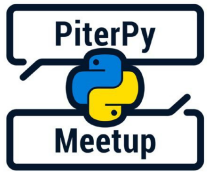
Data Storage Systems

Dmitry Alimov

2018

$ whoami

- Dmitry Alimov (@delimitry)
- Software Engineer
- SPbPython and PiterPy active member & speaker
- SPbPython drinkups co-organizer
- CTF player with SiBears team
- Python Software Foundation (PSF) contributing member



Outline

- Storage data structures
  - B-tree
  - LSM-tree
  - Other indices
- RUM conjecture
- OLTP, OLAP, HTAP
- SQL, NoSQL, NewSQL
- DB in CPython
- Books and references *

* References are in brackets “[ref num]”

Intro

- The amounts of data are constantly growing

Intro

- The amounts of data are constantly growing
- Every year new databases appear, existing ones are improved

Intro

- The amounts of data are constantly growing
- Every year new databases appear, existing ones are improved
- Each database has its own trade-offs

Intro

- The amounts of data are constantly growing
- Every year new databases appear, existing ones are improved
- Each database has its own trade-offs
- Understanding them helps to choose the right one

Intro

- The amounts of data are constantly growing
- Every year new databases appear, existing ones are improved
- Each database has its own trade-offs
- Understanding them helps to choose the right one
- Knowing and understanding of storage internals helps to make better design decisions, troubleshoot problems, tune database

Storage Data Structures

“Wer Ordnung hält, ist nur zu faul zum Suchen”

(He who keeps order is just too lazy
to spend his time searching)

German proverb

Simple datastore

```
def set(key, value):
    with open('main.db', 'a') as db_file:
        db_file.write('{},{}\n'.format(key, value))

def get(key):
    value = None
    with open('main.db', 'r') as db_file:
        for line in db_file:
            k, v = line.split(',')
            if k == key:
                value = v.rstrip()
    return value
```

Simple datastore

```
>>> set('a', 'one')
>>> set('b', 'two')
>>> set('c', 'three')
>>> set('b', 'four')

>>> print(get('a'))
one
>>> print(get('b'))
four
>>> print(get('z'))
None
```

```
$ cat main.db
a,one
b,two
c,three
b,four
```

Questions

- Escaping

```
>>> set('a,b', 'oops')

$ cat main.db
...
a,b,oops  # ???
...
```

Questions

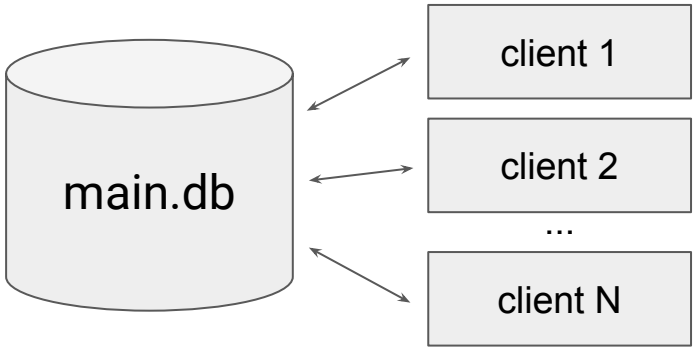
- Escaping
- Deleting

```
>>> delete('c')

$ cat main.db
...
c,<tombstone>
...
```

Questions

- Escaping
- Deleting
- Concurrency



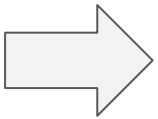
More questions:
Locks
MVCC
Single-writer, multiple-reader
...

Questions

- Escaping
- Deleting
- Concurrency
- Compaction

```
$ cat main.db
a,one
b,two
c,three
b,four
c,<tombstone>
```

Compact



```
$ cat main.db
a,one
b,four
```

Questions

- Escaping
- Deleting
- Concurrency
- Compaction
- Performance

```
$ cat main.db
a,one
b,two
c,three
b,four
```

Insert: O(1), Search: O(n)

Indices

Database indices

- Hash indices [50]

Database indices

- Hash indices [50]
- B-tree [50]

Database indices

- Hash indices [50]
- B-tree [50]
- LSM-tree [54]

Database indices

- Hash indices [50]
- B-tree [50]
- LSM-tree [54]
- Other (Spatial indices (R-trees), BRIN, Log-Structured Hash Table, etc)

Database indices

- Hash indices [50]
- B-tree [50]
- LSM-tree [54]
- Other (Spatial indices (R-trees), BRIN, Log-Structured Hash Table, etc)

Indices speed up read queries, but slow down writes

Databases

B-tree

LSM-tree

B-tree

“B-trees are by far the most important access path structure in database and file systems”

Gray and Reuter, 1992 [31]

“It could be said that the world’s information is at our fingertips because of B-trees”

Goetz Graefe, 2011 [32]

B-tree

Self-balancing tree structure, invented in 1971 by Rudolf Bayer and Ed McCreight

B-tree

Self-balancing tree structure, invented in 1971 by Rudolf Bayer and Ed McCreight
The most widely used indexing structure [1]

B-tree

Self-balancing tree structure, invented in 1971 by Rudolf Bayer and Ed McCreight
The most widely used indexing structure [1]
Used in: MySQL (InnoDB), PostgreSQL, MongoDB, Oracle DB, MS SQL Server, IBM DB2, CouchDB, Couchbase, etc

B-tree

Self-balancing tree structure, invented in 1971 by Rudolf Bayer and Ed McCreight
The most widely used indexing structure [1]
Used in: MySQL (InnoDB), PostgreSQL, MongoDB, Oracle DB, MS SQL Server, IBM DB2, CouchDB, Couchbase, etc

Optimized for paged data access [7]

B-tree

Self-balancing tree structure, invented in 1971 by Rudolf Bayer and Ed McCreight
The most widely used indexing structure [1]
Used in: MySQL (InnoDB), PostgreSQL, MongoDB, Oracle DB, MS SQL Server, IBM DB2, CouchDB, Couchbase, etc

Optimized for paged data access [7]
Branching factor between 50 and 2000 is often used [50]

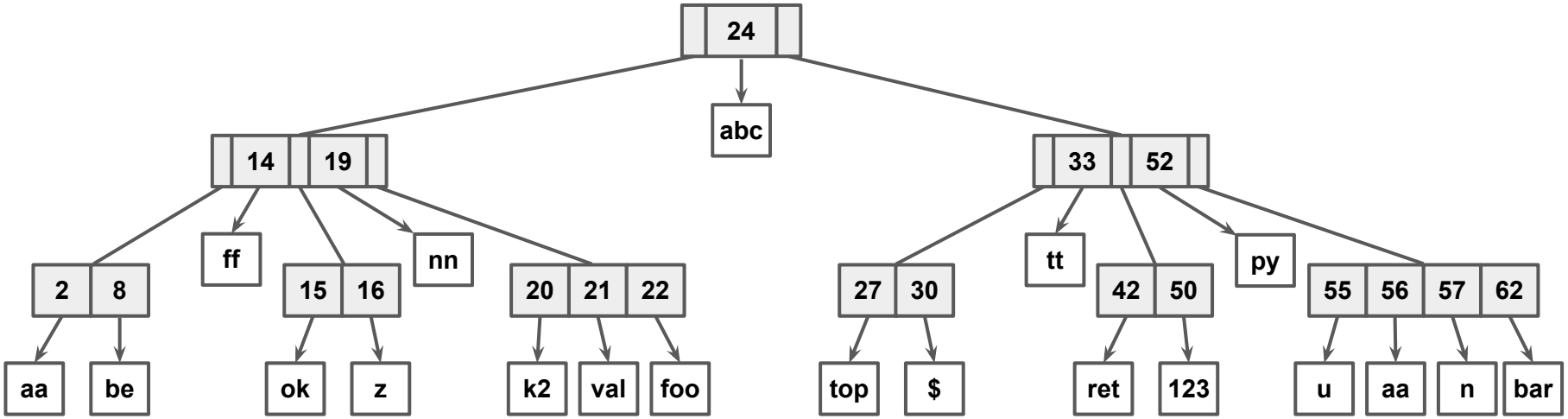
B-tree

Self-balancing tree structure, invented in 1971 by Rudolf Bayer and Ed McCreight
The most widely used indexing structure [1]
Used in: MySQL (InnoDB), PostgreSQL, MongoDB, Oracle DB, MS SQL Server, IBM DB2, CouchDB, Couchbase, etc

Optimized for paged data access [7]
Branching factor between 50 and 2000 is often used [50]

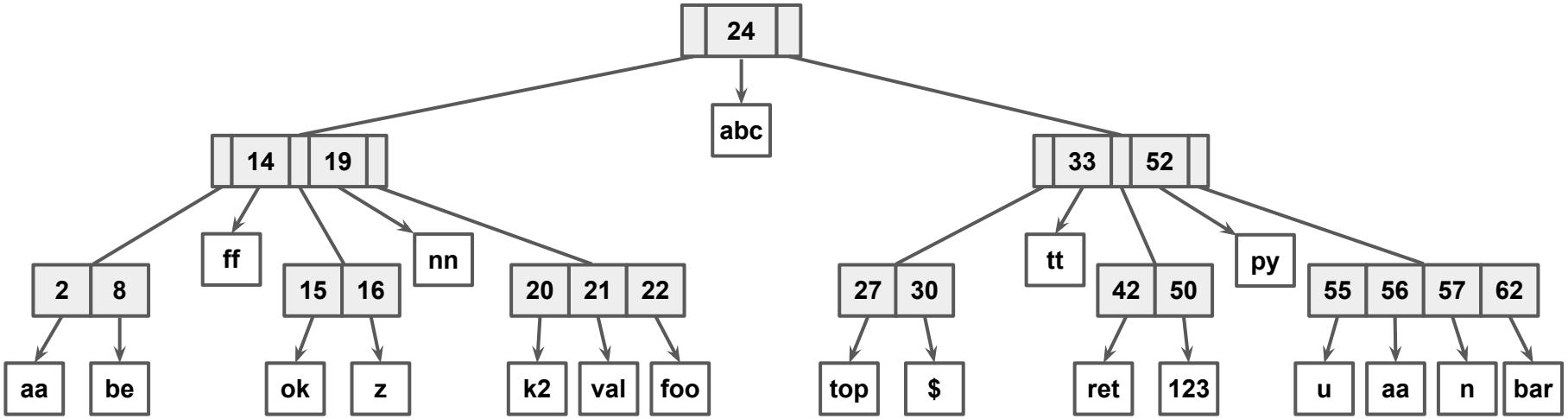
Typically faster for reads [39]

B-tree

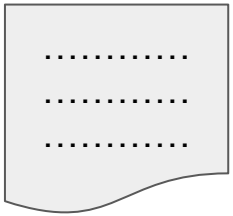


key	value
2	aa
8	be
14	ff
...	...

