

Purgatory는 무엇인가?

강 한구

Index

- Purgatory
- 기존 Purgatory in Kafka
- Timer 구현하는 7가지 방법
- 개선된 Purgatory in Kafka
- 질문으로 알아보는 Purgatory

Purgatory

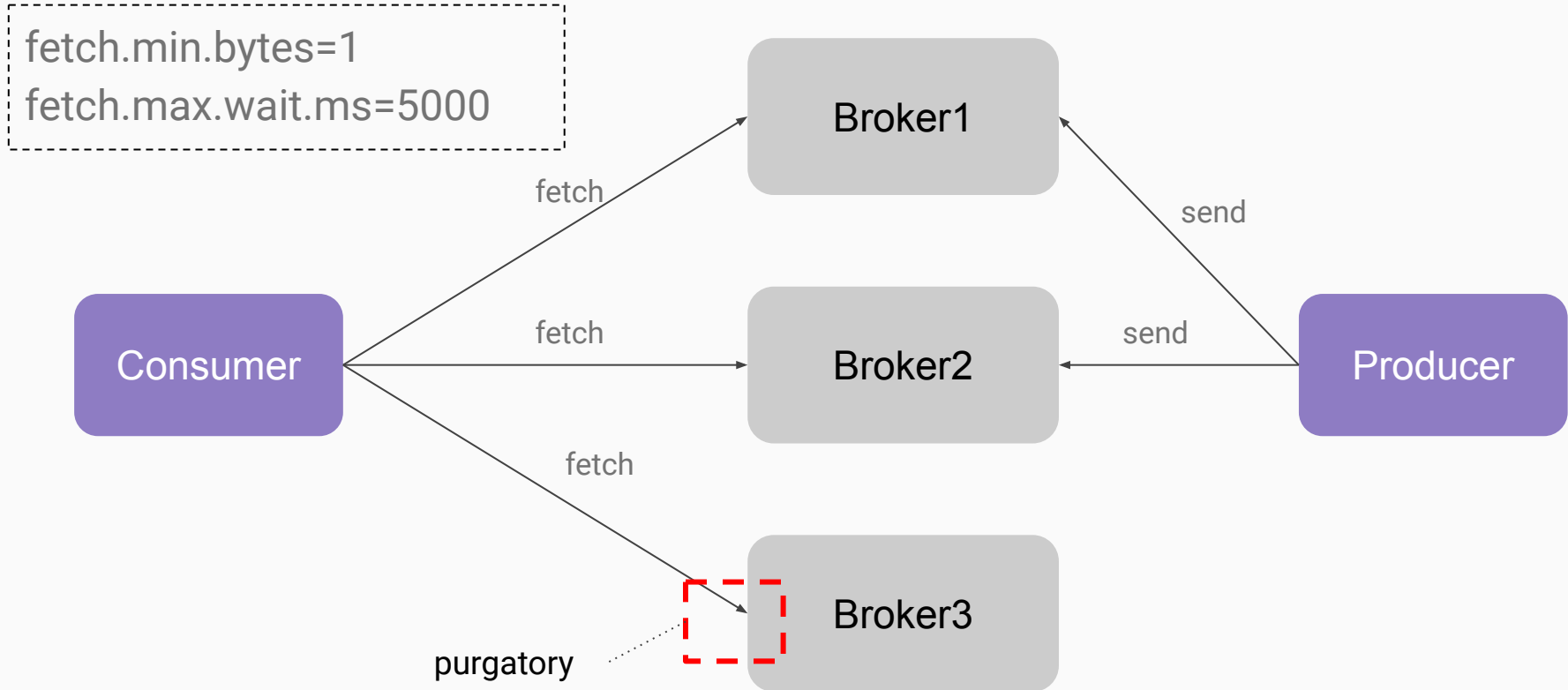
가톨릭 교리에서 죽은 사람의 영혼이 살아있는 동안 지은 죄를 씻고 천국으로 가기 위해 일시적으로 머무른다고 믿는 장소이다.

두산백과 두피디아 terms.naver.com

연옥 지식백과

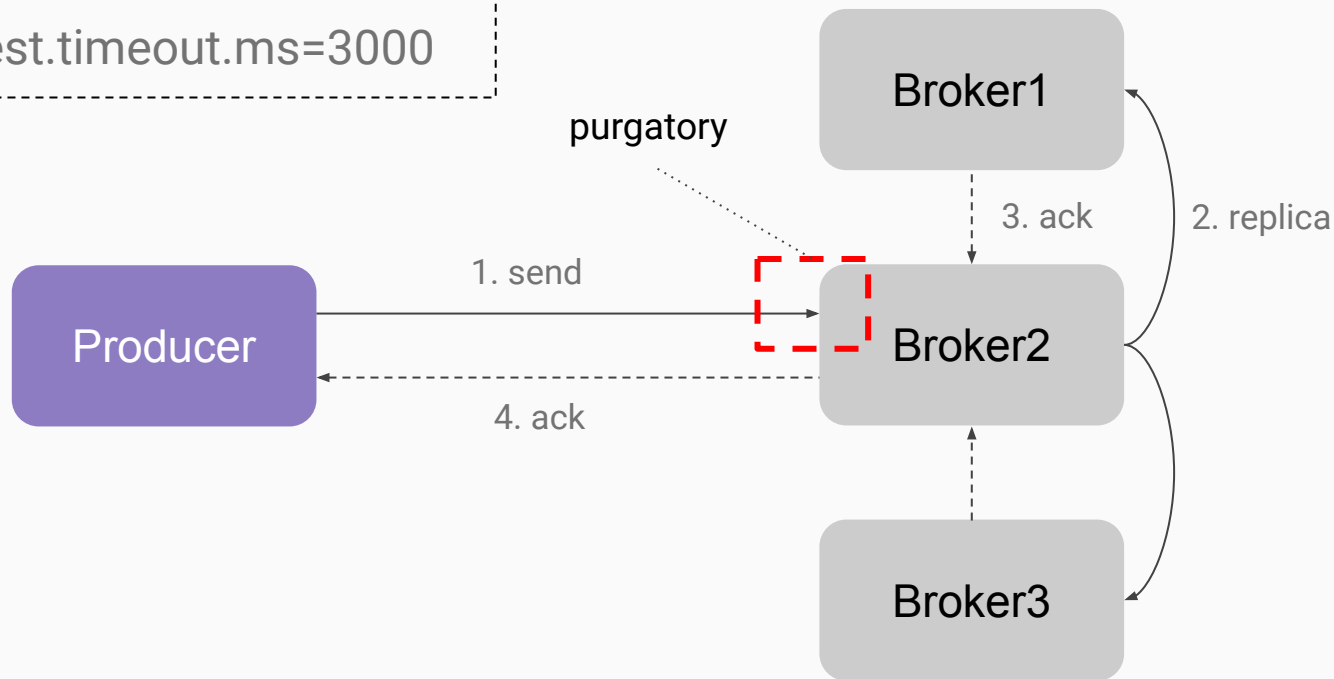
- 요청 조건을 만족하지 않았거나 지정한 시간이 지나지 않은 대기 장소
- 조건을 만족하면 정상 응답, 지정한 시간이 지나면 **Expiry** 처리

Purgatory

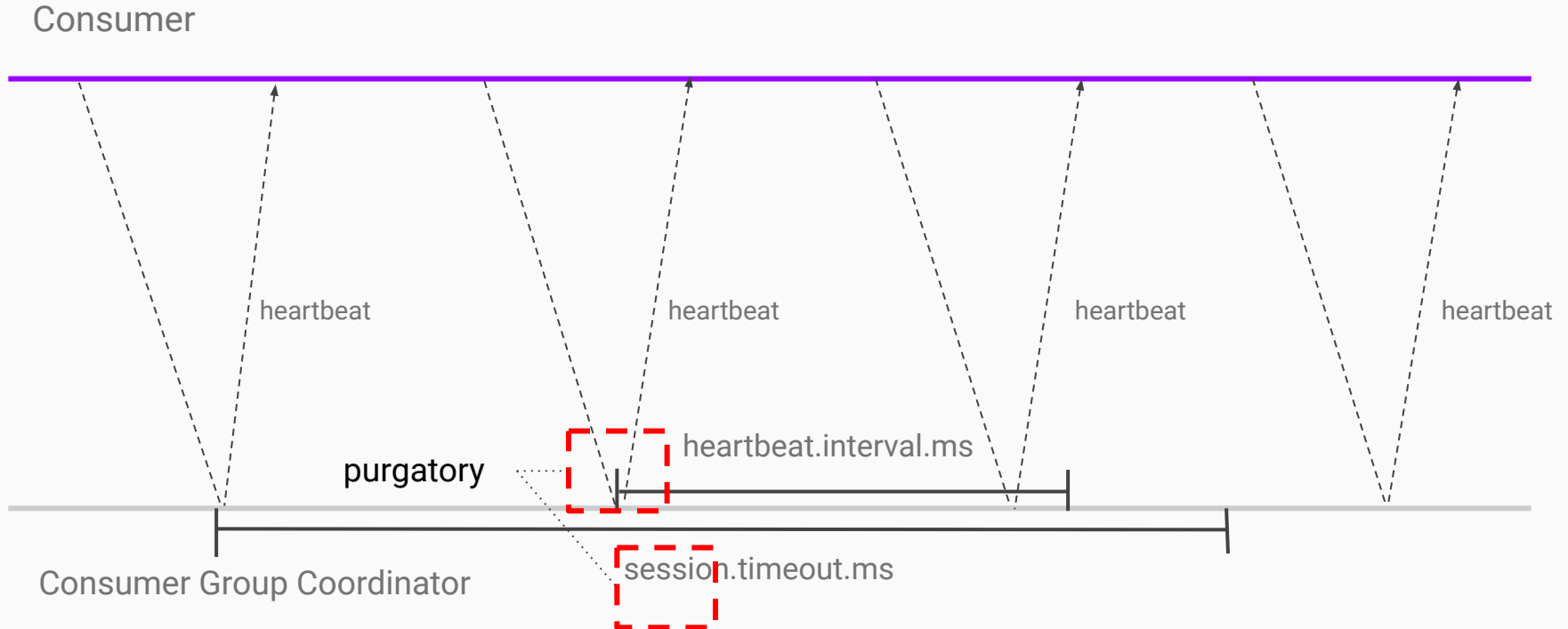


Purgatory

ack=all or ack=-1
request.timeout.ms=3000

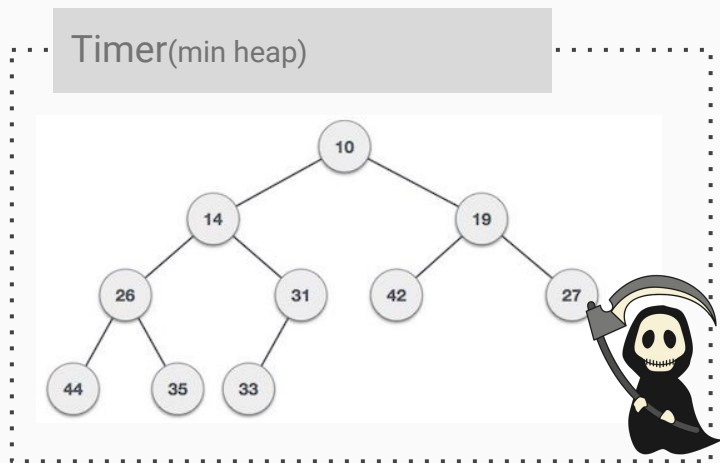


Purgatory



기존 Purgatory in Kafka

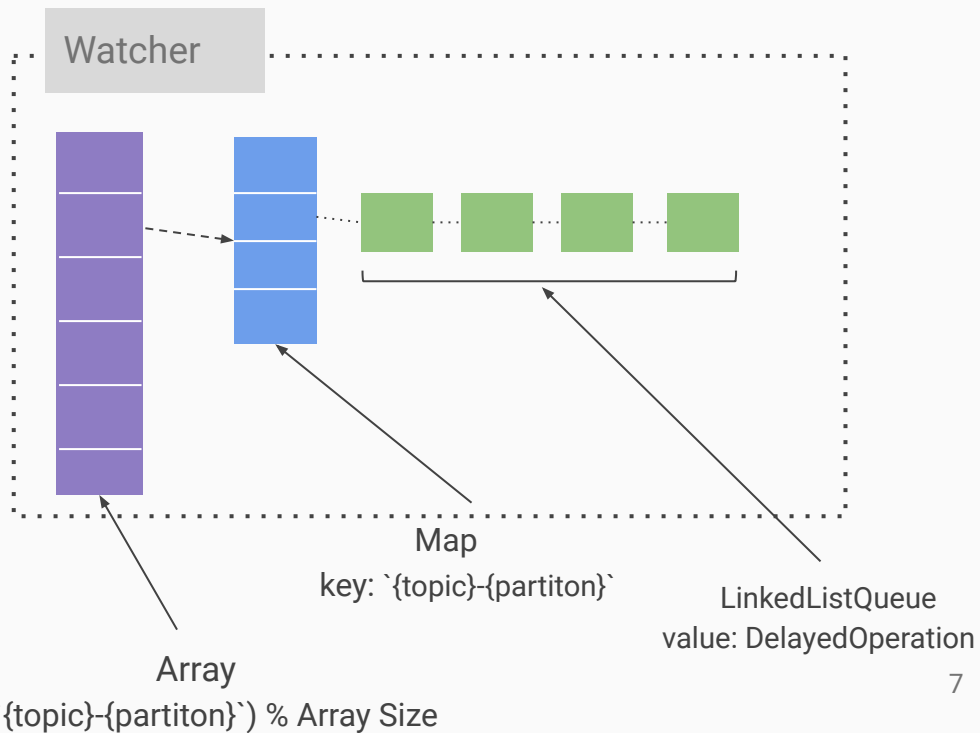
등록된 Expire task의 Expire 체크



완료 또는 Expire 된 task는 바로 삭제 하지 않고 Reaper thread가 주기적으로 Timer, Watcher를 돌면서 task 정리

