



AEROSPIKE

SUMMIT '19

AEROSPIKE

Training Welcome

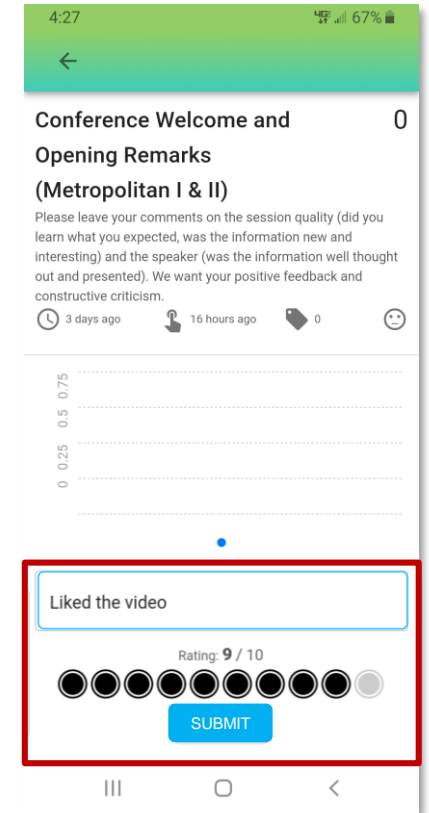
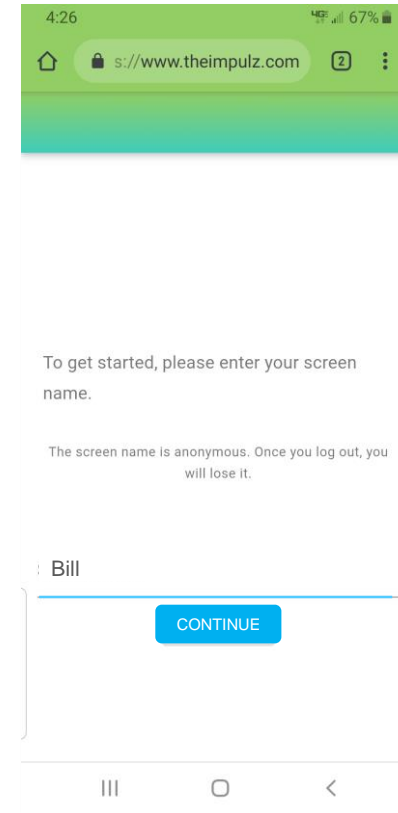
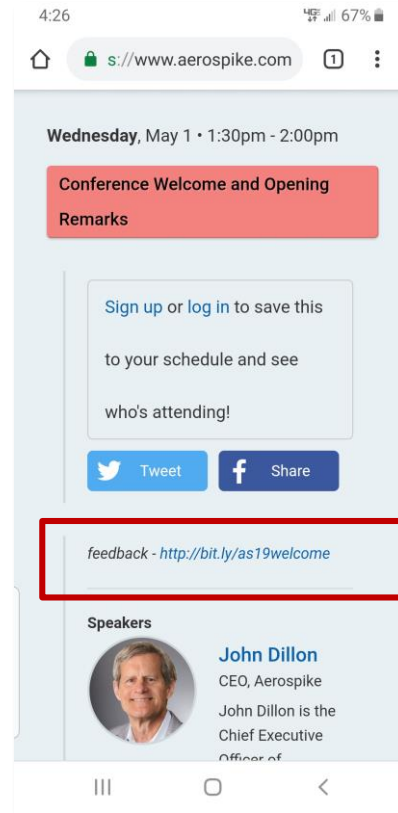
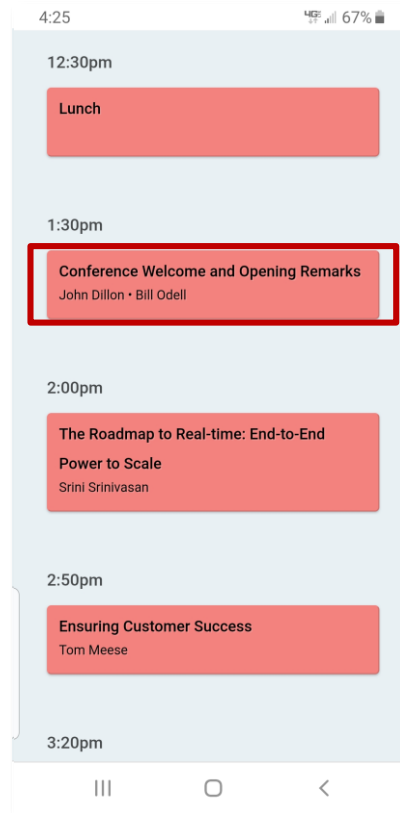
**Tom Meese**  
Director Client Services  
Aerospike

# Training Schedule

7:00 - 8:00AM	<i>Metro Foyer</i> Registration	<i>Metropolitan III</i> Breakfast
8:00 - 8:50AM	<i>Metropolitan I &amp; II</i> Introduction to Aerospike Architecture	
9:00 - 9:45AM	<i>Concordia</i> Advanced Aerospike Features	<i>Olympic</i> Cloud-Ready Aerospike Deployments
9:50 - 10:35AM	Connector for Kafka and JMS, Developing with the REST Client	Strong Consistency Mode in Aerospike
10:35 - 10:55AM	Break	
10:55 - 11:40AM	Designing for Systems of Record and Edge-based Systems	Designing Clusters: Exploiting Rack Aware in Strong Consistency Mode
11:45 - 12:30PM	Key Data Modeling Techniques	Using Automation Tools to Deploy Aerospike
12:30 - 1:30PM	<i>Metropolitan III</i> Lunch	
1:30 - 2:00PM	<i>Metropolitan I &amp; II</i> Conference Welcome and Opening Remarks	

# Agenda: Sessions + Feedback

[aerospike.com/summit/](http://aerospike.com/summit/)





◀EROSPIKE

SUMMIT '19

◀EROSPIKE

## Aerospike Architecture

**Piyush Gupta**  
Director Customer Enablement  
Aerospike

# Module Outline

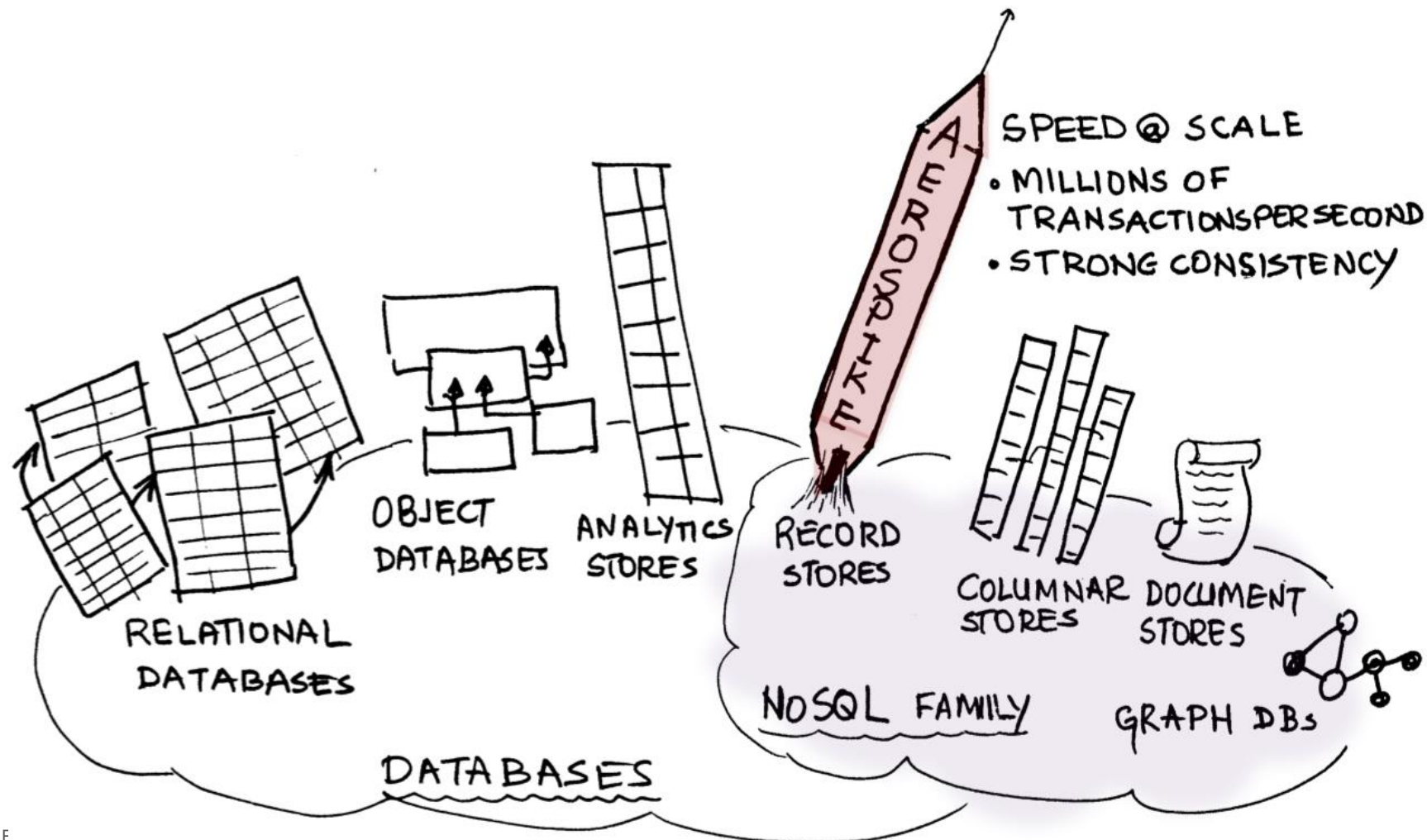
- Introduction to Aerospike.
- Aerospike Data Model
  - How data is organized in Aerospike – Records, Bins, Sets, Namespaces.
  - Storage medium options in Aerospike, Record Location.
- Aerospike Cluster
  - Cluster formation.
  - Cluster state maintenance.
- Data Distribution
  - Succession List, Partition Table
  - Aerospike Client Library and Client connection to cluster.
  - Partition Map.
  - Node loss and node addition.
- Read and Write Transactions



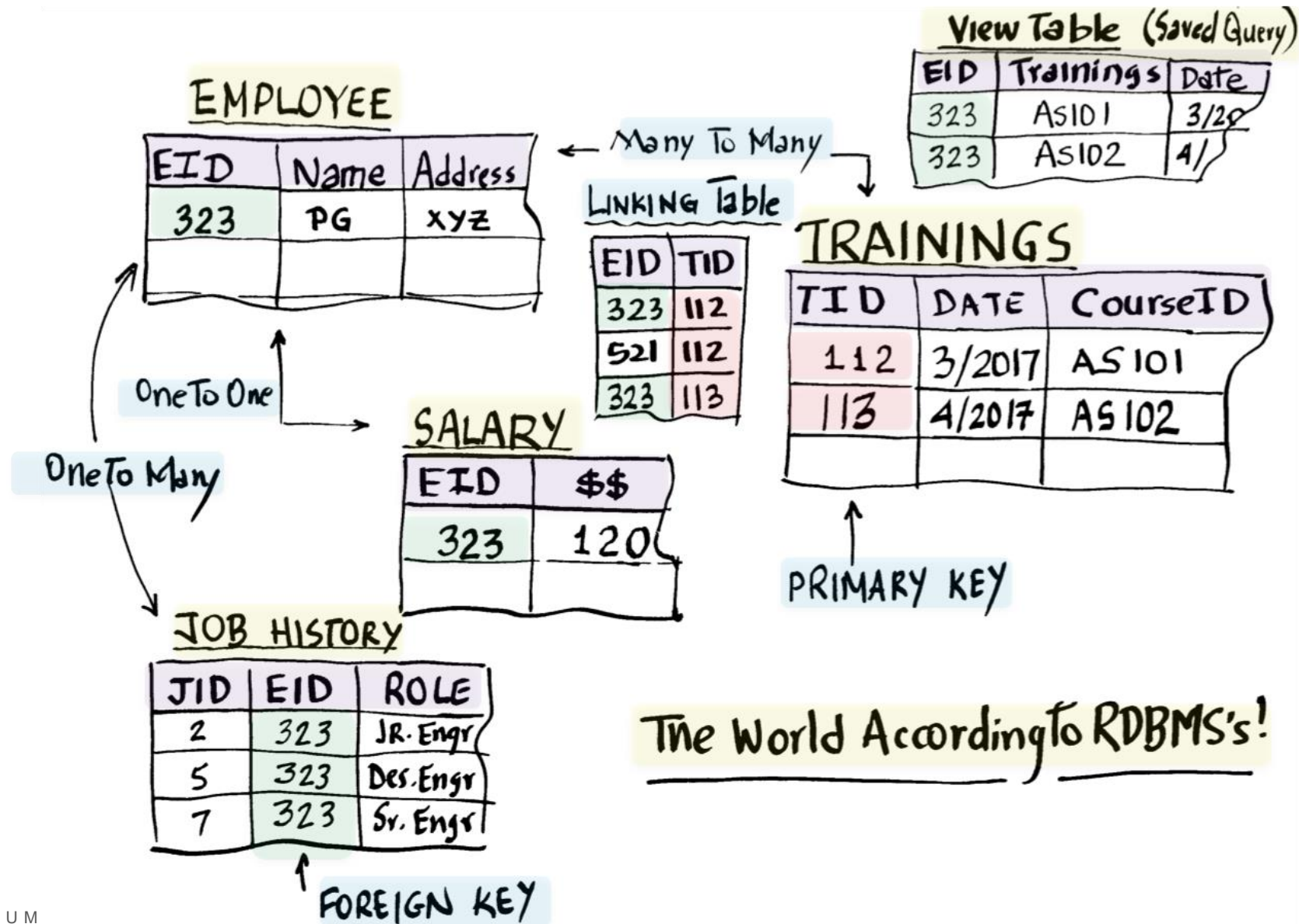
# Introduction to Aerospike

# Introducing Aerospike ...

- Aerospike is a "Record Centric", Distributed, NoSQL Database.



# Relational Data Modeling – Table Centric Schema, 3rdNF



The World According to RDBMS's!



