confluent

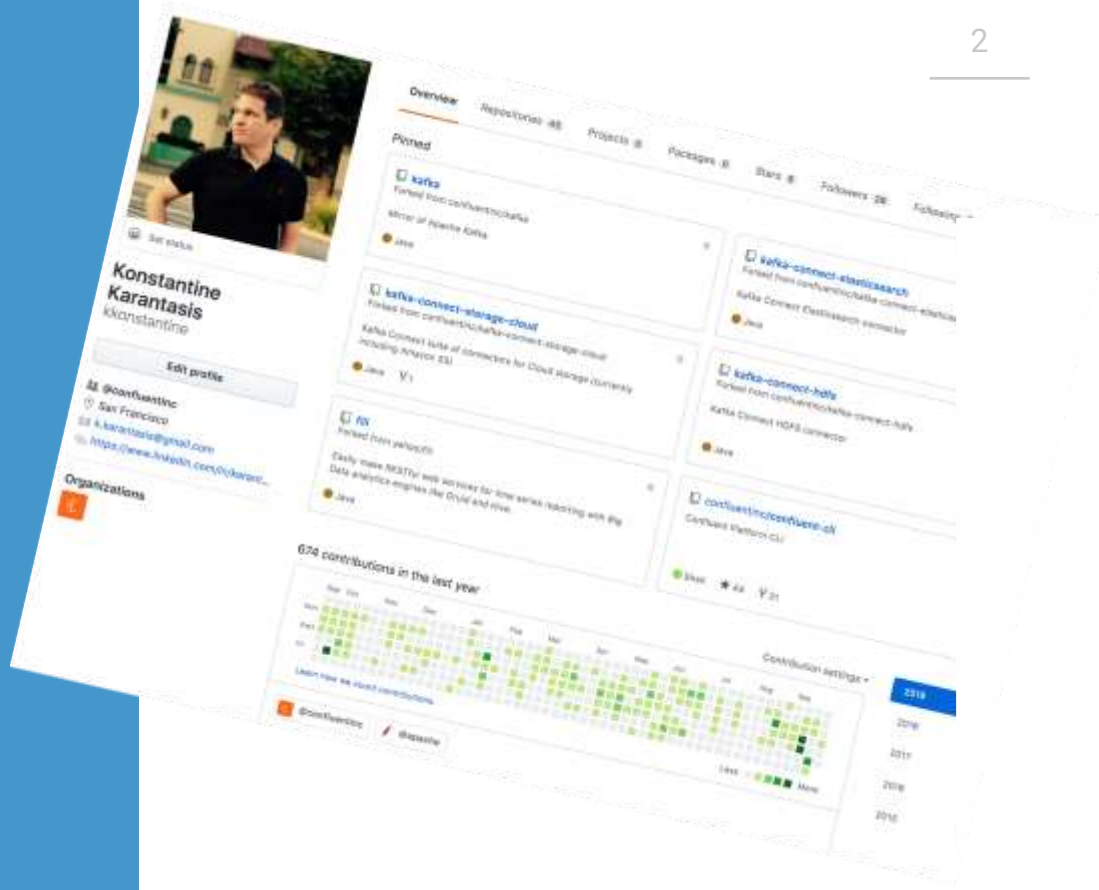
Design and Implementation of Incremental Cooperative Rebalancing

Konstantine Karantasis, PhD
Software Engineer, Confluent, Inc.

Nice to meet you

Contributor:

- Apache Kafka®
- Confluent Platform
- Confluent Community Components



kkonstantine
@karantasis

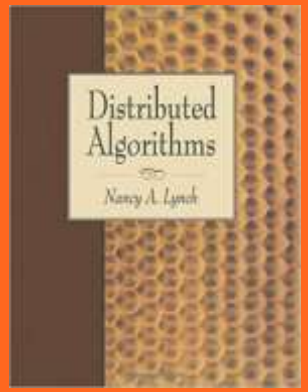
Agenda

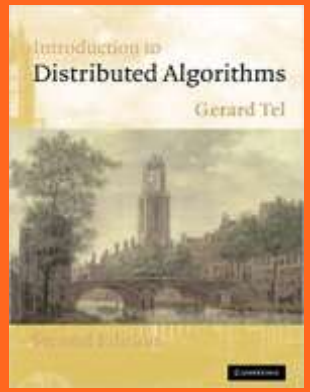
- Load balancing in Kafka Clients
- Challenges
- A new protocol
- Its first implementation
- Is it working?
- Patterns and predictions

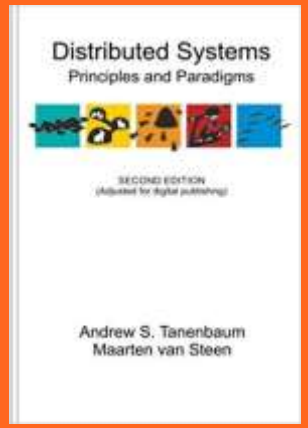
Distributed Systems are intriguing

An abundance of holy grails

- Consistency
- Consensus
- Load balancing







Load balancing in Kafka clients

a.k.a rebalancing

- Built on top of Group Membership
- Broker coordinator assigns the leader
- Load balancing is piggybacked to the Group Membership API calls

Load balancing as an embedded protocol

Two important pairs or Request/Response

- Join group request/response
- Sync group request/response

Kafka clients and resources to be balanced

- Consumer, Streams: **Topic-partitions**
- Connect: **Tasks**
- Schema Registry: **Leadership**

Stop-the-world rebalancing

- In every rebalance clients release their resources
- Resources → global pool
- Releasing resources is expensive
- Reacquiring resources is expensive

Challenges at Large Scale

“There is a coming and a going
A parting and often no–meeting again”

