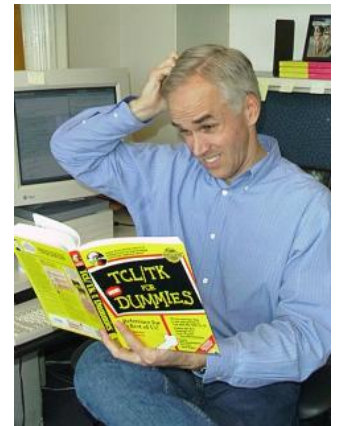


Secondary Indexing in RAMCloud

Ankita Kejriwal
Stanford University

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



Introduction

- **RAMCloud 1.0**
- **Higher-level data models**
 - Without sacrificing latency and scalability
- **Secondary Indexes: lookups and range queries on attributes that are not the primary key**
- **Feedback welcome!**

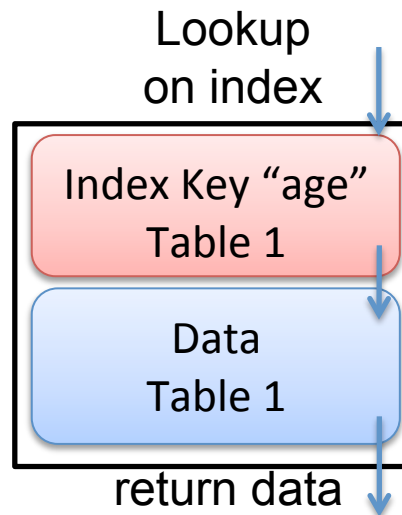
Key Design Issues

- API and RAMCloud object format
- **Index placement / partitioning**
- Index memory allocation
- **Failure / Restoration**
- **Consistency**

Key Design Issues

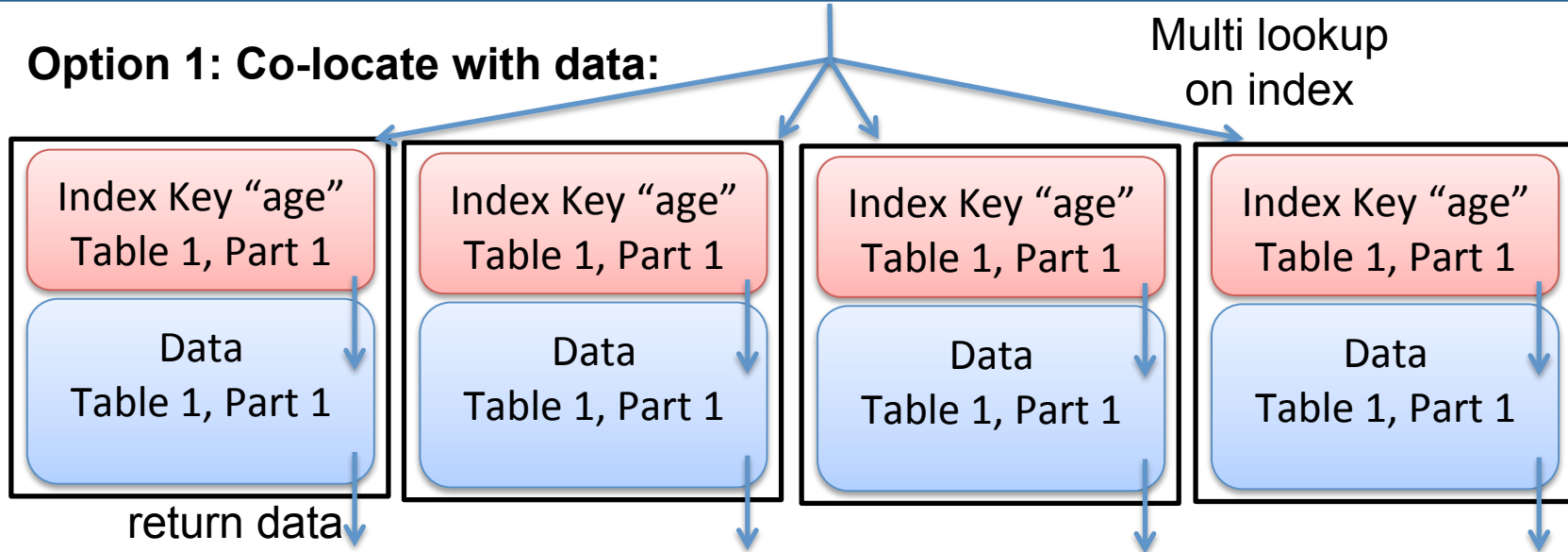
- API and RAMCloud object format
- ➔ ● **Index placement / partitioning**
- Index memory allocation
- **Failure / Restoration**
- **Consistency**

