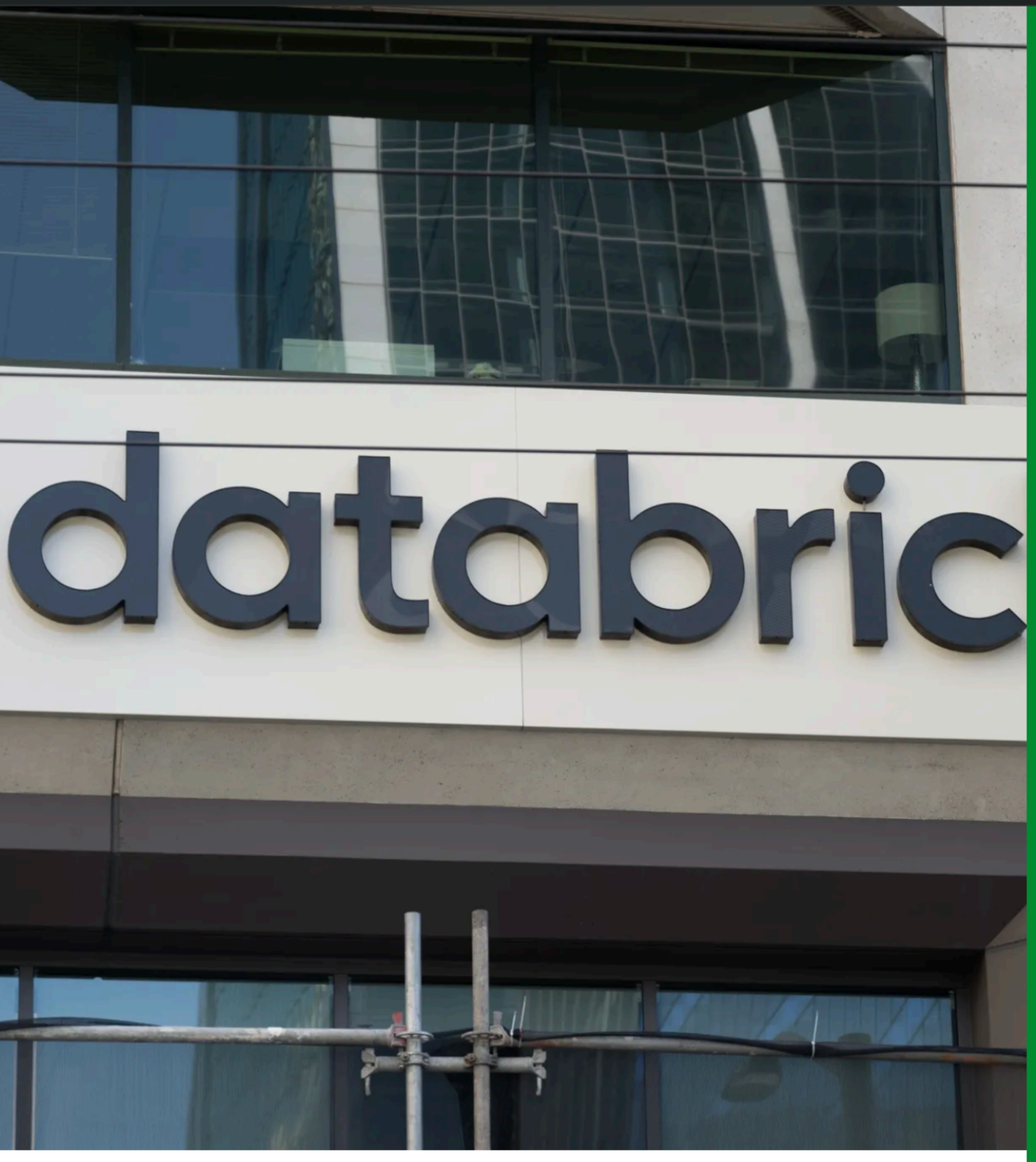


IMAGE CREDITS: SMITH COLLECTION/GADO / GETTY IMAGES

AI

Databricks to buy open source database startup Neon for $1B

Ram Iyer — 5:07 AM PDT · May 14, 2025

DBMS?

Architecture

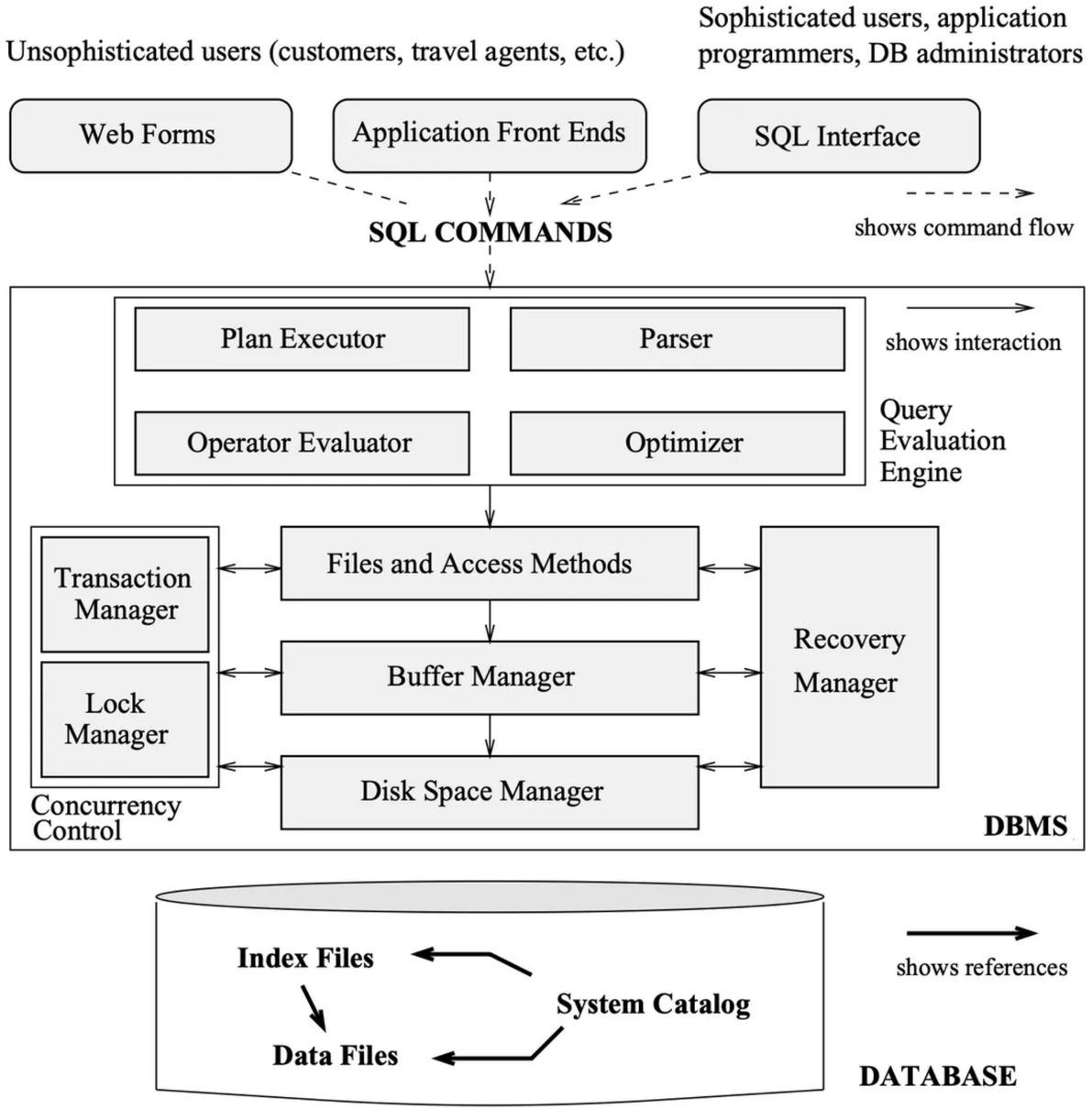
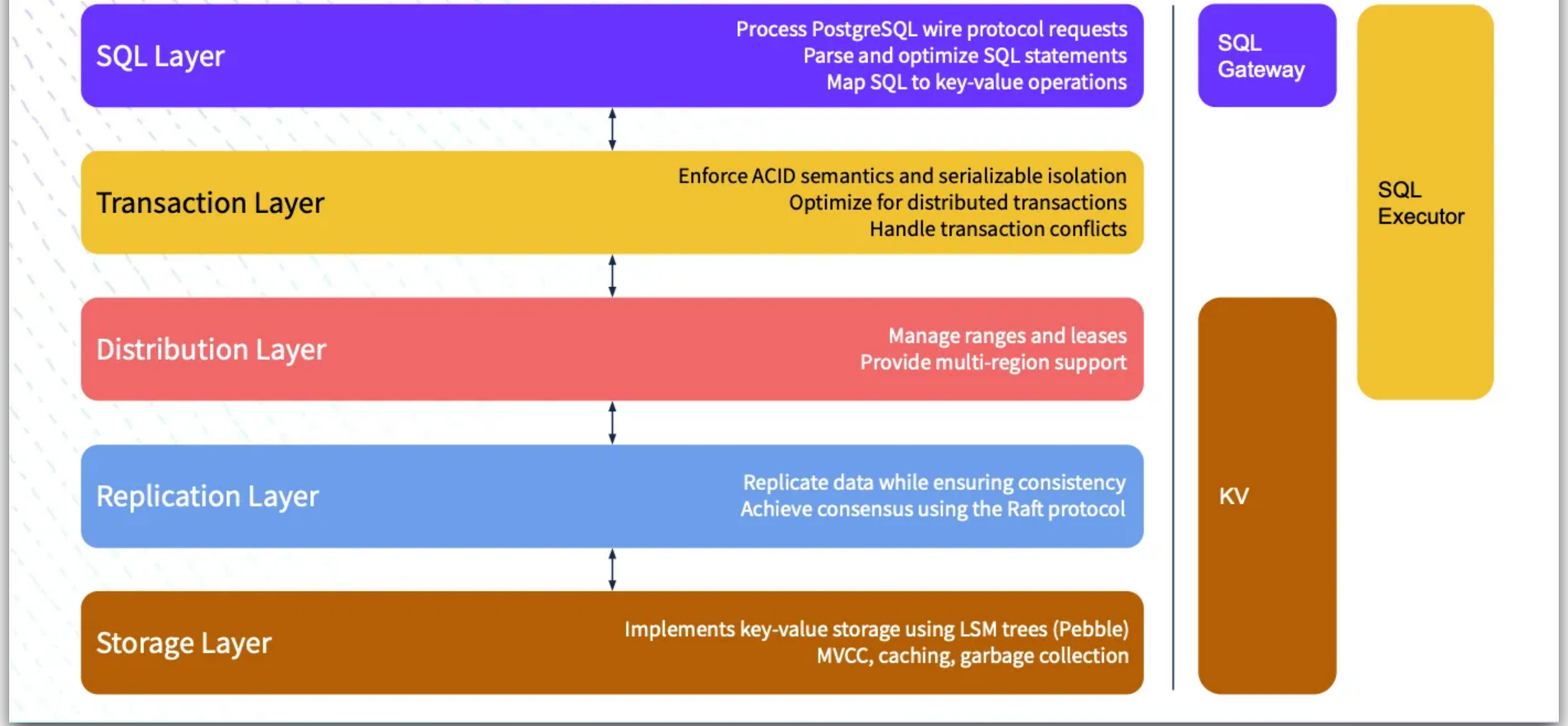
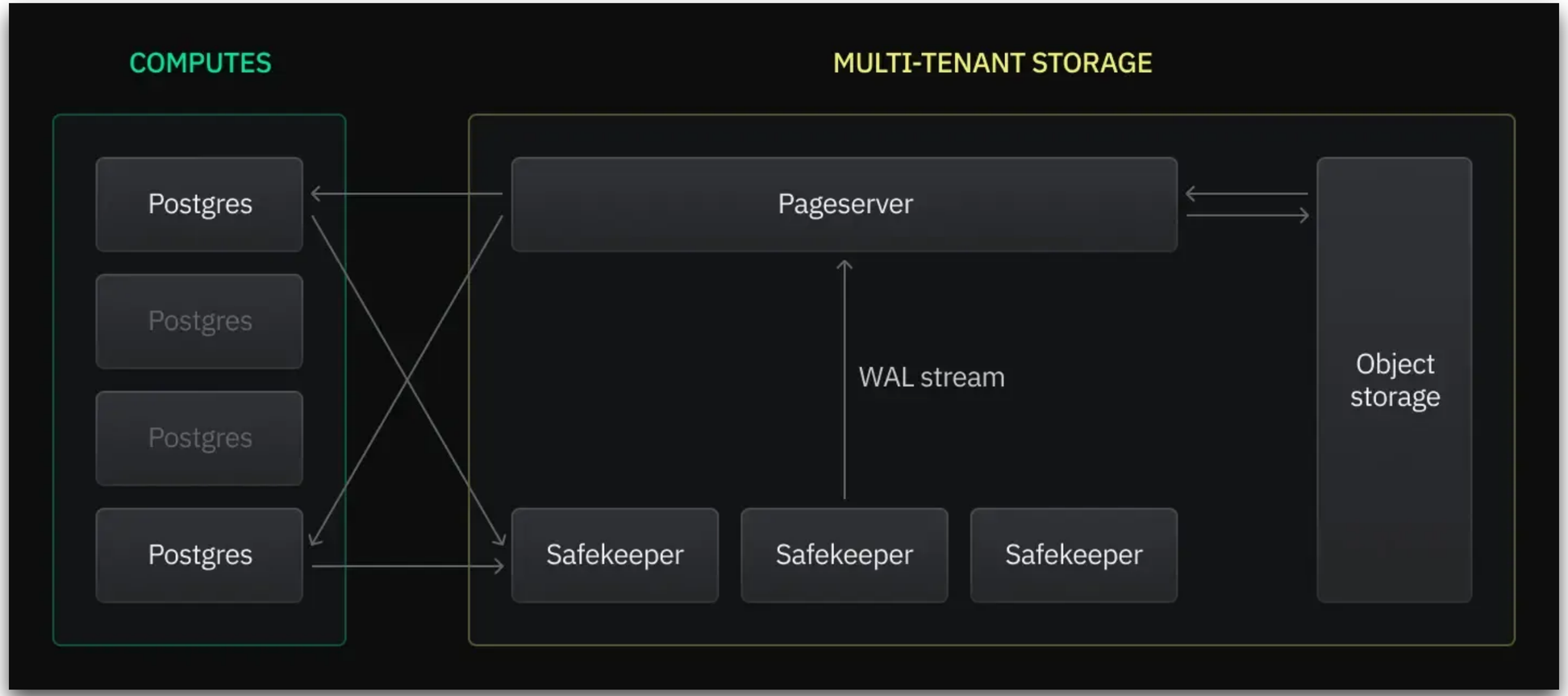
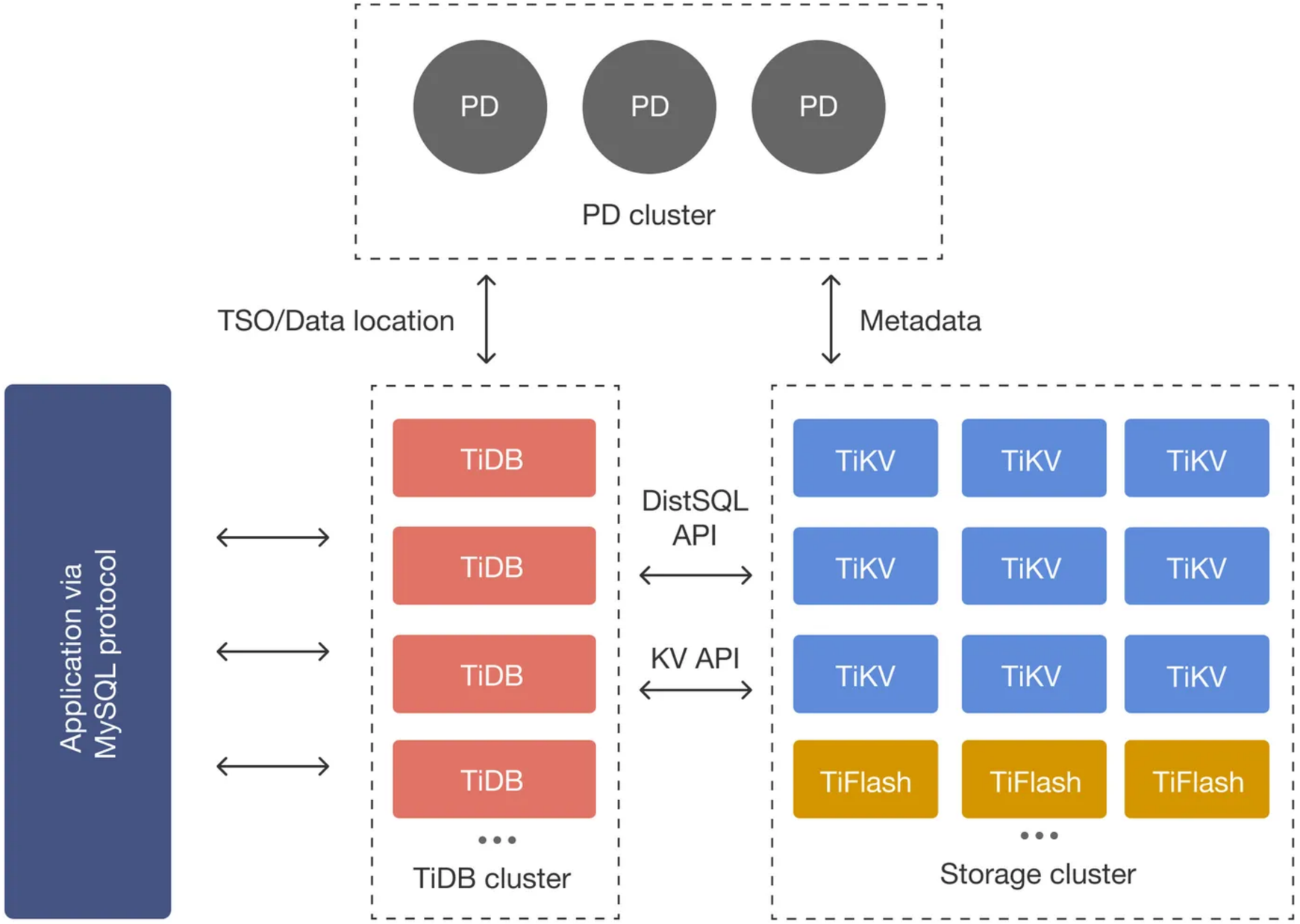


