存储设备及其对系统设计的影响

张佳辰　2019. 9. 27

jczhang@nbjl.nankai.edu.cn

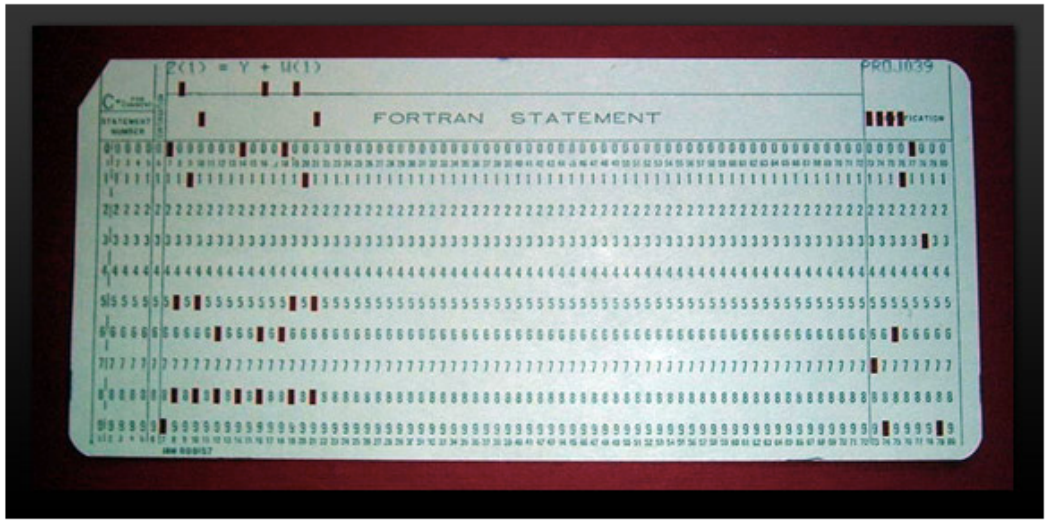
Parallel and Distributed Software Technology Lab

Nankai - Baidu Joint Laboratory

目录

- 存储设备
- 存储I/O性能
- 存储设备对系统设计的影响

存储设备的发展

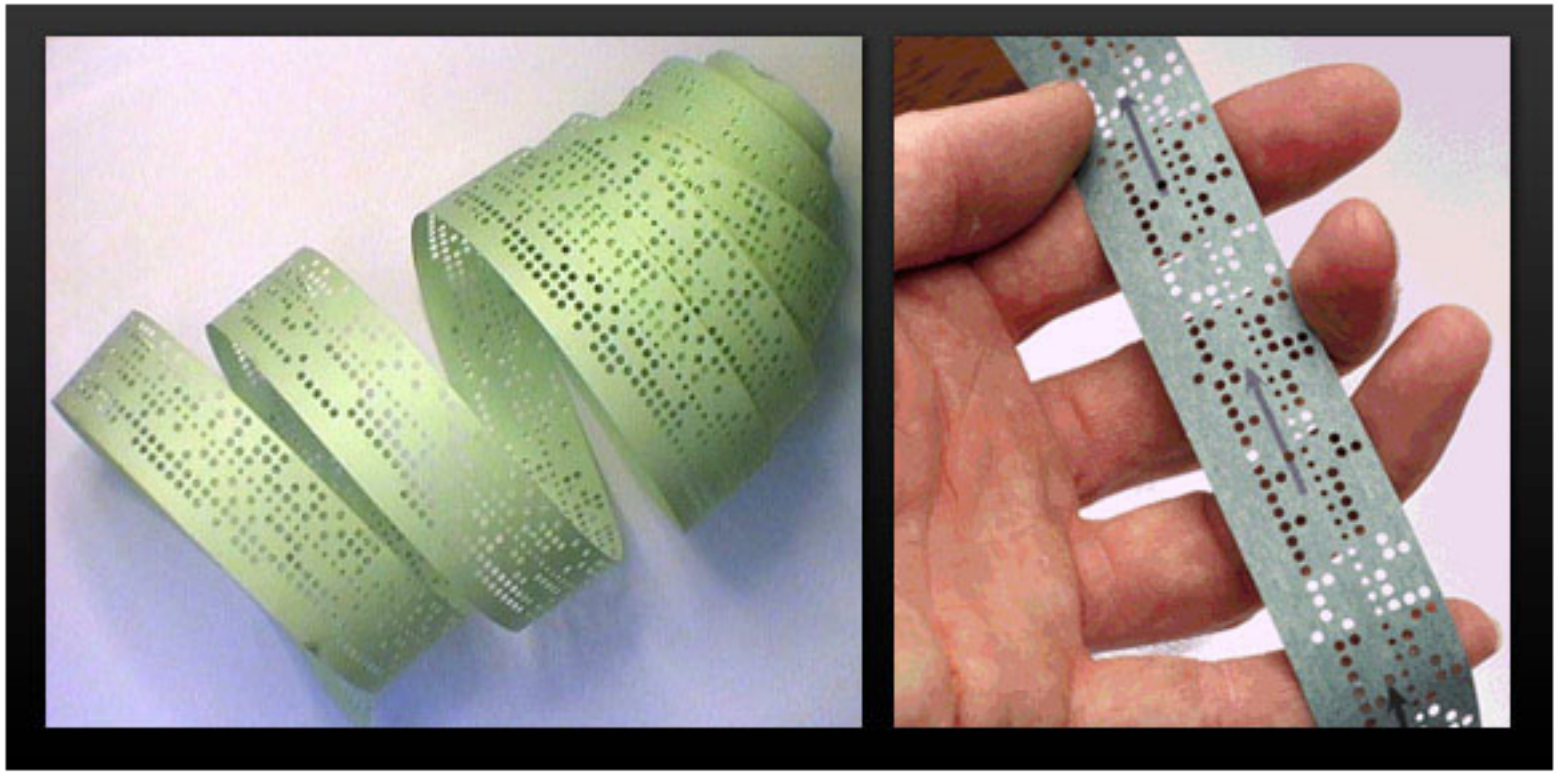


穿孔卡
(一个简单的fortran程序)

早在1725用于机器，1970年代广泛用于计算机。

穿孔卡的Reader 和
穿孔卡Writer

存储设备的发展



穿孔带

存储设备的发展

磁带 (Magnetic Tape)

- 左图 1951年 UNIVAC I 计算机
- 右图 1971年 IBM 3410 Magnetic Tape Subsystem

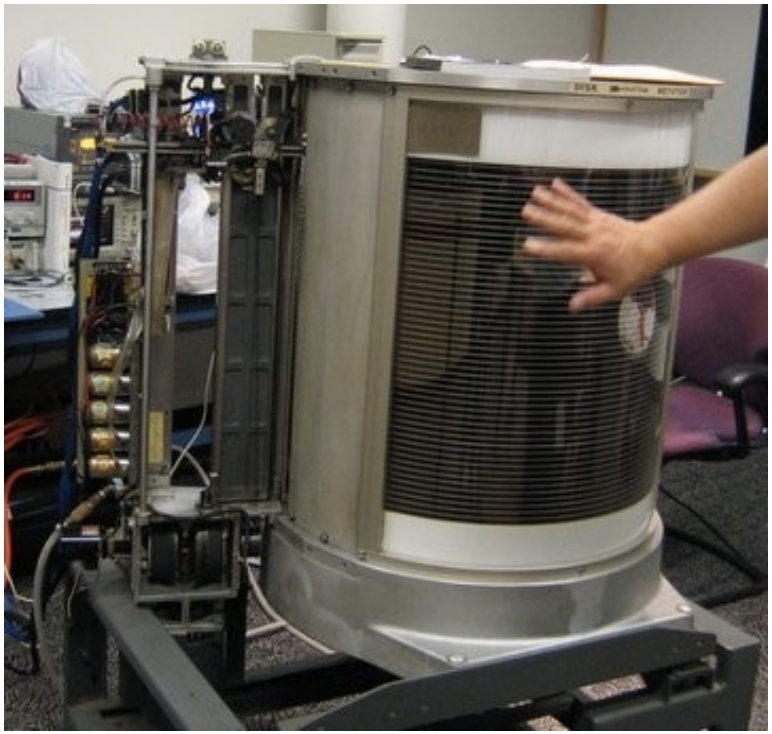
70~80年代用于个人电脑(PC)的小型磁带。

- 速度约 250 Byte/s
- 大小单面约660 KB。

存储设备的发展

磁带

存储设备的发展



硬盘 (Hard Disk Drive)

