大数据导论
Introduction to Big Data

第3讲: 深入理解HDFS

叶允明

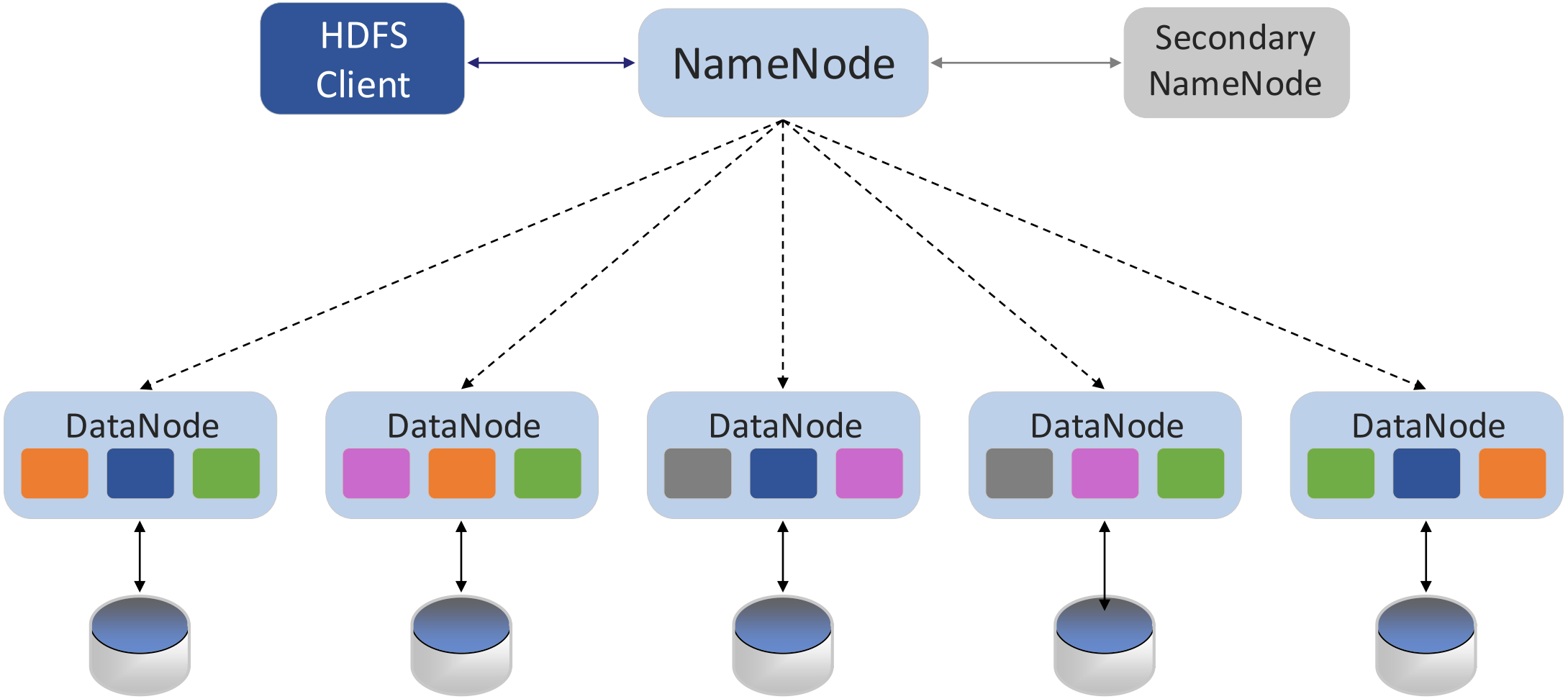
计算机科学与技术学院

哈尔滨工业大学（深圳）

关于数据存储方法的说明

- 本地文件系统
- 关系数据库（MySQL，Oracle，MS SQL Server等）
- NoSQL数据库：MongoDB，Redis，Neo4j、其它OSS等
- 分布式文件系统：大数据存储的重要方法

回顾：HDFS架构


HDFS
Client
NameNode
Secondary
NameNode
DataNode
DataNode
DataNode
DataNode
DataNode

目录

- 深入理解HDFS的存储模型
- 深入理解HDFS的组件
- HDFS的读写流程与容错恢复机制
- 总结与知识拓展

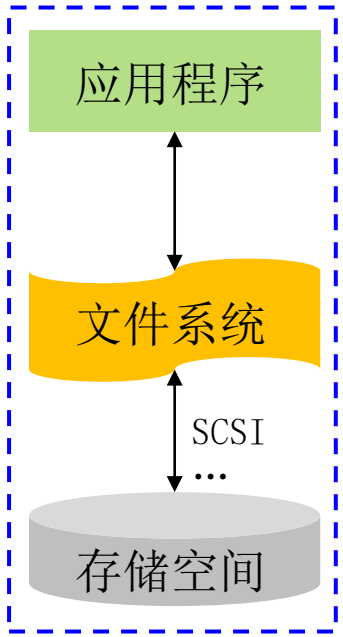
深入理解HDFS的存储模型

基本概念：块存储和文件存储

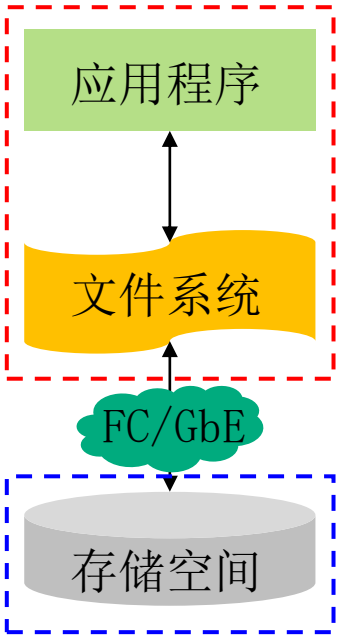
- 块存储和文件存储是两种传统的存储类型
- 块级（典型设备：磁盘阵列）
  - 以扇区为基础，一个或多个连续的扇区组成一个块，也叫物理块。
  - 存在于文件系统与块设备（例如：磁盘驱动器）之间
- 文件级（典型设备：NFS服务器）
  - 指文件系统，单个文件可能由于一个或多个逻辑块组成，且逻辑块之间可不连续分布
  - 逻辑块大于或等于物理块整数倍
- 物理块与文件系统之间的关系
  - 扇区→物理块→逻辑块→文件系统

常见的传统存储方案

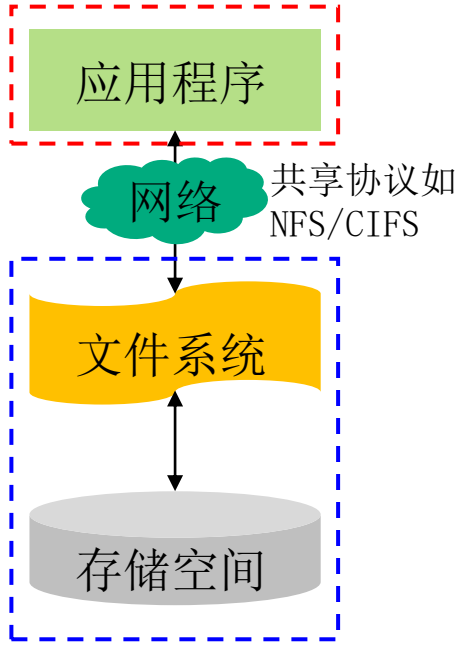
直接存储（DAS）
(Direct Attached Storage)



存储区域网络（SAN）
(Storage Area Network)



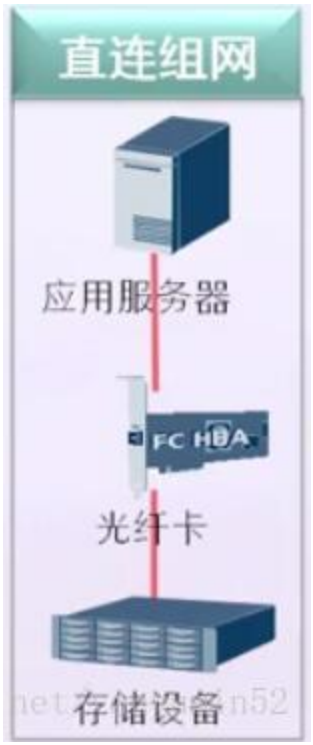
网络连接存储（NAS）
(Network Attached Storage)

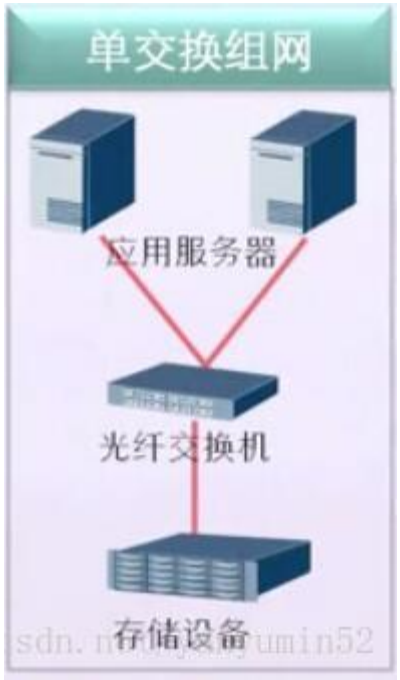


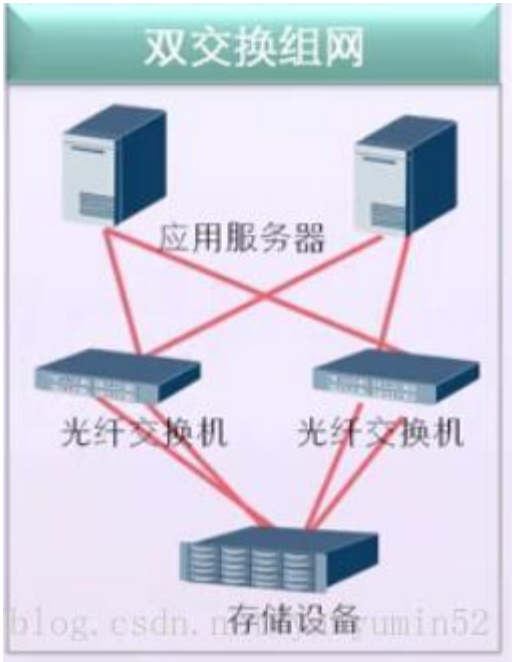
参考：https://zhuanlan.zhihu.com/p/350096155

