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

Ankita Kejriwal

Stanford University

(With Arjun Gopalan, Ashish Gupta, Greg Hill, Zhihao Jia, Stephen Yang and John Ousterhout)

Introduction

- **RAMCloud 1.0 over a year ago**
- **Support 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**
  - 10-14 µs indexed reads
  - 29-37 µs writes/overwrites of objects with one indexed attribute.
- **Work in Progress!**

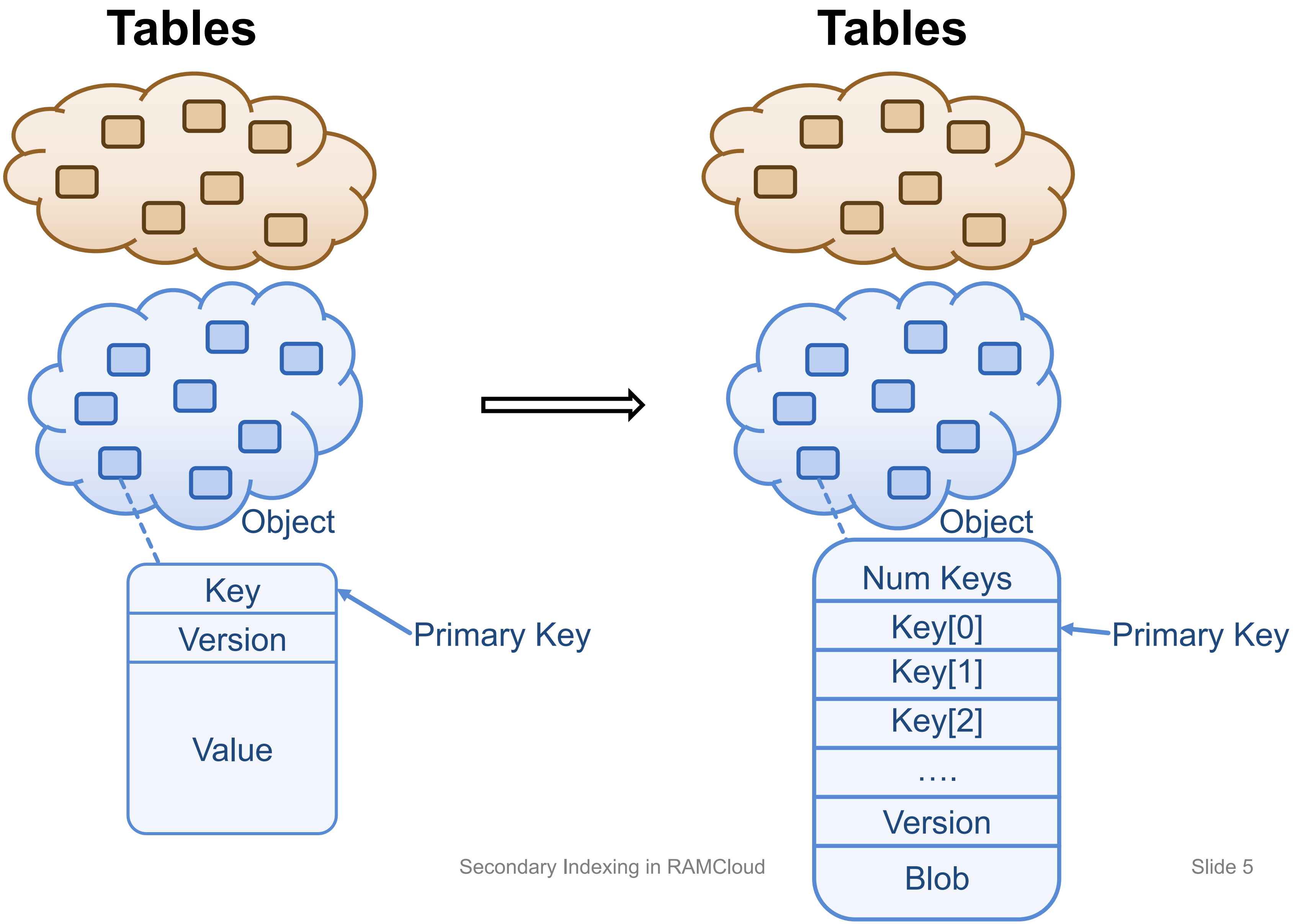
SLIK Status

- **SEDCL Context:**
  - Forum 2014: Design done, implementation underway
  - Retreat 2014: Basic implementation done, preliminary performance numbers
  - Forum 2015: Additional features, cleaner and faster code

Overview

- Object format and API
- Index memory allocation
- Failure / Restoration
- Index placement / partitioning
- Split and migrate index partition
- Consistency

Object Format and API



Tables

Object

Key

Version

Value

Primary Key

Tables

Object

Num Keys

Key[0]

Key[1]

Key[2]

