[18-新型存储设备的文件系统 1](#_Toc105249460)

[review 1](#_Toc105249461)

[瓦式磁盘 4](#_Toc105249462)

[NVM 非易失性内存 12](#_Toc105249463)

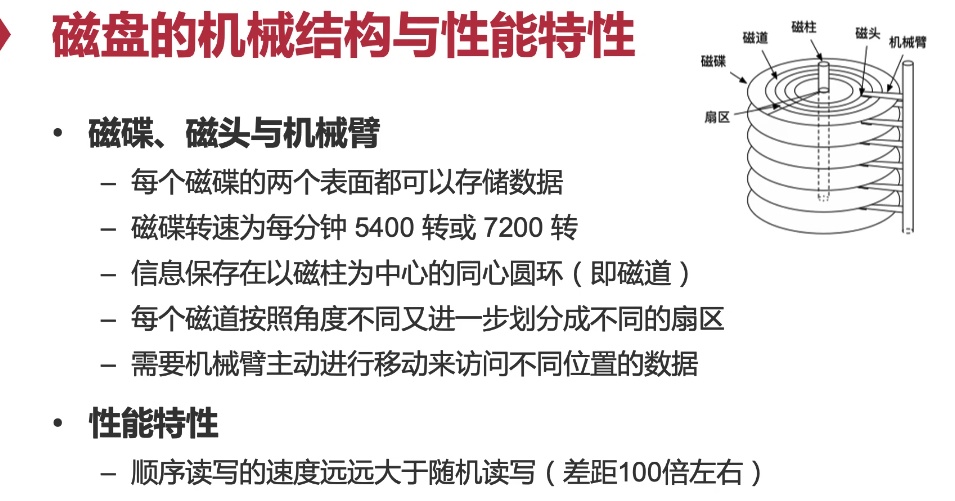
[COW 16](#_Toc105249464)

[非易失性内存文件系统 16](#_Toc105249465)

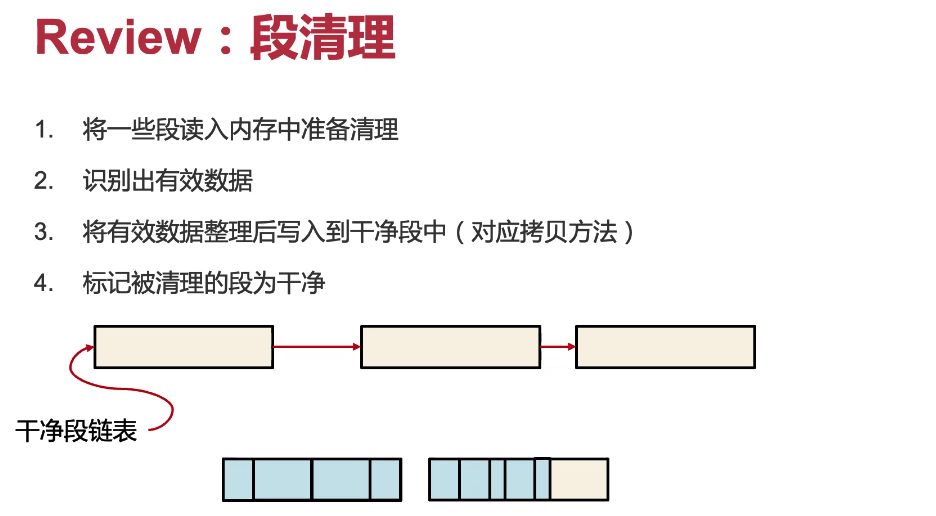
# 18-新型存储设备的文件系统

## review

上节课我们讲的主要是LFS，对于磁盘来说，读性能和写性能是差不多的，因为磁盘的性能瓶颈取决于机械部件，读写本身磁头是读还是写其实并不重要。读写性能是对称的，flash就是不对称的。寿命不用考虑太多，但磁盘的问题在于顺序读写的的差距。



既然有了这个现实之后我们想如何利用这个特性，尽可能往后写。尽可能减少对磁盘的读，磁盘只剩了一个功能：persistence。这样就可以把读和写解耦掉，读用memory搞定。为什么写不用memory搞定呢？没有持久化自然不行。



段太短：顺序写入优势无法体现。跳转太频繁。

太长：段清理时间太长。

有了段清理，等了磁盘空闲，就进行段清理。空闲的段通过指针的方式连接在一起。逻辑上连续的区域。物理上只有段内连在一起。

如果有多个磁盘的话，RAID可不可以写完一个磁盘，有一个磁盘当前写（所有写操作都写这个磁盘），第一个磁盘写完写第二个磁盘，第一个去做清理，这样是不是就可以做一个和raid不一样的磁盘阵列。

一个磁盘性能不够——两个磁盘同时写——有原子性问题



硬件发展相比软件慢很多。block需要整个擦除掉才能再写——物理实现相关。有些时候不完全这样，0不能变成1（要变只能抹掉一层），1可以变成0——只能单向改变

我能不能每次写入的时候先写，第二次覆盖能不能做到只把1变成0（有研究者在做这方面工作，通过一些编码等）

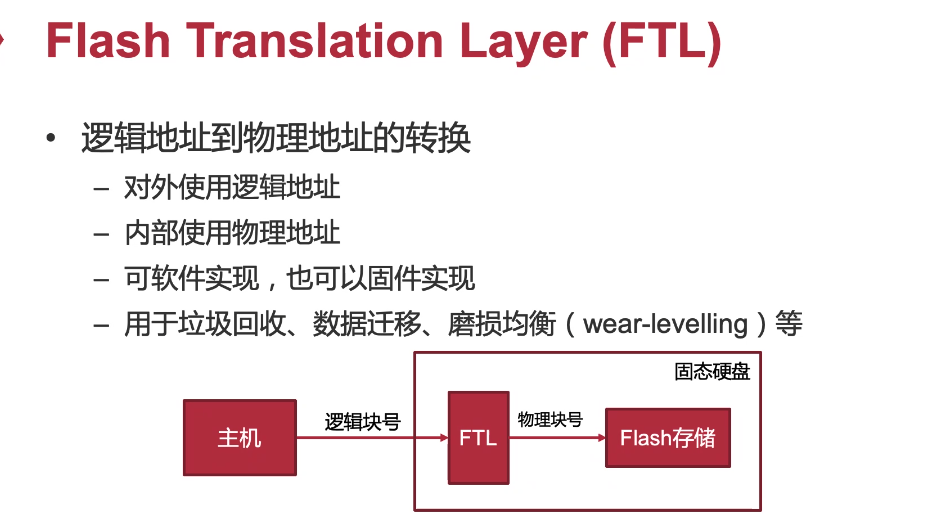
如果有把0变成1的，读的时候再通过反编码的方式。

**软件要了解硬件的特性——才能做优化——打破抽象，看到底层**

随机访问顺序访问十倍以内（慢1~2倍）

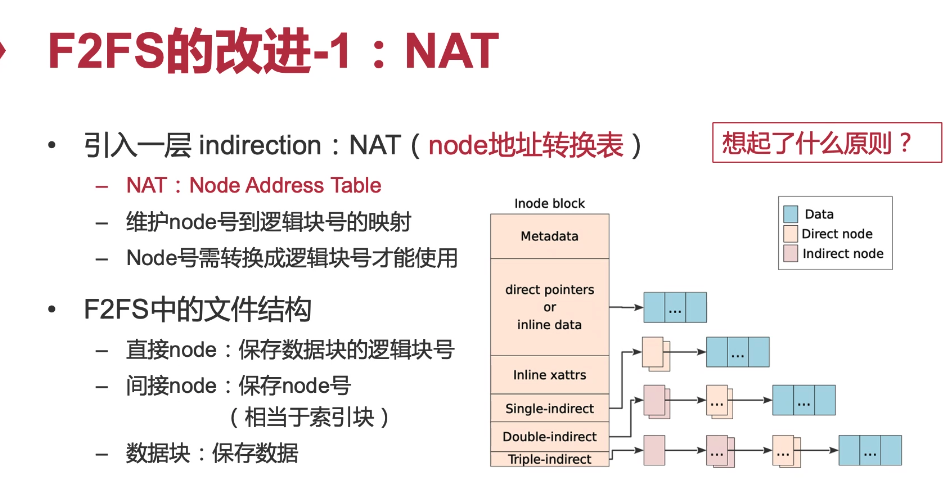
更多的时候来自于ssd内部也有cache，可以做prefetch。随机写pattern不固定。

