SLIK: Scalable Low-Latency Indexes for a Key-Value Store

Ankita Kejriwal

Stanford University

(Joint work with Arjun Gopalan, Ashish Gupta, Zhihao Jia and John Ousterhout)

Introduction

- **RAMCloud 1.0**
- **Higher-level data models**
  - Without sacrificing latency and scalability
- **SLIK:**

  Scalable, Low-latency Indexes for a Key-value Store
- **Lookups and range queries on attributes that are not the primary key (i.e., secondary keys!)**
- **Performance**
  - 10-15 µs indexed reads.
  - 34-43 µs writes/overwrites of objects with one indexed attribute.

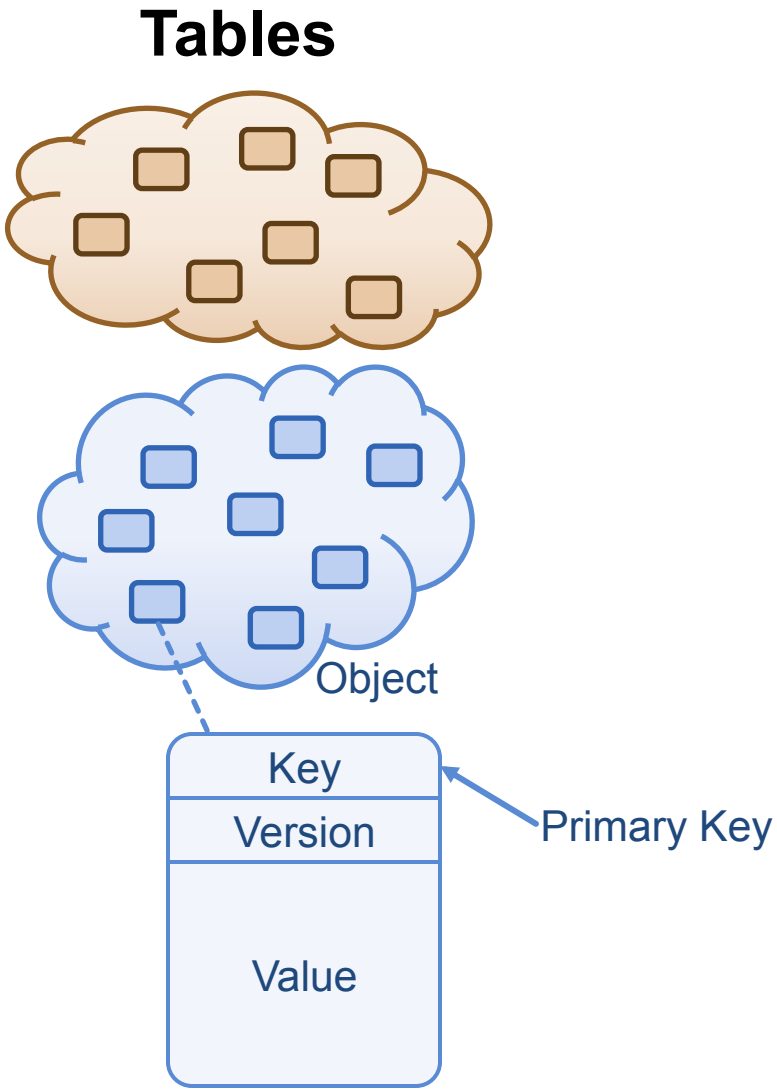
SLIK Status

- **Basic Implementation Done**
- **Paper under submission**
- **Still more to be done**
- **SEDCL Context:**
  - Forum 2014: Design done, implementation underway
  - Retreat 2014: Basic implementation done, have preliminary performance

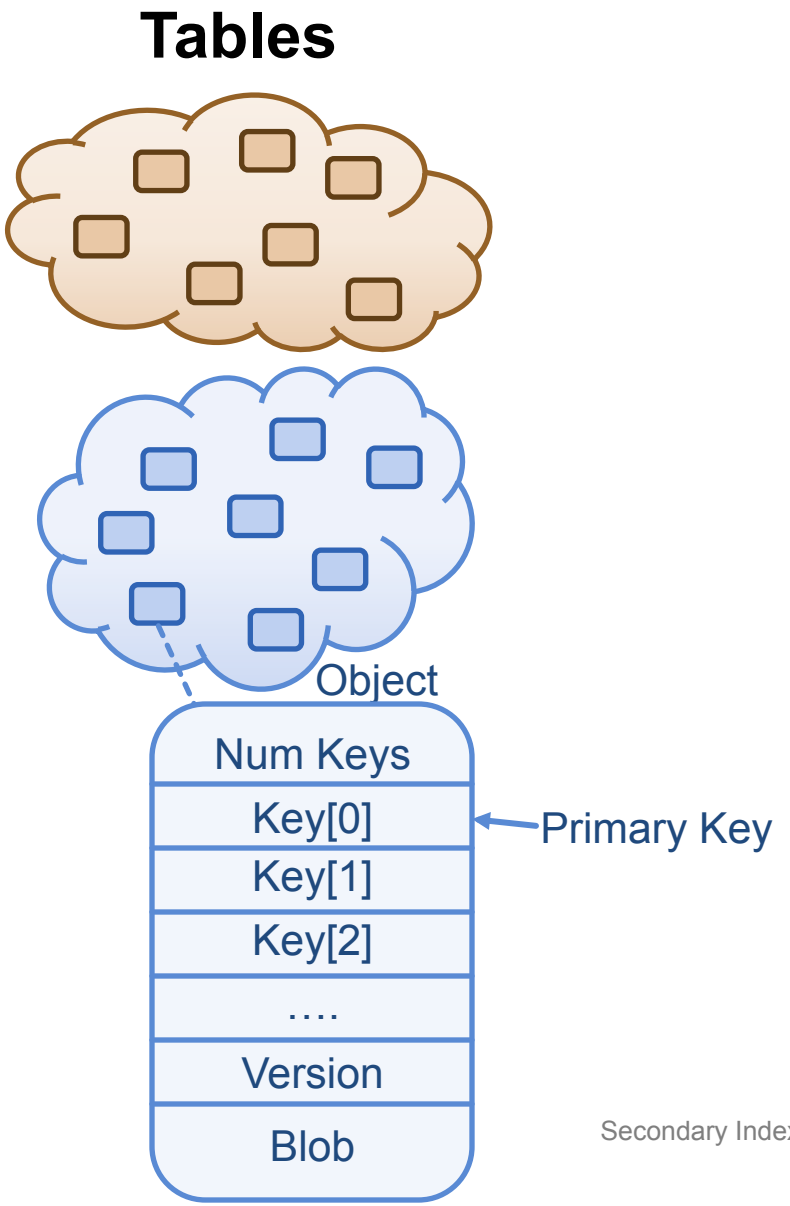
What we've built

- Object format and API
- Index placement / partitioning
- Index memory allocation
- Failure / Restoration
- Consistency

