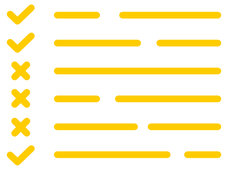
Yandex

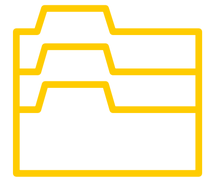
Big Data

Scaling Distributed File System

Databases



Transaction logs

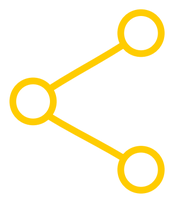


Document stores

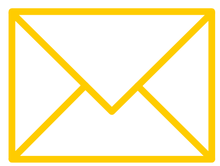
Where does Big Data come from?

Instant messages

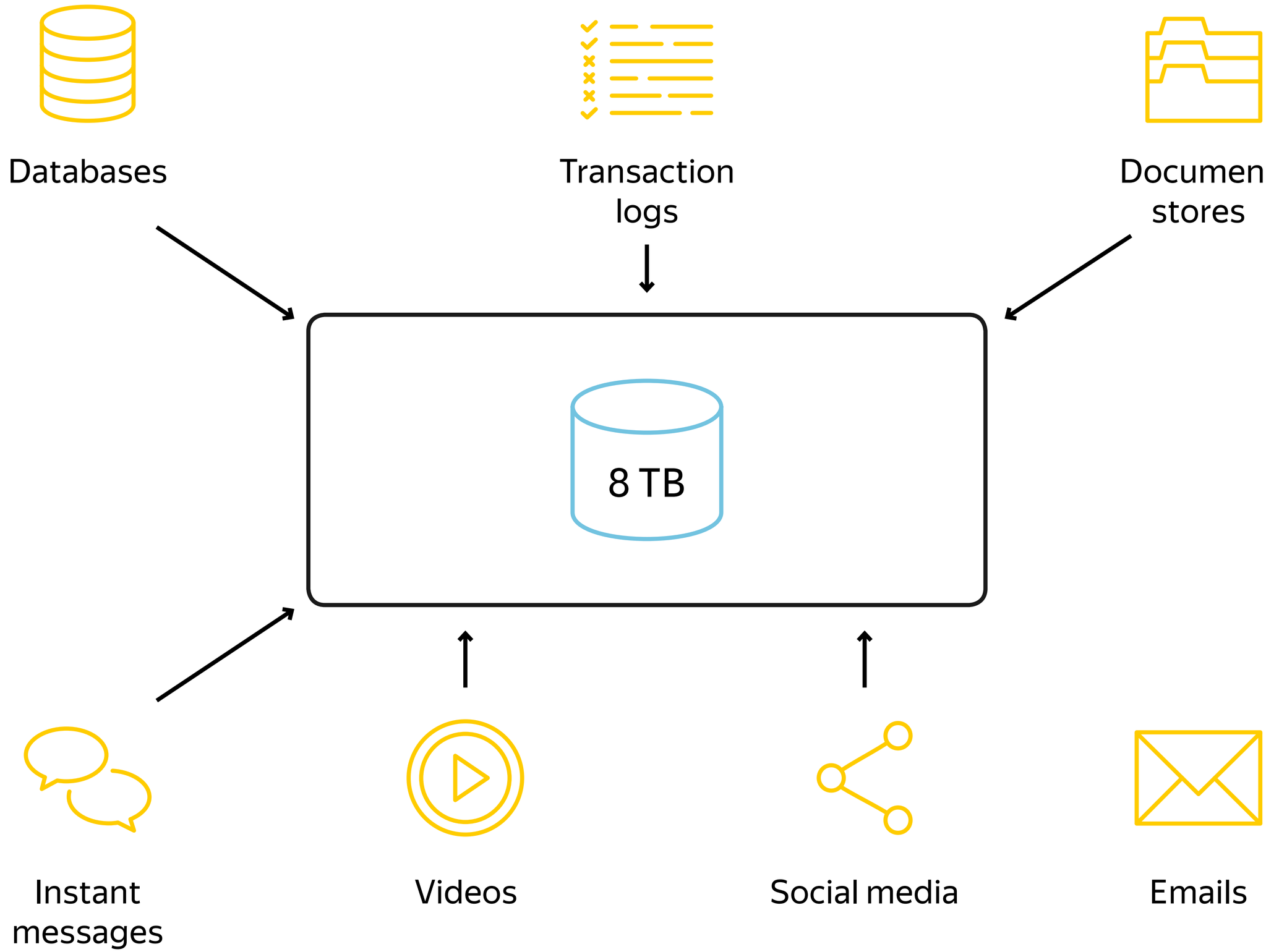
Videos

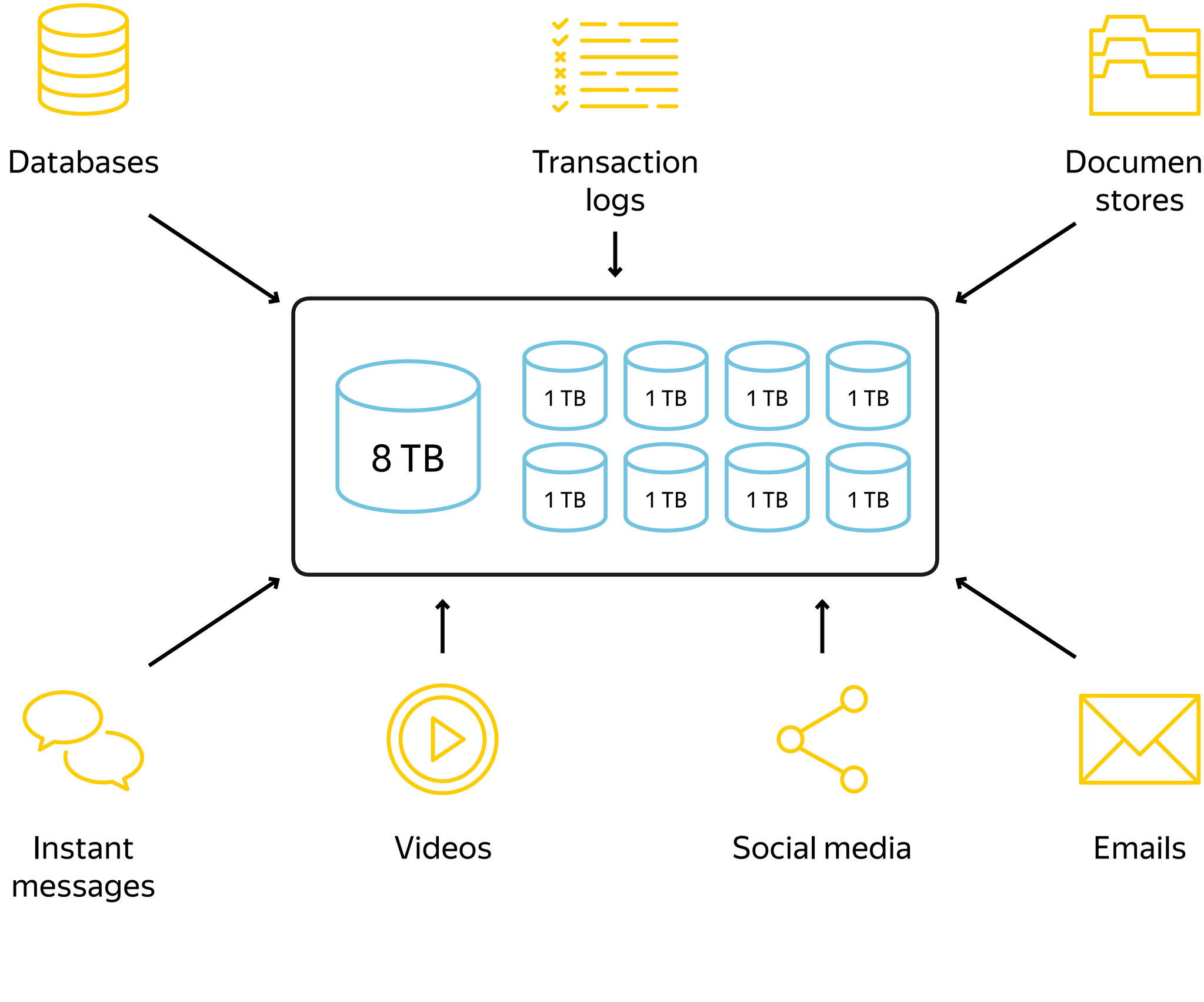


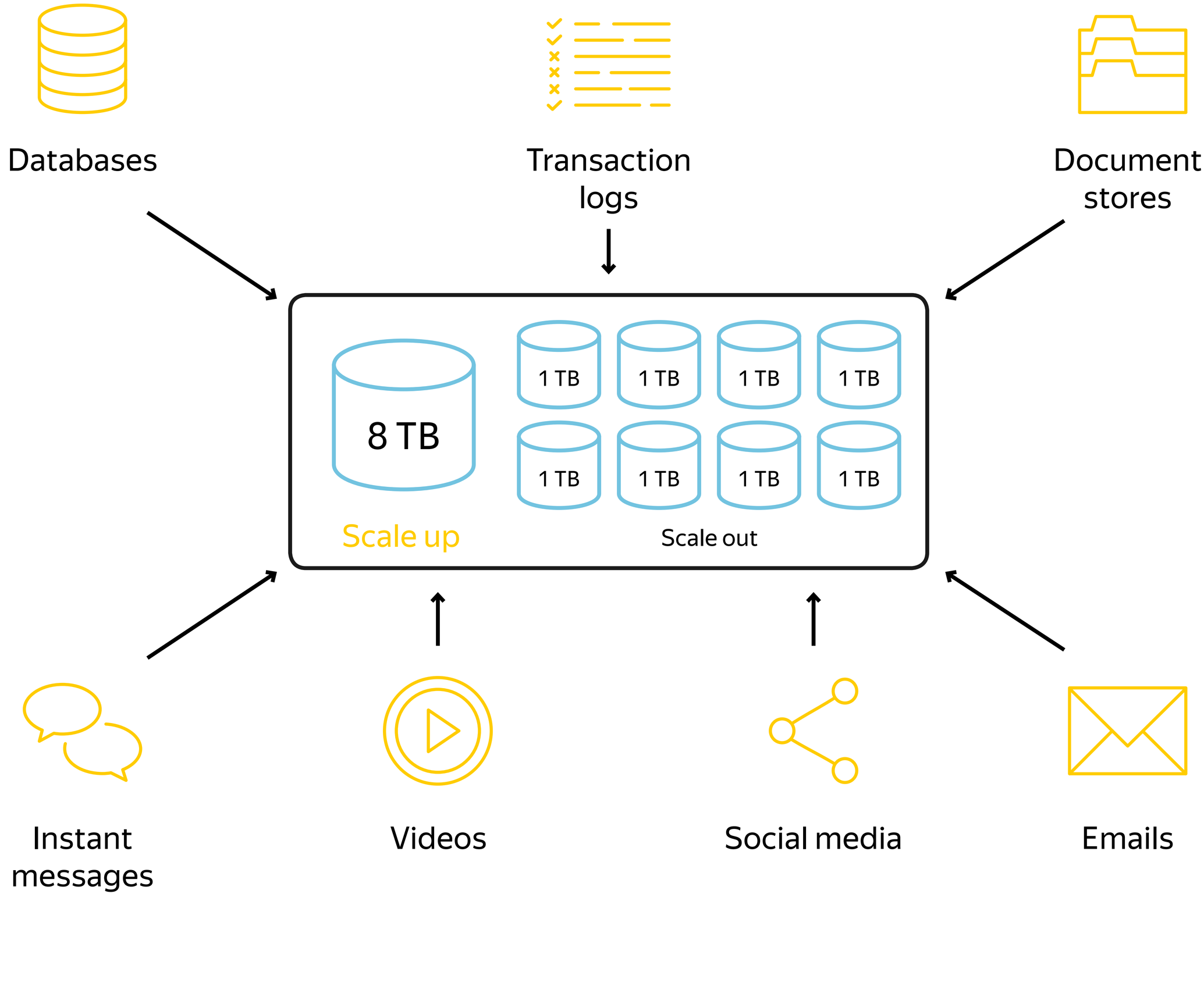
