Cassandra Fundamentals

Module 1

Architecture

LXFT
LISTED
NYSE

Module plan

- C* Layered Architecture
- Tunable Consistency
- Write Path
- Read Path
- Hinted Handoff
- Read Repair
- Anti-entropy Node Repair

Distributed setting problems to deal with

- Node is down
- Network partition
- Dropped mutations
- Process crash before flush
- File corruption

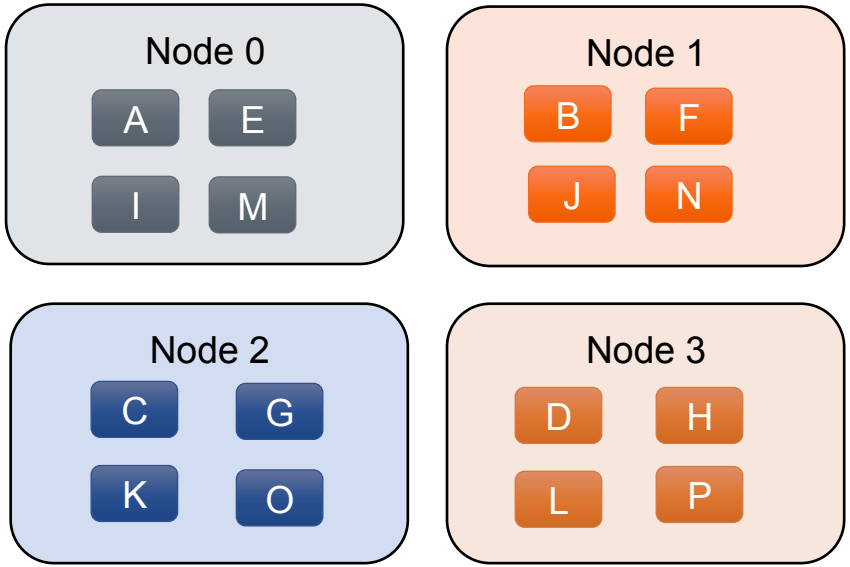
C* Architecture in a flash

- DHT ring
- Every node is in the same role => no SPOF
- O(1) node lookup
- Explicit replication
- Eventually consistent (tunable tradeoff with latency)

C* Architecture Layers

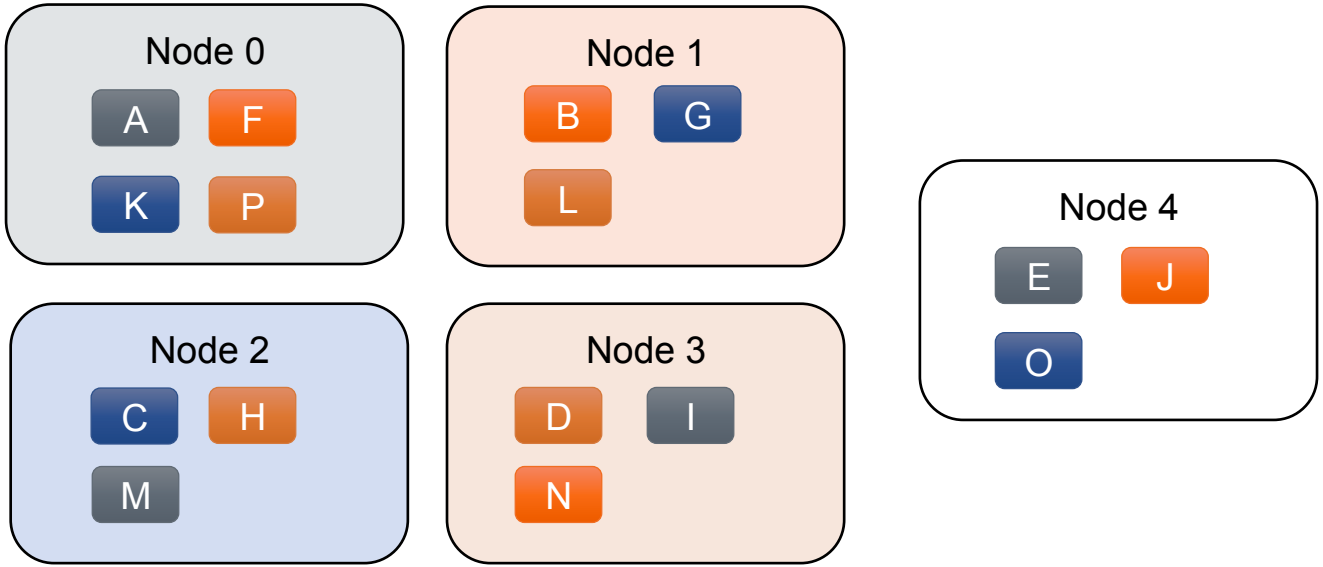
- Middle Layer
  - MemTable
  - SSTable
  - CommitLog
  - BloomFilter
  - Indexes
  - Compaction
- Core Layer
  - Messaging Service
  - Gossip
  - Failure Detection
  - Cluster State
  - Partitioner
  - Replication
- Top Layer
  - Hinted handoff
  - Read repair
  - Anti-entropy node repair
  - Tombstones
  - Bootstrap
  - Monitoring
  - Admin tools

Distributed Hashing



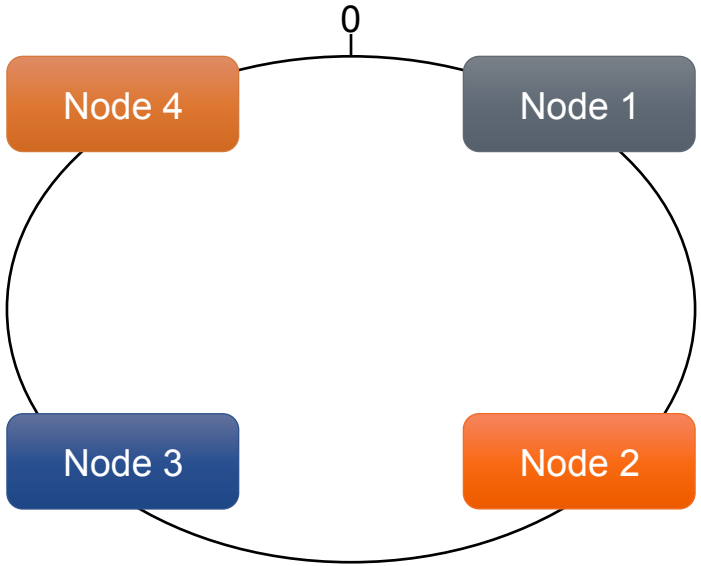
Location = Hash(Key) % # Nodes

Distributed Hashing

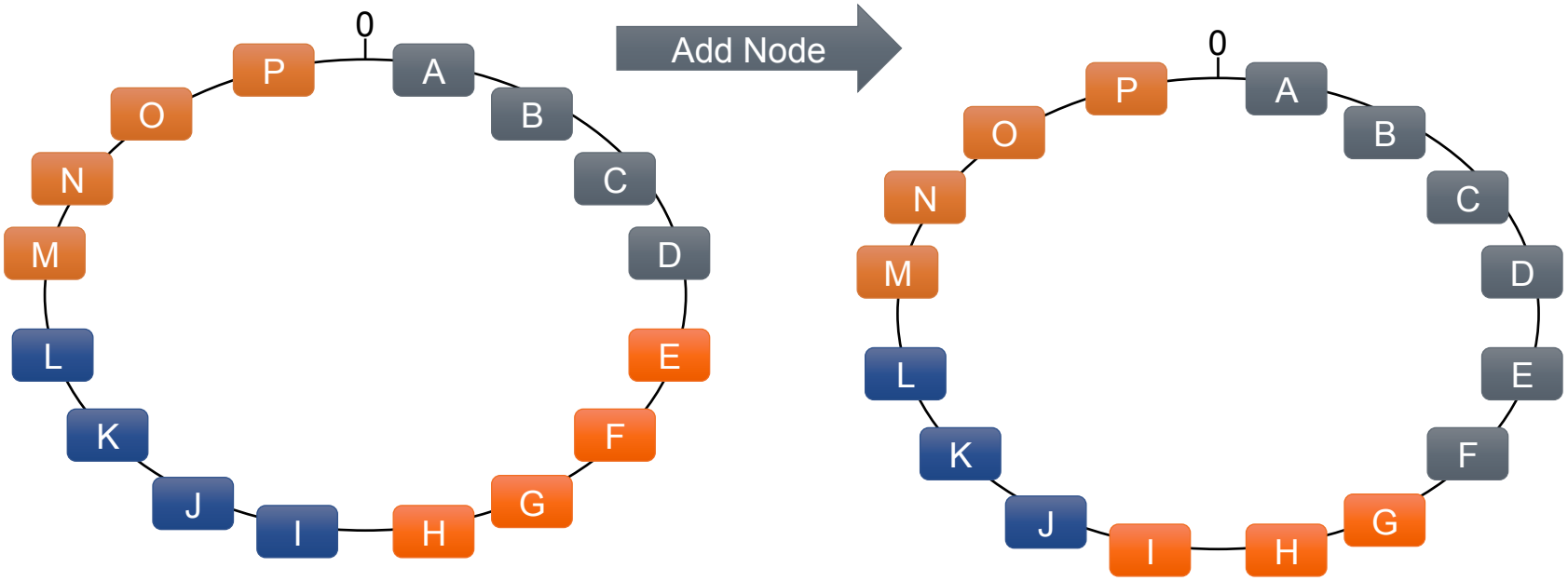


% Data Moved = 100 * N / (N + 1)