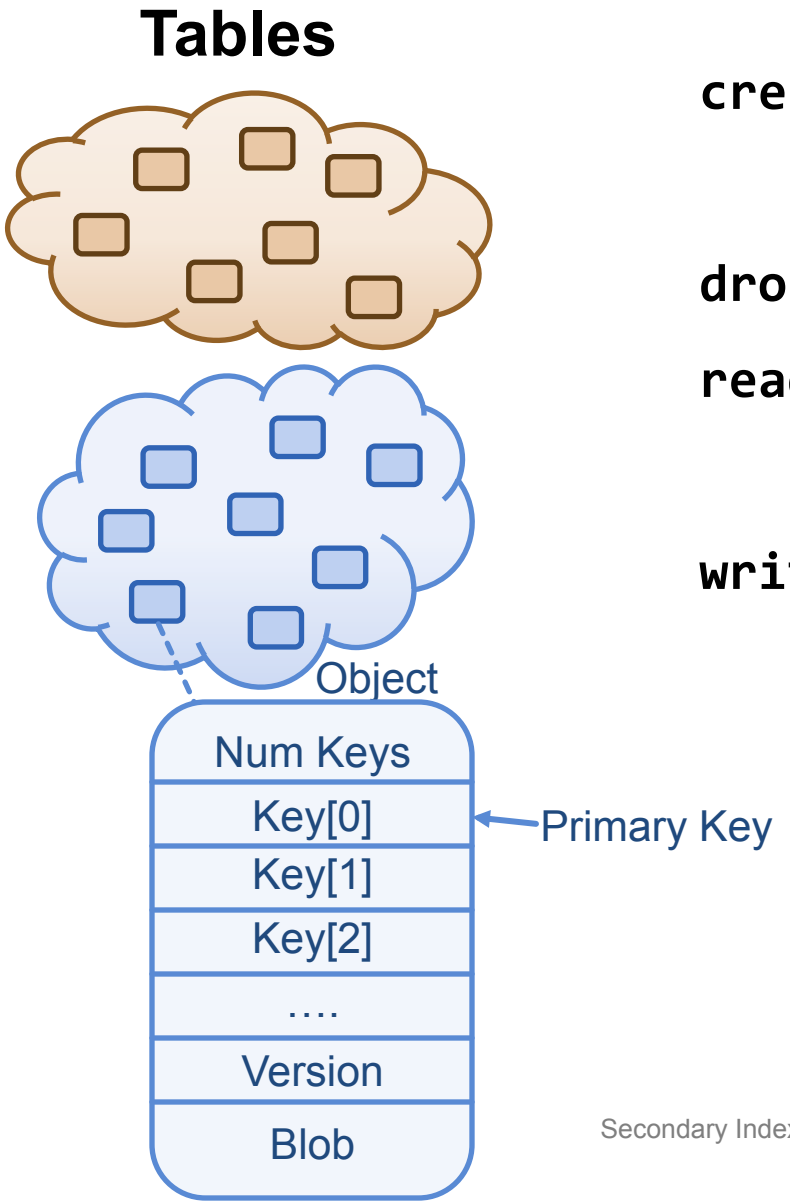
Object Format and API



Object Format and API



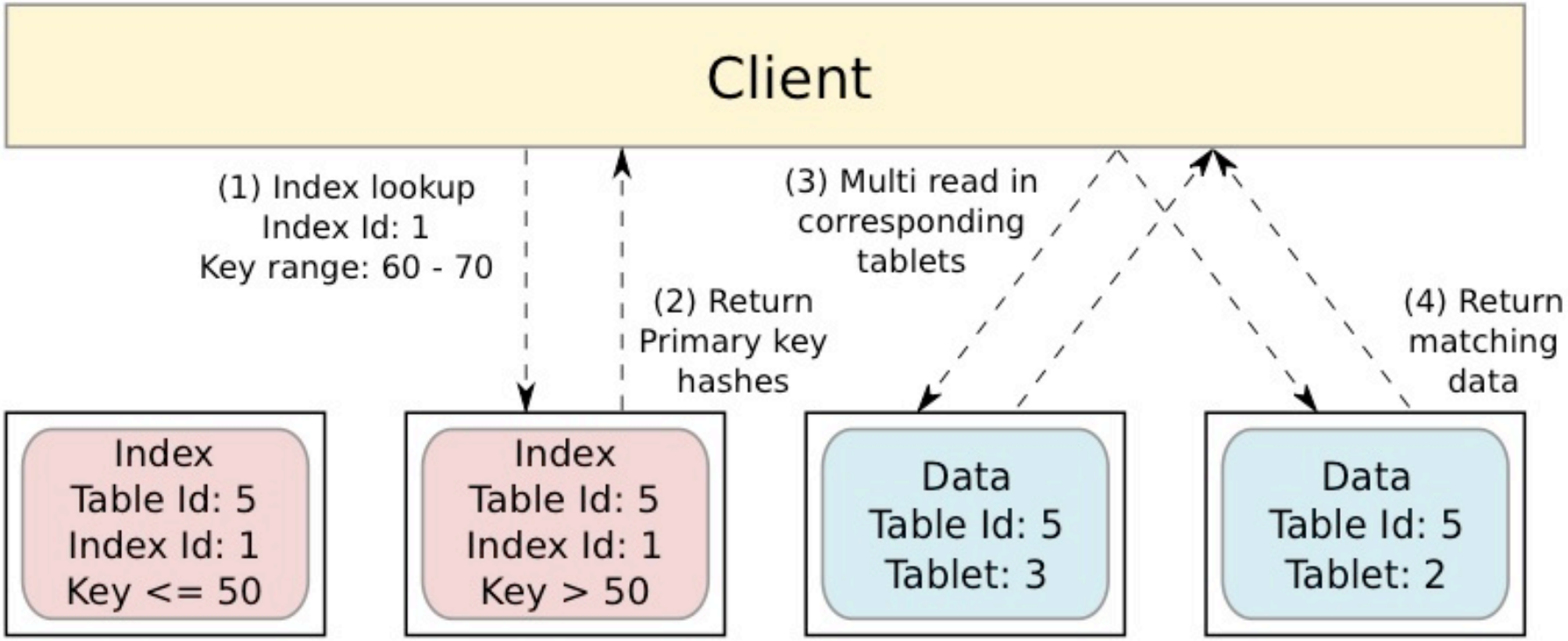
Object Format and API




```
createIndex(tableId, indexId,
            indexType)
dropIndex(tableId, indexId)
readRange(tableId, indexId,
          firstKey, lastKey)
write(tableId, keys, value)
```

