ISCAS
中国科学院软件研究所
Institute of Software,Chinese Academy of Sciences

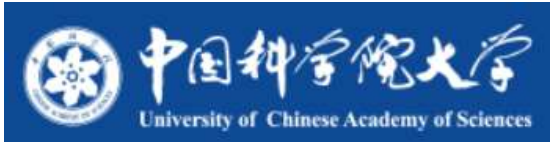
中国科学院大学
University of Chinese Academy of Sciences

文件系统（下）

李鹏

改编声明

- 本课程教学及PPT内容基于**上海交通大学并行与分布式系统研究所**发布的操作系统课程修改，已获得原作者授权，原课程官网：
  - https://ipads.se.sjtu.edu.cn/courses/os/index.shtml
- 本课程修改人为**中国科学院软件研究所**，用于国科大操作系统课程教学。



John K. Ousterhout, Mendel Rosenblum. (1991), *The Design and Implementation of a Log-Structured File System*

日志文件系统

日志结构文件系统 VS 日志文件系统

Journaling File System

- 为解决崩溃一致性问题
- 在日志中跟踪文件系统的所有更改
- 日志与磁盘上的数据分开

Log Structured File Systems

- 优化写操作，提升写入性能
- 在文件系统本身内部维护日志
- 磁盘上的文件系统数据以日志形式组织

日志文件系统（Log-structured FS）

The Design and Implementation of a Log-Structured File System

Mendel Rosenblum and John K. Ousterhout

Electrical Engineering and Computer Sciences, Computer Science Division
University of California
Berkeley, CA 94720
mendel@sprite.berkeley.edu, ouster@sprite.berkeley.edu

Abstract

This paper presents a new technique for disk storage management called a *log-structured file system*. A log-structured file system writes all modifications to disk sequentially in a log-like structure, thereby speeding up both file writing and crash recovery. The log is the only structure on disk; it contains indexing information so that files can be read back from the log efficiently. In order to maintain large free areas on disk for fast writing, we divide the log into *segments* and use a *segment cleaner* to compress the live information from heavily fragmented segments. We present a series of simulations that demonstrate the efficiency of a simple cleaning policy based on cost and benefit. We have implemented a prototype log-

magnitude more efficiently than current file systems.

Log-structured file systems are based on the assumption that files are cached in main memory and that increasing memory sizes will make the caches more and more effective at satisfying read requests[1]. As a result, disk traffic will become dominated by writes. A log-structured file system writes all new information to disk in a sequential structure called the *log*. This approach increases write performance dramatically by eliminating almost all seeks. The sequential nature of the log also permits much faster crash recovery: current Unix file systems typically must scan the entire disk to restore consistency after a crash, but a log-structured file system need only examine the most recent portion of the log.

1991 Symposium on Operating Systems Principles
1992 *ACM Transactions on Computer Systems*

John K. Ousterhout

Sprite 负责人
TCL/TK 作者
Magic VLSI CAD 作者
Co-scheduling 提出者
Raft 一致性协议
目前是Stanford大学教授

Mendel Rosenblum

VMware 联合创始人
目前是Stanford大学教授

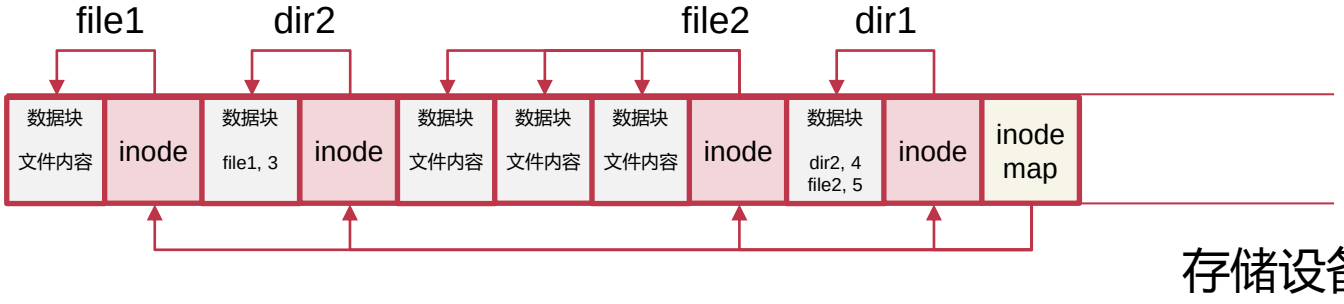
诞生背景(SOSP91)

- **内存大小不断增长**
- **随机I/O性能与顺序I/O性能之间存在巨大的差距**
  - 传输带宽每年增加50%-100%
  - 寻道和旋转延迟每年降低5%-10%
- **现有文件系统在许多常见工作负载上表现不佳**
  - 创建大小为一个块的新文件
    - 创建/写入新inode
    - 更新inode位图
    - 更新数据位图
    - 写入新数据块
- **需要加强对RAID与Flash的支持**

理想的文件系统应
- 专注于写性能
- 尽量利用磁盘的顺序带宽

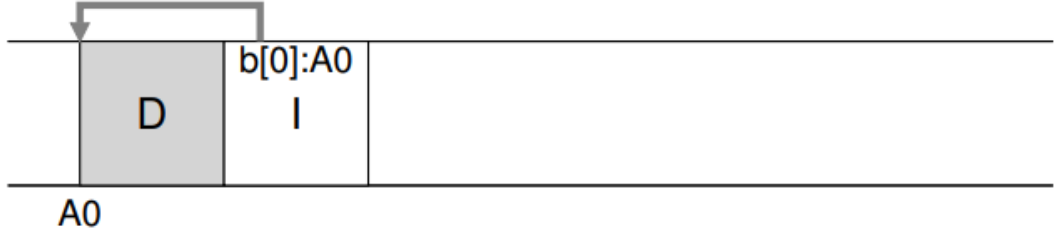
日志文件系统（Log-structured FS）

- 假设：文件被缓存在内存中，文件读请求可以被很好的处理
  - 于是，文件写成为瓶颈
- 块存储设备的顺序写比随机写速度很块
  - 磁盘寻道时间
- 将文件系统的修改以日志的方式**顺序写入**存储设备



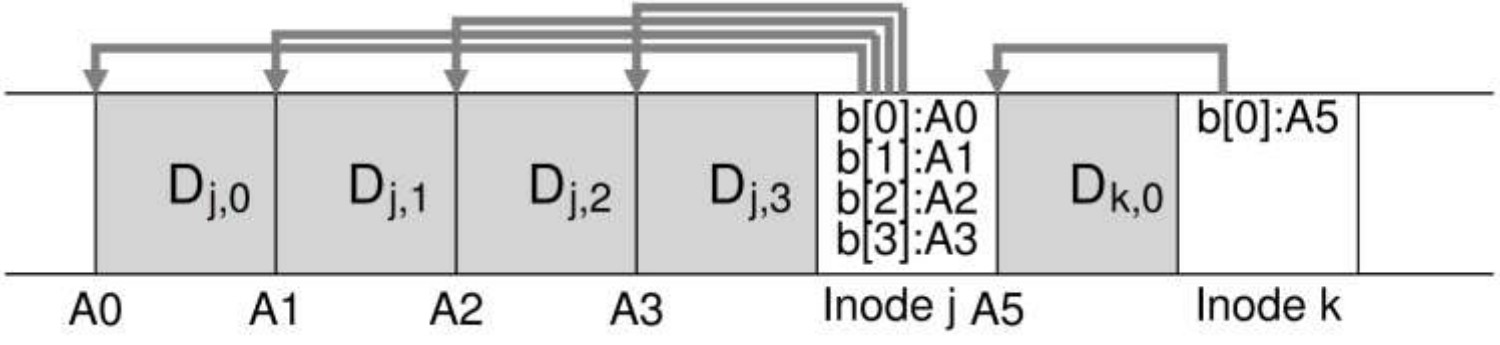
创建大小为一个块的新文件

- **Ext2**
  - 一个新的inode
  - 更新inode位图
  - 更新数据位图
  - 新数据块
- **LFS**
  - 写入两个连续的块
  - 问题
    - 能否写入更多连续的块?
    - inode如何组织?



顺序而高效地写入

- 高效=顺序+连续
- 写缓冲（write buffering）
  - 一次写入的大块更新成为段（Segment）



创建2个文件，一次写入

问题2：inode的组织管理


```
inum | reclen | strlen | name
   5      12       2     .
   2      12       3     ..
  12      12       4     foo
  13      12       4     bar
  24      36      28     foobar_is_a_pretty_longname
```

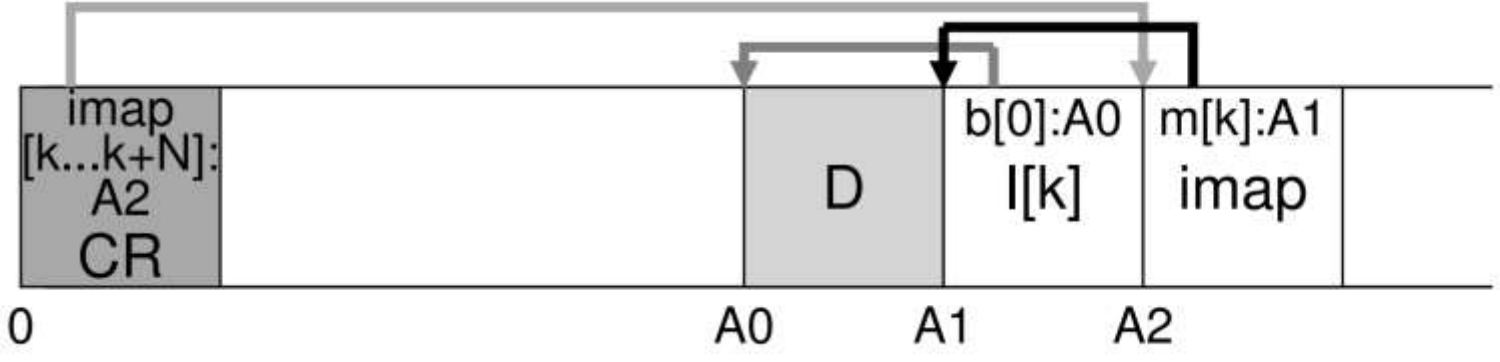
- **LFS中**
  - inode被分散在整个磁盘上
  - 永远不会覆盖，最新的inode会不断移动

