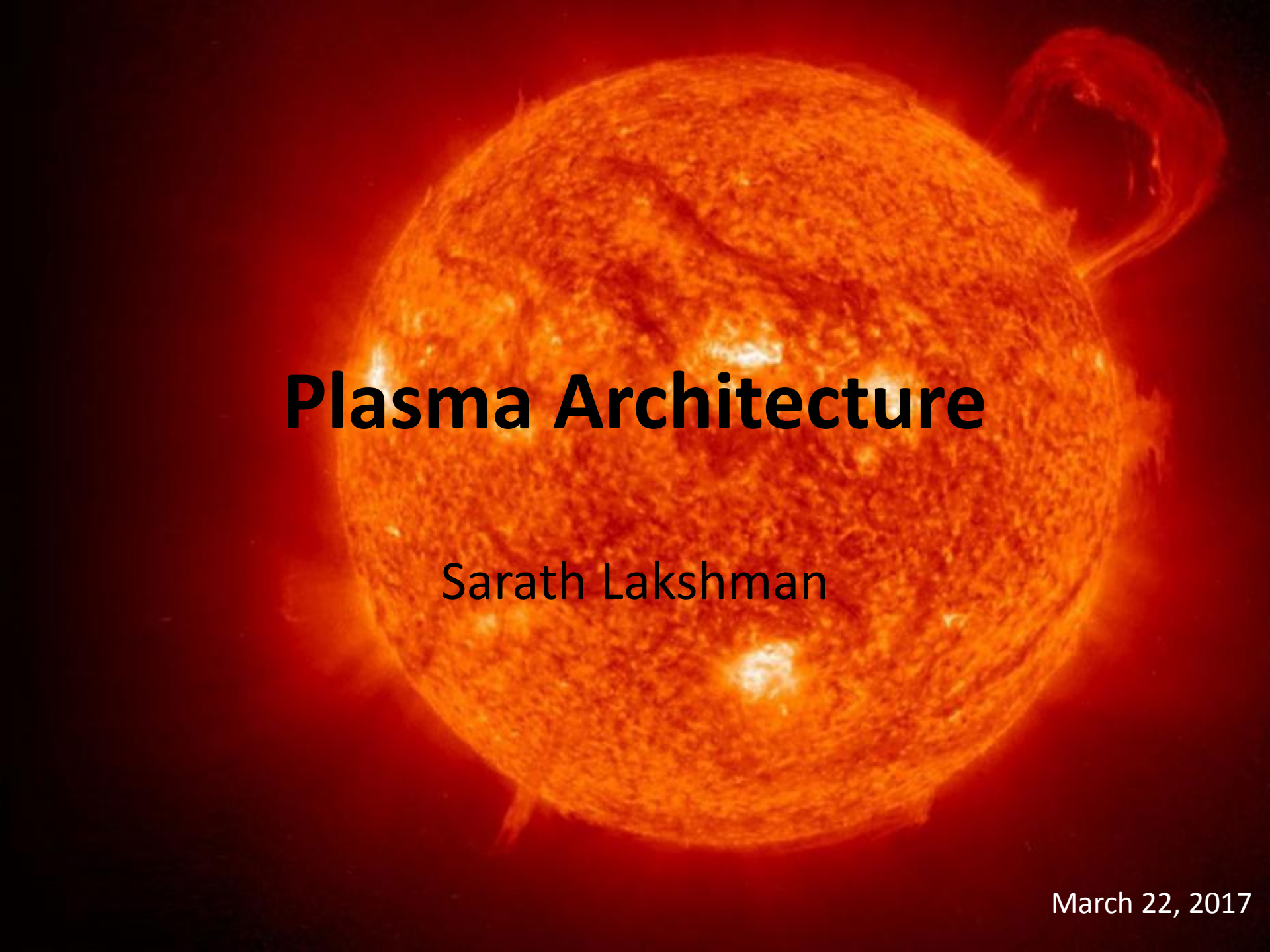


Plasma Architecture

Sarath Lakshman

March 22, 2017

Overview

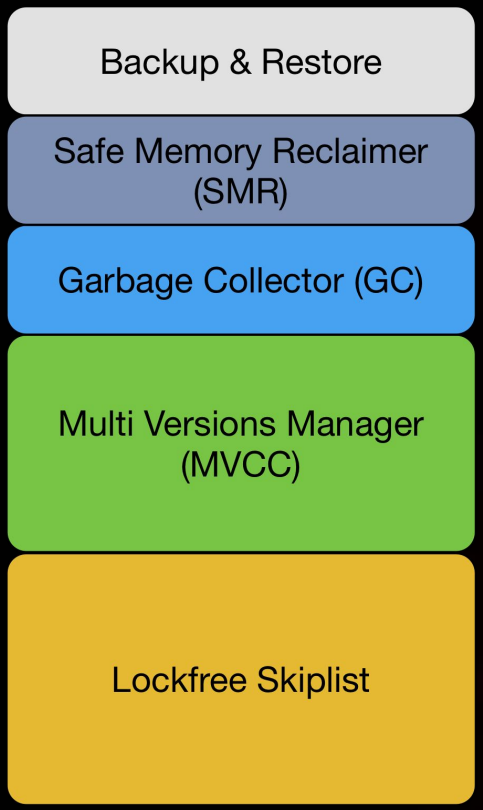
- Background
- Design Considerations
- Architecture
- Enhancements
- Conclusion

Background

- Memory Optimized Indexes rocks!
  - Scale performance linearly with more CPU cores, DRAM
  - Low latency fast in-memory snapshots every 20ms
  - Lock-free data structures everywhere leverages many readers and writers
  - Persistence using non-intrusive full snapshot backups
  - Highly optimized to avoid the cost of back-index (cpu and memory)

Background

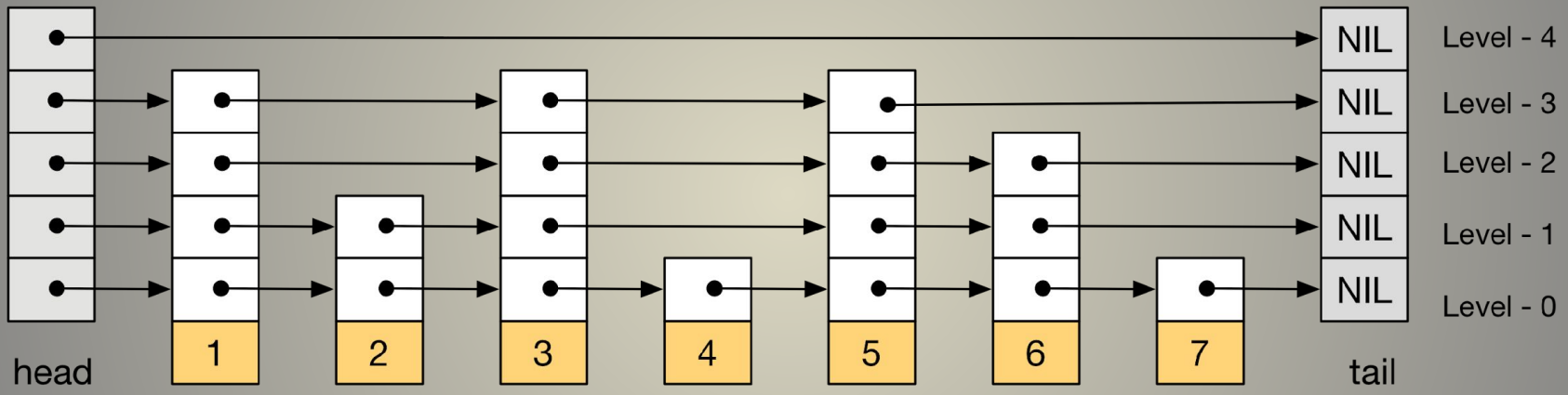
Nitro Storage Engine



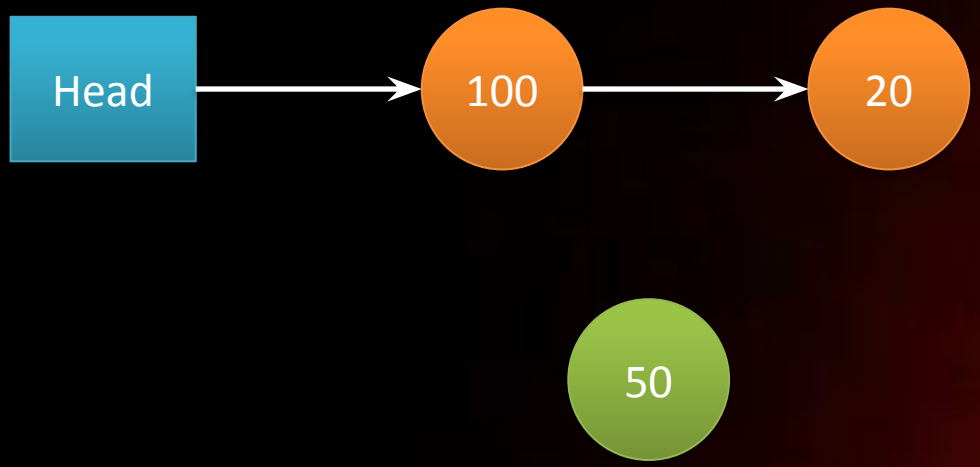
- Create backups from snapshots and recover nitro after restart/crash
- Free items when GCed and not in reference
- Remove items from skiplist which belongs to the unused snapshots
- Create point-in-time immutable snapshots for index scans
- Avoid phantoms and provide scan stability
- Manage index snapshot versions in use
- Implements Insert, Delete, Lookup, Range Iteration
- Concurrent partitioned visitors
- Concurrent bottom-up skiplist build

Background

- Skiplist



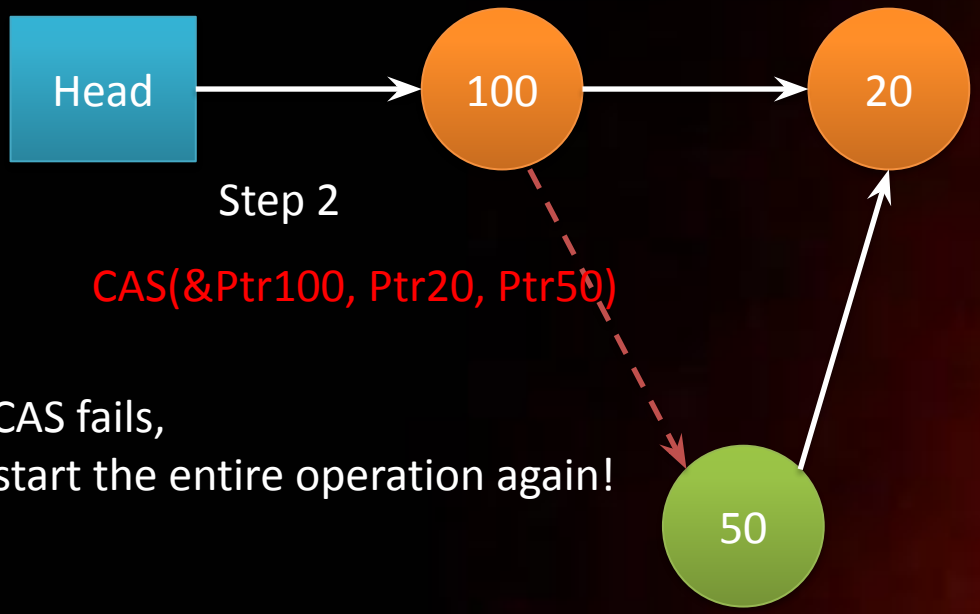
Lockfree linked list



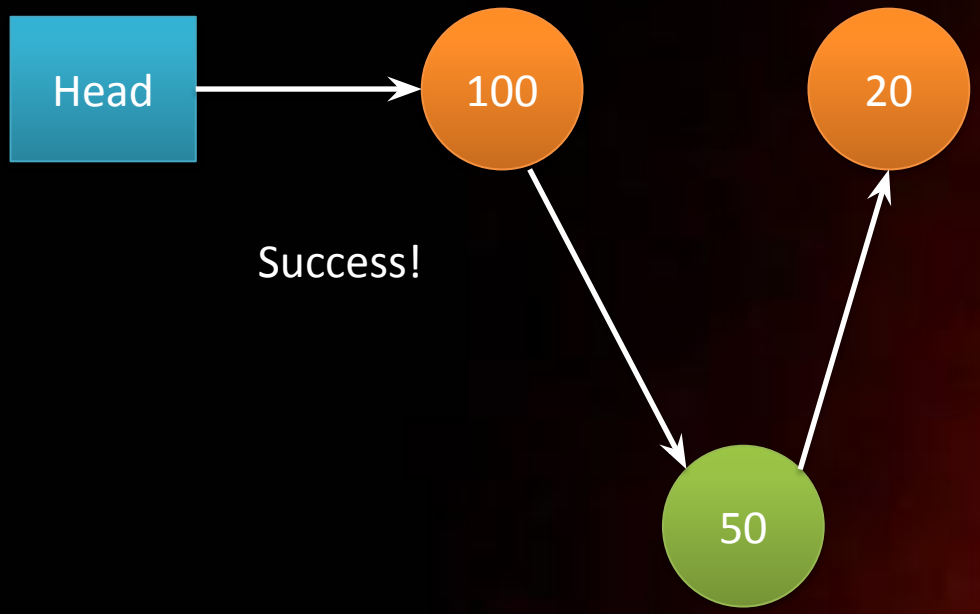
Insert 50 between 100 and 20

Lockfree linked list

Lockfree linked list



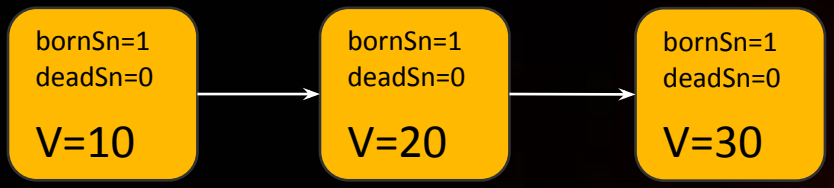
Lockfree linked list



* All the linked list operations illustrated in the upcoming slides are atomic linked list operations

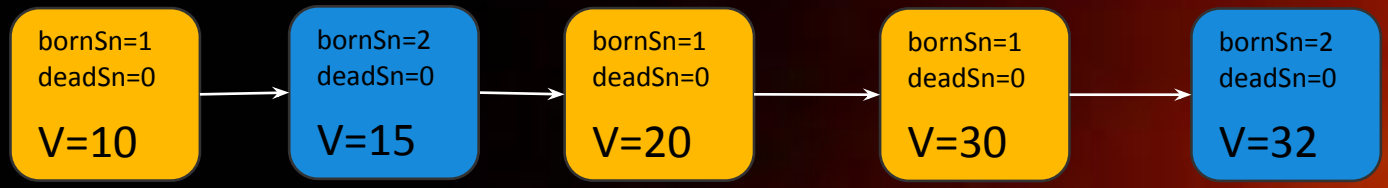
Background

- Nitro MVCC



Lifetime (bornSn, deadSn)

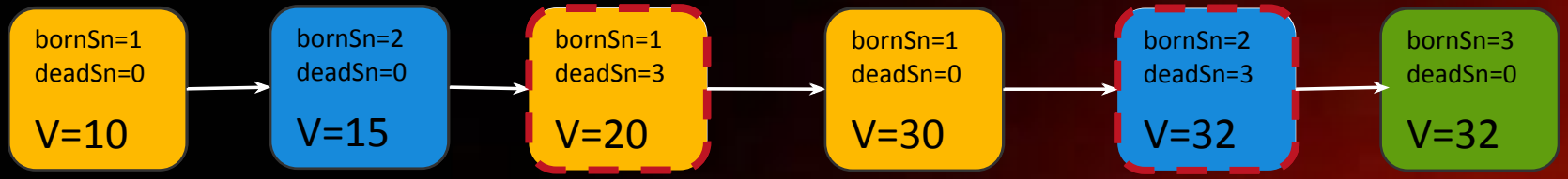
Create Snapshot (Sn=1)



Create Snapshot (Sn=2)

Background

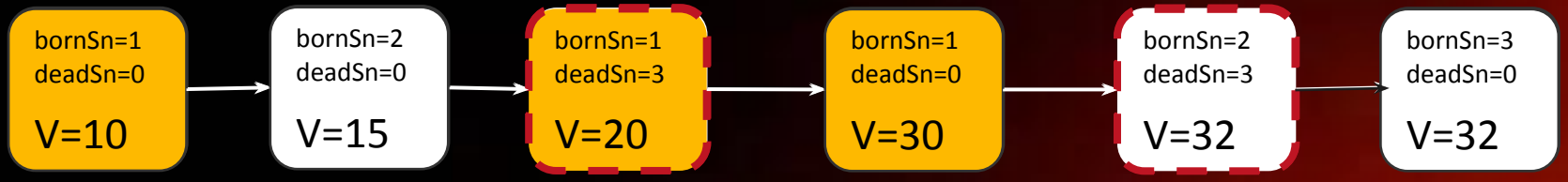
- Nitro MVCC Recap

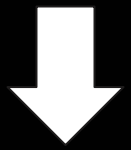


Create Snapshot (Sn=3)

Background

- Nitro MVCC Recap

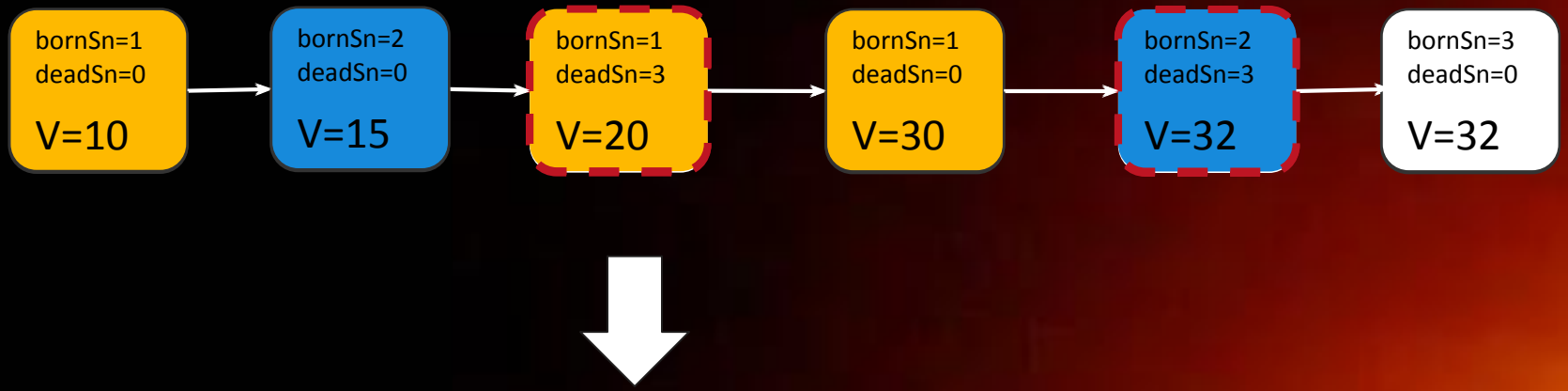




Visibility: Iterator (Sn=1)

