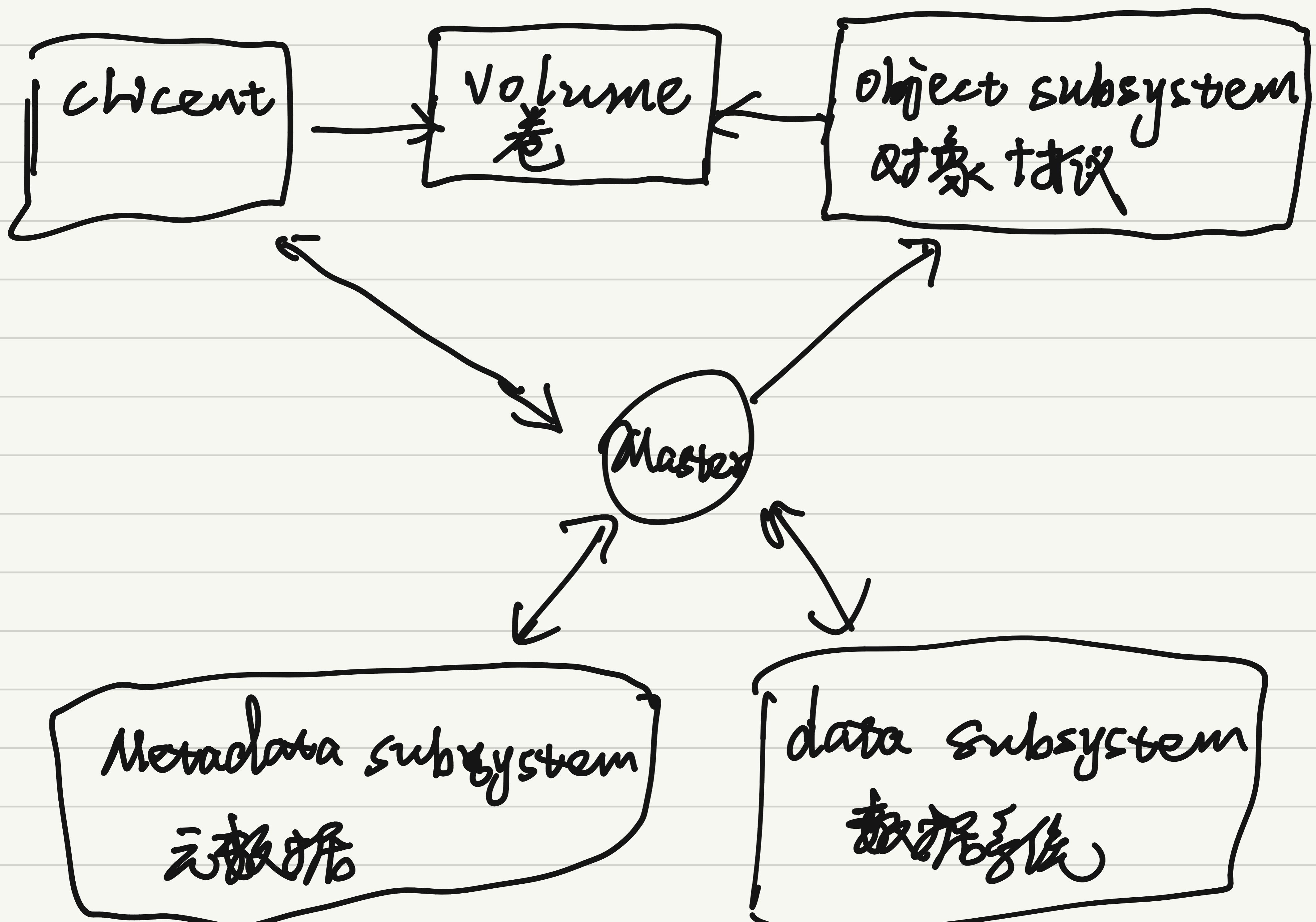




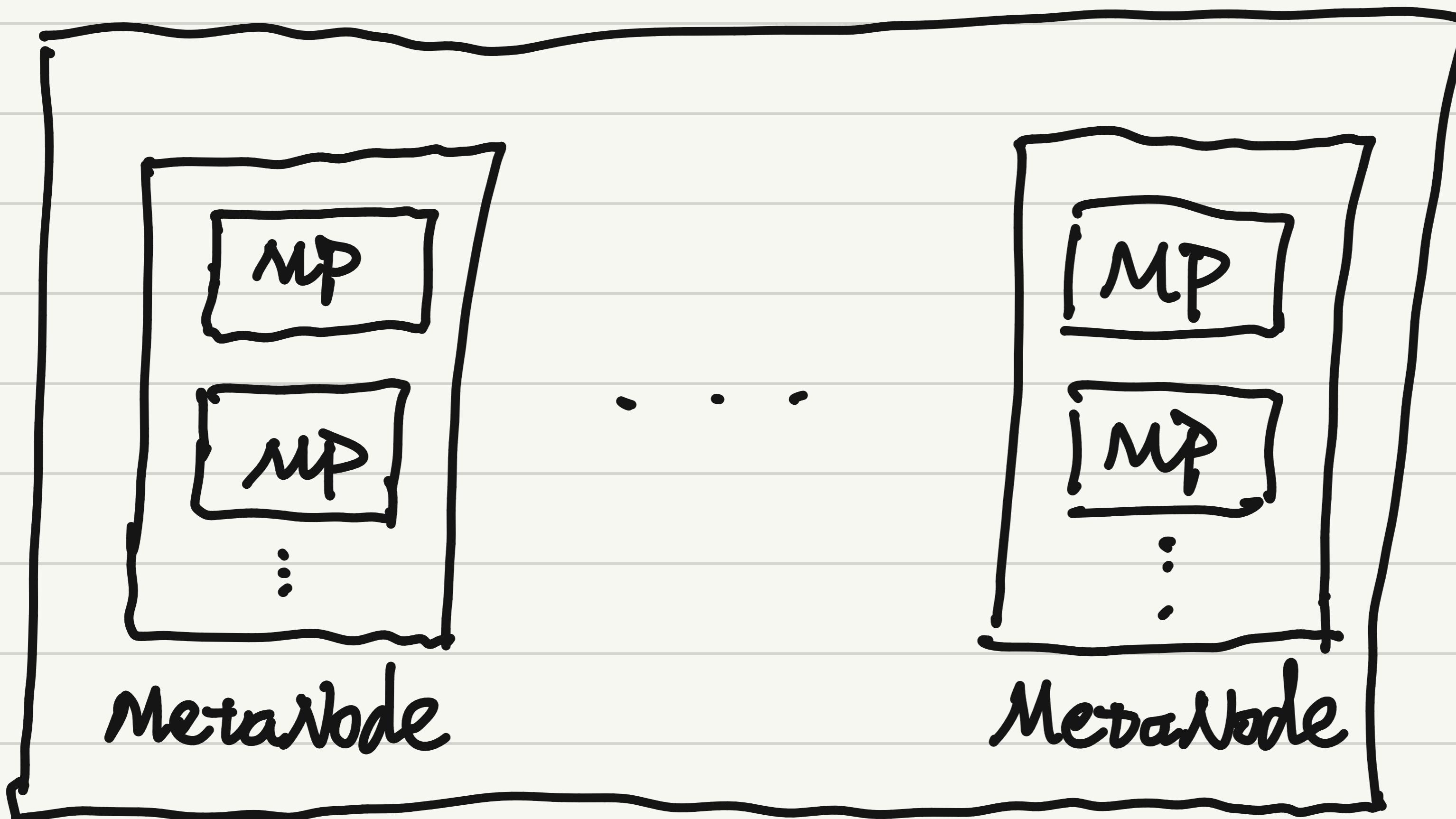
raft

rocksdb.