- Franz Kafka, 1897

Both in the Cloud and On-Prem

Load balancing challenges

1. Scaling up and down
2. Multitenancy
3. Kubernetes process death
4. Rolling bounce

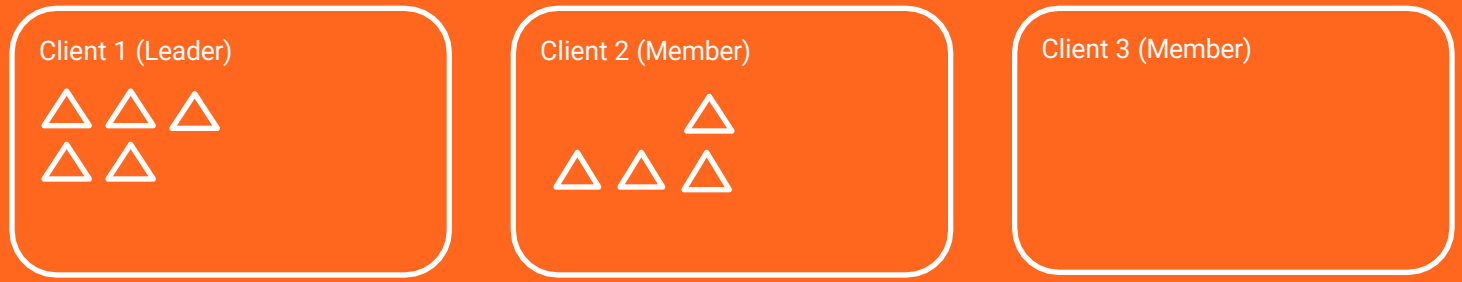
1. Scaling up and down

1. Scaling up and down

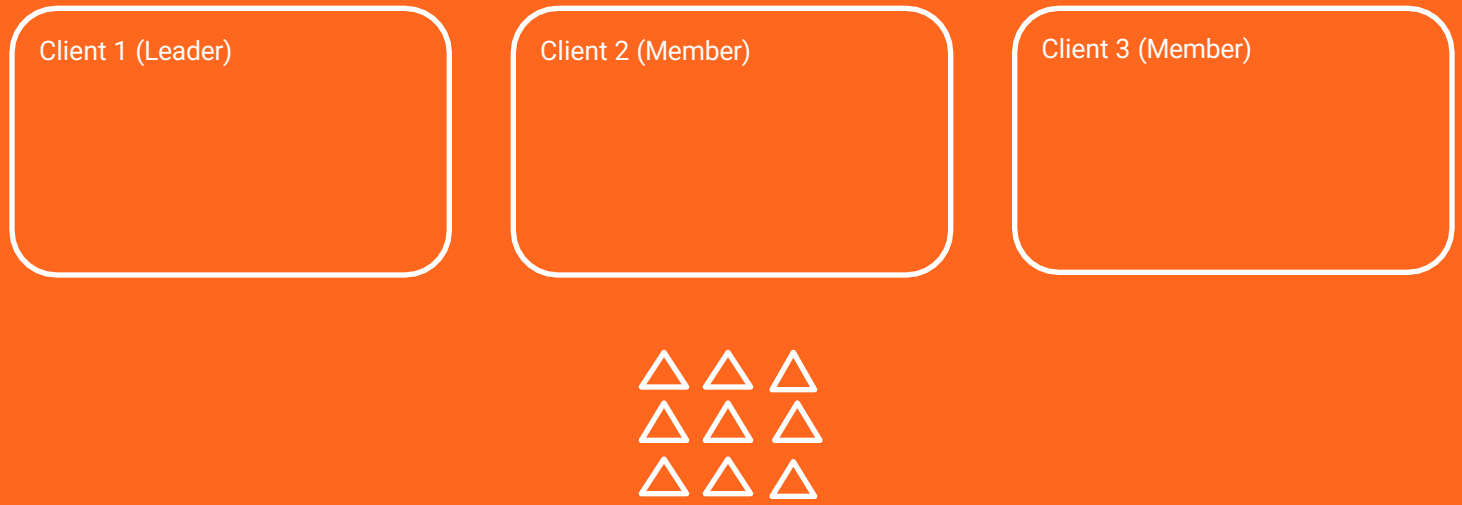
Client 1 (Leader)

Client 2 (Member)

