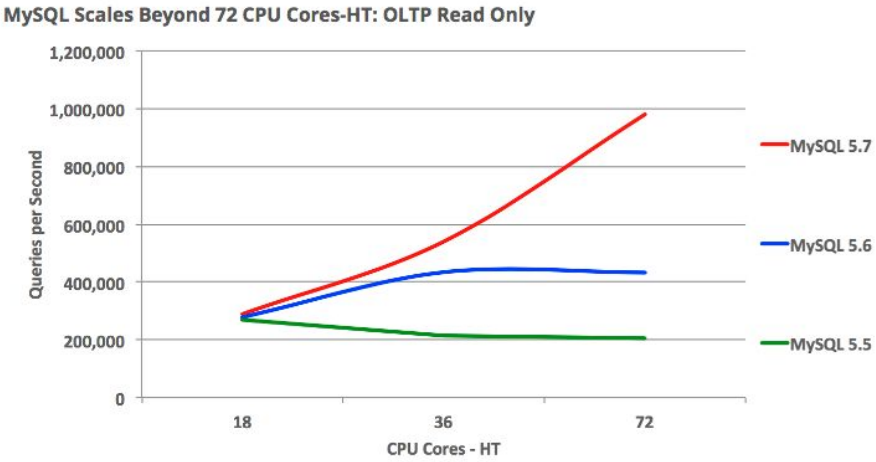
Nitro: A Fast, Scalable In-Memory Storage Engine for NoSQL Global Secondary Index

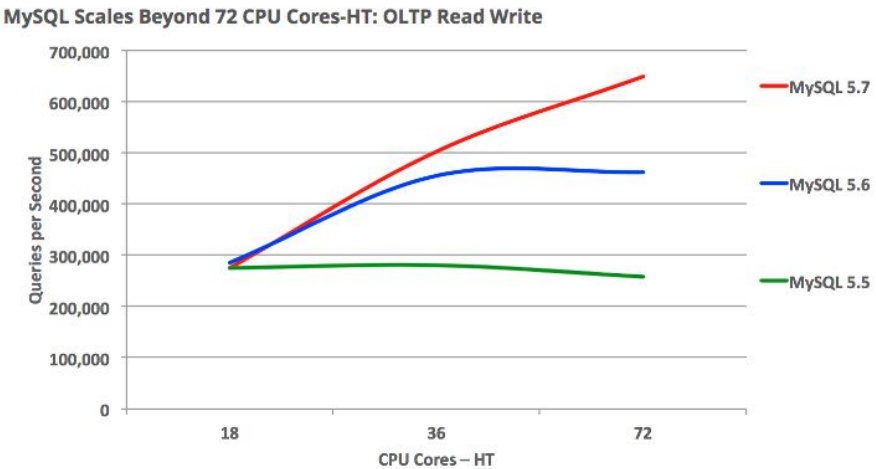
Sarath Lakshman, Sriram Melkote, John Liang, Ravi Mayuram
Couchbase, Inc

