + T_{\text{rotation}} + T_{\text{transfer}}$$



Hard Disk Drives





Disk Scheduling

- In welcher Reihenfolge sollen die Requests abgearbeitet werden?
- **Shortest-Seek-Time-First (SSTF)** → Naheste Spur wählen
- **Nearest-Block-First (NBF)** → Geringste Abweichung in der Adresse des Blocks
- **SCAN** → Bewegt sich hin und her auf der Disk
- **C-SCAN** → Bewegt sich nur von aussen nach innen
- **F-SCAN** → Friert die Warteschlange ein (vermeidet Starvation)
- **Shortest Positioning Time First** → Berücksichtigt Seek Time und Rotational Delay



Disk Scheduling

- **Shortest-Seek-Time-First (SSTF)** → Naheste Spur wählen

Innenseite

0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39

← Plattenrotation

Aussenseite



Disk Scheduling

- **Shortest-Seek-Time-First (SSTF)** → Naheste Spur wählen

Innenseite

0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39

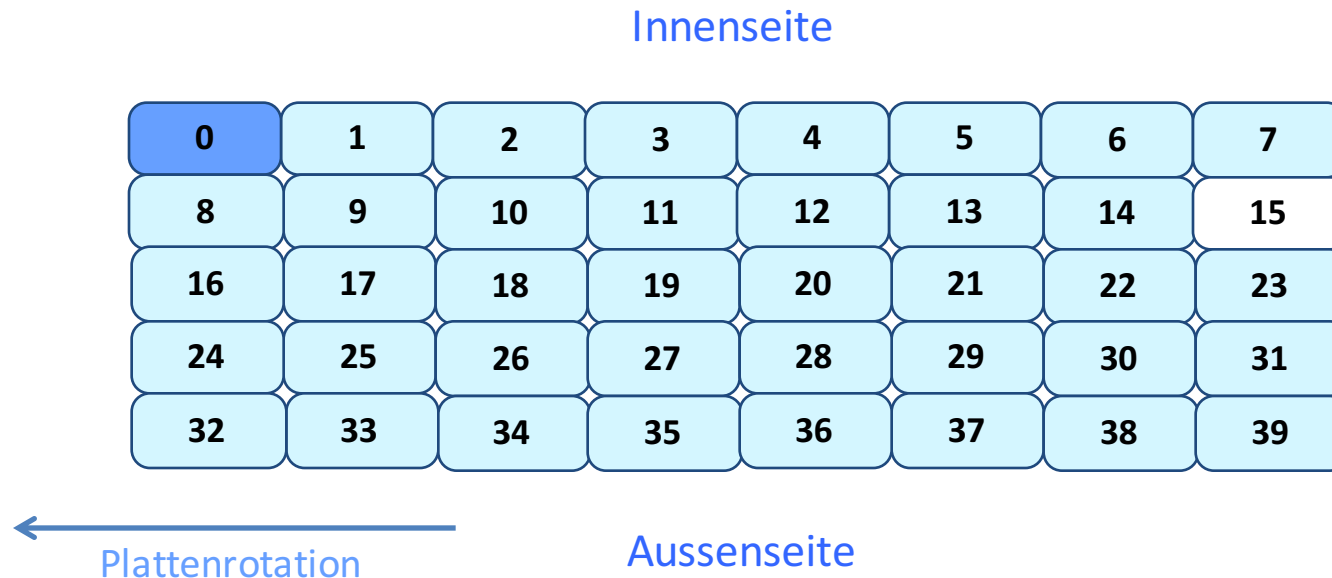
← Plattenrotation

Aussenseite



Disk Scheduling

- **Shortest-Seek-Time-First (SSTF)** → Naheste Spur wählen





Disk Scheduling

- **Shortest-Seek-Time-First (SSTF)** → Naheste Spur wählen

Innenseite

0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39

← Plattenrotation

Aussenseite



Disk Scheduling

- **Shortest-Seek-Time-First (SSTF)** → Naheste Spur wählen

Innenseite

0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39

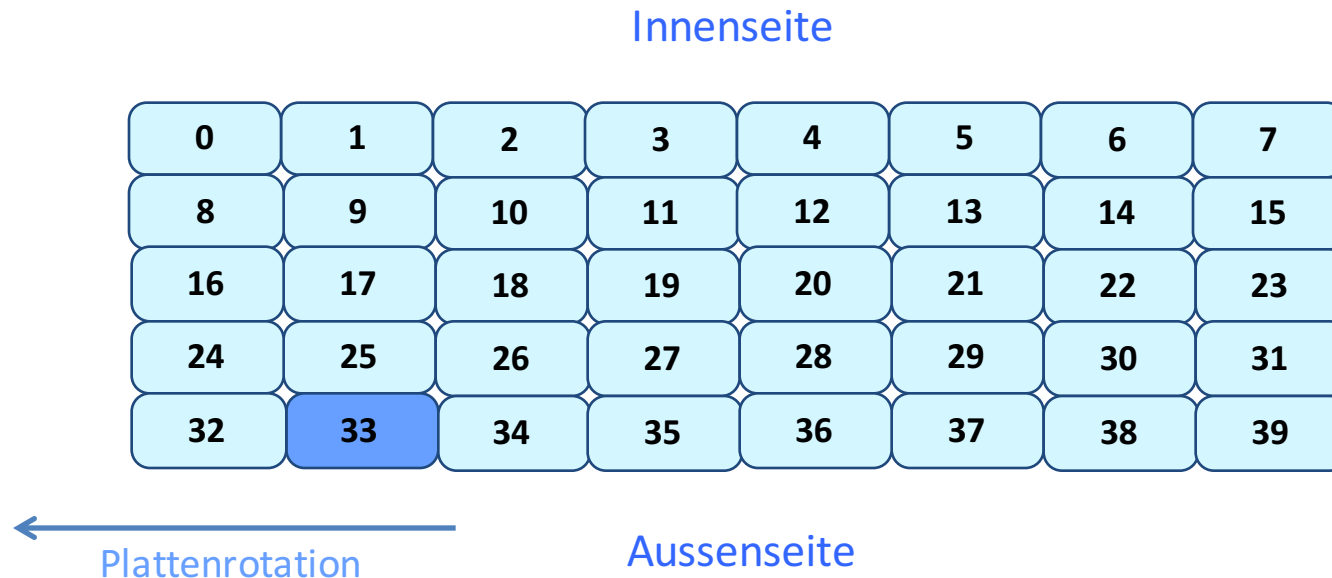
← Plattenrotation

Aussenseite



Disk Scheduling

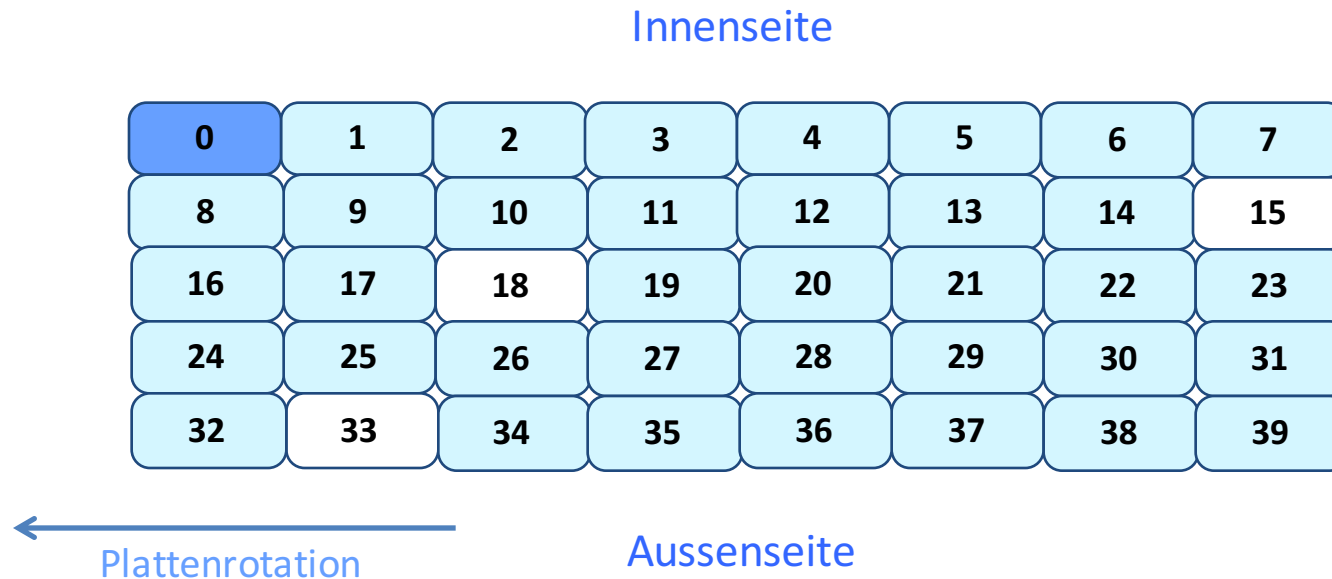
- **Shortest-Seek-Time-First (SSTF)** → Naheste Spur wählen





