Big Data Course

Chap 6-
Database Types

Electrical Engineering department of

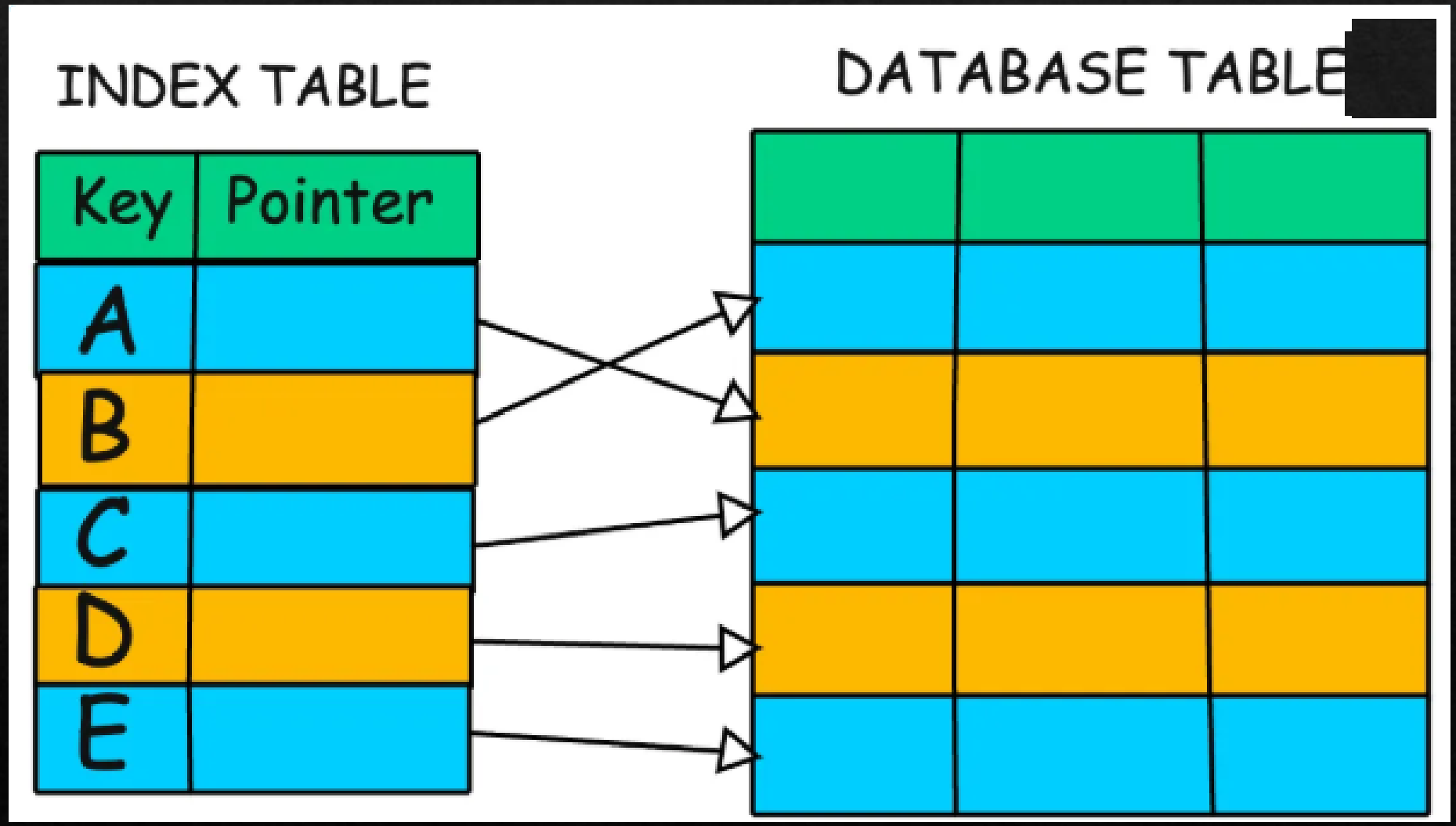
Amirkabir University of technology

Dr. Mohammadreza Pourfard

August 2025

Index

What are Database Indexes?

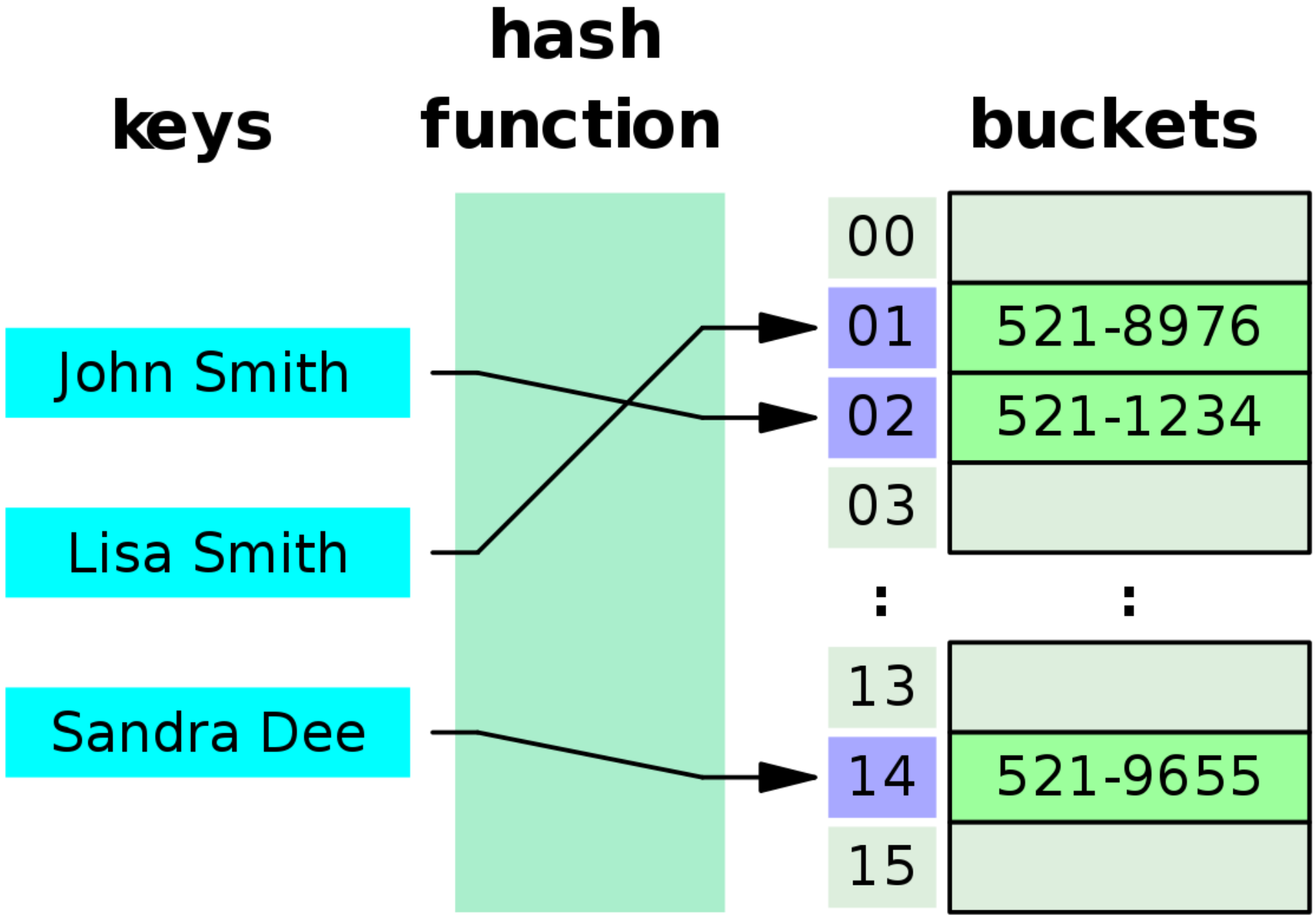


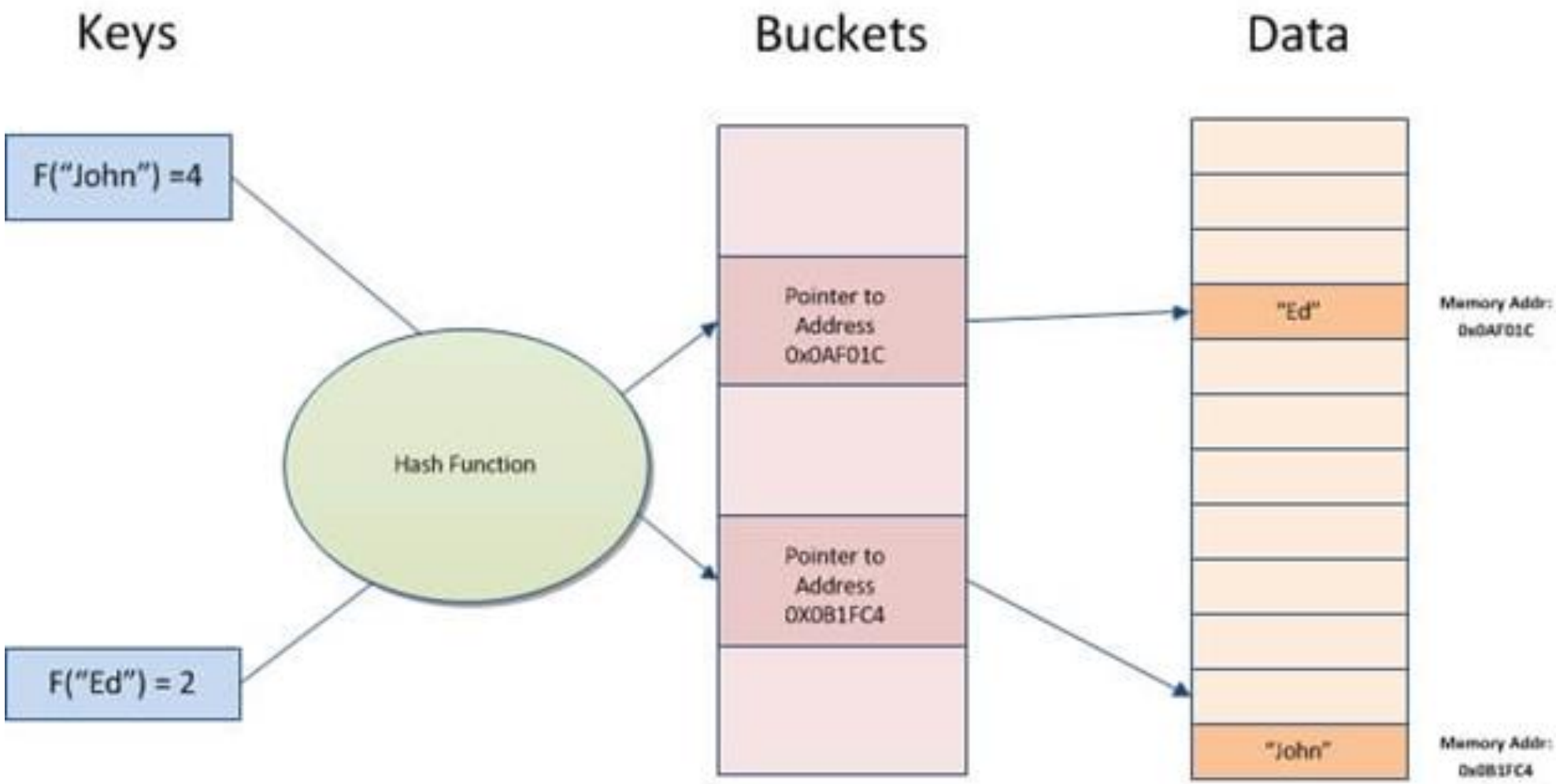
```sql
CREATE TABLE employees (
    id INT PRIMARY KEY,
    first_name VARCHAR(50),
    last_name VARCHAR(50),
    email VARCHAR(100),
    department VARCHAR(50),
    salary DECIMAL(10, 2)
);
```