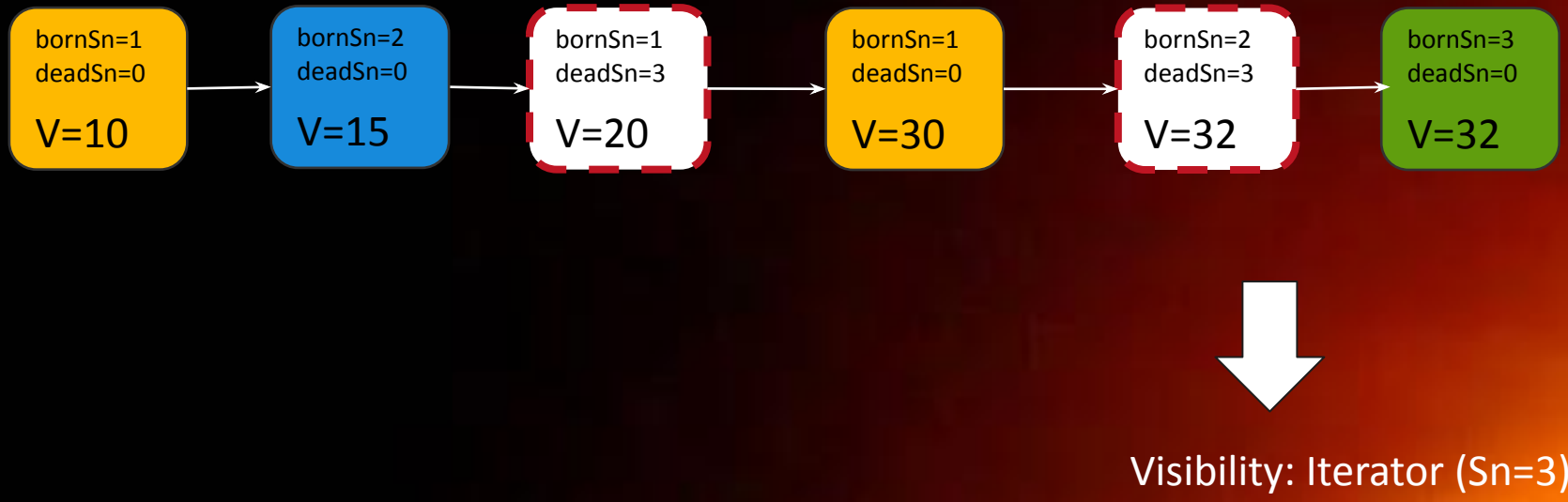
Background

- Nitro MVCC Recap



Background

- Nitro MVCC Recap



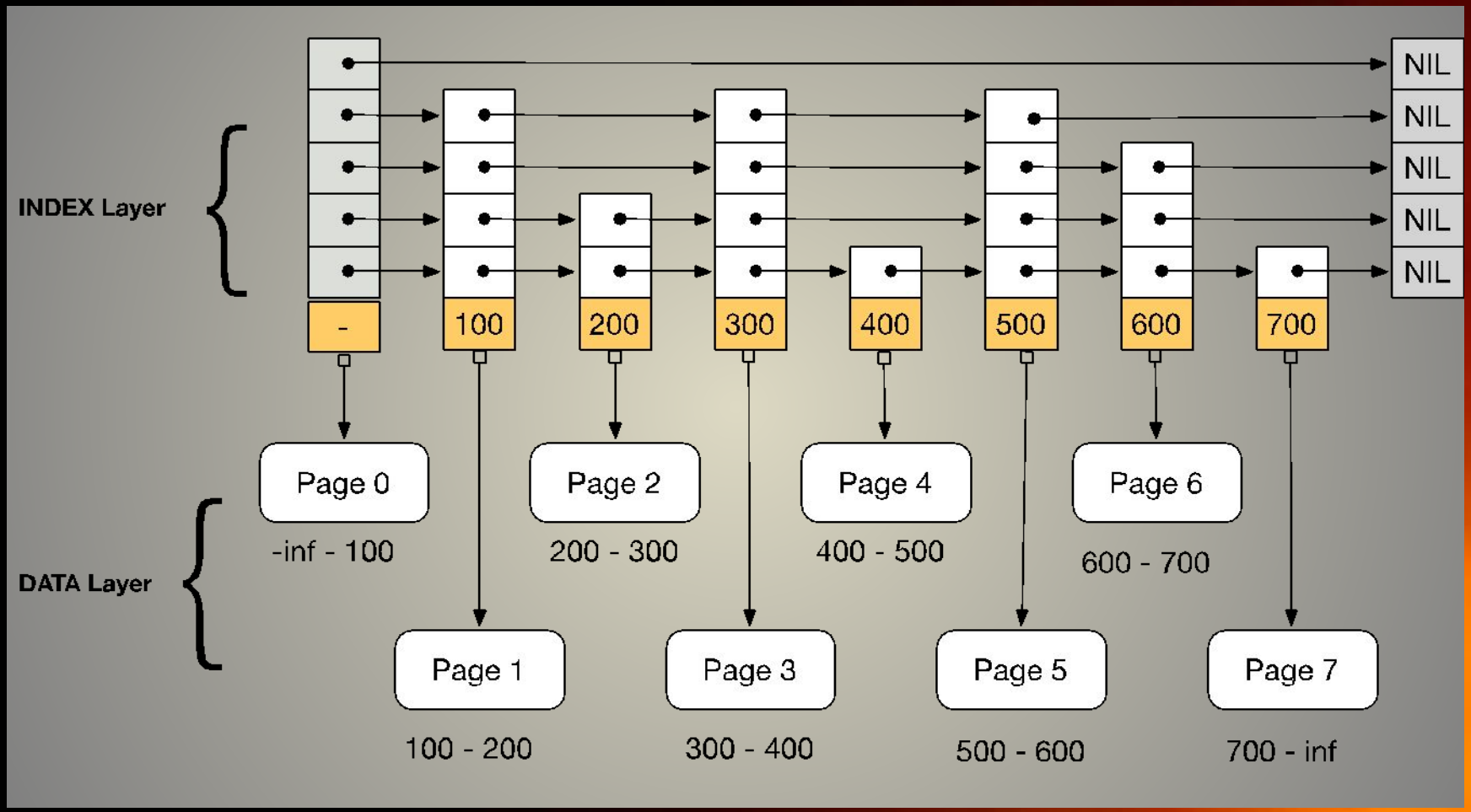
Background

- Memory Optimized Index++
  - Fully memory resident is expensive (Best option for predictable size index with random access)
  - A cache solution that operates between DRAM and Flash is more desirable
  - Full snapshot persistence is expensive as it has significant impact on SSD lifetime
  - Thoughts on extending Nitro - ended up building another storage engine based on same principles

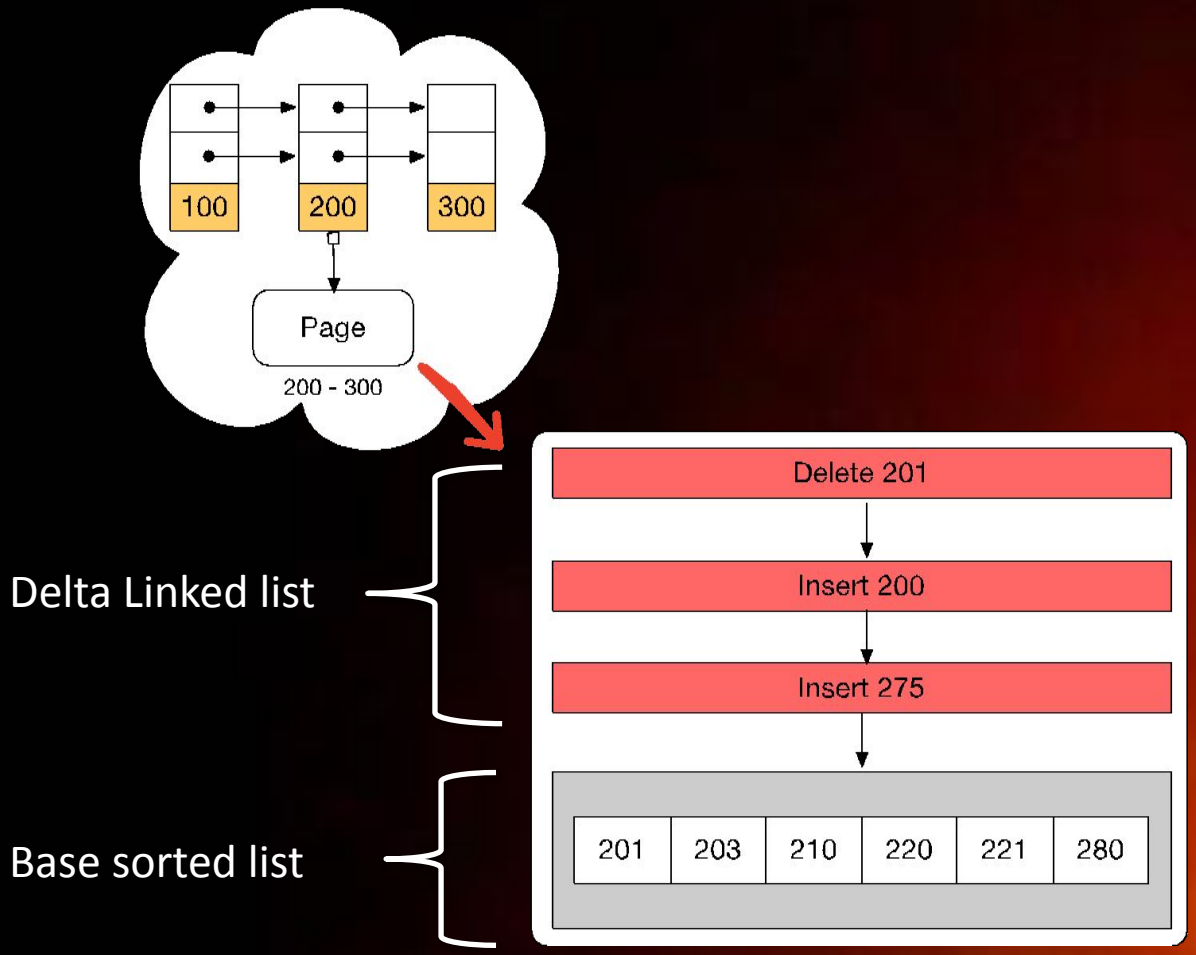
Design Considerations

- Scalability
  - Scale storage engine subsystems almost linearly
- Multicore CPUs
  - Lock-free data structures
- Memory first architecture
  - Persist only when required
- Flash/SSD trends
  - Very good read performance
  - Log structured append only storage
- Fast in-memory snapshots
- Ability to Rollback

Architecture

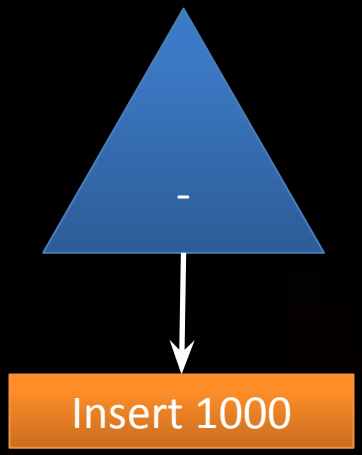


Page


100
200
300
Page
200 - 300
Delete 201
Insert 200
Insert 275
201
203
210
220
221
280
Delta Linked list
Base sorted list

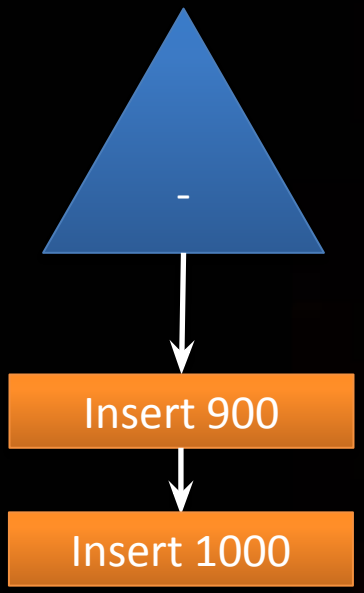
Insert

Index layer



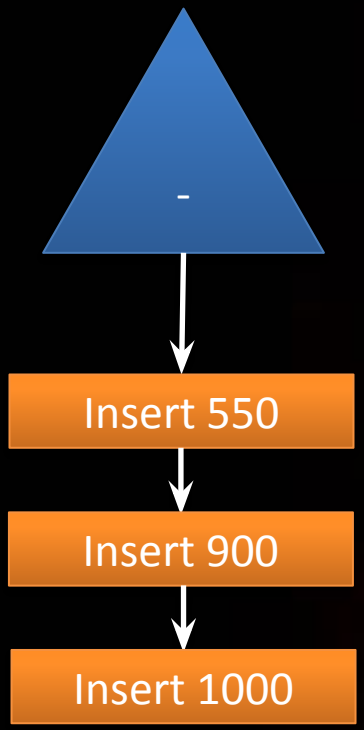
Insert

Index layer



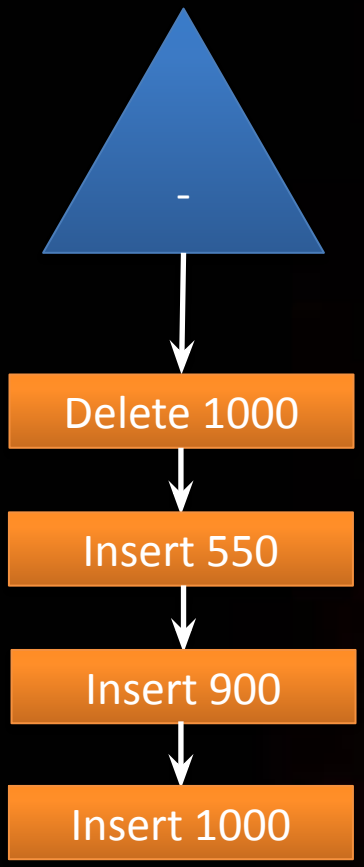
Insert

Index layer



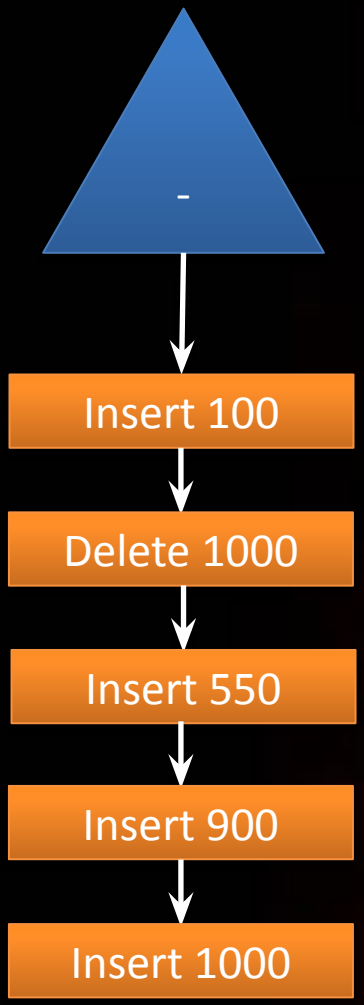
Delete

Index layer



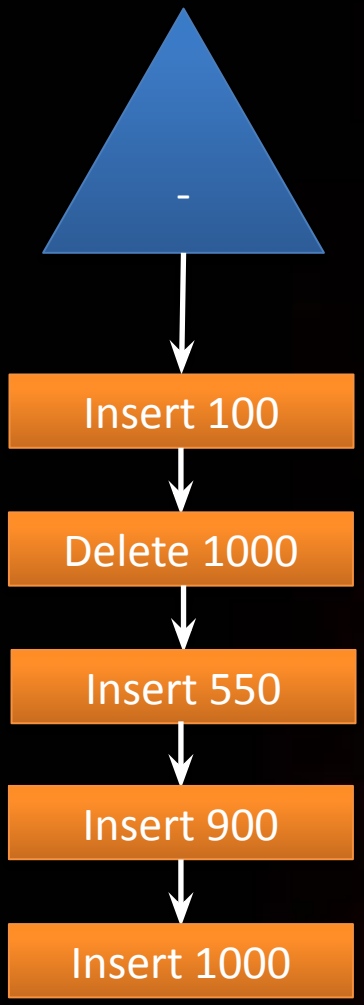
Insert

Index layer



Insert

Index layer

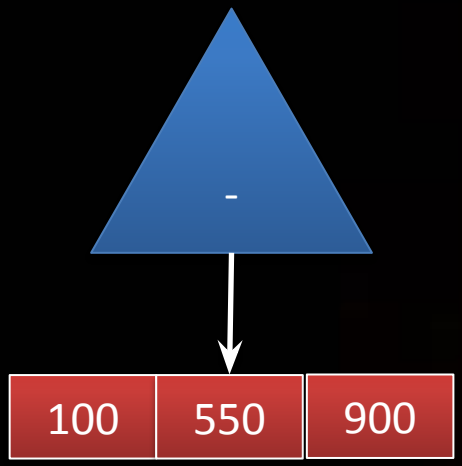


Number of delta operations > Compaction Threshold

Let’s Compact

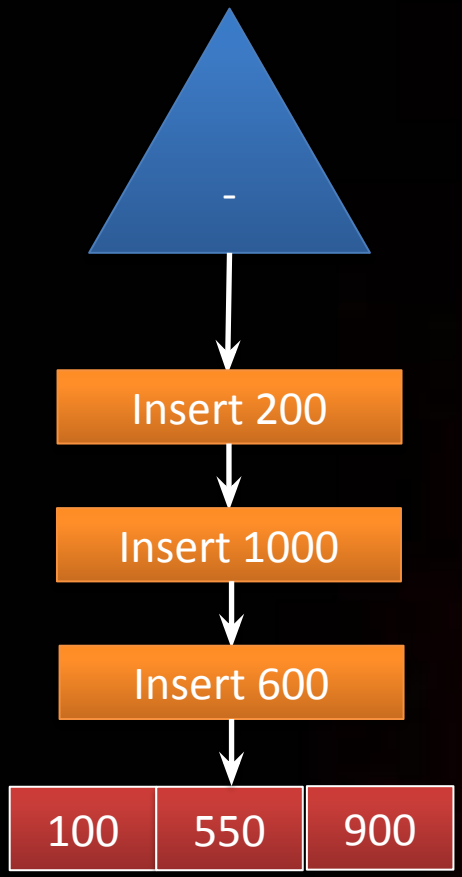
Compaction

Index layer

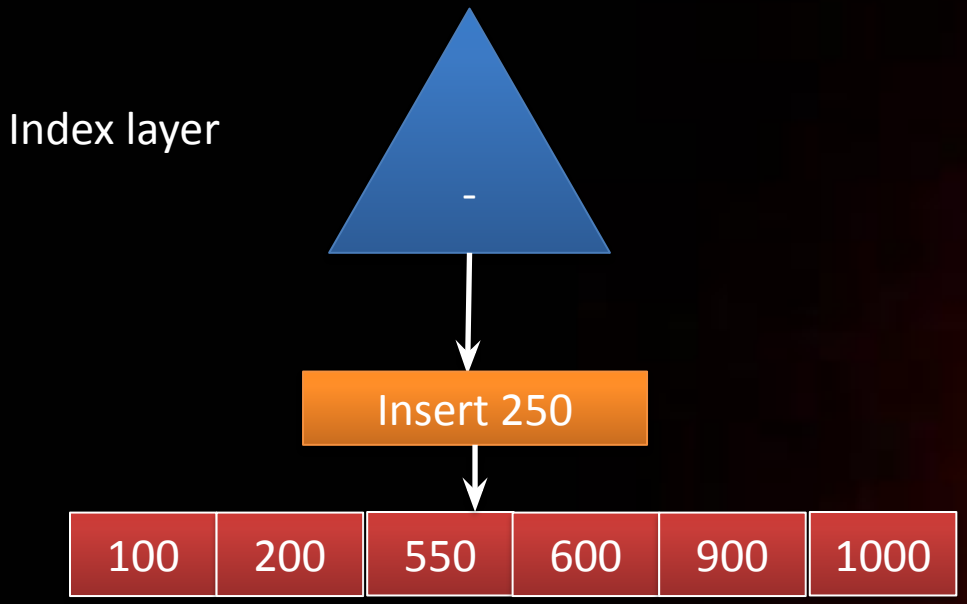


Insert

Index layer



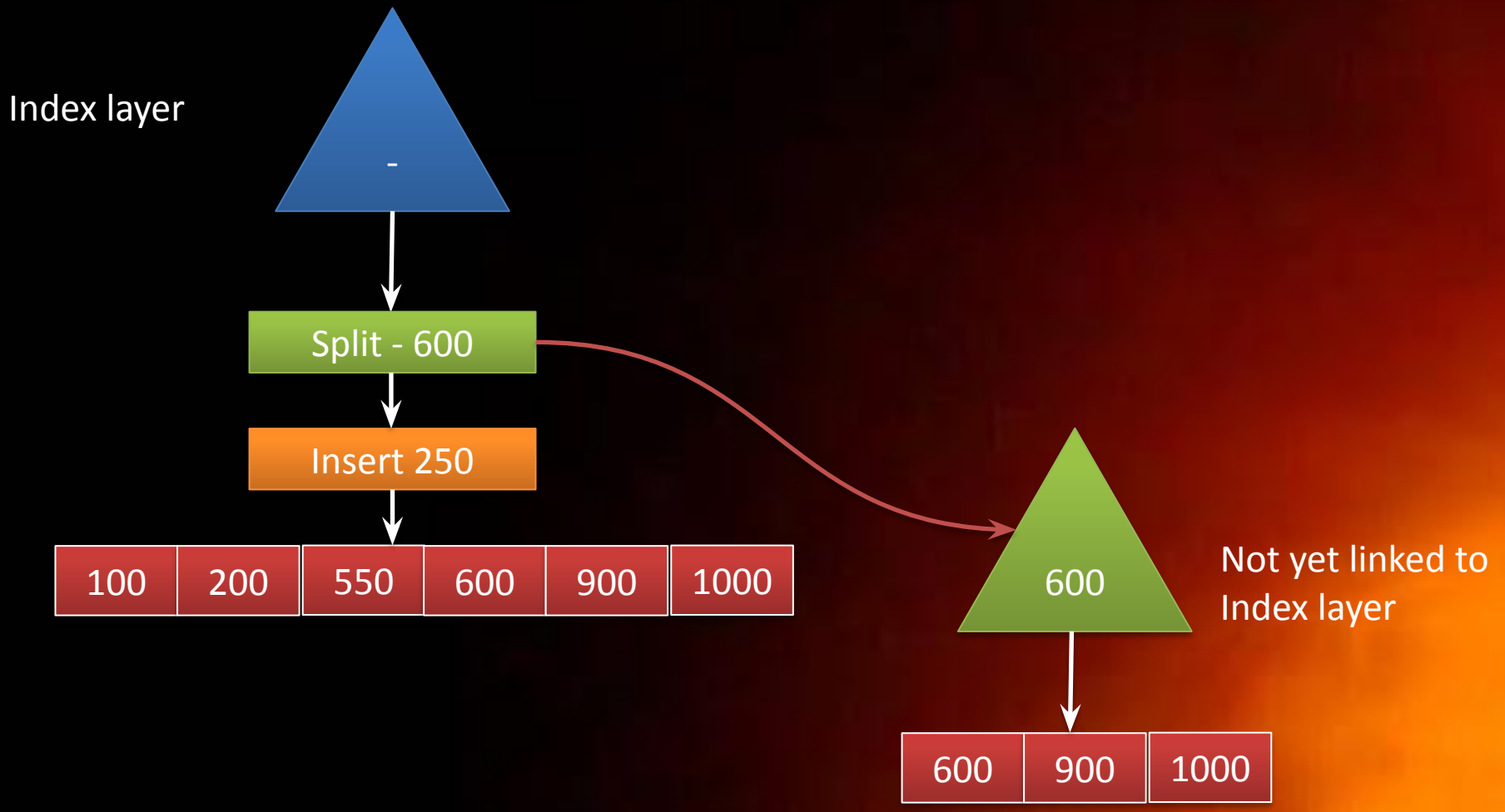
Split



Page has too many items!

