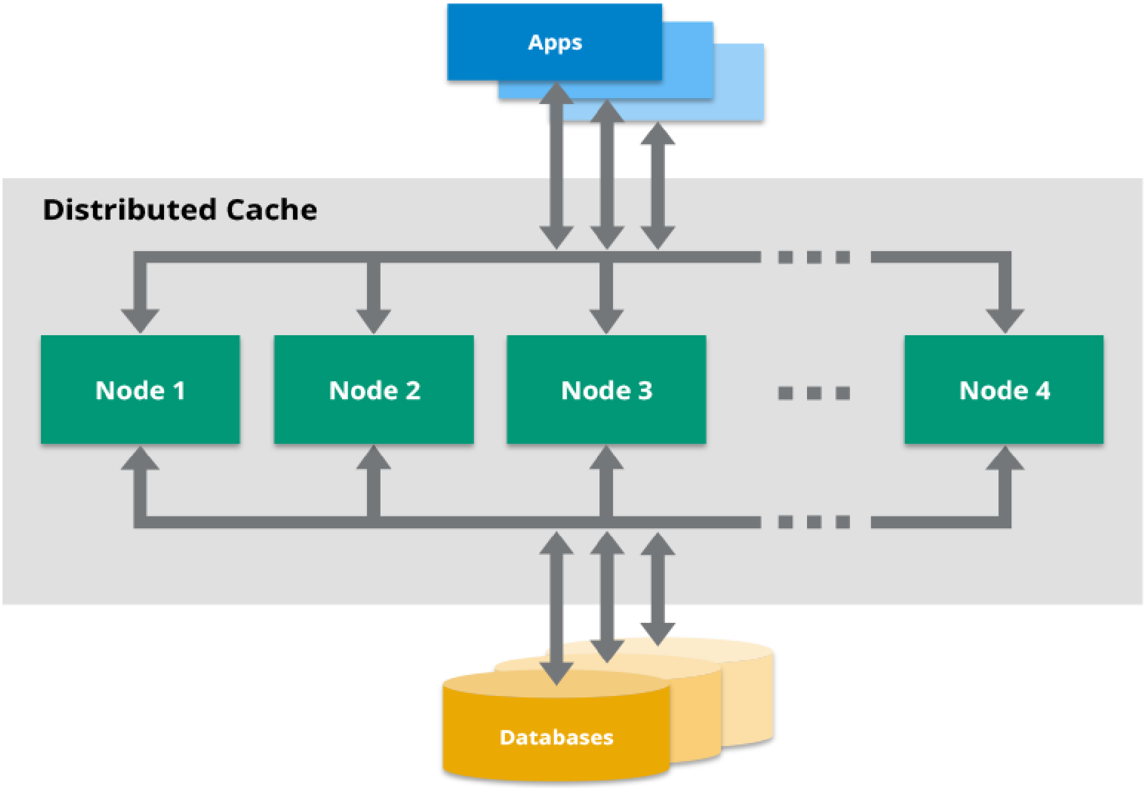
Distributed Caching

What

A distributed cache is a system that pools together the random-access memory (RAM) of multiple networked computers into a single in-memory data store used as a data cache to provide fast access to data.

The distributed architecture allows incremental expansion/scaling by adding more computers to the cluster, allowing the cache to grow in step with the data growth.

