



# Solid State Drives

---

中国科学院大学计算机与控制学院

中国科学院计算技术研究所

2021-11-29





# 内容提要

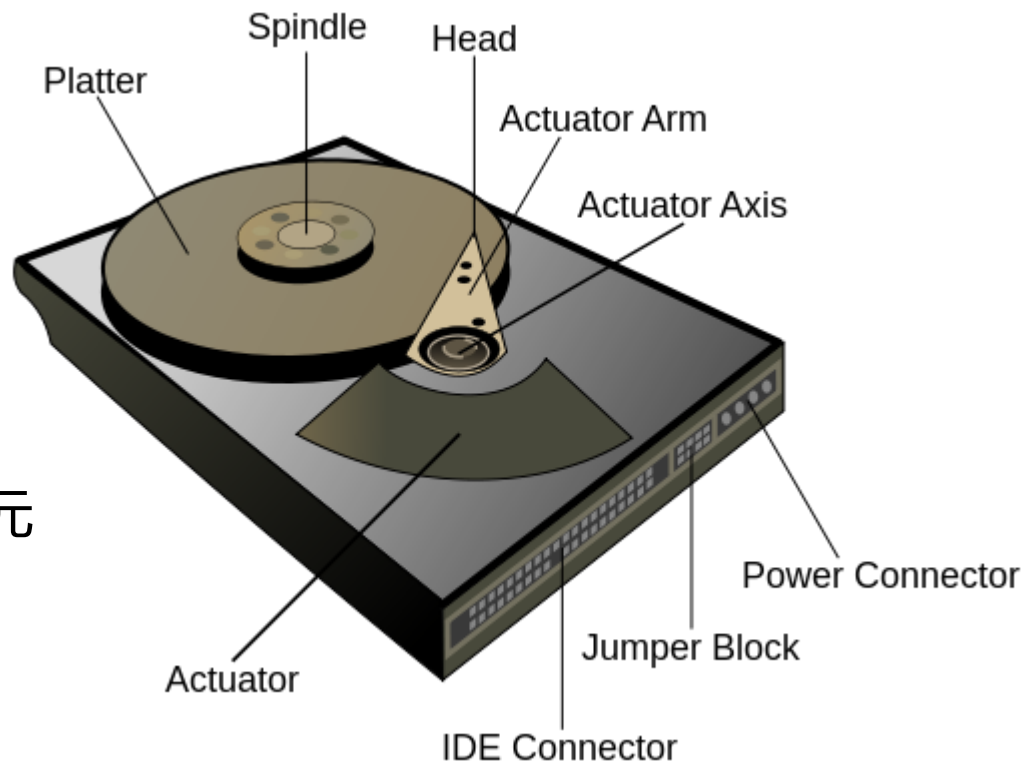
---

- 固态硬盘SSD
  - 闪存组织
  - SSD FTL机制
    - 地址映射
    - 磨损均衡



# 回顾：磁盘的结构

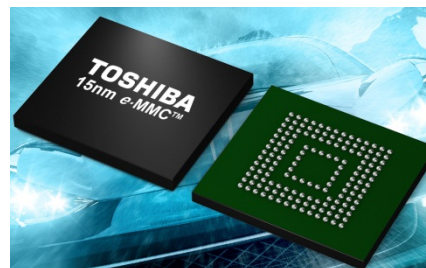
- 盘片：一组
  - 按一定速率旋转
- 磁道 (Track)
  - 位于盘片表面的同心圆
  - 用于记录数据的磁介质
  - bit沿着每条磁道顺序排列
- 扇区 (Sector)
  - 磁道划分为固定大小的单元  
一般为512字节
- 磁头：一组
  - 用于读写磁道上的数据
- 磁臂：一组
  - 用于移动磁头（多个）





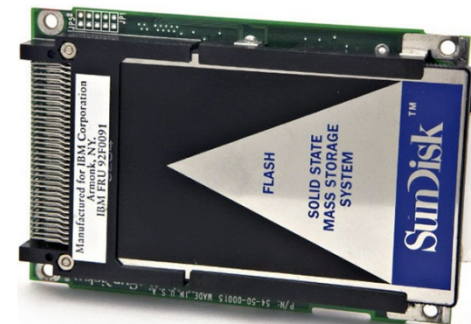
# 闪存 (Flash memory)