Disk Scheduling

➤ SCAN





Disk Scheduling

➤ SCAN

Innenseite

0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39

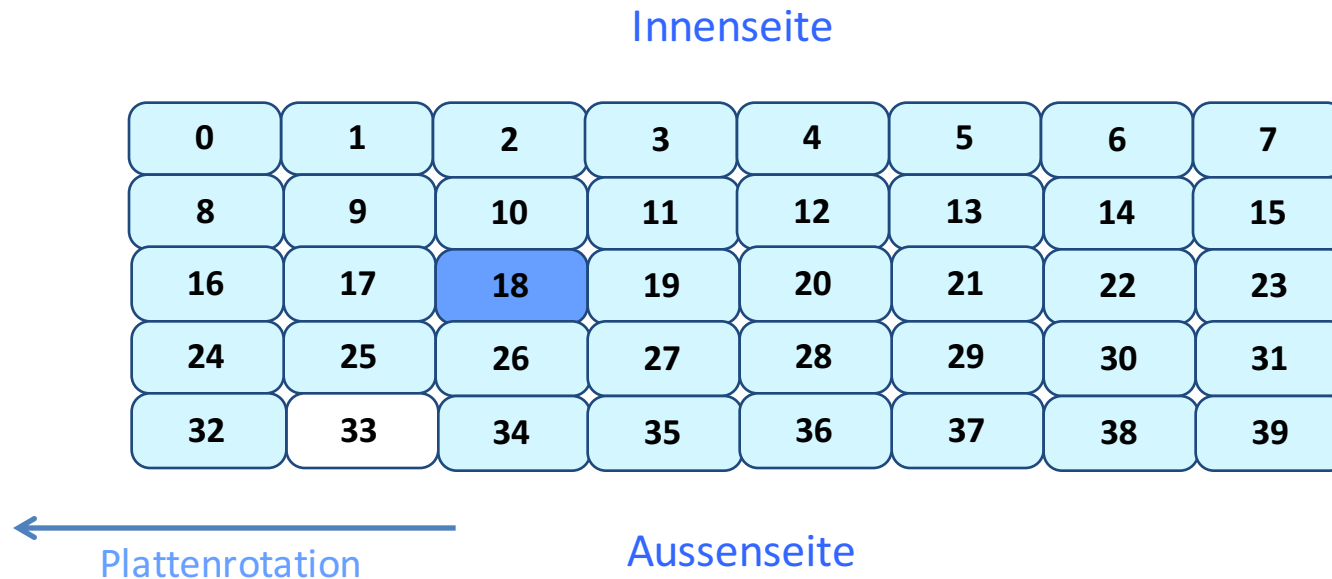
← Plattenrotation

Aussenseite



Disk Scheduling

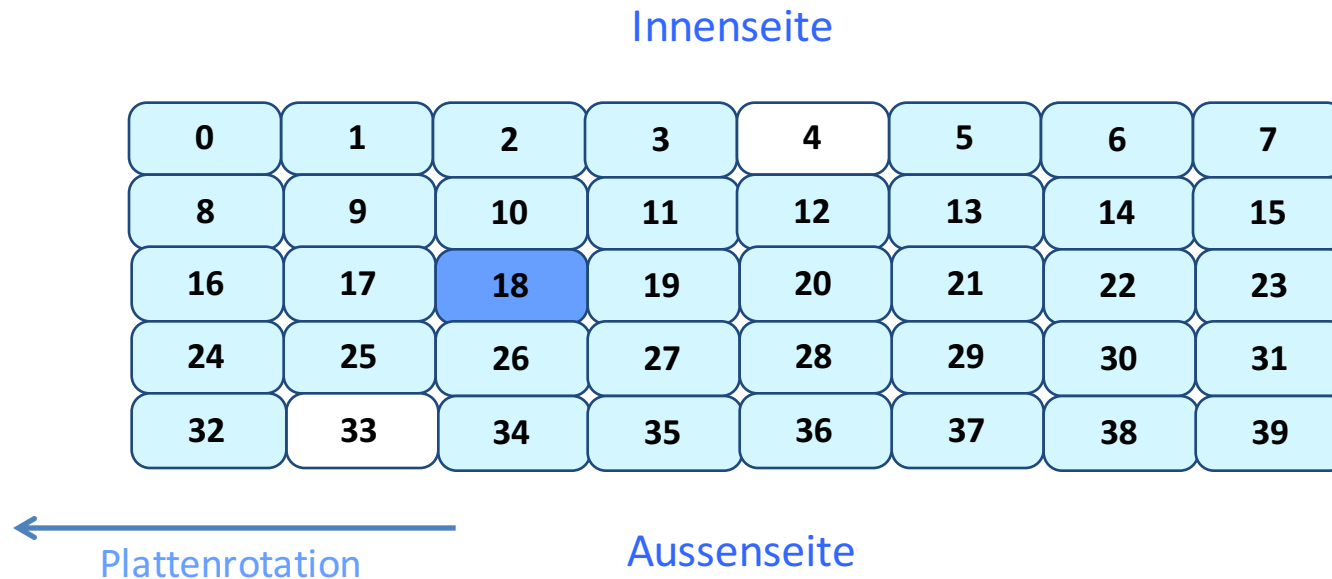
➤ SCAN





Disk Scheduling

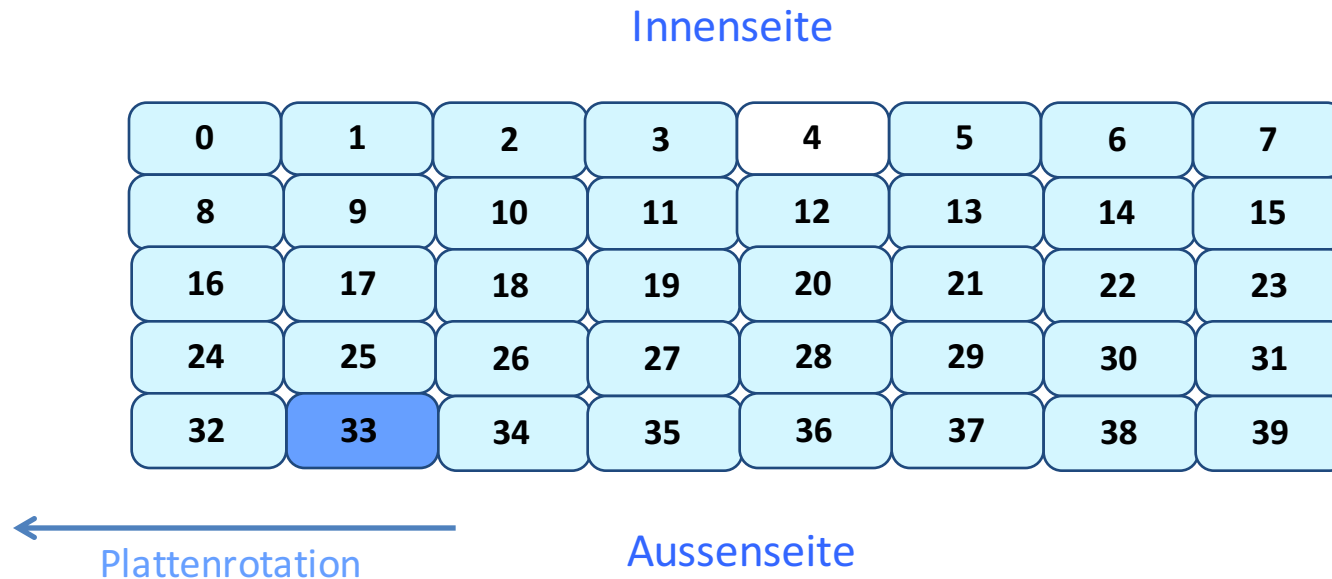
➤ SCAN





Disk Scheduling

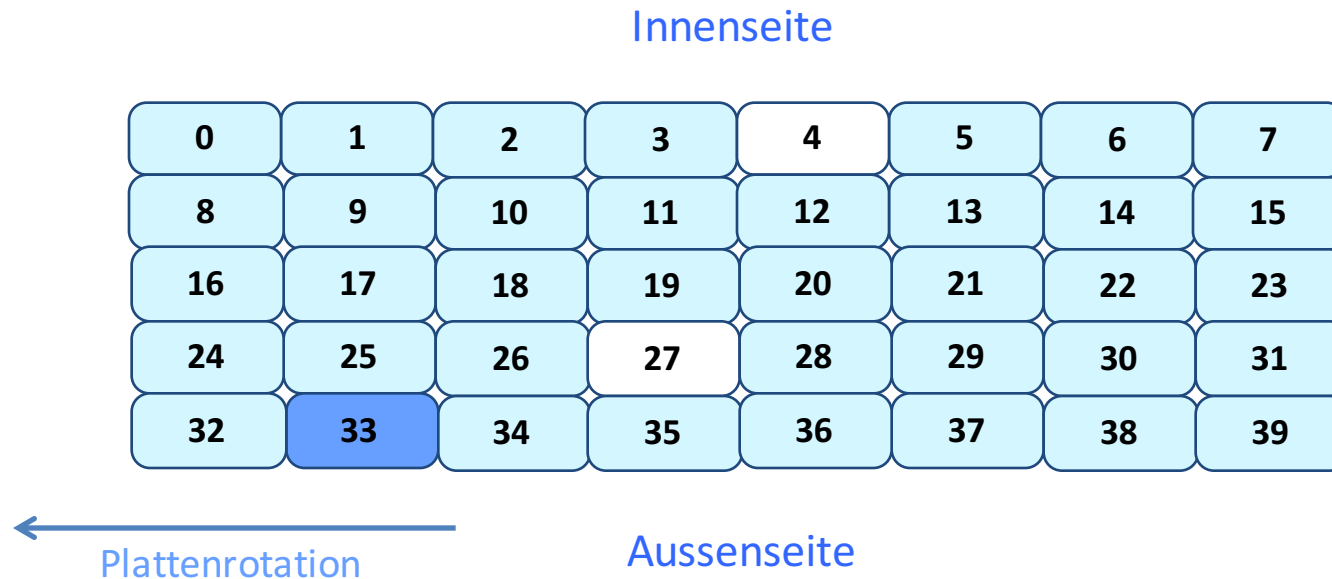
➤ SCAN





Disk Scheduling

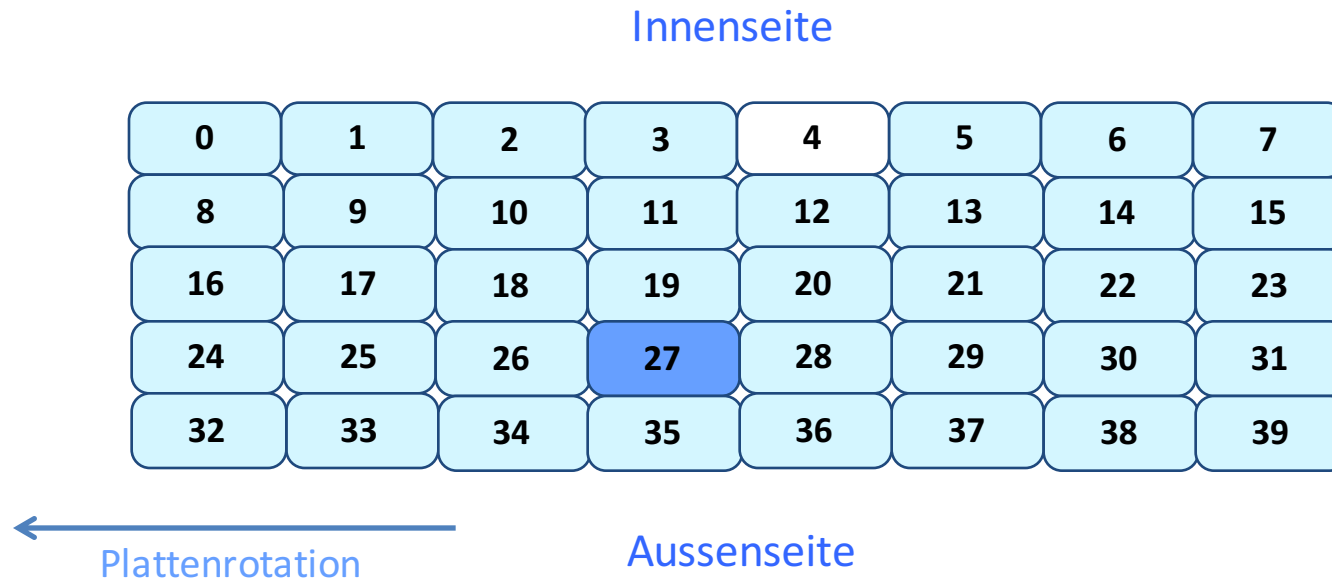
➤ SCAN





Disk Scheduling

➤ SCAN

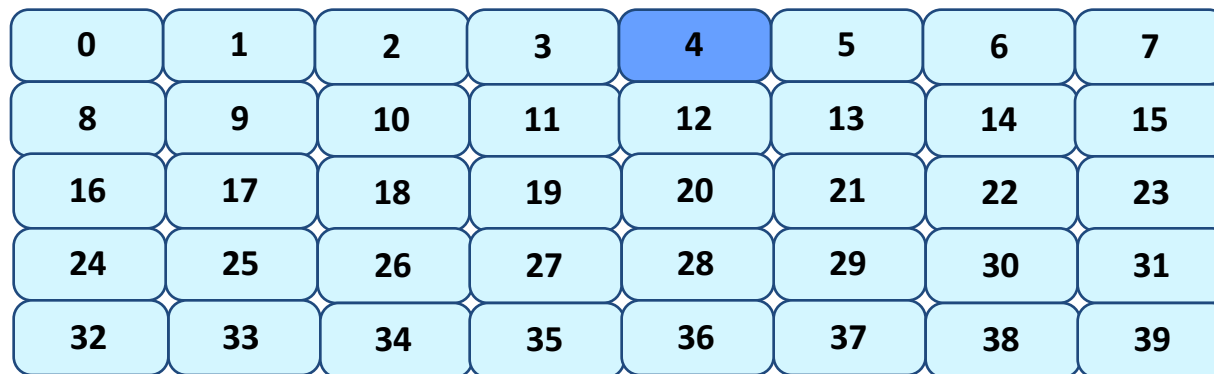




Disk Scheduling

➤ SCAN

Innenseite



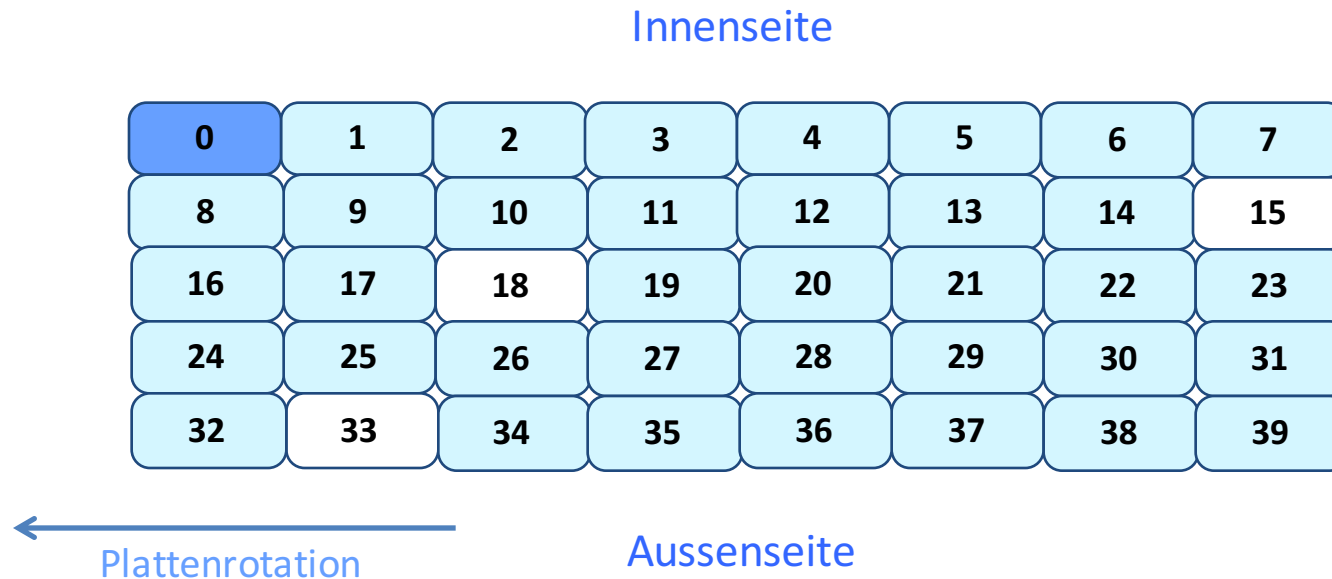
← Plattenrotation

Aussenseite



Disk Scheduling

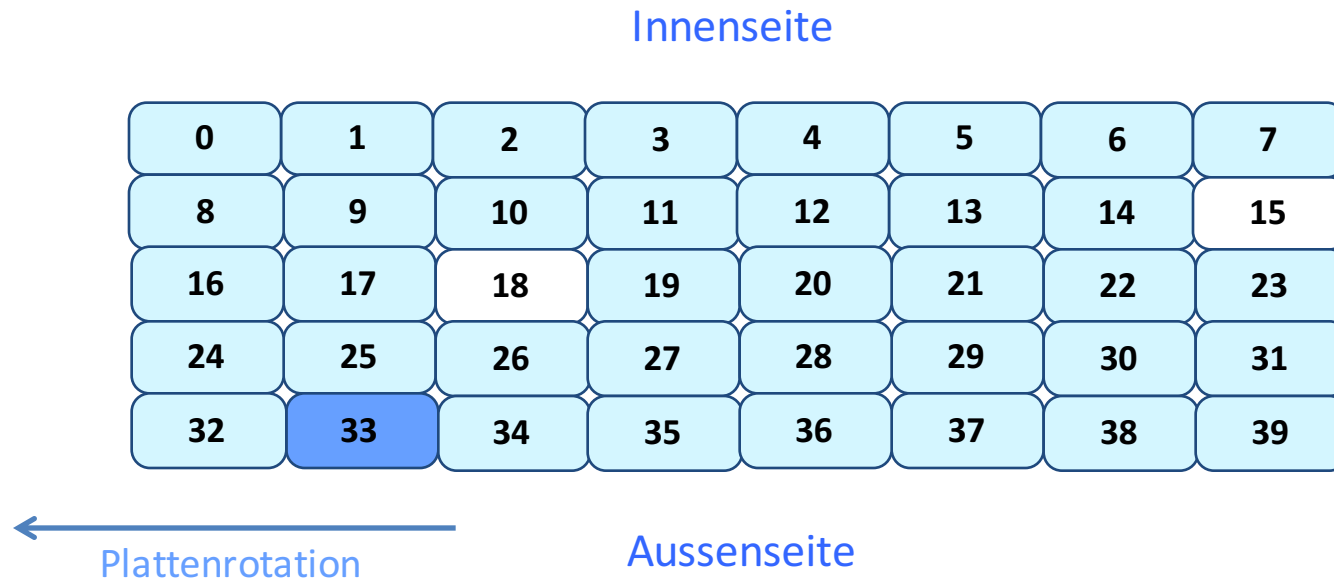
➤ C-SCAN





Disk Scheduling

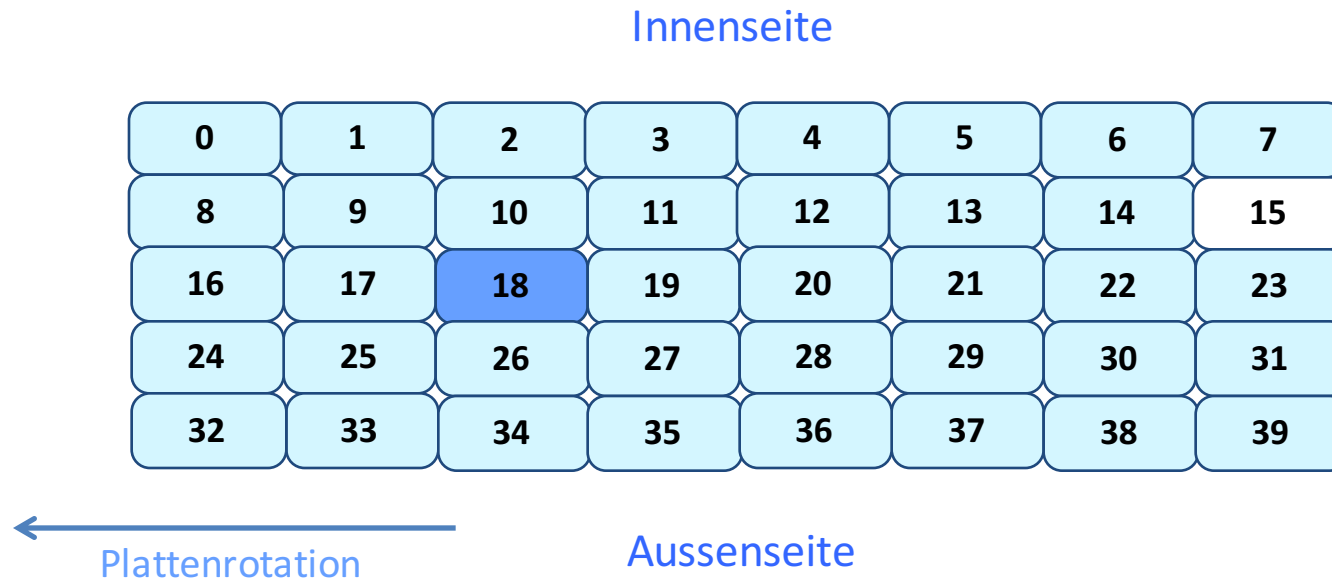
➤ C-SCAN





Disk Scheduling

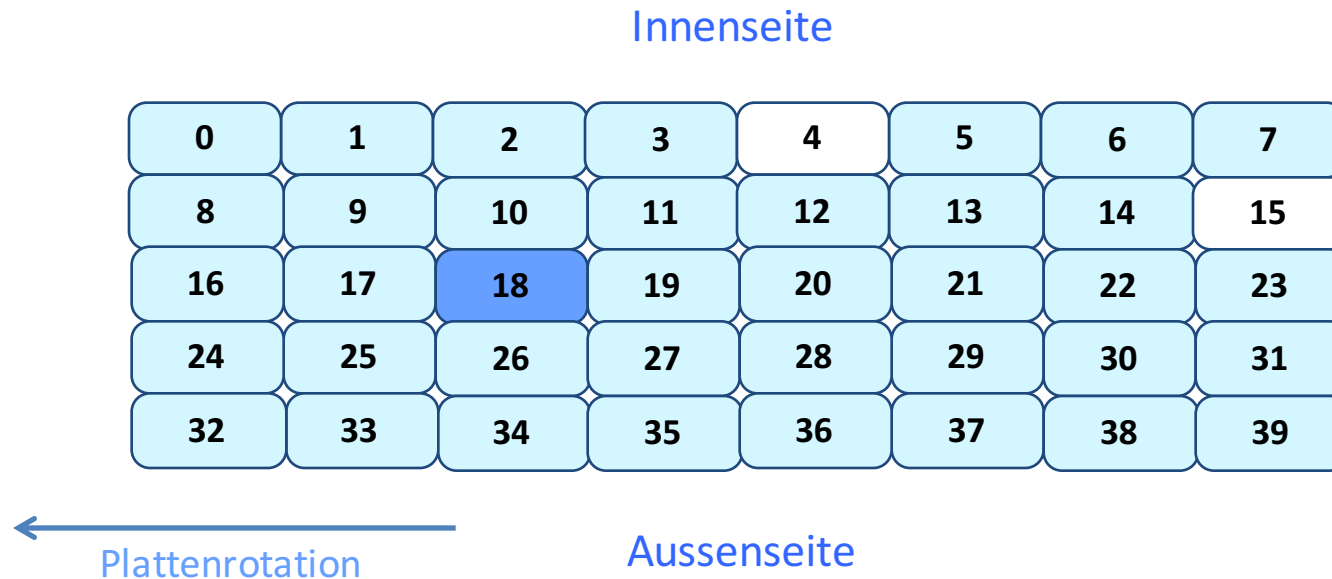
➤ C-SCAN





Disk Scheduling

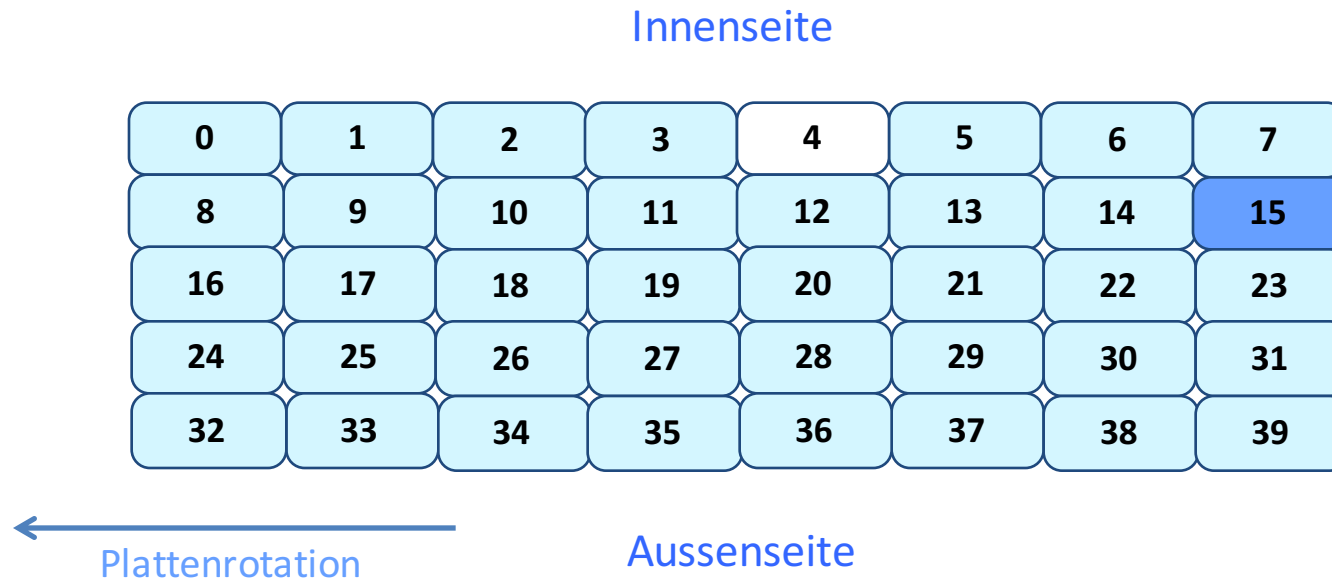
➤ C-SCAN





Disk Scheduling

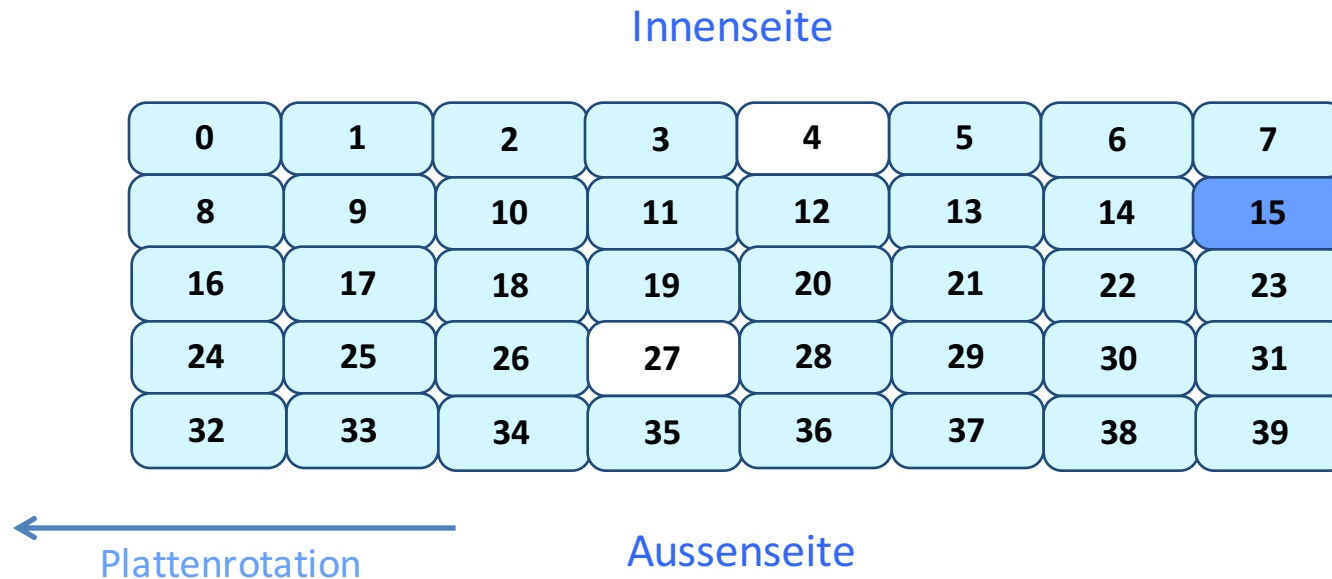
➤ C-SCAN





Disk Scheduling

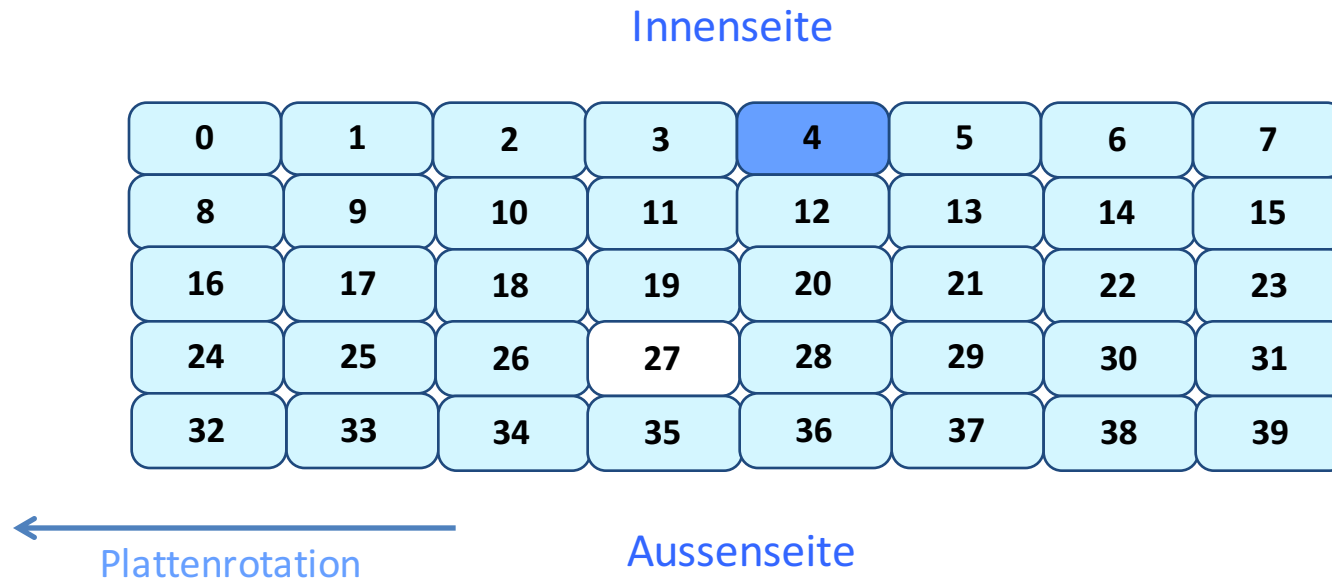
➤ C-SCAN





Disk Scheduling

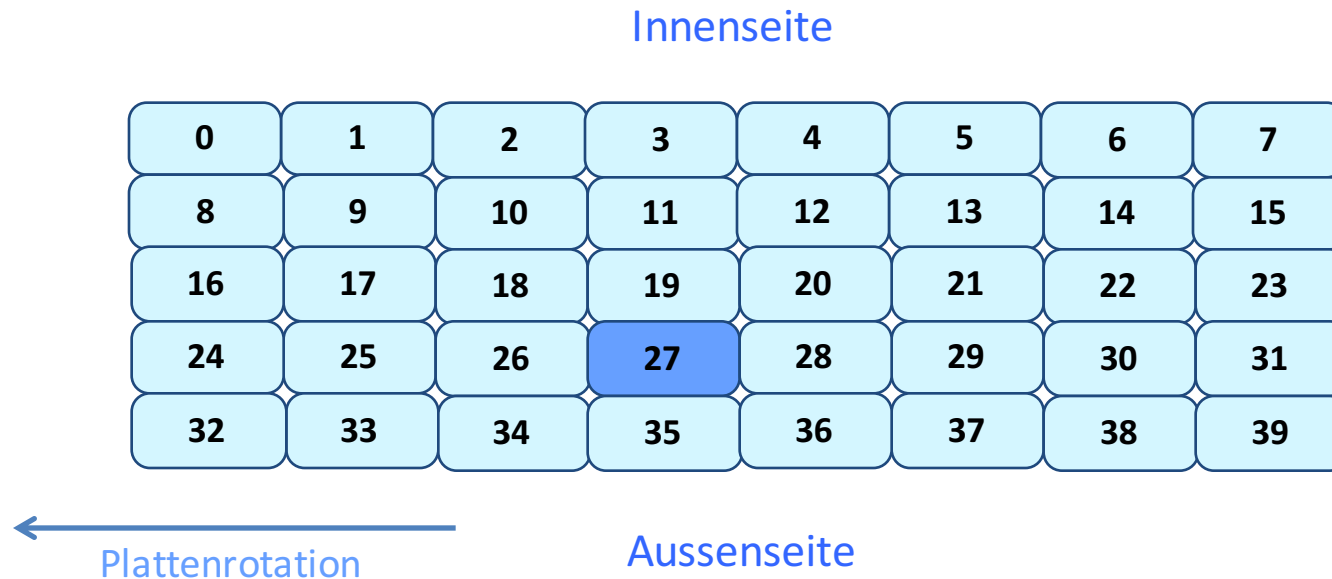
➤ C-SCAN





Disk Scheduling

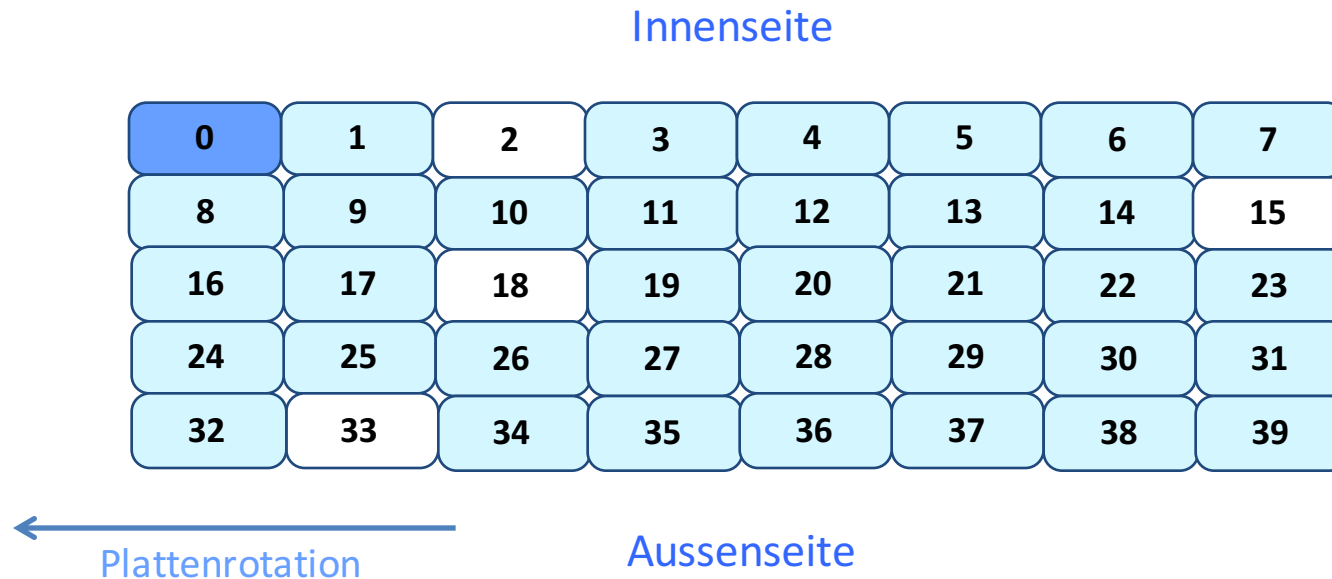
➤ C-SCAN





Disk Scheduling

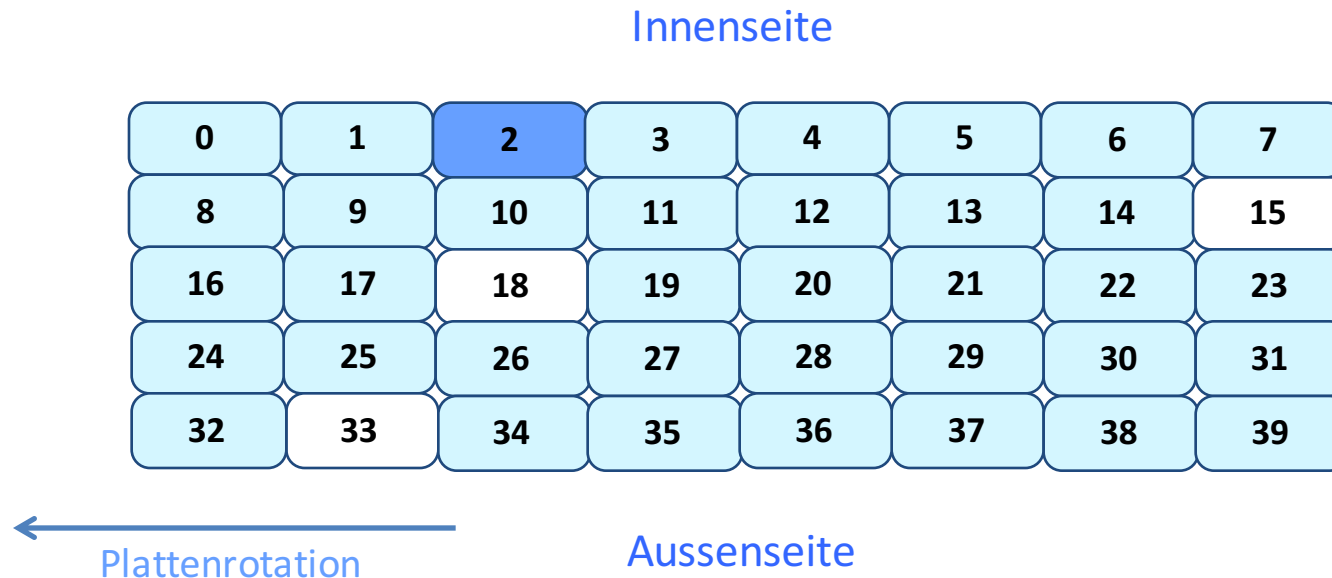
- **SCAN** → Problem Starvation





Disk Scheduling

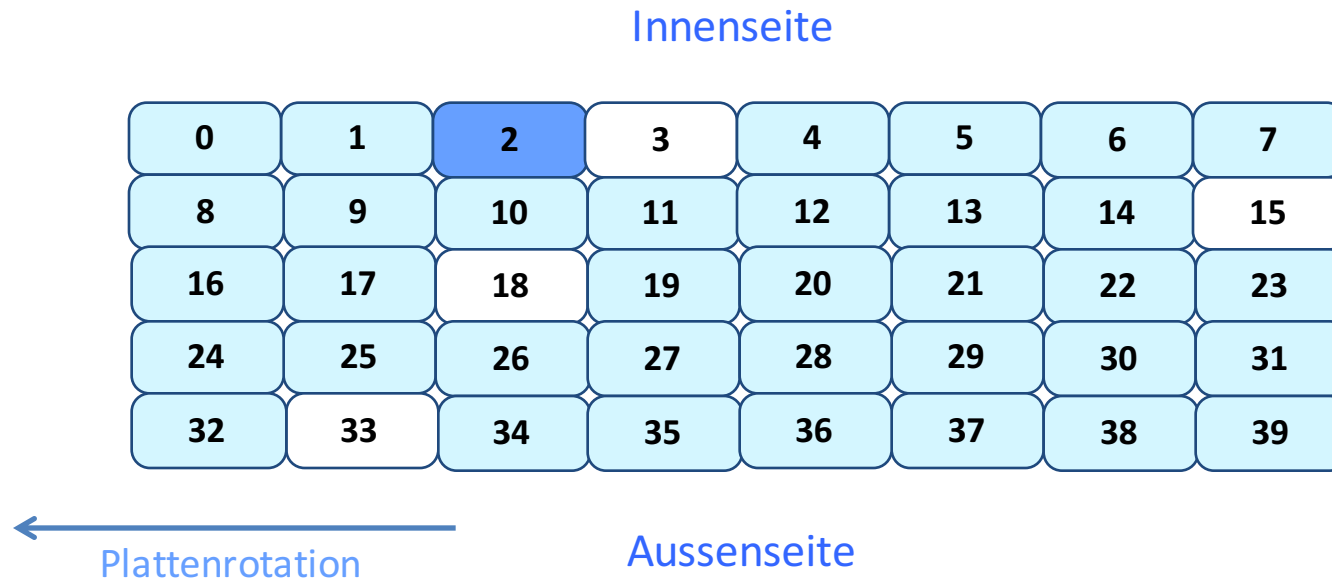
- **SCAN** → Problem Starvation





Disk Scheduling

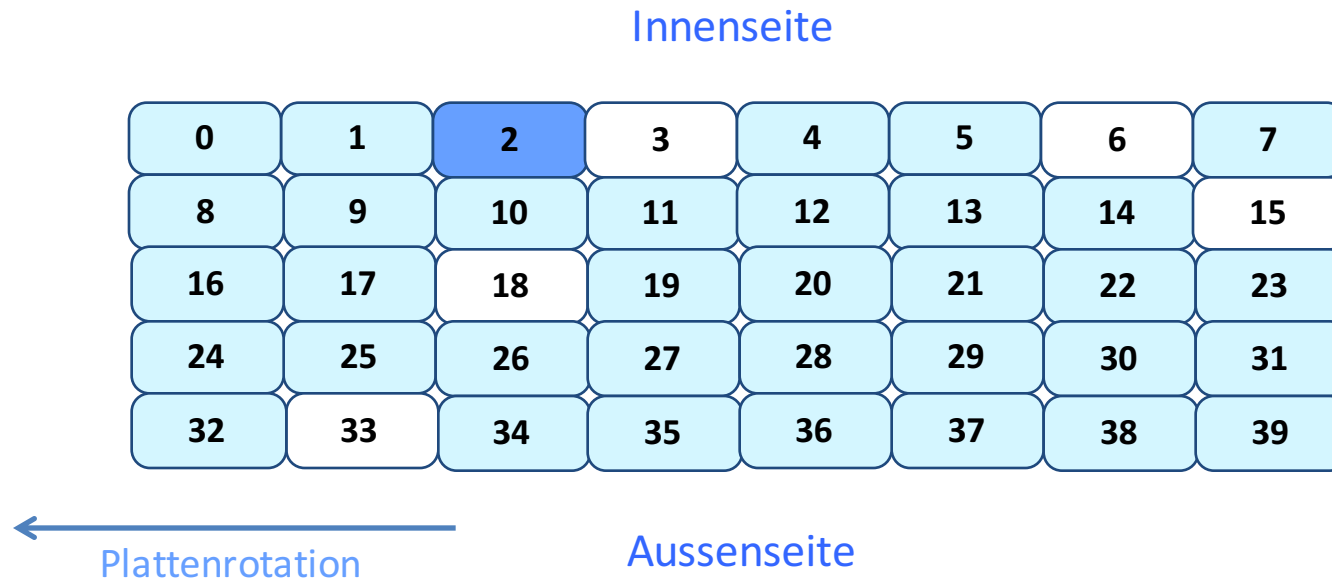
- **SCAN** → Problem Starvation





Disk Scheduling

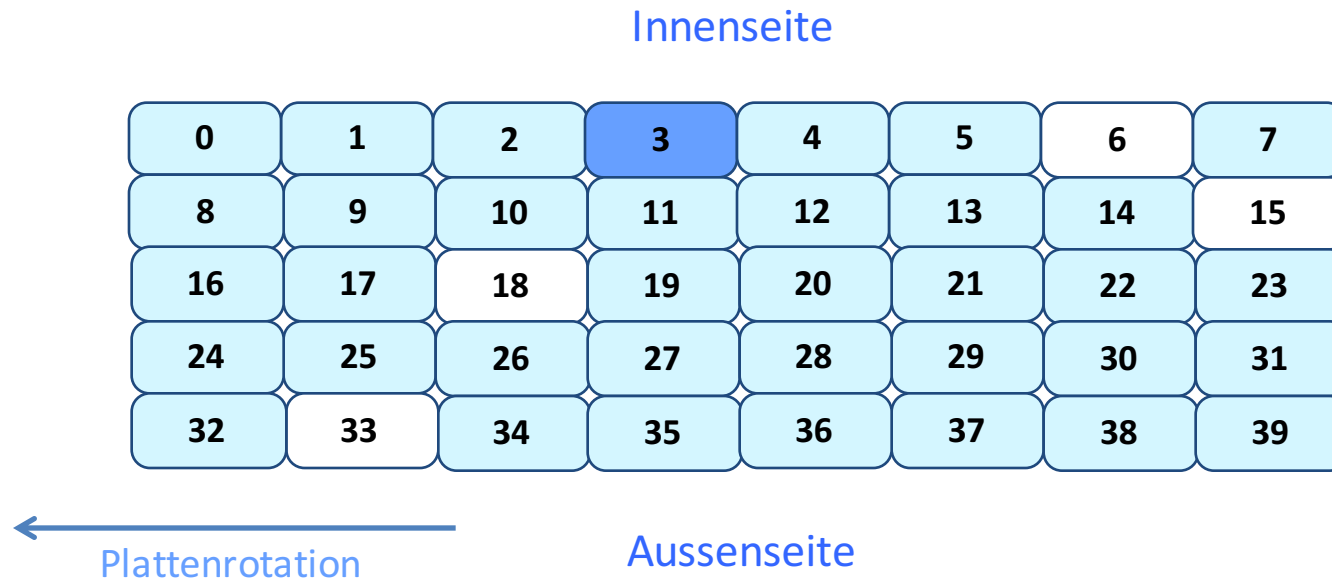
- **SCAN** → Problem Starvation





Disk Scheduling

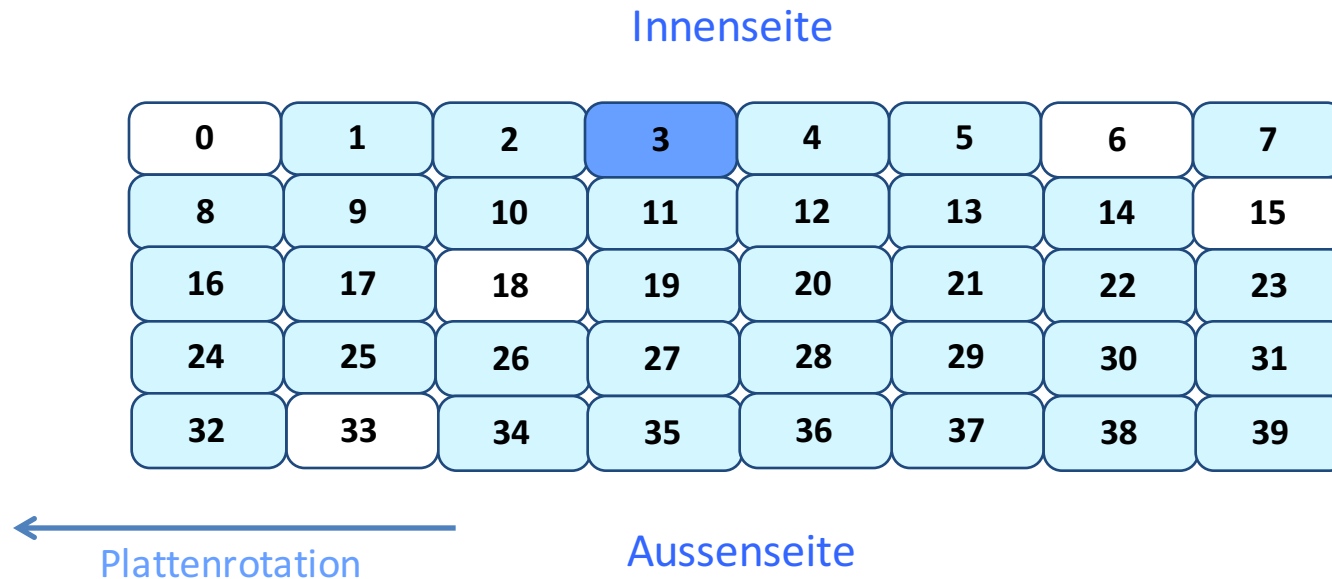
- **SCAN** → Problem Starvation





Disk Scheduling

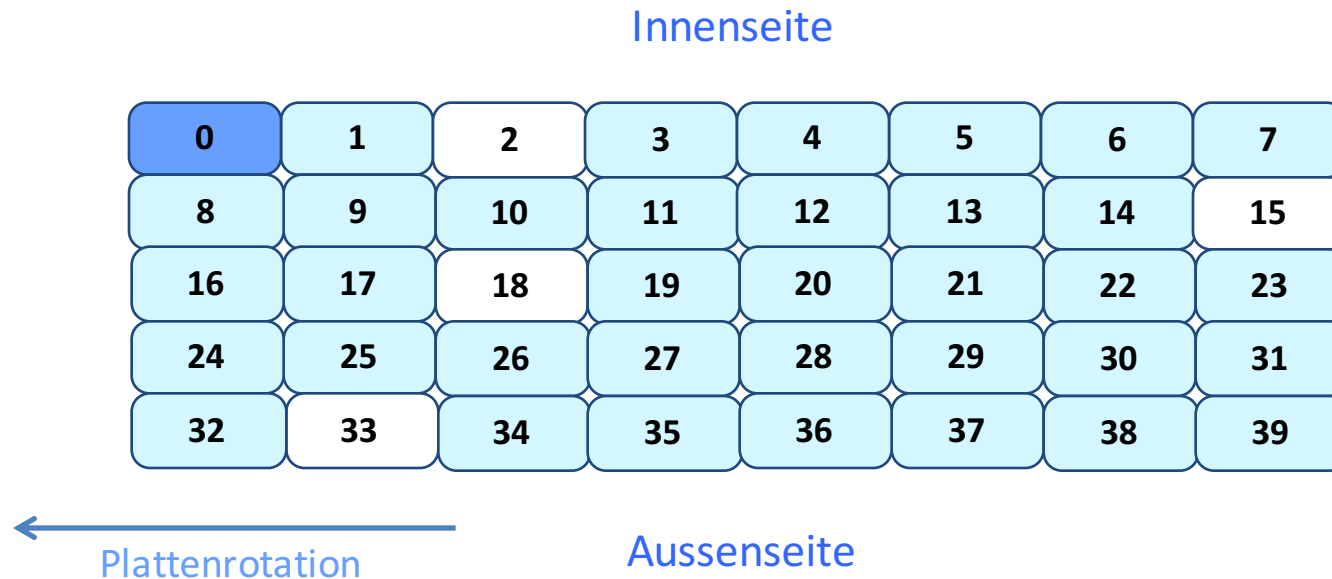
- **SCAN** → Problem Starvation





Disk Scheduling

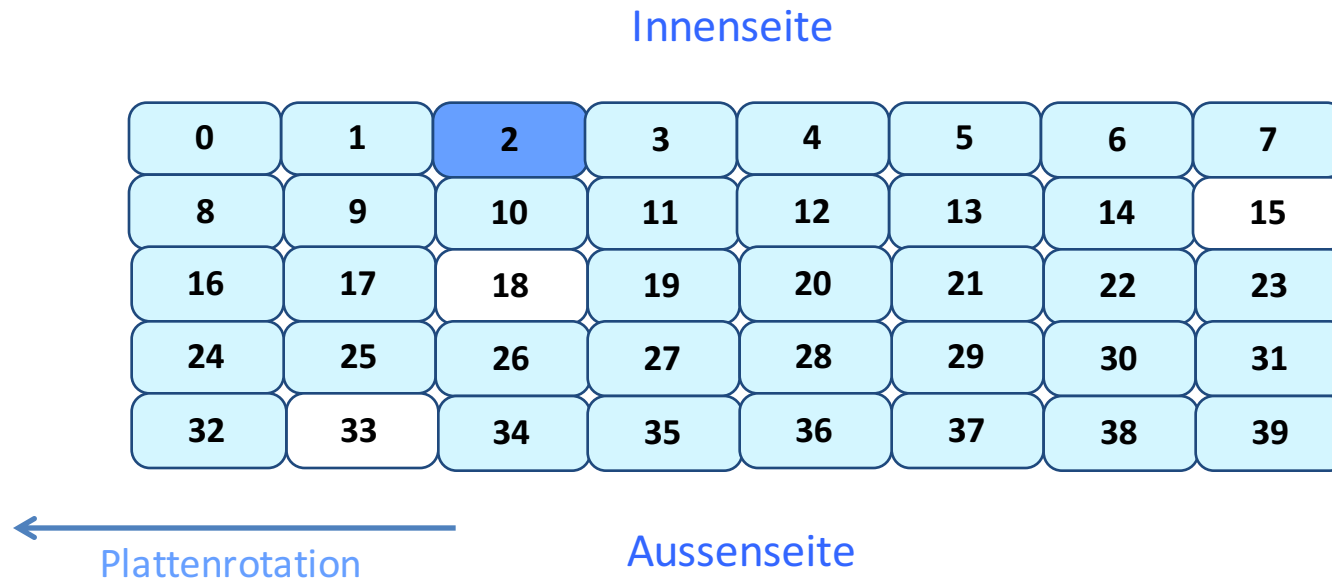
- **F-SCAN** → Lösung Freeze Scan





Disk Scheduling

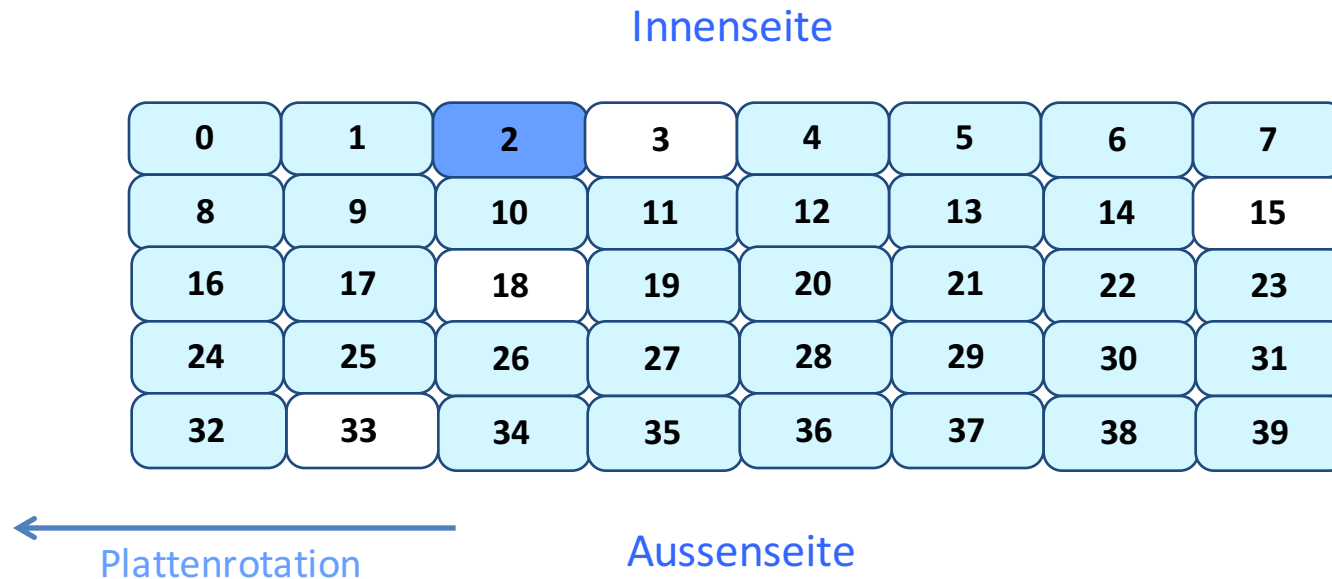
- **F-SCAN** → Lösung Freeze Scan





Disk Scheduling

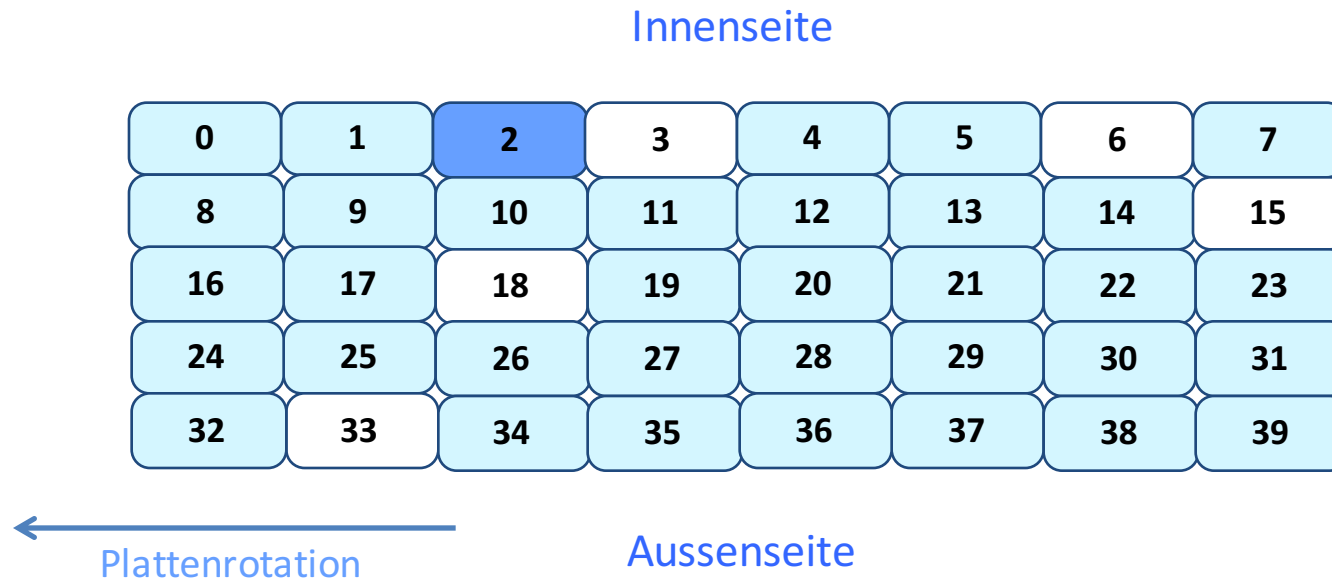
- **F-SCAN** → Lösung Freeze Scan





Disk Scheduling

- **F-SCAN** → Lösung Freeze Scan





Disk Scheduling

- **F-SCAN** → Lösung Freeze Scan

Innenseite

0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39

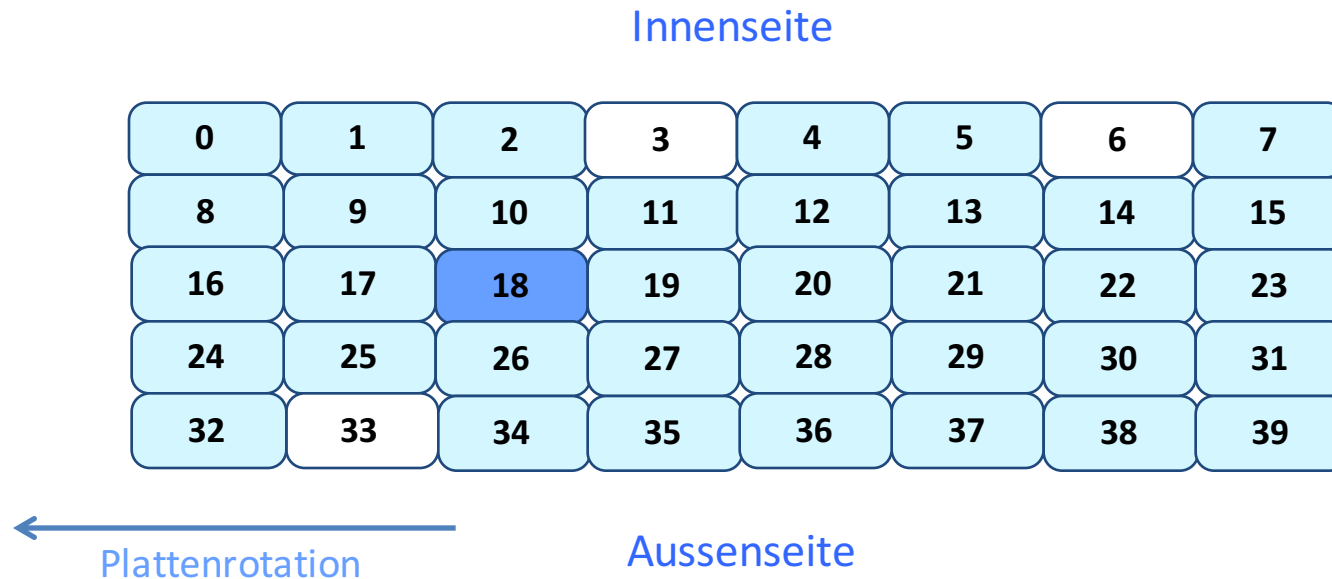
← Plattenrotation

Aussenseite



Disk Scheduling

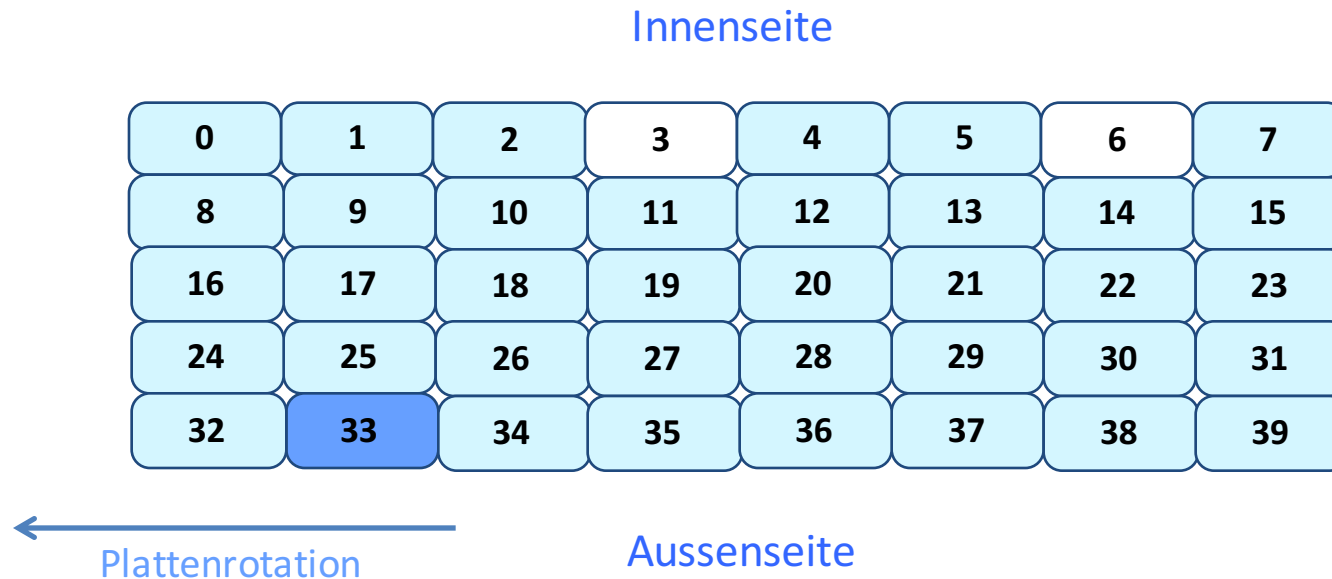
- **F-SCAN** → Lösung Freeze Scan





Disk Scheduling

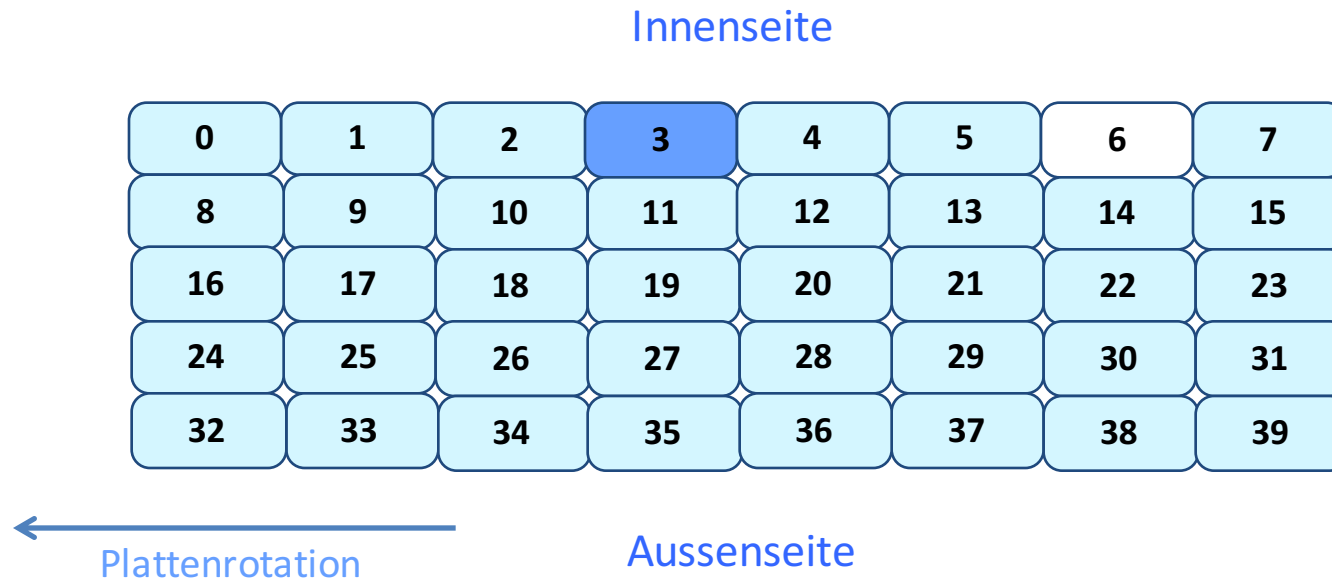
- **F-SCAN** → Lösung Freeze Scan





Disk Scheduling

- **F-SCAN** → Lösung Freeze Scan





Disk Scheduling

- **F-SCAN** → Lösung Freeze Scan

Innenseite

0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39

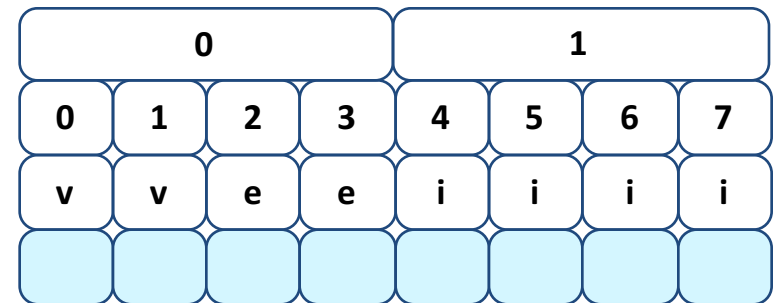
← Plattenrotation

Aussenseite



Flash-Based SSDs

- SSD → Solid-State Storage
- Organisiert in **Blocks** und **Pages**