间接解决方案：inode映射

- **inode num和inode地址之间插入一个间接层**
  - inode map，imap
  - imap(inum)=current inode address
  - 每次inode写入磁盘，imap都会更新
  - imap通常常驻内存
- **问题：imap如何持久化?**

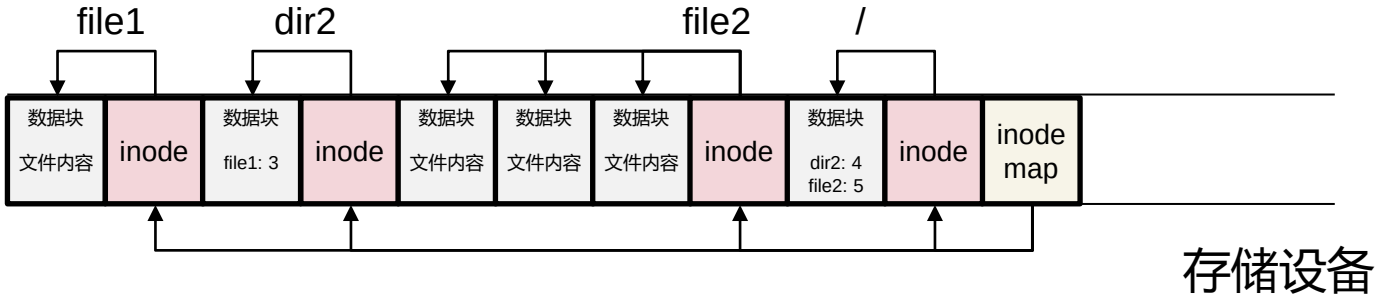
检查点区域

- **检查点区域，Checkpoint Region，CR**
  - 固定且已知的位置
  - 包含指向最新imap的指针（即地址）
  - 检查点区域定期持久化（例如每30秒左右）



举例

- 一个日志文件系统
  - 有4个inode，位置记录在inode map中
  - 对应4个文件分别为：/，/dir2，/file2，/dir2/file1



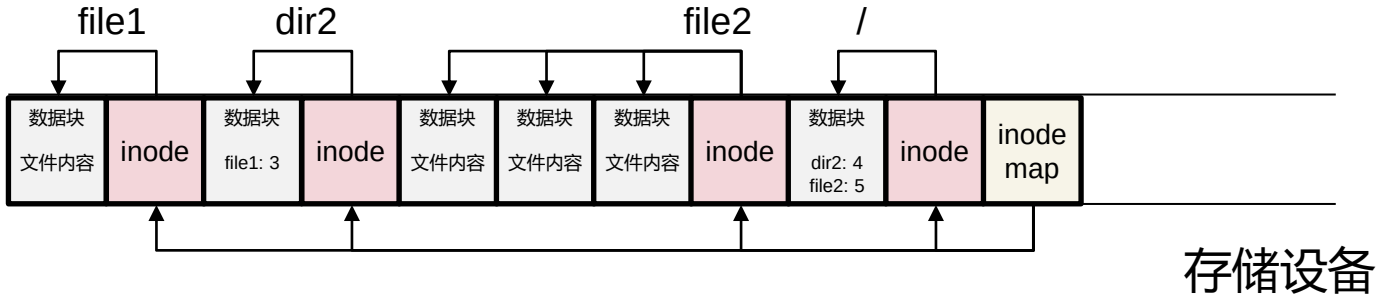
从硬盘中读取文件

- **假设内存中没有任何数据结构**
  - 先读取检查点区域
  - 加载imap

  - 对于指定文件的inum，找到inode地址
  - 读取inode
  - 读取文件内容

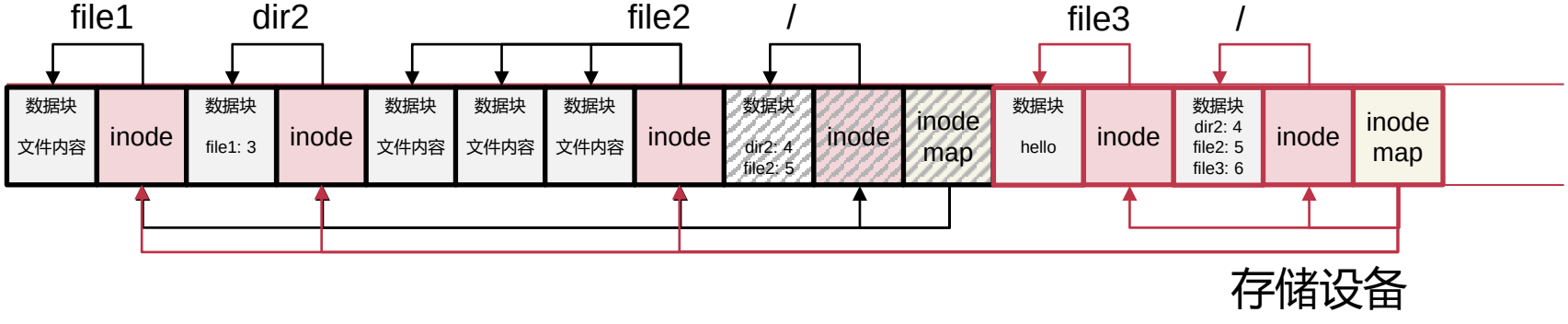
创建文件举例

- echo hello > /file3
  - 创建文件
  - 修改文件数据



创建文件举例

- echo hello > /file3
  - 创建文件
  - 修改文件数据

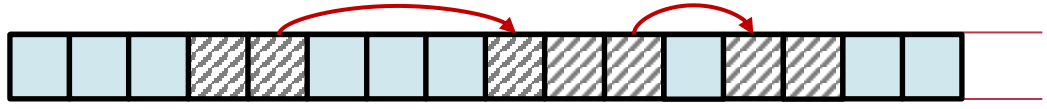


空间回收利用

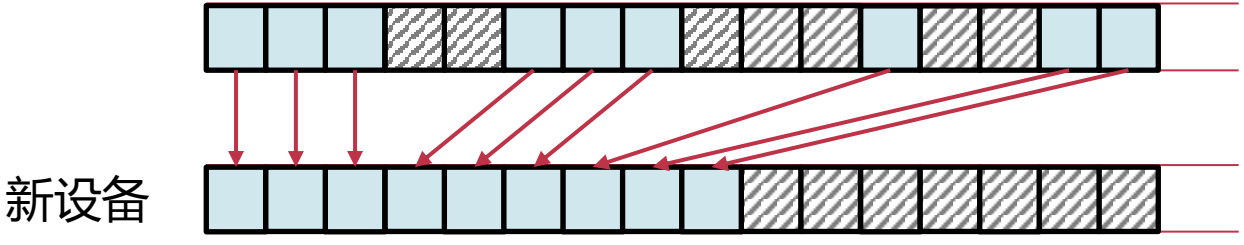
- 存储设备最开始是一个连续的空闲空间
- 随文件系统的使用，日志写入位置会接近存储设备末端
- 需重新利用前面的设备空间
  - 文件系统数据被修改后，此前的块被无效化
  - 如何组织前面无效空间，以重新利用它们?

