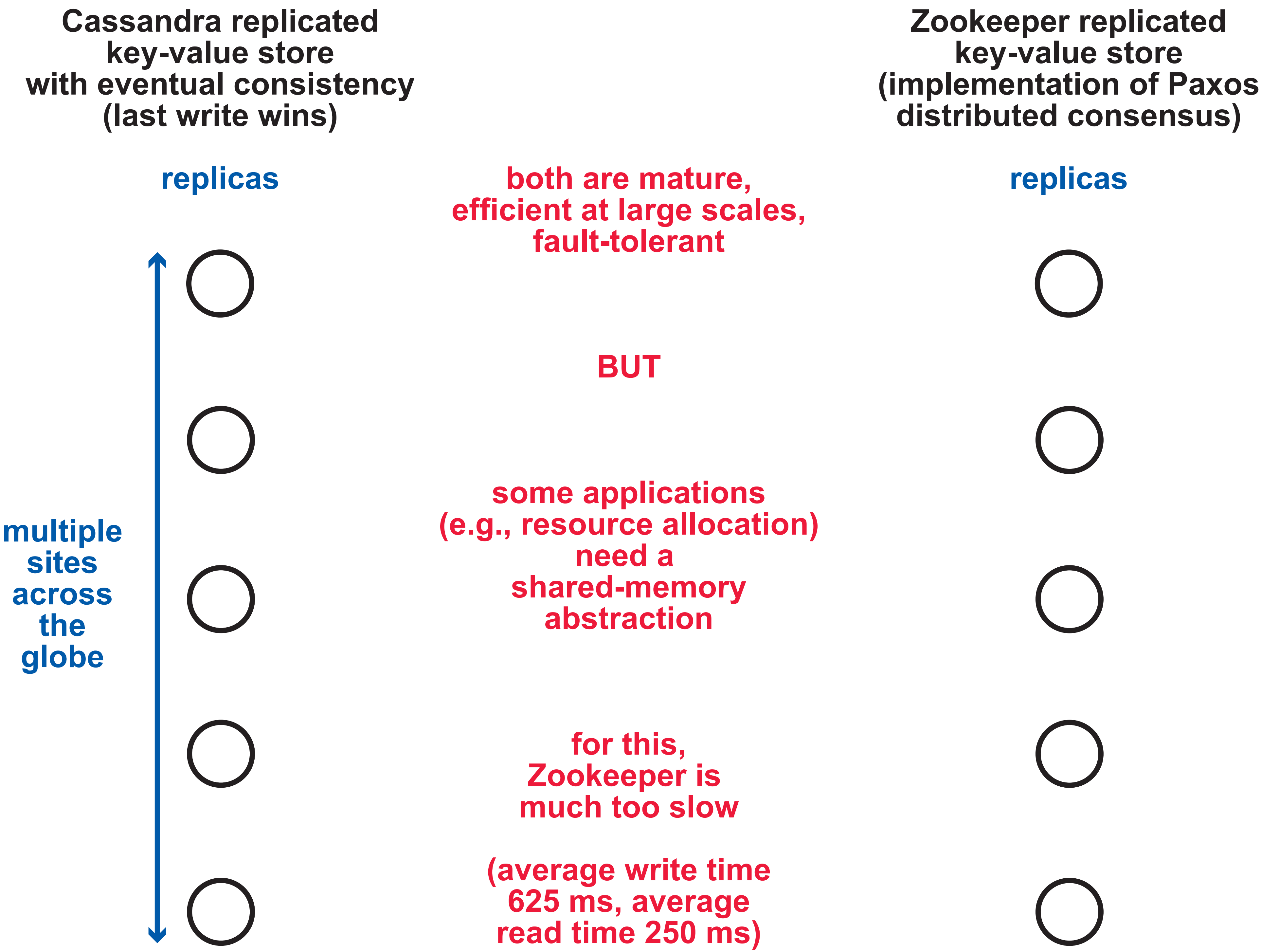
VERIFICATION OF GEO-DISTRIBUTED SERVICES*

Pamela Zave

Princeton University

Princeton, New Jersey

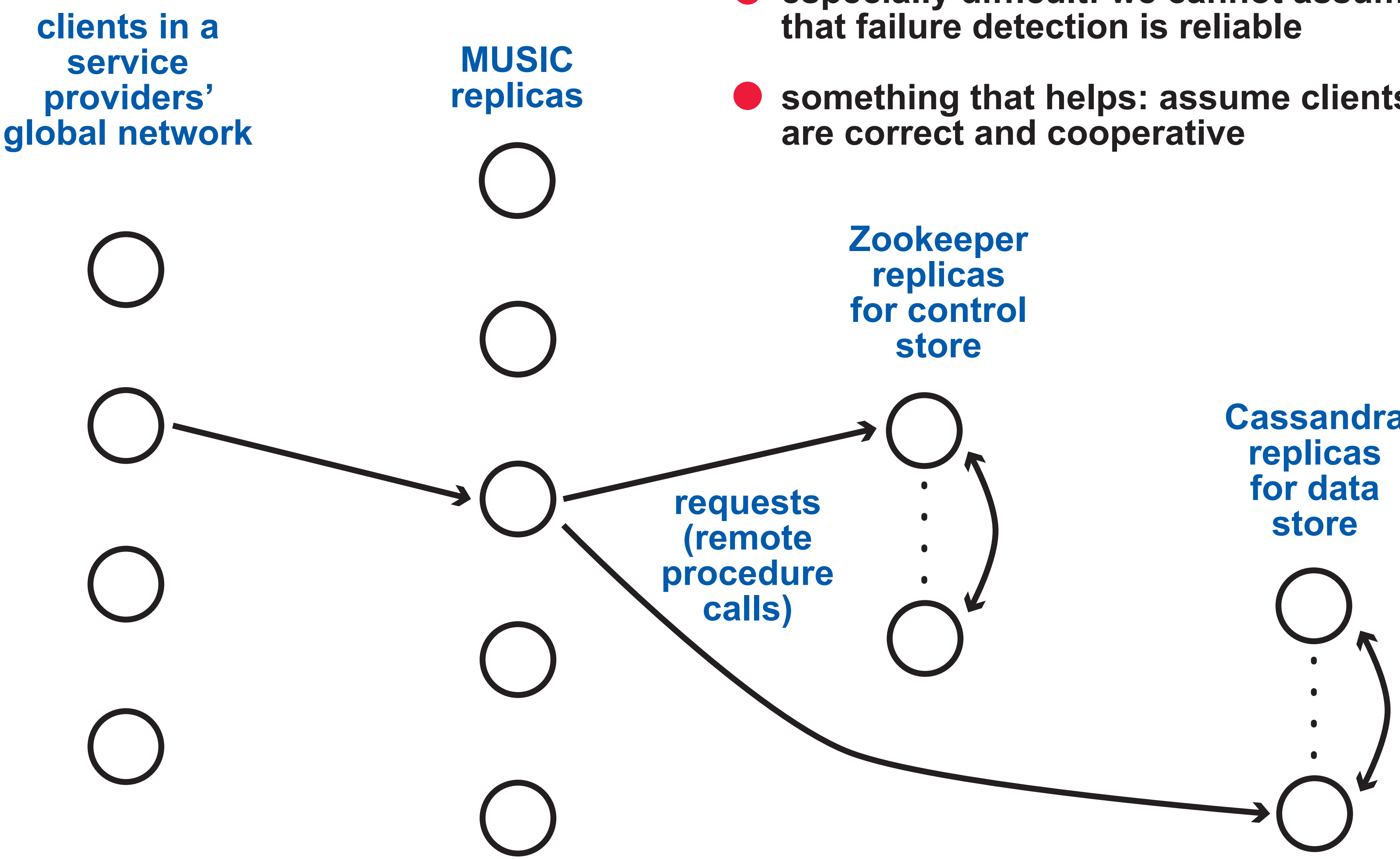
***Joint work with Bharath Balasubramanian, AT&T Labs**


