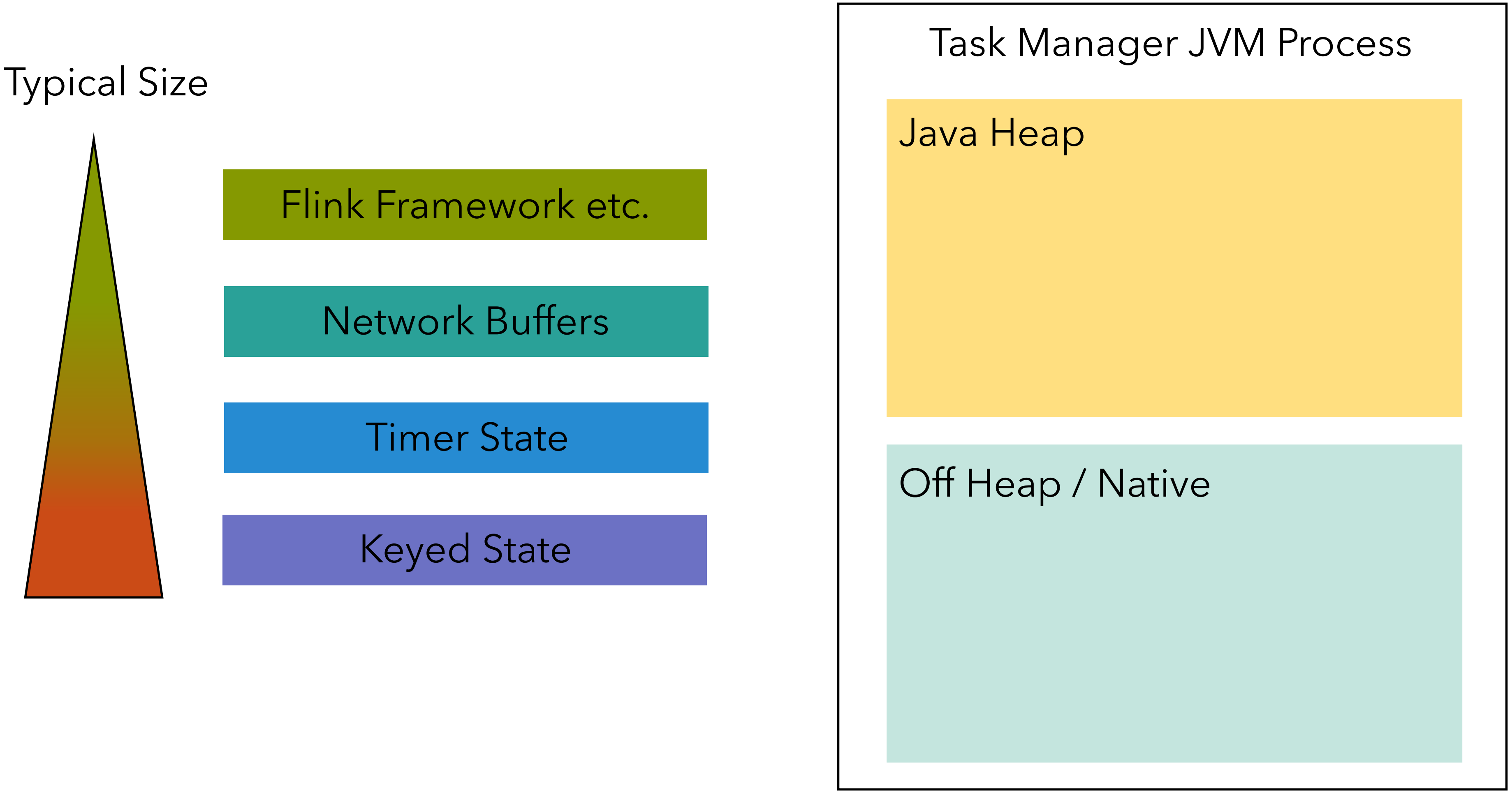
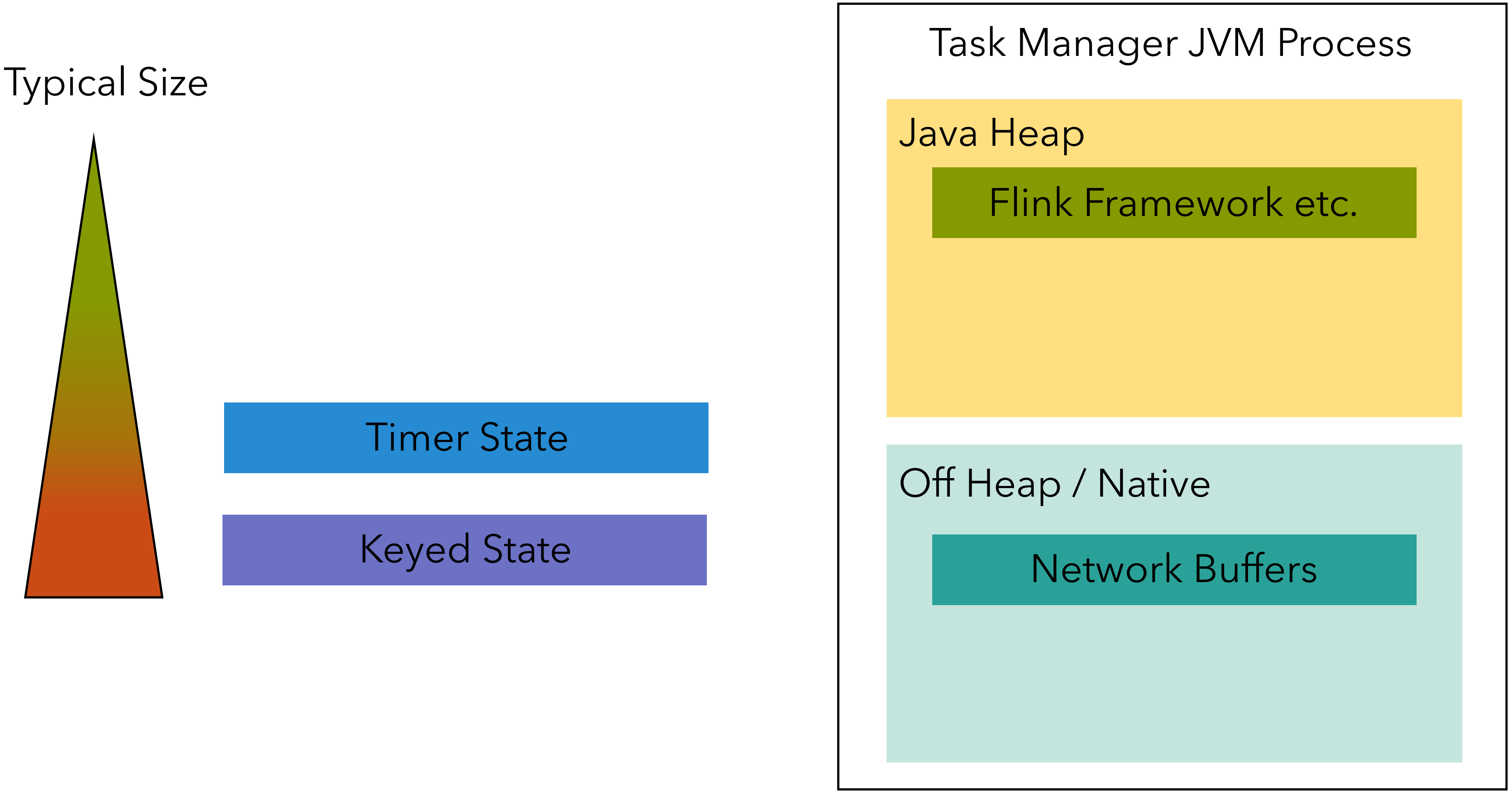
Tuning Statebackends

