TARANTOOL

Vinyl: why we wrote our own write-optimized storage engine rather than chose RocksDB

kostja@tarantool.org
Konstantin Osipov

Plan

- A review of log structured merge tree data structure
- Vinyl engine in Tarantool
- Configuration parameters
- Key use cases

Why build, rather than buy: rocksdb, forestdb, ...

I/O

- session
- parser

Transaction Control

- read your own writes
- multi-engine
- undo log

Write ahead log

- redo log
- async relay
- group replication

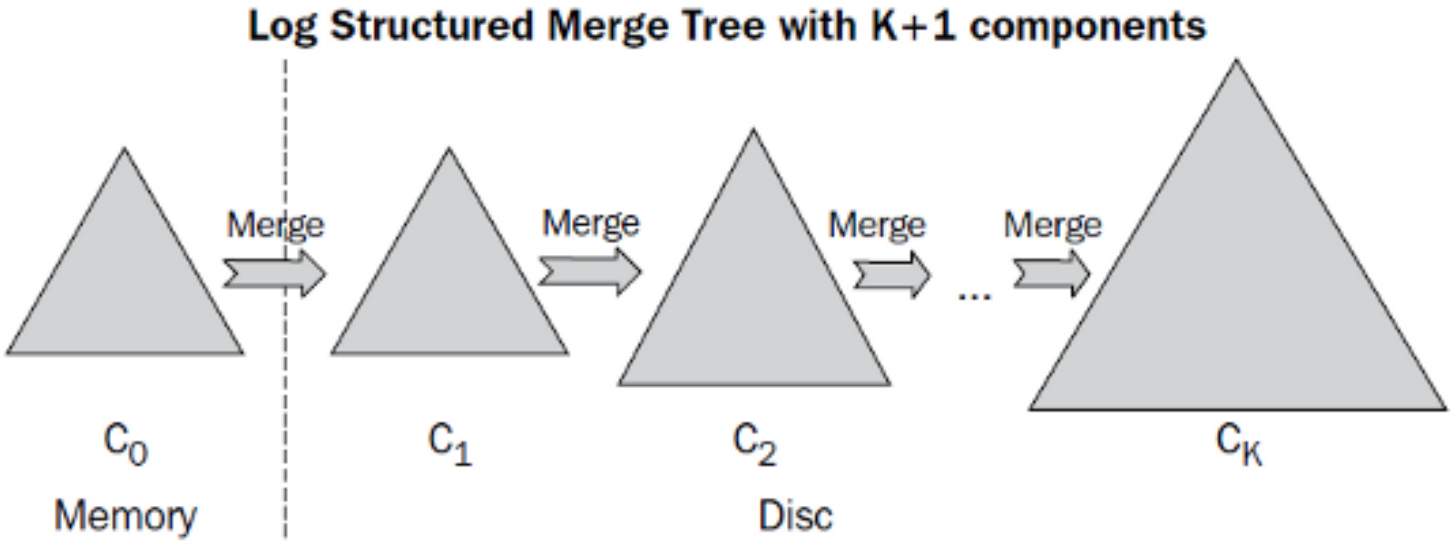
Engine 1

indexing
checkpoint
storage mgmt

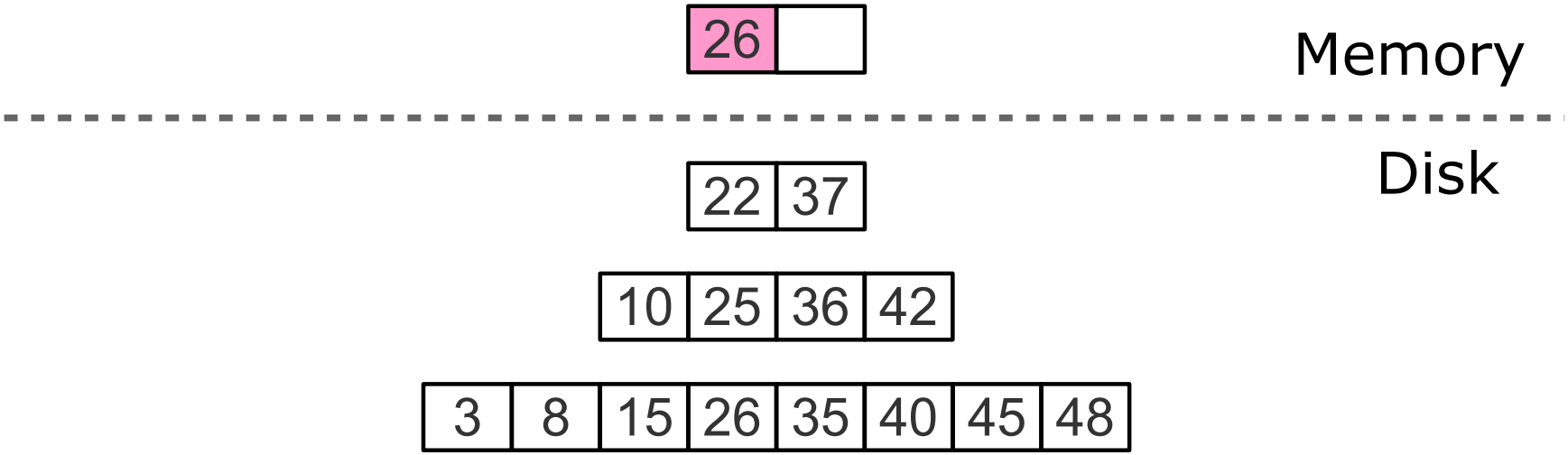
Engine 2

…
…
...