Cassandra replicated key-value store with eventual consistency (last write wins)
Zookeeper replicated key-value store (implementation of Paxos distributed consensus)
replicas
both are mature, efficient at large scales, fault-tolerant
replicas
BUT
multiple sites across the globe
some applications (e.g., resource allocation) need a shared-memory abstraction
for this, Zookeeper is much too slow
(average write time 625 ms, average read time 250 ms)

MUSIC FOR GEO-DISTRIBUTED SERVICES

We have a solution. It is fault-tolerant and efficient, also complex and subtle.

- especially difficult: we cannot assume that failure detection is reliable
- something that helps: assume clients are correct and cooperative

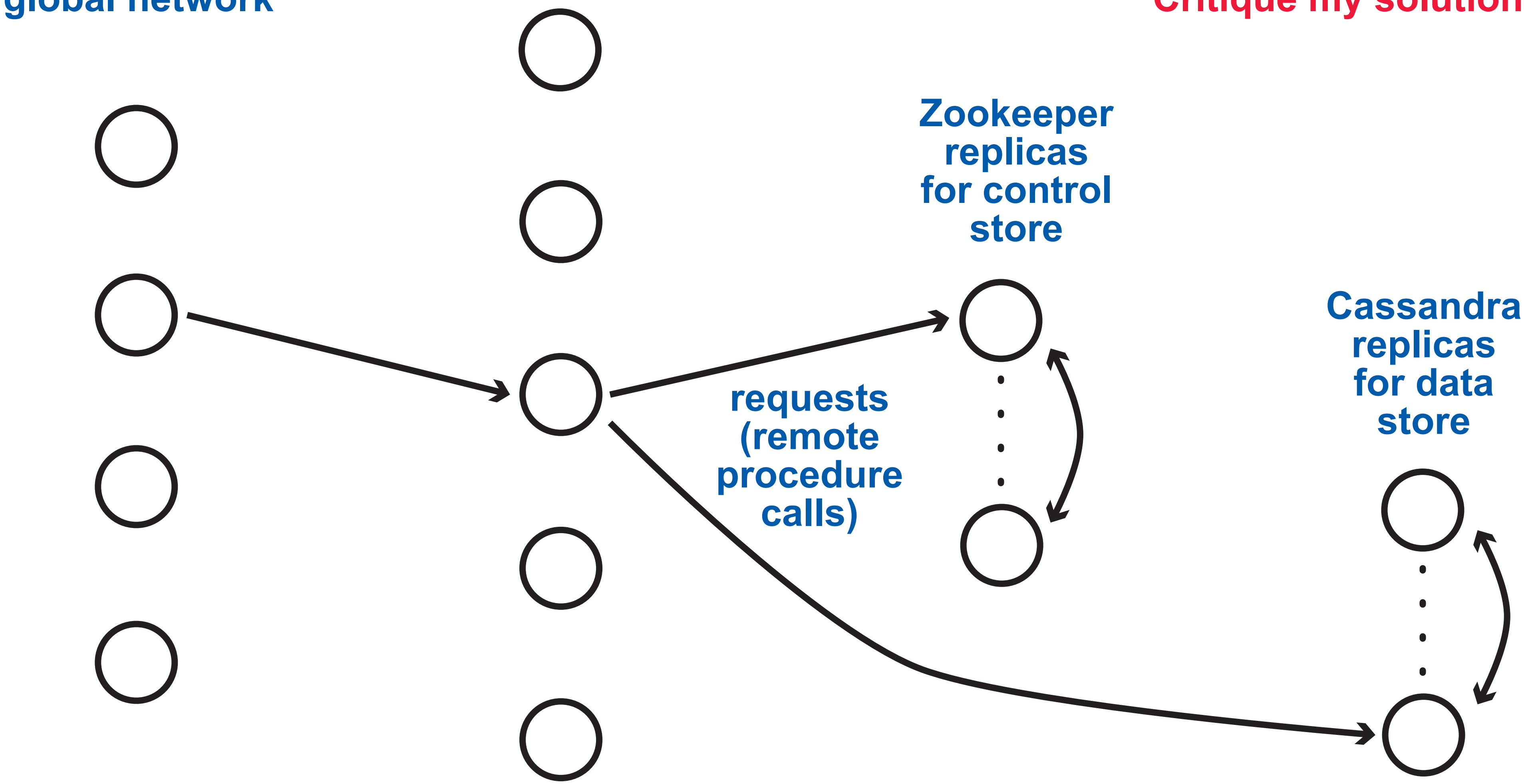


THE TOPIC I WOULD LIKE TO DISCUSS

How do I verify this system?

