Bring Consensus to Data Replication

QCon 2017

Meng Wang

UBER

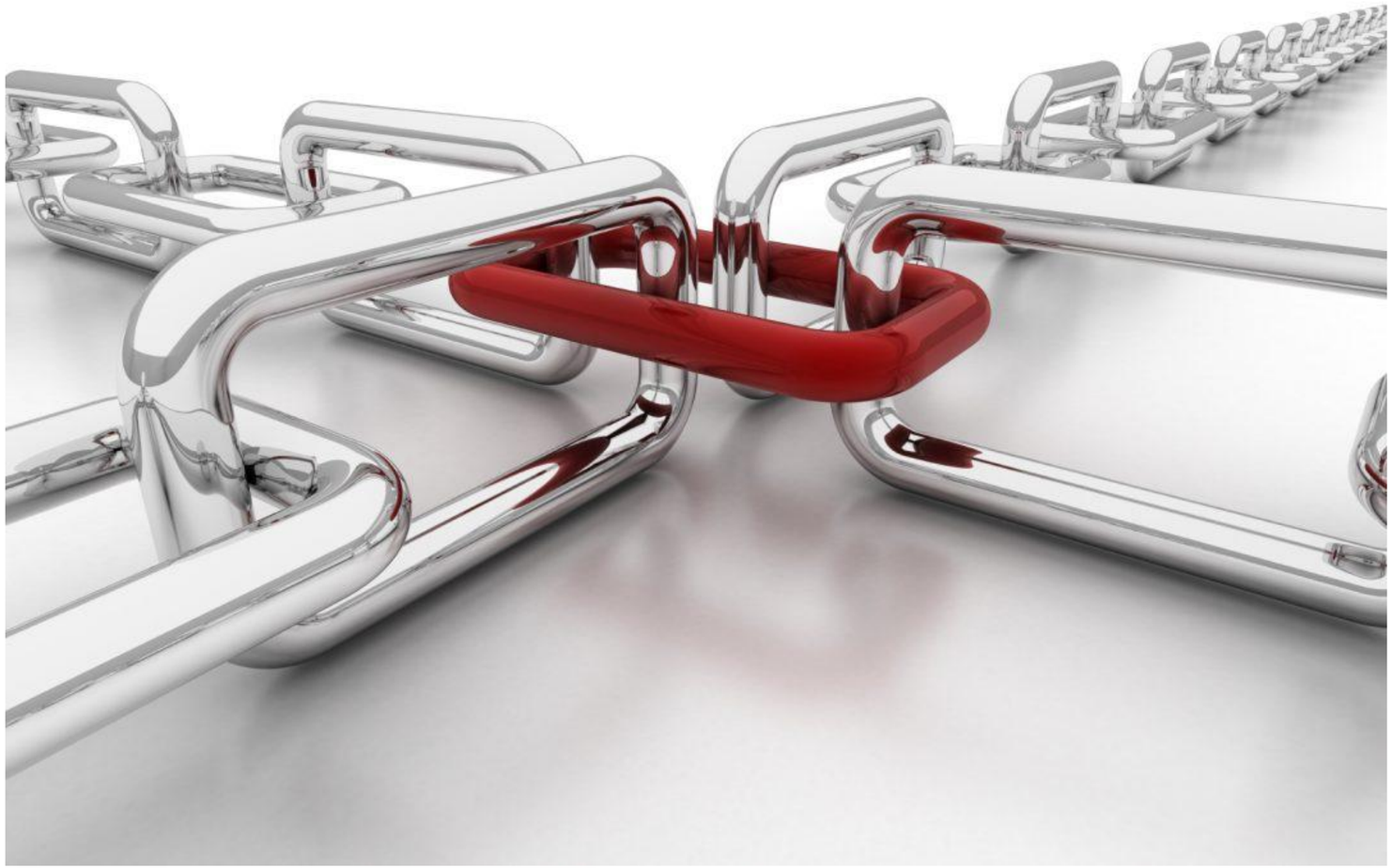
DISTRIBUTED SYSTEM

- Resource sharing
- Openness
- Concurrency
- Scalability
- Fault Tolerance
- Transparency

DISTRIBUTED SYSTEM

- Resource sharing
- Openness
- Concurrency
- Scalability
- Fault Tolerance
- Transparency

SINGLE POINT OF FAILURE



REDUNDANCY

Adding more replicas

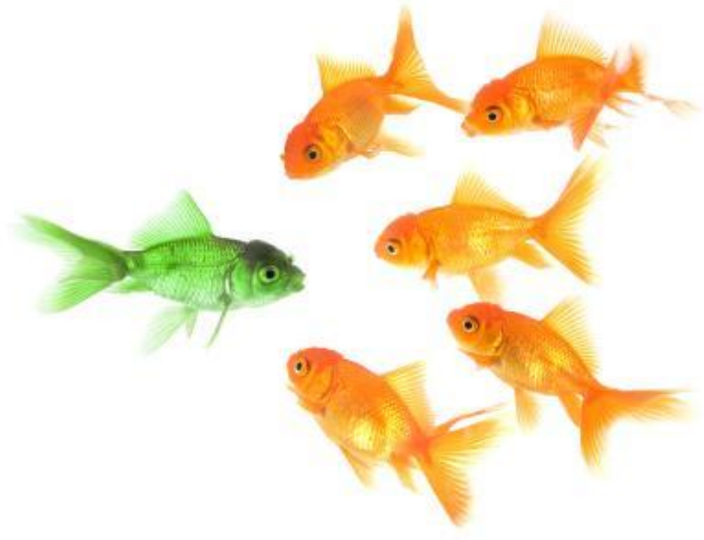
- scale out the reads
- resilient to machine crashes
- more concurrency