Social media

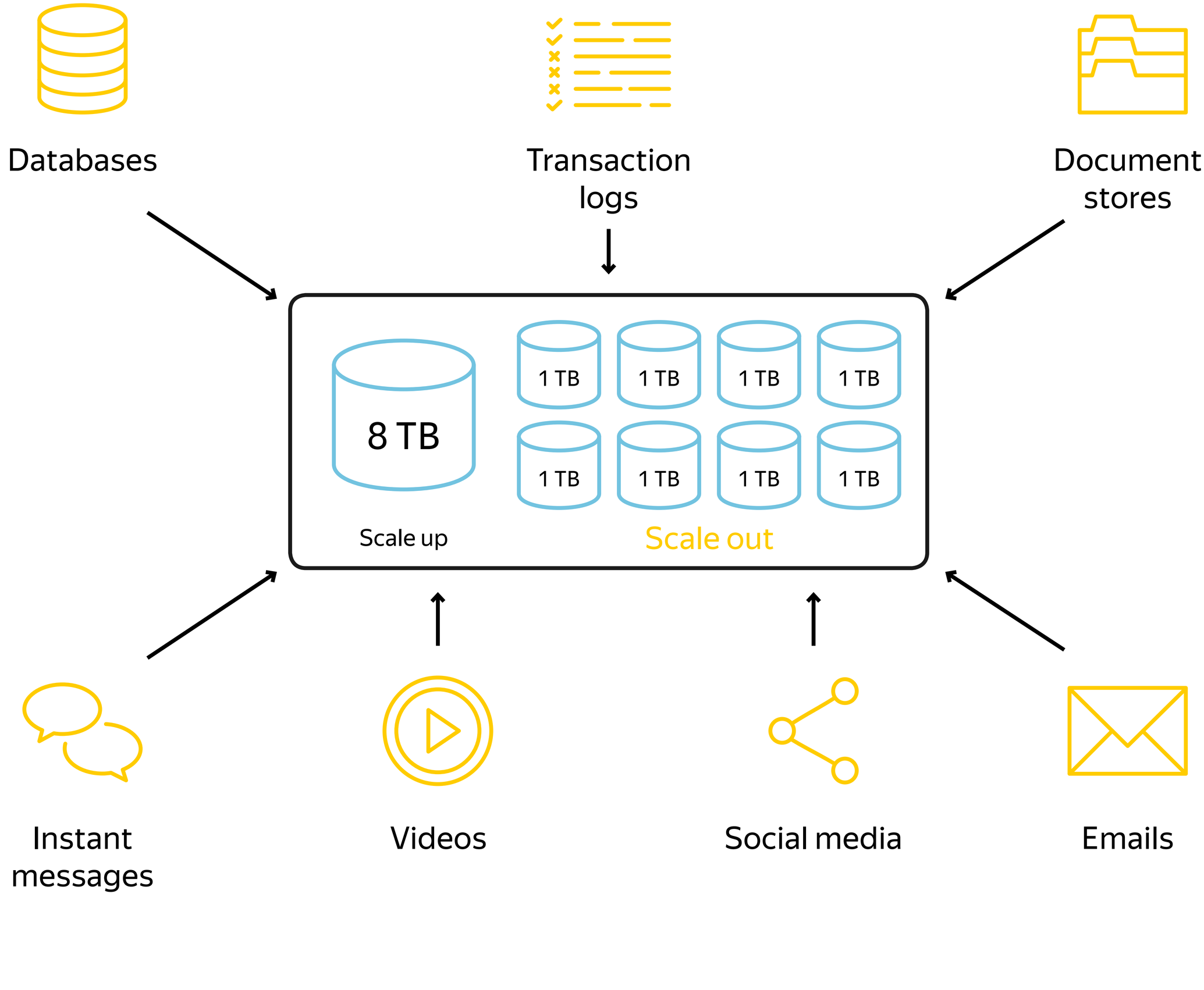


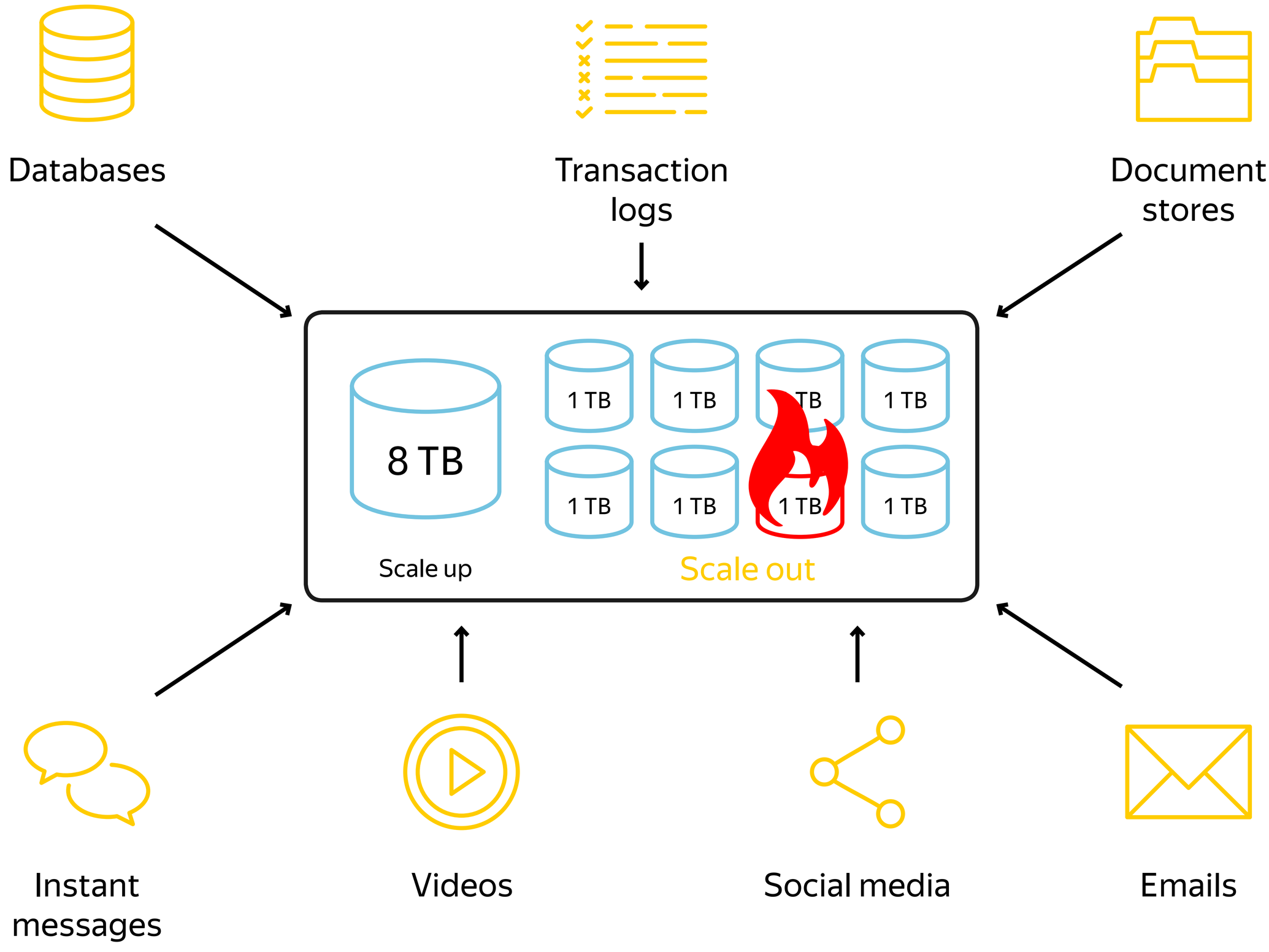
Emails


Databases
Transaction logs
Document stores
8 TB
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
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


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


Databases
Transaction logs
Document stores
8 TB
1 TB
1 TB
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*