The shape of a classical LSM

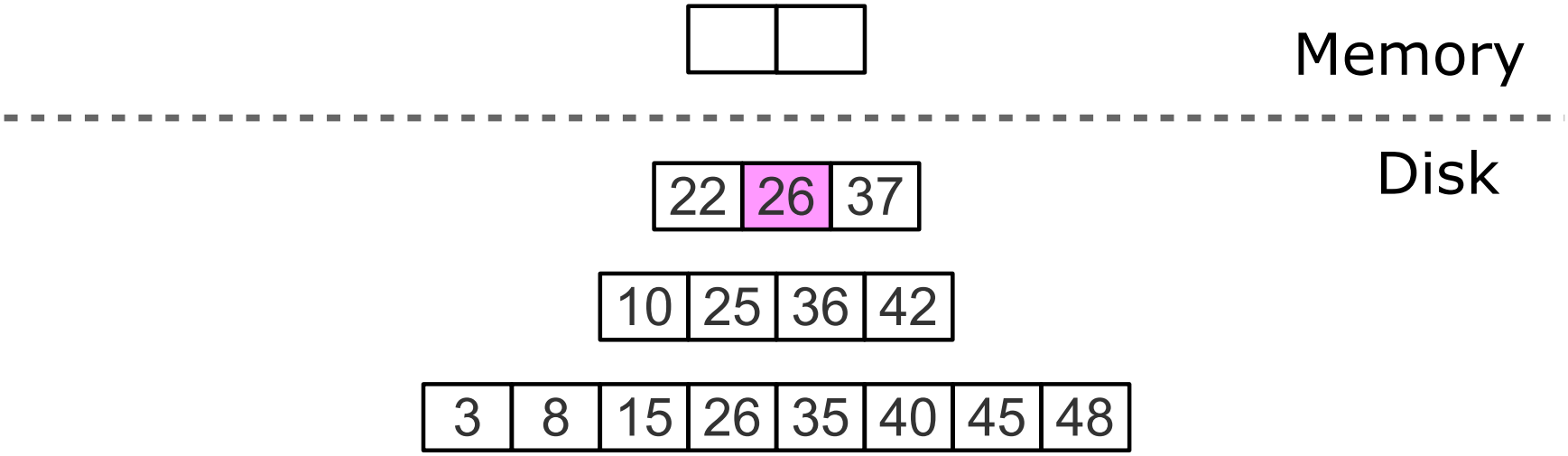


DELETE in an LSM tree

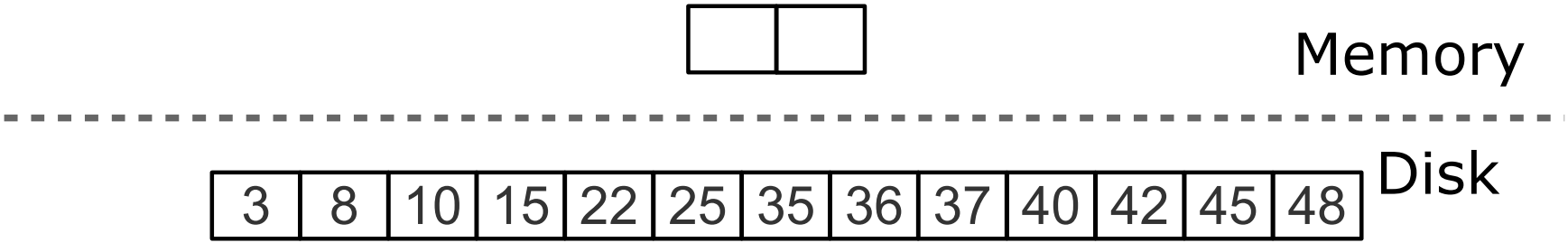


- We can’t delete from append-only files
- Tombstones (delete markers) are inserted into L0 instead