# NoSQL Modeling: Record Centric Data Model

- De-normalization implies duplication of data
  - Queries required dictate Data Model
  - No “Joins” across Tables (No View Table generation)
- Aggregation (Multiple Data Entry) vs Association (Single Data Entry)
  - “Consists of” vs “related to”

## EMPLOYEE

EID	Name	Address	TrngsDate Map
323	PG	xyz	{ AS101 : 3/17 , AS102 : 4/17 ... }

⇒ LIST ALL TRAININGS WHERE EID = 323

## EMPLOYEE\_PII

EID	\$\$	SSN
323	...	...

## TRAININGS

TID	DATE	COURSE	ParticipantsList
112	3/17	AS101	[ 323, 325, 377 ... ]

⇒ LIST ALL PARTICIPANTS WHO HAVE TAKEN COURSE=AS101

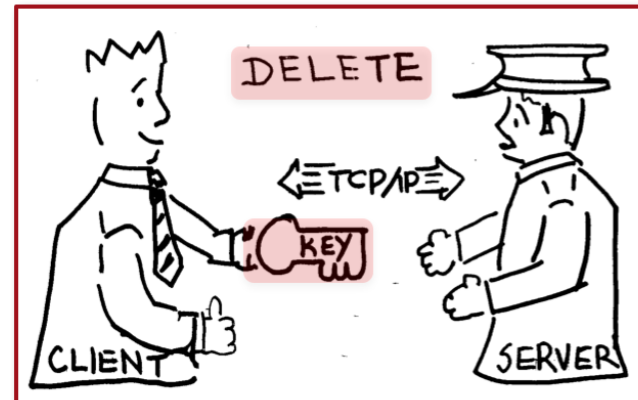
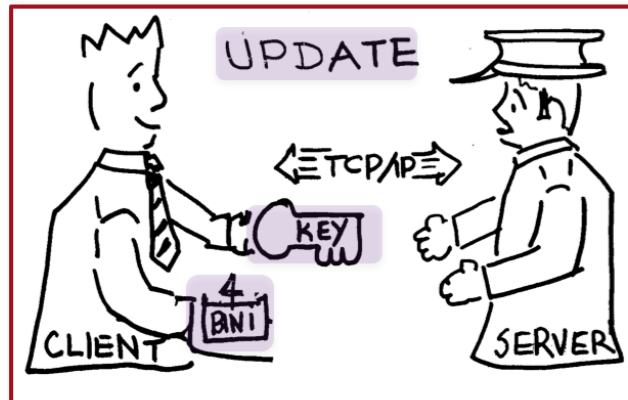
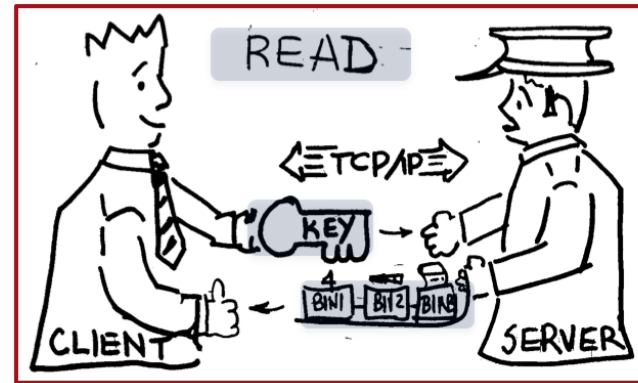
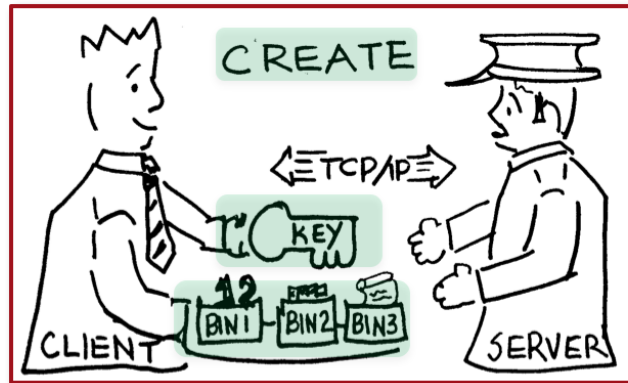
QUERY BASED MODELING FOR NOSQL DBs.



# Aerospike Data Model

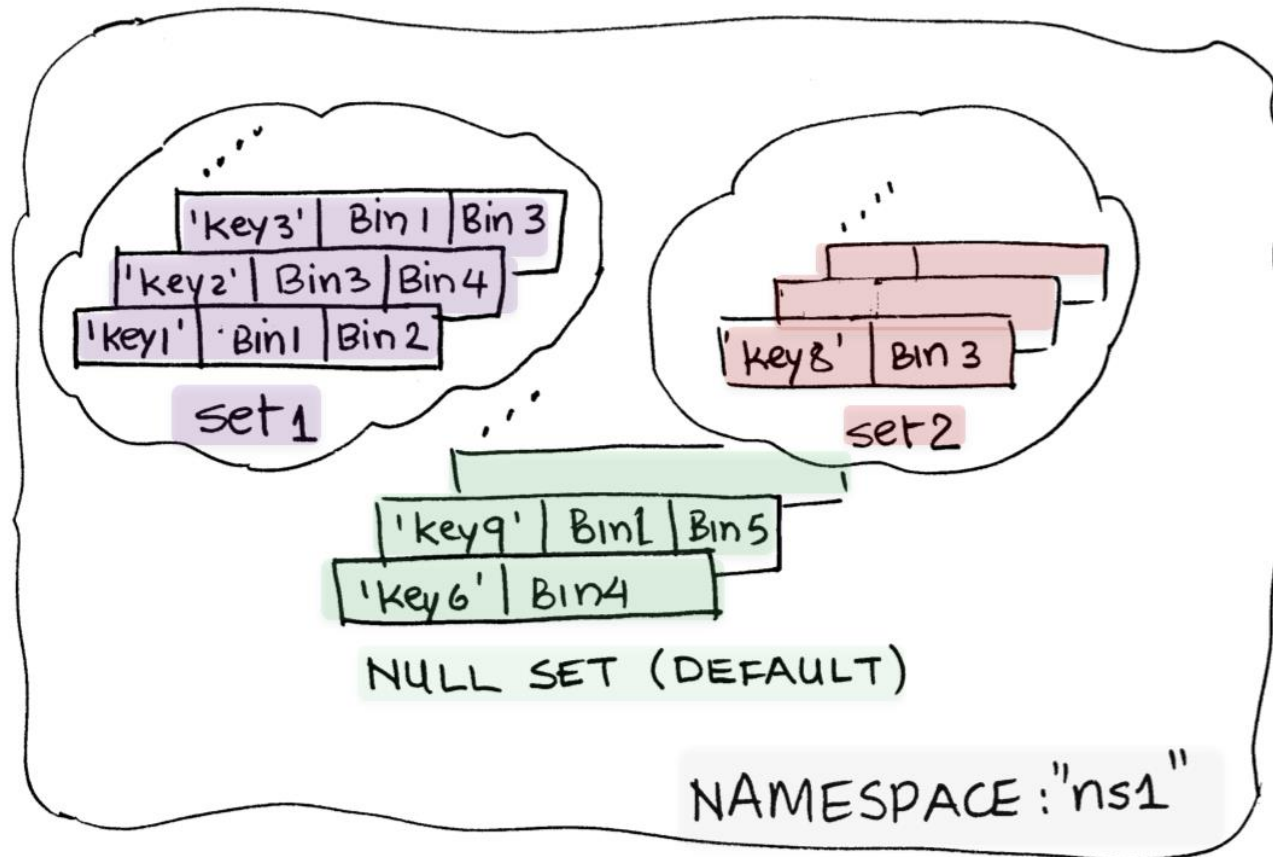
# Records and CRUD Operations

- Records in Aerospike are accessed via a User Assigned "Key".
- A record holds one or more data items in "Bins".
- Perform record **Create** / **Read** / **Update** / **Delete** operations via "Key".



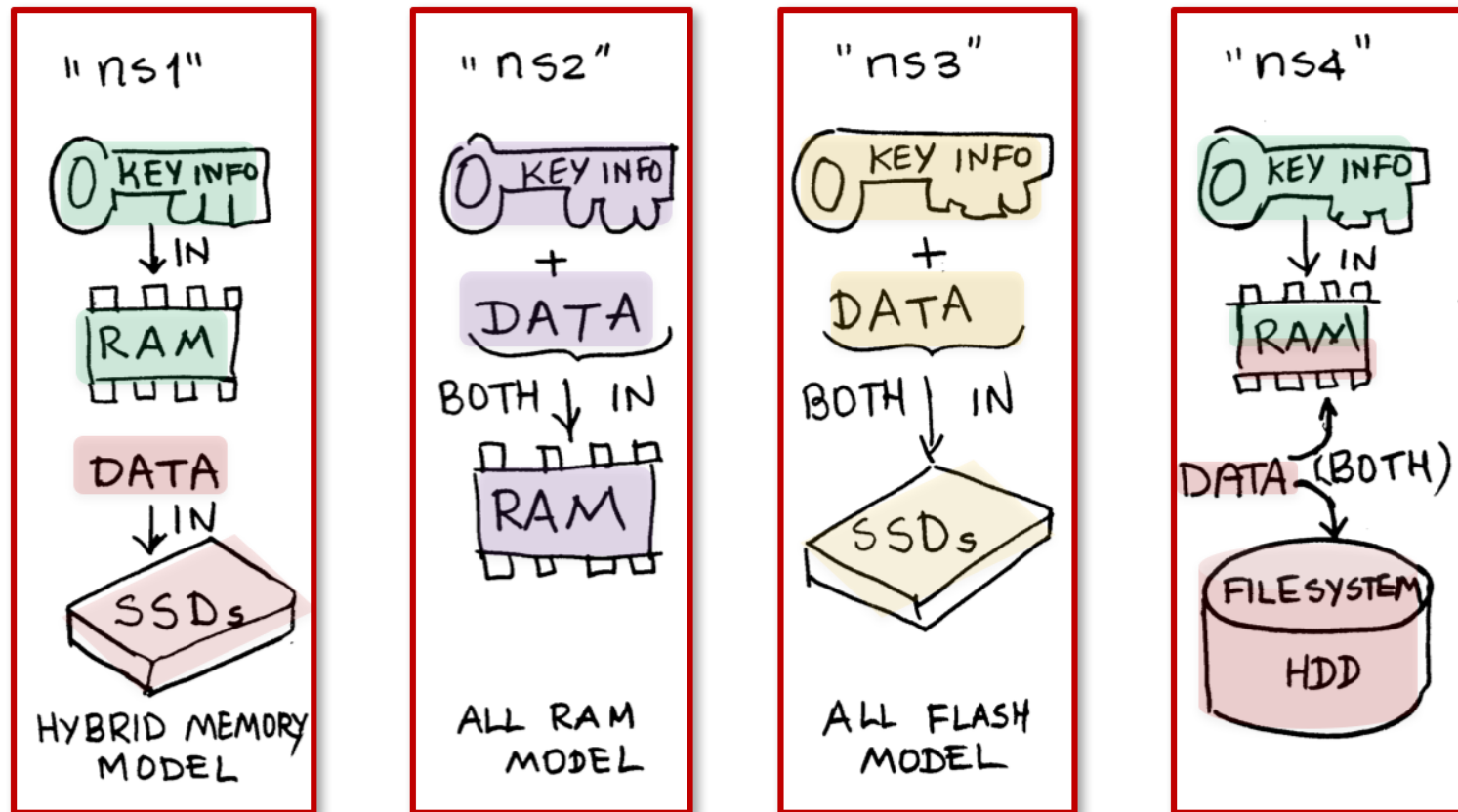
# Sets and Namespaces

- Records always belong to a "Namespace".
- Records may *optionally be* grouped in a "Set" inside a "Namespace".
- By default, all records belong to the **null** set.



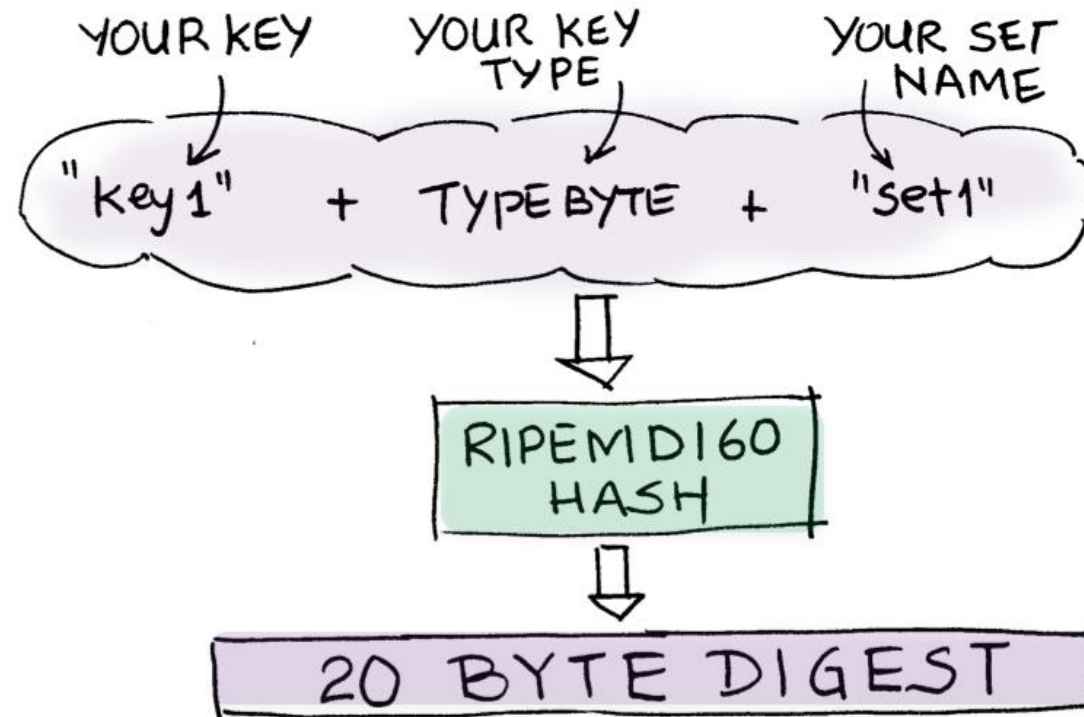
# Namespaces (cont.)