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.

The Google File System, Symposium on Operating Systems Principles (SOSP, 2003)

GFS Key Components

GFS Key Components

- components failures are a norm (→ replication)

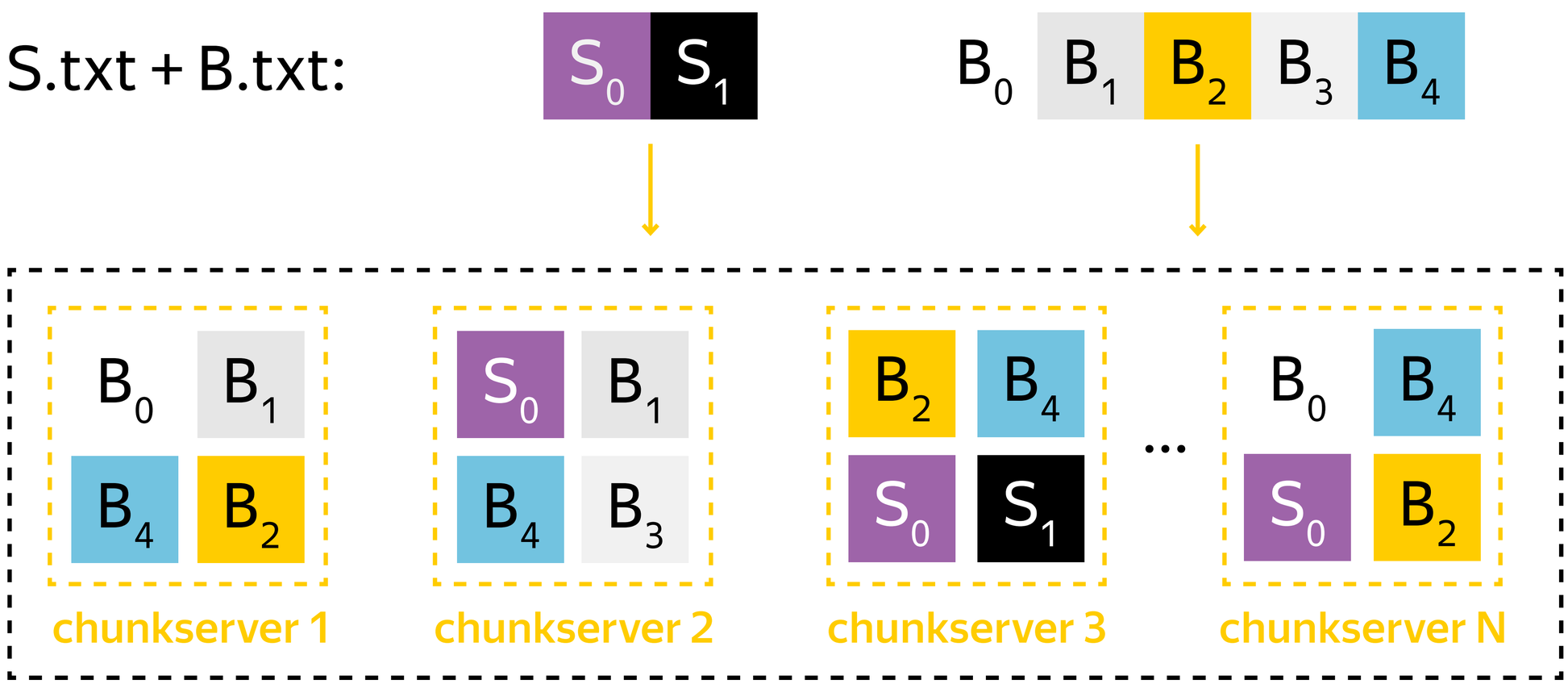
GFS Key Components

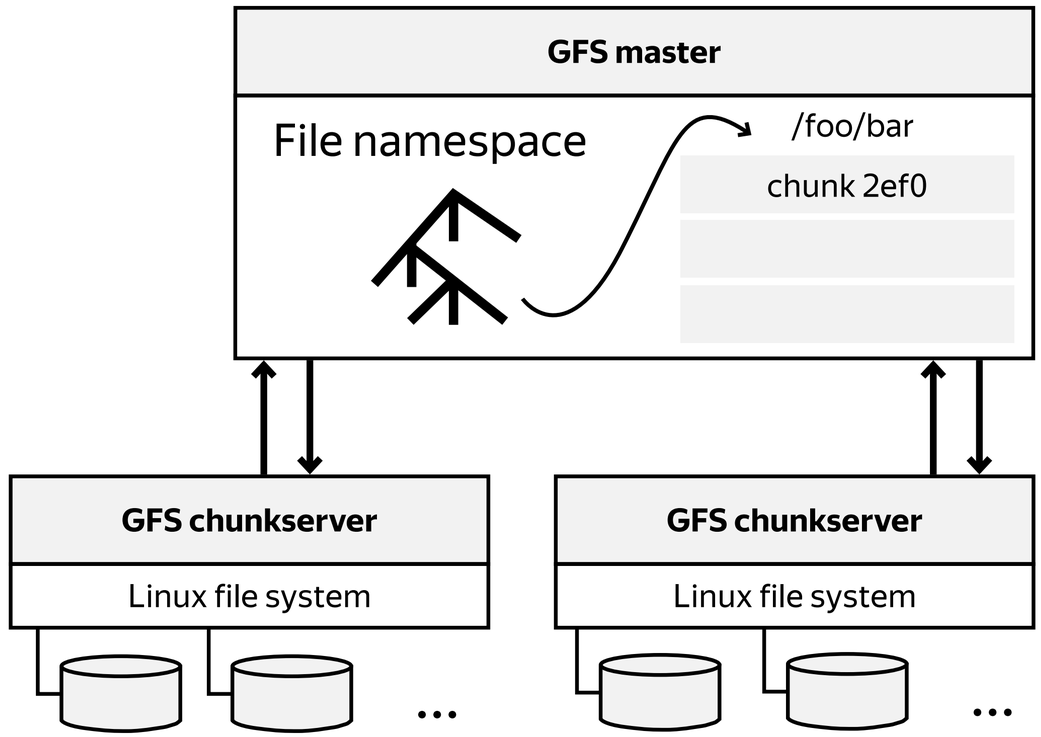
- components failures are a norm (→ replication)
- even space utilisation