Figure 1.3 Architecture of a DBMS

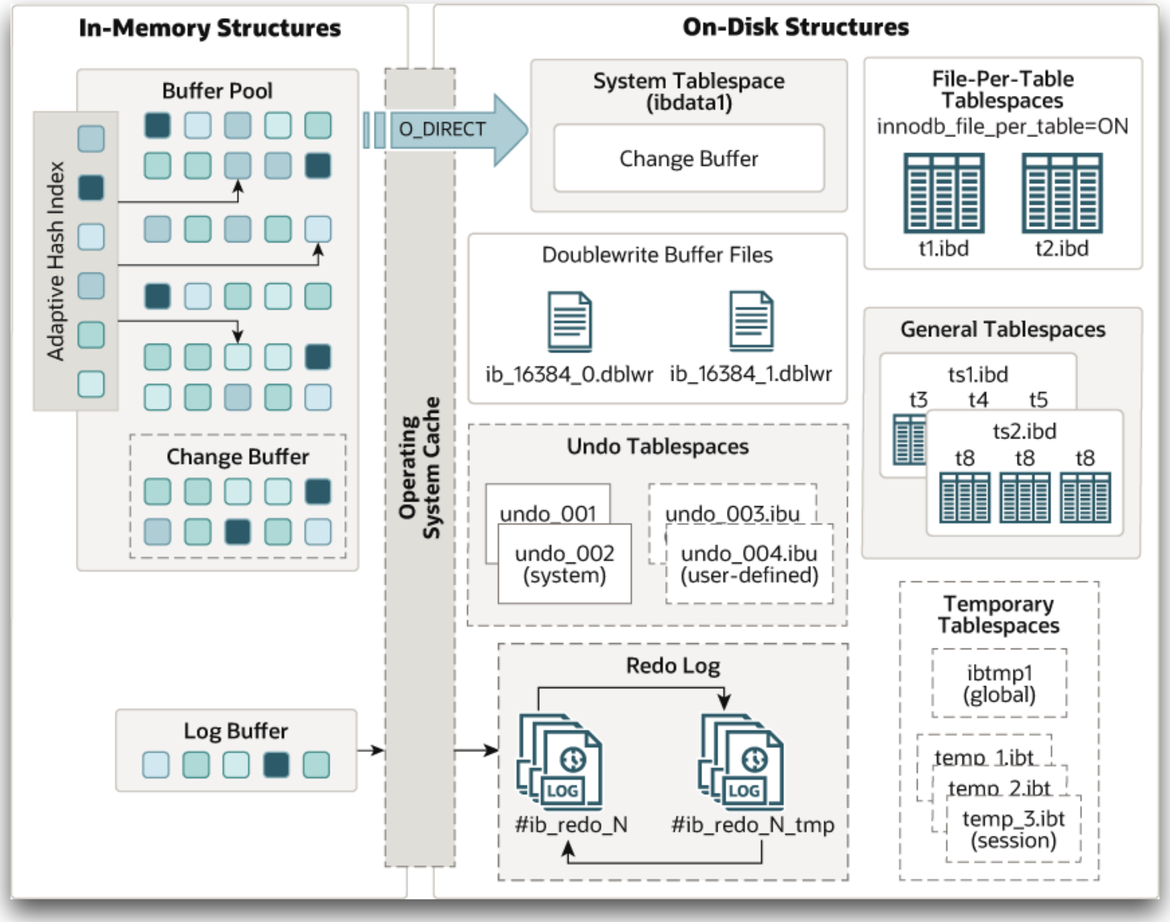
CockroachDB Software Stack

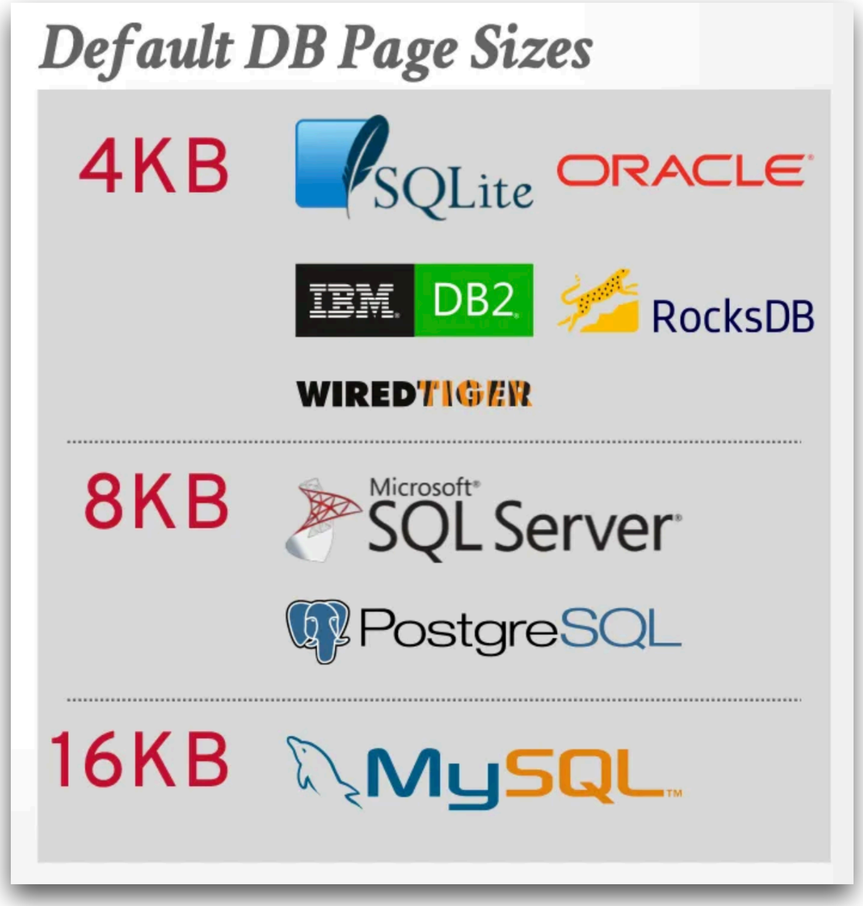


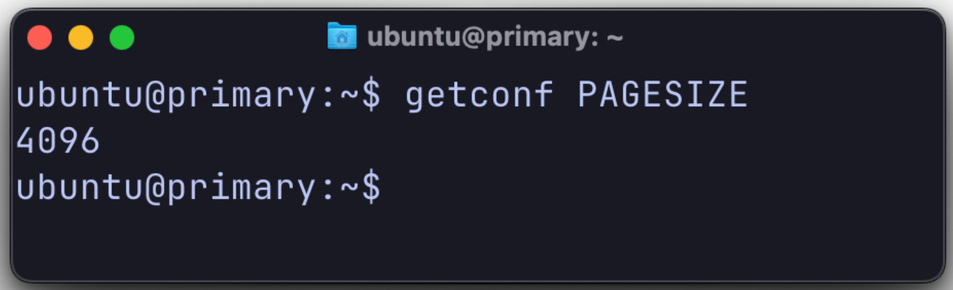

COMPUTES
MULTI-TENANT STORAGE
Postgres
Postgres
Postgres
Postgres
Pageserver
WAL stream
Object storage
Safekeeper
Safekeeper
Safekeeper

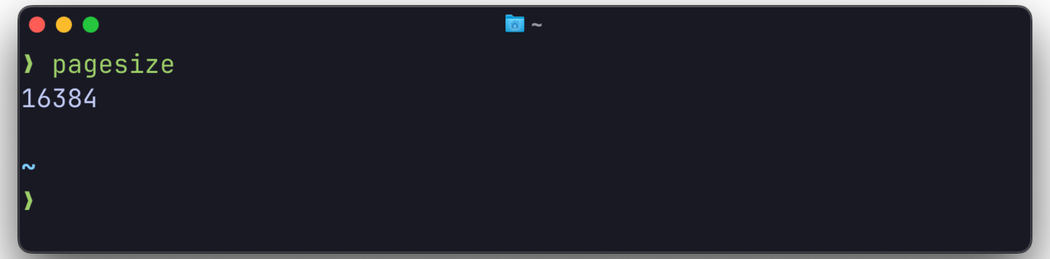

PD
PD
PD
PD cluster
TSO/Data location
Metadata
Application via MySQL protocol
TiDB
TiDB
TiDB
TiDB
TiDB cluster
DistSQL API
KV API
TiKV
TiKV
TiKV
TiKV
TiKV
TiKV
TiKV
TiKV
TiKV
TiFlash
TiFlash
TiFlash
Storage cluster

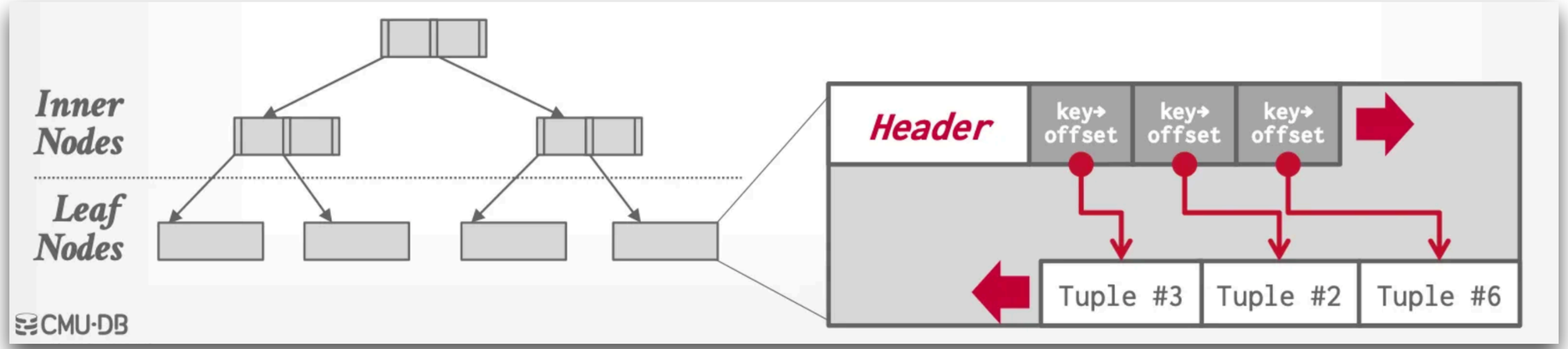
Storage


In-Memory Structures
On-Disk Structures
Buffer Pool
Adaptive Hash Index
Change Buffer
Log Buffer
O_DIRECT
Operating System Cache
System Tablespace (ibdata1)
Change Buffer
Doublewrite Buffer Files
ib_16384_0.dblwr
ib_16384_1.dblwr
Undo Tablespaces
undo_001
undo_002 (system)
undo_003.ibu
undo_004.ibu (user-defined)
Redo Log
LOG
#ib_redo_N
#ib_redo_N_tmp
File-Per-Table Tablespaces
innodb_file_per_table=ON
t1.ibd
t2.ibd
General Tablespaces
ts1.ibd
t3
t4
t5
ts2.ibd
t8
t8
t8
Temporary Tablespaces
ibtmp1 (global)
temp_1.ibt
temp_2.ibt
temp_3.ibt (session)


Default DB Page Sizes
4KB
SQLite
ORACLE
IBM DB2
RocksDB
WIREDTIGER
8KB
Microsoft SQL Server
PostgreSQL
16KB
MySQL


ubuntu@primary: ~
ubuntu@primary:~$ getconf PAGESIZE
4096
ubuntu@primary:~$


~
❯ pagesize
16384
~
❯


Inner Nodes
Leaf Nodes
CMU·DB
Header
key→ offset
key→ offset
key→ offset
Tuple #3
Tuple #2
Tuple #6