Index Placement

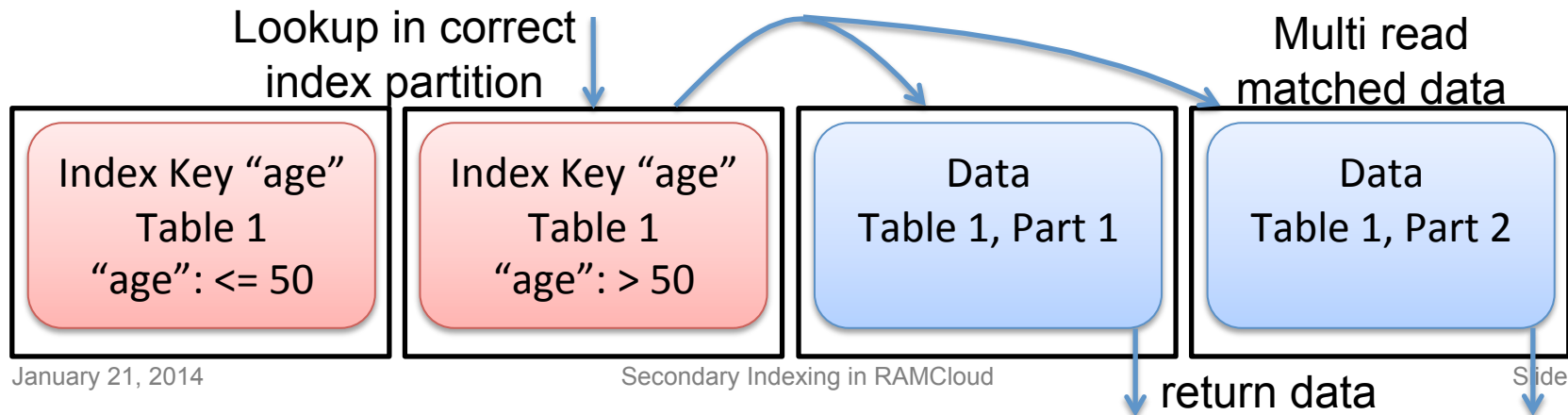


Index Partitioning

Option 1: Co-locate with data:



Option 2: Partition based on index key:



Index Partitioning

- **Index lookup:**

- Assume data + index on n servers
- Opt 1: multiLookup to n servers + local reads
- Opt 2: lookup to index server + multiRead to x servers
 - $x \in [0, n-1]$
 - For small n : expect $x \approx n-1$
 - For large n : expect $x \ll n$
- Option 2 more scalable

- **Index entry format:**

- `<index key, primary key hash>`

Key Design Issues

- API and RAMCloud object format
- **Index placement / partitioning**
- Index memory allocation
- ➔ ● **Failure / Restoration**
- **Consistency**