- 1984: NOR flash, 日本东芝公司, Fujio Masuoka
- 1987: NAND flash, 日本东芝公司
- 全电子器件, 无机械部件
- 非易失性存储
- 1992: SSD原型, SandDisk



## NOR闪存 vs. NAND闪存

- NOR是字节寻址, NAND是页寻址
- NOR读延迟是比NAND低100x
- NOR的擦除时间比NAND高300x
- NOR用于取代 ROM, 可执行代码
- NAND用于大容量持久化存储设备





# 闪存

- 信息存储方式

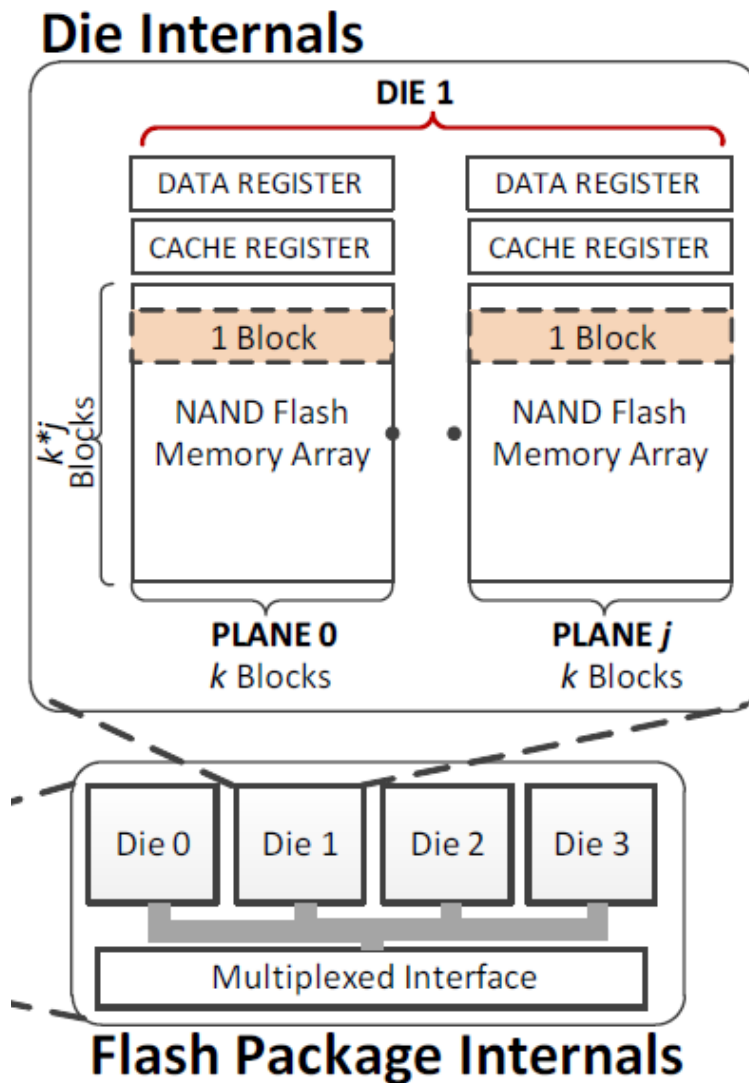
- SLC: 1 bit/cell, 2个值 : 0/1
- MLC: 2 bits/cell, 4个值 : 00/01/10/11
- TLC: 3 bits/cell, 8个值
- QLC: 4 bits/cell, 16个值





# NAND闪存组织

- Flash package
  - 多个Die
- Die
  - 多个Plane/Bank
- Plane/Bank
  - 很多块 ( 擦除块 )
  - 一些寄存器
- **块** (Block/Erase Block)
  - 很多页
- **页** (Page)
  - 很多cell





# 闪存组织

- 页

- 由数据区与OOB ( Out Of Band ) 区构成
- 数据区用于存储实际数据
- OOB区用于记录
  - ECC
  - 状态信息 : Erased/Valid/Invalid
  - Logic page number
- 页大小
  - SLC通常2KB~8KB , TLC通常4KB~16KB
- 64+页/块
- 块大小
  - SLC通常128KB、 256KB...
  - TLC通常2MB , 4MB , 8MB...