Consistent Hashing



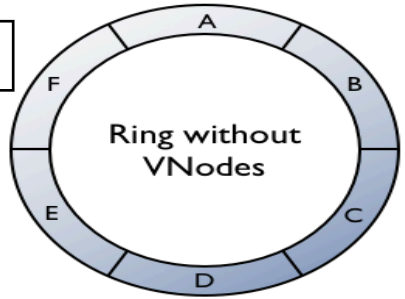
Consistent Hashing

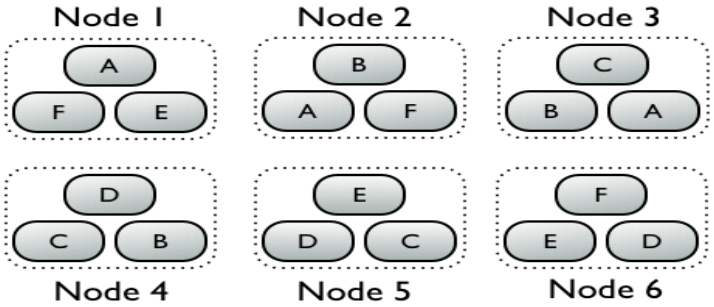


% Data Moved = 100 * 1 / N

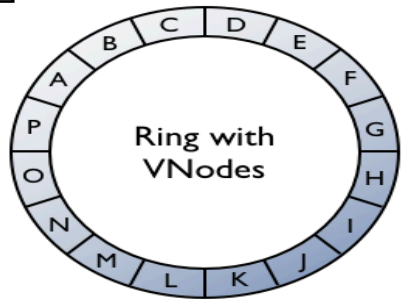
Virtual Nodes

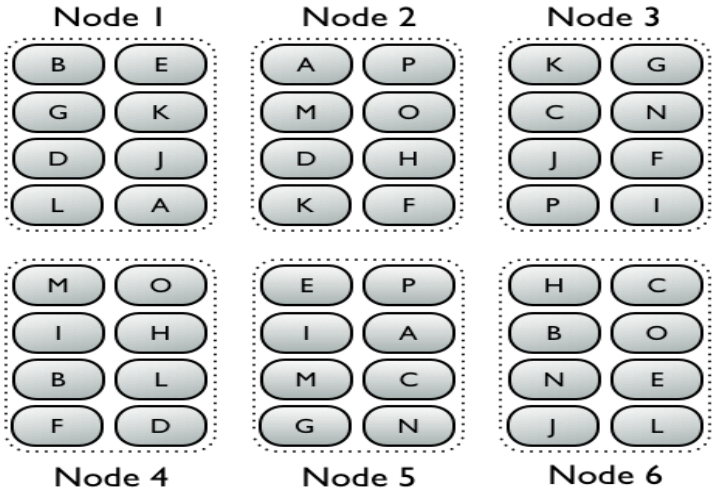
initial_token





num_tokens

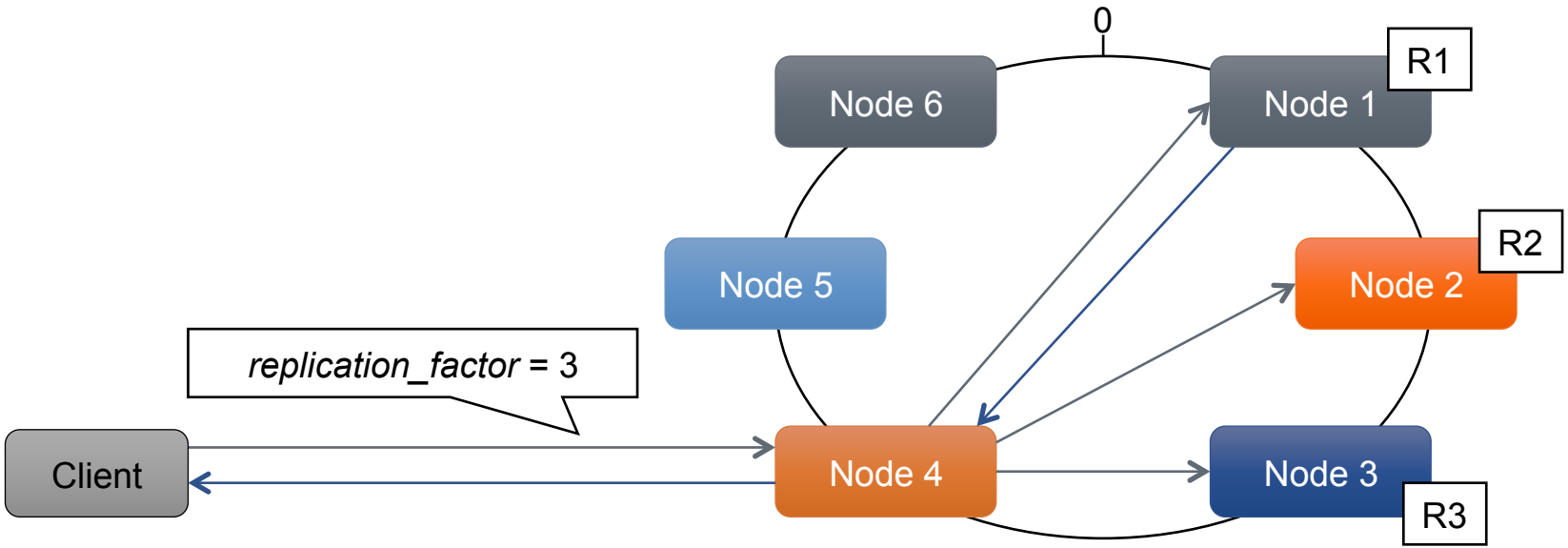




C* Write Basics

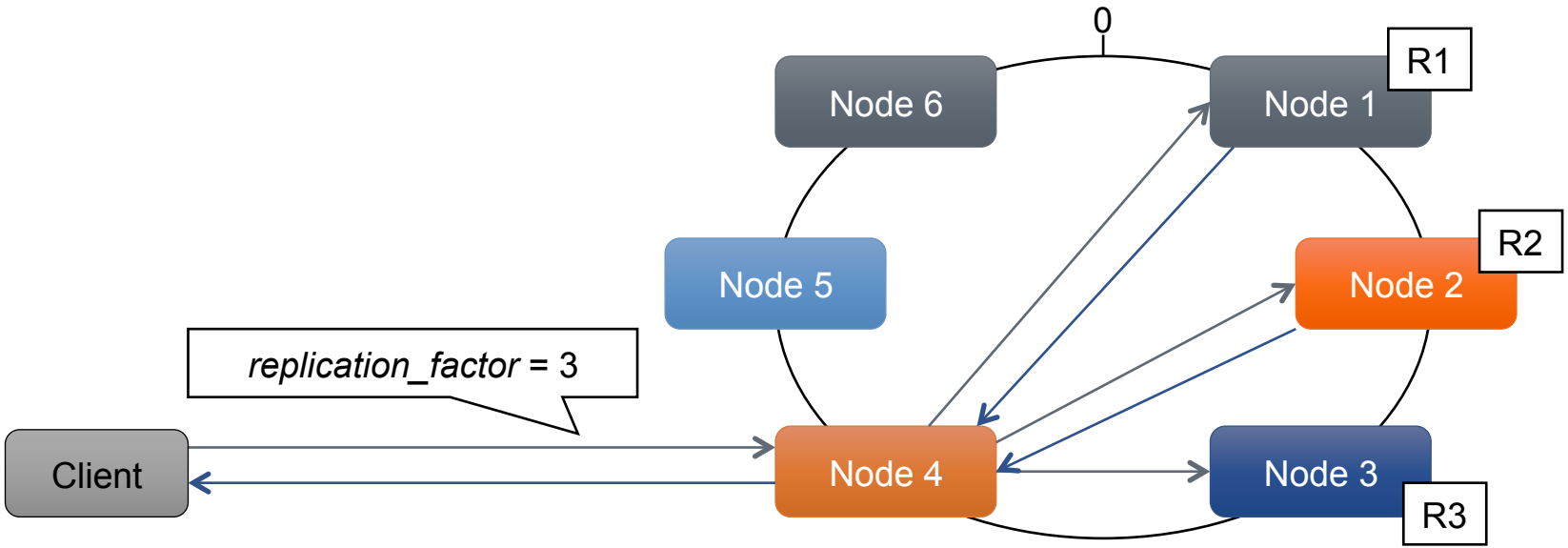
- Determine (all) replica nodes in all DCs
- Send to all replicas in local DC
- Send to one replica in each remote DC
  - It will propagate it to peers
- All respond back to coordinator

