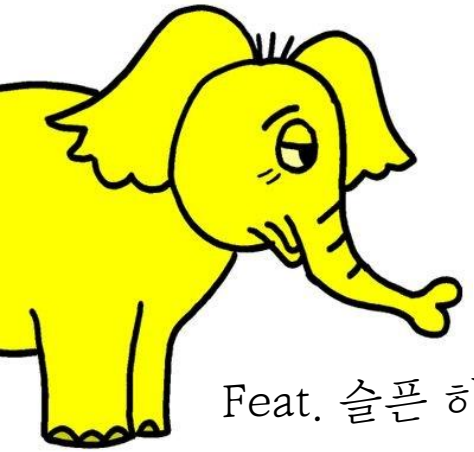
HADOOP week 1

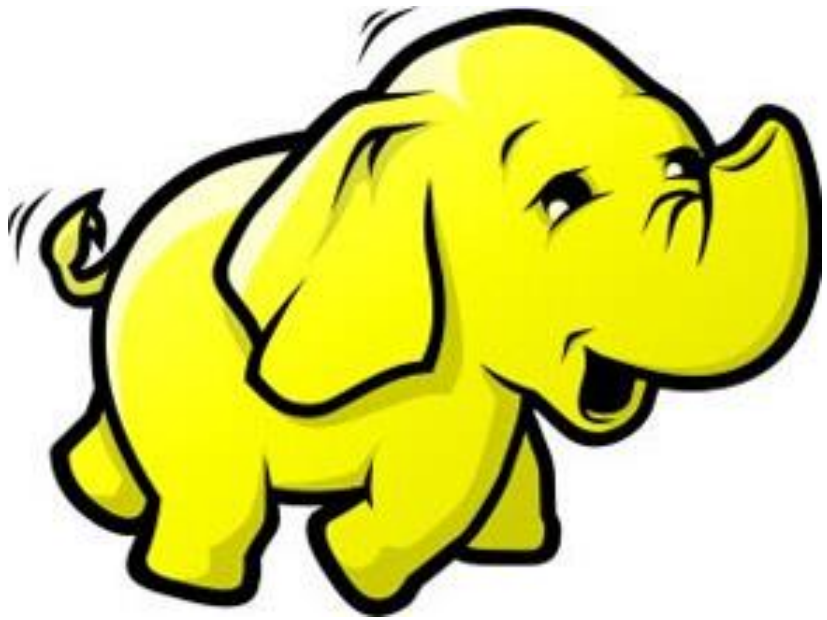
한 솔



Feat. 슬픈 하둡이

목차

1. 분산처리 필요성과 하둡의 등장



1. 1 빅데이터의 시대

- 3V
- Volume
- Velocity
- Variety

1.2 하둡이란?

- 대용량 데이터를 분산 처리할 수 있는, 자바 기반의 오픈소스 프레임워크
- *분산 파일 시스템*인 HDFS에 데이터를 저장하고, *분산 처리 시스템*인 MAP REDUCE를 이용해 데이터 처리

Why Hadoop?

- **저렴한 구축 비용**

  → 소프트웨어 라이센스 비용X, 필요한 만큼 리눅스 서버 추가하면 됨

- **비용 대비 빠른 데이터 처리**

  → 여러 대의 서버에 데이터를 저장하고 데이터가 저장된 각 서버에서 동시에 데이터 처리

- **장애를 대비한 특성**

  → 데이터 복제본을 저장하므로 데이터의 유실, 장애 발생시 데이터 복구 가능

하둡 에코시스템


워크 플로우
(Oozie,
Airflow,
Azkaban,
NiFi)
데이터
시각화
(Zeppelin)
대화형 질의 처리
(Impala, Presto)
데이터 웨어하우스
(Hive, Tajo)
스크립트 처리
(Pig)
머신러닝
(Mahout)
인메모리 처리
(Spark)
분산 데이터 배치 처리
(MapReduce)
분산 클러스터 리소스 관리
(YARN, Mesos)
분산 코디네이터
(Zookeeper)
데이터
직렬화
(Avro,
Thrift)
분산 데이터베이스
(HBase)
컬럼 기반 스토리지
(Kudu)
분산 데이터 저장
(HDFS)
스트리밍 데이터 수집
(Chuckwa, Flume, Scribe)
DBMS 데이터 수집
(Sqoop, Hiho)
분산 메세지 처리
(Kafka)

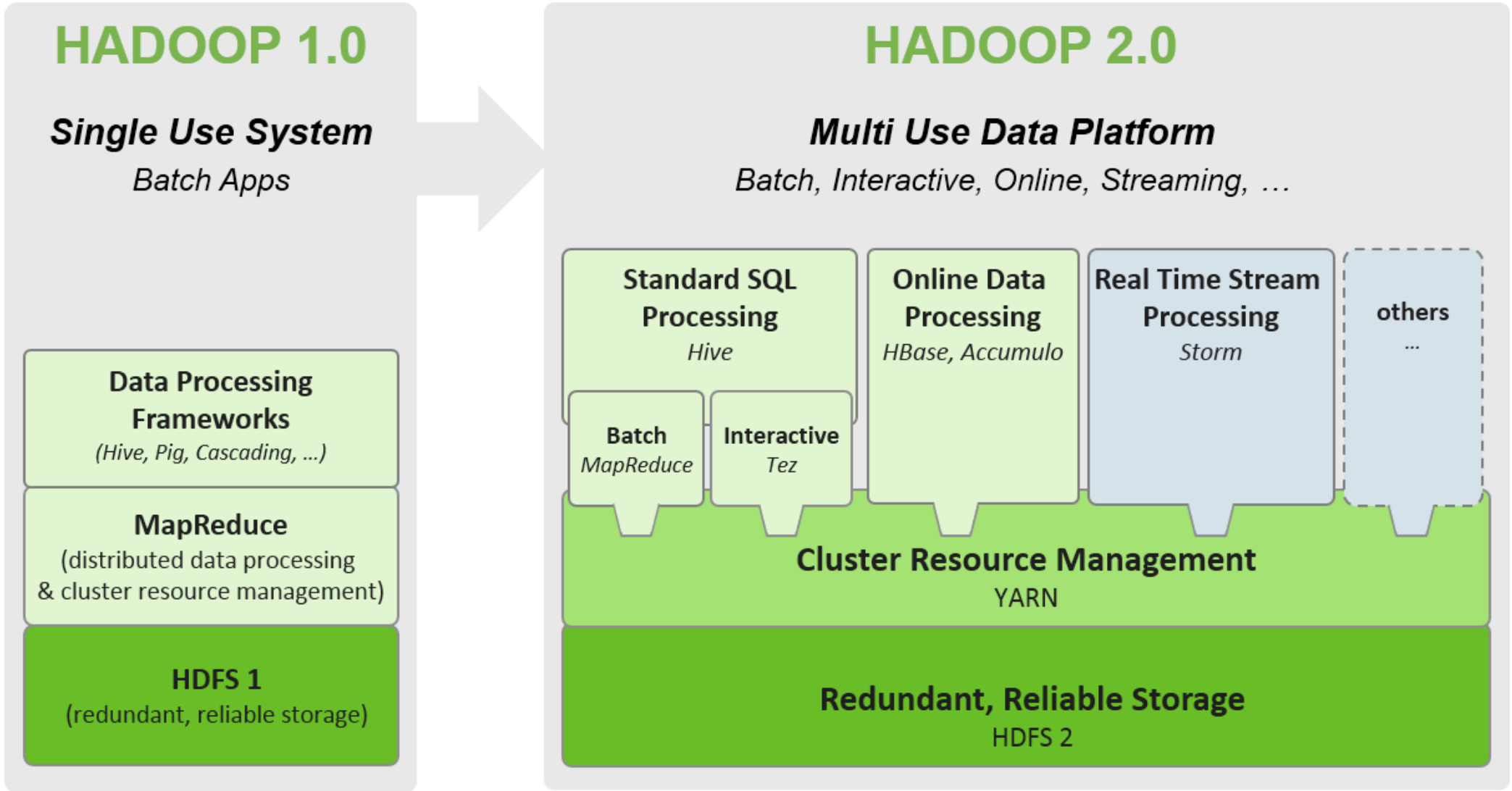
2. 하둡2.0 소개

2.1 하둡2 등장배경

- 하둡을 중심으로 한 에코시스템이 활성화되면서 다양한 방법으로 데이터를 저장하고 분석할 수 있게 됨.
- BUT!!
- –〉 리소스 관리
- –〉 안정성 문제

2.2 하둡2 특징

- YARN
- 네임노드 고가용성
- HDFS 페더레이션
- HDFS 스냅샷



Interact with all data in multiple ways simultaneously

3. HDFS

1) HDFS란?

- Hadoop Distributed File System
- 수십 TB, PB 이상의 대용량 파일을 분산된 서버에 저장하고, 많은 클라이언트가 저장된 데이터를 빠르게 처리할 수 있게 설계

2) HDFS의 목표

장애 복구

- HDFS 는 장애를 신속히 감지하고 대처할 수 있게 설계됨
- HDFS는 데이터를 저장하면, 복제 데이터도 함께 저장되어 데이터 유실 방지함
- 분산 서버 간에 주기적으로 상태를 체크해 빠른 시간에 장애를 인지하고, 대처 가능하게 함.

HDFS

스트리밍 방식의 데이터 접근

- HDFS에 파일을 저장하거나, 저장된 파일을 조회하려면 스트리밍 방식으로 데이터에 접근
- HDFS는 기존의 파일 시스템과는 달리 배치 작업에 적합하도록 설계돼 있고, 낮은 데이터 접근 지연시간보다는 높은 데이터 처리량에 중점을 둠.

HDFS

대용량 데이터 저장

- HDFS는 하나의 파일이 기가바이트에서 테라바이트 이상의 사이즈로 저장될 수 있게 설계됨
- 높은 데이터 전송 대역폭과 하나의 클러스터에서 수백 대의 노드를 지원할 수 있어야 함.
- 하나의 인스턴스에서는 수백만 개 이상의 파일을 지원함

HDFS

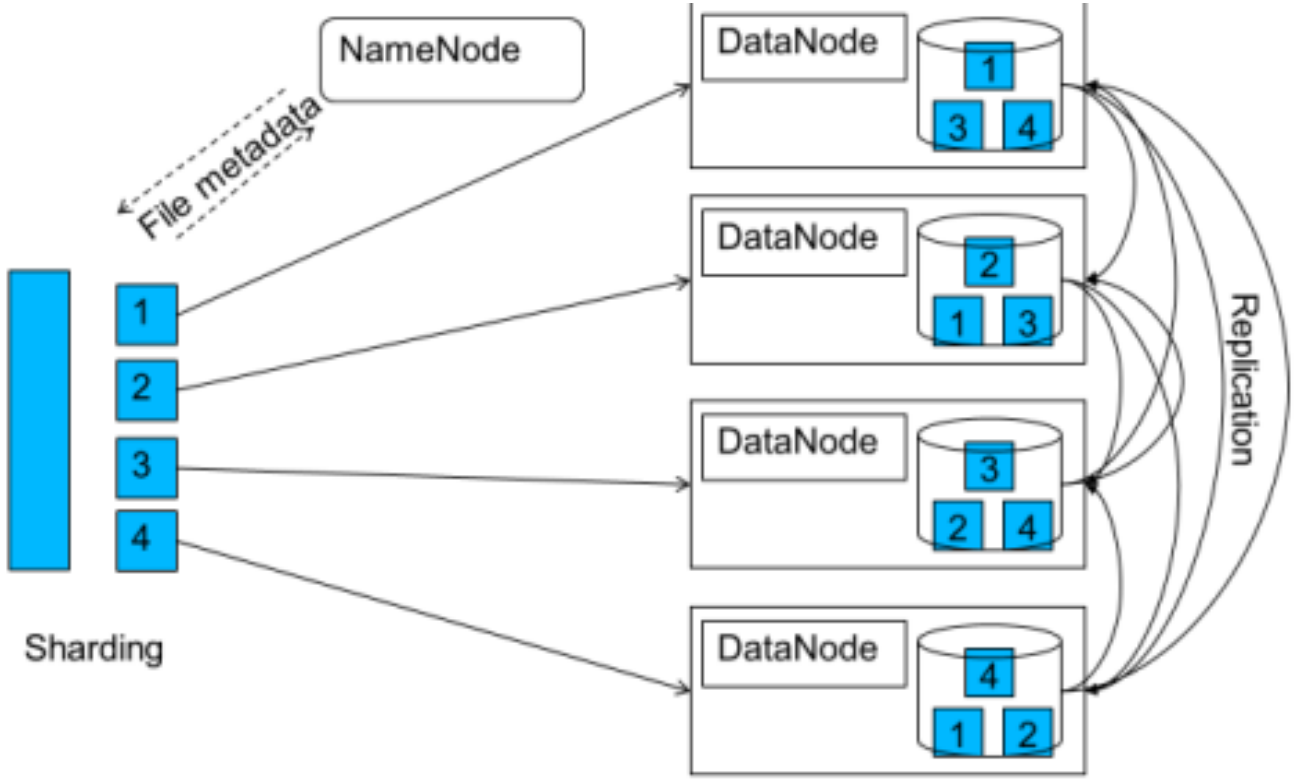
데이터 무결성

- HDFS는 한 번 저장한 데이터는 추가로 수정 불가하고, 읽기만 가능하게 하여 데이터 무결성을 유지
- 데이터 수정은 불가하지만 파일 이동, 삭제, 복사 가능하도록 인터페이스 제공
- 하둡 2.0 알파버전부터는 HDFS에 저장된 파일에 append가 제공