- 3 mögliche Operationen
 - Read (Page): Random Access
 - Erase (Block)
 - Program (Page)
- 3 mögliche Zustände einer Page
 - Invalid (Anfangszustand)
 - Erased
 - Valid





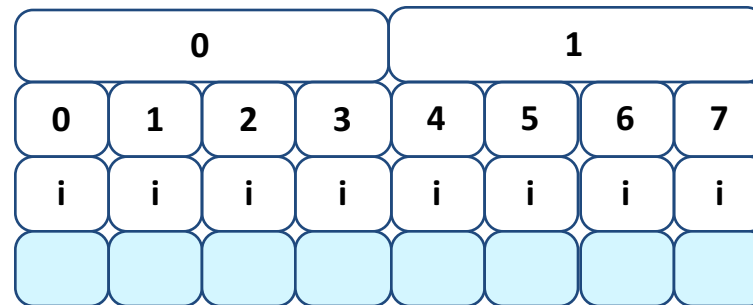
Flash-Based SSDs

- **Flash Translation Layer (FTL)**
 - Übersetze logische Requests in Read, Erase and Program
- **Direct Mapped** → Logische Adresse = Physikalische Adresse
 - Pages müssen gesucht, erased und geschrieben werden
- **Log-Structured FTL** → Benutzt „Logging“
 - Nächster freier Platz im Speicher wird verwendet
 - Benötigt Mapping Table (Page / Block Level): Kann sehr gross werden
 - Benötigt Garbage Collector: „Dead Blocks“ finden
- **Hybrid Mapping** → Kombination aus Page und Block Level Mapping



Flash-Based SSDs

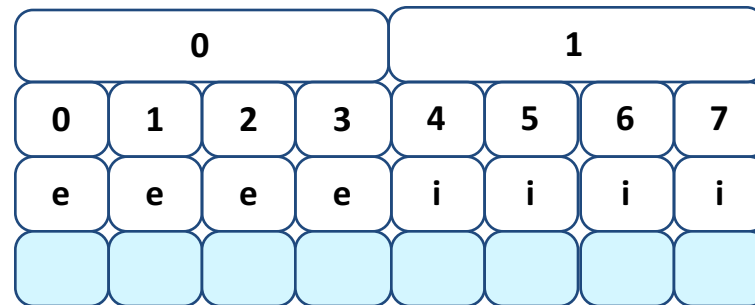
- 3, 4, 0 (alles Schreibzugriffe)
- Direct Mapped





Flash-Based SSDs

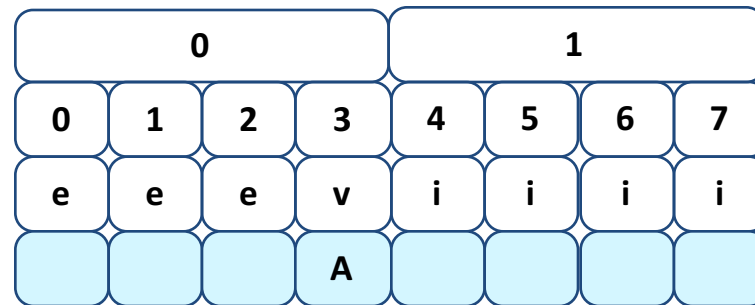
- 3, 4, 0 (alles Schreibzugriffe)
- Direct Mapped





Flash-Based SSDs

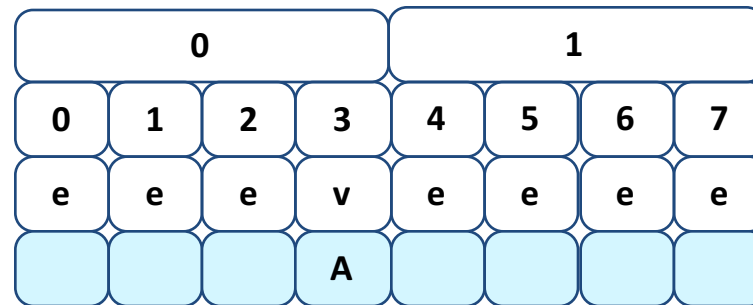
- 3, 4, 0 (alles Schreibzugriffe)
- Direct Mapped





Flash-Based SSDs

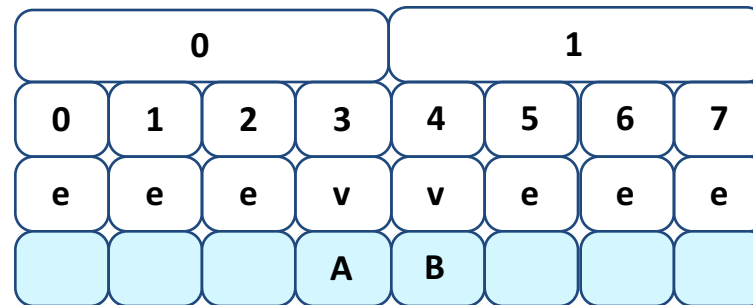
- 3, 4, 0 (alles Schreibzugriffe)
- Direct Mapped





Flash-Based SSDs

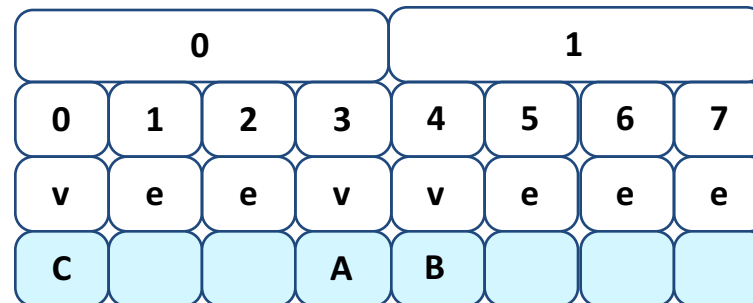
- 3, 4, 0 (alles Schreibzugriffe)
- Direct Mapped





Flash-Based SSDs

- 3, 4, 0 (alles Schreibzugriffe)
- Direct Mapped

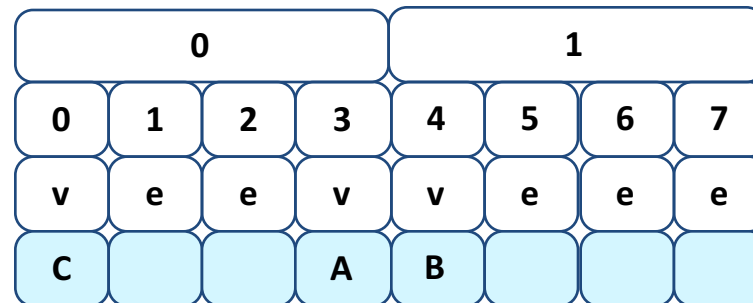




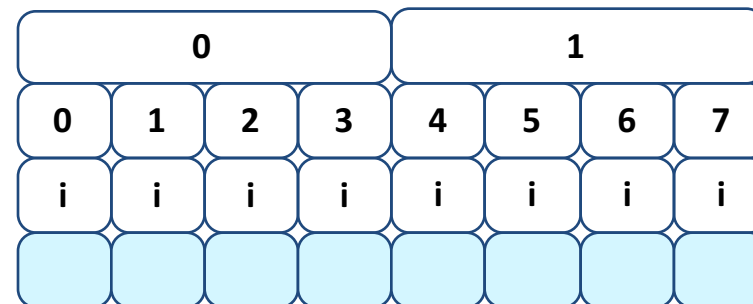
Flash-Based SSDs

➤ 3, 4, 0 (alles Schreibzugriffe)

➤ Direct Mapped



➤ Log Structure

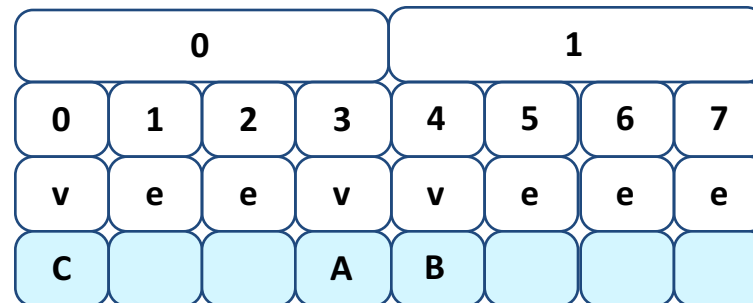




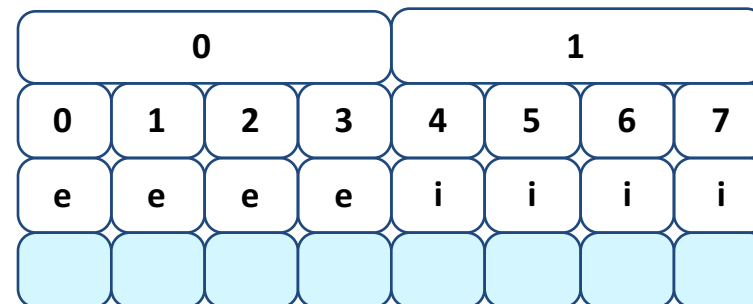
Flash-Based SSDs

➤ 3, 4, 0 (alles Schreibzugriffe)

➤ Direct Mapped



➤ Log Structure

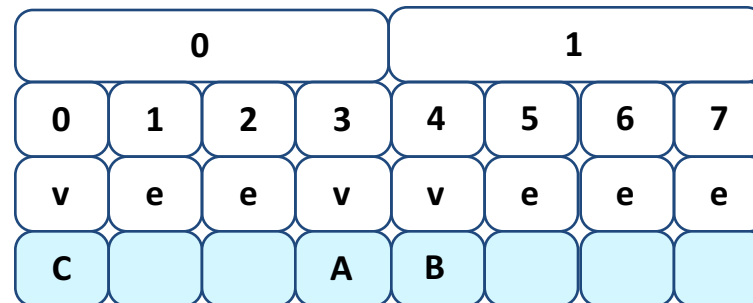




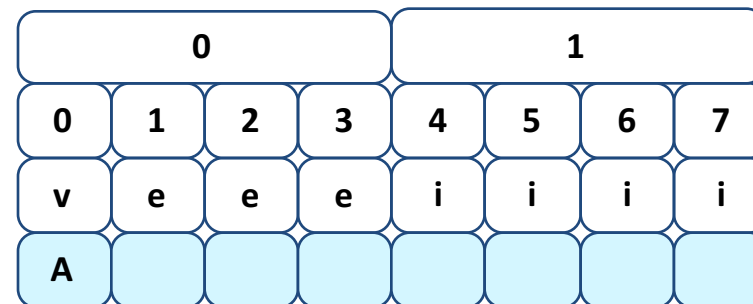
Flash-Based SSDs

- 3, 4, 0 (alles Schreibzugriffe)

- Direct Mapped



- Log Structure



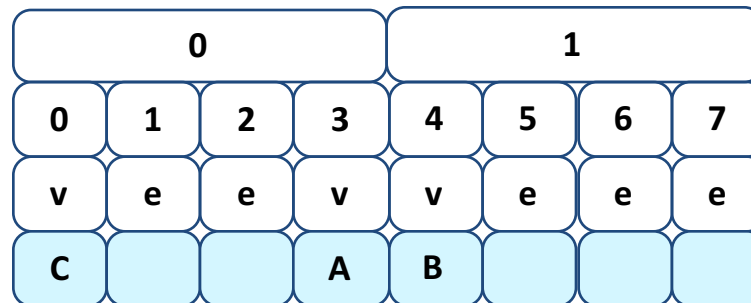
3 → 0



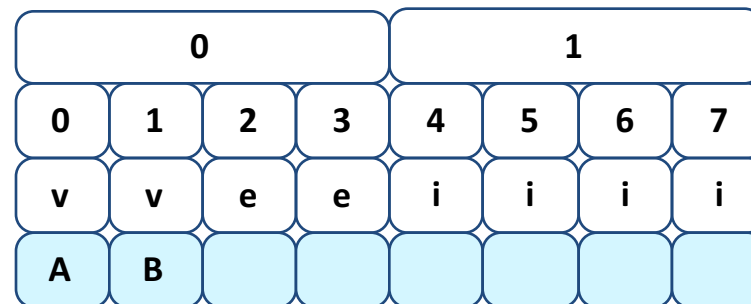
Flash-Based SSDs

- 3, 4, 0 (alles Schreibzugriffe)

- Direct Mapped



- Log Structure



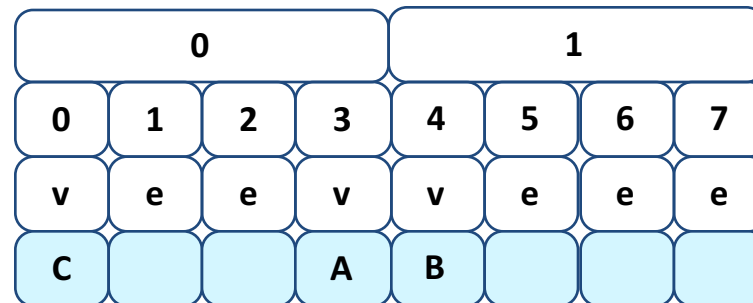
3 → 0
4 → 1



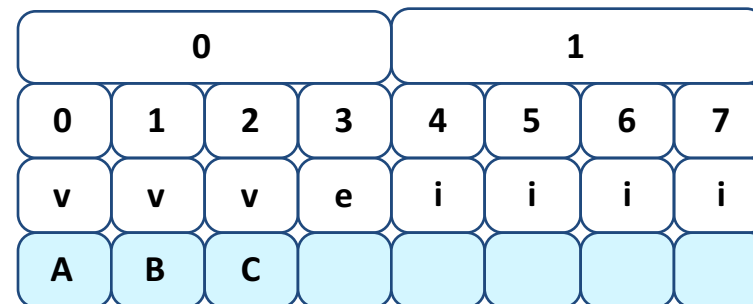
Flash-Based SSDs

- 3, 4, 0 (alles Schreibzugriffe)

- Direct Mapped



- Log Structure



3 → 0
4 → 1
0 → 2



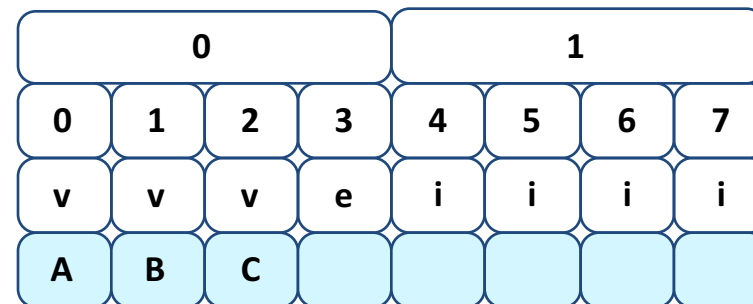
Flash-Based SSDs

➤ 3, 4, 0 (alles Schreibzugriffe)

➤ Direct Mapped



➤ Log Structure



3 → 0
4 → 1
0 → 2



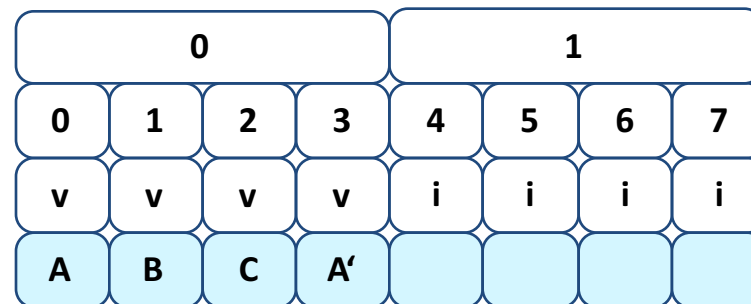
Flash-Based SSDs

- 3, 4, 0 (alles Schreibzugriffe)

- Direct Mapped



- Log Structure



3 → 3
4 → 1
0 → 2



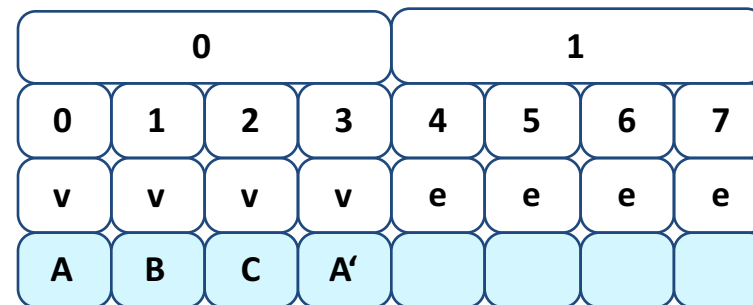
Flash-Based SSDs

➤ 3, 4, 0 (alles Schreibzugriffe)

➤ Direct Mapped



➤ Log Structure



3 → 3
4 → 1
0 → 2



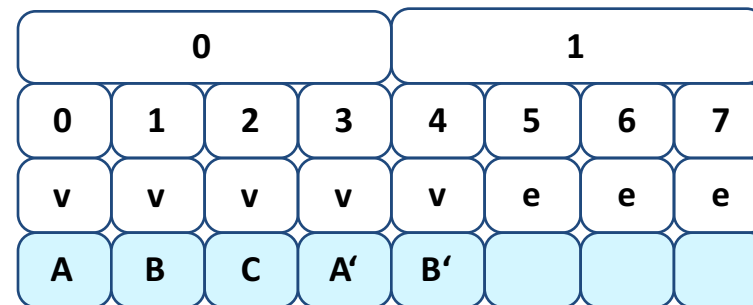
Flash-Based SSDs

➤ 3, 4, 0 (alles Schreibzugriffe)

