SDLE

Local-First Collaborative Shopping List

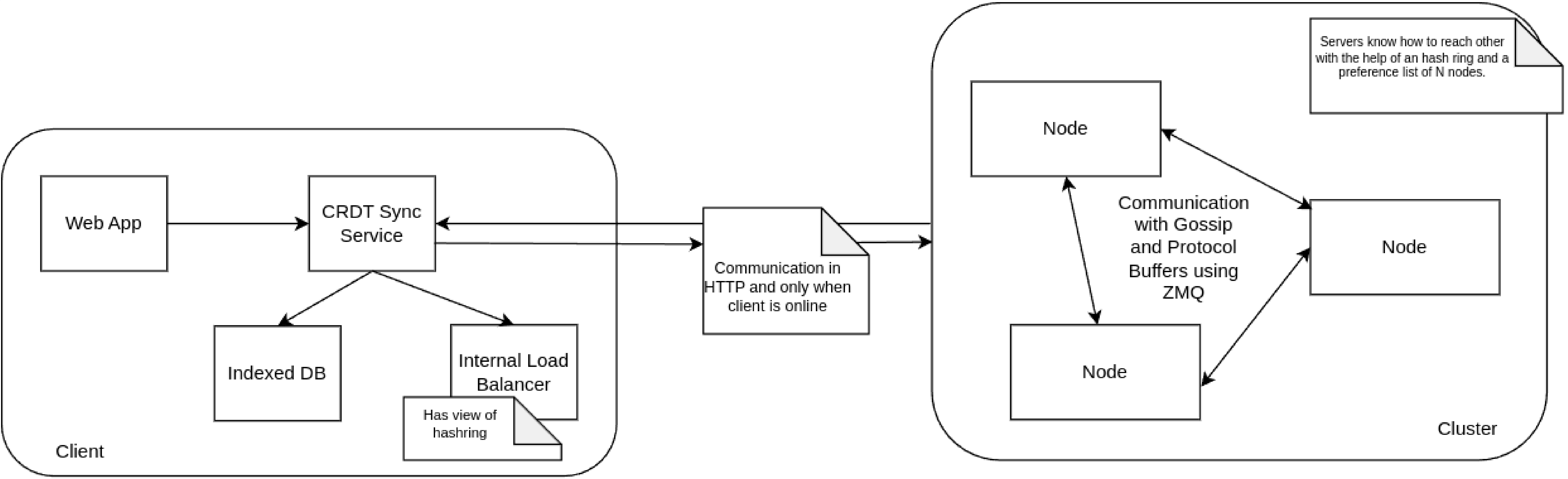
T07 - G13

Henrique Silva

João Fernandes

Tomás Palma

Architecture Overview

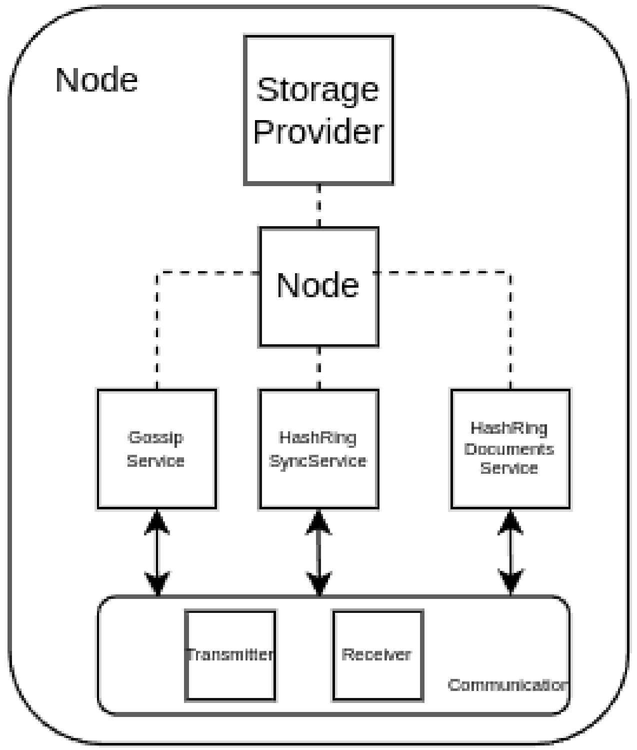


Has view of hashring

Communication in HTTP and only when client is online

Communication with Gossip and Protocol Buffers using ZMQ

Servers know how to reach other with the help of an hash ring and a preference list of N nodes.



Serialization and Deserialization

Frontend to Backend

{JSON}

- Better support with javascript
- Less planned work to implement

Backend (Between server nodes)

