Pivotal

Raft in RabbitMQ

Daniil Fedotov
GitHub: hairyhum
Twitter: @hairyhum_

Original talk

Spring One Platform 2017

Karl Nilsson @kjnilsson
Michael Klishin @michaelklishin

Pivotal

Pivotal

Pivotal and RabbitMQ

Invested in RabbitMQ

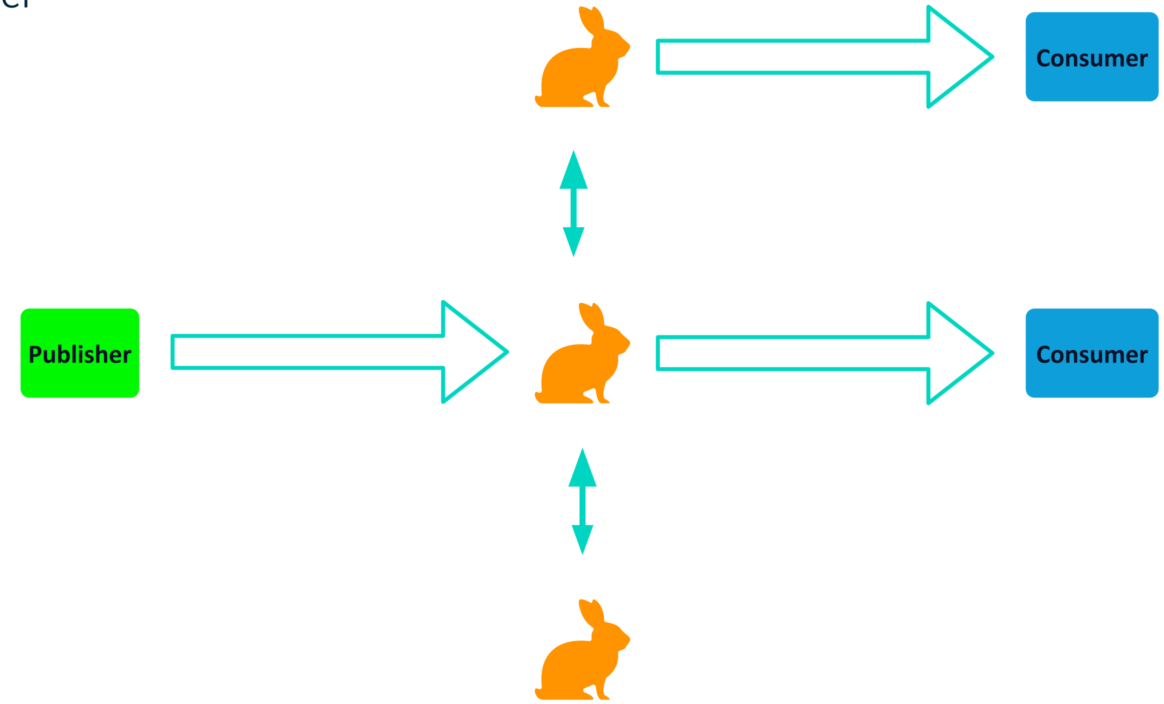
- Sponsors RabbitMQ development
- Provides RabbitMQ services as part of the Cloud Foundry platform
  - RabbitMQ “tile”
- Provides commercial support for RabbitMQ
- https://www.rabbitmq.com/services.html

The following feature may never happen. This is R&D. No promises.

What is RabbitMQ?

What is RabbitMQ

- Message broker



What is RabbitMQ

- Messaging broker
- Multi-protocol (AMQP 0-9-1, AMQP 1.0, MQTT, STOMP, ...)
- Started in 2006
- Broad ecosystem including Spring support
- Learn more at rabbitmq.com

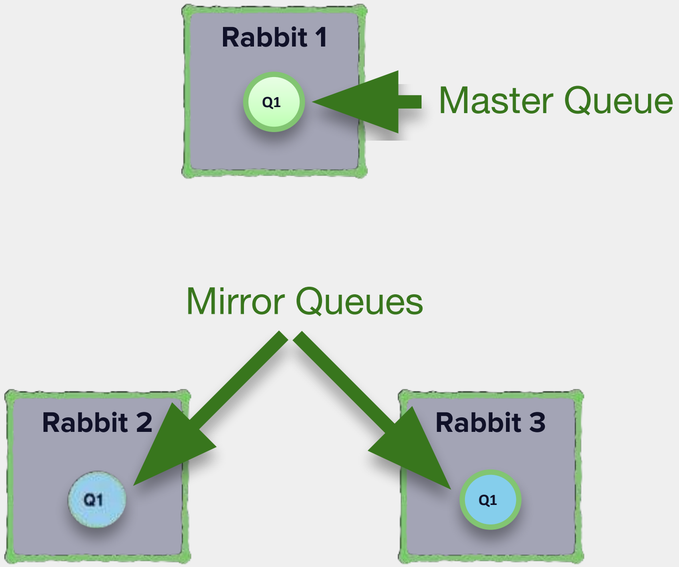
RabbitMQ nodes can form clusters

- Balance load (connections, traffic, I/O, ...) between nodes
- Replicate queue contents
- Tolerate node failures

High Availability in RabbitMQ

RabbitMQ High Availability

- Replication of data and operations
- Message replication is done at the queue level
- Called “Queue Mirroring”
- In a cluster of RabbitMQ nodes a queue can have a mirror on one or more nodes
- Provides fail-over and redundancy



RabbitMQ Queue Mirroring

- Internally uses a component called “Guaranteed Multicast” to replicate queue operations and message data
- Provides replication and total ordering of operations
- Ordering matters:

[ENQ + ENQ + DEQ] != [DEQ + ENQ + ENQ]

