Cassandra: Beyond Bigtable

Jonathan Ellis CTO, DataStax

Five years of Cassandra

DATASTAX

Jul-08 Jul-09 May-10 Feb-11 Dec-11 Oct-12 Jul-13

DSE

Bigtable + Dynamo

DATASTAX

Bigtable + Dynamo

DATASTAX

- LSMT / SSTables

Bigtable + Dynamo

DATASTAX

- LSMT / SSTables
  - Runtime “column” (cell) definition

Bigtable + Dynamo

DATASTAX

- LSMT / SSTables
  - Runtime “column” (cell) definition
  - Schema-agnostic

Bigtable + Dynamo

DATASTAX

- LSMT / SSTables
  - Runtime “column” (cell) definition
  - Schema-agnostic
  - Size-tiered compaction

Bigtable + Dynamo

DATASTAX

- LSMT / SSTables
  - Runtime “column” (cell) definition
  - Schema-agnostic
  - Size-tiered compaction
- Gossip-based cluster status + failure detection

Bigtable + Dynamo

DATASTAX

- LSMT / SSTables
  - Runtime “column” (cell) definition
  - Schema-agnostic
  - Size-tiered compaction
- Gossip-based cluster status + failure detection
- Hinted handoff

Bigtable + Dynamo

DATASTAX

- LSMT / SSTables
  - Runtime “column” (cell) definition
  - Schema-agnostic
  - Size-tiered compaction
- Gossip-based cluster status + failure detection
- Hinted handoff
- Read repair

Bigtable + Dynamo

DATASTAX

- LSMT / SSTables
  - Runtime “column” (cell) definition
  - Schema-agnostic
  - Size-tiered compaction
- Gossip-based cluster status + failure detection
- Hinted handoff
- Read repair
- Anti-entropy repair

Bigtable + Dynamo

DATASTAX

- LSMT / SSTables
  - Runtime “column” (cell) definition
  - Schema-agnostic
  - Size-tiered compaction
- Gossip-based cluster status + failure detection
- Hinted handoff
- Read repair
- Anti-entropy repair
- Eventually consistent

… with some differences

DATASTAX

… with some differences

DATASTAX

- SuperColumns

… with some differences

DATASTAX

- SuperColumns
- Indexes

… with some differences

DATASTAX

- SuperColumns
- Indexes
- Timestamp-based conflict resolution
  http://www.datastax.com/dev/blog/why-cassandra-doesnt-need-vector-clocks

Bigtable-inspired API

DATASTAX

```
list<ColumnOrSuperColumn> get_slice(
        1:required binary key,
        2:required ColumnParent column_parent,
        3:required SlicePredicate predicate,
        4:required ConsistencyLevel consistency_level)
```

Two years ago

DATASTAX

- CQL: native protocol, prepared statements
- Triggers
- Entity groups
- Smarter range queries enabling Hive predicate push-down
- Blue sky: streaming / CEP
- Ease Of Use

Two years ago

- CQL: native protocol, prepared statements
- Triggers
- Entity groups
- Smarter range queries enabling Hive predicate push-down
- Blue sky: streaming / CEP
- Ease Of Use

Two years ago


- CQL: native protocol, prepared statements
- Triggers
- Entity groups
- Smarter range queries enabling Hive predicate push-down
- Blue sky: streaming / CEP
- Ease Of Use

Two years ago



- CQL: native protocol, prepared statements
- Triggers
- Entity groups
- Smarter range queries enabling Hive predicate push-down
- Blue sky: streaming / CEP
- Ease Of Use

Two years ago

DATASTAX

- CQL: native protocol, prepared statements
- Triggers
- Entity groups
- Smarter range queries enabling Hive predicate push-down
- Blue sky: streaming / CEP
- Ease Of Use

Two years ago

DATASTAX

-  CQL: native protocol, prepared statements
-  Triggers
-  Entity groups
-  Smarter range queries enabling Hive predicate push-down
- Blue sky: streaming / CEP
-  Ease Of Use

Two years ago

DATASTAX

- CQL: native protocol, prepared statements
- Triggers
- Entity groups
- Smarter range queries enabling Hive predicate push-down
- ~~Blue sky: streaming / CEP~~
- Ease Of Use

User defined types

DATASTAX

```
CREATE TYPE address (
  street text,
  city text,
  zip_code int,
  phones set<text>
)

CREATE TABLE users (
  id uuid PRIMARY KEY,
  name text,
  addresses map<text, address>
)

SELECT id, name, addresses.city, addresses.phones FROM users;

      id |    name | addresses.city |         addresses.phones
---------------------+-----------------+--------------------------
 63bf691f | jbellis |         Austin | {'512-4567', '512-9999'}
```

Collection indexing

DATASTAX

```
 CREATE TABLE songs (
  id uuid PRIMARY KEY,
  artist text,
  album text,
  title text,
  data blob,
  tags set<text>
);
```

```
CREATE INDEX song_tags_idx ON songs(tags);

SELECT * FROM songs WHERE 'blues' IN tags;

 id       | album          | artist             | tags                       | title
----------+----------------+--------------------+----------------------------+------------------
 5027b27e | Country Blues | Lightnin' Hopkins | {'acoustic', 'blues'} | Worrying My Mind
```

Cassandra is a...

DATASTAX

- Partitioned row store with extensions
- Typed document database
- Object database
- ?

Paxos / CAS

DATASTAX

Session 1

```
SELECT * FROM users
WHERE username =
'jbellis'
```

```
[empty resultset]
```

```
INSERT INTO users (...)
VALUES ('jbellis', ...)
```

Session 2

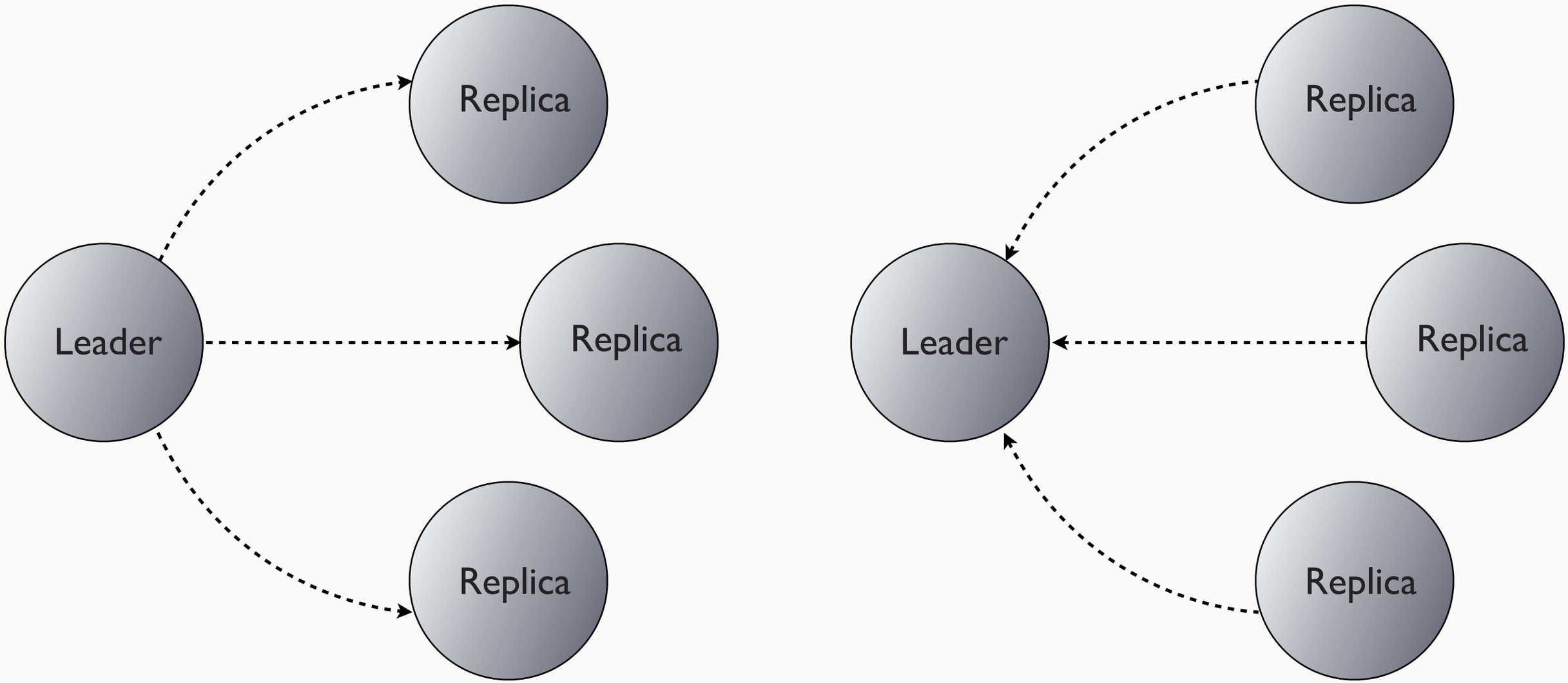
```
SELECT * FROM users
WHERE username =
'jbellis'
```

```
[empty resultset]
```

```
INSERT INTO users (...)
VALUES ('jbellis', ...)
```

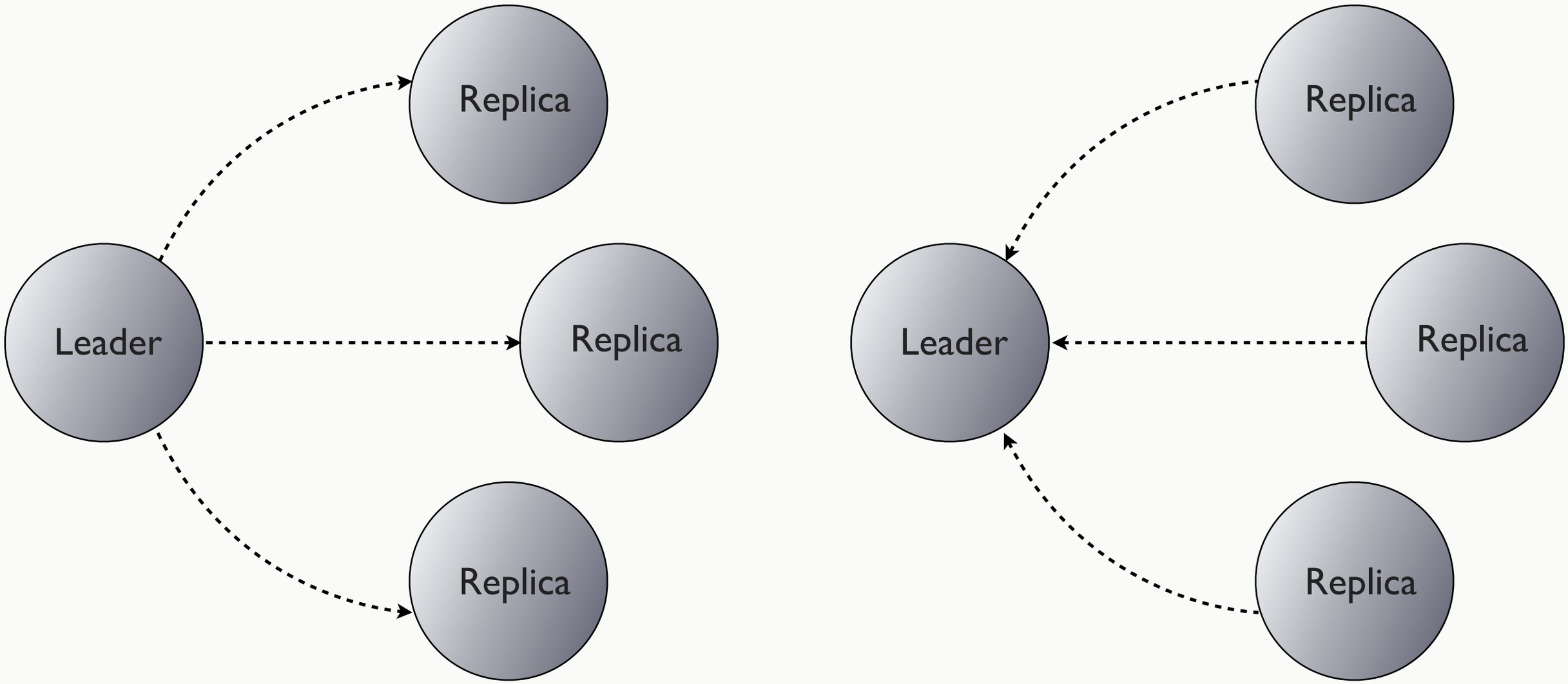
Prepare / promise

DATASTAX

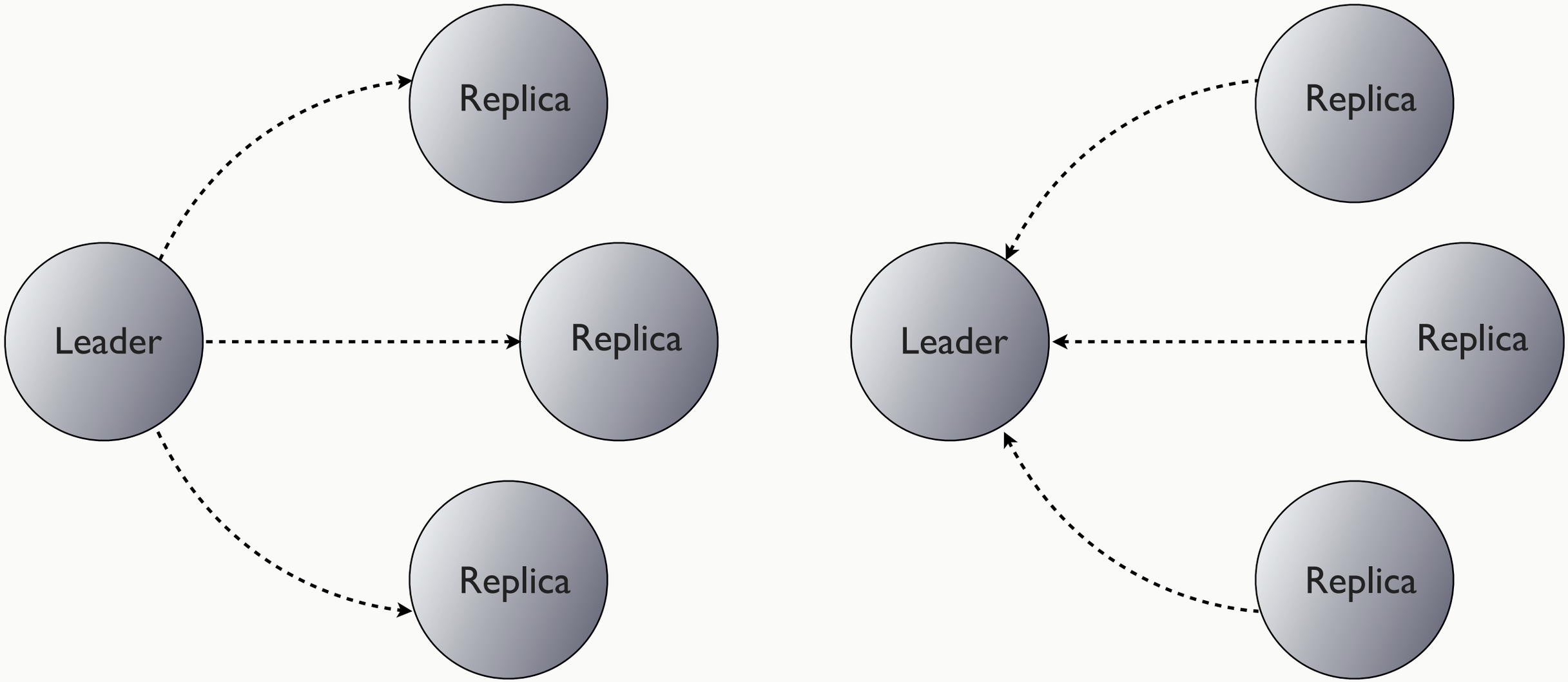


Propose / accept

DATASTAX

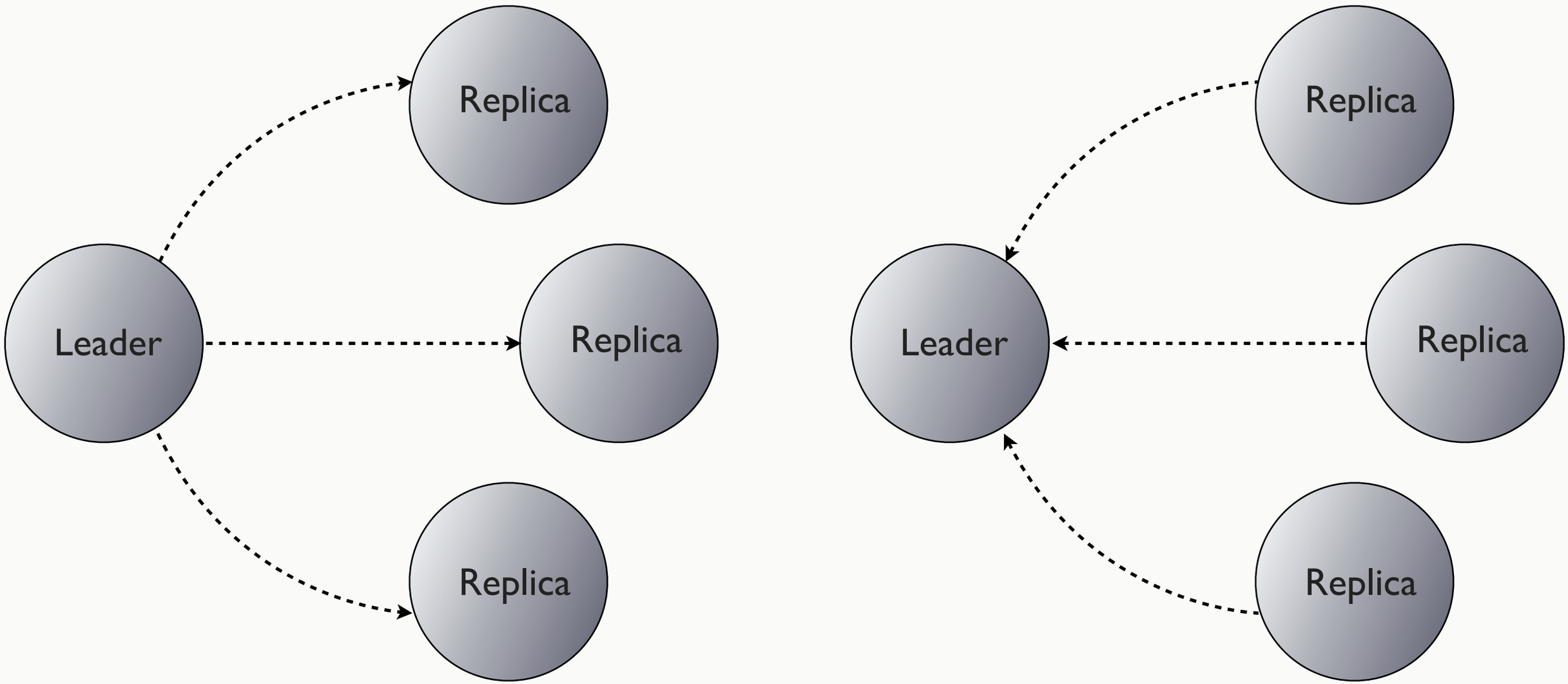


Read / results



Commit / acknowledge

DATASTAX



Paxos state

DATASTAX

```
CREATE TABLE paxos (
    row_key blob,
    cf_id UUID,
    in_progress_ballot timeuuid,
    proposal_ballot timeuuid,
    proposal blob,
    most_recent_commit_at timeuuid,
    most_recent_commit blob,
    PRIMARY KEY (row_key, cf_id)
)
```

Implications

- 4 round trips vs 1 for normal updates
- Paxos state is durable
- Linearizable consistency with no leader election or failover
- ConsistencyLevel.SERIAL
- http://www.datastax.com/dev/blog/lightweight-transactions-in-cassandra-2-0

Syntax

DATASTAX

```
INSERT INTO USERS (username, email, ...)
VALUES (‘jbellis’, ‘jbellis@datastax.com’, ... )
IF NOT EXISTS;
```

```
UPDATE USERS
SET email = ’jonathan@datastax.com’, ...
WHERE username = ’jbellis’
IF email = ’jbellis@datastax.com’;
```

Triggers

DATASTAX

```
CREATE TRIGGER <name> ON <table>
USING <classname>;
```

Trigger implementation

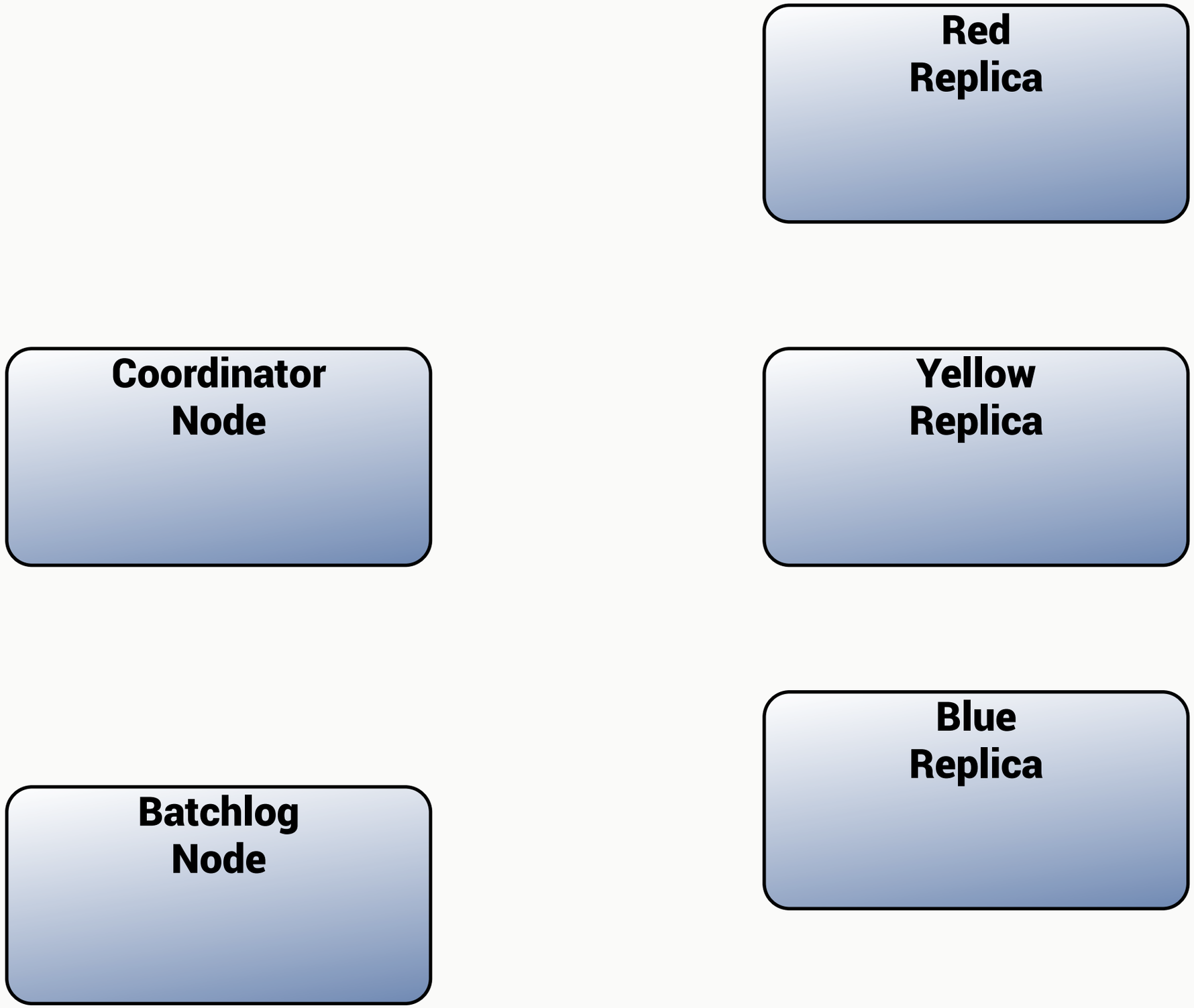
DATASTAX

```
class MyTrigger implements ITrigger
{
    public Collection<RowMutation> augment
    (ByteBuffer key, ColumnFamily update)
    {
        ...
    }
}
```

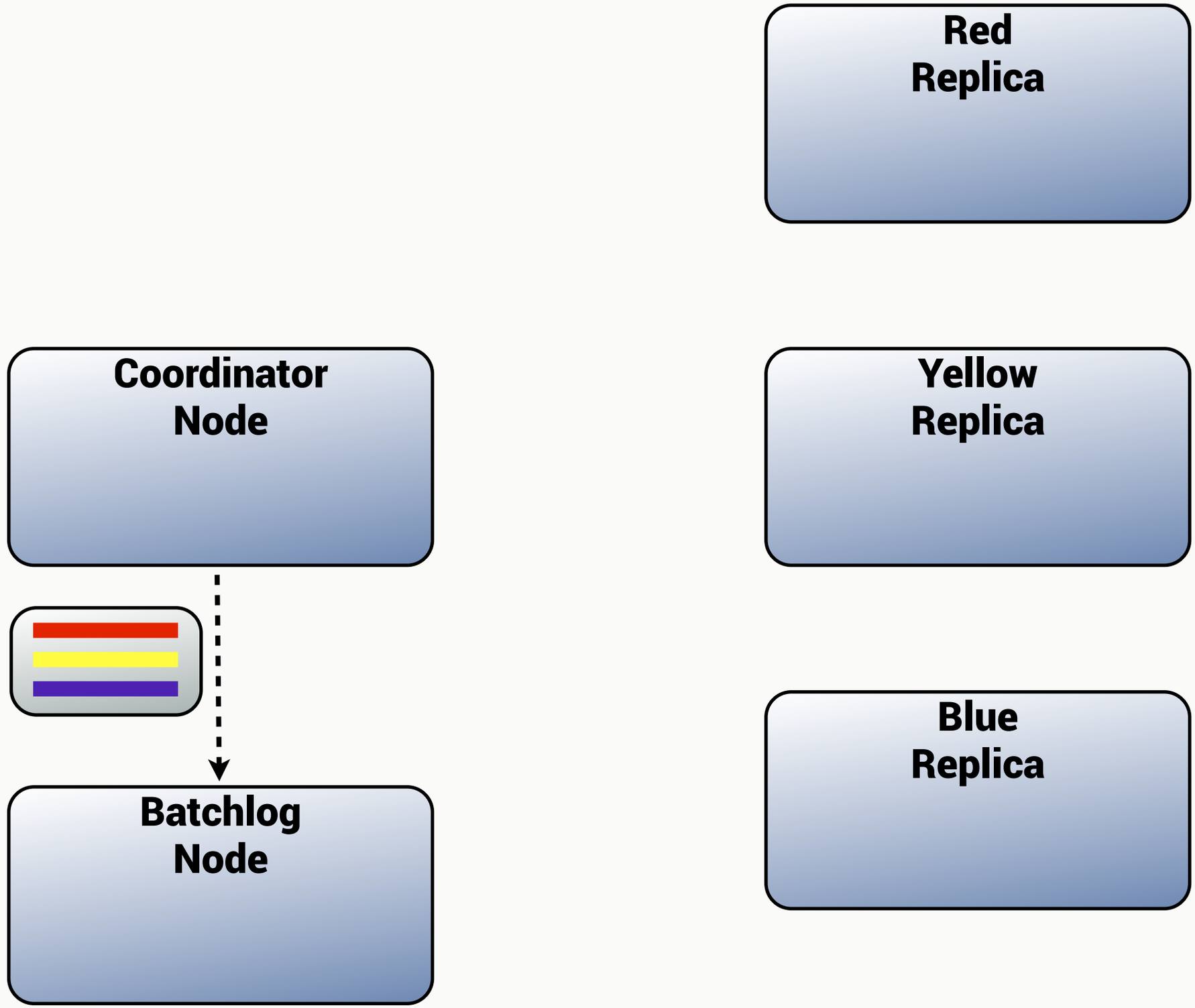
Atomicity?