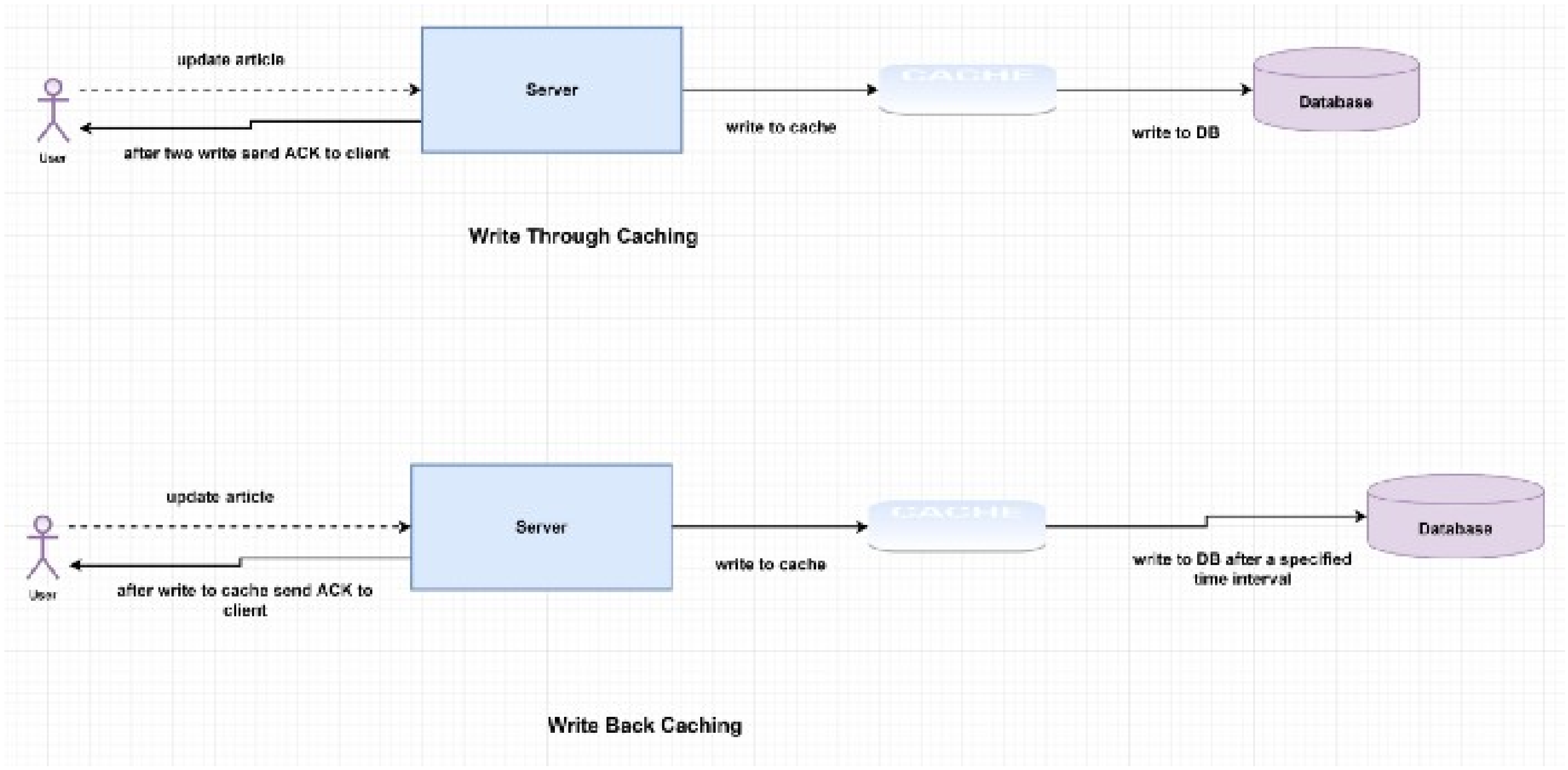


Cache invalidation and cache eviction are basic problems of caching, whether is local or distributed.

Cache Invalidation

It does require some maintenance for keeping cache coherent with the source of truth.

If the cache contains the previous data, then that is called stale data.



Cache Eviction

A cache eviction algorithm is a way of deciding which element to evict when the cache is full.

The following are some of the most common cache eviction policies:

- First In First Out (FIFO)
  The cache evicts the first block accessed first without any regard to how many times it was accessed before.
- Last In First Out (LIFO)

The cache evicts the block accessed most recently first without any regard to how many times it was accessed before.

- **Least Recently Used (LRU)**
  Discards the least recently used items first.
- Most Recently Used (MRU)
  Discards, in contrast to LRU, the most recently used items first.
- Least Frequently Used (LFU)
  Counts how often an item is needed. Those that are used least often are discarded first.

Case Study - Redis

A Redis Enterprise cluster is composed of identical nodes. Redis Enterprise architecture is made up of a management path and data access path.

