



초대용량 멀티테넌트 시큐어 하둡 클러스터 성장통 경험기

이유비 Compute Engine

CONTENTS

1. 데이터플랫폼 C3S
2. Secure 하둡 클러스터가 커지면 무슨 일이 일어날까요?
3. 주제별로 살펴보는 문제해결
4. 클러스터 안정화, 앞으로의 방향

1. 데이터 플랫폼 C3S

1.1 C3S 가 걸어온 길

DEVIEW2020 대용량 멀티테넌트 시큐어 하둡 클러스터 운영 경험기

- <https://deview.kr/2020/sessions/394>

DEVIEW2019 대용량 멀티테넌트 시큐어 하둡 클러스터를 시행착오 없이 만들기

- <https://deview.kr/2019/schedule/323>

DEVIEW2018 C3, 데이터 처리에서 서빙까지 가능한 하둡 클러스터

- <https://deview.kr/2018/schedule/231>

DEVIEW2017 멀티테넌트 하둡 클러스터 운영 경험기

- <https://deview.kr/2017/schedule/193>

DEVIEW2016 Apache Slider 를 이용한 멀티테넌트 하둡 클러스터

- <https://deview.kr/2016/schedule#session/168>

1.2 무중단 운영의 어려움

Batch 작업 + Longlived 애플리케이션

- MapReduce, Spark, Hive on Tez ...
- HBase, Zookeeper, Kafka, Trino, Elasticsearch, Opentsdb ...
(-> YARN Service)

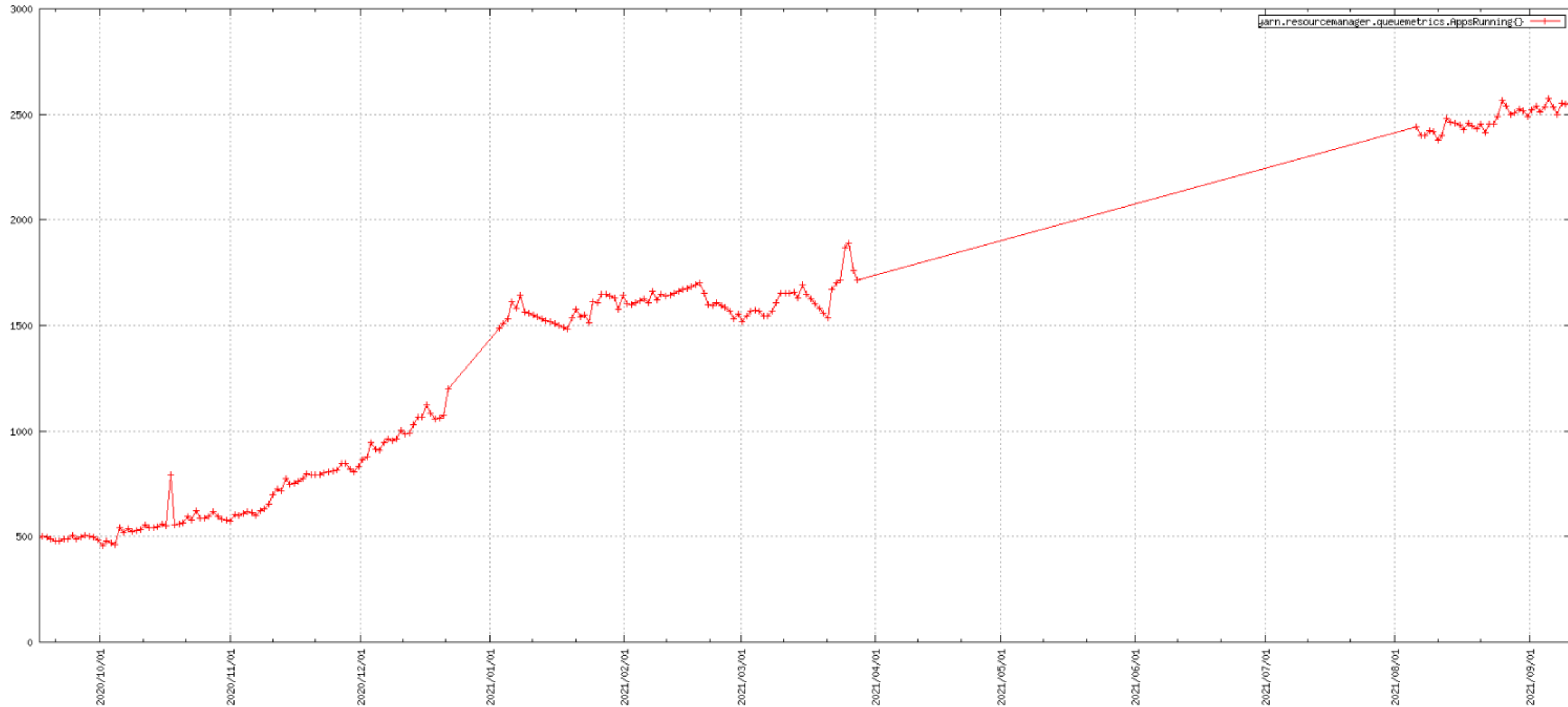
=> Batch 작업만 있는 것이 아니라 클러스터 중단은 쉽지 않다.

1.3 계속하여 성장중

- 3개의 secure 클러스터 운영, 제각기 규모 성장중
- 2개의 insecure 클러스터는 정리하며 secure 클러스터로 편입중

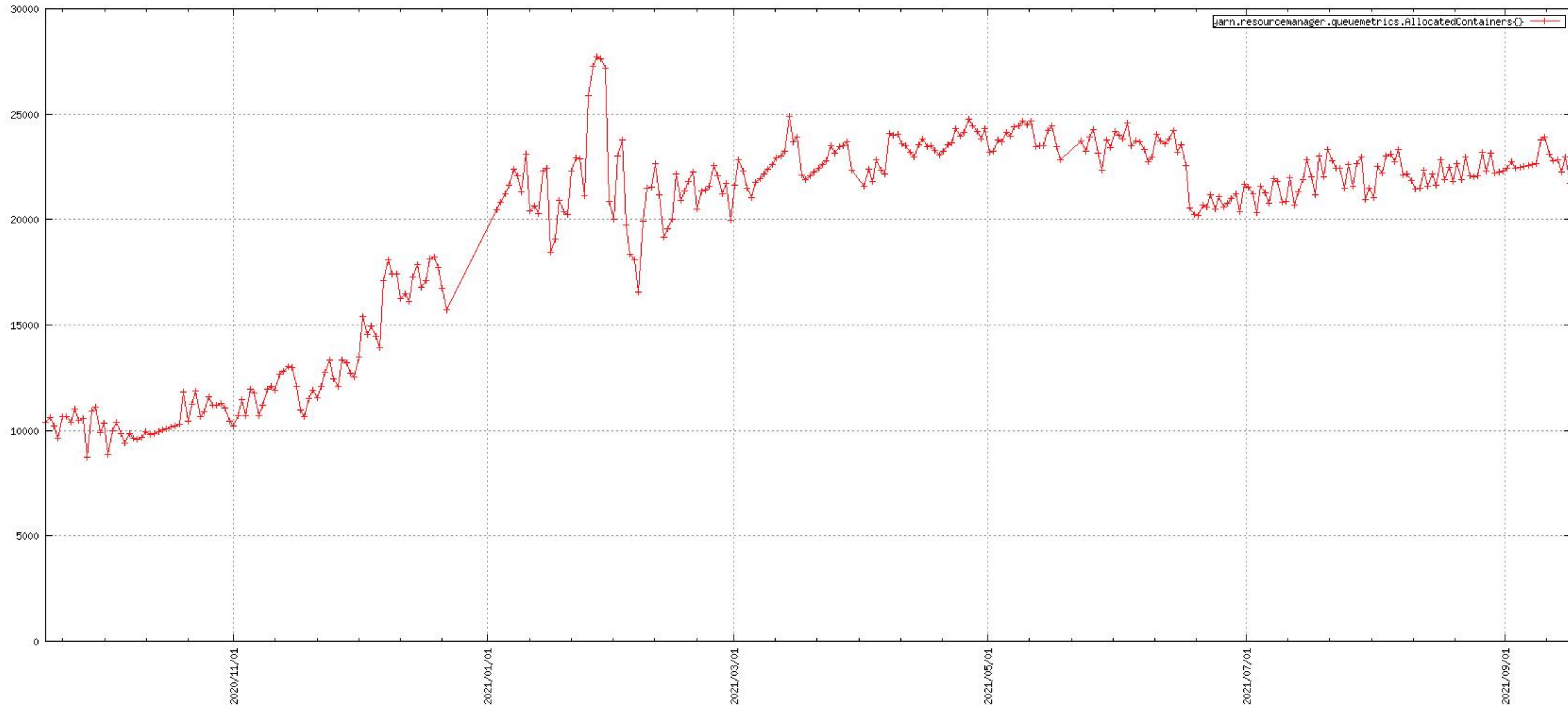
1.3 계속하여 성장중

- 일단위 최대 동시실행 작업 건수 1년 추이 (약 500 -> 2,500)



1.3 계속하여 성장중

- 일단위 최대 동시실행 컨테이너 건수 1년 추이 (약 11,000 -> 23,000)



2. Secure 하둡 클러스터가 커지면 무슨 일이 일어날까요?

2.1 LDAP

- 하둡 내의 계정 및 그룹관리를 위한 서비스
- 클러스터 장비 user, group LDAP 과 연동 중
- RM, Ranger, Ambari UI 등 LDAP 로그인 연동

LDAP high load

- 간헐적으로 NodeManager 로그에서 “User not found” 에러 발생
- Multi master LDAP 2대로 운영 중이었는데, 클러스터 규모가 커지면서 발생하기 시작
- LDAP 이 부하를 견디지 못하는 것으로 판단, 이 문제로 작업실패가 발생함

=> 간헐적으로 작업들의 실패 발생

2.2 Registry DNS Server

Registry DNS Server

- YARN Service Registry 에 기반하여 DNS 서비스를 제공하는 서버
- YARN Service Registry 정보는 모두 Zookeeper에 저장
- 노드매니저와 같은 노드에 기동하고, 해당 노드에서 발생한 모든 DNS 요청은 해당 노드에서 기동 중인 Registry DNS Server에서 처리

Zookeeper High Load

- 클러스터 장비 투입 -> Registry DNS Server 증가 -> Zookeeper 서버 부하 증가

=> Zookeeper와 연결 중인 대몬들의 불안정한 동작

2.3 ResourceManager

ResourceManager

- 클러스터의 모든 리소스를 관리하고 스케줄링을 담당
- 즉, 클러스터의 가장 중요한 대몬 중 하나

ResourceManager High Load

- 제출하는 작업량 증가
- NodeManager 증가 -> NodeManager 와 ResourceManager의 heartbeat 증가
- Heap 사용량 증가, 긴 gc 로 인한 Zookeeper session timeout

=> 작업 스케줄링 지연, 예정되지 않은 RM failover 등 여러 이슈 발생

3. 주제별로 살펴보는 문제해결

3.1 LDAP High Load

LDAP 의 높은 부하

- 2대의 multi master 로 부하를 견딜 수 있을 것이라 생각
- 클러스터 규모가 커짐에 따라 스케일 아웃으로 대응할 예정이었음
- 생각보다 이른 시기에 LDAP 부하에 따른 문제가 발견

3.1 LDAP High Load

해결방안

- LDAP 설정 최적화
- 스케일아웃

⇒ 저희가 문제를 해결한 방법인 LDAP 설정 최적화를 살펴보겠습니다.

3.1 LDAP High Load

LDAP 설정 최적화

1. sssd.conf 설정
2. 데이터베이스 인덱스 추가

3.1 LDAP High Load

sssd.conf 설정

```
[domain/LDAP]

# 86400(slave 24h) / 10800(master 3h)
entry_cache_timeout = 86400
enumerate = false
sudo_provider = none
...

[nss]

entry_cache_nowait_percentage = 80
```

/etc/sss/sssd.conf

3.1 LDAP High Load - sssd.conf

entry_cache_timeout

LDAP 에 다시 물어보지 않고 얼마나 캐싱할지

얼마의 시간 간격으로 LDAP 서버에
user, group 요청을 하고 있는지 조사

```
[domain/LDAP]
# 86400(slave 24h) / 10800(master 3h)
entry_cache_timeout = 86400
enumerate = false
sudo_provider = none
...

[nss]
entry_cache_nowait_percentage = 80
```

/etc/sss/sss.conf

3.1 LDAP High Load - sssd.conf

entry_cache_timeout

user 요청은 50, 60분에, group 요청은 60, 120분에 몰리는 현상

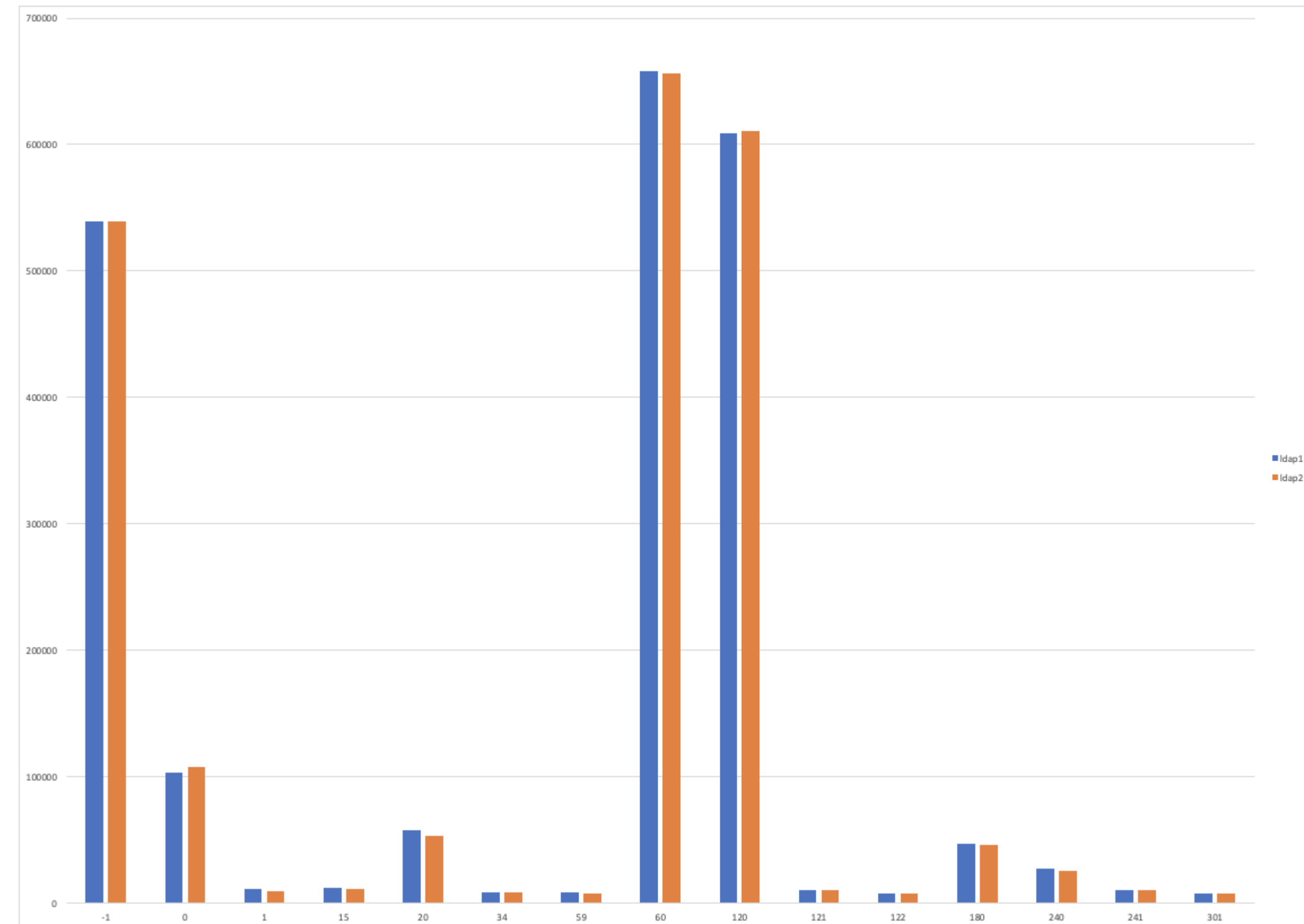
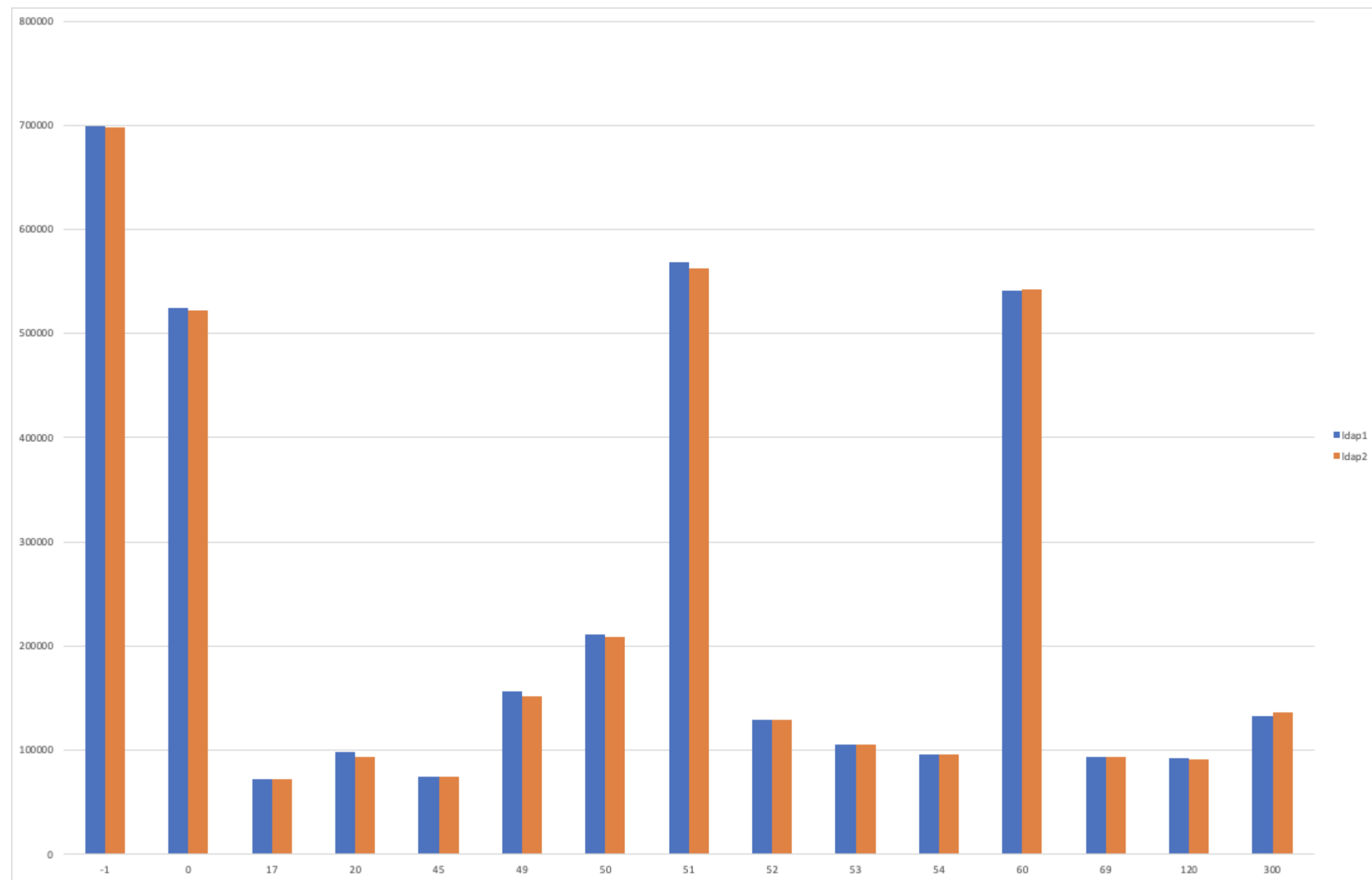
```
[domain/LDAP]
# 86400(slave 24h) / 10800(master 3h)
entry_cache_timeout = 86400
enumerate = false
sudo_provider = none
...

[nss]
entry_cache_nowait_percentage = 80
```

/etc/sss/sssd.conf

group

user



3.1 LDAP High Load - sssd.conf

entry_cache_timeout

불필요한 요청을 줄이기 위해,

slave 장비들은 86400 으로 설정하여 24시간 캐싱

master 장비들은 10800 으로 설정하여 3시간 캐싱

```
[domain/LDAP]
# 86400(slave 24h) / 10800(master 3h)
entry_cache_timeout = 86400
enumerate = false
sudo_provider = none
...

[nss]
entry_cache_nowait_percentage = 80
```

/etc/sss/sss.conf

3.1 LDAP High Load - sssd.conf

`entry_cache_nowait_percentage = 80`

`entry_cache_timeout * entry_cache_nowait_percentage / 100`

위 시간 이후 요청이 온다면 `entry_cache_timeout` 이 되기 전이라도 캐시 업데이트

=> 두 값을 적절히 설정하여 LDAP 에 요청을 많이 하지 않도록 한다.

```
[domain/LDAP]
# 86400(slave 24h) / 10800(master 3h)
entry_cache_timeout = 86400
enumerate = false
sudo_provider = none
...

[nss]
entry_cache_nowait_percentage = 80
```

/etc/sss/sss.conf

3.1 LDAP High Load - sssd.conf

enumerate = false

- 모든 사용자, 그룹 정보를 받아 로컬에 캐시하는 기능
- enumerate 수행시에도 initgroups 는 수행되지 않는다.
(= LDAP 에 직접 또 요청한다.)
- C3S 에서는 이 기능을 유의미하게 사용하지 않았다.

⇒ 꼭 필요한 기능이 아니라면 부하를 줄이기 위해 false 로 설정

```
[domain/LDAP]
# 86400(slave 24h) / 10800(master 3h)
entry_cache_timeout = 86400
enumerate = false
sudo_provider = none
...

[nss]
entry_cache_nowait_percentage = 80
```

/etc/sss/sss.conf

3.1 LDAP High Load - sssd.conf

sudo_provider = none

ldap_sudo_full_refresh_interval, ldap_sudo_smart_refresh_interval
두 설정에 따라 증분, 전체 검색

=> sudo 를 ldap 으로 관리하지 않는 곳에선 불필요하므로 끄기

```
[domain/LDAP]
# 86400(slave 24h) / 10800(master 3h)
entry_cache_timeout = 86400
enumerate = false
sudo_provider = none
...

[nss]
entry_cache_nowait_percentage = 80
```

/etc/sss/sss.conf

3.1 LDAP High Load – 인덱스 추가

데이터베이스 인덱스 추가

찾은 요청이 오는 항목에 대해 인덱스를 추가

아래 LDAP 로그를 보고 uid, memberUid 에 대해서 먼저 추가, 테스트

```
<= bdb_equality_candidates: (uid) not indexed  
<= bdb_equality_candidates: (memberUid) not indexed
```


3.1 LDAP High Load – 인덱스 추가

데이터베이스 인덱스 추가

LDAP 계정으로 진행하면 LDAP 다운타임 없이 index 추가가 가능하다.



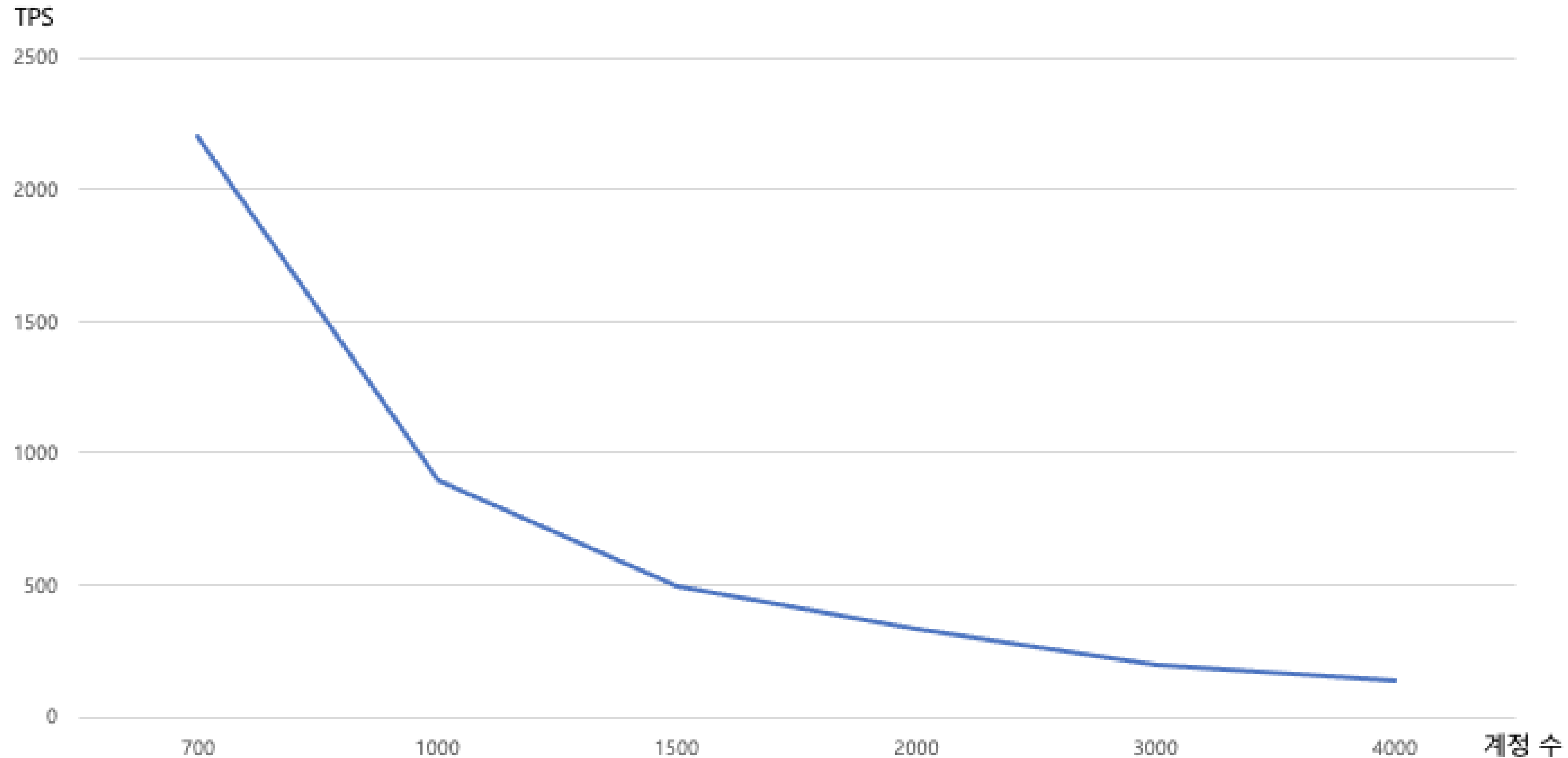
```
$ cat index.ldif
dn: olcDatabase={2}hdb,cn=config
changetype: modify
add: olcDbIndex
olcDbIndex: uid, memberUid eq

# ldap 계정으로 로그인 후, index 추가
$ /usr/bin/ldapmodify -D "cn=config" -H ldaps://<ldap_server>:636 -W -f ./index.ldif
```

3.1 LDAP High Load – 인덱스 추가

uid 인덱스 추가하기 전

계정 수에 따라 TPS 가 다르다.

