新型存储技术
及其在分布式系统中的应用

张佳辰　2018-12-21

jczhang@nbjl.nankai.edu.cn

Parallel and Distributed Software Technology Lab

Nankai - Baidu Joint Laboratory

“外存”

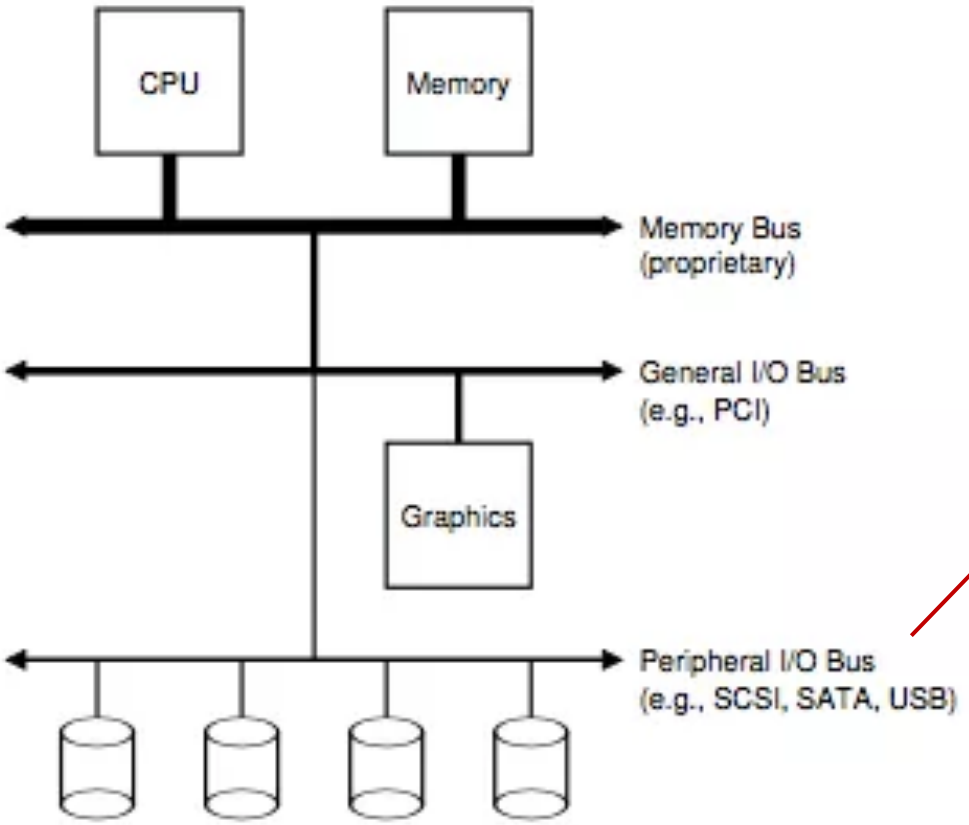
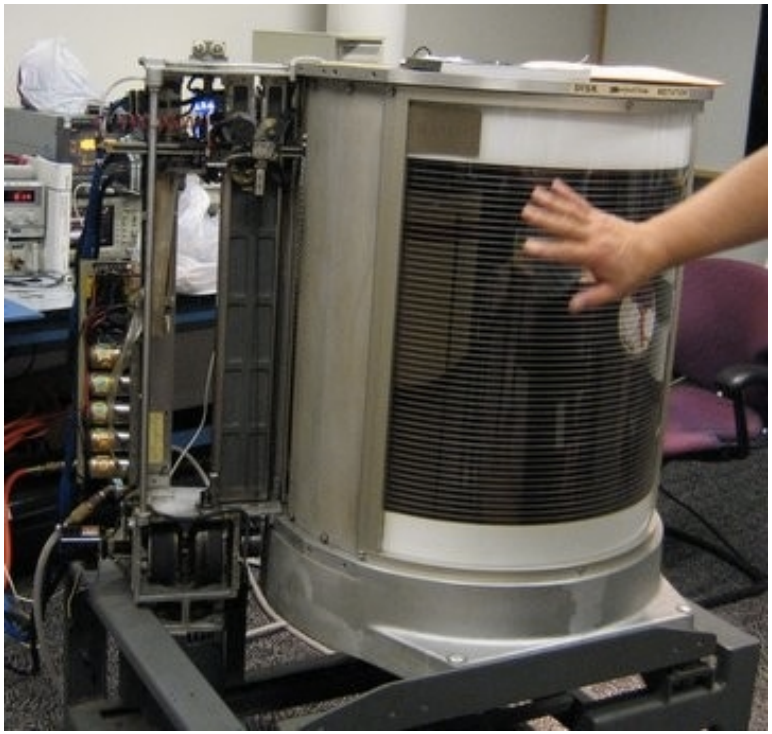


Figure 36.1: **Prototypical System Architecture**

存储也被称为“外存”。

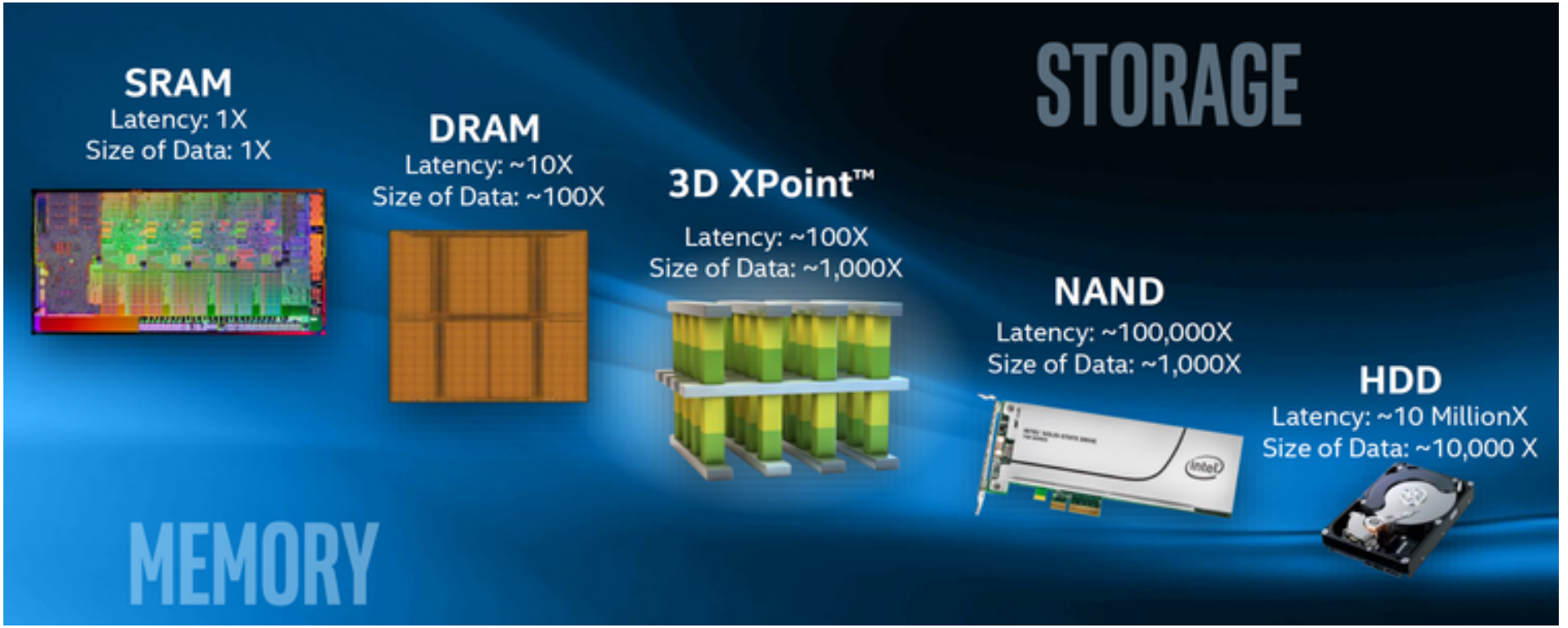
存储设备的总线是离内存总线较远的外设总线，比较慢。

存储技术的发展



50年代， IBM磁盘，容量5MB

存储技术的发展


SRAM
Latency: 1X
Size of Data: 1X
DRAM
Latency: ~10X
Size of Data: ~100X
3D XPoint™
Latency: ~100X
Size of Data: ~1,000X
NAND
Latency: ~100,000X
Size of Data: ~1,000X
HDD
Latency: ~10 MillionX
Size of Data: ~10,000 X
STORAGE
MEMORY

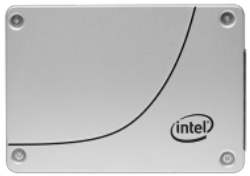
目录

- 连接更高速SSD — NVMe
  - NVMe 性能
  - NVMe 协议
  - 高性能给存储系统带来的挑战
  - NVMe性能优化
- 横跨内存与存储的 Persistent Memory
  - 内存、存储与PM
  - PM编程模型
  - 基于PM的存储系统设计
  - PMDK
- 新型分布式存储系统
  - RDMA
  - NVMe over Fabrics (NVMe + RDMA)
  - 分布式PM文件系统Octopus (PM + RDMA)

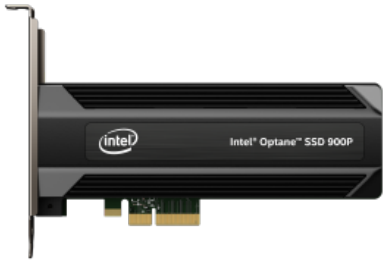
目录

- 连接更高速SSD — NVMe
  - NVMe 性能
  - NVMe 协议
  - 高性能给存储系统带来的挑战
  - NVMe性能优化
- 横跨内存与存储的 Persistent Memory
  - 内存、存储与PM
  - PM编程模型
  - 基于PM的存储系统设计
  - PMDK
- 新型分布式存储系统
  - RDMA
  - NVMe over Fabrics (NVMe + RDMA)
  - 分布式PM文件系统Octopus (PM + RDMA)

NVMe 性能 – 高带宽



SATA SSD



PCIe SSD

协议	接口	接口速度	存储介质
NVMe	PCIe	3.94 GB/s (x 4形式)	NAND Flash / 3D XPoint
AHCI	SATA	600 MB/s	NAND Flash

NVMe 性能 – 低延迟

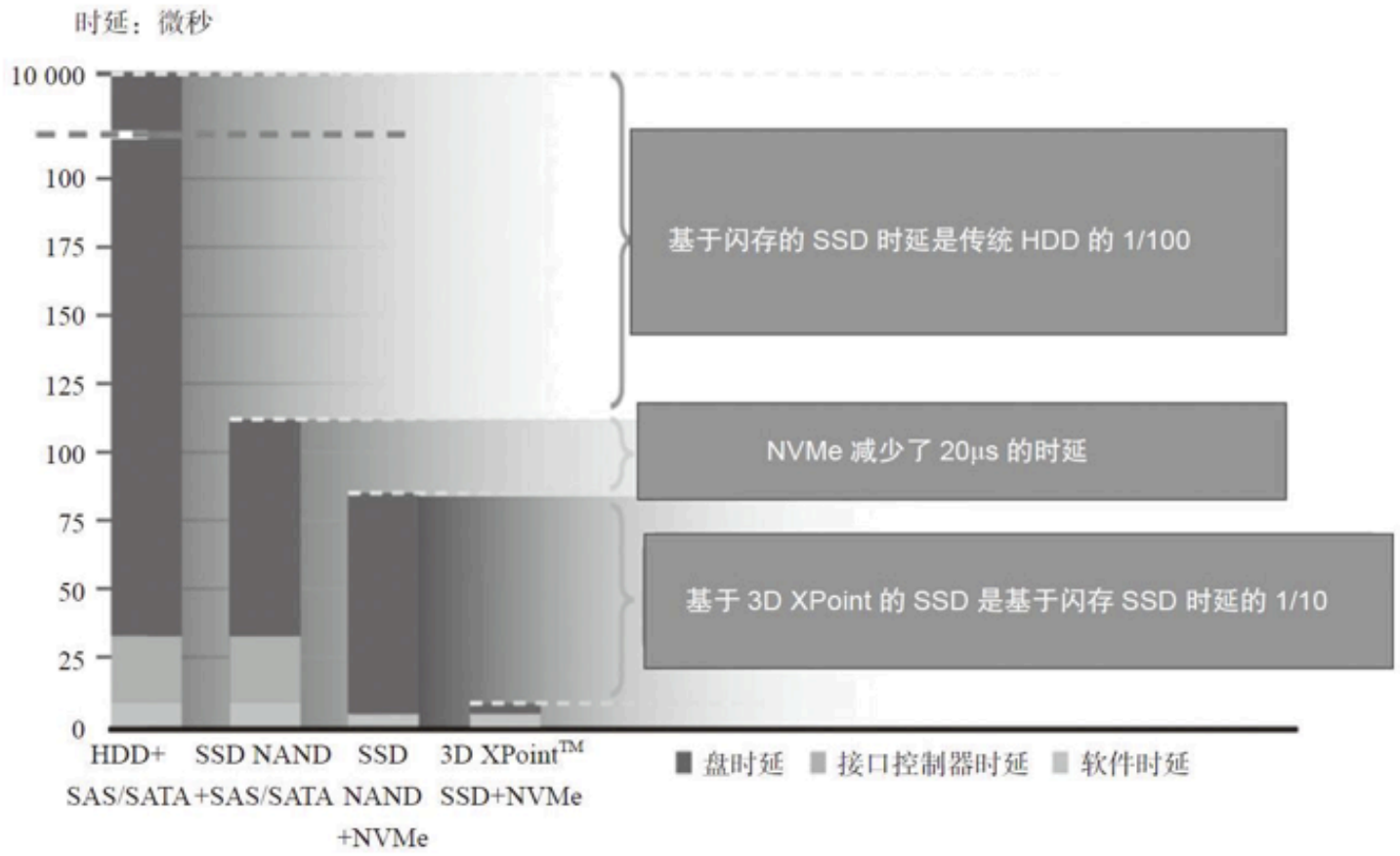
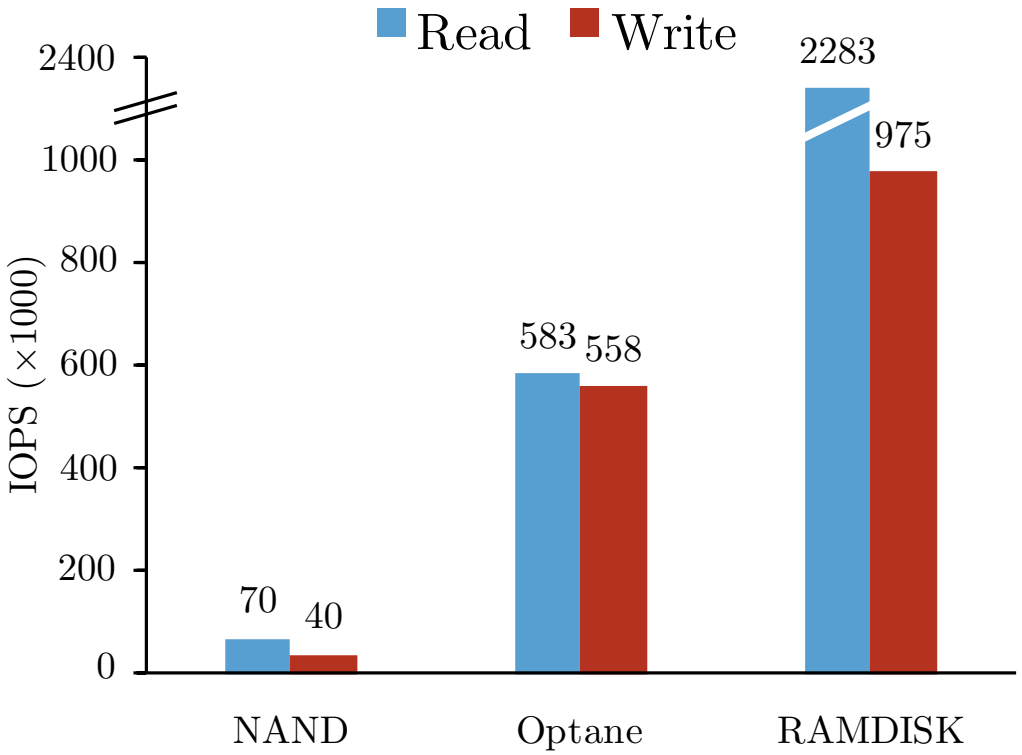


图6-1　时延对比

图摘自《深入浅出SSD：固态存储核心技术、原理与实战》

NVMe 性能 – 高并发



NAND SATA SSD 、3D XPoint NVMe SSD 和 内存模拟盘 的4K IOPS性能对比。

图来自：J. Zhang et al. “Performance Analysis of 3D XPoint SSDs in Virtualized and non-Virtualized Environments,” *2018 IEEE 24th Int. Conf. Parallel Distrib. Syst.*, pp. 51-60, 2018.