HDFS

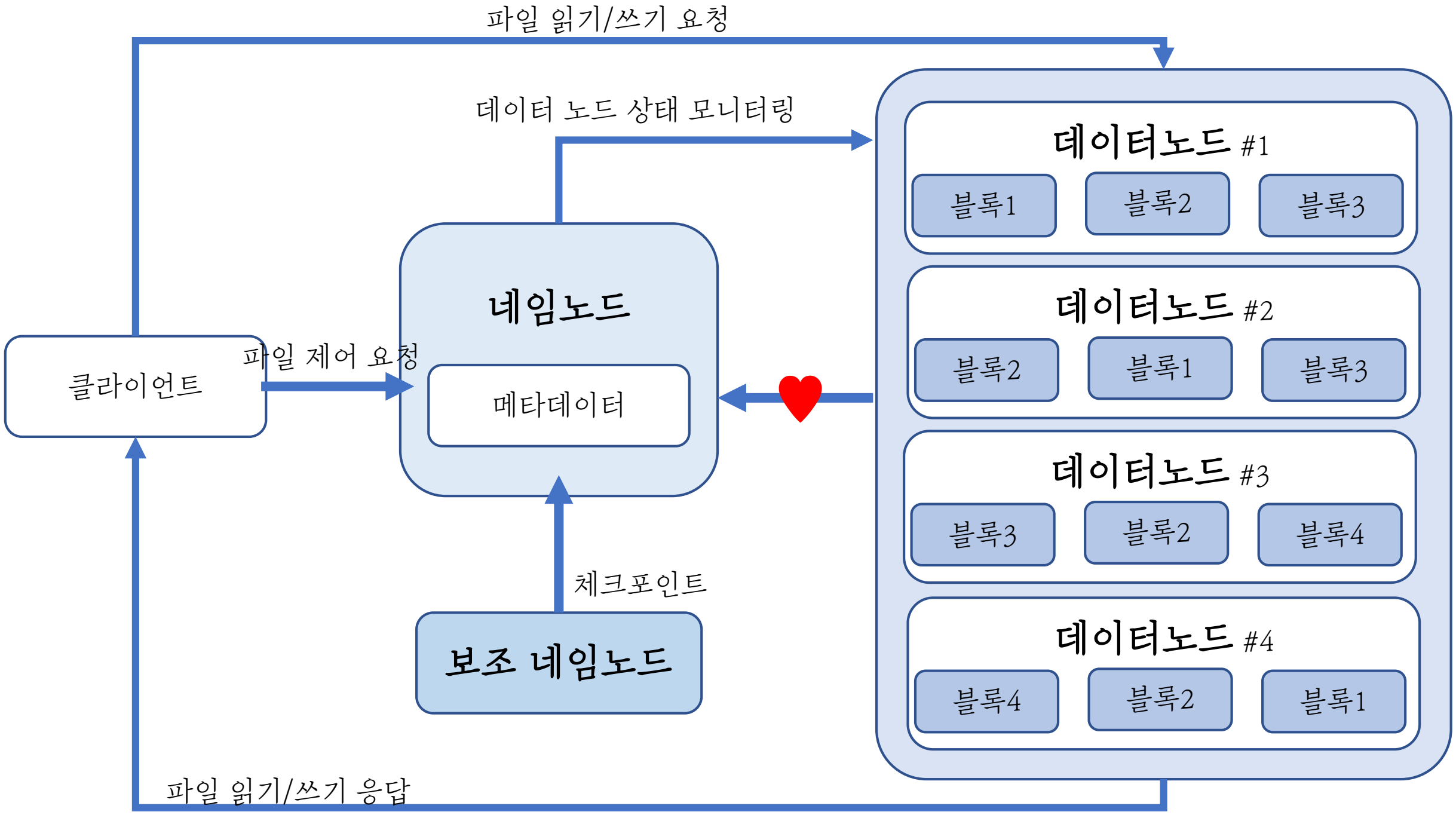
3) HDFS 아키텍쳐 – 블록 구조의 파일 시스템

- HDFS에 저장하는 파일은 특정 크기의 블록으로 나눠져 **분산된 서버에** 저장 ➔ 여러 개의 블록은 여러 서버에 나눠서 저장. 따라서 대용량 데이터가 저장 가능



마스터-슬레이브 아키텍처

- 마스터 서버 : 네임노드
- 슬레이브 서버 : 데이터노드


파일 읽기/쓰기 요청
데이터 노드 상태 모니터링
네임노드
메타데이터
클라이언트
파일 제어 요청
체크포인트
보조 네임노드
데이터노드 #1
블록1
블록2
블록3
데이터노드 #2
블록2
블록1
블록3
데이터노드 #3
블록3
블록2
블록4
데이터노드 #4
블록4
블록2
블록1
파일 읽기/쓰기 응답

네임노드

1) **메타데이터 관리** : 파일 시스템 이미지 + 파일에 대한 블록 매핑 정보
2) **데이터노드 모니터링** : 하트비트를 이용해 데이터노드 실행 상태, 용량 모니터링
3) **블록 관리** : when 데이터노드 장애발생, 용량부족
4) **클라이언트 요청 접수** : 클라이언트가 HDFS에 접속하려면 반드시 네임노드에 먼저 접속

데이터노드

- 클라이언트가 HDFS에 저장하는 파일을 로컬디스크에 유지
- 네임노드에게 3초마다 하트비트 전송

보조네임노드(Secondary namenode)

- 네임노드는 메모리에서 메타데이터를 처리하므로 서버가 재부팅될 경우 모든 메타데이터가 유실 될 수 있다.

→ HDFS는 editslog와 fsimage 두 개의 파일 생성

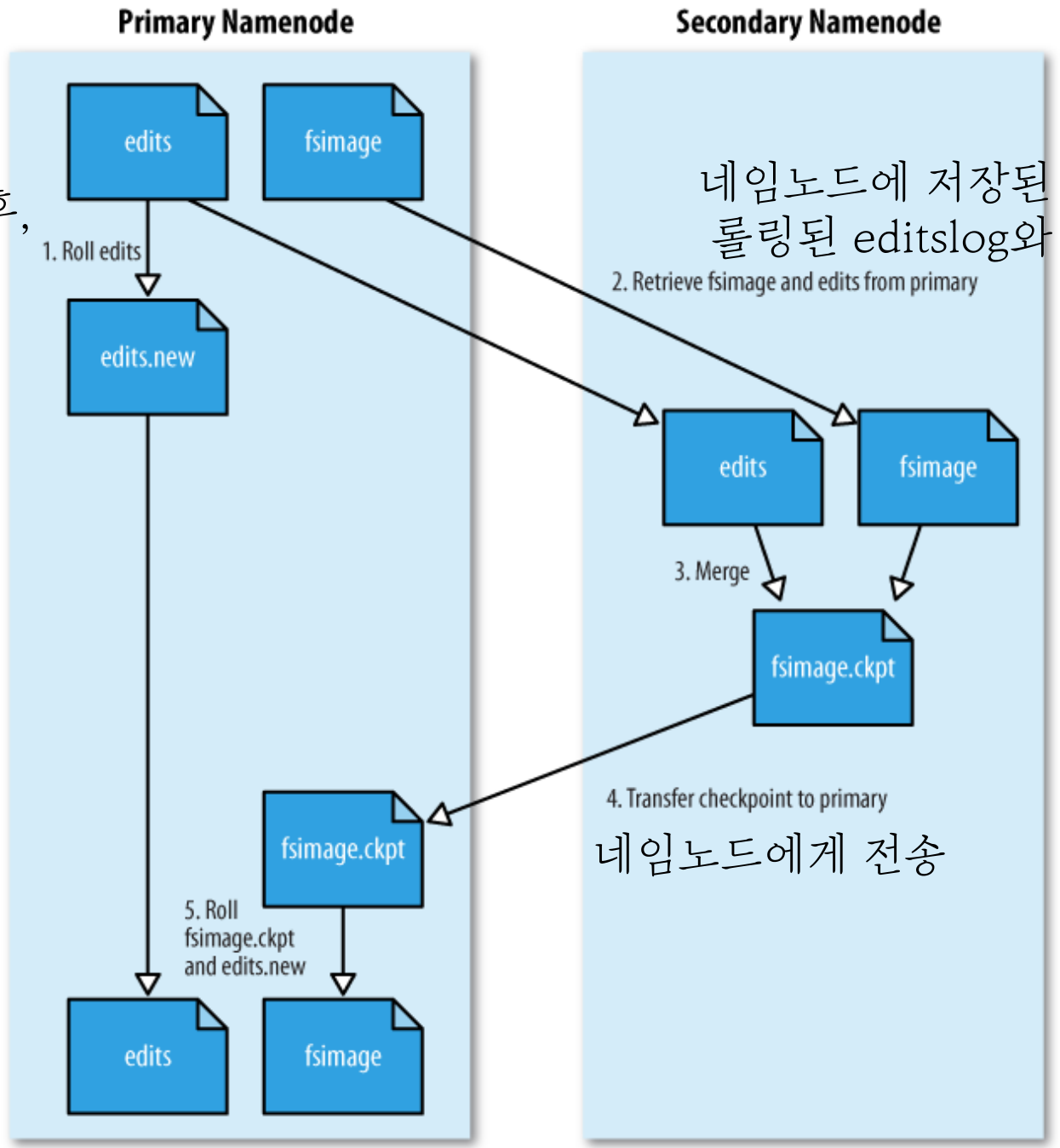
1) editslog : HDFS의 모든 변경 이력 저장

2) fsimage : 메모리에 저장된 메타데이터의 파일 시스템 이미지(파일명, 디렉터리, 크기, 권한)를 저장

- 네임 노드가 구동되는 경우 위 두 파일을 사용한다.
- **Secondary namenode : 주기적으로 네임노드의 fsimage 갱신**

- 1) 네임노드가 구동되면 fsimage와 editslog 조회
- 2) 메모리에 fsimage를 로딩해 파일 시스템 이미지 생성
- 3) **editslog에 기록된 변경이력을 로딩된 파일 시스템 이미지에 적용**
- 4) 적용된 파일 시스템 이미지를 이용해 fsimage 파일 갱신
- 5) editslog 초기화

