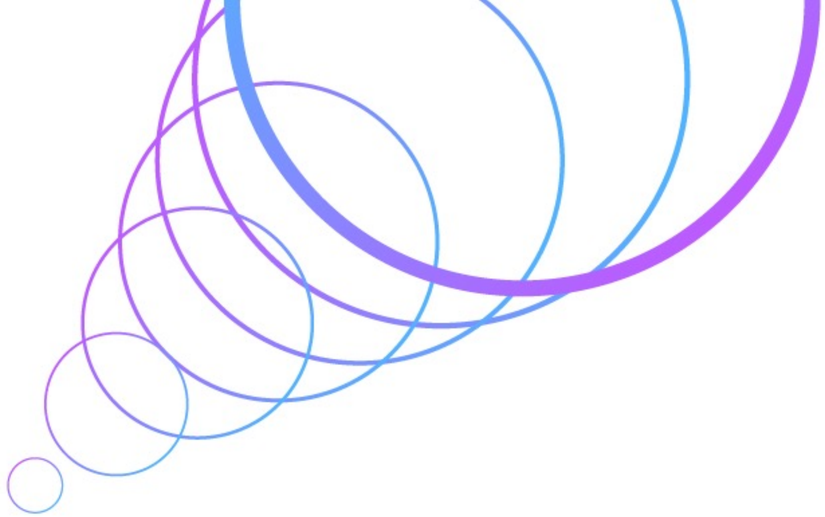


DEVIEW 2021

# 네이버 스토리지 플랫폼의 진화
# Nubes

문상철 NAVER PLATFORM LABS

# CONTENTS

1. 네이버의 분산파일시스템
2. 진화의 필요성
3. Nubes
4. 현재와 미래

# 1. 네이버의 분산파일시스템

# 1.1 분산파일시스템?

DEVIEW 2021

## 정의

분산 시스템

- 독립적인 여러 컴퓨터가 네트워크로 연결되어 동작하나, 사용자 입장에서 마치 한대의 컴퓨터처럼 보이는 시스템

분산 파일 시스템

- Distributed File System (DFS)
- 높은 데이터 안전성/고성능/대용량 스토리지를 제공하기 위한 분산시스템

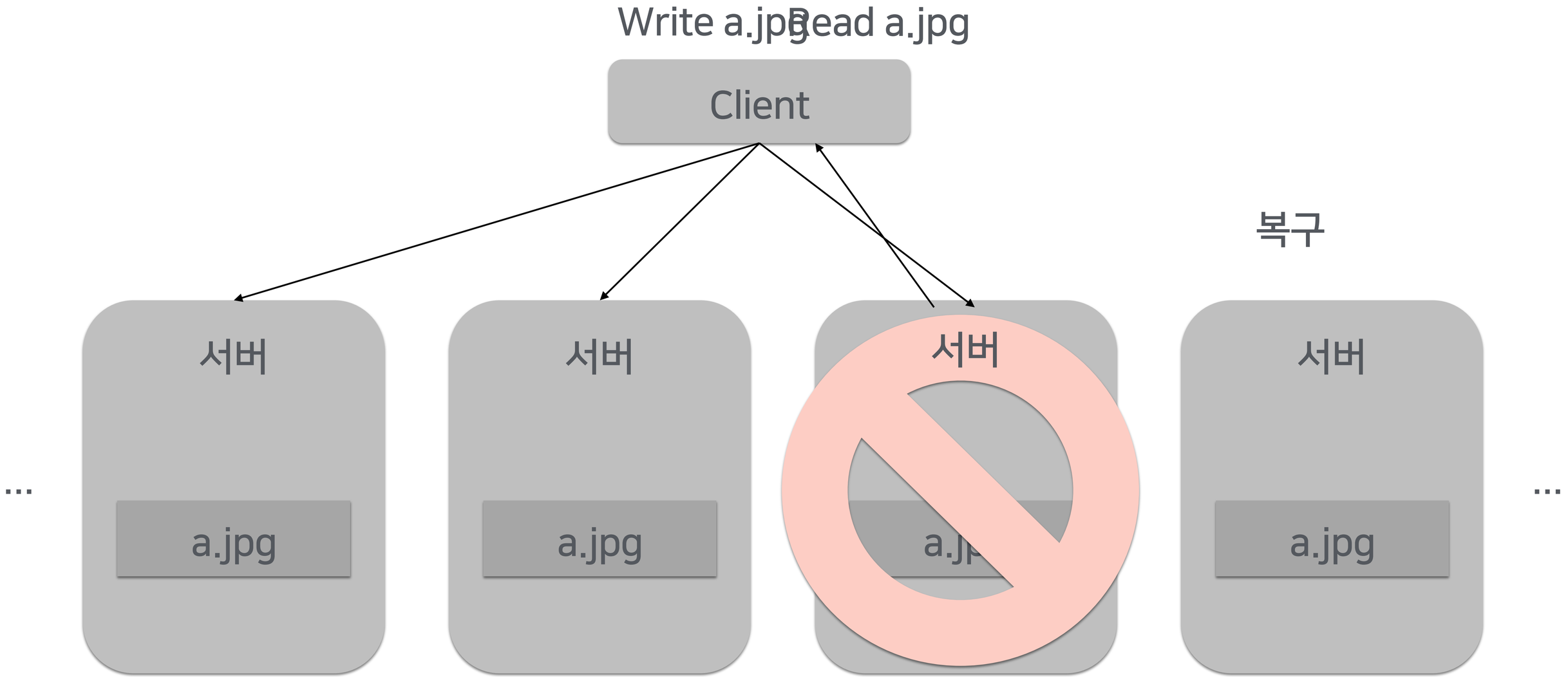