多通道ssd和磁盘完全不一样——有多个可以同时做读写的方法（磁盘虽然有多个磁面，大部分只用一个磁头读写）

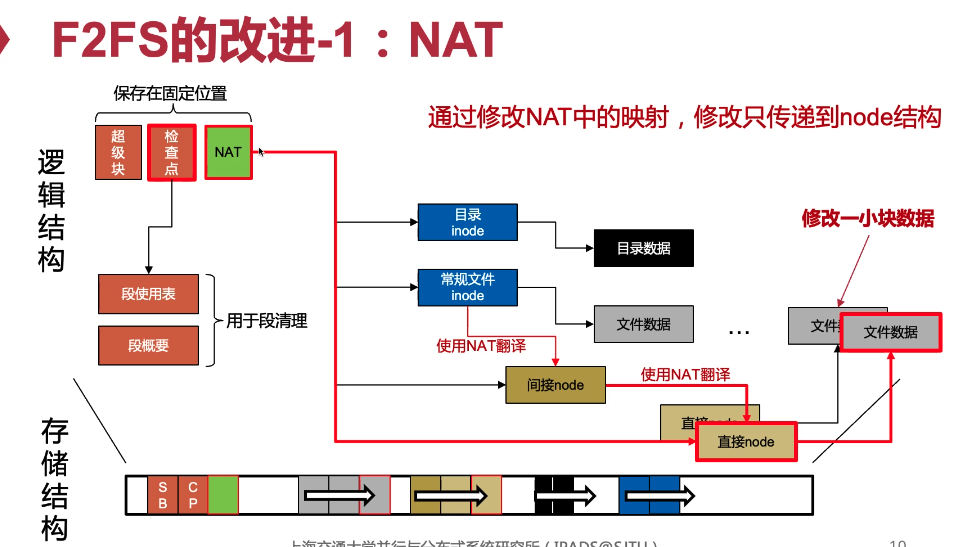


整个ssd的磨损均衡就比较容易做。

然后我们发现说：既然每次写都要擦掉一大块——pattern跟LFS很像。



直接用LFS有写放大的问题，为了解决这个问题：用NAT，通过node address



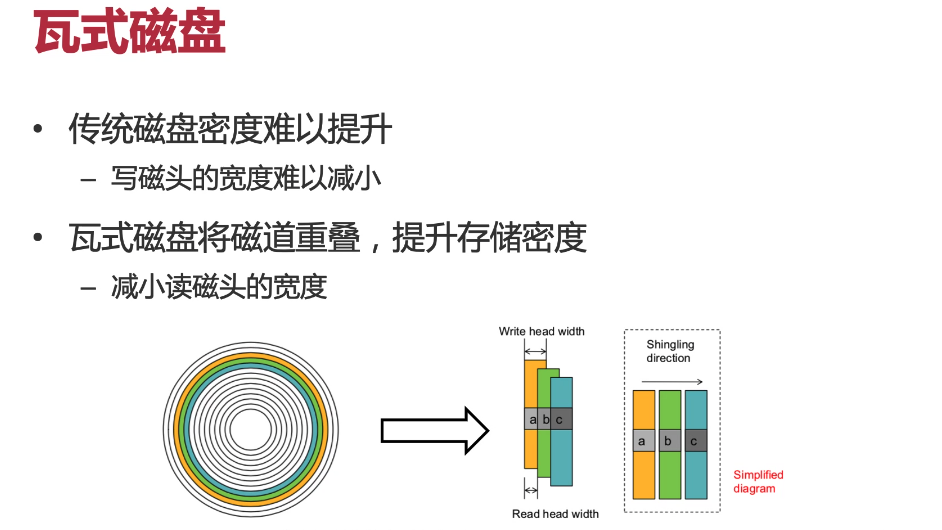
大家要想哪些是硬件做的（FTL），哪些是软件做的（NAT）

FTL：硬件做更合适。买的1个T ssd有可能实际上会有1.1T，剩下的空间来做缓存，来做清理等等。这里有一个产品的边界在这里。

NAT因为你有inode，你这个FS长成这个样子，你不用F2FS，你可以用FAT，Ext等等，磁盘制造商不管。所以在inode里面才要引入NAT这样一张表，这是软件改不是硬件。