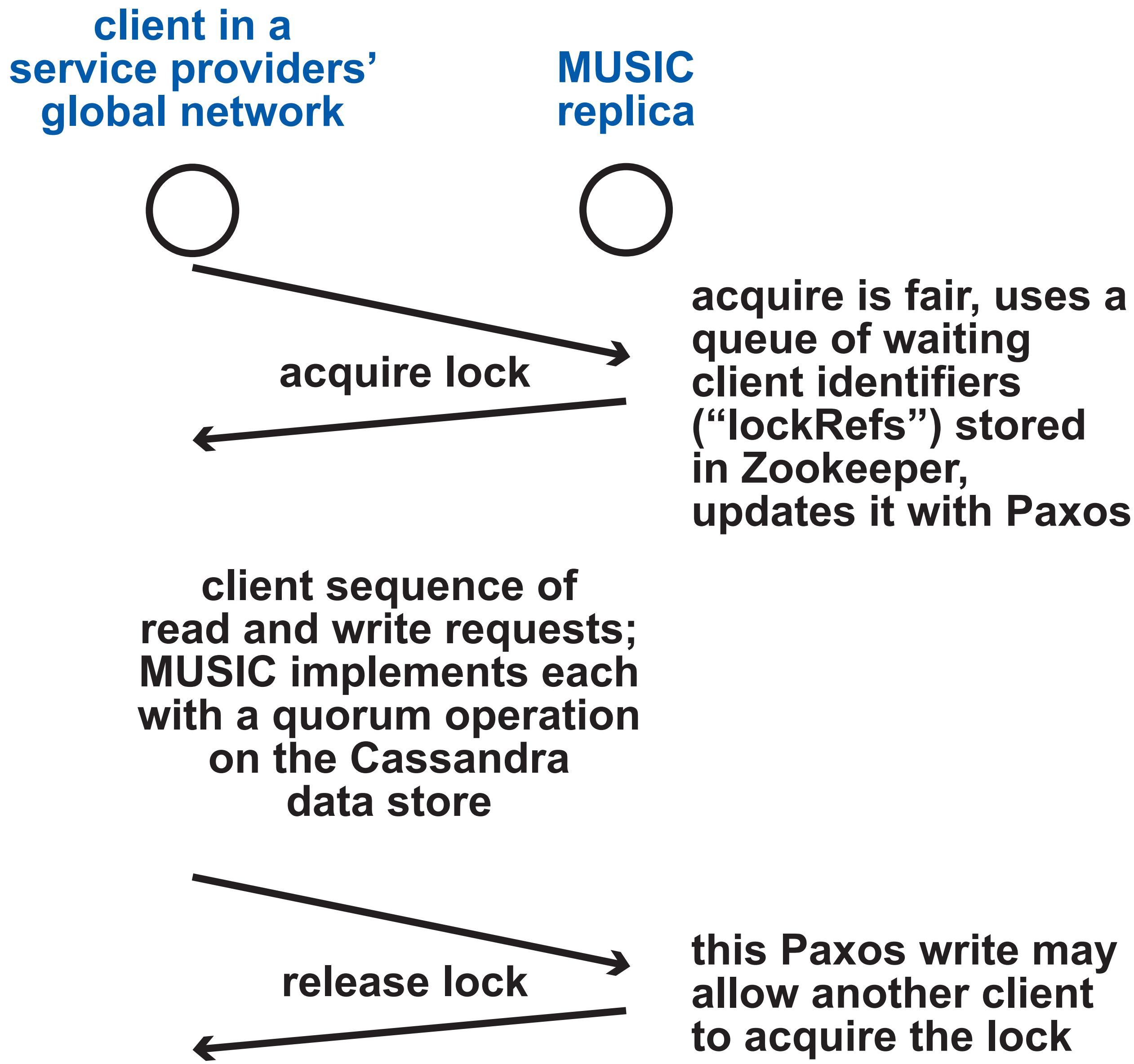
- too much concurrency!
- too much inter-dependent distributed state!
- too many failure possibilities, and detection has false positives!

Critique my solution!



SOLUTION 1: BASIC IDEA

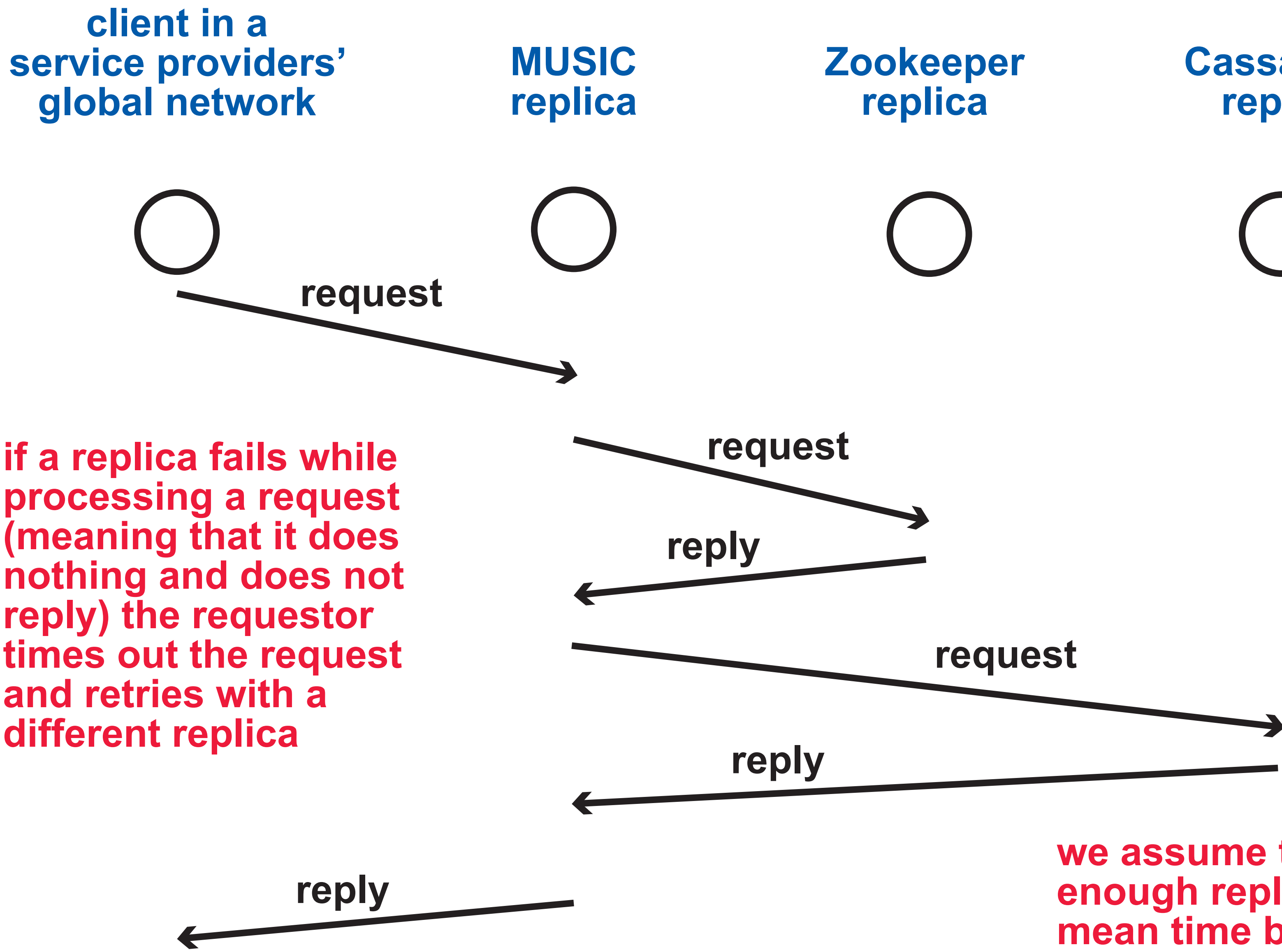
FOR ONE “ENTRY CONSISTENT” KEY IN THE DATA STORE