기존 editslog롤링 후,
editslog.new생성



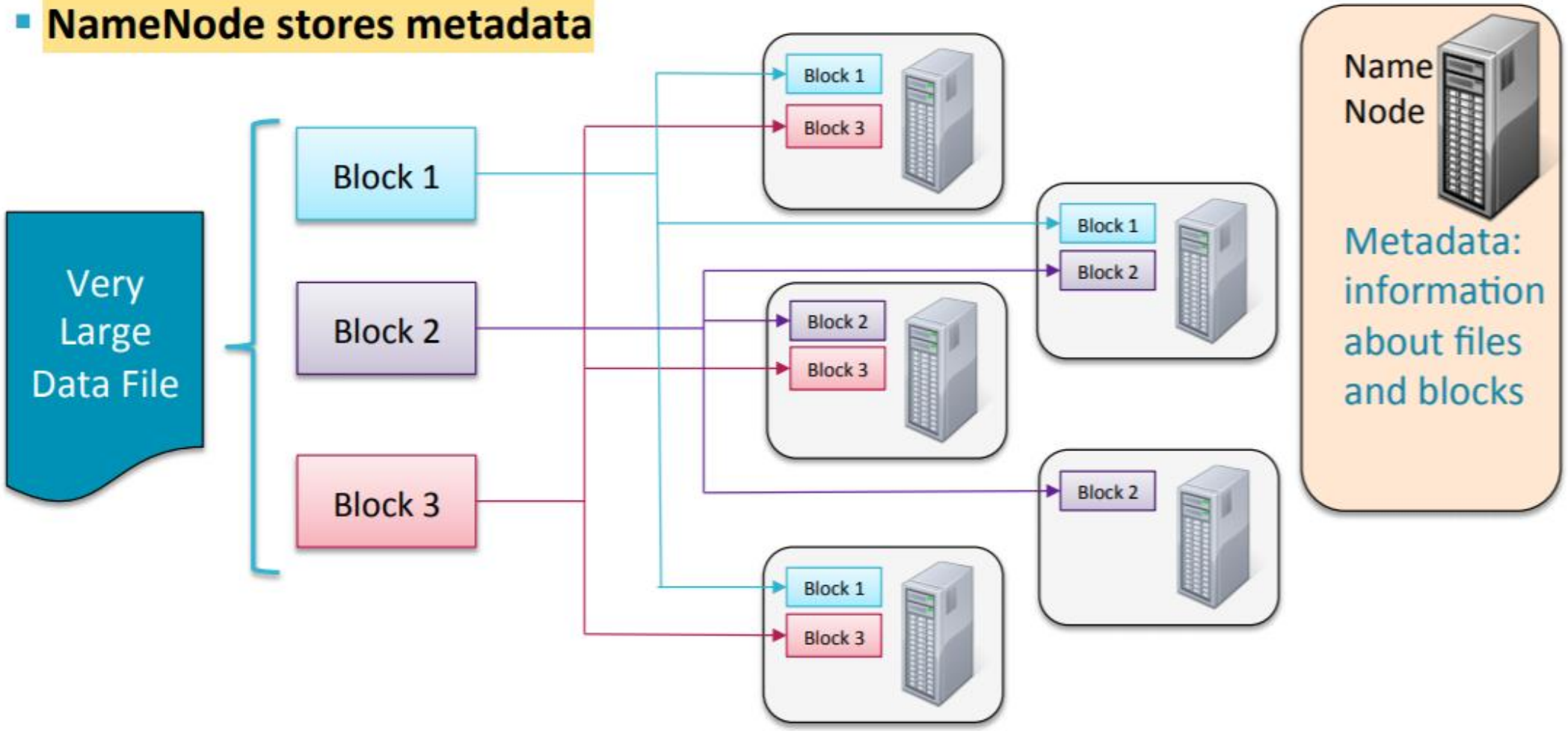
네임노드에 저장된
롤링된 editslog와 fsimage다운로드

네임노드에게 전송

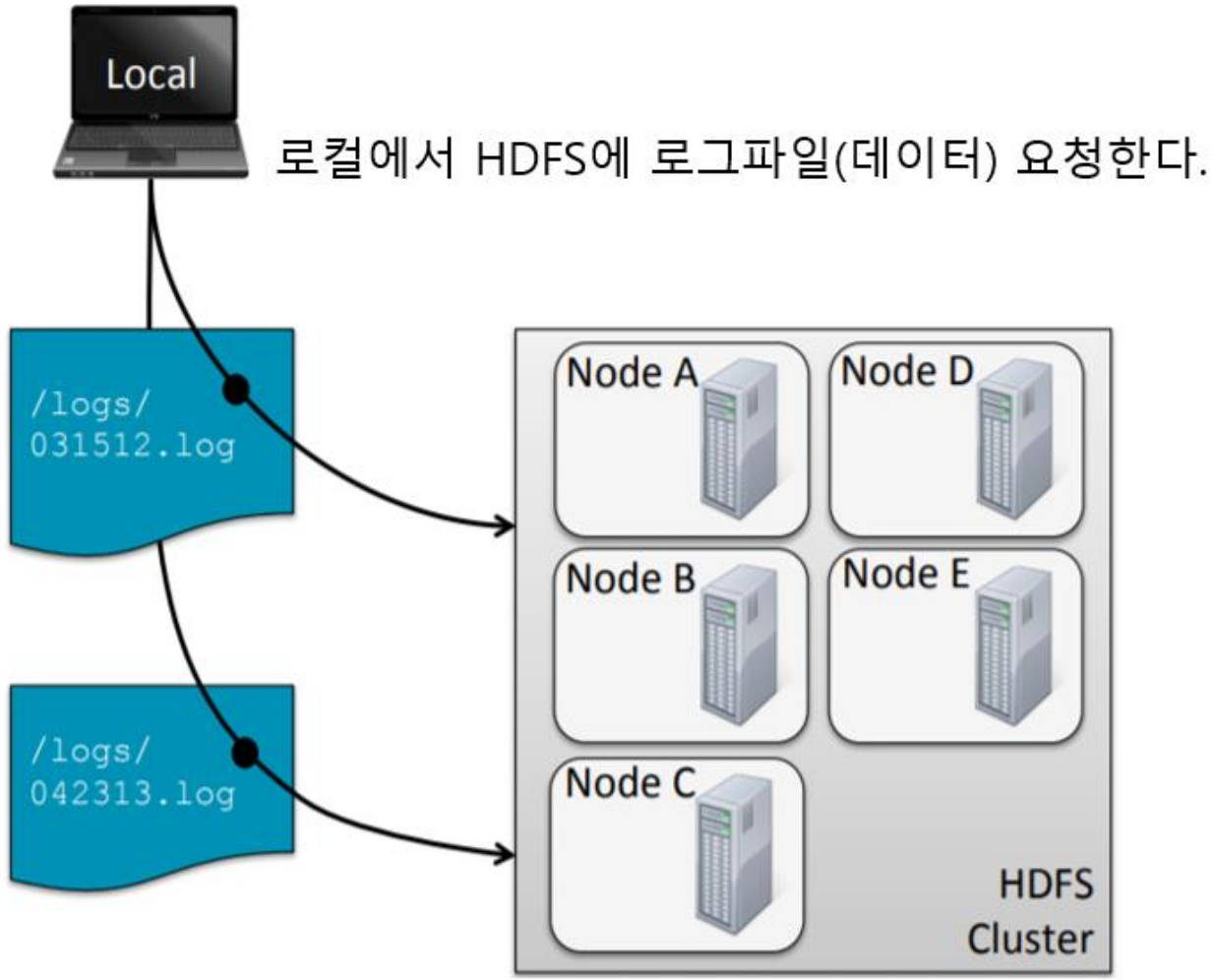
변경 이력 적용 후 새로운
fsimage.ckpt생성

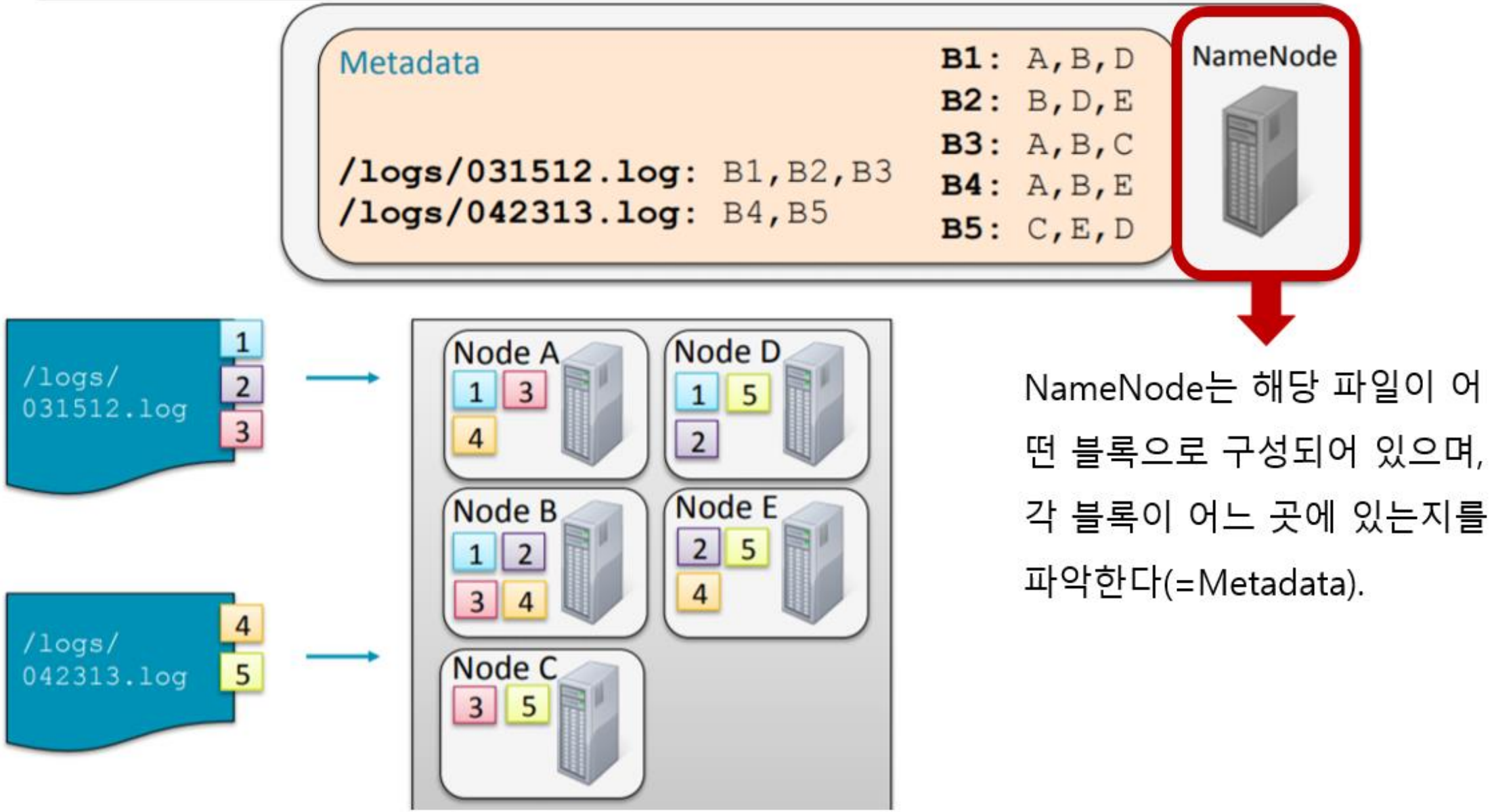
4) HDFS의 파일 저장

- **NameNode stores metadata**

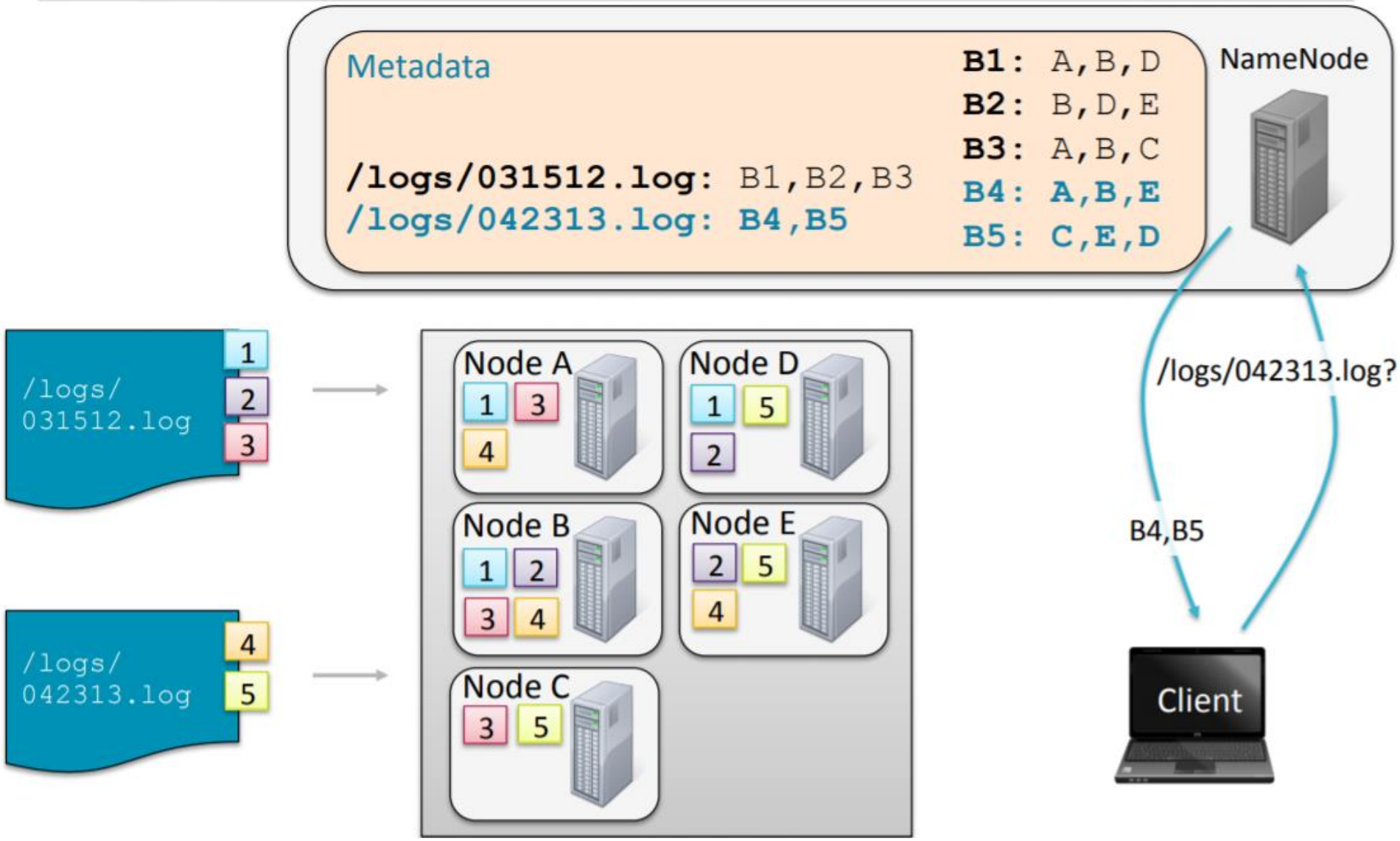


5) HDFS의 파일 읽기

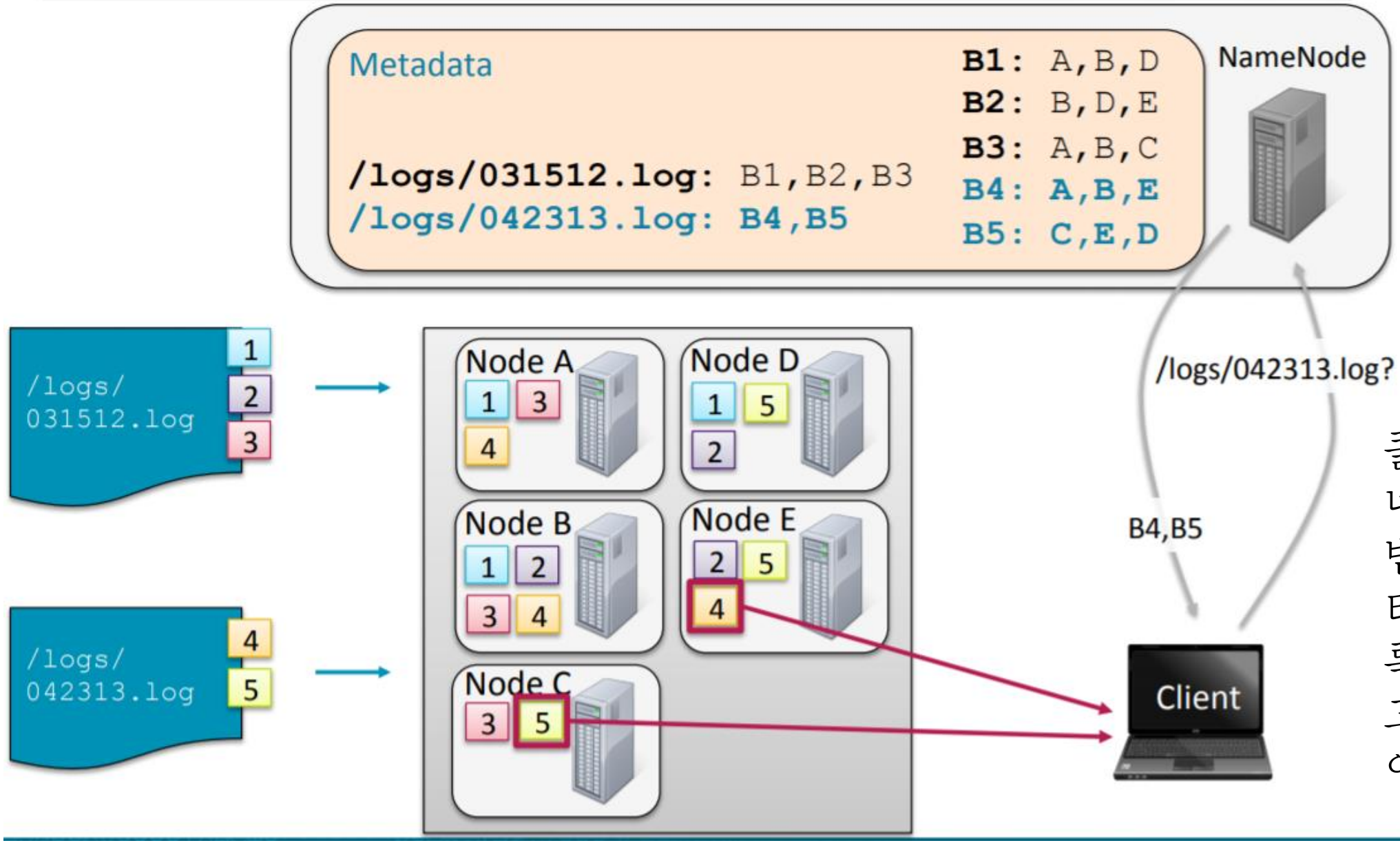

Local
로컬에서 HDFS에 로그파일(데이터) 요청한다.
/logs/
031512.log
/logs/
042313.log
Node A
Node D
Node B
Node E
Node C
HDFS
Cluster



NameNode는 해당 파일이 어떤 블록으로 구성되어 있으며, 각 블록이 어느 곳에 있는지를 파악한다(=Metadata).