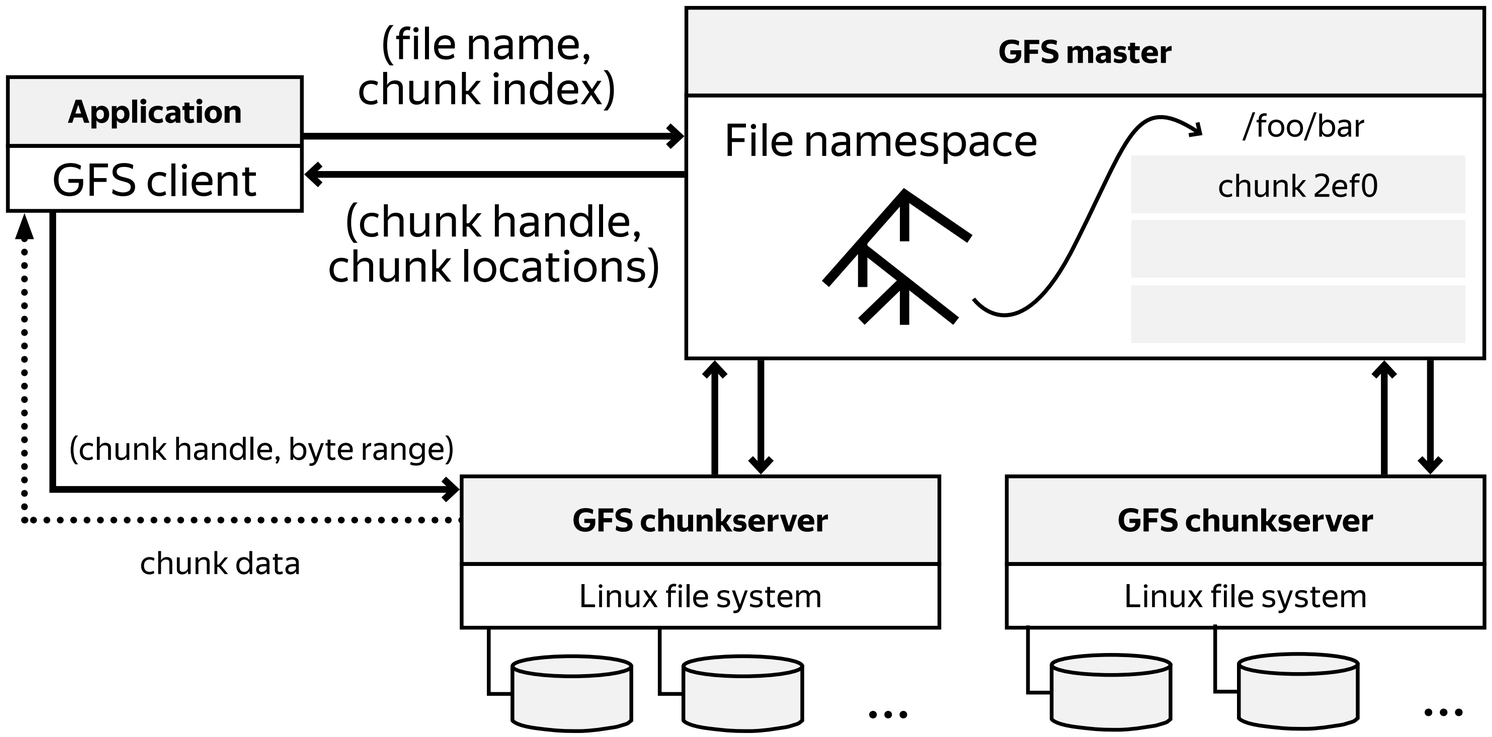
GFS Key Components

- components failures are a norm (→ replication)
- even space utilisation
- write-once-read-many

Replication

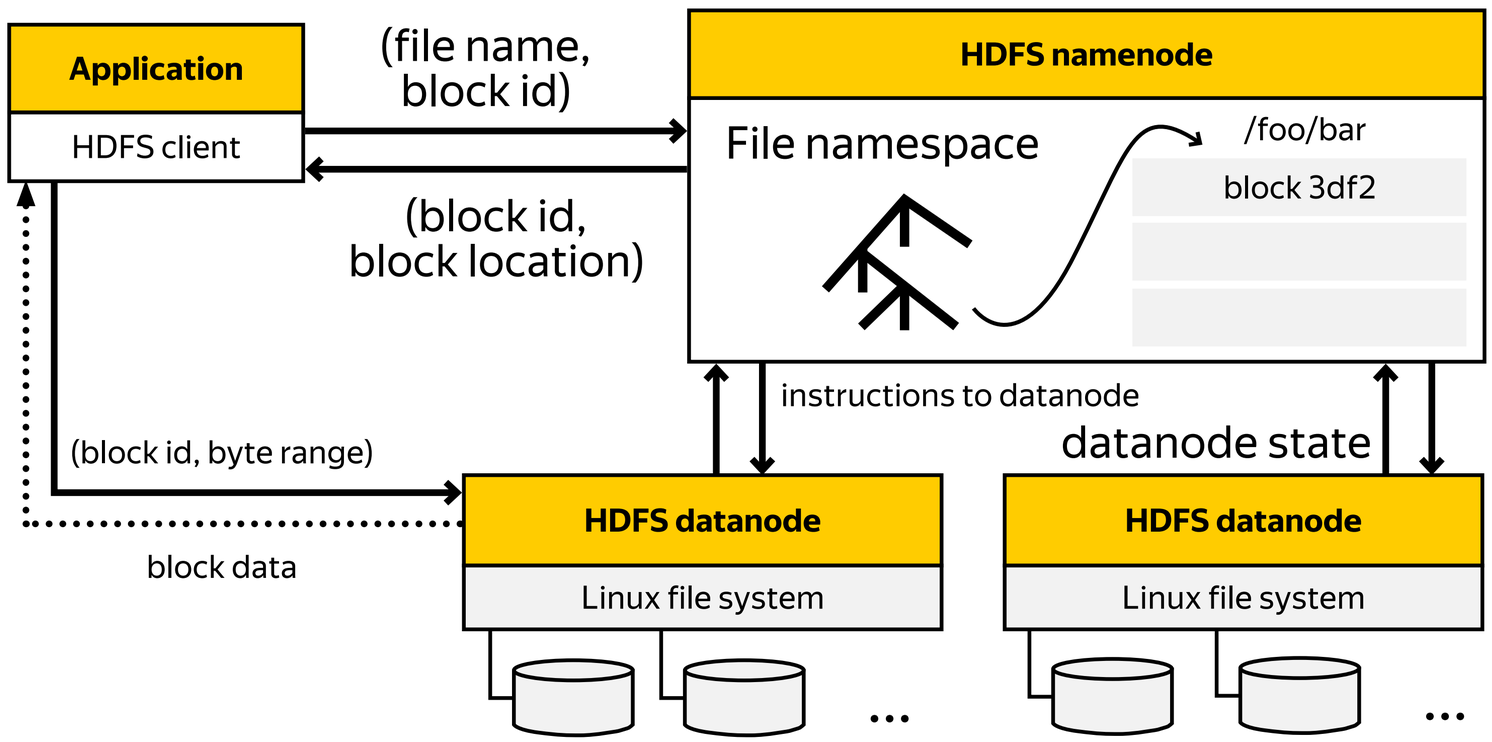



GFS master
File namespace
/foo/bar
chunk 2ef0
GFS chunkserver
Linux file system
GFS chunkserver
Linux file system

