
CSE 3421

Introduction to Computer Architecture

Chapter 6:

Solid State Device Storage

Xiaodong Zhang

Time units and data size units (Review)

□ Time units in the computing world

- 1 millisecond (ms) = 10^{-3} second
- 1 microsecond (us) = 10^{-6} second
- 1 nanosecond (ns) = 10^{-9} second
- 1 picosecond (ps) = 10^{-12} second
- 1 femtosecond (fs) = 10^{-15} second
- 1 attosecond (as) = 10^{-18} second

□ Data size units in the computing world

- 1 kilobyte (KB) = 10^3 Bytes (2 sectors in HDD, 512 Bytes * 2)
- 1 megabytes (MB) = 10^6 Bytes (4 MB capacity in first HDD in 1956)
- 1 gigabytes (GB) = 10^9 Bytes (first GB disk was in 1997)
- 1 terabytes (TB) = 10^{12} Bytes (first TB disk was in 2007)
- 1 petabytes (PB) = 10^{15} Bytes (first PB disk: “Glass” in 5D, in 2020)
- 1 exabytes (EB) = 10^{18} Bytes (a 64-bit address maps to 18+ EB)
- 1 zettabyte (ZB) = 10^{21} Bytes (Human will produce 175 ZB data, 2025)
- 1 yottabyte (YB) = 10^{24} Bytes (many YB, including DNA data, in 2040)

A Scientific Discovery started a Revolution in Disks

- **Giant Magneto-resistance (GMR)** was discovered in 1988
 - By **Peter Gruenberg** (Germany) and **Albert Fert** (France)
 - **Giant resistance changes in materials made of alternating and very thin (nanometer thin) layers when exposed to magnetic fields.**
 - This discovery lays a foundation to increase the HDD density
 - First GMR based commercial HDD of 16 GB by IBM appeared in 1997.
 - Starting 2007, 1,000 +GB (TeraBytes) HDDs are available in the market
 - Next generation fast and high-density memory: Magneto-resistive RAM
 - Gruenberg and Fert received the **2007 Physics Nobel Prize** for GMR

5 Minute Rule (an economic model)

- **First version:** Jim Gray and Franco Putzolu (1987, SIGMOD)
 - **Background:** disk capacity is low and expensive, latency is not an issue
 - Accessing 1 KB data in disk costs \$2,000, but only \$5 in main memory
 - Rule: **pages referenced every 5 minutes should be memory resident**
- This experienced formula considers
 - Ratio between Disk Price and DRAM Price per MB
 - Disk Price(\$) /DRAM Price per MB (\$)
 - The higher the ratio, the longer the delay to the disk
 - The basic delay unit is the disk access time for a Mbytes
 - **Access interval** = Disk Price * Disk Time / DRAM price = 5 min (4.44)
 - Disk price = \$20,000, Disk time for a MB = 66.7 second, 1 MB = \$5,000
 - Note, the disk bandwidth (1987) is 15 KB /s, thus 66.67s for a MB

Evolution of the 5-Minute Rule

- **First version:** Jim Gray and Franco Putzolu (1987, SIGMOD)
 - **Background:** disk capacity is low and expensive, latency is not an issue
 - Accessing 1 KB data in disk costs \$2,000, but only \$5 in main memory
 - Rule: **pages referenced every 5 minutes should be memory resident**
- **Second version:** Jim Gray and P. Shenoy (2000, ICDE)
 - **Background:** capacity is up 1,000x, bandwidth only 40X, very low price
 - 5 minute rule becomes **a caching rule for performance** due to:
 - (1) Disk accesses slow 10X per decade; (2) disk scanning time increases
- **A recent version:** G. Graefe (CACM, 2009)
 - **Background:** SSD is still expensive, disk space is almost free, low speed
 - For small size blocks, 5-minute rule holds between DRAM/SSD
 - For a very large size blocks, 5-minute rule holds between SSD/disks

What happens today for 5 Minute Rule?

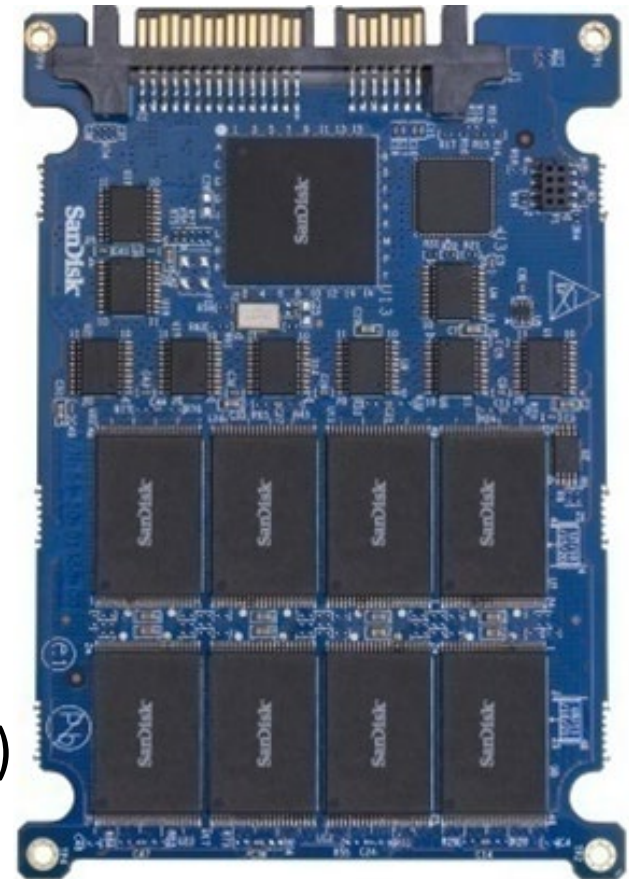
- 1 MB = \$0.016, Disk time for a MB = 10 seconds, disk price = \$200
 - Note: disk bandwidth is 100 KB /s, thus 10 seconds for a MB
- The time ratio and price ratio should be equivalent
 - Ratio between Disk Price and DRAM Price per MB
 - Disk Price(\$) /DRAM Price per MB (\$)
 - The higher the ratio, the longer the delay to the disk
 - The basic delay unit is the disk access time for a Mbytes
 - Access interval = Disk Price * Disk Time / DRAM price = 35 hours (34.72)

HDD Improvement has been focused on Density

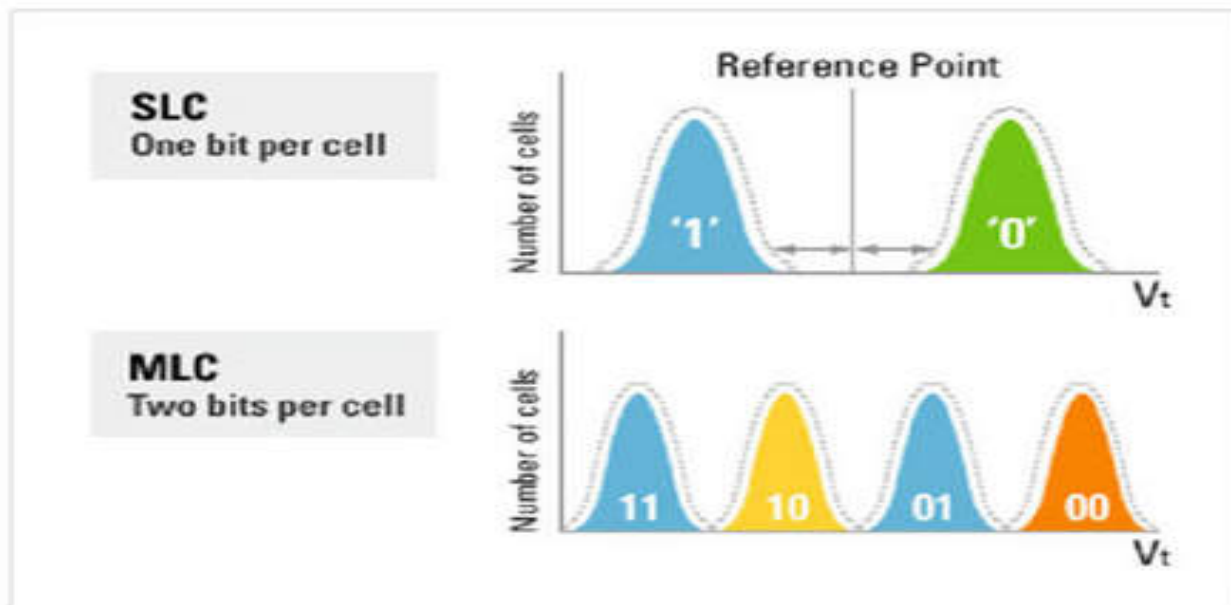
- Huge capacity disks with low price and small size still have
 - **Low speed and high energy consumption** (current stage)
 - High capacity causes high access latency (for more than 10 years)
- Specific issues and concerns
 - Capacity/bandwidth increases significantly , so does latency
 - Space is almost free, but to access data is increasingly more expensive
 - Economic model: a disk should be infrequently accessed for archival
 - DRAM buffer can address the performance issues, but not the power
- **A fast and low power storage is highly desirable.**