Let’s split the page

Split

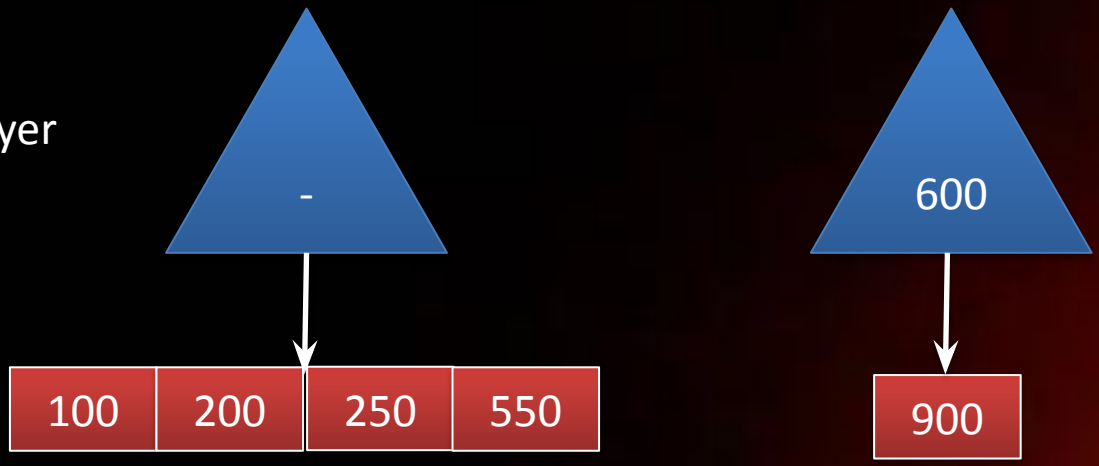

Index layer
-
Split - 600
Insert 250
100
200
550
600
900
1000
600
Not yet linked to Index layer
600
900
1000

Split


Index layer
-
600
Linked to the index layer
Split - 600
600
900
1000
Insert 250
100
200
550
600
900
1000

Merge

Index layer

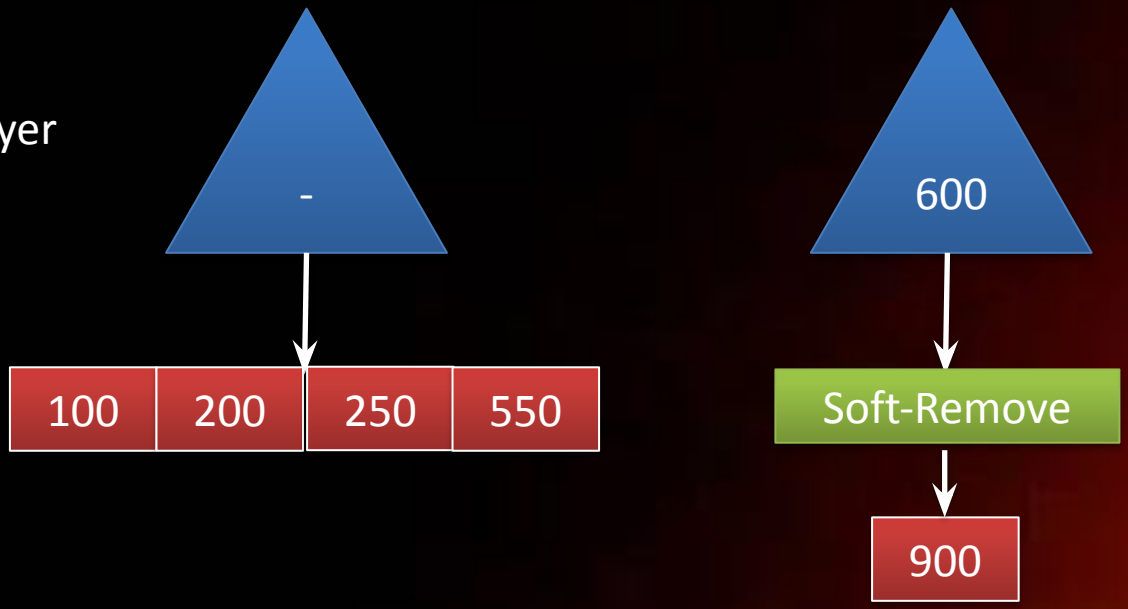


Page is under filled

Let’s merge

Merge

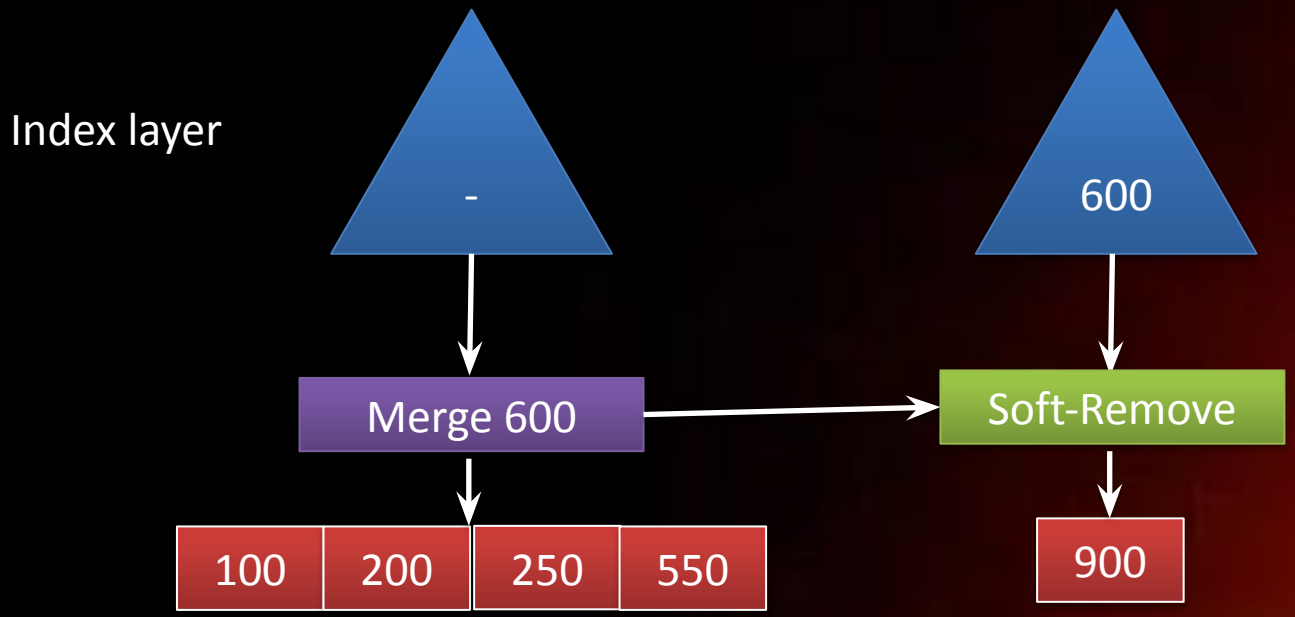
Index layer



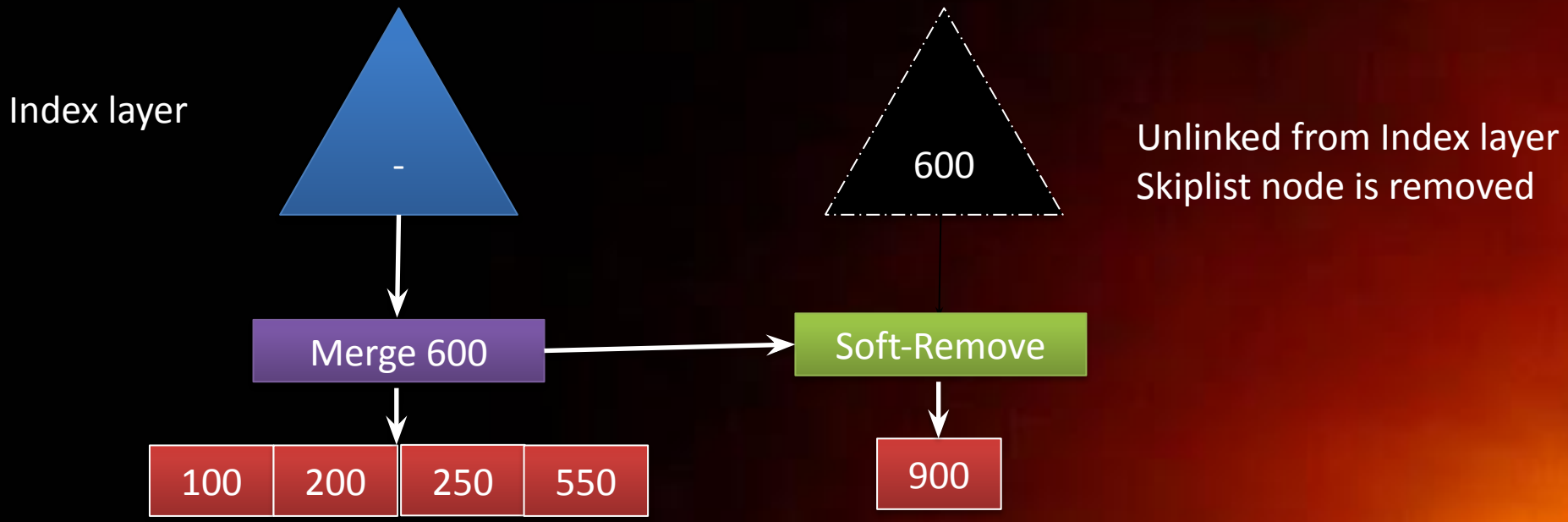
Writer threads will avoid modifying this page as it observes soft-remove delta

The corresponding thread should complete merge operation before inserting

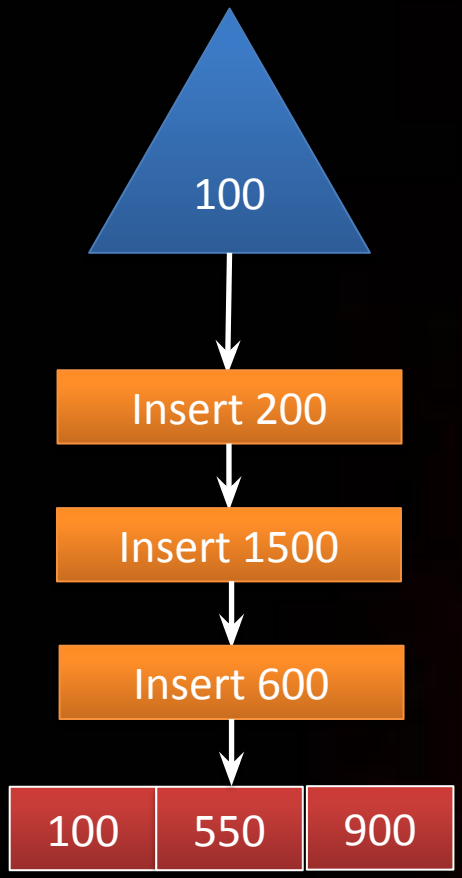
Merge

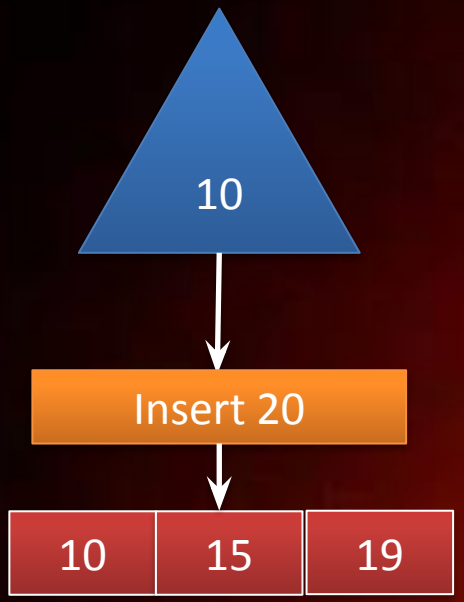


Merge


Index layer
-
600
Unlinked from Index layer
Skiplist node is removed
Merge 600
Soft-Remove
100
200
250
550
900

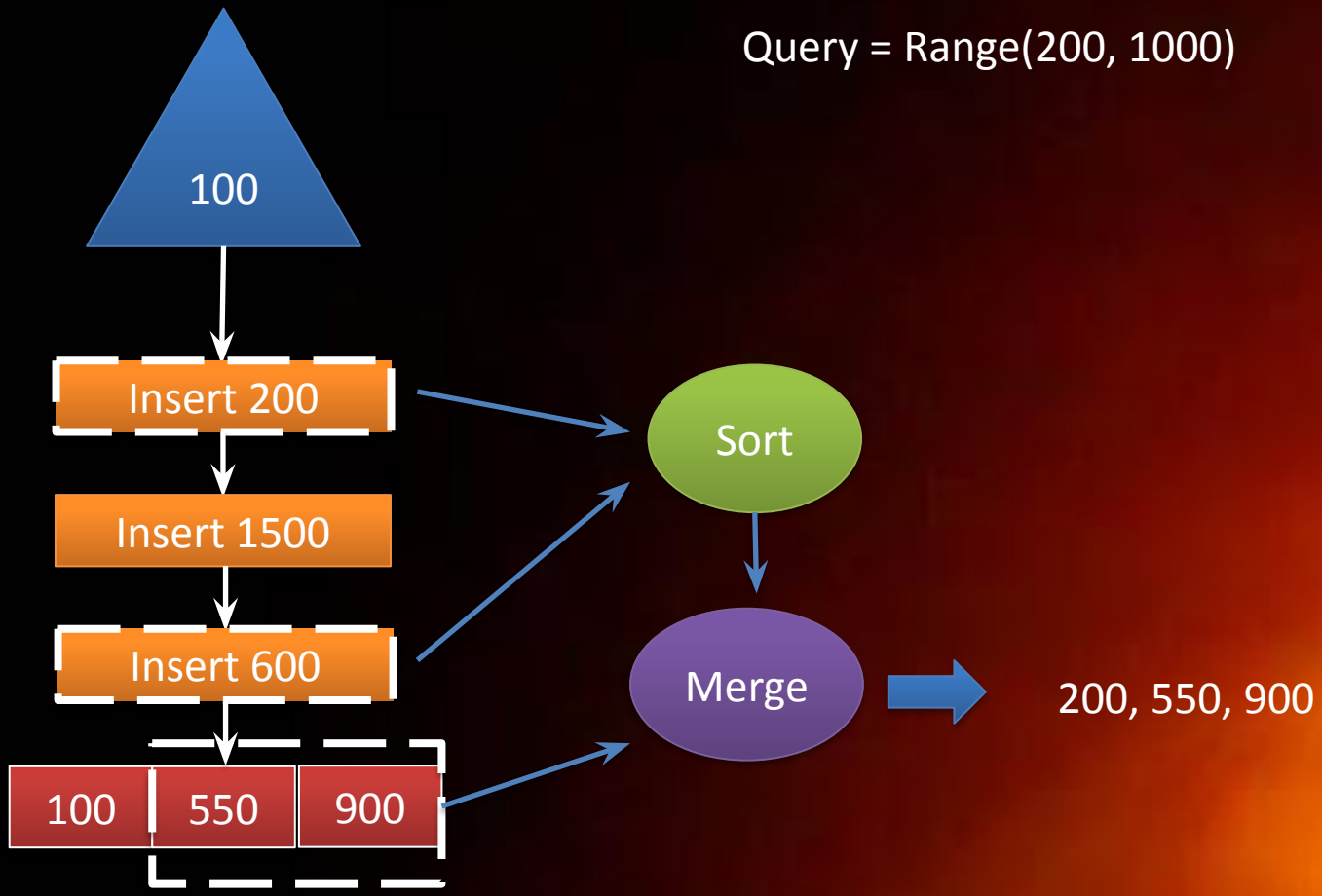
Range queries





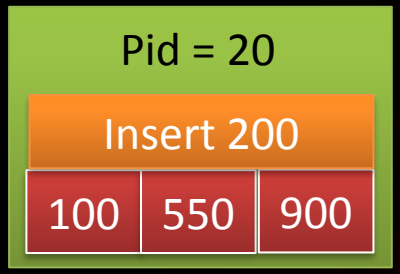
Query = Range(200, 1000)