备课页

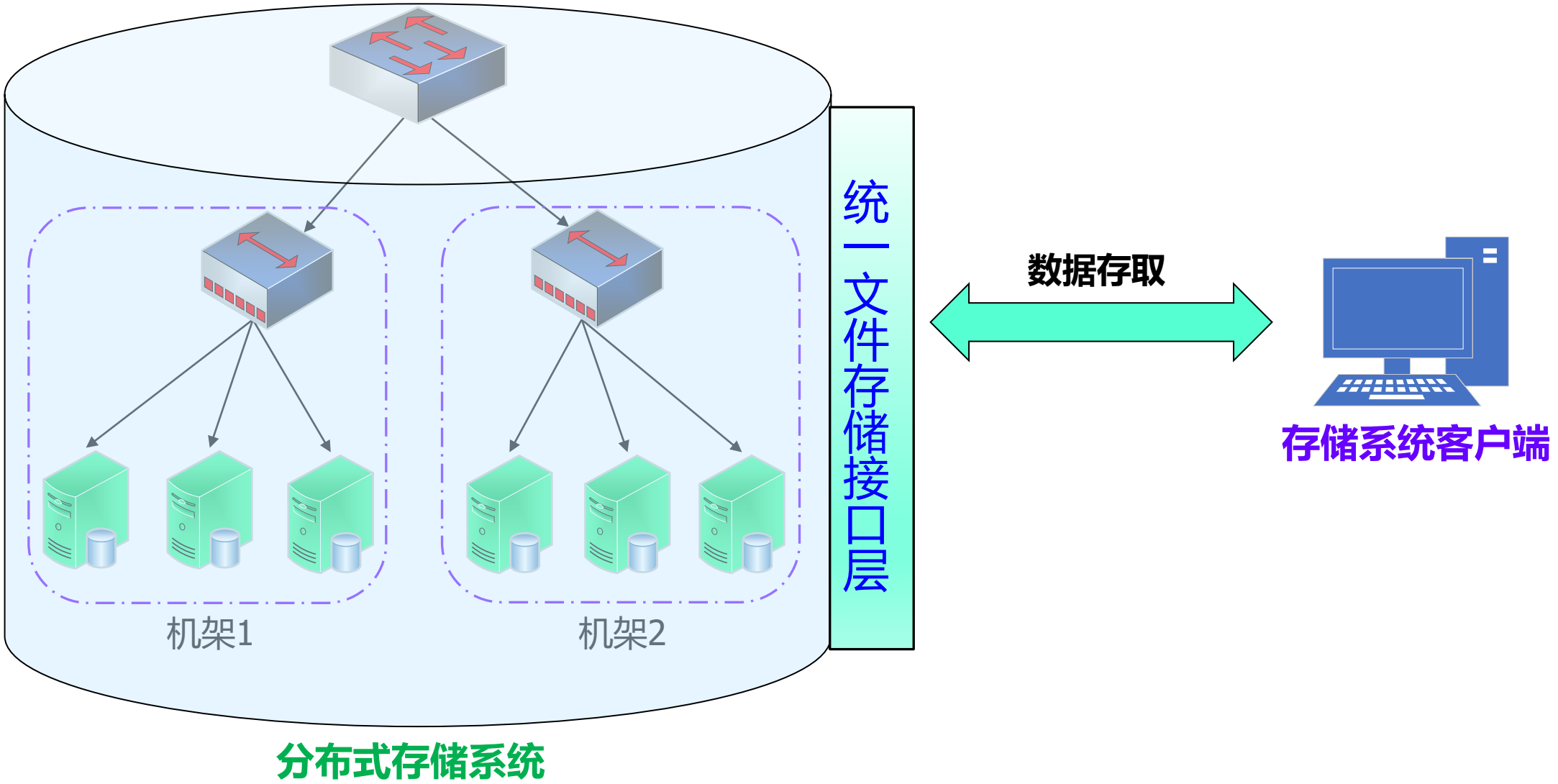

SAN存储
存储区域网络(Storage Area Networks，SAN)：是一个用在服务器和存储资源之间的、专用的、高性能的网络体系。 SAN是独立于LAN的服务器后端存储专用网络。 SAN采用可扩展的网络拓扑结构连接服务器和存储设备，每个存储设备不隶属于任何一台服务器，所有的存储设备都可以在全部的网络服务器之间作为对等资源共享
客户机
LAN
服务器
服务器
ETH Switch
FC Switch
存储阵列设备
存储阵列设备


直连组网
应用服务器
FC HBA
光纤卡
存储设备


单交换组网
应用服务器
光纤交换机
存储设备

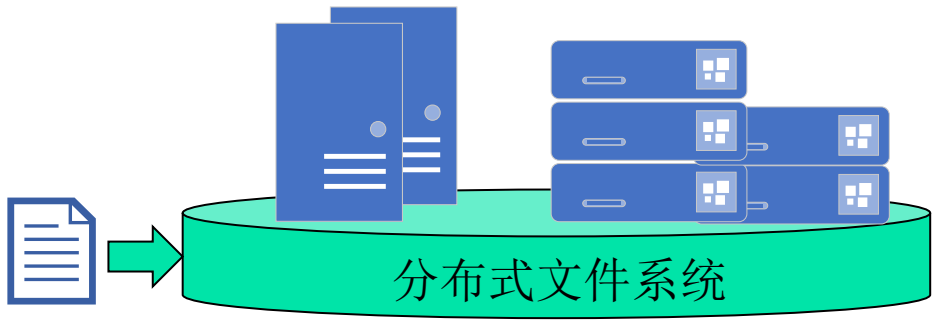

双交换组网
应用服务器
光纤交换机
光纤交换机
存储设备

基于分布式文件系统的大数据存储



基于分布式文件系统的存储方案优势

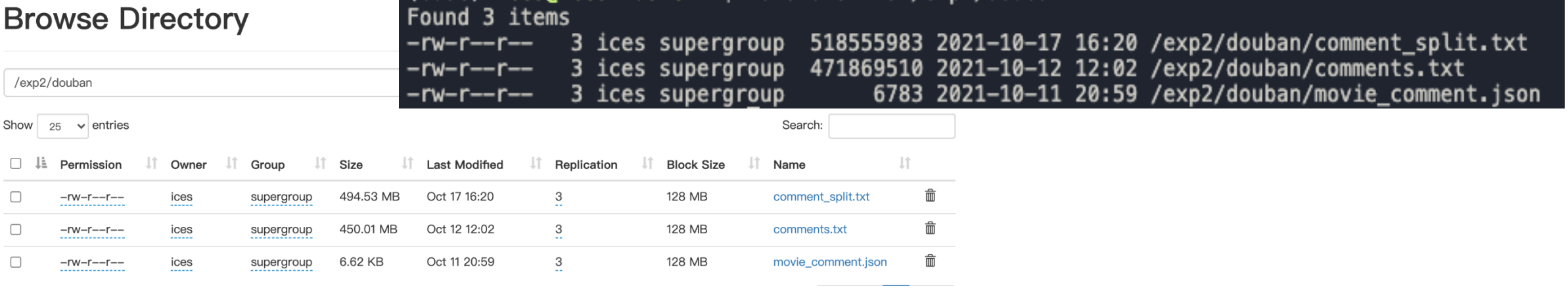
- 分布式文件系统改变了数据存储和管理方式，相对于本地文件系统具有很多优势：
  - 低成本
  - 易扩展
  - 强可靠
  - 高可用



- 用户无需关心数据是存储在哪个节点上，可以如同使用本地文件系统一样存储和管理分布式文件系统里的数据。

HDFS的逻辑存储模型：文件系统

- 以“文件”为基本的逻辑存储单位，形成文件系统
- 分级的文件系统：其命名空间包含目录、文件
  - 用户可象使用普通文件系统一样创建、删除目录和文件，在目录间转移文件、重命名文件

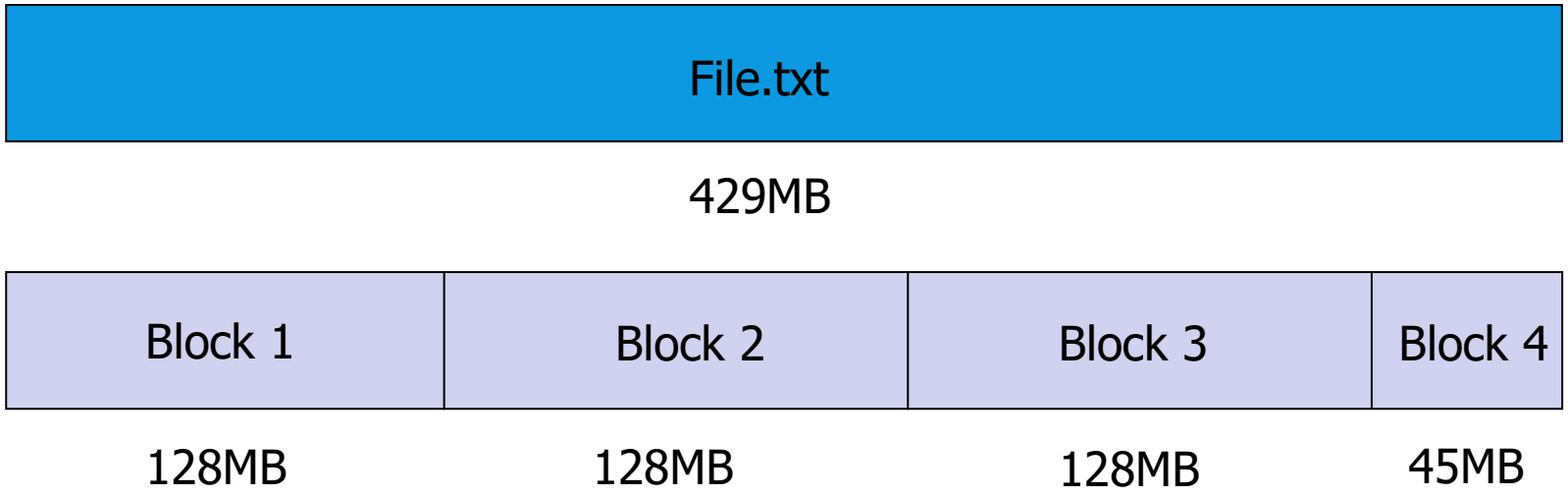



```
(base) ices@ices-master:~$ hdfs dfs -ls /exp2/douban
Found 3 items
-rw-r--r--   3 ices supergroup  518555983 2021-10-17 16:20 /exp2/douban/comment_split.txt
-rw-r--r--   3 ices supergroup  471869510 2021-10-12 12:02 /exp2/douban/comments.txt
-rw-r--r--   3 ices supergroup       6783 2021-10-11 20:59 /exp2/douban/movie_comment.json
```