Failure / Restoration

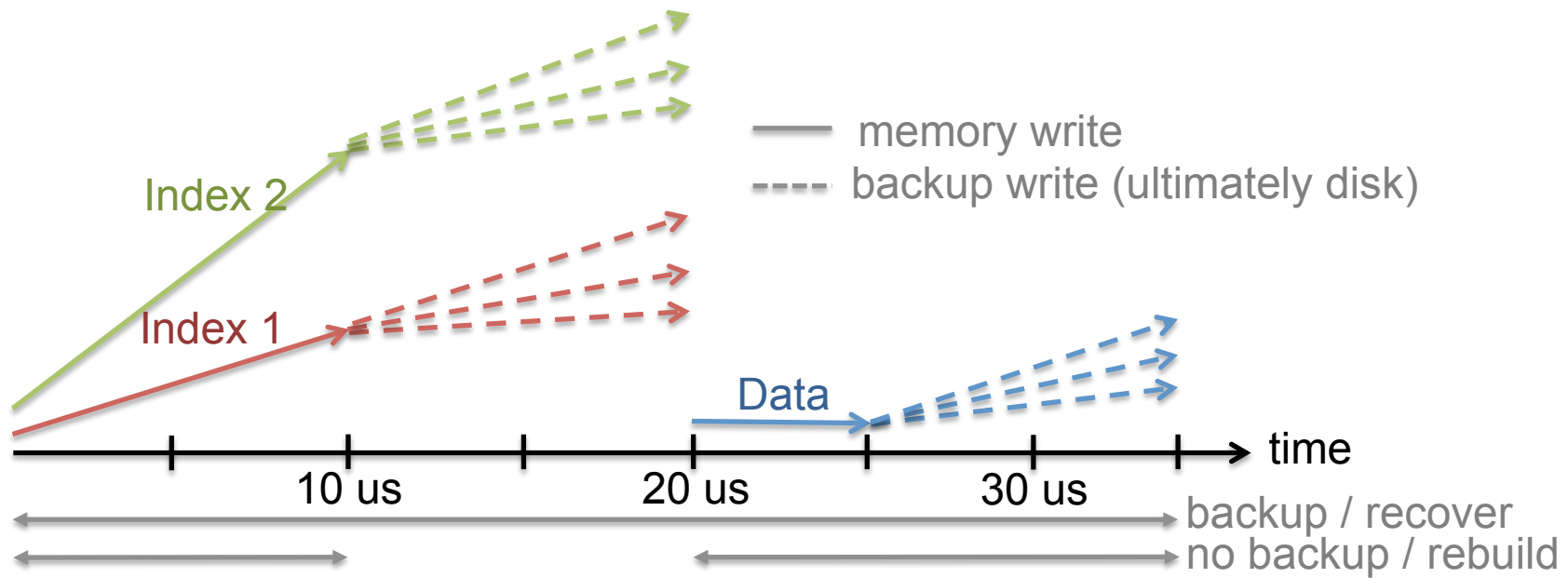
- **Tablet Server**

- Doesn't affect indexes
- “Normal” RAMCloud data recovery

- **Index server**

- Backup / Recover
- No backup / Rebuild

Failure/Restoration: Write Latency



	Latency	# Mem writes	# Backup writes	# Msgs from data to index servers	# Msgs to backups
No indexing	15 us	1	R	0	R
Indexing w/ backup/restore	35 us	$m+1$	$R*(m+1)$	m	$R*(m+1)$
Indexing w/ no-backup/rebuild	25 us	$m+1$	R	m	R

Failure/Restoration: Restoration Time

- **Recovery: Similar to RAMCloud data recovery: 1-2 s**
- **Rebuild: Cost analysis:**

Setting	Index partition to be recovered	1 GB
	Size of index entries	50 B (42 for key + 8 for keyhash)
	Num of index entries	$2 * 10^7$
Data master	Max memory bandwidth	35 GB/s
	Memory bw with overheads	20 GB/s
	Hash table size (10% of total mem)	25 GB (for 256 GB machine)
	Time to scan hash table	1.25 s
	Time to compare hash info from bucket	negligible
	Num objects to check if all match	$2.5 * 10^9$ (for 100B objects)
	Cache miss time	$0.5 * 10^9$ cache miss / s
	Total cache miss time	5.12 s
Network	Bandwidth	1 GB/s
	Time to transfer over network	1 s
Index Recovery Master	Time per object to insert	1.5 us
	Total time to insert	30 s
	Total time to insert with parallelization	1 s

