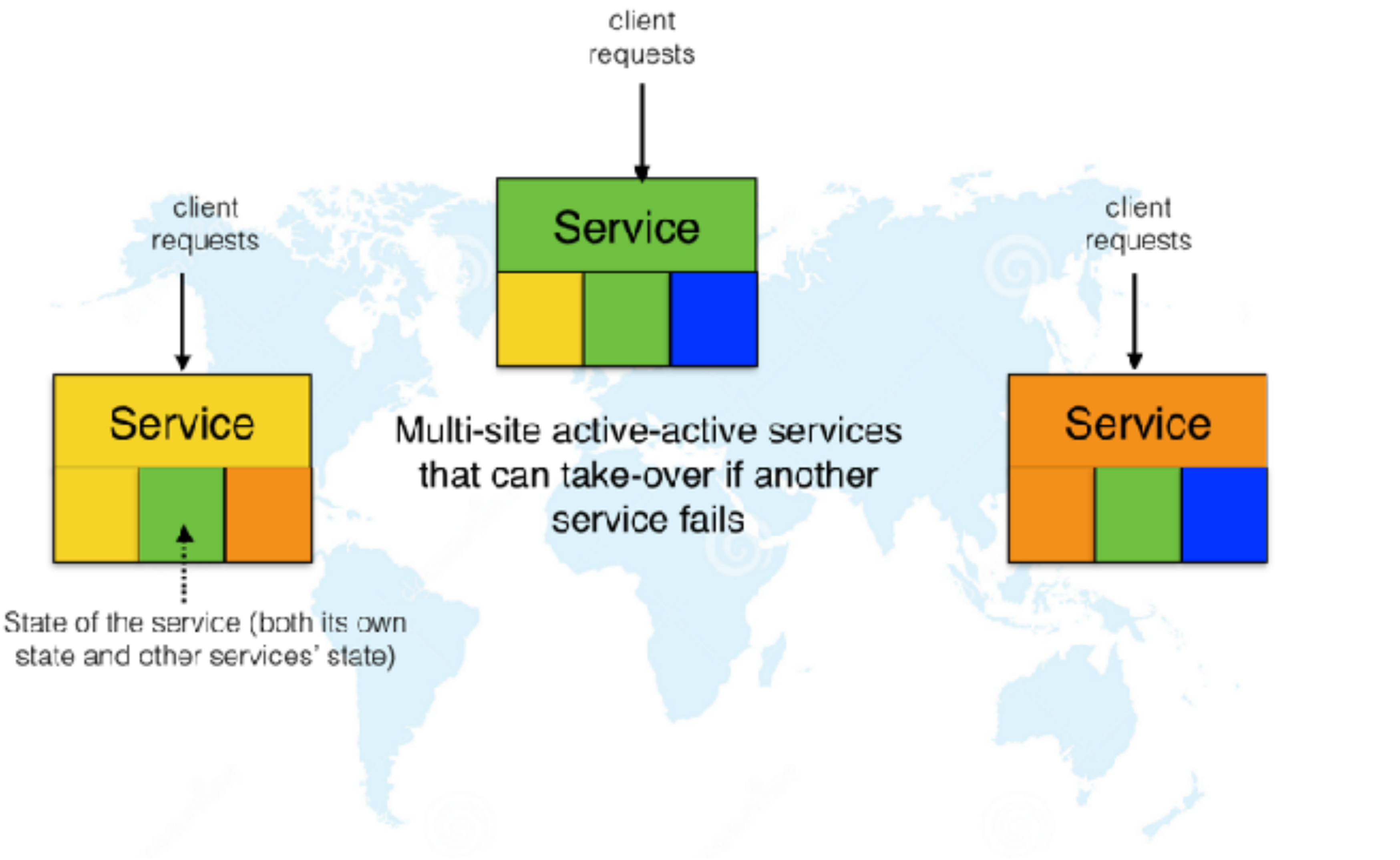
Multi-site State Coordination Service (MUSIC)

Presented by Bharath Balasubramanian,
Network Cloud Infrastructure Research, ATT Labs Research

Goal

A common state-coordination/management platform (MUSIC) to build VNFs with 5 9s of availability on 3 9s (or lower) software and infrastructure in a cost-effective manner.

