

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": "blesk"
  }
}
```

```
PUT tweets/tweet/426674590560305150
{
  ...
  "text": "...",
  "user": {
```

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

```
    "name": "Boaz Leskes",
    "screen_name": "bleskes",
```

Multiple doc updates - with seq#

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

2

```
PUT tweets/tweet/426674590560305150
{
  "text": "...",
  "user": {
```

1

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

3

```
    "name": "Boaz Leskes",
    "screen_name": "bleskes",
```

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

5

4

3

2

1

4

3

2

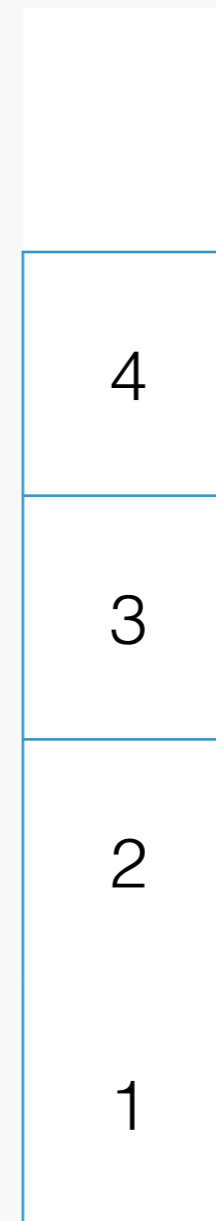
1

Primary Replica File Based Sync

Primary



Replica

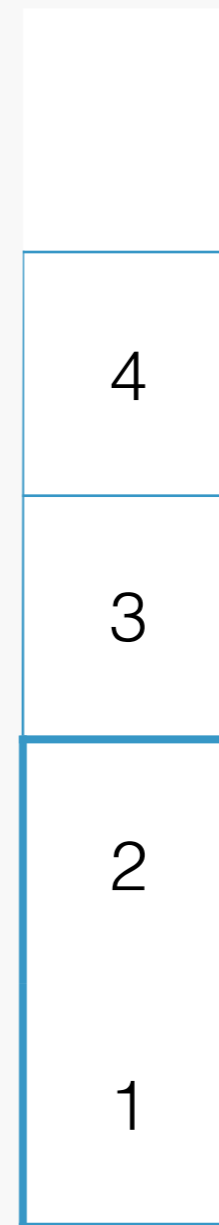


Primary Replica File Based Sync

Primary



Replica



Primary Replica Seq# Based Sync

Primary

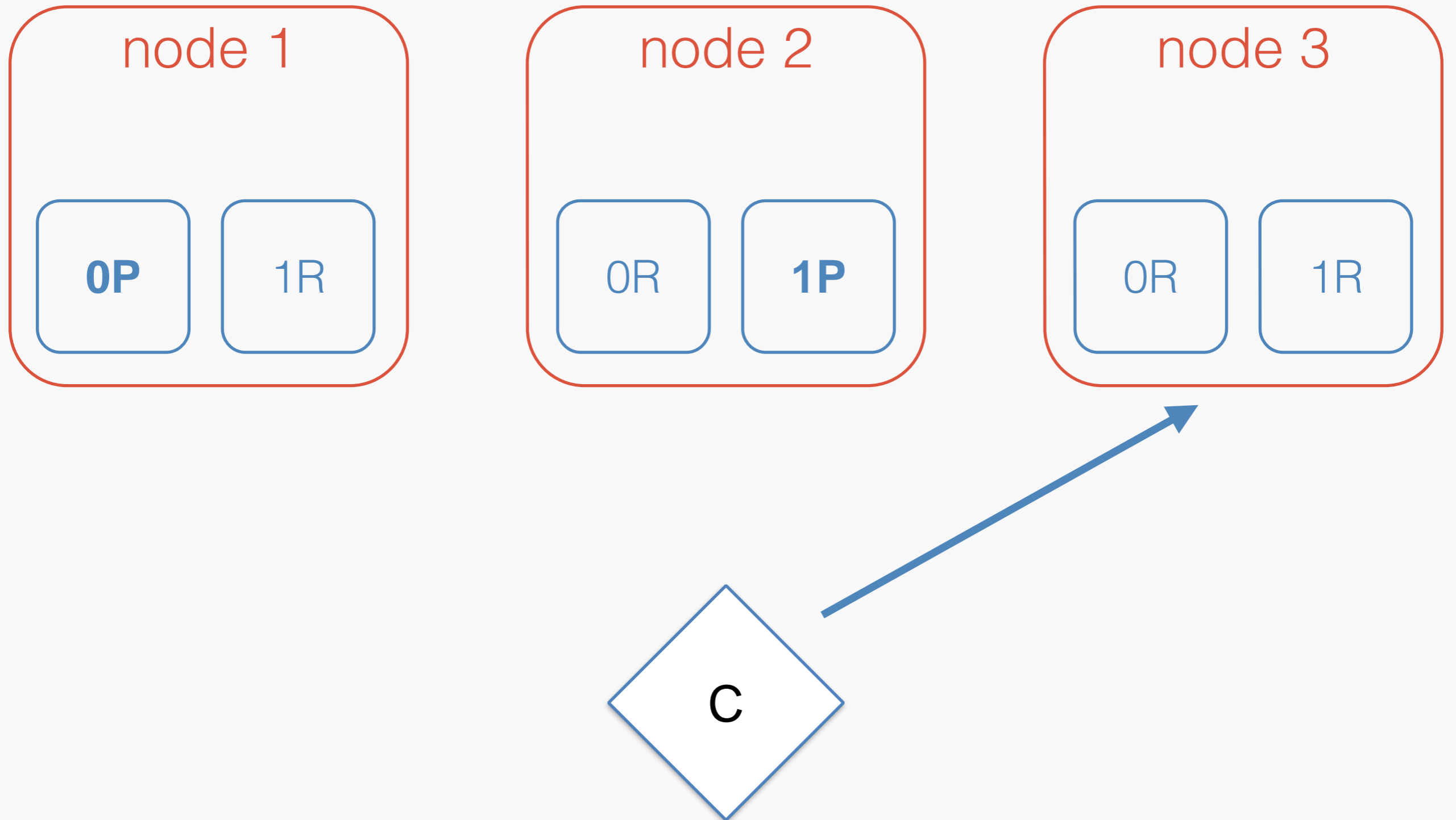


Replica

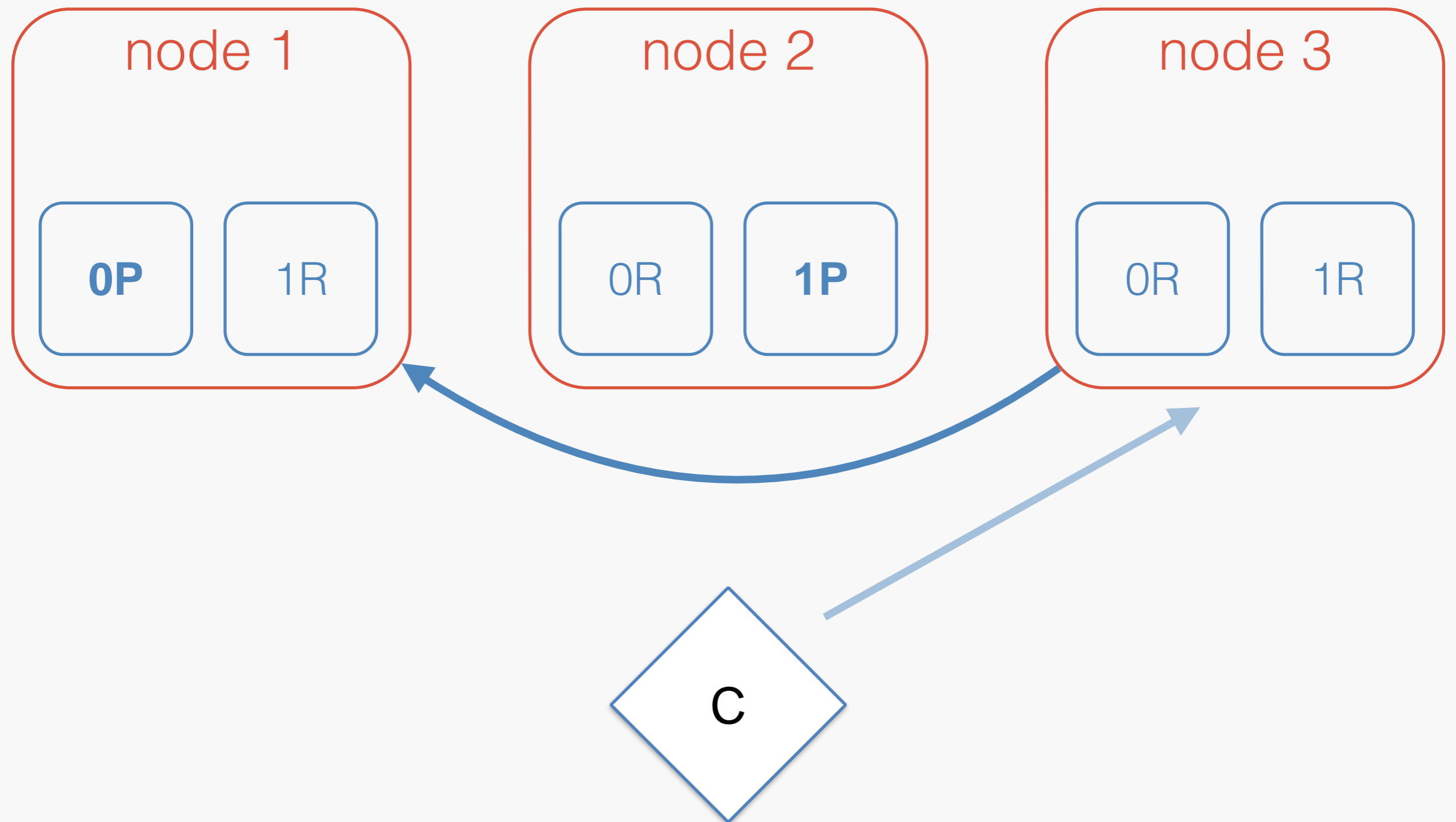


Indexing essentials

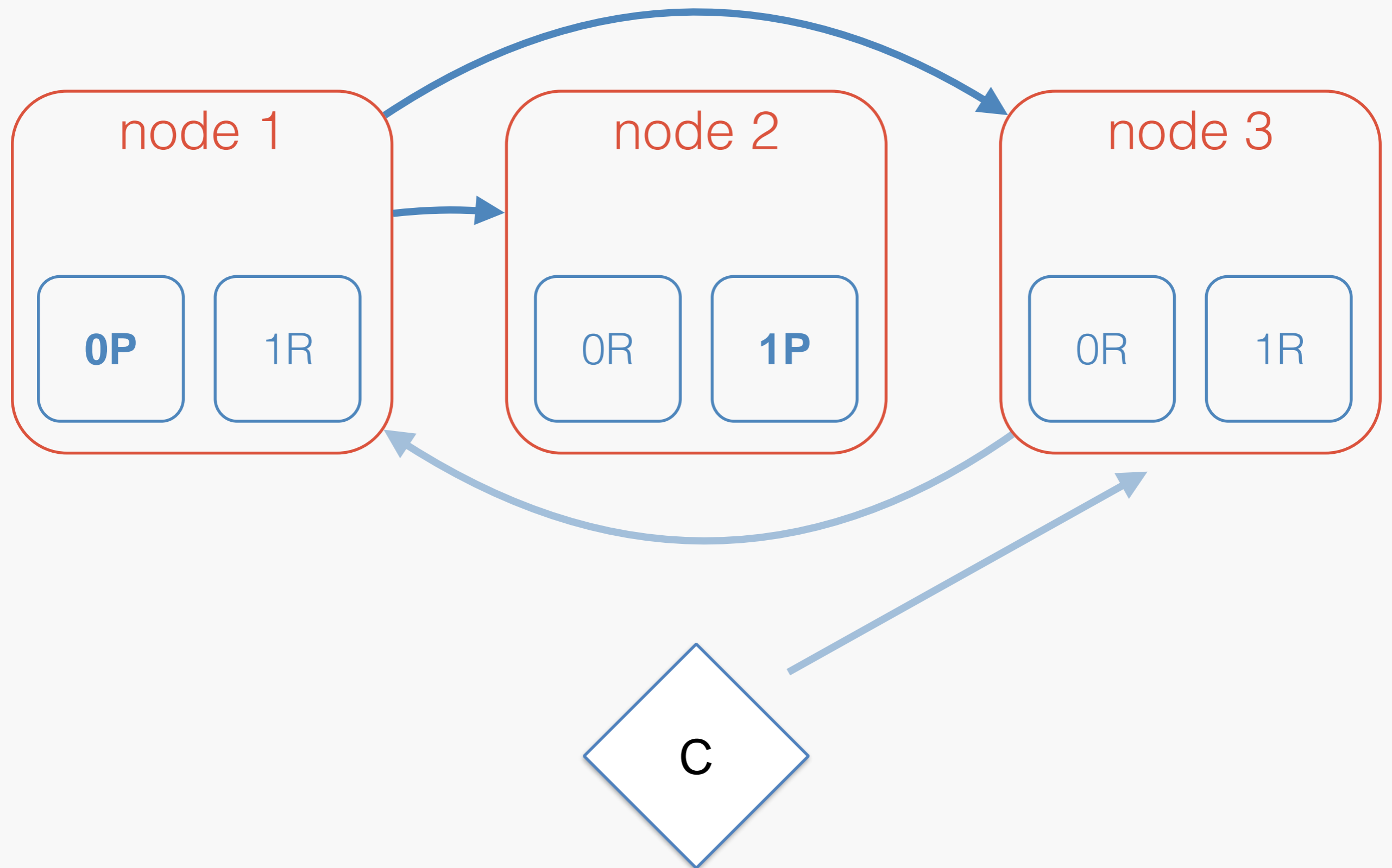
Indexing essentials



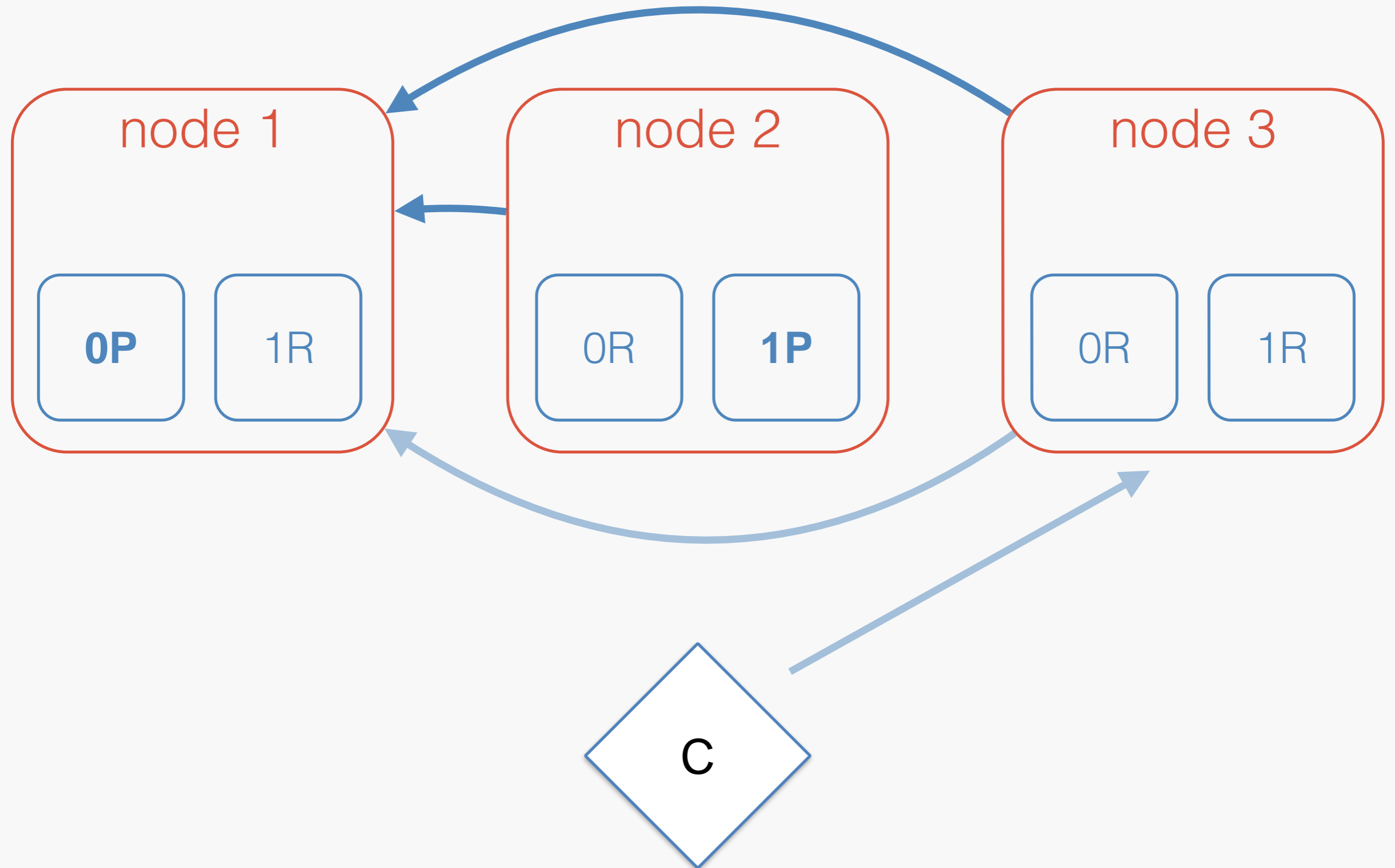
Indexing essentials



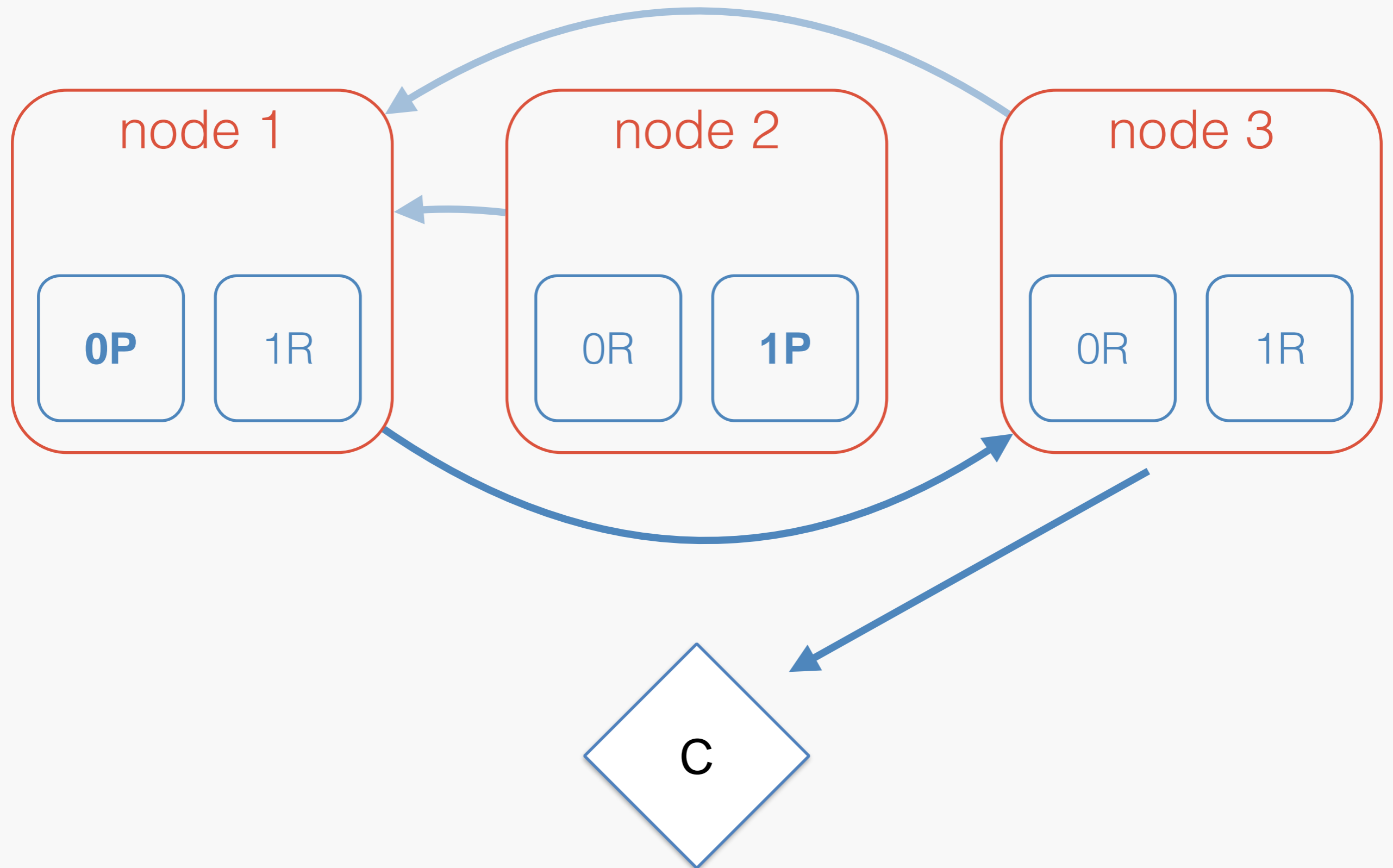
Indexing essentials



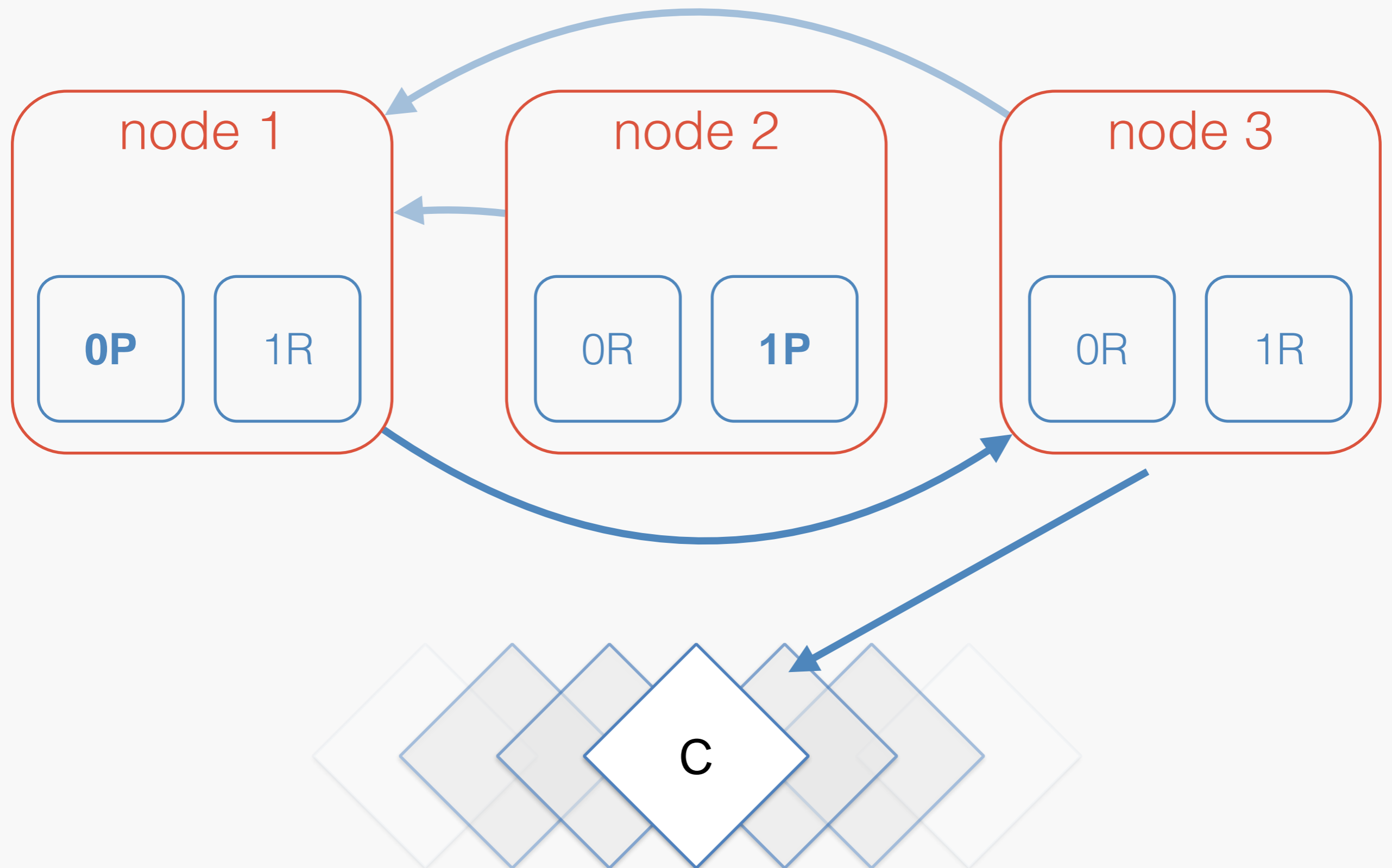
Indexing essentials



Indexing essentials



Indexing essentials

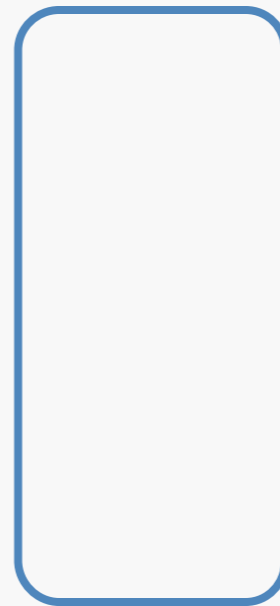


