



# Datenbanksysteme

## Postgres interne Speicherverwaltung und Zugriffsstrukturen

Prof. Dr. Patrick Cato

# Wo speichert Postgres die Daten?



- Der Speicherort kann mittels `SHOW data_directory;` ermittelt werden
- Im „base directory“ sind die Datenbanken und Metadaten gespeichert
- Jede Datenbanken sowie Relation hat einen Objectidentifier (oid) und jede Datenbank erhält einen eigenen Ordner

`SHOW data_directory;`

data_directory	
	text
1	C:/Program Files/PostgreSQL/14/data

Programme > PostgreSQL > 14 > data			
	Name	Änderungsdatum	Typ
	base	03.04.2023 15:25	Dateiordner
	global	03.04.2023 15:26	Dateiordner
	log	15.03.2023 09:05	Dateiordner
	pg_commit_ts	06.05.2022 17:19	Dateiordner
	pg_dynshmem	06.05.2022 17:19	Dateiordner
	pg_logical	03.04.2023 15:28	Dateiordner
	pg_multixact	06.05.2022 17:19	Dateiordner
	pg_notify	06.05.2022 17:19	Dateiordner
	pg_replslot	06.05.2022 17:19	Dateiordner
	pg_serial	06.05.2022 17:19	Dateiordner
	pg_snapshots	06.05.2022 17:19	Dateiordner
	pg_stat	03.04.2023 15:07	Dateiordner
	pg_stat_tmp	03.04.2023 15:35	Dateiordner
	pg_subtrans	06.05.2022 17:19	Dateiordner
	pg_tblspc	06.05.2022 17:19	Dateiordner
	pg_twophase	06.05.2022 17:19	Dateiordner
	pg_wal	03.04.2023 15:33	Dateiordner
	pg_xact	06.05.2022 17:19	Dateiordner
	current_logfiles	03.04.2023 15:07	Datei
	pg_hba.conf	06.05.2022 17:19	CONF-Datei
	pg_ident.conf	06.05.2022 17:19	CONF-Datei
	PG_VERSION	06.05.2022 17:19	Datei
	postgresql.auto.conf	06.05.2022 17:19	CONF-Datei
	postgresql.conf	06.05.2022 17:19	CONF-Datei
	postmaster.opts	03.04.2023 15:07	OPTS-Datei
	postmaster.pid	03.04.2023 15:07	PID-Datei

# Wie kann die Zuordnung des Identifier zur Datenbank ermittelt werden?

```
1 SELECT oid, datname  
2 FROM pg_database;
```

Wrap Contents

**Execute** or **Discard**

Rows 7		<input type="button" value="CSV"/>	<input type="button" value="JSON"/>	<input type="button" value="CSV"/>	<input type="button" value="JSON"/>
oid	datname				
13754	postgres				
16394	praktikum				
1	template1				
13753	template0				
25071	kiklausur				
25139	ffiklausur				
25175	instgram				

# Im Ordner liegen die Dateien sowie Verwaltungsdaten



- In den Dateien sind die Rohdaten sowie Metadaten (beispielsweise Secondary Index gespeichert)
- Um die Zuordnung der Tabellendaten auf die Dateien herauszufinden, muss folgende Query angewendet werden:

```
select * from pg_class;
```

- Das File (default) wird max. 1GB groß. Dann gibt es ein neues, zusätzliches File.

```
1 select * from pg_class;
```

Wrap Contents

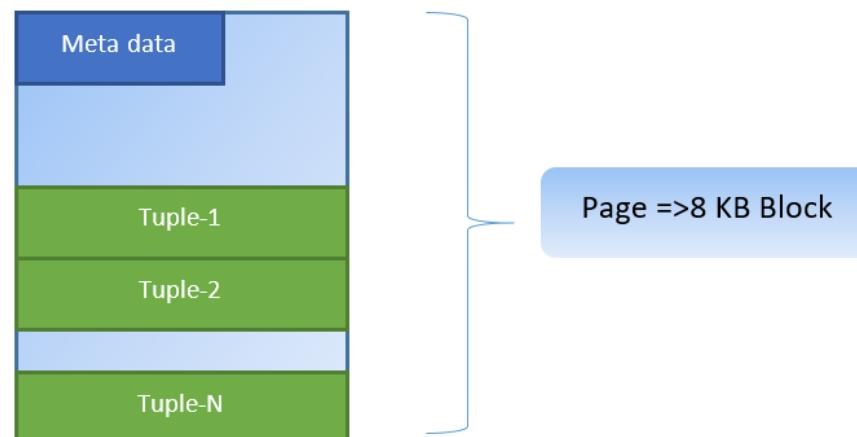
Execute or Discard

oid	relname	relnamespace	reltype	reloftype	relowner	relam	relfilenode	reltablespace	relpages	reltuples	relallvisible	reltoastrelid	relhasindex	relisshared	relpersistence	relkind	relnatts	relchecks	r
25180	caption_tags_id...	2200	0	0	10	0	25180	0	1	1	0	0	false	false	p	S	3	0	false
25186	comments_id_seq	2200	0	0	10	0	25186	0	1	1	0	0	false	false	p	S	3	0	false
25191	followers_id_seq	2200	0	0	10	0	25191	0	1	1	0	0	false	false	p	S	3	0	false
25196	hashtags_id_seq	2200	0	0	10	0	25196	0	1	1	0	0	false	false	p	S	3	0	false
25200	hashtags_posts_i...	2200	0	0	10	0	25200	0	1	1	0	0	false	false	p	S	3	0	false
25206	likes_id_seq	2200	0	0	10	0	25206	0	1	1	0	0	false	false	p	S	3	0	false
25212	photo_tags_id_seq	2200	0	0	10	0	25212	0	1	1	0	0	false	false	p	S	3	0	false
25187	followers	2200	25189	0	10	2	25187	0	2713	425852	2713	0	true	false	p	r	4	0	false
25220	posts_id_seq	2200	0	0	10	0	25220	0	1	1	0	0	false	false	p	S	3	0	false
25213	posts	2200	25215	0	10	2	25213	0	239	12932	239	0	true	false	p	r	8	2	false
25227	pg_toast_25221	99	0	0	10	2	25227	0	0	-1	0	0	true	false	p	t	3	0	false
25228	pg_toast_25221...	99	0	0	10	403	25228	0	1	0	0	0	false	false	p	i	2	0	false
25192	hashtags	2200	25194	0	10	2	25192	0	2	184	0	0	true	false	p	r	3	0	false
25229	users_id_seq	2200	0	0	10	0	25229	0	1	1	0	0	false	false	p	S	3	0	false
25239	caption_tags_pkey	2200	0	0	10	403	25239	0	56	19495	0	0	false	false	p	i	1	0	false
25241	caption_tags_us...	2200	0	0	10	403	25241	0	56	19495	0	0	false	false	p	i	2	0	false
25243	comments_pkey	2200	0	0	10	403	25243	0	168	60410	0	0	false	false	p	i	1	0	false
25245	followers_leader...	2200	0	0	10	403	25245	0	1171	425852	0	0	false	false	p	i	2	0	false
25247	followers_pkey	2200	0	0	10	403	25247	0	1171	425852	0	0	false	false	p	i	1	0	false
25249	hashtags_pkey	2200	0	0	10	403	25249	0	2	184	0	0	false	false	p	i	1	0	false
25251	hashtags_posts_...	2200	0	0	10	403	25251	0	91	32385	0	0	false	false	p	i	2	0	false
25253	hashtags_posts_...	2200	0	0	10	403	25253	0	91	32385	0	0	false	false	p	i	1	0	false
25255	hashtags_title_key	2200	0	0	10	403	25255	0	2	184	0	0	false	false	p	i	1	0	false
25257	likes_pkey	2200	0	0	10	403	25257	0	2065	752009	0	0	false	false	p	i	1	0	false
25259	likes_user_id_po...	2200	0	0	10	403	25259	0	2897	752009	0	0	false	false	p	i	3	0	false
25261	photo_tags_pkey	2200	0	0	10	403	25261	0	46	15902	0	0	false	false	p	i	1	0	false
25263	photo_tags_user...	2200	0	0	10	403	25263	0	46	15902	0	0	false	false	p	i	2	0	false
25265	posts_pkey	2200	0	0	10	403	25265	0	38	12932	0	0	false	false	p	i	1	0	false
25267	users_pkey	2200	0	0	10	403	25267	0	17	5345	0	0	false	false	p	i	1	0	false
25176	caption_tags	2200	25178	0	10	2	25176	0	125	19495	125	0	true	false	p	r	4	0	false
25221	users	2200	25223	0	10	2	25221	0	109	5345	109	25227	true	false	p	r	10	1	false

<input type="checkbox"/> Name	Änderungsdatum	Typ	Größe
13587_fsm	03.04.2023 15:08	Datei	24 KB
13587_vm	03.04.2023 15:08	Datei	8 KB
13590	03.04.2023 15:08	Datei	0 KB
13591	03.04.2023 15:08	Datei	8 KB
25176	03.04.2023 15:29	Datei	1.000 KB
25176_fsm	03.04.2023 15:29	Datei	24 KB
25176_vm	03.04.2023 15:29	Datei	8 KB
25180	03.04.2023 15:29	Datei	8 KB
25181	03.04.2023 15:29	Datei	7.880 KB
25181_fsm	03.04.2023 15:29	Datei	24 KB
25181_vm	03.04.2023 15:29	Datei	8 KB
25186	03.04.2023 15:29	Datei	8 KB
25187	03.04.2023 15:31	Datei	21.704 KB
25187_fsm	03.04.2023 15:31	Datei	24 KB
25187_vm	03.04.2023 15:31	Datei	8 KB
25191	03.04.2023 15:31	Datei	8 KB
25192	03.04.2023 15:31	Datei	16 KB
25192_fsm	03.04.2023 15:31	Datei	24 KB
25196	03.04.2023 15:31	Datei	8 KB
25197	03.04.2023 15:31	Datei	1.408 KB
25197_fsm	03.04.2023 15:31	Datei	24 KB
25197_vm	03.04.2023 15:31	Datei	8 KB
25200	03.04.2023 15:31	Datei	8 KB
25201	03.04.2023 15:33	Datei	38.320 KB
25201_fsm	03.04.2023 15:33	Datei	32 KB
25201_vm	03.04.2023 15:33	Datei	8 KB
25206	03.04.2023 15:33	Datei	8 KB
25207	03.04.2023 15:33	Datei	1.064 KB
25207_fsm	03.04.2023 15:33	Datei	24 KB
25207_vm	03.04.2023 15:33	Datei	8 KB
25212	03.04.2023 15:33	Datei	8 KB
25213	03.04.2023 15:33	Datei	1.912 KB
25213_fsm	03.04.2023 15:33	Datei	24 KB
25213_vm	03.04.2023 15:33	Datei	8 KB
25220	03.04.2023 15:33	Datei	8 KB
25221	03.04.2023 15:33	Datei	872 KB
25221_fsm	03.04.2023 15:33	Datei	24 KB
25221_vm	03.04.2023 15:33	Datei	8 KB
25227	03.04.2023 15:24	Datei	0 KB
25228	03.04.2023 15:24	Datei	8 KB
25229	03.04.2023 15:33	Datei	8 KB
25239	03.04.2023 15:25	Datei	448 KB
25241	03.04.2023 15:25	Datei	448 KB
25243	03.04.2023 15:25	Datei	1.344 KB
25245	03.04.2023 15:25	Datei	9.368 KB
25247	03.04.2023 15:25	Datei	9.368 KB
25249	03.04.2023 15:25	Datei	16 KB
25251	03.04.2023 15:25	Datei	728 KB
25253	03.04.2023 15:25	Datei	728 KB

### Definition Page

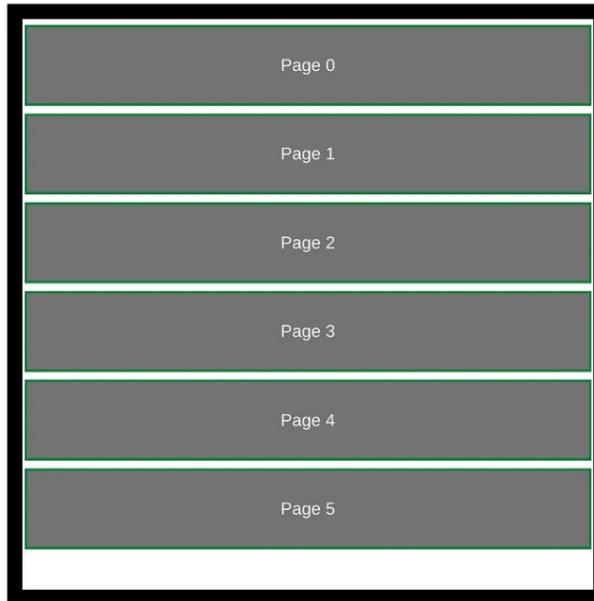
Eine Page ist die grundlegende Einheit der Datenorganisation in PostgreSQL. Die Größe einer Page ist standardmäßig 8 Kilobyte, kann aber je nach Systemkonfiguration variieren. Jede Page besteht aus mehreren Tupeln (Datensätzen) und zugehörigen Verwaltungsinformationen (Metadaten). Die Pages werden verwendet, um Daten effizient in der Datenbank zu speichern und abzurufen. Jede Tabelle in PostgreSQL besteht aus einer oder mehreren Pages, abhängig von der Größe der gespeicherten Daten.