- . Timer task 등록, 삭제가 빈번한데 시간복잡도 $O(\log n)$
- . Reaper thread 주기를 짧게 하면 정리는 되지만 cpu cost 높음
- 주기를 길게 하면 cpu cost는 낮지만 정리 되지 않아 OOM 유발

등록된 Expire task의 조건 만족 판단



Timer 구현하는 7가지 방법

Client API

Start: Expire task 등록

Stop: 등록된 Expire task 삭제

Timer

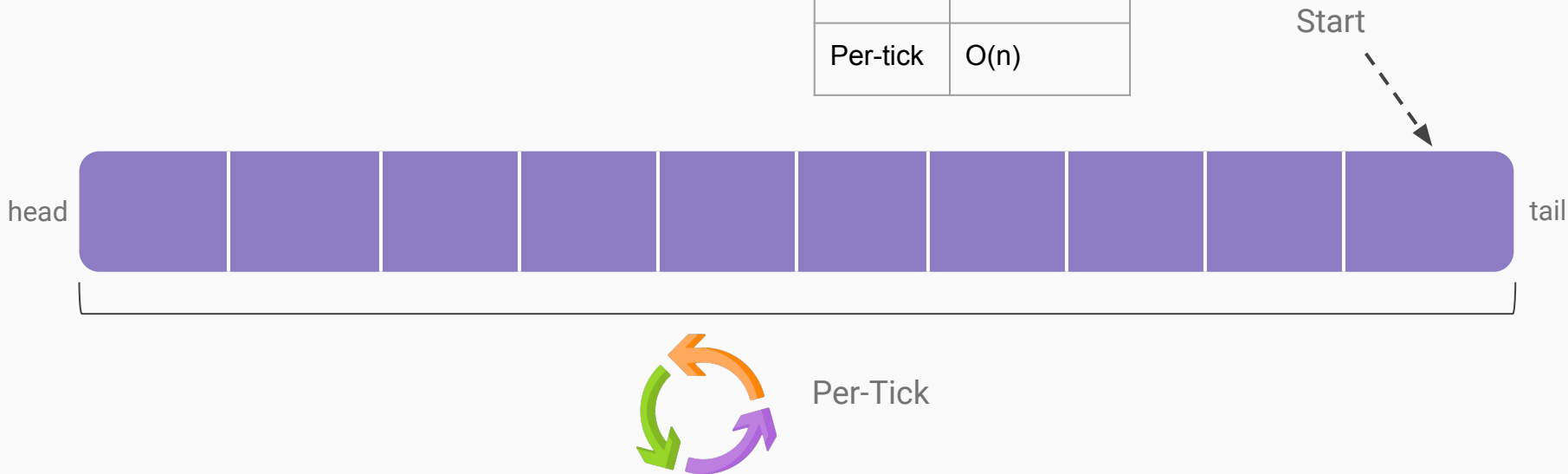
Per-tick book-keeping: 매 Time-Unit마다 Expire 될 task 탐색

Expiry processing: Expire 됐을 때 처리 로직

Timer 구현하는 7가지 방법

scheme 1: Unordered Timer List

	시간 복잡도
Start	$O(1)$
Per-tick	$O(n)$



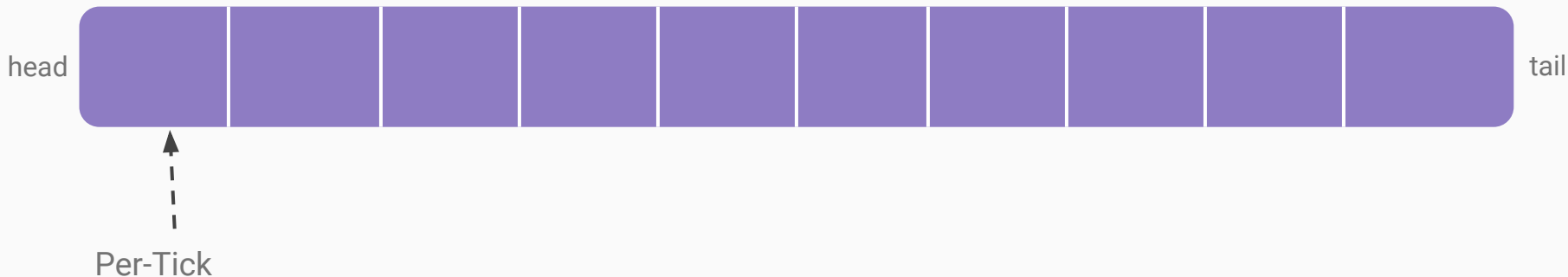
Timer 구현하는 7가지 방법

Scheme 2: Ordered List Timer List

Start ※ list가 expire 기준 정렬을 유지하는 위치 탐색 후 삽입

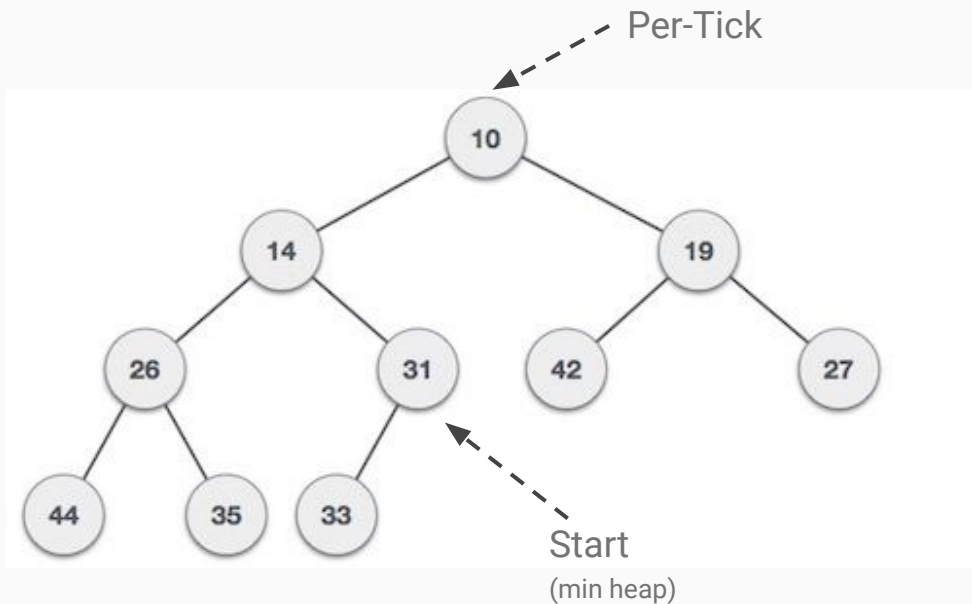


	시간 복잡도
Start	$O(n)$
Per-tick	$O(1)$



Timer 구현하는 7가지 방법

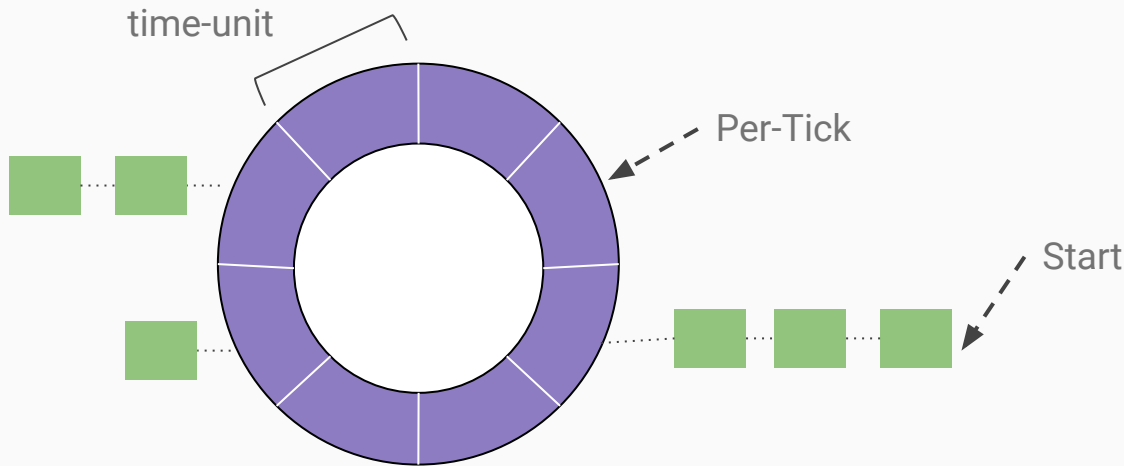
Scheme 3: Tree-based Timer (min heap)



	시간 복잡도
Start	$O(\log n)$
Per-tick	$O(1)$

Timer 구현하는 7가지 방법

Scheme 4: Simple Timing Wheels



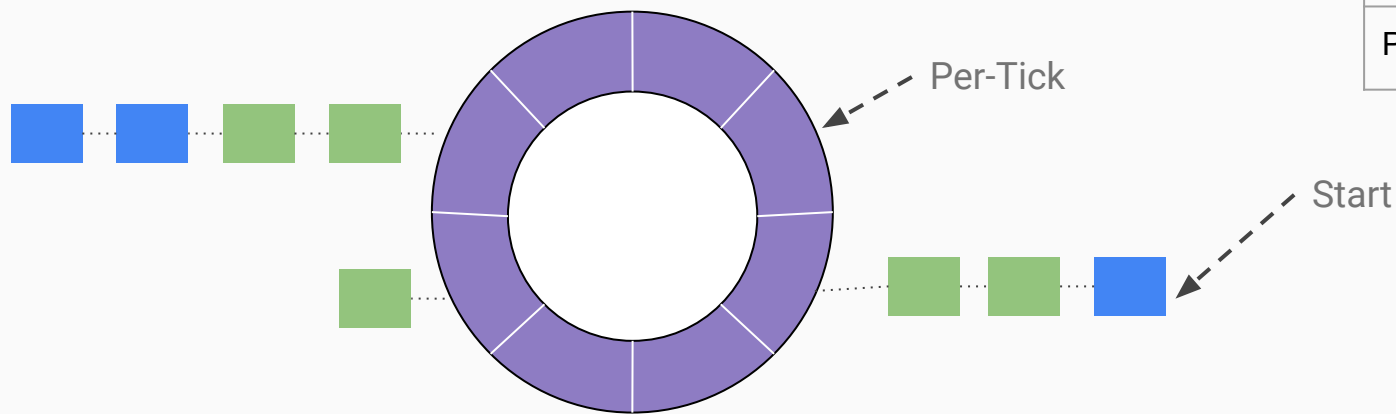
	시간 복잡도
Start	$O(n)$
Per-tick	$O(1)$

. time-unit이 1초라고 가정하면 위 Timing Wheel은 최대 8초 Expire 가능

. 8초 이상인 Expire task는 오류 발생, $(\text{Expire Time} \% 8)$ 값에 해당되는 bucket에 Expire task 추가

Timer 구현하는 7가지 방법

Scheme 5: Hashing Timing Wheel with Ordered Timer List



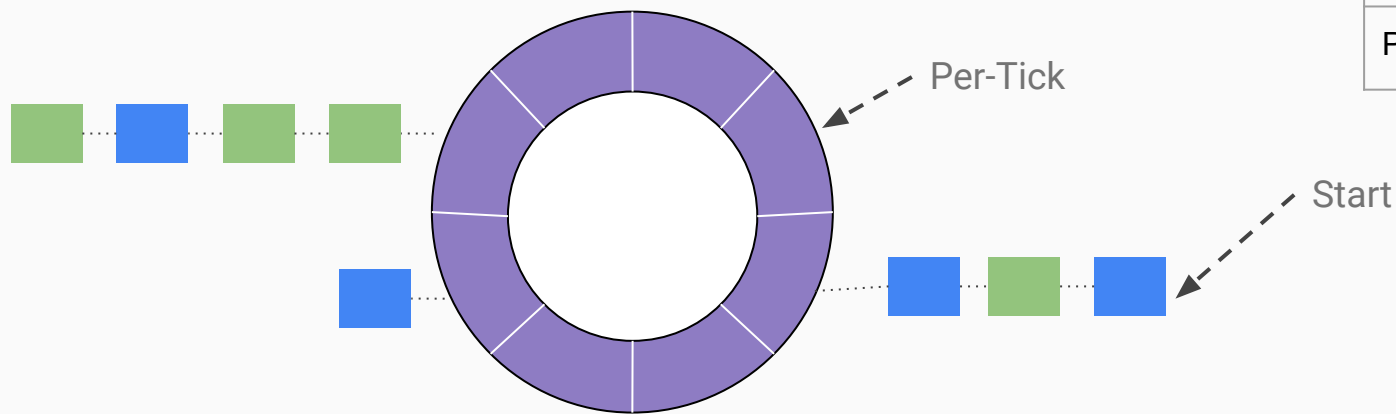
	시간 복잡도
Start	$O(1)$
Per-tick	$O(n)$

. 하나의 Timing wheel로 8초 이상의 Expire도 가능한 개념

. 8초 이상인 Expire task도 오류 발생 하지 않고, $(\text{Expire Time} \% 8)$ 값에 해당되는 bucket에 Expire task 추가