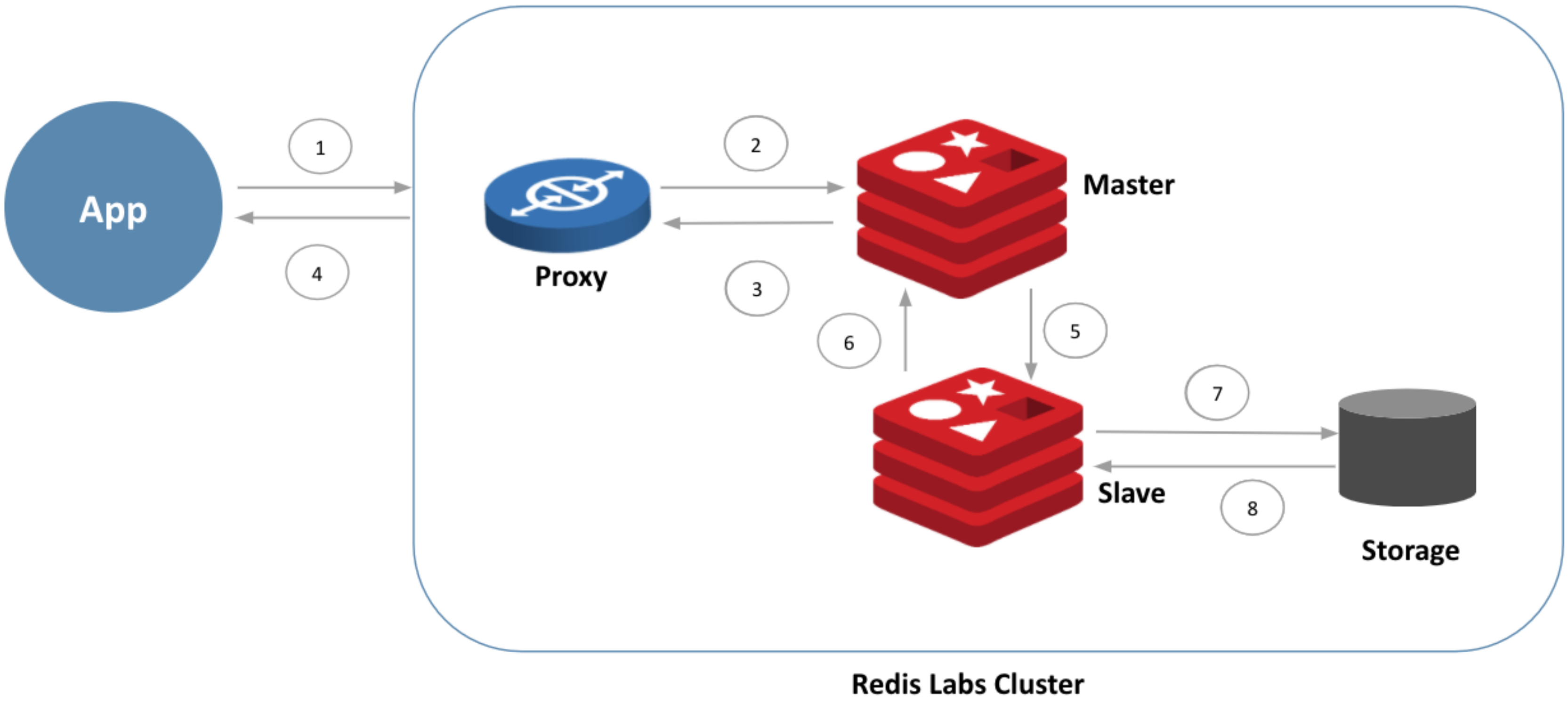
- **Management path** includes the cluster manager, proxy and secure REST API/UI for programmatic administration. In short, **cluster manager** is responsible for orchestrating the cluster, placement of database shards as well as detecting and mitigating failures. **Proxy** helps scale connection management.

- **Data Access path** is composed of **master** and **replica** Redis shards. Clients perform data operations on the master shard. Master shards maintain replica shards using the in-memory replication for protection against failures that may render master shard inaccessible.

Here, we focus on the Data Access path.