- In a "Namespace", we store information related to the **key** of a record **and** the **data** associated with the record.
- A namespace defines the **storage medium** for storing this information.



# Record Digest

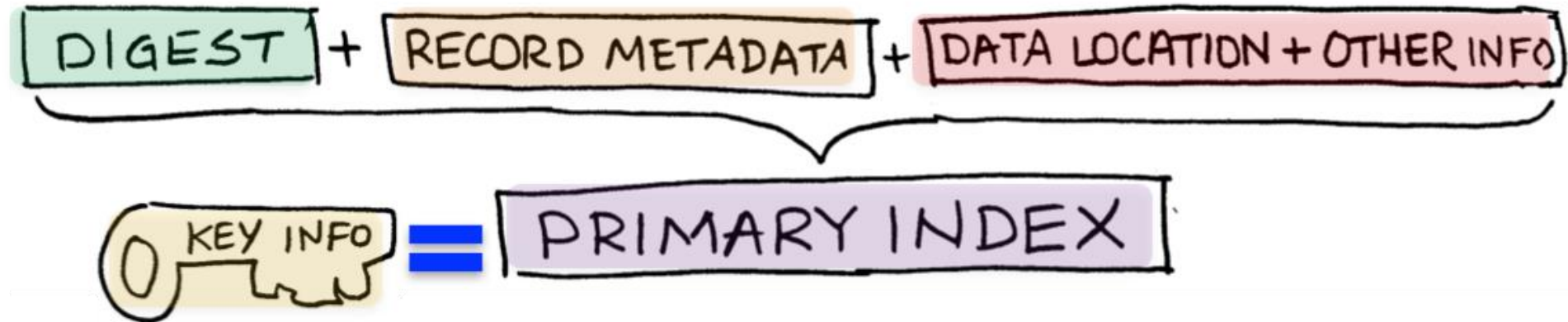
- Exploring "Key Info": When you specify a key, e.g., "key1" – a String, in "set1" to fetch a record from namespace "ns1" ....



- Digest is RIPEMD160 hash (160 bits or 20 bytes) of "key1"+ key type id + set name.

# Primary Index (PI)

- Client, sends the "digest" (instead of your key) to the server.

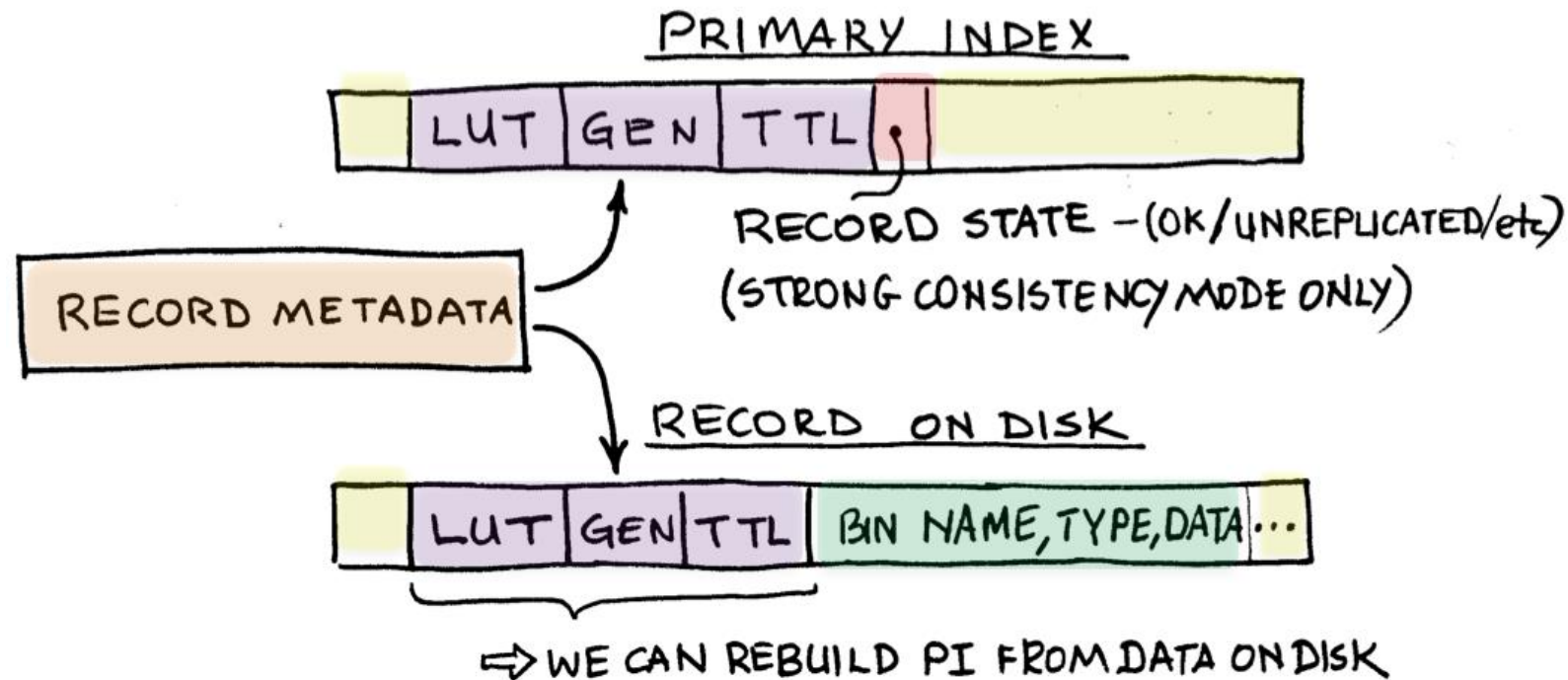


- Server stores the digest along with record storage location information and other record metadata. This "key info" is the **Primary Index (PI)** of the record. PI is fixed **64 bytes** always.
- For the requested digest, server first finds the Primary Index. The PI provides the record data location. Server then fetches the record data.
- Enterprise Edition stores **Primary Index in Linux shared RAM** – survives Aerospike process graceful shutdown and restart (**Fast Restart feature**).

# PI - Record Metadata

Record Metadata comprises:

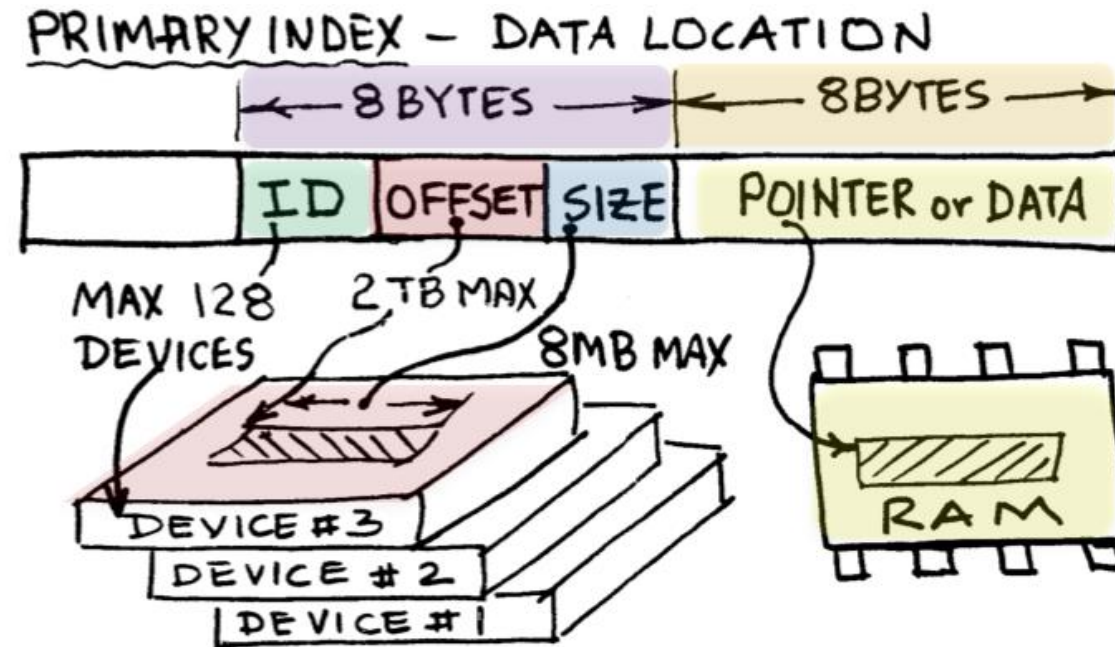
- A record's **Last Update Time (LUT)**
- **Generation (GEN)**, rolling counter, bumps every time a record is updated.
- **TTL – Time-to-live is 0 by default – i.e. live-for-ever** (otherwise 10 years max)





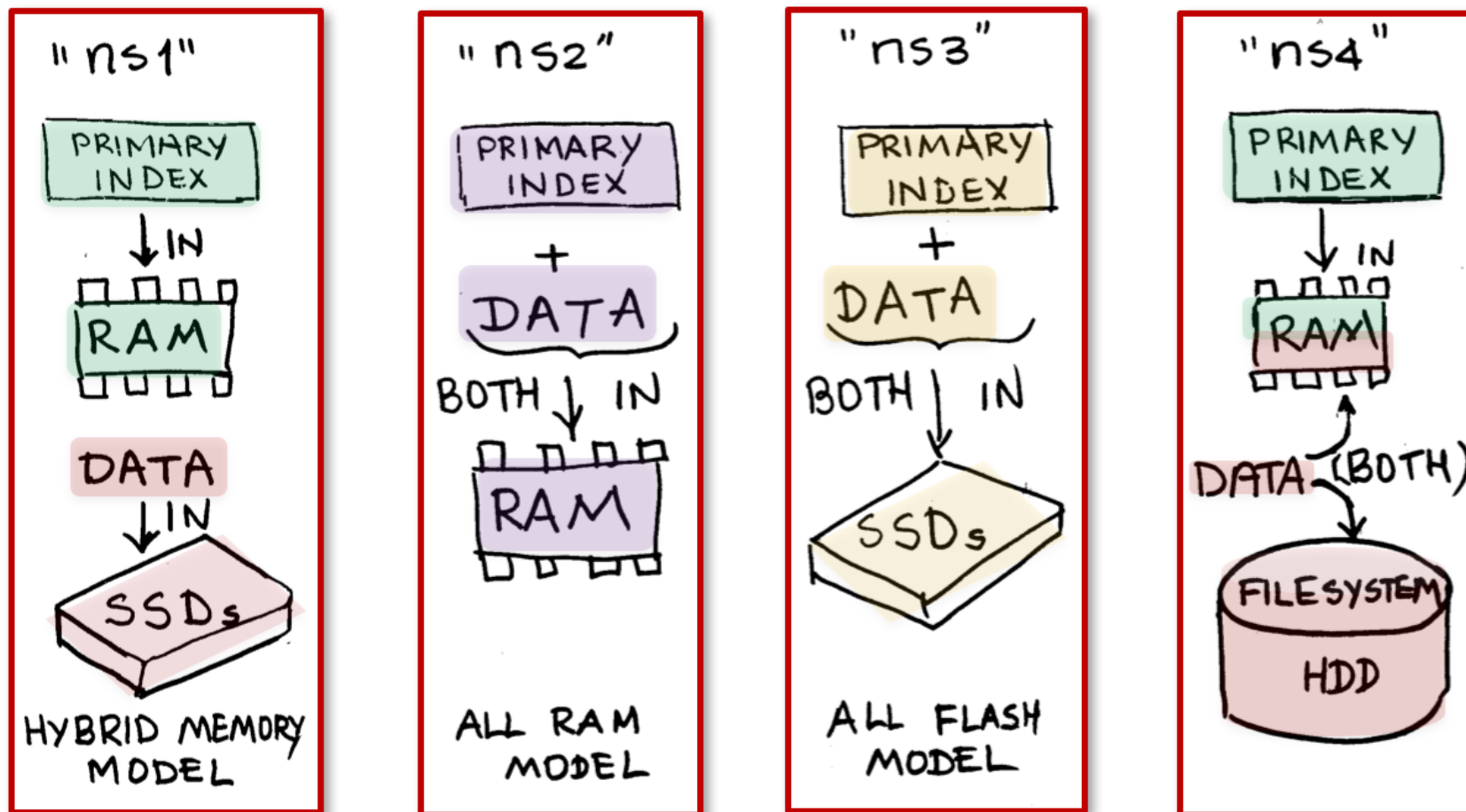
# PI - Record Data Location

- **Data-on-Device (Disk):** (8 bytes) Device ID (max 128 devices), Offset (max 2TB), Size (max 8MB).
- **Data-in-memory:** (8 bytes) Pointer to data in RAM, no 8MB size constraint.
- Data can be in **both memory and device**.
- Special Case - **Data-in-index:** If data is on disk & memory & is single-bin integer or float, we can store it in the PI 's memory pointer bytes.



# Namespaces (cont.)

- Depending on the Data Model, most users define multiple namespaces.
- Namespaces (max 32) are defined when Aerospike is configured.