Timer 구현하는 7가지 방법

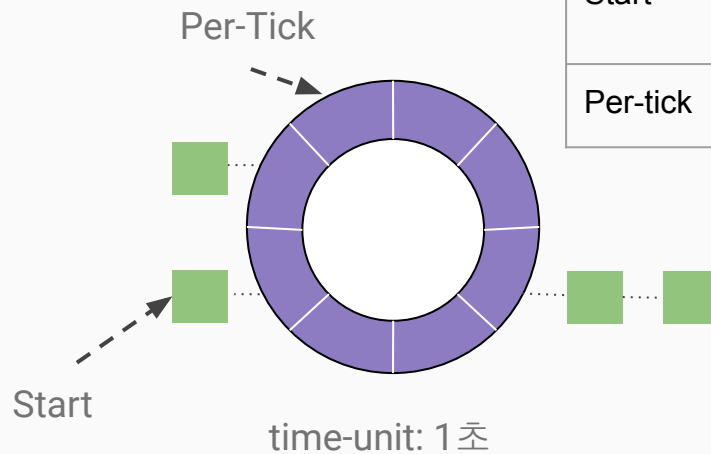
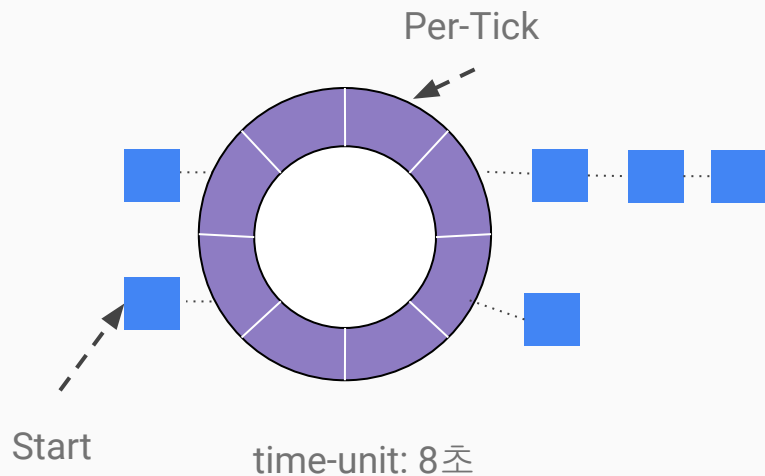
Scheme 6: Hashing Timing Wheel with UnOrdered Timer List



	시간 복잡도
Start	$O(n)$
Per-tick	$O(1)$

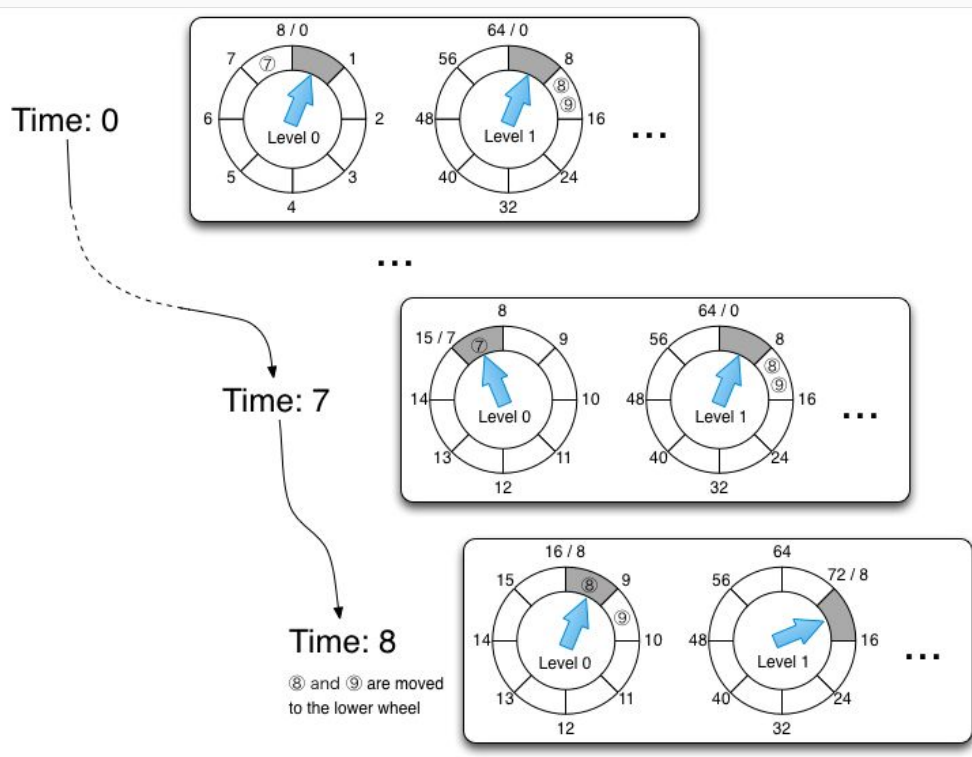
Timer 구현하는 7가지 방법

Scheme 7: Hierarchical Timing Wheel



	시간 복잡도
Start	$O(m)$ m: wheel size
Per-tick	$O(1)$

Timer 구현하는 7가지 방법



time_unit 1초 Level 0 timing wheel
time_unit 8초 Level 1 timing wheel

7초 Expire task 7번
8초 Expire task 8번
9초 Expire task 9번

Timer 구현하는 7가지 방법

Scheme	Name	Start	Per-Tick
1	UnOrdered Timer List	$O(1)$	$O(n)$
2	Ordered Timer List	$O(n)$	$O(1)$
3	Tree-based Timer	$O(\log(n))$	$O(1)$
4	Simple Timing Wheel	$O(1)$	$O(1)$
5	Hashing Timing Wheel w/ Ordered Timer List	$O(n)$ worst $O(1)$ average	$O(1)$
6	Hashing Timing Wheel w/ UnOrdered Timer List	$O(1)$	$O(n)$ worst $O(1)$ average
7	Hierarchical Timing Wheel	$O(m)$ <small>m: number of wheels</small>	$O(1)$



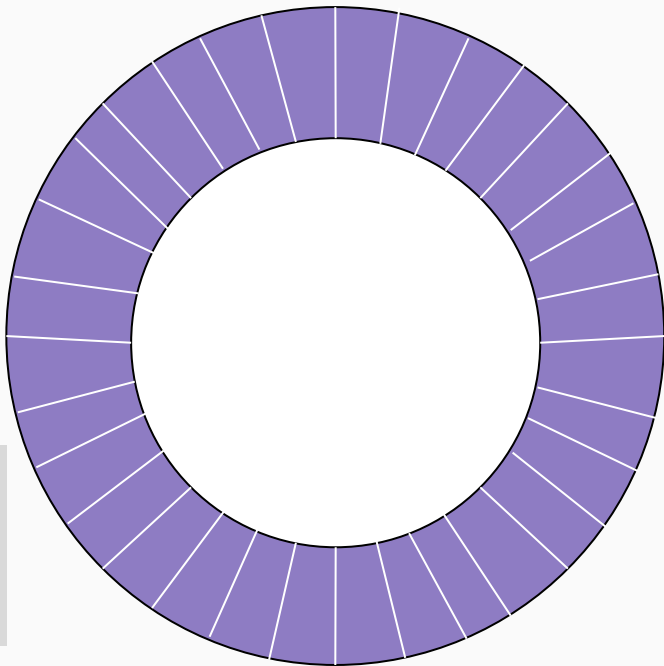
Timer 구현하는 7가지 방법

Scheme 4로 0 ~ 41 expire 가능한 timing wheel을 만든다면?



bucket 갯수가 Expire 시간 만큼 증가한다는
단점은 있지만

Expire가 제한 적이고 max값이 크지 않은
서비스에선 **scheme4**로 충분하다고 생각

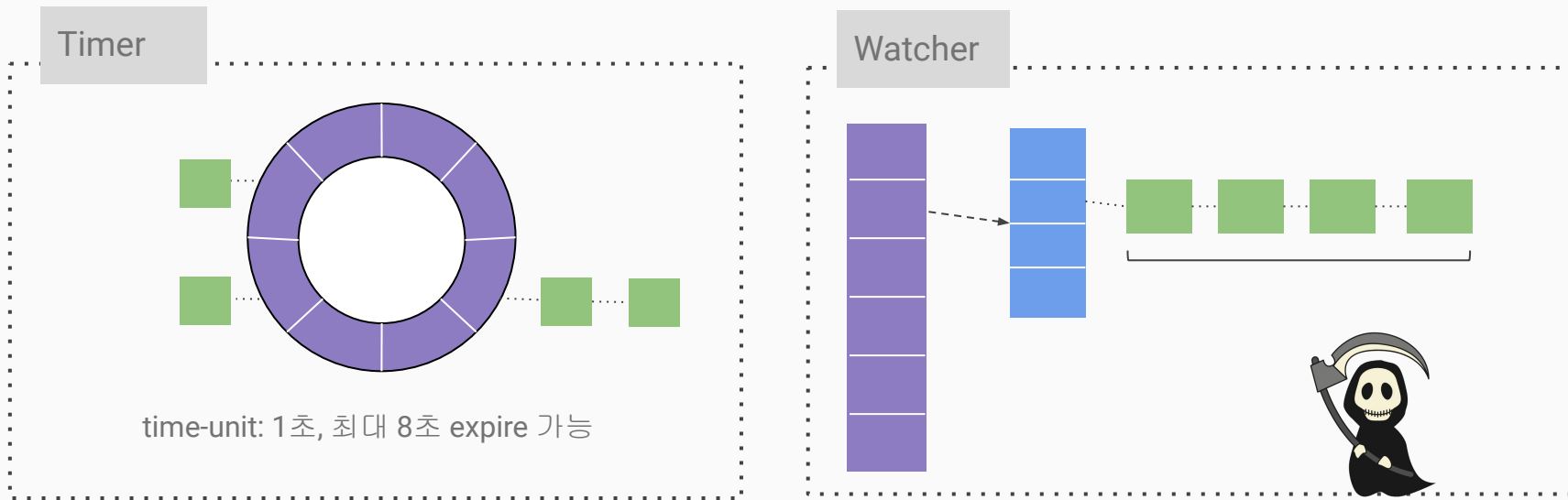


time-unit: 1 초

bucket: 42개 (그림은 42개 아님)

interval: 42초

개선된 Purgatory in Kafka



장점: 조건이 만족되면 Watcher와 Timer에서 바로 삭제가 가능하다. Reaper Thread가 탐색의 범위가 Watcher로 제한되어 부하 감소
Timer에 Expire task를 등록하고 삭제하는 시간 복잡도가 각각 $O(m)$, $O(1)$

