Designing Concurrent Distributed Sequence Numbers for Elasticsearch

Boaz Leskes
@bleskes

Sequence numbers - Why[TF]?

Document level versioning

```
PUT tweets/tweet/605260098835988500
{
    "created_at": "Mon Jun 01 06:30:27 +0000 2015",
    "id": 605260098835988500,
    "text": "Looking forward for awesomeness #bbuzz”,
    "user": {
        "name": "Boaz Leskes",
        "screen_name": "bleskes",
    }
}
```

```
{
   "_index": "tweets",
   "_type": "tweet",
   "_id": "605260098835988500",
   "_version": 3,
   …
}
```

Multiple doc updates

```
PUT tweets/tweet/605260098835988500
{
    …
    "text": "…",
    "user": {
        "name": "Boaz Leskes"
        "screen_name": "bleskes"
    }
}
```

```
PUT tweets/tweet/426674590560305150
{
    …
    "text": "…",
    "user": {
        "name": "Boaz Leskes",
        "screen_name": "bleskes",
```

```
PUT tweets/tweet/605260098835988500
{
    …
    "text": "…",
    "user": {
        "name": "Boaz Leskes",
        "screen_name": "bleskes",
    },
    "retweet_count": 1
}
```

Multiple doc updates - with seq#

```
PUT tweets/tweet/605260098359885002
{
    …
    "text": "…",
    "user": {
        "name": "Boaz Leskes"
        "screen_name": "bleskes"
    }
}'
```

```
PUT tweets/tweet/426674590560305150 1
{
    …
    "text": "…",
    "user": {
                                    Leskes",
                                    "bleskes",
```

```
PUT tweets/tweet/605260098359885003
{
    …
    "text": "…",
    "user": {
        "name": "Boaz Leskes",
        "screen_name": "bleskes",
    },
    "retweet_count": 1
}
```

Sequence # == ordering of changes

- meaning they can be sorted, shipped, replayed

RFC: Changes API #440

Open clintongormley opened this issue on Nov 1, 2014 · 12 comments

clintongormley commented on Nov 1, 2014

The changes API allows a user to "follow" an index. First it returns all of the existing documents, much like the scroll API, but then it allows the user to continue listening for newer changes.

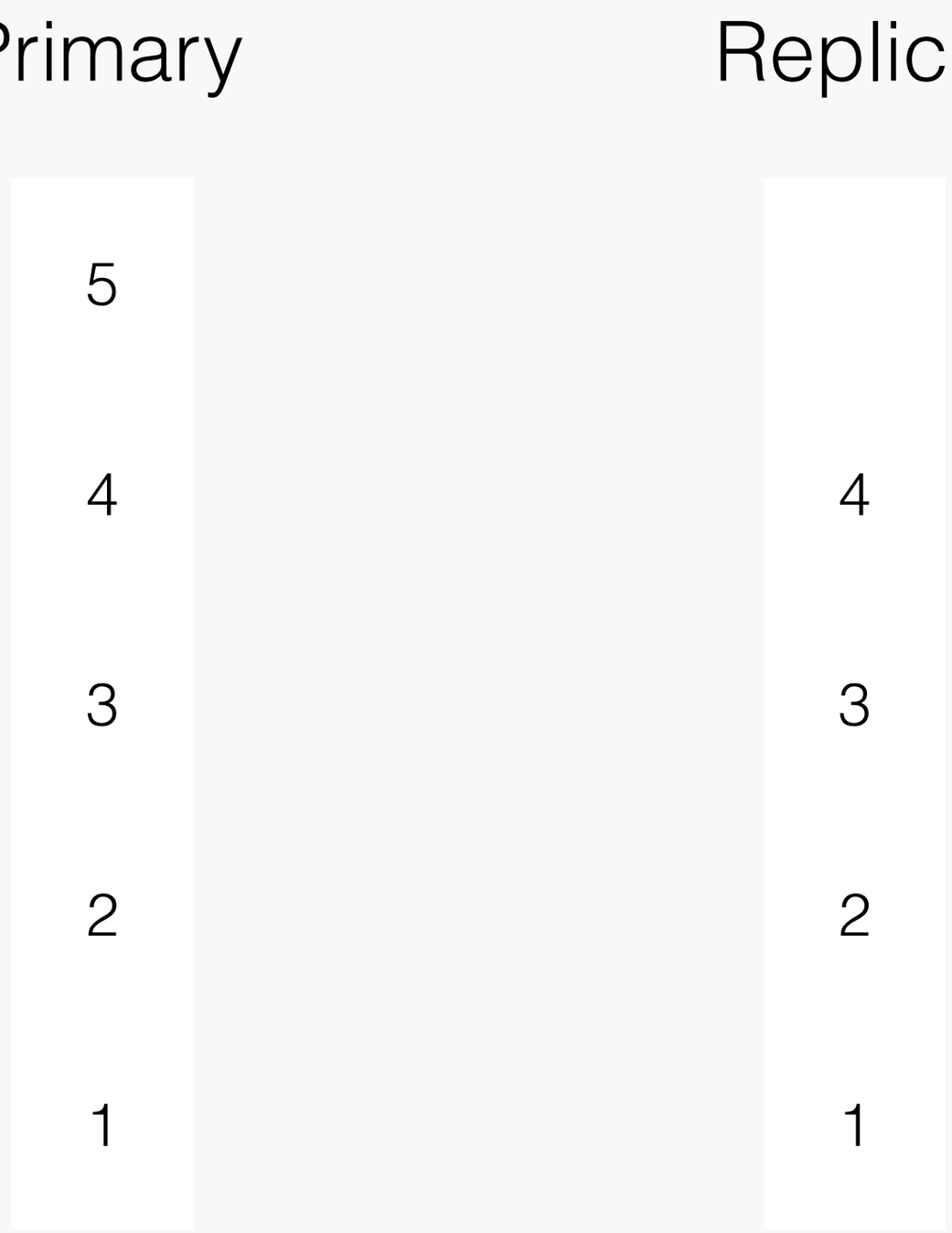
Note: The changes API depends on sequence IDs (#125).

Registering a listener

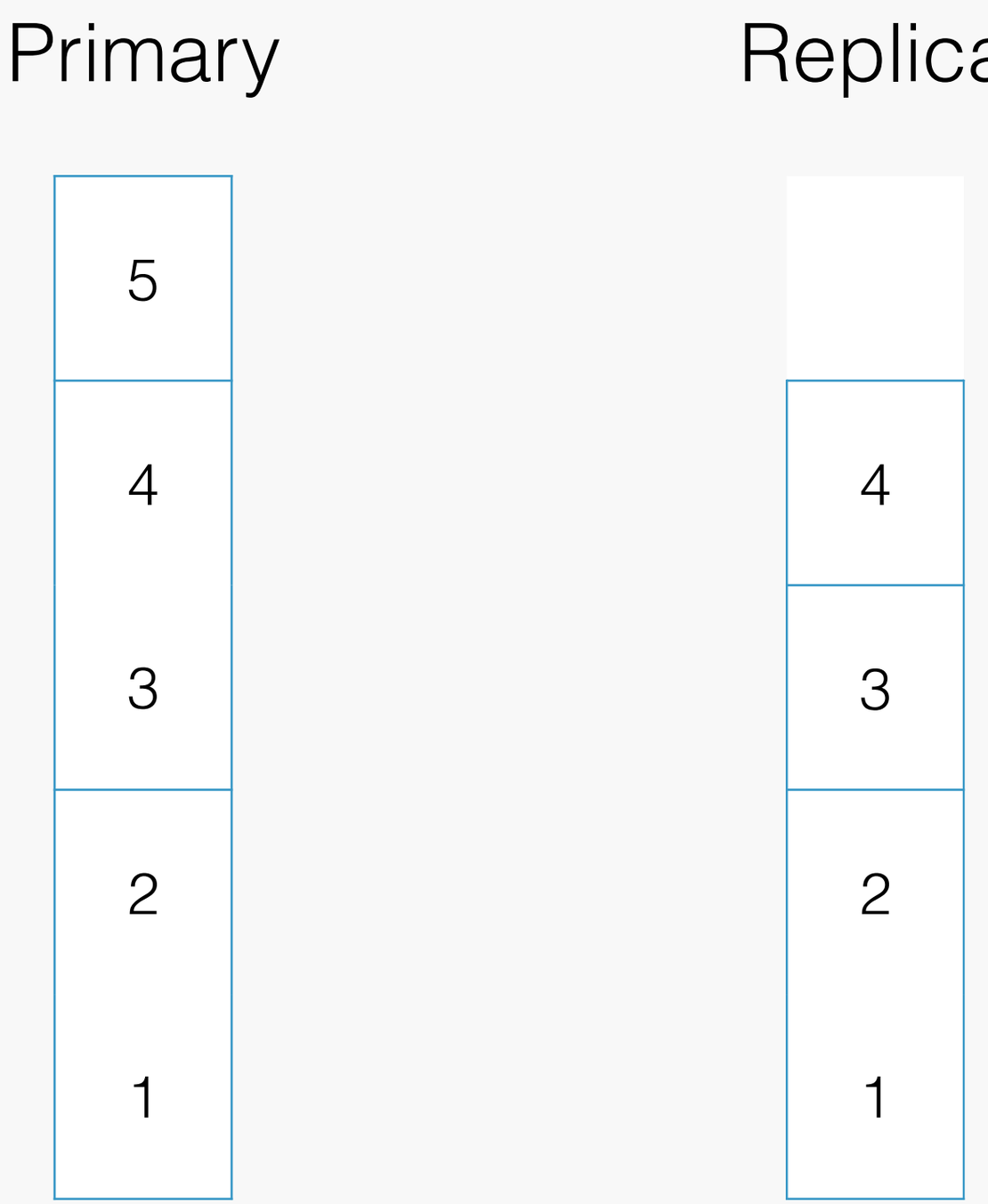
First, the user registers their `_changes` listener (defaults settings shown for clarity):

```
POST /{indices}/{types}/_changes/register
{
  "size": 10,           # return 10 results per shard
  "filter": {           # return all results in the index
    "match_all": {}
  },
  "_source": true,      # return the full source
  "timeout": "60s"      # inactivity timeout
}
```