Presenter: Xiaoyao Qian • 04.04.2017

4 million entries/sec

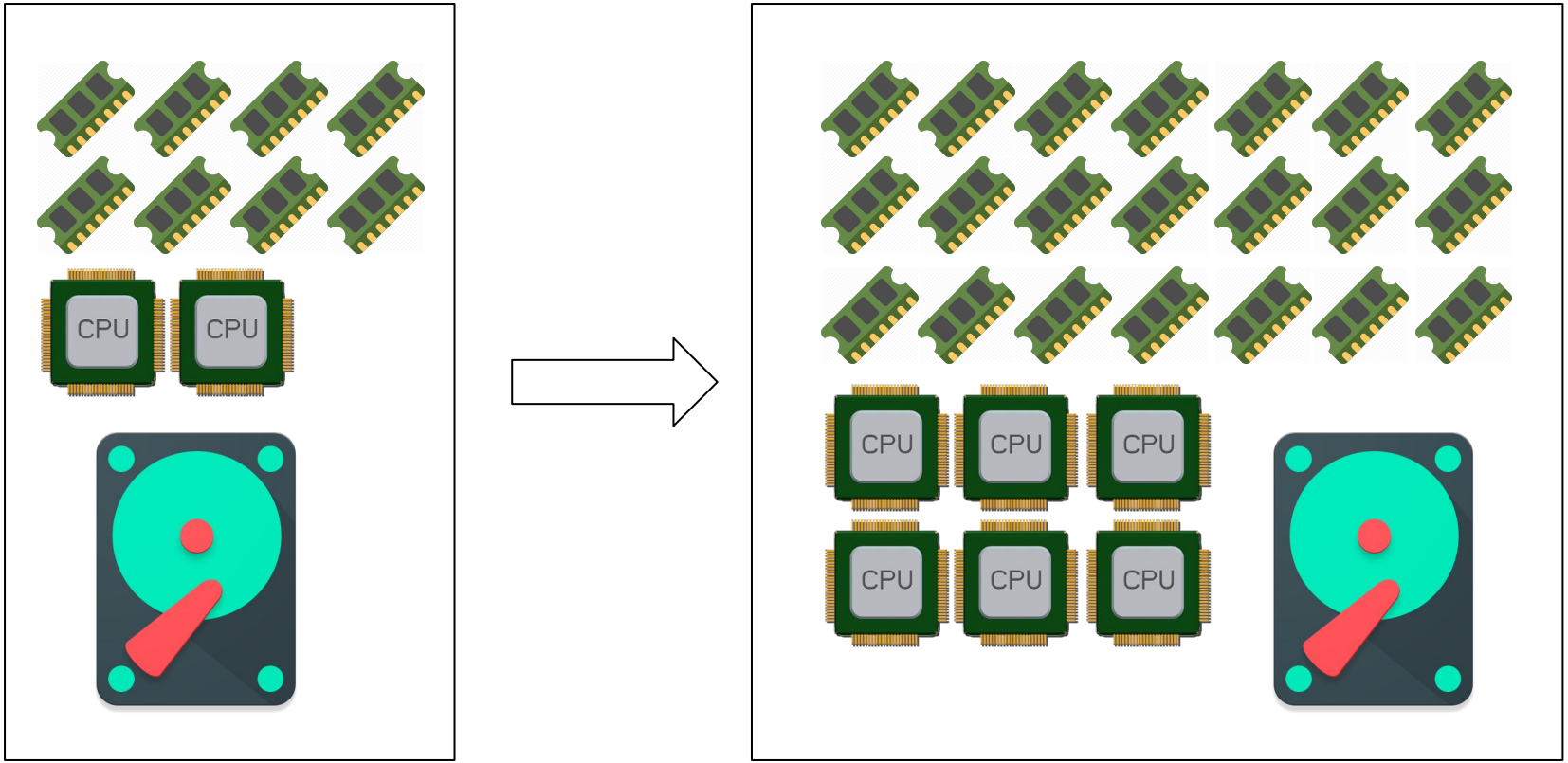
10 million lookups/sec

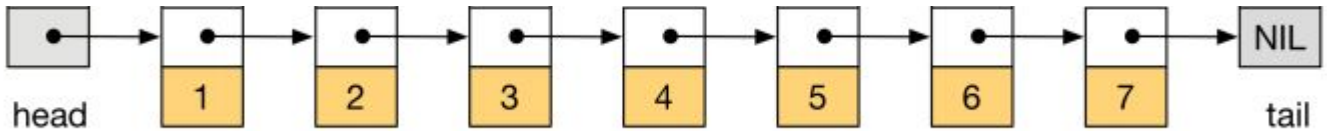




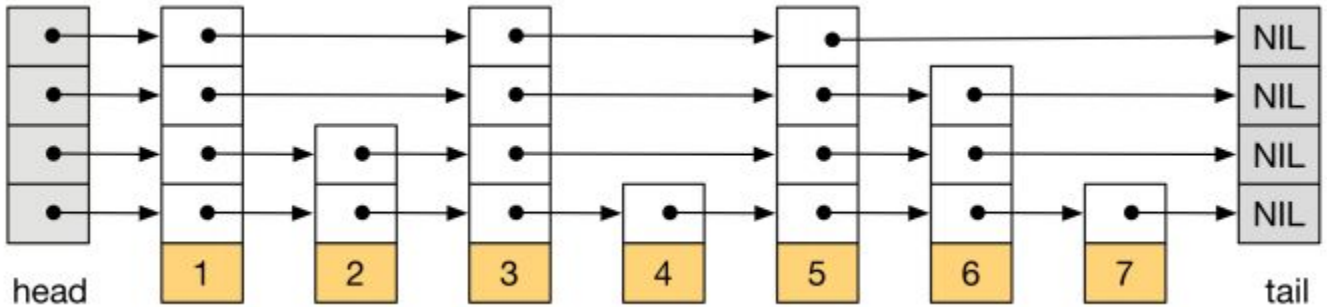
https://www.mysql.com/why-mysql/benchmarks/