# 架构



# Metadata subsystem

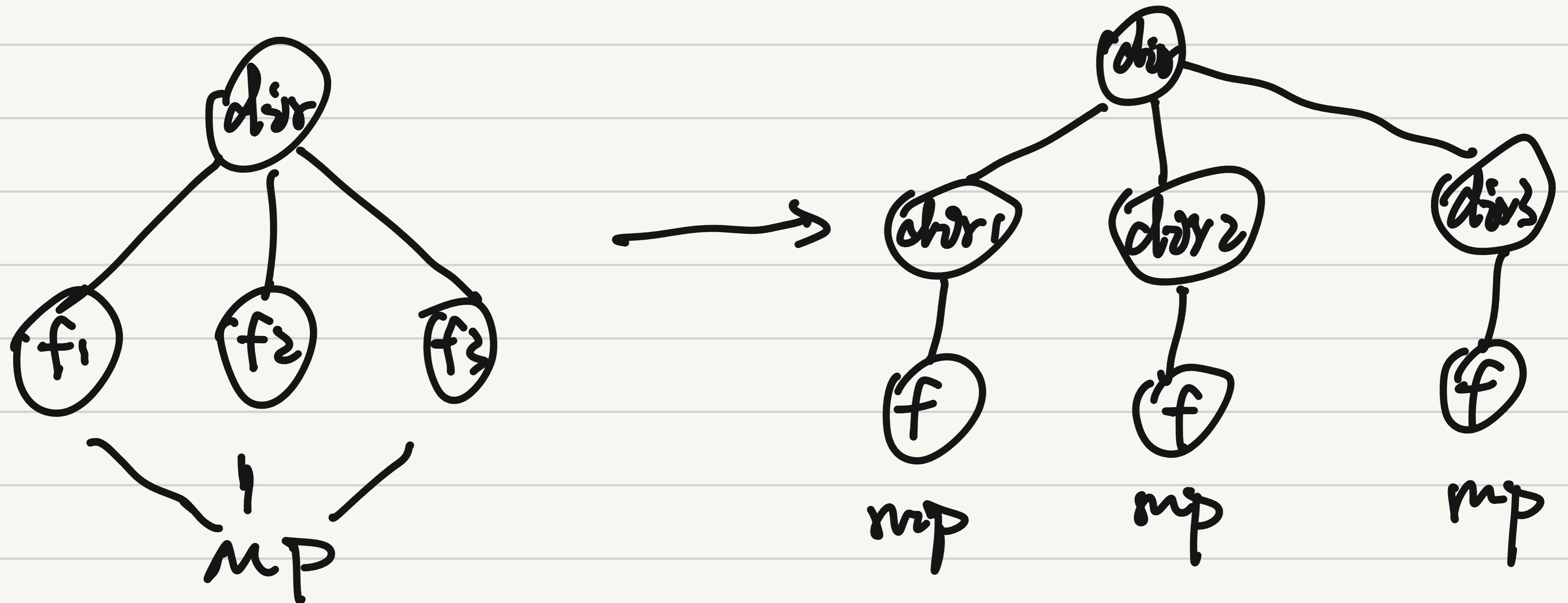


元数据分布：inode 和 dentry (文件目录索引节点)

