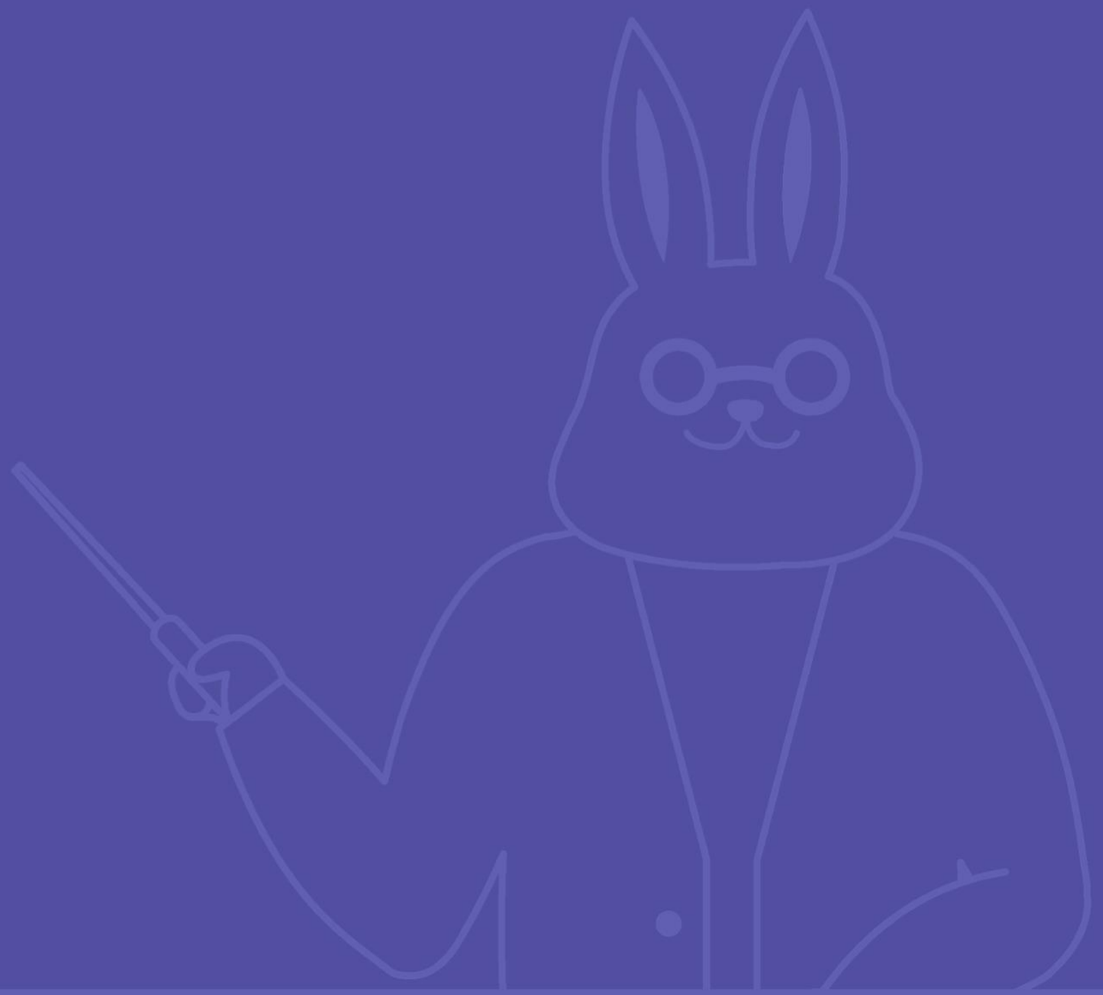
MongoDB 기초

04 고급 활용 기능



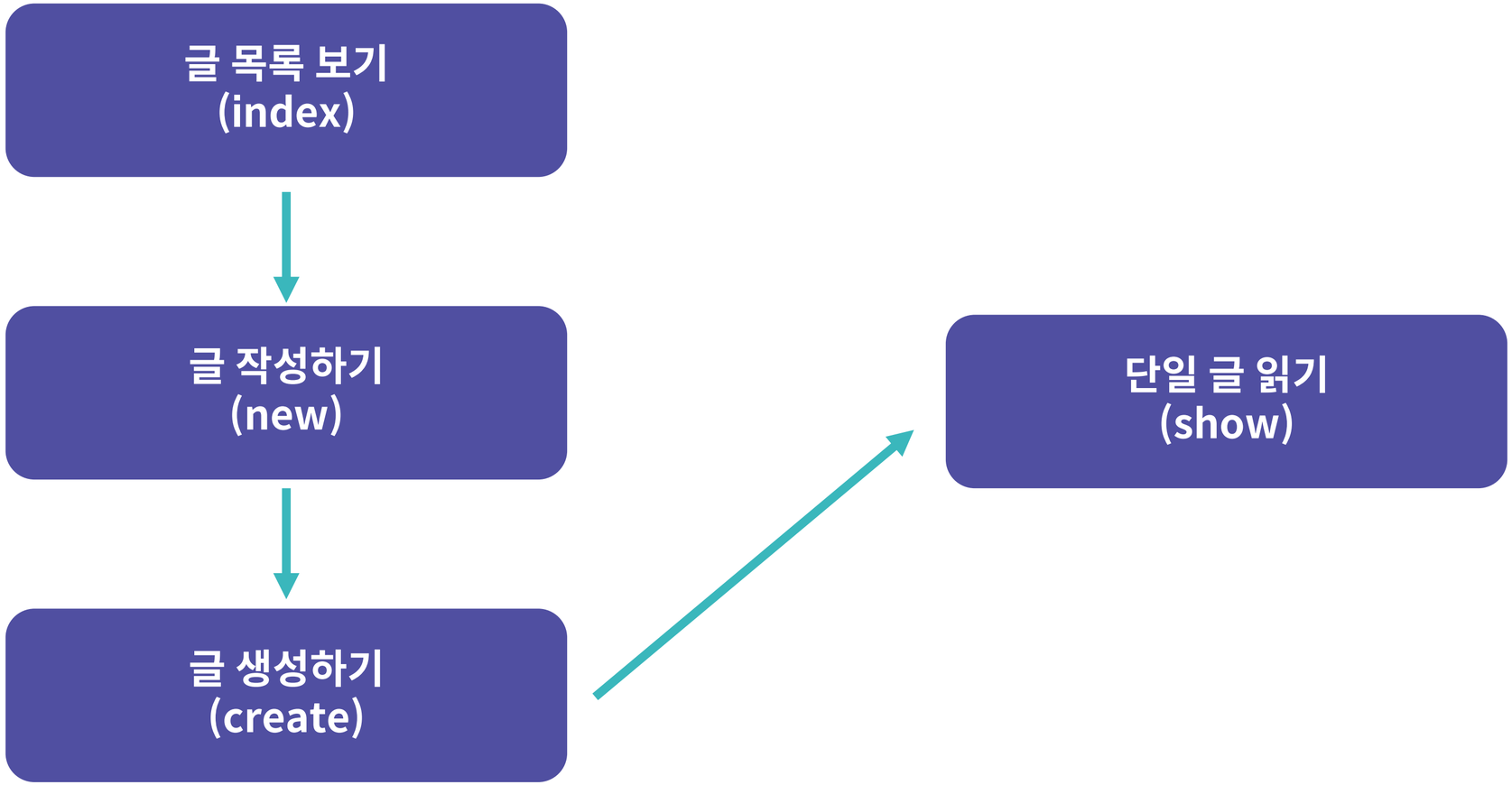
/* elice */

목차

01

Flask와 연결하기

어떻게 만드는지 살펴볼 Flask 페이지 구성



저장할 도큐먼트 형식

post 도큐먼트 구조 예시

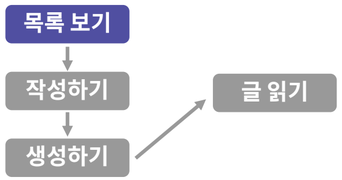
```
{
    "_id": ObjectId("542c2b97 bac059 5474 108b48"),
    "title": "title of post",
    "content": "html 문서 입니다"
}
```

Flask 설정

백엔드

```
import pymongo
from bson import ObjectId
import csv
from flask import Flask, render_template, request, redirect

app = Flask(__name__)
client = pymongo.MongoClient('localhost', 27017)
db = client.get_database("elice")
col = db.get_collection("post")
```

글 목록 보기



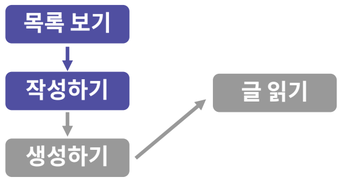
백엔드

```
@app.route("/", methods=['GET'])
def index():
    documents = col.find()
    return render_template('index.html', documents=documents)
```

index.html

```
{% for doc in documents %}
  <li><a href="/post/{{ doc._id }}">{{ doc.title }}</a></li>
{% endfor %}
```

글 작성하기



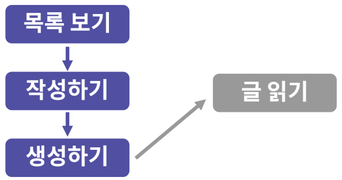
백엔드

```
@app.route("/new")
def new():
    return render_template('main.html')
```

main.html

```
<h2>Add a Post</h2>
<form action="/create" method="POST">
  <table>
    <tr><td>Title: </td><td><input name="title"></td></tr>
    <tr><td>Content: </td><td><input name="content"></td></tr>
  </table>
</form>
```

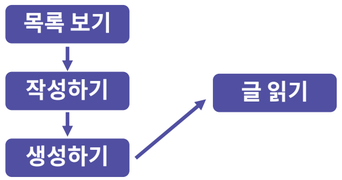
글 생성하기



백엔드

```
@app.route("/create", methods=['POST'])
def save():
    data = {"title": request.form['title'],
        "content": request.form['content']}
    res = col.insert_one(data)
    return redirect(f"/post/{res.inserted_id}")
```

단일 글 읽기



백엔드

```
@app.route("/post/<_id>", methods=['GET'])
def show(_id):
    post = col.find({ "_id": ObjectId(_id) })[0]
    return render_template('show.html', post=post)
```

show.html

```
<h2>{{ post.title }}</h2>
_id: {{ post._id }}<br>
Content: {{ post.content }}<br>
```