Range queries

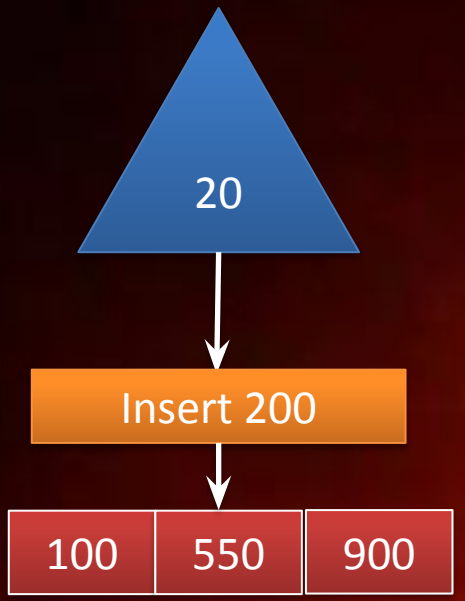


Persistence

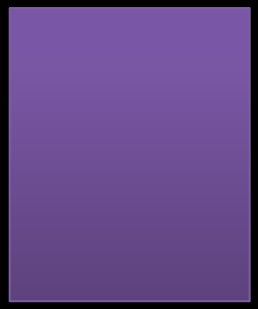
Marshal Page - >



Size = 300 bytes



Append-only log file

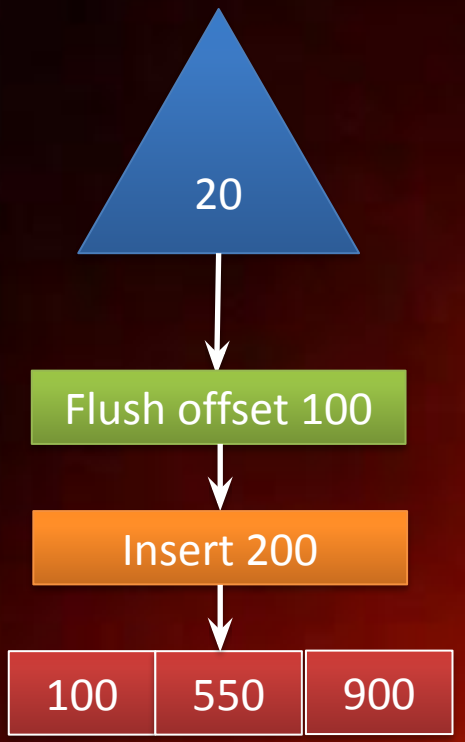


tailOffset = 100

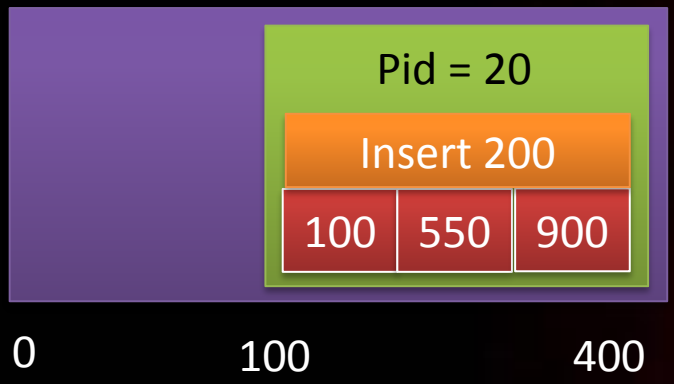
0

100

Persistence

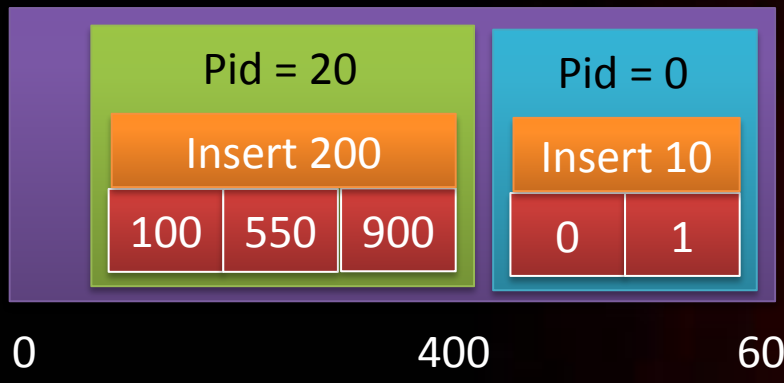


Append-only log file

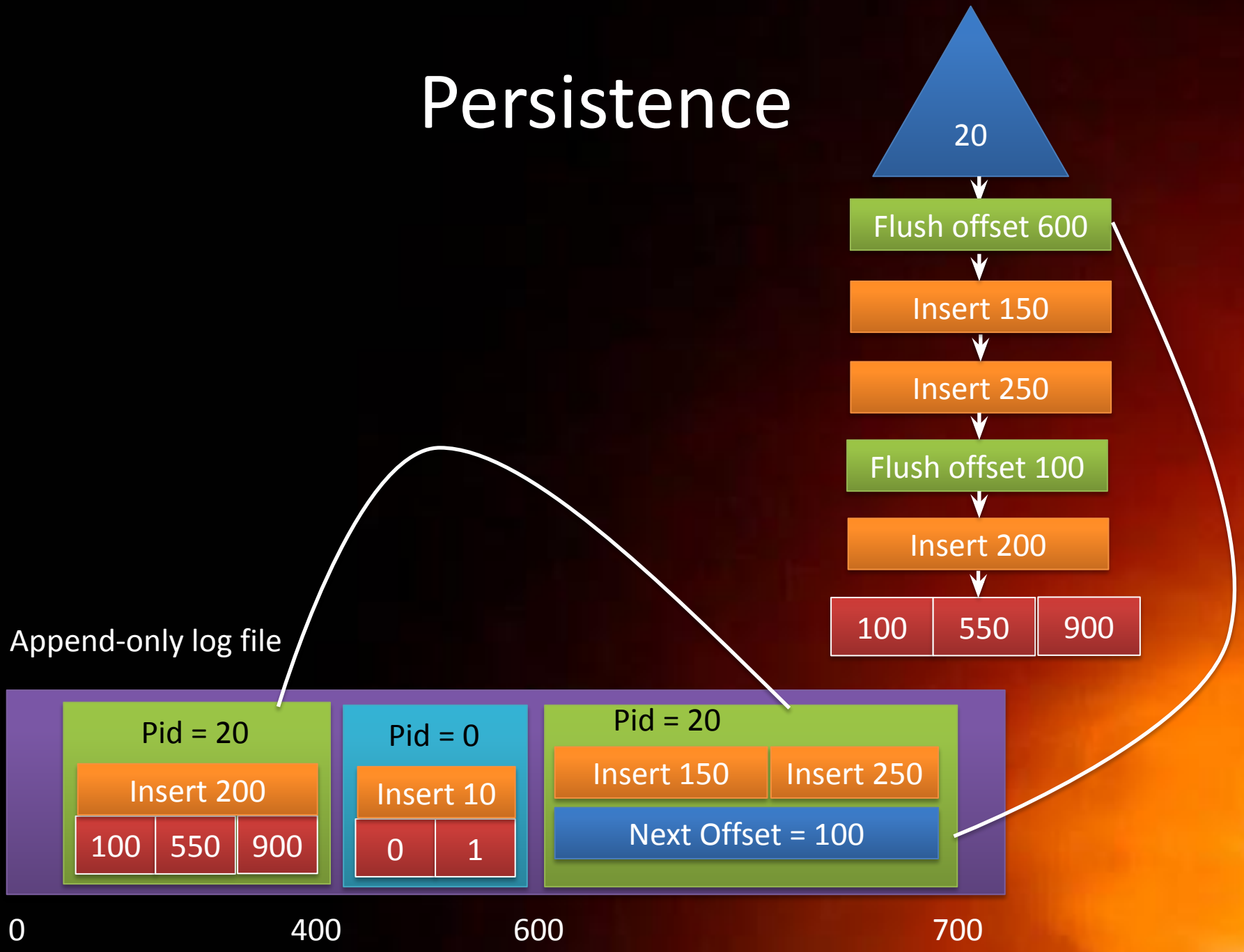


Persistence

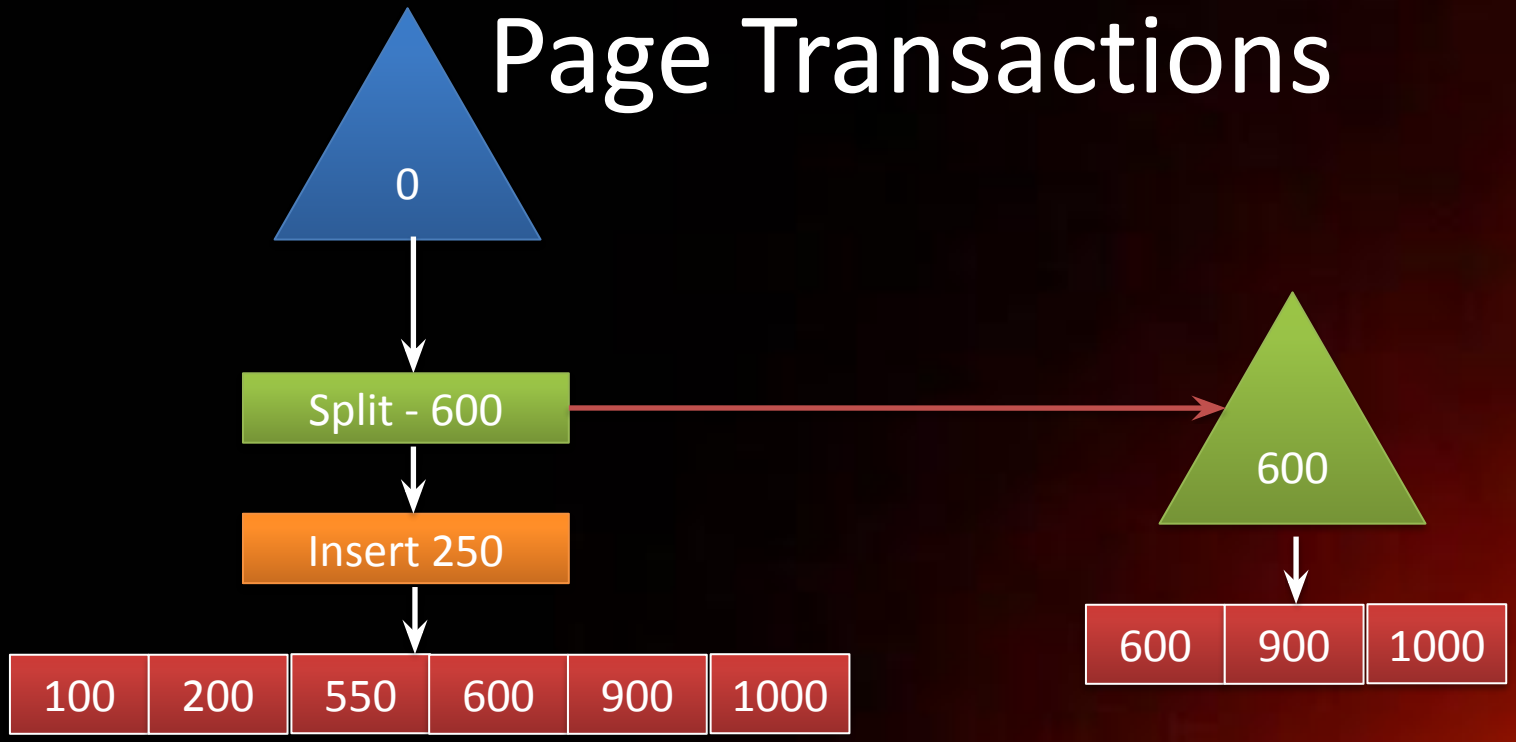
Append-only log file



Persistence

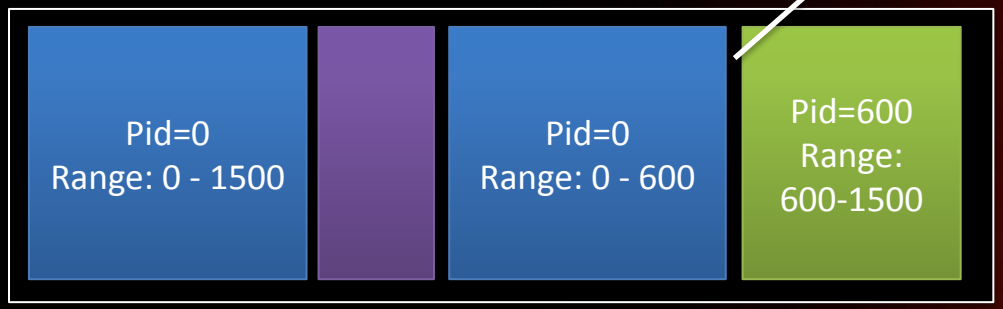


Page Transactions



Crash!

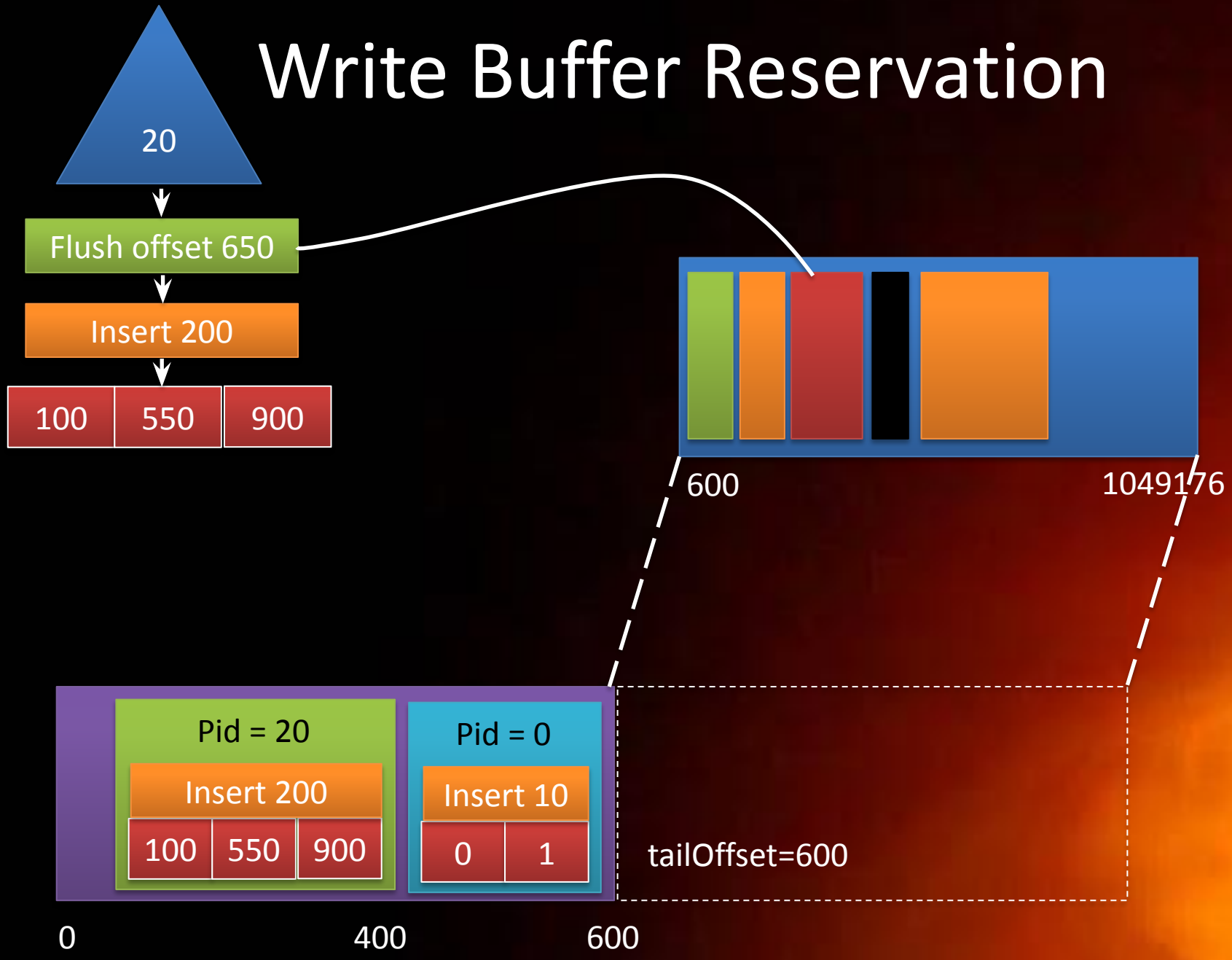
Log



Concurrent Log Writes

- A lockfree write buffer of size 1MB (SSD friendly)
- Concurrent threads can reserve allocations in the log at an offset
- Once all concurrent writers finish copying to the write buffer, the last writer calls pwrite() with 1MB chunk
- A thread can read from the write buffer as well (If a flush delta with an offset-not-yet on log exists in a page)
- Multiple writer buffers are used alternately to allow pipelining while pwrite() blocking

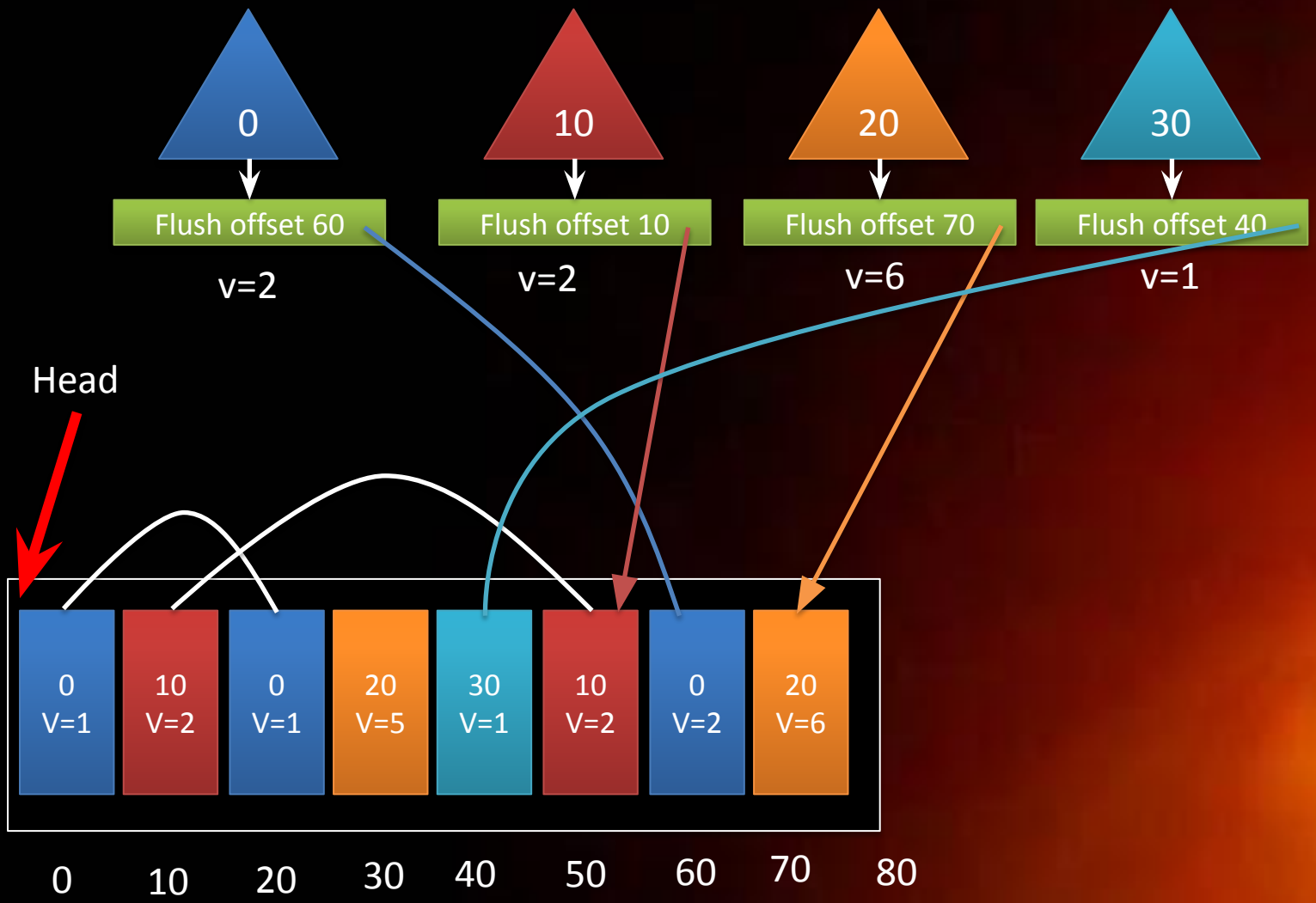
Write Buffer Reservation



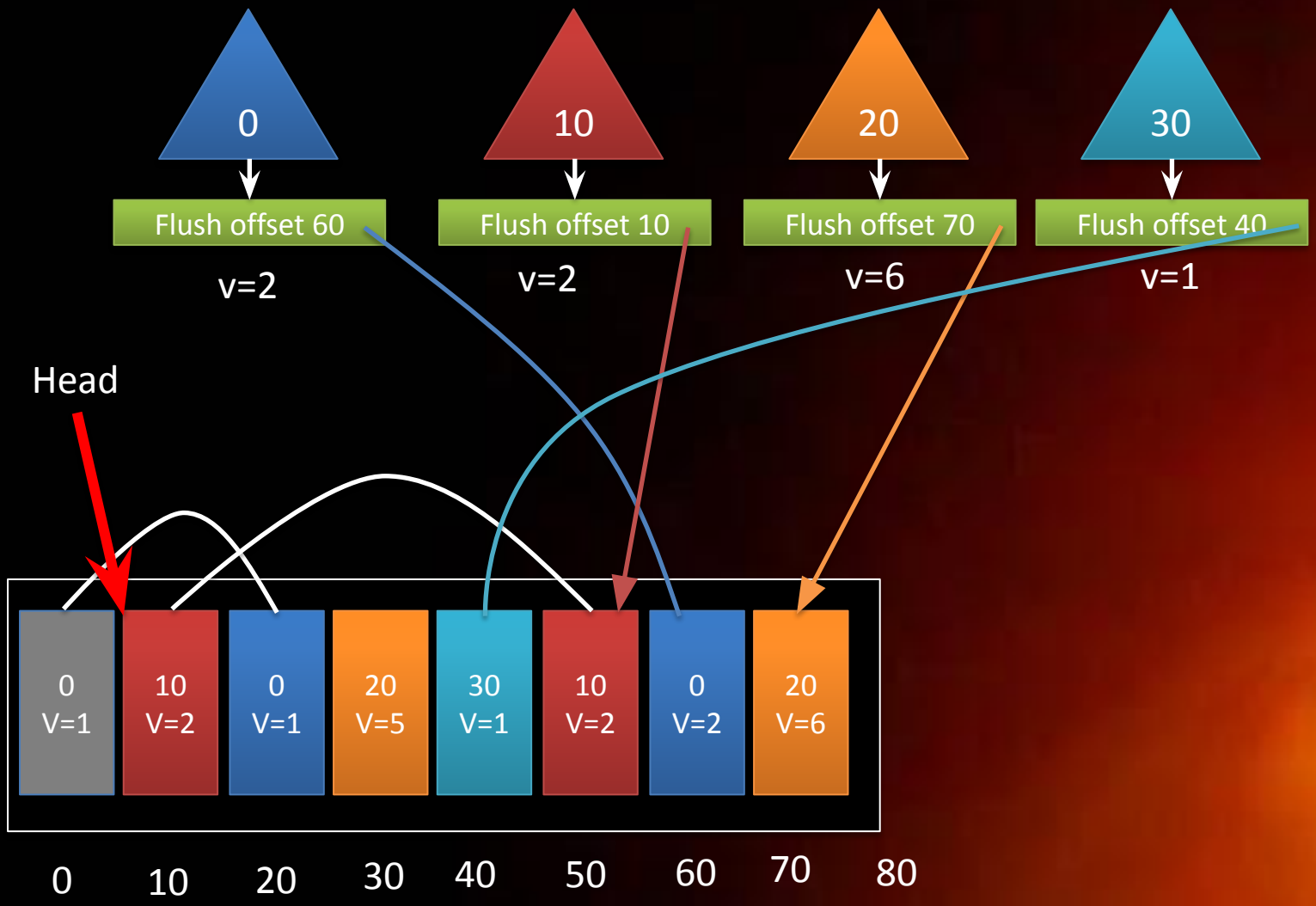
Log GC/Cleaning

- Log structured file system style log garbage collection
- Start from head of the log, relocate valid pages to the the end
- Trim the log at GC'ed offset
- The log is always growing at the tail
- Incremental log GC based on fragmentation threshold