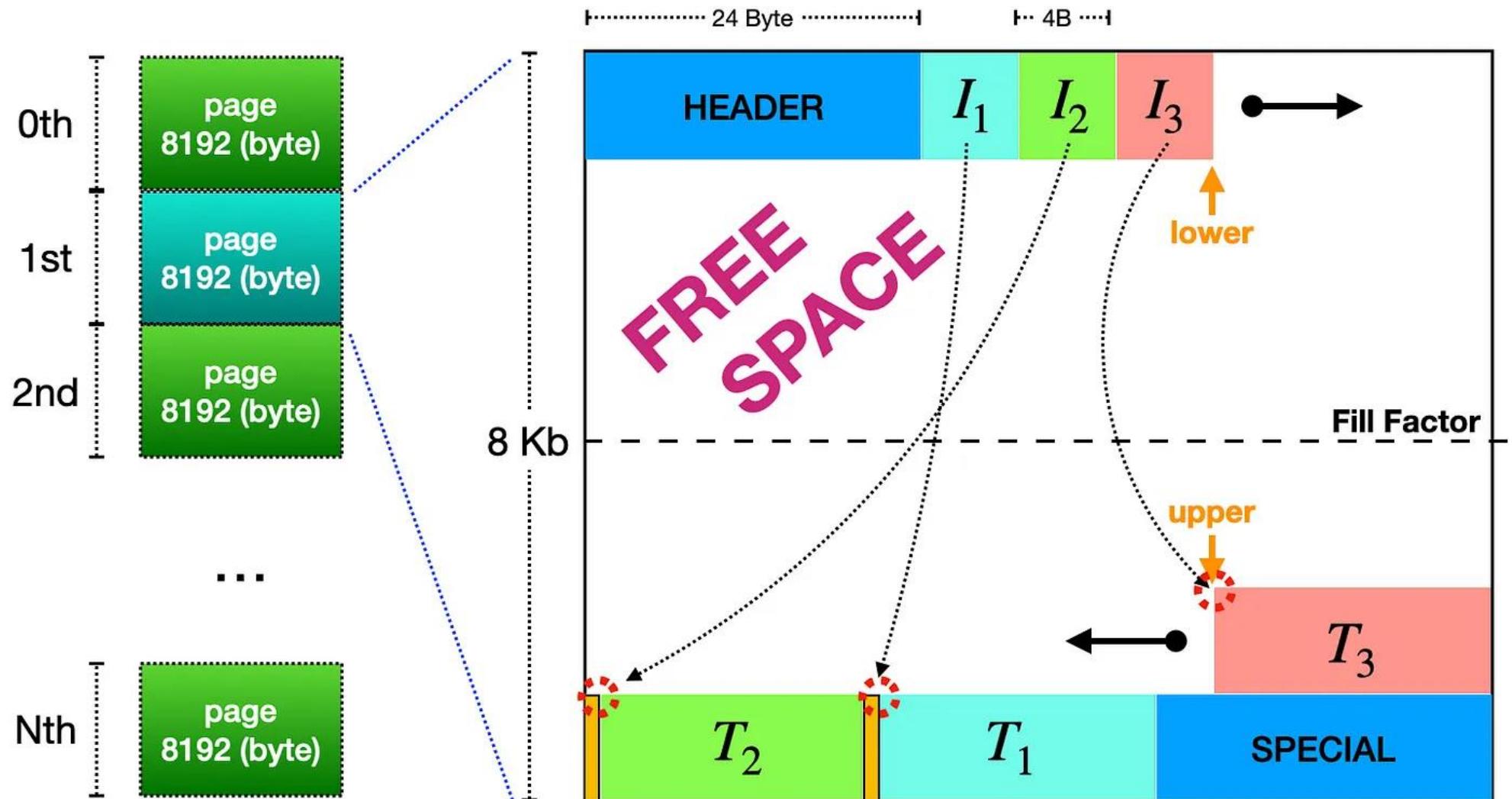




### Heap File

**Heap File:** Ein Heap File (auch Data File) ist eine physische Datei auf der Festplatte, die eine ungeordnete Sammlung von Pages enthält. In PostgreSQL wird der Hauptteil der Daten in Heap Files gespeichert. Ein Heap File gehört normalerweise zu einer bestimmten Tabelle, und jede Tabelle kann aus einem oder mehreren Heap Files bestehen. (Default 1GB).





## ctid

In PostgreSQL bezeichnet ctid eine virtuelle Systemspalte, die für "TID" (Tuple Identifier) steht. Der ctid ist ein interner Zeiger auf die physische Speicherposition eines Tupels (Datensatzes) innerhalb der Datenbank. Jedes Tupel in einer Tabelle hat einen eindeutigen ctid, der aus zwei Komponenten besteht:

**Blocknummer:** Dies ist der erste Teil des ctid und gibt die interne Nummer des physischen Heapfiles an, in dem sich das Tupel befindet.

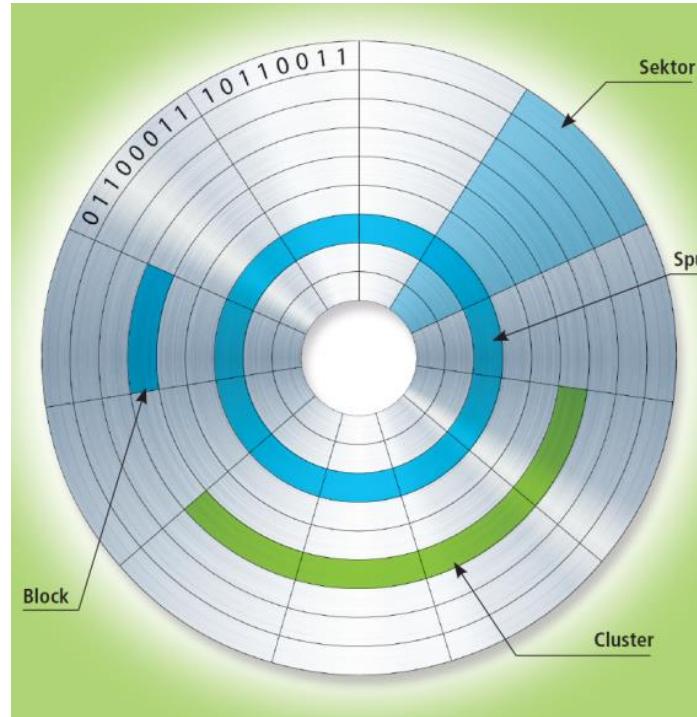
**Tupelindex innerhalb des Blocks:** Dies ist der zweite Teil des ctid und gibt die Position des Tupels innerhalb des Blocks an.

```
SELECT ctid, * FROM escooter;
```

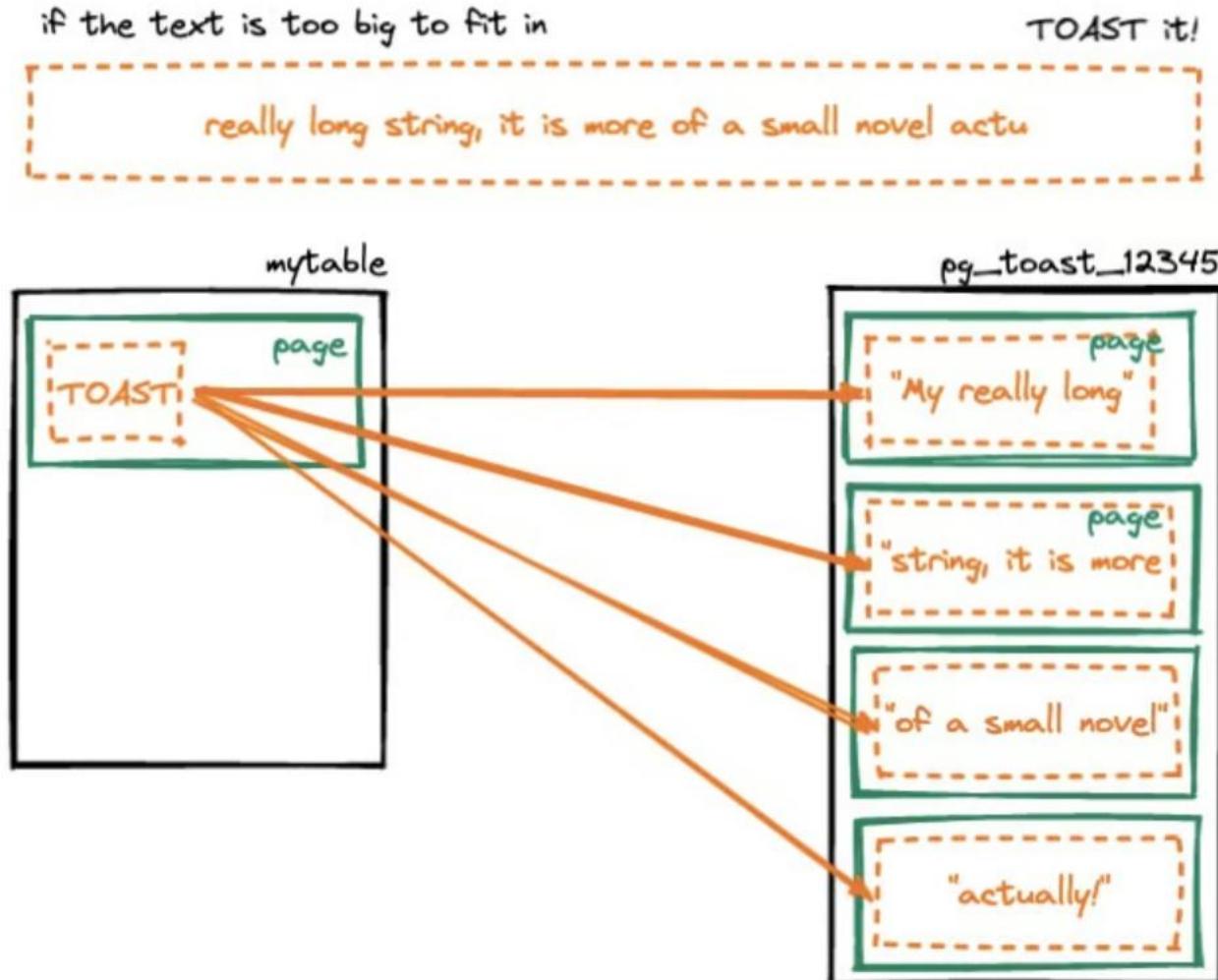
## Festplattenblock ist nicht das gleiche wie eine Page/Block in Postgres

Achtung – anderer Kontext andere Bedeutung!

Block: In PostgreSQL ist ein Block ein Synonym für eine Page. Im Kontext von Festplatten bezieht sich ein Block auf die kleinste adressierbare Speichereinheit. (HDDs default 512 bytes)



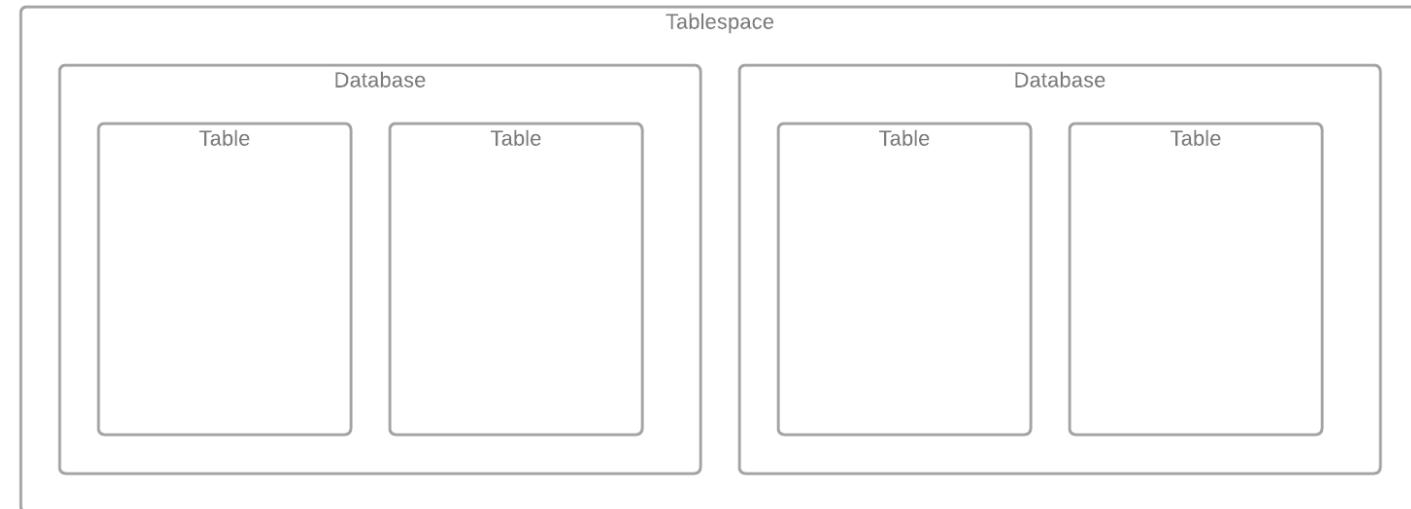
## The Oversized-Attribute Storage Technique (TOAST)



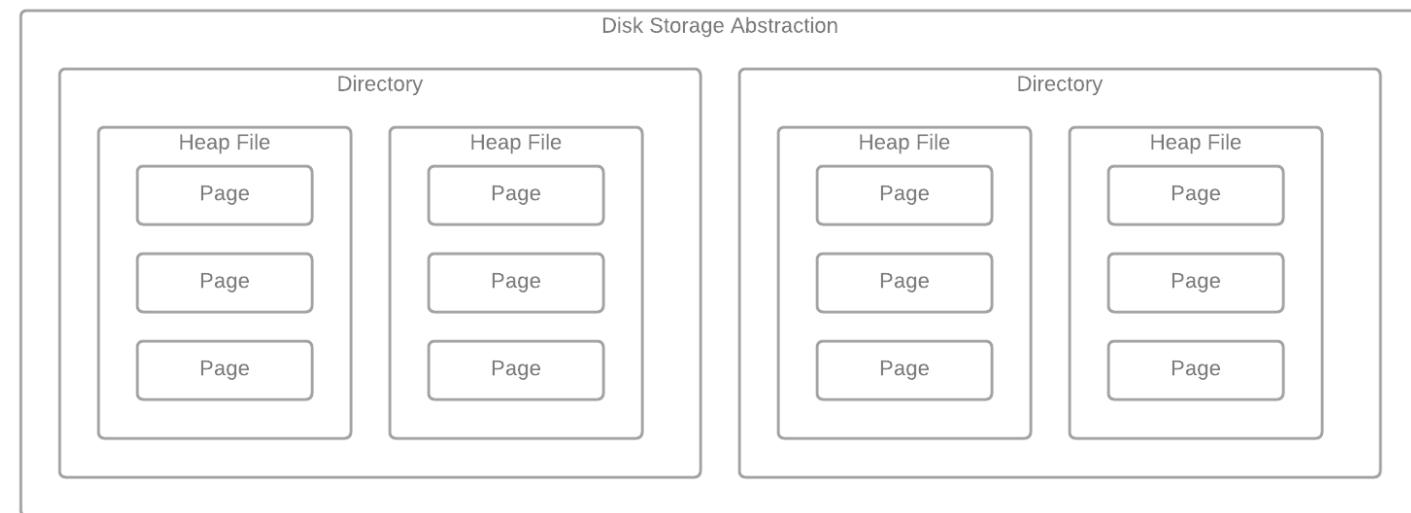
Ist die Größe einer Tabelle oder eines Index größer als 1 GByte, werden weitere Dateien ebenfalls mit einer Maximalgröße von 1 GByte angelegt. Die neuen Dateinamen haben eine Sequenznummer hinter dem Punkt.

```
$ ls -l 16501*
-rw----- 1 postgres postgres 1073741824 Apr  2 19:40 16501
-rw----- 1 postgres postgres   831381504 Apr  2 19:39 16501.1
-rw----- 1 postgres postgres      491520 Apr  2 19:39 16501_fsm
-rw----- 1 postgres postgres       8723 Apr  2 19:39 16501_vm
```

