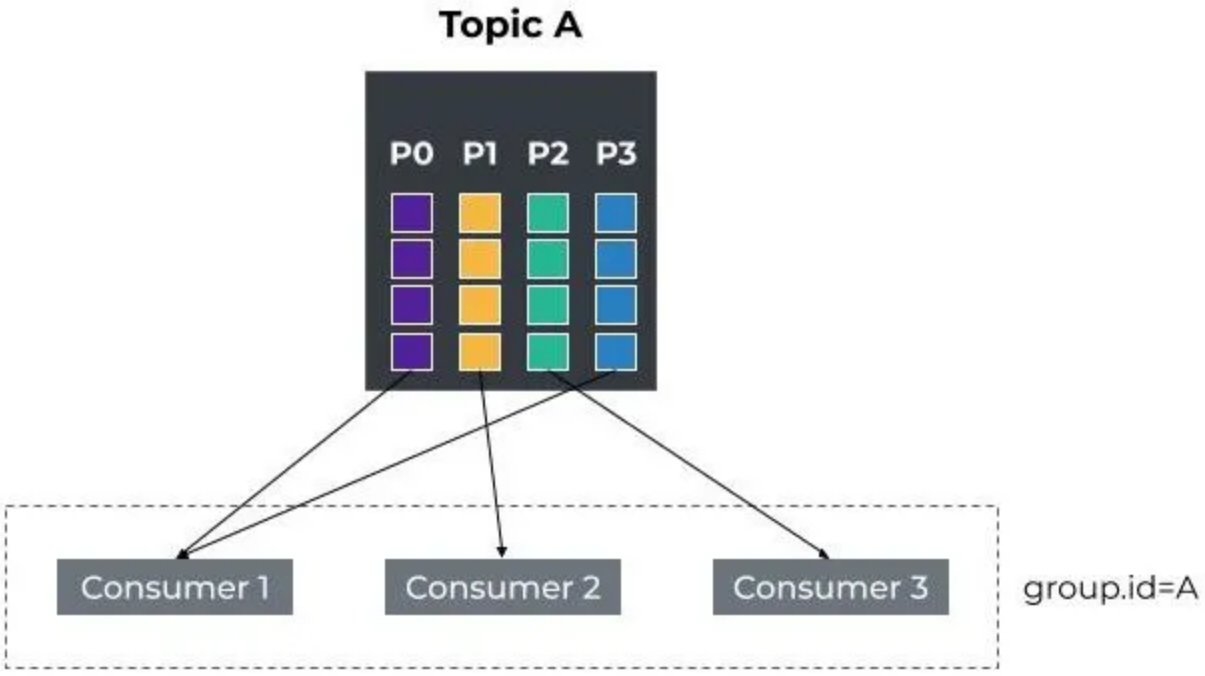
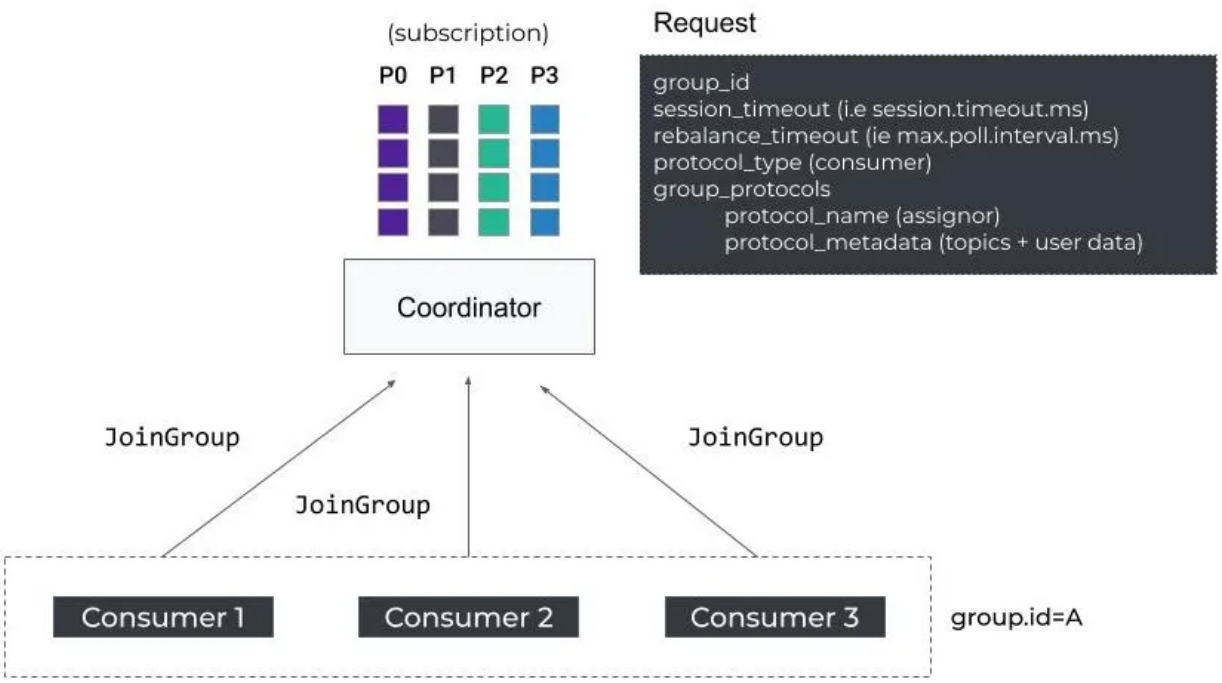
Rebalance Protocol



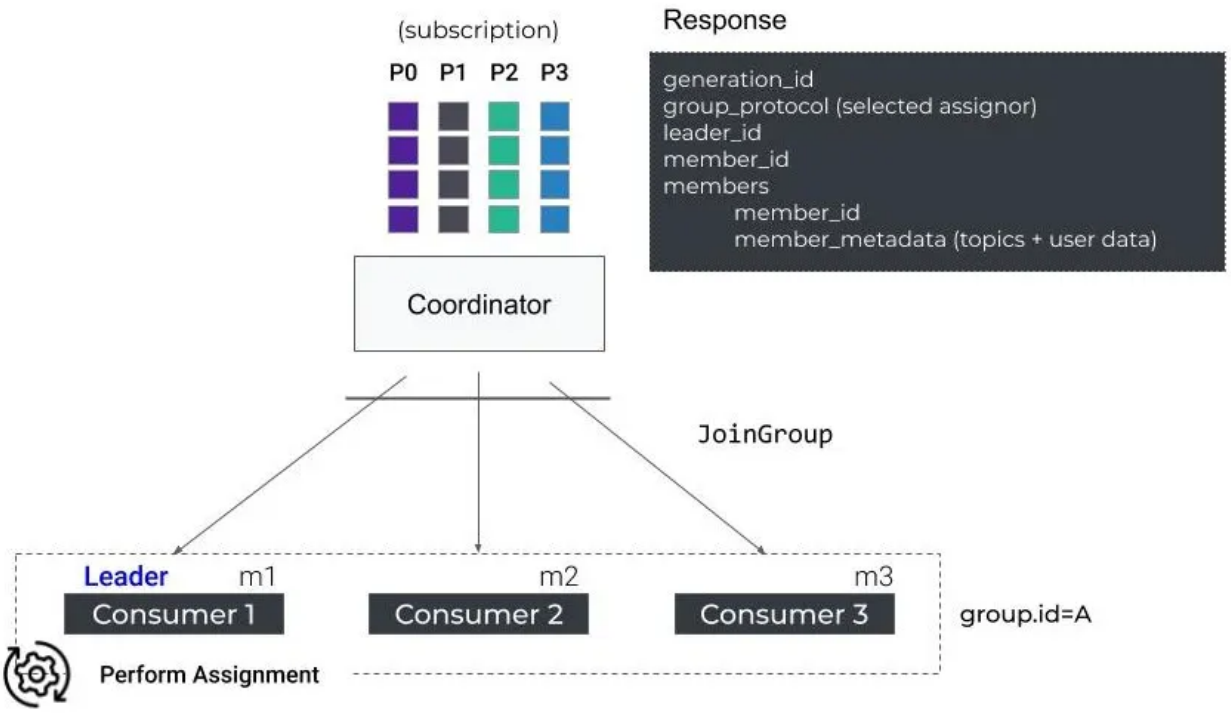
Rebalance/Rebalancing: the procedure that is followed by a number of distributed processes that use Kafka clients and/or the Kafka coordinator to form a common group and distribute a set of resources among the members of the group