02

세 가지 집계 방법론과 효율성

Find 명령으로 할 수 없는 것

지금까지 배운 Find 명령어로는 **집계**와 관련된 내용을 수행하지 못함

예를 들어 강의 수강평 점수 **평균** 몇 점이고,

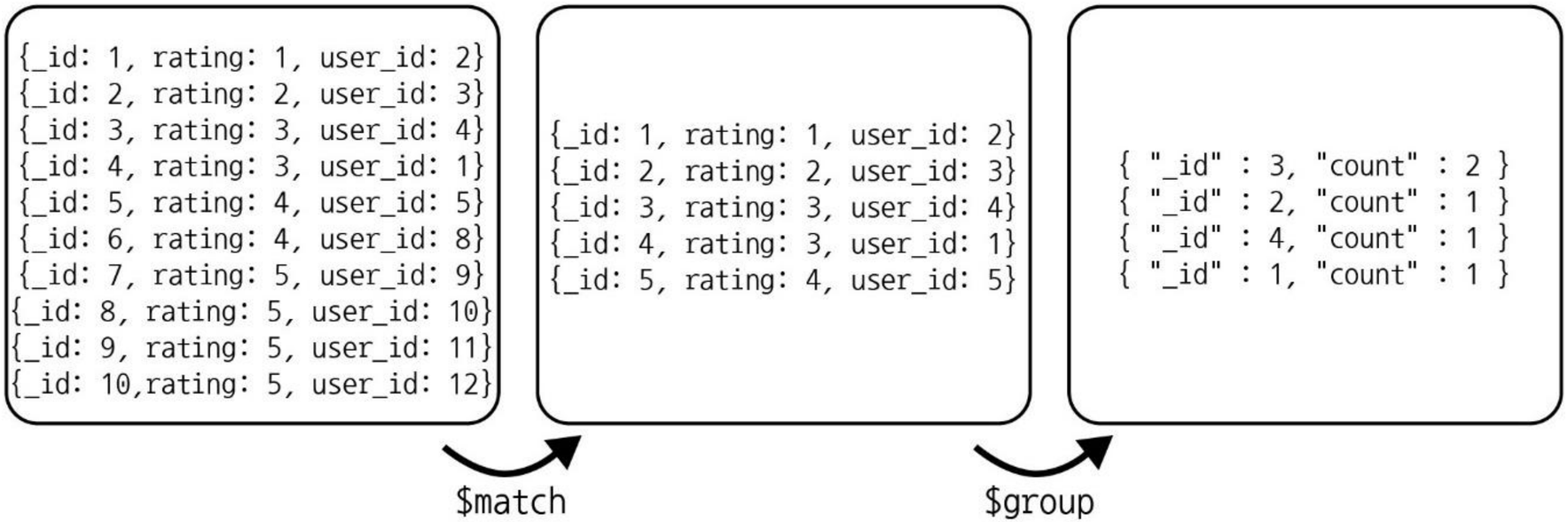
점수별로 얼마나 **분포**했는지 알아내는 것

집계 명령 수행 방법

도큐먼트를 집계하는 방법은 크게 세가지가 있다

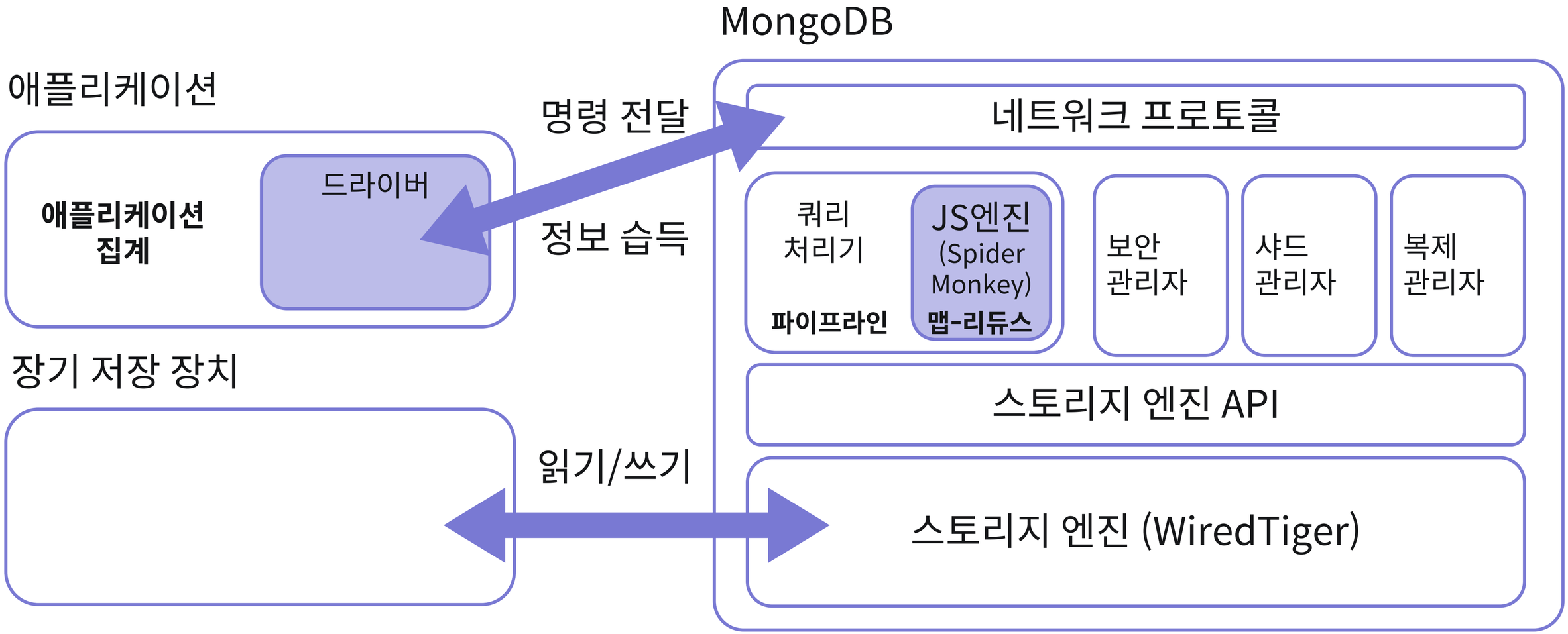
1. 데이터베이스의 정보를 불러와 **애플리케이션 단계에서 집계**하는 방법
2. MongoDB의 **맵-리듀스** 기능을 이용하는 방법
3. MongoDB의 **집계 파이프라인** 기능을 이용하는 방법

집계 명령의 특징



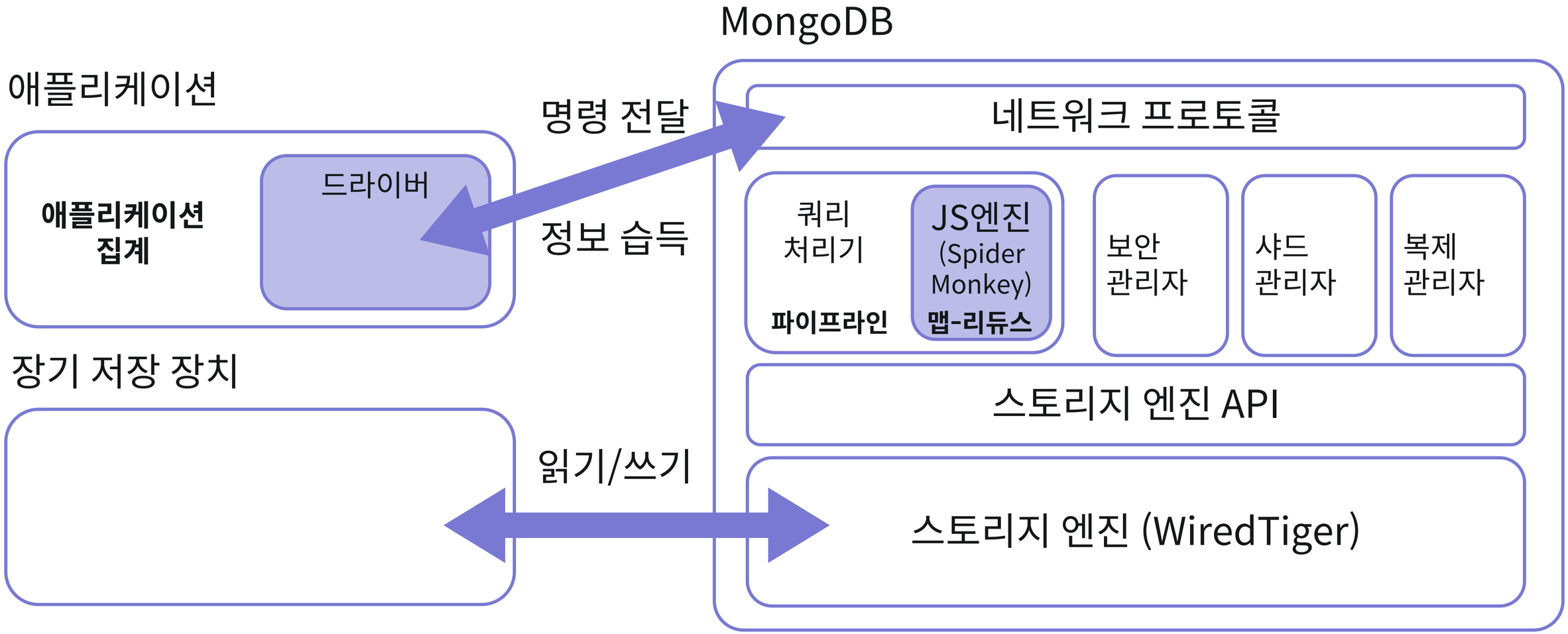
원본 데이터보다 결과 데이터의 양이 더 적다
집계 연산을 데이터 처리 초기 단계에서 할수록 유리하다

