Sailing the Database Seas: Applying SRE principles at scale

SRECon EMEA
Tuesday 29 October 2024, Dublin

Booking.com

martin.alderete@booking.com
ioannis.androulidakis@booking.com

$ whoami

- Ioannis Androulidakis (**@ioandr**)
- Site Reliability Engineer at Booking.com
- Database Engineering Team, Application Data Services
- Diploma in Electrical & Computer Engineering, NTUA
- Operating Systems, Storage, Observability
- Linux enthusiast, OSS contributor

$ whoami

- Martin Alderete (**@malderete**)
- Principal Site Reliability Engineer at Booking.com
- Database Engineering Team, Application Data Services
- 15+ years of experience in Distributed Systems, System-level Programming and Site Reliability
- Passionate about OSS (also contributor), member of different technical groups

The Journey

1. Database Reliability at Booking.com
2. SLIs and SLOs for Distributed Database Systems
3. Automating MySQL Capacity Planning
4. Postmortem Culture
5. Q/A

The Journey

1. **Database Reliability at Booking.com**
2. SLIs and SLOs for Distributed Database Systems
3. Automating MySQL Capacity Planning
4. Postmortem Culture
5. Q/A

Databases at Booking.com

MySQL is our main relational datastore

Single primary for WRITEs, **tens/hundreds** of replicas for READs
In general, choose **availability** over consistency [1]

MySQL clusters
> 250

MySQL servers
> 10K

QPS
100M+

Services
Transactions, Fintech, Payments, Frontend, Customer Support

Multi-cloud

Our fleet spans across different platforms:

- Bare-Metal (ServerDB)
- Private Cloud (Openstack)
- Public Cloud (AWS)
  - Self-managed (EC2)
  - Managed (RDS)

Multi-region

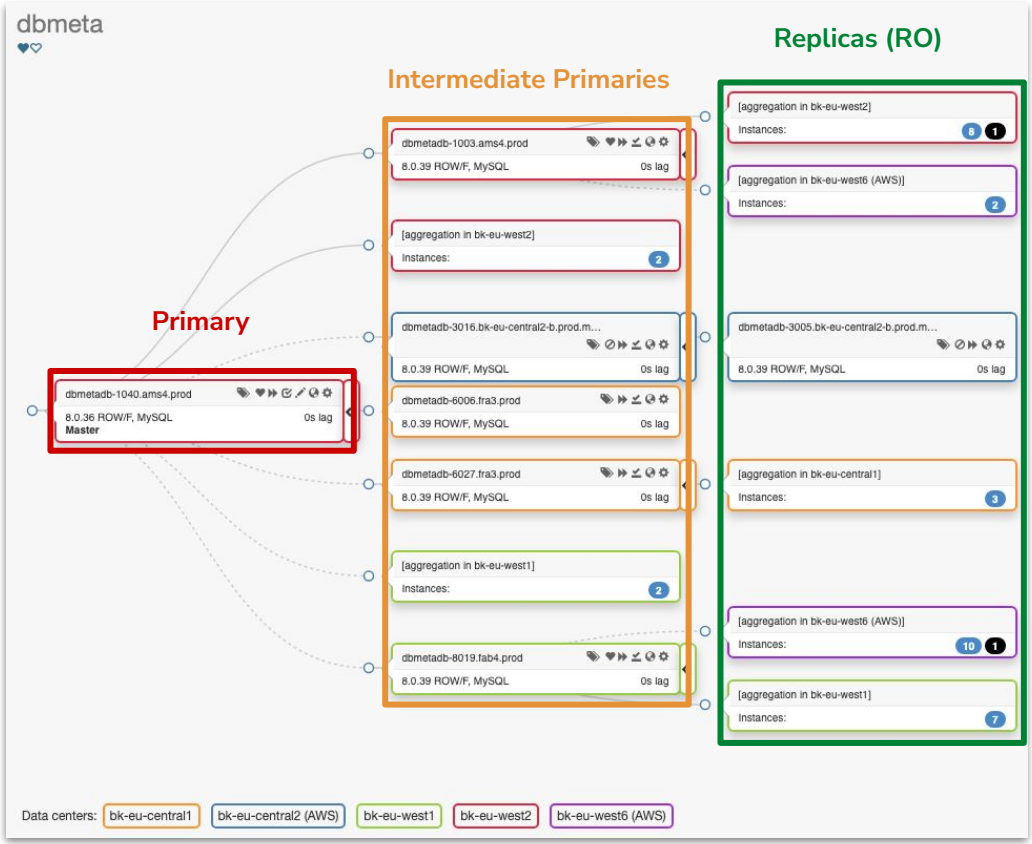
Always deploy in > 2 regions (typically **3 regions** in Europe)

Ownership

We do **not** own data
We do **not** own tables or schemas
We **do** own database infrastructure

1. https://en.wikipedia.org/wiki/CAP_theorem / https://en.wikipedia.org/wiki/PACELC_theorem

Topology of semi-sync MySQL cluster



Database Reliability Engineering (DBRE)

Set of **tools** and **practices** around database systems to ensure they are **reliable**, **scalable** and **compliant** with regulations.

Provide database expertise

Promote best practices for databases, automate database-related operations, provide self-service tools

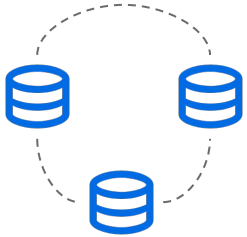
Participate in on-call rotation

Support and debug database issues across services, join firefighting calls, cooperate with peer SREs

Deliver database observability

Implement database monitoring, alert on SLOs, measure performance

SRE Principles for Database Engineering



Monitoring Distributed Systems [1]

Database clusters are distributed across **clouds** and **locations**. Database **primaries** and **replicas** are different by nature.

Eliminating Toil [2]

Running databases at scale comes with a variety of **repetitive operational** tasks that are common across clusters. Aim for automation.

Postmortem culture [3]

Outages **WILL** happen. Do not solve the same problem twice. Educate people and understand systems better.

1. https://sre.google/sre-book/monitoring-distributed-systems
2. https://sre.google/sre-book/eliminating-toil
3. https://sre.google/sre-book/postmortem-culture

The Journey

1. Database Reliability at Booking.com
2. **SLIs and SLOs for Distributed Database Systems**
3. Automating MySQL Capacity Planning
4. Postmortem Culture
5. Q/A

Database performance is sensitive to users

- Full table scans → long-running processes
- Non-indexed, complex queries → slow, expensive
- Database-as-a-queue → locking issues, timeouts
- READ from primary instead of replicas → increased load on primary
- Bulk UPDATE/DELETE → replication delay
- Online Schema Changes (OSC) on huge tables → contention, I/O load
- Excessive number of client connections → hit max connections limit
- ...

Which SLIs we reject for database SLOs

1

Query Latency

The time it takes for a MySQL server to execute a query and return result

2

Throughput

The number of queries a MySQL server can handle per second

3

Replication Delay

The amount of time a MySQL replica is behind its replication source

“If you cannot measure it (correctly) you cannot improve it”

Decide on the right depth!

Customers **don’t** care about
`innodb_buffer_pool_pages_total`
or `cpu_usage_total`

Which SLIs we use for database SLOs

1

Query Latency

The time it takes for a MySQL server to execute a query and return result

2

Throughput

The number of queries a MySQL server can handle per second

3

Replication Delay

The amount of time a MySQL replica is behind its replication source

1

Read Availability

MySQL replicas respond to read requests from clients

2

Write Availability

MySQL primaries respond to write requests from clients

3

Replication Running

MySQL replicas receive changes from source and apply them locally

Read Availability SLO

If at least one successful read probe is achieved the data point is a **pass / OK**, otherwise a **miss / KO**.

The individual measurements are composed over time by taking the percentage of **OK** measurements with respect to the total number of measurements.

```
ten_secondly.mysql.$cluster.read_availability.$region.ok
ten_secondly.mysql.$cluster.read_availability.$region.ko
```

Target

99.99% ("four nines")

Schedule

The SLI is calculated **24/7 every 10 sec**

Who

MySQL Availability Reporter (Golang). Runs with multiple instances across all regions

How

Execute a read probe (SELECT) against **two random replicas** of every MySQL cluster

Timeouts

Connect: **2 sec**, Read: **5 sec**

Write Availability SLO

If at least one successful write probe is achieved the data point is a **pass / OK**, otherwise a **miss / KO**.

The individual measurements are composed over time by taking the percentage of **OK** measurements with respect to the total number of measurements.

```
secondly.mysql.$cluster.write_availability.$region.ok
secondly.mysql.$cluster.write_availability.$region.ko
```

Target

99.97% ("three nines seven")

Schedule

The SLI is measured **24/7 every 1 sec**

Who

MySQL Availability Reporter (Golang). Runs with multiple instances across all regions

How

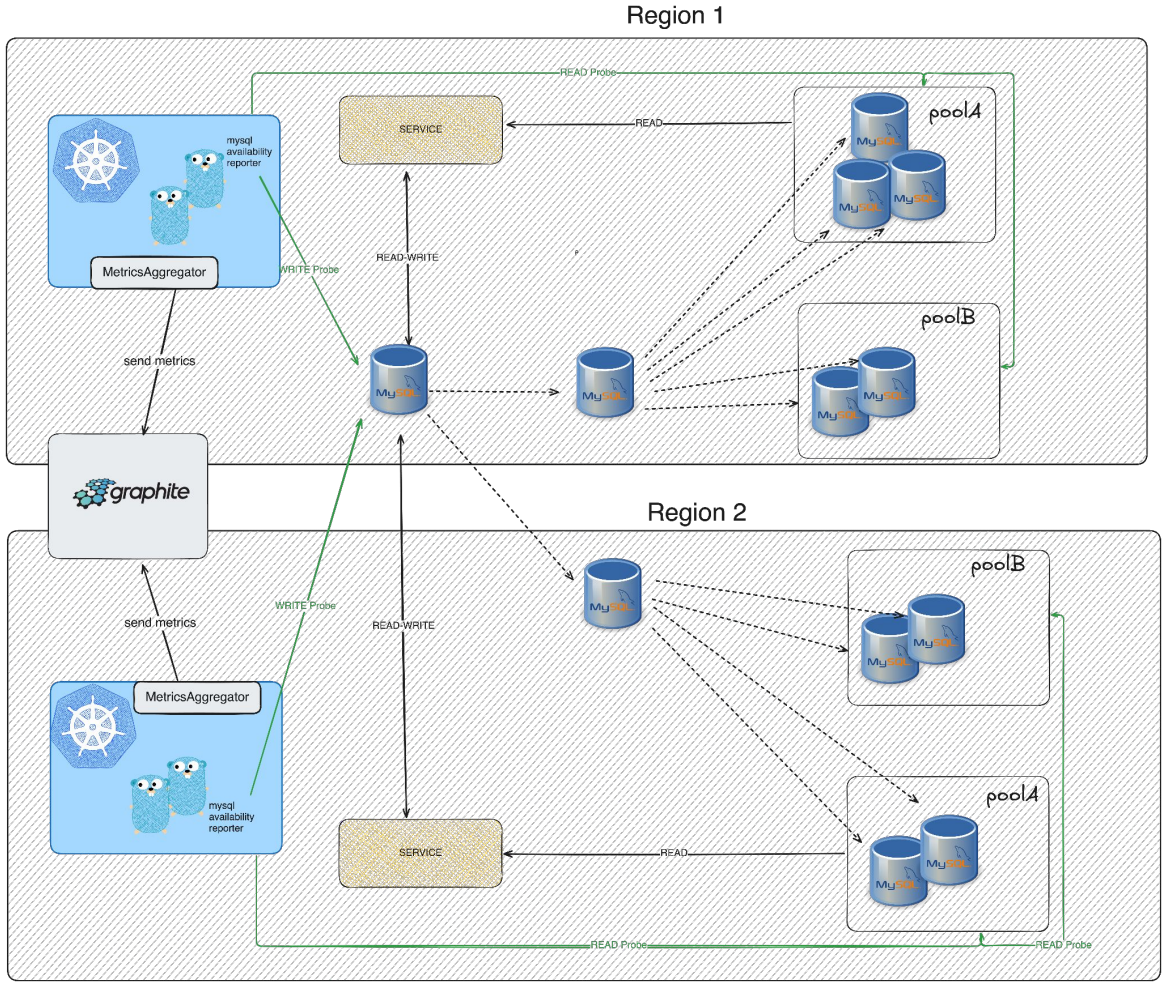
Execute a write probe (INSERT) against **the (single) primary** of every MySQL cluster