Seth Wiesman, Solutions Architect

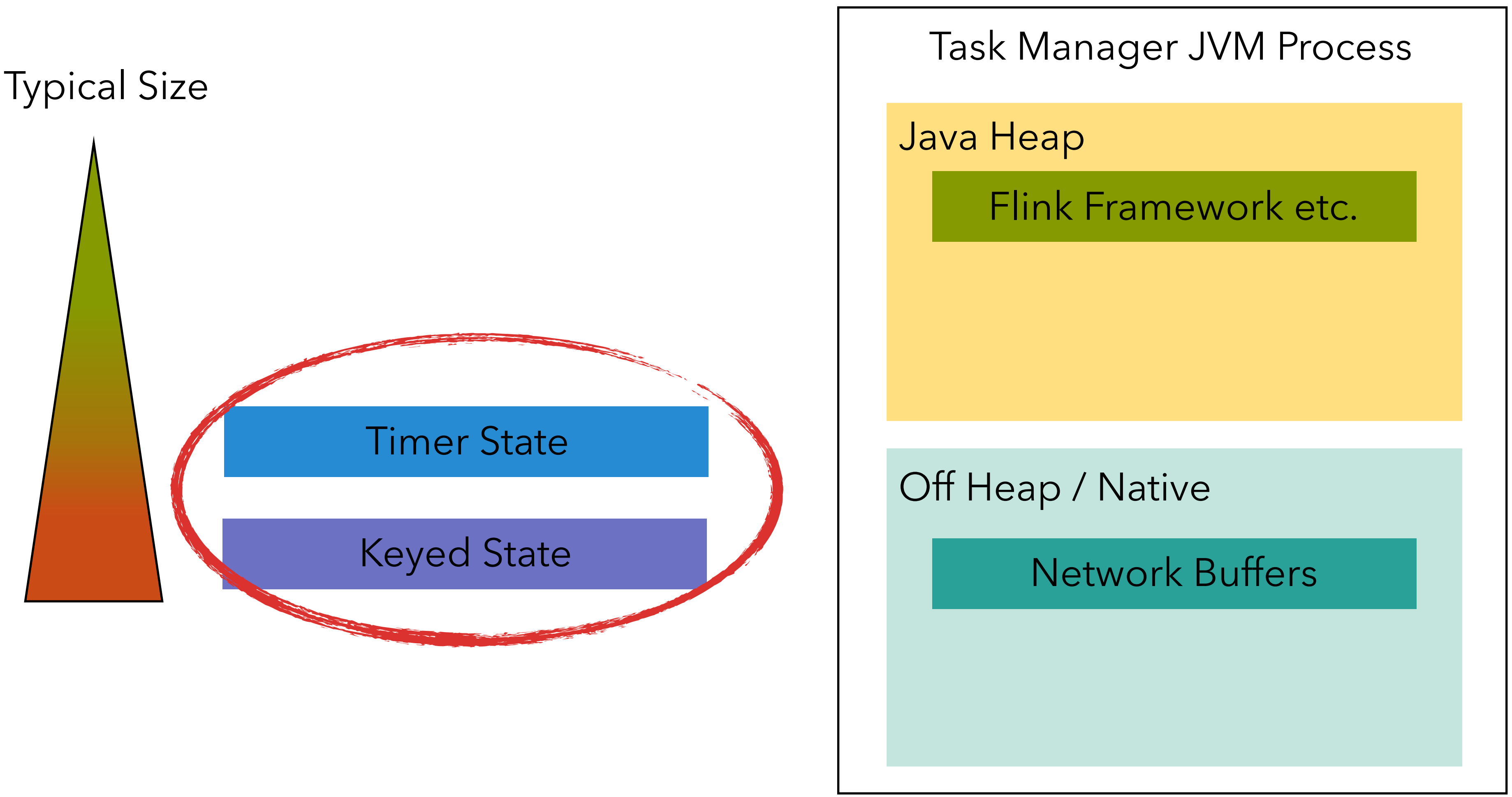
Task Manager Process Memory Layout



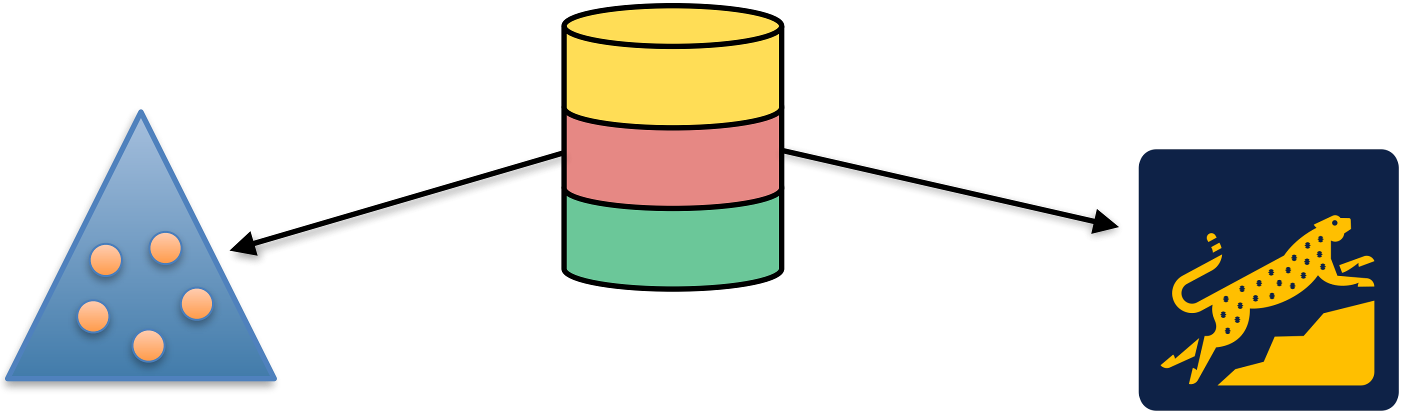
Task Manager Process Memory Layout



Task Manager Process Memory Layout



Keyed State Backends



Based on Java Heap Objects

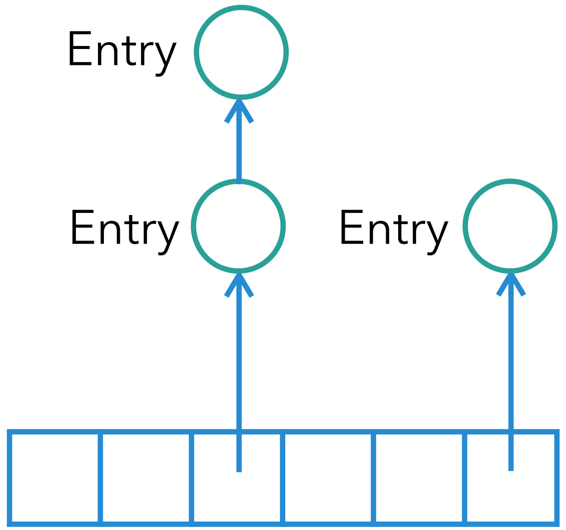
Based on RocksDB

Heap Keyed State Backend

- State lives as Java objects on the heap
- Organized as chained hash table, key ↦ state
- One hash table per registered state
- Supports asynchronous state snapshots
- Data is de / serialized only during state snapshot and restore

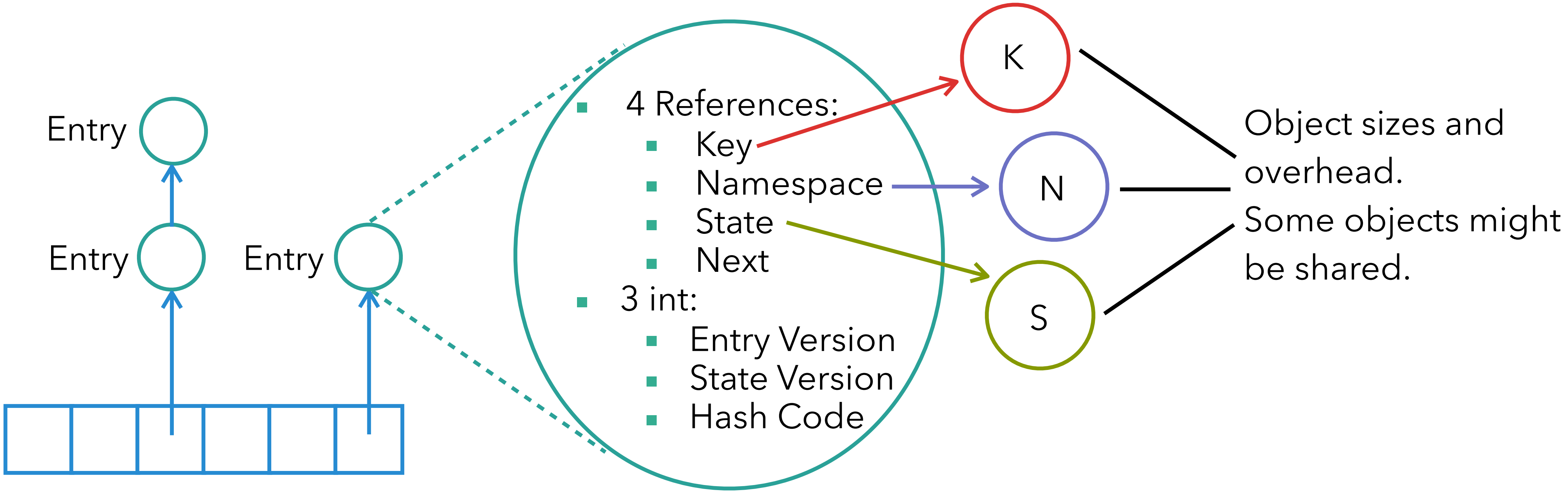
- Highest Performance
- Affected by garbage collection overhead / pauses
- Currently no incremental checkpoints
- High memory overhead of representation
- State is limited by available heap memory

Heap State Table Architecture



- Hash buckets (Object[]), 4B-8B per slot
- Load factor <= 75%
- Incremental rehash

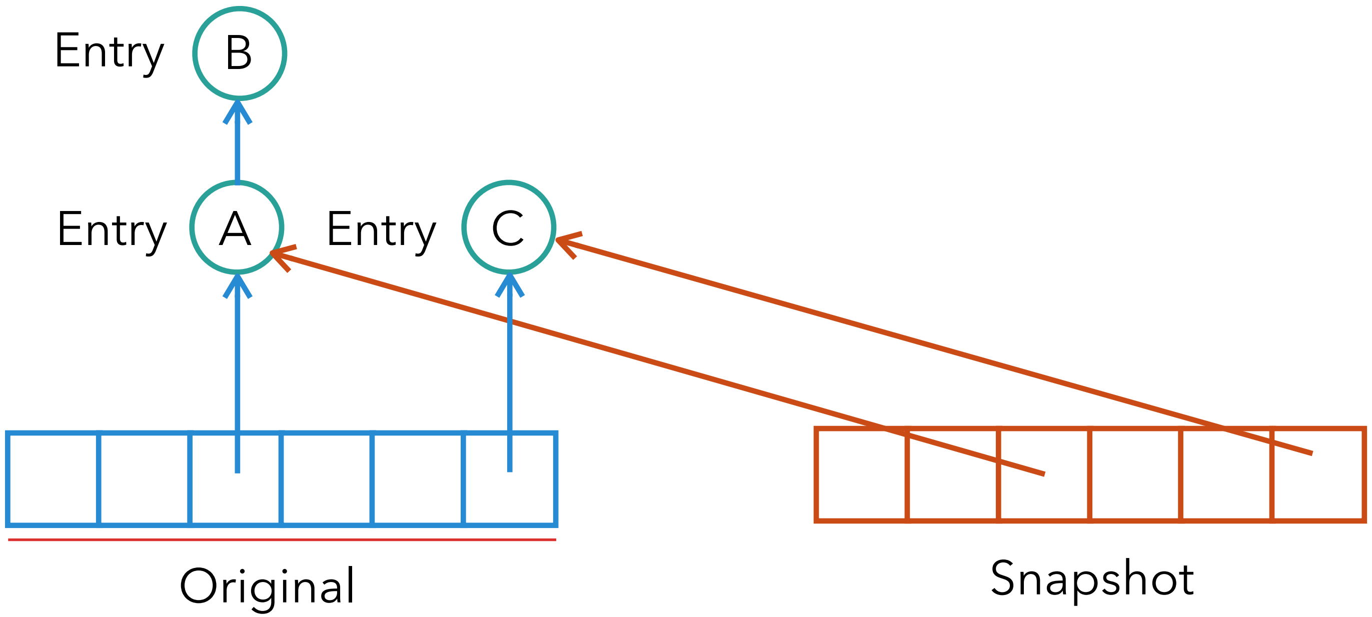
Heap State Table Architecture



- Hash buckets (Object[]), 4B-8B per slot
- Load factor <= 75%
- Incremental rehash