CHALLENGES

consistency

synchronization

fence off stale write



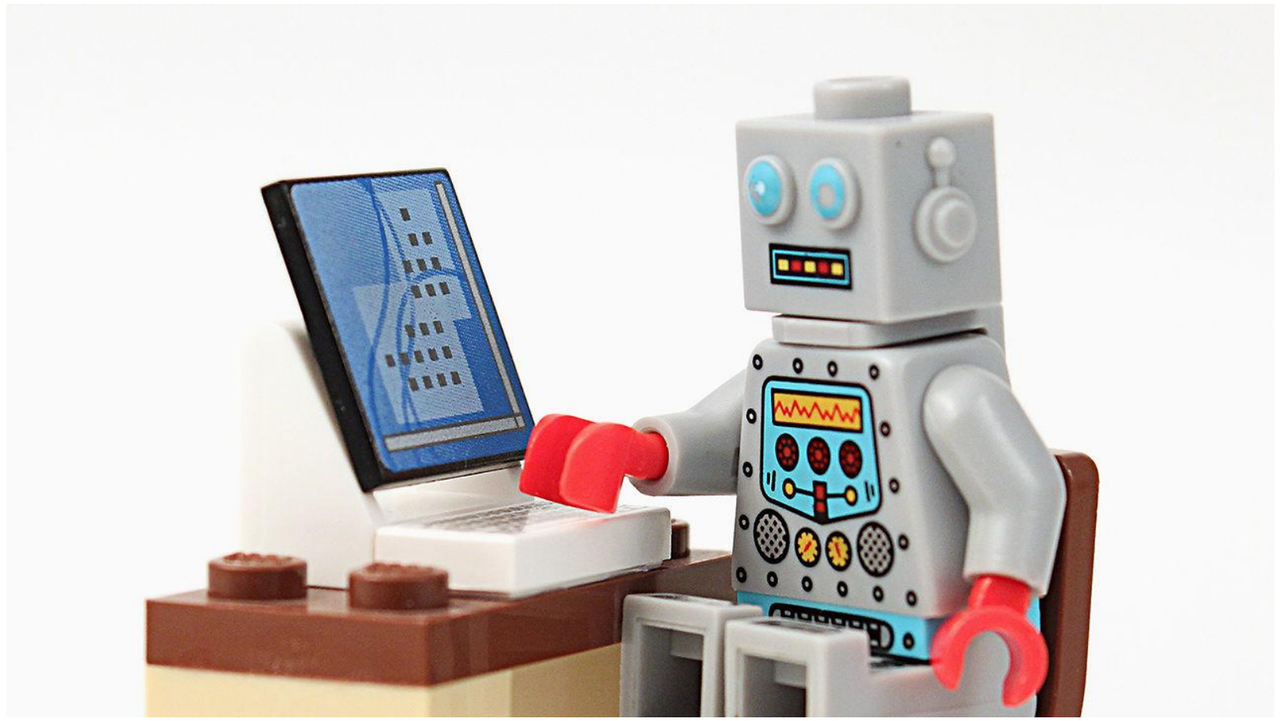
OUR FOCUS

Focusing on our four 9s’ availability

Availability %	Downtime per year	Downtime per month	Downtime per week	Downtime per day
99% (“two nines”)	3.65 days	7.20 hours	1.68 hours	14.4 minutes
99.9% (“three nines”)	8.76 hours	43.8 minutes	10.1 minutes	1.44 minutes
99.99% (”four nines”)	52.56 minutes	4.38 minutes	1.01 minutes	8.66 seconds

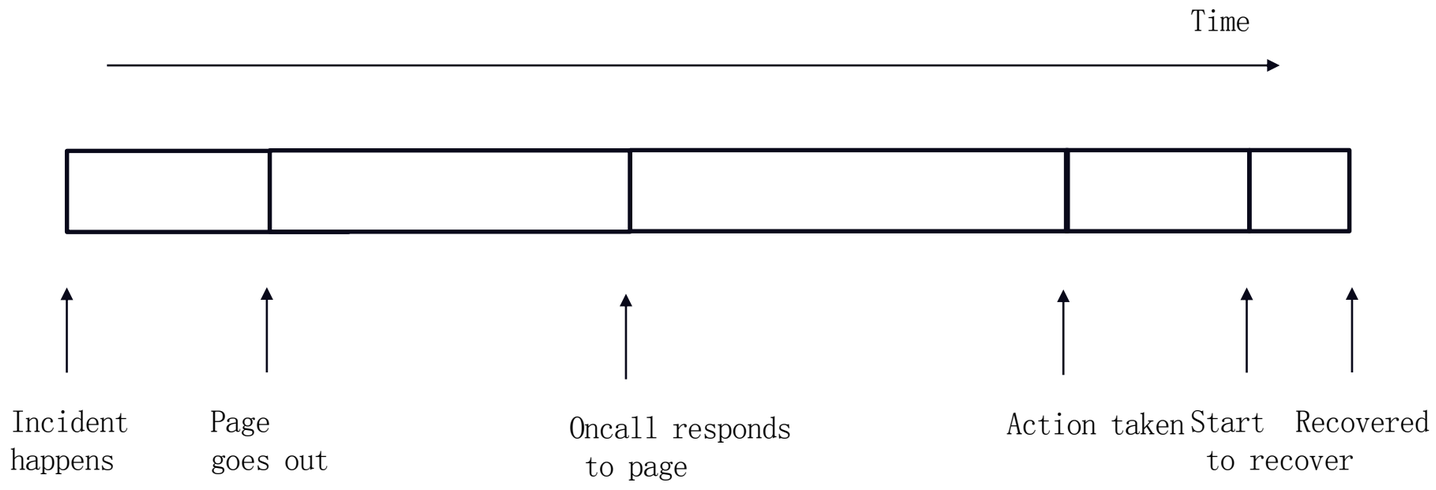
OUR FOCUS

Take human out of the equation



OUR FOCUS

Take human out of the equation



OUR FOCUS

Take human out of the equation

- System reacts to a subset of failure modes CORRECTLY
- Human makes judgement call in the disastrous scenarios.

OUR FOCUS

Linearizability

- If a write to a key (which identifies a piece of data) is successfully applied, all the subsequent reads via the same key must return the same data written by this particular write or some later write;
- If a read of a key returns some data, all subsequent reads via the same key must return the same data or some data from some later write.

METHODOLGY

- Replicated state machine
  - A type of active replication
  - All replicas receive and process the same sequence of client requests

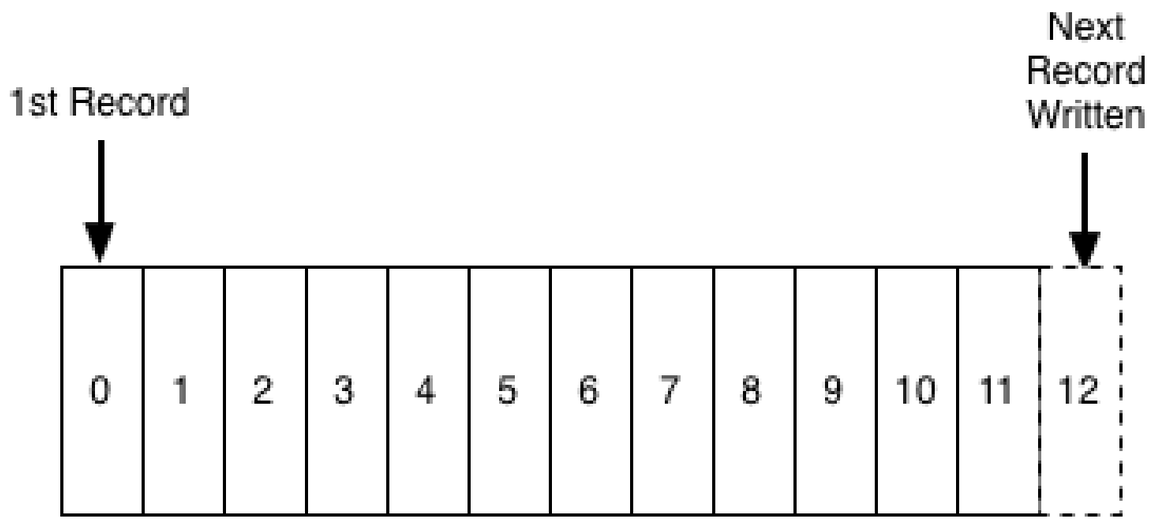
- Consensus protocol to maintain the replicated log consumed by replicas
  - Elect leader automatically
  - Tolerate node failure (non-Byzantine failures)
  - Maintain consistency among nodes