protobuf
Protocol Buffers

- Binary format with better performance

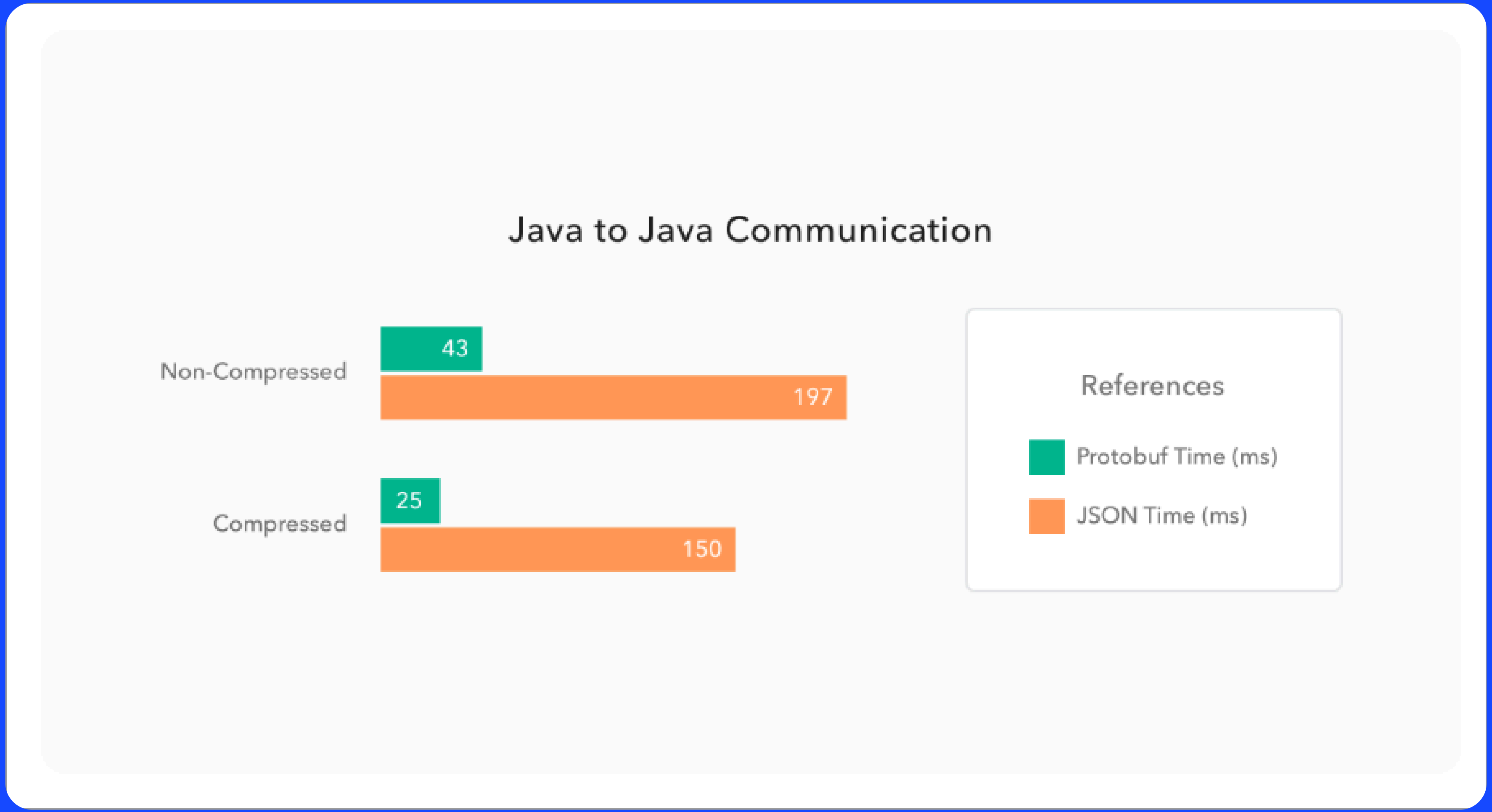
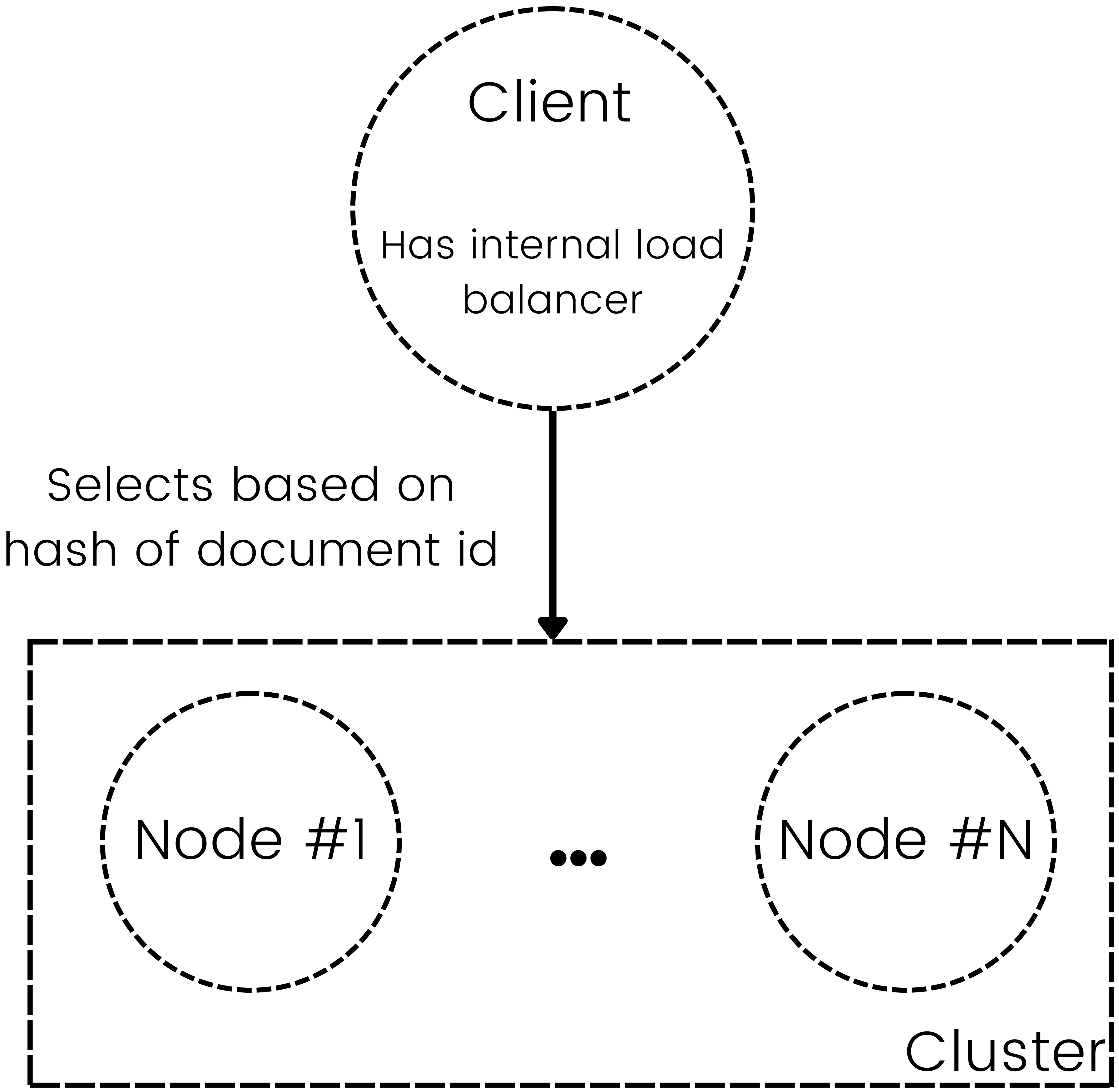


