Data Science

Big Data Analysis

노기섭 교수

(kafa46@hongik.ac.kr)

Lecture Goals

- 빅데이터 소개
- Hadoop의 배경 및 구성요소
- Spark 등장 배경 및 개념
- Spark 활용한 간단 실습

빅데이터 소개

빅데이터의 역사와 발전, 그리고 중요성

■ **2000년대 중반부터 본격 등장, 데이터 폭증**

- 스마트폰·인터넷·SNS 확산과 함께 정형 데이터 중심에서 영상, 로그, IoT 등 다양한 데이터로 확대

■ **기존 RDBMS 한계로 분산처리 기술 등장**

- 데이터 규모·속도·다양성(3V)을 감당하기 어려워져 Google 논문 기반 오픈소스 Hadoop 생태계가 구축

■ **빅데이터의 진정한 가치는 분석과 활용**

- 데이터를 수집·저장하는 것보다, 의미 있는 인사이트 도출과 데이터 기반 의사결정(Data-Driven)이 핵심 가치

■ **다양한 산업에 필수 경쟁력으로 자리 잡음**

- 추천 시스템, 마케팅, 금융 보안, 제조 품질, 의료 진단 등 산업 전반에서 빅데이터 분석이 성과와 경쟁력을 좌우

■ **AI와 결합하며 지능형 데이터 활용 시대로 진화**

- 머신러닝·딥러닝과 실시간 분석 기술 결합으로 초개인화 서비스, 자동화, 자율 시스템, 디지털 트윈 등으로 확대 중

빅데이터와 AI의 상호 발전 관계

- **빅데이터는 AI의 학습 재료, AI는 빅데이터의 가치 증폭 도구**
  - 대규모 데이터는 머신러닝·딥러닝 성능 향상에 필수
  - AI는 방대한 데이터를 분석·가치화 → 시너지 효과
- **머신러닝·딥러닝의 발전으로 데이터 활용 방식이 혁신**
  - 통계 기반 분석에서 벗어나 대용량 데이터 패턴 분석·예측·의사결정 가능
  - 데이터 확보 및 관리 전략이 강화
- **빅데이터 기술과 AI의 결합이 표준화되며 지능형 자동화 시대로 진입**
  - Hadoop·Spark·클라우드 파이프라인 + AI 모델 학습·예측이 하나의 Data-AI Cycle 형성
  - 산업 전반의 지능화·자동화를 가속화

3V of Big Data

Volume (데이터 양)

- 데이터의 규모는 과거와 비교할 수 없을 만큼 폭발적으로 증가
  - 유튜브에는 매분 수백 시간 이상의 영상이 업로드
  - 삼성/TSMC 반도체 공정 라인에서 수집되는 센서 데이터는 하루 수 TB

Velocity (데이터 생성 및 처리 속도)

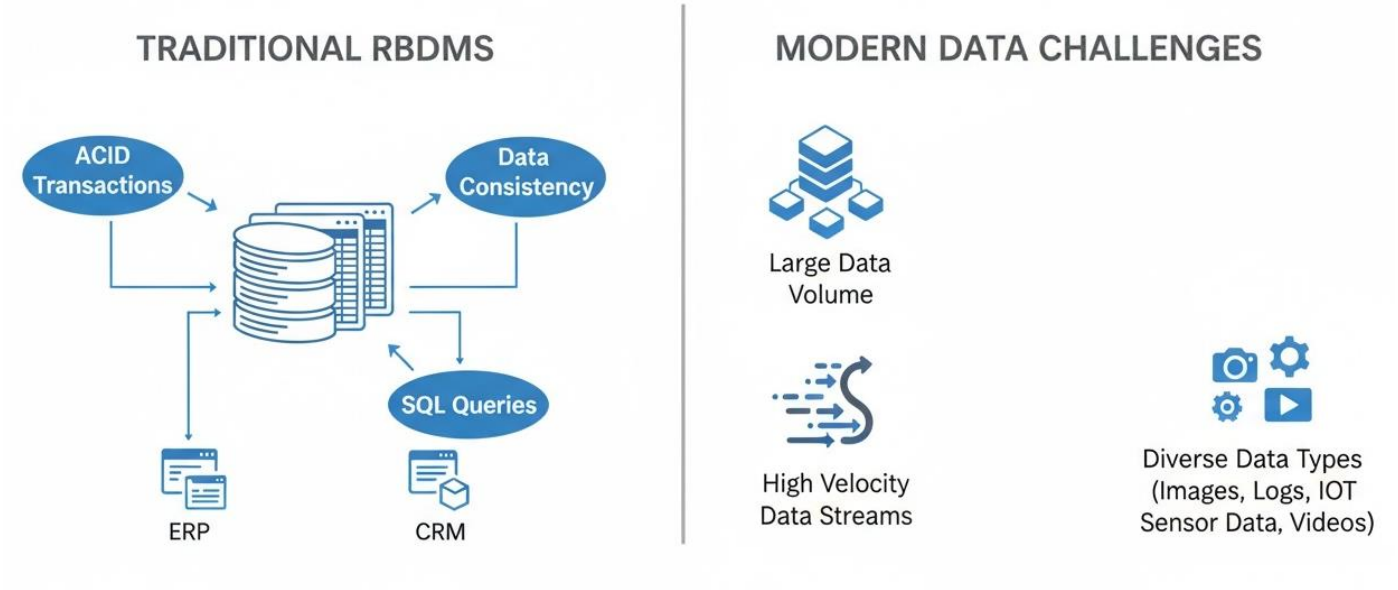
- 데이터는 매우 빠른 속도로 생성 + 실시간 처리
  - 금융권의 이상 거래 탐지 시스템은 거래 발생 즉시 분석하고 승인 여부를 판단
  - 자율주행차는 센서 데이터를 실시간 분석하여 주변 상황을 즉시 판단

Variety (데이터 형태의 다양성)

- 정형 데이터뿐 아니라 반정형, 비정형 데이터 등 다양한 형태로 존재
  - 카카오톡 대화 내용(텍스트), 사진·영상 파일, 위치 정보(GPS)는 서로 형태가 다른 데이터

전통 RDBMS와 제약사항의 등장

전통 RDBMS



- 기업 데이터 관리의 중심 기술로 오랫동안 사용
- 데이터 정합성, 트랜잭션(ACID) 보장, SQL 기반 질의 등 강력한 기능 제공
- 현재도 중요한 역할을 수행하고 있으나, 변화한 데이터 환경을 처리하기에는 한계 존재

[참고] 데이터베이스 - ACID 소개

Atomicity (원자성)

- 트랜잭션은 "모두 실행되거나, 전혀 실행되지 않거나" 둘 중 하나만 가능, 일부 적용은 불허
- 예: A → B로 10만 원 송금, 출금(–10만) + 입금(+10만) 두 단계지만하나라도 실패하면 둘 다 취소

Consistency (일관성)

- 트랜잭션 수행 전과 후에 데이터는 항상 규칙(제약조건)을 만족, 잘못된 데이터가 DB에 들어가는 걸 방지
    - 예: 음수(마이너스) 잔액 금지 규칙이 있다면, 트랜잭션 후에도 반드시 해당 규칙 준수

Isolation (고립성)

- 여러 트랜잭션이 동시에 실행돼도 서로의 중간 결과를 볼 수 없음, 각각 독립된 공간에서 실행되는 것처럼 동작
    - 예: A가 데이터 수정 중이면, B는 그 "수정 완료된 상태"만 볼 수 있어.

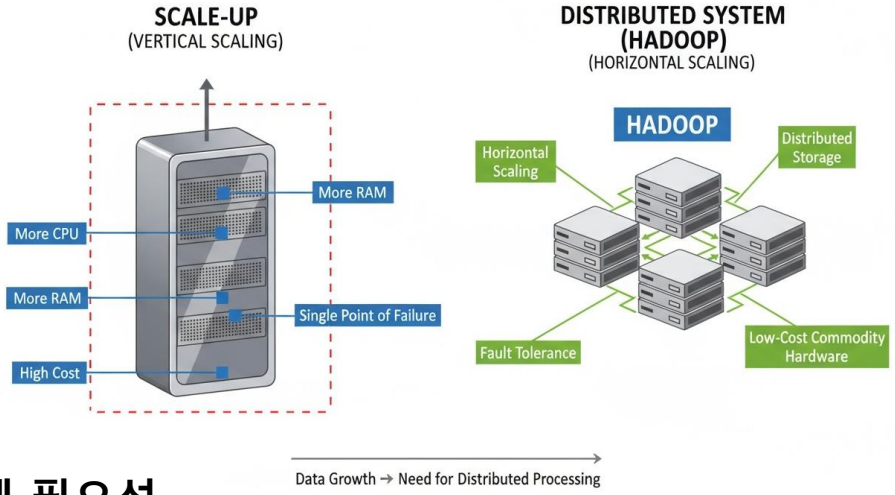
Durability (지속성)

- 트랜잭션이 성공적으로 끝나고 나면, 전원이 나가도 결과는 유지
- DBMS가 디스크에 안전하게 기록

전통 RDBMS의 한계

Scale-Up 한계

- RDBMS는 서버 성능을 높이는 방식(Scale-Up)에 의존
- 비용 증가와 단일 장애 위험으로 대규모 데이터 처리에 한계

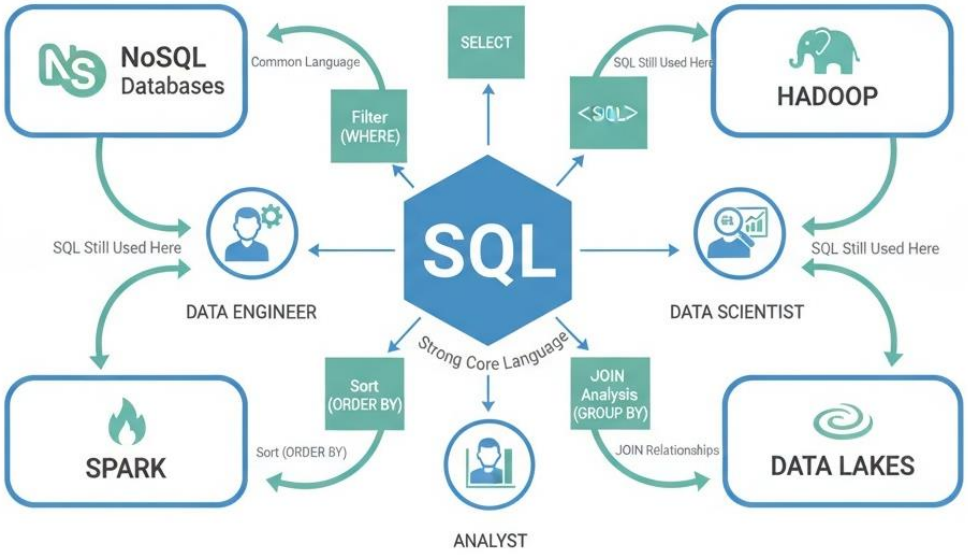


분산 시스템 필요성

- 데이터 폭증과 실시간 처리 요구 증가로 여러 서버에 분산 저장·처리하는 방식이 필요
- 이를 해결하기 위해 Google의 모델을 기반으로 Hadoop이 등장

SQL Basics for Big Data

SQL in Big Data



- 빅데이터 환경에서도 SQL은 여전히 가장 널리 사용되는 데이터 처리 언어
- 데이터 엔지니어, 데이터 과학자, 분석가 모두가 공통적으로 사용할 수 있는 표준 언어
- NoSQL, Hadoop, Spark 환경에서도 SQL 기반 처리 방식으로 확장 중
- 조회(SELECT), 조건 필터링, 정렬, 그룹 분석, 조인 등 다양한 기능을 제공

