Amazon Dynamo

A Highly Available Key-value Store

Present by Jian Fang

jianf@cmu.edu

What is Dynamo

- Eventually consistent key-value store
- Support scalable highly available data access
- Optimized for availability to maximize customer satisfaction

Why not RDBMS?

- Only need primary-key access
- RDBMS have limited scalability
- RDBMS require expensive hardware and skillful administrators

Amazon’s Requirements

- Objects are less than 1MB
- No operations span for multiple data
- <300ms response time for 99.9% requests
- Heterogeneous commodity hardware infrastructure
- Decentralized, loosely coupled services
- Highly available(always writable)

Techniques used in Dynamo

- Consistent Hashing
- Vector clocks
- Sloppy Quorum and Hinted handoff
- Merkle trees
- Gossip-based membership protocol

Interfaces

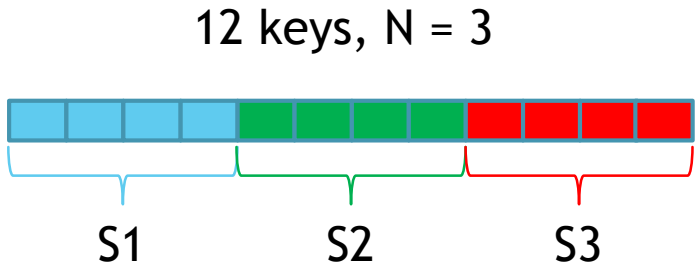
- Key-value storage system with operators:
  - Get(key): return a single or a list of objects with conflicting versions
  - Put(key, context, object): context contains the version information
- MD5 hashing is applied on the key to generate 128-bit identifier

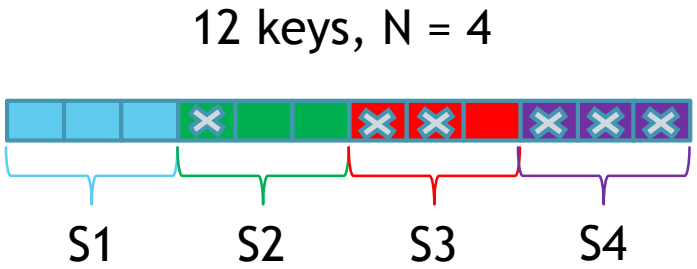
Partitioning

- Scale Incrementally
- Consistent Hashing
- Variant of Consistent Hashing

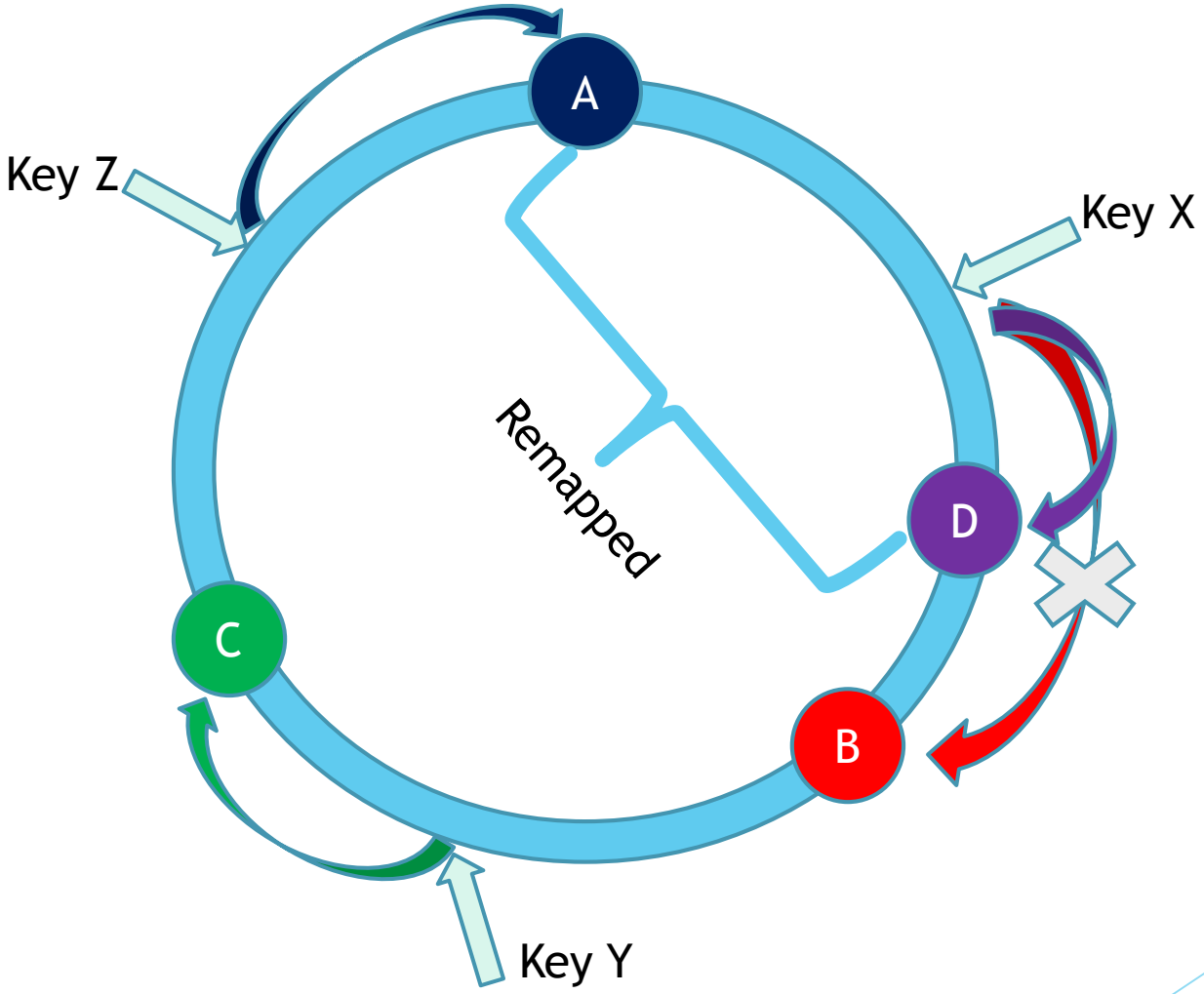
Consistent Hashing

- Simple Non-Consistent Hashing
  - $Hash(key)\ mod\ N$
  - What if N = N + 1
  - 6 keys(a half) remapped
- Consistent Hashing
  - Only K/N keys need to be remapped