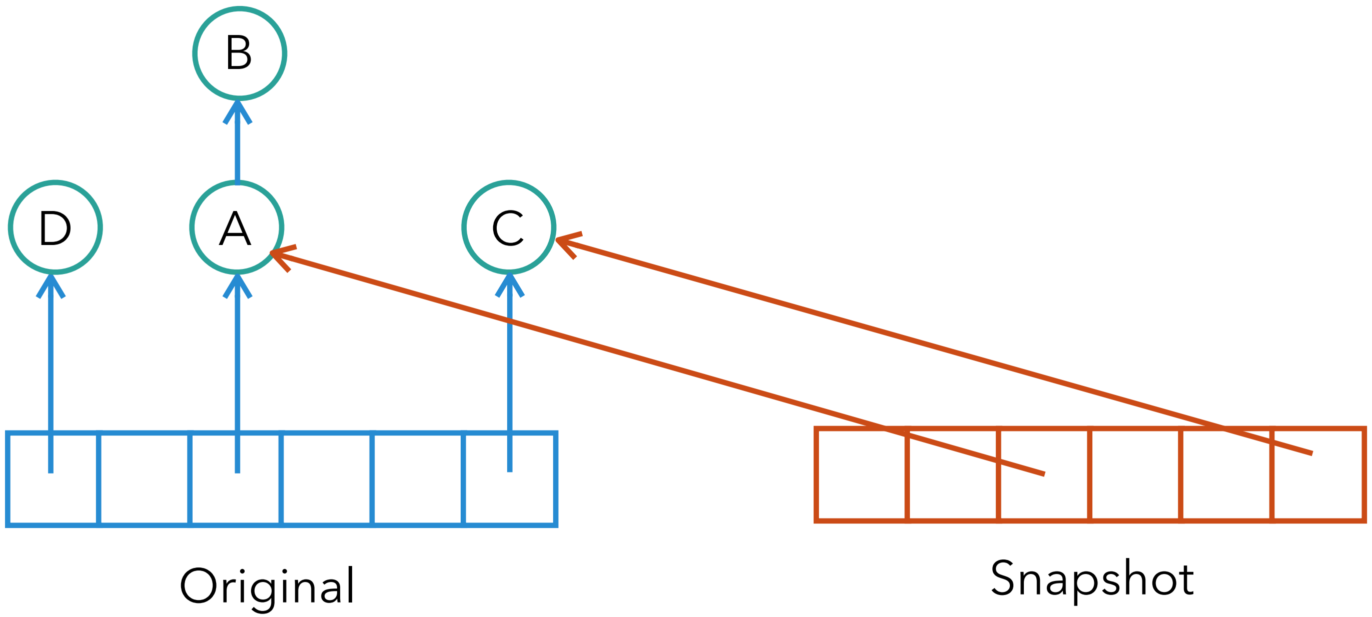
4 x (4B-8B)
+ 3 x 4B
+ ~8B-16B (Object overhead)

Heap State Table Snapshot



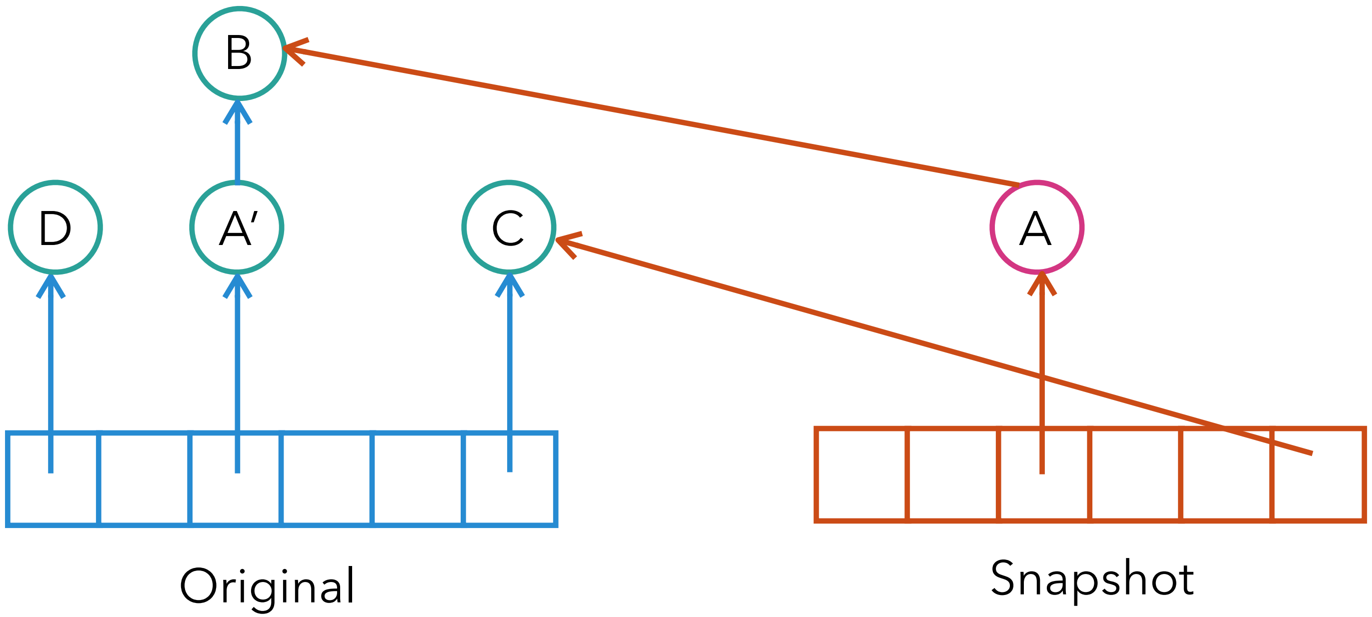
Copy of hash bucket array is snapshot overhead

Heap State Table Snapshot



No conflicting modification = no overhead

Heap State Table Snapshot



Modifications trigger deep copy of entry - only as much as required. This depends on what was modified and what is immutable (as determined by type serializer).
Worst case overhead = size of original at time of snapshot.

Heap Backend Tuning Considerations

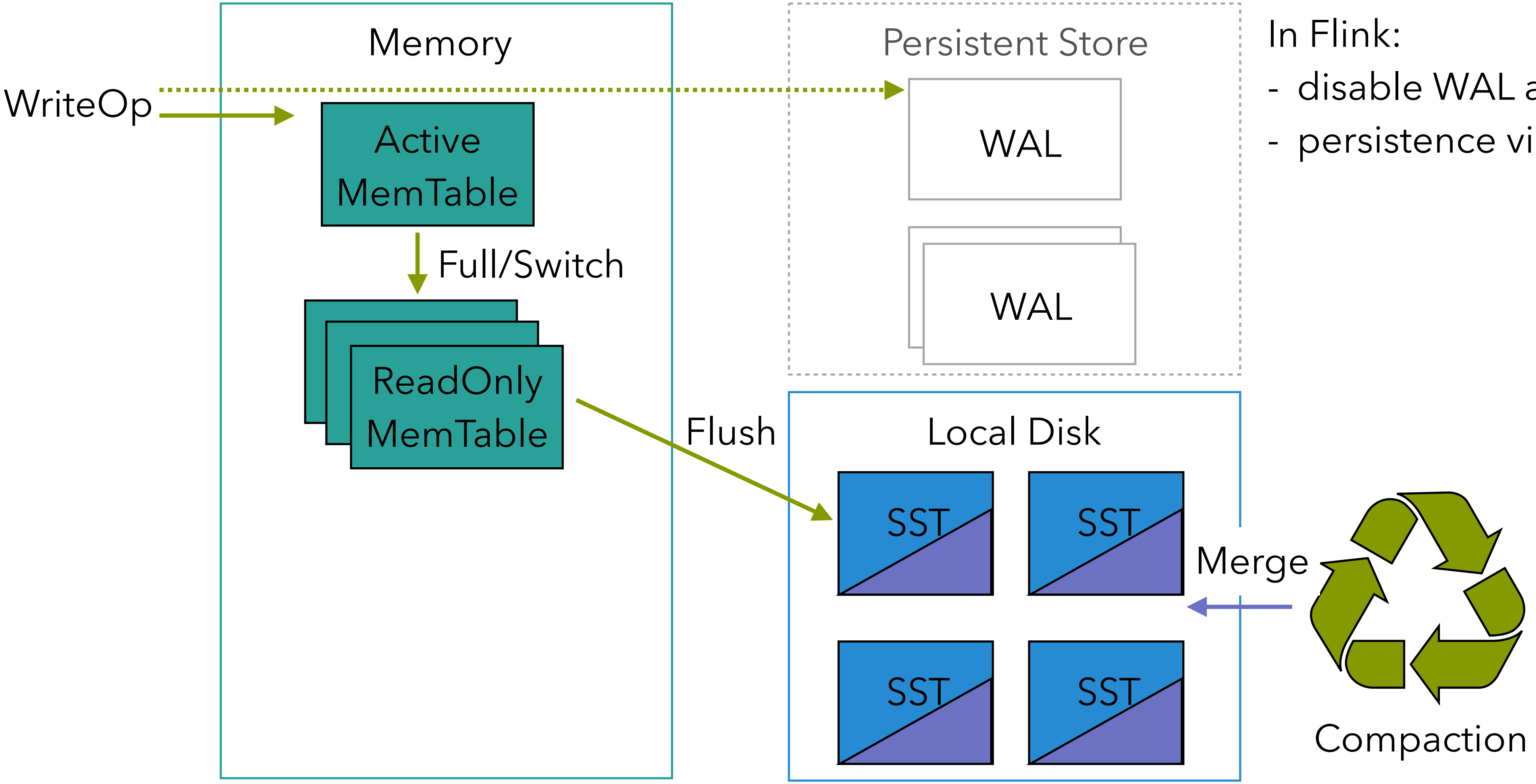
- Choose TypeSerializers with efficient copy-methods
- Flag immutability of objects where possible to avoid copy completely
- Flatten POJOs / avoid deep objects
  - Reduces object overheads and following references
- GC choice / tuning
- Scale out using multiple task managers per node

RocksDB Keyed State Backend Characteristics

- State lives as serialized byte-strings in off-heap memory and on local disk
- One column family per registered state (~table)
- Key / Value store, organized as a log-structured merge tree (LSM tree)
  - Key: serialized bytes of <keygroup, key, namespace>
- LSM naturally supports MVCC
- Data is de / serialized on every read and update

- Not affected by garbage collection
- Relatively low overhead of representation
- LSM naturally supports incremental snapshots
- State size is limited by available local disk space
- Lower performance (~ order of magnitude compared to Heap state backend)