Failure/Restoration: Restoration Time

- **Recovery: Similar to RAMCloud data recovery: 1-2 s**
- **Rebuild: Cost analysis:**

Setting	Index partition to be recovered	1 GB
	Size of index entries	50 B (42 for key + 8 for keyhash)
	Num of index entries	$2 * 10^7$
Data master	Max memory bandwidth	35 GB/s
	Memory bw with overheads	20 GB/s
	Hash table size (10% of total mem)	25 GB (for 256 GB machine)
	Time to scan hash table	1.25 s
	Time to compare hash info from bucket	negligible
	Num objects to check if all match	$2.5 * 10^9$ (for 100B objects)
	Cache miss time	$0.5 * 10^9$ cache miss / s
	Total cache miss time	5.12 s
Network	Bandwidth	1 GB/s
	Time to transfer over network	1 s
Index Recovery Master	Time per object to insert	1.5 us
	Total time to insert	30 s
	Total time to insert with parallelization	1 s

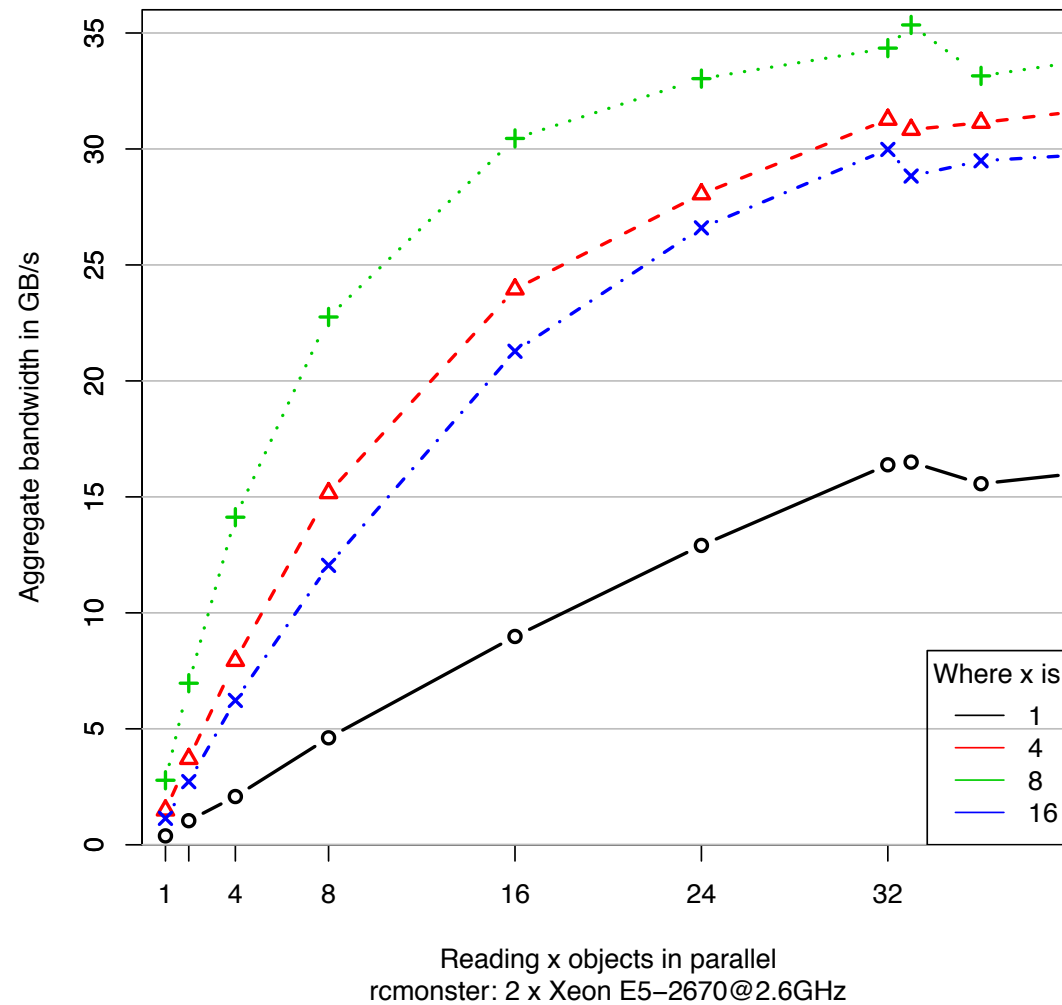
Failure/Restoration: Restoration Time

- **Recovery: Similar to RAMCloud data recovery: 1-2 s**
- **Rebuild: Cost analysis:**

Setting	Index partition to be recovered	1 GB
	Size of index entries	50 B (42 for key + 8 for keyhash)
	Num of index entries	$2 * 10^7$
Data master	Max memory bandwidth	35 GB/s
	Memory bw with overheads	20 GB/s
	Hash table size (10% of total mem)	25 GB (for 256 GB machine)
	Time to scan hash table	1.25 s
	Time to compare hash info from bucket	negligible
	Num objects to check if all match	$2.5 * 10^9$ (for 100B objects)