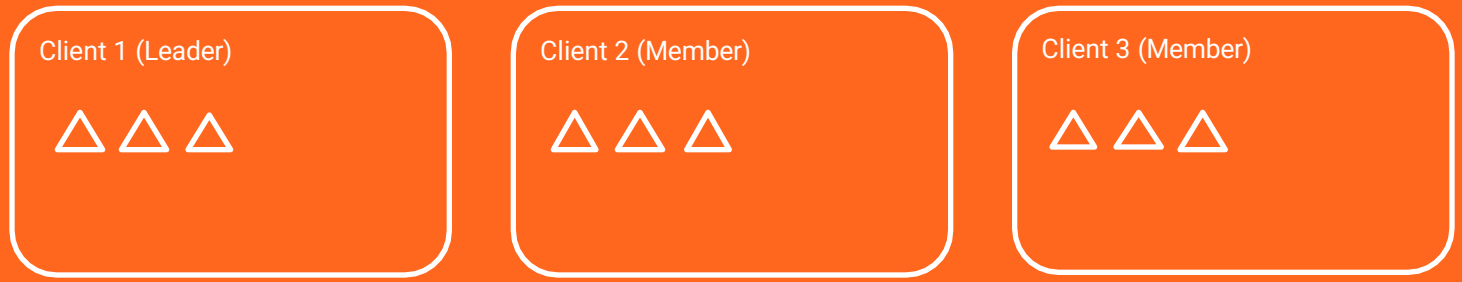
1. Scaling up and down



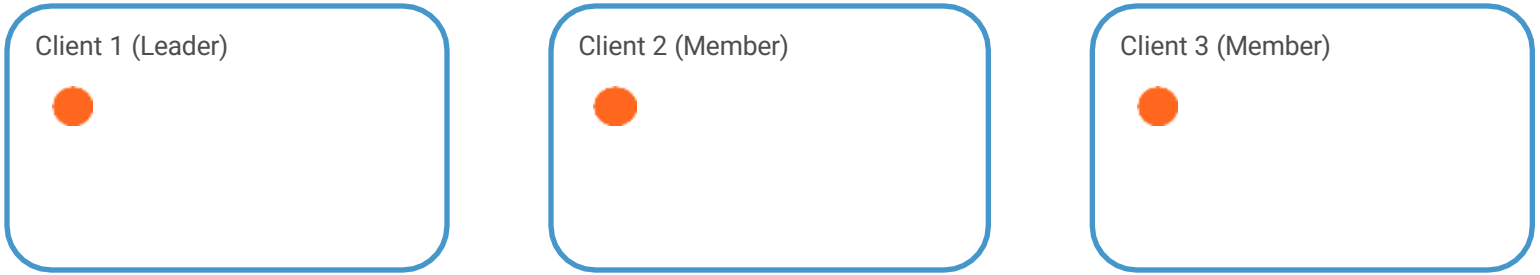
1. Scaling up and down



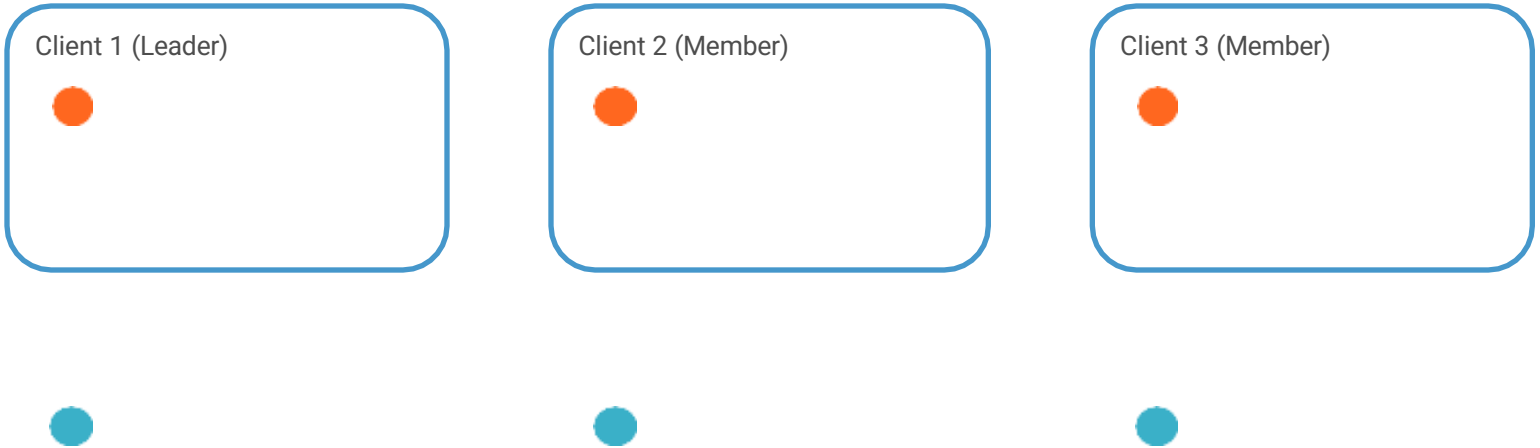
1. Scaling up and down



2. Multitenancy

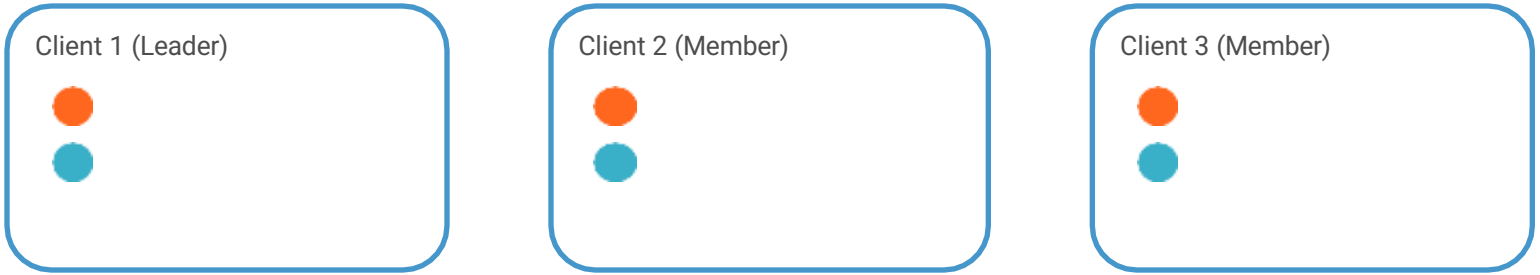


2. Multitenancy

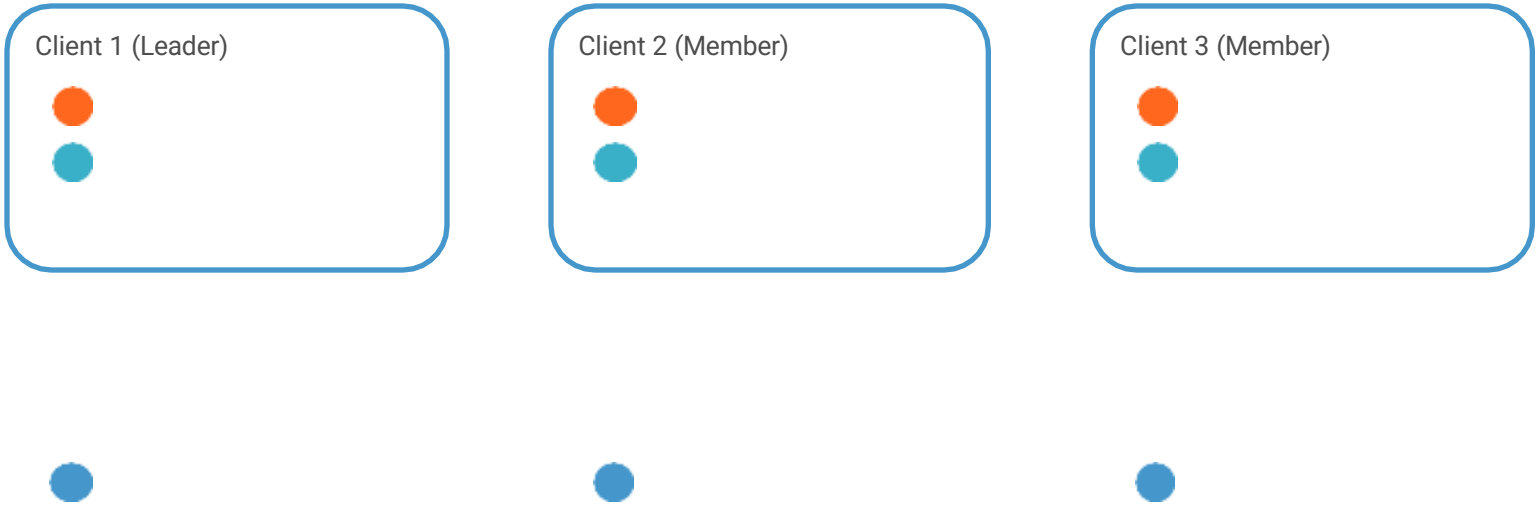


2. Multitenancy

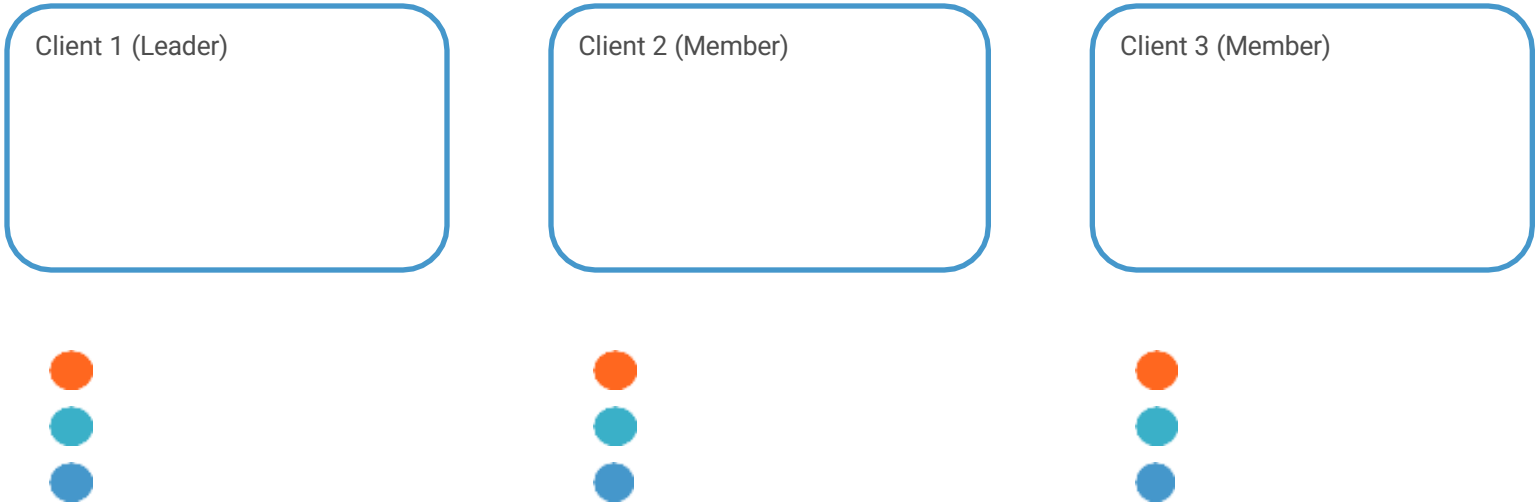
2. Multitenancy



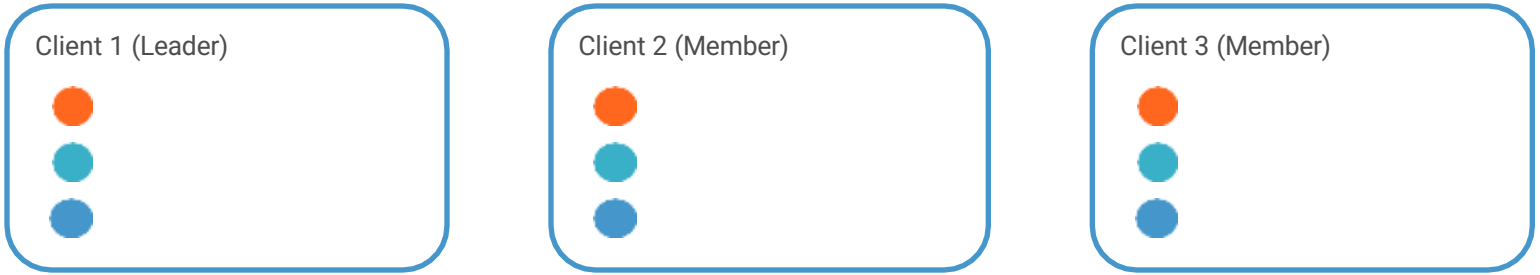
2. Multitenancy



2. Multitenancy



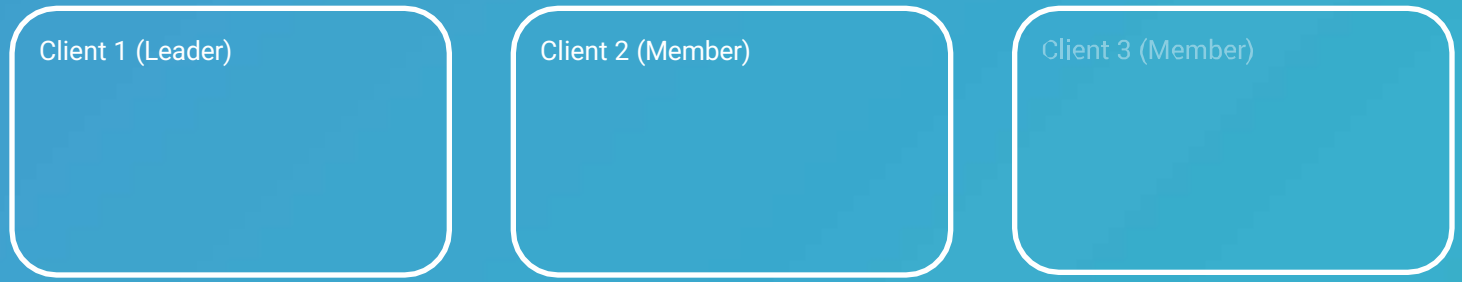
2. Multitenancy



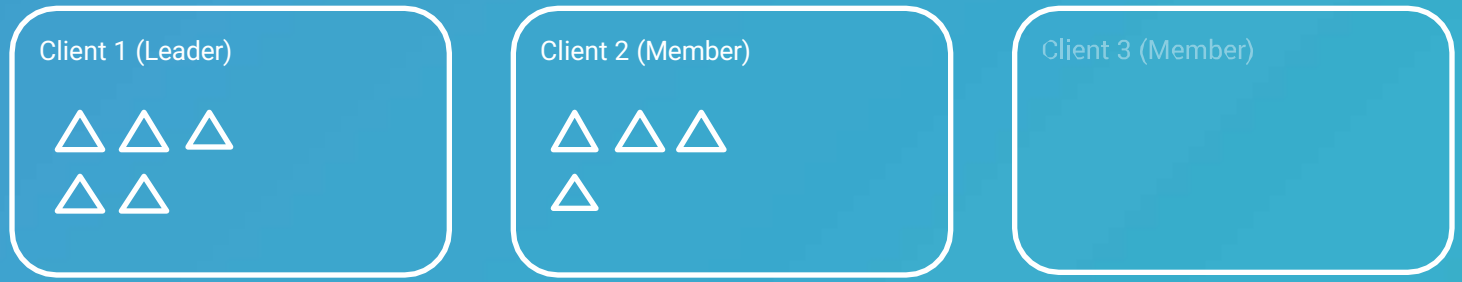
3. Kubernetes process death

3. Kubernetes process death

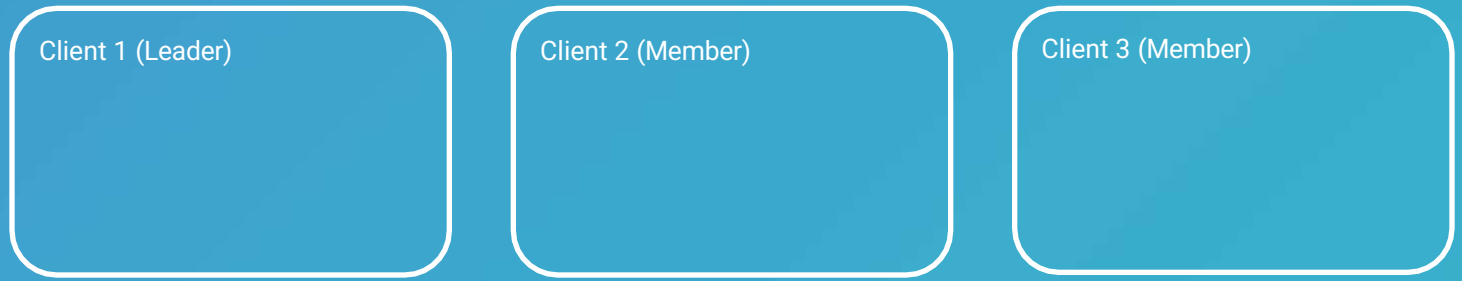
3. Kubernetes process death