Index Placement / Partitioning

- **Goal: Scalability**
- **Distribute index and table independently**

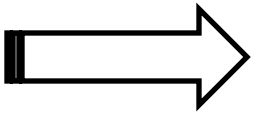


Index Memory Allocation

- **Index structured as Btree**
  - Panthema STX B+ Tree open source package
- **Mapped tree onto RAMCloud objects**

Failure / Restoration

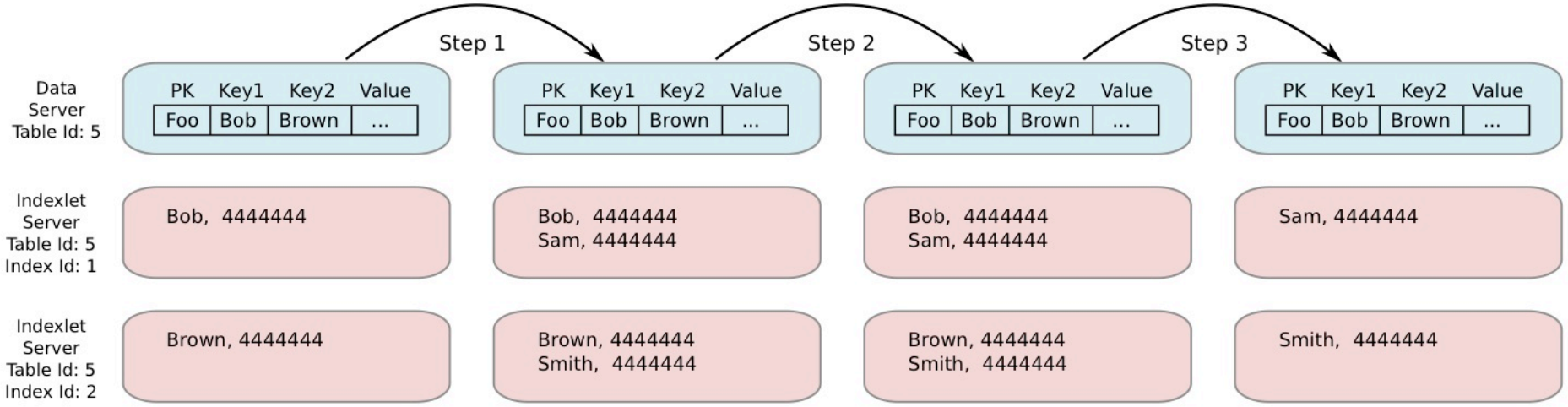
Index stored in RAMCloud log



Use RAMCloud Crash Recovery

Consistency

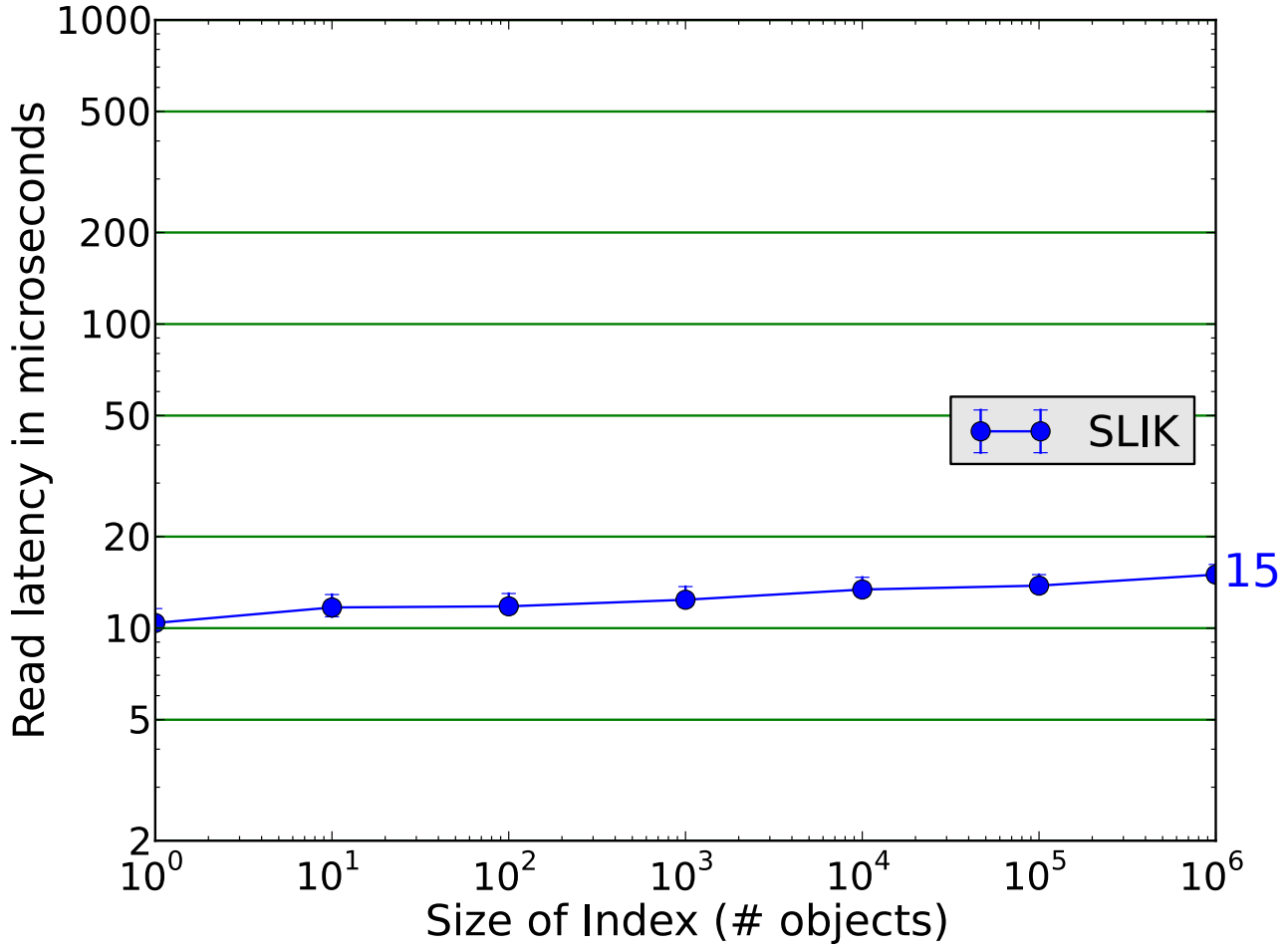
- **Indexed object writes: distributed operation**
- **Goal: Strong consistency**
- **Goal: Avoid transactions**
- **Solution: Ordered write + longer index lifespan + liveness determined by object**
- **Downside: Sometimes, leak memory**