	Cache miss time	$0.5 * 10^9$ cache miss / s
	Total cache miss time	5.12 s
Network	Bandwidth	1 GB/s
	Time to transfer over network	1 s
Index Recovery Master	Time per object to insert	1.5 us
	Total time to insert	30 s
	Total time to insert with parallelization	1 s

Memory Benchmark

Random reads from array of $2 * 10^8$ objects of size 64 B on rcmonster



Key Design Issues

- API and RAMCloud object format
- **Index placement / partitioning**
- Index memory allocation
- **Failure / Restoration**
- ➔ ● **Consistency**

Consistency

- **At any time, data is consistent with index entries corresponding to it, if:**
 - If data X exists, X is reachable from all key indexes.
 - Data returned to client is consistent with key used to look it up.
- **Provides linearizability**
 - Tradeoff with performance
- **Also desirable:**
 - Dangling pointers are not accumulating.
 - Memory footprint will not increase beyond what is necessary.

Consistency

- **Simple solution:**

- Lock indexes and tablets for the entire duration of index update – affects scalability and performance

- **Our solution: Key Idea:**

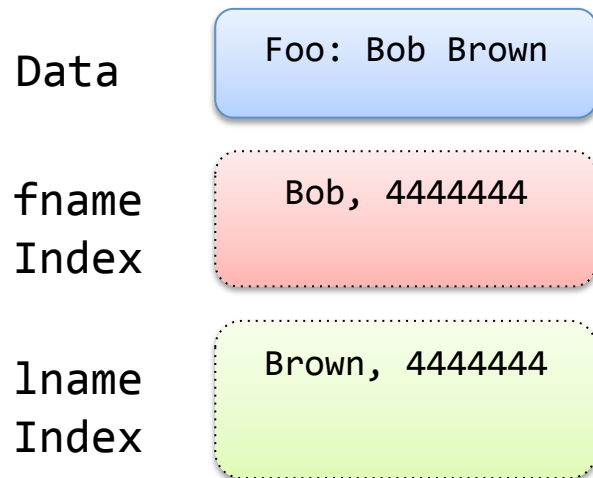
- Writing object is the commit point

- **Interesting situations:**

- For multi-threaded write/read, non-locking, no failures
- For multi-threaded write/write, non-locking, no failures
- Failure of an Index Server
- Failure of Master Server