MongoDB와 웹 클라이언트의 통신 구조



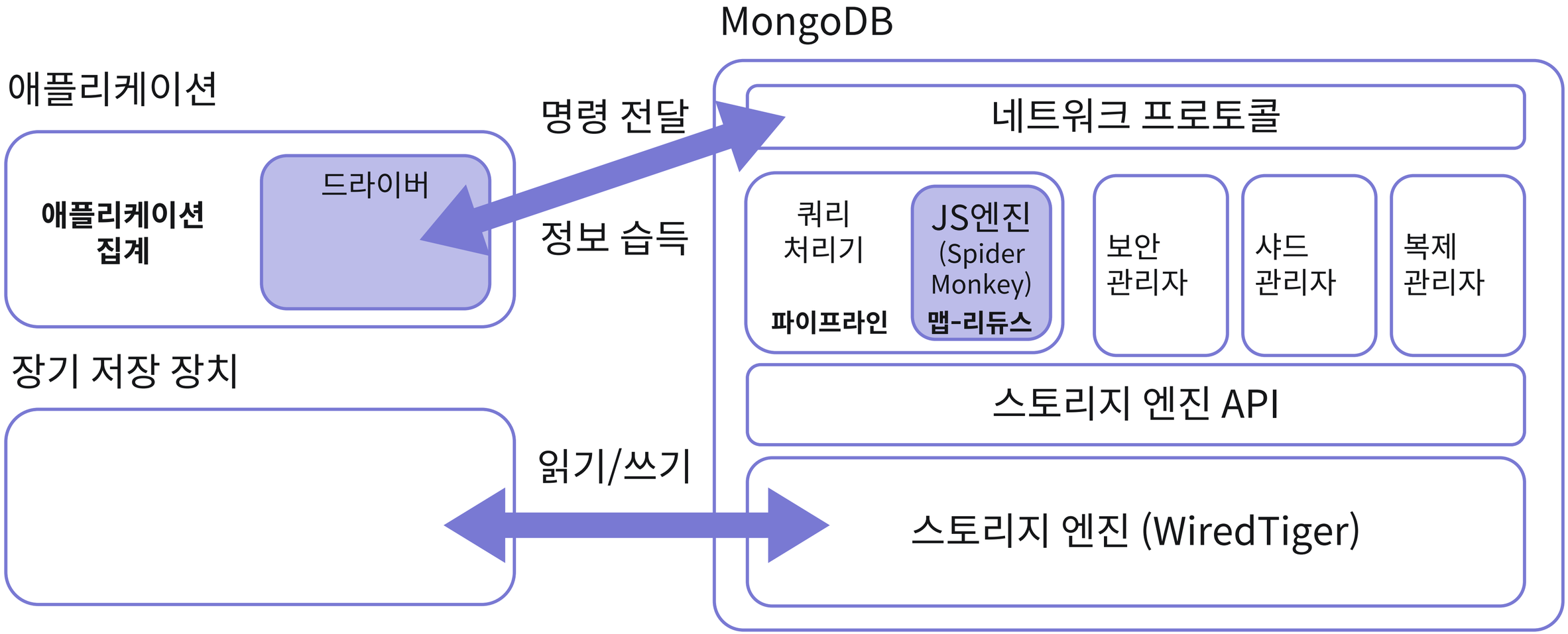
처리 위치: 애플리케이션 내부, JS 엔진, MongoDB 내부

집계 방식에 따른 처리 속도



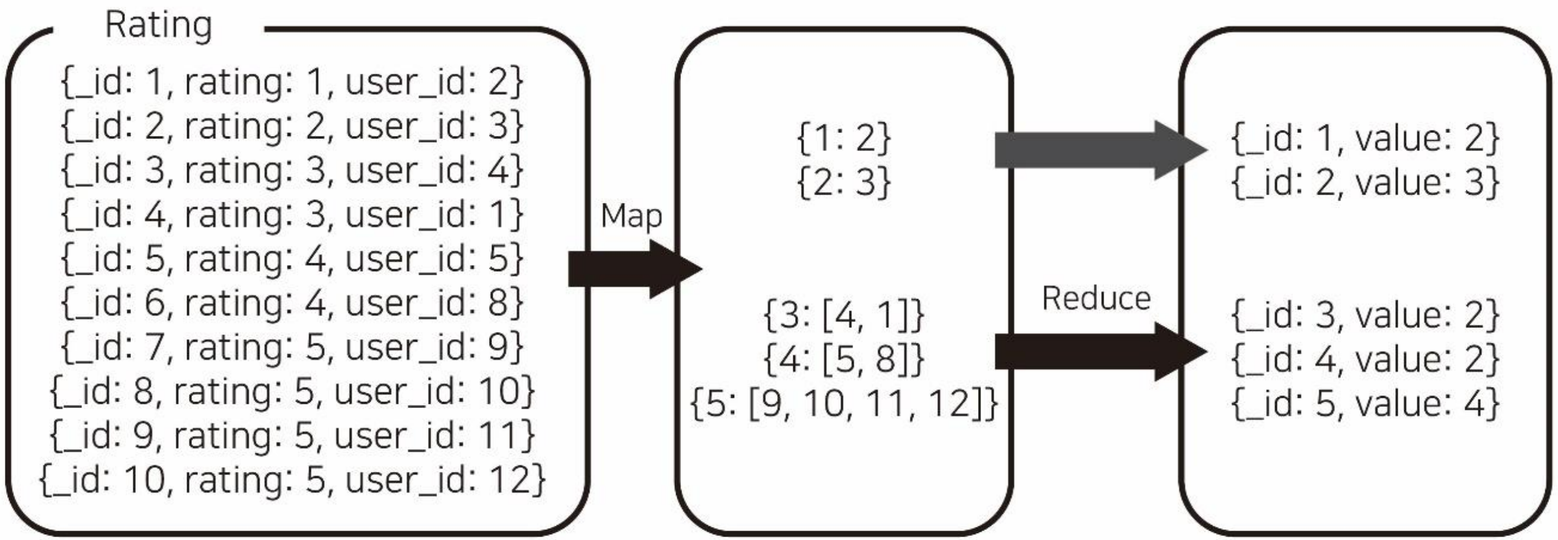
처리 속도: 애플리케이션 < 맵-리듀스 < 집계 파이프라인

집계 방식에 따른 자유도



자유도: 애플리케이션 > 맵-리듀스 > 집계 파이프라인

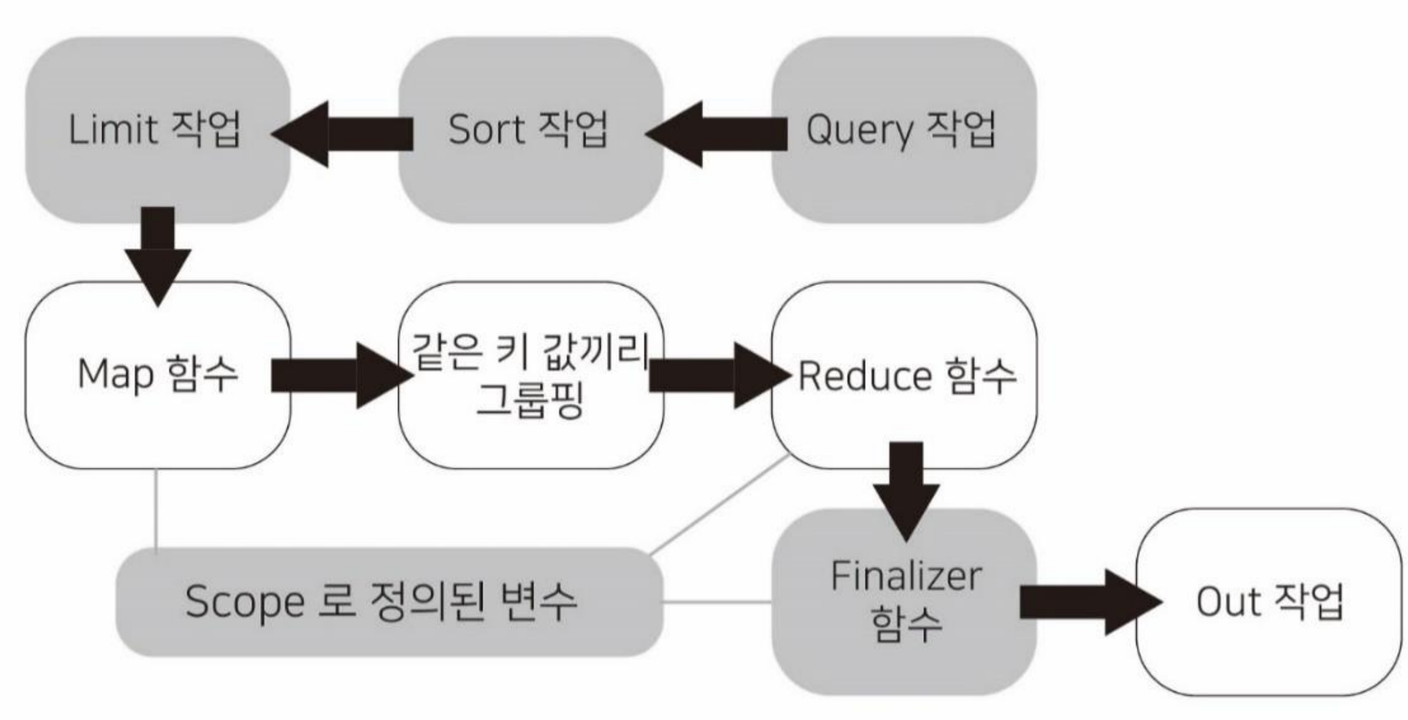
맵-리듀스 소개



맵핑함수: 관련된 정보끼리 그룹화

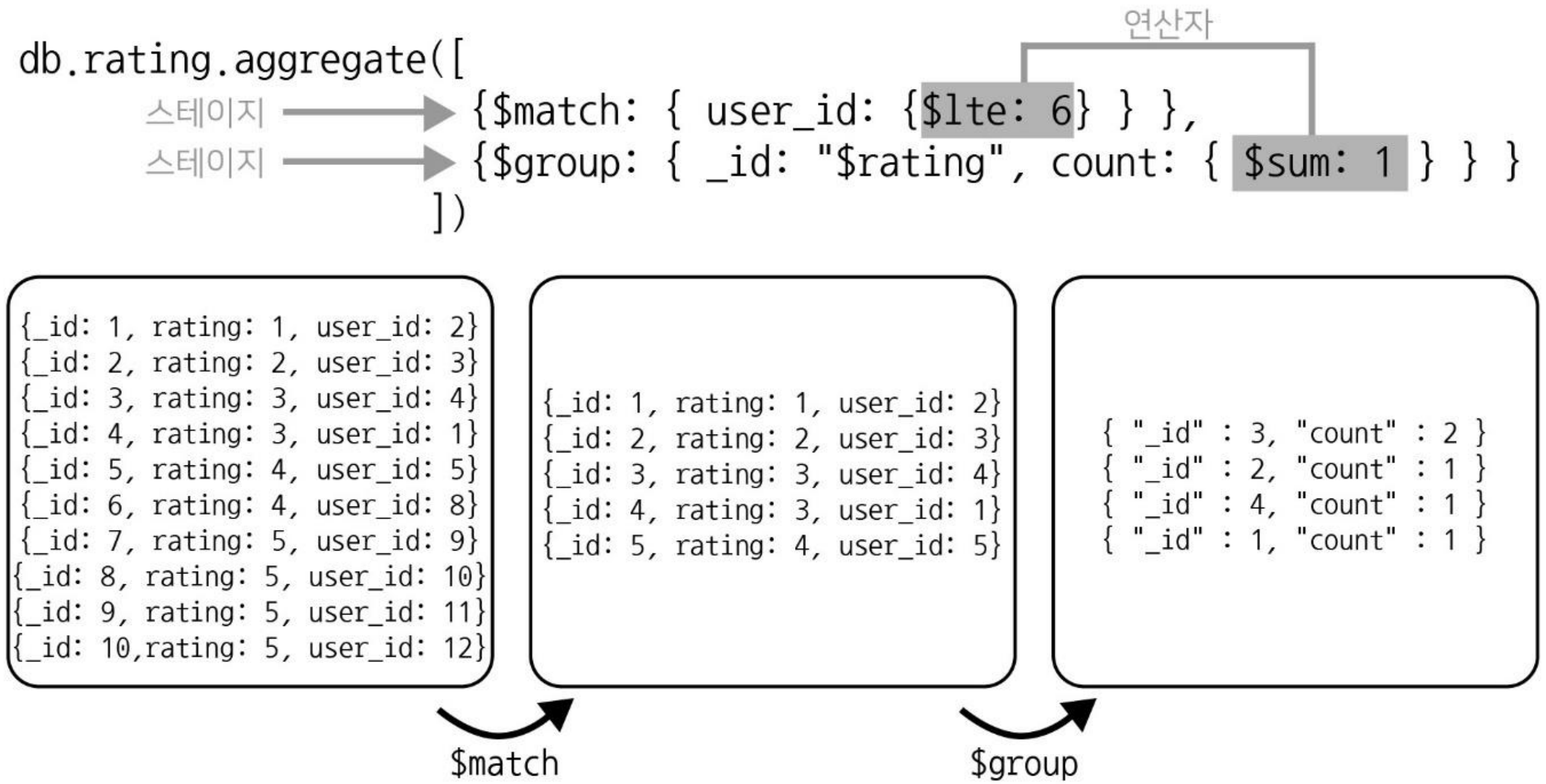
리듀스 함수: 그룹 내 정보들을 집계 연산 (ex. 평균, 길이, 합산 ...)

맵-리듀스 함수의 데이터 처리 과정



흰색으로 표시된 단계가 필수 단계

집계 파이프라인 소개



파이프라인이란

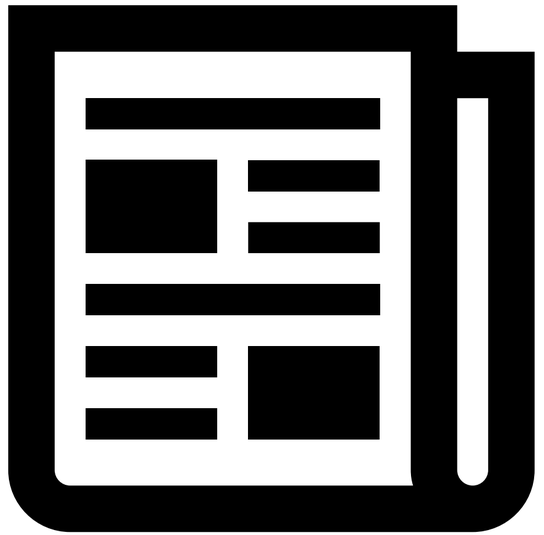
한 데이터 처리 단계의 출력이 다음 단계의 입력으로 연결된 구조

03

인덱스 개요

프레드는 기사 수집하는

취미를 가지고 있다



하지만 매번 한 페이지씩 넘기며 기사를 찾는다

기사를 나중에 쉽고 빠르게 찾기 위한 방법?

색인이나 목차를 만드는 것!

- 국내 체류 외국인, 건보료 체납 때 의료비 전액 본인 부담 | 32
- 과민성 장 증후군, 소변검사로 진단 | 55
- 과천시, 관악산·청계산 생태길 정비사업 추진 | 91
- ...