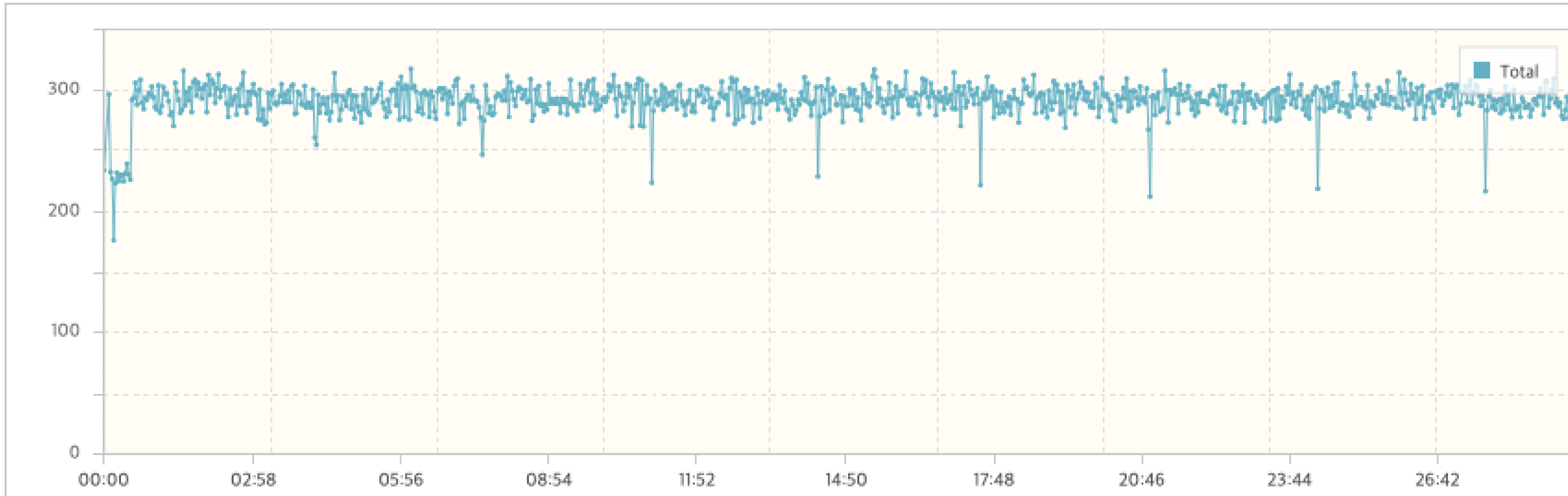


3.1 LDAP High Load – 인덱스 추가

uid 인덱스 추가하기 전

약 300 TPS (당시 C3S 계정 수 상황)

TPS ⓘ

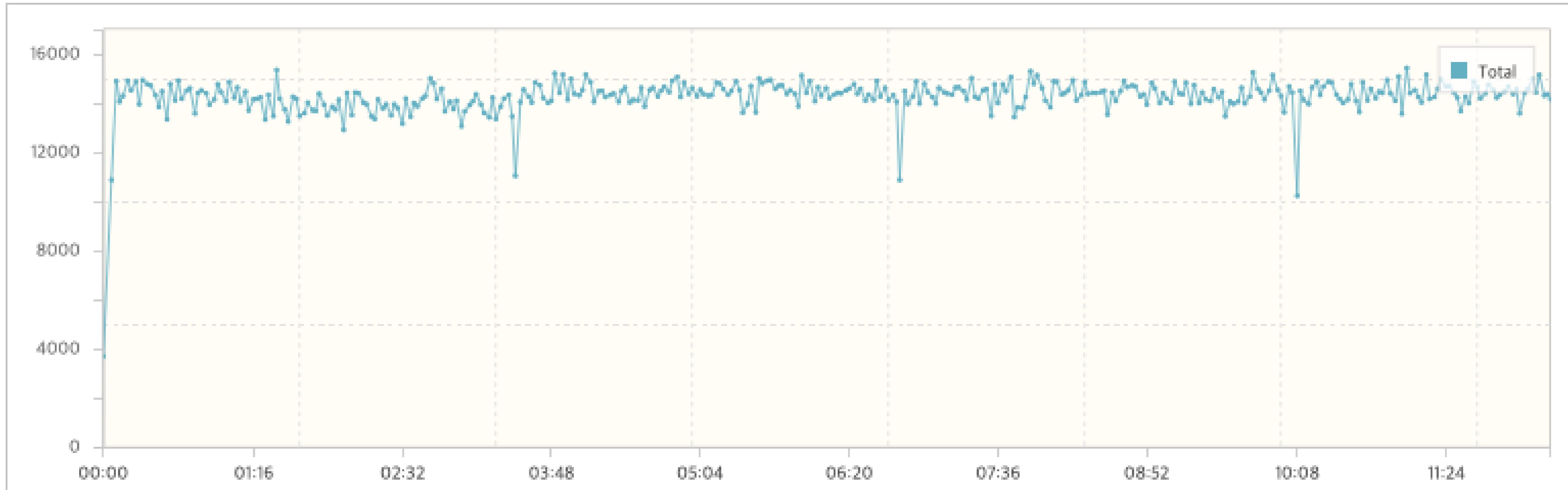


3.1 LDAP High Load – 인덱스 추가

uid 인덱스 추가 후

약 14,000 TPS (계정 개수와 상관 없이 일정)

TPS ⓘ



3.1 LDAP High Load – 인덱스 추가

다른 항목에 대해서도 인덱스 추가

uidNumber 쿼리에 대해서 부하 발생을 경험

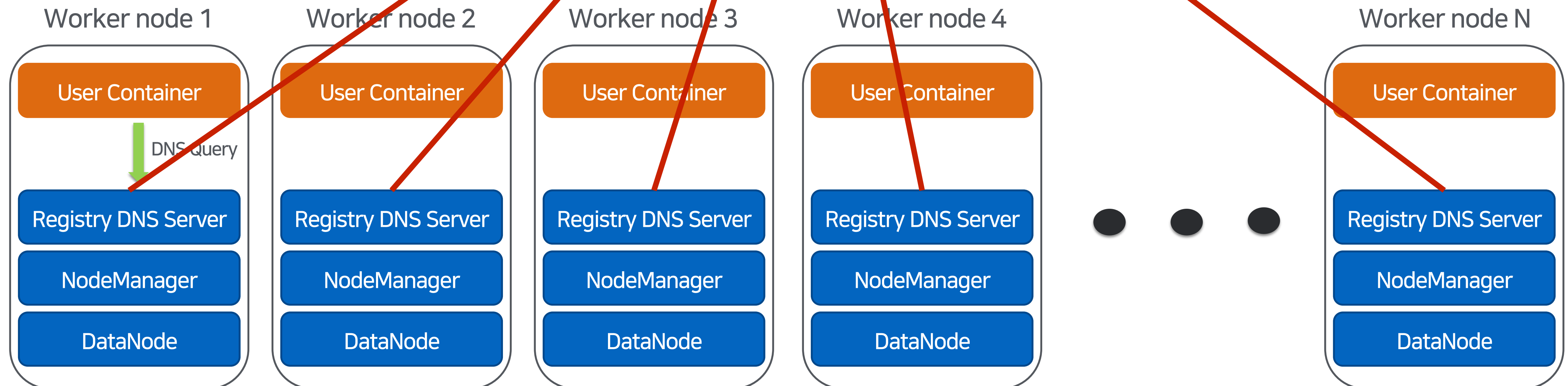
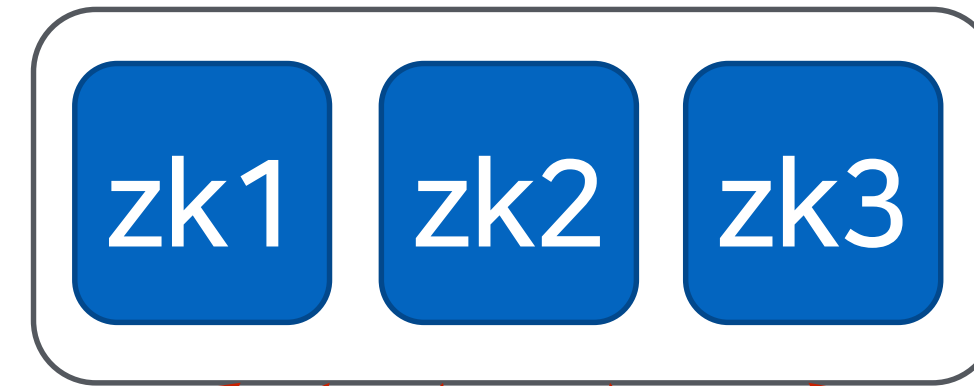
부하가 생길 수 있는 다른 항목에 대해서도 미리 인덱스를 만들기

```
$ cat index.ldif
dn: olcDatabase={2}hdb,cn=config
changetype: modify
add: olcDbIndex
olcDbIndex: uid eq
olcDbIndex: memberUid eq
olcDbIndex: uidNumber eq
olcDbIndex: gidNumber eq
olcDbIndex: description eq
```

3.2 Registry DNS Server

초기 구조

처음엔 DNS 서버 부하만 고려
장비대수 ↑ → 주키퍼 부하 ↑



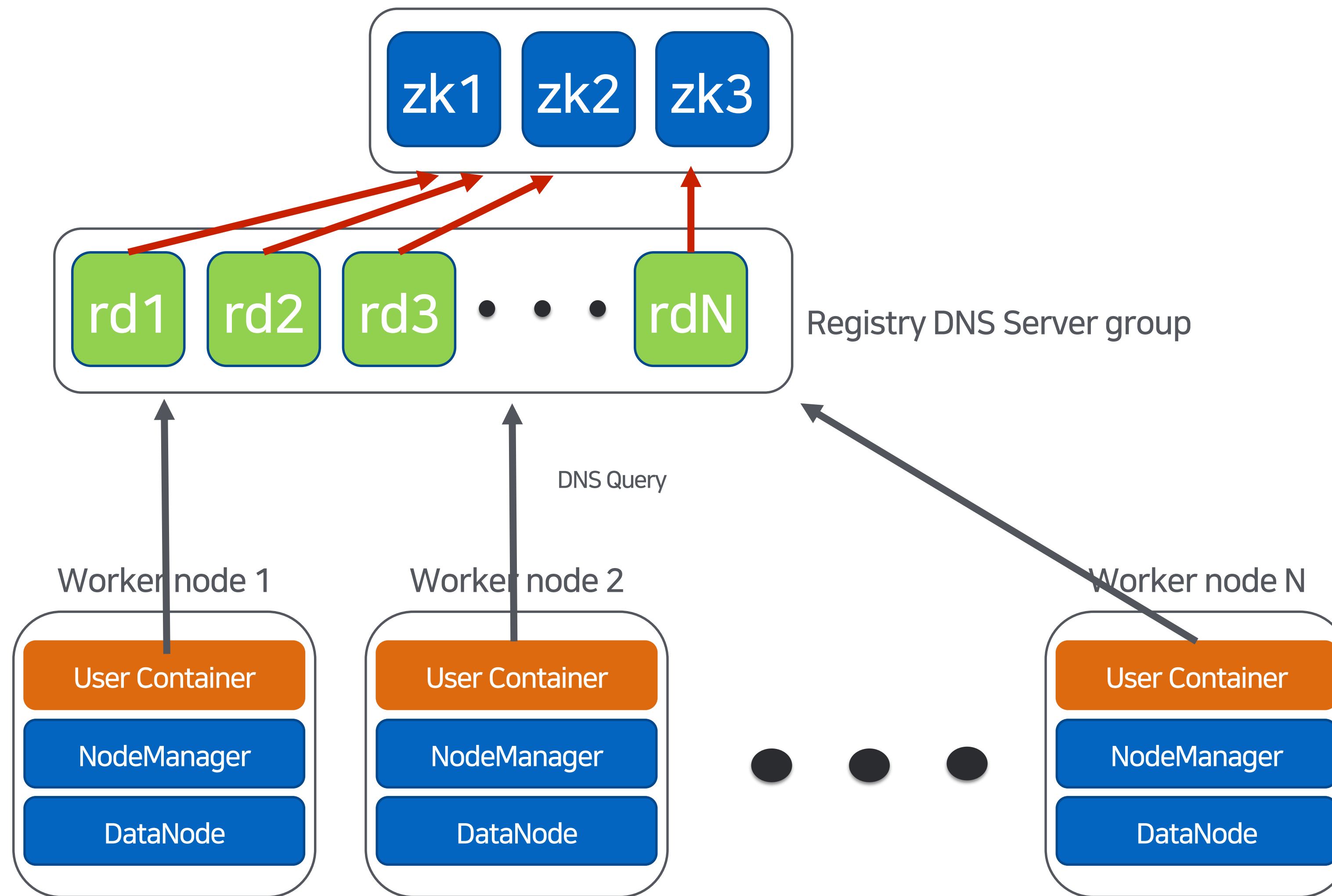
3.2 Registry DNS Server

현재 구조

장비대수 ↑

→ DNS 서버 그룹 부하 ↑
(일정 수 유지)

→ 주키퍼 부하 일정



rd : Registry DNS Server

3.2 Registry DNS Server

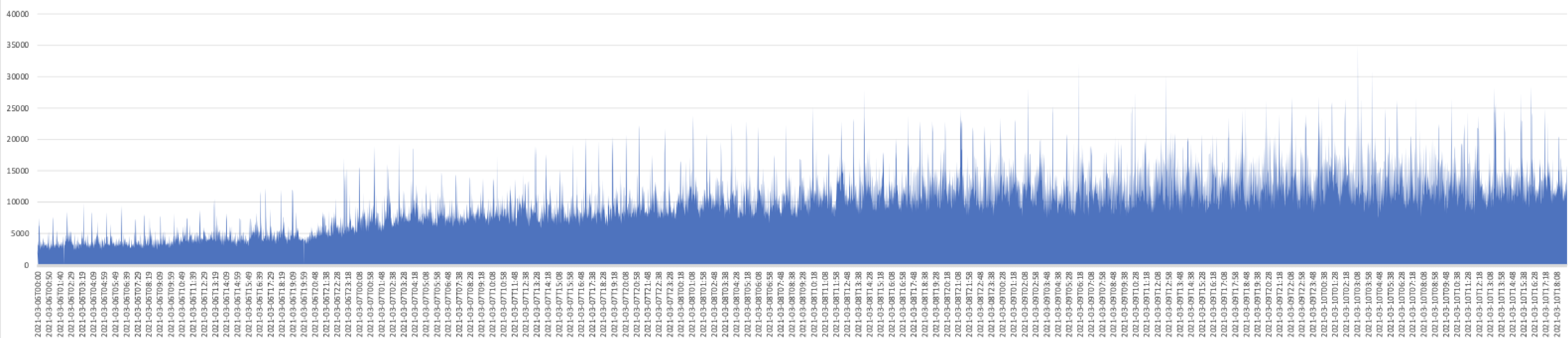
분산되던 요청량을 버틸 수 있을까?

클러스터 총 요청량

2021 3/1 ~ 3/10 조사

peak 시 udp 약 34,000 qps (tcp 약 1,000 qps)

2021 3/1~3/10 YAN Registry DNS udp qps



3.2 Registry DNS Server

DNS 서버 한대당 처리량은?

Hadoop 3.1.2, UDP 기준 5,000 qps

⇒ 34,000 qps 를 처리하기 위해 6대 이상이 필요함

클러스터 규모에 비해 6대면 적긴 하지만... DNS 서버 역할만 쓰기 아깝다.
원인을 알 수 없이 종종 DNS 요청 처리를 못하는 불안정성도 존재한다.
성능도 안나오는 것 같은데... 개선해볼까?

3.2 Registry DNS Server

성능에 영향을 끼치는 로그 출력 부분 수정

- 코드 수정 없이 log4j.properties 설정으로도 가능
- 5,000 -> 12,000 qps (UDP)



```
} else if (sr.isSuccessful()) {  
    List<RRset> rrsets = sr.answers();  
-    LOG.info("found answers {}", rrsets);  
+    LOG.debug("found answers {}", rrsets);  
    for (RRset rrset : rrsets) {  
        addRRset(name, response, rrset, Section.ANSWER, flags);  
    }  
    addNS(response, zone, flags);  
}
```

3.2 Registry DNS Server

TCP 처리시 불필요한 sleep 제거

- non-blocking -> blocking
- tcp 응답 속도 개선

```
private ServerSocketChannel openTCPChannel(InetAddress addr, int port)
    throws IOException {
    ServerSocketChannel serverSocketChannel = ServerSocketChannel.open()
    try {
        serverSocketChannel.socket().bind(new InetSocketAddress(addr, port)
-       serverSocketChannel.configureBlocking(false);
+       serverSocketChannel.configureBlocking(true);
    }
}
```

[RegistryDNS.java](#)

```
public void serveNIOTCP(ServerSocketChannel serverSocketChannel,
...
    final SocketChannel socketChannel = serverSocketChannel.accept();
    if (socketChannel != null) {
        executor.submit(new Callable<Boolean>() {
            @Override
            public Boolean call() throws Exception {
                nioTCPClient(socketChannel);
                return true;
            }
        });
    } else {
        Thread.sleep(500);
    }
}
```

3.2 Registry DNS Server

그 외 개선점

- UDP 처리 중 에러 발생시 스레드가 죽어 처리가 불가능한 상황 해결
(DNS 서버 프로세스는 살아있음)
- 요청 수에 따라 스레드 수가 늘어나는 `newCachedThreadPool` 대신 `newFixedThreadPool` 사용
 - => `newCachedThreadPool` 은 과도한 요청시 비정상 동작
 - => 처리량 및 안정성 개선
- UDP 로 응답 가능한 사이즈보다 큰 경우 TCP 로 처리하도록 개선
 - => DNS 서버 connection timed out 문제 해결

3.2 Registry DNS Server

요청 처리량 개선

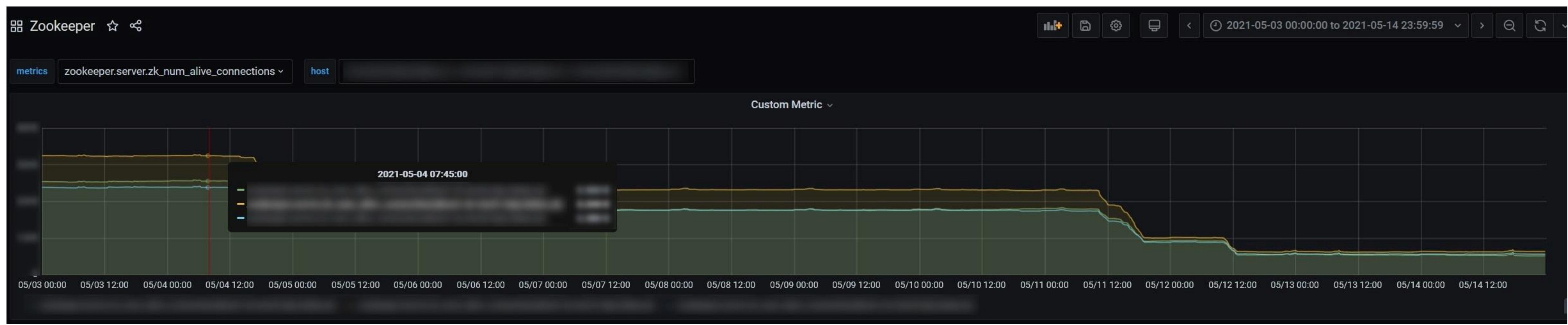
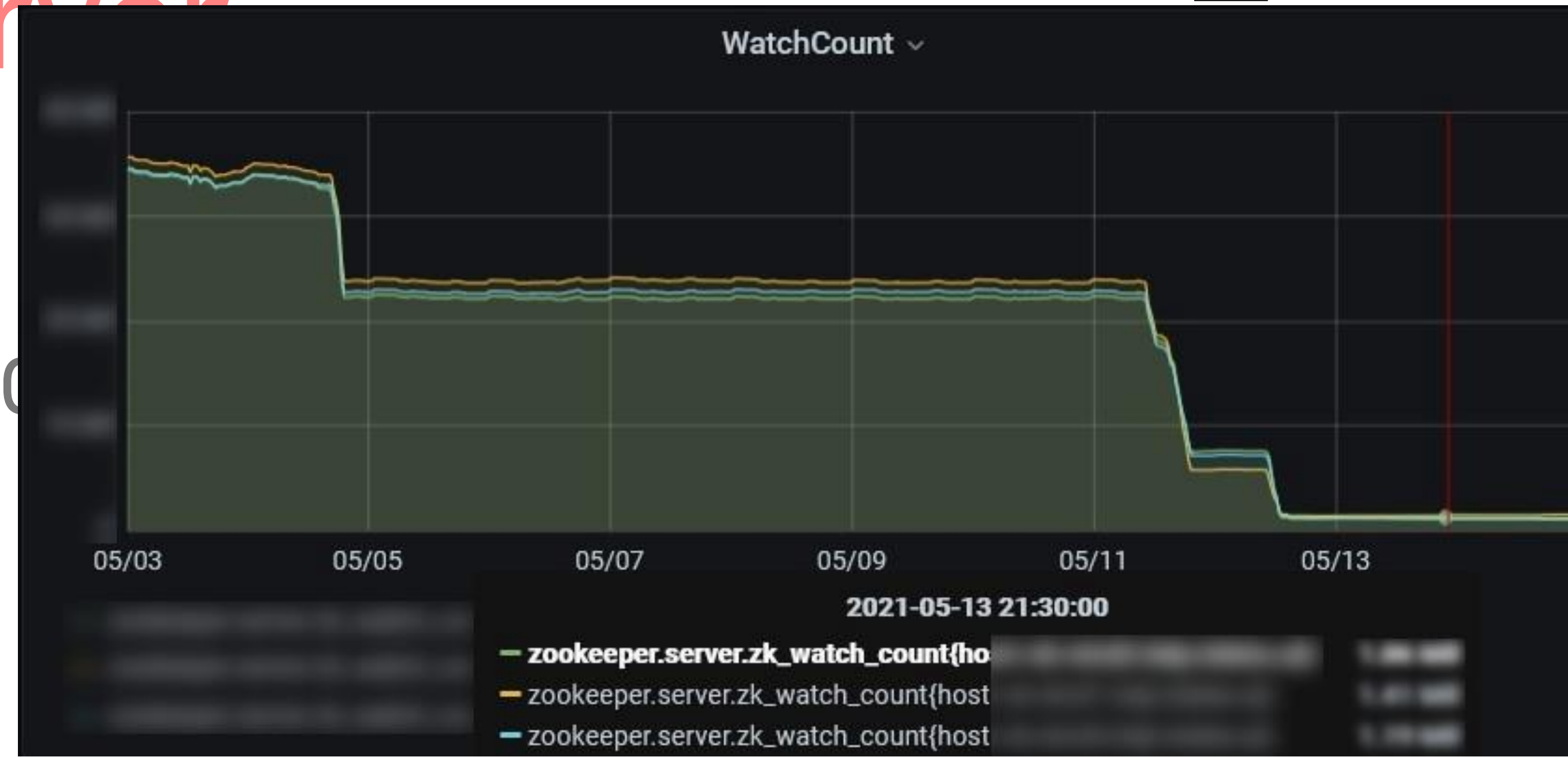
- 1대당 5,000 qps -> 40,000 ~ 60,000 qps (UDP)

3.2 Registry DNS Server

요청 처리량 개선

- 1대당 5,000 qps -> 40,000 ~ 60,000

주키퍼 부하 감소



3.2 Registry DNS Server

Contribution

- DNS Query on DNS Registry got connection timed out at times.
<https://issues.apache.org/jira/browse/HADOOP-17859>
- improve YARN Registry DNS Server qps
<https://issues.apache.org/jira/browse/HADOOP-17861>

3.3 Resource Manager

클러스터 규모가 커지며 대두된 이슈들 (Hadoop 3.1.2)

- RM failover 후 컨테이너 할당 정체
- refreshQueues 수행 후 Deadlock
- RM Memory leak
- non-exclusive 자원 제어
- 자원 preemption 실패
- YARN Service AM 문제
- ...

3.3.1 RM failover 후 컨테이너 할당 정체

RM JMX 를 살펴보던 중...

- 특정 큐 할당 대기 중인 Pending 자원이 음수로 나오는 현상을 발견

```
{
  "beans": [
    ...
    {
      "name" : "Hadoop:service=ResourceManager,name=QueueMetrics,q0=root,q1=batch",
      "modelerType" : "QueueMetrics,q0=root,q1=batch",
      "tag.Queue" : "root.batch",
      ...
      "PendingMB" : -835584,
      "PendingVCores" : -354,
      "PendingContainers" : -104,
      ...
    }
  ]
}
```

3.3.1 RM failover 후 컨테이너 할당 정체

원인을 찾아보자.