同时正是硬件有并发优势：所以写入的时候可以多log并行。

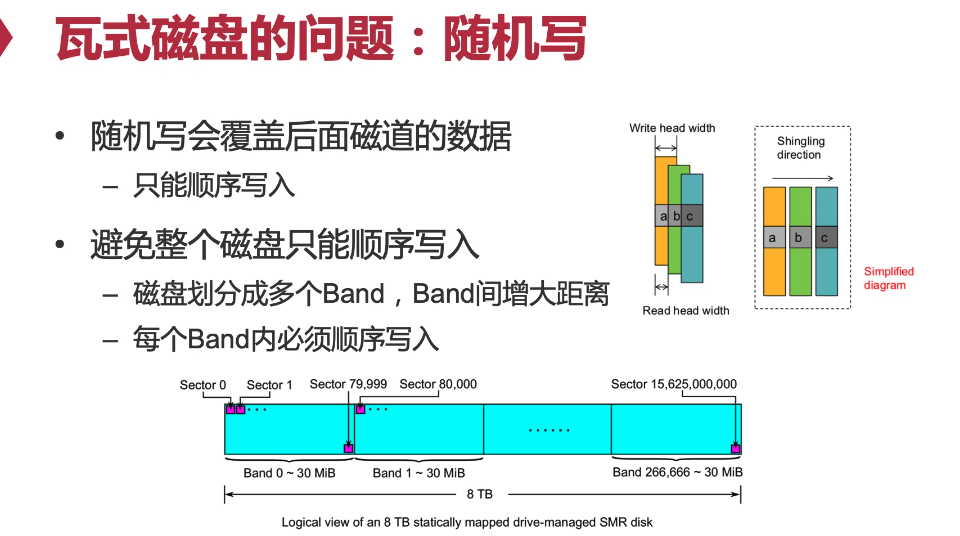
## 瓦式磁盘



**读写用到的磁头宽度是不一样的**

顺序写没问题：先写一圈再写一圈。

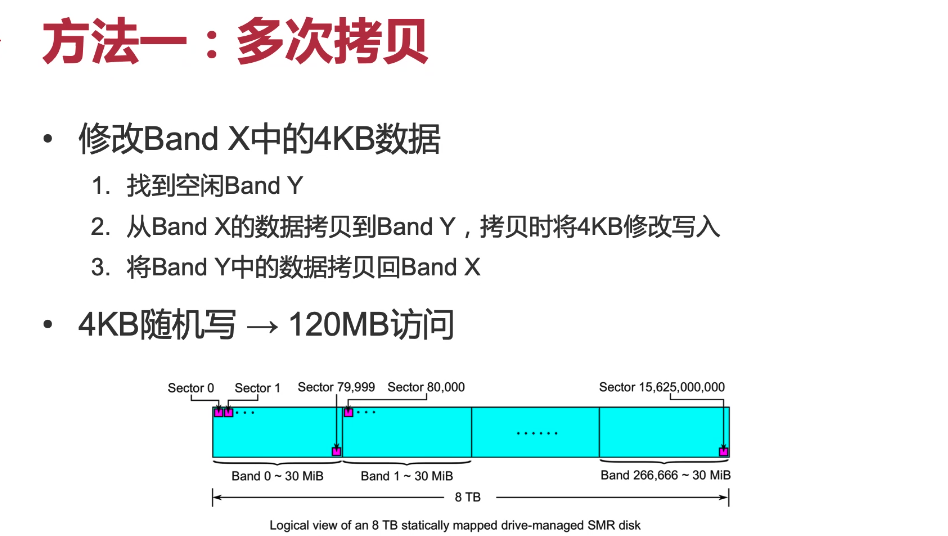
随机写就没办法了。



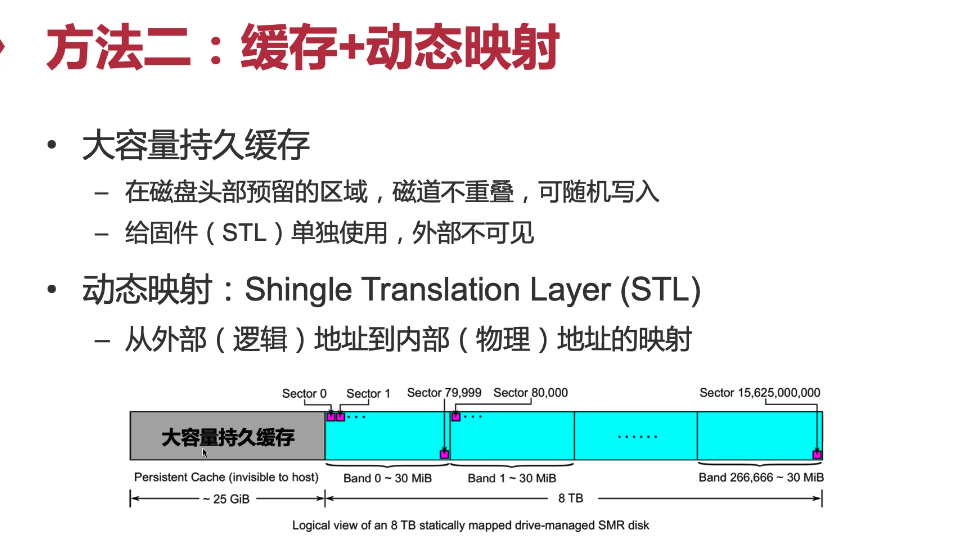
每个块之间不重叠的，每一块之内必须顺序写。

Q：要改band中的数据怎么办？

A：

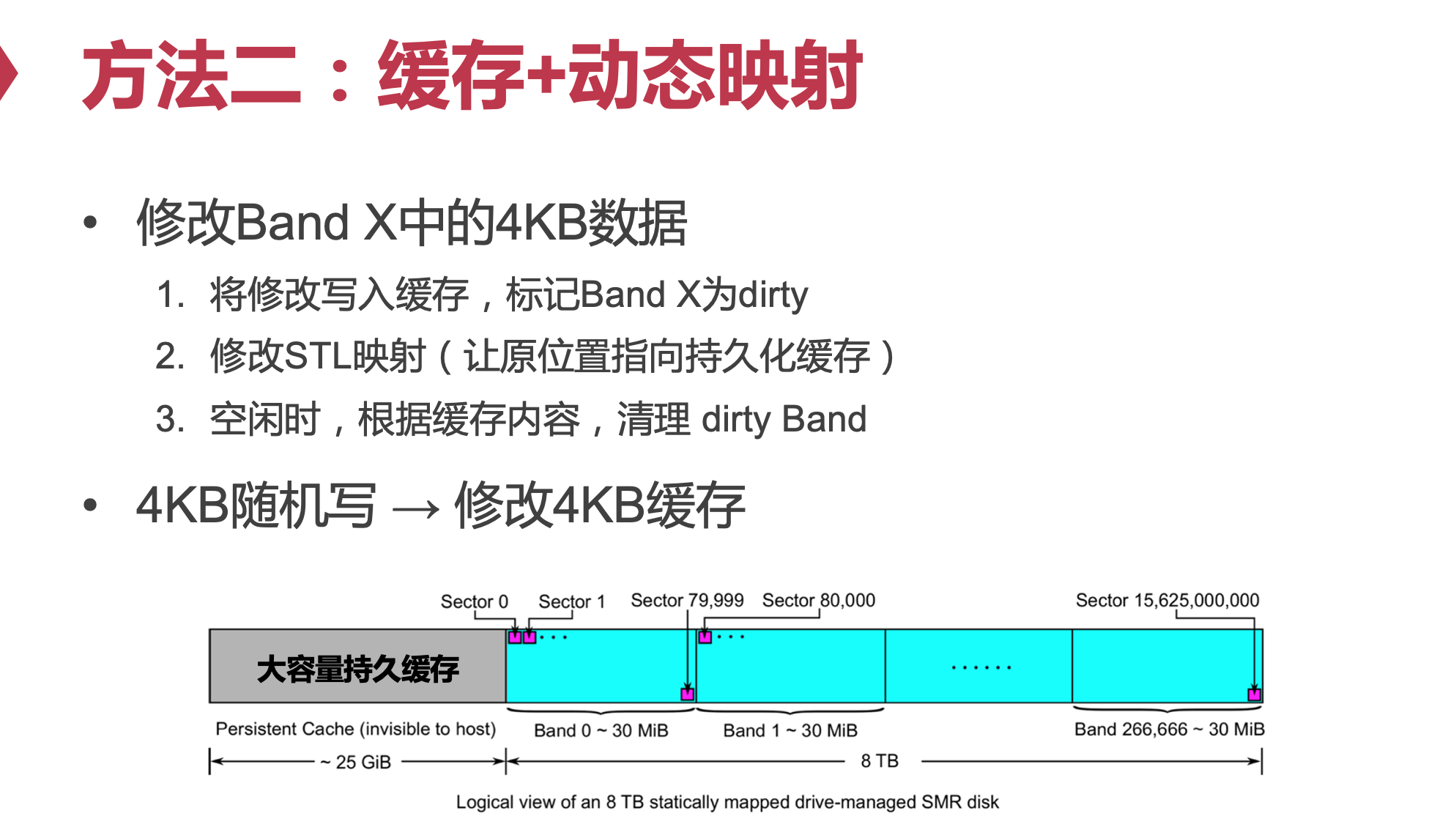


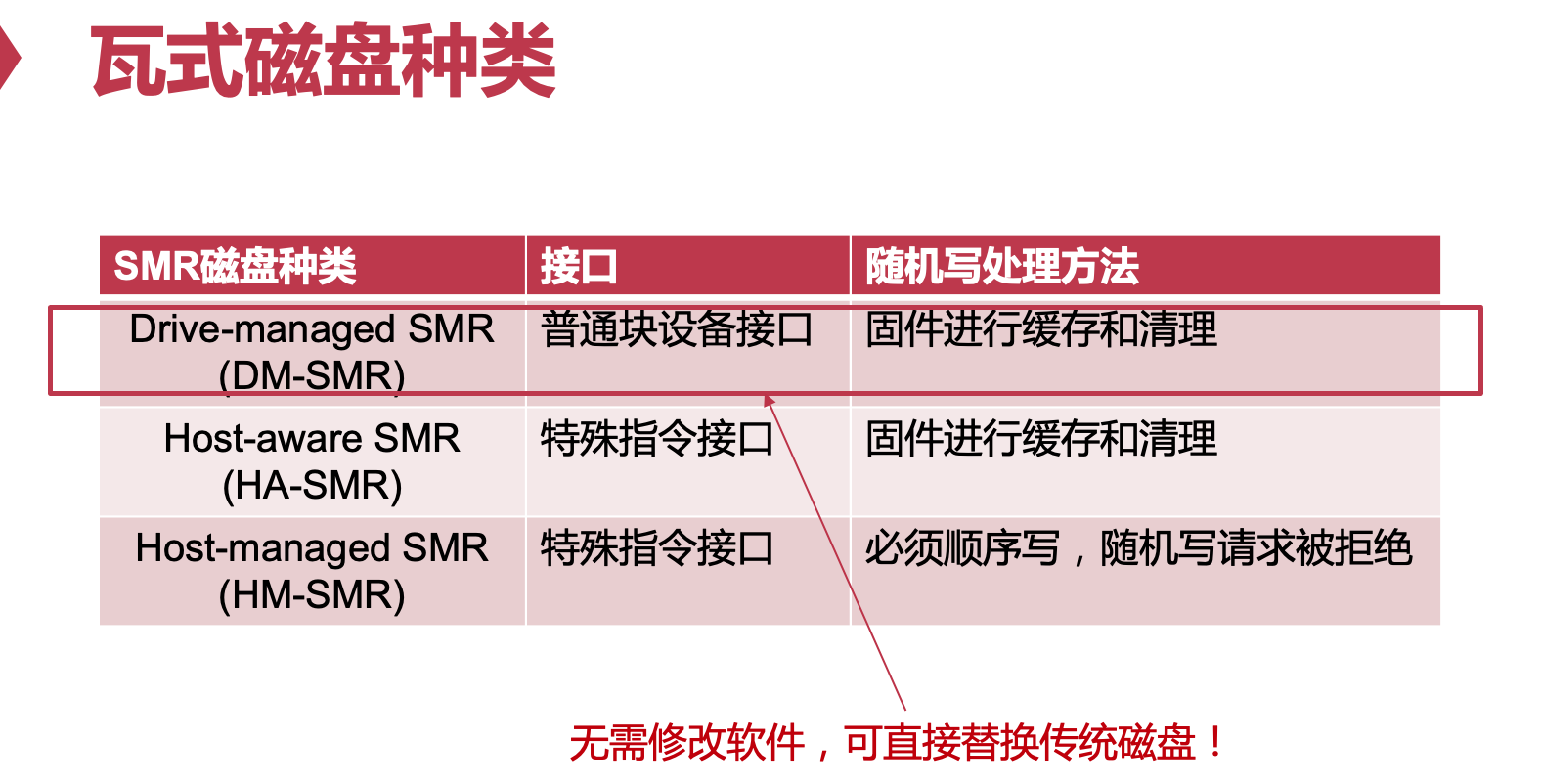
band band之间可以不用考虑覆盖的问题。

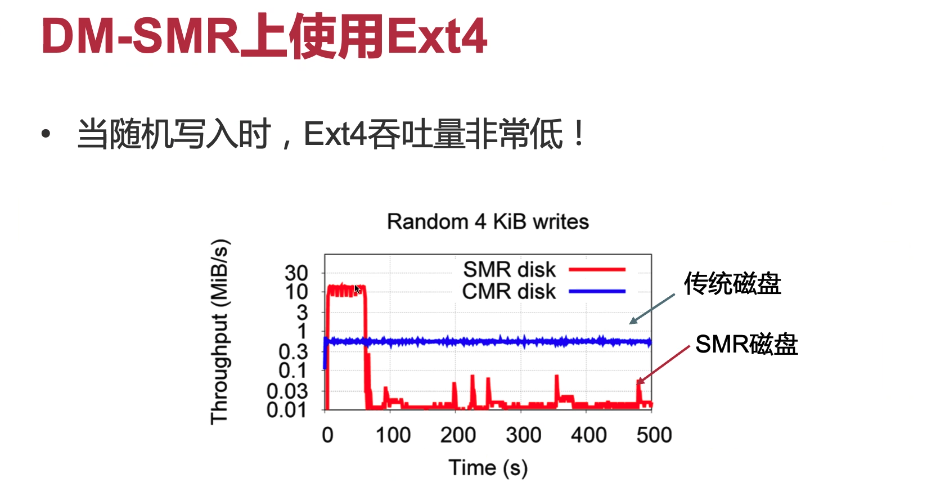


磁盘头部放一个磁道不重叠的传统的磁盘，写的时候先写到这个区域，然后再做一个映射。

先写到这个区域里：修改STL表格。磁盘比较空的时候，把缓存里的数据一点一点清理到后面。



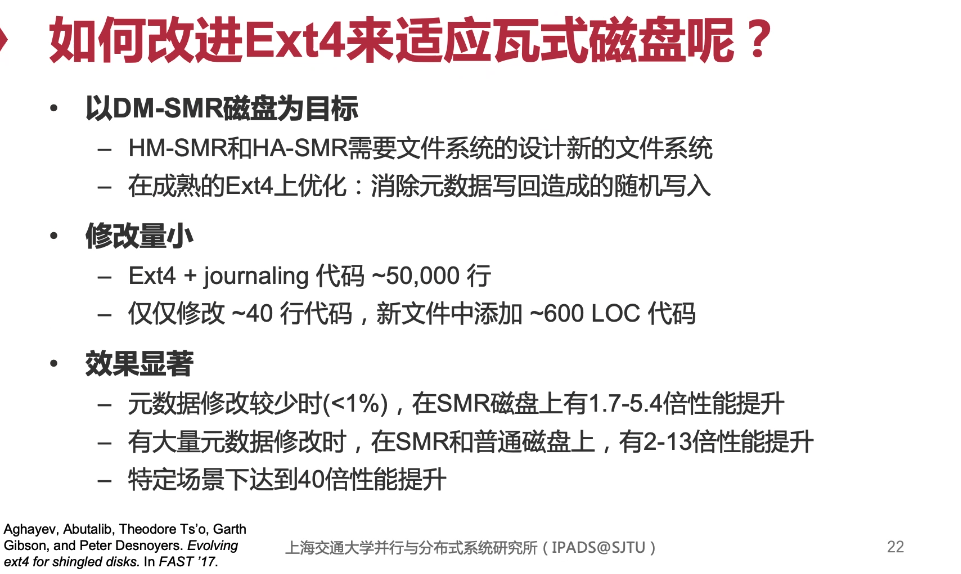