Timeouts

Connect: **2 sec**, Write: **8 sec**

MySQL Availability Reporter

- In-house **Golang** project
- Runs on Kubernetes and works **across platforms**
- **Imitates** MySQL's consumers (apps)
- **Knows** about Service Discovery
- **Monitors** READ and WRITE availability with **probes**
- Stores (aggregated) metrics in remote **Graphite** instance



Replication Running SLO

If replication is running between the replica and the primary (source) the data point is considered **pass / OK**, otherwise a **miss / KO**.

The individual measurements are composed over time by taking the percentage of **OK** measurements with respect to the total number of measurements.

```
ten_secondly.mysql.$cluster.$region.replication_running.ok
ten_secondly.mysql.$cluster.$region.replication_running.ko
```

Target

99.95% ("three nines five")

Schedule

The SLI is measured **24/7 every 10 sec**

Who

Badmin Agent (Python). Runs with a single instance inside every MySQL server

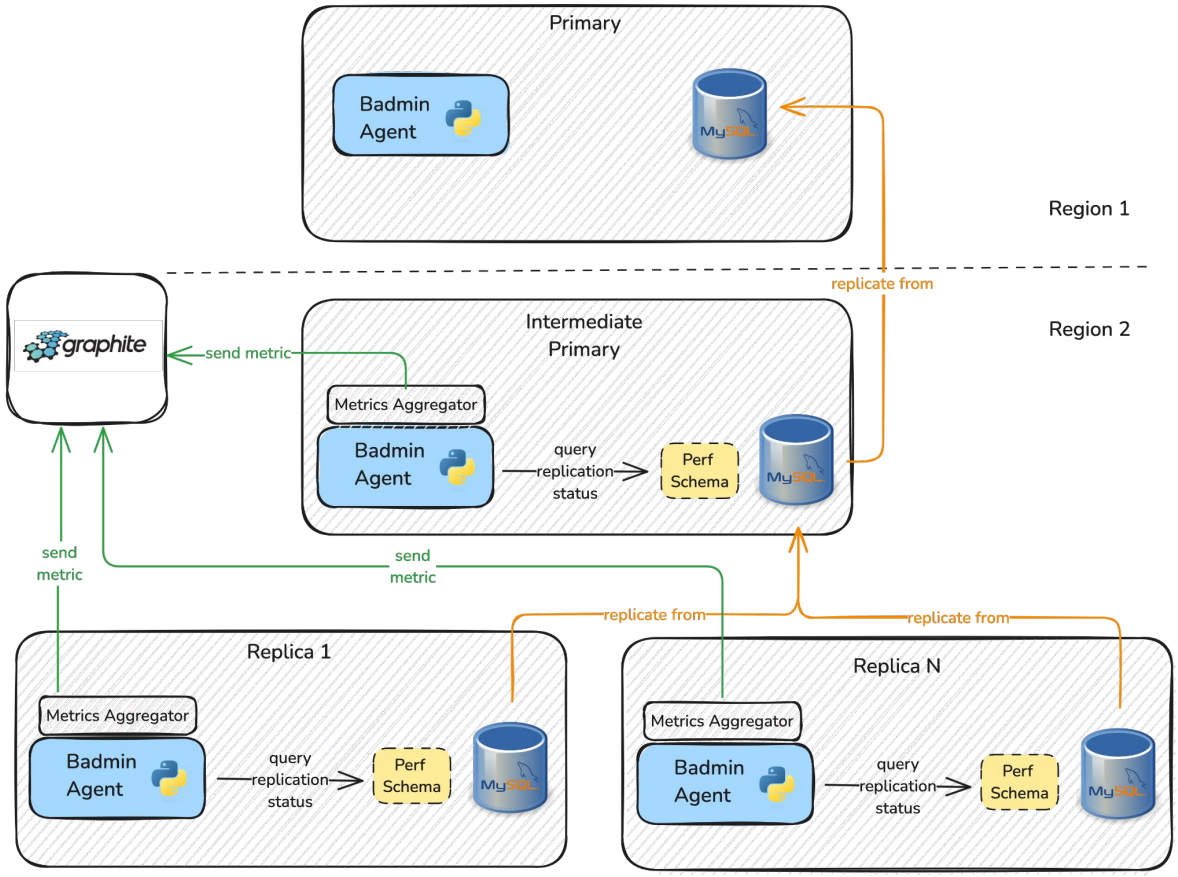
How

Query the performance schema[1] for replication status[2] of **every replica** of every MySQL cluster

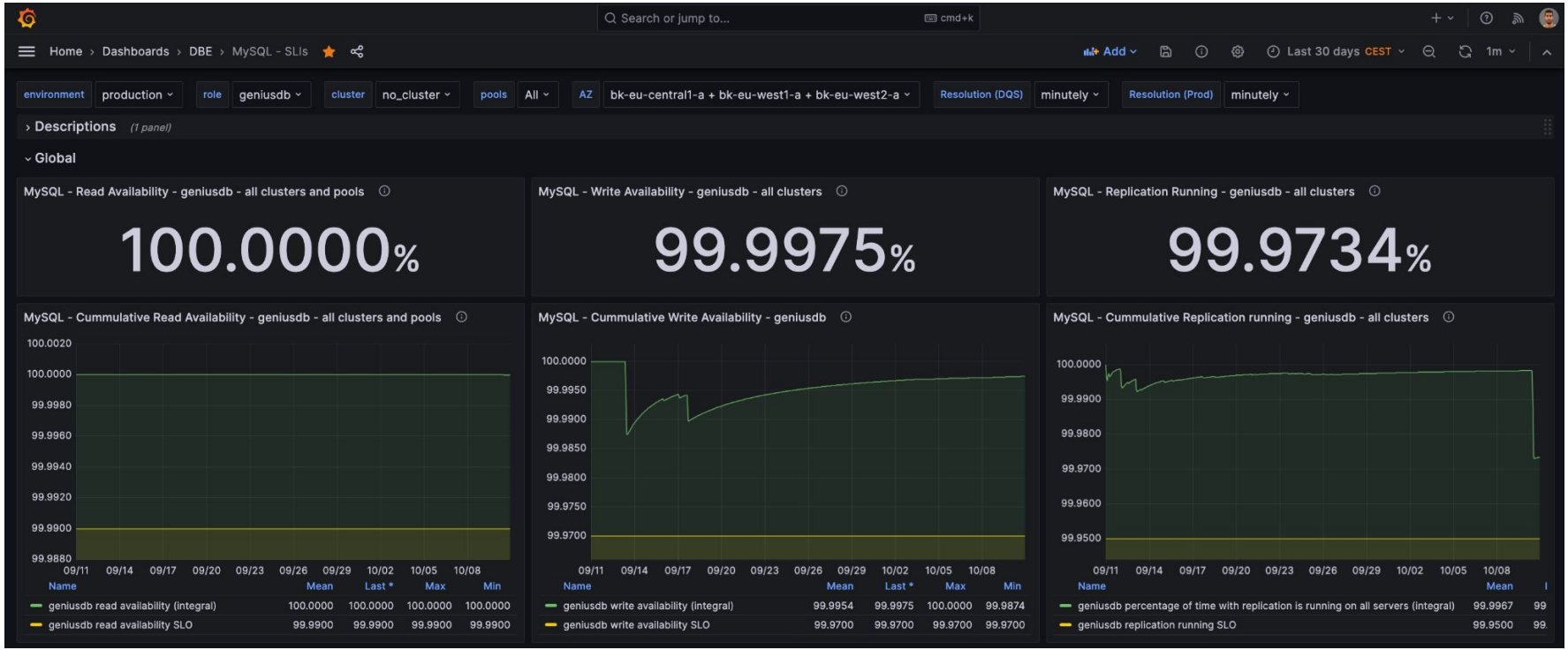
1. https://dev.mysql.com/doc/refman/8.4/en/performance-schema.html
2. https://dev.mysql.com/doc/refman/8.4/en/show-replica-status.html

Badmin Agent

- In-house **Python** project (10+ years old)
- **Sits** next to every MySQL server
- **Provides** functionality and automation for DBA tasks
- **Queries** MySQL performance schema for replication status
- Stores (aggregated) metrics in remote **Graphite** instance