REPLICATED STATE MACHINE

- Deterministic state machines running on a collection of servers.
- Each state machine computes identical copies of the same data.
- The system can continue to operate even if some servers are down.

REPLICATED LOG

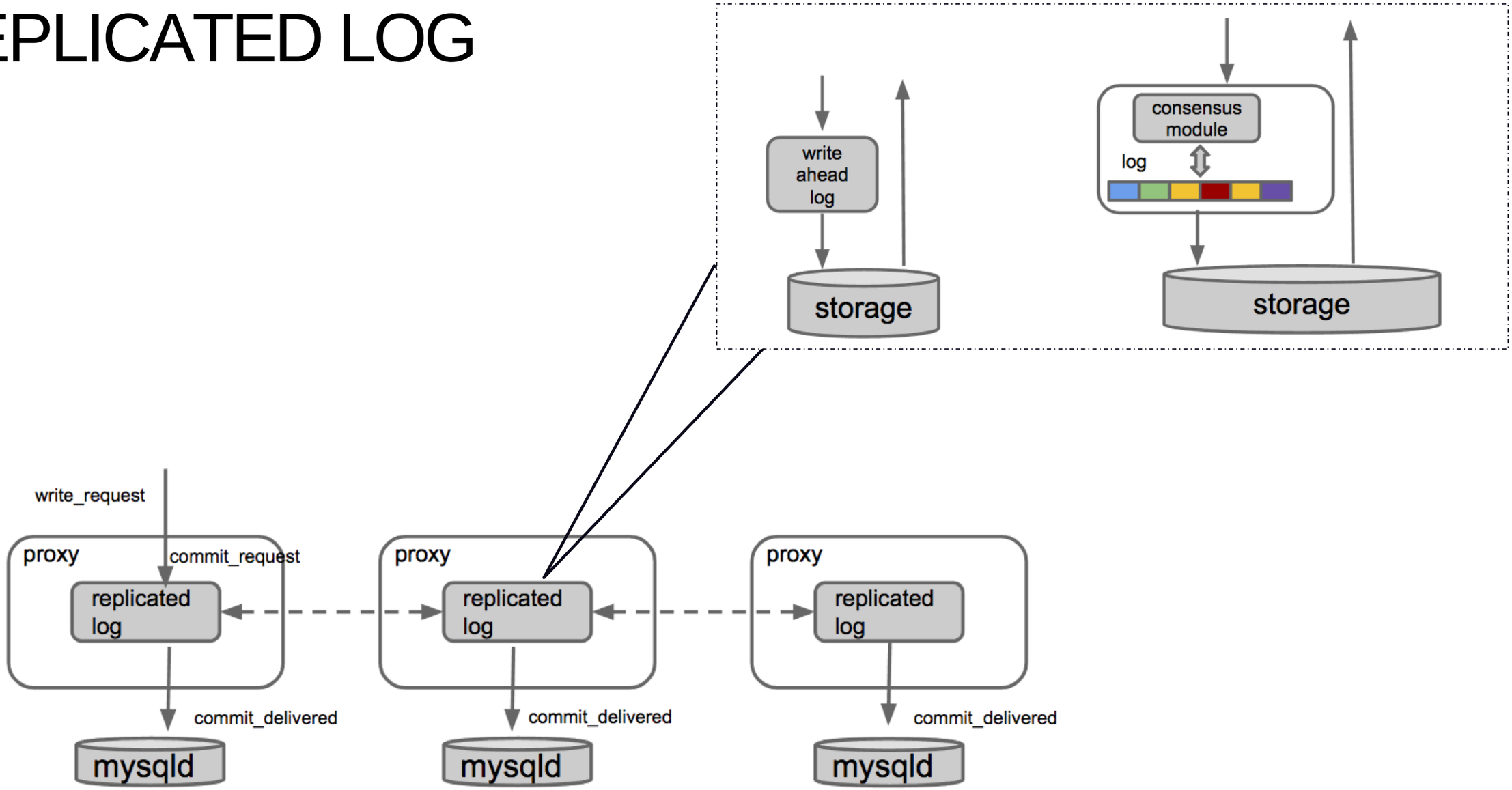
- Log is an append-only and totally ordered abstraction.
- Log replication makes every replica see the exact order of entries.
- If we treat every write to DB as an entry in the log, applying these entries on the same starting snapshot in the same order will yield same ending snapshot.
- Replicated log is maintained by consensus protocol.



Replicated State Machine

- Each server stores a log containing a series of commands which are executed in order by the state machine.
- Each log contains the same commands in the same order.
- State machine is deterministic so each computes the same state and has the same sequence of outputs.