Concurrent Indexing

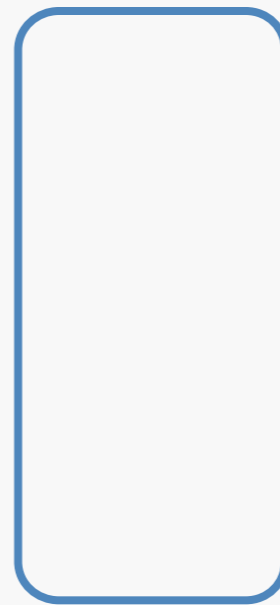
Primary



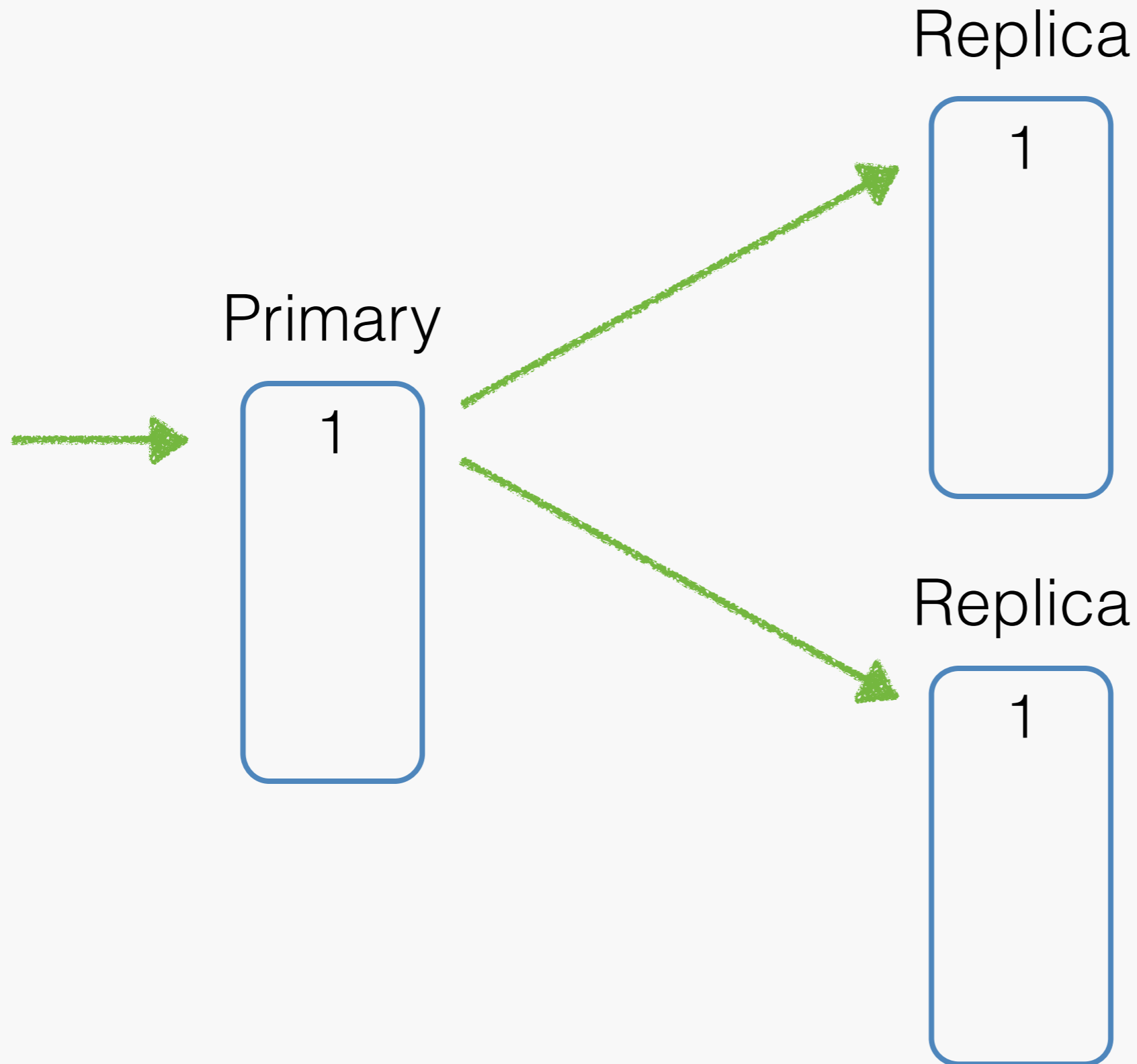
Replica



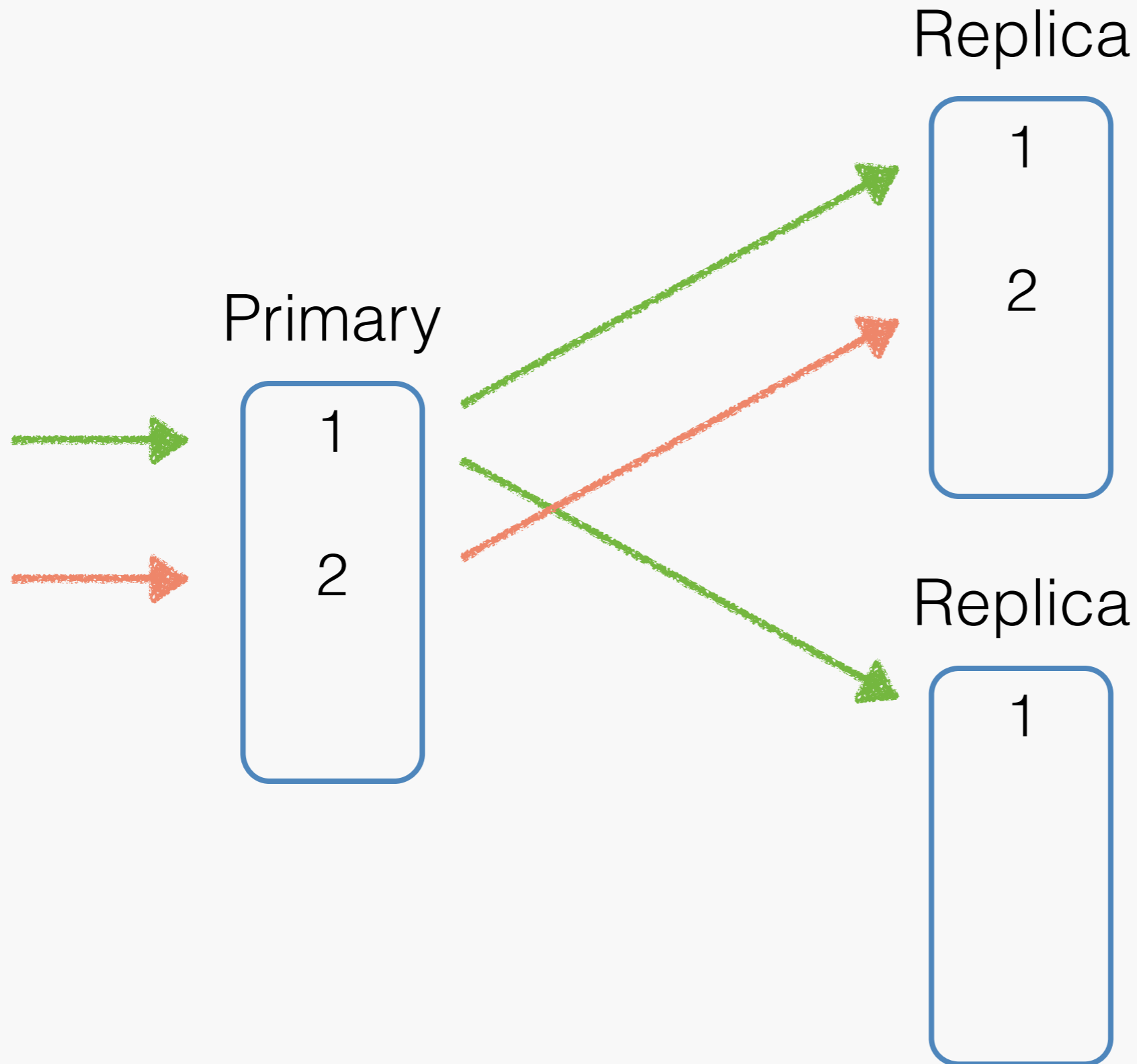
Replica



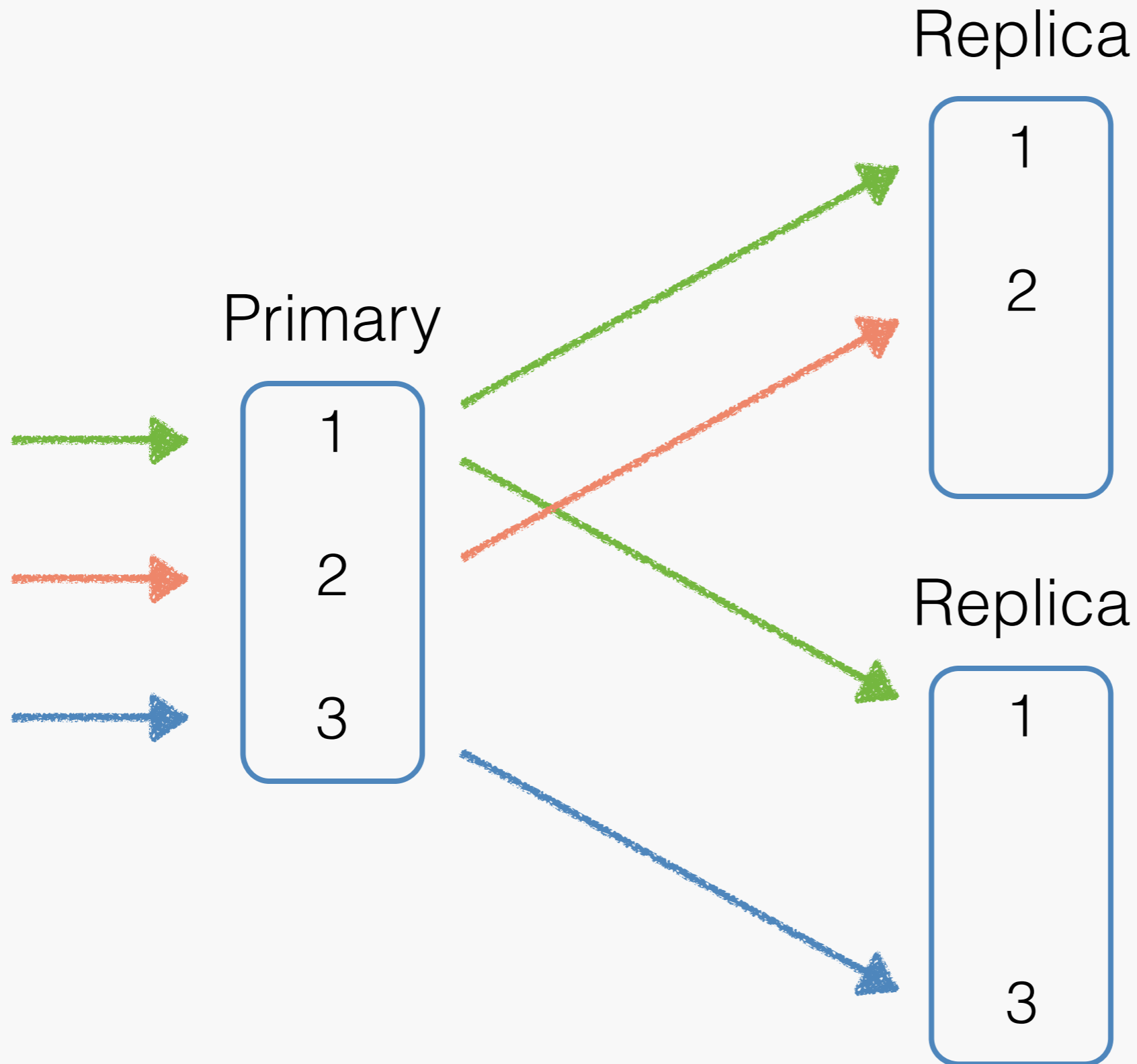
Concurrent Indexing



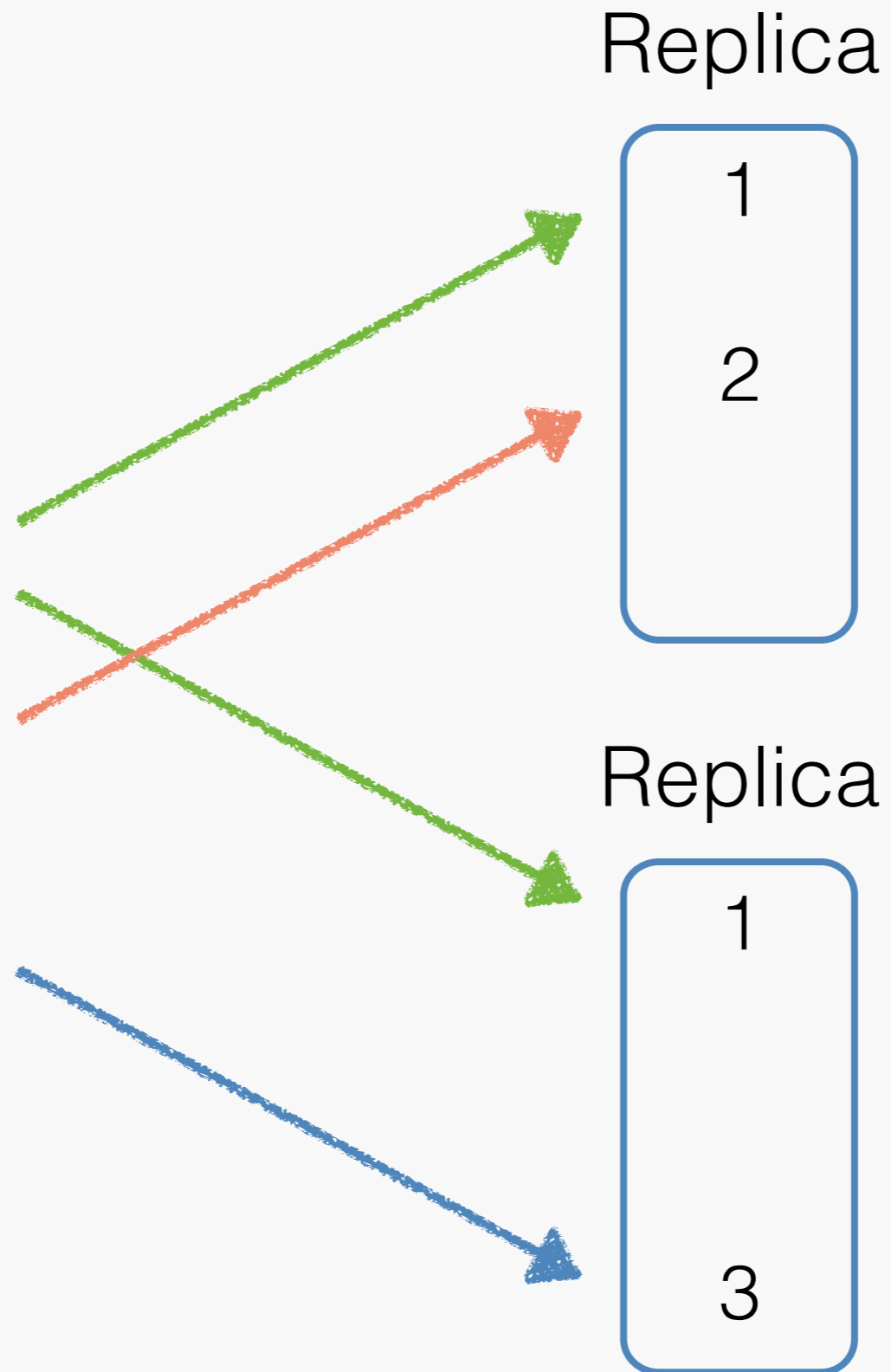
Concurrent Indexing



Concurrent Indexing



Concurrent Indexing



Requirements

- Correct :)
- Fault tolerant
- Support concurrency

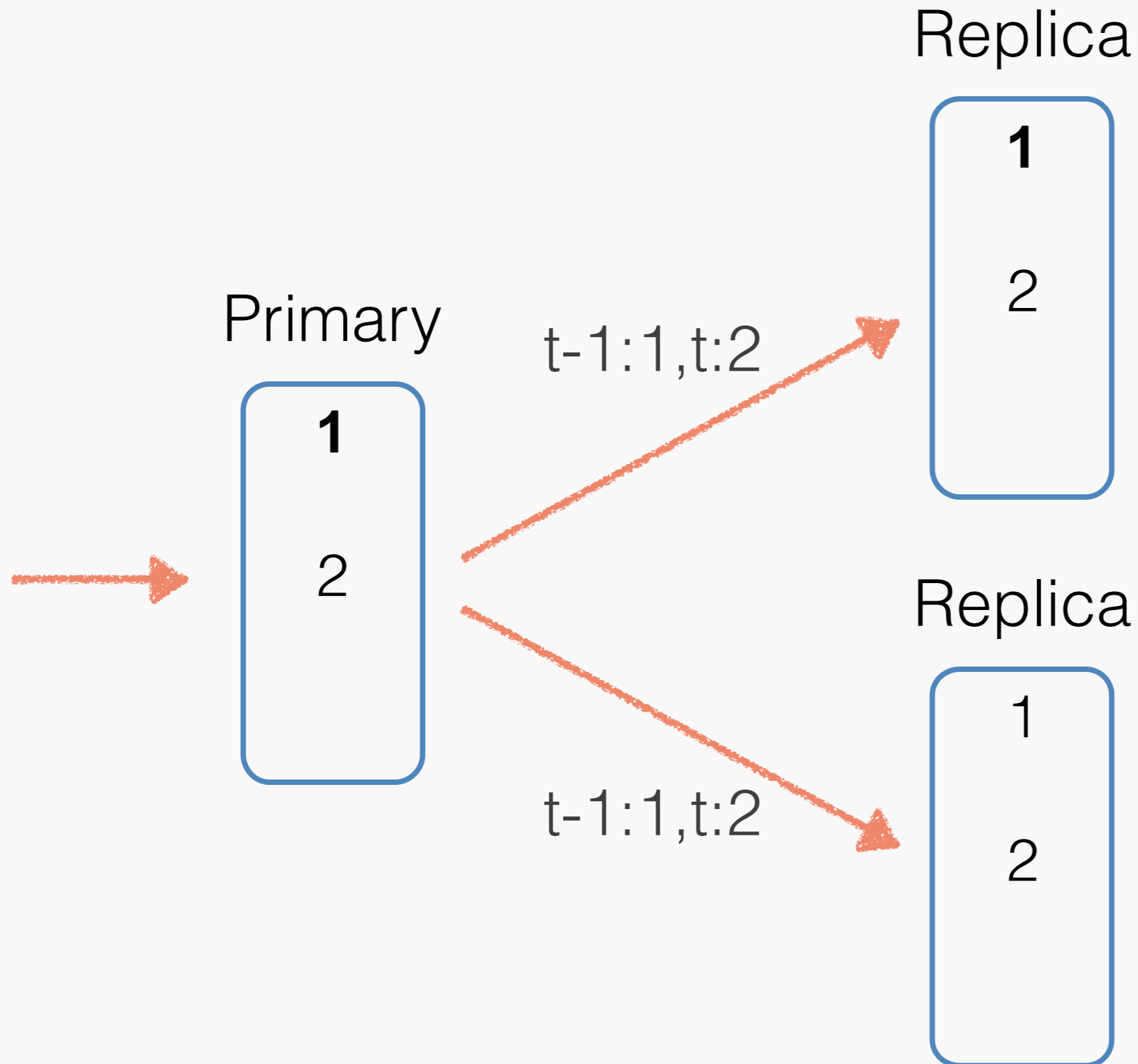
Consistency Algorithm

For example, Raft

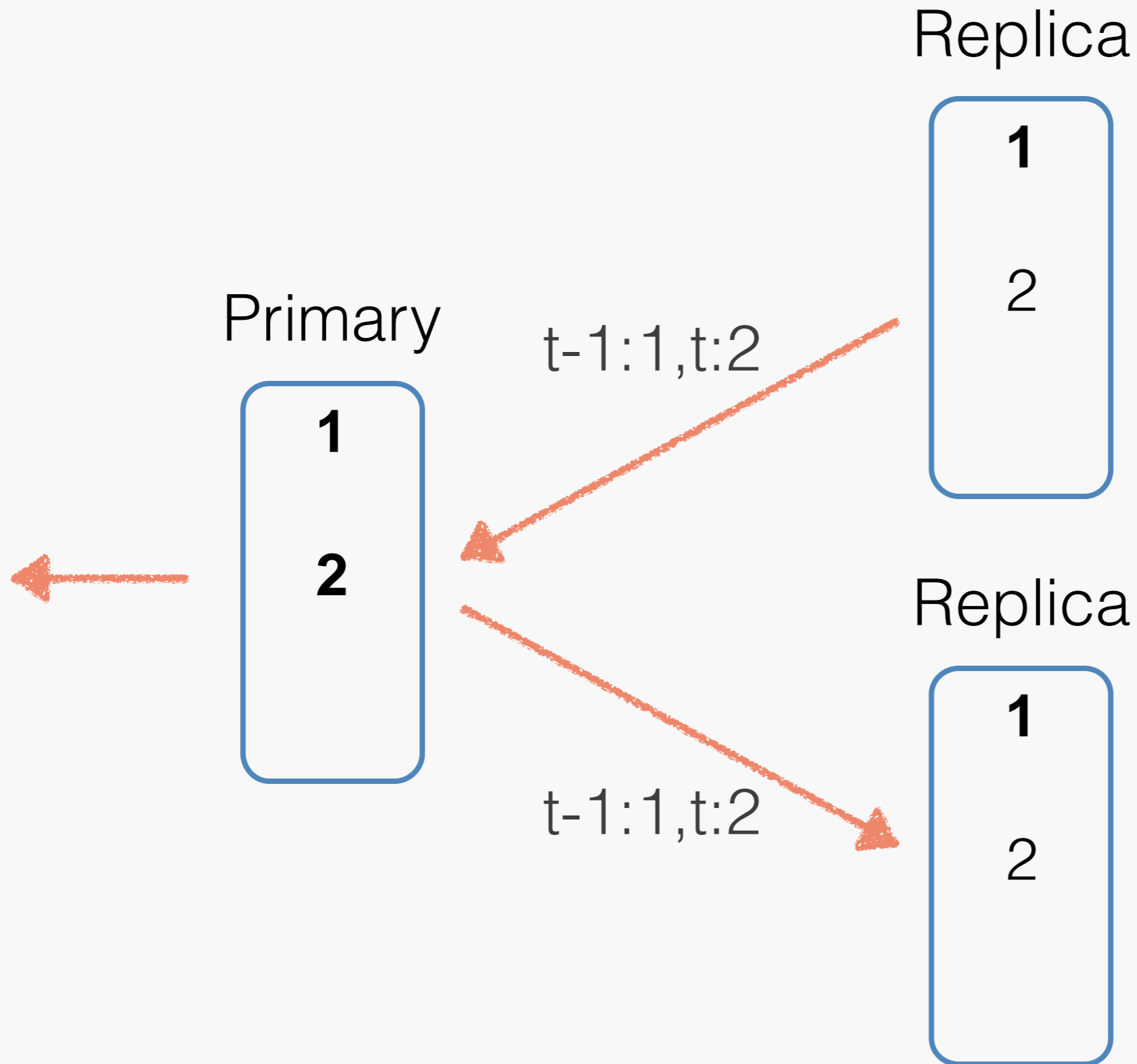
Raft Consensus Algorithm

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

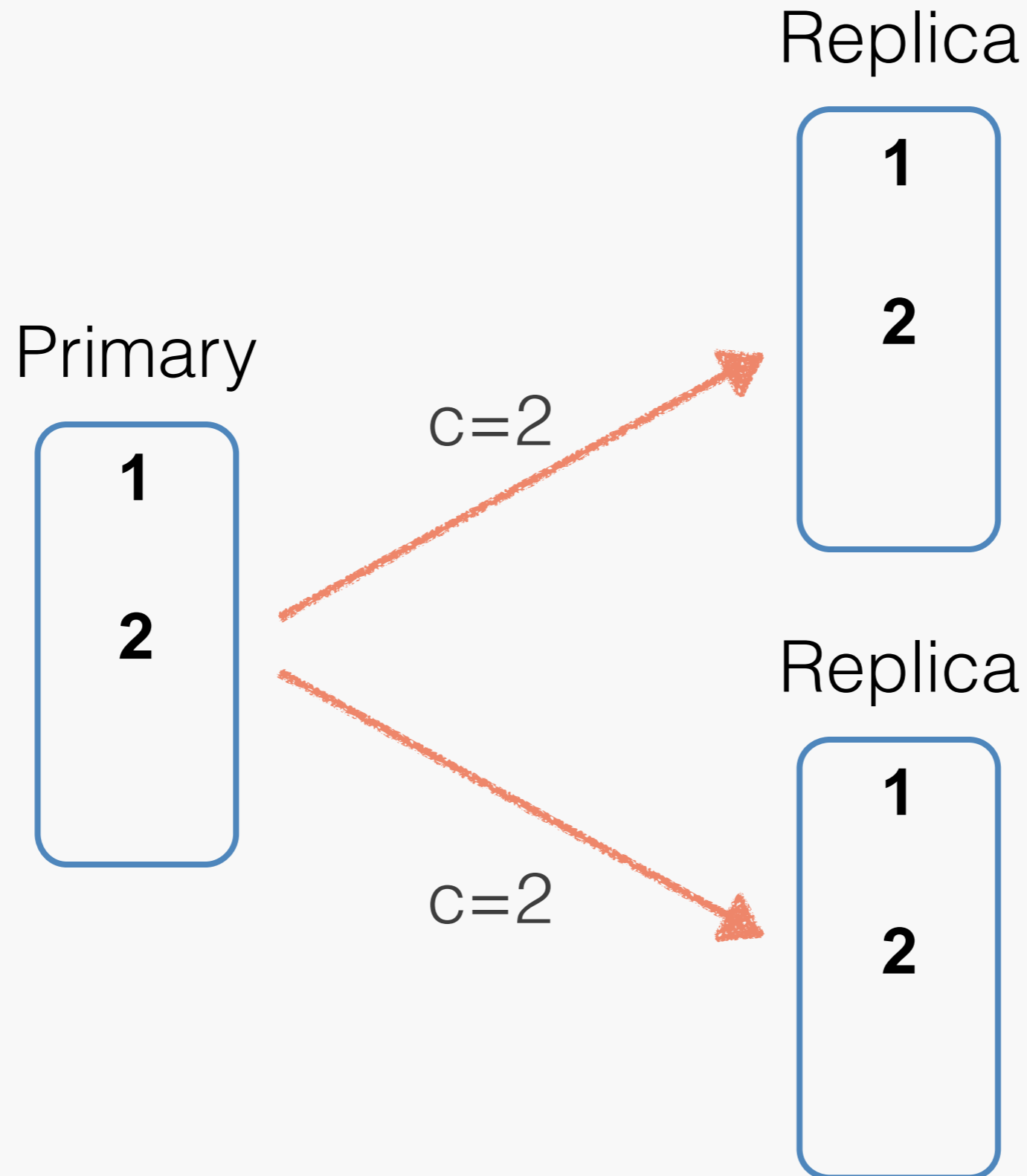