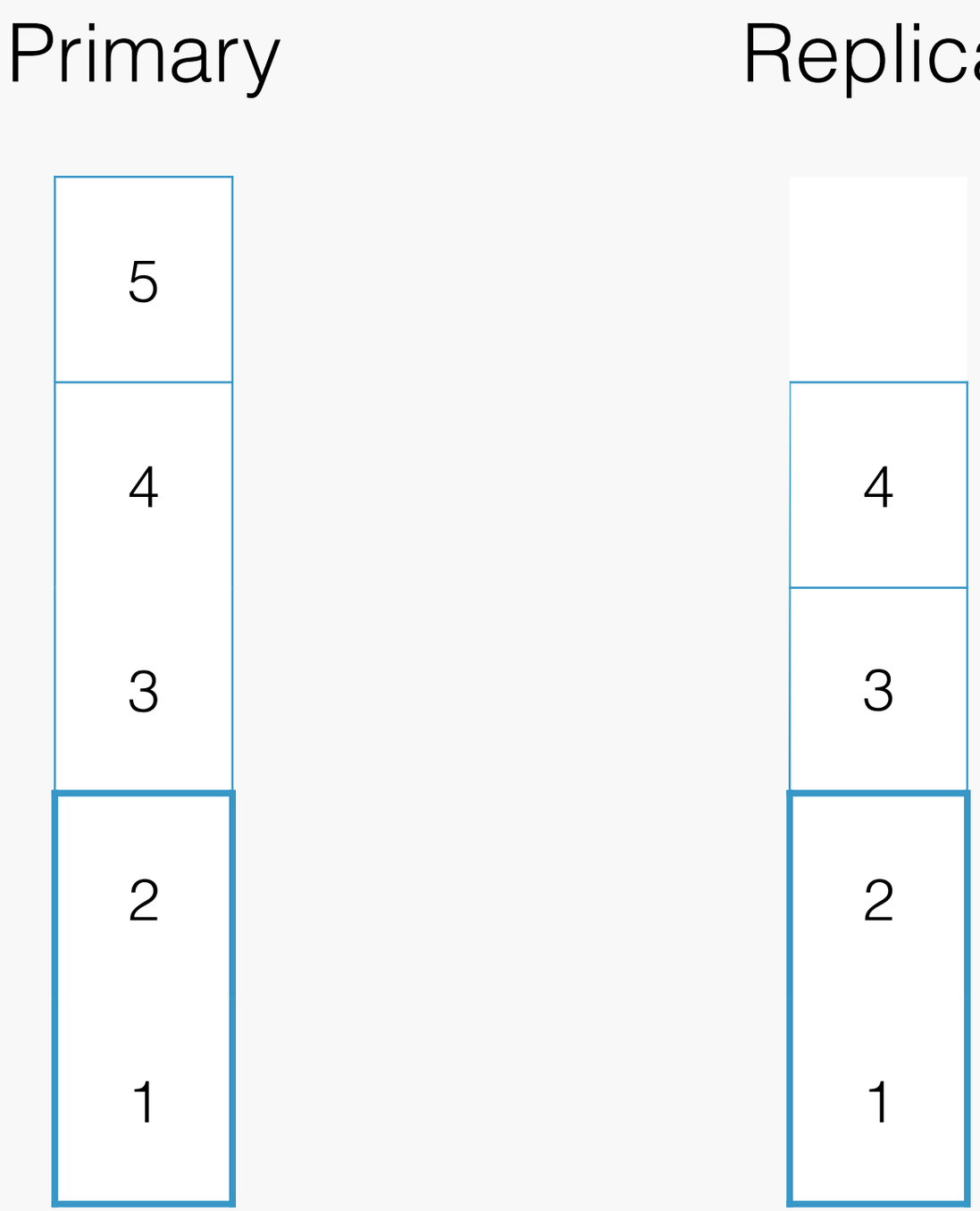
Primary Replica Sync



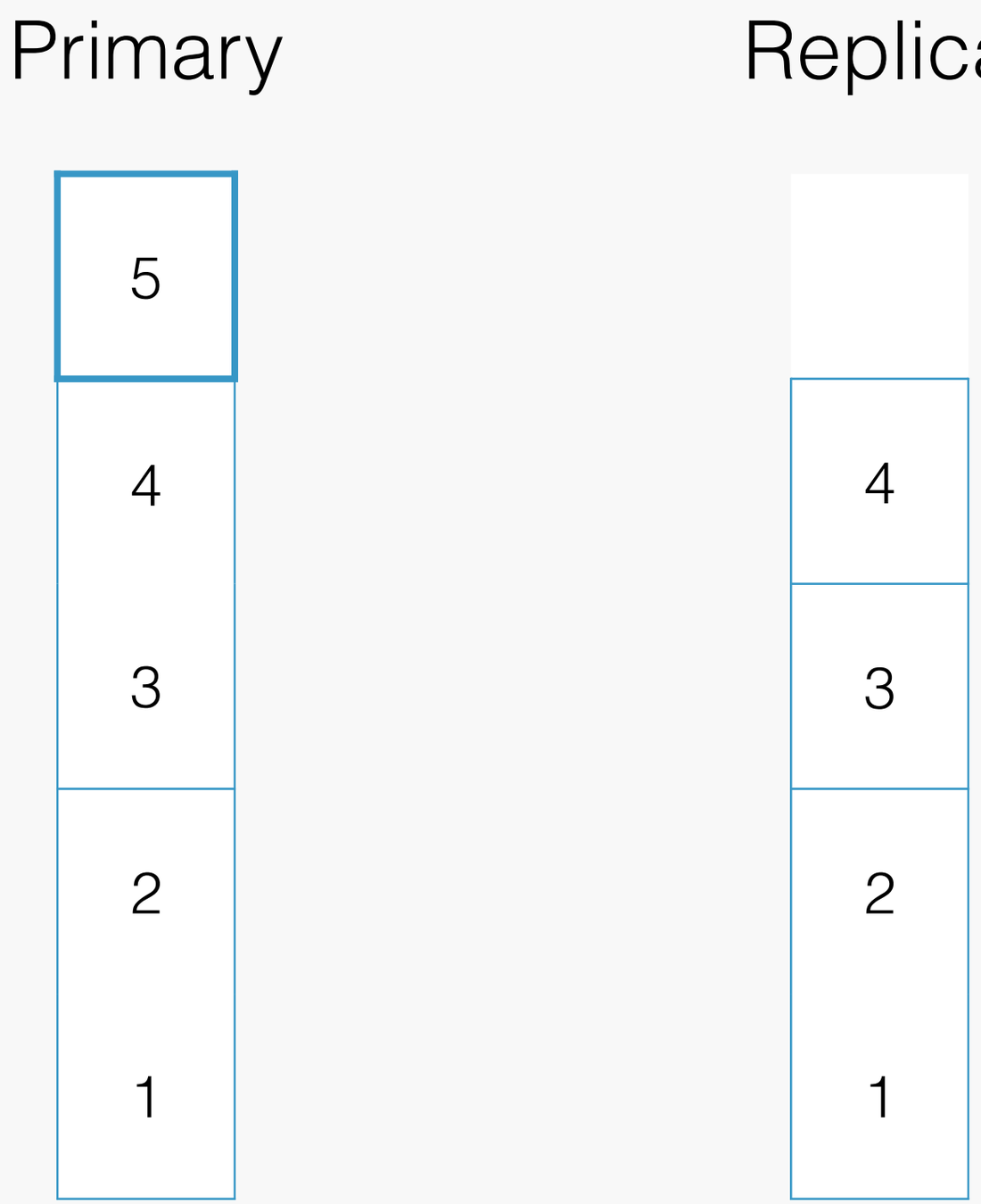
Primary Replica File Based Sync



Primary Replica File Based Sync

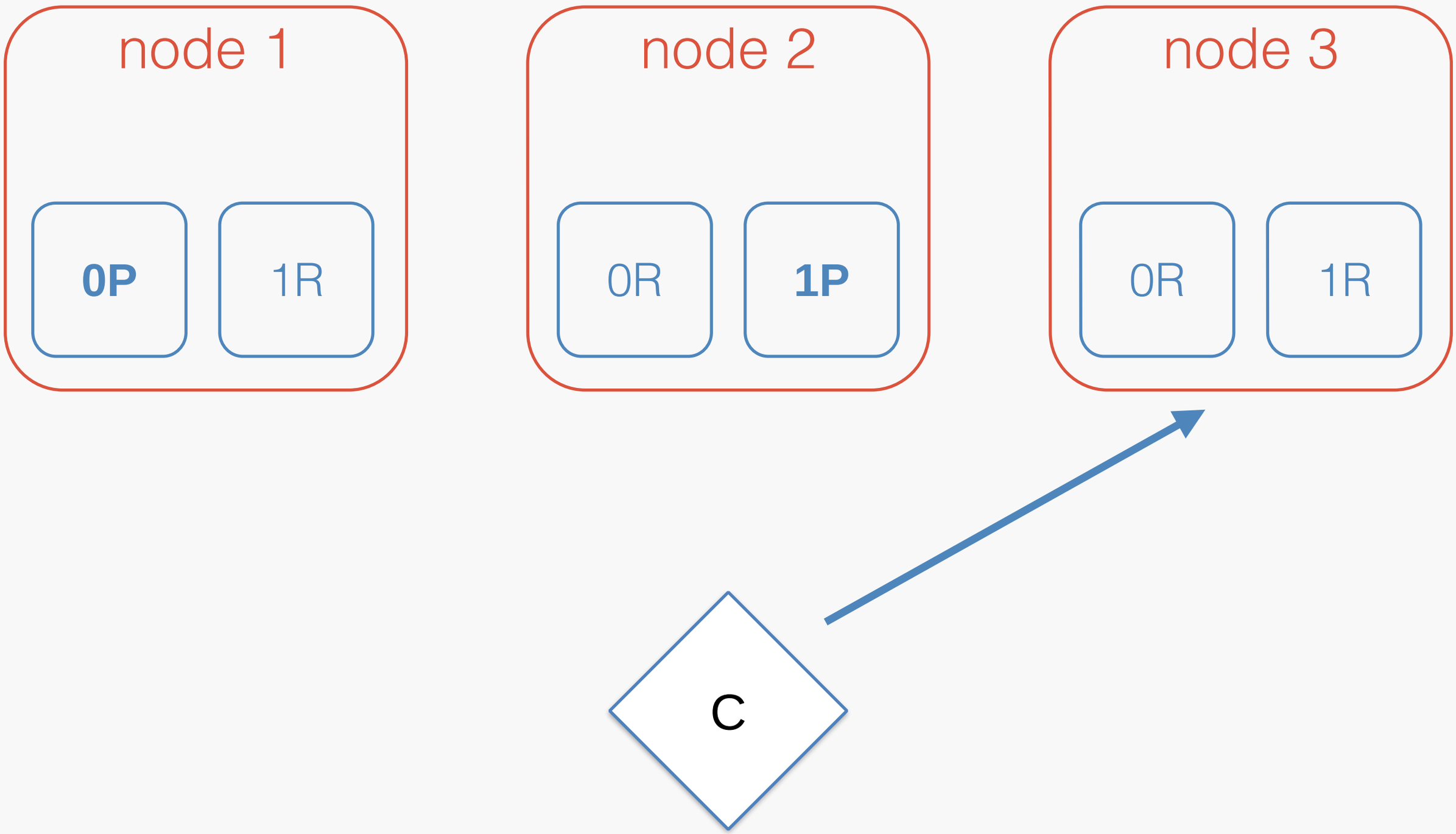


Primary Replica Seq# Based Sync

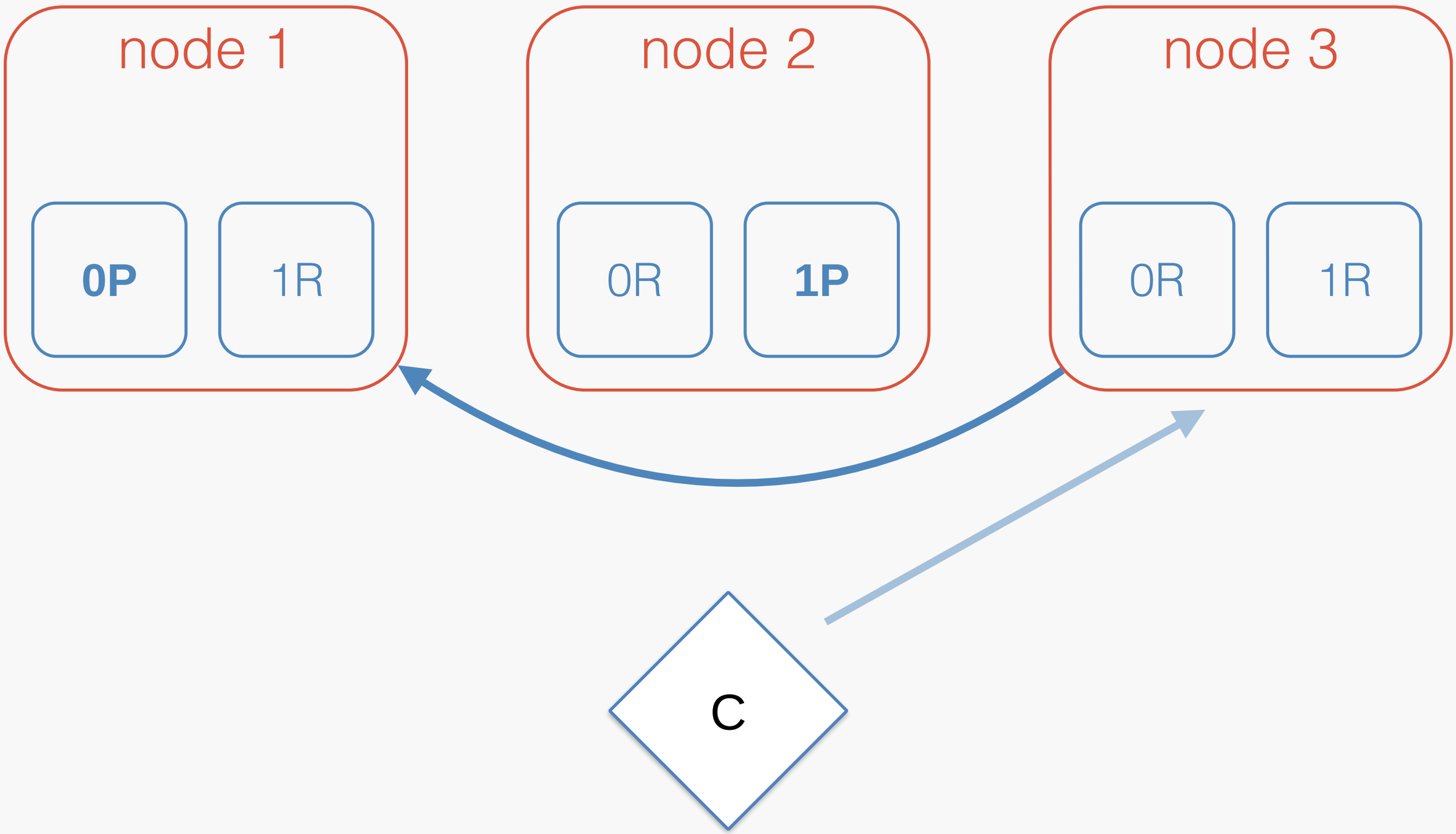


Indexing essentials

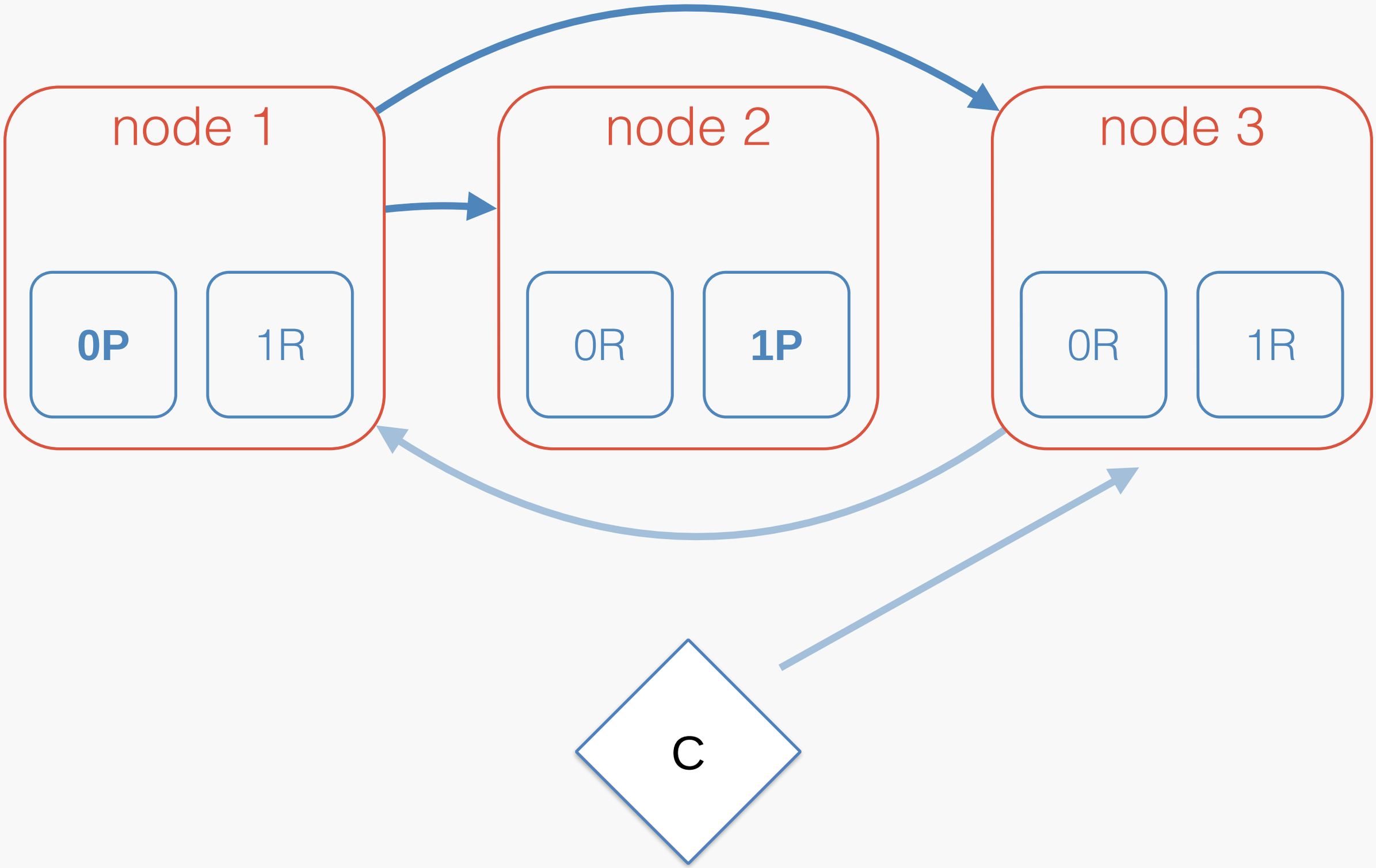
Indexing essentials



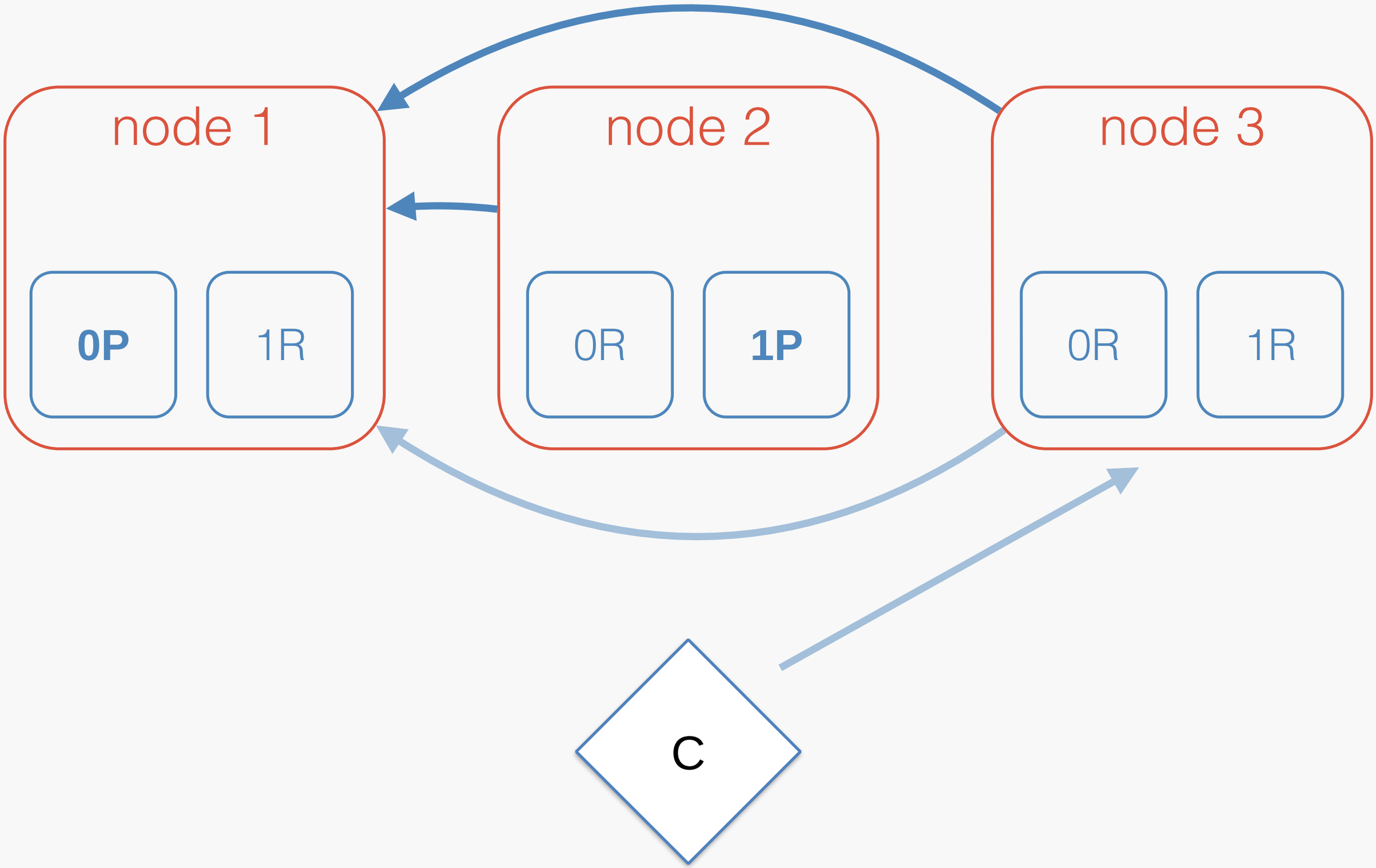
Indexing essentials