- <https://issues.apache.org/jira/browse/YARN-8984> 참고
- <https://issues.apache.org/jira/browse/YARN-9195> 참고
- RM failover 후 수행되는 AMRMClientImpl#registerApplicationMaster
부분을 따라가 보자.
 - > AMRMClientImpl#removeFromOutstandingSchedulingRequests

3.3.1 RM failover

음수 요청을 해버린다.

```
private void removeFromOutstandingSchedulingRequests(
    Collection<Container> containers) {
    ...
    for (Container container : containers) {
        if (container.getAllocationTags() != null &&
            !container.getAllocationTags().isEmpty()) {
            List<SchedulingRequest> schedReqs =
                this.outstandingSchedRequests.get(container.getAllocationTags());
            if (schedReqs != null && !schedReqs.isEmpty()) {
                synchronized (schedReqs) {
                    Iterator<SchedulingRequest> iter = schedReqs.iterator();
                    while (iter.hasNext()) {
                        SchedulingRequest schedReq = iter.next();
                        if (schedReq.getPriority().equals(container.getPriority()) &&
                            schedReq.getAllocationRequestId() ==
                                container.getAllocationRequestId()) {
                            int numAllocations =
                                schedReq.getResourceSizing().getNumAllocations();
                            numAllocations--;
                            if (numAllocations == 0) {
                                iter.remove();
                            } else {
                                schedReq.getResourceSizing()
                                    .setNumAllocations(numAllocations);
                            }
                        }
                    }
                }
            }
        }
    }
    ...
}
```

3.3.1 RM failover 프로세스

음수 요청을 해버린다.

outstanding

=> 더 필요한 컨테이너 요청

```
private void removeFromOutstandingSchedulingRequests(
    Collection<Container> containers) {
    ...
    for (Container container : containers) {
        if (container.getAllocationTags() != null &&
            !container.getAllocationTags().isEmpty()) {
            List<SchedulingRequest> schedReqs =
                this.outstandingSchedRequests.get(container.getAllocationTags());
            if (schedReqs != null && !schedReqs.isEmpty()) {
                synchronized (schedReqs) {
                    Iterator<SchedulingRequest> iter = schedReqs.iterator();
                    while (iter.hasNext()) {
                        SchedulingRequest schedReq = iter.next();
                        if (schedReq.getPriority().equals(container.getPriority()) &&
                            schedReq.getAllocationRequestId() ==
                                container.getAllocationRequestId()) {
                            int numAllocations =
                                schedReq.getResourceSizing().getNumAllocations();
                            numAllocations--;
                            if (numAllocations == 0) {
                                iter.remove();
                            } else {
                                schedReq.getResourceSizing()
                                    .setNumAllocations(numAllocations);
                            }
                        }
                    }
                }
            }
        }
    }
}
```

3.3.1 RM failover 프

음수 요청을 해버린다.

outstanding

⇒ 더 필요한 컨테이너 요청

outstanding 요청 중

이미 할당된 컨테이너 차감

```
private void removeFromOutstandingSchedulingRequests(
    Collection<Container> containers) {
    ...
    for (Container container : containers) {
        if (container.getAllocationTags() != null &&
            !container.getAllocationTags().isEmpty()) {
            List<SchedulingRequest> schedReqs =
                this.outstandingSchedRequests.get(container.getAllocationTags());
            if (schedReqs != null && !schedReqs.isEmpty()) {
                synchronized (schedReqs) {
                    Iterator<SchedulingRequest> iter = schedReqs.iterator();
                    while (iter.hasNext()) {
                        SchedulingRequest schedReq = iter.next();
                        if (schedReq.getPriority().equals(container.getPriority()) &&
                            schedReq.getAllocationRequestId() ==
                                container.getAllocationRequestId()) {
                            int numAllocations =
                                schedReq.getResourceSizing().getNumAllocations();
                            numAllocations--;
                            if (numAllocations == 0) {
                                iter.remove();
                            } else {
                                schedReq.getResourceSizing()
                                    .setNumAllocations(numAllocations);
                            }
                        }
                    }
                }
            }
        }
    }
}
```

3.3.1 RM failover 후

음수 요청을 해버린다.

failover 후 outstanding 요청이 사라졌을 때,
이미 할당된 컨테이너 수가 더 많아
음수가 되어버릴 가능성이 있다.

⇒ 이런 경우 더 이상 해당 큐에
스케줄링이 안되는 현상 확인

```
private void removeFromOutstandingSchedulingRequests(
    Collection<Container> containers) {
    ...
    for (Container container : containers) {
        if (container.getAllocationTags() != null &&
            !container.getAllocationTags().isEmpty()) {
            List<SchedulingRequest> schedReqs =
                this.outstandingSchedRequests.get(container.getAllocationTags());
            if (schedReqs != null && !schedReqs.isEmpty()) {
                synchronized (schedReqs) {
                    Iterator<SchedulingRequest> iter = schedReqs.iterator();
                    while (iter.hasNext()) {
                        SchedulingRequest schedReq = iter.next();
                        if (schedReq.getPriority().equals(container.getPriority()) &&
                            schedReq.getAllocationRequestId() ==
                                container.getAllocationRequestId()) {
                            int numAllocations =
                                schedReq.getResourceSizing().getNumAllocations();
                            numAllocations--;
                            if (numAllocations == 0) {
                                iter.remove();
                            } else {
                                schedReq.getResourceSizing()
                                    .setNumAllocations(numAllocations);
                            }
                        }
                    }
                }
            }
        }
    }
}
```

3.3.1 RM failover 후 컨테이너 할당 정체

RM failover 후 컨테이너 할당이 막히는 부분만 해결

- Failover 후 Outstanding 요청을 정확히 복원 못함.
- 근본적인 수정은 아니기에 Contribution 은 하지 못함.

3.3.1 RM failover 후 컨테이너 할당 정책

고친 부분은 단 2곳

- AMRMClientImpl#removeFromOutstandingSchedulingRequests
-> AM 이 음수 요청을 하지 않도록 한다.

3.3.1 RM failover 후 컨테이너 할당 정책

고친 부분은 단 2곳

- AMRM
- > AM

```
private void removeFromOutstandingSchedulingRequests(
    Collection<Container> containers) {
    ...
    int numAllocations =
        schedReq.getResourceSizing().getNumAllocations();
    numAllocations--;
-   if (numAllocations == 0) {
+   if (numAllocations <= 0) {
+       if (numAllocations < 0) {
+           LOG.info("NumAllocations cannot be negative : " + numAllocations);
+       }
        iter.remove();
    } else {
        schedReq.getResourceSizing()
            .setNumAllocations(numAllocations);
    }
}
```

3.3.1 RM failover 후 컨테이너 할당 정책

고친 부분은 단 2곳

- AMRMClientImpl#removeFromOutstandingSchedulingRequests
-> AM 이 음수 요청을 하지 않도록 한다.
- SingleConstraintAppPlacementAllocator#validateAndSetSchedulingRequest
-> RM 에 음수 요청이 오더라도 무시한다.

3.3.1 RM failover 후 컨테이너 할당 정책

고친 부분은 단 2곳

- AMRMClient
- > AM 이
- SingleContainer
- > RM 에

```

private void validateAndSetSchedulingRequest(SchedulingRequest
    newSchedulingRequest)
    throws SchedulerInvalidResourceRequestException {
    // Check sizing exists
    if (newSchedulingRequest.getResourceSizing() == null
        ...
    }

+ // Check allocations >= 0
+ if (newSchedulingRequest.getResourceSizing().getNumAllocations() < 0) {
+     throwExceptionWithMetaInfo(
+         "numAllocation in ResourceSizing field must be >= 0, "
+         + "updating schedulingRequest failed.");
+ }

// Check execution type == GUARANTEED
...

```

requests
SchedulingRequest

3.3.1 RM failover 후 컨테이너 할당 정체

고칠 시간이 없어요!

RM 에 패치를 적용할 시간이 없을 때 임시 해결방법?

- 음수가 되어버린 yarn service 앱의 특정 컨테이너 개수를 임시로 아주 많이 늘려서 outstanding 이 양수로 바뀔 수 있도록 한다.
- 해당 큐에 정상적으로 컨테이너가 할당되기 시작하면 임시로 늘렸던 앱의 컨테이너 개수를 원상 복구 한다.

3.3.2 refreshQueues 수행 후 Deadlock

큐 설정 변경

- yarn rmdadmin -refreshQueues
- Deadlock 발생시 컨테이너 할당 막힘
- 임시 방편으로 Active RM failover 필요
- 앞서 "RM failover 후 컨테이너 할당 정체" 가 해결되지 않았다면 연달아 문제 발생

3.3.2 refreshQueues 수행 후 Deadlock



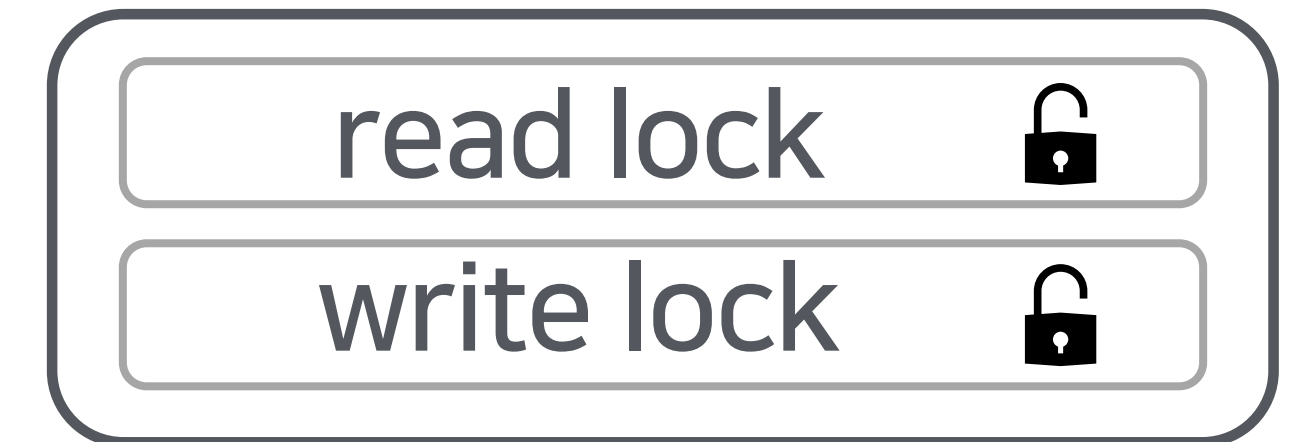
AsyncScheduleThread (worker 노드에 컨테이너를 할당하는 쓰레드)



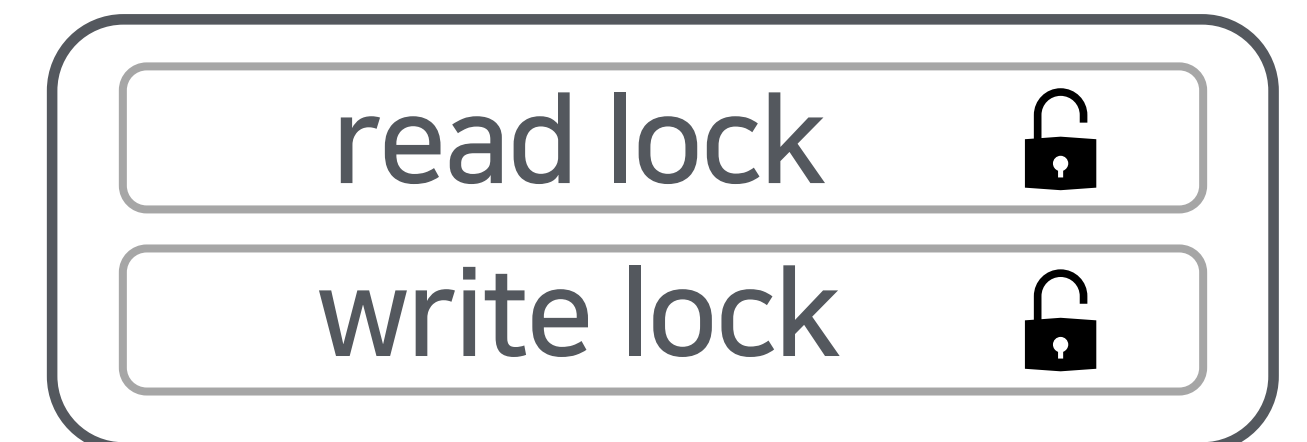
RefreshQueue thread (refreshQueue 명령어 처리하는 쓰레드)
(IPC - yarn.resourcemanager.admin.address 8033)



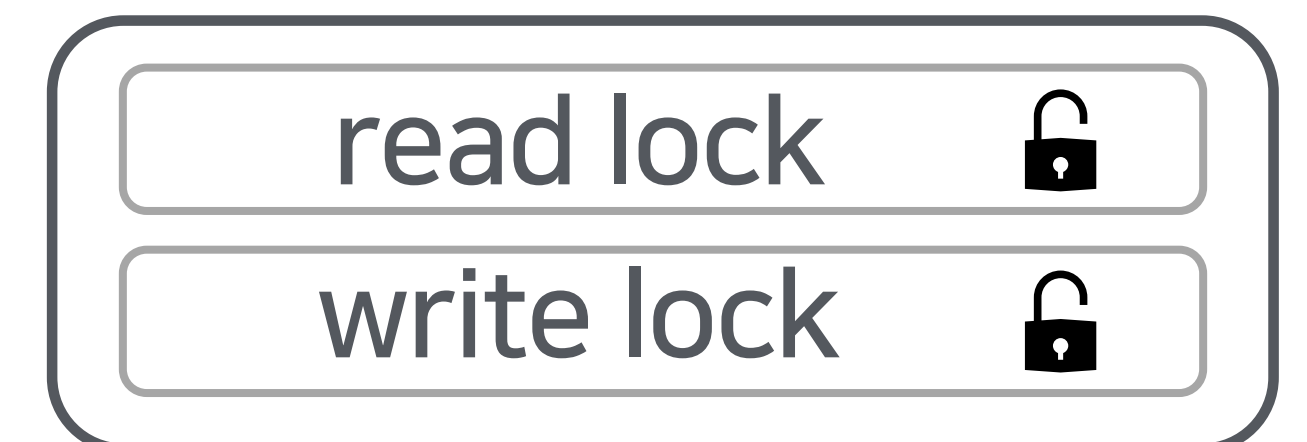
Scheduler interface thread (AM 자원 할당 / 해제 요청 처리)
(IPC - yarn.resourcemanager.scheduler.address 8030)