네임노드는 클라이언트가 요청한 파일이 어떤 블록으로 이루어져 있는지, 각 블록이 어느 노드에 저장되어 있는지 위치를 알려줌



클라이언트는 네임노드에게 받은 메타데이터를 통해서 블록들을 가져오고, 이를 조합하여 파일을 구성

6) HDFS 명령어

- 파일 목록 보기 –ls, lsr

  hdfs dfs -ls [디렉터리]
- 파일 용량 확인 –du, dus

  hdfs dfs -du [디렉터리]
- 파일 내용 보기 -cat, text

  hdfs dfs -cat [파일]
- 디렉터리 생성 -mkdir

  hdfs dfs -mkdir [디렉터리]

6) HDFS 명령어

- 파일 복사 -put(copyFromLocal), get(copyToLocal), getmerge
  hdfs dfs -put [로컬디렉터리 | 파일] [목적지 디렉터리 | 파일]
  hdfs dfs -get [소스디렉터리 | 파일] [로컬 디렉터리 | 파일]
- 파일 이동 - mv, moveFromLocal
- 파일 삭제 –rm
- 디렉터리 삭제 –rmr
- 카운트값 조회 -count
- 파일 마지막 내용 확인 –tail

잠깐..앞으로.. 하둡2 특징

- YARN
- 네임노드 고가용성(HA)
- HDFS 퍼데레이션
- HDFS 스냅샷

7) 네임노드 HA(고가용성)

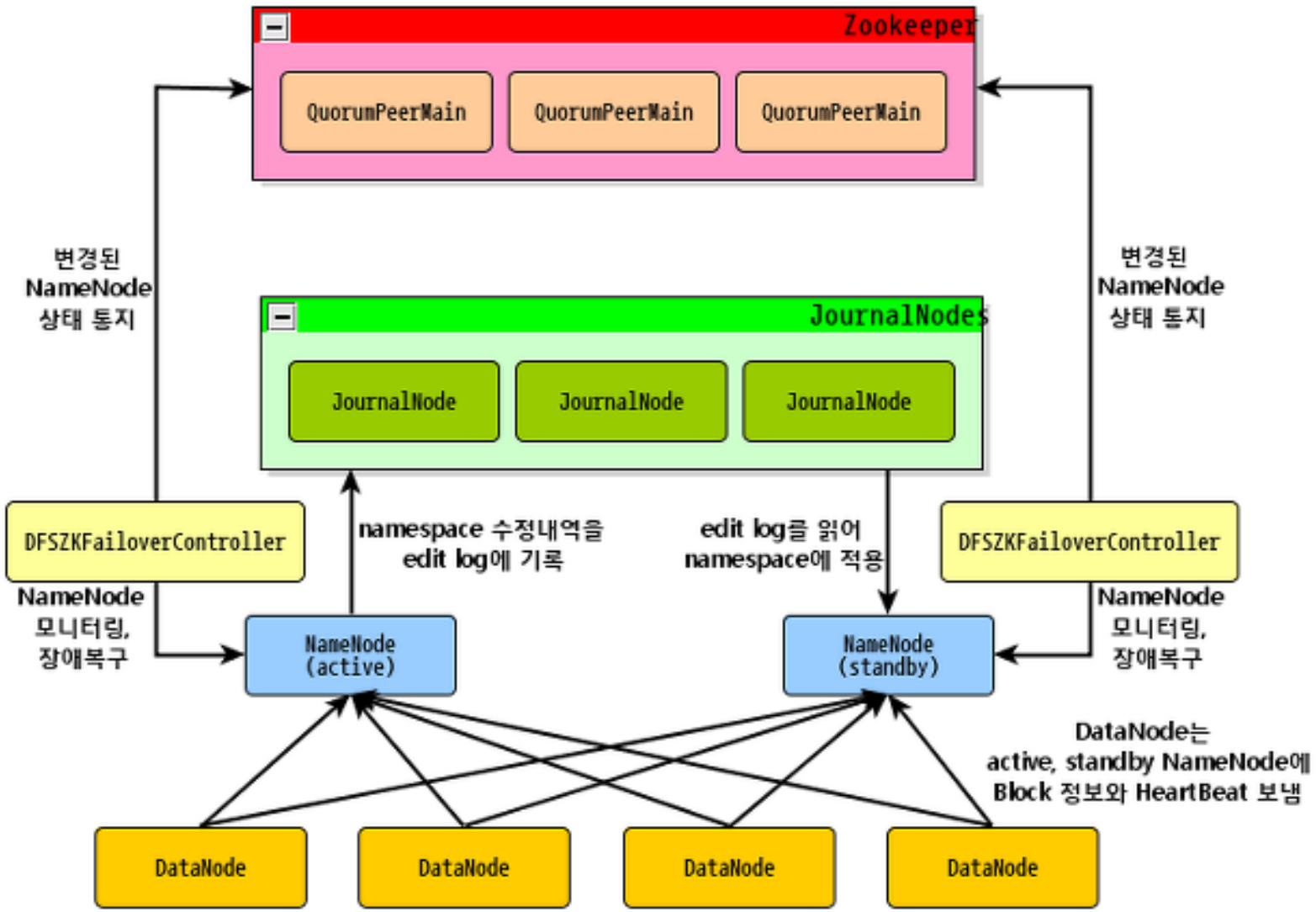
- 하둡 2.0이전 : 네임노드는 **SPOF(단일 고장점**, 시스템 구성 요소 중에서, 동작하지 않으면 전체 시스템이 중단되는 요소)
- 네임노드가 HDFS의 모든 데이터의 메타 정보를 관리하므로 네임노드 장애 발생 시 큰 문제

- ➔ 2.0부터 네임노드의 **고가용성** 지원

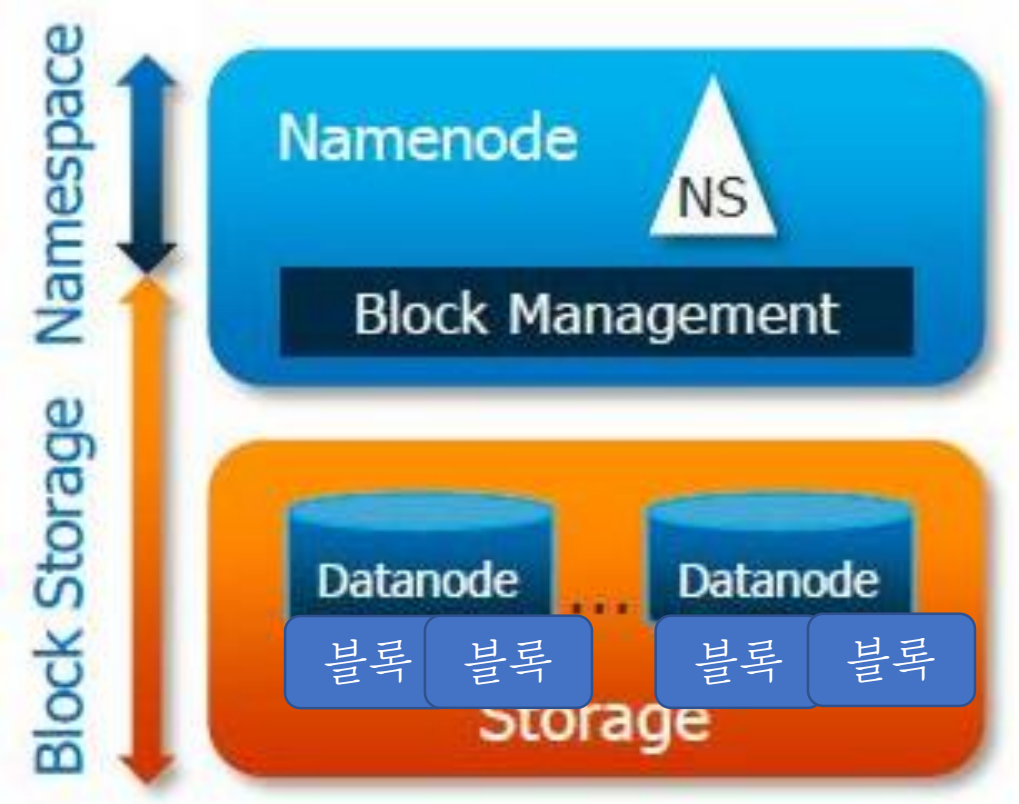
* 가용성 = 시스템 장애 발생 후 정상으로 돌아오는 상태를 분석하는 척도

따라서, 고가용성 = 고장 없이 오랫동안 지속적으로 정상 운영 가능

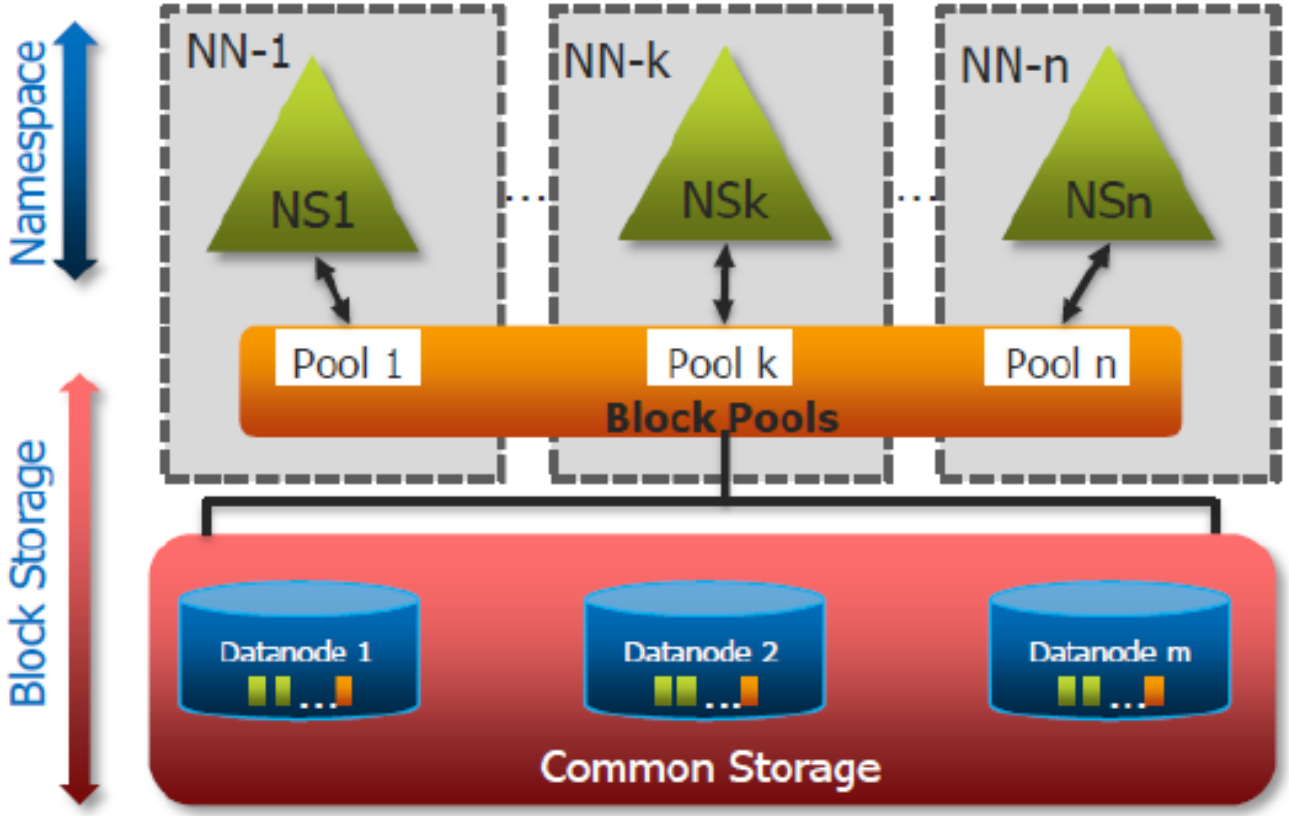
7) 네임노드 HA(고가용성)



8) HDFS 페더레이션



기존 하둡1 HDFS 아키텍처

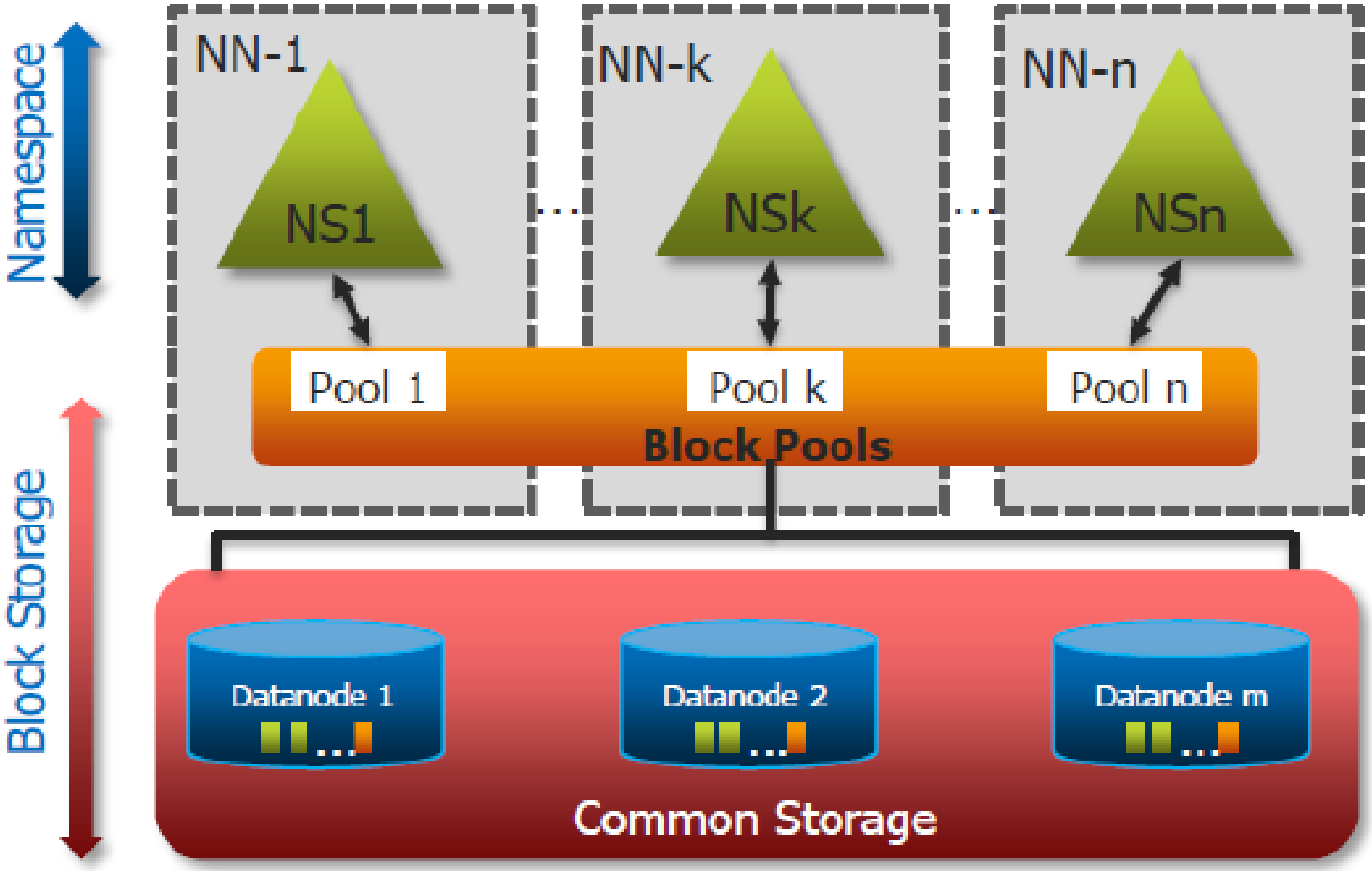


HDFS 페더레이션 아키텍처

하둡1 HDFS 아키텍처의 문제점

- **네임노드 확장 문제** : 데이터노드와 달리 네임노드는 수평적 확장 불가능 ➔ 메모리 크기의 제한
- **동일 네임노드 사용 문제** : 모든 하둡 클라이언트가 하나의 동일한 네임노드 사용
- **네임스페이스와 블록 관리의 과도한 코드 공유** : 두 개의 서로 다른 기능이 코드로 연결되어 있어 복잡도가 높고 해당 기능 확장이 어려움

하나의 하둡 클러스터에 여러 개의 네임노드 구동!!



9) HDFS 스냅샷

- 스냅샷을 이용해 언제든지 복원 가능한 시점으로 HDFS 복원 가능
- 스냅샷에 저장된 전체 파일 시스템의 상태 복원 or 특정 데이터만 선택해서 복원
- 특정 디렉터리에 대해 스냅샷 설정 → 해당 디렉터리는 스냅샷 디렉터리가 됨.