DELETE in an LSM tree



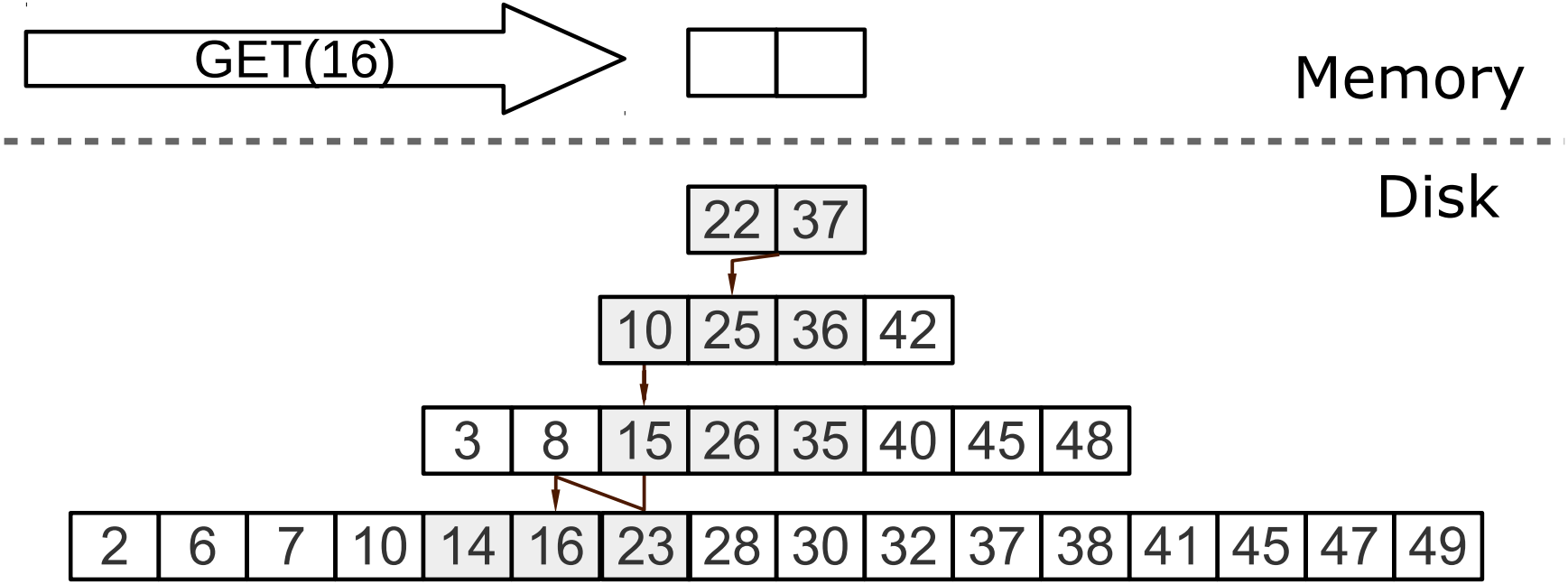
DELETE in an LSM tree



SELECT in LSM tree

- Search in all levels, until the key is found
- Optimistic for point queries
- Merge sort in case of range select

SELECT in LSM tree

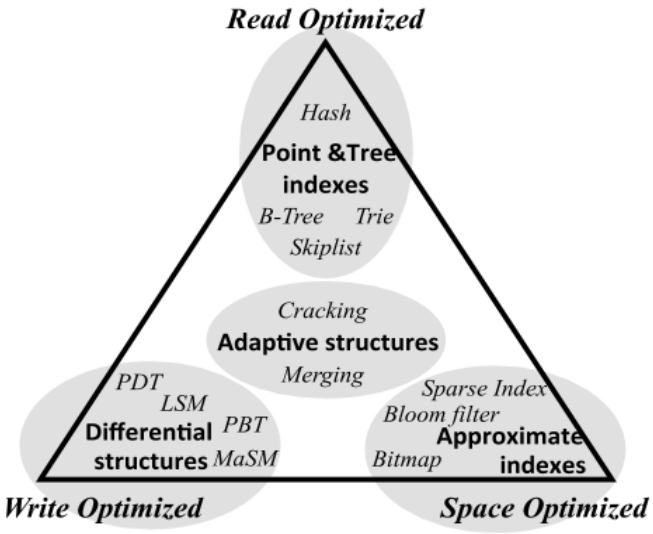


LSM: the algorithmic limit

	LSM tree	**B-tree**
Search	$K * O(\log_2 N/B)$	$O(\log_B N)$
Delete	$O(\log_2(N)/B)$	$O(\log_B N)$
Insert	$O(\log_2(N)/B)$	$O(\log_B N)$

RUM conjecture