Tunable Consistency



INSERT INTO table (column1, …) VALUES (value1, …) USING CONSISTENCY ONE

Tunable Consistency



INSERT INTO table (column1, …) VALUES (value1, …) USING CONSISTENCY QUORUM

Tunable consistency at write

- Coordinator waits for specific count of replicas responses (blocking)
- Consistency Levels:
  - ANY
  - ONE
  - TWO
  - THREE
  - LOCAL_QUORUM
  - EACH_QUORUM
  - ALL

Write modes

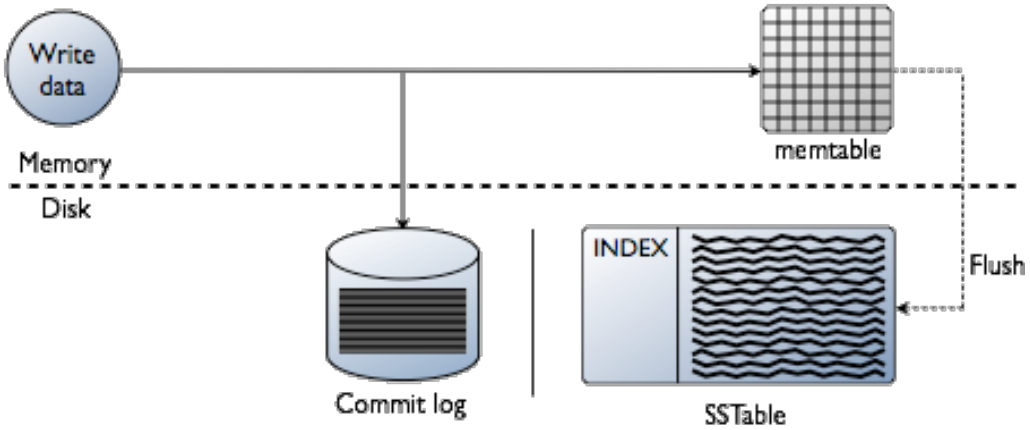
- Quorum write
  - Waits until quorum is reached (blocking).
- Async write
  - Sends write request to any node. That node will push data to appropriate nodes, but return to client immediately.
  - If destination node is down, write data+hint to another node (hinted handoff). Every 10 mins Harvester will find hints and resolve them.
  - Coordinator stores incoming mutation (atomic batch) on two peers in same DC
  - On successful completion coordinator delete that batches
  - In case of dead coordinator, peers will replay batches (not deleted) – every 60 seconds

C* Write

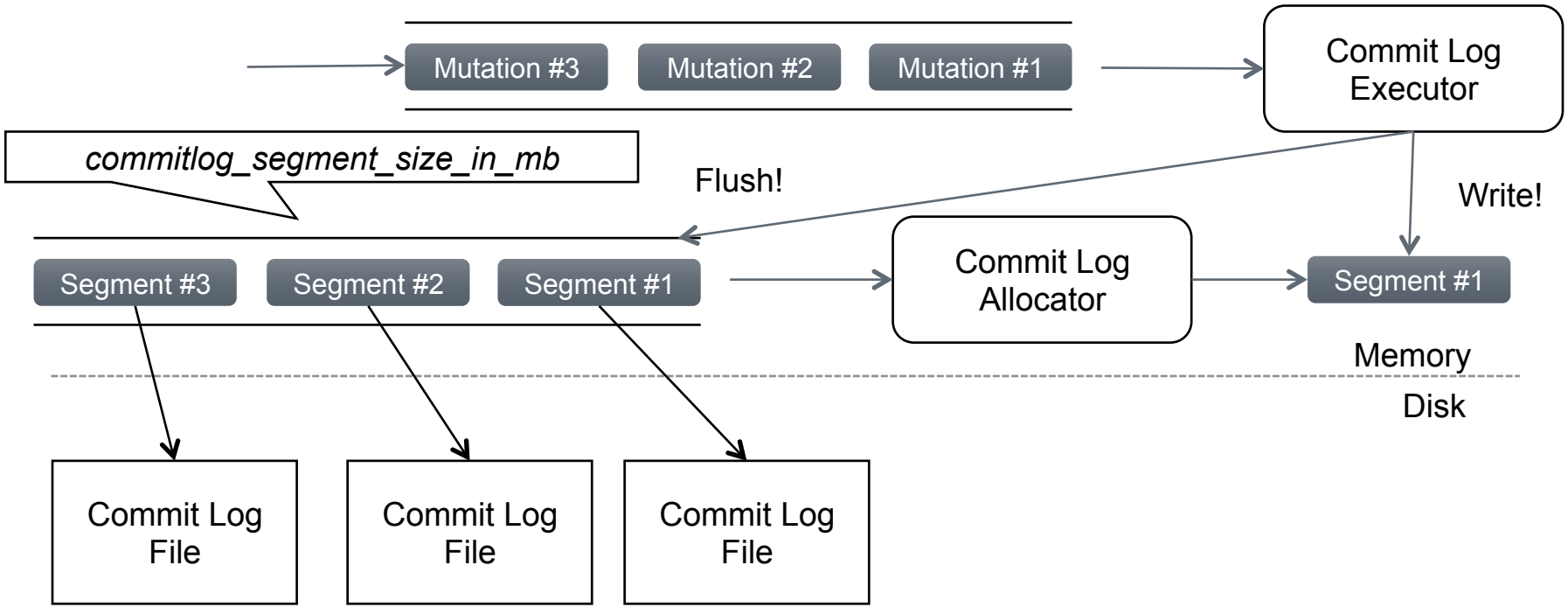
- No reads, no seeks, atomic within Column Family, always writable
- Write to disk commit log (sequential)
- After log is written, send to the appropriate nodes
- Periodic compactions (merging SSTables, recreating Indexes, combining columns, tombstones discarding)

C* Write Path at Each Node

- Write to local Commit Log
- Update appropriate MemTables (one for each Column Family)
- MemTables are flushed when (any of)
  - out of space
  - too many keys (def. 128)
  - time is passed (client provided)
- MemTables are flushed to
  - SSTable (Sorted Strings)
  - SSTable Index = (key, offset) pairs
  - Bloom filter (all keys in SSTable)
- When data is in SSTables, commit log to be deleted



Commit Log

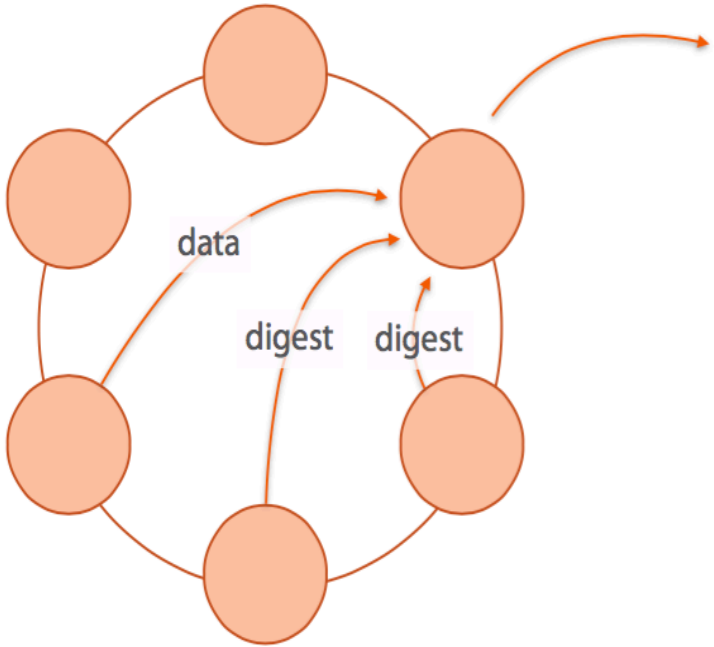

Mutation #3
Mutation #2
Mutation #1
Commit Log Executor
commitlog_segment_size_in_mb
Flush!
Write!
Segment #3
Segment #2
Segment #1
Commit Log Allocator
Segment #1
Memory
Disk
Commit Log File
Commit Log File
Commit Log File