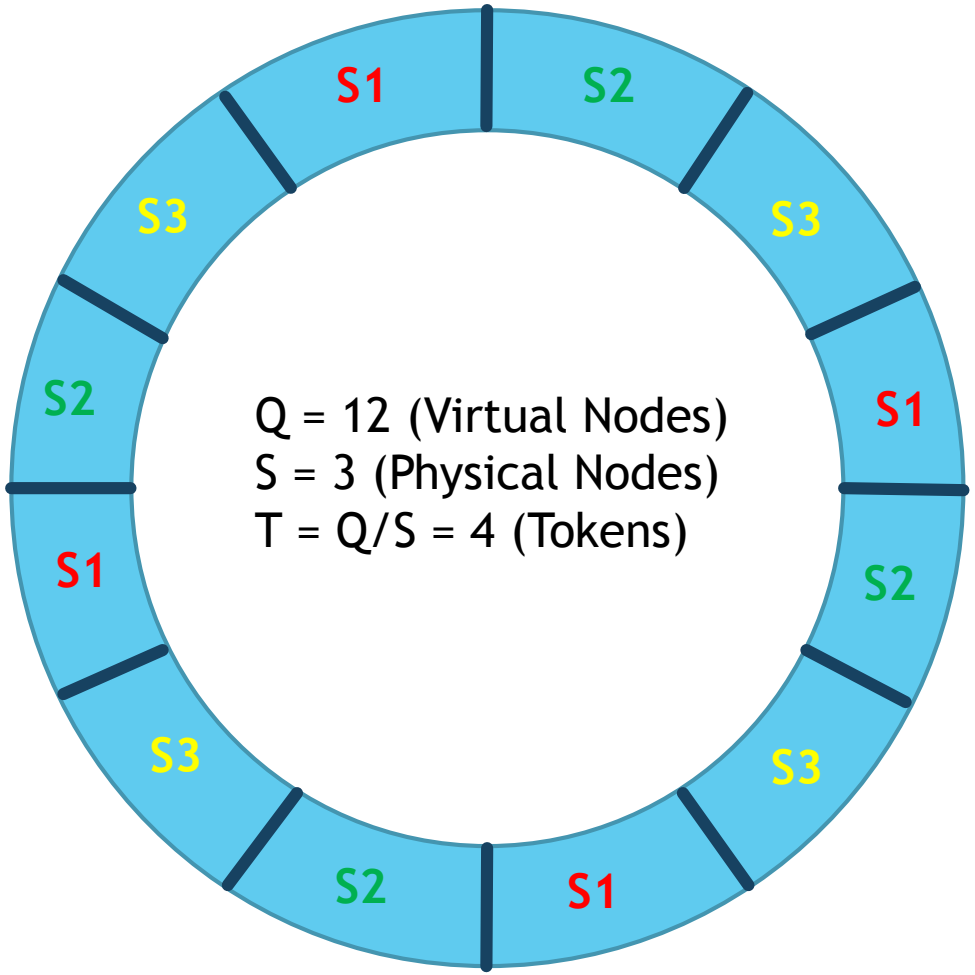
Consistent Hashing



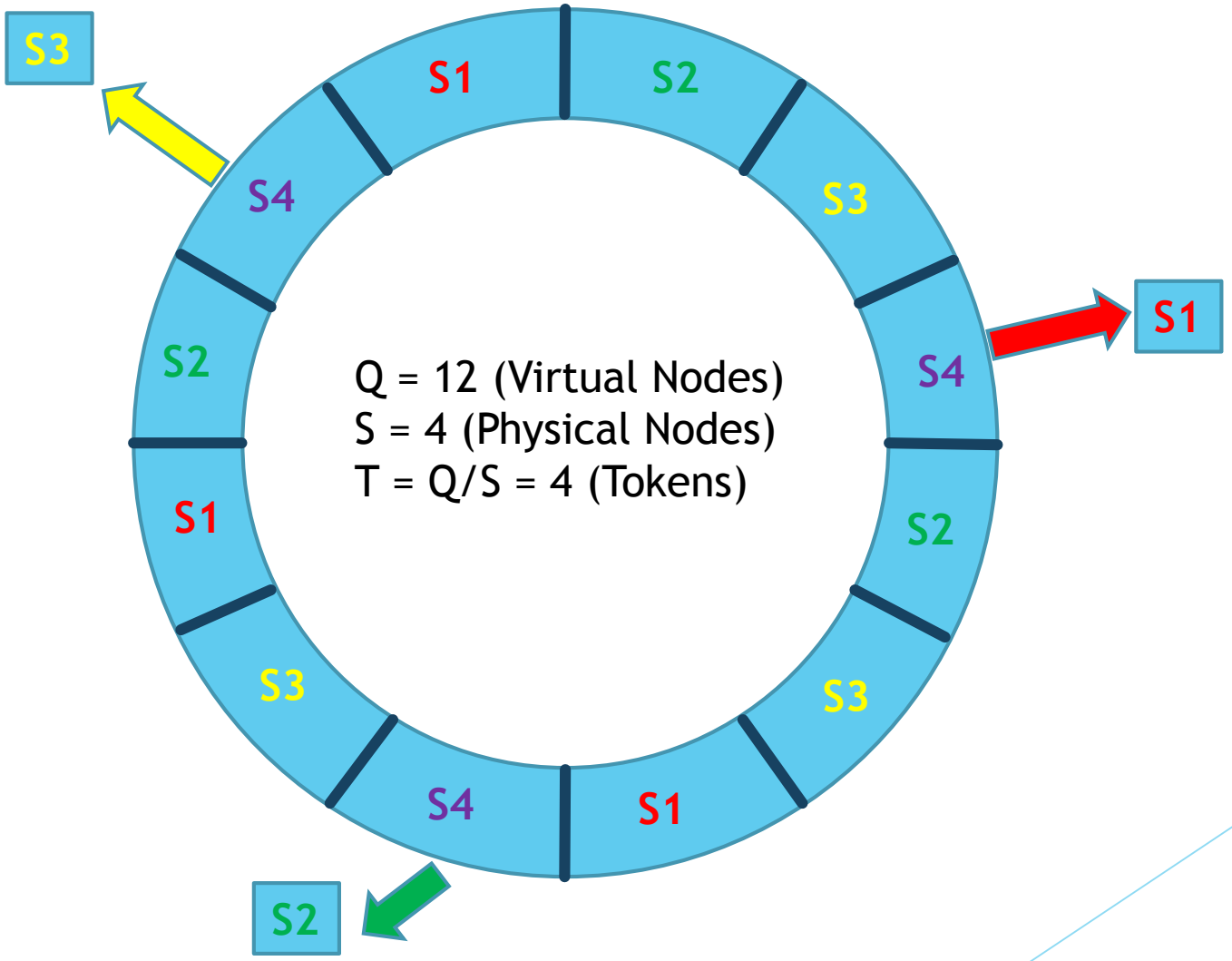
Consistent Hashing

- Not good enough
  - Non-uniform load distribution
  - No heterogeneity in node's performance
- Variant of Consistent Hashing
  - Virtual Nodes

Variant of Consistent Hashing