/wikibooks 디렉터리에 대한 스냅샷 생성 예시

1) 스냅샷 허용하도록 권한 설정

```
hdfs dfsadmin -allowSnapshot /wikibooks
```

2) 스냅샷 생성

```
hdfs dfs -createSnapshot /wikibooks wiki_snapshot
```

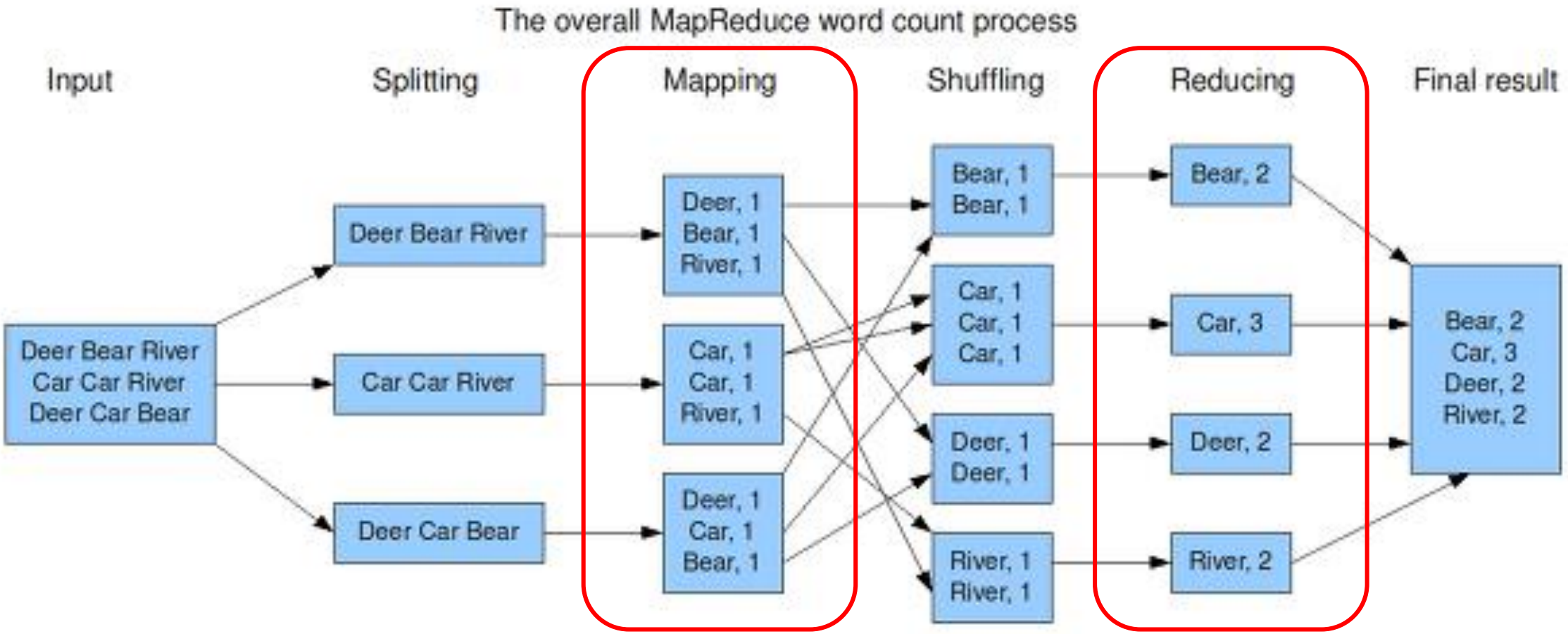
잠깐..다시..앞으로.. 하둡2 특징

- YARN
- 네임노드 고가용성
- HDFS 퍼데레이션
- HDFS 스냅샷

4. MAP REDUCE

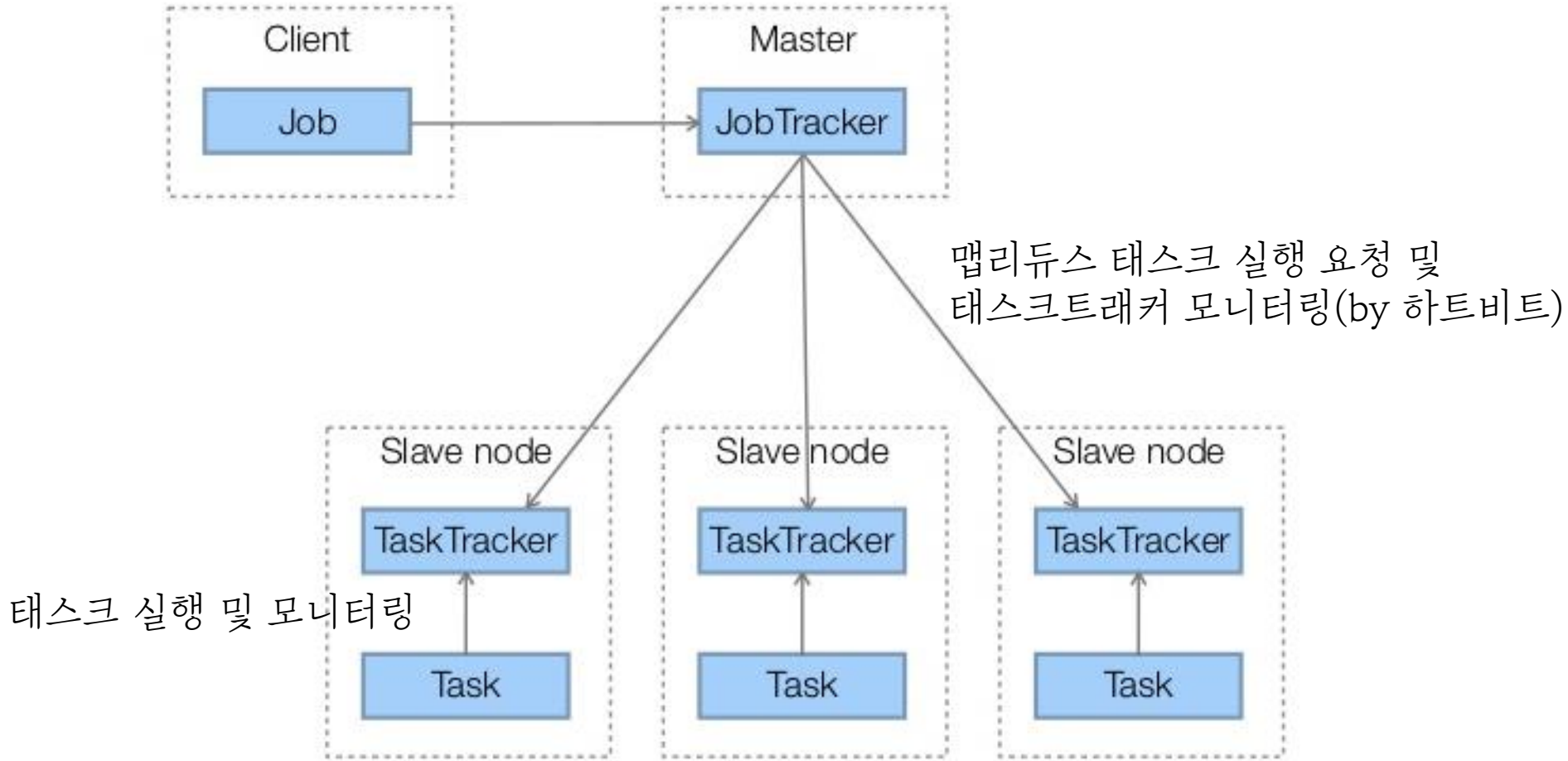
맵리듀스

- HDFS에 저장된 파일을 분석할 수 있도록 해주는 프레임워크
- 맵 + 리듀스
- 맵 : 입력 파일을 한 줄씩 읽어서 데이터 변형
- 리듀스 : 맵의 결과 데이터를 집계

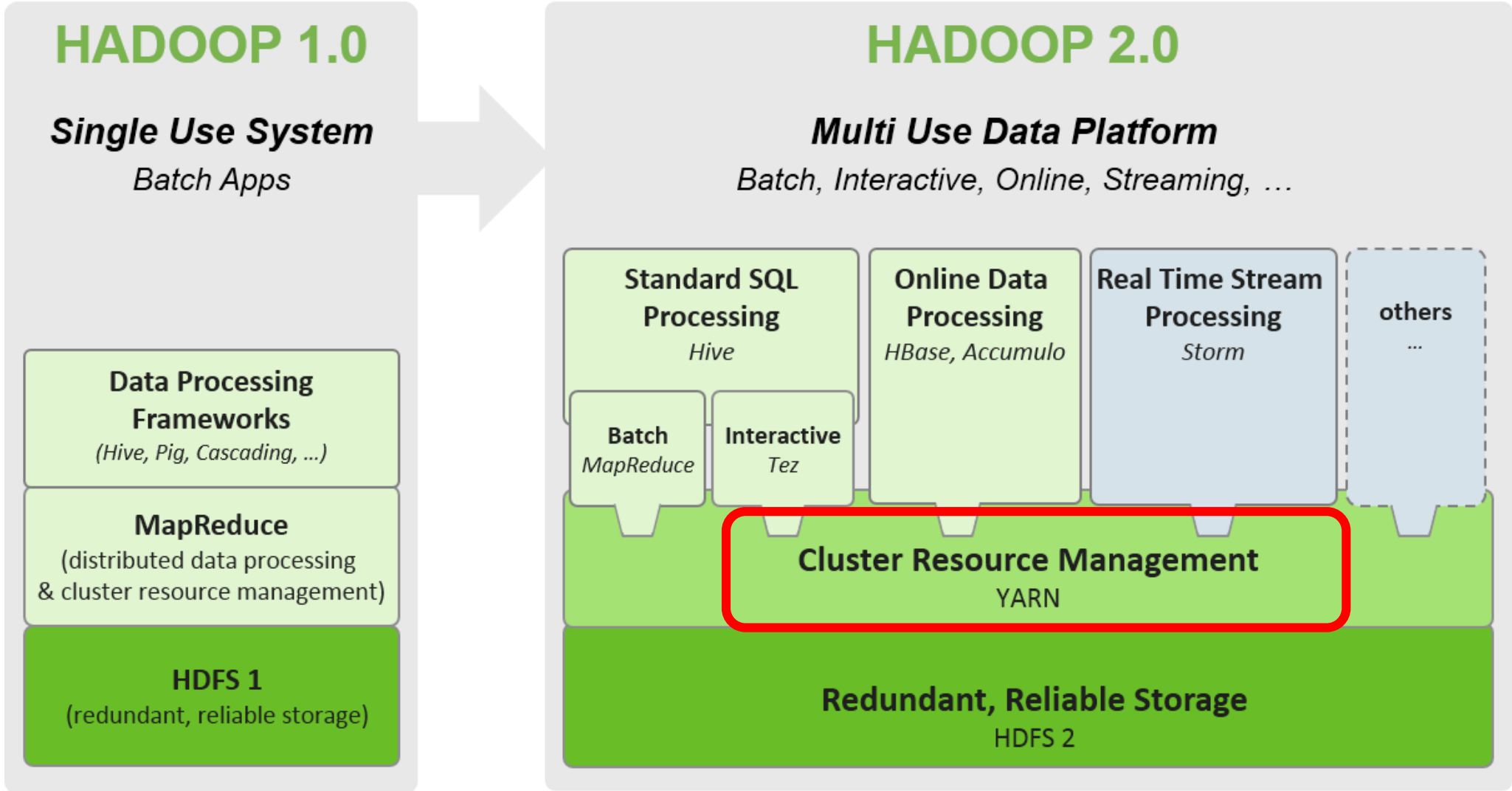

The overall MapReduce word count process
Input
Splitting
Mapping
Shuffling
Reducing
Final result
Deer Bear River
Car Car River
Deer Car Bear
Deer Bear River
Car Car River
Deer Car Bear
Deer, 1
Bear, 1
River, 1
Car, 1
Car, 1
River, 1
Deer, 1
Car, 1
Bear, 1
Bear, 1
Bear, 1
Car, 1
Car, 1
Car, 1
Deer, 1
Deer, 1
River, 1
River, 1
Bear, 2
Car, 3
Deer, 2
River, 2
Bear, 2
Car, 3
Deer, 2
River, 2

맵리듀스 시스템 구성

- 마스터 역할을 하는 **잡트래커** + 슬레이브 서버 역할을 하는 **태스크트래커**



5. YARN

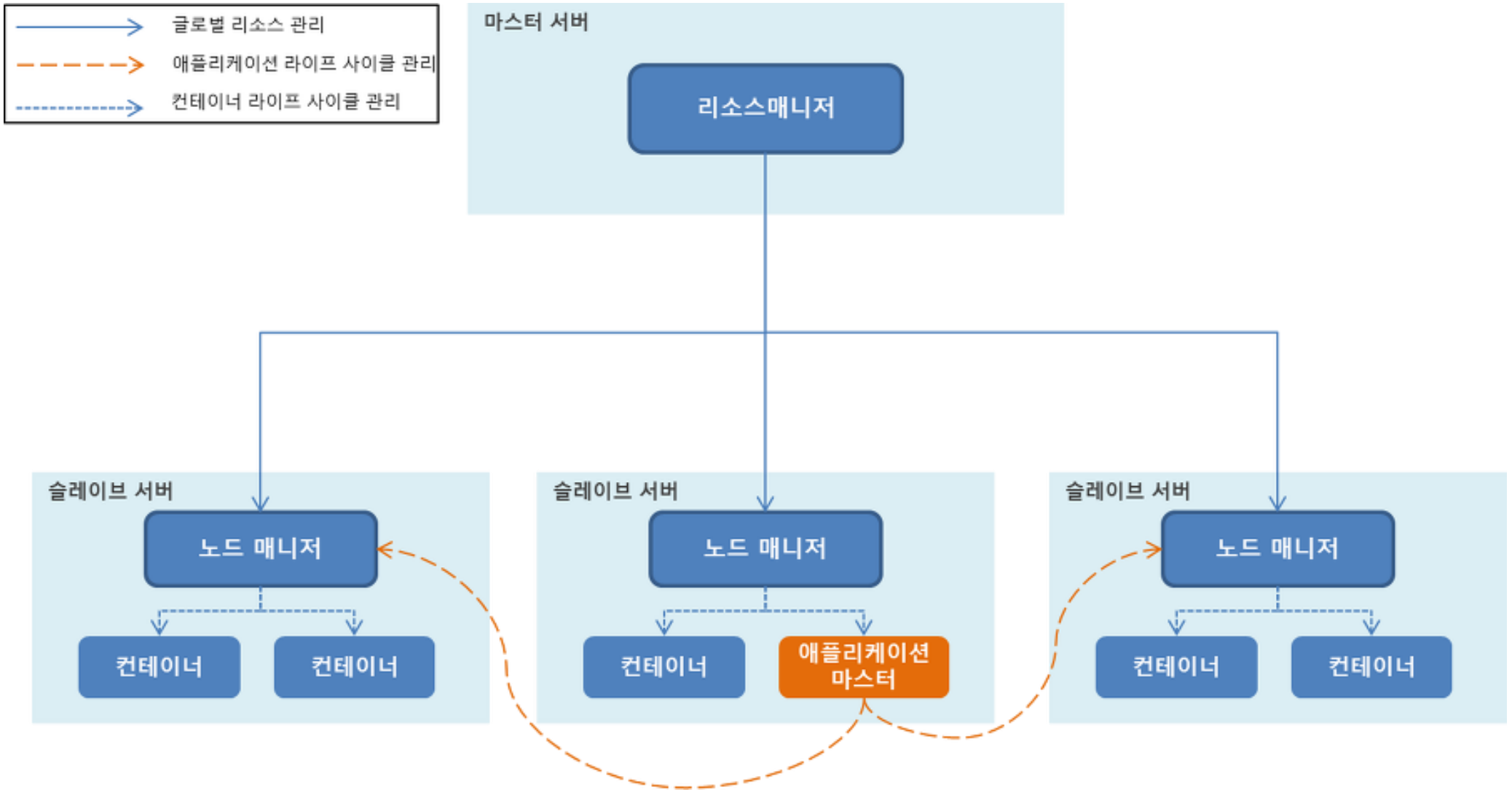


Interact with all data in multiple ways simultaneously

YARN의 등장 배경

1) 맵리듀스의 단일 고장점(Singe Point of Failure, SPOF) : 잡트래커
2) 잡트래커의 메모리 이슈
3) 맵리듀스 리소스 관리 방식 ; 슬롯
4) 클러스터 확장성
5) 버전 통일성