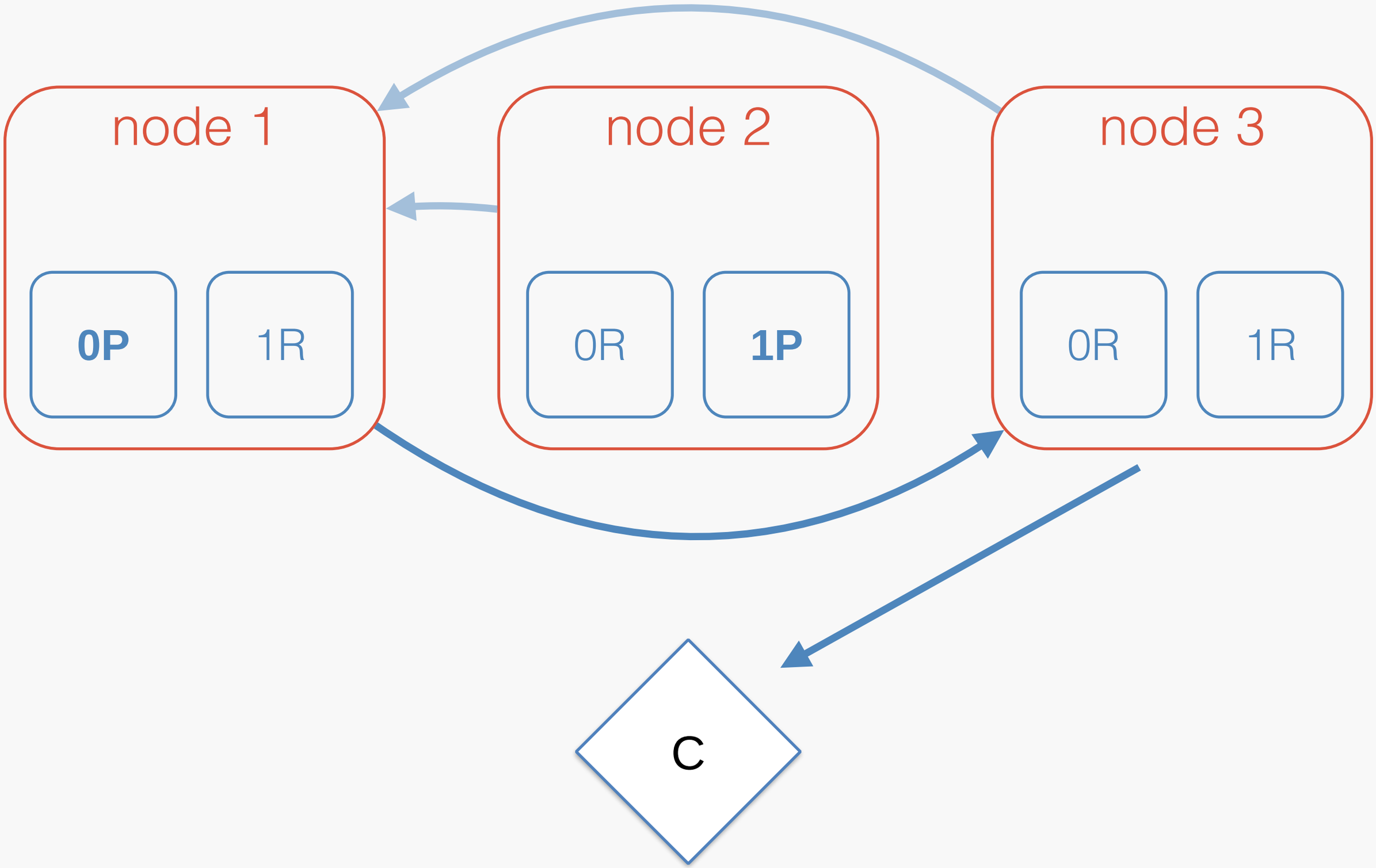
Indexing essentials



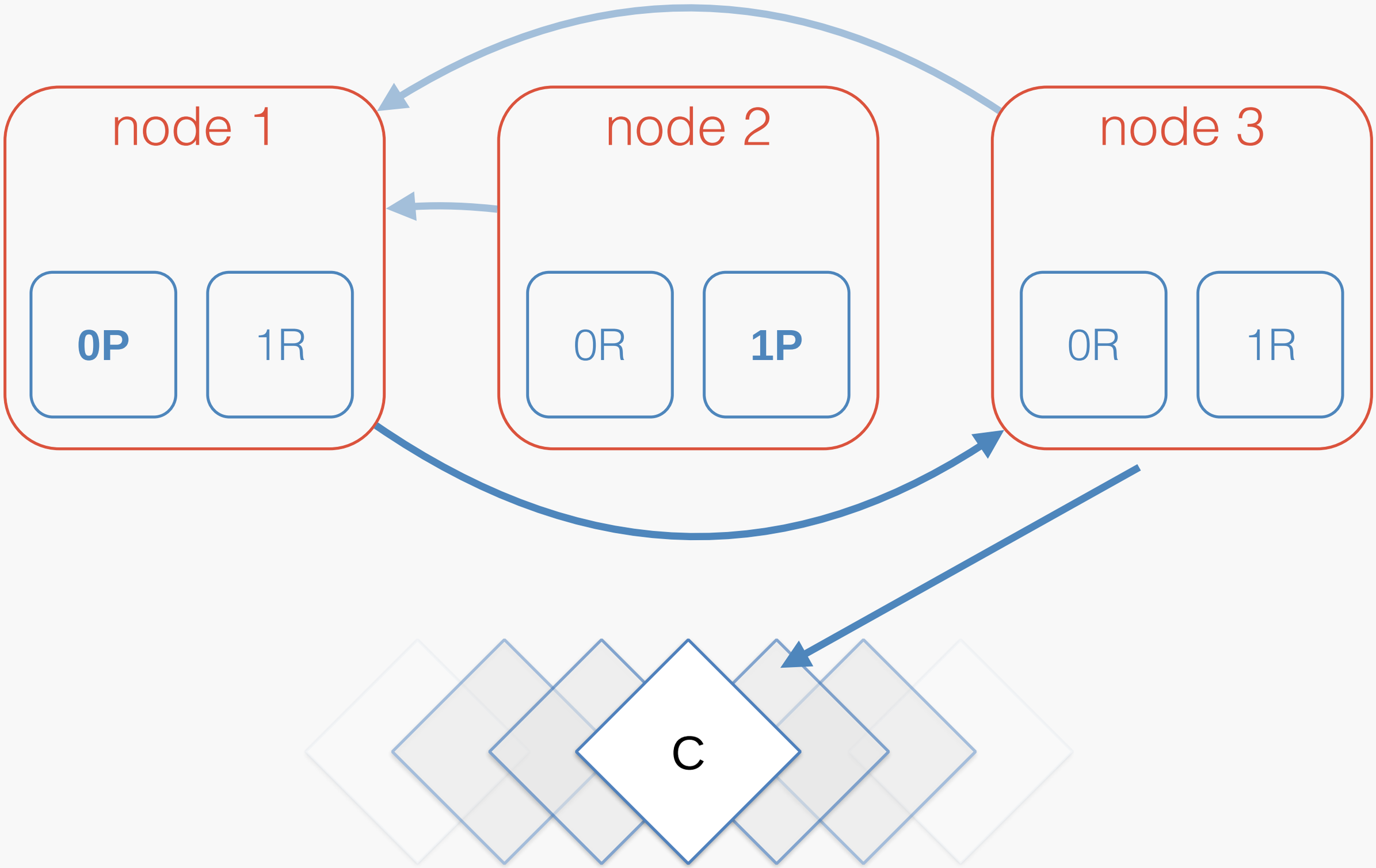
Indexing essentials



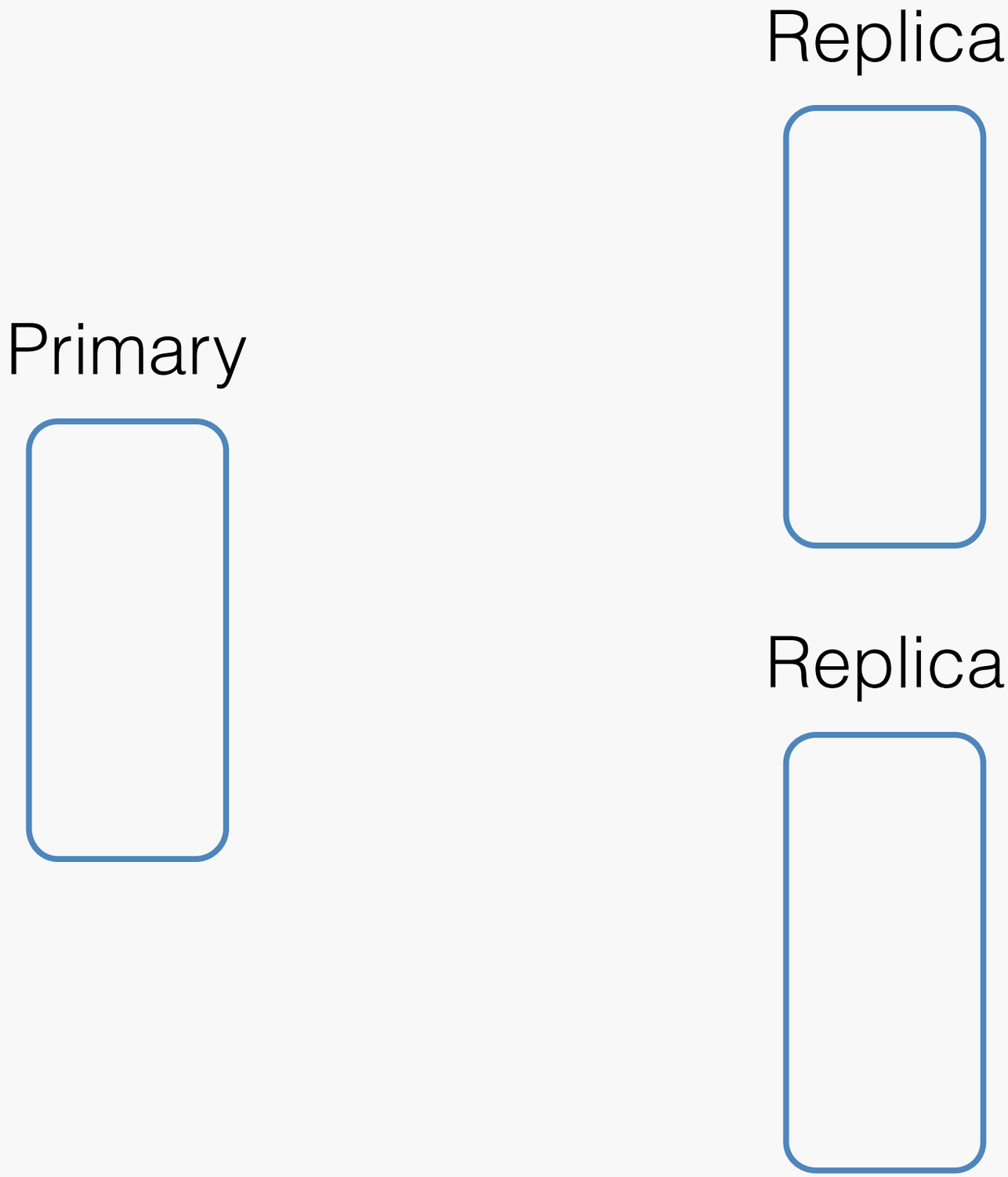
Indexing essentials



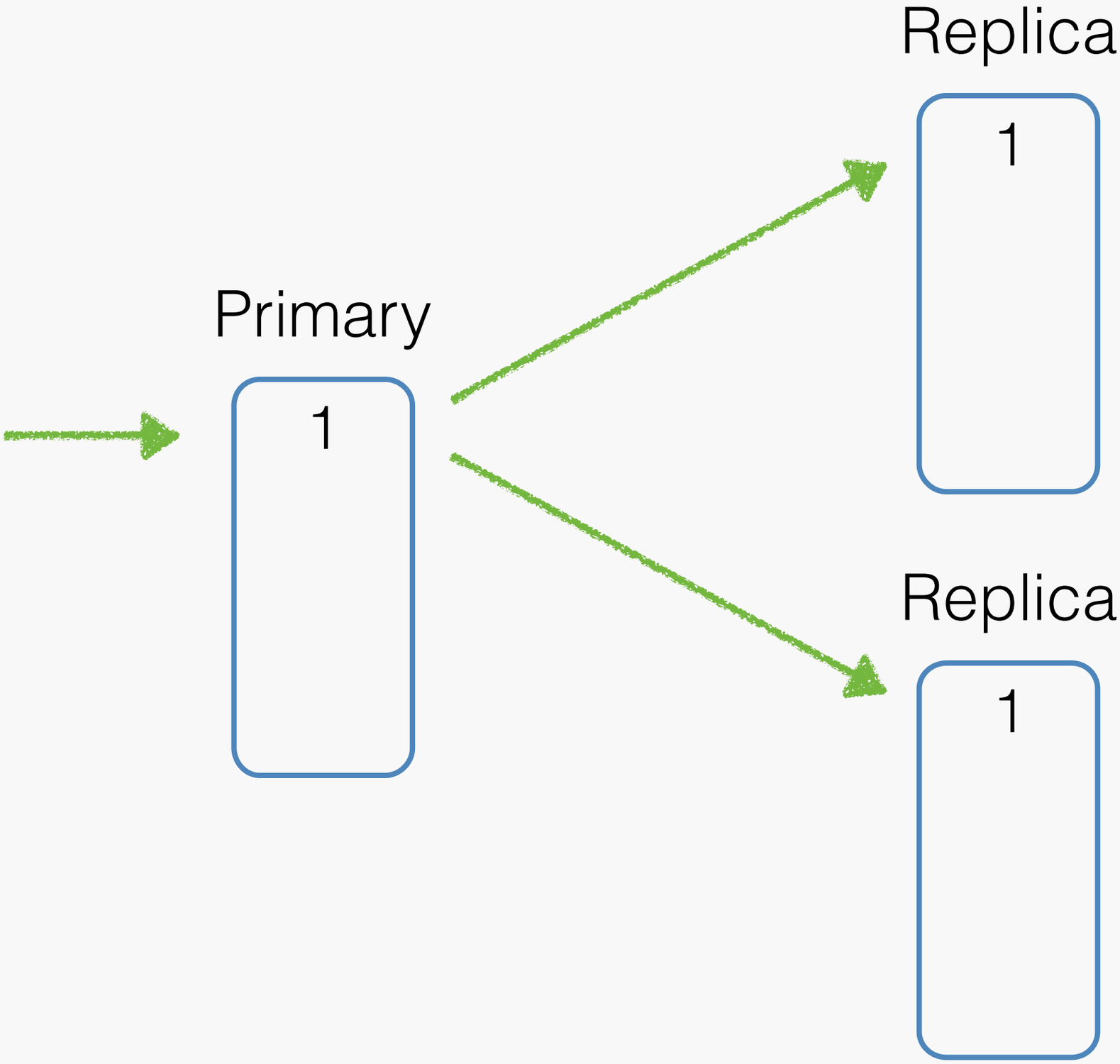
Indexing essentials



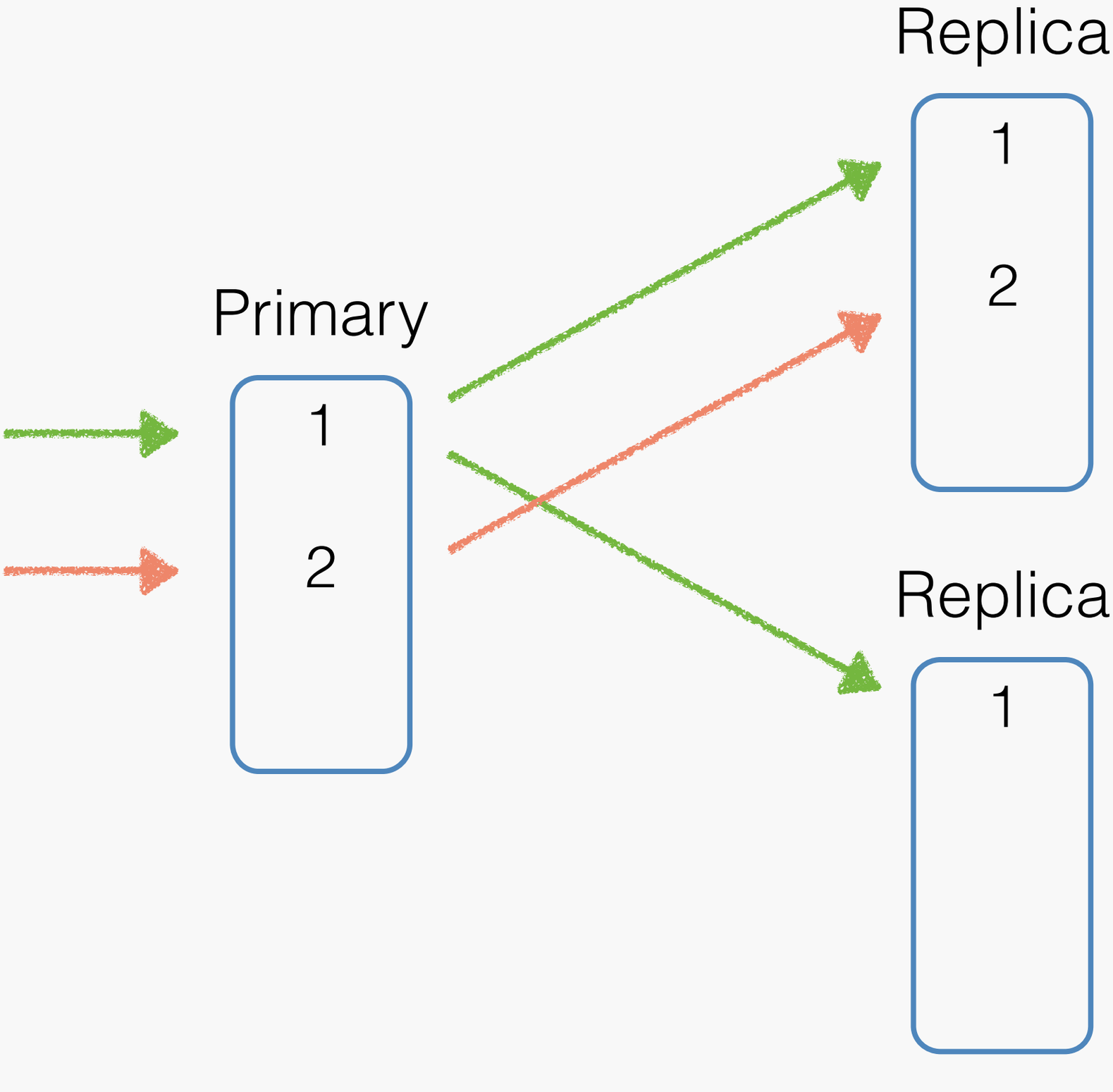
Concurrent Indexing



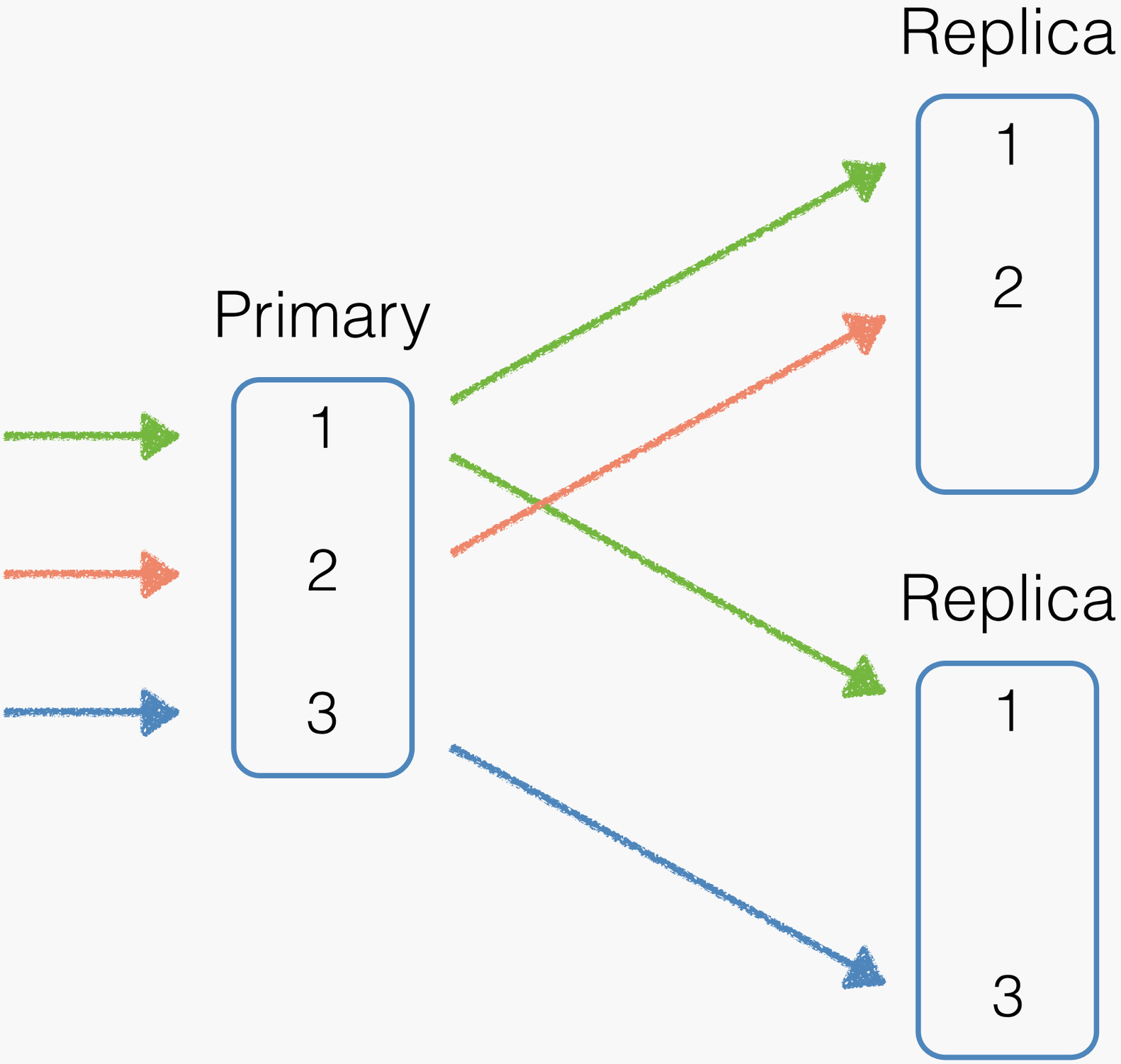
Concurrent Indexing



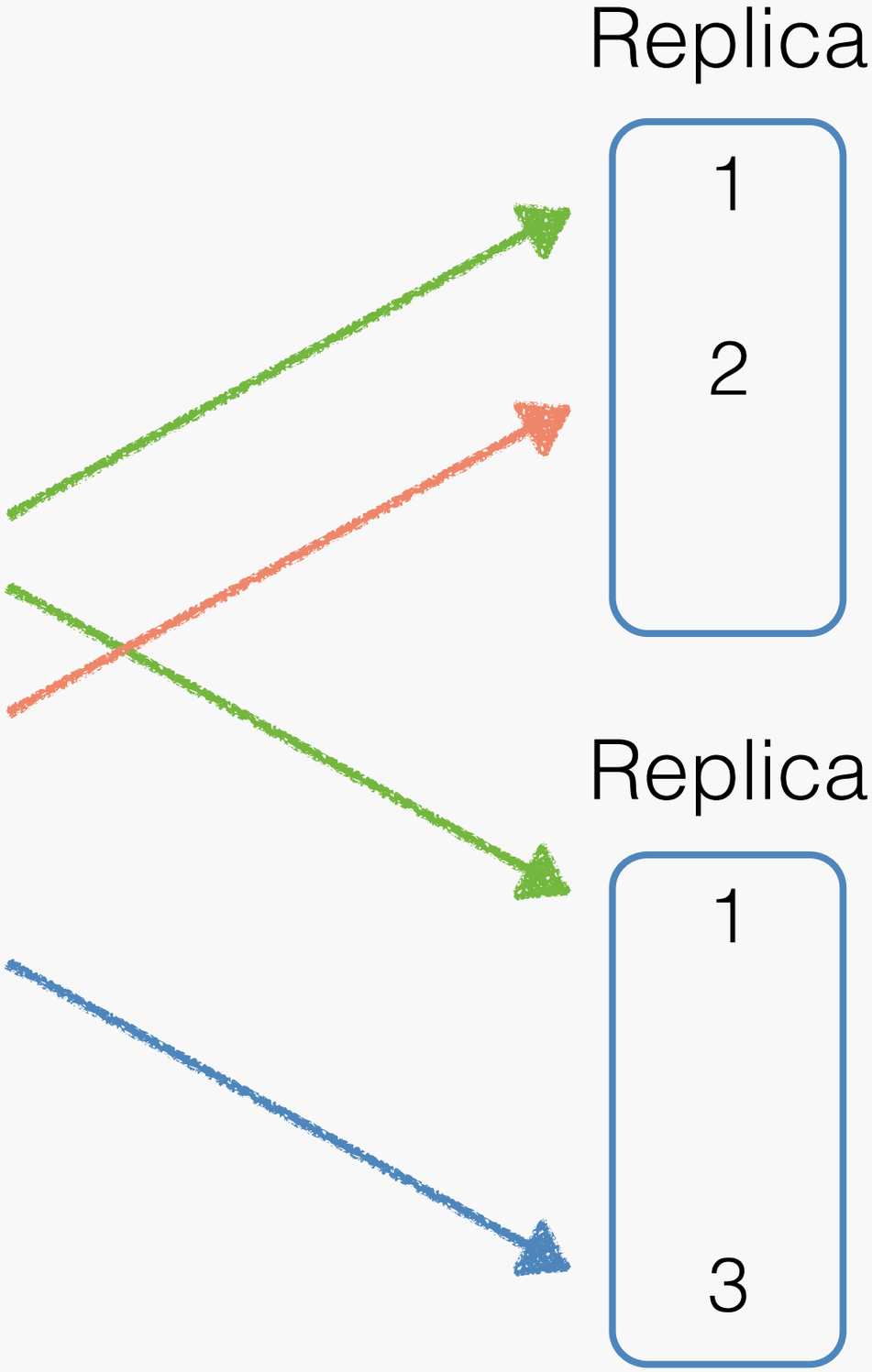
Concurrent Indexing