### Logical Layout



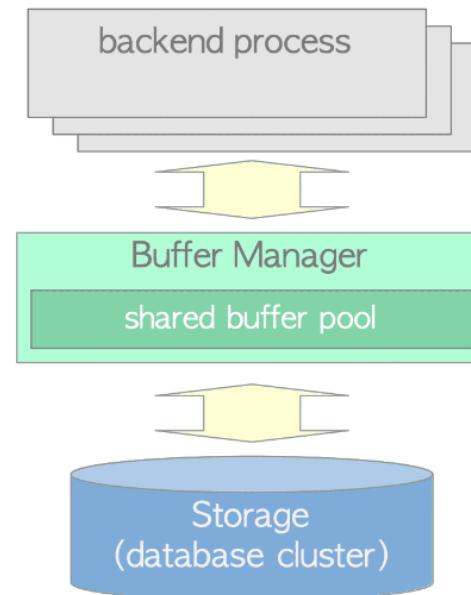
### Physical Layout



# Optimierung durch Puffer/Buffer

## Buffer Manager

Der Buffer Manager in Postgres ist ein System-Modul, das für das effiziente Laden, Speichern und Verwalten von Datenblöcken (Pages) im Arbeitsspeicher (Buffer) verantwortlich ist. Der Buffer Manager ermöglicht es, häufig verwendete Pages im Arbeitsspeicher zu halten, um den Zugriff auf Daten zu beschleunigen und die Anzahl der Zugriffe auf die langsameren Datenträger zu reduzieren.



- Das Datenbanksystem nutzt (in der Regel) einen „Buffer“/Puffer um Blöcke (Pages) im Hauptspeicher zu cachen
- In Postgres wird dieser als Shared Buffer bezeichnet
- Ziel: Anzahl der Festplattenzugriffe reduzieren (Disk I/O)
- Alle Speicherzugriffe der Ausführungseinheit gehen über den Buffer Manager
- Buffer Manager prüft, ob Page bereits im Speicher ist, wenn nicht wird Page in Shared Buffer geladen

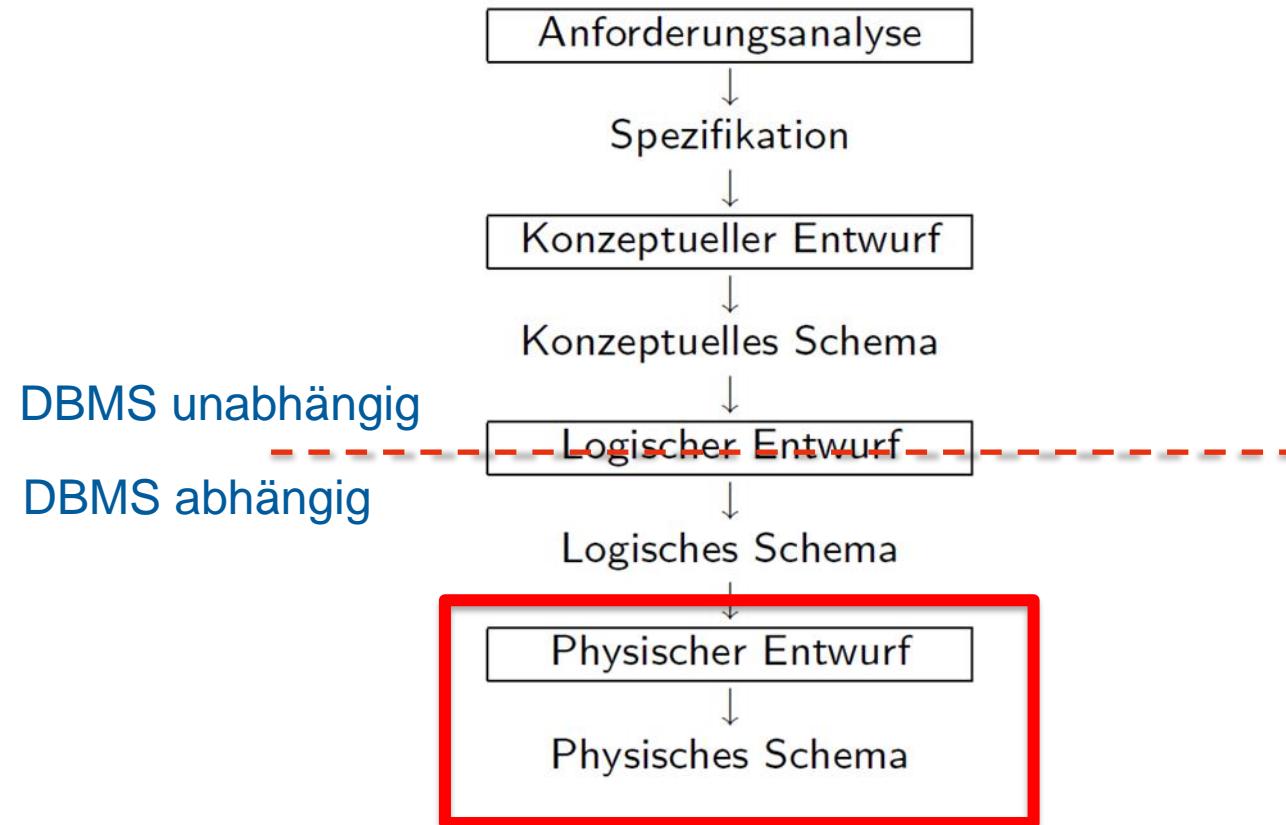


## Zugriffsoptimierung durch Indexe und Storage

Prof. Dr. Patrick Cato

Technische Hochschule Ingolstadt

# Einordnung



```
Table "cd.bookings"
 Column |          Type          | Collation | Nullable | Default
-----+-----+-----+-----+-----+
bookid | integer          |           | not null |
facid  | integer          |           | not null |
memid  | integer          |           | not null |
starttime | timestamp without time zone |           | not null |
slots   | integer          |           | not null |

Indexes:
"bookings_pk" PRIMARY KEY, btree (bookid)
"bookings_facid_memid" btree (facid, memid)
"bookings_facid_starttime" btree (facid, starttime)
"bookings_memid_facid" btree (memid, facid)
"bookings_memid_starttime" btree (memid, starttime)
"bookings_starttime" btree (starttime)

Foreign-key constraints:
"fk_bookings_facid" FOREIGN KEY (facid) REFERENCES cd.facilities(facid)
"fk_bookings_memid" FOREIGN KEY (memid) REFERENCES cd.members(memid)
```

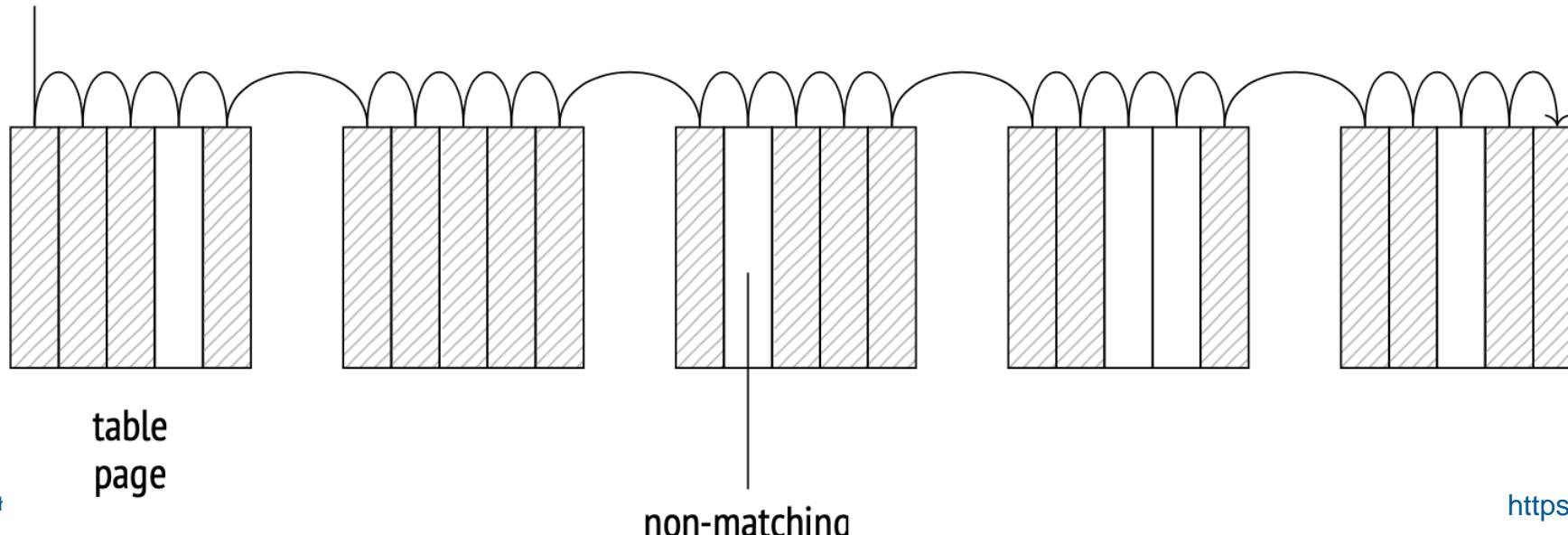
# Suchoptimierung durch B-Trees / B-Bäume

## Problemraum: Finde ein bestimmtes Buch



## Sequentieller Scan

Ein sequenzieller Scan ist ein grundlegender Algorithmus zur Suche nach einem bestimmten Element in einer Liste oder einem Array, indem er sequenziell von Anfang bis Ende durchläuft und jedes Element mit dem gesuchten Element vergleicht, bis es gefunden wird oder das Ende der Liste erreicht ist. Im Kontext von PostgreSQL wird das Heap File sequenziell gelesen, ohne dabei einen Index zu verwenden.



## Lösungsraum: Karteikarte

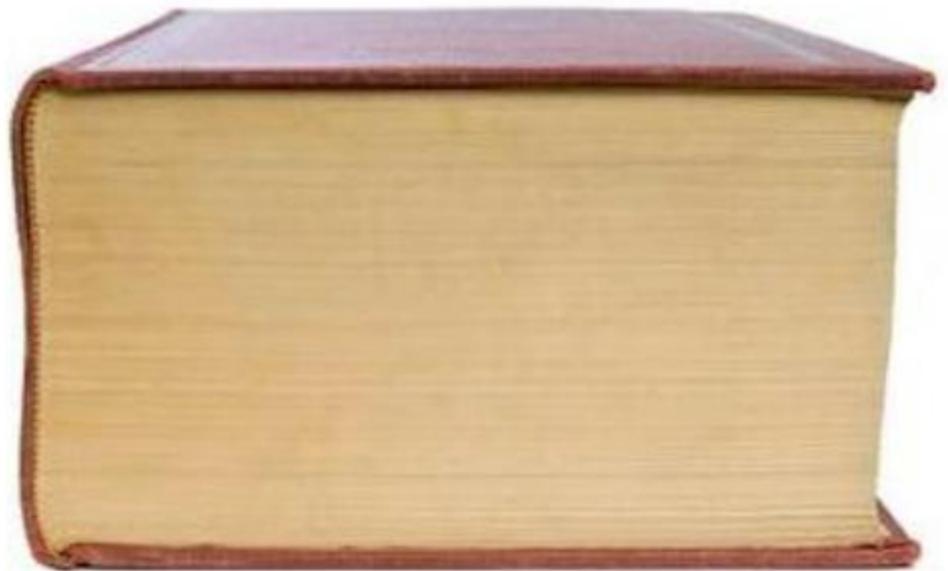




### Index Scan

In PostgreSQL ist der Index Scan eine Abfrage-Methode, die in der Regel den B-Tree-Index auf einer oder mehreren Spalten einer Tabelle nutzt, um schnell auf die gesuchten Daten zuzugreifen. Der Index Scan arbeitet durch das Durchlaufen des Indexbaums, um den Indexeintrag (oder die Indexeinträge) zu finden, der auf den abgefragten Wert oder den abgefragten Bereich verweist.