Performance

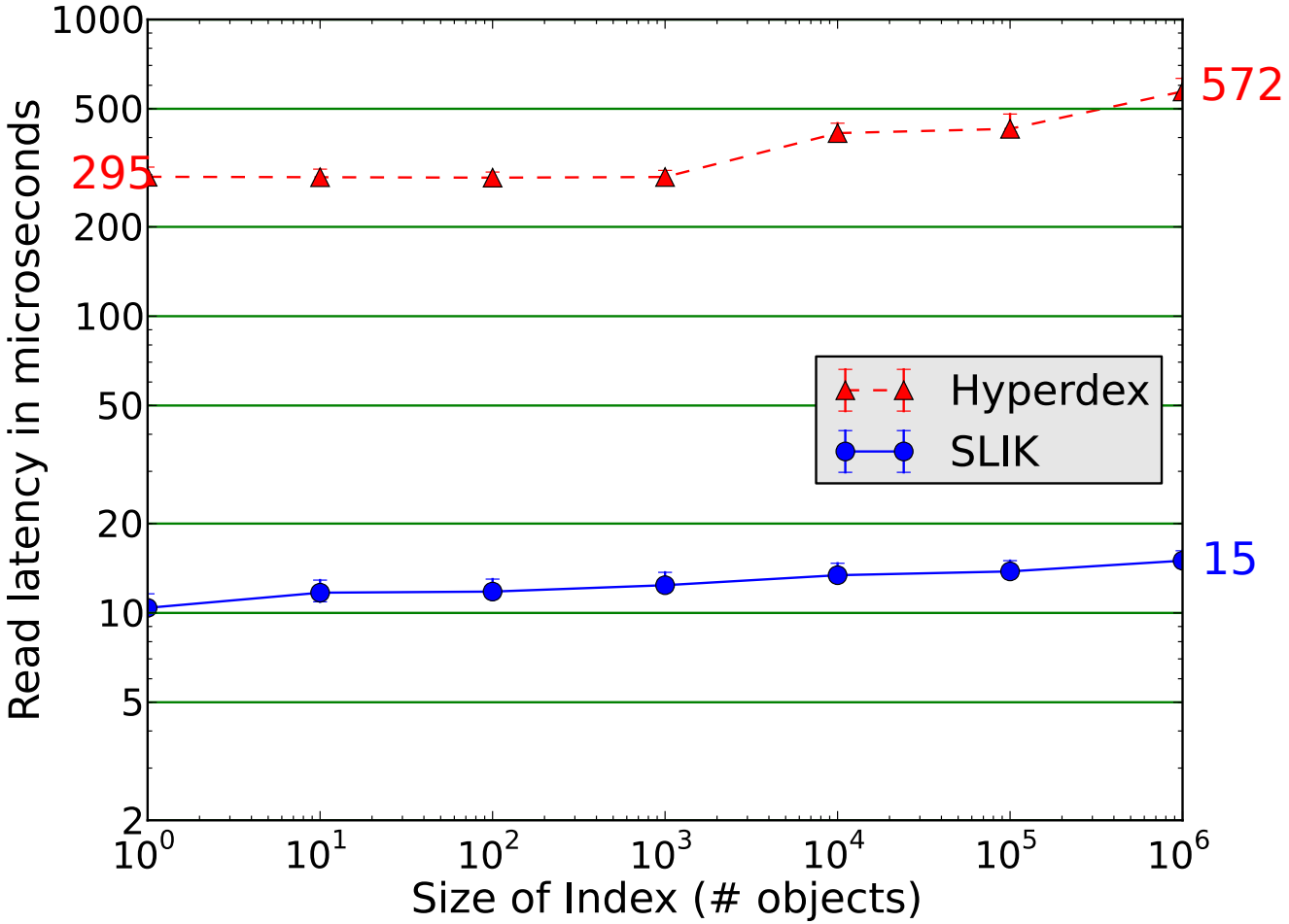
Basic Latency: Read

Varying num of objects; Each object: PK (30B) + IndexKey (30B) + Blob (100B)