NVMe 协议 – 多队列

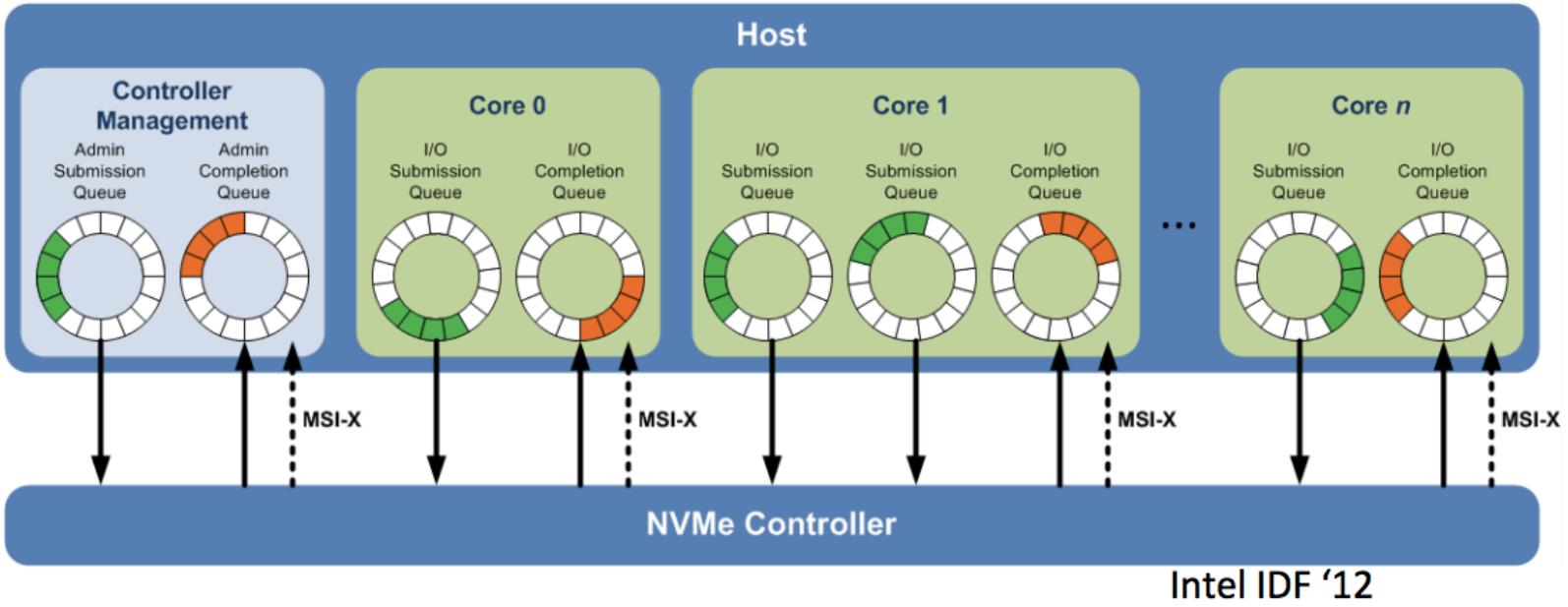
- NVMe 比AHCI的完成队列更多、更深：

协议	最大队列个数	最大队列深度
NVMe	65536	65536
AHCI	1	32

- NVMe 的队列分为4种：

  I/O SQ (Submission Queue)
  I/O CQ (Completion Queue)
  Admin SQ
  Admin CQ

NVMe 协议 – 多队列



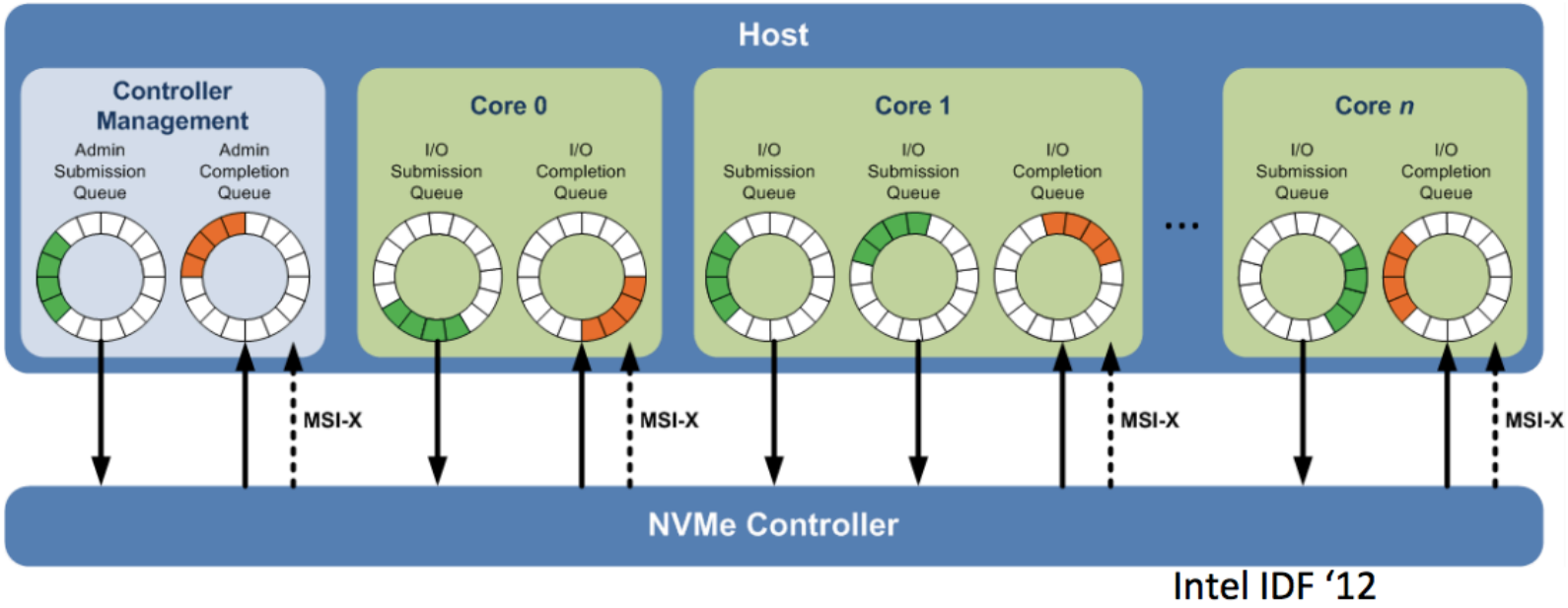
Intel IDF '12

- Admin SQ/CQ 每个系统只有一对。
- I/O SQ/CQ可以最多有65535对：

  每个物理核最多1个CQ；

  但每个核可以有多个SQ，因此可以实现线程粒度的不同优先级。
- SSD中还有称为Doorbell的寄存器，负责记录SQ/CQ的完成情况。

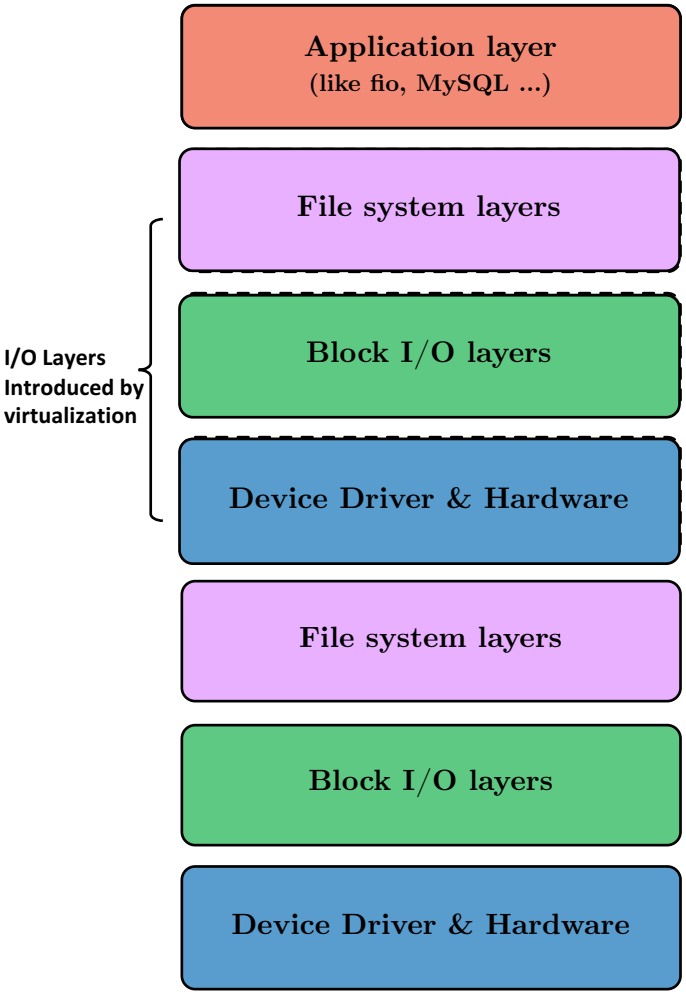
NVMe 协议 – 命令处理流程



1. 主机写命令到SQ； (Host写SQ)
2. 主机写SQ的DB，通知SSD取指； (Host通知SSD)
3. SSD到SQ中取指执行，并写执行结果到CQ； (SSD执行并写CQ)
4. 然后SSD发中断通知主机指令完成； (SSD通知Host)
5. 收到中断，主机处理CQ，查看指令完成状态； (Host处理完成)
6. 通过DB回复SSD：指令执行已完！ (Host通知SSD完成)

摘自《深入浅出SSD：固态存储核心技术、原理与实战》

高性能给存储系统带来的挑战



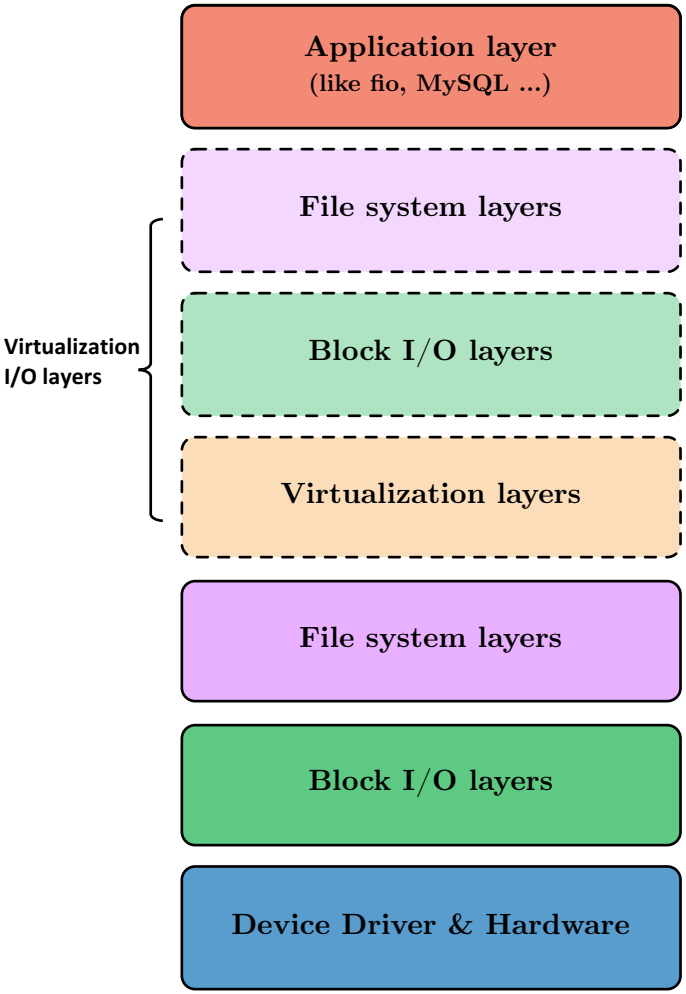
操作系统的存储IO栈本身就很复杂：

- I/O 请求会经过：application layer, file system layers, block I/O layers, device driver 最终到达硬件。

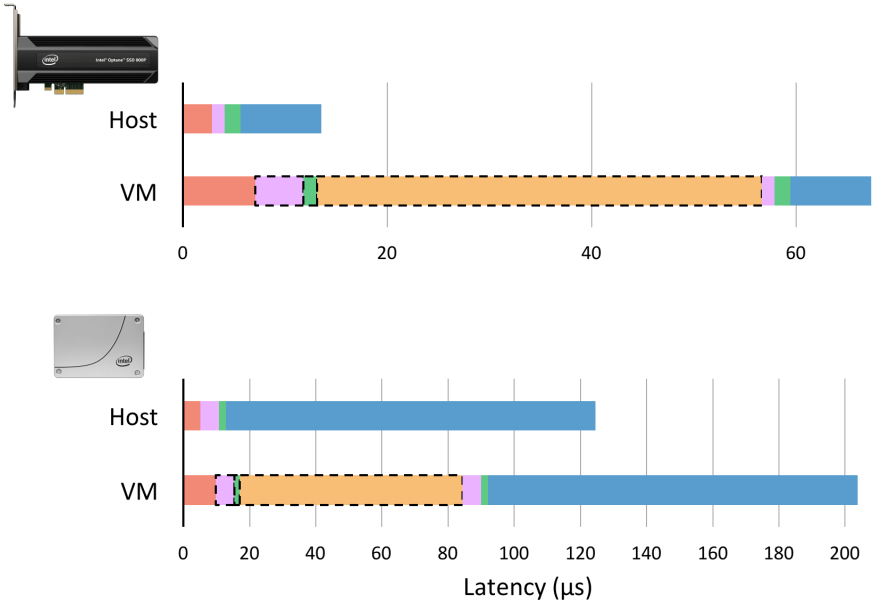
IO路径在虚拟化环境中基本上翻倍了：

- 首先，QEMU等虚拟化管理器引入了很多层次用于模拟磁盘设备或管理镜像文件；
- 其次，Guest OS有引入了额外的block layers和file system layers。