# 1.1 네이버의 분산파일시스템



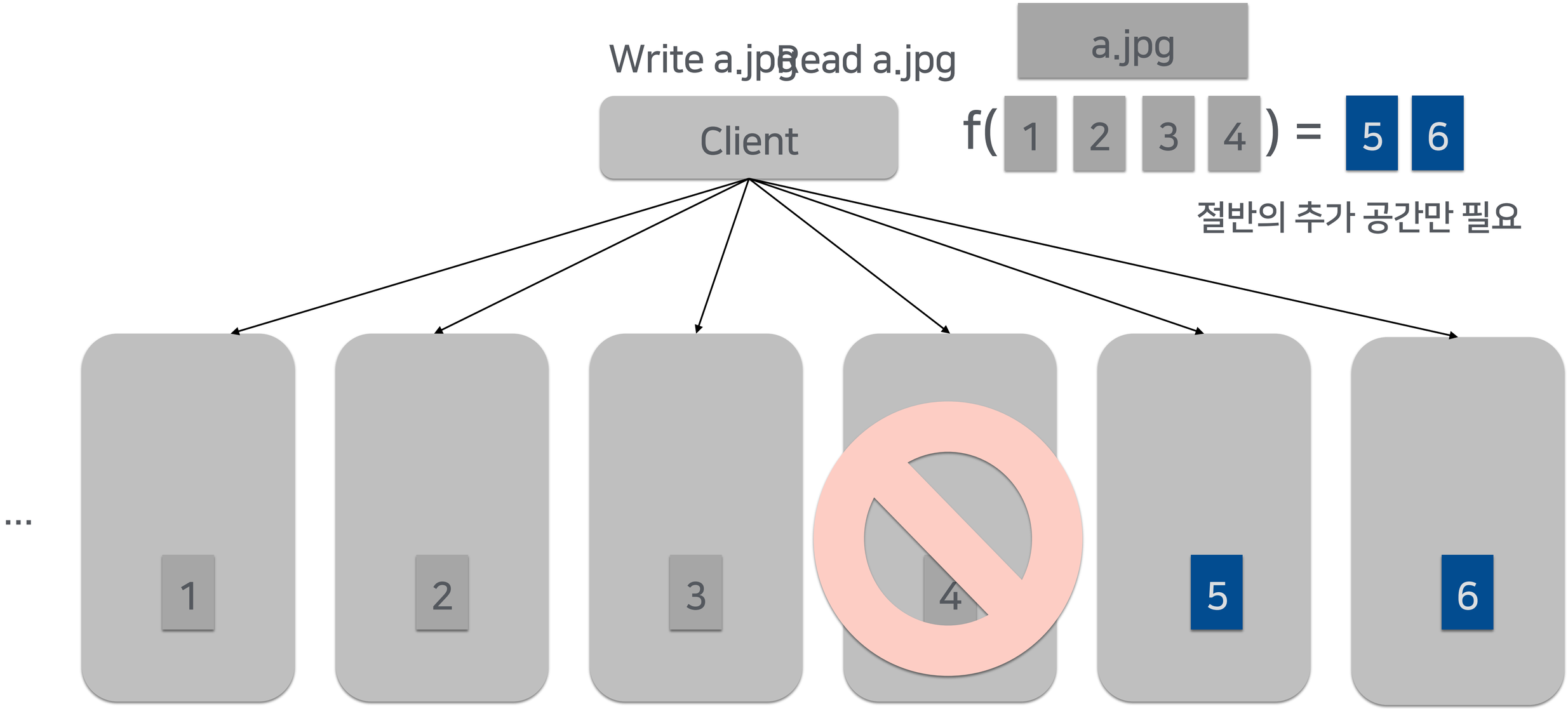
- 역사
  - 2007년 OwFS 개발을 시작으로 지금까지 네이버 대부분의 서비스에서 사용 중

- 종류
  - OwFS : 데이터를 여러 서버에 **복제**하여 고장에 대비
  - Papyrus : 복제 대신 Erasure Code를 활용하여 고장에 대비
  - Cornbox : 복제 기반으로, **크기가 작은 파일 저장에 특화**

# 1.2 OwFS / Cornbox의 복제



# 1.3 Papyrus의 Erasure Code



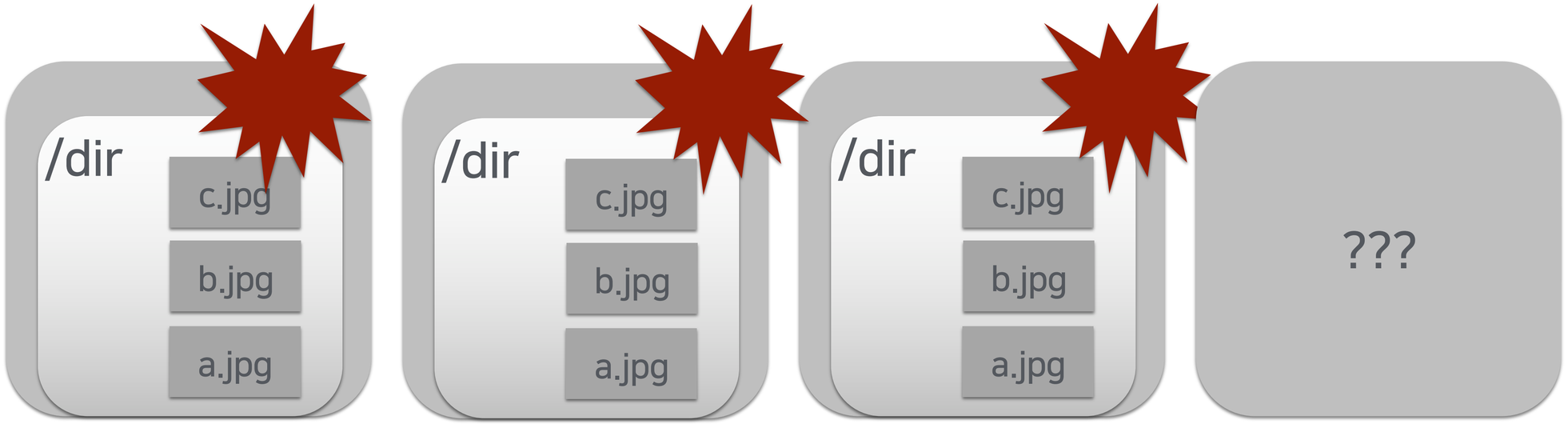
# 2. 진화의 필요성

# 2.1 기존 Naver DFS들의 제약사항

/dir /a.jpg
/b.jpg
/c.jpg
...