B-tree durability

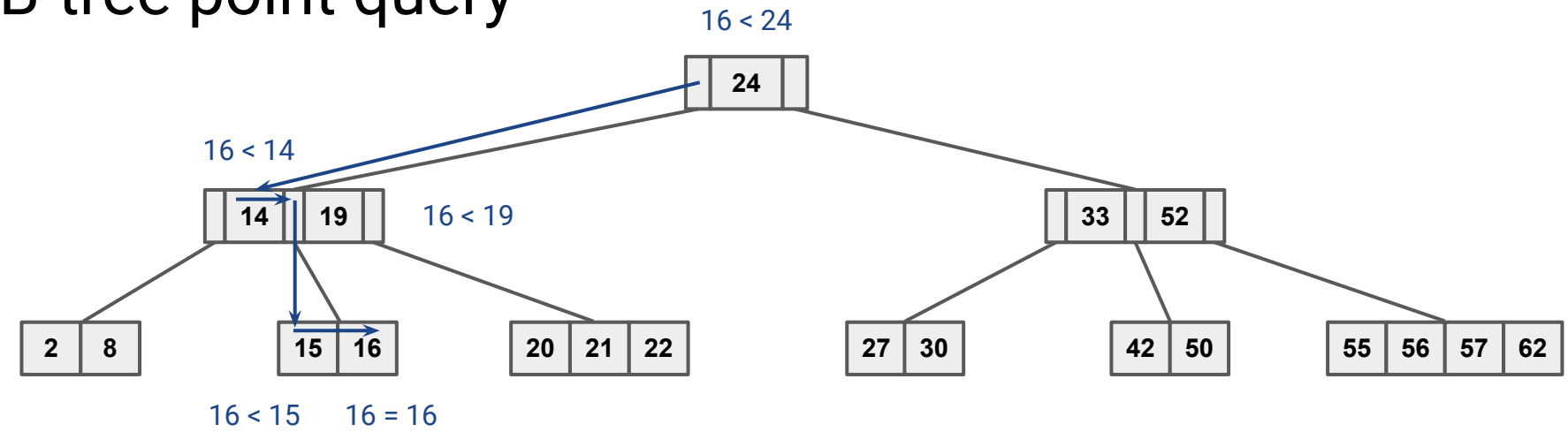


Write-ahead log (WAL)



key	value
2	aa
8	be
14	ff
...	...

B-tree point query

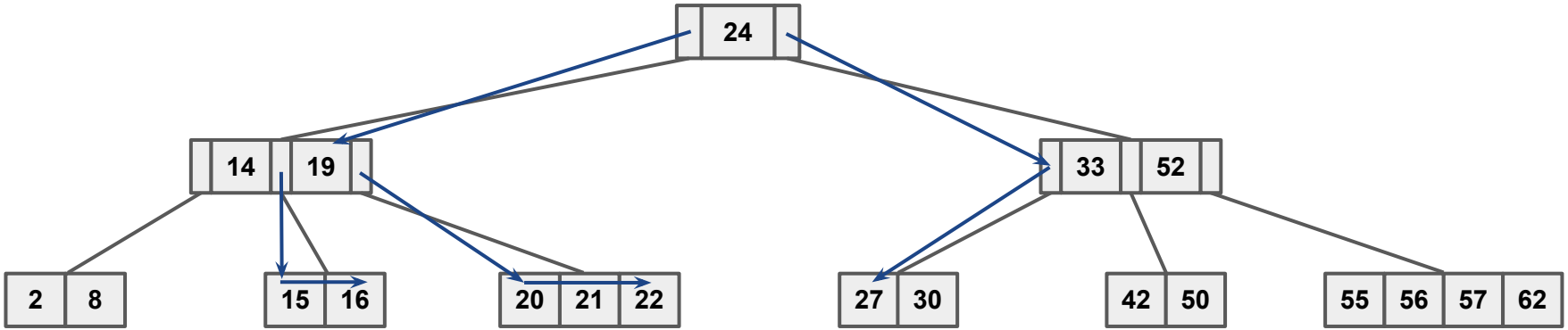


select value with key 16

Insert: $O(\log_B N)$

Search: $O(\log_B N)$

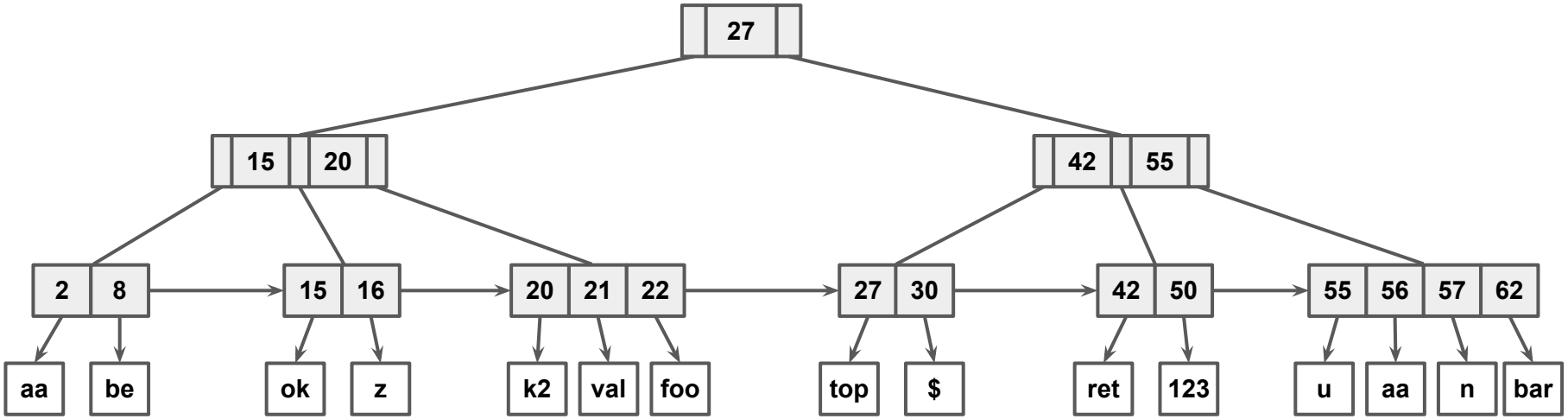
B-tree range query



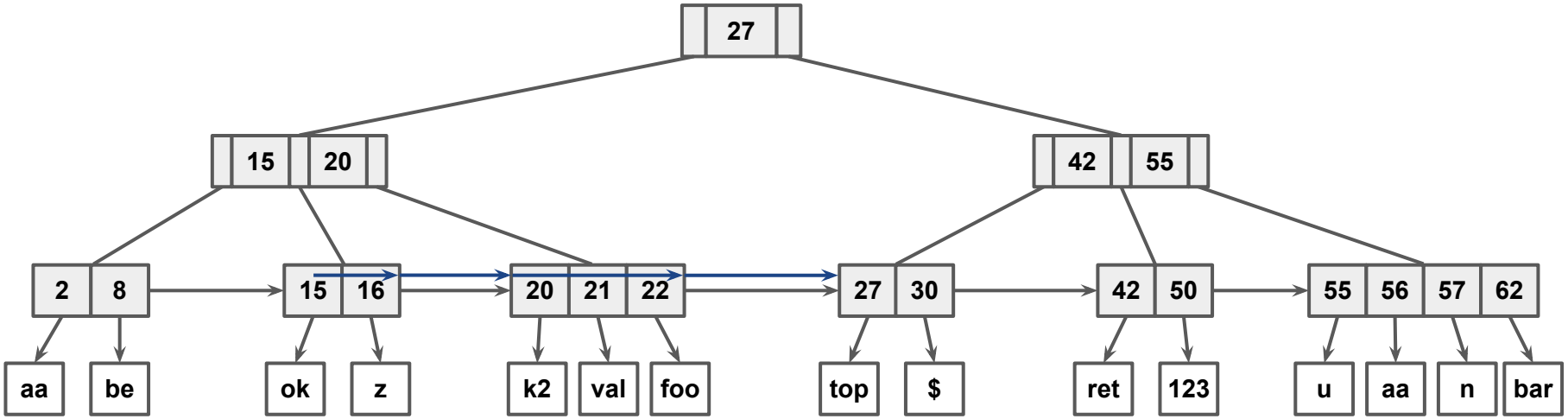
select values with keys in [15...27]

Non-optimal :(

B+ tree


27
15
20
42
55
2
8
15
16
20
21
22
27
30
42
50
55
56
57
62
aa
be
ok
z
k2
val
foo
top
$
ret
123
u
aa
n
bar

B+ tree range query



select values with keys in [15...27]