Concurrent Indexing



Concurrent Indexing



Requirements

- Correct :)
- Fault tolerant
- Support concurrency

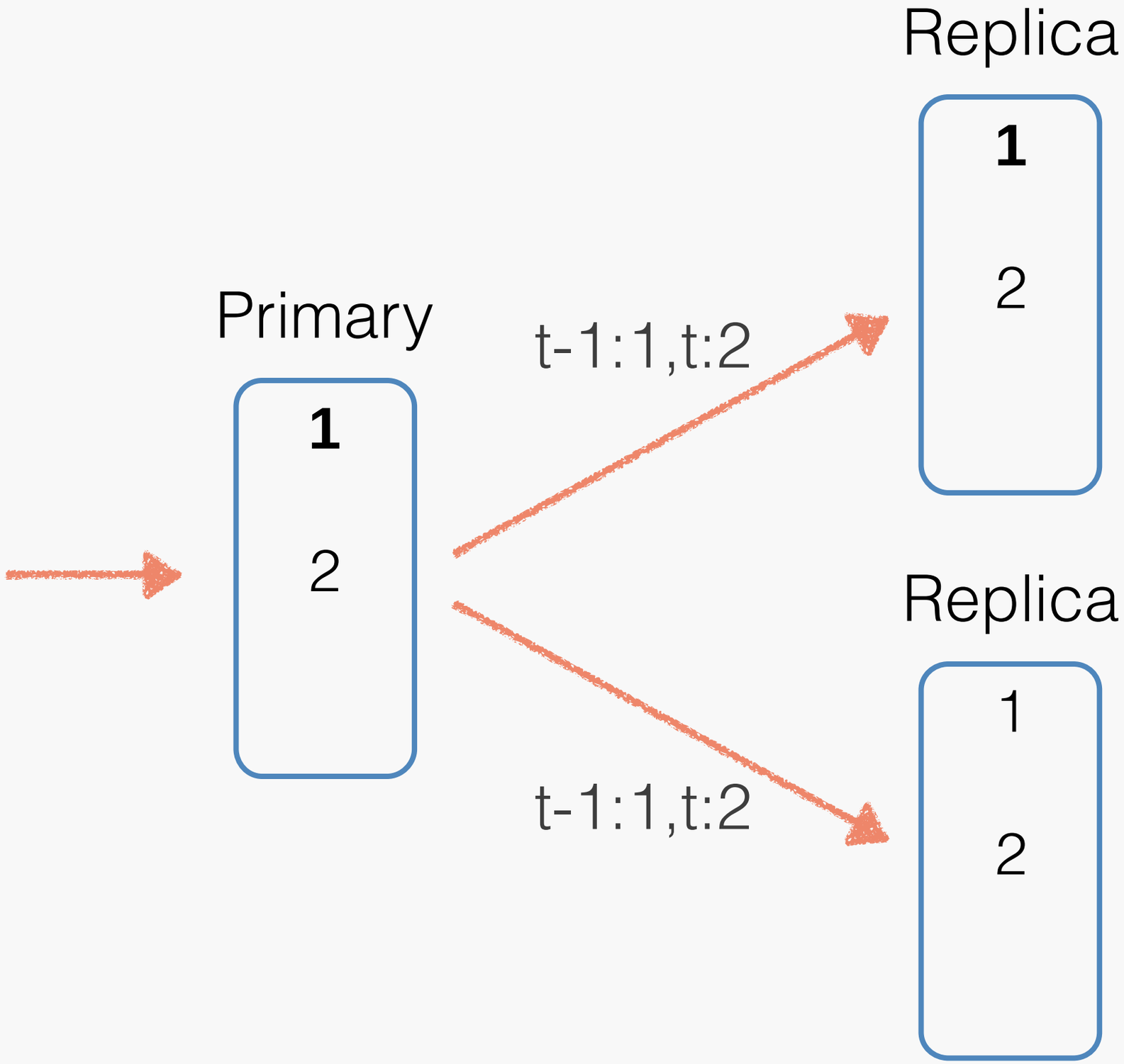
Consistency Algorithm

For example, Raft

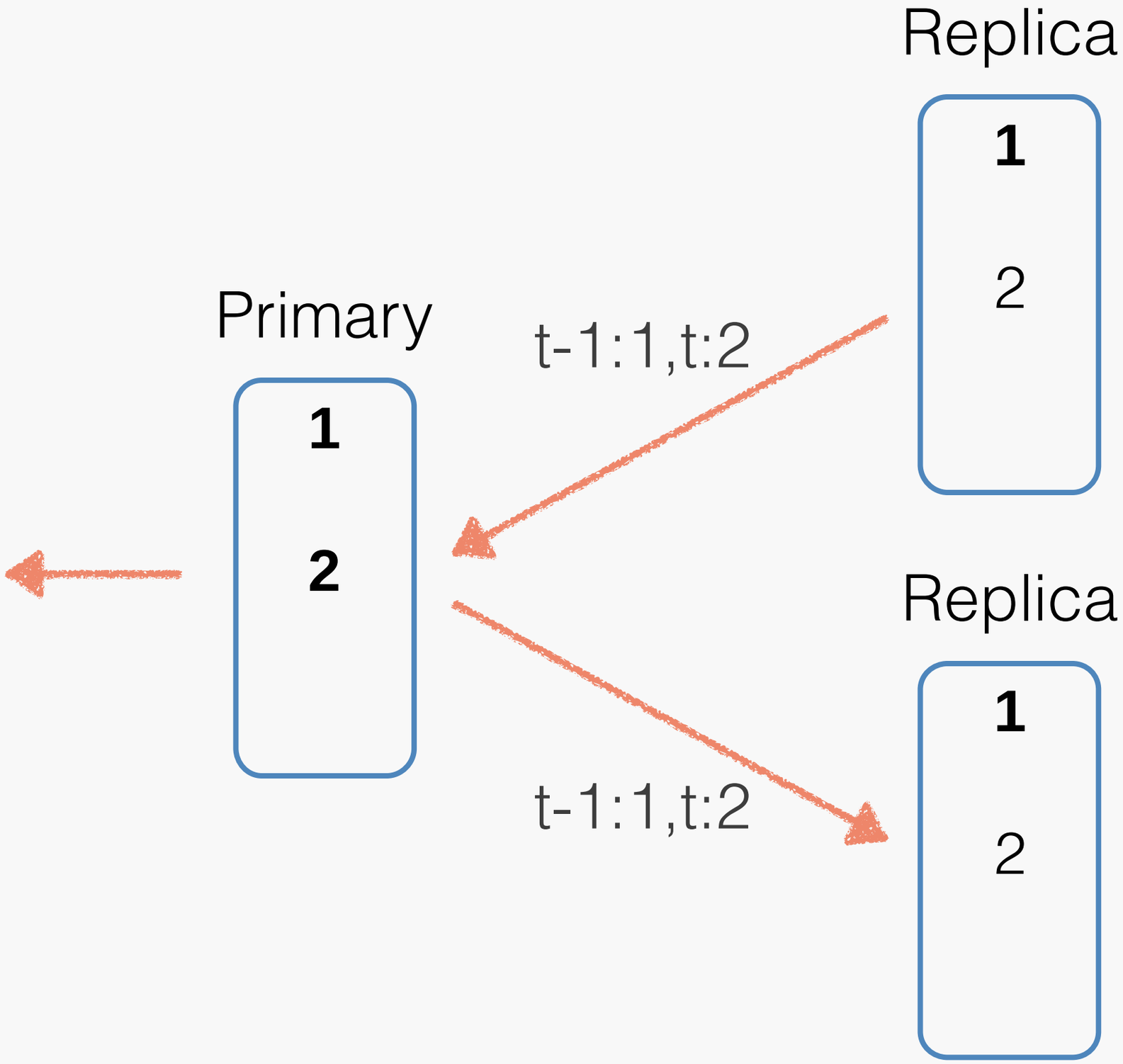
Raft Consensus Algorithm

- Built to be understandable
- Leader based
- Modular (election + replication)
- See https://raftconsensus.github.io/
- Used by Facebook's HBase port & Algolia for data replication

