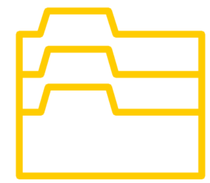
Yandex

Big Data

Scaling Distributed File System

Databases

Transaction logs

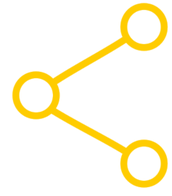


Document stores

Where does Big Data come from?

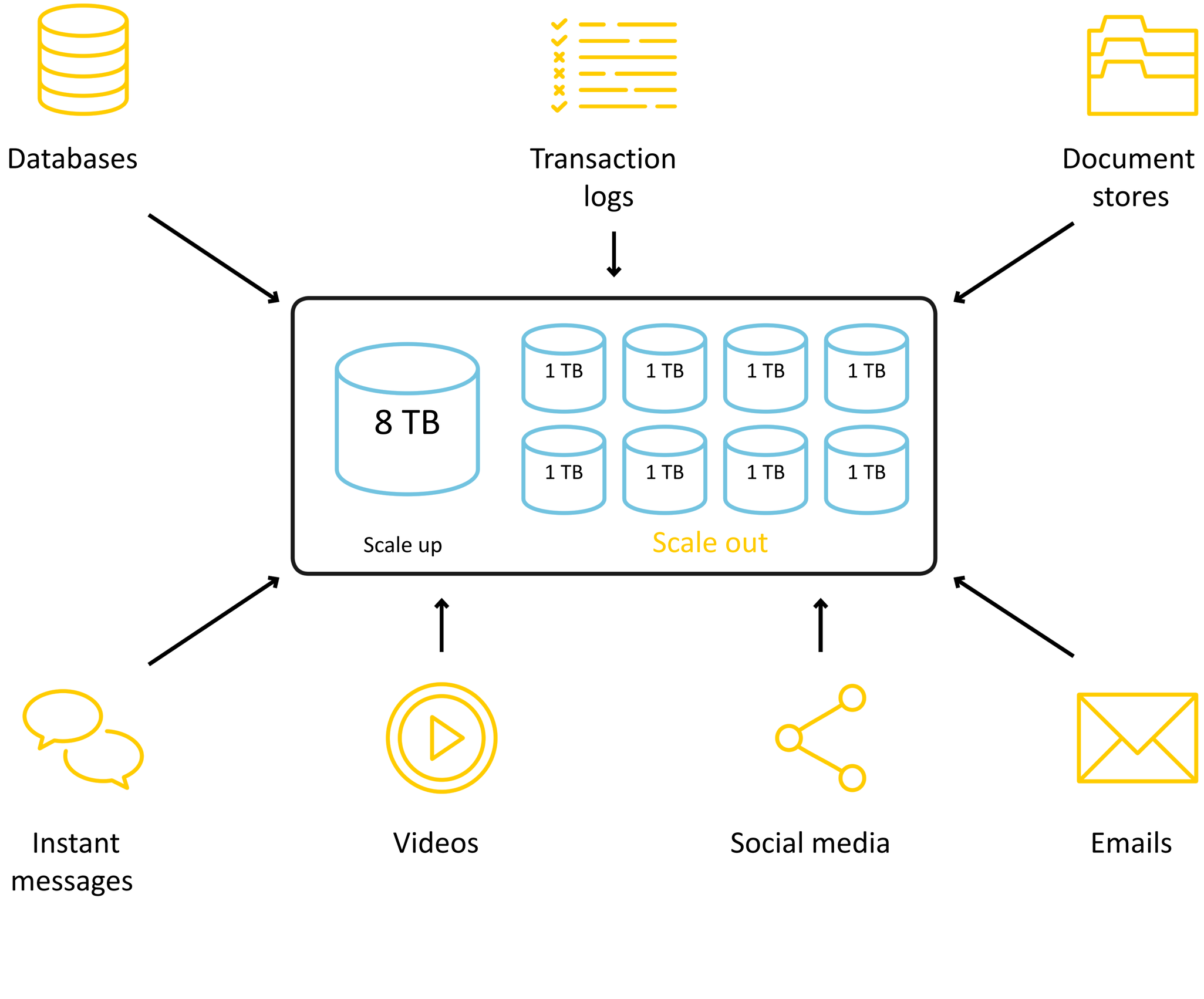
Instant messages

Videos



Social media

Emails


Databases
Transaction logs
Document stores
8 TB
1 TB
1 TB
1 TB
1 TB
1 TB
1 TB
1 TB
1 TB
Scale up
Scale out
Instant messages
Videos
Social media
Emails

The Google File System

Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung

Google*

2003

ABSTRACT

We have designed and implemented the Google File System, a scalable distributed file system for large distributed data-intensive applications. It provides fault tolerance while running on inexpensive commodity hardware, and it delivers high aggregate performance to a large number of clients.