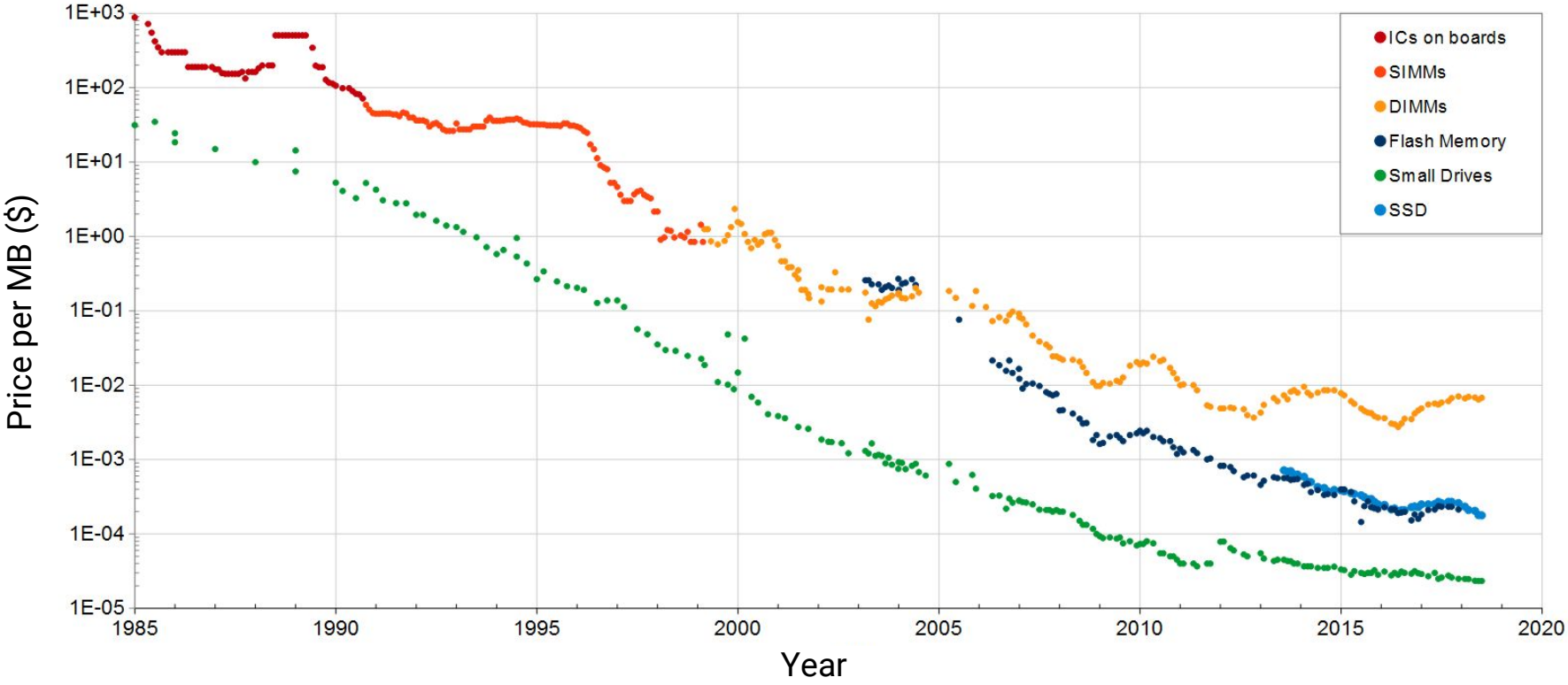
Insert: $O(\log_B N)$

Search: $O(\log_B N)$

RangeQuery: $O(\log_B N + k)$

Memory vs Disk

Memory Prices [34]



Memory vs Disk [56, 57, 58, 59]

Operation	Time, ns *	Comment
Memory access	100	
SSD random read	16 000 (16 µs)	
HDD seek	4 000 000 (4 ms)	
SSD I/O	50 000 - 150 000 (50 - 150)	
HDD I/O	1 000 000 - 10 000 000 (1 - 10 ms)	
Read 1 MB sequentially from memory	9 000 (9 µs)	
Read 1 MB sequentially from SSD	200 000 (200 µs)	22x memory
Read 1 MB sequentially from HDD	2 000 000 (2 ms)	10x SSD, 220x memory

* Numbers for 2015

LSM-tree

LSM-tree (Log-structured merge-tree)

Patrick O'Neil et al., introduced in 1996 [54]

LSM-tree (Log-structured merge-tree)

Patrick O'Neil et al., introduced in 1996 [54]
Used in: LevelDB, RocksDB, Cassandra, HBase, BigTable, InfluxDB, ScyllaDB, SQLite4, Tarantool, MongoDB (WiredTiger), etc.

LSM-tree (Log-structured merge-tree)

Patrick O'Neil et al., introduced in 1996 [54]
Used in: LevelDB, RocksDB, Cassandra, HBase, BigTable, InfluxDB, ScyllaDB, SQLite4, Tarantool, MongoDB (WiredTiger), etc.

Google’s Bigtable paper in 2006 [1, 53]

LSM-tree (Log-structured merge-tree)

Patrick O'Neil et al., introduced in 1996 [54]
Used in: LevelDB, RocksDB, Cassandra, HBase, BigTable, InfluxDB, ScyllaDB, SQLite4, Tarantool, MongoDB (WiredTiger), etc.

Google’s Bigtable paper in 2006 [1, 53]

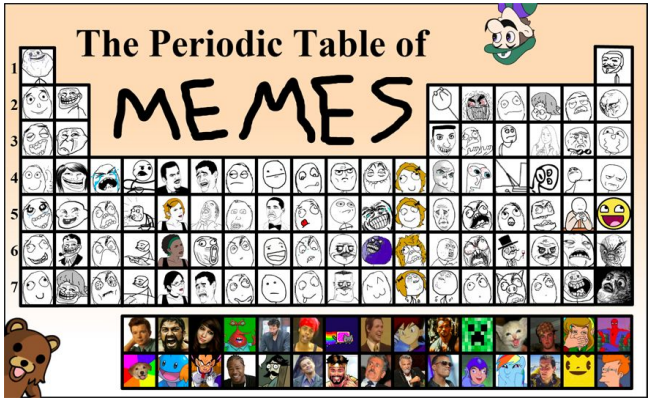
Memtable (B-tree, Skip List, etc)

LSM-tree (Log-structured merge-tree)

Patrick O'Neil et al., introduced in 1996 [54]
Used in: LevelDB, RocksDB, Cassandra, HBase, BigTable, InfluxDB, ScyllaDB, SQLite4, Tarantool, MongoDB (WiredTiger), etc.

Google’s Bigtable paper in 2006 [1, 53]

Memtable (B-tree, Skip List, etc)



https://www.themarysue.com/periodic-meme-table/

LSM-tree (Log-structured merge-tree)

Patrick O'Neil et al., introduced in 1996 [54]
Used in: LevelDB, RocksDB, Cassandra, HBase, BigTable, InfluxDB, ScyllaDB, SQLite4, Tarantool, MongoDB (WiredTiger), etc.

Google’s Bigtable paper in 2006 [1, 53]

Memtable (B-tree, Skip List, etc)
Sorted String Table (SSTable) - immutable

LSM-tree (Log-structured merge-tree)

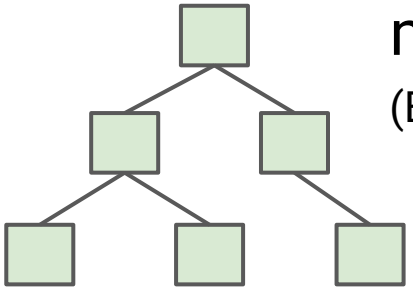
Patrick O'Neil et al., introduced in 1996 [54]
Used in: LevelDB, RocksDB, Cassandra, HBase, BigTable, InfluxDB, ScyllaDB, SQLite4, Tarantool, MongoDB (WiredTiger), etc.

Google’s Bigtable paper in 2006 [1, 53]

Memtable (B-tree, Skip List, etc)
Sorted String Table (SSTable) - immutable

Typically faster for writes [39]

LSM-tree



memtable

(B-tree, Skip List, etc)

Memory

Disk

SSTable index

key	file offset
aaaa	0
...	...

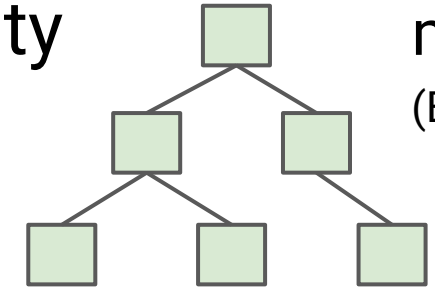
Bloom filter

0	1	0	1	1

SSTable data

key	value
aaaa	124
bbb	7351
...	...
zzzz	7

LSM-tree durability



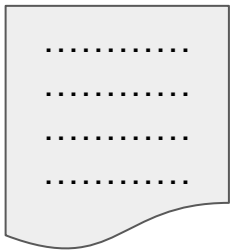
memtable

(B-tree, Skip List, etc)

Memory

Disk

Commit log



SSTable index

key	file offset
aaaa	0
...	...