- 충분히 많은 디렉토리를 만들어 골고루 써야 함

- 사용자가 부하분산 및 용량 관리를 직접 신경써야 함



# 2.2 DFS마다 서로 다른 기능 및 접근 방식



- C, Java 이외의 언어 및 환경에 대한 지원 문제
- DFS들간 서로 다른 기능, 호환되지 않는 API

# 2.2 데이터 저장 비용 문제

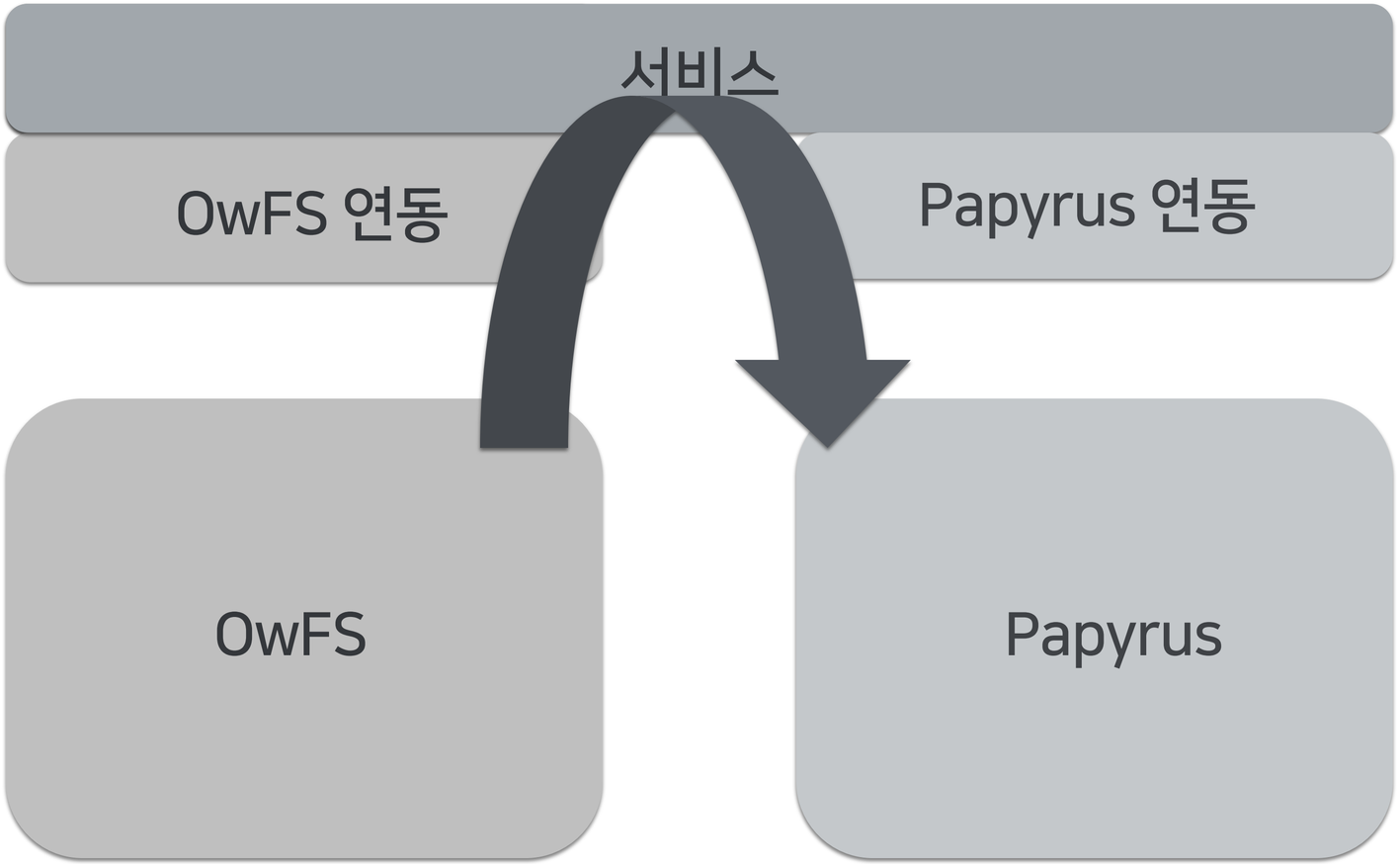
DEVIEW 2021

- 비용
  - 2021년 9월 기준 총 데이터량 360+ PB, 복제시  물리 용량 1 EB가 넘음
  - 디스크/서버/상면/전기/운영 비용

- 비용 절감 방안
  - 데이터 안정성을 위한 **복제본** 방식
    -> Erasure Code 기반 저장 방식으로의 교체 필요성

- 전체 저장 비용을 절반까지 줄일 수 있음

# 해결해야할 문제




- 새 DFS들마다
  서비스 연동 개발

- 서비스 무중단 데이터
이전 방법 필요

# 2.3 DFS들의 다양한 구성

DEVIEW 2021

- 필요한 성능 및 비용에 따라
  - HDD, SSD, NVME …
- 데이터의 활용도에 따라
  - 고속 매체에 골고루 분산시켜 저장하기
  - 저비용 매체에 밀집해서 저장 후 Power off
- 서비스 측 요청에 따라
  - 전용 스토리지 장비
  - 공용 스토리지
  - 임시 스토리지

그 때마다 반복되는

스토리지 연동과 데이터 이전!

- 인프라에서 요청
  - 노후 스토리지 장비 교체
  - IDC 이전

# 그냥 스토리지가 다 알아서 해주면 안 되나

# 3. Nubes

# Nubes

DEVIEW 2021

- 라틴어로 '구름’
- 네이버 표준 오브젝트 스토리지
- 2016년부터 연구 개발 시작, 2018년 첫 적용
- 2021년 네이버의 대표 서비스들을 비롯한 많은 서비스들 사용 중

# Nubes

- 제약 없는 저장 공간
- REST API 기반
  - 업로드/다운로드, 정보조회, 리스팅, 삭제
  - 병렬업로드, 파일 변경 및 이동, 중복제거, FastCopy, Attributes 등
- 유연한 스토리지 구성 및 Seamless 데이터 이전

# 3.1 Object Storage의 구조



# 3.2 메타데이터 Layer

DEVIEW 2021

## 부하 분산 및 용량 관리에서 해방!

- 자유로운 디렉토리 구조
- 무제한의 파일 개수
- 무한 파일 사이즈
- 부하 분산에 대한 고민도 할 필요 없도록

- 디렉토리 및 파일의 메타데이터 저장 및 관리를 확장성 있게 설계하는게 핵심!

# 3.3 메타데이터 분산 저장



- 메타데이터를 분산 데이터베이스에 샤딩해서 저장

- 그 샤딩 룰은?
- 좋은 샤드키를 선정하기 위한 3가지 규칙

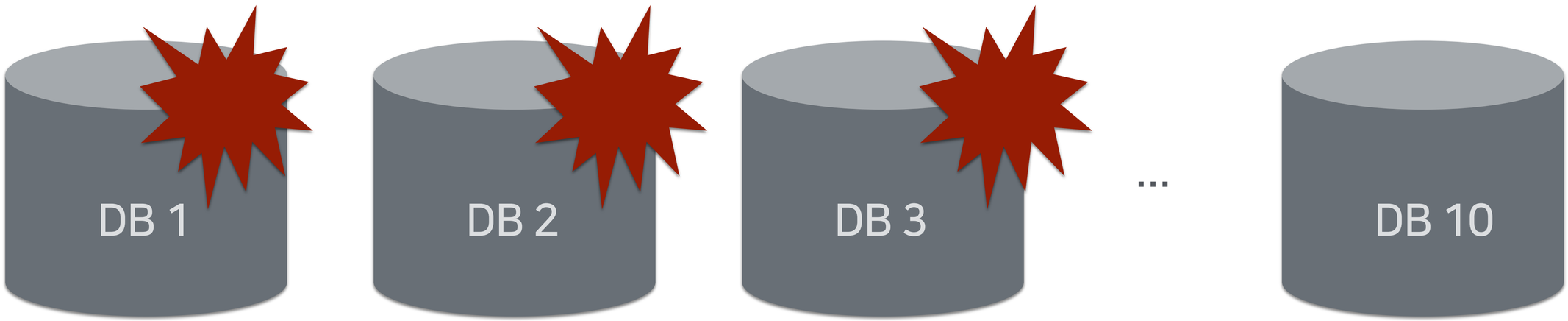
출처:https://docs.mongodb.com/manual/core/sharding-choose-a-shard-key/#std-label-sharding-shard-key-selection

# 샤드키 선정 룰 1

Cardinality 샤드 키가 충분히 넓은 범위를 갖는가?