제목을 기준으로 한 색인

하지만 일반적인 경우 기사의 분류 별로 찾는 일이 더 많았다

- 경제 | 1, 10, 67, …
- 사회 | 4, 32, 56, 91, …
- 과학 | 3, 55, 88, …
- …

분류를 기준으로 한 색인

하지만 분류만으로 기사를 찾기에는 너무 불편했다
또한 매번 기사가 추가될 때마다 색인을 수정해야 한다

- 경제
  - 비트코인 반감기에 쏠리는 기대감…분석가 "바닥 쳤다" 확신 | 1
  - 버핏의 아내 위한 유언장엔… 인덱스 펀드에 90% 투자하라 | 10
  - 산업은행 200억 투자한 '화승', 분식회계 의혹 | 67
- 사회
  - 국내 체류 외국인, 건보료 체납 때 의료비 전액 본인 부담 | 32
  - 과천시, 관악산·청계산 생태길 정비사업 추진 | 91

…

분류-제목을 기준으로 한 색인

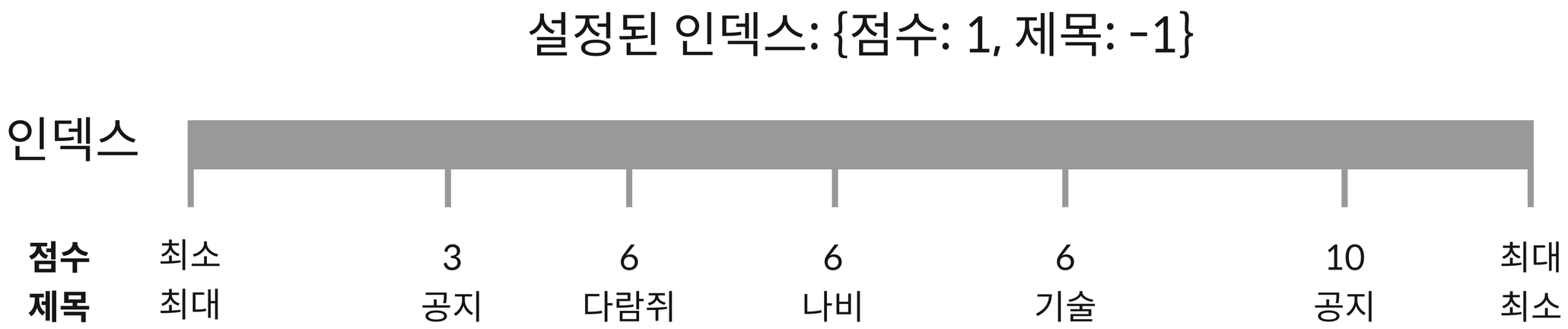
분류만으로 찾을 때도 쓸 수 있지만 가나다순으로 찾을 순 없다

- 국내 체류 외국인, 건보료 체납 때 의료비 전액 본인 부담
  사회 | 32
- 과민성 장 증후군, 소변검사로 진단
  과학 | 55
- 과천시, 관악산·청계산 생태길 정비사업 추진
  사회 | 91

제목-분류를 기준으로 한 색인

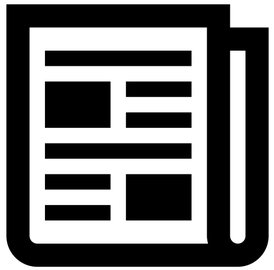
순서가 바뀌니 분류를 기준으로 찾기 힘들다

인덱스의 기능



인덱스는 앞선 예시의 색인과 거의 같은 기능을 수행한다
인덱스는 검색과 순서 정렬을 효율적으로 만들어준다

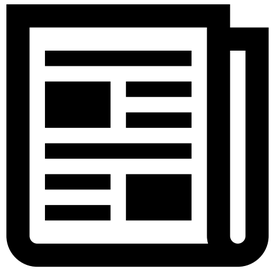
앞선 예시에서 알 수 있는 인덱스의 특징



색인이 없으면
매번 한 페이지씩 넘기며 기사를 찾을 수밖에 없다

쿼리를 수행할 때 인덱스가 없다면 모든 도큐먼트를 일일이 조회해야 한다
인덱스는 쿼리 작업을 매우 **효율적**으로 만든다

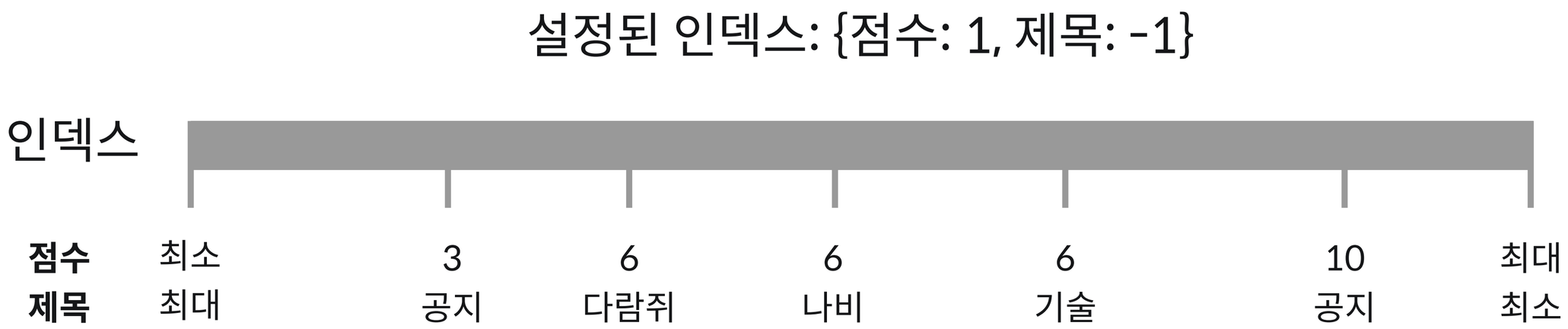
앞선 예시에서 알 수 있는 인덱스의 특징



매번 기사가 추가될 때마다 색인을 수정해야 한다

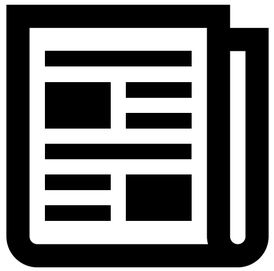
인덱스를 만들면 도큐먼트 생성 수정 시
인덱스를 업데이트해야 하기 때문에 **속도 저하**가 있다

인덱스의 종류



- 단순 인덱스: 하나의 필드를 기준으로 생성한 인덱스
- 복합 인덱스: 다수의 필드를 기준으로 생성한 인덱스

앞선 예시에서 알 수 있는 인덱스의 특징

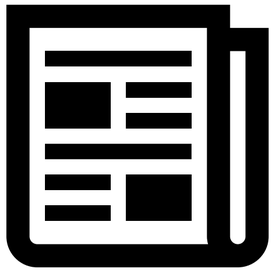


제목으로 찾으려면 제목 색인이 필요하고,
분류와 제목으로 찾으려면 분류-제목 색인이 필요하다

하나의 필드만 조회할 때는 **단순 인덱스**로 충분하지만

다수의 필드를 대상으로 조회를 할 때는 **복합 인덱스**가 유용하다

앞선 예시에서 알 수 있는 인덱스의 특징



분류-제목을 기준으로 한 색인은
분류만으로 찾을 때도 쓸 수 있지만 가나다순으로 찾을 순 없다

a-b 복합 인덱스는 a 단순 인덱스와 같은 기능을 하므로
대체할 수 있다

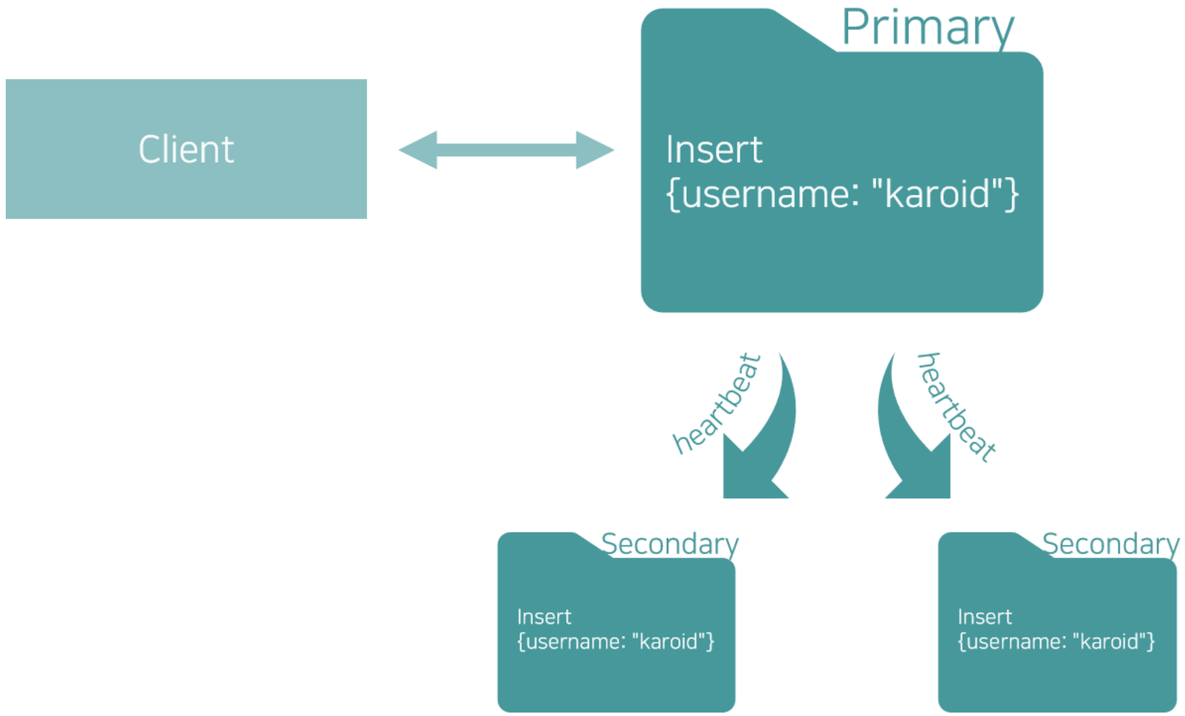
정리

1. 인덱스는 쿼리 작업을 매우 **효율적**으로 만든다
2. 인덱스를 만들면 도큐먼트 생성 수정 시 **속도 저하**가 생긴다
3. 다수의 필드를 대상으로 조회를 할 때는 **복합 인덱스**가 유용하다
4. a-b **복합 인덱스**는 a **단순 인덱스**와 같은 기능을 할 수 있다

04

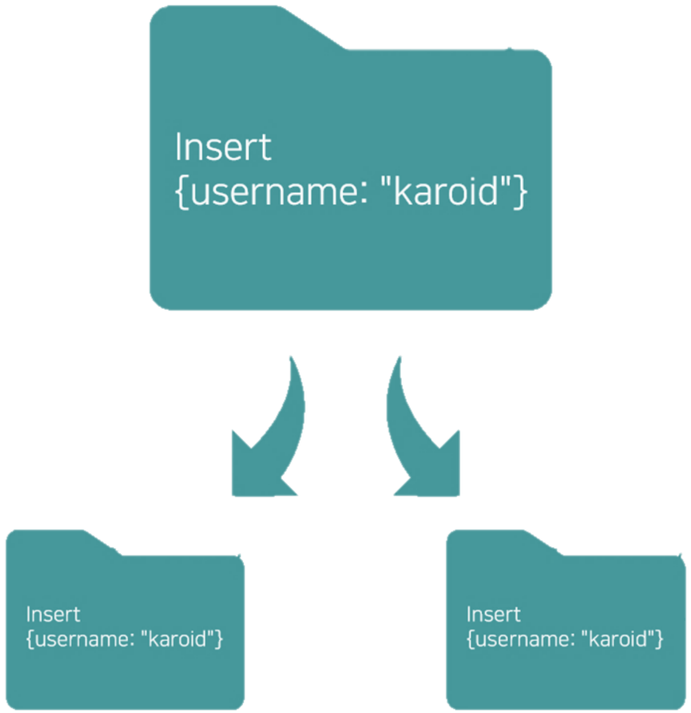
복제 세트 이해하기

복제 세트 정의



복제 세트는 같은 정보를 공유하는 Data Set이다

복제 세트는 왜 만들까?