Any updates that are issued to the database are typically performed with the following flow shown below;

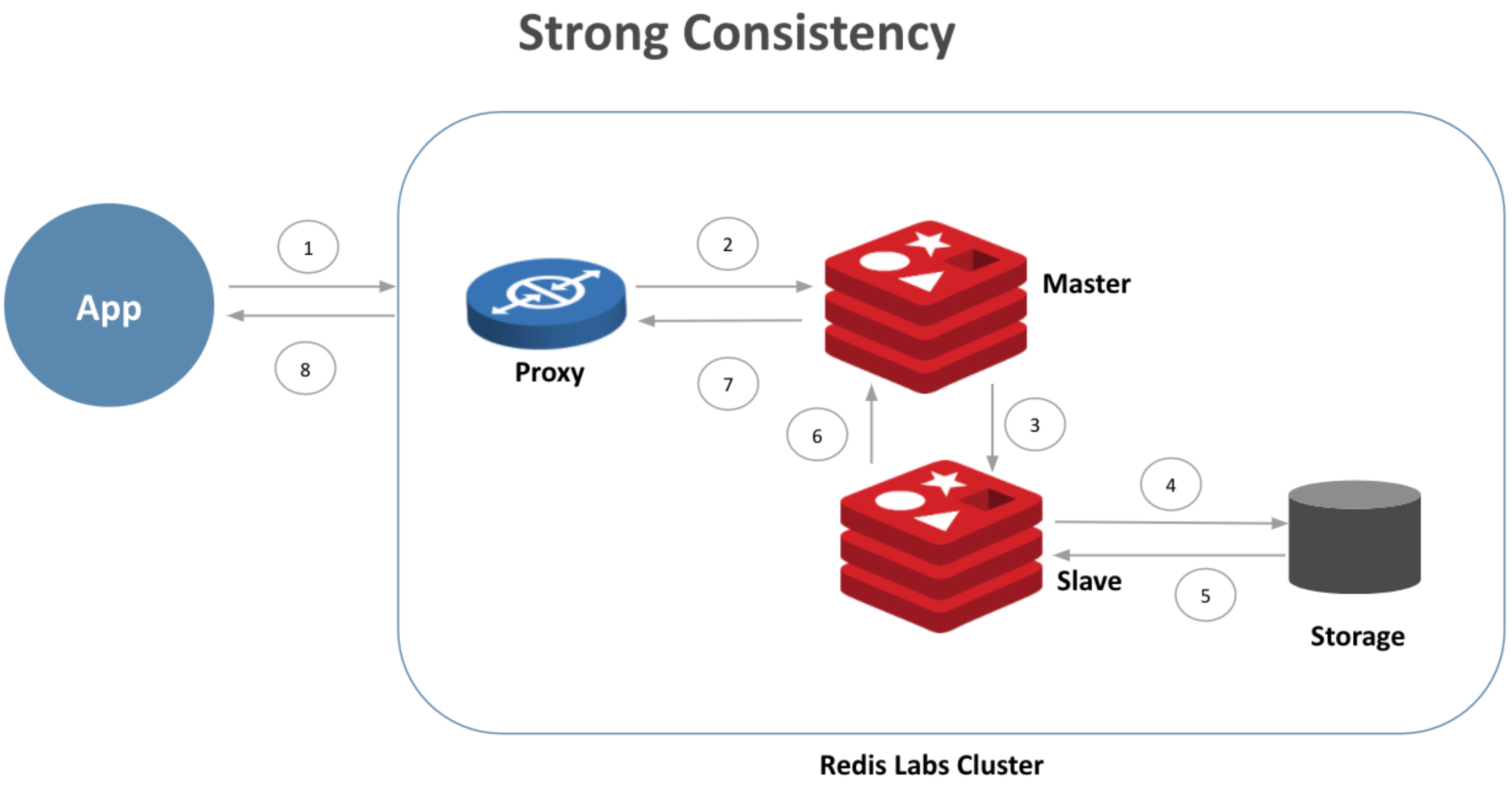
Weak Consistency



1. Application issues a write,
2. Proxy communicates with the correct master “shard” in the system that contains the given key,
3. The acknowledgment is sent to proxy once the write operation completes
4. The proxy sends the acknowledgment back to the application.

Independently, the write is communicated from master to replica and replication acknowledges the write back to the master. These are steps 5 and 6.
Independently, the write to a replica is also persisted to disk and acknowledged within the replica. These are steps 7 and 8.

Redis uses the WAIT command in order to achieve strong consistency as below.



More details at https://medium.com/nerd-for-tech/understanding-redis-in-system-design-7a3aa8abc26a

References*

https://medium.com/rtkal/distributed-cache-design-348cbe334df1
https://hazelcast.com/glossary/distributed-cache/
https://docs.redis.com/latest/rs/concepts/