JoinGroup



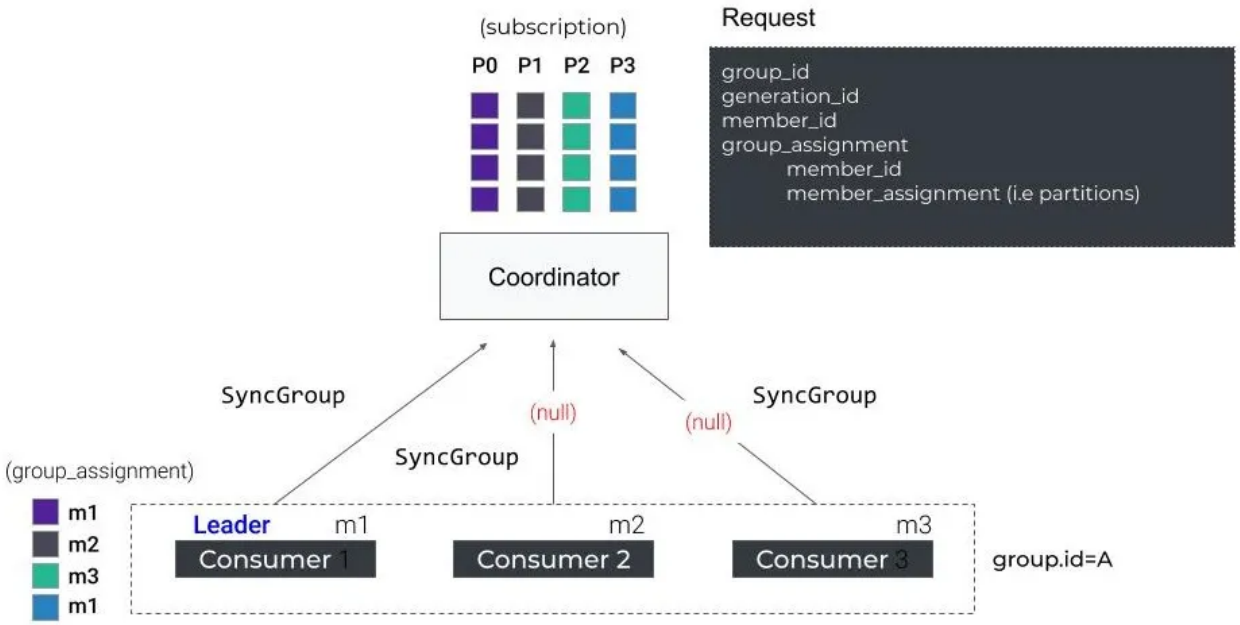
Consumer — Rebalance Protocol — SyncGroup Request

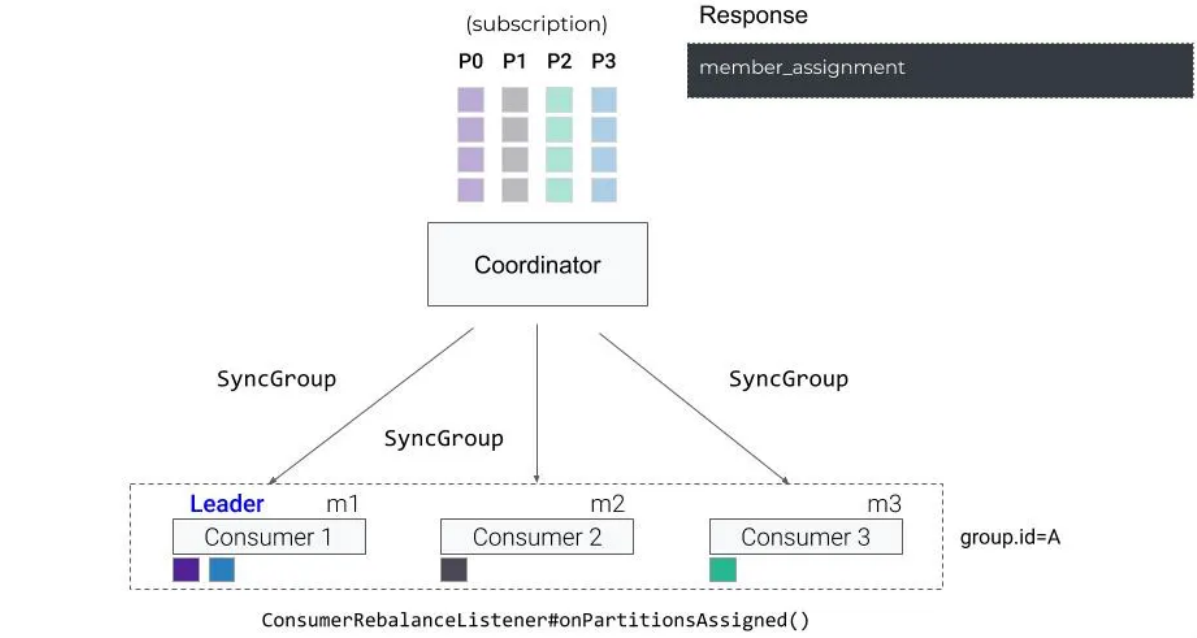
The `JoinGroup` acts as a barrier, meaning that the coordinator doesn't send responses as long as all consumer requests are not received
(i.e `group.initial.rebalance.delay.ms)` or rebalance timeout is reached.



> The group leader is responsible for executing the partitions assignments locally.