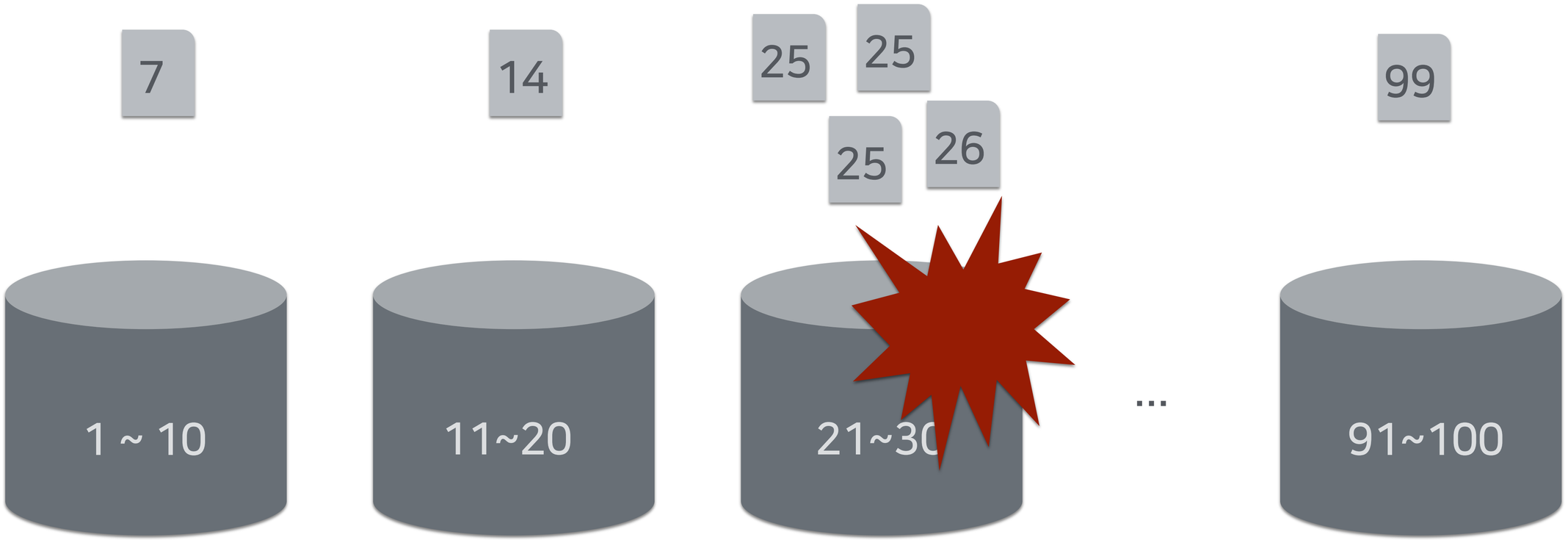
Key Range : 1 ~ 3

Key Range : 1 ~ 1000 OK



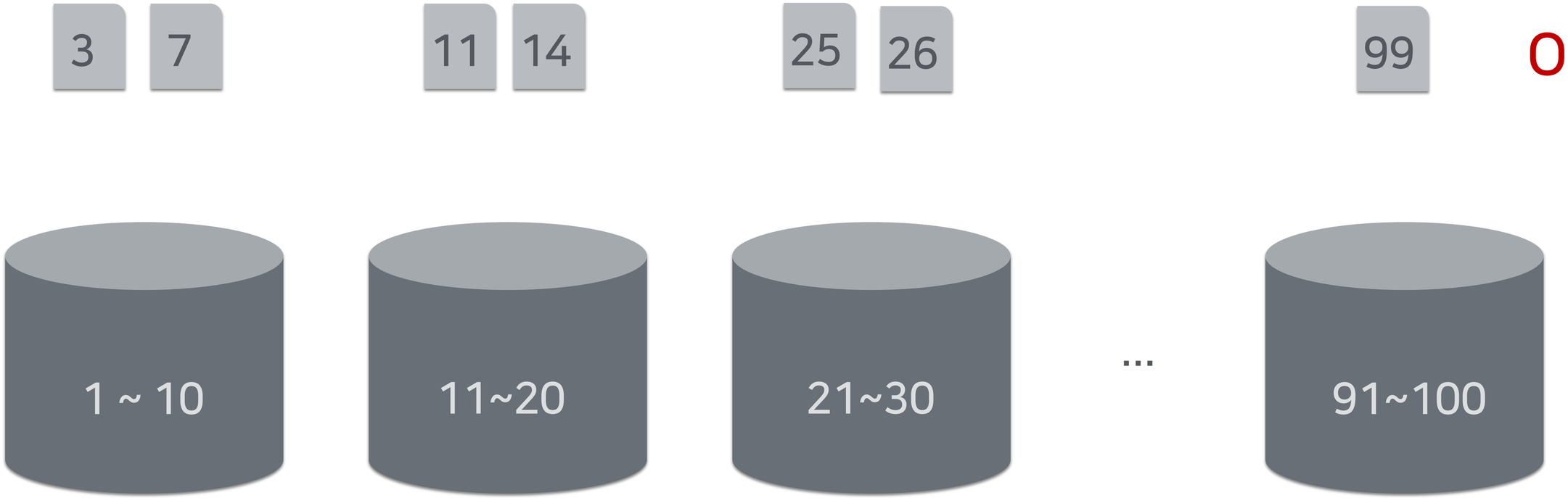
# 샤드키 선정 룰 2

Frequency 샤드키가 특정 값에 치중해있진 않은가?



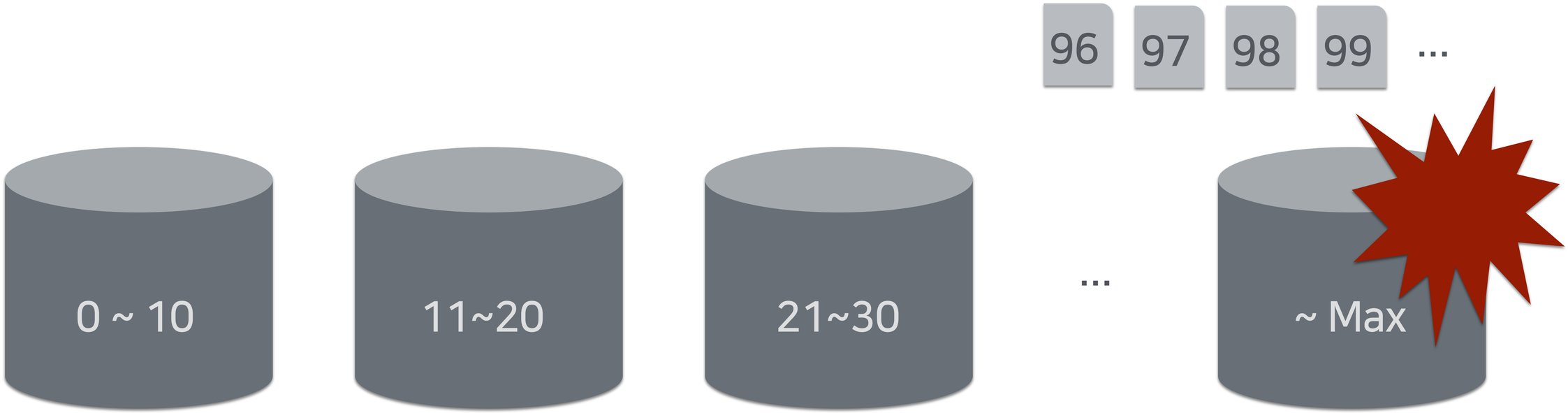
# 샤드키 선정 룰 2

Frequency 샤드키가 특정 값에 치중해있진 않은가?