- 整个HDFS集群中只有一个统一的命名空间（由一个名称节点管理）

HDFS的逻辑存储模型：数据块

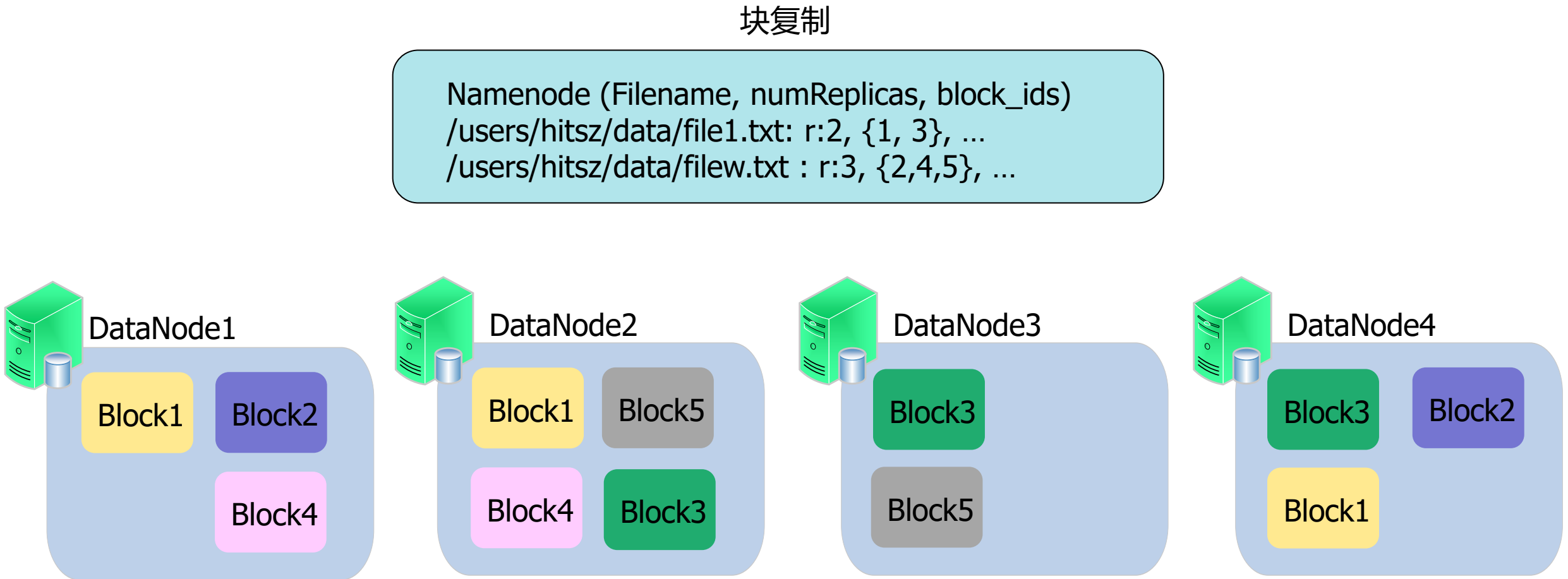
- 每个HDFS文件以一个或多个数据块进行存储（每个块64MB/128MB）



- 块的大小远远大于普通文件系统，可以最小化寻址开销

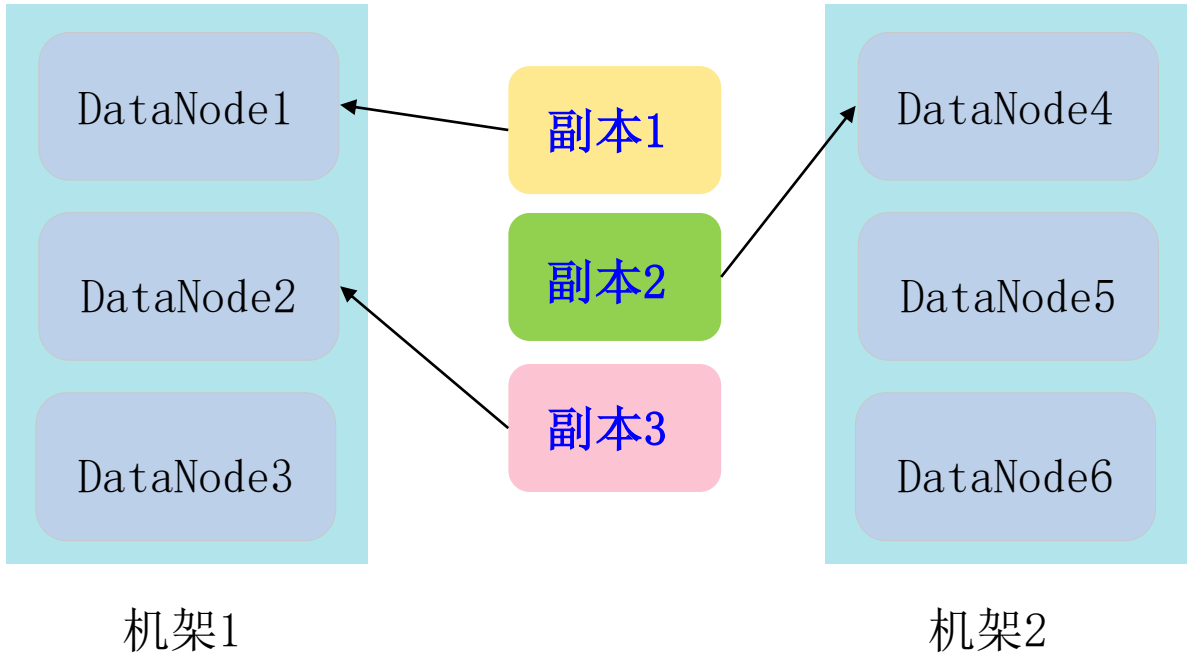
数据块的物理存储模型

- 分布式多副本冗余存储



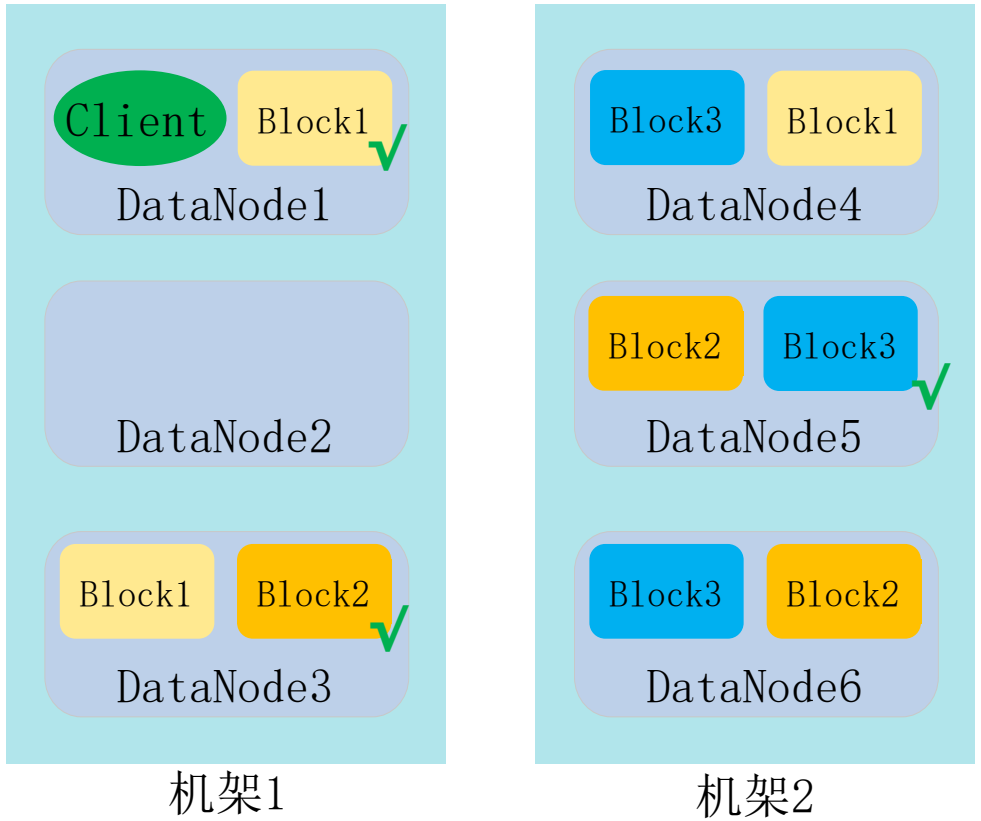
数据存放策略

- **第一个副本：**放置在上传文件的数据节点；如果是集群外提交，则随机挑选一台磁盘不太满、CPU不太忙的节点
- **第二个副本：**放置在与第一个副本不同的机架的节点上
- **第三个副本：**与第一个副本相同机架的其他节点上
- 更多副本：随机节点

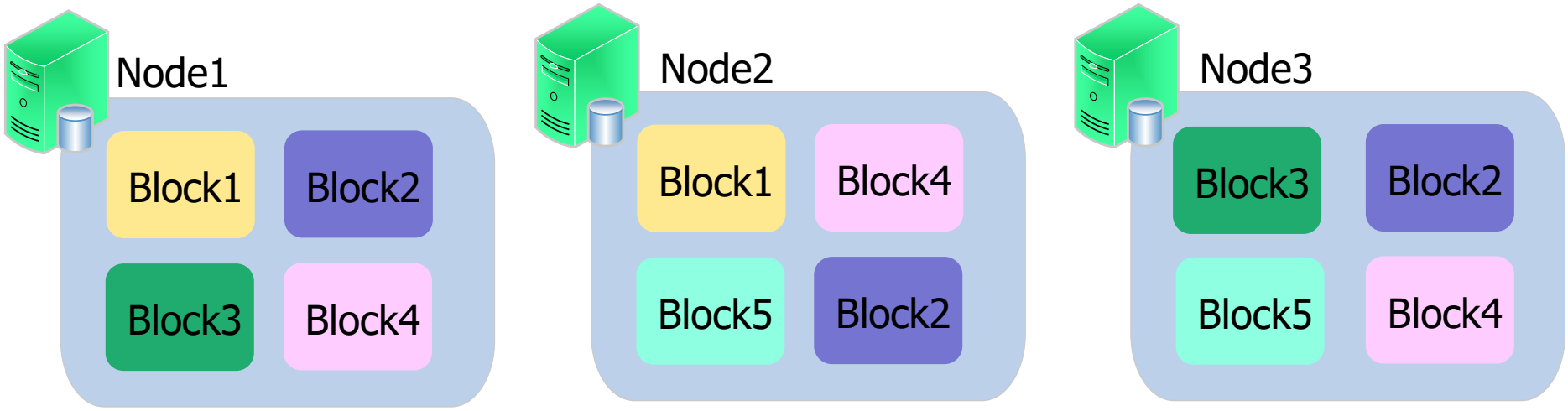


数据读取策略

- 机架ID获取：HDFS提供了一个API可以确定一个数据节点所属的机架ID，客户端也可以调用API获取自己所属的机架ID
- 副本读取策略：（1）优先选择同一节点；（2）次选同一机架；（3）随机读取