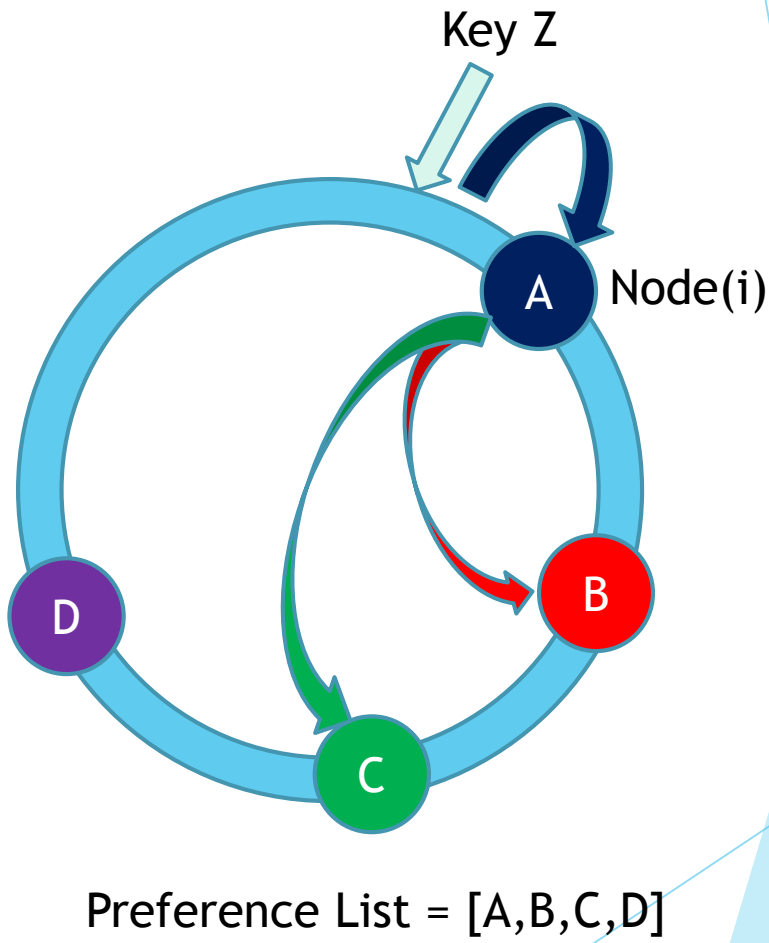


Variant of Consistent Hashing



Replication

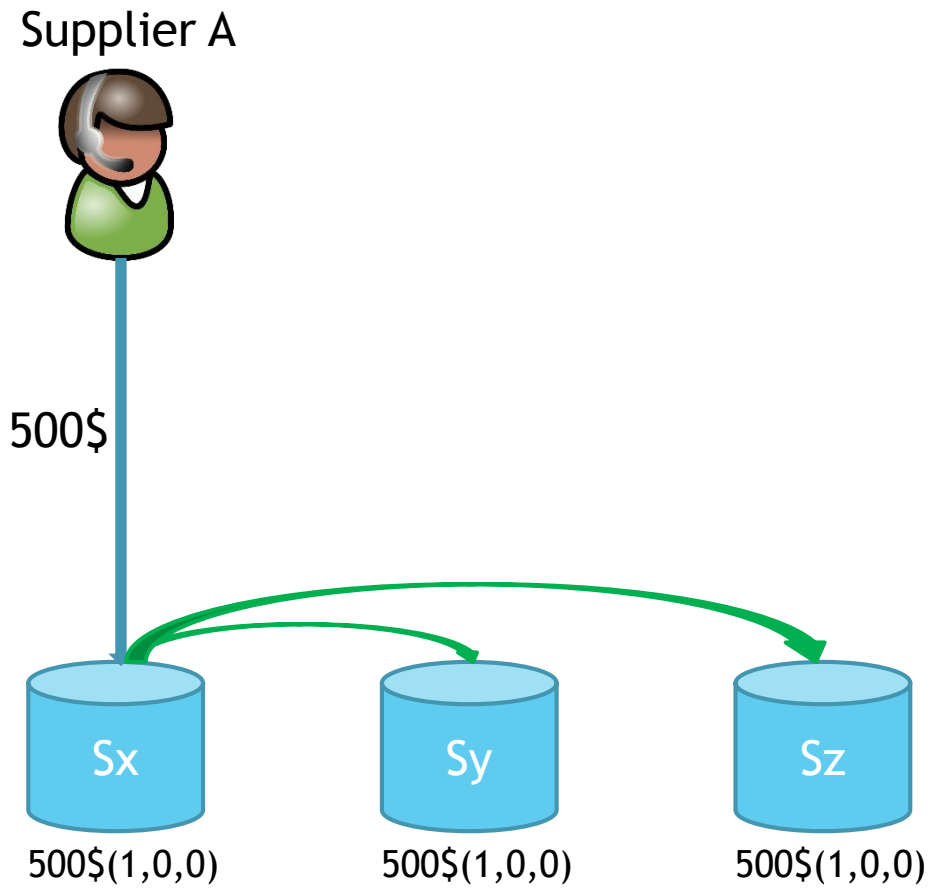
- A coordinator Node(i)
- (N-1) clockwise successor nodes as replicas
- Node(i) update all other (N-1) replicas
- A preference list of nodes
  - List size > N