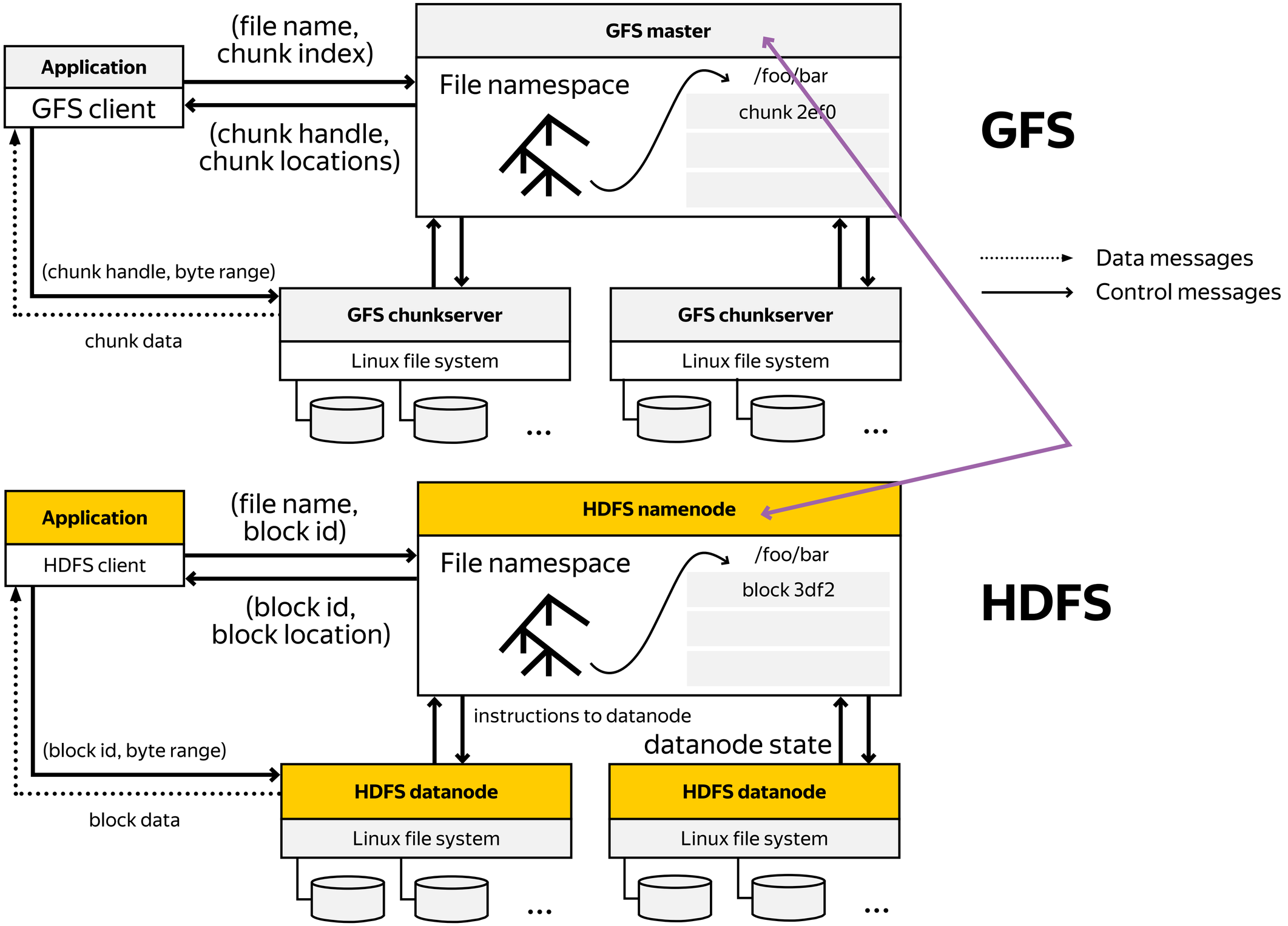

Application
GFS client
(file name, chunk index)
(chunk handle, chunk locations)
GFS master
File namespace
/foo/bar
chunk 2ef0
(chunk handle, byte range)
chunk data
GFS chunkserver
Linux file system
GFS chunkserver
Linux file system

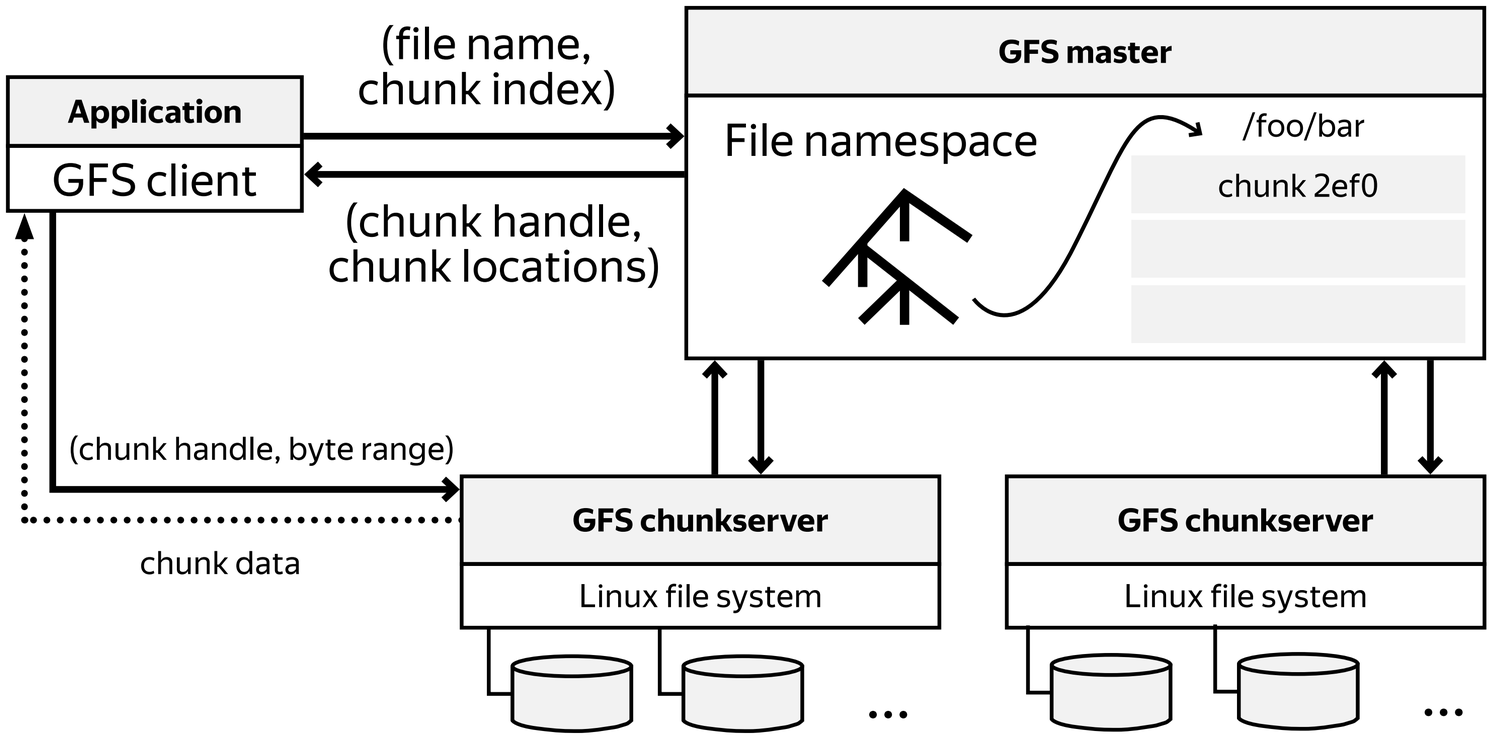
GFS


Data messages
Control messages


Application
HDFS client
(file name, block id)
(block id, block location)
HDFS namenode
File namespace
/foo/bar
block 3df2
instructions to datanode
datanode state
(block id, byte range)
block data
HDFS datanode
Linux file system
HDFS datanode
Linux file system