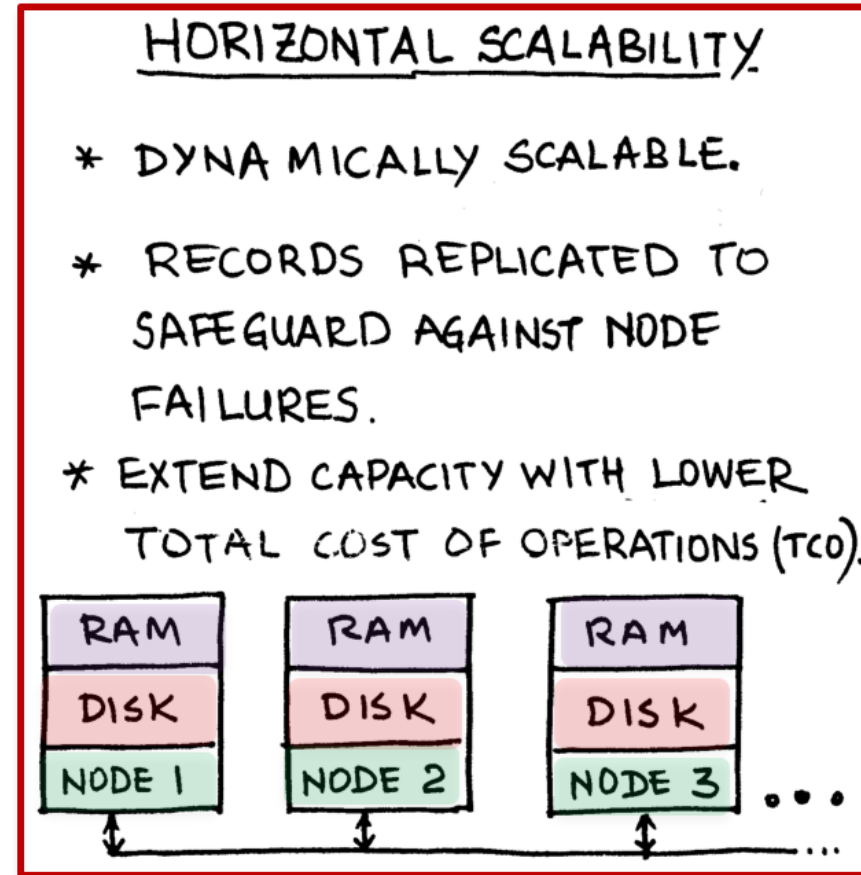
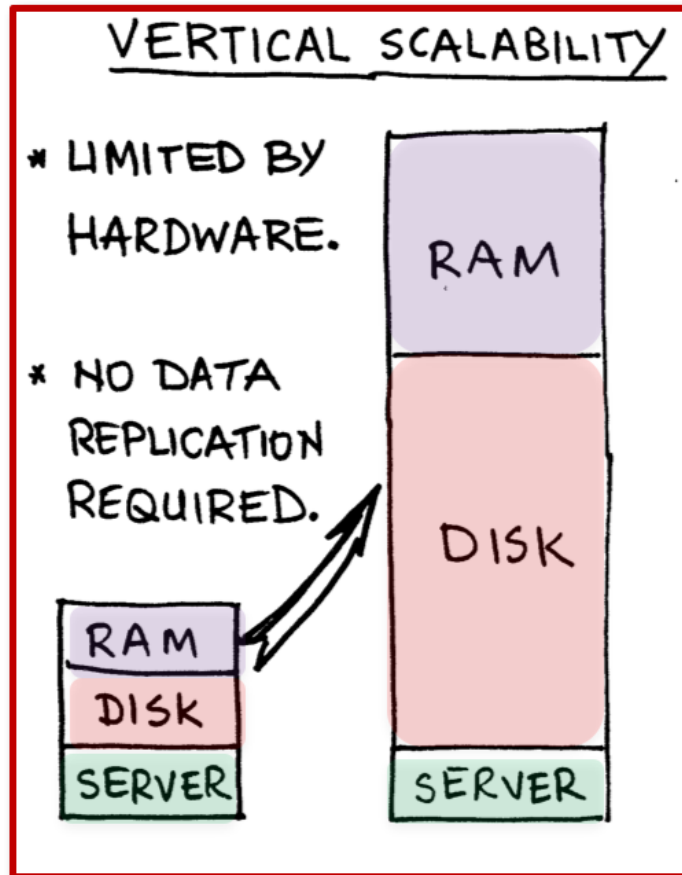




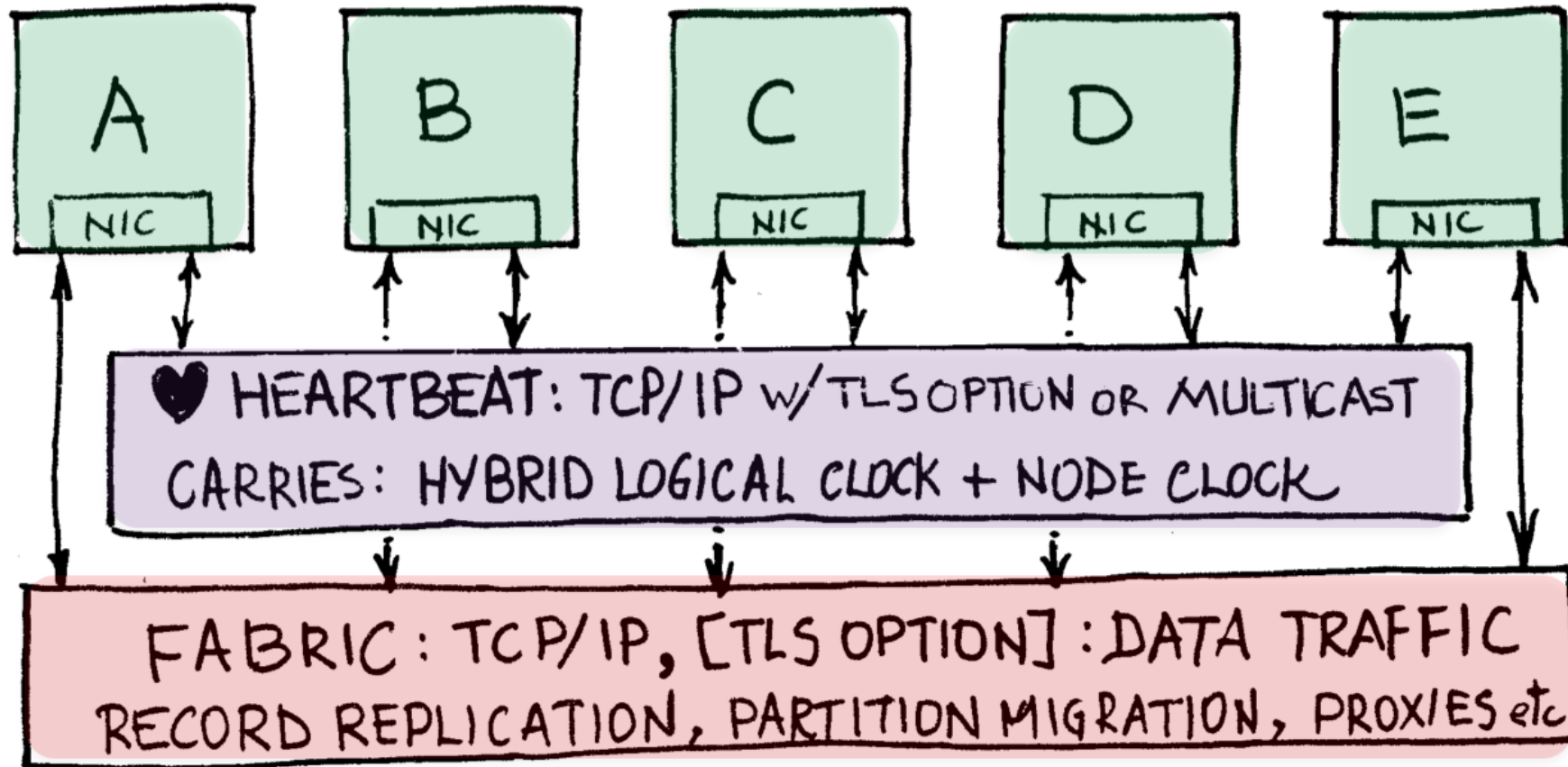
# Aerospike Cluster

# Horizontal Scalability

- Aerospike is a **Horizontally Scalable Distributed** NoSQL Database. You can **store billions of records**, hundreds of **terabytes of data**.
  - Databases running on a single server can be scaled vertically.

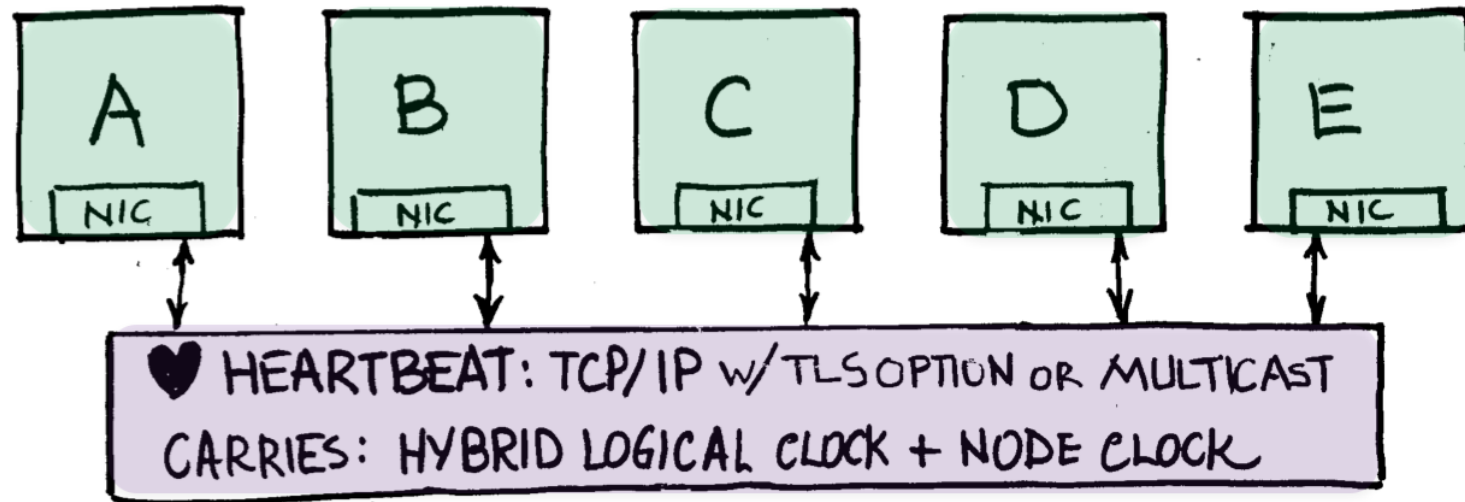


# Aerospike Cluster Formation



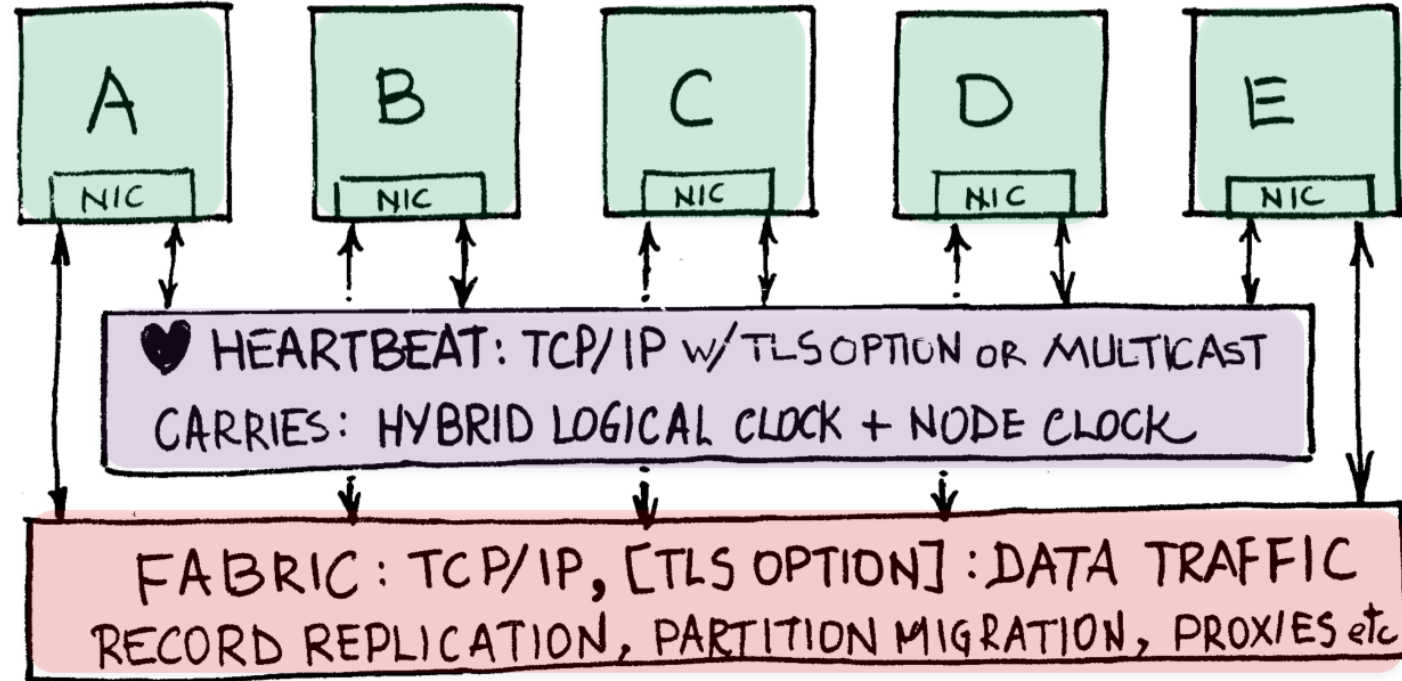
- Aerospike uses a [Shared Nothing Architecture](#) (vs Master-Slave).
- Nodes discover each other (Gossip Protocol) and form a cluster (Paxos).