PROPERTIES

- all writes to the data store are sequential
- every read gets the most recent value
- client requests succeed provided that . . .
  . . . client can connect to a MUSIC replica, . . .
  . . . MUSIC replica can connect to a quorum of ZK/Cass replicas
- only acquire and release require Paxos consensus writes (one each)

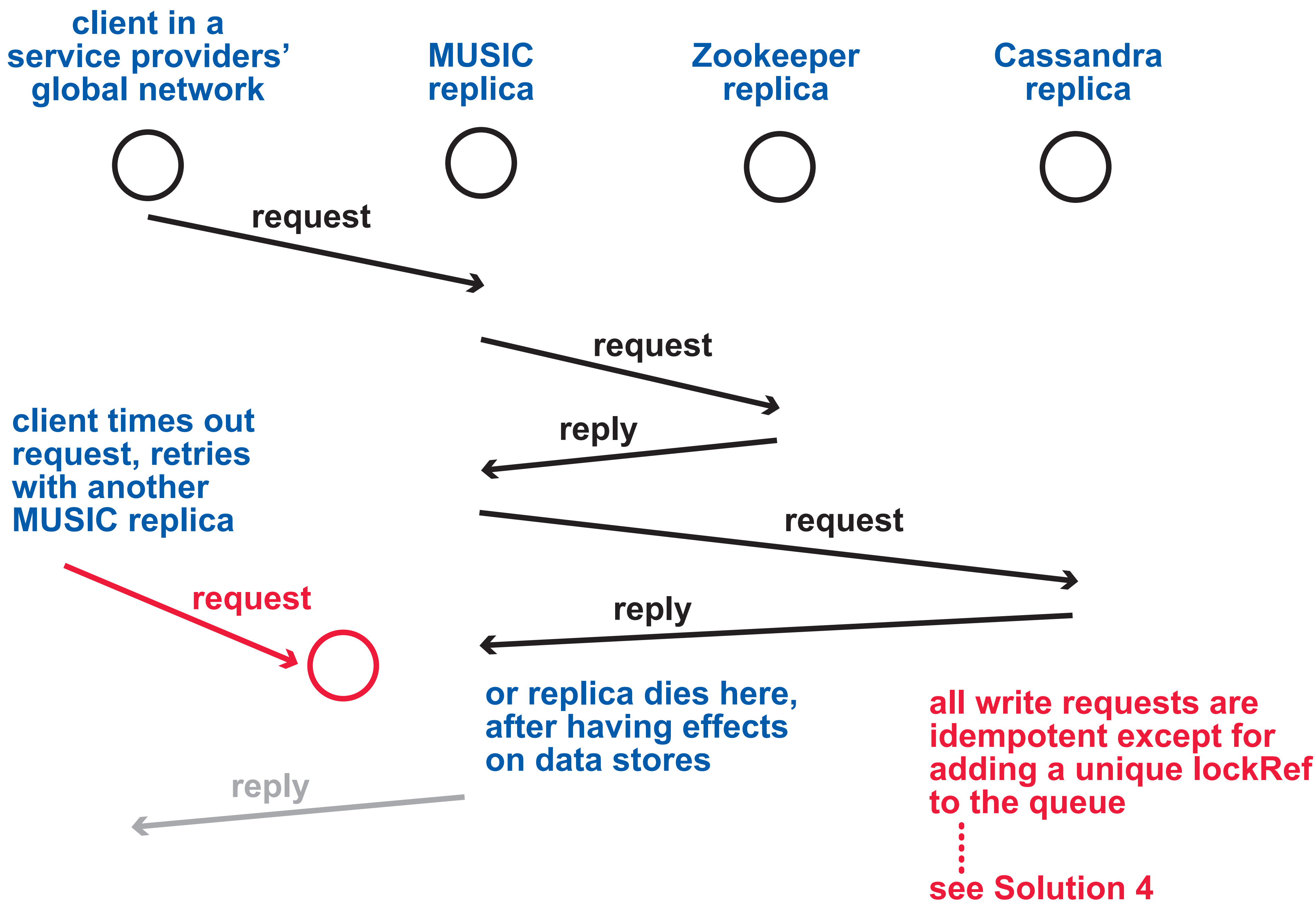
SOLUTION 2: REPLICA FAILURE DURING OPERATION