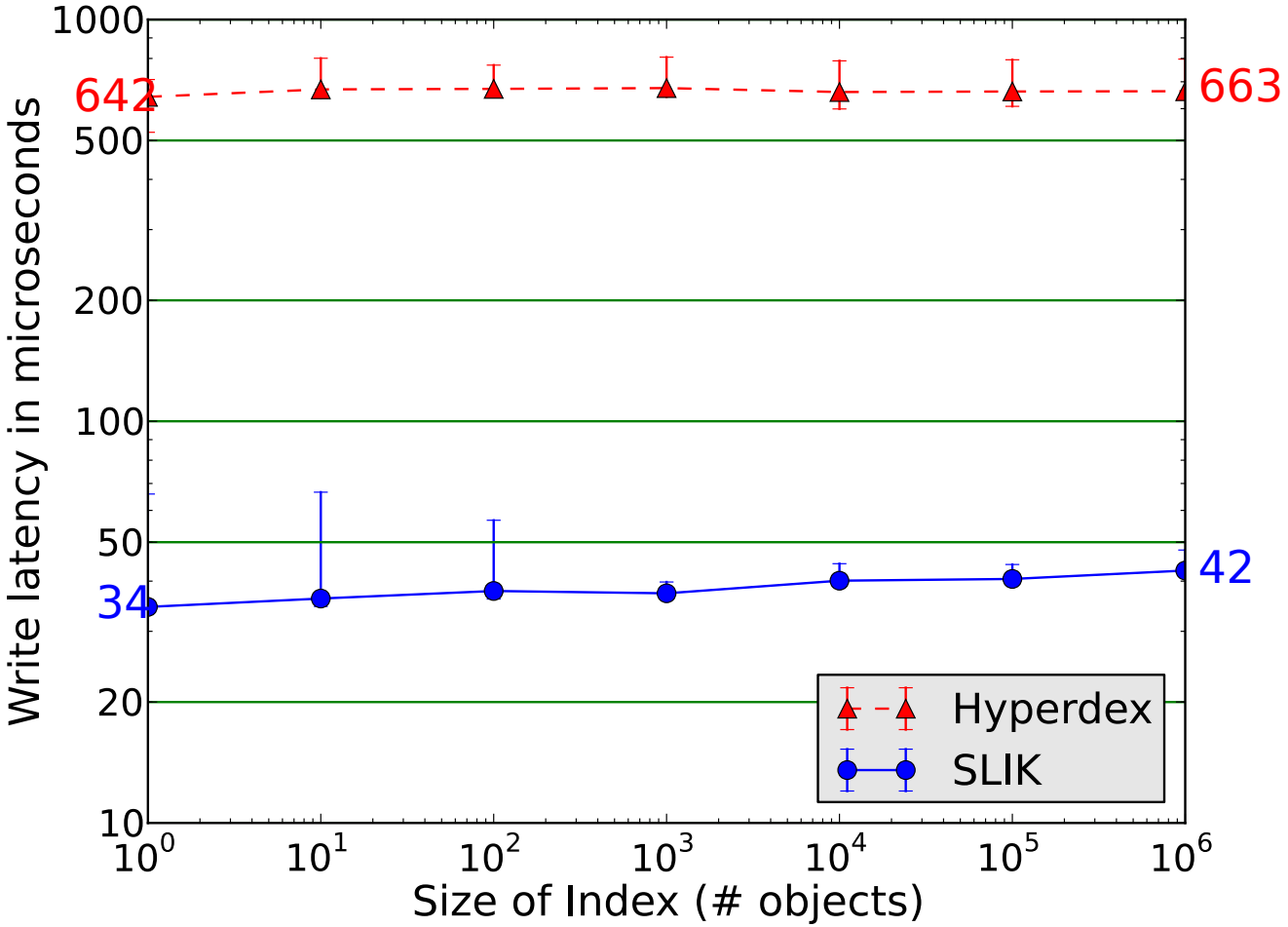
Basic Latency: Read

Varying num of objects; Each object: PK (30B) + IndexKey (30B) + Blob (100B)