➤ Direct Mapped



➤ Log Structure



3 → 3
4 → 4
0 → 2

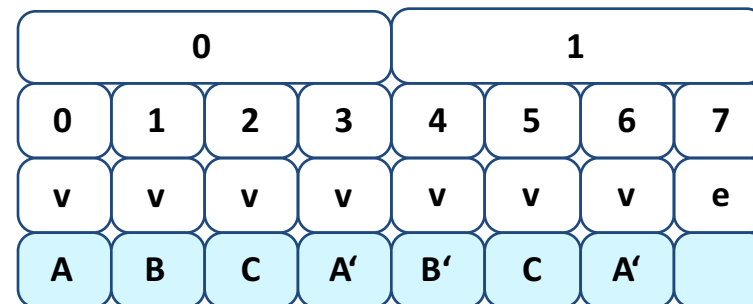


Flash-Based SSDs

- 3, 4, 0 (alles Schreibzugriffe)
- Direct Mapped



- Log Structure



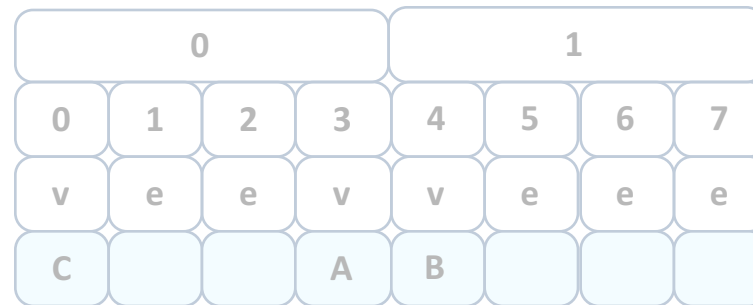
3 → 6
4 → 4
0 → 5



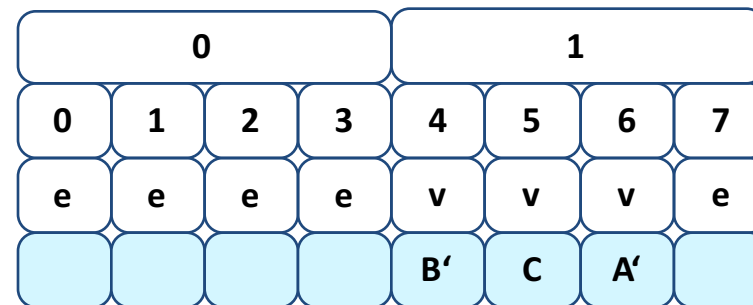
Flash-Based SSDs

➤ 3, 4, 0 (alles Schreibzugriffe)

➤ Direct Mapped



➤ Log Structure

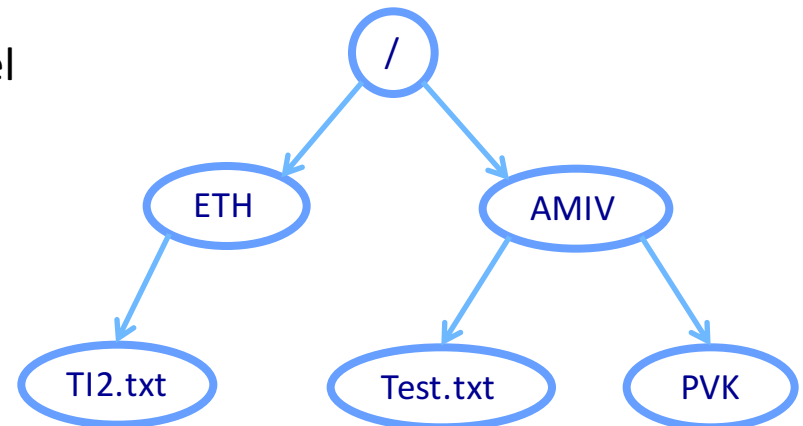


3 → 6
4 → 4
0 → 5



Files und Directories

- Speicher virtualisieren
- **File** → Linearer Array, der beschrieben oder gelesen werden kann
 - Haben einen Low-Level Namen: Meistens seine Inode Nummer
- **Directory** → Liste mit Paaren aus Namen und Inode Nummern
 - Besitzen ebenfalls einen Low-Level Namen
- **Directory Tree** → Root Directory als Wurzel
- **Absolute Pfadnamen** → Von Wurzel aus





Files und Directories

- Datei erstellen → `int fd = open("foo", O_CREAT | O_WRONLY | O_TRUNC);`
 - Open gibt File Descriptor zurück (int-Wert)
 - Platziert Datei in aktuellem Directory
- Datei lesen → `cat foo`
- Datei schreiben → `echo hello > foo`
- Datei umbenennen → `mv foo bar`
- Dateiinformationen → `stat foo`
- Dateien löschen → `rm foo`
- Directory erstellen → `mkdir foo`
- Directory löschen → `rmdir foo`



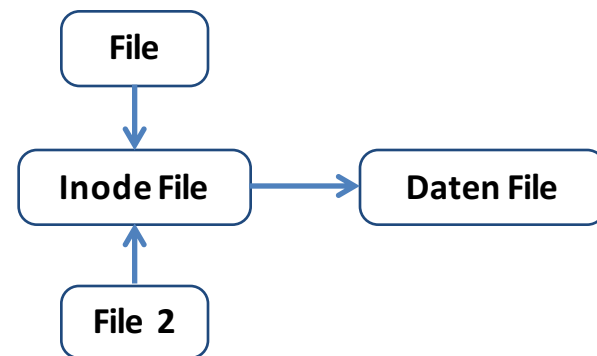
Files und Directories

➤ **Hard Link** → In file file2

- Referenz auf selbe Inode Nummer
- Datei wird nicht kopiert
- Keine Unterscheidung zwischen „Original“ und „Kopie“
- Keine Hard Links auf Directories oder Files anderer Disk Partitions
- Reference / Link Count in Inode

➤ **Symbolic / Soft Link** → In `-s file file2`

- Soft Link ist ein spezielles File
- Daten = Pfadname der anderen Datei
- Dangling References





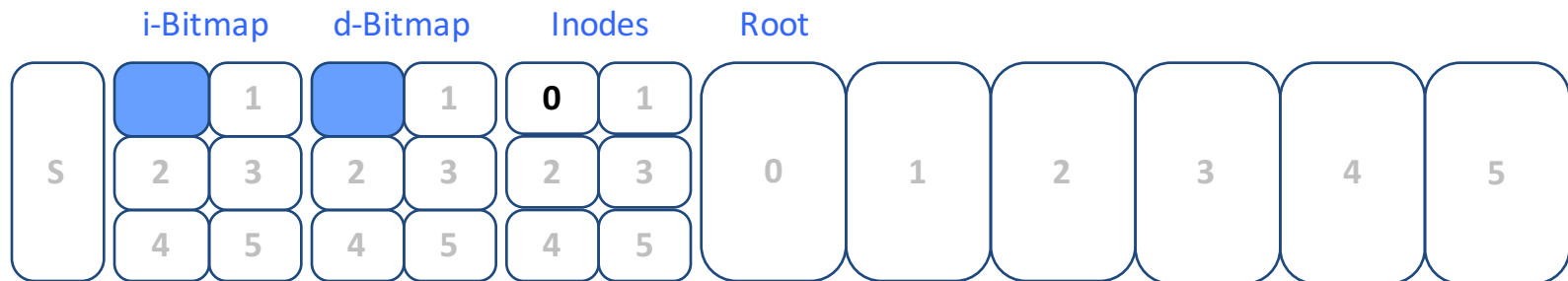
File System Implementation

- **Blöcke** → Grundeinheit des Disks
- Aufteilung des Disks
 - Datenregion
 - Inode Table
 - Data Bitmap
 - Inode Bitmap
 - Superblock
- **Inodes** → Direct / Indirect Pointer (Multi-Level Index)
 - Indirect: Berechne Anzahl möglicher Pointer



File System Implementation

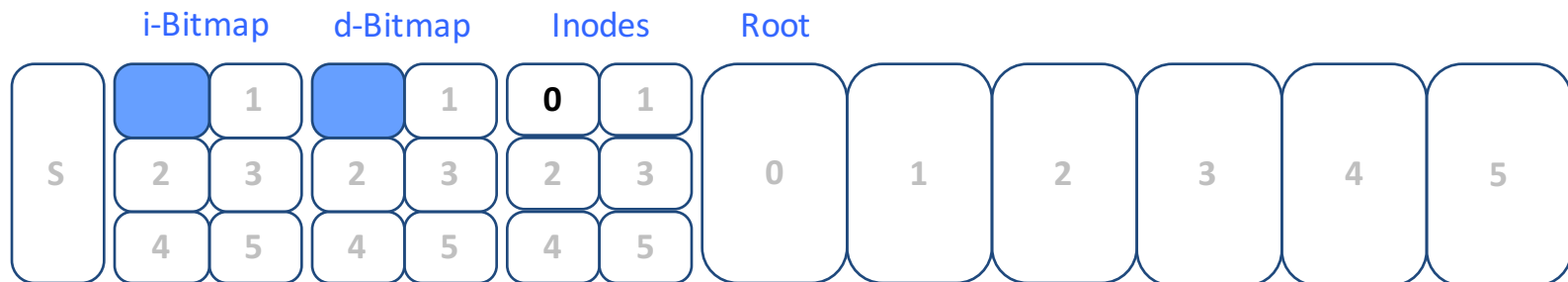
- Only root directory





File System Implementation

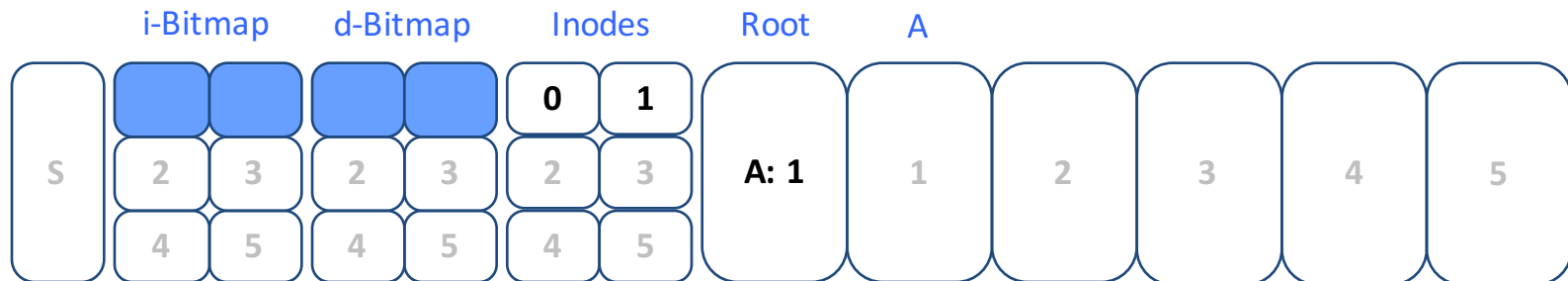
- Only root directory
- Create directory A (1 Block)





File System Implementation

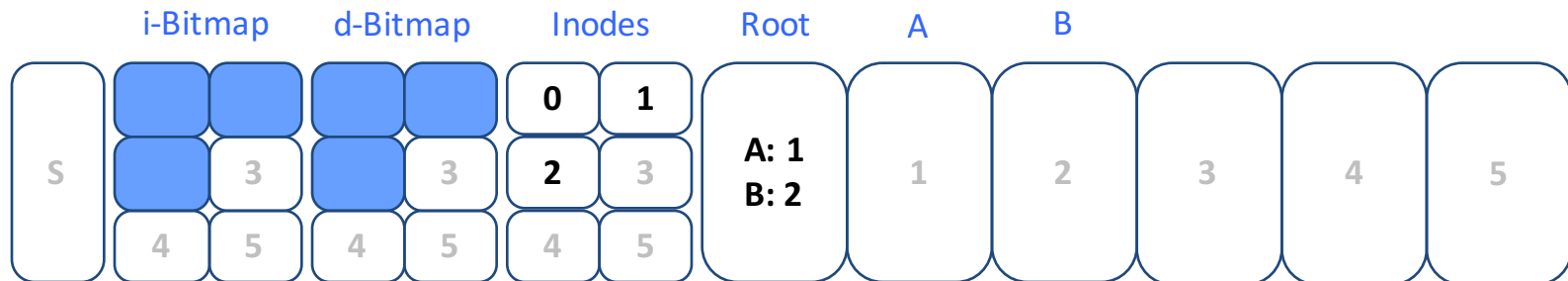
- Only root directory
- Create directory A (1 Block)





File System Implementation

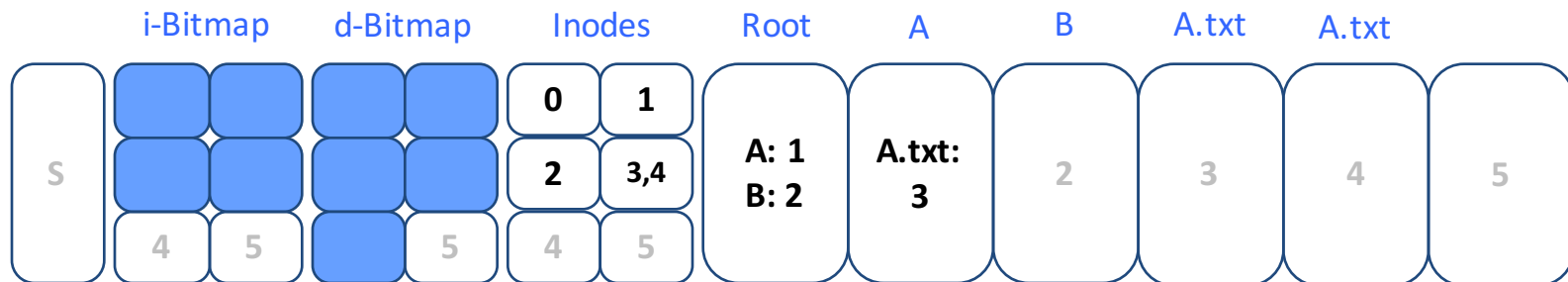
- Only root directory
- Create directory A (1 Block)
- Create directory B (1 Block)





File System Implementation

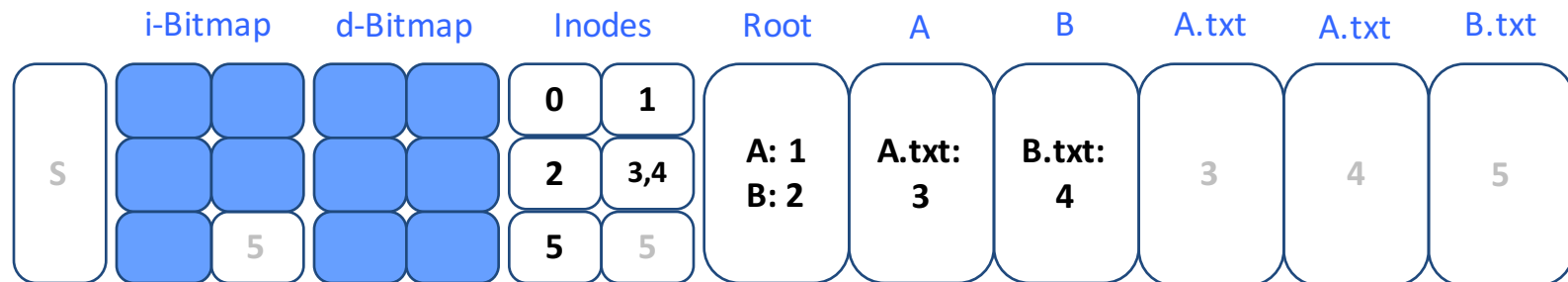
- Only root directory
- Create directory A (1 Block)
- Create directory B (1 Block)
- Create file A.txt (2 Blocks) in A





File System Implementation

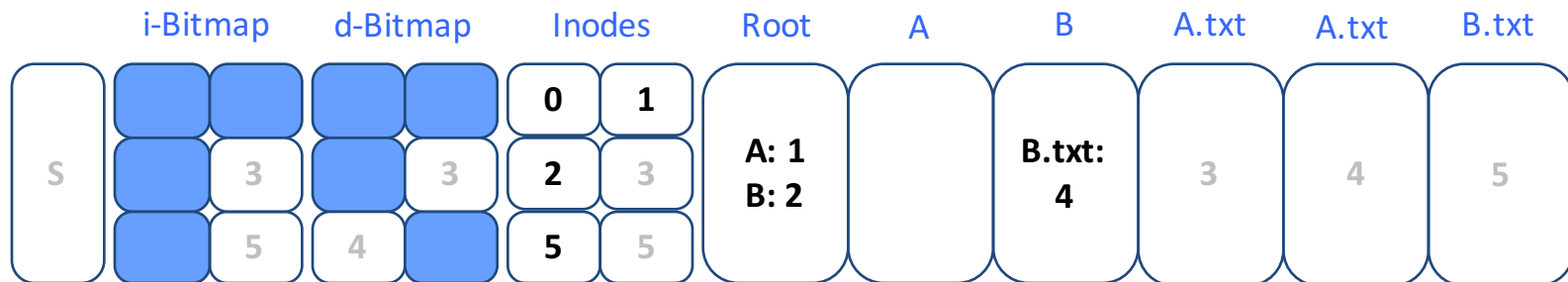
- Only root directory
- Create directory A (1 Block)
- Create directory B (1 Block)
- Create file A.txt (2 Blocks) in A
- Create file B.txt (1 Block) in B





File System Implementation

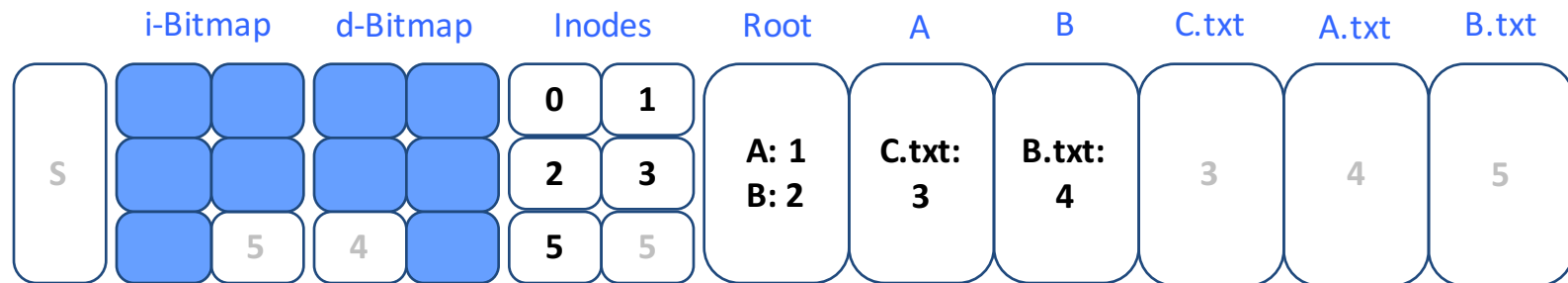
- Delete file A.txt





File System Implementation

- Delete file A.txt
- Create file C.txt (1 Block) in A





Empfohlene Übungen

➤ Serie 9

- Aufgabe 1 (Basic)
- Aufgabe 2 (Basic)
- Aufgabe 3 (Basic)



Dictionaries & Hashing

- Grundlagen & Binary Search Trees
- Hashing
- Collisions
- Worst Case

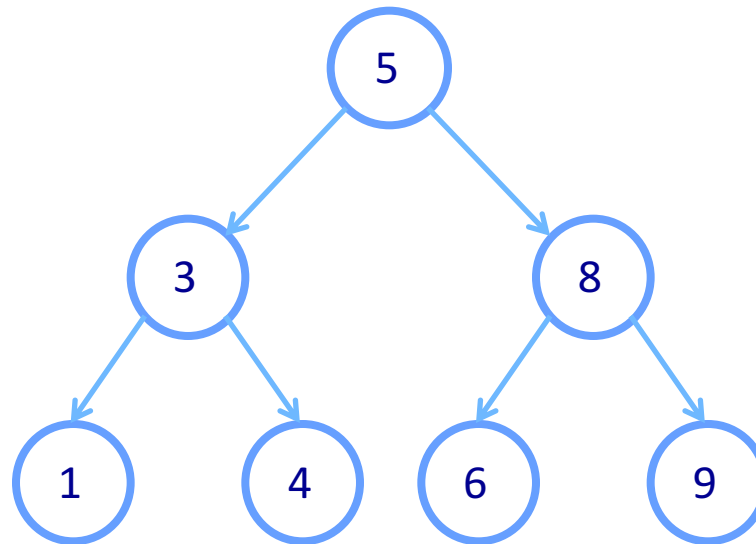


Grundlagen & Binary Search Trees

- Objekte speichern → Jedes Objekt wird durch „Key“ (Nummer) identifiziert
- Wichtigste Operationen → **Search, insert, delete**
 - Static Dictionary → Nur search möglich
 - Dynamic Dictionary → Auch insert und delete möglich
- **Binary Search Tree** → Konzept, um Objekte zu speichern
 - Linkes Kind immer kleinerer Key
 - Rechtes Kind immer grösserer Key
 - Average $O(\log n)$
 - Worst Case $O(n)$



Grundlagen & Binary Search Trees



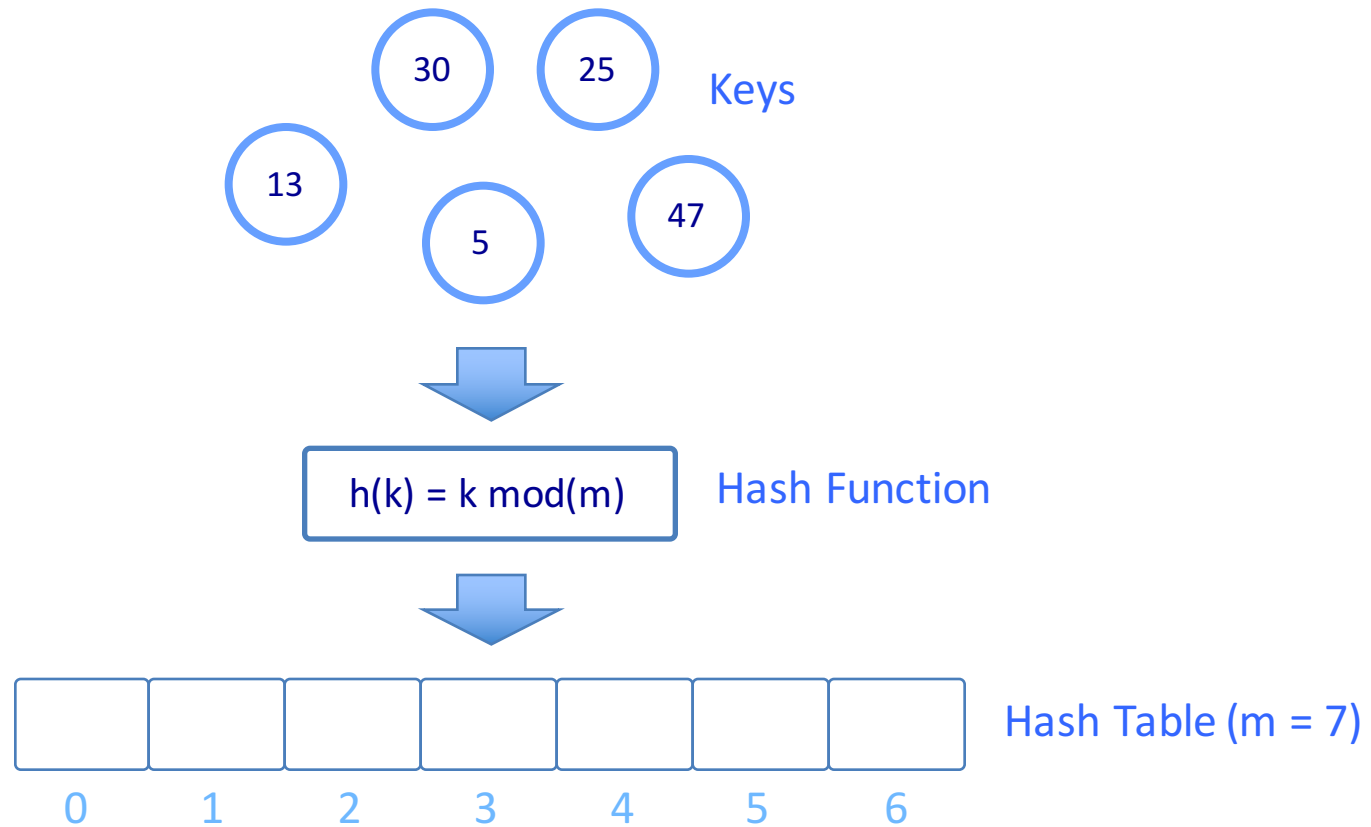


Hashing

- **Keys** → Identifikation für Objekt
- **Universe (U)** → Alle möglichen Keys
- **Key Set (N)** → Für das Problem relevante Keys
 - Subset of U
 - $|N| = n$
- **Hash Table (M)** → Array, wo die Keys gespeichert werden
 - Länge m
- **Bucket $M[i]$** → Platz in Array
- **Hash Function $h(k)$** → Weist jedem Key in U einen Bucket im Array zu
 - Simple Beispiel: $h(k) = k \bmod(m)$