REPLICATED LOG

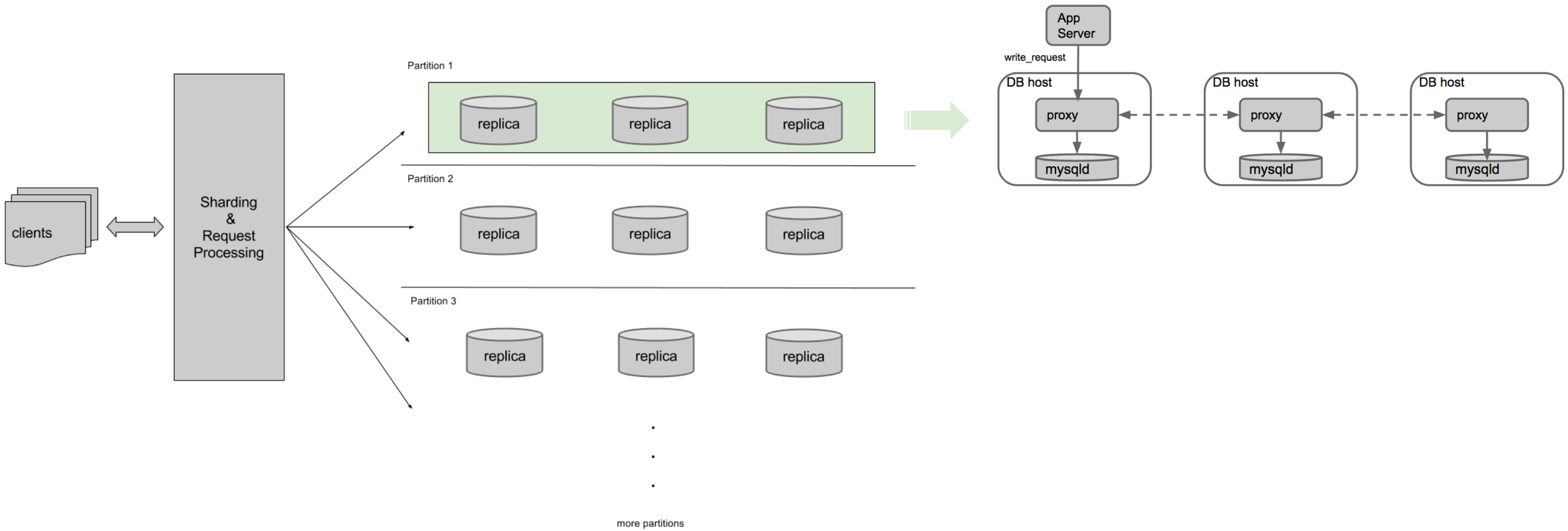


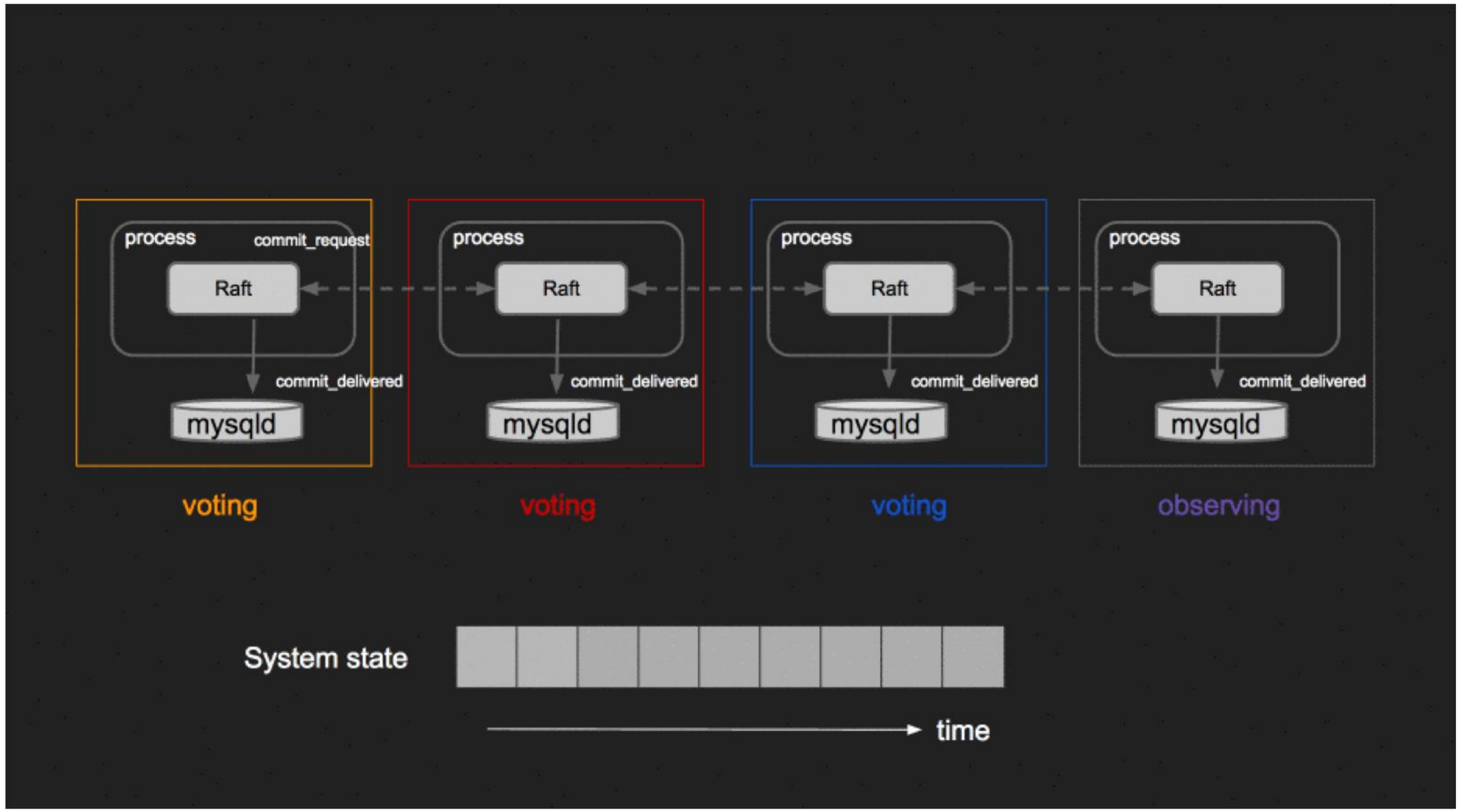
RAFT

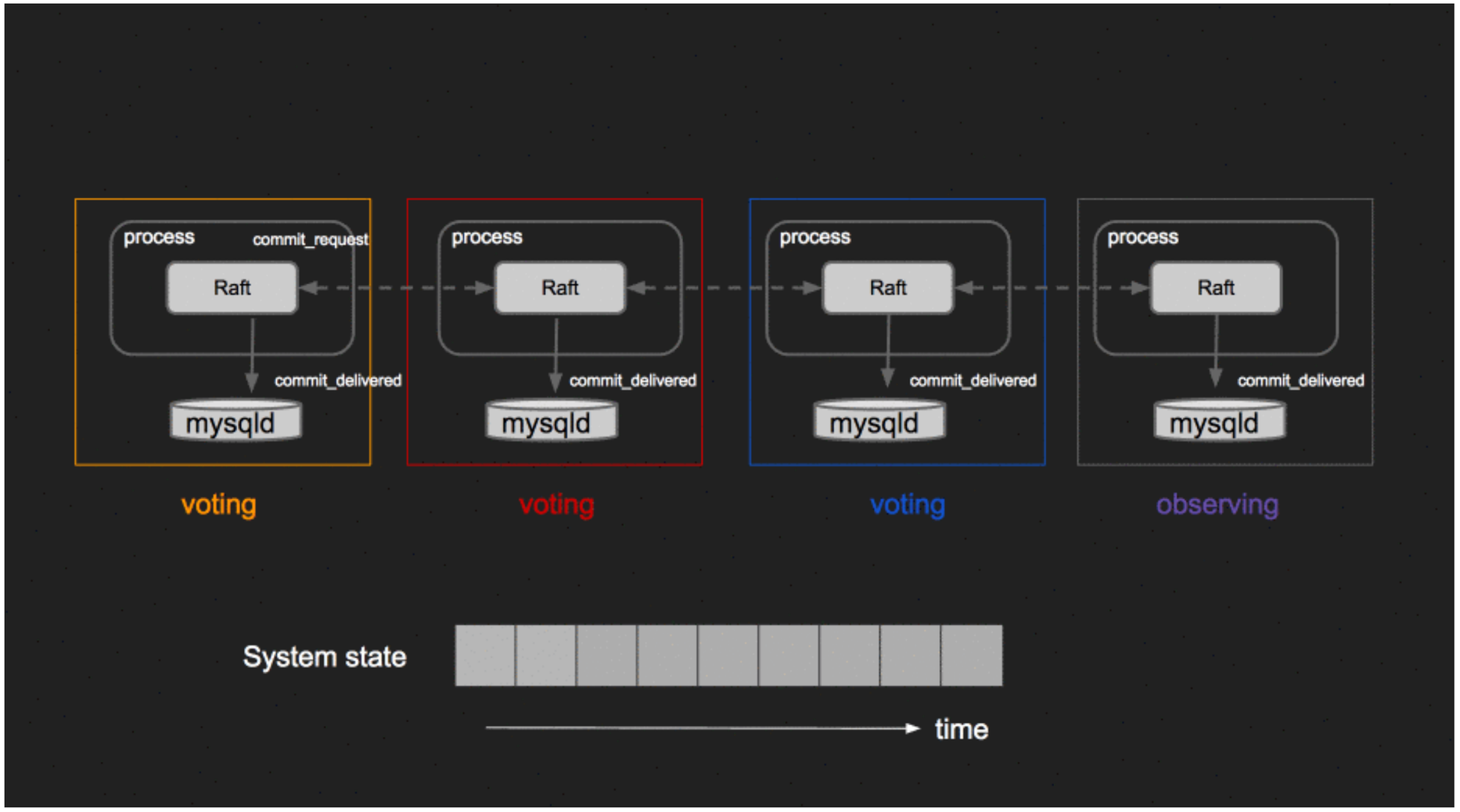
- Election Safety
  At most one leader can be elected in a given term.
- Leader Append-Only
  A leader never overwrites or deletes entries in its log; it only appends new entries.
- Log Matching
  If two logs contain an entry with the same index and term, then the logs are identical in all entries up through the given index.
- Leader Completeness
  If a log entry is committed in a given term, then that entry will be present in the logs of the leaders for all higher-numbered terms.