3. Kubernetes process death

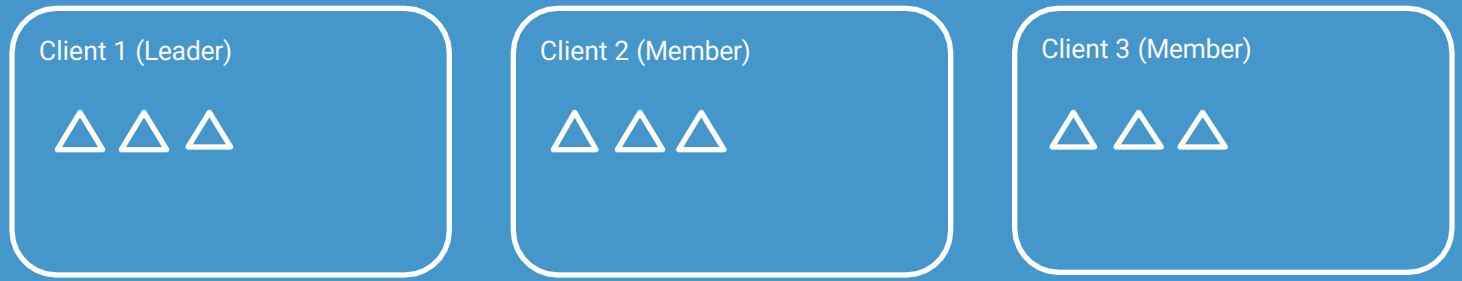


3. Kubernetes process death



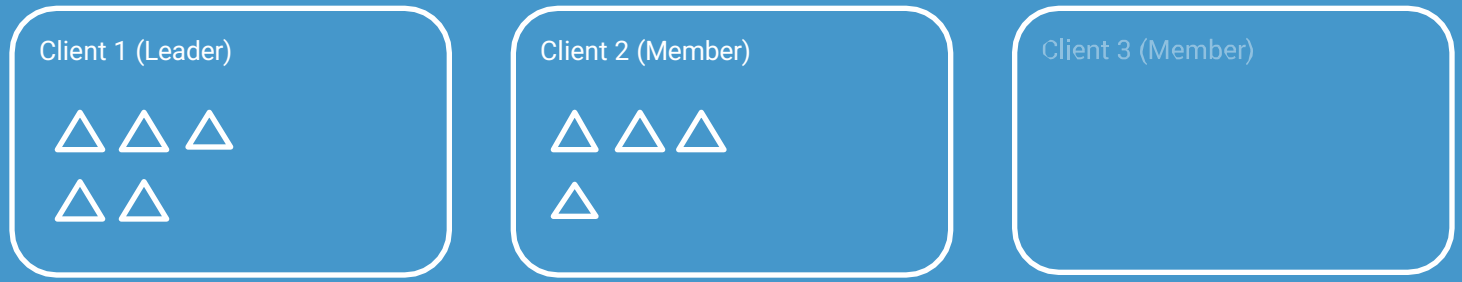
3. Kubernetes process death

4. Rolling restarts



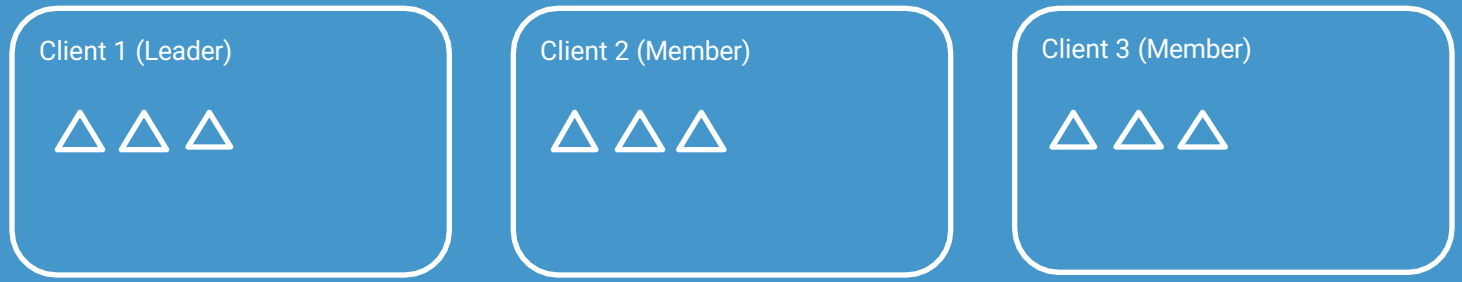
4. Rolling restarts

4. Rolling restarts

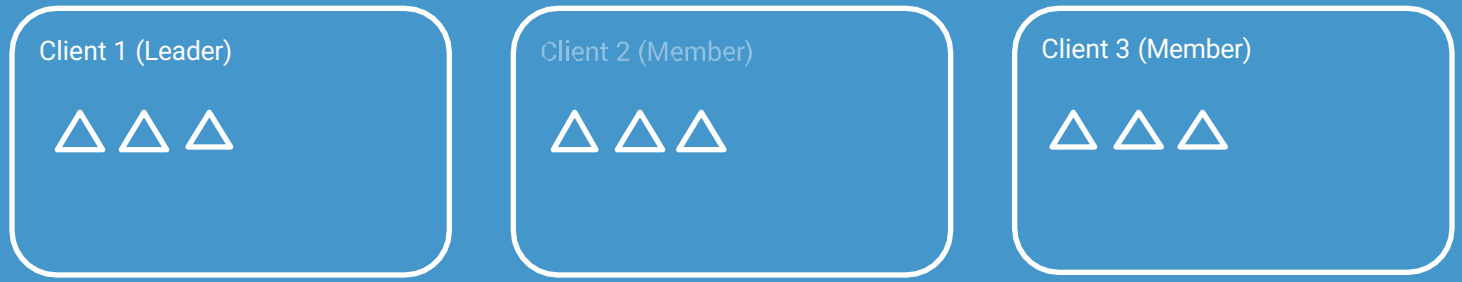


4. Rolling restarts

4. Rolling restarts

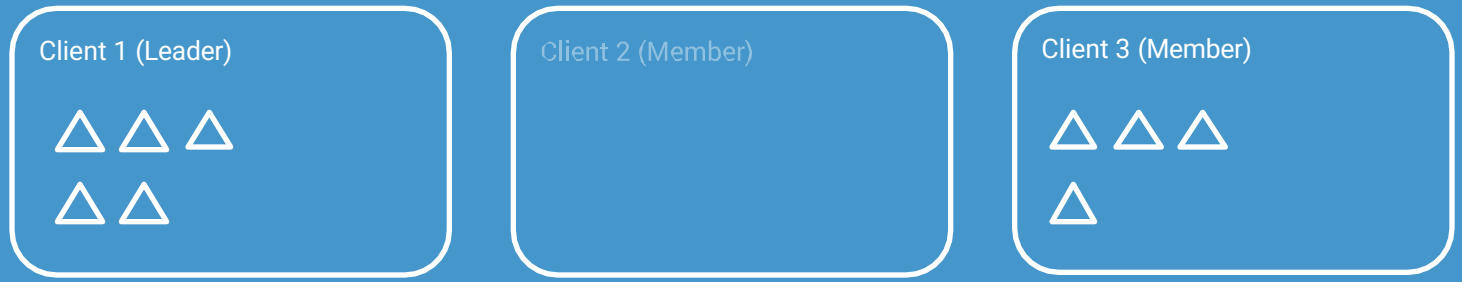


4. Rolling restarts



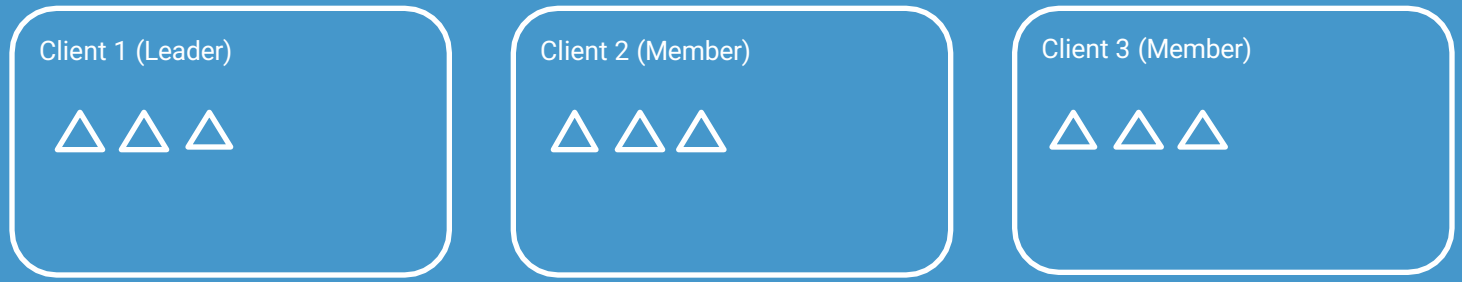
4. Rolling restarts

4. Rolling restarts



4. Rolling restarts

4. Rolling restarts



Kafka clients load balancing revisited

Do not stop-the-world

It’s expensive

Incremental Cooperative Rebalance

