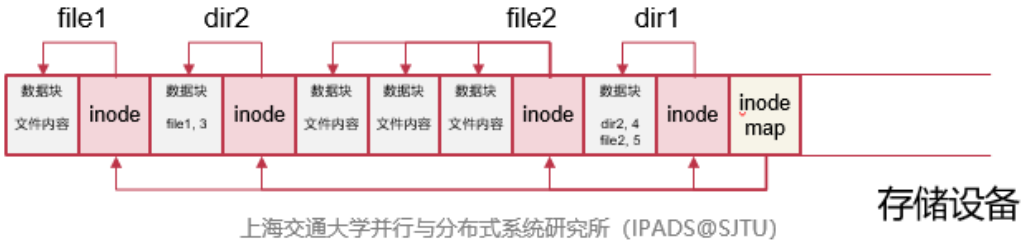
Lecture16 Log File System

1. 日志文件系统

> 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



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
  - 目录修改日志

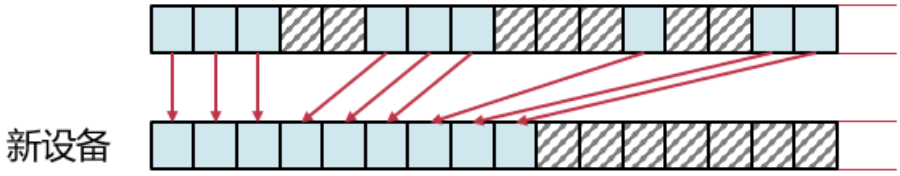
空间回收利用

- 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
- 需要重新利用前面的设备空间

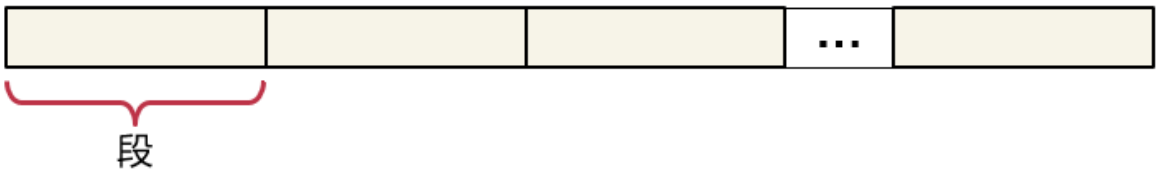
空间回收管理办法

- 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

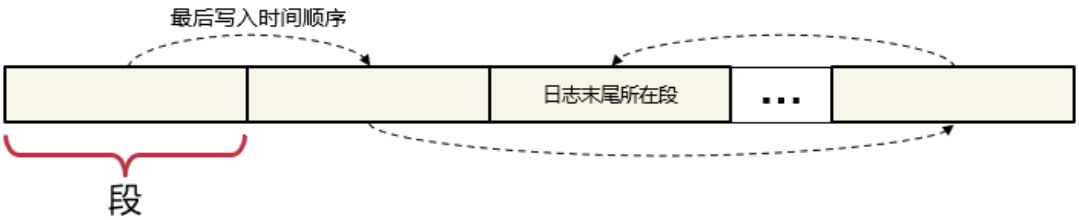


> *串联和拷贝两种方法的结合：段*



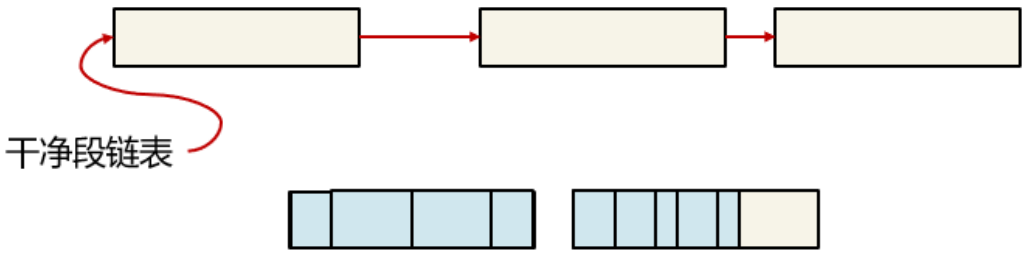
- 一个设备被拆分为定长的区域，称为段
  - 段大小需要足以发挥出顺序写的优势，512KB、1MB等
- 每段内只能顺序写入
  - 只有当段内全都是无效数据之后，才能被重新使用
- 干净段用链表维护（对应串联方法）
- **段使用表**