DATASTAX

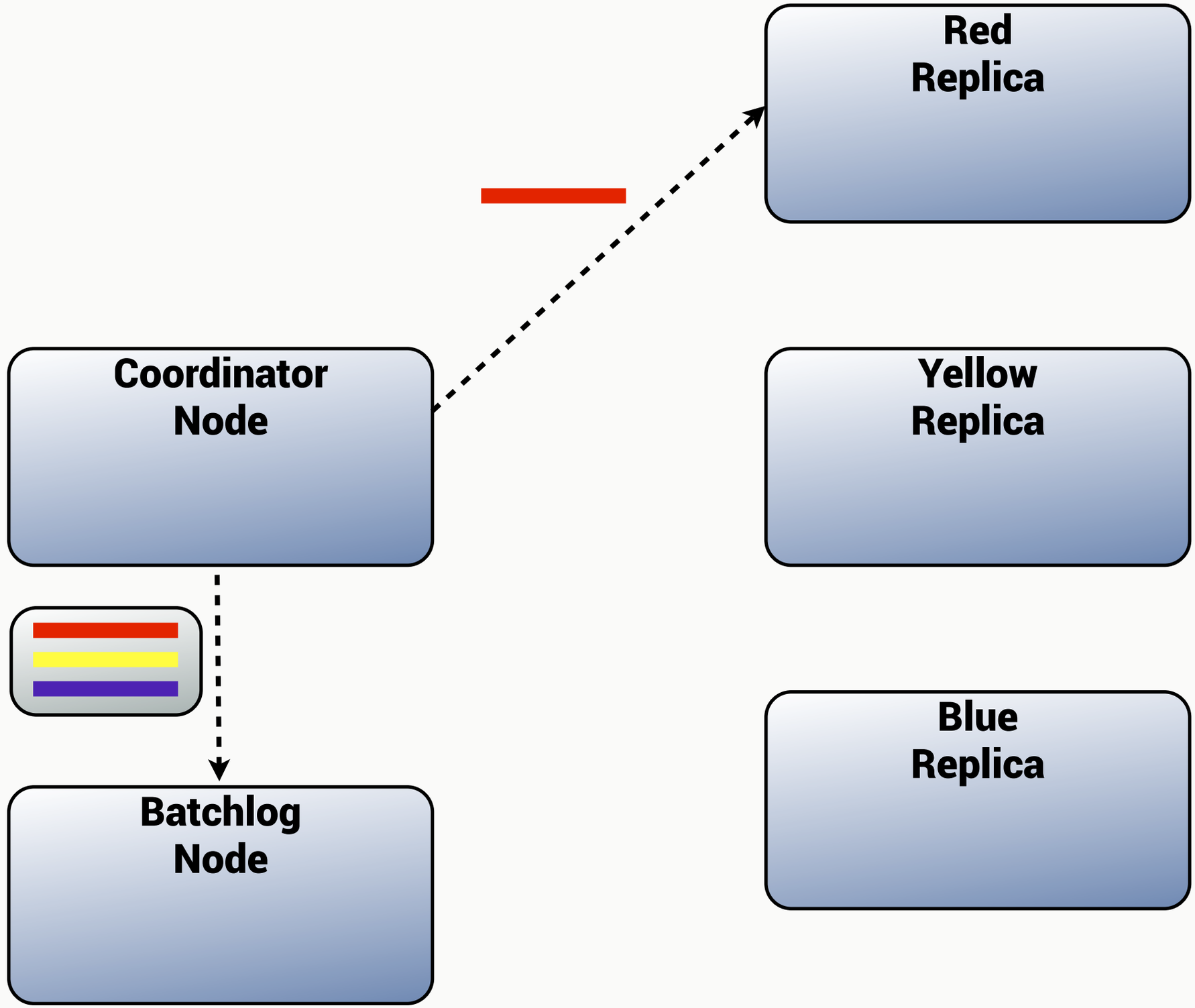
Batches


Red
Replica
Coordinator
Node
Yellow
Replica
Blue
Replica
Batchlog
Node

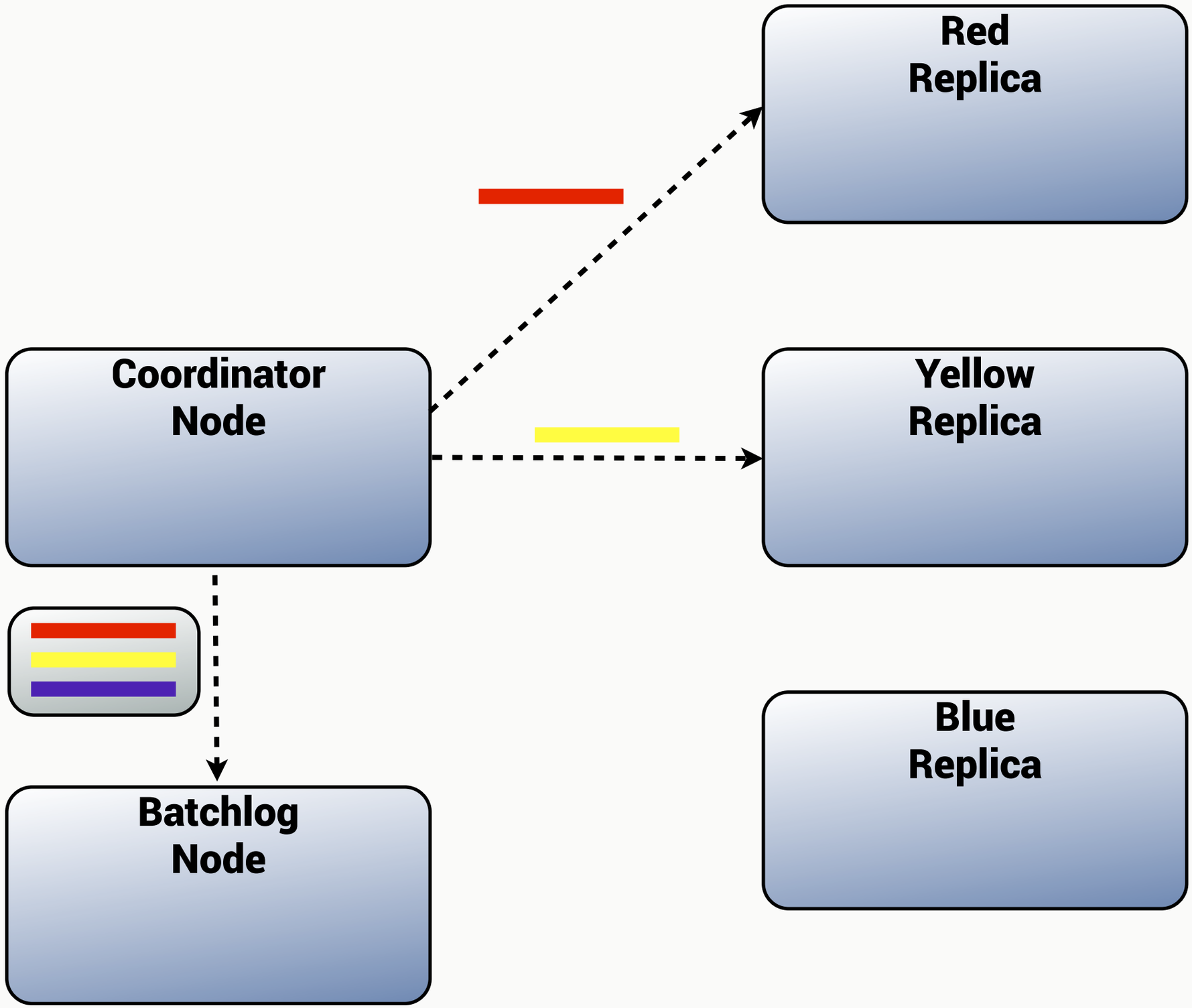
Batches


Red
Replica
Coordinator
Node
Yellow
Replica
Batchlog
Node
Blue
Replica

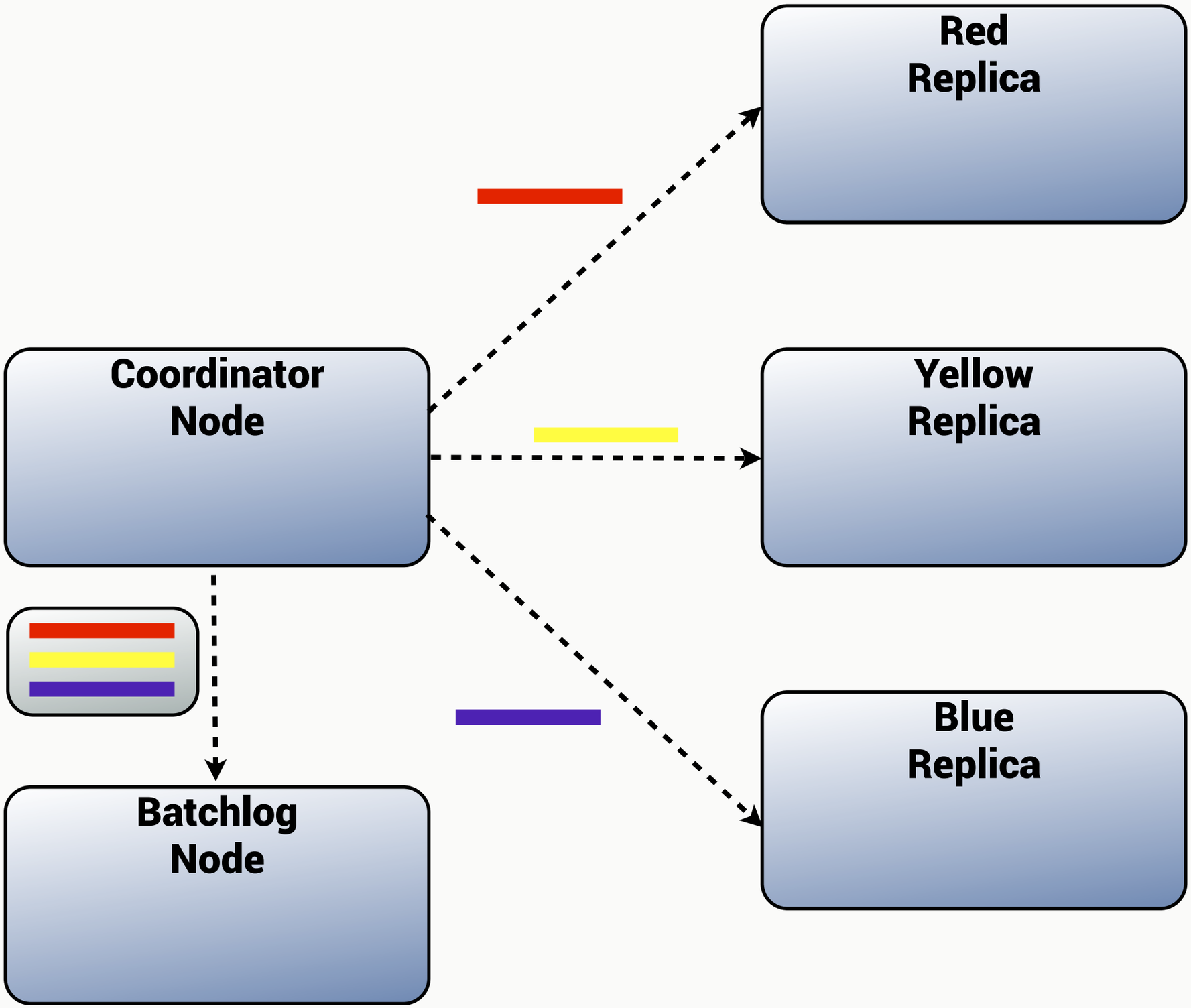
Batches


Red
Replica
Coordinator
Node
Yellow
Replica
Batchlog
Node
Blue
Replica

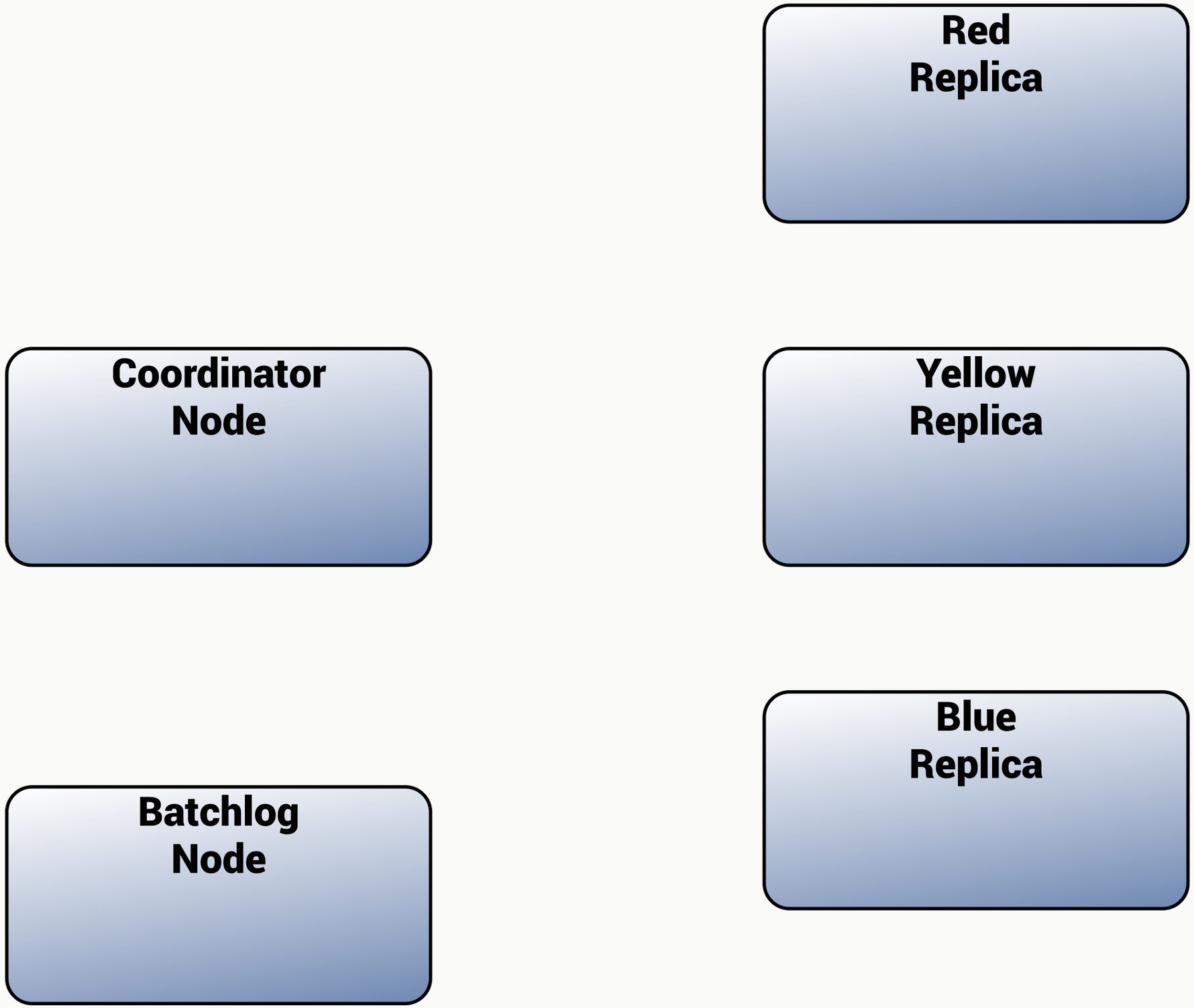
Batches


Red
Replica
Coordinator
Node
Yellow
Replica
Batchlog
Node
Blue
Replica

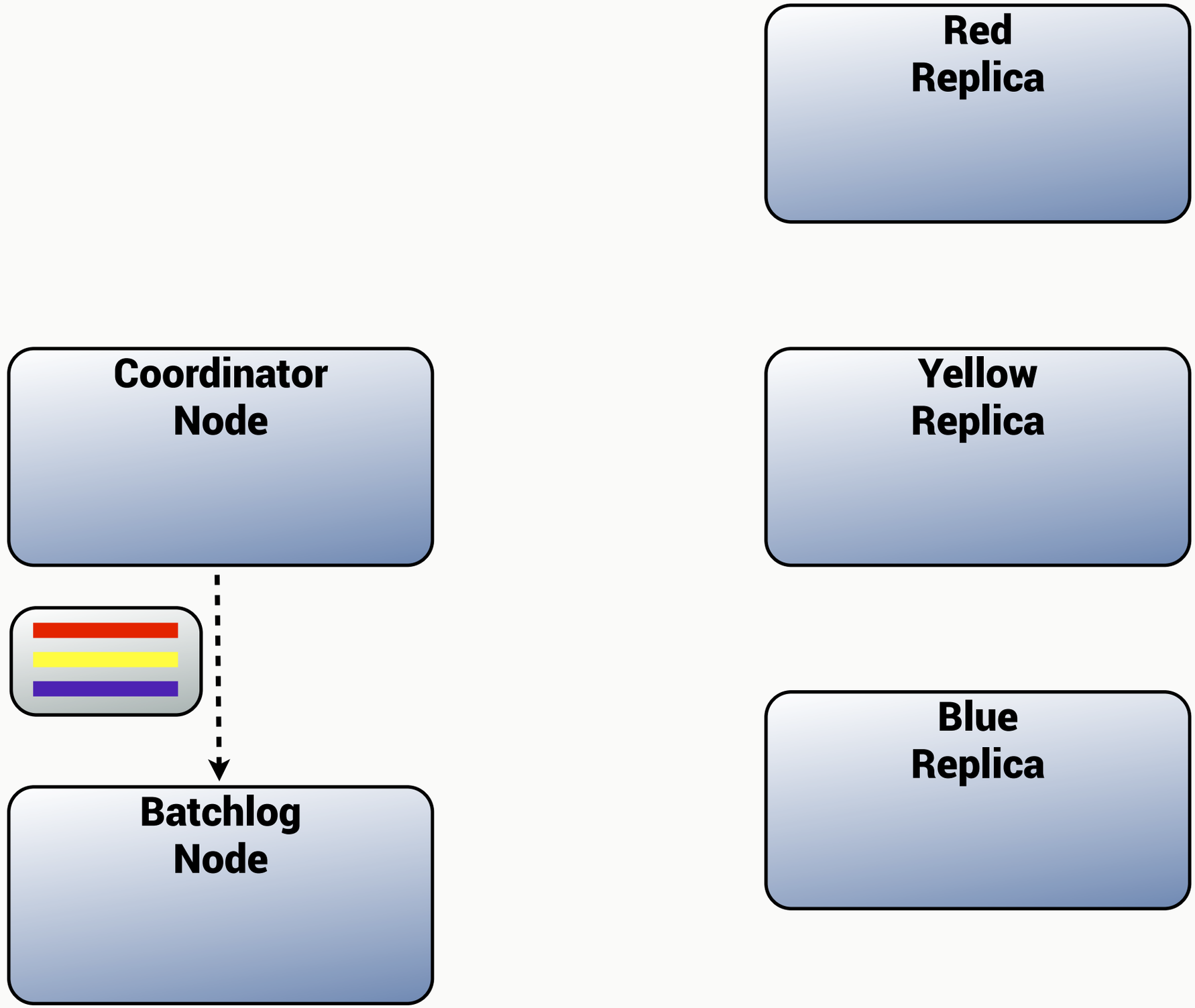
Batches


Red
Replica
Coordinator
Node
Yellow
Replica
Blue
Replica
Batchlog
Node

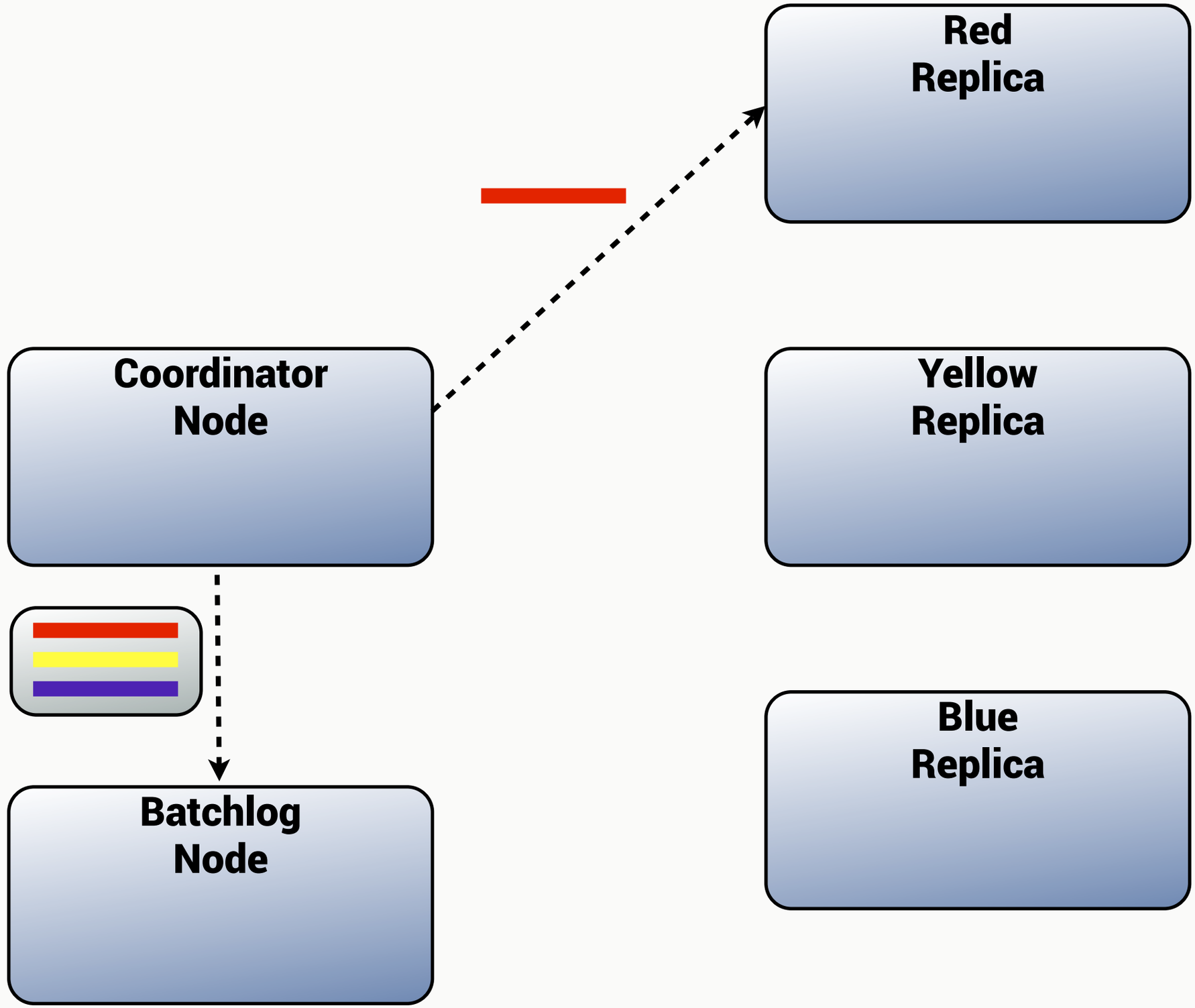
Batches


Red
Replica
Coordinator
Node
Yellow
Replica
Blue
Replica
Batchlog
Node

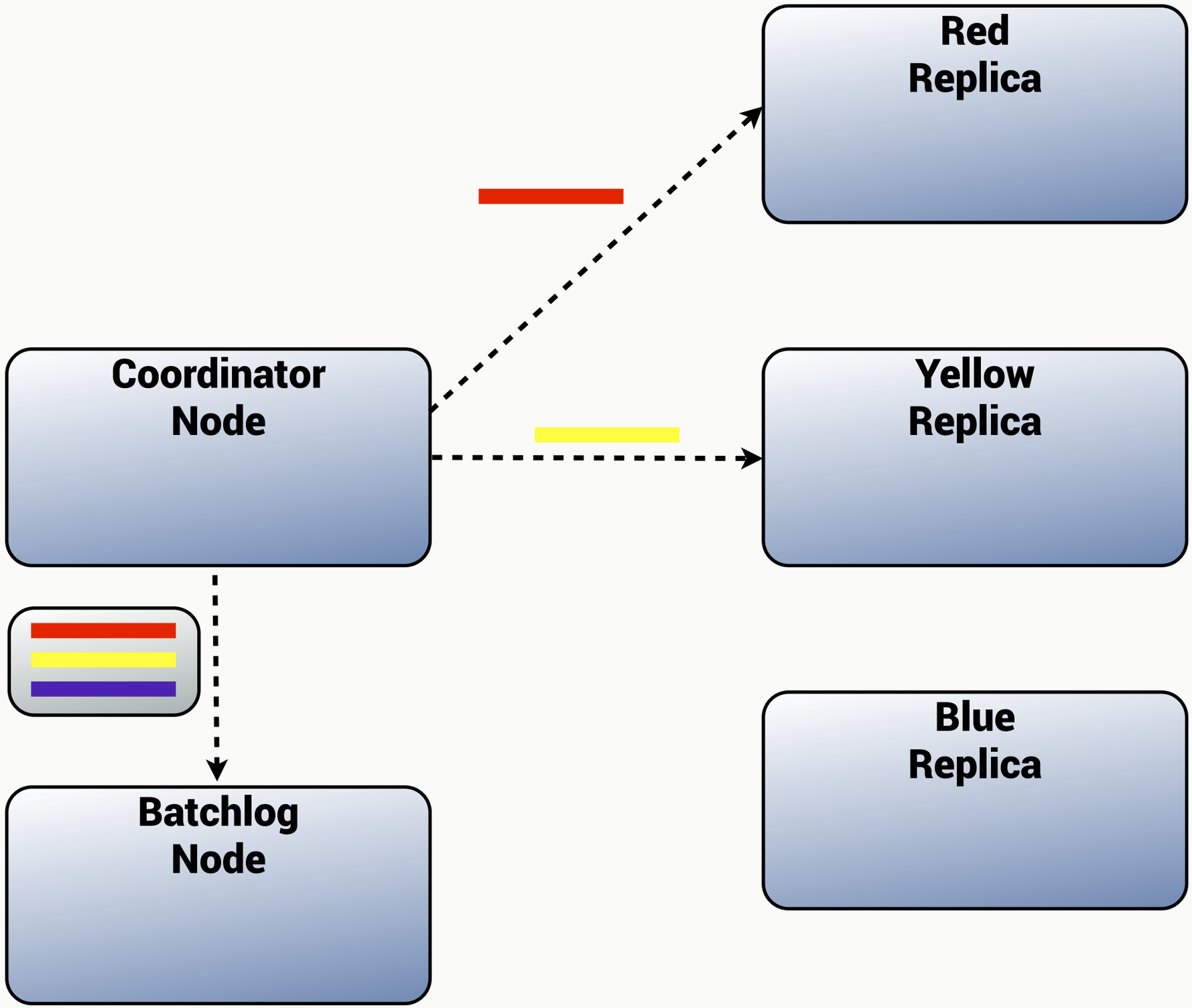