Flash Memory based Solid State Drive

- Solid State Drive (SSD)
 - A **semiconductor** device
 - Mechanical components free
- Technical merits
 - Low latency (e.g. 75 μ s)
 - **High bandwidth** (e.g. 250MB/sec)
 - **Low power:** 0.06 (idle)~2.4w (active)
 - Shock resistance
 - Lifespan: 100GB/day \rightarrow 5 years (x25-M)



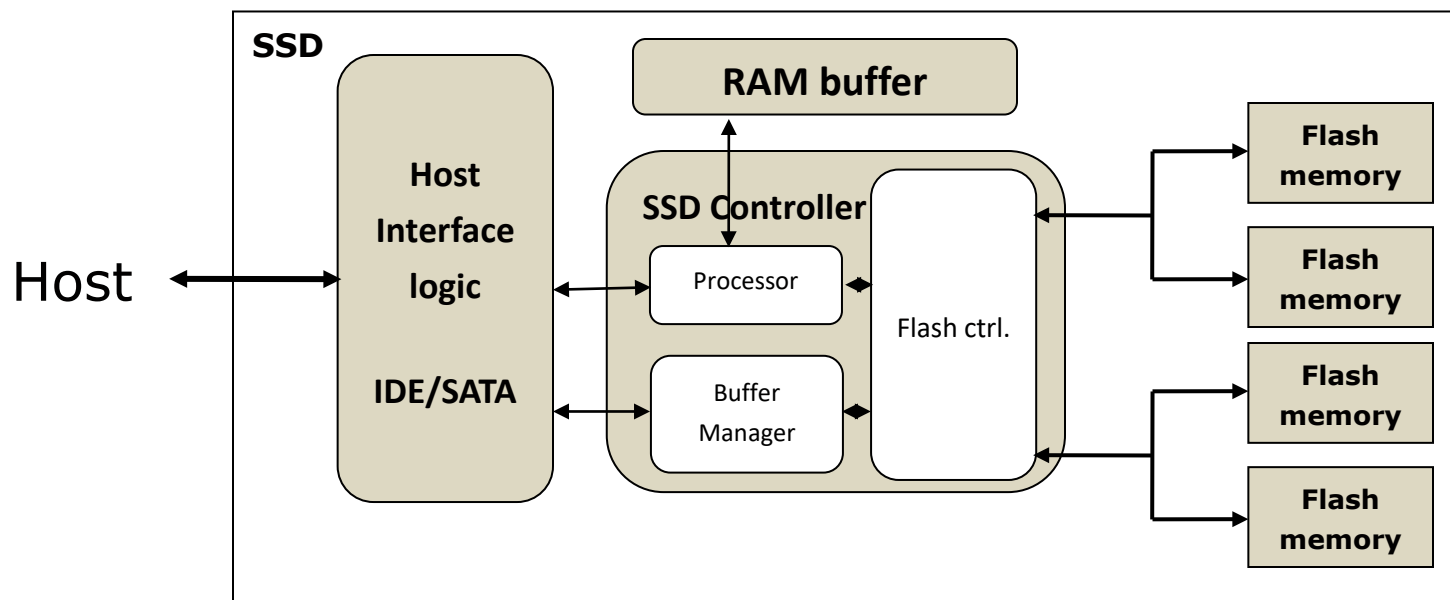
Single Level Cell (SLC) vs. Multi-level Cell (MLC)



- **SLC:** two voltage states for 0 and 1 (1 bit per cell)
 - Low density, high access speed, high endurance, low power, high cost
- **MLC:** four voltage states for 00, 01, 10, 11 (2 bits per cell)
 - High density, lowered access speed, lowered endurance (10x), increased power, **low cost**
- **TLC:** triple level cell, 3 bits in each cell => an enhanced MLC

Flash Memory based Solid State Drive

- Architecture of solid state drives (SSD)
 - Host interface logic – SATA, IDE, SCSI, etc.
 - **SSD Controller** – processor, buffer manager, flash controller
 - Integrated/Dedicate RAM buffer
 - An array of **flash memory** packages



Adapted from USENIX'08 (Agrawal et al.)

Performance- and Power-Efficient



120x HDDs

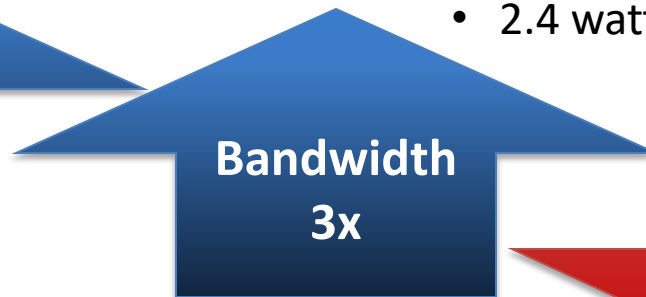
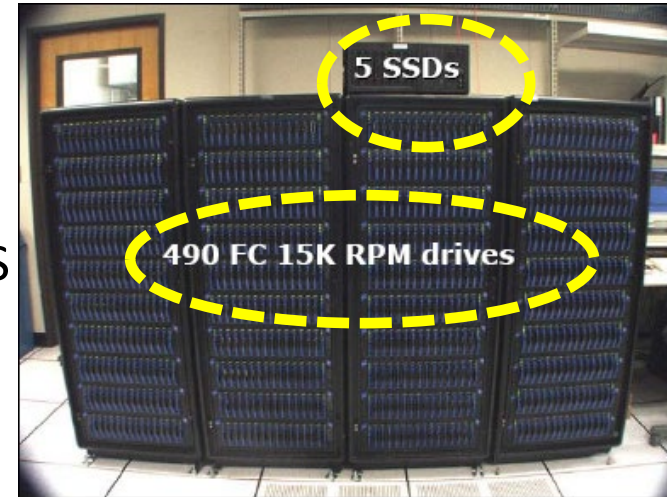
- 36,000 IOPS
- 12GB/sec
- 1,452 watts
- 8.8TB
- Per HDD

- 100MB/sec (R/W)
- 300 IOPS
- 12.1 watts (active)

• 120x SSDs

- 4,200,000 IOPS
- 36GB/sec
- 288 watts
- 3.8TB
- Per SSD

- 250/170MB/sec (R/W)
- 35,000 IOPS (Read)
- 2.4 watts (active)



Challenge 1: Affordability

- A full-SSD based storage solution

- Example: Gordon HPC Cluster*

- Data-centric Scientific App.
 - 64TB DRAM + 300TB Flash
 - \$20,000,000 funding from NSF
 - 3-4 years lifespan



