MUSIC: Multi-site State Coordination for Distributed Services

Bharath Balasubramanian, Pamela Zave, Kaustubh Joshi, Shankar Narayanan,
Gueyoung Jung, Matti Hiltunen and Richard Schlichting
Cloud Software Research, AT&T Labs-Research

Multi-site replicated services

Most services (storage, compute etc) replicated across sites/dc for:

Reliability

Availability

Locality

Need for Coordination

Complex replicated services often need *coordination.* Examples:

How will I ensure only one of the replicas is active?

On failure, will a new active have up to date state?

How can I synchronize state across replicas?

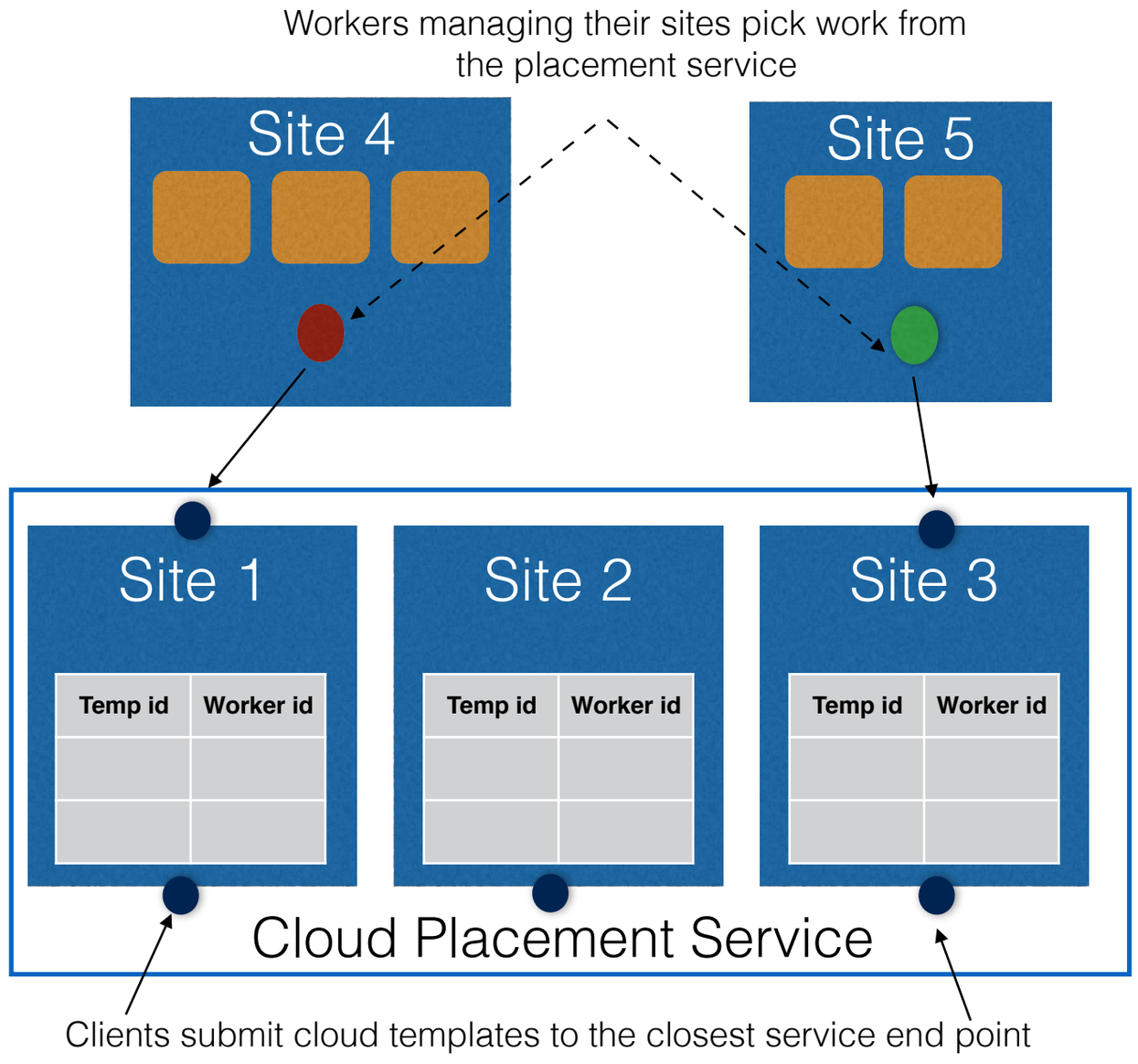
How do I ensure exclusive access to shared state among several active replicas?