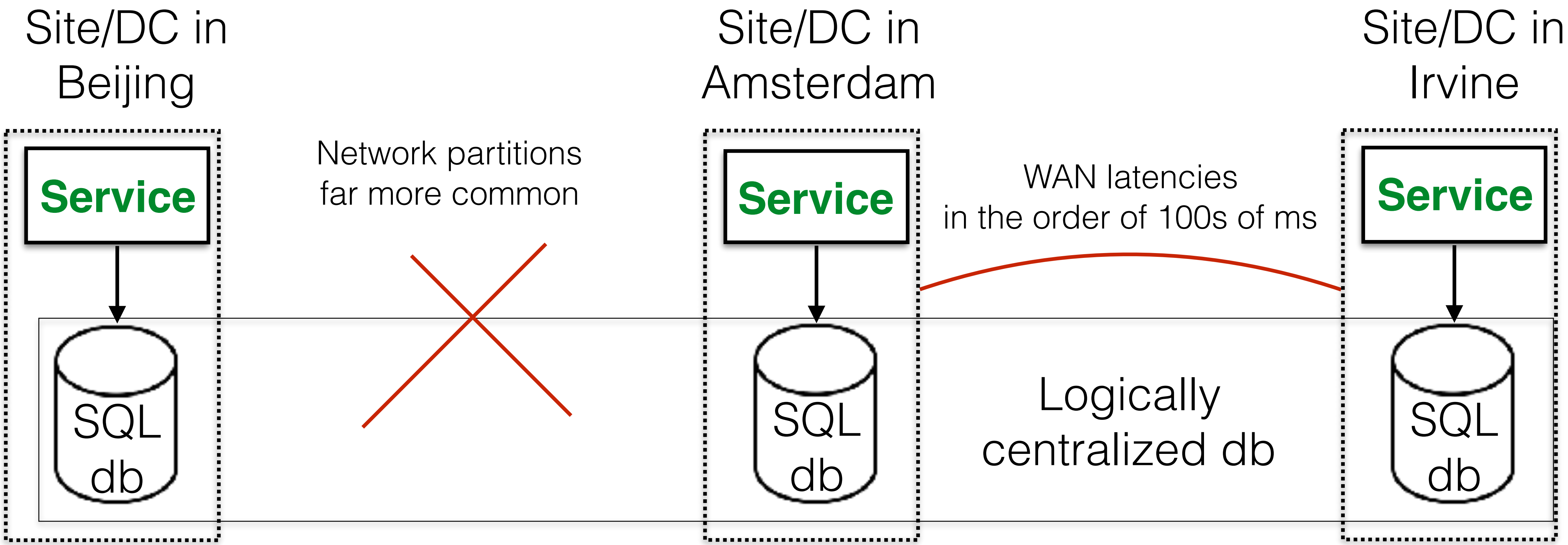
To achieve 5 9s availability Network Services (ONAP, VNFs, Edge/IoT services) need to support multi-site, active-active services with efficient failover.


client requests
Service
client requests
Service
client requests
Service
Multi-site active-active services that can take-over if another service fails
State of the service (both its own state and other services' state)

Need of the hour

- Manage state of services across thousands of geo-distributed sites.
- Provide resiliency/coordination protocols to partition state across replicas and ensure correct and efficient failover of state ownership during site-failures and site-partitions.

Current Practice: Will not scale.



Rooted in the philosophy "the single-site solution should more or less work across geo-distributed sites". E.g. attempts to use mariaDB clustering as is across sites — may not scale and/or allow partitioned operation!

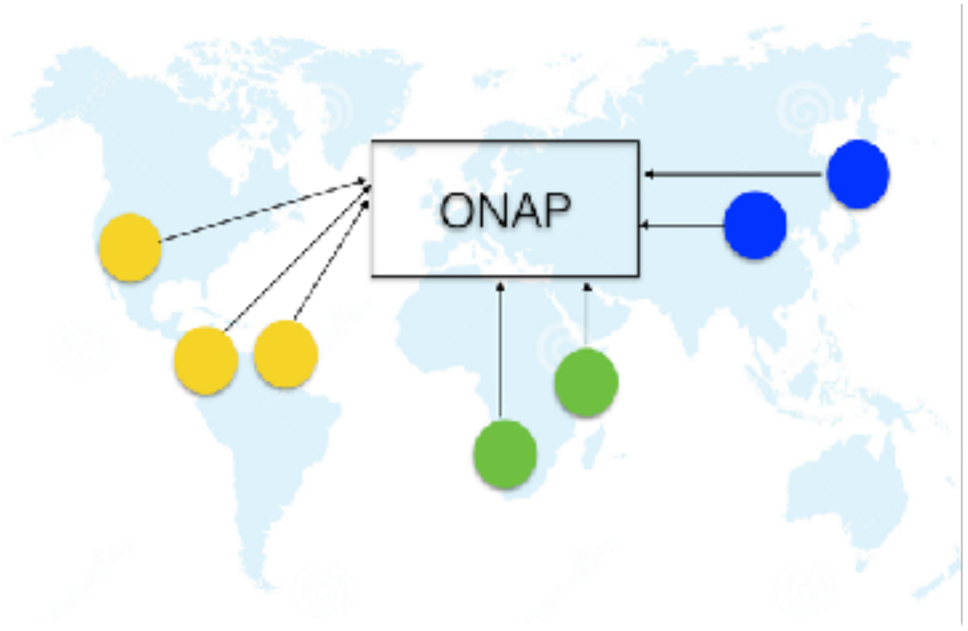
Current Practice: Often wasteful and erroneous.

Each team building its own solution — wasteful and can often be erroneous due to complex distributed protocols, *replete with corner cases*. E.g.

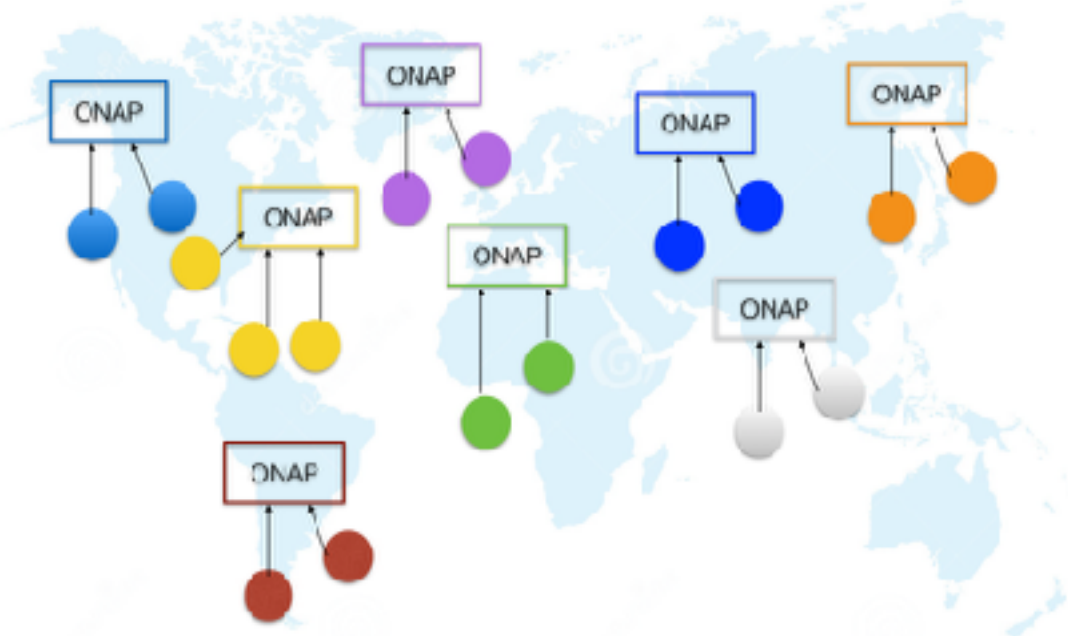
- How do you access state in a mutually exclusive manner when required despite failures/split brain issues?
- Does the new active have the latest information on failover?