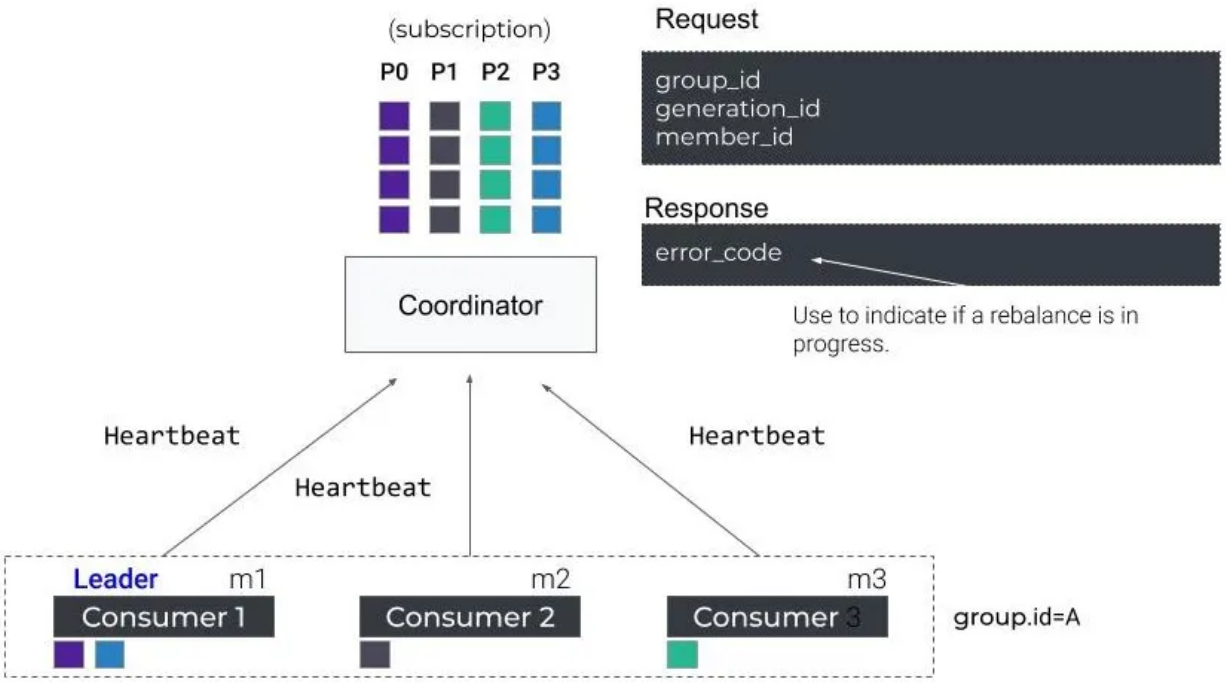
SyncGroup


(subscription)
P0 P1 P2 P3
Request
group_id
generation_id
member_id
group_assignment
member_id
member_assignment (i.e partitions)
Coordinator
SyncGroup
(null)
SyncGroup
(null)
SyncGroup
(group_assignment)
m1
m2
m3
m1
Leader
m1
Consumer 1
m2
Consumer 2
m3
Consumer 3
group.id=A


(subscription)
P0 P1 P2 P3
Response
member_assignment
Coordinator
SyncGroup
SyncGroup
SyncGroup
Leader
m1
Consumer 1
m2
Consumer 2
m3
Consumer 3
group.id=A
ConsumerRebalanceListener#onPartitionsAssigned()