高性能给存储系统带来的挑战



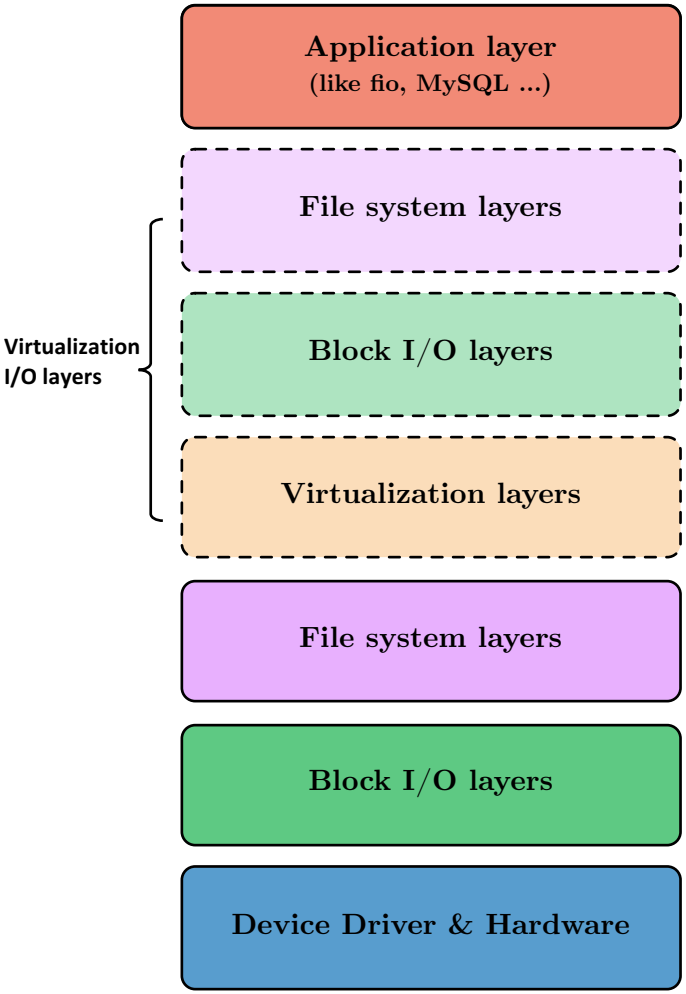
Latency breakdown: (Test env. : Fio 4K read, ext4, Linux, QEMU)

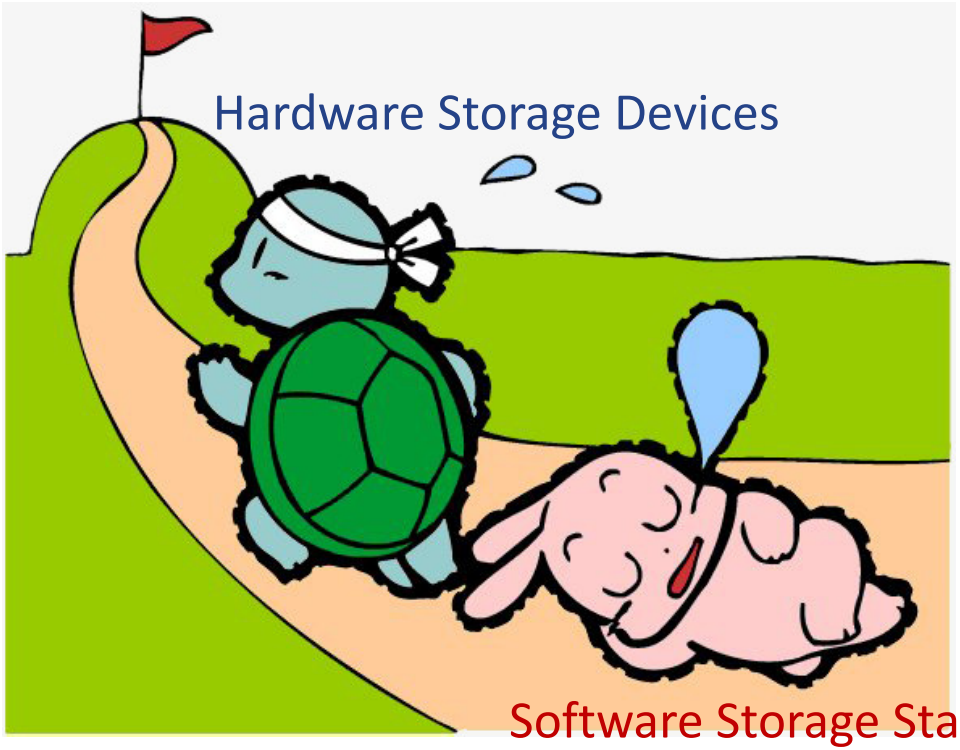


对于NVMe接口的 Optane SSD (上图):

- 硬件部分占了更小部分延迟；(蓝色部分)
- Virtualization引入的层次占得比例最大。(虚线框中)

高性能给存储系统带来的挑战





Stop sleeping, hardware is catching up!

(存储系统的软件优化更为重要！)

NVMe性能优化 – 用户态NVMe驱动SPDK

SPDK(Storage Performance Development Kit)基于DPDK(Data Plane Development Kit)。

(DPDK基于UIO或者VFIO。UIO和VFIO都支持在用户态开发IO设备驱动程序。)

SPDK的核心就是一套用户态的NVMe驱动，利用一下手段减少延迟：

- 减少了内核存储栈层次；

  (SPDK IO Path: 应用层→SPDK→NVMe硬件)

- 为每个线程创建一对Queue Pair而不是每个核，减少锁竞争；

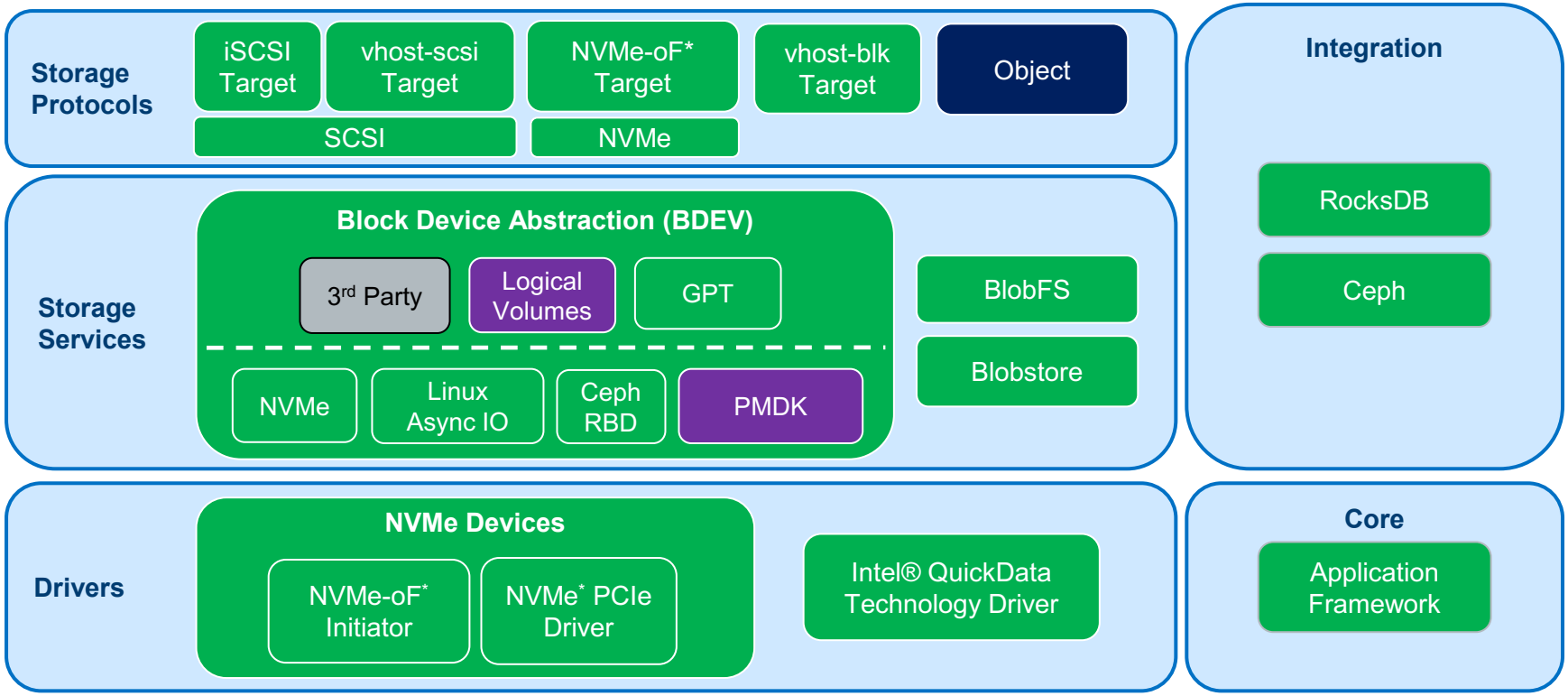
  (相反，内核默认为每个物理核创建一对Queue Pair)

- Polling 而非 interrupt，降低延迟。

  (SPDK以一个专用核来进行polling)

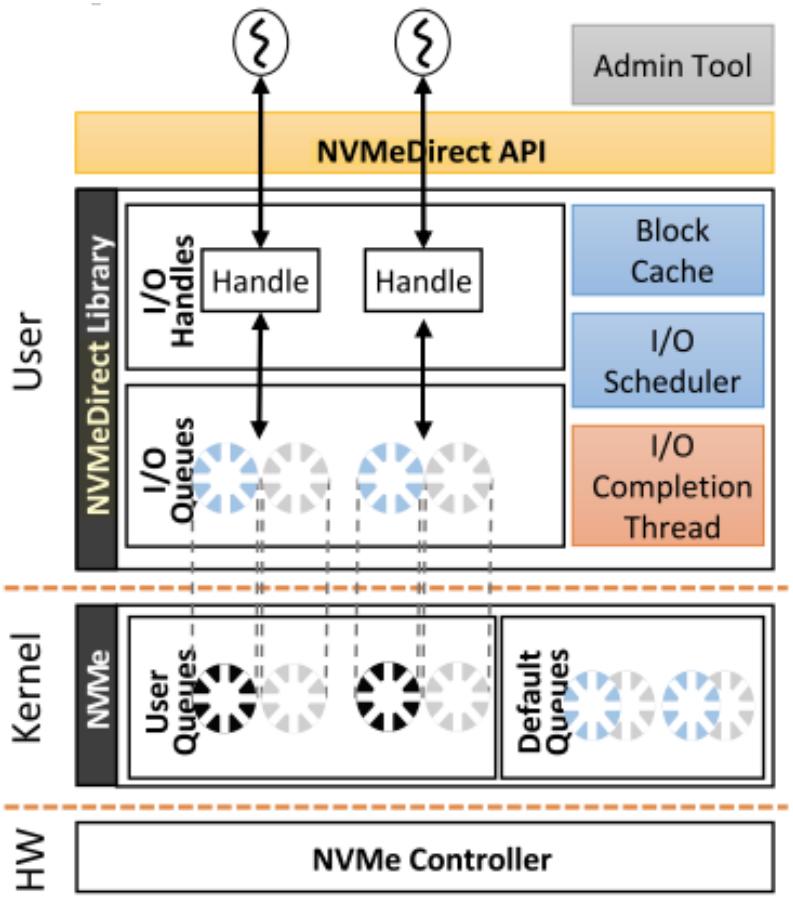
图摘自《深入浅出SSD：固态存储核心技术、原理与实战》

NVMe性能优化 – 用户态NVMe驱动SPDK



SPDK 整体架构

NVMe性能优化 – 用户态驱动NVMeDirect



NVMeDirect也是一种NVMe用户态驱动。

将内核中的Queue Pairs映射到用户态的NVMeDirect库中。

与SPDK不同的是，它利用了内核原有NVMe驱动作为控制层。

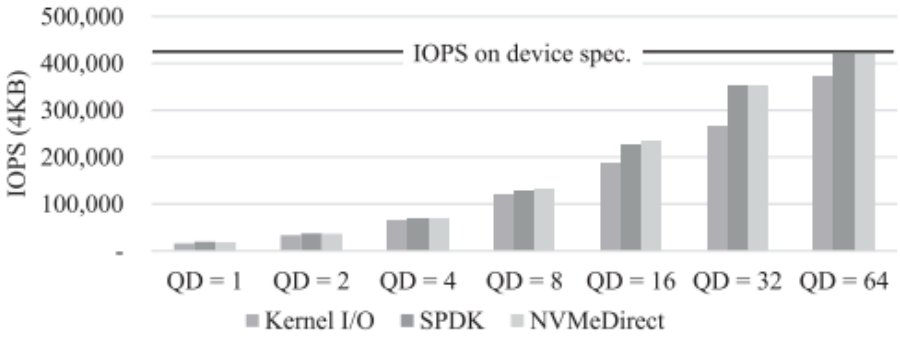
图来自： NVMeDirect 2.0: An Enhanced User-space I/O Framework for NVMe SSDs

NVMe性能优化 – 用户态NVMe驱动

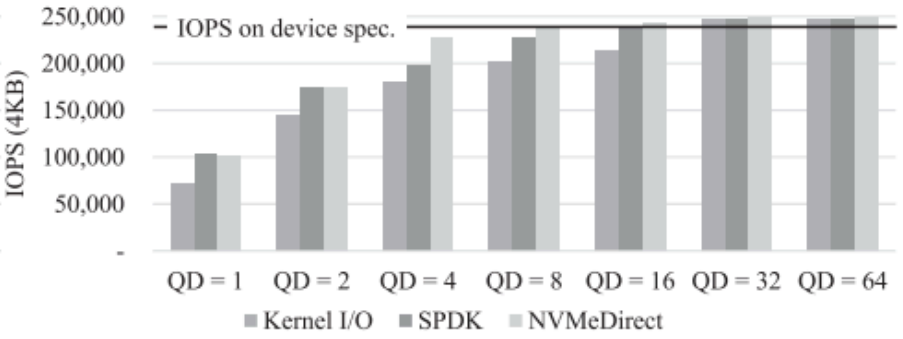
	NVMeDirect 2.0	SPDK
Developed by	Computer System Labs @ SKKU	Intel
Key components	• Memory mapping • Ioctl()	• DPDK • libpciaccess
	• Access in user-space • **Partition-level management** • **Co-existence with kernel driver**	• Access in user-space • **Device-level management**
Framework features	• **User-level filesystem: ForestFS** - Supports directory structures - Supports SYNC/ASYN APIS - Supports writing to any offset of file • **User-level page cache** • **I/O function wrapper** • **Support for various I/O engines** - **Can be used with SPDK storage engine**	• **User-level filesystem: BlobFS** - Supports only a flat namespace - Supports SYNC API only - Do not support in-place update • **NVMe-oF support**

两种用户态NVMe驱动的对比

NVMe性能优化 – 用户态NVMe驱动



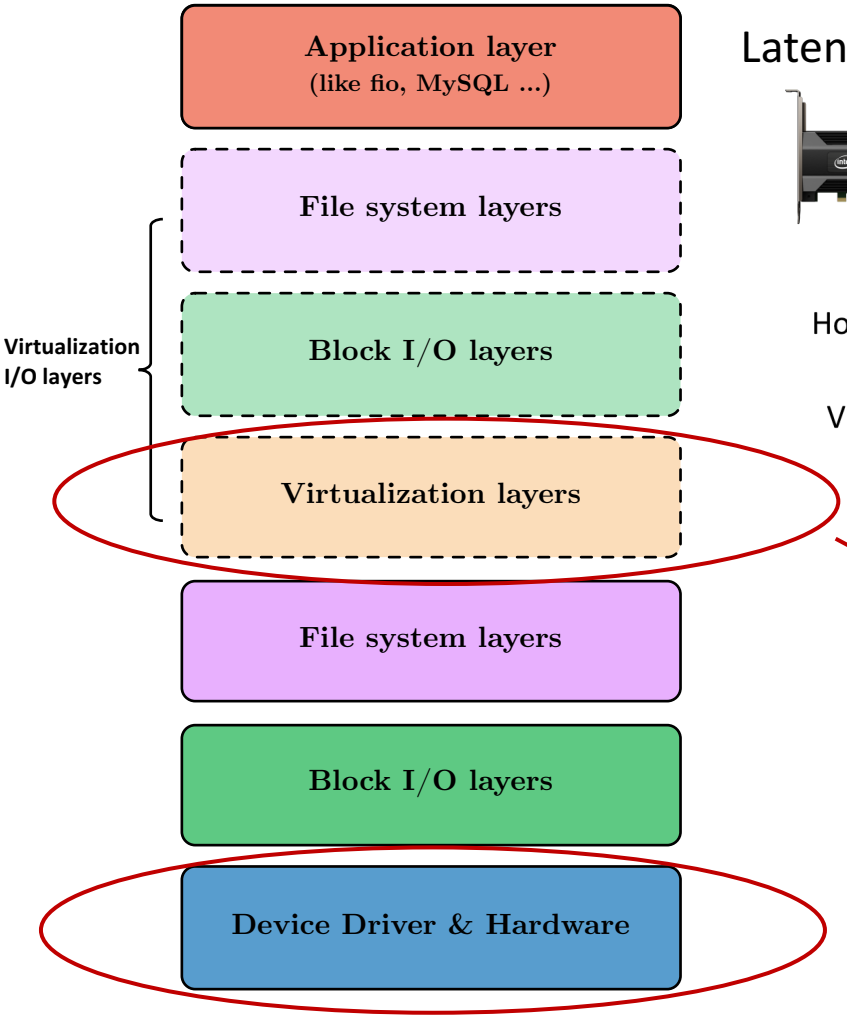
(a) Random read performance comparison.