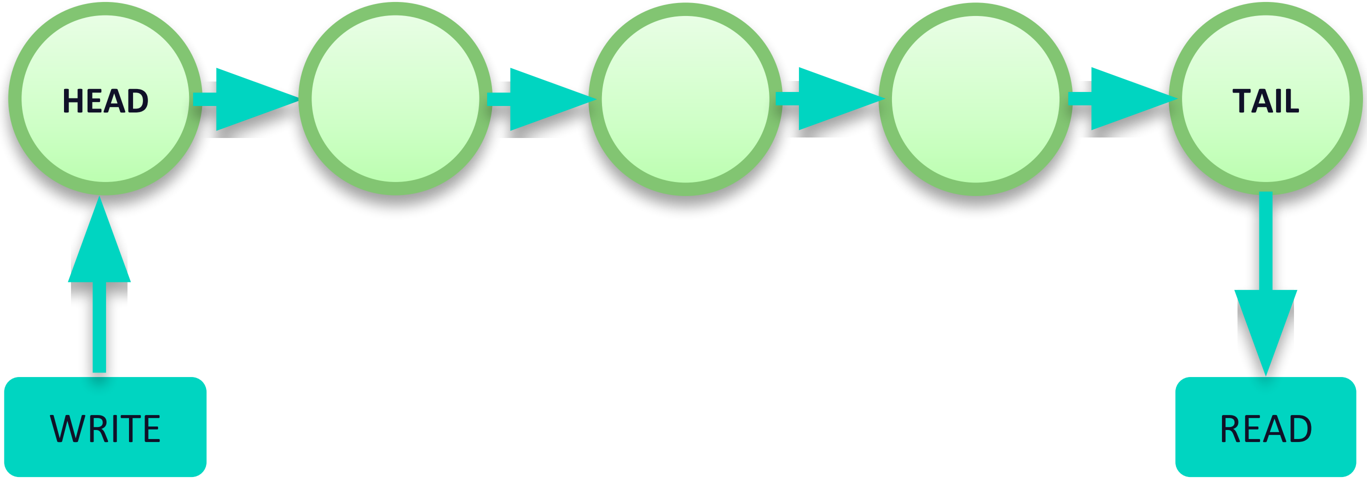
Chain Replication

Chain Replication ensures strong consistency and good availability guarantees in "fail-stop" scenarios.

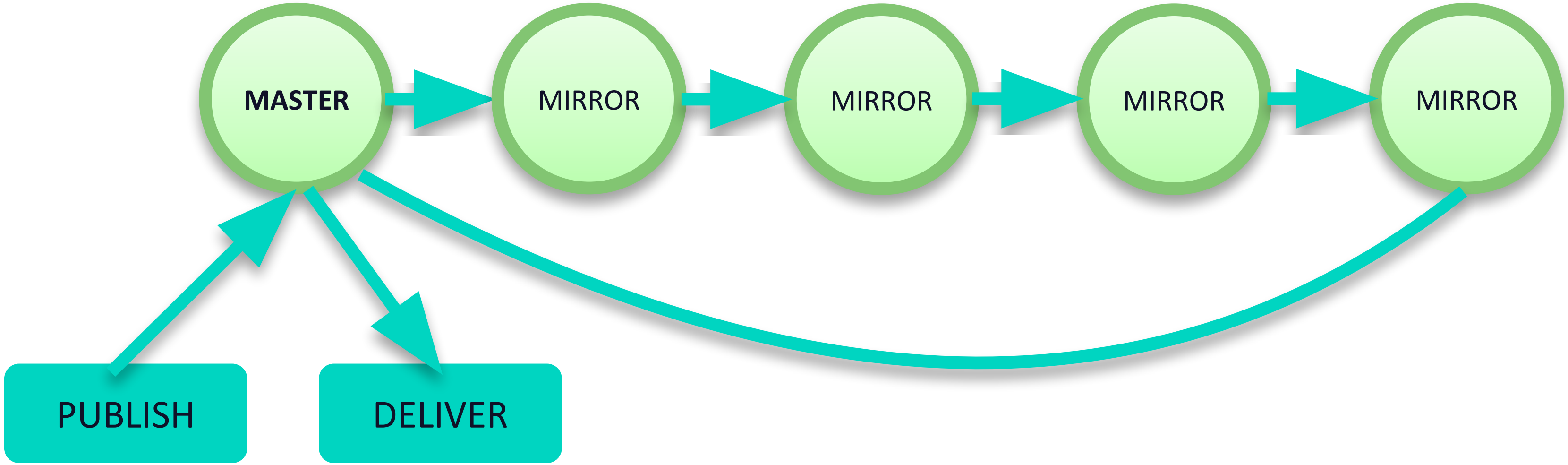
Chain Replication for Supporting High Throughput and Availability (Robbert van Renesse, Fred B. Schneider)