Example of Hash File Organization

bucket 0

bucket 1

15151	Mozart	Music	40000

bucket 2

32343	El Said	History	80000
58583	Califieri	History	60000

bucket 3

22222	Einstein	Physics	95000
33456	Gold	Physics	87000
98345	Kim	Elec. Eng.	80000

bucket 4

12121	Wu	Finance	90000
76543	Singh	Finance	80000

bucket 5

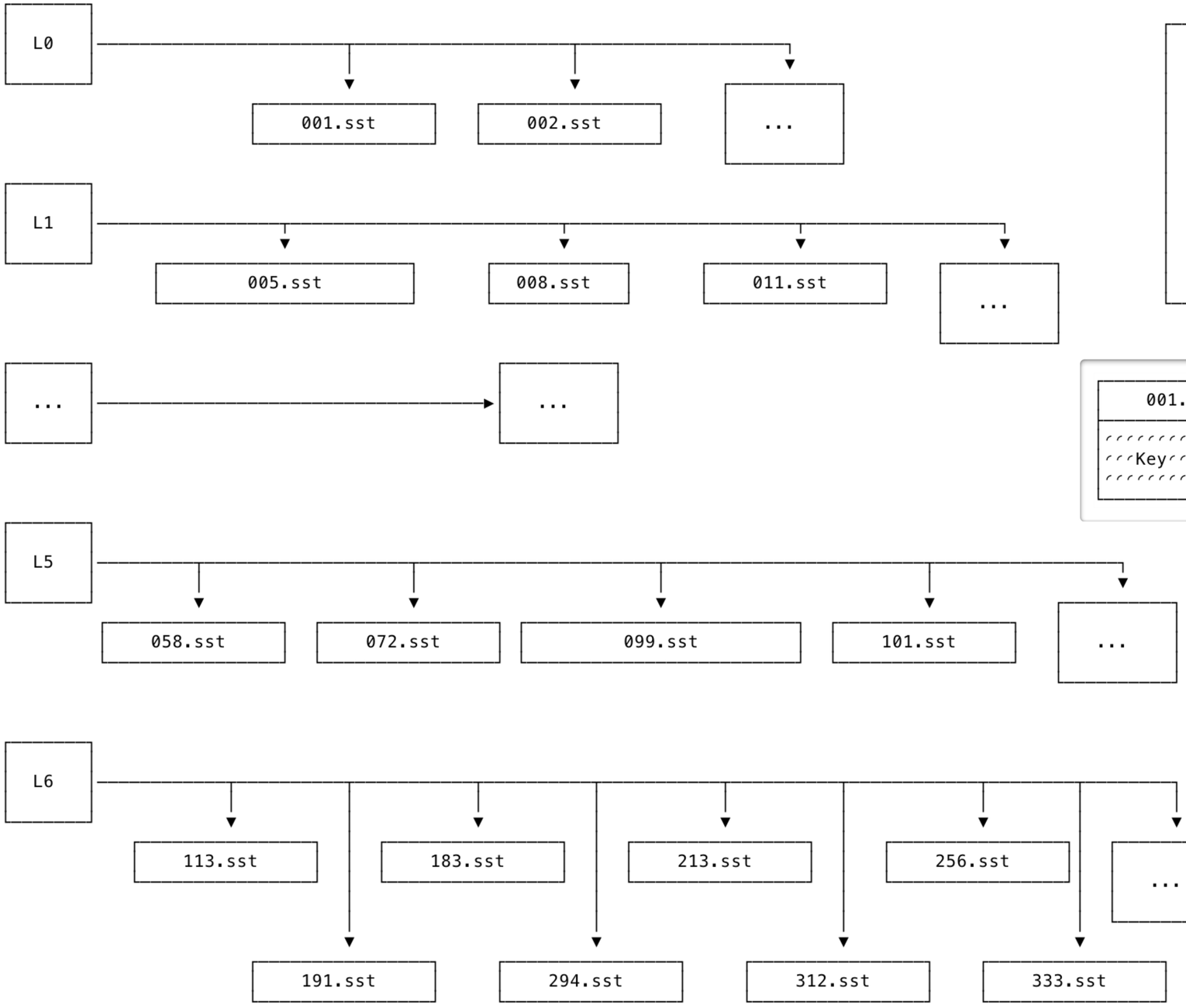
76766	Crick	Biology	72000

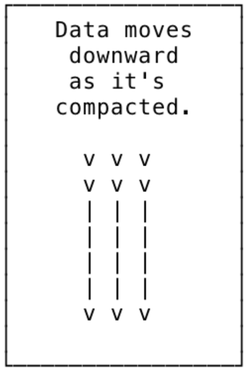
bucket 6

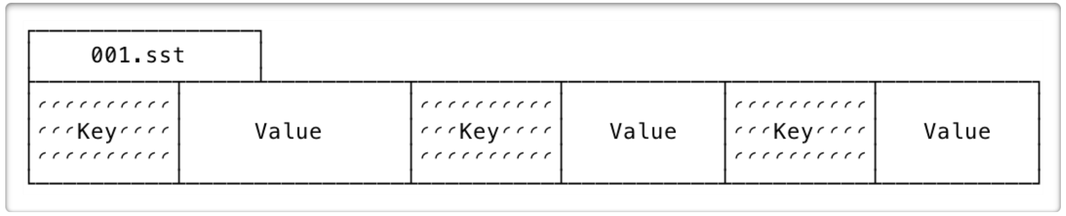
10101	Srinivasan	Comp. Sci.	65000
45565	Katz	Comp. Sci.	75000
83821	Brandt	Comp. Sci.	92000

bucket 7

Hash file organization of *instructor* file, using *dept_name* as key.


L0
001.sst
002.sst
...
L1
005.sst
008.sst
011.sst
...
...
...
L5
058.sst
072.sst
099.sst
101.sst
...
L6
113.sst
183.sst
213.sst
256.sst
...
191.sst
294.sst
312.sst
333.sst

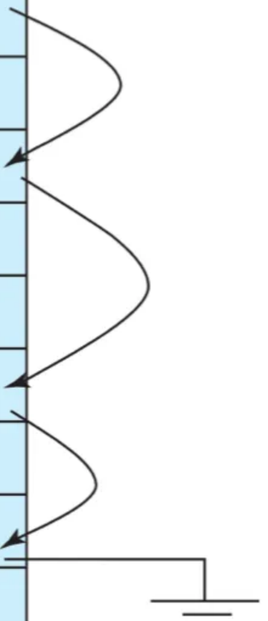

Data moves
downward
as it's
compacted.

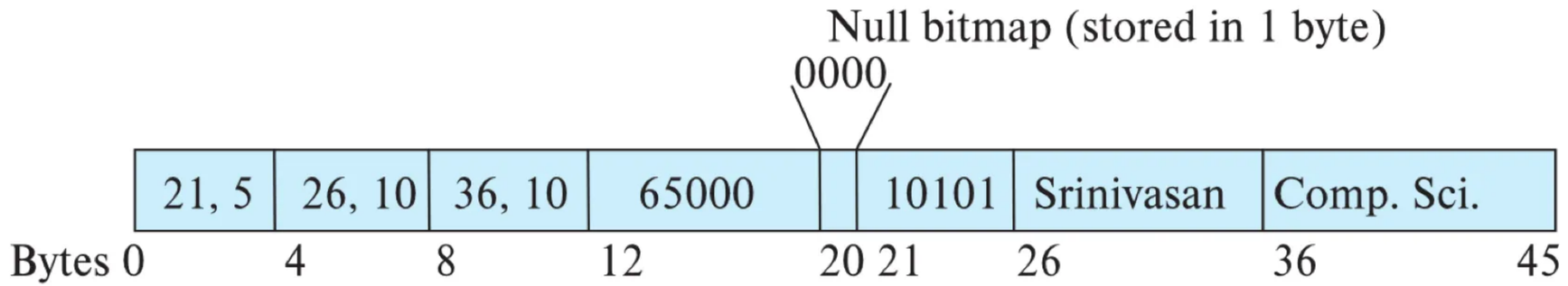

001.sst
Key
Value
Key
Value
Key
Value

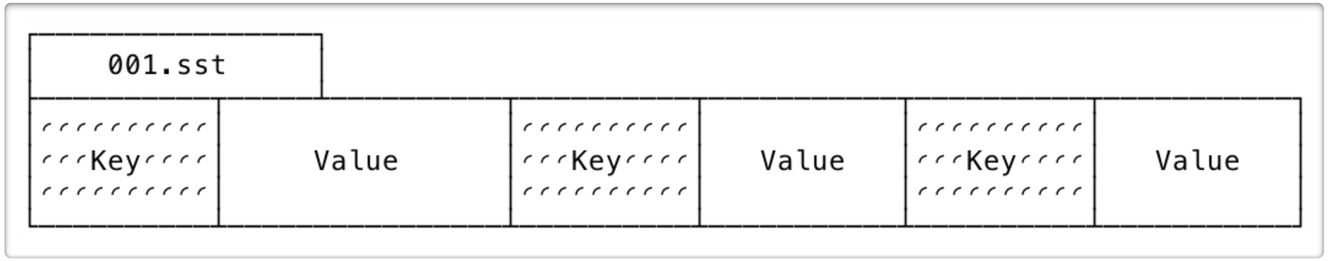
Page

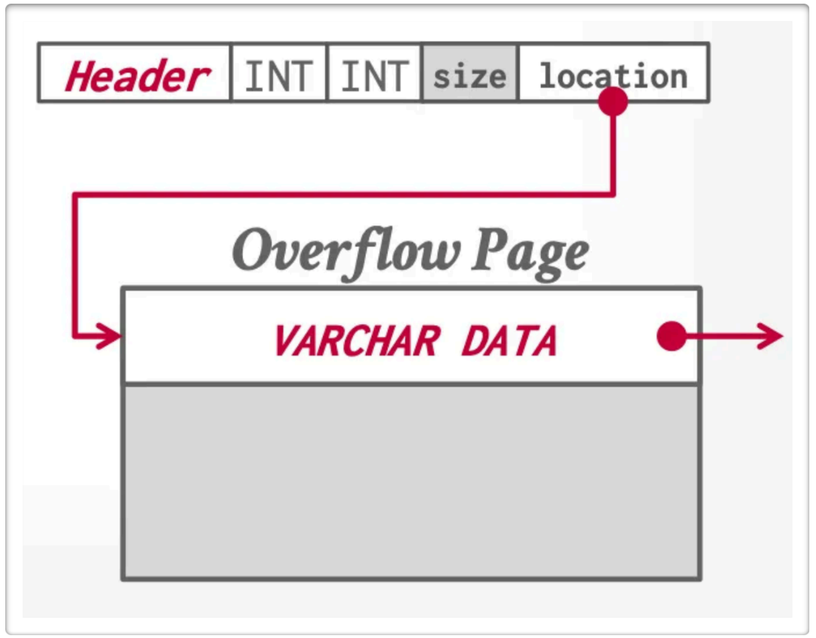
record 0	10101	Srinivasan	Comp. Sci.	65000
record 1	12121	Wu	Finance	90000
record 2	15151	Mozart	Music	40000
record 11	98345	Kim	Elec. Eng.	80000
record 4	32343	El Said	History	60000
record 5	33456	Gold	Physics	87000
record 6	45565	Katz	Comp. Sci.	75000
record 7	58583	Califieri	History	62000
record 8	76543	Singh	Finance	80000
record 9	76766	Crick	Biology	72000
record 10	83821	Brandt	Comp. Sci.	92000

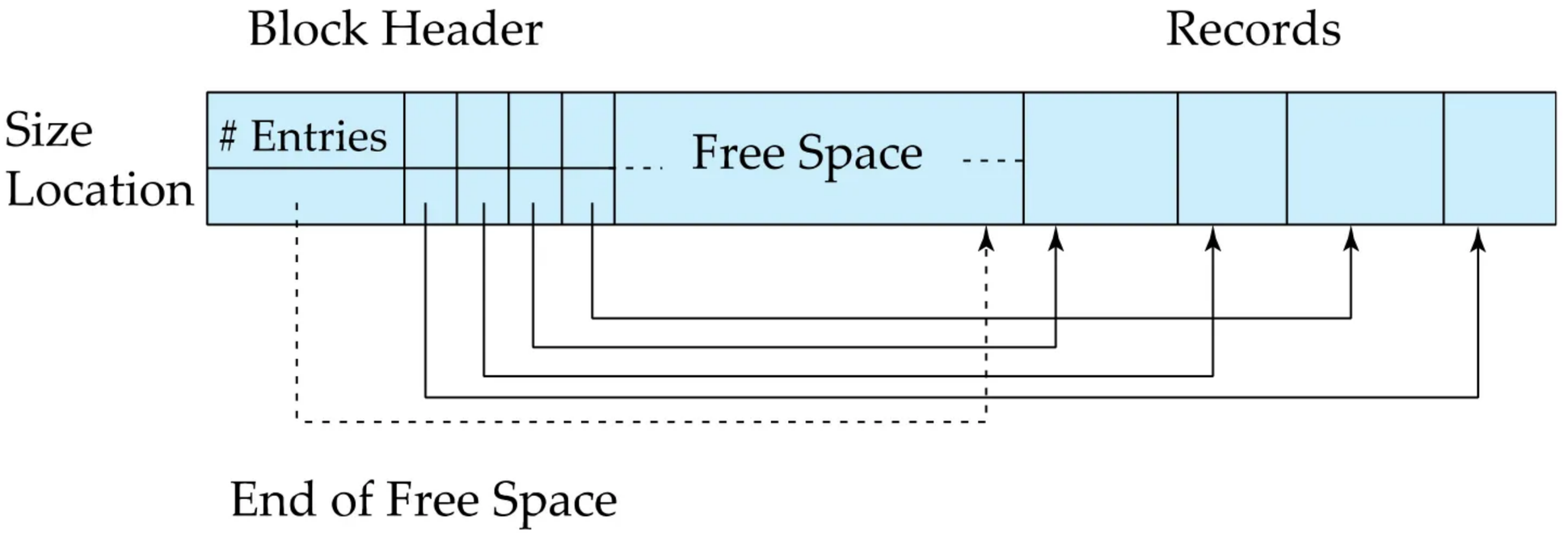
header				
record 0	10101	Srinivasan	Comp. Sci.	65000
record 1				
record 2	15151	Mozart	Music	40000
record 3	22222	Einstein	Physics	95000
record 4				
record 5	33456	Gold	Physics	87000
record 6				
record 7	58583	Califieri	History	62000
record 8	76543	Singh	Finance	80000
record 9	76766	Crick	Biology	72000
record 10	83821	Brandt	Comp. Sci.	92000
record 11	98345	Kim	Elec. Eng.	80000




Null bitmap (stored in 1 byte)
0000
21, 5
26, 10
36, 10
65000
10101
Srinivasan
Comp. Sci.
Bytes 0
4
8
12
20 21
26
36
45


001.sst
Key
Value
Key
Value
Key
Value


Header
INT
INT
size
location
Overflow Page
VARCHAR DATA


Block Header
Records
Size
Location
Entries
Free Space
End of Free Space

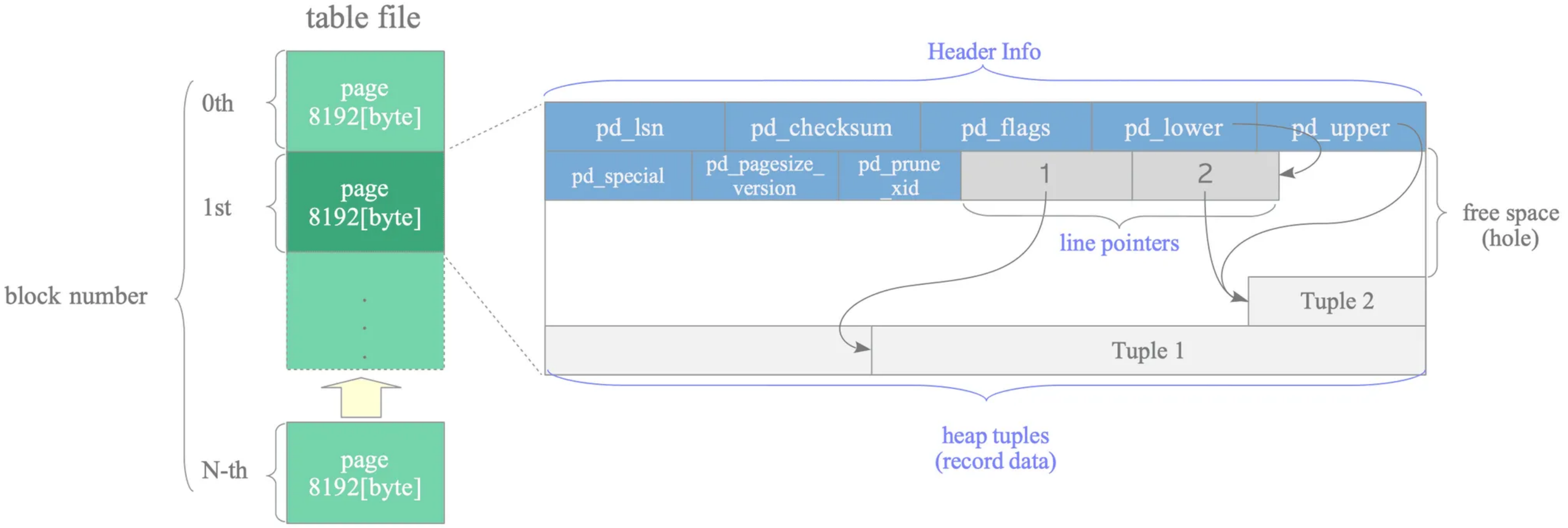
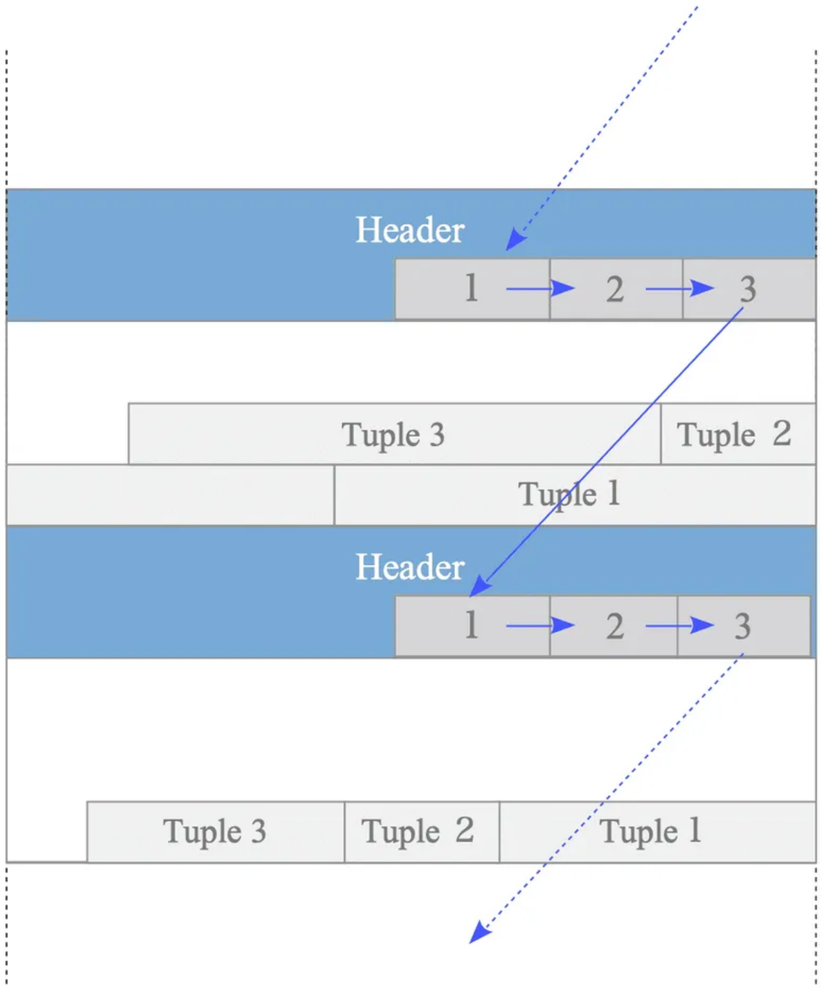


table file
0th
1st
N-th
block number
page
8192[byte]
page
8192[byte]
page
8192[byte]
Header Info
pd_lsn
pd_checksum
pd_flags
pd_lower
pd_upper
pd_special
pd_pagesize_
version
pd_prune
_xid
1
2
line pointers
free space
(hole)
Tuple 2
Tuple 1
heap tuples
(record data)