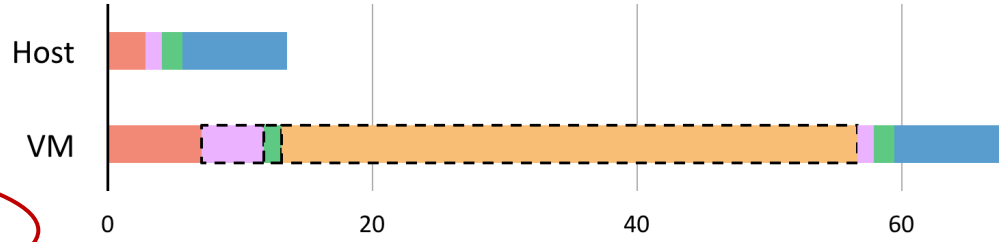
(b) Random write performance comparison.

[1]H. Kim, Y.-S. Lee, and J.-S. Kim, “NVMeDirect: A User-space I/O Framework for Application-specific Optimization on NVMe SSDs,” *Hotstorage*, 2016.

NVMe性能优化 – 虚拟环境



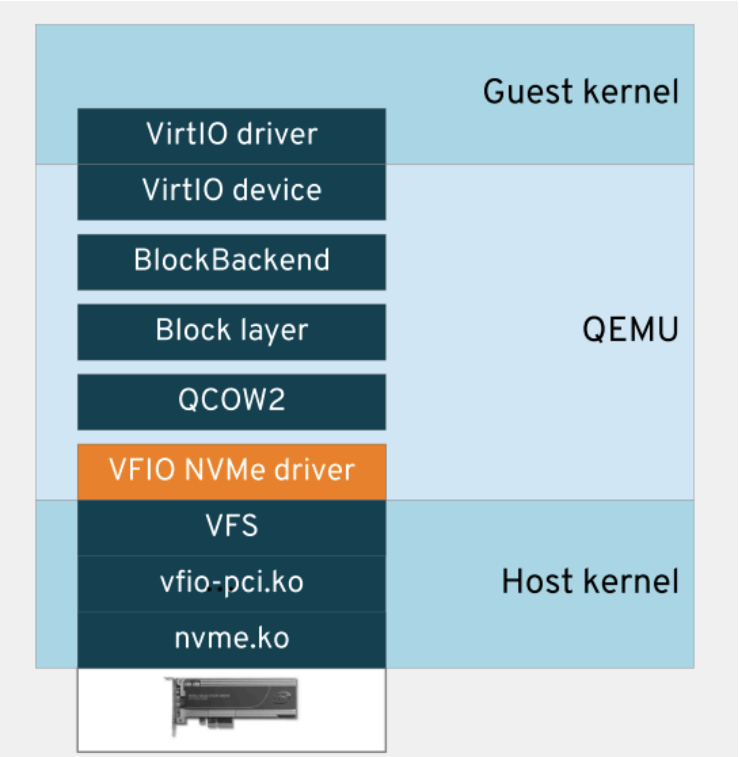
Latency breakdown: (Test env. : Fio 4K read, ext4, Linux, QEMU)



虚拟化存储性能的主要瓶颈在这。

SPDK等用户态NVMe优化的

NVMe性能优化 – 虚拟环境



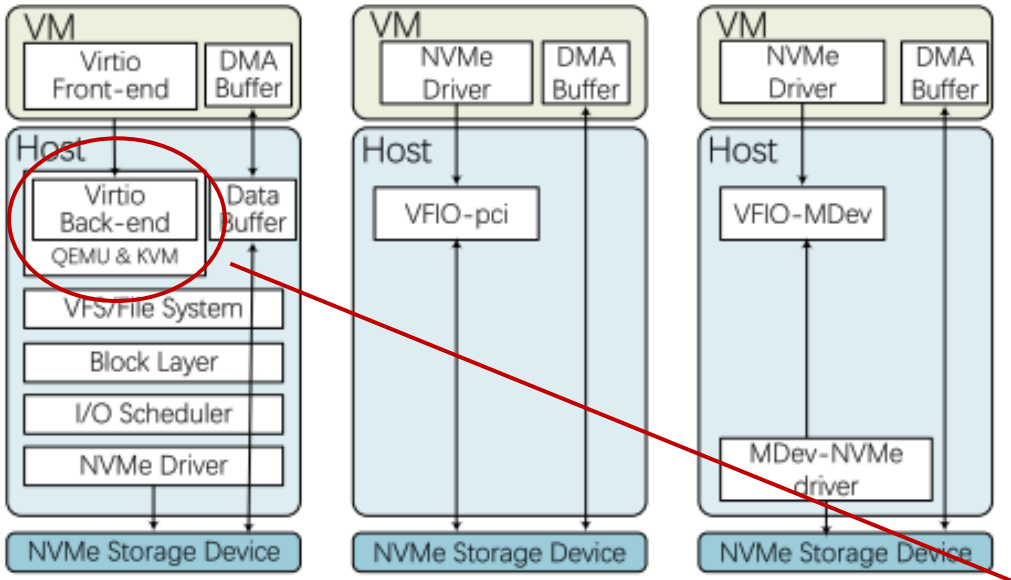
IOPS Improvement over Linux-aio

(IOPS)	Relative
rand-read-1-req	+12%
rand-read-4-req	+20%
rand-write-1-req	+22%
rand-write-4-req	+12%
rand-rw-1-req	+3%
rand-rw-4-req	+22%

还是将操作系统内核NVMe驱动替换为用户态的VFIO驱动，
但这个驱动被集成到QEMU中。

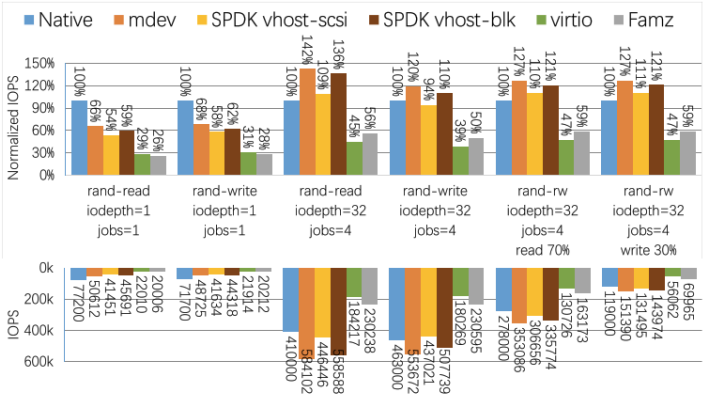
[1] F. Zheng and S. S. Engineer, “Userspace NVMe Driver in QEMU,” *KVM Forum*, 2017.

NVMe性能优化 – 虚拟环境



(a) Virtio (b) VFIO (c) MDev-NVMe

Figure 1: NVMe virtualization comparisons.



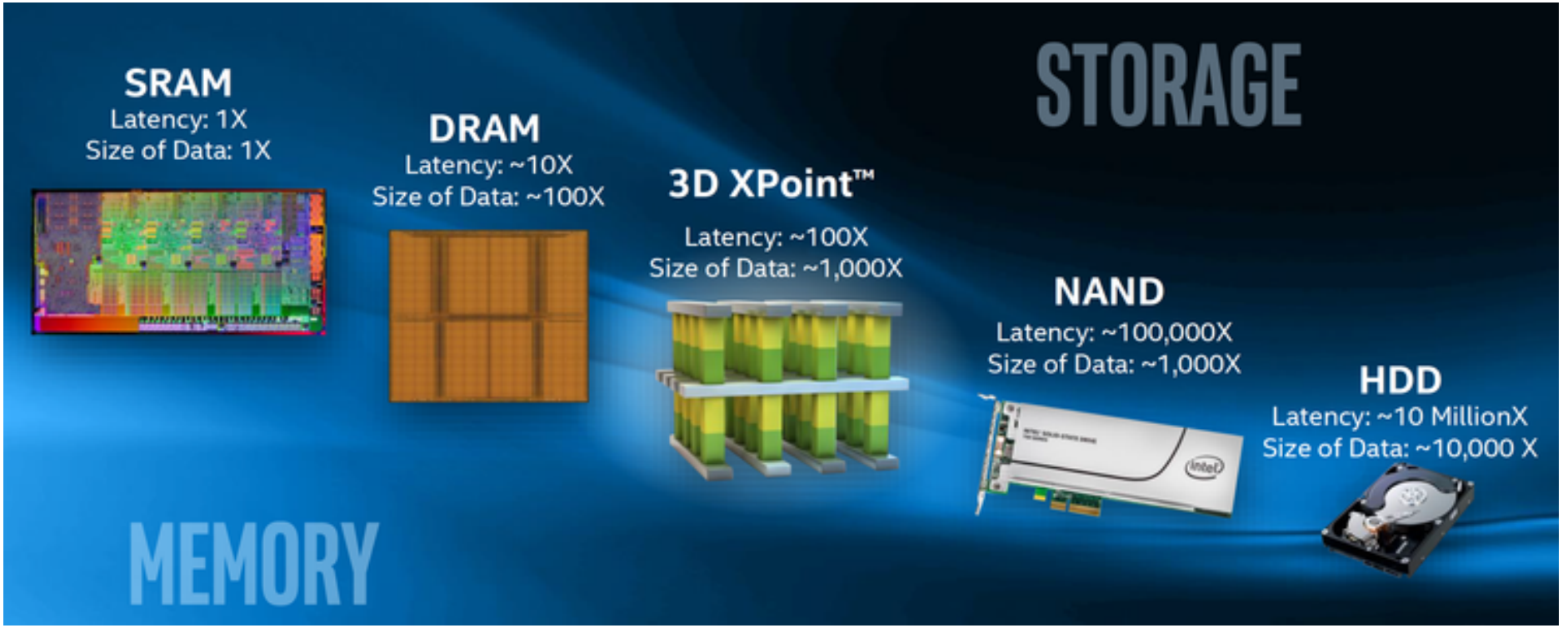
这种性能很好，但是去掉了QEMU的镜像管理层。

[1] B. Peng, “MDev-NVMe : A NVMe Storage Virtualization Solution with Mediated,” 2018.

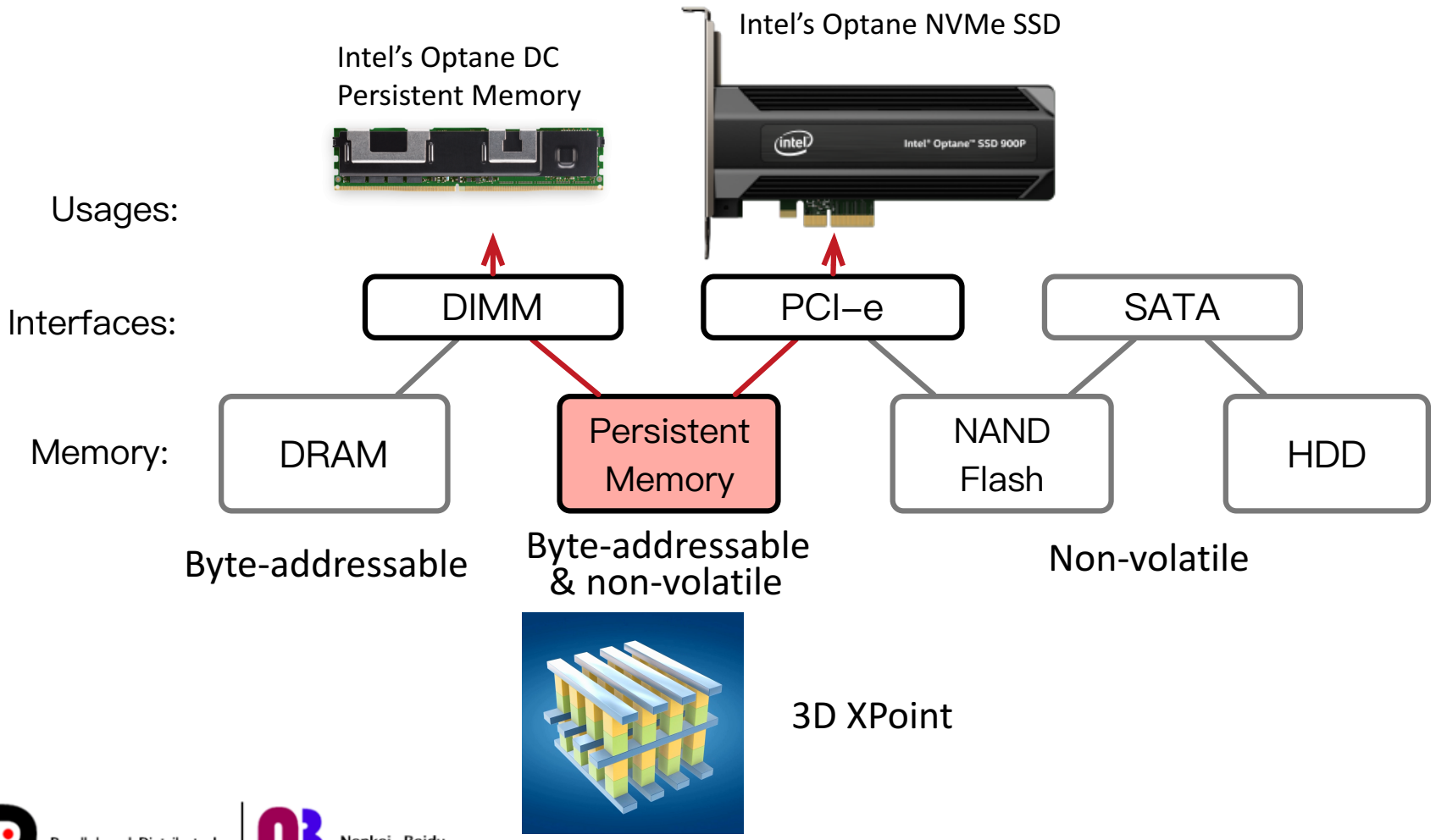
目录

- 连接更高速SSD — NVMe
  - NVMe 性能
  - NVMe 协议
  - 高性能给存储系统带来的挑战
  - NVMe性能优化
- 横跨内存与存储的 Persistent Memory
  - 内存、存储与PM
  - PM编程模型
  - 基于PM的存储系统设计
  - PMDK
- 新型分布式存储系统
  - RDMA
  - NVMe over Fabrics (NVMe + RDMA)
  - 分布式PM文件系统Octopus (PM + RDMA)