Raft - appendEntries



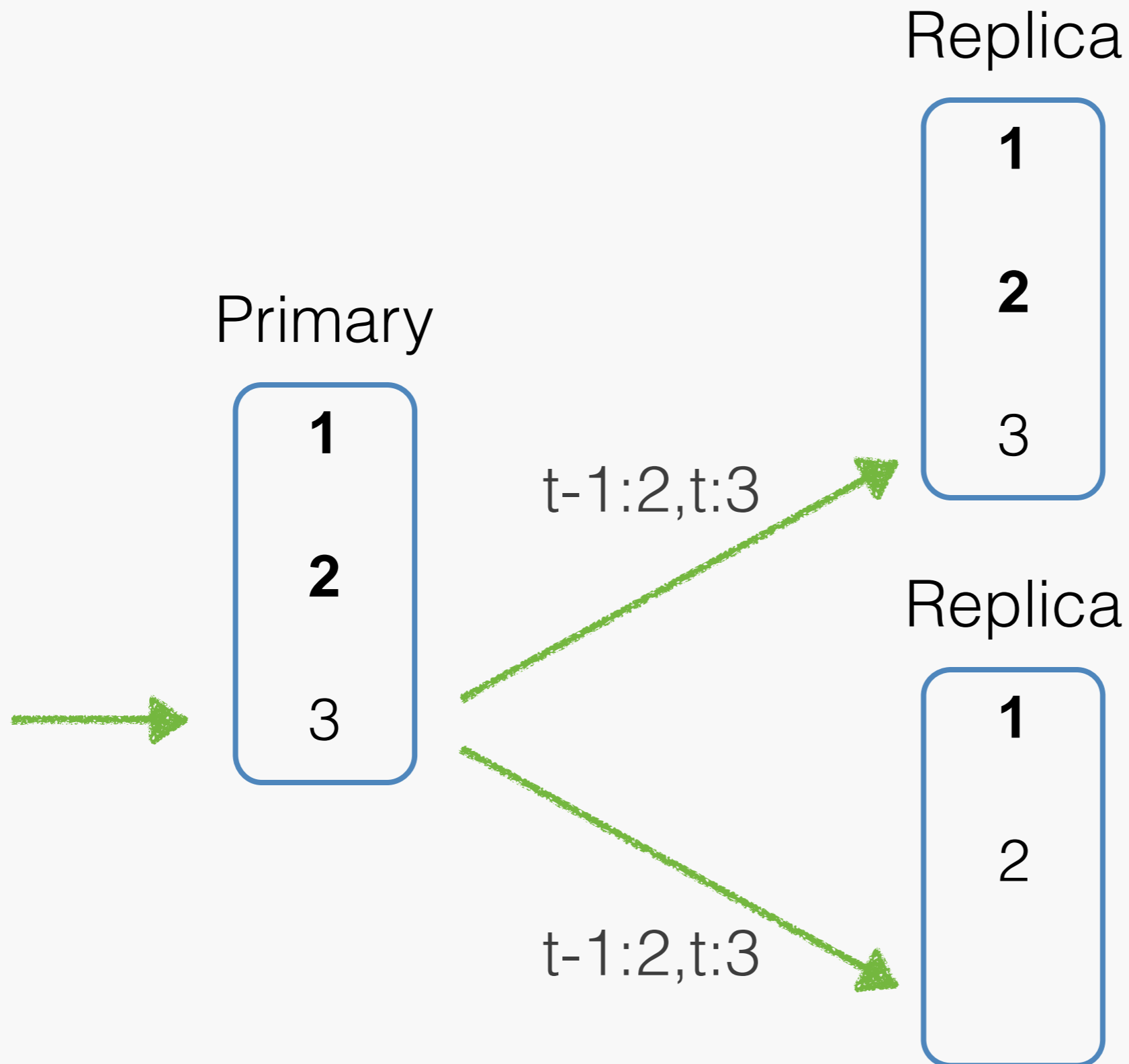
Raft - commit on quorum



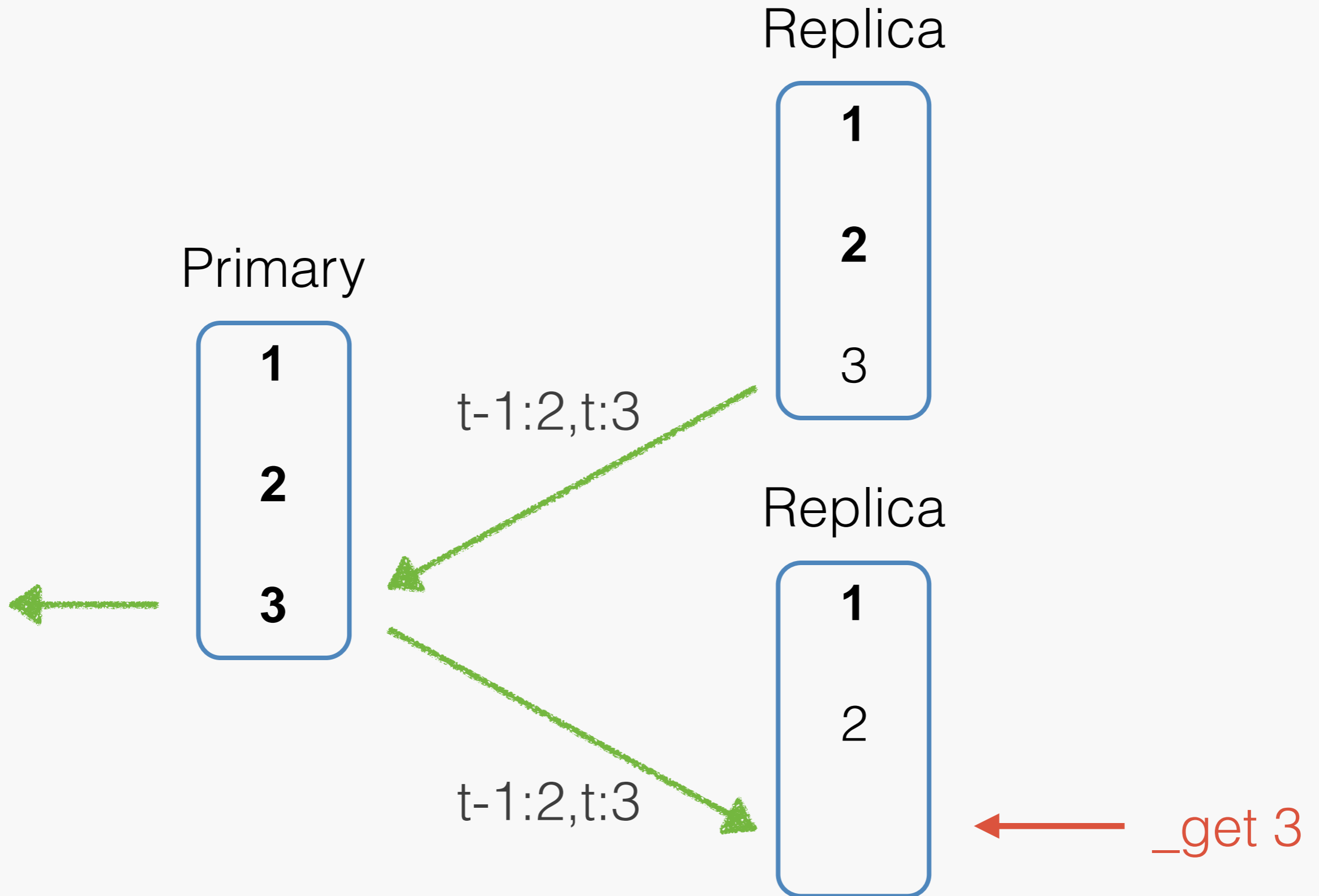
Raft - broadcast* commit



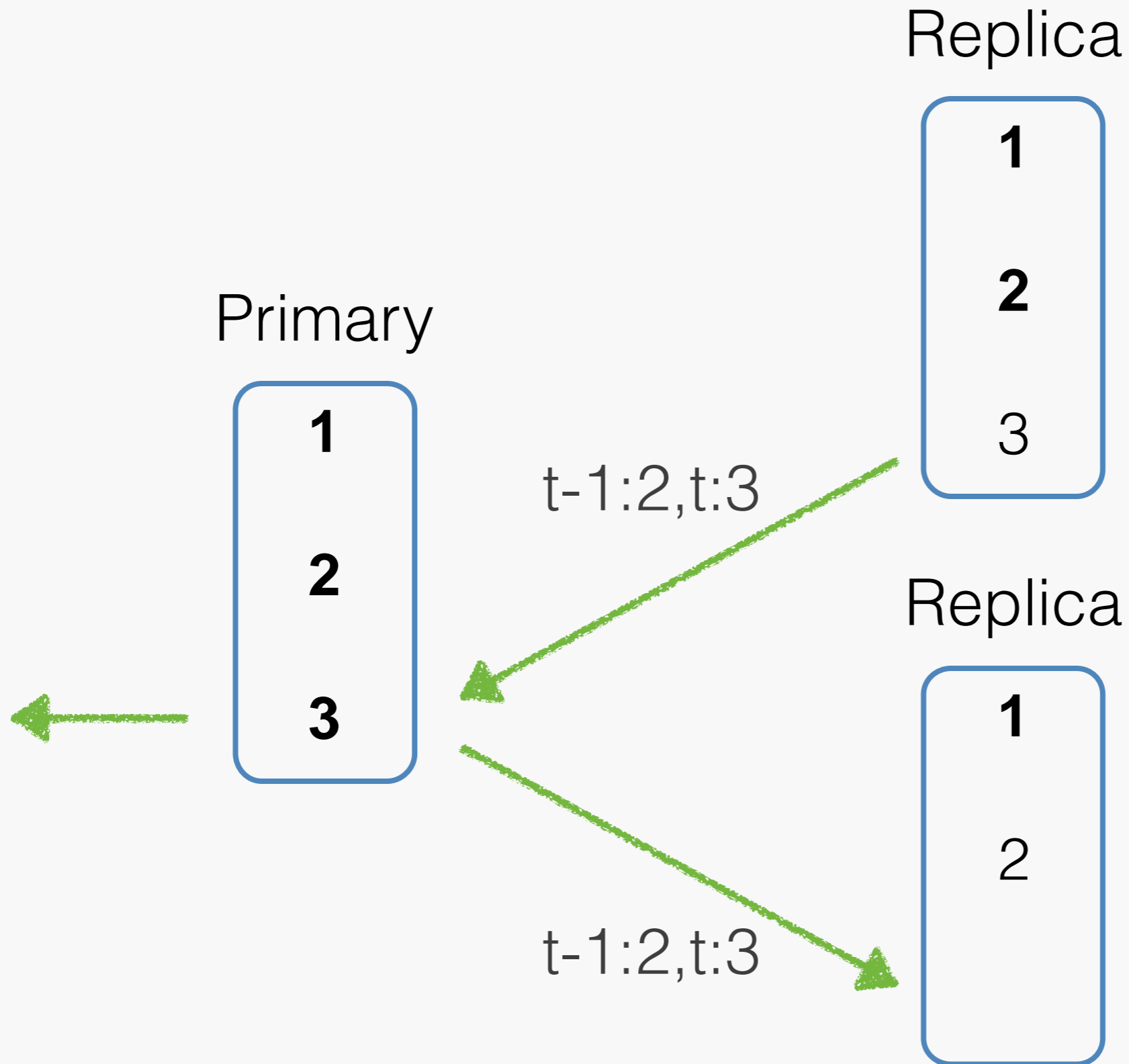
Raft - primary failure



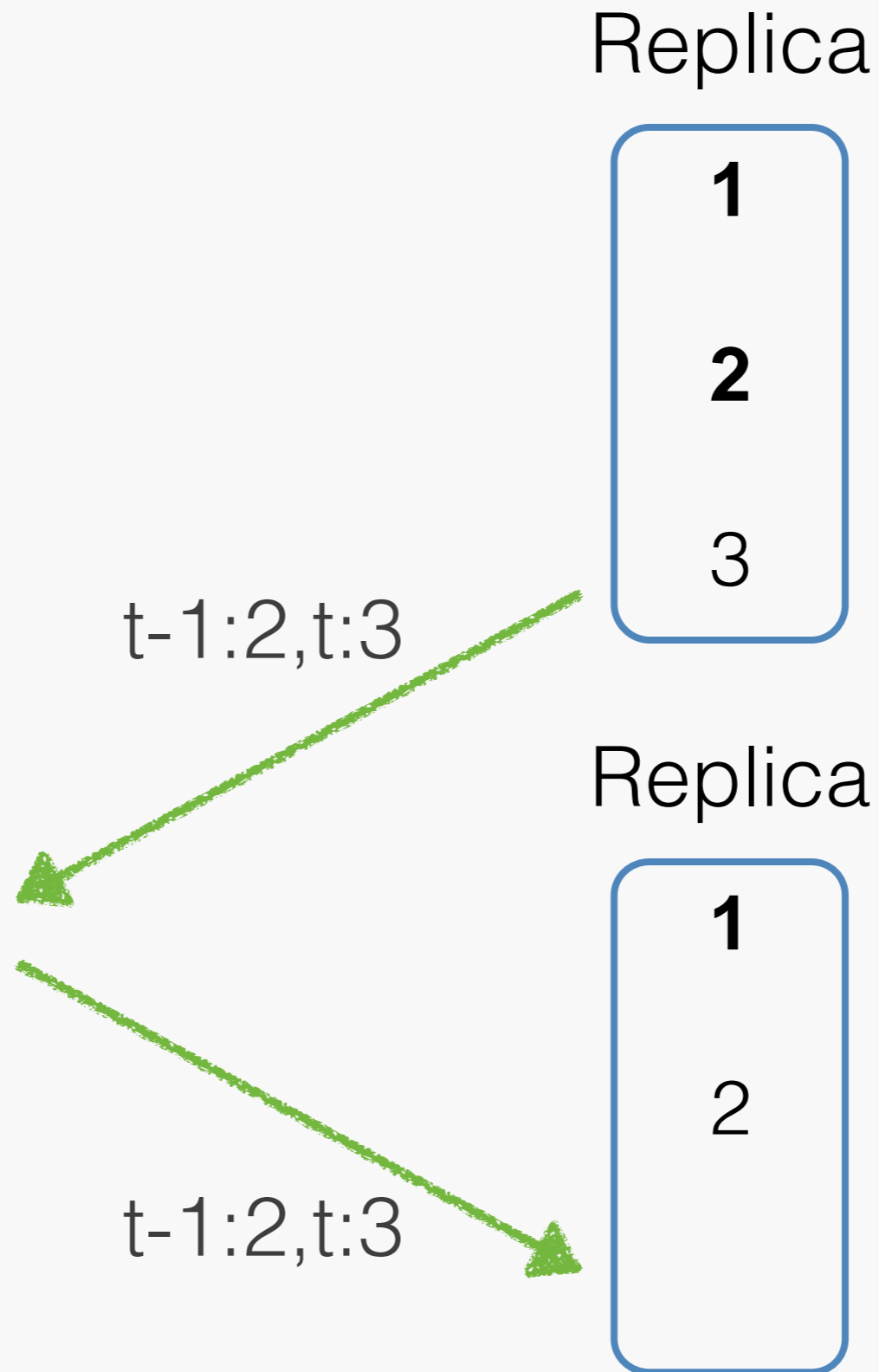
Raft - ack on quorum



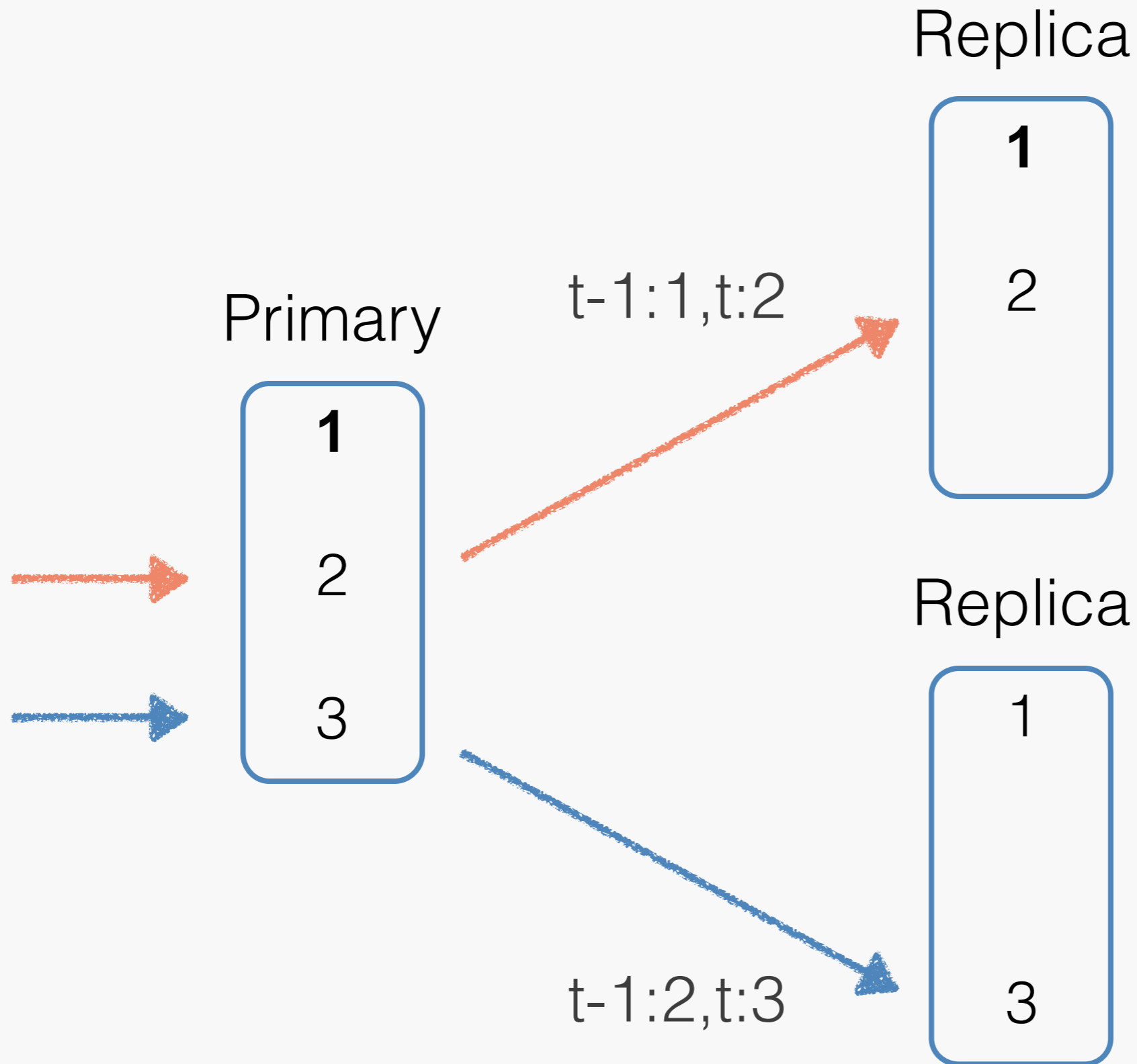
Raft - primary failure



Raft - primary failure



Raft - concurrent indexing?



Raft

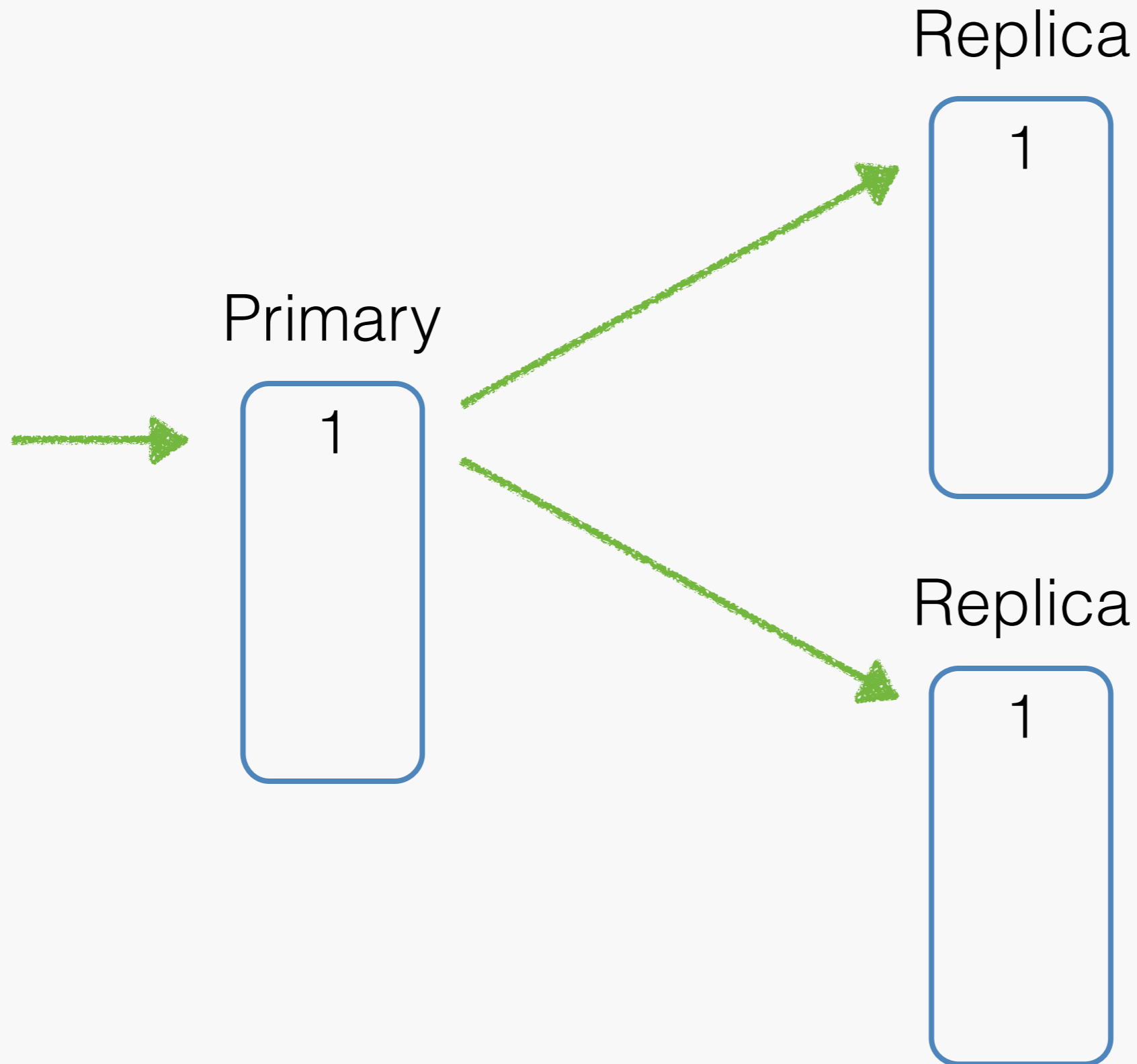
- Simple to understand
- Quorum means:
 - Lagging shards don't slow down indexing
- but
- Read visibility issues
- Tolerates up to quorum - 1 failures
- Needs at least 3 copies for correctness