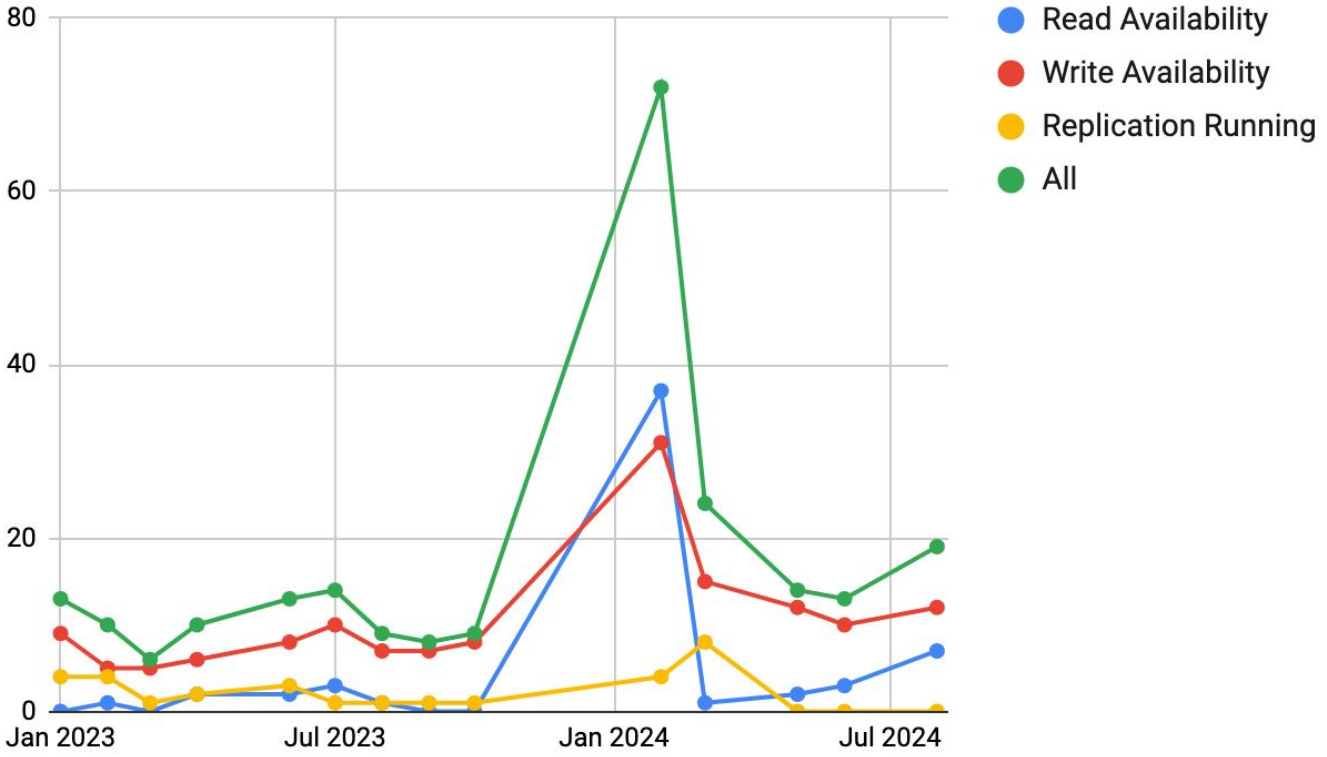


MySQL SLIs/SLOs in action - Genius DB

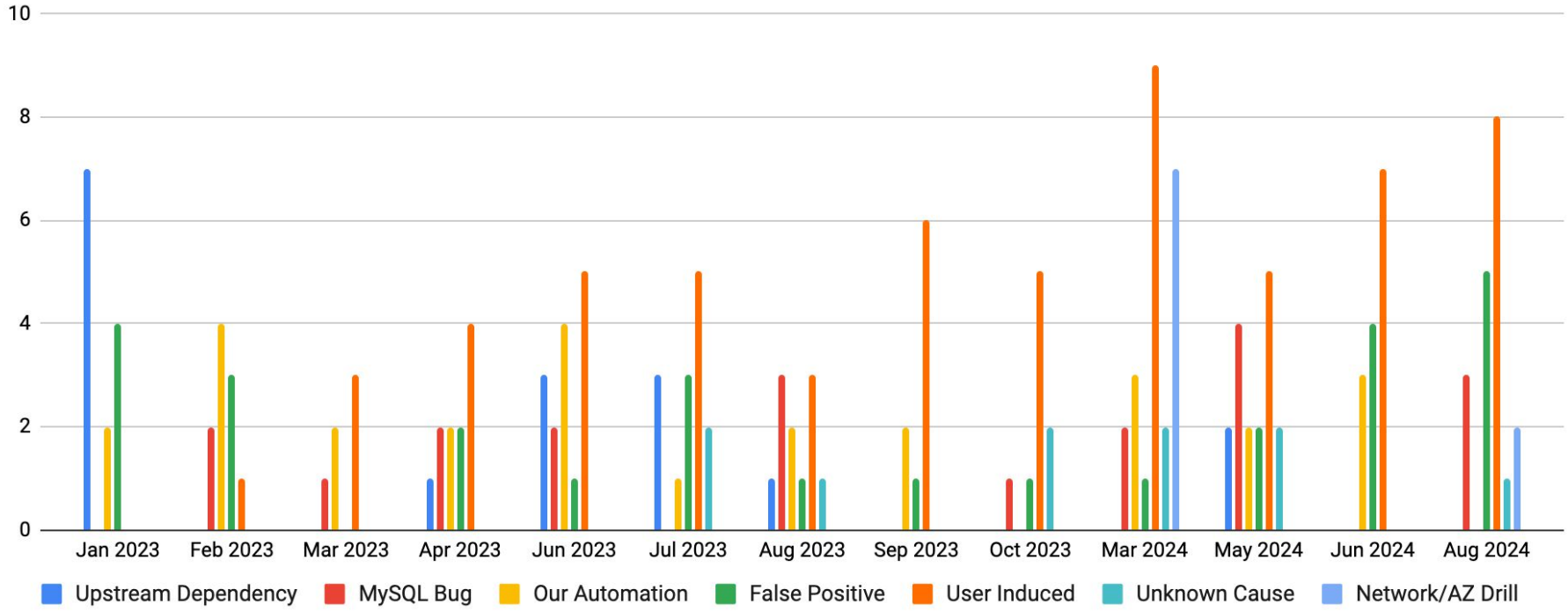


MySQL SLO Violations

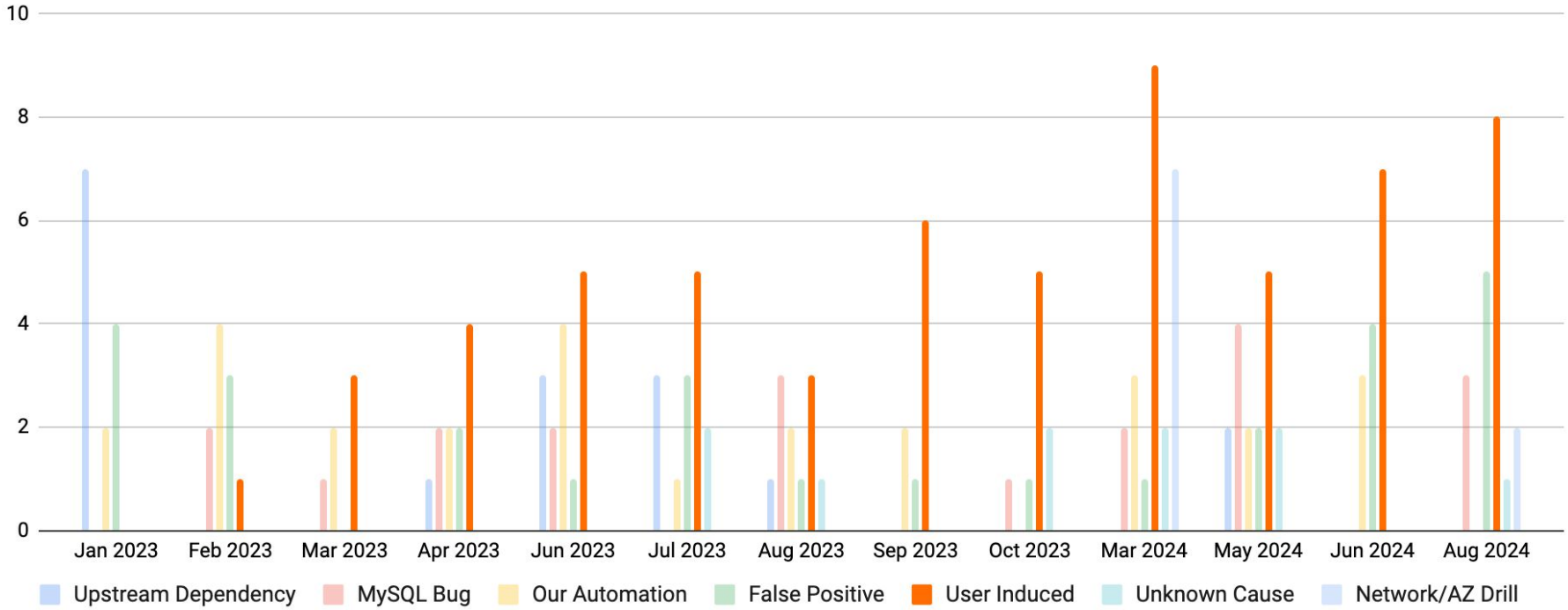
- Write Availability SLO is violated more often
- The number SLO violations is small given the number of MySQL clusters
- Spikes can happen



Who violates our MySQL SLOs?



Who violates our MySQL SLOs?



Takeaways

Choose SLIs that are 100% under your control to define your database SLOs

This helps you avoid “false positives” and stay within your SLO budget. Reduce “alert fatigue”

Decide where it makes sense to measure your SLI

Depending on the metric, you might need to measure close to the database server or mimic “how the application sees it”

Don’t pursue much reliability than what’s strictly necessary

Aiming at a 100% SLO is a bad decision. Use your SLO budget and always educate developers

The Journey

1. Database Reliability at Booking.com
2. SLIs and SLOs for Distributed Database Systems
3. **Automating MySQL Capacity Planning**
4. Postmortem Culture
5. Q/A

Database tasks introduce TOIL

- Server provisioning, configuration, maintenance
- Schema splits, changes
- Password rotation, grants
- Switchover/failover of primaries
- Daily backups and restores of volumes
- MySQL upgrades
- Fleet sizing
- Connection management
- Heal broken replication
- ...