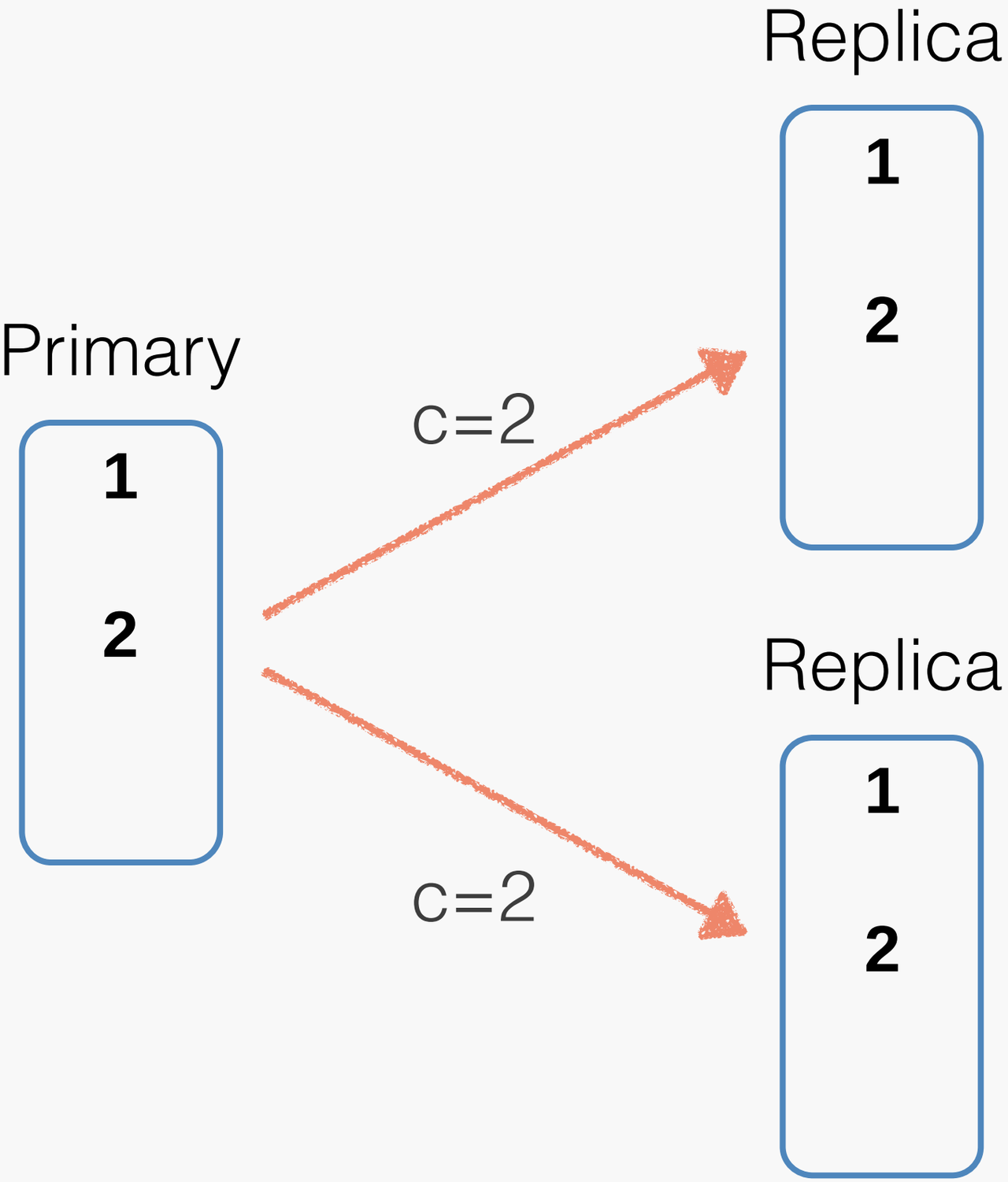
Raft - appendEntries



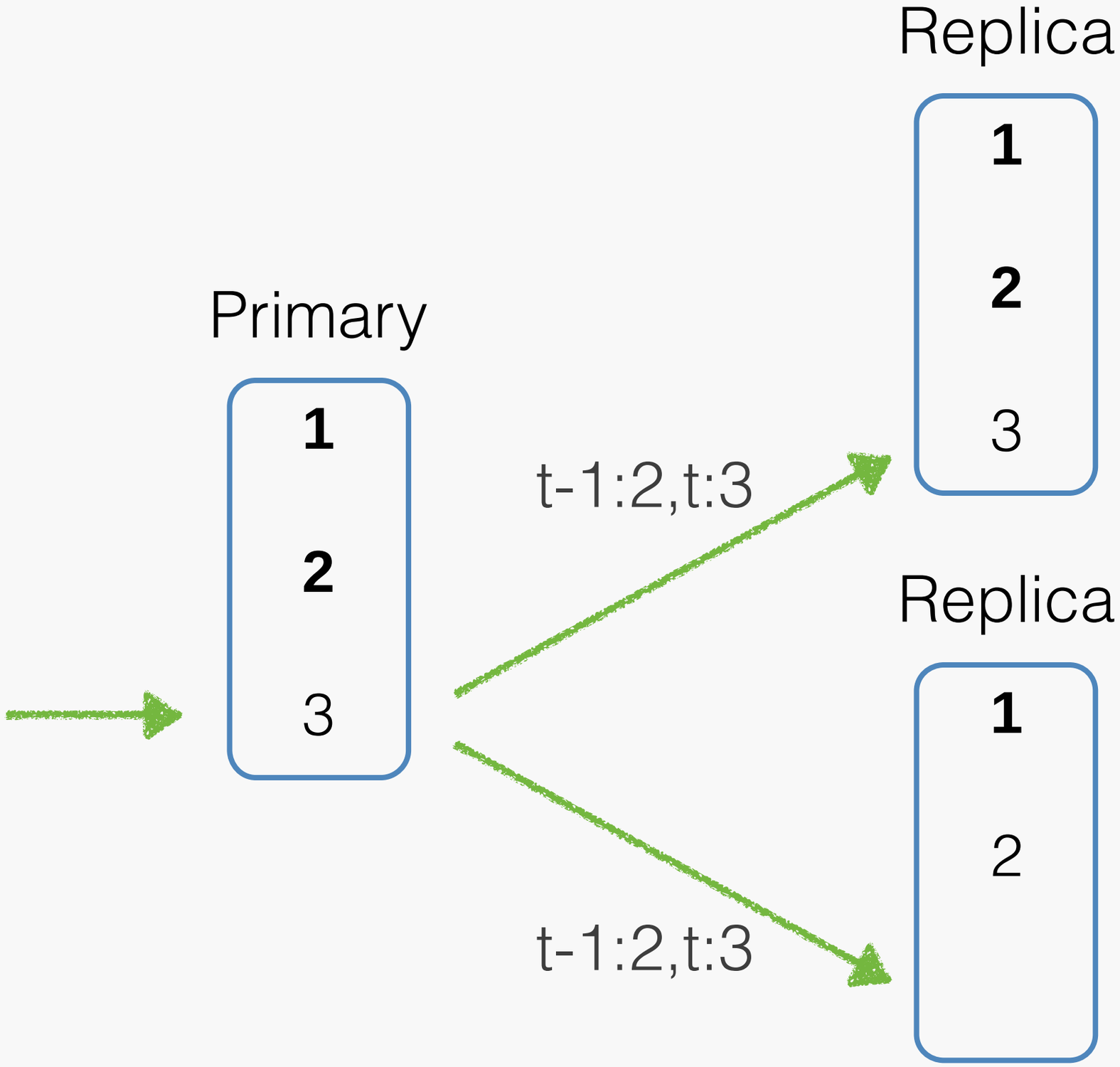
Raft - commit on quorum



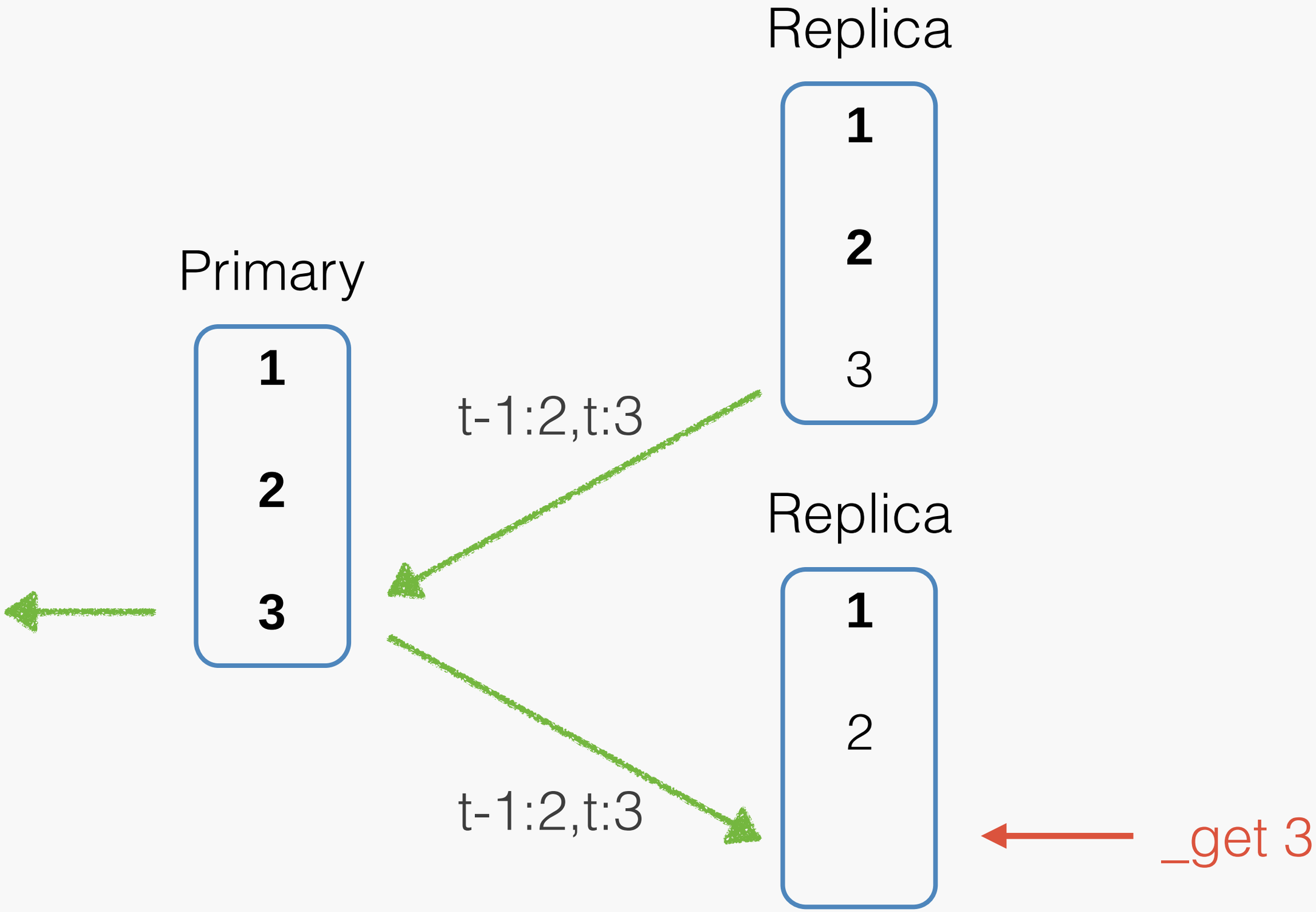
Raft - broadcast* commit



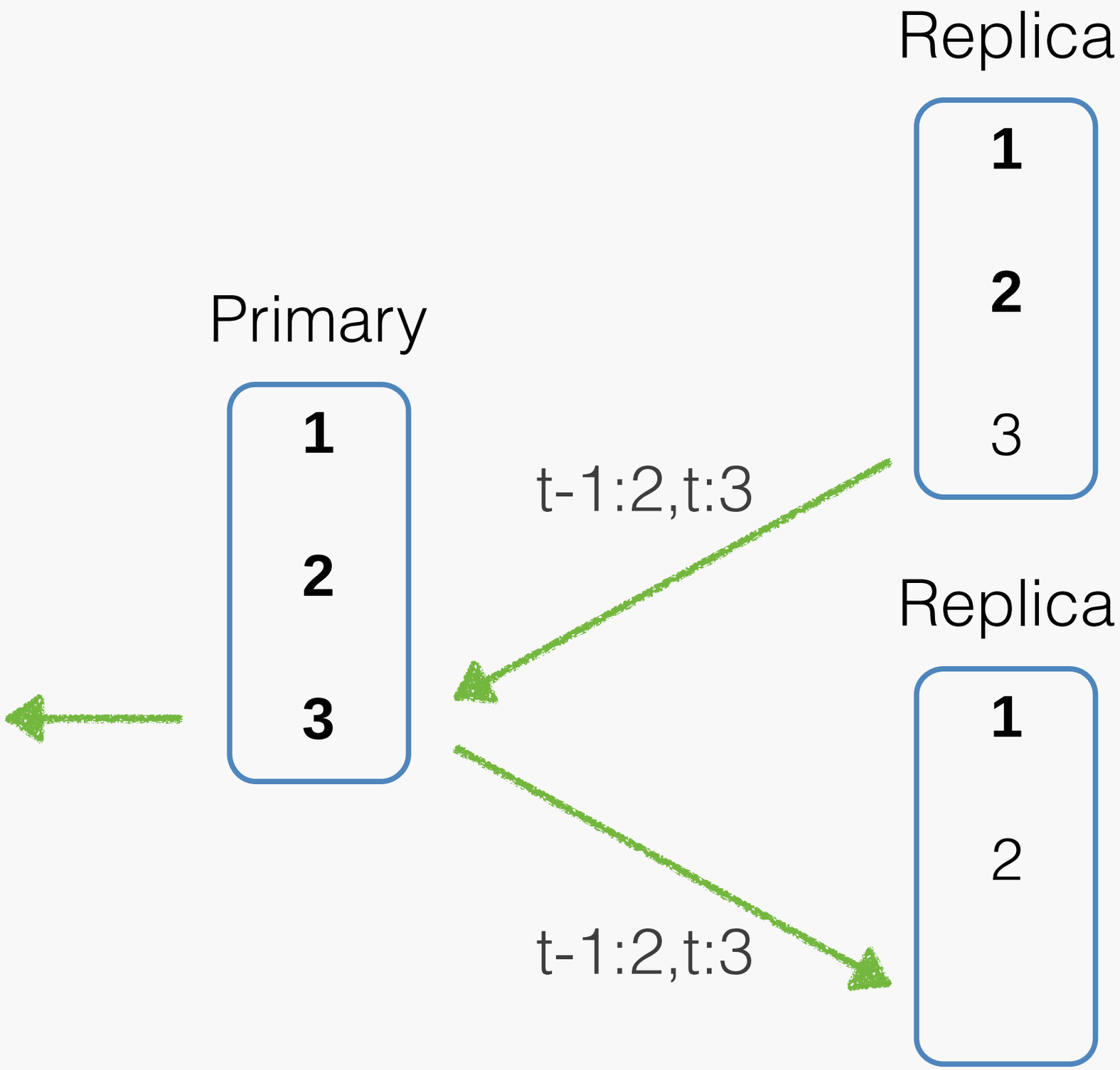
Raft - primary failure



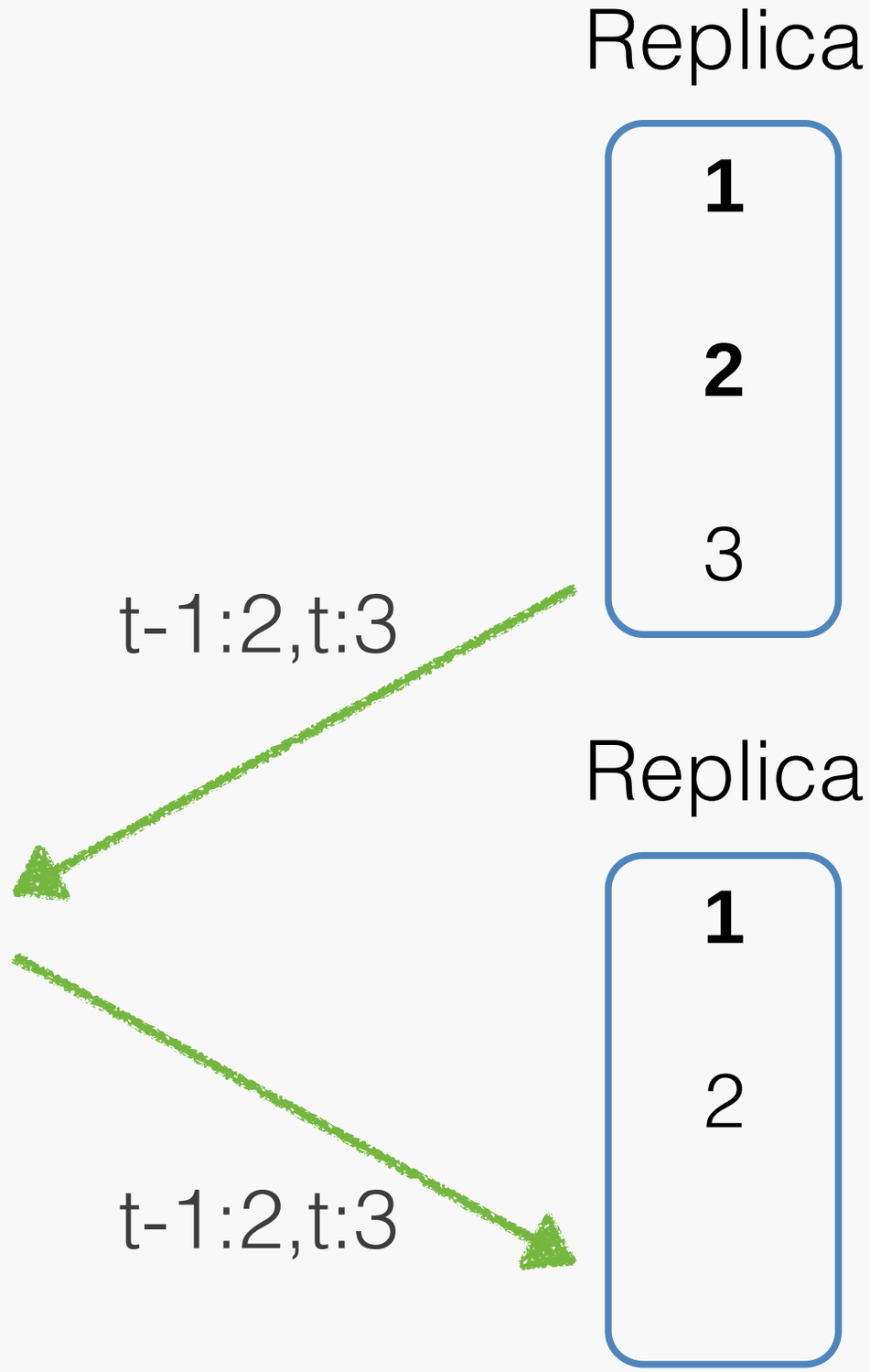
Raft - ack on quorum



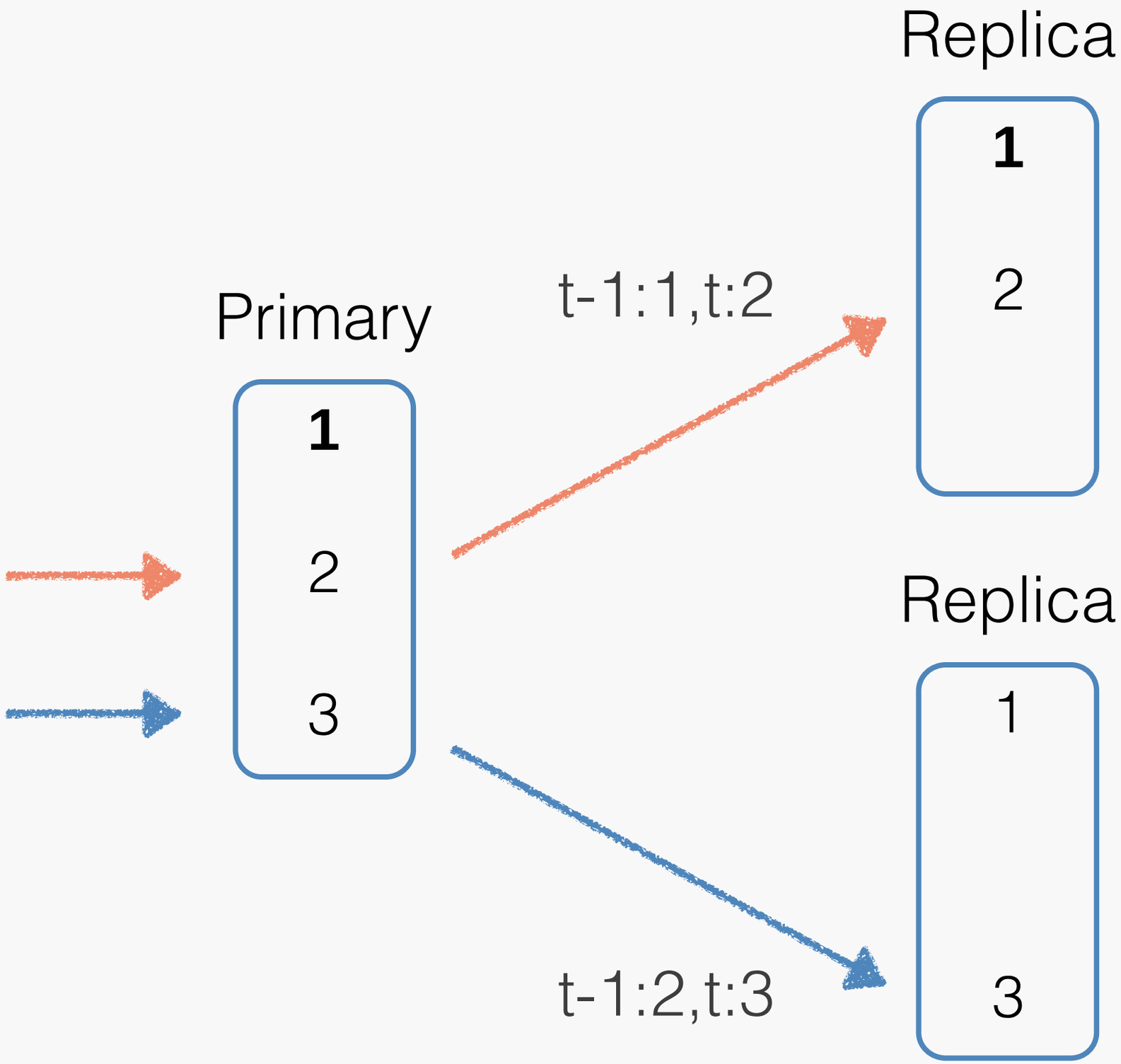
Raft - primary failure



Raft - primary failure



Raft - concurrent indexing?



Raft