Motivation


CPU
CPU
CPU
CPU
CPU
CPU
CPU

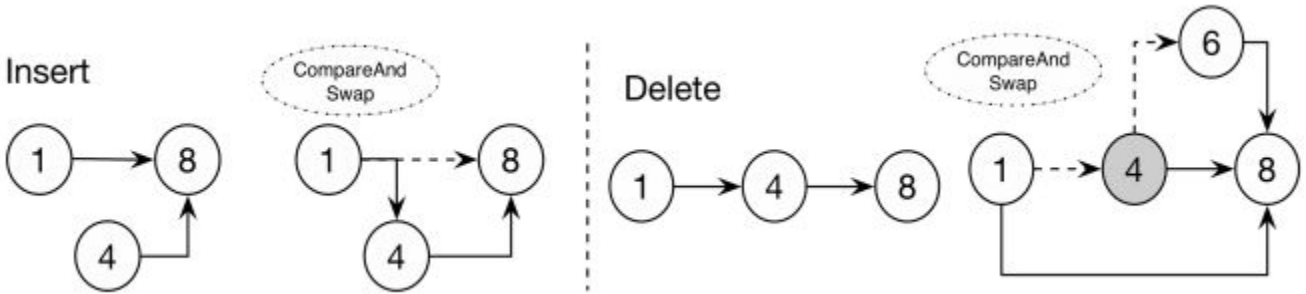


Ordered Linked List

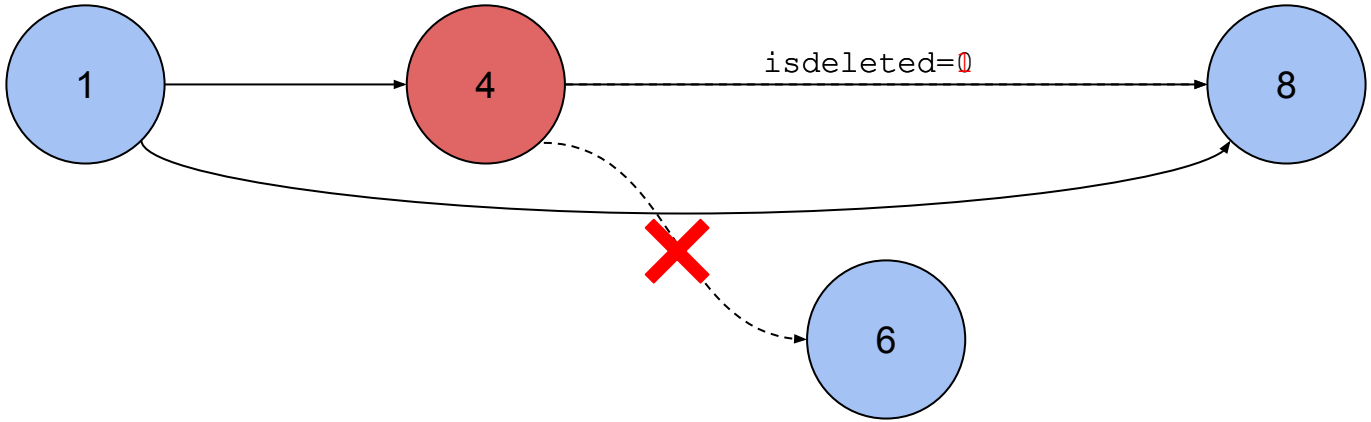


n: #nodes in next level
f: fanout factor

Avg O(logN): insert, lookup, delete



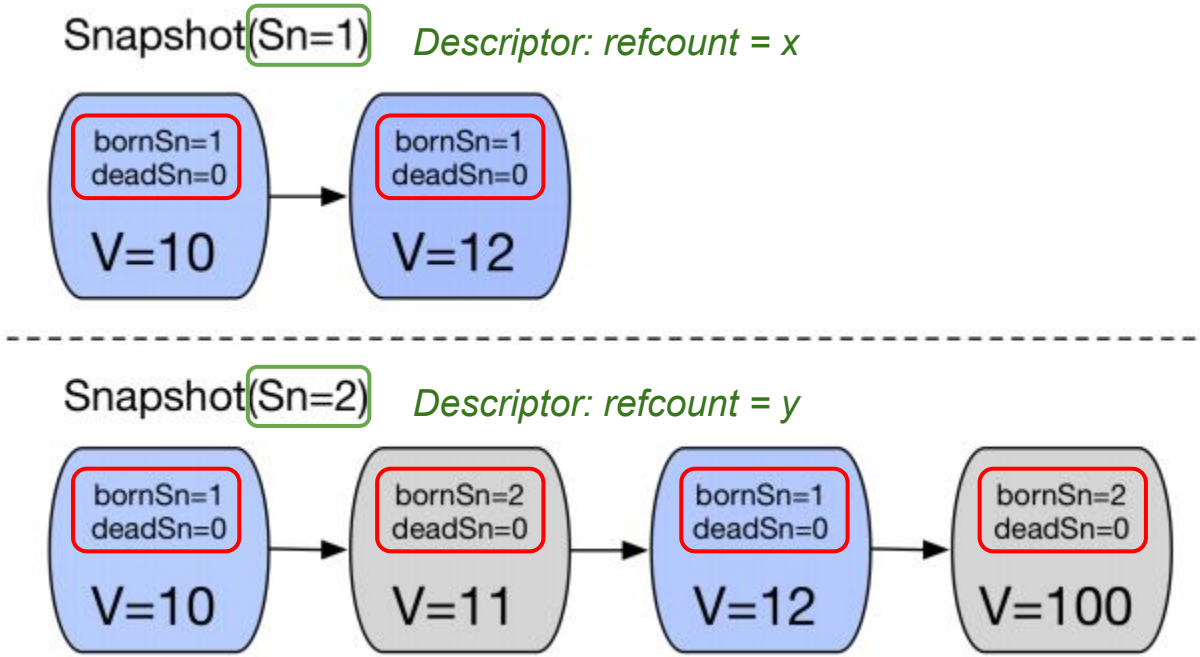
Lock-free List Operations



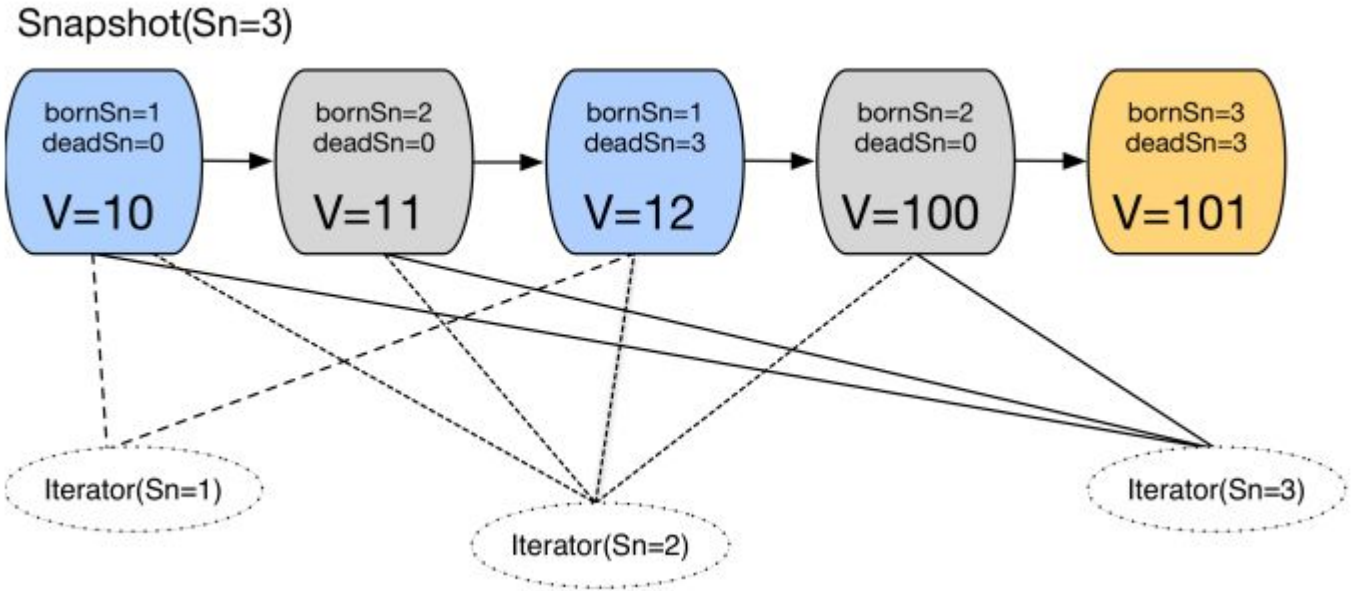
DoubleCAS

MVCC: Multi-Version Concurrency Control

- Immutable snapshots
- Fast and low overhead snapshots
- Avoid phantom reads
- Memory efficiency
- Fast and scalable garbage collection