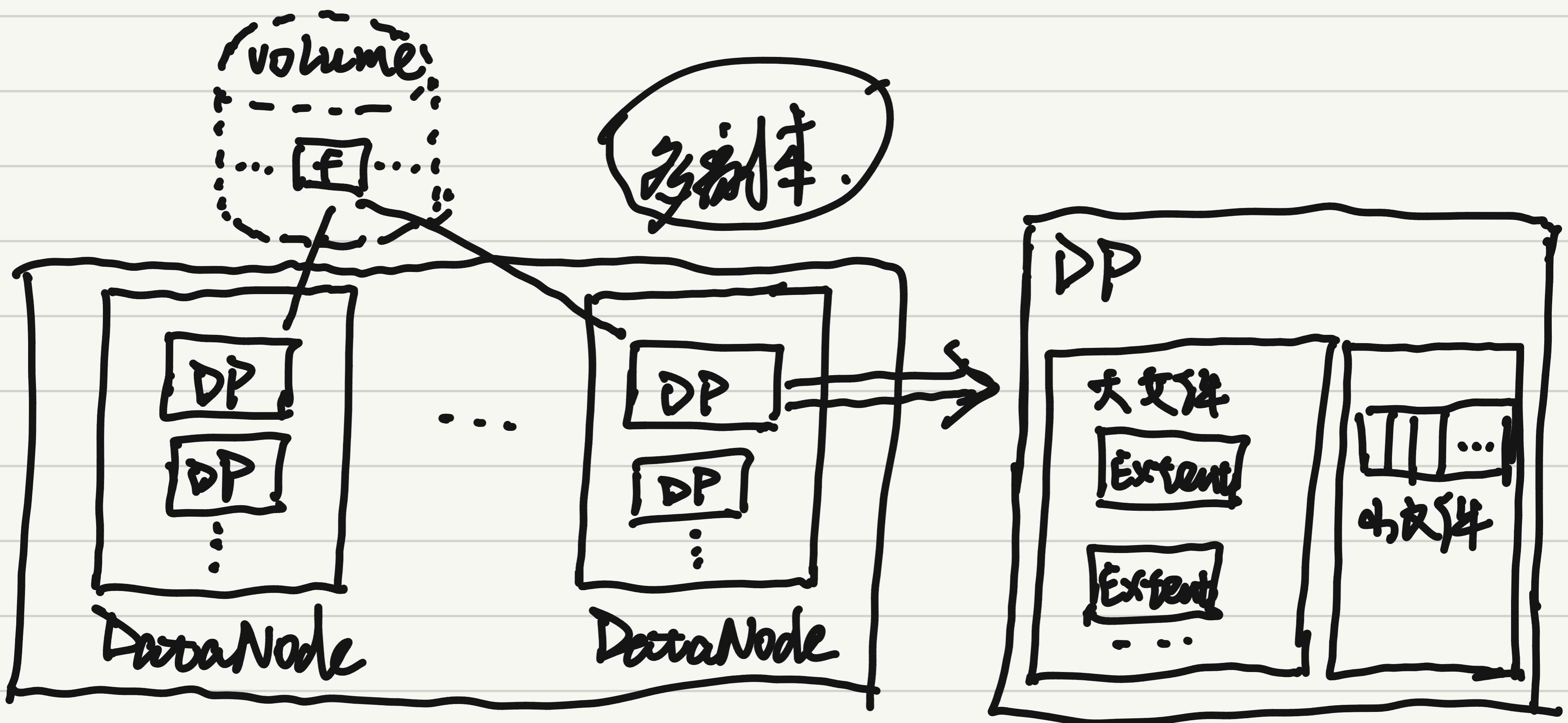
MP 管理一定范围 inode

子结点 dentry 存在父目录的 MP 分区中

热点问题：虚拟目录，手动操作热点目录



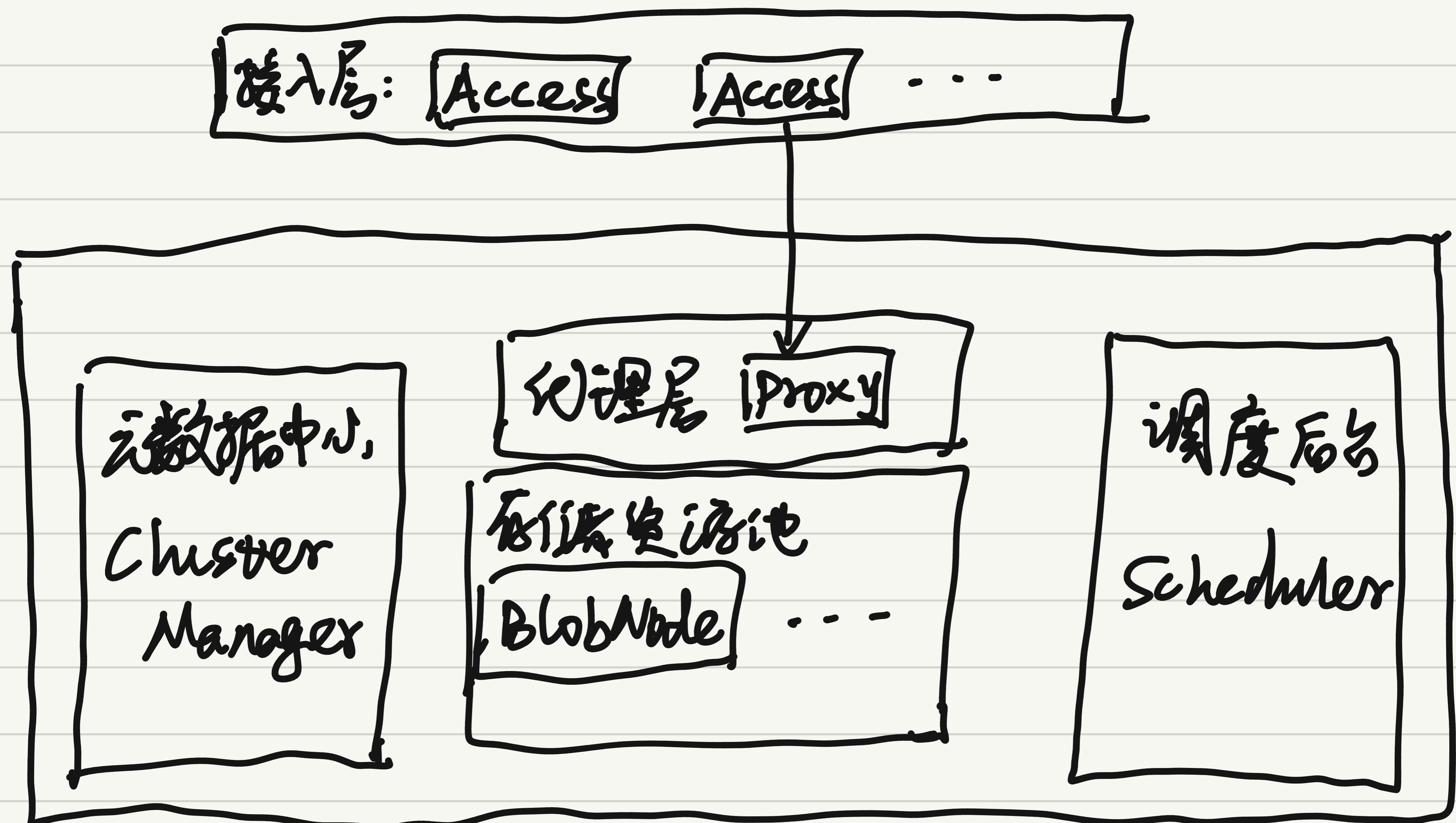
# DataNode subsystem



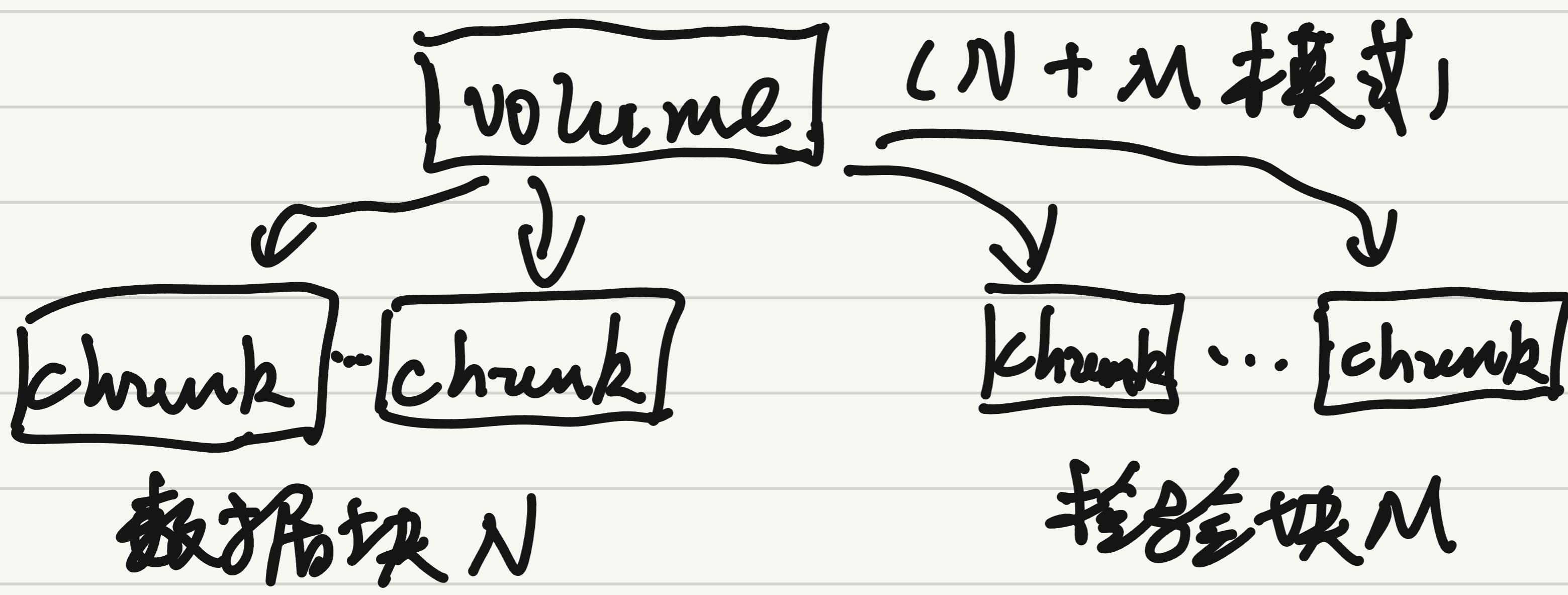
大文件通过一个一个Extent组成 (新文件以Extent的0偏移)  
小文件一个个聚合组成一个Extent (需记录每个小文件的偏移  
(小文件通过文件穿洞接口fallocate实现分配))  
数据复制: { 按顺序写入时: 按主备复制方式, 先复制主设备 }  
随机写入时: 采用Multi-Raft复制协议

磁盘会被组织在一个一个DP上.