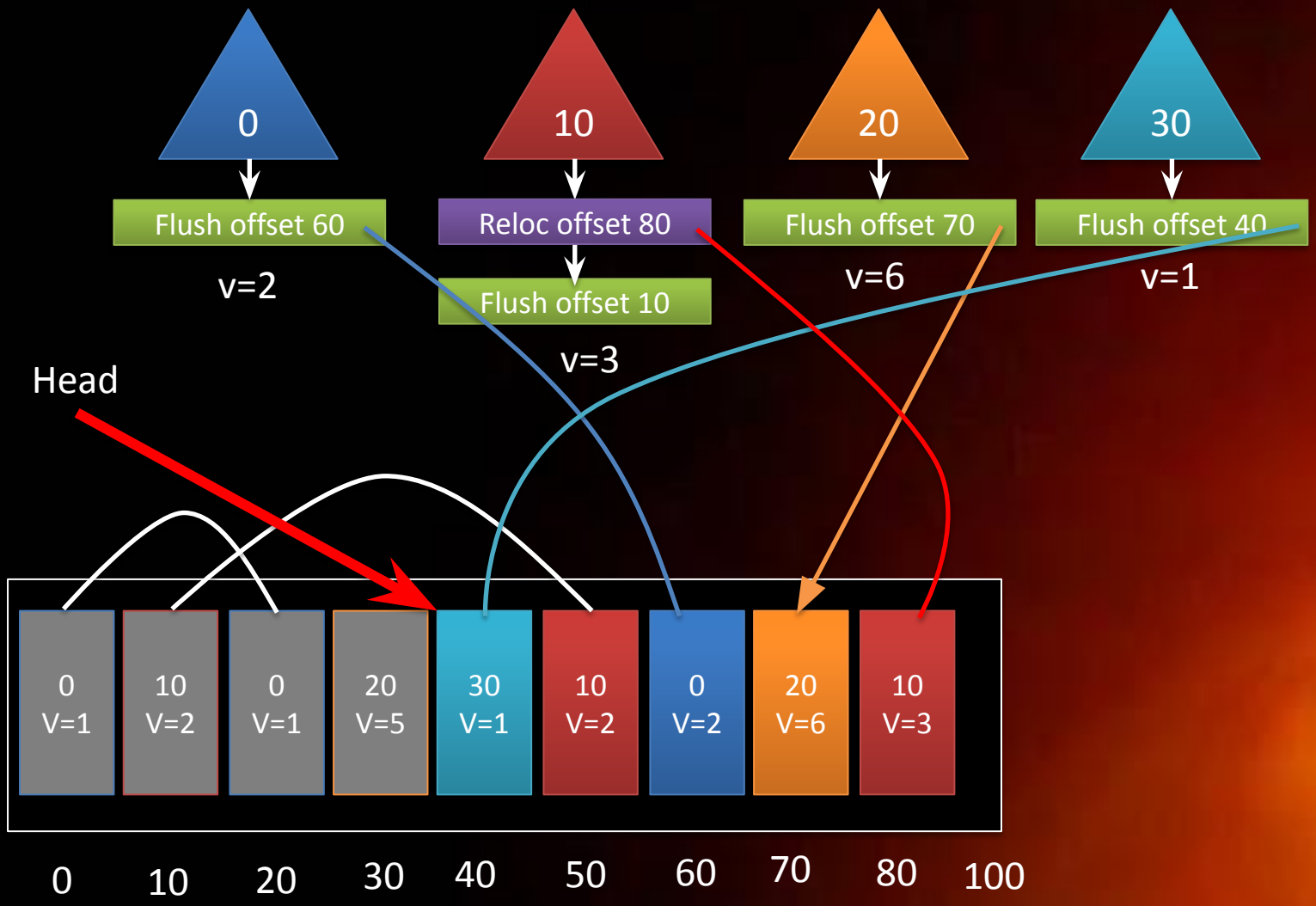
Log GC


0
10
20
30
Flush offset 60
Flush offset 10
Flush offset 70
Flush offset 40
v=2
v=2
v=6
v=1
Head
0 V=1
10 V=2
0 V=1
20 V=5
30 V=1
10 V=2
0 V=2
20 V=6
0
10
20
30
40
50
60
70
80

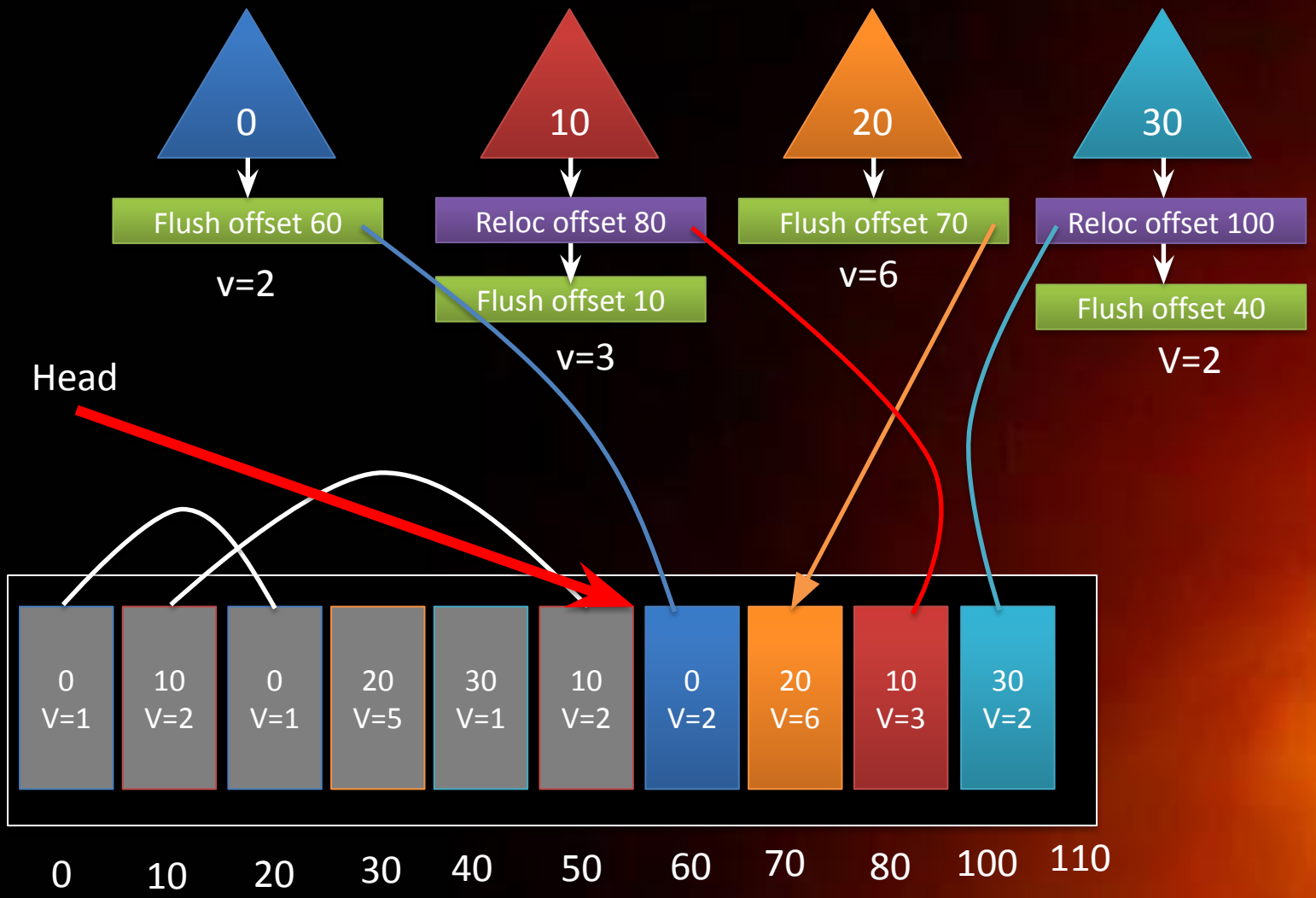
Log GC


0
10
20
30
Flush offset 60
Flush offset 10
Flush offset 70
Flush offset 40
v=2
v=2
v=6
v=1
Head
0 V=1
10 V=2
0 V=1
20 V=5
30 V=1
10 V=2
0 V=2
20 V=6
0
10
20
30
40
50
60
70
80

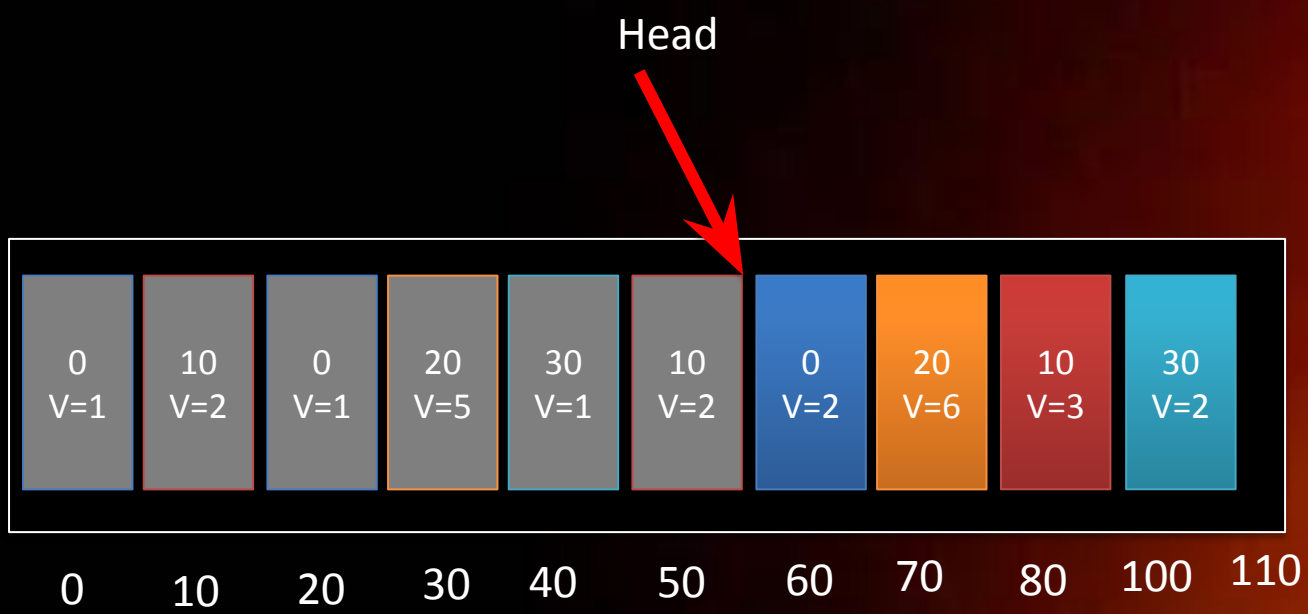
Log GC


0
10
20
30
Flush offset 60
Reloc offset 80
Flush offset 70
Flush offset 40
v=2
Flush offset 10
v=6
v=1
v=3
Head
0 V=1
10 V=2
0 V=1
20 V=5
30 V=1
10 V=2
0 V=2
20 V=6
10 V=3
0
10
20
30
40
50
60
70
80
100

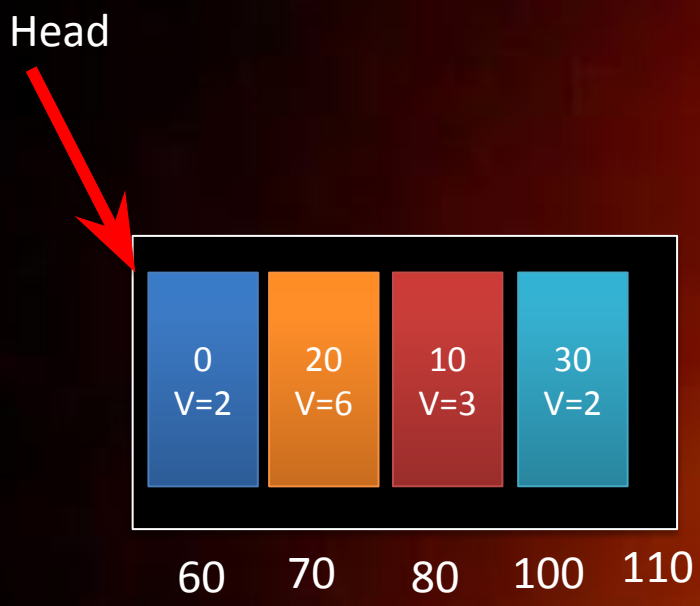
Log GC


0
10
20
30
Flush offset 60
Reloc offset 80
Flush offset 70
Reloc offset 100
v=2
Flush offset 10
v=6
Flush offset 40
v=3
V=2
Head
0 V=1
10 V=2
0 V=1
20 V=5
30 V=1
10 V=2
0 V=2
20 V=6
10 V=3
30 V=2
0
10
20
30
40
50
60
70
80
100
110

Log GC Trim


Head
0
V=1
10
V=2
0
V=1
20
V=5
30
V=1
10
V=2
0
V=2
20
V=6
10
V=3
30
V=2
0
10
20
30
40
50
60
70
80
100
110

Log GC Trim

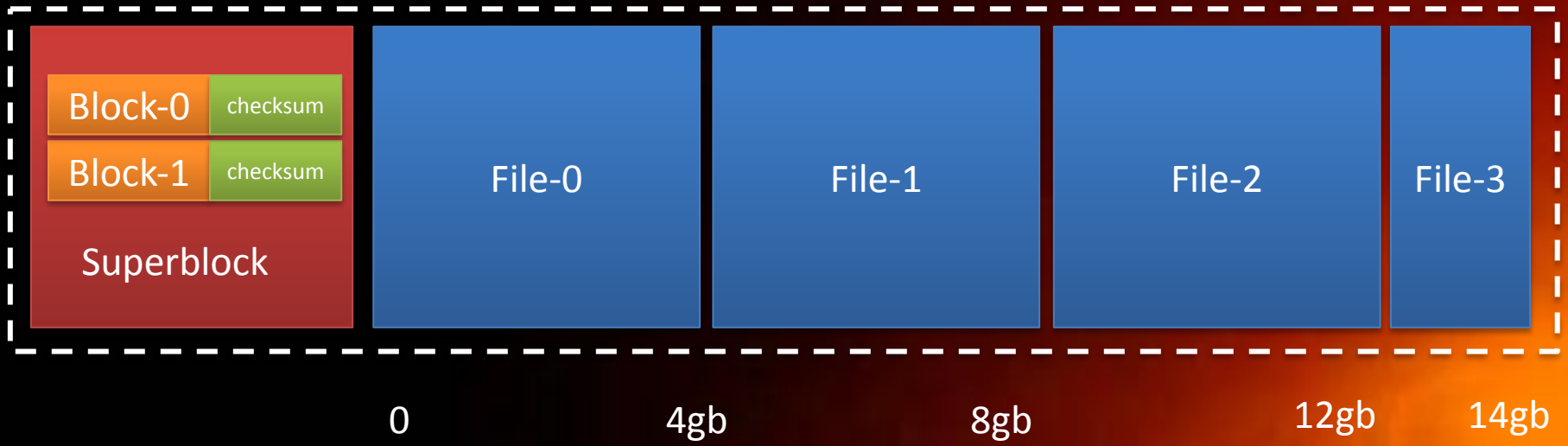


Log implementation

- Plasma Log is implemented as a collection of fixed size files of 4GB
- Each file chunk represents a range of log address of 4GB
- Log has a superblock file which tracks current logHeadOffset, logTailOffset
- Superblock has two 4KB blocks to store metadata with checksum
- If crash occurs during a write/fsync, log init will use the correct logHead, logTail offset by verifying the checksum for two blocks
- Log Trim command deletes a log file chunk when GCHead moves beyond 4 GB alignment
- The 64 bit offsets can live for 500 years (250 MB/s)

Log implementation

Trim operation can be implemented using file delete



For Linux, a Linux fs holepunching based single file log implementation (optional)

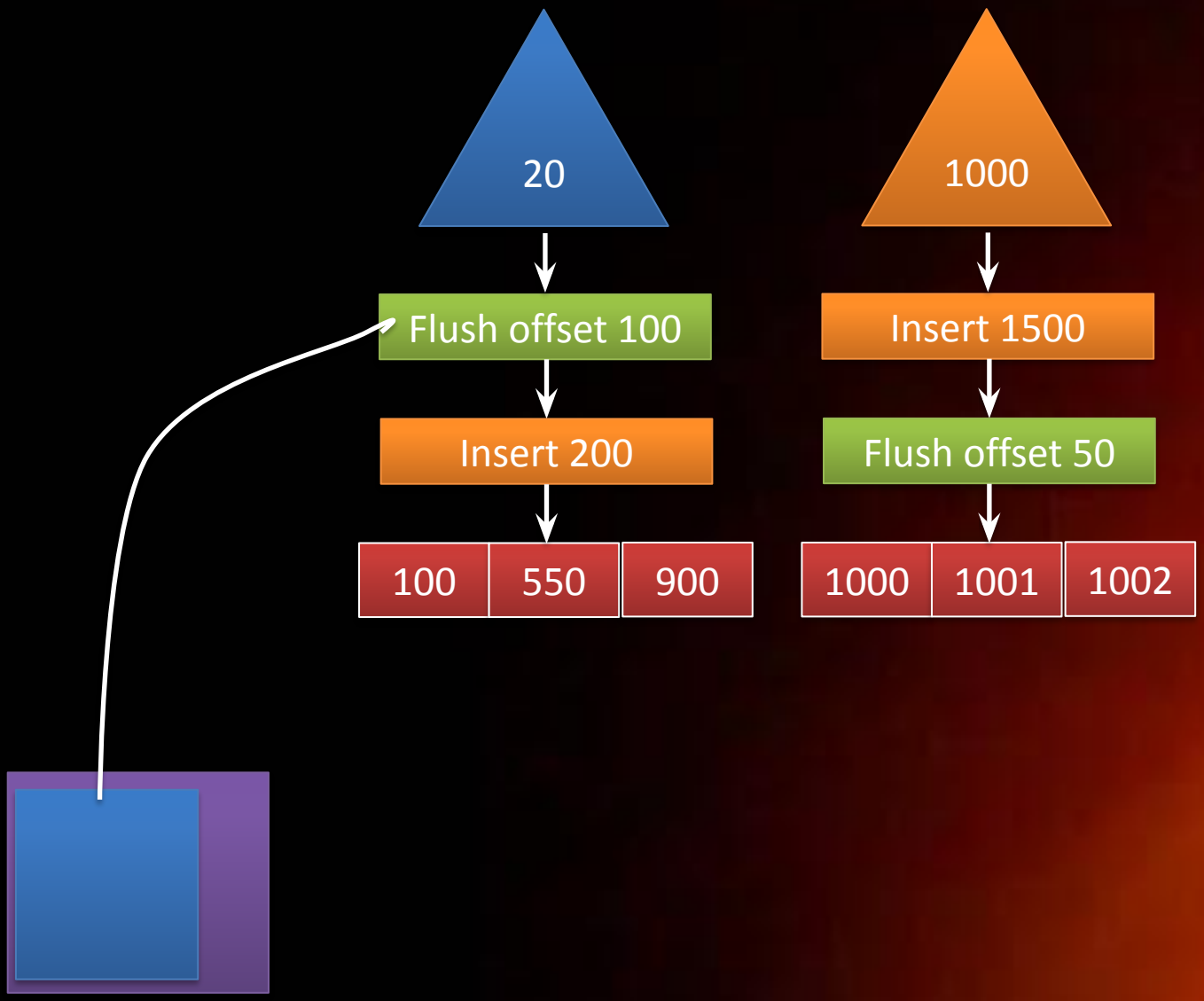
Lockfree file segment reclamation

- The GC will move headOffset after cleaning
- We cannot trim the log until all readers have moved off the files-to-be-deleted
- Reference counting with file locks will become single point of contention
- Pages can be partially on Flash or Memory. Hence we have to to-be-deleted files for DB access
- Instead of locking, we implement a lockfree algorithm (SFR – Safe File Reclamation) to track potential reader access to a file and remove them safely.
- Based on ideas from EPOCH based memory reclamation algorithm