一开始比传统磁盘高一个数量级

一旦过了一定时间马上掉下来了。

一开始全部写在大容量持久缓存（不用真的随机，只要记录映射就可以了）一旦写完了之后开始进入band区域，就要以30M，30M这样的粒度去写



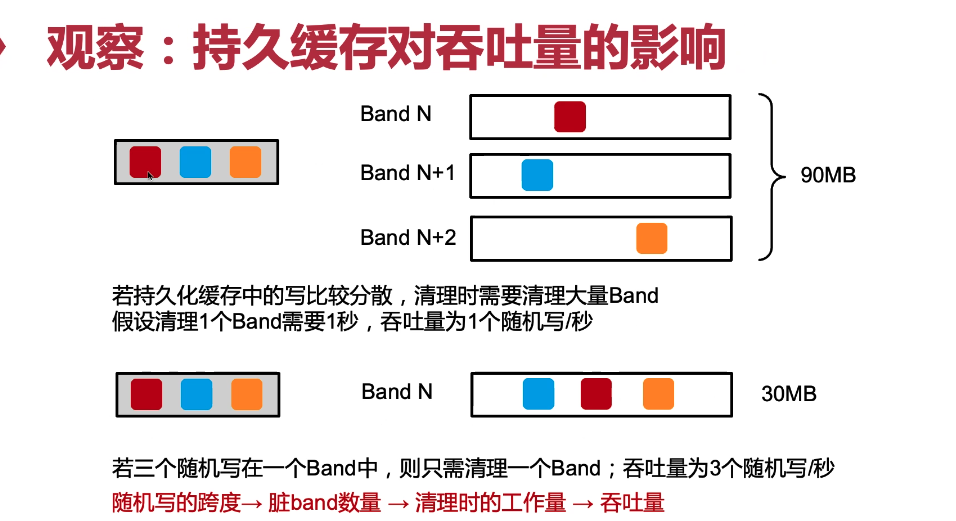
第一种思路：设计一个新的fs

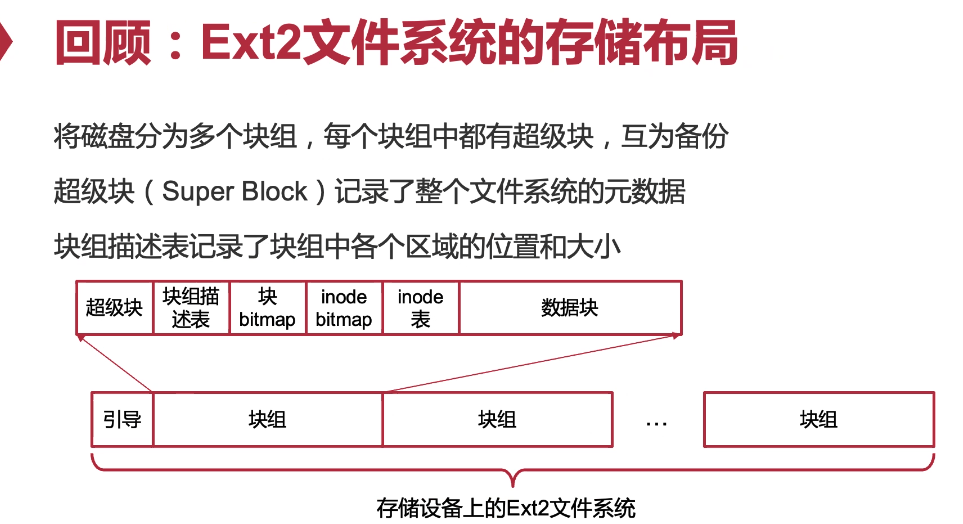
第二种思路：fs最好还是不要变来变去，ext4已经很成熟了，bug都有人找到了

> 开源社区：能不能复用已有的东西解决问题，生态发展是一个成长的过程，新东西谁都说不准到底行不行

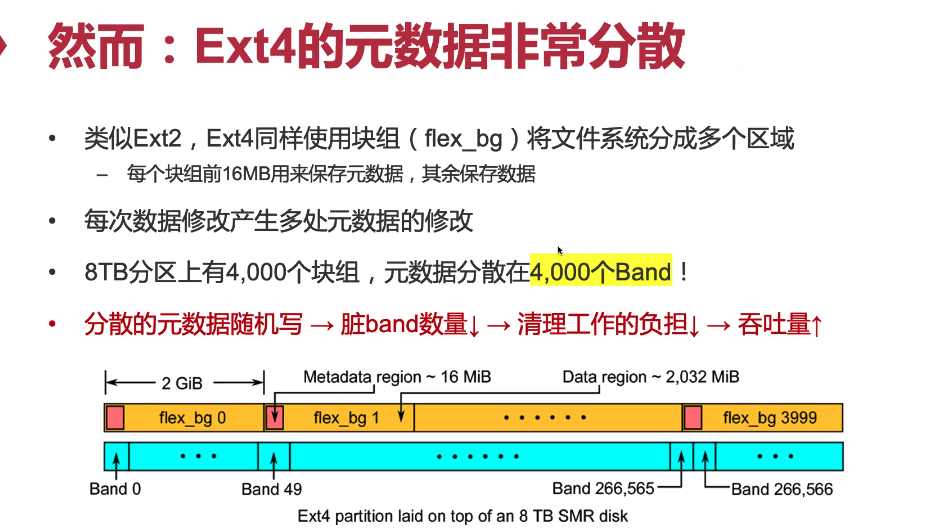
一般在商用平台上能提升数倍提升就非常好了

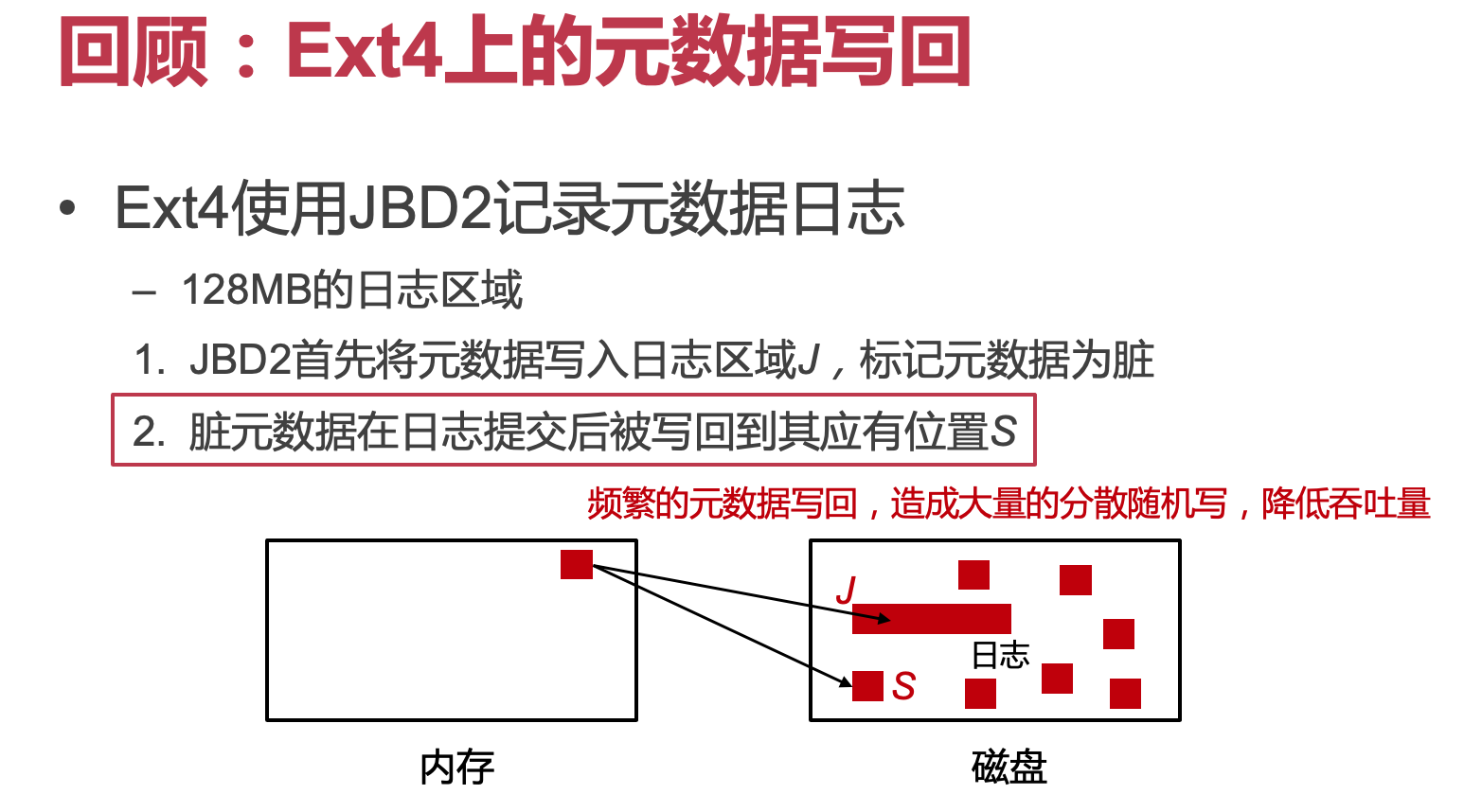
\*\*以下笔记不全，need录像\*\*



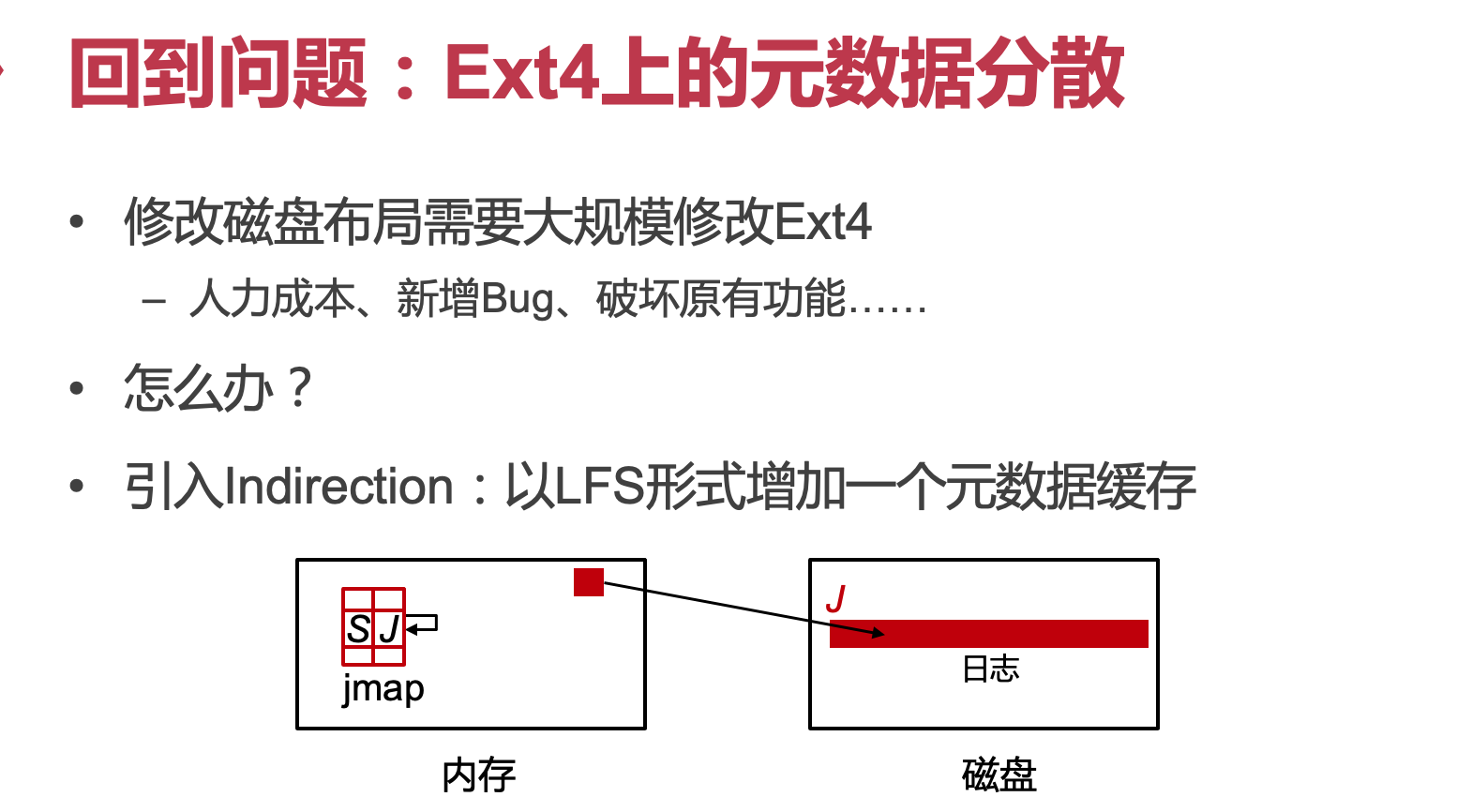


块组：元数据和数据的距离较短

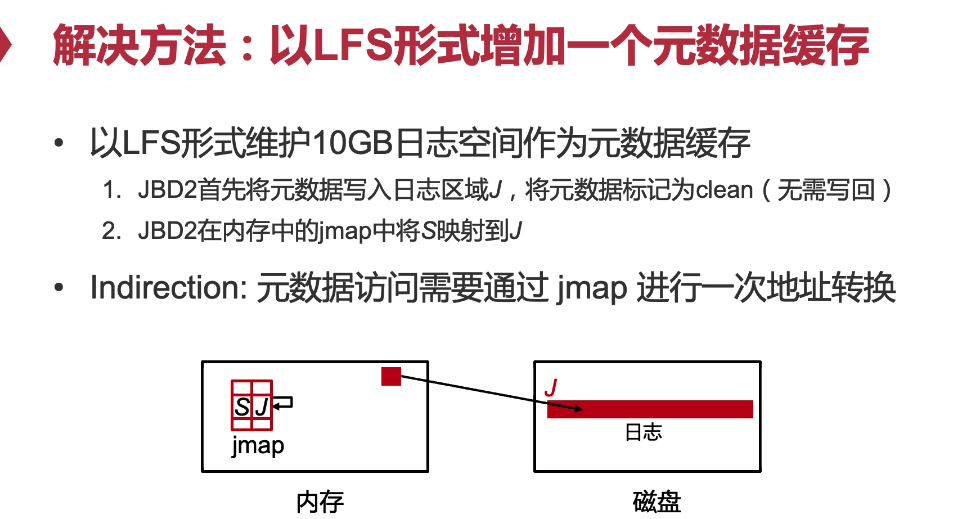