While sharing many of the same goals as previous distributed file systems, our design has been driven by observations of our application workloads and technological environment, both current and anticipated, that reflect a marked departure from some earlier file system assumptions. This has led us to reexamine traditional choices and explore radically different design points.

The file system has successfully met our storage needs. It is widely deployed within Google as the storage platform for the generation and processing of data used by our service as well as research and development efforts that require large data sets. The largest cluster to date provides hundreds of terabytes of storage across thousands of disks on over a thousand machines, and it is concurrently accessed by hundreds of clients.

In this paper, we present file system interface extensions designed to support distributed applications, discuss many aspects of our design, and report measurements from both micro-benchmarks and real world use.

1. INTRODUCTION

We have designed and implemented the Google File System (GFS) to meet the rapidly growing demands of Google's data processing needs. GFS shares many of the same goals as previous distributed file systems such as performance, scalability, reliability, and availability. However, its design has been driven by key observations of our application workloads and technological environment, both current and anticipated, that reflect a marked departure from some earlier file system design assumptions. We have reexamined traditional choices and explored radically different points in the design space.

First, component failures are the norm rather than the exception. The file system consists of hundreds or even thousands of storage machines built from inexpensive commodity parts and is accessed by a comparable number of client machines. The quantity and quality of the components virtually guarantee that some are not functional at any given time and some will not recover from their current failures. We have seen problems caused by application bugs, operating system bugs, human errors, and the failures of disks, memory, connectors, networking, and power supplies. Therefore, constant monitoring, error detection, fault tolerance, and automatic recovery must be integral to the system.

Second, files are huge by traditional standards. Multi-GB

GFS Key Components

> components failures are a norm (→ replication)

시스템의 오류를 당연하다고 여겨 대응방안으로 replica 활용

> even space utilisation

파일을 일정한 사이즈로 나누어 여러 머신에 분배

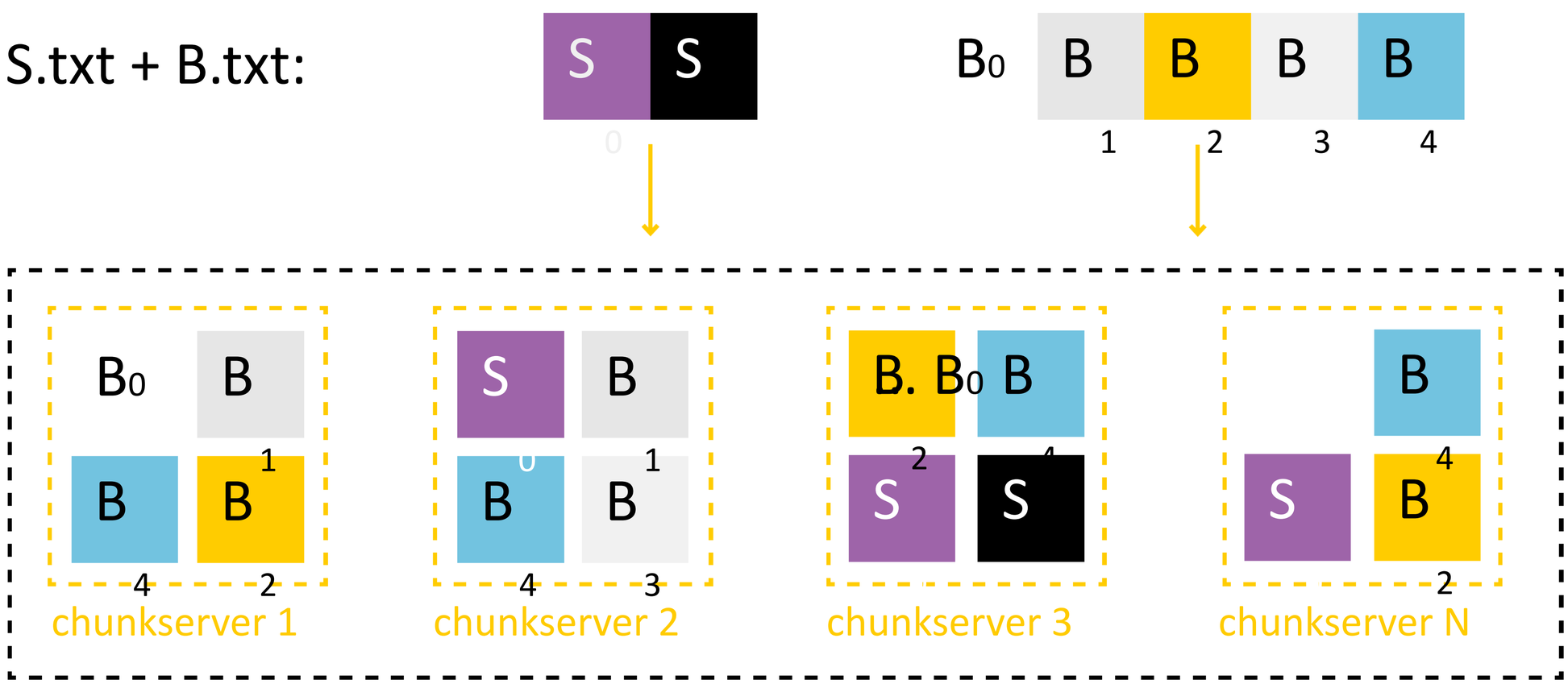
> write-once-read-many : data usage pattern