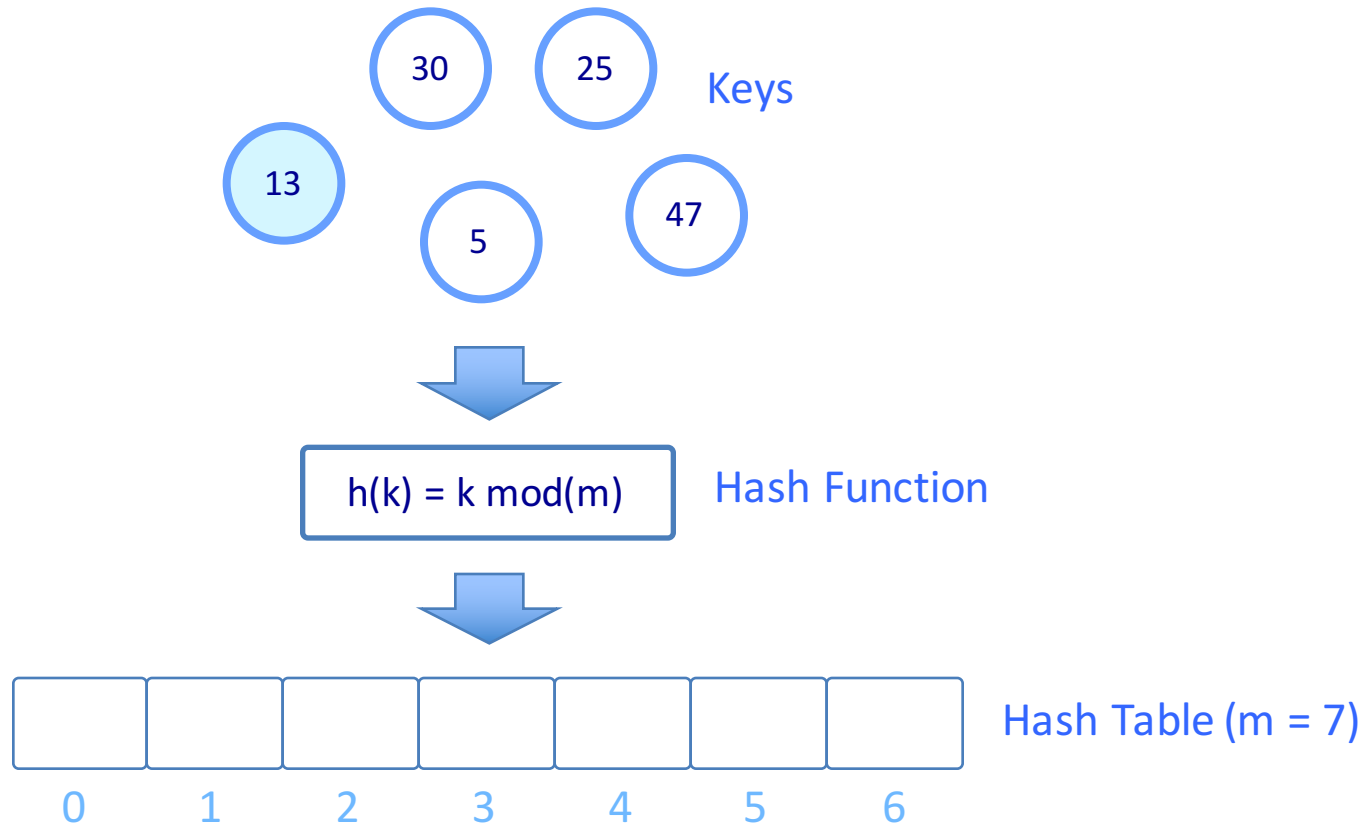


Hashing



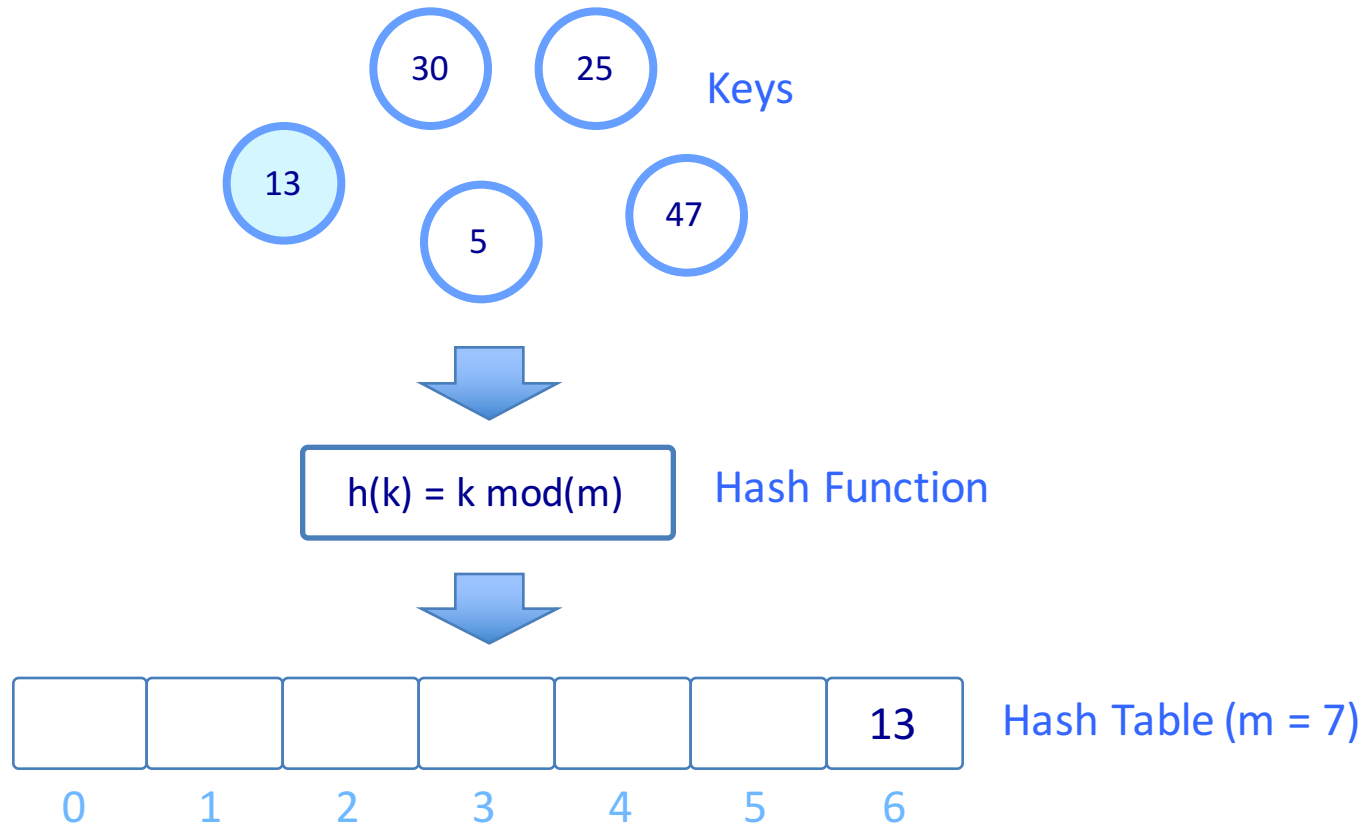


Hashing



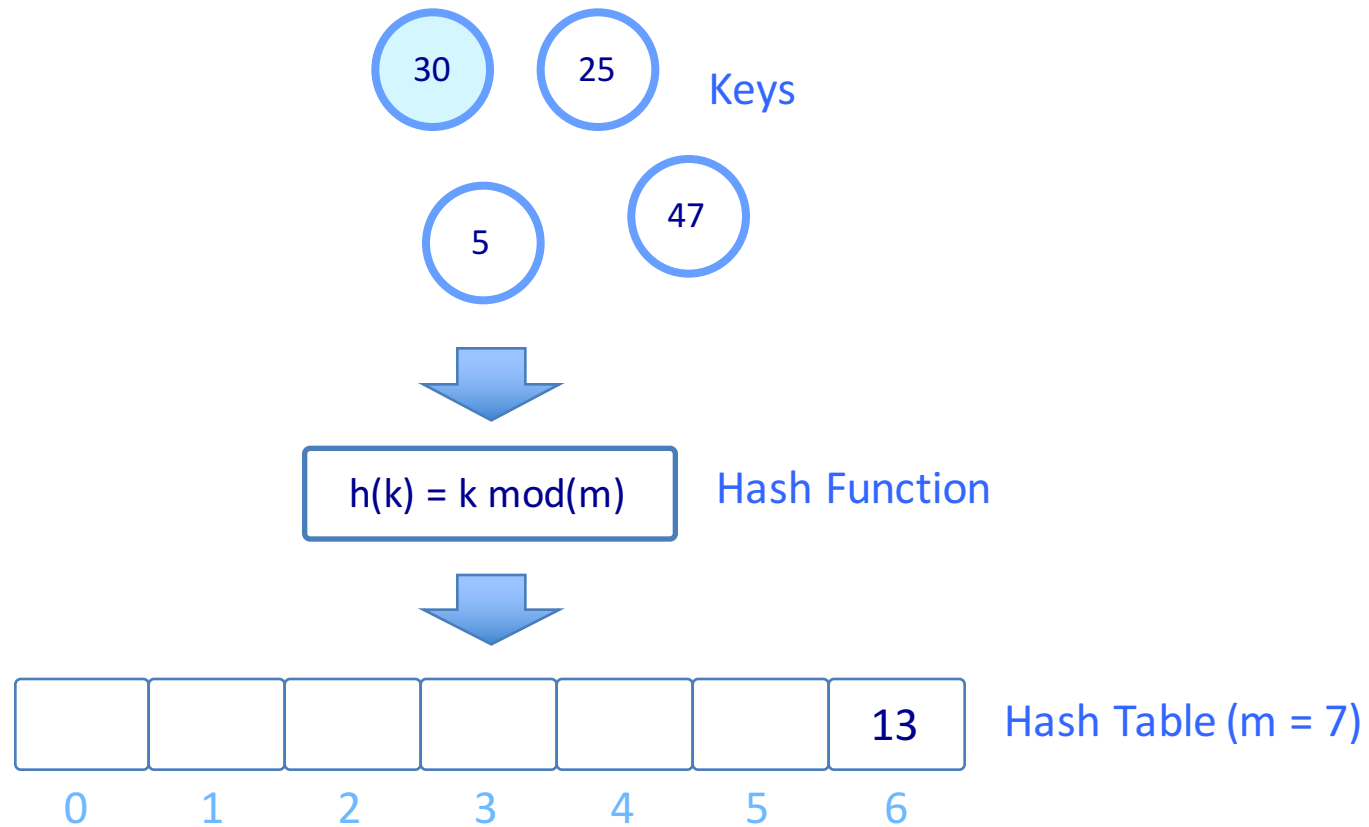


Hashing



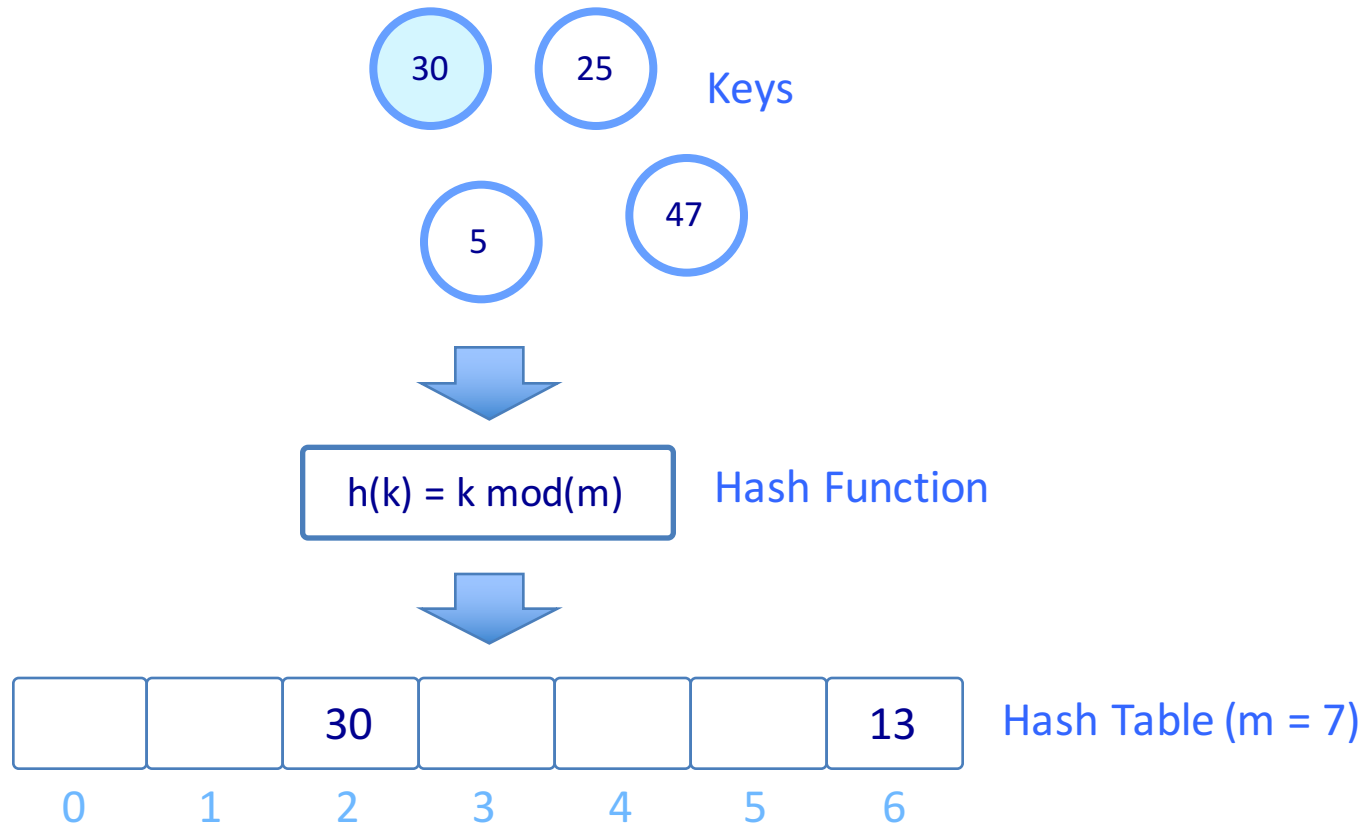


Hashing



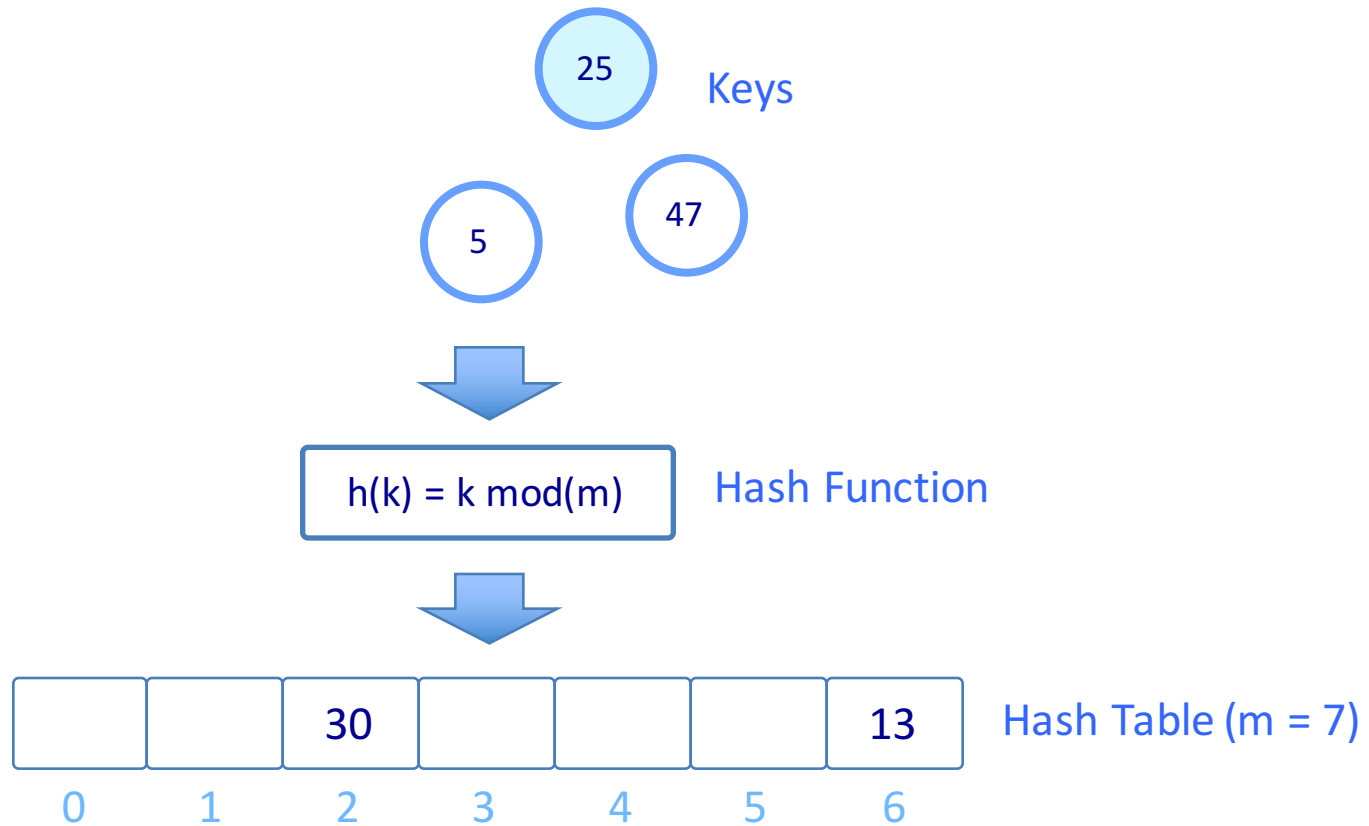


Hashing



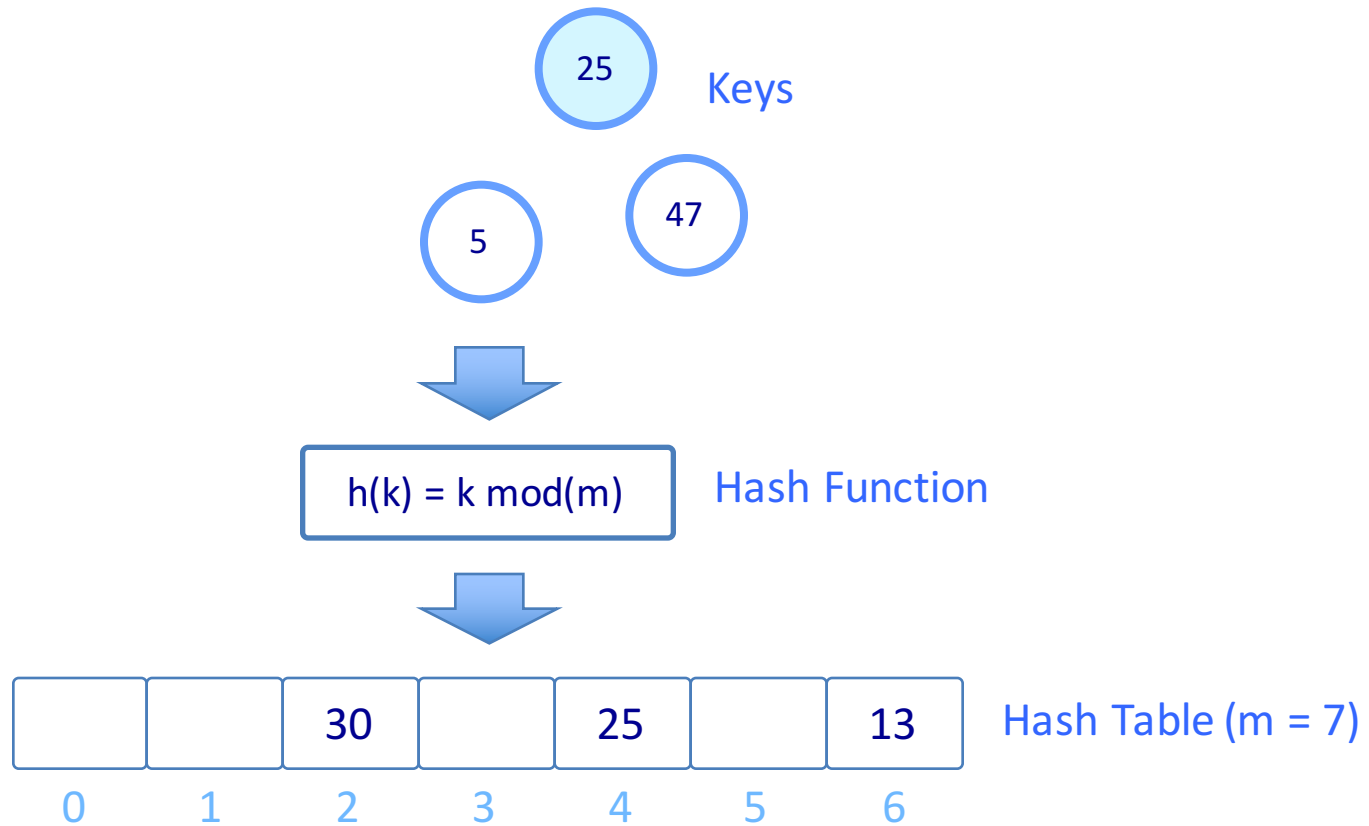


Hashing



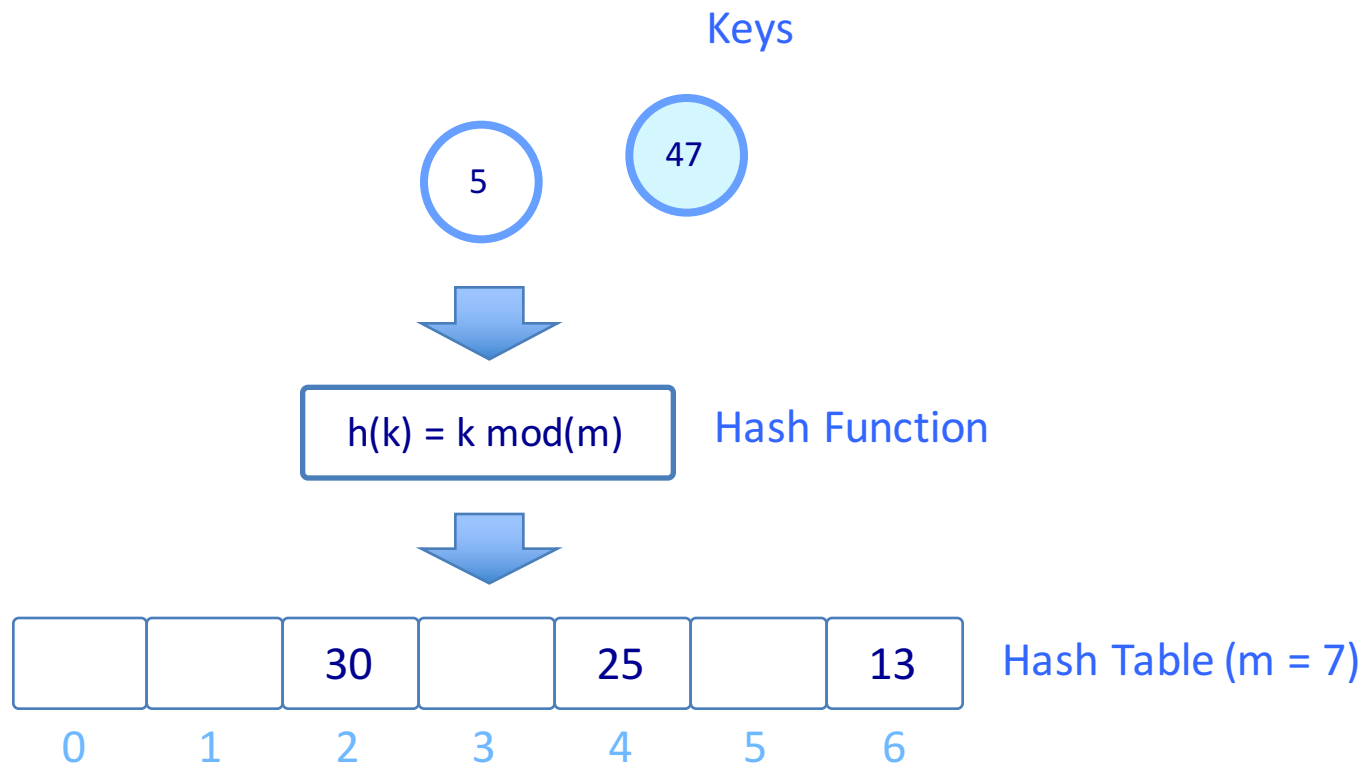


Hashing



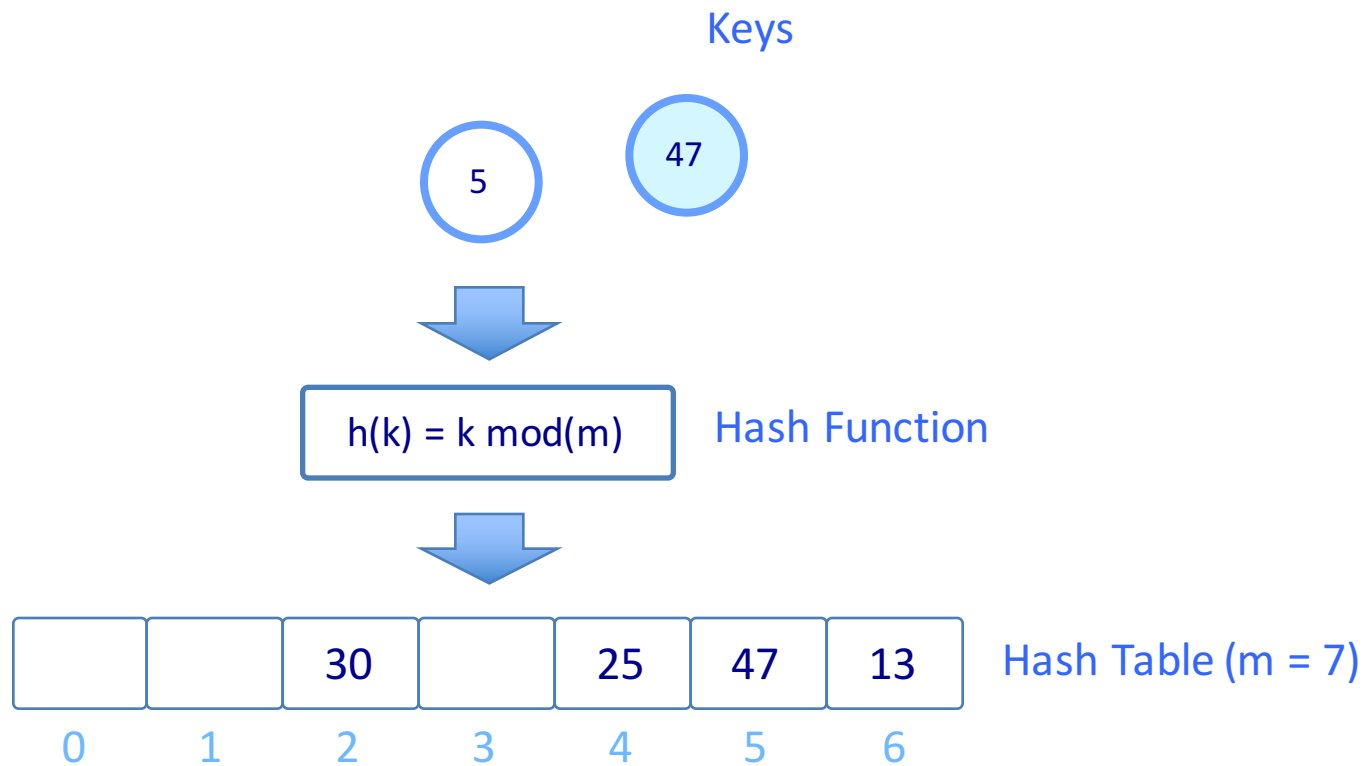


Hashing





Hashing





Hashing

Keys

5



$h(k) = k \bmod(m)$

Hash Function



		30		25	47	13
0	1	2	3	4	5	6

Hash Table ($m = 7$)



Hashing

Keys

5



$$h(k) = k \bmod(m)$$

Hash Function



		30		25	47	13
0	1	2	3	4	5	6

Hash Table (m = 7)





Hashing

- **Collision** → Falls für zwei unterschiedliche Keys k & l gilt: $h(k) = h(l)$
 - **Hashing with Chaining** (Perfect Static Hashing): Pointer z.B. auf Liste
 - **Hashing with Probing**: Alternativen Bucket finden
- **Load factor** $\alpha = n / m$
 - In Java, falls $\alpha > 0.75$ → Grösse von Hash Table wird verdoppelt
- Wie gross soll die Hash Table gewählt werden
 - Nicht zu gross, damit Memory gespart wird
 - Nicht zu klein, um Zahl der Collisions klein zu halten
- **Universal Hashing** → Falls h in universalen Familie von Hash Functions ist, gilt:
 - $\Pr[h(k) = h(l)] = 1/m$ → Die Keys werden gut über die Hash Table verteilt



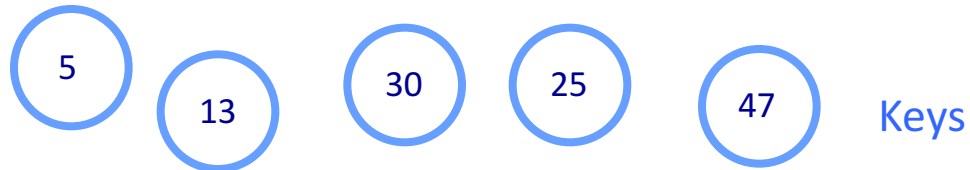
Perfect Static Hashing (Hashing with Chaining)

Algorithm 6.12 Perfect Static Hashing

Input : fixed set of keys N



Perfect Static Hashing (Hashing with Chaining)





Perfect Static Hashing (Hashing with Chaining)

Algorithm 6.12 Perfect Static Hashing

Input : fixed set of keys N

Output : Primary hash table M and secondary hash tables M_i



Perfect Static Hashing (Hashing with Chaining)

Algorithm 6.12 Perfect Static Hashing

Input : fixed set of keys N

Output : Primary hash table M and secondary hash tables M_i

Function: $N_i := \{k \in N : h(k) = i\}$

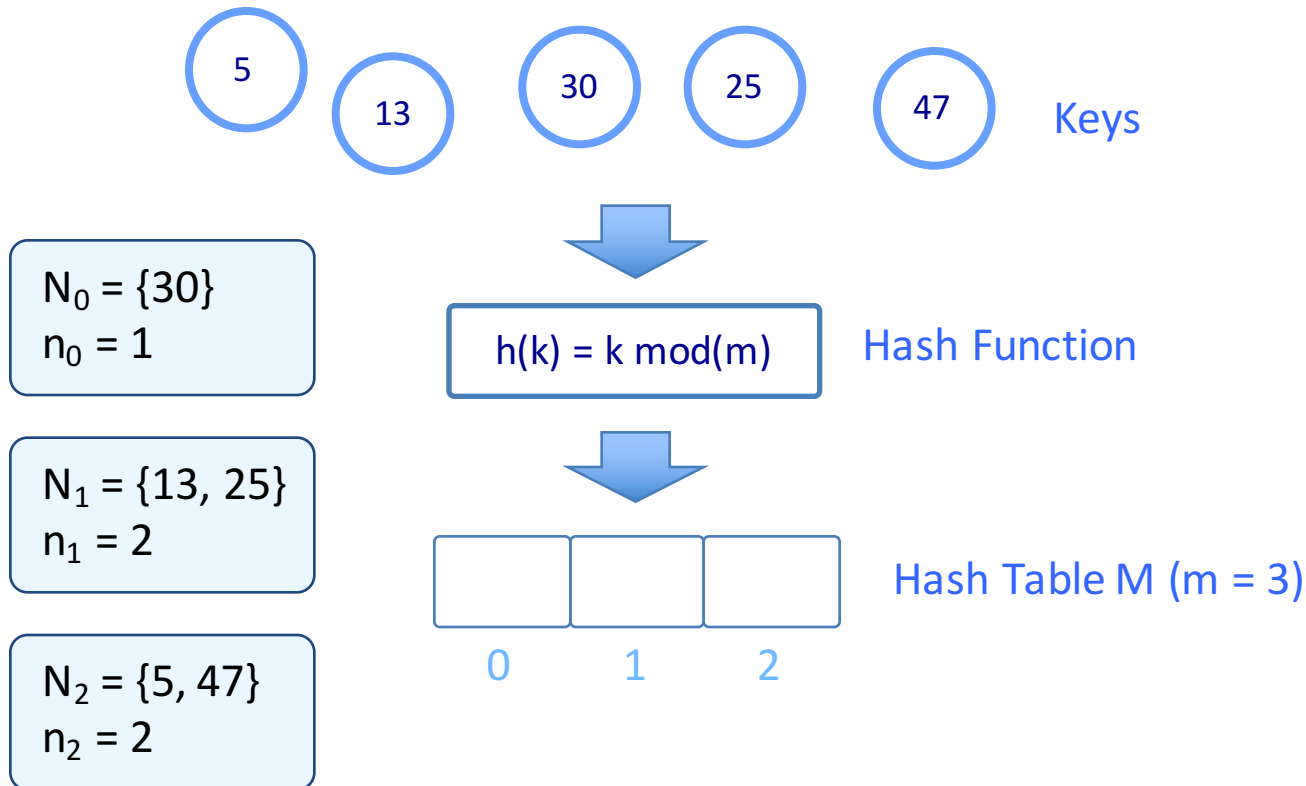
Function: $n_i := |N_i|$

$N_i \rightarrow$ Menge von allen Keys, welche auf Bucket i abgebildet werden

$n_i \rightarrow$ Anzahl Elemente in N_i



Perfect Static Hashing (Hashing with Chaining)





Perfect Static Hashing (Hashing with Chaining)

Algorithm 6.12 Perfect Static Hashing

Input : fixed set of keys N

Output : Primary hash table M and secondary hash tables M_i

Function: $N_i := \{k \in N : h(k) = i\}$

Function: $n_i := |N_i|$

- 1: $M :=$ hash table with n buckets
- 2: **repeat**
- 3: $h :=$ hash function $N \rightarrow M$
- 4: **until** $C(h, N) < n$



Finde Hash Function, welche max. n Collisions generiert



Perfect Static Hashing (Hashing with Chaining)

Algorithm 6.12 Perfect Static Hashing

Input : fixed set of keys N

Output : Primary hash table M and secondary hash tables M_i

Function: $N_i := \{k \in N : h(k) = i\}$

Function: $n_i := |N_i|$

1: $M :=$ hash table with n buckets

2: **repeat**

3: $h :=$ hash function $N \rightarrow M$

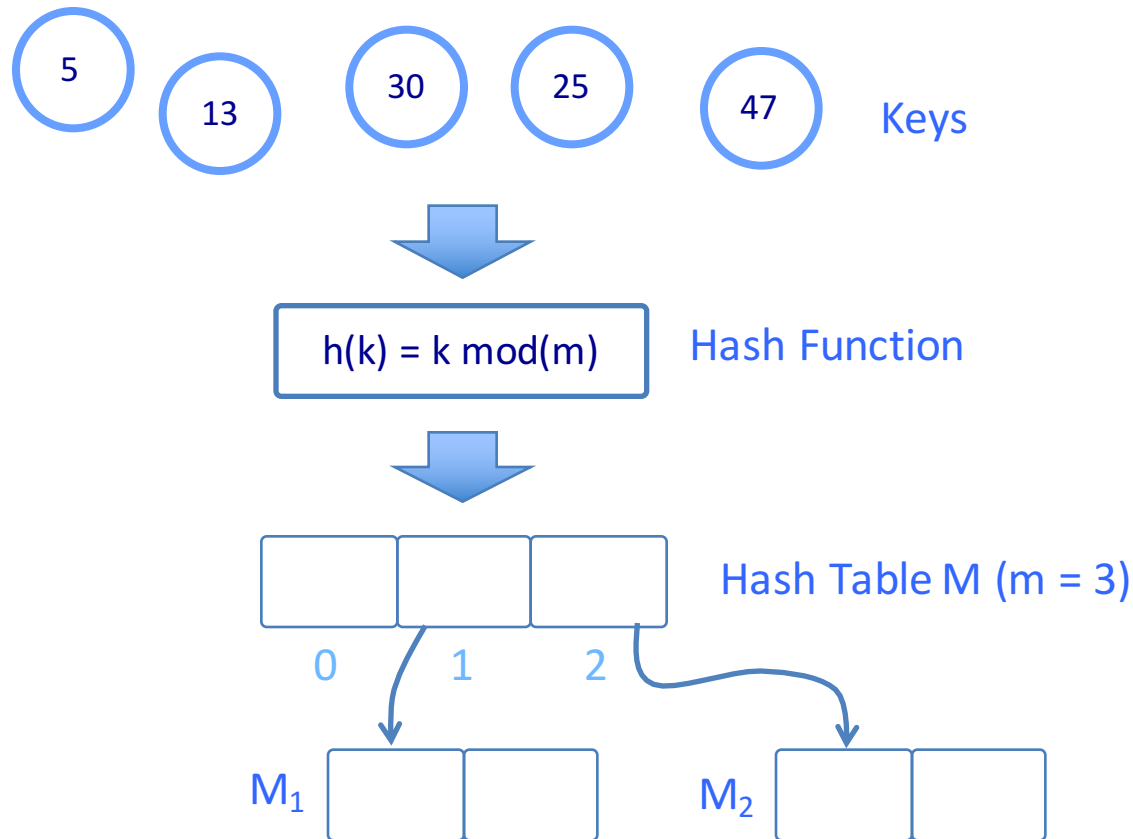
4: **until** $C(h, N) < n$

5: **for** $i \in M$ **do**

6: $M_i :=$ hash table with $2^{\binom{n_i}{2}} = n_i(n_i - 1)$ buckets



Perfect Static Hashing (Hashing with Chaining)





Perfect Static Hashing (Hashing with Chaining)

Algorithm 6.12 Perfect Static Hashing

Input : fixed set of keys N

Output : Primary hash table M and secondary hash tables M_i

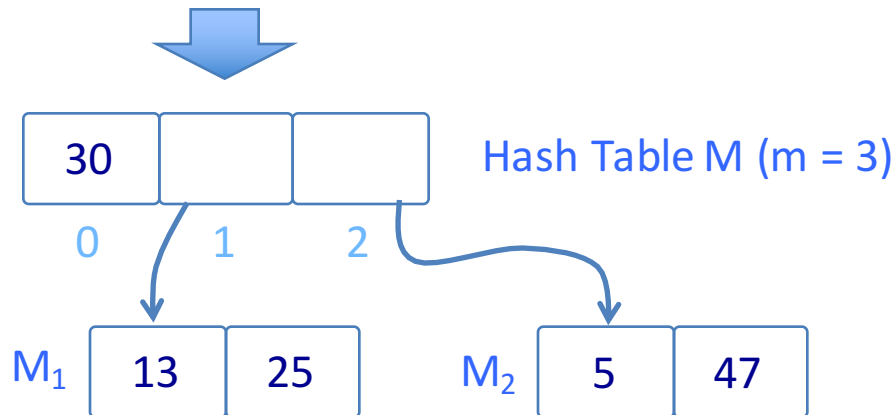
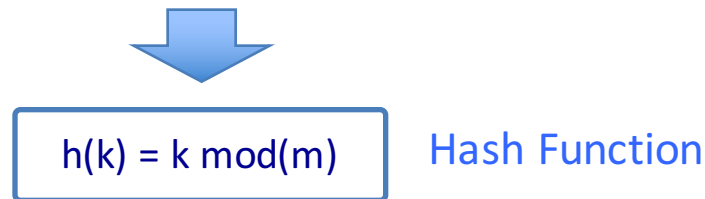
Function: $N_i := \{k \in N : h(k) = i\}$

Function: $n_i := |N_i|$

- 1: $M :=$ hash table with n buckets
 - 2: **repeat**
 - 3: $h :=$ hash function $N \rightarrow M$
 - 4: **until** $C(h, N) < n$
 - 5: **for** $i \in M$ **do**
 - 6: $M_i :=$ hash table with $2^{\binom{n_i}{2}} = n_i(n_i - 1)$ buckets
 - 7: **repeat**
 - 8: $h_i :=$ hash function $N_i \rightarrow M_i$
 - 9: **until** $C(h_i, N_i) < 1$
 - 10: **end for**
 - 11: **return** $(M, h, (M_i)_{i \in [m]}, (h_i)_{i \in [m]})$
-



Perfect Static Hashing (Hashing with Chaining)



Die Grösse von M und allen M_i zusammen ist stets kleiner als $3n$.