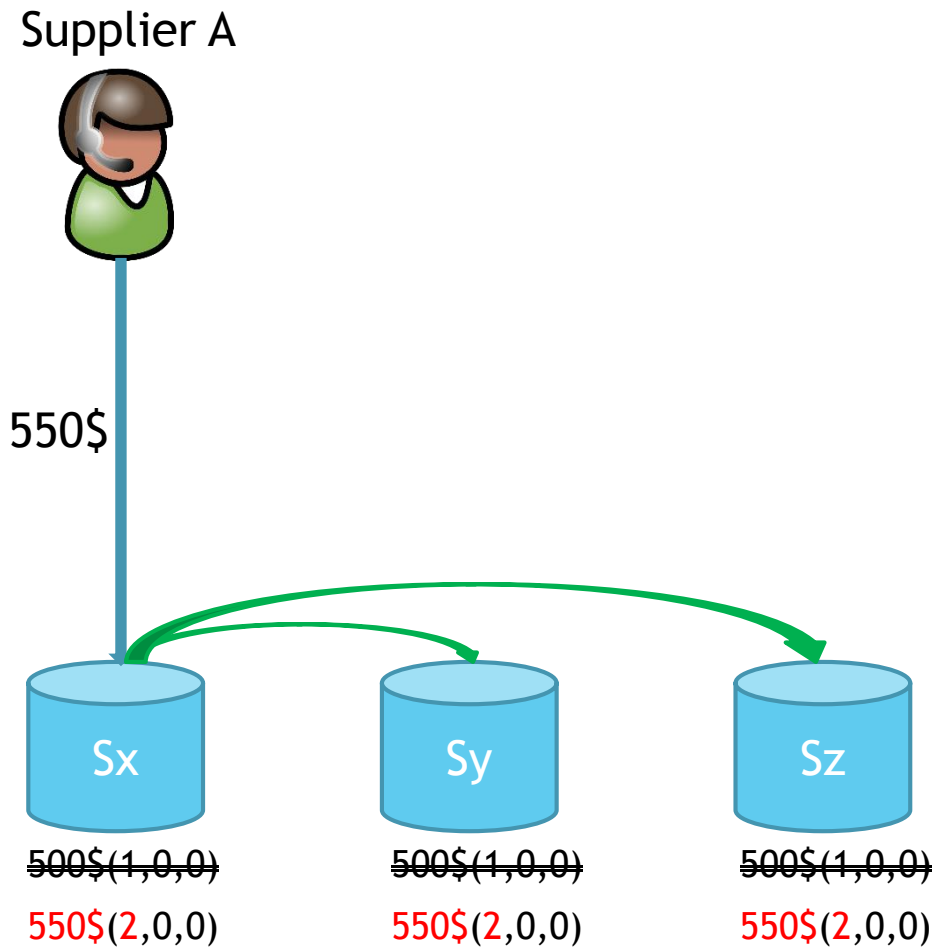
Data Versioning

- Eventual Consistency
- Put() is returned before updating all replicas
- Get() can return multiple versions for the same key
- Data mutation as new version
- Vector Clock

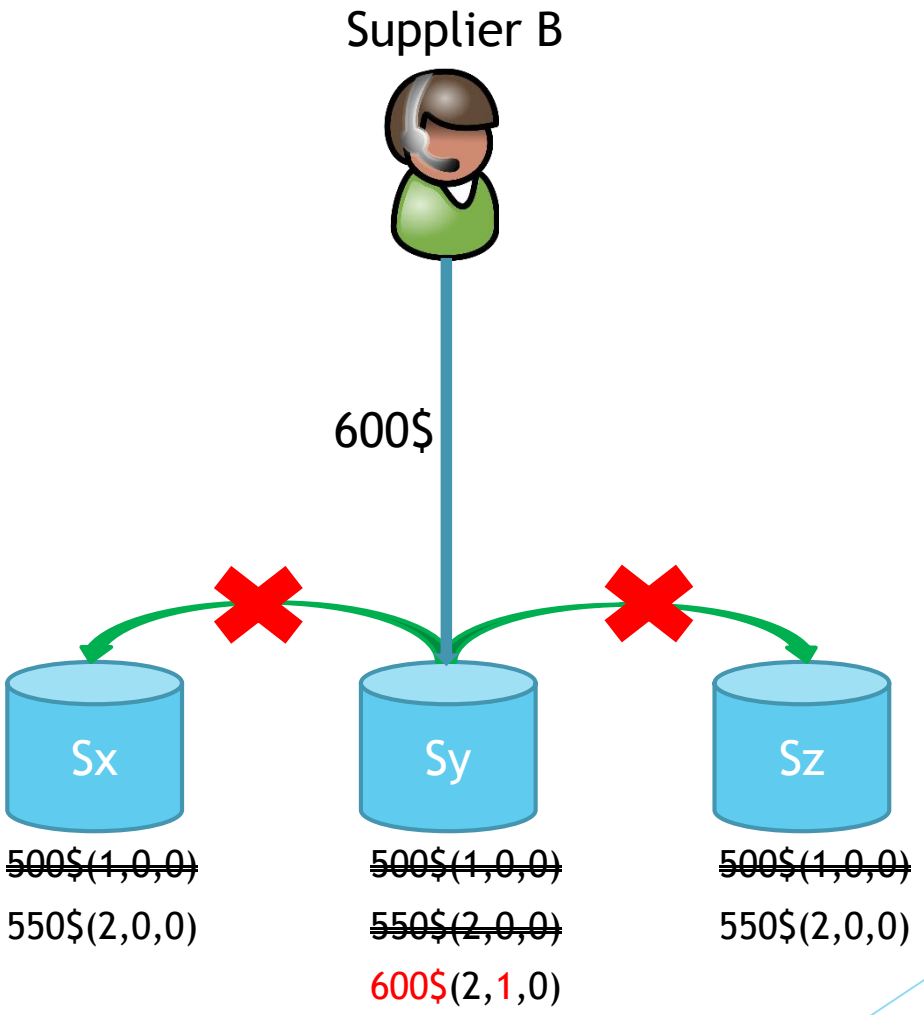
Vector Clock(Example)