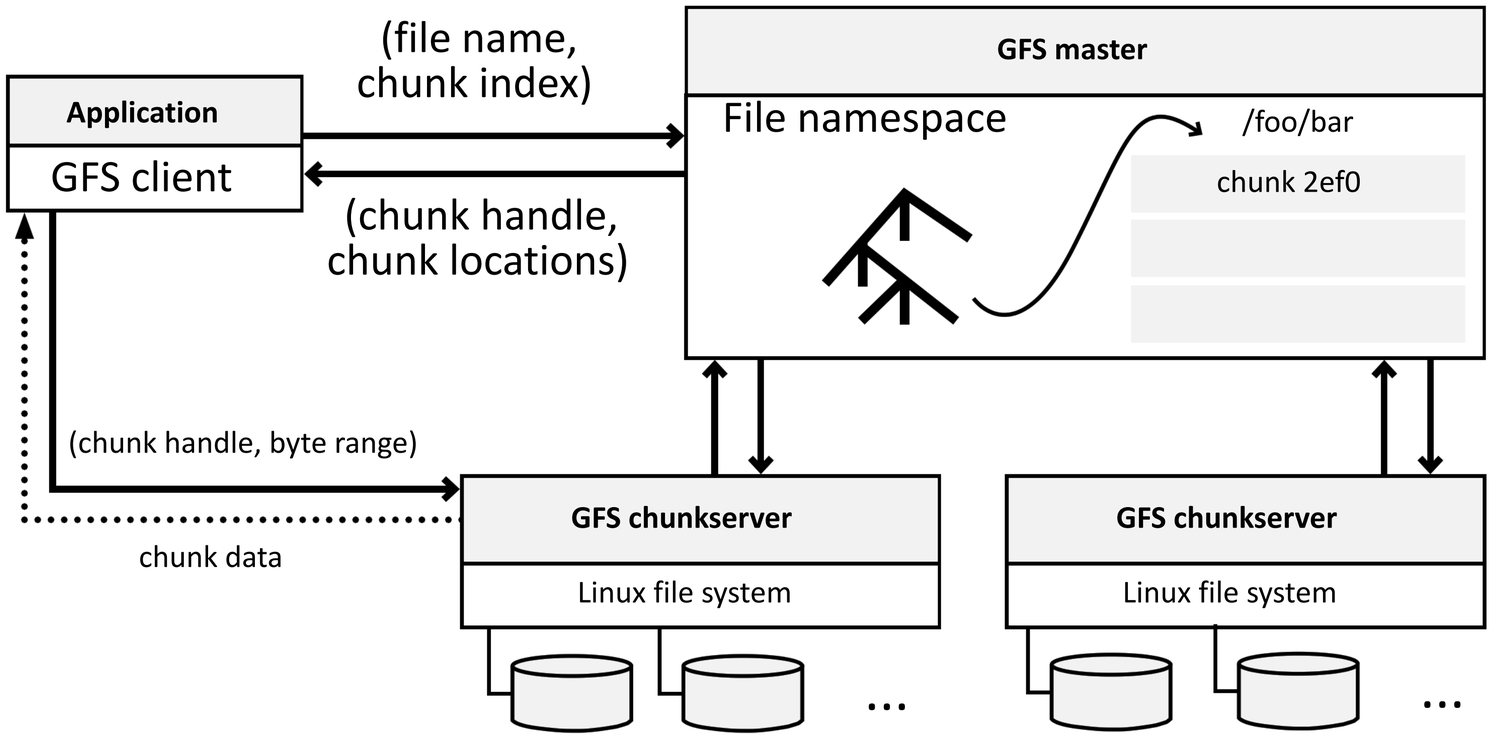
파일 중간 수정이 불가하다.