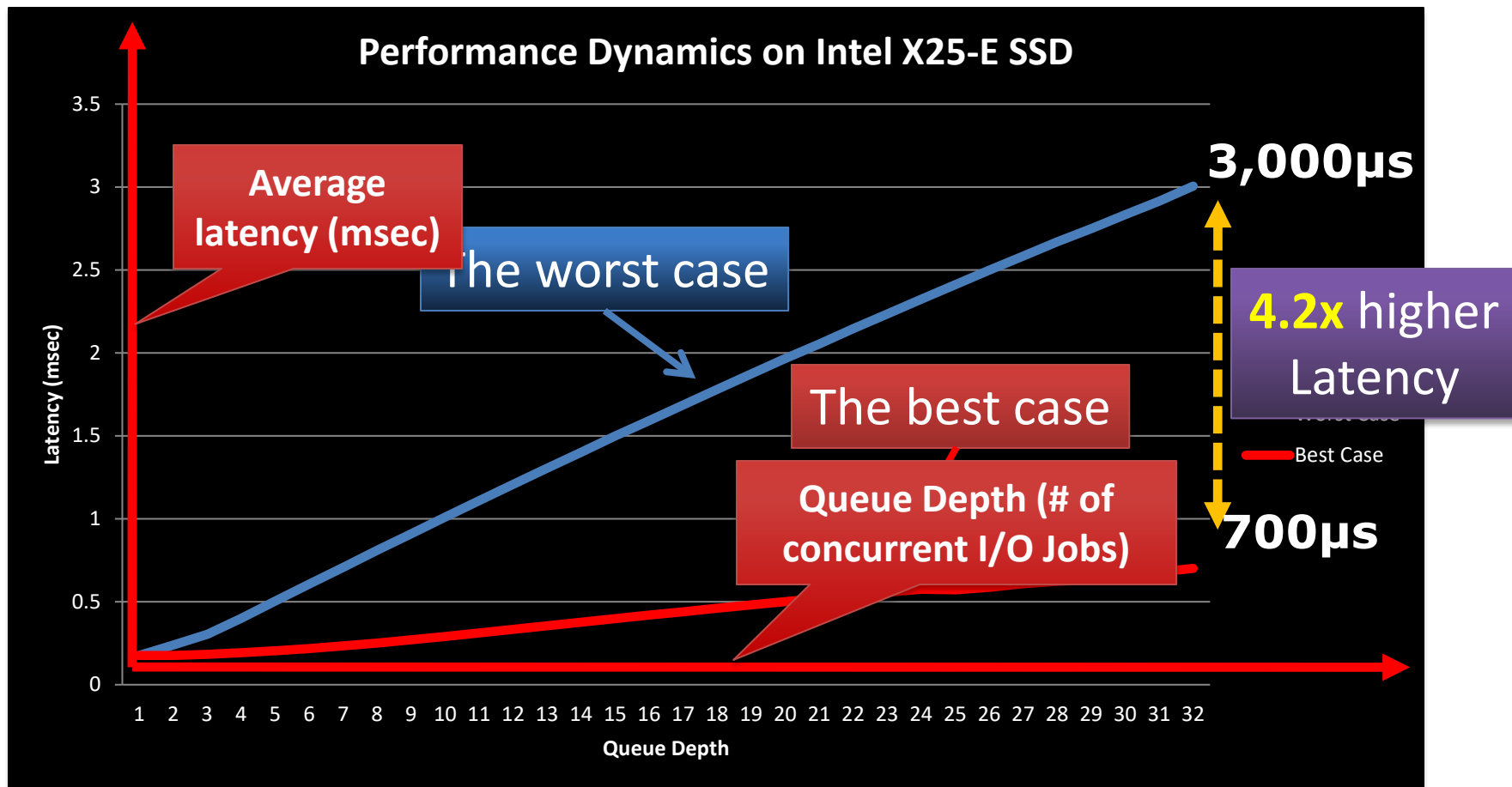
\$5,000,000/year

- In reality

- High performance comes from very high cost
 - Not affordable for most data centers

Challenge 2: Performance Dynamics

- Random read 4KB in the 1024MB space with 1~32 I/O jobs (different data allocations among flash chips result in different performance)

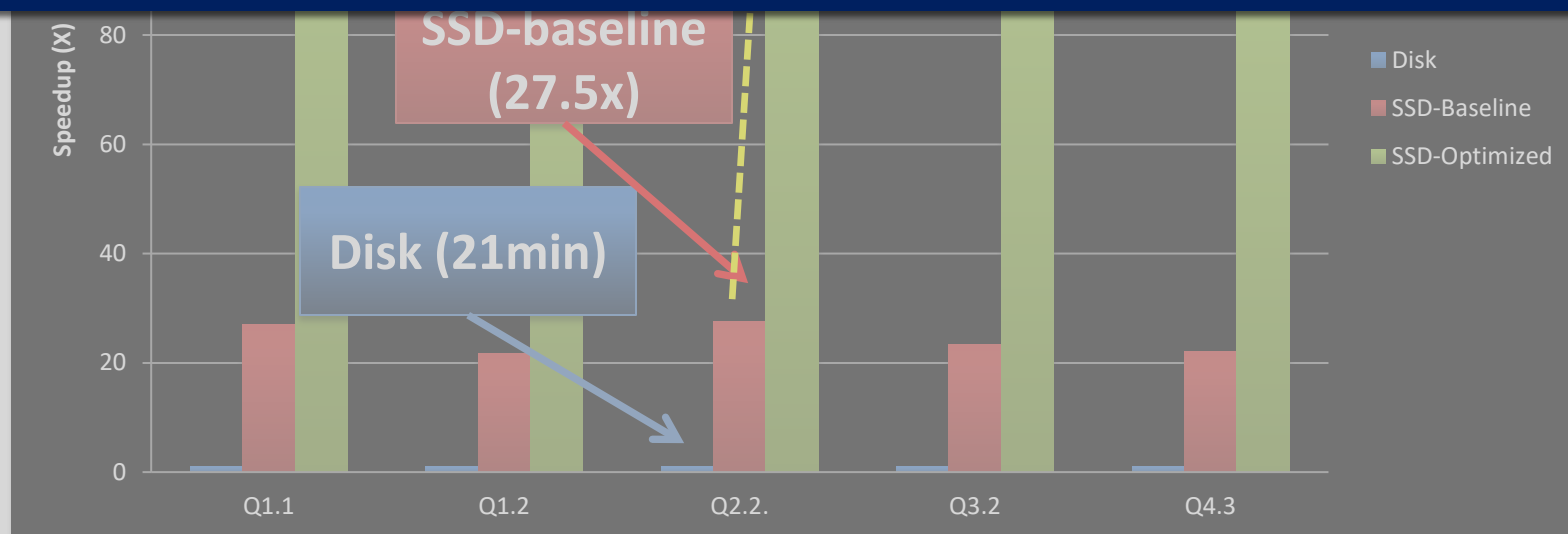


Challenge 3: Resource underutilization

- Database query executions

Star Schema Benchmark (SSB) Query 25-E SSD
SSD-opt. (127.2x)

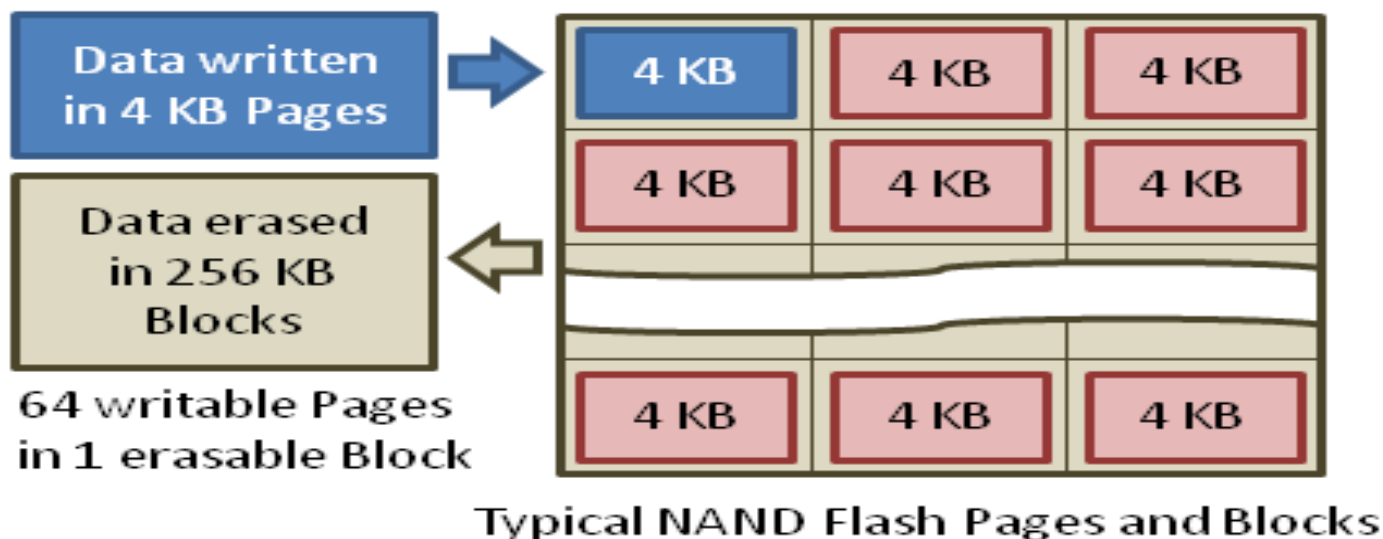
However, the high performance potential of SSD cannot be automatically tapped without **extensive efforts**.