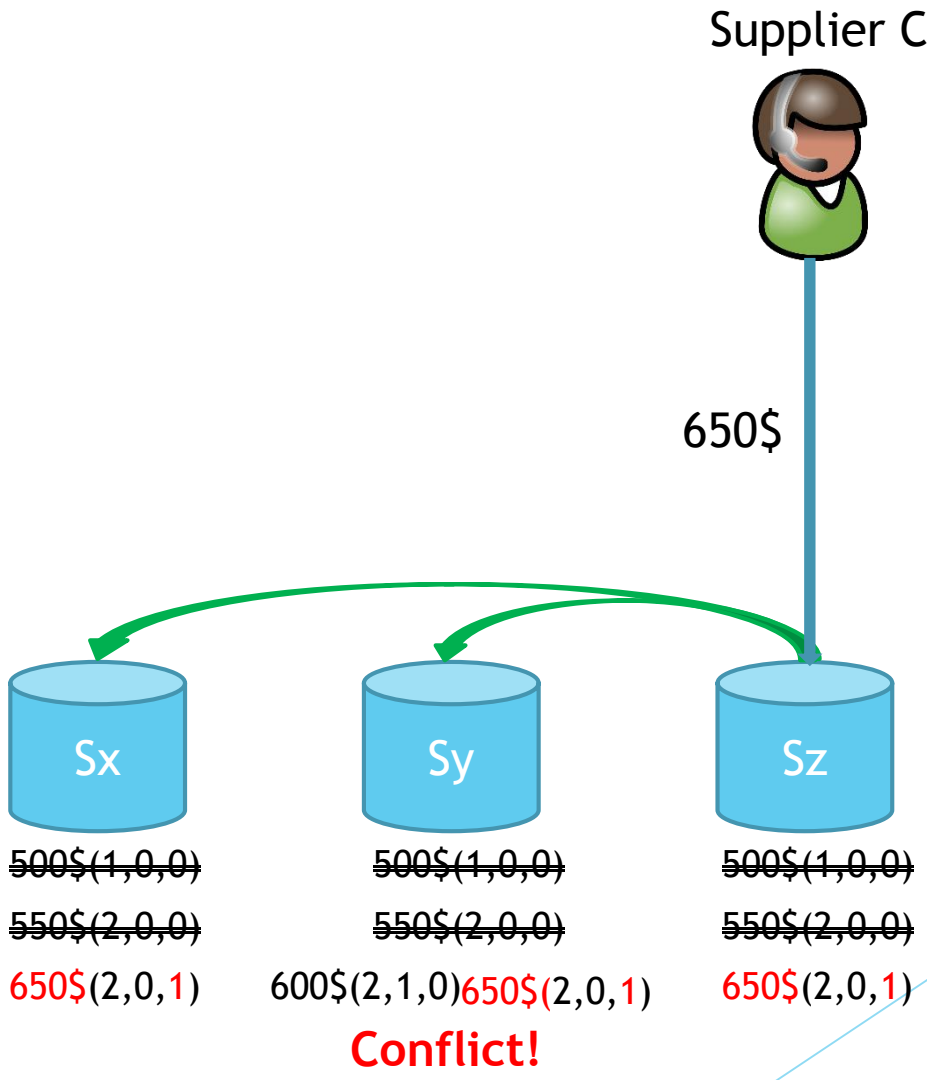
Vector Clock(Example)



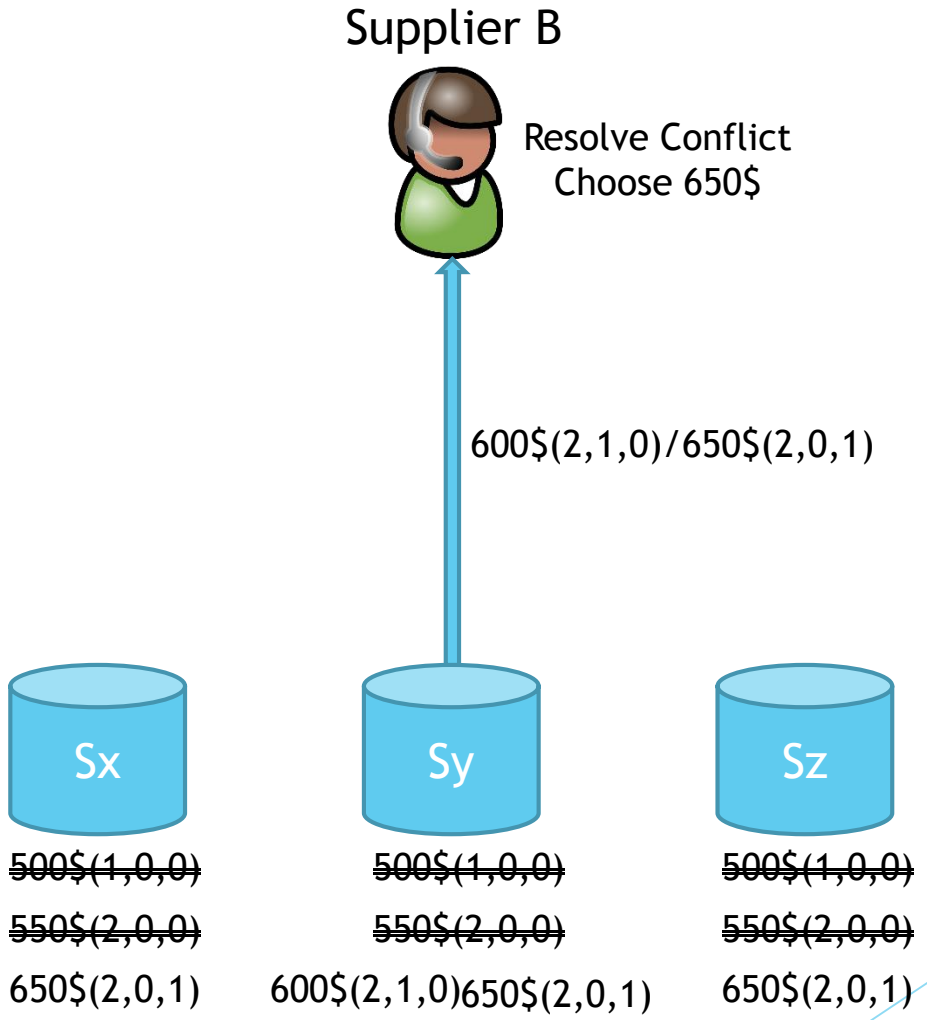
Vector Clock(Example)