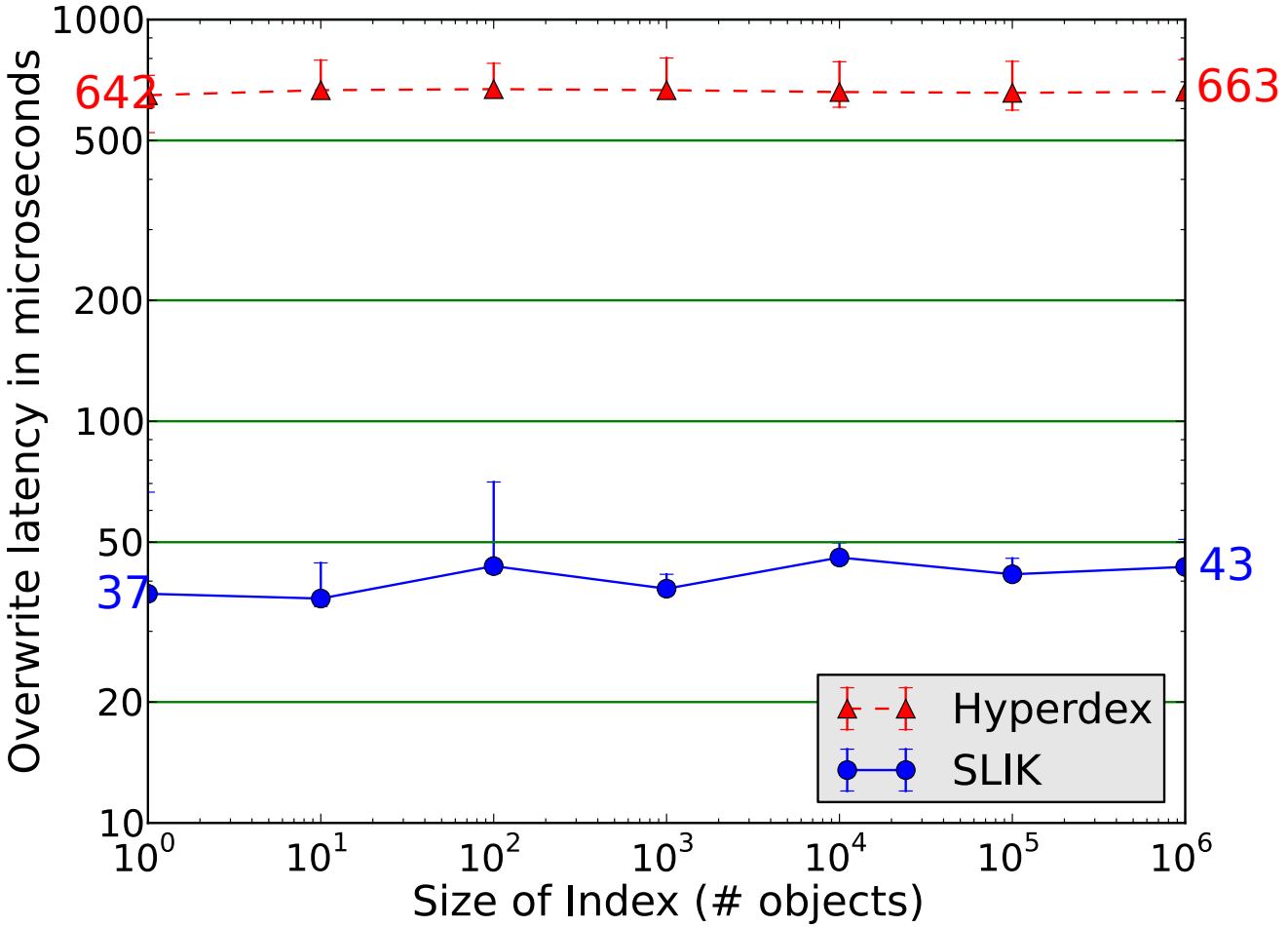
Basic Latency: Write

Varying num of objects; Each object: PK (30B) + IndexKey (30B) + Blob (100B)



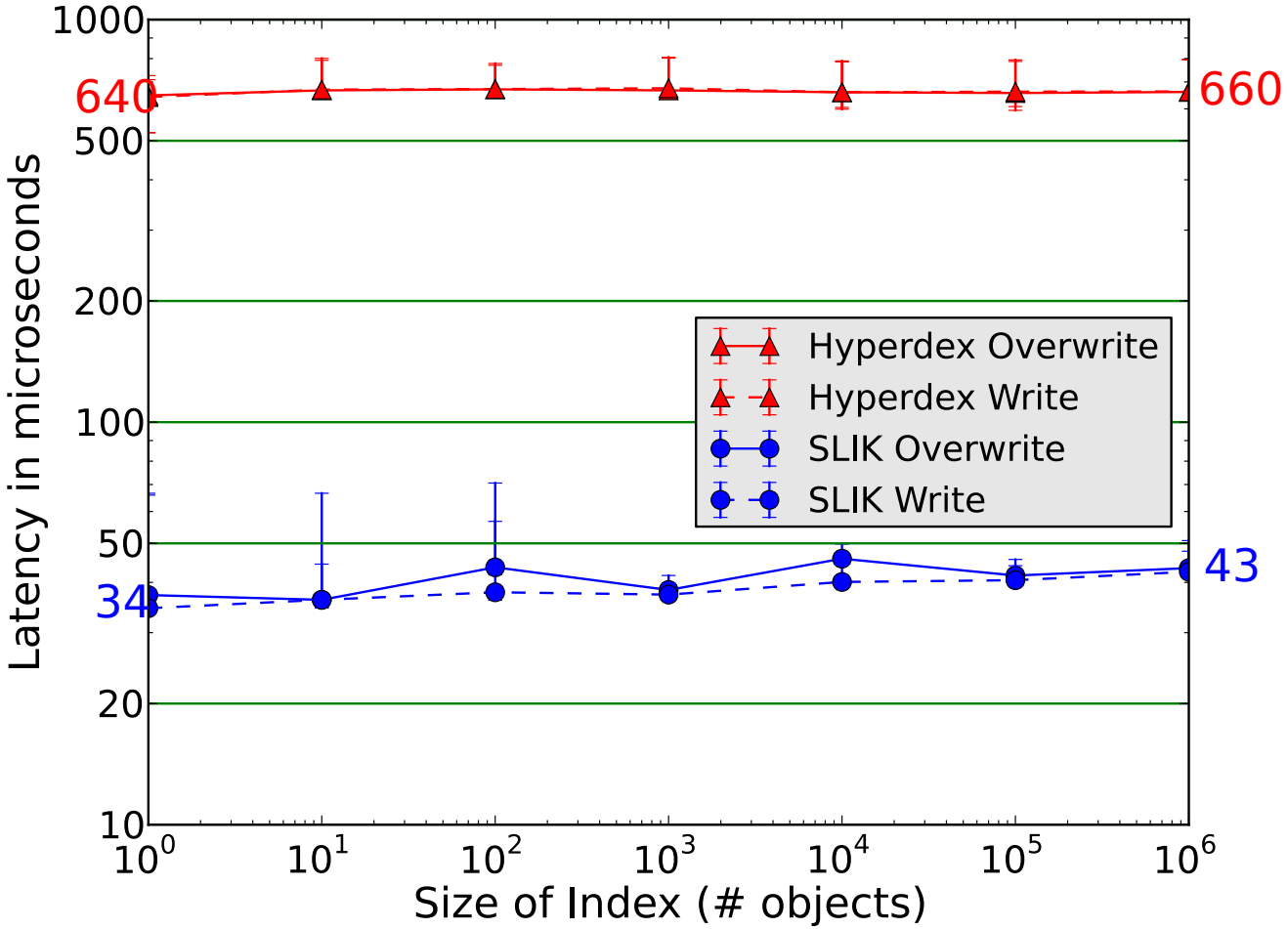
Basic Latency: Overwrite

Varying num of objects; Each object: PK (30B) + IndexKey (30B) + Blob (100B)