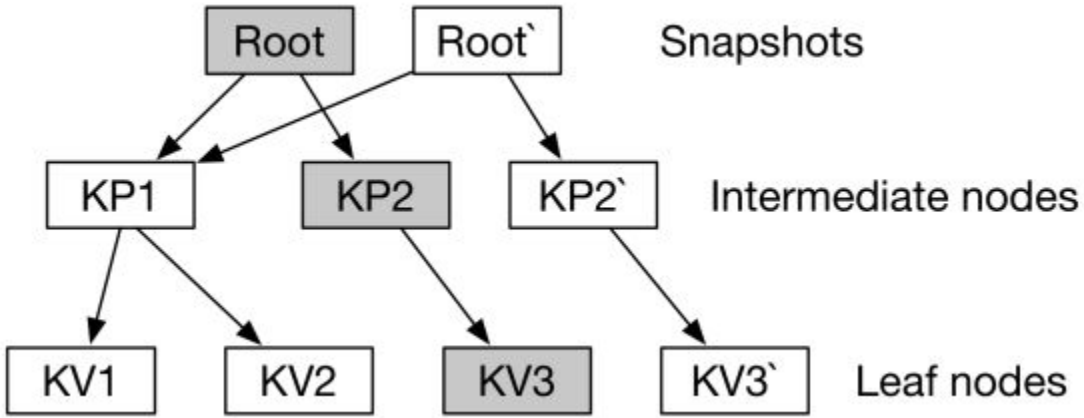


MVCC primitives: *lifetime and descriptor*



Snapshot Iteration

filter with `bornSn>termSn && deadSn>=termSn`

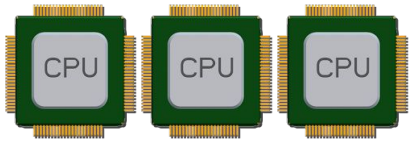


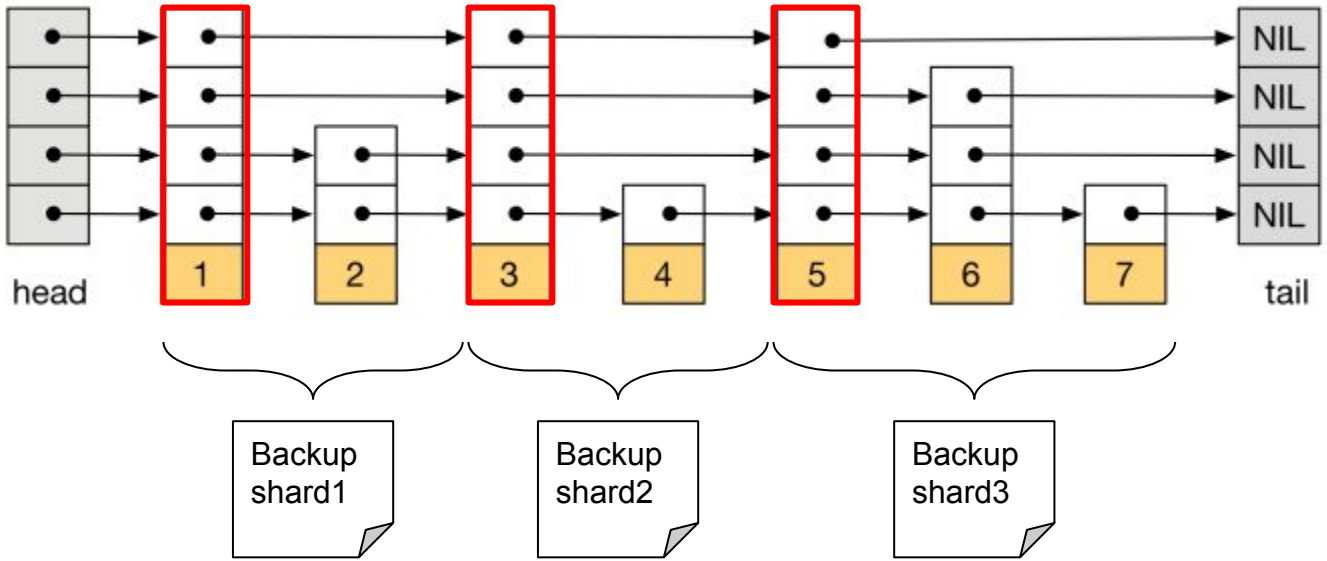
Comparison with Copy-On-Write B+ Tree
(COW B+)

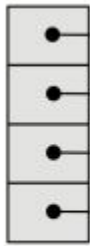
1. The snapshot Sn(x) descriptor shows refcount = 0
2. The previous snapshot Sn(x-1) has been garbage collected, i.e garbage collection of snapshots can only be performed in the sequential order of the snapshot `termSn`
3. #gc_workers = #concurrent_writers
4. Writers keep track of `deadList` which is attached to the snapshot descriptor. Whenever a node is marked as deleted, add to `deadList`.
5. GC workers use `deadList` of a snapshot to perform physical node removal from the skiplist

1. Traverse level 0 linked list of the skiplist, and write out the entries into data files
2. All entries that don't belong to the snapshot are ignored
3. Node metadata (i.e lifetime) are not serialized. They can be recreated during recovery

✓ Minimum backup file size
✓ Compression friendly
✓ Since skiplist is ordered, the data written to disk is also ordered
× Could block garbage collection

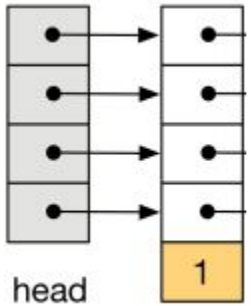

CPU
CPU
CPU


NIL
NIL
NIL
NIL
head
1
2
3
4
5
6
7
tail
Backup shard1
Backup shard2
Backup shard3



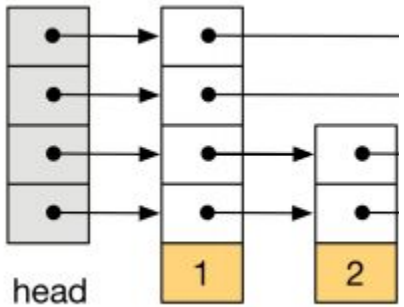
Buf: [nil, nil, nil, nil]

Recovery



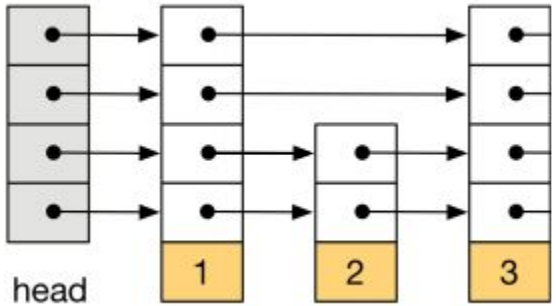
Buf: [nil, nil, nil, nil] -> [n1, n1, n1, n1]