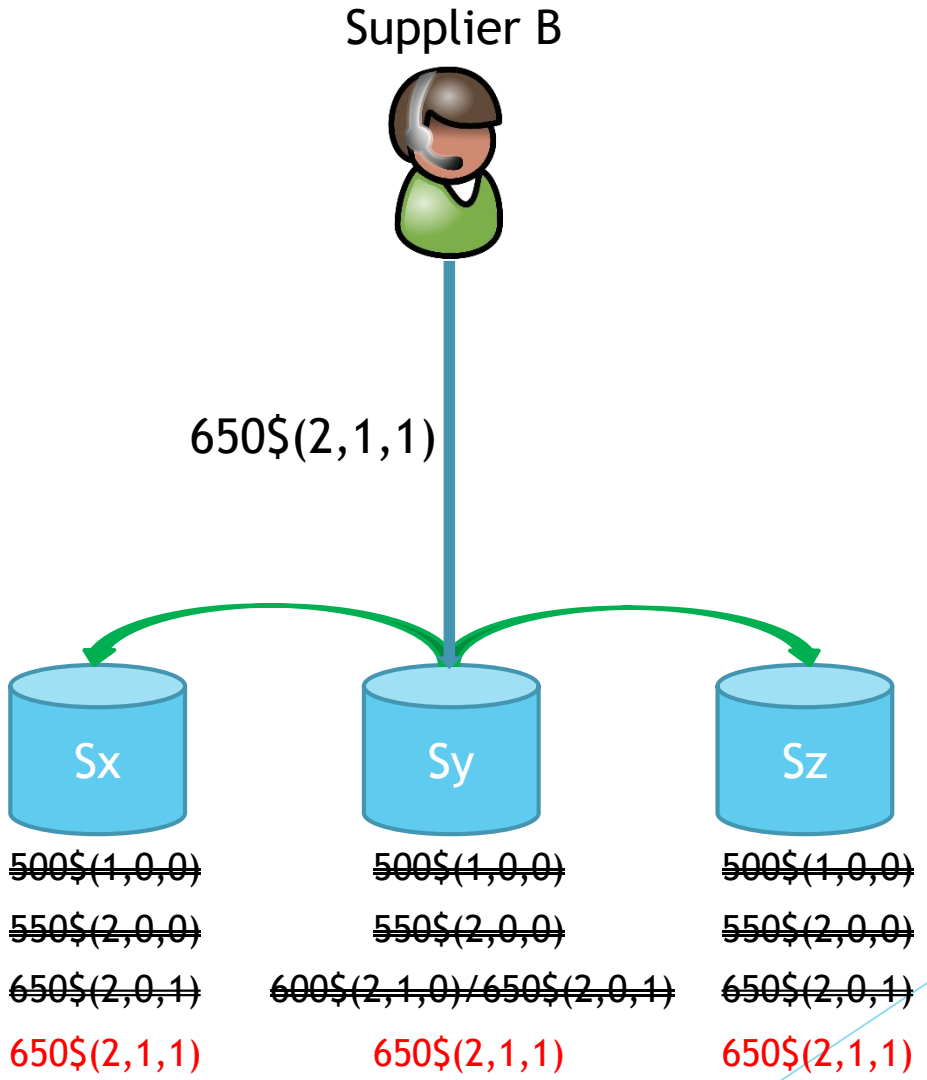
Vector Clock(Example)



Vector Clock(Example)

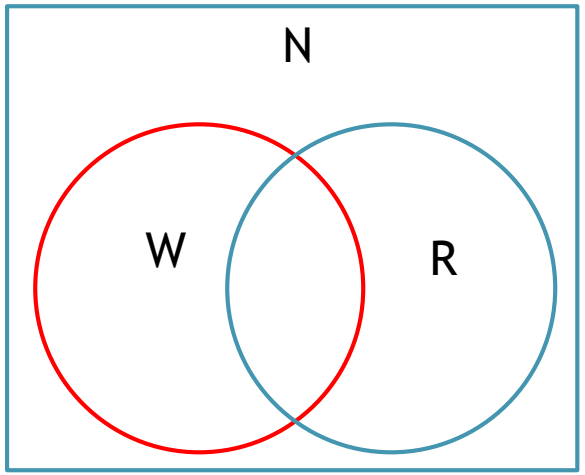


Vector Clock(Example)