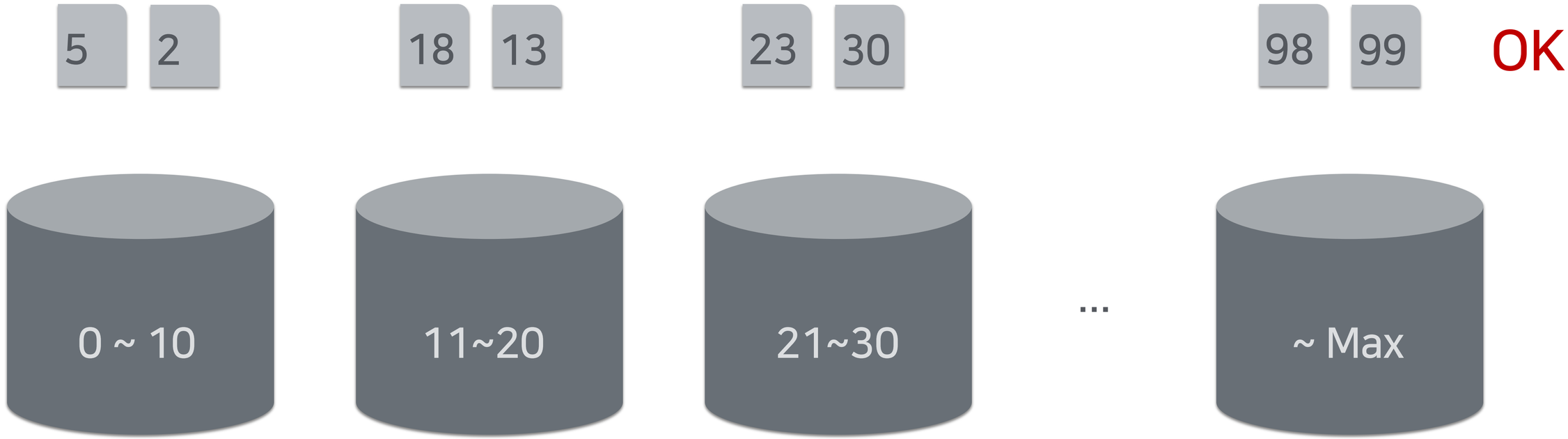
# 샤드키 선정 룰 3

Monotonically changing key 샤드키가 특정 방향으로 커지는 값은 아닌가?



# 샤드키 선정 룰 3

Monotonically changing key 샤드키가 특정 방향으로 커지는 값은 아닌가?