Hashing with Probing

- Drei Strategien → Linear Probing, Quadratic Probing, Double Hashing

Type	$h_i(k)$	\approx cost successful	\approx cost unsuccessful
Linear probing	$h(k) + i$	$\frac{1}{2} \left(1 + \frac{1}{(1-\alpha)^2} \right)$	$\frac{1}{2} \left(1 + \frac{1}{1-\alpha} \right)$
Quadratic probing	$h(k) + i^2$	$\frac{1}{1-\alpha} + \ln \frac{1}{1-\alpha} - \alpha$	$1 + \ln \frac{1}{1-\alpha} - \frac{\alpha}{2}$
Double hashing	$h_1(k) + i \cdot h_2(k)$	$\frac{1}{1-\alpha}$	$\frac{1}{\alpha} \ln \left(\frac{1}{1-\alpha} \right)$

→ 7

21		30		11		13
0	1	2	3	4	5	6

$$h(k) = k \bmod(7)$$

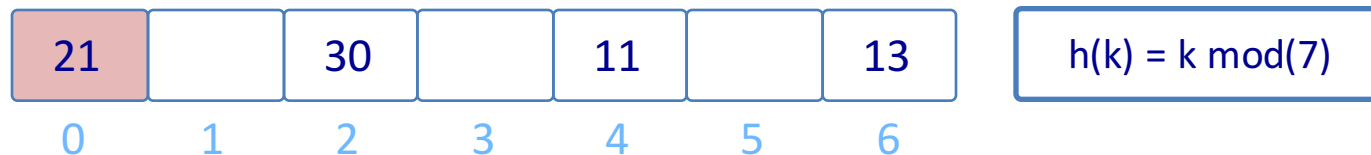


Hashing with Probing

- Drei Strategien → Linear Probing, Quadratic Probing, Double Hashing

Type	$h_i(k)$	\approx cost successful	\approx cost unsuccessful
Linear probing	$h(k) + i$	$\frac{1}{2} \left(1 + \frac{1}{(1-\alpha)^2} \right)$	$\frac{1}{2} \left(1 + \frac{1}{1-\alpha} \right)$
Quadratic probing	$h(k) + i^2$	$\frac{1}{1-\alpha} + \ln \frac{1}{1-\alpha} - \alpha$	$1 + \ln \frac{1}{1-\alpha} - \frac{\alpha}{2}$
Double hashing	$h_1(k) + i \cdot h_2(k)$	$\frac{1}{1-\alpha}$	$\frac{1}{\alpha} \ln \left(\frac{1}{1-\alpha} \right)$

→ 7





Hashing with Probing

- Drei Strategien → Linear Probing, Quadratic Probing, Double Hashing

Type	$h_i(k)$	\approx cost successful	\approx cost unsuccessful
Linear probing	$h(k) + i$	$\frac{1}{2} \left(1 + \frac{1}{(1-\alpha)^2} \right)$	$\frac{1}{2} \left(1 + \frac{1}{1-\alpha} \right)$
Quadratic probing	$h(k) + i^2$	$\frac{1}{1-\alpha} + \ln \frac{1}{1-\alpha} - \alpha$	$1 + \ln \frac{1}{1-\alpha} - \frac{\alpha}{2}$
Double hashing	$h_1(k) + i \cdot h_2(k)$	$\frac{1}{1-\alpha}$	$\frac{1}{\alpha} \ln \left(\frac{1}{1-\alpha} \right)$

→ 7

21	7	30		11		13
0	1	2	3	4	5	6

$$h(k) = k \bmod(7)$$



Hashing with Probing

- Drei Strategien → Linear Probing, Quadratic Probing, Double Hashing

Type	$h_i(k)$	\approx cost successful	\approx cost unsuccessful
Linear probing	$h(k) + i$	$\frac{1}{2} \left(1 + \frac{1}{(1-\alpha)^2} \right)$	$\frac{1}{2} \left(1 + \frac{1}{1-\alpha} \right)$
Quadratic probing	$h(k) + i^2$	$\frac{1}{1-\alpha} + \ln \frac{1}{1-\alpha} - \alpha$	$1 + \ln \frac{1}{1-\alpha} - \frac{\alpha}{2}$
Double hashing	$h_1(k) + i \cdot h_2(k)$	$\frac{1}{1-\alpha}$	$\frac{1}{\alpha} \ln \left(\frac{1}{1-\alpha} \right)$

→ 14

21	7	30		11		13
0	1	2	3	4	5	6

$$h(k) = k \bmod(7)$$



Hashing with Probing

- Drei Strategien → Linear Probing, Quadratic Probing, Double Hashing

Type	$h_i(k)$	\approx cost successful	\approx cost unsuccessful
Linear probing	$h(k) + i$	$\frac{1}{2} \left(1 + \frac{1}{(1-\alpha)^2} \right)$	$\frac{1}{2} \left(1 + \frac{1}{1-\alpha} \right)$
Quadratic probing	$h(k) + i^2$	$\frac{1}{1-\alpha} + \ln \frac{1}{1-\alpha} - \alpha$	$1 + \ln \frac{1}{1-\alpha} - \frac{\alpha}{2}$
Double hashing	$h_1(k) + i \cdot h_2(k)$	$\frac{1}{1-\alpha}$	$\frac{1}{\alpha} \ln \left(\frac{1}{1-\alpha} \right)$

→ 14

21	7	30		11		13
0	1	2	3	4	5	6

$$h(k) = k \bmod(7)$$



Hashing with Probing

- Drei Strategien → Linear Probing, Quadratic Probing, Double Hashing

Type	$h_i(k)$	\approx cost successful	\approx cost unsuccessful
Linear probing	$h(k) + i$	$\frac{1}{2} \left(1 + \frac{1}{(1-\alpha)^2} \right)$	$\frac{1}{2} \left(1 + \frac{1}{1-\alpha} \right)$
Quadratic probing	$h(k) + i^2$	$\frac{1}{1-\alpha} + \ln \frac{1}{1-\alpha} - \alpha$	$1 + \ln \frac{1}{1-\alpha} - \frac{\alpha}{2}$
Double hashing	$h_1(k) + i \cdot h_2(k)$	$\frac{1}{1-\alpha}$	$\frac{1}{\alpha} \ln \left(\frac{1}{1-\alpha} \right)$

→ 14

21	7	30		11		13
0	1	2	3	4	5	6

$$h(k) = k \bmod(7)$$



Hashing with Probing

- Drei Strategien → Linear Probing, Quadratic Probing, Double Hashing

Type	$h_i(k)$	\approx cost successful	\approx cost unsuccessful
Linear probing	$h(k) + i$	$\frac{1}{2} \left(1 + \frac{1}{(1-\alpha)^2} \right)$	$\frac{1}{2} \left(1 + \frac{1}{1-\alpha} \right)$
Quadratic probing	$h(k) + i^2$	$\frac{1}{1-\alpha} + \ln \frac{1}{1-\alpha} - \alpha$	$1 + \ln \frac{1}{1-\alpha} - \frac{\alpha}{2}$
Double hashing	$h_1(k) + i \cdot h_2(k)$	$\frac{1}{1-\alpha}$	$\frac{1}{\alpha} \ln \left(\frac{1}{1-\alpha} \right)$

→ 14

21	7	30		11		13
0	1	2	3	4	5	6

$$h(k) = k \bmod(7)$$



Hashing with Probing

- Drei Strategien → Linear Probing, Quadratic Probing, Double Hashing

Type	$h_i(k)$	\approx cost successful	\approx cost unsuccessful
Linear probing	$h(k) + i$	$\frac{1}{2} \left(1 + \frac{1}{(1-\alpha)^2} \right)$	$\frac{1}{2} \left(1 + \frac{1}{1-\alpha} \right)$
Quadratic probing	$h(k) + i^2$	$\frac{1}{1-\alpha} + \ln \frac{1}{1-\alpha} - \alpha$	$1 + \ln \frac{1}{1-\alpha} - \frac{\alpha}{2}$
Double hashing	$h_1(k) + i \cdot h_2(k)$	$\frac{1}{1-\alpha}$	$\frac{1}{\alpha} \ln \left(\frac{1}{1-\alpha} \right)$

→ 14

21	7	30	14	11		13
0	1	2	3	4	5	6

$$h(k) = k \bmod(7)$$



Hashing with Probing

- Drei Strategien → Linear Probing, Quadratic Probing, Double Hashing

Type	$h_i(k)$	\approx cost successful	\approx cost unsuccessful
Linear probing	$h(k) + i$	$\frac{1}{2} \left(1 + \frac{1}{(1-\alpha)^2} \right)$	$\frac{1}{2} \left(1 + \frac{1}{1-\alpha} \right)$
Quadratic probing	$h(k) + i^2$	$\frac{1}{1-\alpha} + \ln \frac{1}{1-\alpha} - \alpha$	$1 + \ln \frac{1}{1-\alpha} - \frac{\alpha}{2}$
Double hashing	$h_1(k) + i \cdot h_2(k)$	$\frac{1}{1-\alpha}$	$\frac{1}{\alpha} \ln \left(\frac{1}{1-\alpha} \right)$

→ 7

21		30		11		13
0	1	2	3	4	5	6

$$h(k) = k \bmod(7)$$



Hashing with Probing

- Drei Strategien → Linear Probing, Quadratic Probing, Double Hashing

Type	$h_i(k)$	\approx cost successful	\approx cost unsuccessful
Linear probing	$h(k) + i$	$\frac{1}{2} \left(1 + \frac{1}{(1-\alpha)^2} \right)$	$\frac{1}{2} \left(1 + \frac{1}{1-\alpha} \right)$
Quadratic probing	$h(k) + i^2$	$\frac{1}{1-\alpha} + \ln \frac{1}{1-\alpha} - \alpha$	$1 + \ln \frac{1}{1-\alpha} - \frac{\alpha}{2}$
Double hashing	$h_1(k) + i \cdot h_2(k)$	$\frac{1}{1-\alpha}$	$\frac{1}{\alpha} \ln \left(\frac{1}{1-\alpha} \right)$

→ 7

21		30		11		13
0	1	2	3	4	5	6

$$h(k) = k \bmod(7)$$



Hashing with Probing

- Drei Strategien → Linear Probing, Quadratic Probing, Double Hashing

Type	$h_i(k)$	\approx cost successful	\approx cost unsuccessful
Linear probing	$h(k) + i$	$\frac{1}{2} \left(1 + \frac{1}{(1-\alpha)^2} \right)$	$\frac{1}{2} \left(1 + \frac{1}{1-\alpha} \right)$
Quadratic probing	$h(k) + i^2$	$\frac{1}{1-\alpha} + \ln \frac{1}{1-\alpha} - \alpha$	$1 + \ln \frac{1}{1-\alpha} - \frac{\alpha}{2}$
Double hashing	$h_1(k) + i \cdot h_2(k)$	$\frac{1}{1-\alpha}$	$\frac{1}{\alpha} \ln \left(\frac{1}{1-\alpha} \right)$

→ 7

21	7	30		11		13
0	1	2	3	4	5	6

$$h(k) = k \bmod(7)$$



Hashing with Probing

- Drei Strategien → Linear Probing, Quadratic Probing, Double Hashing

Type	$h_i(k)$	\approx cost successful	\approx cost unsuccessful
Linear probing	$h(k) + i$	$\frac{1}{2} \left(1 + \frac{1}{(1-\alpha)^2} \right)$	$\frac{1}{2} \left(1 + \frac{1}{1-\alpha} \right)$
Quadratic probing	$h(k) + i^2$	$\frac{1}{1-\alpha} + \ln \frac{1}{1-\alpha} - \alpha$	$1 + \ln \frac{1}{1-\alpha} - \frac{\alpha}{2}$
Double hashing	$h_1(k) + i \cdot h_2(k)$	$\frac{1}{1-\alpha}$	$\frac{1}{\alpha} \ln \left(\frac{1}{1-\alpha} \right)$

→ 14

21	7	30		11		13
0	1	2	3	4	5	6

$$h(k) = k \bmod(7)$$



Hashing with Probing

- Drei Strategien → Linear Probing, Quadratic Probing, Double Hashing

Type	$h_i(k)$	\approx cost successful	\approx cost unsuccessful
Linear probing	$h(k) + i$	$\frac{1}{2} \left(1 + \frac{1}{(1-\alpha)^2} \right)$	$\frac{1}{2} \left(1 + \frac{1}{1-\alpha} \right)$
Quadratic probing	$h(k) + i^2$	$\frac{1}{1-\alpha} + \ln \frac{1}{1-\alpha} - \alpha$	$1 + \ln \frac{1}{1-\alpha} - \frac{\alpha}{2}$
Double hashing	$h_1(k) + i \cdot h_2(k)$	$\frac{1}{1-\alpha}$	$\frac{1}{\alpha} \ln \left(\frac{1}{1-\alpha} \right)$

→ 14

21	7	30		11		13
0	1	2	3	4	5	6

$$h(k) = k \bmod(7)$$

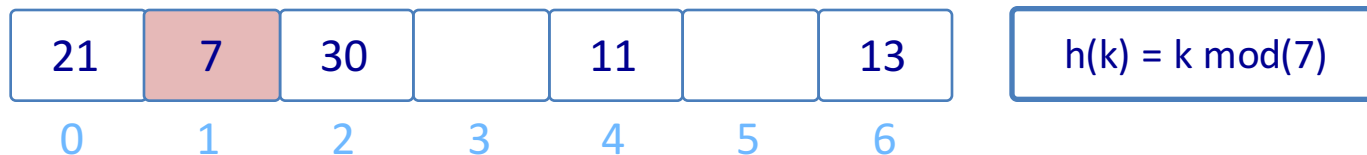


Hashing with Probing

- Drei Strategien → Linear Probing, Quadratic Probing, Double Hashing

Type	$h_i(k)$	\approx cost successful	\approx cost unsuccessful
Linear probing	$h(k) + i$	$\frac{1}{2} \left(1 + \frac{1}{(1-\alpha)^2} \right)$	$\frac{1}{2} \left(1 + \frac{1}{1-\alpha} \right)$
Quadratic probing	$h(k) + i^2$	$\frac{1}{1-\alpha} + \ln \frac{1}{1-\alpha} - \alpha$	$1 + \ln \frac{1}{1-\alpha} - \frac{\alpha}{2}$
Double hashing	$h_1(k) + i \cdot h_2(k)$	$\frac{1}{1-\alpha}$	$\frac{1}{\alpha} \ln \left(\frac{1}{1-\alpha} \right)$

→ 14



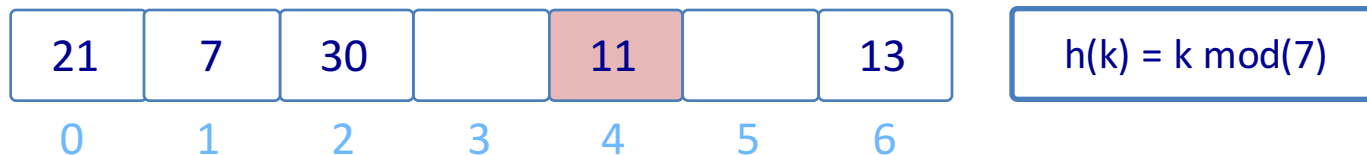


Hashing with Probing

- Drei Strategien → Linear Probing, Quadratic Probing, Double Hashing

Type	$h_i(k)$	\approx cost successful	\approx cost unsuccessful
Linear probing	$h(k) + i$	$\frac{1}{2} \left(1 + \frac{1}{(1-\alpha)^2} \right)$	$\frac{1}{2} \left(1 + \frac{1}{1-\alpha} \right)$
Quadratic probing	$h(k) + i^2$	$\frac{1}{1-\alpha} + \ln \frac{1}{1-\alpha} - \alpha$	$1 + \ln \frac{1}{1-\alpha} - \frac{\alpha}{2}$
Double hashing	$h_1(k) + i \cdot h_2(k)$	$\frac{1}{1-\alpha}$	$\frac{1}{\alpha} \ln \left(\frac{1}{1-\alpha} \right)$

→ 14





Hashing with Probing

- Drei Strategien → Linear Probing, Quadratic Probing, Double Hashing

Type	$h_i(k)$	\approx cost successful	\approx cost unsuccessful
Linear probing	$h(k) + i$	$\frac{1}{2} \left(1 + \frac{1}{(1-\alpha)^2} \right)$	$\frac{1}{2} \left(1 + \frac{1}{1-\alpha} \right)$
Quadratic probing	$h(k) + i^2$	$\frac{1}{1-\alpha} + \ln \frac{1}{1-\alpha} - \alpha$	$1 + \ln \frac{1}{1-\alpha} - \frac{\alpha}{2}$
Double hashing	$h_1(k) + i \cdot h_2(k)$	$\frac{1}{1-\alpha}$	$\frac{1}{\alpha} \ln \left(\frac{1}{1-\alpha} \right)$

→ 14

21	7	30		11		13
0	1	2	3	4	5	6

$$h(k) = k \bmod(7)$$

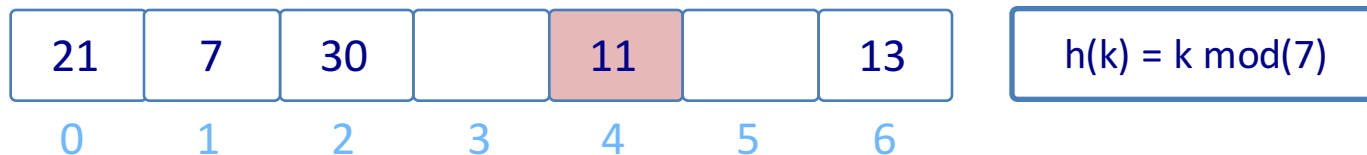


Hashing with Probing

- Drei Strategien → Linear Probing, Quadratic Probing, Double Hashing

Type	$h_i(k)$	\approx cost successful	\approx cost unsuccessful
Linear probing	$h(k) + i$	$\frac{1}{2} \left(1 + \frac{1}{(1-\alpha)^2} \right)$	$\frac{1}{2} \left(1 + \frac{1}{1-\alpha} \right)$
Quadratic probing	$h(k) + i^2$	$\frac{1}{1-\alpha} + \ln \frac{1}{1-\alpha} - \alpha$	$1 + \ln \frac{1}{1-\alpha} - \frac{\alpha}{2}$
Double hashing	$h_1(k) + i \cdot h_2(k)$	$\frac{1}{1-\alpha}$	$\frac{1}{\alpha} \ln \left(\frac{1}{1-\alpha} \right)$

→ 14



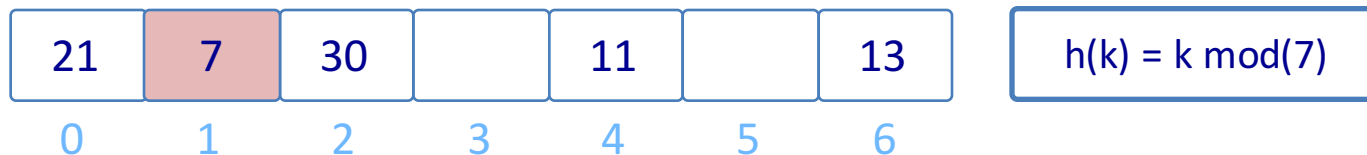


Hashing with Probing

- Drei Strategien → Linear Probing, Quadratic Probing, Double Hashing

Type	$h_i(k)$	\approx cost successful	\approx cost unsuccessful
Linear probing	$h(k) + i$	$\frac{1}{2} \left(1 + \frac{1}{(1-\alpha)^2} \right)$	$\frac{1}{2} \left(1 + \frac{1}{1-\alpha} \right)$
Quadratic probing	$h(k) + i^2$	$\frac{1}{1-\alpha} + \ln \frac{1}{1-\alpha} - \alpha$	$1 + \ln \frac{1}{1-\alpha} - \frac{\alpha}{2}$
Double hashing	$h_1(k) + i \cdot h_2(k)$	$\frac{1}{1-\alpha}$	$\frac{1}{\alpha} \ln \left(\frac{1}{1-\alpha} \right)$

→ 14





Hashing with Probing

- Drei Strategien → Linear Probing, Quadratic Probing, Double Hashing

Type	$h_i(k)$	\approx cost successful	\approx cost unsuccessful
Linear probing	$h(k) + i$	$\frac{1}{2} \left(1 + \frac{1}{(1-\alpha)^2} \right)$	$\frac{1}{2} \left(1 + \frac{1}{1-\alpha} \right)$
Quadratic probing	$h(k) + i^2$	$\frac{1}{1-\alpha} + \ln \frac{1}{1-\alpha} - \alpha$	$1 + \ln \frac{1}{1-\alpha} - \frac{\alpha}{2}$
Double hashing	$h_1(k) + i \cdot h_2(k)$	$\frac{1}{1-\alpha}$	$\frac{1}{\alpha} \ln \left(\frac{1}{1-\alpha} \right)$

→ 14

21	7	30		11		13
0	1	2	3	4	5	6

$$h(k) = k \bmod(7)$$



Cuckoo Hashing (Alternativer Ansatz)

Algorithm 6.19 Cuckoo Hashing Insert

Input : key $k \in U$ we want to insert; counter **limit** specifying the maximum number of tries

Data Structures: arrays M_1, M_2 of equal size

Functions : hash functions $h_1 : U \rightarrow M_1, h_2 : U \rightarrow M_2$; chosen independently and uniformly at random from universal families

$k = 14, \text{limit} = 7$

16		10		4		22
0	1	2	3	4	5	6

$$h_1(k) = k - 1 \bmod(7)$$

7		23		11		27
0	1	2	3	4	5	6

$$h_2(k) = k \bmod(7)$$



Cuckoo Hashing (Alternativer Ansatz)

```
1: if  $M_1[h_1(k)] = k$  or  $M_2[h_2(k)] = k$  then
2:   return
3: end if
4:  $t := 1$ 
5: while  $t \leq \text{limit}$  do
6:   swap  $k$  with  $M_1[h_1(k)]$ 
7:   if  $k = \perp$  then
8:     return
9:   end if
10:  swap  $k$  with  $M_2[h_2(k)]$ 
11:  if  $k = \perp$  then
12:    return
13:  end if
14:   $t := t + 1$ 
15: end while
16: rehash()
17: CuckooHashingInsert( $k, \text{limit}$ )
```

Element schon vorhanden

Tausche k mit $M_1[h_1(k)]$

Tausche k mit $M_2[h_2(k)]$



Cuckoo Hashing (Alternativer Ansatz)

Algorithm 6.19 Cuckoo Hashing Insert

Input : key $k \in U$ we want to insert; counter **limit** specifying the maximum number of tries

Data Structures: arrays M_1, M_2 of equal size

Functions : hash functions $h_1 : U \rightarrow M_1, h_2 : U \rightarrow M_2$; chosen independently and uniformly at random from universal families

$k = 14, \text{limit} = 7$

16		10		4		22
0	1	2	3	4	5	6

$$h_1(k) = k - 1 \bmod(7)$$

7		23		11		27
0	1	2	3	4	5	6

$$h_2(k) = k \bmod(7)$$



Cuckoo Hashing (Alternativer Ansatz)

Algorithm 6.19 Cuckoo Hashing Insert

Input : key $k \in U$ we want to insert; counter **limit** specifying the maximum number of tries

Data Structures: arrays M_1, M_2 of equal size

Functions : hash functions $h_1 : U \rightarrow M_1, h_2 : U \rightarrow M_2$; chosen independently and uniformly at random from universal families

$k = 14, \text{limit} = 7$

16		10		4		22
0	1	2	3	4	5	6

$$h_1(k) = k - 1 \bmod(7)$$

7		23		11		27
0	1	2	3	4	5	6

$$h_2(k) = k \bmod(7)$$



Cuckoo Hashing (Alternativer Ansatz)

Algorithm 6.19 Cuckoo Hashing Insert

Input : key $k \in U$ we want to insert; counter **limit** specifying the maximum number of tries

Data Structures: arrays M_1, M_2 of equal size

Functions : hash functions $h_1 : U \rightarrow M_1, h_2 : U \rightarrow M_2$; chosen independently and uniformly at random from universal families

$k = 22, \text{limit} = 7$

16		10		4		14
0	1	2	3	4	5	6

$$h_1(k) = k - 1 \bmod(7)$$

7		23		11		27
0	1	2	3	4	5	6

$$h_2(k) = k \bmod(7)$$



Cuckoo Hashing (Alternativer Ansatz)

Algorithm 6.19 Cuckoo Hashing Insert

Input : key $k \in U$ we want to insert; counter **limit** specifying the maximum number of tries

Data Structures: arrays M_1, M_2 of equal size

Functions : hash functions $h_1 : U \rightarrow M_1, h_2 : U \rightarrow M_2$; chosen independently and uniformly at random from universal families

$k = 22, \text{limit} = 7$

16		10		4		14
0	1	2	3	4	5	6

$$h_1(k) = k - 1 \bmod(7)$$

7		23		11		27
0	1	2	3	4	5	6

$$h_2(k) = k \bmod(7)$$



Cuckoo Hashing (Alternativer Ansatz)

Algorithm 6.19 Cuckoo Hashing Insert

Input : key $k \in U$ we want to insert; counter **limit** specifying the maximum number of tries

Data Structures: arrays M_1, M_2 of equal size

Functions : hash functions $h_1 : U \rightarrow M_1, h_2 : U \rightarrow M_2$; chosen independently and uniformly at random from universal families

$k = \text{NULL}, \text{limit} = 7$

16		10		4		14
0	1	2	3	4	5	6

$$h_1(k) = k - 1 \pmod{7}$$

7	22	23		11		27
0	1	2	3	4	5	6

$$h_2(k) = k \pmod{7}$$



Empfohlene Übungen

➤ Serie 6

- Aufgabe 2 (Basic)
- Aufgabe 5 (Advanced)



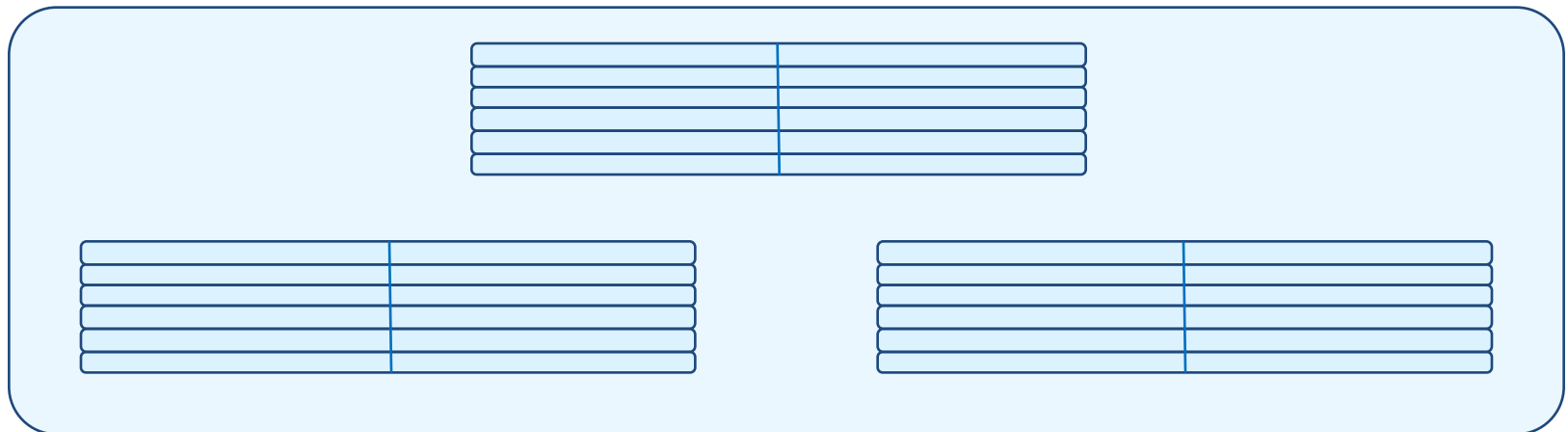
Inhalt

- Grundlagen Datenbanken
- SQL Basics
- Modeling
- Joins
- Keys & Constraints
- Indexing



Grundlagen Datenbanken

- **Tabelle** mit Rows und Columns
- **Row / Zeile** → Enthalten alle die gleichen Felder
- **Column** → Jedes Feld hat eine Spalte
- **Datenbank** → Ansammlung von Tabellen





SQL Basics

➤ Data (Column) Types

- CHARACTER(m) → Fixe Länge, max. m
- CHARACTER VARYING(m) → Variable Länge, max. m
- BIT(m) & BIT VARYING(m)
- NUMERIC, DECIMAL, INTEGER, SMALLINT
- FLOAT, REAL, DOUBLE PRECISION
- DATE, TIME
- INTERVAL → Zeitintervall



SQL Basics

➤ Datenbank initialisieren

```
CREATE DATABASE moviedb; → Erstellen einer neuen Datenbank
USE moviedb;
CREATE TABLE movies (
    title CHARACTER VARYING(200) NOT NULL,
    director CHARACTER VARYING(200) DEFAULT 'Steven Spielberg',
    year INTEGER
);
```



SQL Basics

➤ Datenbank initialisieren

```
CREATE DATABASE moviedb;  
USE moviedb; → Benütze diese Datenbank für nachfolgende Befehle  
CREATE TABLE movies (  
    title CHARACTER VARYING(200) NOT NULL,  
    director CHARACTER VARYING(200) DEFAULT 'Steven Spielberg',  
    year INTEGER  
);
```




SQL Basics

➤ Datenbank initialisieren

```
CREATE DATABASE moviedb;  
USE moviedb;  
CREATE TABLE movies ( → Erstelle Tabelle mit versch. Feldern  
    title CHARACTER VARYING(200) NOT NULL,  
    director CHARACTER VARYING(200) DEFAULT 'Steven Spielberg',  
    year INTEGER  
);
```