1956年， IBM的世界上第一块硬盘，容量5MB

存储设备的发展

软盘 70到90年代广泛使用

存储设备的发展

光盘 1978年面世

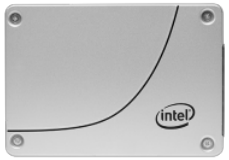
存储设备的发展

SSD (Solid State Driver)

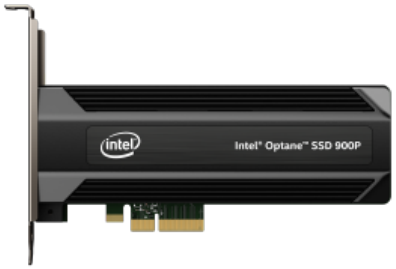
现在流行的存储设备...

Tape

HDD

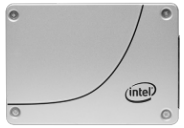


SATA SSD



NVMe SSD

新兴存储设备...

Tape **SMR HDD** HDD SATA SSD NVMe SSD **Persistent Memory**

备份数据 成本更低

在线数据 性能更高

目录

- 存储设备
- 存储I/O性能
  - 性能指标
  - 测试方法
- 存储设备对系统设计的影响

存储性能指标

延迟 (Latency) 完成一次操作的时间。

带宽 (Bandwidth) 读/写速度，通常单位为MB/s 或 GB/s。

IOPS (I/O operations per second) 每秒的操作数。

	DRAM 内存	持久型内存	HDD/SSD 存储
			
延迟	**~100 ns**	**~250 ns**	**10us ~ 10ms**
单条读带宽	~20 GB/s	~ 7 GB/s	120 MB ~ 2.5 GB/s
单条写带宽	~13.5 GB/s	~ 2.3 GB/s	80 MB ~ 2.2 GB.s

存储性能 -- fio测试延迟

(最小)延迟 (Latency) 完成一次操作的时间。

fio 读延迟脚本：

```
1   [global]
2   filename=fio_testdata
3   size=20G
4   runtime=30
5   ioengine=libaio
6   direct=1
7   ramp_time=10            # start measuring after warm-up time
8
9   [read]
10  readwrite=randread
11  blocksize=4K
```

存储性能 -- fio测试带宽

带宽 (Bandwidth) 读/写速度，通常单位为MB/s 或 GB/s。

fio 读带宽脚本：

```
1   [global]
2   filename=fio_testdata
3   size=20G
4   runtime=30
5   ioengine=libaio
6   direct=1
7   ramp_time=10            # start measuring after warm-up time
8
9   [read]
10  readwrite=read
11  numjobs=16
12  blocksize=64k
13  offset_increment=128m   # each job starts at a different offset
```

存储性能 -- fio测试IOPS

IOPS (I/O operations per second) 每秒的操作数。

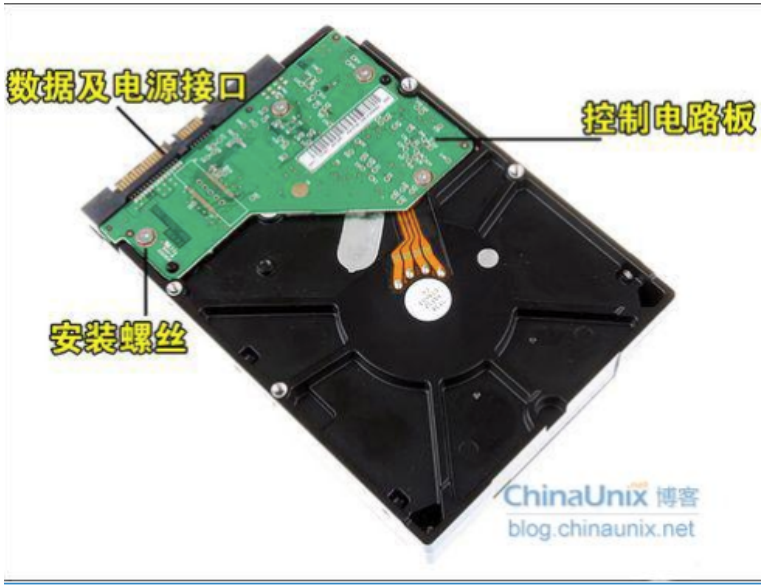
fio 读IOPS脚本：

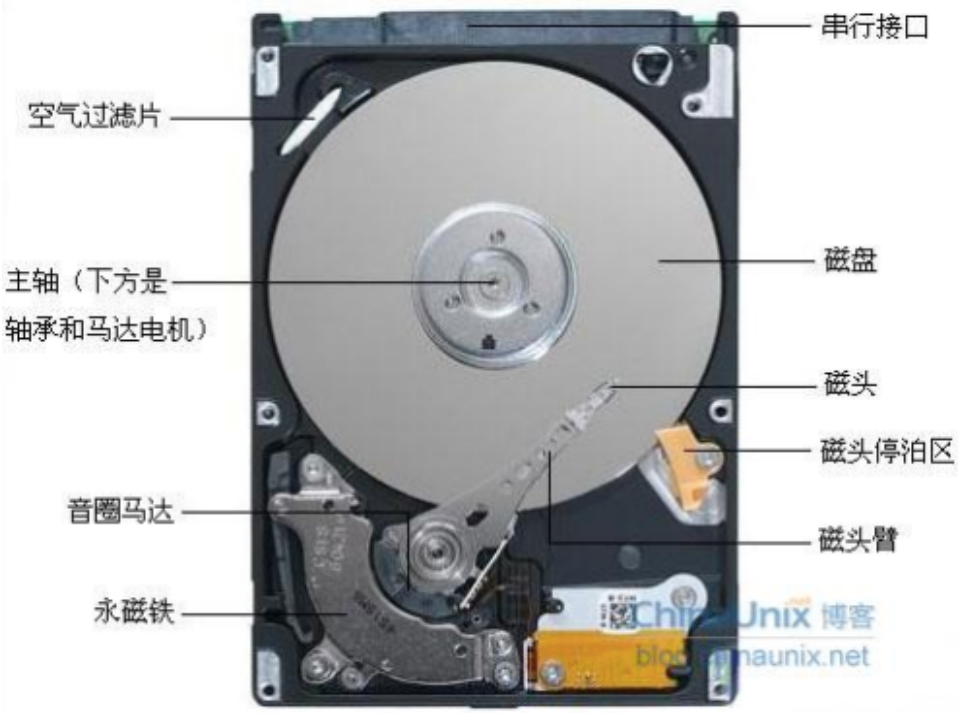
```
1   [global]
2   filename=fio_testdata
3   size=20G
4   runtime=30
5   ioengine=libaio
6   direct=1
7   ramp_time=10            # start measuring after warm-up time
8
9   [read]
10  readwrite=randread
11  numjobs=64
12  blocksize=4K
```