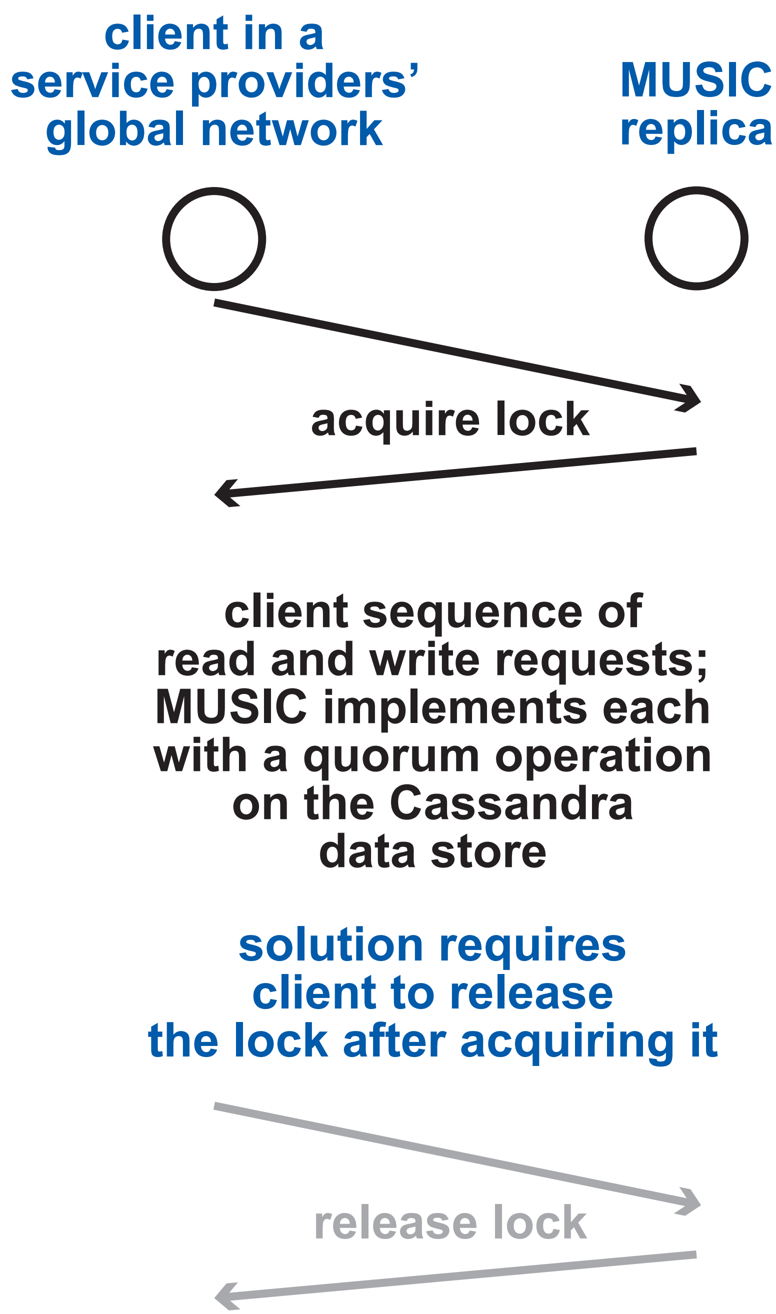
if a replica fails while processing a request (meaning that it does nothing and does not reply) the requestor times out the request and retries with a different replica

we assume that there are enough replicas, and the mean time between failure is long enough, so that each request will eventually succeed

SOLUTION 3: FALSE REPLICA-FAILURE DETECTION



SOLUTION 4: CLIENT FAILURE DURING OPERATION



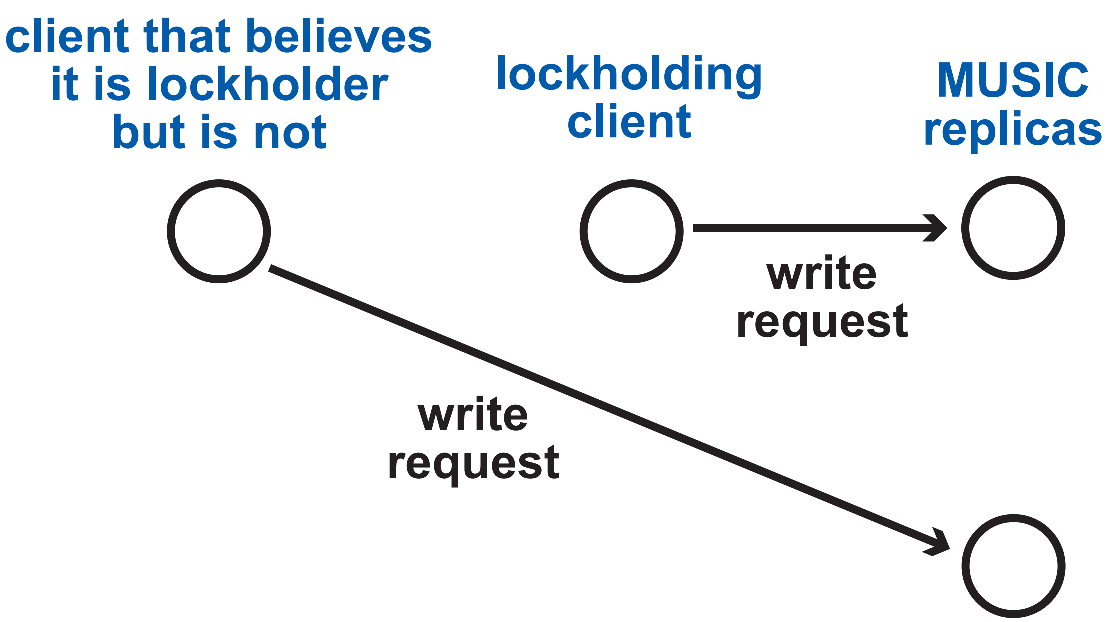
if a MUSIC replica detects that a lockholding client has died, it can forcibly release the lock

this takes care of the lockRef that got on the queue without the knowledge of a client—when it is granted the lock, a MUSIC replica will forcibly release the lock

SOLUTION 5: FALSE CLIENT-FAILURE DETECTION

PREEMPTS THE LOCK OF A LIVE LOCKHOLDER!

This means that multiple clients might simultaneously believe that they hold the lock.