Goal: Address the challenges at large scale

Why **incremental**:

- No need to reach final state within a single rebalance round
- A grace period is configurable
- The protocol converges smoothly to a state of balanced load

Why **cooperative**:

- Resource revocation and release is graceful

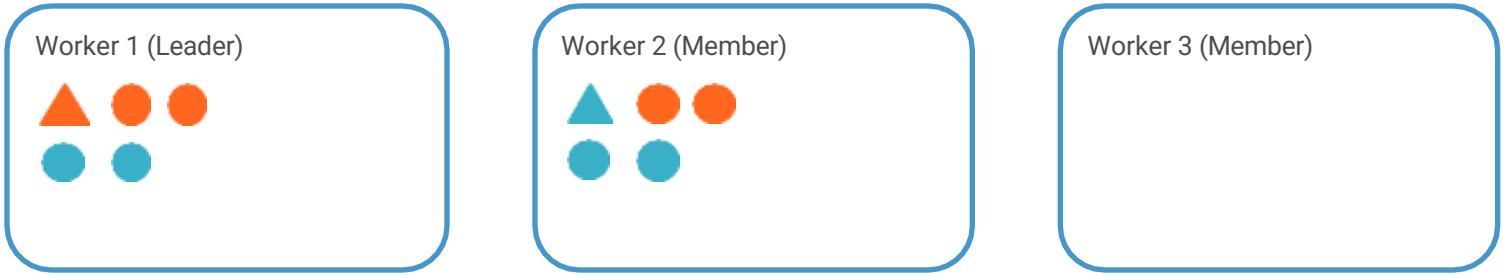
KIP-415

- Adopted in Apache Kafka 2.3
- Does not require broker coordinator upgrade
- Easy upgrades and extensions
- The Connect protocol was extended to include:
  a) assigned tasks
  b) revoked tasks
  c) a scheduled delay to perform another rebalance

https://cwiki.apache.org/confluence/display/KAFKA/KIP-415%3A+Incremental+Cooperative+Rebalancing+in+Kafka+Connect