目录

- 存储设备
- 存储I/O性能
- 存储设备对系统设计的影响
  - HDD 和 SMR HDD
  - Nand Flash SSD
  - Persistent Memory

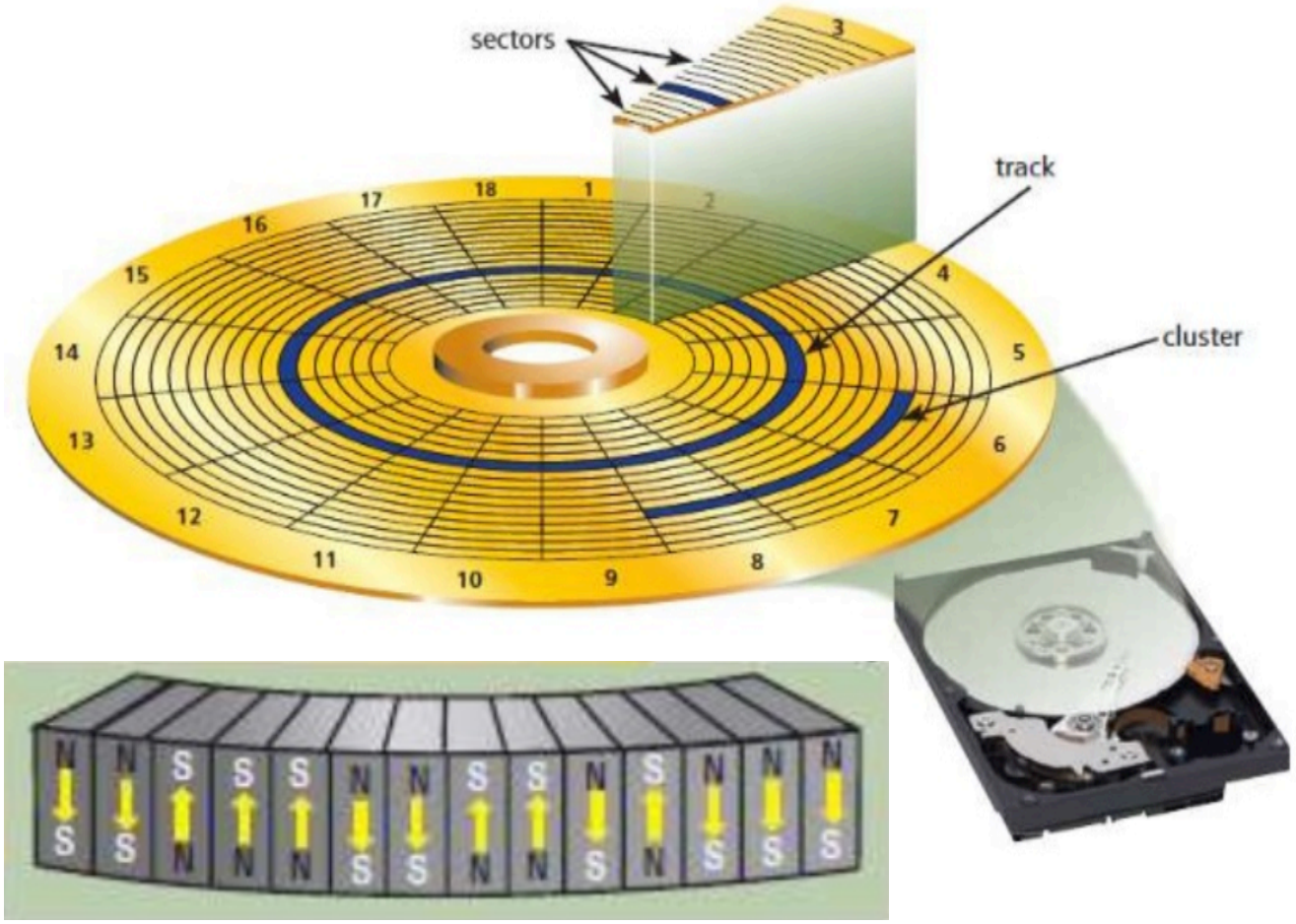
HDD 磁盘存储原理





内部构造

HDD 磁盘存储原理



存储原理

HDD 磁盘I/O性能

- 两种HDD的随机/顺序：

	(SCSI盘) Cheetah	(SATA盘) Barracuda
$R_{I/O}$ Random	0.66 MB/s	0.31 MB/s
$R_{I/O}$ Sequential	125 MB/s	105 MB/s

HDD随机读写性能很差！

- HDD随机读写延迟模型：

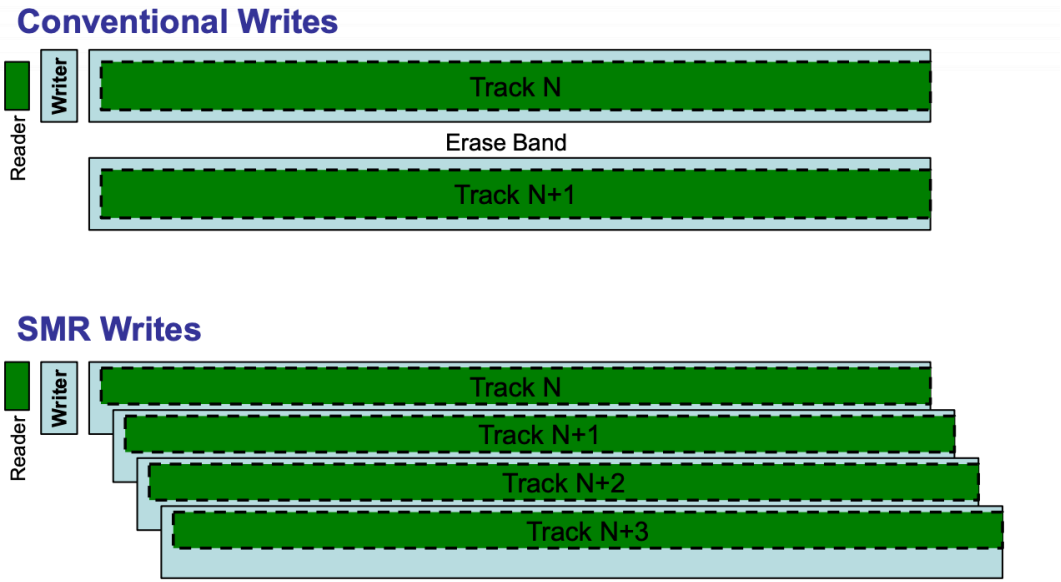
$$T_{I/O} = T_{seek} + T_{rotation} + T_{transfer}$$

- 随机读写情况下：

$$T_{seek} = 4\,ms,\ T_{rotation} = 2\,ms,\ T_{transfer} = 30\,microsecs$$

《**Operating Systems: Three Easy Pieces**》, Remzi H. Arpaci–Dusseau and Andrea C., Arpaci–Dusseau, Arpaci–Dusseau Books , March, 2015 (Version 0.80)