Chart generated by Okta with the performance of
500 GET requests between spring boot applications

Client-Side Load Balancing (1 /2)


Client
Has internal load balancer
Selects based on hash of document id
Node #1
...
Node #N
Cluster

Client-Side Load Balancing (2/2)

- Reduces latency as it reduces round hops
- Not necessarily insecure since the client is controlled by the web server serving the web app

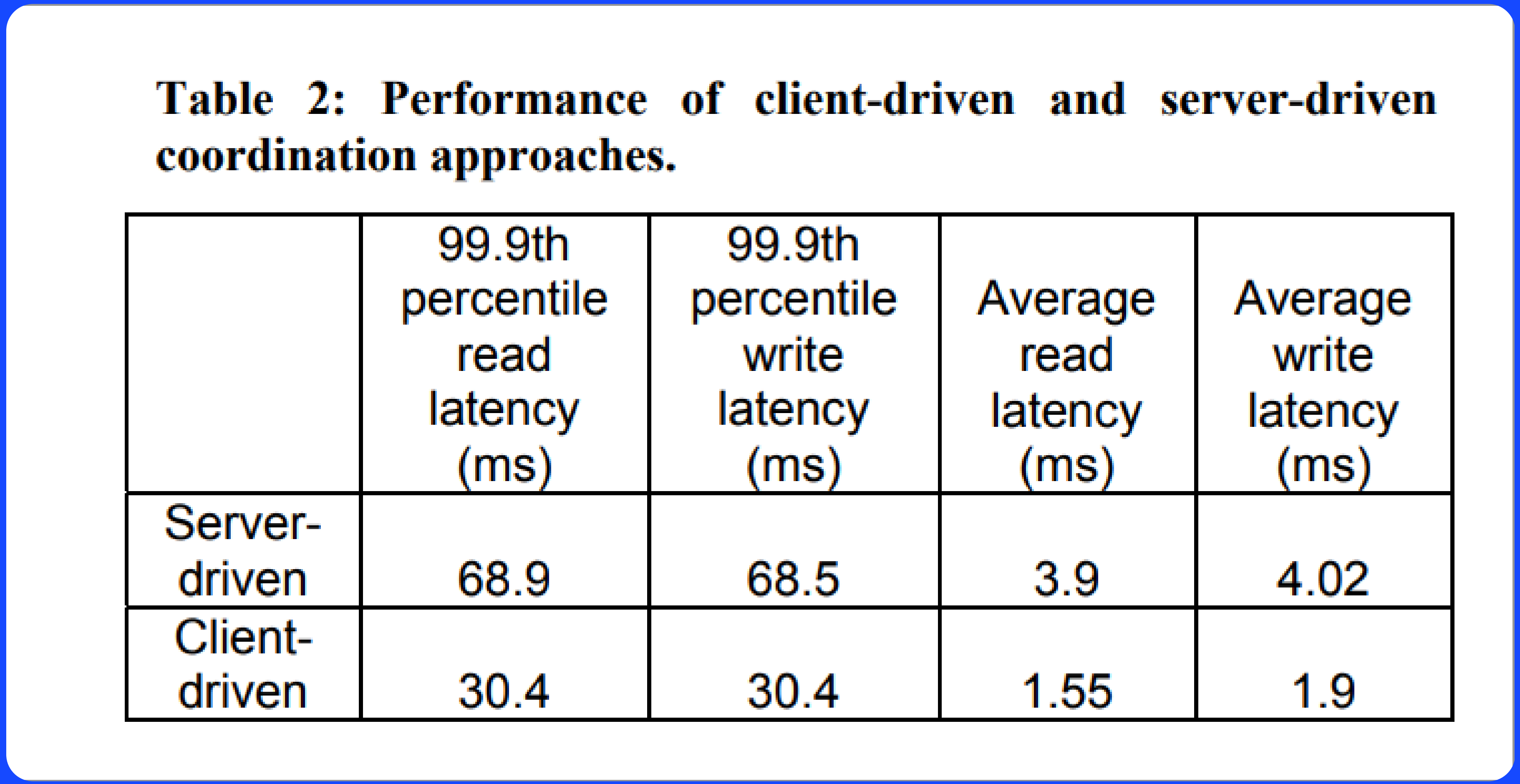


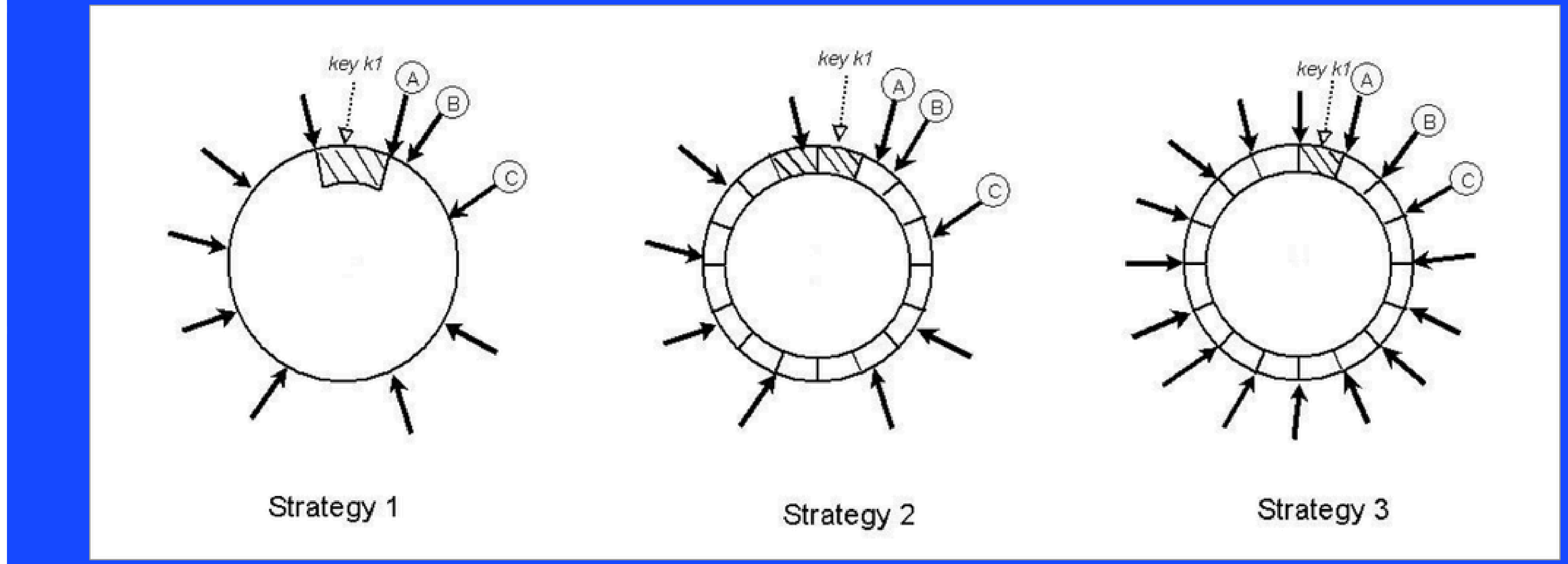
Table 2: Performance of client-driven and server-driven coordination approaches.