기초적인 SQL 문법 (1/3)

데이터베이스 테이블: **users**

name	age	city
Tom	25	Seoul
Alice	32	Busan
David	41	Incheon
Hannah	30	Daegu

데이터 조회 (SELECT)

```
SELECT name, age
FROM users;
```

```
| name    | age     |
|---------|---------|
| Tom     | 25      |
| Alice   | 32      |
| David   | 41      |
| Hannah  | 30      |
```

조건 조회 (WHERE)

```
SELECT name, age
FROM users
WHERE age >= 30;
```

```
| name    | age     |
|---------|---------|
| Alice   | 32      |
| David   | 41      |
| Hannah  | 30      |
```

정렬 (ORDER BY)

```
SELECT name, age
FROM users
ORDER BY age DESC;
```

```
| name    | age     |
|---------|---------|
| David   | 41      |
| Alice   | 32      |
| Hannah  | 30      |
| Tom     | 25      |
```

기초적인 SQL 문법 (2/3)

데이터베이스 테이블: **employees**

employee_name	department	salary
Tom	HR	4200
Alice	HR	4600
David	Sales	5100
Hannah	Sales	4900
Chris	Sales	5300
Emily	IT	6000
John	IT	6200
Sarah	IT	5800

그룹 분석
(GROUP BY + 집계함수)

```
-- 부서별 평균 급여를
-- 계산하여 출력
SELECT department, AVG(salary) AS avg_salary
FROM employees
GROUP BY department;
```

department	avg_salary
HR	4400
Sales	5100
IT	6000

기초적인 SQL 문법 (3/3)

데이터베이스 테이블: **customers**

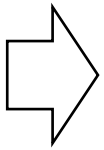
customer_id	customer_name	city
1	Tom	Seoul
2	Alice	Busan
3	David	Incheon
4	Hannah	Daegu

데이터베이스 테이블: **orders**

order_id	customer_id	product	amount
101	1	Keyboard	50
102	1	Monitor	200
103	2	Mouse	25
104	3	Laptop	1200
105	3	Headset	80
106	4	Webcam	70

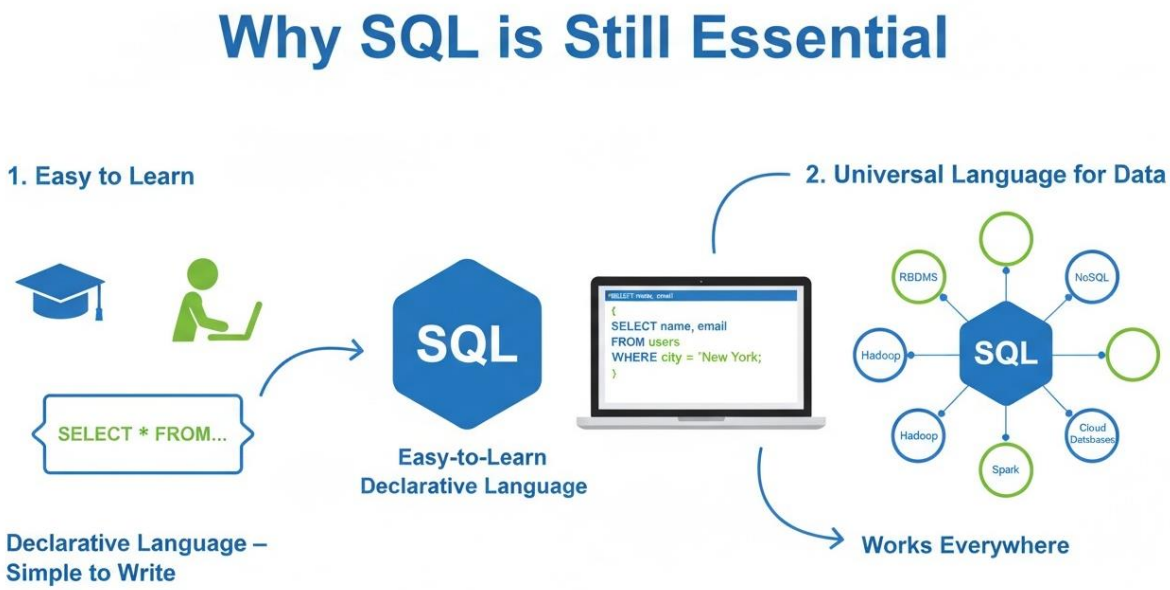
테이블 조인 (JOIN)

```
-- 주문 테이블과 고객 테이블을
-- customer_id 기준으로 조인하여
-- 주문 번호, 고객 이름, 주문 금액을 함께 조회
SELECT o.order_id, c.customer_name, o.amount
FROM orders AS o
JOIN customers AS c
   ON o.customer_id = c.customer_id;
```



```
| order_id | customer_name | amount |
|----------|---------------|--------|
|  101     |  Tom          |  50    |
|  102     |  Tom          |  200   |
|  103     |  Alice        |  25    |
|  104     |  David        |  1200  |
|  105     |  David        |  80    |
|  106     |  Hannah       |  70    |
```

왜 SQL은 여전히 중요한가?



- **Low Learning Curve: 배우기 쉽다.**
- **선언적 언어: 복잡한 데이터 처리 과정을 단순한 질의 형태로 표현 가능**
- **범용성: SQL은 데이터 분석의 범용 언어로 자리 잡음**
  - 다양한 데이터베이스와 빅데이터 엔진이 SQL 인터페이스를 채택

전통 SQL 발전의 한계

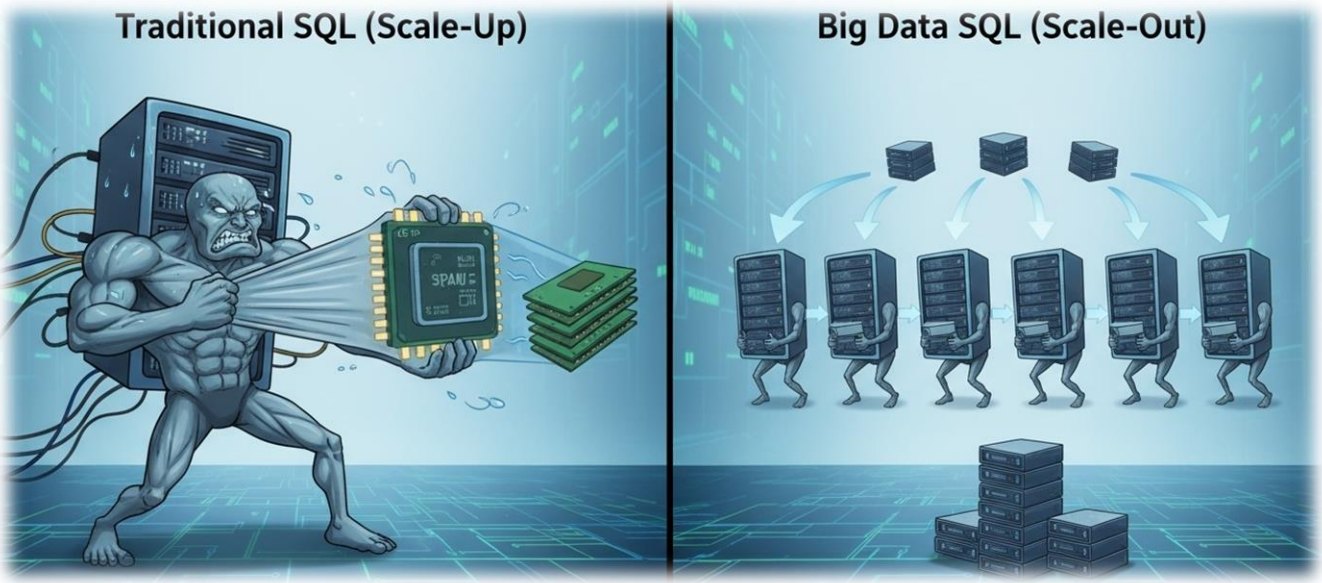
전통 SQL은 단일 서버 기반 RDBMS에 최적화

- 정형 데이터(표 형태) 처리에 매우 효율적
- 기업의 고객/회원/주문 데이터와 같은 구조화된 데이터를 빠르게 저장·조회·관리하는 데 최적

데이터 폭증과 다양성 증가로 단일 서버 방식 한계 도달

- 접속로그, 동영상, 웹 클릭 데이터, 이미지, 센서 등 다양한 데이터가 등장
- Scale-Up만으로는 성능·용량·비용 문제를 해결할 수 없어 효율이 떨어짐

빅데이터 SQL 등장



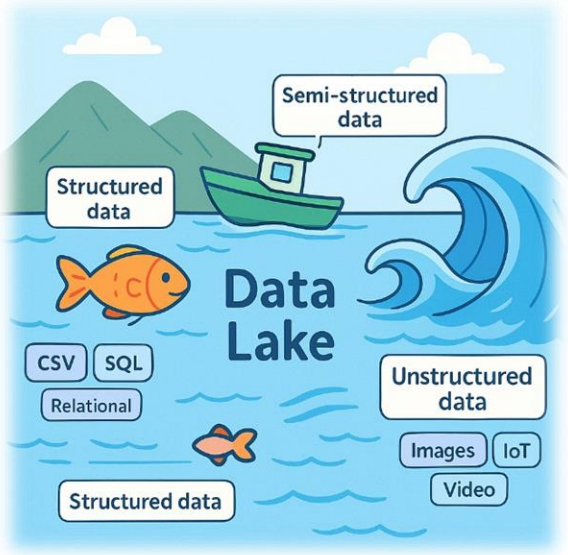
빅데이터 SQL

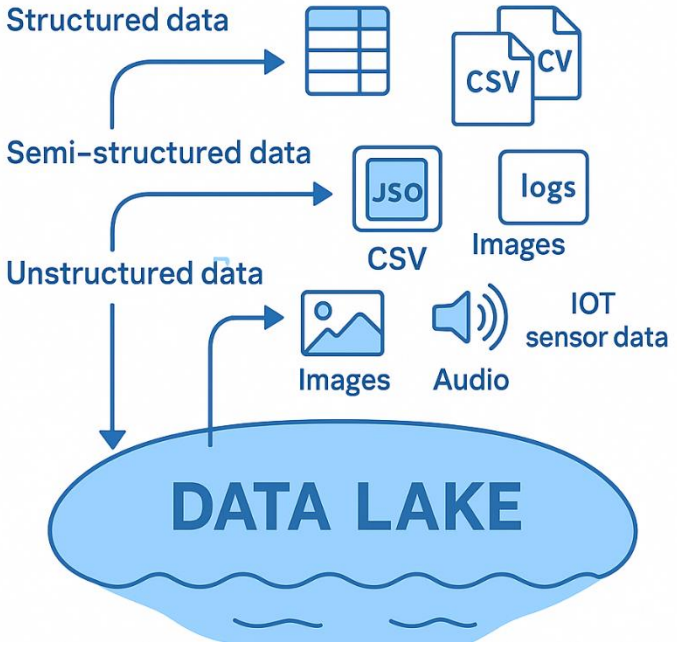
- 데이터를 여러 서버에 분산 저장
- 작업을 병렬로 나누어 동시에 처리하는 Scale-Out 방식을 사용
  - 1대의 강력한 서버보다, 일반 성능의 서버 10대가 동시에 처리하도록 설계된 구조
- 로그, JSON, IoT 센서 데이터 같은 반정형/비정형 데이터까지 SQL로 처리할 수 있도록 기능 확장