Commit Log

- *commitlog_sync*
  1. *periodic* (default)
     - *commitlog_sync_period_in_ms* (default: 10 seconds)
  2. *batch*
     - *commitlog_batch_window_in_ms*

C* Read Basics

- Read from any node
- Partitioner
- Wait for R responses (sync)
- Wait for N-R responses (in background), then do read repair
- Slower than reads



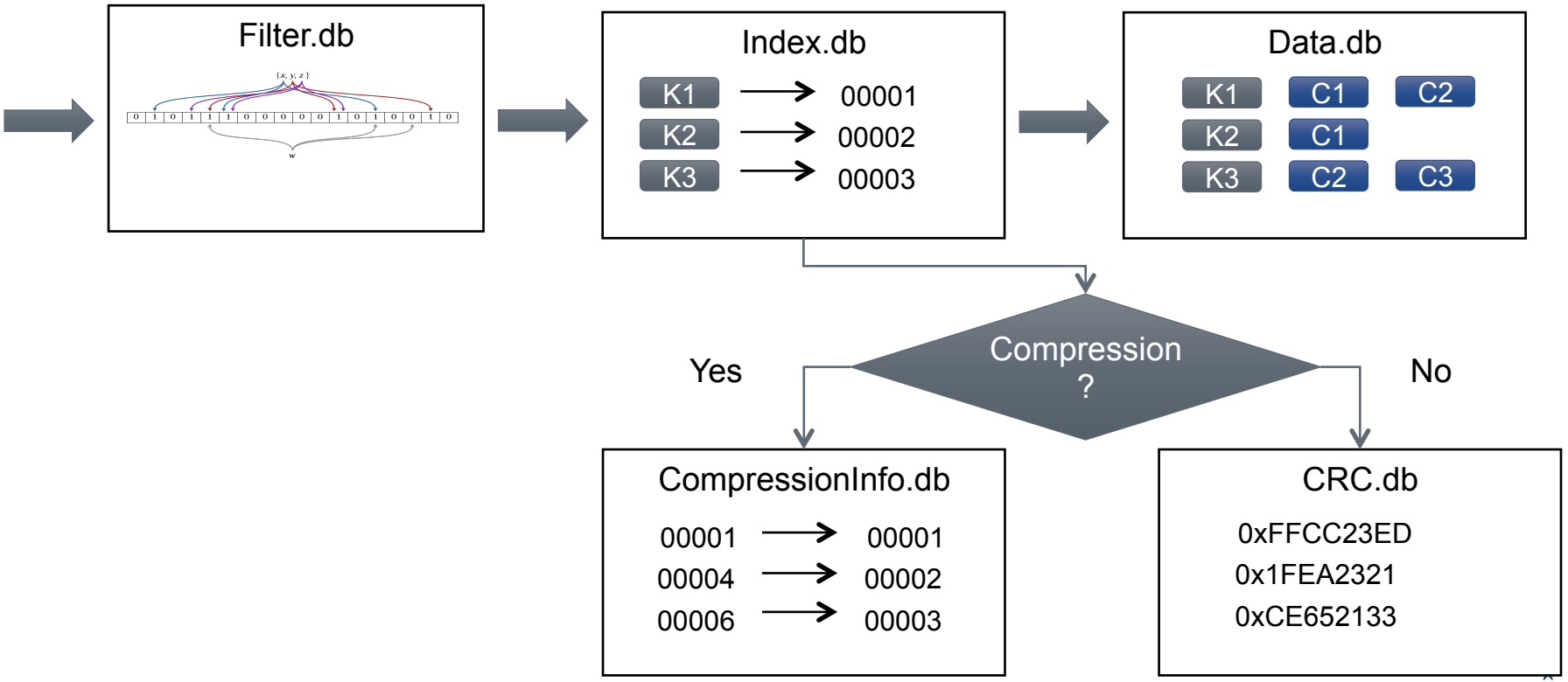
Each node need to read all SSTables for that ColumnFamily + Memtable

- Bloom Filter to prefilter
- Index to direct read

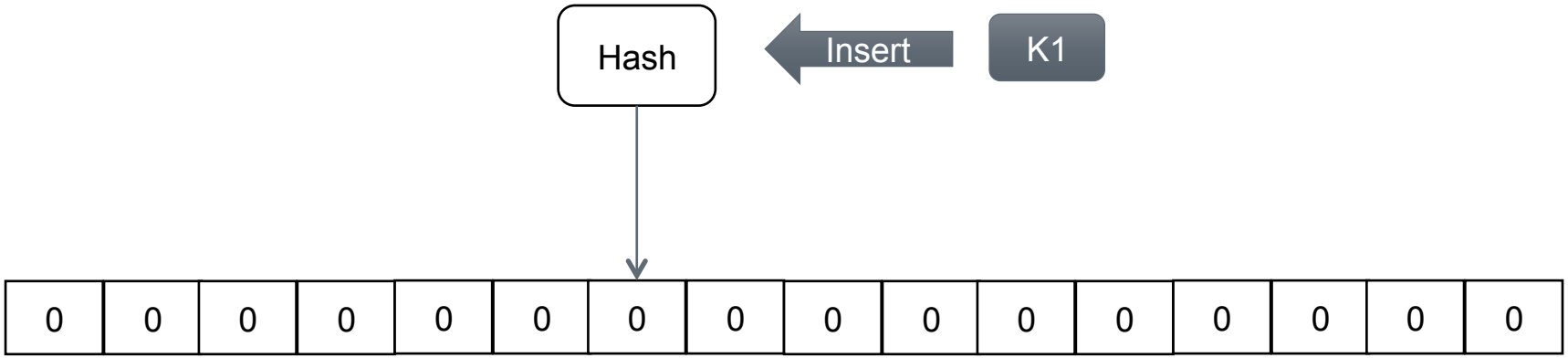
Memtable

- Concurrent**SkipList**Map<RowPosition, AtomicSortedColumns> rows;
- AtomicSortedColumns.Holder
  - DeletionInfo deletionInfo; // tombstone
  - **SnapTree**Map<ByteBuffer, Column> map;
- Goals
  - Fast operations
  - Fast concurrent access
  - Fast in-order iteration
  - Atomic/Isolated operations within a column family

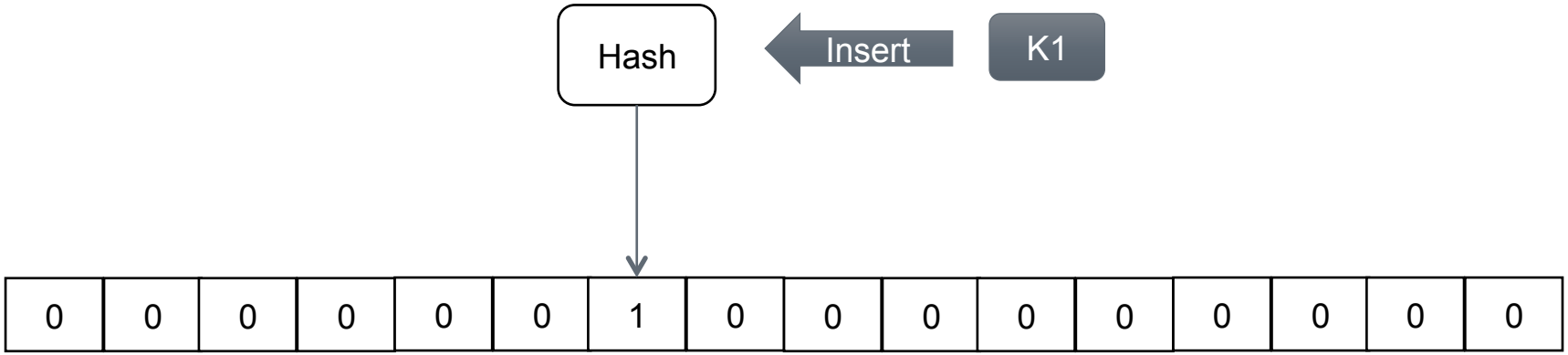
SSTable


Filter.db
{x, y, z}
0 1 0 1 1 1 0 0 0 0 0 1 0 1 0 0 1 0
w
Index.db
K1 → 00001
K2 → 00002
K3 → 00003
Data.db
K1 C1 C2
K2 C1
K3 C2 C3
Compression ?
Yes
No
CompressionInfo.db
00001 → 00001
00004 → 00002
00006 → 00003
CRC.db
0xFFCC23ED
0x1FEA2321
0xCE652133