分布式文件块存储的优点

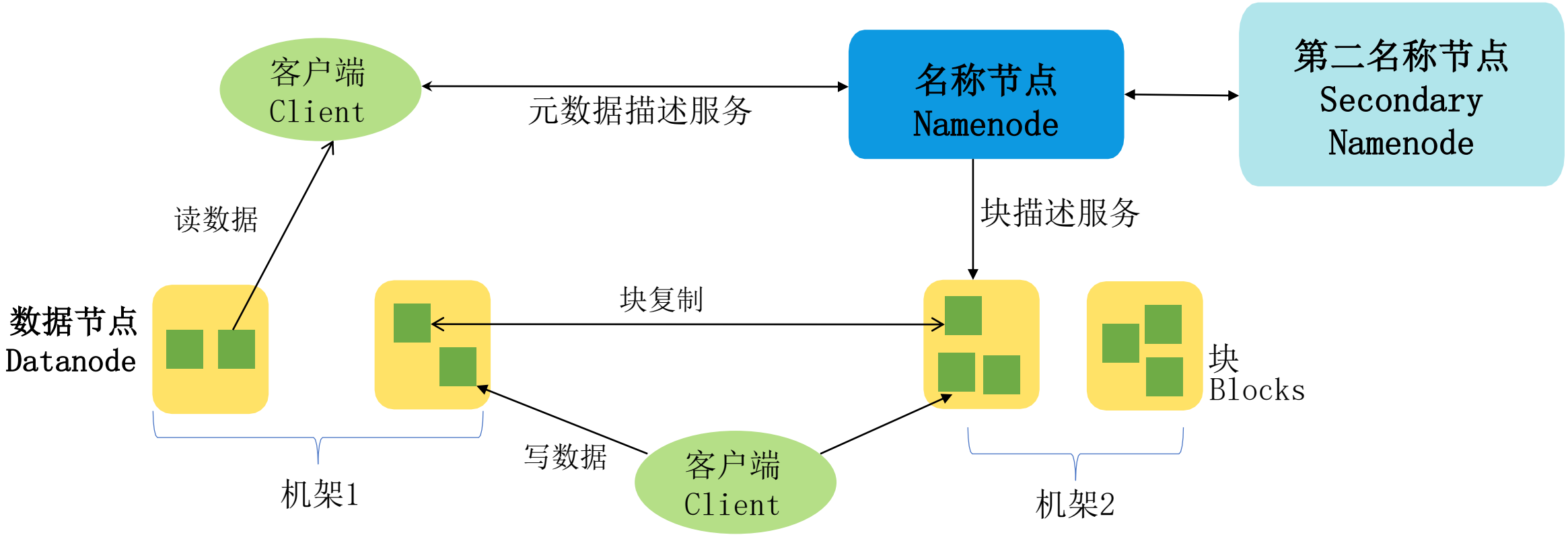


- 支持大文件存储
- 保证数据可靠性
- 加快数据存取速度：
  - ✓ 单个文件的分布式块存取
  - ✓ 多个文件分布式并行读取
- 简化了存储系统设计

深入理解HDFS的组件

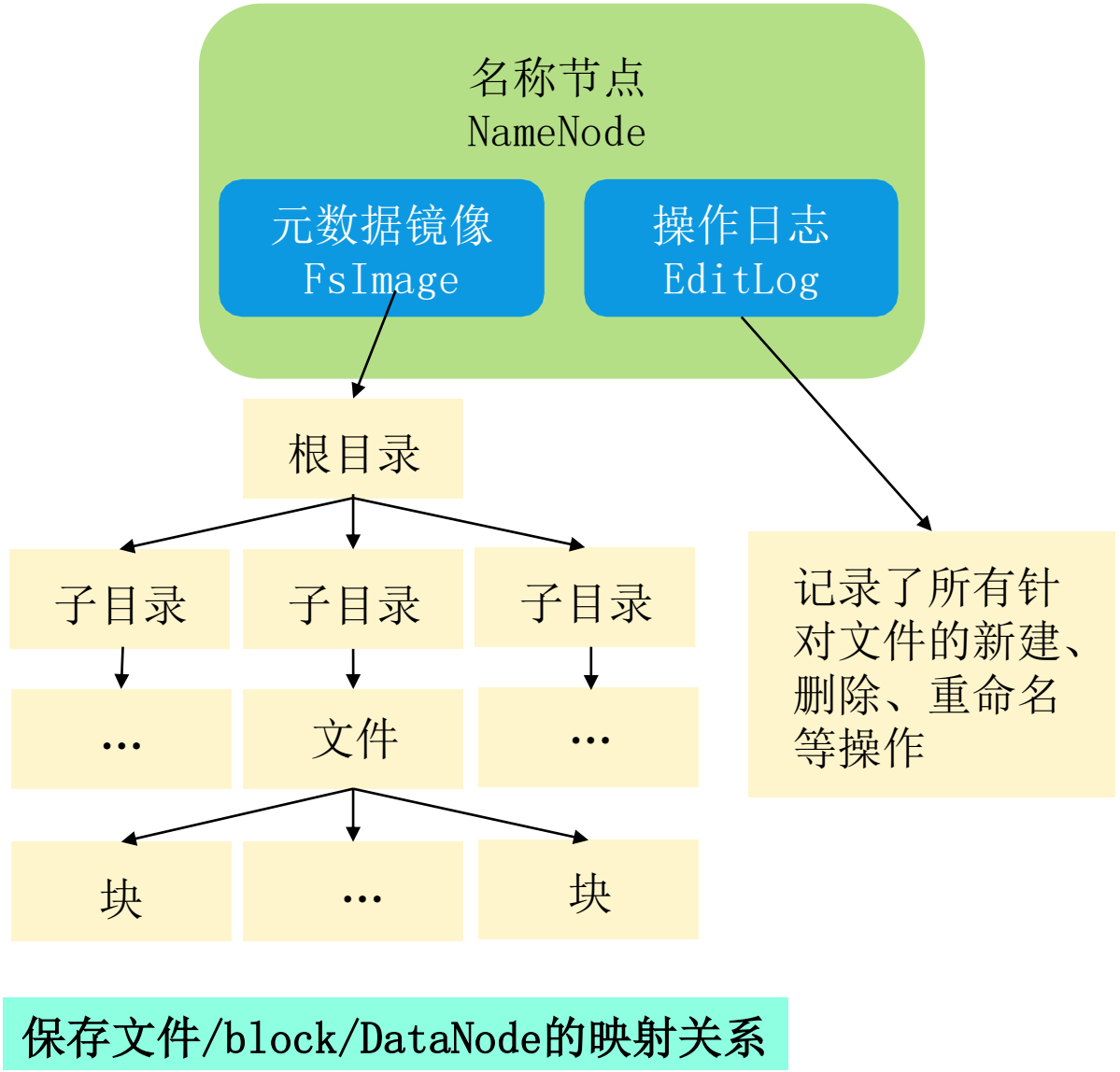
HDFS的基本组成

- HDFS采用了主从（Master/Slave）体系结构，易于实现和管理



名称节点（NameNode）

- 名称节点是分布式文件系统中的管理者
- 命名空间和元数据管理
- 核心数据结构FsImage和EditLog
  - ✓ FsImage：维护文件系统树及元数据
  - ✓ EditLog：操作日志文件记录

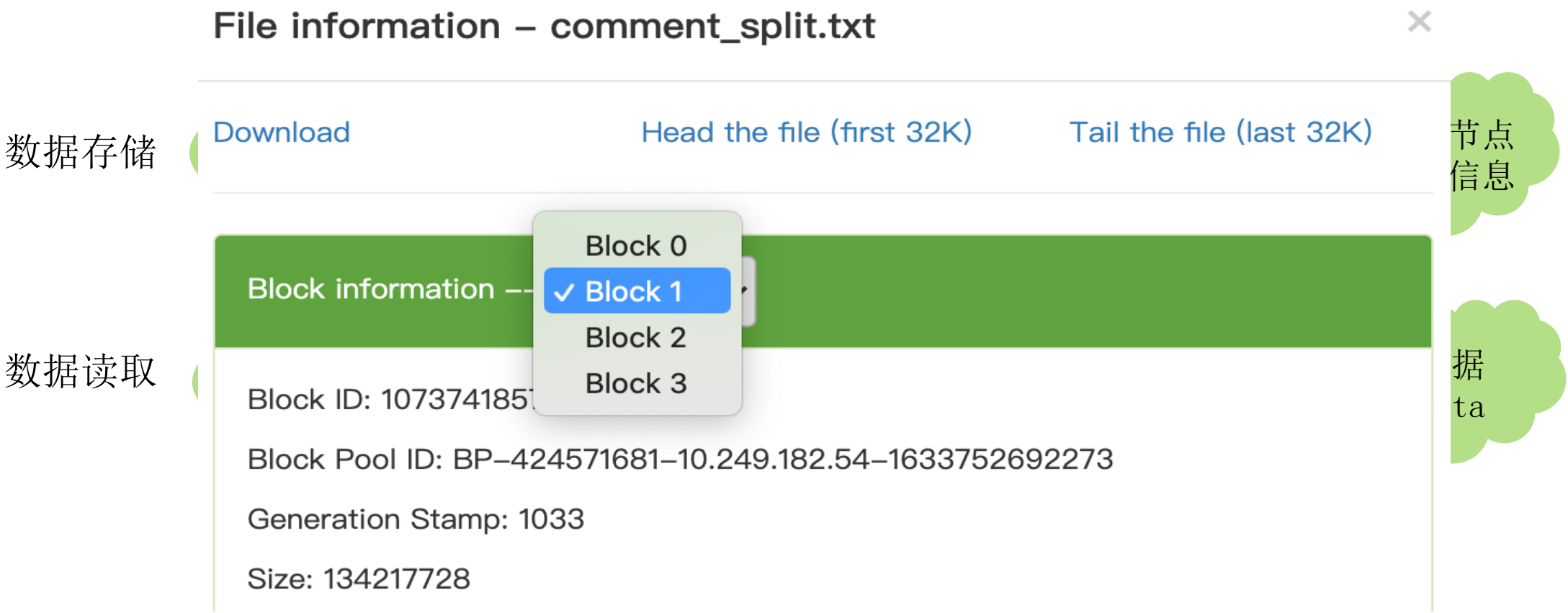


第二名称节点（Secondary NameNode）

- 用来保存名称节点中HDFS元数据的备份，并减少名称节点重启时间
- 一般是单独运行在一台机器上。

数据节点（Data Node）

- 数据节点负责数据的存储和读取
- “主动汇报”：定期向名称节点发送自己所存储的块的列表。
- 数据节点中的数据块会被保存在各自节点的本地Linux文件系统中



HDFS客户端

- HDFS客户端是用户操作HDFS最常用的方式，HDFS在部署时都提供了客户端
- HDFS客户端是一个库，暴露了HDFS文件系统接口，这些接口隐藏了HDFS实现中的大部分复杂性
- 客户端提供一个类似于POSIX（可移植操作系统界面）的文件系统接口，因此用户在编程时无需知道Namenode和Datanode也可实现其功能。

HDFS读写接口（Java）

- FileSystem是一个通用文件系统的抽象基类，可以被分布式文件系统继承，所有可能使用Hadoop文件系统的代码，都要使用这个类
- Hadoop为FileSystem这个抽象类提供了多种具体实现，DistributedFileSystem就是FileSystem在HDFS文件系统中的具体实现
- FileSystem的open()方法返回的是一个输入流FSDataInputStream对象，在HDFS文件系统中，具体的输入流就是DFSInputStream；FileSystem中的create()方法返回的是一个输出流FSDataOutputStream对象，在HDFS文件系统中，具体的输出流就是DFSOutputStream。

```
Configuration conf = new Configuration();
FileSystem fs = FileSystem.get(conf);
FSDataInputStream in = fs.open(new Path(uri));
FSDataOutputStream out = fs.create(new Path(uri));
```

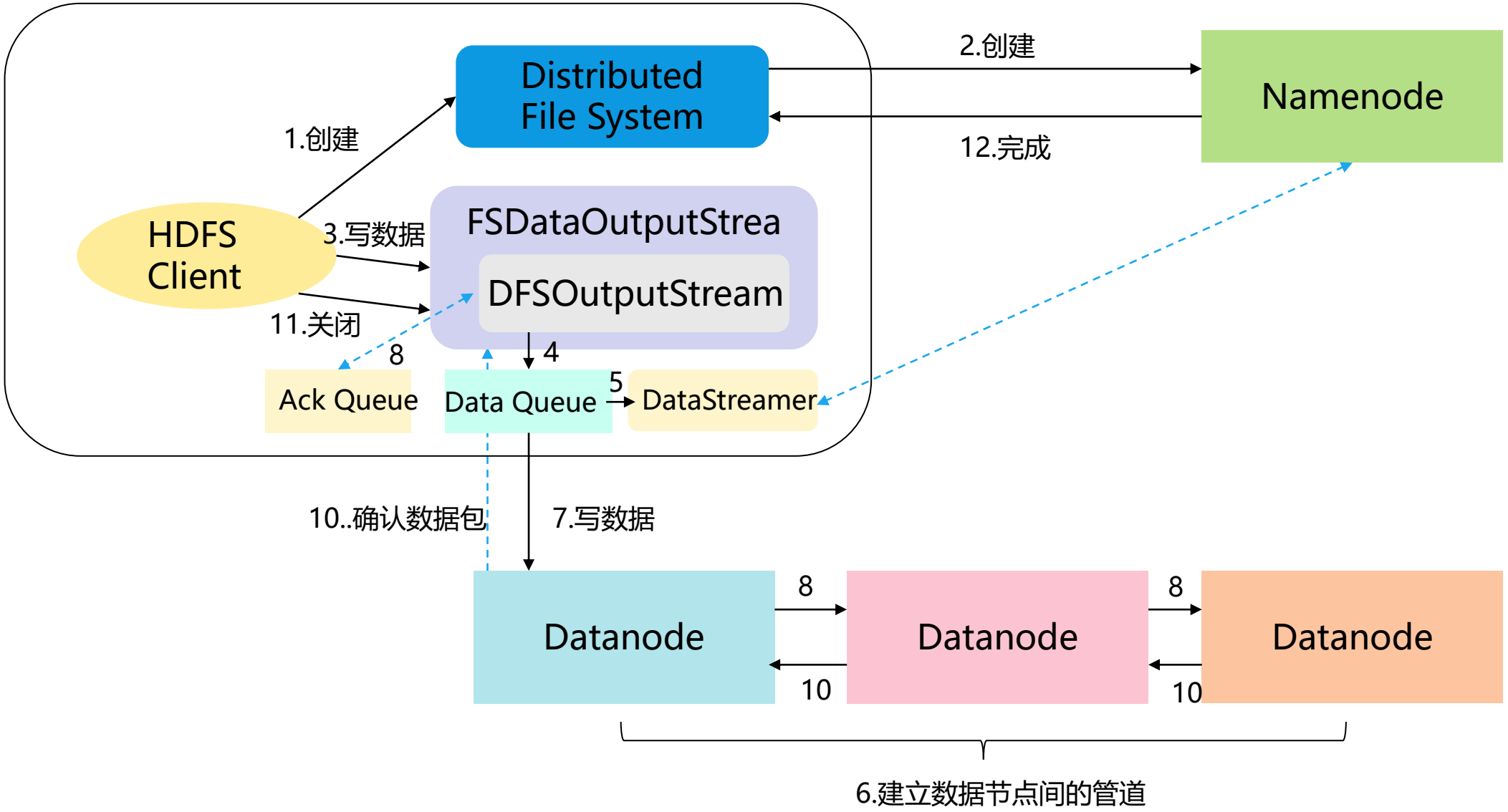
HDFS的读写流程与容错恢复机制

HDFS数据读写过程——写数据请求

- 首先，client请求NameNode，要将文件（可能需分为多个块）写入到HDFS。
- NameNode会给client赋予写权限，并为client提供可以写入数据的DataNode的IP地址。NameNode在选择可写入数据的DN的规则是：结合了DN的健康状态、复制因子、机架感知等因素
- 假如复制因子是3(默认值)，那么会为每个block返回三个IP地址。