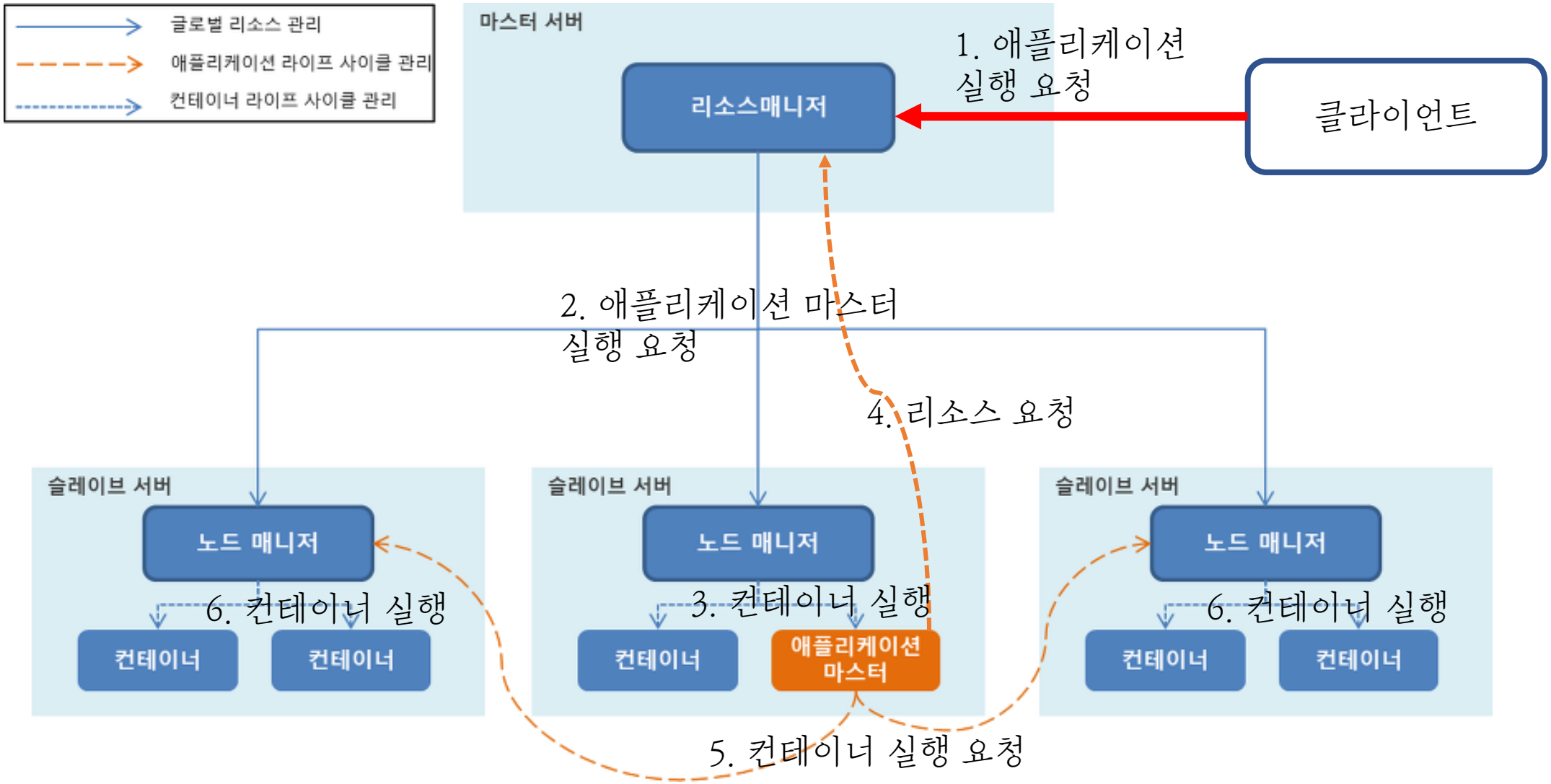
1) 얀 아키텍처


글로벌 리소스 관리
애플리케이션 라이프 사이클 관리
컨테이너 라이프 사이클 관리
마스터 서버
리소스매니저
슬레이브 서버
노드 매니저
컨테이너
컨테이너
슬레이브 서버
노드 매니저
컨테이너
애플리케이션 마스터
슬레이브 서버
노드 매니저
컨테이너
컨테이너

컴포넌트

1) **리소스매니저** : 전체 클러스터에서 가용한 모든 시스템 자원 관리
2) **노드매니저** =: 맵리듀스의 태스크트래커
3) **컨테이너** : 노드매니저가 실행되는 서버의 시스템 자원(cpu, 메모리, 디스크, 네트워크) 표현
4) **애플리케이션마스터** : 하나의 애플리케이션을 관리하는 마스터 서버, 컨테이너에서 실행됨

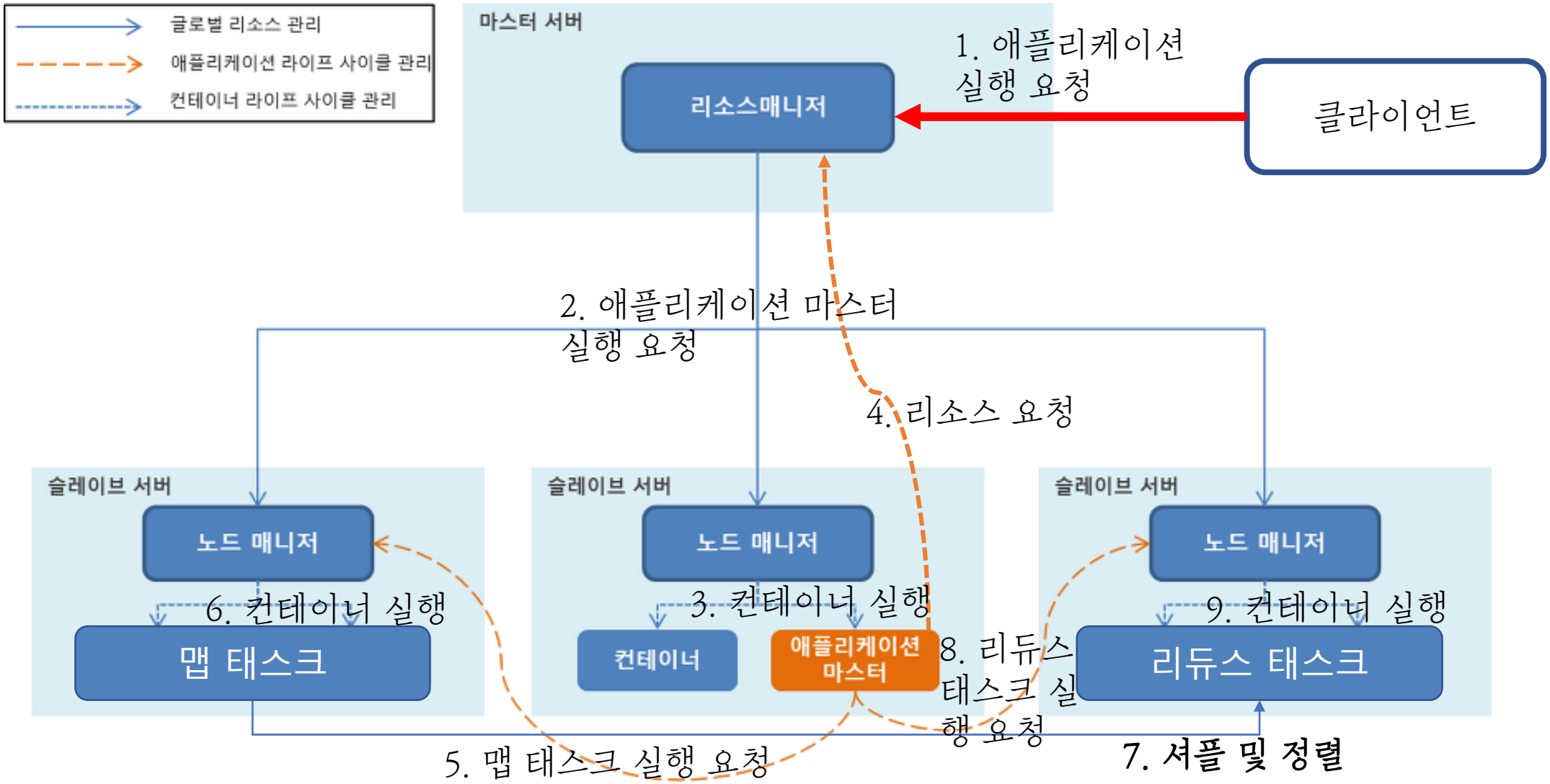
2) 얀 작업흐름


글로벌 리소스 관리
애플리케이션 라이프 사이클 관리
컨테이너 라이프 사이클 관리
마스터 서버
리소스매니저
1. 애플리케이션 실행 요청
클라이언트
2. 애플리케이션 마스터 실행 요청
4. 리소스 요청
슬레이브 서버
노드 매니저
6. 컨테이너 실행
컨테이너
컨테이너
슬레이브 서버
노드 매니저
3. 컨테이너 실행
컨테이너
애플리케이션 마스터
슬레이브 서버
노드 매니저
6. 컨테이너 실행
컨테이너
컨테이너
5. 컨테이너 실행 요청

3) 보조 서비스(Auxiliary Service)

- 맵리듀스는 '셔플' 작업이 필요함.
- 얀 클러스터에서 맵리듀스 애플리케이션 실행 시, 컨테이너에서 실행 중이던 맵 태스크 종료시 컨테이너 함께 종료 → 맵 태스크는 리듀스 태스크에게 정보전달 불가→ 셔플 실행 불가 → **보조서비스 제공**!
- 노드매니저 간의 서비스 제어를 위한 기능, 서로 다른 노드매니저 사이에 데이터를 전달하거나 다른 노드매니저를 제어할 수 있음.
- 맵 태스크를 실행하는 노드매니저와 리듀스 태스크를 실행하는 노드매니저 간의 셔플 가능