- Simple to understand
- Quorum means:
  - Lagging shards don’t slow down indexing

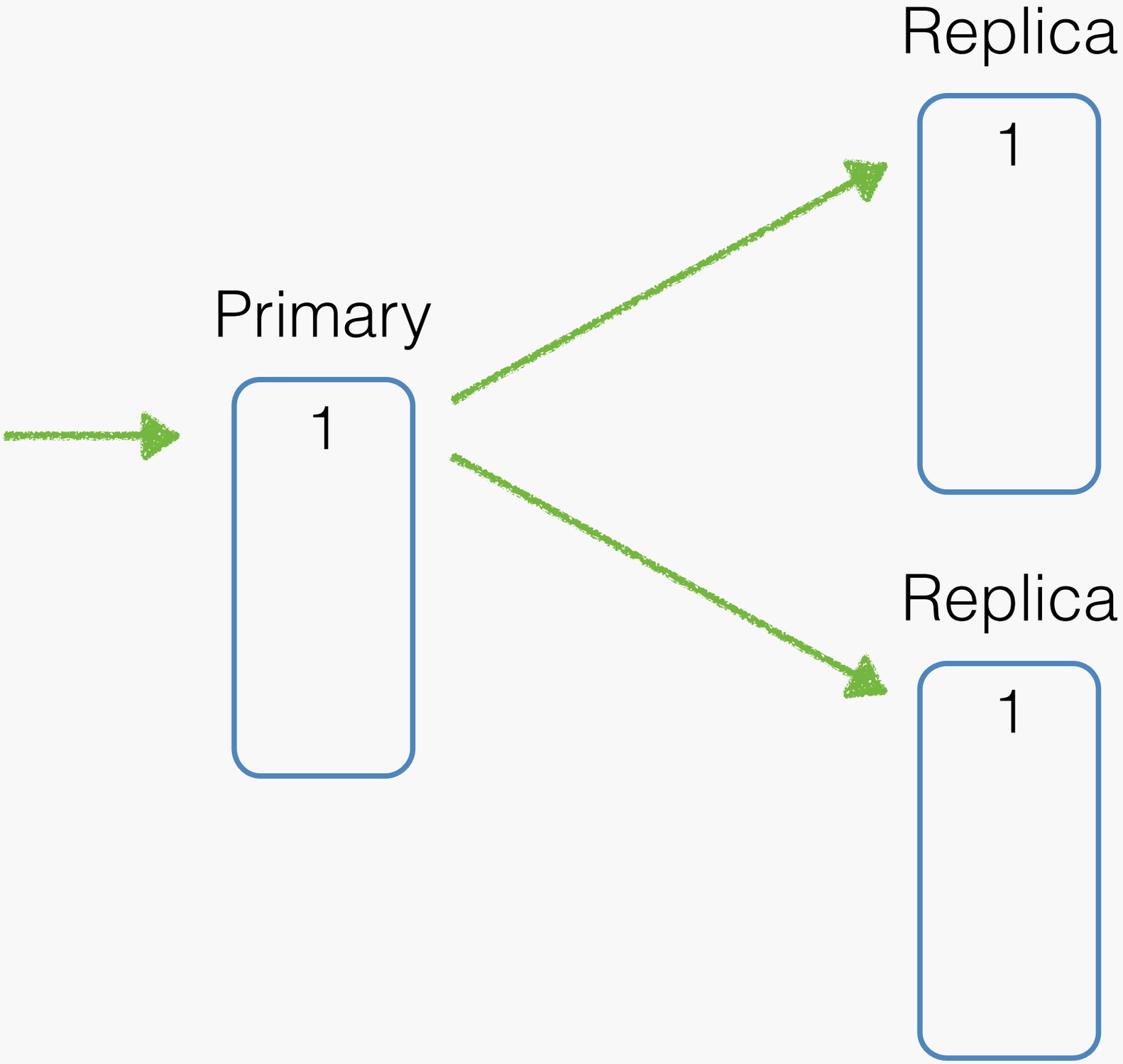
    but
  - Read visibility issues
  - Tolerates up to quorum - 1 failures
  - Needs at least 3 copies for correctness
- Challenges with concurrency

Master-Backup replication

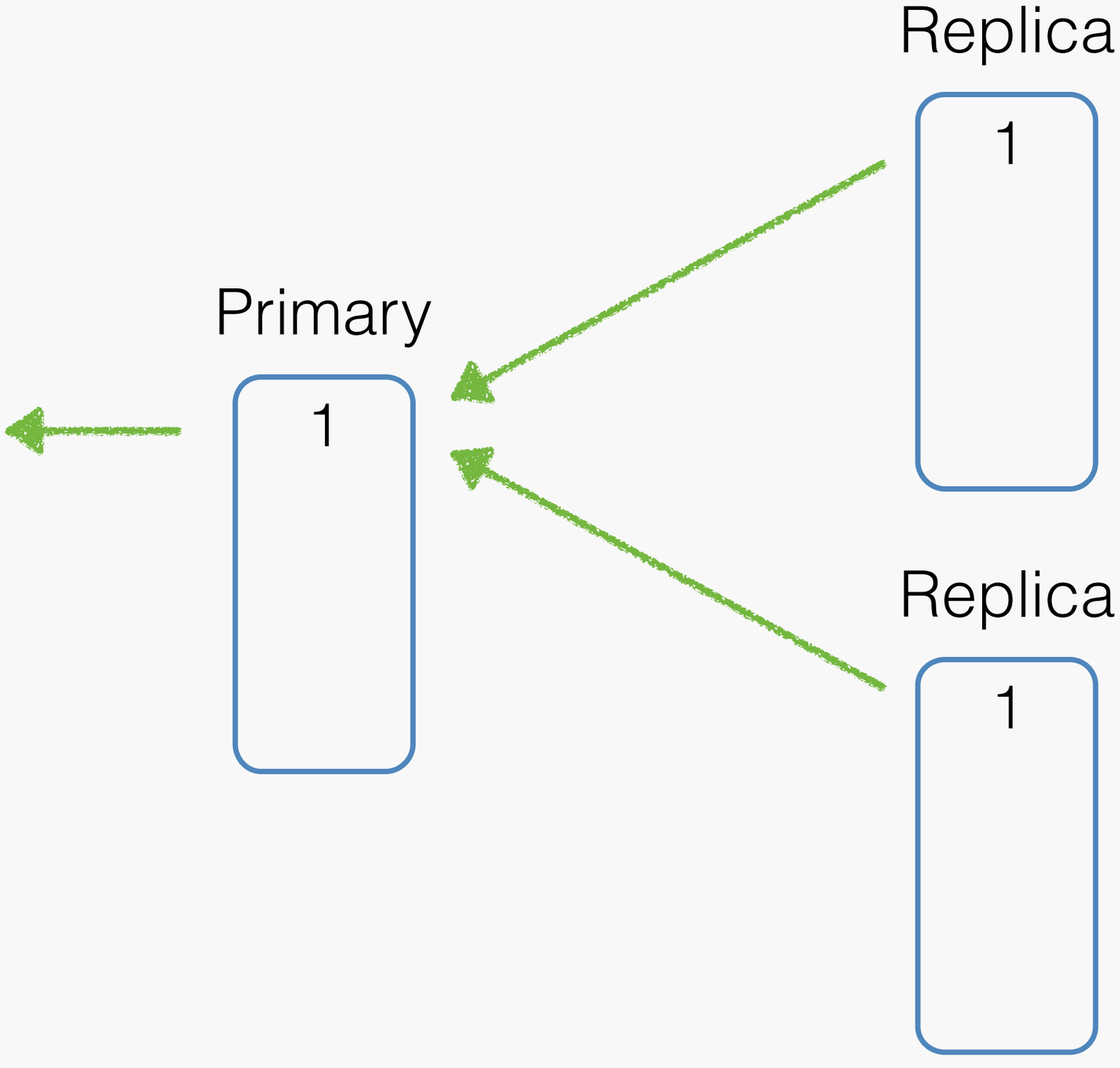
Master Backup Replication

- Leader based
- Writes to all copies before ack-ing.
- Used by Elasticsearch, Kafka, RAMCloud (and many others)