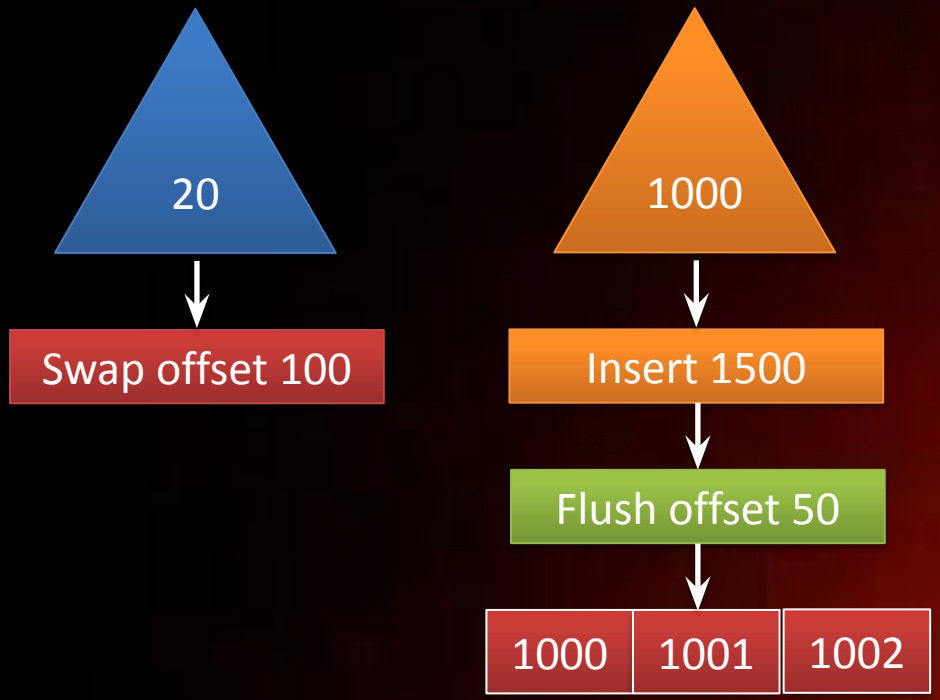
Page swapping

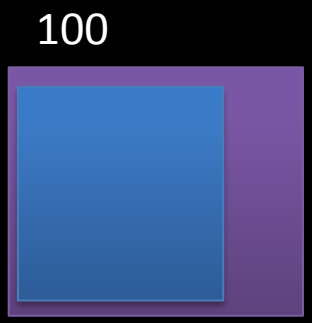
- When there is memory pressure, background swapper evicts pages from memory
- A read on evicted page will bring the page back into the cache
- An insert into an evicted page does not require a page swapin. Writes can can append insert/delete delta into the page similar to LSM trees

Page swapping

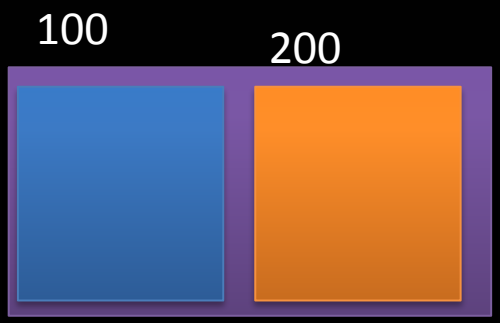


Page swapping

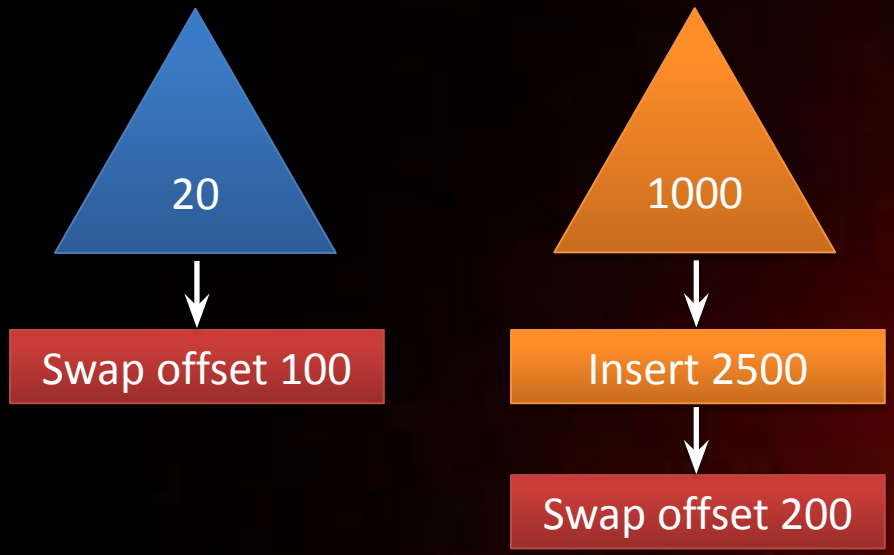


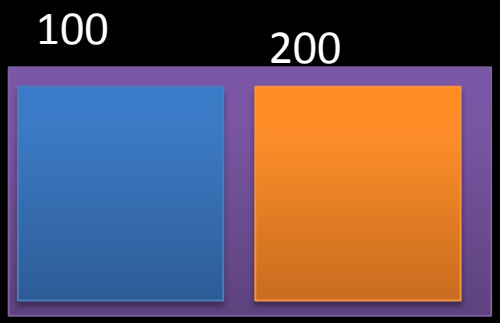


Page swapping

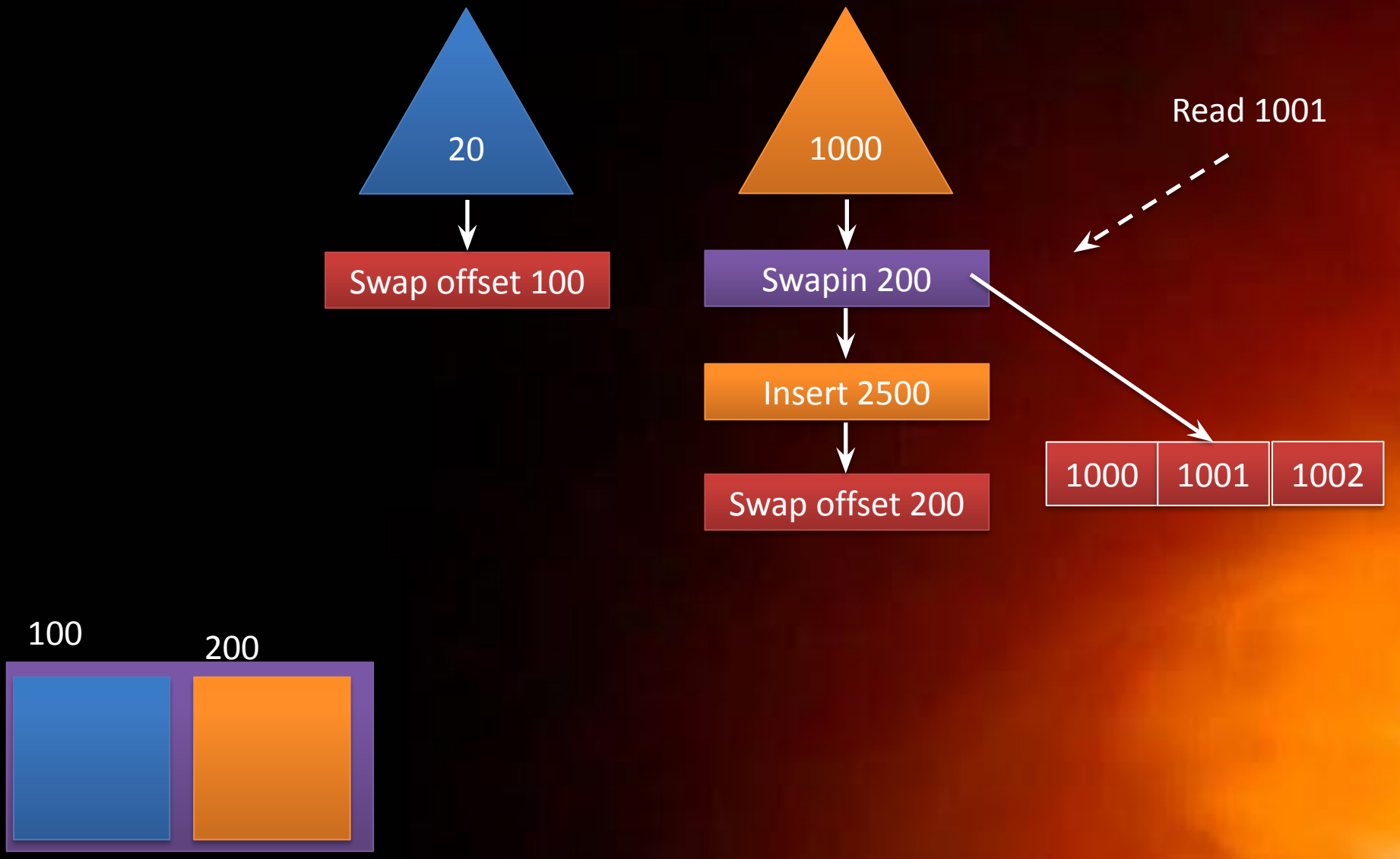


Page swapping





Page swapping



Cache Management

- Clock: Approximate LRU Algorithm
- A concurrent variant of Clock is implemented on top of lock-free skiplist index layer

Crash Recovery

- The index layer is rebuild by replaying the log from head to tail
- In future, Nitro style periodic backup on index layer can performed for facilitating instant recovery

Snapshots

- Nitro style MVCC is implemented to provide point-in-time snapshots
- Every Insert/Delete operation is tagged with 8 byte snapshot number
- Entries belonging to dead snapshots are lazily garbage collected as part of page compaction operation

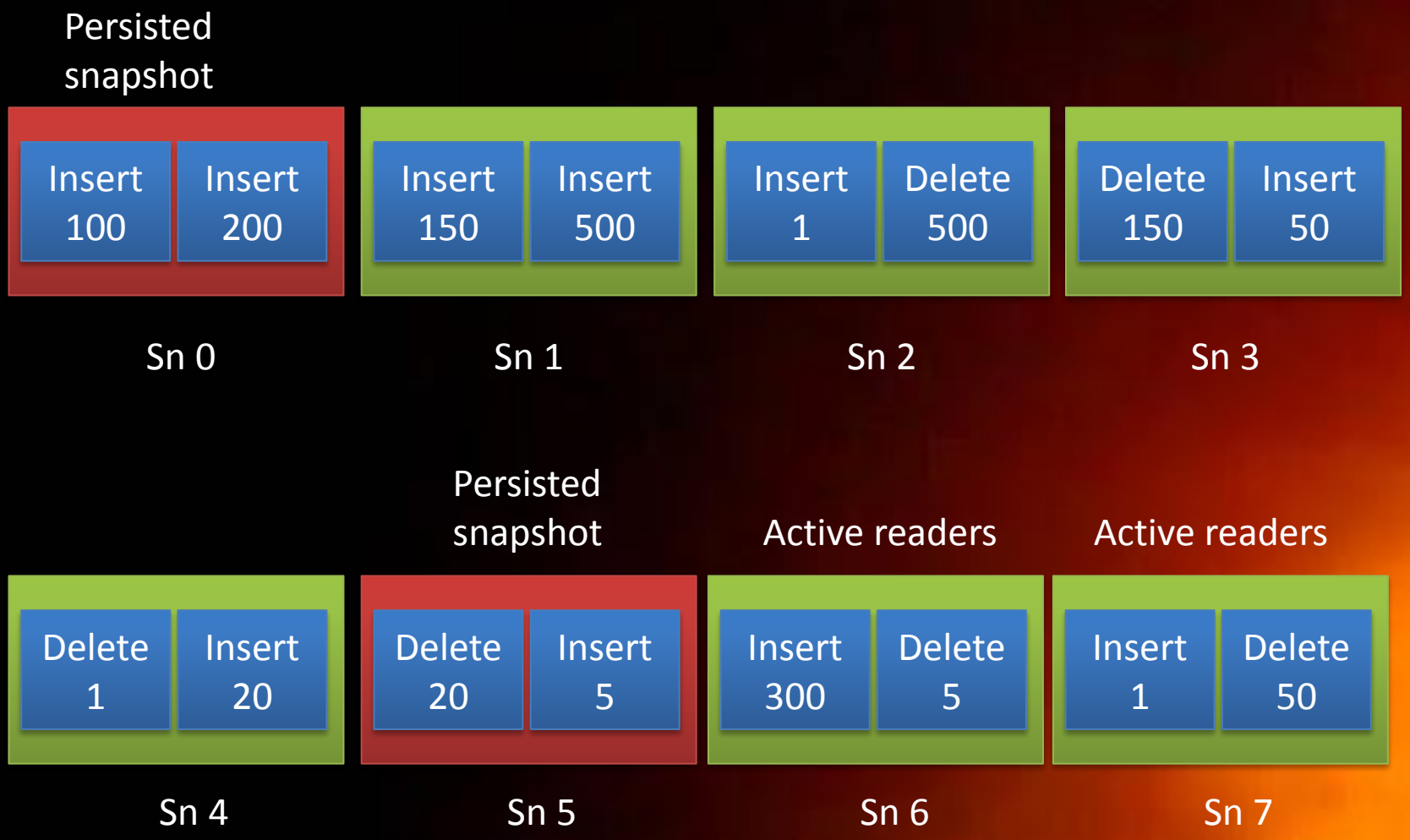
Persistent Snapshots/Rollback

- Plasma allow to create recovery points on a Plasma snapshot
- It is non-blocking an runs in the background
- GSI creates persistent snapshots every 10mins
- Two persisted snapshots are maintained to make sure that we can always rollback atleast 10 mins old data

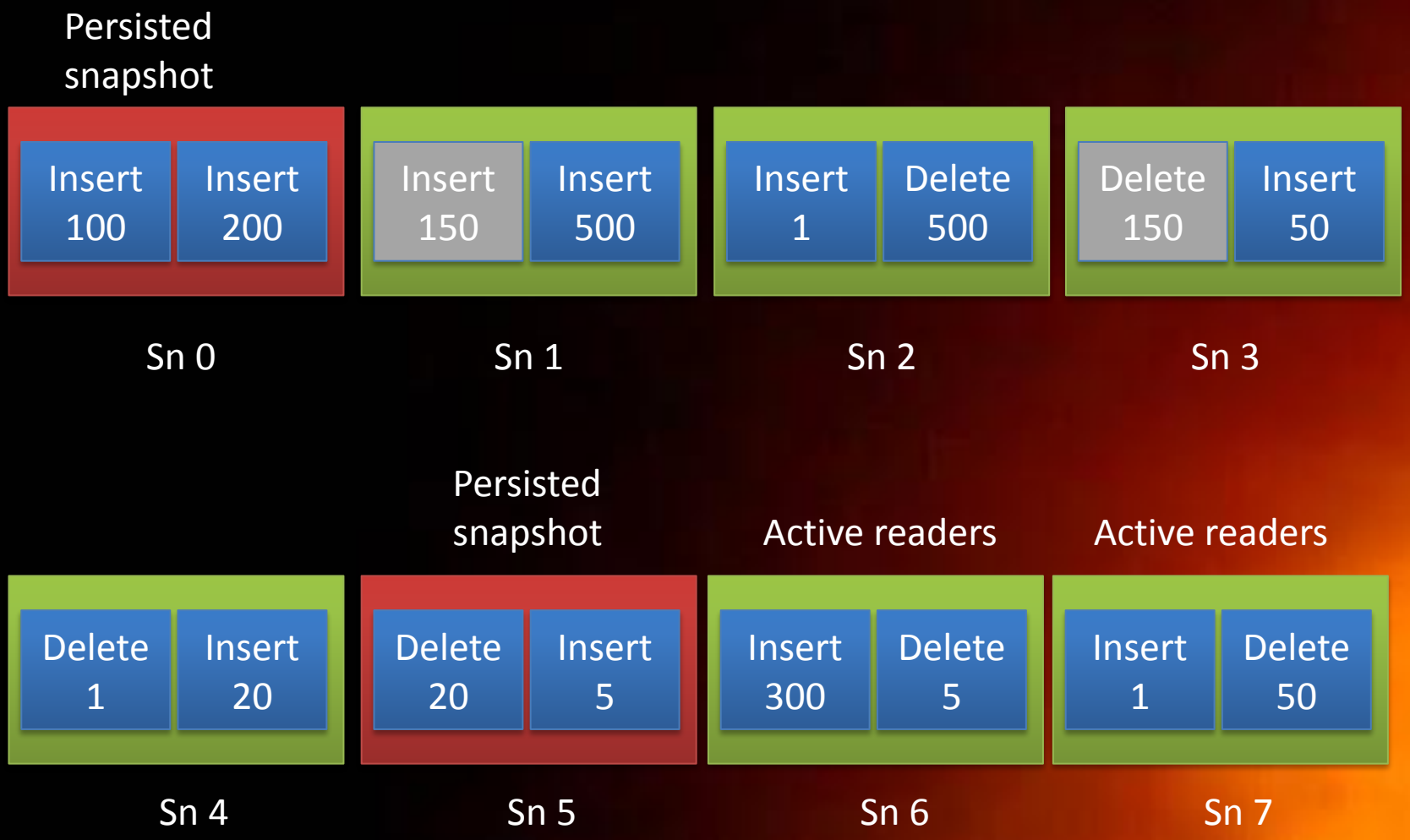
Overhead of Persistent Snapshot

- GSI creates in-memory snapshots every 20ms
- It results in 30000 snapshots every 10 mins
- If we lock a snapshot, future garbage MVCC garbage collections cannot be proceeded. ie., a entry in snapshot 0 may be deleted in snapshot 10000. If we remove the item, a read from snapshot 0 will miss entries and cause data loss
- For maintaining persistent snapshot, we have preserve correctness of snapshot 0