Batches


Red
Replica
Coordinator
Node
Yellow
Replica
Batchlog
Node
Blue
Replica

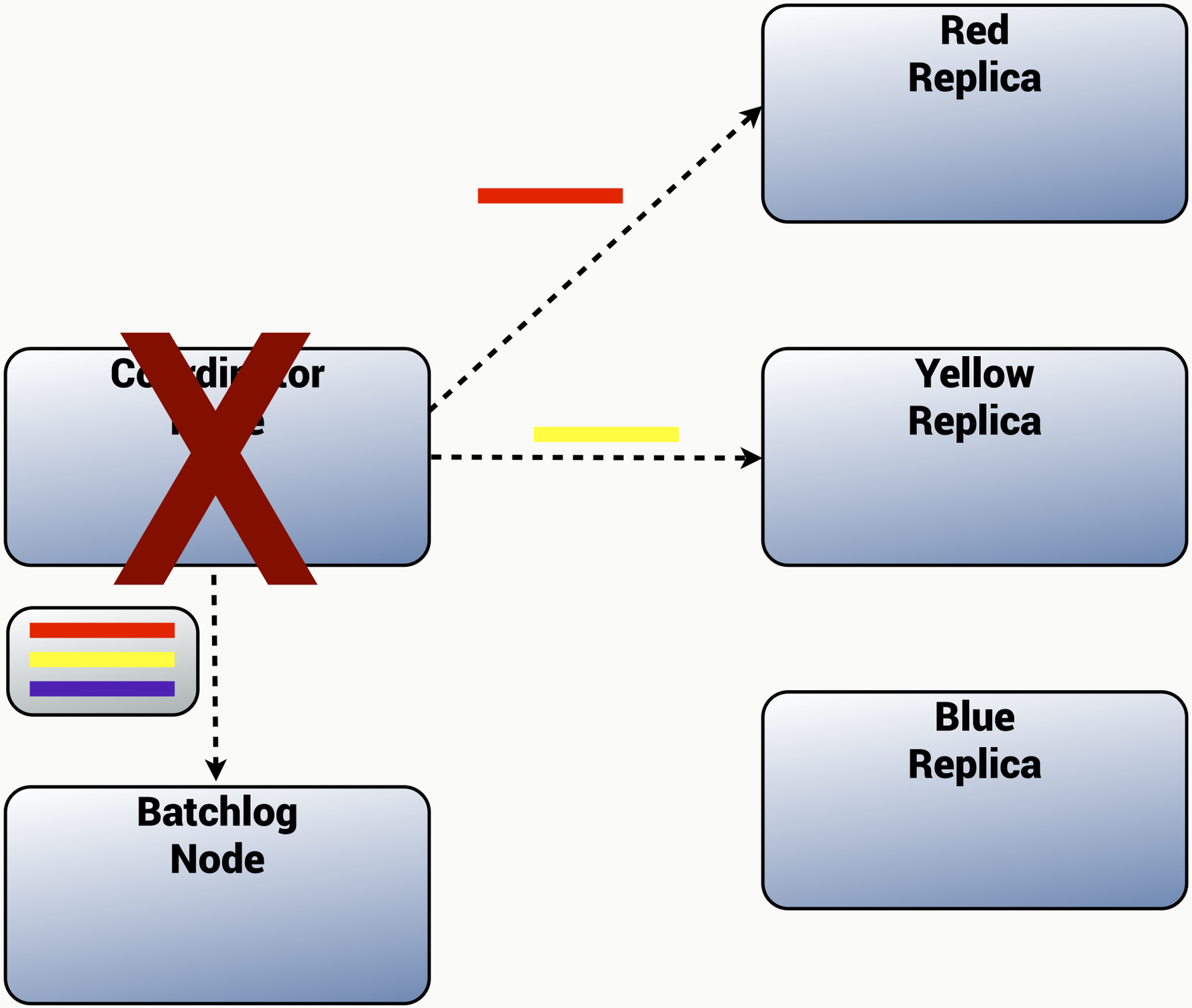
Batches


Red
Replica
Coordinator
Node
Yellow
Replica
Batchlog
Node
Blue
Replica

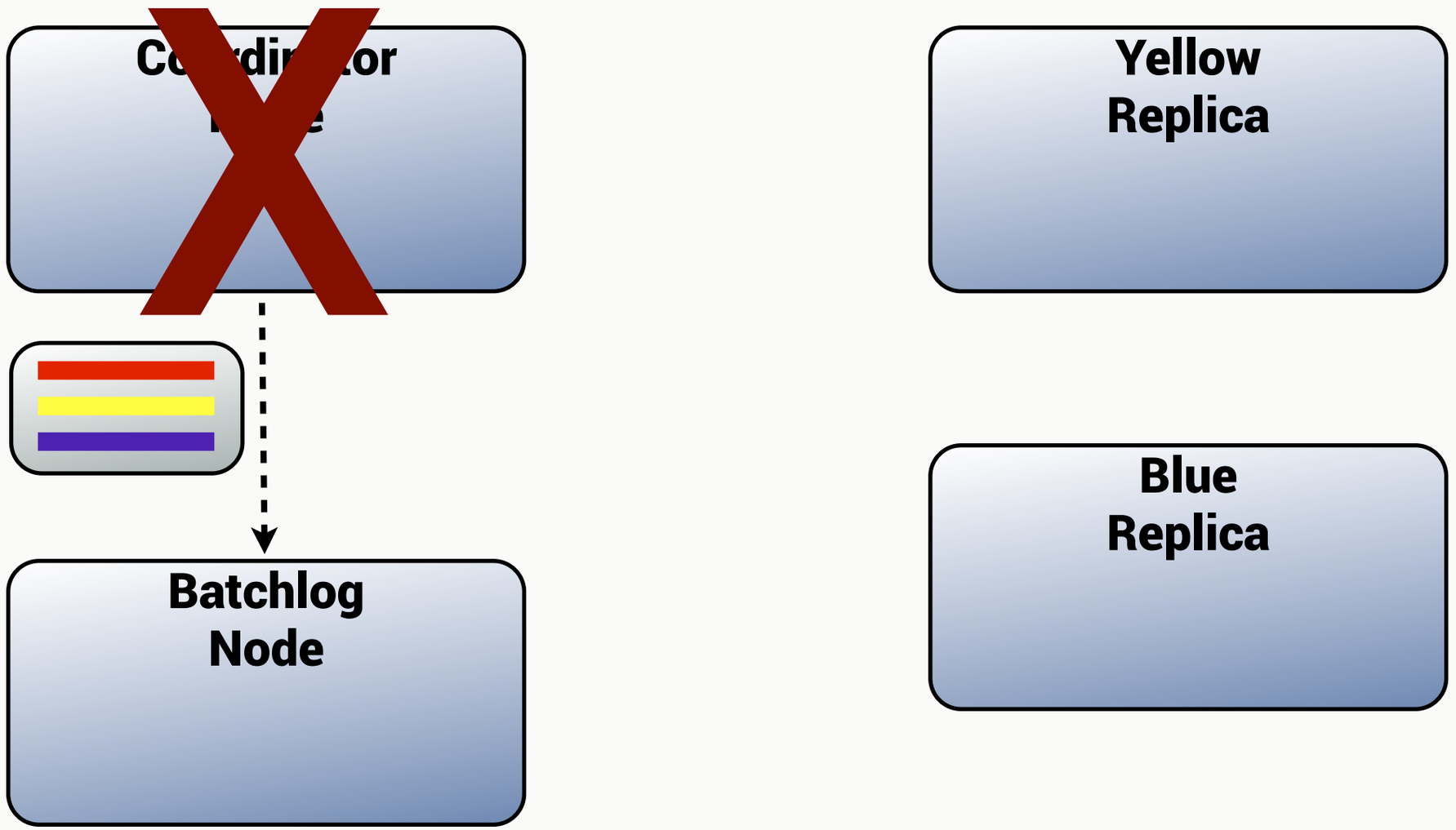
Batches


Red
Replica
Coordinator
Node
Yellow
Replica
Batchlog
Node
Blue
Replica

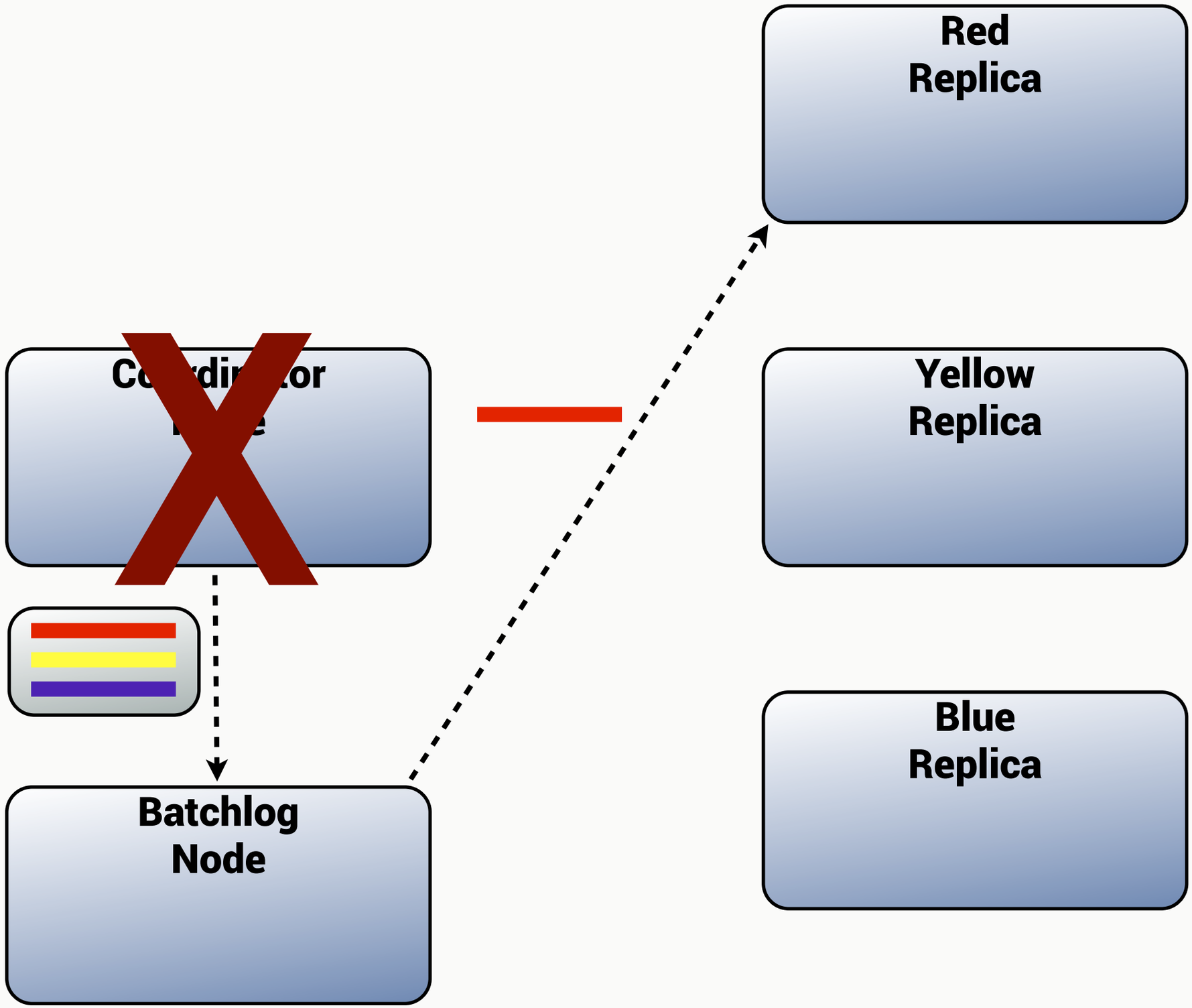
Batches



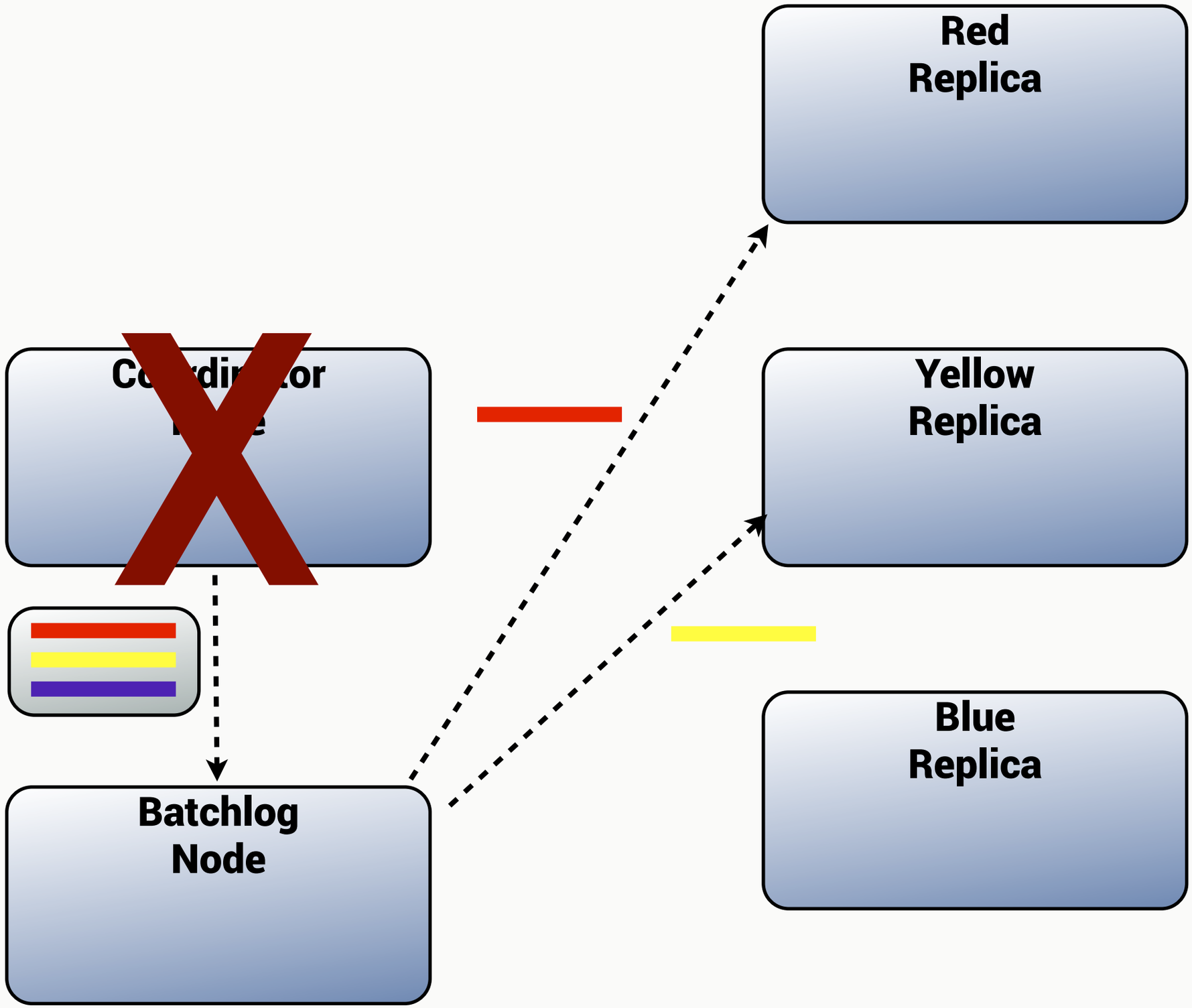
Batches



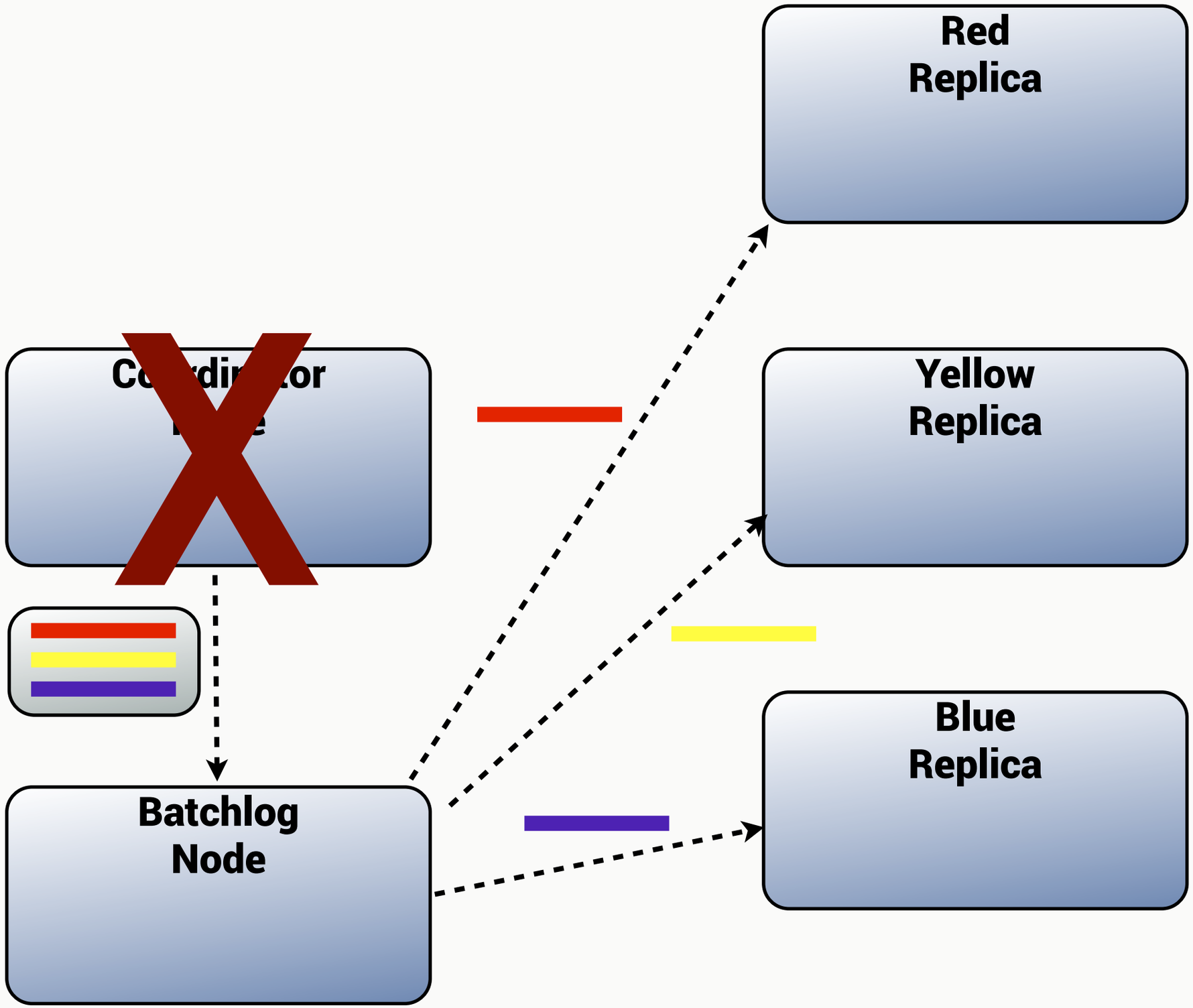
Batches



Batches



Batches

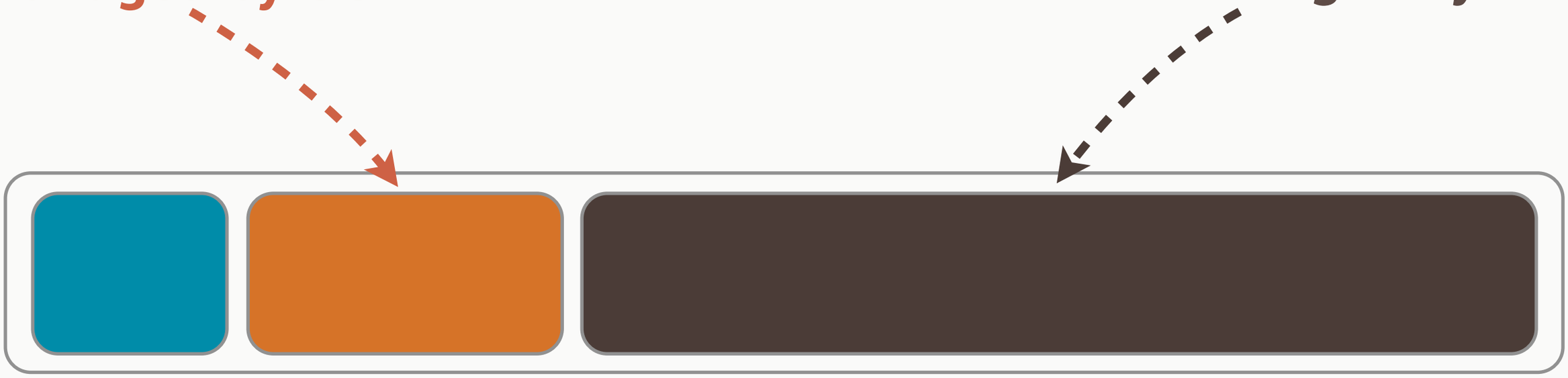


On-Heap/Off-Heap

DATASTAX

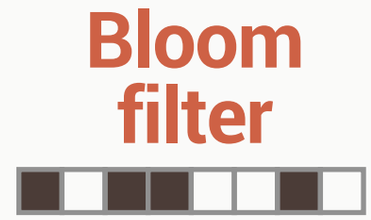
On-Heap
Managed by GC

Off-Heap
Not managed by GC



Java Process

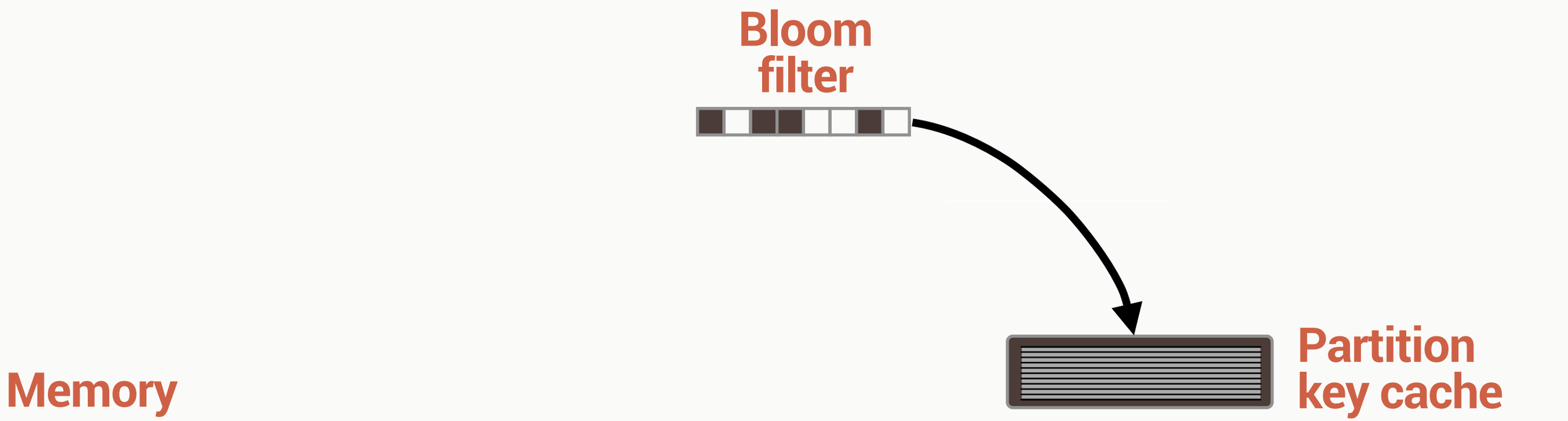
Read path (per sstable)