*Problemraum: Finde ein bestimmtes Kapitel/Abbildung!*



---

## Inhaltsübersicht

Kurzfassung.....	III
Abstract.....	IV
Inhaltsübersicht.....	V
Inhaltsverzeichnis .....	VI
Abbildungsverzeichnis .....	IX
Tabellenverzeichnis .....	XII
Abkürzungsverzeichnis .....	XIII
1 Einleitung .....	1
2 Forschungsdesign .....	21
3 Grundlagen und Stand der Forschung.....	27
4 Anforderungen der Datenverarbeitung.....	81
5 Anwendungsfälle für Big Data.....	125
6 Umsetzung in einem Big-Data-System .....	172
7 Zusammenfassung und Ausblick .....	220
Literaturverzeichnis .....	231
Anhang .....	248

## Problemraum: Suche richtigen Weg



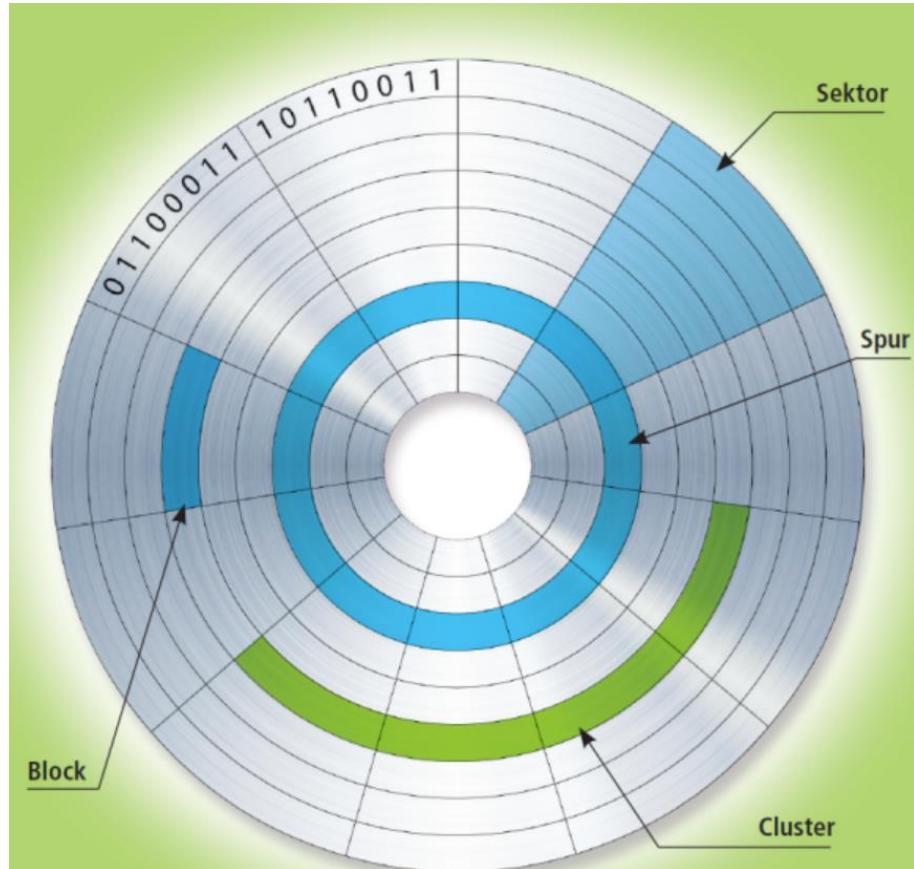
## Lösungsraum: Schilder



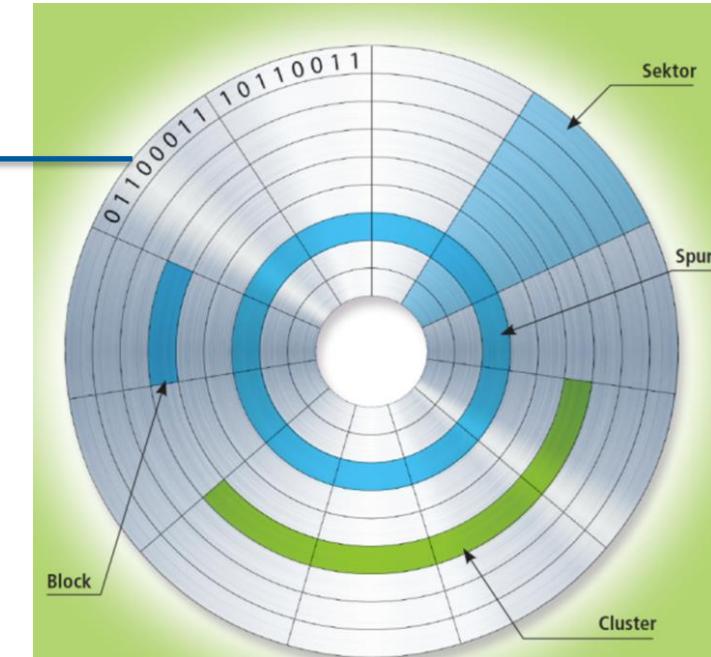
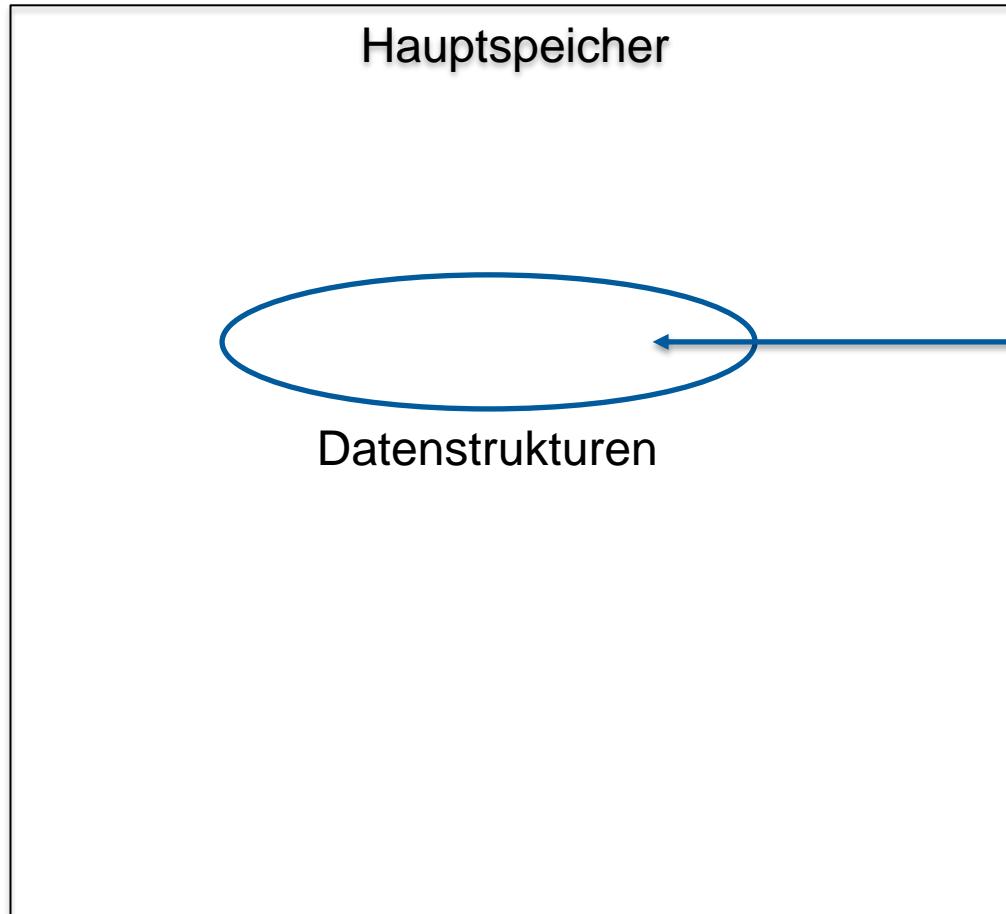


„Gebe mir einen bestimmten Datensatz möglichst schnell  
bzw. ressourceneffizient“

## Begrifflichkeiten: Festplatte (HDD)



Datenzugriff erfolgt über Hauptspeicher (RAM). Block von Festplatte muss in Hauptspeicher geladen werden



## Beispielrelation employees

<b>id</b>	<b>name</b>	<b>abteilung</b>	....
1	John	...	
2	Tom	...	
3	Anid	...	
4	Sarah	...	
5	Loredana	...	
6	Maike	...	
7	Anita	...	
8	Hartmut	...	
9	Siglinde	...	
....			
100	...	...	....

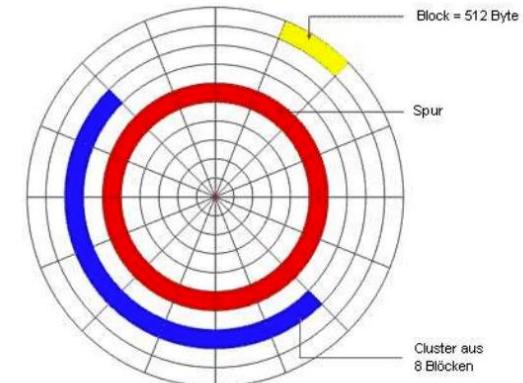
id	10
Name	50
Abteilung	10
spalte x	8
spalte y	50

---

Größe 128 Bytes

<b>id</b>	<b>name</b>	<b>abteilung</b>	...
1	John	...	
2	Tom	...	
3	Anid	...	
4	Sarah	...	
5	Loredana	...	
6	Maike	...	
7	Anita	...	
8	Hartmut	...	
9	Siglinde	...	
....			
100			

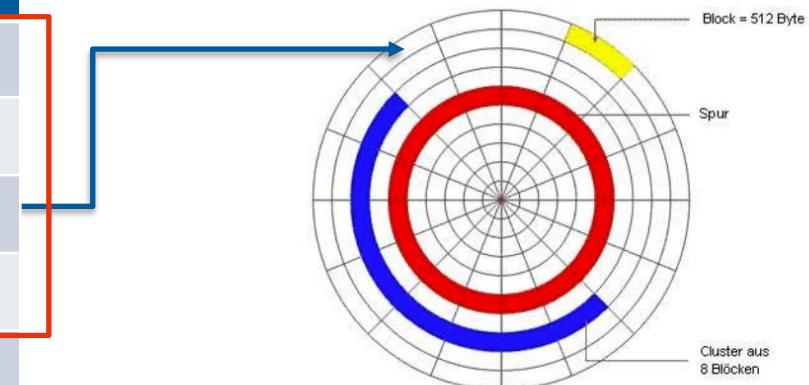
128 Bytes



Ein Block 512 Bytes

In einen Block passen 4 Zeilen  
(512/128)

<u>id</u>	name	abteilung	...
1	John	...	
2	Tom	...	
3	Anid	...	
4	Sarah	...	
5	Loredana	...	
6	Maike	...	
7	Anita	...	
8	Hartmut	...	
9	Siglinde	...	
....			
100			



$$\text{Anzahl Blöcke} = \frac{100}{4} = 25$$

Erläuterung: Pro Block können 4 Rows gespeichert werden.

Problem: Wenn ich einen bestimmten Datensatz suche (select \* from employees where id= „5“) müssen im worst case alle 25 Blöcke durchsucht werden.

## Optimierung: Mit Index als Pointer auf die Blockadresse

<u>id</u>	Record Pointer
1	Blockadresse 1
2	Blockadresse 1
3	Blockadresse 1
4	Blockadresse 1
5	Blockadresse 2
6	Blockadresse 2
7	Blockadresse 2
8	Blockadresse 2
9	Blockadresse 3
....	
100	Blockadresse 25

Der Index wird ebenfalls auf Platte persistiert

Speicherbetrag für einen Indexeintrag = (id + Recordpointer) = ( 10 Bytes + 6 Bytes) = 16 Bytes

Anzahl der Indexeinträge pro Block:  $\frac{512 \text{ Bytes}}{16 \text{ Bytes}} = 32$

Anzahl der Blöcke für Index =  $\frac{16 * 100}{512} = 3,125 \rightarrow 4$  Blöcke für den Index

→ Statt 25 Blöcke für einen Datensatz zu durchsuchen, sind jetzt nur noch 5 Blöcke notwendig ( 4 für Index + Datenblock)



Wenn statt 100 Einträge 1000 Einträge existieren: Statt 4 Blöcke für Indexsuche werden 32 Blöcke benötigt:

$$\frac{1000 * 16 \text{ Bytes}}{512 \text{ Bytes}} = 31,25$$

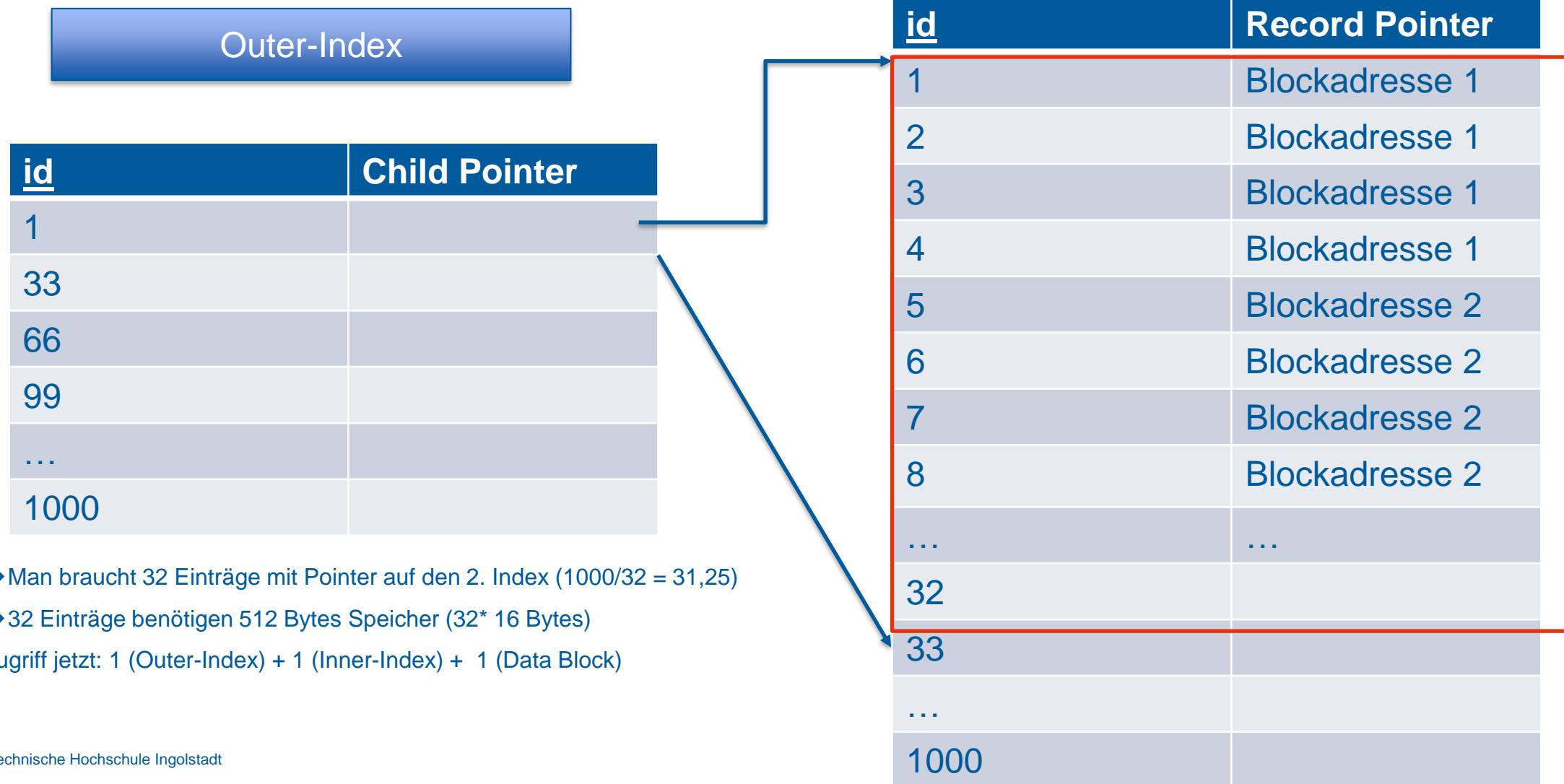
→ Weitere Optimierung durch Multi-Level Indizes (Ein Index vom Index)