GFS는 batch computation을 위해 만들어졌기 때문

DFS 구현을 간단하게 하기 위함

Replication



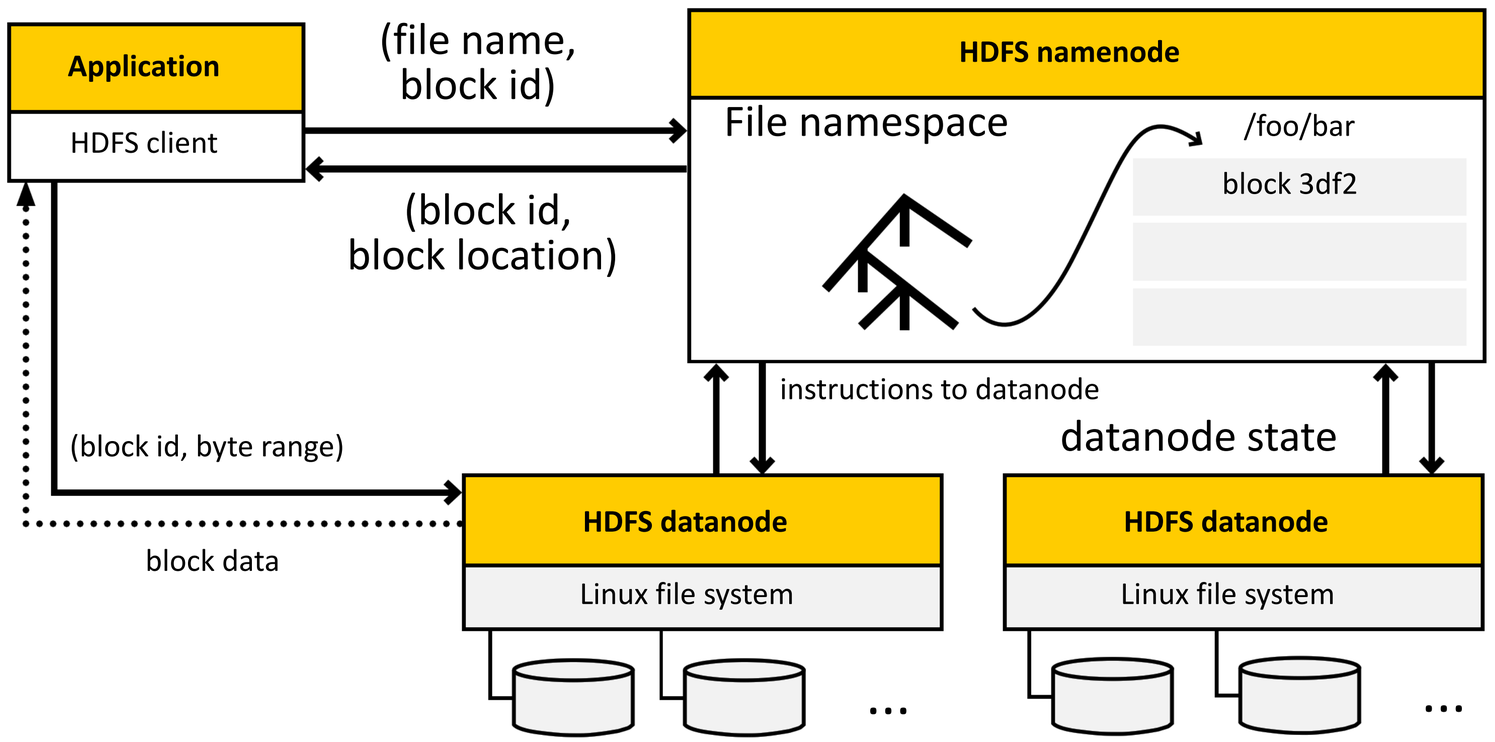


Master node