HDFS

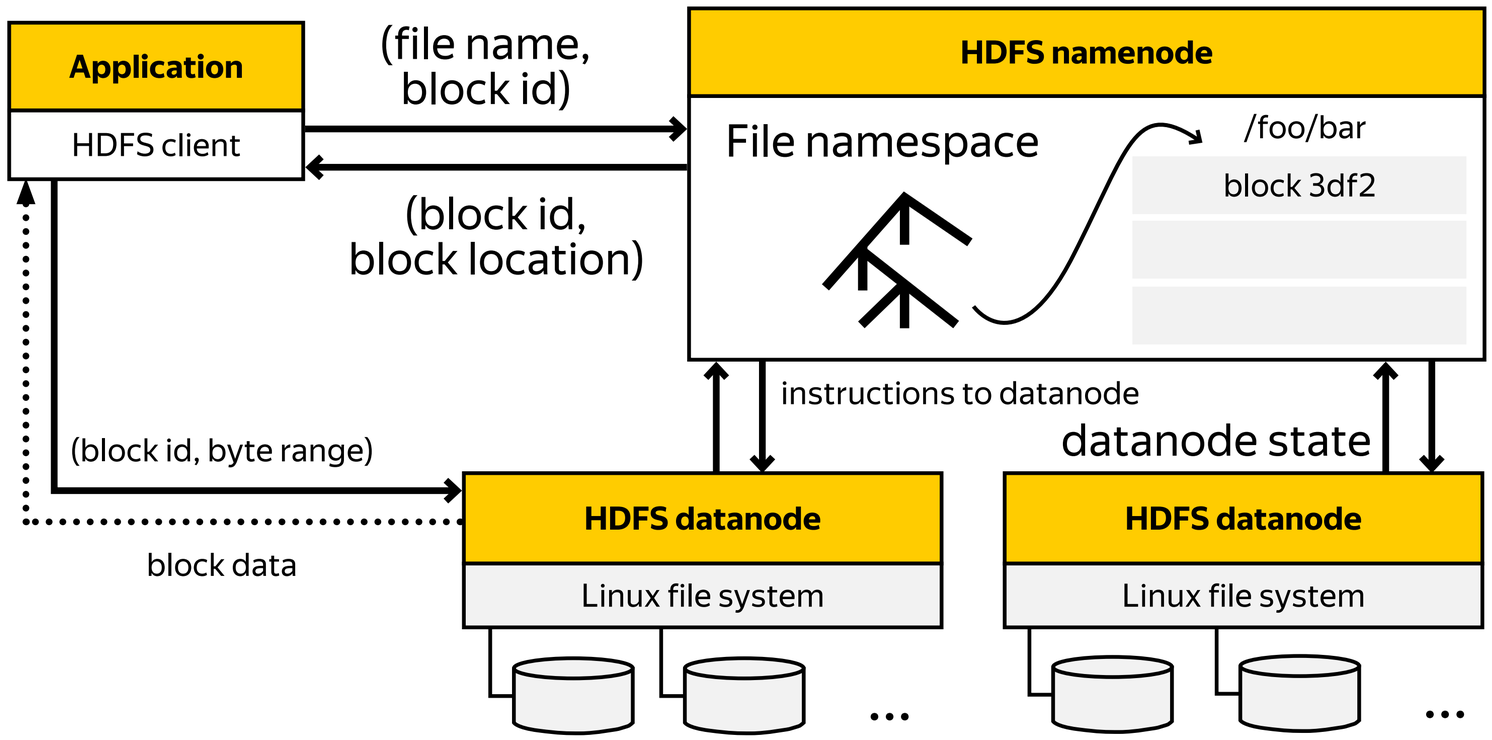

GFS master
Application
GFS client
(file name, chunk index)
(chunk handle, chunk locations)
File namespace
/foo/bar
chunk 2ef0
(chunk handle, byte range)
chunk data
GFS chunkserver
Linux file system
GFS chunkserver
Linux file system
GFS
Data messages
Control messages
HDFS namenode
Application
HDFS client
(file name, block id)
(block id, block location)
File namespace
/foo/bar
block 3df2
instructions to datanode
datanode state
(block id, byte range)
block data
HDFS datanode
Linux file system
HDFS datanode
Linux file system
HDFS