Bloom filter

0	1	0	1	1

SSTable data

key	value
aaaa	124
bbb	7351
...	...
zzzz	7

SSTables compact & merge

SSTable (old)

key	value
aaaa	123
aab	1
...	...
zzzz	7

SSTable (new)

key	value
aaaa	124
foo	7351
...	...
zzzz	<deleted>

SSTable (merged)

key	value
aaaa	124
aab	1
foo	7351
...	...
yy	222

Leveled & Size-tiered compaction

Bloom filter [73]

Created by Burton Howard Bloom in 1970

Bloom filter [73]

Created by Burton Howard Bloom in 1970
Space-efficient probabilistic data structure

Bloom filter [73]

Created by Burton Howard Bloom in 1970
Space-efficient probabilistic data structure

Used by:
Google Bigtable, Apache HBase, Cassandra, and PostgreSQL [79]

Bloom filter [73]

Created by Burton Howard Bloom in 1970
Space-efficient probabilistic data structure

Used by:

Google Bigtable, Apache HBase, Cassandra, and PostgreSQL [79]

Akamai - to prevent "one-hit-wonders" from being stored in its disk caches

Bloom filter [73]

Created by Burton Howard Bloom in 1970
Space-efficient probabilistic data structure

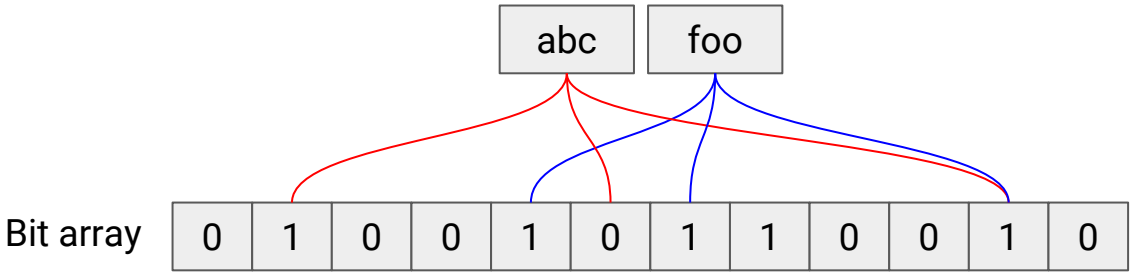
Used by:
Google Bigtable, Apache HBase, Cassandra, and PostgreSQL [79]
Akamai - to prevent "one-hit-wonders" from being stored in its disk caches
The Google Chrome - to identify malicious URLs

Bloom filter [73]

Created by Burton Howard Bloom in 1970
Space-efficient probabilistic data structure

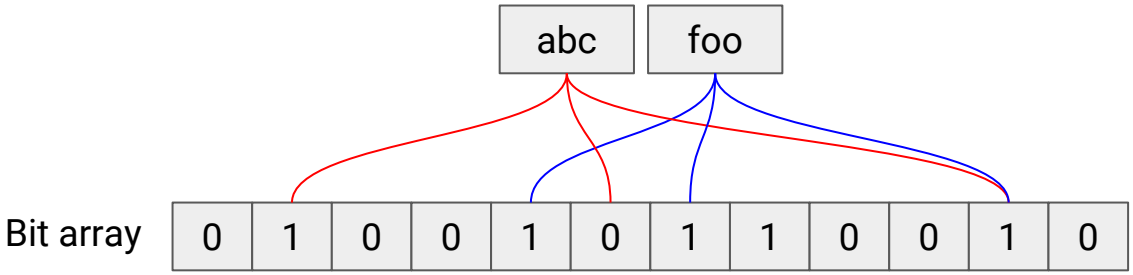
Used by:
Google Bigtable, Apache HBase, Cassandra, and PostgreSQL [79]
Akamai - to prevent "one-hit-wonders" from being stored in its disk caches
The Google Chrome - to identify malicious URLs
Medium - to avoid recommending articles a user has previously read

Bloom filter

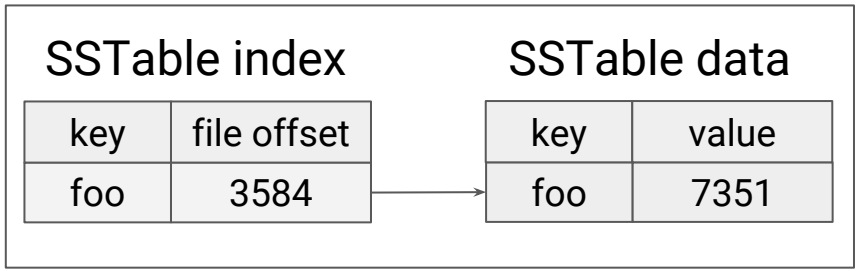


key “foo” probably exists in SSTable
key “abc” definitely not

Bloom filter



key “foo” probably exists in SSTable
try to get its value



SSTable

R-tree

R-tree

Proposed by Antonin Guttman in 1984 [82]

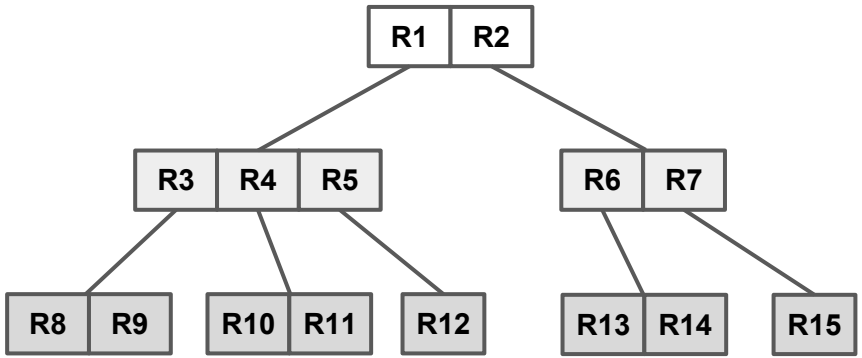
R-tree

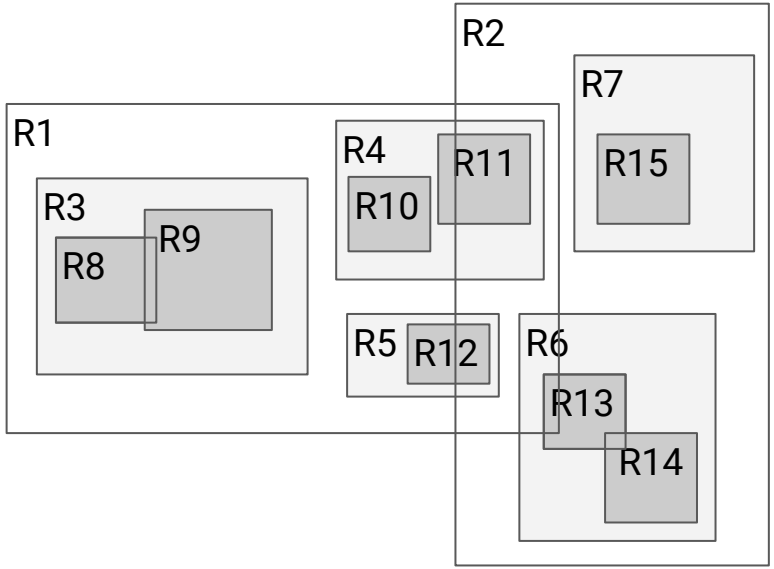
Proposed by Antonin Guttman in 1984 [82]
Tree data structure for indexing spatial information such as geographical coordinates, rectangles or polygons [82]

R-tree

Proposed by Antonin Guttman in 1984 [82]
Tree data structure for indexing spatial information such as geographical coordinates, rectangles or polygons [82]
Common operation on spatial data is a search for all objects in an area [83], e.g.: “Find all shops within 1 km of my current location”

R-tree


R1
R2
R3
R4
R5
R6
R7
R8
R9
R10
R11
R12
R13
R14
R15


R2
R7
R1
R4
R11
R15
R3
R10
R9
R8
R5
R12
R6
R13
R14

BRIN

Block Range Index (BRIN) [80]

Proposed by Alvaro Herrera of 2ndQuadrant in 2013 as Minmax index [80, 81]

Block Range Index (BRIN) [80]

Proposed by Alvaro Herrera of 2ndQuadrant in 2013 as Minmax index [80, 81]
Designed for large tables (best for ordered set)

Block Range Index (BRIN) [80]

Proposed by Alvaro Herrera of 2ndQuadrant in 2013 as Minmax index [80, 81]
Designed for large tables (best for ordered set)

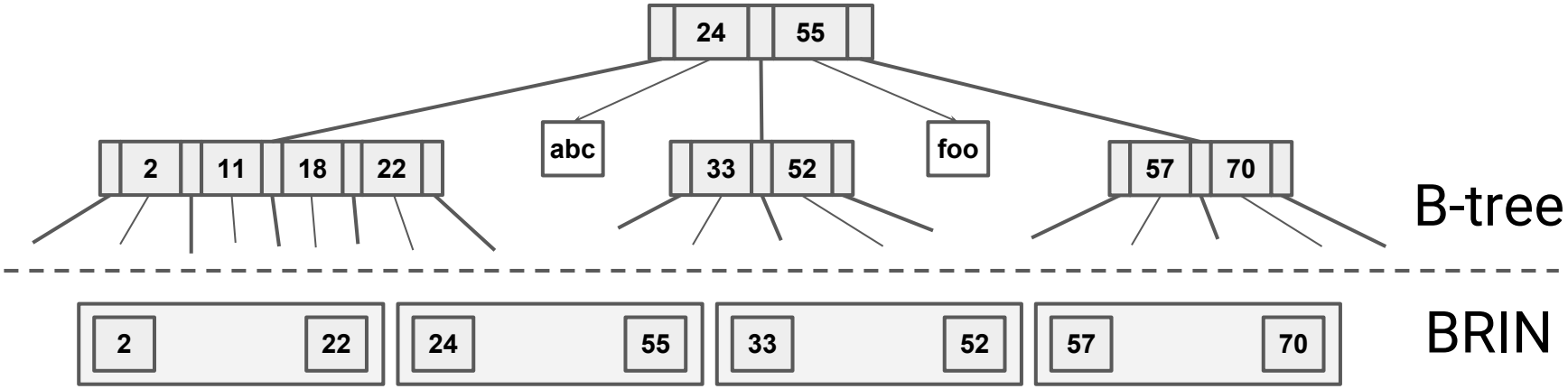
Used in: PostgreSQL

Block Range Index (BRIN) [80]