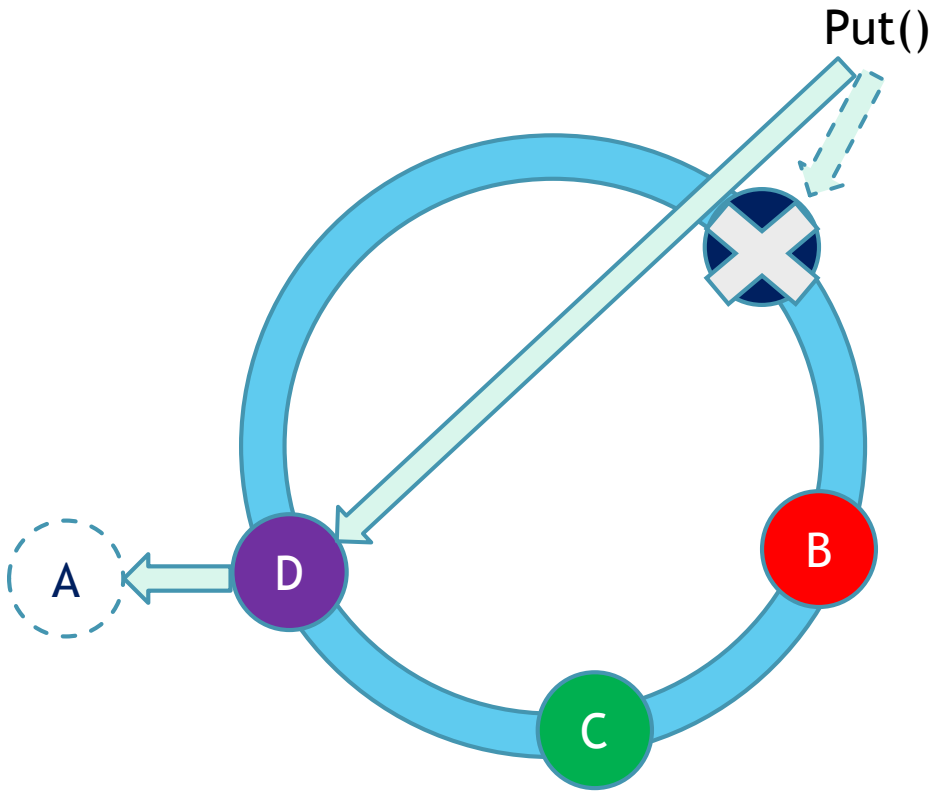


Processing get() and put()

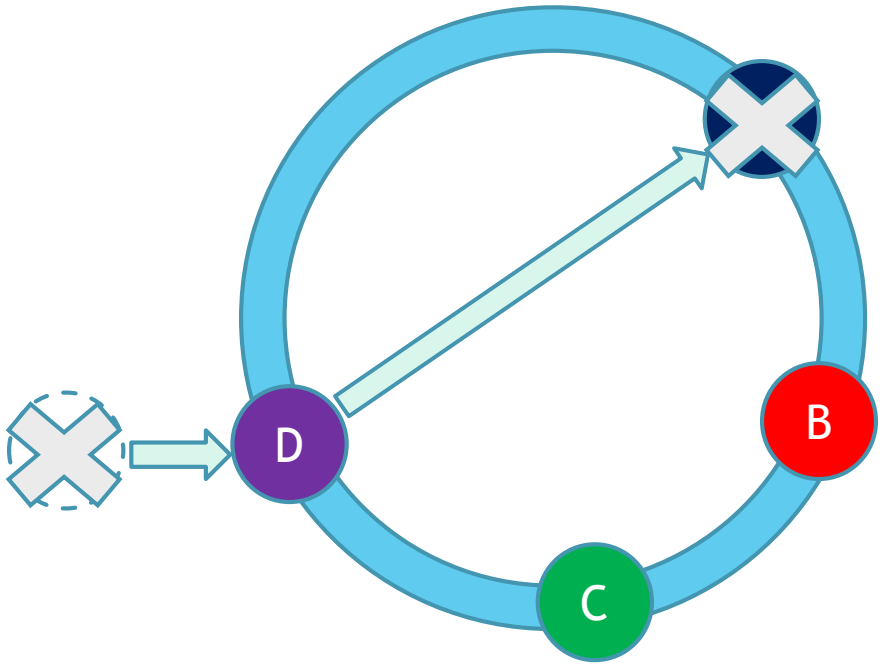
- How to select a coordinator node
  - Load balancer (server-driven)
  - Partition aware client library (client-driven)
- Quorum-like system for consistency
  - W + R > N
  - Typical value: W=2 R=2 N=3