Challenge 4: Reliability of SSD

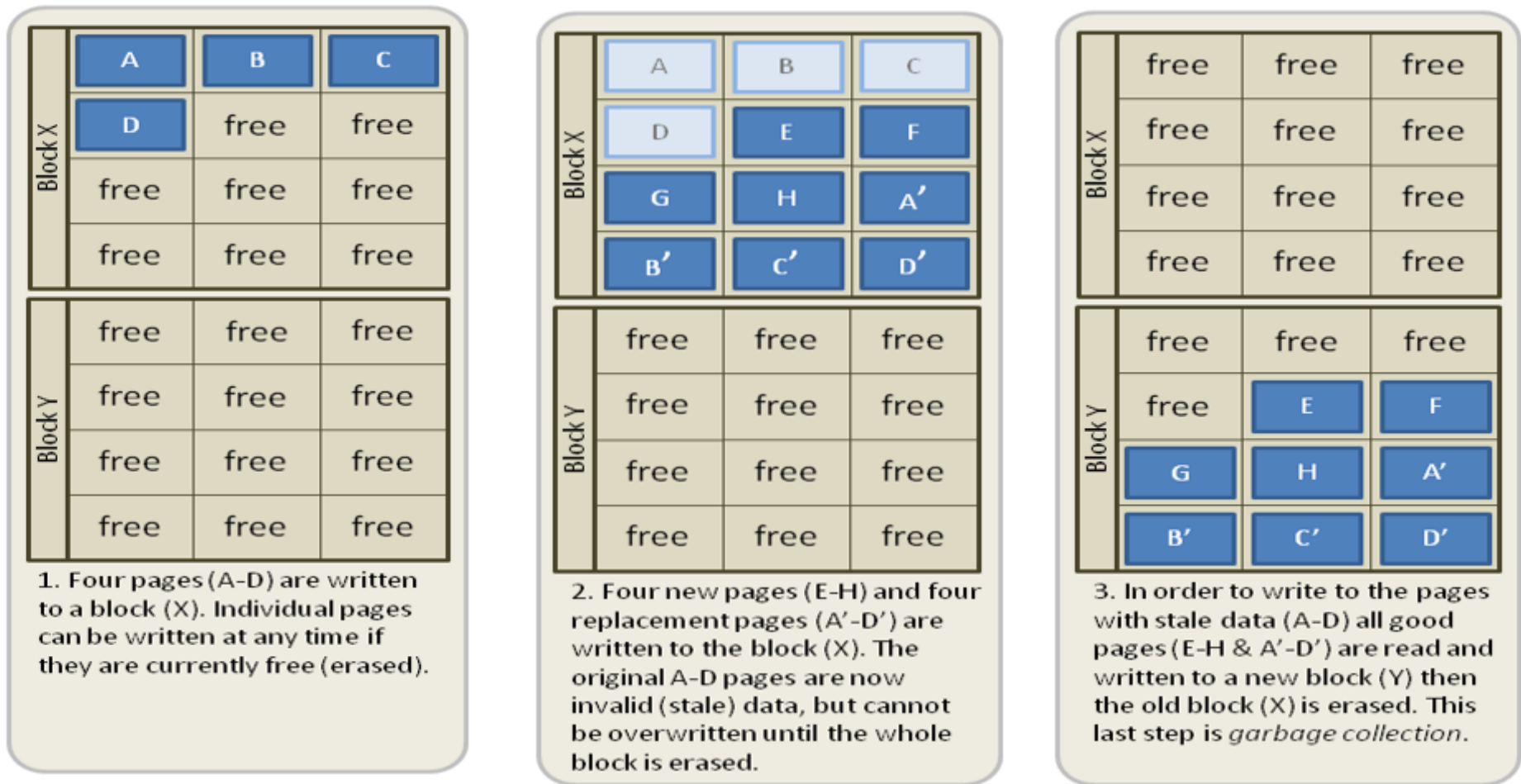
- **Program-erase cycles** (P/E cycles):
 - Most commercially available flash products are guaranteed to withstand around 100,000 P/E cycles.
 - In reality, the P/E cycles will decrease for low end SSD products so that the price can be affordable.
 - Wear leveraging: dynamically remapping blocks in order to spread write operations between sectors.
- **Read Disturb errors**
 - Cause other cells near being read to change over time if they are not rewritten. (noise is heard and disturbed by next door neighbor if the wall is thin)
- **Limited Capacity**
 - Trade-off between large capacity and reliability subject to a given price.
 - A large portion of the space is used for over-provisioning.

SSD Internal 1: Write and Erase



- **Data write:** data cannot be overwritten (erase before write)
 - Data unit to write is **page** (4K-8K), and can only write in an erased unit
- **Data Erase:** to prepare another write in SSD
 - Data unit to erase is a **block** (e.g. 64-128 pages)
 - Flash memory can be only **programmed** and **erased** a limited number of times (**P/E cycles**: e.g. SLC for 100,000, MLC for 10,000)

SSD Internal 2: Garbage Collection



- As some pages in block X are no longer needed (stale), GC does
 - (1) valid pages are moved to free block Y, (2) entire block X is erased
 - The process is expensive and can be done as a background process

Where are **data storages** in the memory hierarchy ?

- **Registers**
 - On chip and is closest to ALU, **32 - 64 bits** each
 - **Compiler** directly uses available registers during the code generation
 - Registers can store base memory address, but **no mapping** between R and M.
- **DRAM and on-chip caches**
 - Memory page size: **4KB - 8KB**
 - Cache block size: **4 Bytes – 32+ Bytes**
 - Virtual addresses are translated into physical memory addresses by **OS**
 - DRAM pages are managed by the **page table**
 - Both DRAM memory and caches **share the same memory addresses**
 - DM, set-associative and F-associative caches are memory address related
- **HDD and SSD (independent of memory address)**
 - HDD block size = 512 Bytes, SSD block size = 64-128 pages (256KB to 512 KB)
 - Memory \Leftrightarrow HDD, a page is broken into multiple 512 Byte Logical Blocks (LB)
 - Memory \Leftrightarrow SSD, a set of LBs map to pages sequentially in a SSD block

Where are **data storages** in the memory hierarchy ? ²⁰

(1) Registers

- **Registers**
 - Also called processor registers or CPU registers
 - On chip and is closest to ALU, **32 - 64 bits** each
 - **Compiler** directly uses available registers during the code generation
 - Assembly programmers directly use registers
 - Registers can store base memory address, but **no mapping** between Registers and Memory.
 - Registers are not under the management of OS
 - **Registers is in the virtual space**

Where are **data storages** in the memory hierarchy ? ²¹

(2) on-chip caches and off-chip DRAM

- Cache and memory are in **physical space of execution**
- Cache block size: **4 Bytes – 32+ Bytes**
- Memory page size: **4KB - 8KB**
- Virtual pages are allocated in physical memory pages by **OS**
- Virtual and physical pages are indexed in **page table**
- Both memory and cache **share the same memory address**
- For a given memory address, it points to
 - a **Byte** in a **page** and in a segment of multiple pages; which **memory bank** the page in
 - a block of direct-mapped, set-associative, fully-associative cache
 - If cache is partitioned by pages, which cache region it is

Where are **data storages** in the memory hierarchy ? ²²

(3) Storage – Hard Disk Drive

- HDD's location (sector/track) is **independent** of memory address
- **HDD block size** = 512 Bytes (a sector on a track)
- Memory ⇔ HDD, a virtual or physical page is broken into multiple 512 Byte **Logical Blocks (LB)** for disk to find a sequence sectors to store
- A **directory** in **disk controller** handles the mapping between file-name + LB and PB (physical sectors in the disks)

Where are **data storages** in the memory hierarchy ? ²³

(4) Storage – Solid State Drive

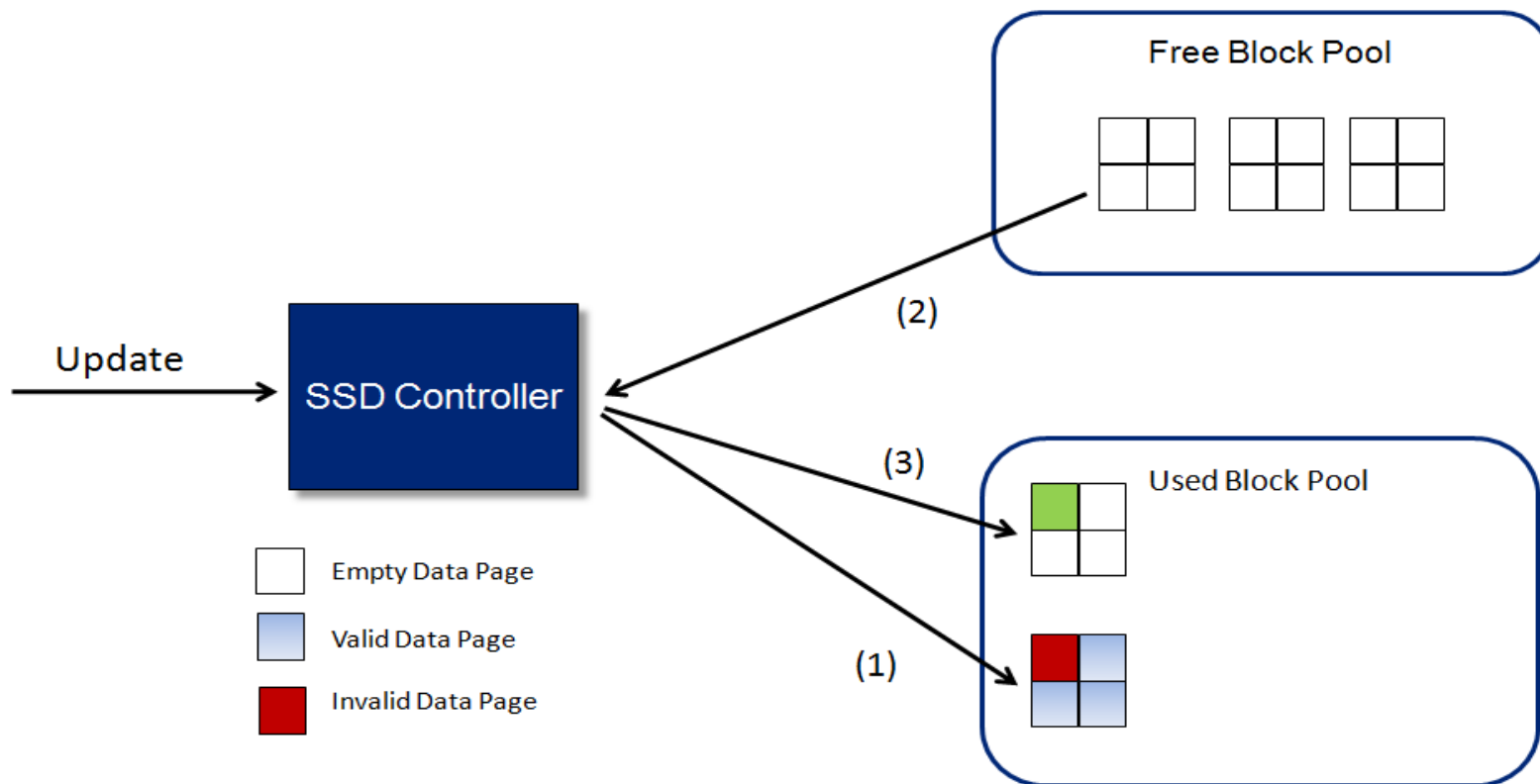
- SSD's location (page/block/domain) is **independent** of memory address
- **SSD block size** = 64-128 pages (1 page = 2 KB)
- Memory \Leftrightarrow SSD, a virtual or physical page is written in a free page in a block in SSD
- A **mapping table** in **SSD controller** for File System to handle the mapping between file-name + pages by # and where they are located in SSD (page/block/domain)