	99.9th percentile read latency (ms)	99.9th percentile write latency (ms)	Average read latency (ms)	Average write latency (ms)
Server-driven	68.9	68.5	3.9	4.02
Client-driven	30.4	30.4	1.55	1.9

Dynamo Paper

Ring partioning scheme (1 /2) - Strategies Explored

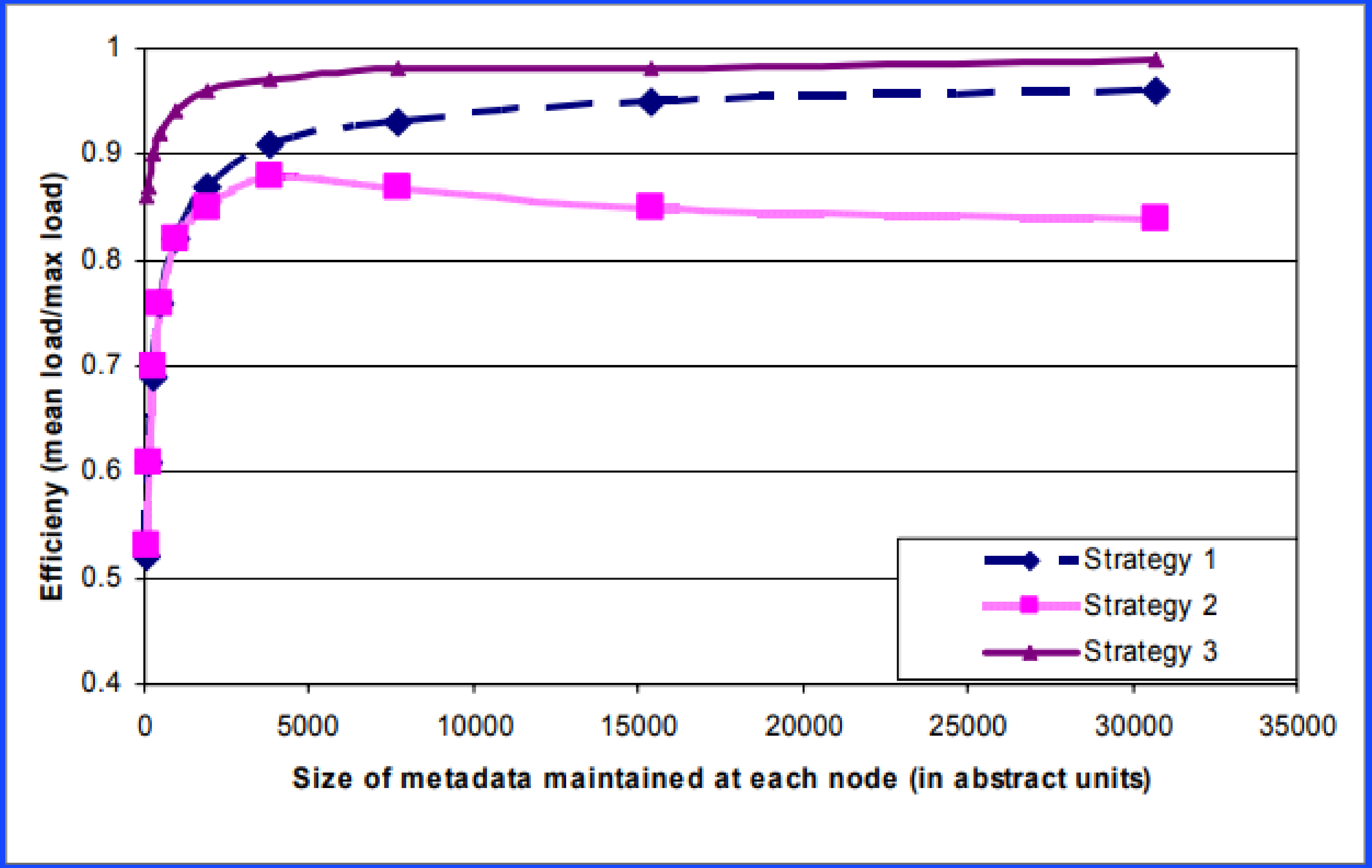
STRATEGY NAME	FUNCTIONING
STRATEGY 1	Assign randomly the hash of a node to the ring
STRATEGY 2	Assign random hashes to nodes and decide which node to take which key based on well defined partitions
STRAGEGY 3	Divide the hash ring into well defined partitions and then assign partitions to nodes