ATT Use-case 1: Multi-site Placement Service

- Cloud placement service replicated across sites.

- Clients submit app templates to nearest replica and site-workers pick these templates and place them if they have resources.

- *Need for coordination*: Ensure that each template is picked up by only one worker



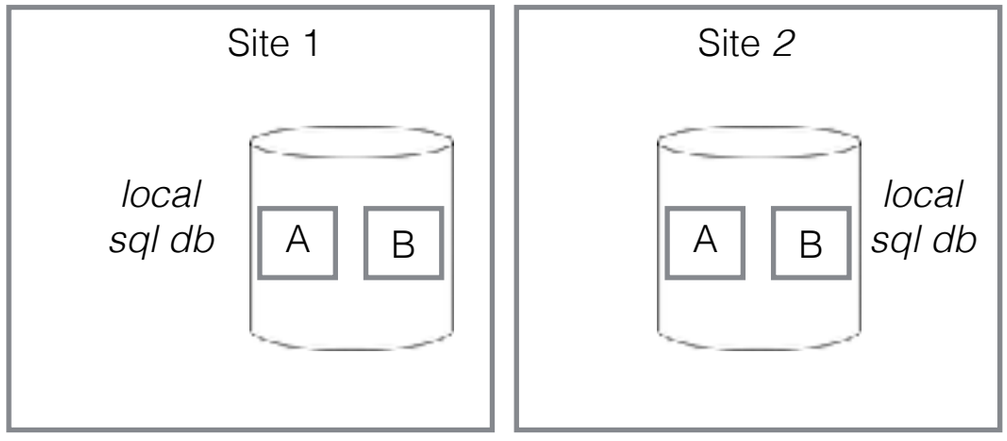
ATT Use-case 2: Multi-site DB Cache

- Many applications need databases for transactionality and complex queries and joins

- But what about multi-site distributed set-ups?

  - Lazy asynchronous replication causes correctness issues while synchronous replication can cause performance issues across the WAN

- *Need for coordination*: can we get transactionality within sites but flexible mirroring options across?



Current Approaches

1. Maintain state in eventually-consistent stores like Cassandra or MongoDB
    - Eventual consistency can cause *correctness issues*. e.g. same template picked up by multiple workers, e.g. media server has stale view of a call.
2. Maintain state in a strongly-consistent store like Zookeeper, etcd or Consul
    - Strong consistency on *each* write is *expensive and partition-intolerant* across WAN. e.g. client submitting template does not need strong consistency.

Problem

No existing coordination service for managing access to logically shared state that scales for multi-site replicated services.

Concurrent systems do it…

A rich set of primitives such as semaphores, mutexes, and barriers have evolved over time to enable coordination in multi-threaded systems.

For distributed systems?

Analogous primitives for distributed systems such as leader election, mutual exclusion, 2-phase commit typically restricted to use within a site.

The few multi-site solutions are very specific to applications such as quota maintenance/rate limiting.

For distributed systems?

Analogous primitives for distributed systems such as leader election, mutual exclusion, 2-phase commit typically restricted to use within a site.

The few multi-site solutions are very specific to applications such as quota maintenance/rate limiting.

What is the challenge?

Concurrent systems rely on an underlying memory model that is *sequentially consistent* : Each read of a register will see the latest write.

This enables strong coordination patterns in multi-thread systems. E.g. a process in a critical section has exclusive access to the most up-to- date copy of data protected by the critical section.

Sequential Consistency in multi-site distributed systems?

Very hard to achieve in geo-distributed systems with network partitions and high WAN latencies.

CAP theorem (paraphrased): To tolerate network partitions (P), one must choose between sequential consistency (C) or availability (A).