# 闪存的操作接口

- Read, Erase, Program (写)
- 读：read a page
  - 读的粒度是页
  - 读很快，读延迟在几十微秒 (us)
  - 读延迟与位置无关，也与上一次读的位置无关 (和磁盘不同)
- 擦除：erase a block
  - 把整个块写成全1
  - 擦除的粒度是块，必须整块擦除
  - 很慢：擦除时间为几个毫秒 (ms)
  - 需软件把块内有效数据拷贝到其它地方
- 写：program a page
  - 擦除后才能写，因为写只能把1变成0
  - 写的粒度是页
  - 写比读慢，比擦除快，写延迟在几百微秒 (us)





# 页的状态

---

Invalid, Erased, Valid

- 初始状态为Invalid
- 读：不改变页的状态
- 擦除：块内所有页的状态变为Erased
- 写
  - 只能写状态为Erased的页
  - 写完成，页状态变为Valid



# 闪存的性能和可靠性

## • 性能

- 写延迟比读高10倍以上
- 写延迟波动幅度大
- 擦除很慢：~ 磁盘定位延迟
- 延迟随密度增加而增长

Device	Read ( $\mu$ s)	Write ( $\mu$ s)	Erase (ms)	P/E Cycles
SLC	25	200-300	1.5-2	100,000
MLC	50	600-900	~3	10,000
TLC	~75	~900-1350	~4.5	1,000
QLC				200

## • 可靠性

- 磨损
  - 擦写次数有上限，随密度增加而减少
- 干扰
  - 读写一个页，相邻页中一些位的值发生翻转

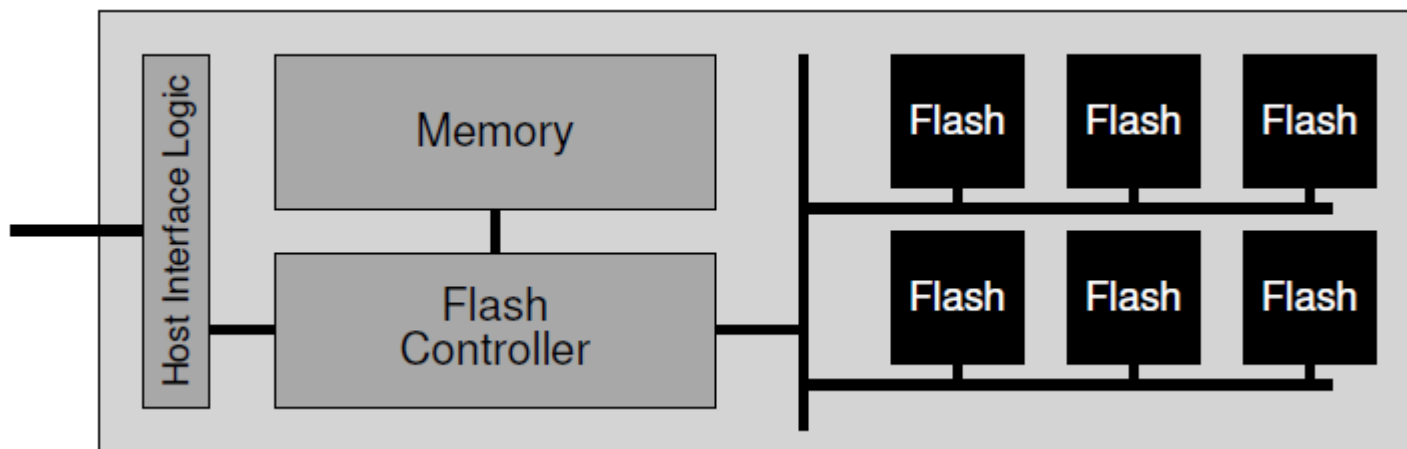
### 闪存特性：

1. 读延迟很低：随机读的性能远优于磁盘
2. 写慢：必须先擦除再写，ms级 ~ 磁盘写
3. 磨损：每个块擦写次数有上限



# 基于闪存的SSD

- 用很多闪存芯片来构成一个持久化存储设备SSD



- 多个闪存芯片：并行I/O，提高 I/O性能
- 与主机的接口：提供标准块设备接口
- 数据缓存和缓冲：SRAM/DRAM
- 闪存控制器（硬件）和FTL（固件）：控制逻辑
  - 主机命令转换成闪存命令 (Read/Erase/Program)
  - 逻辑块地址转换成闪存的物理地址 (页/块)
  - 缓存替换
  - 减少写干扰：块内的页顺序写