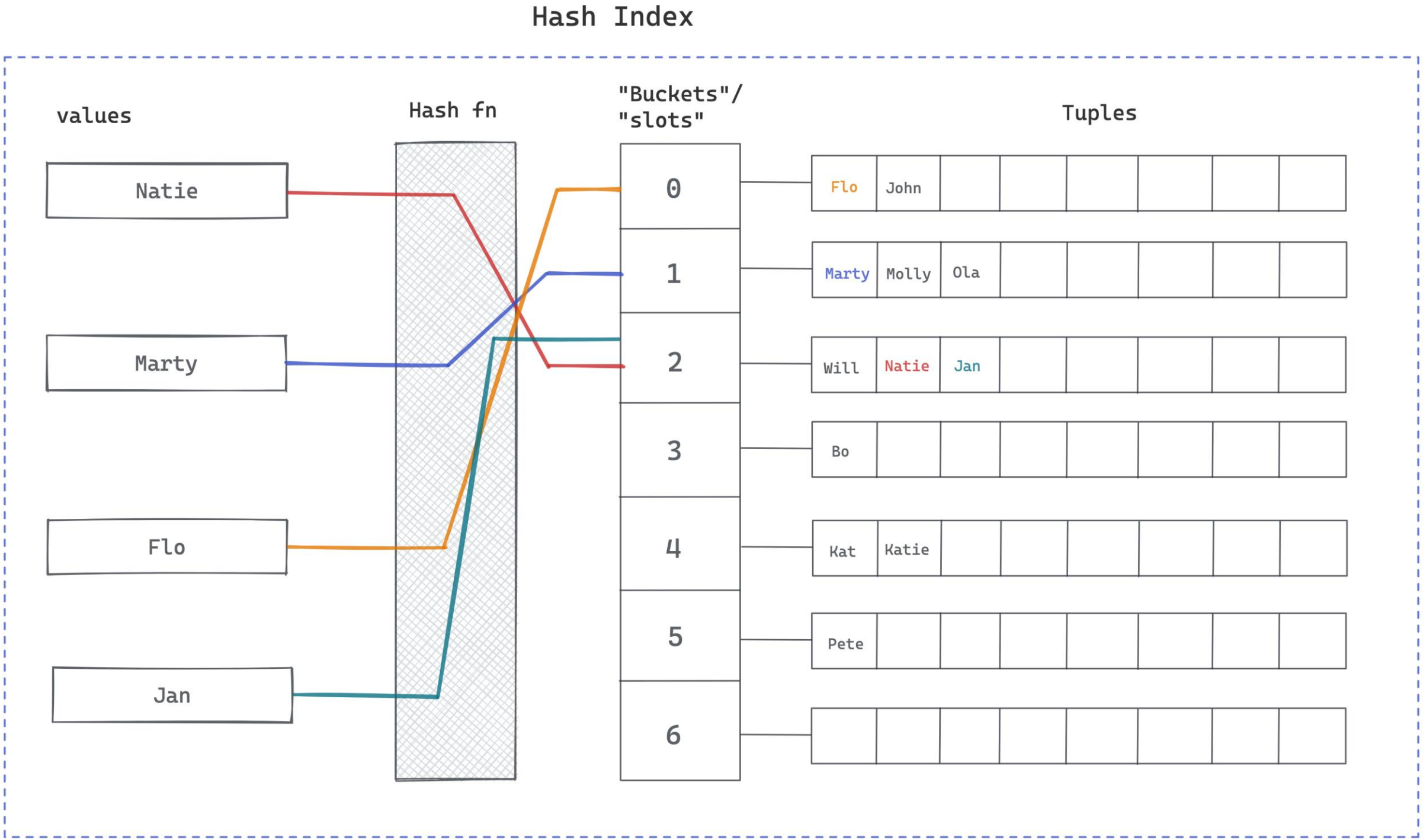
```sql
CREATE INDEX idx_last_name ON employees (last_name);
```

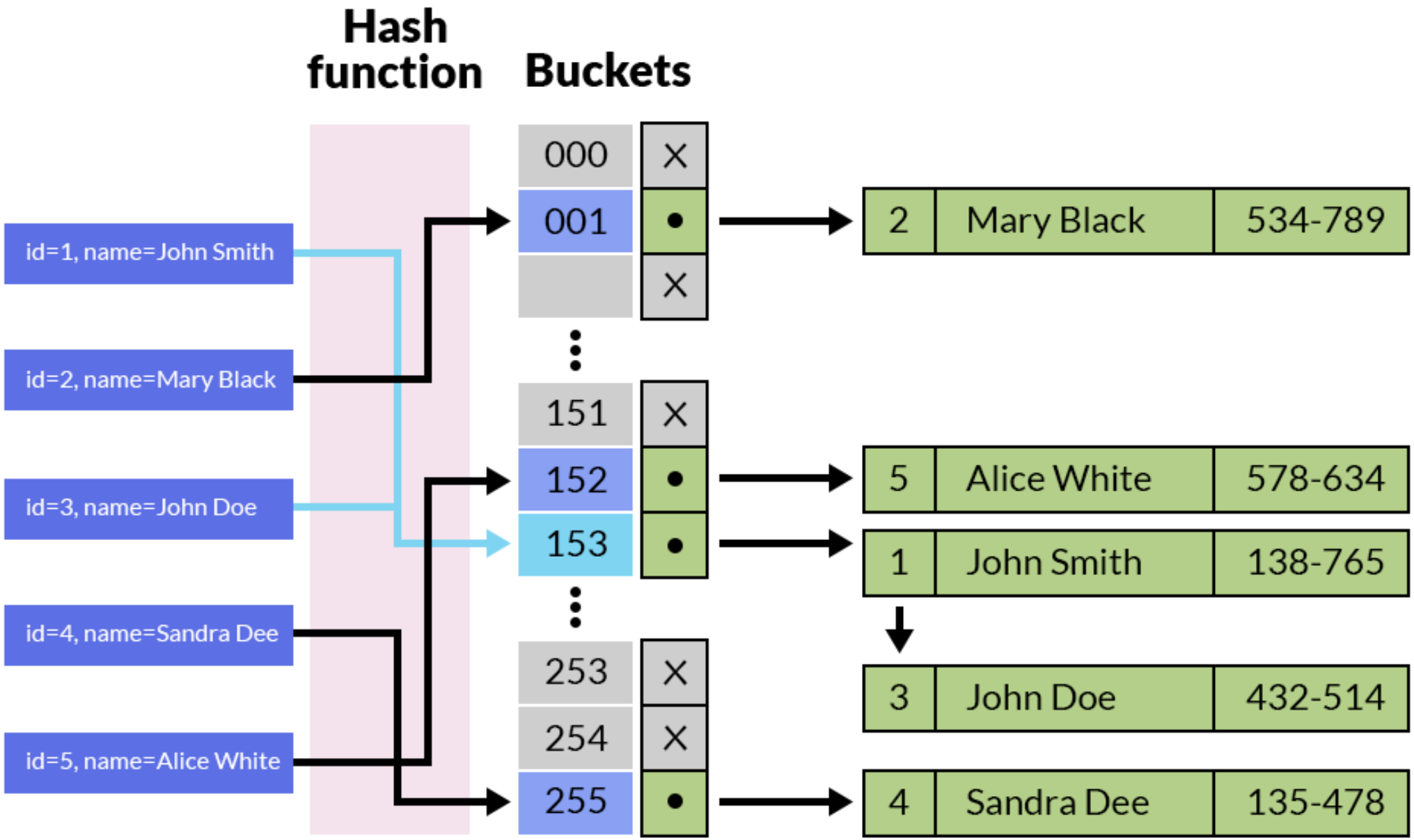
```sql
SELECT * FROM employees WHERE last_name = 'Smith';
```

```sql
CREATE INDEX idx_full_name ON employees (first_name, last_name);
```


keys
hash function
buckets
John Smith
Lisa Smith
Sandra Dee
00
01
02
03
13
14
15
521-8976
521-1234
521-9655


Keys
Buckets
Data
F("John") =4
F("Ed") = 2
Hash Function
Pointer to Address 0x0AF01C
Pointer to Address 0X0B1FC4
"Ed"
"John"
Memory Addr: 0x0AF01C
Memory Addr: 0x0B1FC4


Hash Index
values
Hash fn
"Buckets"/
"slots"
Tuples
Natie
Marty
Flo
Jan
0
1
2
3
4
5
6
Flo
John
Marty
Molly
Ola
Will
Natie
Jan
Bo
Kat
Katie
Pete


Hash function
Buckets
id=1, name=John Smith
id=2, name=Mary Black
id=3, name=John Doe
id=4, name=Sandra Dee
id=5, name=Alice White
000
001
151
152
153
253
254
255
2
Mary Black
534-789
5
Alice White
578-634
1
John Smith
138-765
3
John Doe
432-514
4
Sandra Dee
135-478

How do Database Indexes Work?

- **Index Creation**: The database administrator creates an index on a specific column or set of columns.
- **Index Building**: The database management system builds the index by scanning the table and storing the values of the indexed column(s) along with a pointer to the corresponding data.
- **Query Execution**: When a query is executed, the database engine checks if an index exists for the requested column(s).
- **Index Search**: If an index exists, the database searches the index for the requested data, using the pointers to quickly locate the data.
- **Data Retrieval**: The database retrieves the requested data, using the pointers from the index.

Benefits of Database Indexes

- **Faster Query Performance**: Indexes can significantly improve query performance especially for large datasets by reducing the amount of data that needs to be scanned.
- **Reduced CPU Usage**: By reducing the number of rows that need to be scanned, indexes can decrease CPU usage and optimize resource utilization.
- **Rapid Data Retrieval**: Indexes enable quick data retrieval for queries that involve equality or range conditions on the indexed columns.
- **Efficient Sorting**: Indexes can also be used to efficiently sort data based on the indexed columns, eliminating the need for expensive sorting operations.
- **Better Data Organization**: Indexes can help maintain data organization and structure, making it easier to manage and maintain the database.

Types of Database Indexes

Indexes based on Structure and Key Attributes:

- **Primary Index:** Automatically created when a primary key constraint is defined on a table. Ensures uniqueness and helps with super-fast lookups using the primary key.
- **Clustered Index:** Determines the order in which data is physically stored in the table. A clustered index is most useful when we're searching in a range. Only one clustered index can exist per table.
- **Non-clustered or Secondary Index:** This index does not store data in the order of the index. Instead, it provides a list of virtual pointers or references to the location where the data is actually stored.

Types of Database Indexes

- **Indexes based on Data Coverage:**
- **Dense index:** Has an entry for every search key value in the table. Suitable for situations where the data has a small number of distinct search key values or when fast access to individual records is required.
- **Sparse index:** Has entries only for some of the search key values. Suitable for situations where the data has a large number of distinct search key values.

Specialized Index Types:

- **Bitmap Index:** Excellent for columns with low cardinality (few distinct values). Common in data warehousing.
- **Hash Index:** A index that uses a hash function to map values to specific locations. Great for exact match queries.
- **Filtered Index:** Indexes a subset of rows based on a specific filter condition. Useful to improve query speed on commonly filtered columns.
- **Covering Index:** Includes all the columns required by a query in the index itself, eliminating the need to access the underlying table data.
- **Function-based index:** Indexes that are created based on the result of a function or expression applied to one or more columns of a table.
- **Full-Text Index**: A index designed for full-text search, allowing for efficient searching of text data.
- **Spatial Index:** Used for indexing geographical data types.

What Data Structure do Indexes use?

- **B-Tree (Balanced Tree)**
- **Hash Tables**
- **Bitmaps**
- Burrows–Wheeler (Strings of Characters)

Burrows–Wheeler transform

- The **Burrows–Wheeler transform** (**BWT**) rearranges a character string into runs of similar characters, in a manner that can be reversed to recover the original string.

Burrows–Wheeler transform

Transformation				
1. Input	2. All rotations	3. Sort into lexical order	4. Take the last column	5. Output
^BANANA$	^BANANA$ $^BANANA A$^BANAN NA$^BANA ANA$^BAN NANA$^BA ANANA$^B BANANA$^	ANANA$^B ANA$^BAN A$^BANAN BANANA$^ NANA$^BA NA$^BANA ^BANANA$ $^BANANA	ANANA$^B ANA$^BAN A$^BANAN BANANA$^ NANA$^BA NA$^BANA ^BANANA$ $^BANANA	BNN^AA$A

Inverse transformation

Input

BNN^AA$A

Add 1	Sort 1	Add 2	Sort 2
B	A	BA	AN
N	A	NA	AN
N	A	NA	A$
^	B	^B	BA
A	N	AN	NA
A	N	AN	NA
$	^	$^	^B
A	$	A$	$^

Add 3	Sort 3	Add 4	Sort 4
BAN	ANA	BANA	ANAN
NAN	ANA	NANA	ANA$
NA$	A$^	NA$^	A$^B
^BA	BAN	^BAN	BANA
ANA	NAN	ANAN	NANA
ANA	NA$	ANA$	NA$^
$^B	^BA	$^BA	^BAN
A$^	$^B	A$^B	$^BA

Add 5	Sort 5	Add 6	Sort 6
BANAN	ANANA	BANANA	ANANA$
NANA$	ANA$^	NANA$^	ANA$^B
NA$^B	A$^BA	NA$^BA	A$^BAN
^BANA	BANAN	^BANAN	BANANA
ANANA	NANA$	ANANA$	NANA$^
ANA$^	NA$^B	ANA$^B	NA$^BA
$^BAN	^BANA	$^BANA	^BANAN
A$^BA	$^BAN	A$^BAN	$^BANA

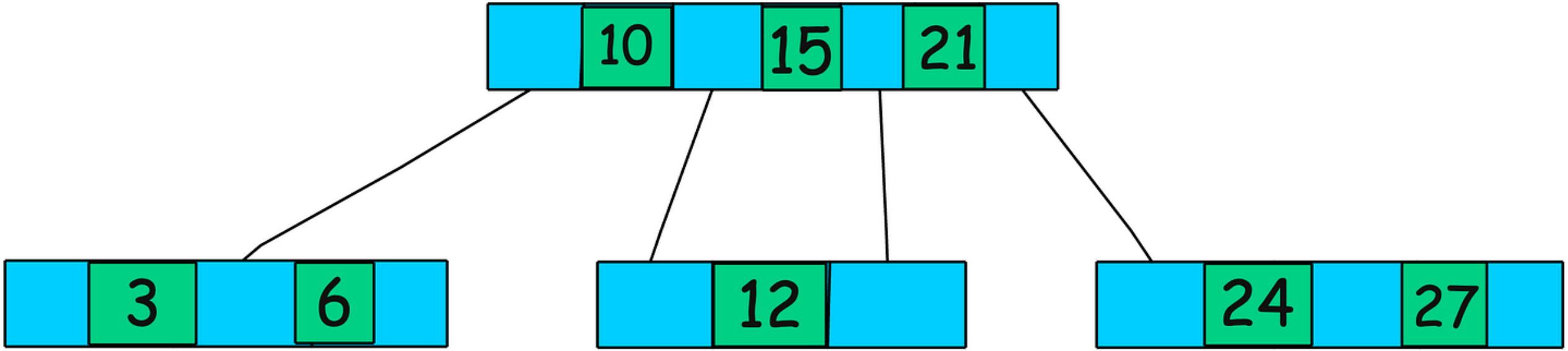
Add 7	Sort 7	Add 8	Sort 8
BANANA$	ANANA$^	BANANA$^	ANANA$^B
NANA$^B	ANA$^BA	NANA$^BA	ANA$^BAN
NA$^BAN	A$^BANA	NA$^BANA	A$^BANAN
^BANANA	BANANA$	^BANANA$	BANANA$^
ANANA$^	NANA$^B	ANANA$^B	NANA$^BA
ANA$^BA	NA$^BAN	ANA$^BAN	NA$^BANA
$^BANAN	^BANANA	$^BANANA	^BANANA$
A$^BANA	$^BANAN	A$^BANAN	$^BANANA