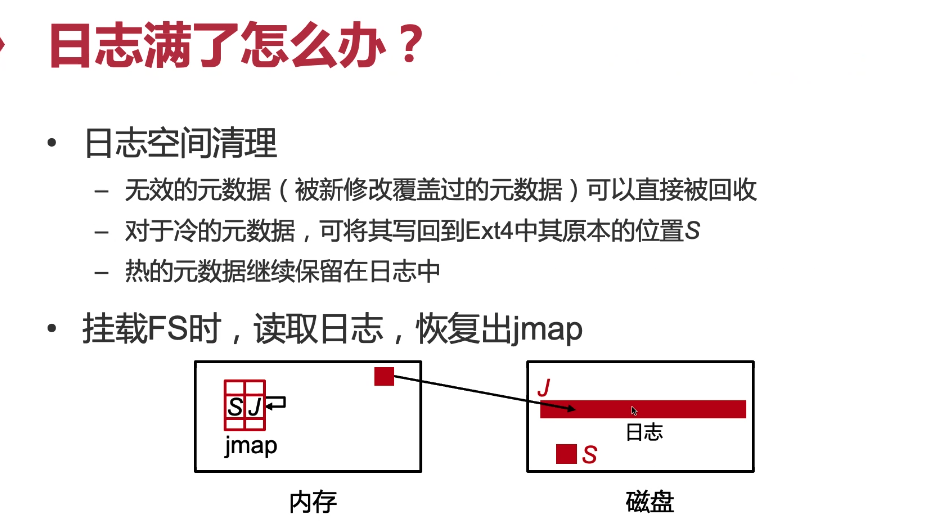
把日志变大就行了：把日志增加变长变大：把S写回的时间间隔变长一点



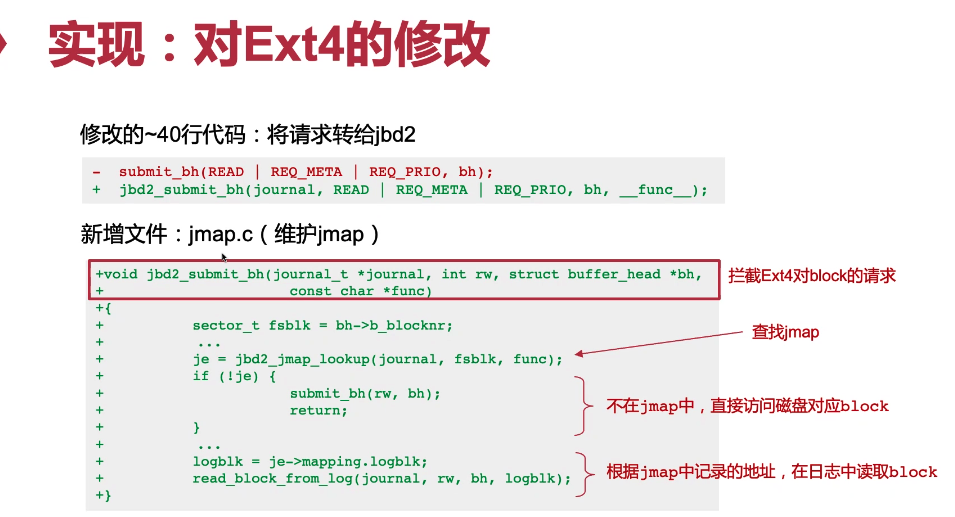
只要把jmap映射好，要访问的时候只要知道jmap在什么地方就行了。

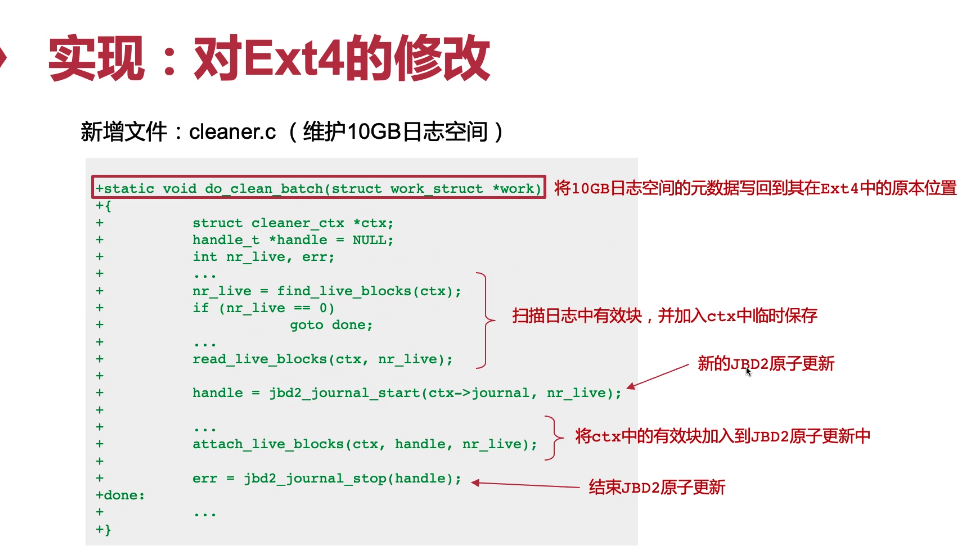


每次挂载FS的时候读取日志，恢复出jmap



冷热信息记在内存里或者磁盘上，crash这件事不是经常发生，一旦发生冷热数据有那么重要吗？所以这个统计类信息适合放在memory里，所以放在jmap里





\*\*以下笔记全\*\*

## NVM 非易失性内存

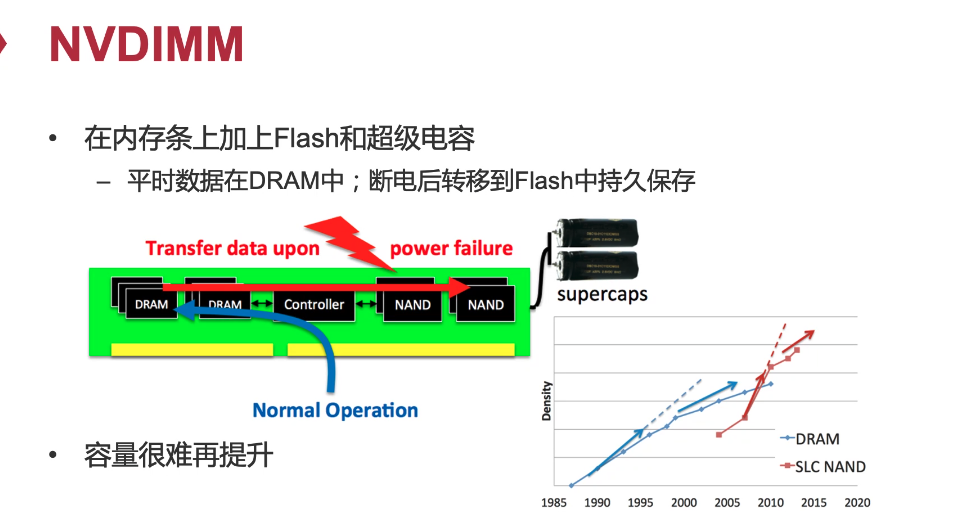
**会对计算机系统产生颠覆性影响的特性**

传统的文件接口现在变成了内存的接口

以前不可能像访问内存去访问磁盘

为什么内存数据不能持久化：掉电数据就没了

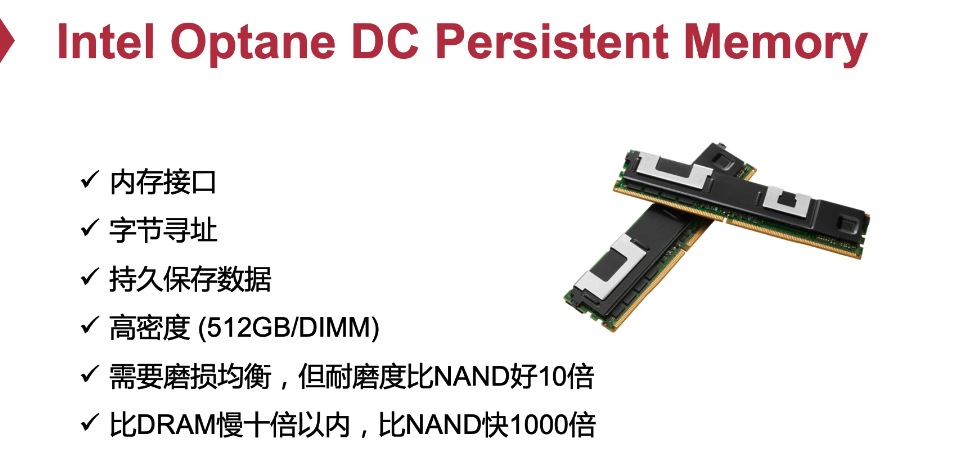
之前也有人做过尝试