Our approach: Relentless Automation

Reliability is our top priority: alerts alone are not **enough**

Choose **proactive** over reactive

Our fleet **grows faster** than our database engineering teams

Why we automated MySQL Capacity Planning

Capacity planning is critical for **Keeping The Lights On** (KTLO)

- **Risk mitigation:** undersized MySQL clusters are a risk for business (peak load)
- **Cost control:** oversized MySQL clusters increase our bill
- **Scalability:** time-consuming, repetitive task common for all MySQL clusters
- **Continuous Learning:** examine new hardware profiles and workloads, know your tools and infrastructure better
- **Analytics:** vertical vs horizontal scaling, forecasting, reporting

MySQL Capacity Planning: Design Decisions

01

Schedule at peak hour daily

We choose to run capacity tests on MySQL servers at the time of peak traffic, every day

02

Run against our production fleet

We choose to run capacity tests on MySQL servers serving production traffic

03

Ability to stop capacity test at any time

In case of emergency (e.g., outage) we want a killswitch to stop a capacity test immediately

04

Prioritize business continuity over cost efficiency

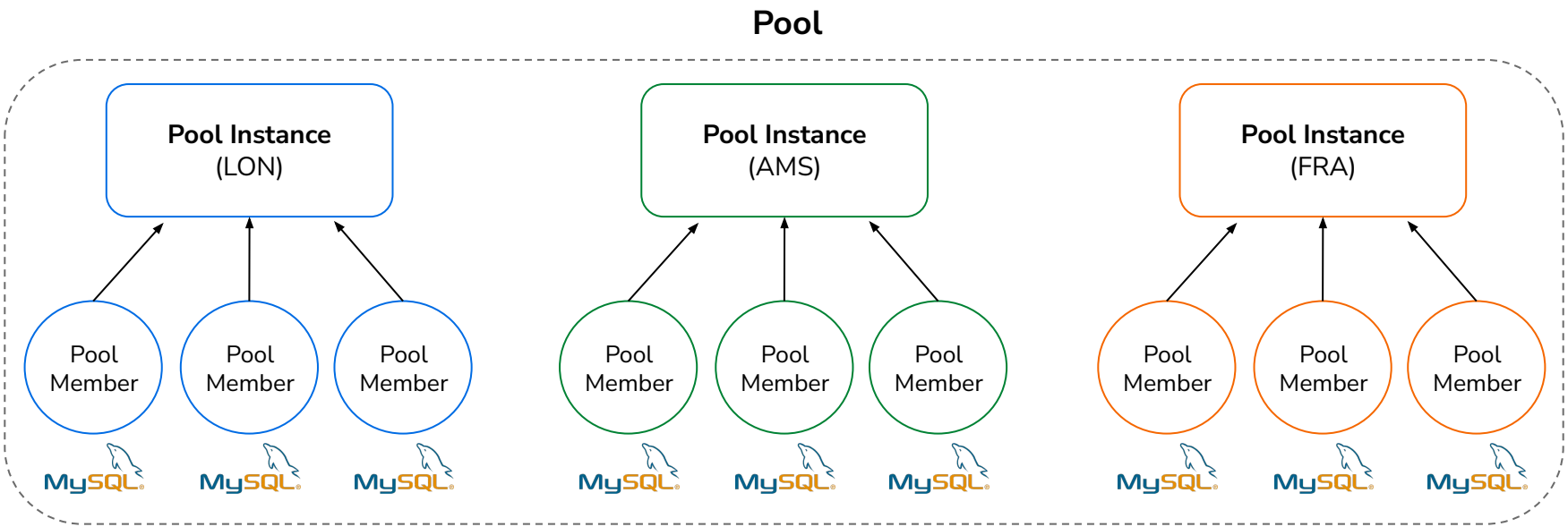
Introduce safety measures to ensure that MySQL pools always have sufficient capacity to serve traffic

The building block for capacity planning: MySQL pools

Logical definition of **READ** query workload coming from specific services

→ Isolation, separation of concerns, solve “noisy neighbour” problem

Pools have multiple **instances** in different **regions**



MySQL Capacity Metrics

Retrieved from Prometheus mysqld_exporter [1]

- **`mysql.capacity_planning.$cluster.$pool.$region.Com_select`**

  `irate()[2m]` of the number of executed SELECT queries on the MySQL server

- **`mysql.capacity_planning.$cluster.$pool.$region.Threads_connected`**

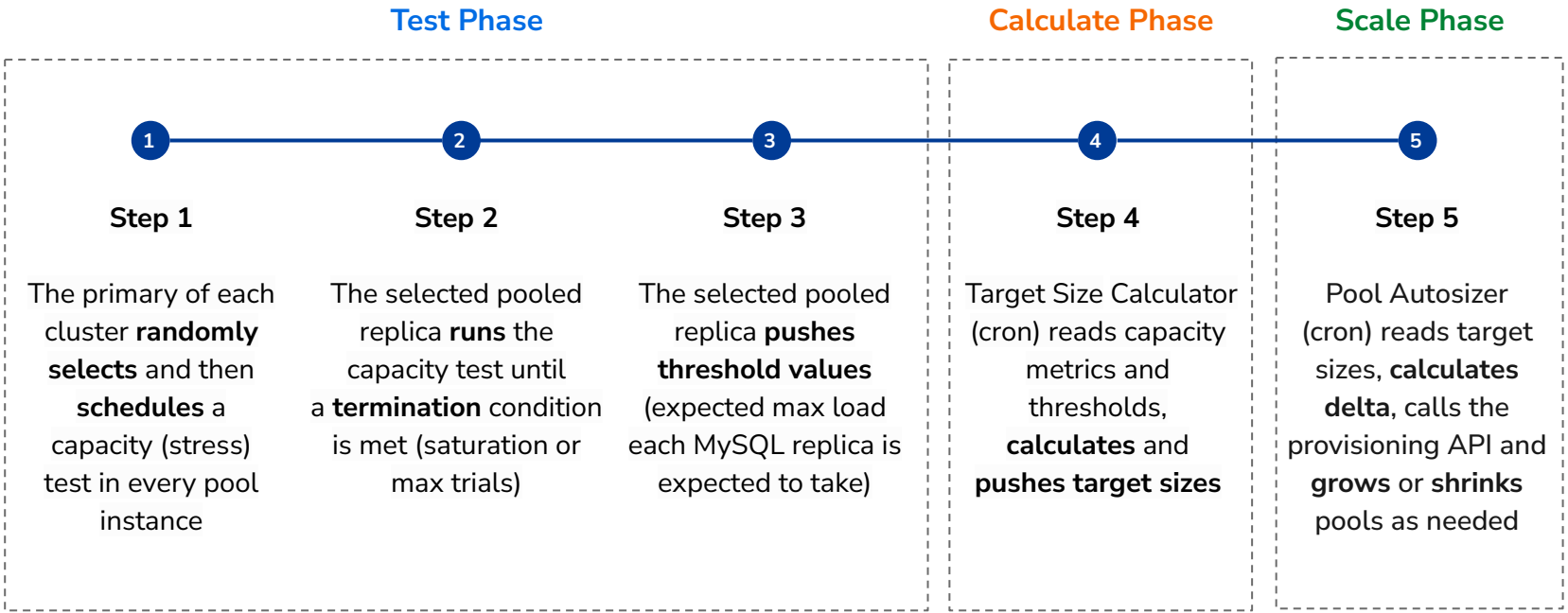
  The number of threads (clients) connected to the MySQL server

- **`mysql.capacity_planning.$cluster.$pool.$region.Threads_running`**

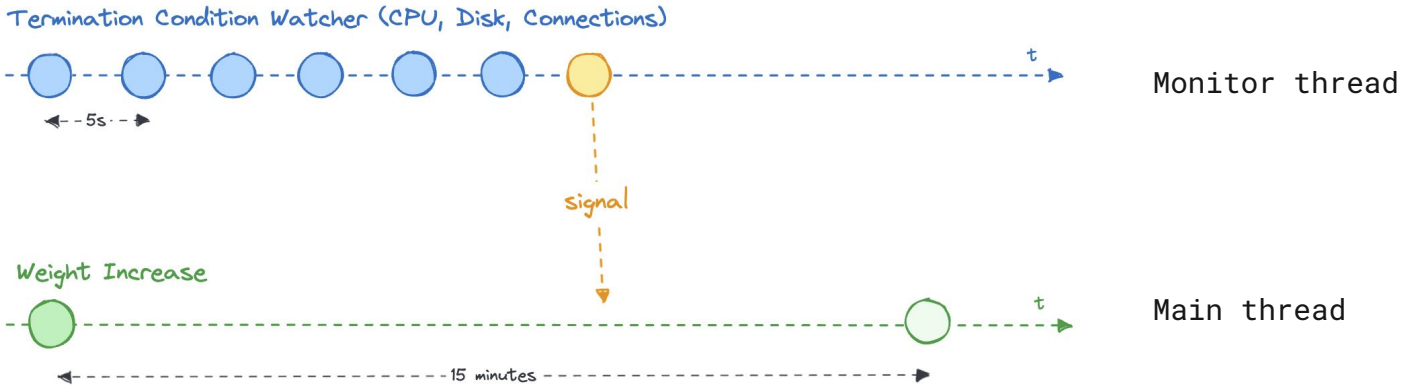
  The number of database connections with an active query

1. https://github.com/prometheus/mysqld_exporter (Our fork, sorry!)

Phased Approach: Test, Calculate, Scale



Increase traffic, monitor saturation



Every 15 min we increase the **weight** of the pool member until saturation point **or** `MAX_TRIALS`

Detect saturation: **CPU utilization**, **disk read latency**, **MySQL max connections**[1]

1. https://dev.mysql.com/doc/refman/8.4/en/server-system-variables.html#sysvar_max_user_connections

Target sizes for MySQL pools

Regional Target Size

`ceil(metric_value/threshold)`

metric value = **p90** value in the last **24h**
(emitted by mysqld exporter)
threshold = **median** over the last **7 days**
(emitted by the capacity test)

- Keep the **maximum** target size found after checking all 3 capacity metrics
- Respect empirical upper and lower **thresholds** to prevent drastic changes in pool sizes

Global Target Size

Sum of regional target sizes for all `N` regions divided by `N-1` (over provisioning for redundancy)

Extra adjustments:

1) **Minimum** target size adjustment (business critical clusters)

2) **Additive** adjustment (budget for maintenance, hardware failures, etc.)

Example

Let's assume a pool with instances in 3 regions and regional target sizes of 5, 5 and 2, respectively.