Current Practice: Does not address future needs.

Most current designs barely address active-active needs, and mostly ignore *federation* that is crucial for IoT/Edge. E.g. ONAP for Edge/5G and in fact the entire network!



Monolithic design of
ONAP to control VNFs will not scale to the edge.



Federated design of
ONAP to control VNFs is much more practical.

Our solution: MUSIC

- Identify common state management requirements across ONAP components and micro-services.
- Provide a common state-management platform specifically tailored for multi-site geo-distributed replication.
- Provide rich resiliency/coordination recipes on to of MUSIC that ONAP components can simply configure and use.

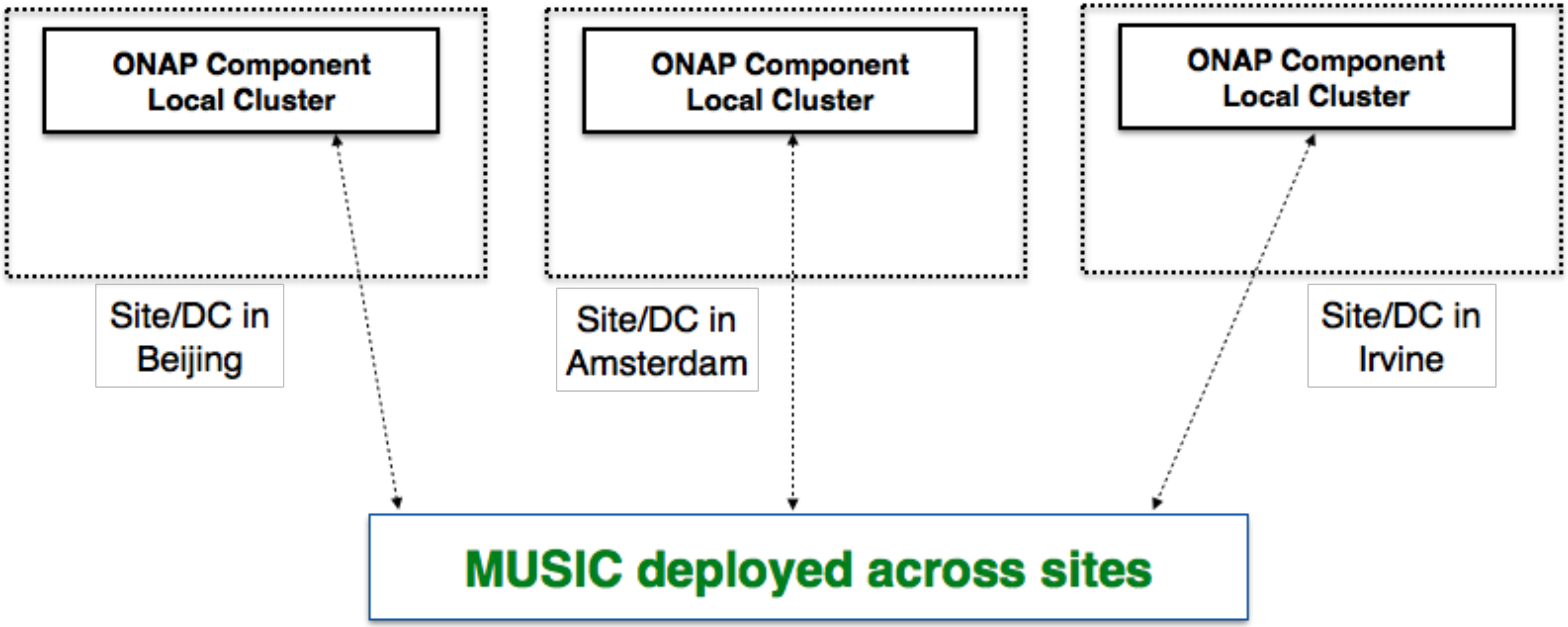
MUSIC Building Blocks

- **MUSIC** provides a sharded eventually-consistent data-store (Cassandra) wherein the access to the keys can be protected using a locking service (Zookeeper)
  - ONAP components can achieve fine-grained flexible consistency on their replicated state across sites.
- Recipes built on top of MUSIC:
  - **mdbc**: A plugin for components to migrate seamlessly from SQL usage to MUSIC
  - **prom**: Policy driven ownership management of state to partition and failover state in a consistent manner
  - **musicCAS**: Distributed compare and set across keys to perform atomic updates
  - **musicQ**: A queue API across sites in which key management is carefully done to ensure efficient sorting

MUSIC Implementation

- Thin shim layer over two production tested open source tools — Apache Zookeeper and Cassandra.
- Less than 10,000 lines of code to obtain a common state-management platform for 5 9s of availability!

