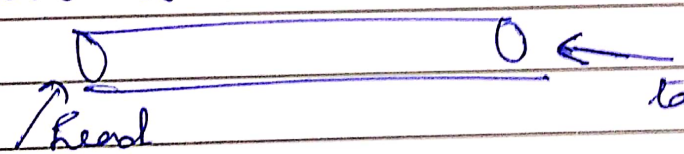
DESIGN A CACHE SYSTEM:-

Use-case:

1. LRU Cache
   - 1.1 Add to Cache | Insert to cache
   - 1.2 Read from Cache
   - 1.3 delete from Cache | Replace
2. Eviction Policy from cache
3. distributed System
4. Concurrent System
5. 100% Availability System
6. ACID Property

CONSTRAINST

1. Space for HashTable for hash existence for efficient search
2. Insert to HashTable + Insert to queue



3. Random Replacement Eriction Policy
   W- Tiny URL Eriction Policy

HIGH LEVEL DESIGN:-

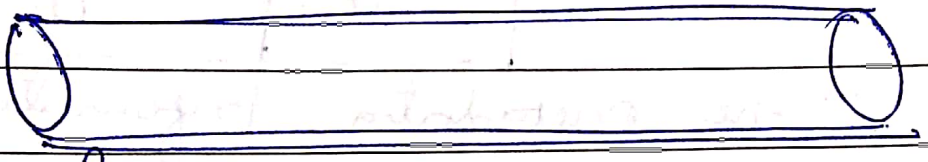
CN1 — M1 — Slave (M2, M3)

CN2 — M2 — (M1, M3)

CN3 — M3 — (M1, M2)

- ~~Non distributed~~ Architecture
1. Sharding
2. Non-Shardable data Architecture
3. Partition
4. Master-Slave Replication model