Hinted Handoff



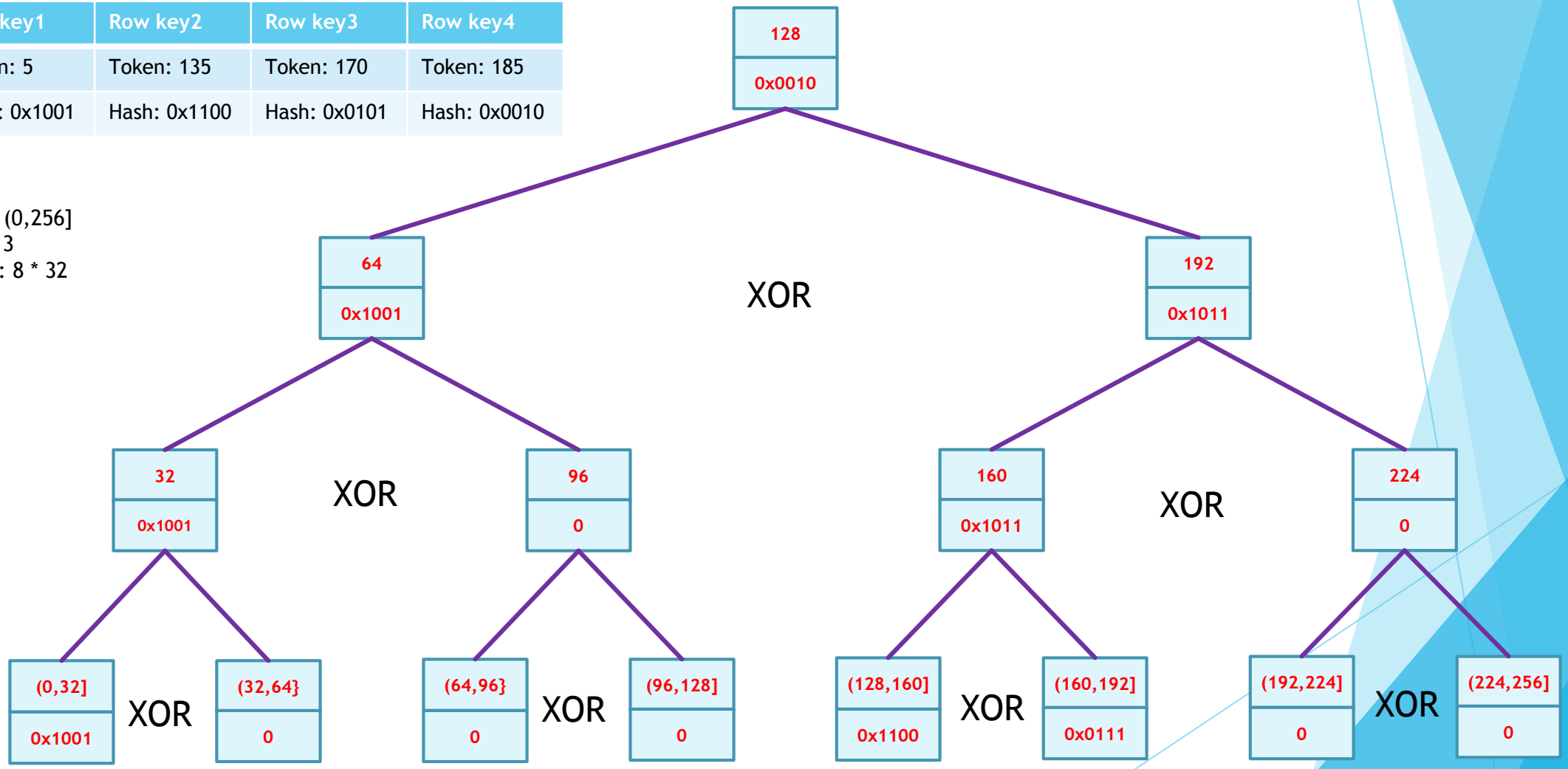
Hinted Handoff



Replica Synchronization(Merkle Tree)

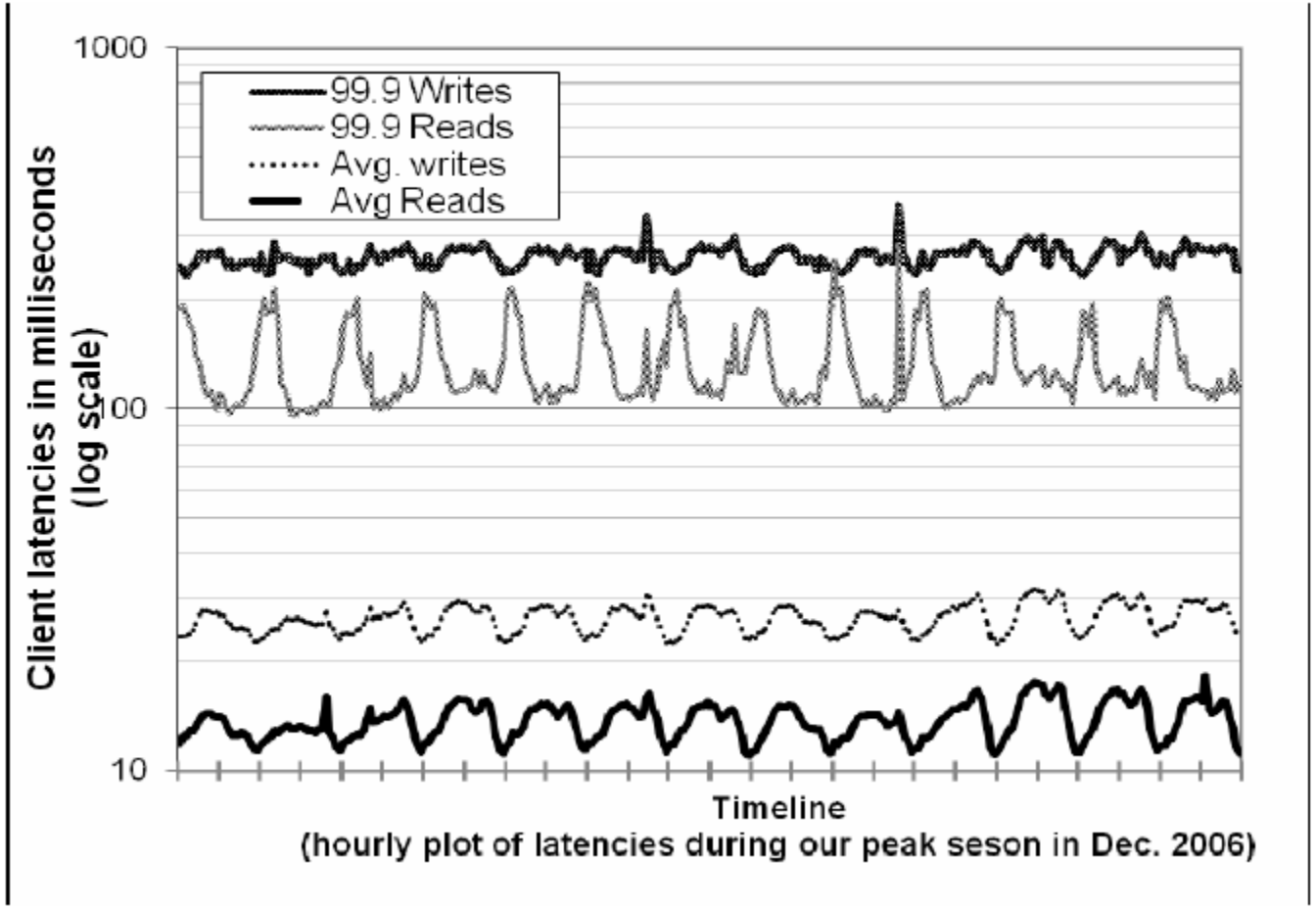
Row key1	Row key2	Row key3	Row key4
Token: 5	Token: 135	Token: 170	Token: 185
Hash: 0x1001	Hash: 0x1100	Hash: 0x0101	Hash: 0x0010

Range: (0,256]
Depth: 3
Tokens: 8 * 32



Example from: http://bit.ly/1fUa0CS

Performance



Q&A

Thank you!