GFS, C++

Data messages

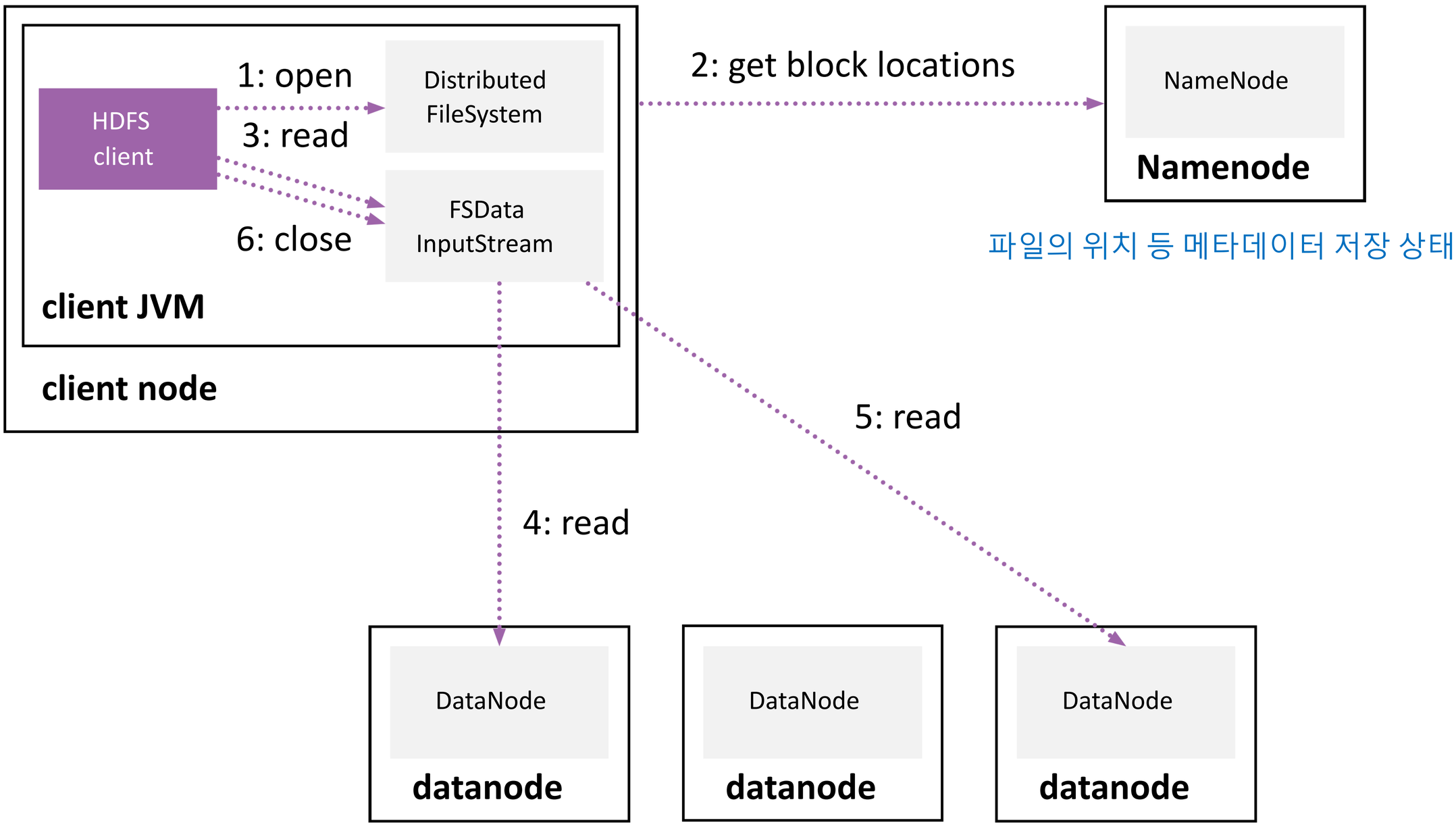
Control messages



Name node

HDFS, Java