大电容，一旦断电，立刻把内存里的数据写到flash

controller把数据写到NAND这个flash里面去，写完之后电容就没电了

cons：依然是dram，容量很难提升



1. 内存接口

2. 内存寻址

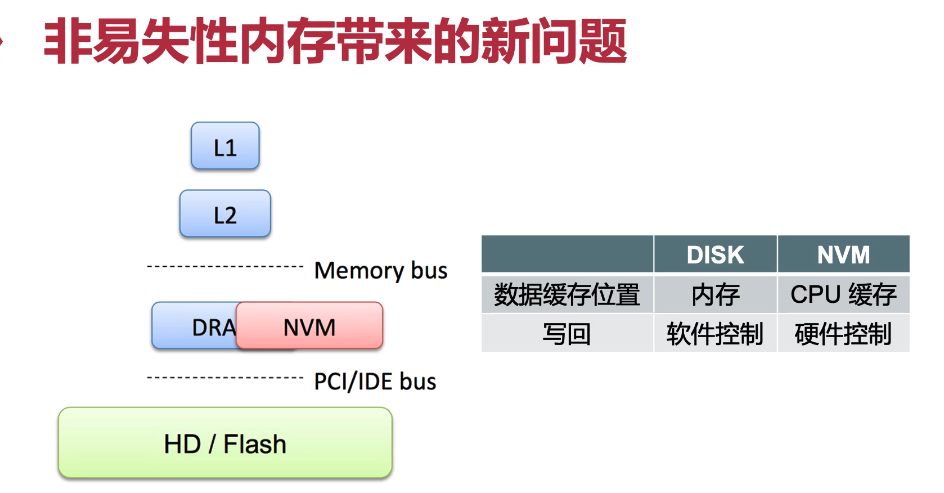
内存和flash之间差4个数量级

现在在flash和内存间加了一层，有了这个之后带来新问题

磁盘的话数据缓存在内存里，不会落盘，需要通过软件在某些时间写到磁盘上

本来内存到磁盘没写回就丢失

现在是CPU cache的数据断电之后还是会丢——加电容，断电就把cpu cache写到nvm里面去

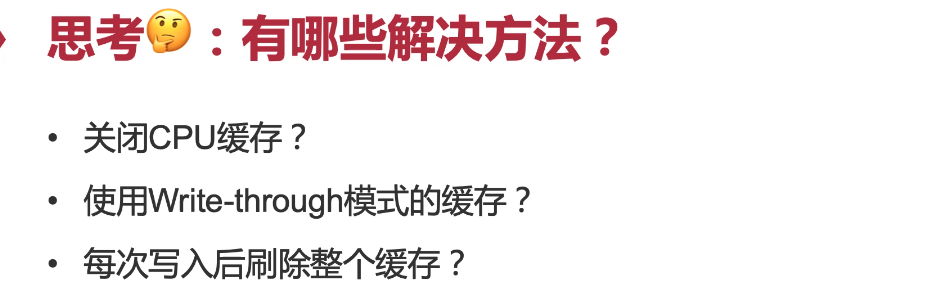


问题上移，本来是memory和磁盘的一致性



写两个数据A，B写到nvm就会变成B，A



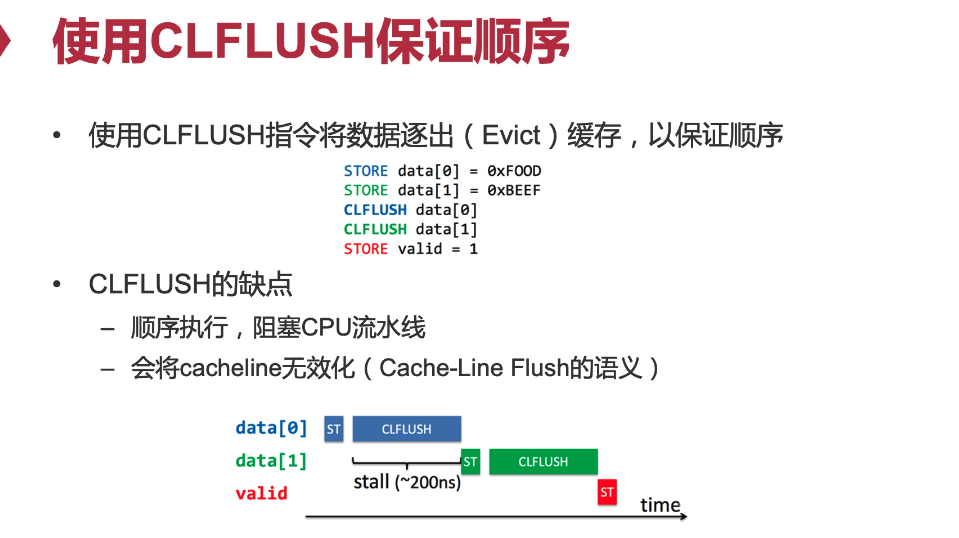


关闭CPU缓存：性能下降，无法接受

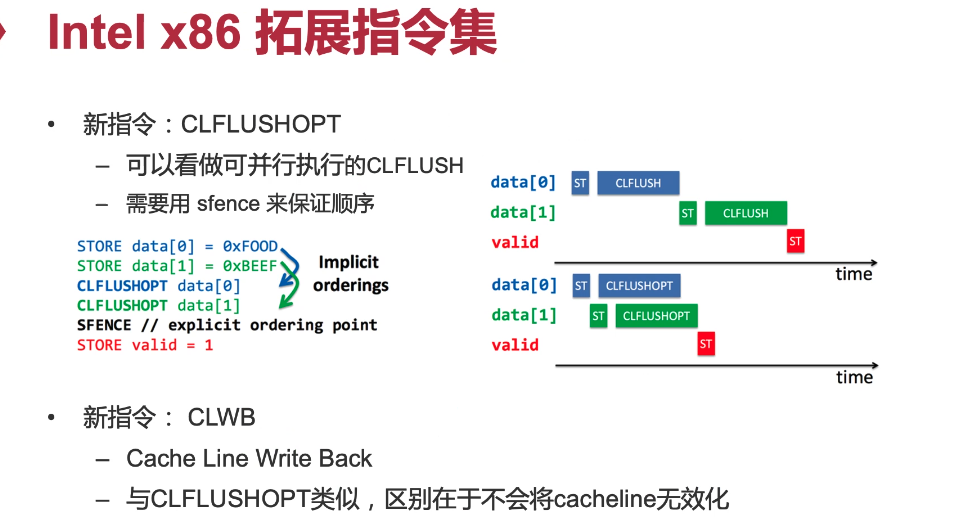
写的时候写穿，读的时候cache——性能也会变差