The ubiquitous fight between the Read, the Update, and the Memory overhead of access methods for modern data systems



Key LSM challenges in Web/OLTP

- Slow reads: **read amplification**
- Potentially write the same data multiple times: **write amplification**
- Keep garbage around: **space amplification**
- Response times affected by background activity: **latency spikes**

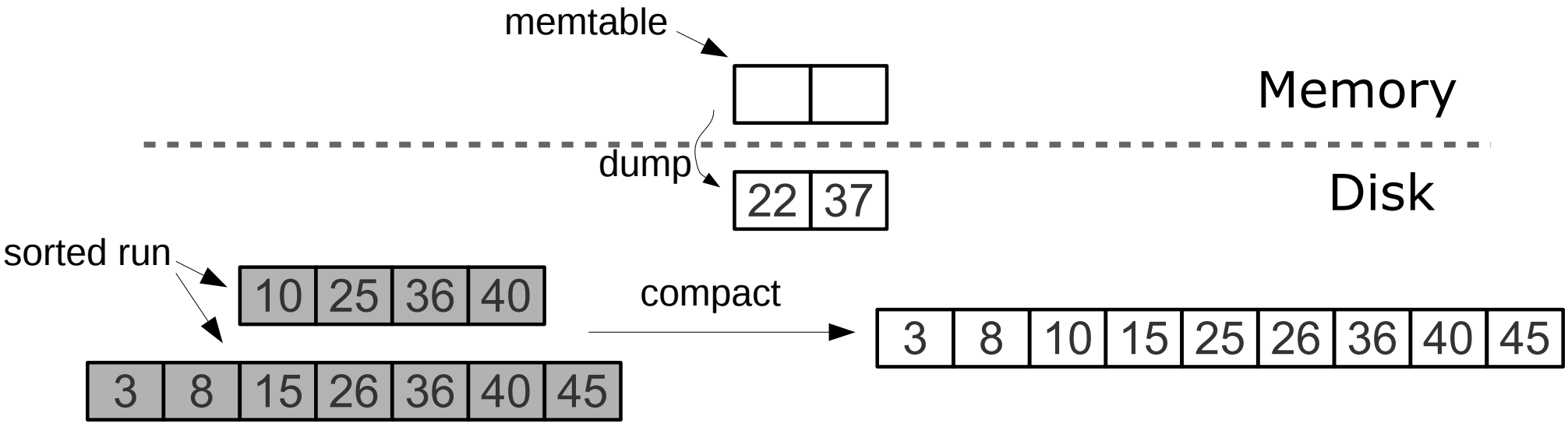
Vinyl: memtable, sorted run, dump & compact