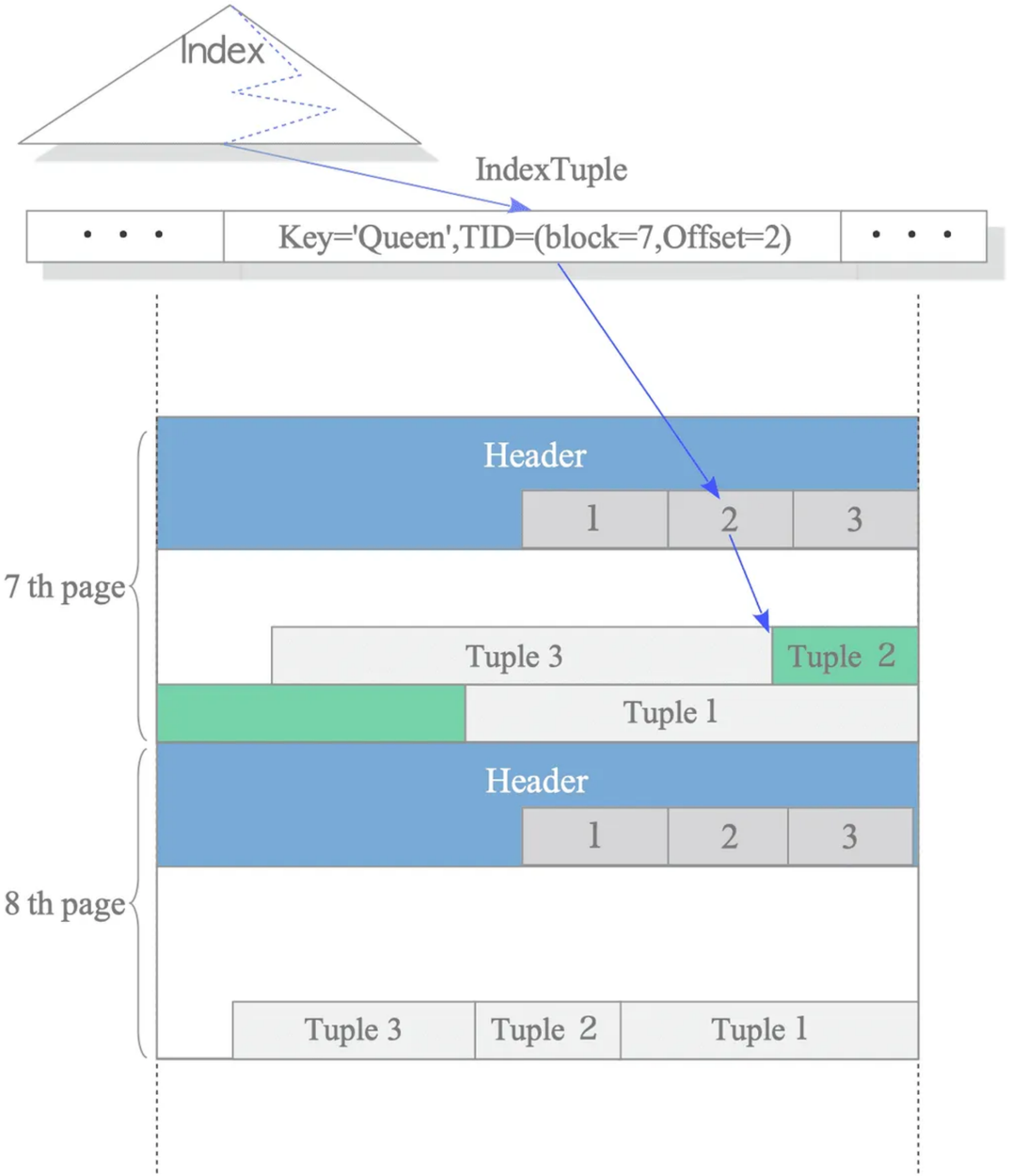
(a) Sequential Scan

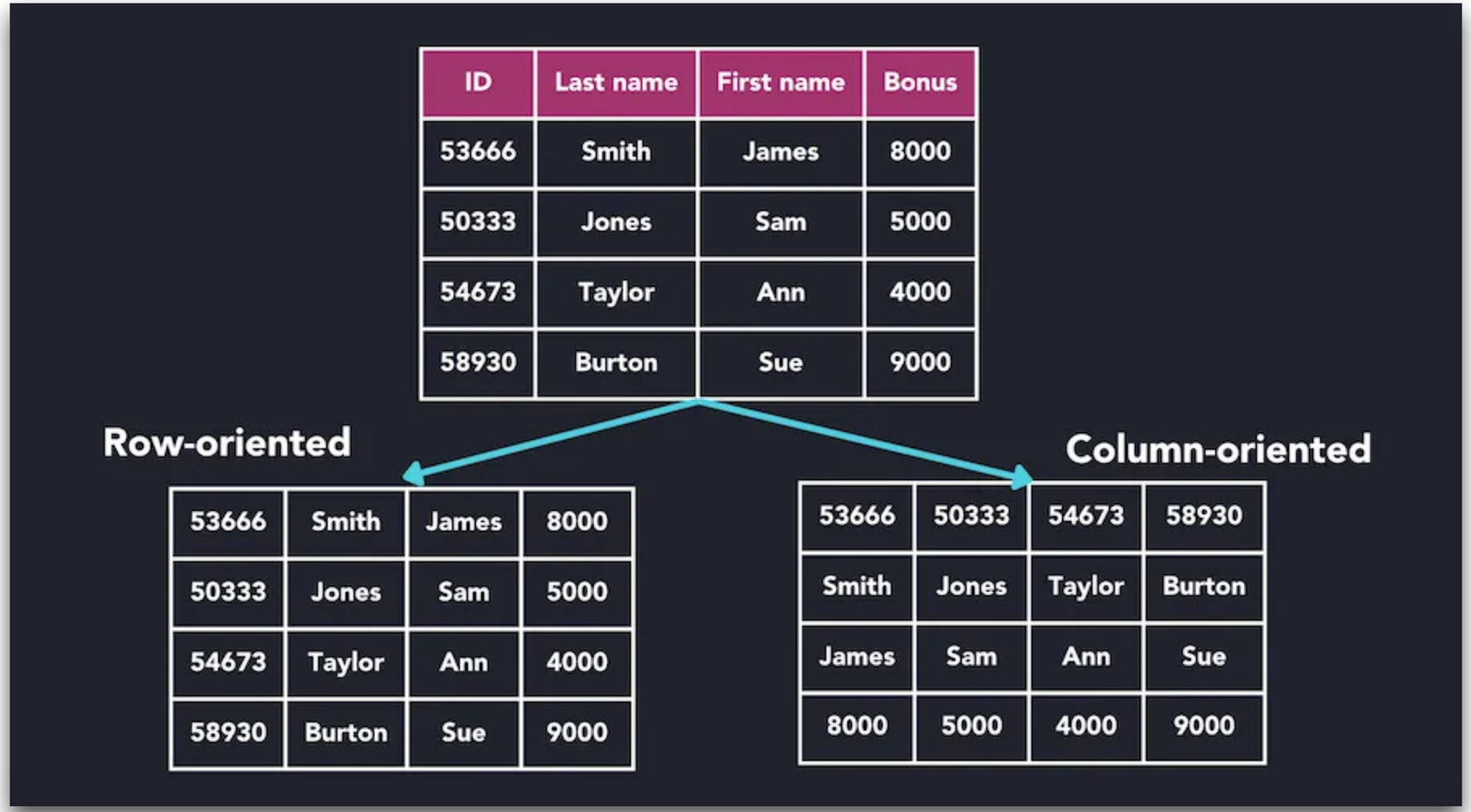
SELECT * FROM tbl;



(b) Index Scan

SELECT * FROM tbl WHERE col = 'Queen';




ID
Last name
First name
Bonus
53666
Smith
James
8000
50333
Jones
Sam
5000
54673
Taylor
Ann
4000
58930
Burton
Sue
9000
Row-oriented
53666
Smith
James
8000
50333
Jones
Sam
5000
54673
Taylor
Ann
4000
58930
Burton
Sue
9000
Column-oriented
53666
50333
54673
58930
Smith
Jones
Taylor
Burton
James
Sam
Ann
Sue
8000
5000
4000
9000

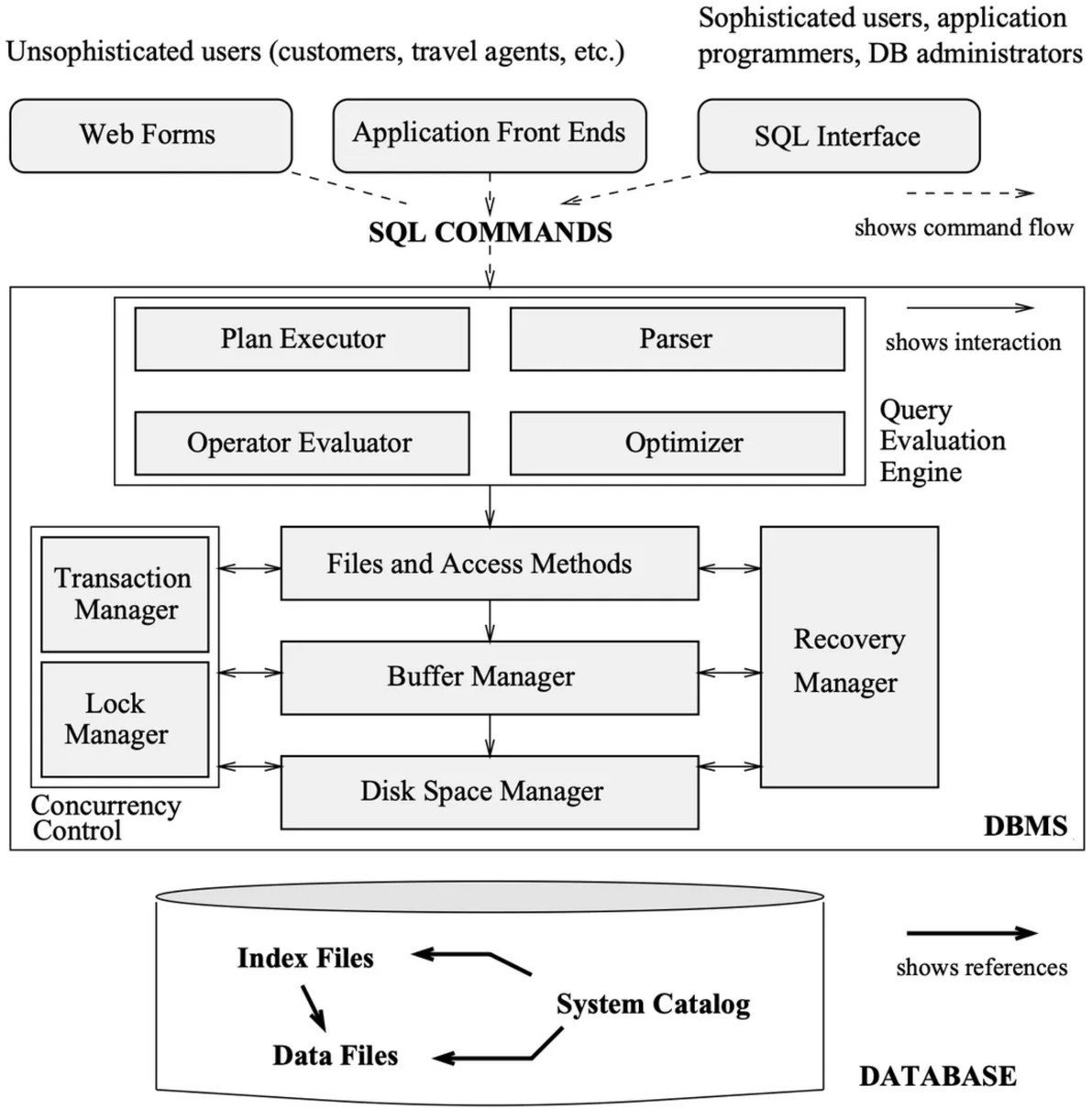
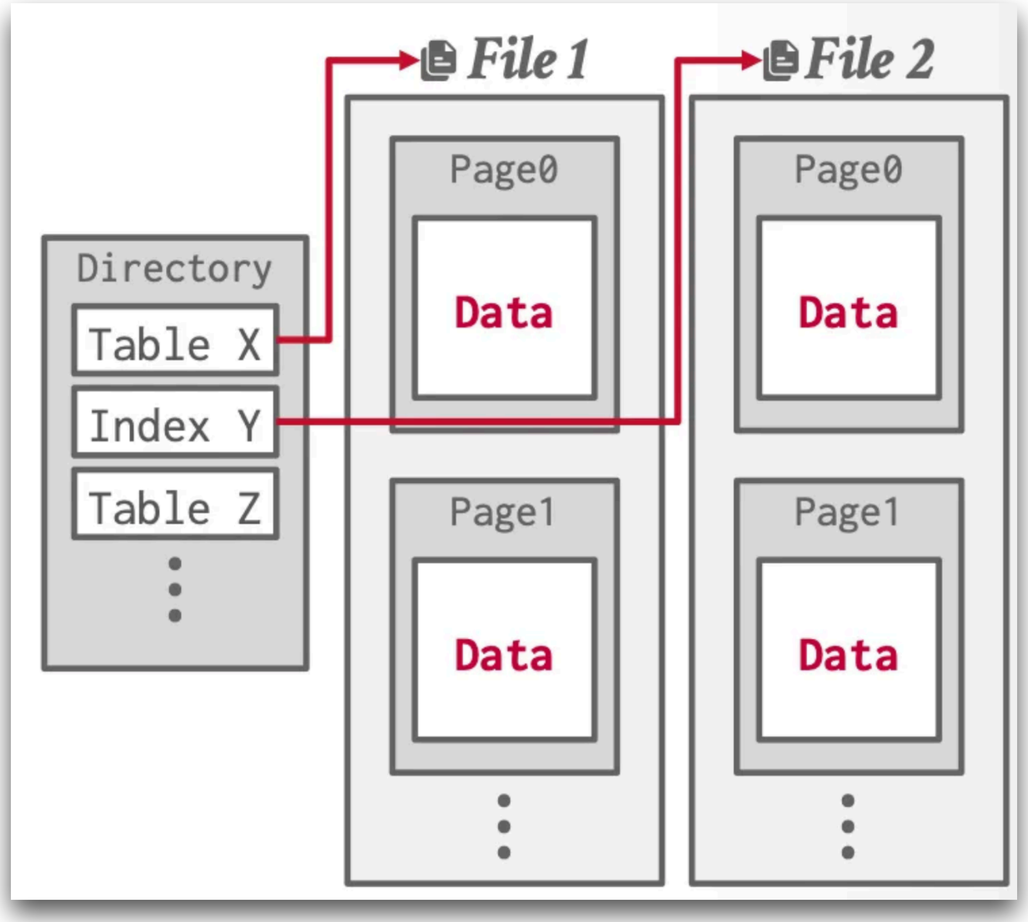


Figure 1.3 Architecture of a DBMS

Catalogue


File 1
File 2
Directory
Table X
Index Y
Table Z
Page0
Data
Page1
Data
Page0
Data
Page1
Data