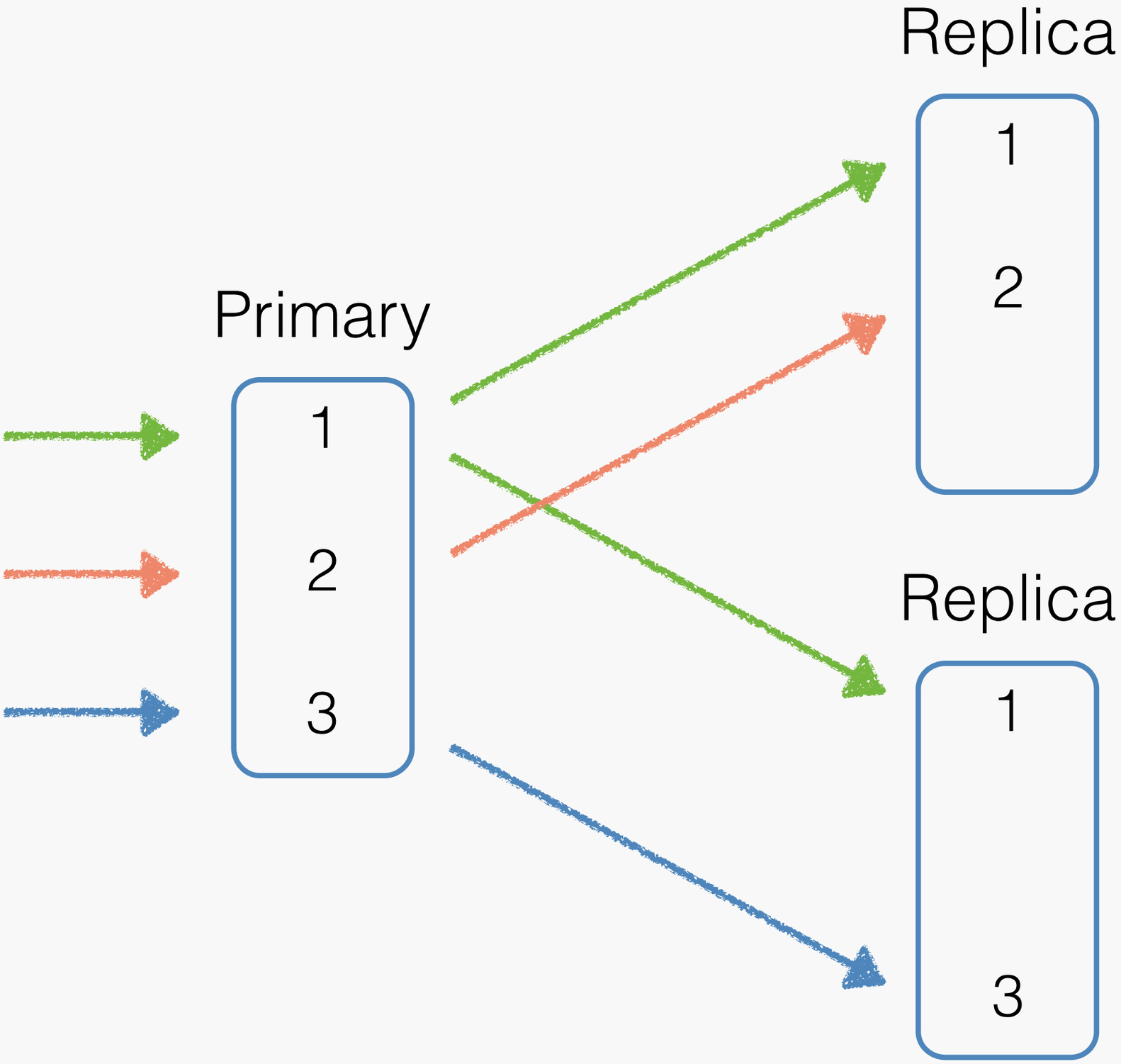
Master-Backup - indexing



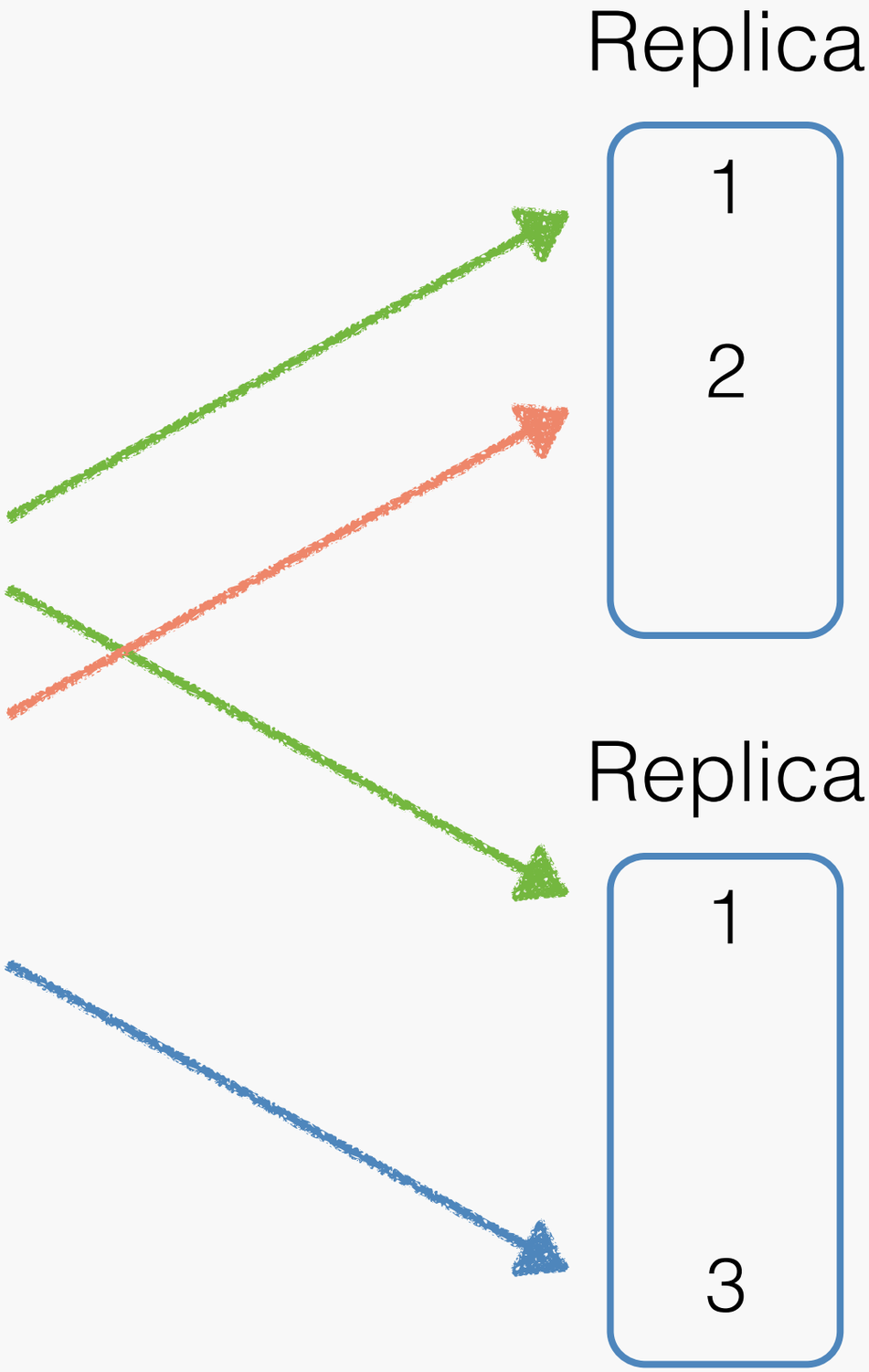
Master-Backup - indexing



Master-Backup - concurrency/failure



Master-Backup - concurrency/failure



Master-Backup replication

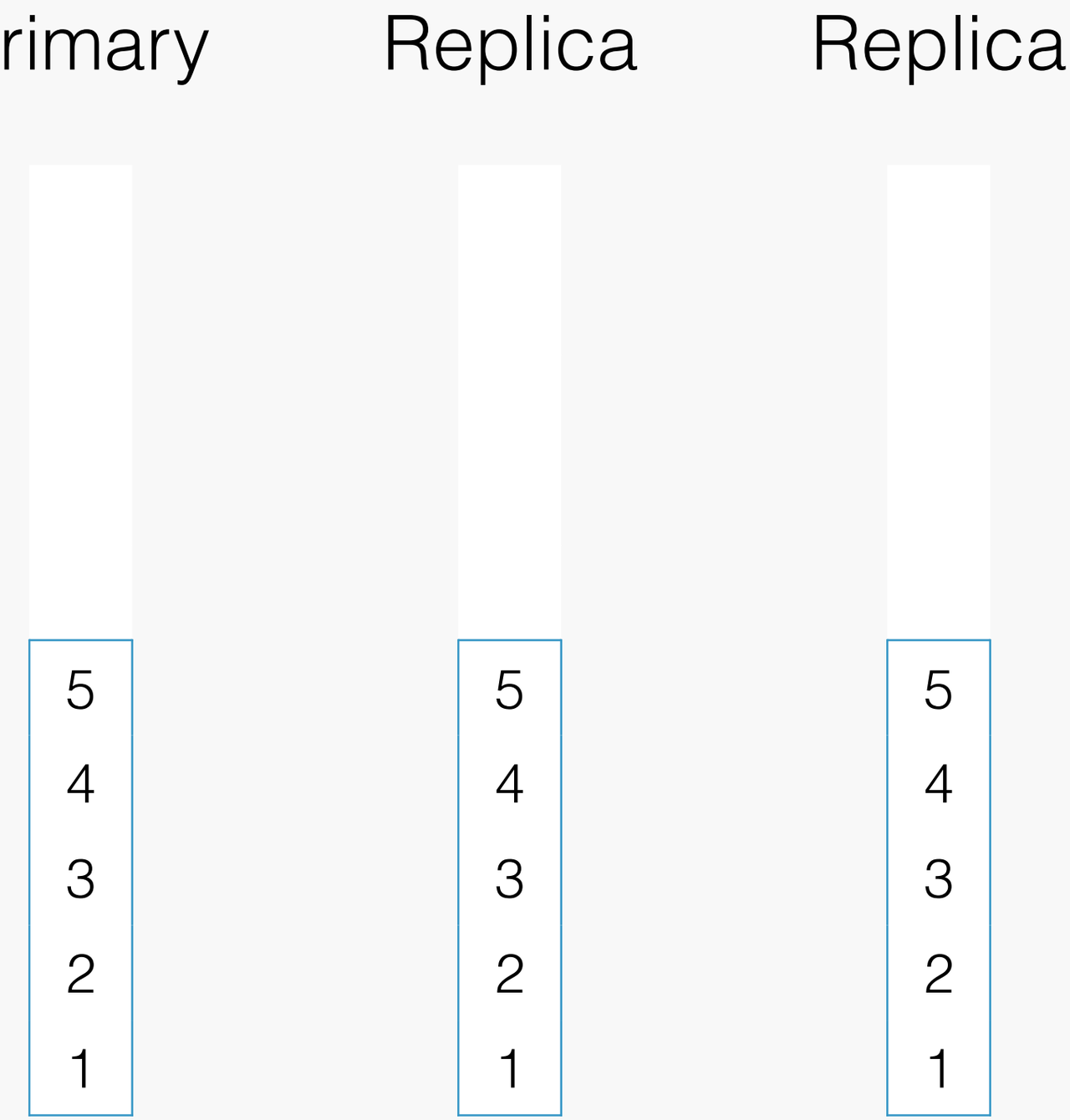
- Simple to understand
- Write to all before ack means:
  - No read visibility issues
  - Tolerates up to N-1 failures

  but

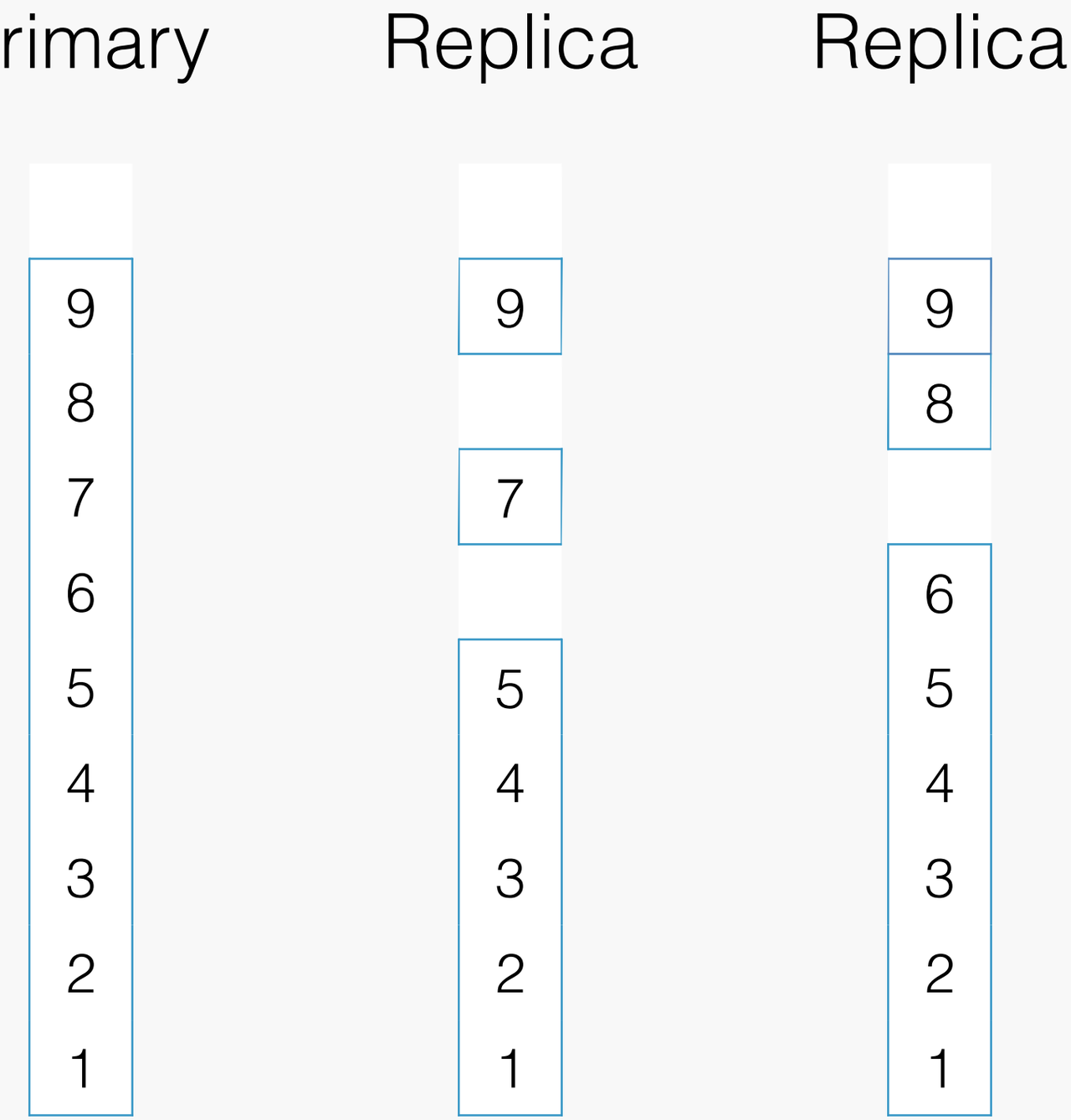
  - A lagging shard slows indexing down (until failed)
- Easier to work with concurrency
  - Rollbacks on failure are more frequent
  - No clear commit point