- 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



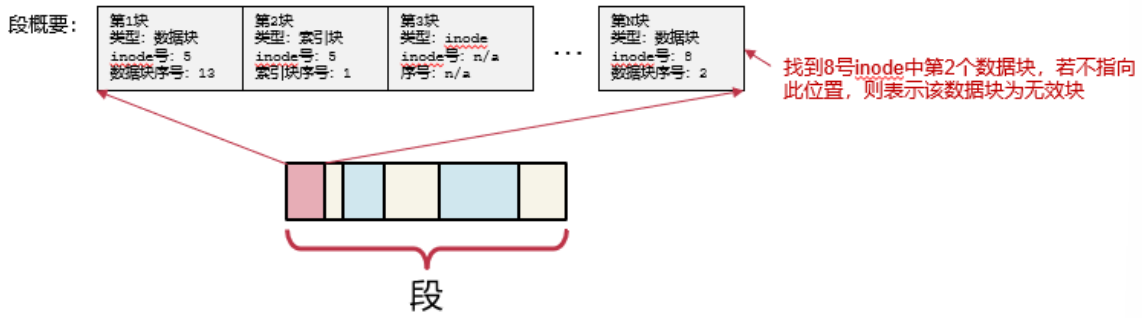
- **段清理**

1. 将一些段读入内存中准备清理 ✓
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 ✓



> 难点：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



> 和内存的swap机制很像，需要从物理地址->虚拟地址

挂载和恢复

- **方法-1：扫描所有日志，重建出整个文件系统的内存结构**

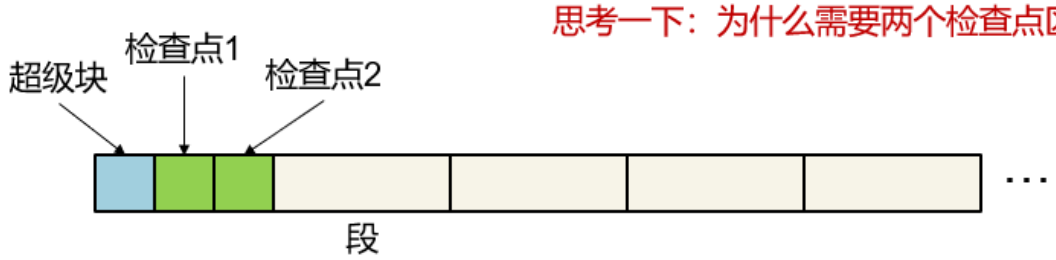
  > 缺点：大量无效数据也被扫描

- **方法-2：定期写入检查点(checkpoint)**
  - 写入前的有效数据，可以通过检查点找到
  - 只需扫描检查点之后写入的日记
  - 减少挂载/恢复时间

检查点(Checkpoint)

- **检查点内容**
  - inode map的位置（可找到所有文件的内容）
  - 段使用表
  - 当前时间
  - 最后写入的段的指针

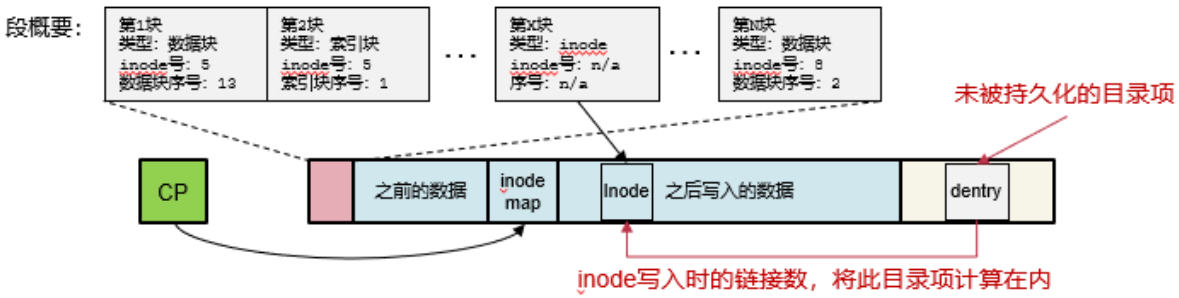
思考一下：为什么需要两个检查点区域?



> A：防止写入checkpoint时崩溃

恢复：前滚(roll-forward)

- 尽量恢复检查点后写入的数据
- 通过段概要里面的新inode，恢复新的inode
  - 其inode中的数据块会被自动恢复
- 未被inode“认领”的数据块会被删除



- 段概要无法保证inode链接数的一致性
  - e.g. inode被持久化，但是指向其的目录项未被持久化

恢复：目录修改日志

- 目录修改日志
  - 记录了每个目录操作的信息
    - create, link, rename, unlink
  - 以及操作的具体信息
    - 目录项位置，内容，inode的链接数
- 目录修改日志的持久化在目录修改之前
  - 恢复时根据目录修改日志保证inode的链接数是一致的