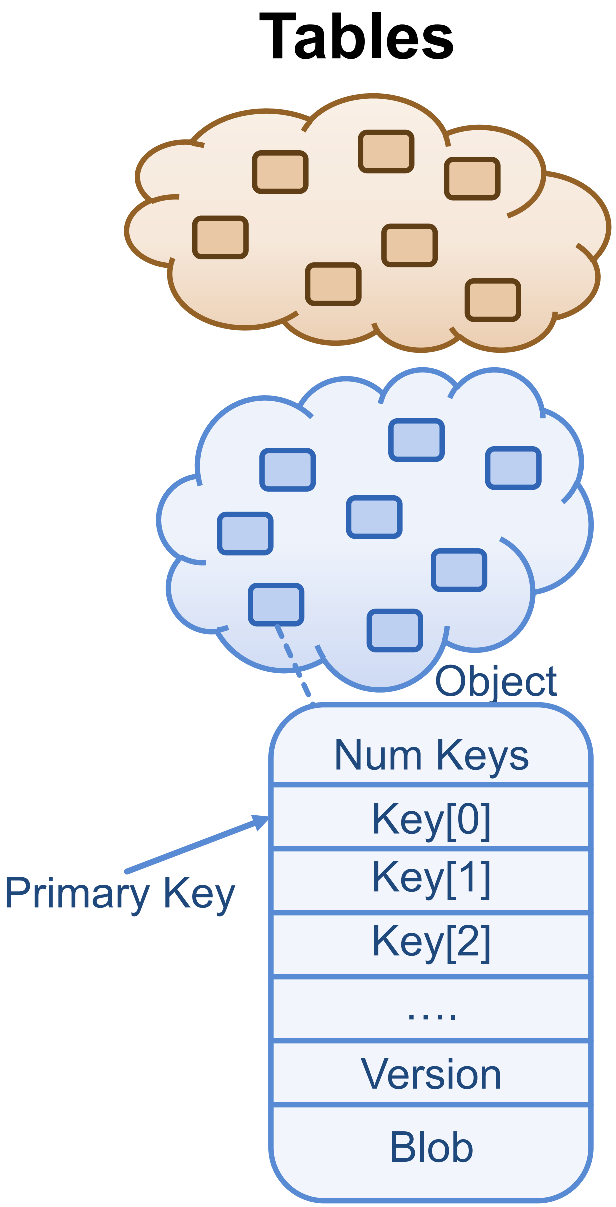
….

Version

Blob

Primary Key

Object Format and API



Num Keys

Key[0]

Key[1]

Key[2]

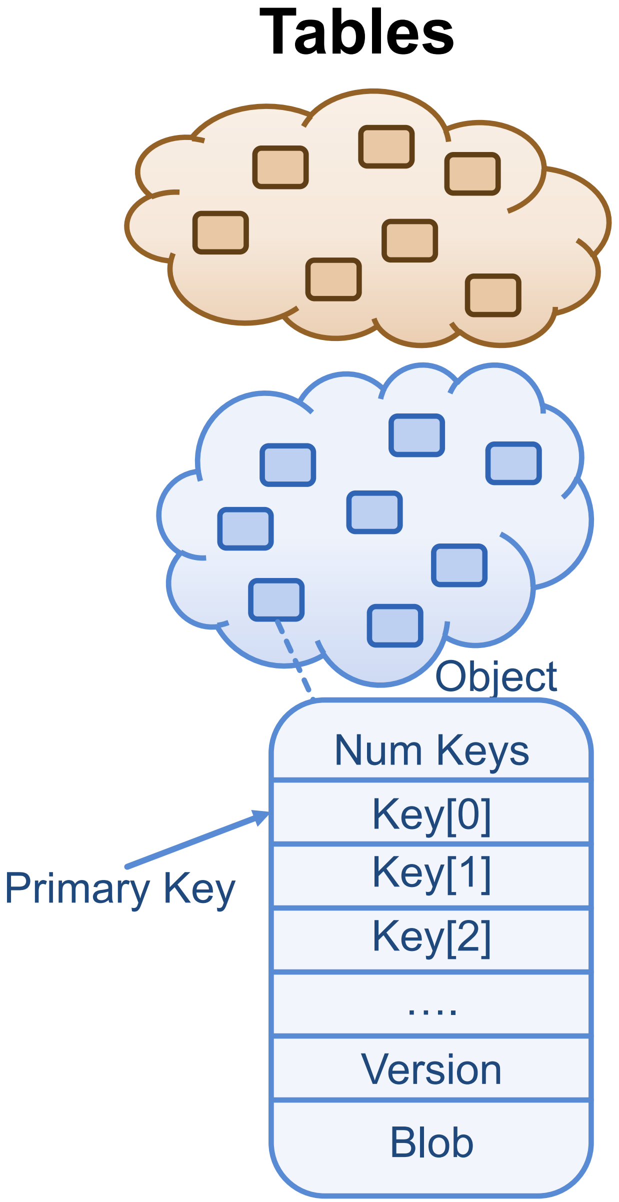
….

Version

Blob

```
createIndex(tableId, indexId,
                 indexType)
dropIndex(tableId, indexId)
```

Object Format and API



Tables

Object

Num Keys

Key[0]

Key[1]

Key[2]

....

Version

Blob

Primary Key

```
createIndex(tableId, indexId,
            indexType)
dropIndex(tableId, indexId)

write(tableId, keys, value)

readRange(tableId, indexId,
          firstKey, lastKey)
```

→ New streaming interface

- → Easier to use
- → Faster
- → Discovered consistency issue (eek!)

Index Memory Allocation

- **Index structured as Btree**
  - Originally: Open source Btree package (Panthema STX B+ Tree)
  - Now: In-house implementation
    - Allow variable sized keys
    - Simpler, more efficient code path
    - Efficient inserts
    - Approx 1 µs faster in both reads and writes!
- **Map tree nodes onto RAMCloud objects**