Output

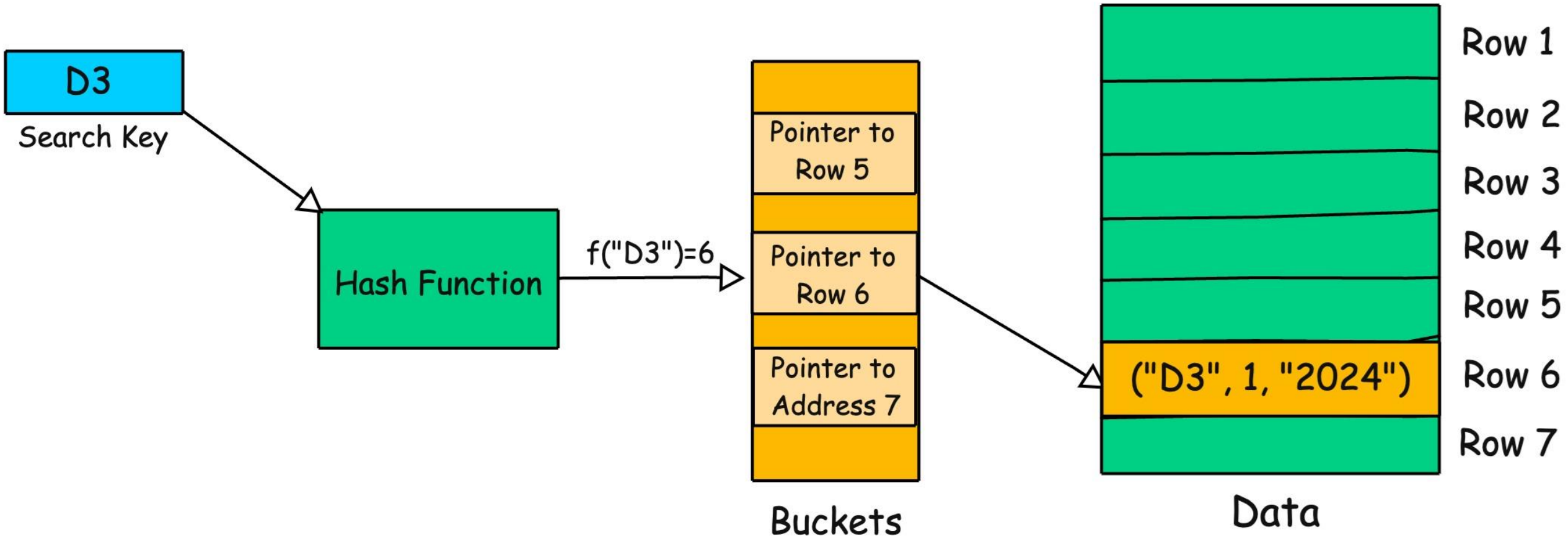
^BANANA$

Benefits of B-Tree

- **Self-Balancing:** B-trees ensure that the 'height' of the tree stays balanced even when inserting or deleting data. This ensures logarithmic time complexity for insertion, deletion, and searching.
- **Ordered:** B-trees keep the data sorted, making range queries ("find all orders between date X and Y") and inequality comparisons very fast.
- **Disk-Friendly:** B-trees are designed to work well with disk-based storage. A single node of a B-tree often corresponds to a disk block, minimizing disk access operations.



blog.algomaster.io


D3
Search Key
Hash Function
f("D3")=6
Pointer to Row 5
Pointer to Row 6
Pointer to Address 7
Buckets
("D3", 1, "2024")
Row 1
Row 2
Row 3
Row 4
Row 5
Row 6
Row 7
Data

Bitmaps

A	0 1 1 0 1
B	1 0 0 1 1
C	0 1 0 1 0
D	0 0 1 0 0
E	1 0 1 0 0

How to use Database Indexes Smartly?

To get the most out of database indexes, consider these best practices:

- **Identify Query Patterns**: Analyze the most frequent and critical queries executed against your database to determine which columns to index and which type of index to use.
- **Index Frequently Used Columns**: Consider indexing columns that are frequently used in WHERE, JOIN, and ORDER BY clauses.
- **Index Selective Columns:** Indexes are most effective on columns with a good spread of data values (high cardinality). Indexing a gender column might be less beneficial than one with a unique customer_id.
- **Use Appropriate Index Types**: Choose the right index type for your data and queries.
- **Consider Composite Indexes**: For queries involving multiple columns, consider creating composite indexes that encompass all relevant columns. This reduces the need for multiple single-column indexes and improves query performance.
- **Monitor Index Performance**: Regularly monitor index performance, remove unused indexes and adjust your indexing strategy as the database workload evolves.
- **Avoid Over-Indexing**: Avoid creating too many indexes, as this can lead to increased storage requirements and slower write performance.
  - Indexes take up extra disk space since they're additional data structures that need to be stored alongside your tables.
  - Every time you insert, update, or delete data in a table with an index, the index needs to update too. This can slightly slow down write operations.

To summarize, indexes are a powerful tool to optimize database query performance.

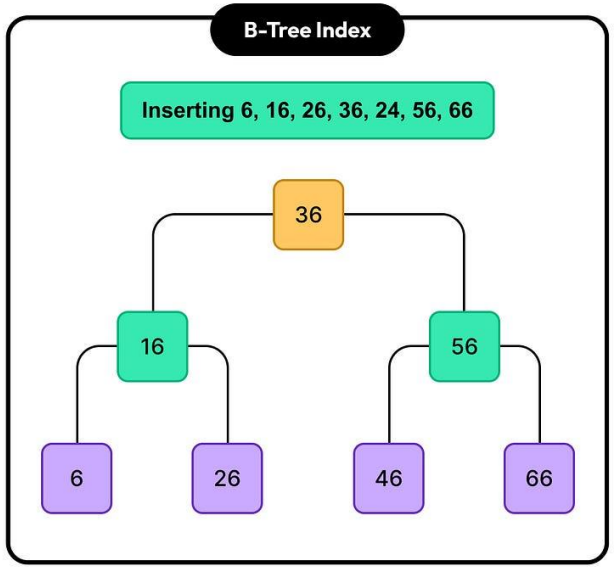
But remember to choose the right column and index type, monitor performance, and avoid over-indexing to get the most out of them.