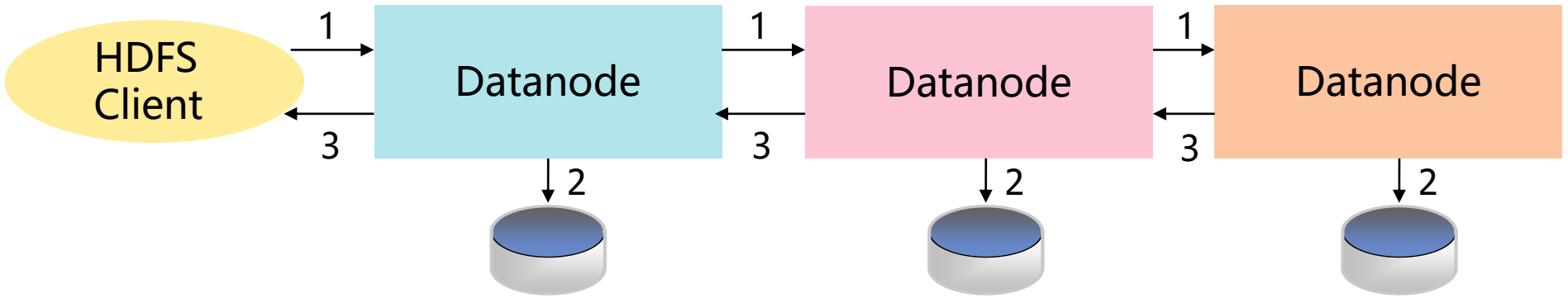
- 整体的数据复制流程分三个阶段:1.流水线建立；2.复制数据；3.关闭流水线

HDFS数据读写过程——写数据



写数据——流水线(pipeline)机制

- 创建流水线：Client通过连接各个块的ip列表来为每个块创建流水线
- 块写入：Client向流水线中写入块数据
- 关闭流水线：当数据块复制到所有DN后，按ip地址列表相反方向，依次反馈写入成功信息至Client，最终Client再反馈给NameNode以更新元数据信息



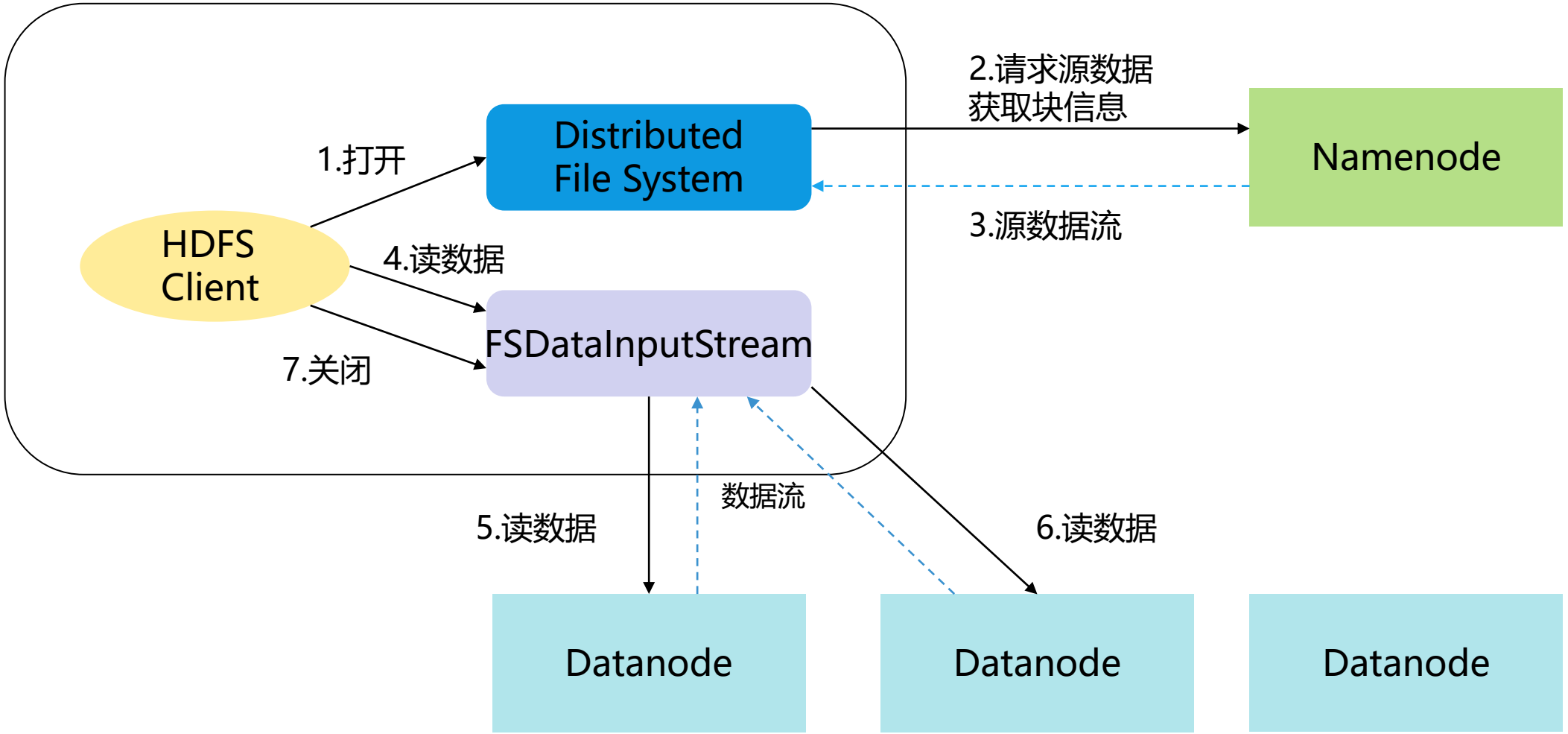
HDFS数据读写过程——写数据

```
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.FSDataOutputStream;
import org.apache.hadoop.fs.Path;
public class Chapter3 {
        public static void main(String[] args) {
                try {
                        Configuration conf = new Configuration();
                        conf.set("fs.defaultFS","hdfs://localhost:9000");
                        conf.set("fs.hdfs.impl",
                                "org.apache.hadoop.hdfs.
                                 DistributedFileSystem");
                        FileSystem fs = FileSystem.get(conf);
                        byte[] buff = "Hello world".getBytes(); // 要写入的内容
                        String filename = "test"; //要写入的文件名
                        FSDataOutputStream os = fs.create(new Path(filename));
                        os.write(buff,0,buff.length);
                        System.out.println("Create:"+ filename);
                        os.close();
                        fs.close();
                } catch (Exception e) {
                        e.printStackTrace();
                }
        }
}
```

HDFS数据读写过程——读数据请求

- Client请求NameNode要读取文件。
- NameNode根据自己的元数据信息，反馈给client一个DataNode的列表(包含所有的块)。
- Client连接DataNode，读取块数据，合并成文件

HDFS数据读写过程——读数据



HDFS数据读写过程——读数据

```
import java.io.BufferedReader;
import java.io.InputStreamReader;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.fs.FSDataInputStream;

public class Chapter3 {
        public static void main(String[] args) {
                try {
                        Configuration conf = new Configuration();
                        conf.set("fs.defaultFS","hdfs://localhost:9000");
                conf.set("fs.hdfs.impl","org.apache.hadoop.hdfs.DistributedFileSystem");
                        FileSystem fs = FileSystem.get(conf);
                        Path file = new Path("test");
                        FSDataInputStream getIt = fs.open(file);
                        BufferedReader d = new BufferedReader(new InputStreamReader(getIt));
                        String content = d.readLine(); //读取文件一行
                        System.out.println(content);
                        d.close(); //关闭文件
                        fs.close(); //关闭hdfs
                } catch (Exception e) {
                        e.printStackTrace();
                }
        }
}
}
```

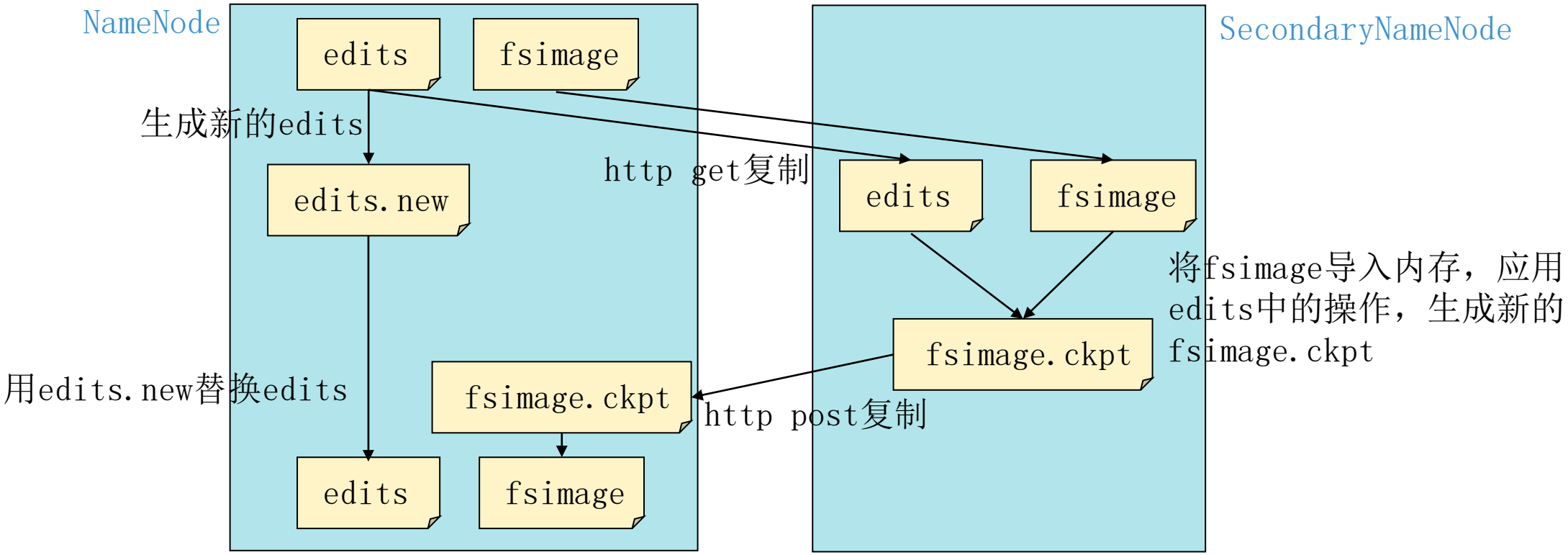
HDFS通信协议

- HDFS是一个部署在集群上的分布式文件系统，因此很多数据需通过网络进行传输
- 所有的HDFS通信协议都是构建在TCP/IP协议基础之上的
- 客户端通过一个可配置的端口向名称节点主动发起TCP连接，并使用客户端协议与名称节点进行交互
- 名称节点和数据节点之间则使用数据节点协议进行交互
- 客户端与数据节点的交互是通过RPC（Remote Procedure Call）来实现的。在设计上，名称节点不会主动发起RPC，而是响应来自客户端和数据节点的RPC请求。