内存、存储和Persistent Memory



内存、存储和Persistent Memory



Non-Volatile Memory (NVM)

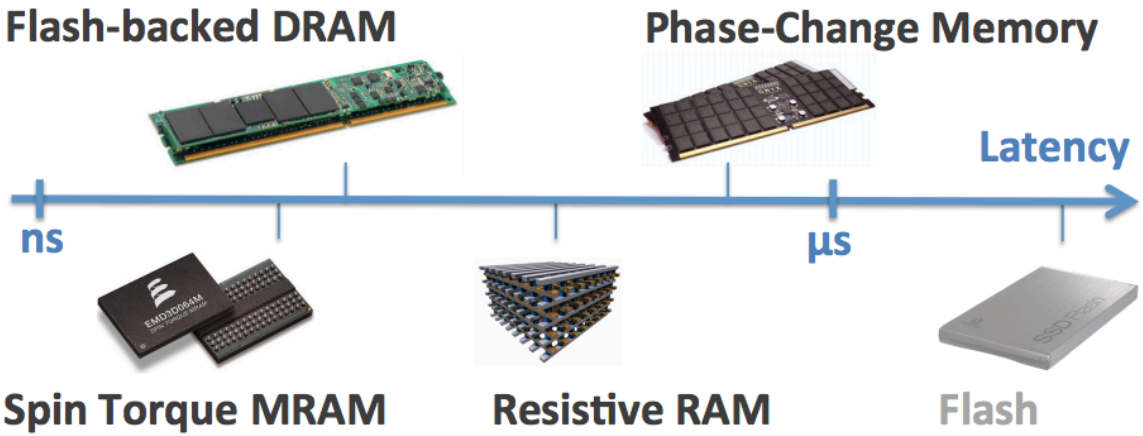
- 很多种别名：

Non-Volatile Memory (NVM)	广义上指非易失存储介质； 狭义上必须连接到DIMM接口。
NVDIMM	多指DRAM+电池+Flash。
Persistent Memory (PM)	一般指DIMM接口的NVM。
Storage Class Memory (SCM)	一般指DIMM接口的NVM。

- 很多种NVM技术：

种类	评价
DRAM + Flash + 电池	速度和DRAM一样，昂贵，做不大。
PCM	基于相变的非易失介质，比DRAM稍慢。
Memristor	忆阻器，比DRAM稍慢。
3D XPoint	最接近商用，有人怀疑就是PCM技术。

Non-Volatile Memory (NVM)



Non-volatile

Volatile

特性	PCM	EEPROM	NOR Flash	NAND Flash	DRAM
掉电数据不丢失	是	是	是	是	否
支持字节级访问	是	是	是	否	是
需要擦除	否	否	是	是	否
软件复杂性	简单	简单	一般	复杂	简单
功耗	低	低	低	低	高
写带宽	1 ～ 15MB/s	13 ～ 30KB/s	0.5 ～ 2MB/s	>10MB/s	>100MB/s
读延迟	50 ～ 100ns	200ns	70 ～ 100ns	15 ～ 50µs	20 ～ 80ns
写次数	100 万次	10 万～ 100 万次	10 万次	1000 ～ 10 万次	无限次

[1] 《深入浅出SSD：固态存储核心技术、原理与实战》

Persistent Memory – NAND Flash Memory原理

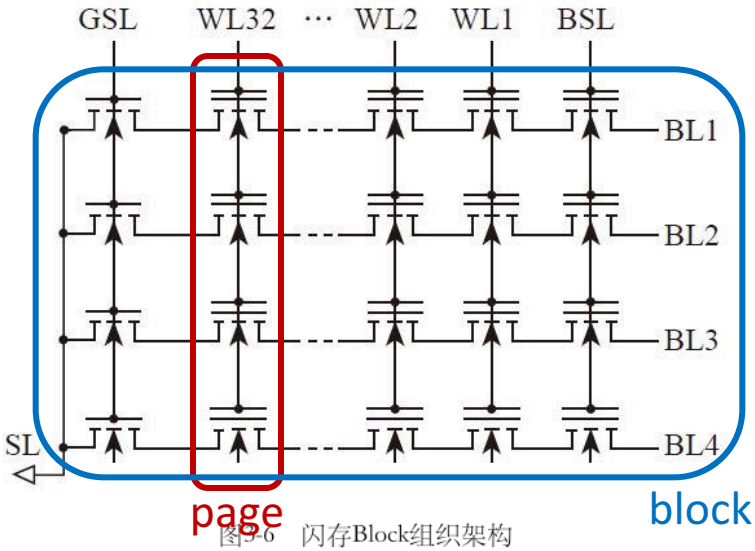


图3-6 闪存Block组织架构

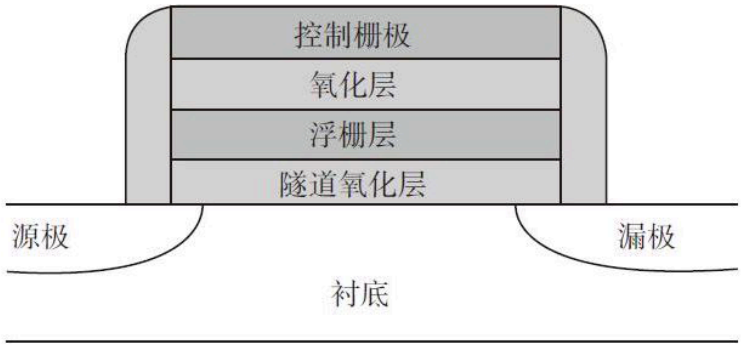


图3-1 浮栅晶体管结构

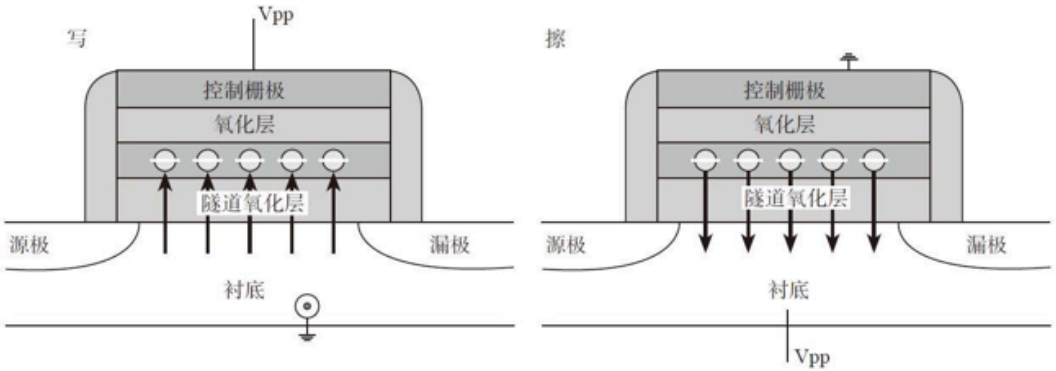


图3-2 左：写原理；右：擦除原理

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》

Persistent Memory – Phase Change Memory原理

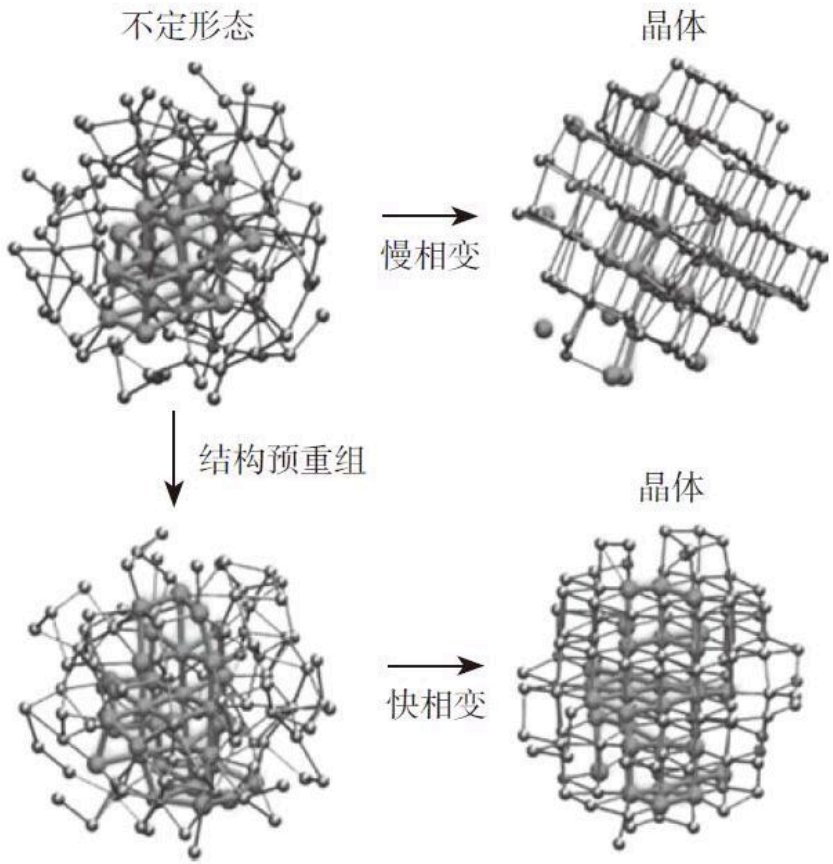


图3-25　相变晶体原理

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

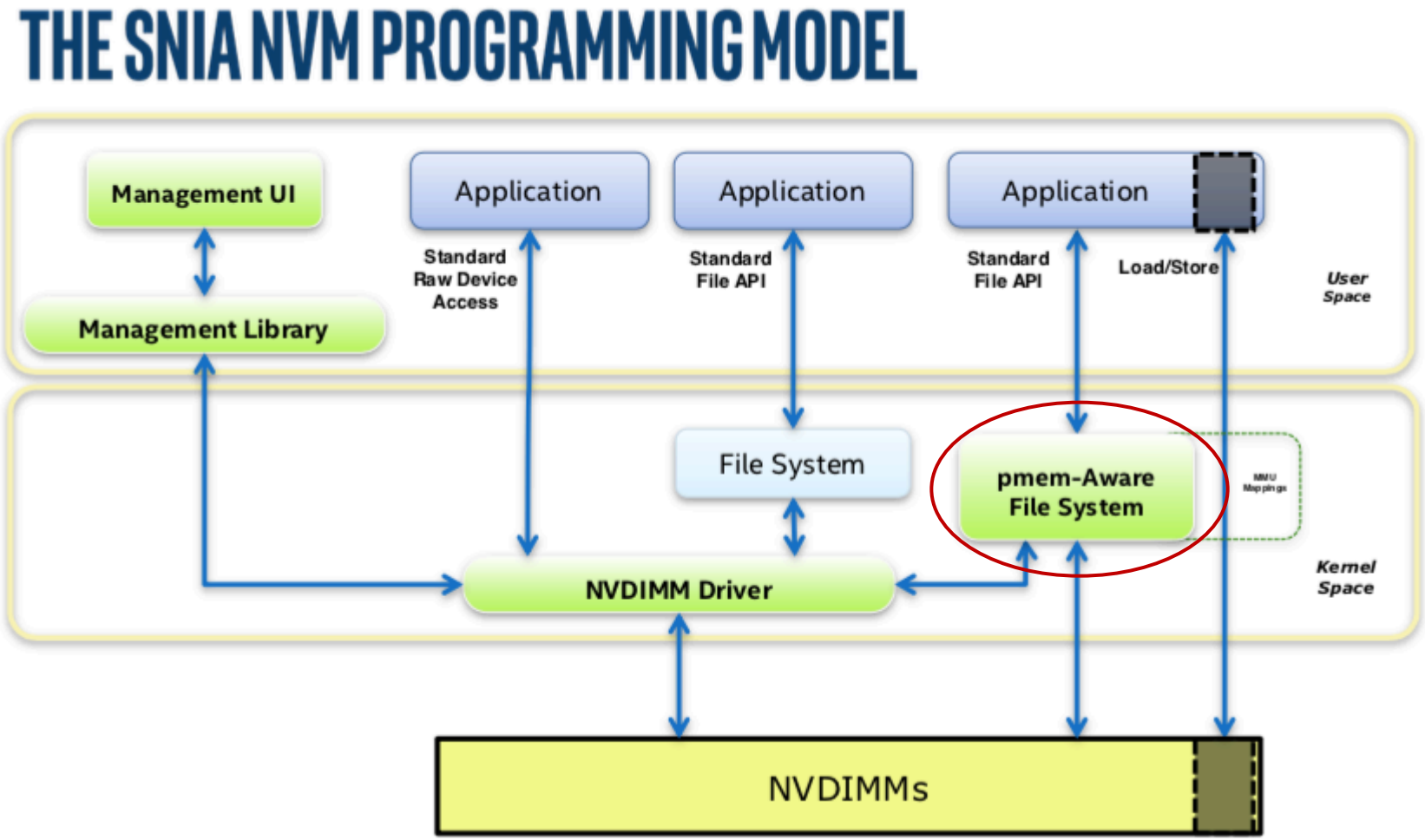
一般用准金属合金GST以各种方式加热和冷却材料来操纵相态之间的变化来实现信息的存储。

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编程模型 – PM与块设备



摘自 Paul Luse 《SPDK & PMDK: Two Open Source Development Kits for the Evolving NVM Landscape》

PM编程模型 – PM-aware文件系统

表 V: PM 感知文件系统相关工作

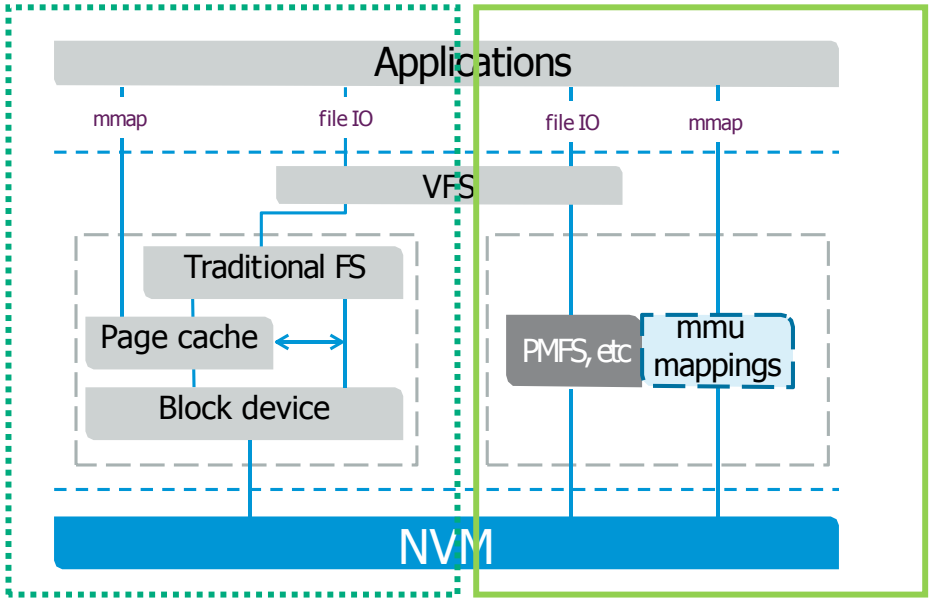
	崩溃一致性模型
PMFS [20]	事务型，logging
NOVA [17]	事务型，log-structured
SoupFS [15]	顺序写，soft-update
ext4-DAX	事务型，logging
XFS-DAX	事务型，logging
Octopus [16] (分布式)	事务型，本地 logging