Database Index Internals

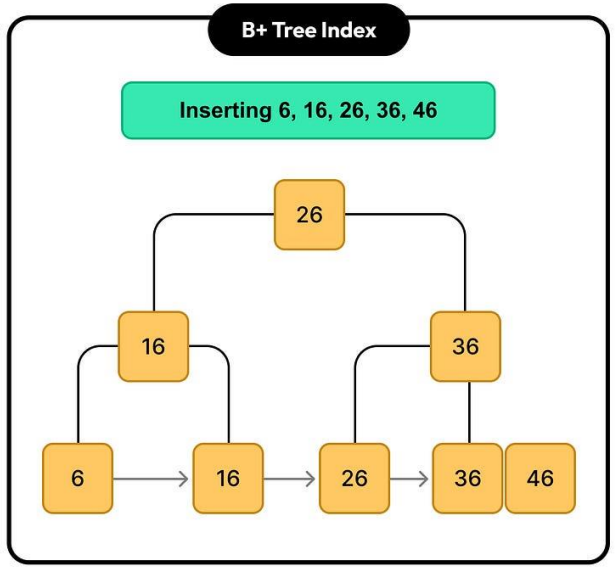
Key Data Structures

ByteByteGo

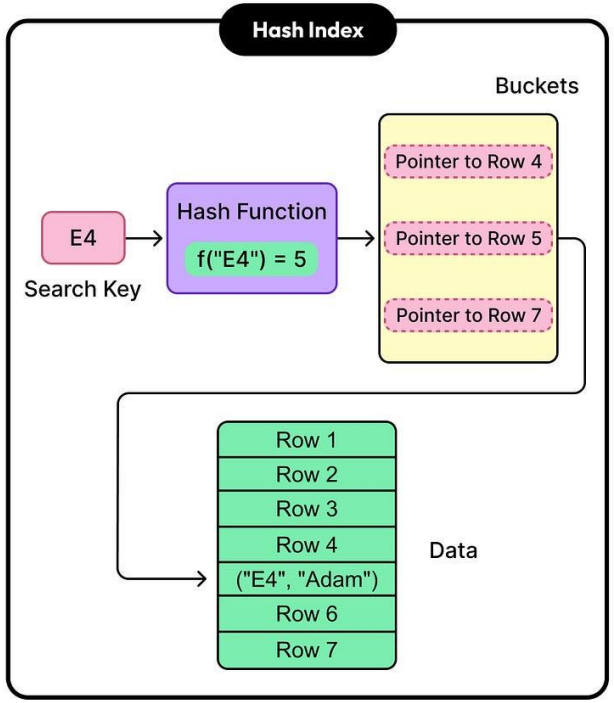
B-Tree Index



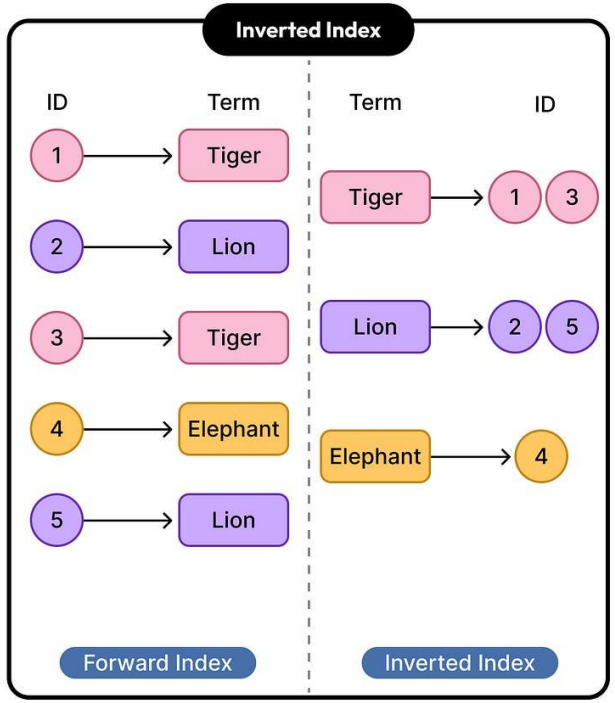
B+ Tree Index

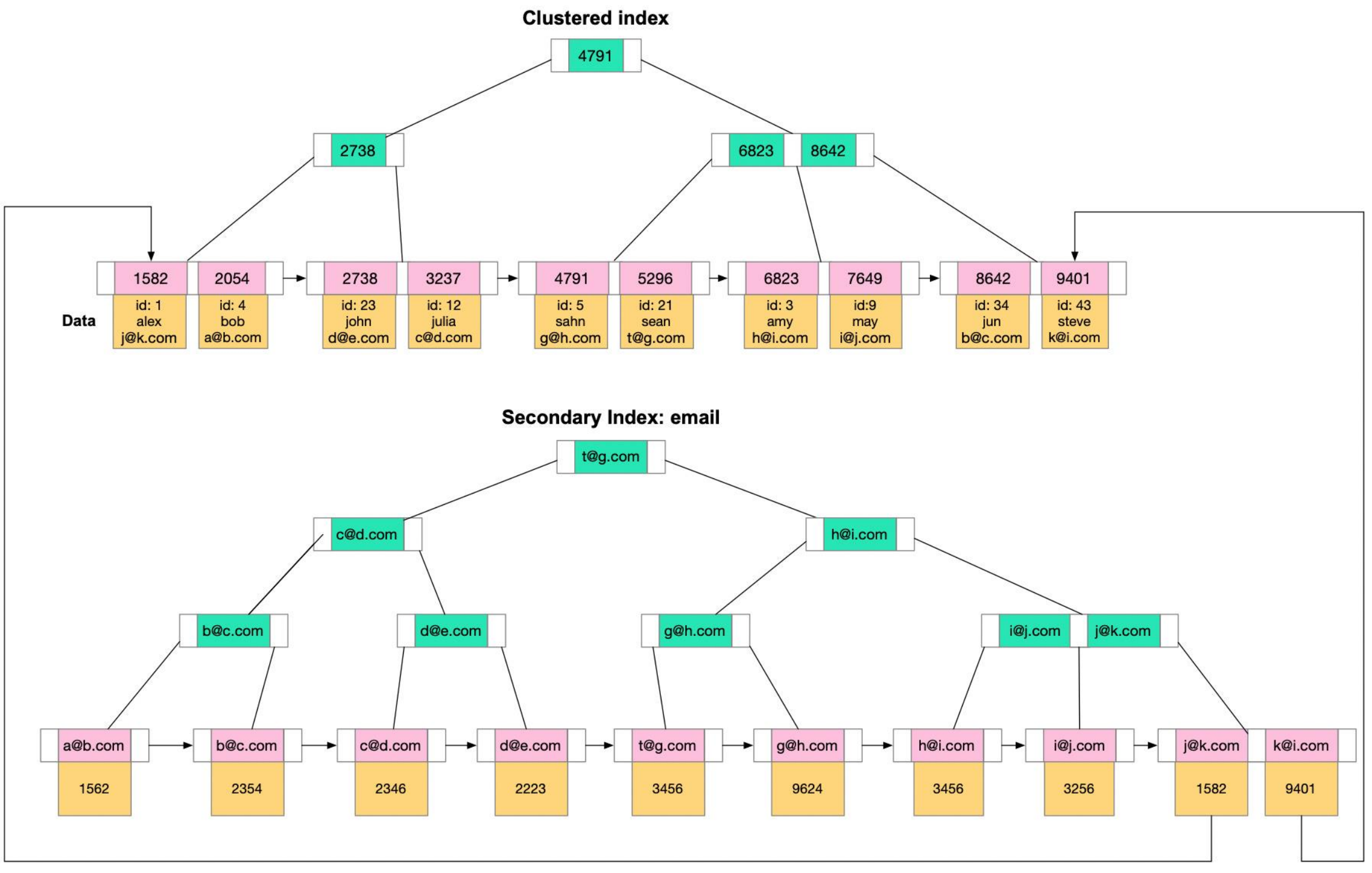


Hash Index



Inverted Index




Clustered index
4791
2738
6823
8642
Data
1582
id: 1
alex
j@k.com
2054
id: 4
bob
a@b.com
2738
id: 23
john
d@e.com
3237
id: 12
julia
c@d.com
4791
id: 5
sahn
g@h.com
5296
id: 21
sean
t@g.com
6823
id: 3
amy
h@i.com
7649
id:9
may
i@j.com
8642
id: 34
jun
b@c.com
9401
id: 43
steve
k@i.com
Secondary Index: email
t@g.com
c@d.com
h@i.com
b@c.com
d@e.com
g@h.com
i@j.com
j@k.com
a@b.com
1562
b@c.com
2354
c@d.com
2346
d@e.com
2223
t@g.com
3456
g@h.com
9624
h@i.com
3456
i@j.com
3256
j@k.com
1582
k@i.com
9401

Databases

توضیح	پایگاه داده مناسب	کاربرد
نیاز به تراکنش دقیق و روابط پیچیده	PostgreSQL / Oracle SQL	سامانه بانکی
داده متنوع و متغیر، مقیاس بالا	MongoDB / Cassandra	شبکه اجتماعی
سرعت بالا و real-time	Redis / DynamoDB	اپلیکیشن چت
داده توزیع‌شده، حجم بسیار زیاد	HBase / Cassandra	تحلیل کلان‌داده‌ها
داده توزیع‌شده، حجم بسیار زیاد	HBase / Cassandra	تحلیل کلان‌داده‌ها
سیستم هیبریدی	ترکیبی از MySQL (برای سفارش‌ها) و MongoDB (برای محصولات)	فروشگاه آنلاین

نوع NoSQL	توضیح	مثال سیستم
Key–Value	ذخیره داده به صورت جفت کلید و مقدار	Redis, DynamoDB
Document-Oriented	ذخیره داده به صورت سند JSON یا BSON	MongoDB, CouchDB
Column-Based	ذخیره داده به صورت ستون‌ها،	Cassandra, HBase
Graph-Based	ذخیره داده به صورت گره و یال، مناسب برای شبکه‌ها	Neo4j, Amazon Neptune

مورد استفاده	مناسب‌تر SQL	مناسب‌تر NoSQL
سیستم‌های مالی و بانکی	✅	❌
شبکه‌های اجتماعی با داده متغیر	❌	✅
گزارش‌گیری پیچیده (Analytics)	✅	❌
سیستم‌های real-time و مقیاس بزرگ	❌	✅
داده‌های ساختارمند و پایدار	✅	❌
داده‌های نیمه‌ساخت‌یافته (JSON, Logs, Sensors)	❌	✅

http://www.linkedin.com/in/williamhgates

Bill Gates

Greater Seattle Area | Philanthropy

Summary

Co-chair of the Bill & Melinda Gates Foundation. Chairman, Microsoft Corporation. Voracious reader. Avid traveler. Active blogger.

Experience

Co-chair • Bill & Melinda Gates Foundation
2000 – Present

Co-founder, Chairman • Microsoft
1975 – Present

Education

Harvard University
1973 – 1975

Lakeside School, Seattle

Contact Info

Blog: thegatesnotes.com
Twitter: @BillGates

users table

user_id	first_name	last_name	summary
251	Bill	Gates	Co-chair of … blogger.

region_id	industry_id	photo_id
us:91	131	57817532

regions table

id	region_name
us:7	Greater Boston Area
us:91	Greater Seattle Area

industries table

id	industry_name
43	Financial Services
48	Construction
131	Philanthropy

positions table

id	user_id	job_title	organization
458	251	Co-chair	Bill & Melinda Gates F…
457	251	Co-founder, Chairman	Microsoft

education table

id	user_id	school_name	start	end
807	251	Harvard University	1973	1975
806	251	Lakeside School, Seattle	NULL	NULL

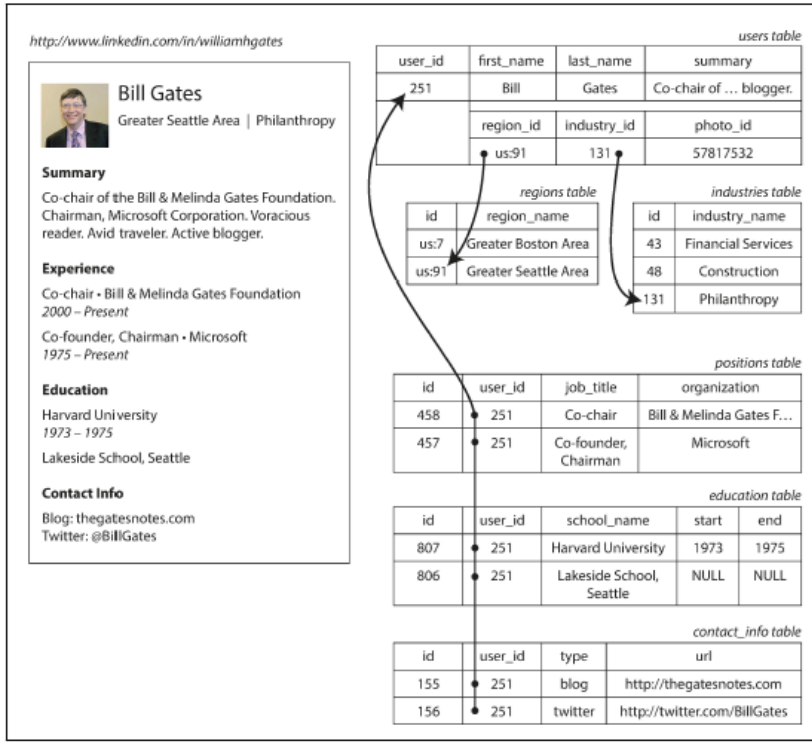
contact_info table

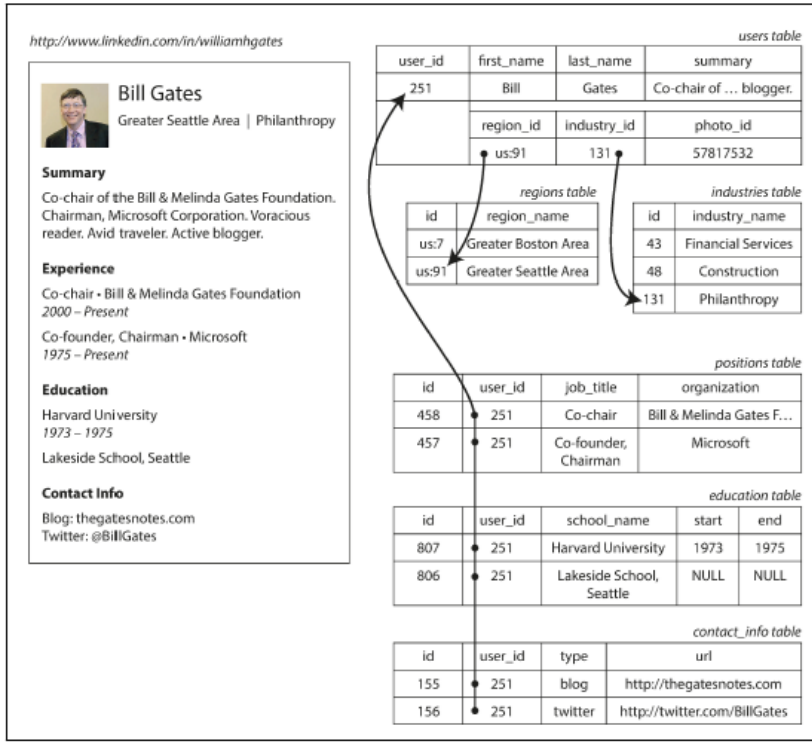
id	user_id	type	url
155	251	blog	http://thegatesnotes.com
156	251	twitter	http://twitter.com/BillGates

Representing a LinkedIn profile as a JSON document

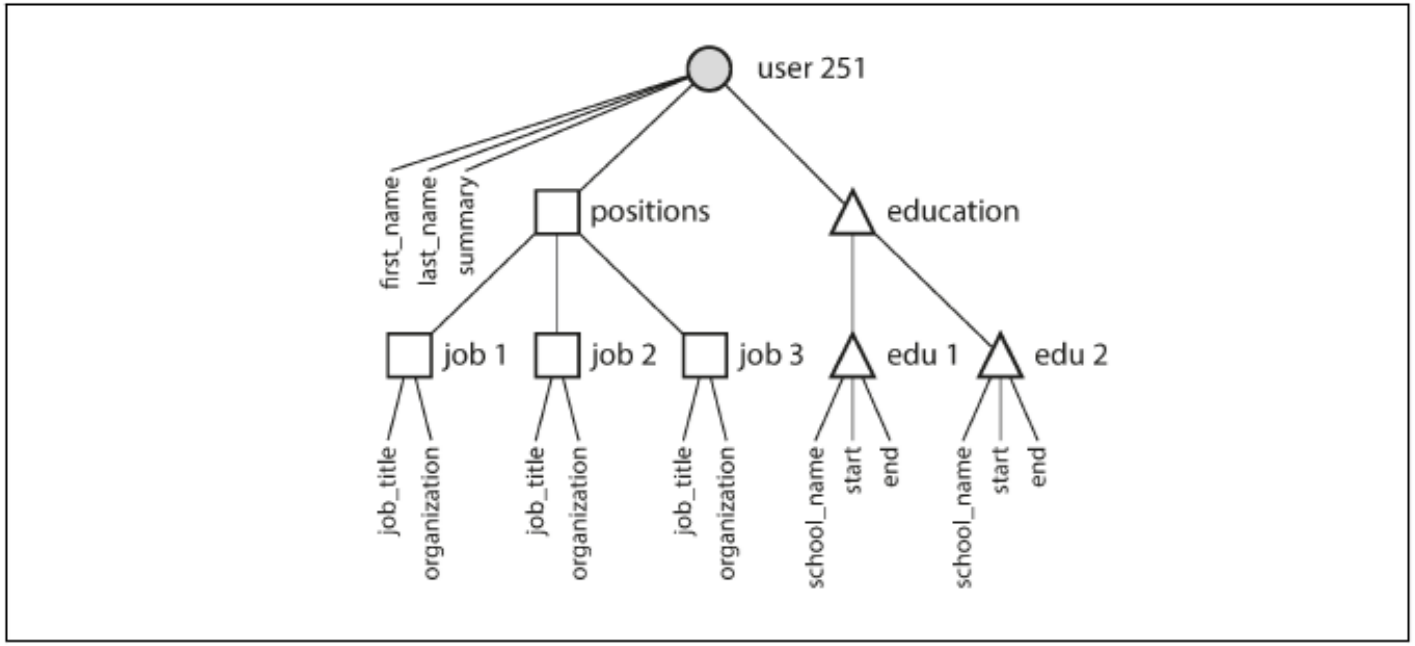
```
{
  "user_id":     251,
  "first_name":  "Bill",
  "last_name":   "Gates",
  "summary":     "Co-chair of the Bill & Melinda Gates... Active blogger.",
  "region_id":   "us:91",
  "industry_id": 131,
  "photo_url":   "/p/7/000/253/05b/308dd6e.jpg",

  "positions": [
    {"job_title": "Co-chair", "organization": "Bill & Melinda Gates Foundation"},
    {"job_title": "Co-founder, Chairman", "organization": "Microsoft"}
  ],
  "education": [
    {"school_name": "Harvard University",       "start": 1973, "end": 1975},
    {"school_name": "Lakeside School, Seattle", "start": null, "end": null}
  ],
  "contact_info": {
    "blog":    "http://thegatesnotes.com",
    "twitter": "http://twitter.com/BillGates"
  }
}
```





- Consistent style and spelling across profiles
- Avoiding ambiguity (e.g., if there are several cities with the same name)
- Ease of updating—the name is stored in only one place, so it is easy to update across the board if it ever needs to be changed (e.g., change of a city name due to political events)
- Localization support—when the site is translated into other languages, the standardized lists can be localized, so the region and industry can be displayed in the viewer's language
- Better search—e.g., a search for philanthropists in the state of Washington can match this profile, because the list of regions can encode the fact that Seattle is in Washington (which is not apparent from the string `"Greater Seattle Area"`)



One-to-many relationships forming a tree structure.

http://www.linkedin.com/in/williamhgates

Bill Gates

Greater Seattle Area | Philanthropy

Summary

Co-chair of the Bill & Melinda Gates Foundation. Chairman, Microsoft Corporation. Voracious reader. Avid traveler. Active blogger.