http://www.cs.cornell.edu/home/rvr/papers/OSDI04.pdf

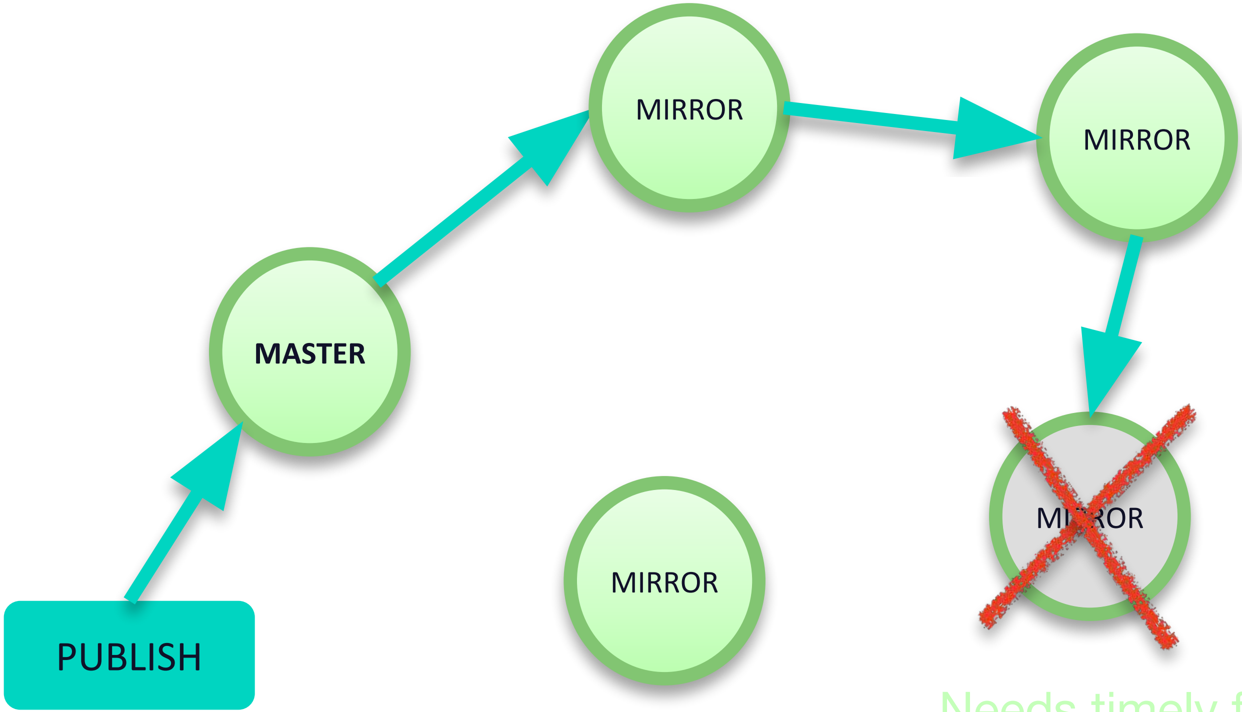
Chain Replication



RabbitMQ mirrored queue ring

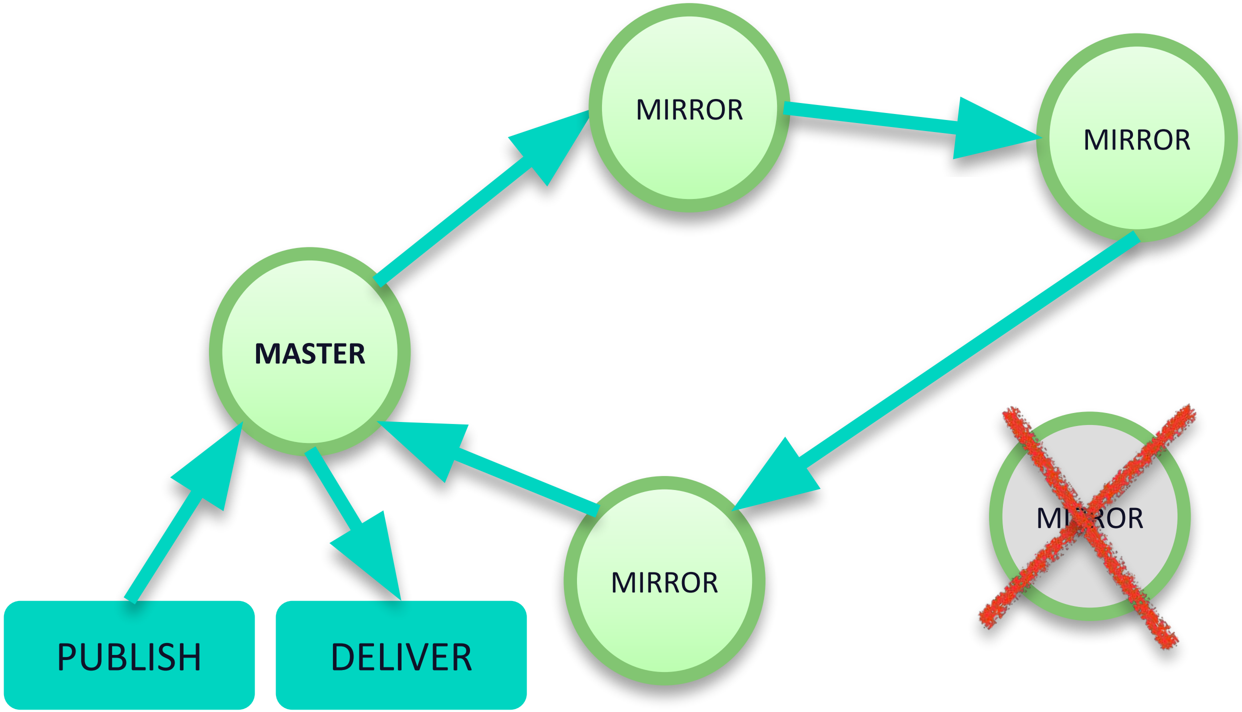


RabbitMQ Queue Mirror failure detection

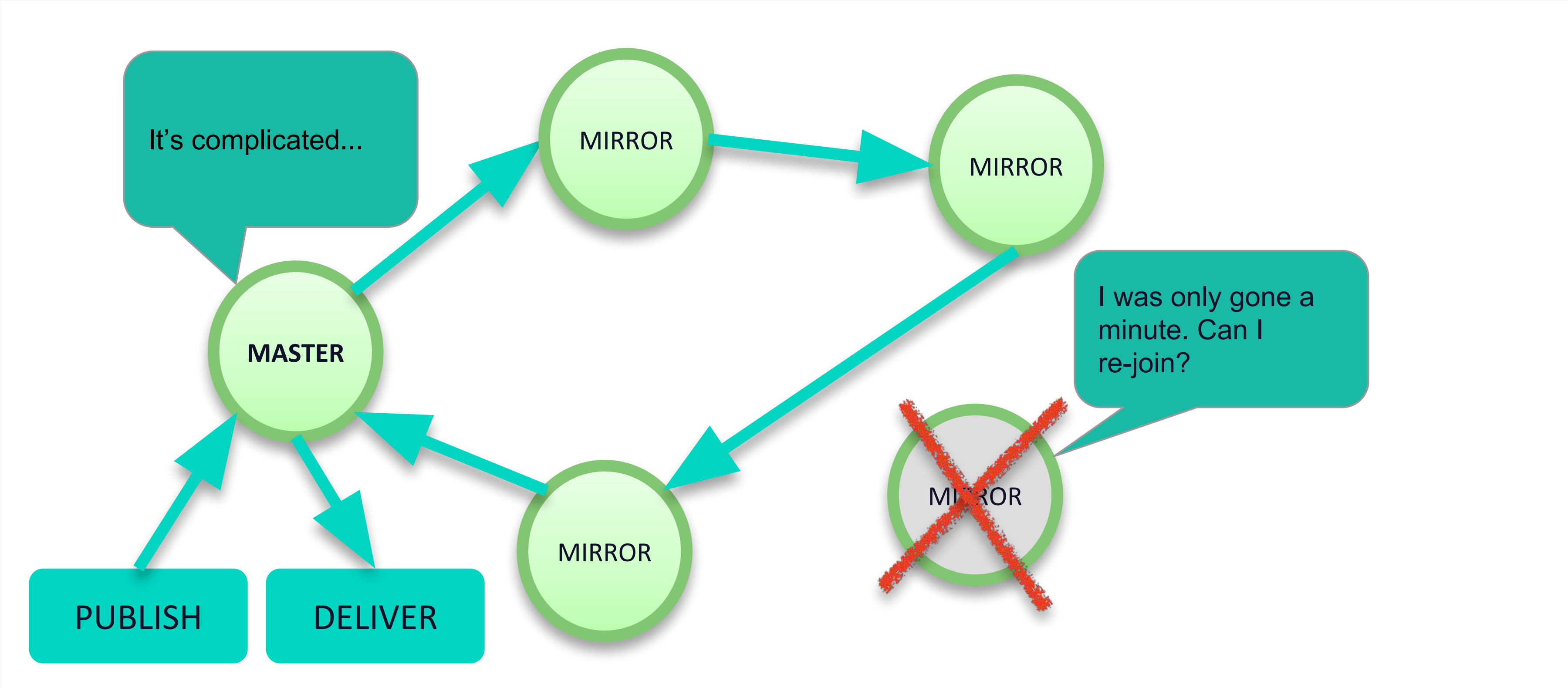


Needs timely fault detection

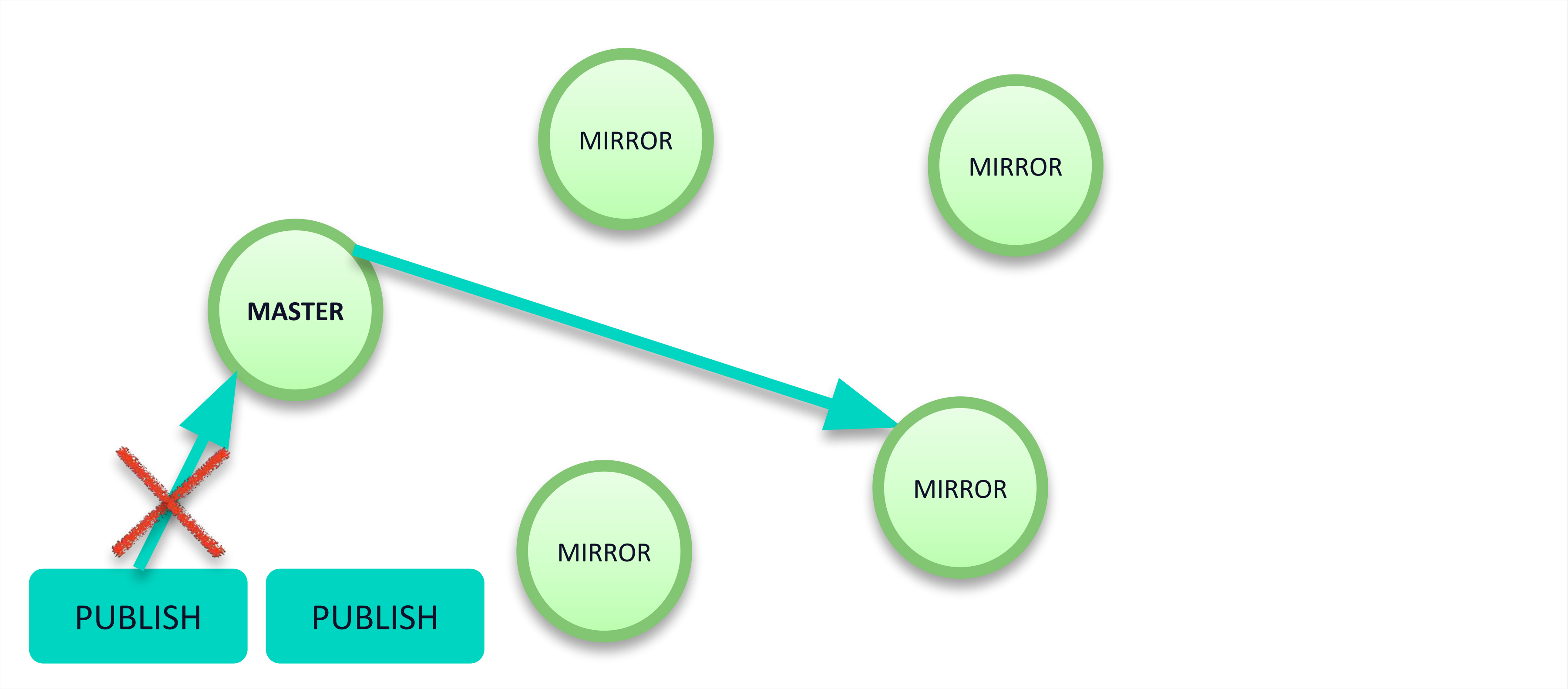
RabbitMQ Mirrored Queue reforms ring



RabbitMQ Queue reforms ring