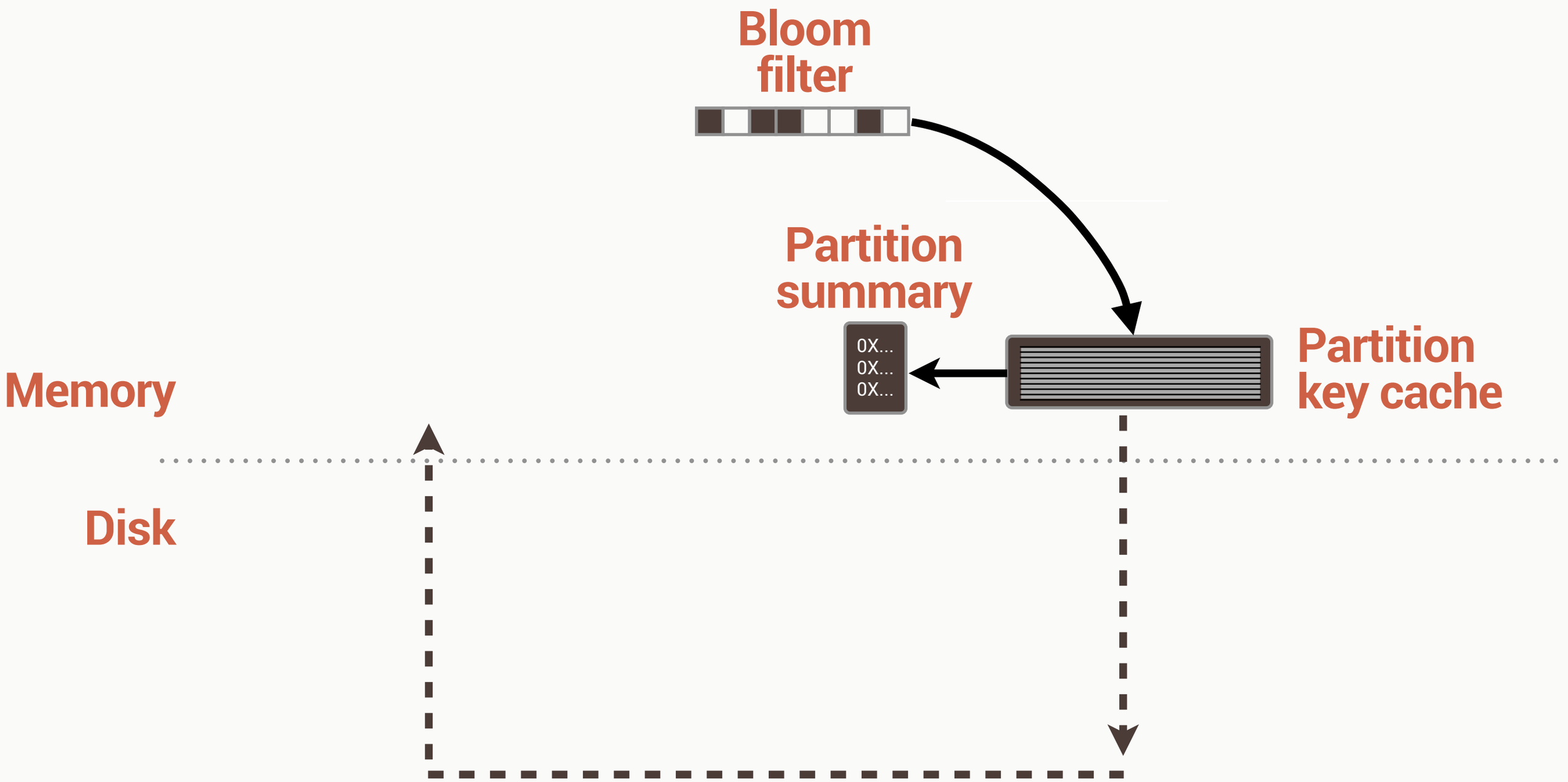
Memory

Disk

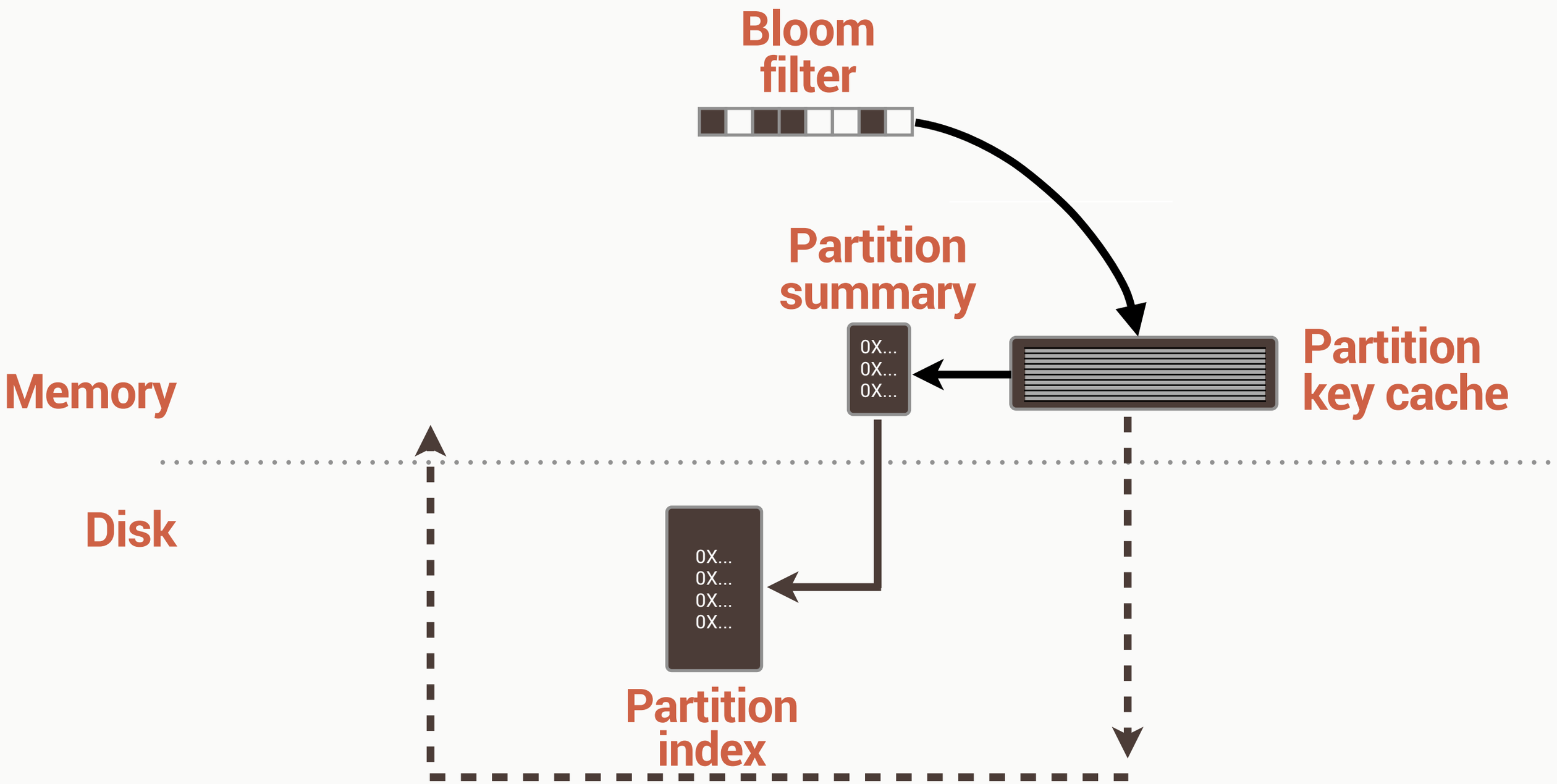
Read path (per sstable)



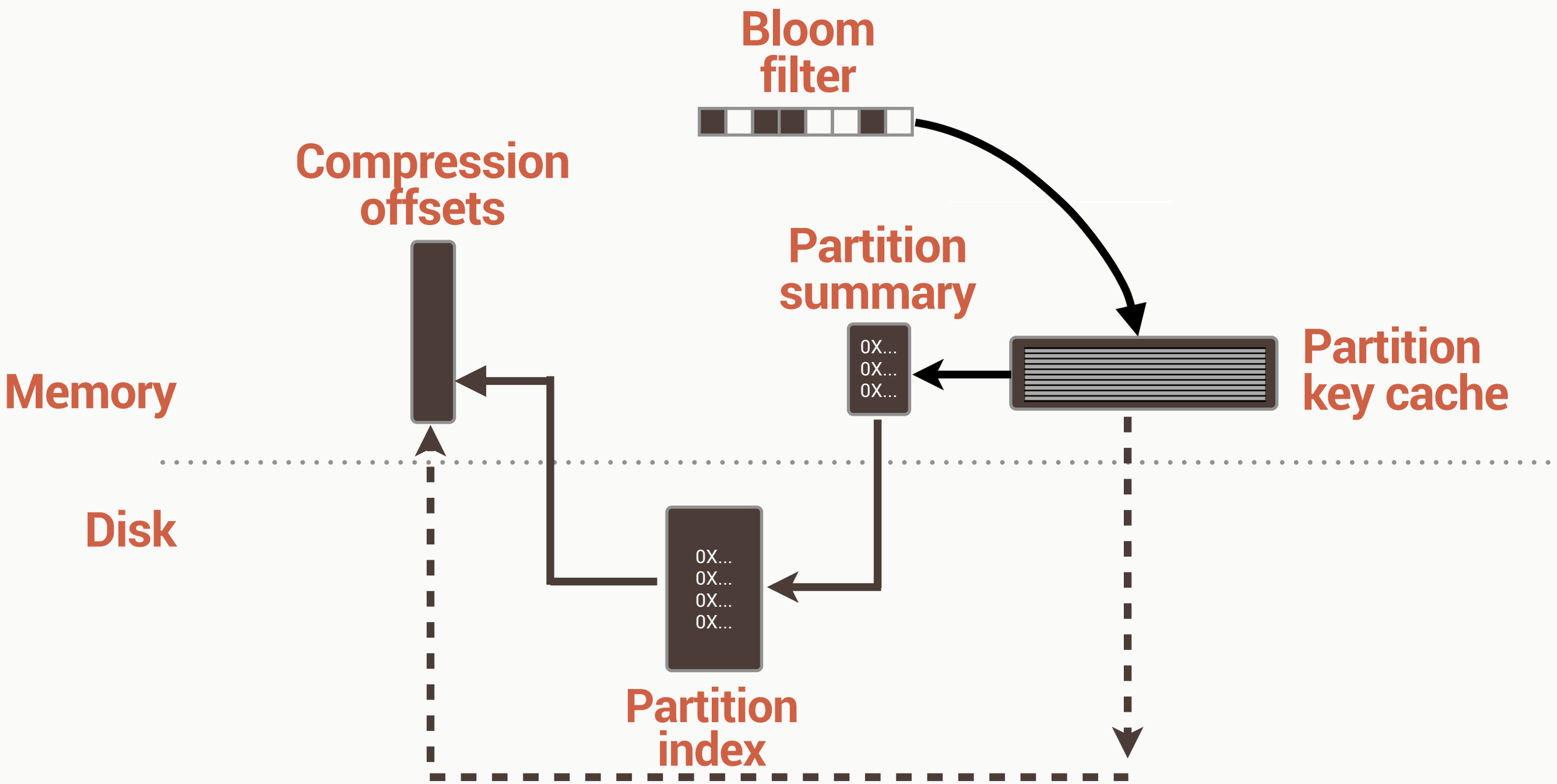
Read path (per sstable)



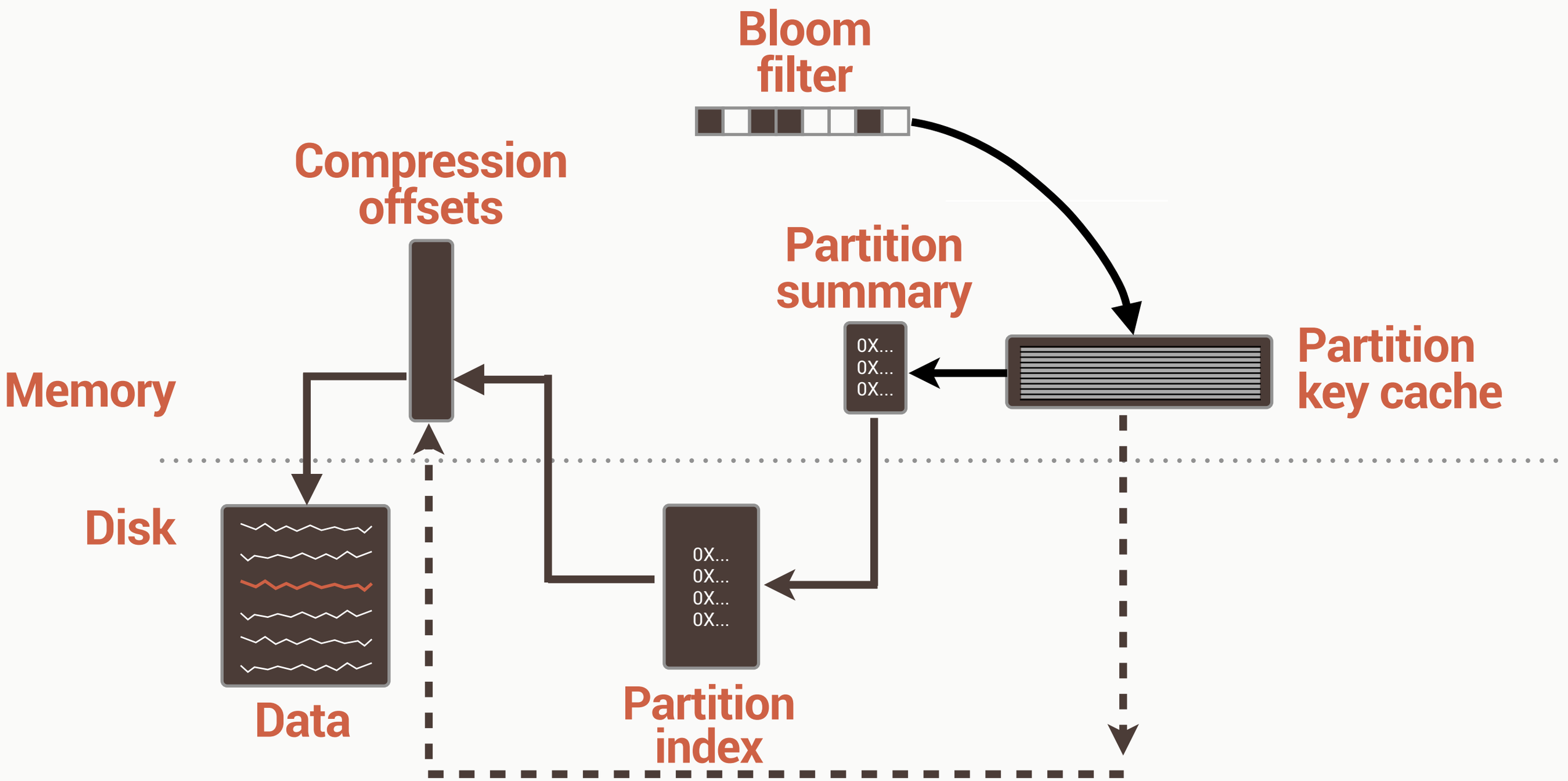
Read path (per sstable)



Read path (per sstable)



Read path (per sstable)

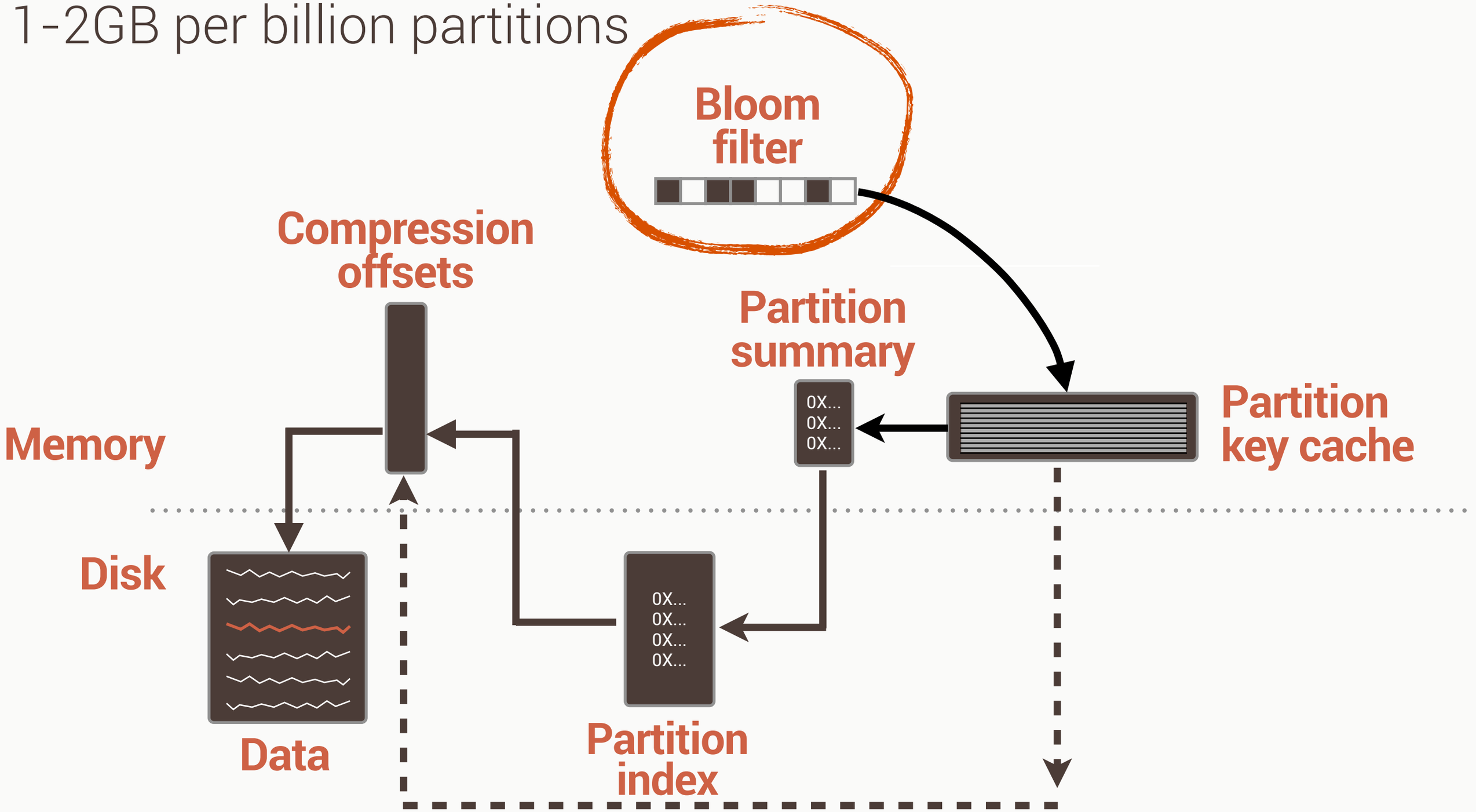


Off heap in 2.0

DATASTAX

Partition key bloom filter

1-2GB per billion partitions

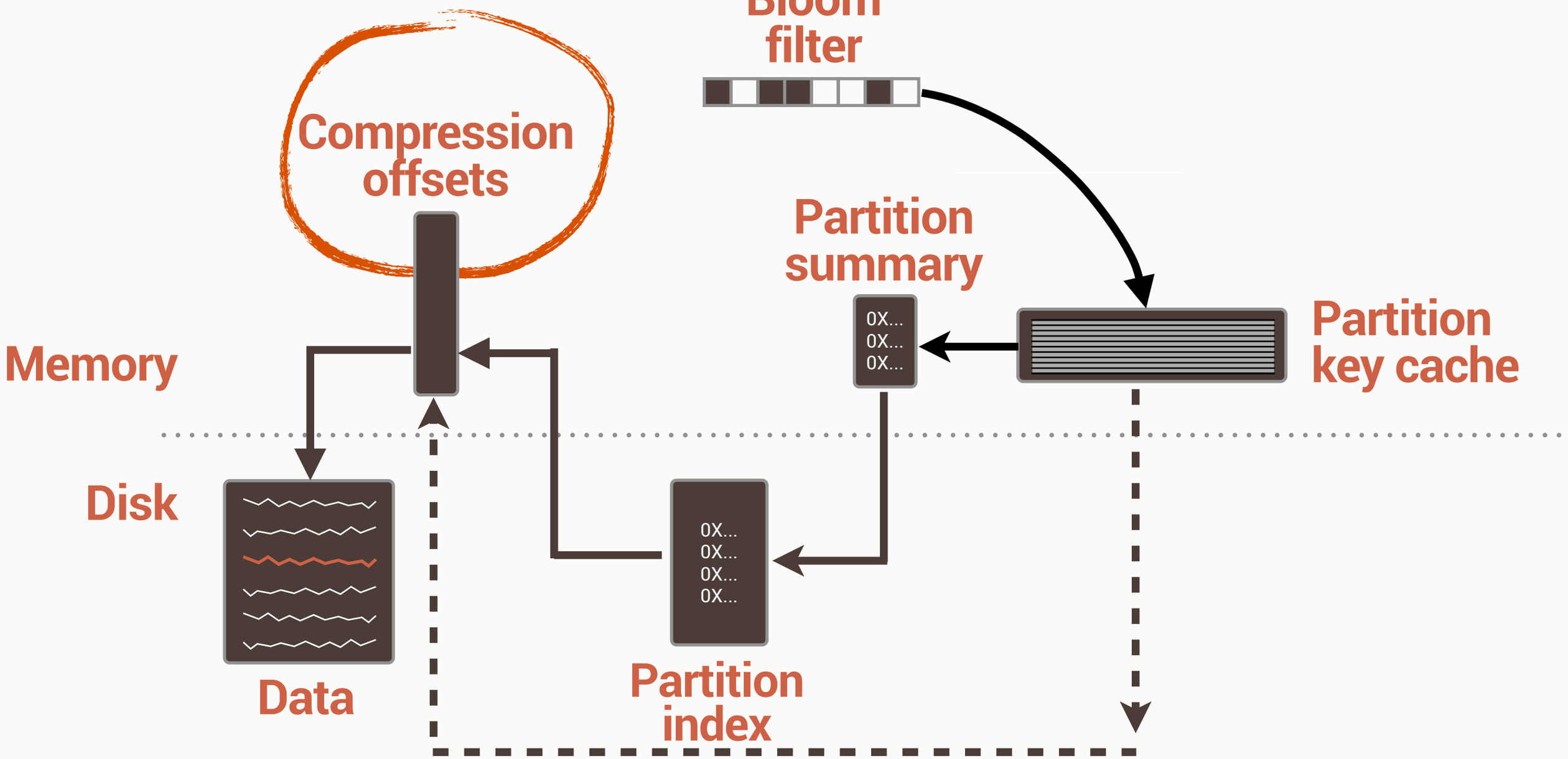


Off heap in 2.0

DATASTAX

Compression metadata

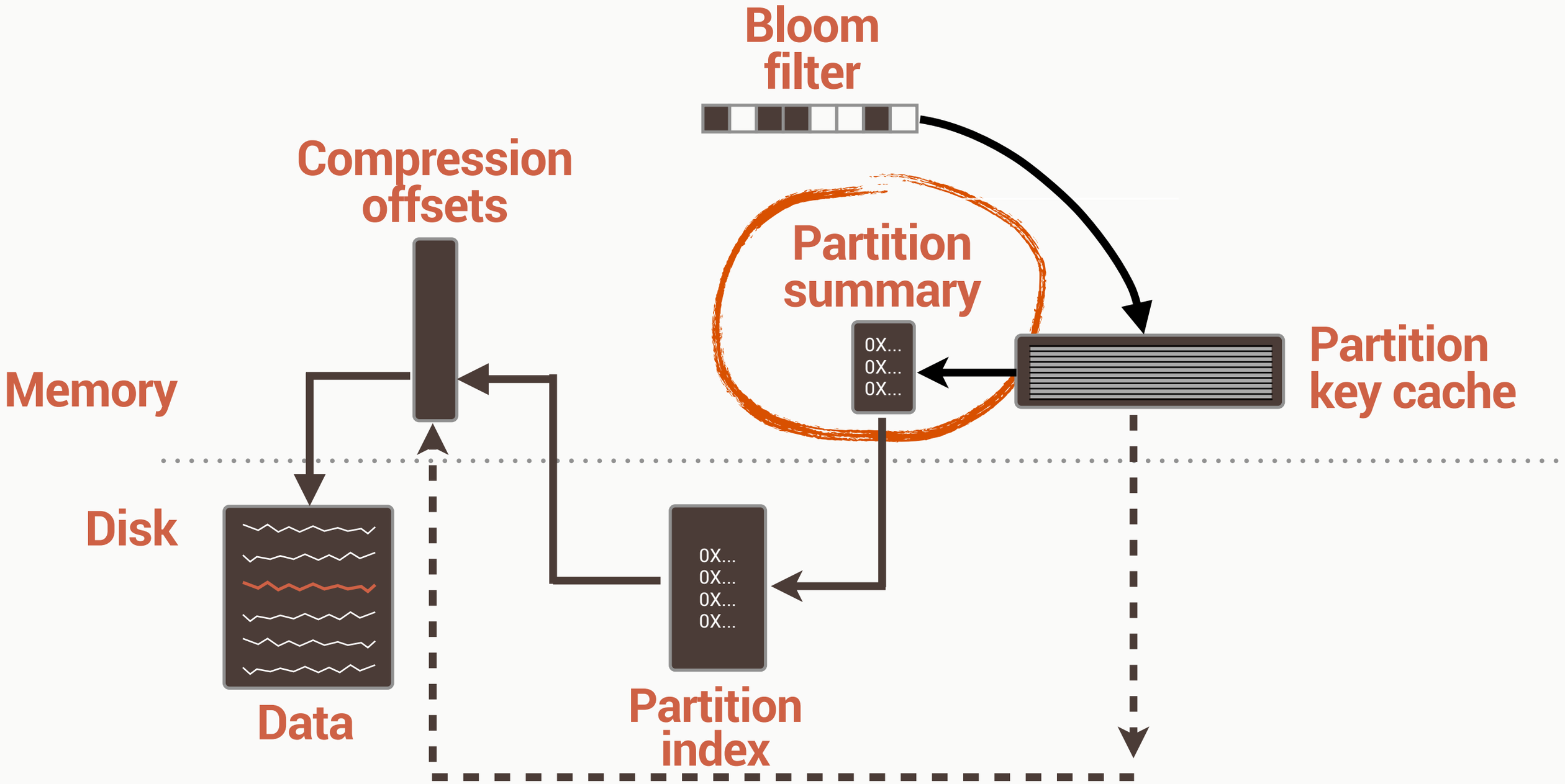
~1-3GB per TB compressed



Off heap in 2.0

DATASTAX

Partition index summary
(depends on rows per partition)

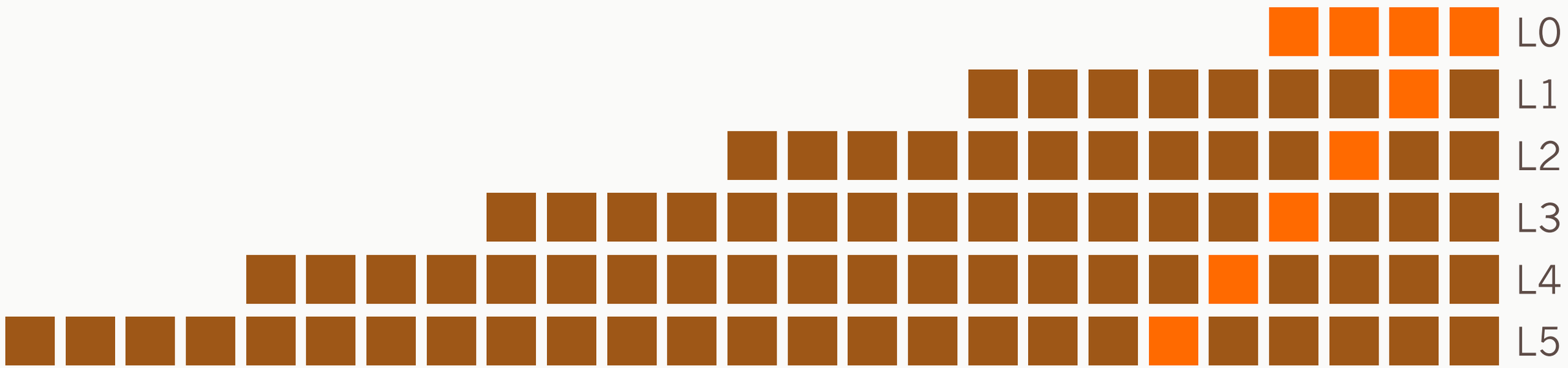


Compaction

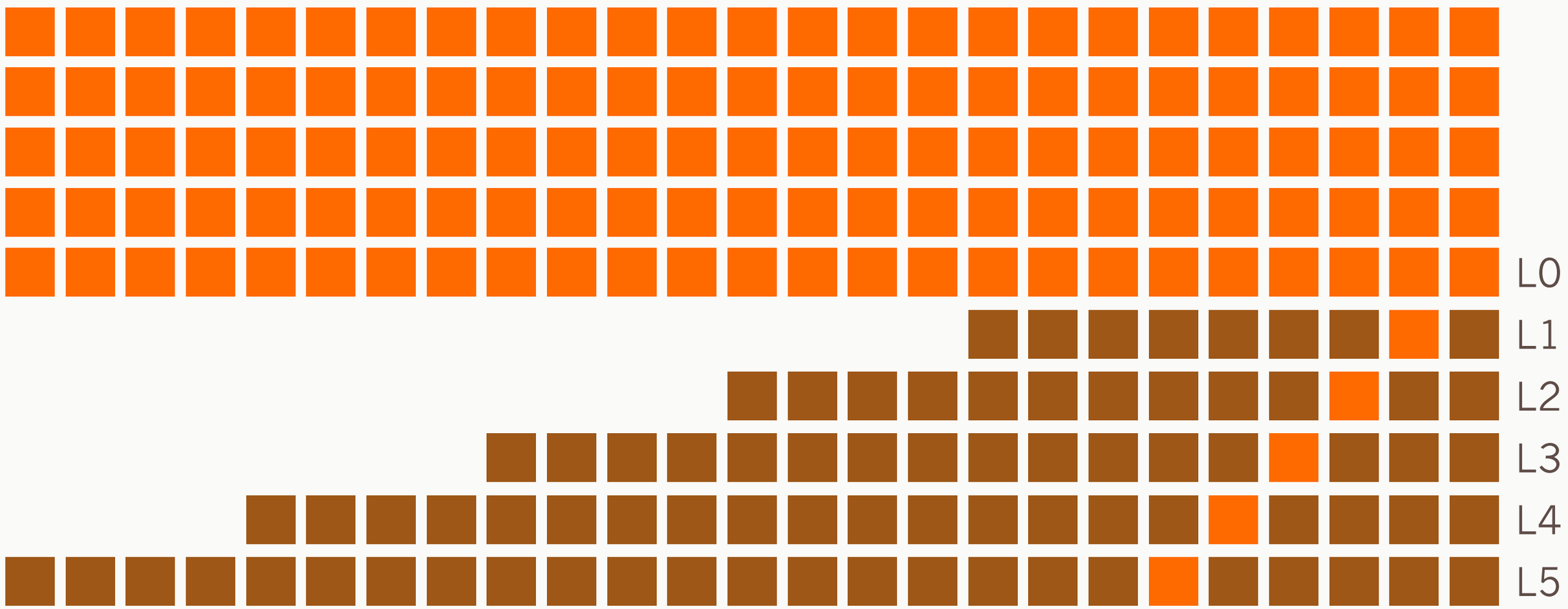
- Size-tiered
- Leveled
- Others?