- 普通文件系统我们常用的文件接口：
  open → read/write → fsync
- PM-aware(DAX)文件系统的接口：
  open → mmap → memcpy → msync

PM编程模型 – PM-aware文件系统

通过RAMDISK模拟块设备的形式兼容传统文件系统

基于持久性内存重新构建字节粒度的持久性文件系统



- 普通文件系统我们常用的文件接口：
  open → read/write → fsync
- PM-aware(DAX)文件系统的接口：
  open → mmap → memcpy → msync

图摘自舒继武《大数据时代的存储系统若干变化的思考》

PM存储系统设计问题 – 相关工作

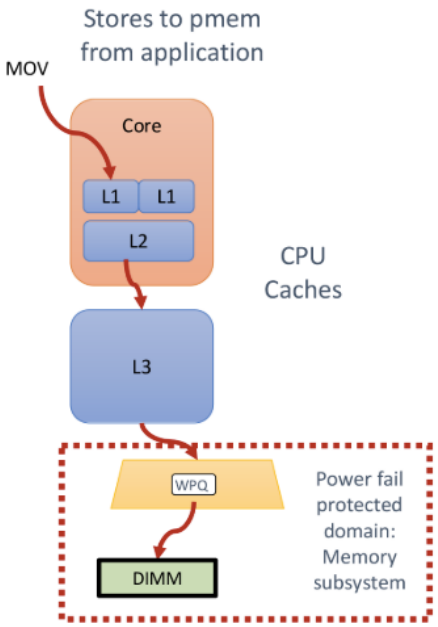
近年PM的相关工作很多。

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。

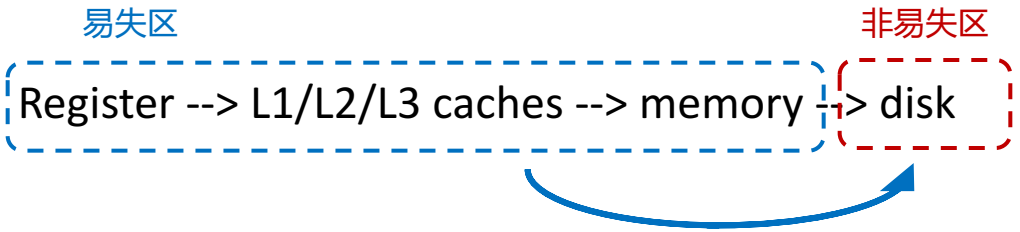
表 I: 相关工作分类

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：

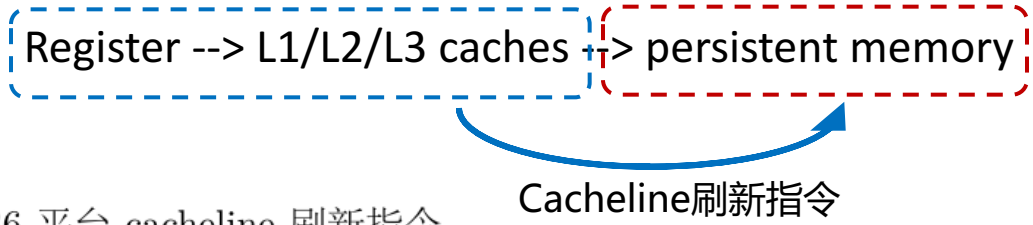


表 III: x86 平台 cacheline 刷新指令

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 – 原子可见性

为保证某段内存在某个时间仅被一个线程访问，需要注意的是原子可见性的问题(Atomicity或Visibility)，当前x86平台可以实现以下粒度的原子可见性：

8 字节原子更新 64 位处理器的 store 命令都是 8 字节粒度的原子更新。

16 字节原子更新 现代处理器可以使用加 LOCK 前缀的 cmpxchg16b 指令实现 16 字节的原子写。

64 字节 (cacheline 粒度) 原子更新 利用硬件事务型内存 (hardware transactional memory, HTM) 可以在事务内保证可以保证某个 cacheline 的所有数据 不被刷入内存。
(要原子地更新某个 cacheline，首先用 XBEGIN 开始一个 RTM 事务，然后在是事务内对目标 cacheline 进行所需的修改，最后写 XEND 结束这个 事务。在事务完成后，再进行 clflush 等命令，保证被修 改过的 cacheline 就被原子地写回 PM。)

PM存储系统设计问题 – 崩溃一致性

存储系统的崩溃一致性：

崩溃原子性(Failure Atomicity)问题，又可以称为崩溃一致性(Crash Consistency)问题。此问题是在设计存储系统中一定会遇到的问题。
(举个例子：Linux ext4等文件系统称为日志文件系统，因为它靠日志机制来保证崩溃一致性)

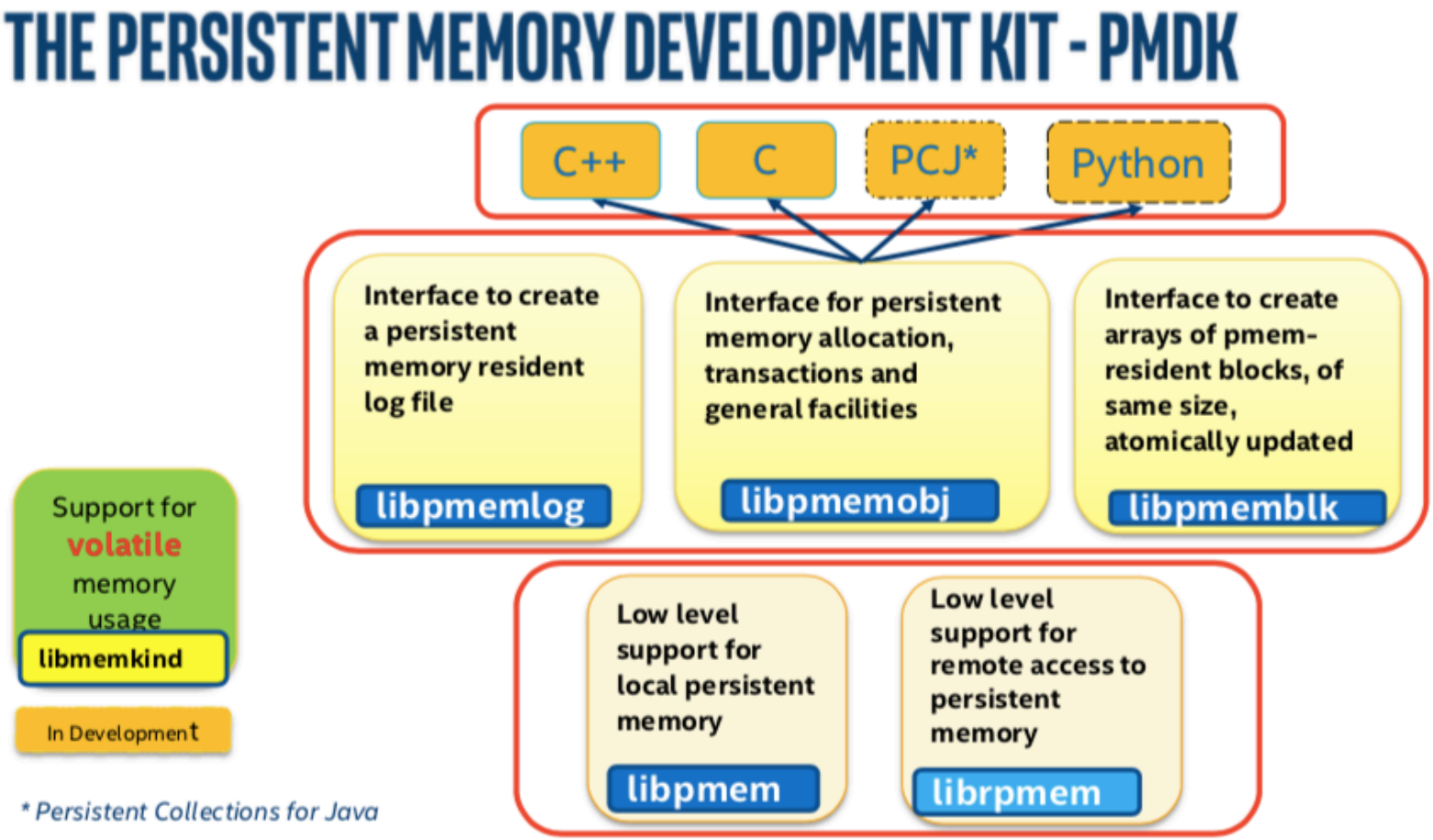
基于PM的存储系统的崩溃一致性：

x86平台的Failure atomicity 仅为8字节。注意：虽然cache以64字节的cacheline大小为粒度进行cache到内存的数据传输，但硬件上仍然分8次(每次8字节)进行传输，所以即使是同一cacheline，也会被系统崩溃或断电打断。

因此，任何大于8字节的持久化操作，都要用额外的崩溃一致性机制保证整个存储一系统的一致性。

常用的方法有顺序写、日志、日志结构等方法。

PMDK



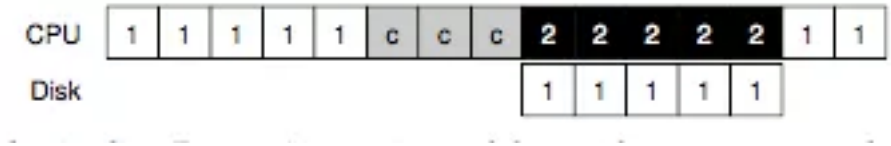
摘自 Paul Luse 《SPDK & PMDK: Two Open Source Development Kits for the Evolving NVM Landscape》

目录

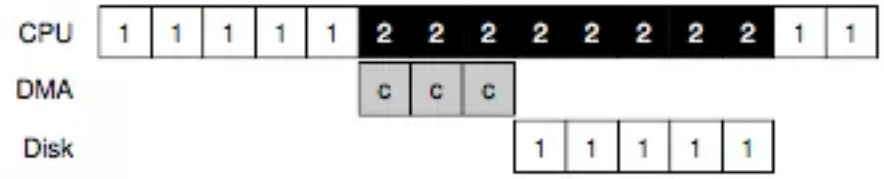
- 连接更高速SSD — NVMe
  - NVMe 性能
  - NVMe 协议
  - 高性能给存储系统带来的挑战
  - NVMe性能优化
- 横跨内存与存储的 Persistent Memory
  - 内存、存储与PM
  - PM编程模型
  - 基于PM的存储系统设计
  - PMDK
- 新型分布式存储系统
  - RDMA
  - NVMe over Fabrics (NVMe + RDMA)
  - 分布式PM文件系统Octopus (PM + RDMA)

先回顾一下DMA – 作用

DMA即Direct Memory Access，它把CPU从I/O传输中“解放”出来。



没有DMA：数据传输由CPU负责。



有DMA：数据传输从CPU卸载到DMA。

1：要进行IO的进程1。
2：另外一个进程2。
c：以字长进行设备和内存间的数据传输

图片来自：《**Operating Systems: Three Easy Pieces**》, Remzi H. Arpaci–Dusseau and Andrea C., Arpaci–Dusseau, Arpaci–Dusseau Books , March, 2015 (Version 1.00)

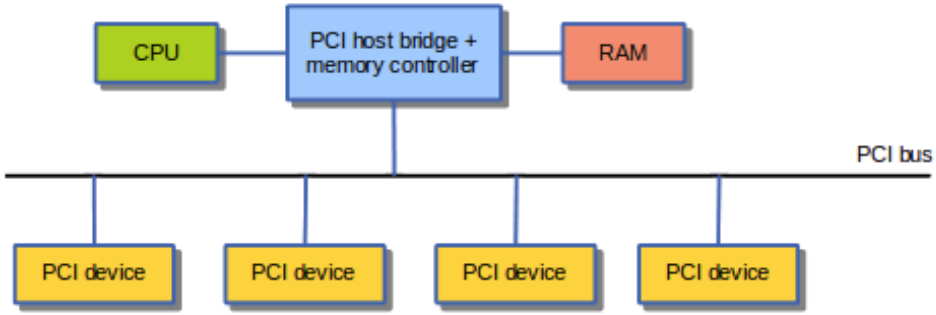
先回顾一下DMA – 历史

80年代：

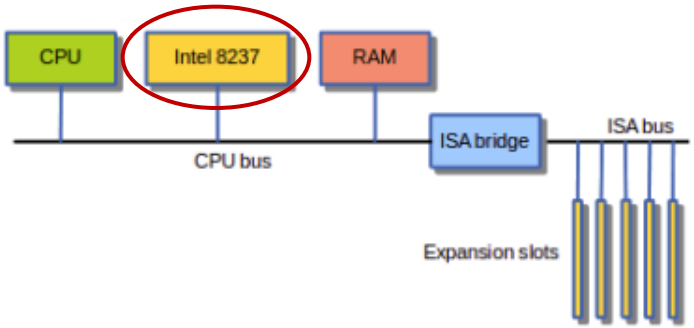
专门的DMA controller(如Intel 8237)服务于所有使用DMA的IO设备。

90年代及现在：

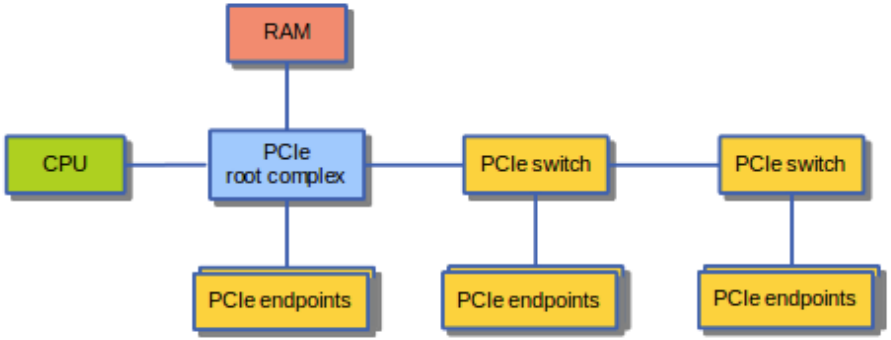
PCI及PCIe总线上不再有专门的DMA controller，DMA engine位于各个设备上。



PCI 总线(1992年)