Failure / Restoration

- **Map tree nodes onto RAMCloud objects**

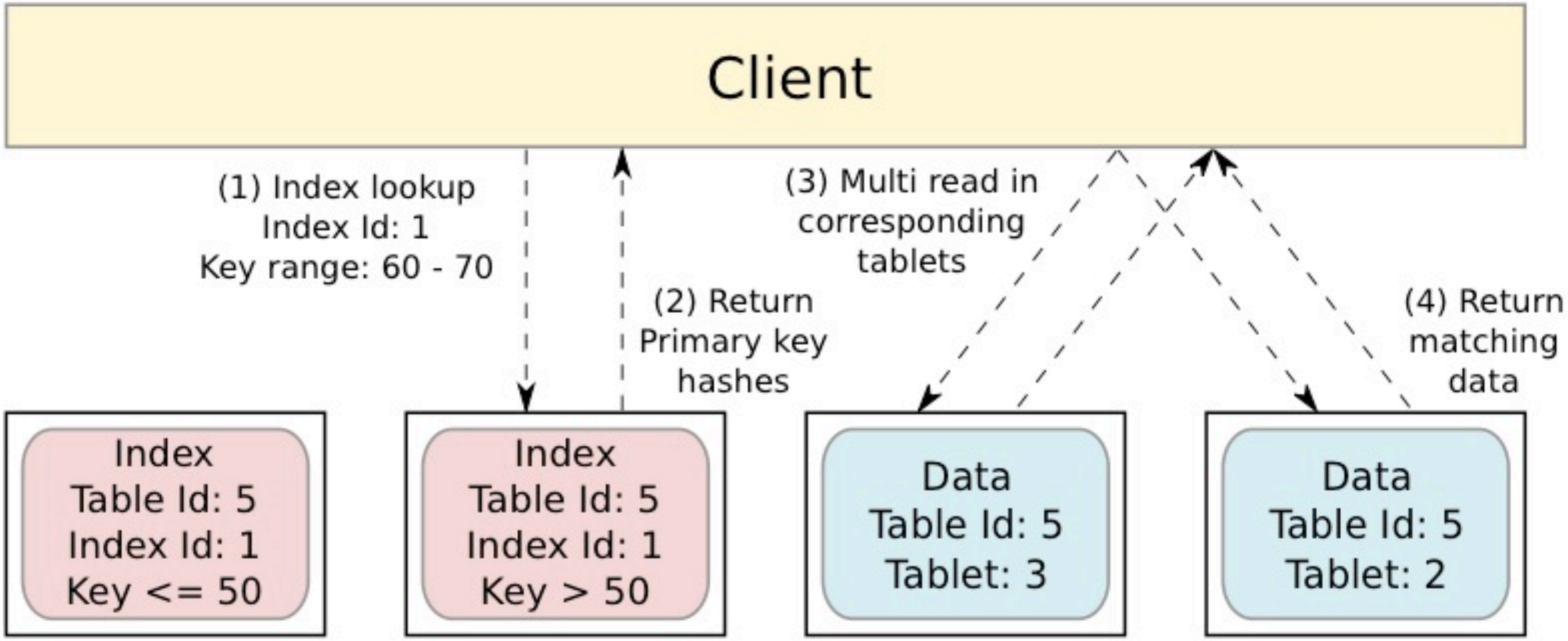
→ Index stored in RAMCloud log

→ Index crash recovery ~

(Pre-existing) Object Crash Recovery

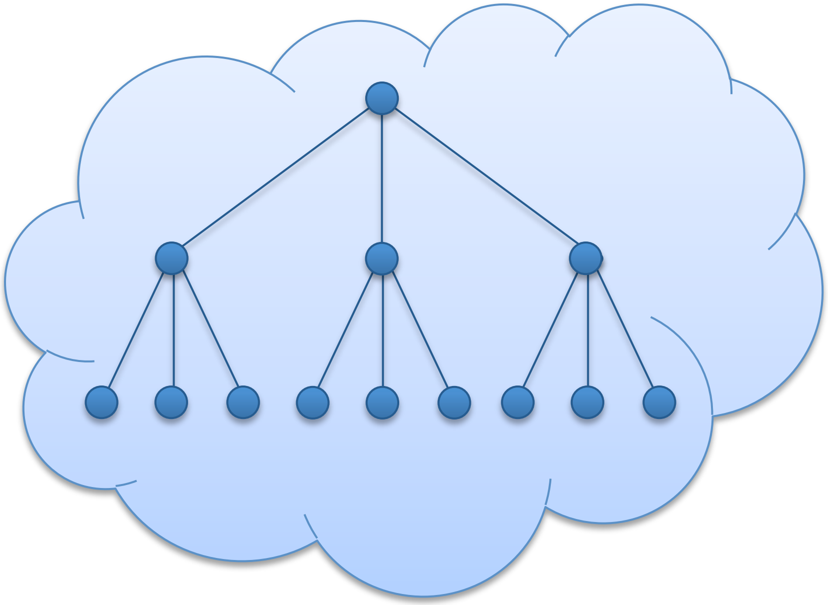
Index Placement / Partitioning

- Goal: Scalability
- Range Partitioning
- Distribute index and table independently



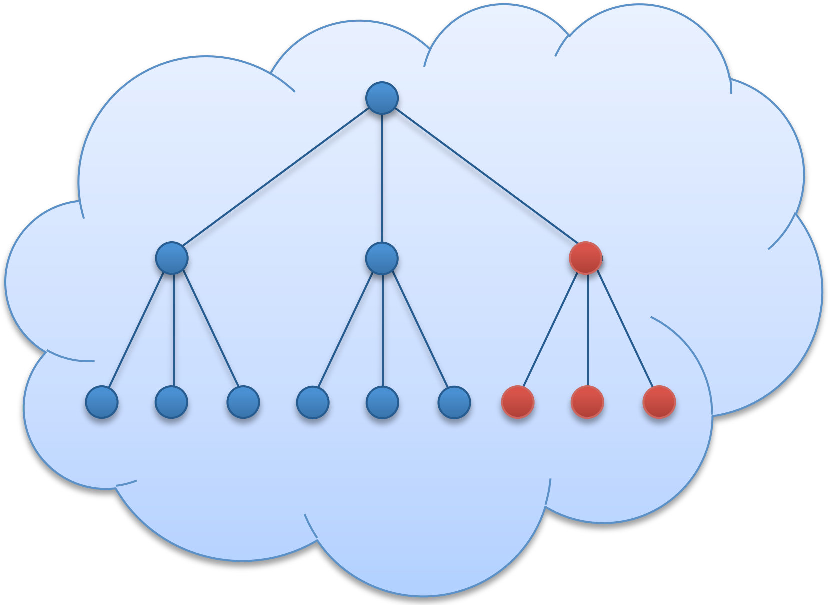
Split and Migrate Index Partition

- **Goals:**
  - Split an index partition
  - Migrate one of the resulting partitions to a different server
  - Allow concurrent reads/writes