SMR HDD 磁盘存储原理

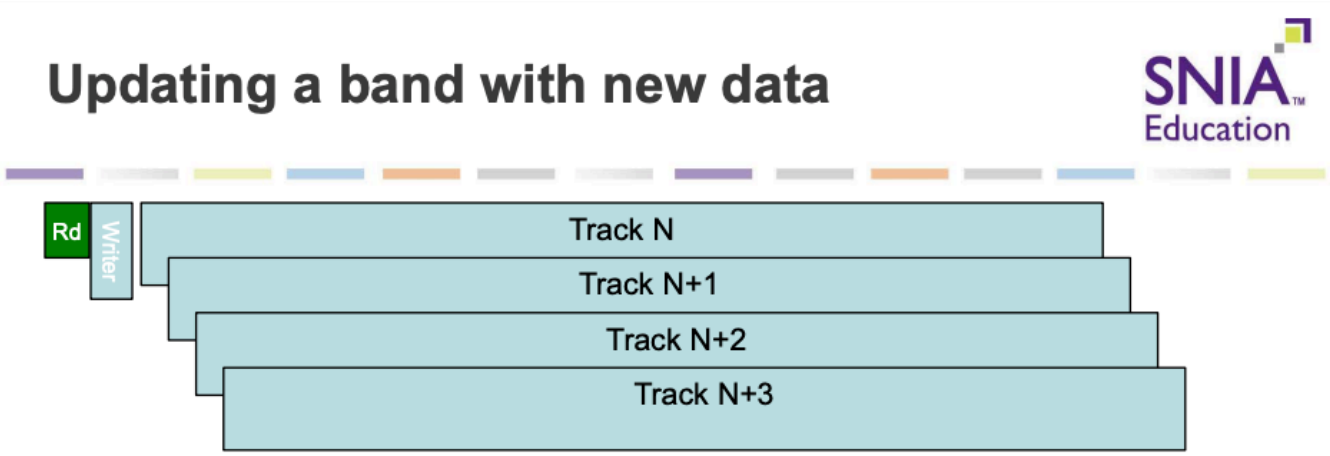


叠瓦式磁记录 HDD

Shingled Magnetic Recording (SMR) HDD

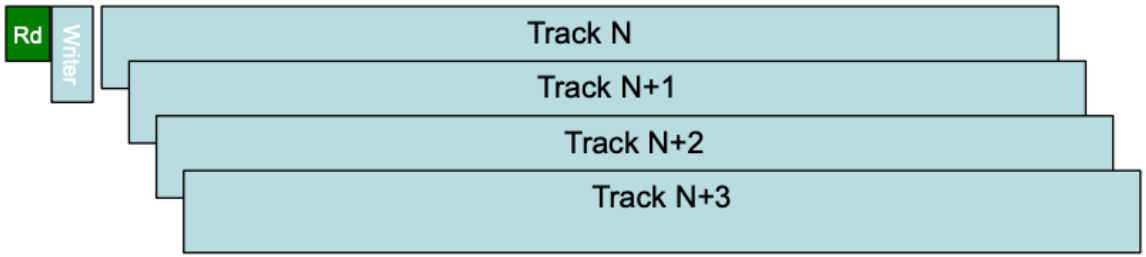
- 存储密度
- 随机读写性能更差

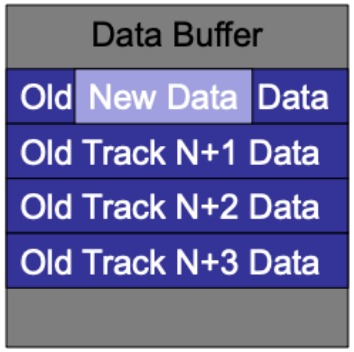
SMR HDD 磁盘存储原理



SMR HDD 磁盘存储原理

Updating a band with new data

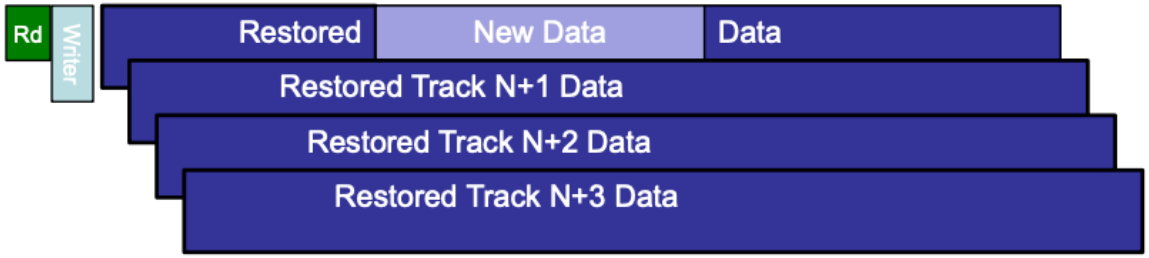


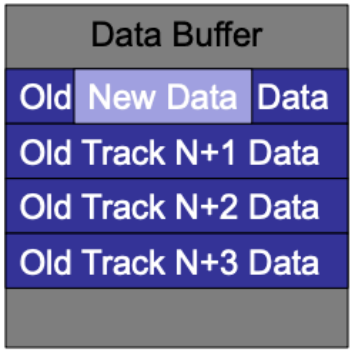


1. Read old data
2. Merge with new data

SMR HDD 磁盘存储原理

Updating a band with new data



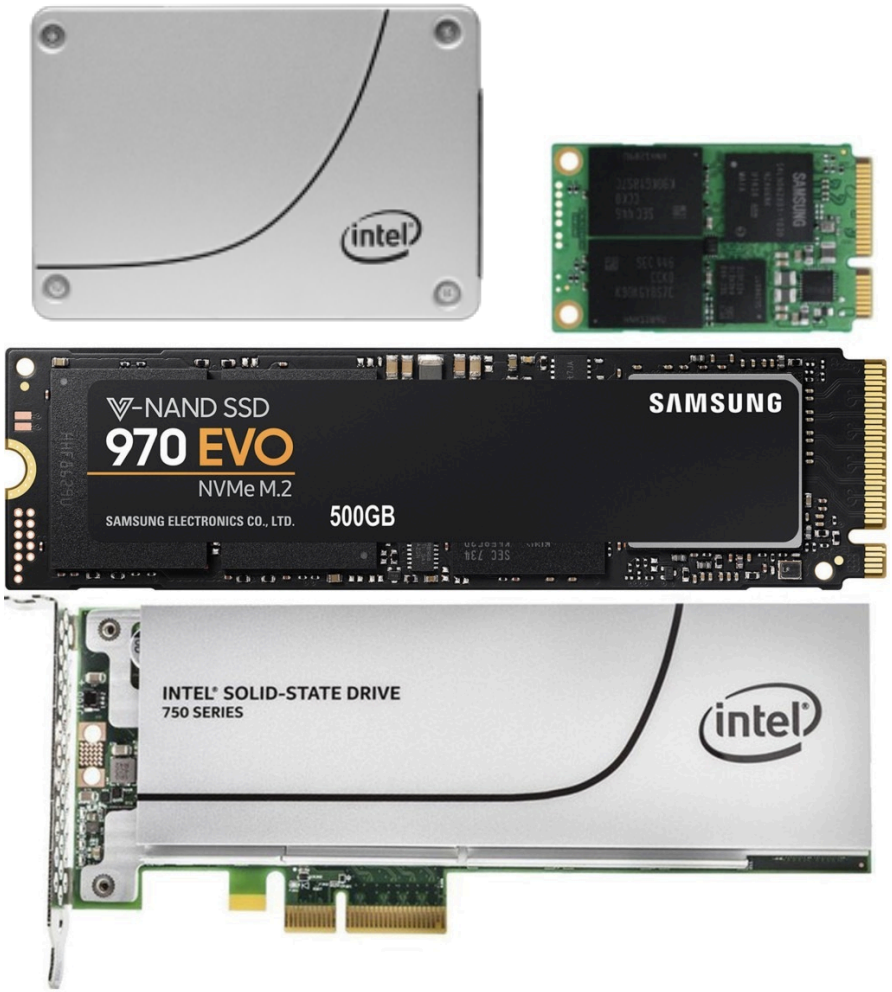


1. Read old data
2. Merge with new data
3. Write new data, refreshing old data

HDD 和 SMR HDD I/O性能优化

- **更大的读写单元**：减小随机性，充分利用顺序读写带宽
- **内存作磁盘cache**：利用数据访问的空间/时间局部性，用更快的内存加快IO。
- **透明压缩**：压缩磁盘块可以减少IO带宽，提高性能。
- **IO合并**：将多次空间连续但是提交时间乱序的小IO合为一次连续大IO。
- **使用Log structured 方式优化写性能**：用追加的方式将随机写转为顺序写。