Lakehouse 시대의 SQL

Data Lake

- 데이터 레이크(Data Lake)는 조직이 수집하는 다양한 형태의 데이터를 가공하지 않은 원천(raw) 상태 그대로 저장할 수 있는 대규모 저장소
- 정형 데이터뿐 아니라, 반정형·비정형 데이터를 모두 담아둘 수 있으며, 필요할 때 꺼내서 처리·분석하는 방식으로 활용



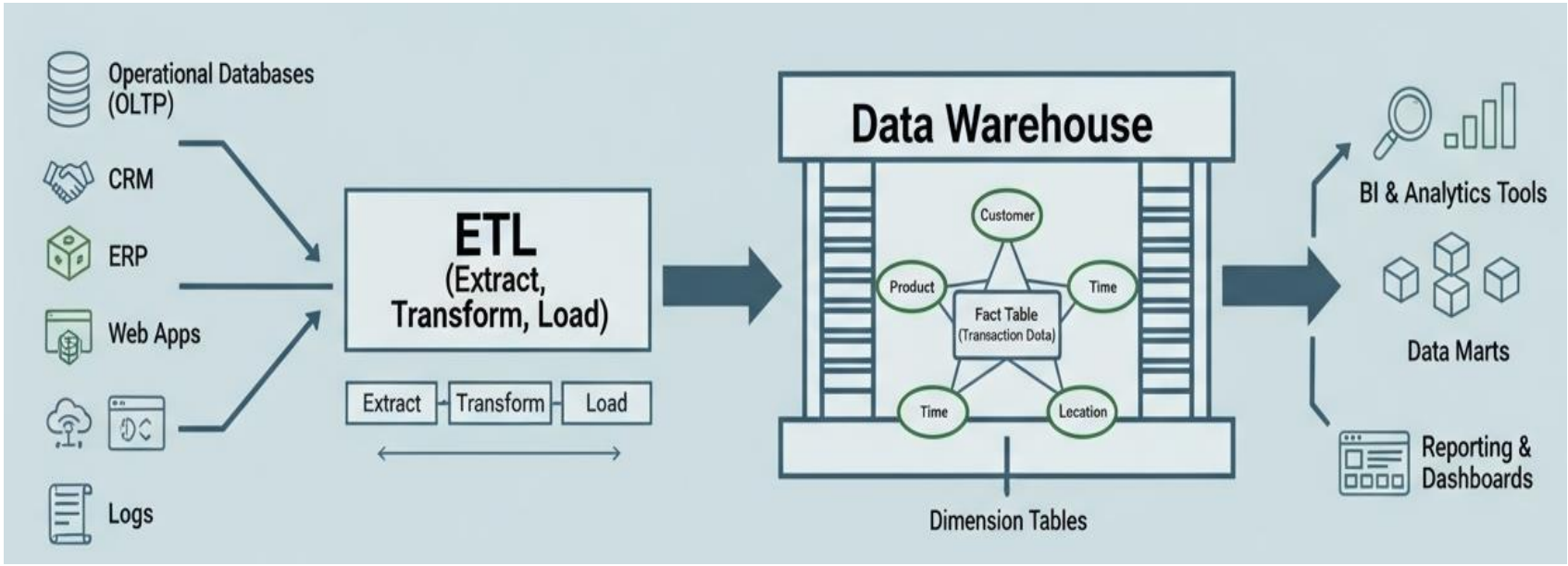


Data Warehouse

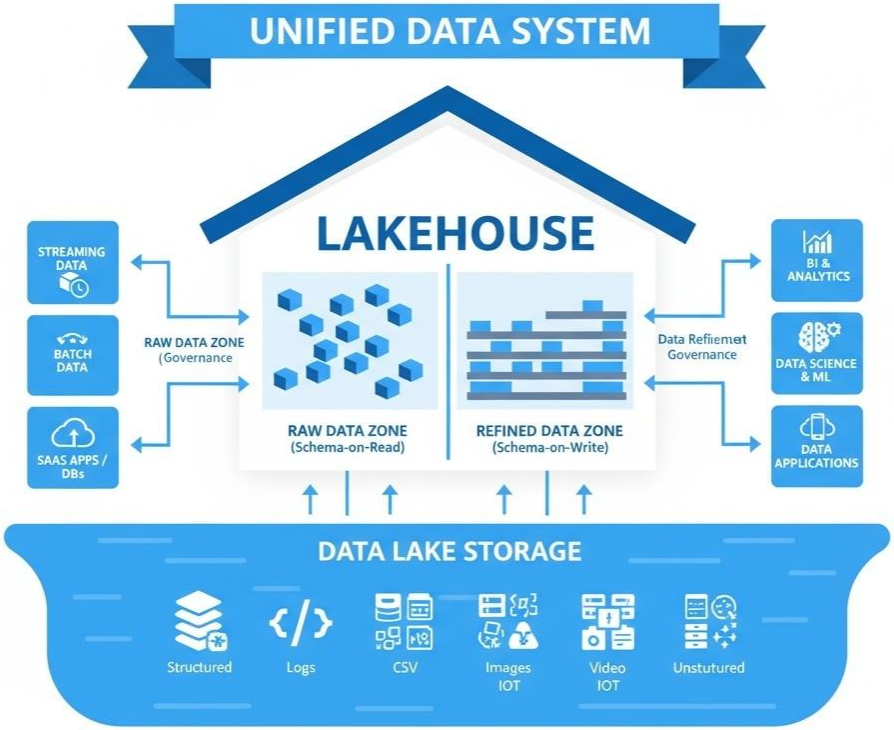
Data Warehouse

- 기업의 의사결정과 업무 지원을 위해, 정제/구조화된 데이터를 저장하는 중앙집중형 데이터 저장소
- 업무 시스템에서 수집된 데이터를 ETL (추출·변환·적재, Extract, Transform, Load) 과정 수행

➔ 데이터 품질을 보장한 뒤 저장하며, 경영 분석, BI (Business Intelligence) 리포트, 통계 분석 등에 활용

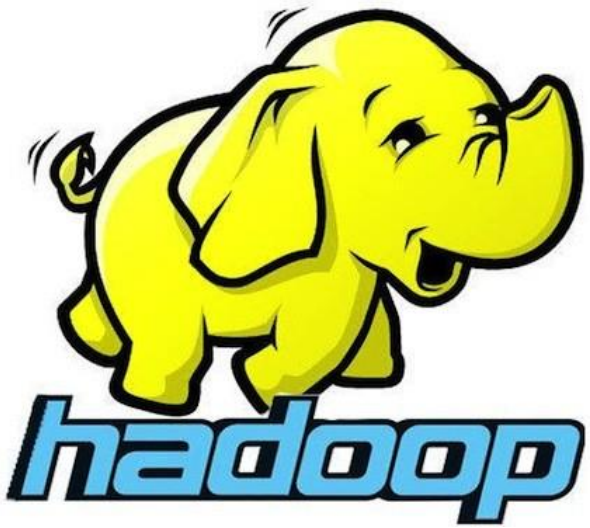


Lakehouse



- Data Lake의 유연성과 Data Warehouse의 구조적 분석 능력을 결합한 통합 데이터 아키텍처
- 원본 데이터를 저장하면서도, 동시에 정제·가공된 데이터 분석 환경을 제공

  ➔ AI/ML 분석과 BI 분석을 모두 처리할 수 있는 통합형 플랫폼


hadoop

Hadoop 소개

Hadoop

- 대규모 데이터를 분산 환경에서 저장하고 처리하기 위해 개발된 오픈소스 프레임워크
- 저렴한 범용 서버 여러 대를 묶어 하나의 대형 시스템처럼 동작
- **대용량(Volume) · 고속 생성(Velocity) · 다양성(Variety)** 빅데이터 처리의 태동기에 핵심 기술

Apache?

- **Apache Software Foundation**에서 지원하는 프로젝트
- 오픈소스: 무료, 전 세계 개발자 커뮤니티가 참여
- 높은 신뢰성과 넓은 생태계 (업계 표준으로 자리잡은 기술이 많음)
- 아파치 HTTP(웹)서버, 아파치 Hadoop 등

Hadoop이 등장한 배경

- **데이터 폭증과 기존 RDBMS의 한계**
  - 기업과 서비스에서 생성되는 데이터가 TB → PB 단위로 증가
  - 단일 서버 성능을 올리는 Scale-Up 방식은 비용과 구조적 한계
- **정형 중심 처리 구조의 한계**
  - RDBMS는 스키마가 고정된 정형 데이터 처리에 최적화
  - 웹 로그, 클릭, SNS, IoT 데이터 등 비정형 데이터 처리 한계
- **분산 파일 시스템과 병렬 처리에 대한 필요성**
  - 여러 서버에 저장하고 동시에 처리할 수 있는 기술 요구
- **구글의 분산 기술 논문 공개**
  - Google의 GFS(Google File System), MapReduce 논문이 발표
  - 오픈소스 형태의 Hadoop 프로젝트 개발 ➔ 누구나 분산 데이터 처리를 구현 가능한 시대가 열림
    - **Apache License 2.0**으로 배포되는 **완전한 오픈소스 (누구나 다운로드, 설치, 수정, 배포 가능)**

- 최근 기업들은 Spark + Cloud + Lakehouse 방식으로 이동 중
- Hadoop 생태계 일부는 계속 쓰이고 있고, 특히 HDFS, YARN, Hive, HBase는 여전히 활용됨

논문 다운로드

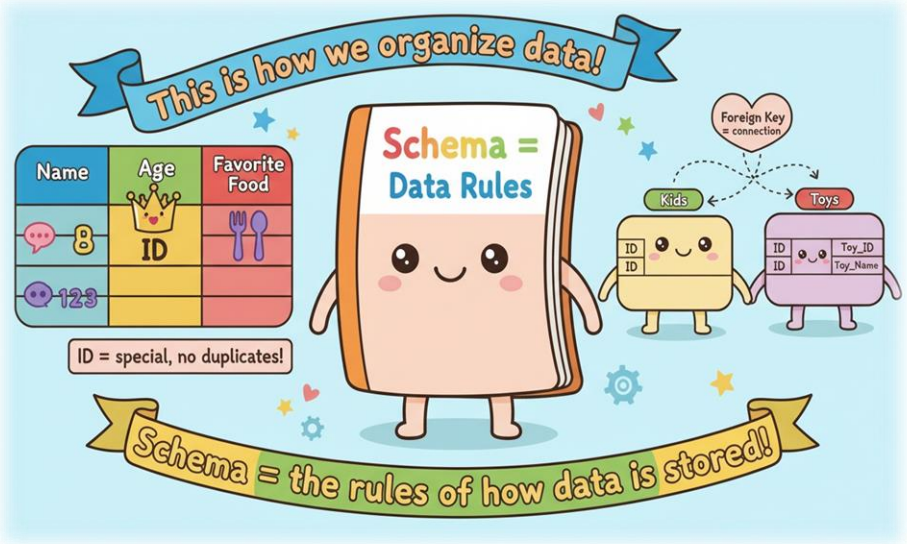
- The Google File System (GFS)
- MapReduce

[참고] 스키마??