ParentQueue

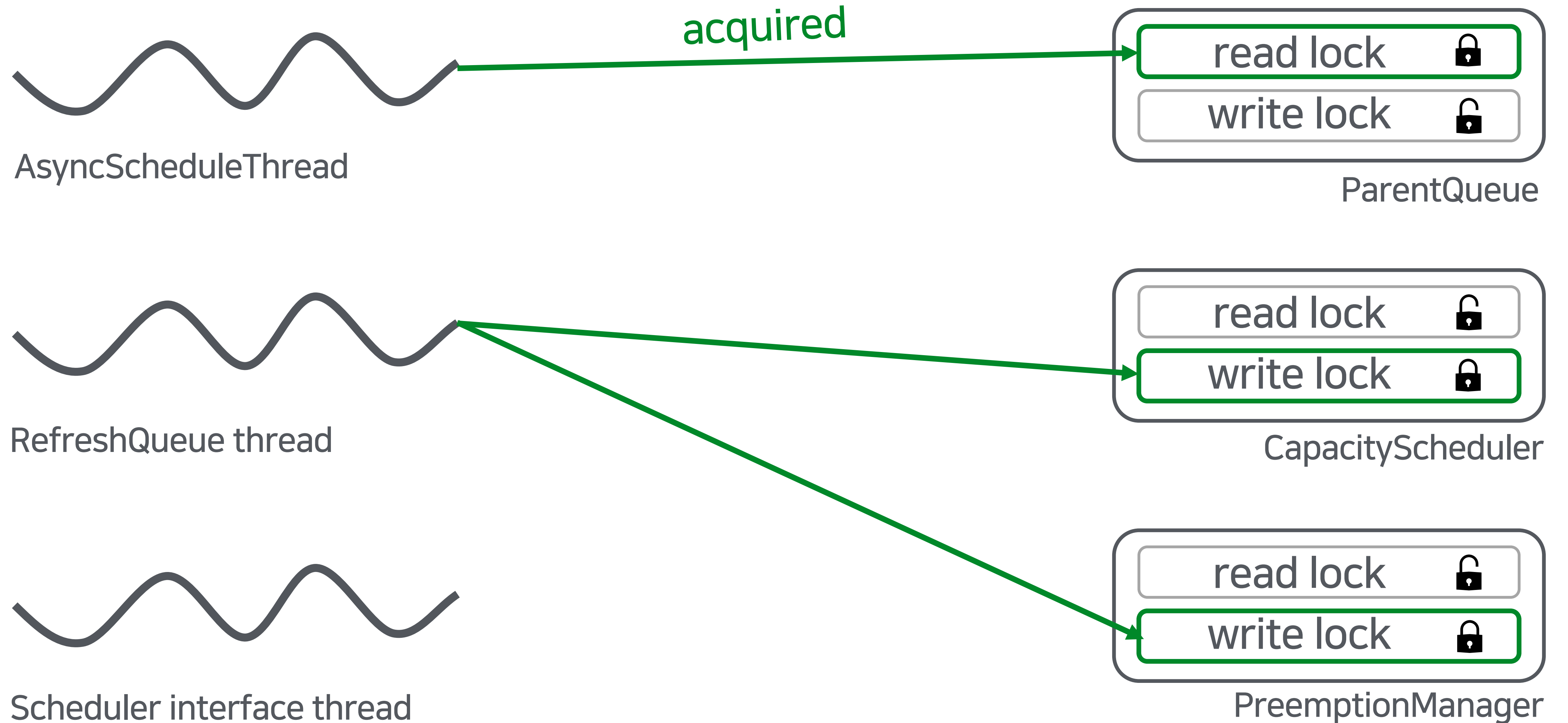


CapacityScheduler



PreemptionManager

3.3.2 refreshQueues 수행 후 Deadlock



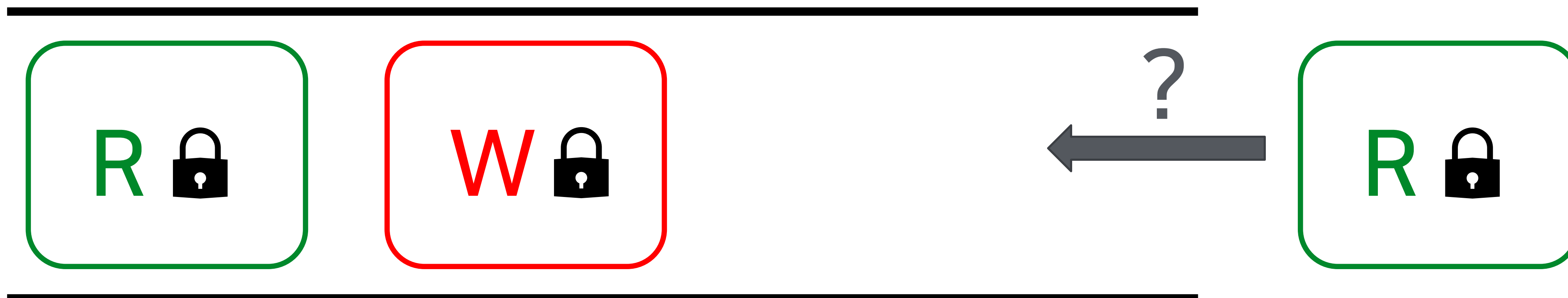
3.3.2 refreshQueues 수행 후 Deadlock

ReentrantReadWrite lock

- Read Preference
- Write Preference

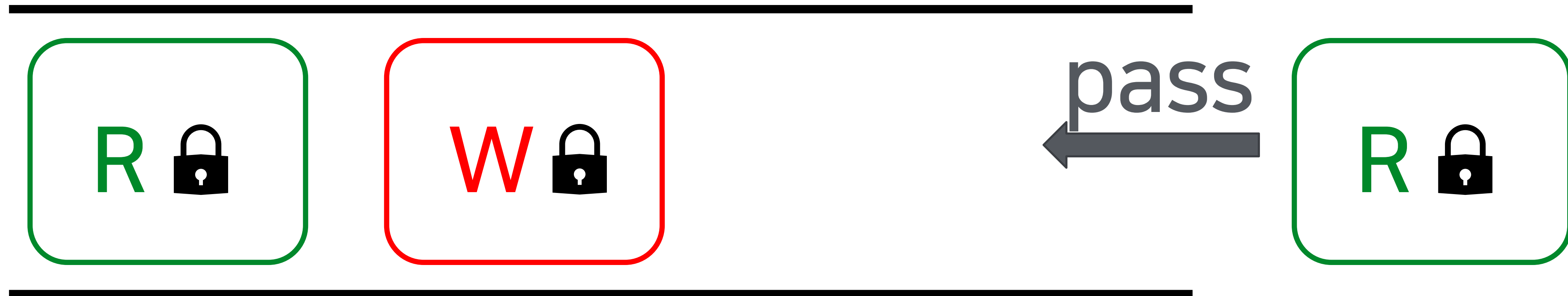
3.3.2 refreshQueues 수행 후 Deadlock

ReentrantReadWrite lock – Read Preference



3.3.2 refreshQueues 수행 후 Deadlock

ReentrantReadWrite lock – Read Preference



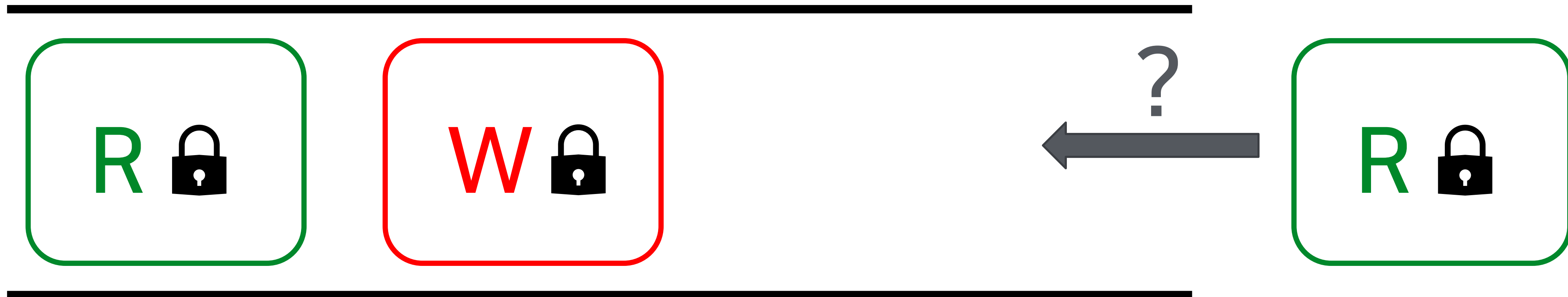
3.3.2 refreshQueues 수행 후 Deadlock

ReentrantReadWrite lock – Read Preference



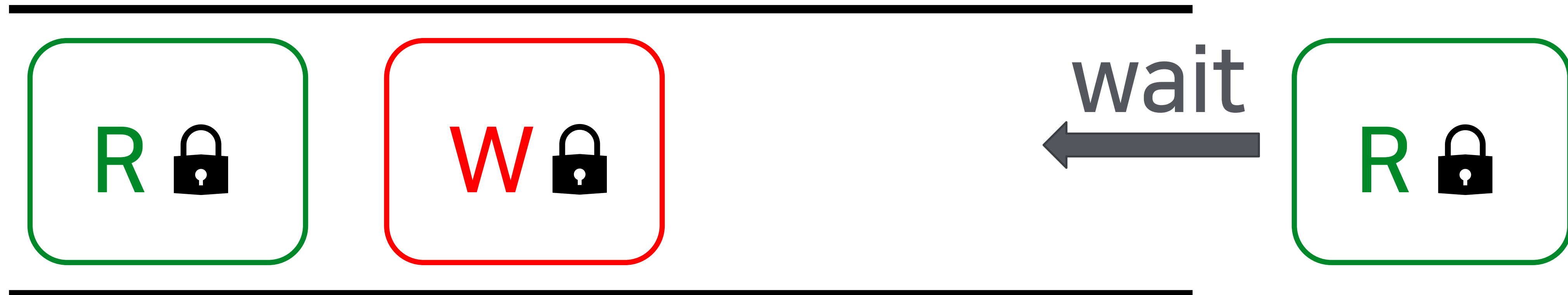
3.3.2 refreshQueues 수행 후 Deadlock

ReentrantReadWrite lock – Write Preference



3.3.2 refreshQueues 수행 후 Deadlock

ReentrantReadWrite lock – Write Preference



3.3.2 refreshQueues 수행 후 Deadlock

ReentrantReadWrite lock – Write Preference

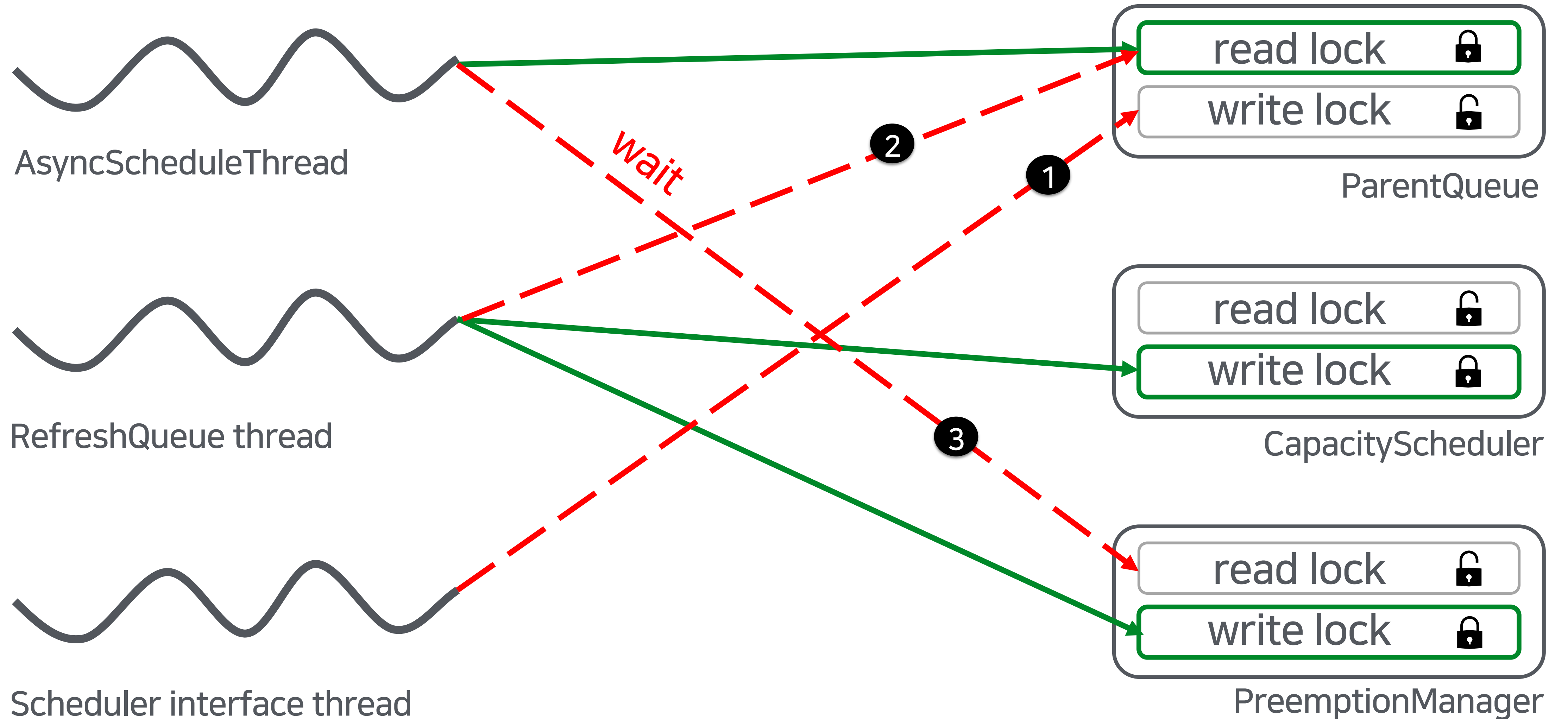


3.3.2 refreshQueues 수행 후 Deadlock

ReentrantReadWrite lock – Write Preference

- 직접 테스트를 해보았을 때, C3S 클러스터 ResourceManager 에서는 Write Preference 를 따름

3.3.2 refreshQueues 수행 후 Deadlock

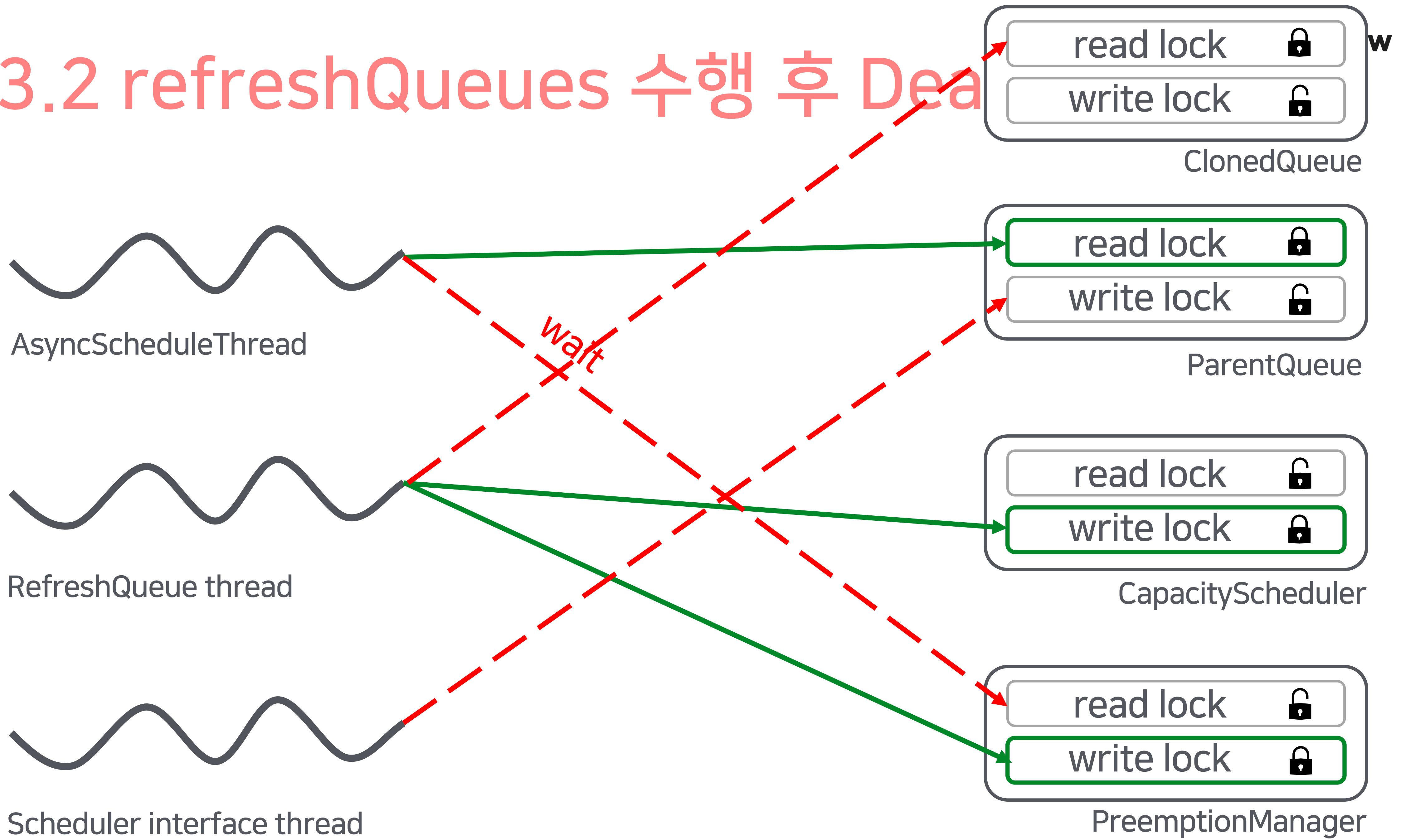


3.3.2 refreshQueues 수행 후 Deadlock

YARN-9163 적용

- ParentQueue 를 clone 하여 lock 을 회피

3.3.2 refreshQueues 수행 후 Dea



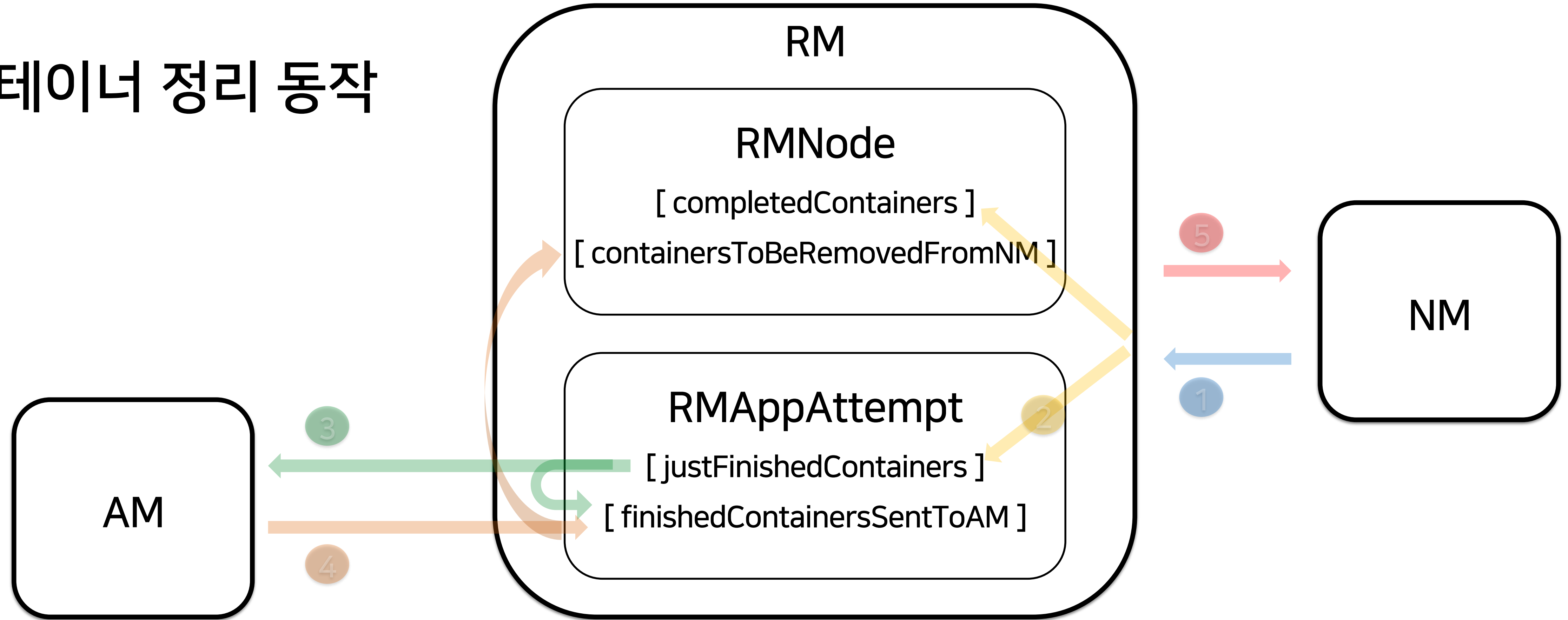
3.3.2 refreshQueues 수행 후 Deadlock

이것도 고칠 시간이 없어요!

- refresh queue 가 몇초내로 반응을 안할 때, 빠르게 failover 로 임시 해결

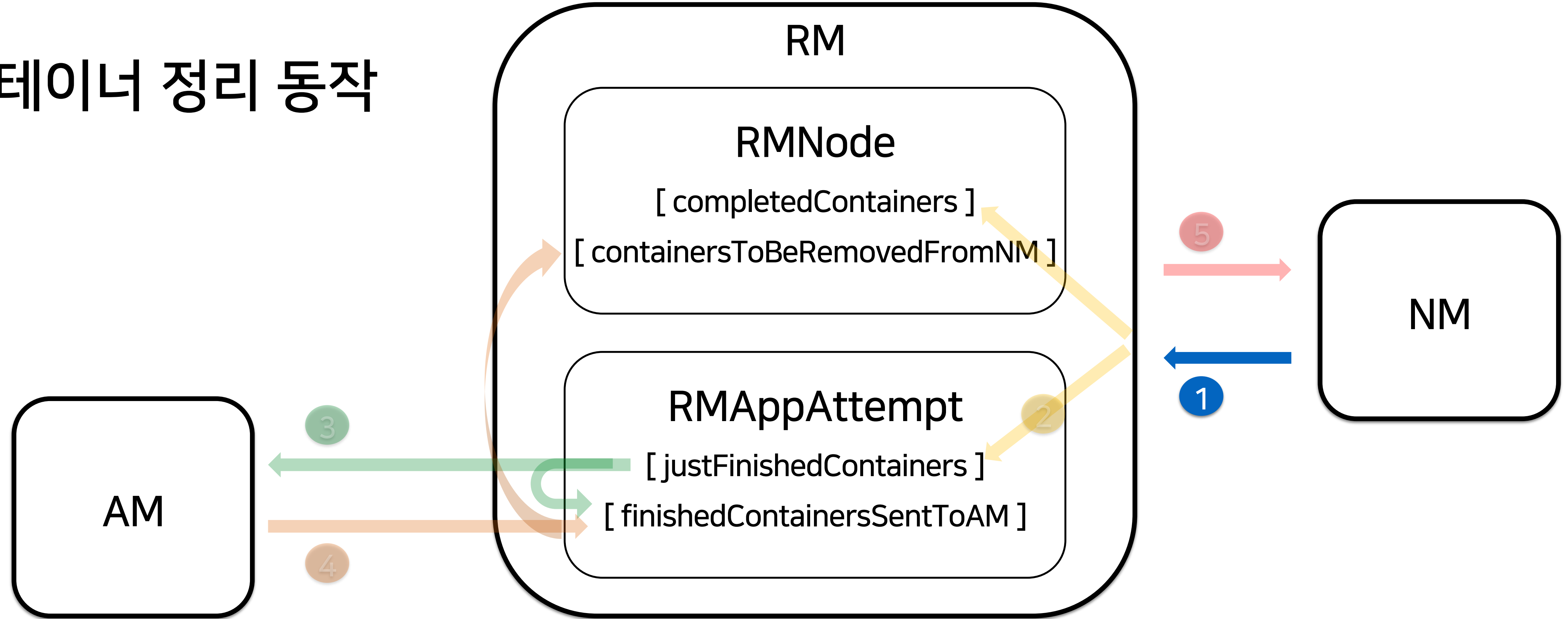
3.3.3 RM memory leak

컨테이너 정리 동작



3.3.3 RM memory leak

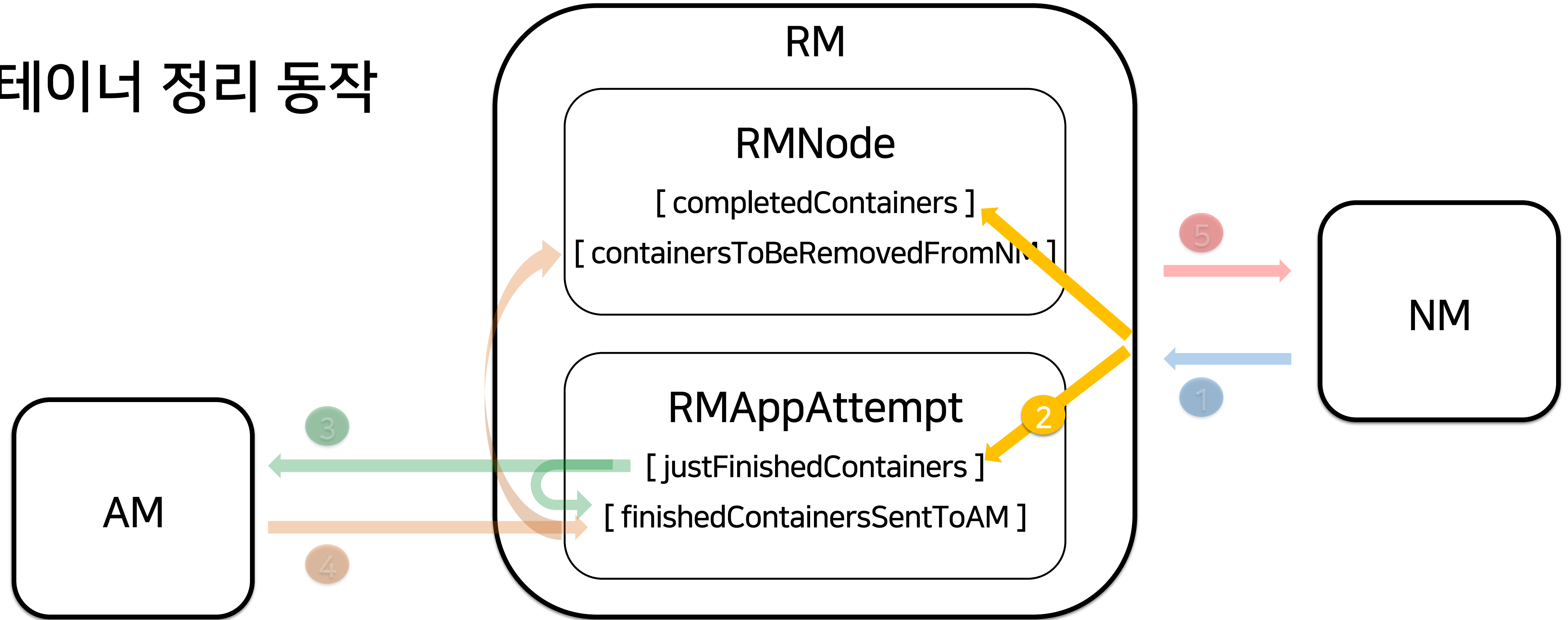
컨테이너 정리 동작



1 NM 가 완료된 컨테이너를 Heartbeat 에 실어 보낸다.

3.3.3 RM memory leak

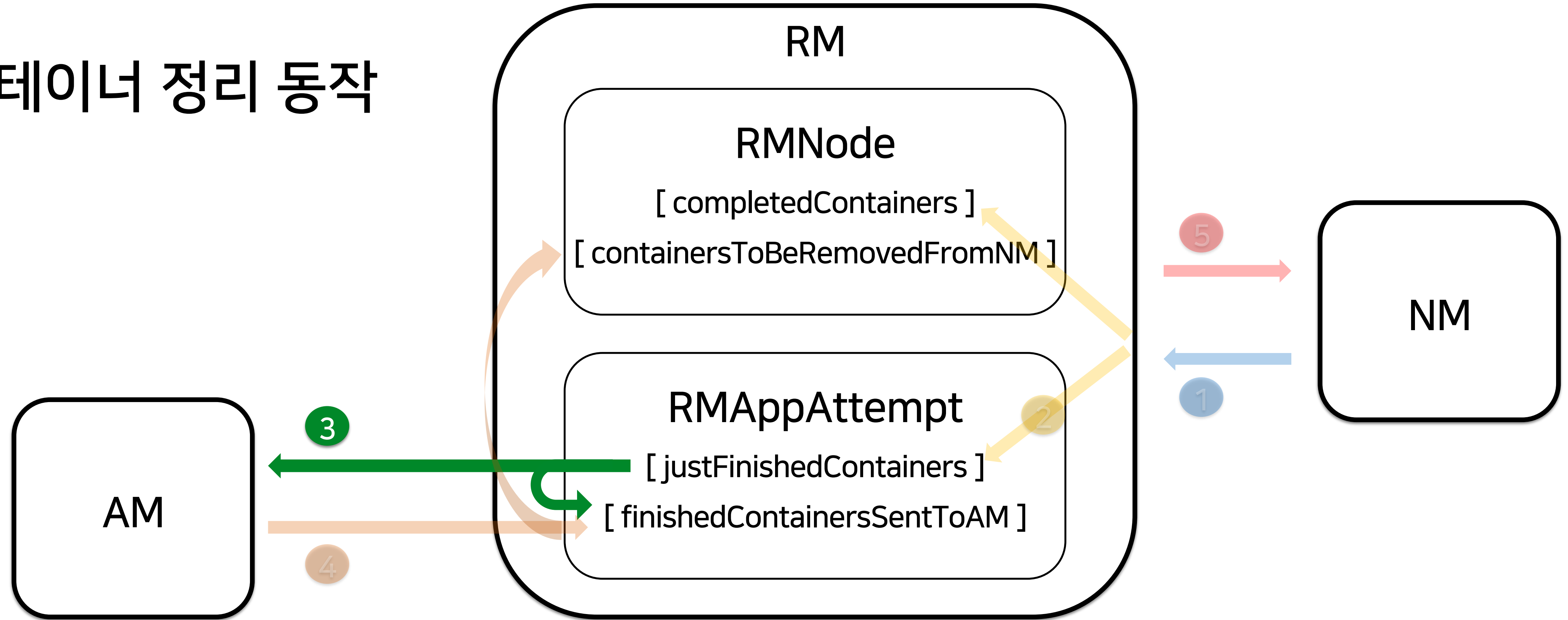
컨테이너 정리 동작



2 완료된 컨테이너 정보가 등록된다.

3.3.3 RM memory leak

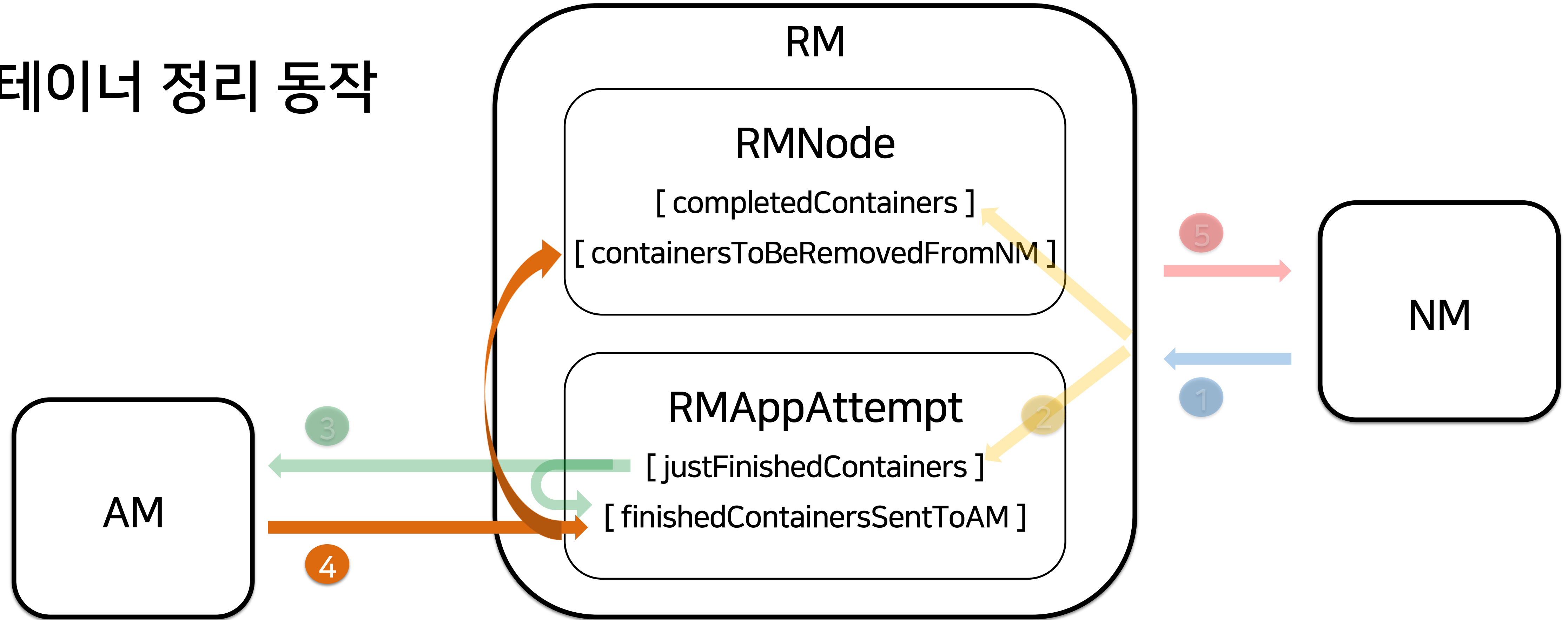
컨테이너 정리 동작



- 3** AM Heartbeat response 로 Finished Container 가 전달되면서 finishedContainersSentToAM 으로 이동

3.3.3 RM memory leak

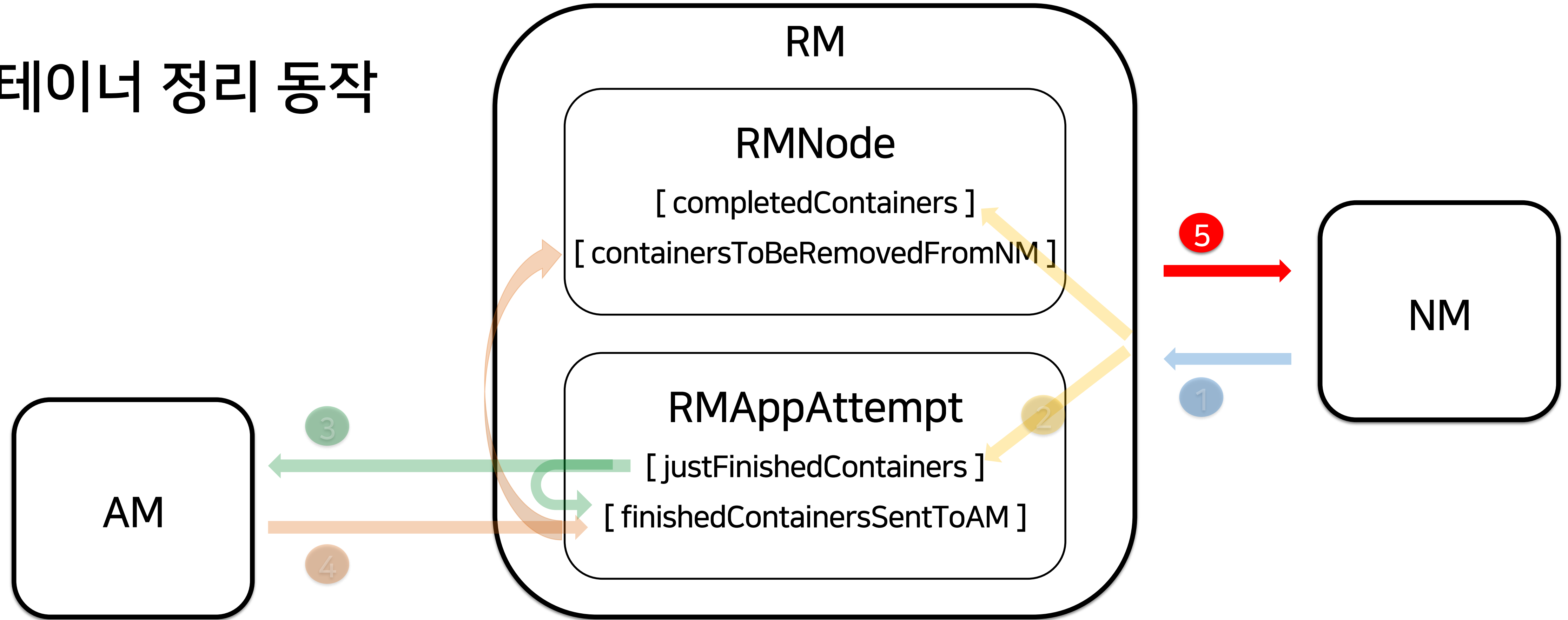
컨테이너 정리 동작



4 다음번 AM Heartbeat 전달되면서 finishedContainersSentToAM 정리 & containersToBeRemovedFromNM 등록

3.3.3 RM memory leak

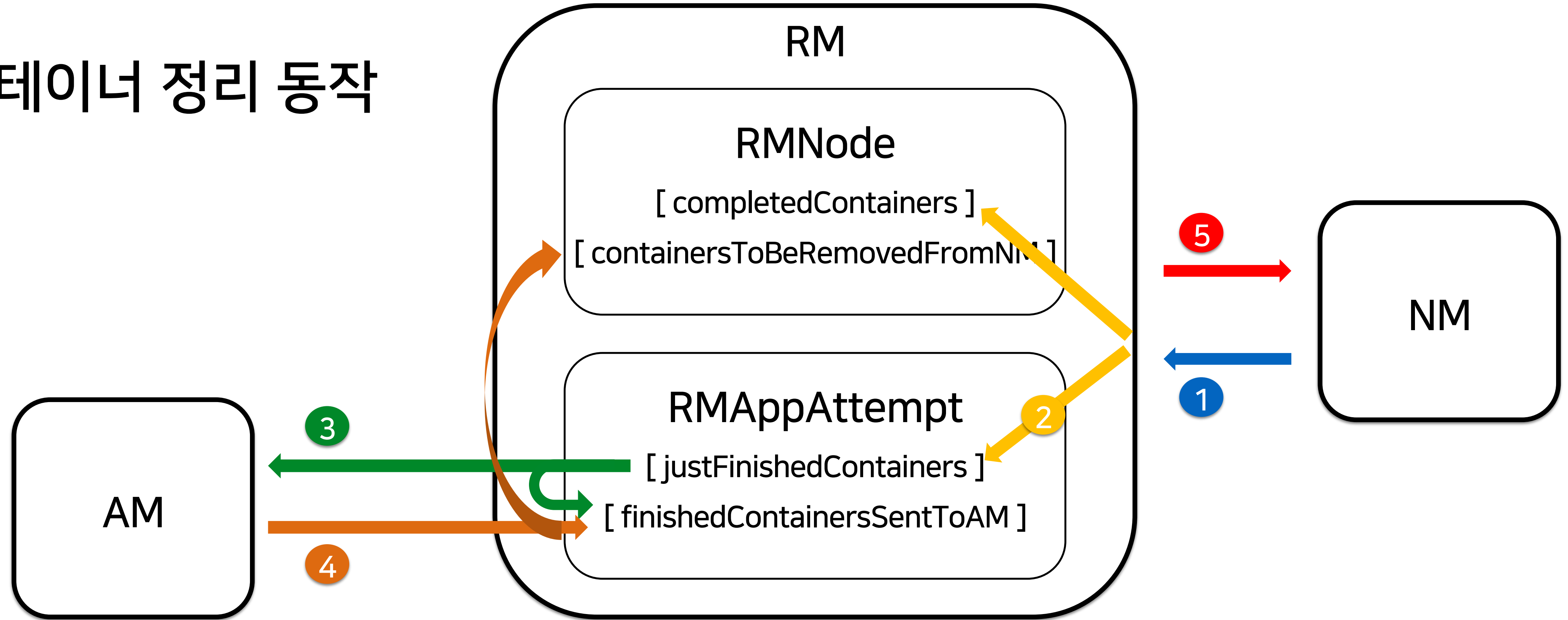
컨테이너 정리 동작



5 다음 NM Heartbeat 시점에 containersToBeRemovedFromNM 에 등록된 컨테이너를 completedContainers 에서 제거한다.