# 分布式存储系统架构.



集群分配空间  $= \text{volume}_1 + \text{volume}_2 + \dots$

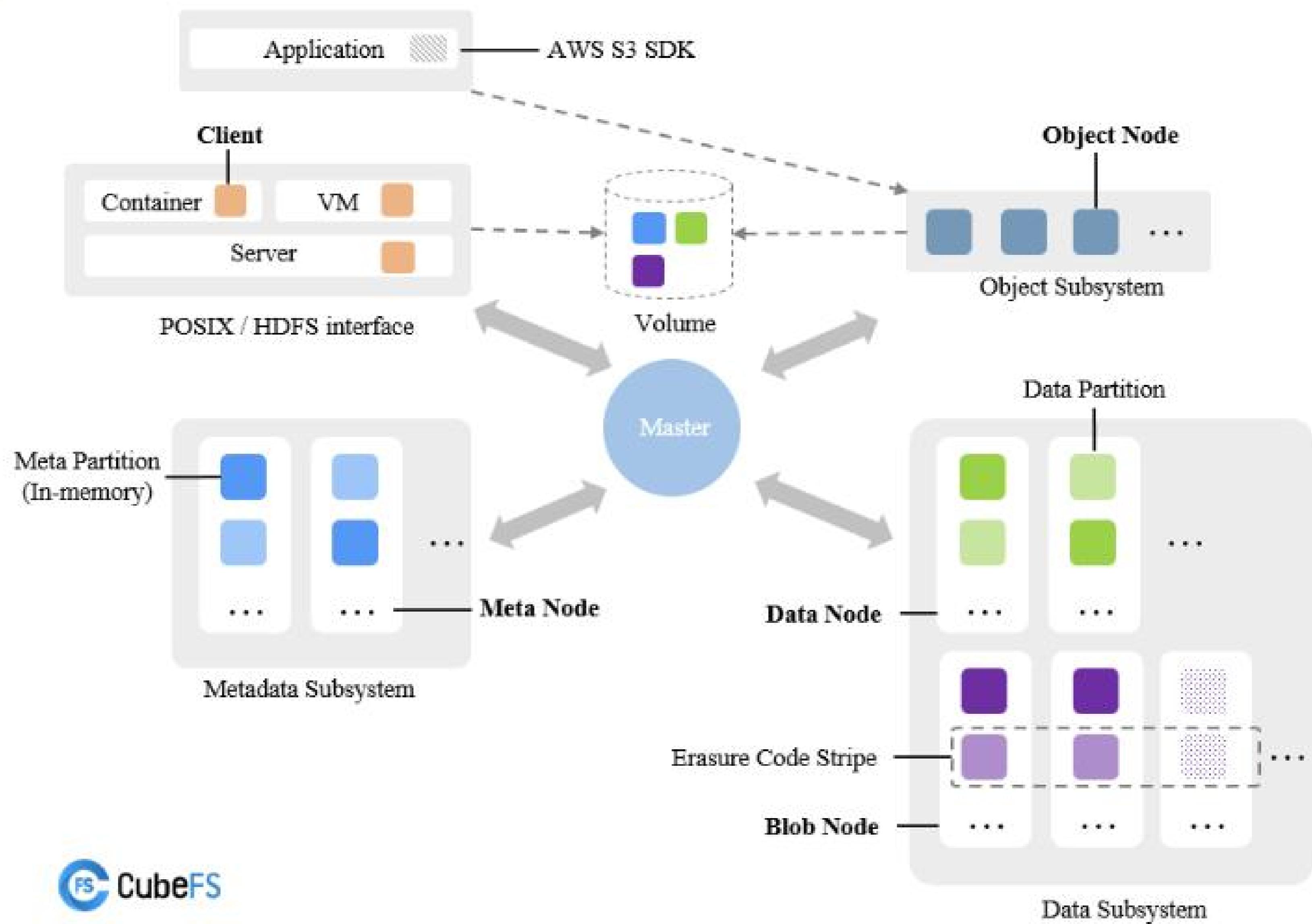


用户数据拆分成  $N$  个 Blob 块，Blob 块按照  $N+M$  模式  
拆分成  $N$  个 share 块，并输出出  $M$  个 share 块  
子块存到一个可用 volume 的 chunk 中