Size-tiered compaction

Leveled compaction

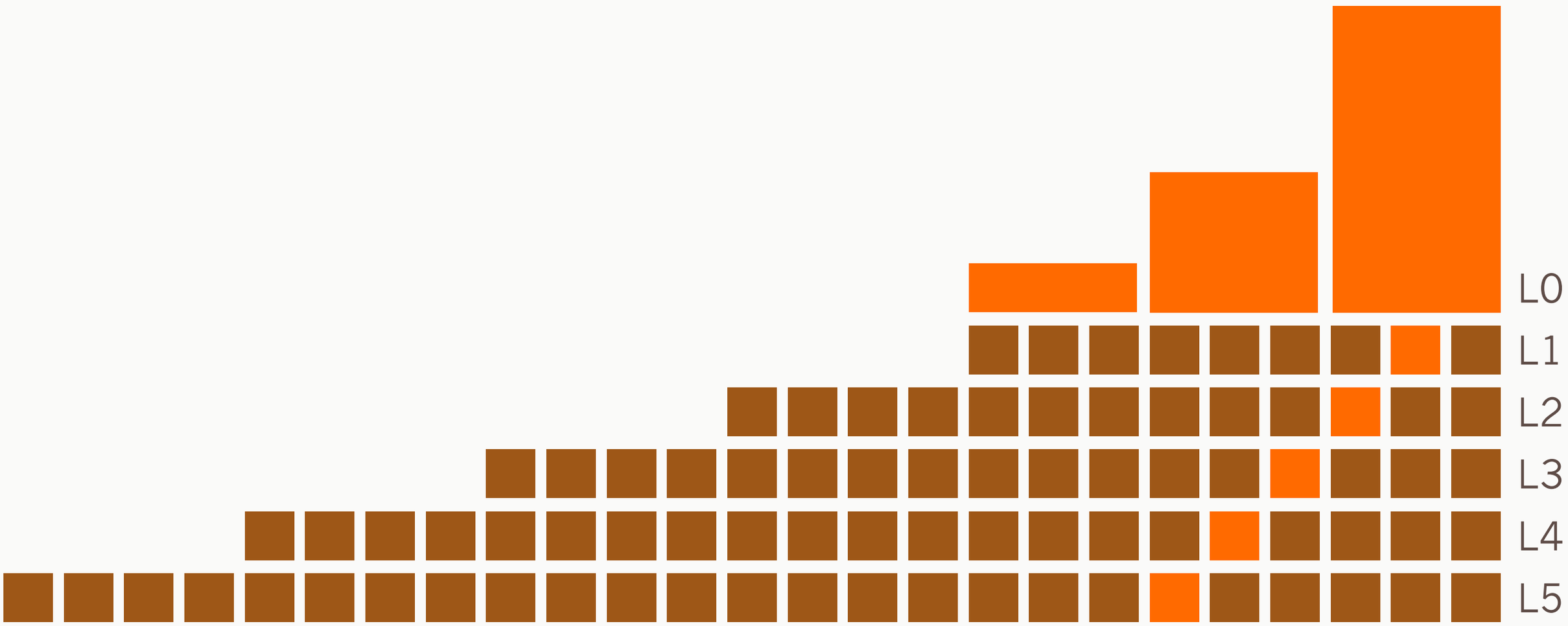


Sad leveled compaction



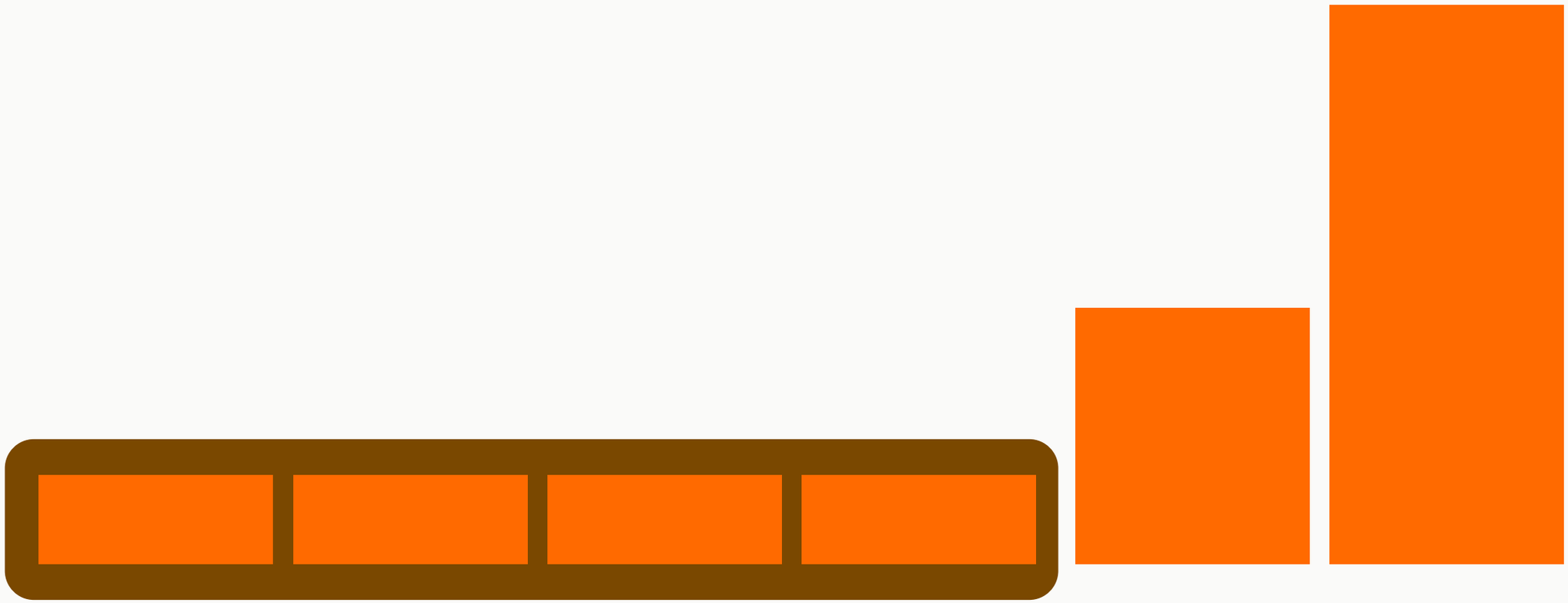
STCS in L0

DATASTAX

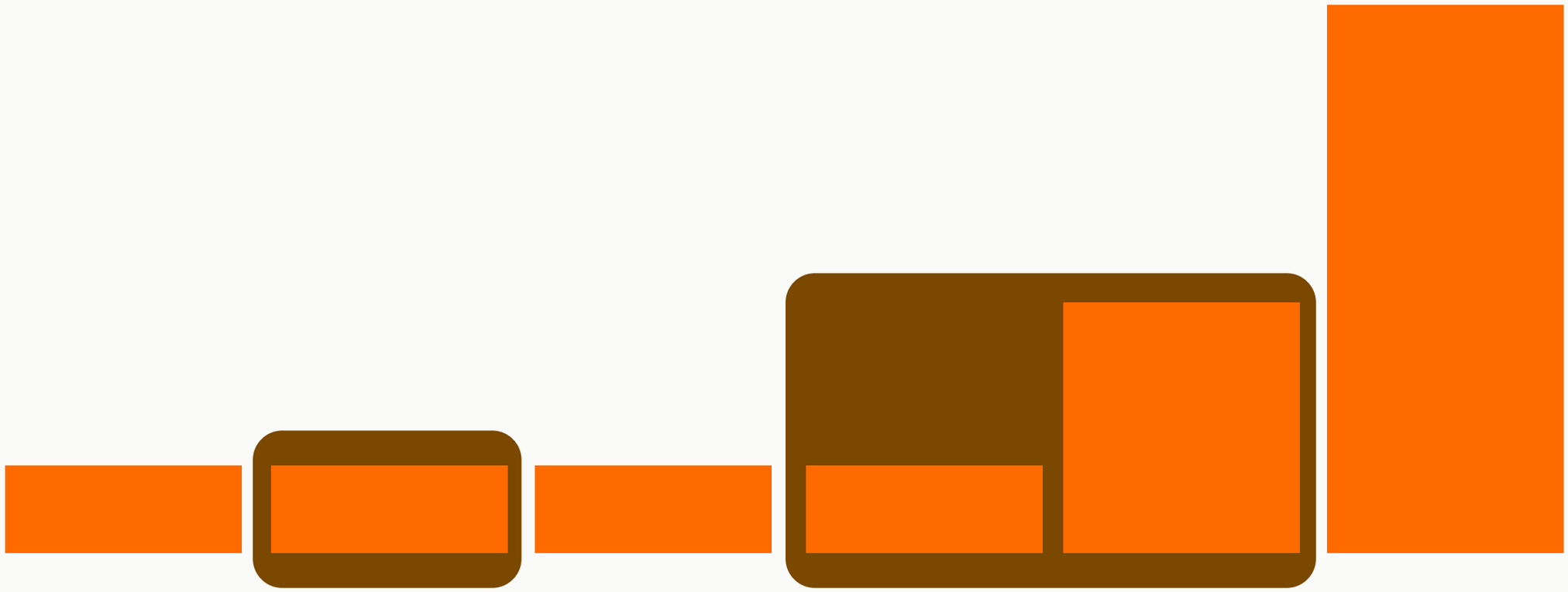


HLL and compaction

HLL and compaction



HLL and compaction



Data-aware compaction?

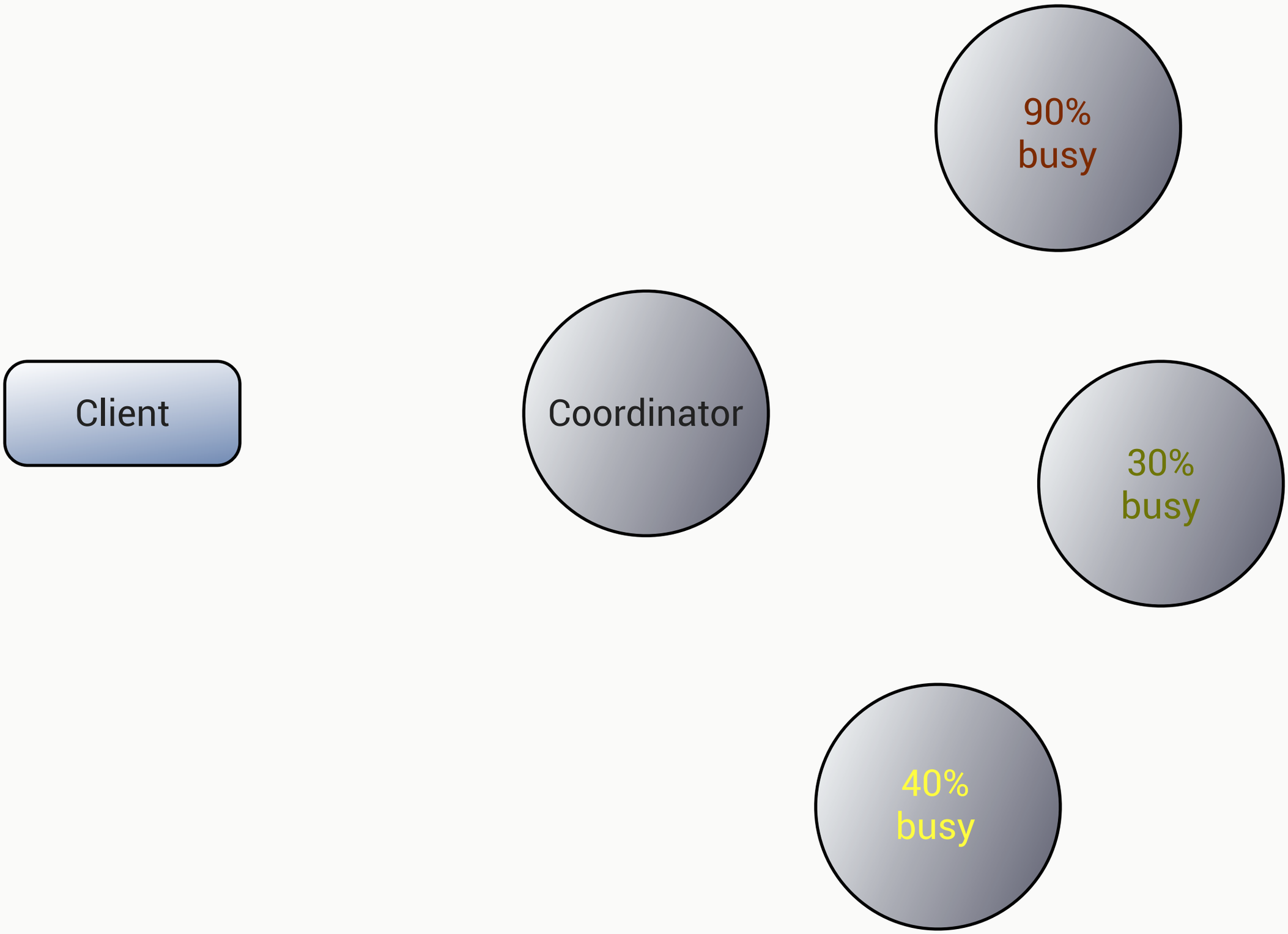
- Append-only workloads
  - No compaction necessary in trivial case; still needed for clustered scans
- Append-mostly workloads?
  - Bounded window for out-of-order updates