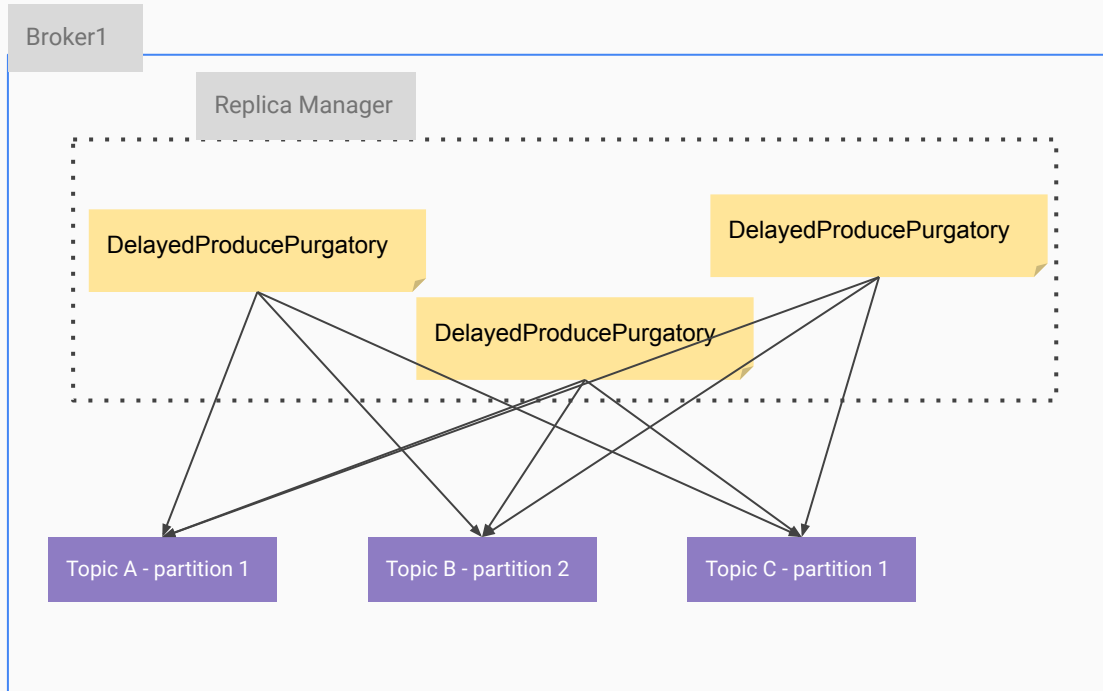
단점: Timer에서 계층별 migration 하는 추가 작업이 필요

개선된 Purgatory in Kafka

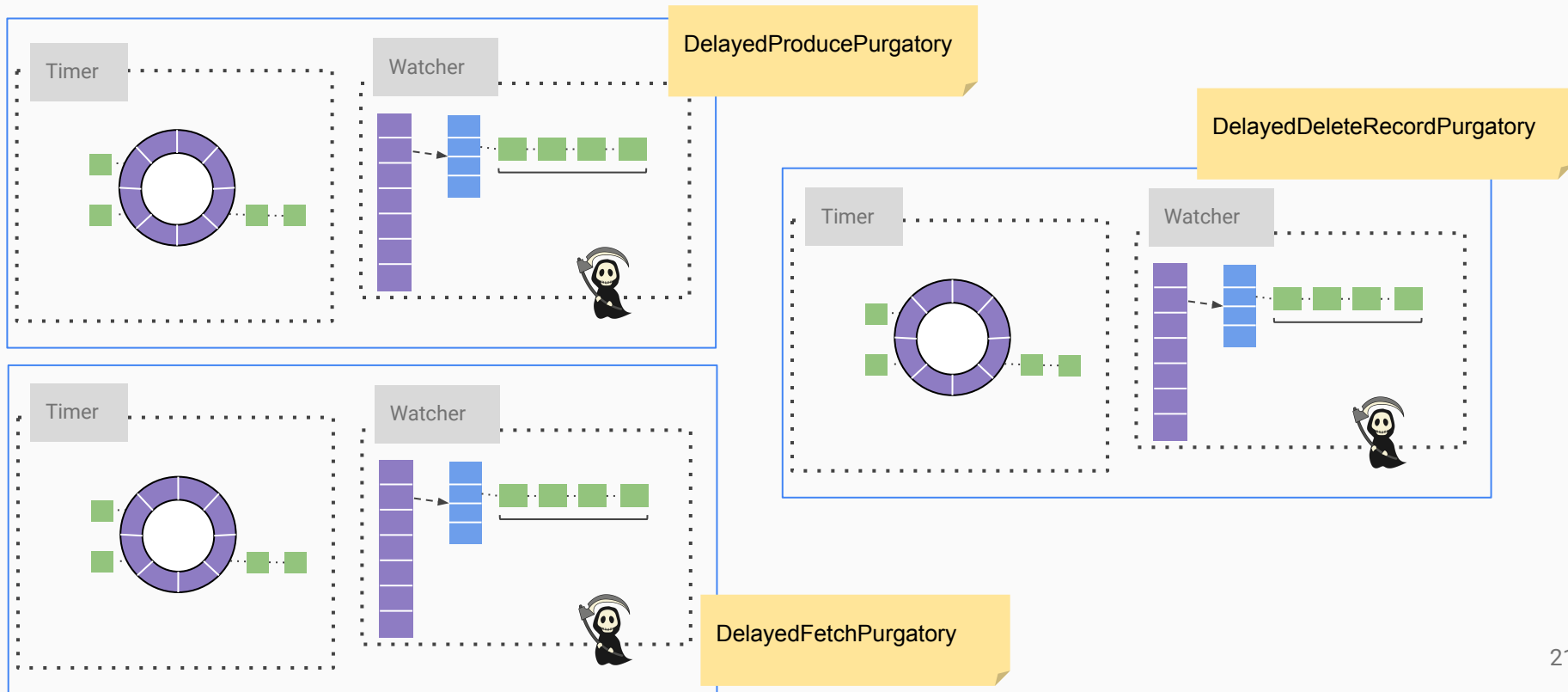
```
val delayedOperations = new DelayedOperations(
    topicPartition,
    replicaManager.delayedProducePurgatory,
    replicaManager.delayedFetchPurgatory,
    replicaManager.delayedDeleteRecordsPurgatory)

new Partition(topicPartition,
    replicaLagTimeMaxMs = replicaManager.config.replicaLagTimeMaxMs,
    interBrokerProtocolVersion = replicaManager.config.interBrokerProtocolVersion,
    localBrokerId = replicaManager.config.brokerId,
    time = time,
    isrChangeListener = isrChangeListener,
    delayedOperations = delayedOperations,
    metadataCache = replicaManager.metadataCache,
    logManager = replicaManager.logManager,
    alterIsrManager = replicaManager.alterIsrManager)
```

Broker마다 ReplicaManager 하나씩 존재.
그 안에 Purgatory가 생성되어있는데
Partition 생성될 때 주입합니다.

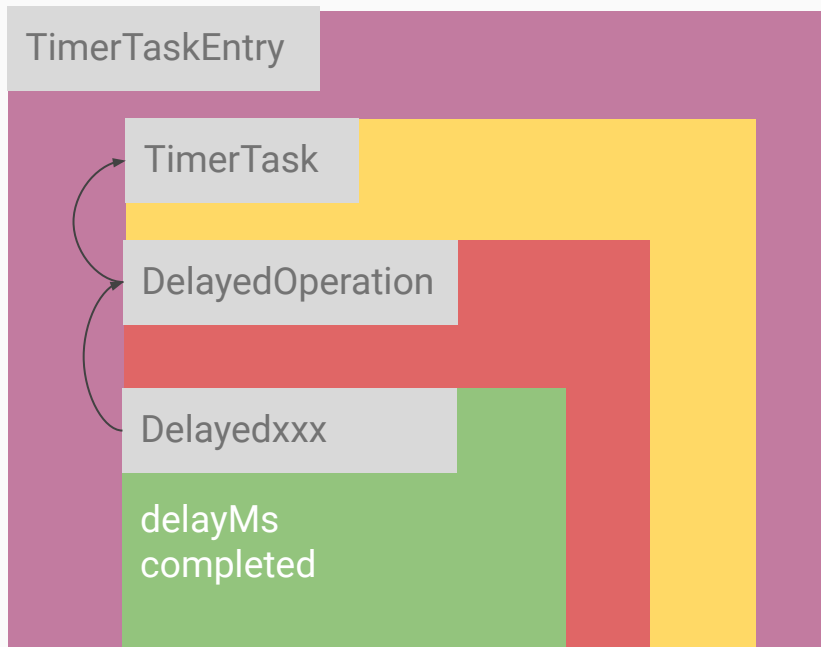


개선된 Purgatory in Kafka



개선된 Purgatory in Kafka

```
DelayedCreatePartitions (kafka.server)
DelayedDeleteRecords (kafka.server)
DelayedDeleteTopics (kafka.server)
DelayedElectLeader (kafka.server)
DelayedFetch (kafka.server)
DelayedFuture (kafka.server)
DelayedHeartbeat (kafka.coordinator.group)
DelayedJoin (kafka.coordinator.group)
DelayedProduce (kafka.server)
DelayedRebalance (kafka.coordinator.group)
DelayedSync (kafka.coordinator.group)
```

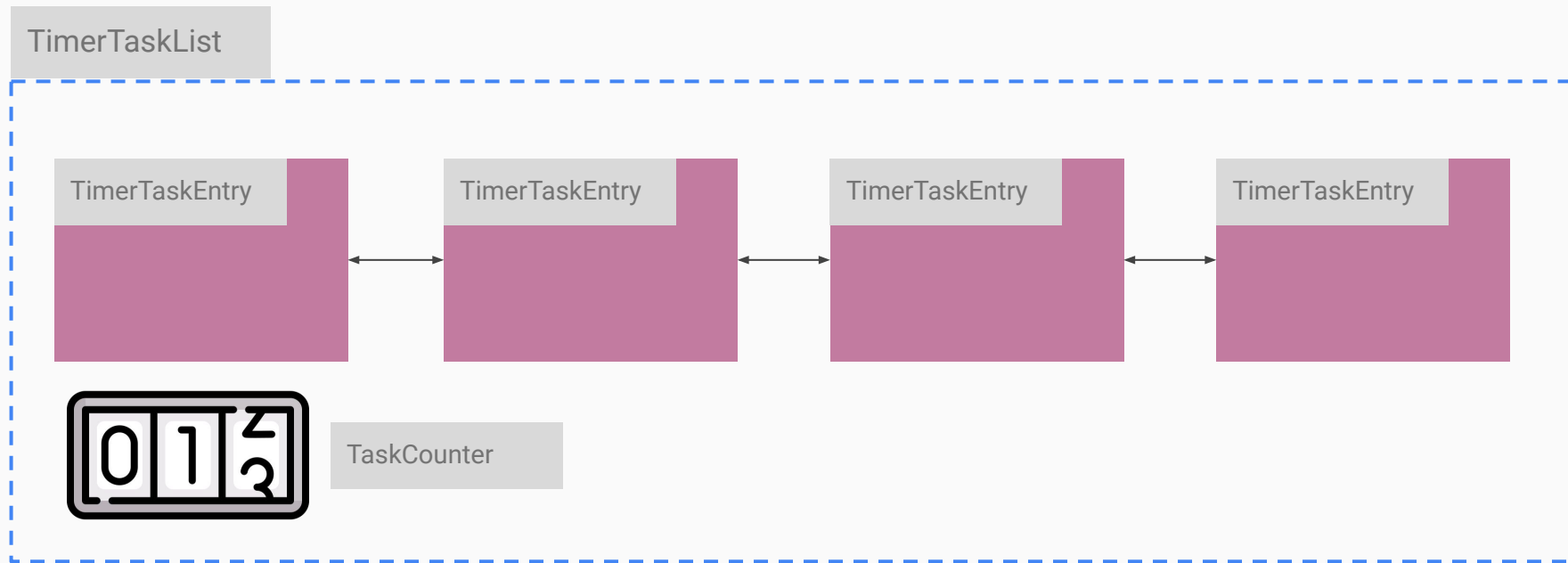


```
class TimerTaskEntry {
    TimerTask
    ...
}
```

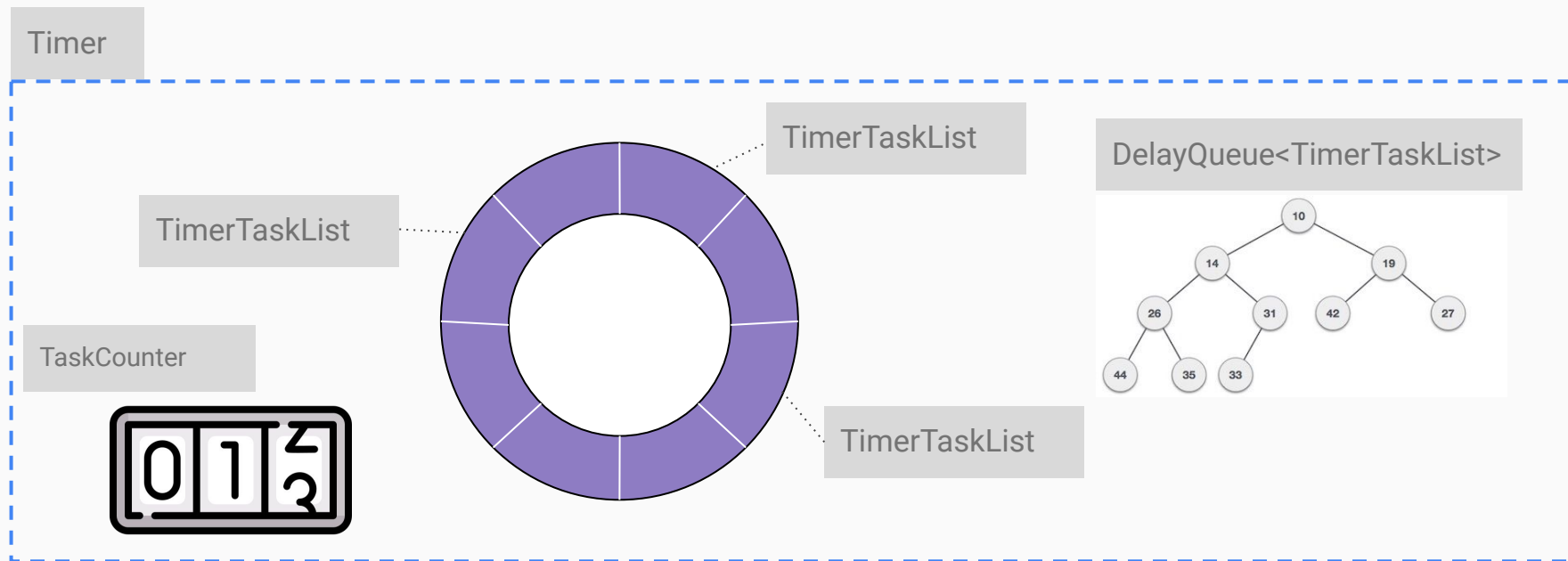
```
class DelayedOperation
extends TimerTask
```

```
class Delayedxxx
extends
    DelayedOperation
```