Nand Flash based SSD



Nand Flash 存储介质

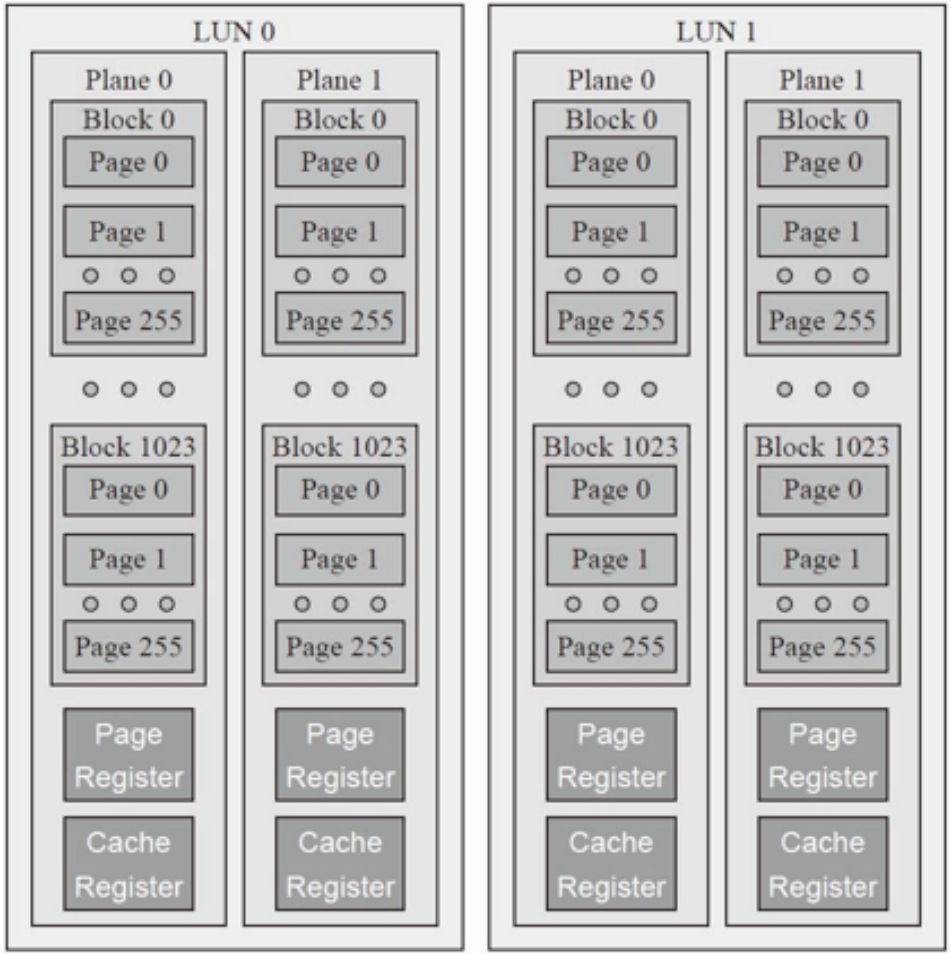


图3-7　闪存内部组织架构

Nand Flash 存储介质

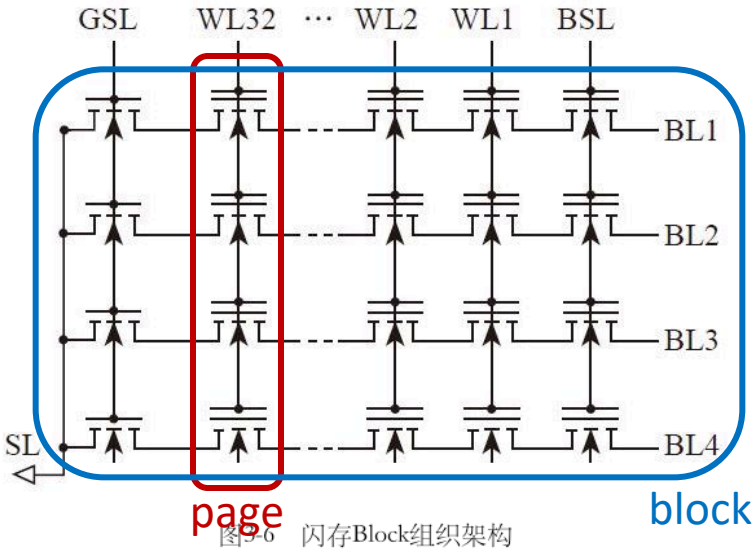


图3-6 闪存Block组织架构

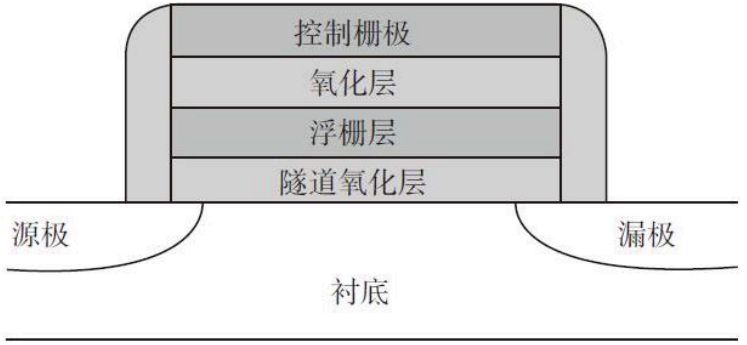


图3-1 浮栅晶体管结构

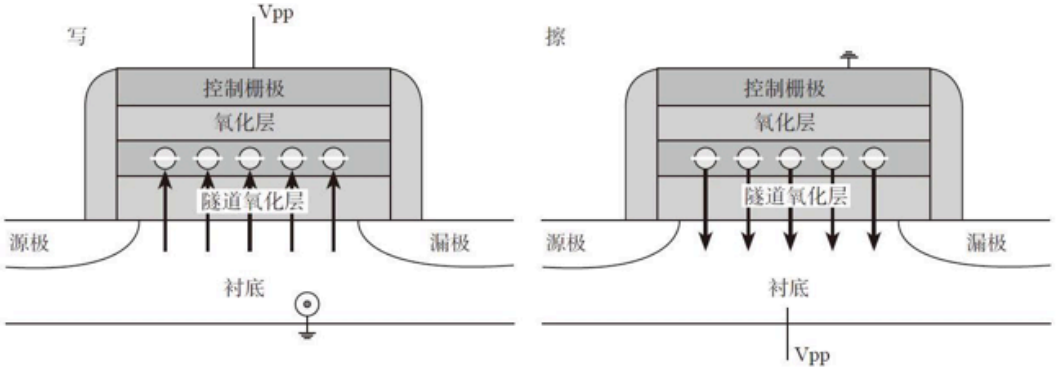


图3-2 左：写原理；右：擦除原理

写：控制栅极加正电，衬底接地，电子被“吸入”浮栅层。 (写为0)

擦：衬底(P-well)加正电，电子被“吸”回来，数据被擦除。 (擦为1)