Rapid Read Protection

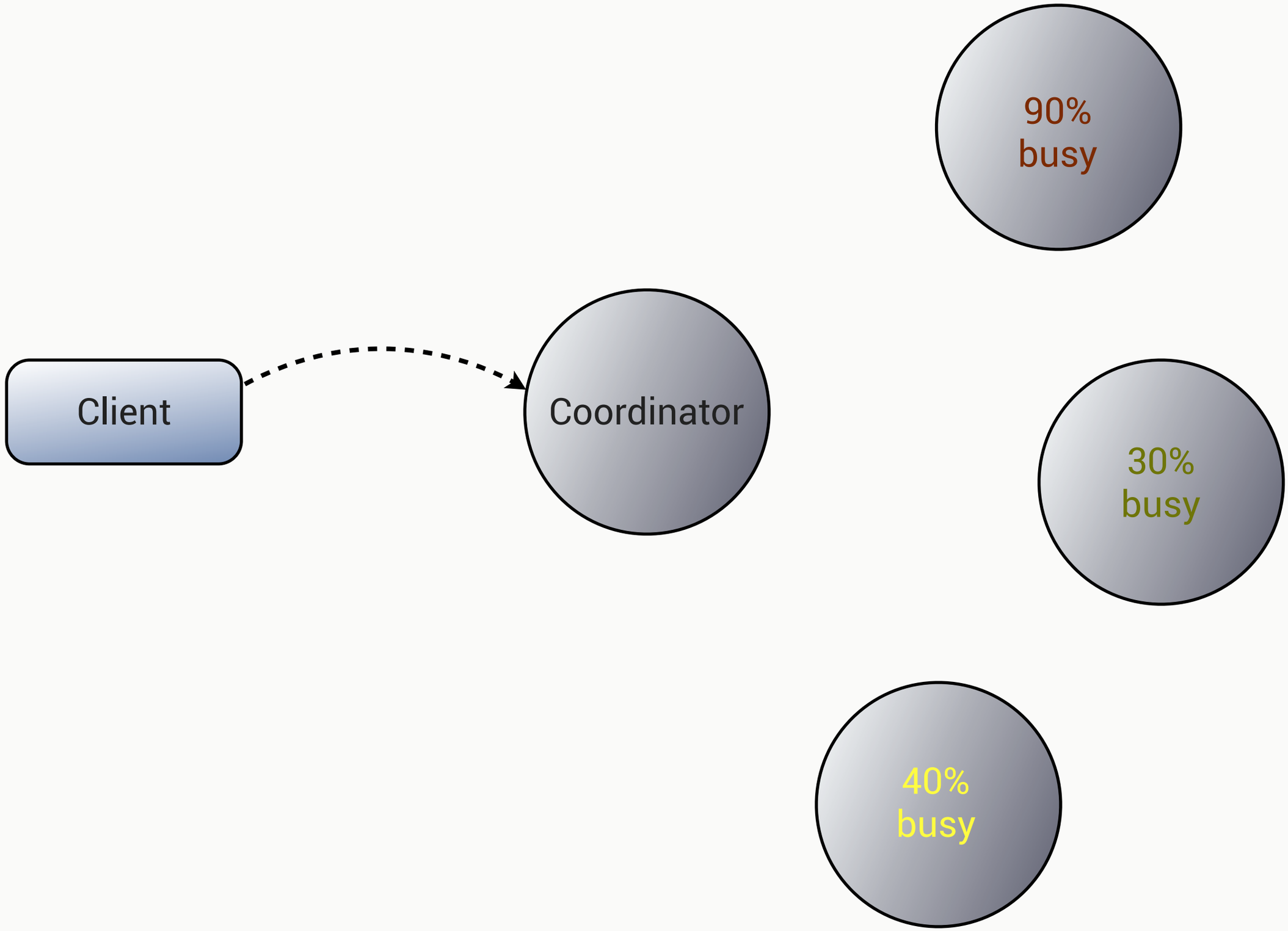
DATASTAX

NONE

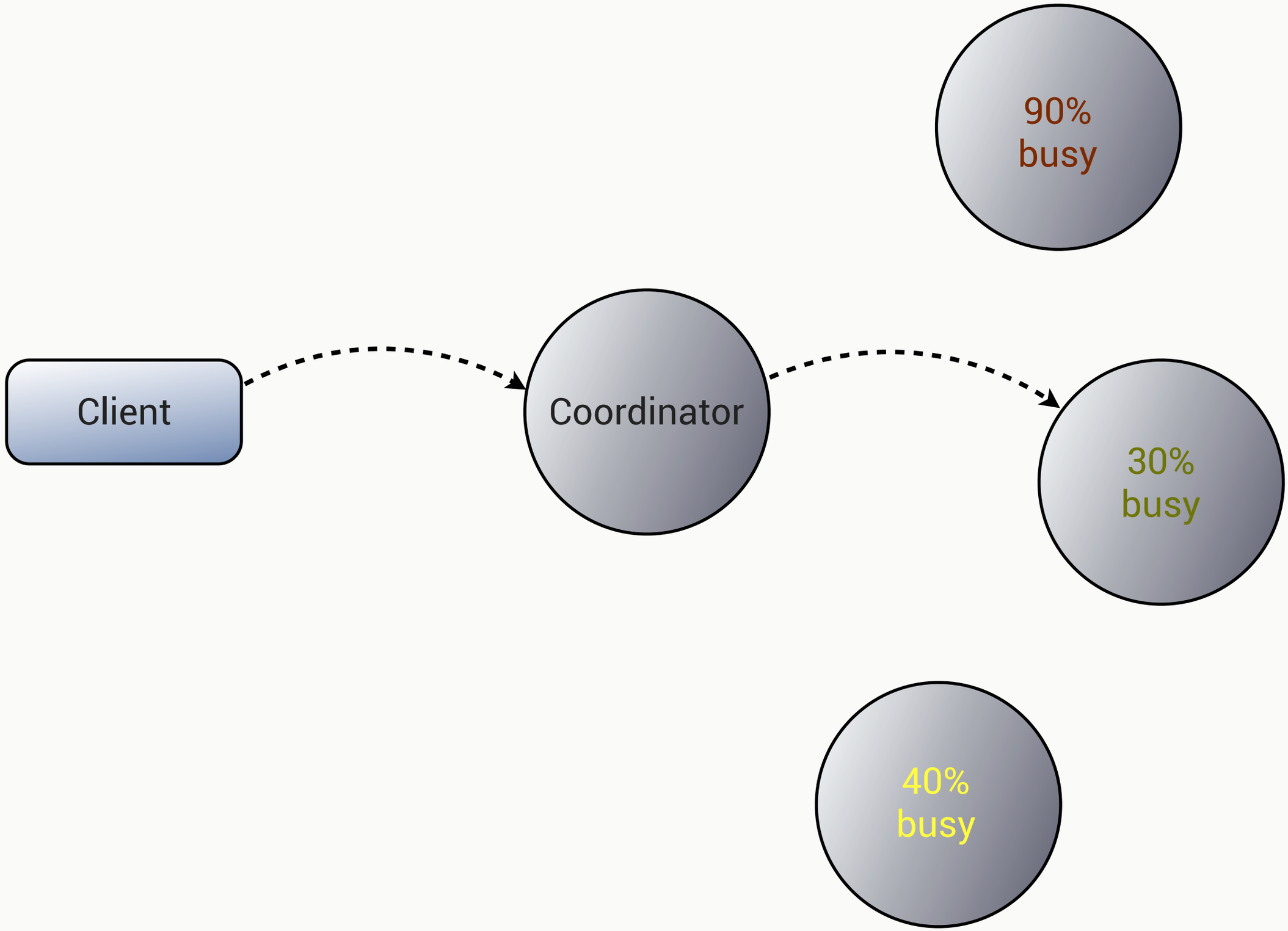
Typical reads



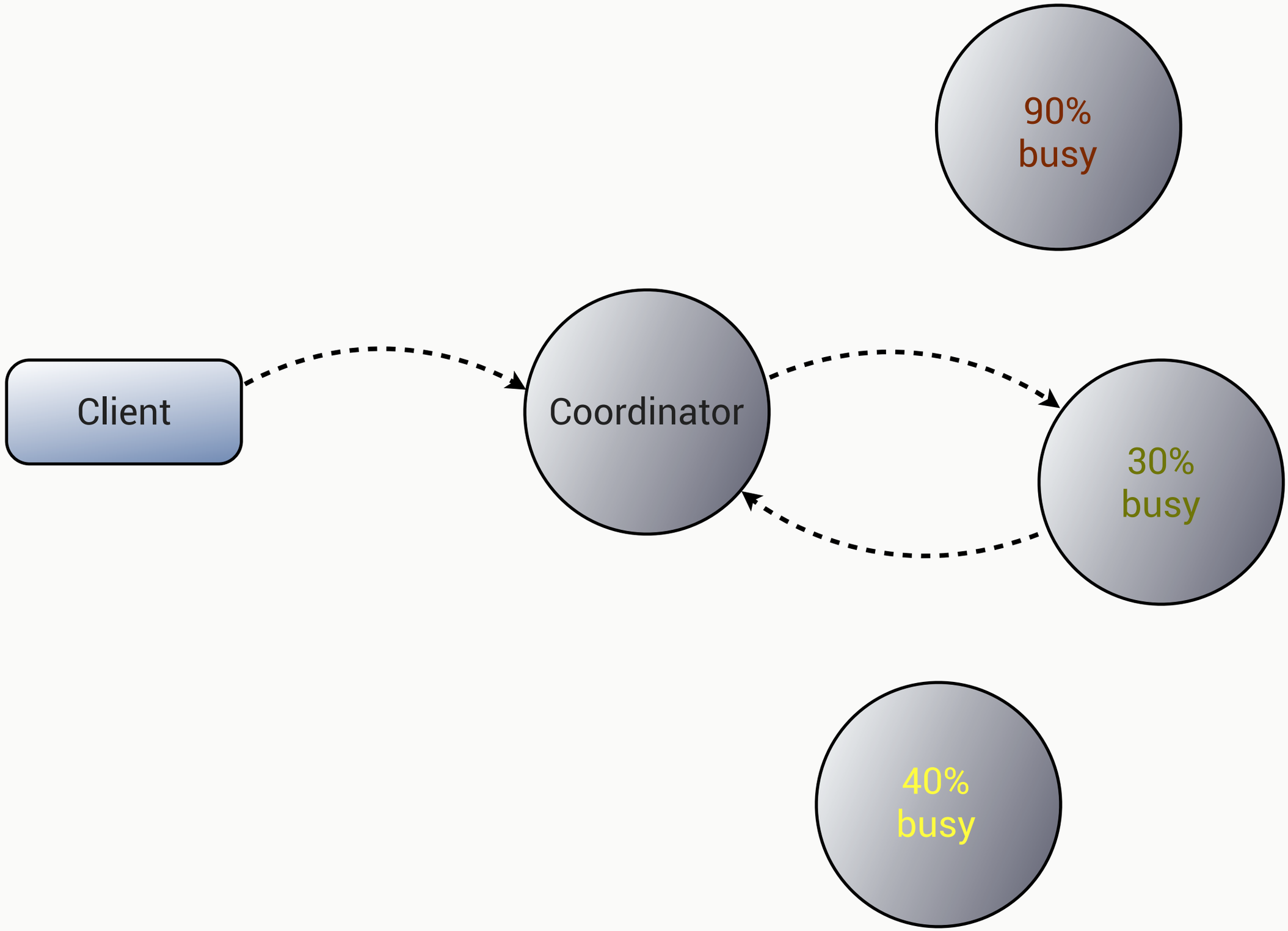
Typical reads



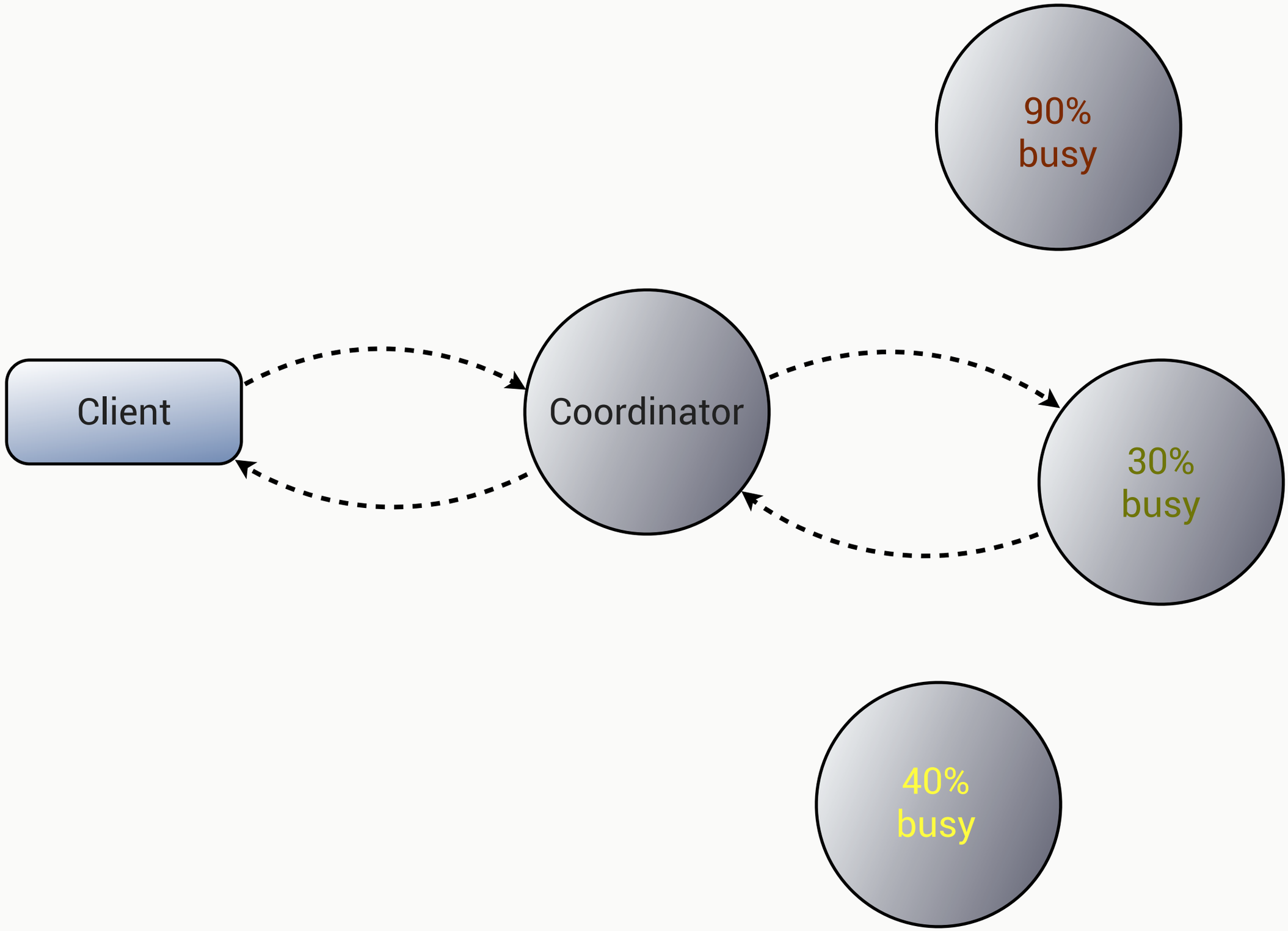
Typical reads



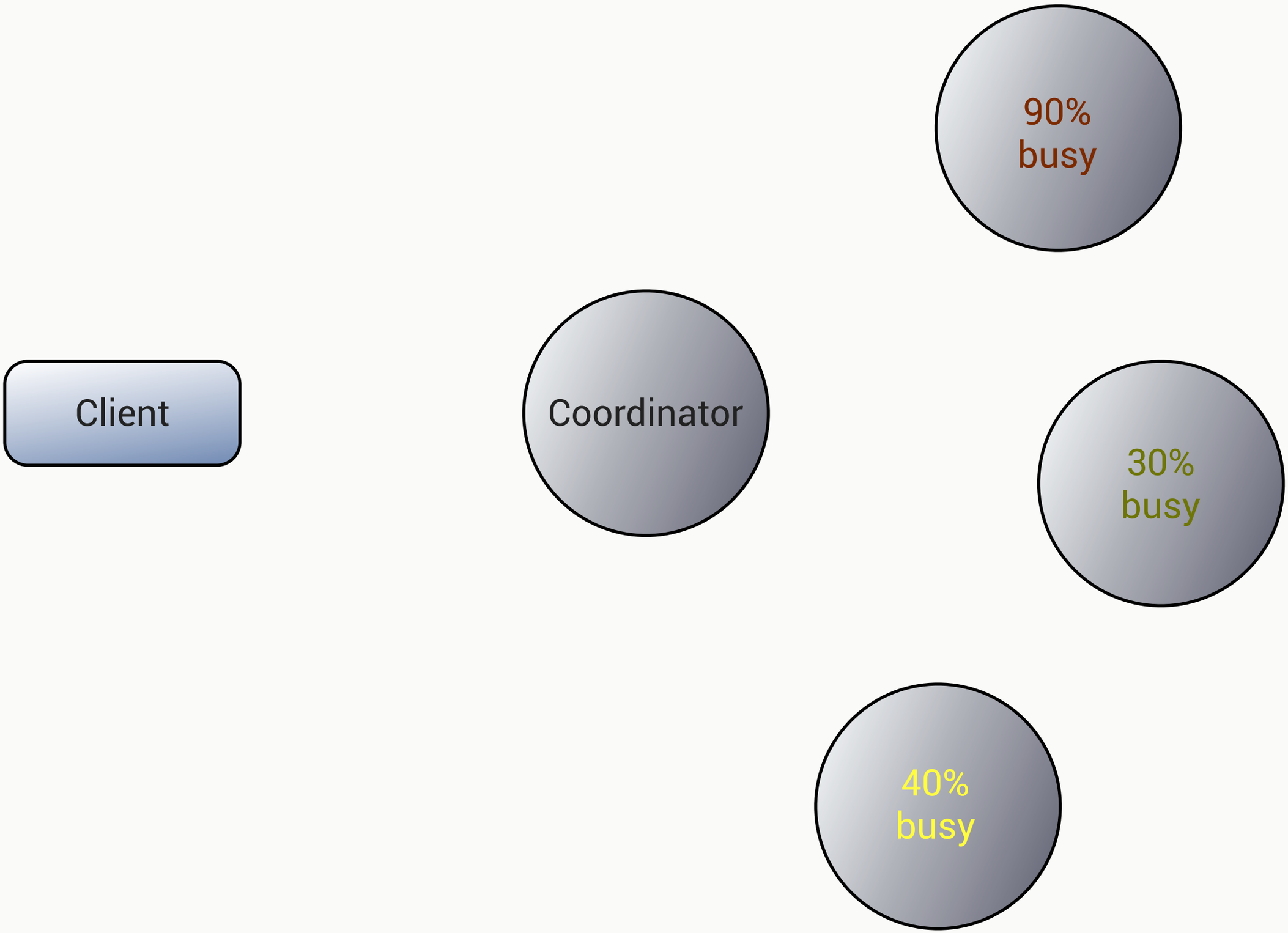
Typical reads



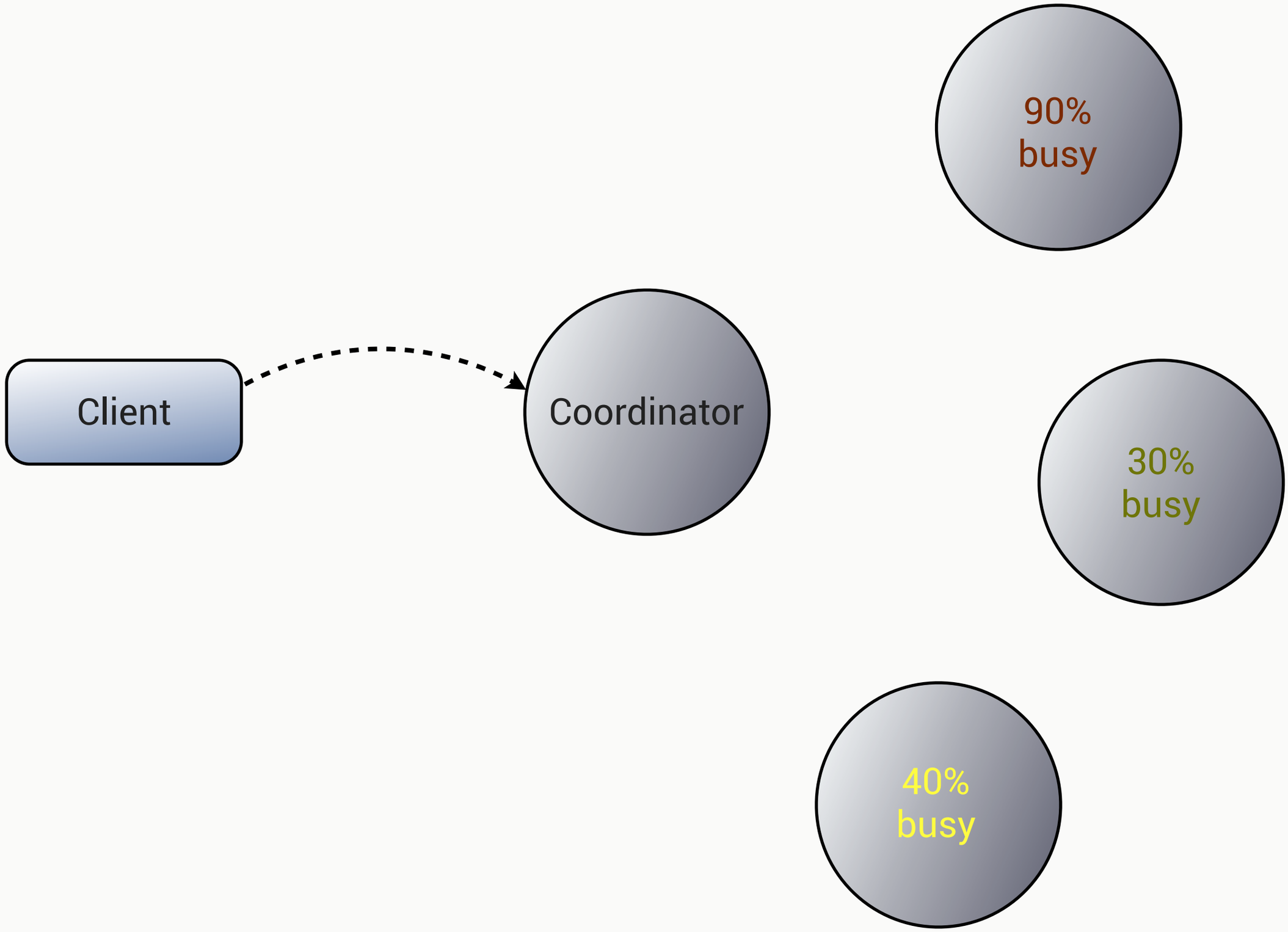
Typical reads



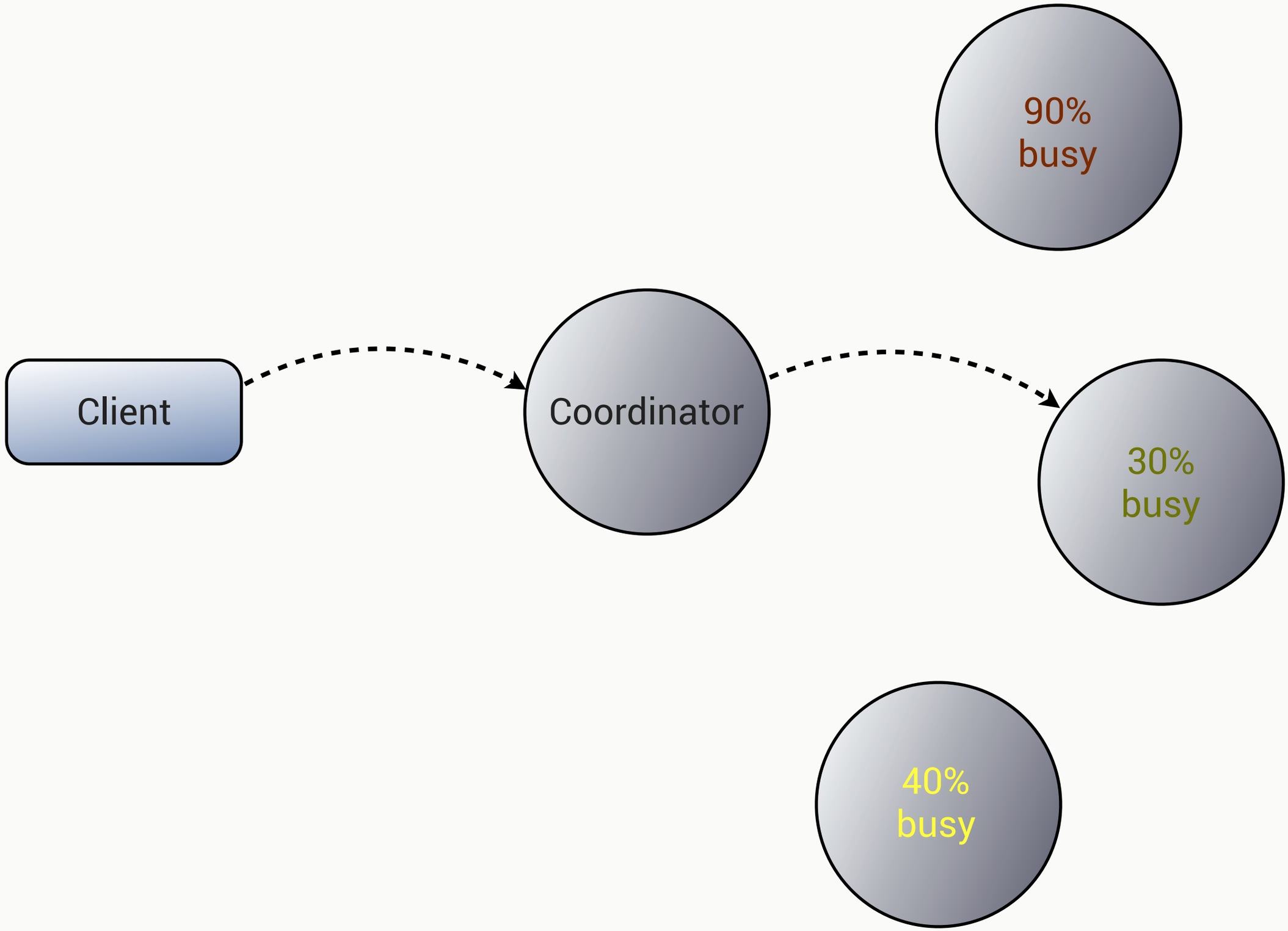
A failure



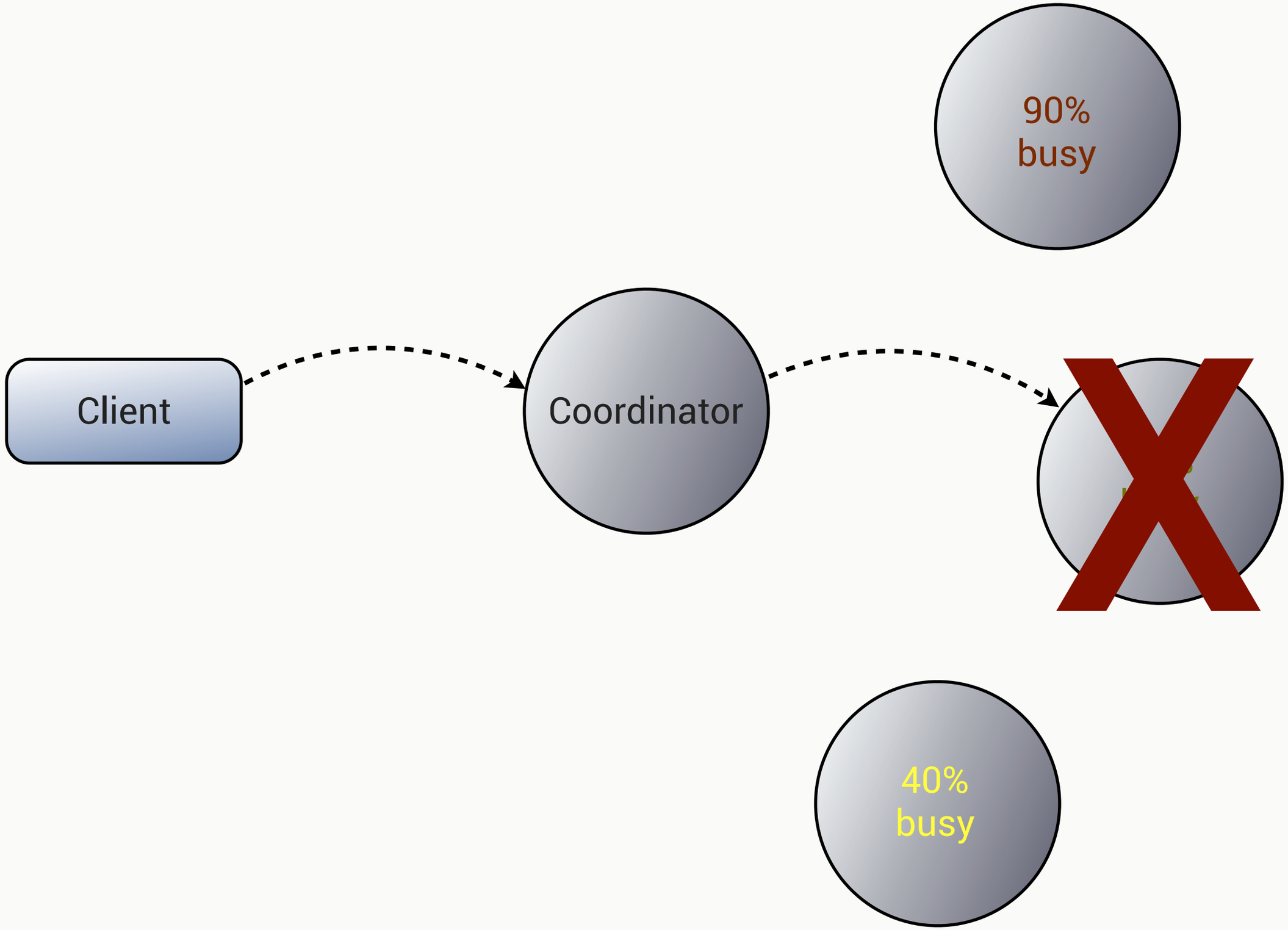
A failure



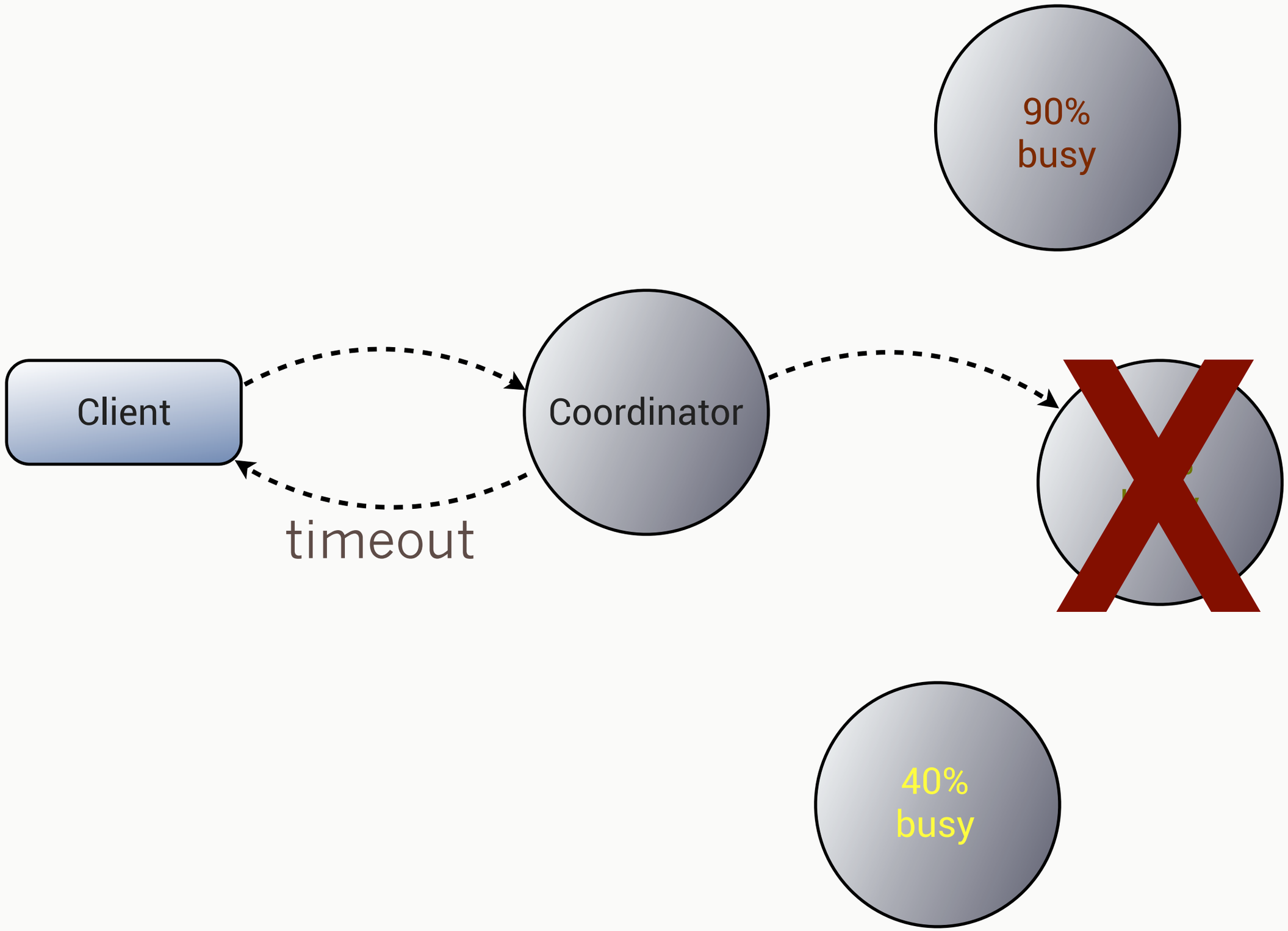
A failure



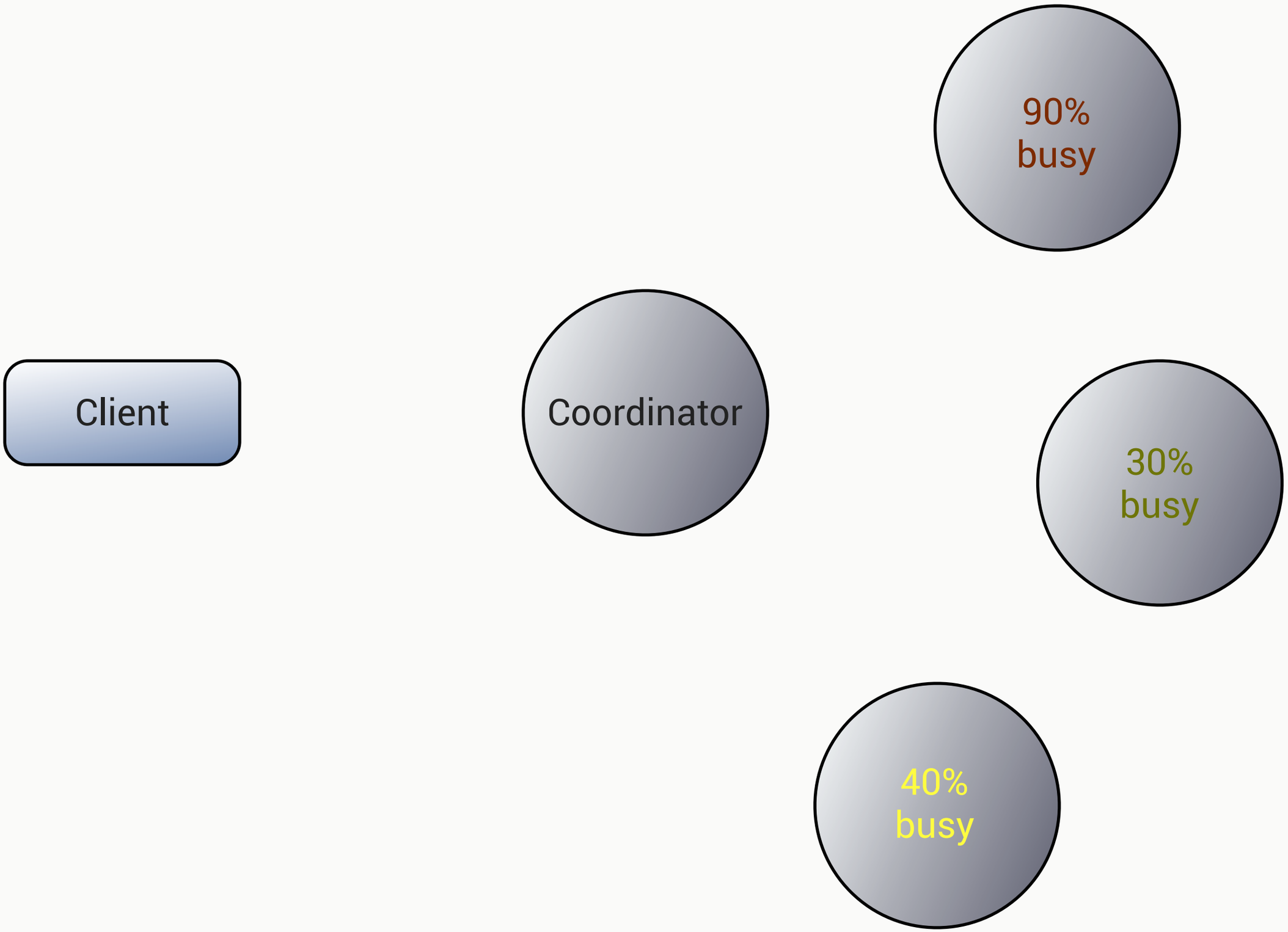
A failure



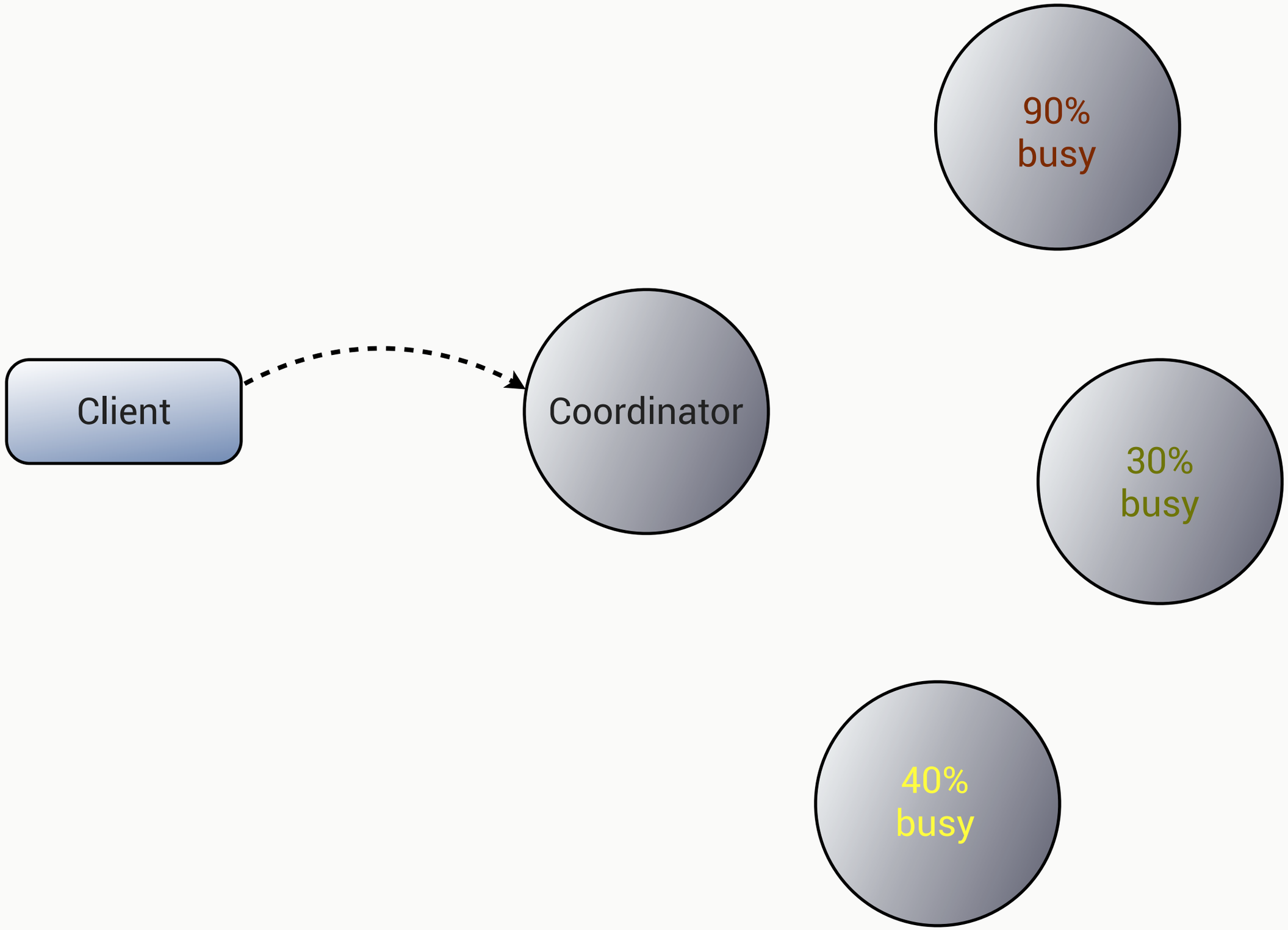
A failure



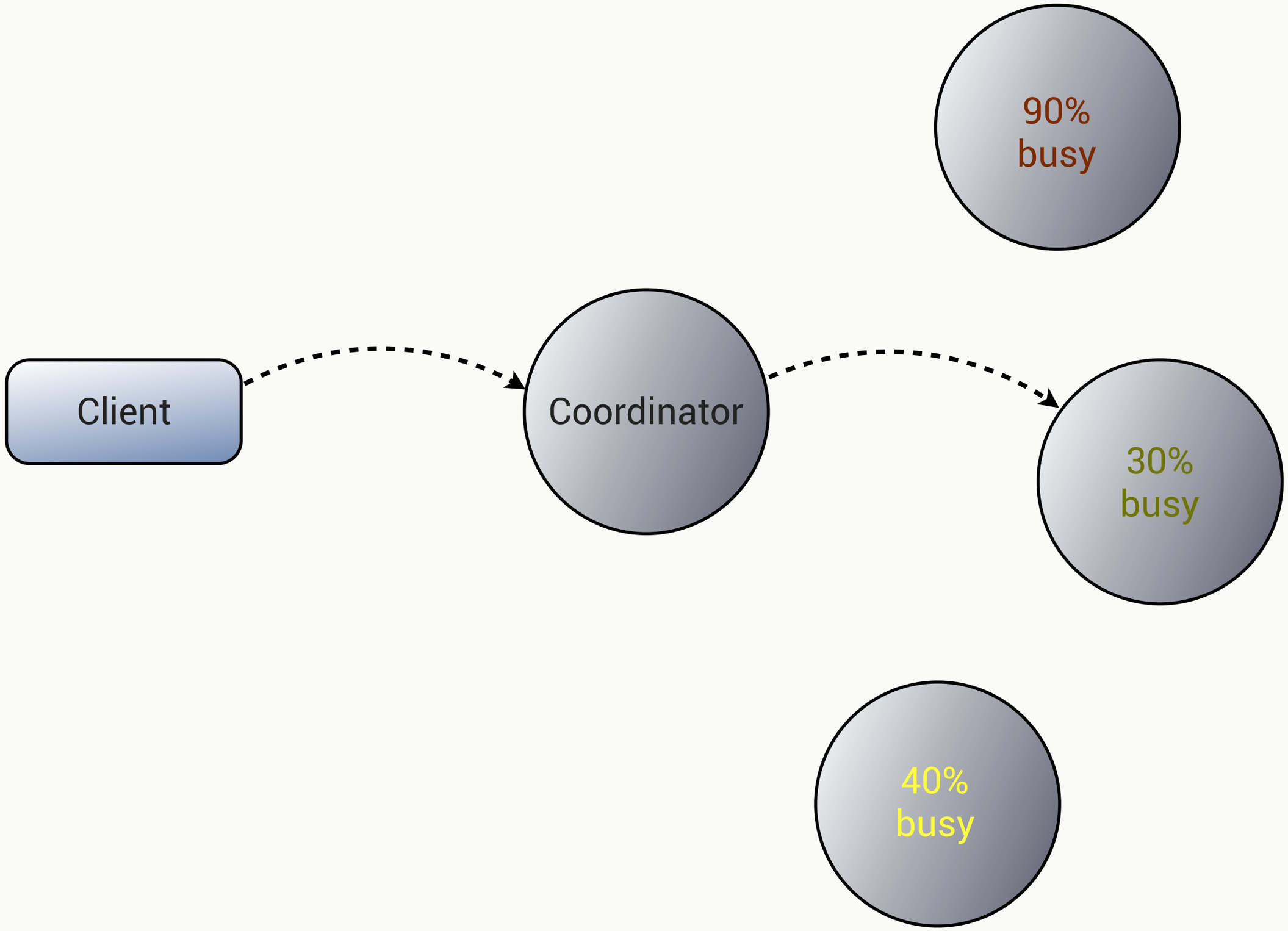
Failure with read protection



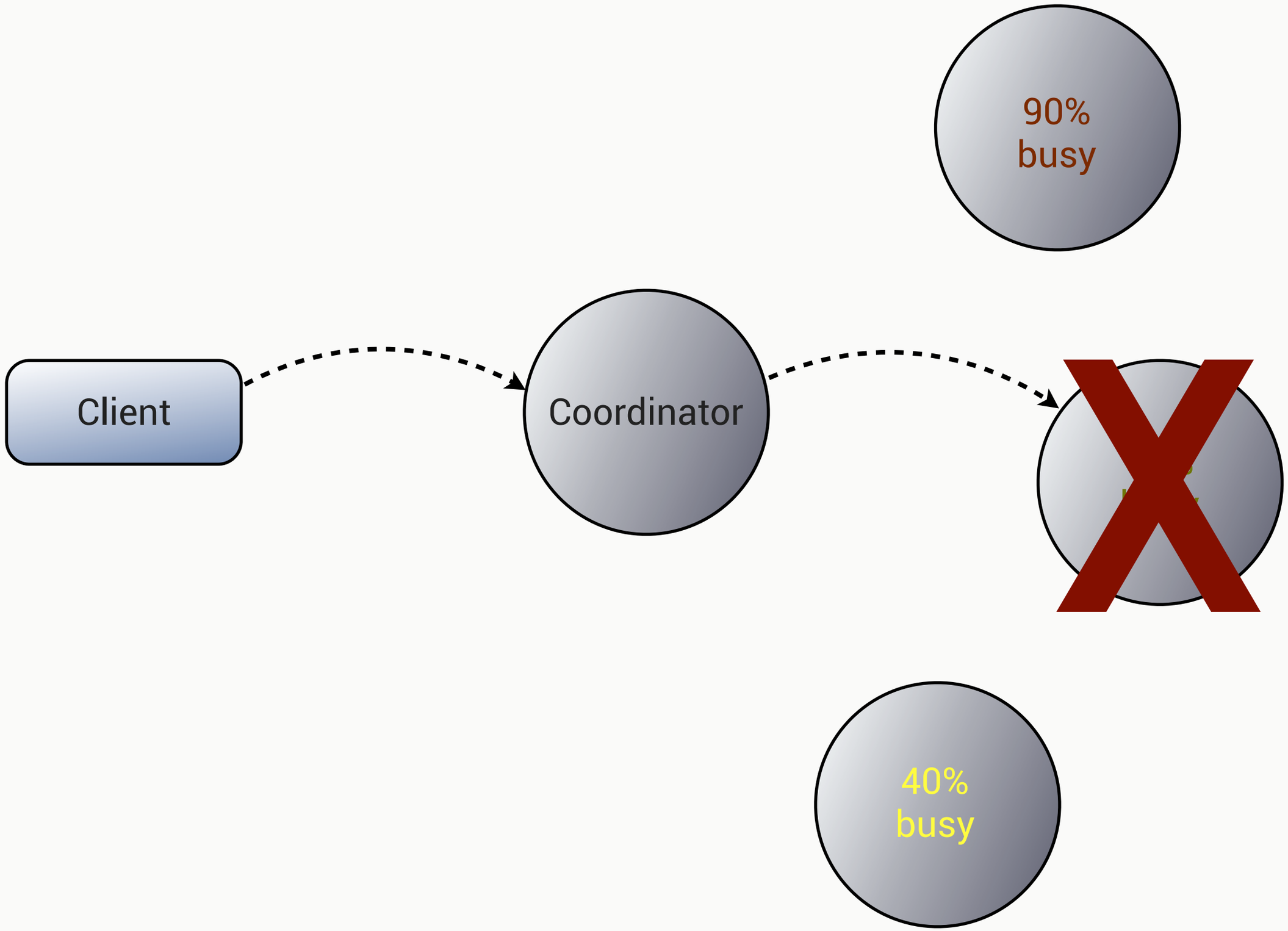
Failure with read protection



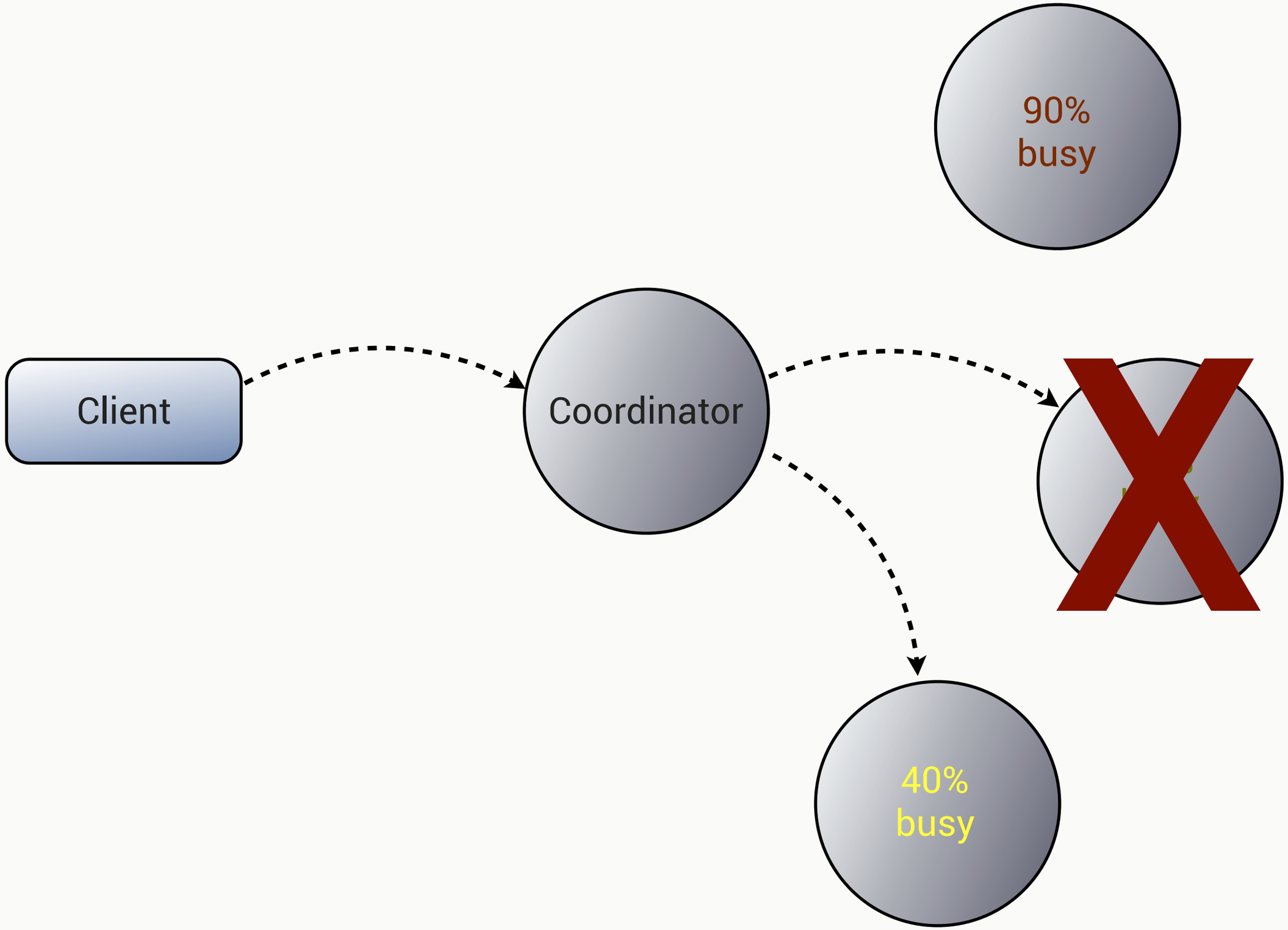
Failure with read protection



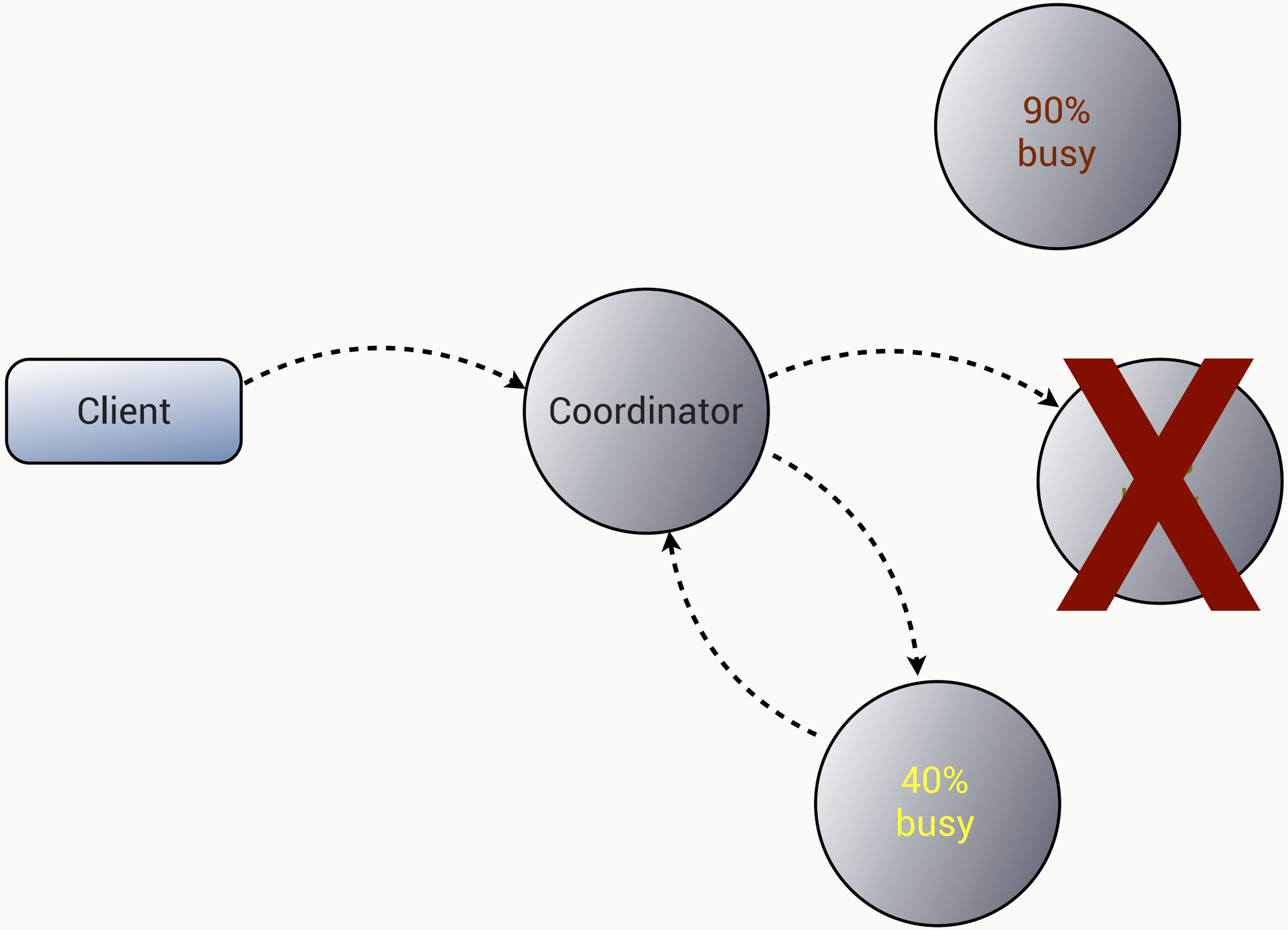
Failure with read protection



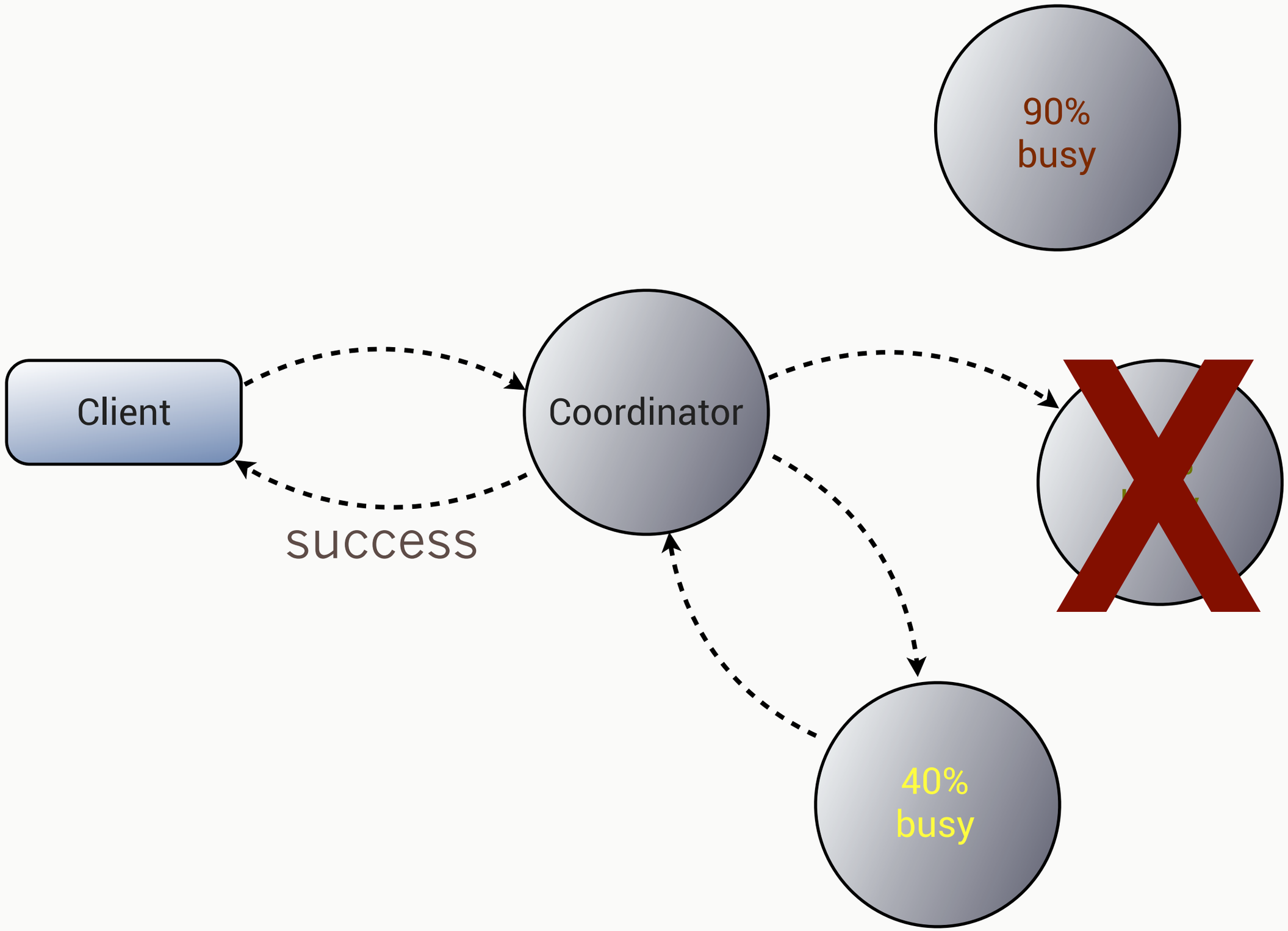
Failure with read protection



Failure with read protection



Failure with read protection



Latency (mid-compaction)

6692c50412ef7d - speculative_retry=NONE

```
Averages from the middle 80% of values:
interval_op_rate          : 39156
interval_key_rate         : 39156