- State Machine Safety
  If a server has applied a log entry at a given index to its state machine, no other server will ever apply a different log entry for the same index.

SYSTEM SET-UP

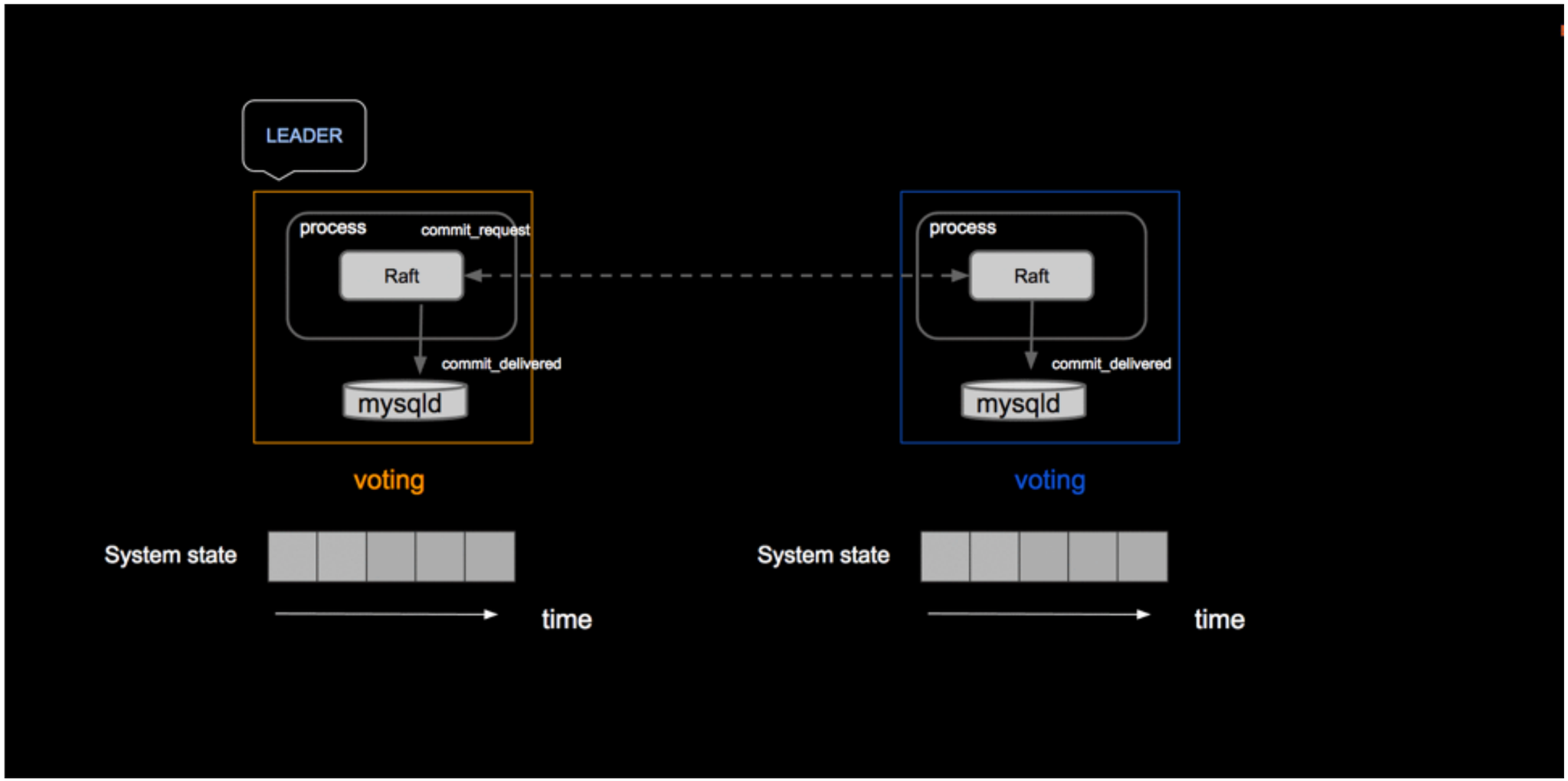
Note that we don't have a 'global' Raft set-up for the entire system; rather the data is partitioned into multiple partitions each of which has its own Raft set-up. Sharding is out of scope of this talk but the overall system architecture looks like the following.