Direct Usage


ONAP Component Local Cluster

Site/DC in Beijing

ONAP Component Local Cluster

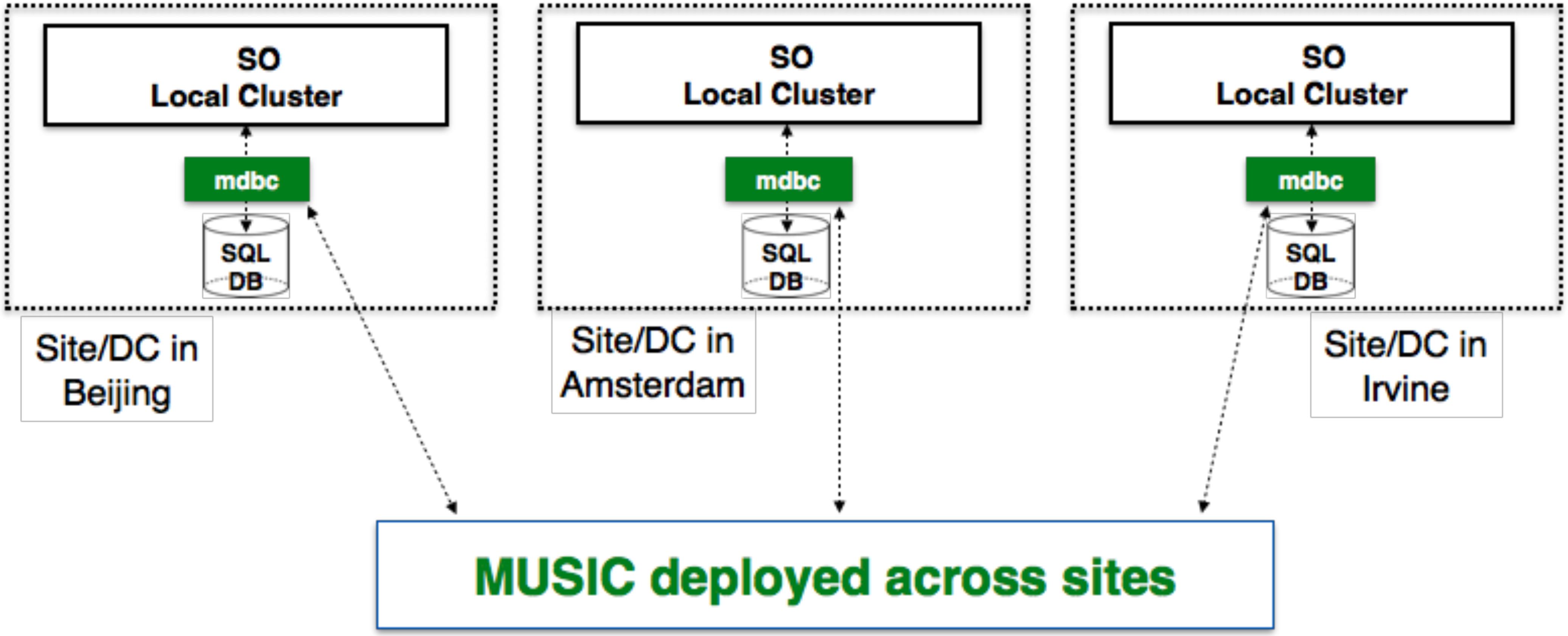
Site/DC in Amsterdam

ONAP Component Local Cluster

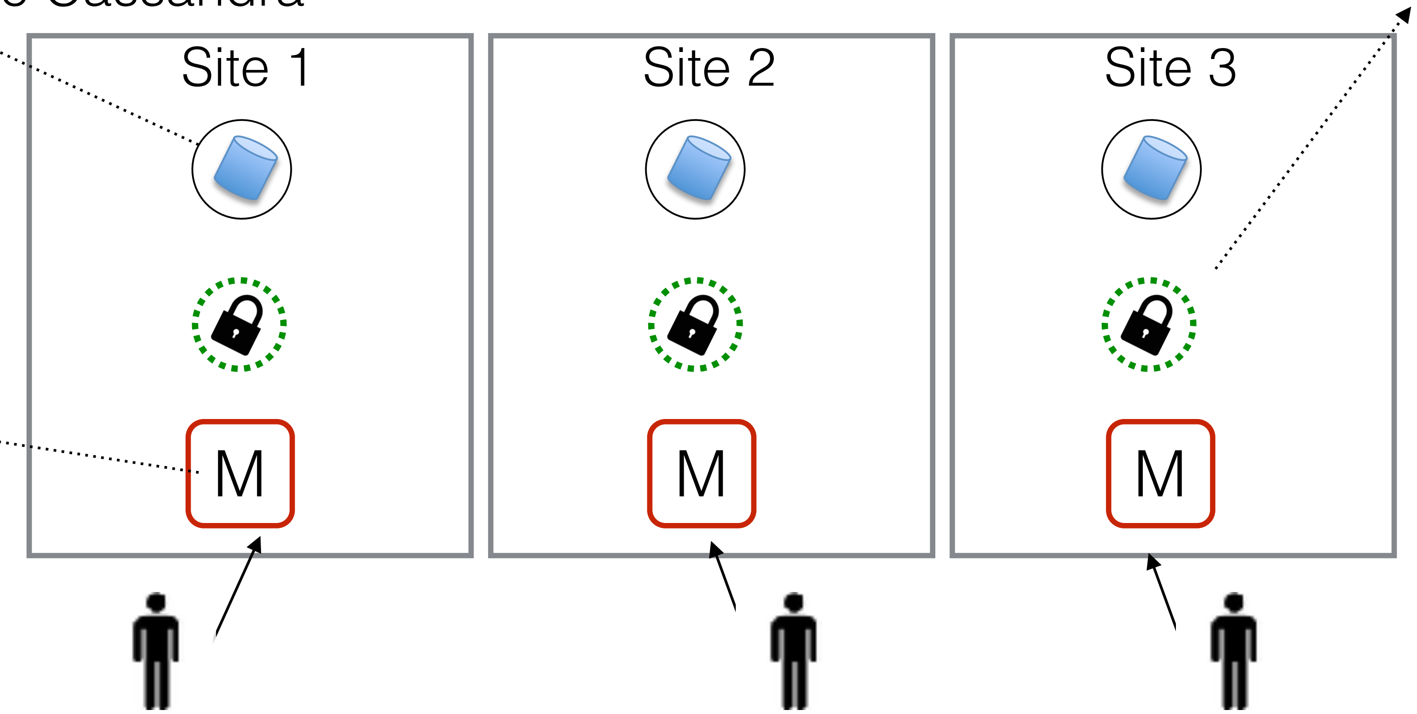
Site/DC in Irvine

MUSIC deployed across sites

For SQL components



MUSIC Architecture



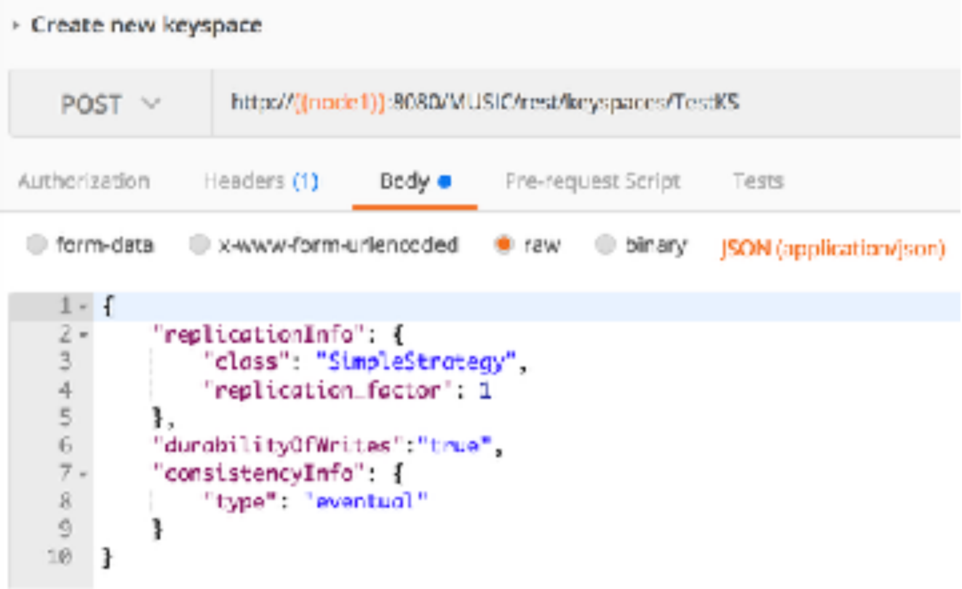
dataStore Replica part of an eventually-consistent
Key-value store like Cassandra