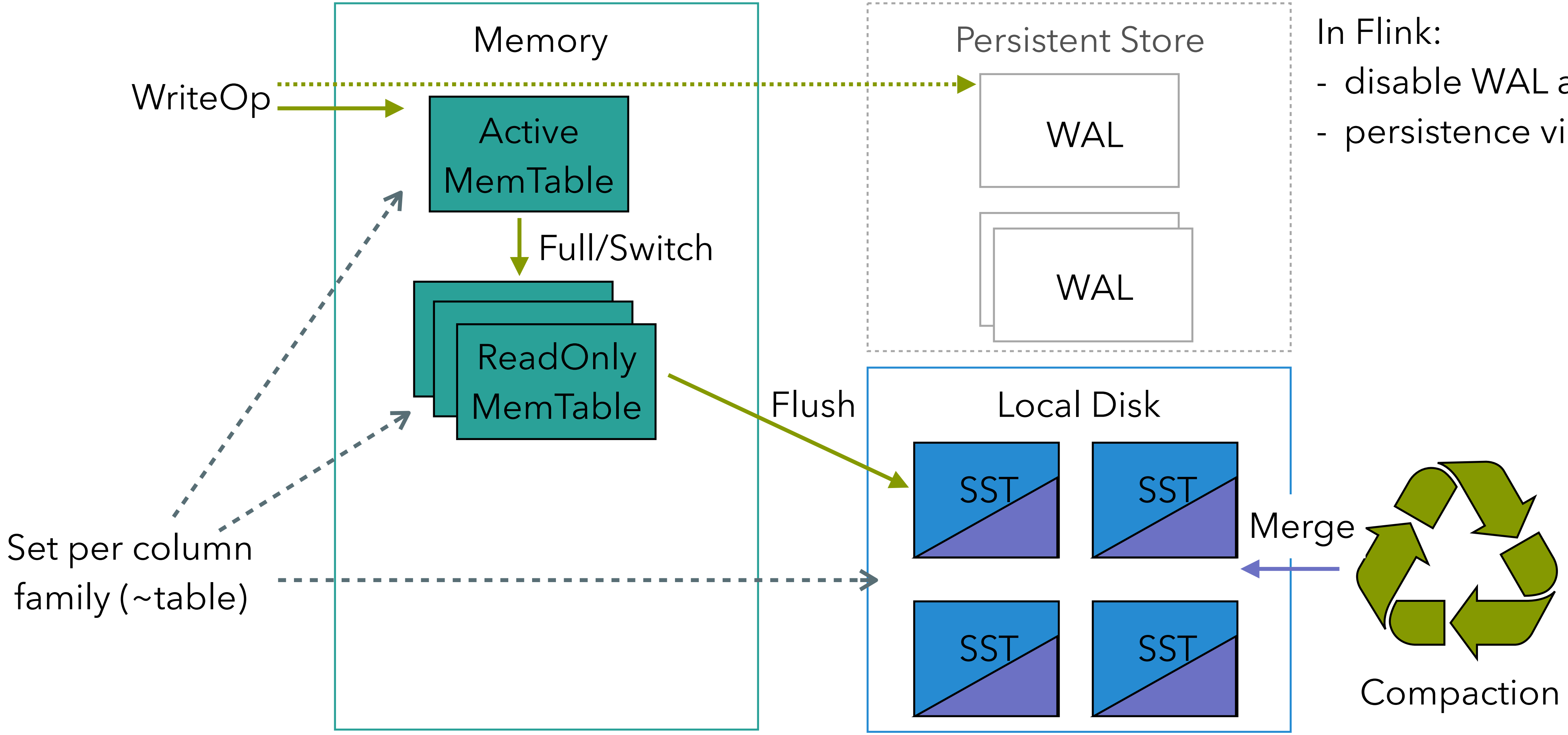
RocksDB Architecture



In Flink:

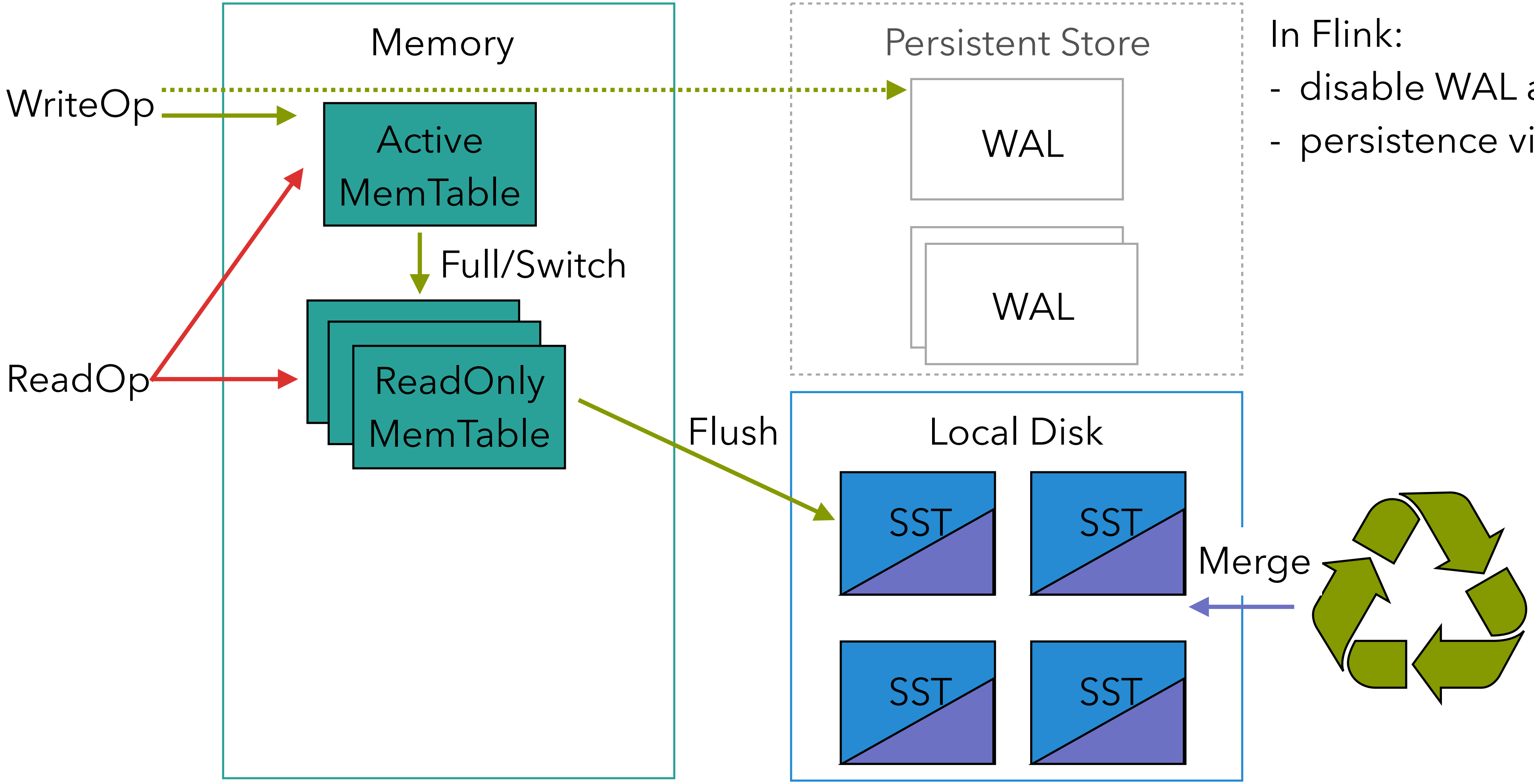
- disable WAL and sync
- persistence via checkpoints

RocksDB Architecture



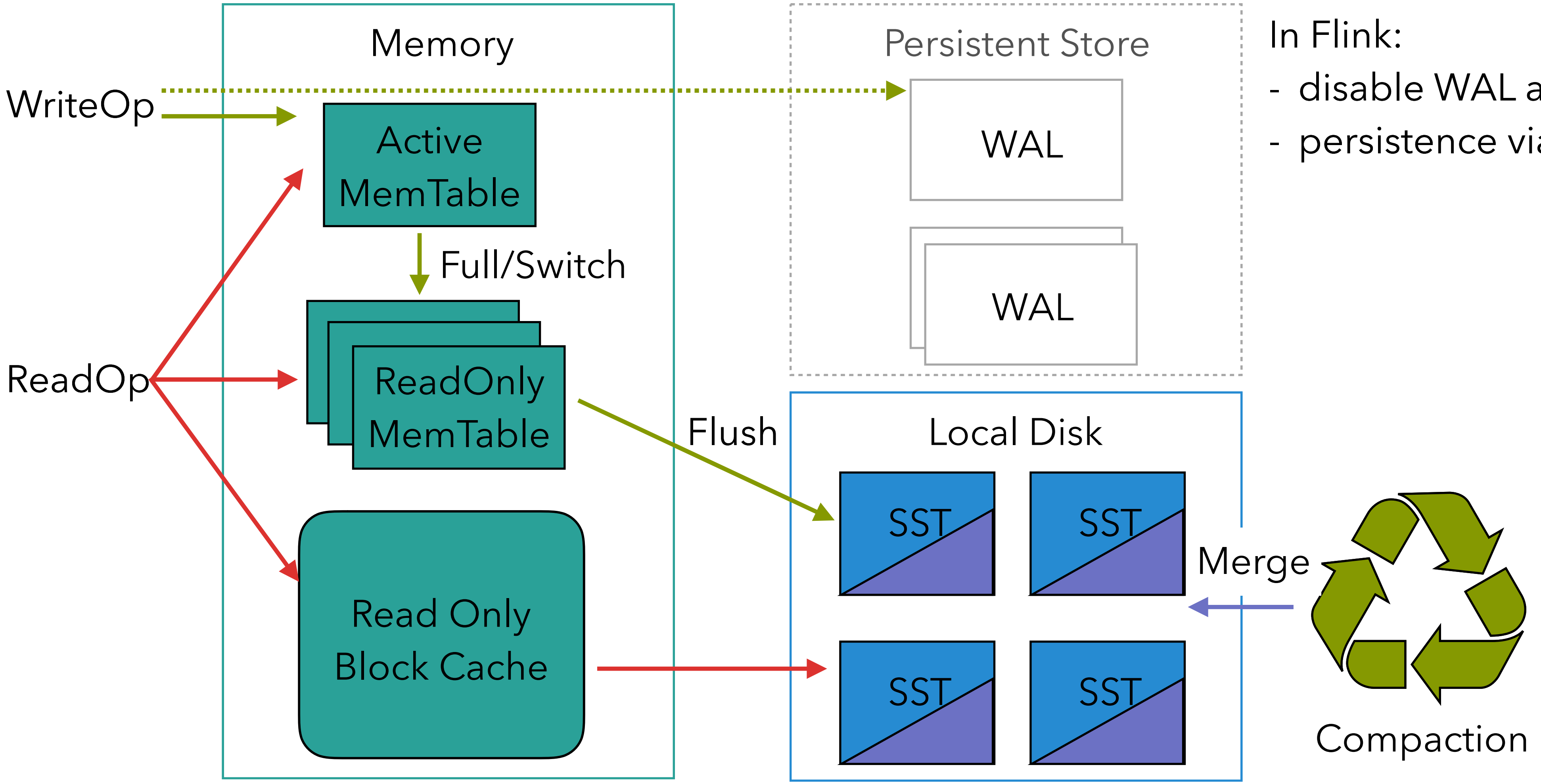
In Flink:
- disable WAL and sync
- persistence via checkpoints