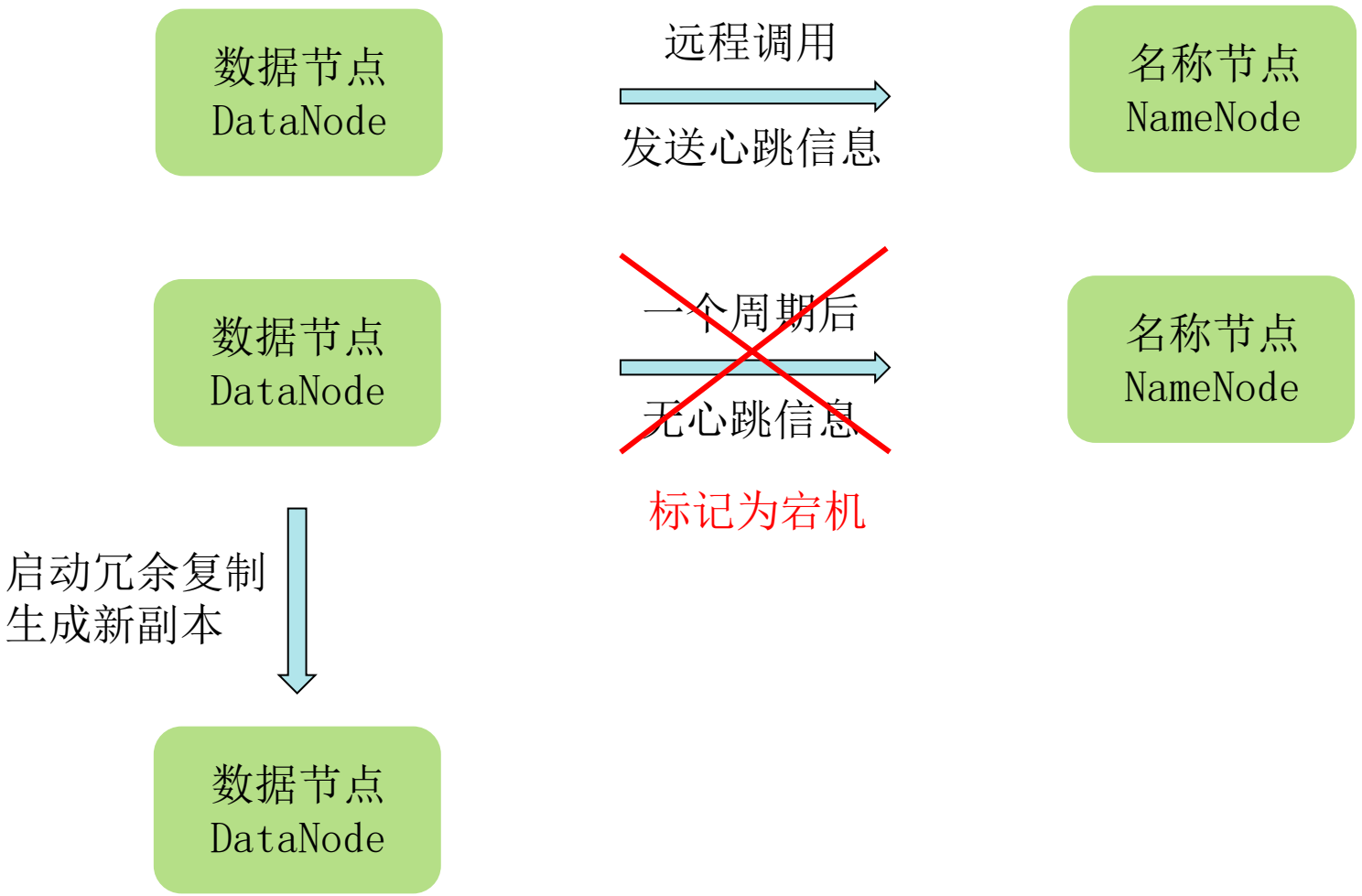
HDFS的容错与恢复机制

- HDFS的容错假设：“硬件出错看作一种常态，而不是异常”
- 以“检测节点和数据错误并进行自动恢复”为主要目标
- 主要包括以下几种情形：
  - 名称节点出错
  - 数据节点出错
  - 数据出错

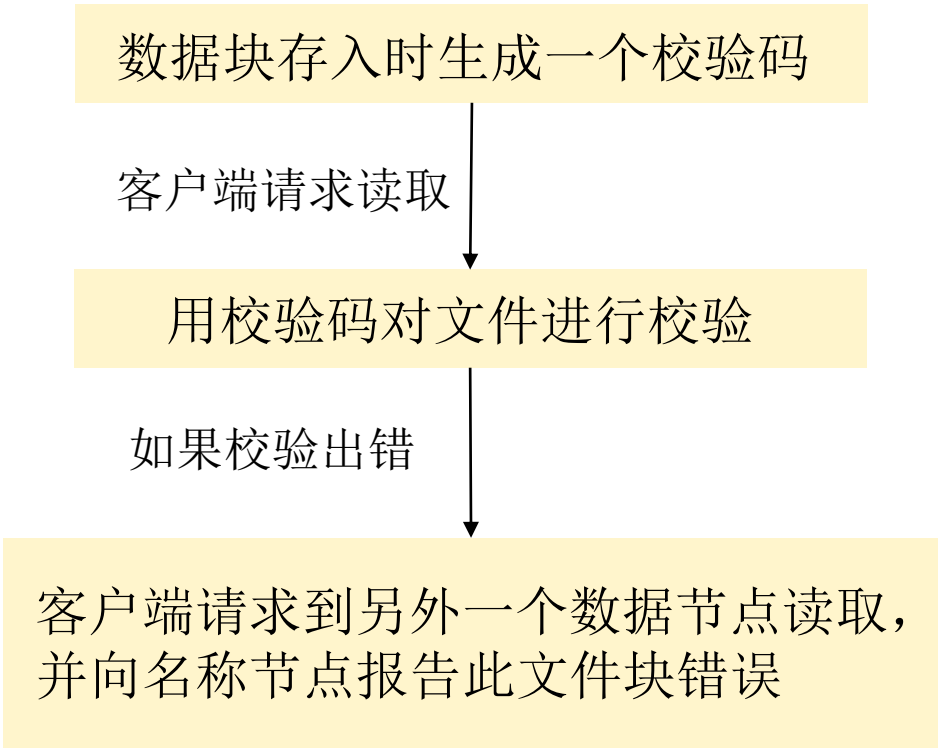
名称节点错误与恢复方法



数据节点错误与恢复方法

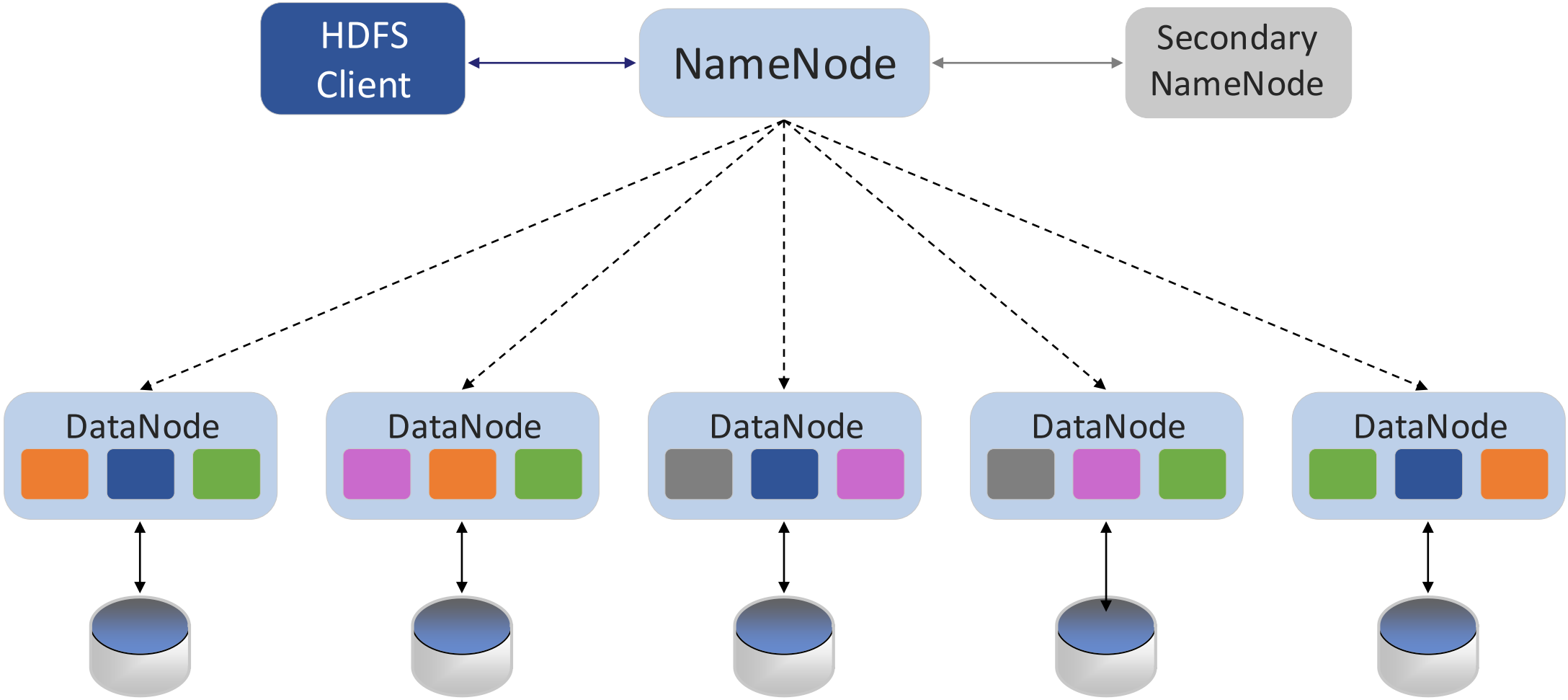


数据校验错误与恢复方法



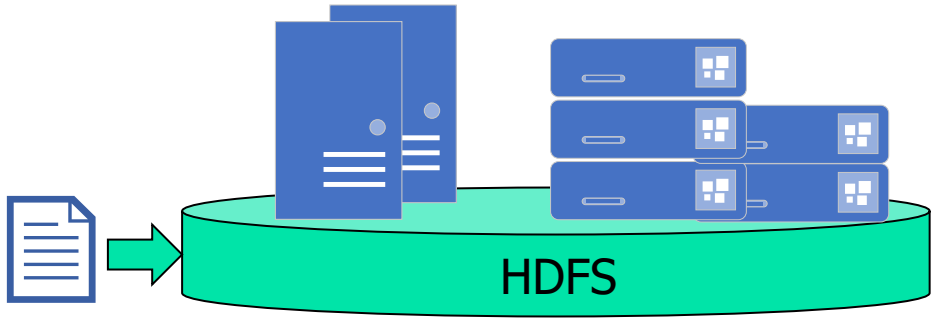
总结与知识拓展

HDFS的总体架构


HDFS
Client
NameNode
Secondary
NameNode
DataNode
DataNode
DataNode
DataNode
DataNode

HDFS基本原理

- 基本原理：
  - 将文件切分成等大的数据块，分别存储到多台机器上。
  - 每个数据块存在多个备份。
  - 将数据切分、容错、负载均衡等功能透明化。
  - 可将HDFS看成是一个巨大、具有容错性的磁盘。