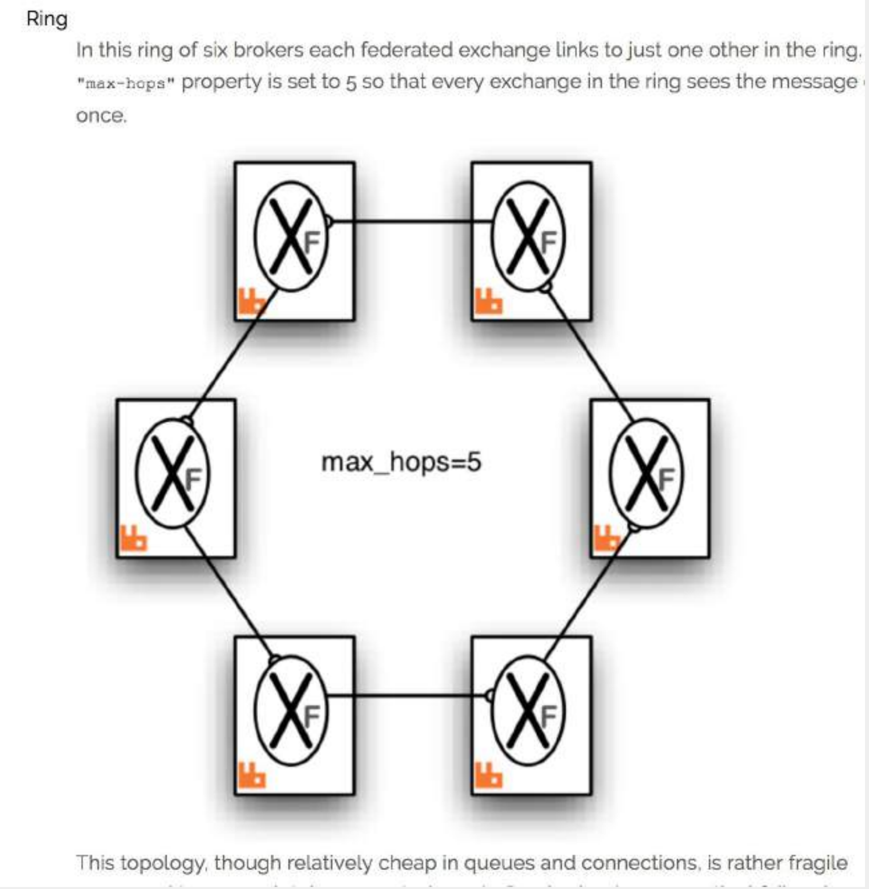


RabbitMQ Queue mirror sync



RabbitMQ documentation wisdom

This topology [Ring] through relatively cheap in queues and connections, is rather ***fragile***



Ring

In this ring of six brokers each federated exchange links to just one other in the ring. "max-hops" property is set to 5 so that every exchange in the ring sees the message once.