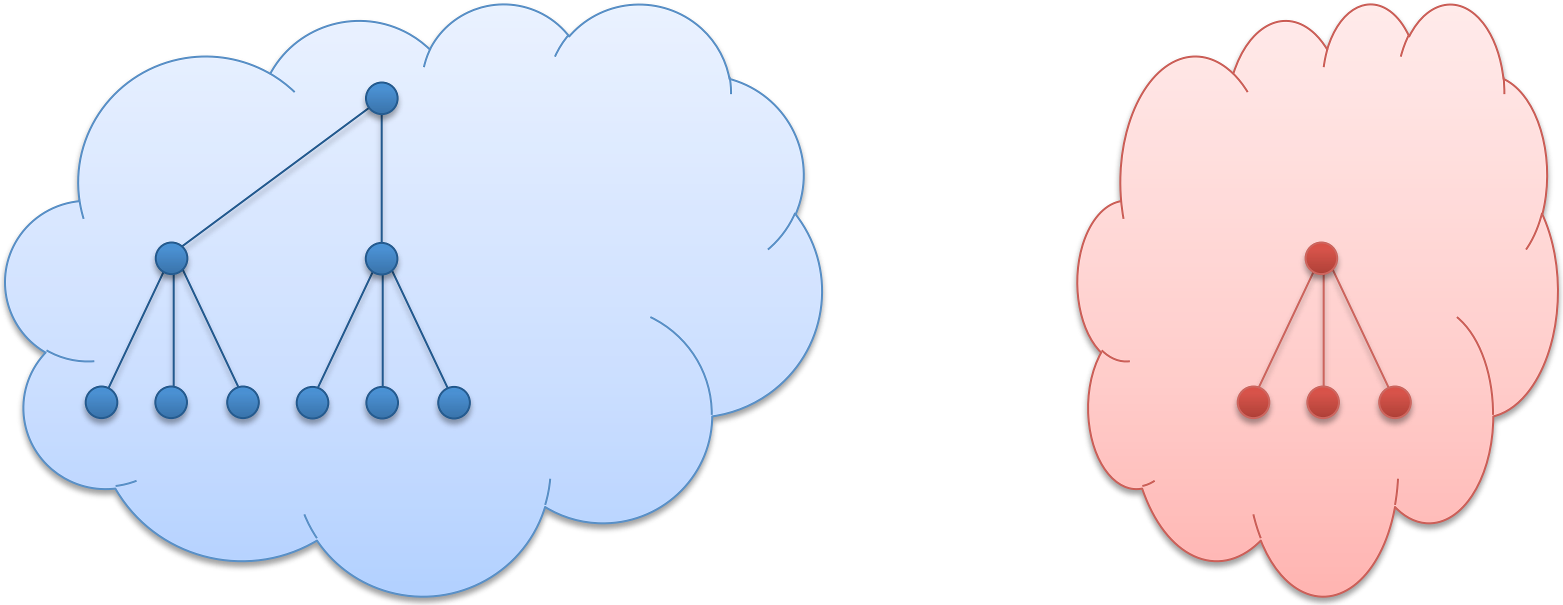
Split and Migrate Index Partition

- **Goals:**
  - Split an index partition
  - Migrate one of the resulting partitions to a different server
  - Allow concurrent reads/writes



Split and Migrate Index Partition

- **Goals:**
  - Split an index partition
  - Migrate one of the resulting partitions to a different server
  - Allow concurrent reads/writes

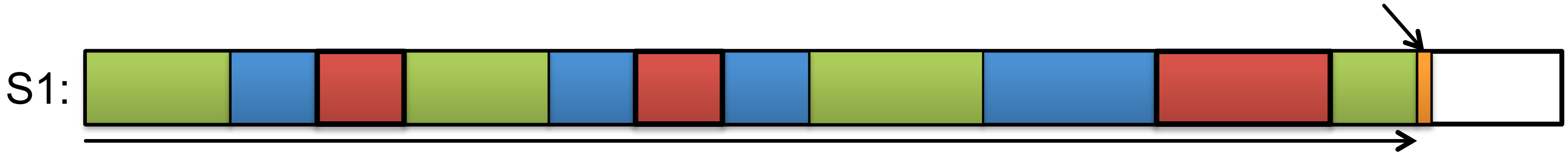


Split and Migrate Index Partition

- **Solution: Take advantage of RAMCloud Log Structure**
  - Example: S1: [A to Z] → S1: [A to L] and S2: [M to Z]

metadata:
S1: A to Z (state: OK => servicing requests)
S2: M to Z (state: Recovering => not servicing requests)

head of log



walk the log and migrate relevant objects to S2

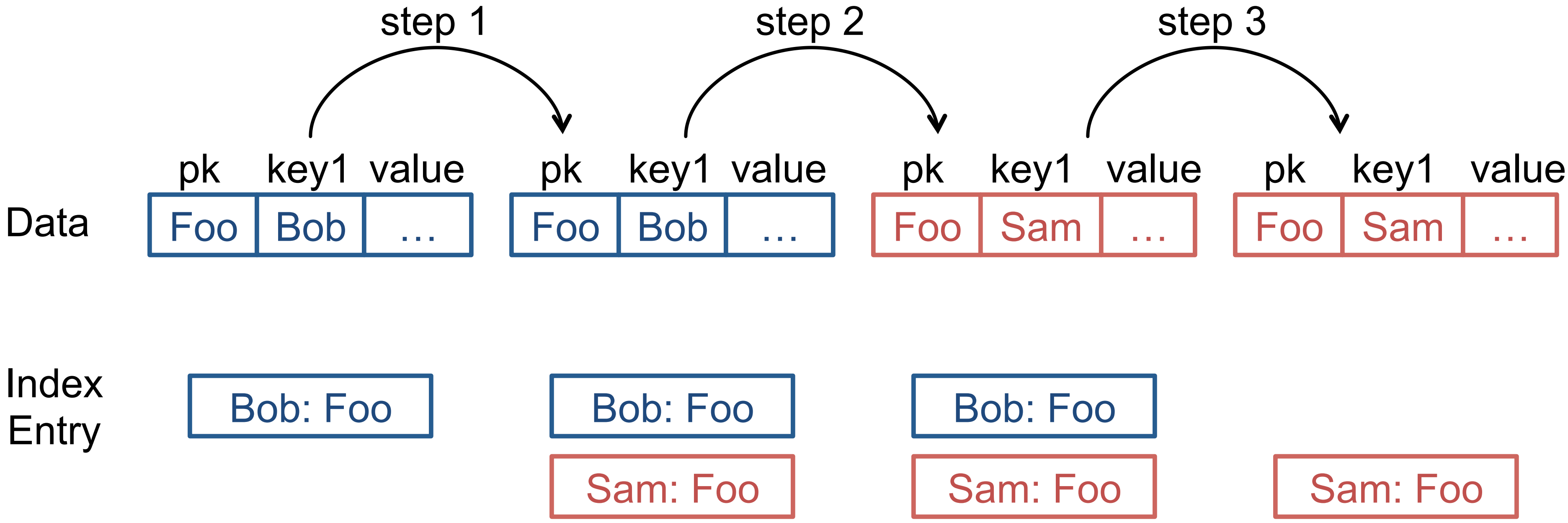
lock head
metadata: S1: A to L
unlock
send new data (if any)