# „Index für den Index“ für 1000 Einträge

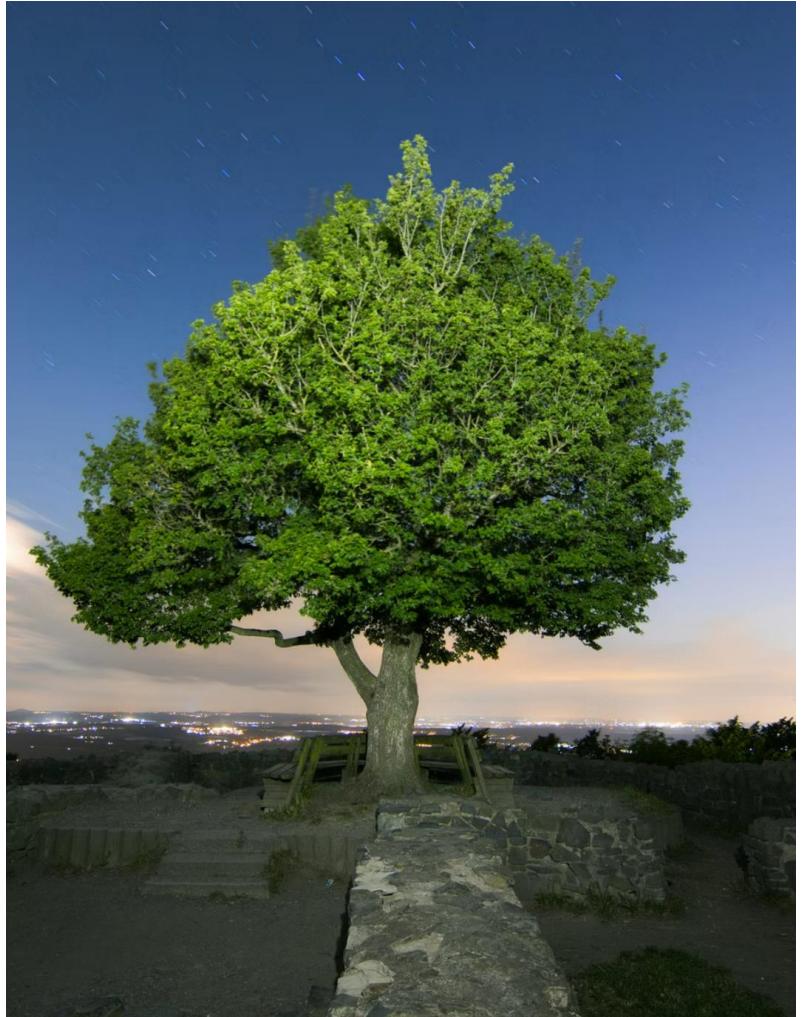


Inner-Index

1 Block für Index auf Disk



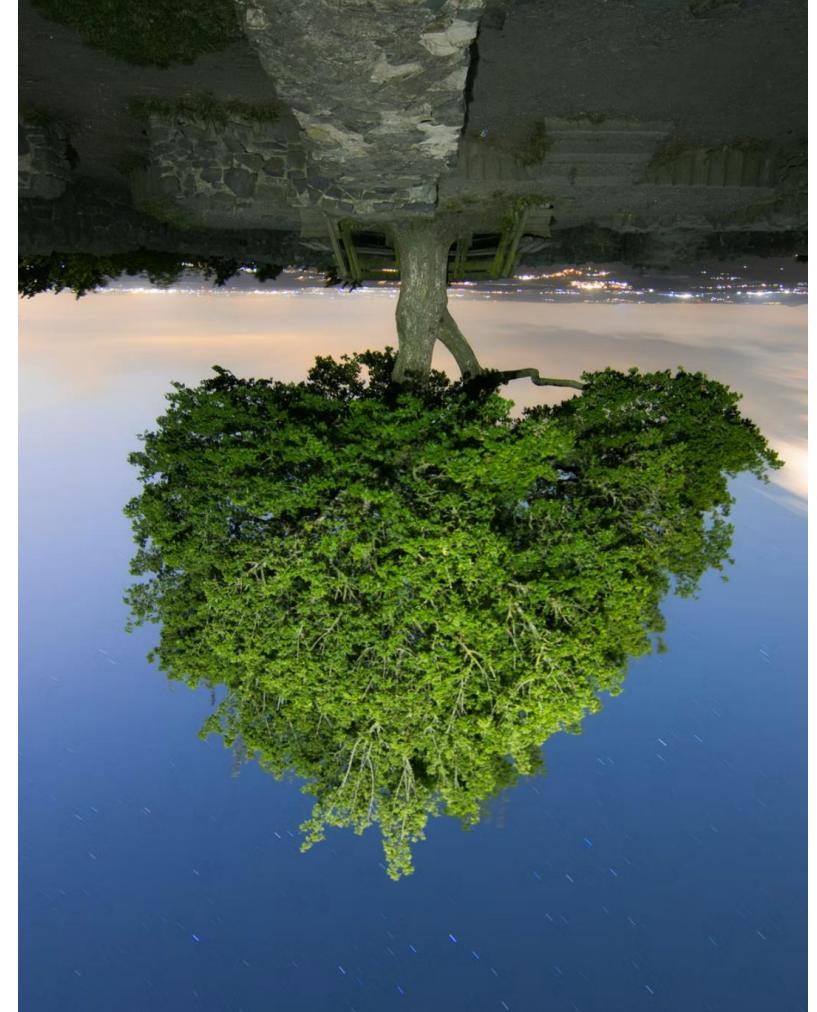
# Baum / Datenstruktur „Tree“

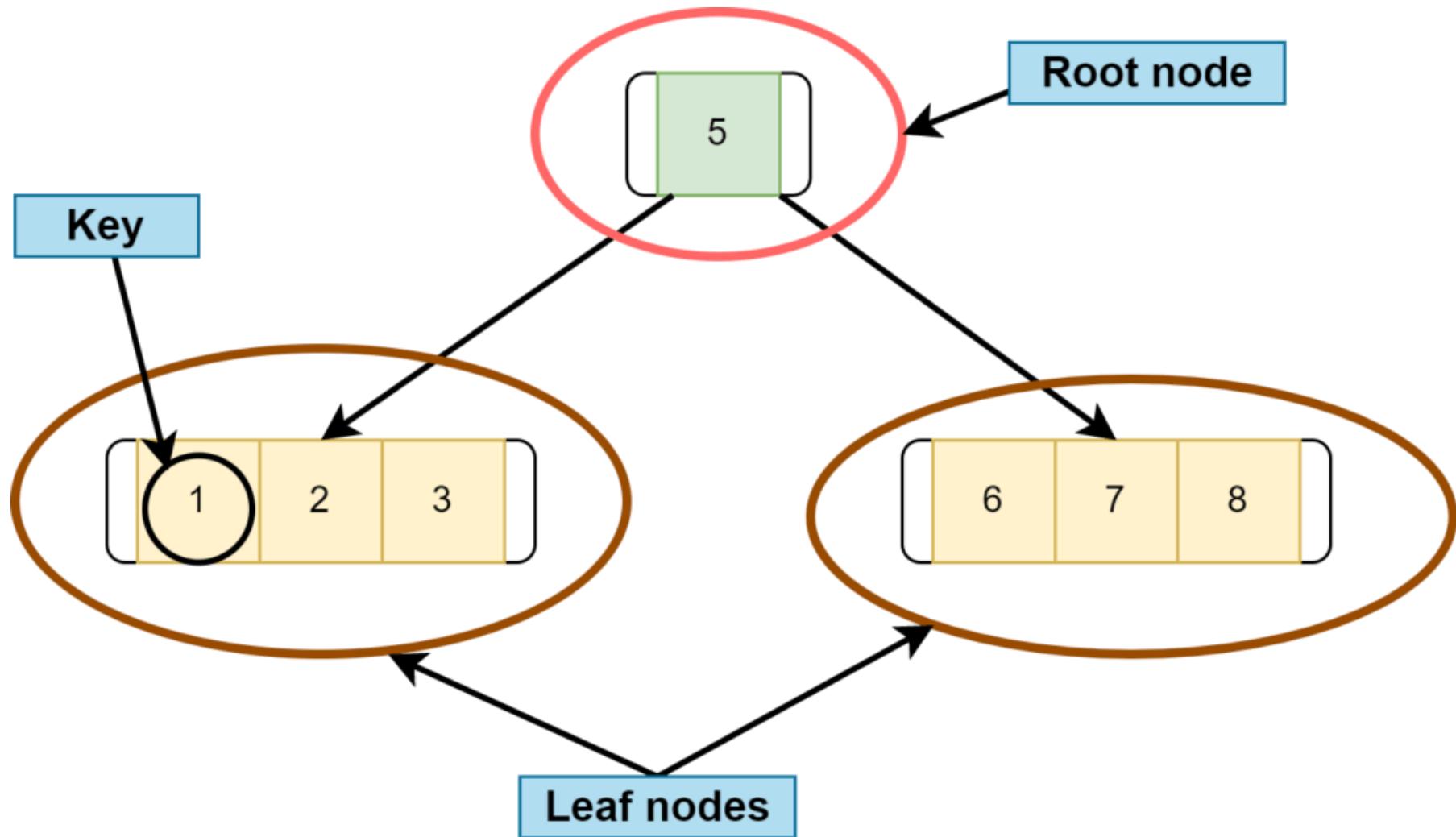


Root (Node)

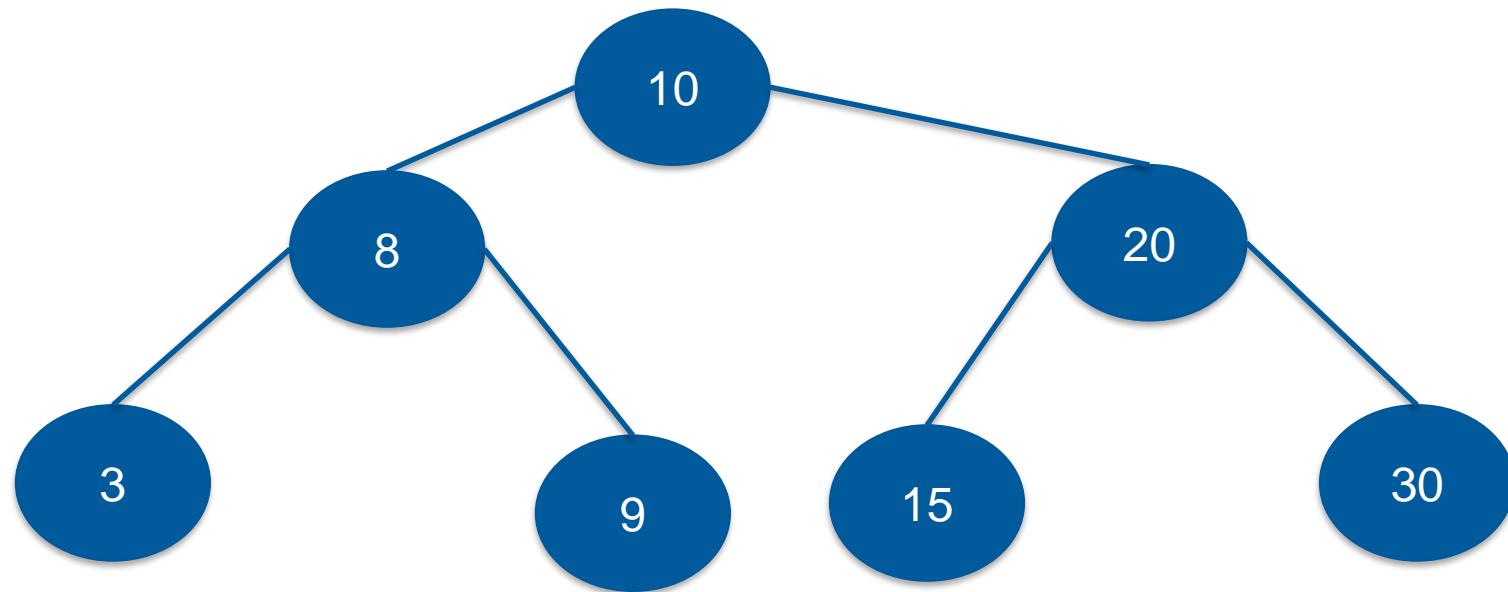
Internal Node

Leaf Node





Suchbäume: *Binary Search Tree (BST)*, um ein Key Value Pair zu finden



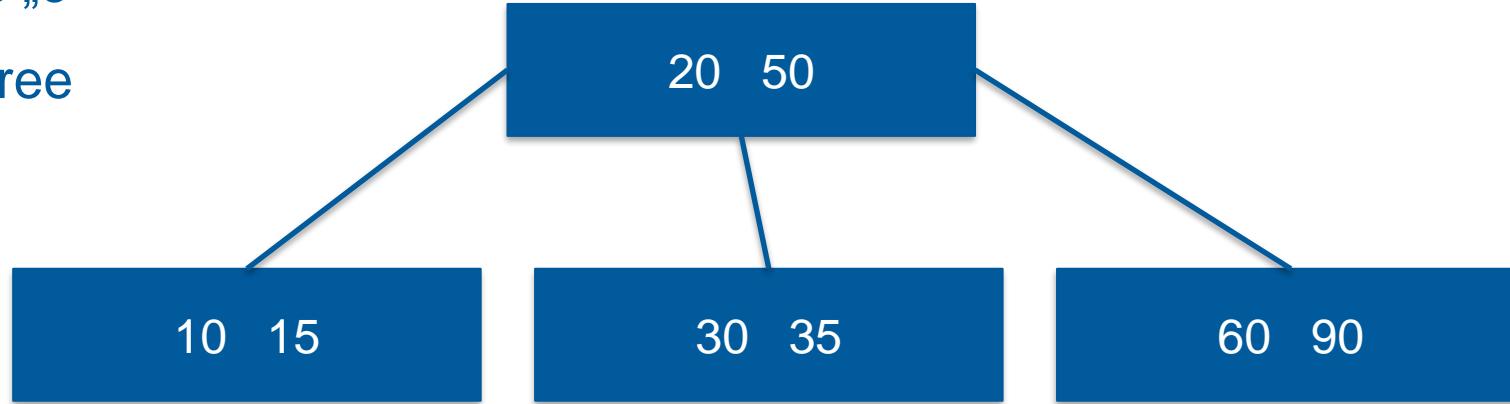
## B-Bäume sind spezialisierte M-Way Search Trees