RocksDB Architecture



In Flink:
- disable WAL and sync
- persistence via checkpoints

RocksDB Architecture



In Flink:

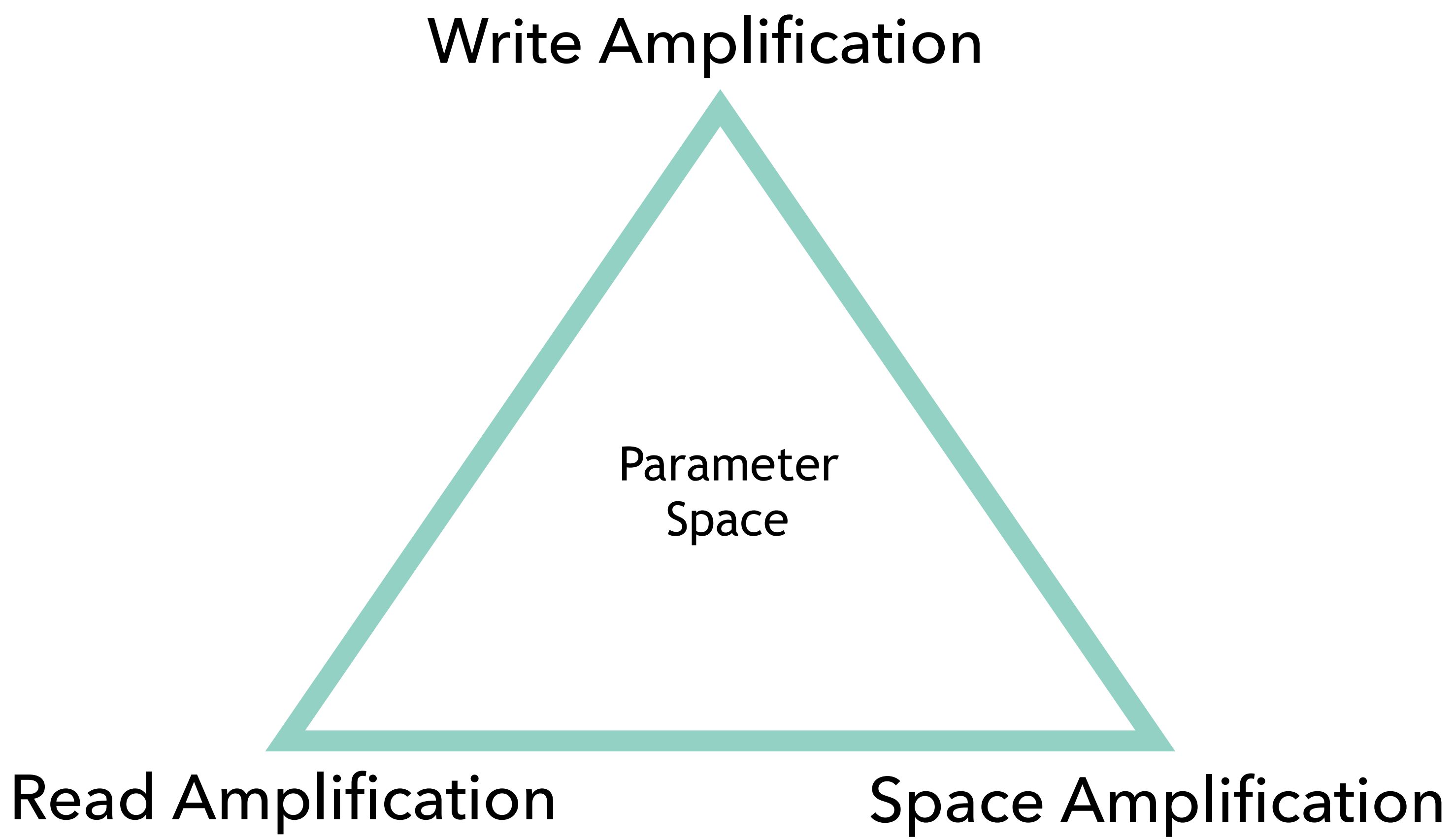
- disable WAL and sync
- persistence via checkpoints

RocksDB Resource Consumption

- One RocksDB instance per operator subtask
- block_cache_size
  - Size of the block cache
- write_buffer_size
  - Max size of a MemTable
- max_write_buffer_number
  - The maximum number of MemTable’s allowed in memory before flush to SST file
- Indexes and bloom filters
  - Optional
- Table Cache
  - Caches open file descriptors to SST files
  - Default: unlimited!

Performance Tuning

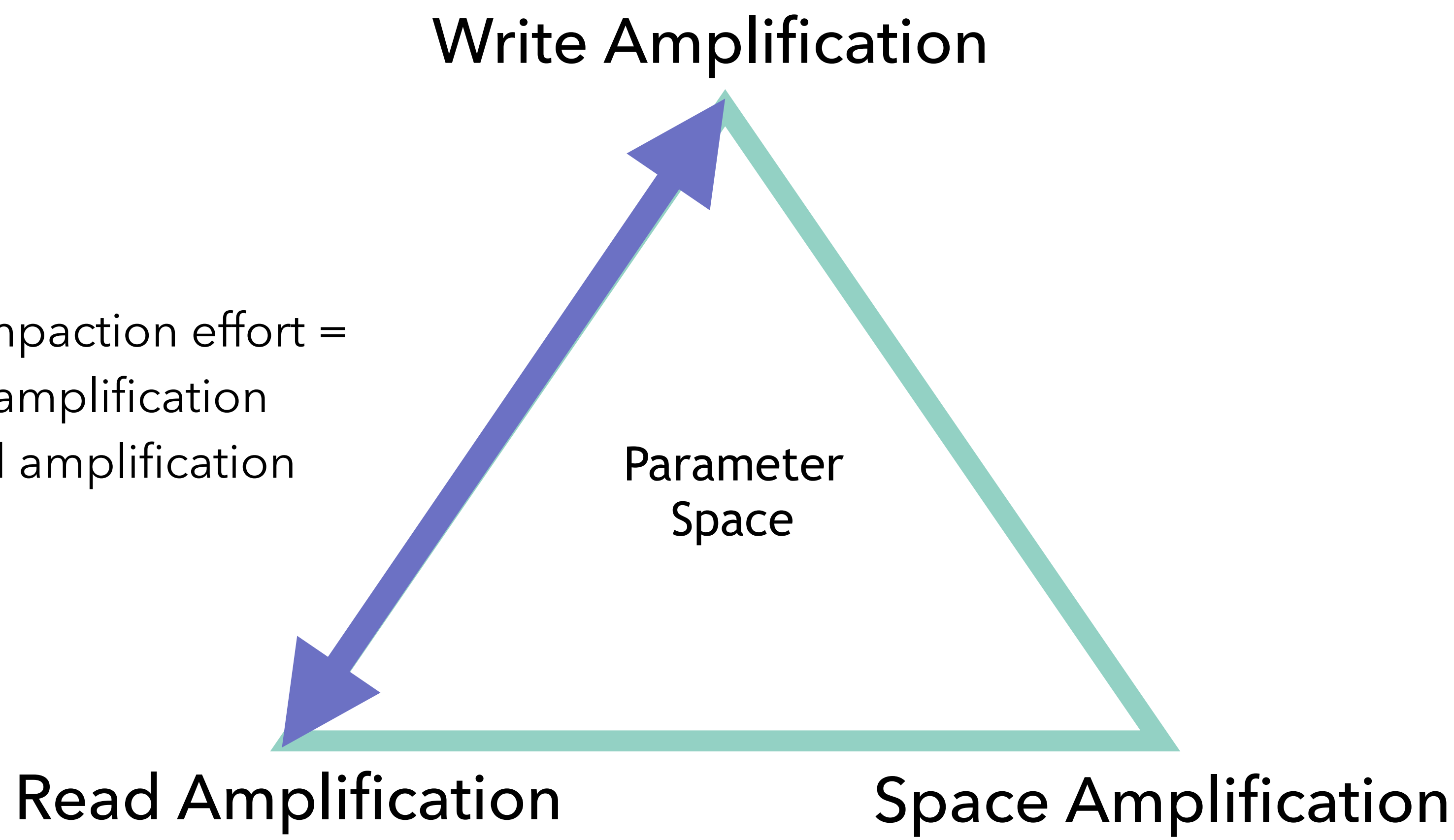
Amplification Factors



More details: https://github.com/facebook/rocksdb/wiki/RocksDB-Tuning-Guide

Performance Tuning

Amplification Factors



Example: More compaction effort = increased write amplification and reduced read amplification