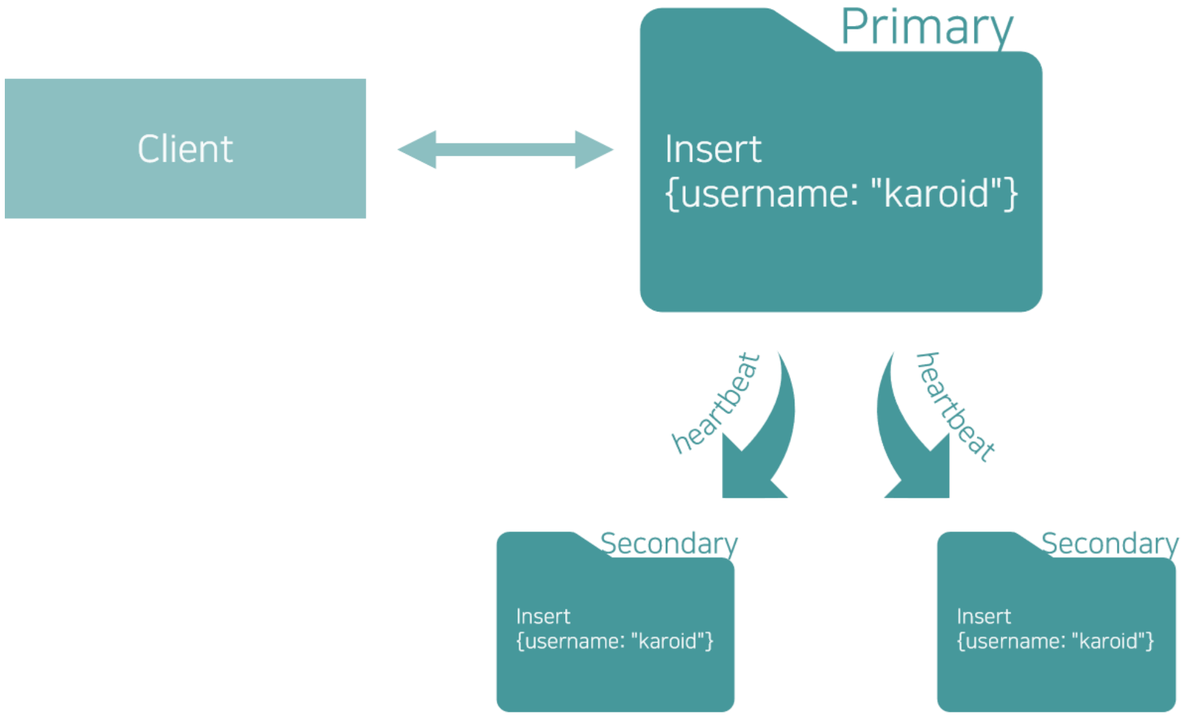


왜 굳이 복제를 할까?

1. 높은 가용성을 위해서!
2. 정보의 안전한 보호를 위해

3. Read 속도를 빠르게 하기 위해서

복제 세트의 구성원

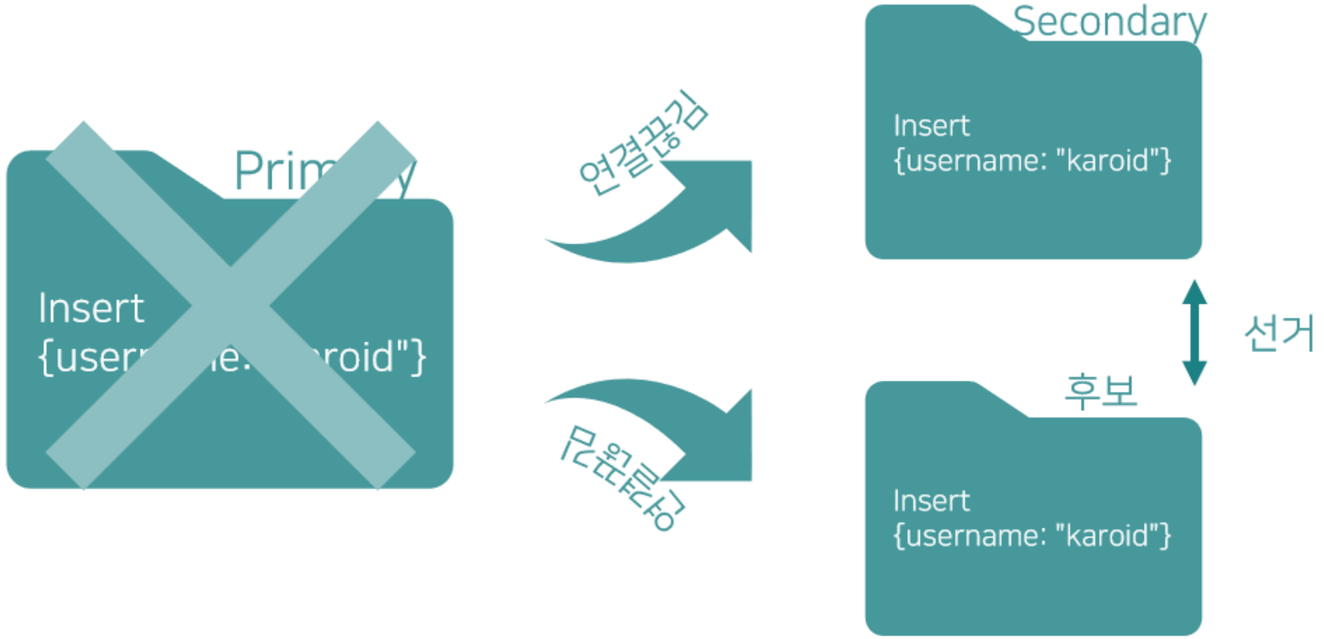


클라이언트로부터 직접
RW 요청 수행

프라이머리의 정보 상태를 갱신하며
복제 상태를 유지

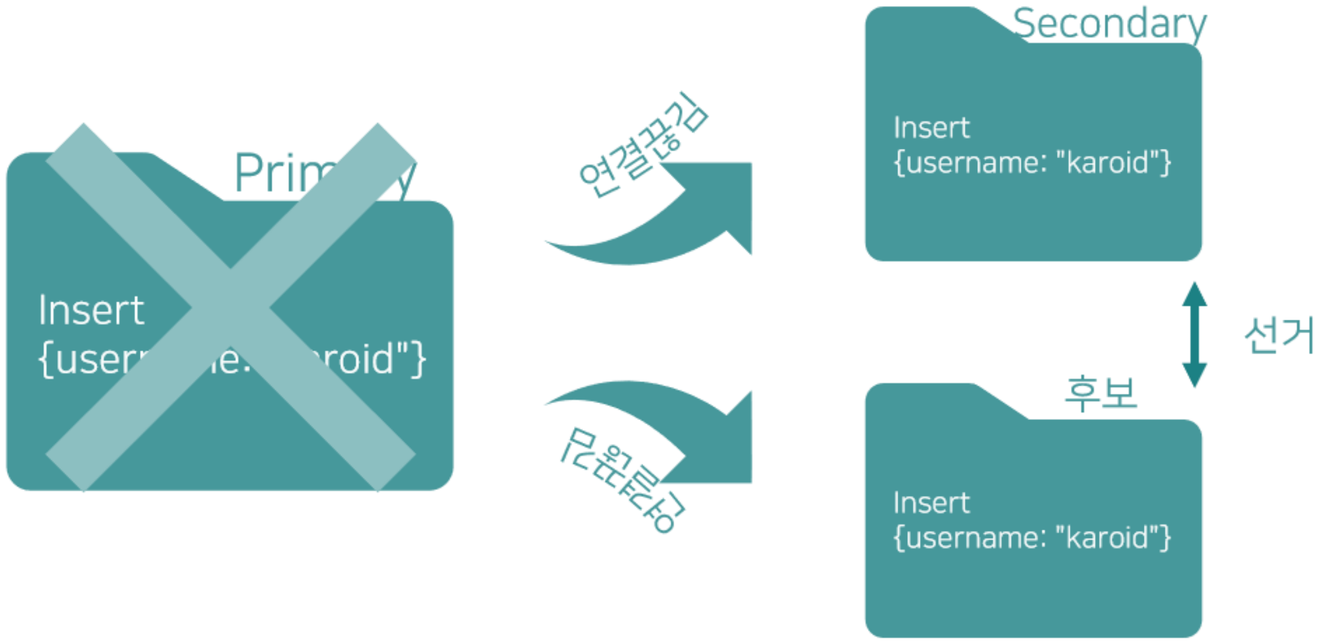
복제 세트는 프라이머리, 세컨더리, 아비터 구성원으로 이루어져 있다
Heartbeat으로 서로의 상태를 확인한다

복제 세트의 선거



프라이머리가 무슨 이유에서든지 죽게 되면,
복제 세트 구성원 중 과반수의 세컨더리가 이를 감지하여
선거를 개최하기로 결정한다

복제 세트의 선거



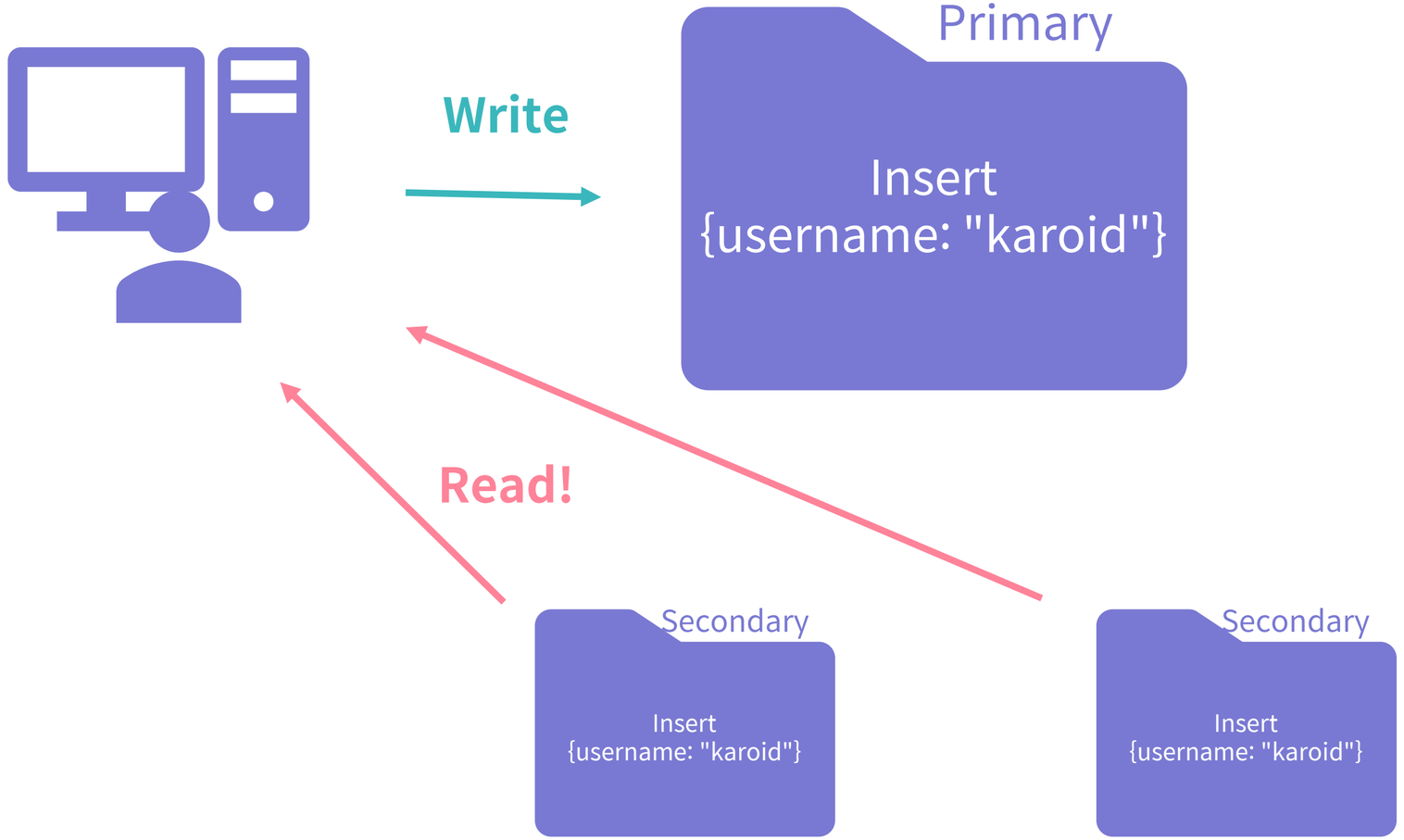
세컨더리와 아비터는 새로운 프라이머리를 뽑는 **투표**를 하게 된다
우선순위가 높은 순서대로 세컨더리는 프라이머리 후보가 되고,
과반수의 찬성표를 받은 세컨더리는 **프라이머리가 된다**