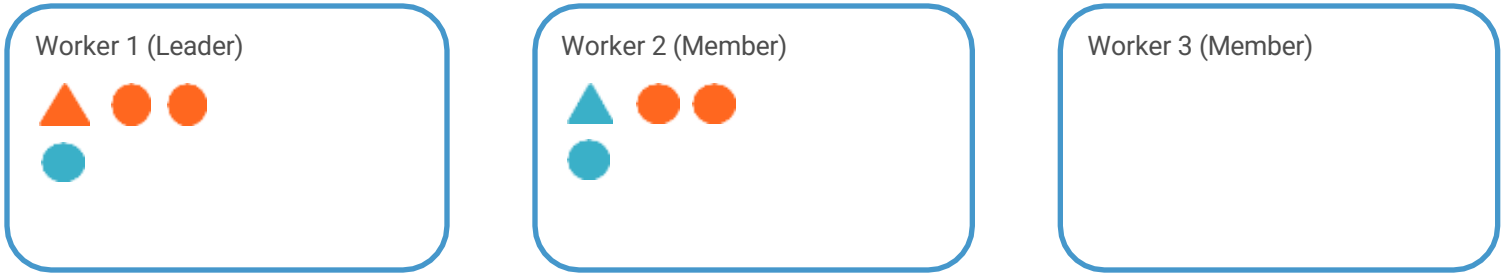
A new worker joins the group

A new worker joins



A new worker joins

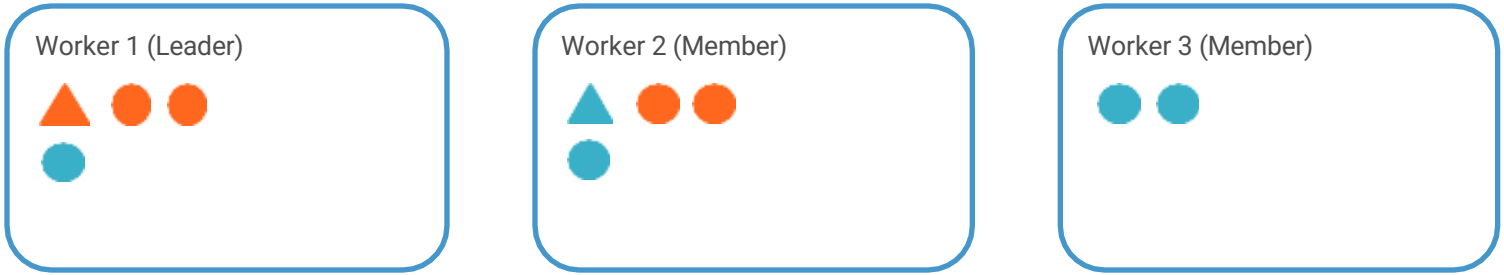
1st Rebalance



Revoked:

A new worker joins

2nd Rebalance



But what about failures or restarts?

Rebalance with a delay

- Start and stop can be expensive
- Avoid unnecessary task shuffling
- Don’t interfere with running tasks

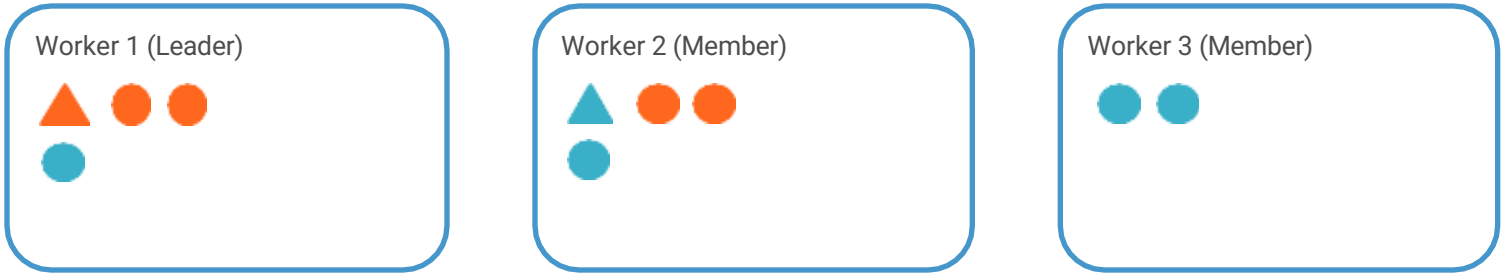
Postpone task shuffling with:

`scheduled.rebalance.max.delay.ms`

Affects only “lost” tasks

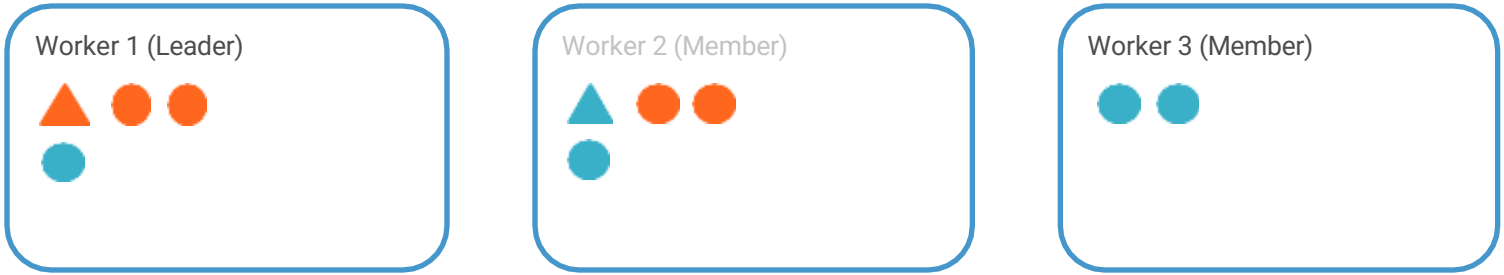
An existing worker leaves and bounces back