读：以page为单位，要读的page栅极加电压，通过读BL来确定0 / 1值。

注意：衬底加电压整个block会被擦，所以并非字节寻址。

图摘自《深入浅出SSD：固态存储核心技术、原理与实战》

Flash 作存储的复杂性

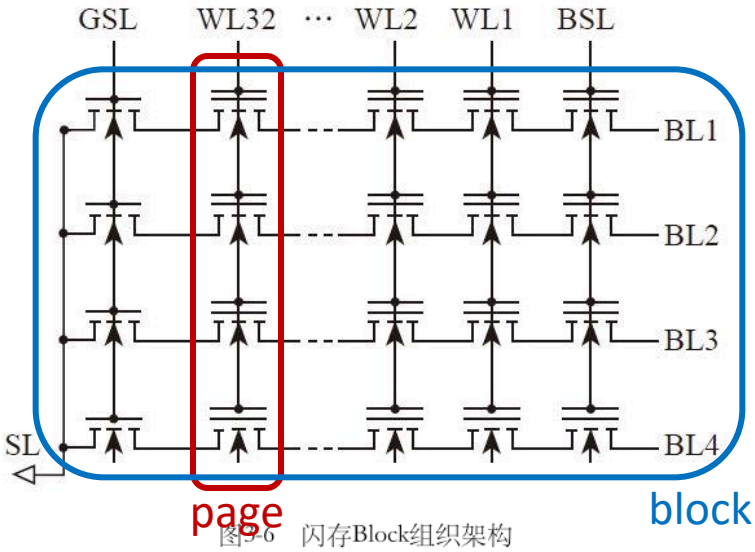


图3-6 闪存Block组织架构

- 最小读写单元为page。
- 擦除后的page只能写一次，要覆盖写page需要先擦除，擦除单位为block。
- 每个block存在最大擦除次数。

- 由于Block size大于page size，所以可能需要进行“读-修改-写操作”(Read-Modify-Write, RMW) 造成**写放大问题**。Over-provison(OP)和提前的垃圾回收(GC)可以减少SSD写放大!
- 由于每个block存在最大擦除次数，即使用寿命，所以需要**磨损均衡**。

图摘自《深入浅出SSD：固态存储核心技术、原理与实战》

管理的Nand Flash的SSD FTL层

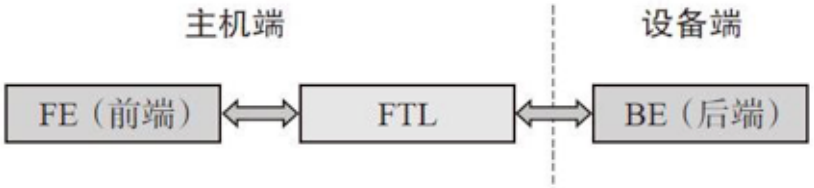


图4-1　FTL在主机端

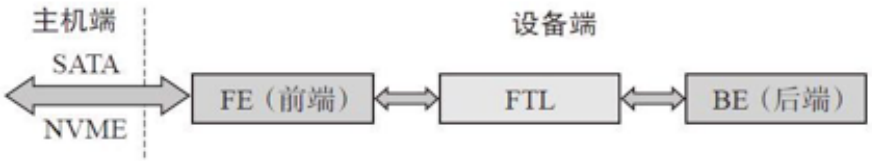


图4-2　FTL在设备端

一般实现于设备端，主要功能：

- 管理逻辑块-物理块的映射
- 负责垃圾回收(GC)和over-provison(OP)
- 磨损平衡

SSD FTL层的OP

	CH0			CH1			CH2			CH3		
	1	5	9	2	6	10	3	7	11	4	8	12
Block 0	13	17	21	14	18	22	15	19	23	16	20	24
	25	29	33	26	30	34	27	31	35	28	32	36
	37	41	45	38	42	46	39	43	47	40	44	48
Block 1	49	53	57	50	54	58	51	55	59	52	56	60
	61	65	69	62	66	70	63	67	71	64	68	72
	73	77	81	74	78	82	75	79	83	76	80	84
Block 2	85	89	93	86	90	94	87	91	95	88	92	96
	97	101	105	98	102	106	99	103	107	100	104	108
	109	113	117	110	114	118	111	115	119	112	116	120
Block 3	121	125	129	122	126	130	123	127	131	124	128	132
	133	137	141	134	138	142	135	139	143	136	140	144
	145	149	153	146	150	154	147	151	155	148	152	156
Block 4	157	161	165	158	162	166	159	163	167	160	164	168
	169	173	177	170	174	178	171	175	179	172	176	180
Block 5												

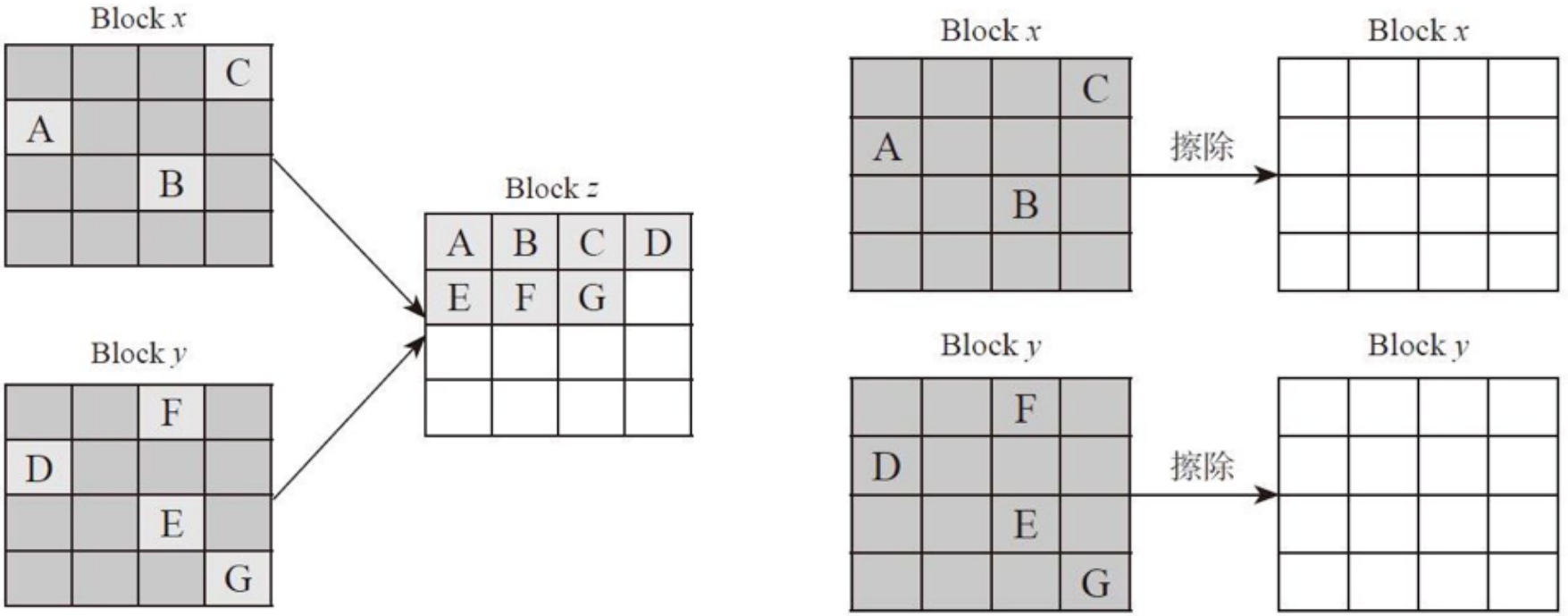
考虑覆盖写block 0 中的某一page：

1) Block 5不作OP， 暴露给主机：
需要读block 0中 9 个page到buffer，
修改其中1个page，擦除block 0，
再写入9页，造成了“写放大”。