This topology, though relatively cheap in queues and connections, is rather fragile

RabbitMQ Queue Mirroring

- Sensitive to network partitions
- Recovery can cause a queue sync (blocking)
- Recovery can cause message loss (jepsen test)
- Replication is a linear algorithm
- Availability relies on fault-detection (which is hard)
- Distributed systems are hard

We can do better!

Road to Raft

- We need stronger consistency guarantees and totally ordered operations
- Predictable behaviour during failure scenarios
- Safe queue master “fail-over”
- Better availability during recovery

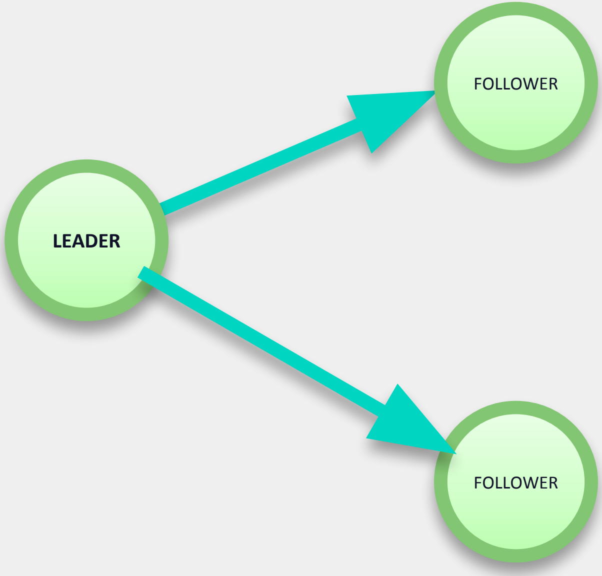
Pivotal

Raft

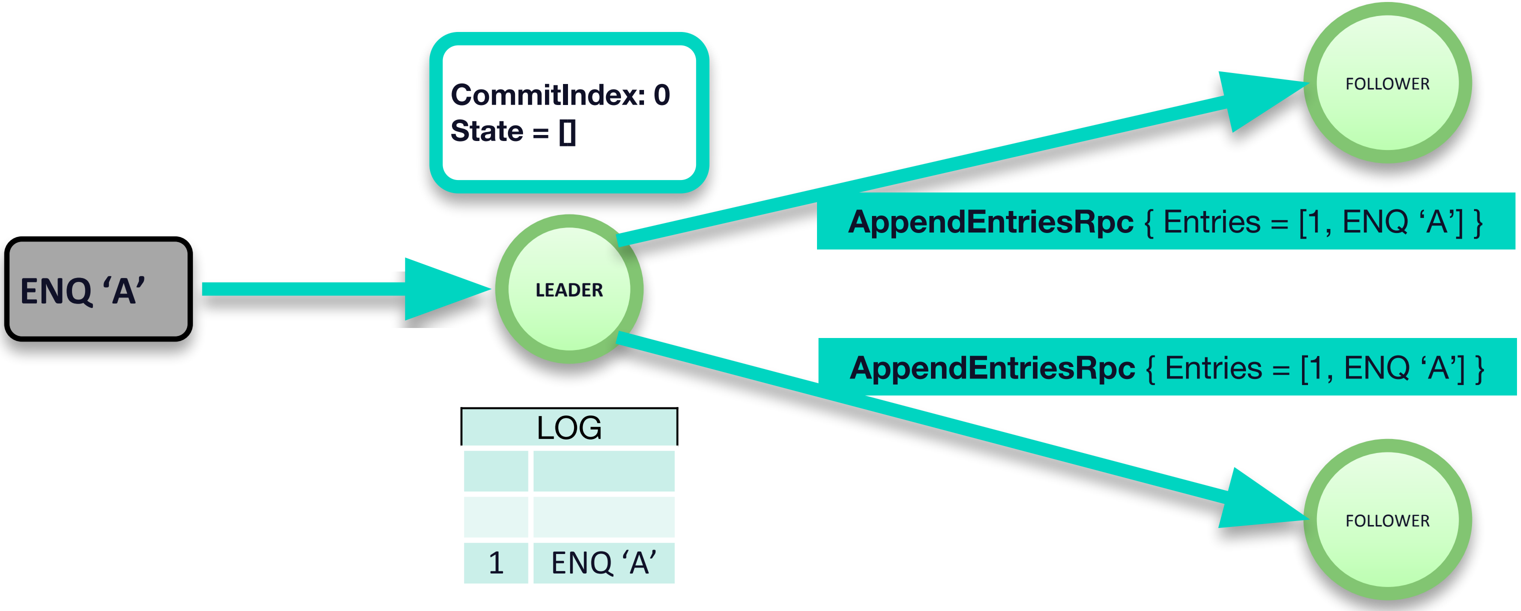
- A group of algorithms for reaching consensus in a distributed system
- Similar problem space to RabbitMQ queue mirroring
- Oriented towards implementers
- Proven
    - Multiple implementations
    - Industry use
        - etcd
        - Consul
        - CockroachDB
    - TLA+ specification
- Requires no external dependencies

Raft provides

- A state machine log abstraction
- Leader-follower model
- State machine log replication
- Well-defined algorithms important for implementers
- Recovery



Raft protocol: replicate entry



CommitIndex: 0
State = []