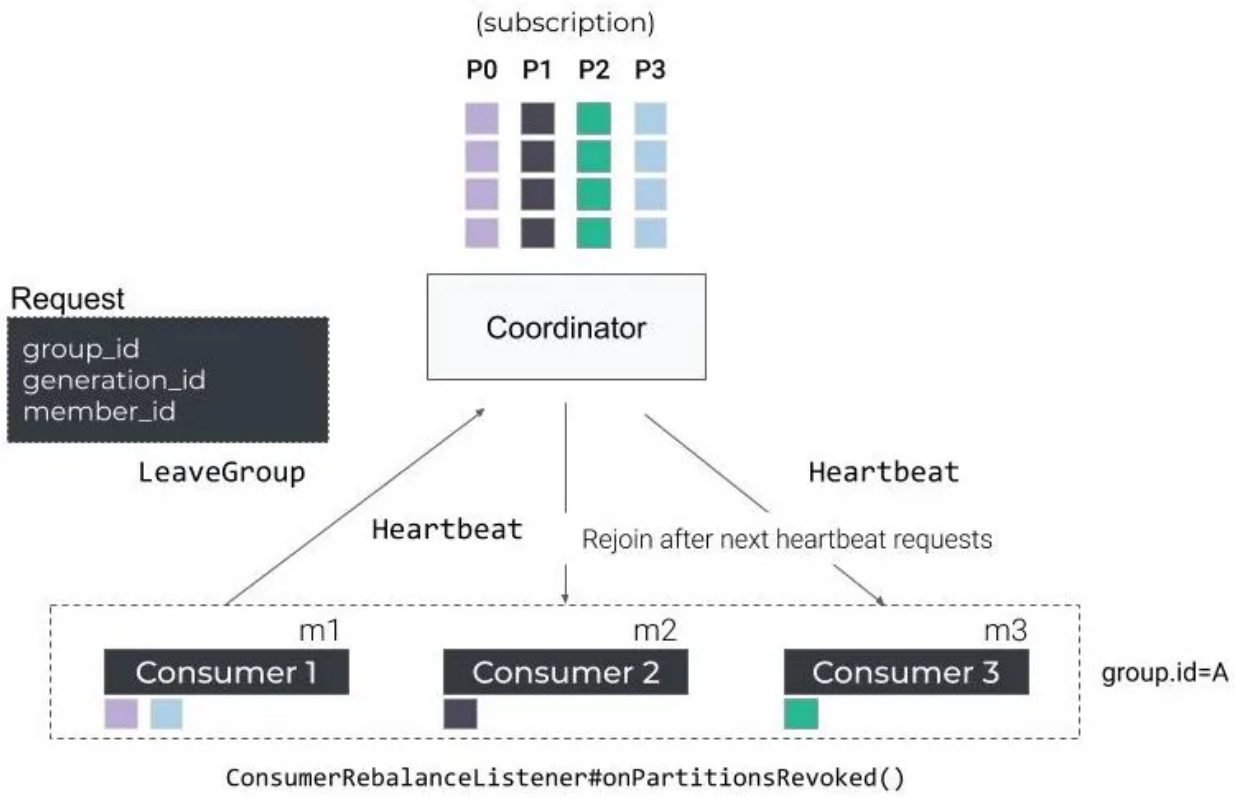
18

Heartbeat

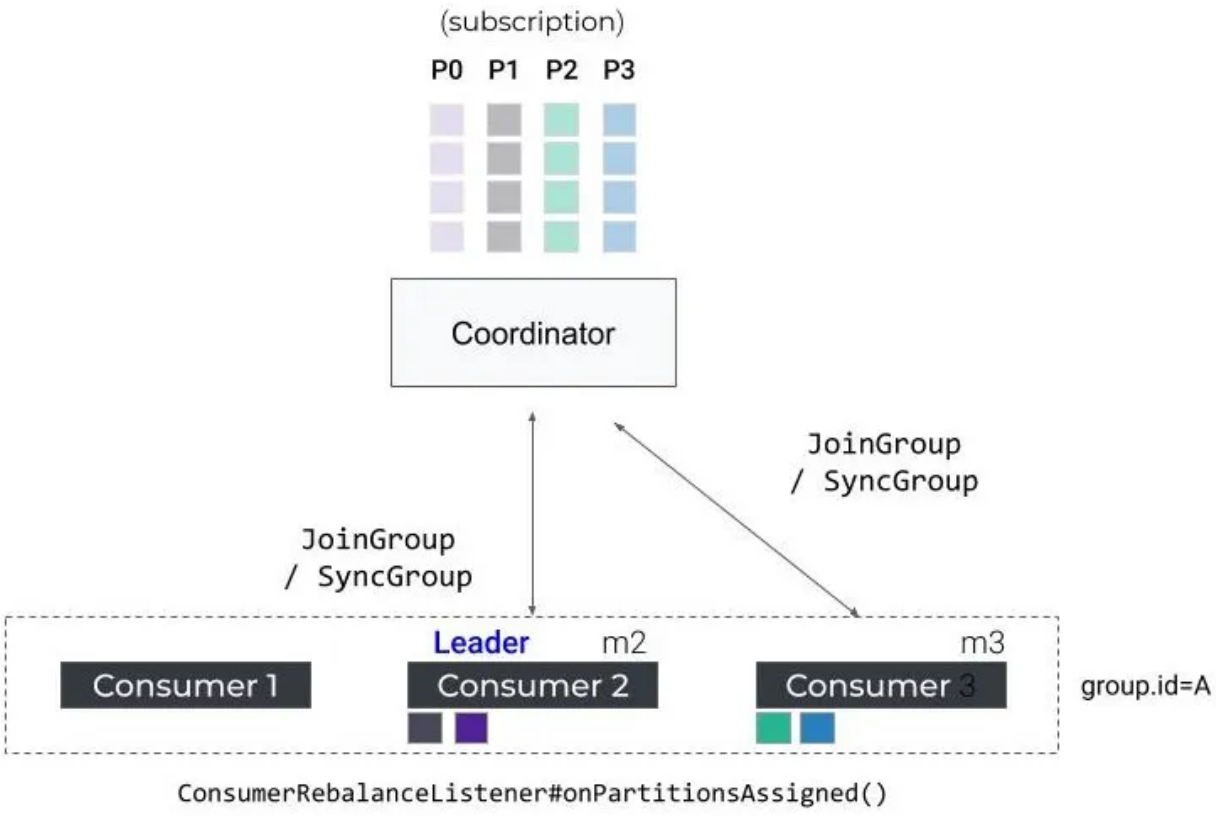


Some caveats

- The first limitation of the rebalance protocol is that we cannot simply rebalance one member without stopping the whole group `( stop-the-world effect ).`



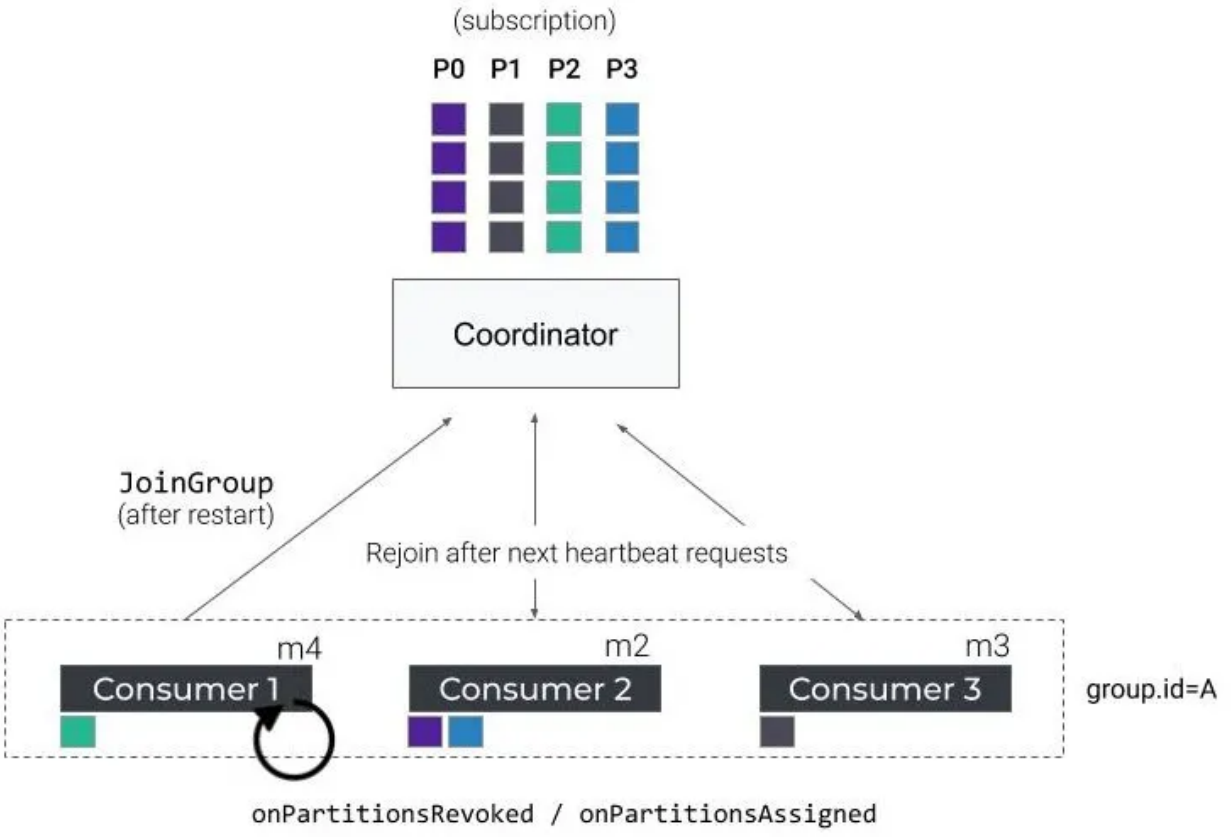
Remaining consumers will be notified that a rebalance must be performed on the next `Heartbeat` and will initiate a new `JoinGroup/SyncGroup` round-trip in order to reassign partitions.



> During the entire rebalancing process, i.e. as long as the partitions are not reassigned, consumers no longer process any data

But what would happen, if, for example, the consumer was just restarting after a transient failure ?

Well, the consumer, while rejoining the group, will trigger a new rebalance causing all consumers to be stopped (once again).