An existing worker bounces



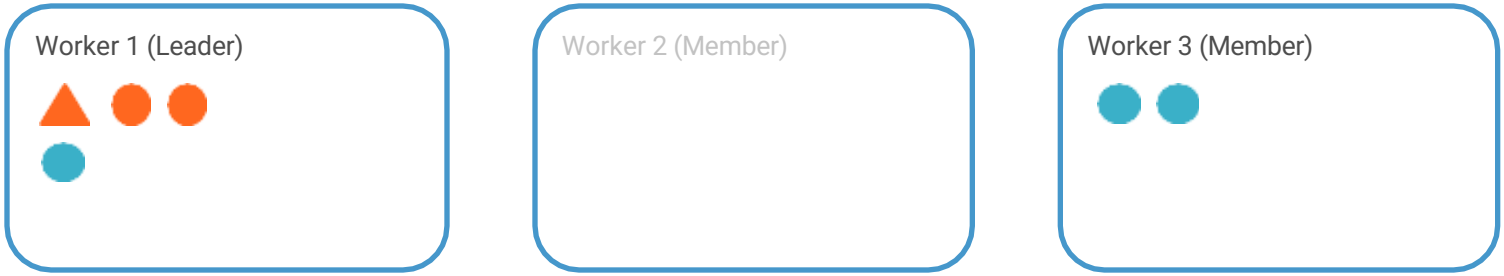
An existing worker bounces

1st Rebalance



An existing worker bounces

1st Rebalance

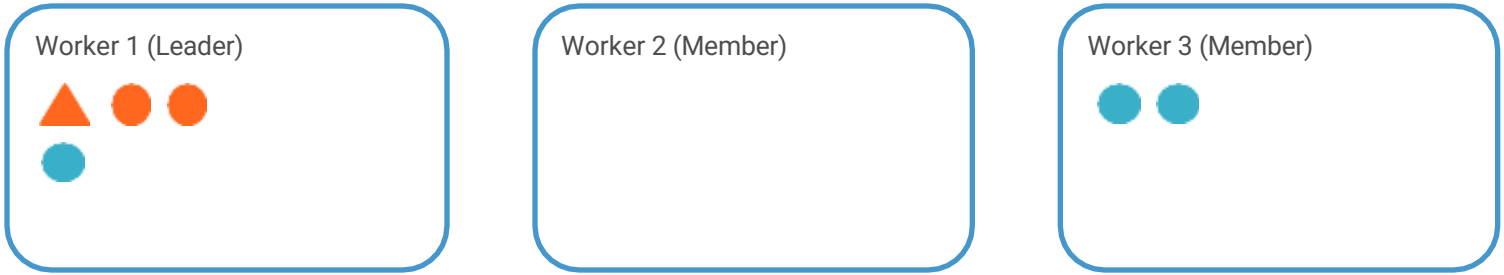


Unassigned:

scheduled.rebalance.max.delay.ms goes in effect. Starts with a default value of **5 min**

An existing worker bounces

2nd Rebalance



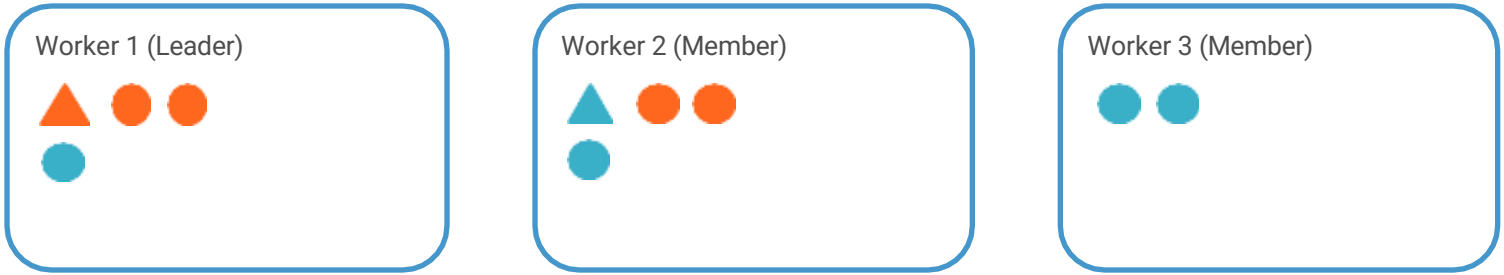
Unassigned:

Worker 2 returns within the initial delay of **5 min**

scheduled.rebalance.max.delay.ms is still in effect. Value is between **0** and **5 min**

An existing worker bounces

3rd Rebalance

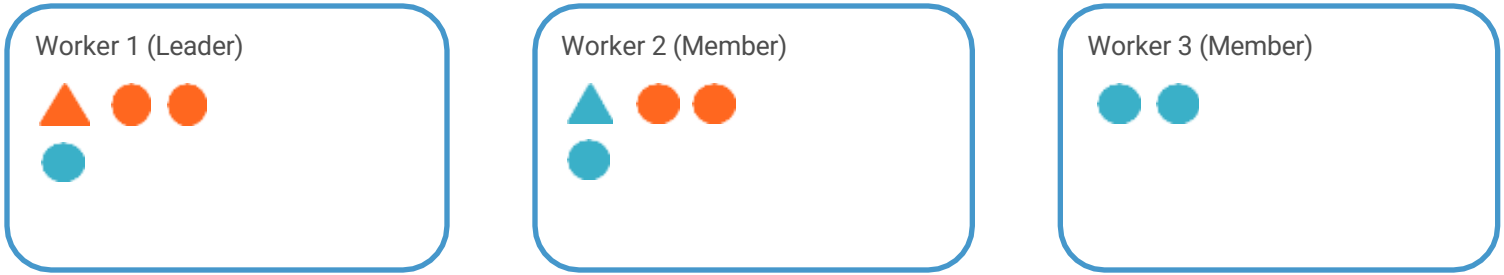


scheduled.rebalance.max.delay.ms has expired

Upon expiration all the workers rejoin the group triggering a 3rd rebalance

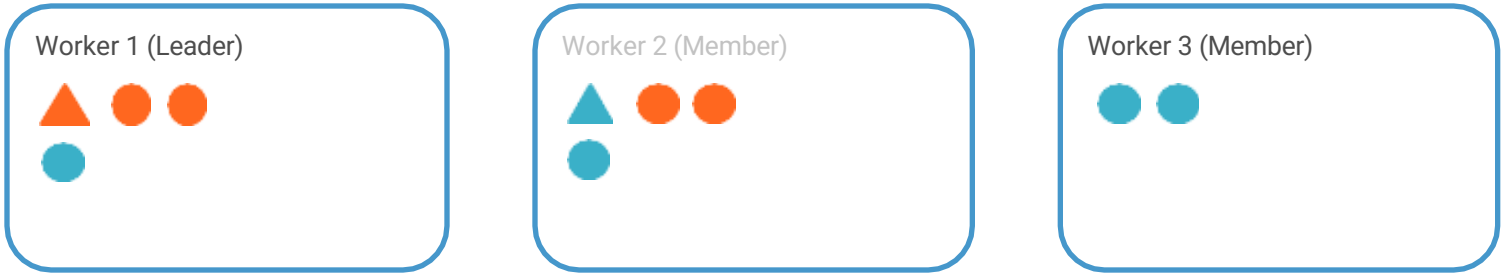
An existing worker leaves permanently

An existing worker leaves permanently



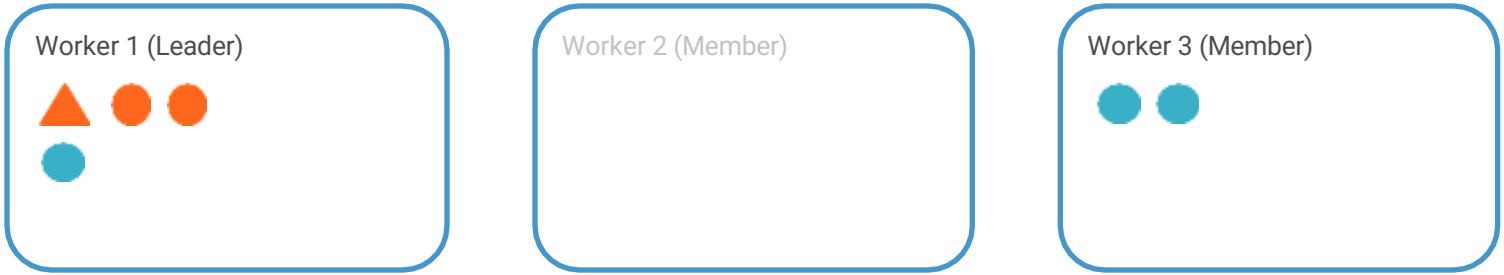
An existing worker leaves permanently

1st Rebalance



An existing worker leaves permanently

1st Rebalance

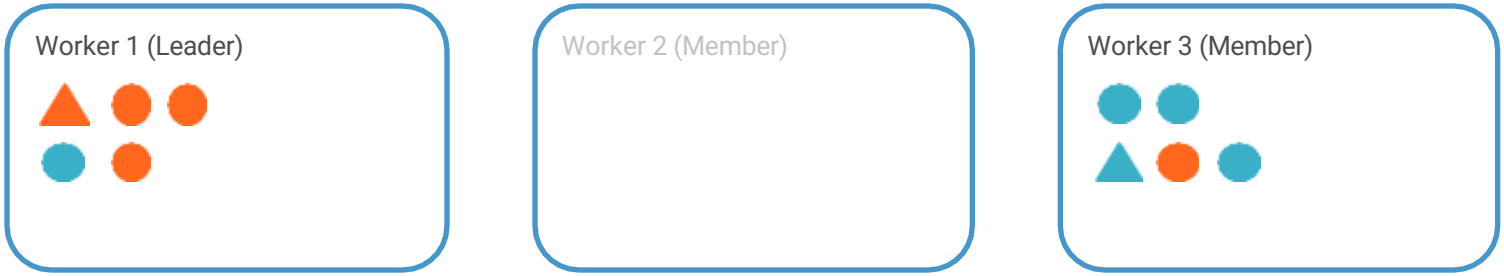


Unassigned:

scheduled.rebalance.max.delay.ms goes in effect. Starts with a default value of **5 min**