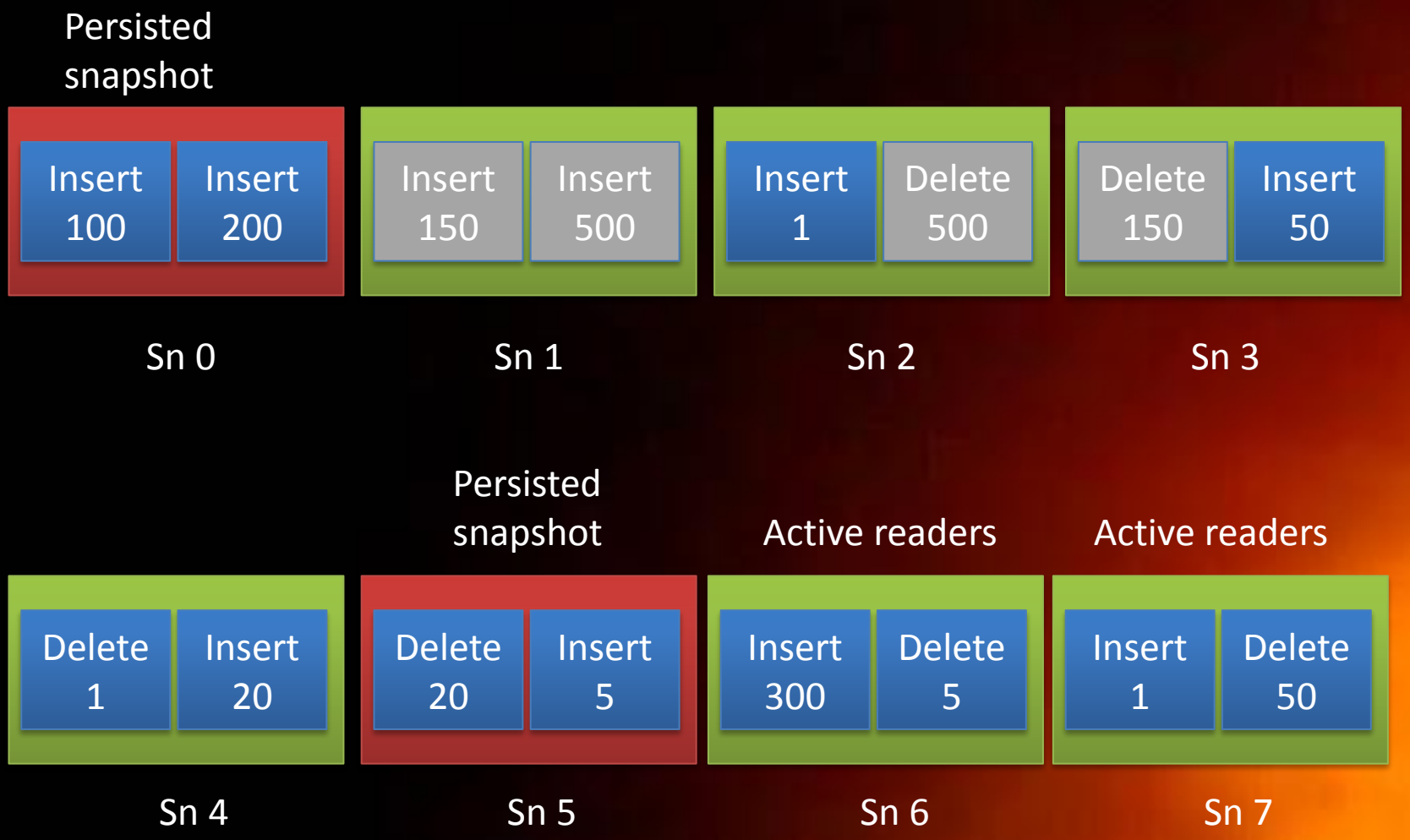
MVCC Garbage Collection



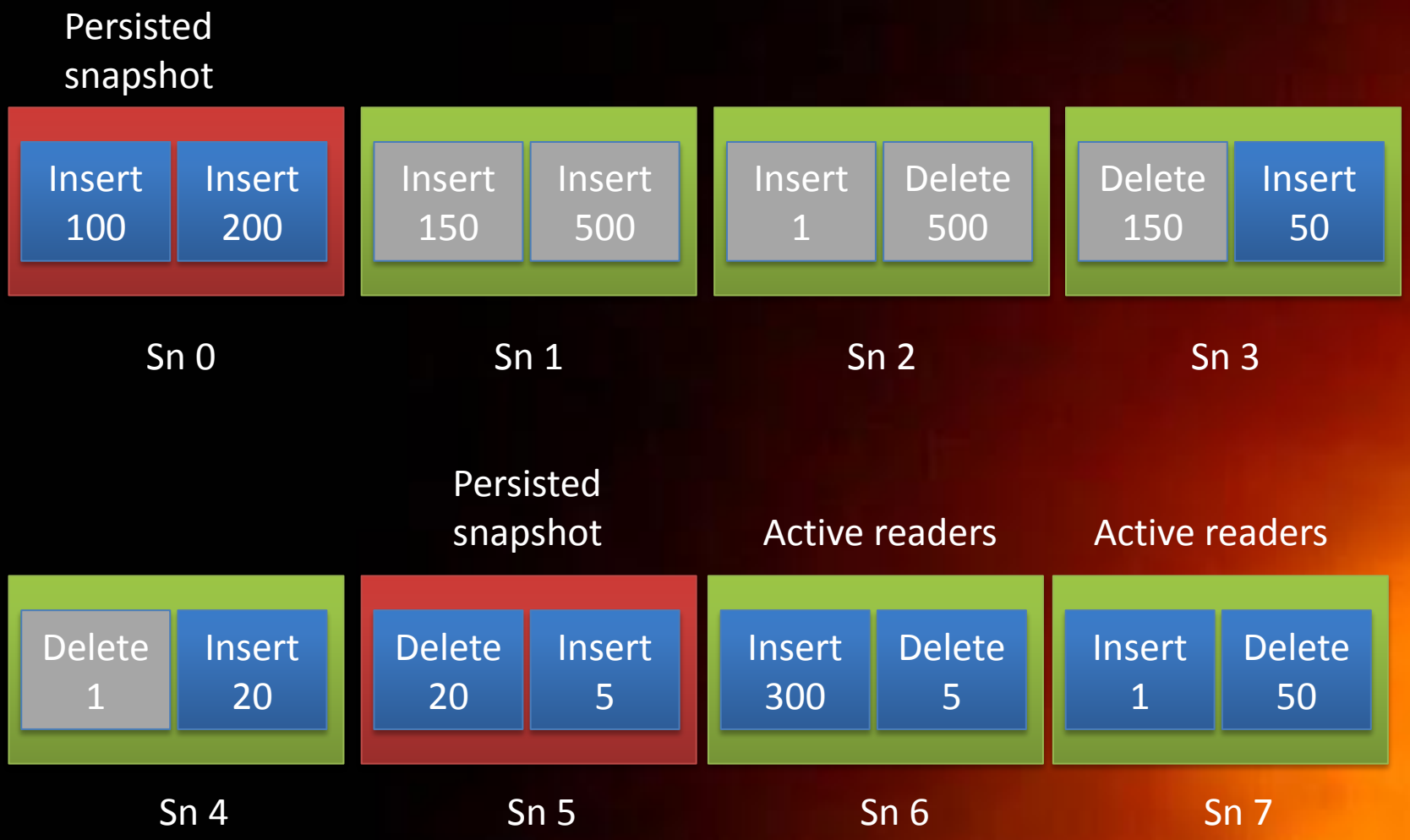
MVCC Garbage Collection



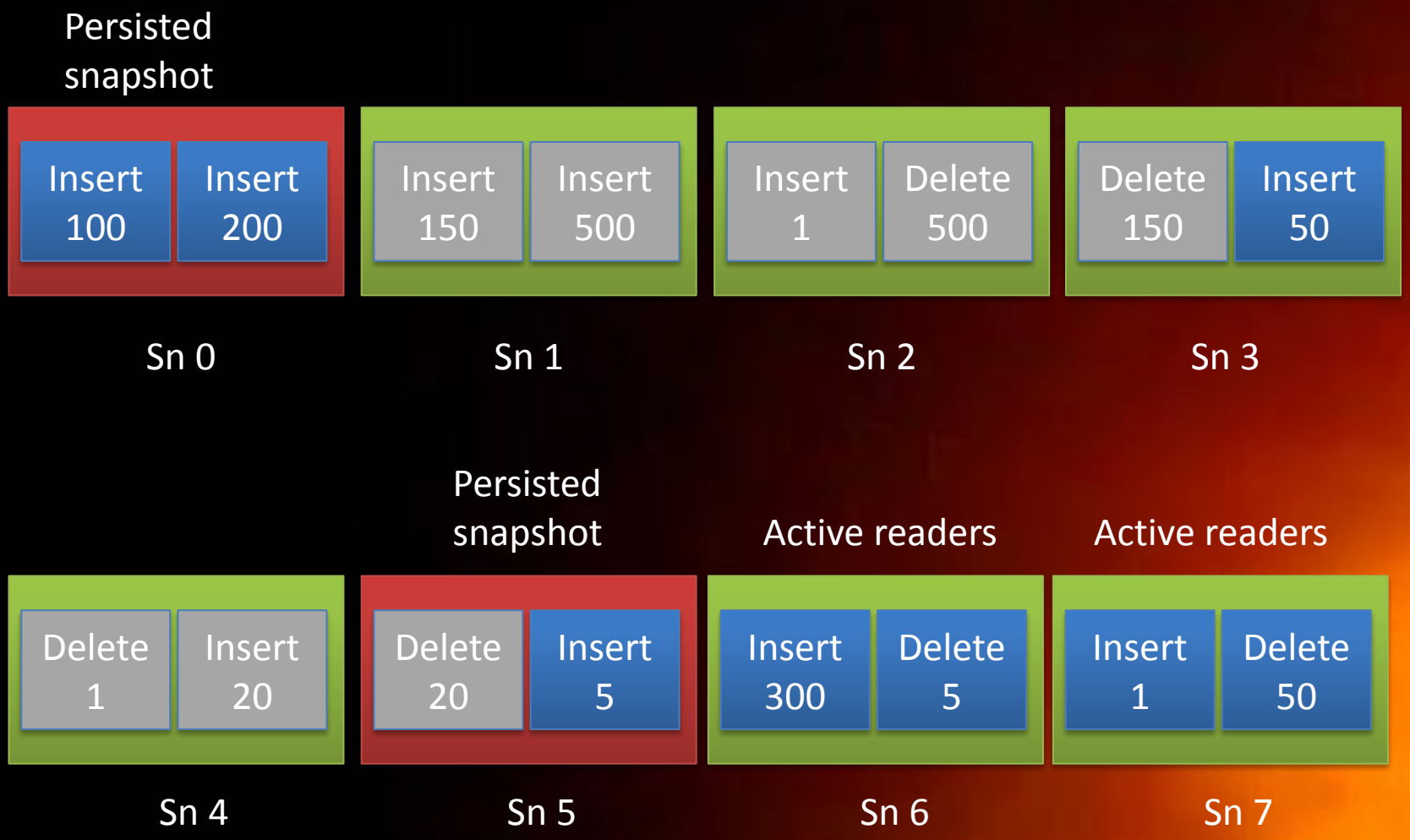
MVCC Garbage Collection



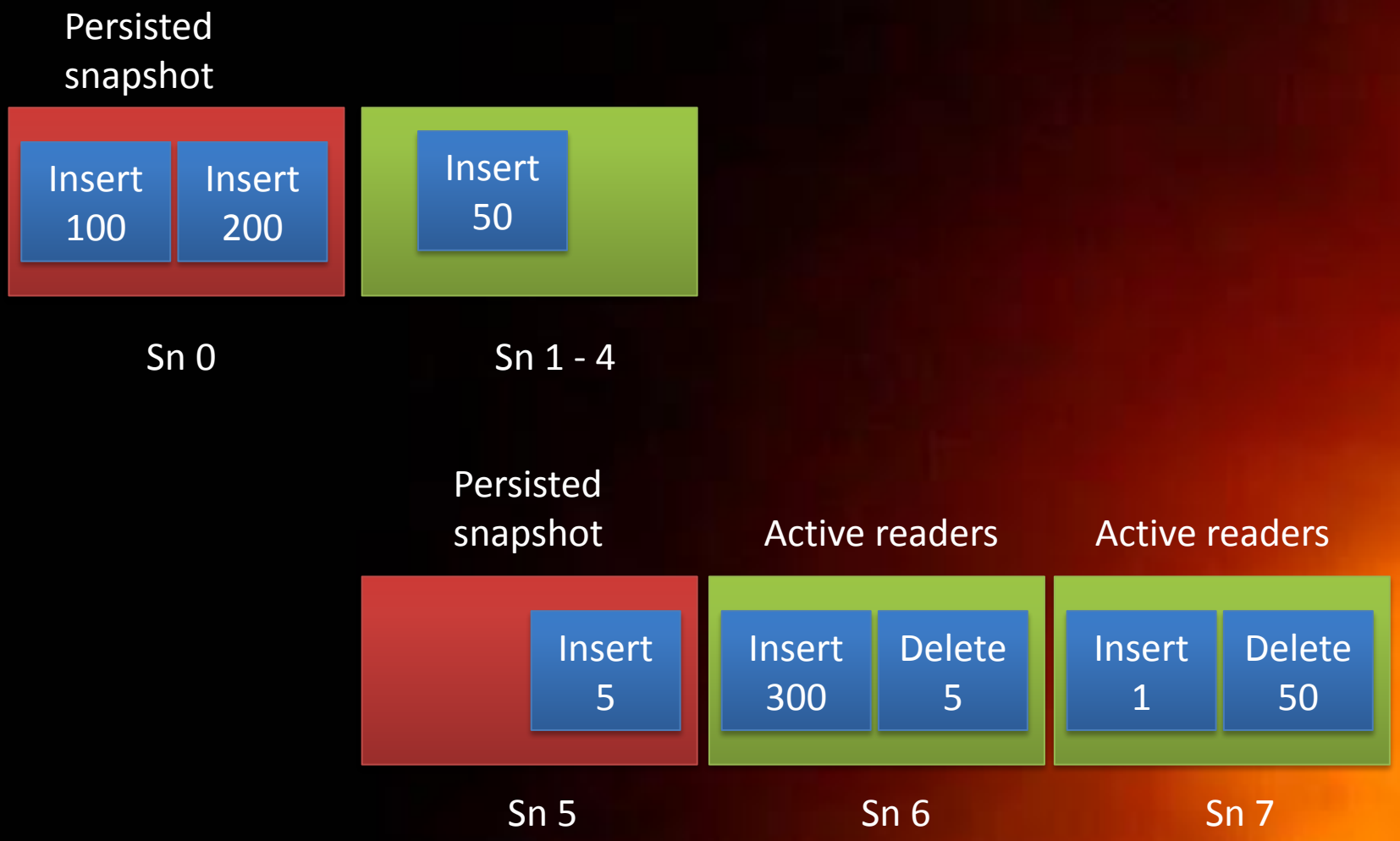
MVCC Garbage Collection



MVCC Garbage Collection



MVCC Garbage Collection

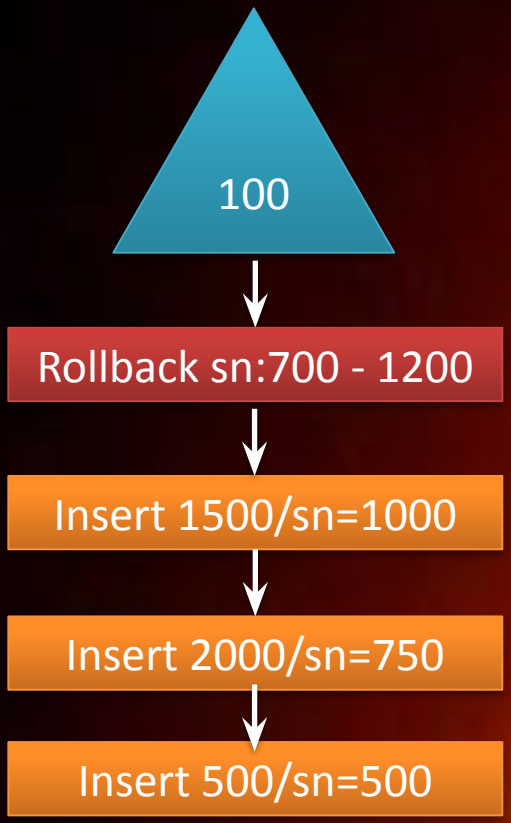


MVCC Garbage Collection

- We eliminate this problem by performing a special visibility aware MVCC garbage collection within these 30000 snapshots
- BigIdea: We know that snapshot 0 will be used in the future, but snapshot 1- 2910 are not going to be used and hence invisible snapshots. We garbage collect by merging the snapshots 1-2910 (Significant memory and storage savings)
- We keep atmost n+1 versions of a doc's entry (n = no of persisted snapshots)

Rollback

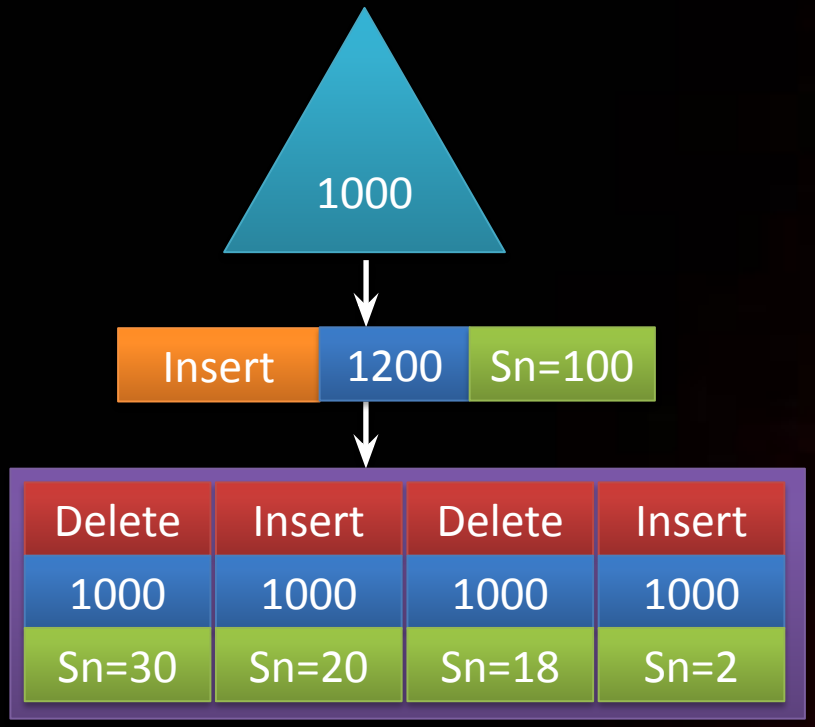
- Add rollback delta which invalidates future snapshots



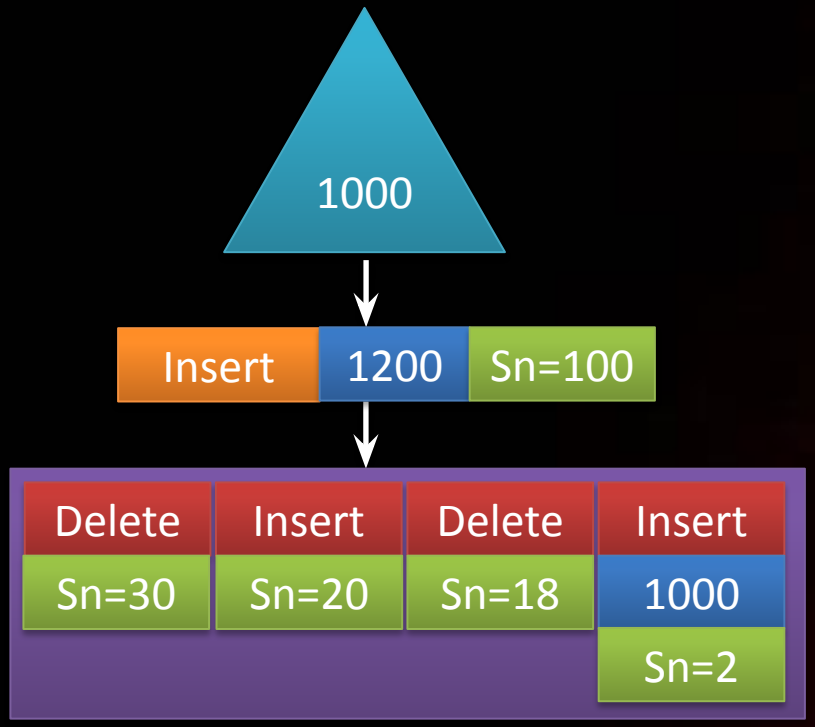
Memory Management

- Plasma integrates with Nitro Safe Memory Reclaimer
- JEMalloc is used for allocations

Key Compression



Key Compression

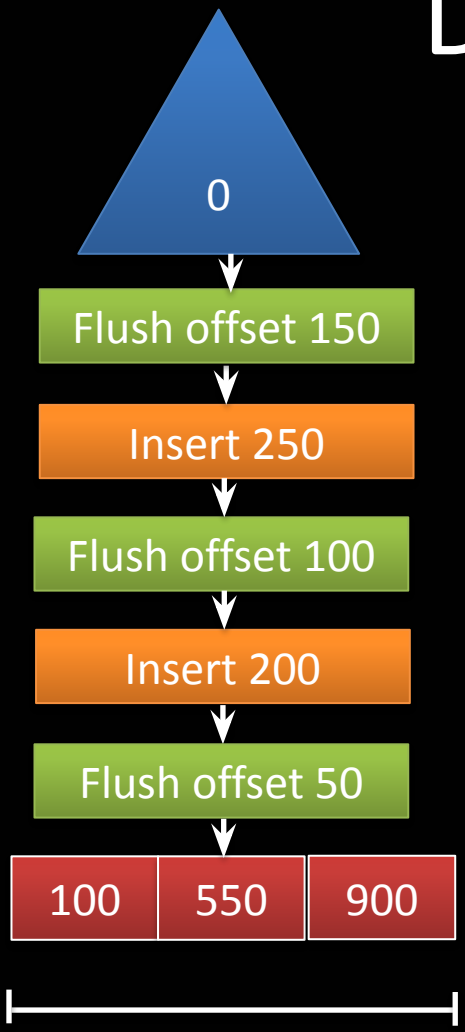


Same technique can be extended to compress Index entries with high cardinality emitted from different documents

[Bangalore, doc-100]
[Bangalore, doc-1]
[Bangalore, doc-200]

[doc-100]
[doc-1]
[Bangalore, doc-200]