lockStore Replica part of a strongly-consistent
store like Zookeeper

MUSIC Node
running core MUSIC
algorithms

Clients accessing MUSIC through a REST API

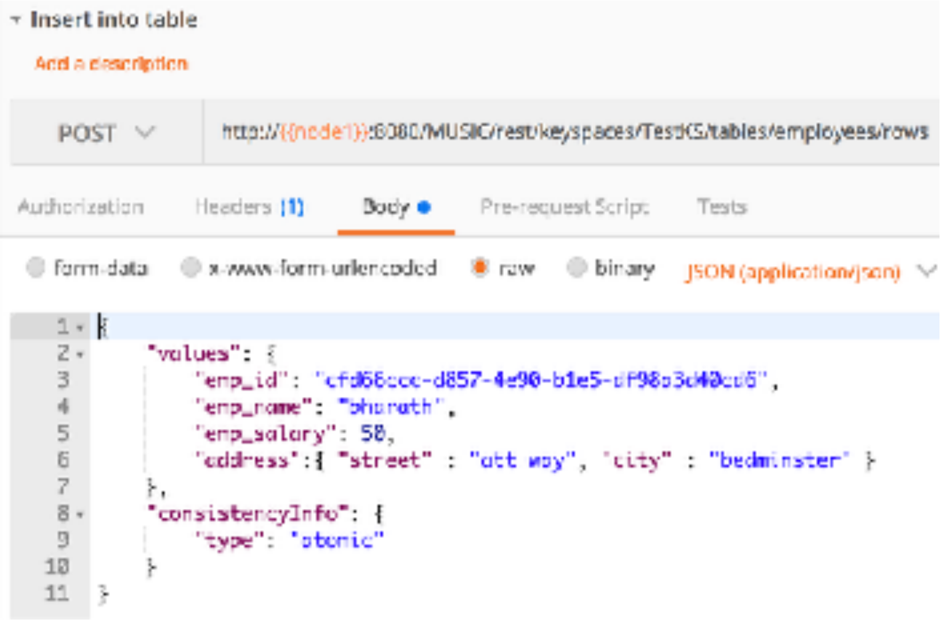
MUSIC Data API



Create new keyspace

POST http://{{node1}}:8080/MUSIC/rest/keyspaces/TestKS

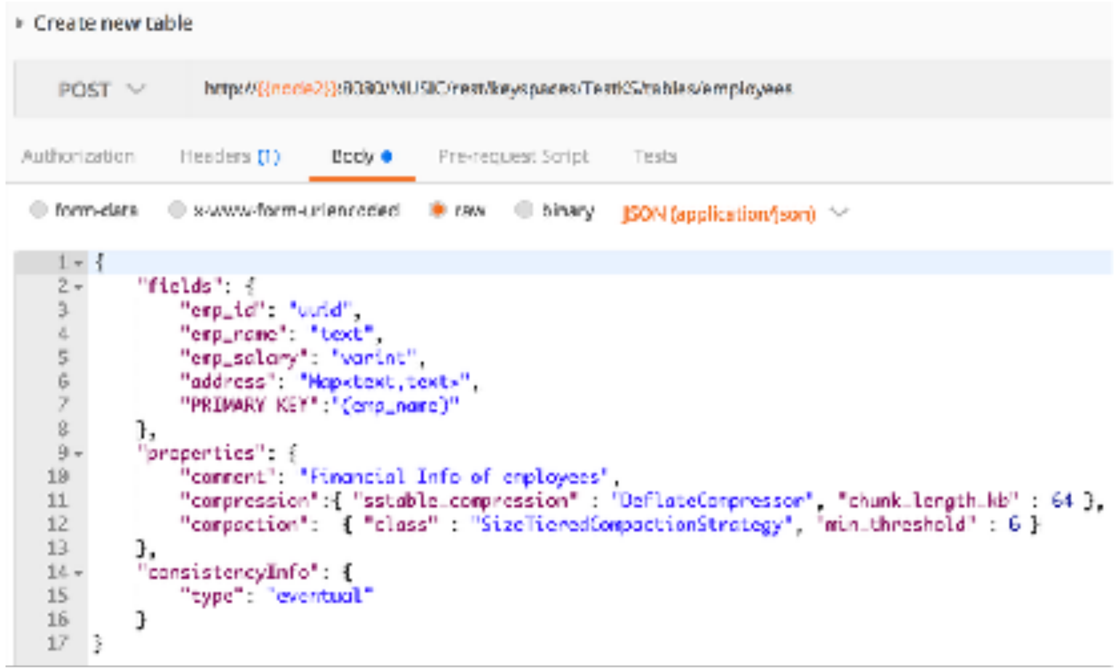
```
{
    "replicationInfo": {
        "class": "SimpleStrategy",
        "replication_factor": 1
    },
    "durabilityOfWrites":"true",
    "consistencyInfo": {
        "type": "eventual"
    }
}
```



Insert into table

POST http://{{node1}}:8080/MUSIC/rest/keyspaces/TestKS/tables/employees/rows

```
{
    "values": {
        "emp_id": "cfd66ccc-d857-4e90-b1e5-df98a3d40cd6",
        "emp_name": "bharath",
        "emp_salary": 50,
        "address":{ "street" : "att way", "city" : "bedminster" }
    },
    "consistencyInfo": {
        "type": "atomic"
    }
}
```