flush cache就跟flush disk一样：落盘——cache数据flush到memory里面去

有什么区别：flush cache是软件可控的



store只能在CLFLUSH后执行



clflushopt：可以跟store同时运行，在pipeline里也可以同时运行

sfence：barrier

见上右图，快了差不多一倍

还有个缺点：依然是flush语义，写完之后cacheline就没用了

所以引入一个新的指令：clwb，不会将cacheline invalid

### COW



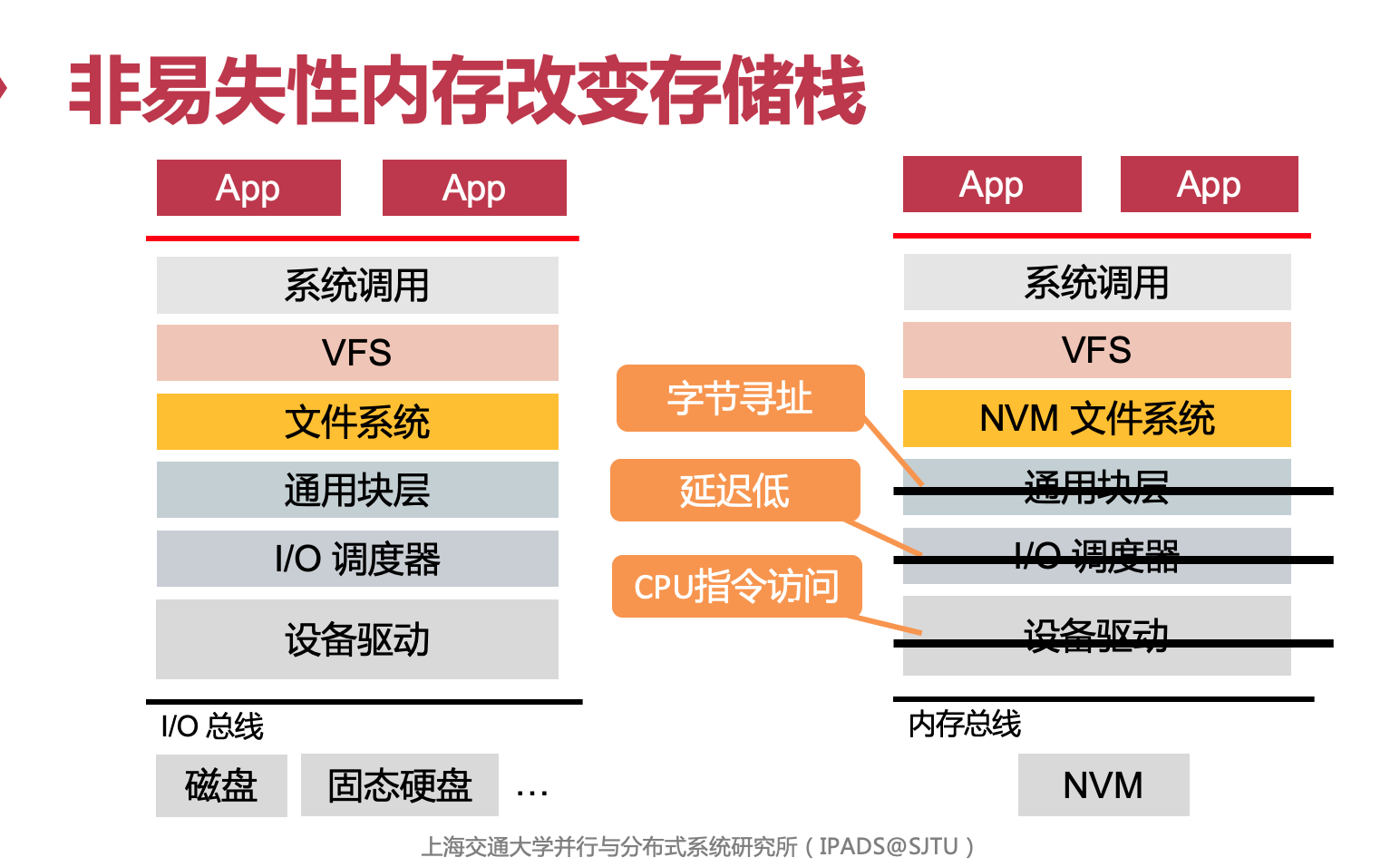
为了保证原子性：R必须放在Z和Y之后

粒度就可以变的很细：原来是4k，现在是byte级别

## 非易失性内存文件系统

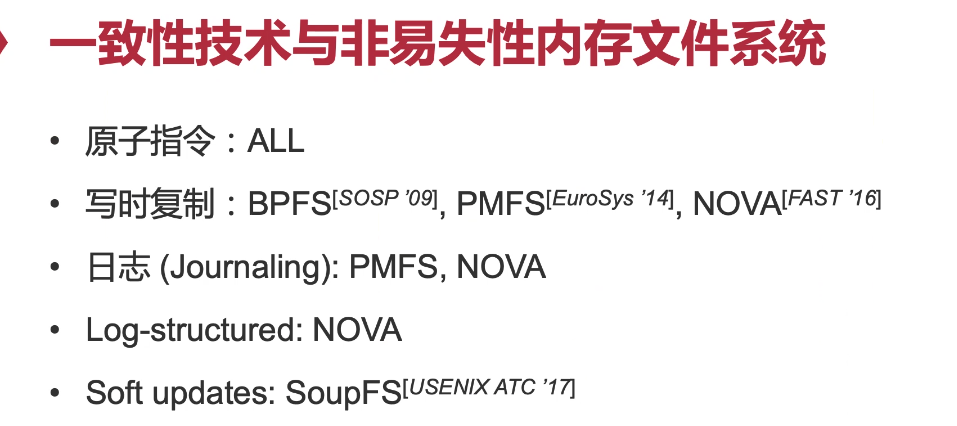
明明是内存，为什么提供文件接口

除了load store还提供read write



VFS对上提供vnode，都是内存中的一些数据结构，不管底层是FAT还是ext4，对上都是vnode。还没讲虚拟化：虚拟机里面有这么多，虚拟机外面也有这么多，6+6就是12层