Dilemma

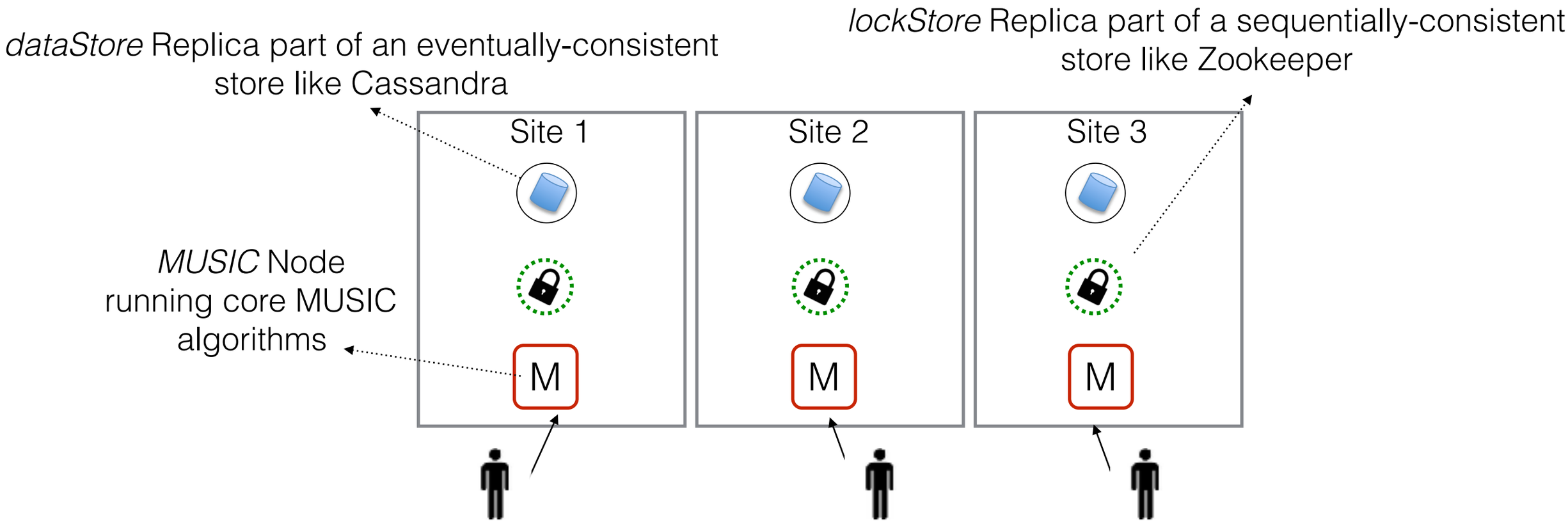
Coordination patterns need sequential consistency.

However, sequential consistency is very hard to achieve in multi-site distributed systems.

Our solution

A multi-site coordination service (MUSIC) that maintains replicated state in a highly scalable (AP) key-value store and explicitly provides a locking service (CP) to protect access to shared state.

Architecture



MUSIC Basic Usage

MUSIC provides the abstraction of a replicated key-value store, where access to the keys can be controlled using locks. To use MUSIC, a client issues a request to a MUSIC node of its choice.

The MUSIC operations are divided among CP and AP operations based on whether they are operations involving a critical section or not respectively.

MUSIC CP Operations

Using locks, a client can access the store in a critical section with respect to one or more keys.

createLockRef takes a set of keys and returns a ***lockRef***, which is a ticket good for one critical section only.

acquireLock (***lockRef***) returns true for only one ***lockRef*** and also ensures that replicas of keys in the key-set have the most recent values.

```
K = {key1, key2};
lockRef = createLockRef (K);
while (acquireLock (lockRef) != true)
      skip;
//critical section
v1 = criticalGet(lockRef, key1);
v1`= v1+1;
criticalPut(lockRef, key1, v1`);
v2 = criticalGet(lockRef, key2);
v2` = v2 * v1`;
criticalPut(lockRef, key2, v2`);
releaseLock(lockRef);
```

MUSIC CP Operations

The lock holder can perform ***criticalGets*** and ***criticalPuts*** that read and write to a majority of MUSIC replicas respectively.

Since the critical operations all require a majority of MUSIC replicas, they are CP operations.

```
K = {key1, key2};
lockRef = createLockRef (K);
while (acquireLock (lockRef) != true)
     skip;
//critical section
v1 = criticalGet(lockRef, key1);
v1`= v1+1;
criticalPut(lockRef, key1, v1`);
v2 = criticalGet(lockRef, key2);
v2` = v2 * v1`;
criticalPut(lockRef, key2, v2`);
releaseLock(lockRef);
```

MUSIC AP Operations

The ***put*** and ***get*** write and read to the key at any of the MUSIC replicas.

While the ***get*** is enabled for all keys, ***puts*** are enabled only for keys on which critical operations will never be attempted.