Experience

Co-chair • Bill & Melinda Gates Foundation
2000 – Present

Co-founder, Chairman • Microsoft
1975 – Present

Education

Harvard University
1973 – 1975

Lakeside School, Seattle

Contact Info

Blog: thegatesnotes.com
Twitter: @BillGates

users table

user_id	first_name	last_name	summary
251	Bill	Gates	Co-chair of … blogger.

region_id	industry_id	photo_id
us:91	131	57817532

regions table

id	region_name
us:7	Greater Boston Area
us:91	Greater Seattle Area

industries table

id	industry_name
43	Financial Services
48	Construction
131	Philanthropy

positions table

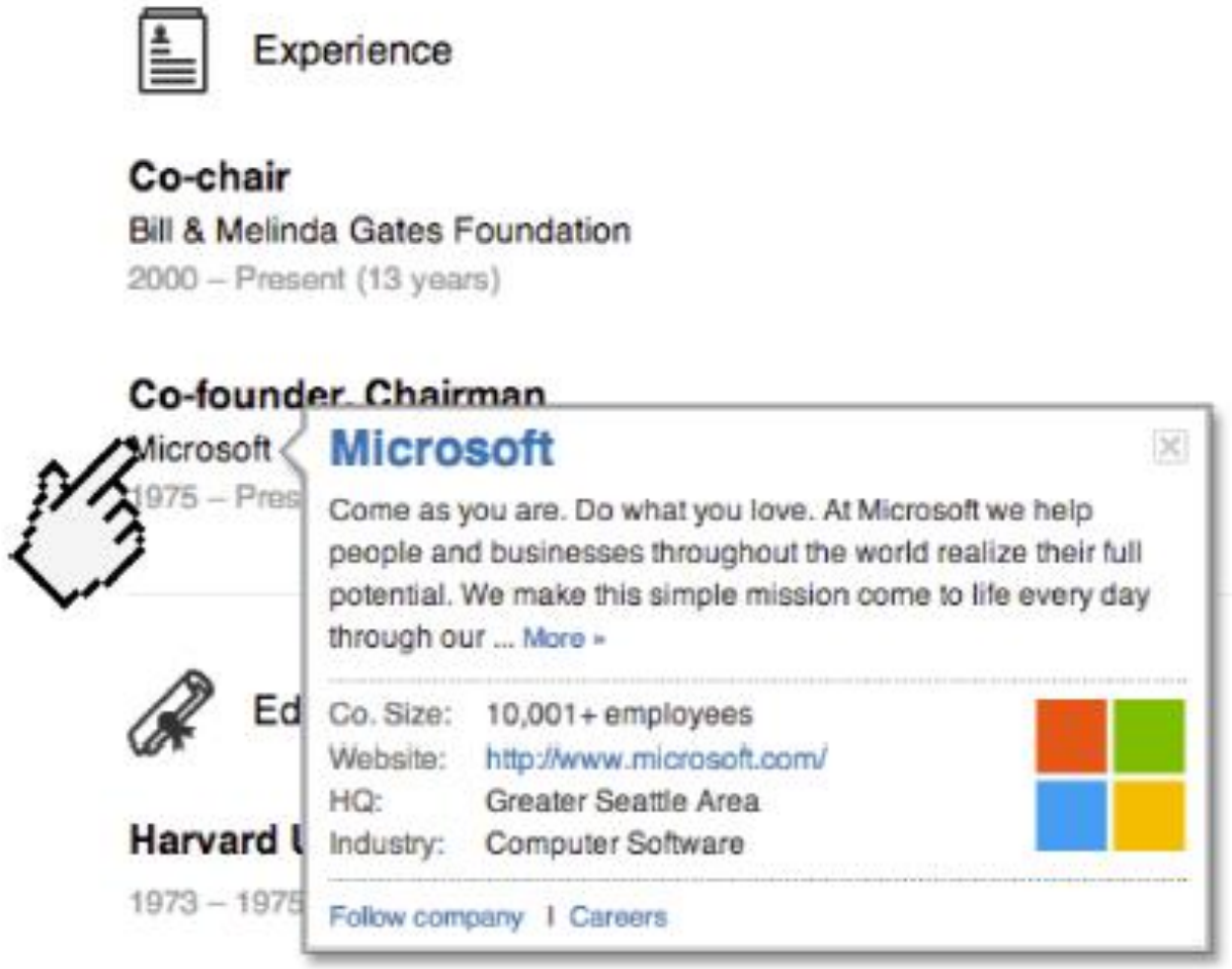
id	user_id	job_title	organization
458	251	Co-chair	Bill & Melinda Gates F…
457	251	Co-founder, Chairman	Microsoft

education table

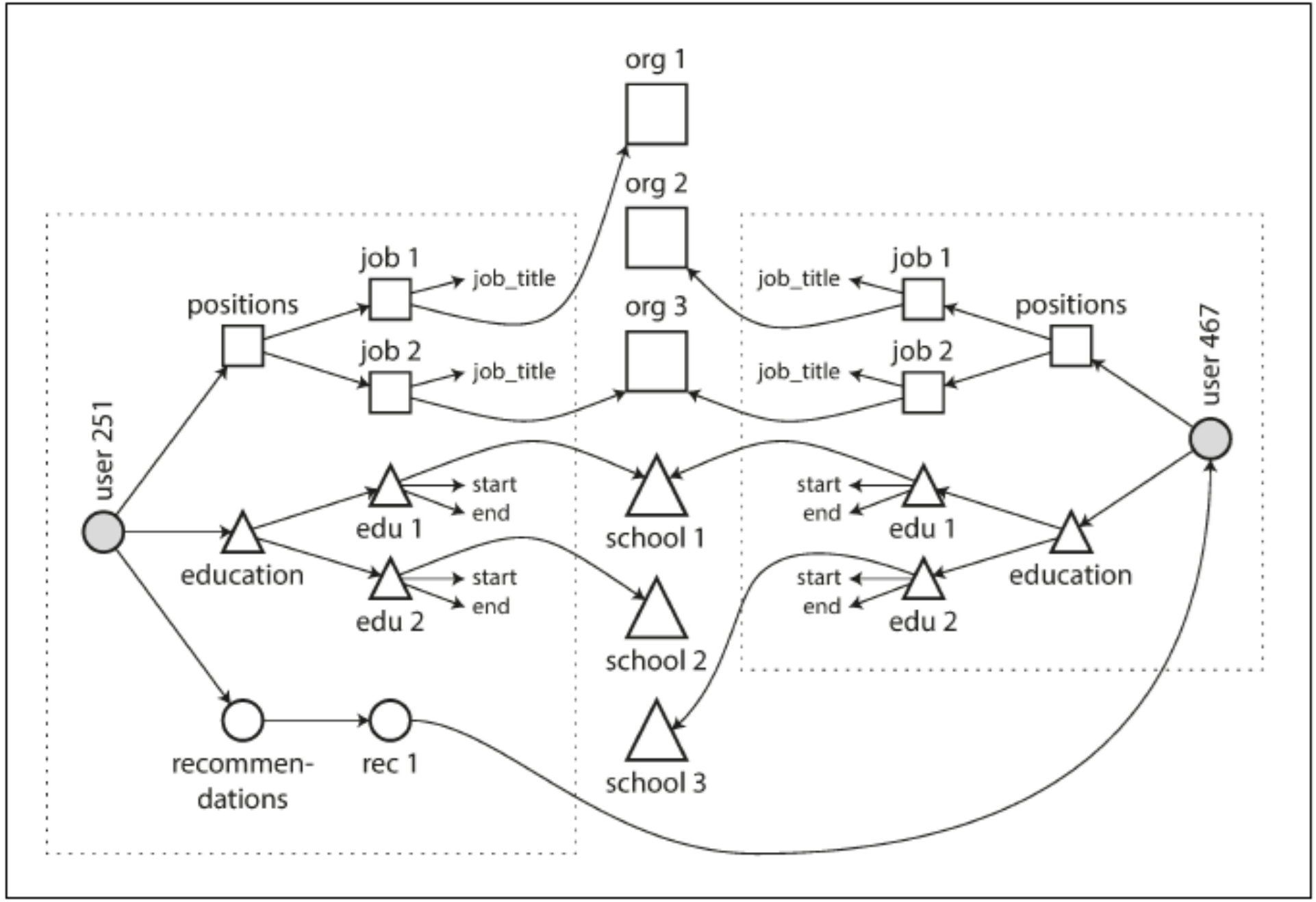
id	user_id	school_name	start	end
807	251	Harvard University	1973	1975
806	251	Lakeside School, Seattle	NULL	NULL

contact_info table

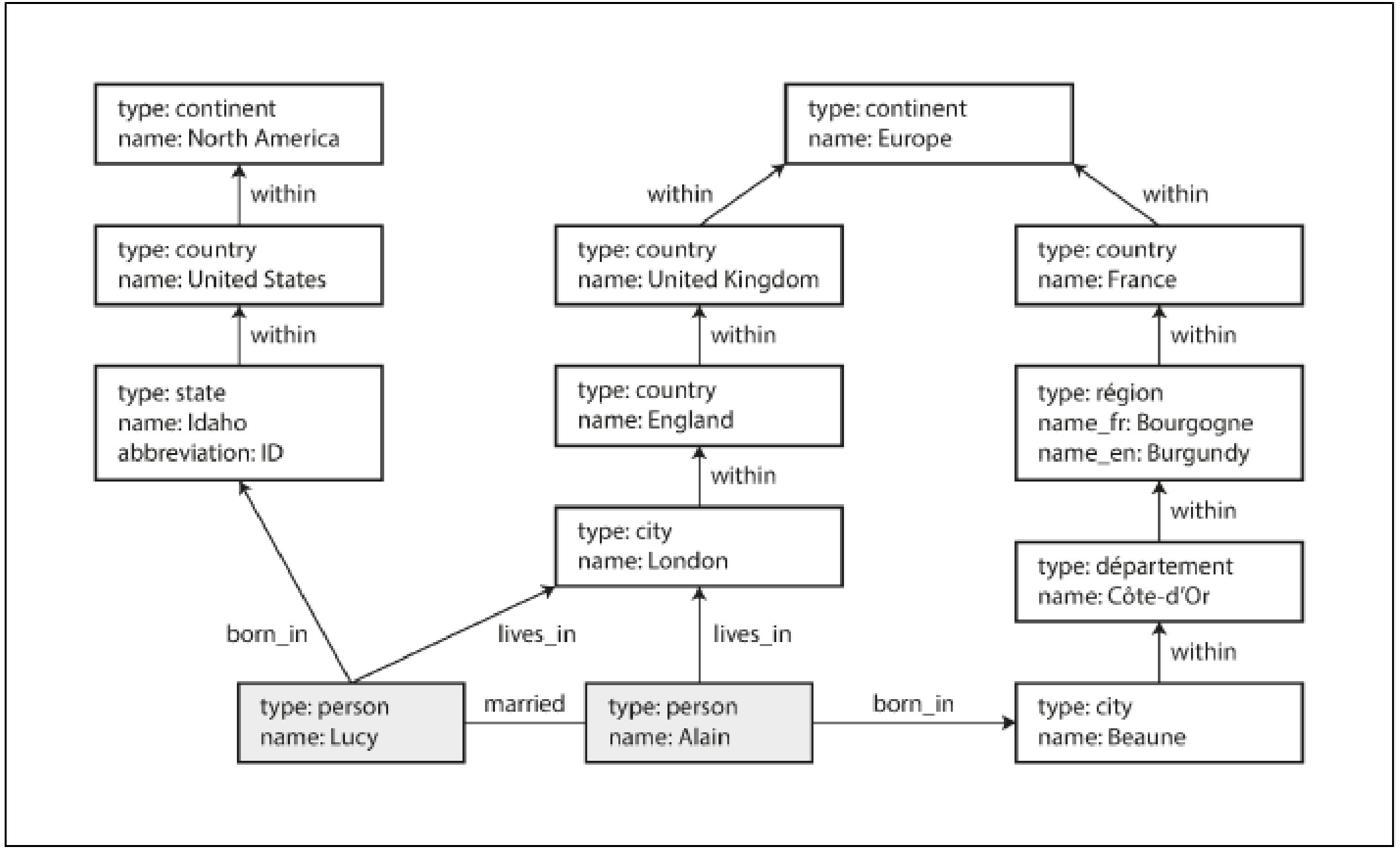
id	user_id	type	url
155	251	blog	http://thegatesnotes.com
156	251	twitter	http://twitter.com/BillGates



The company name is not just a string, but a link to a company entity. Screenshot of linkedin.com.



Extending résumés with many-to-many relationships.



Example of graph-structured data (boxes represent vertices, arrows represent edges).