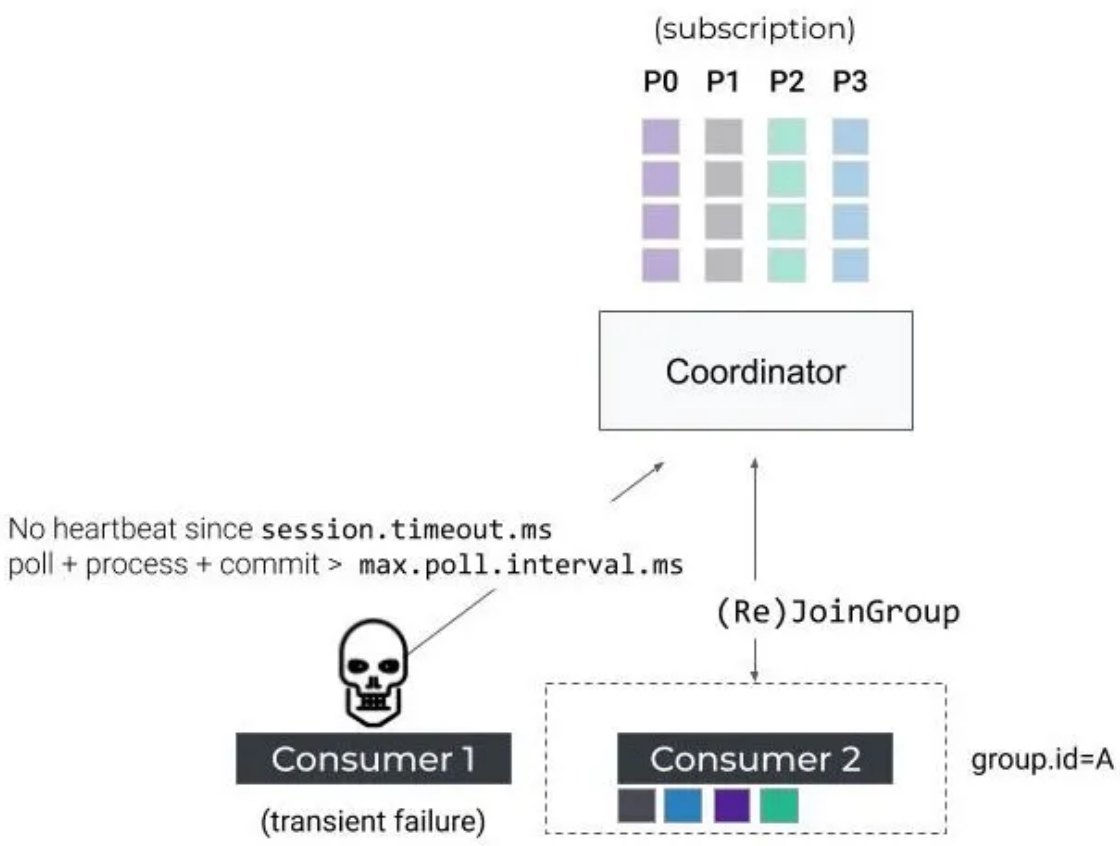


Another reason that can lead to a restart of a consumer is a `rolling upgrade of the group.`

This scenario is unfortunately disastrous for the consumption group. Indeed, with a group of three consumers, such operation will trigger 6 rebalances that could potentially have a significant impact on messages processing.

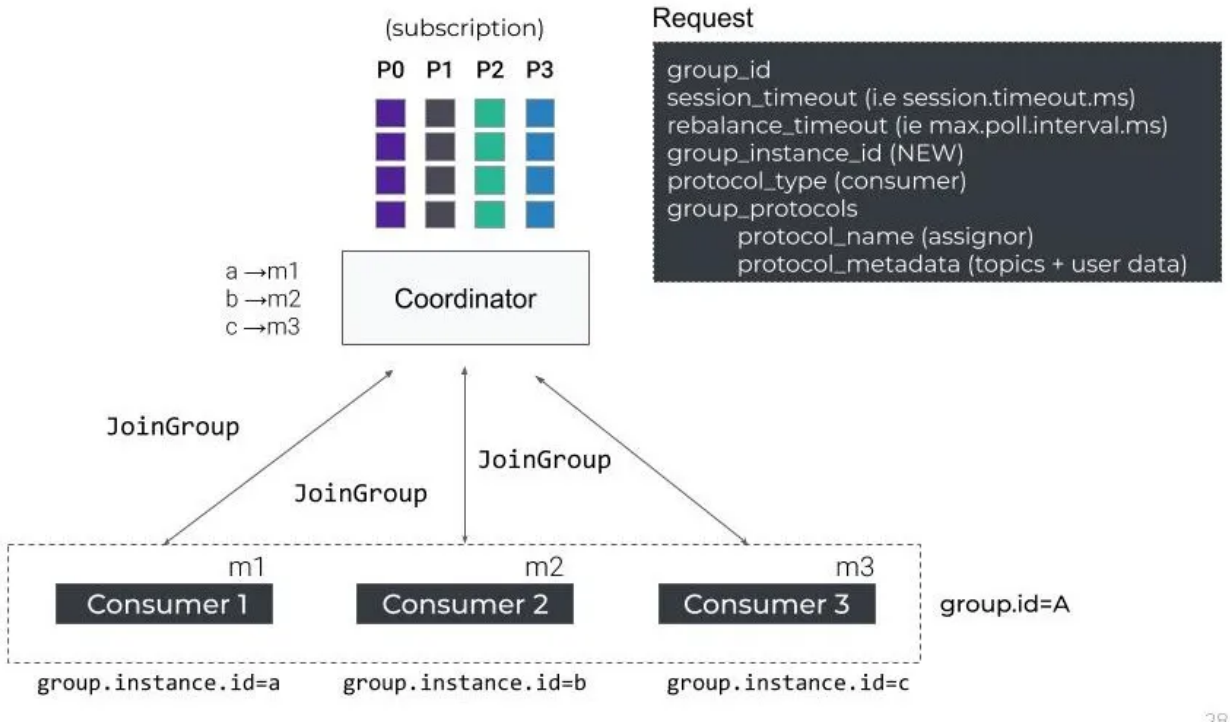
Finally, a common problem when running Kafka consumers, in Java, is either missing a heartbeat request, due to a network outage or a long GC pause, or not invoking the method `KafkaConsumer#poll()`, periodically, due to an excessive processing time.

- In the first case, the coordinator is not receiving a heartbeat for more than `session.timeout.ms` milliseconds and considers the consumer dead.
- In the second one, the time needed for processing polled records is superior to `max.poll.inteval.ms`.



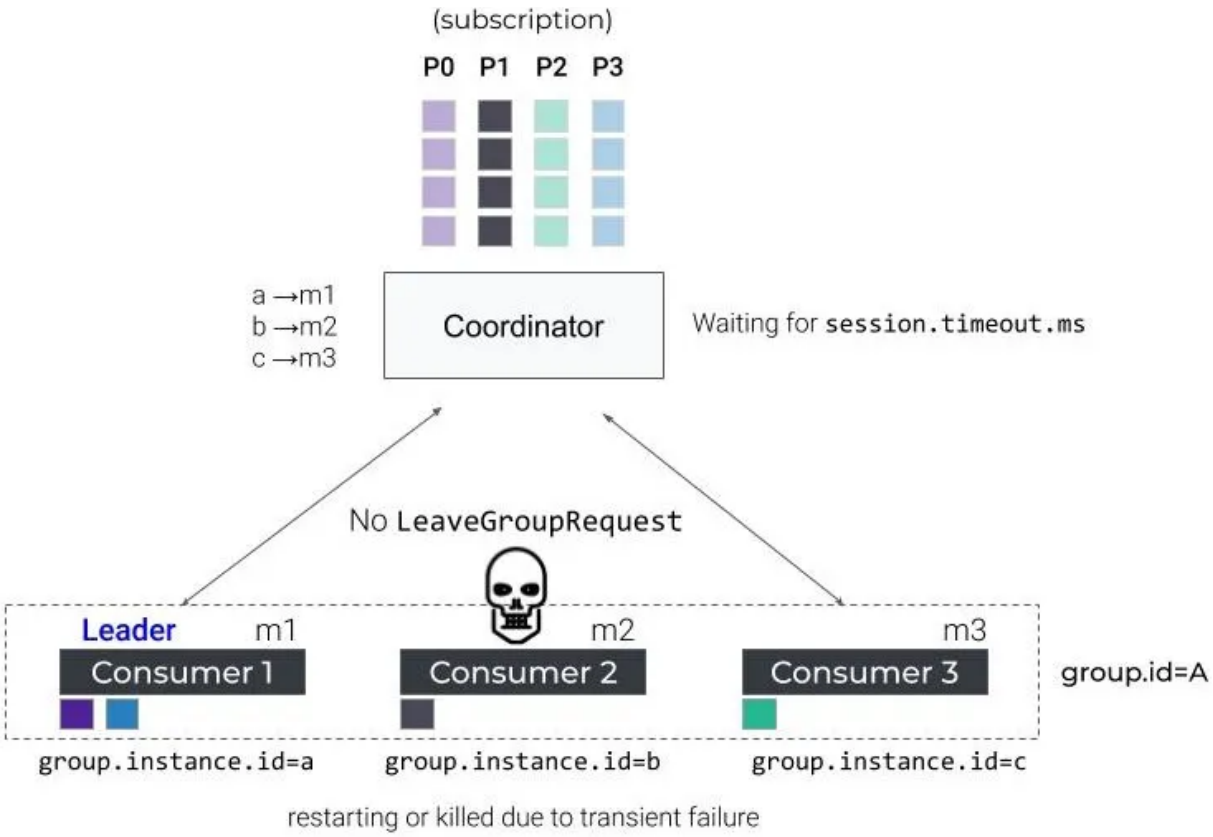
Static Membership

To reduce consumer rebalances due to transient failures, Apache Kafka 2.3 introduces the concept of Static Membership