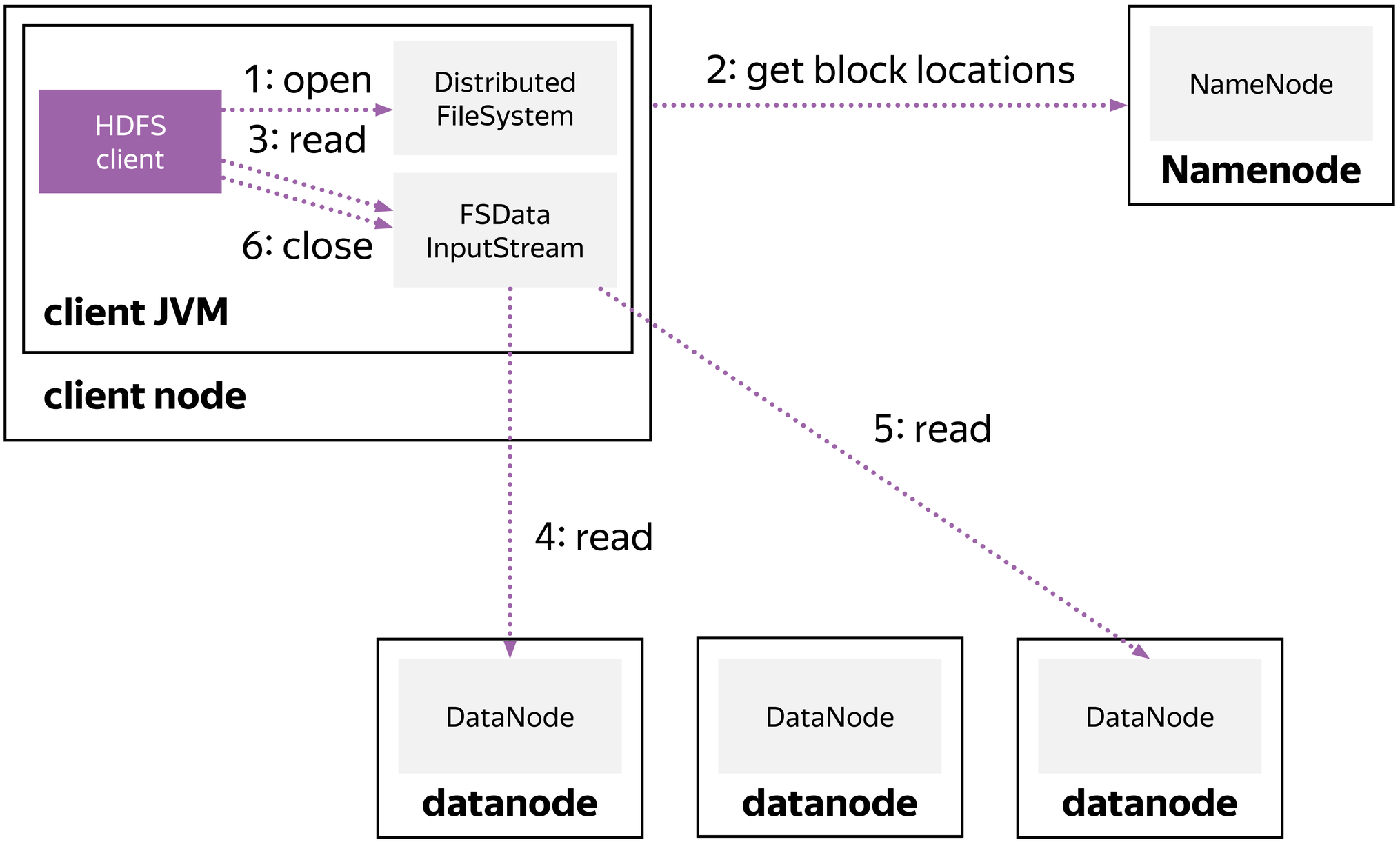
GFS, C++

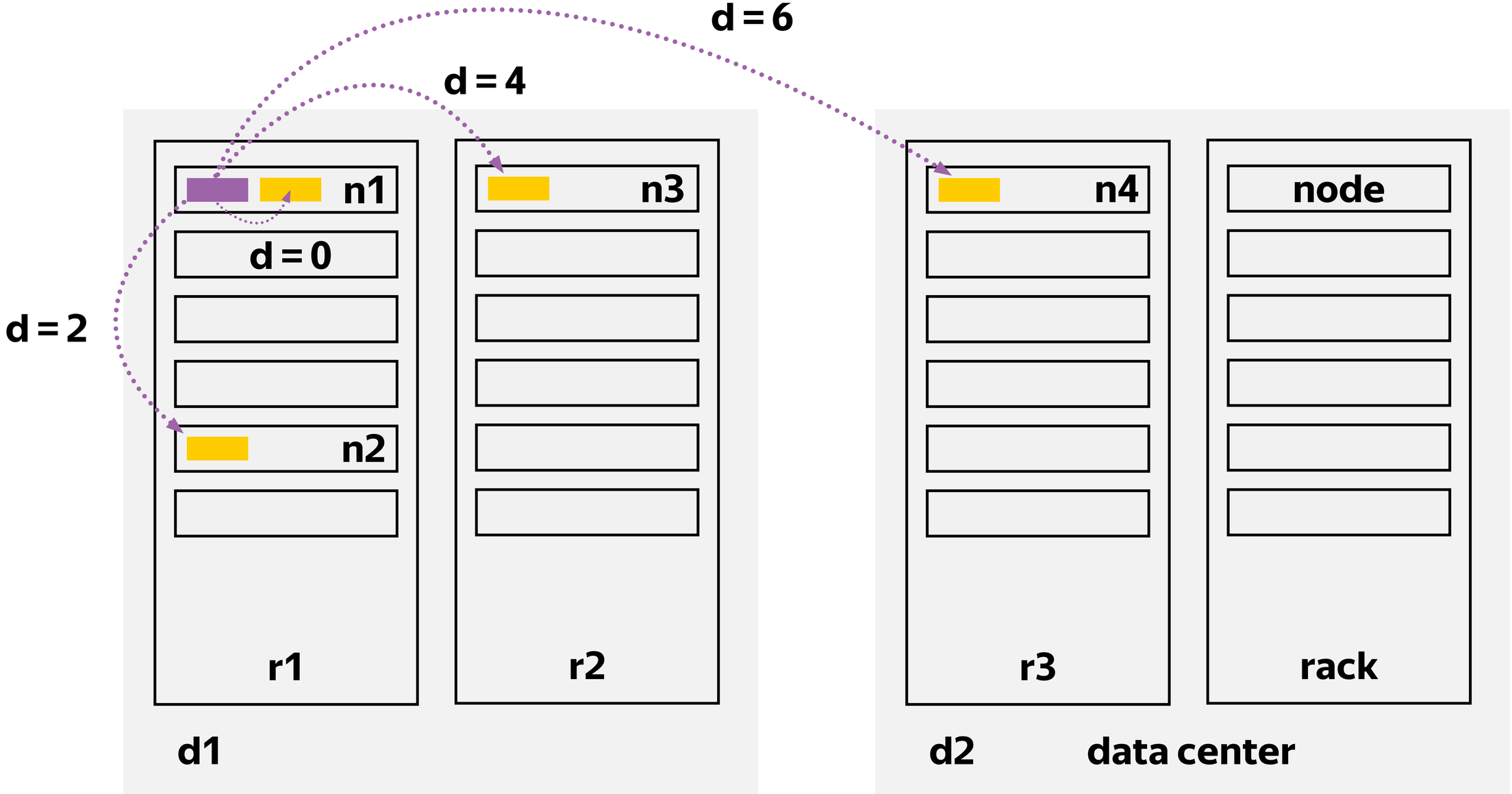
Data messages

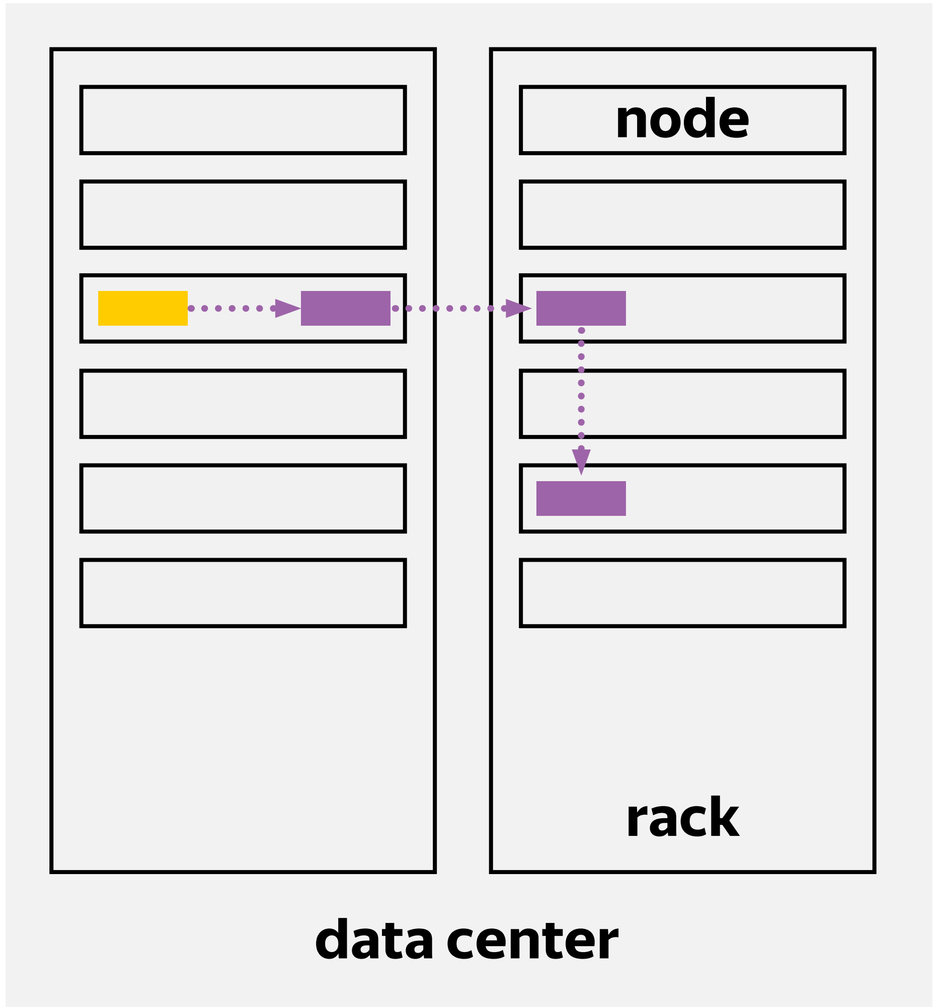
Control messages

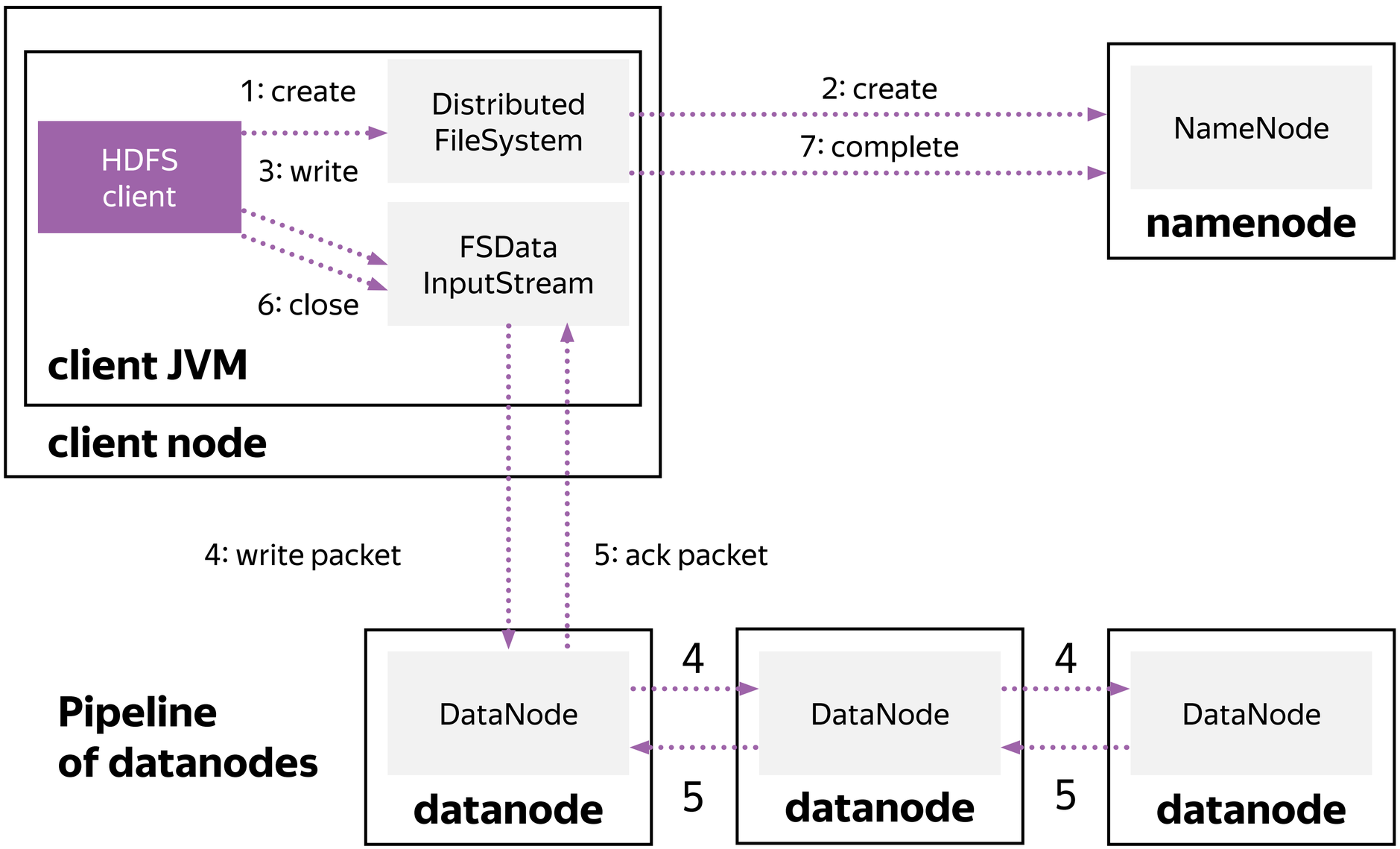


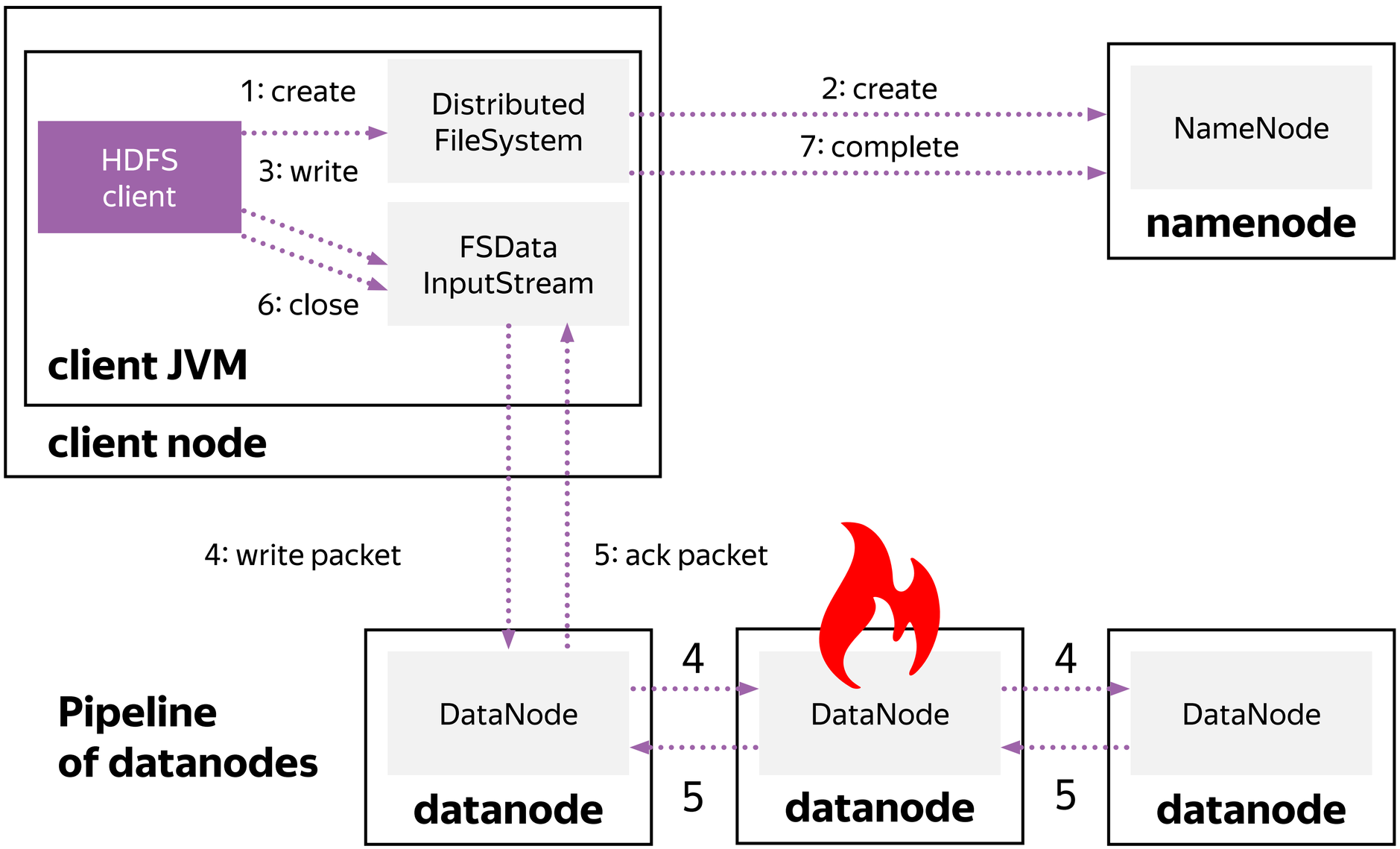
HDFS, Java

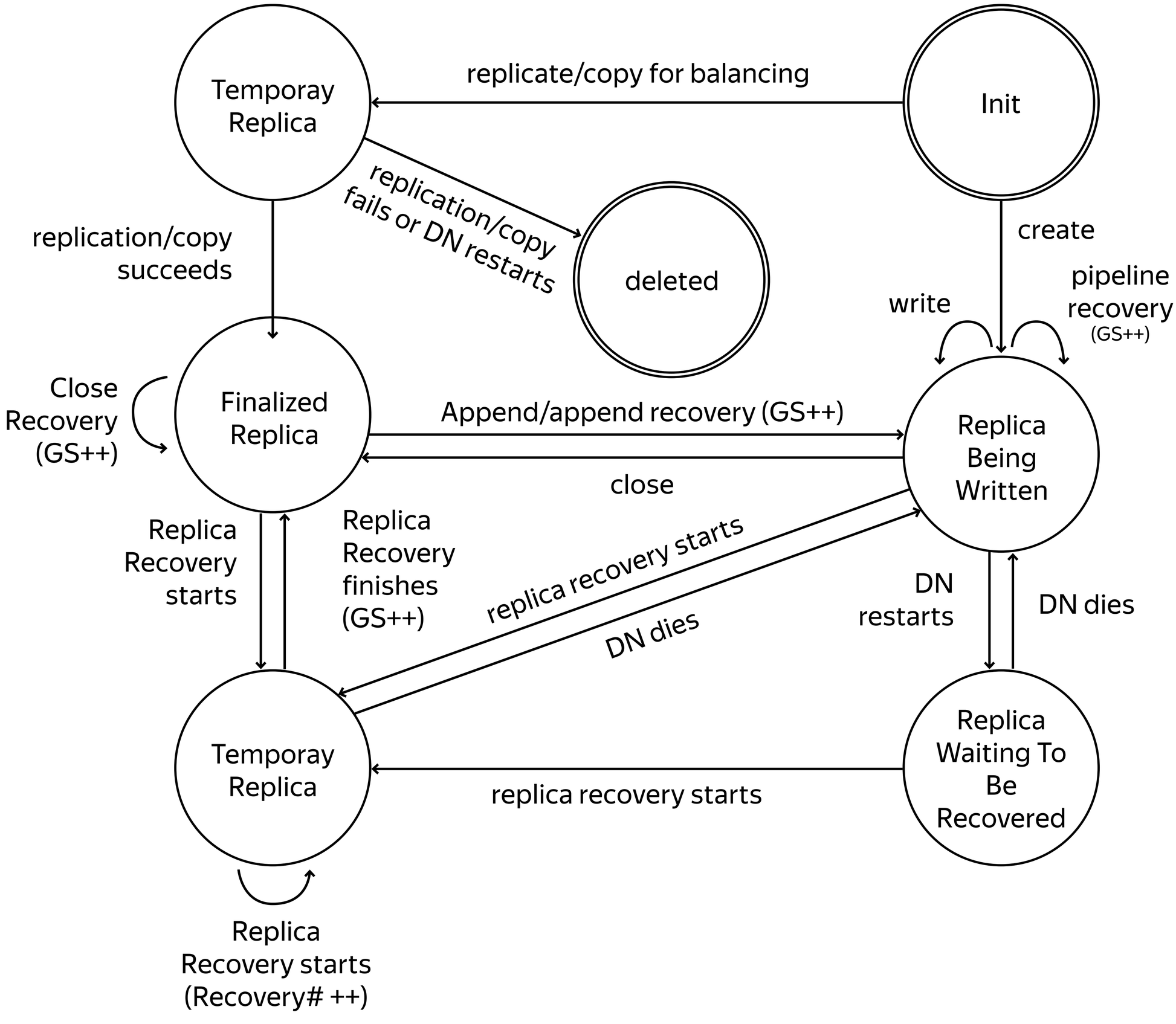

HDFS client
1: open
3: read
6: close
Distributed FileSystem
FSData InputStream
client JVM
client node
2: get block locations
NameNode
Namenode
4: read
5: read
DataNode
datanode
DataNode
datanode
DataNode
datanode


d = 6
d = 4
d = 2
n1
n3
n4
node
d = 0
n2
r1
r2
r3
rack
d1
d2
data center


node
rack
data center


1: create
HDFS client
3: write
6: close
Distributed FileSystem
FSData InputStream
client JVM
client node
2: create
7: complete
NameNode
namenode
4: write packet
5: ack packet
Pipeline of datanodes
DataNode
datanode
4
5
DataNode
datanode
4
5
DataNode
datanode


1: create
HDFS client
3: write
6: close
Distributed FileSystem
FSData InputStream
client JVM
client node
2: create
7: complete
NameNode
namenode
4: write packet
5: ack packet
Pipeline of datanodes
DataNode
datanode
4
5
DataNode
datanode
4
5
DataNode
datanode


Temporay Replica
replicate/copy for balancing
Init
replication/copy succeeds
replication/copy fails or DN restarts
deleted
create
write
pipeline recovery (GS++)
Close Recovery (GS++)
Finalized Replica
Append/append recovery (GS++)
Replica Being Written
close
Replica Recovery starts
Replica Recovery finishes (GS++)
replica recovery starts
DN dies
DN restarts
DN dies
Temporay Replica
Replica Waiting To Be Recovered
replica recovery starts
Replica Recovery starts (Recovery# ++)

Summary

Summary

- you can **explain** what vertical and horizontal scaling is

Summary

- you can **explain** what vertical and horizontal scaling is
- you can **list** server roles in HDFS

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

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

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