스키마 (Schema)

- 어원
  - 그리스어 'σχήμα (skhēma)'에서 유래
  - 형태, 모습, 구조, 틀, 외형을 의미

→ 어떤 것이 어떤 형태로 존재하는지를 표현



데이터베이스 스키마 (데이터의 설계도)

- 데이터를 조직하기 위한 구조적 틀이라는 원래 의미가 그대로 이어진 개념

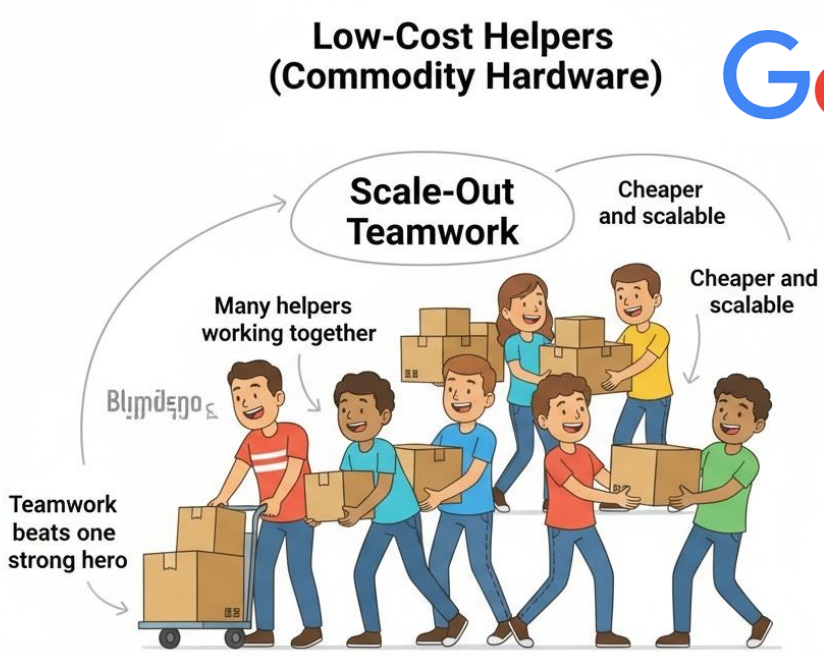
어떤 테이블이 있는지?　　어떤 컬럼이 있는지?

데이터 타입은 무엇인지?　　제약조건(규칙)은 무엇인지?

Hadoop의 특징 (1/4)

저비용 분산 처리

- 범용 하드웨어(Commodity Hardware)를 여러 대 연결하여 대규모 데이터를 저장·처리
- 고가 장비 없이도 수평 확장이 가능해 비용 효율적



Hadoop의 특징 (2/4)

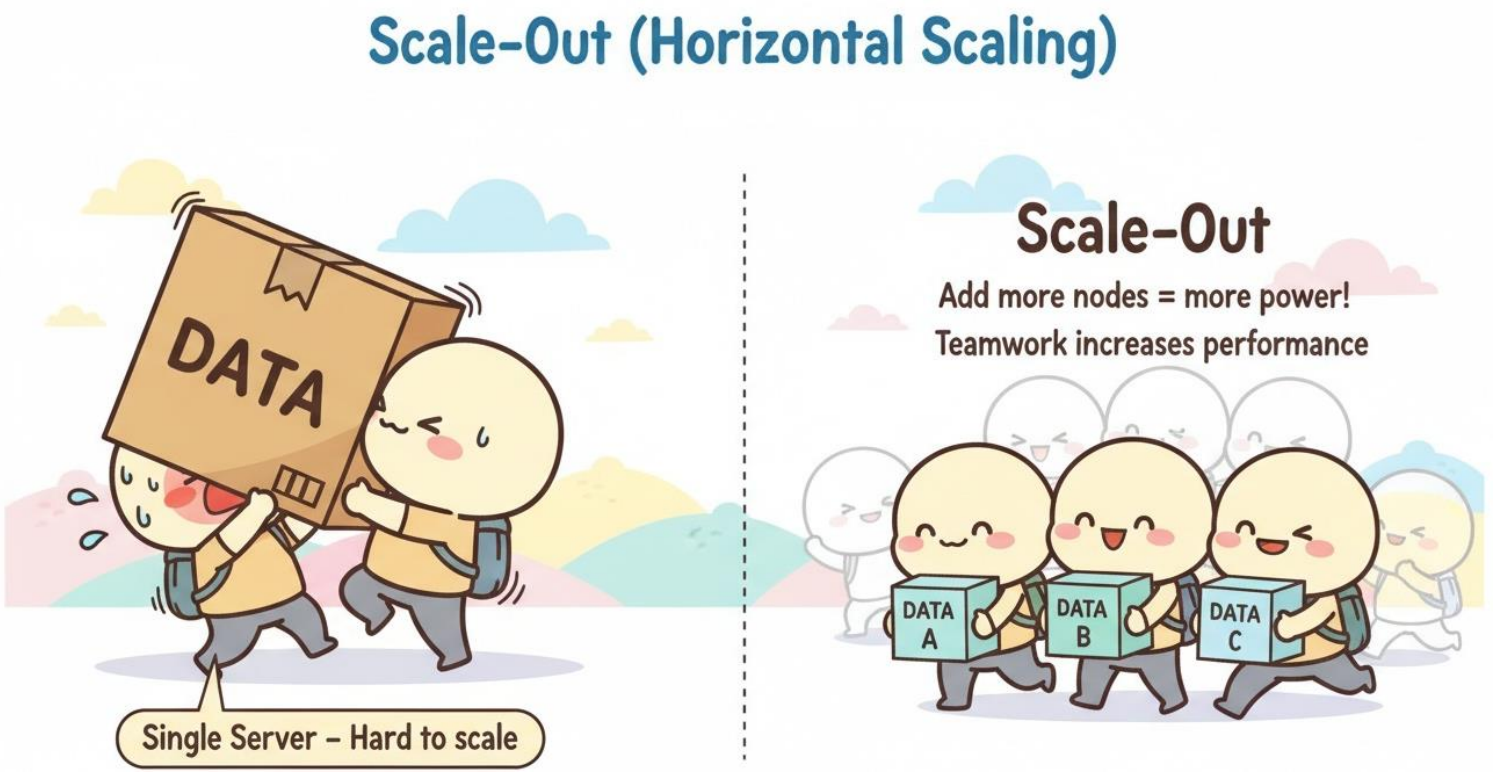
고장 허용(Fault Tolerance)

- 일부 노드 장애가 발생해도 자동 복제 및 재처리 메커니즘을 통해 작업이 지속
- 데이터 복제(Replication)로 안정성을 확보

Hadoop의 특징 (3/4)

■ 수평 확장성(Scale-Out)

- 노드(저렴한 서버 등) 추가하는 것만으로 성능과 저장 용량을 증가
- 데이터 증가에 유연하게 대응



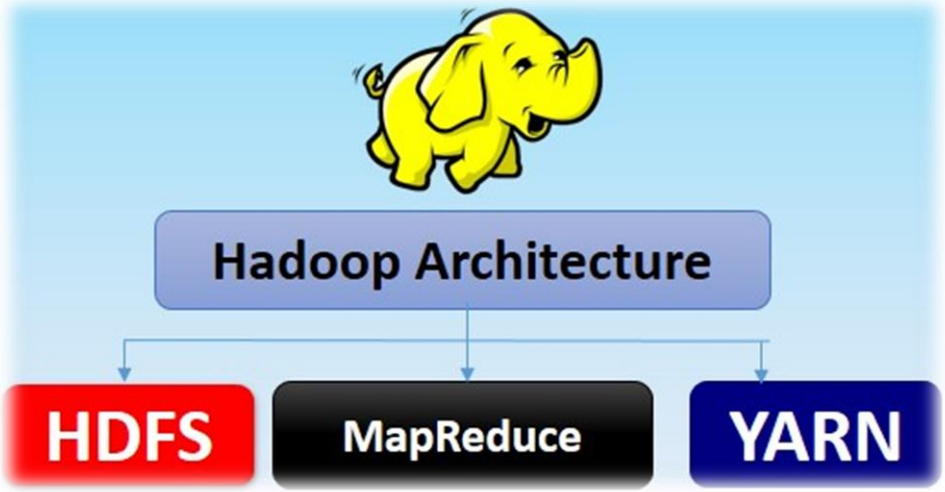
Hadoop의 특징 (4/4)

■ 대용량 데이터 처리에 최적화

- 대규모 데이터셋을 여러 노드에 분산 저장하고, 병렬로 작업을 수행하여 처리 속도 향상

Hadoop의 핵심 구성 요소

- **HDFS (Hadoop Distributed File System)**
  - 대규모 데이터를 여러 노드에 분산 저장하는 파일 시스템
  - 한 번 기록된 대규모 데이터를 빠르게 읽는 데 최적화

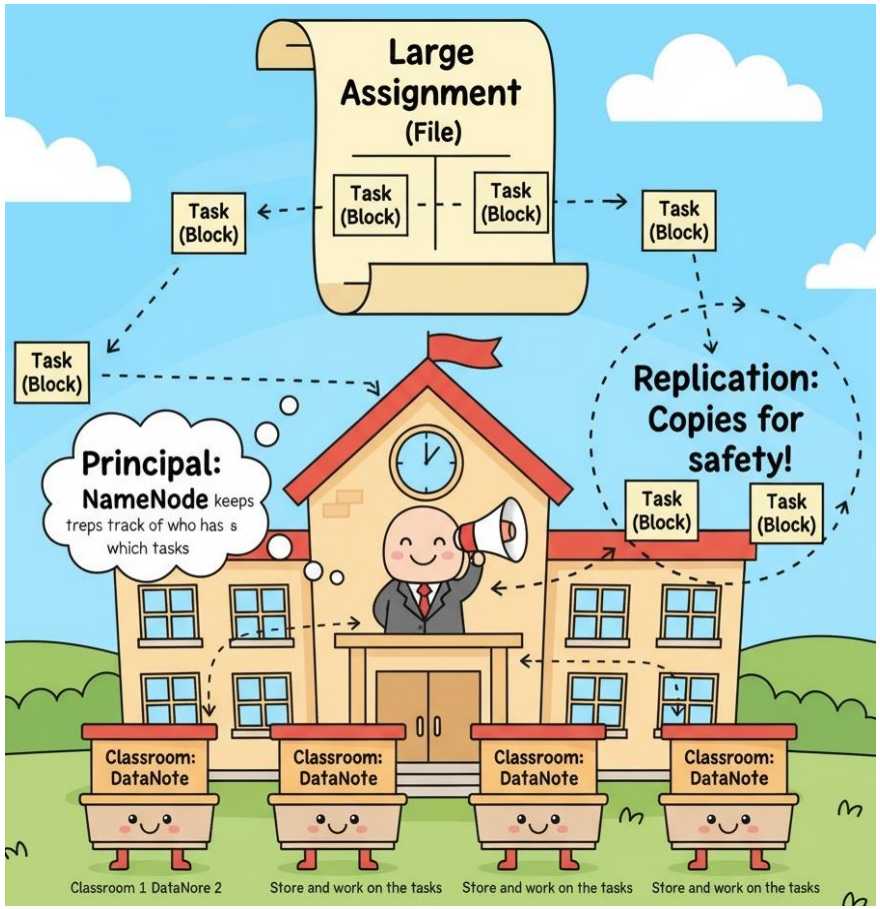


- **MapReduce**
  - 데이터 분산 처리를 위한 프로그래밍 모델
  - 작업을 쪼개어 여러 노드에서 병렬 처리하고 결과를 합산
- **YARN (Yet Another Resource Negotiator)**
  - 클러스터 자원(CPU, 메모리 등)을 관리하고, 다양한 응용 프로그램 실행을 조정 (자원 관리자 역할)
- **Hadoop Ecosystem**
  - Hive, Pig, HBase, Sqoop, Flume, Oozie, Zookeeper 등 다양한 도구로 확장된 Hadoop 기반 생태계