- Stores statements, not values:
  - REPLACE,
  - DELETE,
  - UPSERT

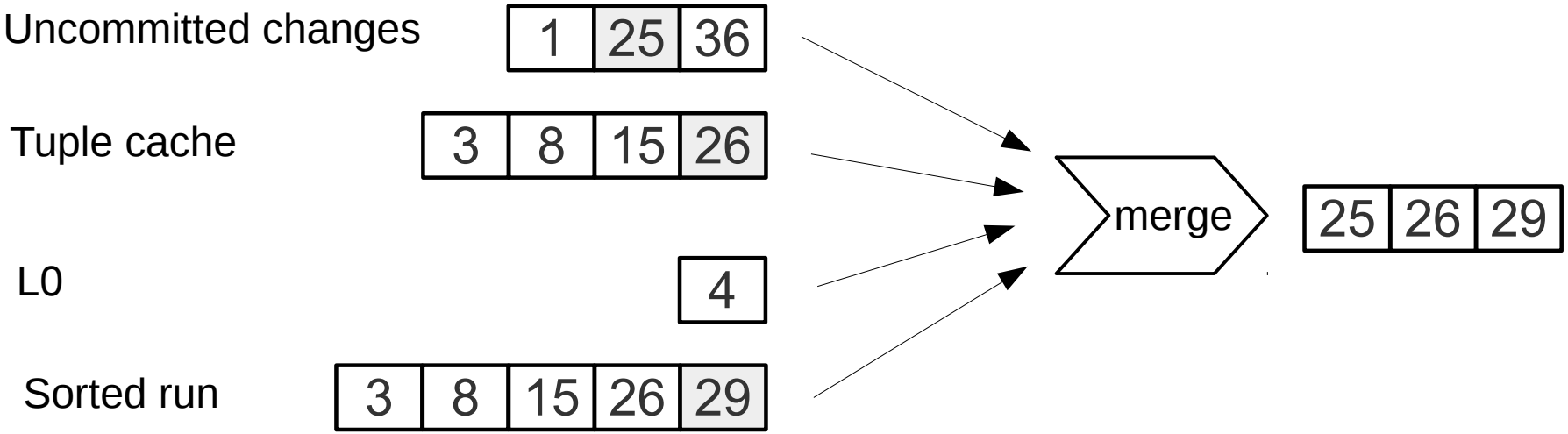
key	lsn	op_code	value

- Every statement is marked by LSN
- Append-only files, garbage collected after checkpoint
- Transactional log of all filesystem changes: vylog