복제 세트의 선거

만약 별문제가 없으면 투표권자들은 **찬성표**를 던지지만,
다음과 같은 경우에는 **반대표**를 던진다

primary가 아직 제대로 작동하는데?
후보 너보다 내가 더 최신 데이터를 전달받았어
… 등등

복제 세트의 또 하나의 장점



세컨더리를 활용해 읽기 기능을 확장할 수 있다

정리

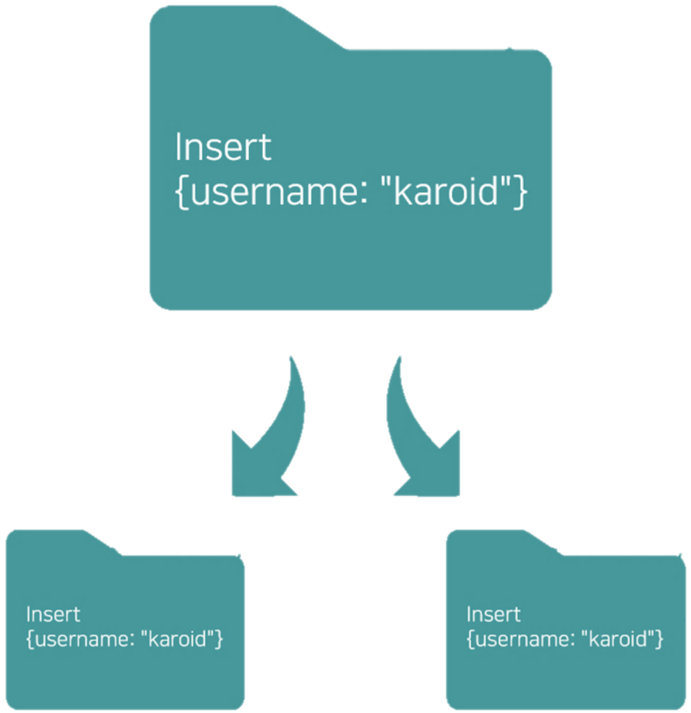
복제 세트를 만드는 이유

1. 높은 가용성
2. 정보의 안전한 보호
3. Read 속도를 빠르게 하기 위해서

05

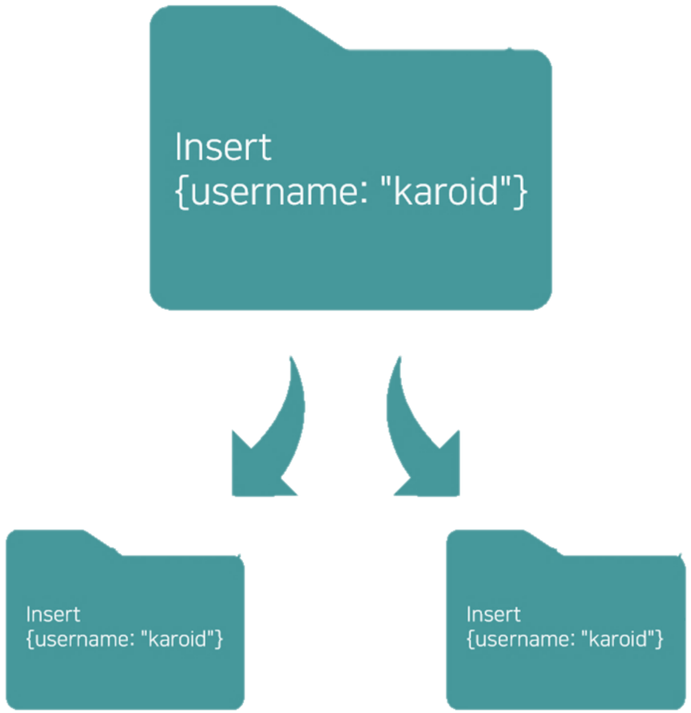
Read-Concern과 Write-Concern

Read-Concern과 Write-Concern이 필요한 이유



복제 세트에서 구성원의 정보가 동기화되는
데에는 필연적으로 **시간이 필요**

Read-Concern과 Write-Concern이 필요한 이유



Read-Concern

어느 정도 동기화 수준을 기준으로
쓰기 작업을 마무리할지 설정

Write-Concern

어느 정도 동기화 수준을 기준으로
정보를 읽어올지 설정

Read-Concern과 Write-Concern 설정 방법

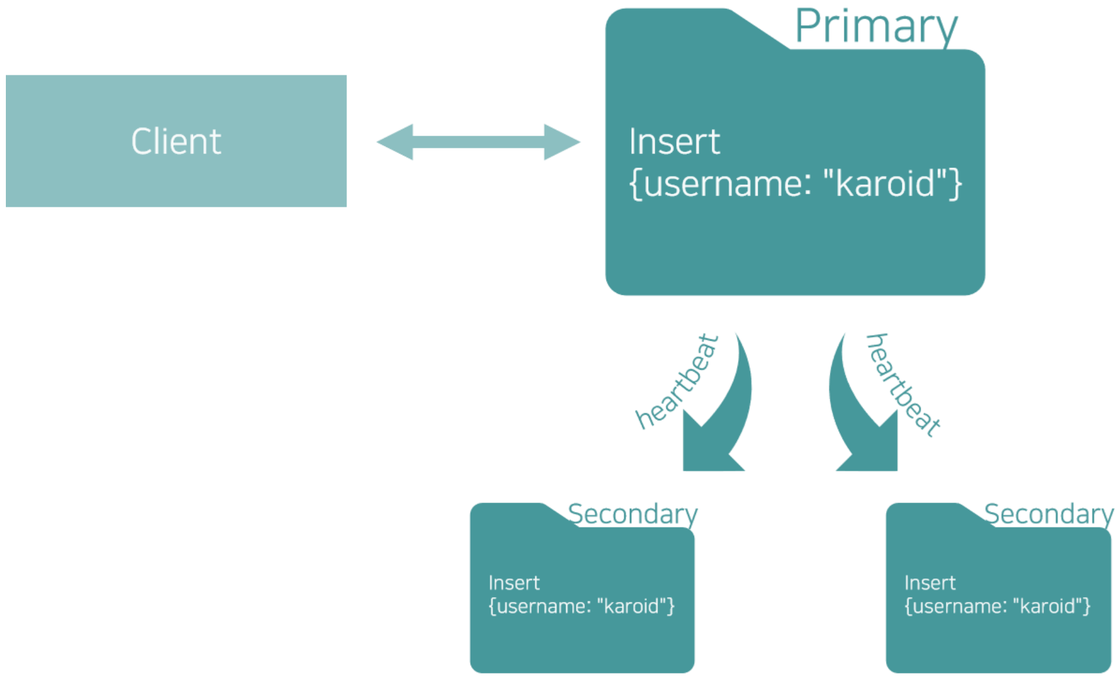
백엔드

```
import pymongo
from pymongo.write_concern import WriteConcern
from pymongo.read_concern import ReadConcern

client = pymongo.MongoClient('localhost', 27017)
db = client.get_database("elice")

rc = ReadConcern(level='majority')
wc = WriteConcern(w=1, wtimeout=200, j=True)
col = db.get_collection("post", write_concern=wc, read_concern=rc)
```

Read-Concern 설정

설정 예시

```
ReadConcern(level='majority')
```



local

연결된 인스턴스에서만 정보를 불러온다

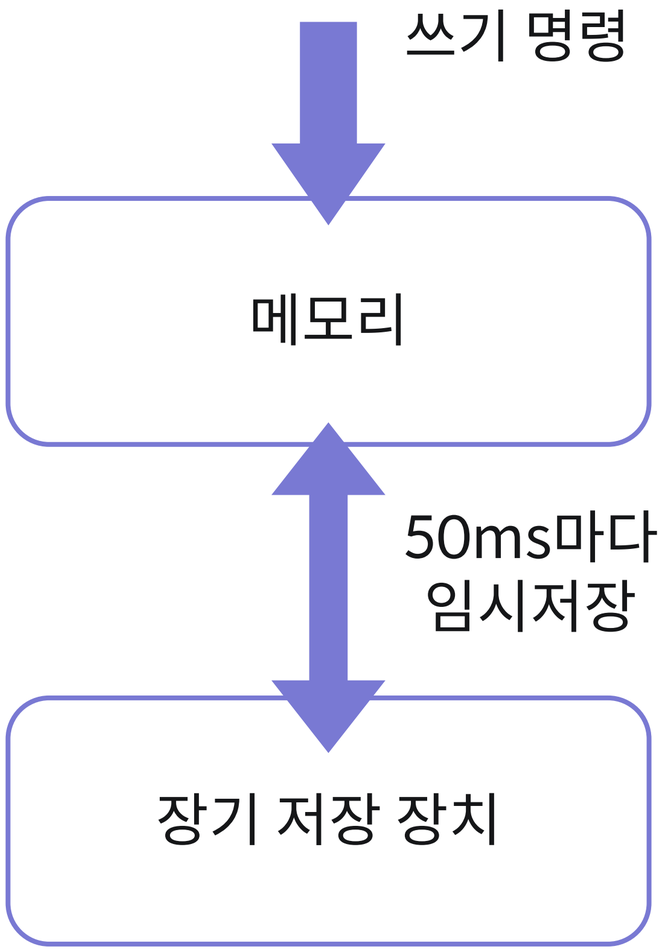
majority

복제 세트 대다수에 저장된 정보로 불러온다

linearizable

시간 제한 내에 복제 세트 구성원의 정보를
확인해서 대다수에 저장된 정보로 불러온다

MongoDB의 쓰기 작업과 저널링



메모리에 정보를 저장하는 시간이
디스크에 저장하는 시간보다 훨씬 빠르다

쓰기 작업은 메모리에 변경 사항을 남겼다가 일정
주기(50ms)로 디스크에 **변경 사항**을 기록한다

이처럼 디스크에 변경사항을 임시로 저장하는
작업을 **저널링**이라고 한다

Write-Concern과 저널링

설정 예시

```
WriteConcern(w=1, wtimeout=200, j=True)
```

필드	설명
w	복제 세트의 어느 정도의 구성원에 쓰기 작업이 완료되어야 전체 쓰기 작업이 완료되었다고 판단할지 결정하는 옵션, 숫자나 문자열로 지정할 수 있다
j	이 옵션이 true 값을 가지면 변경 사항을 바로 저널링해서 만약 장애가 발생하더라도 문제가 없게 만든다, 기본값은 false
wtimeout	w 옵션에서 설정한 구성원들을 **기다릴 수 있는 최대 시간(ms)**, 주어진 시간이 지나도 정해진 구성원에 쓰기 작업이 마무리되지 않으면 에러를 반환하지만 이미 실행한 쓰기 작업 자체는 취소되지 않는다 쓰기 작업이 무한정 지연되는 것을 막기 위한 옵션으로 w 값이 1보다 커야 사용할 수 있다