HDFS特点

- 总体而言，HDFS实现了：
  - 兼容廉价的硬件设备
  - 流数据读写
  - 大数据集的存储访问
  - 简单的文件模型
  - 强大的跨平台兼容性
- HDFS特殊的设计，在实现上述优良特性的同时，也使得自身具有一些应用局限性
  - 不适合低延迟数据访问
  - 无法高效存储大量小文件
  - 不支持多用户写入及任意修改文件

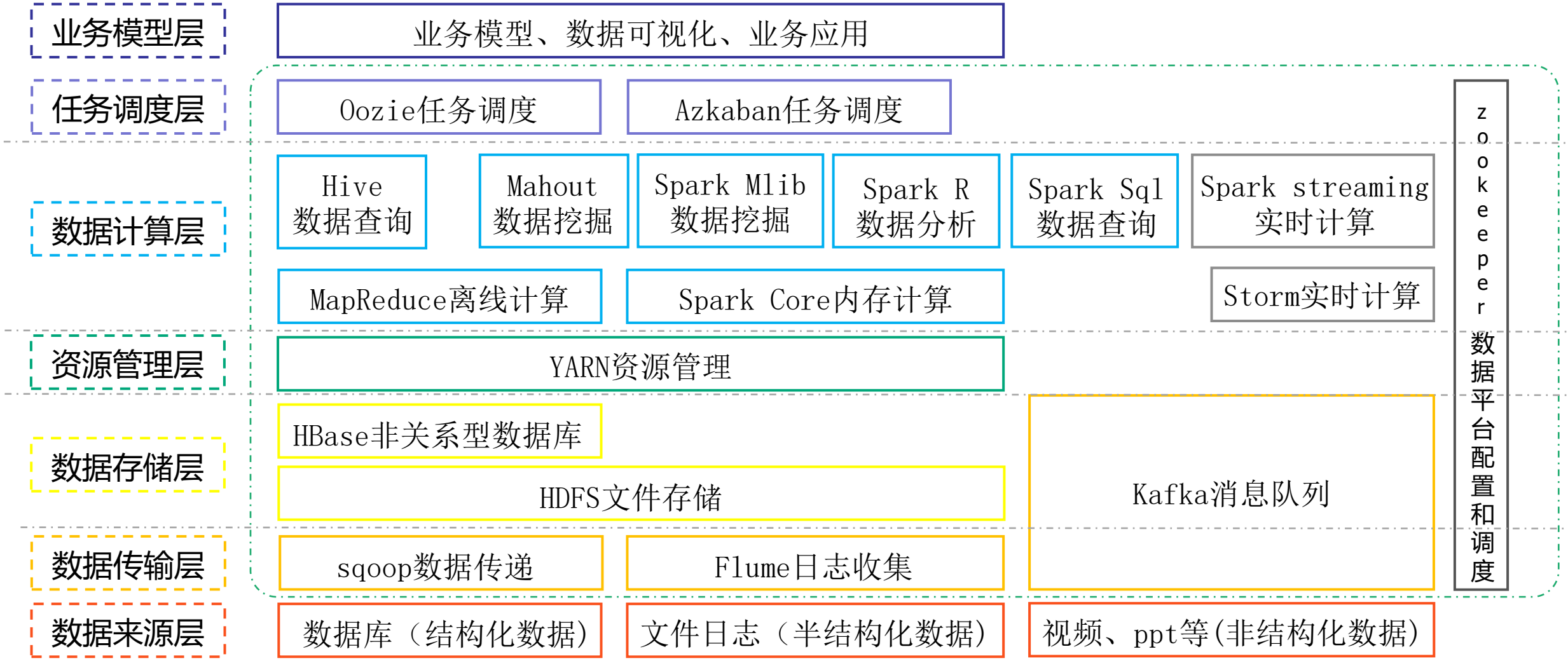
HDFS体系结构的局限性

- HDFS只设置唯一一个名称节点，这样做虽然大大简化了系统设计，但也带来了一些明显的局限性，具体如下：
  - 命名空间的限制：名称节点是保存在内存中的，因此，名称节点能够容纳的对象（文件、块）的个数会受到内存空间大小的限制。
  - 性能的瓶颈：整个分布式文件系统的吞吐量，受限于单个名称节点的吞吐量。
  - 隔离问题：由于集群中只有一个名称节点，只有一个命名空间，因此，无法对不同应用程序进行隔离。
  - 集群的可用性：一旦这个唯一的名称节点发生故障，会导致整个集群变得不可用

其它常用的分布式存储系统

- GFS (Google File System): Google公司为了满足本公司需求而开发的基于Linux的专有分布式文件系统
- Lustre:大规模的、安全可靠的，具备高可用性的集群文件系统，它是由SUN公司开发和维护的
- TFS (Taobao File System):面向互联网服务的分布式文件系统，主要针对海量的非结构化数据，为淘宝提供海量小文件存储
- Ceph、MooseFS、GlusterFS、GridFS等
- Google Colossus FS/Facebook Tectonics FS/SeaweedFS

Hadoop生态

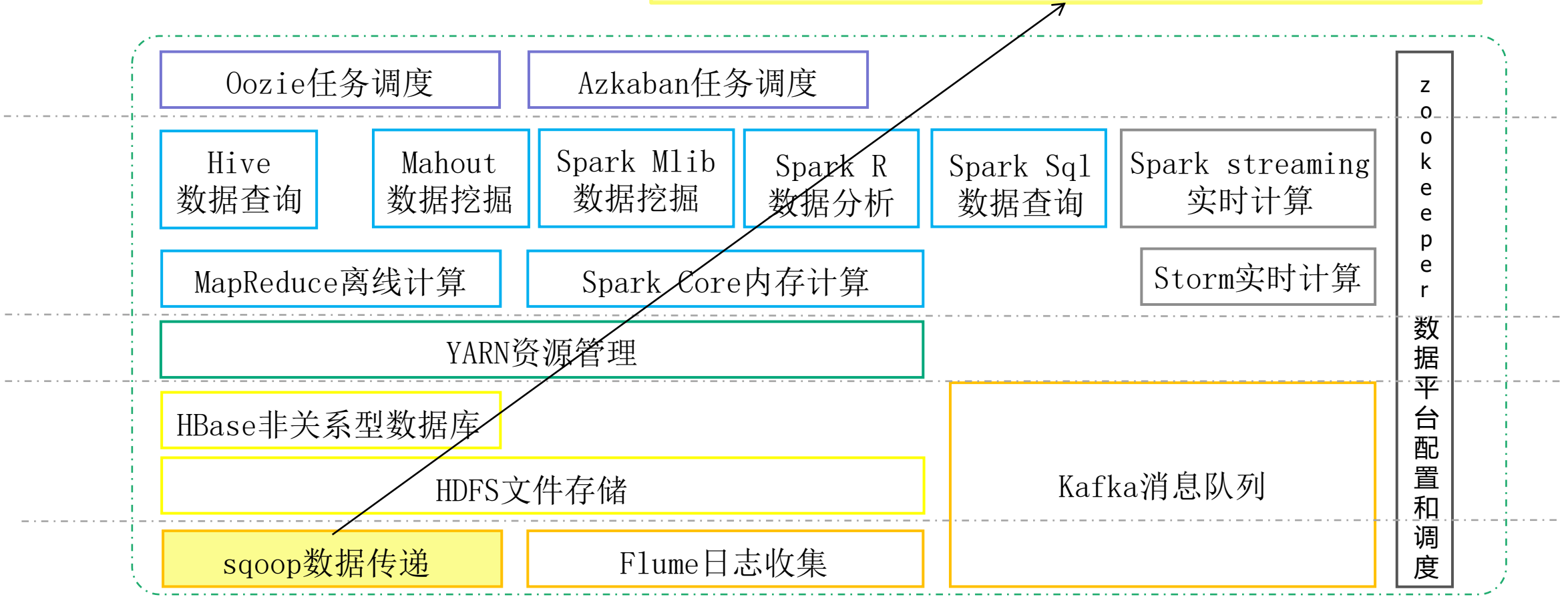

业务模型层
业务模型、数据可视化、业务应用
任务调度层
Oozie任务调度
Azkaban任务调度
数据计算层
Hive
数据查询
Mahout
数据挖掘
Spark Mlib
数据挖掘
Spark R
数据分析
Spark Sql
数据查询
Spark streaming
实时计算
MapReduce离线计算
Spark Core内存计算
Storm实时计算
资源管理层
YARN资源管理
数据存储层
HBase非关系型数据库
HDFS文件存储
Kafka消息队列
数据传输层
sqoop数据传递
Flume日志收集
数据来源层
数据库（结构化数据）
文件日志（半结构化数据）
视频、ppt等(非结构化数据)
zookeeper
数据平台配置和调度

Hadoop组件功能

组件	功能
HDFS	分布式文件系统
MapReduce	分布式编程模型
YARN	资源管理框架
Zookeeper	可扩展协调系统
AVRO	序列化框架
Hbase	分布式数据库
Hive	数据仓库
Pig	脚本语言
Sqoop	同步处理工具

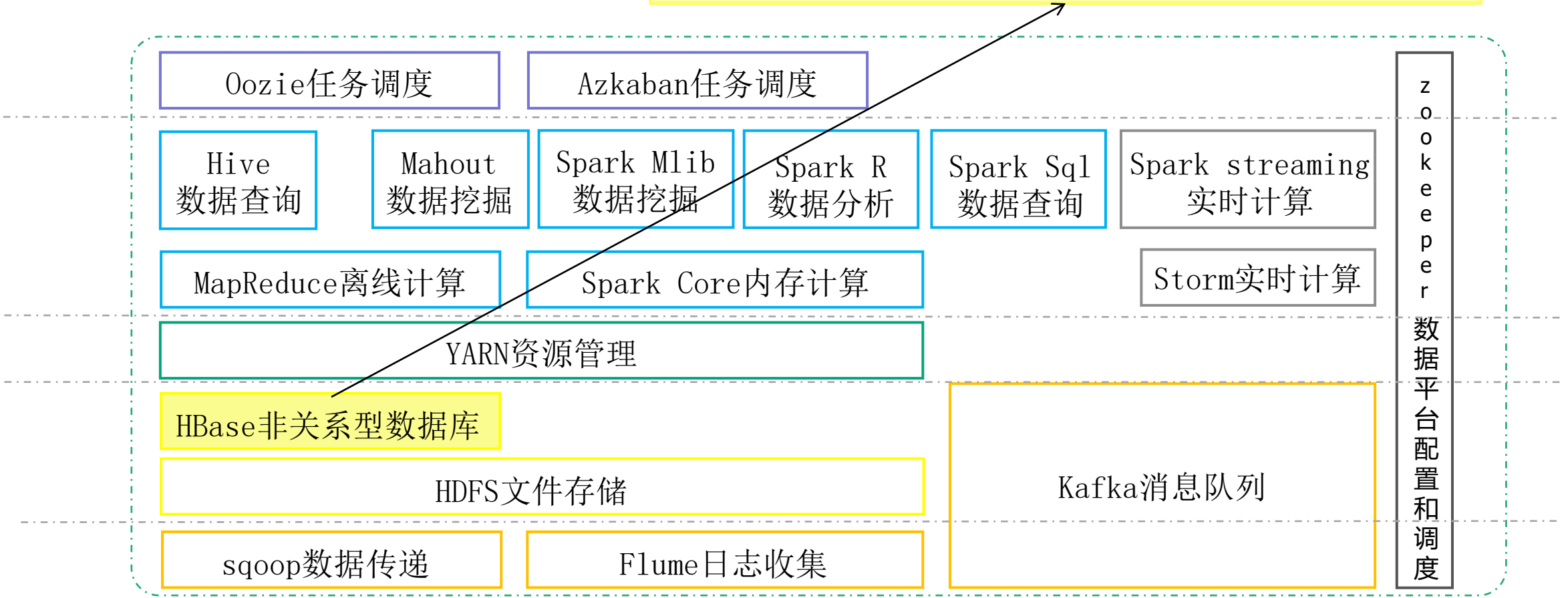
Hadoop组件功能

用于在Hadoop与传统的数据库间进行数据的传递，可以将一个关系型数据库（如： MySQL ,Oracle等）中的数据导进到Hadoop的HDFS中，也可以将HDFS的数据导进到关系型数据库