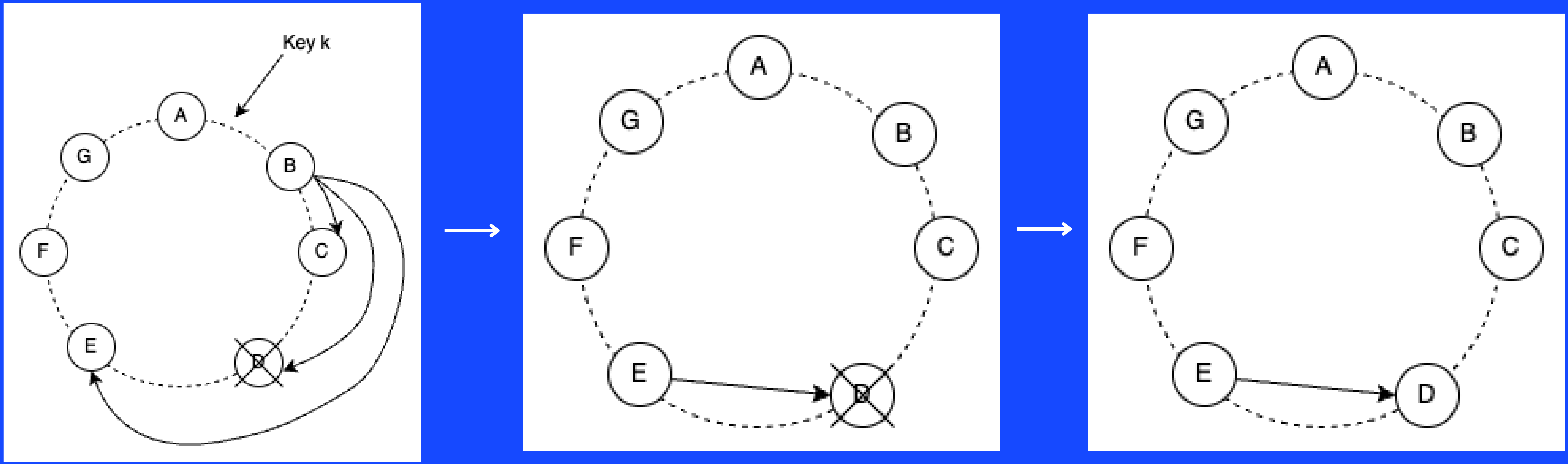
Dynamo Paper

Ring partioning scheme (2 /2) - Decision

- Strategy 2 was discarded
- Strategy 1 and Strategy 3 do not have significant efficiency differences
- Since strategy 1 was simpler to develop, that was the chosen one
- The hashring is modeled as a TreeMap<BigInteger, NodeIdentifier> where each node has more than one entry