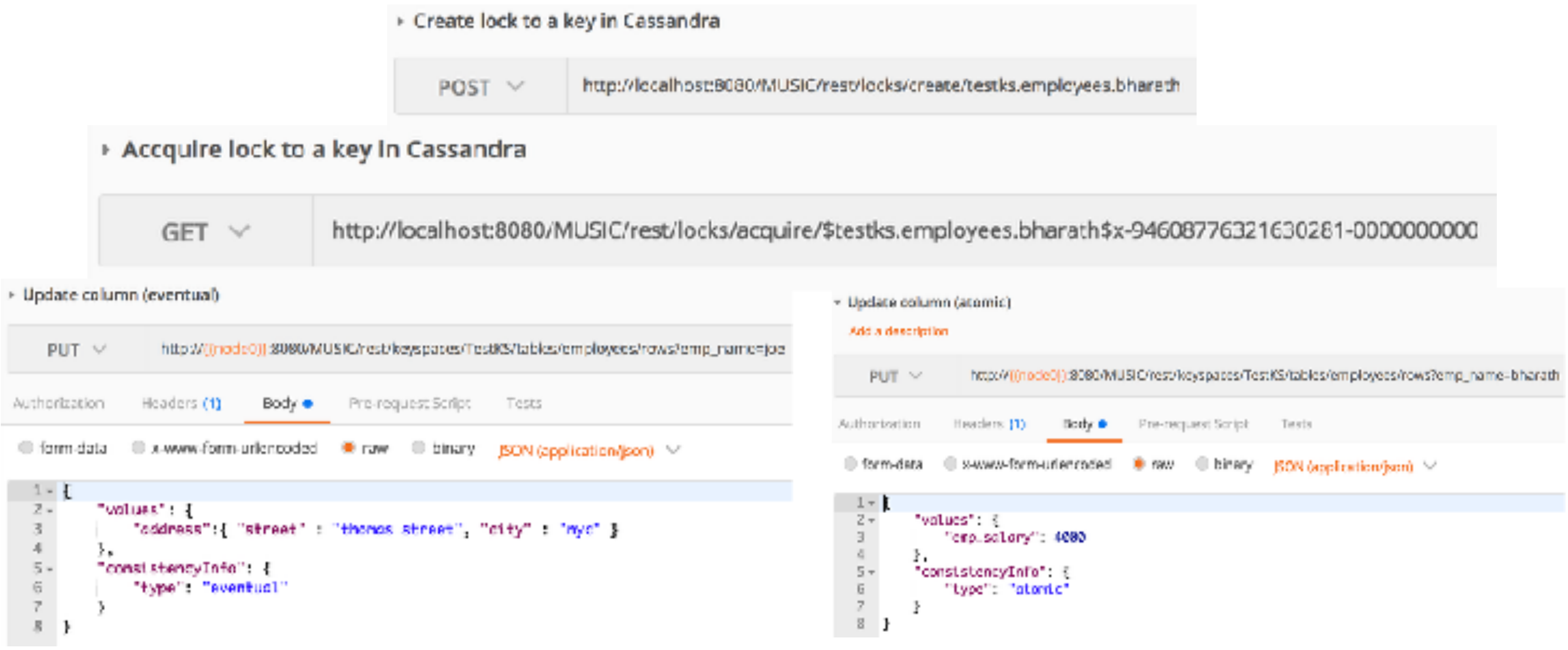
Create new table

POST http://{{node2}}:8080/MUSIC/rest/keyspaces/TestKS/tables/employees

```
{
    "fields": {
        "emp_id": "uuid",
        "emp_name": "text",
        "emp_salary": "varint",
        "address": "Map<text,text>",
        "PRIMARY KEY":"(emp_name)"
    },
    "properties": {
        "comment": "Financial Info of employees",
        "compression":{ "sstable_compression" : "DeflateCompressor", "chunk_length_kb" : 64 },
        "compaction": { "class" : "SizeTieredCompactionStrategy", "min_threshold" : 6 }
    },
    "consistencyInfo": {
        "type": "eventual"
    }
}
```

The basic REST API provides a REST+JSON wrapper around the standard Cassandra API. While this is useful in itself, there is more…

MUSIC Locking API



Create lock to a key in Cassandra

POST http://localhost:8080/MUSIC/rest/locks/create/testks.employees.bharath

Accquire lock to a key in Cassandra