Since both these operations need just a single MUSIC replica, they are partition-tolerant AP operations.

```
v1 = get(key1);
v1`=v1+1;
put(key1, v1`);
```

MUSIC Properties

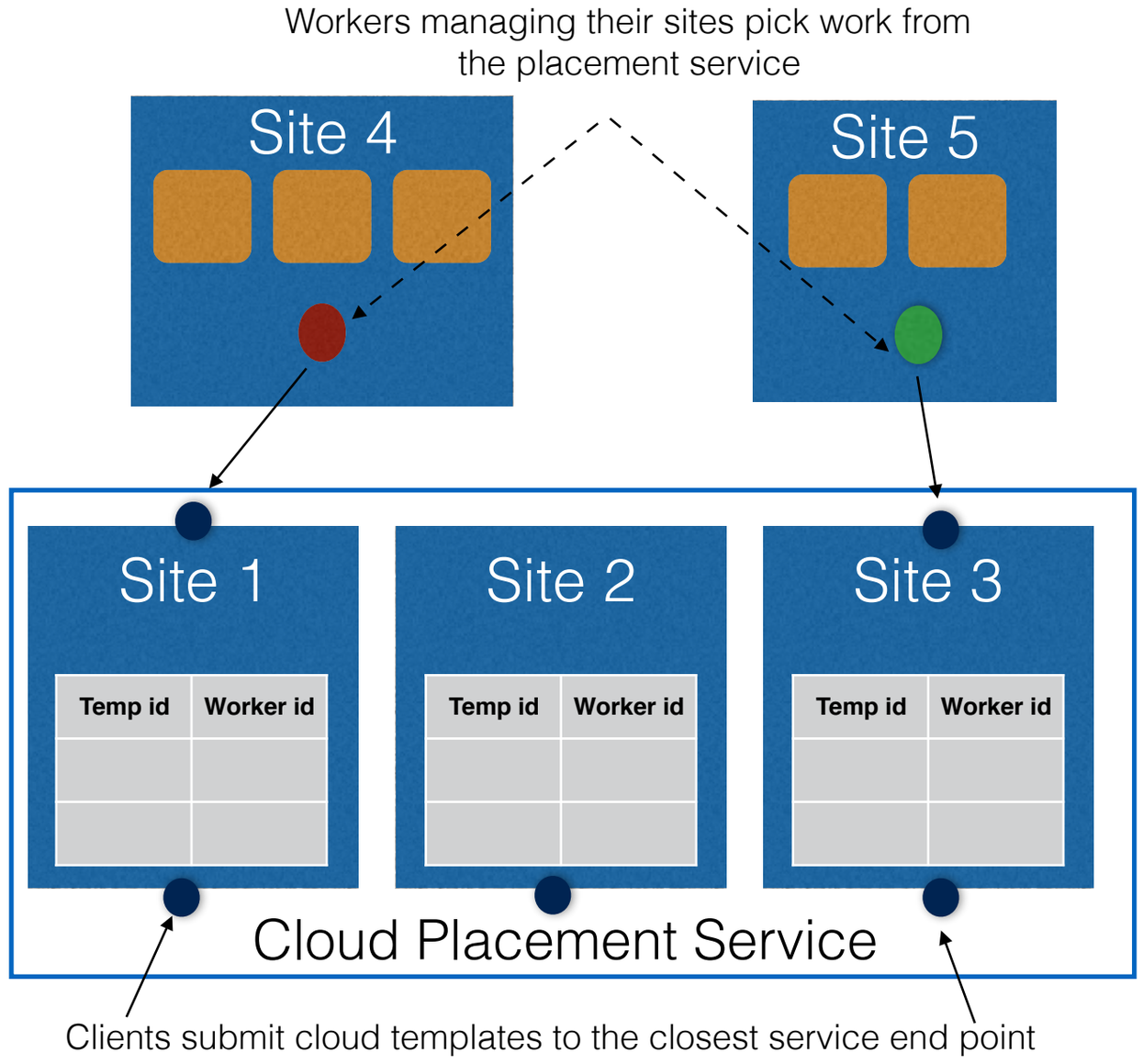
When a client acquires a lock to a set of keys, the client is guaranteed a version that reflects the most recent update to the key.

When a client performs reads and writes to locked keys, the client experiences sequential consistency.

Due to the subtle nature of properties, we are verifying the safety properties formally using the Spin model checker and the Alloy analyzer.