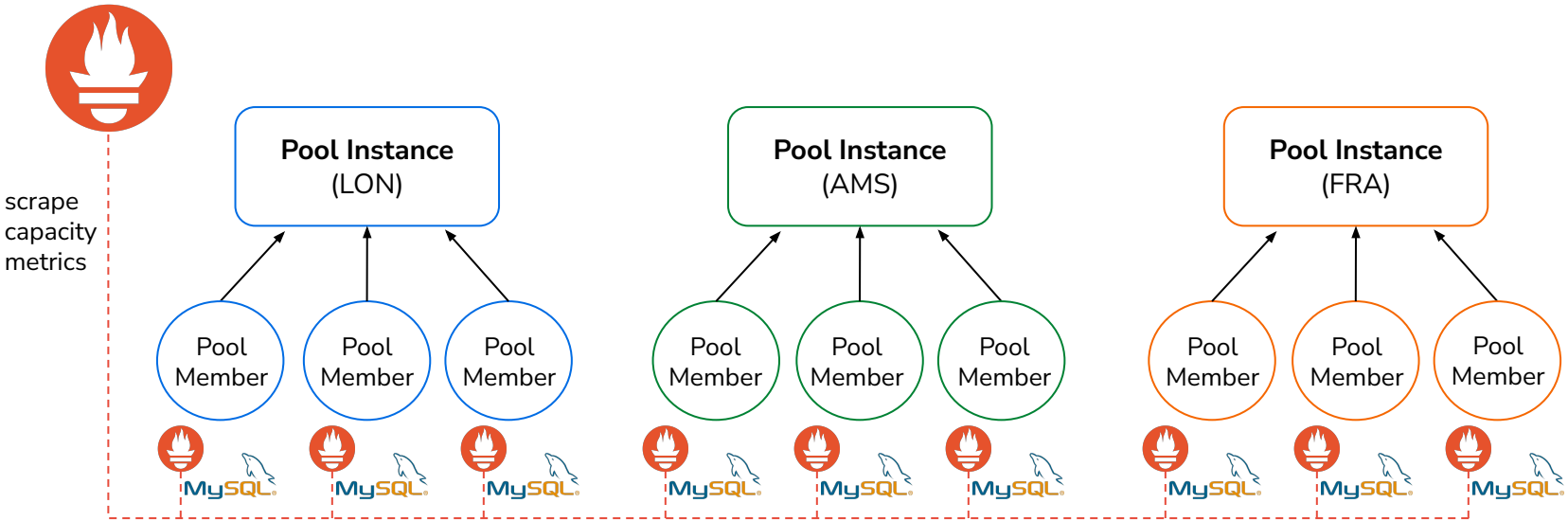
Without over-provisioning we would get
12 / 3 = **4** replicas per region

With over-provisioning we get
12 / 2 = **6** replicas per region

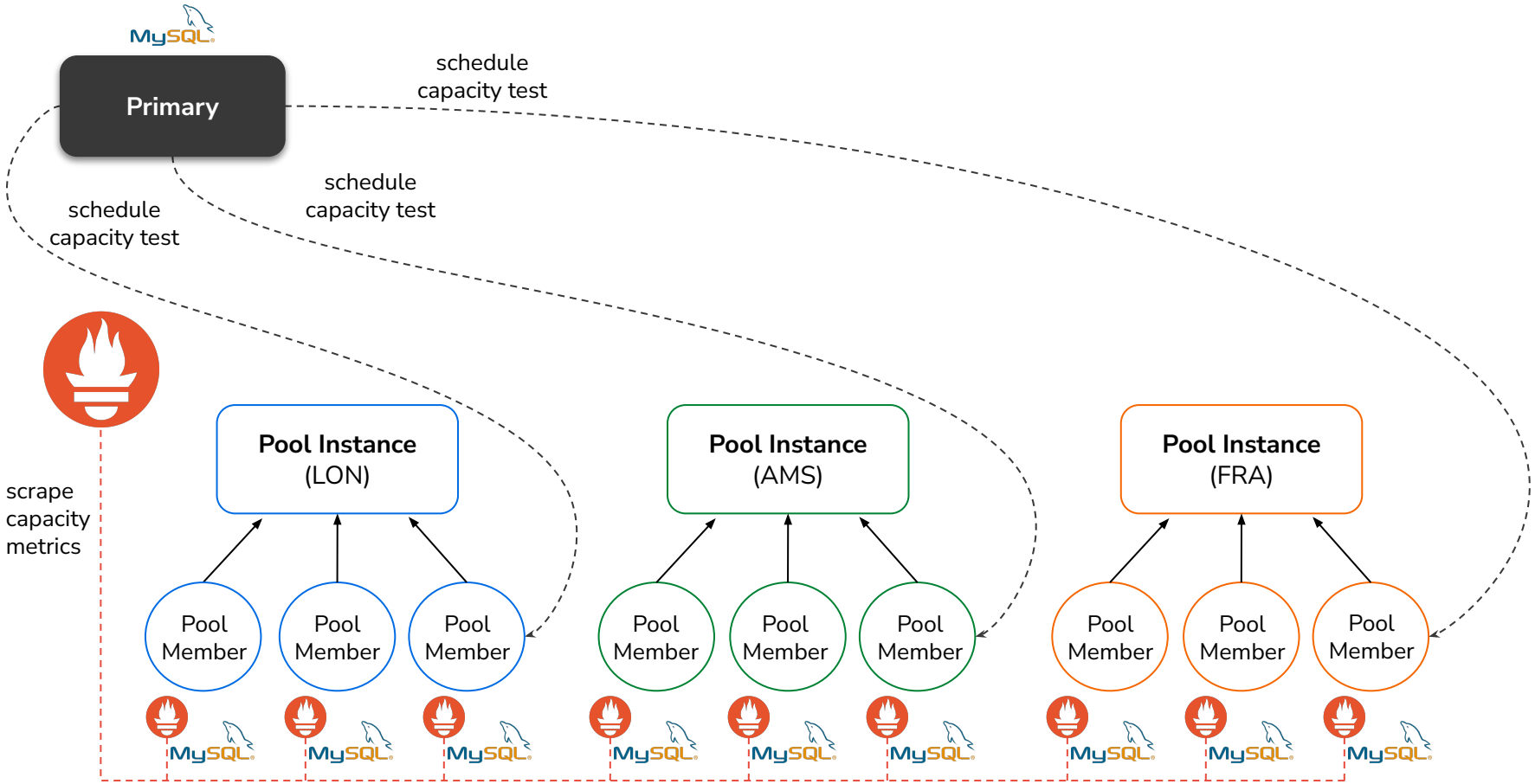
Assuming a regional failure, each pool instance should be able to handle
1.5x times the traffic that it normally handles

Test Phase


MySQL
Primary

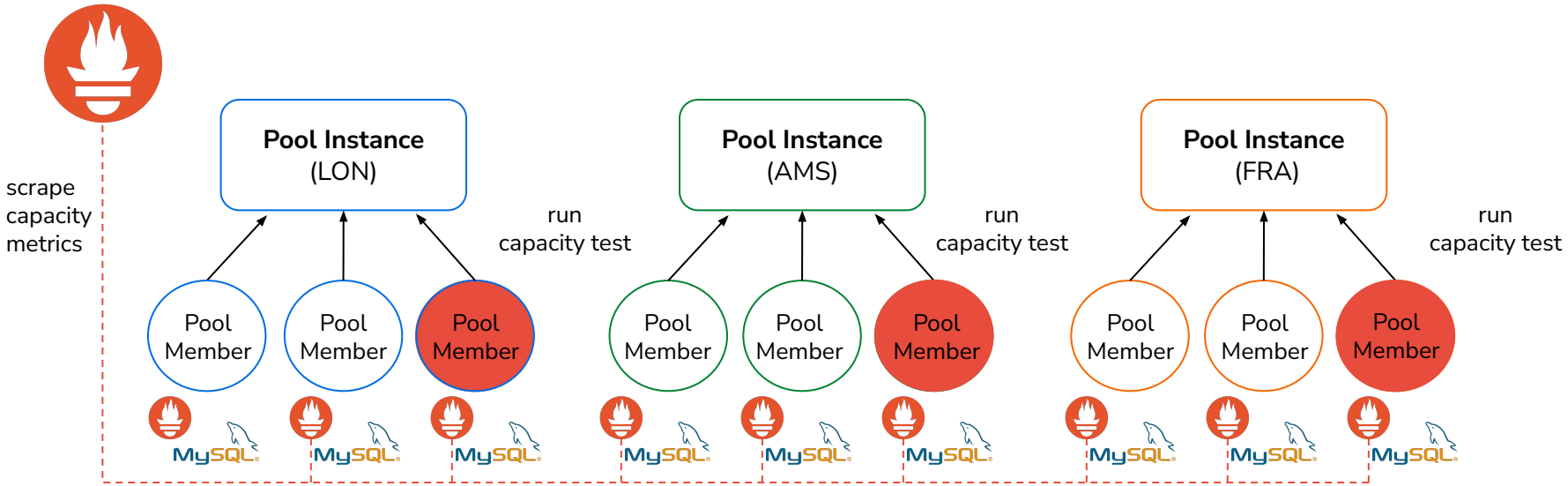

scrape
capacity
metrics
Pool Instance
(LON)
Pool Instance
(AMS)
Pool Instance
(FRA)
Pool Member
Pool Member
Pool Member
Pool Member
Pool Member
Pool Member
Pool Member
Pool Member
Pool Member
MySQL
MySQL
MySQL
MySQL
MySQL
MySQL
MySQL
MySQL
MySQL