# Aerospike Cluster Maintained using Heartbeat Messages



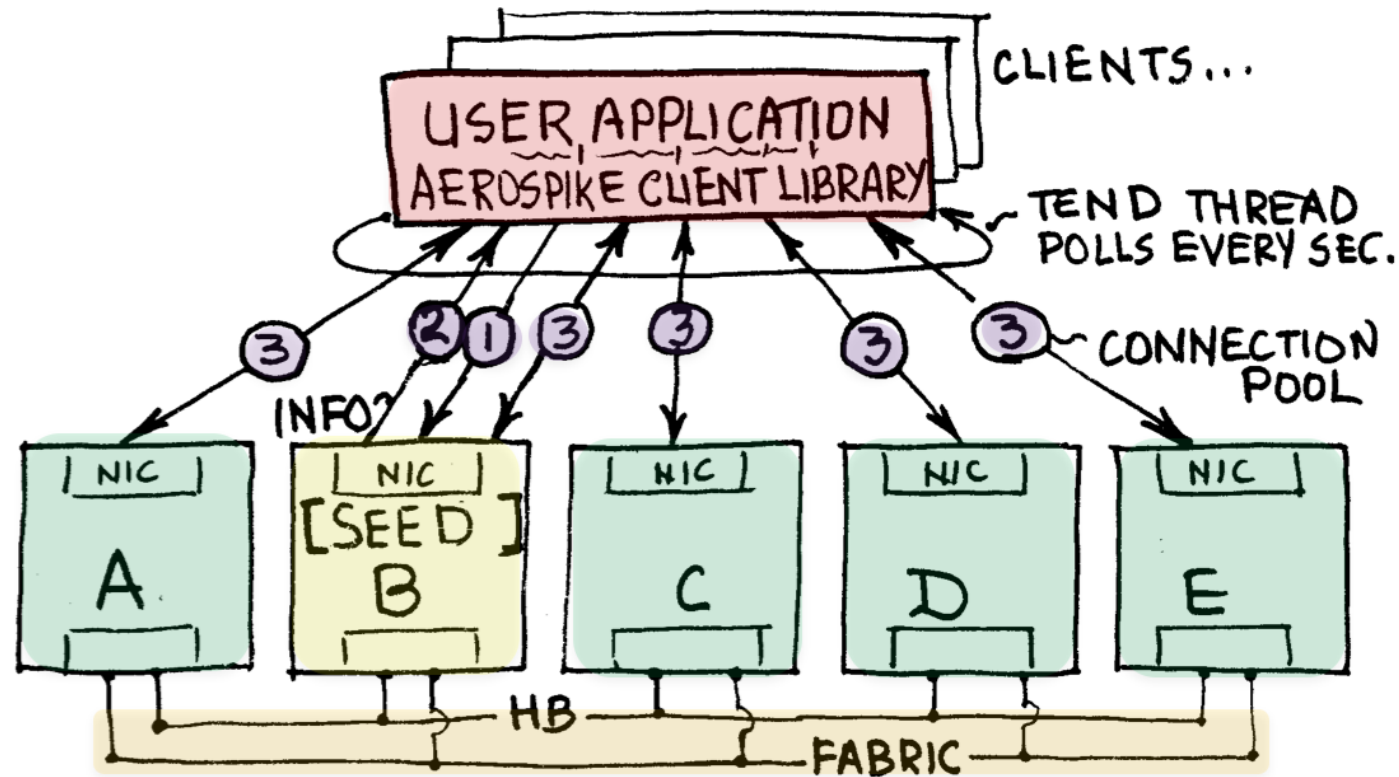
- Cluster membership maintained by periodic (150 ms) exchange of lightweight "Heartbeat" messages. 10 missed heartbeats → node down.
- Heartbeat can be configured to use Multicast or TCP/IP ([mesh mode](#)).
- May encrypt TCP/IP HB messages using TLS (Transport Layer Security).
- TCP/IP (mesh mode) messages exchanged between every node pair.
- Multicast – each node sends HB to router, router broadcasts to all nodes.

# Fabric: The Inter-node Data Layer



- Fabric carries all data traffic between nodes on TCP/IP.
- Latency and bandwidth of fabric will affect performance.
- Aerospike cluster nodes should preferably be on the same LAN.
- Fabric traffic can be encrypted (TLS).
- **Coming soon: Compressed data movement on fabric.**

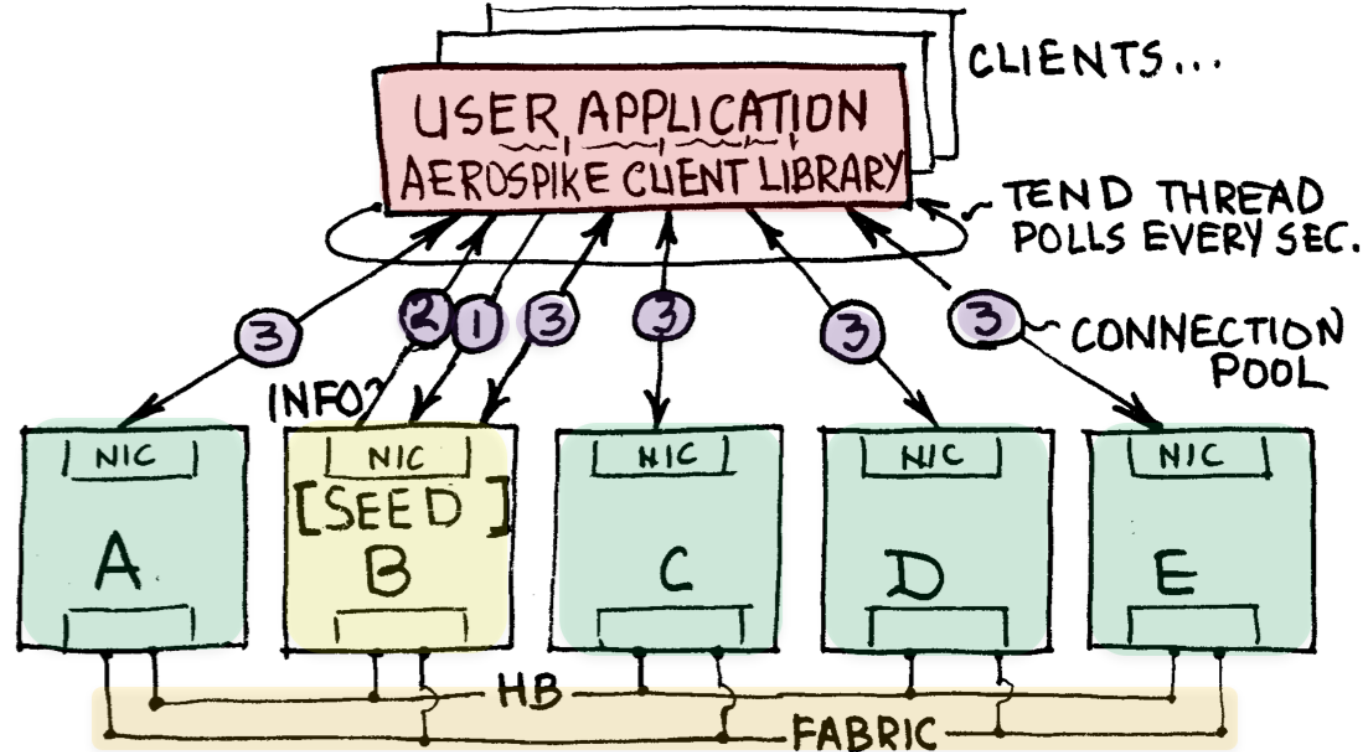
# Aerospike Front End



- User Application binds to Aerospike Client Library (ACL), supplied.
- ACL exposes CRUD operations to user application.
- ACL talks to Aerospike Server over TCP/IP using 'Wire Protocol' – a format for data and cluster state information exchange.



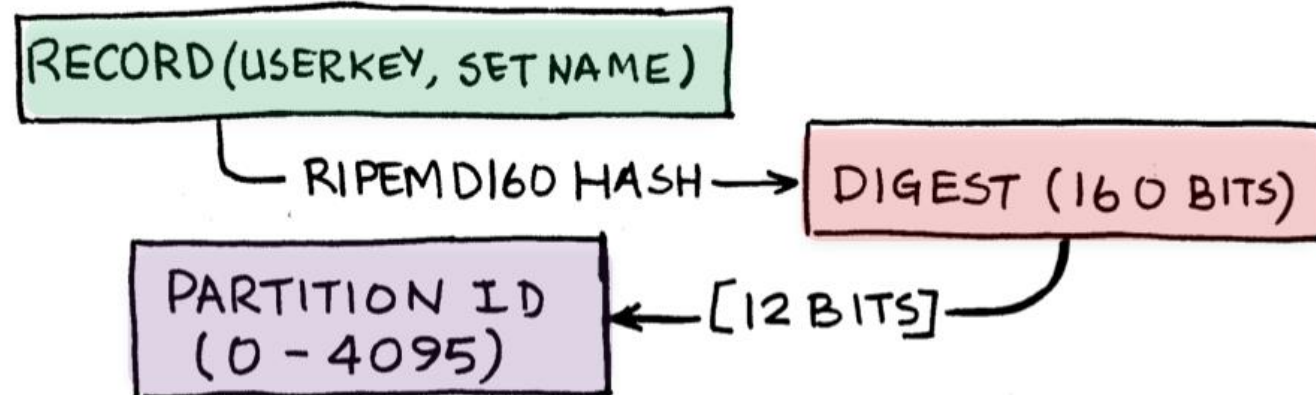
# Aerospike Front End



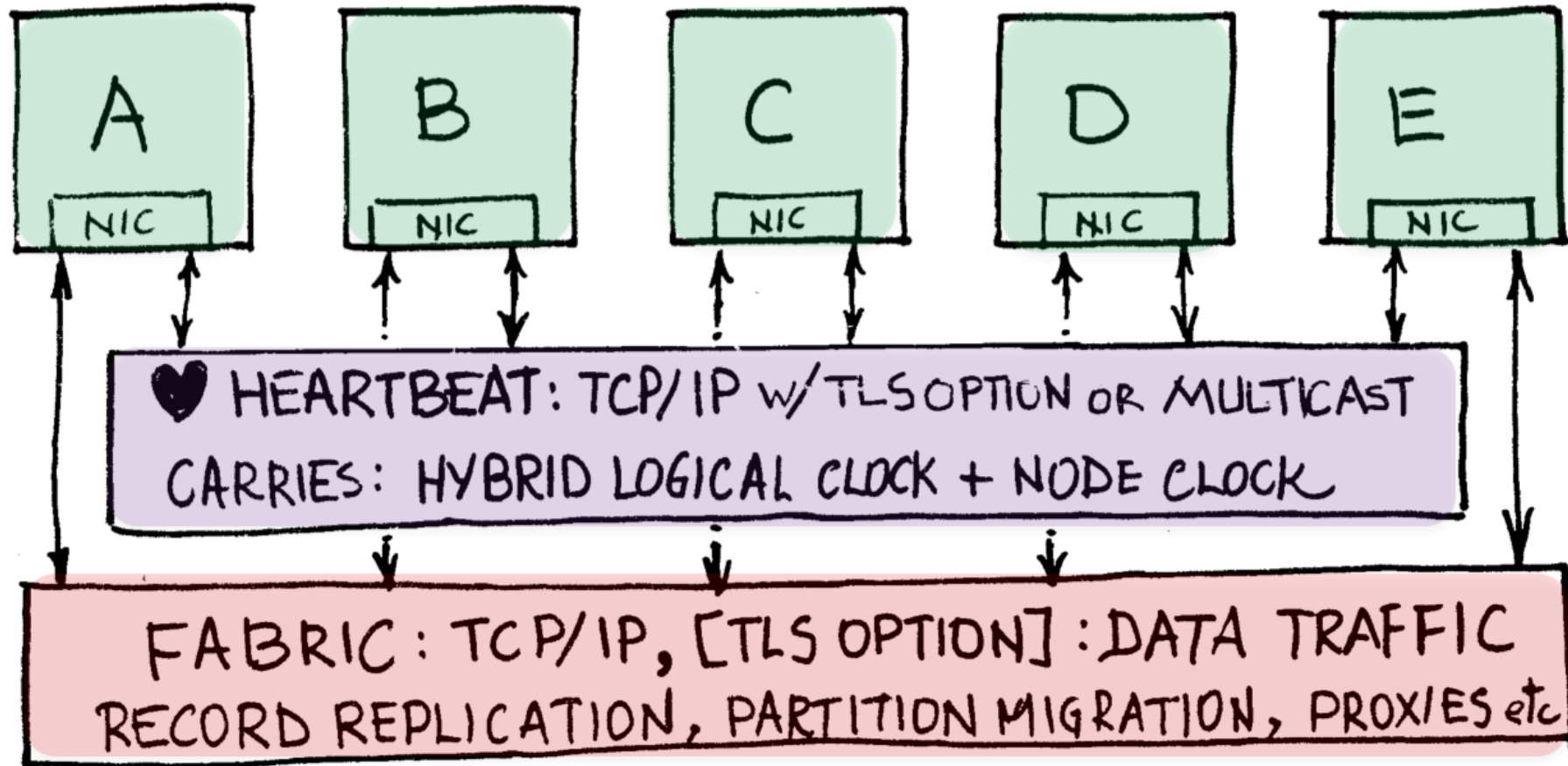
- User Application (via ACL) opens connection to a Seed Node [1].
- Seed node sends cluster nodes connection info to ACL [2].
- ACL opens direct connection pool (~300 connections) to each node.

# Record Digest and Partition ID

- In Aerospike, we have the master record and its replica(s).
- If Replication Factor = 2, we have Master and One Replica.
- Aerospike stores every record in a "Partition".
- **Aerospike distributes records into 4096 partitions.**
- **A record's Digest uniquely identifies its Partition ID.**
- Every node holds Master Records belonging to some partitions and Replica Records belonging to some other set of partitions.
- ACL can find a record's partition using its Digest, as follows:



# Aerospike Cluster Formation



- When a cluster forms (using Paxos protocol), each node is assigned partition ownership.



# Data Distribution

# Partition Table Generation – Legacy Algorithm

- For each partition: Hash each Node Id with the Partition ID & sort hash value in descending order → Node succession list for that partition.
- Deterministic succession list based on Node ID.

PARTITION TABLE

PARTM ID	R1	R2	R3	R4	R5
0	B	D	E	A	C
1	E	C	A	D	B
⋮	⋮	⋮	⋮	⋮	⋮
4094	C	B	A	E	D
4095	D	E	A	B	C

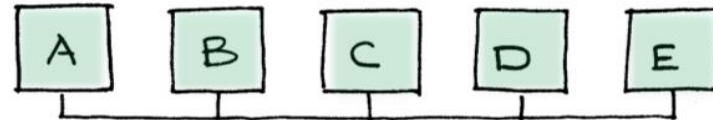
SUCCESSION LIST FOR PARTITION 1

- **Node ID** is Fabric Port + MAC address of NIC card *or user assigned (recommended)*.

# Partition Table

- Cluster forms using Paxos algorithm and a Partition Table is generated.
- Each row in the Partition Table is the Succession List for that partition.