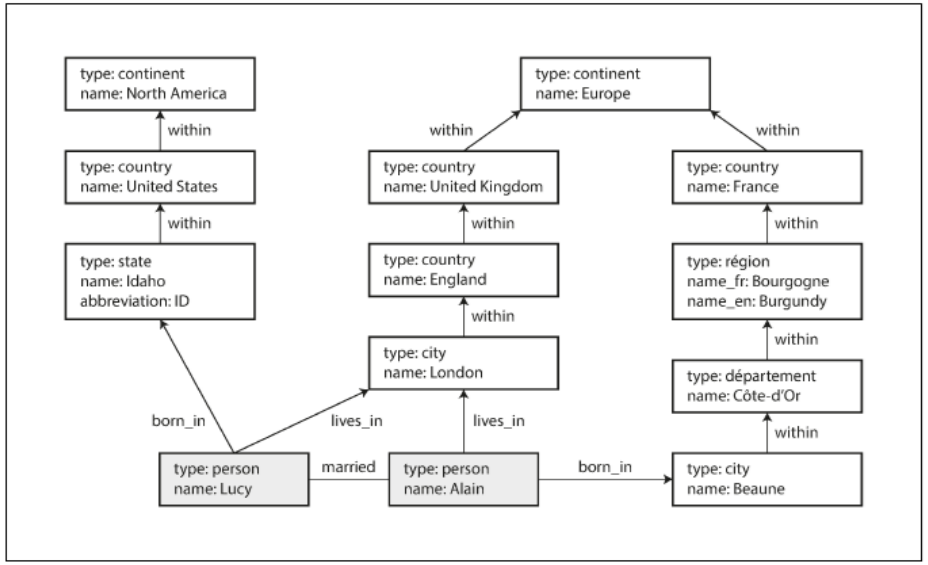
Property Graphs

In the property graph model, each vertex consists of:

- A unique identifier
- A set of outgoing edges
- A set of incoming edges
- A collection of properties (key-value pairs)

Each edge consists of:

- A unique identifier
- The vertex at which the edge starts (the *tail vertex*)



Example of graph-structured data (boxes represent vertices, arrows represent edges).

Representing a property graph using a relational schema

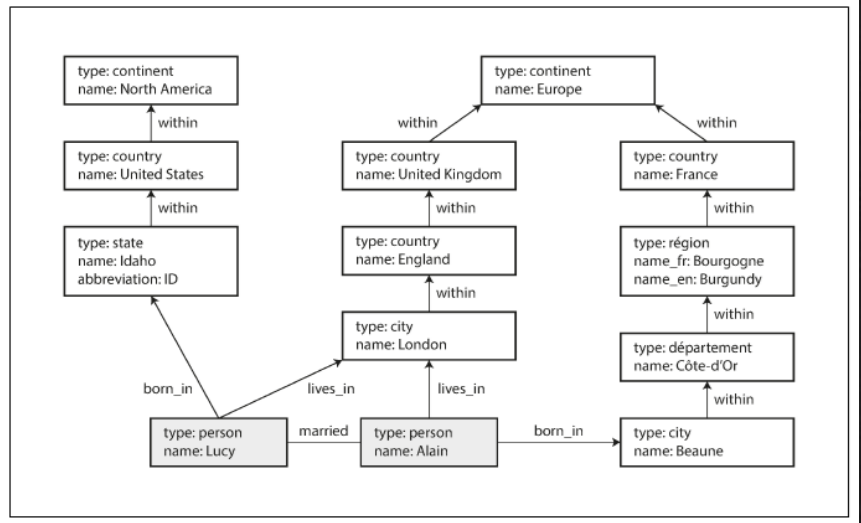
```
CREATE TABLE vertices (
    vertex_id   integer PRIMARY KEY,
    properties  json
);

CREATE TABLE edges (
    edge_id     integer PRIMARY KEY,
    tail_vertex integer REFERENCES vertices (vertex_id),
    head_vertex integer REFERENCES vertices (vertex_id),
    label       text,
    properties  json
);

CREATE INDEX edges_tails ON edges (tail_vertex);
CREATE INDEX edges_heads ON edges (head_vertex);
```

A subset of the data in Figure 2-5, represented as a Cypher query

```
CREATE
  (NAmerica:Location {name:'North America', type:'continent'}),
  (USA:Location      {name:'United States', type:'country'  }),
  (Idaho:Location    {name:'Idaho',         type:'state'    }),
  (Lucy:Person       {name:'Lucy' }),
  (Idaho) -[:WITHIN]->  (USA)  -[:WITHIN]-> (NAmerica),
  (Lucy)  -[:BORN_IN]-> (Idaho)
```



Example of graph-structured data (boxes represent vertices, arrows represent edges).

Cypher query to find people who emigrated from the US to Europe

```
MATCH
  (person) -[:BORN_IN]->  () -[:WITHIN*0..]-> (us:Location {name:'United States'}),
  (person) -[:LIVES_IN]-> () -[:WITHIN*0..]-> (eu:Location {name:'Europe'})
RETURN person.name
```

- خلاصه:

ویژگی	توضیح
ساختار داده	جفت کلید-مقدار
نوع پایگاه داده	NoSQL
کاربرد اصلی	ذخیره‌سازی سریع داده‌های عظیم
مثال‌ها	Redis، DynamoDB، Riak، LevelDB
کاربرد در Big Data	خروجی MapReduce، کش داده، مدیریت session، و داده‌های real-time

در حوزه **Big Data** (کلان‌داده)، مفهوم **Key-Value** (**کلید-مقدار**) یکی از بنیادی‌ترین و ساده‌ترین مدل‌های ذخیره‌سازی داده است. در این مدل، داده‌ها به‌صورت جفت‌های **کلید (Key)** و **مقدار (Value)** ذخیره و بازیابی می‌شوند.

🔹 تعریف ساده:

هر **کلید (Key)** مانند یک شناسهٔ یکتا است که به **یک مقدار (Value)** اشاره می‌کند.
شما می‌توانید با داشتن کلید، به سرعت مقدار مربوطه را پیدا کنید — بدون نیاز به جستجو در کل پایگاه داده.

🔹 در حوزه‌ی Big Data چه کاربردی دارد؟

در **سیستم‌های کلان‌داده**، حجم بسیار بالایی از اطلاعات باید به سرعت خوانده و نوشته شود.
مدل **Key-Value** برای این منظور عالی است، چون:

- جستجو با کلید بسیار سریع است (O(1) یا نزدیک به آن)
- ساختار داده‌ها ساده و قابل توزیع بین چندین سرور است
- برای ذخیره‌سازی افقی (horizontal scaling) بسیار مناسب است

MapReduce

- در MapReduce، خروجی هر مرحله معمولاً به‌صورت جفت‌های key-value است.
  مثلا:

Copy code

```
"word" → count
"apple" → 57
"banana" → 34
```

2. **پایگاه‌داده‌های Key-Value** برای مدیریت داده‌های عظیم و سریع:
    - مثال‌ها:
        - **Redis**
        - **Amazon DynamoDB**
        - **Cassandra** (در اصل wide-column اما مبتنی بر key-value)
        - **Riak**
        - **LevelDB**

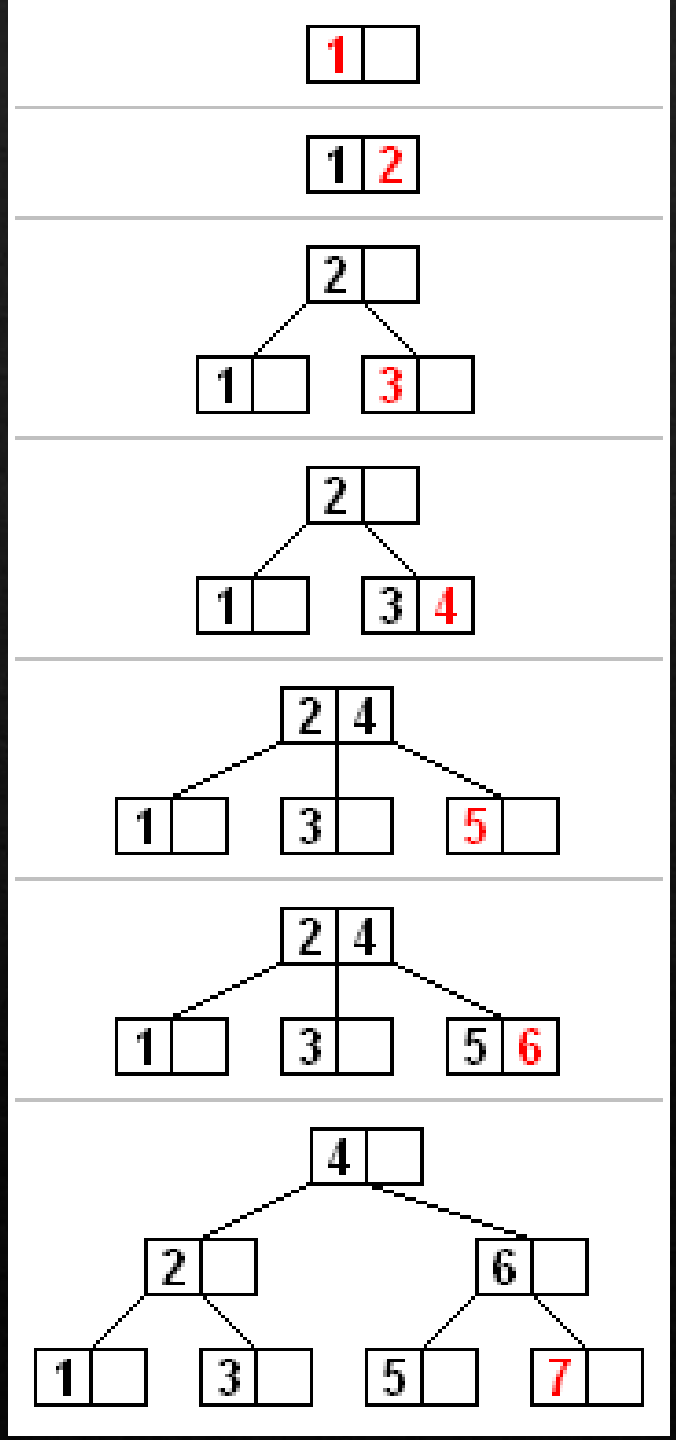
B-Tree vs LSM-Tree

DATABASE STORAGE ENGINES

B-TREE

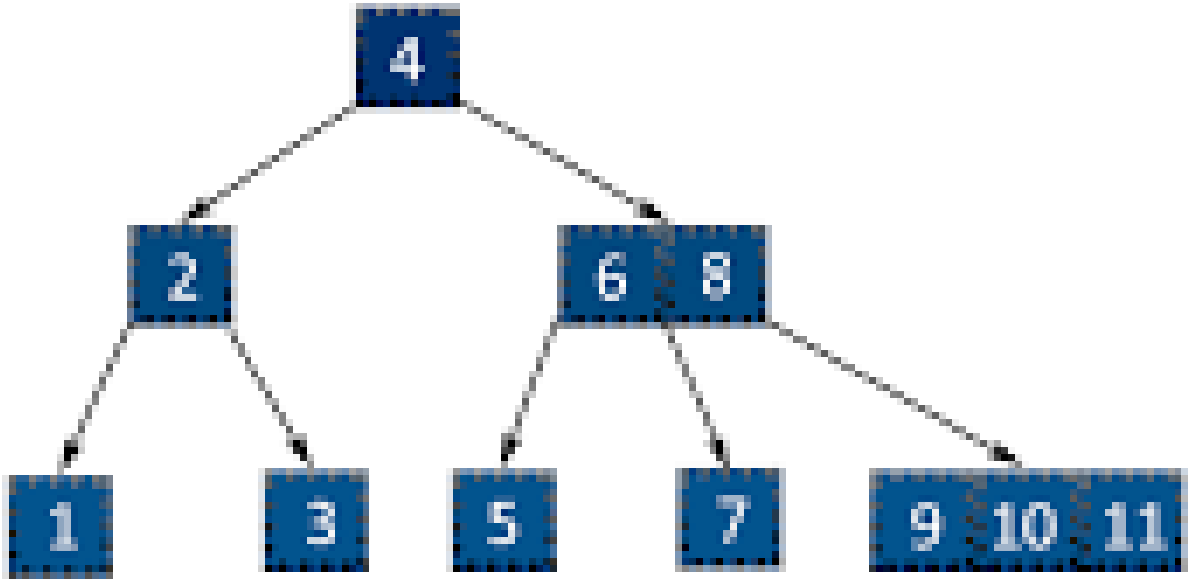
LSM TREE

A B-tree insertion example with each iteration. The nodes of this B-tree have at most 3 children (Knuth order 3).

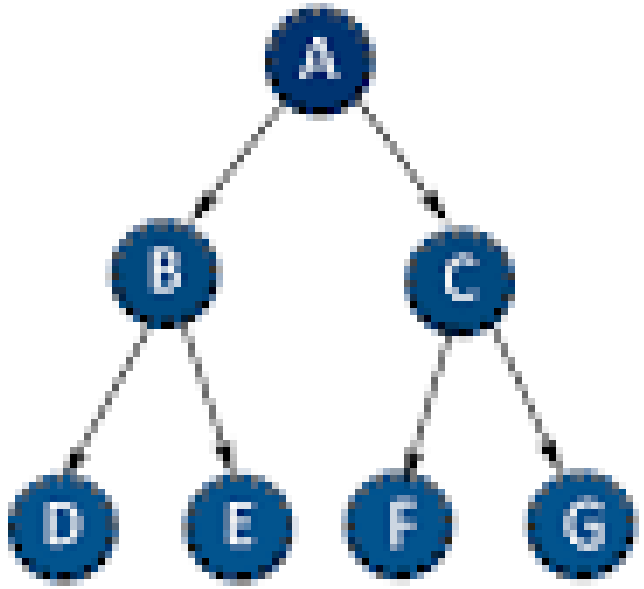


In computer science, a B-tree is a self-balancing tree data structure that maintains sorted data and allows searches, sequential access, insertions, and deletions in logarithmic time. The B-tree generalizes the binary search tree, allowing nodes to have more than two children.