3.3.3 RM memory leak

컨테이너 정리 동작

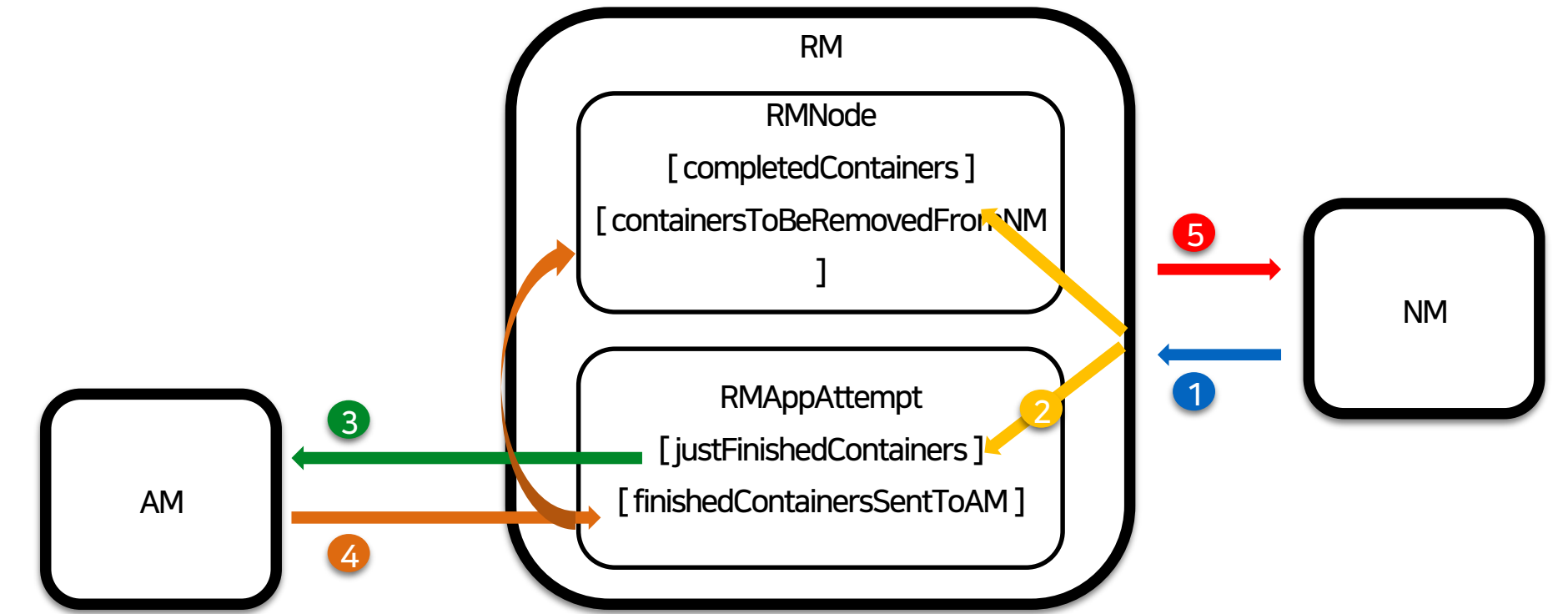


3.3.3 RM memory leak

일련의 과정 중, 실패시 메모리 누수

3, 4 과정 도중 AM 이 죽거나 작업 완료로 종료
=> YARN-10467 + YARN-10895 (YARN service)

사용자가 앱을 직접 죽이는 경우 (CLI or WebUI)
=> YARN-10895



3.3.3 RM memory leak

메모리 누수 관련 패치 정리

- YARN-9642: Failover 발생시 Standby RM 의 Scheduler 쪽 메모리가 정리되지 않는 이슈
- YARN-10467: 일반 작업 종료 혹은 중간 실패시 메모리가 정리되지 않는 이슈
- YARN-10895: YARN-10467 을 YARN service 에 대해서도 적용 + 사용자에게 의한 작업 kill 시에 메모리가 정리되지 않는 이슈 해결 (Contribution)

3.3.4 YARN Service API 서버 분리

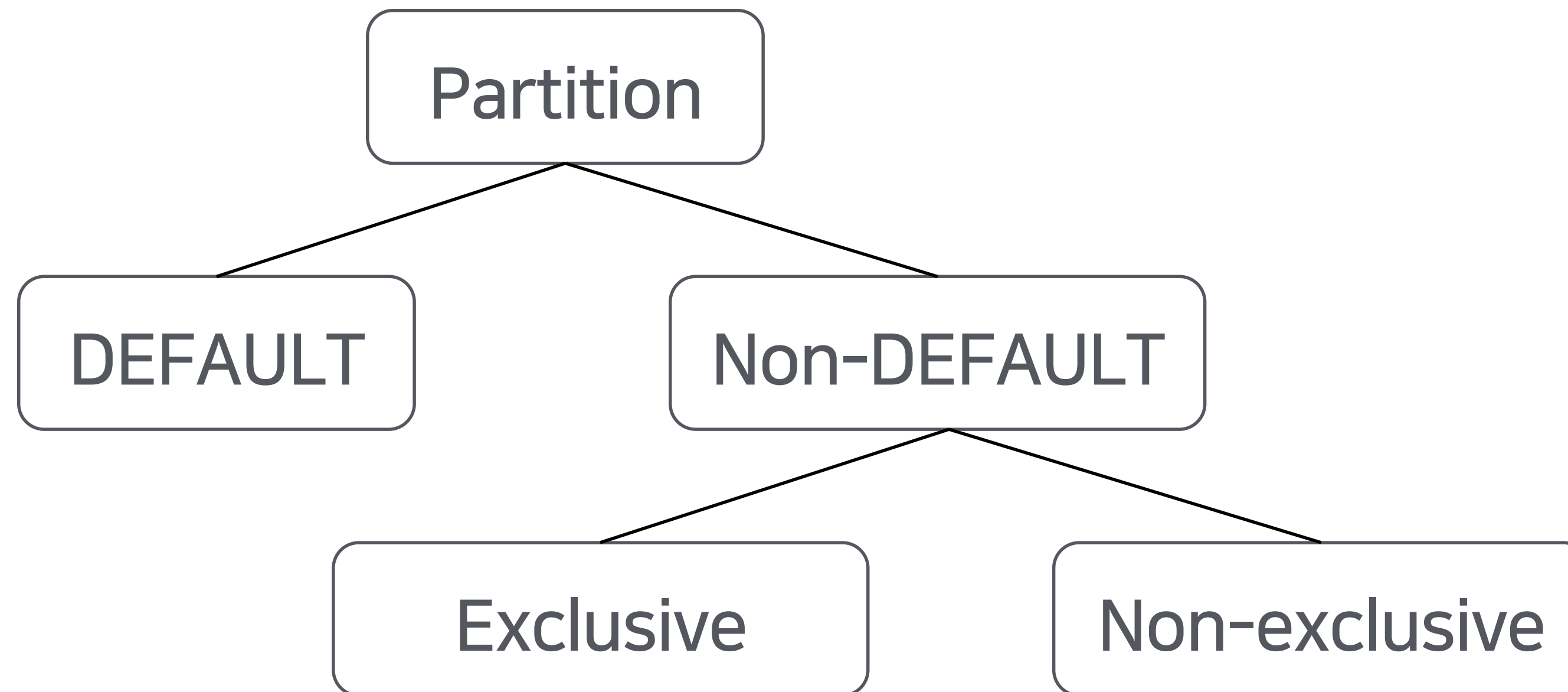
YARN-10898

- Active RM 에서만 제공되는 API 서버를 분리

얻을 수 있는 이점

- 로드밸런서를 사용하는 이중화로 RM 로드 분산
- YARN Service API 호출이 RM 에 영향을 주지 않도록 할 수 있다.
- YARN Service API 서버 로그를 분리하여 문제 상황 파악이 가능하다.

3.3.5 non-exclusive 자원 제어



3.3.5 non-exclusive 자원 제어

자원 공유

- Non-exclusive 파티션 남는 자원 DEFAULT 파티션과 공유
- 빌려쓰던 중 Non-exclusive 파티션 작업 제출시 Preemption

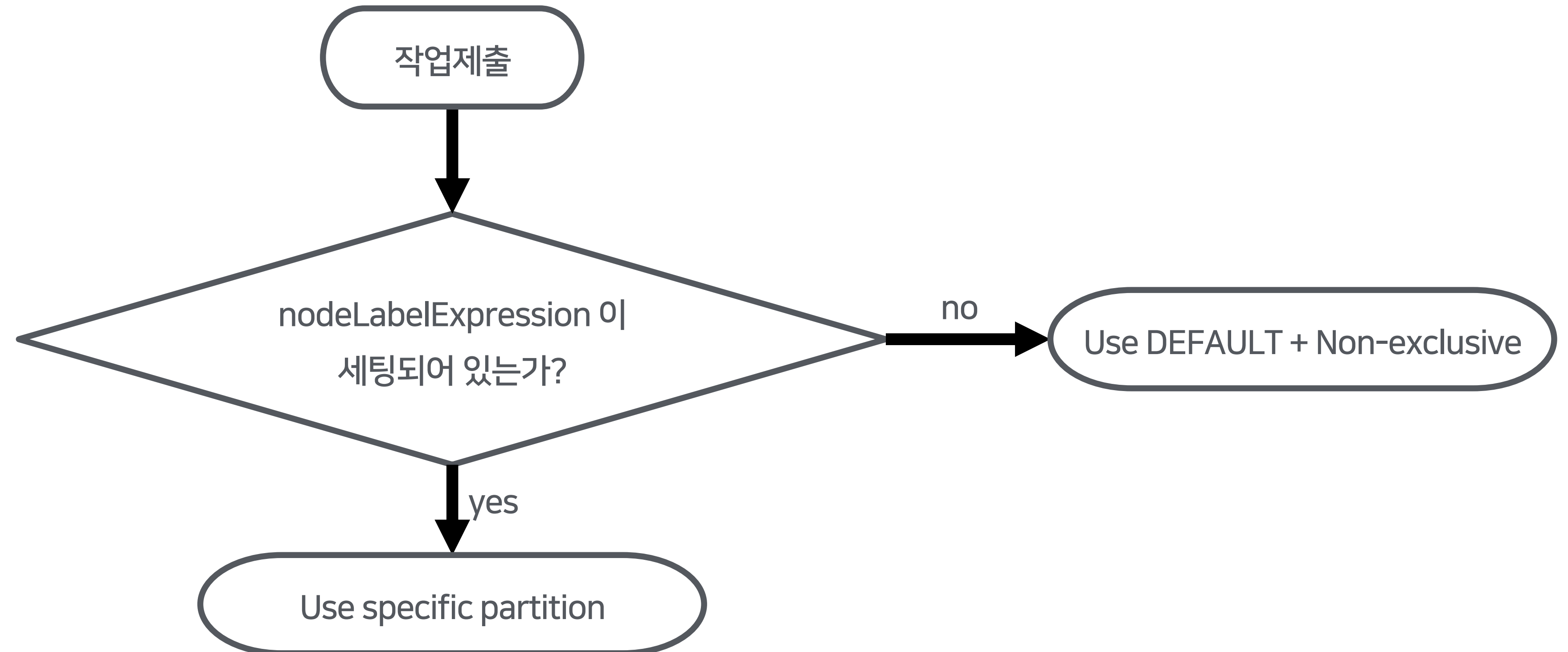
3.3.5 non-exclusive 자원 제어

안티 패턴

- 특정 파티션의 자원의 사용률은 거의 100%
- 오랜기간 구동되는 배치 작업이나 long-lived 앱이 Non-exclusive 빌려쓰는 경우
어차피 곧 Preemption 발생
 - => 자원 할당 낭비
 - => Preemption 비용 발생
 - => 작업 지연

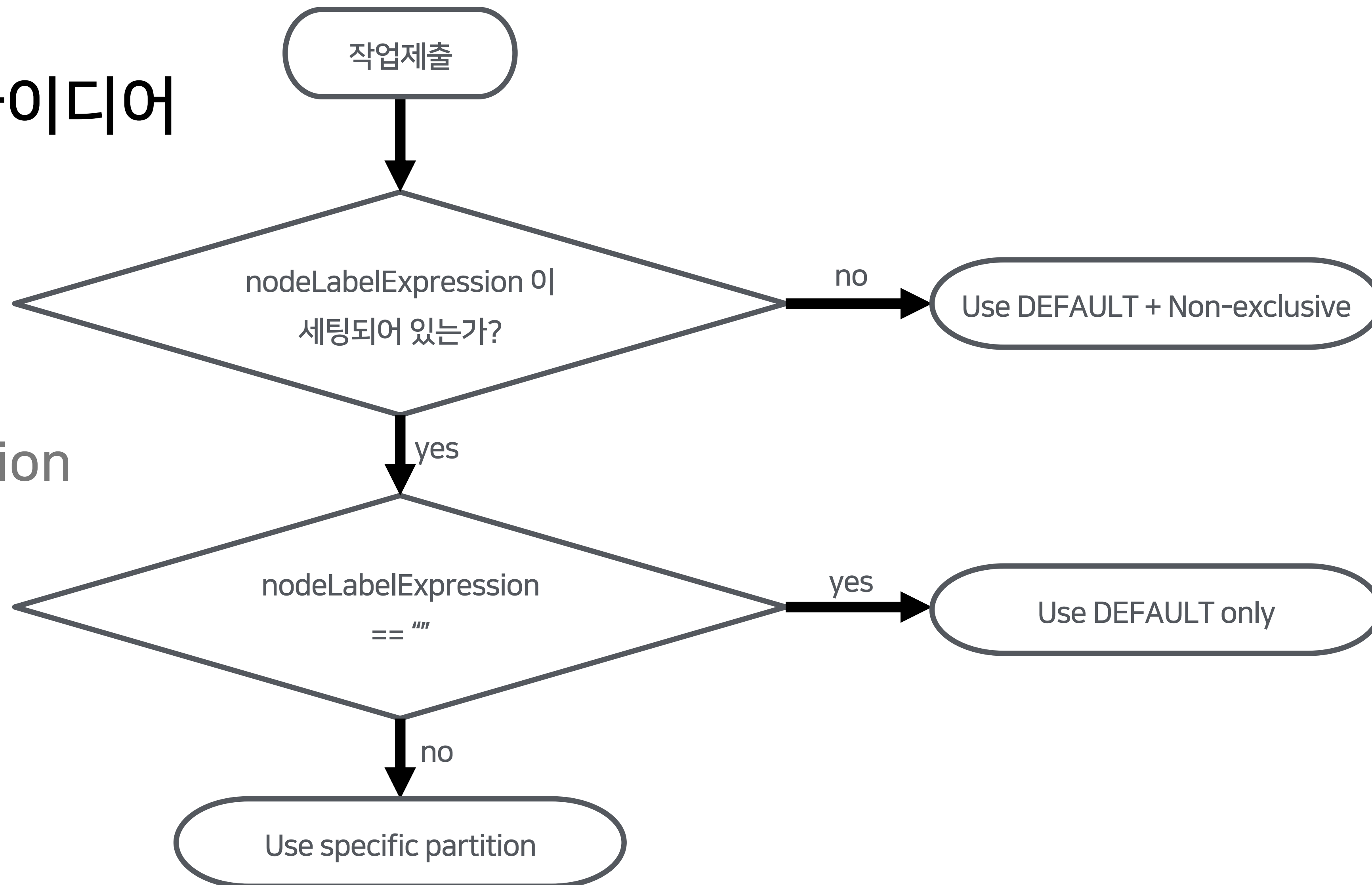
3.3.5 non-exclusive 자원 제어

기존



3.3.5 non-exclusive 자원 제어

YARN-10899 아이디어



nodeLabelExpression
명시적으로 빈값 세팅

3.3.5 non-exclusive 자원 제어

YARN-10899

- 사용자가 자신의 작업 패턴에 따라 non-exclusive 유틸리티 자원을 쓰지 말지 결정 가능
- non-exclusive 파티션 utilization 을 높이면서도 작업 지연 문제 해결
- 모든 프레임워크에서 nodeLabelExpression 설정을 지원하지 않는다.
=> 모든 작업에 대해 적용할 수는 없는 기능

3.3.5 non-exclusive 자원 제어

MapReduce

`mapreduce.job.node-label-expression`

Spark

`spark.yarn.am.nodeLabelExpression`

`spark.yarn.executor.nodeLabelExpression`

3.3.5 non-exclusive 자원 제어

YARN Service

다음 패치들이 적용되어 있다면 placement_policy 로 제어 가능

- <https://issues.apache.org/jira/browse/YARN-8095>
- <https://issues.apache.org/jira/browse/YARN-9307> (Hadoop 3.1)
- <https://issues.apache.org/jira/browse/YARN-7863> (Hadoop 3.2)

3.3.5 non-exclusive 자원 제어

YARN Service

다음 패치들이 적용된

- <https://issues.apache.org/jira/browse/YARN-1095>
- <https://issues.apache.org/jira/browse/YARN-1307> (Hadoop 3.1)
- <https://issues.apache.org/jira/browse/YARN-1363> (Hadoop 3.2)

```

...
    "placement_policy": {
      "constraints": [
        {
          "type": "ANTI_AFFINITY",
          "scope": "NODE",
          "target_tags": [
            "WS"
          ],
          "node_partitions": [
            ""
          ]
        }
      ]
    }
  ]
}

```

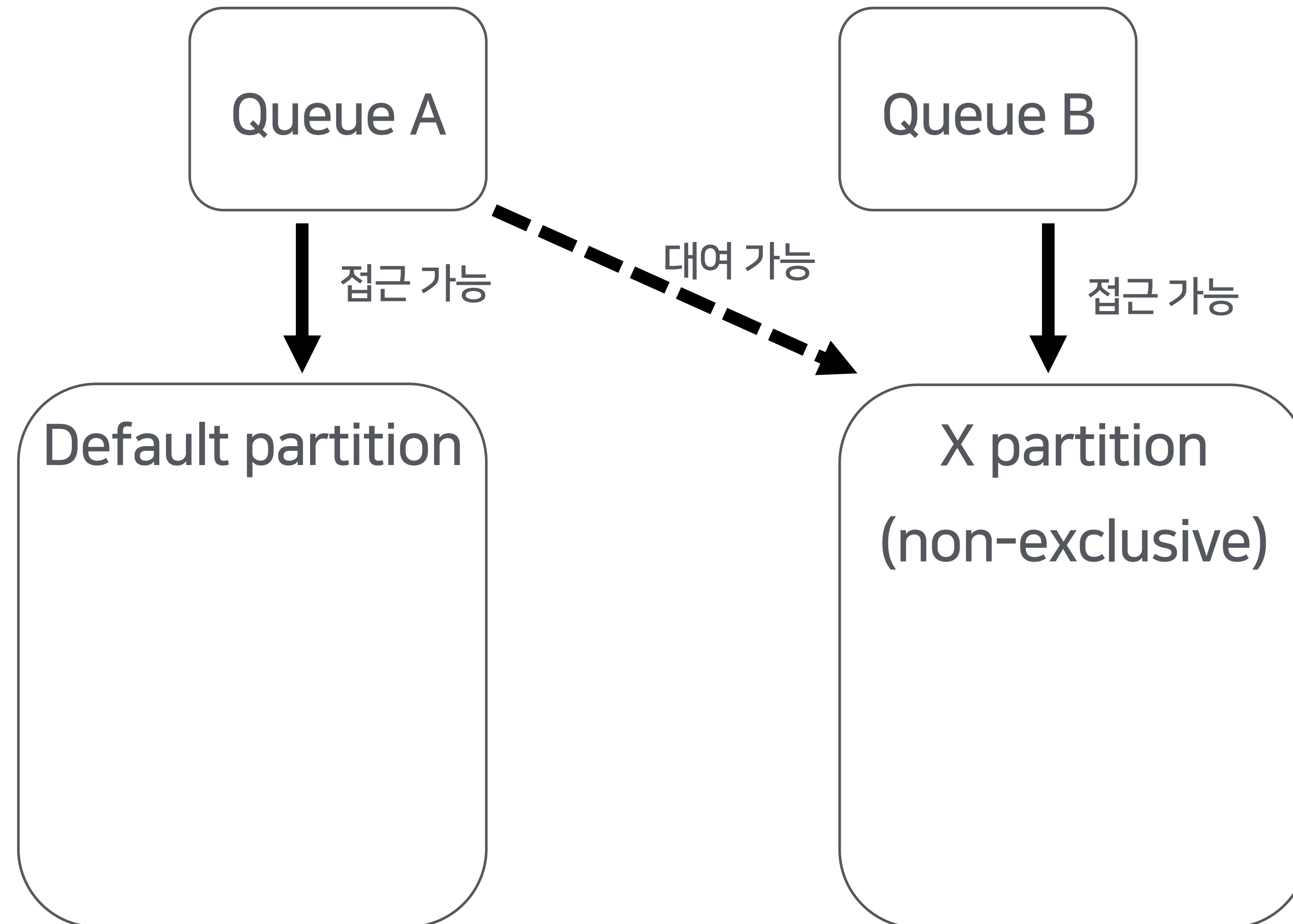
어 가능

[1095](#)

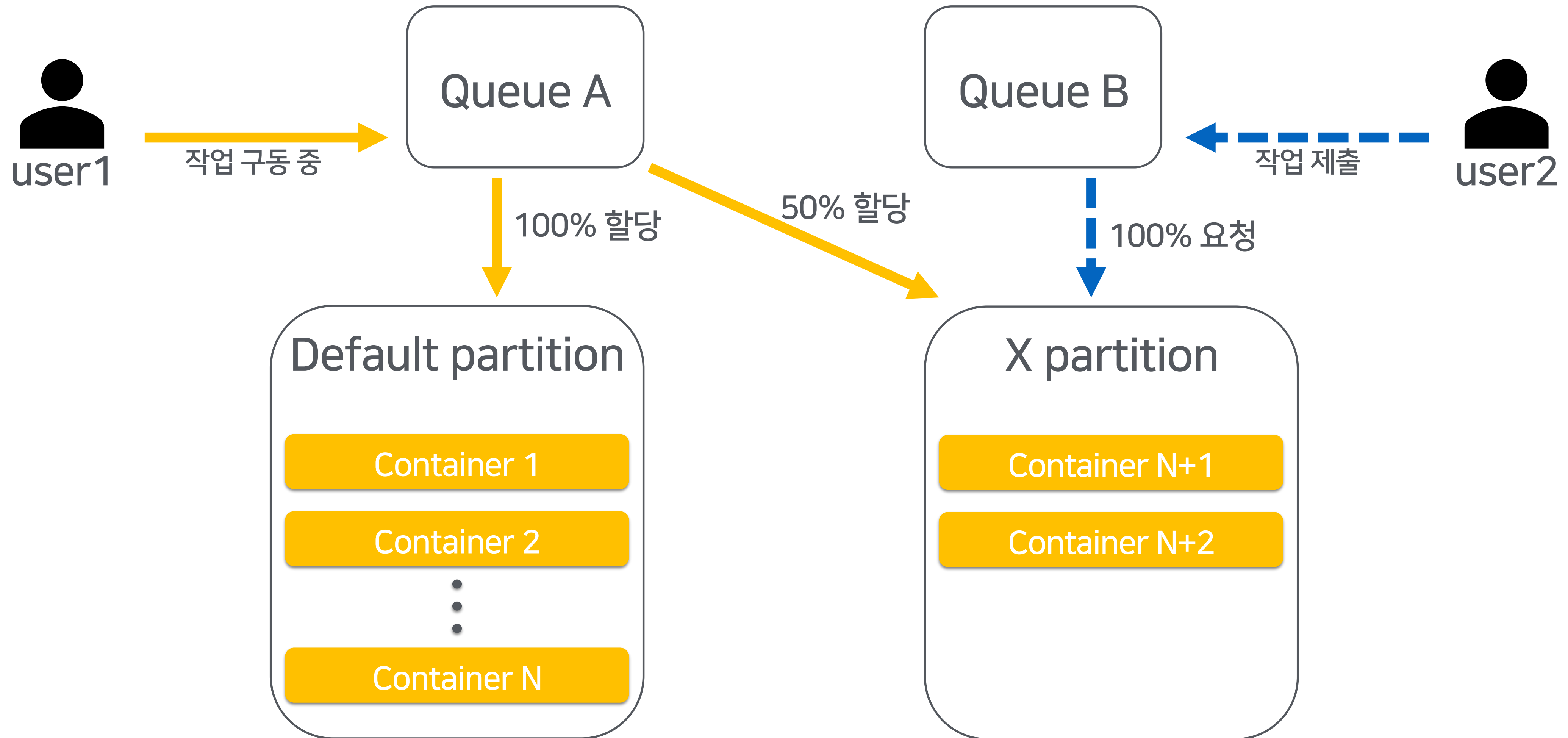
[1307](#) (Hadoop 3.1)

[1363](#) (Hadoop 3.2)