HDFS (Hadoop Distributed File System) - 1/2

■ HDFS (Hadoop Distributed File System) → 장애 발생해도 안정적 운영

- 대용량 데이터를 여러 노드에 분산 저장하는 Hadoop의 핵심 분산 파일 시스템



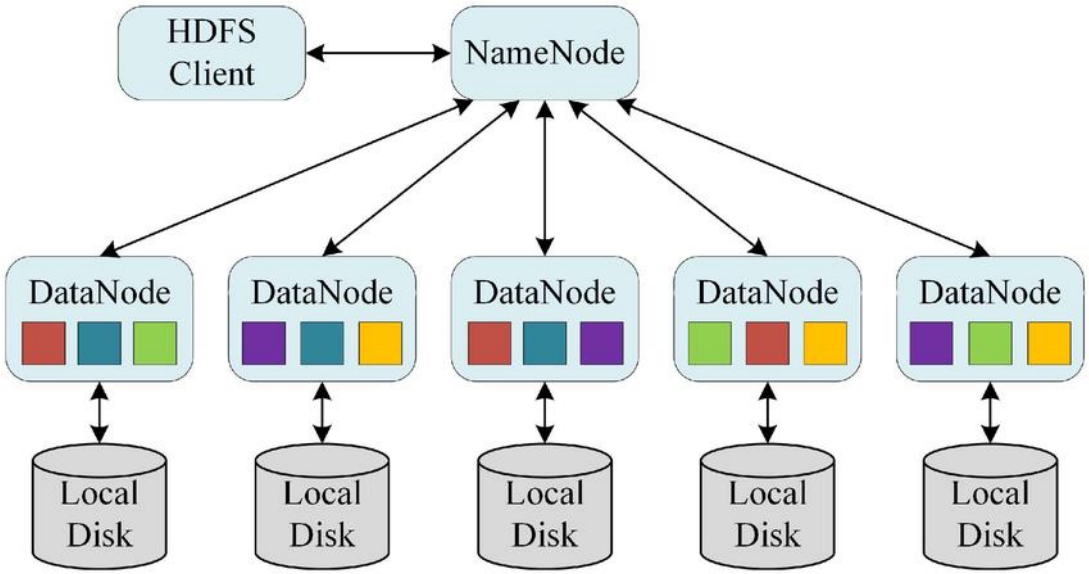
교장 = NameNode
교장은 직접 과제를 보관하거나 해결하지 않는다.
어떤 과제(Block)가 어떤 반(DataNode)에 배정되었는지 기록/관리

각 반 = DataNode
실제로 과제를 수행하고 보관하는 곳은 반(DataNode)이다.
각 반은 자신에게 배정된 과제(Block)를 가지고 있고, 정상적으로 보관 중인지 교장에게 주기적으로 보고(Heartbeat) 한다.

과제를 쪼개서 나누어 주기 = 블록 단위 저장
한 명에게 주기엔 너무 큰 과제라면, 과제를 여러 조각(Block)으로 나누어 여러 반에 나누어 줄 수 있다.
HDFS도 큰 파일을 블록 단위로 쪼개 여러 DataNode에 저장한다.

여러 반에 같은 과제 복사본 주기 = 복제(Replication)
혹시 어떤 반에서 사고가 나서 과제를 잃어버릴 수 있기 때문에 같은 과제 블록을 여러 반에 복사(기본 3개) 해서 나누어 준다.
한 반이 망가져도 다른 반에서 과제를 다시 가져올 수 있다.

HDFS (Hadoop Distributed File System) - 2/2



데이터 저장 방식: 블록(Block)

- 대용량 파일을 일정 크기의 블록(Block) 단위로 나누어 저장
- 기본 블록 크기: 보통 128MB 또는 256MB
- 일반 파일 시스템보다 훨씬 크다

데이터 신뢰성 확보: 복제(Replication)

- 파일 블록(Block)은 기본적으로 여러 노드에 복제(기본값: 3개)되어 저장
- 한 노드가 장애가 나더라도 다른 노드에 복제본이 존재하
- 데이터 유실 없이 안정적으로 서비스

데이터 관리 시스템

마스터-슬레이브 구조로 동작

크게 두 종류의 노드로 구성

NameNode

- 파일 시스템의 메타데이터를 관리하는 핵심 노드
- 파일이 어떤 블록으로 나뉘었으며, 각 블록이 어떤 DataNode에 저장되었는지에 대한 정보를 저장/관리
- 파일 이름, 블록 위치, 접근 권한 등

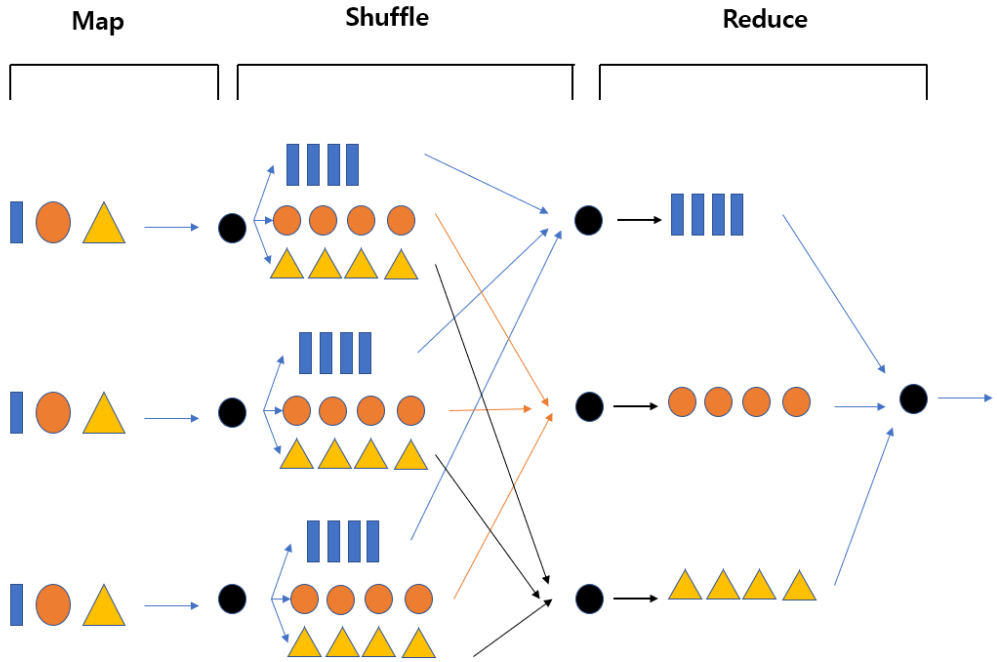
DataNode

- 실제 데이터 블록을 저장/관리
- NameNode의 지시에 따라 블록을 저장·전송
- 주기적으로 상태 정보를 NameNode에 보고
- "정상 동작 중", "블록 저장 완료" 등

MapReduce (1/7)

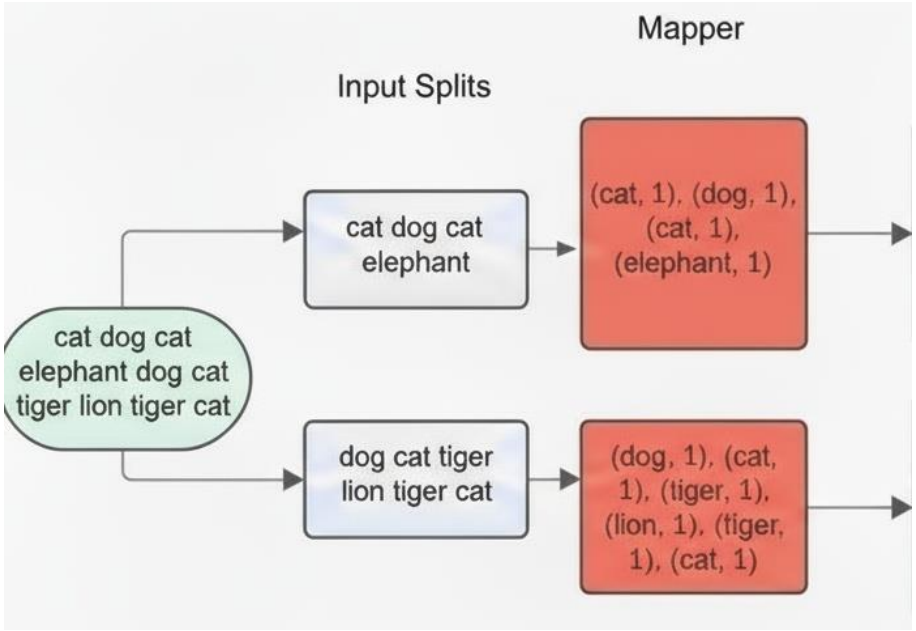
MapReduce

- 하둡이라는 플랫폼 위에서 돌아가는 데이터 처리 엔진 (프로그래밍 방식)
- 대용량 데이터를 분산 환경에서 병렬 처리하기 위한 프로그래밍 모델
- 복잡한 데이터 처리 작업을 두 단계(Map + Reduce 단계)로 나누어 여러 노드에서 동시에 실행



MapReduce (2/7)

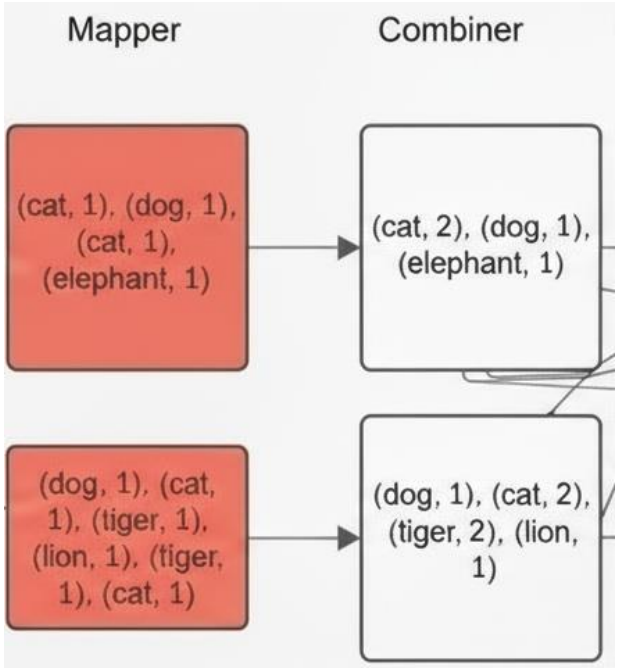
- **Map** 단계 — 데이터를 "쪼개서 가공"하는 단계
  - 데이터를 여러 조각으로 나누어 여러 작업자(Map Task)가 병렬 처리
  - 각 Map Task는 데이터를 처리하여 (Key, Value) 형태의 중간 결과를 생성



MapReduce (3/7)

Combiner 단계 (Optional)

- MapReduce에서 Combiner는 선택적으로 사용되는 단계
- Map에서 생성된 중간 결과를 로컬에서 한 번 더 합쳐주는 역할
- Map 작업이 끝나면 보통 (Key, Value) 형태의 결과가 많이 생성
  - 이 값을 그대로 Shuffle 단계로 보내면 네트워크 전송량이 매우 커질 위험성 존재
  - Map 작업 내에서 동일한 Key를 가진 값들을 미리 합산하여 데이터 양 감소 시킴
  - (예시) Map 작업에서 "cat"이라는 단어가 5번 등장했다면,
    - 원래는 (cat, 1)을 5번 보내야 하지만,
    - Combiner가 있다면 이를 미리 (cat, 5)로 묶어서 Shuffle 계층으로 전송



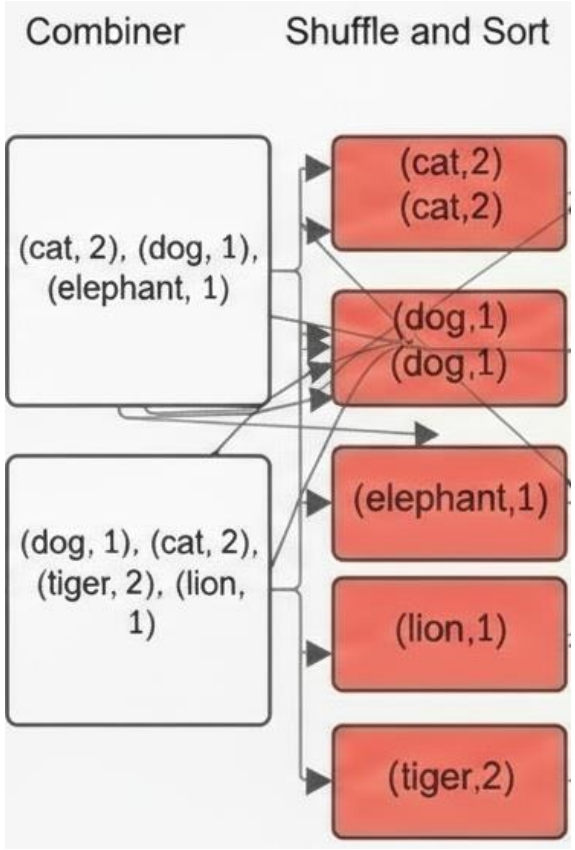
MapReduce (4/7)

Shuffle 단계 — "같은 Key끼리 모으는 과정"

- Map 단계에서 만들어진 중간 결과는 Key 값이 섞여 있으므로 같은 Key를 가진 값들끼리 모아서 그룹화하는 과정
  - 서로 다른 Map에서 생성된 결과 중 같은 Key끼리 재분배
  - Key 기준으로 정렬 및 그룹화가 이루어진다.
  - 이후 Reduce Task로 전달할 준비가 완료된다.