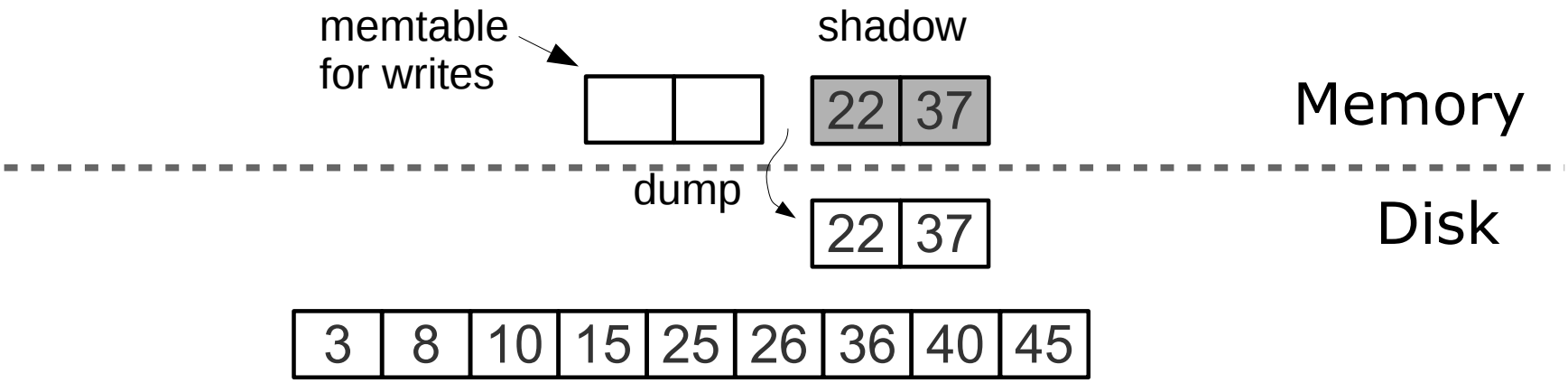
Memtable, sorted run, dump & compact



Vinyl: read [25, 30)