SSD Internal 3: Write Amplification

- **Measurement**
 - $(\text{data written to the flash memory}) / (\text{data written by the host})$
- **Worst case**
 - To write a 4K page to a 512K block, where only one page is invalid
 - Garbage collection writes 127 valid pages (508K data) to a free block, and then erase the entire block before writing the 4K page to the new block
 - $\text{Write Amplification} = 512\text{K} / 4\text{K} = 128$

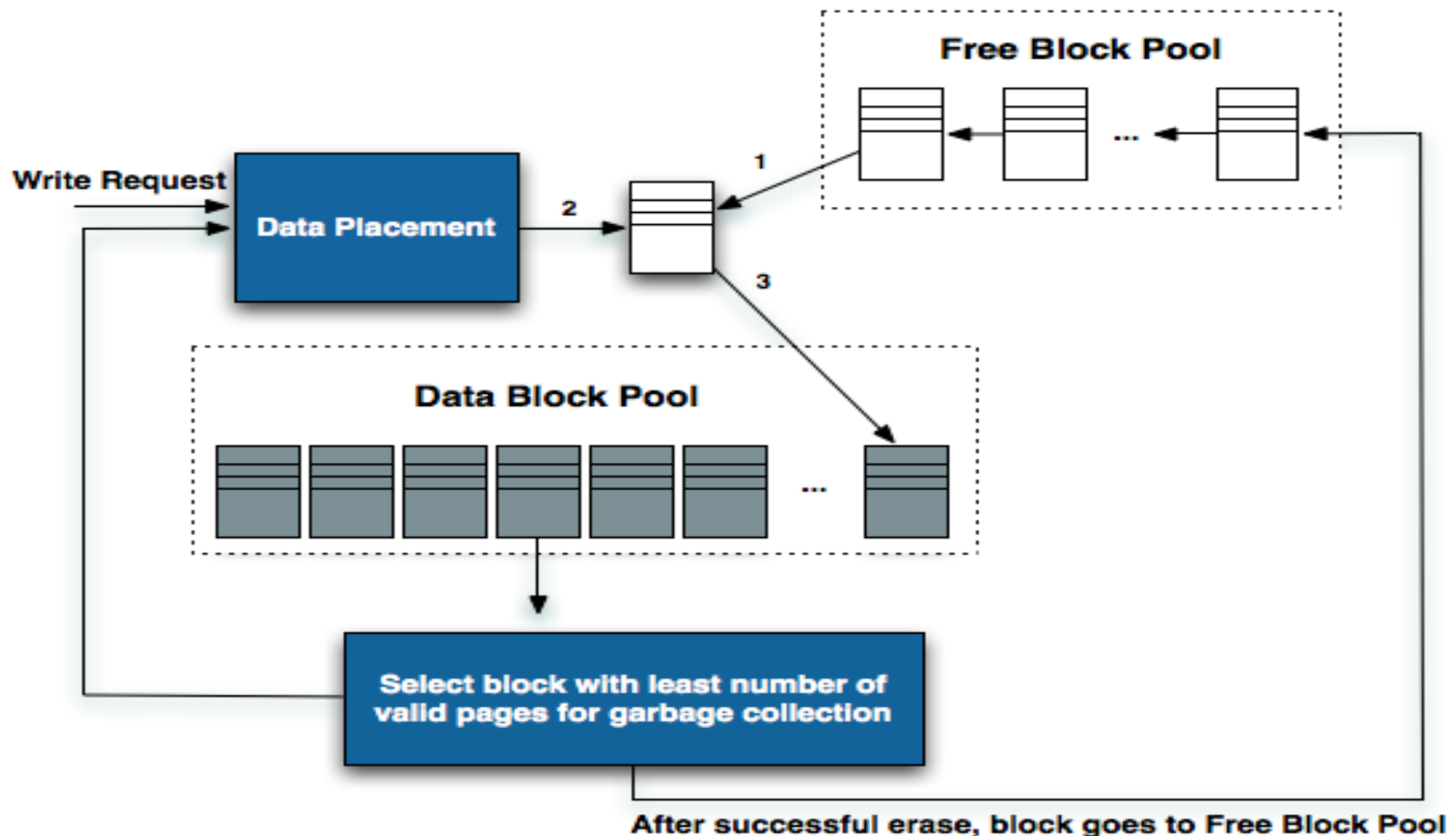
SSD Internal 4: Wear Leveling

- Balance the demand and supply between used and free block pools
 - (1) upon an update request, mark the page invalid, (2) fetch an empty block from free pool, (3) update the page and return the block to the used pool