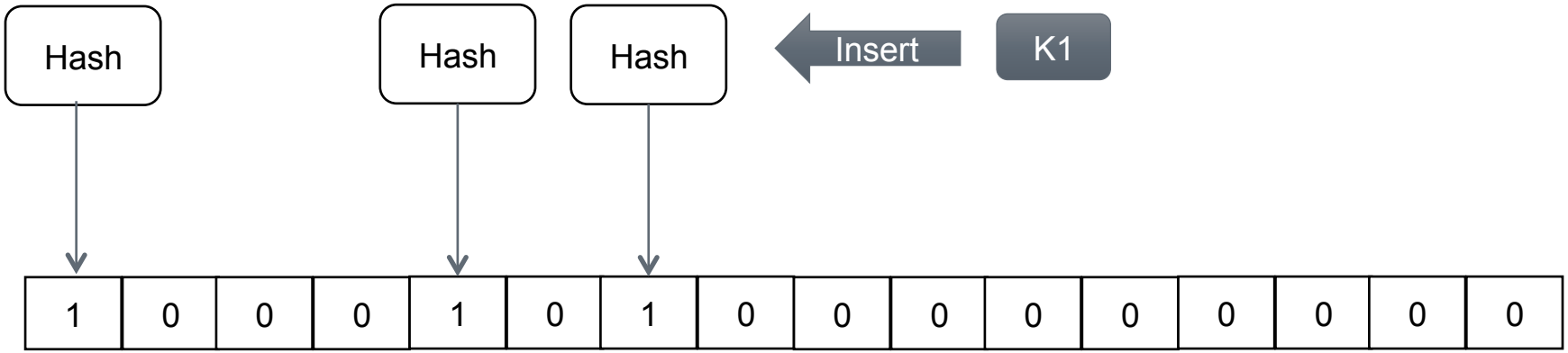
Bloom Filter



Bloom Filter

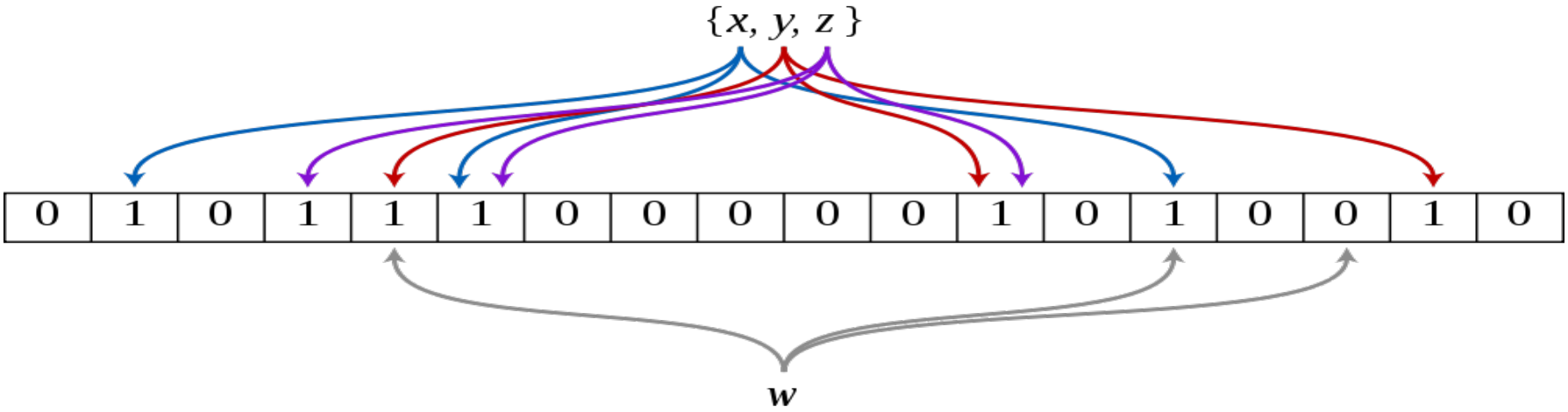


Bloom Filter




```
hash = murmur3(key) # creates two hashes
for i in count(hash):
        result[i] = abs(hash[0] + i * hash[1]) % num_keys)
```