latency median            : 0.8
latency 95th percentile   : 2.6
latency 99.9th percentile : 48.8
Total operation time      : 00:26:10
cmd: -n 60000000 -o read -i 5 -K 20
```

6692c50412ef7d - speculative_retry=90percentile

```
Averages from the middle 80% of values:
interval_op_rate          : 38523
interval_key_rate         : 38523
latency median            : 0.8
latency 95th percentile   : 3.0
latency 99.9th percentile : 14.5
Total operation time      : 00:26:35
cmd: -n 60000000 -o read -i 5 -K 20
```

6692c50412ef7d - speculative_retry=ALWAYS

```
Averages from the middle 80% of values:
interval_op_rate          : 34823
interval_key_rate         : 34823
latency median            : 0.8
latency 95th percentile   : 2.6
latency 99.9th percentile : 36.5
Total operation time      : 00:29:11
cmd: -n 60000000 -o read -i 5 -K 20
```

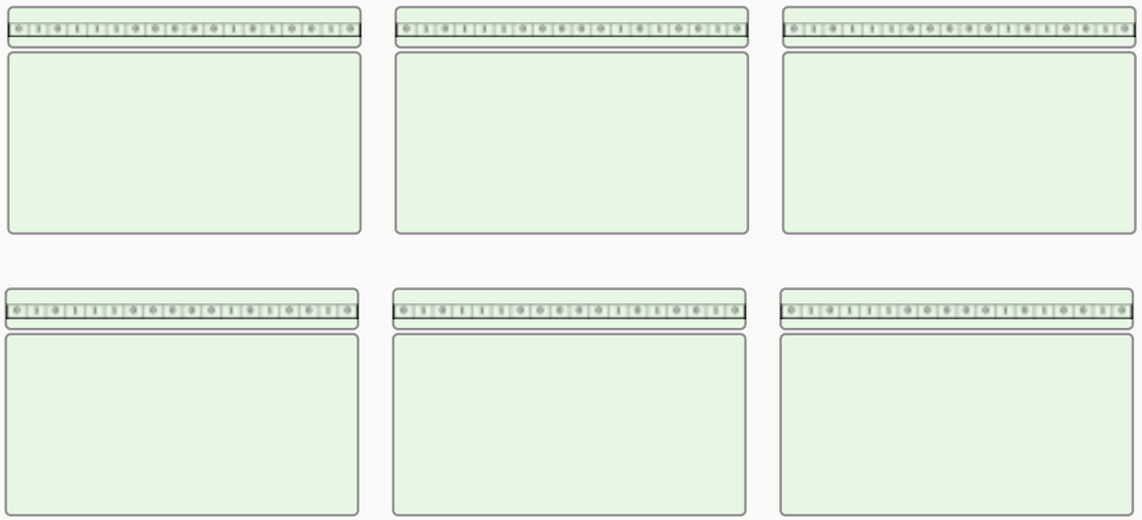
6692c50412ef7d - speculative_retry=75percentile

```
Averages from the middle 80% of values:
interval_op_rate          : 36764
interval_key_rate         : 36764
latency median            : 0.8
latency 95th percentile   : 3.0
latency 99.9th percentile : 16.8
Total operation time      : 00:27:41
cmd: -n 60000000 -o read -i 5 -K 20
```