- Challenges with concurrency

Master-Backup replication

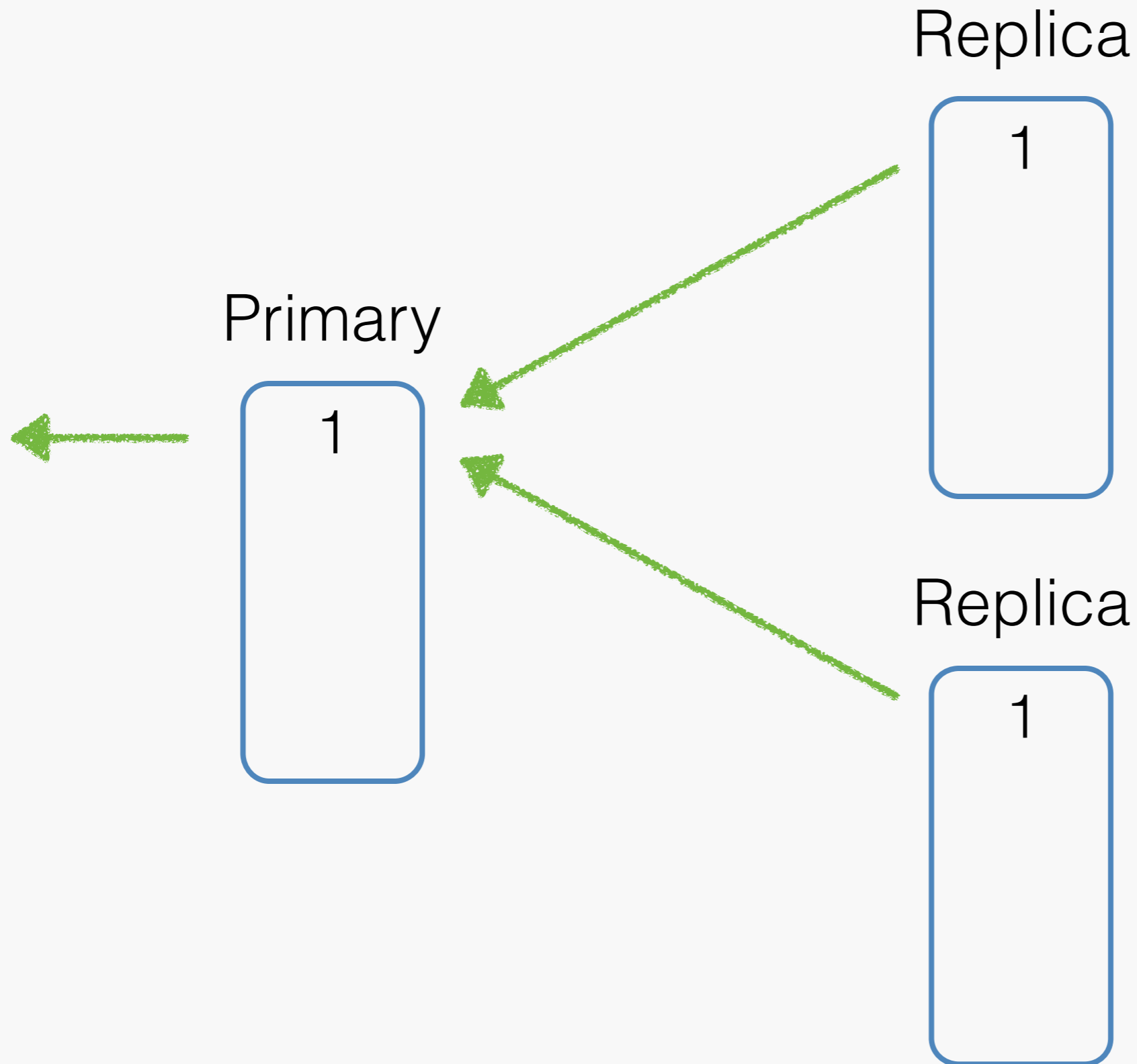
Master Backup Replication

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

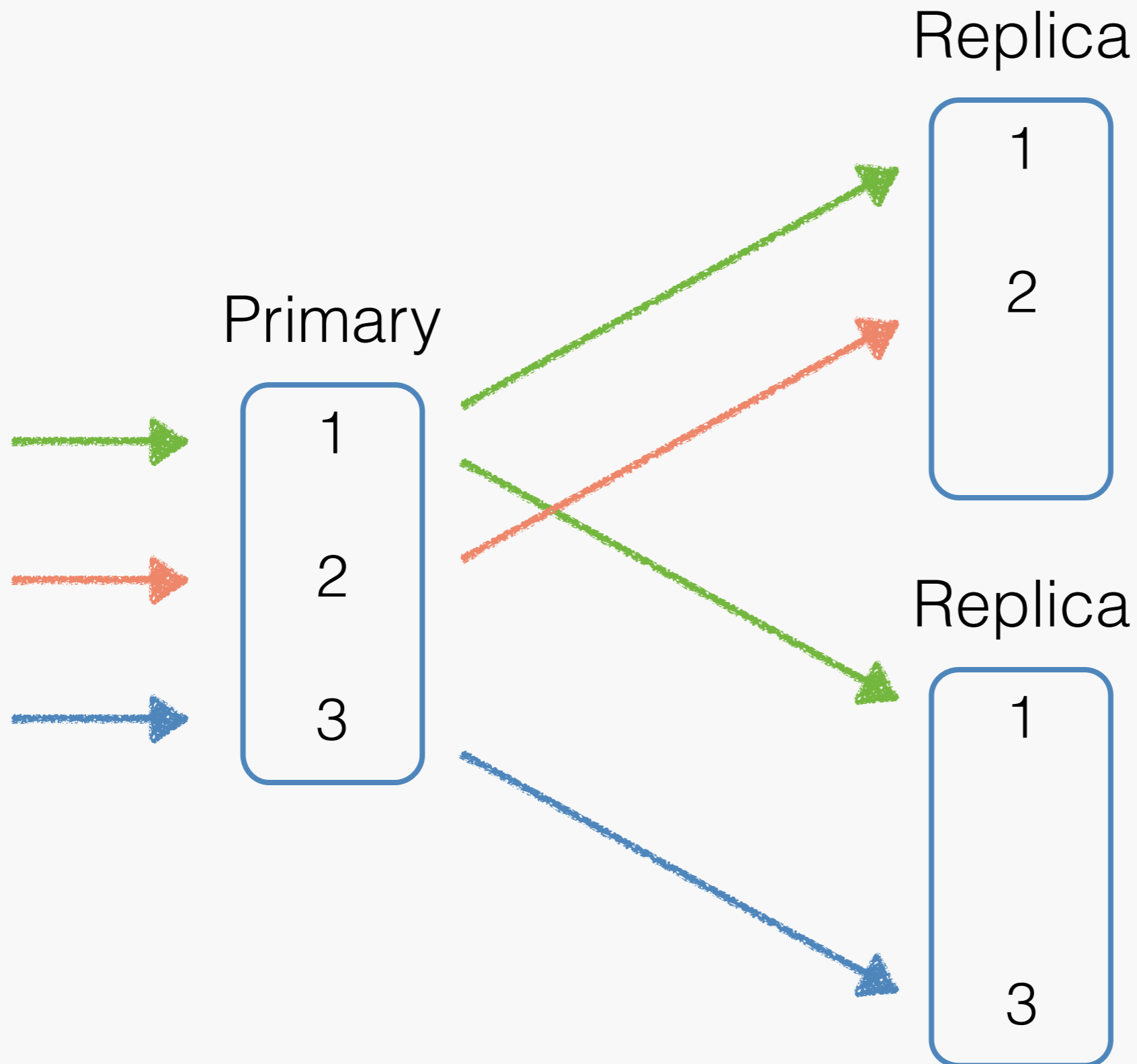
Master-Backup - indexing



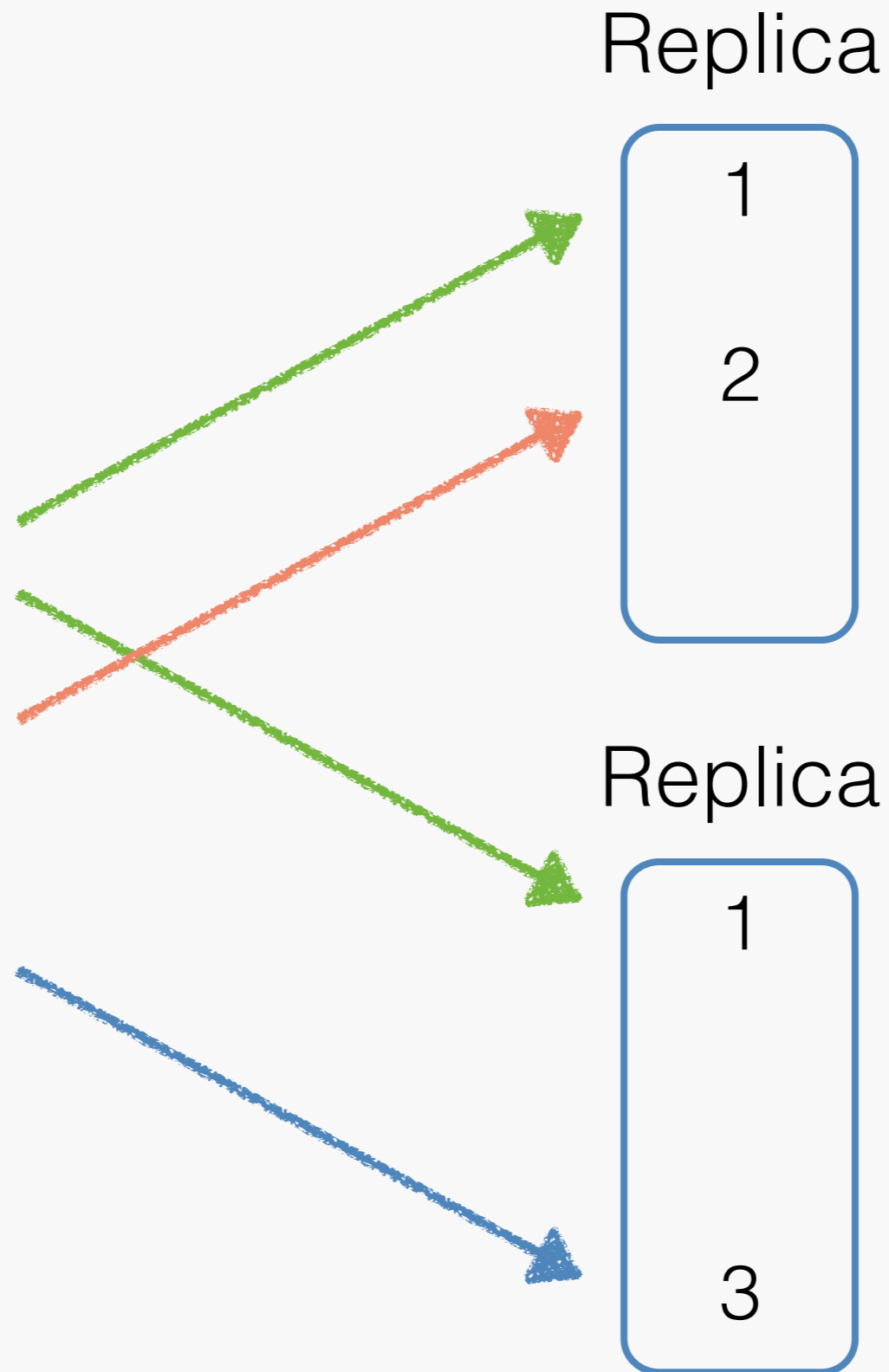
Master-Backup - indexing



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

Primary

Replica

Replica



Failure, Rollback and Commitment

Primary

Replica

Replica

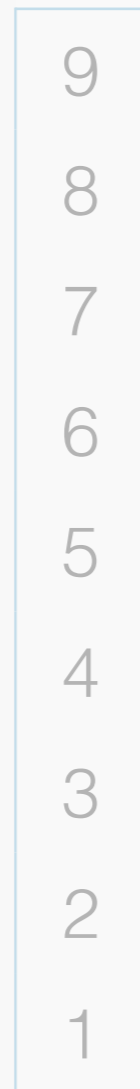


Failure, Rollback and Commitment

Primary

Replica

Replica

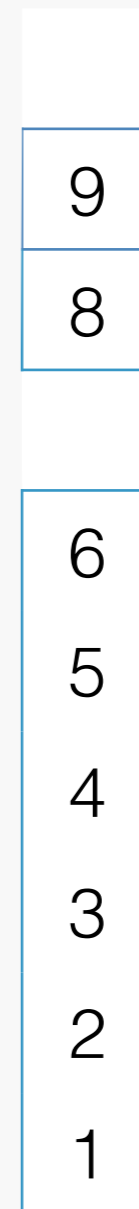


Primary knows what's "safe"

Primary

Replica

Replica