# 最简单的FTL：直接映射

- Direct mapping
  - 逻辑块的第N块直接映射到物理的第N页 (假设逻辑块与物理页都为4KB)
- 读操作容易：读逻辑第k块
  - 读物理第k页
- 写操作麻烦：写逻辑第k块
  - 第k页所在闪存块 (记为B0)
  - 把B0整个块读出来
  - 把B0整个块擦除
  - B0中的旧页和新的第k页：以顺序方式一页一页再写入B0
- 缺陷：写性能极差
  - 每写一个页，要读整个块、擦除整个块、写整个块
  - 写放大
  - 小粒度随机写性能比磁盘还差



# FTL改进: 异地更新

- 核心思想：异地更新 (out-of-place update)
  - 不再执行原地更新
  - 每次写页，写到一个新位置（新的物理页地址）
- 页级映射
  - 映射表：LBN  $\rightarrow$  物理页地址PPN, 页级映射表
- 写一个逻辑块k
  - 寻找一个空闲页p（例如当前擦除块中下一个空闲页p）
  - 在映射表中记录: 逻辑块  $k \rightarrow$  物理页 p
- 读一个逻辑块k
  - 查映射表，获得逻辑块k对应的物理页地址p
  - 读物理页p



# 页级映射

例子：依次写逻辑块100, 101, 2000和2001 (a1, a2, b1, b2)

Block:	0				1				2			
Page:	00	01	02	03	04	05	06	07	08	09	10	11
Content:												
State:	i	i	i	i	i	i	i	i	i	i	i	i

Block:	0				1				2			
Page:	00	01	02	03	04	05	06	07	08	09	10	11
Content:												
State:	E	E	E	E	i	i	i	i	i	i	i	i

Table: 100 → 0

Memory

Block:	0				1				2			
Page:	00	01	02	03	04	05	06	07	08	09	10	11
Content:	a1											
State:	V	E	E	E	i	i	i	i	i	i	i	i

Flash  
Chip

Table: 100 → 0 101 → 1 2000 → 2 2001 → 3

Memory

Block:	0				1				2			
Page:	00	01	02	03	04	05	06	07	08	09	10	11
Content:	a1	a2	b1	b2								
State:	V	V	V	V	i	i	i	i	i	i	i	i

Flash  
Chip



# 页级映射

- 页级映射表：LBN  $\rightarrow$  PPN
  - 整个放在内存中
  - 持久化：利用页的OOB区来保存映射表
  - 随着写页而被写到闪存
  - 掉电或重启，扫描OOB区恢复映射表
- 优点
  - 性能好：减少写放大
  - 可靠性好：映射关系被自动写入闪存
- 问题
  - 重写产生垃圾页
    - 每次写到新位置，导致原先页的内容无效
  - 内存开销大
    - 映射表全部放内存
    - 映射表的大小与SSD容量成正比



# 垃圾页 (garbage/dead page)

例子：依次写逻辑块100, 101, 2000和2001 (a1, a2, b1, b2)

Table:	100	→	0	101	→	1	2000	→	2	2001	→	3	Memory
Block:	0				1				2				Flash Chip
Page:	00	01	02	03	04	05	06	07	08	09	10	11	
Content:	a1	a2	b1	b2									
State:	V	V	V	V	i	i	i	i	i	i	i	i	

再写逻辑块100和101 (c1, c2)

Table:	100	→	4	101	→	5	2000	→	2	2001	→	3	Memory
Block:	0				1				2				Flash Chip
Page:	00	01	02	03	04	05	06	07	08	09	10	11	
Content:	a1	a2	b1	b2	c1	c2							
State:	V	V	V	V	V	V	E	E	i	i	i	i	