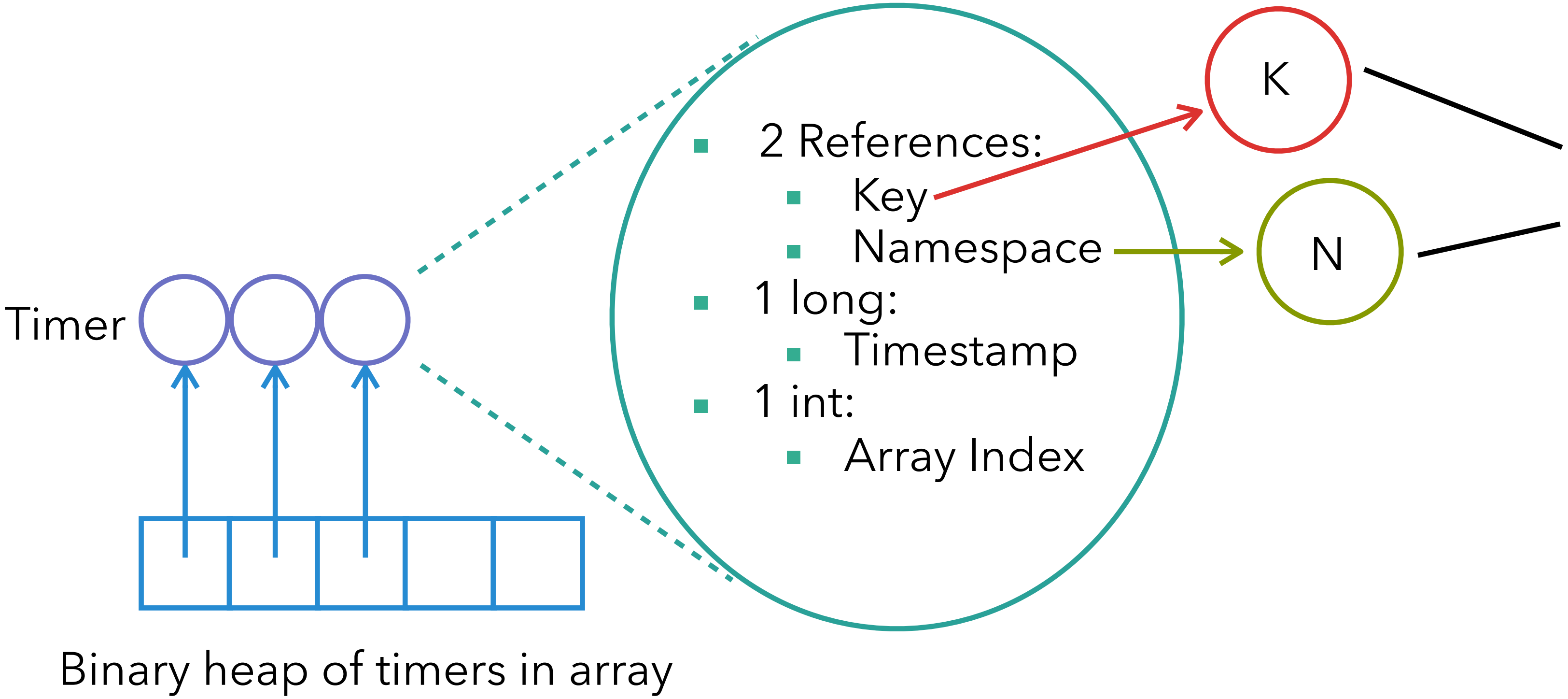
More details: https://github.com/facebook/rocksdb/wiki/RocksDB-Tuning-Guide

General Performance Considerations

- Use efficient TypeSerializer’s and serialization formats
- Decompose user code objects
  - ValueState<List<Integer>> → ListState<Integer>
  - ValueState<Map<Integer, Integer>> → MapState<Integer, Integer>
- Use the correct configuration for your hardware setup
- Consider enabling RocksDB native metrics to profile your applications
- File Systems
  - Working directory on fast storage, ideally local SSD. Could even be memory.
  - EBS performance can be problematic

Timer Service

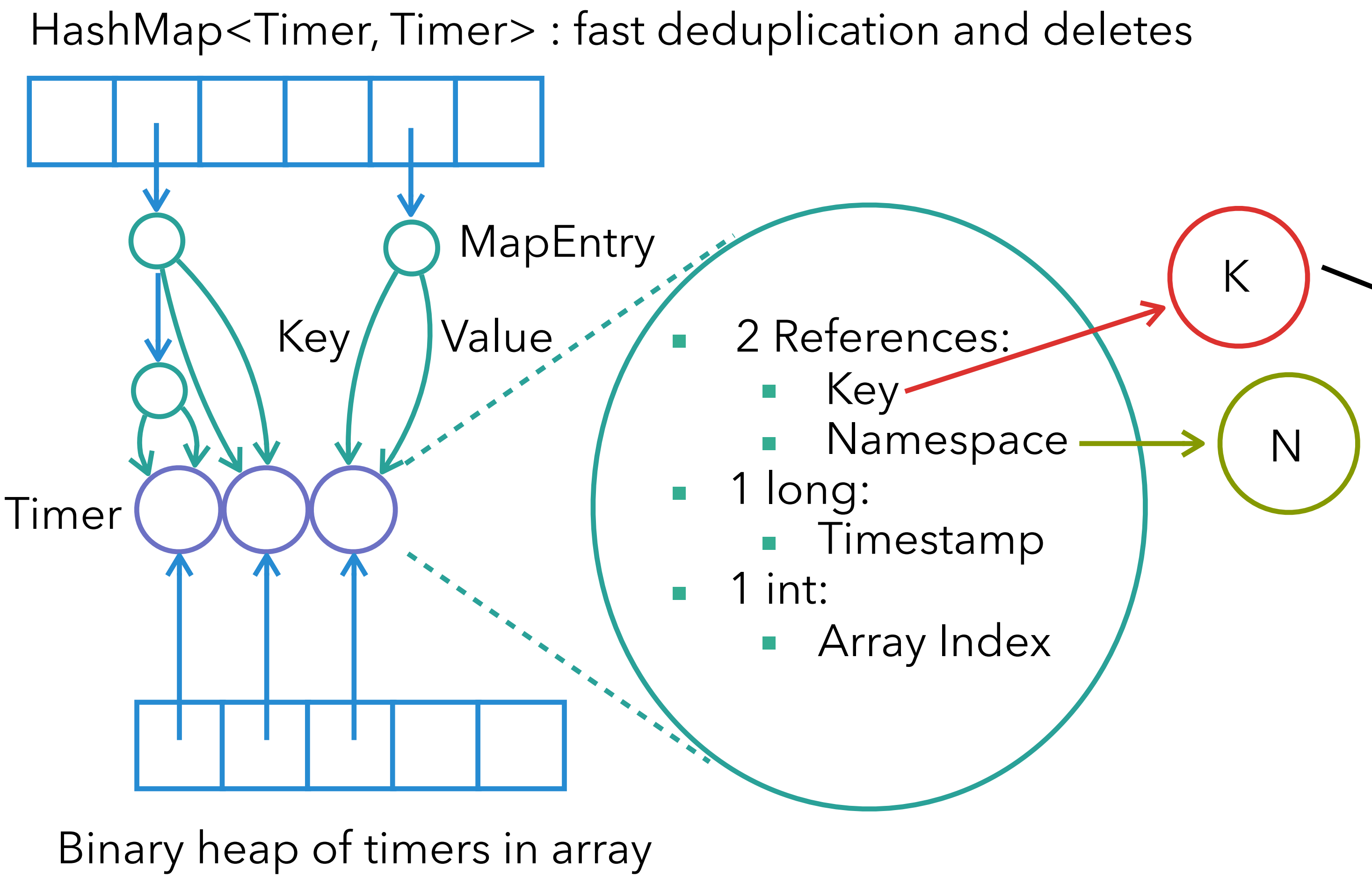
Heap Timers



Object sizes and overhead.
Some objects might be shared.

Peek: O(1)
Poll: O(log(n))
Insert: O(log(n))
Delete: O(n)
Contains O(n)

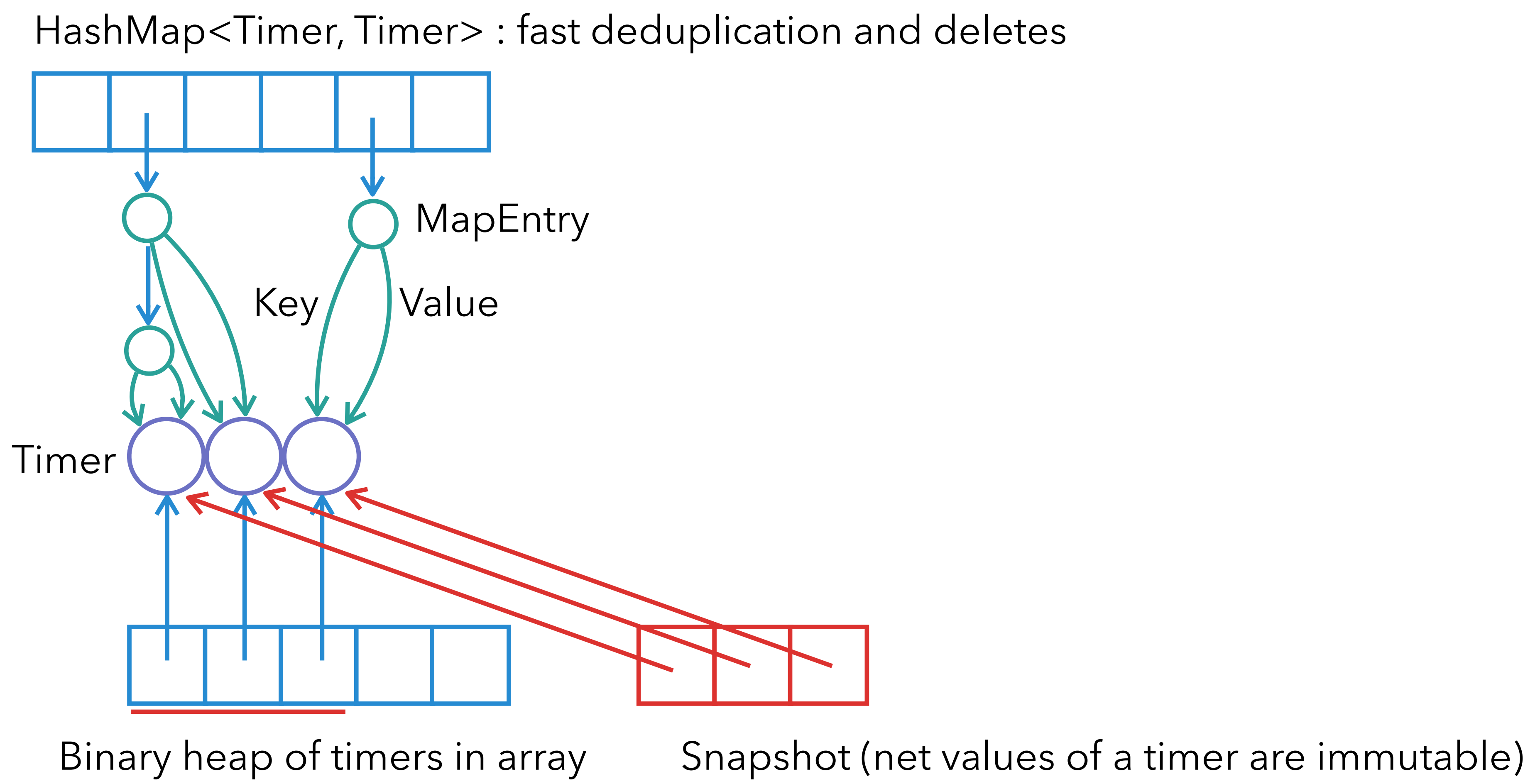
Heap Timers



Object sizes and overhead.
Some objects might be shared.