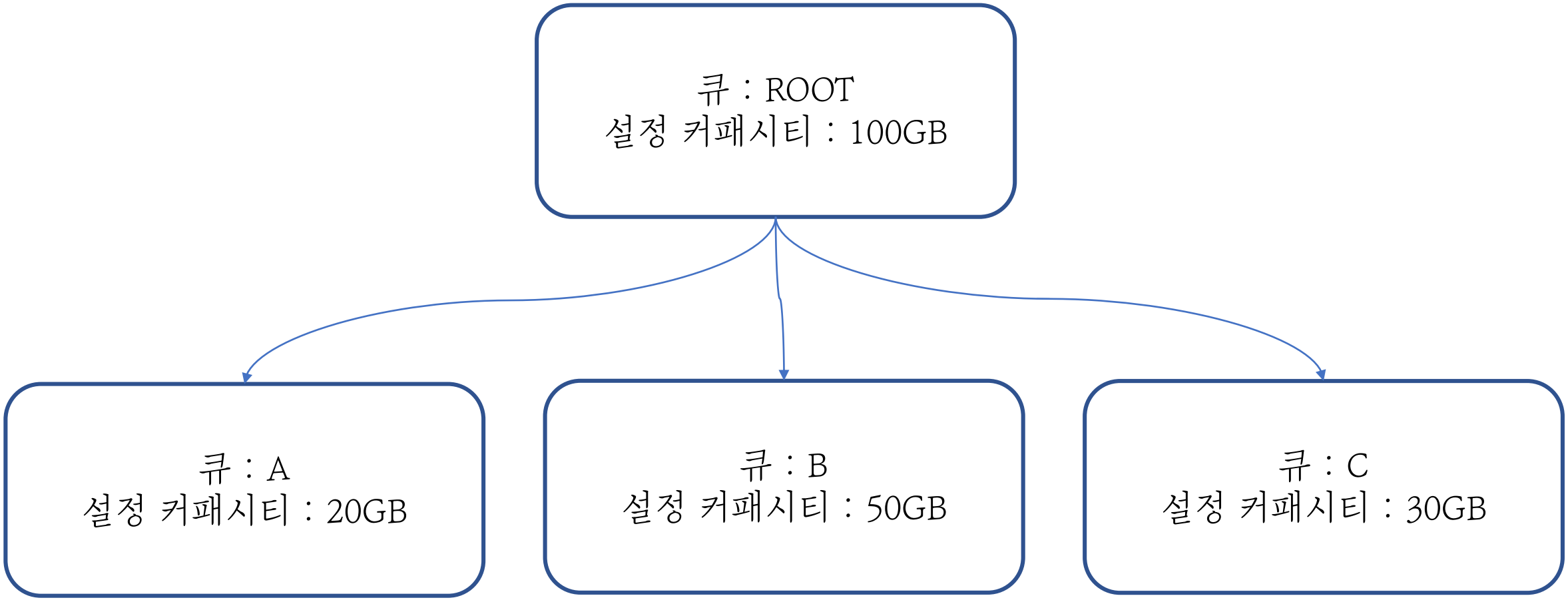
얀 클러스터에서 맵리듀스 애플리케이션 동작

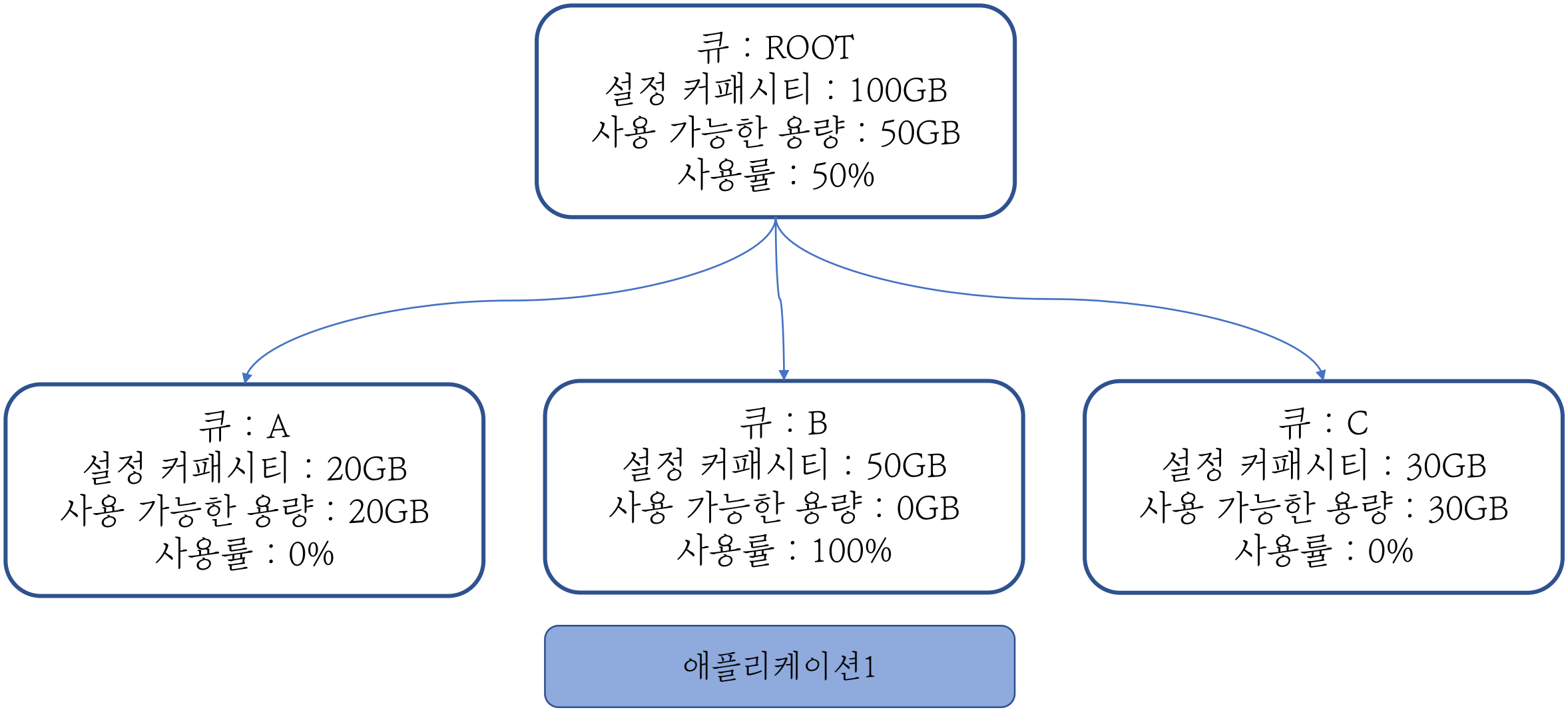


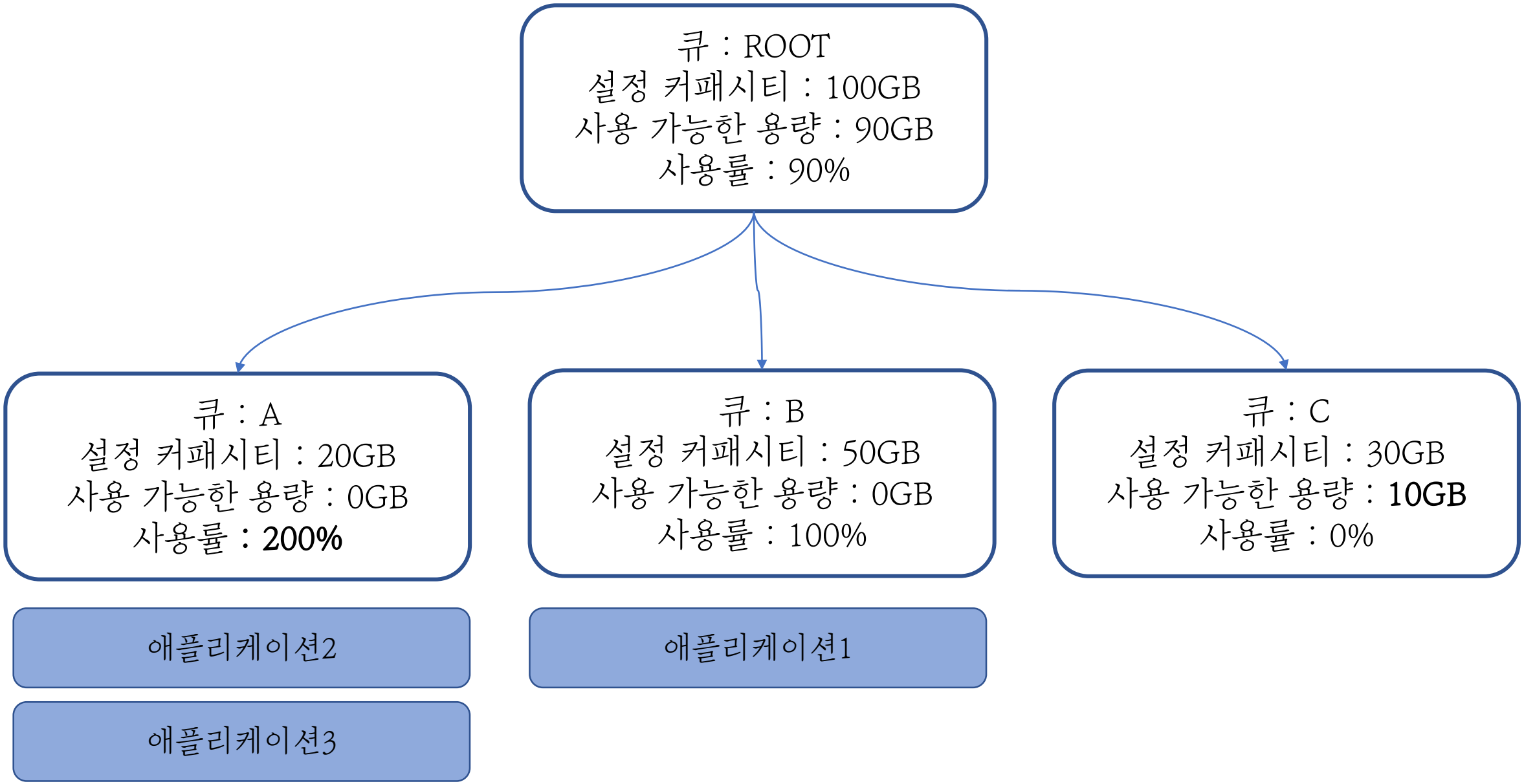
4) 프리엠션

- 효율적으로 리소스를 관리하고, 얀 애플리케이션의 안정성을 보장하기 위해 프리엠션 기능을 제공

- 얀의 커패시티 스케줄러는 계층적인 큐 형식으로 리소스 정의




큐 : ROOT
설정 커패시티 : 100GB
사용 가능한 용량 : 50GB
사용률 : 50%
큐 : A
설정 커패시티 : 20GB
사용 가능한 용량 : 20GB
사용률 : 0%
큐 : B
설정 커패시티 : 50GB
사용 가능한 용량 : 0GB
사용률 : 100%
큐 : C
설정 커패시티 : 30GB
사용 가능한 용량 : 30GB
사용률 : 0%
애플리케이션1


큐 : ROOT
설정 커패시티 : 100GB
사용 가능한 용량 : 90GB
사용률 : 90%
큐 : A
설정 커패시티 : 20GB
사용 가능한 용량 : 0GB
사용률 : 200%
큐 : B
설정 커패시티 : 50GB
사용 가능한 용량 : 0GB
사용률 : 100%
큐 : C
설정 커패시티 : 30GB
사용 가능한 용량 : 10GB
사용률 : 0%
애플리케이션2
애플리케이션3
애플리케이션1

프리엠션이란?

- 자원을 공평하게 할당받지 못하는 상황에서, 다른 애플리케이션의 자원을 회수할 수 있는 기능
- 프리엠션 활성화 할 경우 : 일정한 주기마다 프리엠션 실행,
- 수용량 이상의 자원을 할당받은 애플리케이션의 컨테이너 강제 회수 / 컨테이너 자체를 종료하여 필요한 자원 확보

5) 타임라인 서비스

- 얀 클러스터에서 실행되는 애플리케이션의 히스토리 상태와 주요 메트릭 정보를 유지하기 위해 → JobHistoryServer 제공
- 문제점

1) JobHistoryServer는 맵리듀스 애플리케이션만을 대상으로 함.
2) 이력 저장용 스토리지도 HDFS로 제한되어 있어 RDBMS나 NoSQL같은 다른 종류의 스토리지 사용 불가

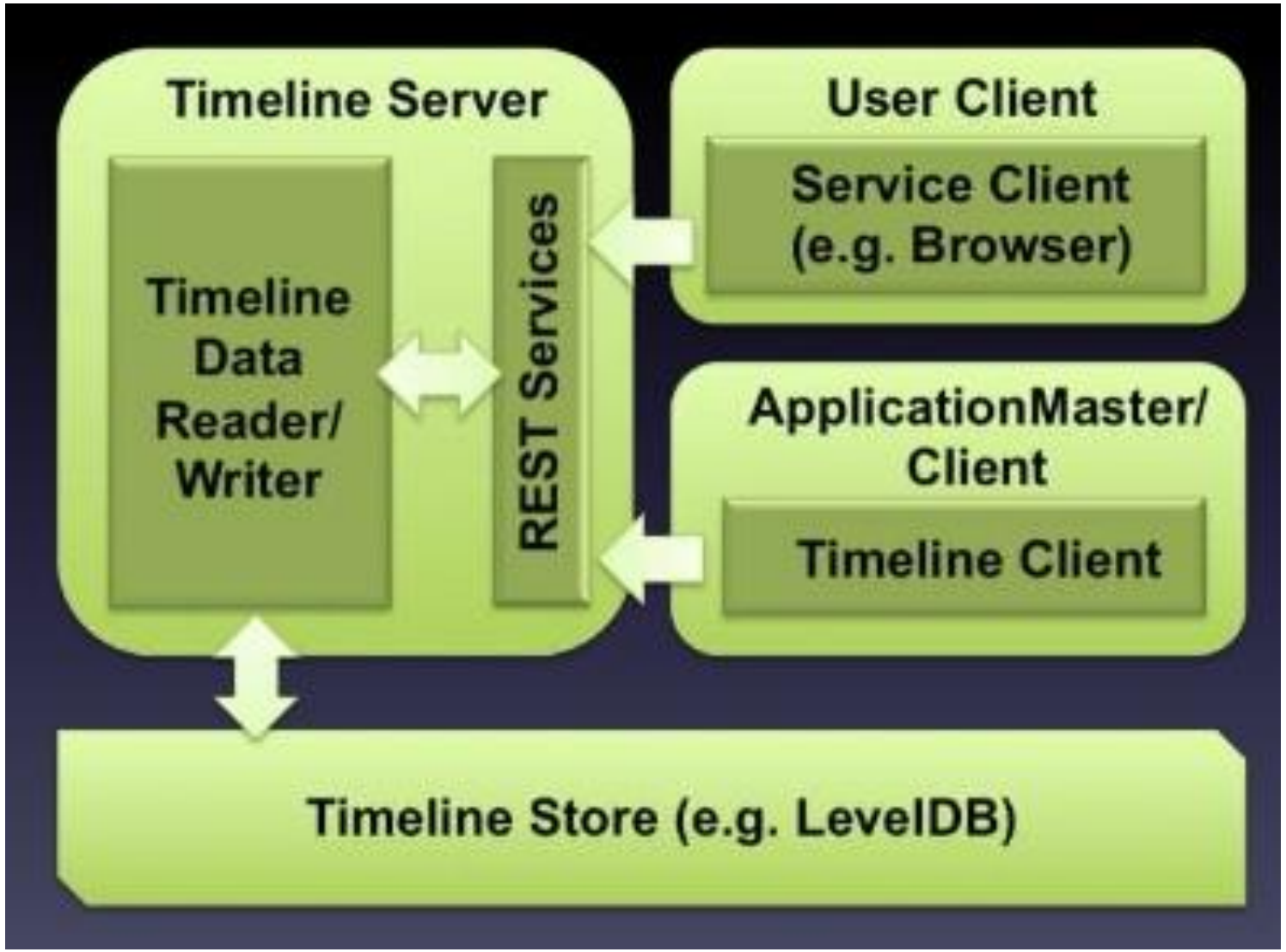
- → 타임라인 서비스 : 맵리듀스 뿐 아니라 다른 종류의 애플리케이션에도 적용 가능

타임라인 서비스 아키텍처

- 타임라인 저장소 : JobHistoryServer가 HDFS에만 데이터를 저장하는 것과 달리, 다양한 스토리지를 저장소로 활용할 수 있음. Ex. 레벨DB(NoSQL)
- 타임라인 클라이언트
- 서비스 클라이언트
- REST API

- 타임라인 서비스는 기본적으로 비활성화, yarn-site.xml에 속성을 추가해주어야 한다.