How to read data

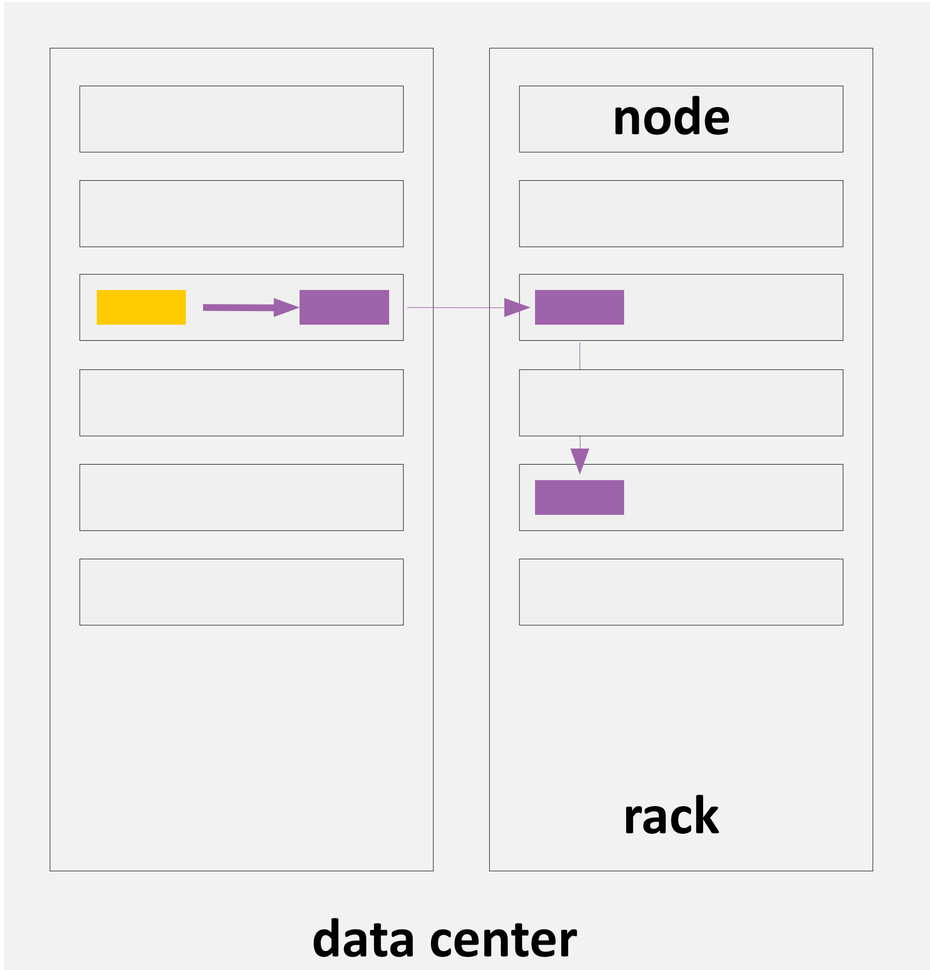


파일의 위치 등 메타데이터 저장 상태

Read 중 failure 발견 시 다른 block으로 자동으로 대체되어 처리한다.