metadata:
S1: A to L (state: OK)
S2: M to Z (state: OK)

t

Consistency

- **Indexed object writes: distributed operation**
- **Goal: Strong consistency**
- **Goal: Avoid transactions**
- **Solution:**
  - Longer index lifespan (via ordered writes)
  - Use object to determine index entry liveness (filter invalid index entries)



step 1 step 2 step 3

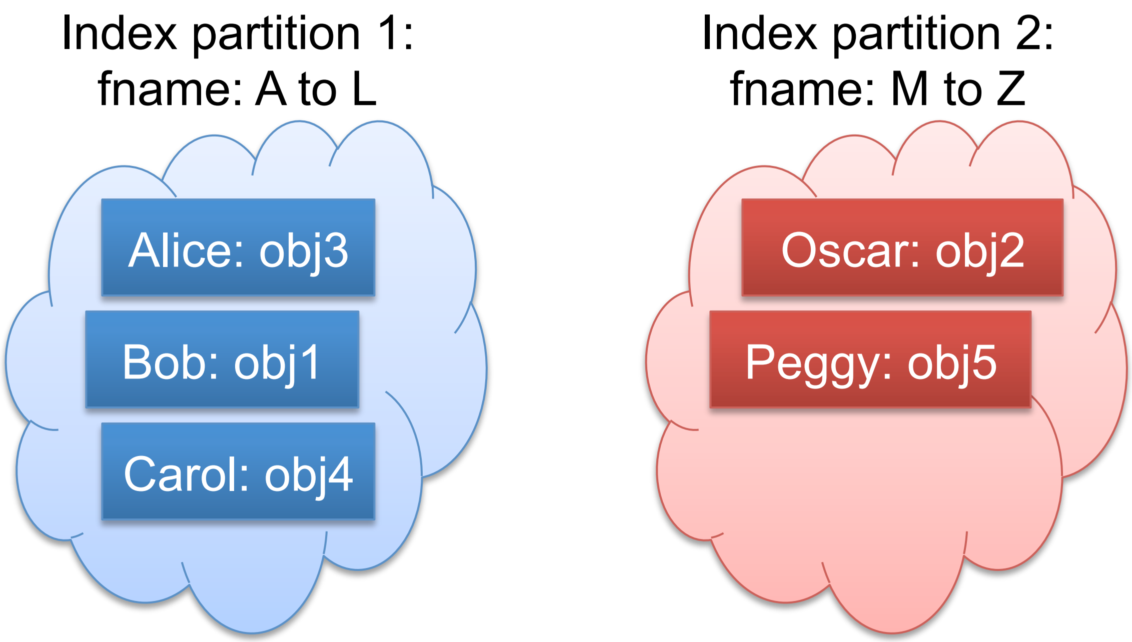
pk key1 value pk key1 value pk key1 value pk key1 value