Proposed by Alvaro Herrera of 2ndQuadrant in 2013 as Minmax index [80, 81]
Designed for large tables (best for ordered set)

Used in: PostgreSQL
Other vendors have similar features: Oracle "storage indexes", Netezza “zone maps”, Infobright “data packs”, MonetDB, Apache Hive, ORC, Parquet [80, 81]

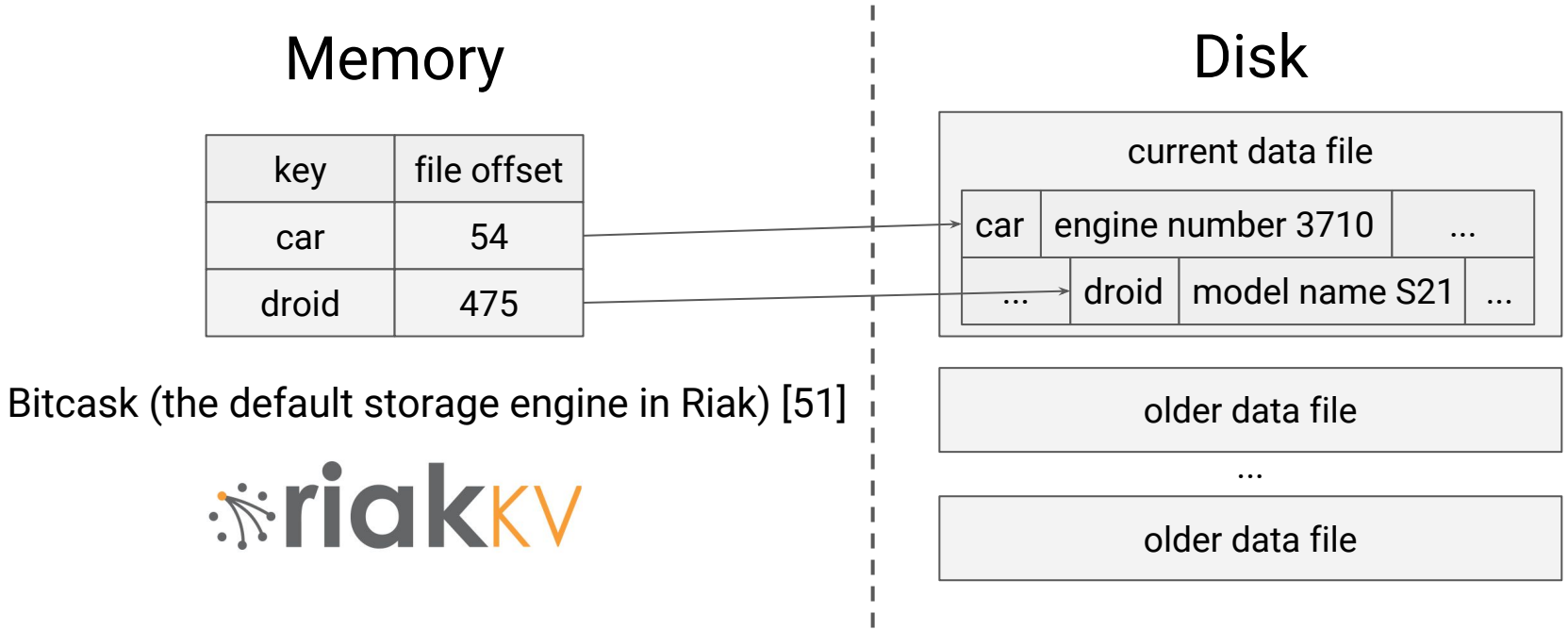
B-tree vs BRIN



block range	min value	max value
1	2	22
2	24	55
3	33	52
4	57	70

Log-Structured Hash Table

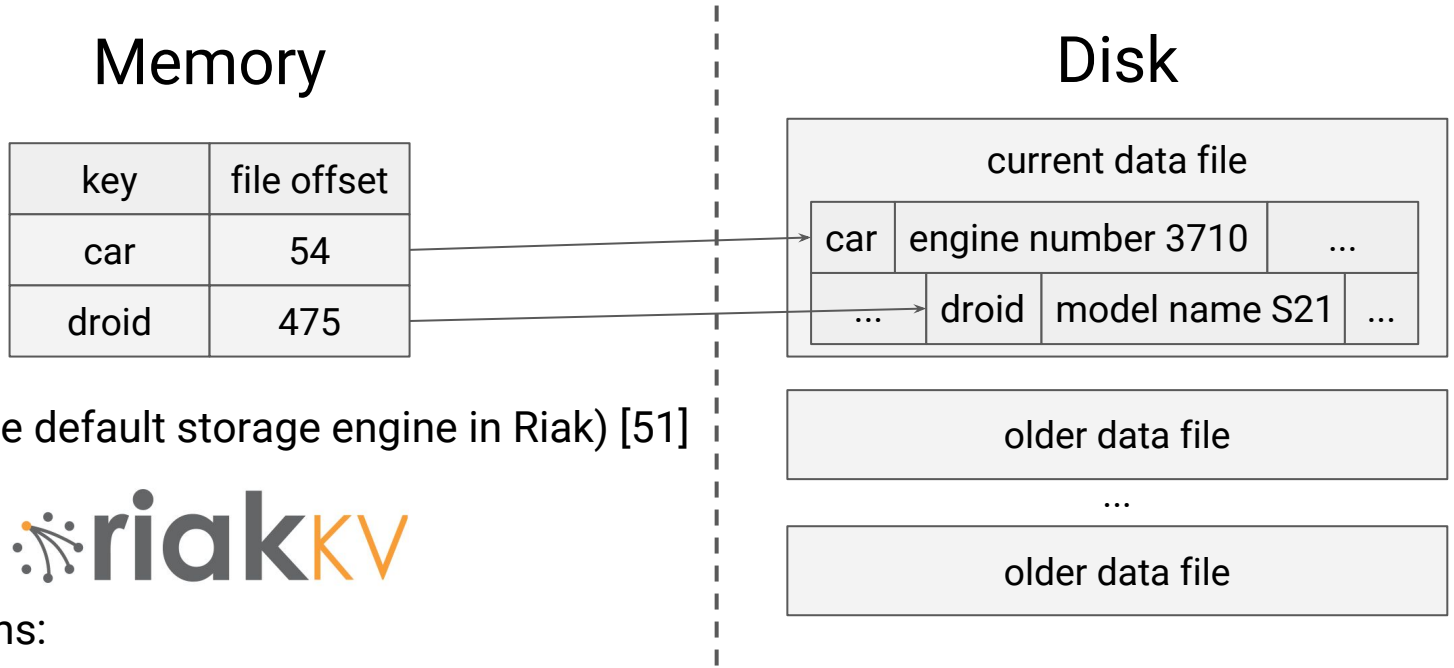
Log-Structured Hash Table



Bitcask (the default storage engine in Riak) [51]

riakKV

Log-Structured Hash Table



Bitcask (the default storage engine in Riak) [51]

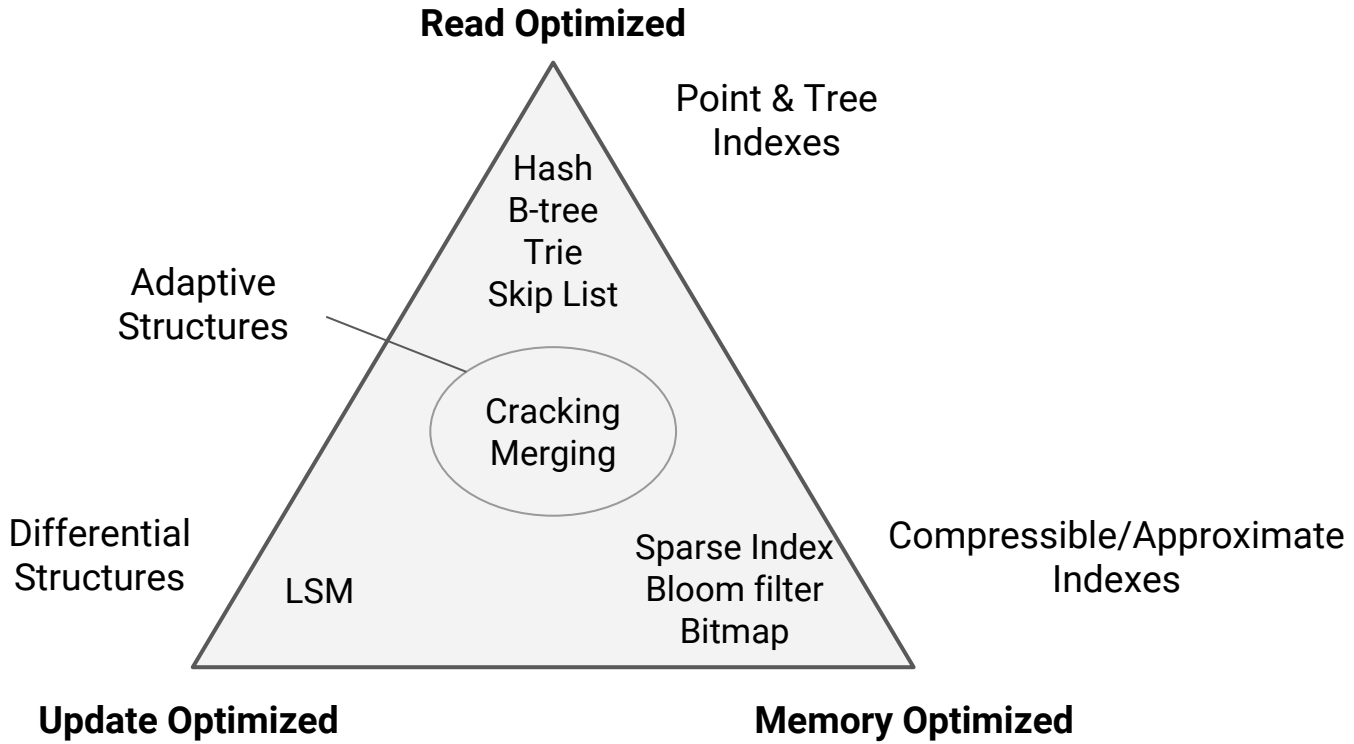
riakKV

Limitations:

1) must fit in memory
2) ranges not efficient

RUM Conjecture

RUM Conjecture [38, 39]



Amplifications

Amplifications

- Read amplification — amount of work done per logical read operation [49]

Amplifications

- Read amplification — amount of work done per logical read operation [49]
- Write amplification — amount of work done per write operation [49]
  Writing 1 byte -> writing a page (up to 16 KB for some models) [36]

Amplifications

- Read amplification — amount of work done per logical read operation [49]
- Write amplification — amount of work done per write operation [49]
  Writing 1 byte -> writing a page (up to 16 KB for some models) [36]
- Space amplification — ratio of the size of DB to the size of the data in DB [49]

Amplifications

- Read amplification — amount of work done per logical read operation [49]
- Write amplification — amount of work done per write operation [49]
  Writing 1 byte -> writing a page (up to 16 KB for some models) [36]
- Space amplification — ratio of the size of DB to the size of the data in DB [49]

The SPAce, Read Or Write theorem (SPARROW) [46]
RA is inversely related to WA, and WA is inversely related to SA

Amplifications

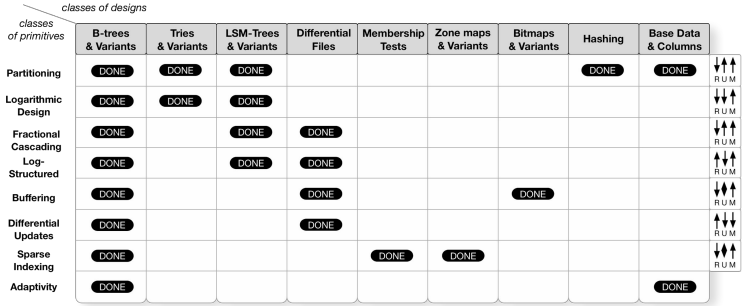
- Read amplification — amount of work done per logical read operation [49]
- Write amplification — amount of work done per write operation [49]
  Writing 1 byte -> writing a page (up to 16 KB for some models) [36]
- Space amplification — ratio of the size of DB to the size of the data in DB [49]

The SPAce, Read Or Write theorem (SPARROW) [46]
RA is inversely related to WA, and WA is inversely related to SA

Amplification and other issues are heavily dependent on workload, configuration of the engine, and the specific implementation [48]

Interesting projects

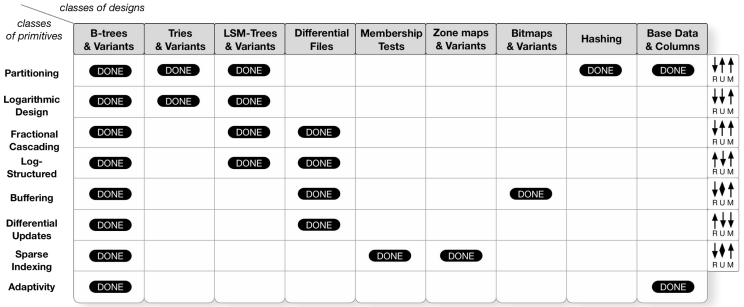
The periodic table of data structures [41]



Interesting projects

The periodic table of data structures [41]

Data calculator [42, 43]
Interactive, semi-automated design of data structures

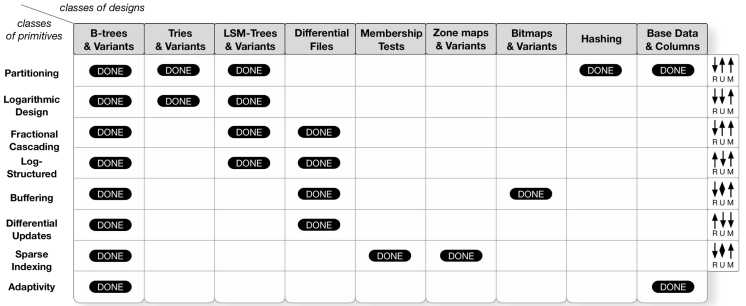


Interesting projects

The periodic table of data structures [41]

Data calculator [42, 43]
Interactive, semi-automated design of data structures

CrimsonDB [45]
A self-designing key-value store



OLTP/OLAP

OLTP vs OLAP

In the early days of business data processing, a write to the database typically corresponded to a commercial transactions [1]

OLTP vs OLAP

In the early days of business data processing, a write to the database typically corresponded to a commercial transactions [1]
Databases started being used for many different kinds of applications. Because applications are interactive, the access pattern became known as online transaction processing (OLTP)

OLTP vs OLAP

In the early days of business data processing, a write to the database typically corresponded to a commercial transactions [1]
Databases started being used for many different kinds of applications. Because applications are interactive, the access pattern became known as online transaction processing (OLTP)
Databases also started being increasingly used for data analytics

OLTP vs OLAP

In the early days of business data processing, a write to the database typically corresponded to a commercial transactions [1]
Databases started being used for many different kinds of applications. Because applications are interactive, the access pattern became known as online transaction processing (OLTP)
Databases also started being increasingly used for data analytics
Databases for online analytical processing (OLAP) was called a Data Warehouse

OLTP vs OLAP

In the early days of business data processing, a write to the database typically corresponded to a commercial transactions [1]
Databases started being used for many different kinds of applications. Because applications are interactive, the access pattern became known as online transaction processing (OLTP)
Databases also started being increasingly used for data analytics
Databases for online analytical processing (OLAP) was called a Data Warehouse
Hybrid transaction/analytical processing (HTAP) [84]

Row oriented vs Column oriented DBMS

datetime	name	age
10/10/2018 12:01:23	John	54
10/10/2018 12:01:24	Alice	22

Columns

datetime
10/10/2018 12:01:23
10/10/2018 12:01:24

name
John
Alice

age
54
22

Rows

datetime	name	age
10/10/2018 12:01:23	John	54
10/10/2018 12:01:24	Alice	22

Compression:
RLE, LZW, etc

Column oriented & time series DBs

Apache Parquet, ClickHouse, C-Store, Greenplum, MonetDB, Vertica, etc.

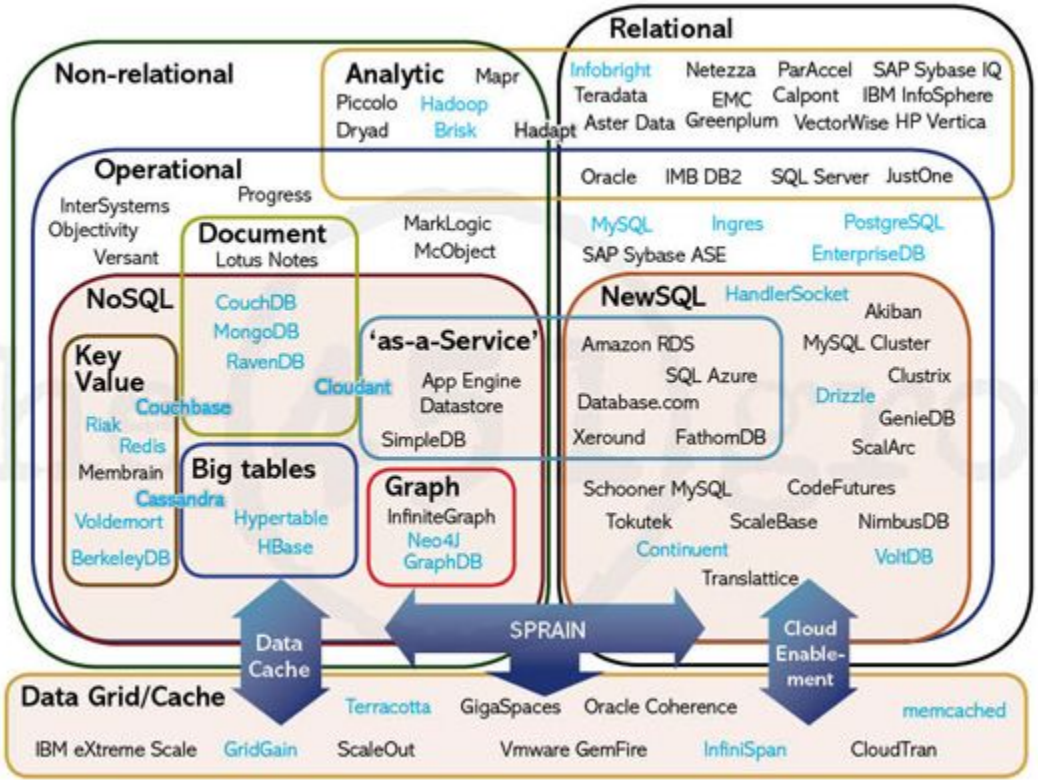
Time series databases (TSDB): Druid, Akumuli, InfluxDB, Riak TS, etc.