Hash Table
For faster Occurrence Retrieval



Queue for Insertion and Eviction

Read

Key → hash generate → decide which Node to avoid skewed distribution of hash

Hash + Action → [Read / Write] → Node decision

Read — Replicated Nodes
Write = Master Node

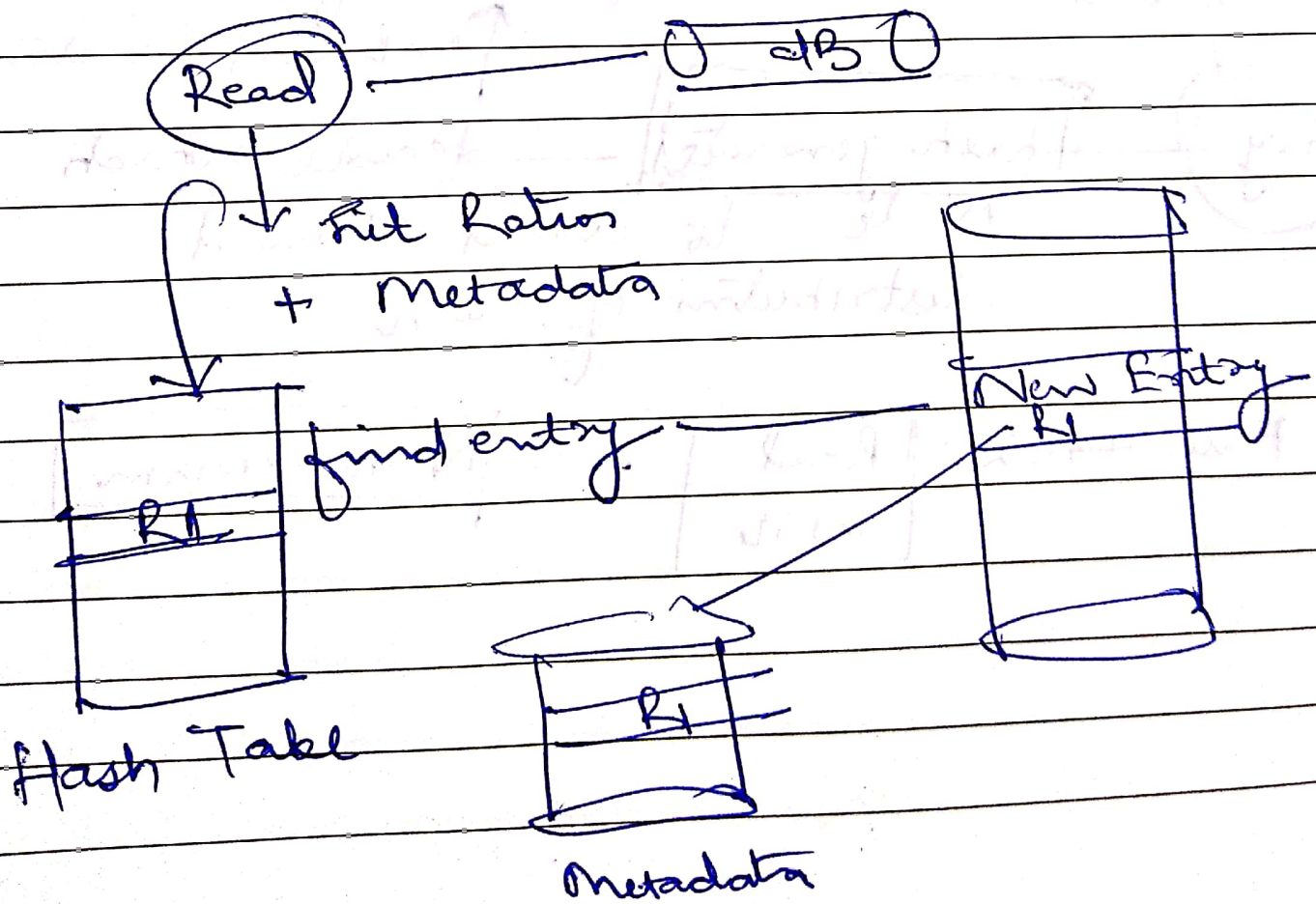
SnapShot Versioning — dirty Read allow

Write — Record level lock
to ensure Atomic operation
+ Easy Rollback
transaction

1. LRU Cache :-

Cache Entry — Cache hit Ratio | Entry | Entry Set

+ Cache Metadata Information



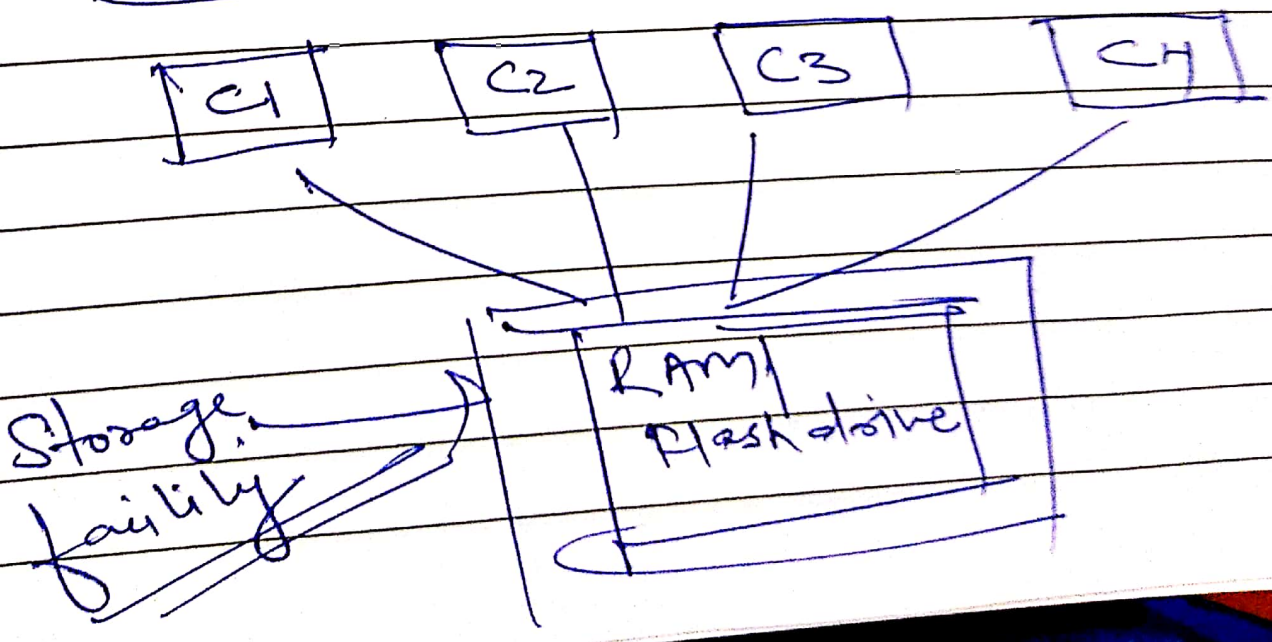
b. If R1 available read from Hashtable to find ratio of availability

c. Replace Cache victim based on Cache hit Ratio + Metadata

2. Eviction Policy
   1. TW- Tiny LRU
   2. Garbage Collection Mechanism
   3. Expiration of Record manually set during write

3. Concurrent System
   Key → hash + Action
   Master — write
   Replicated Slave — Read
   Snapshot Versioning fetch/Read
   Non Shorable Architecture

4. distributed System + 100% Availability



5. Eviction Policy

Aging —— default expiration

When update content extent default expiration