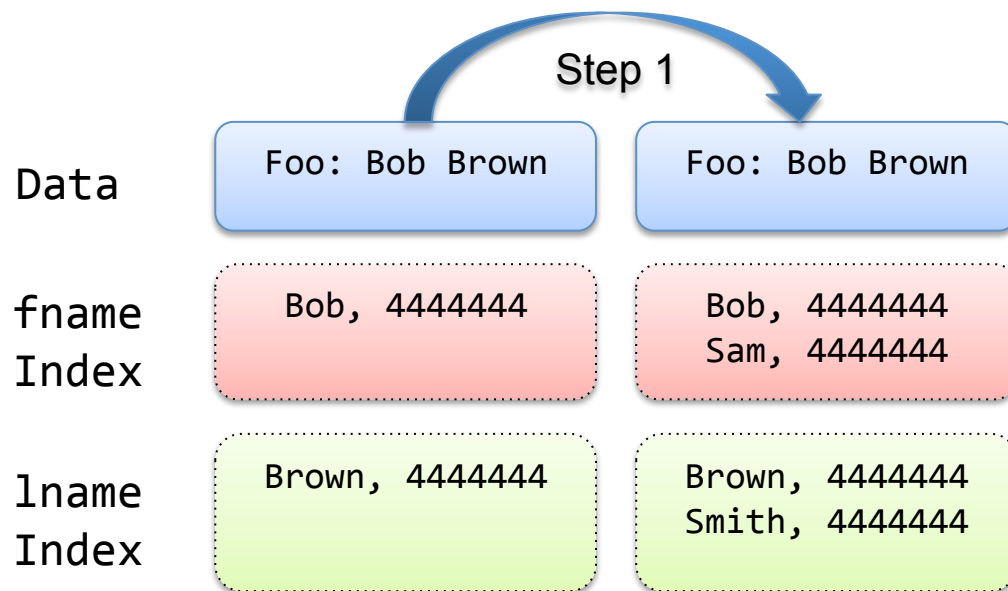
Consistency

- Multi-threaded write/read, non-locking, no failures: Object Update
- There exists time x , s.t.: at time $< x$, client can lookup old data; at time $\geq x$, it can lookup the new data.