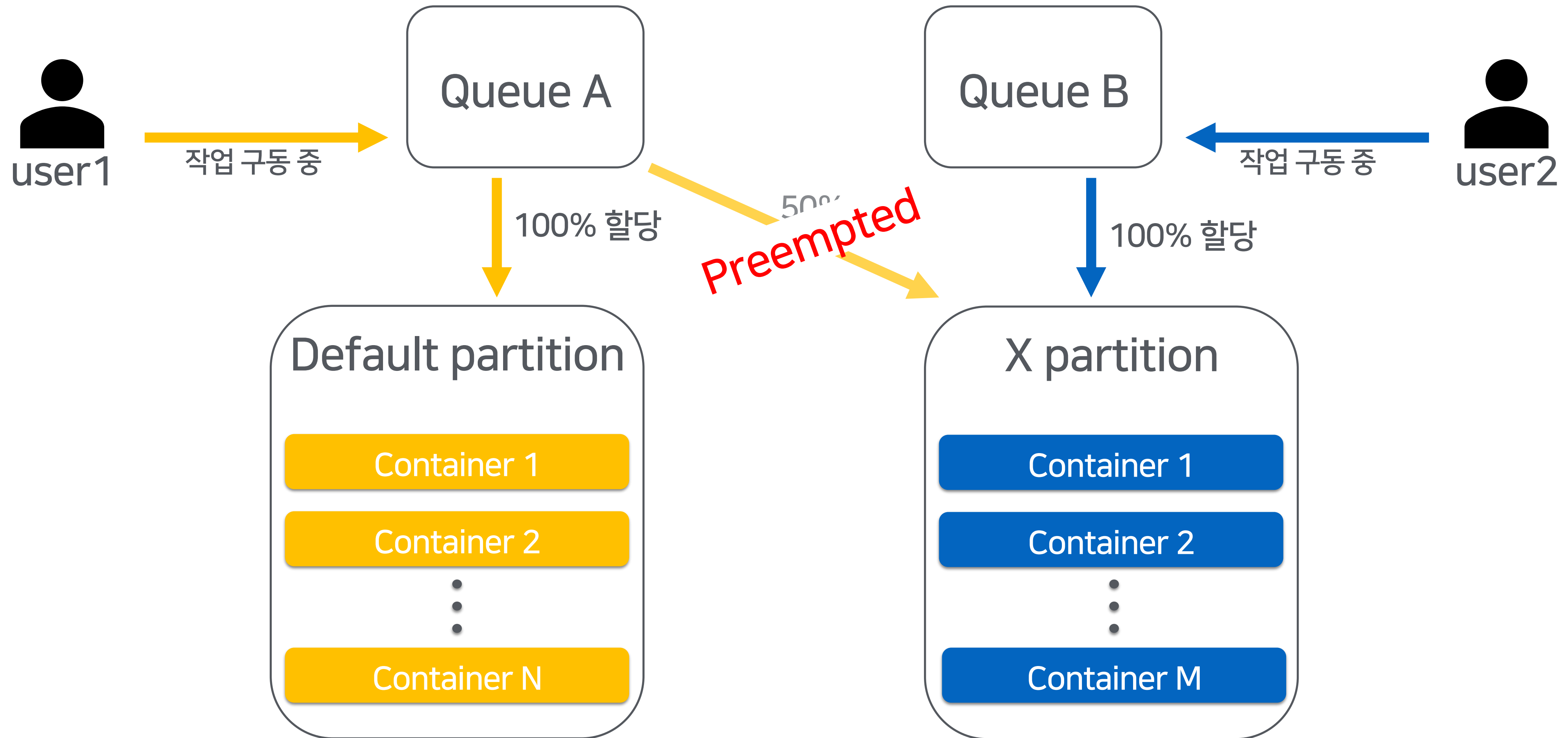
3.3.6 자원 Preemption 실패



3.3.6 자원 Preemption 실패



3.3.6 자원 Preemption 실패



3.3.6 자원 Preemption 실패

YARN-10892

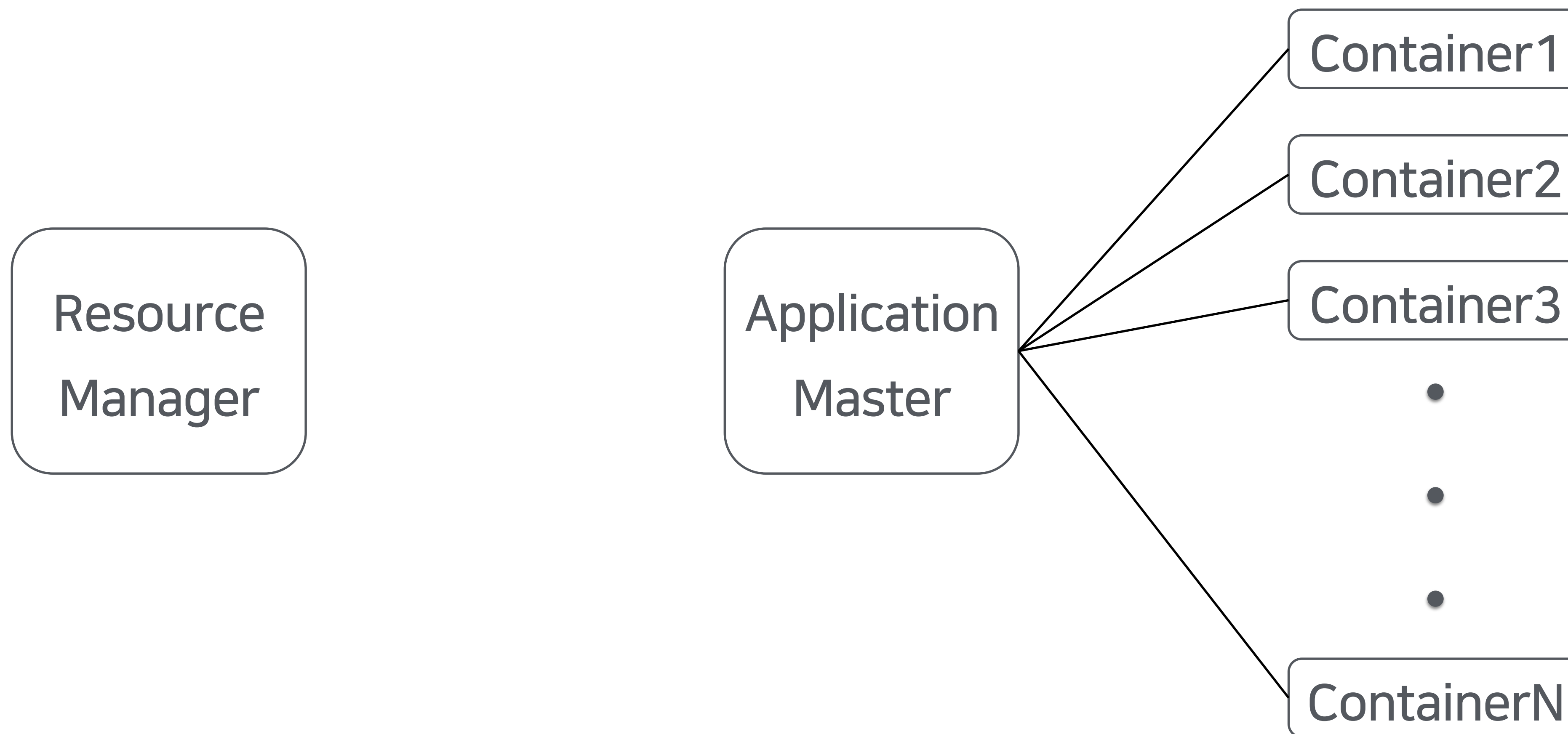
- 앞선 상황에서 Preemption 이 안되는 버그 수정
- HashMap 구조체 Iteration 중 remove 수행
 - > java.util.ConcurrentModificationException
- Iteration 용 keyset 으로 clone 해서 사용

3.3.7 YARN Service 컨테이너 요청 누락

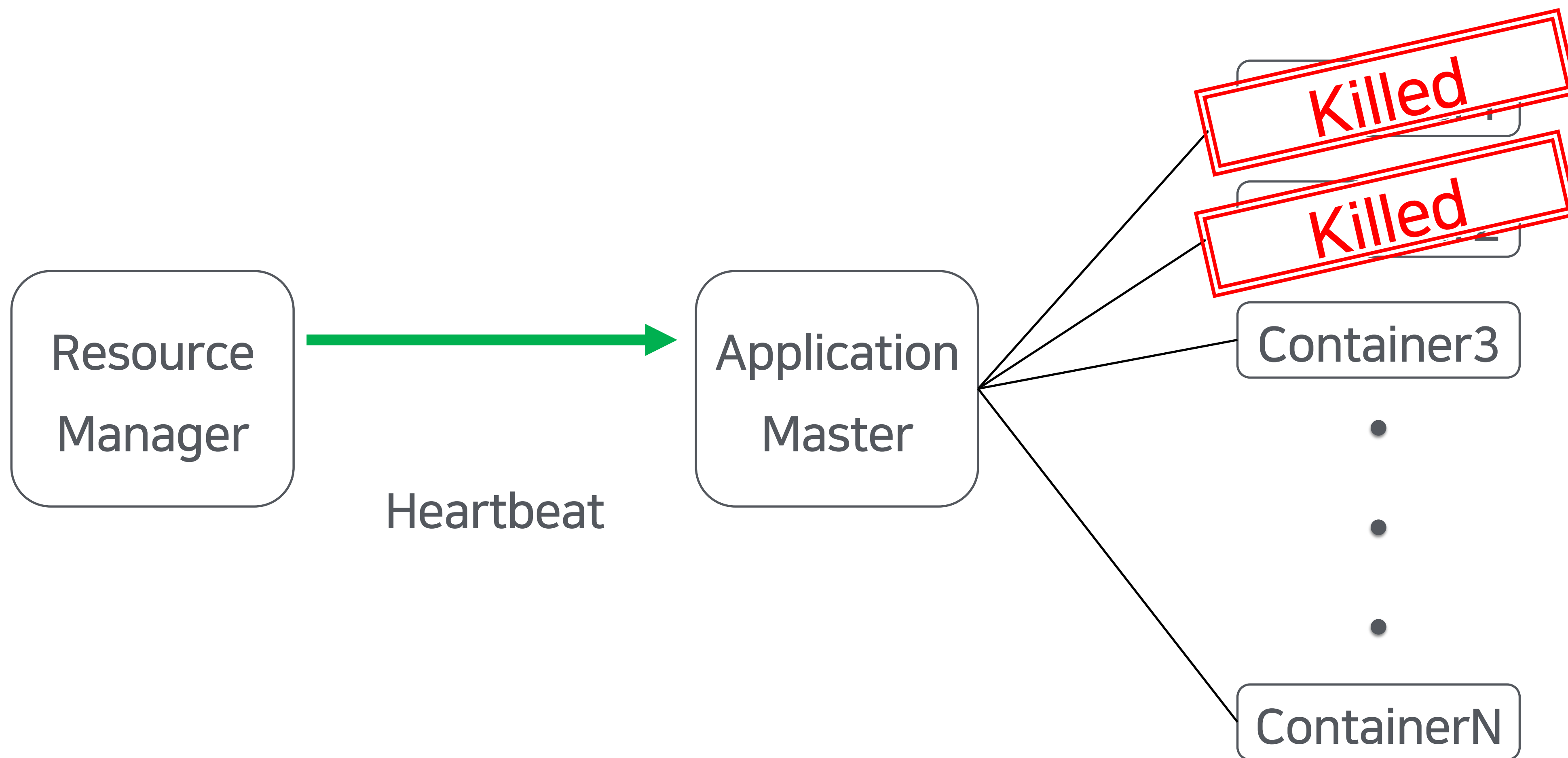
YARN Service 컨테이너가 죽으면

- AM 이 RM 에게 죽은 컨테이너 수만큼 재요청
- 이 문제는 YARN Service 의 placement_policy 존재시 발생
- AM - RM 간 Heartbeat 찰나에 여러 컨테이너가 죽으면 여러개가 아닌 1개만 요청

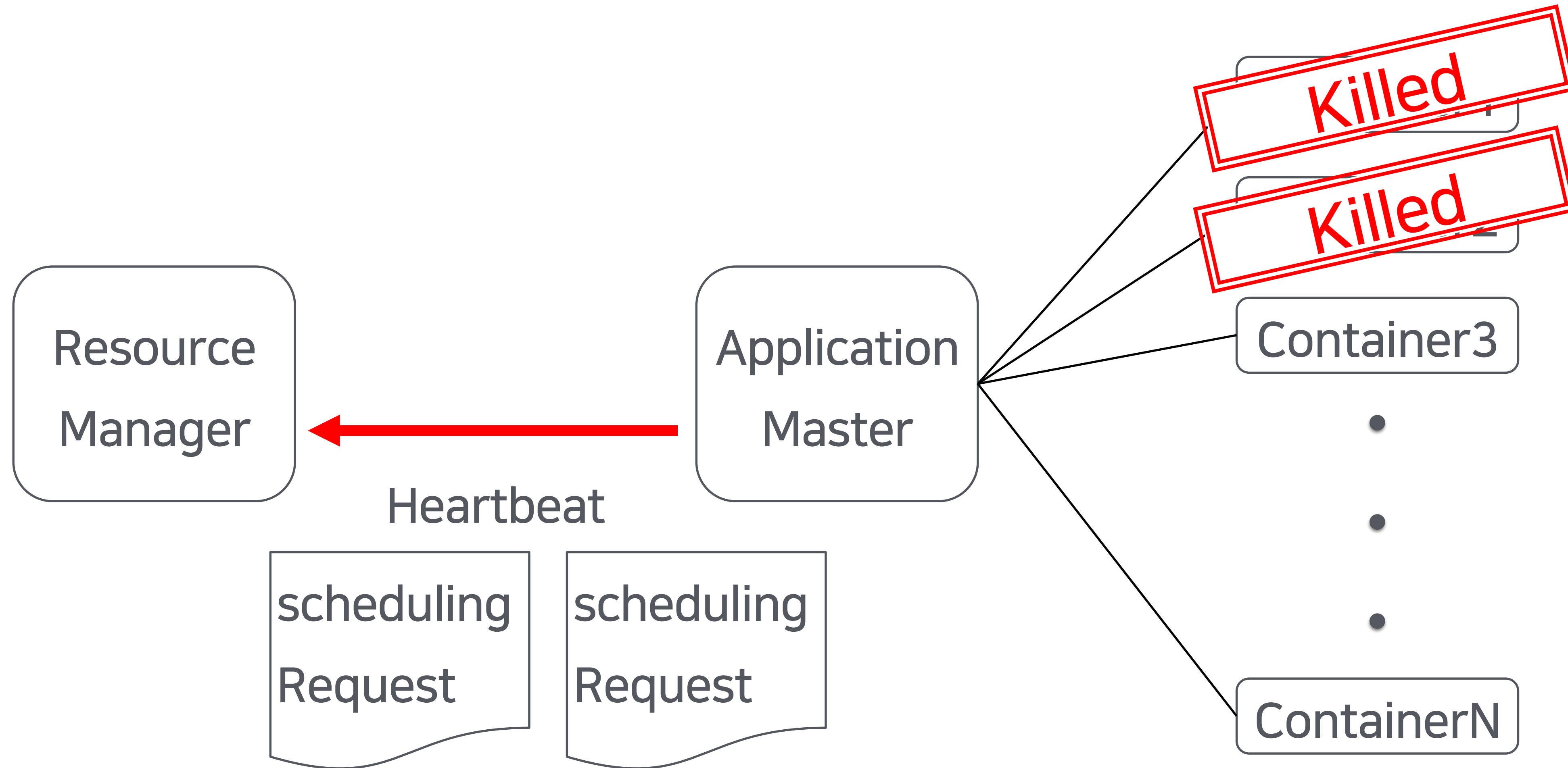
3.3.7 YARN Service 컨테이너 요청 누락



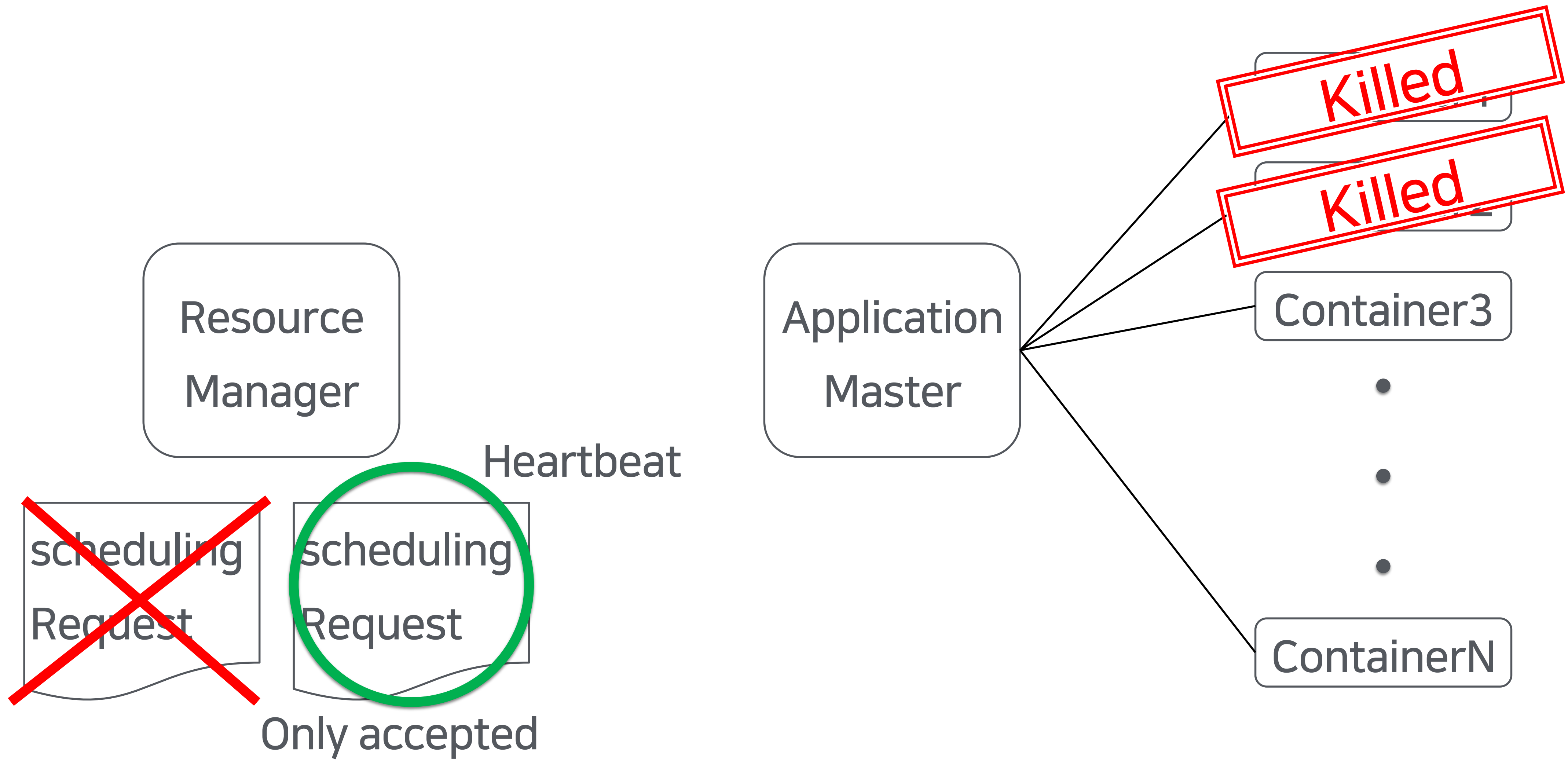
3.3.7 YARN Service 컨테이너 요청 누락



3.3.7 YARN Service 컨테이너 요청 누락



3.3.7 YARN Service 컨테이너 요청 누락



3.3.7 YARN Service 컨테이너 요청 누락

YARN-10968

- AMRMClientImpl 에서 Heartbeat 시점에 "기존요청 + 신규요청"
- 불필요하게 추가 컨테이너가 할당되고 반환되는 경우가 있을 수 있다.
 - > 필요한 컨테이너보다 적게 할당되고 멈추는 경우는 막을 수 있다.

3.3.8 YARN Service AM 에서 getContainerStatuses 실패

getContainerStatuses

- AM 이 NM 에게 container 상태를 알아오기 위함 (+ @)
- SaslException 발생

```
2020-06-28 15:19:10,941 [pool-5-thread-4] ERROR instance.ComponentInstance - [COMPINSTANCE hbasemaster-0 :  
container_e63_????????????????_?????_??_??????] Failed to get container status on host:port, will try again  
javax.security.sasl.SaslException: DIGEST-MD5: digest response format violation. Mismatched response. [Caused by  
org.apache.hadoop.ipc.RemoteException(javax.security.sasl.SaslException): DIGEST-MD5: digest response format  
violation. Mismatched response.]  
    at sun.reflect.NativeConstructorAccessorImpl.newInstance0(Native Method)  
...  
    at com.sun.proxy.$Proxy46.getContainerStatuses(Unknown Source)  
    at org.apache.hadoop.yarn.client.api.impl.NMClientImpl.getContainerStatus(NMClientImpl.java:339)  
...
```

3.3.8 YARN Service AM 에서 getContainerStatuses 실패

증상

- NM 에서 실행 중인 Docker 컨테이너 restartPolicy 에 의해 재구동 되었을 때
-> 변경된 컨테이너 ip 를 추적하지 못함, 잘못된 DNS lookup 가능성
- YARN Service upgrade 시 NM 으로 reinitialize 요청 실패 가능성

3.3.8 YARN Service AM 에서 getContainerStatuses 실패


NM Token
마스터키 생성

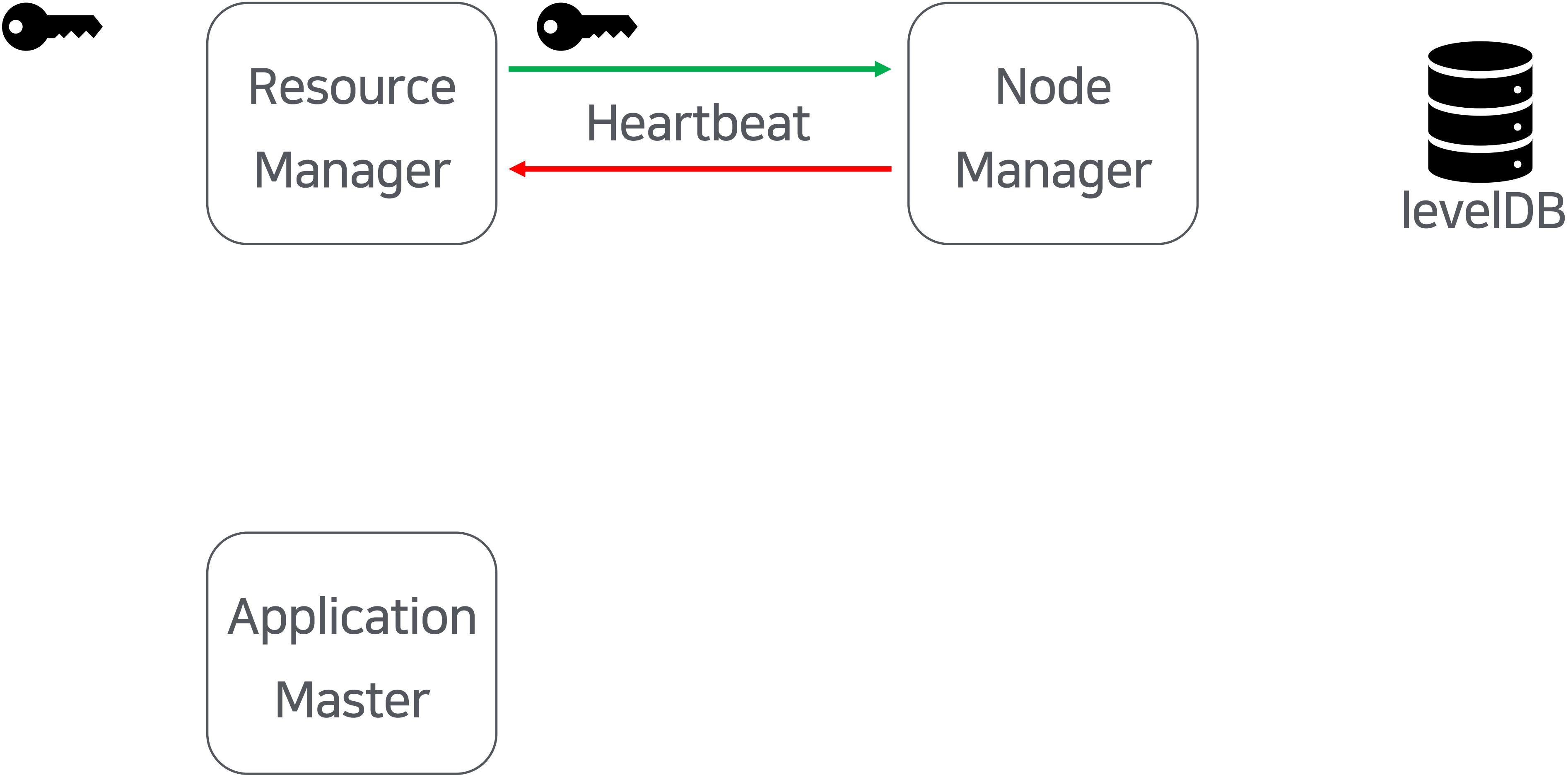
Resource
Manager

Node
Manager

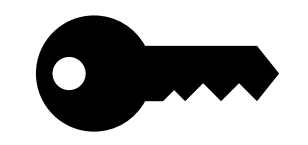

levelDB

Application
Master

3.3.8 YARN Service AM 에서 getContainerStatuses 실패



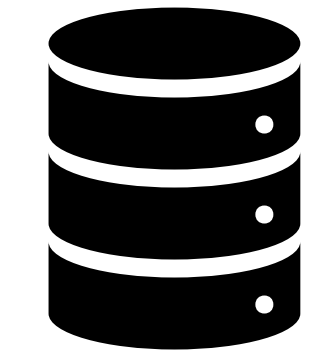
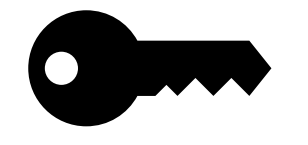
3.3.8 YARN Service AM 에서 getContainerStatuses 실패



Resource
Manager

Node
Manager

Current



levelDB

Application
Master

3.3.8 YARN Service AM 에서 getContainerStatuses 실패


마스터키 생성

Resource
Manager

Node
Manager

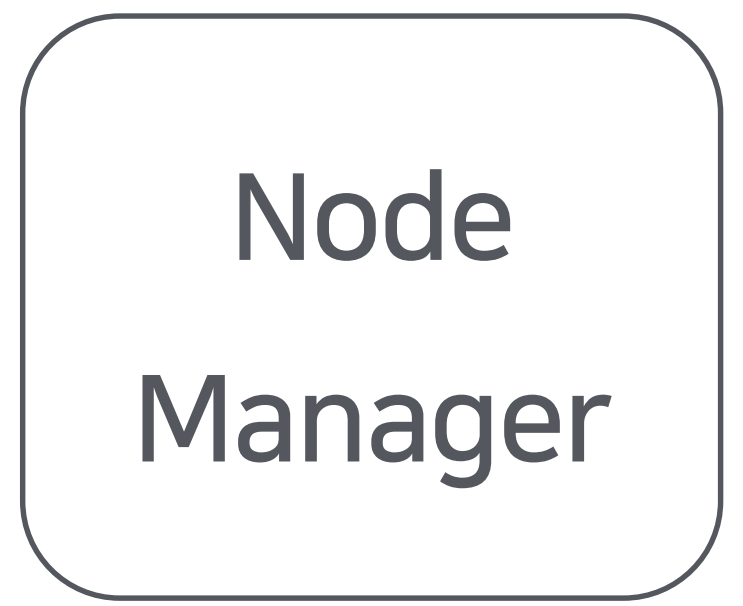
Current 

Application
Master



3.3.8 YARN Service AM 에서 getContainerStatuses 실패


NM Token
마스터키 생성



Current 

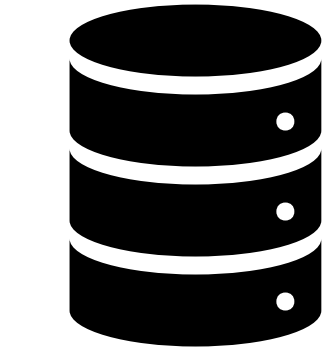
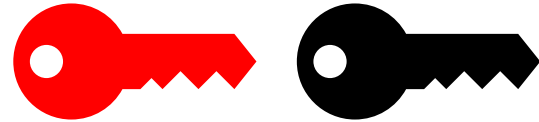