# CubeFS

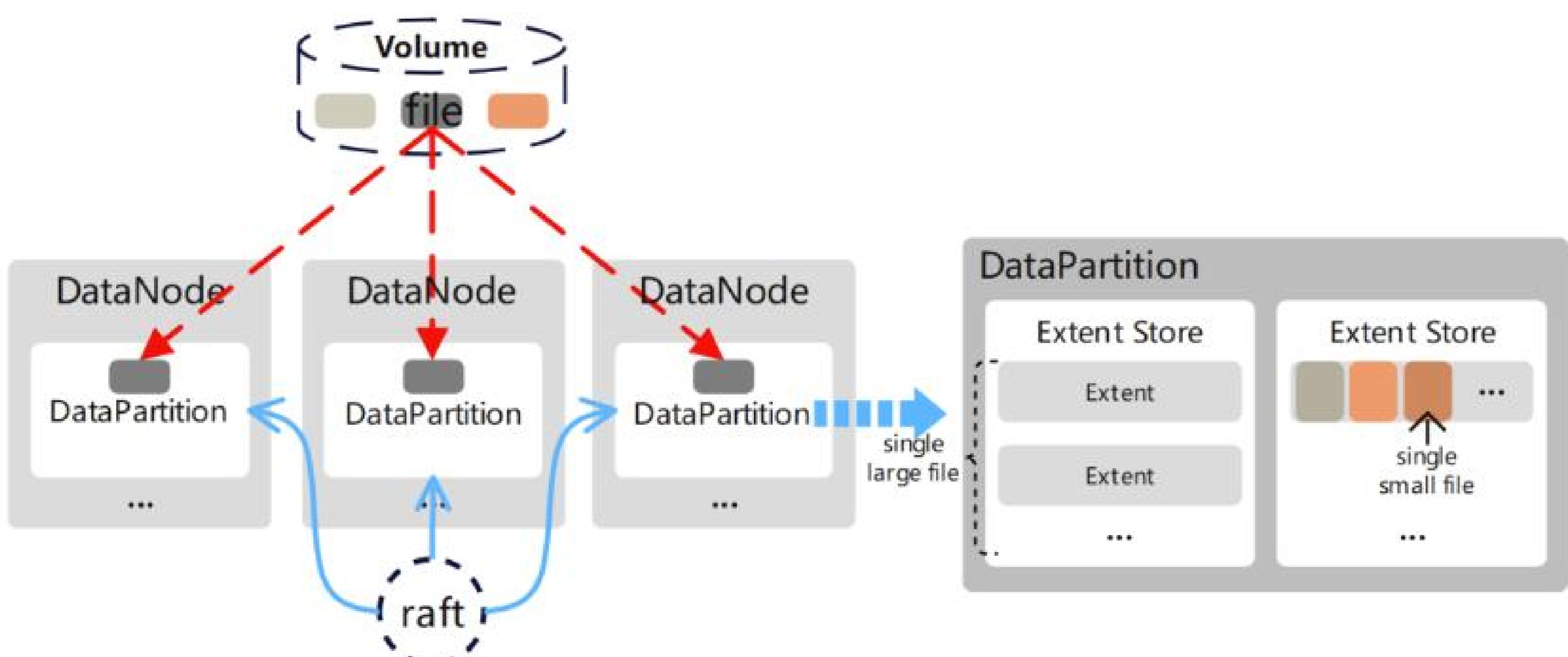
## 1. 系统架构：

### 01 系统架构



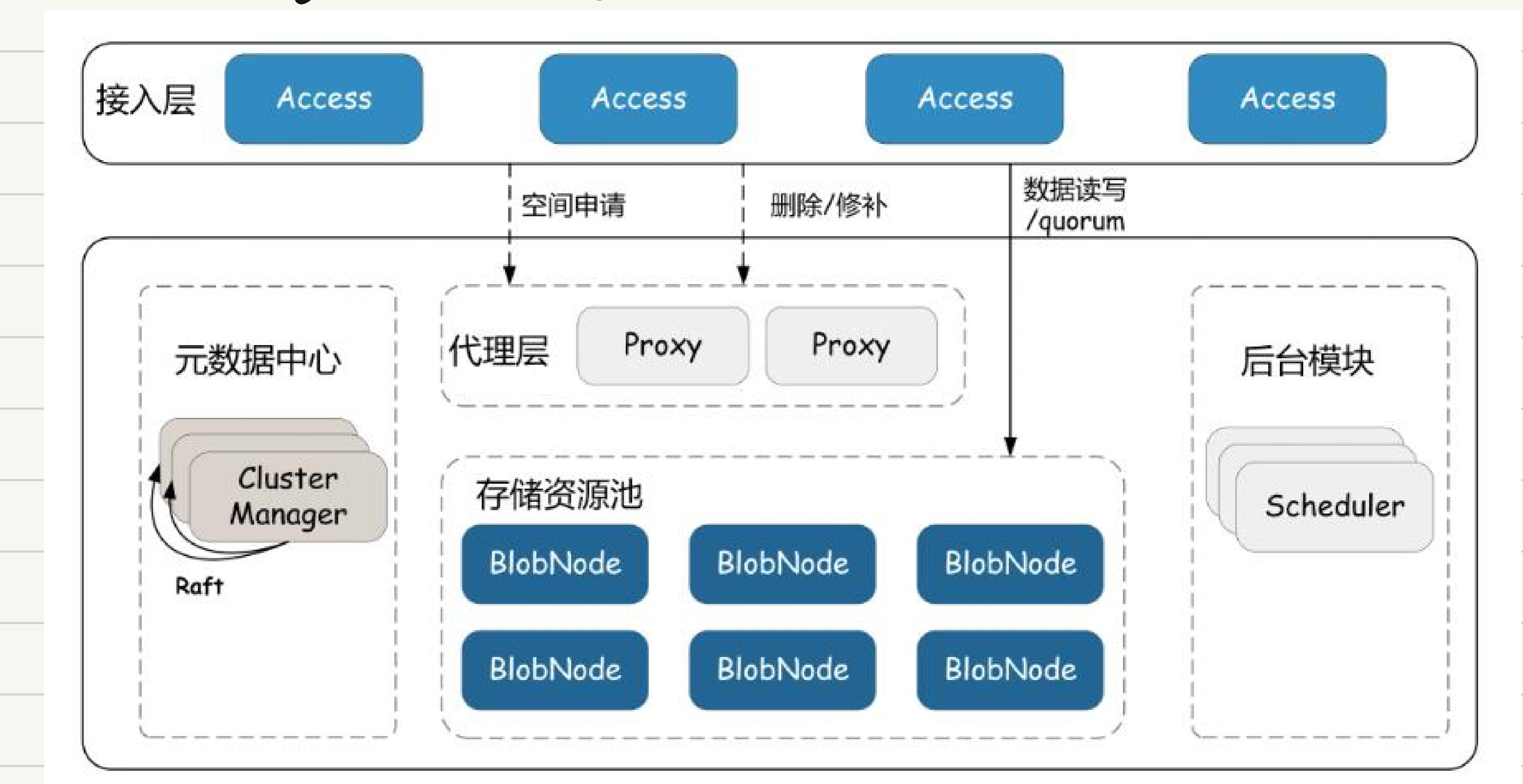
## 2. CubeFS 存储两种设计

### (1). 多副本子系统

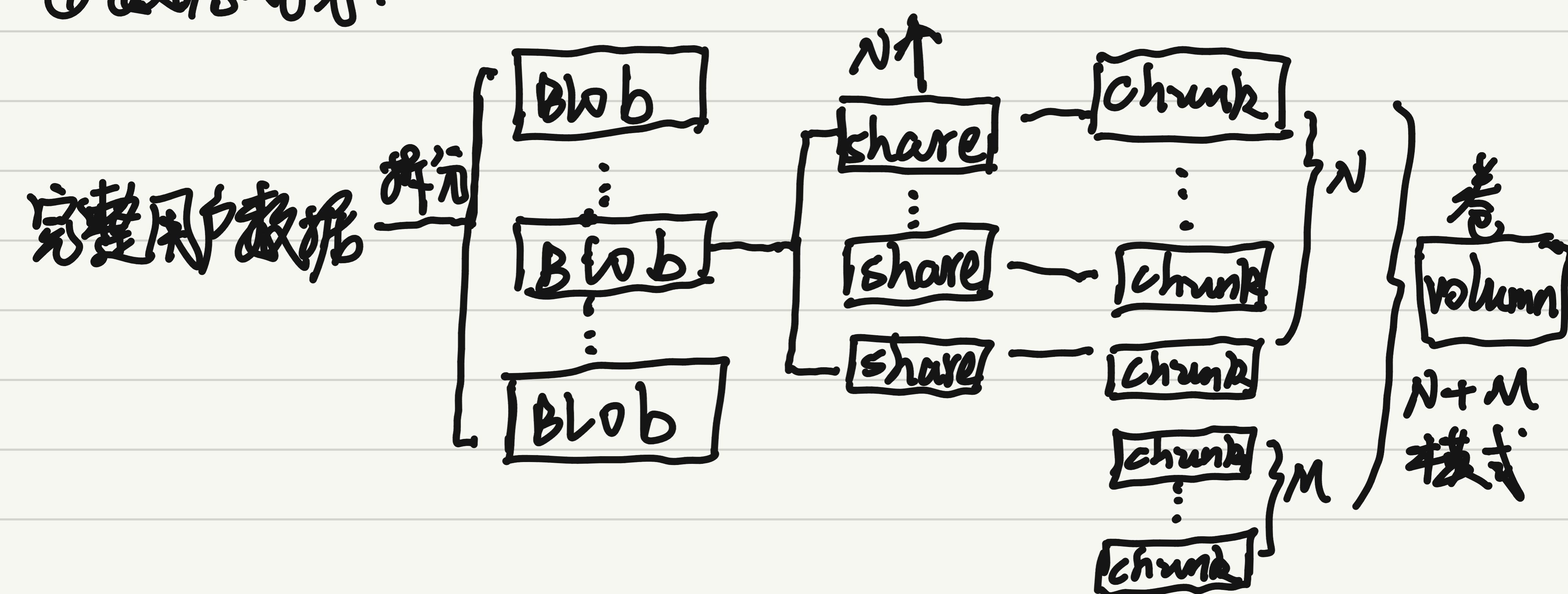


- ① 大文件由一个个 Extent 组成，写入从 Extent 0 编号写入
- ② 小文件聚合组成一个 Extent 存储（会记录每个文件偏移）
- ③ 数据复制：{ 顺序写入时：按主备复制方式，先复制主设备  
随机写入时：采用 Multi-Raft 复制协议 }