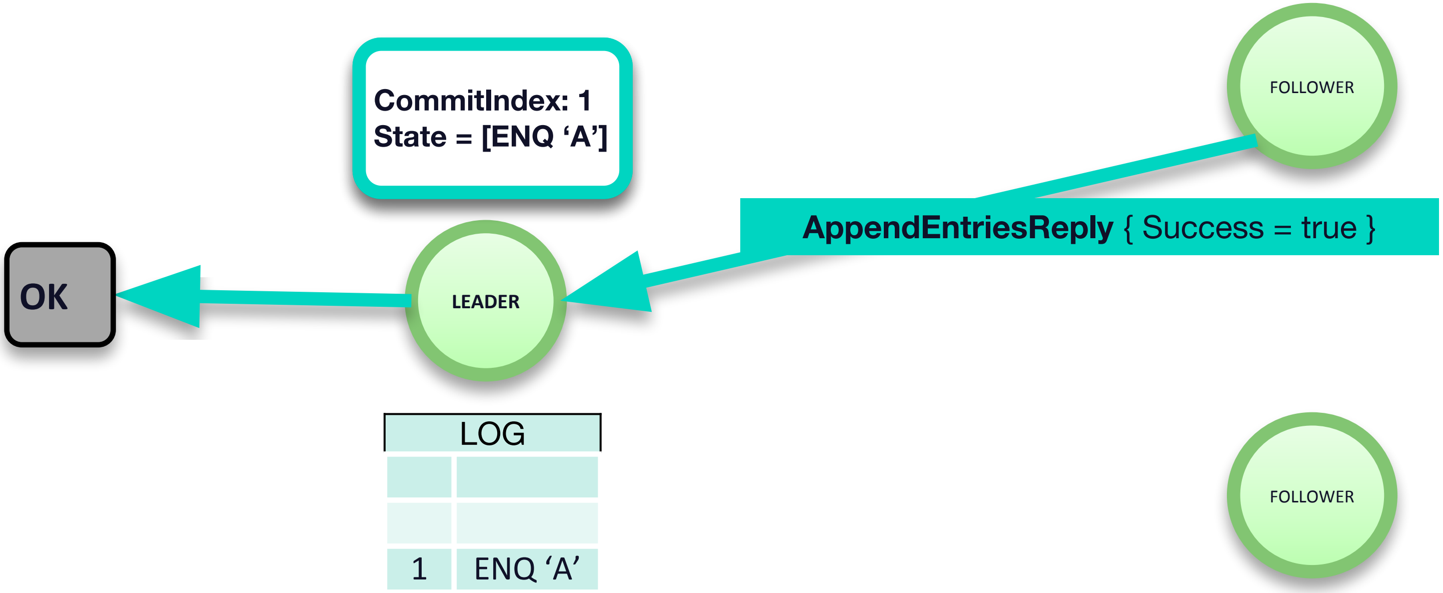
ENQ ‘A’

AppendEntriesRpc { Entries = [1, ENQ ‘A’] }

AppendEntriesRpc { Entries = [1, ENQ ‘A’] }

LOG	
1	ENQ ‘A’

Raft protocol: reply + commit



CommitIndex: 1
State = [ENQ ‘A’]

AppendEntriesReply { Success = true }

LOG	
1	ENQ ‘A’

A raft

Learn more at raft.github.io

How does Raft compare

Taking action

What to do when detecting a (potential) failure?

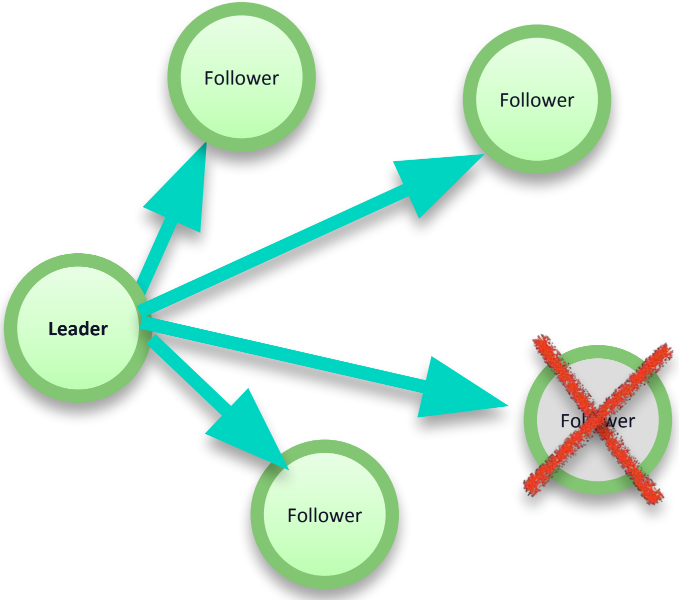
A. Nothing
   - most reliable / least useful

B. Try to “fix stuff”
   - evict down nodes, reform topology
   - communicate changes to other nodes

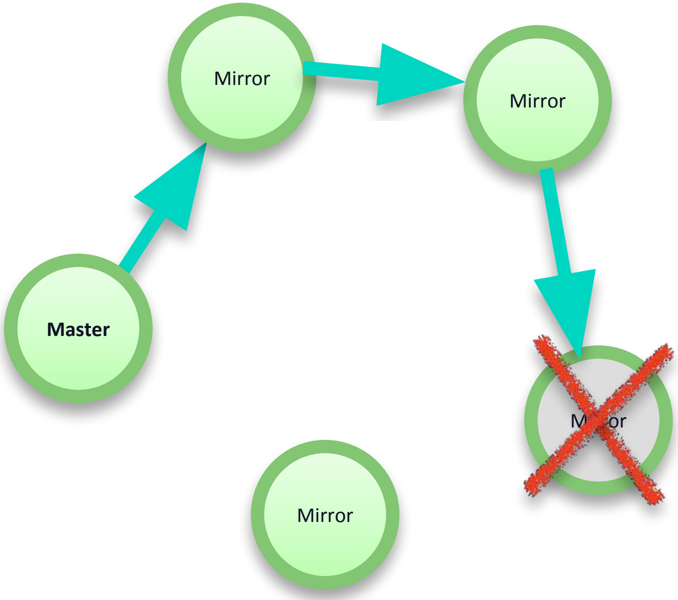
C. The minimum required
   - regain / retain availability and consistency

Raft vs Queue Mirroring failure handling

Raft



Queue mirroring



In response to (potential) failure:

Raft either:

- Does nothing
- Does the minimum required
    - ensures consistency
    - regains availability

RabbitMQ queue mirroring:

- Must always do *something*
- Must coordinate taken action

Down the rabbit hole

Raft in RabbitMQ

- Can be adopted in multiple areas incrementally
- Area of focus: queue mirroring
- Coordination, leader election
  - Cluster federation
  - Shovel
  - Delayed message exchange
- Message store data replication
  - Messages
- Distributed data and state storage
  - Internal metadata store (vhosts, users, permissions, queues, ...)