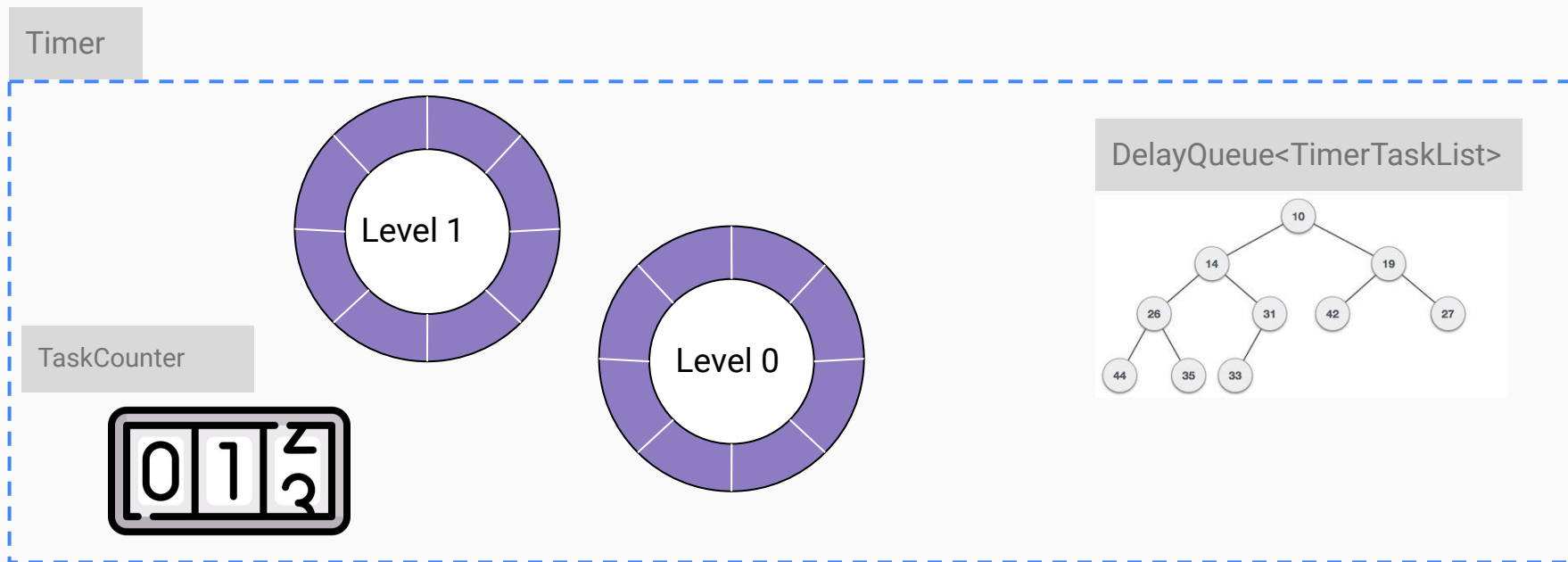
개선된 Purgatory in Kafka



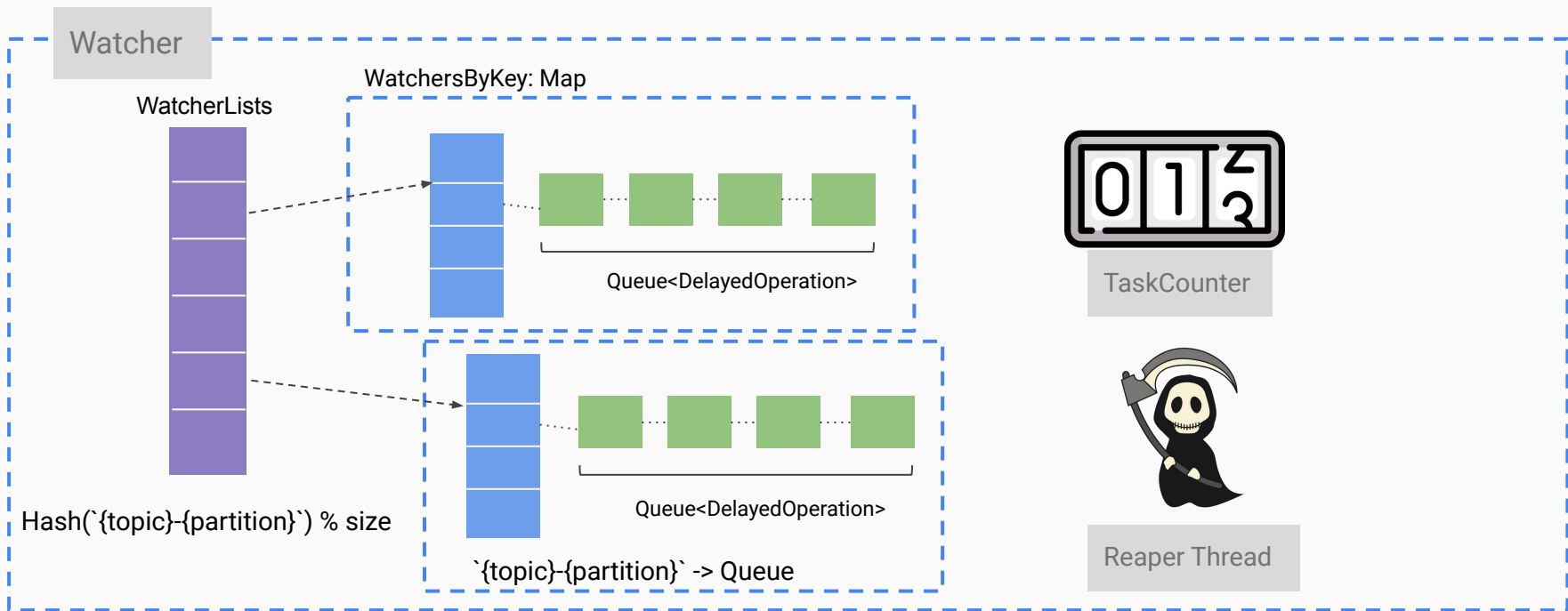
개선된 Purgatory in Kafka



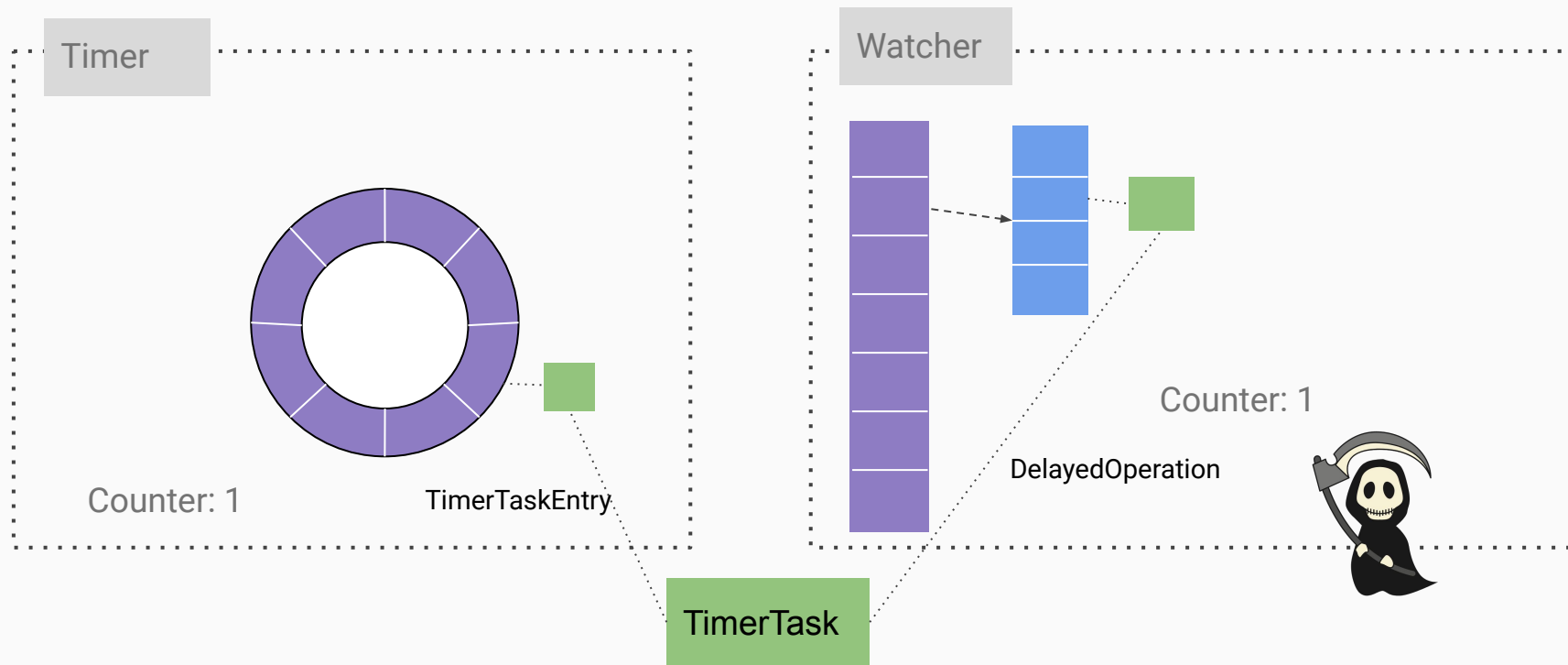
개선된 Purgatory in Kafka



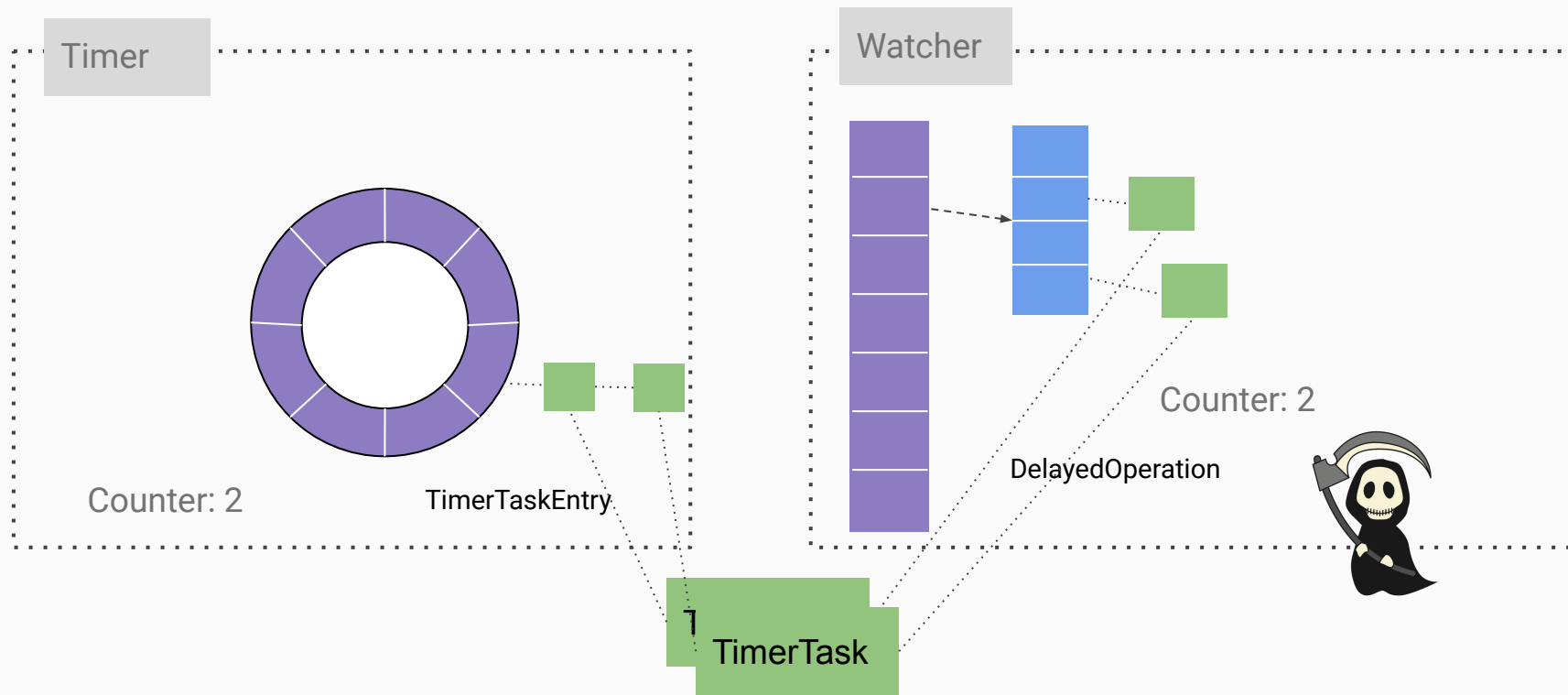
개선된 Purgatory in Kafka



개선된 Purgatory in Kafka

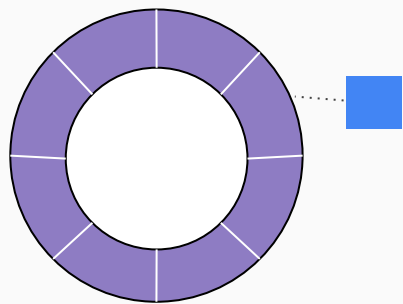


개선된 Purgatory in Kafka

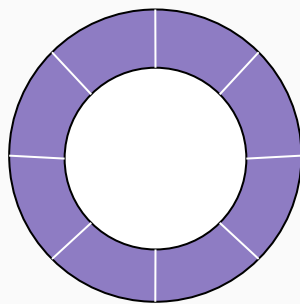


질문으로 알아보는 Purgatory

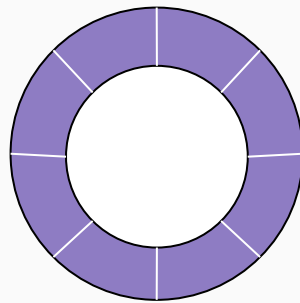
Q1. `fetch.max.wait.ms = 92000ms`로 했을 때 Timer 동작은 어떻게?