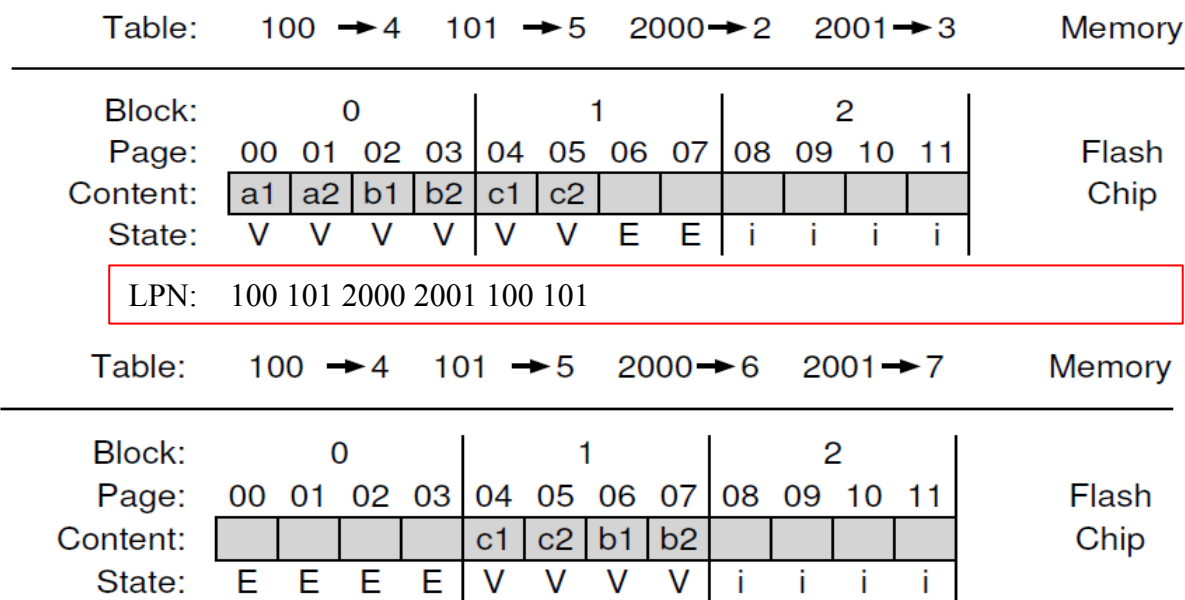
垃圾页





# 垃圾回收

- 思想
  - 选择一个含垃圾页的块
  - 把其中的活页 (live page)拷贝到其他块中 (读 & 重写)
  - 回收整个块，并把它擦除
- 如何判断活页？(live page vs. garbage page)
  - 每页记录它对应的逻辑块地址 (OOB区)
  - 查映射表, 如果映射表记录的PPN == 该页，是 live page



垃圾回收后



# 垃圾回收

- 问题：开销非常大
  - 活页需拷贝：读 & 写
  - 开销与活页所占的比例成正比
- 解决办法：超配（over-provisioning）
  - 实际物理空间比用户所见空间更大：多15%~45%
    - 例如，用户看到100GB的SSD, 实际上内部是120GB
    - GC时将数据写入over-provisioning space，减少对性能的影响
  - GC一般在SSD后台执行，尽量在设备不忙时执行，受限于空闲页数量



# 块级映射：block-level mapping

- 块级映射
  - 逻辑地址空间划分为chunk，chunk size=擦除块（物理块）size
  - 映射表：chunk#  $\rightarrow$  擦除块（物理块）地址PBN
- 读一个逻辑块
  - 逻辑块地址 = chunk# || 偏移
  - 用chunk#查映射表，获得对应的擦除块地址PBN
  - 物理页地址 = PBN || 偏移

例：依次写逻辑块2000, 2001, 2002, 2003 (a, b, c, d)

Table:	500	→	4	Memory									
Block:	0				1				2				Flash Chip
Page:	00	01	02	03	04	05	06	07	08	09	10	11	
Content:					a	b	c	d					
State:	i	i	i	i	V	V	V	V	i	i	i	i	