5 NODE CLUSTER, REPLICATION FACTOR (RF)=2



PARTITION TABLE

PARTM ID	R1	R2	R3	R4	R5
0	B	D	E	A	C
1	E	C	A	D	B
:	:	:	:	:	:
4094	C	B	A	E	D
4095	D	E	A	B	C

# Partition Map

- First RF # of entries in the succession list yields the Partition Map.

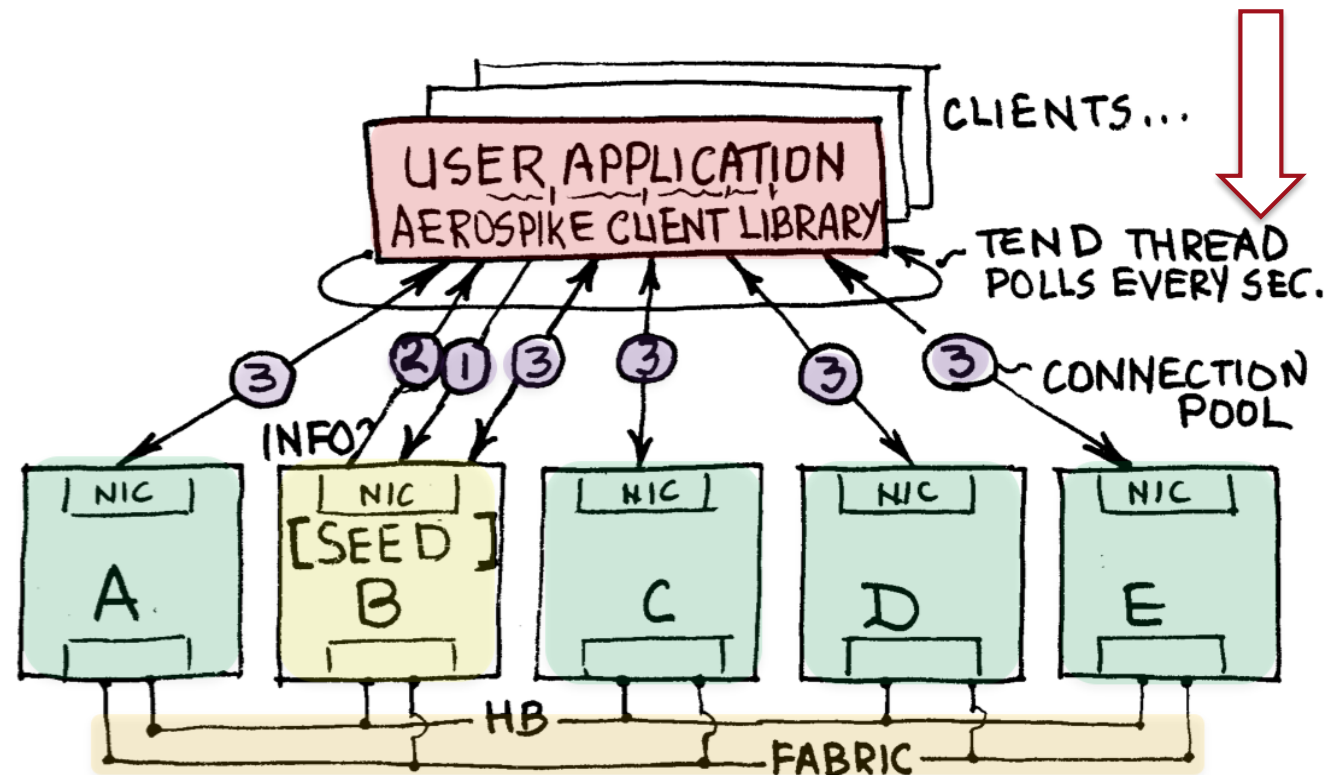
## PARTITION TABLE

PARTM ID	R1	R2	R3	R4	R5
0	B	D	E	A	C
1	E	C	A	D	B
⋮	⋮	⋮	⋮	⋮	⋮
4094	C	B	A	E	D
4095	D	E	A	B	C

MASTER REPLICAS  
PARTITION MAP (RF=2)

# Partition Map

- Every second, **ACL tend thread queries each node for Partition Version.**
- Cluster change triggers Paxos re-clustering and bumps Partition Version.
- When ACL detects change in Partition Version, it **re-builds the Partition Map** by querying each node for its Master and Replica(s) ownership.





# Partition Map - Losing a Node

5 NODE CLUSTER, REPLICATION FACTOR (RF)=2



- When a node is lost (e.g., node C), succession list moves left.

- Some partition examples below:

- Partition 1:** C was Replica, A becomes new Replica. Partition data migrates from Master E to A. Two copies of data restored upon completion of migration.
- Partition 4094:** C was Master, **Replica B gets promoted to new Master.** Typically, B will have full data. A becomes the new Replica. Partition data will be migrated from from B to A.

PARTITION TABLE

PARTM ID	R1	R2	R3	R4	R5
0	B	D	E	A	<del>X</del>
1	E	<del>X</del>	A	D	B
⋮	⋮	⋮	⋮	⋮	⋮
4094	<del>X</del>	B	A	E	D
4095	D	E	A	B	<del>X</del>

MASTER REPLICATION FACTOR (RF=2)  
PARTITION MAP

# Partition Map - Node Returns

5 NODE CLUSTER, REPLICATION FACTOR (RF)=2



PARTITION TABLE

PARTY ID	R1	R2	R3	R4	R5
0	B	D	E	A	<del>X</del>
1	E	<del>X</del>	A	D	B
:	:	:	:	:	:
4094	<del>X</del>	B	A	E	D
4095	D	E	A	B	<del>X</del>

MASTER REPLICATION  
PARTITION MAP (RF=2)



PARTITION TABLE

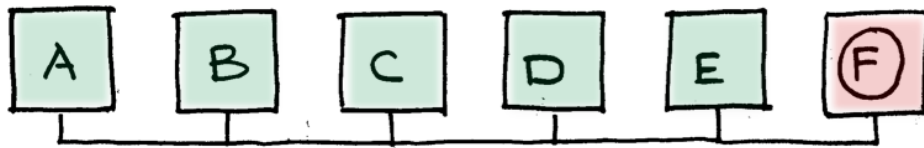
PARTY ID	R1	R2	R3	R4	R5
0	B	D	E	A	C
1	E	C	A	D	B
:	:	:	:	:	:
4094	C	B	A	E	D
4095	D	E	A	B	C

MASTER REPLICATION  
PARTITION MAP (RF=2)

- When node C rejoins the cluster with same Node ID, C will come back in the original position in the succession list. (C will start taking new replica updates if in Master or Replica position.)
- Existing records' updates are identified using record metadata and changed records migrate between C and adjacent node, as needed. ([Rapid Rebalance – Enterprise Edition only](#)).

# Partition Map - Adding a Node

6 NODE CLUSTER, REPLICATION FACTOR (RF)=2



- New node F is added to the cluster.
- F may land anywhere in a partition's succession list.

PARTITION TABLE

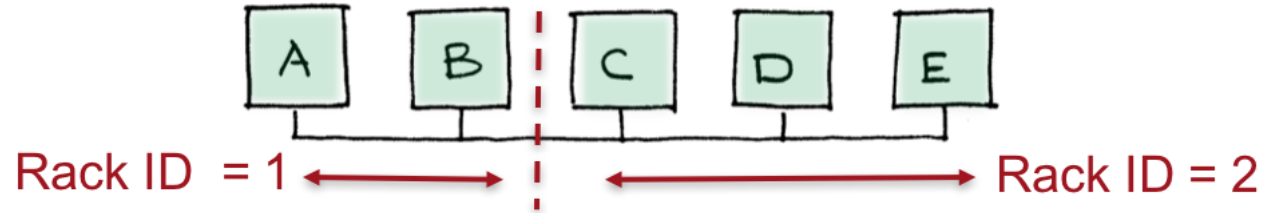
PARTM ID	R1	R2	R3	R4	R5	R6
0	B	F	D	E	A	C
1	F	E	C	A	D	B
⋮	⋮	⋮	⋮	⋮	⋮	
4094	C	B	F	A	E	D
4095	D	E	A	B	C	F

MASTER REPLICATION  
PARTITION MAP (RF=2)

- **Partition 0:** Node F joins as Replica, B remains Master and fills data into F ([Fill Migration](#)).
- **Partition 1:** Node F joins as Master, E continues to act as Master till it finishes filling data into F. When this fill migration completes, F becomes new Master ([Master-Handoff](#)).

# Rack Aware Configuration (EE only)

5 NODE CLUSTER, REPLICATION FACTOR (RF)=2



PARTITION TABLE

PARTY ID	R1	R2	R3	R4	R5
0	B	D	E	A	C
1	E	C	A	D	B
⋮	⋮	⋮	⋮	⋮	⋮
4094	C	B	A	E	D
4095	D	E	A	B	C

MASTER REPLICATION  
PARTITION MAP (RF=2)

Adjust for Rack Awareness



PARTITION TABLE

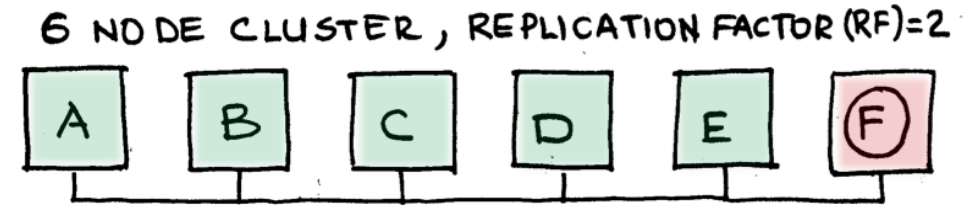
PARTY ID	R1	R2	R3	R4	R5
0	B	D	E	A	C
1	E	C	A	D	B
⋮	⋮	⋮	⋮	⋮	⋮
4094	C	B	A	E	D
4095	D	E	A	B	C

MASTER REPLICATION  
PARTITION MAP (RF=2)

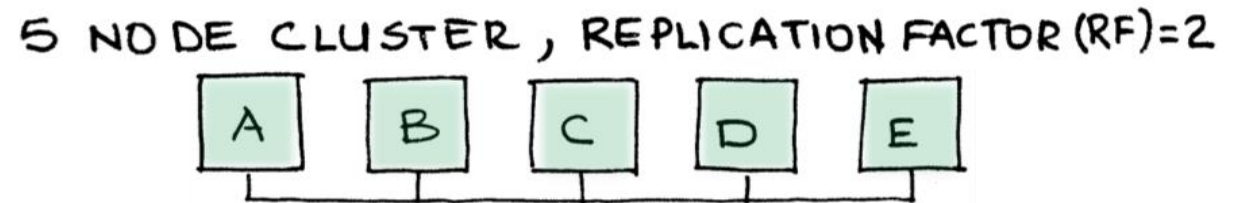
- Nodes A & B declared with Rack ID = 1. Nodes C, D & E declared with Rack ID = 2.
- Partition Map / Succession List:** Aerospike modifies the Partition Table such that Master and Replica are always on different racks.
- Note:** Partition 1 – C & A swapped. Partition 4095 – E & A swapped. Actual algorithm has additional complexities, e.g., handling uniform balancing etc.

# Cluster Capacity

- We have a 6 node cluster:  $4096/6 = \sim 683$  master partitions per node.



- **Lose node F** → 5 node cluster:  $4096/5 = \sim 819$  master partitions per node.

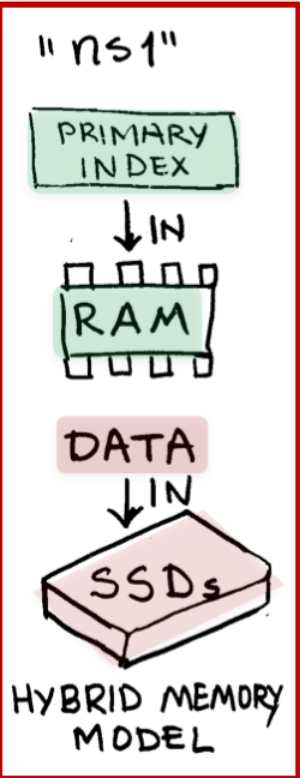
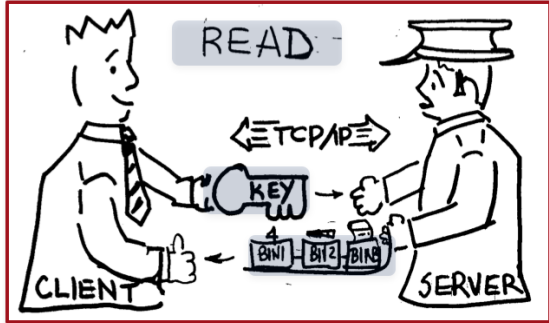
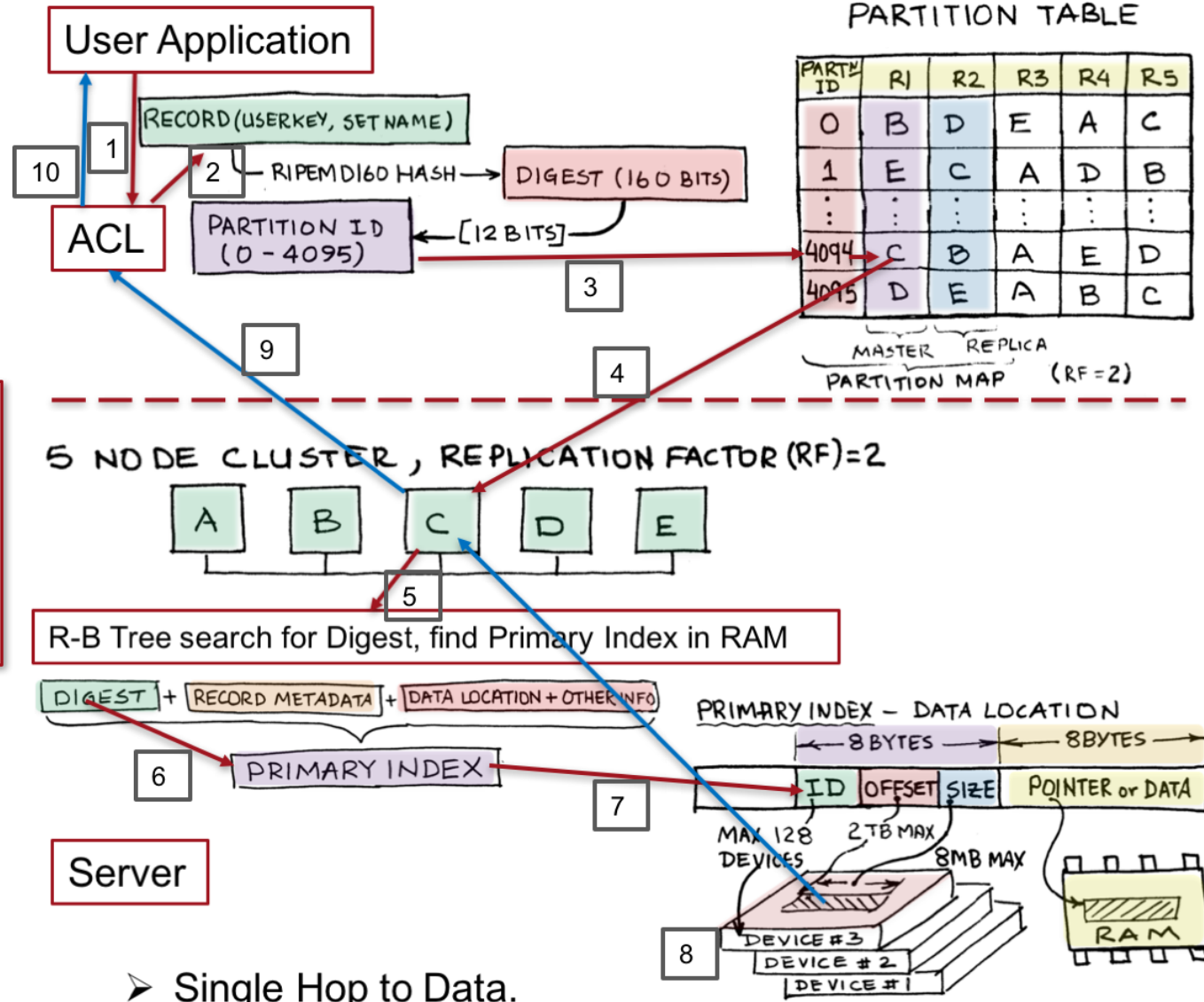