M-Way heißt: Jeder Knoten kann  $m$  Kinder enthalten und  $m-1$  Keys

Ordnung/Degree „3“

3-Way Search Tree

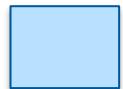
$K_1 < k_2 < \dots$



2 keys  
3 children  
Daher 3-way search trees

# Ableitung der Datenstruktur für Indexierung

## 4-Way Search Tree



Child Pointer (CP) auf den nächsten Knoten

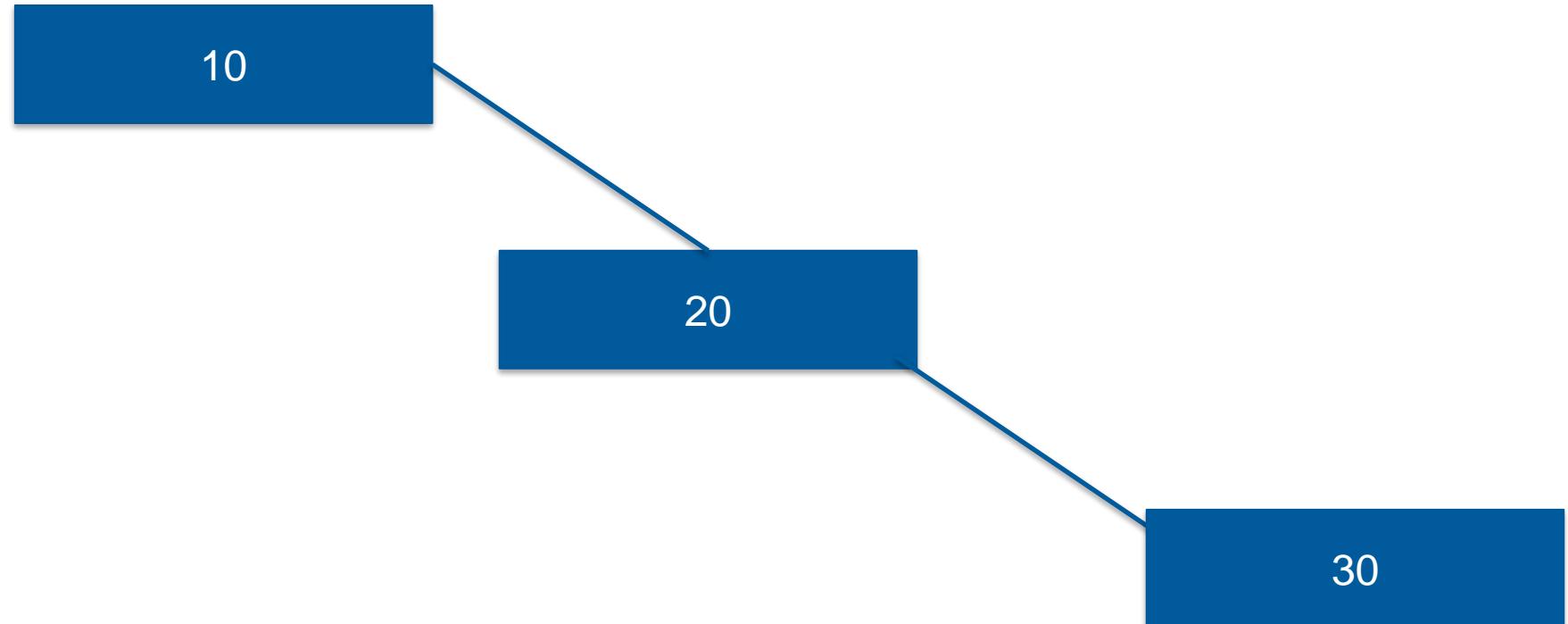


Record Pointer (RP) auf den Datensatz (Blockadresse)

Bei M-Way Search trees kann es zu sogenannten „unbalanced search trees“ (unausgeglichenen Suchbäumen) kommen.

Keys: 10, 20 ,30 (Degree 10)

**Kernproblem:**  
**Erstellungsprozess ist  
nicht hart definiert  
(keine Guidelines)**

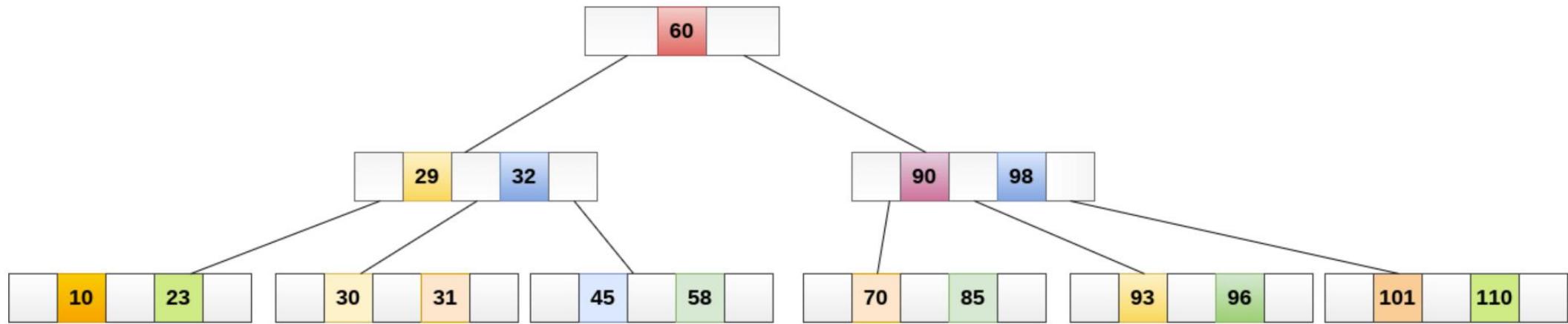


## Definition in Anlehnung an Knuth (1998) / Guidelines

B-Trees sind sozusagen M-Way Search Trees mit Regeln / Guidelines (“balanced”)

1. If the root node is a leaf node (only node in the tree), then it will have no children and will have at least one key. If the root node is a non-leaf node, then it will have at least 2 children and at least one key.
2. The keys of each node of a B tree should be stored in the ascending order
3. Every internal node except the root has at least  $\lceil m/2 \rceil$  children (refer to Point 1)
4. Every node has at most  $m$  children
5. All leaf at same level
6. A non-leaf node with  $k$  children contains  $k-1$  keys
7. Creation process is Bottom-up (Anmerkung: Beim Split geht die mittlere Zahl nach oben)

## Beispiel B-Tree



<https://www.cs.usfca.edu/~galles/visualization/BTree.html>

## Aufgabe



Erstellen Sie einen B-Baum der Ordnung  $m=3$ . Fügen Sie nacheinander in der vorgegebenen Reihenfolge die folgenden Werte in den untenstehenden Baum ein:  
14, 19, 15, 18, 16. Zeichnen Sie den Baum nach jedem Split-Vorgang (Endergebnis).

Ausgangssituation:

11	22
----	----



19, 15, 18, 16



15, 18, 16



18, 16





# Index / Secondary Index

## Definition Datenbankindex und Secondary Index

Ein Index (auch Datenbankindex) ist eine redundante Datenstruktur, die die Suche und das Sortieren nach Datensätzen beschleunigt. Der Primary Key wird in der Regel von jedem Datenbanksystem automatisch indexiert. Bei PostgreSQL wird standardmäßig ein Index auf den Primary Key angelegt und hierfür ein B-Tree erstellt.

Den automatisch angelegten Index auf den Primary Key bezeichnet man auch als Primary Index. Auf dem Datenbanksystem können weitere Indizes, sogenannte sekundäre Indizes, angelegt werden, wenn die Performance nicht ausreichend ist.

Achtung: Ein sekundärer Index bzw. der B-Tree muss bei Insert, Deletes, Updates ebenfalls angepasst werden. Daher sollten nicht zu viele sekundäre Indizes pro Tabelle/Collection definiert werden. Manche Datenbanken (z. B. Amazon DynamoDB) begrenzen die Anzahl der weiteren Indizes (z. B. 5)



## Secondary Index in PostgreSQL

```
CREATE INDEX index_name
ON table_name (column1, column2, ...);
```

```
CREATE INDEX idx_lastname
ON Persons (LastName);
```

```
CREATE INDEX idx_pname
ON Persons (LastName, FirstName);
```

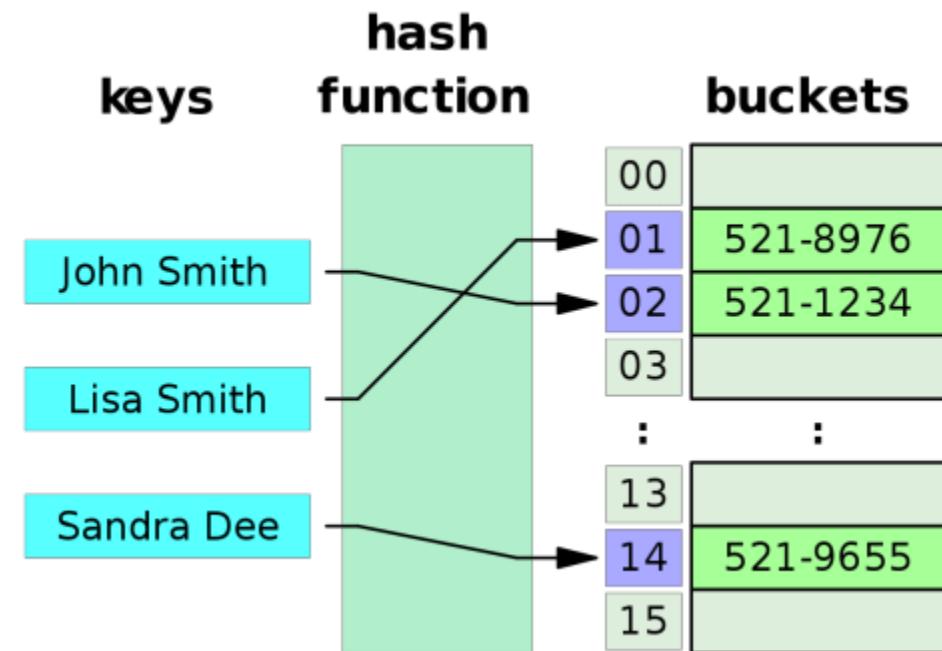
# Weitere Indextypen in Postgres

# Hash Index

Hash-Indizes werden normalerweise für Abfragen verwendet, die nach bestimmten Werten suchen, da sie sehr schnell sind, wenn es darum geht, bestimmte Datensätze zu finden. Wenn eine Tabelle eine hohe Anzahl von eindeutigen Schlüsselwerten enthält, kann die Verwendung eines Hash-Index die Abfrageleistung erheblich verbessern.

## Optimierung in der WHERE clauses

Nicht geeignet für Bereichssuchen!



```
CREATE TABLE employees (
    id SERIAL PRIMARY KEY,
    name TEXT NOT NULL,
    salary NUMERIC(10, 2) NOT NULL
);
```

```
CREATE INDEX idx_users_id ON users USING HASH (id);
```

B-Tree

Multifunktionsindex. In über 98% der Use Cases ausreichend

Hash

Optimierung von einfachen Vergleichen in Where Clause

GiST

Geometry, full text search

SP-GiST

Clustered Data, z. B. Datumsfelder. Viele gleiche Felder haben das gleiche Jahr

GIN

Falls ein Feld JSON Daten enthält

BRIN

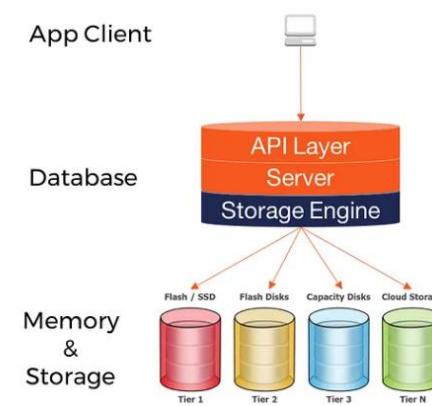
Spezialisierter Index für sehr große Datenmenge

# Storage Engine

## Storage Engine

In einem Datenbankmanagementsystem (DBMS) bezieht sich der Begriff "Storage Engine" auf die Komponente, die für die Speicherung, das Lesen und das Schreiben von Daten auf die Festplatte verantwortlich ist.

Eine Storage Engine ist im Wesentlichen eine Software-Schicht, die zwischen der Datenbank-Engine und der Festplatte liegt. Sie ist für die Umsetzung von Daten in physikalische Speicherstrukturen, wie Dateien oder Tabellen, verantwortlich. Wenn eine Anwendung Daten in die Datenbank einfügt oder abruft, interagiert sie mit der Storage Engine, um auf die erforderlichen Daten zuzugreifen.