3.3.8 YARN Service AM 에서 getContainerStatuses 실패



Resource
Manager

Node
Manager



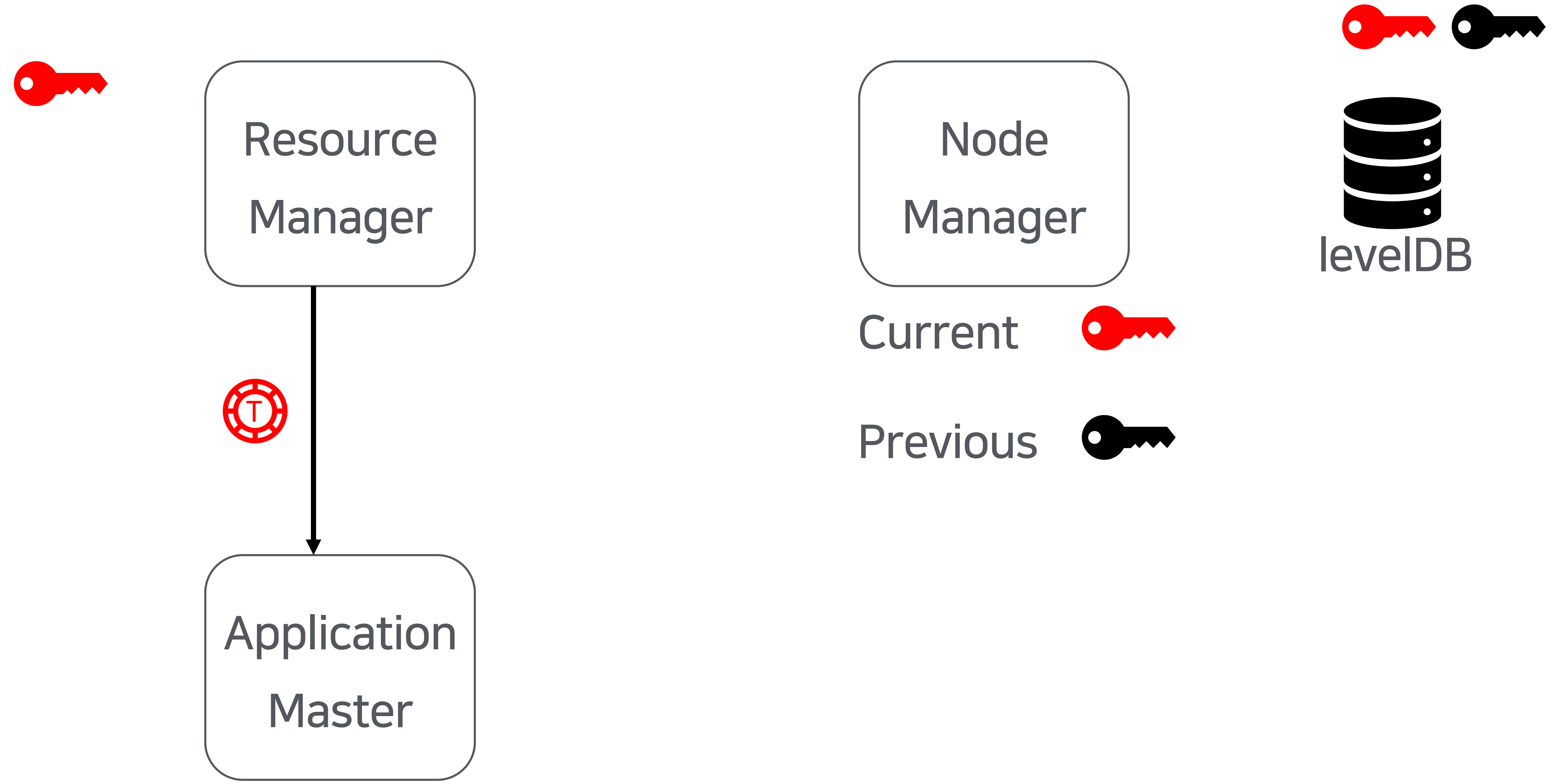
levelDB

Current 

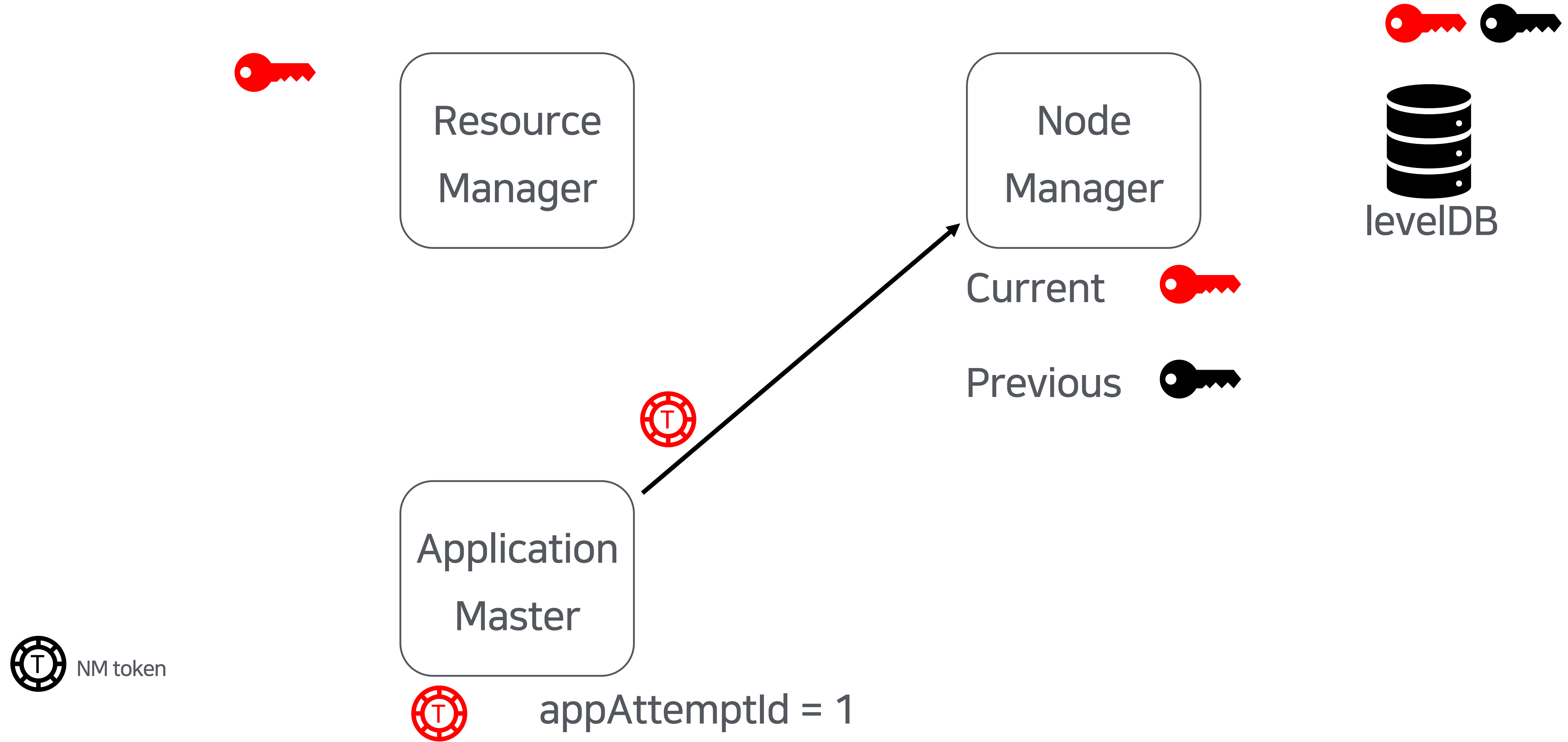
Previous 

Application
Master

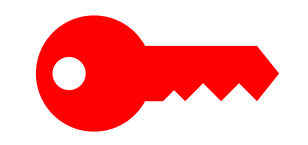
3.3.8 YARN Service AM 에서 getContainerStatuses 실패



3.3.8 YARN Service AM 에서 getContainerStatuses 실패

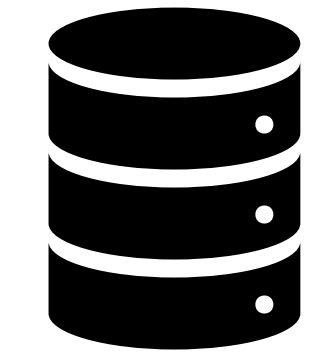
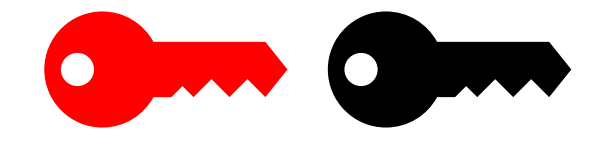


3.3.8 YARN Service AM 에서 getContainerStatuses 실패



Resource Manager

Node Manager



levelDB

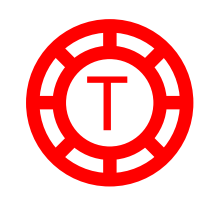
oldMasterKeys = {1 : }

Current

Previous

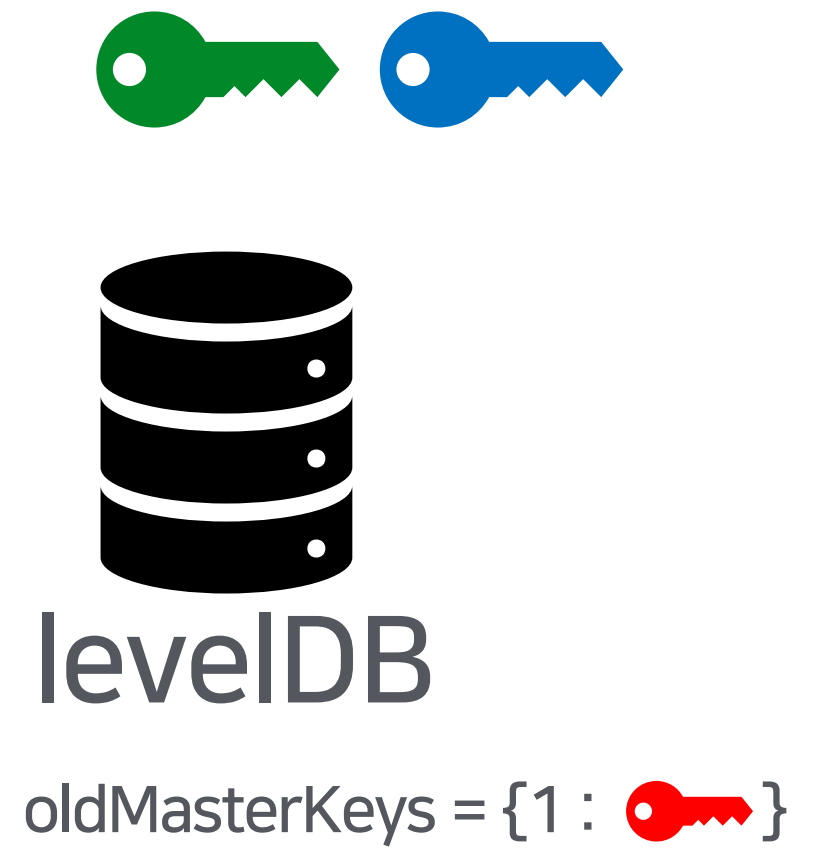
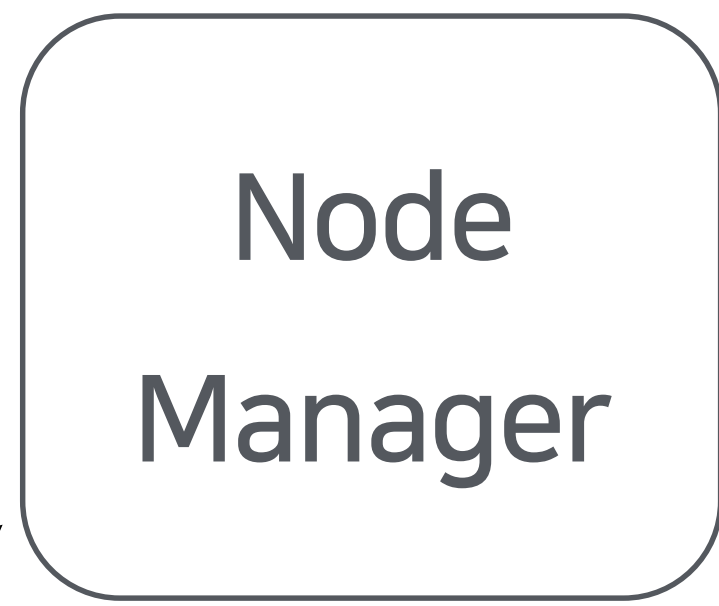
Application Master

call updateAppAttemptKey()
oldMasterKeys = {1 : }



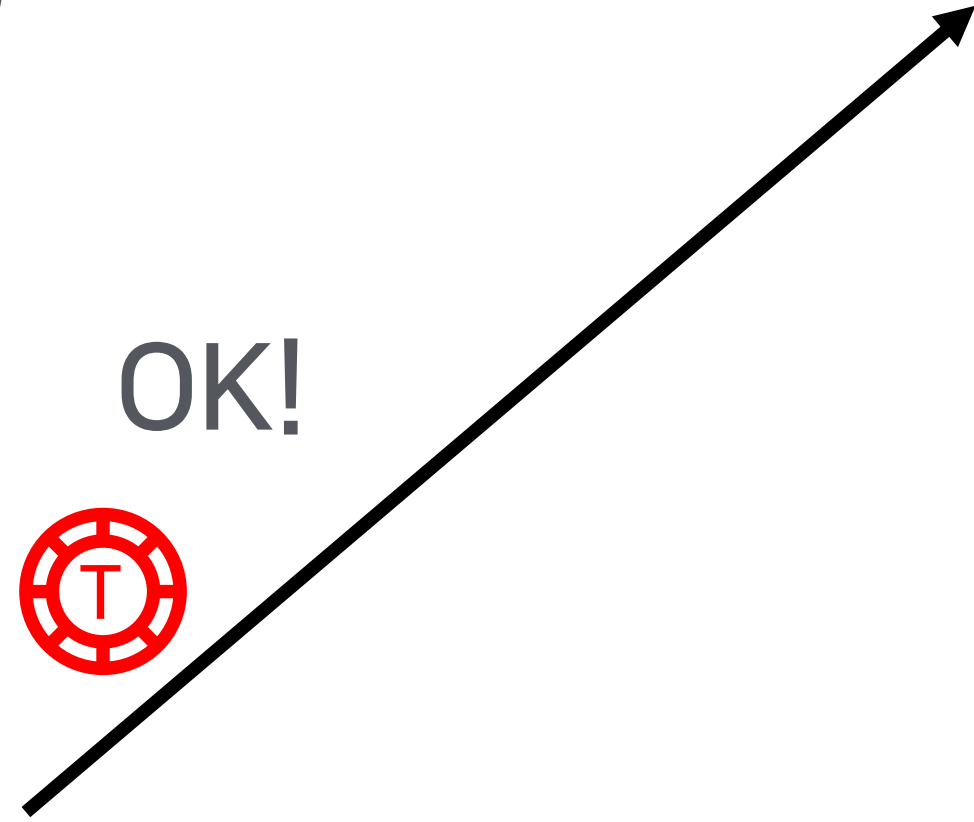
3.3.8 YARN Service AM 에서 getContainerStatuses 실패


NM Token
마스터키 롤링

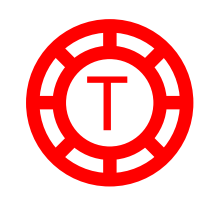


Current 
Previous 

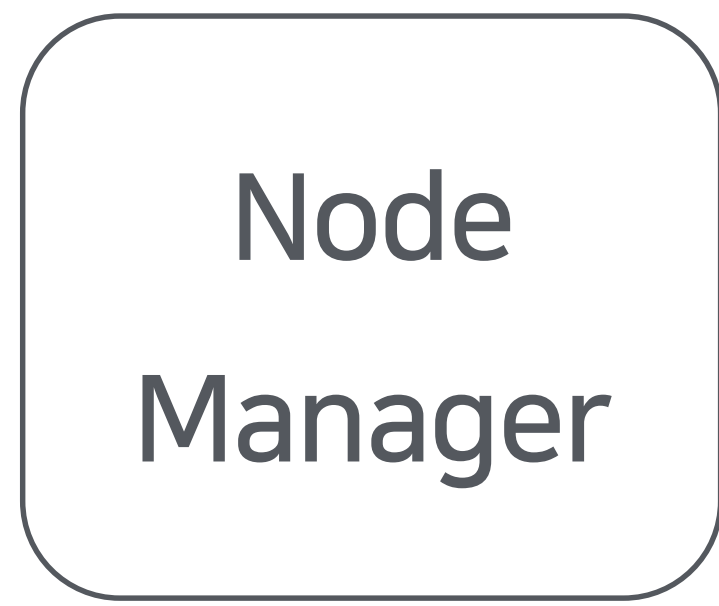
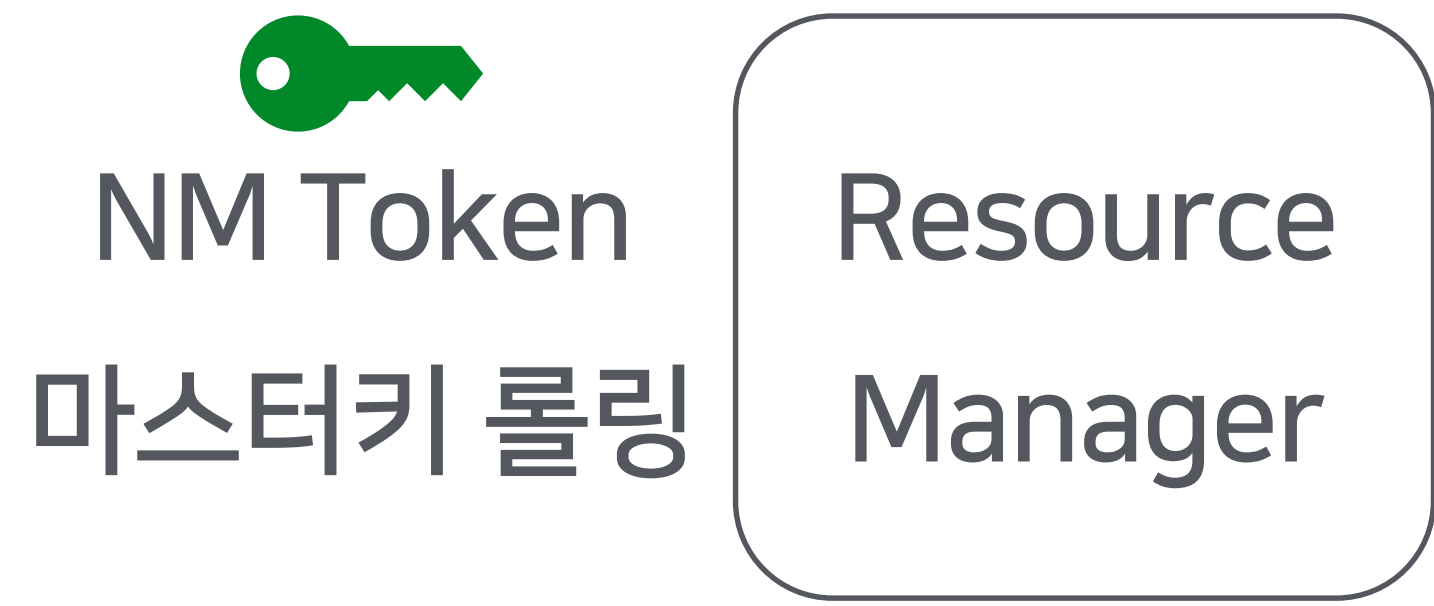
oldMasterKeys = {1 :  }



 NM token



3.3.8 YARN Service AM 에서 getContainerStatuses 실패



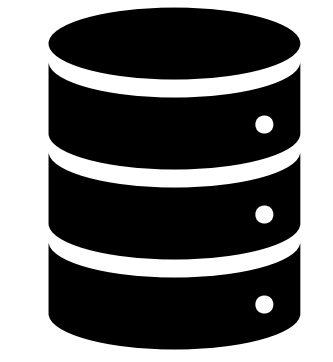
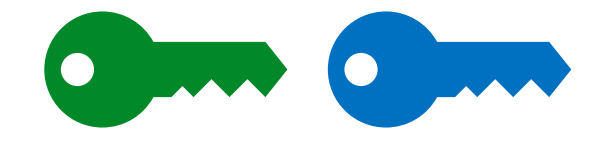
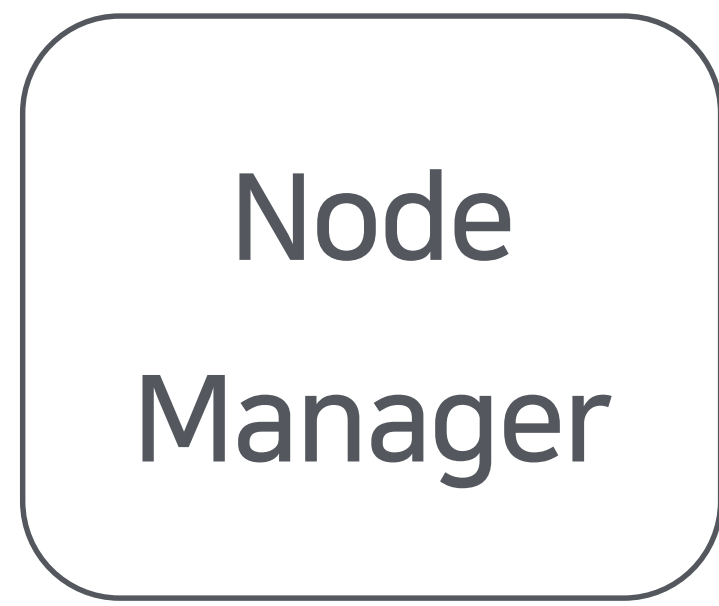
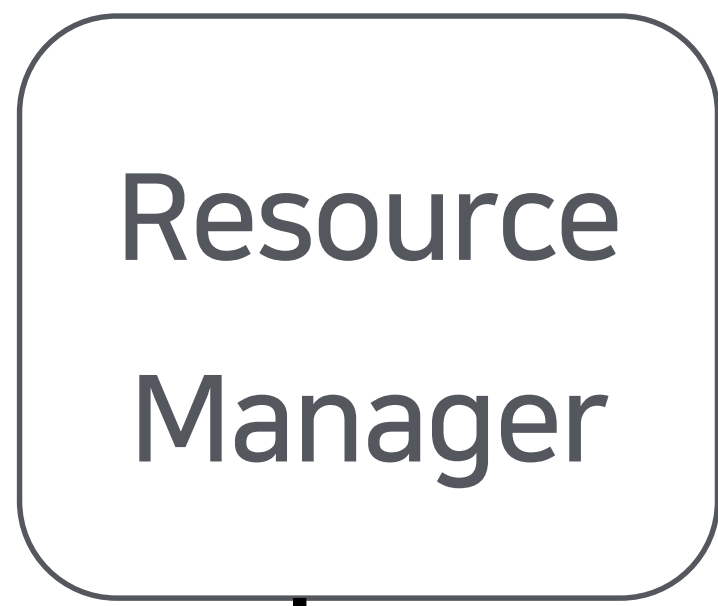
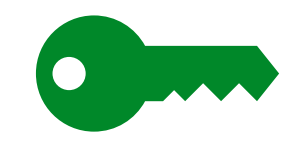
Current

Previous

oldMasterKeys = {1 : }



3.3.8 YARN Service AM 에서 getContainerStatuses 실패



levelDB

oldMasterKeys = {1 : }

Current

Previous

oldMasterKeys = {1 : }

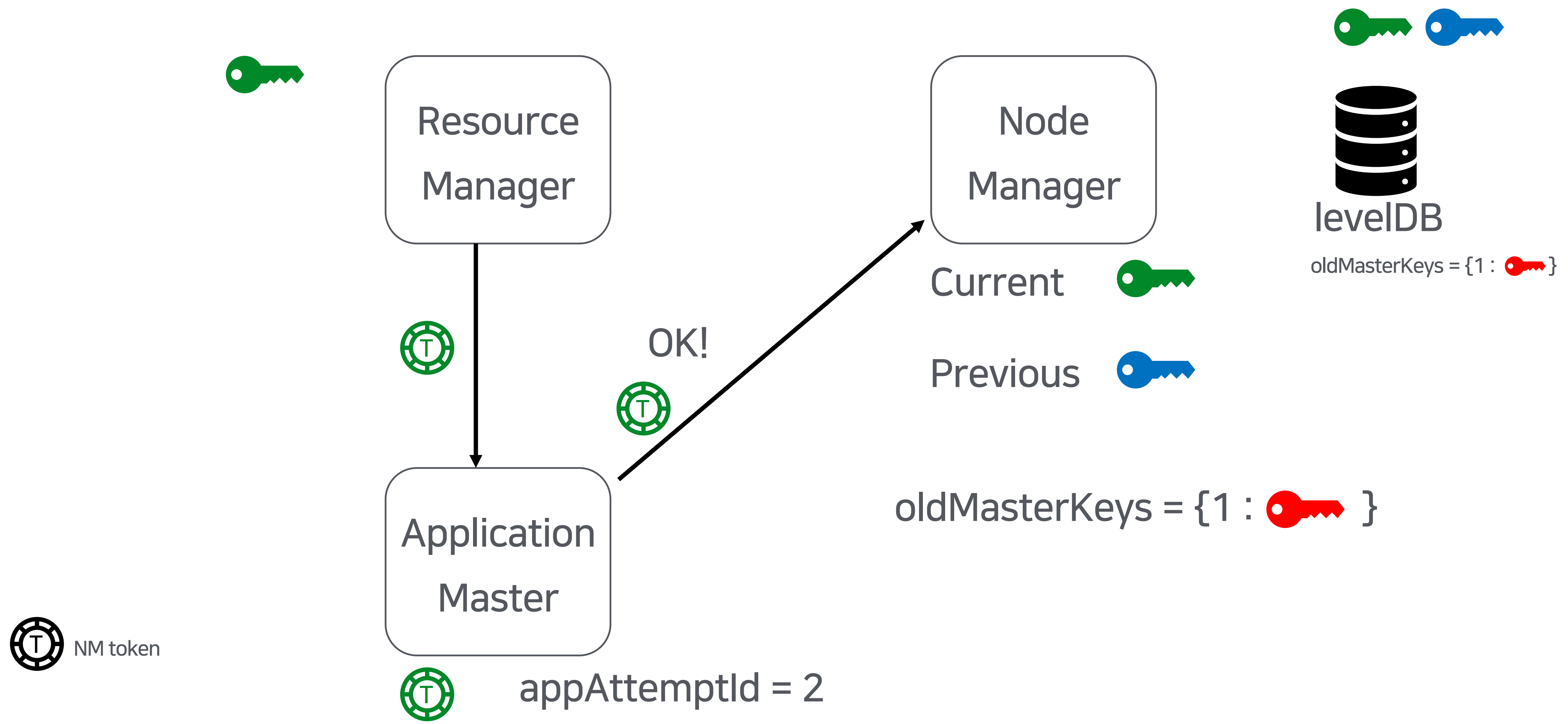


appAttemptId = 2




NM token

3.3.8 YARN Service AM 에서 getContainerStatuses 실패



3.3.8 YARN Service AM 에서 getContainerStatuses 실패


NM Token
마스터키 롤링

yarn.resourcemanager.nm-tokens.master-key-rolling-interval-secs = 86400 (하루)