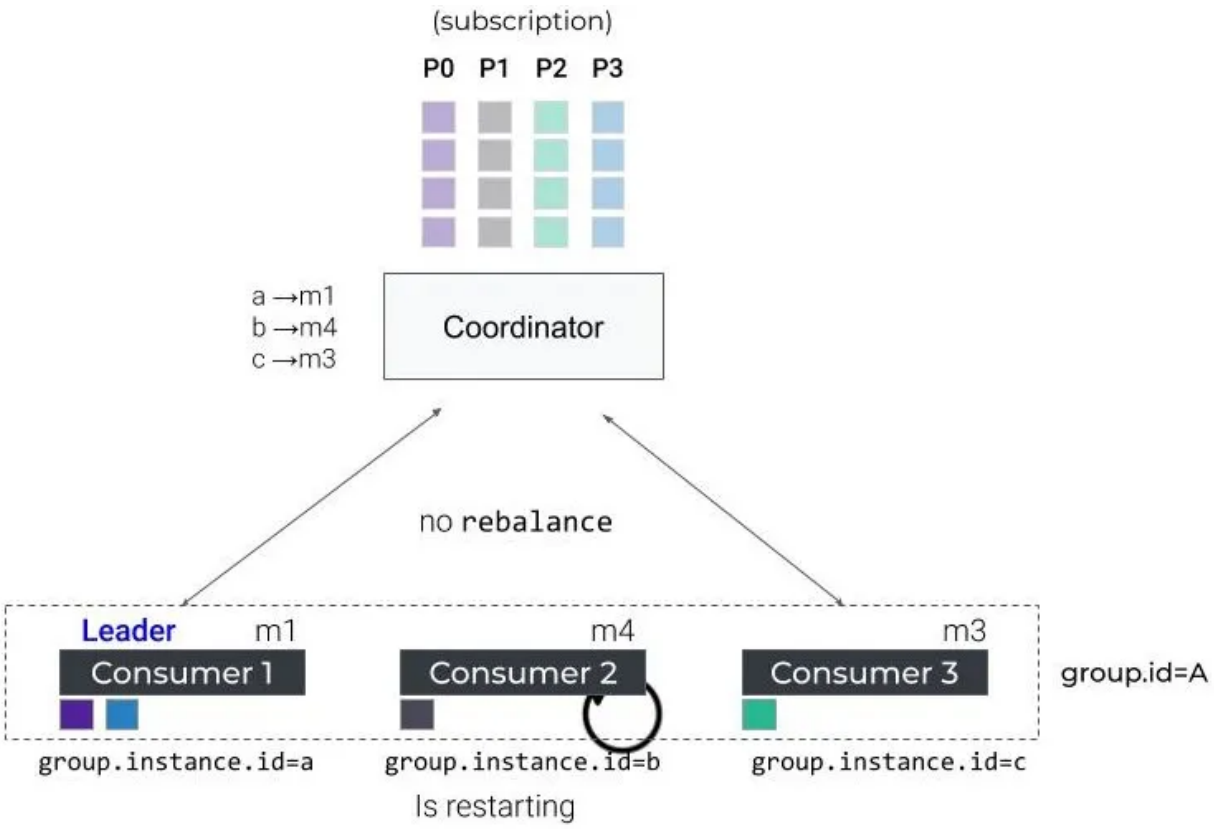


If a consumer is restarted or killed due to a transient failure, the broker coordinator will not inform other consumers that a rebalance is necessary until `session.timeout.ms` is reached.

One reason for that is that consumers will not send `LeaveGroup` request when they are stopped.



When the consumer will finally rejoin the group, the broker coordinator will return the cached assignment back to it, without doing any rebalance.



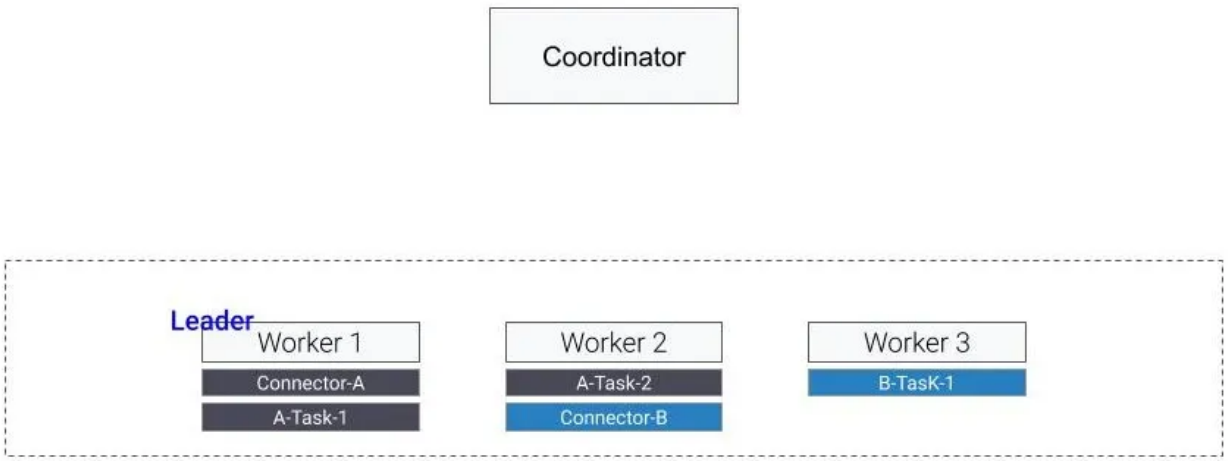
When using static membership, it’s recommended to increase the consumer property `session.timeout.ms` large enough so that the broker coordinator will not trigger rebalance too frequently.