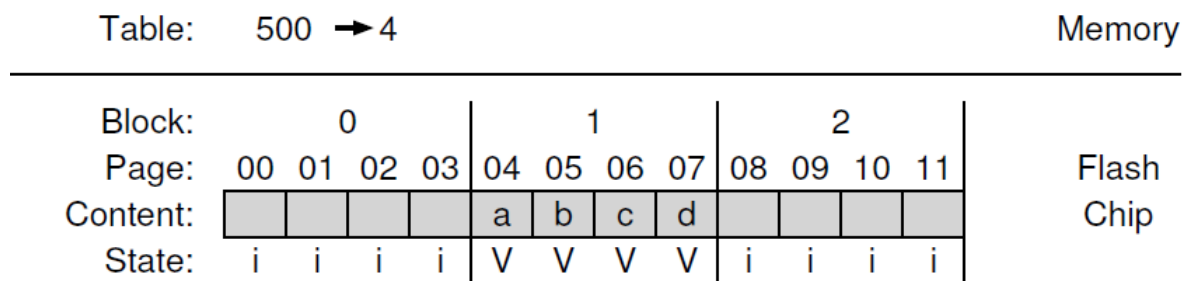


# 块级映射：block-level mapping

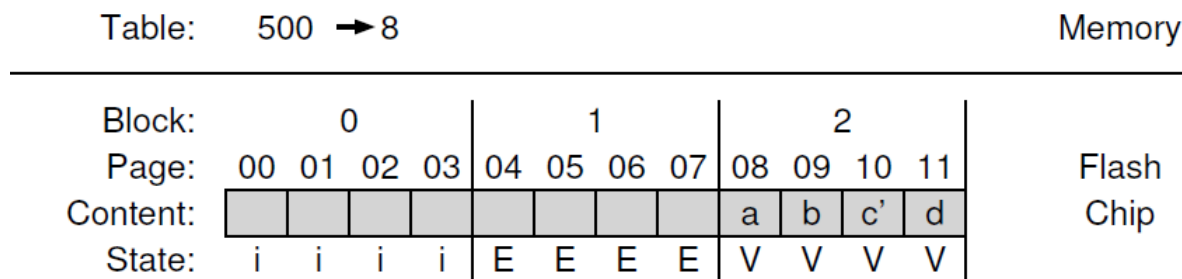
- 问题：小写性能差

- 写粒度小于擦除块：拷贝活页 (读 & 写), 写放大
- 小写很常见：擦除块大 (> 256KB)

例：依次写逻辑块2000, 2001, 2002, 2003 (a, b, c, d)



再写逻辑块2002 (c')





# 混合映射：hybrid mapping

- 思想
  - 将擦除块（物理块）划分为两类：数据块和日志块
  - 写逻辑块时都写入日志块
  - 数据块采用块级映射，数据映射表
  - 日志块采用页级映射，日志映射表
  - 适当的时候把日志块合并为数据块
- 读一个逻辑块
  - 先查日志映射表，按页级映射的方法
  - 如果没找到，再查数据映射表，按块级映射的方法



# 混合映射：hybrid mapping

例：当前逻辑块1000, 1001, 1002, 1003中数据分别为a,b,c,d

Log Table:  
Data Table: 250 → 8

Memory

Block:	0				1				2			
Page:	00	01	02	03	04	05	06	07	08	09	10	11
Content:									a	b	c	d
State:	i	i	i	i	i	i	i	i	V	V	V	V

Flash Chip

依次写逻辑块1000, 1001, 1002, 1003 (a', b', c', d')

Log Table: 1000 → 0 1001 → 1 1002 → 2 1003 → 3  
Data Table: 250 → 8

Memory

Block:	0				1				2			
Page:	00	01	02	03	04	05	06	07	08	09	10	11
Content:	a'	b'	c'	d'					a	b	c	d
State:	V	V	V	V	i	i	i	i	V	V	V	V

Flash Chip



# 混合映射：hybrid mapping

Log Table: 1000 → 0   1001 → 1   1002 → 2   1003 → 3  
Data Table: 250 → 8

Memory

Block:	0				1				2			
Page:	00	01	02	03	04	05	06	07	08	09	10	11
Content:	a'	b'	c'	d'					a	b	c	d
State:	V	V	V	V	i	i	i	i	V	V	V	V

Flash  
Chip

- Switch merge

- 直接把日志块转成数据块: 前提是整个日志块的页序与 chunk 一致
- 把原来的数据块回收擦除
- 优点：开销低，只修改映射表信息，无数据拷贝

Log Table:  
Data Table: 250 → 0

Memory

Block:	0				1				2			
Page:	00	01	02	03	04	05	06	07	08	09	10	11
Content:	a'	b'	c'	d'								
State:	V	V	V	V	i	i	i	i	i	i	i	i

Flash  
Chip

合并后



# 混合映射：hybrid mapping

Log Table: 1000 → 0    1001 → 1  
Data Table: 250 → 8

Block:	0				1				2				Flash Chip
Page:	00	01	02	03	04	05	06	07	08	09	10	11	
Content:	a'	b'							a	b	c	d	
State:	V	V	i	i	i	i	i	i	V	V	V	V	

Memory

- Partial merge

- 把数据块中有效页拷贝到日志块：日志块中页序与chunk一致
- 把日志块转成数据块，把原来的数据块回收擦除
- 有数据拷贝开销

Log Table:  
Data Table: 250 → 0

Block:	0				1				2				Flash Chip
Page:	00	01	02	03	04	05	06	07	08	09	10	11	
Content:	a'	b'	c	d									
State:	V	V	V	V	i	i	i	i	i	i	i	i	