Peek: O(1)
Poll: O(log(n))
Insert: O(log(n))
Delete: O(log(n))
Contains O(1)

Heap Timers


HashMap<Timer, Timer> : fast deduplication and deletes
MapEntry
Key
Value
Timer
Binary heap of timers in array
Snapshot (net values of a timer are immutable)

RocksDB Timers

Column Family - only key, no value

Key Group	Time stamp	Key	Name space
0	20	A	X
0	40	D	Z

…

Key Group	Time stamp	Key	Name space
1	10	D	Z
1	20	C	Y

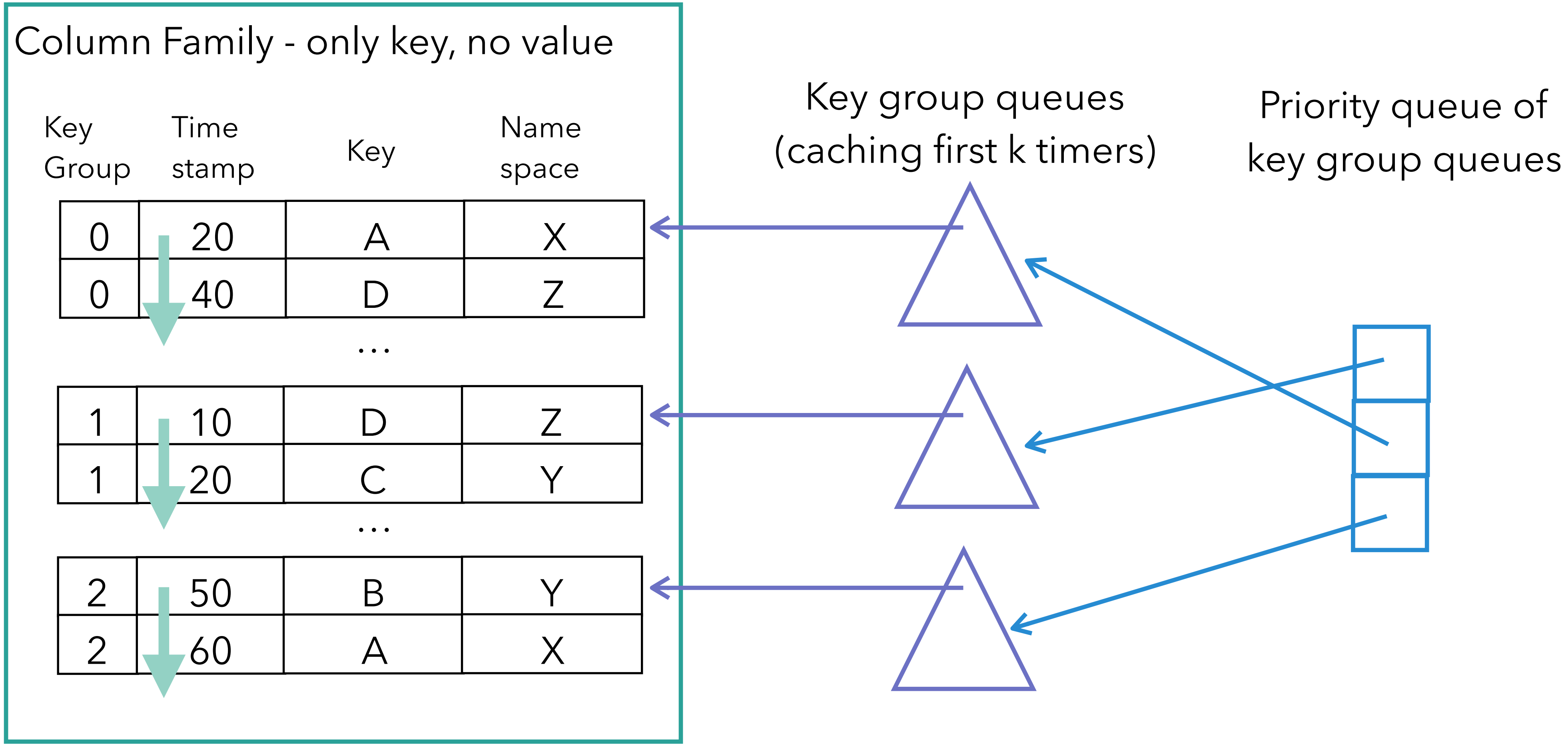
…

Key Group	Time stamp	Key	Name space
2	50	B	Y
2	60	A	X

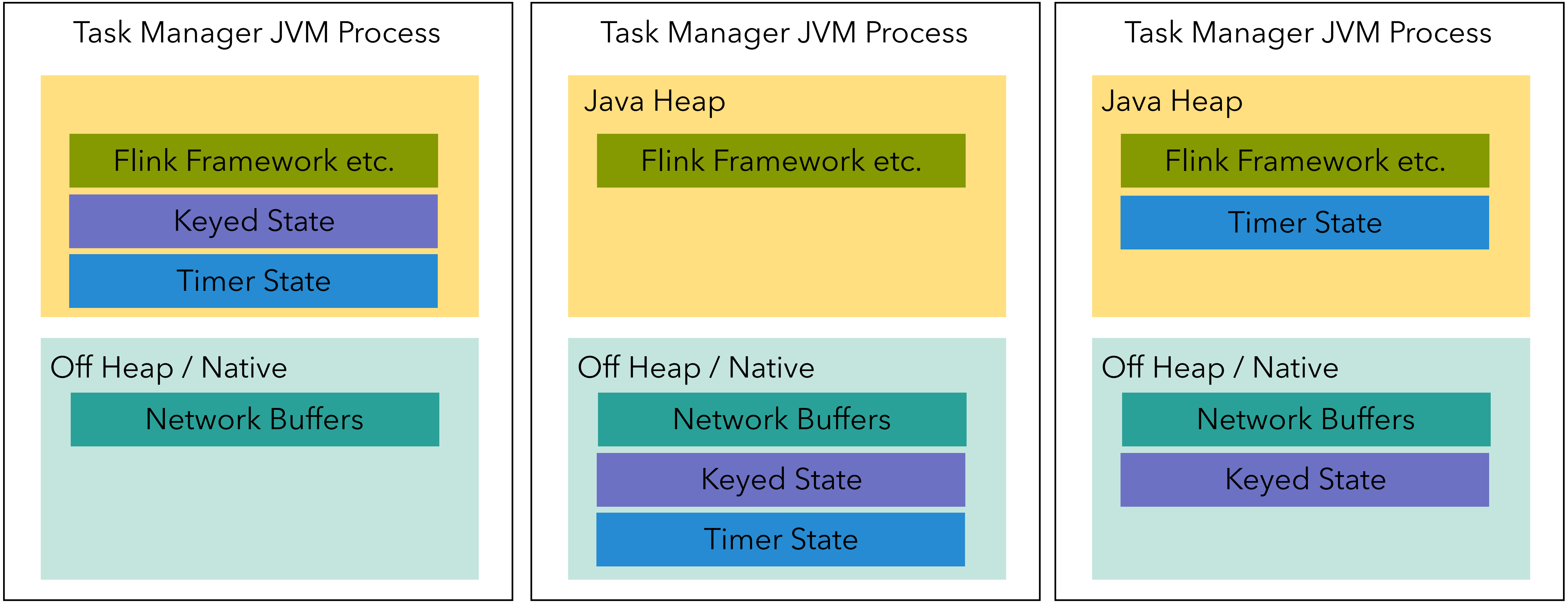
…

Lexicographically ordered
byte sequences as key, no value

RocksDB Timers

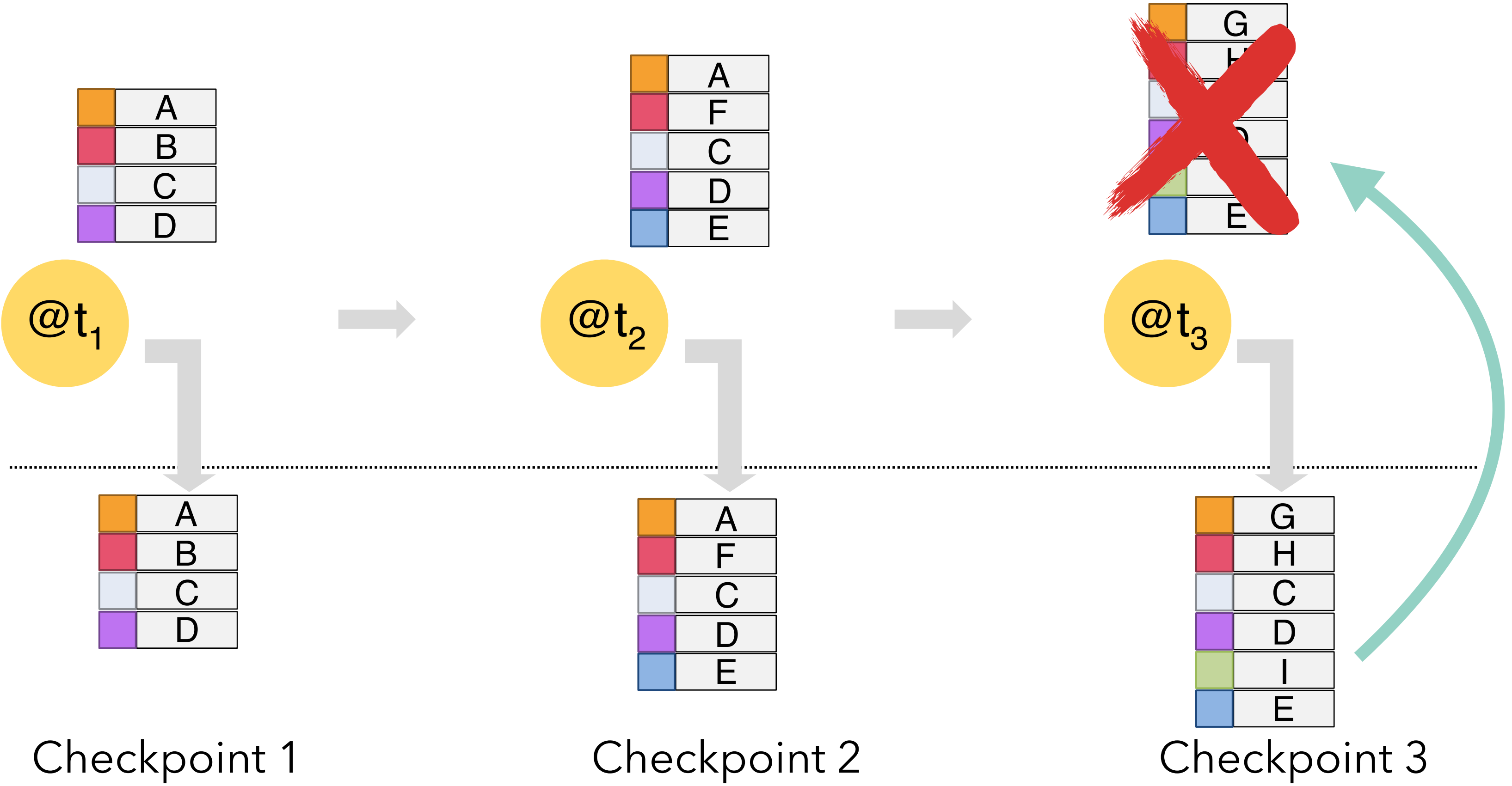


3 Task Manager Memory Layout



Full / Incremental Checkpoints

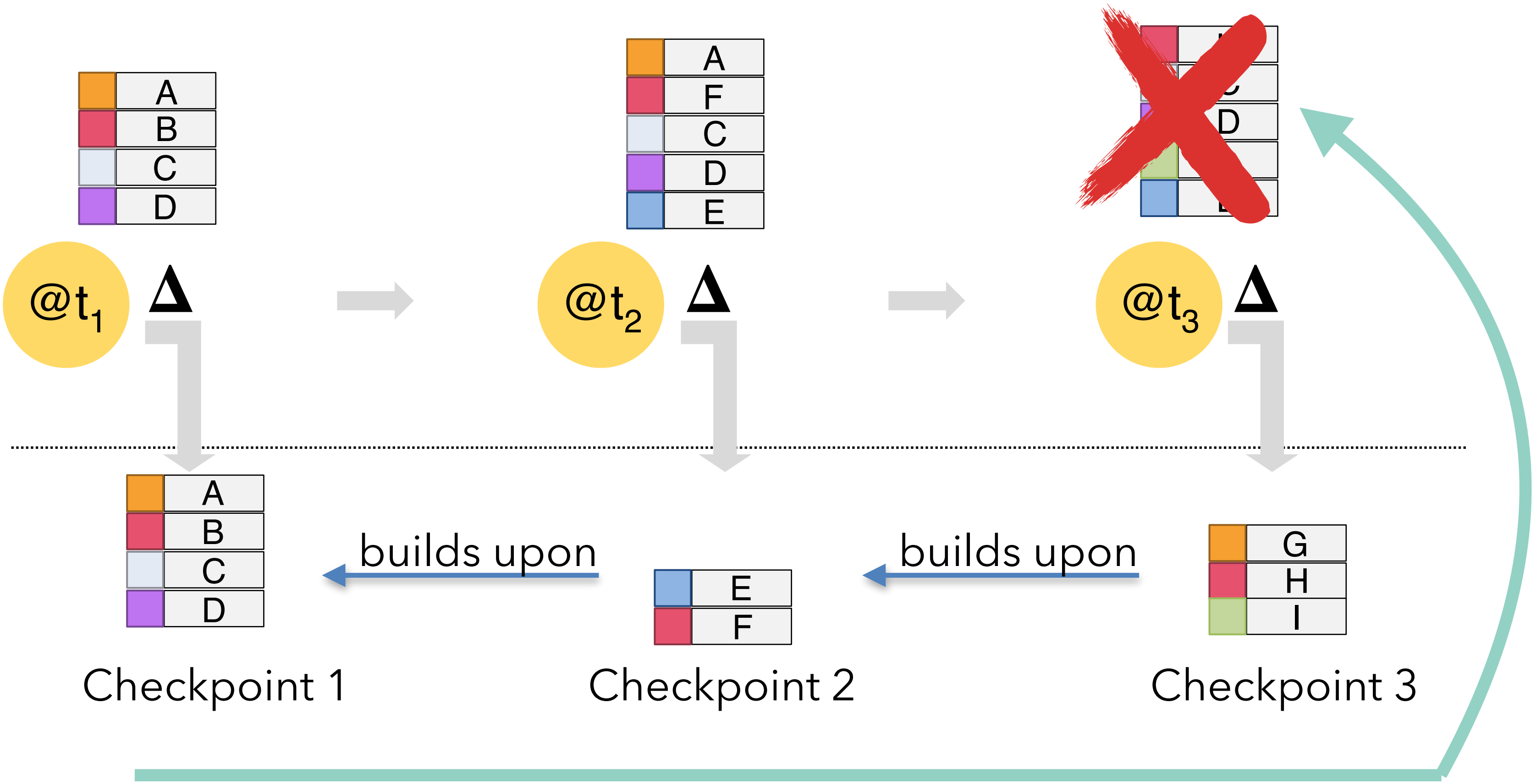
Full Checkpoint



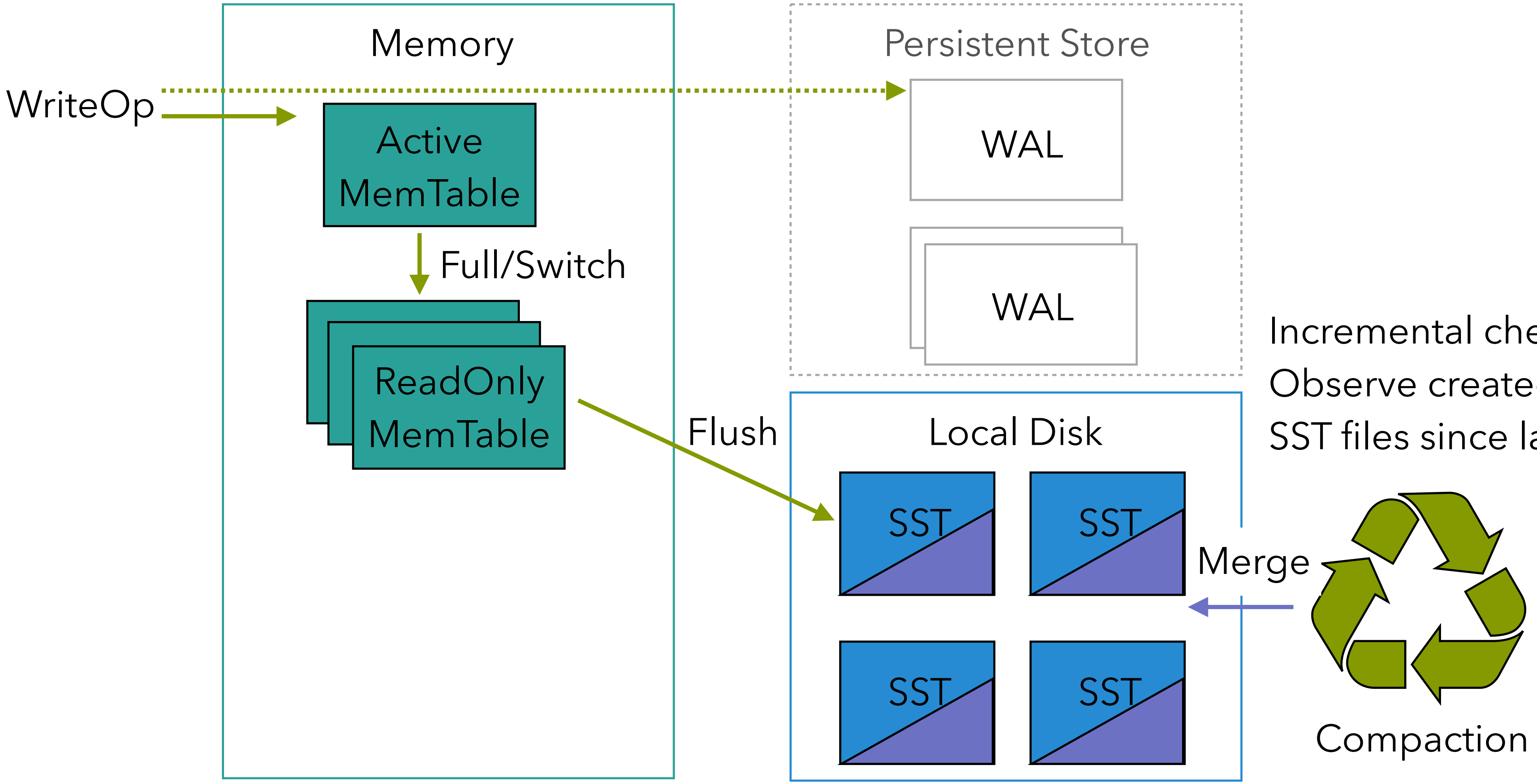
Full Checkpoint Overview

- Creation iterates and writes full database snapshots as a stream to stable storage
- Restore reads data as a stream from stable storage and re-inserts into the state backend
- Each checkpoint is self contained, and size is proportional to the size of full state
- Optional: compression with snappy

Incremental Checkpoint



Incremental Checkpoints with RocksDB



Incremental checkpoint:
Observe created/deleted
SST files since last checkpoint

Incremental Checkpoint Overview

- Expected trade-off: faster* checkpoints, slower recovery
- Creation only copies deltas (new local SST files) to stable storage
- Creates write amplification because we also upload compacted SST files so that we can prune checkpoint history
- Sum of all increments that we read from stable storage can be larger than the full state size
- No rebuild is required because we simply re-open the RocksDB backend from the SST files
- SST files are snappy compressed by default

Questions?