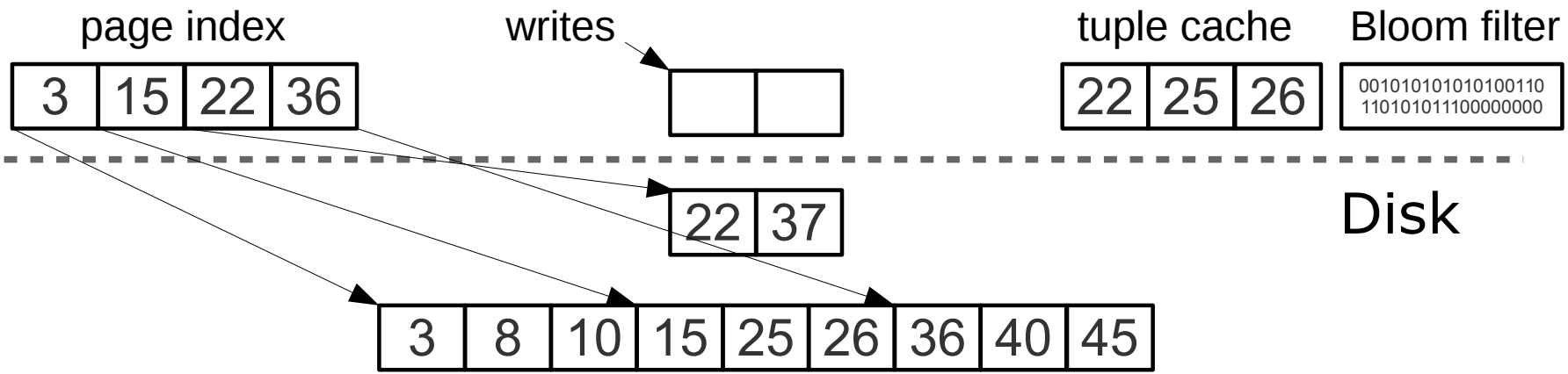


Keeping the latency within limits



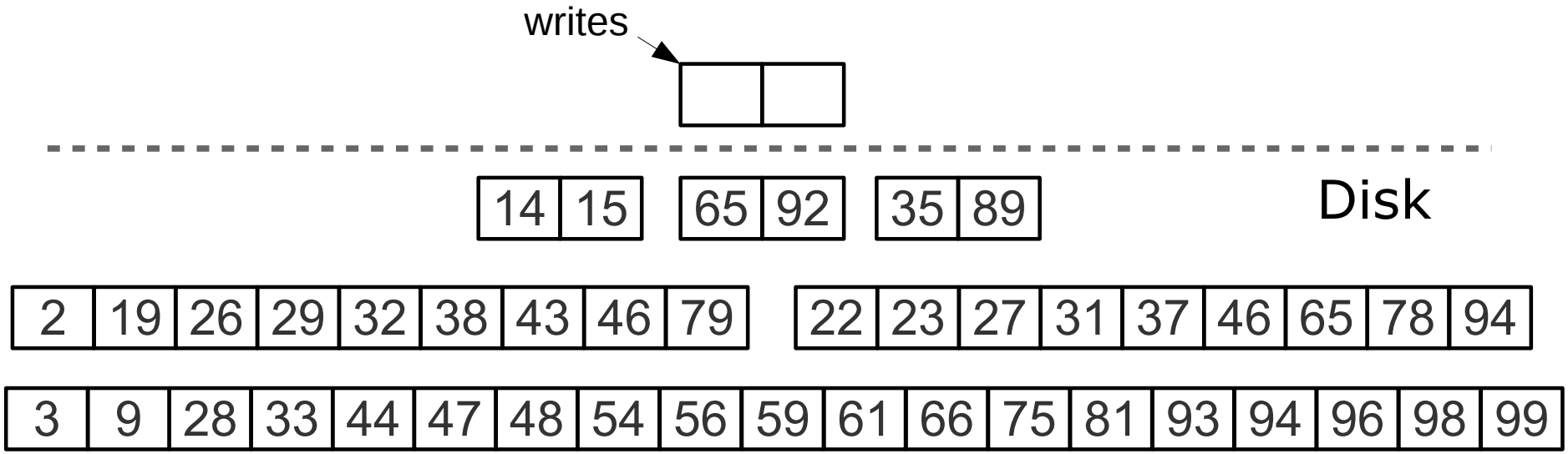
- anticipatory dump
- throttling

Reducing read amplification



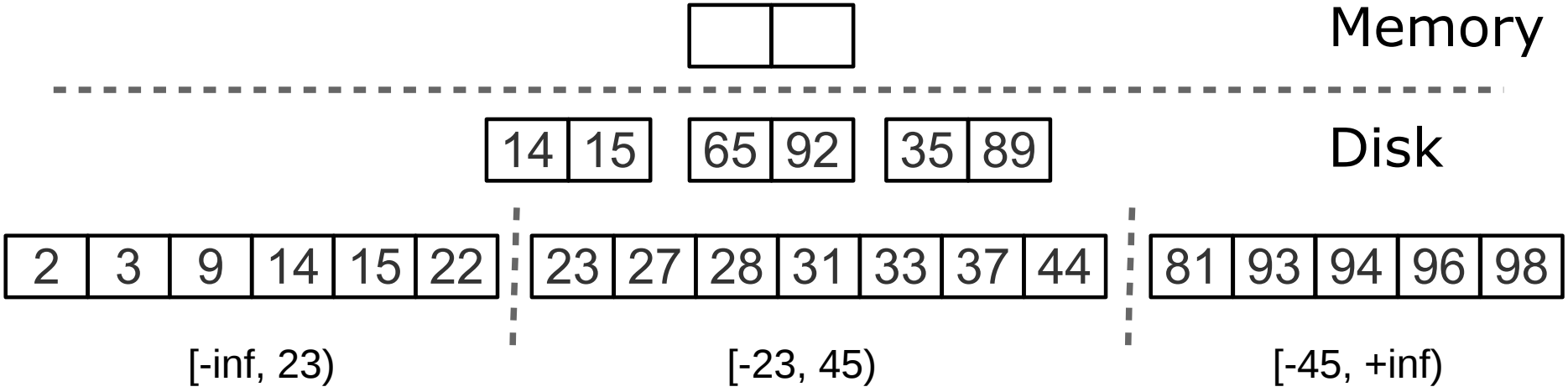
- page index
- bloom filters
- tuple range cache
- multi-level compaction