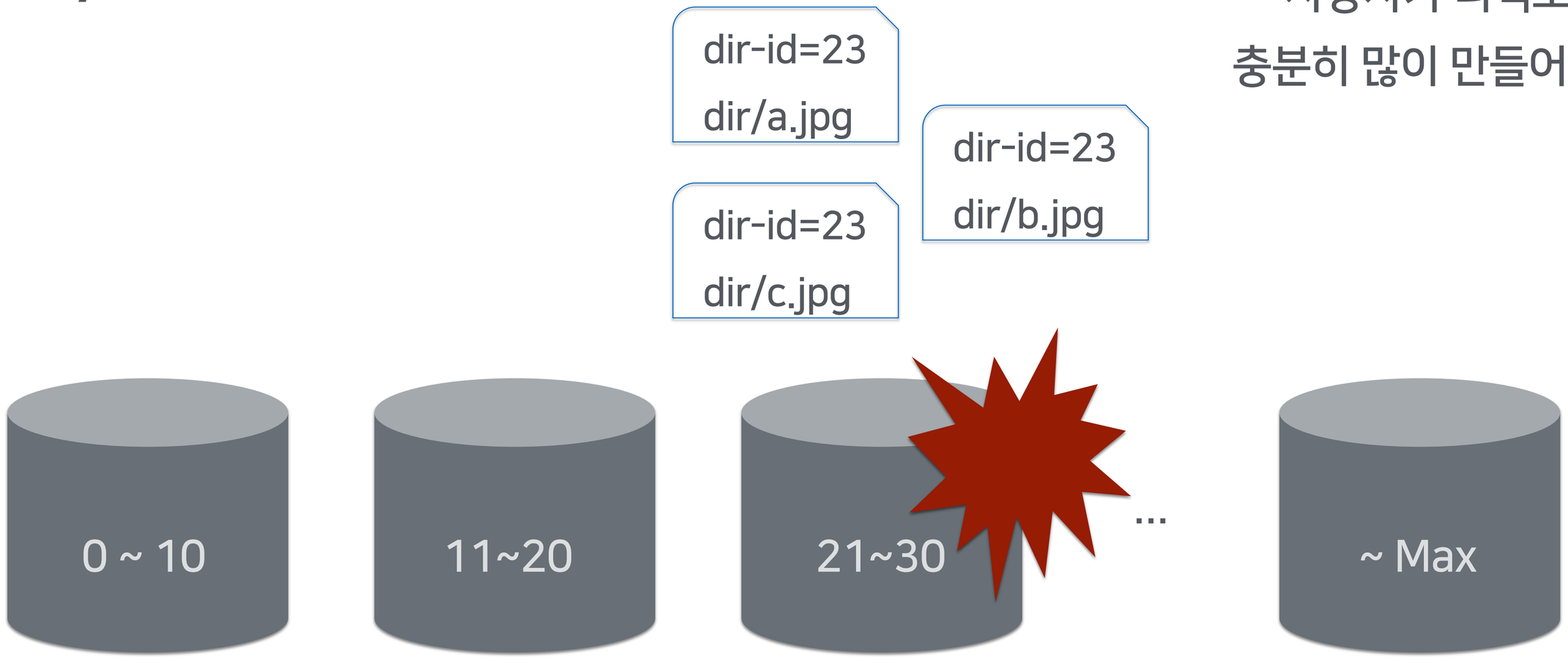


# 3.3 dir을 기준으로 샤딩할 경우

DEVIEW 2021

**/dir/file**

- 사용자가 디렉토리를 충분히 많이 만들어야 함



# 3.3 file을 기준으로 샤딩할 경우

DEVIEW 2021

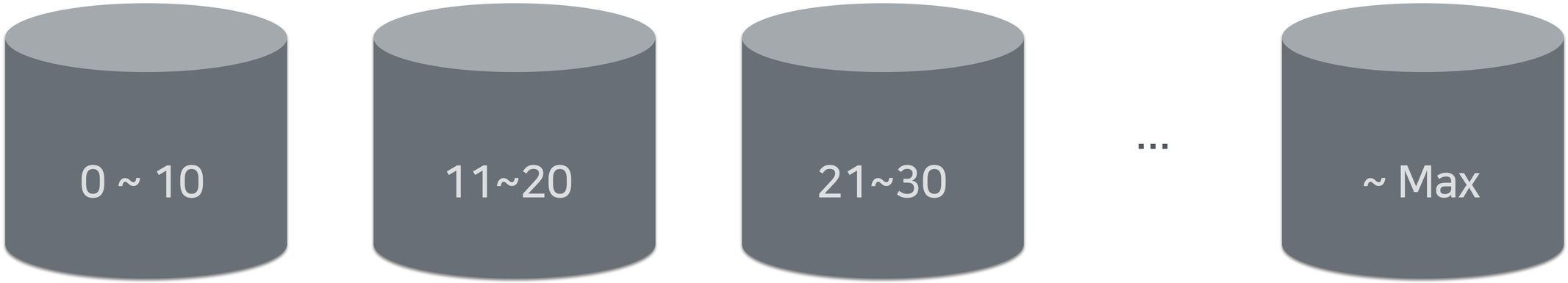
/dir/**file**

file-id=3
dir/a.jpg

file-id=17
dir/c.jpg

file-id=29
dir/b.jpg

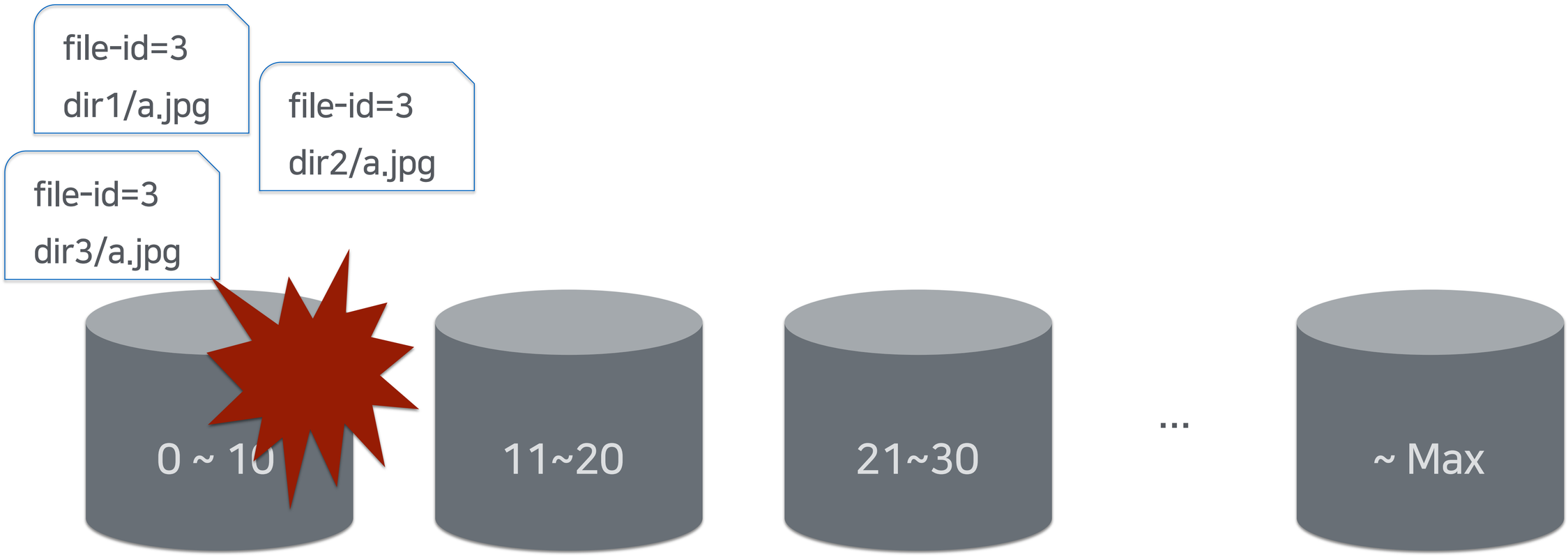
OK



# 3.3 file을 기준으로 샤딩할 경우

DEVIEW 2021

**/dir/file**

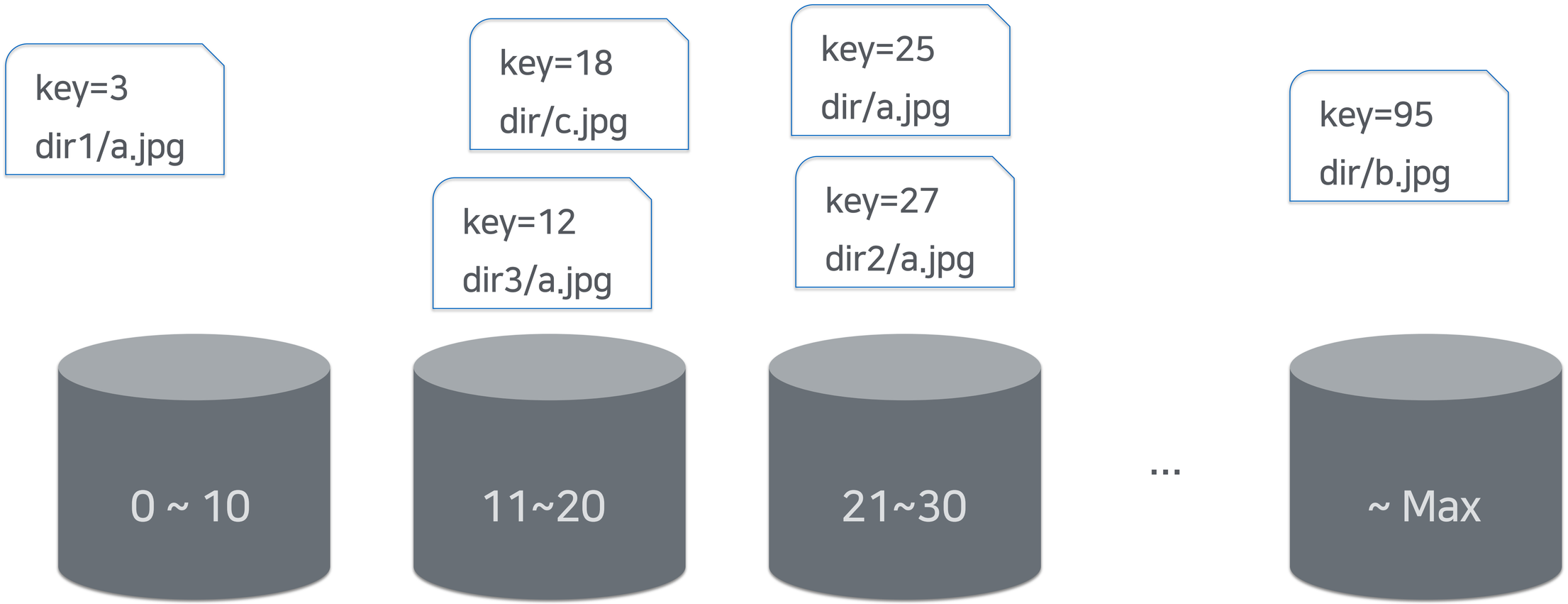