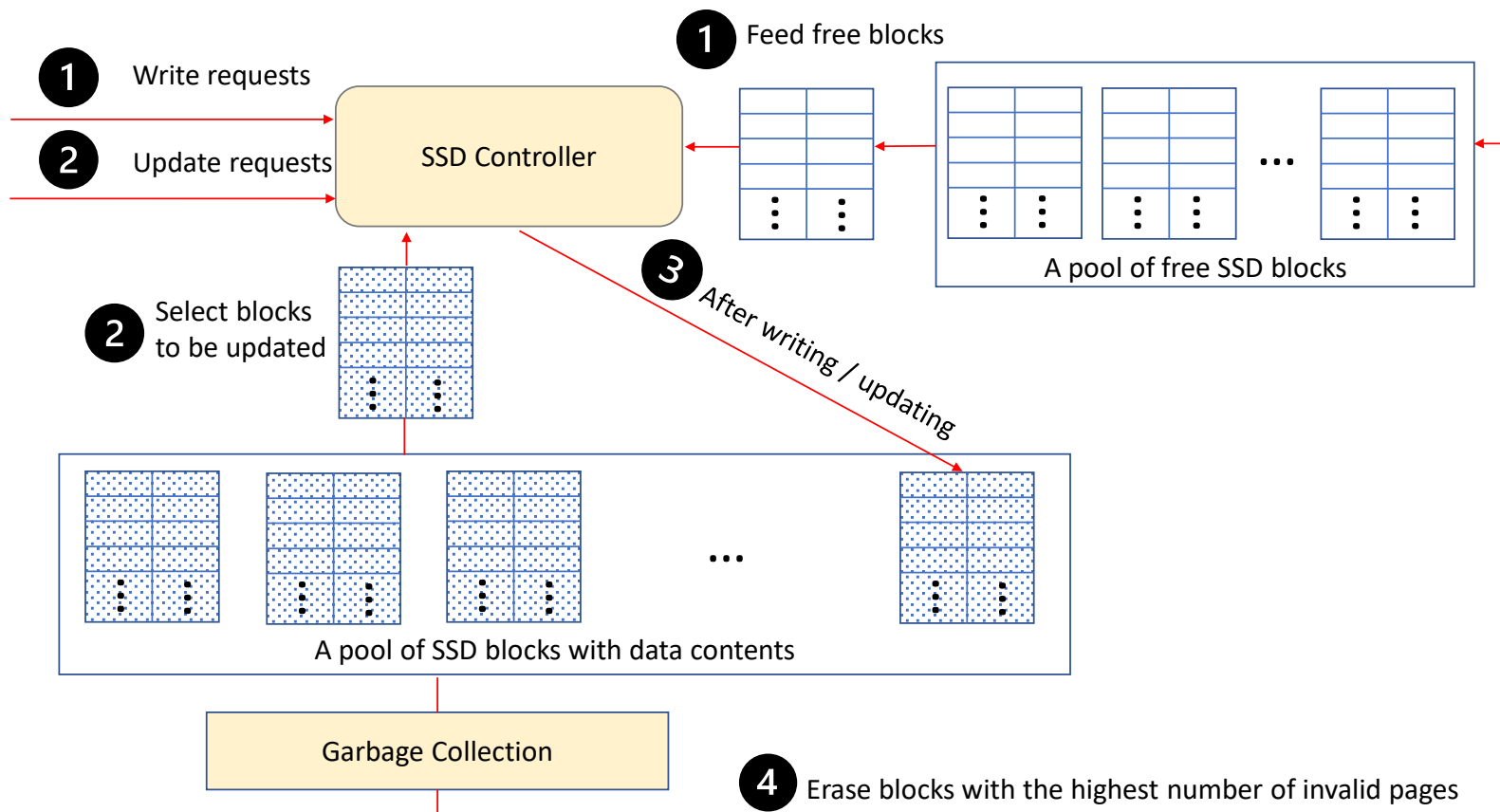


A simple view of SSD writes

- Achieving optimization and effectiveness
 - (1) over-provisioning provides a sufficient large free pool, (2) iteratively write blocks in free pool achieve wear leveling, (3) minimizing GC operations



Put them all together in a loop

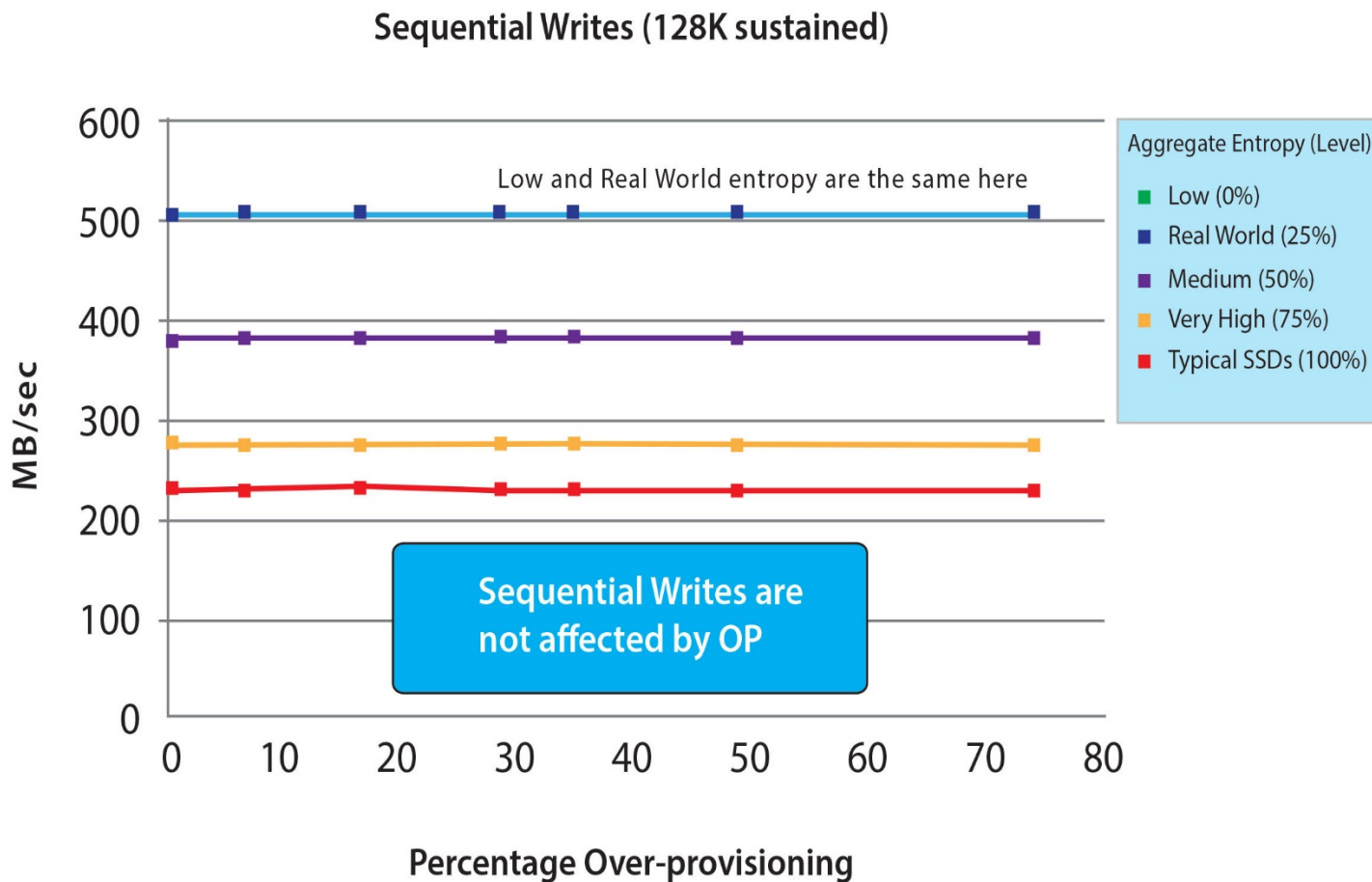


- (1) A write request is responded by a free block
- (2) An update request is responded by the selected block for updating
- (3) A newly written/updated block is put in SSD block pool
- (4) GC collect blocks with highest number of invalid pages to supply new free blocks

SSD Internal 5: Over-provision in the Loop

- **Free Block Pool**
 - Initially all the blocks are in the free pool.
- **Data Block Pool**
 - is the user capacity
 - If the Data Block Pool = Free Block Pool, no over-provision
 - The difference is the over-provision amount
- **Over-provision** will reduce the number of garbage collections
 - Reduce the write amplifications

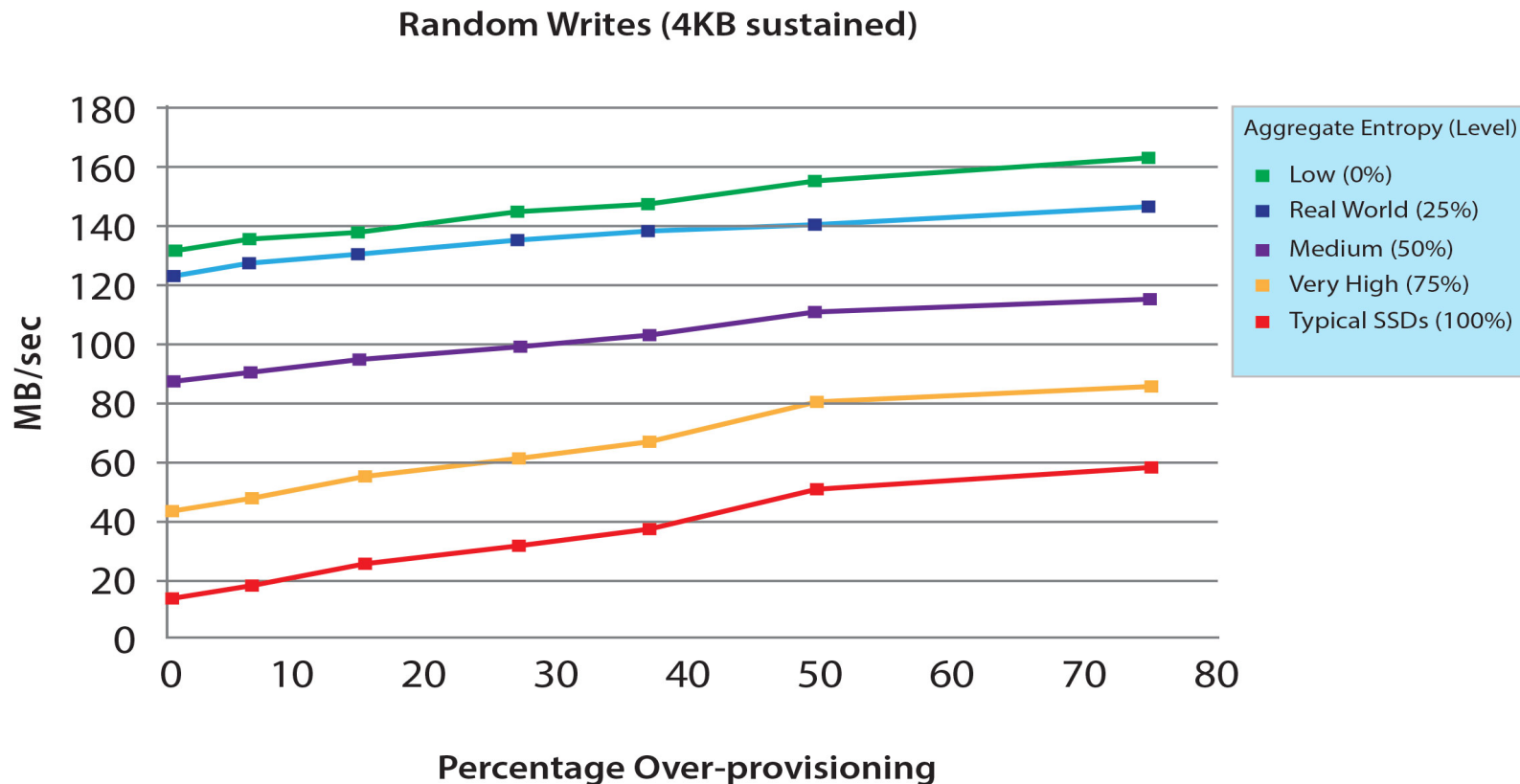
An Ideal Case for SSD: Garbage Collection Free