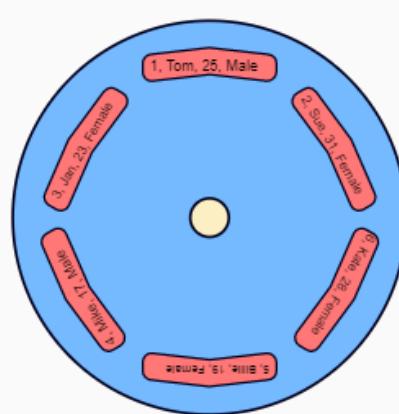
# Vor- und Nachteile von zeilenbasierter Abspeicherung

## Row-Oriented Storage (zeilenbasierte Speicherung)

Data

```
1, Tom, 25, Male  
2, Sue, 31, Female  
3, Jan, 23, Female  
4, Mike, 17, Male  
5, Billie, 19, Female  
6, Kate, 28, Female
```

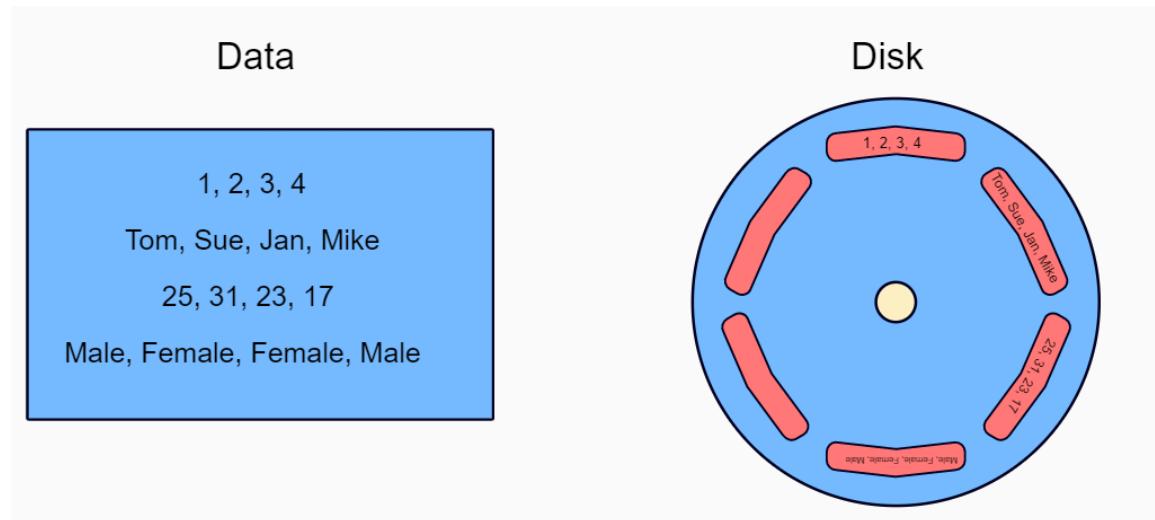
Disk



Das logische Modell wird eins zu eins auf das physische Modell abgebildet und der Datensatz zeilenweise auf Disk/SSD abgelegt (horizontale Partitionierung).

- + Für OLTP (Online Transaction Processing) sehr gut geeignet, weil der Zugriff in der Regel auf wenige bzw. einzelne Datensätze erfolgt
- Nicht geeignet für analytische Workloads (OLAP), weil sehr viele unwichtige Daten eingelesen werden müssen (Disk I/O ist eine sehr teurere Operation)

## Column-Oriented Storage (spaltenbasierte Speicherung)

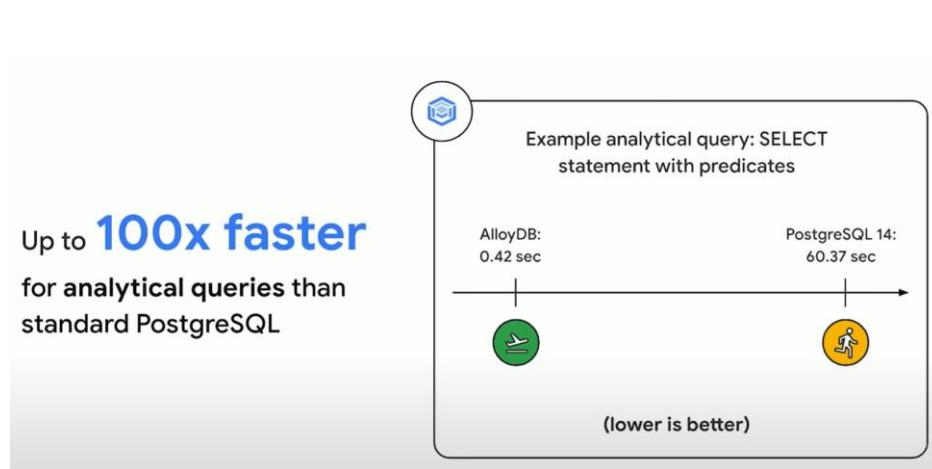


Die Daten werden spaltenweise auf Platte gespeichert

- + Sehr performant wenn nur auf einzelne Spalten zugegriffen werden muss (z. B. bei Aggregationen)
- Nicht geeignet, wenn man auf einzelne Datensätze zugreifen muss da hier dann mehrere Blöcke eingelesen werden müssen

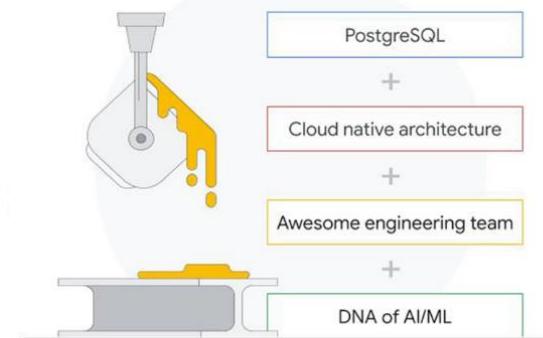
# AlloyDB for PostgreSQL under the hood: Columnar engine

Up to **100x faster**  
for analytical queries than  
standard PostgreSQL



AlloyDB for PostgreSQL

**PostgreSQL in a way that  
only Google can deliver it**



Google Cloud

Sheshadri Ranganath  
Engineering Director, AlloyDB  
for PostgreSQL

May 26, 2022

Recently, at Google I/O, we [announced AlloyDB for PostgreSQL](#), a fully-managed, PostgreSQL-compatible database for demanding, enterprise-grade transactional and analytical workloads. Imagine PostgreSQL plus the best of the cloud: elastic storage and compute, intelligent caching, and AI/ML-powered management. Further, AlloyDB

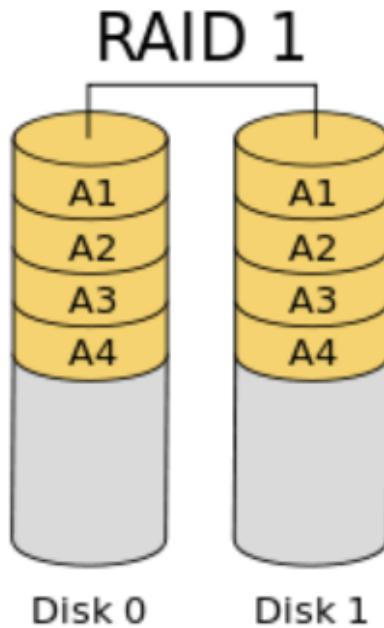
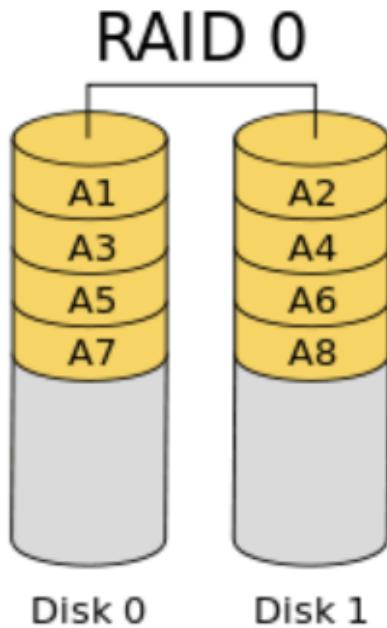
# RAID (Ausfallsicherheit Festplattenebene)

## RAID - Redundant Array of Independent Disks

- Zur Erhöhung der Reliabilität und Performance werden auf Hardwareseite Festplatten in Reihe geschaltet
- Es gibt Hardware RAID (RAID-Controller notwendig) oder Software RAID
- Mehrere physische Disks werden zu einem logischen Verbund Zusammengefasst
- RAID hilft nur beim Ausfall einer Festplatte (nicht bei RAID 0). Wenn gleichzeitig eine zweite Festplatte ausfällt, dann hilft RAID nicht gegen Datenverlust (außer RAID 6)



## RAID 0 vs. RAID 1



RAID0: + Schnelle Performance weil die Lese/Schreiboperation auf zwei Disks verteilt werden

- Keine Ausfallsicherheit

RAID1: + Ausfallsicherheit

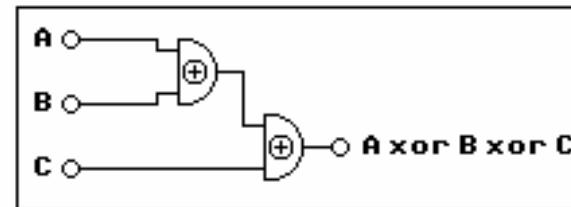
- Schreib/Leseperformance hängt von der langsamsten Disk ab
- kein Performancezugewinn

## Konzept der Parität – Prüfbit / Paritybit

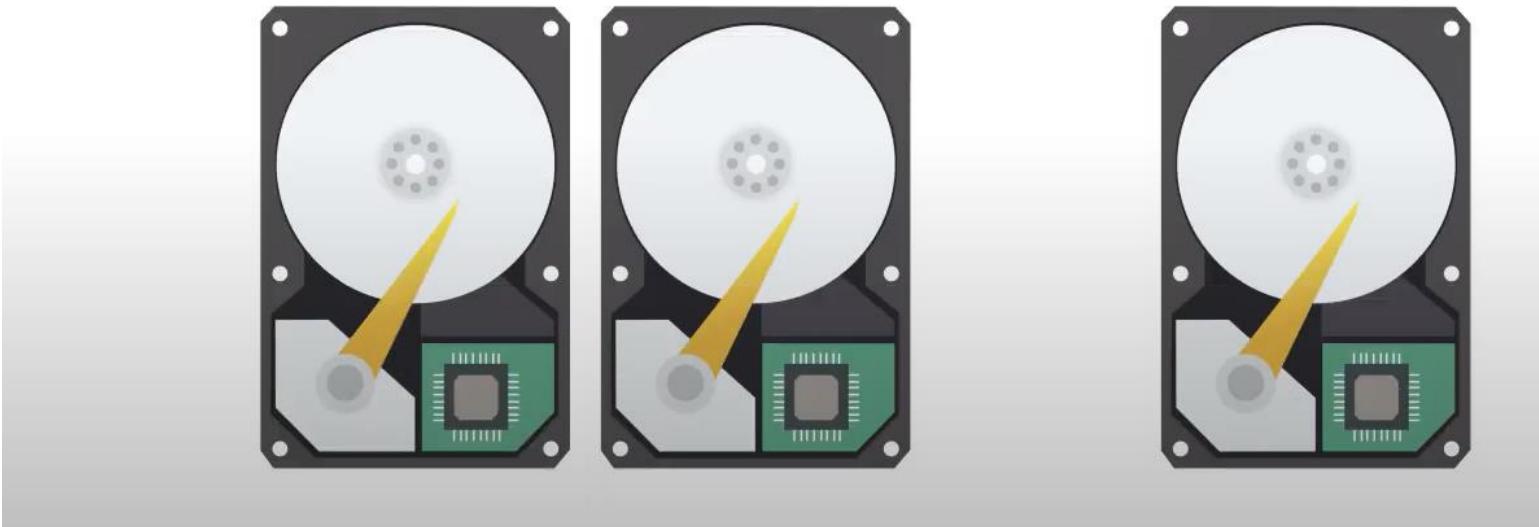
A	B	A <b>XOR</b> B
0	0	0
0	1	1
1	0	1
1	1	0