time-unit: 64초



time-unit: 8초



time-unit: 1초

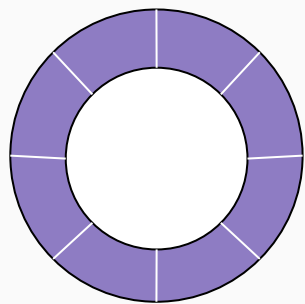
DelayQueue<TimerTaskList>

64

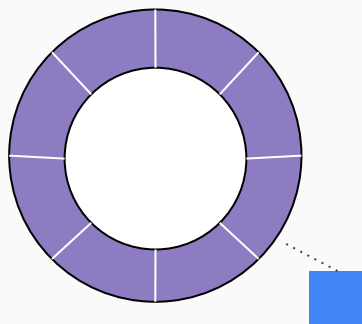
92초는 time-unit 1초 Timing Wheel에 추가되지 않아, time-unit 64초
Timing Wheel에 $1(92 / 64)$ bucket에 Expire task 추가
DelayQueue에는 64초 Expire로 추가

질문으로 알아보는 Purgatory

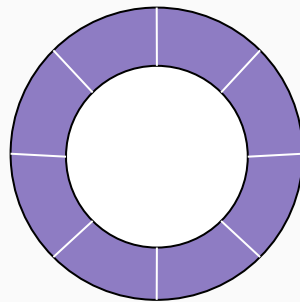
Q1. `fetch.max.wait.ms = 92000ms`로 했을 때 Timer 동작은 어떻게?



time-unit: 64초



time-unit: 8초



time-unit: 1초

DelayQueue<TimerTaskList>

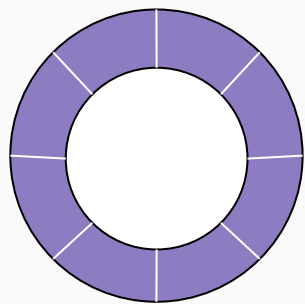
16

64초가 지나면 time-unit 64초 Timing Wheel에 있던 Expire task는 time-unit 8초 Timing Wheel로 migration 한다.

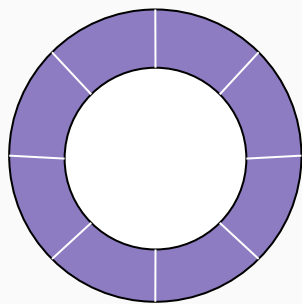
앞에 설명한 방식과 동일하게 Expire task 및 DelayQueue에 추가

질문으로 알아보는 Purgatory

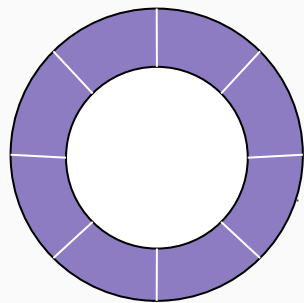
Q1. `fetch.max.wait.ms = 92000ms`로 했을 때 **Timer** 동작은 어떻게?



time-unit: 64초



time-unit: 8초



time-unit: 1초

DelayQueue<TimerTaskList>

2

질문으로 알아보는 Purgatory

Q2. 조건을 만족했을 때 Purgatory에서는 어떤일이 일어나는가?

appendRecord

delayedProducerPurgatory.checkAndComplete

delayedFetchPurgatory.checkAndComplete

delayedDeleteRecordPurgatory.checkAndComplete

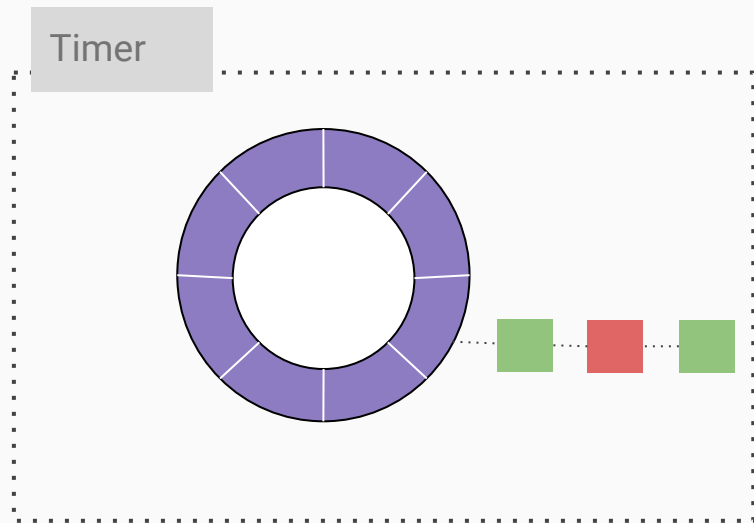
Leader 파티션에 메시지가 추가될 때마다 해당 파티션의 Purgatory 조건 만족 했는지 체크

Watcher

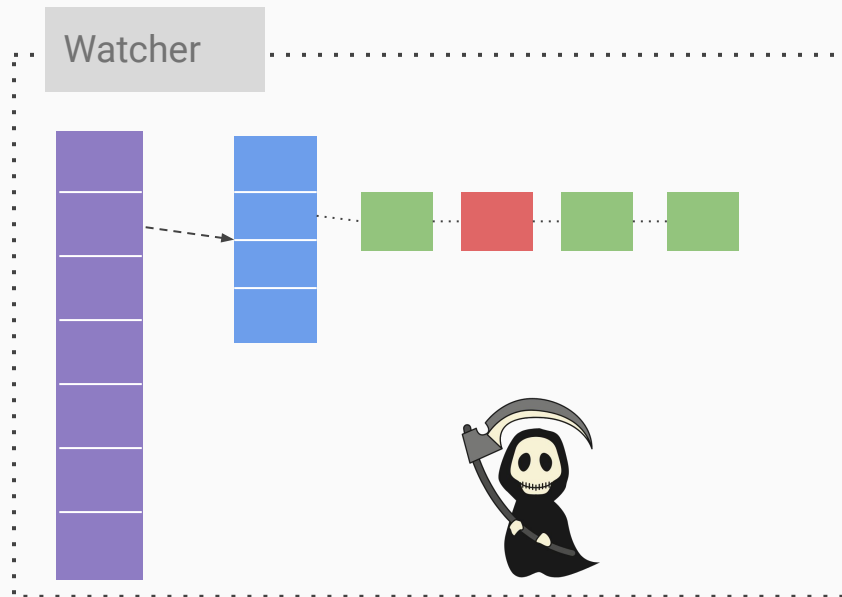


질문으로 알아보는 Purgatory

Q2. 조건을 만족했을 때 Purgatory에서는 어떤일이 일어나는가?

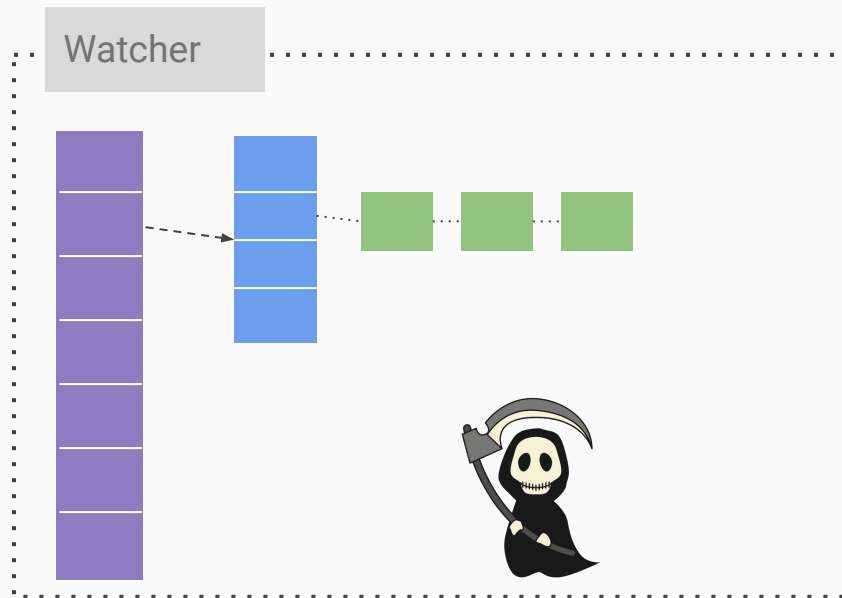
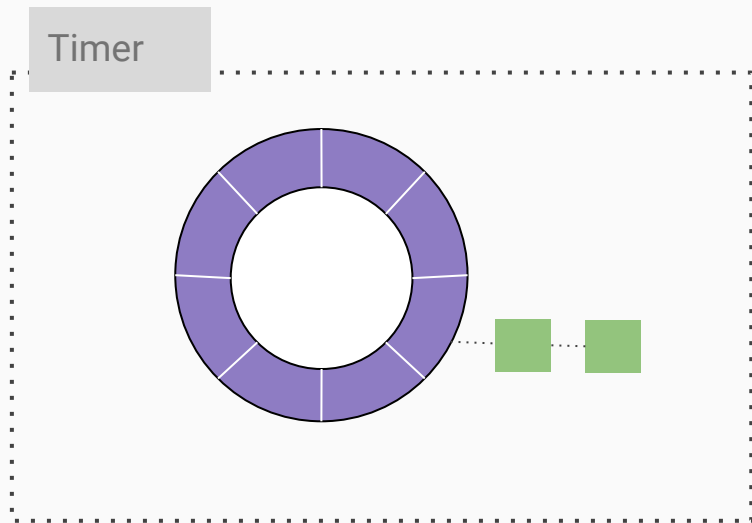


Watcher에서 조건을 만족한 Expire task를 찾았다면 Timer도 삭제 준비
`DelayOperation(Watcher) -> TimerTask -> TimerTaskEntry(Timer)`

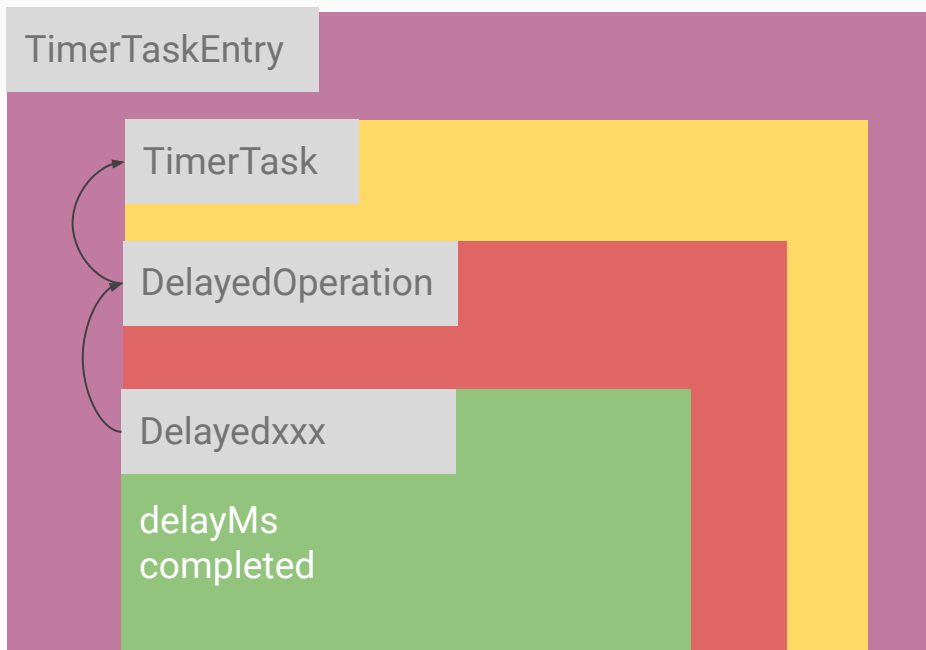


질문으로 알아보는 Purgatory

Q2. 조건을 만족했을 때 Purgatory에서는 어떤일이 일어나는가?