# 3.3 (dir, file)을 기준으로 샤딩



**/dir/file**

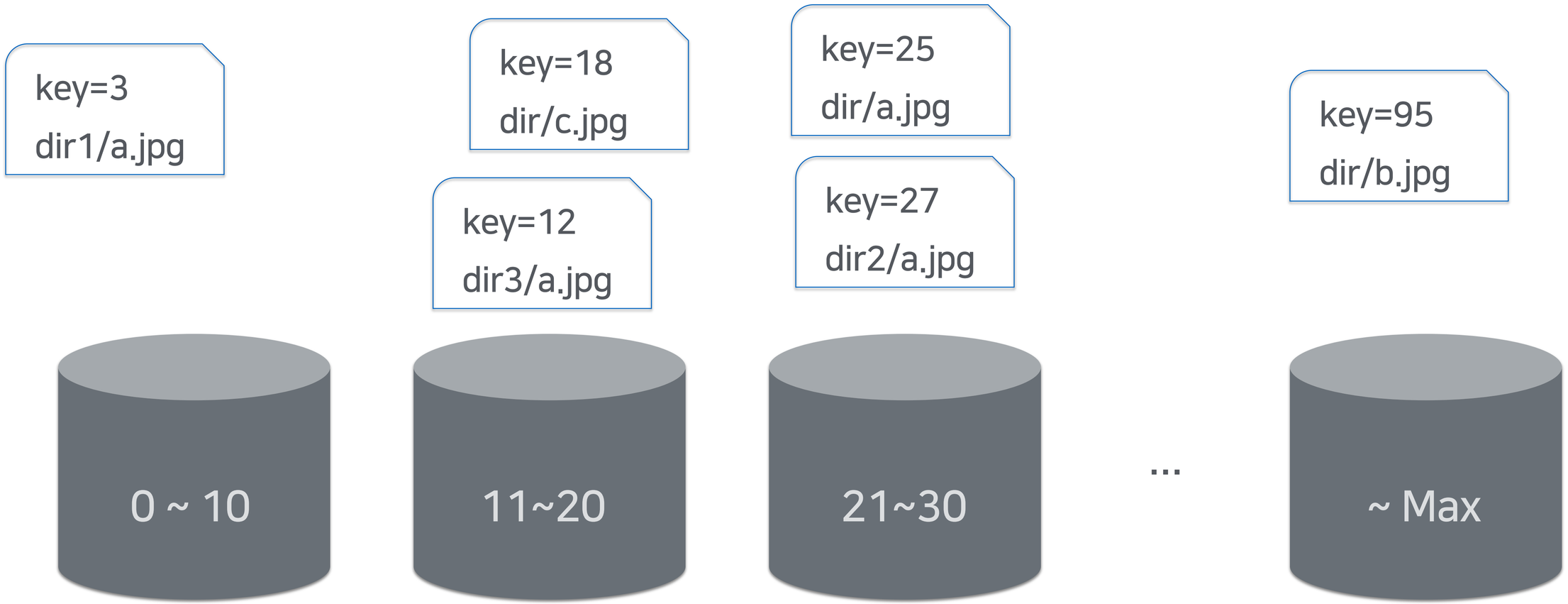
key = hash(dir-id, file-id)



# 3.3 Listing의 부담



## /dir/file



# 3.4 Consistency 모델

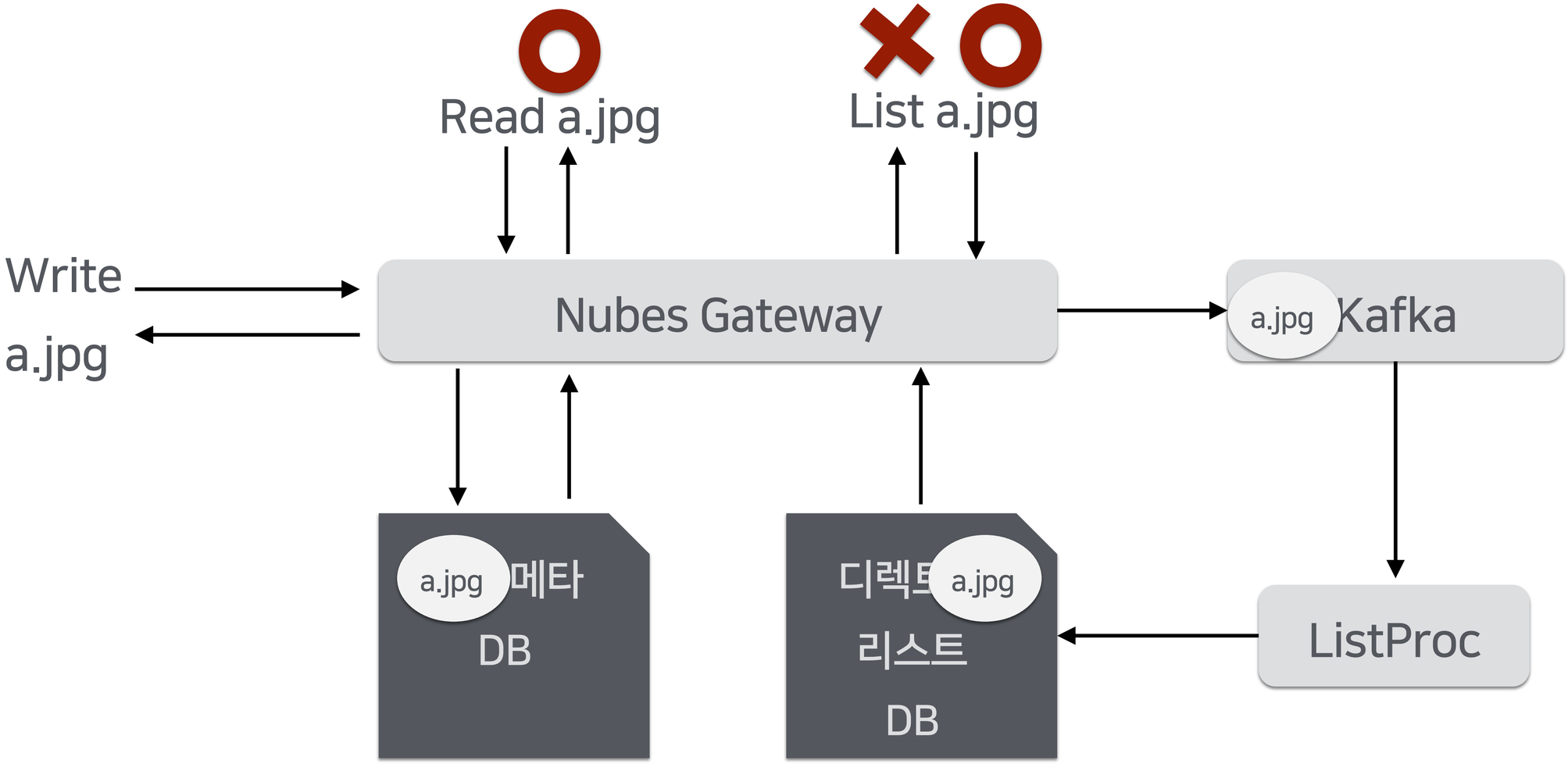


- 읽기 = 파일 읽기 + 디렉토리 리스팅

- 디렉토리 리스팅은 주로 관리용으로 사용

- strong consistency ~ eventual consistency
- 파일 읽기에 대해선 strong consistency 보장
- 디렉토리 리스팅은 부하 상황을 고려하여 천천히 DB에 반영
  -> eventual consistency 보장