Bloom Filter



Config: *bloom_filter_fp_chance, and*
SSTable: number of rows

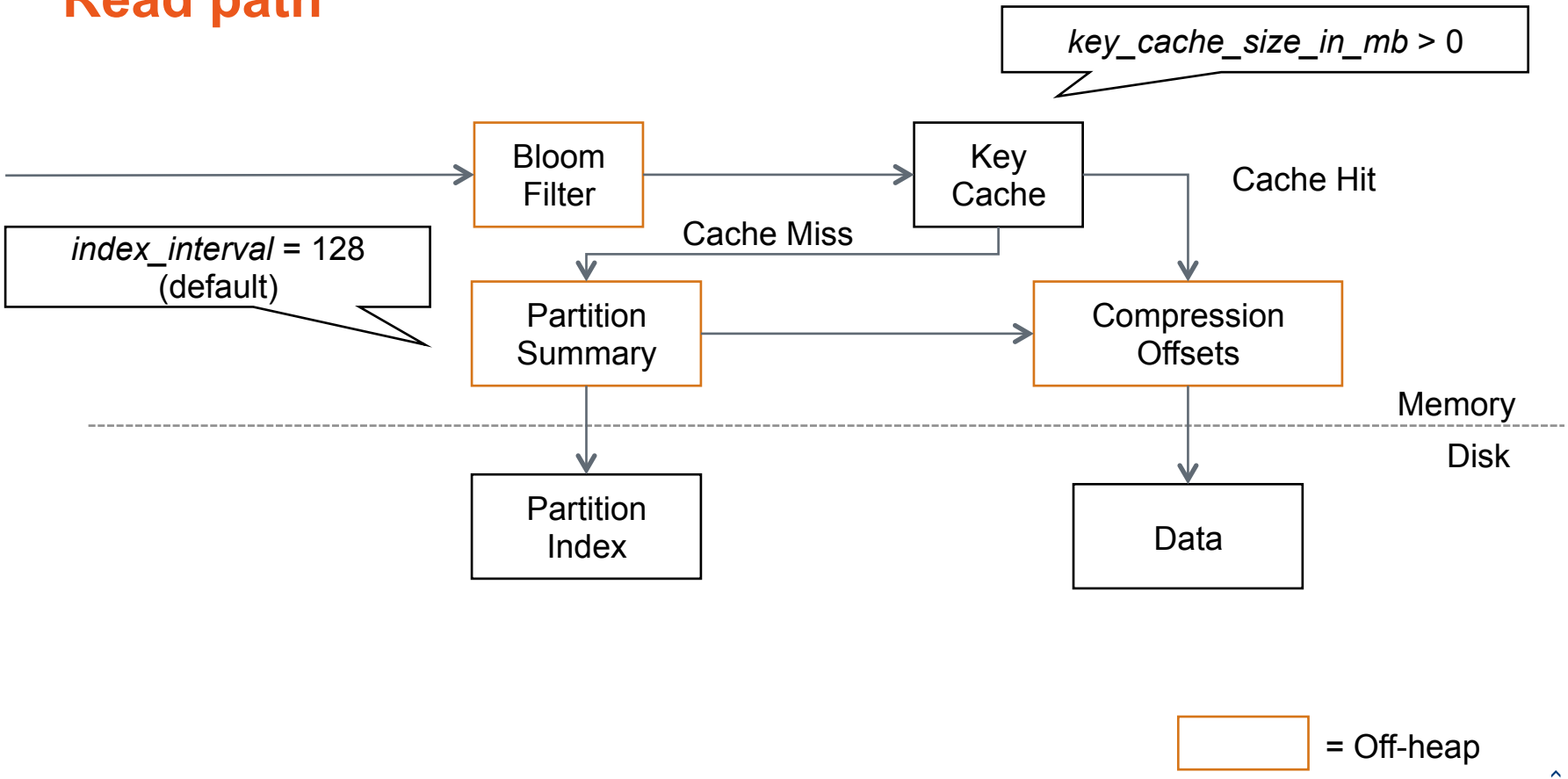
→

Bloom Filter Probability Calculation

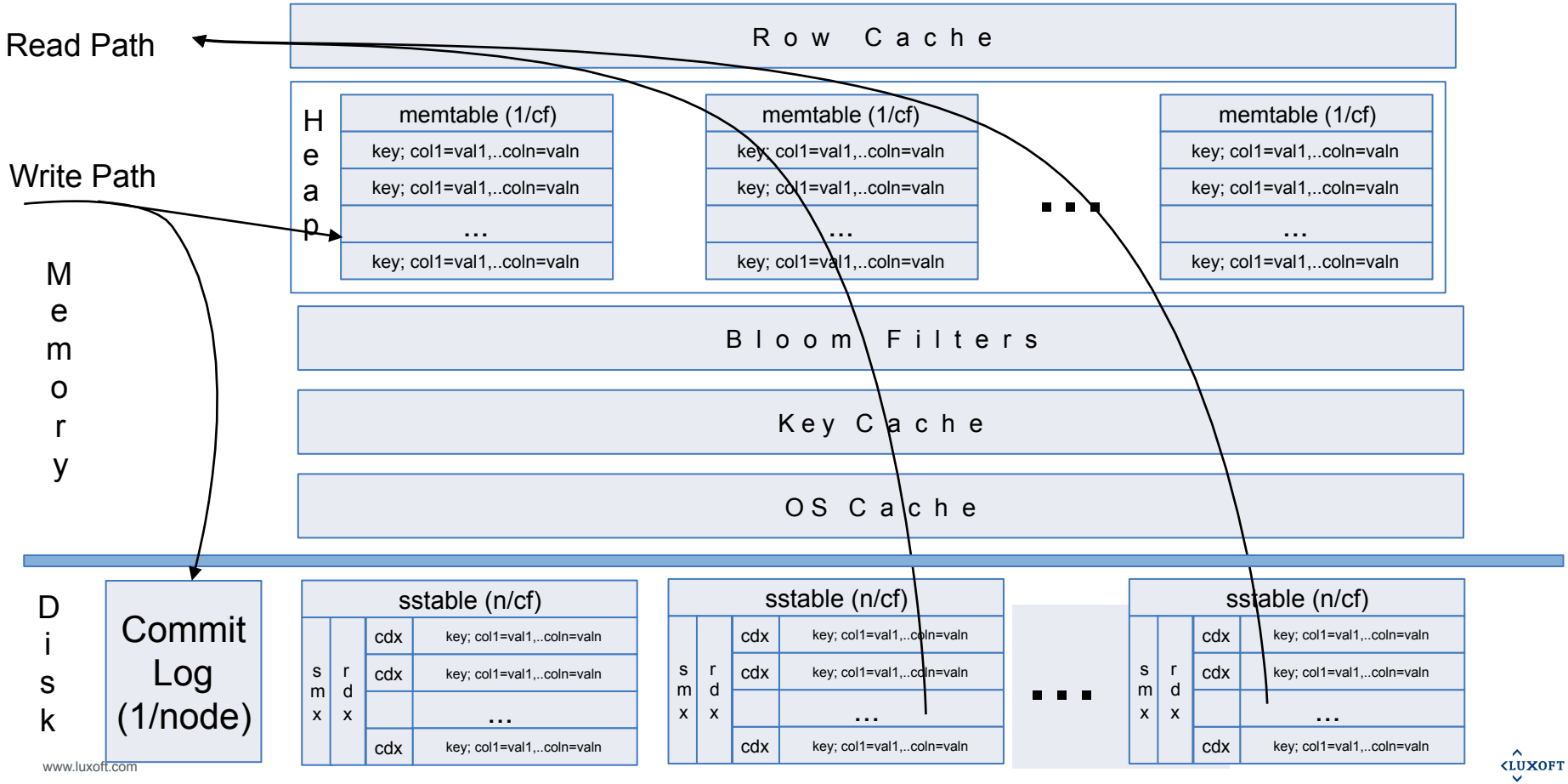
→

Num hashes,
and
Num bits per entry

Read path


key_cache_size_in_mb > 0
Bloom Filter
Key Cache
Cache Hit
Cache Miss
index_interval = 128 (default)
Partition Summary
Compression Offsets
Memory
Disk
Partition Index
Data
= Off-heap

Read / Write Path



C* Consistency

N = replication factor

R = read replica count

W = write replica count

Q = N/2 + 1 = quorum

Consistency: W+R>N

Fast writes: W=1, R=N

Fast reads: W=N, R=1

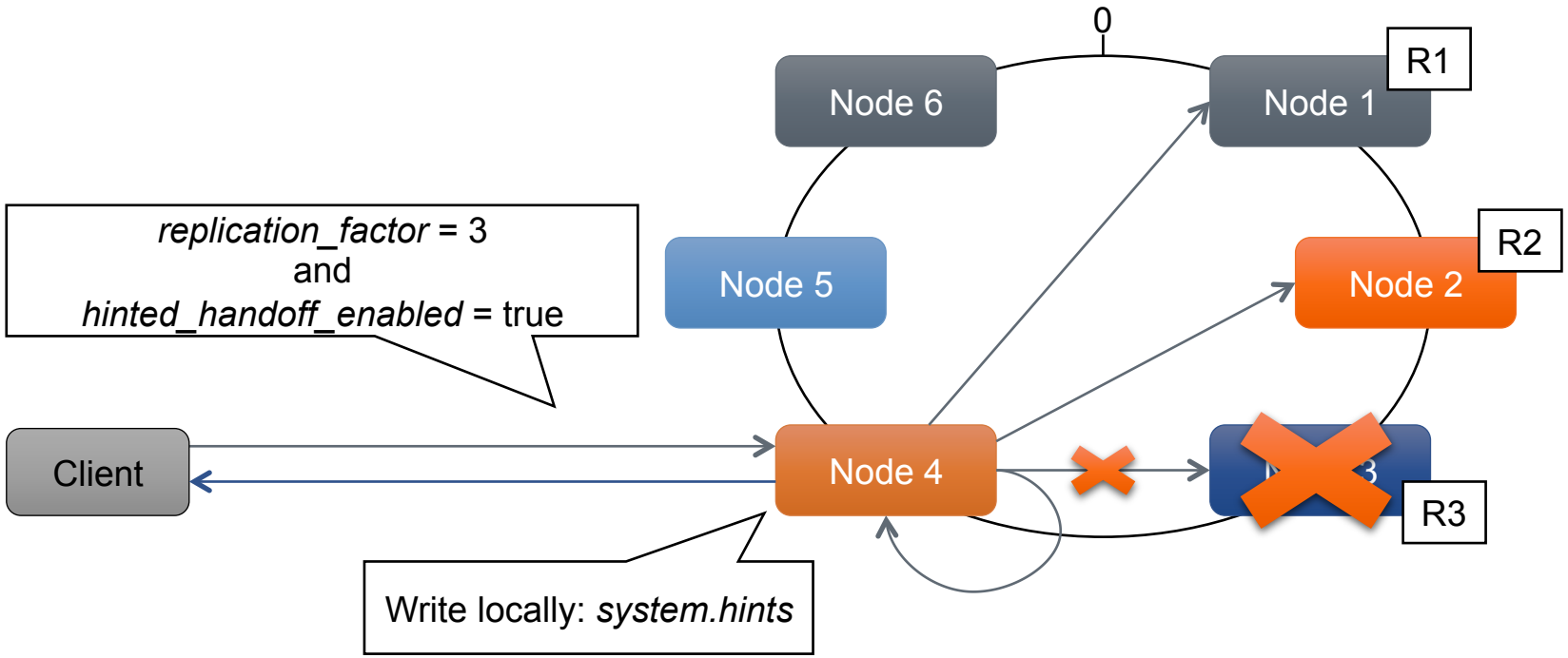
Balanced: W=R=Q

ConsistencyLevel=ONE (1)

ConsistencyLevel=ALL (N)

ConsistencyLevel=QUORUM (Q)

Hinted Handoff



INSERT INTO table (column1, …) VALUES (value1, …) USING CONSISTENCY ANY

Note: Doesn't not count toward consistency level (except ANY)