Write-Concern의 W 옵션

값	설명
0	쓰기 작업이 실제로 수행됐는지 확인하지 않고 쓰기 작업을 완료한다
1	(기본값) 클라이언트와 연결된 인스턴스의 쓰기 작업을 수행하면 전체 쓰기 작업이 완료된다
1 보다 큰 자연수	값으로 갖는 숫자가 복제 세트에서 쓰기 작업을 완료한 구성원 수와 같으면 전체 쓰기 작업이 완료된다 예를 들어 프라이머리 1개와 세컨더리 2개로 이루어진 복제 세트에서 w: 2로 설정되었다면 쓰기 작업은 한 개의 프라이머리와 한 개의 세컨더리에서 쓰기 작업이 실행되면 전체 쓰기 작업이 완료된다
majority	복제 세트에서 대다수의 구성원이 쓰기 작업을 수행하면 전체 쓰기 작업이 완료된다

정리하기

Read-Concern

```
ReadConcern(level='majority')
```

어느 정도 동기화 수준을 기준으로
정보를 읽어올지 설정

Write-Concern

```
WriteConcern(w=1, wtimeout=200, j=True)
```

어느 정도 동기화 수준을 기준으로
쓰기 작업을 마무리할지 설정

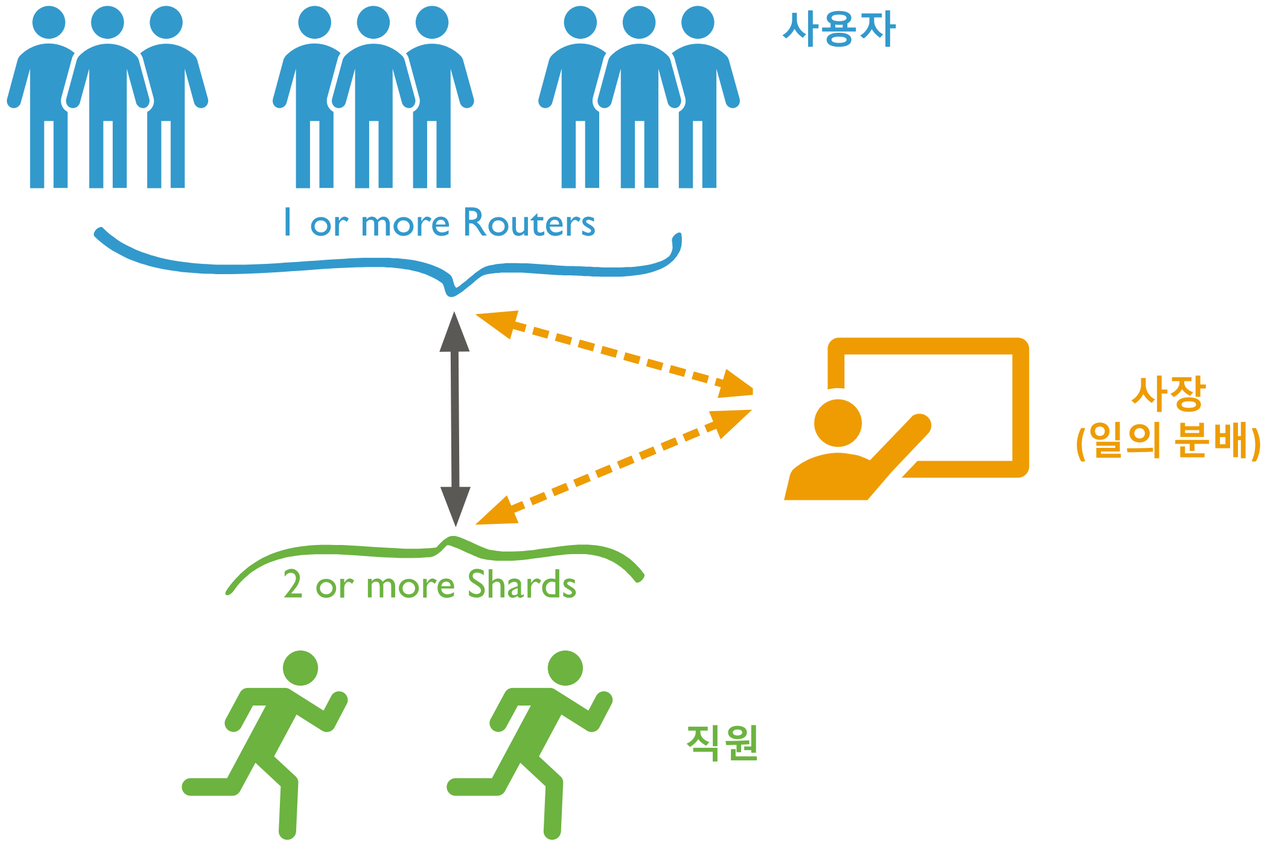
06

샤드 클러스터 이해하기

샤드 클러스터는 왜 필요한가?

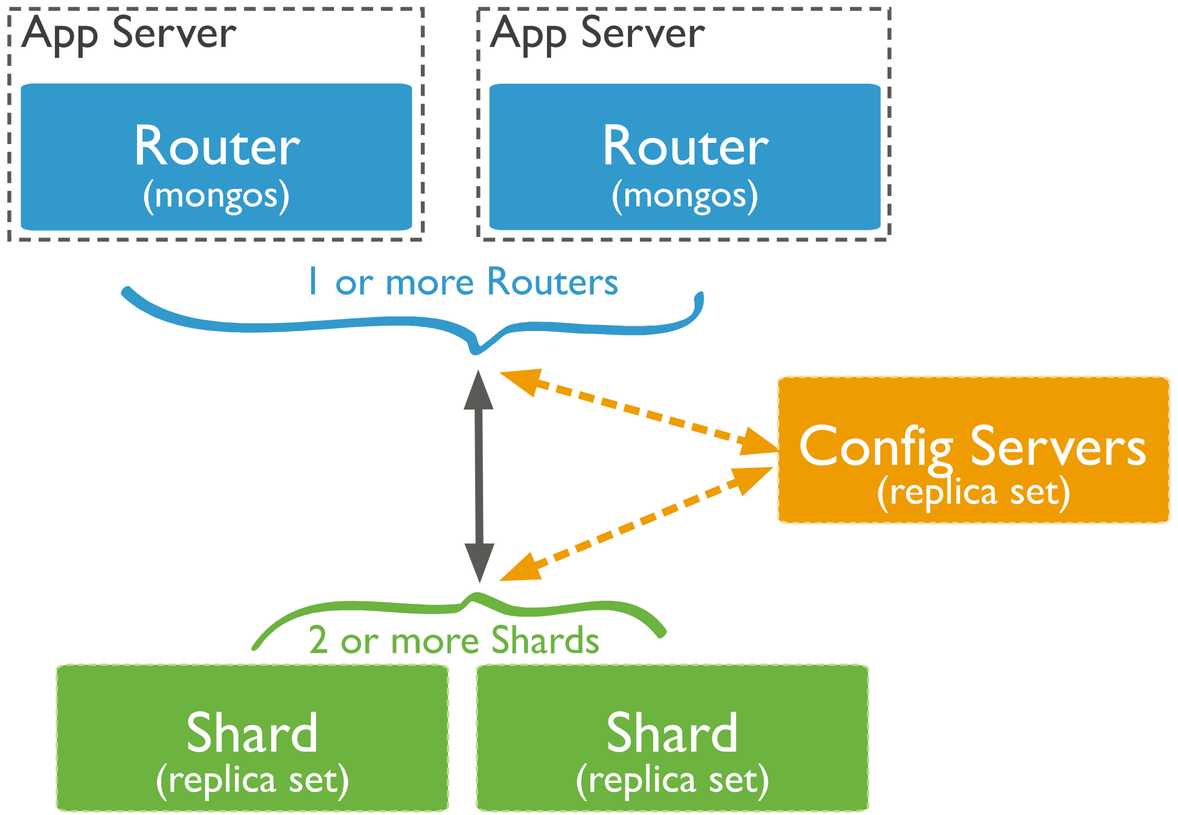
컴퓨터의 모든 장비는 고성능이 될수록 가성비가 안 좋아진다!
성능 2배 좋은 서버를 쓰는 것보다 2개의 서버를 쓰는 게 경제적이다