C-Store

SQL, NoSQL, NewSQL

RDBMS/SQL, NoSQL, NewSQL [72]

	RDBMS/SQL	NoSQL	NewSQL
Relational	Yes	No	Yes
ACID transactions	Yes	No	Yes
SQL support	Yes	No	Yes
Horizontal scalability	No	Yes	Yes
Schemaless	No	Yes	No



Matthew Aslett, The 451 Group [55]

DB in CPython

Python DB API Specification

PEP 248 - v1.0 (Release-Date: 09 Apr 1996 [74])

Python DB API Specification

PEP 248 - v1.0 (Release-Date: 09 Apr 1996 [74])
PEP 249 - v2.0 (Release-Date: 07 Apr 1999 [75])

Python DB API Specification

PEP 248 - v1.0 (Release-Date: 09 Apr 1996 [74])
PEP 249 - v2.0 (Release-Date: 07 Apr 1999 [75])

Implementations are available for:
- PostgreSQL (psycopg2, txpostgres, ...)
- MySQL (mysql-python, PyMySQL, ...)
- MS SQL Server (adodbapi, pymssql, mxODBC, pyodbc, ...)
- Oracle (cx_Oracle, mxODBC, pyodbc, ...)
- etc.

DBs in Python

dbm, gdbm or bsddb

dbm — interfaces to Unix “databases” [76]

DBs in Python

dbm, gdbm or bsddb
dbm — interfaces to Unix “databases” [76]

shelve
“shelf” — persistent, dictionary-like object
The values can be arbitrary Python objects — anything that the **pickle** module can handle, but the keys are strings [77]

DBs in Python

dbm, gdbm or bsddb
dbm — interfaces to Unix “databases” [76]

shelve
“shelf” — persistent, dictionary-like object
The values can be arbitrary Python objects — anything that the **pickle** module can handle, but the keys are strings [77]

https://pixnio.com/food-and-drink/bell-pepper-jar-carfiol-vegetable-food-diet-glass-organic

DBs in Python

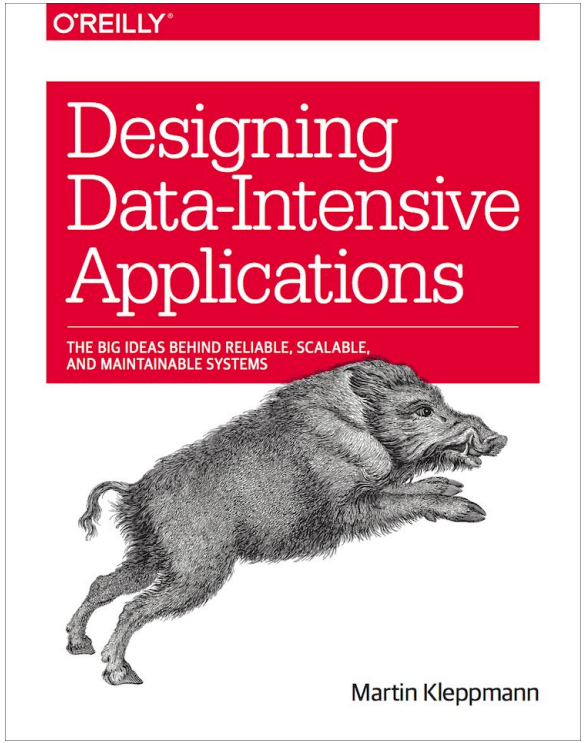
dbm, gdbm or bsddb
dbm — interfaces to Unix “databases” [76]

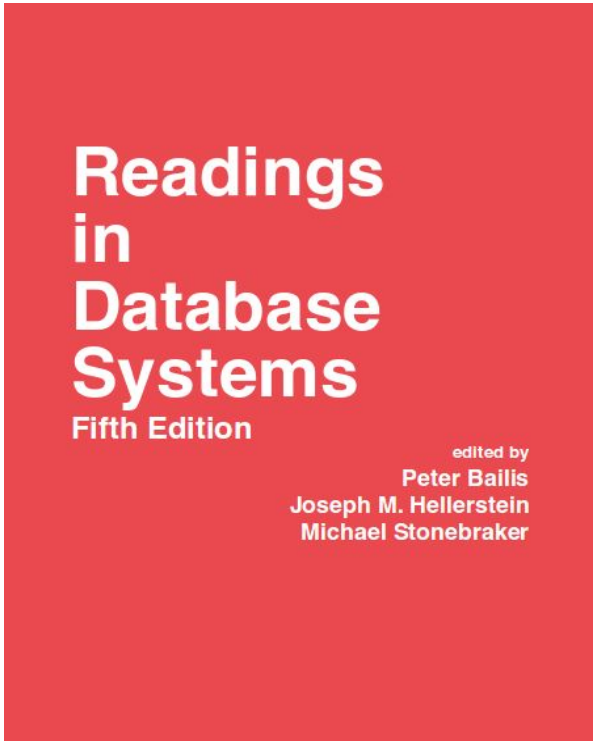
shelve
“shelf” — persistent, dictionary-like object
The values can be arbitrary Python objects — anything that the **pickle** module can handle, but the keys are strings [77]

sqlite3
sqlite3 — DB-API 2.0 interface for SQLite databases [78]

Summary

- OLTP and OLAP
- OLTP:
  - B-tree
  - LSM-tree
  - Other indices
  - RAM, SSD
- OLAP
  - Column-oriented storage
- RUM Conjecture
- Amplifications


O'REILLY
Designing Data-Intensive Applications
THE BIG IDEAS BEHIND RELIABLE, SCALABLE, AND MAINTAINABLE SYSTEMS
Martin Kleppmann


Readings in Database Systems
Fifth Edition
edited by
Peter Bailis
Joseph M. Hellerstein
Michael Stonebraker

Books [1, 28]

Thank you!

Happy databasing!

References

1. Martin Kleppmann: Designing Data-Intensive Applications: The Big Ideas Behind Reliable, Scalable, and Maintainable Systems, 1st edition. O'Reilly Media, 2017. ISBN: 978-1-449-37332-0 (https://dataintensive.net)
2. Alex Petrov: On Disk IO, Part 1: Flavors of IO, medium.com, September 3, 2017. (https://medium.com/databasss/on-disk-io-part-1-flavours-of-io-8e1ace1de017)
3. Alex Petrov: On Disk IO, Part 2: More Flavours of IO, medium.com, September 11, 2017. (https://medium.com/databasss/on-disk-io-part-2-more-flavours-of-io-c945db3edb13)
4. Alex Petrov: On Disk IO, Part 3: LSM Trees, medium.com, September 27, 2017. (https://medium.com/databasss/on-disk-io-part-3-lsm-trees-8b2da218496f)
5. Alex Petrov: On Disk IO, Part 4: B-Trees and RUM Conjecture, medium.com, October 4, 2017. (https://medium.com/databasss/on-disk-storage-part-4-b-trees-30791060741)
6. Alex Petrov: On Disk IO, Part 5: Access Patterns in LSM Trees, medium.com, October 30, 2017. (https://medium.com/databasss/on-disk-io-access-patterns-in-lsm-trees-2ba8dffc05f9)
7. Alex Petrov: Algorithms Behind Modern Storage Systems. Communications of the ACM, volume 61, number 8, pages 38-44, August 2018, doi:10.1145/3209210 (https://queue.acm.org/detail.cfm?id=3220266)

References

8. PostgreSQL 9.2.24 Documentation: Chapter 11. Indexes (https://www.postgresql.org/docs/9.2/static/indexes-types.html)
9. MySQL 8.0 Reference Manual: 15.8.2.2 The Physical Structure of an InnoDB Index (https://dev.mysql.com/doc/refman/8.0/en/innodb-physical-structure.html)
10. Oracle Database Concepts: Indexes and Index-Organized Tables (https://docs.oracle.com/cd/E11882_01/server.112/e40540/indexiot.htm#CNCPT1170)
11. SQL Server Index Design Guide (https://technet.microsoft.com/en-us/library/jj835095(v=sql.110).aspx)
12. IBM Knowledge Center: Table and index management for standard tables (https://www.ibm.com/support/knowledgecenter/en/SSEPGG_11.1.0/com.ibm.db2.luw.admin.perf.doc/doc/c0005424.html)
13. MariaDB Knowledge Base: Storage Engine Index Types (https://mariadb.com/kb/en/library/storage-engine-index-types/)
14. Wiredtiger: Btree vs LSM (https://github.com/wiredtiger/wiredtiger/wiki/Btree-vs-LSM/1cae5a2c73e938fa2095d900f8c25a9ee9a05412)
15. CouchDB The Definitive Guide: The Power of B-trees (http://guide.couchdb.org/draft/btree.html)

References

16. Architecture of SQLite (https://www.sqlite.org/arch.html)
17. The Couchbase Blog: Compaction magic in Couchbase Server 2.0 (https://blog.couchbase.com/compaction-magic-couchbase-server-20/)
18. Apache Cassandra 3.0: Storage engine (https://docs.datastax.com/en/cassandra/3.0/cassandra/dml/dmlManageOndisk.html)
19. Apache HBase: Accordion: HBase Breathes with In-Memory Compaction, blogs.apache.org, April 09, 2017. (https://blogs.apache.org/hbase/entry/accordion-hbase-breathes-with-in)
20. InfluxData Documentation: In-memory indexing and the Time-Structured Merge Tree (TSM) (https://docs.influxdata.com/influxdb/v1.6/concepts/storage_engine/#the-influxdb-storage-engine-and-the-time-structured-merge-tree-tsm)
21. Ilya Grigorik: SSTable and Log Structured Storage: LevelDB, igvita.com, February 06, 2012. (https://www.igvita.com/2012/02/06/sstable-and-log-structured-storage-leveldb/)
22. RocksDB Basics (https://github.com/facebook/rocksdb/wiki/RocksDB-Basics/8e2e3f69e163fbc370b13c3d2baf8ecf798f85e5)
23. SSTable compaction and compaction strategies (https://github.com/scylladb/scylla/wiki/SSTable-compaction-and-compaction-strategies/419412878eea8a9f9775fb718eda2fed2c1d551b)