Buffer Pool

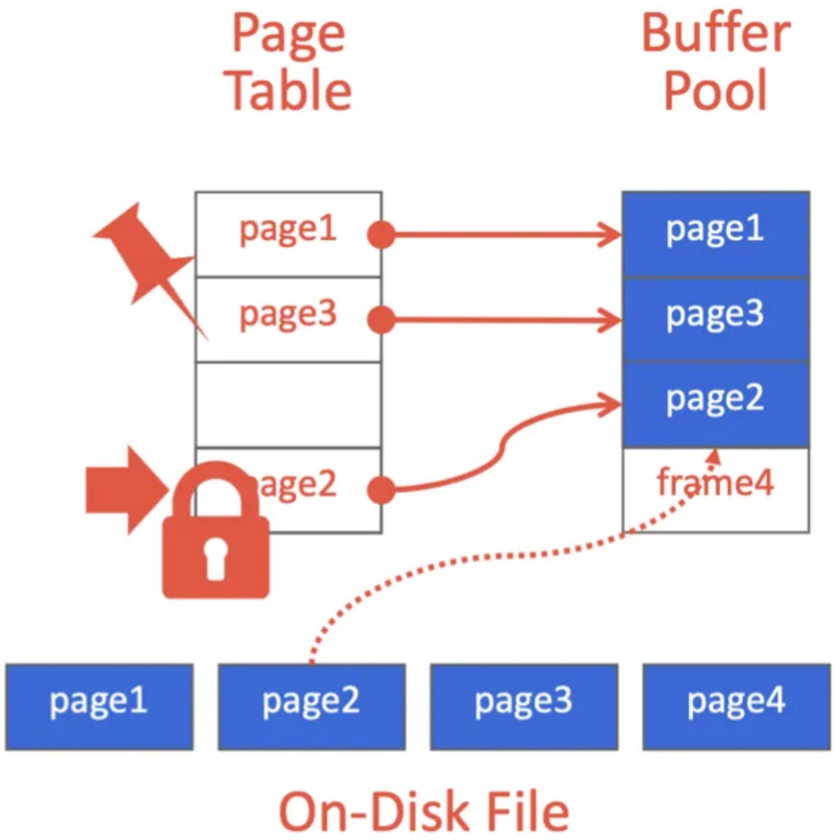
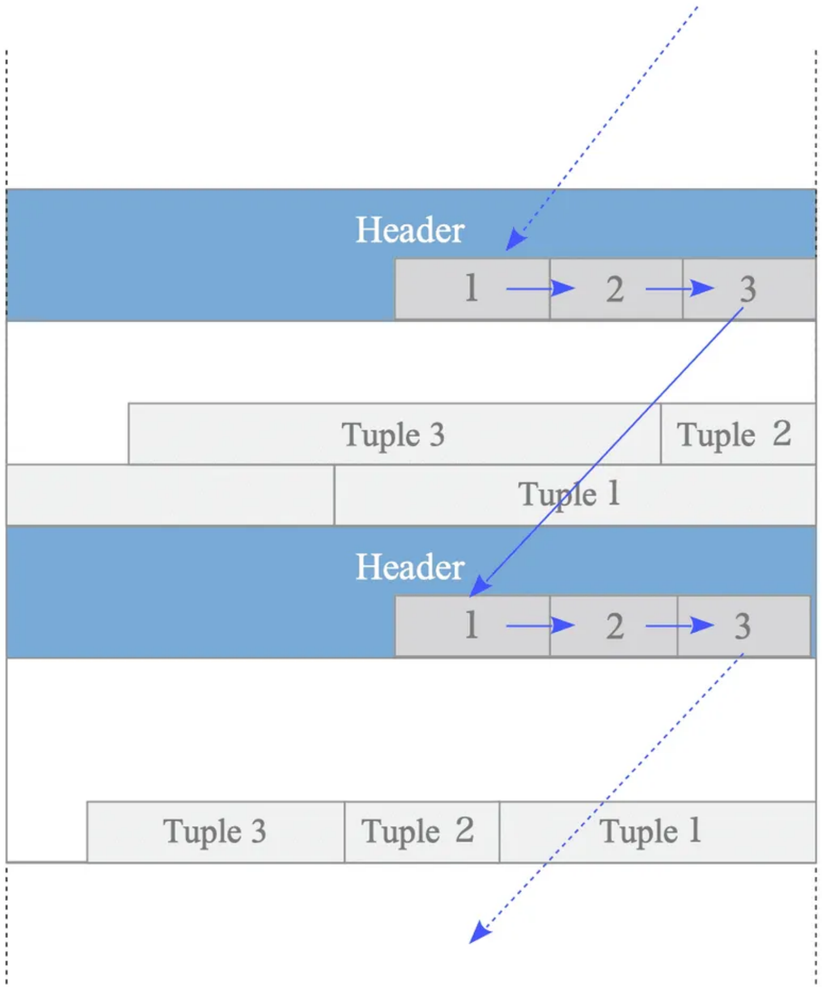


Figure 2: Buffer pool organization and meta-data

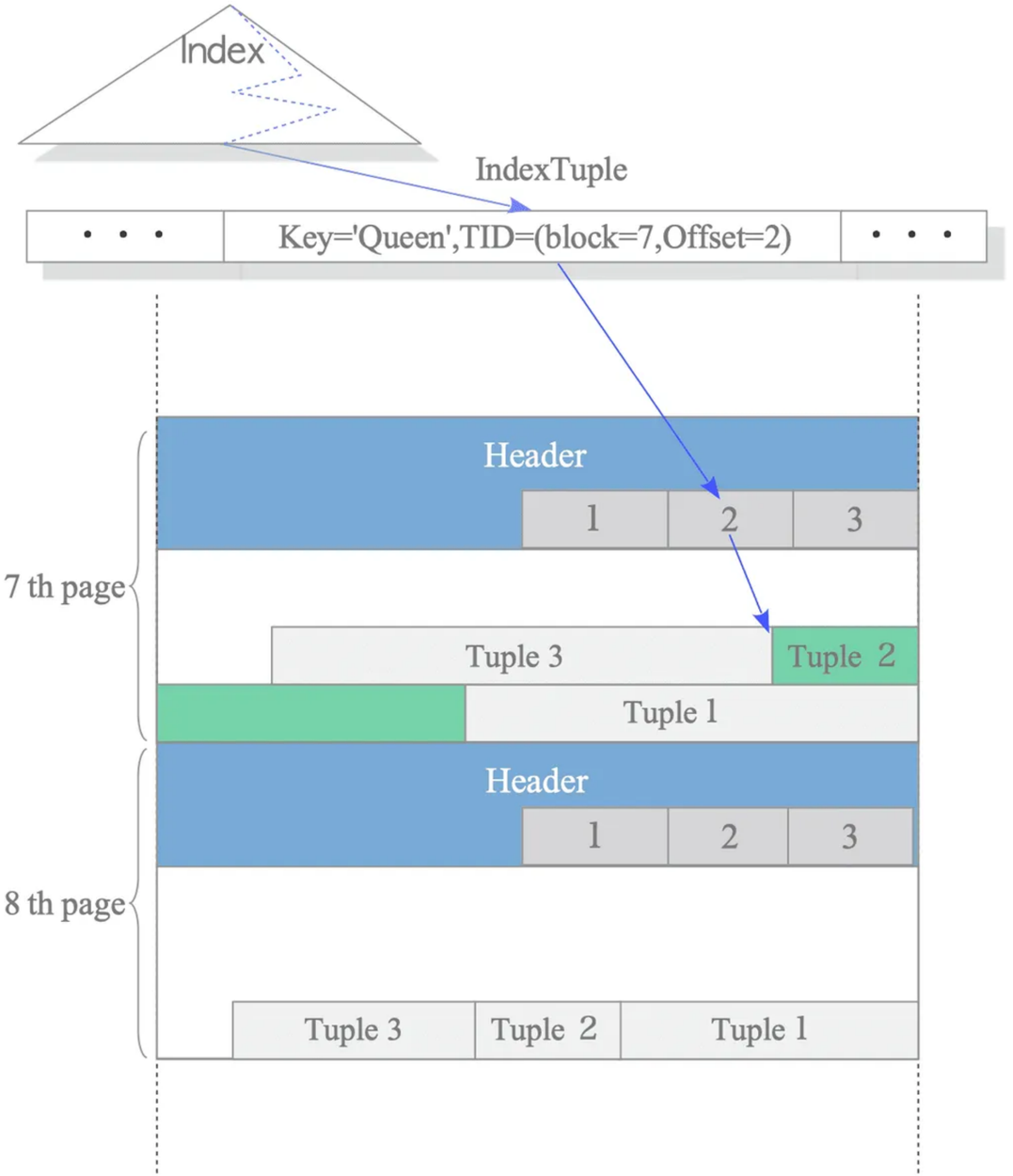
(a) Sequential Scan

SELECT * FROM tbl;

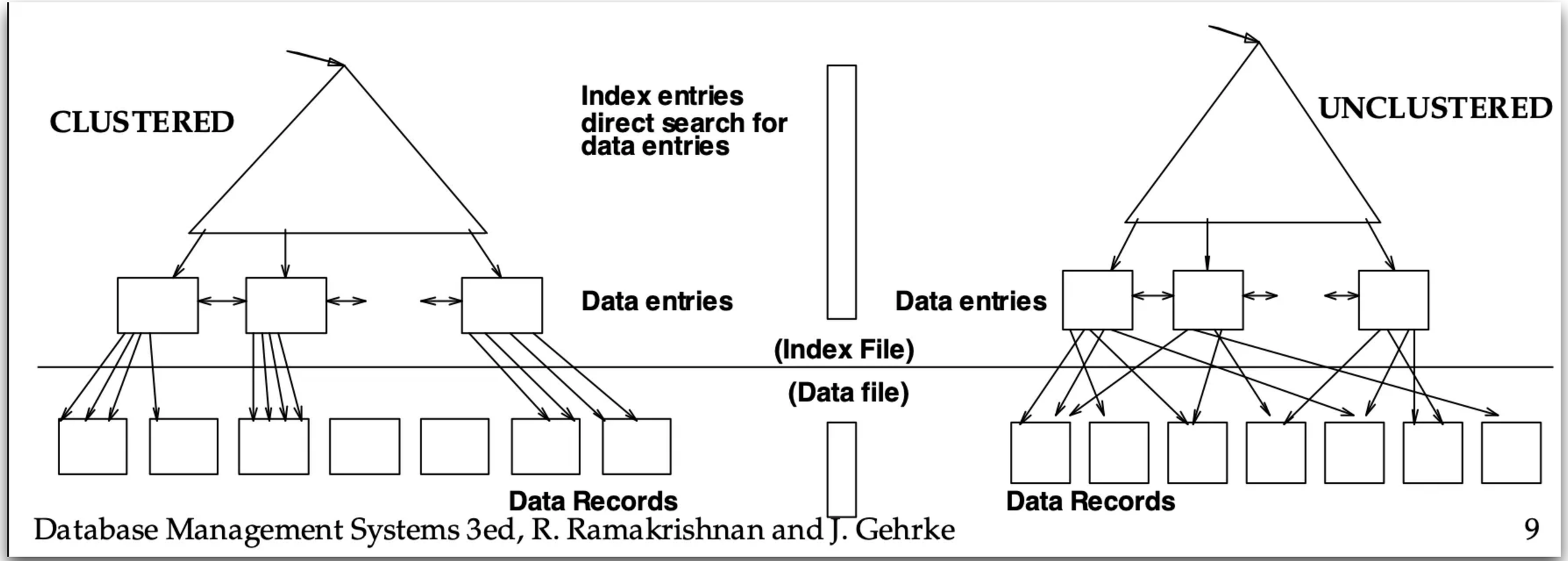


(b) Index Scan

SELECT * FROM tbl WHERE col = 'Queen';



Index


CLUSTERED
Index entries
direct search for
data entries
Data entries
Data Records
(Index File)
(Data file)
UNCLUSTERED
Data entries
Data Records

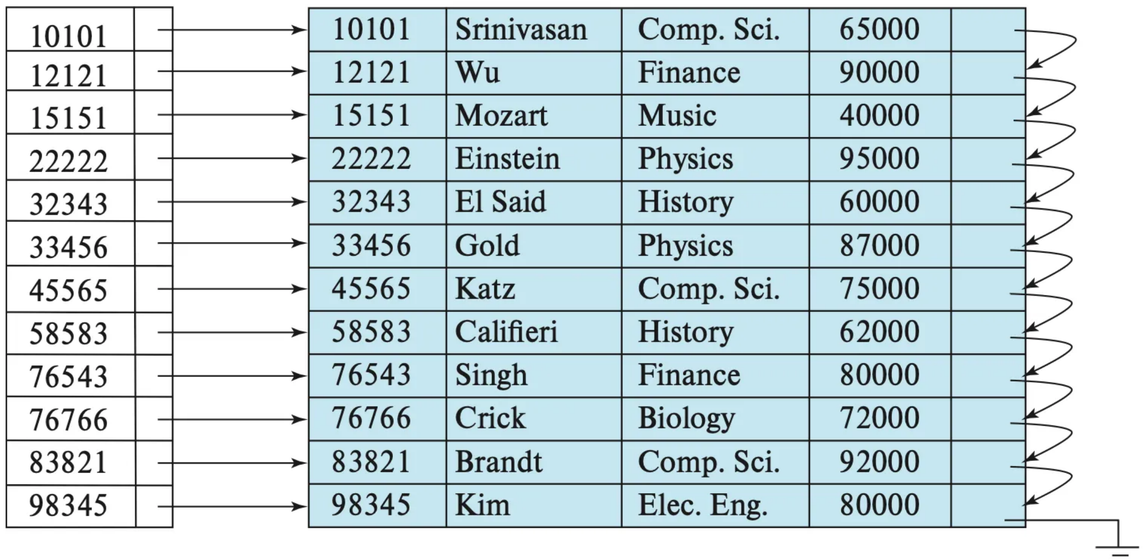


Figure 14.2 Dense index.

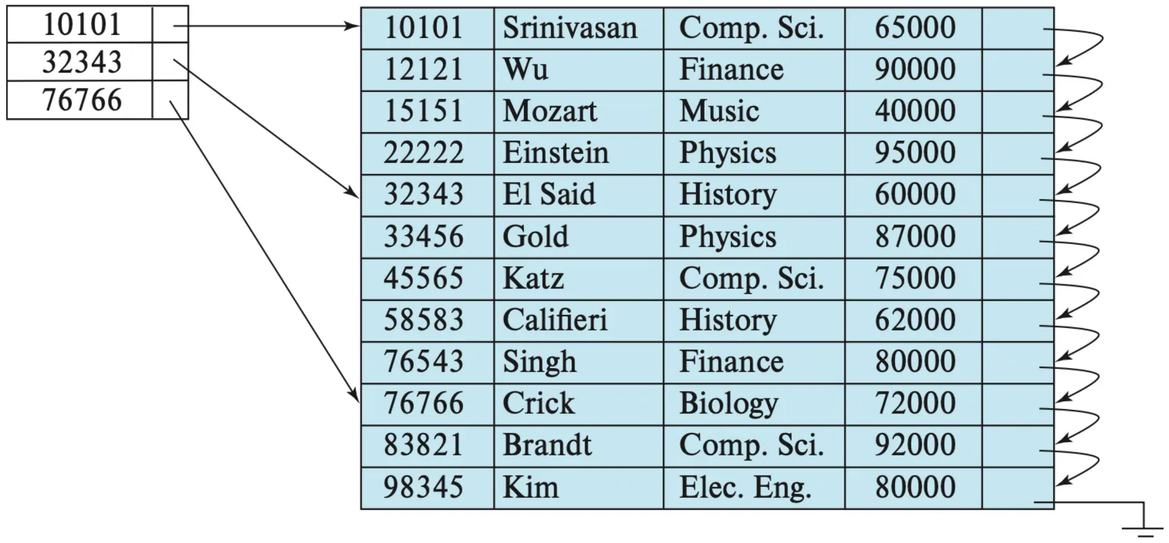


Figure 14.3 Sparse index.