여러 Map 결과에서 같은 Key들을

한 곳으로 모아주는 정리 단계



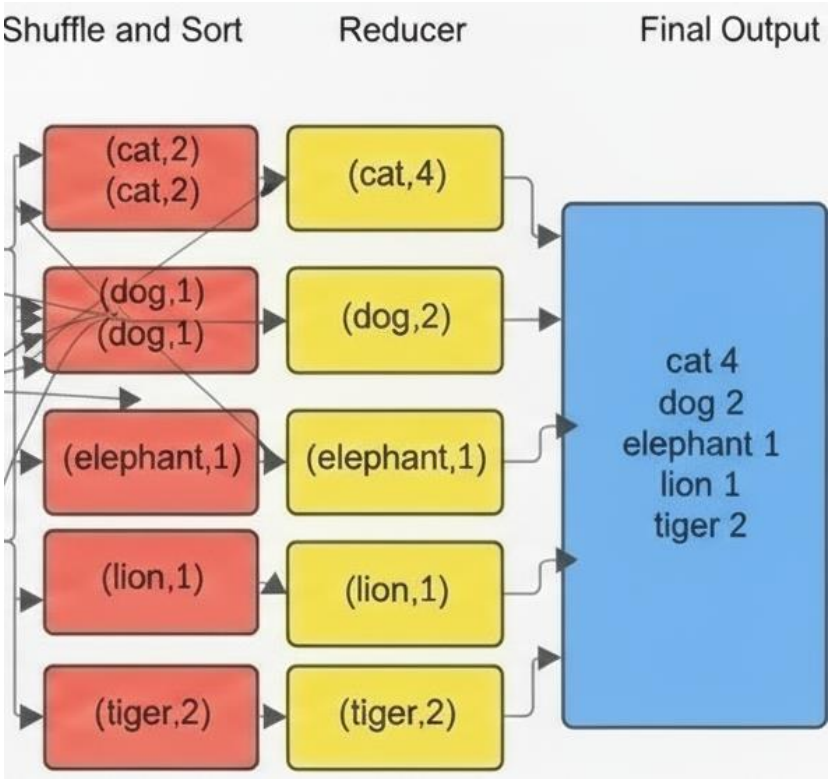
MapReduce (4/7)

Reduce 단계 — Key별로 최종 결과 생성

- 집계, 계산, 요약 수행하여 최종 결과를 생성
  - 같은 Key에 대한 Value들을 하나로 합치는 작업 수행
  - Key별 최종 결과 출력

같은 Key 데이터를 합쳐

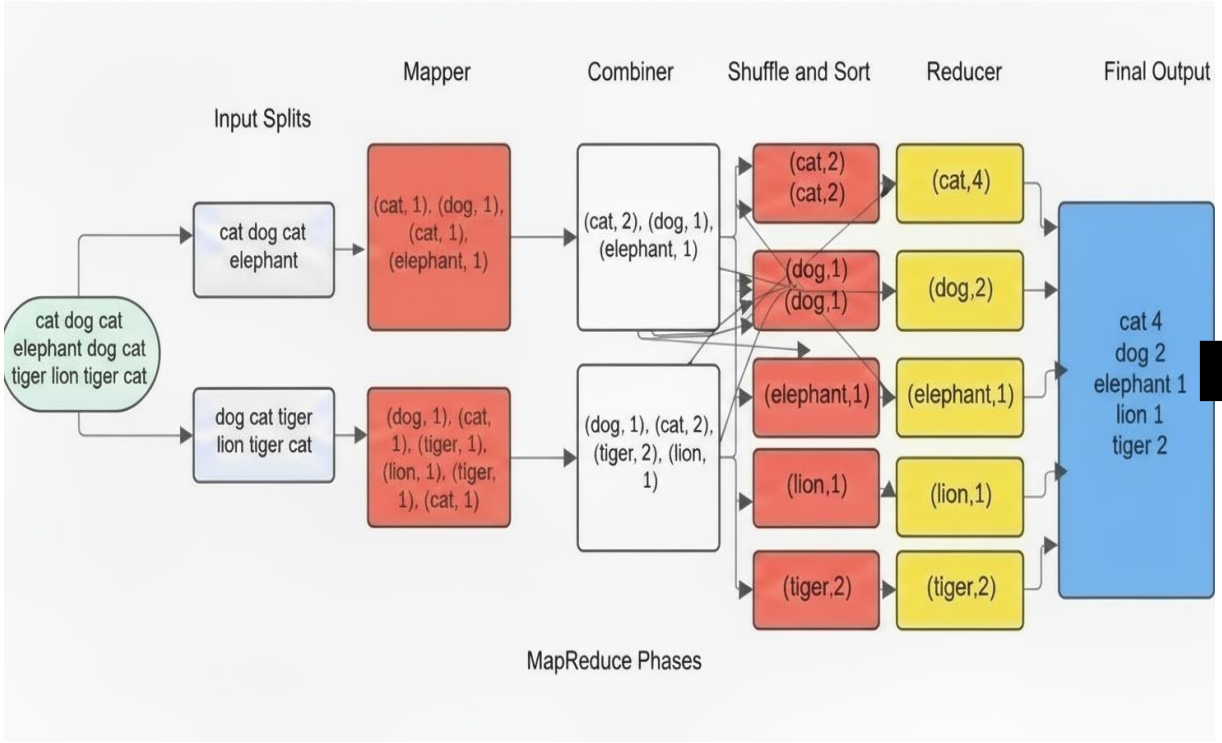
최종 결과를 도출하는 단계



MapReduce (5/7)

Final Output은 어떻게 Data Block으로?

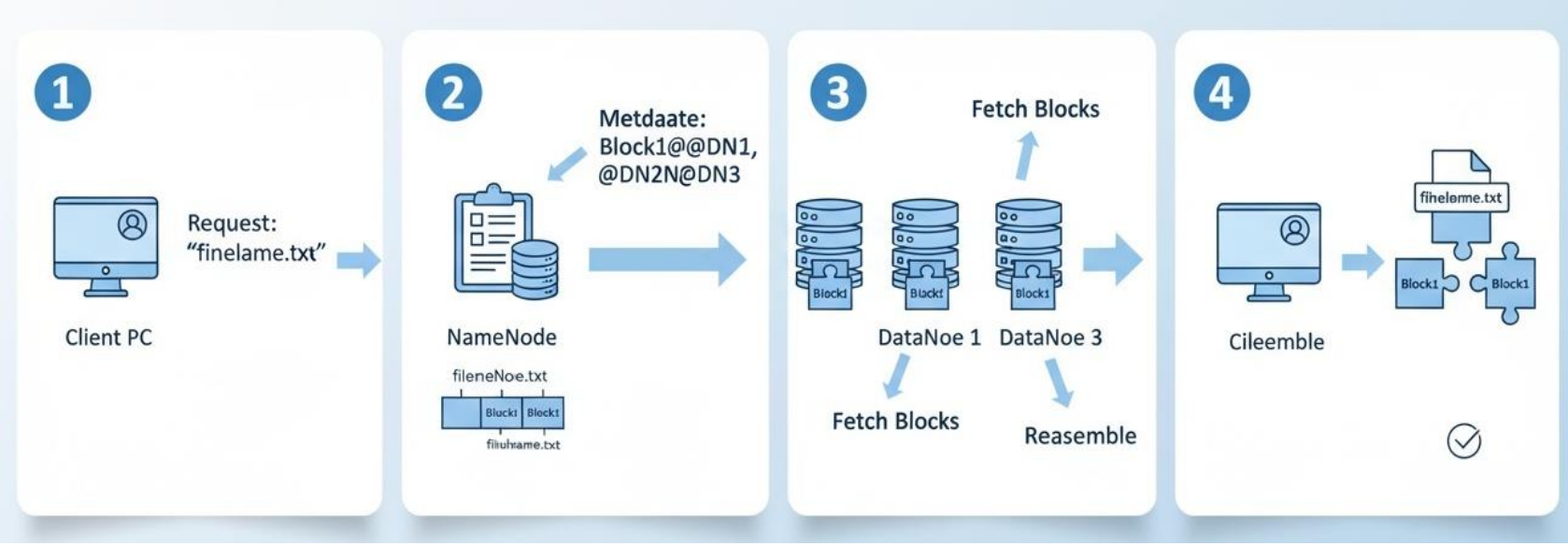
- Hadoop(HDFS)에서는 모든 파일은 자동으로 일정 크기의 블록 단위로 저장
- 예를 들어, Reduce 단계에서 만들어진 최종 결과 파일 크기가 350MB라고 하면



블록 번호	저장 크기
Block 1	128MB
Block 2	128MB
Block 3	94MB

MapReduce (6/7)

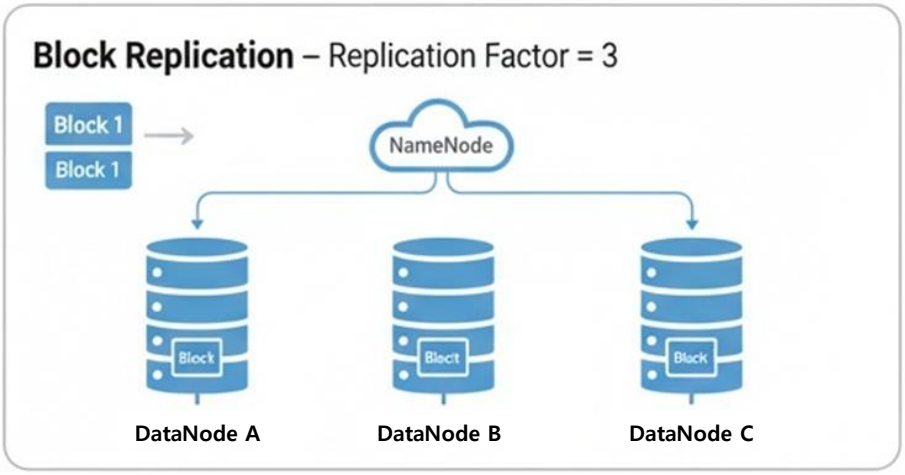
나뉜 블록들은 어떻게 원래 데이터로 복원될까?