Recovery



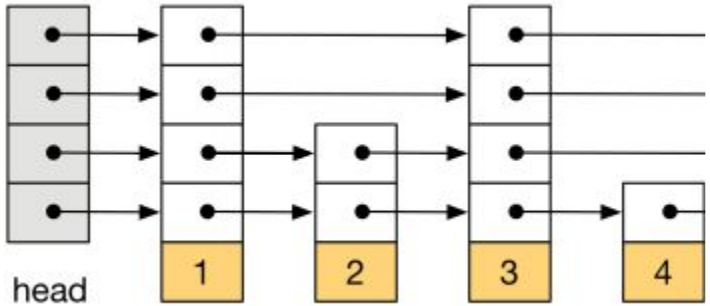
Buf: [n1, n1, n1, n1] -> [n2, n2, n1, n1]

Recovery



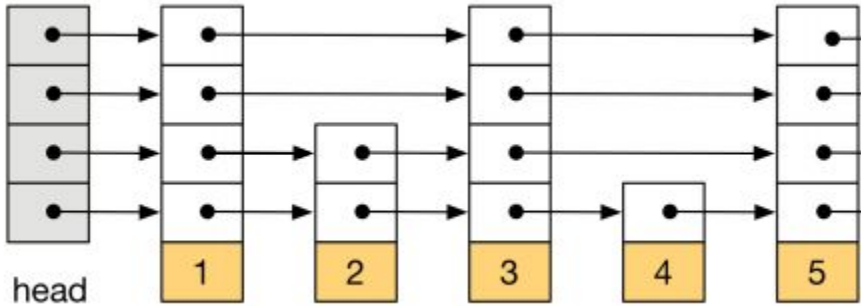
Buf: [n2, n2, n1, n1] -> [n3, n3, n3, n3]

Recovery



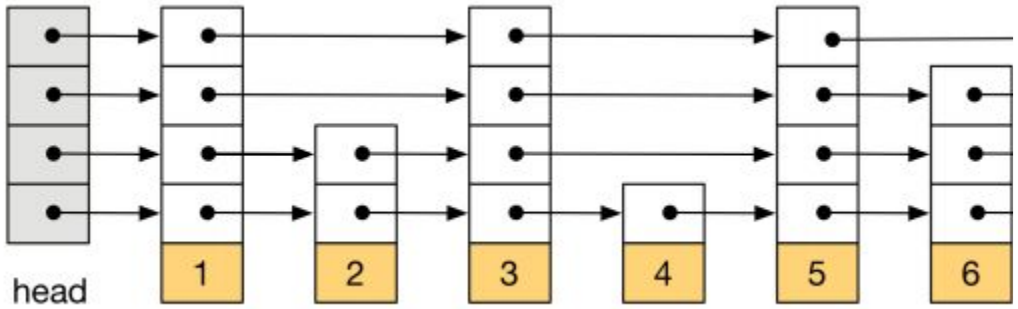
Buf: [n3, n3, n3, n3] -> [n4, n3, n3, n3]

Recovery



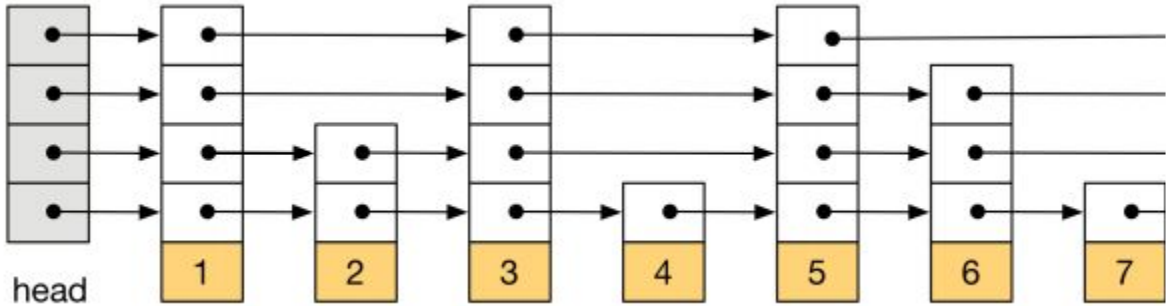
Buf: [n4, n3, n3, n3] -> [n5, n5, n5, n5]

Recovery



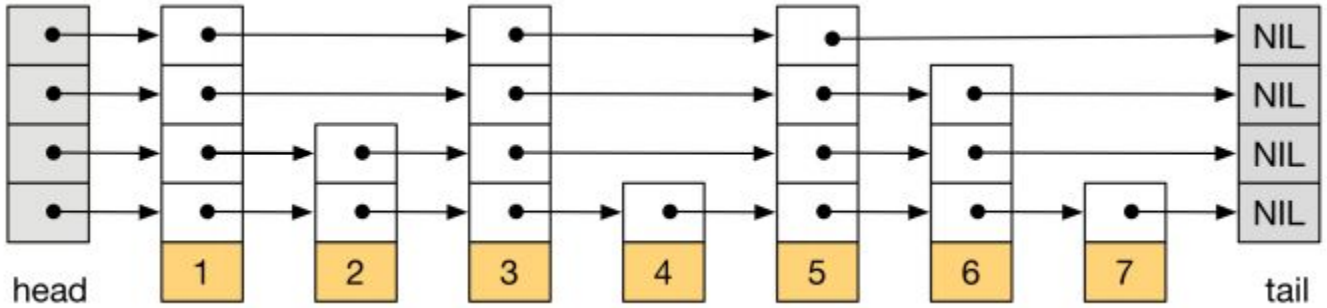
Buf: [n5, n5, n5, n5] -> [n6, n6, n6, n5]