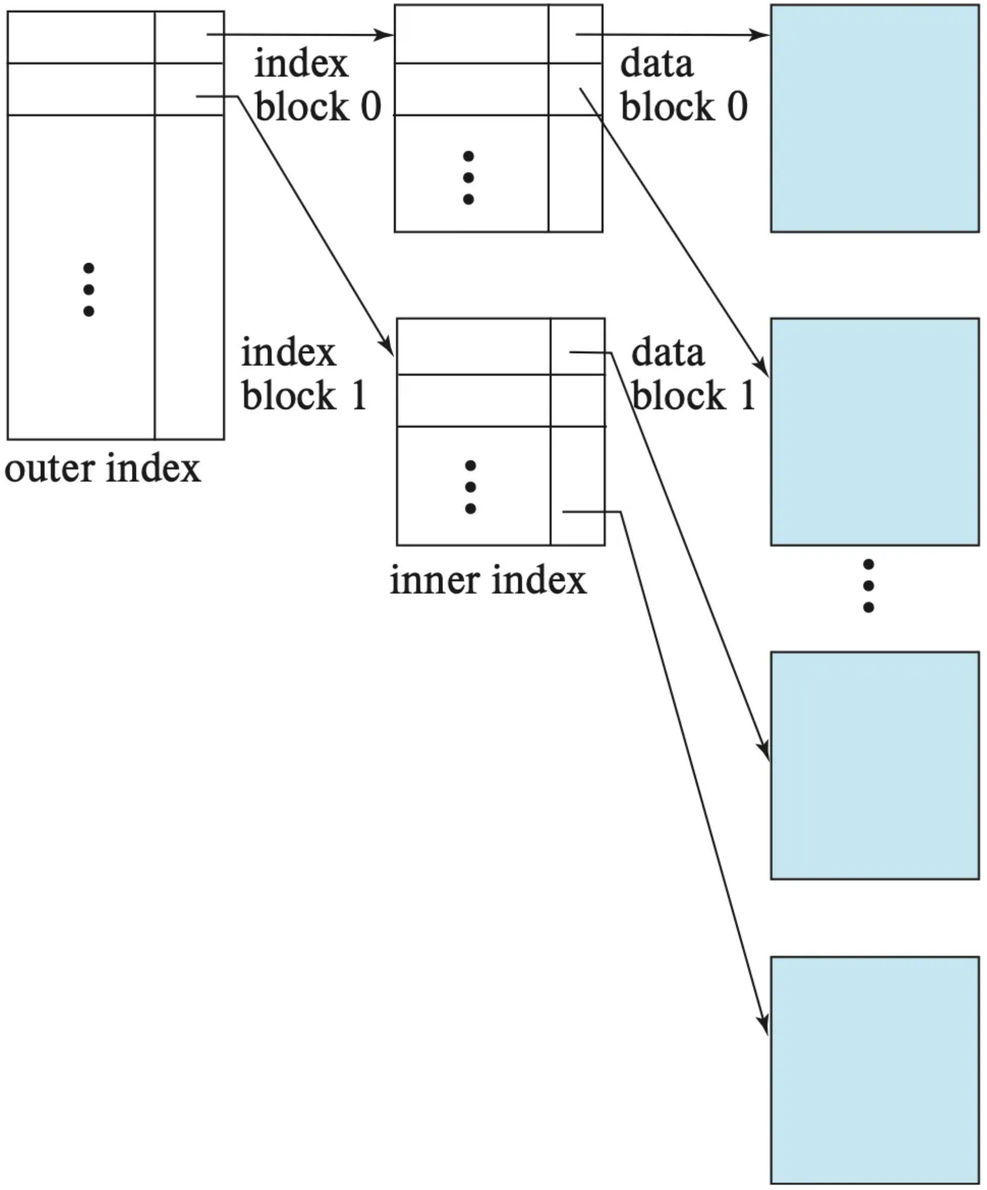
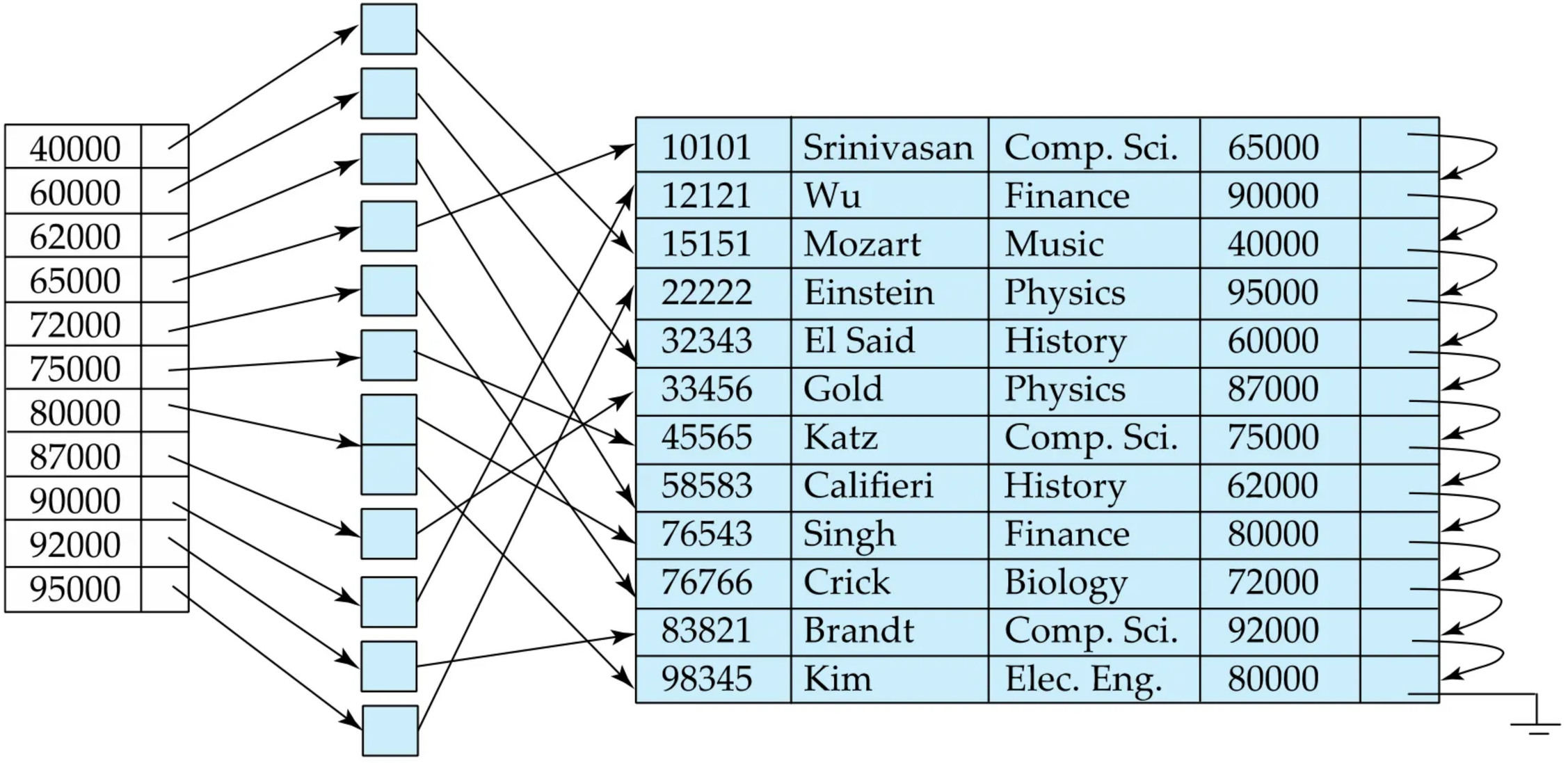


Figure 14.5 Two-level sparse index.

Secondary Indices Example



Secondary index on *salary* field of *instructor*

B+Tree

record number	*ID*	*gender*	*income_level*
0	76766	m	L1
1	22222	f	L2
2	12121	f	L1
3	15151	m	L4
4	58583	f	L3

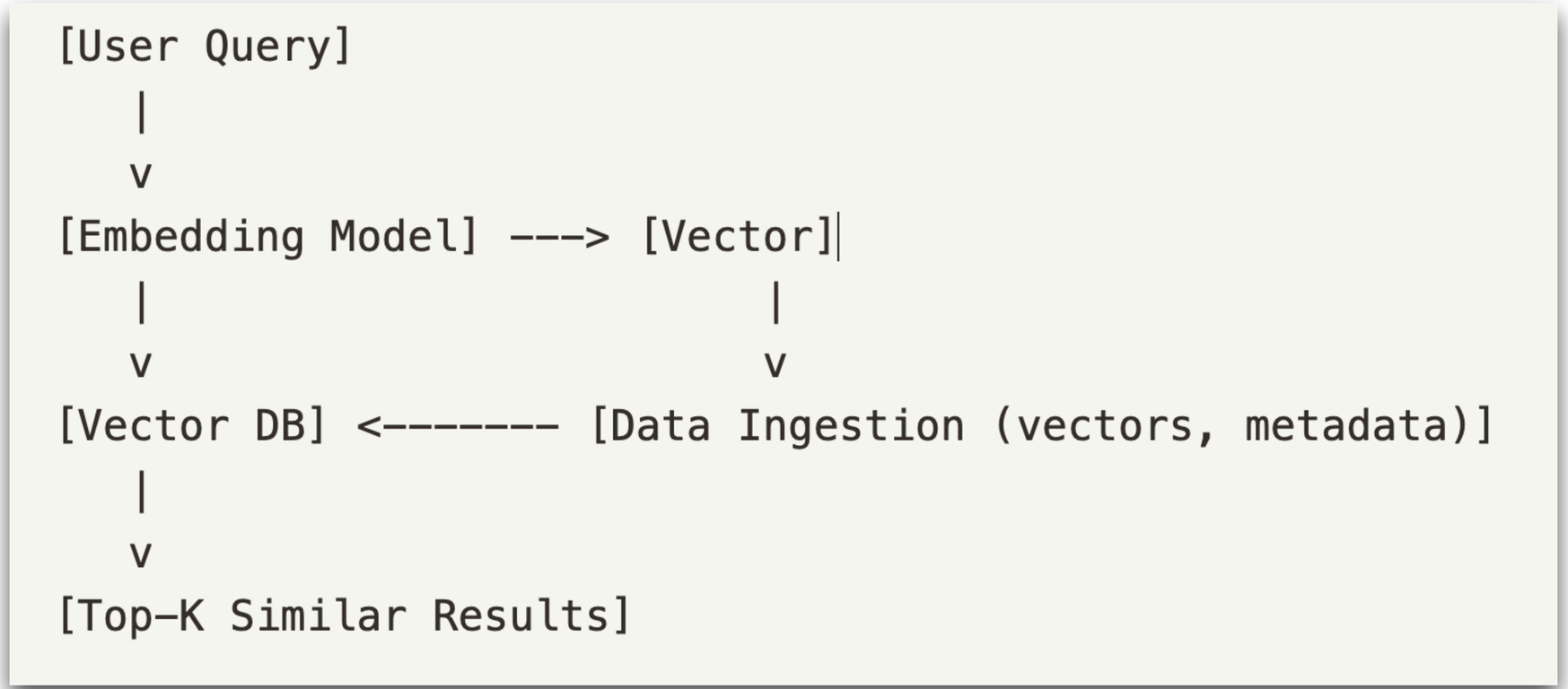
Bitmaps for *gender*

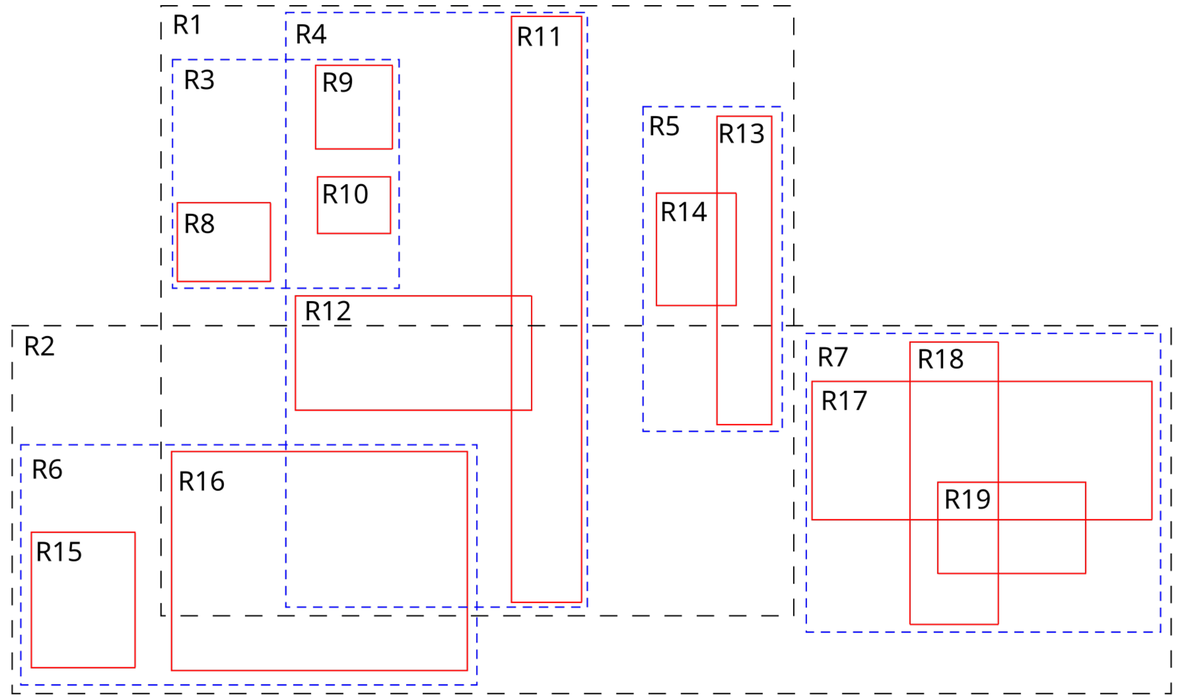
m	10010
f	01101

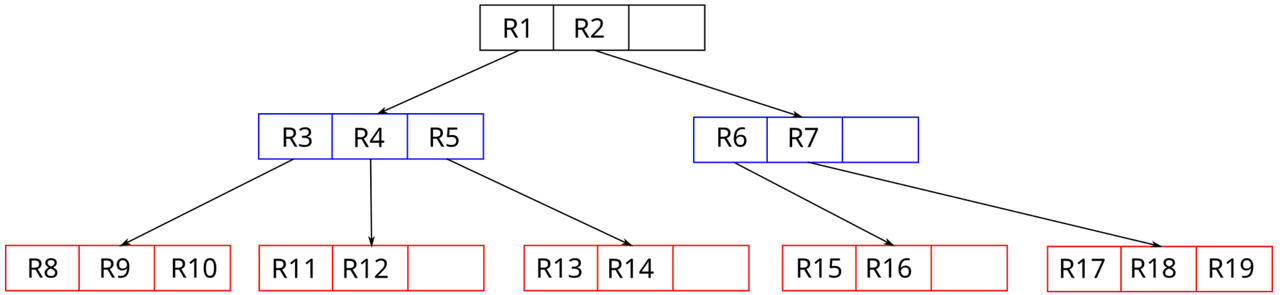
Bitmaps for *income_level*

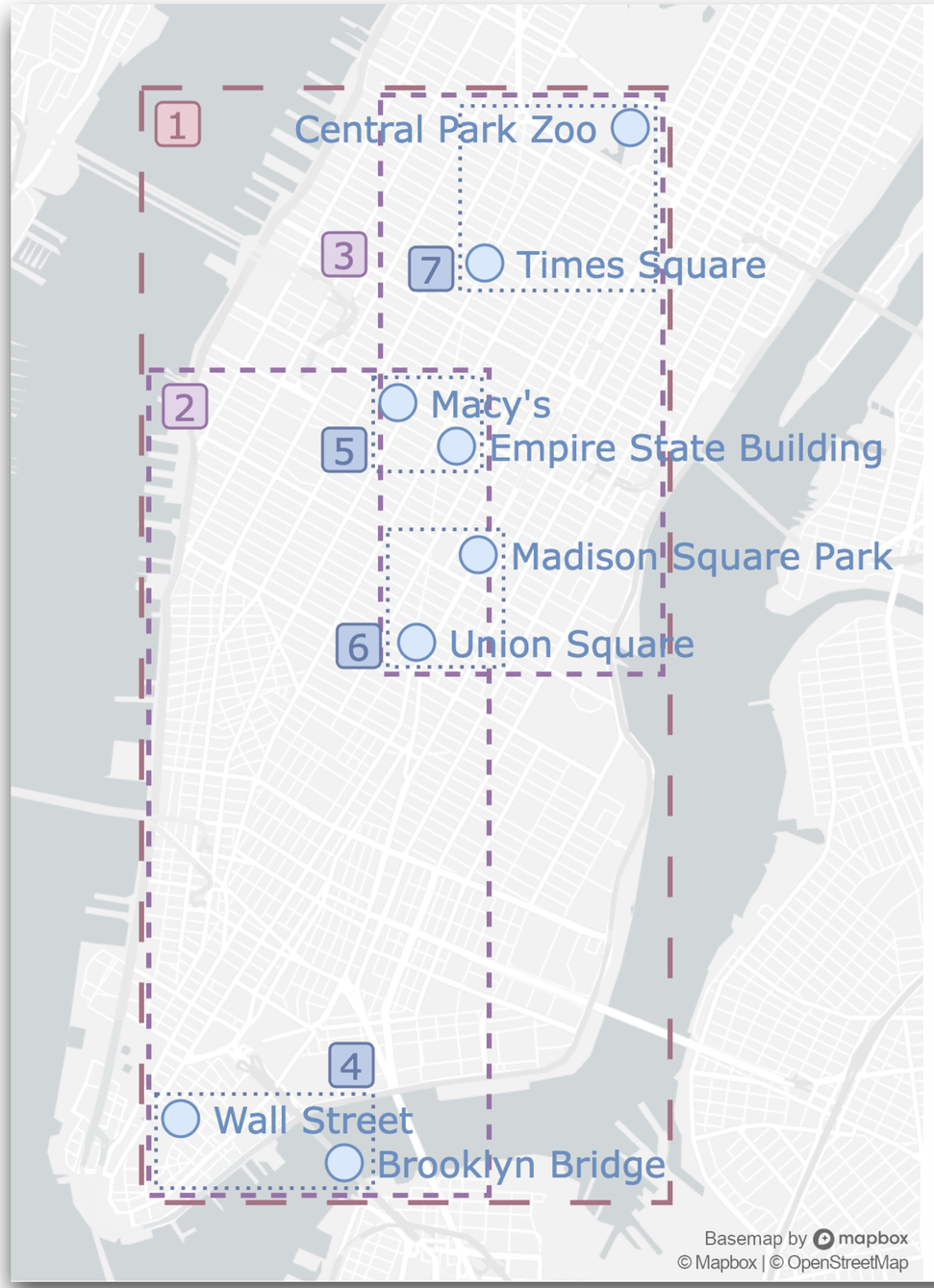
L1	10100
L2	01000
L3	00001
L4	00010
L5	00000