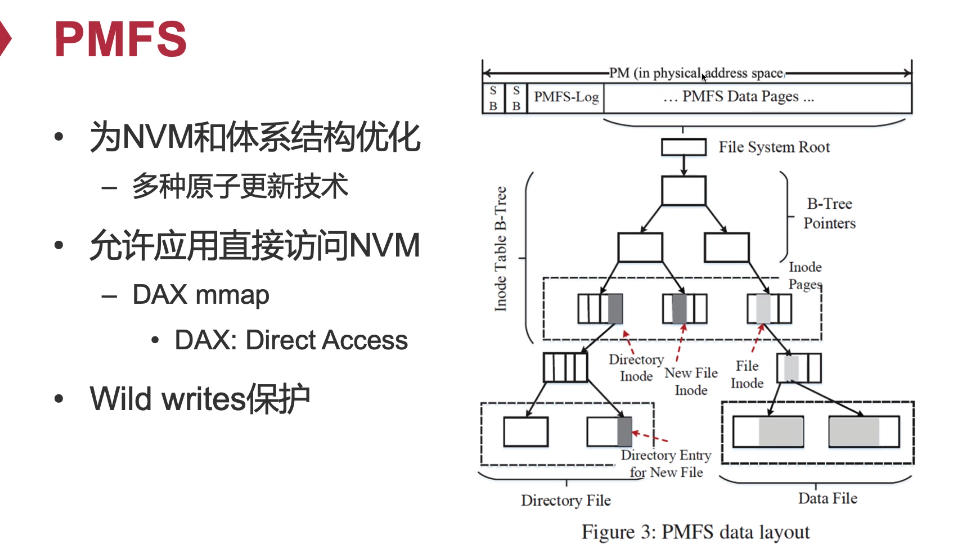
如果底层是NVM，总线就变了，IO总线就变成内存总线，不用调度——足够快，不用驱动，直接CPU指令访问。



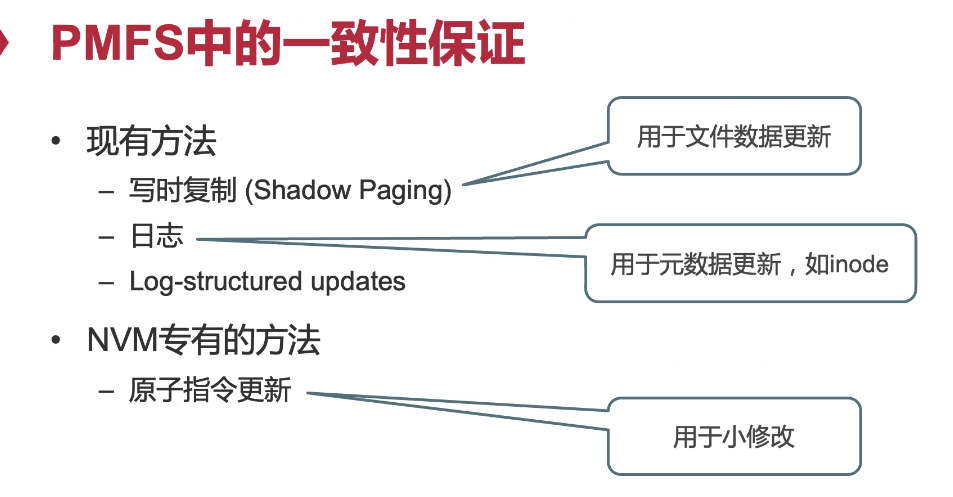
有些使用日志保证原子性，有些使用log structured，有些使用soft update。由于NVM粒度更细，用原子指令也可以保证上层的crash consistency

PMFS：综合前面多种原子指令和体系结构操作

### PMFS

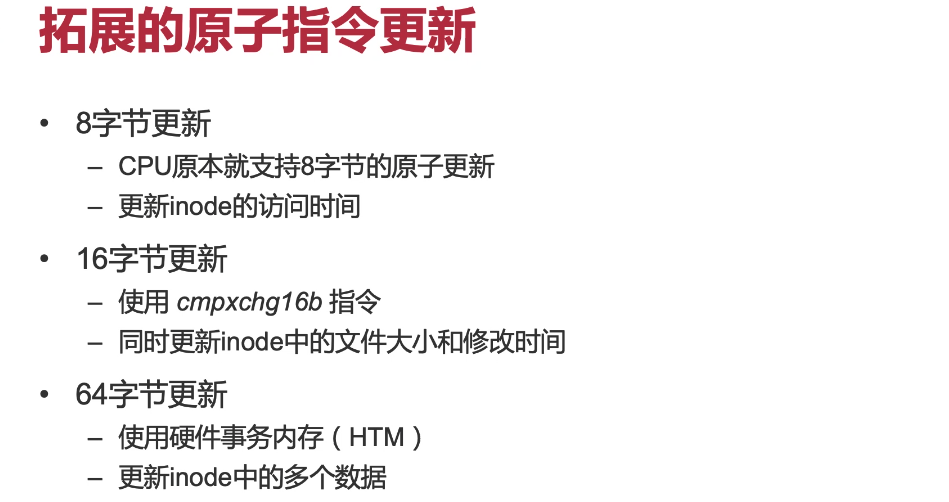


应用可以通过mmap方式访问不一定要read write



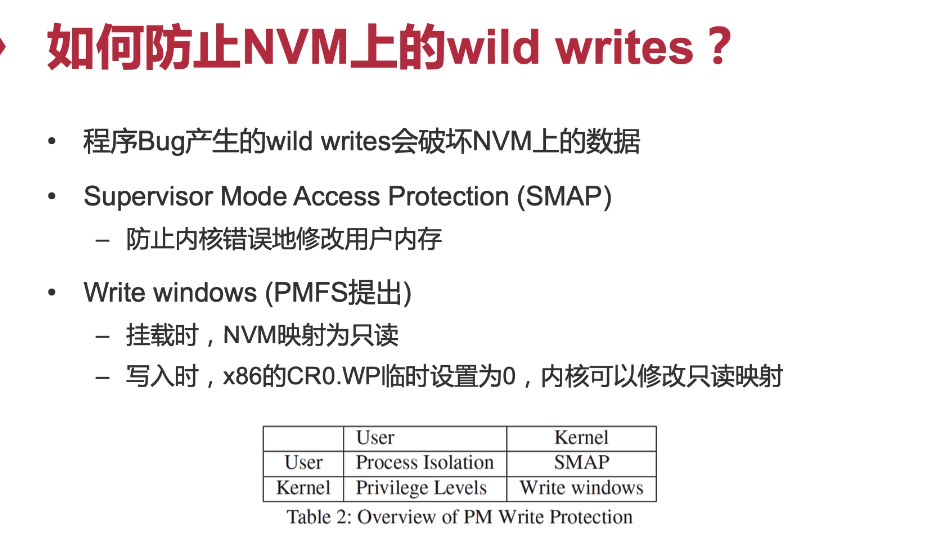
内存这个级别不用考虑随机这个问题

但它不能保证一致性，所以还是多个写一起写



原来用read before write：先把4k读出来，更新完再写回去

16字节更新：trick，它完全可以不这么做，比如用日志之类的方法。但因为nvm粒度细，所以它可以用trick，比如改变一下inode布局。从而避免日志。

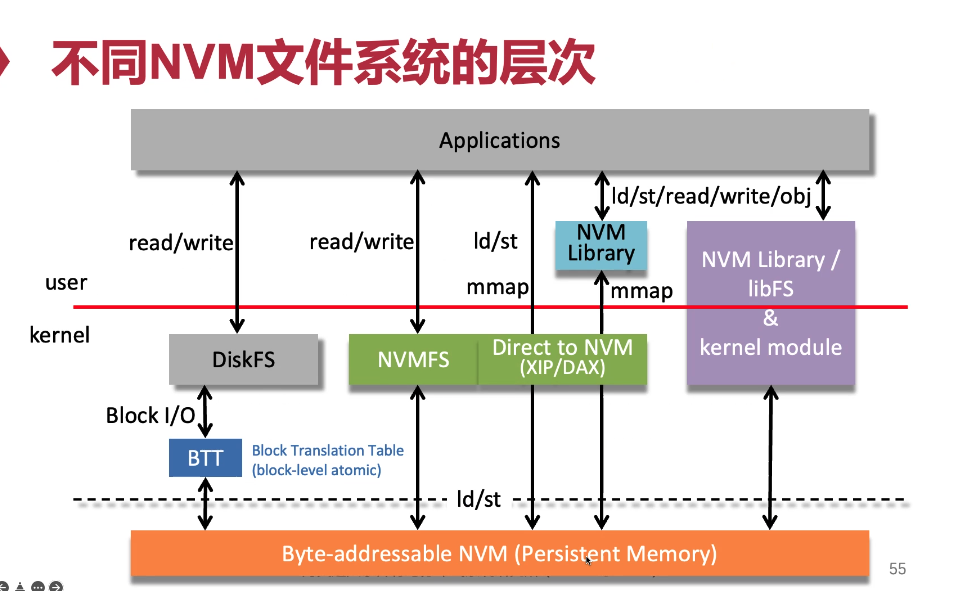


内存不该写的区域写了，以前为什么没有这个问题呢？

内核也有可能跑飞了。

1. smap防止内核错误的修改用户态内存。

2. 标WP，只读。内核每次写NVM的时候把这个开关打开，写完之后再关上。为了写read only不出错，就把write protection关掉。



最早就把NVM当做block设备来用，disk fs还是传统的fs，还是4k 4k的写。

后来产生了NVMFS，对下load store，对上也提供文件系统接口

DAX：直接mmap上去，上层可以直接读写内存

上层还想read write（照顾旧的应用）：变成一个用户态library。

最后：能不能和kernel整合，对上不仅提供read write，能不能进一步提供object抽象？