PROPERTIES

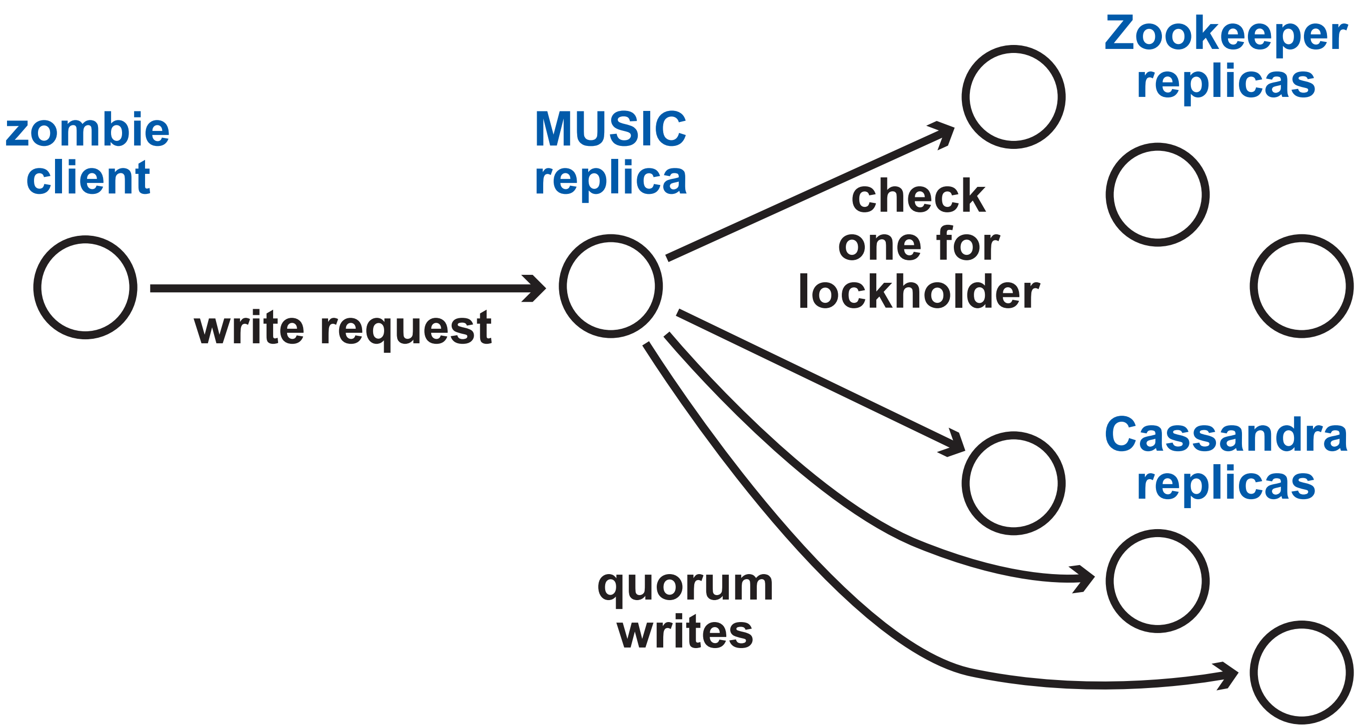
- if a client write operation begins and ends while the client is the lockholder, it succeeds
- if a client write operation begins after the client is the lockholder, it fails
- if a client write operation begins while the client is the lockholder and ends after, then either outcome is acceptable
- if a client continues to request reads and writes when it is not lockholder, it will eventually be told it is not lockholder

SOLUTION 6: MANAGING ZOMBIE CLIENTS

timestamps in the data store have the form (lockRef, time)

higher-order part; lockRefs have temporal order

when a new lock is acquired, MUSIC refreshes a quorum of data values with the new lockRef in its timestamp



if client is no longer the lockholder, eventually the replica will be up-to-date and know this

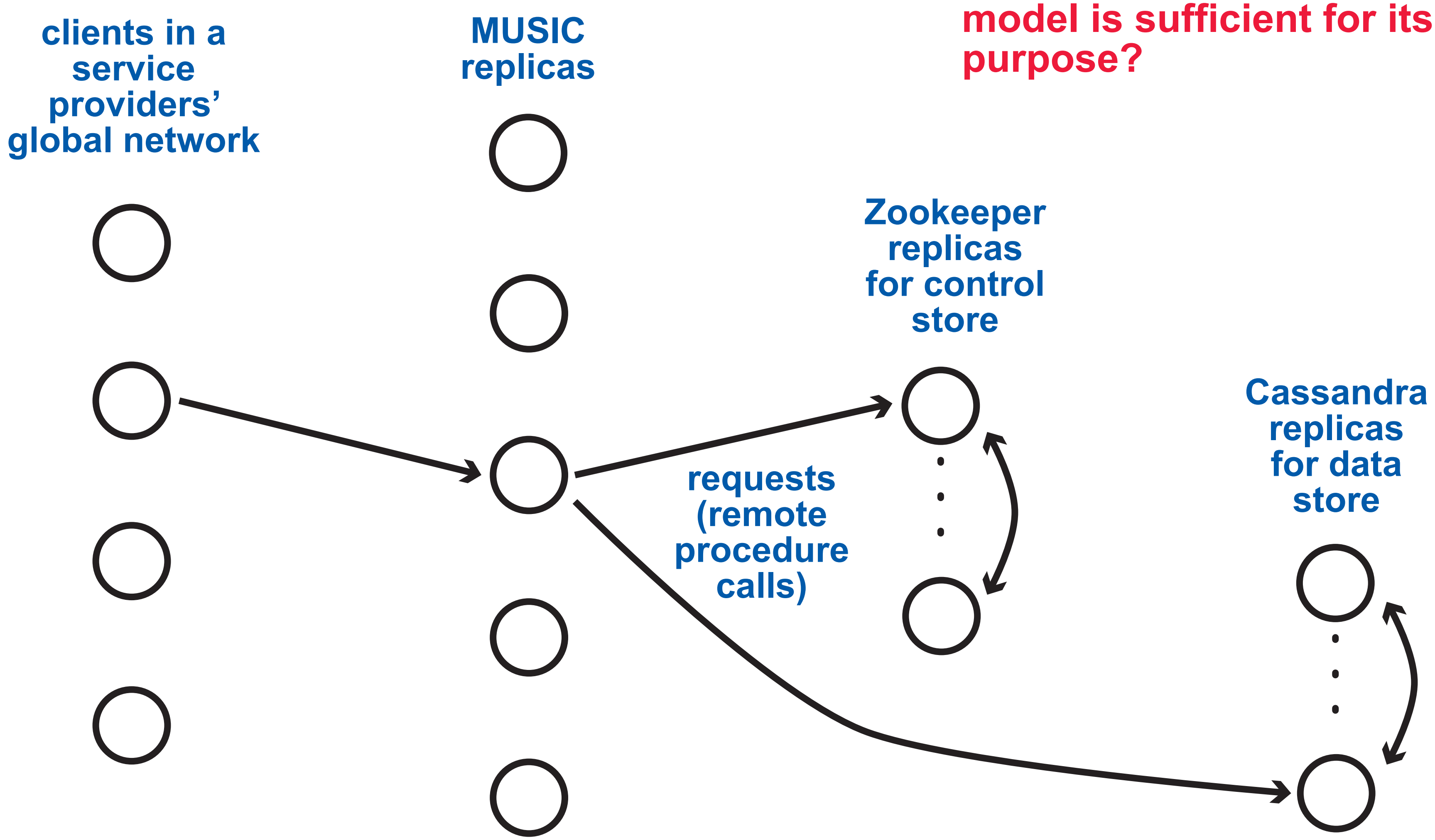
if client is no longer lockholder, at least one of these replicas will ignore the write because of its old timestamp