空间回收管理方法

- 串联：将所有空闲空间用链表串起来
  - 磁盘空间会越来越碎，影响到LFS的大块顺序写的性能

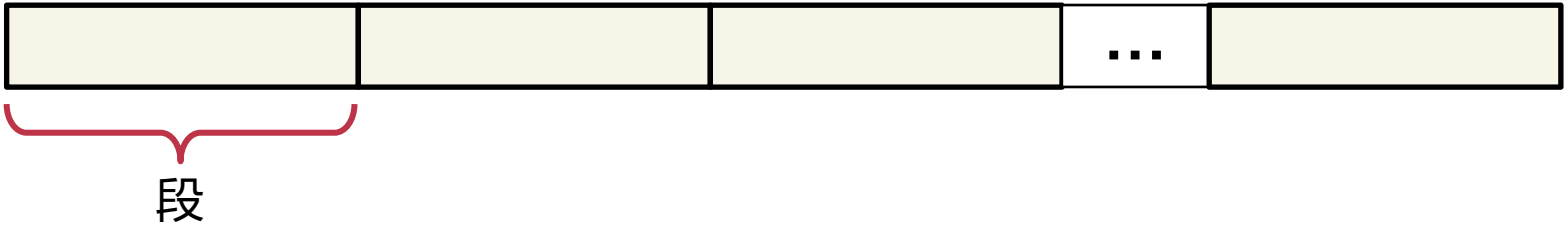


- 拷贝：将所有的有效空间整理拷贝到新的存储设备
  - 数据需要拷贝



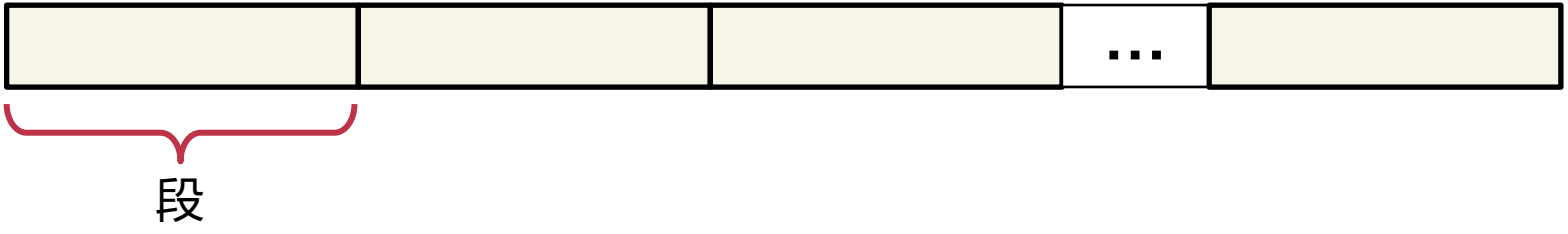
空间回收管理方法：段（Segment）

- 串联
  - 磁盘空间会越来越碎，影响到LFS的大块顺序写的性能
- 拷贝
  - 数据需要拷贝
- 串联和拷贝两种方法的结合：段



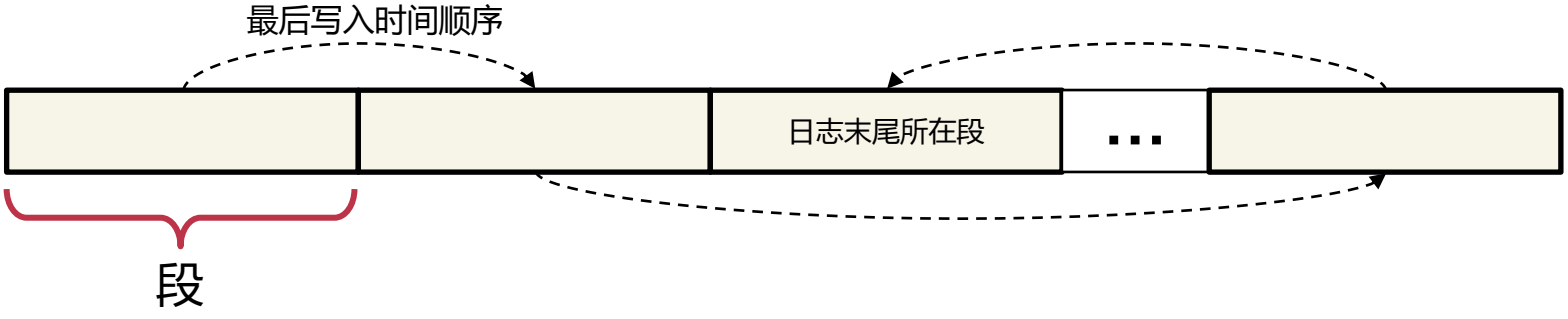
空间回收管理方法：段（Segment）

- 一个设备被拆分为定长的区域，称为段
  - 段大小需要足以发挥出顺序写的优势，512KB、1MB等
- 每段内只能顺序写入
  - 只有当段内全都是无效数据之后，才能被重新使用
- 干净段用链表维护（对应串联方法）



段使用表

- 段使用表
  - 记录每个段中有效字节数
    - 归零时变为干净段
  - 记录了每个段最近写入时间
    - 将非干净段按时间顺序连在一起，形成逻辑上的连续空间