Data Locality

Closeness를 계산하여 가장 가까운 위치에 데이터를 전달하기 위함

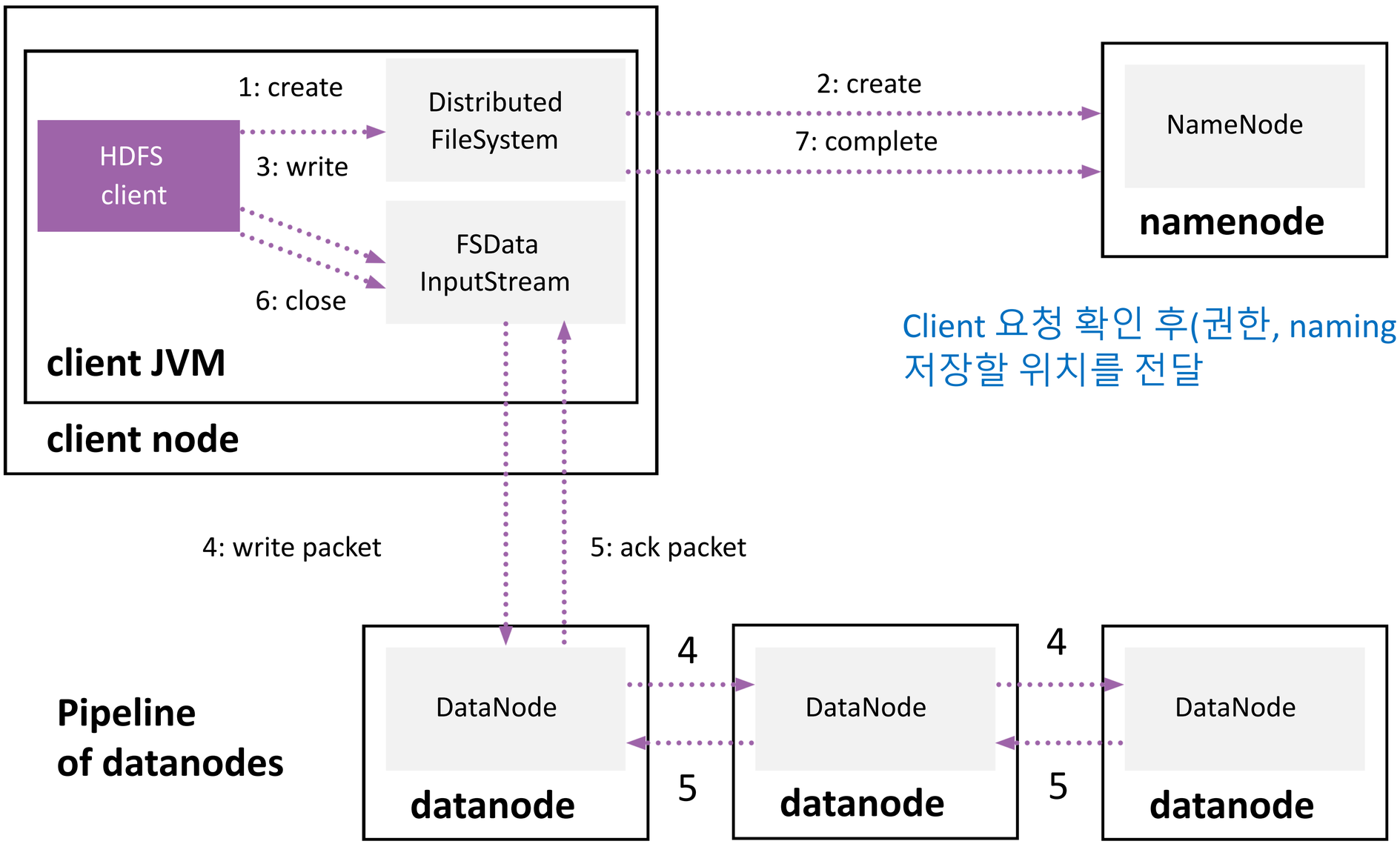


첫번째 복제는 보통 same node에 저장 (다른 node도 가능)

2번째 복제는 다른 rack에 저장

3번째 복제는 다른 rack의 다른 node로 저장한다.

How to write data



Client 요청 확인 후(권한, naming 충돌 등)
저장할 위치를 전달

가장 가까운 datanode에 write하면 datanode가 다른 node로 전달한다.

모두 완료되면 ack packet을 보낸다.

오류 발생 시, namenode에 새로운 위치를 요청 후 다시 처리한다.

Summary

> you can explain what vertical and horizontal scaling is

> you can list server roles in HDFS

> you can explain how topology affects replica placement

> you can explain what chunk / block size is used for

> you can explain in detail how HDFS client reads and writes data

BigDATAteam