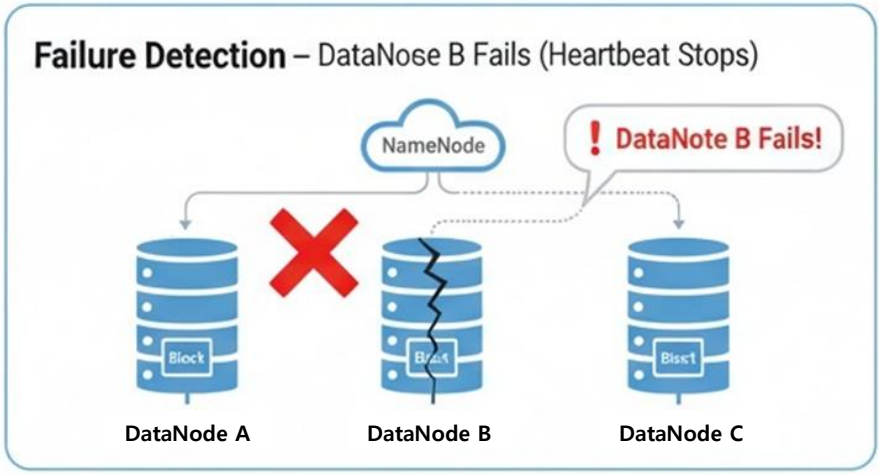


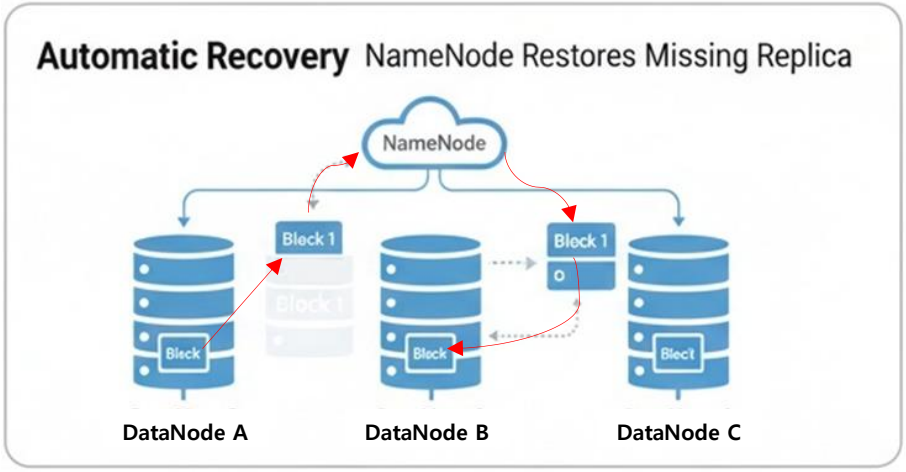
1. 사용자가 파일 읽기 요청을 보낸다.
2. NameNode가 블록 메타데이터를 전달한다.
3. 클라이언트가 각 DataNode에 직접 연결하여 블록을 가져온다.
4. 클라이언트가 블록을 자동으로 병합하여 원본 파일로 복원한다.

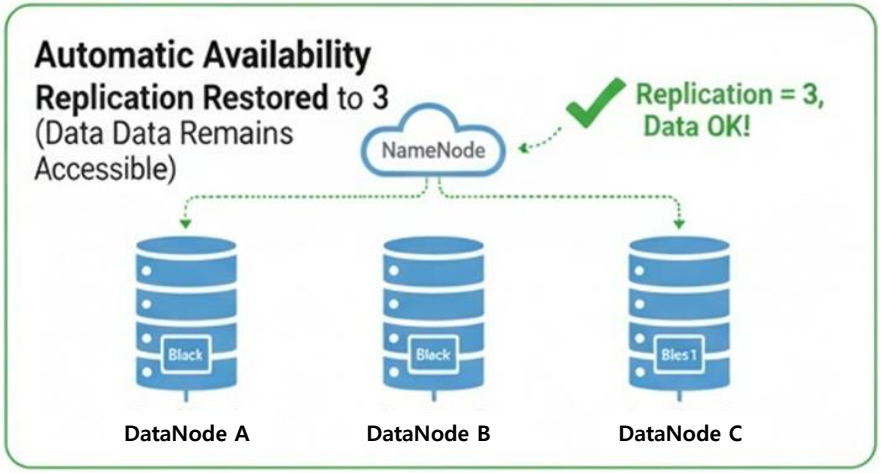
MapReduce (7/7)

Data Recovery









Hadoop Ecosystem

Hadoop Ecosystem

- 대규모 데이터의 수집 → 저장 → 처리 → 분석 → 관리 전 과정을 지원하기 위해

  함께 동작하는 다양한 오픈소스 도구들의 통합 생태계

컴포넌트	역할	쉬운 설명 (비유)
HFDS	데이터 파일 시스템	데이터 저장 창고
YARN	리소스/작업 관리	작업 스케줄링 & 자원 관리하는 교통/도로망
Hive	데이터 웨어하우스	데이터를 정리해 검색·분석하는 도서관/검색 시스템
HBase	분산 Key-Value Store	빠른 조회가 가능한 NoSQL 데이터 창고
Zookeeper	Coordination Service	동물원 사육사는 사자, 호랑이, 원숭이 등 다양한 동물이 멋대로 움직이지 않도록 조율
Kafka (선택)	스트리밍 처리	실시간 데이터 전달을 위한 우편 배송 서비스

Spark

Spark 등장

Spark 등장 배경

- Hadoop 시대에는 대용량 데이터를 처리하기 위해 주로 MapReduce 방식이 사용
- MapReduce는 분산 환경에서 안정적으로 데이터를 처리할 수 있다는 장점
- 그러나, 실제 데이터 분석 및 머신러닝 분야에서는 여러 한계 발생
  - MapReduce는 각 단계마다 디스크(HDFS)에 결과를 저장하는 구조
  - 연산 과정에서 매번 디스크 읽기/쓰기(I/O)가 발생하여 처리
  - 머신러닝 학습, 그래프 분석, 실시간 데이터 처리와 같은 작업은

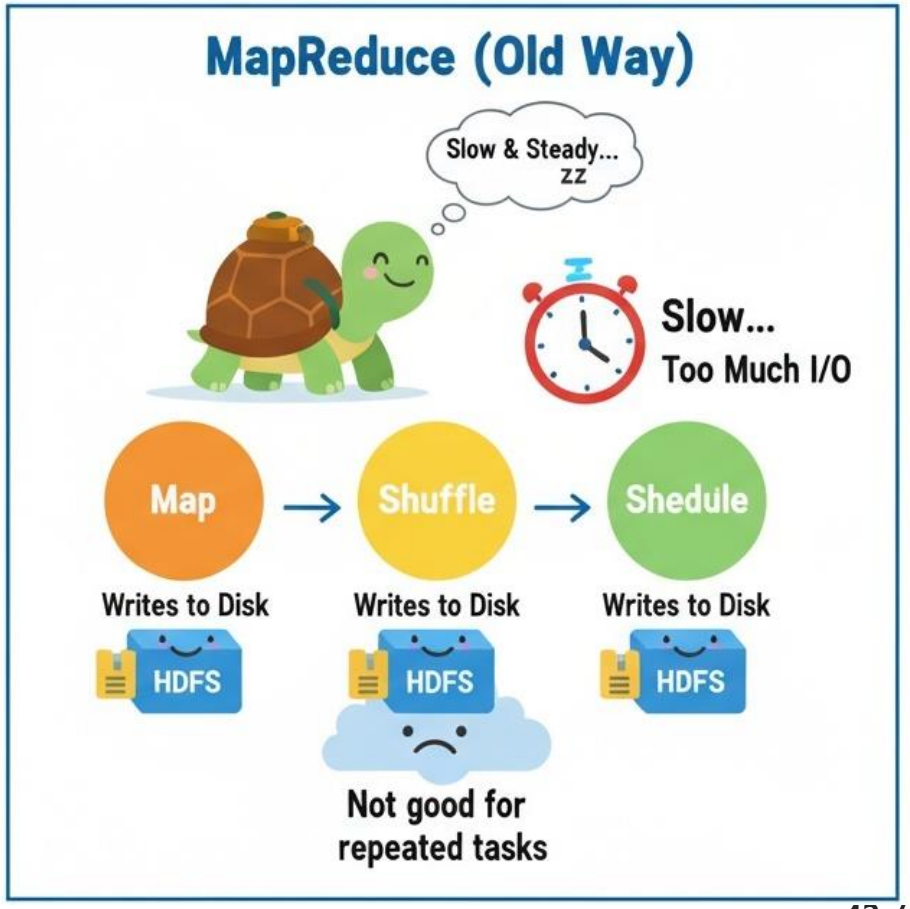
    동일한 데이터를 여러 번 반복적으로 연산

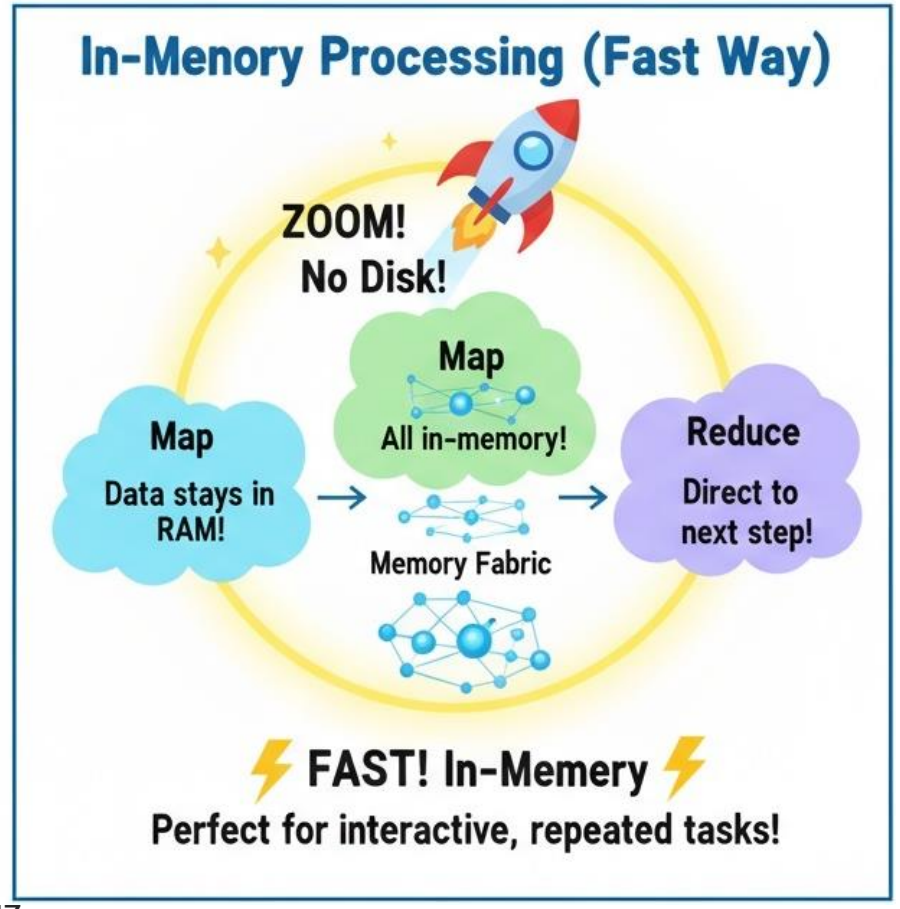
그림으로 살펴보기 (다음 슬라이드)

그림으로 살펴보기 1/3. 처리 속도의 한계

속도 차이 비율 (대략)