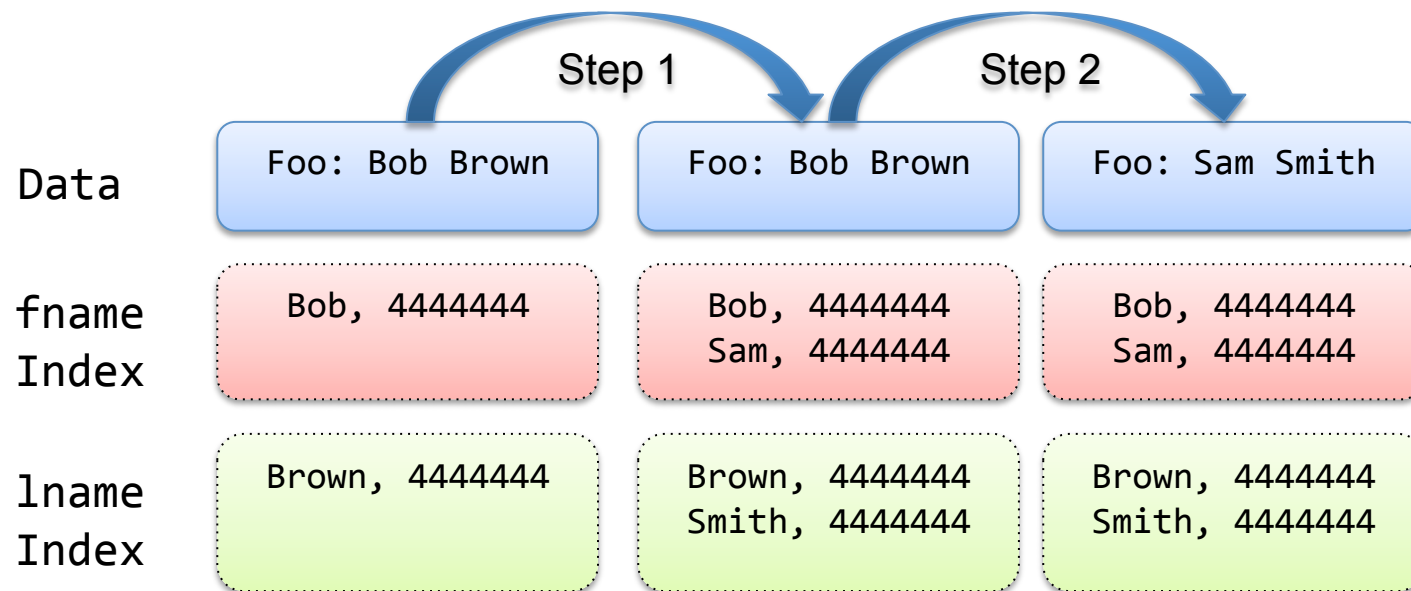
Consistency

- Multi-threaded write/read, non-locking, no failures: Object Update
- There exists time x , s.t.: at time $< x$, client can lookup old data; at time $\geq x$, it can lookup the new data.



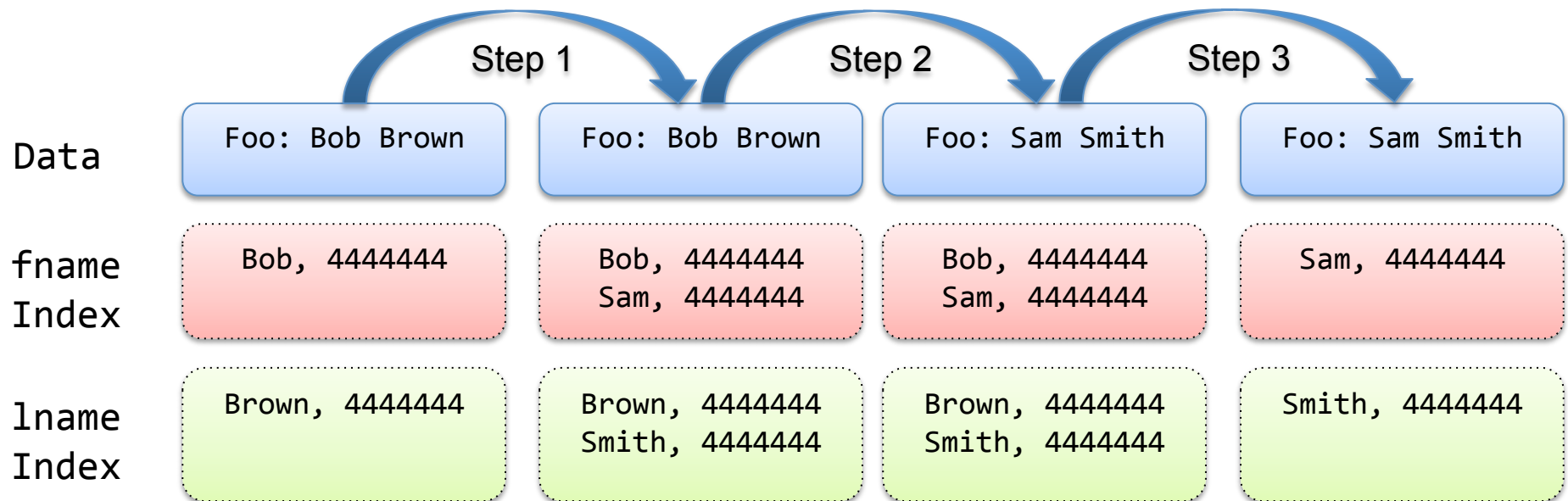
Consistency

- Multi-threaded write/read, non-locking, no failures: Object Update
- There exists time x , s.t.: at time $< x$, client can lookup old data; at time $\geq x$, it can lookup the new data.



Consistency

- Multi-threaded write/read, non-locking, no failures: Object Update
- There exists time x , s.t.: at time $< x$, client can lookup old data; at time $\geq x$, it can lookup the new data.



Summary

- **Secondary Indexes: lookups & range queries on attributes that are not the primary key**
- **Key design issues:**
 - Index partitioning
 - Co-locate with data
 - Partition based on index key
 - Failure / Restoration
 - Backup / recover
 - No backup / rebuild
 - Consistency: Linearizability
 - Key idea: Writing object is the commit point
- **Feedback welcome!**

Thank you!