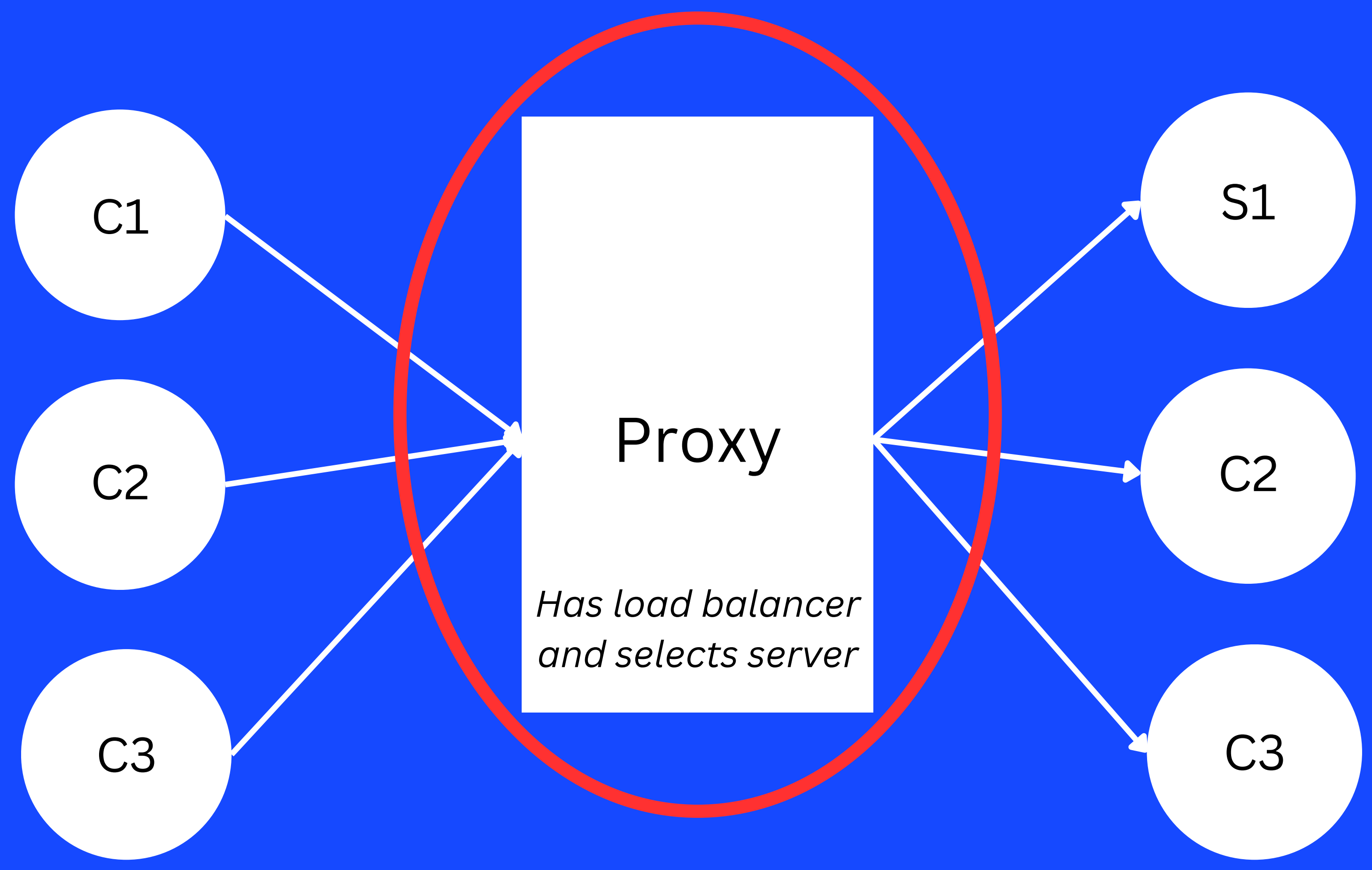


Sloppy quorums & Hinted Handoffs

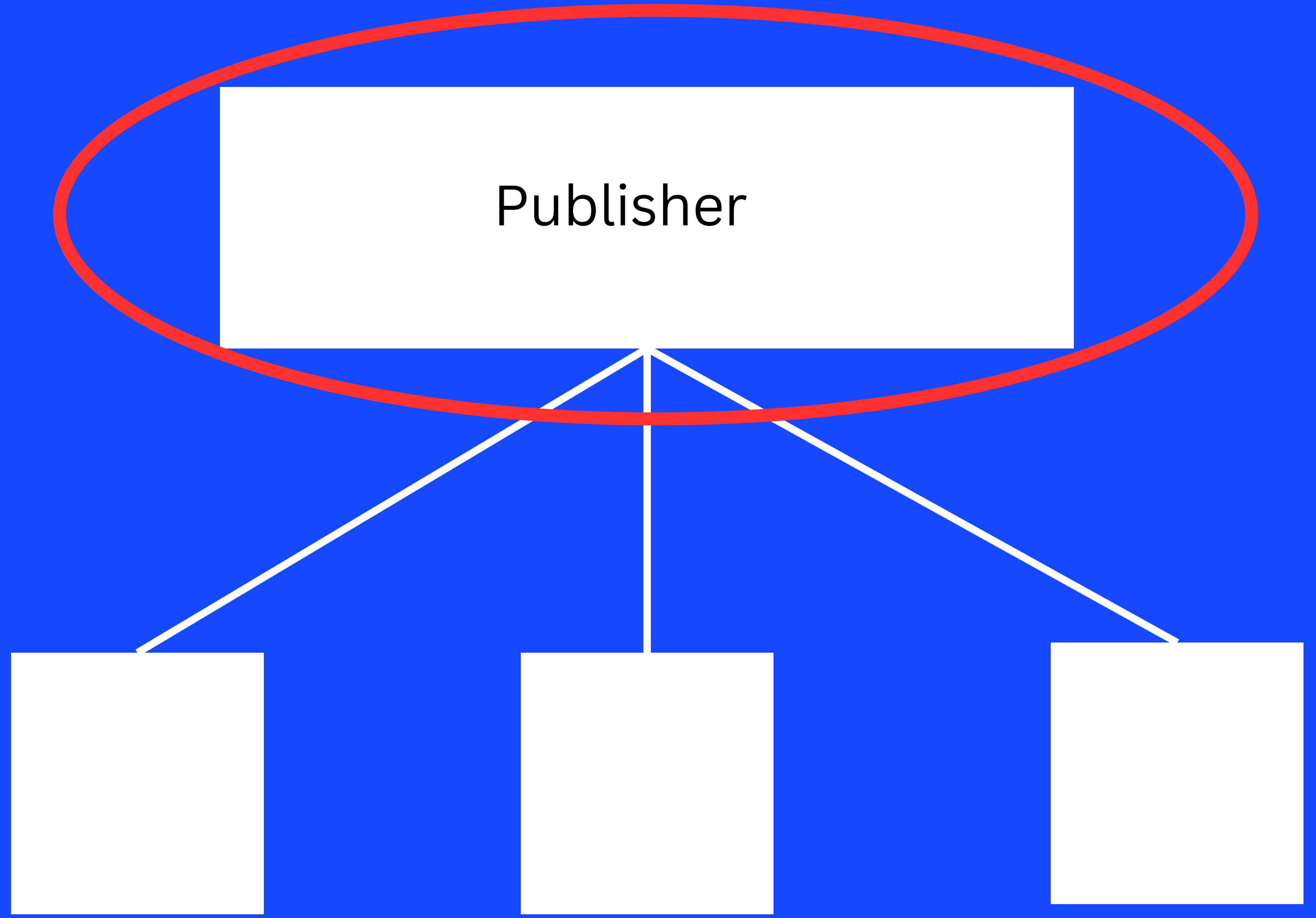


Reason neither pub-sub nor proxy patterns were used (1/2)

1. Single point of failure



Proxy pattern



Pub-Sub pattern

Although fixed by DNS, it remains a centralized approach

Reason neither pub-sub nor proxy patterns were used (2/2)

2. Other notes

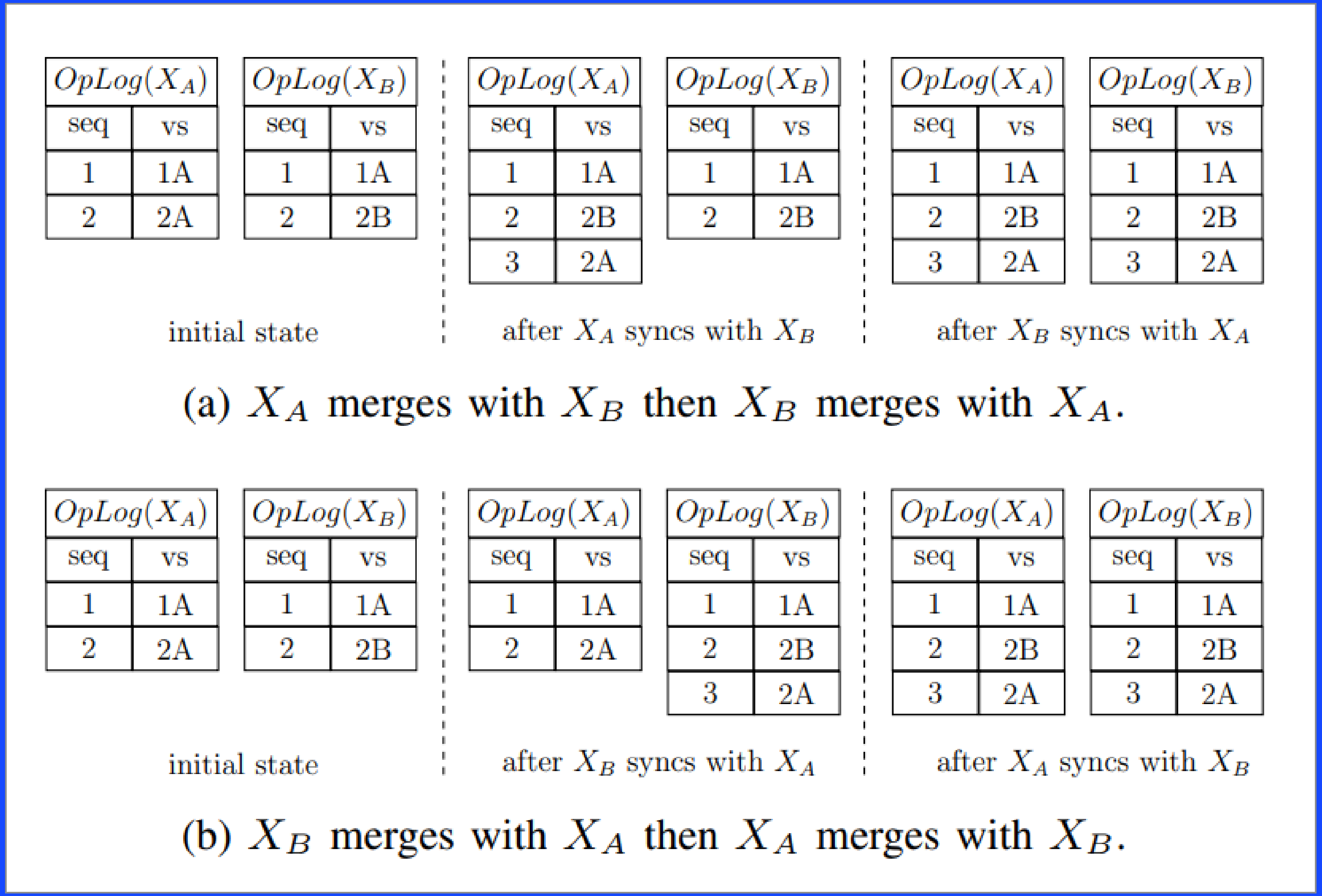
- Even if we had many servers under DNS we would still have to have a conflict resolution mechanism, so we chose to try to implement one.
- The assignment of IP Addresses of a certain name is not decentralized

Log-CRDT

https://sites.cs.ucsb.edu/~ckrintz/papers/ic2e22.pdf

- Used to maintain log of HashRing operations

- KnowledgeLogs can be used to avoid full list scans



$OpLog(X_A)$	
seq	vs
1	1A
2	2A

$OpLog(X_B)$	
seq	vs
1	1A
2	2B

initial state

$OpLog(X_A)$	
seq	vs
1	1A
2	2B
3	2A

$OpLog(X_B)$	
seq	vs
1	1A
2	2B

after X_A syncs with X_B

$OpLog(X_A)$	
seq	vs
1	1A
2	2B
3	2A

$OpLog(X_B)$	
seq	vs
1	1A
2	2B
3	2A

after X_B syncs with X_A

(a) X_A merges with X_B then X_B merges with X_A.