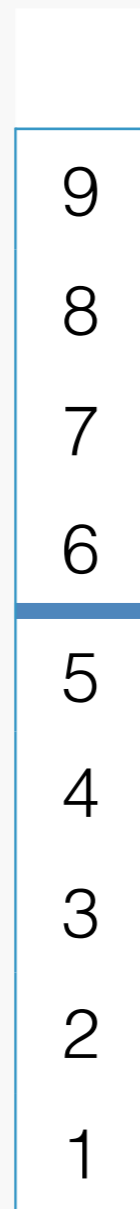


Replicas have a lagging “safe” point

Primary

Replica

Replica



Final words

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

The screenshot shows a GitHub issue page for the repository 'elastic/elasticsearch'. The issue title is 'Add Sequence Numbers to write operations #10708'. It is marked as 'Open' and was opened by 'bleskes' on April 21. The issue has 1 comment and 1,180 views. The issue is categorized with labels: ':Core', 'feature', and 'resiliency'. The issue is currently unstarred (11,410 stars) and has 3,600 forks. The issue description, titled 'Introduction', discusses the challenge of ordering operations on an Elasticsearch shard and proposes a solution using sequence numbers to ensure consistency across replicas.

elastic / elasticsearch

Unwatch 1,180 Unstar 11,410 Fork 3,600

Add Sequence Numbers to write operations #10708

Edit New Issue

Open bleskes opened this issue on Apr 21 · 1 comment

bleskes commented on Apr 21 Owner

Introduction

An Elasticsearch shard can receive indexing, update, and delete commands. Those changes are applied first on the primary shard, maintaining per doc semantics and are then replicated to all the replicas. All these operations happen concurrently. While we maintain ordering on a per doc basis, using [versioning support](#) there is no way to order them with respect to each other. Having such a per shard operation ordering will enable us to implement higher level features such as Changes API (follow changes to documents in a shard and index) and Reindexing API (take all data from a shard and reindex it into another, potentially mutating the data). Internally we could use this ordering to speed up shard recoveries, by identifying which specific operations need to be replayed to the recovering replica instead of falling

Labels

- :Core
- feature
- resiliency

Milestone

No milestone

Assignee

No one—assign yourself



thank you!

<https://github.com/elastic/elasticsearch>

<https://elastic.co>