- For a given node capacity (RAM, DISK), as cluster size decreases, each node is responsible for higher number of partitions, more data.
- When a node is taken out (e.g. rolling upgrade), remaining nodes should be able to still store 2 copies of the data after cluster re-balances automatically.
- When cluster starts hitting capacity limits, add capacity by adding node(s) to the cluster.
- **Adjust cluster size with automatic data re-distribution and rebalancing.**



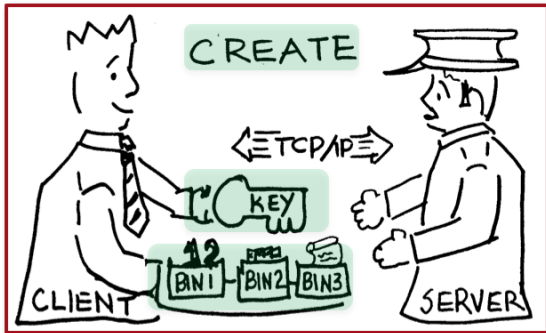
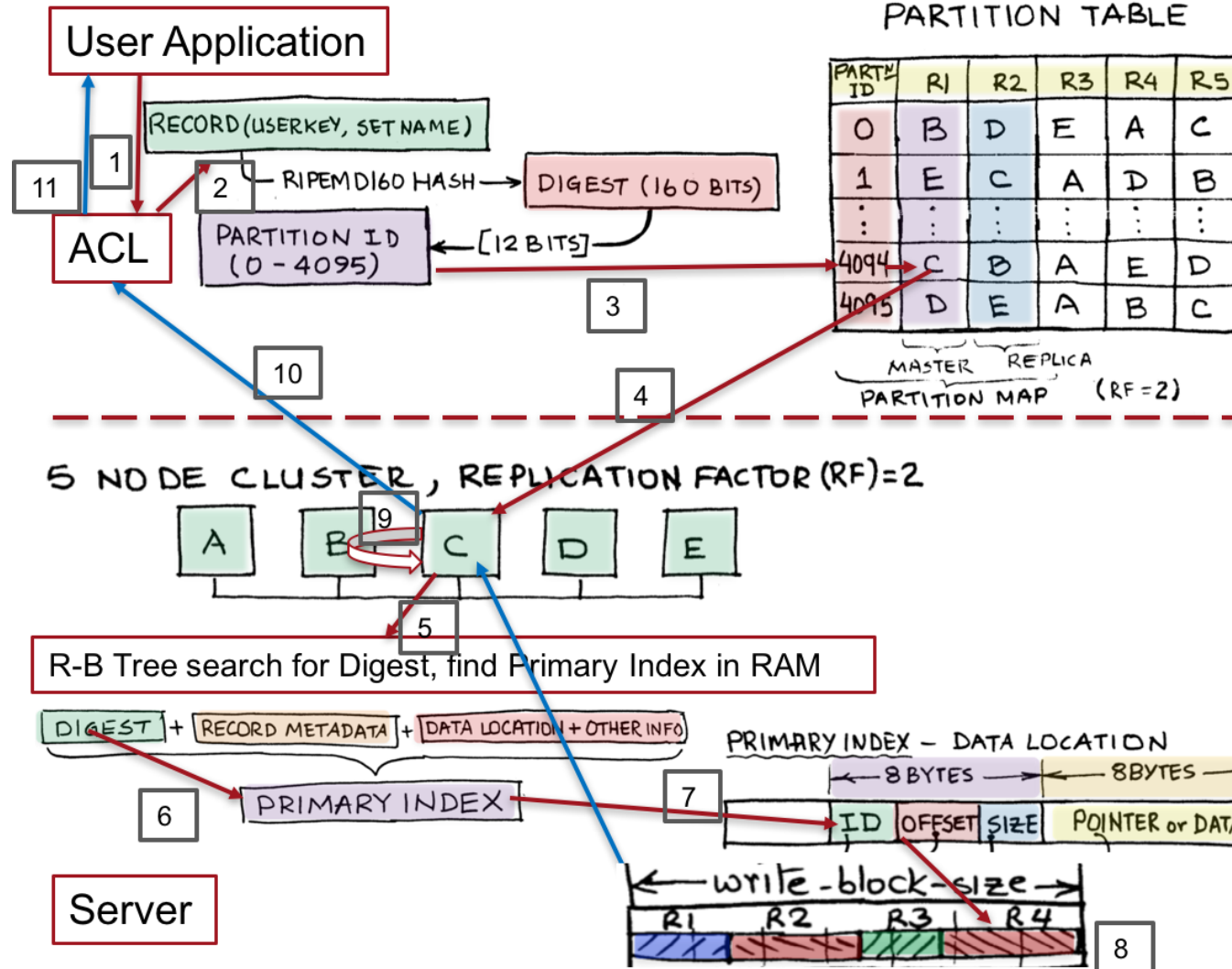
# Read and Write Transactions

# Read Transaction – Hybrid Memory Storage Example



➤ Single Hop to Data.

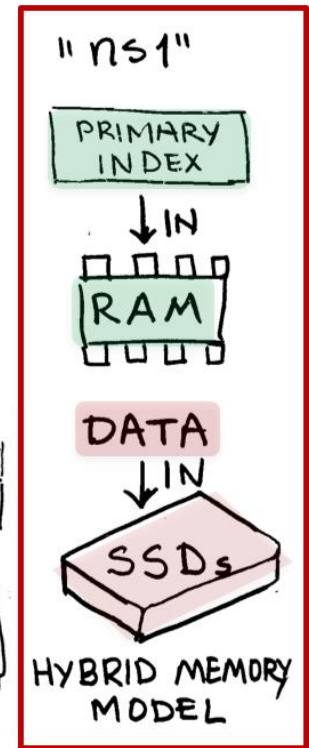
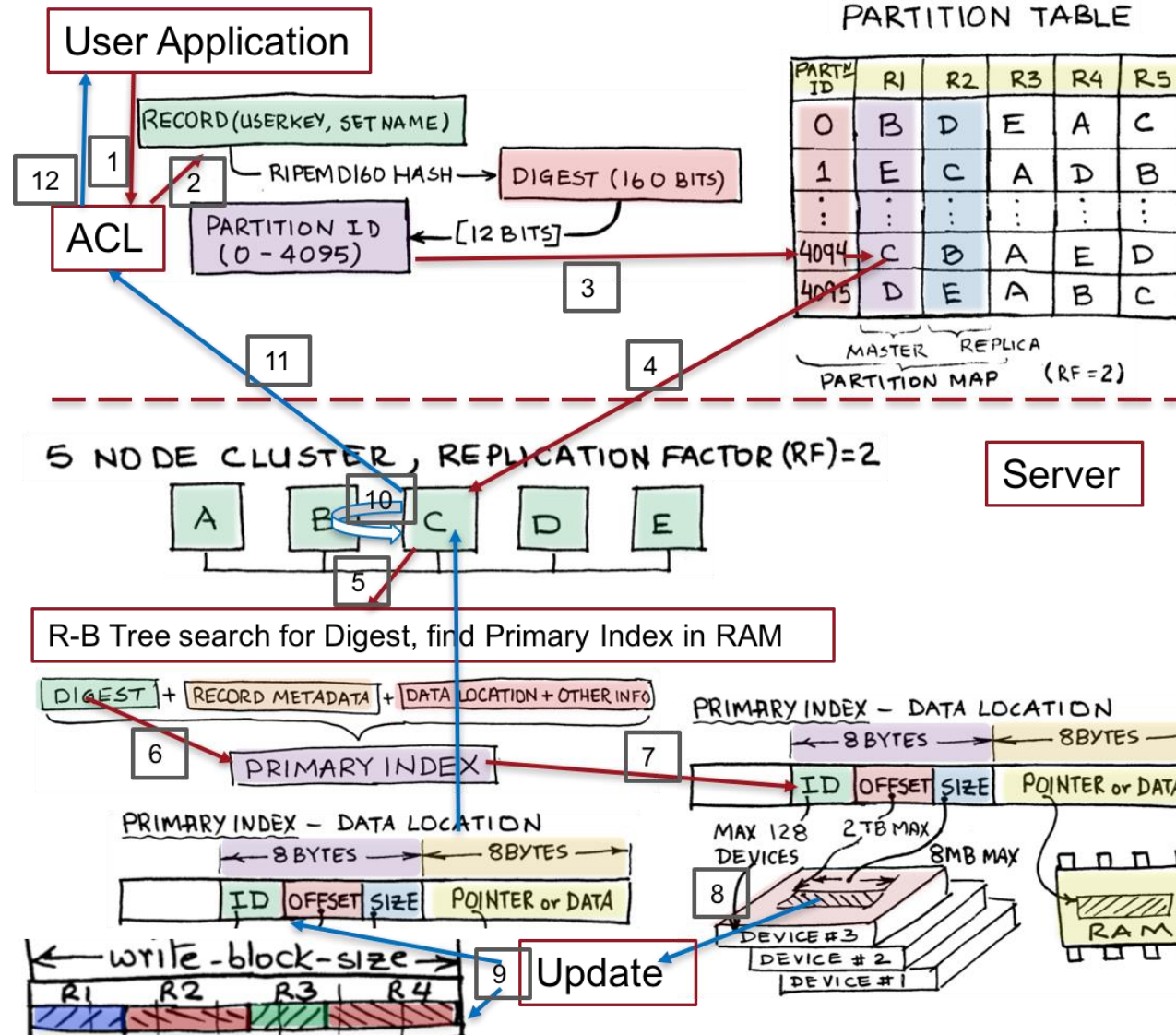
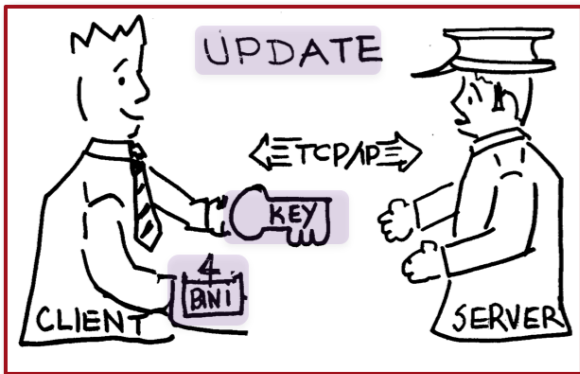
# Create Transaction – Hybrid Memory Storage Example