Test Phase

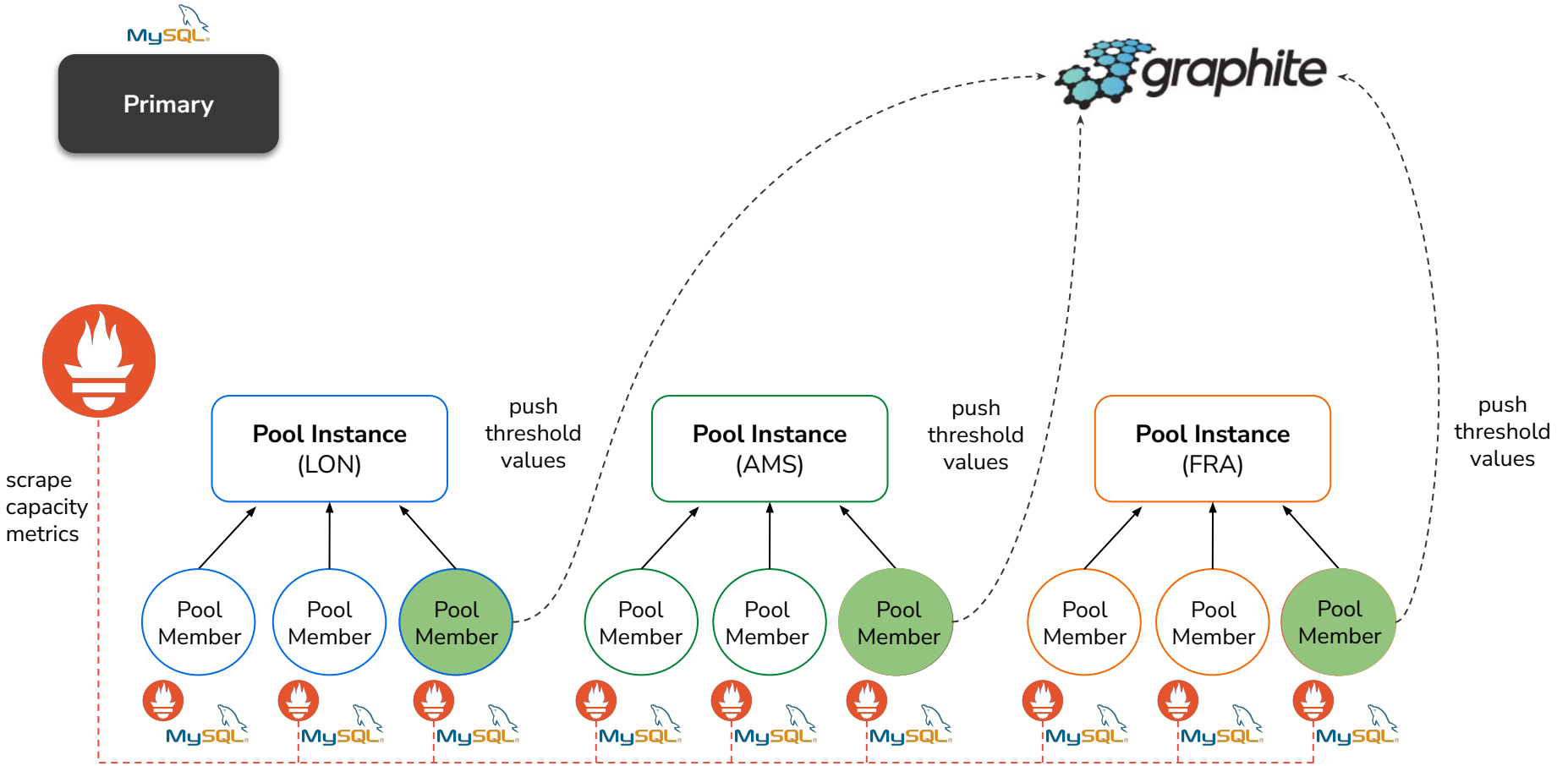

MySQL
Primary
schedule capacity test
schedule capacity test
schedule capacity test
scrape capacity metrics
Pool Instance (LON)
Pool Instance (AMS)
Pool Instance (FRA)
Pool Member
Pool Member
Pool Member
Pool Member
Pool Member
Pool Member
Pool Member
Pool Member
Pool Member
MySQL
MySQL
MySQL
MySQL
MySQL
MySQL
MySQL
MySQL
MySQL

Test Phase

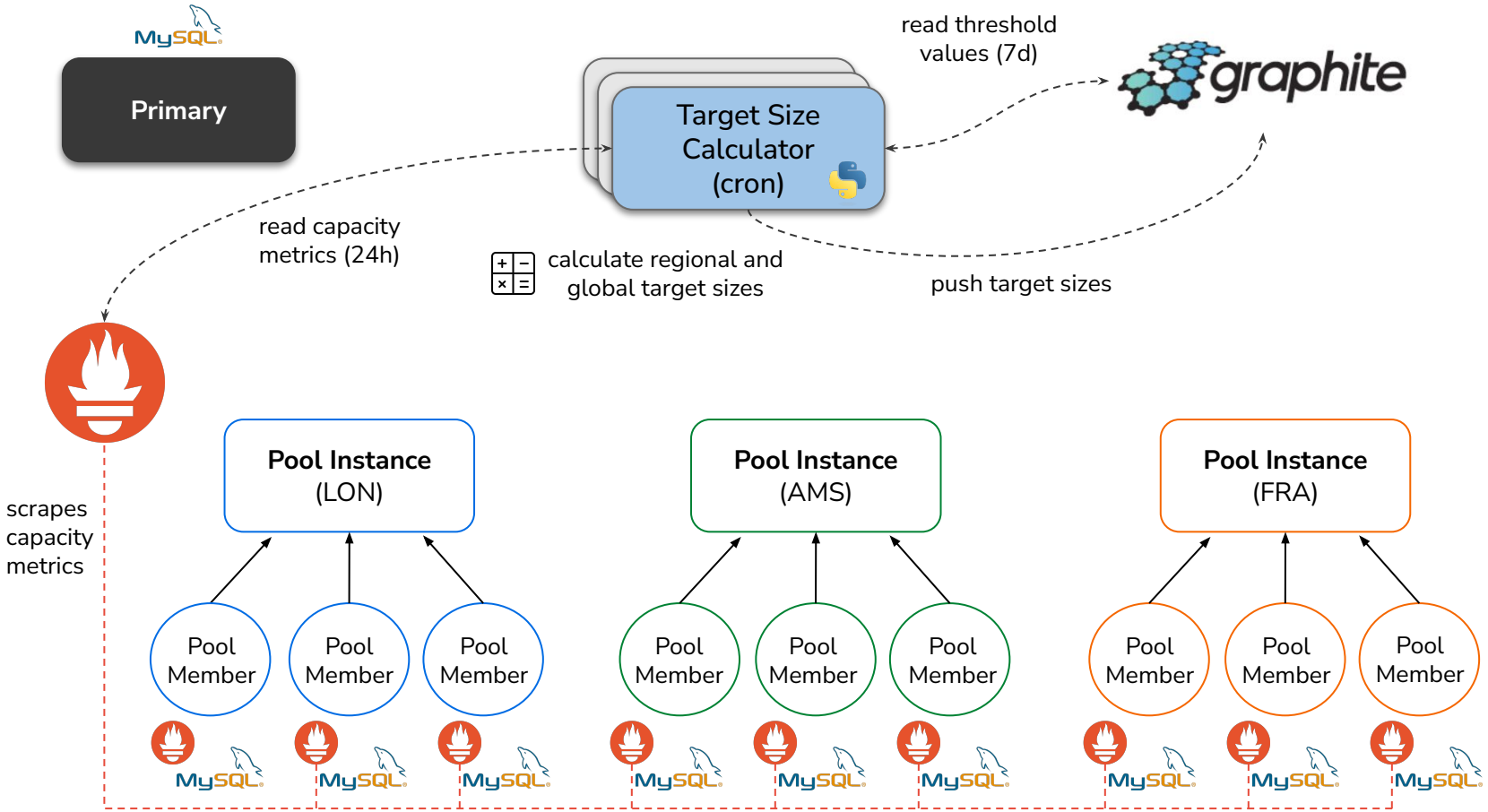

MySQL
Primary


scrape
capacity
metrics
Pool Instance
(LON)
Pool Instance
(AMS)
Pool Instance
(FRA)
run
capacity test
run
capacity test
run
capacity test
Pool
Member
Pool
Member
Pool
Member
Pool
Member
Pool
Member
Pool
Member
Pool
Member
Pool
Member
Pool
Member
MySQL
MySQL
MySQL
MySQL
MySQL
MySQL
MySQL
MySQL
MySQL

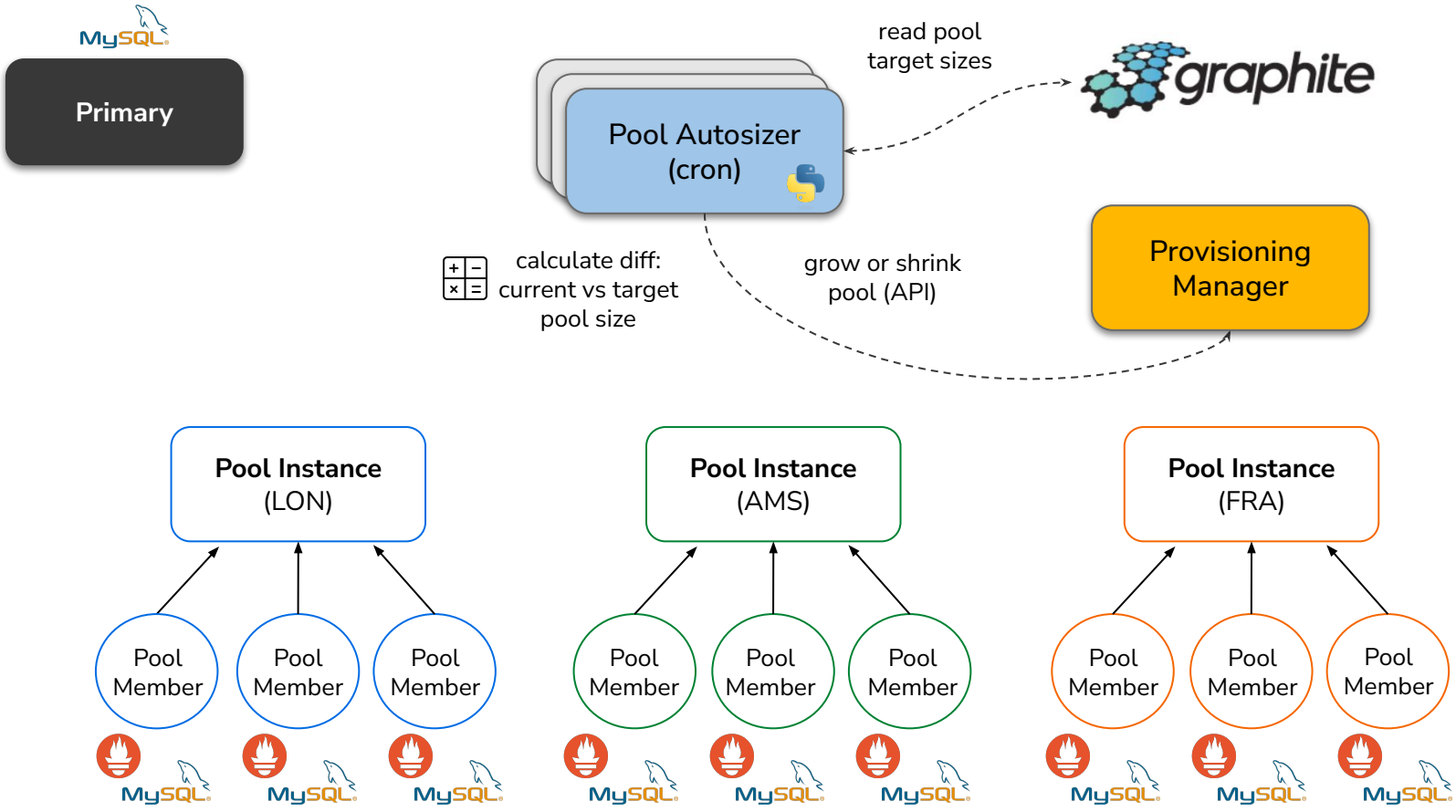
Test Phase


MySQL
Primary
graphite
Pool Instance (LON)
Pool Instance (AMS)
Pool Instance (FRA)
Pool Member
push threshold values
scrape capacity metrics