Recovery



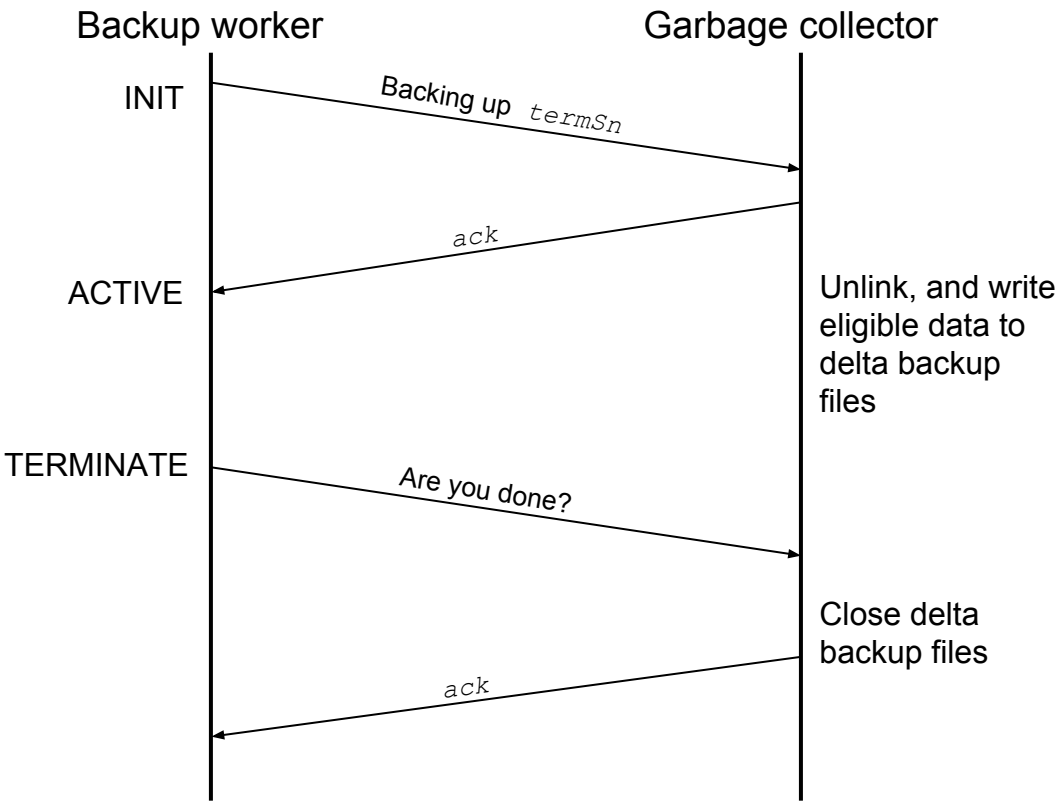
Buf: [n6, n6, n6, n5] -> [n7, n6, n6, n5]

Recovery



Buf: [n7, n6, n6, n5] -> [nil, nil, nil, nil]

Recovery



Non-intrusive Backup

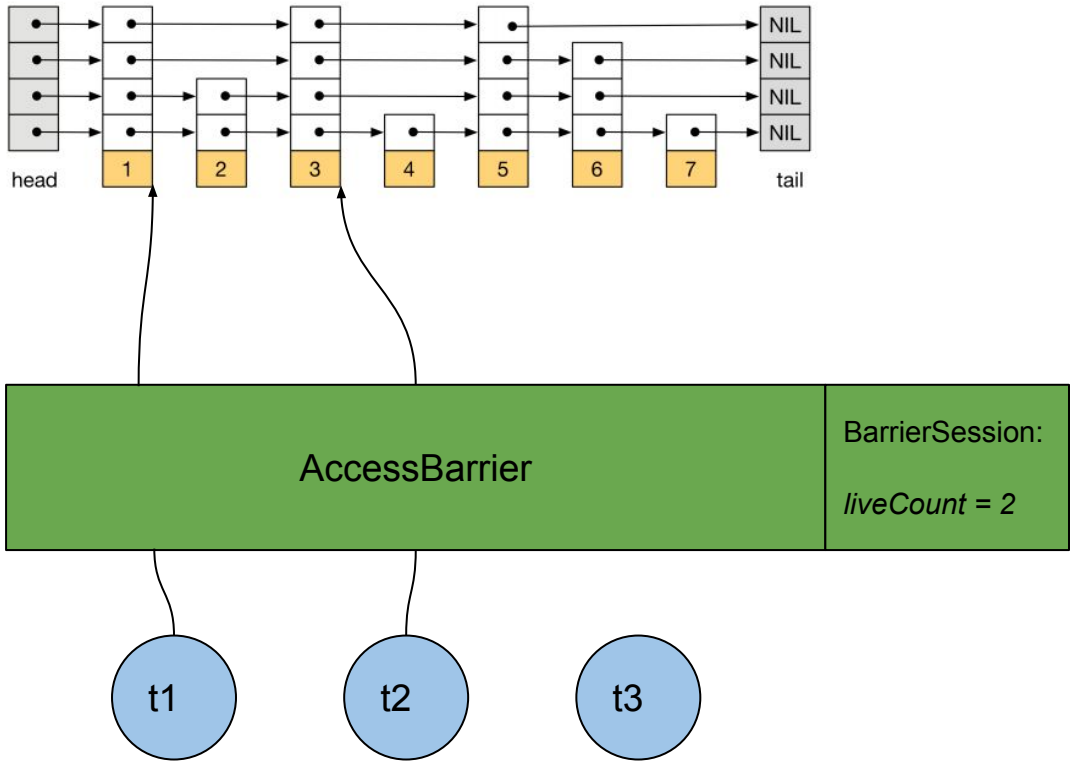
1. Any thread accessing the lock-free skiplist is called an accessor.