SQL Basics

➤ Tabelle mit Einträgen füllen

```
INSERT INTO movies → Welche Tabelle wird verwendet  
  (title, director, year) VALUES  
  (`12 Angry Men`, `Sidney Lumet`, 1957),  
  (`Raiders of the Lost Ark`, `DEFAULT`, 1981),  
  (`Wars of the Worlds`, `DEFAULT`, 2005),  
;
```



SQL Basics

➤ Tabelle mit Einträgen füllen

```
INSERT INTO movies
  (title, director, year) VALUES → Felder auswählen
  (`12 Angry Men`, `Sidney Lumet`, 1957),
  (`Raiders of the Lost Ark`, `DEFAULT`, 1981),
  (`Wars of the Worlds`, `DEFAULT`, 2005),
;
```



SQL Basics

➤ Tabelle mit Einträgen füllen

```
INSERT INTO movies
  (title, director, year) VALUES
  (`12 Angry Men`, `Sidney Lumet`, 1957), → Werte übergeben
  (`Raiders of the Lost Ark`, `DEFAULT`, 1981),
  (`Wars of the Worlds`, `DEFAULT`, 2005),
;
```

➤ Falls Feld nicht gewählt wird → Default-Wert wird gesetzt



SQL Basics

➤ Daten auslesen

```
SELECT * FROM movies;  
SELECT * FROM movies WHERE director = 'Steven Spielberg';  
SELECT title FROM movies WHERE year BETWEEN 1990 AND 1999;  
SELECT * FROM movies WHERE title IS NULL OR director IS NULL;  
SELECT title, director FROM movies WHERE title LIKE '%the%';
```



SQL Basics

➤ Daten auslesen

```
SELECT * FROM movies; → Wähle alle Felder von Tabelle `movies`  
SELECT * FROM movies WHERE director = 'Steven Spielberg';  
SELECT title FROM movies WHERE year BETWEEN 1990 AND 1999;  
SELECT * FROM movies WHERE title IS NULL OR director IS NULL;  
SELECT title, director FROM movies WHERE title LIKE '%the%';
```



SQL Basics

➤ Daten auslesen

```
SELECT * FROM movies;  
SELECT * FROM movies WHERE director = 'Steven Spielberg';  
SELECT title FROM movies WHERE year BETWEEN 1990 AND 1999;  
SELECT * FROM movies WHERE title IS NULL OR director IS NULL;  
SELECT title, director FROM movies WHERE title LIKE '%the%';
```

➤ % → Ersetzt beliebig viele beliebige Zeichen

➤ _ → Ersetzt ein beliebiges Zeichen



SQL Basics

➤ Daten auslesen

```
SELECT MIN(year) FROM movies;  
SELECT AVG(year) FROM movies WHERE director='Sidney Lumet';  
SELECT COUNT(*) FROM movies;  
SELECT COUNT(DISTINCT director) FROM movies;
```

- MIN, MAX, AVG → Beziehen sich auf einzelnes Feld
- COUNT(*)
- COUNT(DISTINCT director)



SQL Basics

➤ Daten auslesen

```
SELECT director, COUNT(title) FROM movies GROUP BY director;  
SELECT director, COUNT(title) FROM movies GROUP BY director  
HAVING COUNT(title)>10;  
SELECT year, director, COUNT(title) FROM movies  
GROUP BY director, year  
ORDER BY year DESC, director ASC;
```

Director	COUNT(title)
Director 1	12
Director 2	5



SQL Basics

➤ Daten auslesen

```
SELECT director, COUNT(title) FROM movies GROUP BY director;  
SELECT director, COUNT(title) FROM movies GROUP BY director  
HAVING COUNT(title)>10;  
SELECT year, director, COUNT(title) FROM movies  
GROUP BY director, year  
ORDER BY year DESC, director ASC;
```

Director	COUNT(title)
Director 1	12



SQL Basics

➤ Daten auslesen

```
SELECT director, COUNT(title) FROM movies GROUP BY director;  
SELECT director, COUNT(title) FROM movies GROUP BY director  
HAVING COUNT(title)>10;  
SELECT year, director, COUNT(title) FROM movies  
GROUP BY director, year  
ORDER BY year DESC, director ASC;
```

Year	Director	COUNT(title)
1995	Director 2	5
1980	Director 1	12



Modeling

- **Ziel** → So wenig duplizierte Info in der Datenbank wie möglich

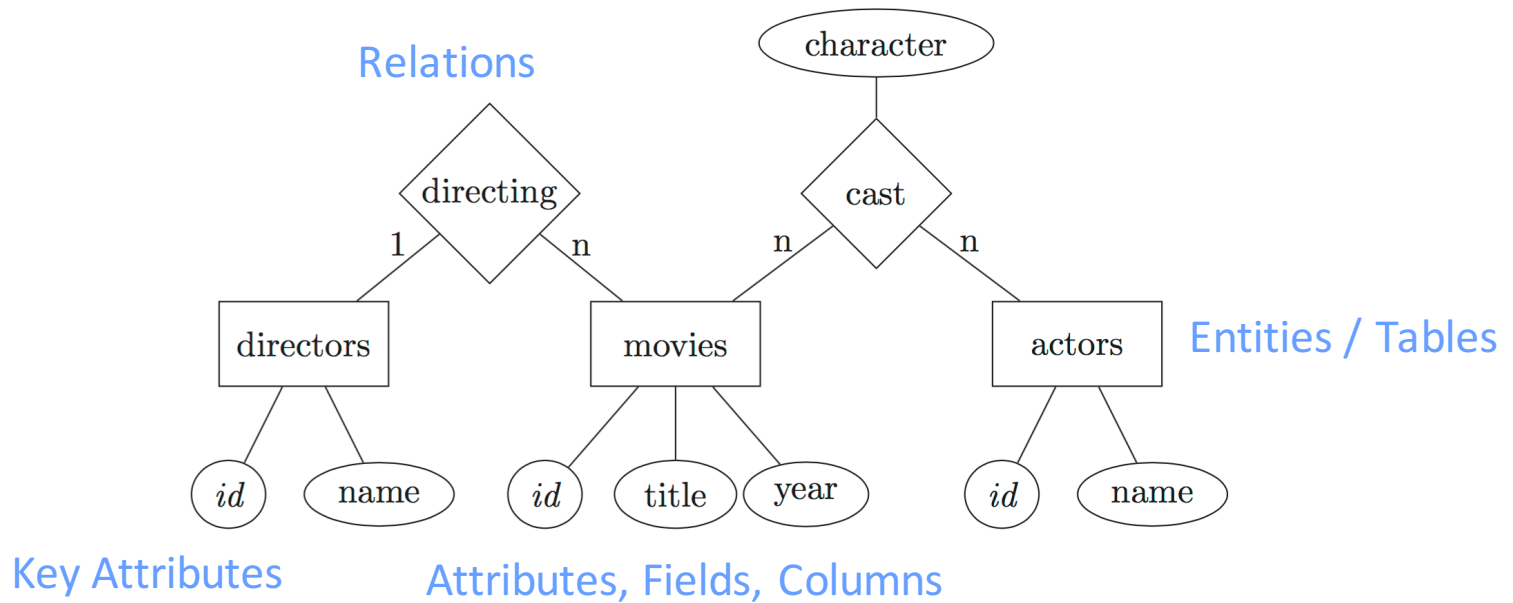
Movie	Director	Year	Actors
Movie 1	Director 2	1995	A, B, D, F
Movie 2	Director 1	2000	A, B
Movie 3	Director 3	1995	B, D, F
Movie 4	Director 1	2012	D

- **Entity-Relationship (ER) Diagramme** → Graphische Repräsentation



Modeling

➤ Entity-Relationship (ER) Diagramme





Modeling

➤ Entity-Relationship (ER) Diagramme

- Jede Tabelle sollte ein Key Attribute haben (ID)
- n-to-n Relations sind Tabellen mit Spalten für jedes Key Attribute
- 1-to-n & 1-to-1 Relations → Wie n-to-n oder Key von 1-Seite in Tabelle speichern

ACTORS	
ID	Name
1	A
2	B

MOVIES			
ID	Title	Year	AC_ID
1	MOV1	1990	2
2	MOV2	2000	1

CAST		
AC_ID	Char.	MO_ID
1	AAA	2
2	BBB	1



Joins

- **Zweck** → Zugang zu Daten aus anderen Tabellen erhalten

```
SELECT movie.title, director.name AS director, movie.year FROM movie
INNER JOIN director ON movie.director id = director.id;
```

- Für `director.name` muss auf Tabelle `director` zugegriffen werden
- AS → Aliasing / Neubenennung
- **INNER JOIN** → Condition für spez. Auswahl, nur Matches werden übernommen
- **LEFT OUTER JOIN** → Matches und alle Zeilen der linken Tabelle
 - Leere Felder werden mit NULL gefüllt
- **RIGHT OUTER JOIN**
- **FULL OUTER JOIN**



Keys & Constraints

- **Unique Key** → Eine / Mehrere Spalten, welche Zeile eindeutig bestimmen
- **Primary Key** → Designated Unique Key
- **Foreign Key** → Referenz auf Primary Key einer anderen Tabelle

```
ALTER TABLE movies ADD CONSTRAINT UNIQUE (actor_id, character, movie_id);  
ALTER TABLE director ADD PRIMARY KEY id;  
ALTER TABLE movies  
    ADD FOREIGN KEY (director_id) REFERENCES director;
```

- Die Kombination an Werten muss einmalig sein



Keys & Constraints

- **Unique Key** → Eine / Mehrere Spalten, welche Zeile eindeutig bestimmen
- **Primary Key** → Designated Unique Key
- **Foreign Key** → Referenz auf Primary Key einer anderen Tabelle

```
ALTER TABLE movies ADD CONSTRAINT UNIQUE (actor_id, character, movie_id);  
ALTER TABLE director ADD PRIMARY KEY id;  
ALTER TABLE movies  
    ADD FOREIGN KEY (director_id) REFERENCES director;
```

- Primary Key setzen → Automatische Garantie für einmalige Einträge



Keys & Constraints

- **Unique Key** → Eine / Mehrere Spalten, welche Zeile eindeutig bestimmen
- **Primary Key** → Designated Unique Key
- **Foreign Key** → Referenz auf Primary Key einer anderen Tabelle

```
ALTER TABLE movies ADD CONSTRAINT UNIQUE (actor_id, character, movie_id);  
ALTER TABLE director ADD PRIMARY KEY id;  
ALTER TABLE movies  
    ADD FOREIGN KEY (director_id) REFERENCES director;
```

- Garantiert, dass Werte der Tabelle Primary Keys sind in referenzierter Tabelle



Indexing

- **Index** → Datenstruktur, welche Suchen beschleunigt
- Ohne Index muss jeweils ganze Tabelle durchsucht werden
- Häufig werden Hash-Tabellen verwendet

```
CREATE INDEX directorid ON director USING HASH (id);
```



Empfohlene Übungen

➤ Serie 7

- Aufgabe 2 (Basic)
- Aufgabe 3 (Advanced)
- Aufgabe 4 (Advanced)



Inhalt

- Concurrency
- RMW Instruktionen
- Semaphore
- A-Lock
- MCS Lock
- Locks für Mengen
- Locks für Hashing



Prozesse

- Prozesse können **independent** oder **cooperating** sein
- Prozesse kommunizieren durch **Shared Memory** oder **Message Passing**
 - Zum Beispiel Arrays für Shared Memory
 - Zum Beispiel Sockets oder Pipes (ls | more) für Message Passing



Concurrency

- **Race Condition** → Resultat einer Ausführung hängt Reihenfolge ab

```
x = 3;  
y = 4;  
z = x * y;
```

```
x = 2;  
y = 3;  
z = x + y;
```

```
x = 3;  
y = 4;  
z = 12;  
x = 2;  
y = 3;  
z = 5;
```

```
x = 2;  
y = 3;  
z = 5;  
x = 3;  
y = 4;  
z = 12;
```

```
x = 3;  
y = 4;  
x = 2;  
z = 6;  
y = 3;  
z = 6;
```

- **Critical Section** → Zeilen in Code von Prozess mit Shared Resources

```
x = 3;  
y = 4;  
x = x + 2;  
z = x * y;
```

```
a = 1;  
x = 3;  
y = 4;  
z = x * y;  
b = 2;  
a = a * b;
```



Concurrency

- **Critical Section Problem** → Wollen koordinierten Ablauf garantieren
 - Mutual Exclusion: Nur ein Prozess in Critical Section
 - Progress: Prozesse in normalem Kontext haben keinen Einfluss
 - Bounded Waiting: Prozesse müssen nicht ewig auf Zutritt warten
- **Preemptive / Nonpreemptive** → Kann Prozess Ressource entnommen werden?
- **Atomare Operation** → „Uninterruptable“



Caches

- Prozessor-eigener Speicher
- Klein aber schnell
- Cache Hit und Cache Miss
- Cache Kohärenz
 - Daten müssen immer up-to-date sein



Read-Modify-Write (RMW) Instruktionen

- „Synchronized“
- Sind atomar!
- **TestAndSet**
- **GetAndIncrement**
- **CompareAndSwap**

```
synchronized int testAndSet () {  
    int prior = value;  
    value = 1;  
    return prior; }
```

```
synchronized int getAndIncrement () {  
    int prior = value;  
    value = value + 1;  
    return prior; }
```

```
synchronized int compareAndSwap (int old, int new) {  
    int prior = value;  
    if (value == old) value = new;  
    return prior; }
```



Locks

- Ansatz für Mutual Exclusion

```
do {  
    acquire lock;  
    critical section  
    release lock;  
    remainder section  
} while (TRUE);
```

- Verwenden RMW Instruktionen, um Lock zu implementieren



Lock mit TestAndSet (TAS) / GetAndSet

➤ RMW Instruktion

```
synchronized int testAndSet () {  
    int prior = value;  
    value = 1;  
    return prior; }
```

➤ Locking

```
Public void lock() {  
    while (state.testAndSet()) {} }
```

➤ Unlock

```
Public void unlock() {  
    state.set(0); }
```



Lock mit TestAndTestAndSet (TTAS)

➤ RMW Instruktion

```
synchronized int testAndSet () {  
    int prior = value;  
    value = 1;  
    return prior; }
```

➤ Locking

```
Public void lock() {  
    while (true) {  
        while (state.get()) {}  
        if (!state.getAndSet())  
            return;  
    }  
}
```



Peterson Algorithmus

➤ Erster (Software) Ansatz für Mutual Exclusion (zweier Prozesse)

➤ Shared Resources

- int turn;

- boolean flag[2];

```
do {  
  
    flag[i] = TRUE;  
    turn = j;  
    while (flag[j] && turn == j);  
  
    critical section  
  
    flag[i] = FALSE;  
  
    remainder section  
  
} while (TRUE);
```



Semaphore

- (Hardware) Ansatz für Mutual Exclusion
- Integer Variable (S)
- 2 atomare Operationen
 - **Wait()**
 - **Signal()**
- 2 Varianten
 - **Counting Semaphore**: Wert von $S > 1$
 - **Binäre Semaphore (Mutex Lock)**
- Problem → Busy Waiting

```
Wait (S) {  
    while S <= 0;  
    S--;  
}
```

```
Signal (S) {  
    S++;  
}
```

```
do {  
    wait (S);  
  
    critical section  
  
    signal(S);  
  
    remainder section  
} while (TRUE);
```

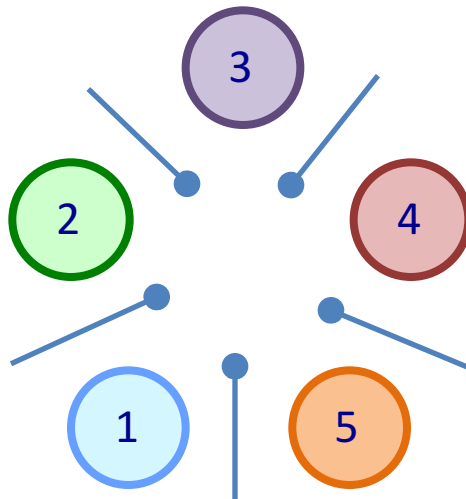


Gefahren bei Semaphoren

- **Deadlocks** → Alle Prozesse sind blockiert
 - Klassisches Beispiel: Dining Philosophers

Prozess 1
Wait (S);
Wait(Q);

Prozess 2
Wait (Q);
Wait(S);



Nimm Löffel links;
Nimm Löffel rechts;
Esse;
Lege Besteck zurück;
Fange von vorne an;

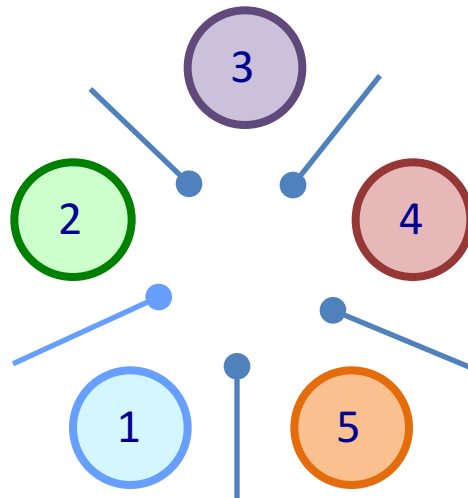


Gefahren bei Semaphoren

- **Deadlocks** → Alle Prozesse sind blockiert
 - Klassisches Beispiel: Dining Philosophers

Prozess 1
Wait (S);
Wait(Q);

Prozess 1
Wait (Q);
Wait(S);



Nimm Löffel links;
Nimm Löffel rechts;
Esse;
Lege Besteck zurück;
Fange von vorne an;

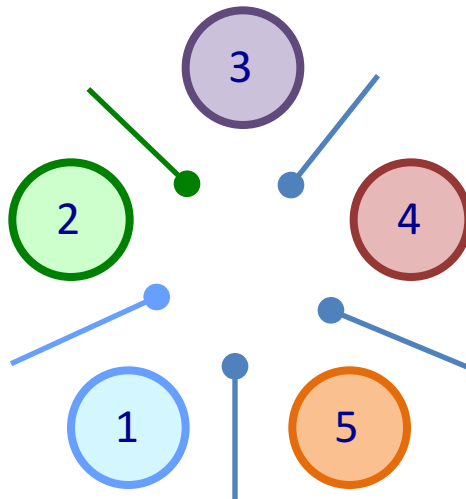


Gefahren bei Semaphoren

- **Deadlocks** → Alle Prozesse sind blockiert
 - Klassisches Beispiel: Dining Philosophers

Prozess 1
Wait (S);
Wait(Q);

Prozess 1
Wait (Q);
Wait(S);



Nimm Löffel links;
Nimm Löffel rechts;
Esse;
Lege Besteck zurück;
Fange von vorne an;

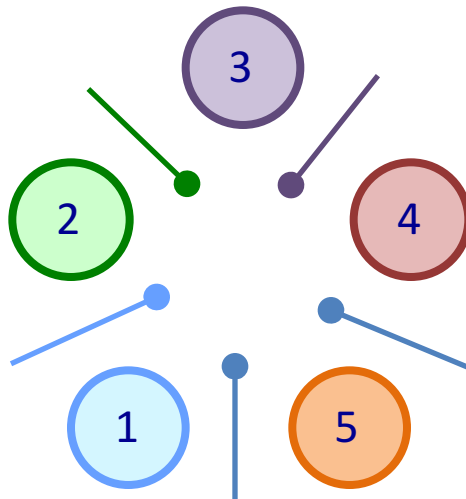


Gefahren bei Semaphoren

- **Deadlocks** → Alle Prozesse sind blockiert
 - Klassisches Beispiel: Dining Philosophers

Prozess 1
Wait (S);
Wait(Q);

Prozess 1
Wait (Q);
Wait(S);



Nimm Löffel links;
Nimm Löffel rechts;
Esse;
Lege Besteck zurück;
Fange von vorne an;

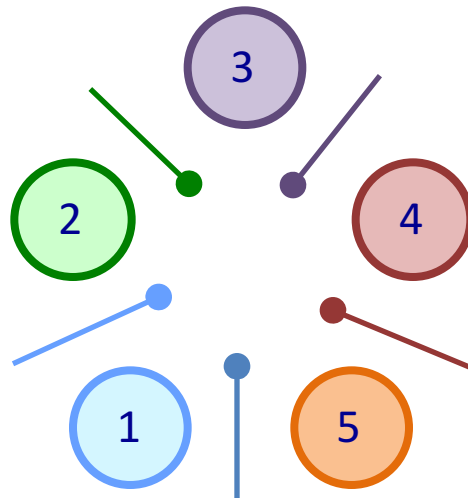


Gefahren bei Semaphoren

- **Deadlocks** → Alle Prozesse sind blockiert
 - Klassisches Beispiel: Dining Philosophers

Prozess 1
Wait (S);
Wait(Q);

Prozess 1
Wait (Q);
Wait(S);



Nimm Löffel links;
Nimm Löffel rechts;
Esse;
Lege Besteck zurück;
Fange von vorne an;

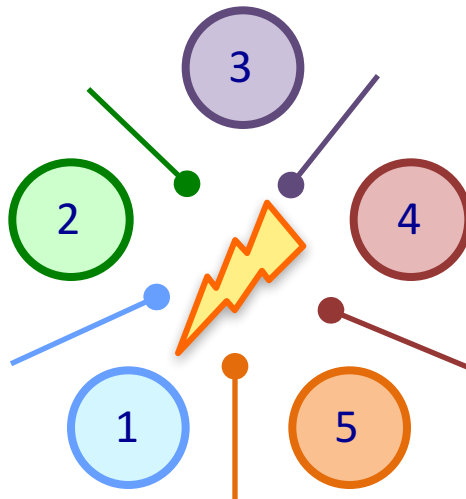


Gefahren bei Semaphoren

- **Deadlocks** → Alle Prozesse sind blockiert
 - Klassisches Beispiel: Dining Philosophers

Prozess 1
Wait (S);
Wait(Q);

Prozess 1
Wait (Q);
Wait(S);



Nimm Löffel links;
Nimm Löffel rechts;
Esse;
Lege Besteck zurück;
Fange von vorne an;



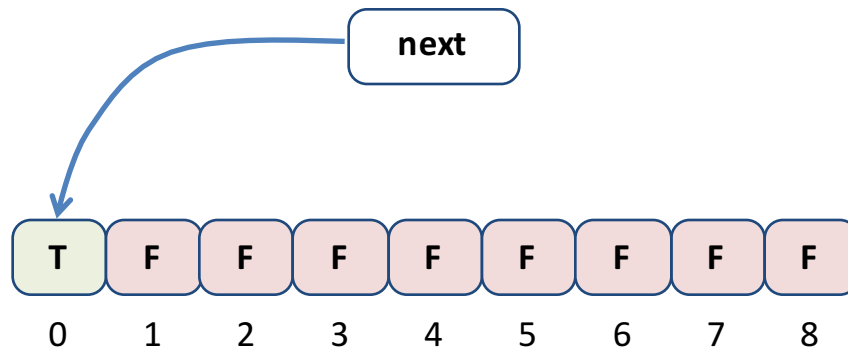
Gefahren bei Semaphoren

- **Starvation** → Gewisse Prozesse verhungern
 - Preemptive / Nonpreemptive?



Anderson Queue Lock (Alock)

- Array-basiertes Lock
- Shared Resource → Tail Field „next“
- RMW Instruktion → GetAndIncrement



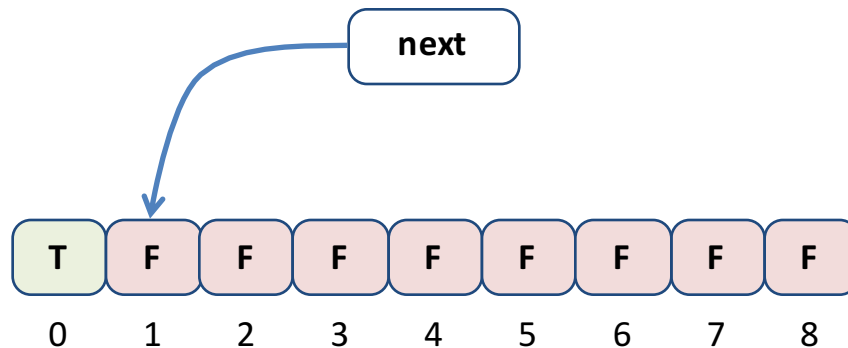
Prozess 1:

- Flag 0 free!
- Lock
- Increment next



Anderson Queue Lock (Alock)

- Array-basiertes Lock
- Shared Resource → Tail Field „next“
- RMW Instruktion → GetAndIncrement



Prozess 1:

- Executing

Prozess 2:

- Flag 1 busy!

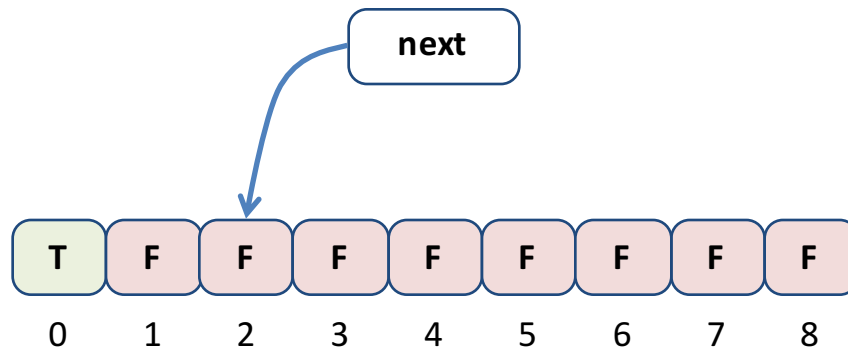
- Spin

- Increment next



Anderson Queue Lock (Alock)

- Array-basiertes Lock
- Shared Resource → Tail Field „next“
- RMW Instruktion → GetAndIncrement



Prozess 1:

- Executing

Prozess 3:

- Flag 2 busy!

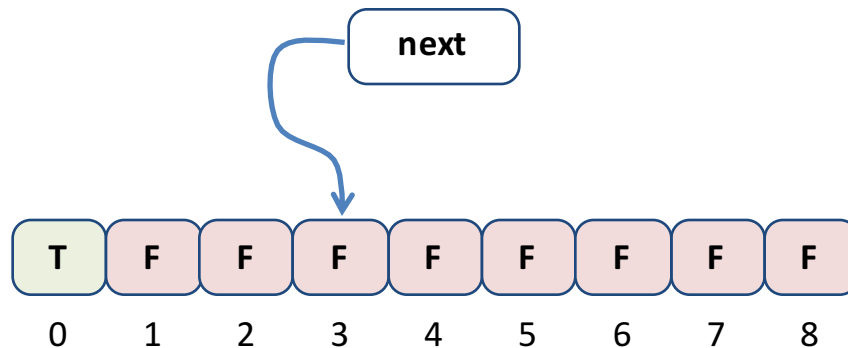
- Spin

- Increment next



Anderson Queue Lock (Alock)

- Array-basiertes Lock
- Shared Resource → Tail Field „next“
- RMW Instruktion → GetAndIncrement



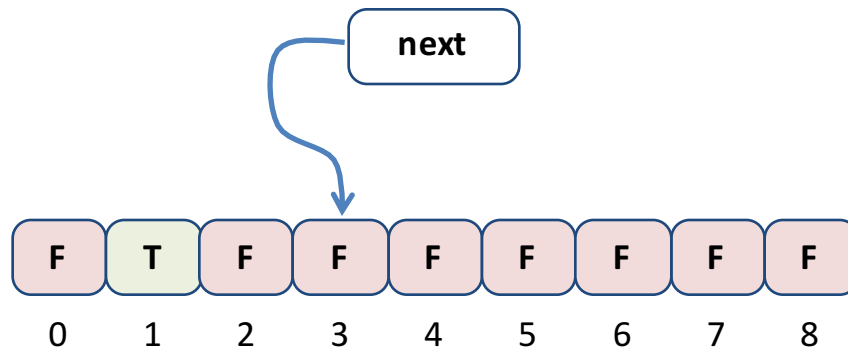
Prozess 1:

- Finished
- Release lock!
- Set slot 1 to true



Anderson Queue Lock (Alock)

- Array-basiertes Lock
- Shared Resource → Tail Field „next“
- RMW Instruktion → GetAndIncrement



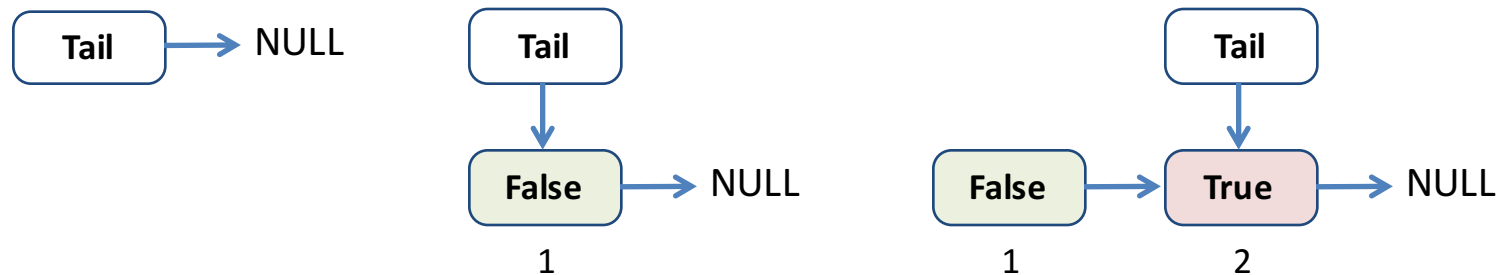
Prozess 2:

- Lock free!
- Lock!
- Execute



MCS Lock

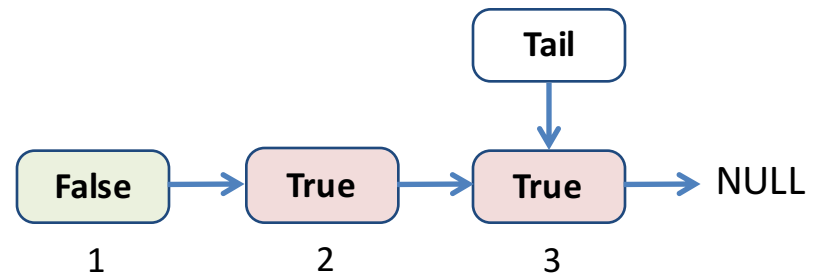
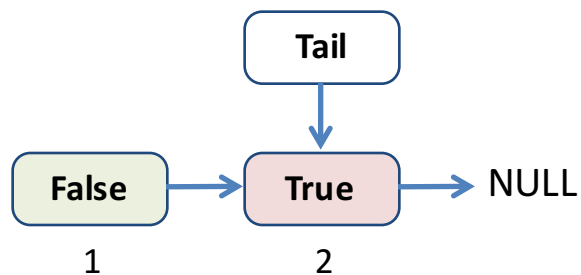
- Linked List statt Array
- Ein Knoten pro Thread / Prozess
- Shared Resource → Tail von Liste





MCS Lock

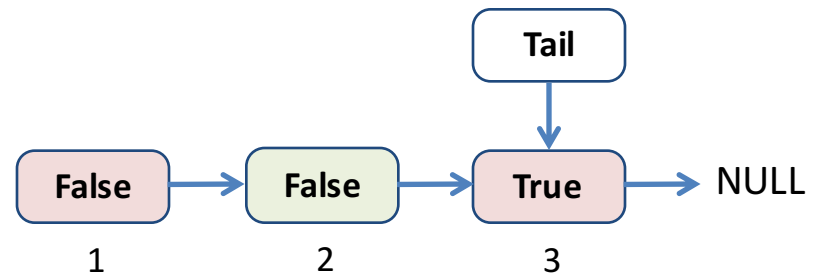
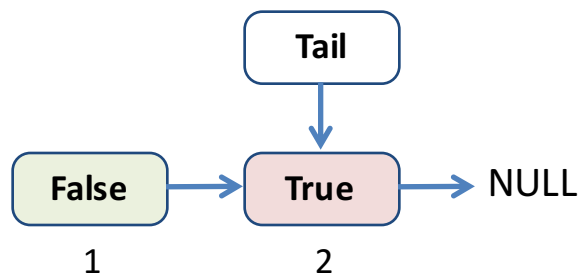
- Linked List statt Array
- Ein Knoten pro Thread / Prozess
- Shared Resource → Tail von Liste