Incremental Cooperative Rebalancing

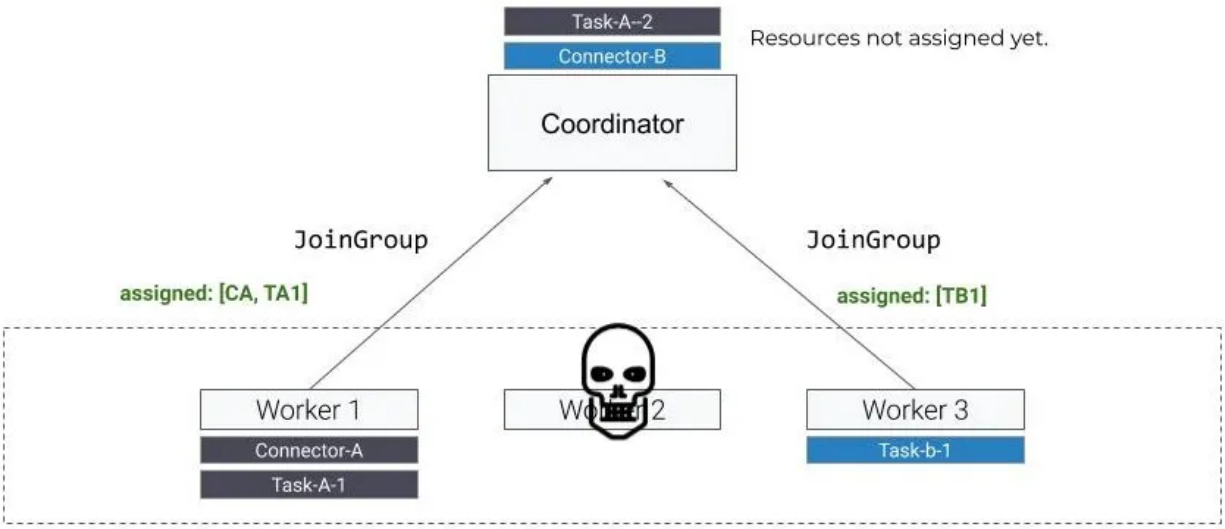
The Incremental Cooperative Rebalancing attempts to solve this problem in two ways :

1) only stop tasks/members for revoked resources.

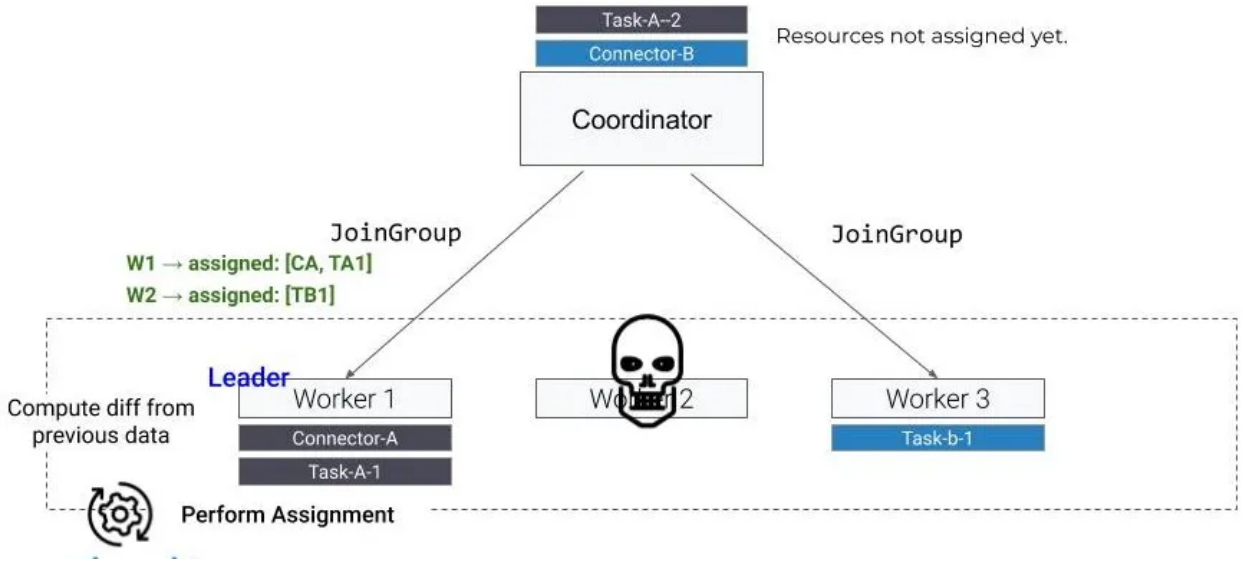
2) handle temporary imbalances in resource distribution among members, either immediately or deferred (useful for rolling restart).



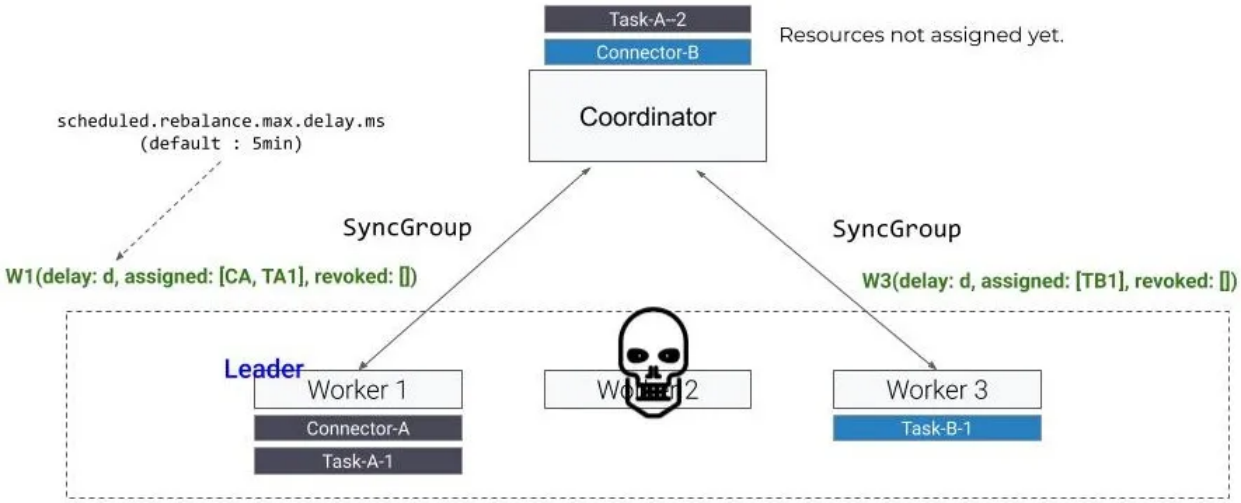
1 — Initial assignment



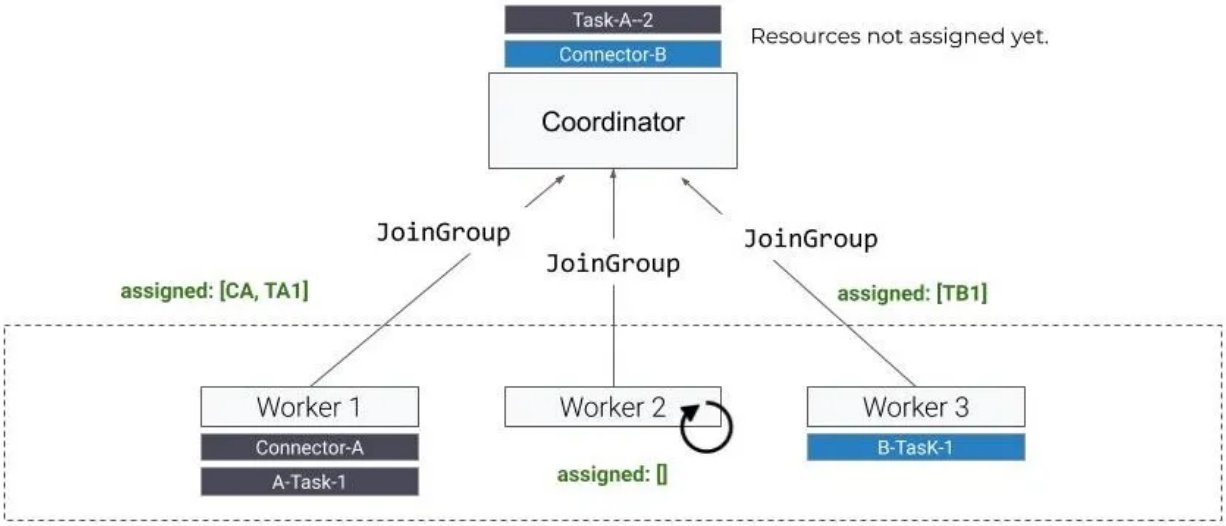
2 — W2 leaves the group and rebalance is triggered (W1, W3 join).