GET http://localhost:8080/MUSIC/rest/locks/acquire/$testks.employees.bharath$x-94608776321630281-0000000000

Update column (eventual)

PUT http://{{node0}}:8080/MUSIC/rest/keyspaces/TestKS/tables/employees/rows?emp_name=joe

```
{
    "values": {
        "address":{ "street" : "thomas street", "city" : "nyc" }
    },
    "consistencyInfo": {
        "type": "eventual"
    }
}
```

Update column (atomic)

Add a description

PUT http://{{node0}}:8080/MUSIC/rest/keyspaces/TestKS/tables/employees/rows?emp_name=bharath

```
{
    "values": {
        "emp_salary": 4000
    },
    "consistencyInfo": {
        "type": "atomic"
    }
}
```

The novel locking API allows the client to create locks on keys (that guarantees mutually exclusive access). Further, the client can chose between eventual operations (no locks) and atomic operations on keys (uses locks in a critical section)!

Example Usage: ONAP HAS

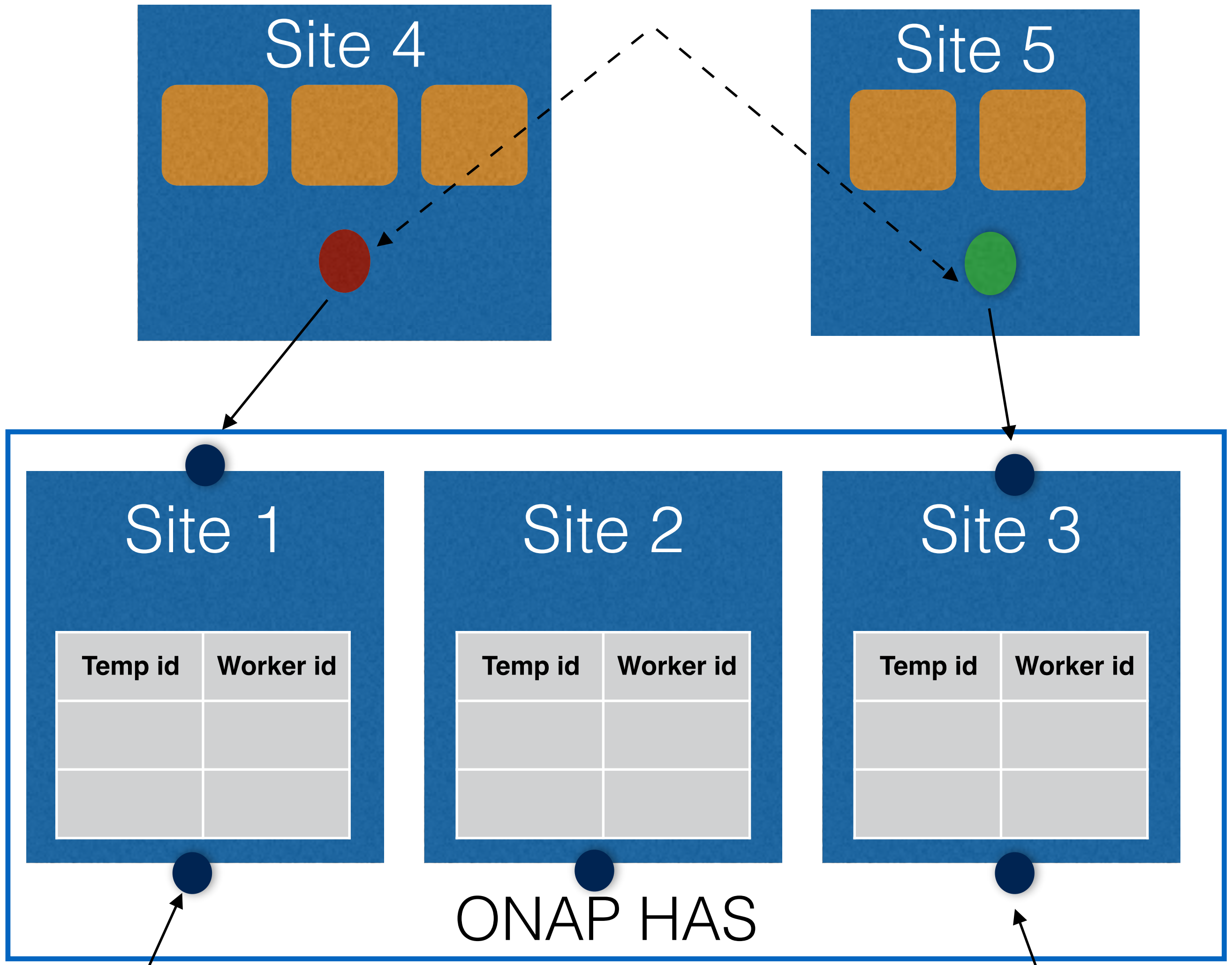
- Based on ONAP homing service (HAS) that is production within ATT