Failure, Rollback and Commitment

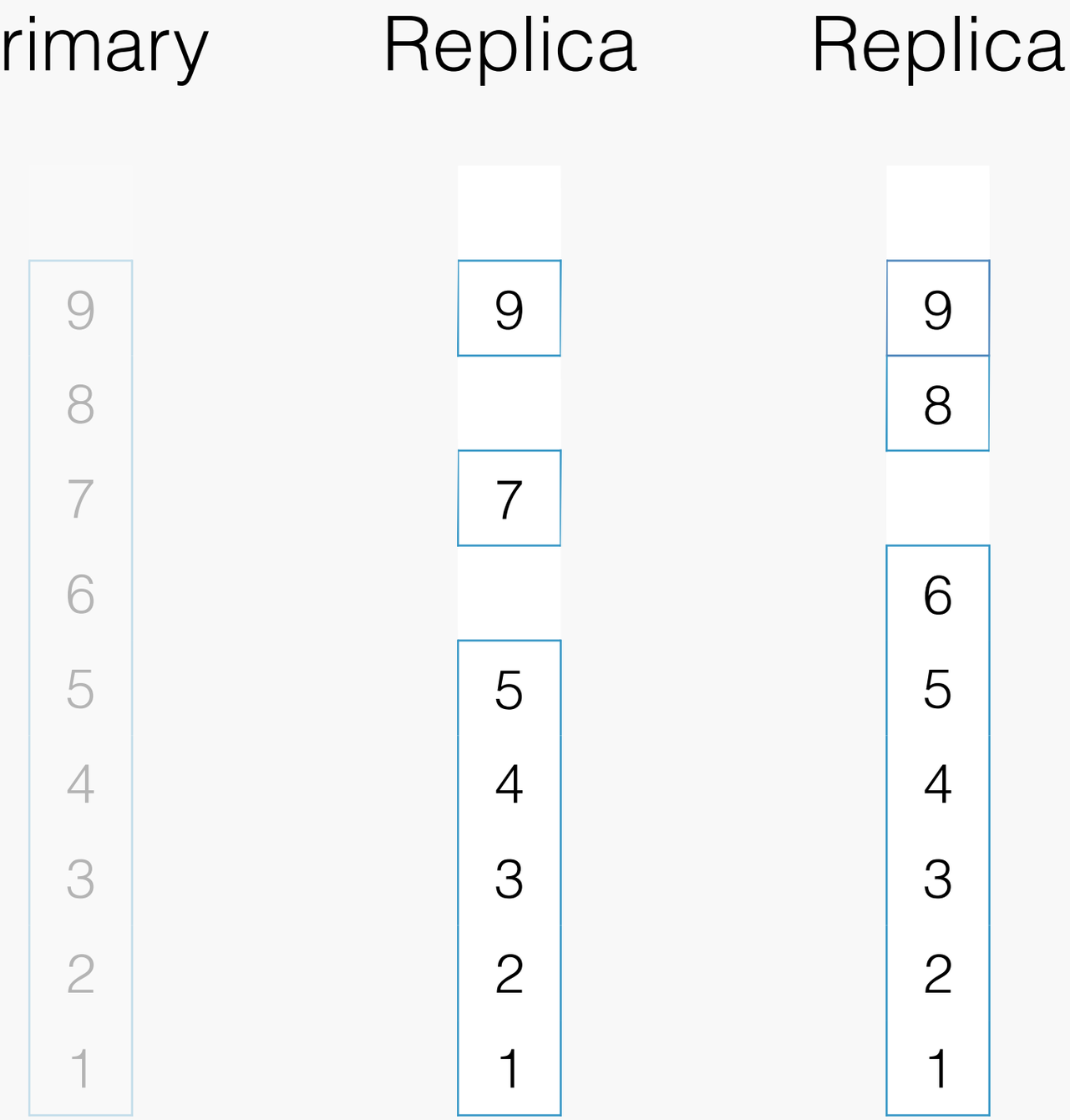
3 histories



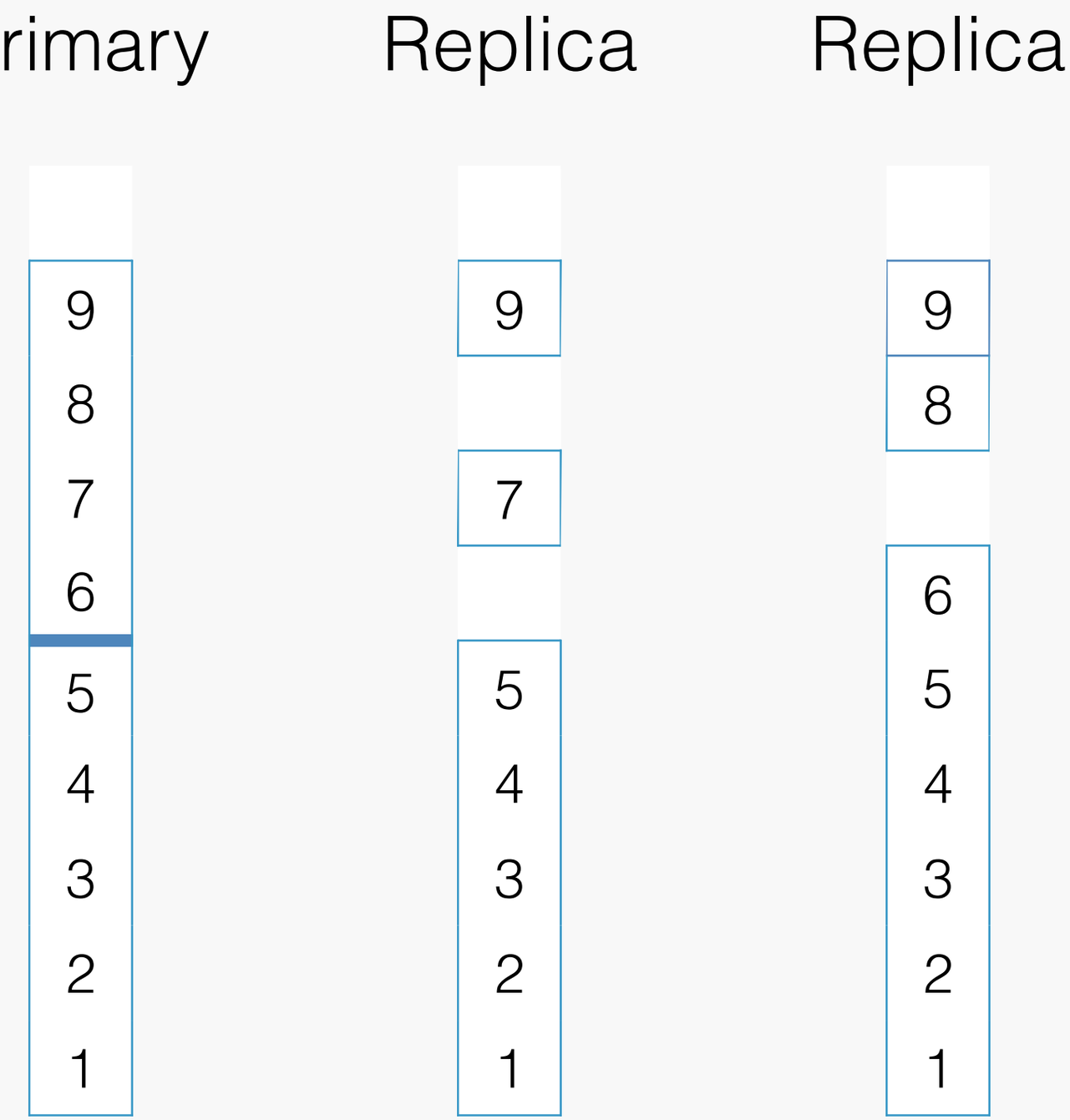
Failure, Rollback and Commitment



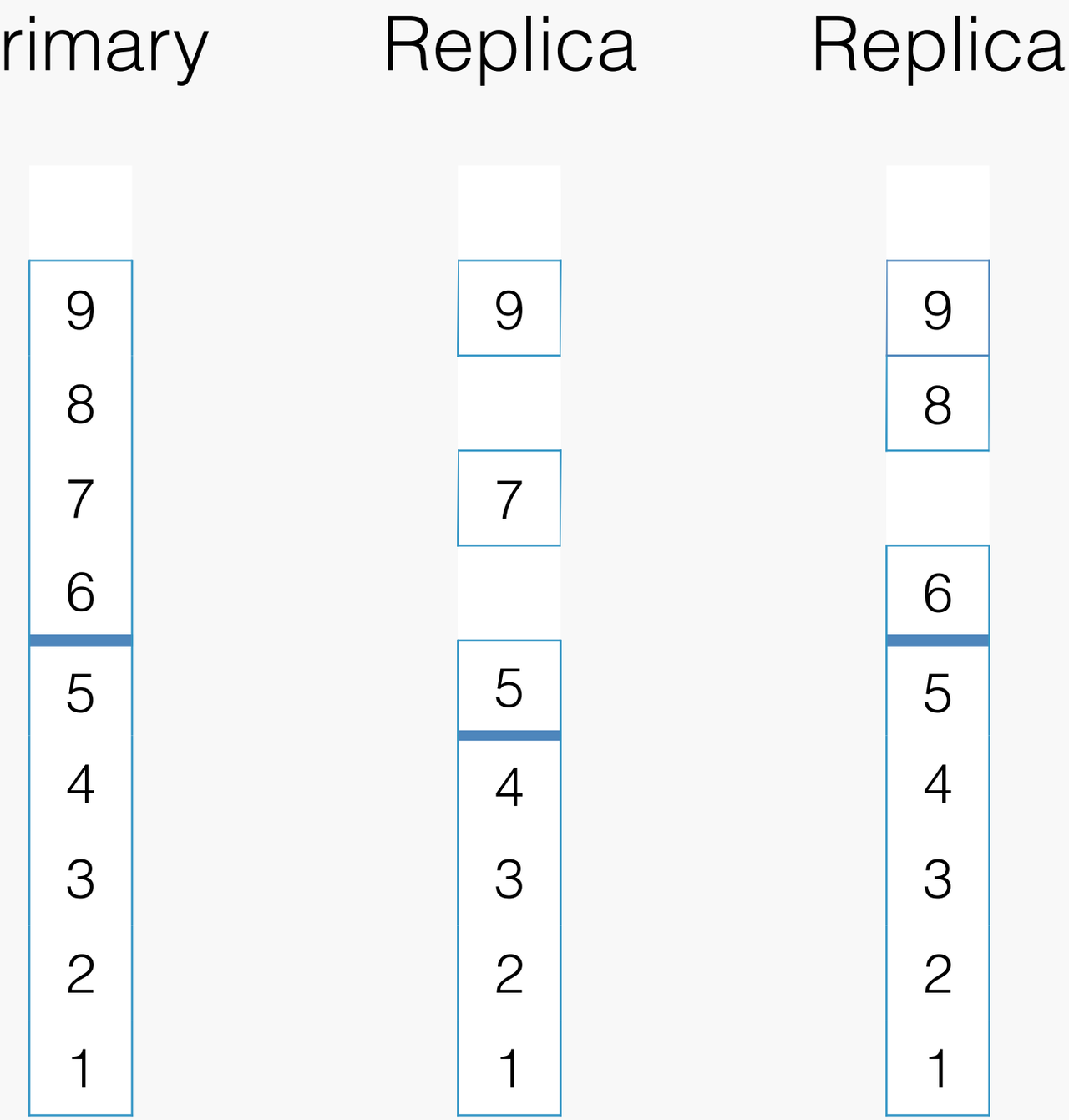
Failure, Rollback and Commitment



Primary knows what's "safe"

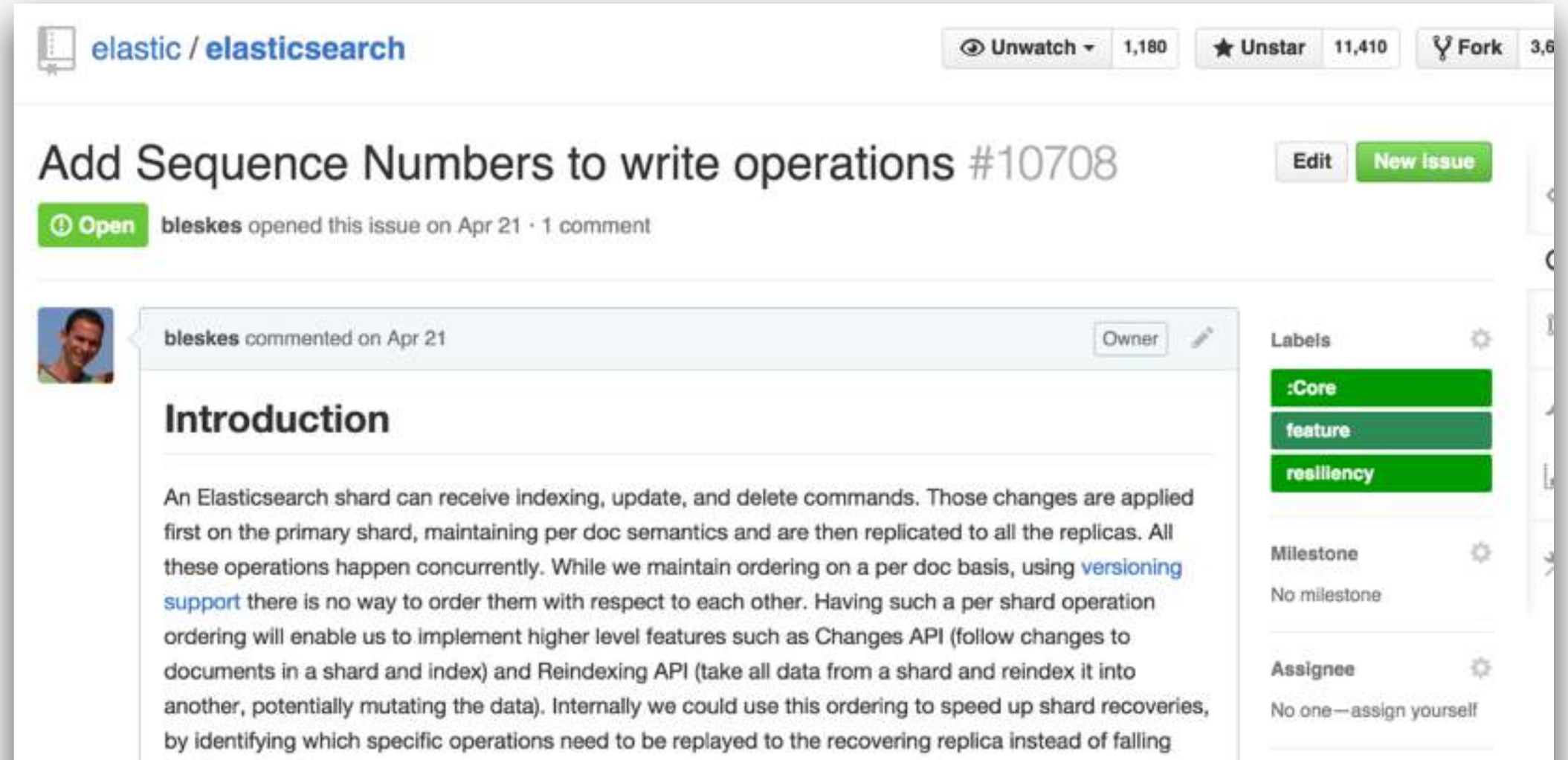


Replicas have a lagging “safe” point



Final words

- Design is pretty much nailed down
- Working on the nitty-gritty implementation details



thank you!

https://github.com/elastic/elasticsearch
https://elastic.co