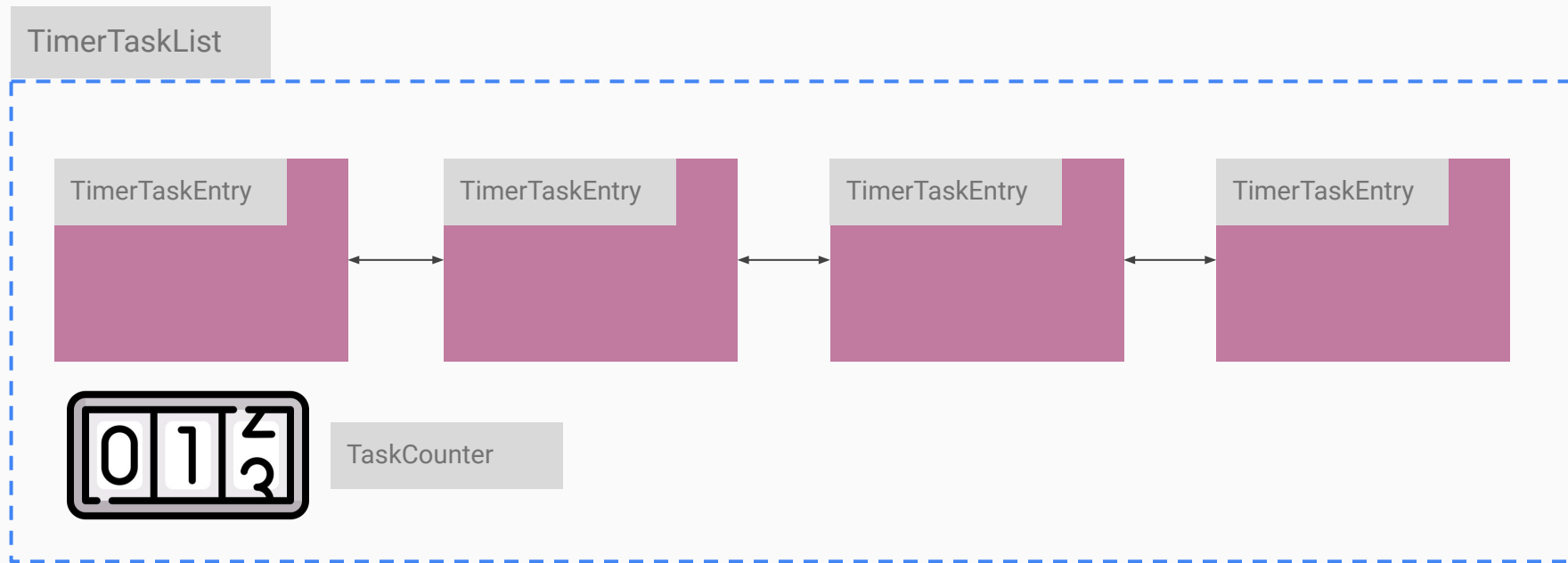


질문으로 알아보는 Purgatory



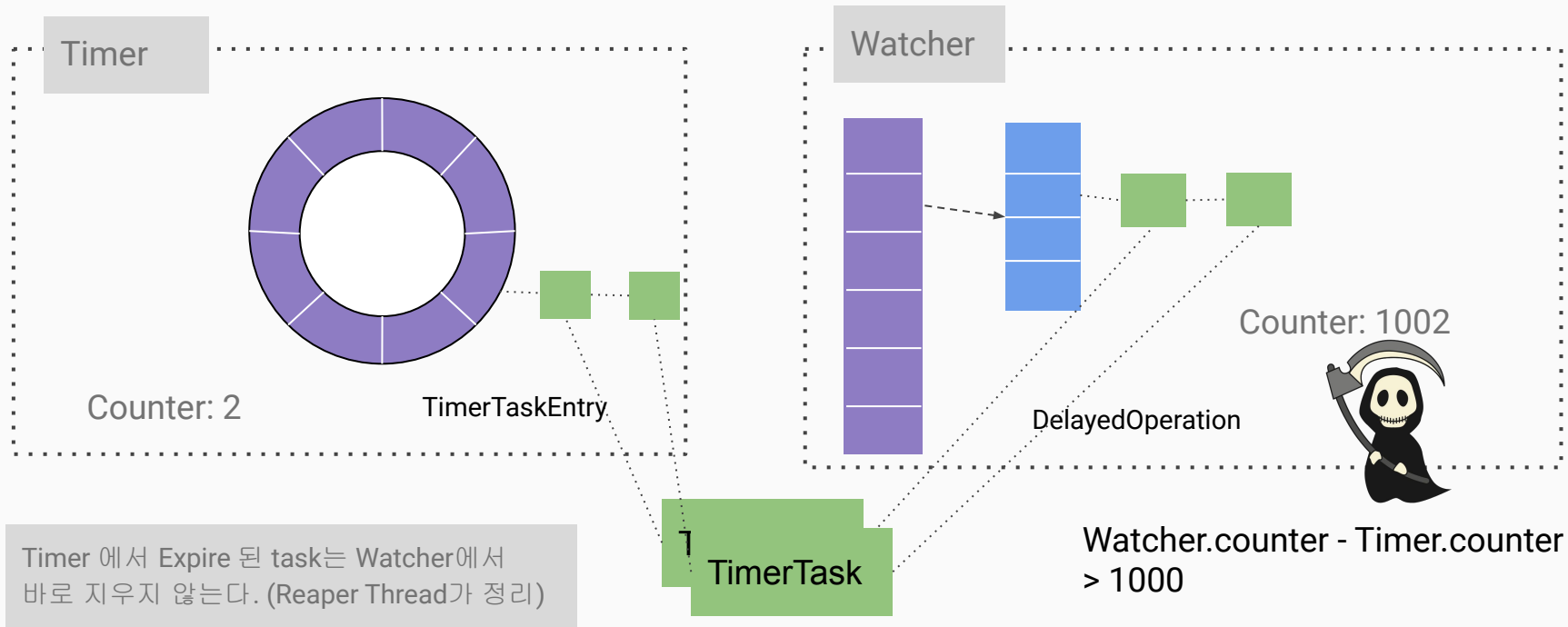
```
DelayedCreatePartitions (kafka.server)
DelayedDeleteRecords (kafka.server)
DelayedDeleteTopics (kafka.server)
DelayedElectLeader (kafka.server)
DelayedFetch (kafka.server)
DelayedFuture (kafka.server)
DelayedHeartbeat (kafka.coordinator.group)
DelayedJoin (kafka.coordinator.group)
DelayedProduce (kafka.server)
DelayedRebalance (kafka.coordinator.group)
DelayedSync (kafka.coordinator.group)
```

질문으로 알아보는 Purgatory



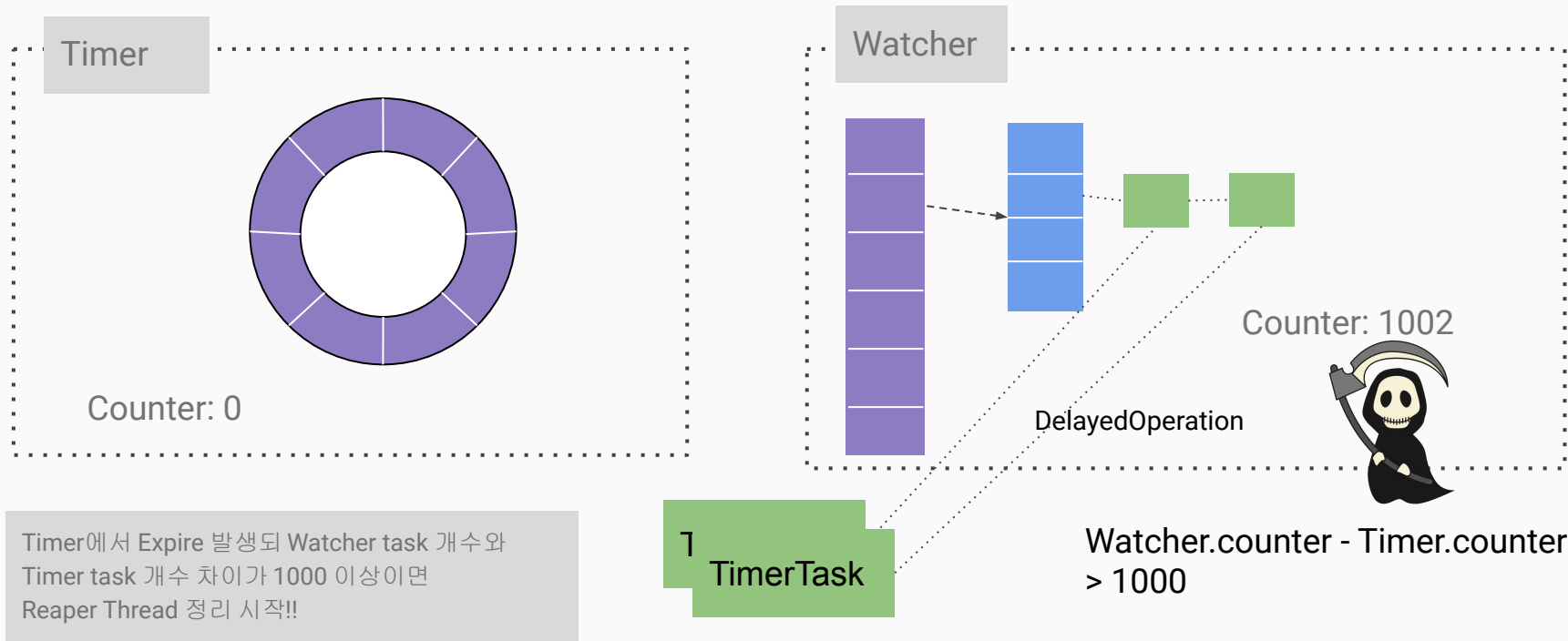
질문으로 알아보는 Purgatory

Q3. 조건을 만족하지 못하고 Expire 될 때 Purgatory에서는 어떤일이 일어나는가?



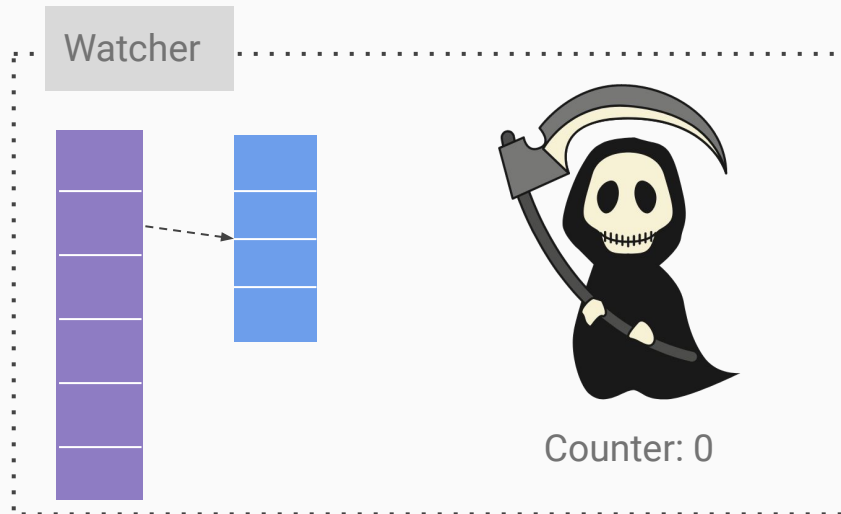
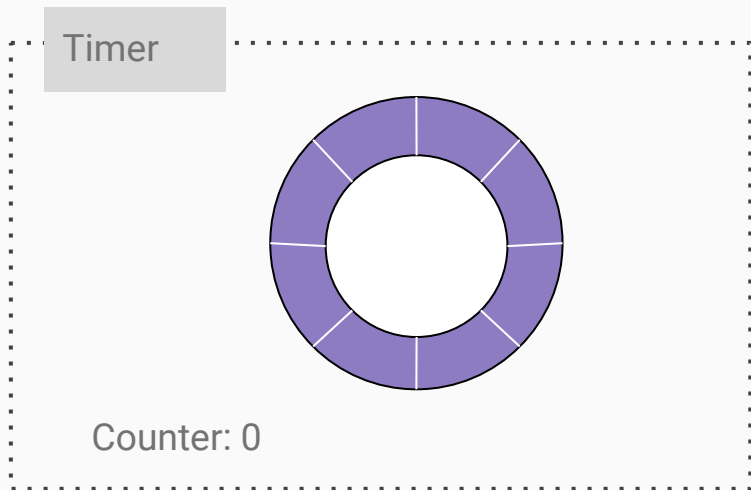
질문으로 알아보는 Purgatory

Q3. 조건을 만족하지 못하고 Expire 될 때 Purgatory에서는 어떤일이 일어나는가?