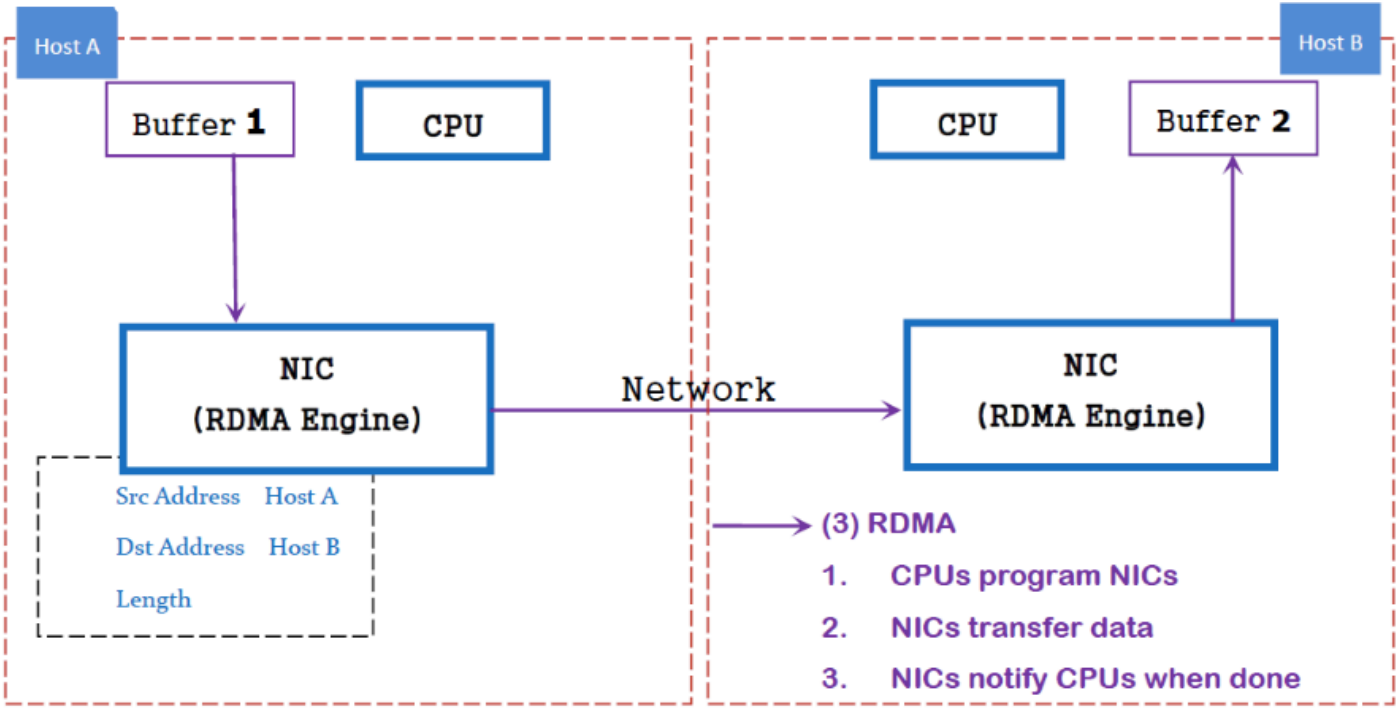
ISA 总线(1981年)



PCI 总线(2004年)

参考：https://geidav.wordpress.com/2014/04/27/an-overview-of-direct-memory-access/

RDMA – 远程的(Remote)DMA



与TCP/IP相比：

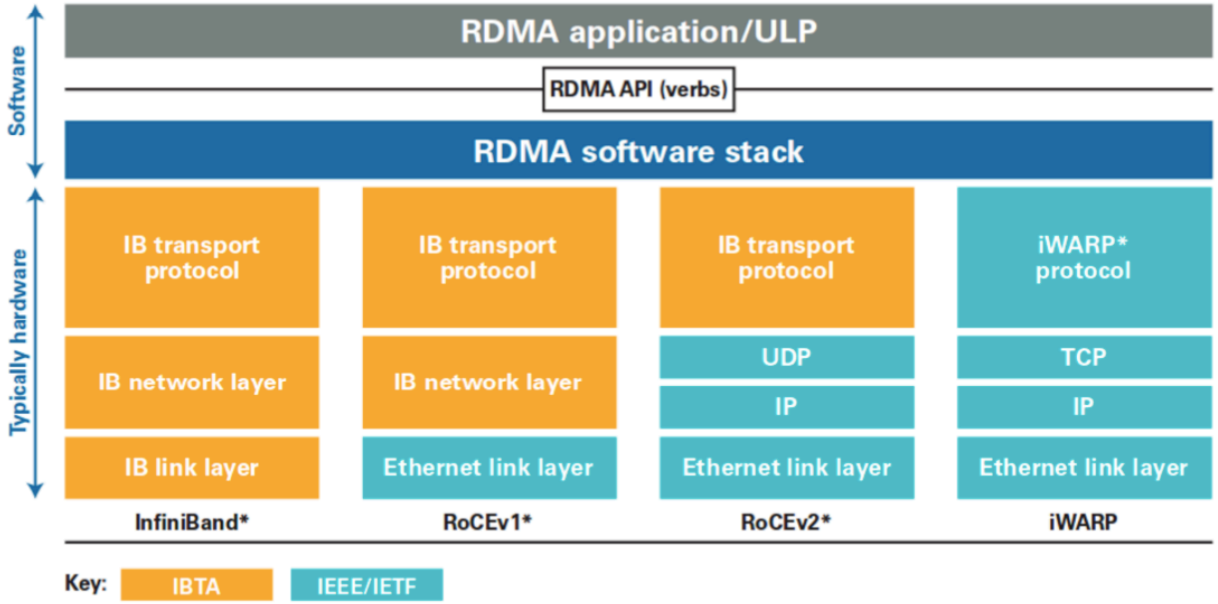
零拷贝：数据直接从一个节点的虚拟内存拷贝到另一个节点的虚拟内存。

绕过内核：操作系统不用参与数据在不同节点间的传输。

异步：线程不会被IO传输所阻塞。

图来自：https://www.cnblogs.com/vlhn/p/7909893.html

RDMA – 硬件和协议



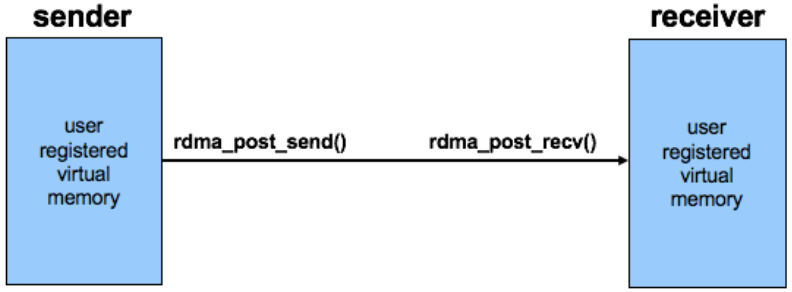
技术	硬件	性能
InfiniBand	特殊网卡和交换机	基于IB协议，性能最好
RoCE	特殊网卡	基于UDP
iWARP	特殊网卡	基于TCP
软件模拟iWARP	无须特殊硬件	性能最差

图来自：https://www.cnblogs.com/vlhn/p/7909893.html

RDMA – 传输类型

❖ **SEND/RECV – similar to “normal” TCP sockets**

–each send on one side must match a recv on other side

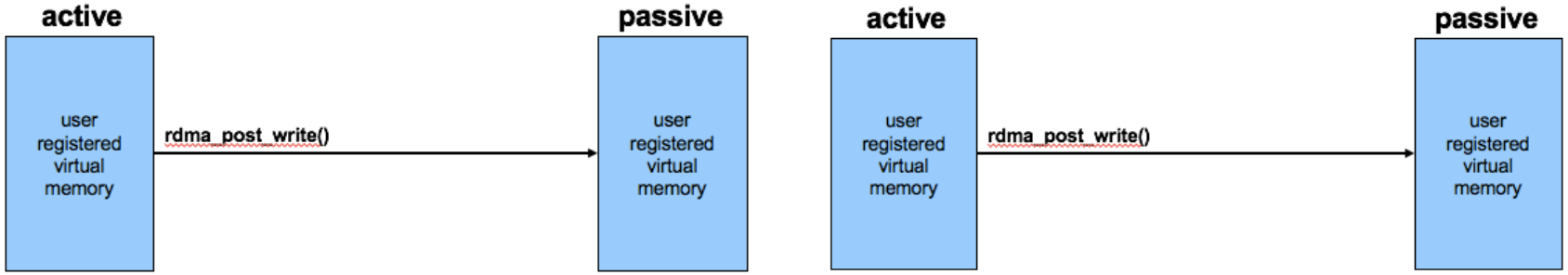


❖ **WRITE/READ – only in RDMA**

(绕过passive端的CPU和内核, passive 端性能是active端的5倍[2])

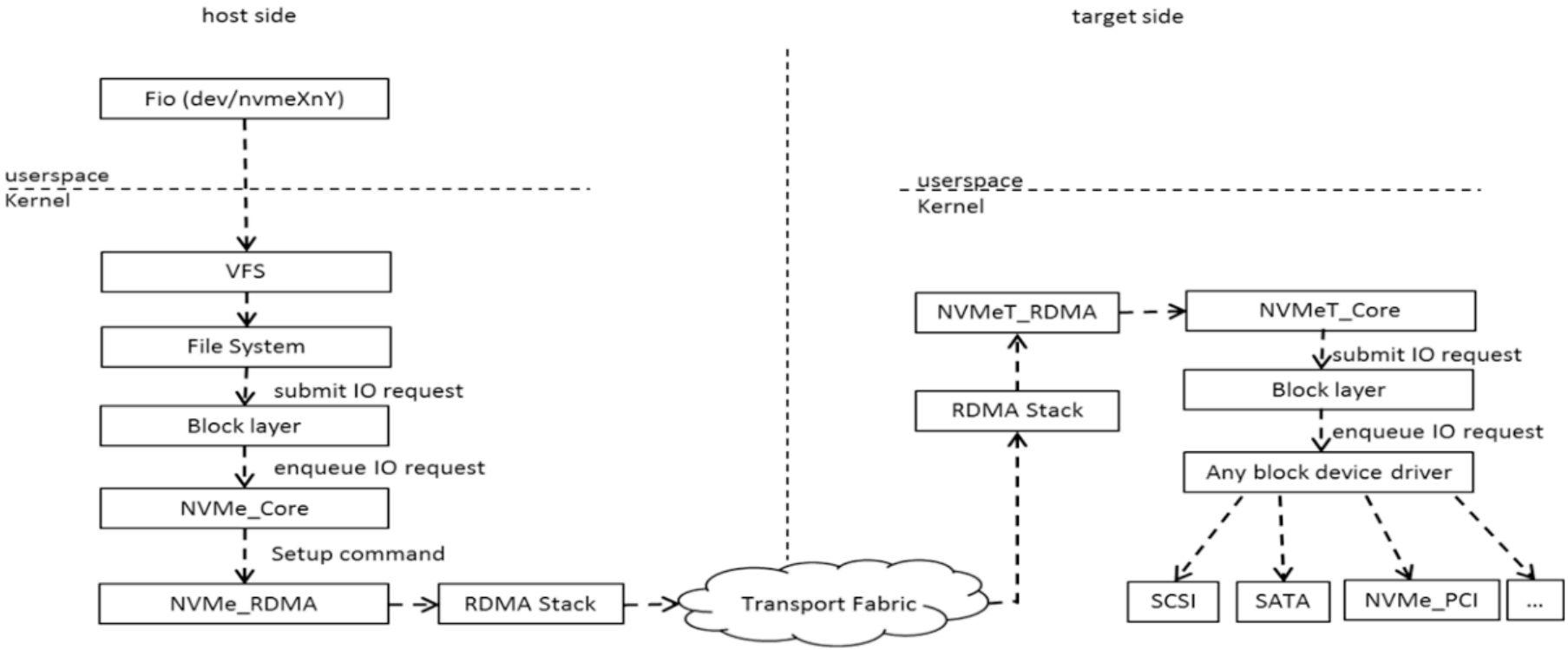
–Write: “pushes” data into remote virtual memory

–Read: “pulls” data out of remote virtual memory



[1]Introduction to RDMA Programming, Robert D. Russell, IOL ’12
[2] M. Su, M. Zhang, K. Chen, Z. Guo, and Y. Wu, “RFP: When RPC is Faster than Server-Bypass with RDMA,” EuroSys, pp. 1–15, 2017.

NVMe over Fabrics (NVMe + RDMA)



(a) NVMe-over-Fabrics Stack

图来自： Z. Guz, H. H. Li, A. Shayesteh, and V. Balakrishnan, "NVMe-over-Fabrics Performance Characterization and the Path to Low-Overhead Flash Disaggregation," *Systor '17*, no. May, 2017.

NVMe over Fabrics (NVMe + RDMA)

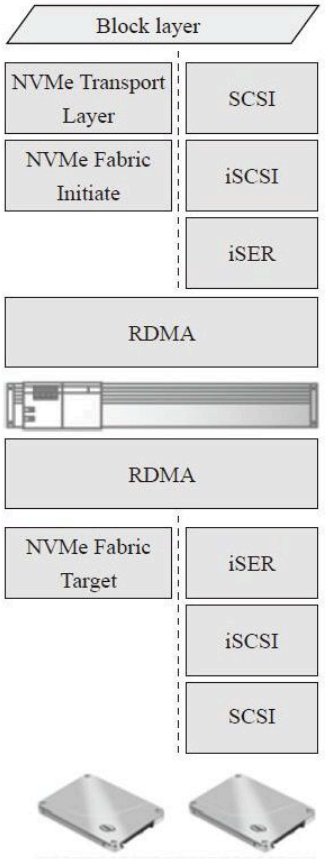


图6-61 NVMe over Fabrics（左）与传统方式协议栈（右）比较

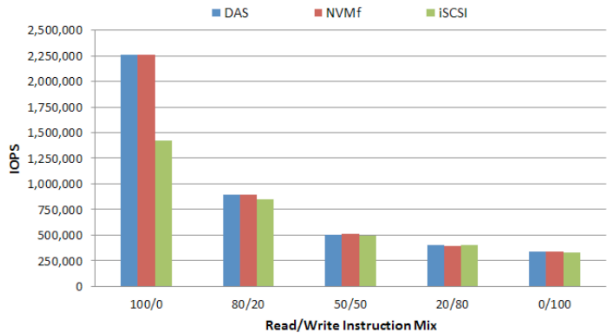


Figure 2: Maximum achievable throughput (in IOPS) for DAS, NVMf, and iSCSI.