MCS Lock

- Linked List statt Array
- Ein Knoten pro Thread / Prozess
- Shared Resource → Tail von Liste





Abortable Locks

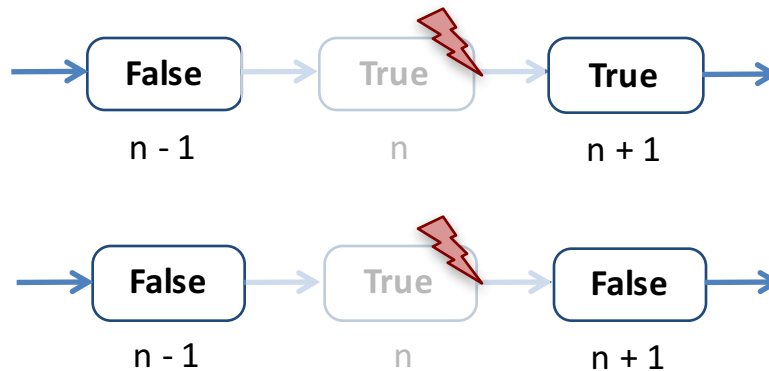
- Prozess will nicht mehr länger auf Lock warten
- Einfach für Semaphore, ...
- Komplizierter für Queue Locks (MSC Locks)





Abortable Locks

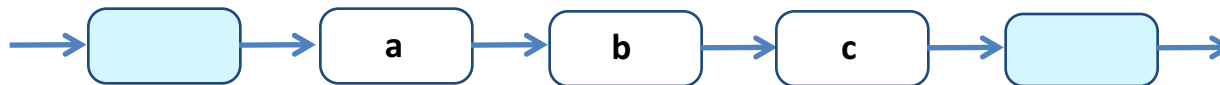
- Lösung → Abortion Flag
 - Gesetz: Aborted
 - Nicht gesetzt: Nicht aborted





Locks für Sets

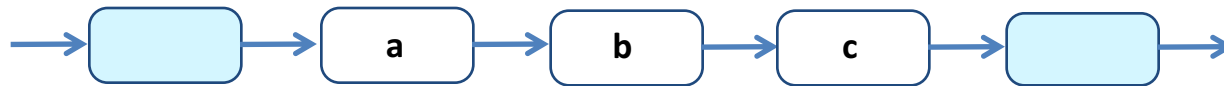
- „Basic Locks“ sind nicht optimal für Mengen von Daten
- Betrachten Listen
- 3 grundlegende Operationen
 - add(x)
 - remove(x)
 - contains(x)





Locks für Sets

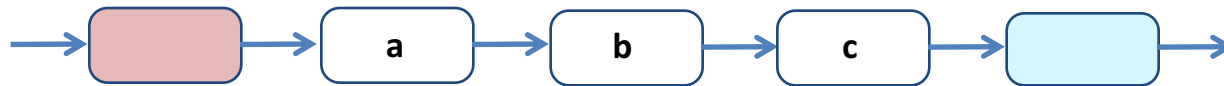
- **Coarse-Grained Locking** → Sperre ganze Liste für jede Operation





Locks für Sets

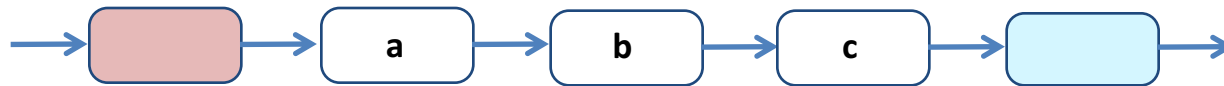
- **Coarse-Grained Locking** → Sperre ganze Liste für jede Operation



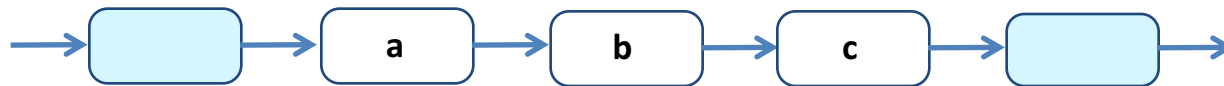


Locks für Sets

- **Coarse-Grained Locking** → Sperre ganze Liste für jede Operation



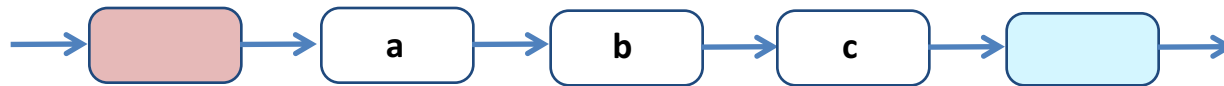
- **Fine-Grained Locking / Hand-Over-Hand Locking**



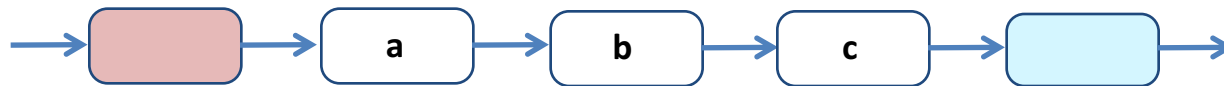


Locks für Sets

- **Coarse-Grained Locking** → Sperre ganze Liste für jede Operation



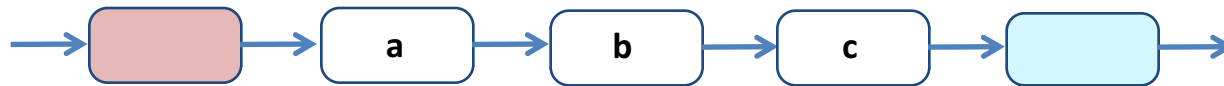
- **Fine-Grained Locking / Hand-Over-Hand Locking**



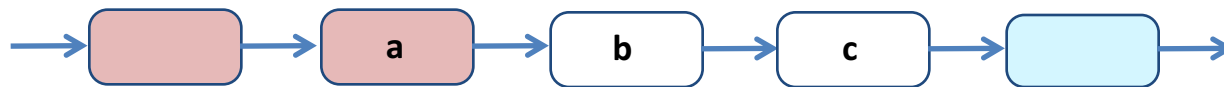


Locks für Sets

- **Coarse-Grained Locking** → Sperre ganze Liste für jede Operation



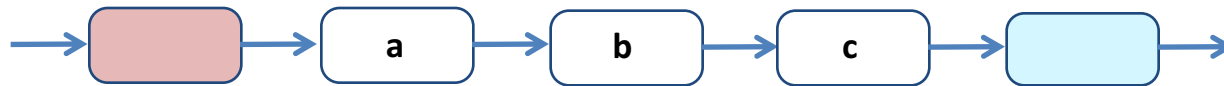
- **Fine-Grained Locking / Hand-Over-Hand Locking**



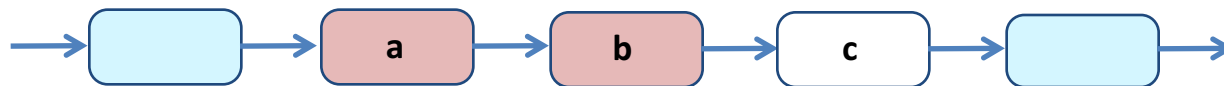


Locks für Sets

- **Coarse-Grained Locking** → Sperre ganze Liste für jede Operation



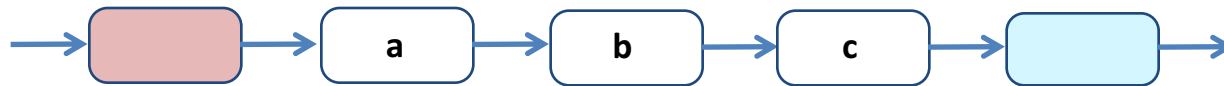
- **Fine-Grained Locking / Hand-Over-Hand Locking**



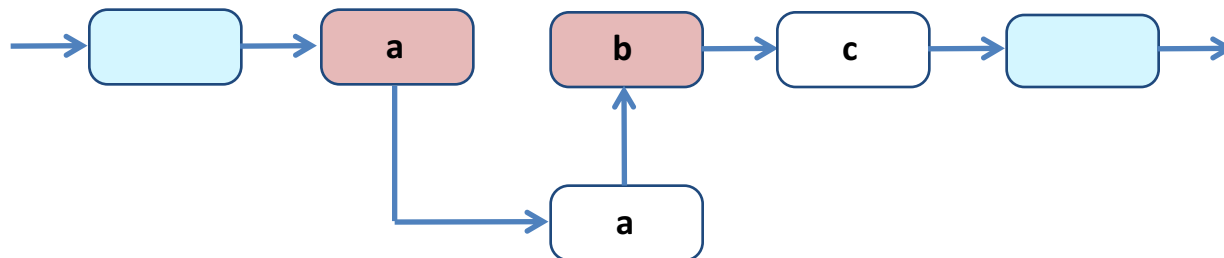


Locks für Sets

- **Coarse-Grained Locking** → Sperre ganze Liste für jede Operation



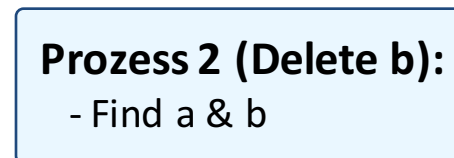
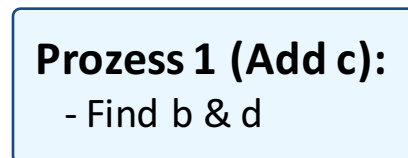
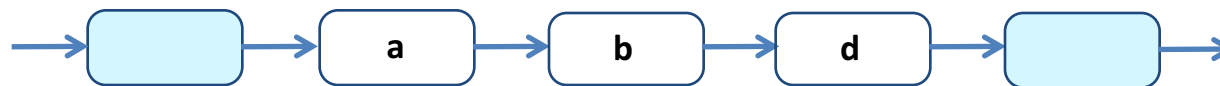
- **Fine-Grained Locking / Hand-Over-Hand Locking**





Locks für Sets

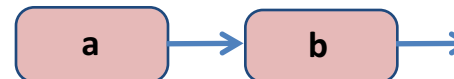
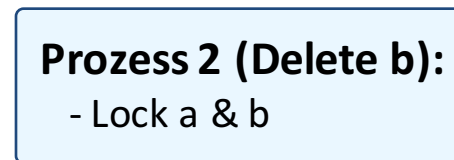
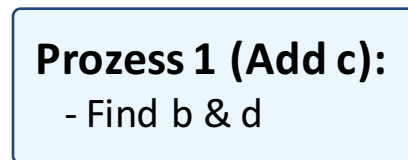
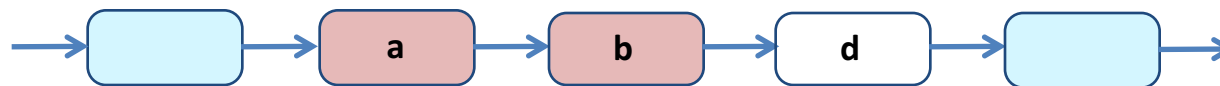
- **Optimistic Synchronization** → Liste traversieren ohne alle Knoten zu locken
 - Wenn Knoten gefunden, locken
 - Nach Locken, überprüfen, dass Predecessor immer noch existiert und auf Knoten zeigt
 - Gute Methode, falls 2x Traversieren ohne Locking günstiger ist als 1x mit Locking





Locks für Sets

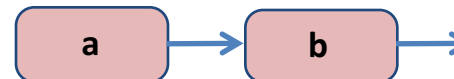
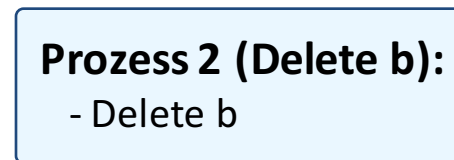
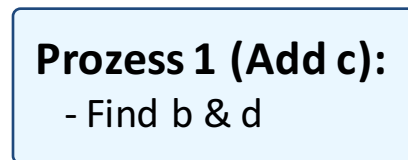
- **Optimistic Synchronization** → Liste traversieren ohne alle Knoten zu locken
 - Wenn Knoten gefunden, locken
 - Nach Locken, überprüfen, dass Predecessor immer noch existiert und auf Knoten zeigt
 - Gute Methode, falls 2x Traversieren ohne Locking günstiger ist als 1x mit Locking





Locks für Sets

- **Optimistic Synchronization** → Liste traversieren ohne alle Knoten zu locken
 - Wenn Knoten gefunden, locken
 - Nach Locken, überprüfen, dass Predecessor immer noch existiert und auf Knoten zeigt
 - Gute Methode, falls 2x Traversieren ohne Locking günstiger ist als 1x mit Locking





Locks für Sets

- **Optimistic Synchronization** → Liste traversieren ohne alle Knoten zu locken
 - Wenn Knoten gefunden, locken
 - Nach Locken, überprüfen, dass Predecessor immer noch existiert und auf Knoten zeigt
 - Gute Methode, falls 2x Traversieren ohne Locking günstiger ist als 1x mit Locking

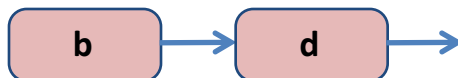


Prozess 1 (Add c):

- Lock b & d

Prozess 2 (Delete b):

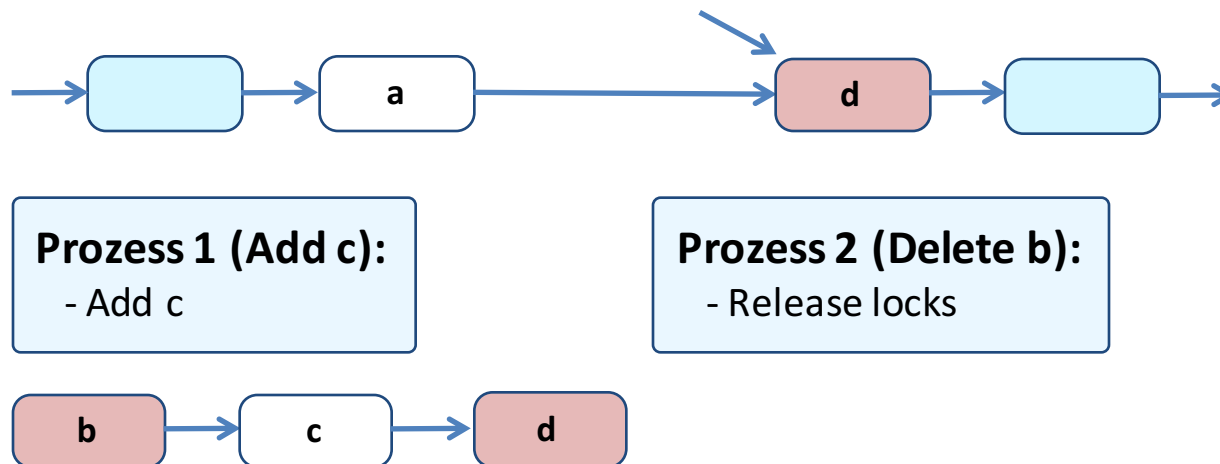
- Release locks





Locks für Sets

- **Optimistic Synchronization** → Liste traversieren ohne alle Knoten zu locken
 - Wenn Knoten gefunden, locken
 - Nach Locken, überprüfen, dass Predecessor immer noch existiert und auf Knoten zeigt
 - Gute Methode, falls 2x Traversieren ohne Locking günstiger ist als 1x mit Locking





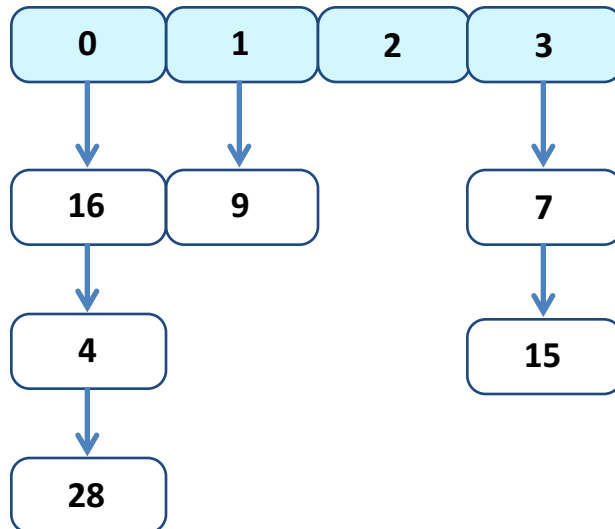
Locks für Sets

- **Lazy Synchronization** → Logisches Löschen
 - Alle Knoten erhalten neues Attribut „Marked“: In Liste oder nicht
 - Funktion Contains: Einmal Liste traversieren ohne Contention oder Waits
 - Funktion Add: Einmal Liste traversieren, Predecessor und Current locken und einfügen
 - Funktion Remove: Zwei Schritte (logisches und physikalisches Löschen), kein Validieren
- **Lock-Free Data Structures** → Verwende RMW Instruktionen
 - z.B. mit CAS: Logisches Löschen und Pointer ändern in einem Schritt
 - Atomic Markable Reference



Locks für Hashing

- Coarse-Grained Locking
- Fine-Grained Locking

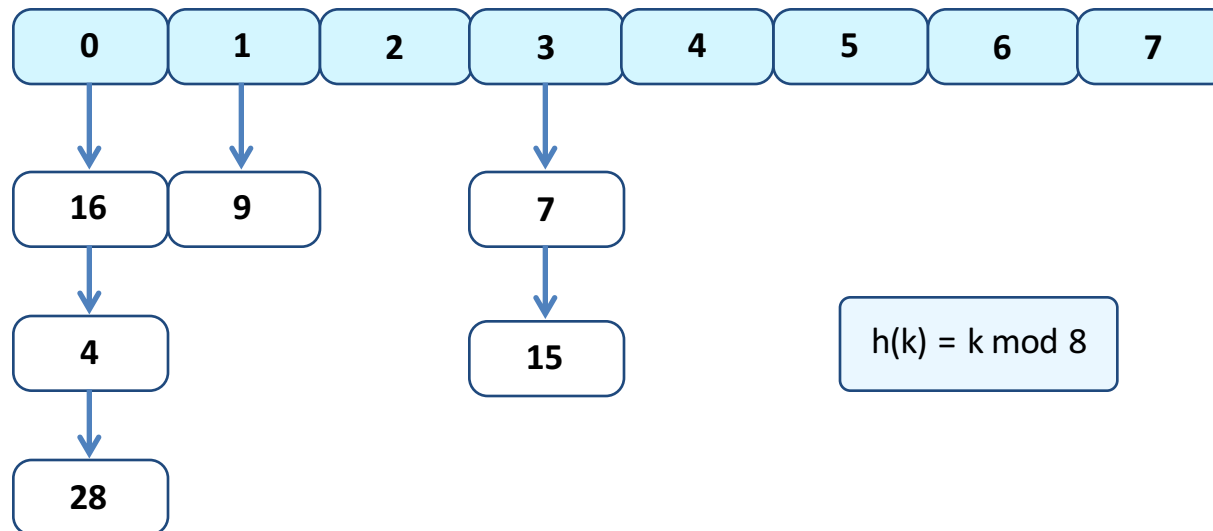


$$h(k) = k \bmod 4$$



Locks für Hashing

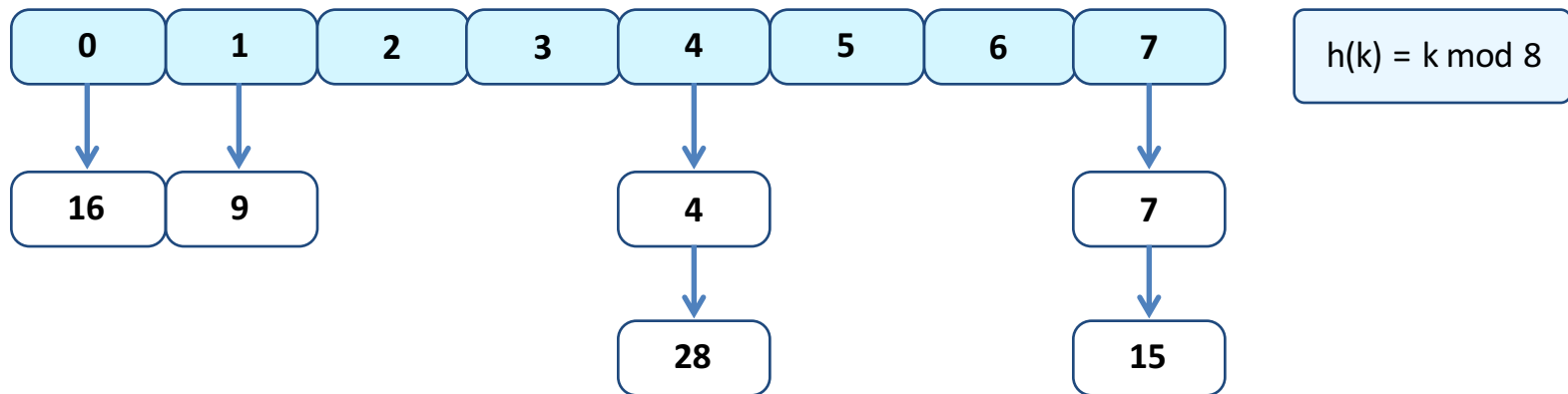
- Hashtabelle ineffizient, falls Listen zu lange werden
- Hashtabelle muss vergrößert und Hashfunktion angepasst werden





Locks für Hashing

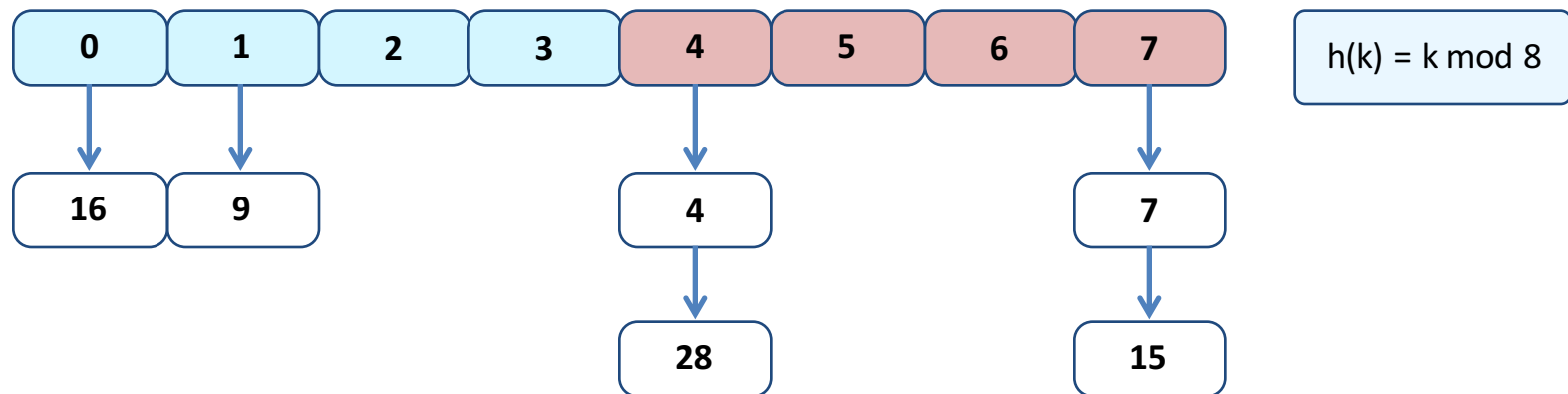
- Vergrössern, falls gewisse Grösse erreicht wird
 - Global Threshold
 - Bucket Threshold





Locks für Hashing

- Problem mit Resizing und Fine-Grained Locking
 - Wir vergrößern die Anzahl Locks nicht!

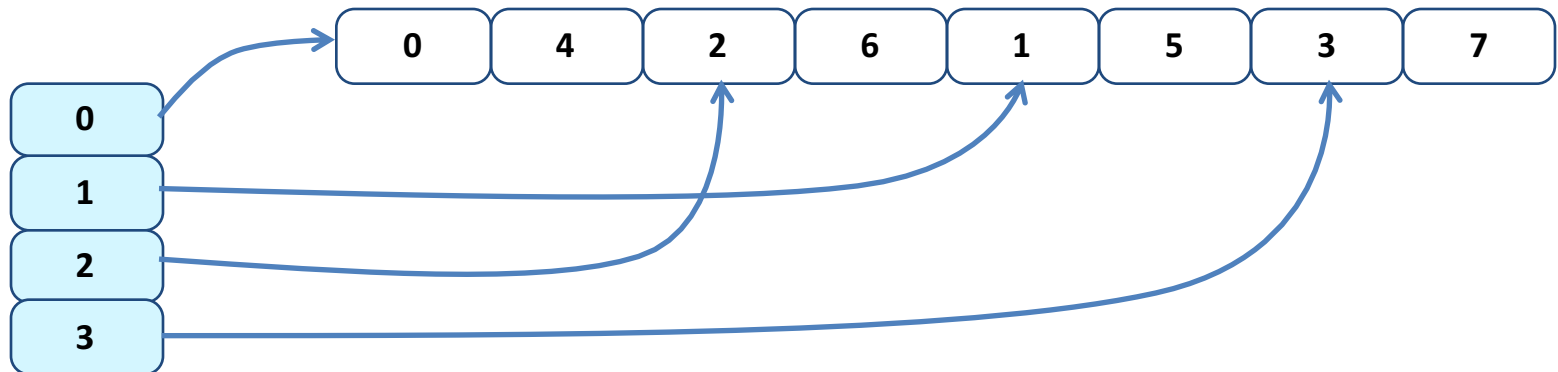




Locks für Hashing

➤ Lock-Free Resizing Problem

- Vergrössern: Bewege die Bucket statt die Elemente
- Elemente sind in Liste gespeichert
- Buckets werden zu „Shortcut-Pointer“
- Liste muss richtig geordnet sein

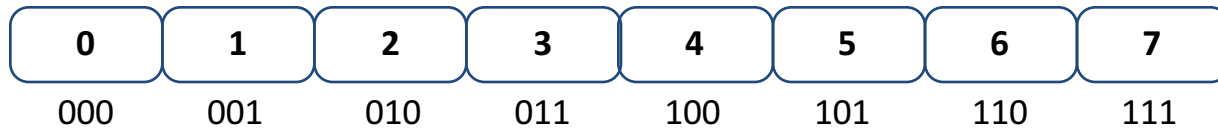




Locks für Hashing

➤ Lock-Free Resizing Problem

- LSB geben Bucket an
- Vergrössern: Bit $i+1$ gibt an, ob Bucket gewechselt werden muss
- Richtigen Index finden: Spiegeln Binärrepräsentation der Keys

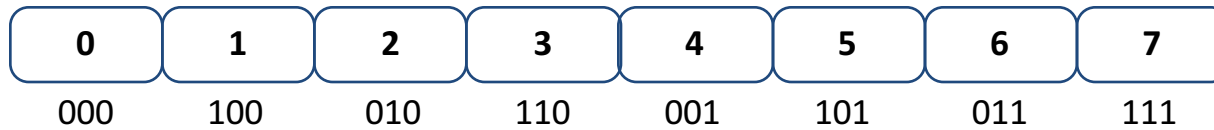




Locks für Hashing

➤ Lock-Free Resizing Problem

- LSB geben Bucket an
- Vergrössern: Bit $i+1$ gibt an, ob Bucket gewechselt werden muss
- Richtigen Index finden: Spiegler Binärrepräsentation der Keys

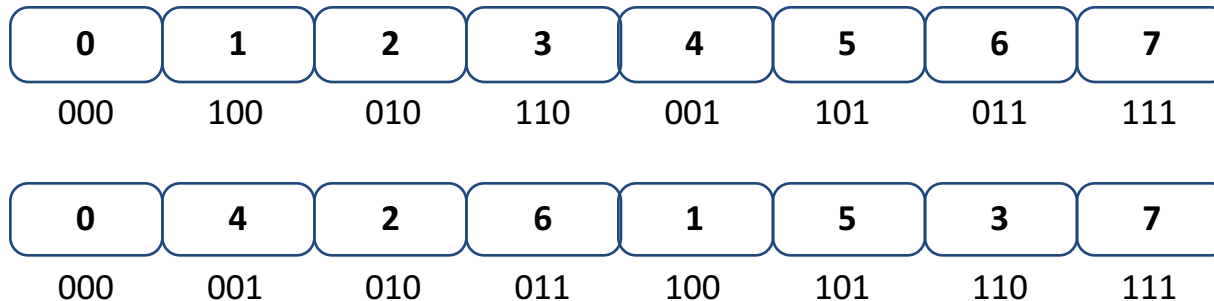




Locks für Hashing

➤ Lock-Free Resizing Problem

- LSB geben Bucket an
- Vergrössern: Bit $i+1$ gibt an, ob Bucket gewechselt werden muss
- Richtigen Index finden: Spiegeln Binärrepräsentation der Keys

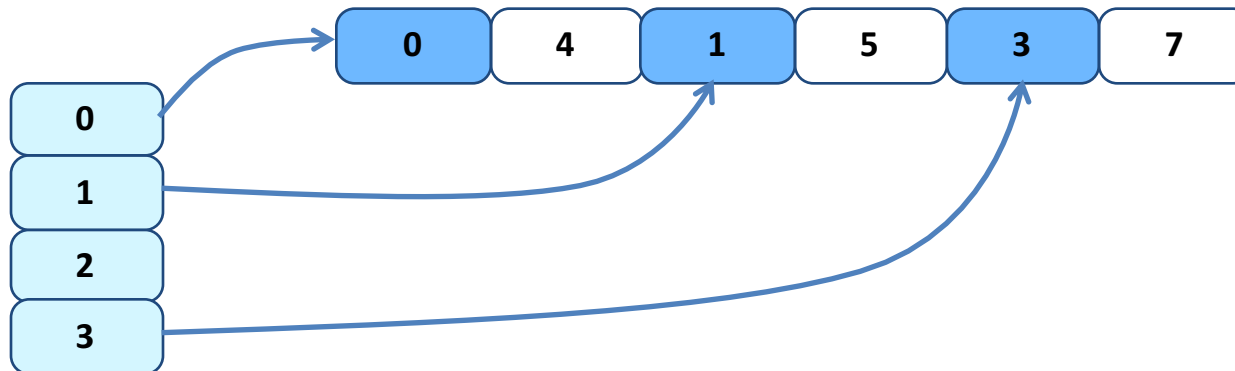




Locks für Hashing

➤ Lock-Free Resizing Problem

- Schwierigkeit: 2 Pointer zeigen auf gewisse Elemente; wie löschen?
- „Sentinel Nodes“: Markierungsknoten





Empfohlene Übungen

➤ Serie 4

- Aufgabe 2 (Basic)
- Aufgabe 3 (Advanced)

➤ Serie 5

- Aufgabe 2 (Advanced)

➤ Serie 8

- Aufgabe 3 (Advanced)



Weiteres Übungsmaterial

➤ Technische Informatik 2

- <http://www.csg.ethz.ch/education/lectures/TI2/FS2015/uebungen>
- Locks, Storage

➤ AMIV

- <https://www.amiv.ethz.ch/studium/unterlagen/99>
- Locks, Kommunikation, Storage

➤ Communication Networks

- http://comm-net.ethz.ch/pdfs/sample_questions.pdf
- Kommunikation

➤ Discrete Event Systems

- <http://dcg.ethz.ch/lectures/des/>
- Markovketten