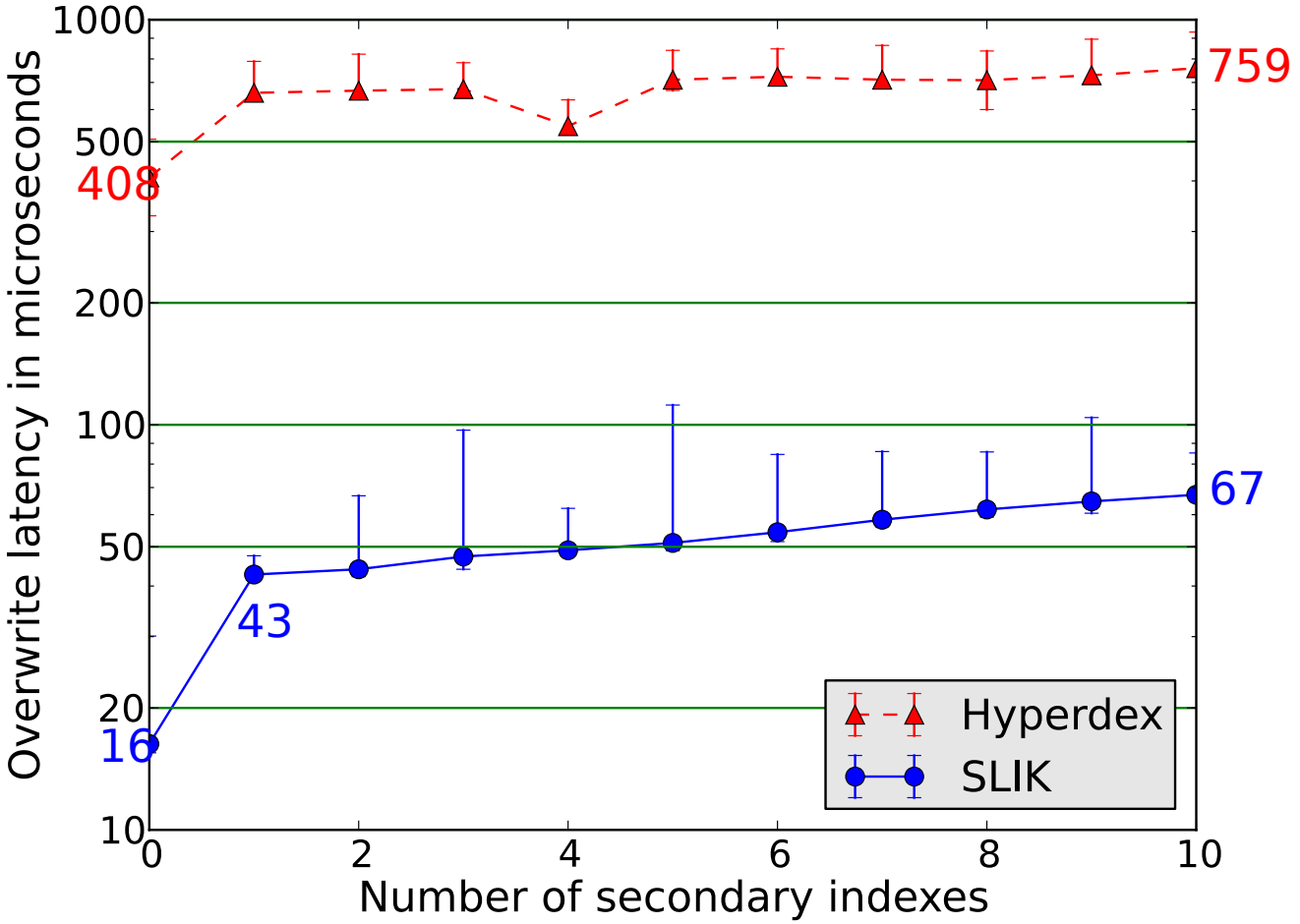
Basic Latency: (Over)write

Varying num of objects; Each object: PK (30B) + IndexKey (30B) + Blob (100B)