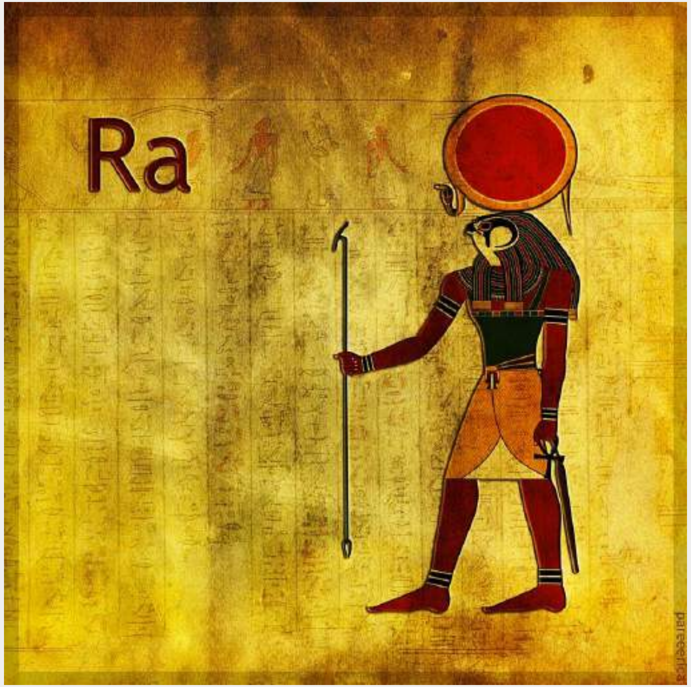
Raft challenges

- The cost of consensus
    - Raft requires “stable storage” (fsync)
    - Requires a quorum
- Cluster formation
    - Seeding
    - RabbitMQ internal concern
- Single leader
    - Scalability
- Uneven cluster sizes required / recommended
    - 3 nodes can tolerate 1 failure
    - 4 nodes can tolerate 1 failure (sic!)
    - 5 nodes can tolerate 2 failures, and so on…

Announcements

Ra: a Raft library

- By Team RabbitMQ
- Open source
- ASL2 / MPL1.1 licensed
- Generically usable, not tied to RabbitMQ
- Tailored for RabbitMQ needs
- Very much a WIP (breaking API changes are likely)
- https://github.com/rabbitmq/ra



Ra: implementation

- Raft cluster per queue
- Many queues = many Raft clusters

- Each node writing to it’s own log file
    - Thousands of concurrent fsync operations
    - A no-go - we tried it

- Raft is chatty
    - High background network usage when idle

Ra: implementation

- Shared Write Ahead Log (WAL)
    - fsync in batches
    - flushed to raft node specific storage periodically

- Storage engine
    - Similar to LSM tree and "append-oriented" stores (LevelDB, RocksDB)
        - Compaction is radically simpler in our case

- Per RabbitMQ node “heartbeat” process
    - Reduce network background usage

Ra: implementation testing

- Unit & integration testing
- Property-based testing □
  - Correctness is essential
- Deployment testing (BOSH)
- Jepsen test
- TLA+ spec for log implementation □

Your data safety is our priority

Quorum Queue

WIP Quorum queue properties

- Separate queue type (queue args)
- Designed not to lose messages as long as more than half the RabbitMQ nodes can still be recovered. Strongly consistent. (with publisher confirms)
- Implemented as a Raft replicated state machine
- Replicated to all RabbitMQ nodes (no ha- policies)
- Designed to be available as long as a quorum of RabbitMQ nodes are reachable. (no queue sync)

WIP Quorum queue trade-offs

- Trades latency for throughput
- Limited features set
  - Doesn’t support policies (maybe ttl, max-len in the future)
- Transparently changes “masters” (leaders) when required.
- Probably more memory use and longer disk use tail
- Only uneven RabbitMQ clusters make sense (3, 5, 7 nodes)

Quorum Queue challenges

RabbitMQ for raft

- All commands are asynchronous
- Flow control and command priorities
- There can be no quorum or partitions on start / stop
- Cluster resize in RabbitMQ style can be hard

Raft for RabbitMQ

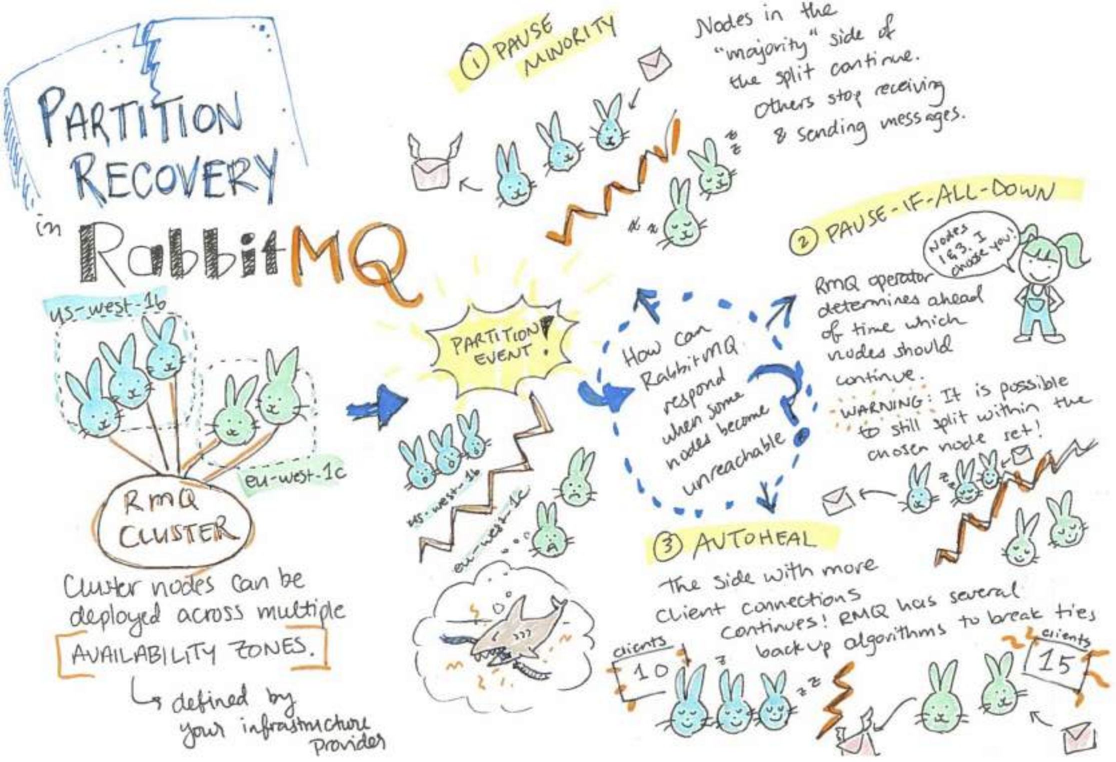
- Channels must keep track of queue states.
- A queue is not a single erlang process anymore and cannot be monitored
- Consumers are a part of the state machine state.
- Queue must have an ID (~ 262K queues limit)