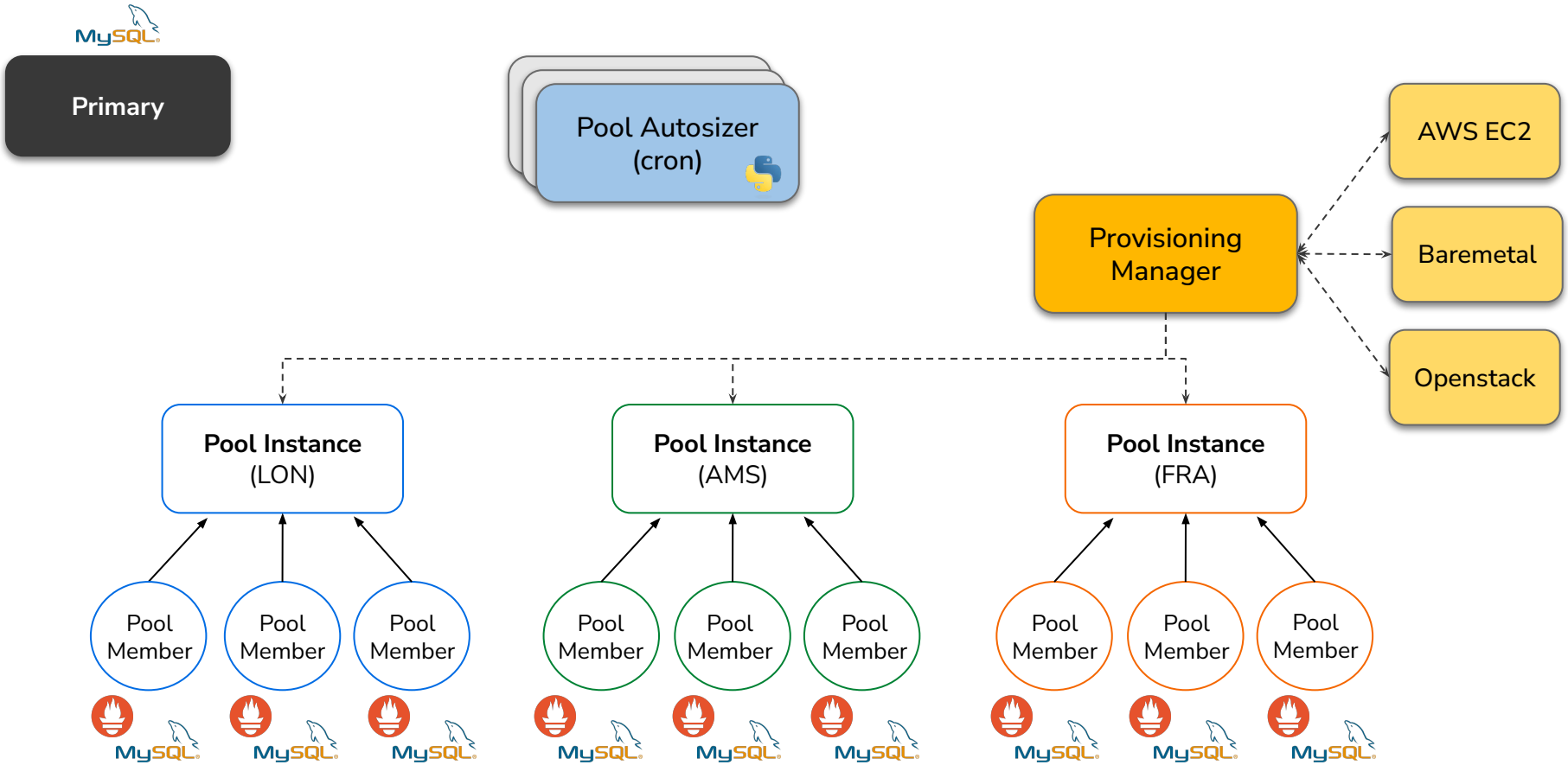
Calculate Phase



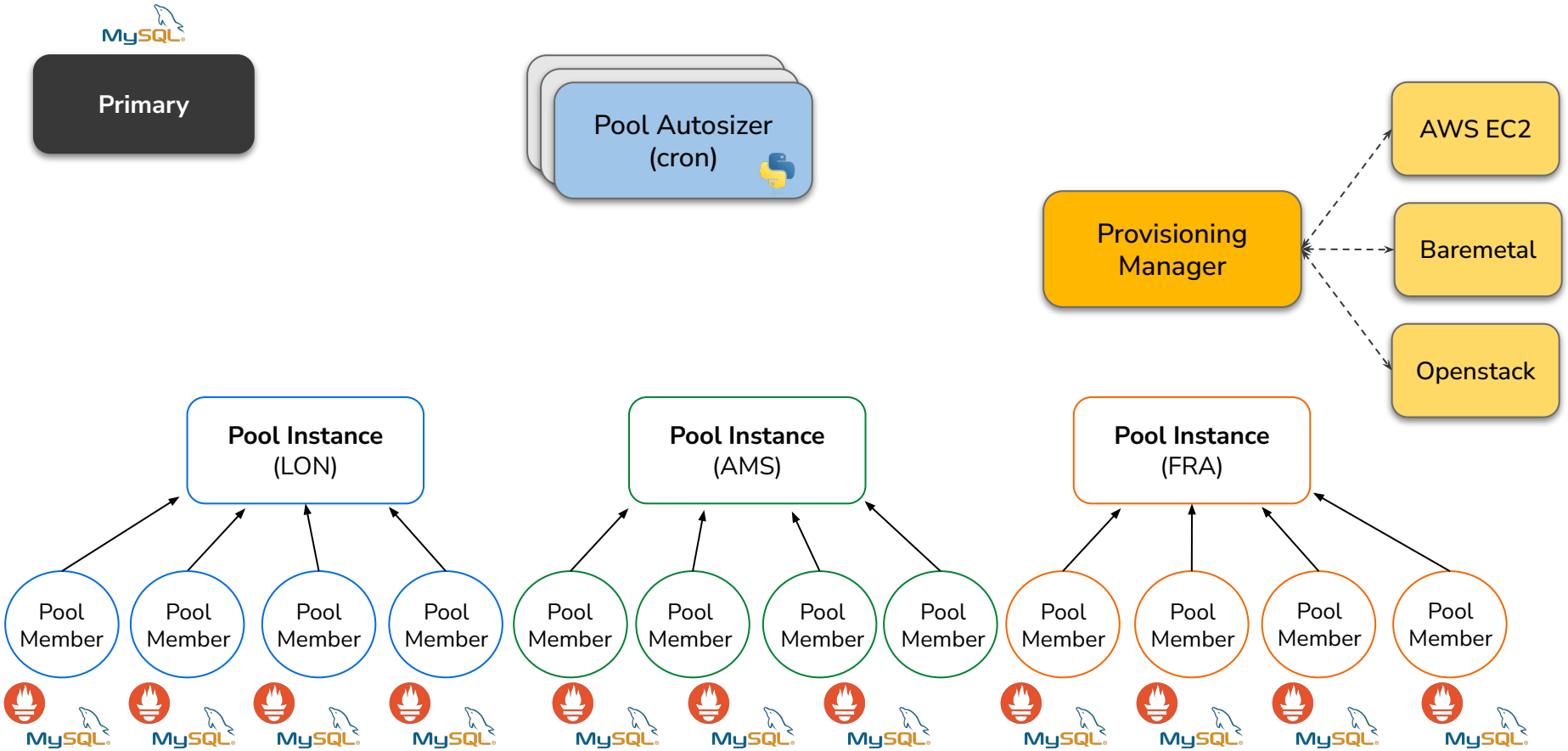
Scale Phase


MySQL
Primary
read pool
target sizes
graphite
Pool Autosizer
(cron)
calculate diff:
current vs target
pool size
grow or shrink
pool (API)
Provisioning
Manager
Pool Instance
(LON)
Pool Instance
(AMS)
Pool Instance
(FRA)
Pool
Member
Pool
Member
Pool
Member
Pool
Member
Pool
Member
Pool
Member
Pool
Member
Pool
Member
Pool
Member
MySQL
MySQL
MySQL
MySQL
MySQL
MySQL
MySQL
MySQL
MySQL

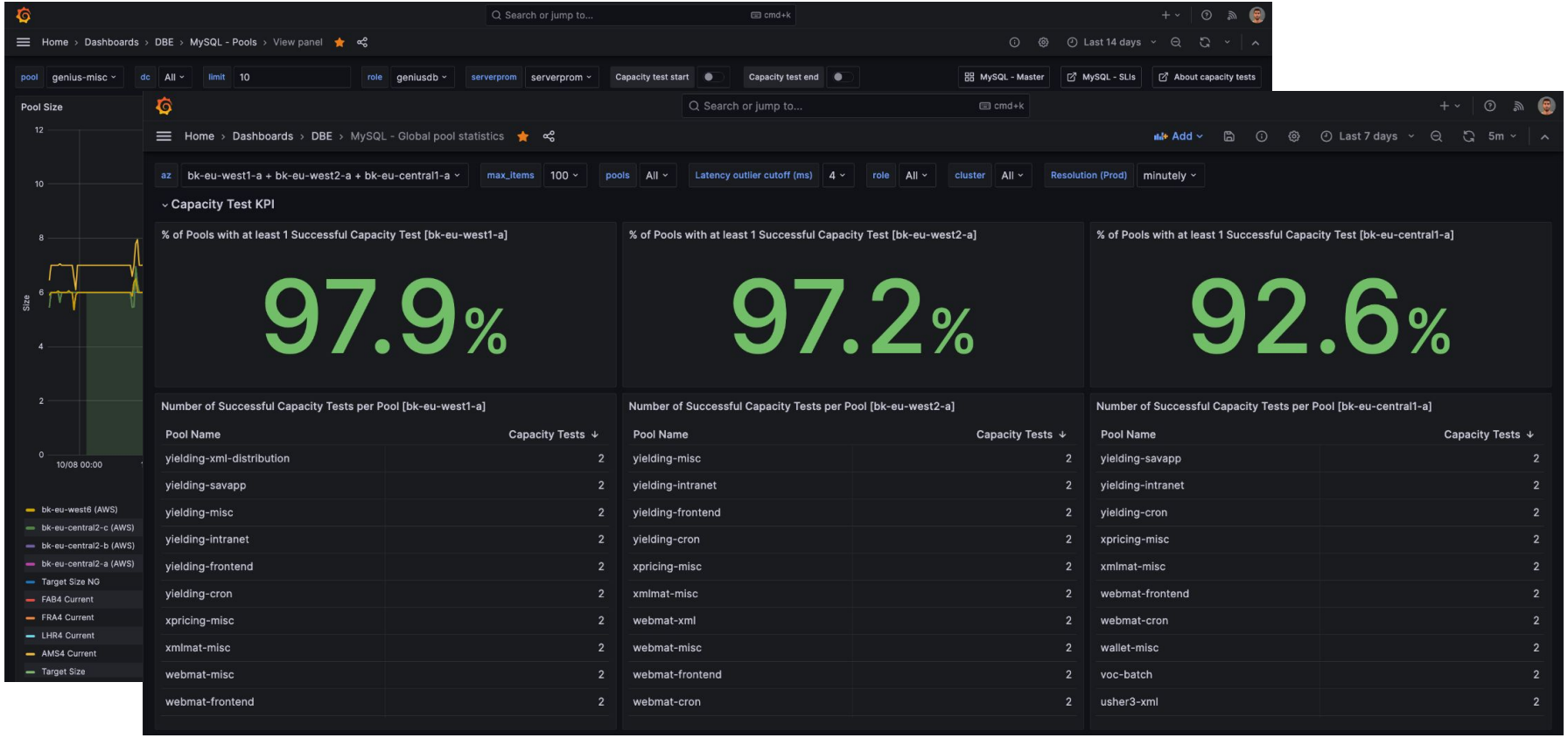
Scale Phase


MySQL
Primary
Pool Autosizer
(cron)
AWS EC2
Provisioning
Manager
Baremetal
Openstack
Pool Instance
(LON)
Pool Instance
(AMS)
Pool Instance
(FRA)
Pool
Member
Pool
Member
Pool
Member
Pool
Member
Pool
Member
Pool
Member
Pool
Member
Pool
Member
Pool
Member
MySQL
MySQL
MySQL
MySQL
MySQL
MySQL
MySQL
MySQL
MySQL