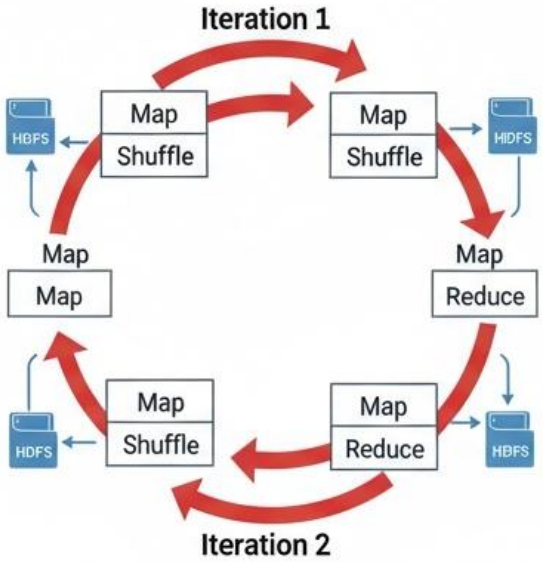
- RAM vs SSD: RAM이 약 100 ~ 1,000배 빠름
- RAM vs HDD: RAM이 약 10,000~100,000배 빠름





그림으로 살펴보기 2/3. 반복·탐색형 연산에 취약

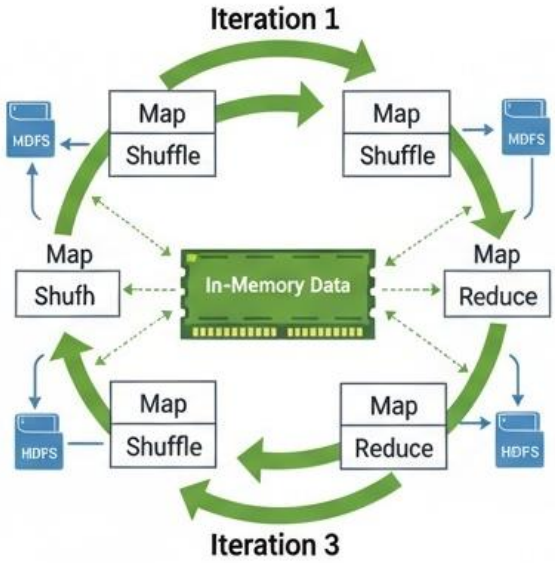
MapReduce (Iteritive Workolad - Slow



Writes to disk every iteration
→ High I/O cost

Not suitable for iteritive algorithms (reads/wites every iteration)

In-Memry Processing (e.gl., Spark) - Fast for Iteration)



Keeps data in RAM →
Low I/O cost

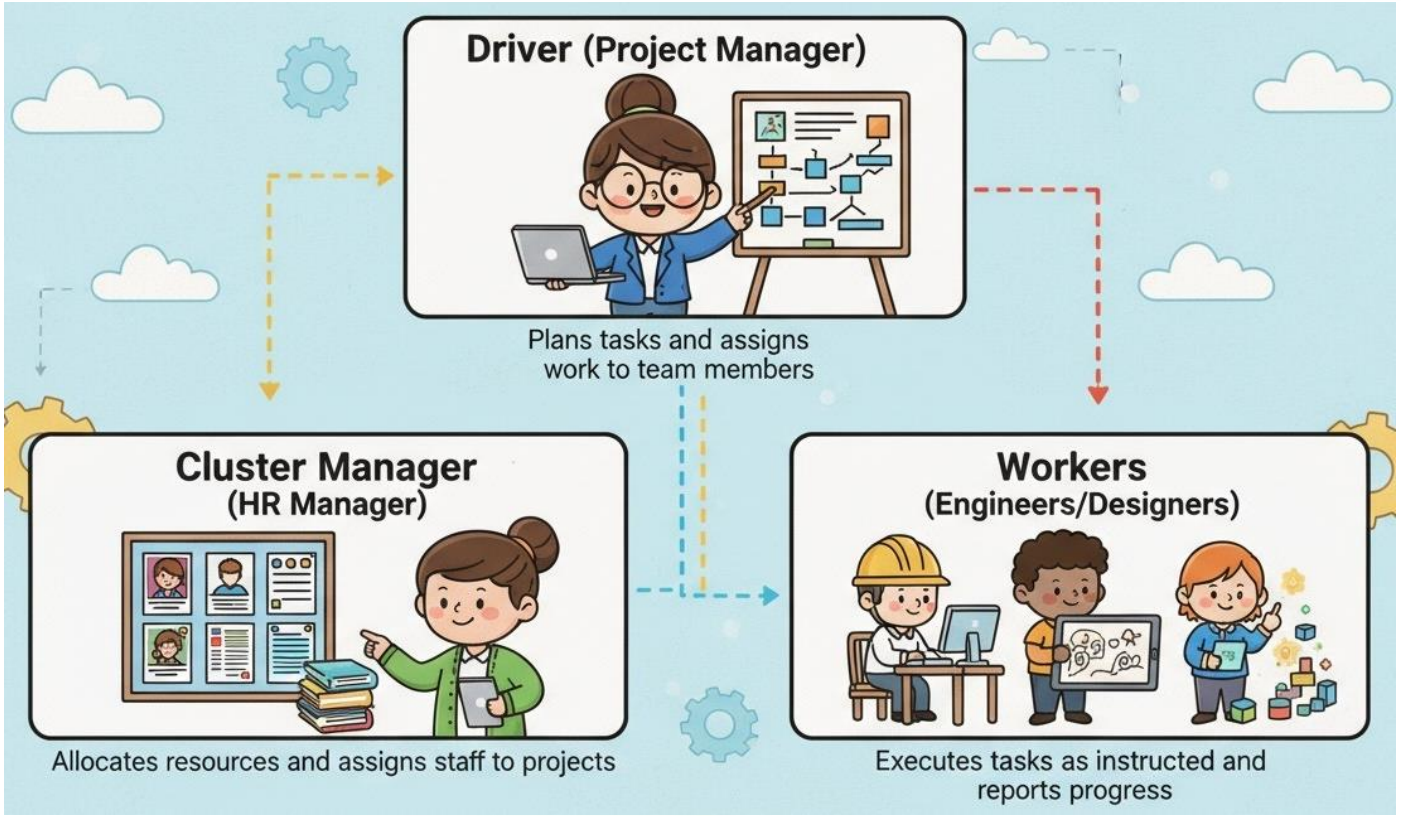
Ideal for iteritive algoritims (avoids disk writes)

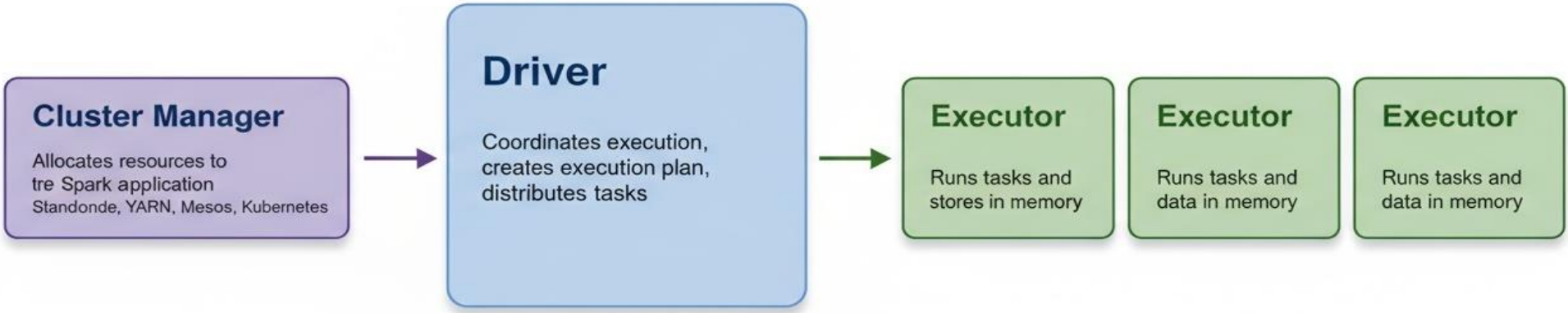
그림으로 살펴보기 3/3. 디스크 vs In-Memory I/O 속도 비교

- **메모리(RAM): 빛의 속도**
- **SSD: 빠른 자동차 속도**
- **HDD: 자전거 속도**

이유	메모리(RAM)	디스크(HDD/SSD)
구조	반도체 기반, 직접 랜덤 접근 가능	• 저장 셀/플래터 기반 • SSD는 더 빠르나 여전히 느림
접근 방식	전기 신호로 즉시 접근	• 블록 단위 접근 • SSD는 페이지/블록 • HDD는 물리 이동 필요
CPU와 거리	매우 가까움 (버스 연결)	• 상대적으로 멀고 • I/O 드라이버 계층 거침

Spark 구조


Driver (Project Manager)
Plans tasks and assigns work to team members
Cluster Manager
(HR Manager)
Allocates resources and assigns staff to projects
Workers
(Engineers/Designers)
Executes tasks as instructed and reports progress


Cluster Manager
Allocates resources to tre Spark application
Standonde, YARN, Mesos, Kubernetes
Driver
Coordinates execution, creates execution plan, distributes tasks
Executor
Runs tasks and stores in memory
Executor
Runs tasks and data in memory
Executor
Runs tasks and data in memory

Driver, Executor, Cluster Manager

구성 요소	기술적 설명	쉬운 비유 설명
Driver	- Spark 애플리케이션의 메인 프로세스 - 실행 계획 수립 및 최적화 - Task를 Executor에 분배하고 결과를 수집	**프로젝트 매니저(PM)** 일정을 계획하고, 팀원들에게 업무를 배정하며 결과를 취합해 보고하는 역할
Executor	- 클러스터 각 노드에서 Task 실행 - 실제 연산, 변환, 집계 수행 - 필요 시 데이터 메모리 캐싱	**팀원(실무자 개발자)** PM이 준 일을 실제로 수행하고 결과물을 만들어내는 사람
Cluster Manager	- 클러스터 자원(CPU, 메모리 등)을 관리 - Executor 실행 위치와 자원 할당 결정 - Standalone, YARN, Mesos, Kubernetes 등 지원	**인사/자원 배정 담당자(HR 매니저)** 누가 어떤 프로젝트에 배정될지, 몇 명이 필요한지, 어떤 장비를 줄지 결정하는 역할

실습: Spark with Jupyter (Docker)

실습 안내 및 코드

https://www.deepshark.org/courses/data_science/w/11_big_data_processing#spark_with_jupyter

실습 환경: Docker

- 설치: https://docs.docker.com/get-started/get-docker/

Docker Container 설치 및 실행

```
docker run -it -p 8888:8888 -p 4040:4040 --name spark jupyter/pyspark-notebook

# PySpark + Jupyter Notebook 환경을 도커로 실행하는 명령어

# docker run -it ₩      # -it : 터미널 상에서 컨테이너와 상호작용 가능하도록 실행
#     -p 8888:8888 ₩    # 로컬PC(왼쪽) 8888 포트를 컨테이너(오른쪽) 8888 포트와 연결
#                       #   → 브라우저에서 Jupyter Notebook 접속 가능
#     -p 4040:4040 ₩    # Spark UI용 포트 연결
#                       #   → http://localhost:4040 에서 Spark 실행 계획/작업 모니터링 가능
#    --name spark ₩     # 컨테이너 이름을 'spark'로 지정
#                       #   → 컨테이너 관리(시작/중지/접속) 시 이름으로 제어 가능
#    jupyter/pyspark-notebook  # 사용할 도커 이미지 (Jupyter + PySpark 환경이 이미 세팅됨)
```

수고하셨습니다 ..^^..