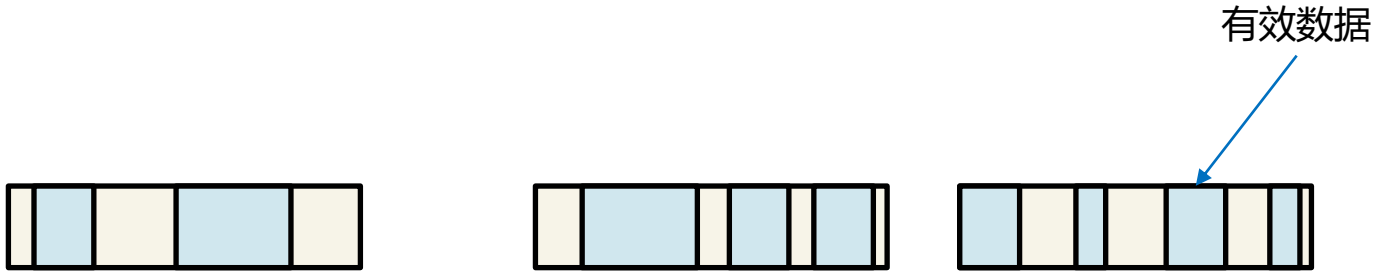


段清理

1. 将一些段读入内存中准备清理

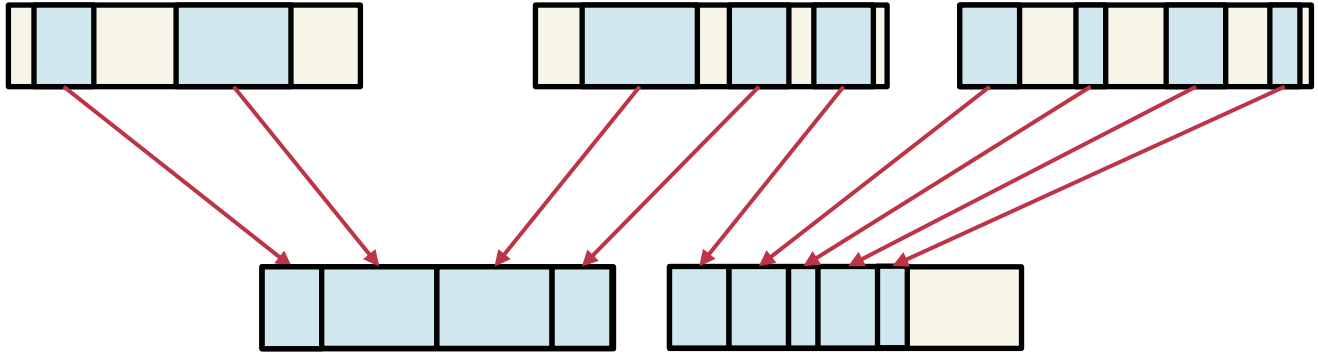
段清理

1. 将一些段读入内存中准备清理
2. 识别出有效数据



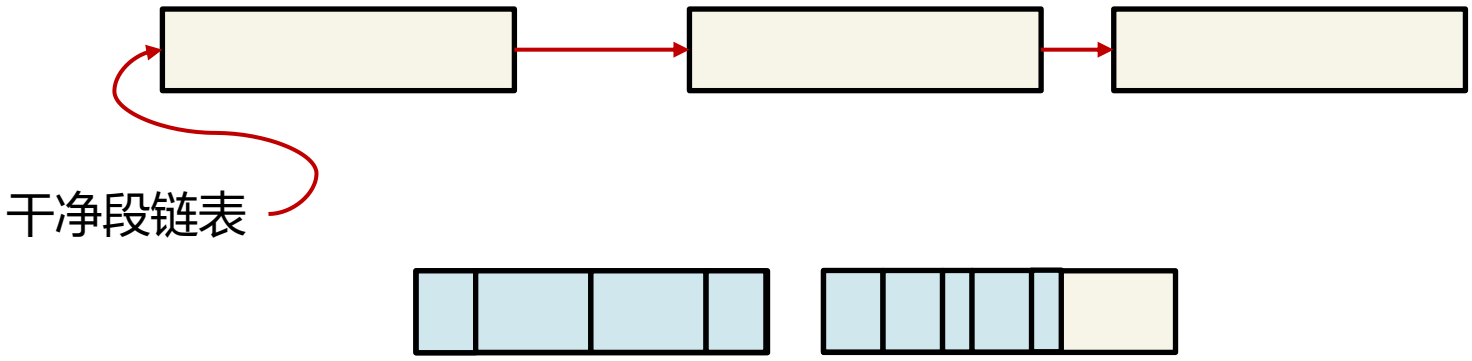
段清理

1. 将一些段读入内存中准备清理
2. 识别出有效数据
3. 将有效数据整理后写入到干净段中（对应拷贝方法）



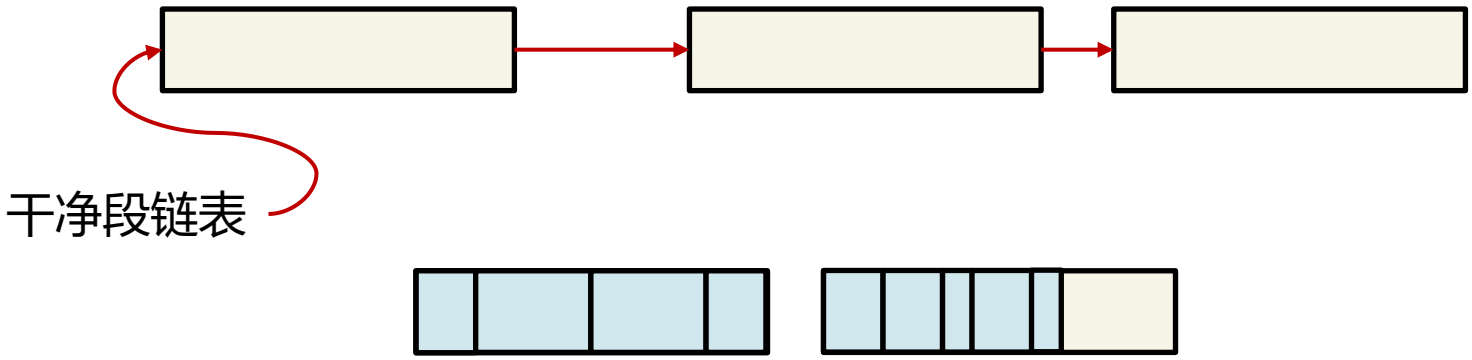
段清理

1. 将一些段读入内存中准备清理
2. 识别出有效数据
3. 将有效数据整理后写入到干净段中（对应拷贝方法）
4. 标记被清理的段为干净



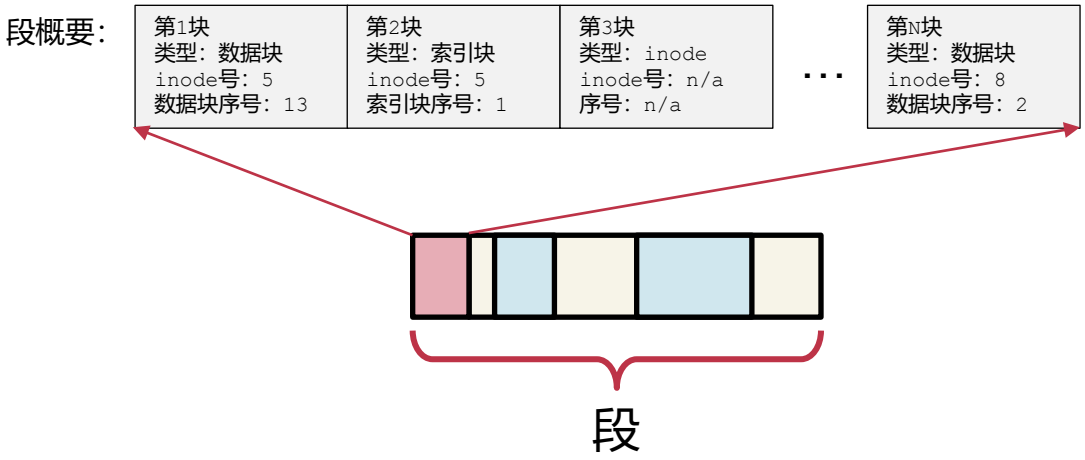
段清理

1. 将一些段读入内存中准备清理 ✓
2. 识别出有效数据 ?
3. 将有效数据整理后写入到干净段中（对应拷贝方法） ✓
4. 标记被清理的段为干净 ✓



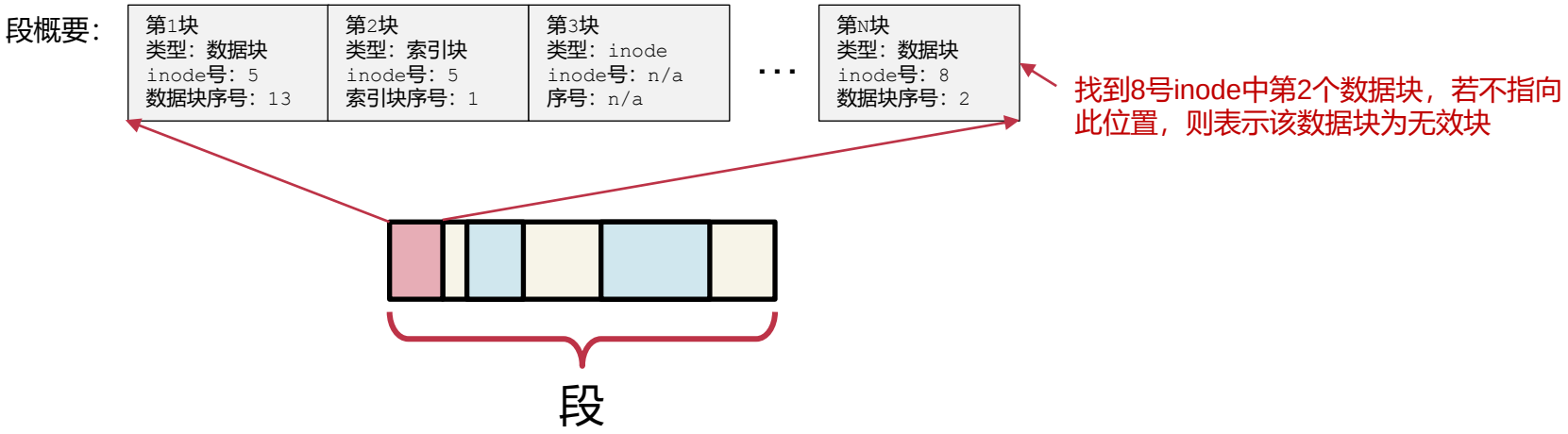
识别有效数据

- 每个段中保存有**段概要（Segment Summary）**
  - 记录每个块被哪个文件的哪个位置所使用
    - 如：数据块可使用inode号和第几个数据块来表示位置



识别有效数据

- 每个段中保存有**段概要（Segment Summary）**
  - 记录每个块被哪个文件的哪个位置所使用
    - 如：数据块可使用inode号和第几个数据块来表示位置
  - 数据块的有效性可通过对比该位置上的现有指针来判断



清理的策略有多种维度

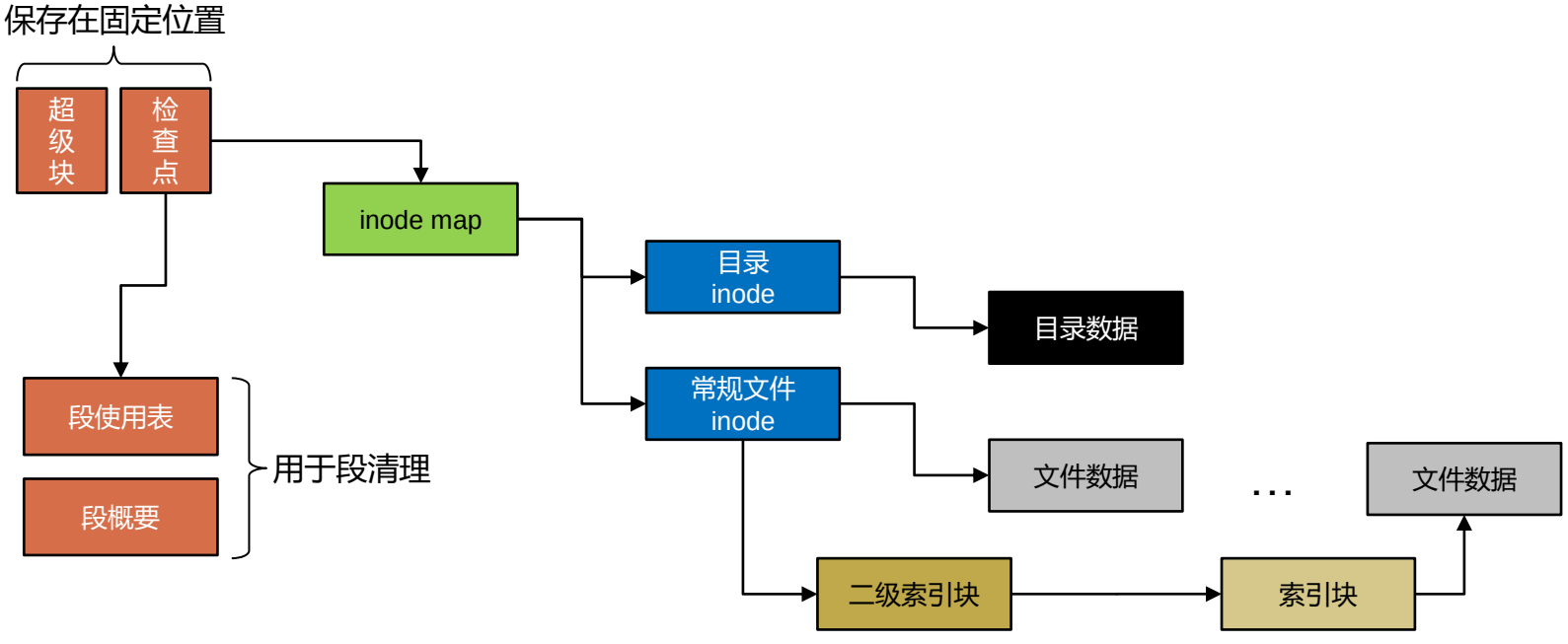
- 清理策略
  - 什么时候执行清理?
    - 后台持续清理？晚上清理？磁盘要满的时候清理?
  - 一次清理多少段？清理哪些段?
  - 有效数据应该以什么顺序排序写入新的段?
    - 维持原顺序？相同目录放一起？相近修改时间放一起?
- 段使用表为策略提供辅助信息
  - 记录每个段中有效字节数
  - 记录每个段最近写入的时间

Sprite LFS的数据结构

- 固定位置的结构
  - 超级块、检查点（checkpoint）区域
- 以Log形式保存的结构
  - inode、间接块（索引块）、数据块
  - inode map：记录每个inode的当前位置
  - 段概要（Segment Summary）：记录段中的有效块
  - 段使用表：记录段中有效字节数、段的最后修改时间

总结：LFS的结构

逻辑结构

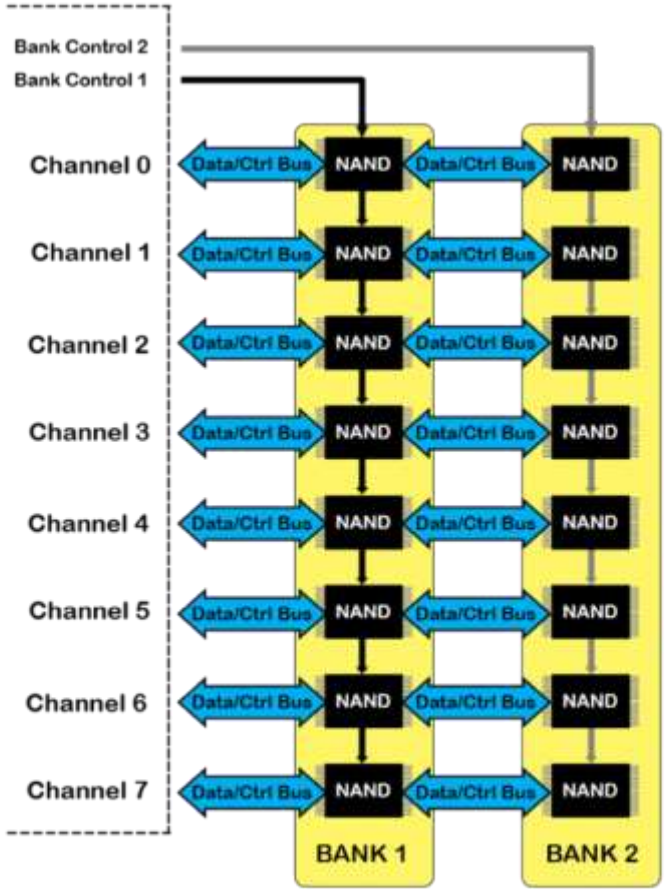


存储结构

闪存盘的文件系统

闪存盘的组织

- **通道（Channel）**
  - 控制器可以同时访问的闪存芯片数量
- **多通道（Multi-channel）**
  - 低端盘有2或4个通道
  - 高端盘有8或10个通道