- **Entropy** reflects the uncertainty level of data
- No garbage collections due to **no delete/modification**

Courtesy of Seagate

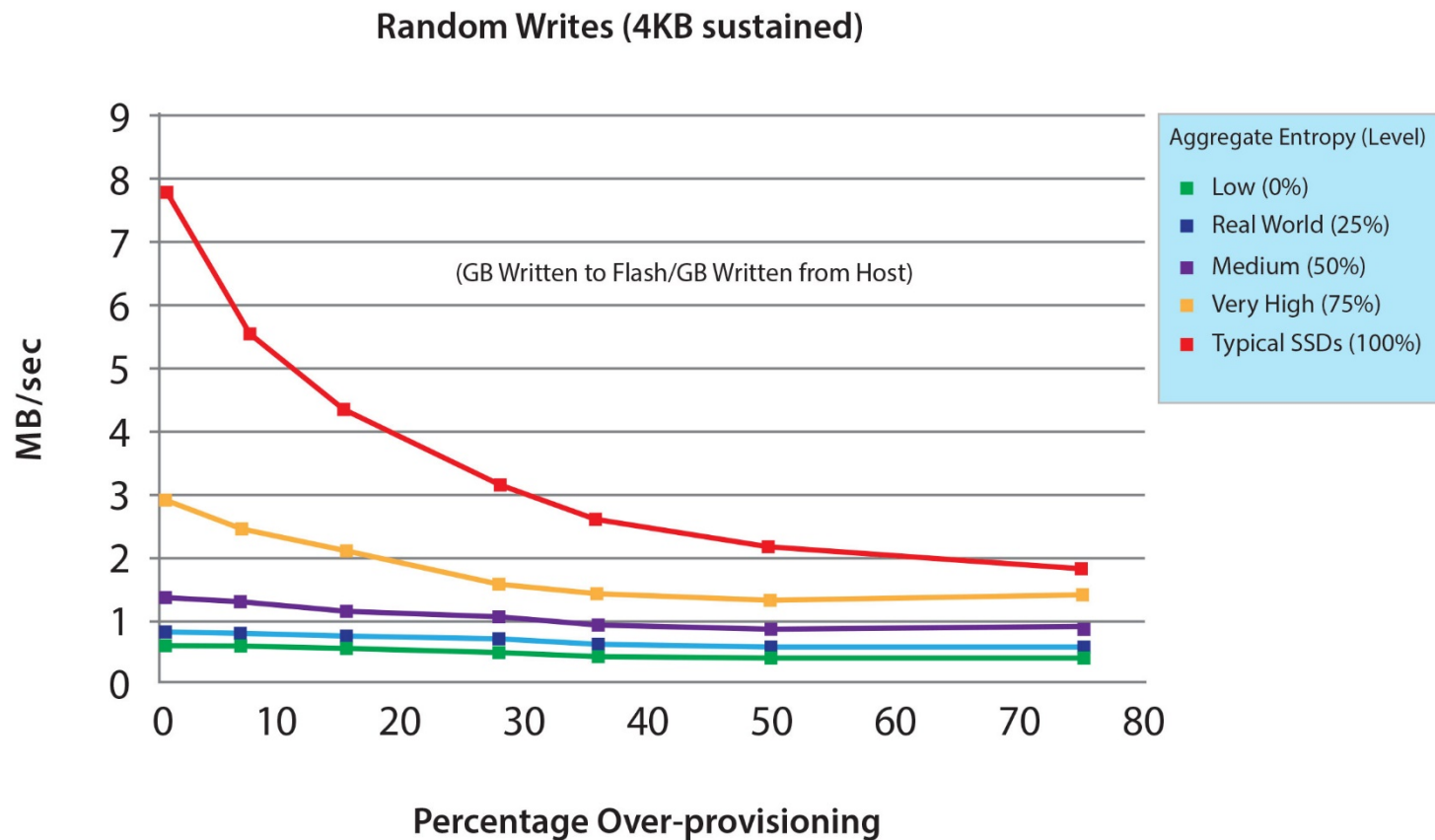
Increasing Write Bandwidth by Over-Provisioning



- **Random writes** moves/invalidates data pages in SSD
- **Garbage collection** is frequent
- **Entropy** reflects uncertainty level of data

Courtesy of Seagate

Reducing Write Amplifications by Over-Provisioning



- **Write Amplification** is reduced by certain % of over-provisioning
- **Entropy** reflects uncertainty level of data

Courtesy of Seagate

SSD “block-device”

- **Block size**
 - Based on logical block size:
 - VM: 4-8 KB, Database: 8 KB, and buffer cache: 1 KB
- **Don't confused with SSD block**
 - a flash memory block contains 64-128 flash pages
 - A flash memory page is the basic SSD unit (2-4 KB each)
 - Flash pages in a block are ordered by numbers
 - LBNs are written/read sequentially inside an SSD block
- **HDD vs SSD**
 - a logic block is stored in multiple sectors in HDD
 - In half, one or multiple flash pages in SSD

Critical Issues

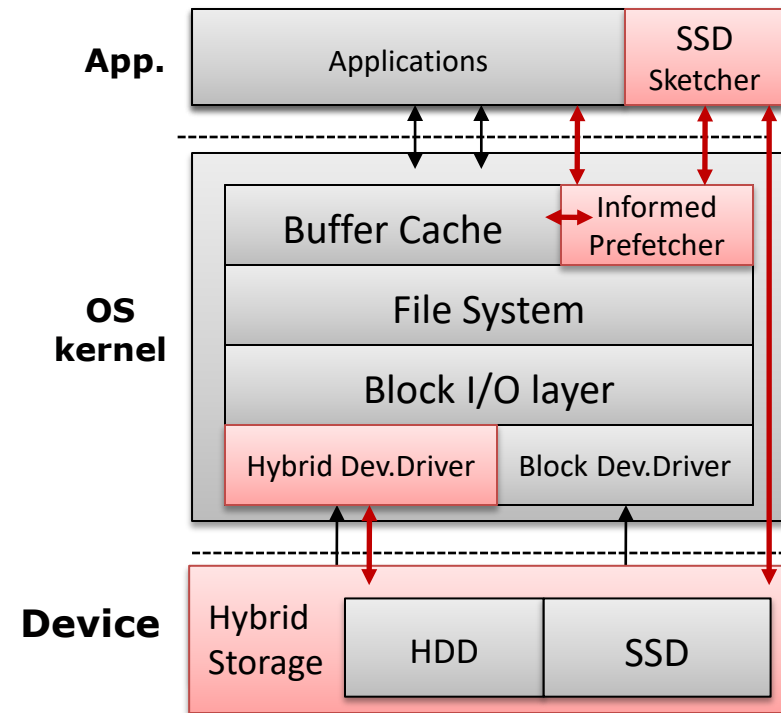
- **Performance dynamics** due to the unknown internals
 - A systematic effort is needed to timely and accurately detect the internal structures of the SSDs
- **Affordability** and **limited capacity** of SSDs
 - A hybrid storage is a best cost- and performance-effective solution
- **Underutilized** rich and hidden storage resources
 - System and application efforts to fully utilize the rich idle/hidden resources, such as internal parallelism
- **Reliability issues** caused by wear-out problem of flash
 - SSD is an ideal home for random reads, but not for frequent writes