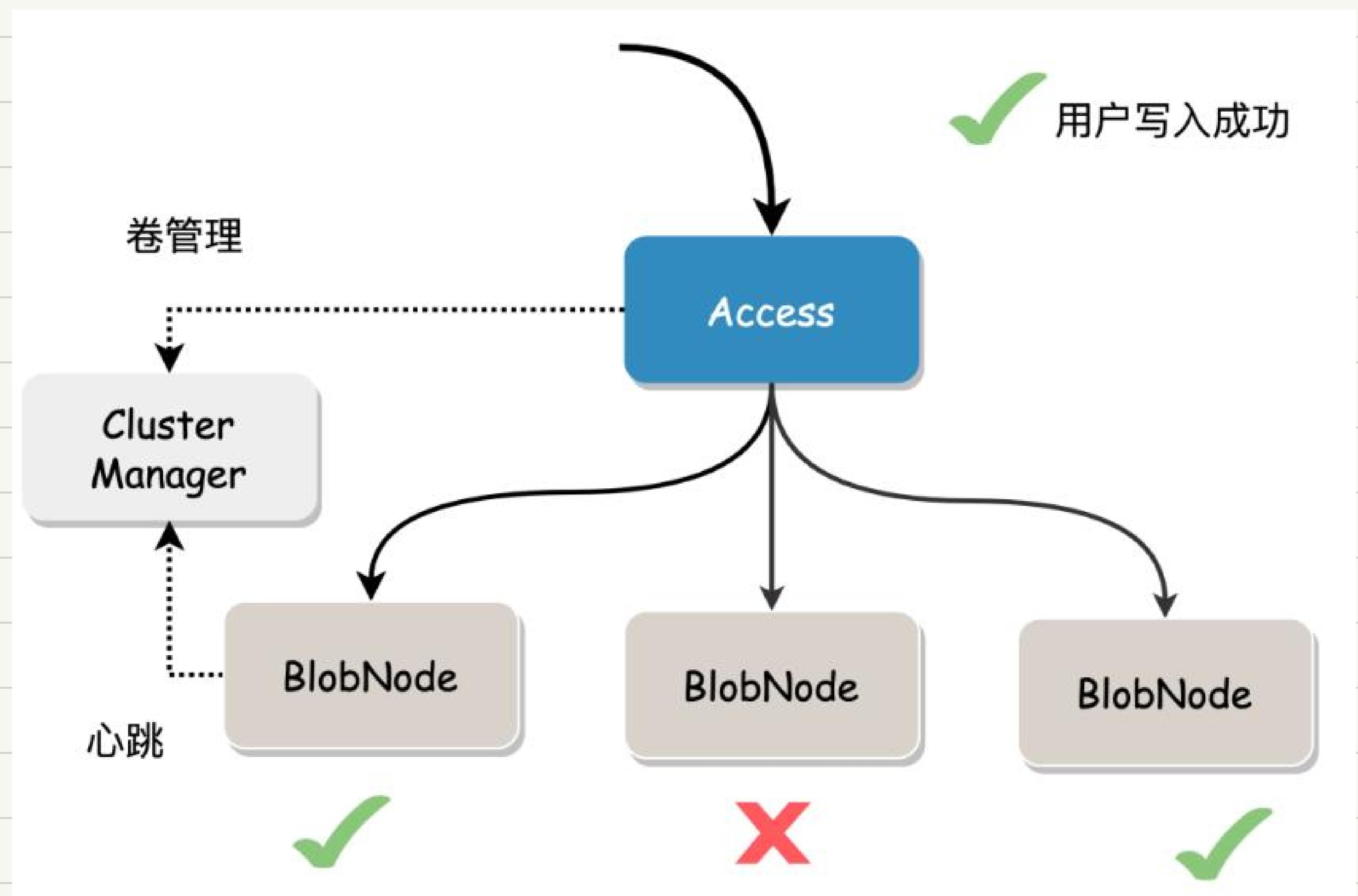
## (2). 约删冗余(ER)系统设计



## ① 数据划分：

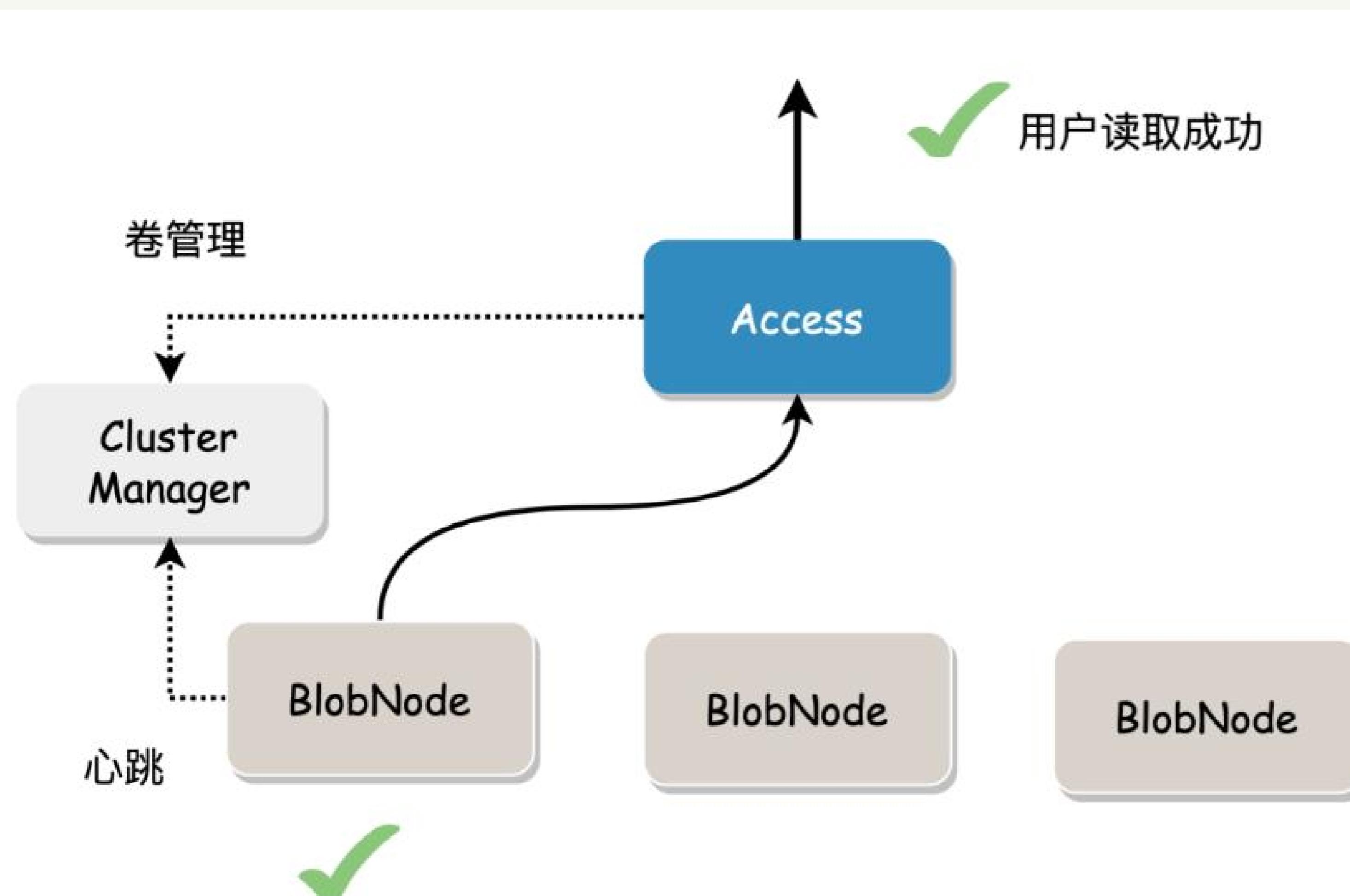


## ② 写流程



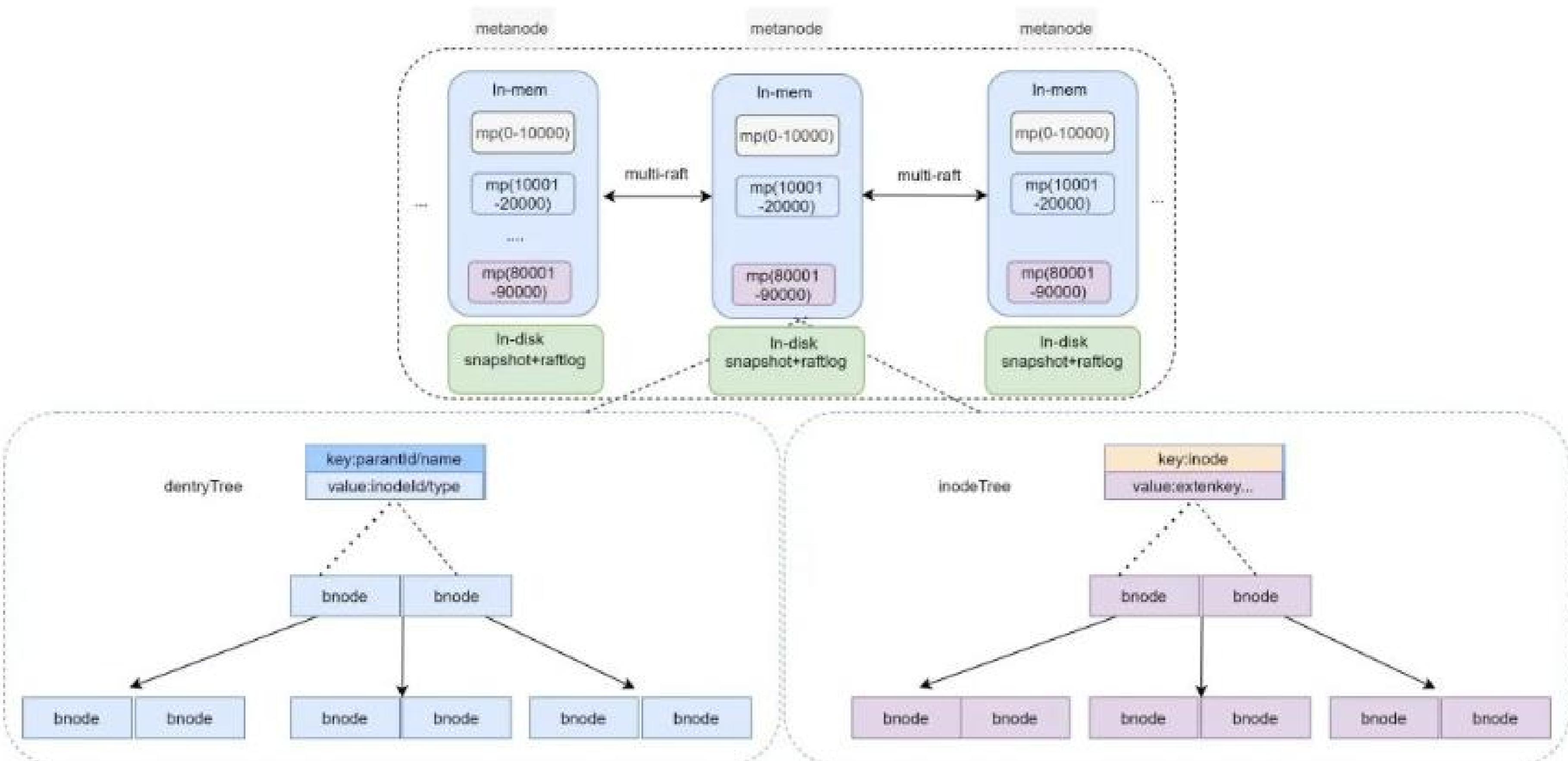
Blob采用Quorum模型，当有( $>n$ )份share写入成功时，即视为写入成功，失败写入的share会作异步修复。

## ③ 读流程：



若读取的块损坏，则会从其它结点，根据文件其它部分的数据和校验位修复。

## 3. 元数据存储.



① 元数据主要拆分成 {  
    inode (文件信息)  
    dentry (目录索引关系)}

其中当前文件的 dentry 信息存储于诸在父目录所在 MP 中

② 热点目录问题：采用虚拟拟目录解决