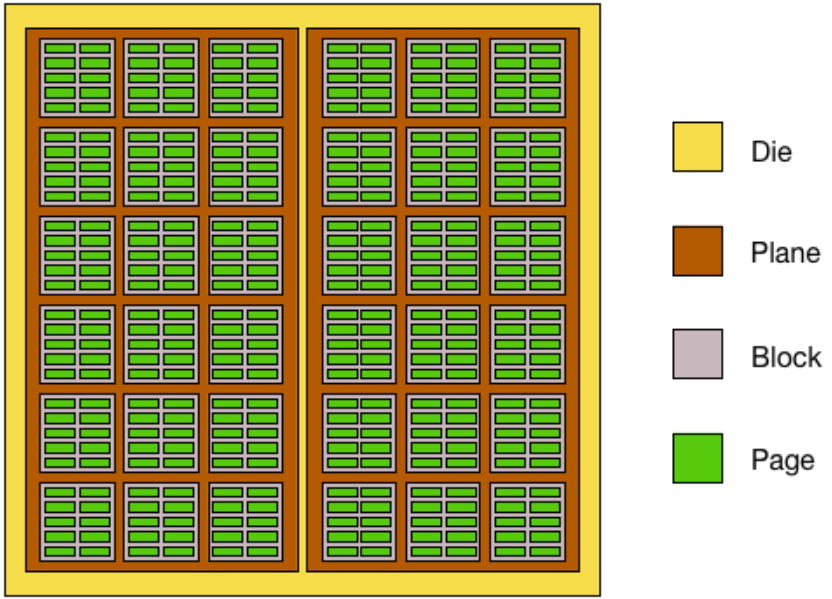


NAND Banks and Channels Illustration

https://www.cactus-tech.com/resources/blog/details/solid-state-drive-primer-8-controller-architecture-channels-and-banks/

闪存盘的组织

- (NAND) 闪存盘组织结构
  - A chip/package
    => 1/2/4 dies
    => 1/2 planes
    => n blocks （块）
    => n pages （页）
    => n cells
    => 1/2/3/4 levels



https://www.extremetech.com/electronics/274944-samsungs-new-v-nand-could-mean-cheap-plentiful-multi-terabyte-ssds

闪存盘的性质

- **非对称的读写与擦除操作**
  - 页 (page) 是读写单元 (8-16KB)
  - 块 (block) 是擦除单元 (4-8MB)
- **Program/Erase cycles**
  - 写入前需要先擦除
  - 每个块被擦除的次数是有限的
- **随机访问性能**
  - 没有寻道时间
  - 随机访问的速度提升，但仍与顺序访问有一定差距

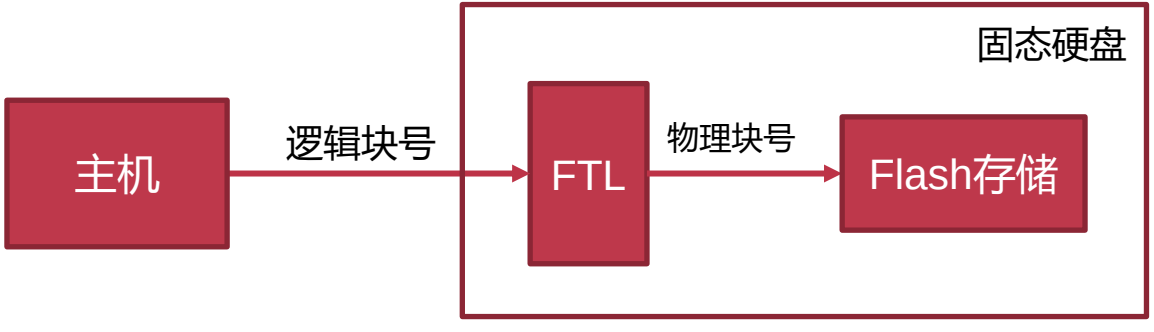
闪存盘的性质

- **磨损均衡**
  - 频繁写入同一个块会造成写穿问题
  - 将写入操作均匀的分摊在整个设备
- **多通道**
  - 高并行性
- **异质Cell**
  - 存储1到4个比特：SLC 、MLC、TLC、 QLC

Flash Translation Layer (FTL)

似曾相识?

- **逻辑地址到物理地址的转换**
  - 对外使用逻辑地址
  - 内部使用物理地址
  - 可软件实现，也可以固件实现
  - 用于垃圾回收、数据迁移、磨损均衡（wear-levelling）等



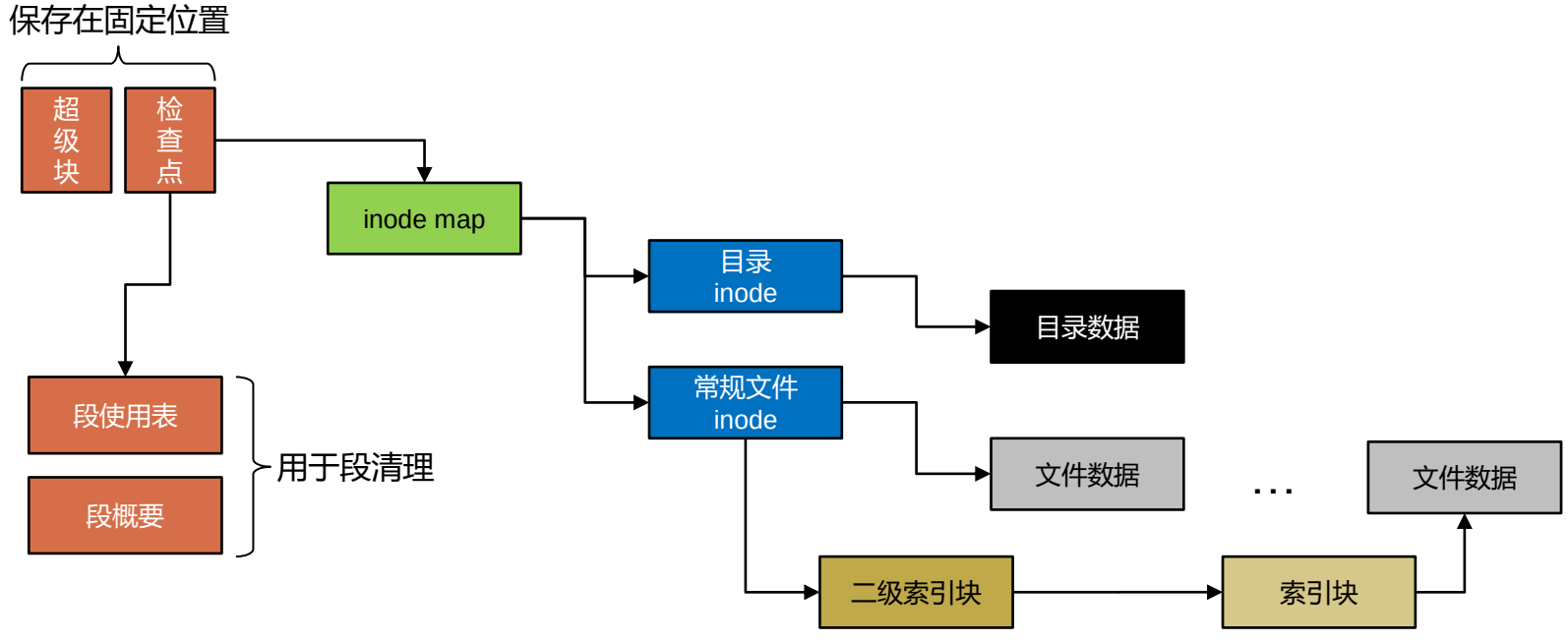
LFS与Flash很相似?

- LFS
  - Segment清理后才能使用
  - 顺序写入
  - 需要清理（垃圾回收）
- Flash
  - Block擦除后才能使用
  - 需要考虑磨损均衡
  - 需要垃圾回收