RabbitMQ partitioning

- Rabbitmq relies on Mnesia
- Mnesia defines partition recovery
- Quorum queues will not help
- But Raft might

Illustration by

Denise Yu @deniseyu21

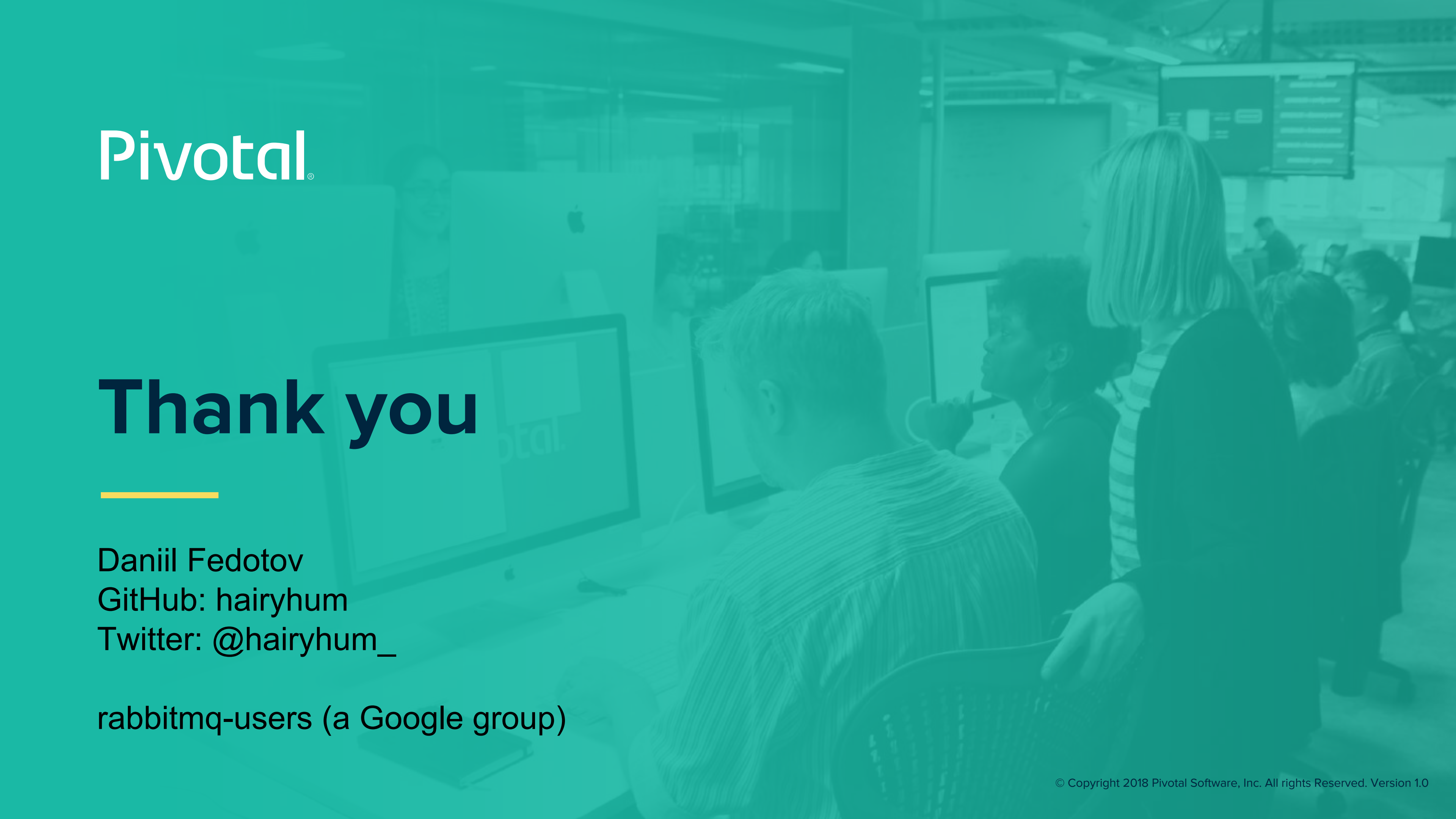

PARTITION RECOVERY
in RabbitMQ
us-west-1b
eu-west-1c
RmQ CLUSTER
Cluster nodes can be deployed across multiple AVAILABILITY ZONES.
↳ defined by your infrastructure provider
PARTITION EVENT!
us-west-1b
eu-west-1c
① PAUSE MINORITY
Nodes in the "majority" side of the split continue. Others stop receiving & sending messages.
How can RabbitMQ respond when some nodes become unreachable
② PAUSE-IF-ALL-DOWN
Nodes 1 & 3, I choose you!
RmQ operator determines ahead of time which nodes should continue.
WARNING: It is possible to still split within the chosen node set!
③ AUTOHEAL
The side with more client connections continues! RMQ has several backup algorithms to break ties
clients 10
clients 15

When will it ship

- Maybe never - but as soon as it is done
- Need to pass strict acceptance criteria (data safety, performance, RabbitMQ integration).
- If it ships it will be an optional feature
- The “classic” queue will still be the default

Summary

- RabbitMQ queue mirroring has fundamental problems
- Raft covers a very similar problem space
- New design promises lots of improvements 😊
- Implementing Raft is non-trivial
- github.com/rabbitmq/ra
- “quorum-queue” branch
- We are still learning
- Distributed systems are still hard



Pivotal

Thank you

Daniil Fedotov
GitHub: hairyhum
Twitter: @hairyhum_

rabbitmq-users (a Google group)