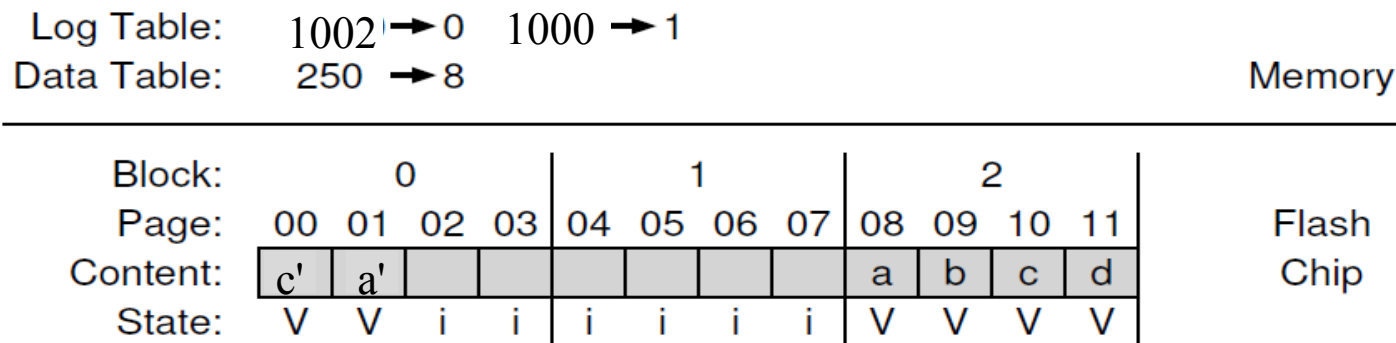
Memory

合并后



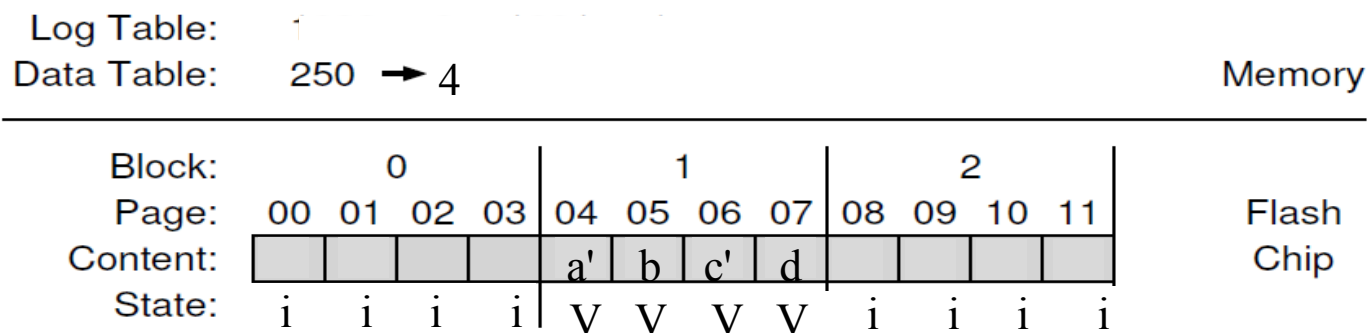


# 混合映射：hybrid mapping



- Full merge

- 分配一个新的日志块，从数据块和日志块分别拷贝有效页到新日志块
- 把新日志块转成数据块
- 把原来的数据块和日志块都回收擦除
- 开销很大：需要拷贝整个物理块的数据（读 & 写）





# 磨损均衡

- 目标
  - 让所有块被擦除的次数近似
- 动态磨损均衡
  - 每次写时，选择擦除次数较少或最少的空闲块
  - 局限性：不同数据的修改频率不同
    - 例子：只写一次的数据（static data），很少写的数据（cold data）
- 静态磨损均衡
  - 动态磨损均衡不考虑不会被回收的物理块，例如长时间不被修改的逻辑块（写冷块）
  - 不再被写，不再有磨损
  - 解决办法：FTL定期重写冷块，将其写入磨损较多的块



# 总结

- 闪存的特性
  - 读延迟很低：读性能远优于磁盘
  - 写慢，擦除慢（ms级）：必须先擦除再写
  - 磨损：每个块擦写次数有上限
- SSD FTL主要功能