2) Block 5 作OP，不暴露给主机：
可以直接写Block 5

图摘自《深入浅出SSD：固态存储核心技术、原理与实战》

SSD FTL层的GC



图摘自《深入浅出SSD：固态存储核心技术、原理与实战》

SSD 的 I/O性能

Intel SSD 750 参数		
容量	400GB	1.2TB
外形尺寸	2.5"15mm SFF-8639 or PCIe Add-In Card (HHHL)	
接口	PCIe3.0 × 4-NVMe	
控制器	Intel CH29AE41AB0	
NAND	Intel 20nm 128Gbit MLC	
顺序读	2200MB/s	2400MB/s
顺序写	900MB/s	1200MB/s
4KB 随机读	430k IOPS	440k IOPS
4KB 随机写	230k IOPS	290k IOPS
空闲功耗	4W	4W
读写功耗	9W/12W	10W/22W
加密支持	N/A	
耐写性	70GB Writes per Day for Five Years	

SSD 不再适用传统的HDD优化

- ~~**更大的读写单元：减小随机性，充分利用顺序读写带宽**~~ (可以用更小读写单元)
- **内存作磁盘cache：**利用数据访问的空间/时间局部性，用更快的内存加快IO。 (作用减弱)
- **透明压缩：**压缩磁盘块可以减少IO带宽，提高性能。(无法提升IO性能，只能节省存储空间)
- ~~**IO合并：将多次空间连续但是提交时间乱序的小IO合为一次连续大IO。**~~(反而增加延迟，越高性能SSD越不需要合并)
- **使用Log structured 方式优化写性能：**用追加的方式将随机写转为顺序写。(意外收获：用在SSD上可以减少写放大)

Phase Change Memory 原理

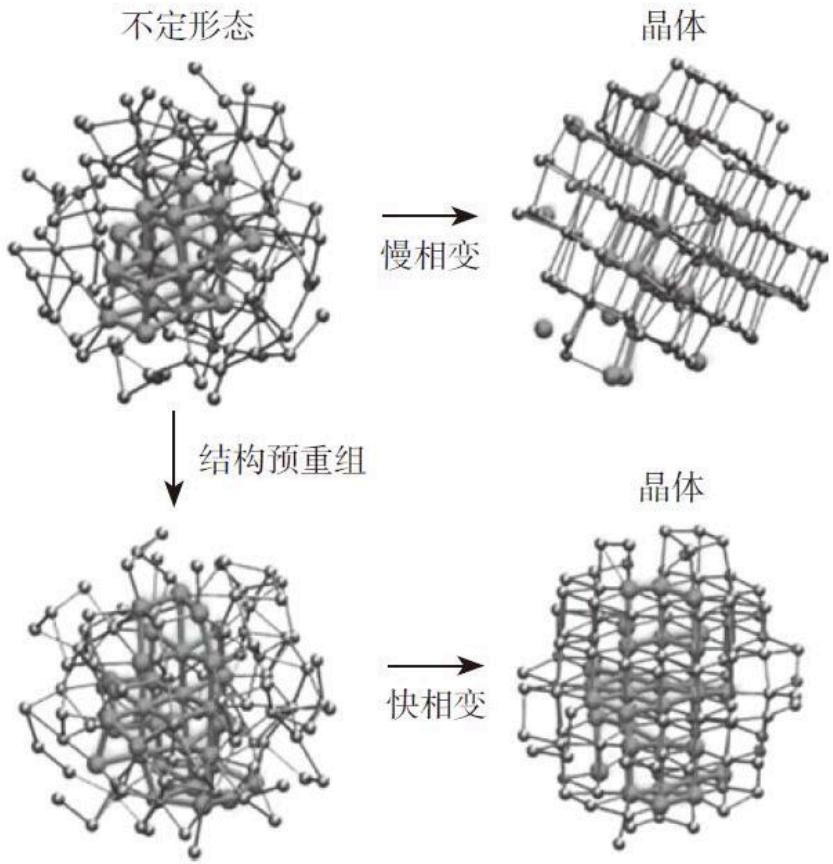


图3-25　相变晶体原理

Phase Change Memory(PCM)利用物质的多种相态存储信息。

一般用准金属合金GST以各种方式加热和冷却材料来操纵相态之间的变化来实现信息的存储。

Numonyx 的相变存储器使用一种含锗、锑、碲的合成材料（Ge2Sb2Te5），多被称为 GST。
Intel 开发的相变存储器使用了硫属化物（Chalcogenides）。

Intel和Micron的3D Xpoint技术被猜测属于PCM的技术。

摘自《深入浅出SSD：固态存储核心技术、原理与实战》

PM存储系统设计问题

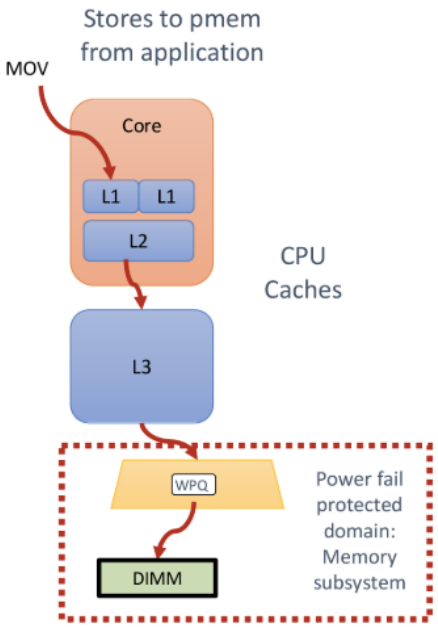
近年PM的相关工作很多。

都涉及到不同类存储系统利用PM时的 持久化、原子可见性、崩溃一致性 等问题。

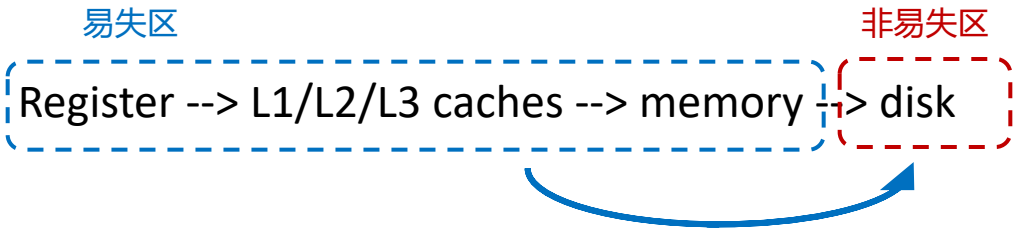
表 I: 相关工作分类

PM 编程模型	PM 数据结构	PM 存储系统
Giles 等 [2] (2018), Sorin 等 [3] (2017), Giles 等 [4] (2015), Zhang 等 [5] (2015), Caulfield 等 [6] (2010).	Persistent RB-tree [7] (2018), Persistent B+-tree [8] (2018), Persistent lock-free queue [9] (2018), BzTree [10] (2018), WORT [11] (2017), FPTree [12] (2016), NV-tree [13] (2015), wB+tree [14] (2015).	**文件系统**: SoupFS [15] (2017), Octopus [16] (2017), NOVA [17] [18] (2016), Sehgal 等 [19] (2015), PMFS [20] (2014). **KV 存储**: NoveLSM [21] (2018), HiKV [22] (2017), PapyrusKV [23] (2017), Persistent Memcached [24] (2017), UDORN [25] (2017), MetraDB [26] (2016). **其他系统**: NV-Dedup [27] (2018), DBMS 应用 PM [28] (2018), SwapX [29] (2017), PM 磁盘 cache [30] (2017), PM 存 ext4 元数据日志 [31] (2017), PM 存 ext4 元数据 [32] [33] (2016), PM 存 Ceph 元数据 [34] (2015).