타임라인 서비스 아키텍처



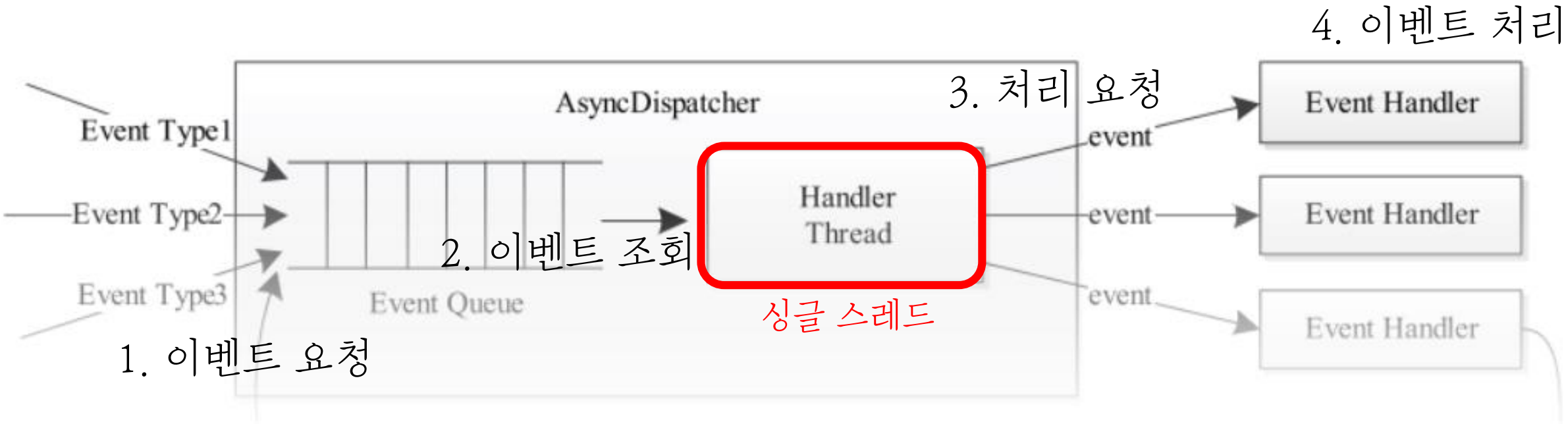
6) 얀 이벤트 처리 방식

- 얀 동작시 내부적으로 다양한 이벤트 발생
- 얀은 효율적으로 이벤트를 처리하기 위해 **비동기 디스패처**와 **스테이트머신**모델 제공

6-1) 비동기 디스패처

- 맵리듀스 : 잡트래커와 태스크트래커의 내부 이벤트를 **멀티 스레딩 방식**으로 처리 → 스레드가 많아질 경우 시스템 많은 자원 소모
- 얀 : **비동기 방식**으로 이벤트 처리 → 비동기 디스패처 제공

비동기 디스패처 동작 과정



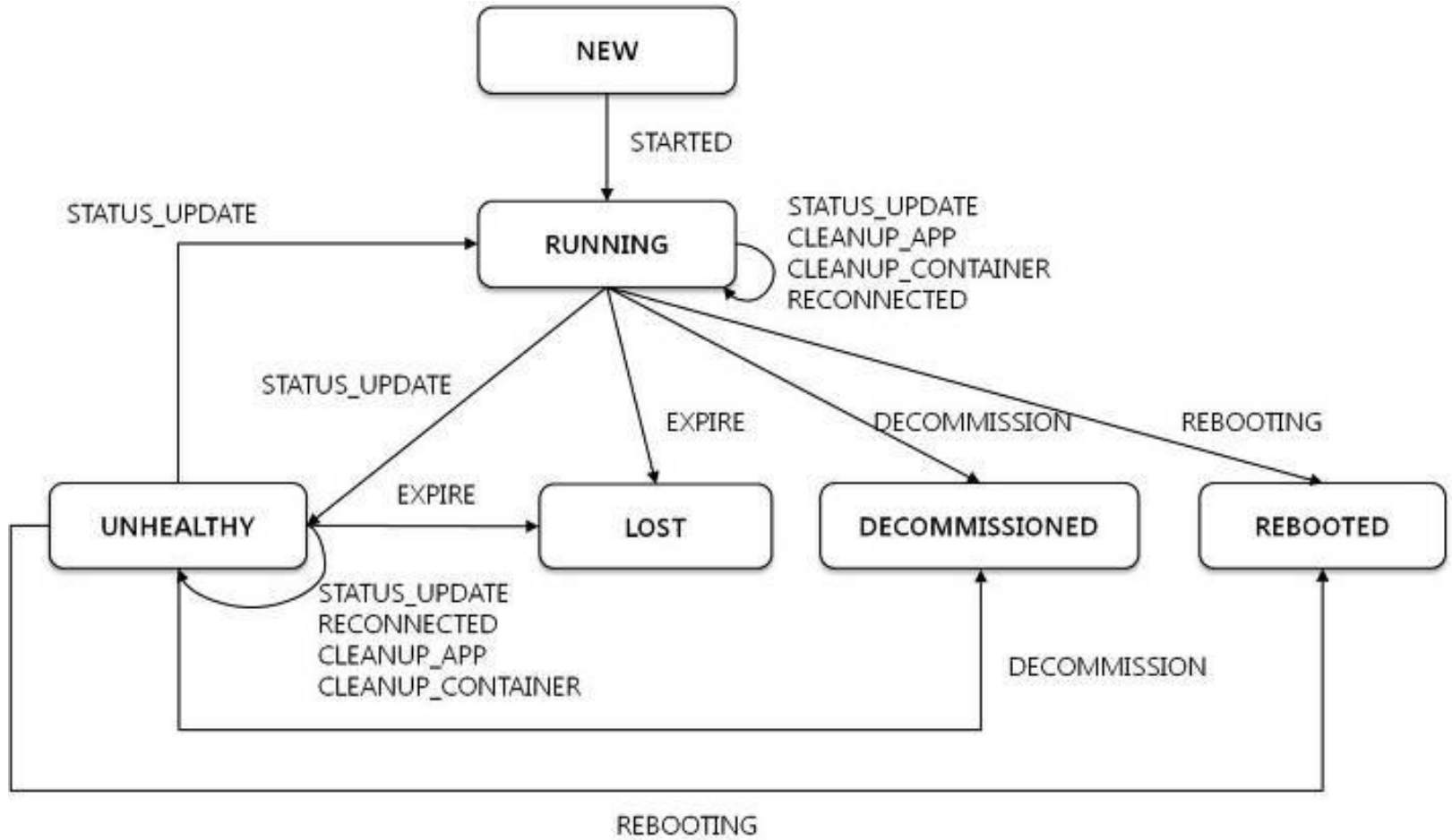
- **싱글 스레드** 기반

멀티 스레드 방식으로 인한 동기화나 데드락 같은 복잡한 상황X

기존 방식에 비해 속도 ↑, 시스템 자원 적게 소모

6-2) 상태 관리

- 리소스 매니저와 노드매니저의 내부 컴포넌터들은 상태 정보가 다양하게 변경



스테이트머신 모델

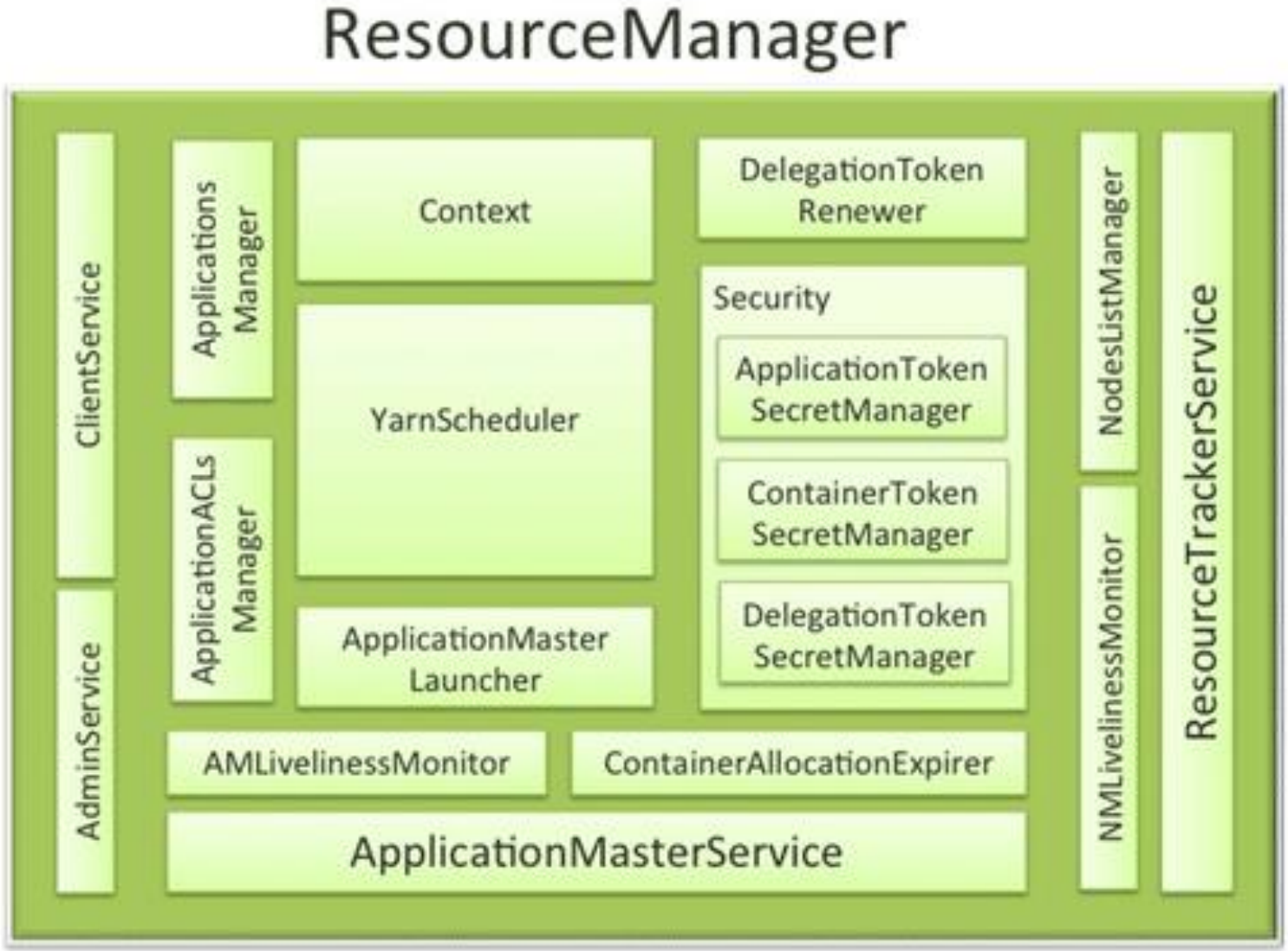
- 상태 변경을 효율적으로 처리하기 위함
- 상태 변경 정보와 관련 이벤트를 명시적으로 설정 ➔ 상태 변경을 자동으로 인지하고 관련된 로직 처리, 잘못된 상태 전환이 발생했을 때 자동으로 장애 처리가 되도록 설정

7) 아키텍처 심화 학습

- **리소스매니저**

–전체 클러스터의 가용한 리소스 스케줄링
–클러스터에서 실행되는 애플리케이션들에게 리소스 중재
–노드매니저와 애플리케이션마스터 제어

리소스매니저 컴포넌트

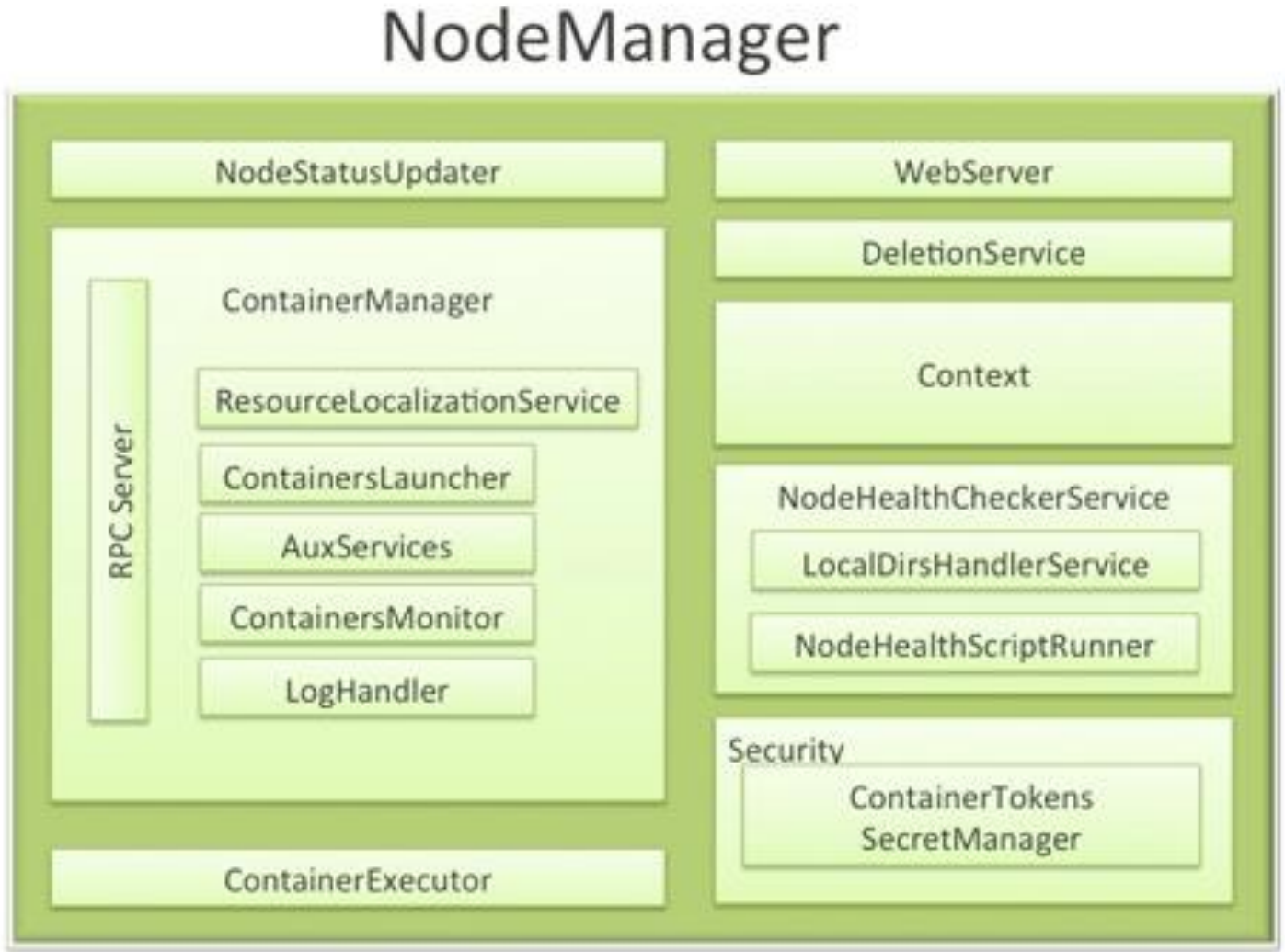

ResourceManager
ClientService
AdminService
Applications Manager
ApplicationACLs Manager
Context
YarnScheduler
ApplicationMaster Launcher
DelegationToken Renewer
Security
ApplicationToken SecretManager
ContainerToken SecretManager
DelegationToken SecretManager
NodesListManager
NMLivelinessMonitor
ResourceTrackerService
AMLivelinessMonitor
ContainerAllocationExpirer
ApplicationMasterService

7) 아키텍처 심화 학습

- **노드매니저**

–컨테이너의 라이프 사이클 관리(컨테이너 실행, 모니터링, 종료)
–실행되는 서버 자원 모니터링, 리소스매니저에게 정보 전송
–보조 서비스 제공

노드매니저 컴포넌트


NodeManager
NodeStatusUpdater
WebServer
DeletionService
ContainerManager
RPC Server
ResourceLocalizationService
ContainersLauncher
AuxServices
ContainersMonitor
LogHandler
Context
NodeHealthCheckerService
LocalDirsHandlerService
NodeHealthScriptRunner
Security
ContainerTokens
SecretManager
ContainerExecutor

끼야냐 ㅇ냐냐앙냐냐냐냐냐
냐냐냐냐냐냐냐냐끄으으으읏

실습 ~~