THE PROBLEMS

- too much concurrency!
- too much inter-dependent distributed state!
- too many failure possibilities, and detection has false positives!

I have an Alloy model that satisfies assertions, and that I can explain and debug—just barely.

How do I reason that this model is sufficient for its purpose?



EVENTS OF THE MODEL

ZOOKEEPER-ONLY EVENTS

propagate more-recent data

CASSANDRA-ONLY EVENTS

propagate more-recent data

MUSIC-ONLY EVENTS

replica failure

replica restart

CLIENT-ONLY EVENTS

failure

restart

all of these change the state of one node only

MUSIC/ZOOKEEPER EVENTS

forced release of lock

CLIENT/MUSIC/ZOOKEEPER EVENTS

enqueue lockref

release lock

critical write accept

critical write reject

these just do a single Zookeeper read (critical write accept, critical write reject)

CLIENT/MUSIC/CASSANDRA EVENTS

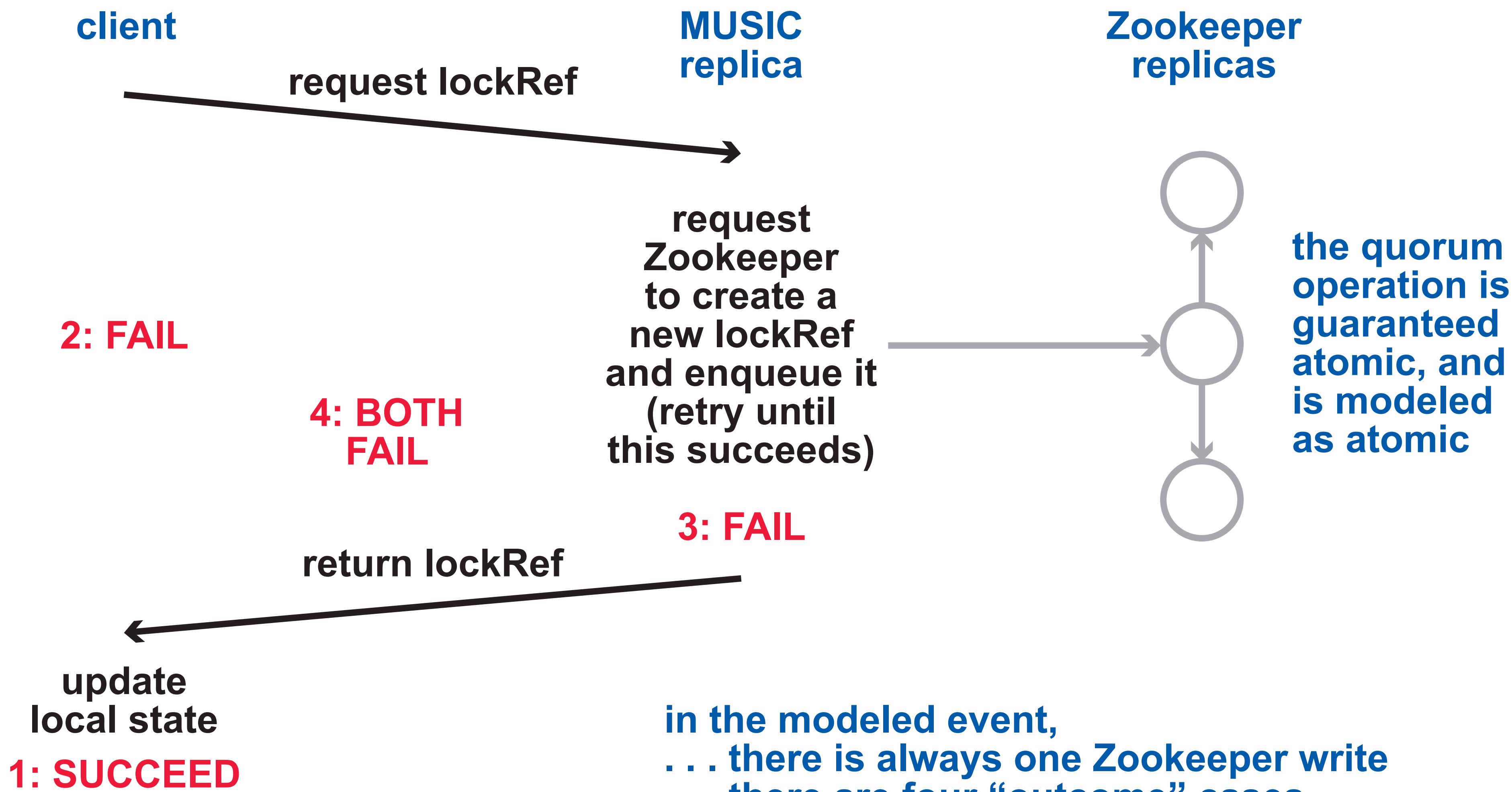
acquire lock

critical write one

critical write finish

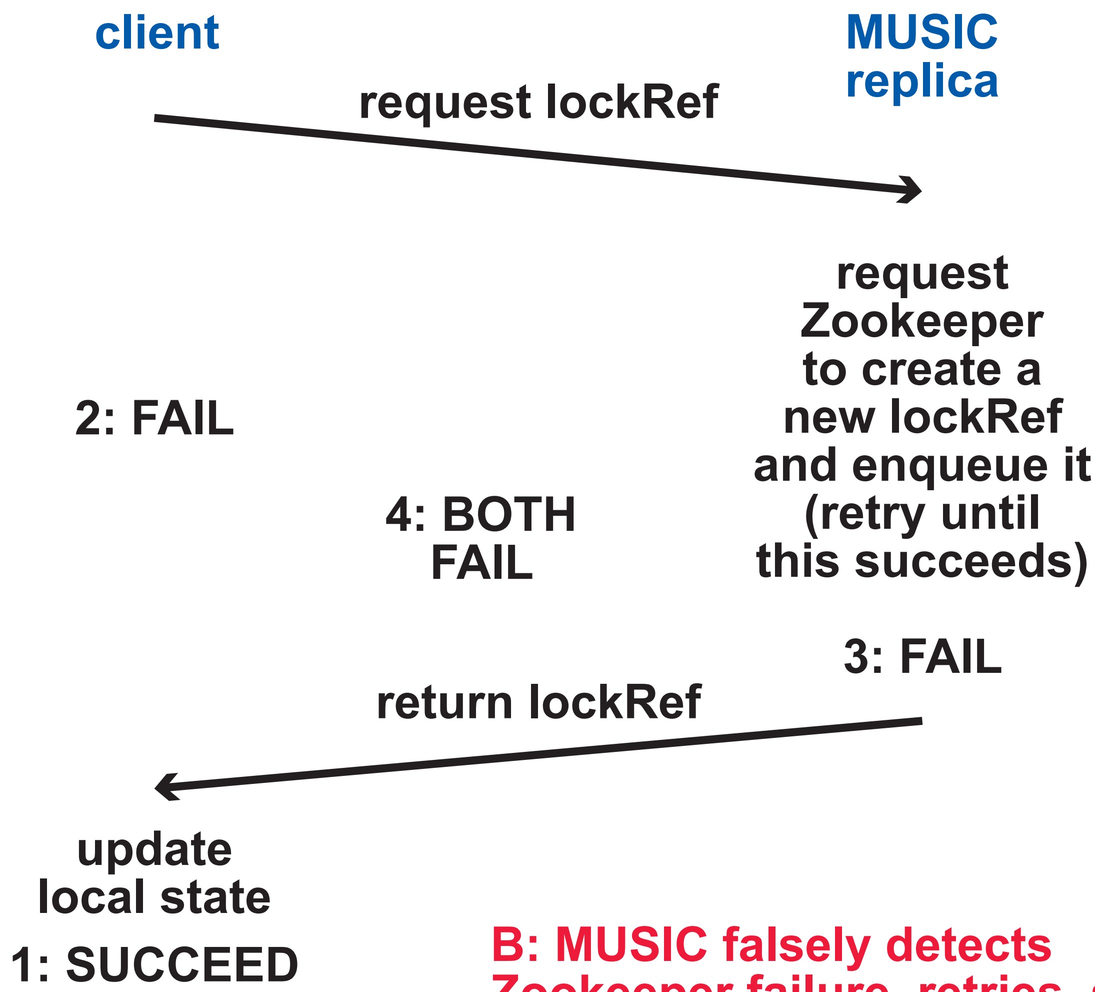
change state of one Cassandra replica (critical write one, critical write finish)

THE ENQUEUE OPERATION MODELED AS ONE EVENT



in the modeled event,
. . . there is always one Zookeeper write
. . . there are four “outcome” cases

THE ENQUEUE OPERATION: WHAT IS MISSING?



A: the Zookeeper write never happens because it is unavailable even after retries; MUSIC reports this to client

Assume this is same as . . .

	client	MUSIC
1:	no-op	no-op
2:	isolated failure	no-op
3:	no-op	isolated failure
4:	isolated failure	isolated failure

B: MUSIC falsely detects Zookeeper failure, retries, so there are two Zookeeper writes

Assume this is same as outcome 4, where a lockRef is enqueued that no client knows about

C: client falsely detects MUSIC failure, retries, so there are two requests

Assume this is also the same as outcome 4 (plus restart after both failures)

CRITICAL WRITE ONE MODELED AS ONE EVENT

client

waiting for critical write to complete

MUSIC replica

waiting for Cassandra quorum write to complete

Cassandra replicas

another Cassandra replica responds to write request— state changes if and only if write request has a more recent timestamp

2: FAIL WHILE CLIENT WAITING (client goes to retry state)