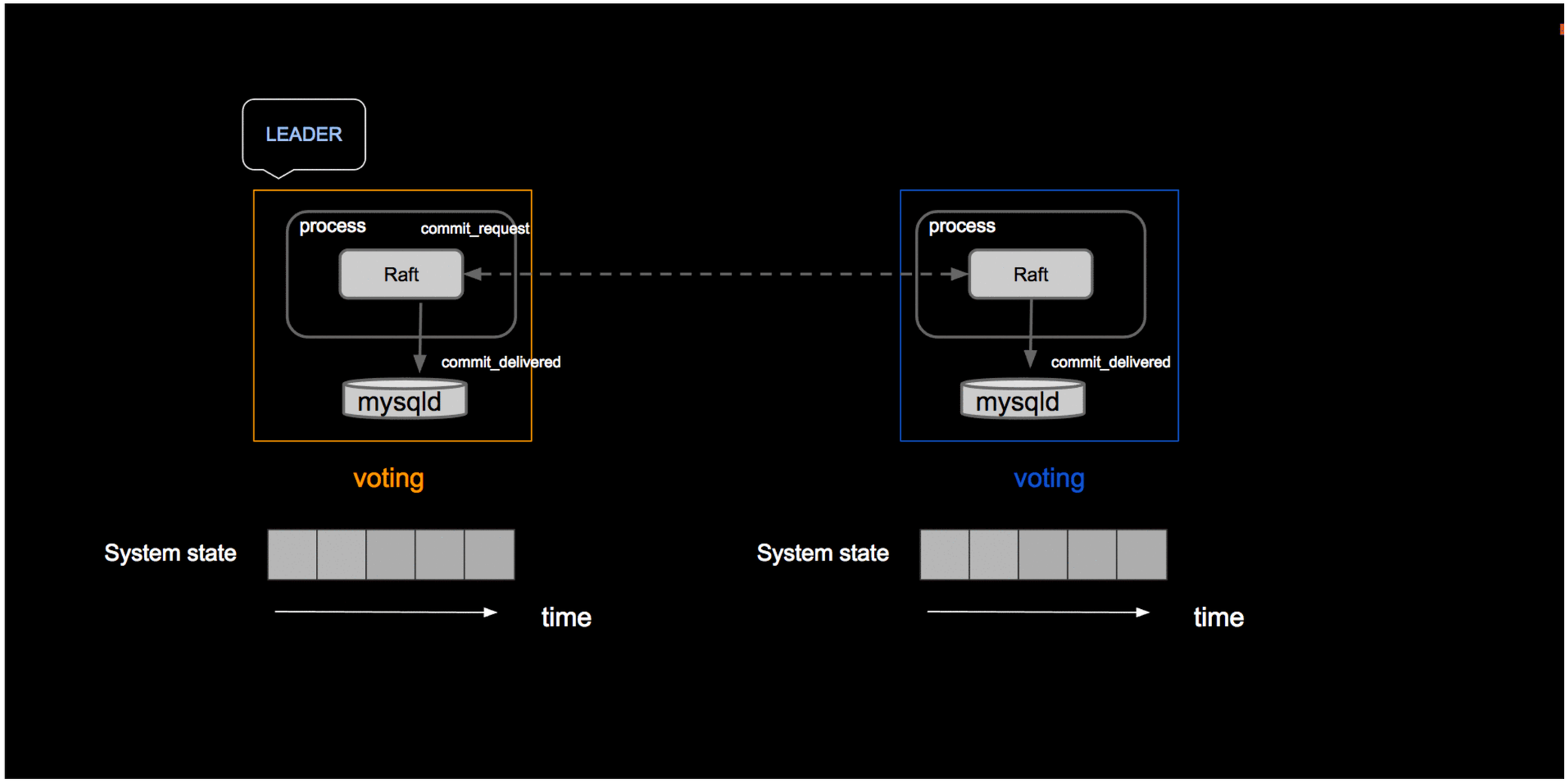

process
commit_request
Raft
commit_delivered
mysqld
voting
process
Raft
commit_delivered
mysqld
voting
process
Raft
commit_delivered
mysqld
voting
process
Raft
commit_delivered
mysqld
observing
System state
time


process
commit_request
Raft
commit_delivered
mysqld
voting
process
Raft
commit_delivered
mysqld
voting
process
Raft
commit_delivered
mysqld
voting
process
Raft
commit_delivered
mysqld
observing
System state
time

STRONGLY CONSISTENT READ



STRONGLY CONSISTENT READ



INTEGRATION WITH MYSQL

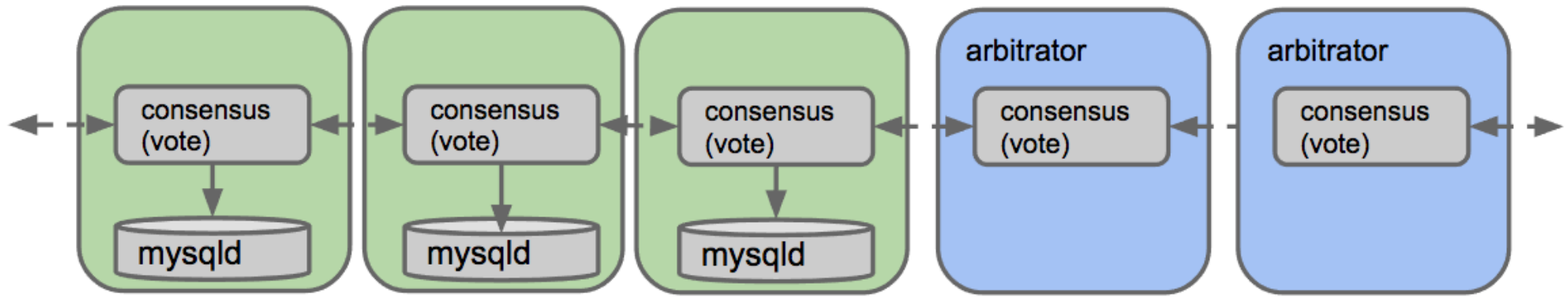
- Statement Idempotency
  - There may be non-idempotent SQL statements. Even if the replicated log is consistent we cannot allow same statement to apply twice.
  - Our solution: using Raft entry index to construct the GTID
- Parallel Execution
  - Raft essentially sequentializes everything which compromises the the parallelism if we would use Mysql directly.
  - Our solution: parallel execution SQL statements based on the shard that the statement targets at.
- Auto Increment ID
  - Mysql's auto increment ID is NOT transactional so we can NOT have individual replica use its own auto increment ids.
  - Our solution: use Raft to maintain the cluster-wide auto increment ID.