$OpLog(X_A)$	
seq	vs
1	1A
2	2A

$OpLog(X_B)$	
seq	vs
1	1A
2	2B

initial state

$OpLog(X_A)$	
seq	vs
1	1A
2	2A

$OpLog(X_B)$	
seq	vs
1	1A
2	2B
3	2A

after X_B syncs with X_A

$OpLog(X_A)$	
seq	vs
1	1A
2	2B
3	2A

$OpLog(X_B)$	
seq	vs
1	1A
2	2B
3	2A

after X_A syncs with X_B

(b) X_B merges with X_A then X_A merges with X_B.

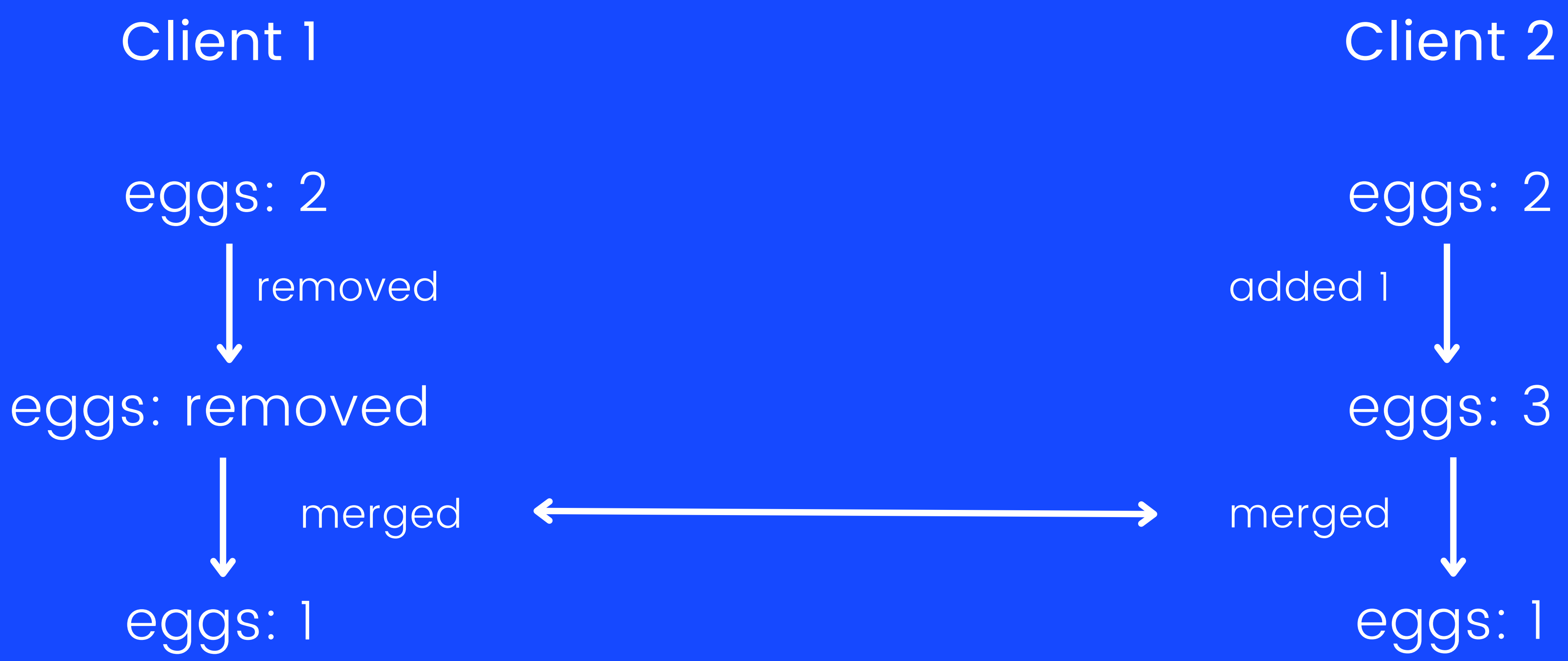
CRDTs

- Dot Context
- Dotted Value
- CCounter (uses: Dotted Value)
- MVRegister (uses: DotContext)
- AWSet (uses: Dot Context and Dotted Value)
- AWMap (uses: Dot Context, Dotted Value and AWSet)
- ShoppingListItem (uses: CCounter and MVRegister)
- ShoppingList (uses: AWMap, Dotted Value and ShoppingListItem)

Removing an element from the list - Counter behaviour

- When an item is removed and updated at the same time, it is not removed and its counter is updated instead

- We use an HashMap<string, DottedValue> (integer DV) as a tombstone to indicate the value of the counter when an item is removed.



Implementation of the CRDT propagation

- PUT and polling GET HTTP requests

- Since this is a local-first app, the client does not need server approval for making changes, storing it locally first and then sync it via the CrdtSyncService if it is online.

References

- https://www.allthingsdistributed.com/files/amazon-dynamo-sosp2007.pdf
- https://sites.cs.ucsb.edu/~ckrintz/papers/ic2e22.pdf
- https://repositorium.sdum.uminho.pt/bitstream/1822/51503/1/Problem-Solution-Counters-PAPOC2016.pdf