Resource Manager

Node Manager

Current 

Previous 


levelDB
oldMasterKeys = {1 :  }

Application Master

oldMasterKeys = {1 :  }

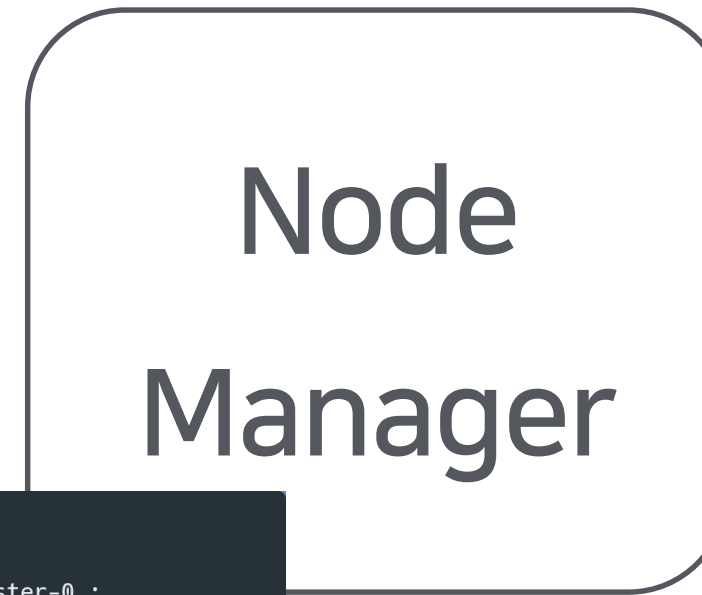
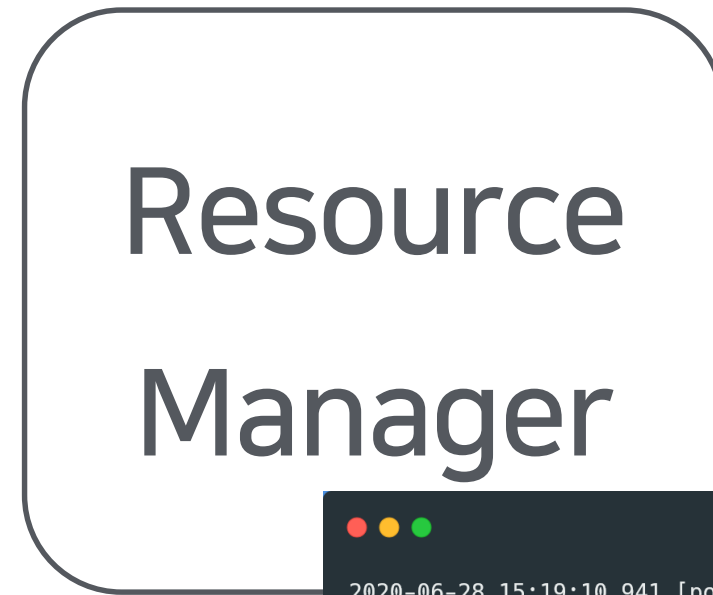
 NM token

 appAttemptId = 2

3.3.8 YARN Service AM 에서 getContainerStatuses 실패

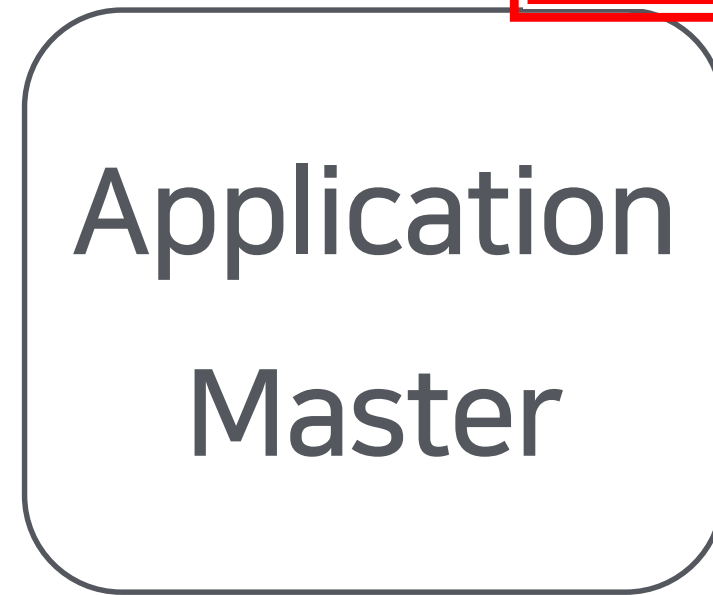

NM Token
마스터키 롤링

yarn.resourcemanager.nm-
tokens.master-key-rolling-interval-
secs = 86400 (하루)

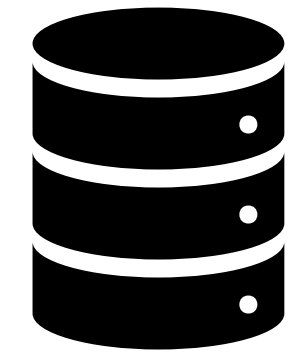


```
2020-06-28 15:19:10,941 [pool-5-thread-4] ERROR instance.ComponentInstance - [COMPINSTANCE hbasemaster-0 :  
container_e63_????????????????_?????_?_?????] Failed to get container status on host:port, will try again  
javax.security.sasl.SaslException: DIGEST-MD5: digest response format violation. Mismatched response. [Caused by  
org.apache.hadoop.ipc.RemoteException(javax.security.sasl.SaslException): DIGEST-MD5: digest response format  
violation. Mismatched response.]  
    at sun.reflect.NativeConstructorAccessorImpl.newInstance0(Native Method)  
    ...  
    at com.sun.proxy.$Proxy46.getContainerStatuses(Unknown Source)  
    at org.apache.hadoop.yarn.client.api.impl.NMClientImpl.getContainerStatus(NMClientImpl.java:339)  
    ...
```

SaslException



appAttemptId = 2



levelDB

oldMasterKeys = {1 :  }

Client
Previous 


oldMasterKeys = {1 :  }

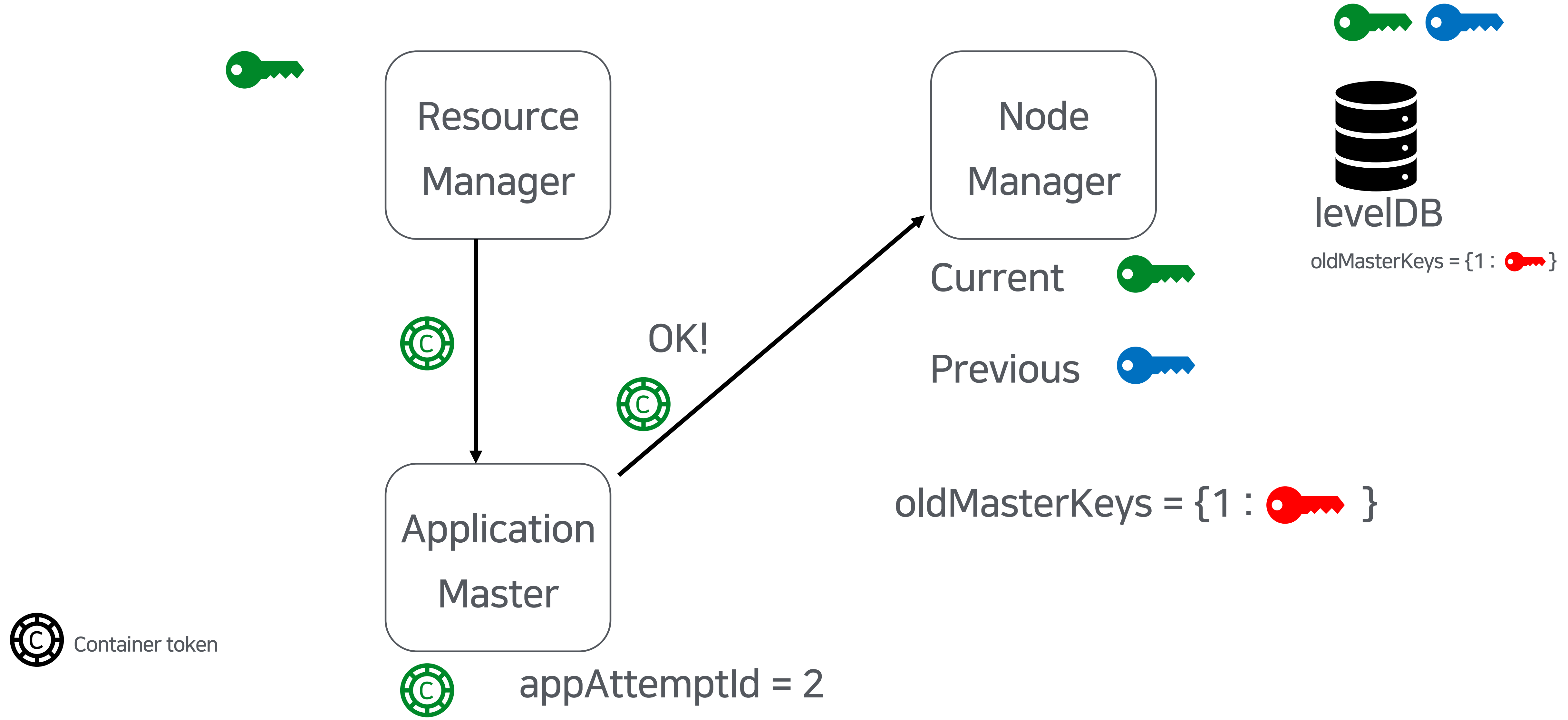


3.3.8 YARN Service AM 에서 getContainerStatuses 실패

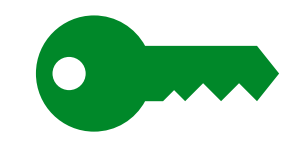
YARN-10969 해결 아이디어

- currentMasterKey 혹은 previousMasterKey 와 key id 가 일치한다면, oldMasterKeys 를 업데이트 하자.

3.3.8 YARN Service AM 에서 getContainerStatuses 실패

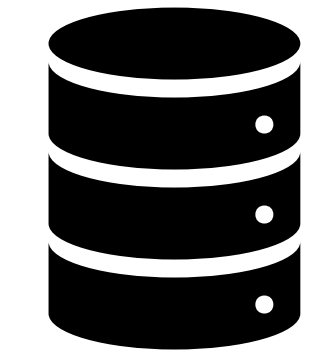
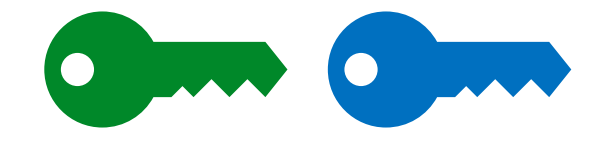


3.3.8 YARN Service AM 에서 getContainerStatuses 실패



Resource Manager

Node Manager



levelDB 업데이트!

oldMasterKeys = {2 :

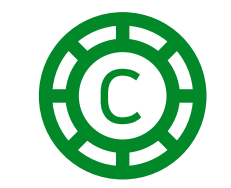
Current

Previous

업데이트!

oldMasterKeys = {2 :

Application Master



appAttemptId = 2



3.3 Resource Manager

여러 이슈들을 결국 해결 (Hadoop 3.1.2)

- RM failover 후 컨테이너 할당 정체 (별도 패치)
- refreshQueues 수행 후 Deadlock (YARN-9163)
- RM Memory leak (YARN-10467, YARN-10895)
- non-exclusive 자원 제어 (YARN-10899)
- 자원 preemption 실패 (YARN-10892)
- YARN Service AM 문제 (YARN-10968, YARN-10969)
- ...

4. 클러스터 안정화, 앞으로의 방향

4.1 Reminds

힘들었던 상반기

- 자원 할당이 안되고,
- 작업도 지연되고,
- RM 메모리는 새고 있고,
- Zookeeper, yarn service 불안정..

(insecure => secure) + 버전 업그레이드 (3.1.2)

=> 보다 나은 안정성을 기대했던 것과는 달리

기존에 믿고 있었던 동작, 추가된 기능을 신뢰할 수 없었다.

4.1 Reminds

한숨 돌린 하반기

- 당장 급한 문제들은 모두 해결
- 조금 더 많은 관찰과 분석이 필요한 문제에 집중

4.2 Contributions

버그수정

- HADOOP-17859 DNS Query on DNS Registry got connection timed out at times. (DNS 서버 타임아웃 문제)
- YARN-10892 YARN Preemption Monitor got `java.util.ConcurrentModificationException` when three or more partitions exists (preemption 안되는 문제)
- YARN-10895 ContainerIdPBImpl objects still can be leaked in `RMNodeImpl.completedContainers` (RM 메모리 누수 문제)
- YARN-10968 SchedulingRequests can be wrong when multiple containers stopped at the same time (RM - AM 간 문제)
- YARN-10969 After RM fail-over, `getContainerStatus` fails from ApplicationMaster to NodeManager (NM - AM 간 문제)

4.2 Contributions

기능개선

- HADOOP-17861 improve YARN Registry DNS Server qps
(DNS 서버 성능 개선)
- YARN-10898 Support standalone YARN Service API SERVER
(RM 으로부터 API 서버 분리)
- YARN-10899 control whether non-exclusive allocation is used
(non-exclusive 자원제어 기능 추가)

4.3 What's next?

보다 나은 사용자 경험으로

- 사용자 스스로 한눈에 모든 작업의 상황을 볼 수 있는 화면
- 보다 친절한 알림 시스템
- 숙련자만을 위한 플랫폼이 아닌, 오늘 시작한 사용자도 쉽게 사용할 수 있도록

더욱더 주도적으로

- 최신 하둡을 따라가며 기여도 함께
- 이것이 최선인가?

WE'RE HIRING!

AI & Data Platform 소개

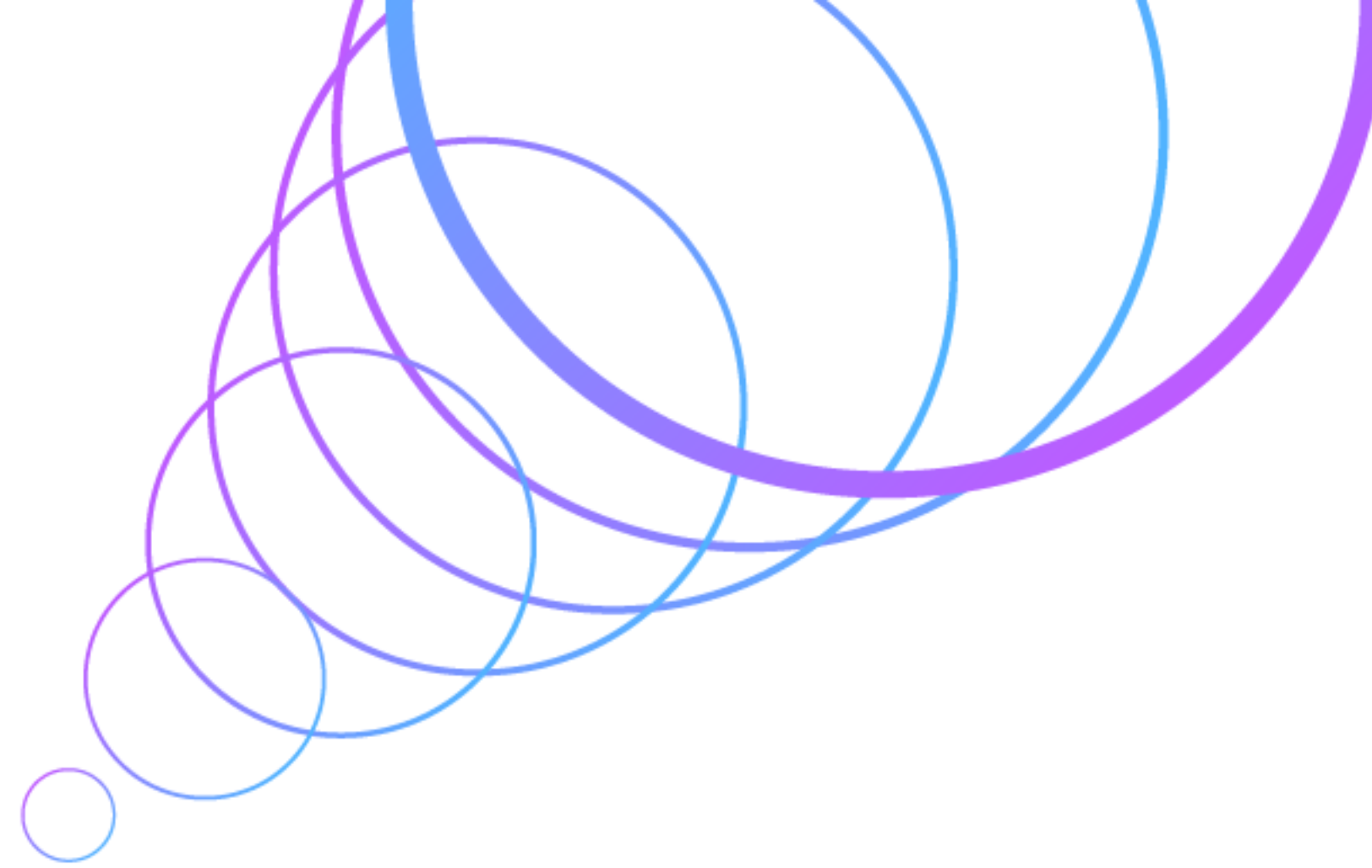
<https://naver-career.gitbook.io/kr/service/search/ai-and-data-platform>

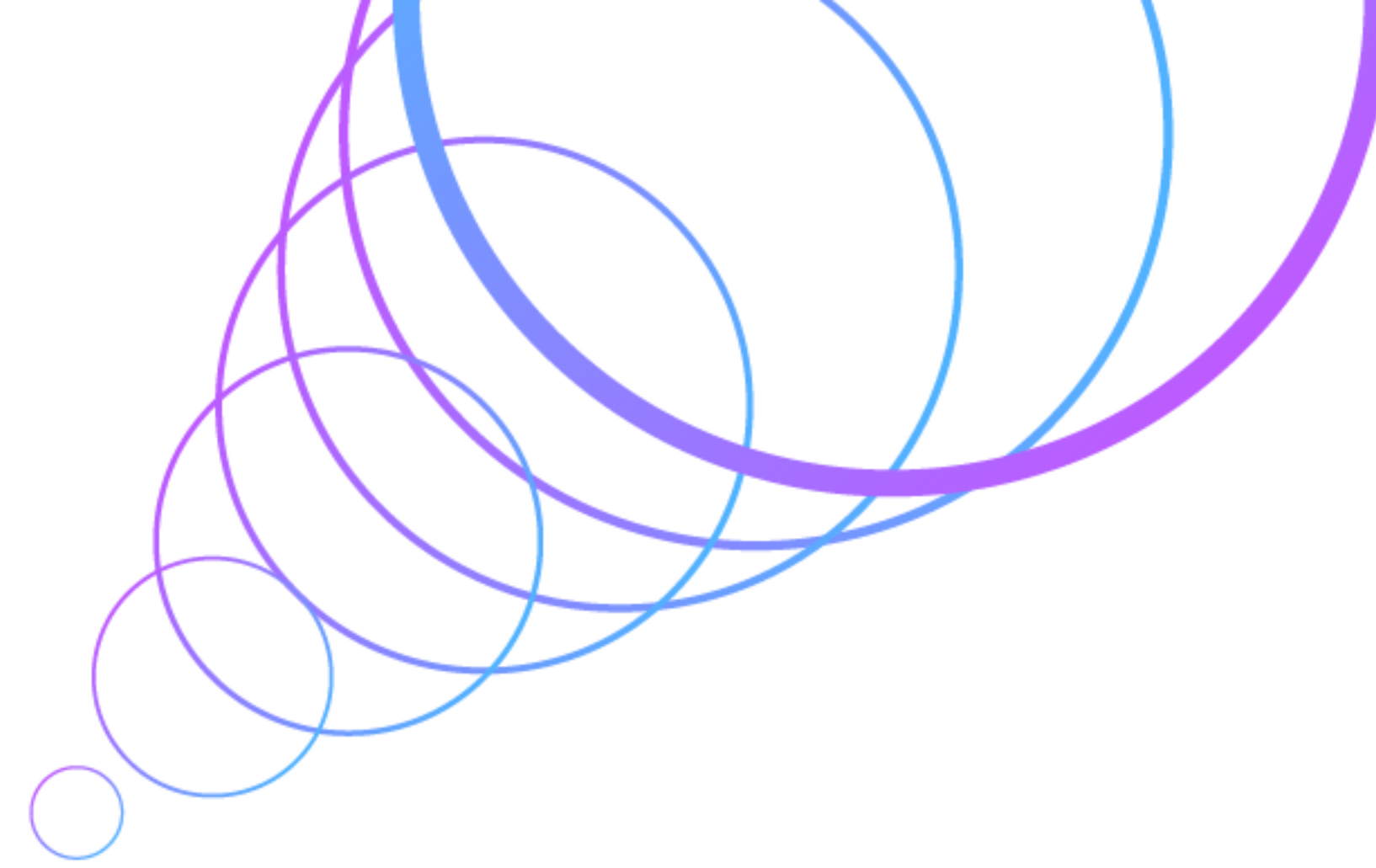
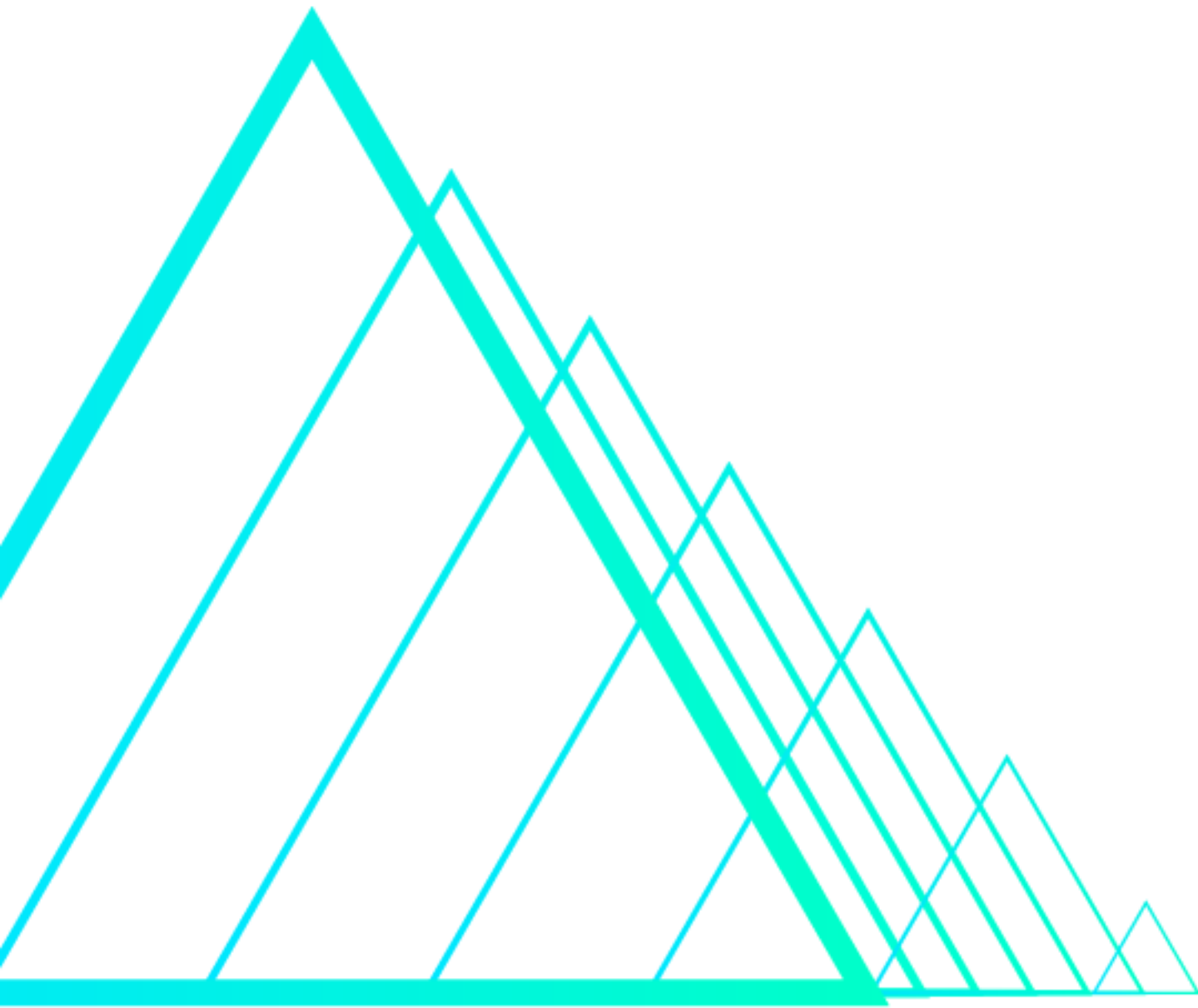
이력서 등록

<https://d2.naver.com/news/7591059>



Q & A





Thank You