RAFT EXTENSIONS

- Observer
  A participant which doesn’t vote nor solicit a vote
- Arbitrator
  A participant which acts as a voter but will relinquish leadership (if allowed by the consensus protocol) if it itself is elected as the leader.
- Leader Lease
- DC awareness/Efficient cross DC data streaming

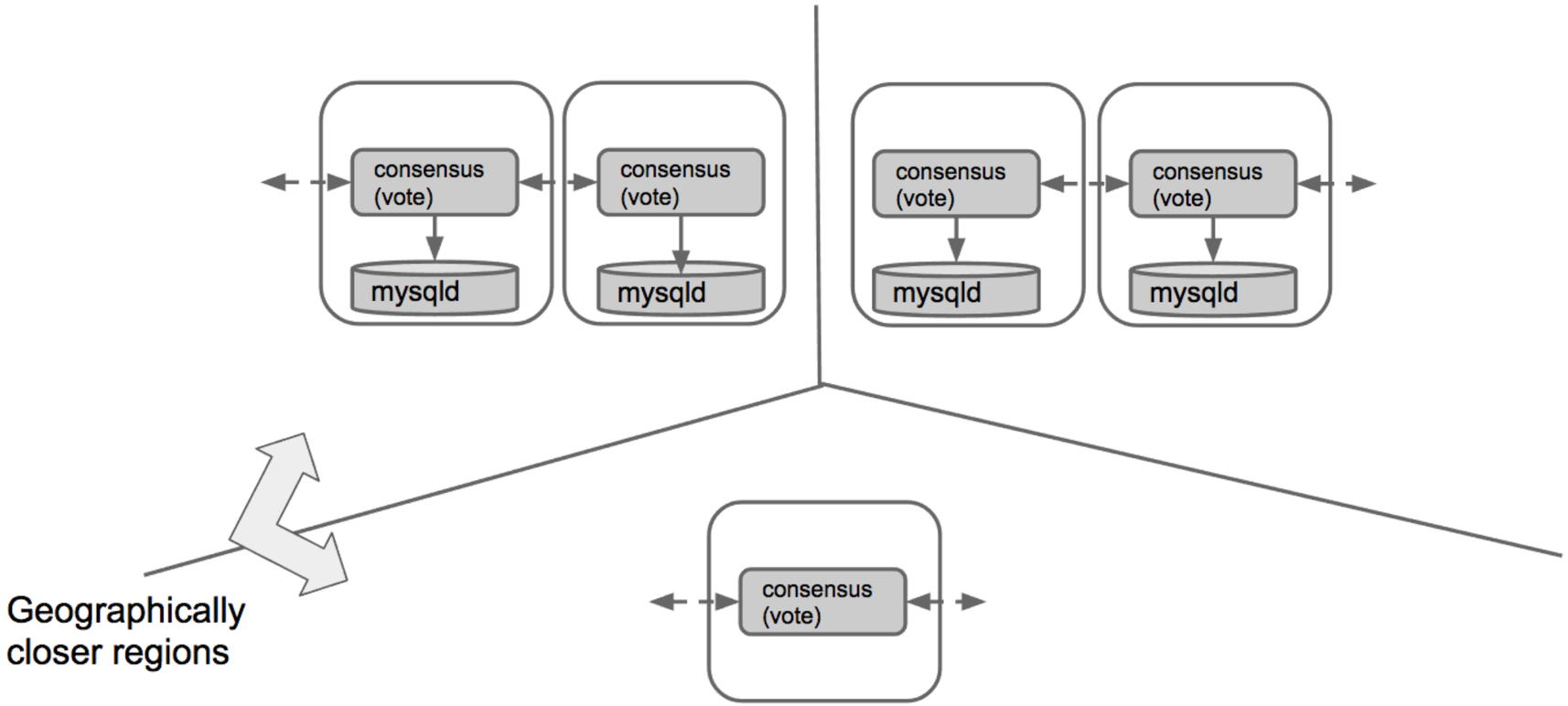
ARBITRATOR

Within one DC we can achieve tolerance of two nodes being down with the cost of 3 copies of data, which is not possible with the vanilla 3-voter set-up.