- Cloud homing service replicated across sites.

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

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

Workers managing their sites pick work from the placement service

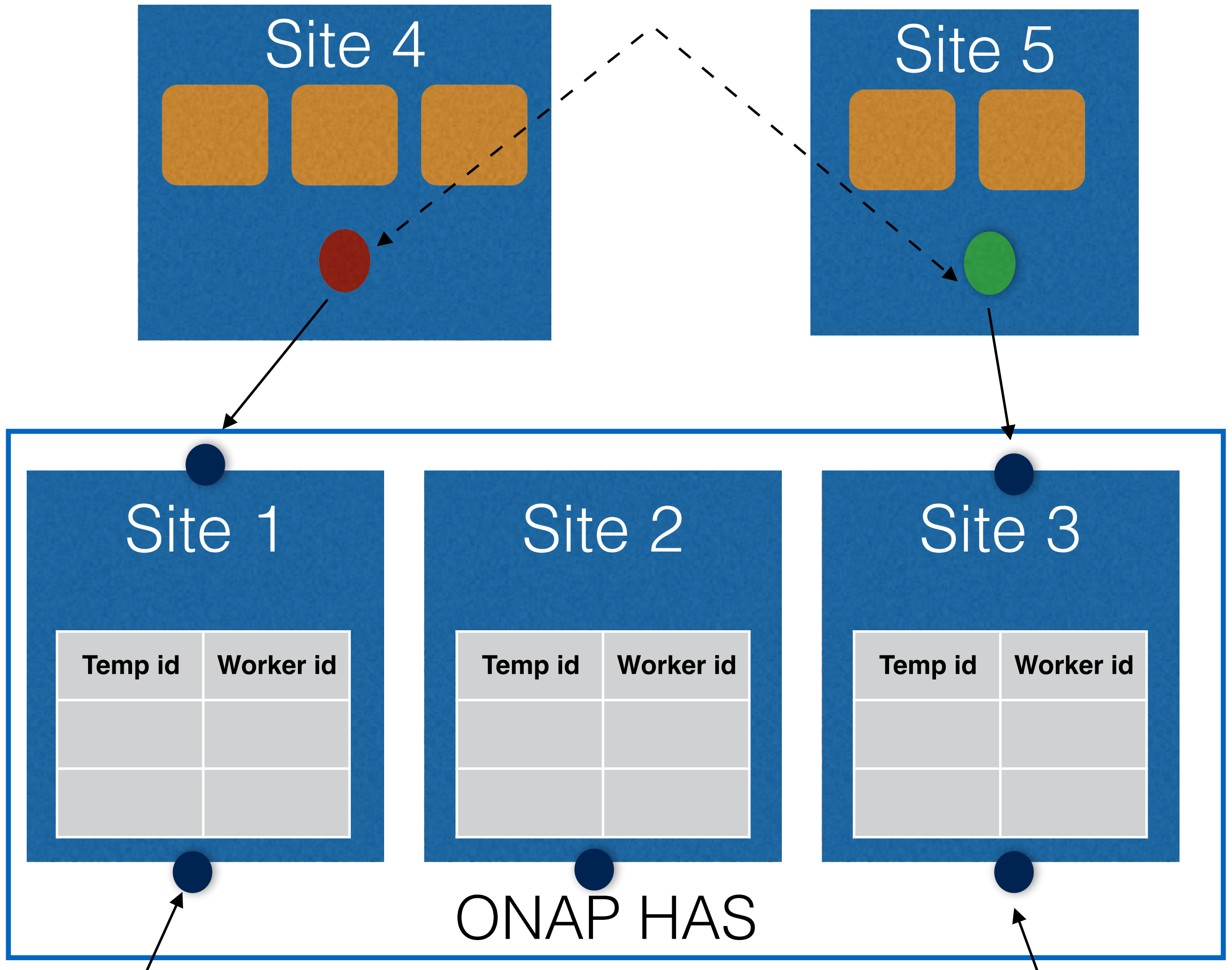


Clients submit cloud templates to the closest service end point

Example Usage: ONAP HAS

- Maintain worker-template mapping in MUSIC
- Clients submit templates using eventual insert/update operations
- 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 atomic operations and performs the actual placement

Workers managing their sites pick work from the placement service



Site 4

Site 5

Site 1

Site 2

Site 3

Temp id	Worker id

ONAP HAS

Clients submit cloud templates to the closest service end point