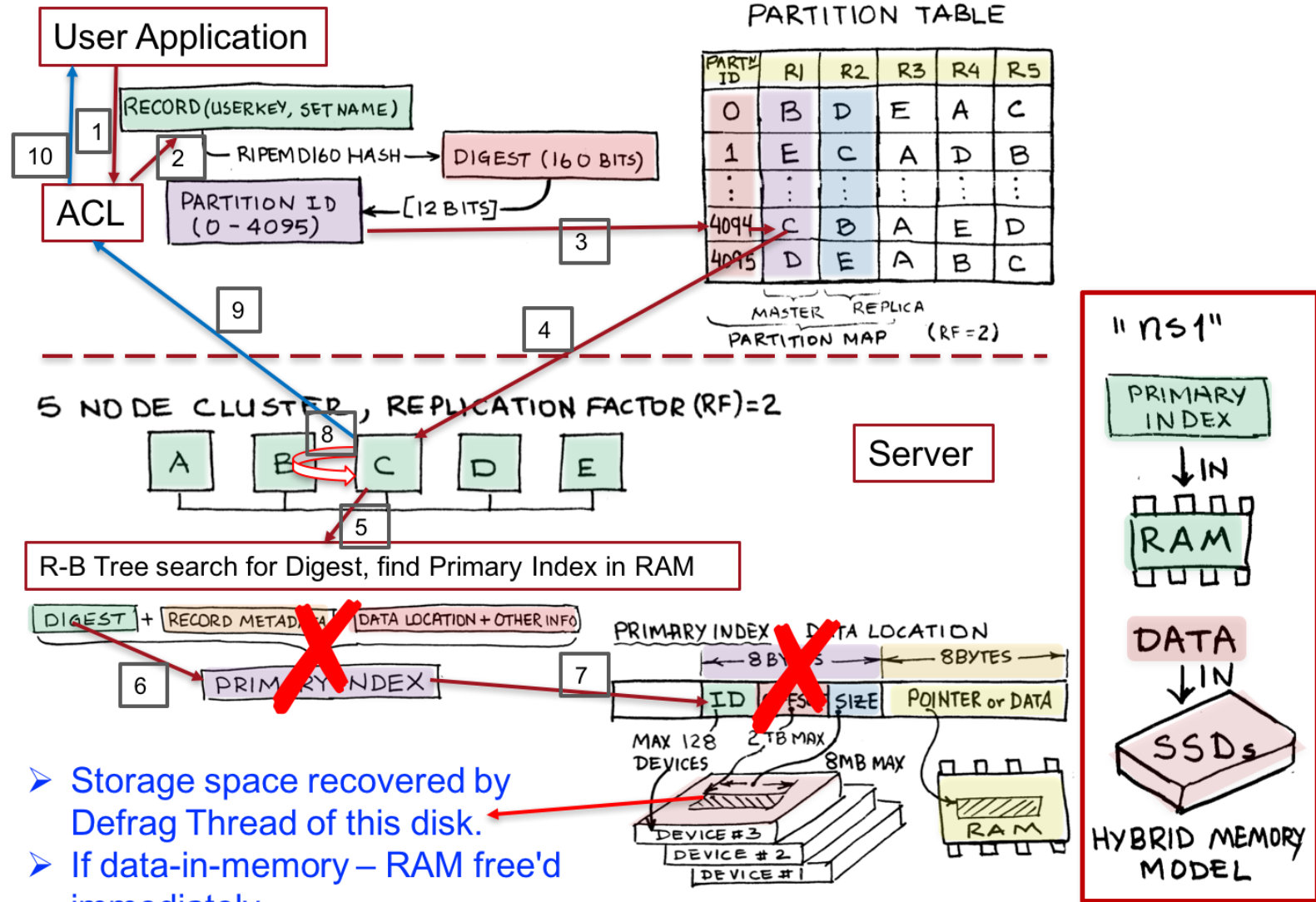
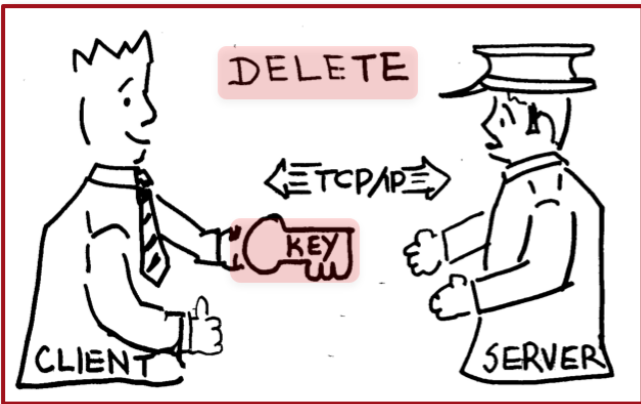
- Writes go to write-block-buffer in RAM, asynchronously flushed to device. Also replicate to replica node (B), then acknowledge client.



# Update Transaction – Hybrid Memory Storage Example

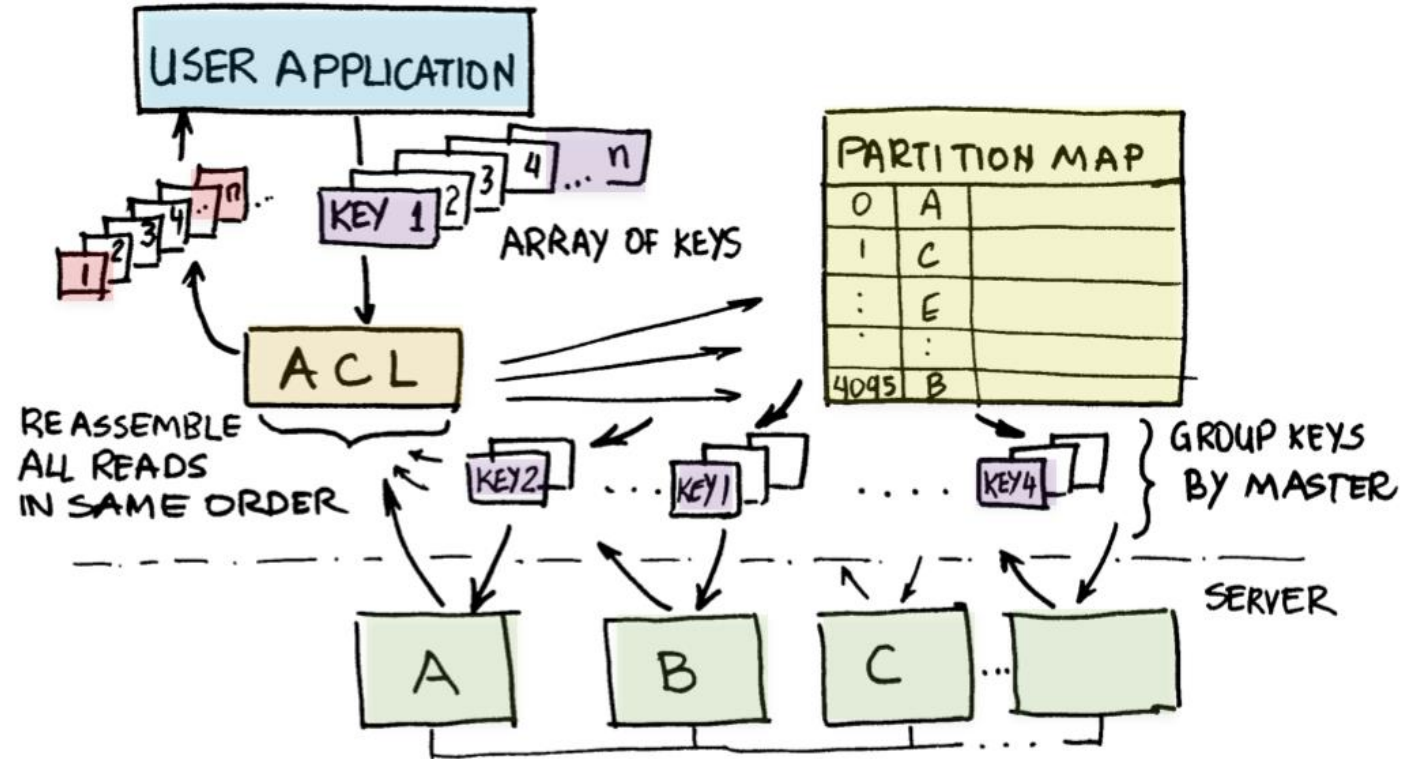


# Delete Transaction – Hybrid Memory Storage Example



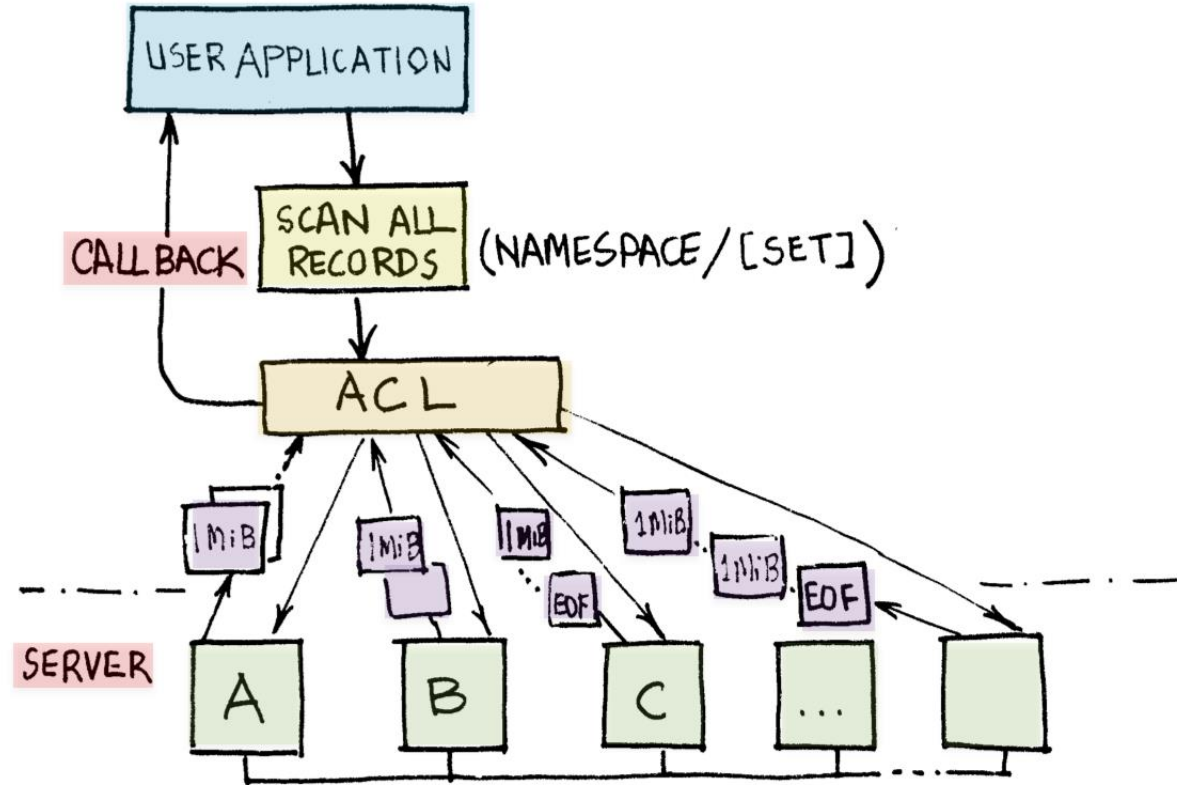
- Storage space recovered by Defrag Thread of this disk.
- If data-in-memory – RAM free'd immediately.

# Batch Index Reads



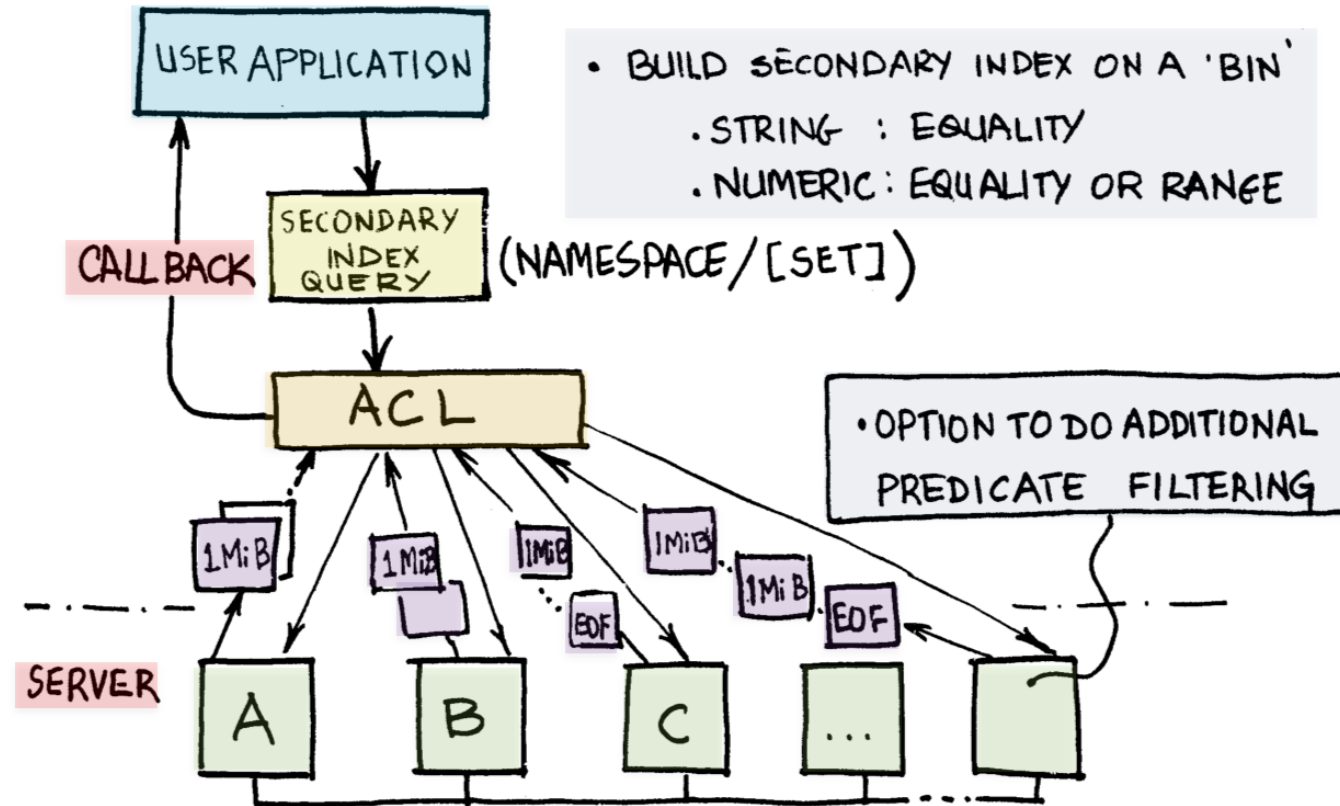
- Client requests to read multiple records using an array of keys.
- ACL returns the result in the same order as the keys in a blocking call.
- **Client must have adequate RAM to hold the entire expected result set.**

# Scans



- Results are **pipelined** in 1 MiB ( $1 \times 2^{20}$  Bytes) buffers.
- Client consumes results as they come, in a callback function.
- Scan ends when each node has returned EOF. Option to **Fail-On-Cluster-Change**.
- Option to do Predicate Filtering or run User Defined Function (**UDF**) on server on each scanned record.

# Secondary Index Query



- Results are **pipelined** in 1 MiB ( $1 \times 2^{20}$  Bytes) buffers.
- Client consumes results as they come, in a callback function.
- SI Query ends when each node has returned EOF. Option to **Fail-On-Cluster-Change**.
- Option to **aggregate** results using a User Defined Function (**Stream UDF**) on entire result set.

AEROSPIKE