B-TREE



BINARY TREE



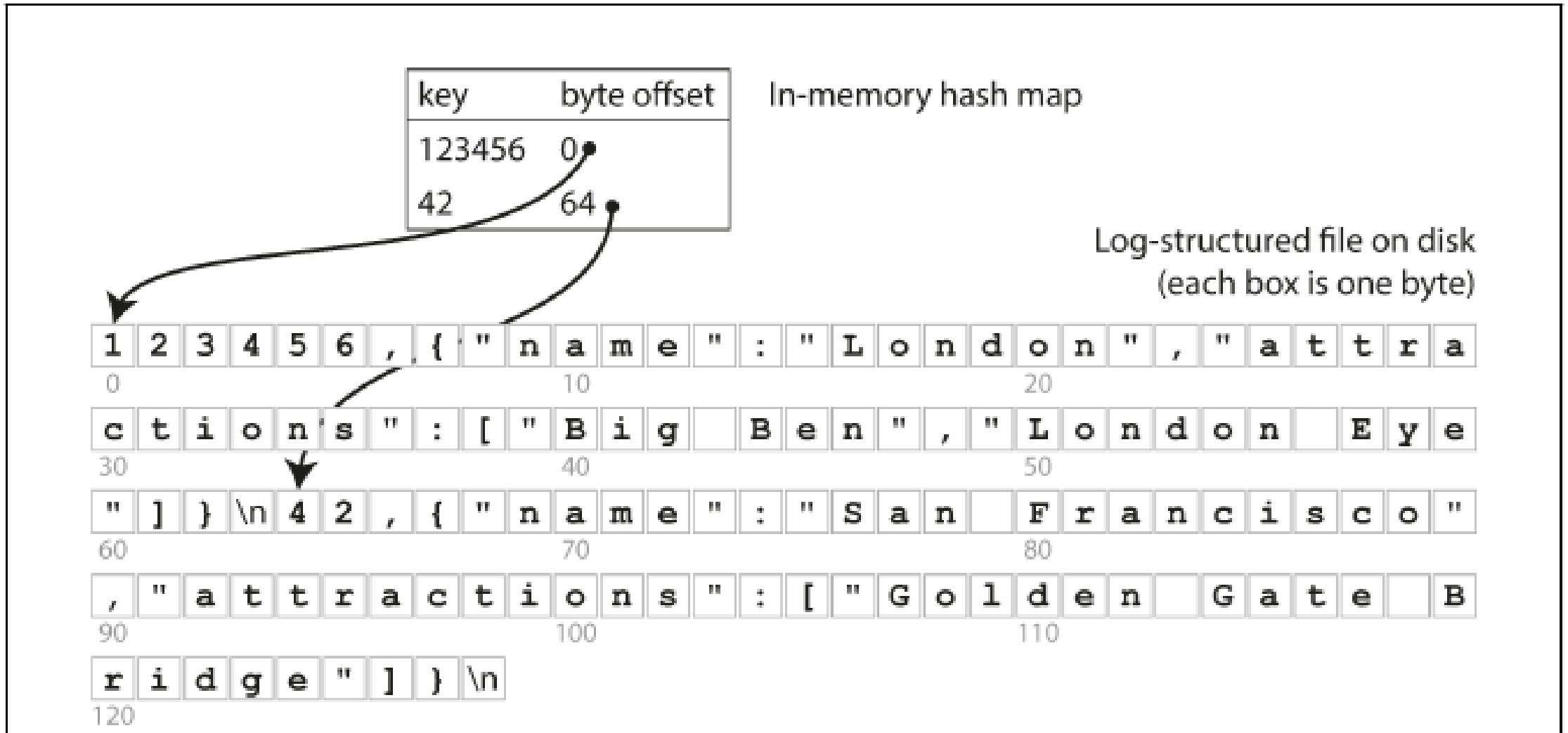


Figure 3-1. Storing a log of key-value pairs in a CSV-like format, indexed with an in-memory hash map.

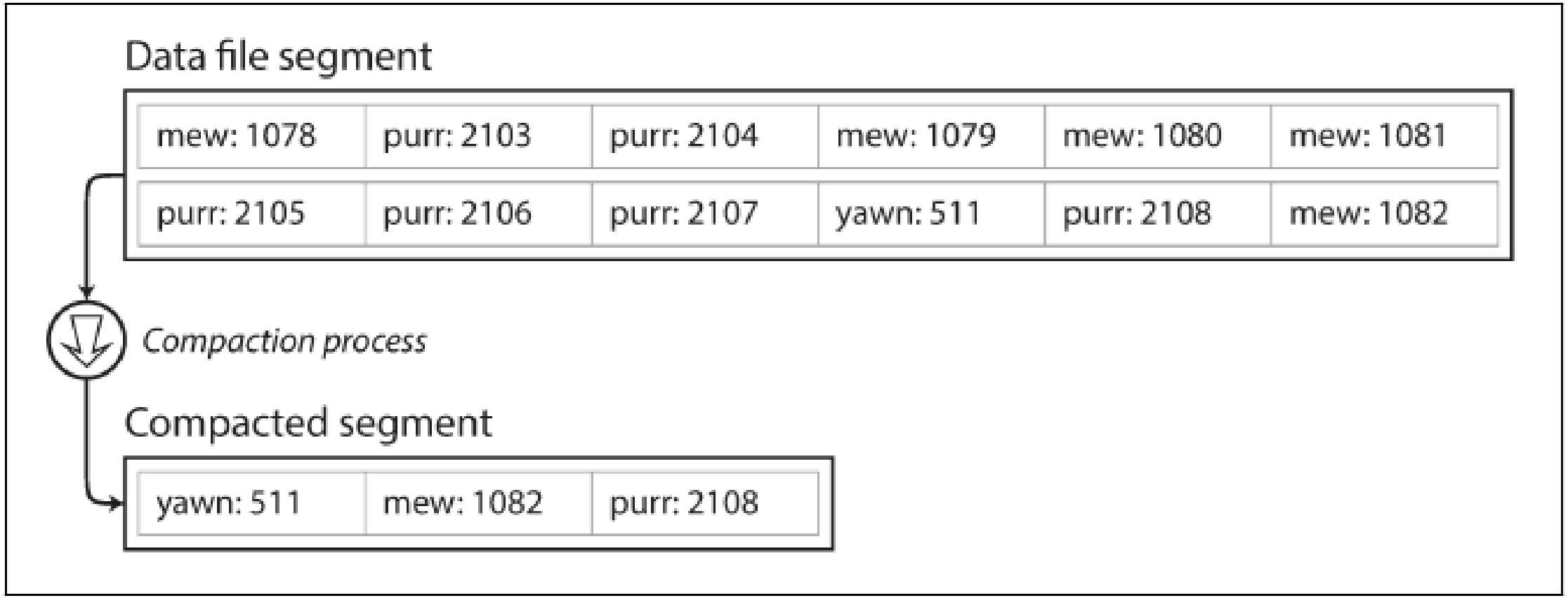


Figure 3-2. Compaction of a key-value update log (counting the number of times each cat video was played), retaining only the most recent value for each key.

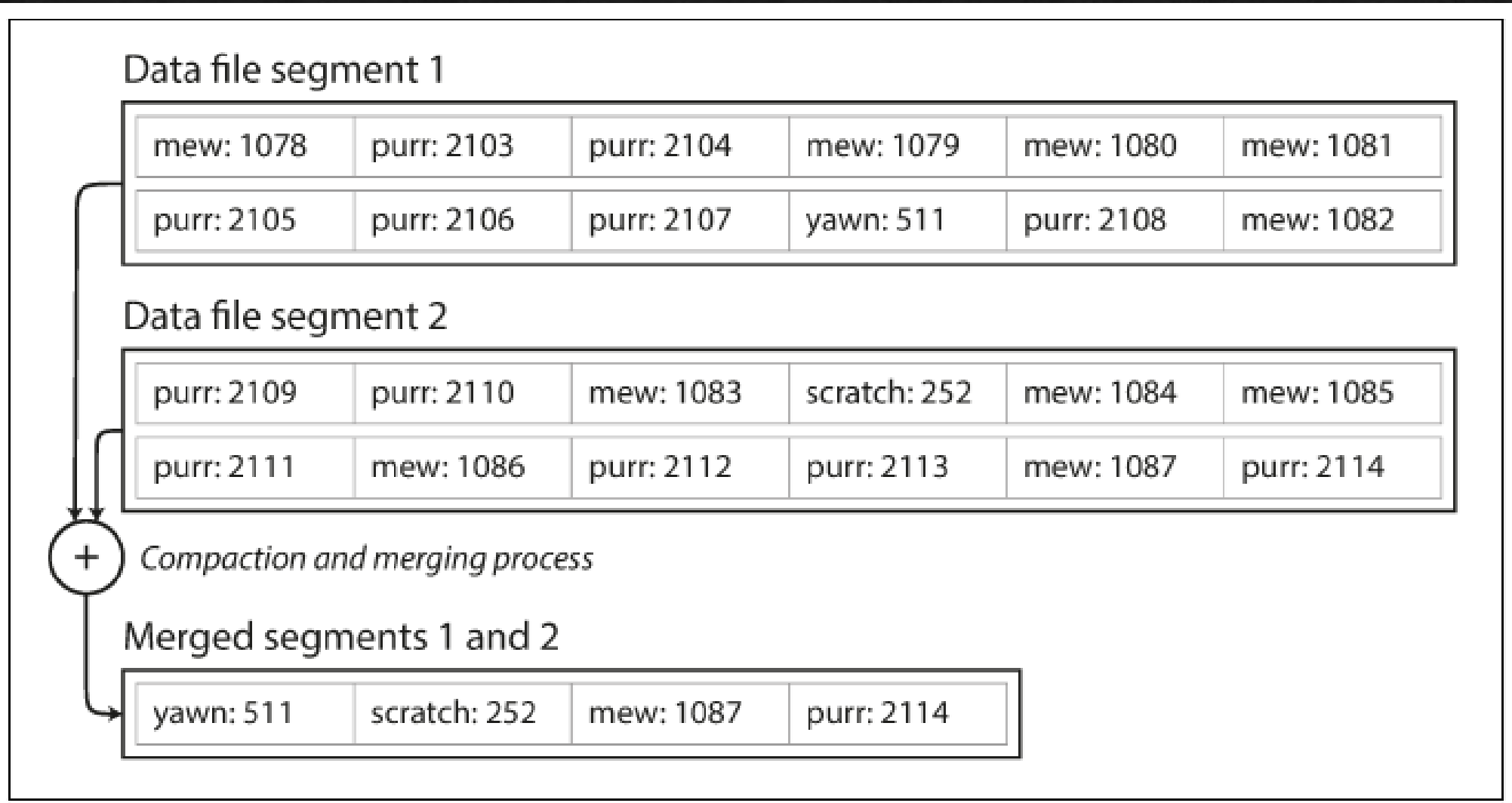


Figure 3-3. Performing compaction and segment merging simultaneously.

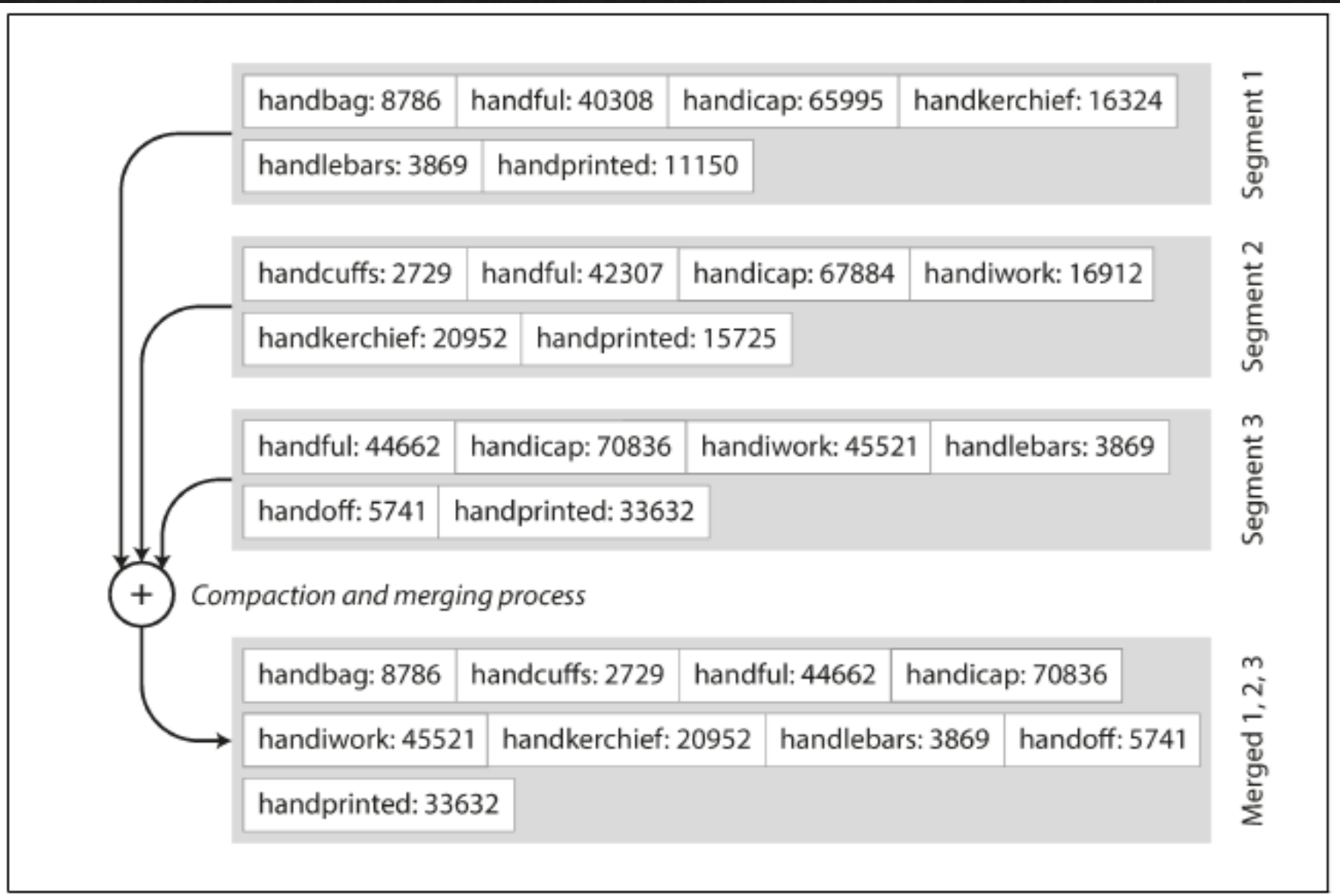


Figure 3-4. Merging several SSTable segments, retaining only the most recent value for each key.