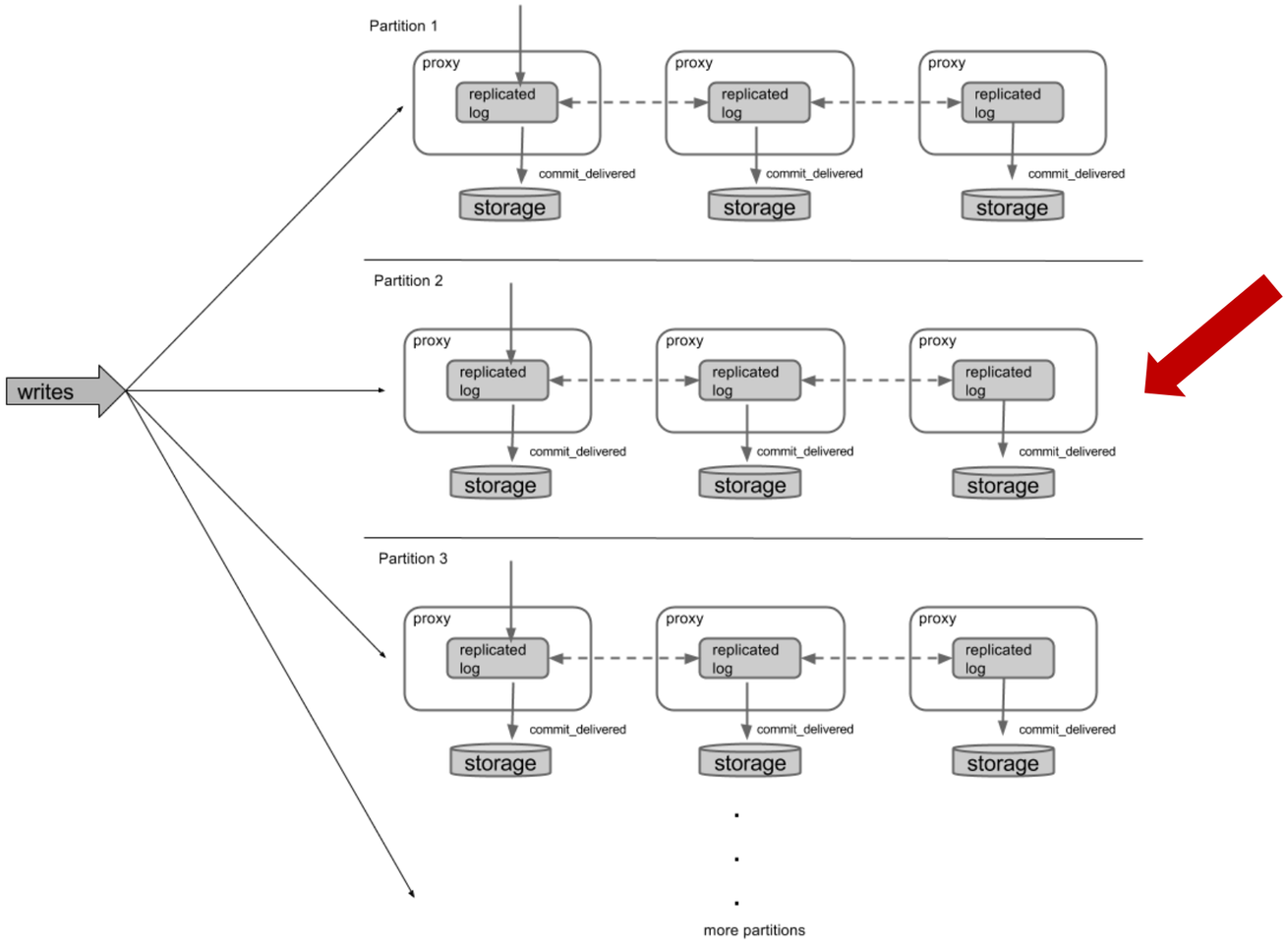


CROSS DC SET-UP

	Leader and its majority are in the same DC	**Leader and its majority are in DIFFERENT DCs.**
Requester and leader are in the same DC	Intra-dc latency	one xdc roundtrip
Requester and leader are in DIFFERENT DCs	one xdc roundtrip	two xdc roundtrips



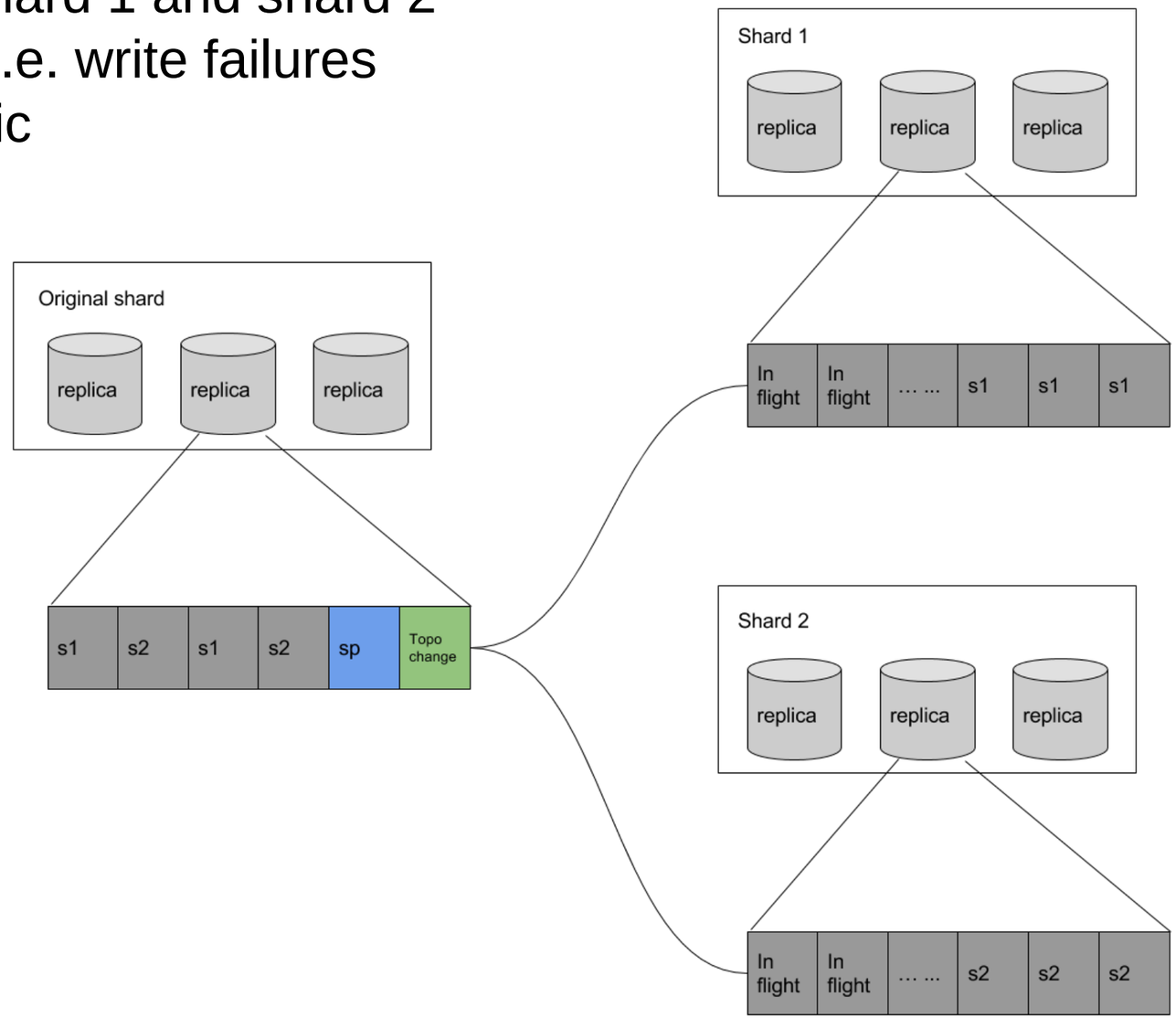
SHARD SPLITTING



Extending capacity??

SHARD SPLITTING

Split original shard into shard 1 and shard 2
- Very short ‘downtime’, i.e. write failures
- No need to stop the traffic
- Split is idempotent



FUTURE WORK

Coordinated Transaction among Replicas

- Cross-shard transactional update

Multiplexing Raft

- Move shard around Raft clusters

THE TEAM

SCOTT CEDERBERG

SHAMIM MOHAMED

HIMANK CHAUDHARY

Mitesh konjeti

LI LI

MENG WANG

https://www.uber.com/careers/

THANK YOU

On behalf of Uber Storage Platform