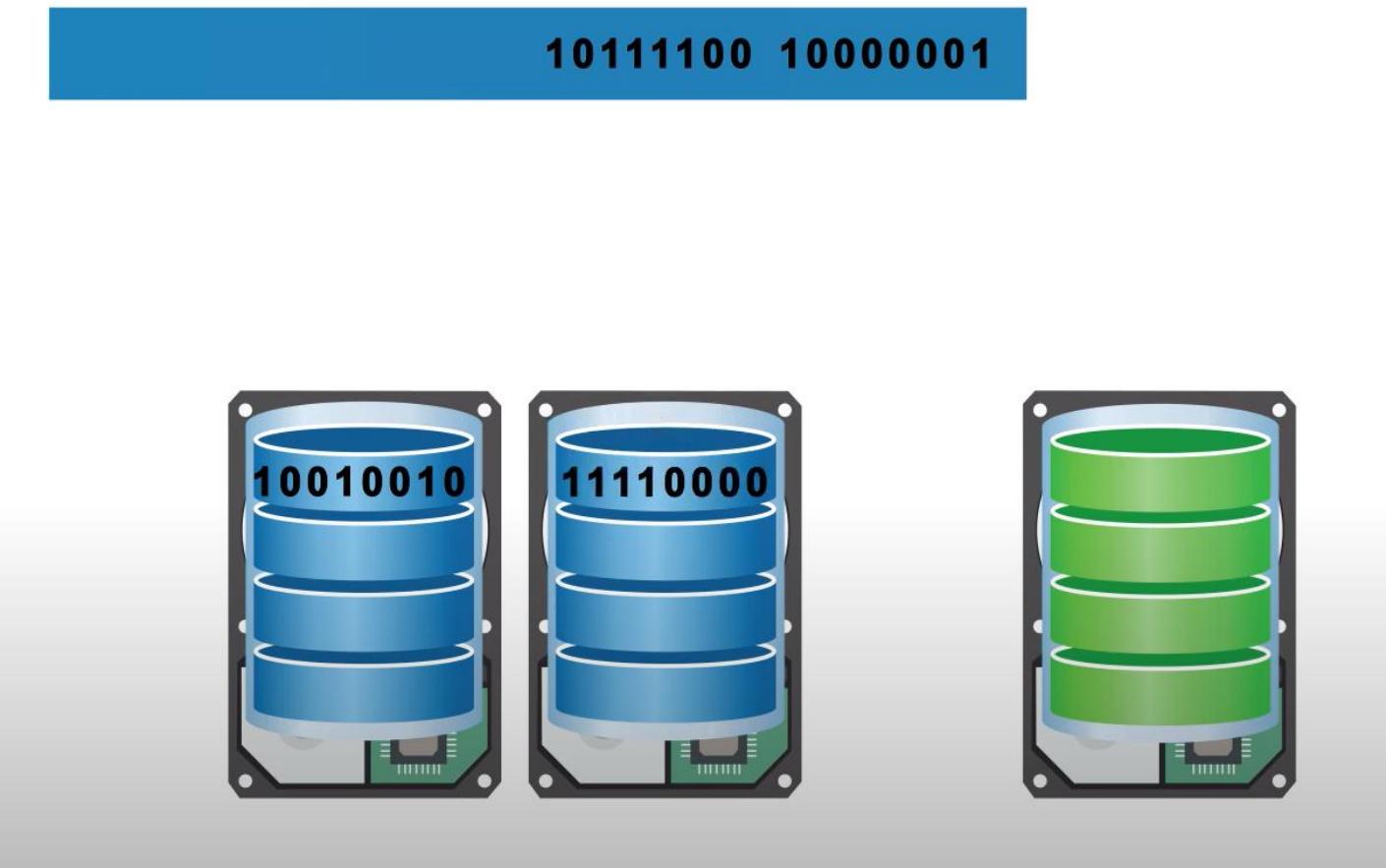
## 3 Eingänge XOR Gatter

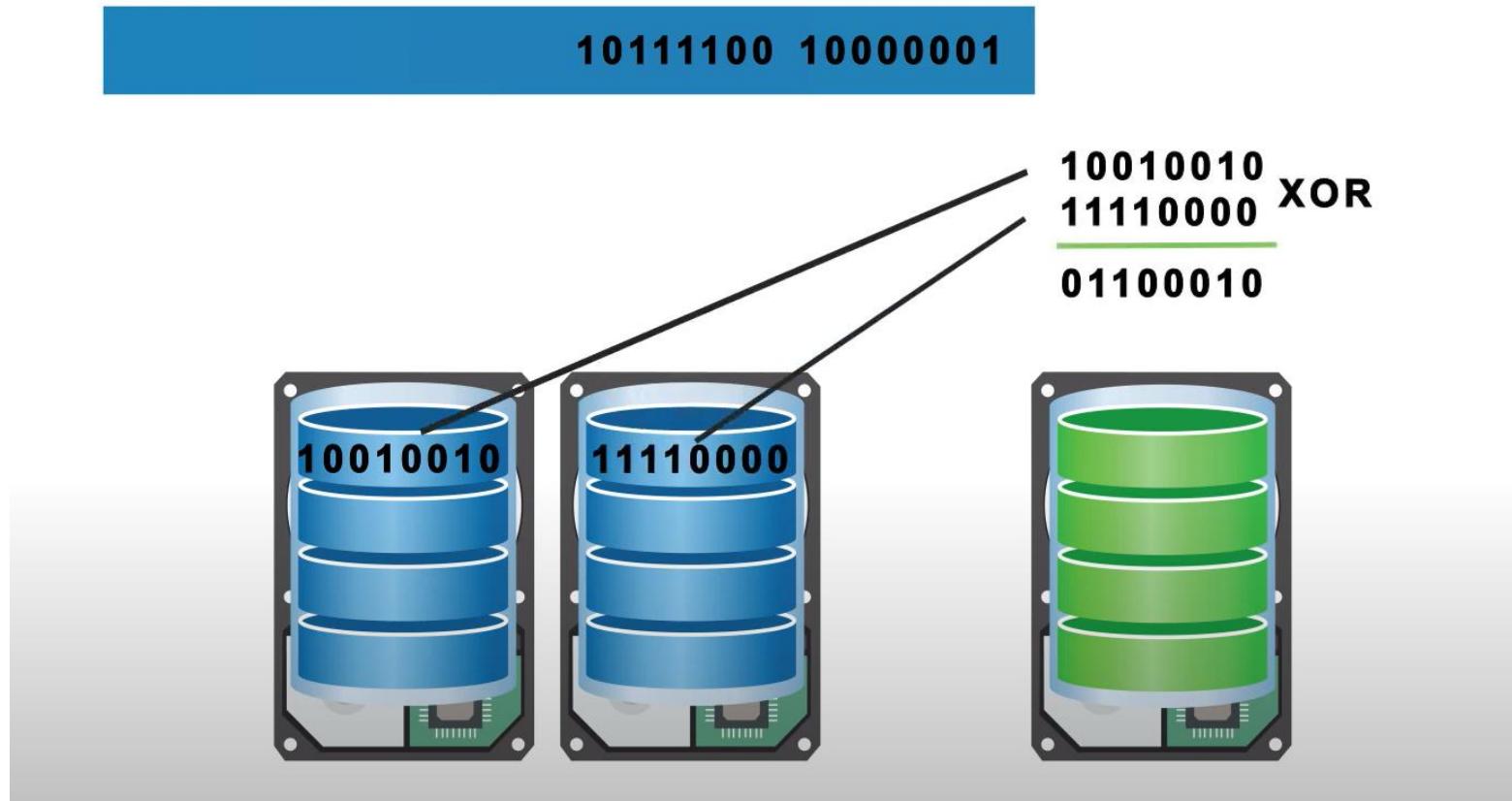
A	B	C	A xor B xor C
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1



**10010010 11110000 10111100 10000001**





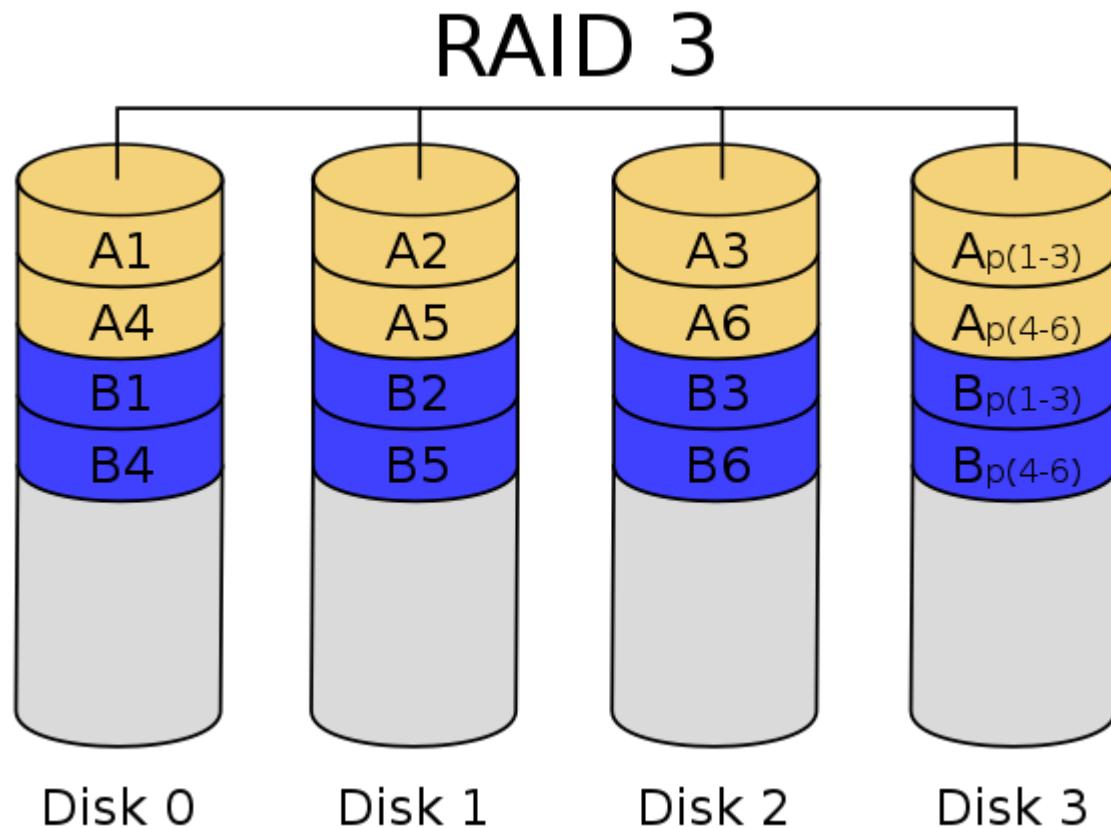


## Konzept der Parität: Wie lauten die Paritätsbits?

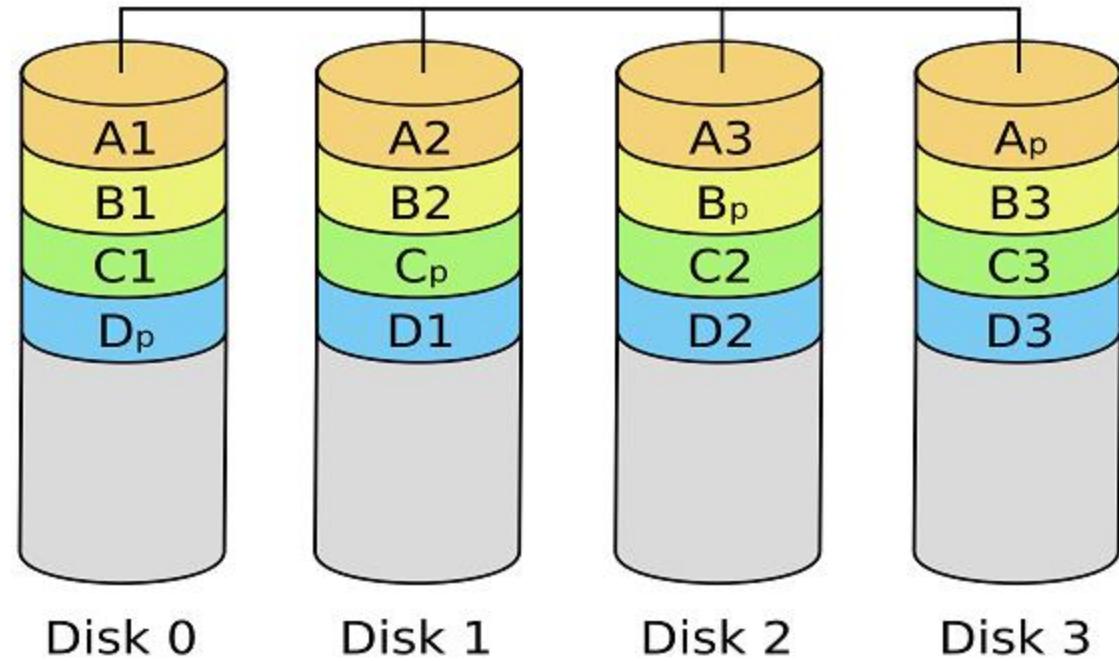
**10111100 10000001**



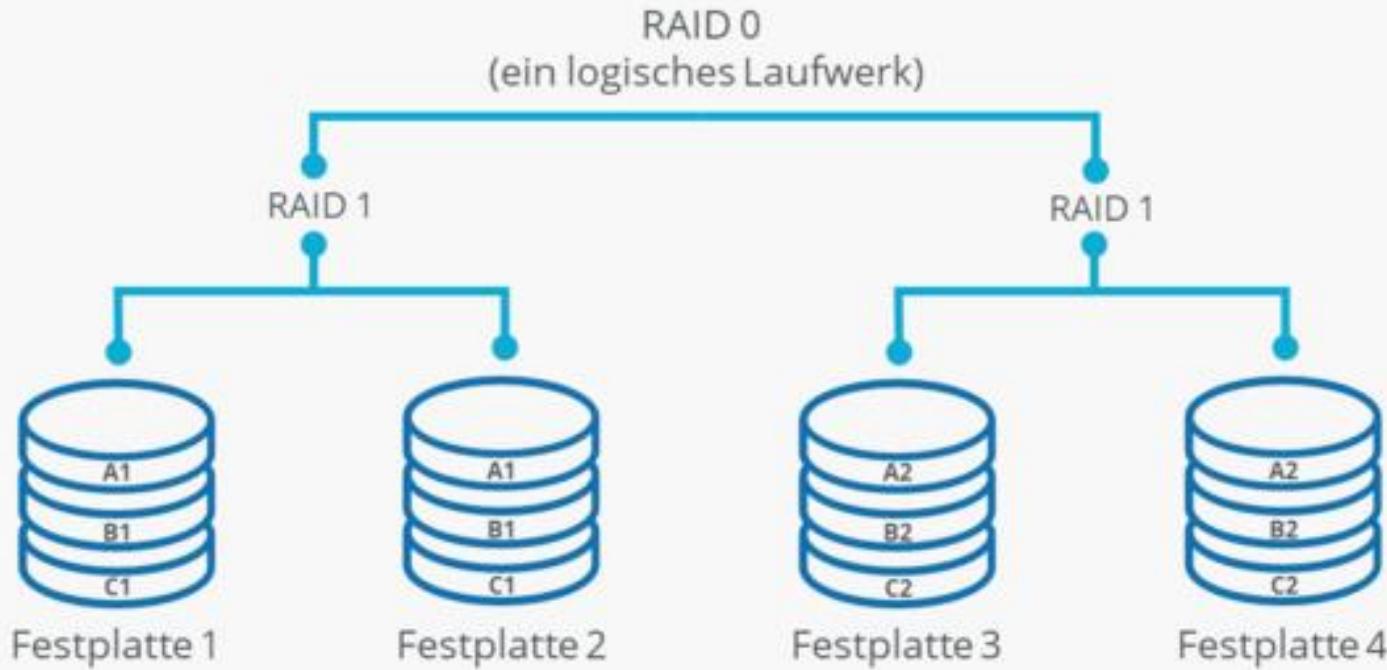




## RAID 5



## RAID 10 (Redundant Array of Independent Disks Level 10)



**IONOS**