Data Foo Bob … Foo Bob … Foo Sam … Foo Sam …

Index Entry Bob: Foo Bob: Foo Bob: Foo

Sam: Foo Sam: Foo Sam: Foo

Consistency Issue



Index partition 1:
fname: A to L

Alice: obj3

Bob: obj1

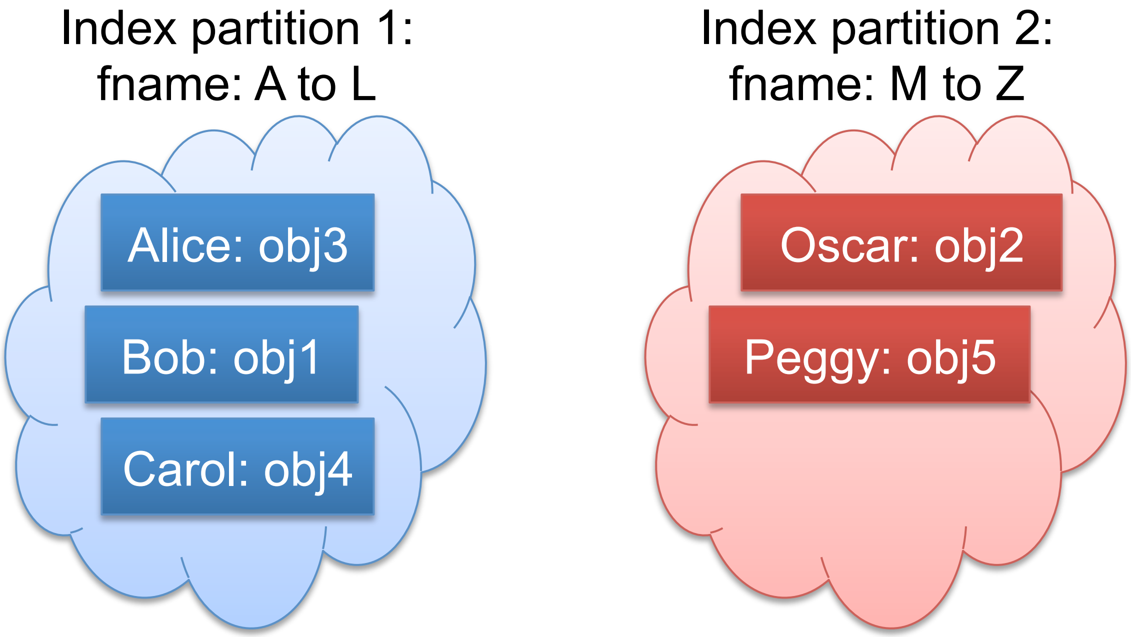
Carol: obj4

Index partition 2:
fname: M to Z

Oscar: obj2

Peggy: obj5

Consistency Issue



Index partition 1:
fname: A to L

Alice: obj3

Bob: obj1

Carol: obj4

Index partition 2:
fname: M to Z

Oscar: obj2

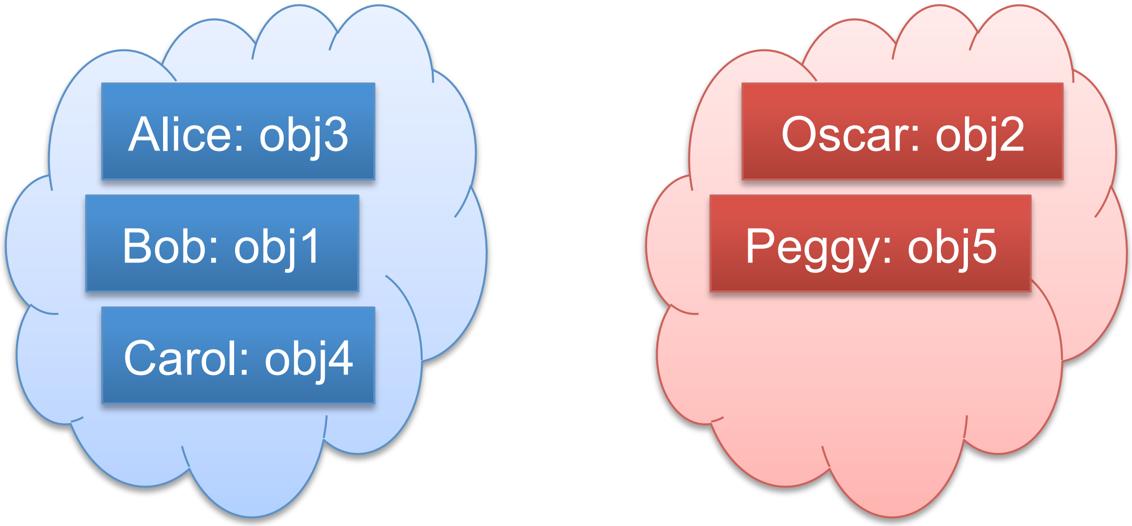
Peggy: obj5

Client 1: Streaming lookup: Find objects with fname between A and Z

Consistency Issue

Index partition 1:
fname: A to L

Index partition 2:
fname: M to Z



Alice: obj3

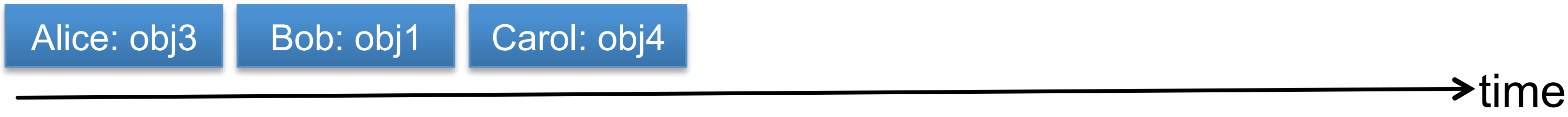
Bob: obj1

Carol: obj4

Oscar: obj2

Peggy: obj5

Client 1: Streaming lookup: Find objects with fname between A and Z



Alice: obj3

Bob: obj1

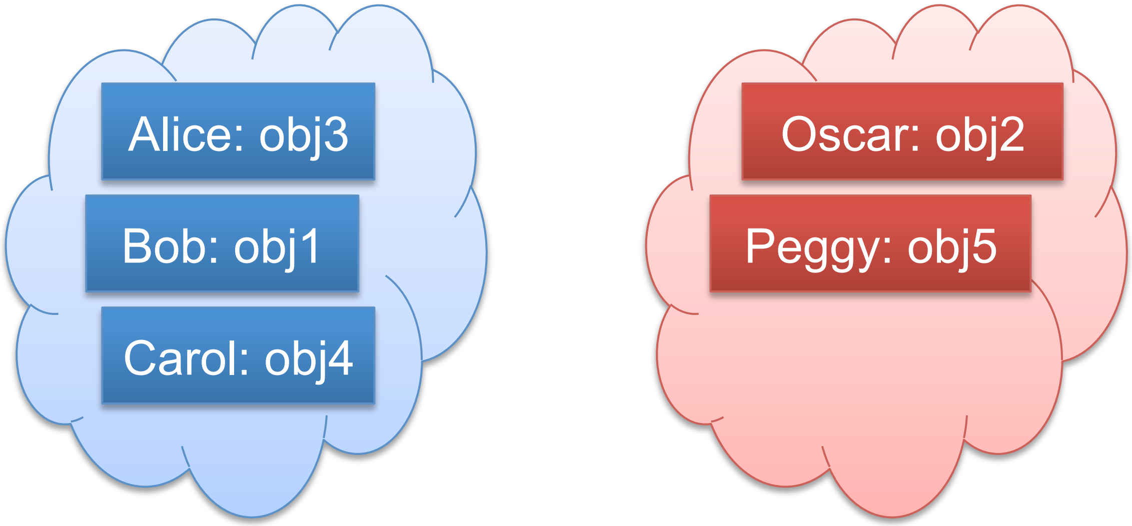
Carol: obj4

time

Consistency Issue

Index partition 1:
fname: A to L

Index partition 2:
fname: M to Z



Alice: obj3