Reducing write amplification



- Multi-level compaction can span any number of levels
- A level can contain multiple runs

Reducing space amplification



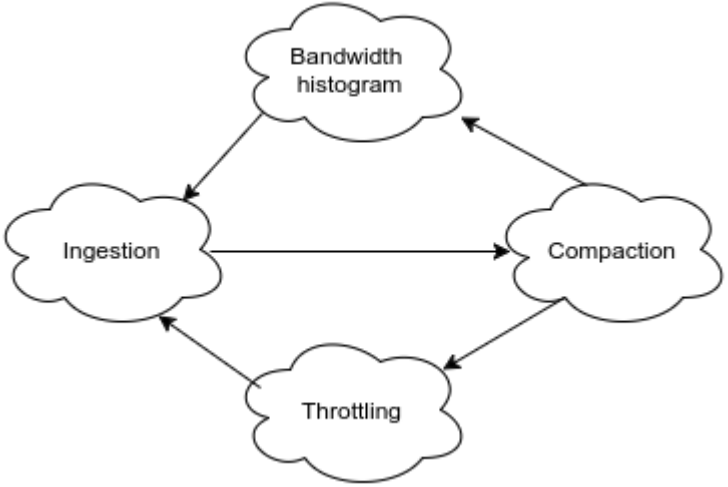
- Ranges reflect a static layout of sorted runs
- Slices connect a sorted run into a range

Secondary key support

- L0 and tuple cache use memtx technology
- This makes it possible to include L0 and tuple cache in checkpoint, so no cold restart
- L1+ stores the value of primary key as tuple id
- Unique secondary keys rely on bloom filter heavily for INSERT, REPLACE
- REPLACE in a non-unique secondary is a blind write, garbage is pruned at compaction

The scheduler

- The only active entity inside vinyl engine
- Maintains two queues: compact and dump
- Each range is in each queue, parallel compaction
- Dump always trumps compact
- Chunked garbage collection for quick memory reclamation
- Is a fiber, so needs no locks



Transactions

- MVCC
- the first transaction to commit wins
- no waits, no deadlocks, only aborts
- yields don’t abort transactions

Replication & backup

Work out of the box

Limitations

- Tuple size is up to 16M
- You need ~5k file descriptors per 1TB of data
- Optimistic transaction control is bad for long read-write transactions
- No cross-engine transactions yet

Configuration

Name	Description	Default
bloom_fpr	Bloom filter max false positive rate	5%
page_size	Approximate page size	8192
range_size	Approximate range size	1G
run_count_per_level	How many sorted runs are allowed on a single level	2
run_size_ratio	Run size ratio between levels	3.5
memory	L0 total size	0.25G
cache	Tuple cache size	0.25G
threads	The number of compaction threads	2

UPSERT

```
space:upsert(tuple, {{operator, field, value}, ...})
```

- Non-reading update or insert
- Delayed execution
- Background upsert squashing prevents upserts from piling up

Thank you! Questions?

@kostja_osipov

fb.com/TarantoolDatabase

kostja@tarantool.org

www.tarantool.org

Links

Leveled compaction in Apache Cassandra
http://www.datastax.com/dev/blog/leveled-compaction-in-apache-cassandra
http://www.scylladb.com/kb/compaction/
https://github.com/facebook/rocksdb/wiki/Universal-Compaction
https://dom.as/2015/04/09/how-innodb-lost-its-advantage/