ATT Multi-site Placement Service over MUSIC

- Maintain worker-template mapping in MUSIC
- When a worker wishes to place a template, it firsts acquires a lock to the template and only if it succeeds, updates it status using critical puts and performs the actual placement

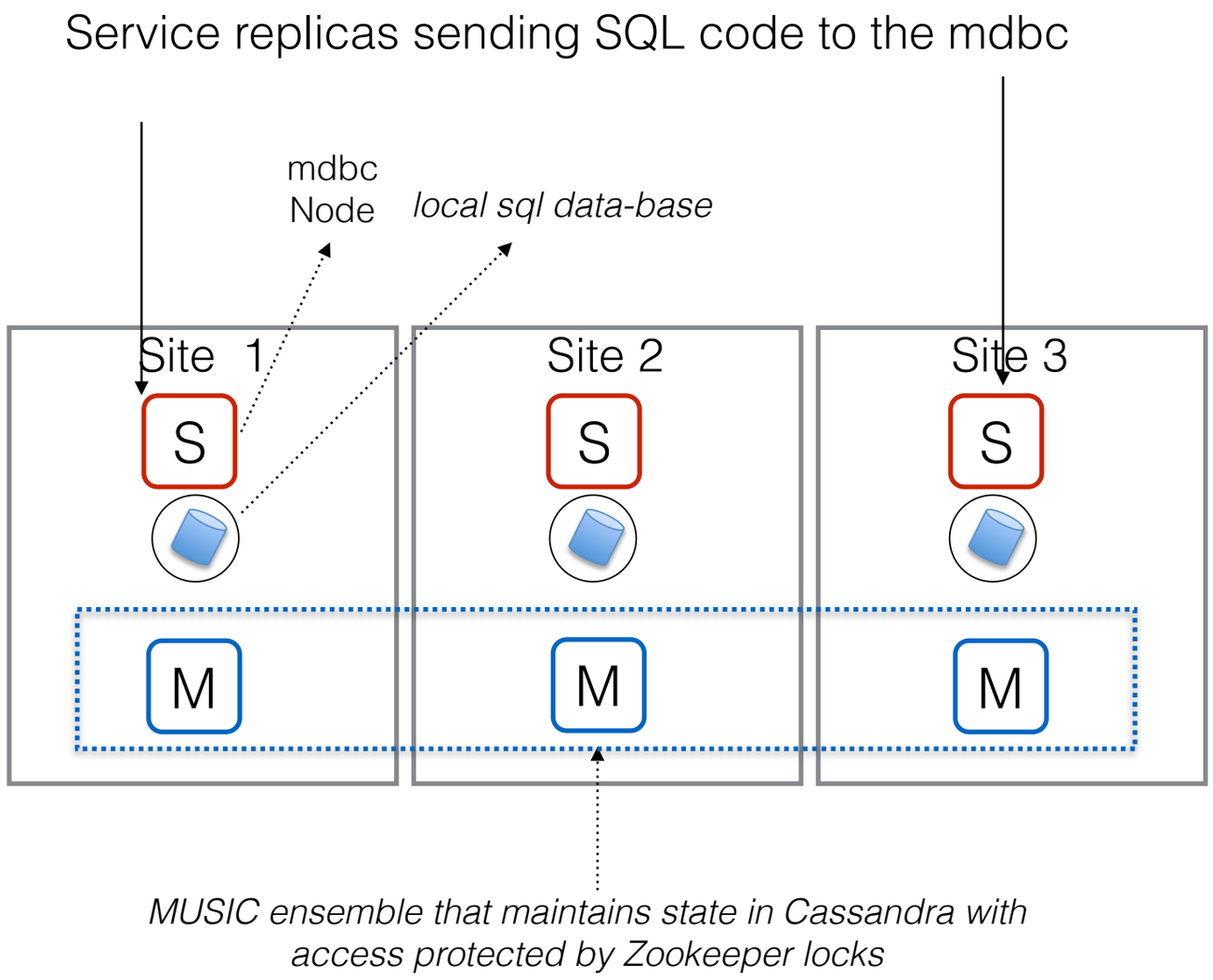


ATT Use-case for a multi-site DB Cache (mdbc) using MUSIC

- mdbc = local sql db + multi-site MUSIC deployment

- Service replicated across multiple sites; writes to and reads from the local mdbc sql database

- mdbc captures local sql writes and propagates it to MUSIC and captures local reads and serves it from MUSIC