F2FS文件系统： Flash Friendly File System

回顾：LFS的结构

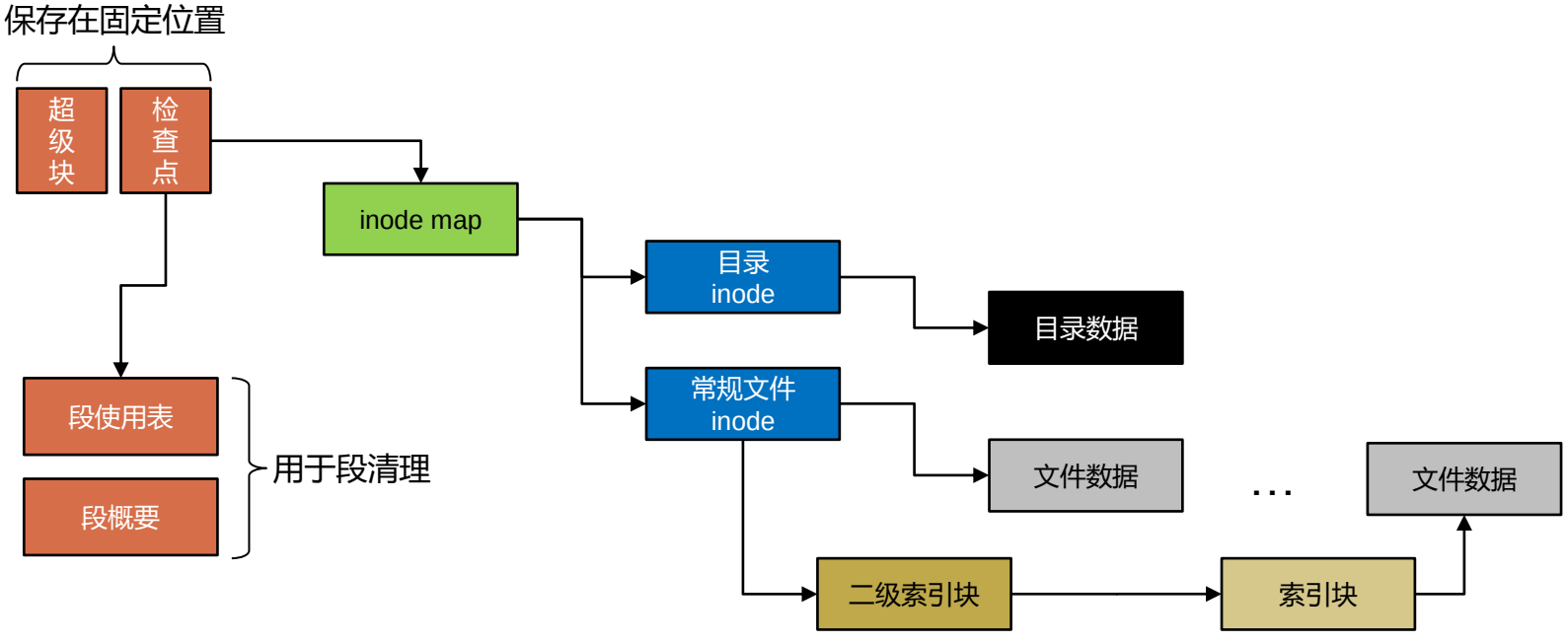
逻辑结构



存储结构

LFS的问题1：递归更新问题

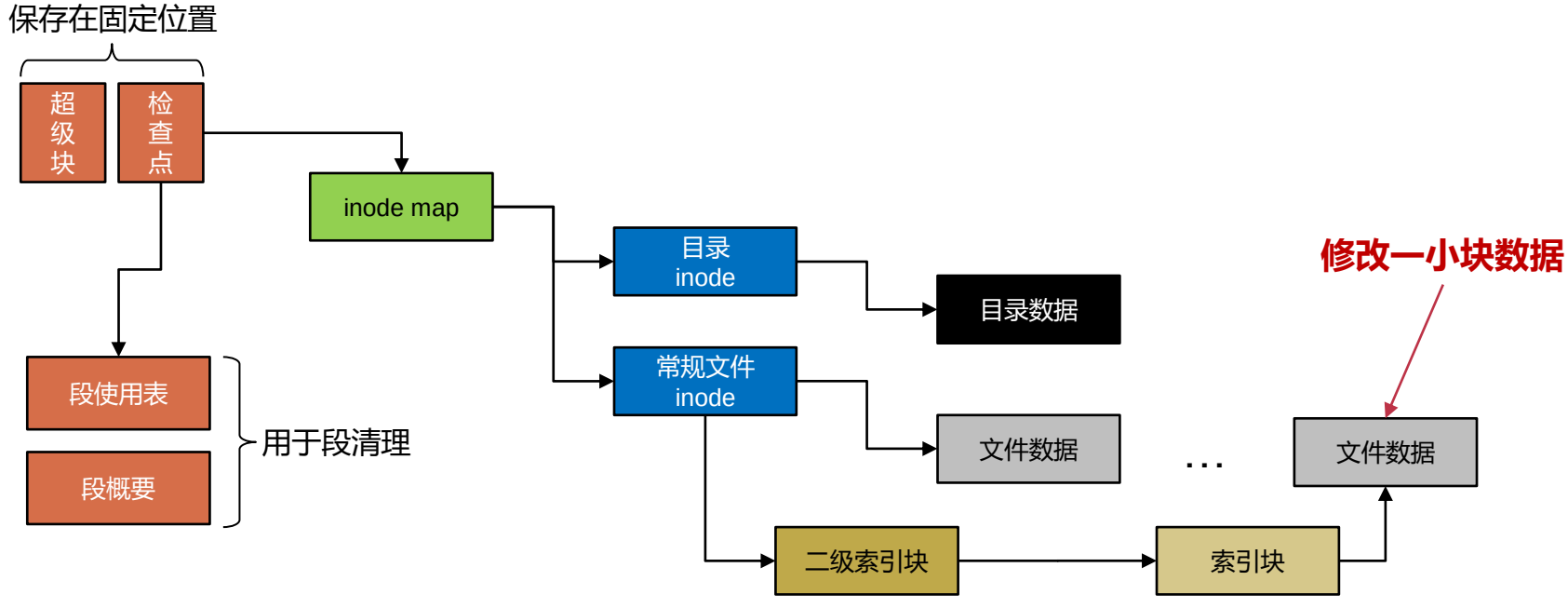
逻辑结构



存储结构

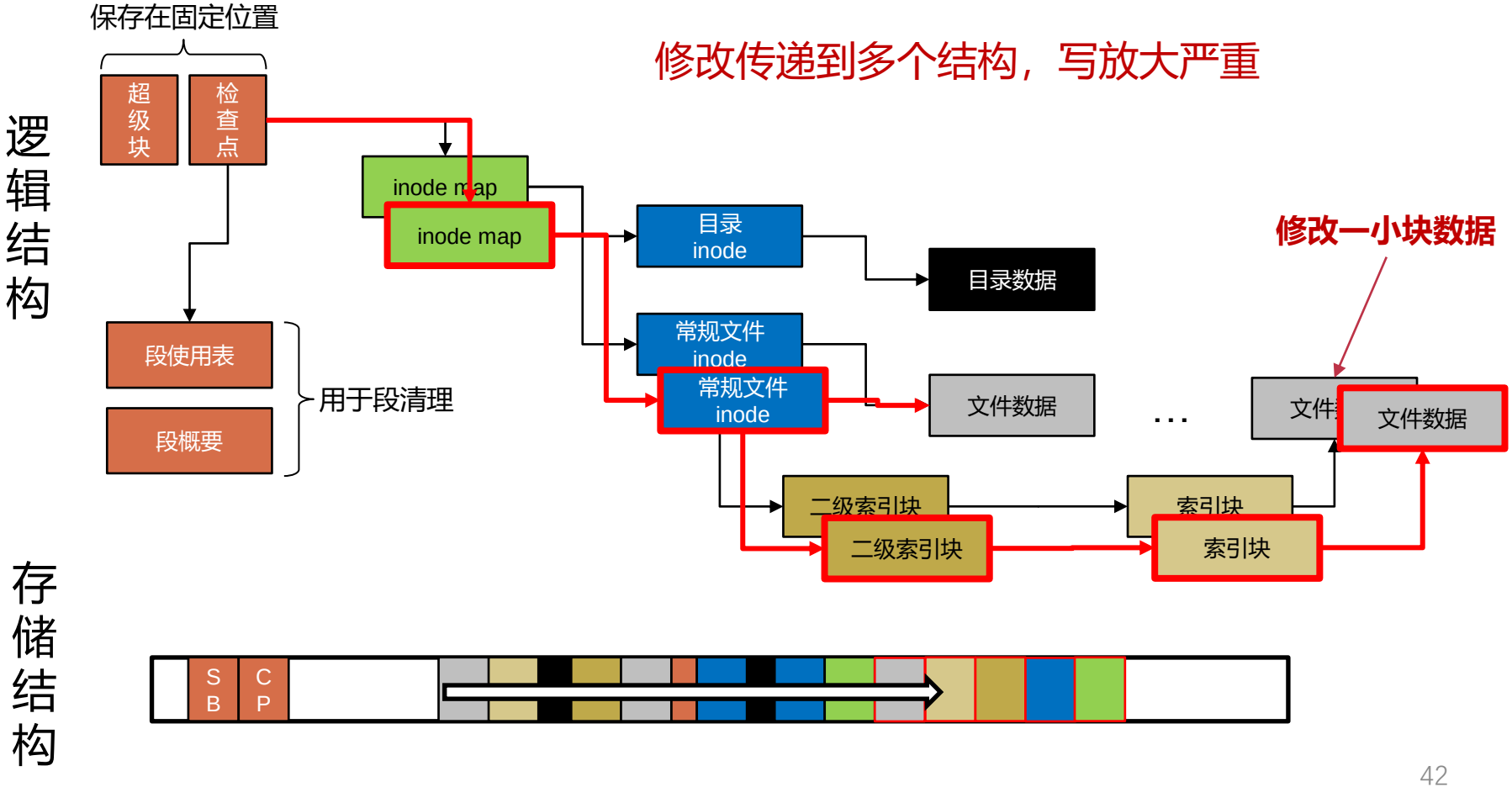
LFS的问题1：递归更新问题

逻辑结构

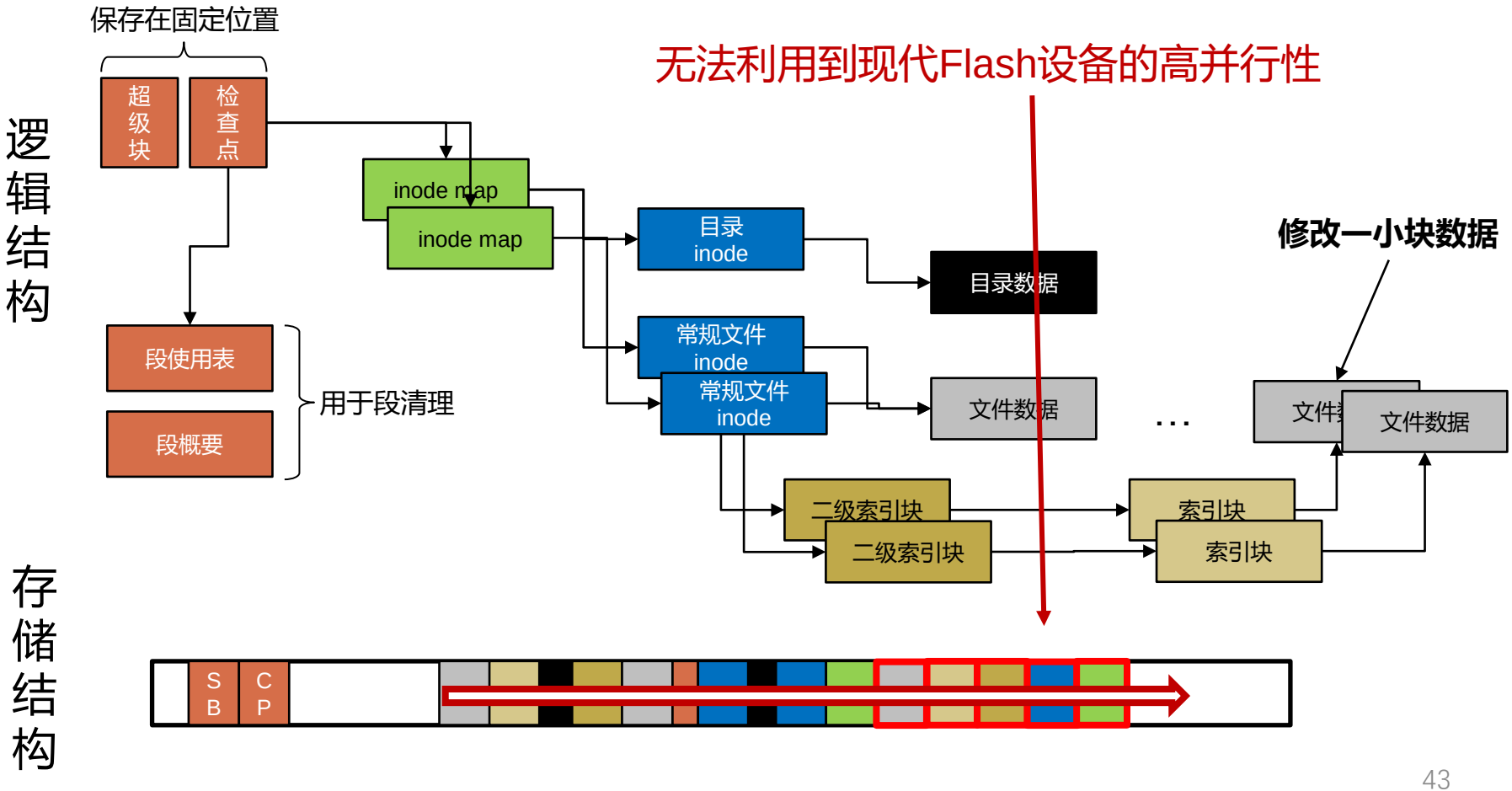


存储结构

LFS的问题1：递归更新问题