샤드 클러스터의 구성



분업을 위해서는 지휘자가 필요하다

샤드 클러스터의 구성

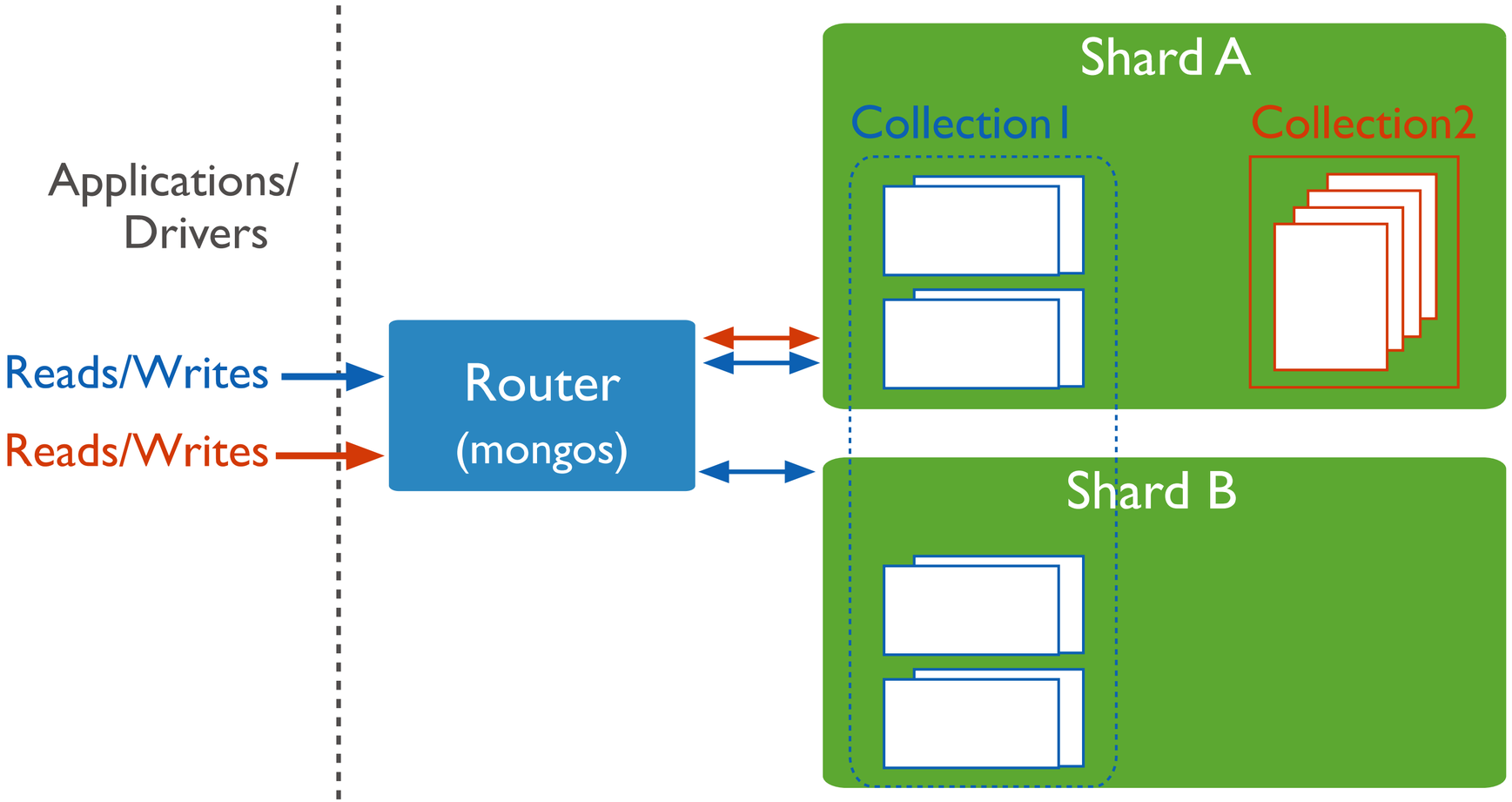


클러스터의 설정값과

샤드의 메타데이터 보유

분산된 정보를 가지고 있다

샤드 클러스터의 작동 방식



샤딩의 기준

범위 샤딩 **해시 샤딩** **구역 샤딩**

샤딩은 특정 필드 값을 기준으로 정보를 분산시킨다
3가지 기준으로 도큐먼트를 분산시킬 수 있다

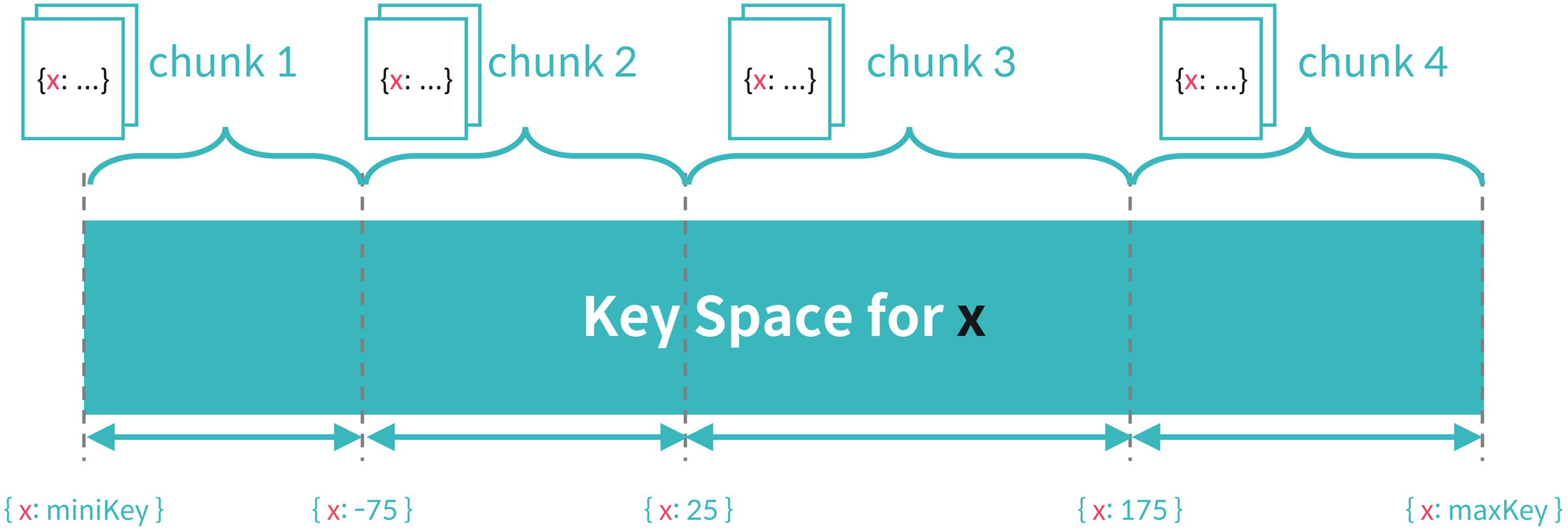
잘된 도큐먼트 분산의 기준

잘된 샤딩의 예

샤딩을 했을 때 일어나는 일반적 현실

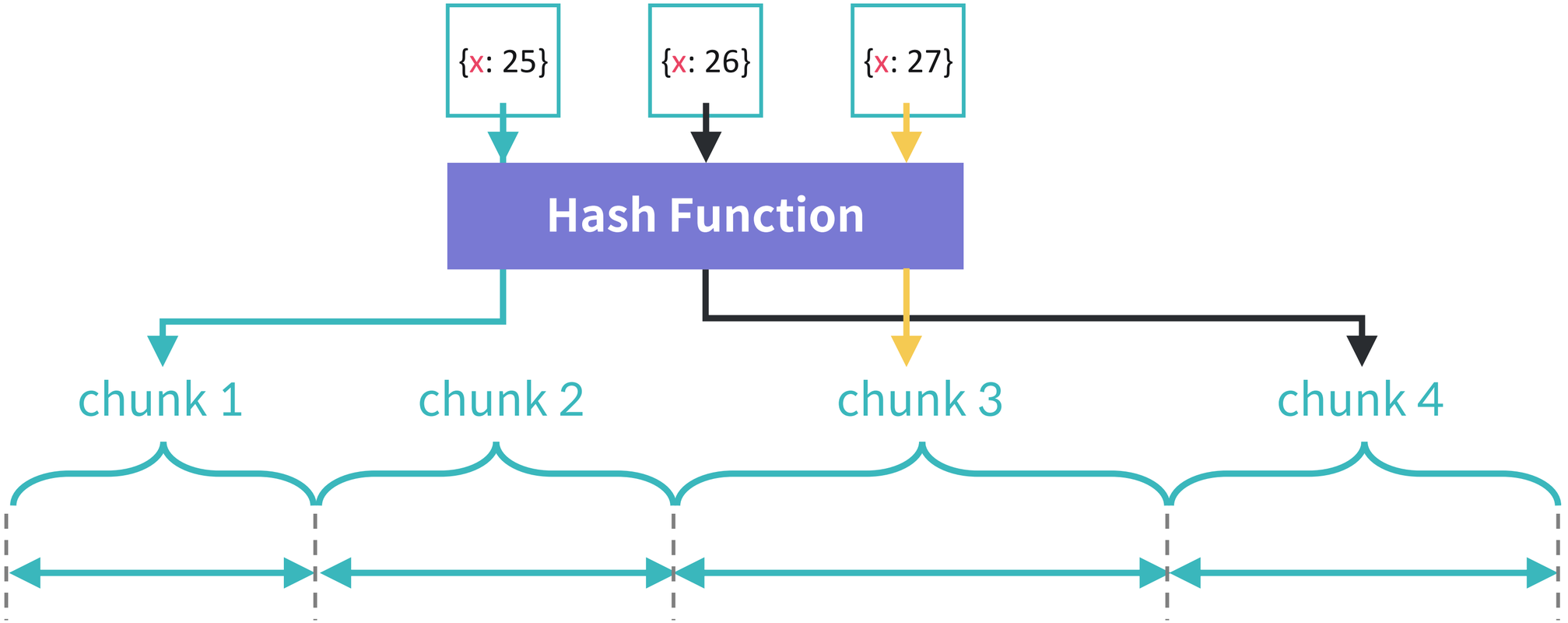
샤드에 정보가 골고루 분산되어서 연산이 골고루 이루어져야 한다

범위 샤딩



범위에 따라 분산하기 때문에 범위 조회 시 유리하다
하지만 특정 범위에 도큐먼트가 쏠리면 해당 샤드만 계속 바빠질 수 있다

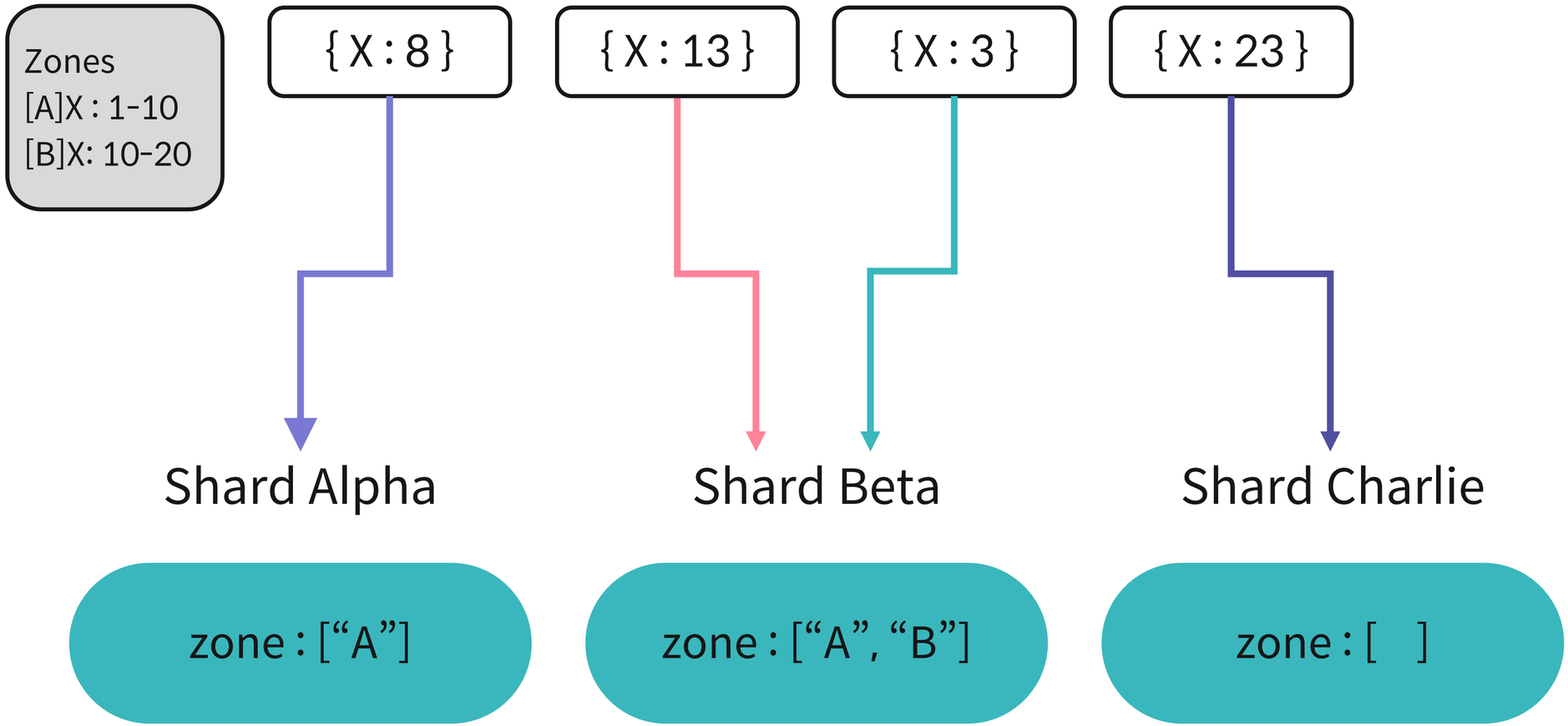
해시 샤딩



장점: 두루두루 저장되어서 한쪽에 몰릴 가능성이 낮다

단점: 범위를 찾는 쿼리를 수행할 때마다 다수의 클러스터에서 찾아야한다

구역 샤딩



개발자가 존을 정해서 분배하는 방식
손이 많이 가지만 최적화된 샤드 구성을 만들 수 있다

강의 마무리 인사

지금까지 강의를 시청해 주셔서 감사합니다

크레딧

/* elice */

코스 매니저
이재성

콘텐츠 제작자
정승호

강사
정승호

감수자
정승호

디자이너
강혜정

연락처

TEL
070-4633-2015

WEB
https://elice.io

E-MAIL
contact@elice.io