Hinted handoff

- Full **write availability** when consistency is not required
- Better response consistency after temporary outages (**read repair**)
- Enabled by default

```
hinted_handoff_enabled : (Default: true)
```

- Is an optional part of write path
- Is an optimization
- Have a TTL
- Is unrelated to consistency level

Hinted handoff: details

- Table `system.hints`
  - `target_id uuid` – node ID concerned by the hint
  - `hint_id timeuuid` – hint ID with a timestamp
  - `message_version int` – internal message service version
  - `mutation blob` – actual data being written
- TTL(mutation)
  - `max_hint_window_in_ms`: default 3 hour (10800000)
- `max_hint_delivery_threads`: default 2
- `hinted_handoff_throttle_in_kb`: default 1024 (per second per thread)

Read repair

- For consistency: all replicas have the most recent version of frequently-read data
- Is configured by Column Family

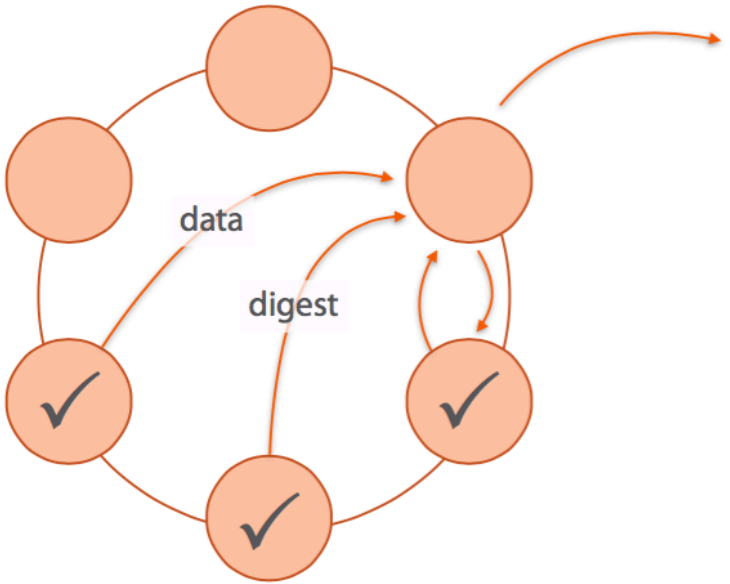
```
read_repair_chance : (Default: 0.1)

dclocal_read_repair_chance : (Deafult: 0.0)
```

- Is a part of read path
- Is a probability to sync data
- Can be Global or DC Local

Read repair: consistent reads

- Determine replicas to invoke
  - ConsistencyLevel vs ReadRepair
- First data node sends full data set, others send digests
- Coordinator waits for ConsistencyLevel
- Compare digests
- If any mismatch
  - Re-request full data sets from same nodes
  - Compare full data sets, send update
  - Block until out-of-data replicas respond
- Return merged data set to client
- Only fixes data that is actually requested. For other data – node repair needed.



Anti-entropy node repair

- Ensure that all data on a replica is made consistent
- Repair inconsistencies on a node that has been down for a while
- When
  - During normal operations as a part of regular maintenance
  - During node recovery after a failure / long outage
  - On nodes with data that is not read frequently
- Manual repair

```
nodetool repair <keyspace> [table] <opts>
```

Anti-entropy node repair: How

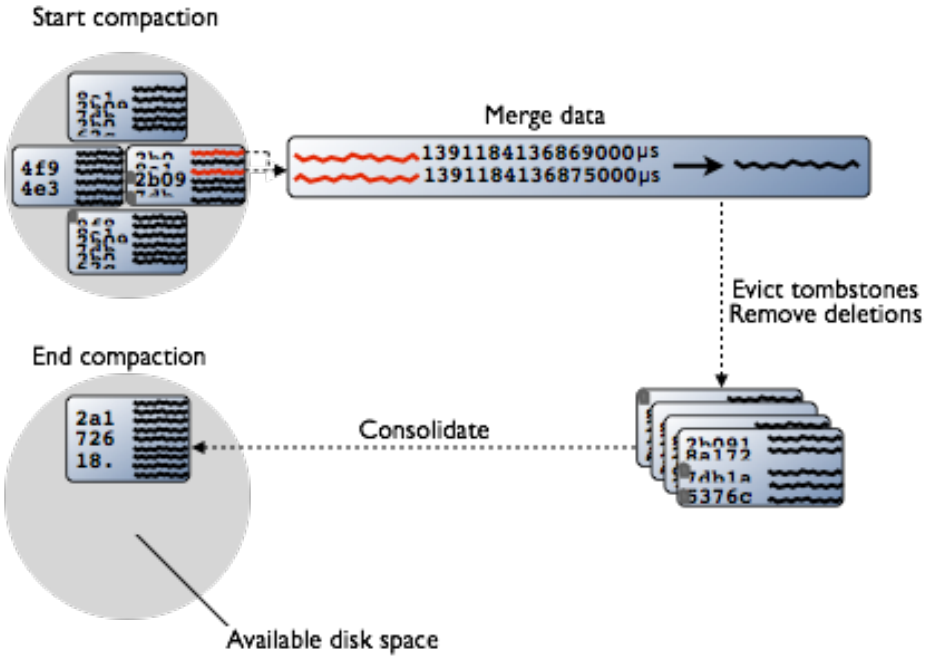
- Determine peer nodes with matching ranges
- Triggers a major validation compaction on peers
  - Do a read part of the compaction stage
- Read and generate the Merkle Tree on each peer
  - Lots of I/O and CPU
- Send trees to inititator
- Initiator compares every tree to every other tree
- If any differences, nodes exchange conflicting ranges
  - Writen out as new local SSTables
  - Overstreaming of partitions

Anti-entropy options

- `-pr (--partitioner-range)`: repairs only the main partition range for that node
  - suitable for periodic repair maintenance; need to do repair on each node
  - unsuitable for recovering: other replicas need to be repaired too
- `-snapshot`: sequential repairs; only one replica at a time do computation; default since 2.0.2
- `-par`: parallel repairs
- `-st`: start token
- `-et`: end token

Compaction

- Merges SSTables with same key
- Finally remove tombstoned data



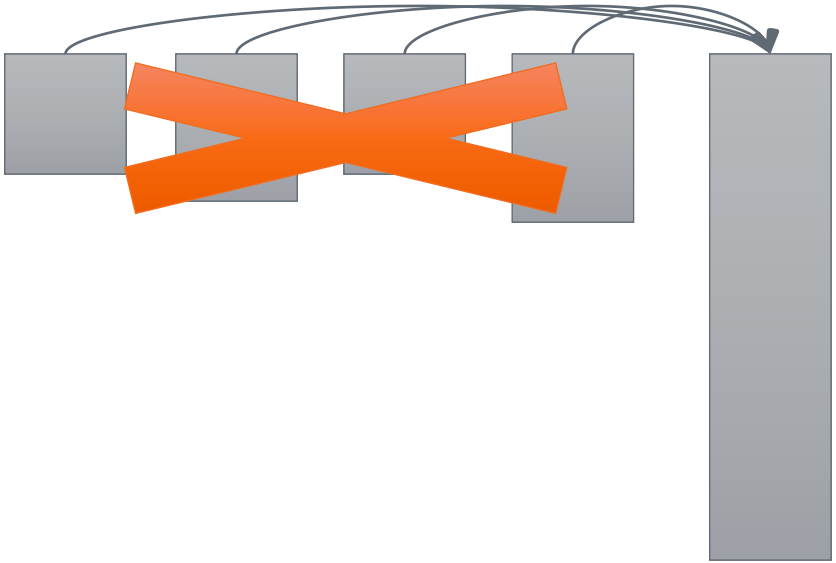
Minor Compaction

- Merge changed SSTables of same size
  - rebuild Bloom Filter and Index
- Started automatically when at least min_compaction_threshold (default: 4) SSTables written