history variable storing which Cassandra replicas have been contacted is updated

1: SUCCEED

3: FAIL AFTER CLIENT HAS DIED

CRITICAL WRITE ONE: WHAT IS MISSING?

client

waiting for critical write to complete

2: FAIL WHILE CLIENT WAITING (client goes to retry state)

A: client fails during this operation

Assume this is same as isolated client failure before or after

3: FAIL AFTER CLIENT HAS DIED

MUSIC replica

waiting for Cassandra quorum write to complete

history variable storing which Cassandra replicas have been contacted is updated

1: SUCCEED

Cassandra replicas

another Cassandra replica responds to write request—state changes if and only if write request has a more recent timestamp

B: client or MUSIC falsely detect MUSIC or Cassandra failure, retry writing the same value

Assume that another write of same value with later timestamp is idempotent.

OTHER SHORTCUTS

CONNECTIONS

- do not model network connections
- so do not model the situation in which a replica is reachable from one place and not another
- since failures do not affect data, seems to be covered by scenarios in which replicas fail or restart quickly between nearly-simultaneous communication attempts

CRITICAL WRITE

- although a critical write can continue while its client fails, restarts, and gets a new lockRef, it cannot continue while client acquires the lock

no reason to believe this would be a problem, but it would expand the system state

OVERVIEW

- 14 events, each with correctness assertions:

  . . . invariant is preserved

  . . . "resilience": a client operation either succeeds,
  results in a dead client,
  or results in a client with operation precondition enabled

  . . . critical write has the correct result (depending on client status)

- 31 event cases (each should have an instantiating predicate, but not all do)

- 37 constraints in the invariant

- scope is 3 clients, 4 lockRefs, 5 MUSIC replicas, 5 Zookeeper replicas,

  5 Cassandra replicas

- about 1200 lines of Alloy code

DISCUSSION QUESTIONS

IS THE MODEL USEFUL?

not useful, indispensable

IS THE ALGORITHM "VERIFIED"?

community standards
vary greatly

IS THERE A MORE SYSTEMATIC WAY TO REASON ABOUT THE SCENARIOS THAT ARE NOT MODELED?

major expansion of the model
is not an option