References

24. Nadav Har'El: Scylla's Compaction Strategies Series: Write Amplification in Leveled Compaction, scylladb.com, January 31, 2018. (https://www.scylladb.com/2018/01/31/compaction-series-leveled-compaction/)
25. Vinyl Architecture (https://github.com/tarantool/tarantool/wiki/Vinyl-Architecture/c83dec9b0719478ef24d6407ba6583faf6ae4547)
26. Tarantool: Storage engines (https://www.tarantool.io/en/doc/1.9/book/box/engines/)
27. SQLite4: LSM Users Guide (https://sqlite.org/src4/doc/trunk/www/lsmusr.wiki)
28. Peter Bailis, Joseph M. Hellerstein, Michael Stonebraker: Readings in Database Systems, 5th Edition, 2015. (http://www.redbook.io/)
29. Douglas Comer: The Ubiquitous B-Tree, ACM Computing Surveys, volume 11, number 2, pages 121–137, June 1979. doi:10.1145/356770.356776 (http://www.ezdoum.com/upload/14/20020512204603/TheUbiquitousB-Tree.pdf)
30. Yinan Li, Bingsheng He, Robin Jun Yang, et al.: Tree Indexing on Solid State Drives, Proceedings of the VLDB Endowment, volume 3, number 1, pages 1195–1206, September 2010. (http://www.vldb.org/pvldb/vldb2010/papers/R106.pdf)

References

31. Jim Gray, Andreas Reuter: Transaction processing: concepts and techniques, Morgan Kaufmann, 1992. ISBN: 978-1-55860-190-1
32. Goetz Graefe: Modern B-Tree Techniques, Now Publishers Inc, 2011. ISBN: 978-1-60198-482-1
33. Niv Dayan: Log-Structured-Merge Trees, Comp115 guest lecture, February 23, 2017. (http://manos.athanassoulis.net/classes/Comp115-Spring2017/slides/Comp115-Guest-Lecture-LSM-Trees.pdf)
34. Memory Prices 1957 to 2018 (https://jcmit.net/memoryprice.htm)
35. Yinan Li, Bingsheng He, Robin Jun Yang, et al.: Tree Indexing on Solid State Drives, Proceedings of the VLDB Endowment, volume 3, number 1, pages 1195–1206, September 2010. (http://www.vldb.org/pvldb/vldb2010/papers/R106.pdf)
36. Emmanuel Goossaert: Coding for SSDs, codecapsule.com, February 12, 2014. (http://codecapsule.com/2014/02/12/coding-for-ssds-part-1-introduction-and-table-of-contents/)
37. Write amplification, en.wikipedia.org. https://en.wikipedia.org/wiki/Write_amplification
38. The RUM Conjecture, daslab.seas.harvard.edu, 2016. http://daslab.seas.harvard.edu/rum-conjecture/
39. Manos Athanassoulis, Michael S. Kester, Lukas M. Maas, et al.: Designing Access Methods: The RUM Conjecture, at 19th International Conference on Extending Database Technology (EDBT), March 2016. doi:10.5441/002/edbt.2016.42 (http://openproceedings.org/2016/conf/edbt/paper-12.pdf)

References

40. Гипотеза RUM, delimitry.blogspot.com, March 13, 2018. (http://delimitry.blogspot.com/2018/03/rum.html)
41. S. Idreos, et al., The Periodic Table of Data Structures, Bulletin of the IEEE Computer Society Technical Committee on Data Engineering, vol. 41, no. 3, pp. 64-75, 2018. (http://sites.computer.org/debull/A18sept/p64.pdf)
42. S. Idreos, K. Zoumpatianos, B. Hentschel, M. Kester, and D. Guo: The Data Calculator: Data Structure Design and Cost Synthesis from First Principles and Learned Cost Models, SIGMOD, 2018.
43. Data Calculator, daslab.seas.harvard.edu, 2018. (http://daslab.seas.harvard.edu/datacalculator/)
44. Периодическая таблица структур данных, delimitry.blogspot.com, October 14, 2018. (http://delimitry.blogspot.com/2018/10/blog-post.html)
45. CrimsonDB: A Self-Designing Key-Value Store, daslab.seas.harvard.edu, 2018. (http://daslab.seas.harvard.edu/projects/crimsondb/)
46. The SPARROW Theorem for performance of storage systems, rocksdb.blogspot.com, October 16, 2013. (http://rocksdb.blogspot.com/2013/10/SparrowTheorem.html)
47. Goetz Graefe: Modern B-Tree Techniques, Foundations and Trends in Databases, volume 3, number 4, pages 203–402, August 2011. doi:10.1561/1900000028 (http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.219.7269&rep=rep1&type=pdf)

References

48. Baron Schwartz: The State of the Storage Engine, dzone.com, March 16, 2015. (https://dzone.com/articles/state-storage-engine)
49. Mark Callaghan: Read, write & space amplification - pick 2, smalldatum.blogspot.com, November 23, 2015. (http://smalldatum.blogspot.com/2015/11/read-write-space-amplification-pick-2_23.html)
50. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein: Introduction to Algorithms, 3rd edition. MIT Press, 2009. ISBN: 978-0-262-53305-8
51. Justin Sheehy and David Smith: Bitcask: A Log-Structured Hash Table for Fast Key/Value Data, Basho Technologies, April 2010. (http://basho.com/wp-content/uploads/2015/05/bitcask-intro.pdf)
52. D. Severance/G. Lohman: Differential Files: Their Applications to the Maintenance of Large Databases, ACM Transactions on Database Systems, volume 1, number 3, pages 256–367, September 1976. doi:10.1145/320473.320484
53. Fay Chang, Jeffrey Dean, Sanjay Ghemawat, et al.: Bigtable: A Distributed Storage System for Structured Data, at 7th USENIX Symposium on Operating System Design and Implementation (OSDI), November 2006. (https://ai.google/research/pubs/pub27898)

References

54. Patrick O'Neil, Edward Cheng, Dieter Gawlick, and Elizabeth O'Neil: The Log-Structured Merge-Tree (LSM-Tree), Acta Informatica, volume 33, number 4, pages 351–385, June 1996. doi:10.1007/s002360050048 (https://www.cs.umb.edu/~poneil/lsmtree.pdf)
55. Matthew AslettNoSQL: NewSQL and Beyond: The answer to SPRAINed relational databases, blogs.the451group.com, April 15th, 2011. (https://blogs.the451group.com/information_management/2011/04/15/nosql-newsql-and-beyond/)
56. Brendan Gregg: Systems Performance: Enterprise and the Cloud, Prentice Hall, October 2013. ISBN: 978-0-13-339009-4
57. Latency Numbers Every Programmer Should Know (https://gist.github.com/jboner/2841832)
58. Peter Norvig: Teach Yourself Programming in Ten Years (http://norvig.com/21-days.html#answers)
59. Latency Numbers Every Programmer Should Know (Interactive Latency) (https://people.eecs.berkeley.edu/~rcs/research/interactive_latency.html)
60. Apache Parquet (https://parquet.apache.org/)
61. ClickHouse (https://clickhouse.yandex/)
62. C-Store (http://db.csail.mit.edu/projects/cstore/)

References

63. Apache Druid (http://druid.io/)
64. Greenplum (https://greenplum.org/)
65. MonetDB (https://www.monetdb.org/)
66. Vertica (https://www.vertica.com/overview/)
67. Akumuli (https://akumuli.org/)
68. InfluxDB (https://www.influxdata.com/)
69. Riak TS (http://basho.com/products/riak-ts/)
70. Eric Evans: NOSQL 2009, blog.sym-link.com, May 12, 2009. (http://blog.sym-link.com/2009/05/12/nosql_2009.html)
71. Eric Evans: NoSQL: What's in a Name?, blog.sym-link.com, October 30, 2009. (http://blog.sym-link.com/2009/10/30/nosql_whats_in_a_name.html)
72. Laurent Guérin: NewSQL: what's this?, labs.sogeti.com, January 22, 2014. (http://labs.sogeti.com/newsql-whats/)
73. Burton H. Bloom: Space/Time Trade-offs in Hash Coding with Allowable Errors, Communications of the ACM, volume 13, number 7, pages 422–426, July 1970. doi:10.1145/362686.362692 (http://www.cs.upc.edu/~diaz/p422-bloom.pdf)

References

74. http://svn.python.org/projects/peps/tags/date2001-07-16/pep-0248.txt (https://www.python.org/dev/peps/pep-0249/)
75. http://svn.python.org/projects/peps/tags/date2001-07-16/pep-0249.txt (https://www.python.org/dev/peps/pep-0249/)
76. Python 3.7.1rc1 documentation: dbm — Interfaces to Unix “databases” (https://docs.python.org/3/library/dbm.html)
77. Python 3.7.1rc1 documentation: shelve — Python object persistence (https://docs.python.org/3/library/shelve.html)
78. Python 3.7.1rc1 documentation: sqlite3 — DB-API 2.0 interface for SQLite databases (https://docs.python.org/3/library/sqlite3.html)
79. Bloom filter (https://en.wikipedia.org/wiki/Bloom_filter)
80. Block Range Index (https://en.wikipedia.org/wiki/Block_Range_Index)
81. PostgreSQL: [RFC] Minmax indexes (https://www.postgresql.org/message-id/20130614222805.GZ5491@eldon.alvh.no-ip.org)

References

82. R-tree (https://en.wikipedia.org/wiki/R-tree)
83. Antonin Guttman: R-Trees: A Dynamic Index Structure for Spatial Searching. SIGMOD '84 Proceedings of the 1984 ACM SIGMOD international conference on Management of data, pp. 47-57, 1984. doi:10.1145/602259.602266. ISBN: 0-89791-128-8 (http://www-db.deis.unibo.it/courses/SI-LS/papers/Gut84.pdf)
84. Hybrid transactional/analytical processing (HTAP) (https://en.wikipedia.org/wiki/Hybrid_transactional/analytical_processing_(HTAP))

Questions