Scale Phase


MySQL
Primary
Pool Autosizer
(cron)
Provisioning
Manager
AWS EC2
Baremetal
Openstack
Pool Instance
(LON)
Pool Instance
(AMS)
Pool Instance
(FRA)
Pool
Member
Pool
Member
Pool
Member
Pool
Member
Pool
Member
Pool
Member
Pool
Member
Pool
Member
Pool
Member
Pool
Member
Pool
Member
Pool
Member

MySQL Pool Autosizing in action



Future Plans

- Explore **new** hardware profiles available in private and public cloud
- **Fine-tune** capacity metric thresholds towards cost efficiency
- **Minimise** errors (e.g., configure grants, database locks)
- Use a **single** sink for metrics (Prometheus vs Graphite)

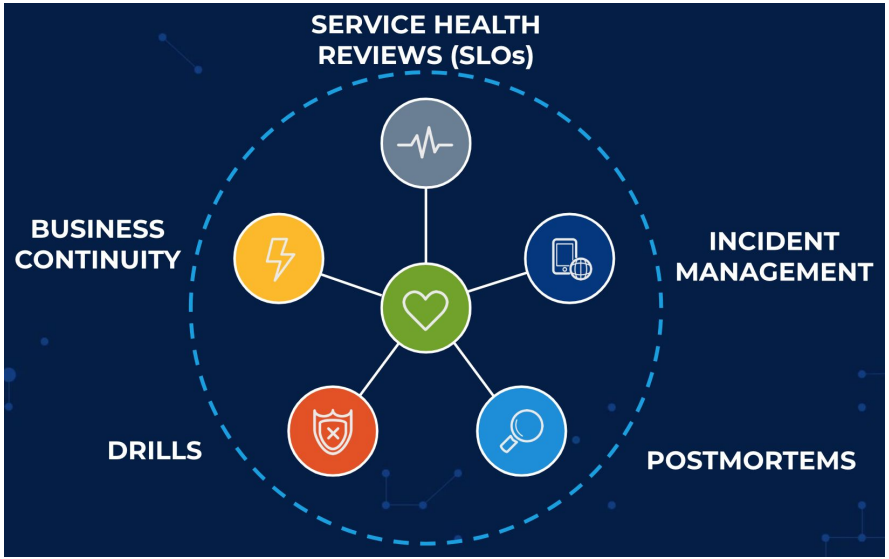
The Journey

1. Database Reliability at Booking.com
2. SLIs and SLOs for Distributed Database Systems
3. Automating MySQL Capacity Planning
4. **Postmortem Culture**
5. Q/A

Systems break, that is life!

stay calm...or try

Our Service Health Program



Service Health Review

Continuous process to reviews the statuses of services we own.

Incident Management

Standard process to enable services to deliver on SLOs

Postmortems / RFO (Reason For Outage)

Continuous learning. Incidents are a NICE opportunity to learn.

Network Drills

Keep our business running with the minimum impact.

Business Continuity

Automated checks of failure recovery aspects (capacity planning, failovers, redundancy, backup, restore, etc).

Postmortem - we embrace the risk together!

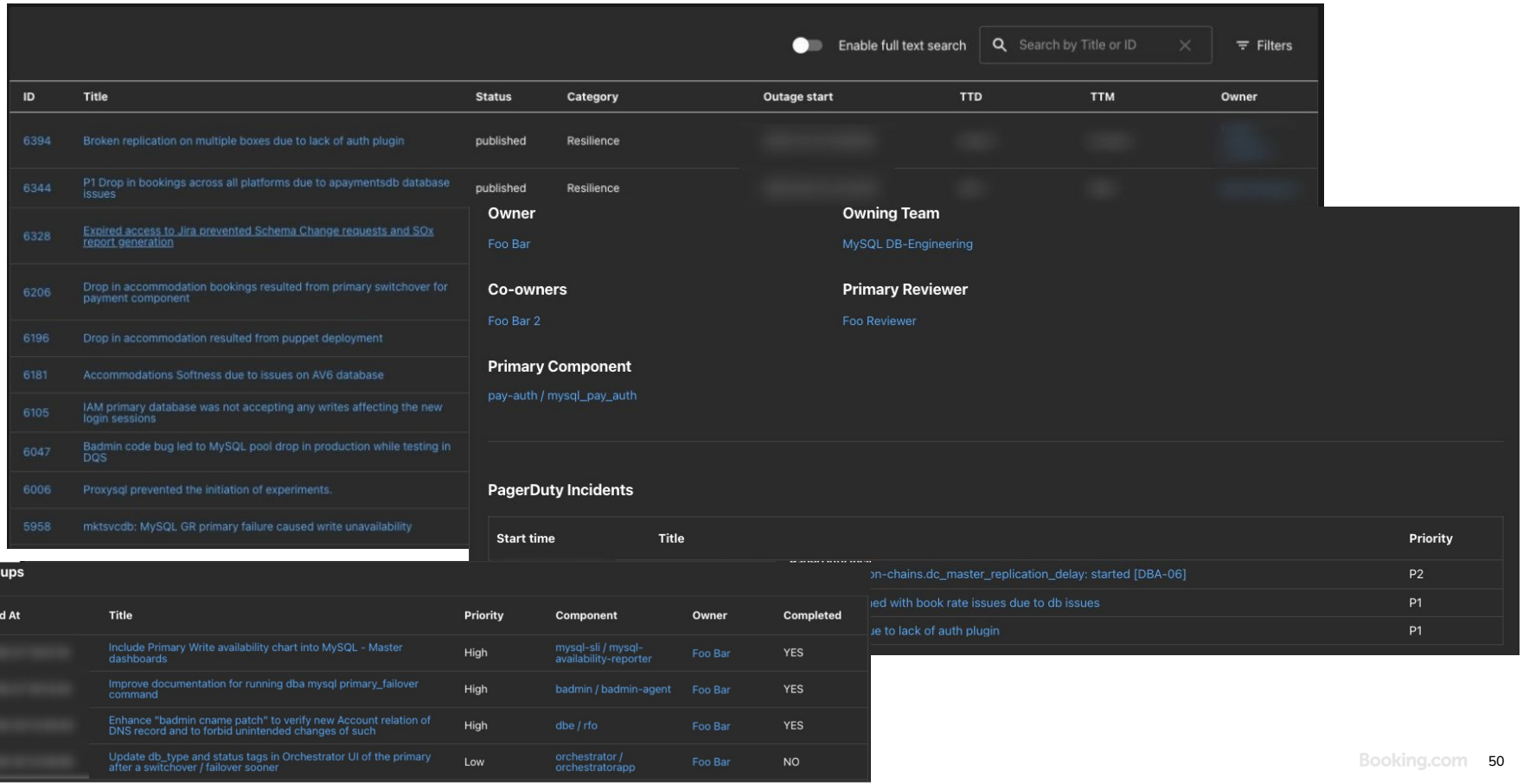
- Drive Service **quality** and avoid **repeated** failures
- Understanding / Eliminating the **root cause**
- **Improve** our systems and processes
- **Learning and continuous improvement**

Blameless

Blameless

Blameless

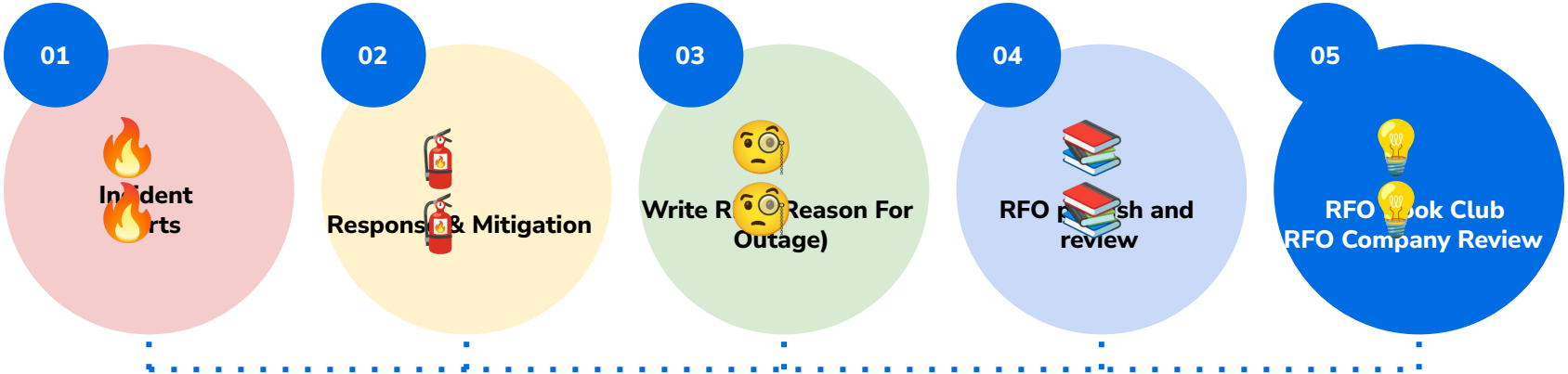
RFO (Reason For Outage) - “RFOPedia”



Some of our RFOs Follows-up

Contributions 🔨💙🎉

It isn’t just a DOCUMENT but a CULTURE!



System breaks!

Incident management and “firefighting”

Postmortem analysis
Write “RFO document”

Publish the RFO making it available for the whole company.

We keep reading, reviewing and learning from past RFOs and eventually come up with “recommendation”, “standard procedures” and tools”

Questions ?

Thank you!

martin.alderete@booking.com
ioannis.androulidakis@booking.com

https://jobs.booking.com/booking/jobs

Booking .com

Other presentations

- MySQL Keynote: Oracle OpenWorld 2018
- Booking.com: Reliable Operations and Rapid Development with MySQL
- Running MySQL on Three Different Platforms at Booking.com

- and more...