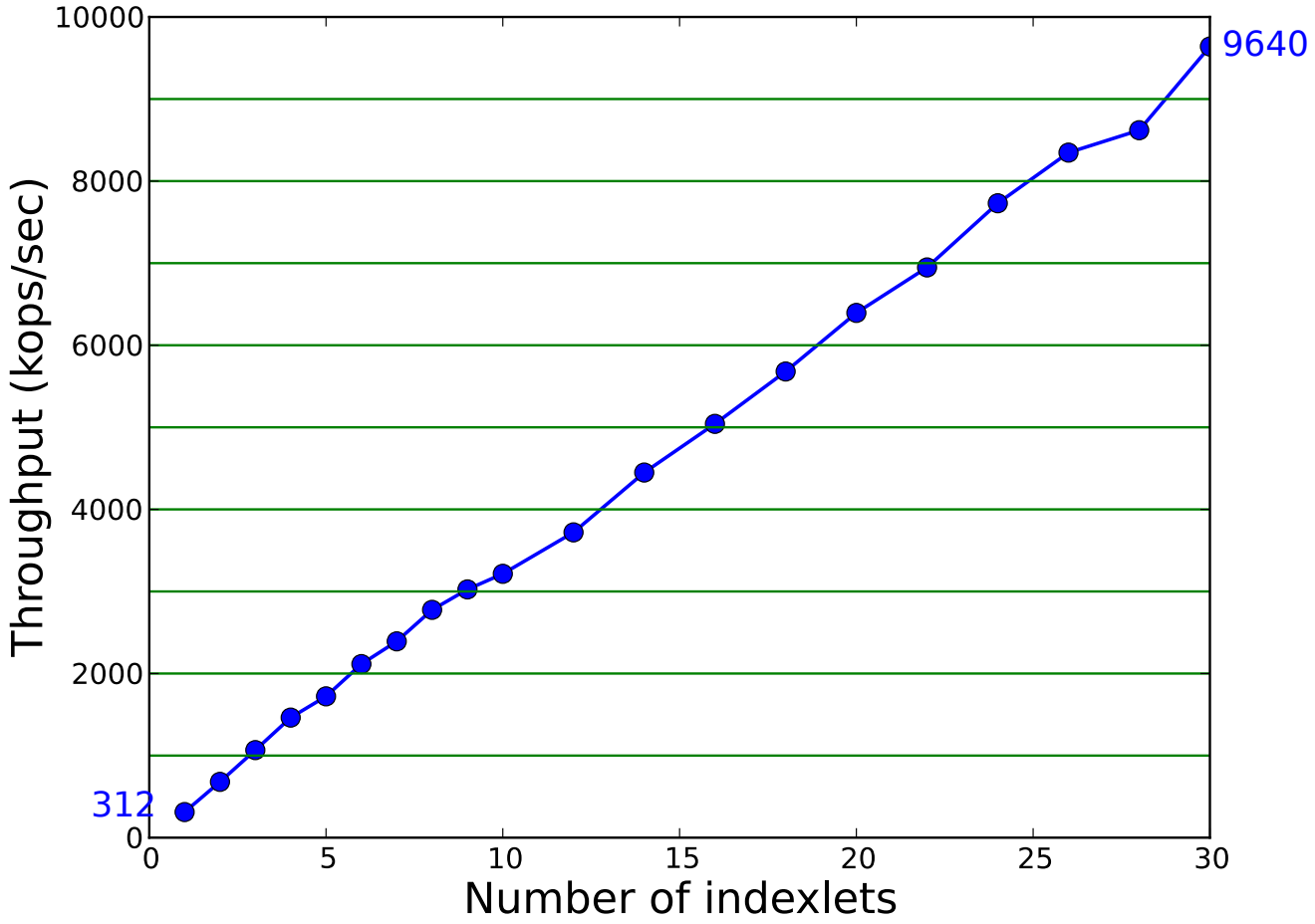
Multiple Secondary Indexes

1 Million objects; Each index and table on different server;
Each object: PK (30B) + Varying number of index key (each 30B) + Blob (100B)



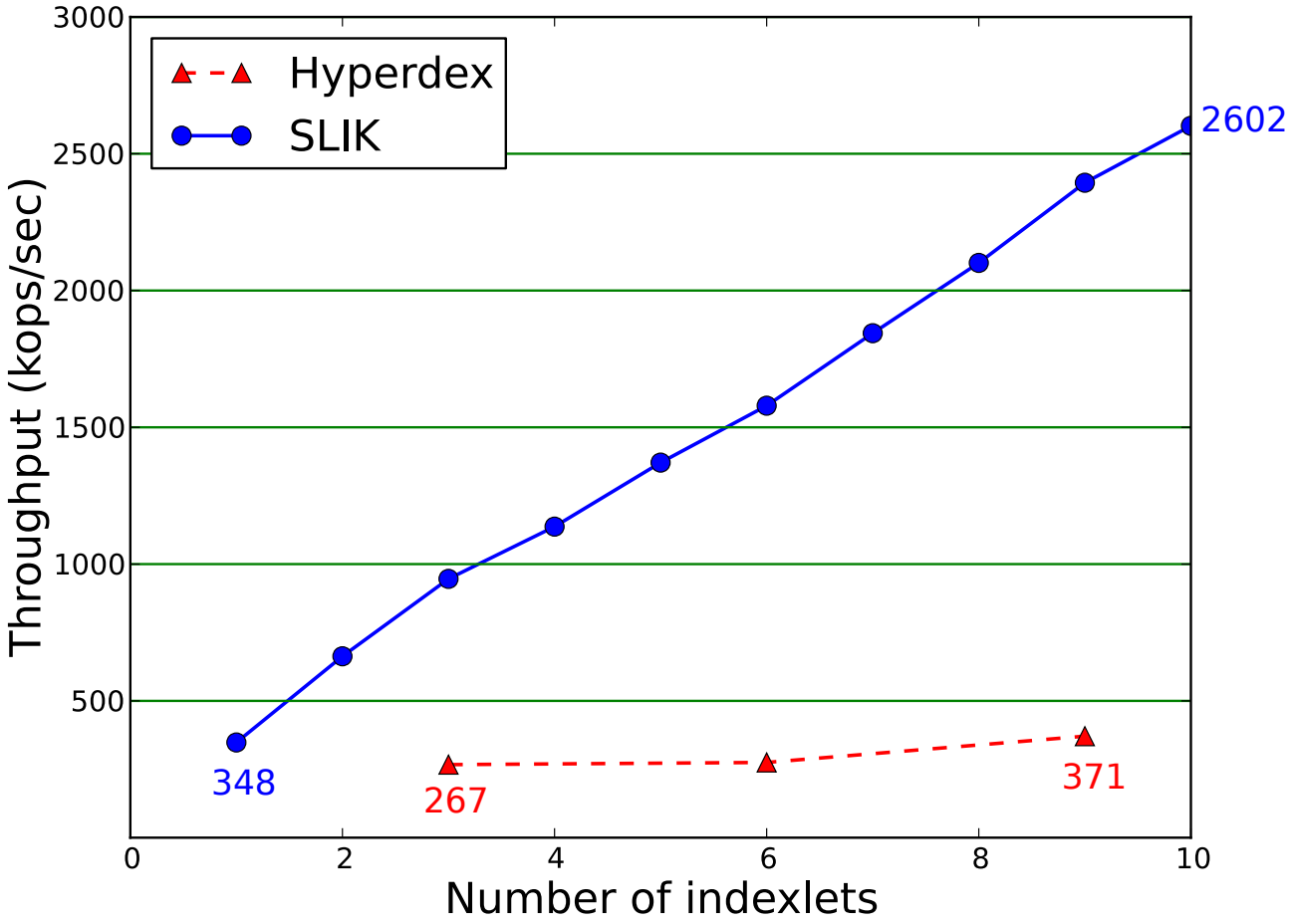
Scalability

1 Million objects; Each indexlet (index partition) on different server.
Index lookups. No data reads.



Scalability

1 Million objects; Each indexlet and tablet on different server.
Indexed reads: Index + data reads.



What's not there

- Client-Side Iterator API
- Coordinator Index Functions Recovery
- Index Partitioning
- B-Tree?

Summary

- **SLIK: lookups & range queries on secondary keys**
- **Basic Latency:**
  - 10-15 µs indexed reads.
  - 34-43 µs writes/overwrites of objects with one indexed attribute.
- **Multiple Secondary Indexes:**
  - 43-67 µs overwrites for objects with 1 to 10 indexes.
- **Scalability:**
  - Linear throughput:

    300 kops/sec –10 million ops/sec for 1 – 30 indexlets
- **Work in progress!**

Thank you!