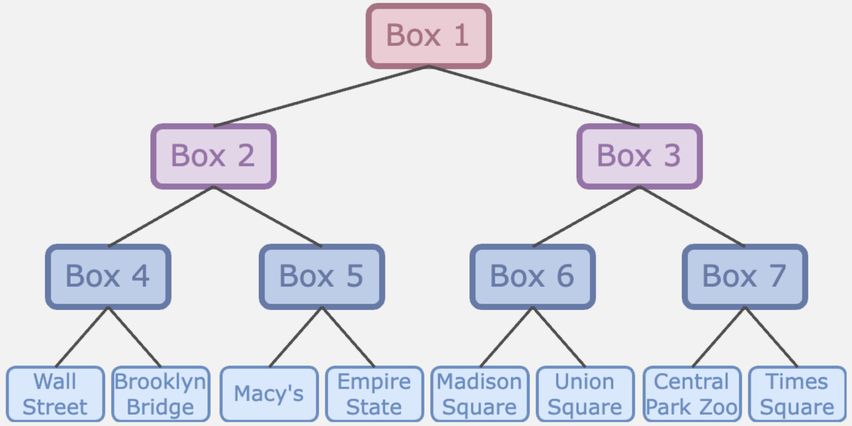
©Silberschatz, Korth and Sudarshan

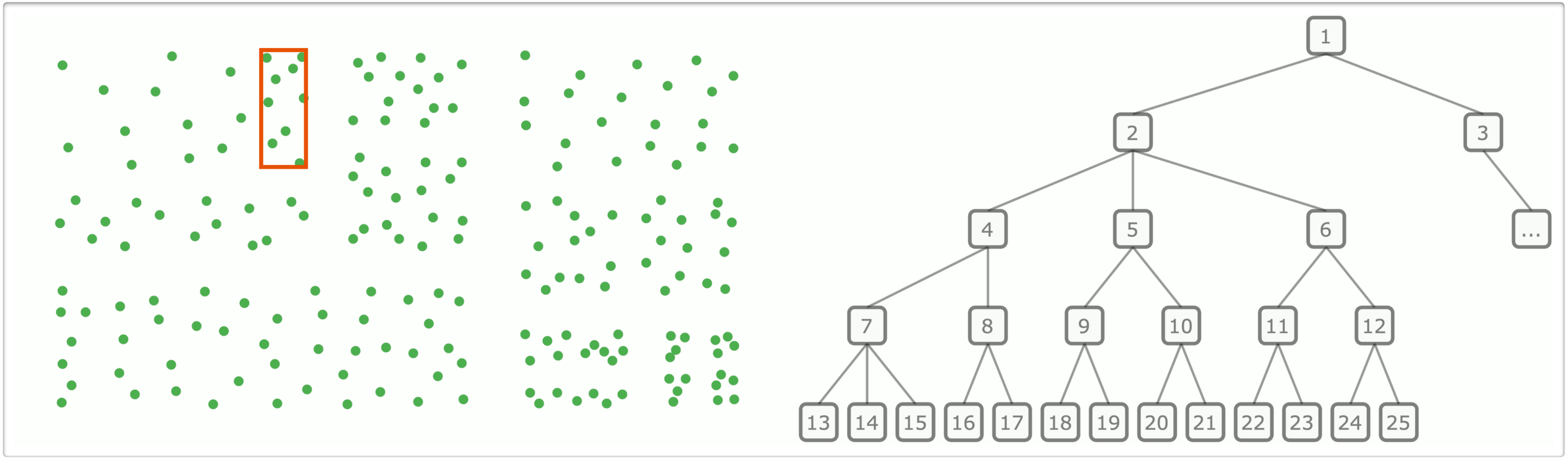

[User Query]
|
v
[Embedding Model] ---> [Vector]
|
v
[Vector DB] <------- [Data Ingestion (vectors, metadata)]
|
v
[Top-K Similar Results]


R1
R4
R11
R3
R9
R5
R13
R10
R14
R8
R12
R2
R7
R18
R17
R6
R16
R19
R15

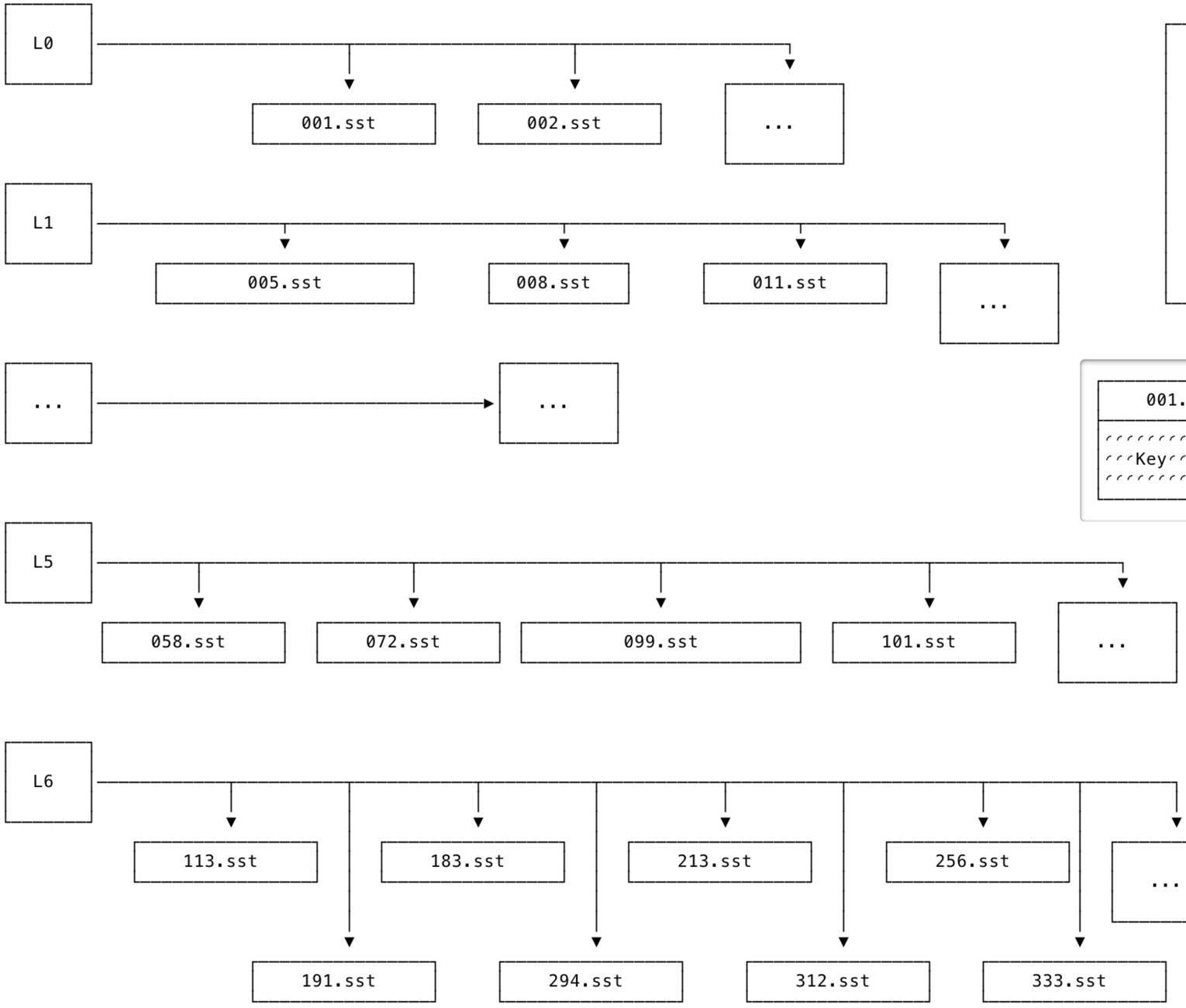

R1
R2
R3
R4
R5
R6
R7
R8
R9
R10
R11
R12
R13
R14
R15
R16
R17
R18
R19

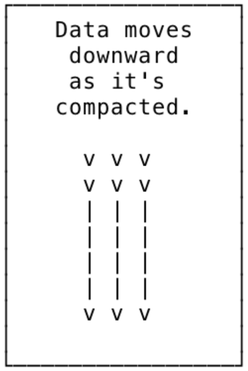

1
Central Park Zoo
3
7
Times Square
2
Macy's
5
Empire State Building
Madison Square Park
6
Union Square
4
Wall Street
Brooklyn Bridge
Basemap by mapbox
© Mapbox | © OpenStreetMap

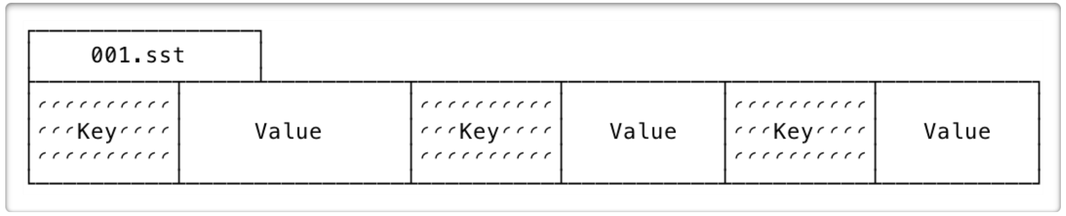

Box 1
Box 2
Box 3
Box 4
Box 5
Box 6
Box 7
Wall Street
Brooklyn Bridge
Macy's
Empire State
Madison Square
Union Square
Central Park Zoo
Times Square

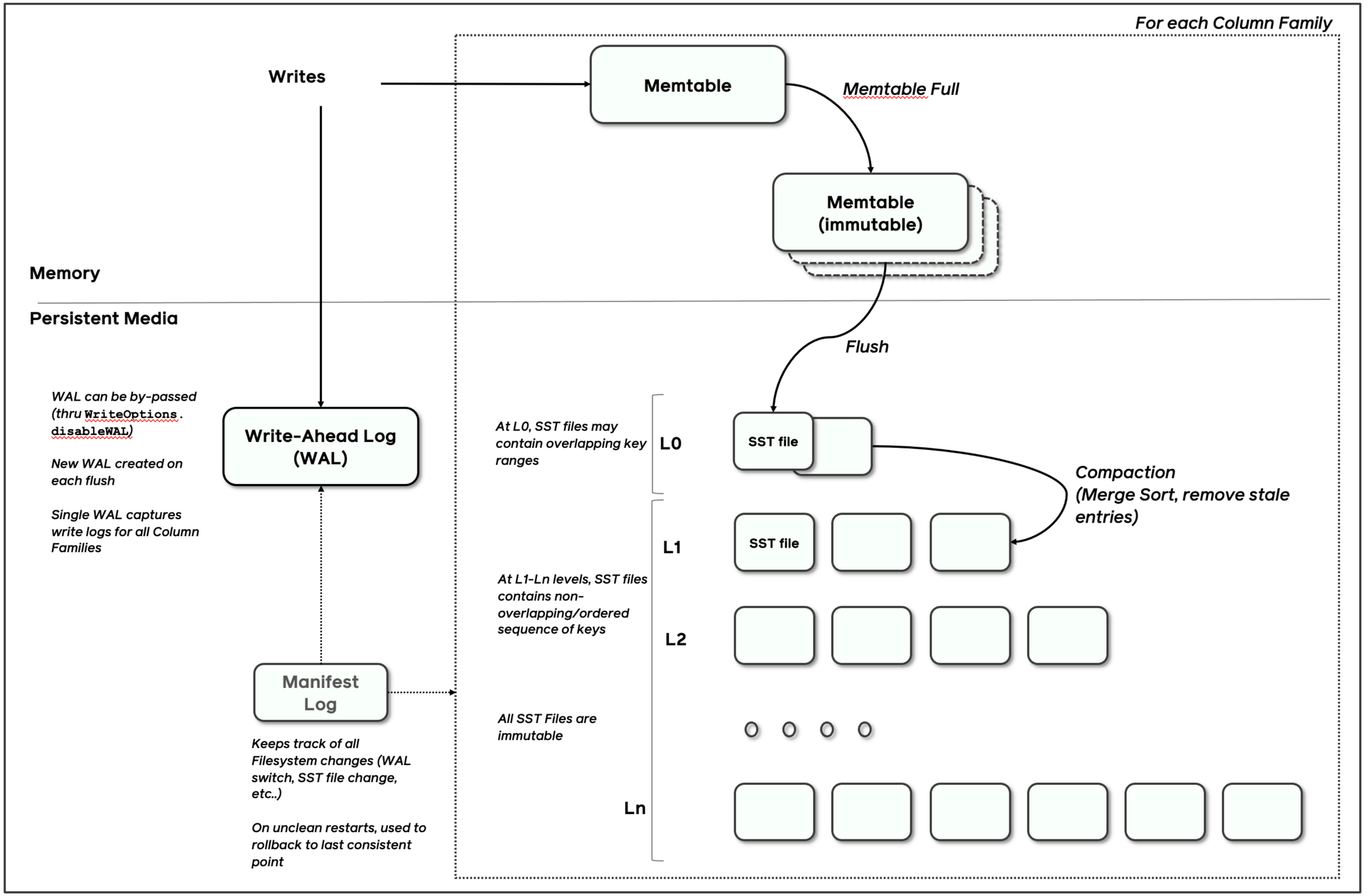

1
2
3
4
5
6
...
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25

LSM


L0
001.sst
002.sst
...
L1
005.sst
008.sst
011.sst
...
...
...
L5
058.sst
072.sst
099.sst
101.sst
...
L6
113.sst
183.sst
213.sst
256.sst
...
191.sst
294.sst
312.sst
333.sst


Data moves
downward
as it's
compacted.