# 3.5 쓰기 및 읽기


Read a.jpg
List a.jpg
Write
a.jpg
Nubes Gateway
a.jpg
Kafka
a.jpg
메타
DB
디렉토
a.jpg
리스트
DB
ListProc

# 3.5 두 가지 샤딩 스타일



| 사용자 연산 처리 | |
|---|---|
| **Hashed-sharding Meta DB** | **Ranged-sharding Meta DB** |

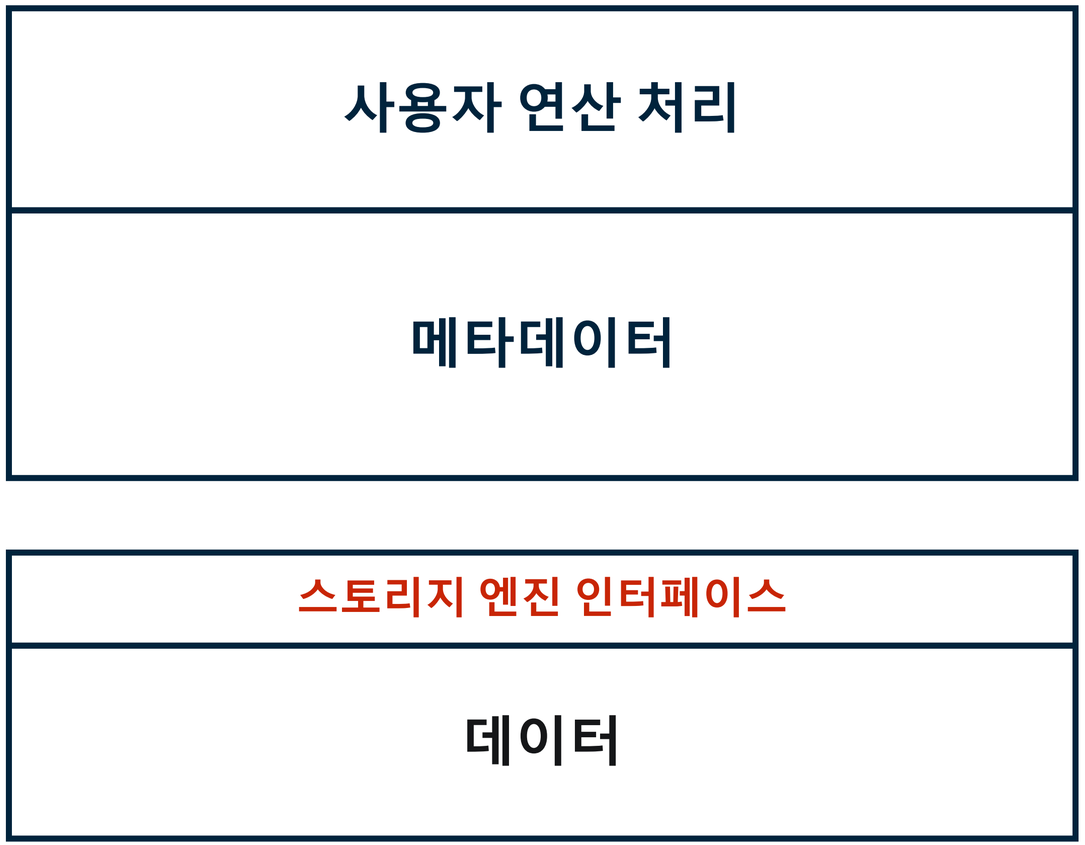
| | | |
|---|---|---|
| 샤딩 룰 | **hash(dir-id, file-id)** 기준 샤딩 | **(dir-id)** 기준 샤딩 |
| 동작 방식 | file write시 바로 insert 후 kafka에 이벤트 발행 | kafka 이벤트를 소비해서 insert |
| 보장 범위 | Strong Consistency 제공 | Eventual consistency 제공 (99%는 수초 내 리스팅 가능) |

# 3.6 Object Storage의 구조



# 3.7 스토리지 엔진 인터페이스






- 데이터 레이어를 물리적으로 분리
- 스토리지 엔진 인터페이스로 Decoupling
- 매우 간단한 API set으로만 구성

# 3.8 데이터 위치 관리 Layer




- 데이터의 위치 정보 및 스토리지 엔진간 이동 관리

# 3.9 위치 관리 Layer의 두 가지 기능


- 데이터가 어떤 스토리지 엔진에 저장되어 있는지 관리하는 모듈

- 스토리지 엔진간 데이터 이동 관리

# 3.10 유연한 스토리지 엔진 구성

# 3.10 스토리지간 데이터 이전



# 3.11 정리



- 제약 없는 저장 공간
  - 확장성있는 메타데이터 저장 구조
  - 자유롭게 디렉토리 구조를 결정 및 사용
  - 부하 분산 및 용량 관리 신경쓸 필요 없음

- 유연한 스토리지 관리
  - 메타데이터와 데이터를 분리하는 스토리지 엔진 인터페이스
  - 런타임에 자유롭게 스토리지 엔진 구성
  - 서비스 영향없이 스토리지 엔진간 데이터 이동

# 4. 현재와 미래

# 4.1 효과



## 비용

- 3 copy 저장에서 ErasureCode 기반 1.5 copy 저장
- 2019~2020년에 걸쳐 MYBOX와 같은 수백PB의 스토리지 교체를 통해 저장 비용을 최대 절반까지 절감
- 동영상, 이미지 등 데이터도 상시로 Cold 스토리지로의 데이터 이전을 통해 저장 비용을 꾸준히 낮추고 있음

# 4.2 효과



## 확장성있는 메타데이터 관리

- 서비스에서 자유롭게 스토리지 사용
- 개발/관리 비용 절감

## 유연한 스토리지 관리

- 서비스 중단, 재개발없이 서비스 중 스토리지를 추가 및 교체 가능
- 서비스 상황에 맞게 임시 스토리지 투입 및 반납 모델 운영

# 4.3 새로운 스토리지 엔진 필요성



- 현재
  - 스토리지 엔진으로는 Naver의 DFS 및 ceph 활용 중

- 필요성
  - 기능도 많고 덩치가 크고 복잡하여 운영 및 유지보수 비용이 발생
  - 스토리지 엔진 역할만 수행하는 DFS가 있으면 운영 복잡성 및 유지보수 비용 감소
  - 고성능/고효율 저장 기술 개발 방향으로 계속 진화

# 4.4 Lemon

DEVIEW 2021

- 특징과 목표
  - Nubes 전용 스토리지 엔진
  - 간단한 API set만 제공하여 복잡도를 낮춤
  - 더 나은 쓰기 가용성 및 더 효율적인 저장 제공을 목표

# 4.5 글로벌 서비스 지원

- 글로벌 진출 서비스들의 요구사항
  - 지역간 network latency 문제
  - 재난에 대비한 데이터 가용성 제공
  - 글로벌 스케일 트래픽 처리를 위한 스토리지 개발 및 운영 제공

# Nubes는 계속 진화 중

# 감사합니다