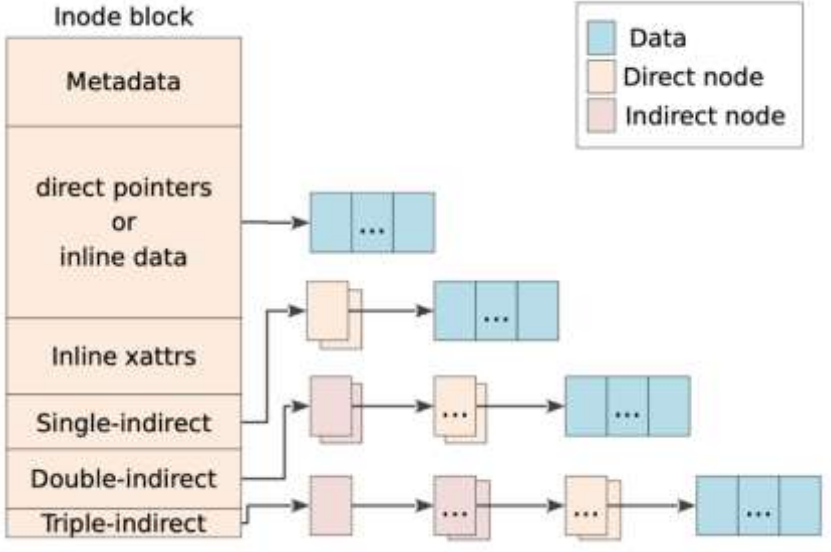


LFS的问题2：单一log顺序写入



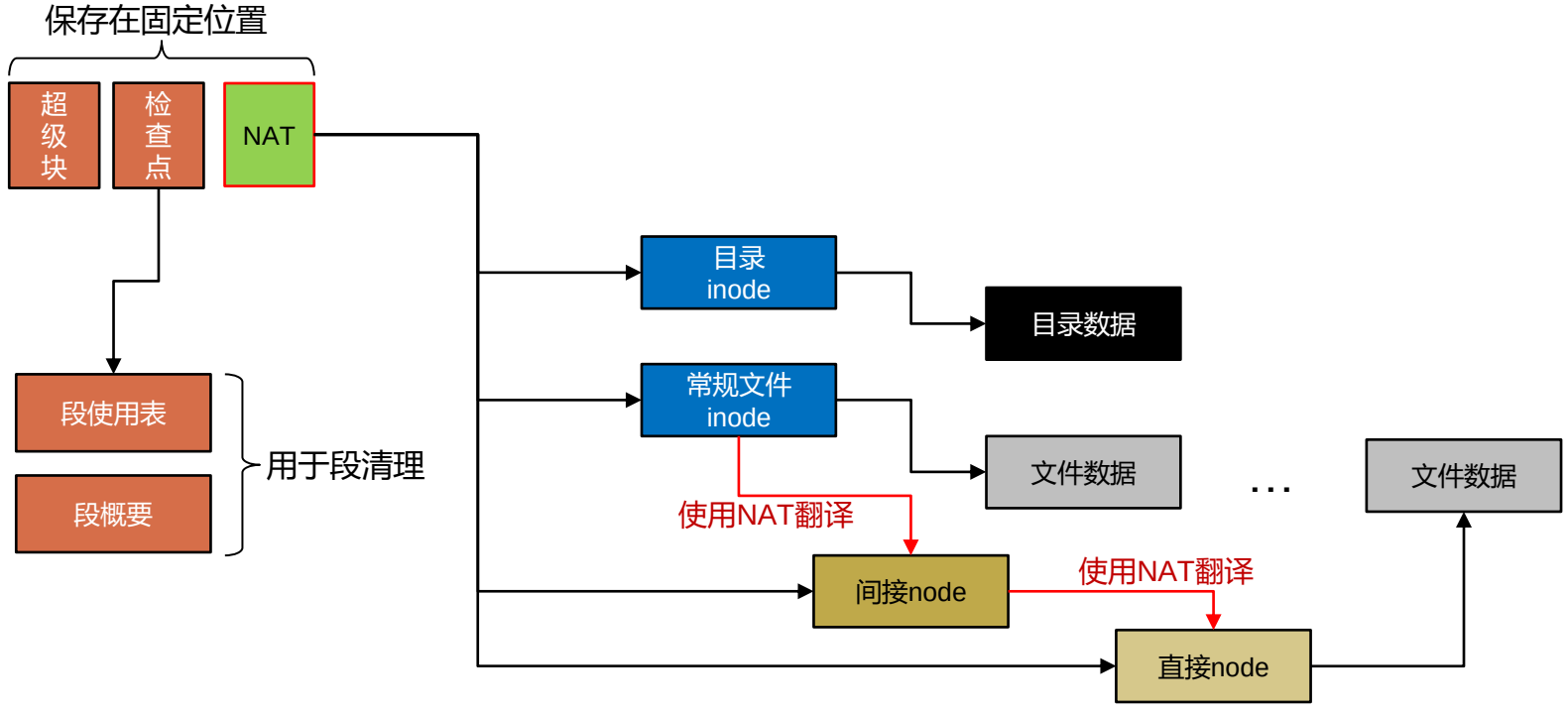
F2FS的改进1：NAT

- **引入一层 indirection：NAT（node地址转换表）**
  - NAT：Node Address Table
  - 维护node号到逻辑块号的映射
  - Node号需转换成逻辑块号才能使用
- **F2FS中的文件结构**
  - 直接node：保存数据块的逻辑块号
  - 间接node：保存node号
    （相当于索引块）
  - 数据块：保存数据