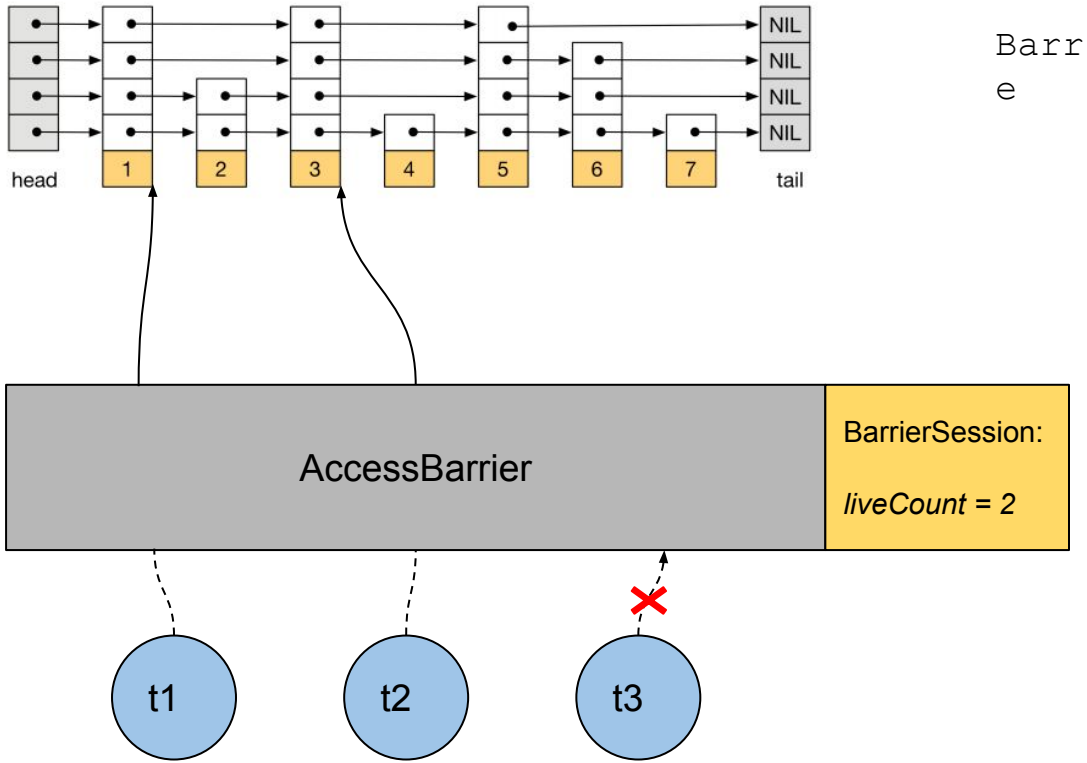
2. If there are no accessors currently present in the skiplist for a node unlinked from the skiplist, it is safe to free the node.

3. If a node n is unlinked at a time, t0. Any accessors that came after t0 will not be able to access the node n or hold a reference to node n.

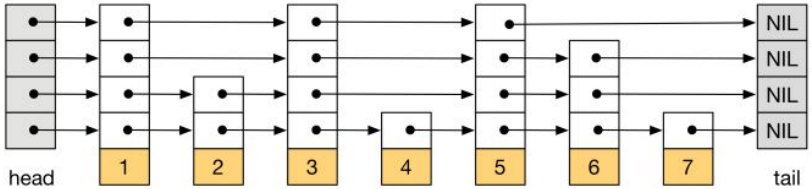
4. If there are k accessors in the skiplist after a node n is unlinked, from (3) we know that it is safe to free node n once k accessors finish their operation.

5. If x nodes are unlinked, it is safe to unlink these x nodes once all the accessors which were present in the skiplist during xth node unlink leave the skiplist.


NIL
NIL
NIL
NIL
head
1
2
3
4
5
6
7
tail
AccessBarrier
BarrierSession:
liveCount = 2
t1
t2
t3




```
BarrierSessionClos
e
```