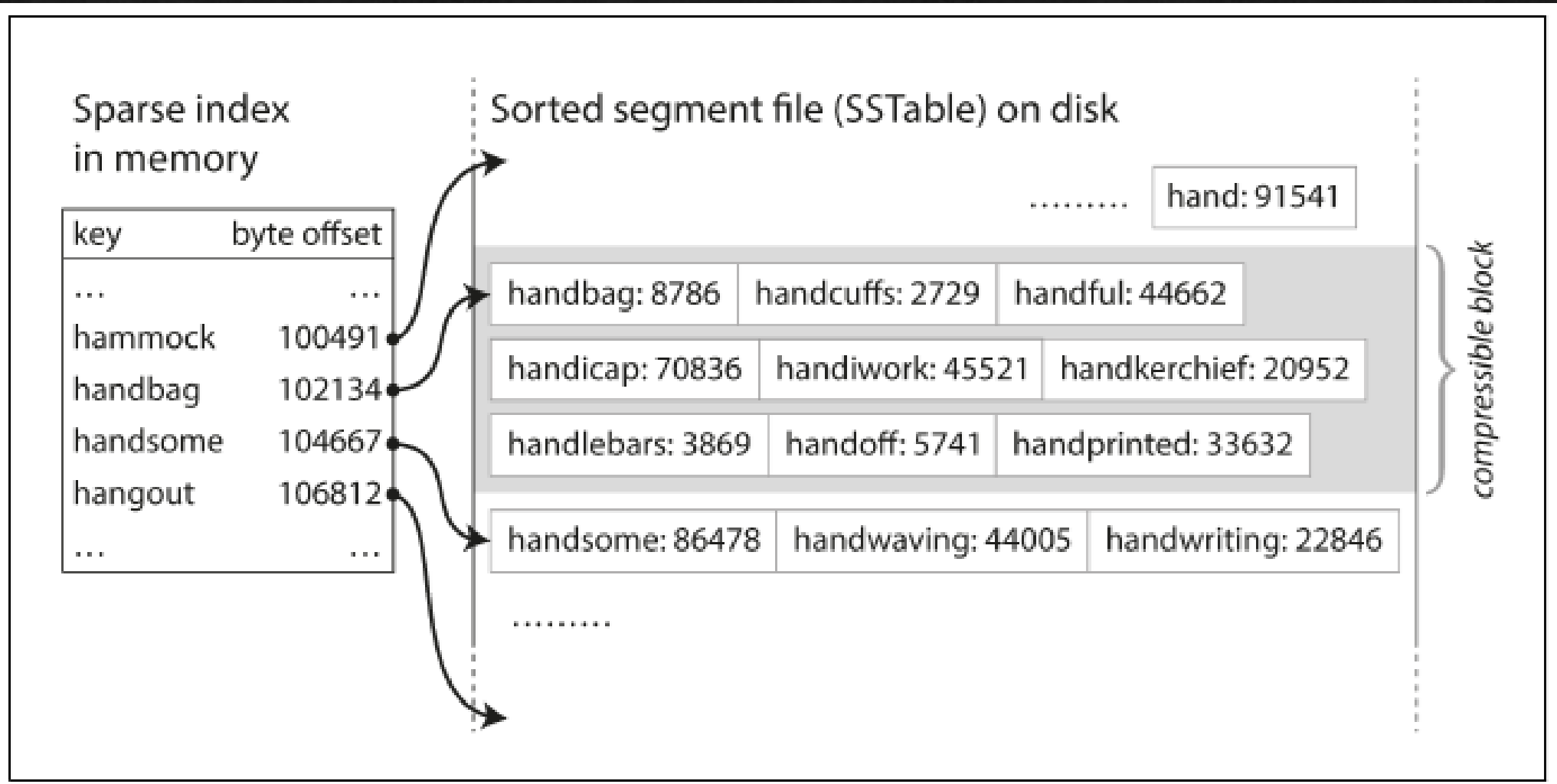
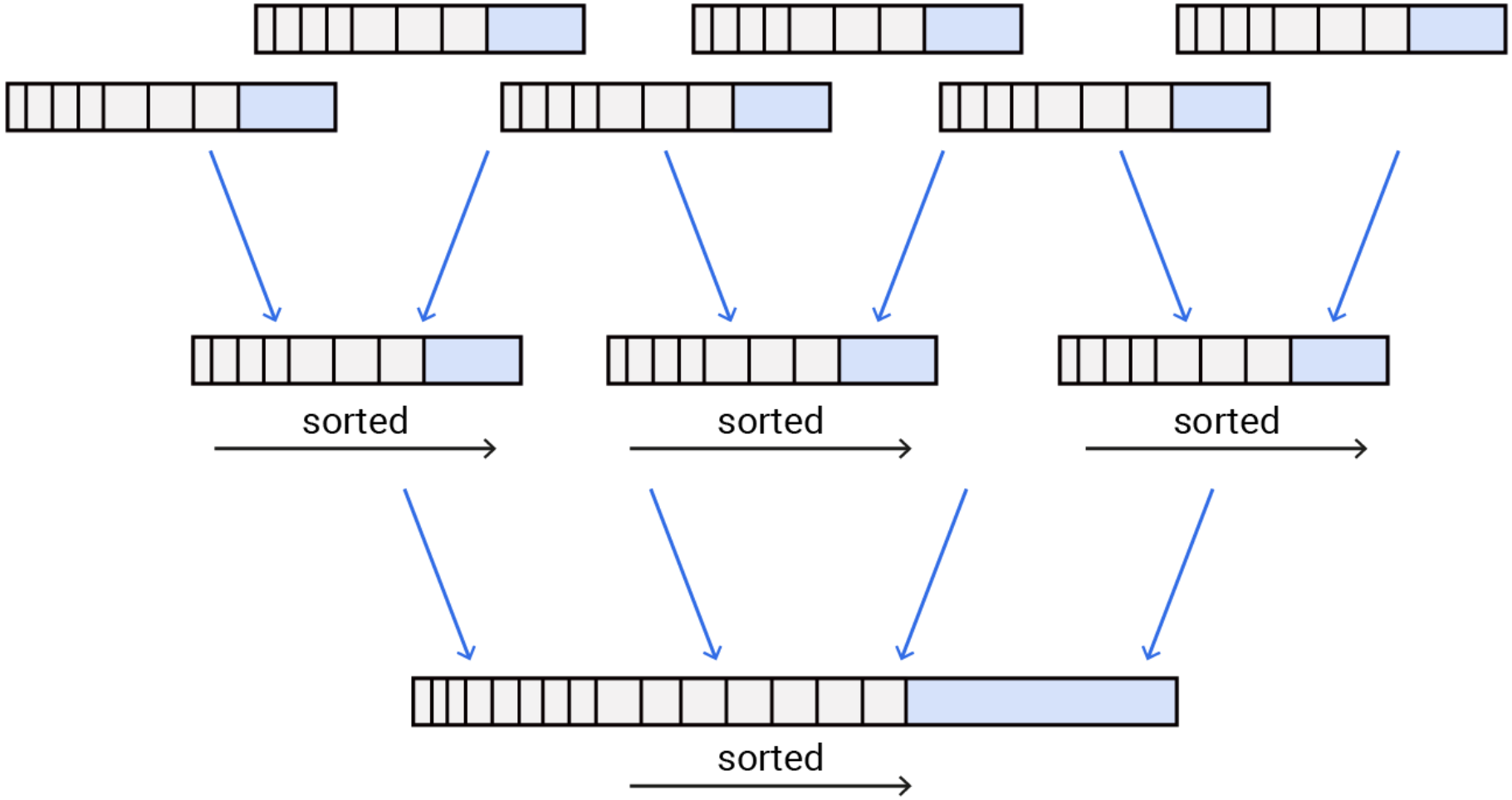
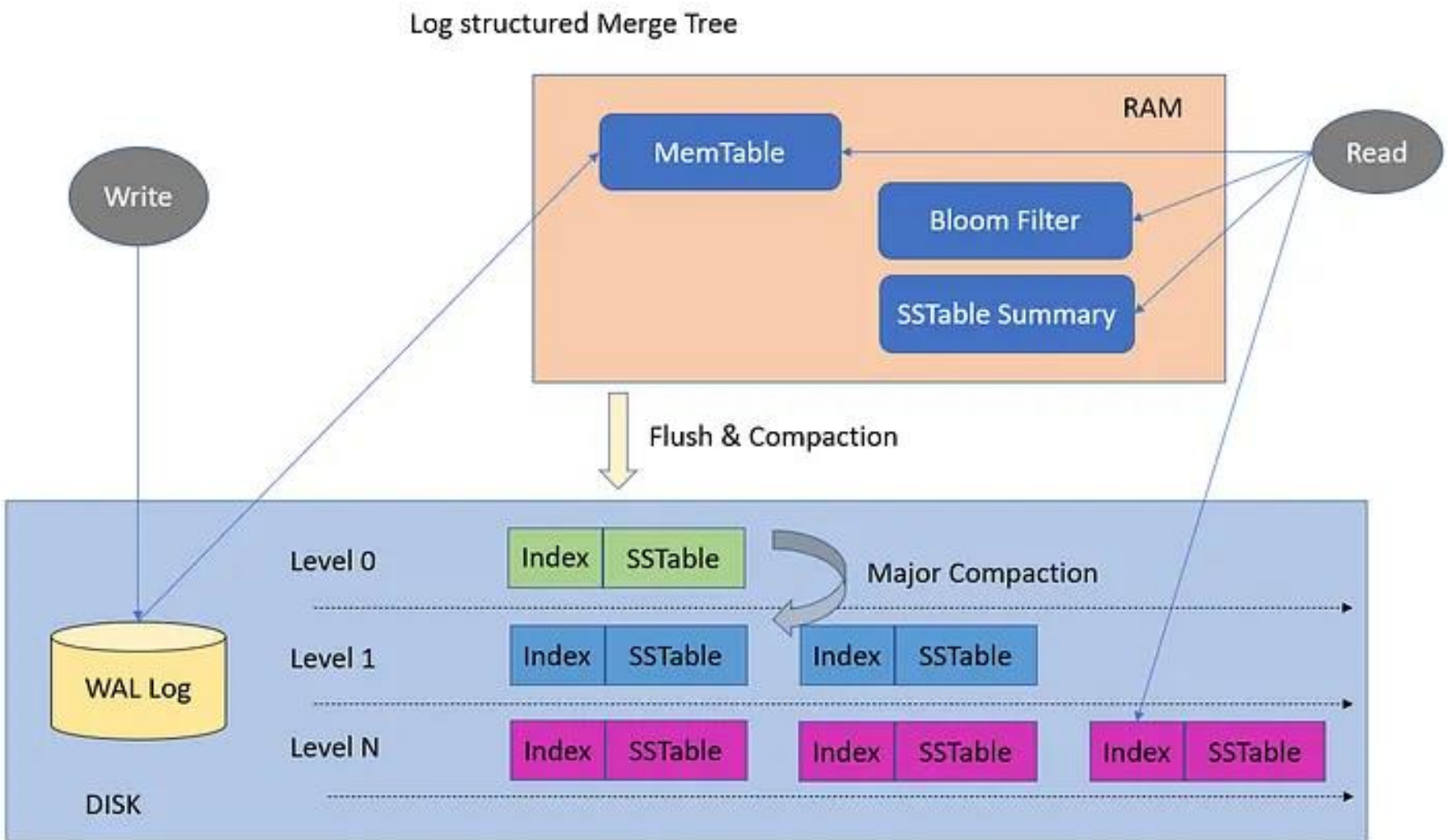


Figure 3-5. An SSTable with an in-memory index.

- An LSM (log-structured merge) tree is a data structure used in modern databases for efficient handling of high-volume writes by buffering data in memory before periodically flushing it to disk



LSM-tree


Log structured Merge Tree
RAM
MemTable
Bloom Filter
SSTable Summary
Read
Write
Flush & Compaction
Level 0
Index
SSTable
Major Compaction
Level 1
Index
SSTable
Index
SSTable
WAL Log
Level N
Index
SSTable
Index
SSTable
Index
SSTable
DISK

Diverse range of LSM use cases