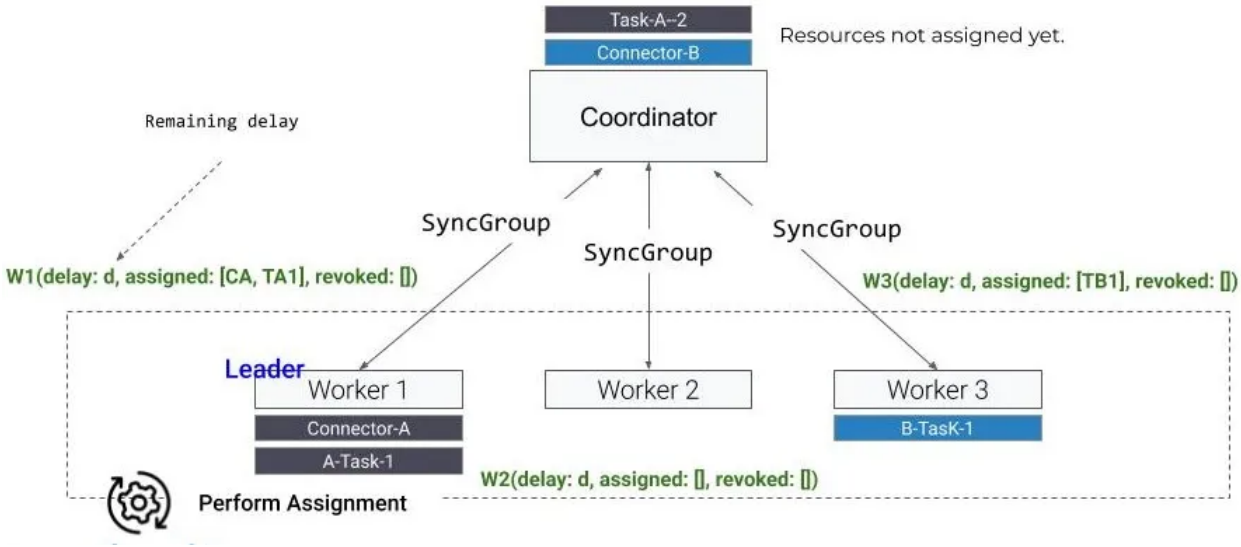
3 — W1 becomes leader and computes assignments



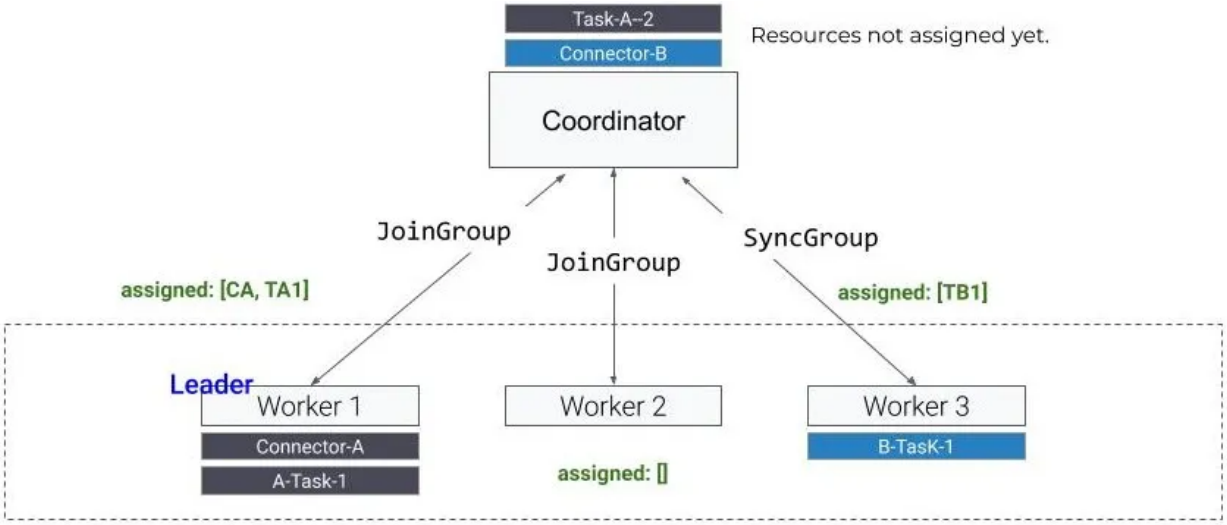
4 — W1, W3 receive assignments



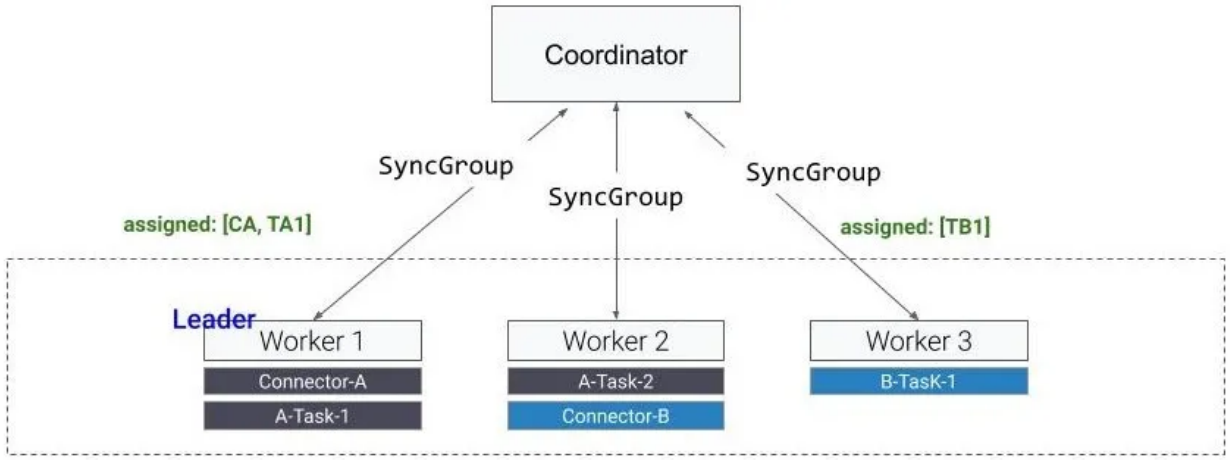
5 — B rejoins the group before delay expire and a rebalance is triggered



6 — W1 becomes leader and computes assignments



7 — W1, W2, W3 receive assignments



8 — After delay, all members join