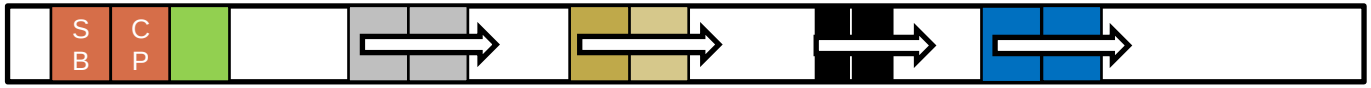


F2FS的改进1：NAT

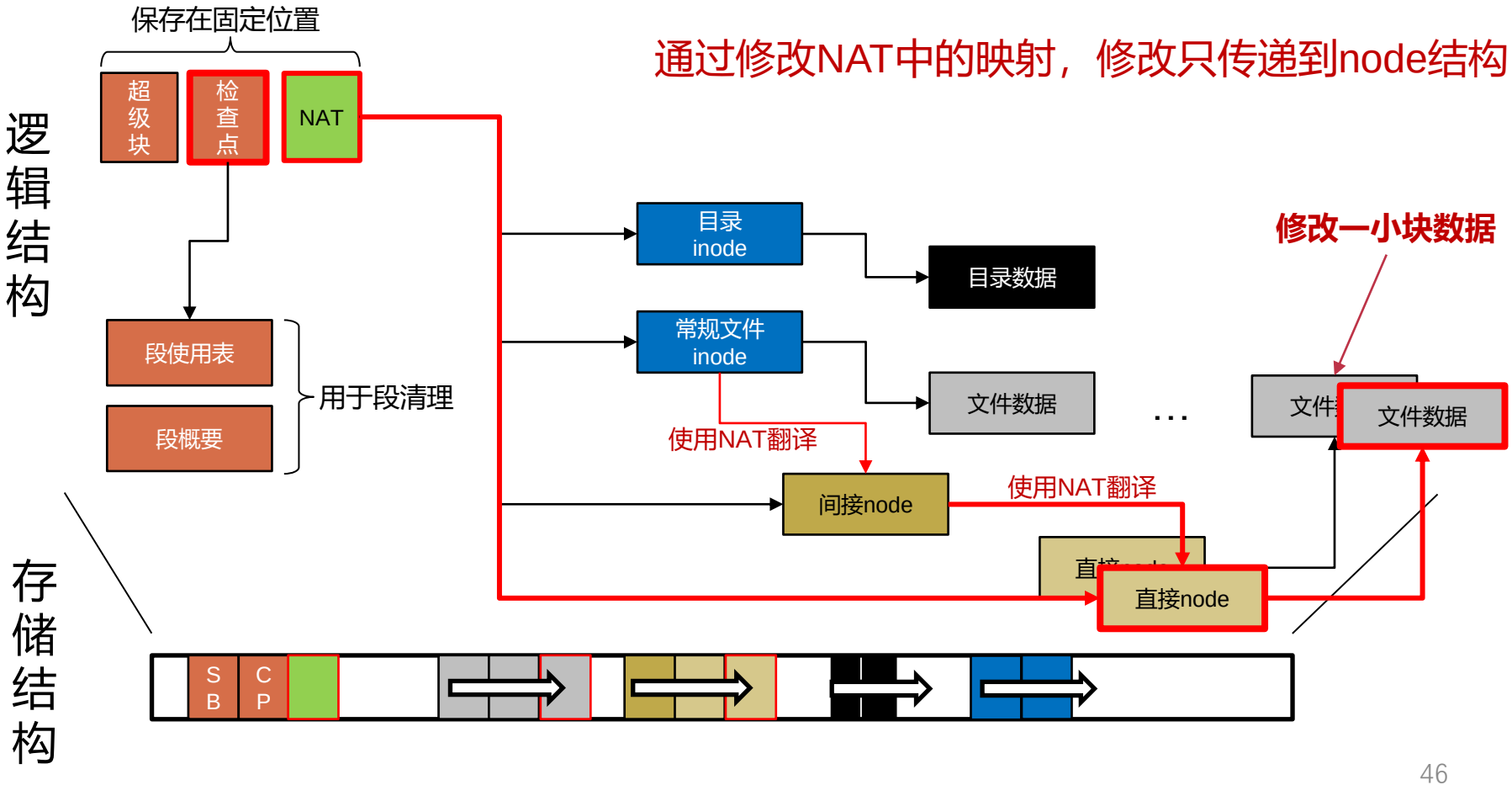
逻辑结构



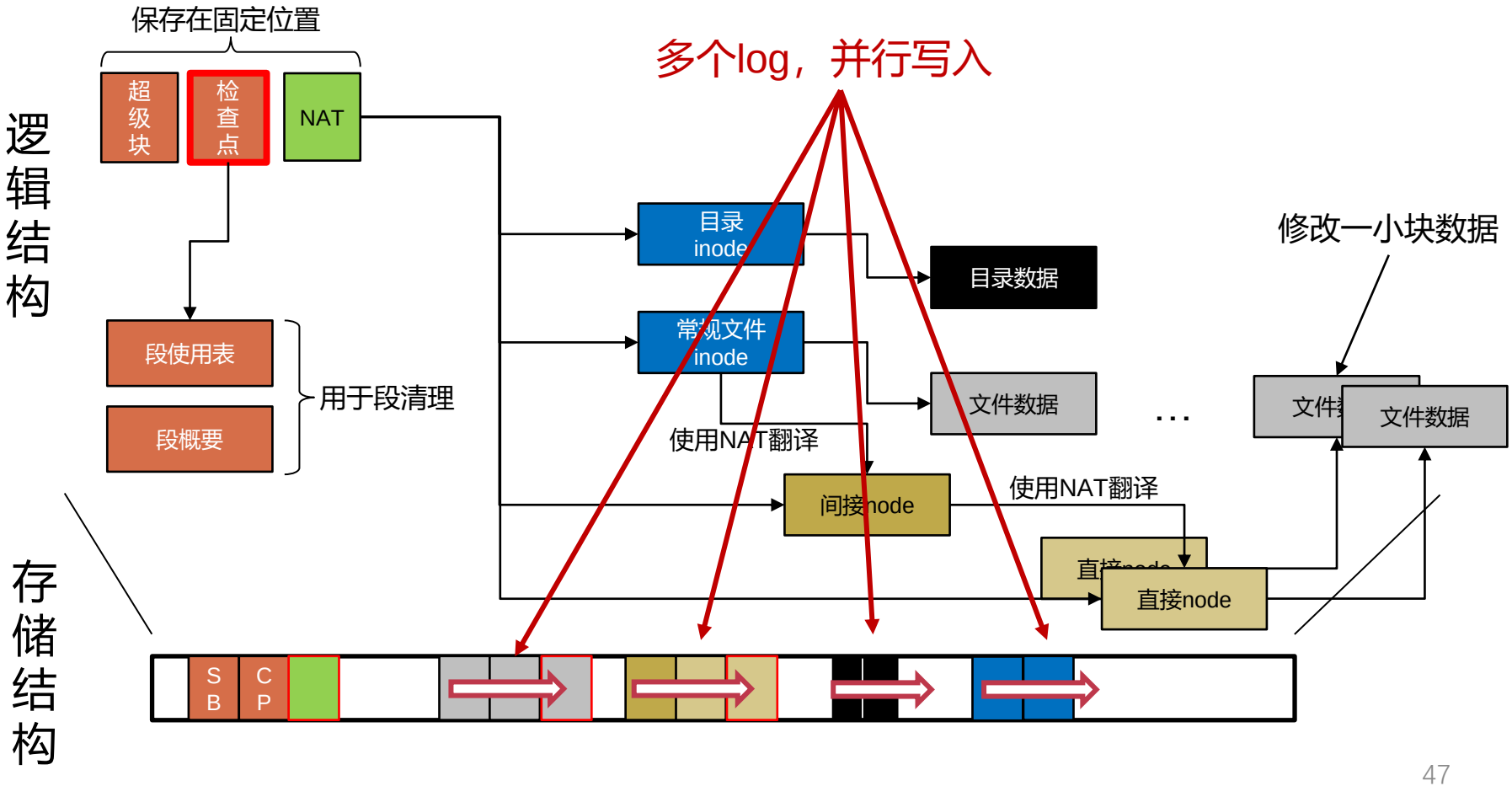
存储结构



F2FS的改进1：NAT

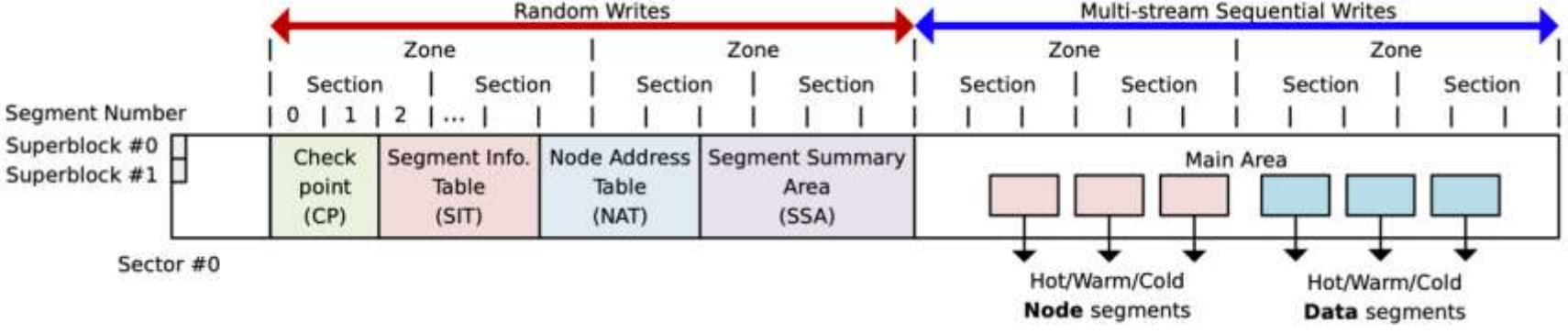


F2FS的改进2：多log并行写入



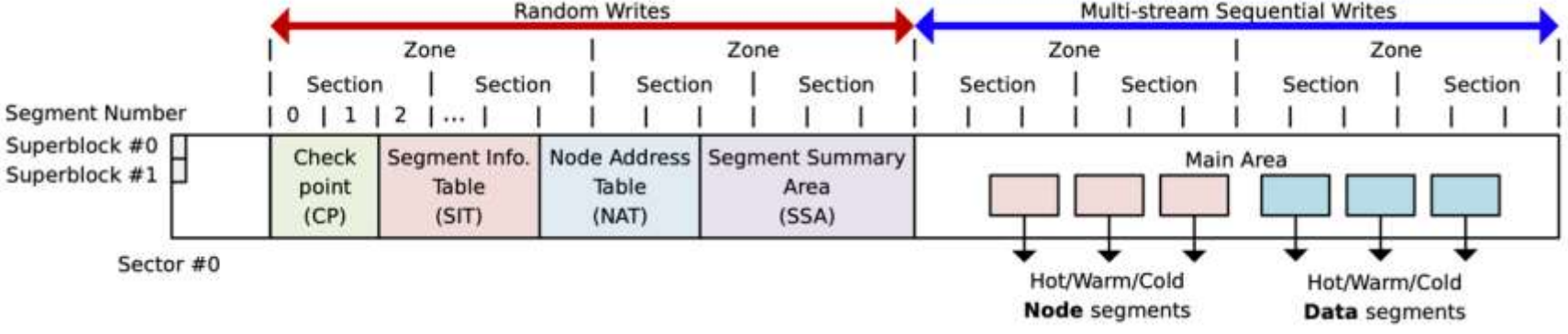
闪存友好的磁盘布局

- 组织层级
  - Block：4KB，最小的读写单位
  - Segment：2MB
  - Section：多个segment（垃圾回收/GC粒度）
  - Zone：多个section



闪存友好的磁盘布局

- 系统元数据（随机写入）
  - 存放在一起：局部性更好
  - CP：检查点
  - SIT：段信息表
  - NAT：node地址转换表
  - SSA：段概要区域
- 数据区（多Log顺序写入）
  - 区分冷/温/热数据
  - 区分文件数据（data segment）与元数据（node segment）



多Log写入

- **按热度将结构分类**
  - 每个类型和热度对应一个log
  - 默认打开6个log
  - 用户可进一步配置

- **根据硬件信息可以进一步调整**
  - 调整zone、section大小
  - 与硬件GC单元对齐等

类型	热度	对象
Node	热	目录的直接node块（包括inode块）
	温	常规文件的间接node块
	冷	间接node块
Data	热	存放目录项的数据块
	温	常规文件的数据块
	冷	被清理过程移动的数据块 用户指定的冷数据块 多媒体文件的数据块

参考资料

- 上海交通大学并行与分布式系统研究所操作系统课程
- 操作系统导论，[美] Remzi H. Arpaci-Dusseau / [美] Andrea C. Arpaci-Dusseau，人民邮电出版社，2019
- 2020 南京大学 “操作系统：设计与实现” ，蒋炎岩
- 文件系统技术内幕，大数据时代的海量数据存储之道，张书宁，电子工业出版社，2021
- 网络文章与图片