An existing worker leaves permanently

2nd Rebalance



scheduled.rebalance.max.delay.ms has expired

Upon expiration, workers rejoin the group triggering a 3rd rebalance

Unassigned tasks are distributed to the existing workers

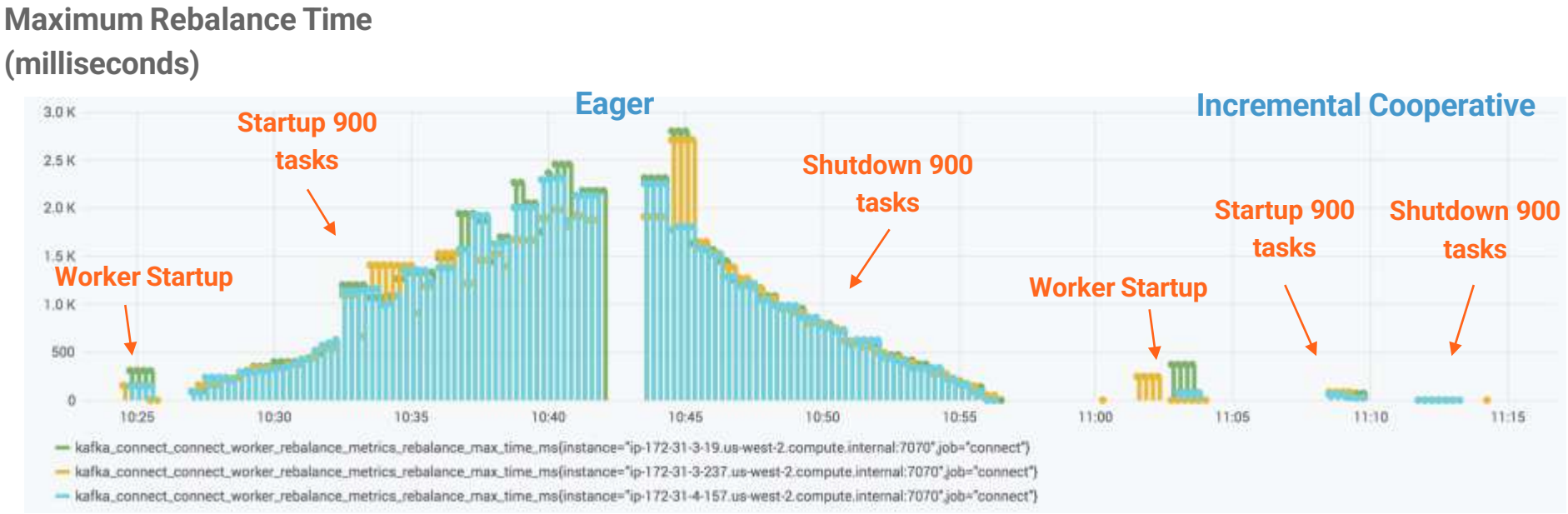
Implementation in Kafka Connect

- Behind the scenes, a state machine with:
  - Three base sets acting as sources of truth for the leader
  - Several derived sets extracted from the base sets
  - Logic to handle the delays
  - Weighted round-robin for task assignment

New Rebalancing in Action

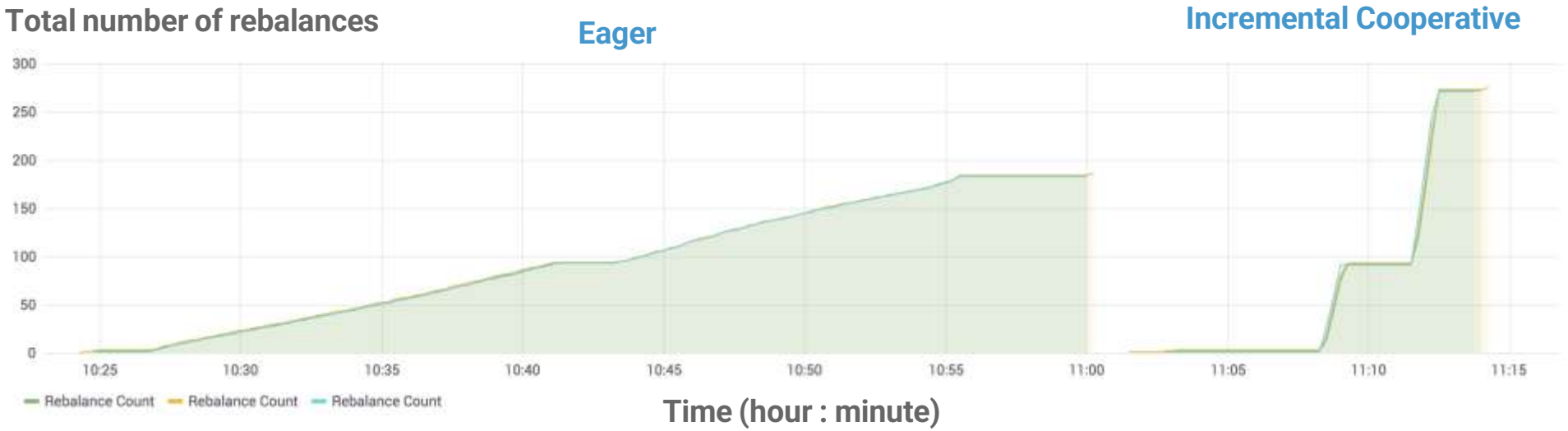
90 connectors
900 tasks
3 workers

Rebalance independently of the current load



With Incremental Cooperative Rebalancing the cost does not scale with the number of tasks

Rebalances become lightweight



More rebalances but less expensive

The impact of rebalancing on throughput

90 S3 Connectors/900 Tasks	Eager Rebalancing	Incremental Cooperative Rebalancing	
Aggregate throughput (MB/s)	252.68	537.81	**2x improvement**
Maximum throughput (MB/s)	0.41	3.82	**9x improvement**

For details, read:

https://www.confluent.io/blog/incremental-cooperative-rebalancing-in-kafka

Patterns and Predictions

Freedom from workarounds

- Fragmented deployments
- Increased timeouts to avoid rebalance storms

Smaller clusters, more of them?

Takeaways

- Running Kafka clients at scale is becoming a reality
- Bigger clusters of clients are now more manageable and can be diverse
- Get it for free! Just upgrade your Connect cluster to version 2.3 and beyond
- Fall back to Eager Rebalancing by setting `connect.protocol` config

Predictions for 2020 and beyond

- Increased number of large scale Connector deployments of thousands of tasks in production

- Rebalancing improvements are coming to the Kafka consumer and Kafka Streams
(KIP-429 and KIP-441)

Stay in touch!

cnfl.io/download

cnfl.io/slack

cnfl.io/blog


confluent