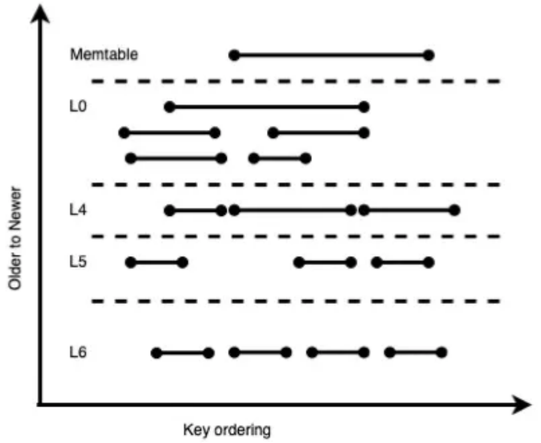

001.sst
Key
Value
Key
Value
Key
Value


For each Column Family
Writes
Memtable
Memtable Full
Memtable (immutable)
Memory
Persistent Media
Flush
WAL can be by-passed (thru WriteOptions.disableWAL)
New WAL created on each flush
Single WAL captures write logs for all Column Families
Write-Ahead Log (WAL)
At L0, SST files may contain overlapping key ranges
L0
SST file
Compaction (Merge Sort, remove stale entries)
L1
SST file
At L1-Ln levels, SST files contains non-overlapping/ordered sequence of keys
L2
Manifest Log
All SST Files are immutable
Ln
Keeps track of all Filesystem changes (WAL switch, SST file change, etc..)
On unclean restarts, used to rollback to last consistent point

Storage

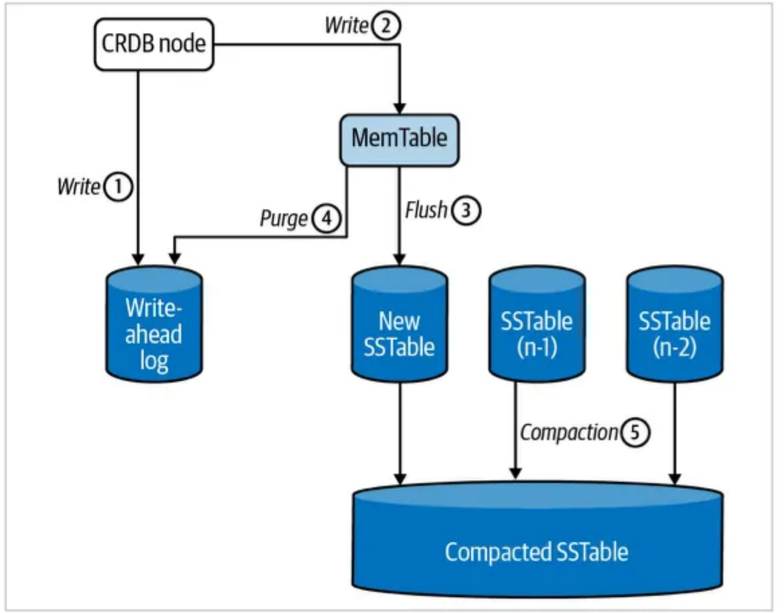
Log-Structured Merge (LSM) Trees

- Pebble implements a log-structured merge tree
- File format is a Sorted String Table (SSTable)
- SSTables exist on multiple levels, numbered L0 to L6
  - L0 (MemTable) contains an unordered set of SSTables, which receives new values.
  - Compaction: Periodically, SSTables are compacted into larger consolidated stores in the lower levels
  - L1 to L6: SSTables are ordered and nonoverlapping so only one SSTable per level could possibly hold a given key
- SSTables are internally sorted and indexed
- Writes are fast since the SSTables are append-only
- Write-ahead log (WAL) ensures that data is not lost in node failure



Storage

Log-Structured Merge (LSM) - Writes

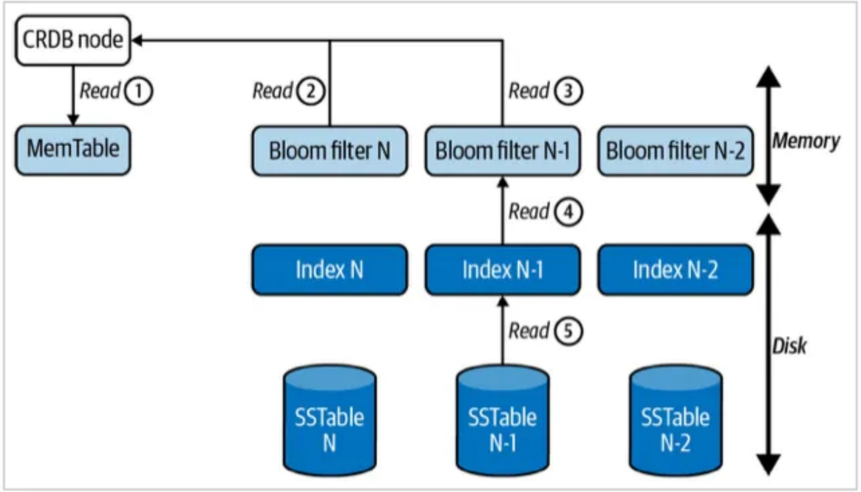


LSM Writes

1. Writes from higher CRDB layers are applied to the write-ahead log (WAL)
2. Then applied to the MemTable
3. Once MemTable reaches a certain size, it is flushed to disk to create a new SSTable
4. WAL can be purged after the MemTable is flushed to disk
5. Multiple SSTable are routinely merged (compacted) into larger SSTables

Storage

SSTables & Bloom Filters - LSM Reads



LSM Reads. First, MemTable is checked, then the bloom filters from the L0 layer down. If a bloom filter returns a possible match, the index is examined and result returned

- **SSTables are indexed but …**
  - Searching through every SSTable index can be expensive with a large number of tables
  - *Bloom Filters* reduce number of lookups
- **Bloom Filters**
  - Compact and quick-to-maintain structure
  - Indicate whether an SSTable *might* contains a key
  - Indicate whether an SSTable does not contain a key (can be skipped)

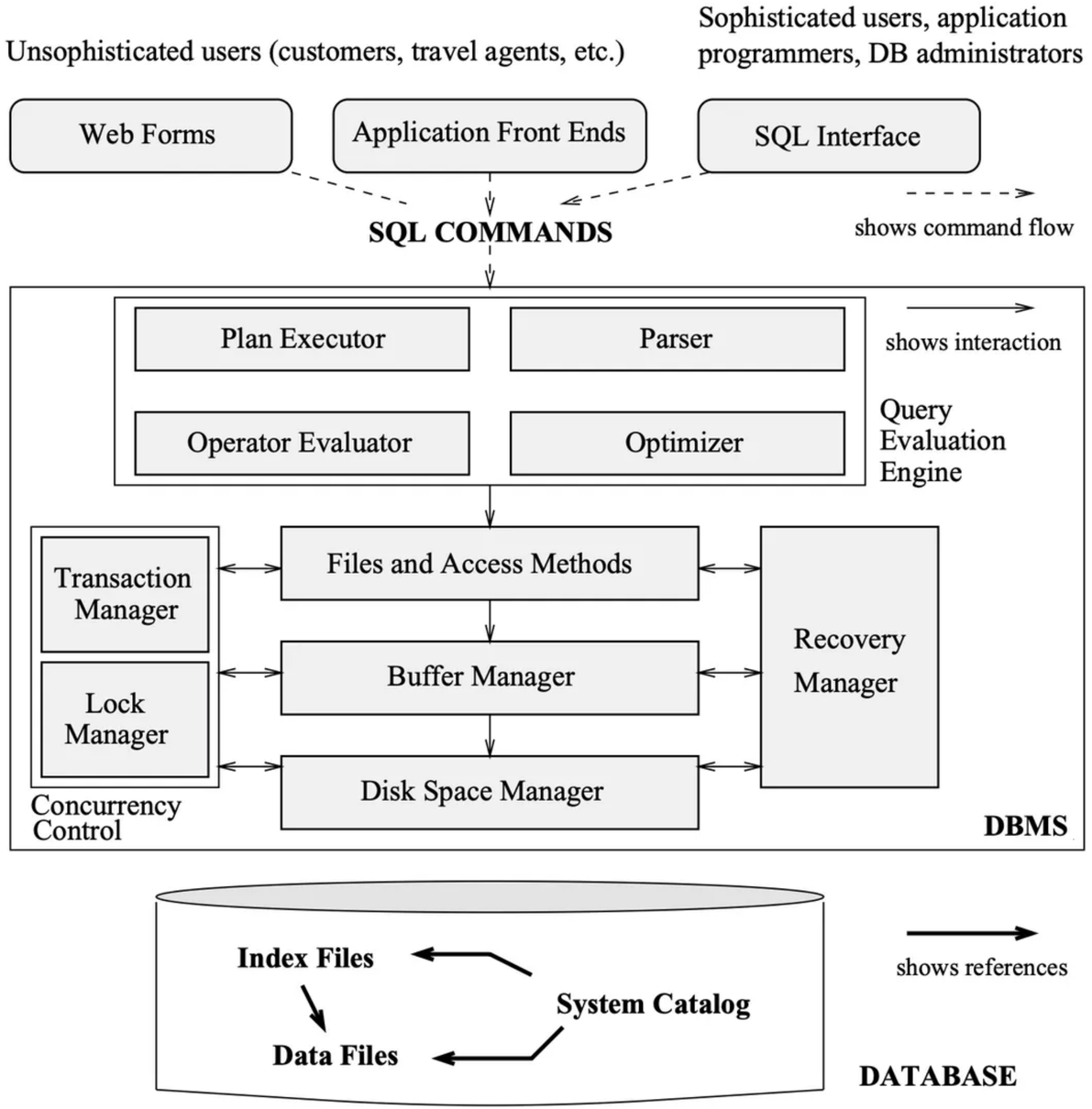
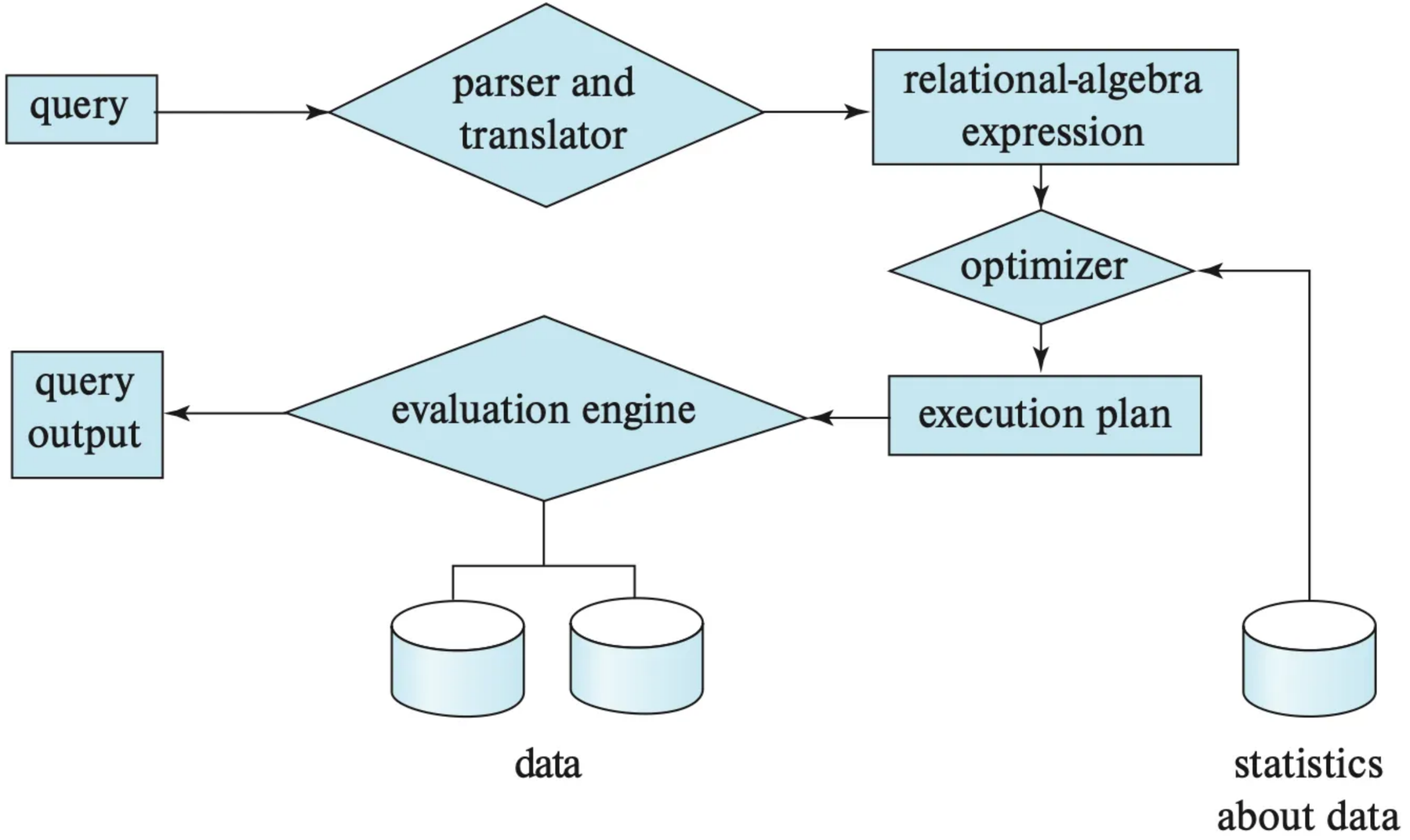


Figure 1.3 Architecture of a DBMS

Query Plan


query
parser and translator
relational-algebra expression
optimizer
query output
evaluation engine
execution plan
data
statistics about data

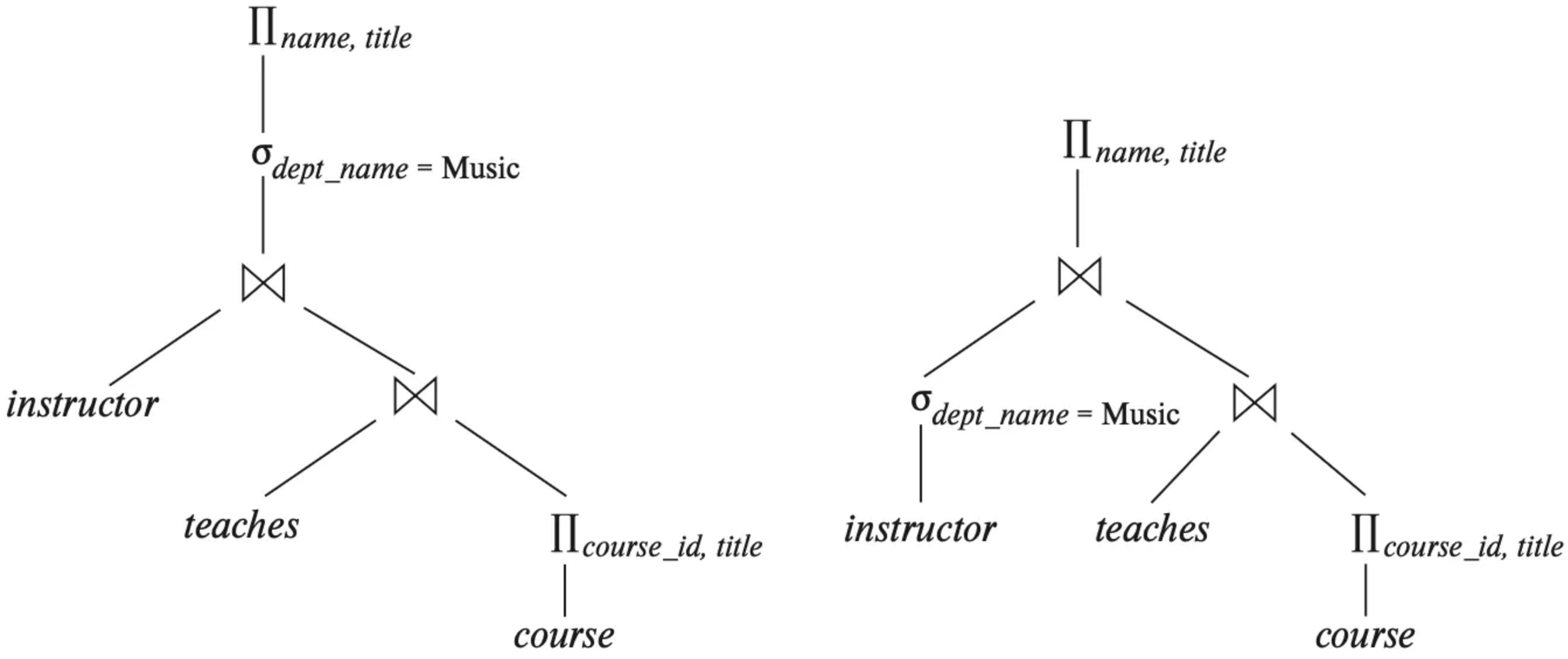


Figure 16.1 Equivalent expressions.