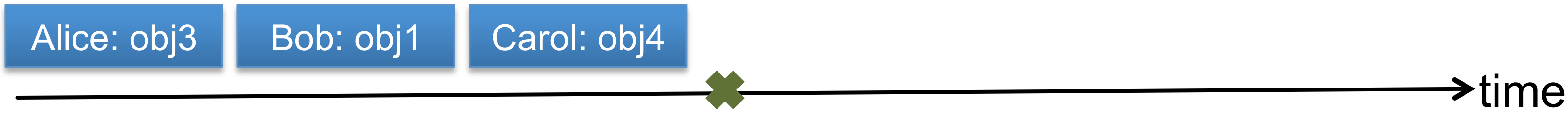
Bob: obj1

Carol: obj4

Oscar: obj2

Peggy: obj5

Client 1: Streaming lookup: Find objects with fname between A and Z



Alice: obj3

Bob: obj1

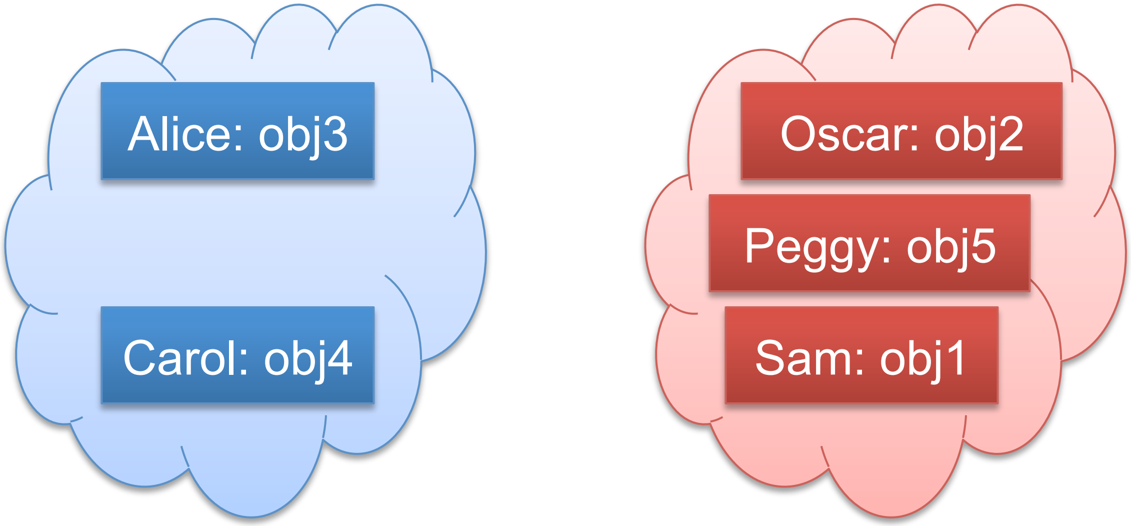
Carol: obj4

Client 2: Modify fname for obj1 to Sam

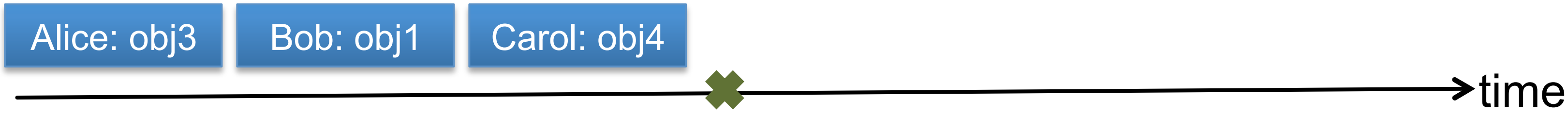
Consistency Issue

Index partition 1:
fname: A to L

Index partition 2:
fname: M to Z

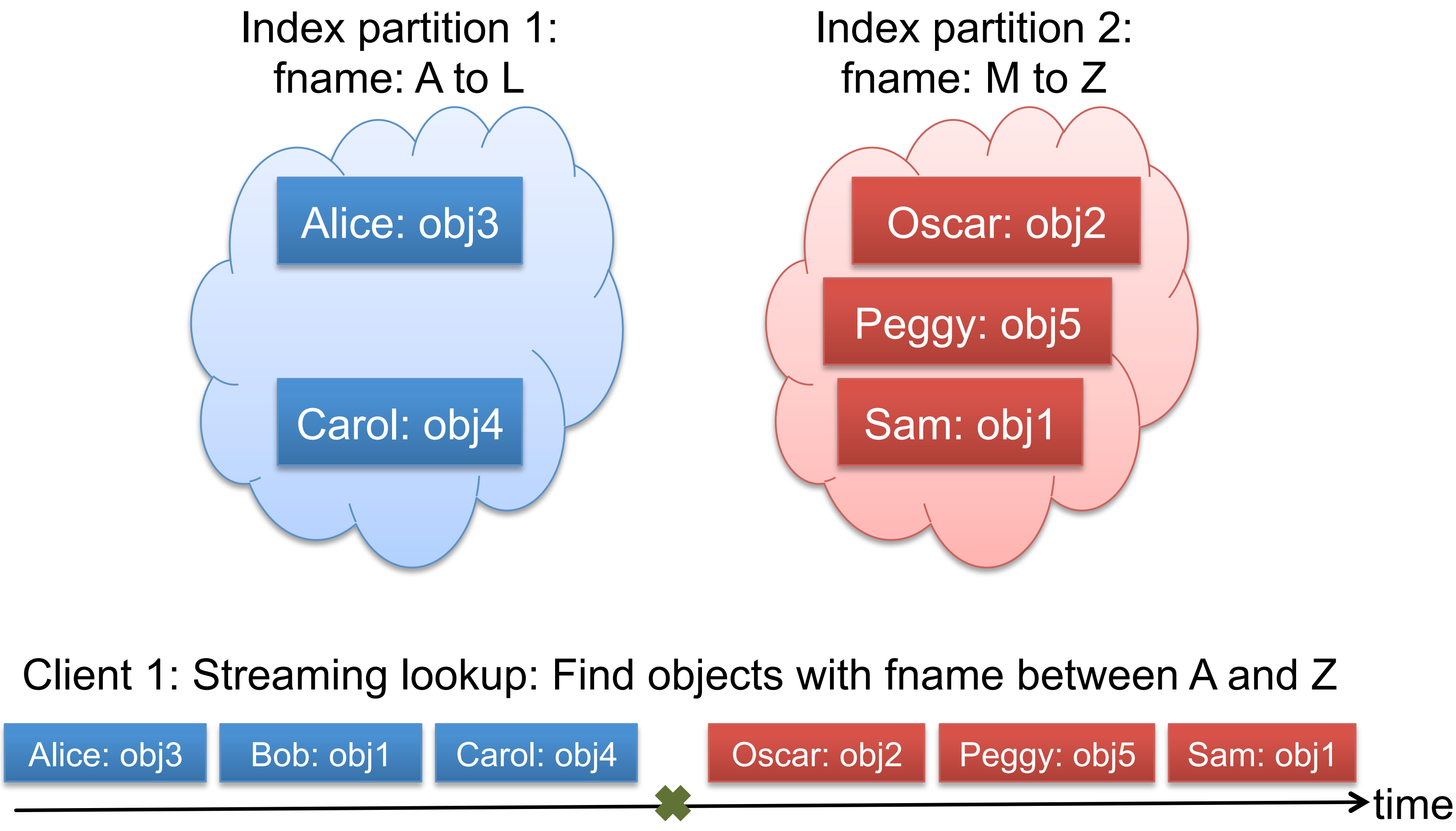


Client 1: Streaming lookup: Find objects with fname between A and Z



Client 2: Modify fname for obj1 to Sam

Consistency Issue



Index partition 1:
fname: A to L

Alice: obj3

Carol: obj4

Index partition 2:
fname: M to Z

Oscar: obj2

Peggy: obj5

Sam: obj1

Client 1: Streaming lookup: Find objects with fname between A and Z

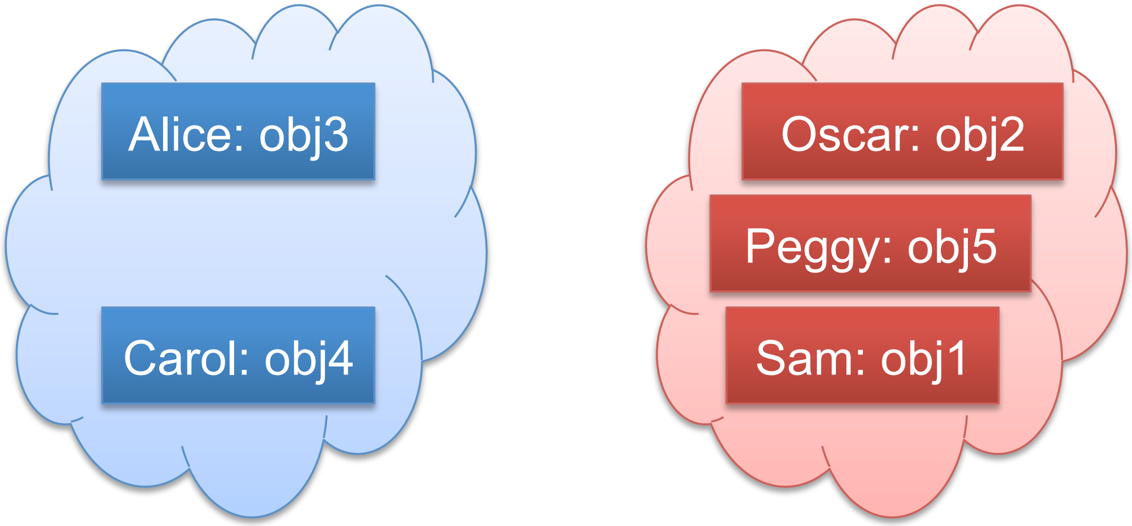
Alice: obj3 | Bob: obj1 | Carol: obj4 | Oscar: obj2 | Peggy: obj5 | Sam: obj1

time

Consistency Issue

Index partition 1:
fname: A to L

Index partition 2:
fname: M to Z



Alice: obj3

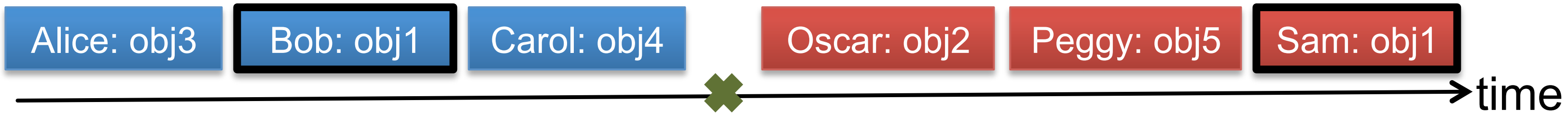
Carol: obj4

Oscar: obj2

Peggy: obj5

Sam: obj1

Client 1: Streaming lookup: Find objects with fname between A and Z



Alice: obj3

Bob: obj1

Carol: obj4

Oscar: obj2

Peggy: obj5

Sam: obj1

time

Client 1 sees obj1 twice!

Consistency Issue

- Streaming lookup with concurrent writes can cause consistency issues
  - Client can see an object multiple times
  - Client can miss an object
- Looking for a solution
  - Nothing simple and scalable so far
  - Ideas?
- Consistency and scale at odds with each other, after all?

Summary

- **SLIK: lookups & range queries (new streaming interface) on secondary keys**
- **Current performance:**
  - 10-14 µs indexed reads (1 µs improvement since retreat ‘14)
  - 29-37 µs writes of objects with one index attribute (>5 µs)
  - 33-49 µs writes/overwrites for objects with 1-10 indexes (>10 µs)
- **Strong-ish consistency (tradeoff with scalability)**
- **Ability to split and migrate partitions while allowing concurrent operations**
- **Work in progress**

Thank you!