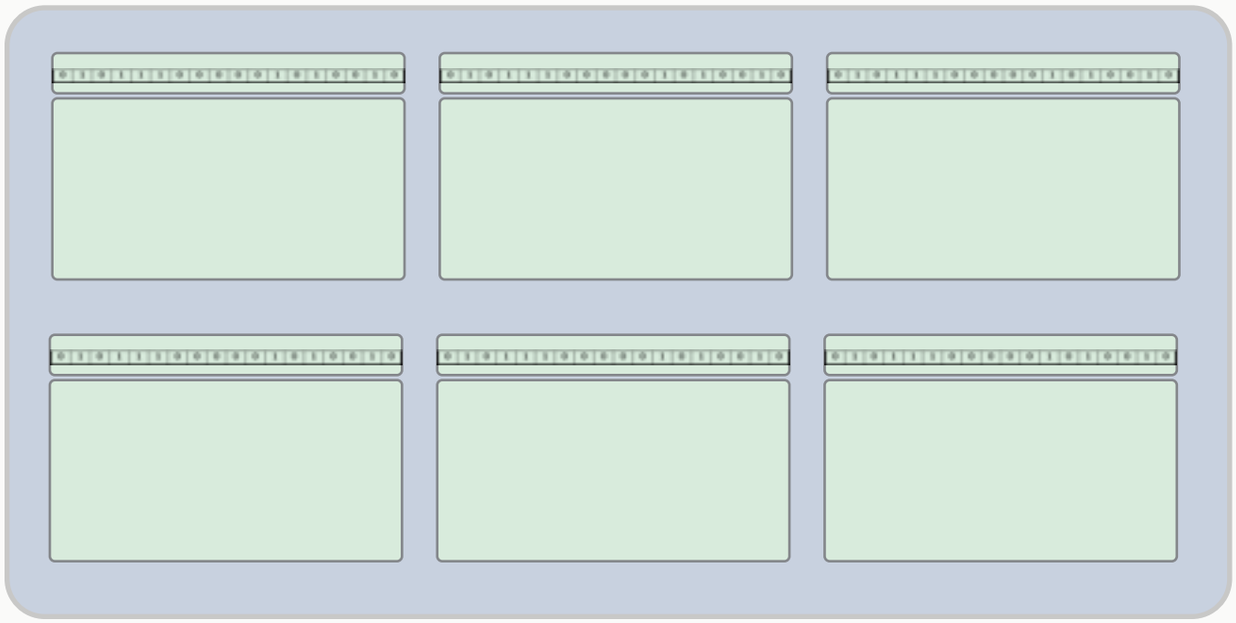
6692c50412ef7d - speculative_retry=99percentile

```
Averages from the middle 80% of values:
interval_op_rate          : 40466
interval_key_rate         : 40466
latency median            : 0.8
latency 95th percentile   : 2.7
latency 99.9th percentile : 19.6
Total operation time      : 00:25:28
cmd: -n 60000000 -o read -i 5 -K 20
```

More-efficient repair

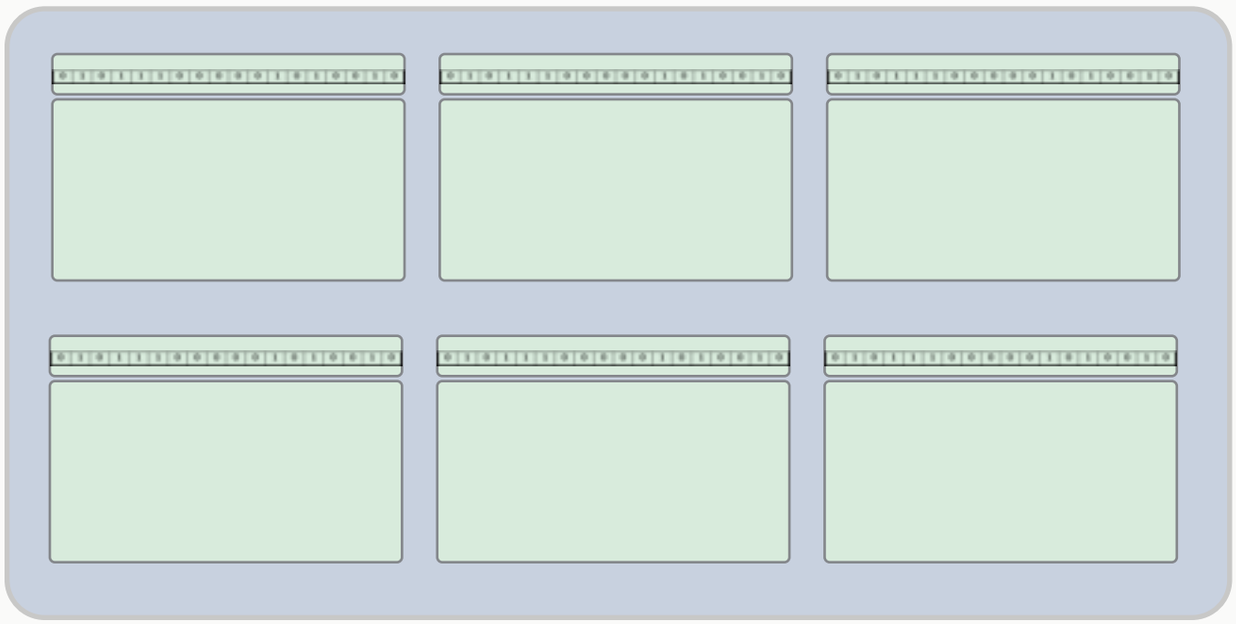


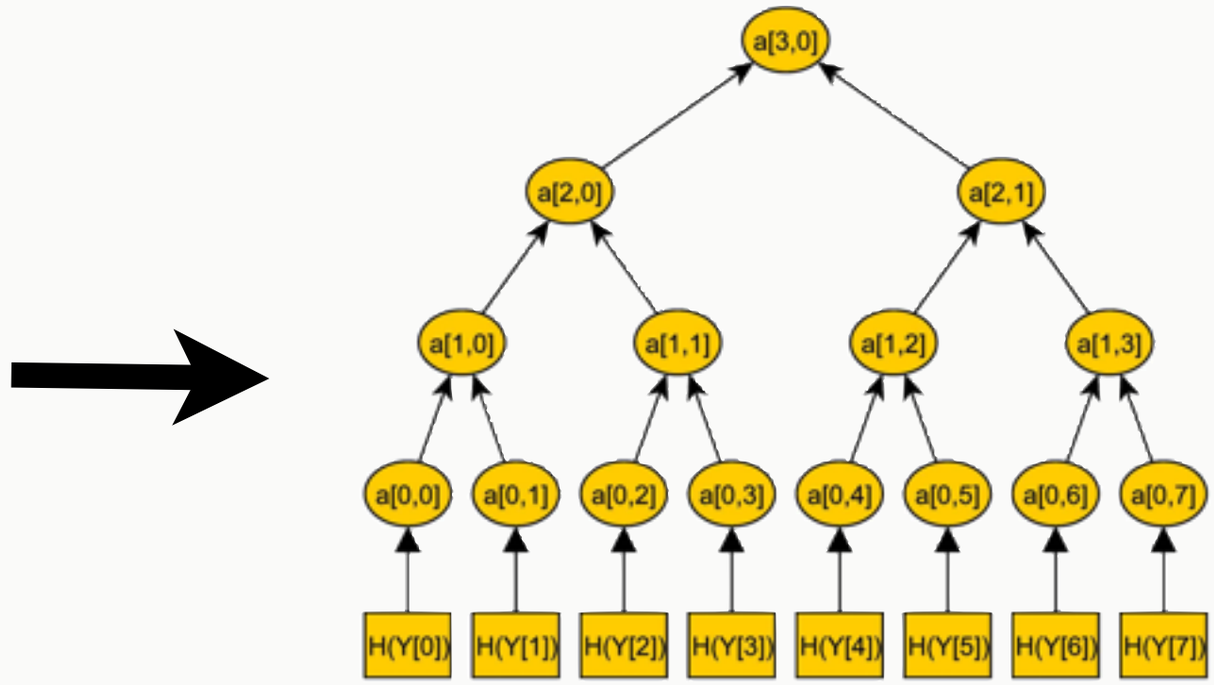
More-efficient repair



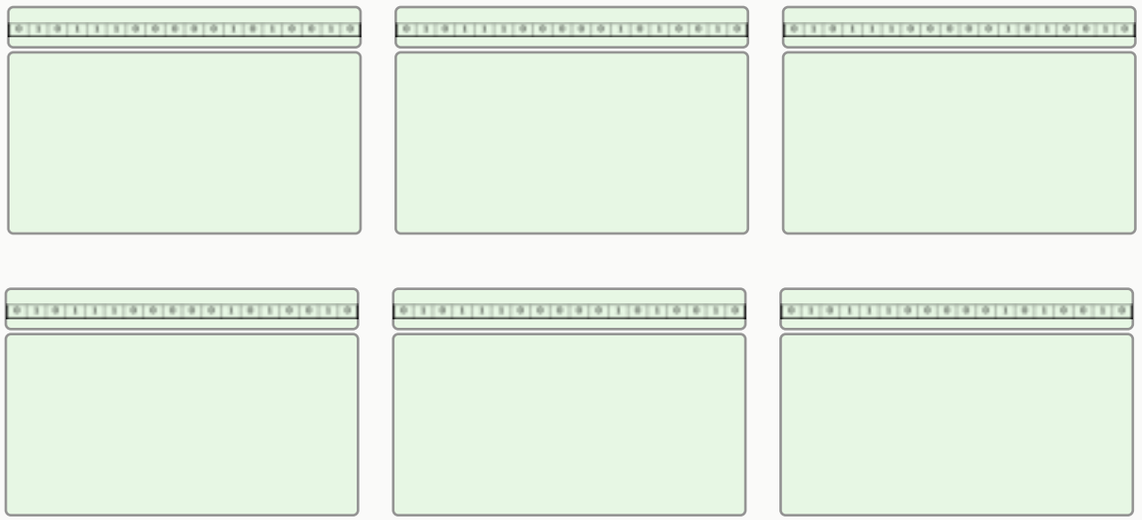
More-efficient repair

DATASTAX

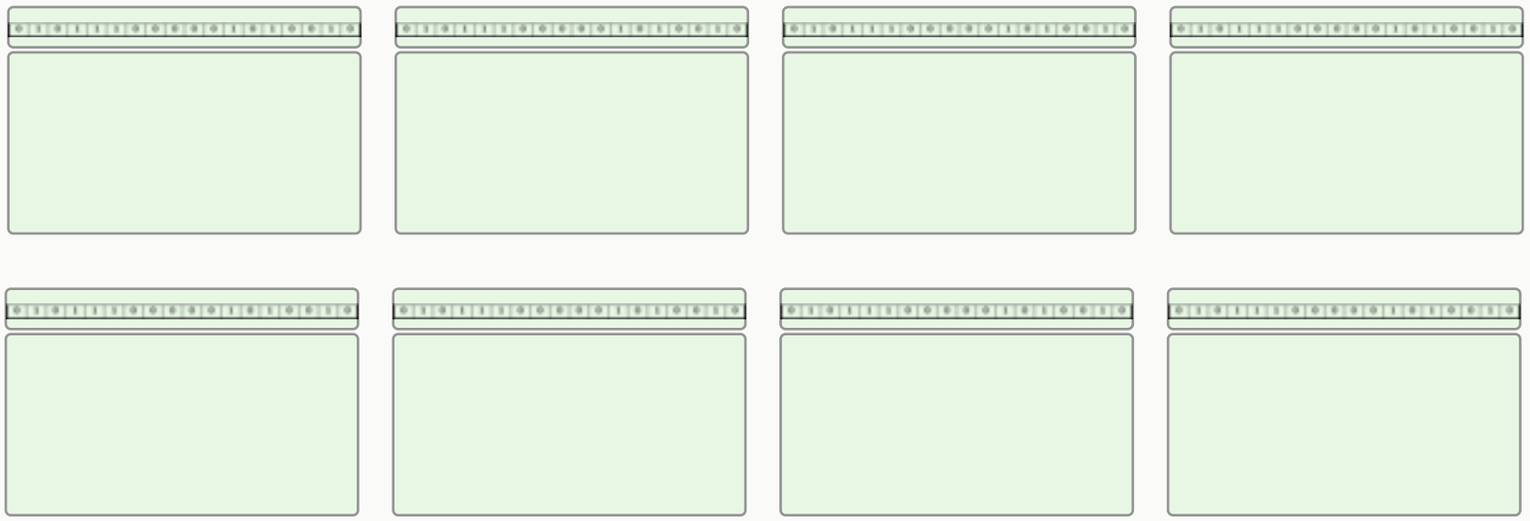




More-efficient repair

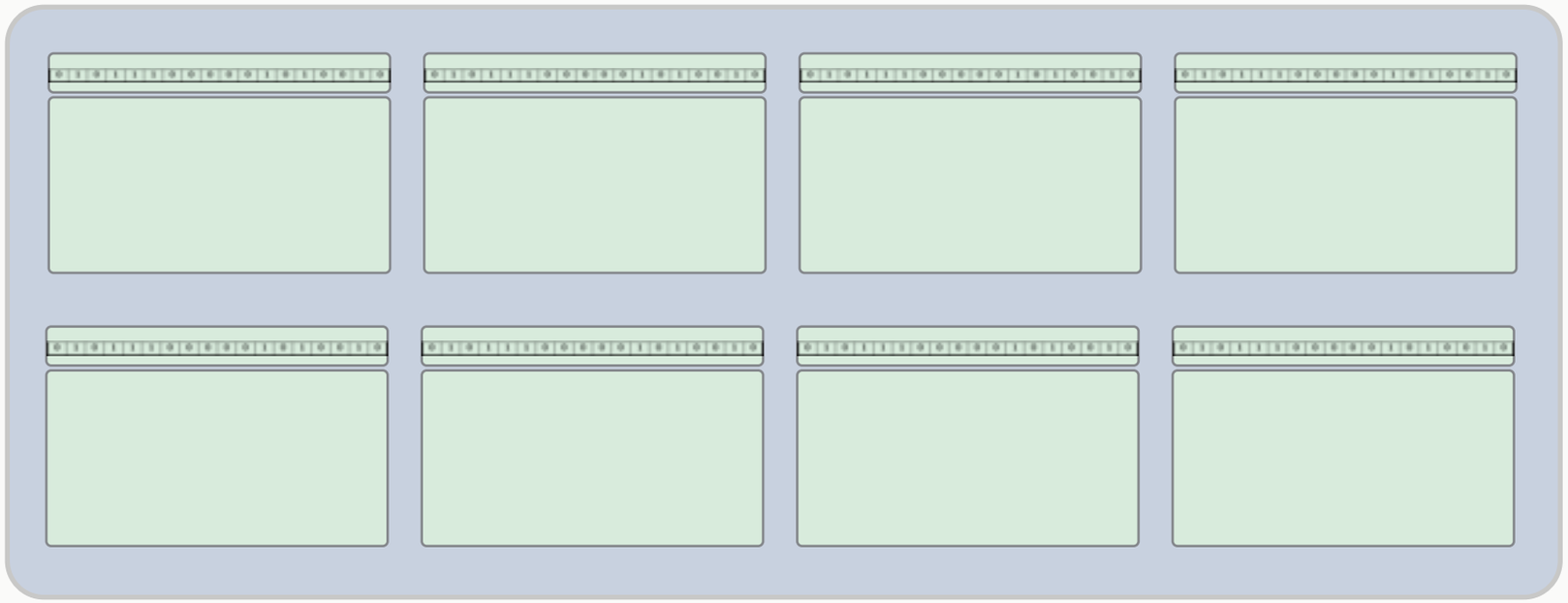


More-efficient repair

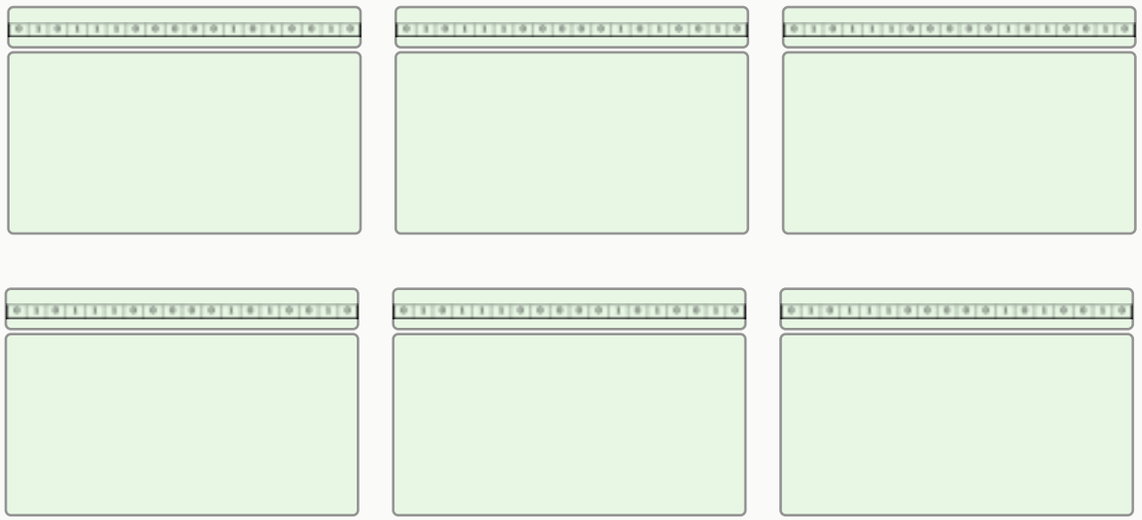


More-efficient repair

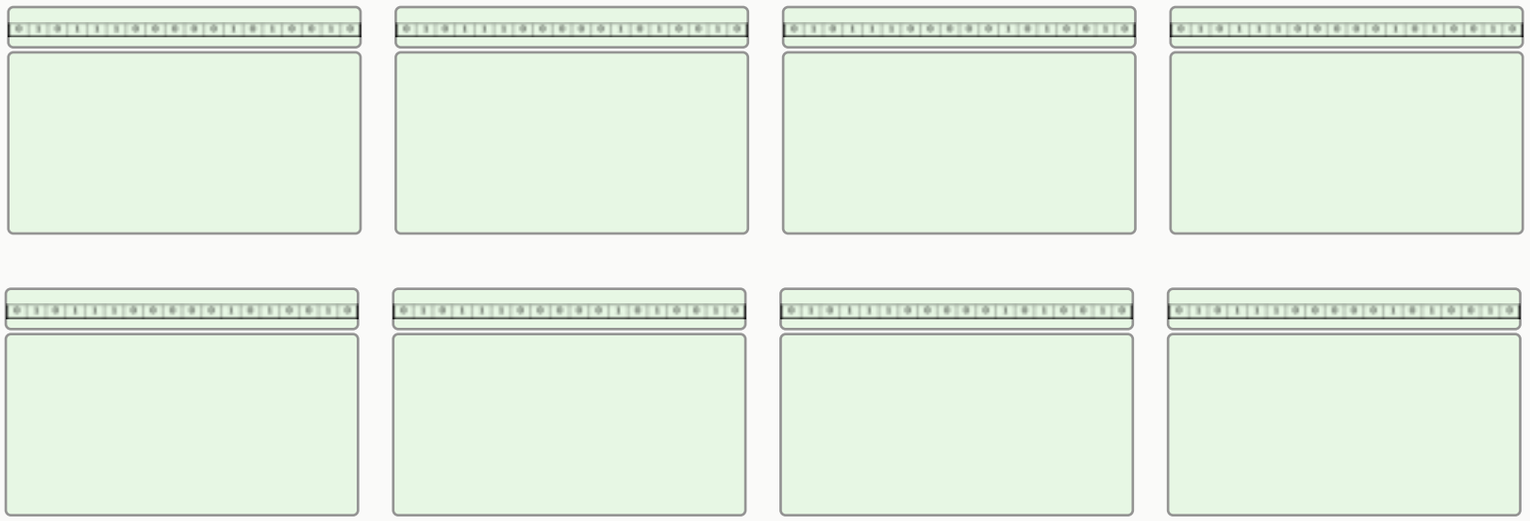
DATASTAX



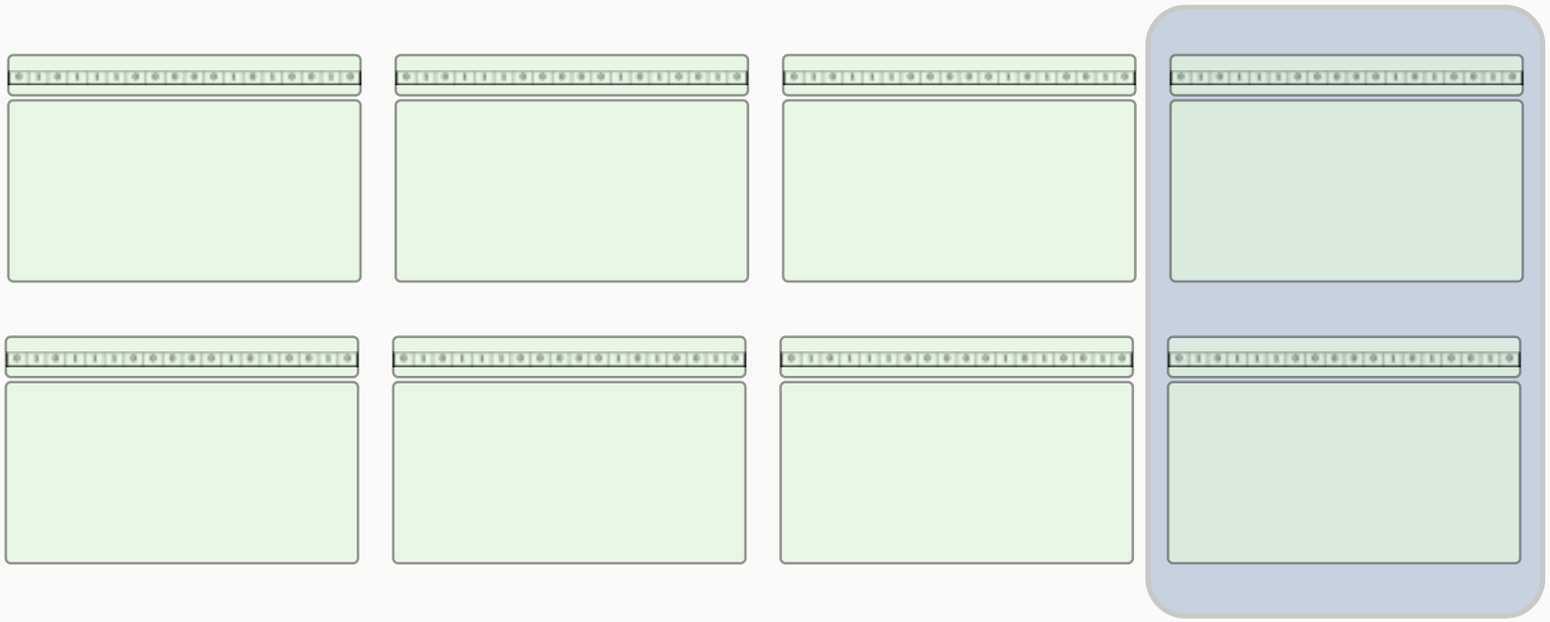
More-efficient repair



More-efficient repair



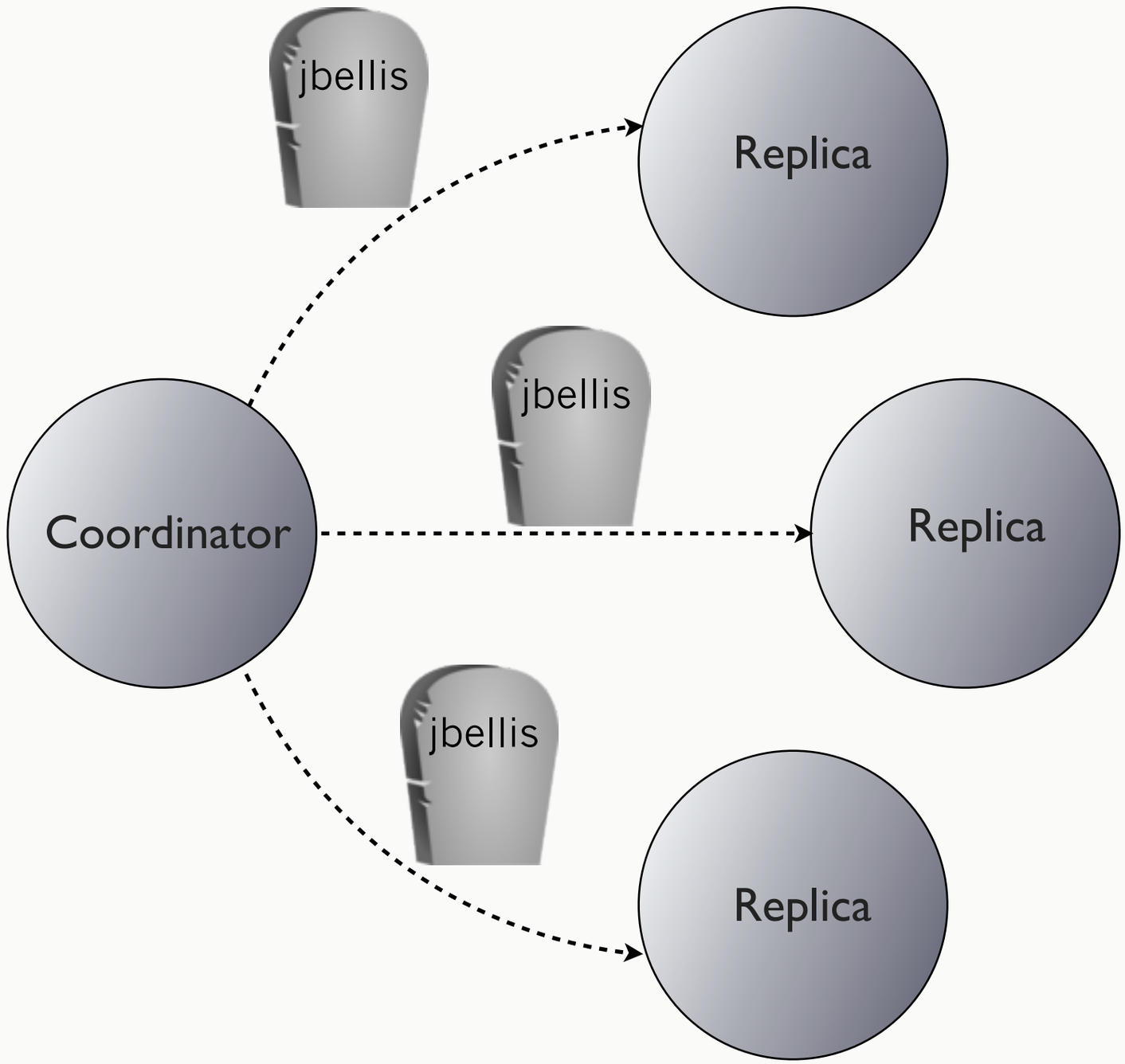
More-efficient repair



Tombstones!

DATASTAX

```
DELETE FROM users
WHERE username = 'jbellis'
```



When can we purge?

DATASTAX

- gc_grace_seconds

Pain points

DATASTAX

Pain points

- Easy to write a query that is O(N) in the number of tombstones
  - Tombstones must be read-repaired

Pain points

- Easy to write a query that is O(N) in the number of tombstones
  - Tombstones must be read-repaired
- Clumsy hammer
  - tombstone_warn_threshold: 1000
  - tombstone_failure_threshold: 100000

Pain points

- Easy to write a query that is O(N) in the number of tombstones
  - Tombstones must be read-repaired
- Clumsy hammer
  - tombstone_warn_threshold: 1000
  - tombstone_failure_threshold: 100000
- http://www.datastax.com/dev/blog/cassandra-anti-patterns-queues-and-queue-like-datasets