Recipes over MUSIC

Multi-site coordination recipes for:

mutual exclusion over shared state

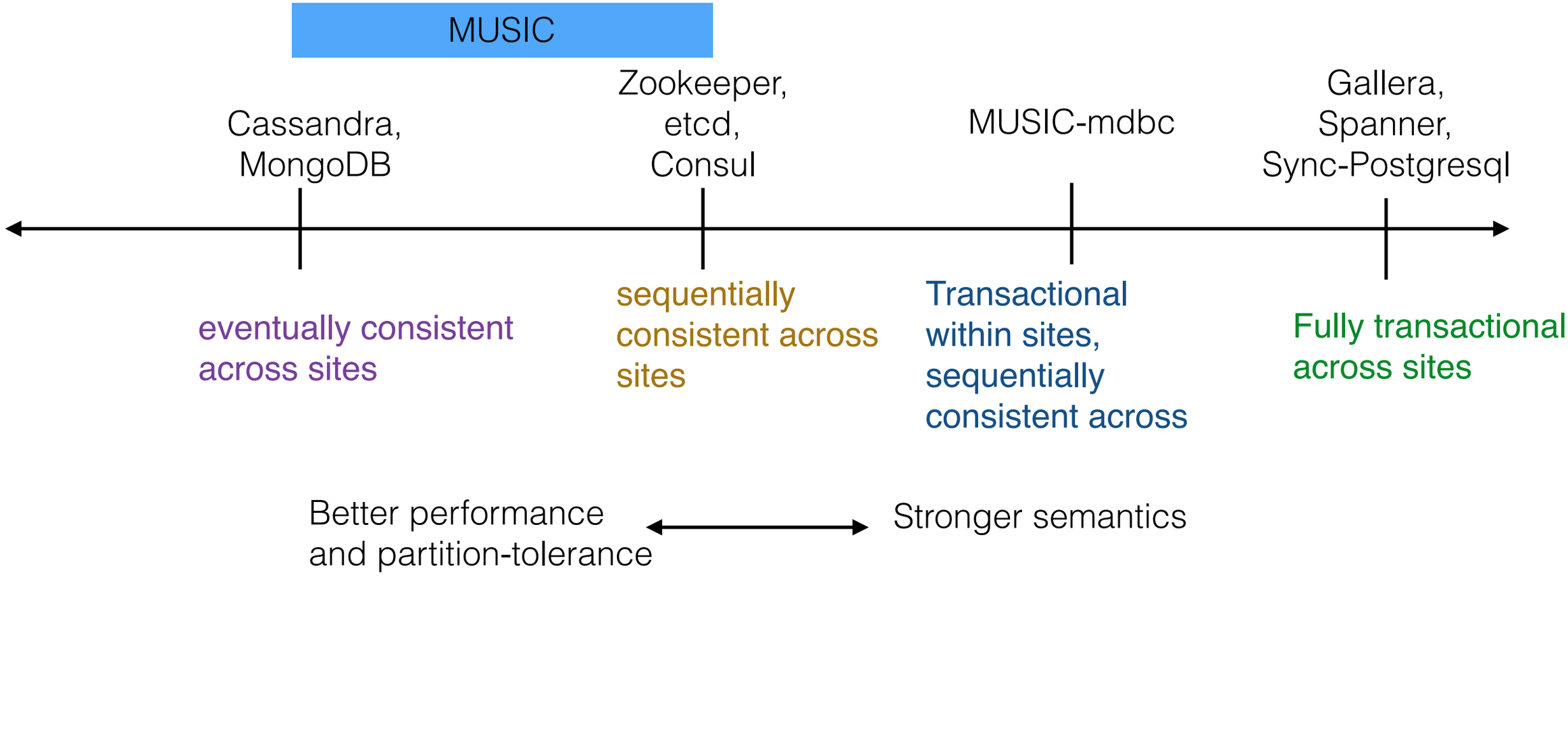
load-balanced active-passive replication

barrier synchronization over distributed state

Stronger data semantics:

Multi-site replicated database cache (mdbc) which allows SQL applications transactional semantics within the site and choice of eventually consistent/ strongly consistent semantics across sites.

MUSIC and other tools



Key Take-Aways

Multi-site coordination is necessary but hard to achieve.

MUSIC abstractions of a key-value store protected by locks enables rich coordination primitives for multi-site replicated services.

It also enables on-demand stronger data semantics on eventually consistent stores.