IOPS性能

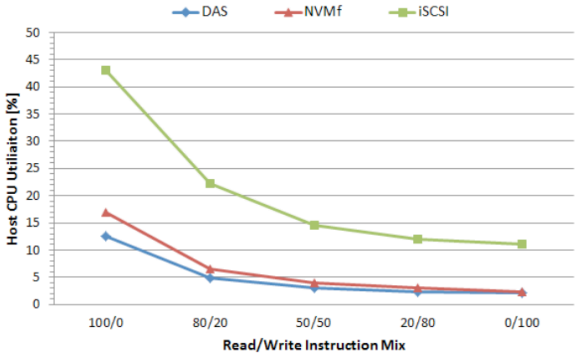


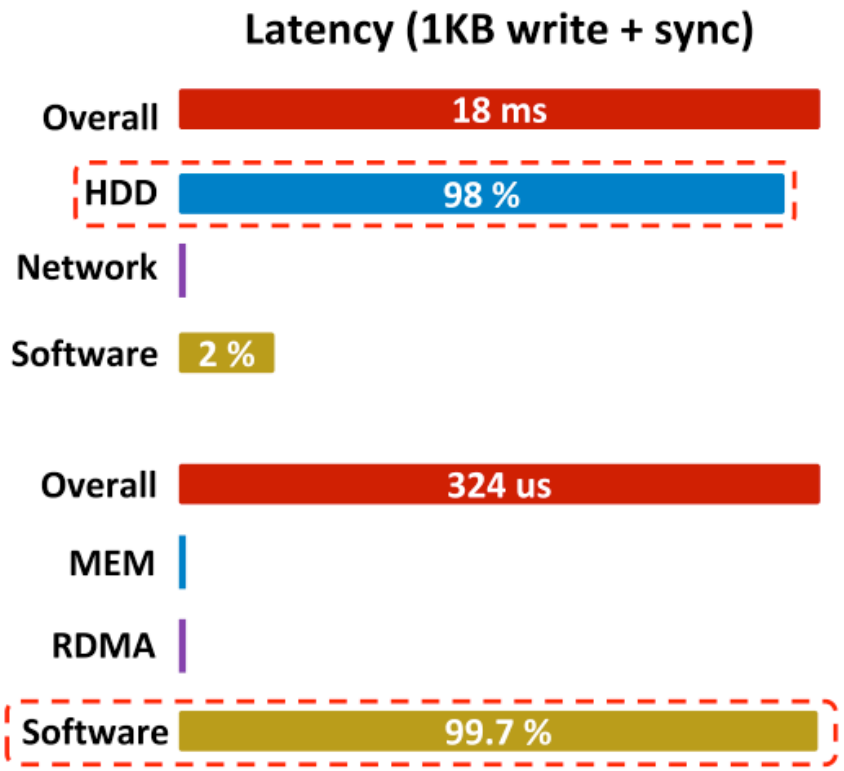
Figure 3: Host CPU utilization for DAS, NVMf, and iSCSI.

CPU占用

[1]《深入浅出SSD：固态存储核心技术、原理与实战》
[2] Z. Guz, H. H. Li, A. Shayesteh, and V. Balakrishnan, “NVMe-over-Fabrics Performance Characterization and the Path to Low-Overhead Flash Disaggregation,” Systor ’17, no. May, 2017.

分布式PM文件系统Octopus (PM + RDMA)

- DiskGluster
  - ✓ 存储：磁盘
  - ✓ 通信：GbE
- MemGluster
  - ✓ 存储：内存
  - ✓ 通信：RDMA



使用PM时，传统分布式文件系统和网络软件开销变得很大。
实现了一种基于RDMA网络并且的PM-aware的文件系统Octopus。

图摘自舒继武《大数据时代的存储系统若干变化的思考》

分布式PM文件系统Octopus (PM + RDMA)

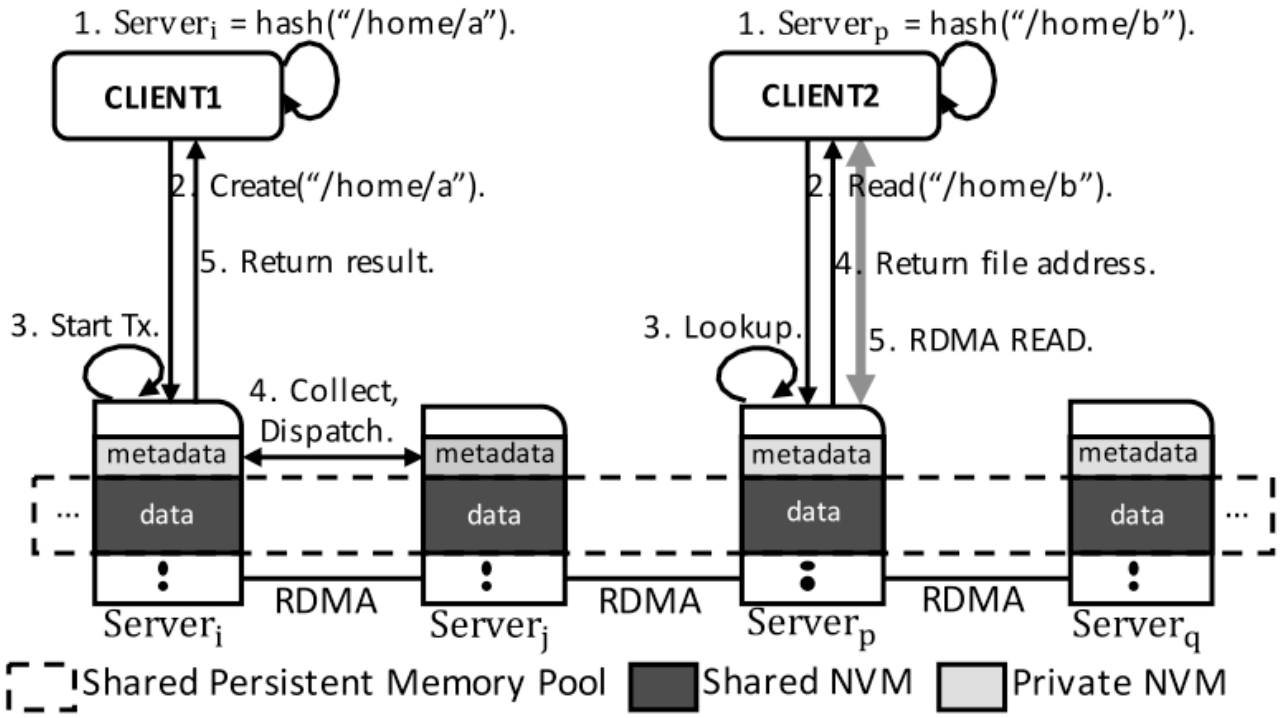


Figure 2: Octopus Architecture

[1] Y. Lu, J. Shu, Y. Chen, and T. Li, "Octopus: an RDMA-enabled Distributed Persistent Memory File System," 2017 {USENIX} Annu. Tech. Conf. ({USENIX} {ATC} 17), pp. 773–785, 2017.

分布式PM文件系统Octopus (PM + RDMA)

若接到client请求直接传输数据，太浪费server的CPU时间。

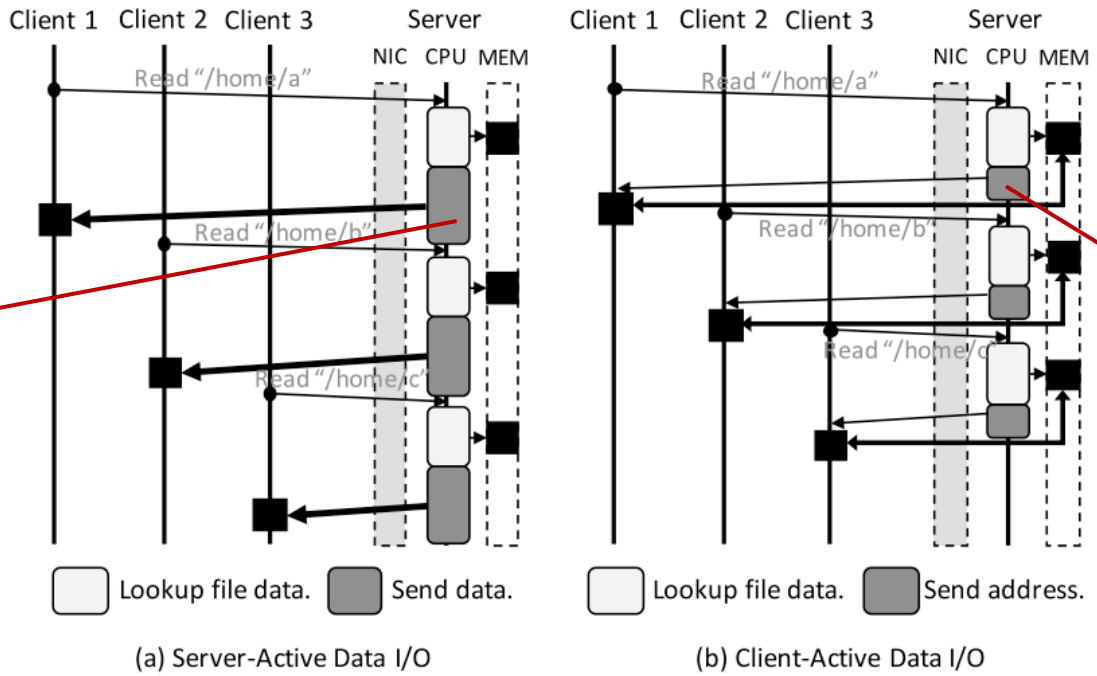


Figure 5: Comparison of Server-Active and Client-Active Modes

Server只返回地址，有client用RDMA read操作pull数据。

为了节省server端CPU资源来服务更多clients，client以read操作从server端pull数据(右图)。

[1] Y. Lu, J. Shu, Y. Chen, and T. Li, "Octopus: an RDMA-enabled Distributed Persistent Memory File System," 2017 {USENIX} Annu. Tech. Conf. ({USENIX} {ATC} 17), pp. 773–785, 2017.

分布式PM文件系统Octopus (PM + RDMA)

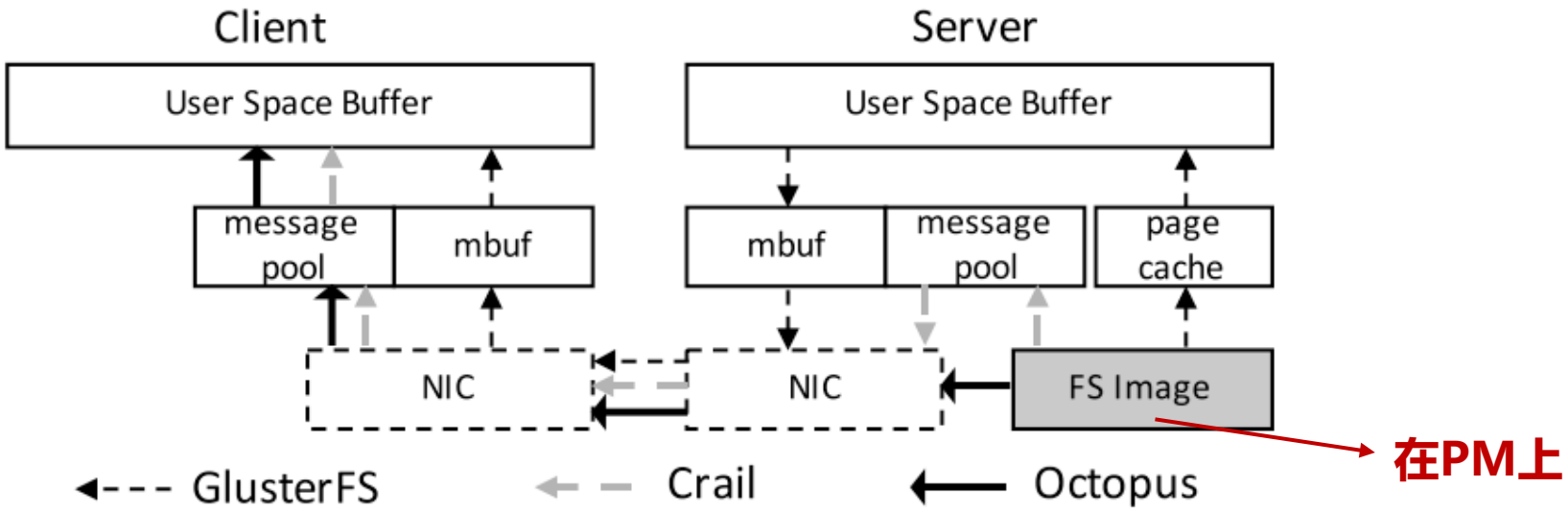


Figure 4: Data Copies in a Remote I/O Request

对于一次文件IO请求：

- 传统的GlusterFS拷贝了7次；
- Crail (PM-aware + 传统网络栈) 拷贝了6次;
- Octopus (PM-aware + RDMA传输) 拷贝4次。

[1] Y. Lu, J. Shu, Y. Chen, and T. Li, “Octopus: an RDMA-enabled Distributed Persistent Memory File System,” 2017 {USENIX} Annu. Tech. Conf. ({USENIX} {ATC} 17), pp. 773–785, 2017.

分布式PM文件系统Octopus (PM + RDMA)

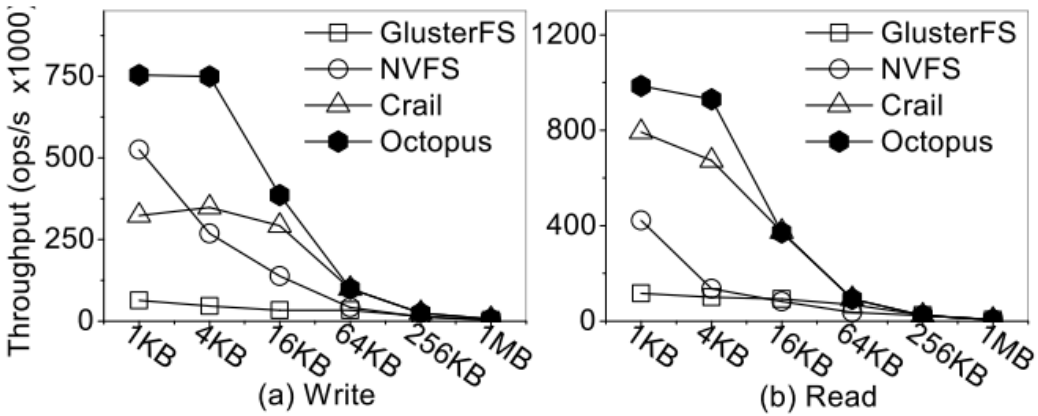


Figure 9: Data I/O Throughput (Multiple Clients)

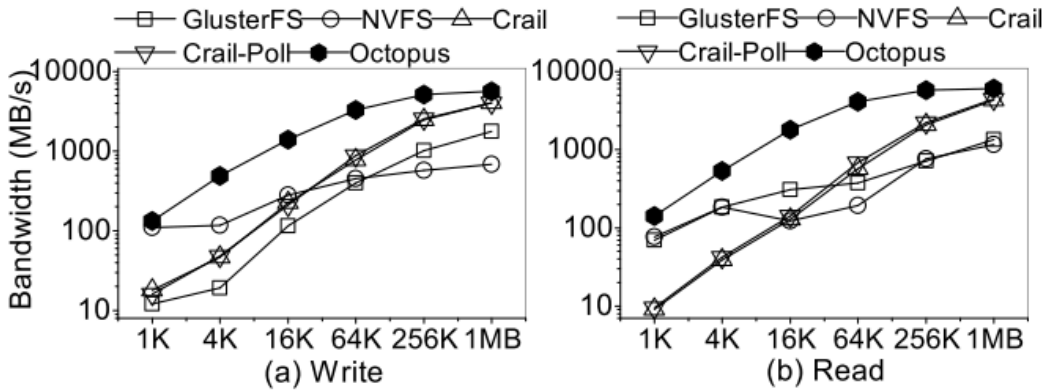


Figure 10: Data I/O Bandwidth (Single Client)

[1] Y. Lu, J. Shu, Y. Chen, and T. Li, “Octopus: an RDMA-enabled Distributed Persistent Memory File System,” 2017 {USENIX} Annu. Tech. Conf. ({USENIX} {ATC} 17), pp. 773–785, 2017.

总结

- 新型SSD协议NVMe，高性能SSD带来的挑战以及相关优化工作。
- 非易失内存 (NVM) 的基本工作原理，Persistent Memory的编程接口，存储系统设计问题。
- RDMA，以及NVMe、Persistent Memory与RDMA的结合。

Questions?

谢谢！