PM存储系统设计问题 – 持久化



- 传统存储设备的持久化：



- PM设备的持久化：

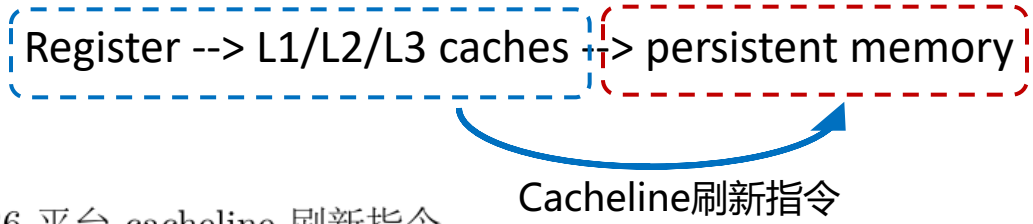
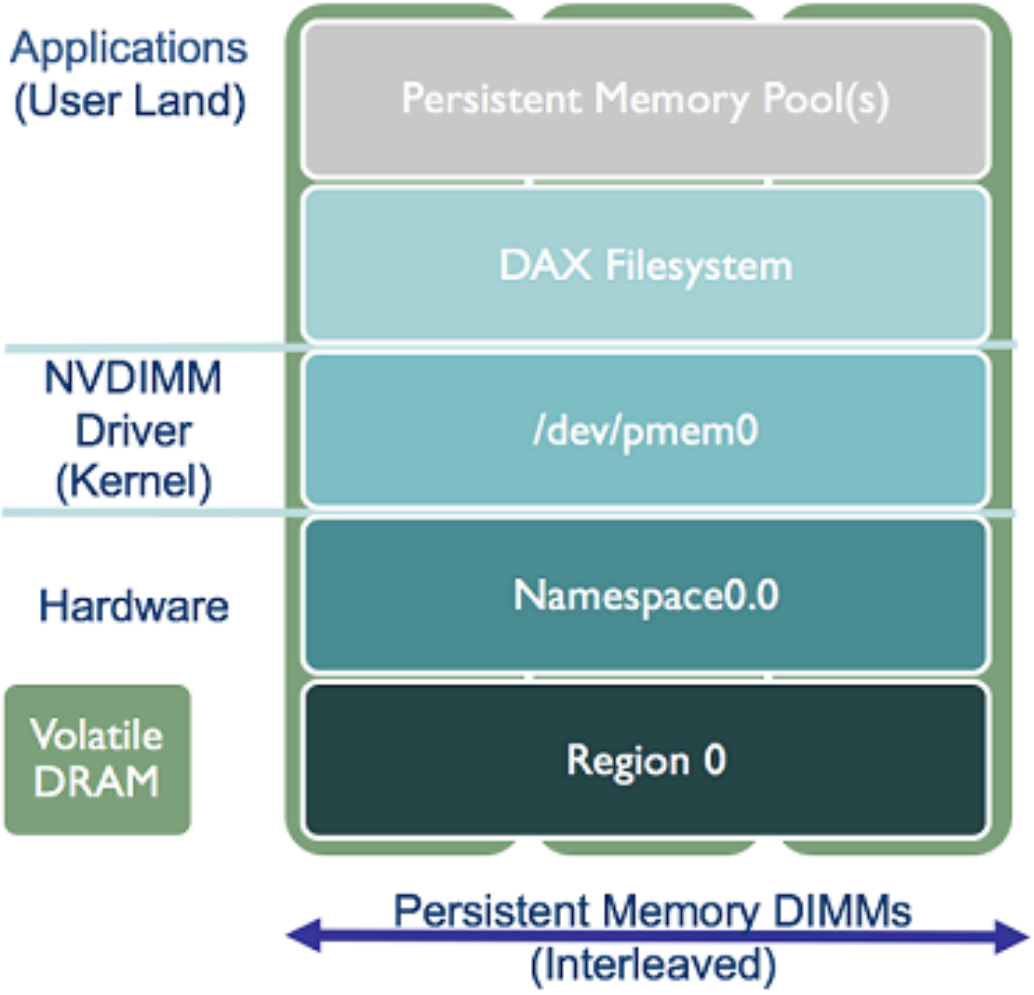


表 III: x86 平台 cacheline 刷新指令

指令	说明
CLFLUSH	几乎所有 CPU 都支持，但没有任何并发性，串行地执行
CLFLUSHOPT + SFENCE	比 CLFLUSH 新，但是不是串行执行的，因此需要 SFENCE
CLWB + SFENCE	相对 CLFLUSHOPT，可以在刷入 PM 后仍然让 cache 保持有效，局部性较好的数据使用此命令可以提升性能
NT stores + SFENCE	即 non-temporal store，直接跳过 cache 的 store 命令，因此不需要刷新 cache
WBINVD	只能在内核模式用，将所有 CPU 的 cache 刷入 PM，一般不用，太影响性能

PM存储系统设计问题 – 更多样的用法



可以直接利用PM-aware FS
或更上层的PMDK封装

对fsdax namespace可以格式化
PM-aware File System
(ext4, XFS已经支持)

多种namespace驱动: **devdax, fsdax, sector, raw, kmem ....**

对app-direct region可分namespace
(与磁盘分区概念类似)

3种region
(ipmctl工具管理)
- **memory mode**（可配百分比）
- **app direct** (interleaved)
- **app direct** (not interleaved)

PM存储系统设计问题 – 更多样的用法

	memory-mode	**app-direct** 作单独numa node内存	**app-direct** 作传统磁盘	**app-direct** 作PM-aware FS	**app-direct** 用PMDK	…
Region	memory mode (interleaved 或not interleaved)	app direct	app direct	app direct	app direct	
namespace		kmem	sector	fsdax	fsdax / devdax	
文件系统			任意文件系统	ext4-dax / xfs-dax	ext4-dax / xfs-dax / no fs	
用户态库					PMDK	

Thanks! Questions?

参考:

[1] The History of Computer Data Storage, in Pictures, https://royal.pingdom.com/the-history-of-computer-data-storage-in-pictures/

[2] 《大话存储2:存储系统架构与底层原理极限剖析》

[3] 《深入浅出SSD：固态存储核心技术、原理与实战》

[4] 硬盘的存储原理和内部架构, https://blog.csdn.net/shaochenshuo/article/details/72821089

[5] https://www.slideshare.net/youthmirza/hard-disk-32607833

[6] 《Operating Systems: Three Easy Pieces》, Remzi H. Arpaci-Dusseau and Andrea C., Arpaci-Dusseau, Arpaci-Dusseau Books , March, 2015 (Version 0.80)

[7] A. Aghayev, T. Ts'o, G. Gibson, and P. Desnoyers, "Evolving Ext4 for Shingled Disks," *15th USENIX Conf. File Storage Technol. (FAST 17)*, vol. 2, pp. 105–120, 2017.

[8] https://www.snia.org/sites/default/files/Dunn-Feldman_SNIA_Tutorial_Shingled_Magnetic_Recording-r7_Final.pdf

[9] https://software.intel.com/en-us/articles/quick-start-guide-configure-intel-optane-dc-persistent-memory-on-linux

[10] D. J. Sorin, "Persistent Memory Programming," *Computer (Long. Beach. Calif).*, vol. 50, no. 3, p. 12, 2017.