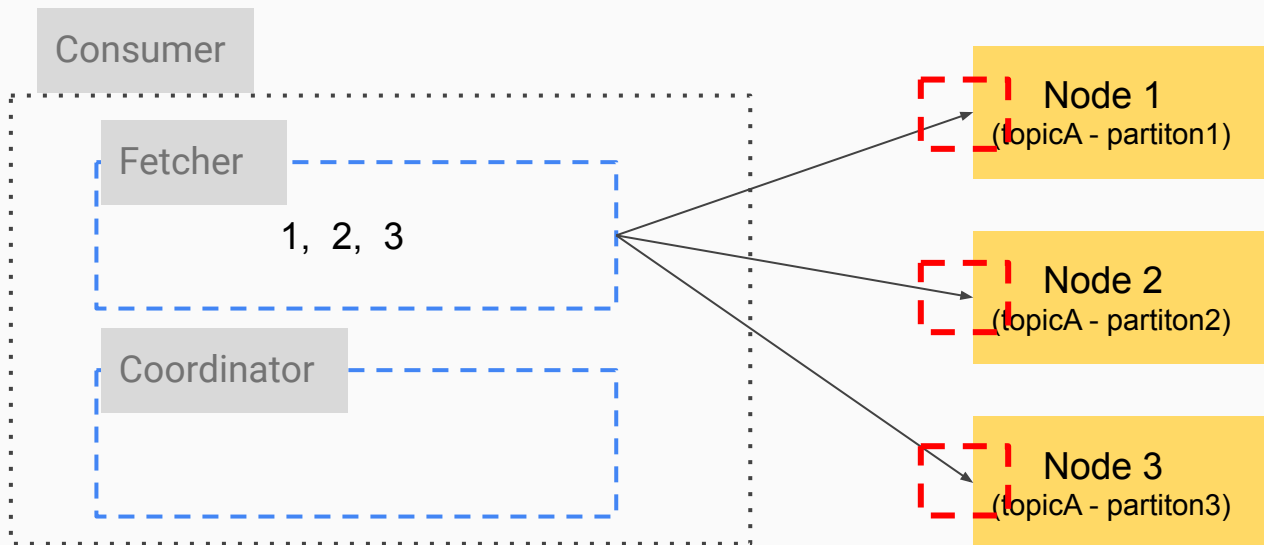
질문으로 알아보는 Purgatory

Q3. 조건을 만족하지 못하고 Expire 될 때 Purgatory에서는 어떤일이 일어나는가?



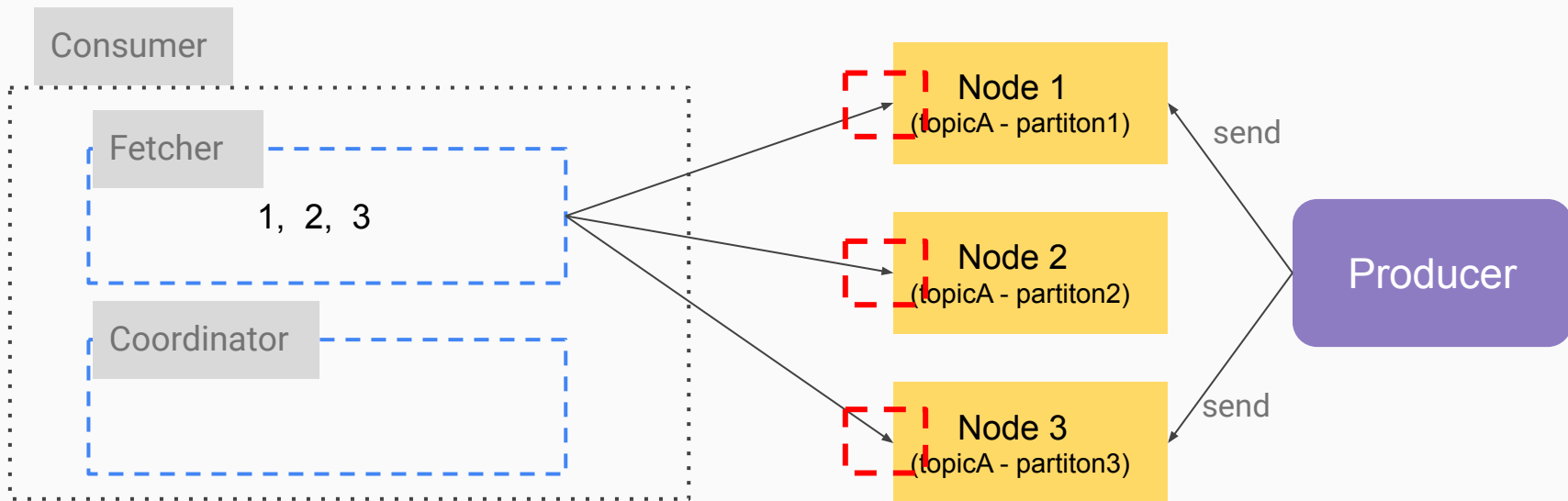
질문으로 알아보는 Purgatory

Q4. 3개 파티션을 **fetch** 중인 **consumer**가 2개는 조건을 만족해서 응답 받고 1개는 대기중 일 때 대기 중인 파티션에 또 **fetch** 요청 하면 안 될것 같다. 어떻게 동작할까?



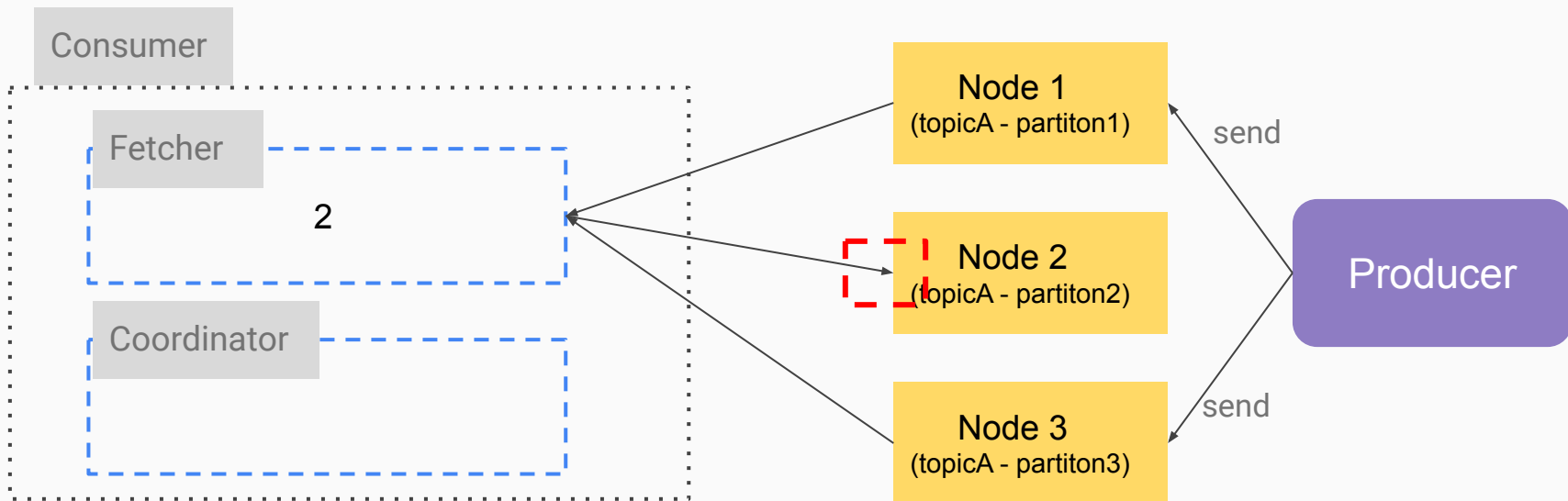
질문으로 알아보는 Purgatory

Q4. 3개 파티션을 **fetch** 중인 **consumer**가 2개는 조건을 만족해서 응답 받고 1개는 대기중 일 때 대기 중인 파티션에 또 **fetch** 요청 하면 안 될것 같다. 어떻게 동작할까?



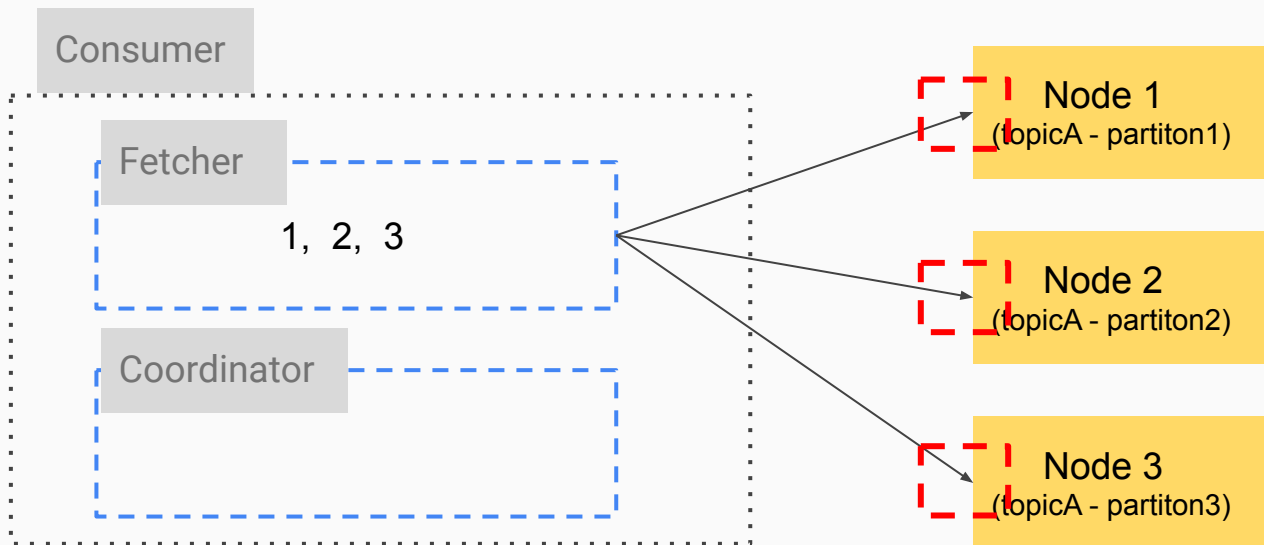
질문으로 알아보는 Purgatory

Q4. 3개 파티션을 fetch 중인 consumer가 2개는 조건을 만족해서 응답 받고 1개는 대기중 일 때 대기 중인 파티션에 또 fetch 요청 하면 안 될것 같다. 어떻게 동작할까?



질문으로 알아보는 Purgatory

Q4. 3개 파티션을 **fetch** 중인 **consumer**가 2개는 조건을 만족해서 응답 받고 1개는 대기중 일 때 대기 중인 파티션에 또 **fetch** 요청 하면 안 될것 같다. 어떻게 동작할까?



정리

- . Expire task 추가 될때는 $O(m)$
- . 조건을 만족하면 **Watcher**와 **Timer**에서 바로 삭제
- . Expire되면 **Timer**에서 바로 삭제,
Watcher에선 바로 삭제 하지 않고 **Reaper Thread**가 정리
- . **Purgatory** 기능을 활용하기 위해서 **Client**에 추가 로직

Q & A

참고한 url

<https://www.confluent.io/ko-kr/blog/apache-kafka-purgatory-hierarchical-timing-wheels/>

<https://blog.acolyer.org/2015/11/23/hashed-and-hierarchical-timing-wheels/>

<https://www.singchia.com/2017/11/25/An-Introduction-Of-Hierarchical-Timing-Wheels/>

<https://d2.naver.com/helloworld/267396>

<https://0x709394.me/How-To%20Design%20A%20Reliable%20Distributed%20Timer>

https://www.alibabacloud.com/blog/speeding-up-logistics-with-a-lightweight-timer-task-scheduling-engine_594197

<https://paulcavallaro.com/blog/hashed-and-hierarchical-timing-wheels/>

<http://www.cs.columbia.edu/~nahum/w6998/papers/sosp87-timing-wheels.pdf>

<https://ferbncode.github.io/Apache-Kafka-and-Request-Purgatory.html>

<https://www.youtube.com/watch?v=AftX7rqx-Uc>

<https://www.sobyte.net/post/2022-01/go-timingwheel/>

<https://developpaper.com/time-wheel-of-learning-technology-series-with-kafka/>

<https://www.slideshare.net/superniu/timing-wheels>

<https://github.com/apache/kafka/blob/trunk/core/src/main/scala/kafka/utils/timer/TimerWheel.scala>

<https://github.com/netty/netty/blob/4.1/common/src/main/java/io/netty/util/HashedWheelTimer.java>