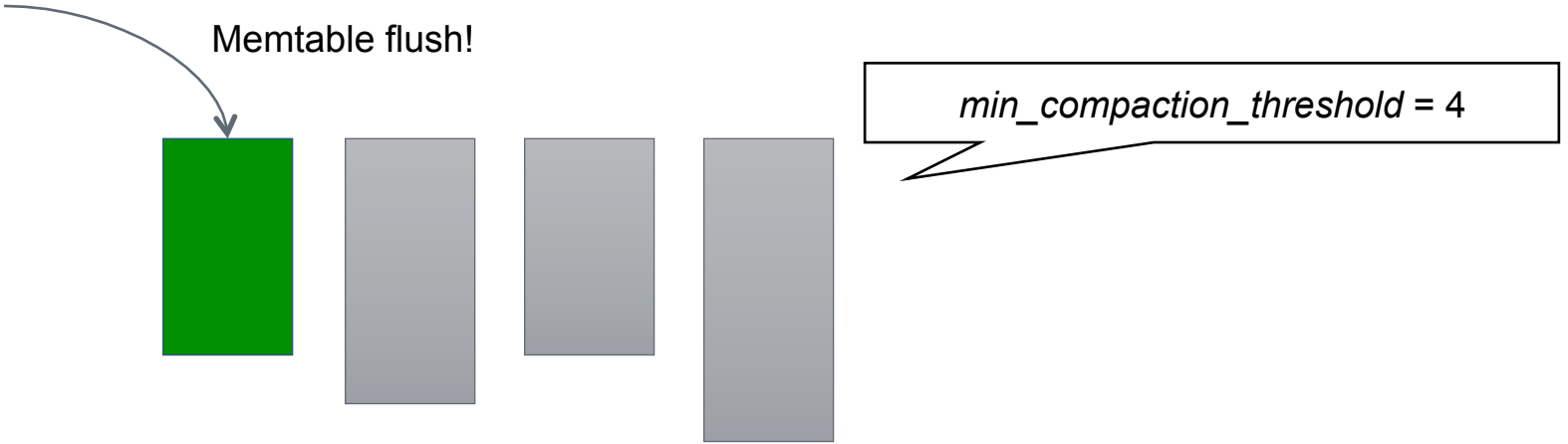
Major Compaction

- Merge all SSTables
- Discard all tombstones
- Manual start
  - `nodetool compact`

Compaction (Size-tiered)

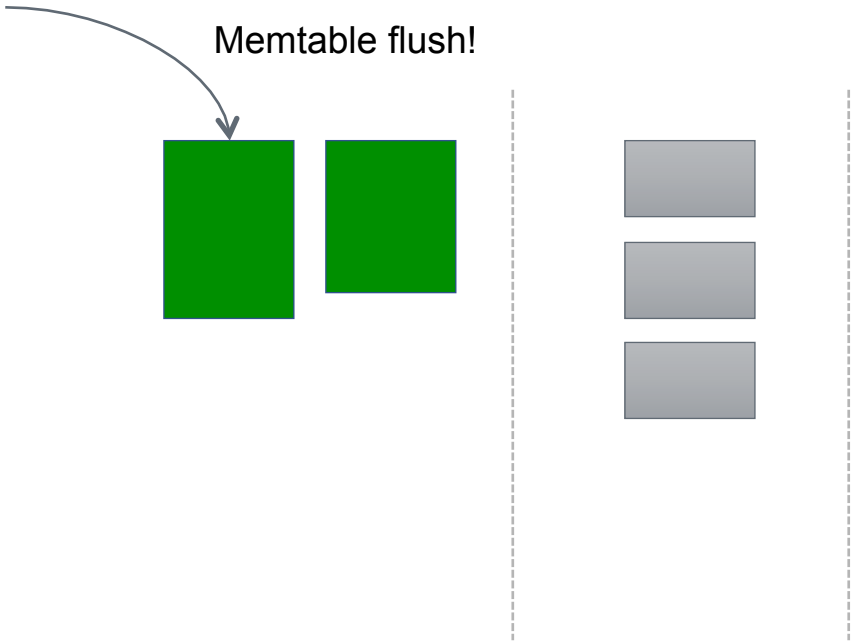


Compaction (Size-tiered)



Compaction (Size-tiered)

Compaction (Leveled)



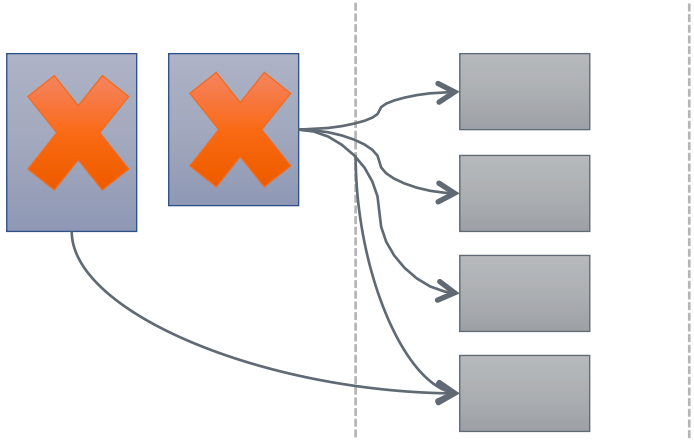
Compaction (Leveled)

sstable_size_in_mb = 160

L0: 160 MB

L1: 160 MB x 10

L2: 160 MB x 100

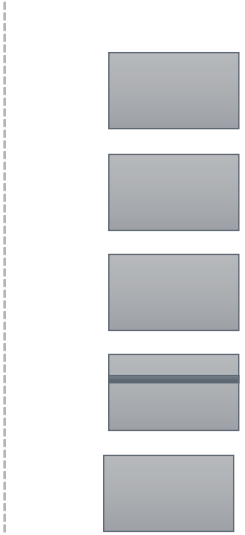


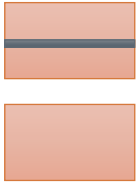
Compaction (Leveled)

L0: 160 MB

L1: 160 MB x 10

L2: 160 MB x 100





…

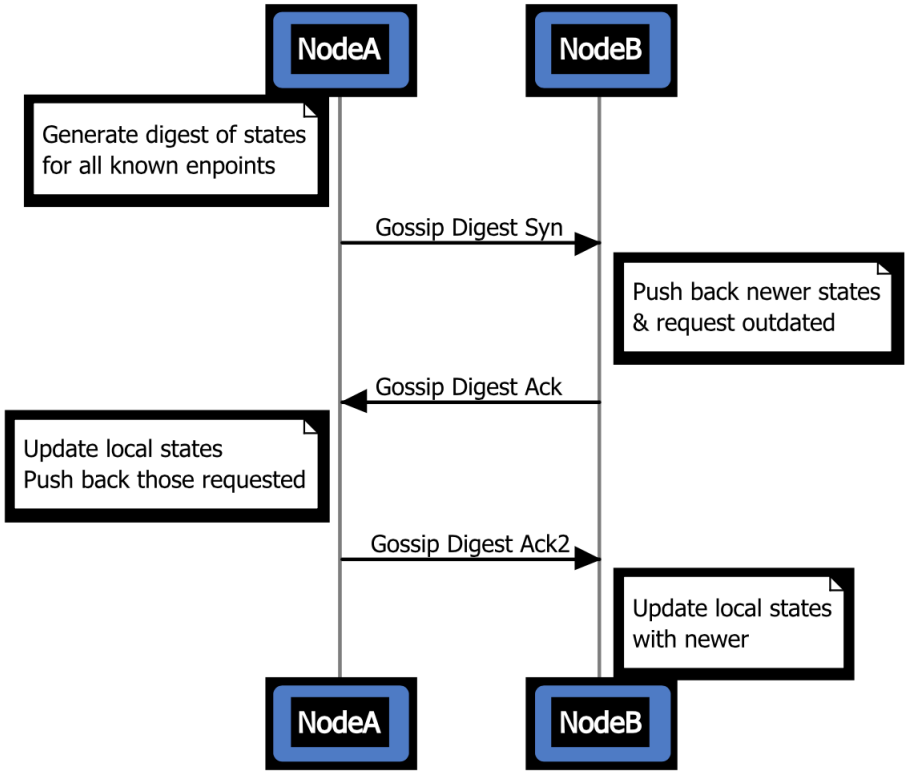
Cluster State Sync: Gossip

- State to exchange
  - Status, Token, Schema version, DC&Rack, Addresses, Data size, Health
- Each node keeps a list of all nodes
  - alive and dead
    - Failure detector (Phi accrual) uses suspicios level for alivenes level
- Each node every second gossip to
  - 1 live node
  - maybe 1 dead node
  - maybe 1 seed (if none of the above – first round)

Gossip: what it DOESN’T do

- Notify about up/down nodes
- Propagate schema
- Transmit data files
- Distribute mutations

Gossip


NodeA
NodeB
Generate digest of states
for all known enpoints
Gossip Digest Syn
Push back newer states
& request outdated
Gossip Digest Ack
Update local states
Push back those requested
Gossip Digest Ack2
Update local states
with newer
NodeA
NodeB

Thank you!

Questions?