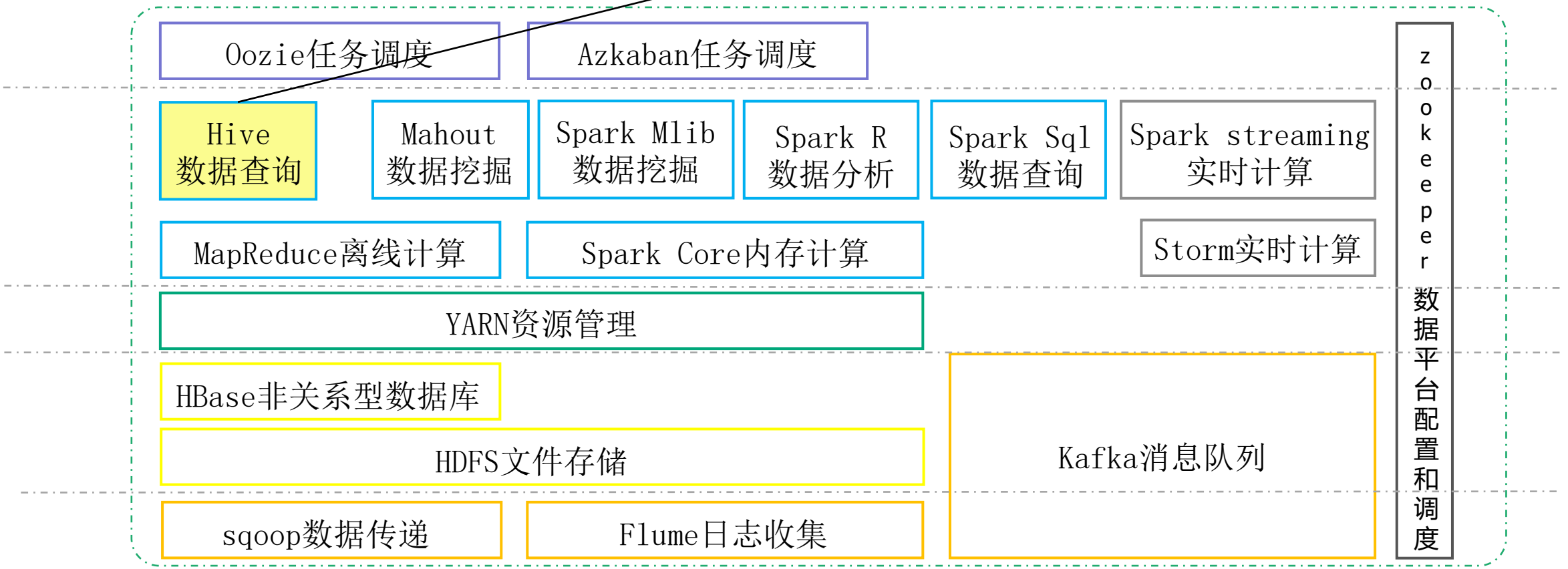
Hadoop组件功能

提供海量数据存储功能，是一种构建在HDFS之上的分布式、面向列的存储系统
对于半结构化或非结构化的数据很适合使用Hbase，因为Hbase支持动态添加列



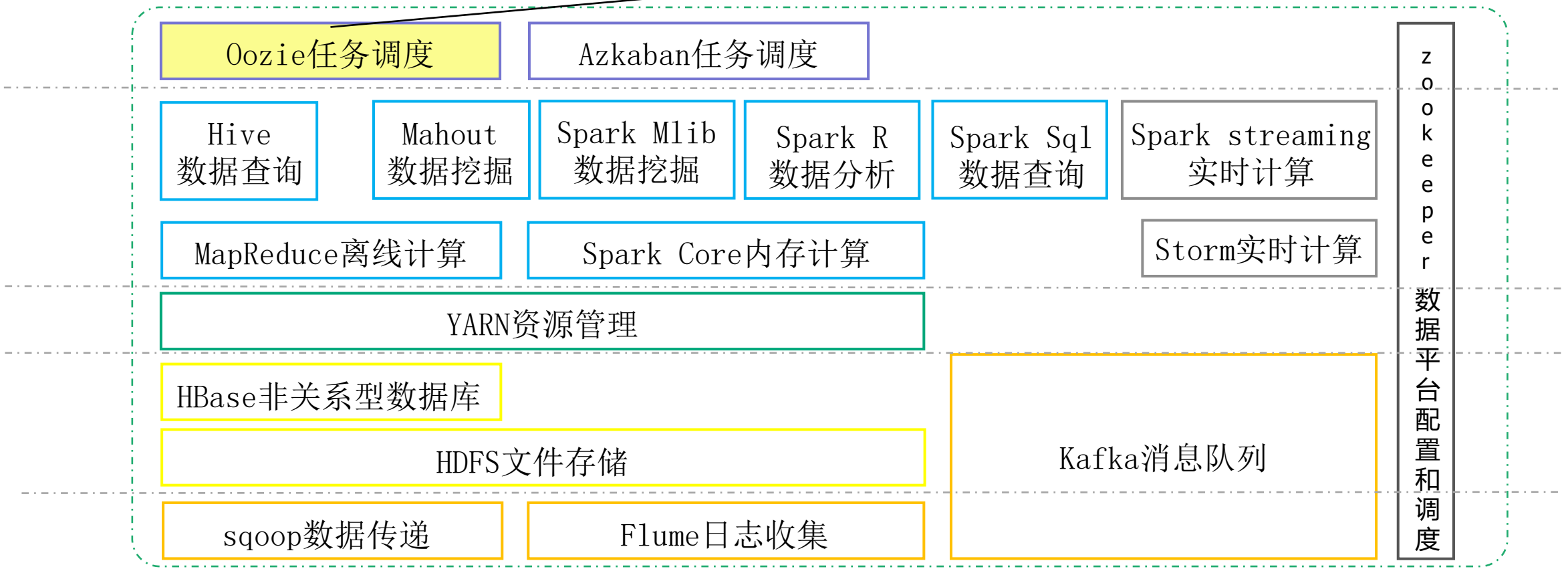
Hadoop组件功能

是基于Hadoop的一个数据仓库工具，可以将结构化的数据文件映射为一张数据库表并提供类sql查询功能



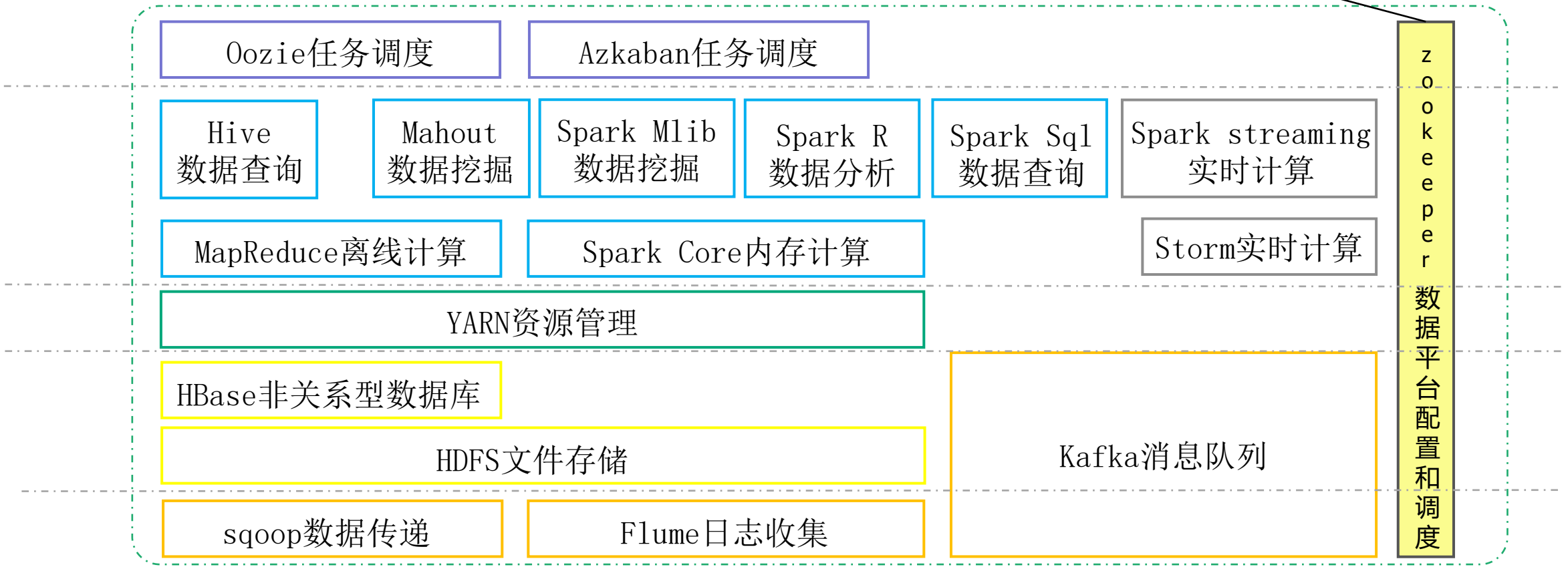
Hadoop组件功能

Oozie是一个用于管理Apache Hadoop作业的工作流调度程序系统，支持多种类型的Hadoop作业（例如Java map-reduce，Streaming map-reduce，Hive，Sqoop和Spark）以及系统特定的工作（例如Java程序和shell脚本）。



Hadoop组件功能

分布应用程序协调系统，为分布式应用提供一致性服务，如配置服务、名字服务等，有助于系统避免单点故障



Hadoop的典型应用

- 百度
  - 百度在2012年其总的集群规模达到近十个，单集群超过2800台机器节点，Hadoop机器总数有上万台机器，总的存储容量超过100PB，已经使用的超过74PB，每天提交的作业数目有数千个之多，每天的输入数据量已经超过7500TB，输出超过1700TB。
  - 百度的Hadoop集群主要应用包括
    - 数据挖掘与分析、日志分析平台、数据仓库系统、推荐引擎系统、用户行为分析系统

Hadoop的典型应用

- 阿里巴巴
  - 阿里巴巴的Hadoop集群截至2012年大约有3200台服务器，大约300000物理CPU核心，总内存100TB，总的存储容量超过60PB，每天的作业数目超过150000个，每天hive query查询大于6000个，每天扫描数据量约为7.5PB，每天扫描文件数约为4亿，存储利用率大约为80%，CPU利用率平均为65%，峰值可以达到80%。阿里巴巴的Hadoop集群拥有150个用户组、4500个集群用户，为淘宝、天猫、一淘、聚划算、CBU、支付宝提供底层的基础计算和存储服务
  - 阿里巴巴的Hadoop集群主要应用包括
    - 搜索支撑、广告系统、数据平台系统、量子统计、淘数据、推荐引擎系统、搜索排行榜

Hadoop的典型应用

- 腾讯
  - 腾讯也是使用Hadoop最早的中国互联网公司之一，截至2012年年底，腾讯的Hadoop集群机器总量超过5000台，最大单集群约为2000个节点，并利用Hadoop-Hive构建了自己的数据仓库系统TDW，同时还开发了自己的TDW-IDE基础开发环境。腾讯的Hadoop为腾讯各个产品线提供基础云计算和云存储服务
  - 腾讯的Hadoop服务于其下各种产品
    - 腾讯社交广告平台、QQ、QQ音乐等