NIL
NIL
NIL
NIL
head
1
2
3
4
5
6
7
tail

`Terminated`


AccessBarrier
BarrierSession:
liveCount = 2

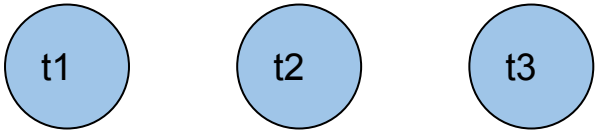

t1
t2
t3

Figure 6: Insert throughput

Figure 7: Get throughput

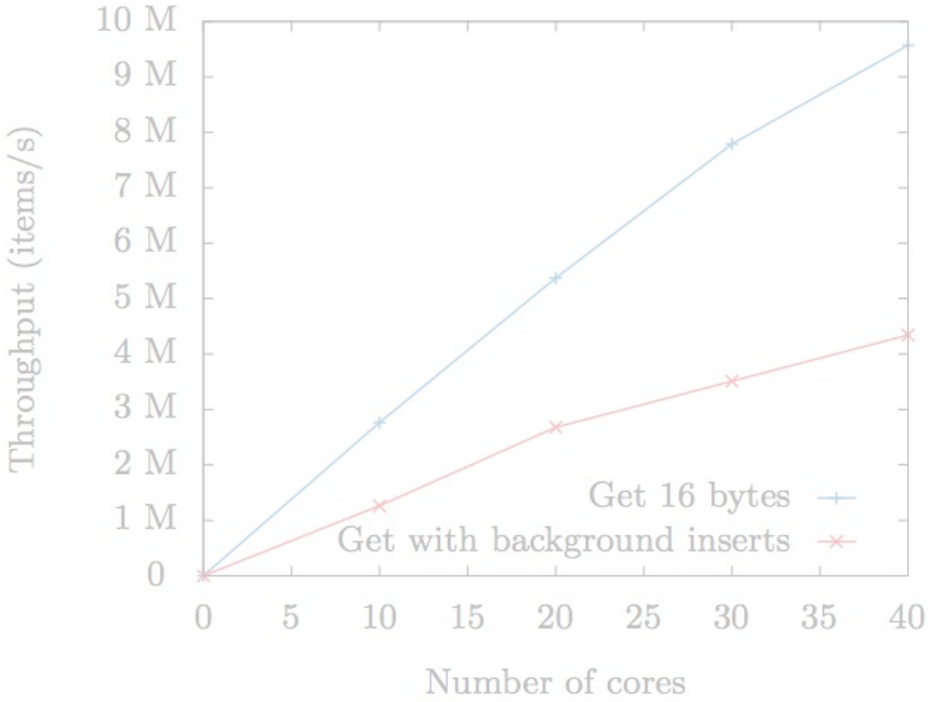


Figure 8: Get with mutations throughput

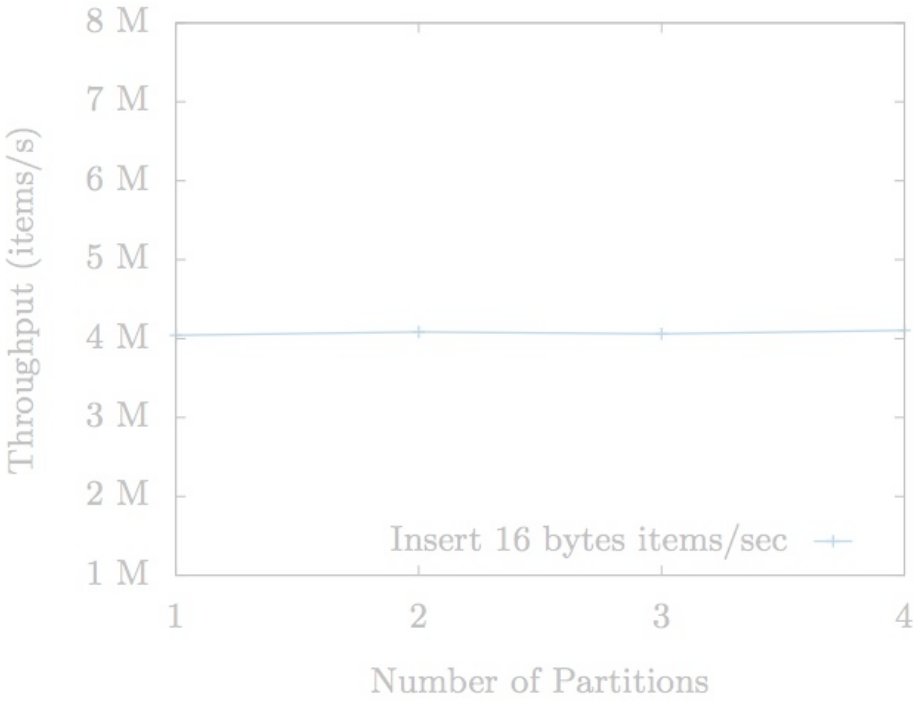
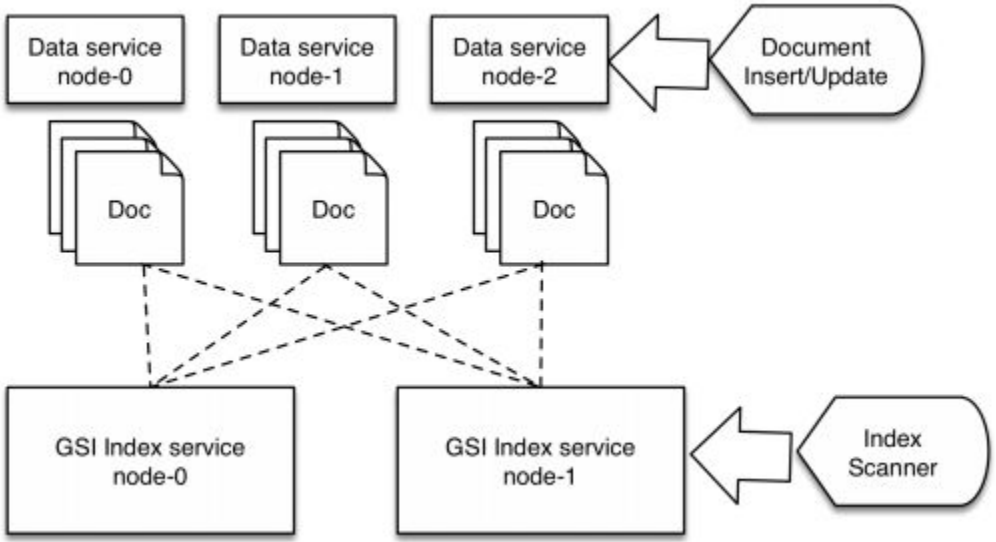


Figure 9: Scaling with number of partitions



Global Secondary Index architecture

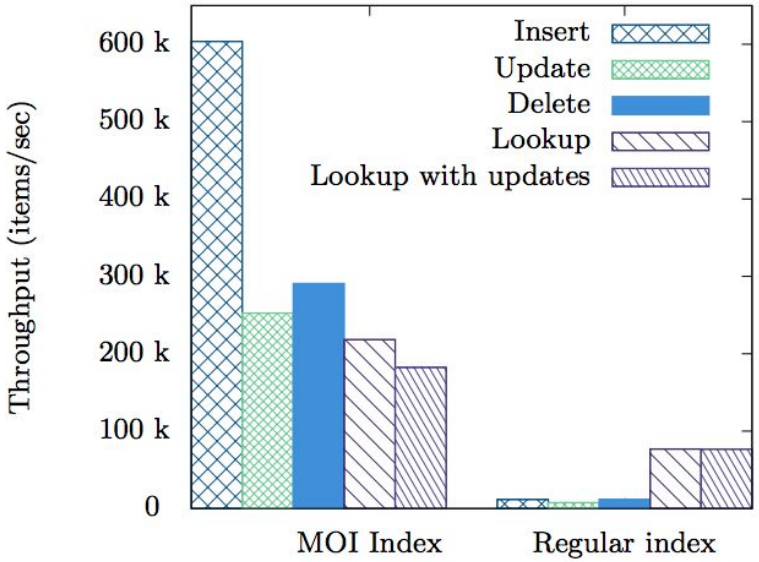


Figure 11: Single index performance

Table 1: GSI index server performance (items/sec)

Operation	MOI Indexes	Regular Indexes
Create Documents	1,658,031	88,102
Update Documents	822,680	70,802
Delete Documents	1,578,316	80,578

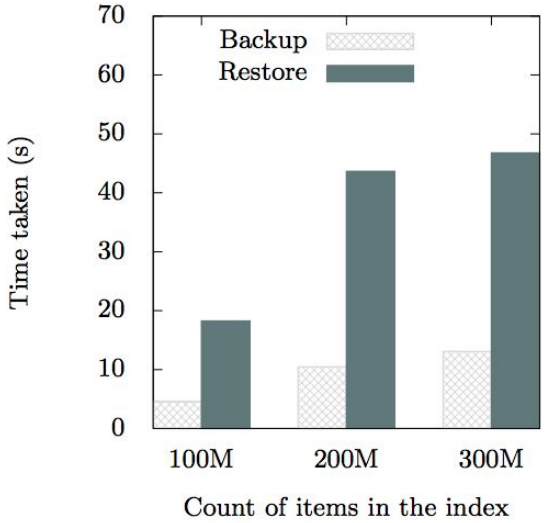


Figure 12: Index recovery performance

“TALK IS CHEAP, SHOW ME THE CODE”

https://github.com/couchbase/nitro

~15,000 lines of code

mainly in Golang, with a little C/C++

Apache 2.0 Licence

Questions & Discussions

1. #GC_workers = #writers? Wouldn’t that be too intense?
2. Skiplist may not be good in cache utilization because of not consecutive memory. Can this be optimized?
3. How can a single large index be distributed?