Default configuration

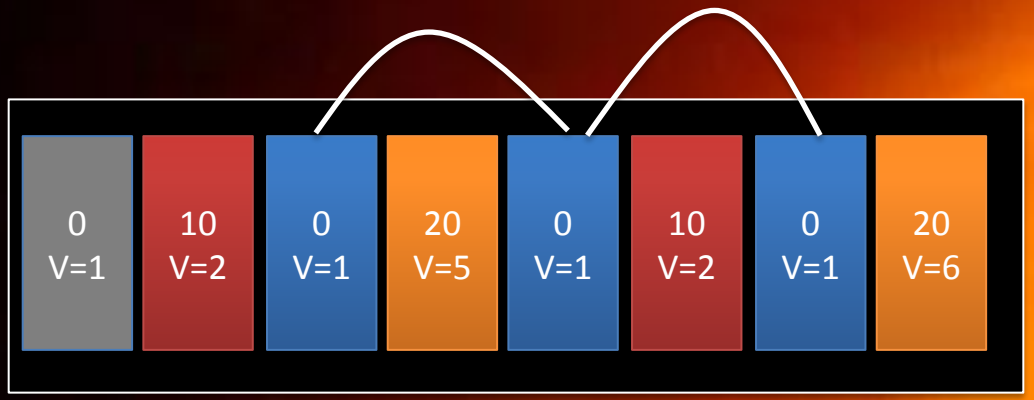


Compaction threshold = 200

Split threshold = 400

Merge threshold = 25

Max page segments



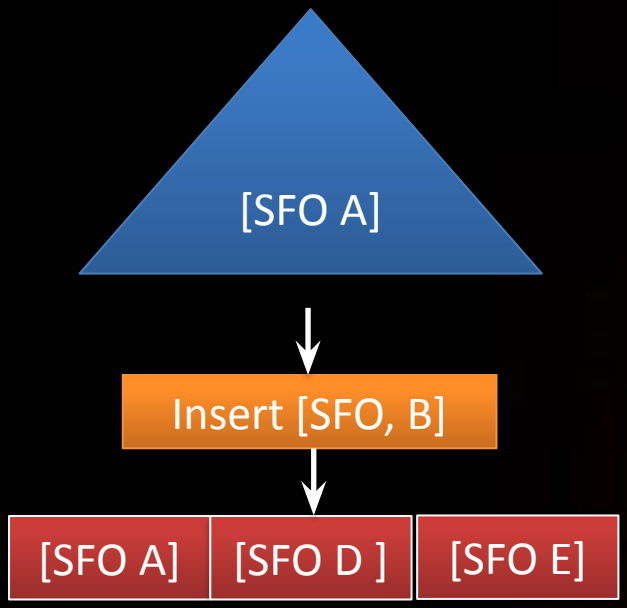
Highlights

- Concurrent Incremental persistence
  - Significantly reduces write amplification
- Avoids wandering tree problem
  - COW B+ tree is expensive
  - Plasma writes only affects data layer pages
- Incremental log garbage collection
  - Avoids writer vs compactor catch-up problem
  - Predictable write performance
- Allows write/read optimization tradeoffs
  - Tunable delta length, page split/merge thresholds and max page segments allows to tradeoff performance factors appropriately for cpu vs memory vs write amplification

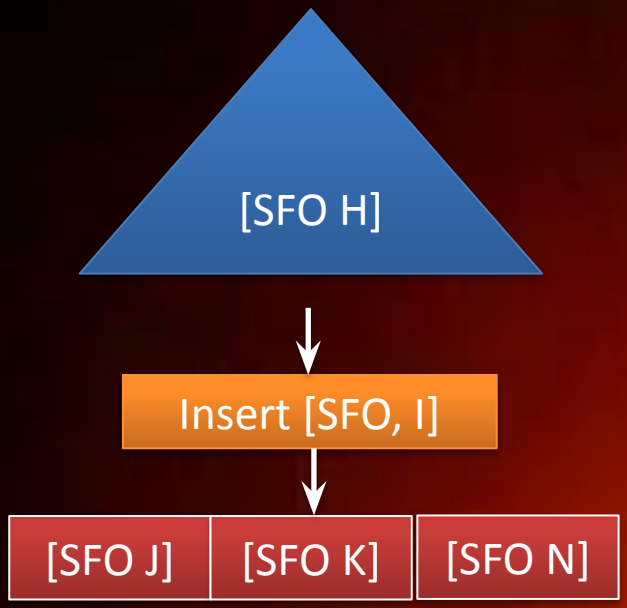
Future Enhancements

- Prefix compression
  - Each page has a fixed low and high key
  - This property allows Plasma to avoid duplicating common page key prefix across all the keys
  - This can significantly reduce storage requirements for Couchbase N1QL universal Index

Prefix compression



KeyRange: ([SFO A], [SFO H])



KeyRange: ([SFO H], [SFO P])

Prefix compression

KeyRange: ([SFO A], [SFO H])

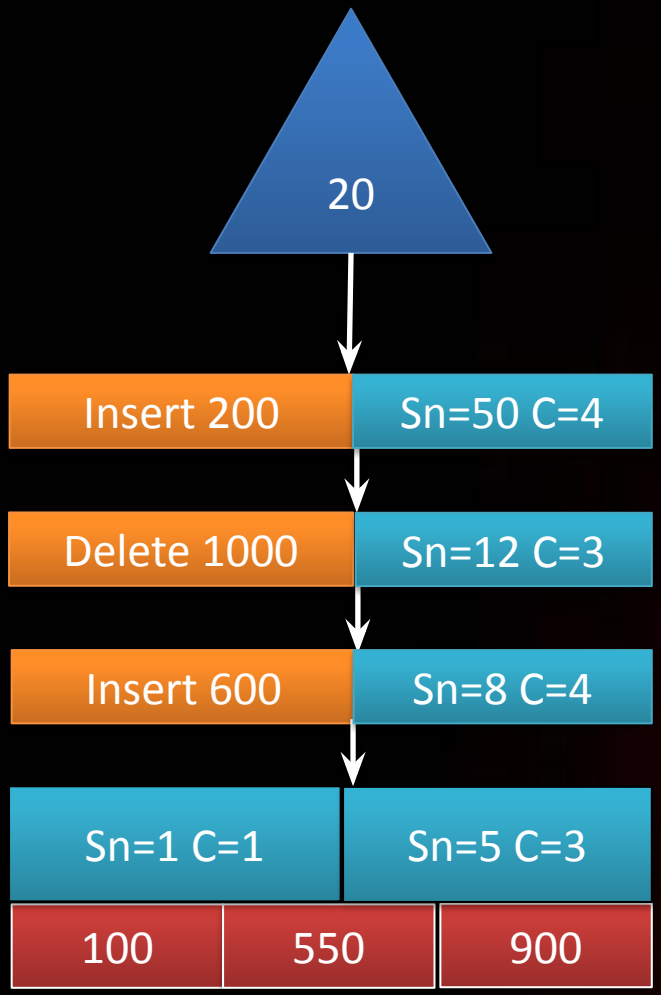
KeyRange: ([SFO H], [SFO P])

Compressed Pages

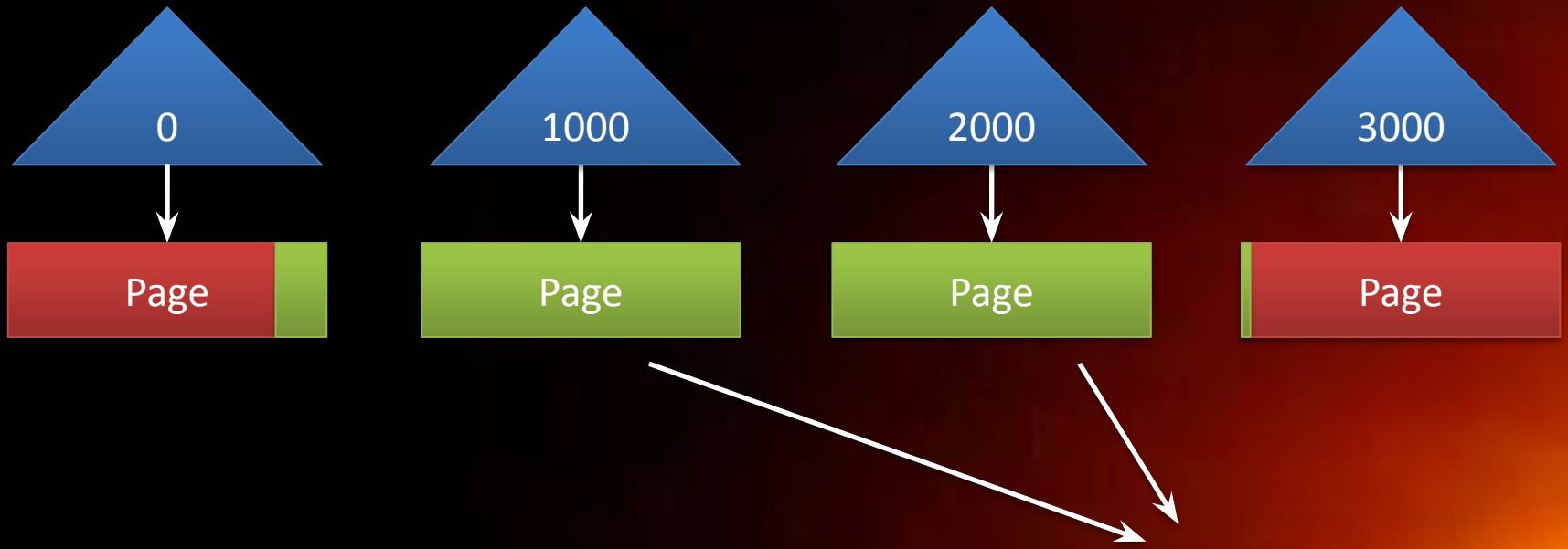
Future Enhancements

- Fast range count/Limited reductions
  - Every insert/delete into the page can track count for an MVCC snapshot with O(1) operation
  - Page compactions will be aware of the snapshot level counts for the page
  - A range count query on a large range can fully cover several pages. Count can be retrieved without reconstructing the ordering of items for fully covered pages
  - Using large page sizes can significantly speed up count

Count/Limited reductions



Count/Limited reductions



RangeCount(800, 3002)

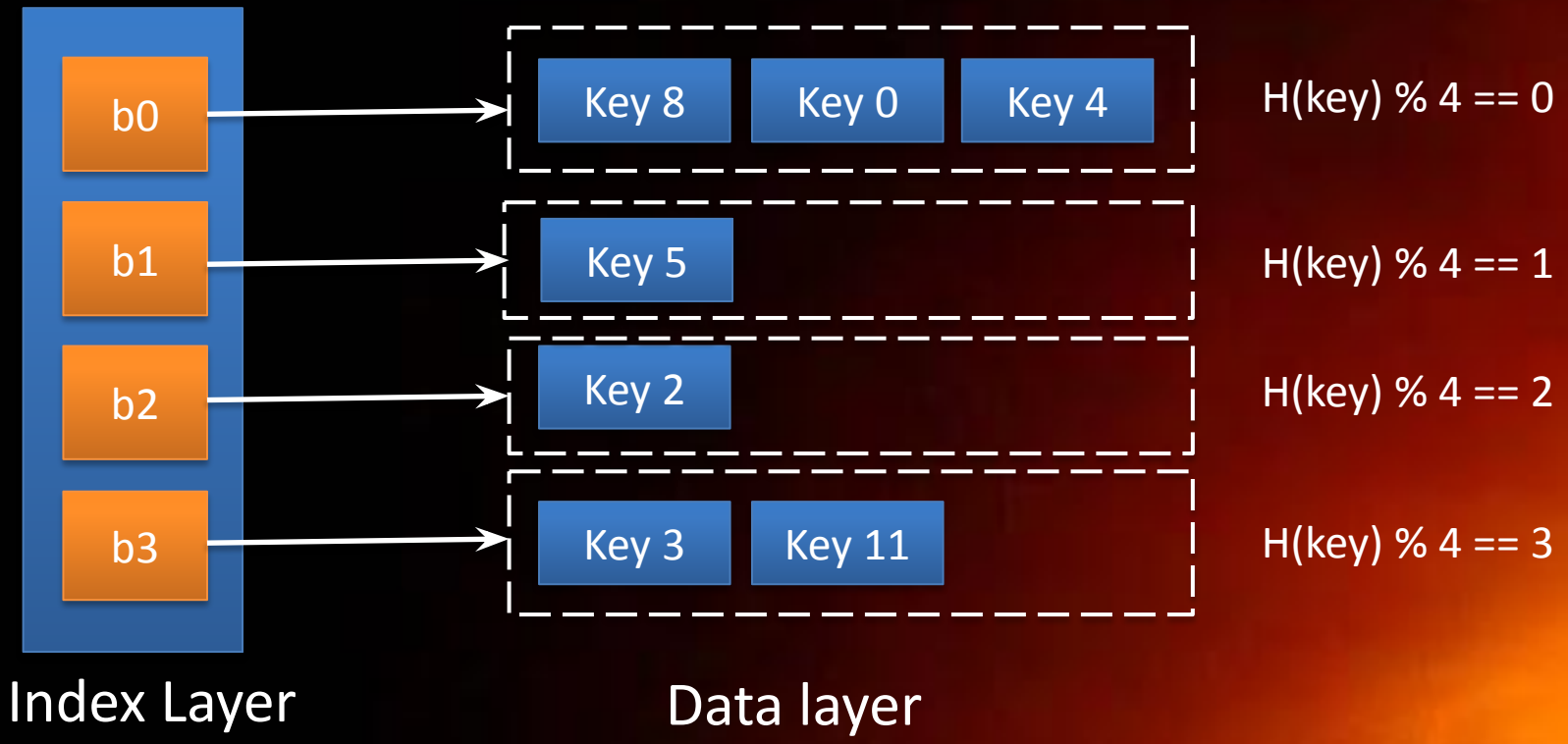
Future Enhancements

- Reverse Iterator
  - Skiplist Index layer has only forward pointer
  - The challenge is to efficiently iterate in reverse order without using any prev node linked list

Future Enhancements

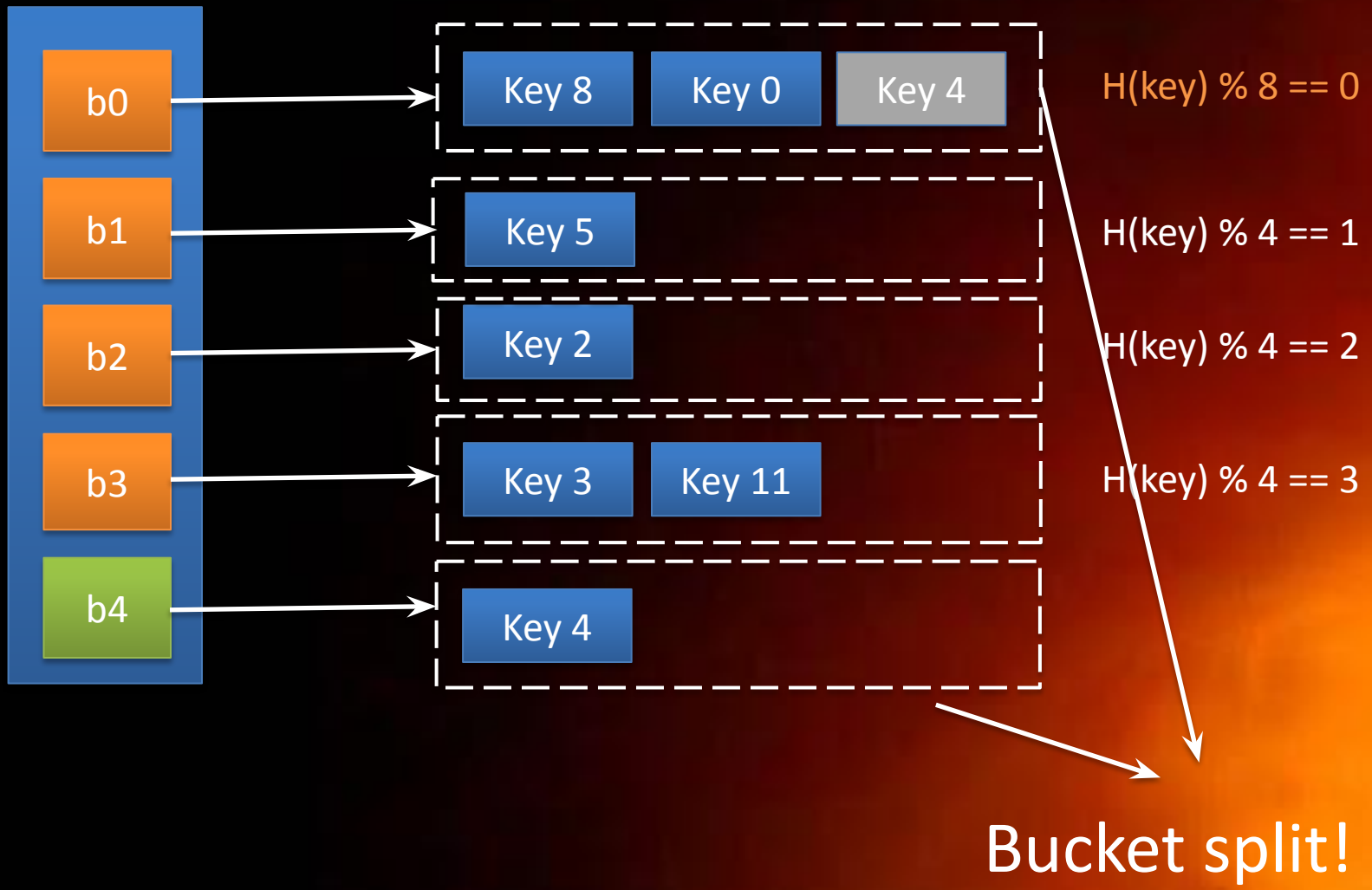
- A hash table Index implementation based on the same ideas
- Significantly improve GSI BackIndex performance for Plasma
- GSI Hash Index

HT Design

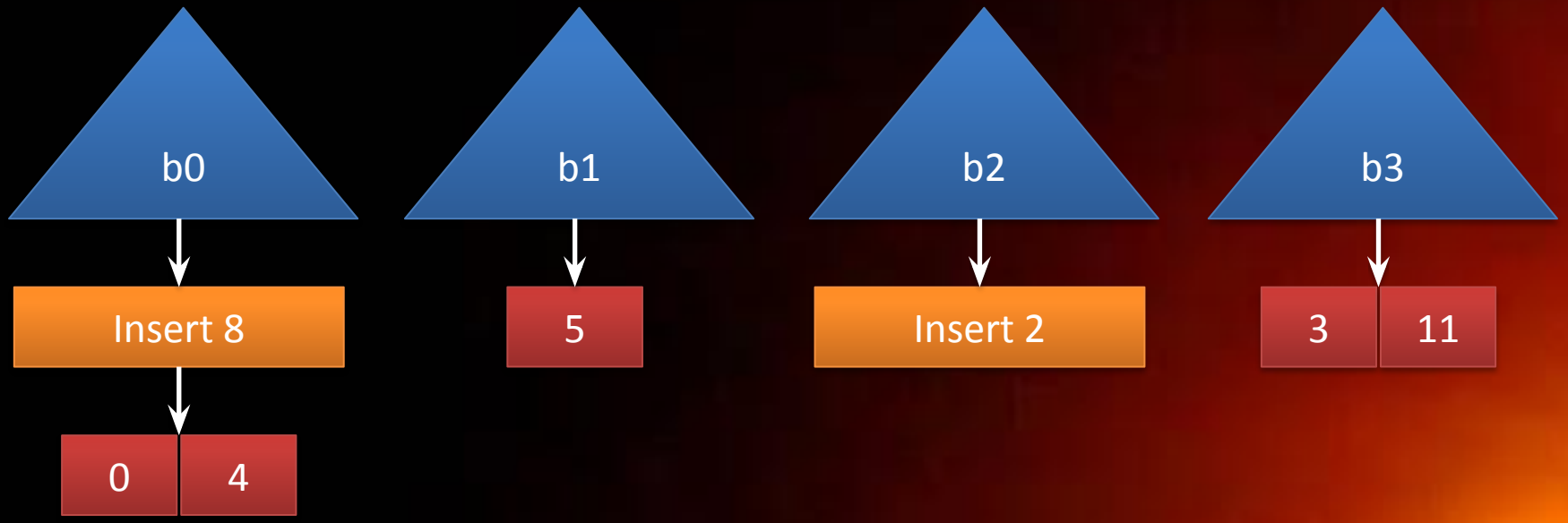


HashTable buckets

Linear Hashing



Plasma HashTable



h(key) % 4 == 0

h(key) % 4 == 1

h(key) % 4 == 2

h(key) % 4 == 3

Future Enhancements

- Non-volatile Memory (Intel 3DX point)
  - Plasma log can be implemented based on pmem.io APIs
  - Plasma configuration can be tuned to leverage high random read capability of pmem devices

Hot/Cold classification

- It is common to have majority idle documents in OLTP workload
- Classify index into two stores, Hot and Cold
- Allows to reduce write amplification in random workload

Conclusion

- Plasma presents a highly scalable KV storage engine
- Plasma allows various tunable tradeoffs for performance, cpu, memory and SSD write amplification
- Plasma features a memory first write optimized, read tunable design