Critical Issues

- **Performance dynamics** due to the unknown internals
 - A systematic effort is needed to timely and accurately detect the internal structures of the SSDs
- **Affordability** and **limited capacity** of SSDs
 - A hybrid storage is a best cost- and performance-effective solution
- **Underutilized** rich and hidden storage resources
 - An effective solution is desirable to utilize the rich idle/hidden resources, in particular internal parallelism
- **Reliability issues** caused by wear-out problem of flash
 - Technical advances are improving lifespan (e.g. 100GB/day → 5 years)

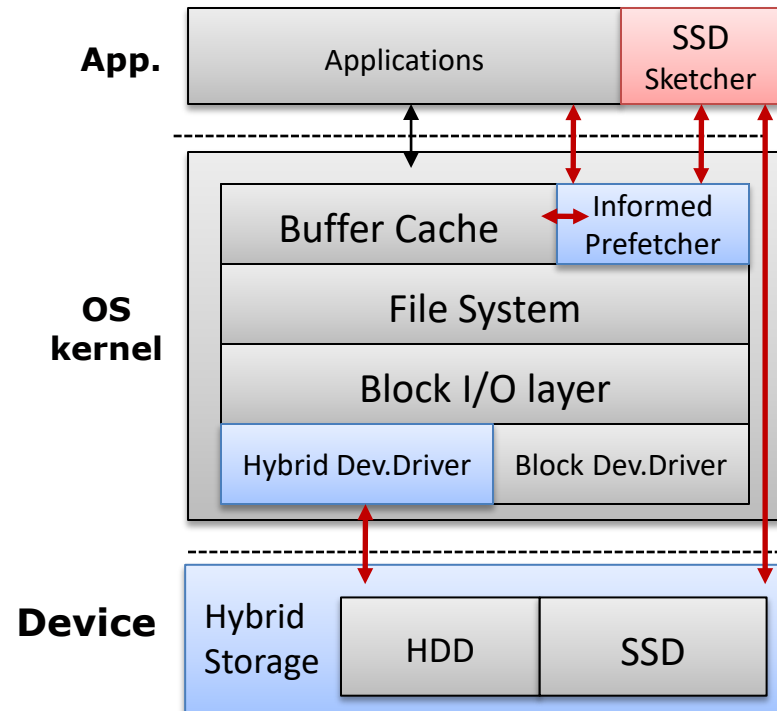
A Framework for a hybrid storage system

- **SSD Sketcher** – Detecting SSD internal structures
- **Hystor** – Providing hybrid storage services
- **Prefetcher** – utilizing the internal resources of SSD
- Other efforts by applications to fully utilize parallelisms.

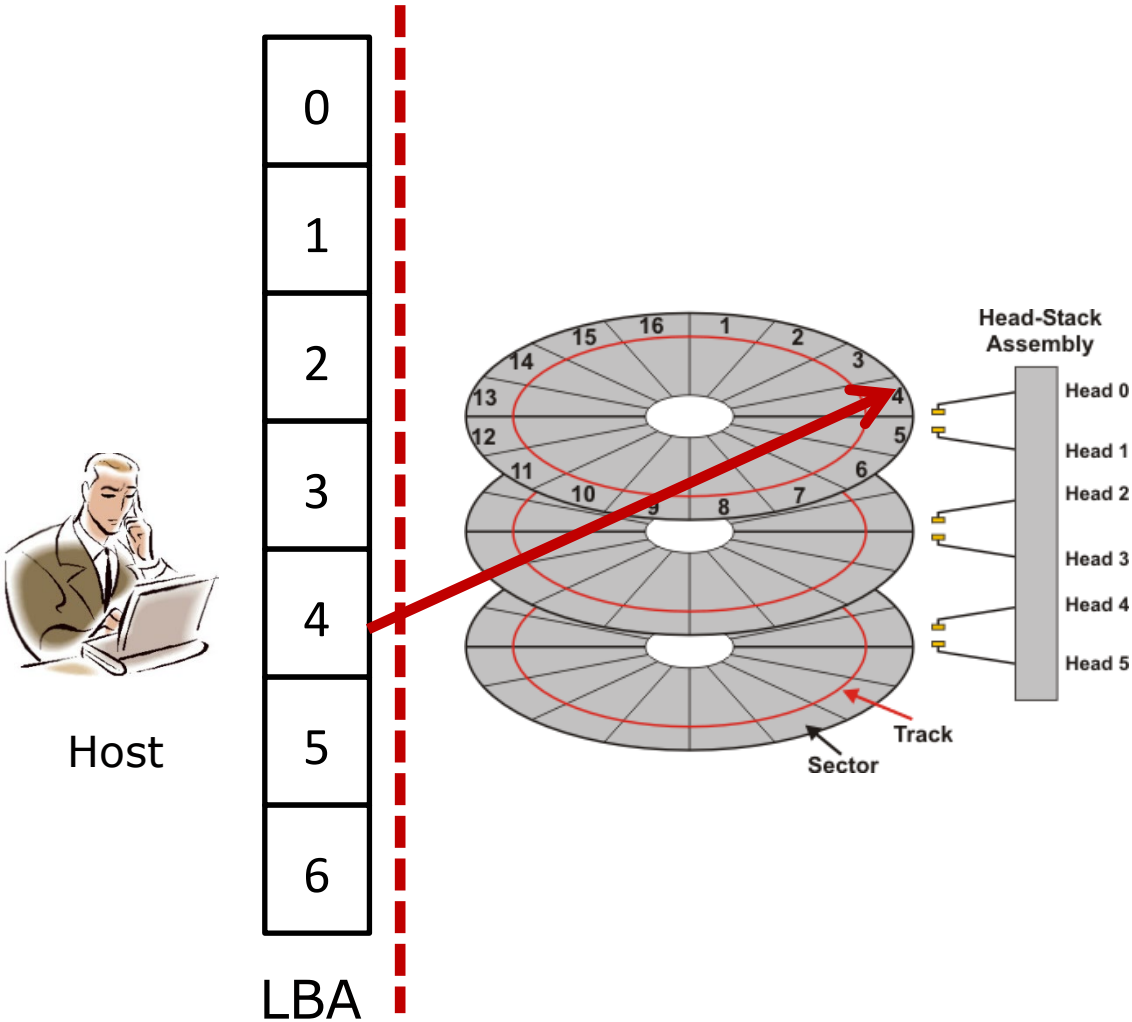


Outline

- Introduction
- Sketching SSD internals
- Hystor: A hybrid storage system
- Exploiting Internal Parallelism
- Conclusion
- Future Work



Physical data layout in HDD



```
Read (LBA, size)
Write(LBA, size)
```

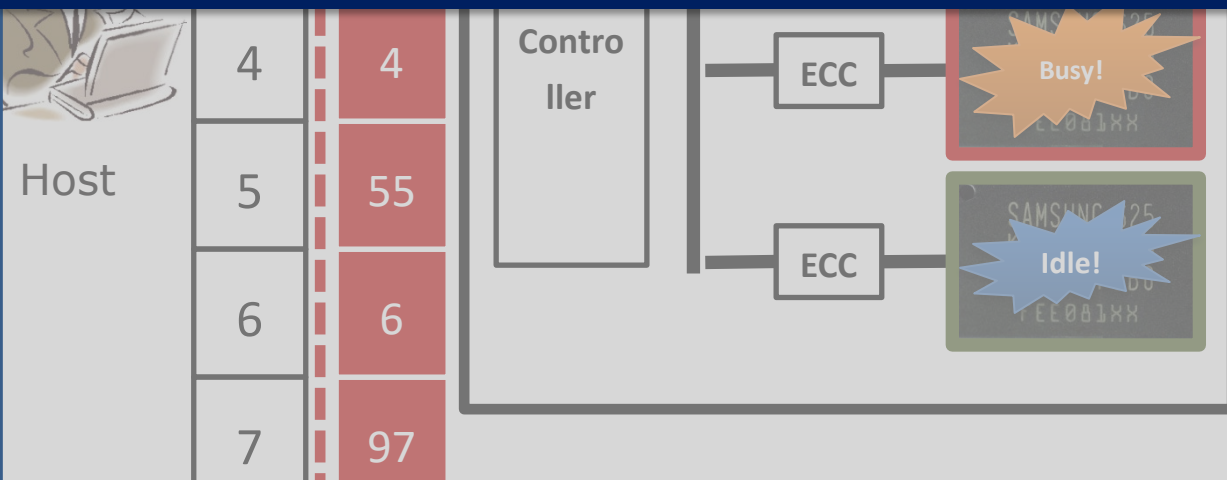
- Data are stored on the surfaces of disk platters
- An array of **logical block addresses** (LBAs) as a logical interface
- LBAs are **statically** mapped to physical block addresses (PBAs) contiguously

Physical data layout in SSD



- Data are stored in flash memory chips

The physical data mapping is an architectural feature, we must know the internal structures of SSDs.



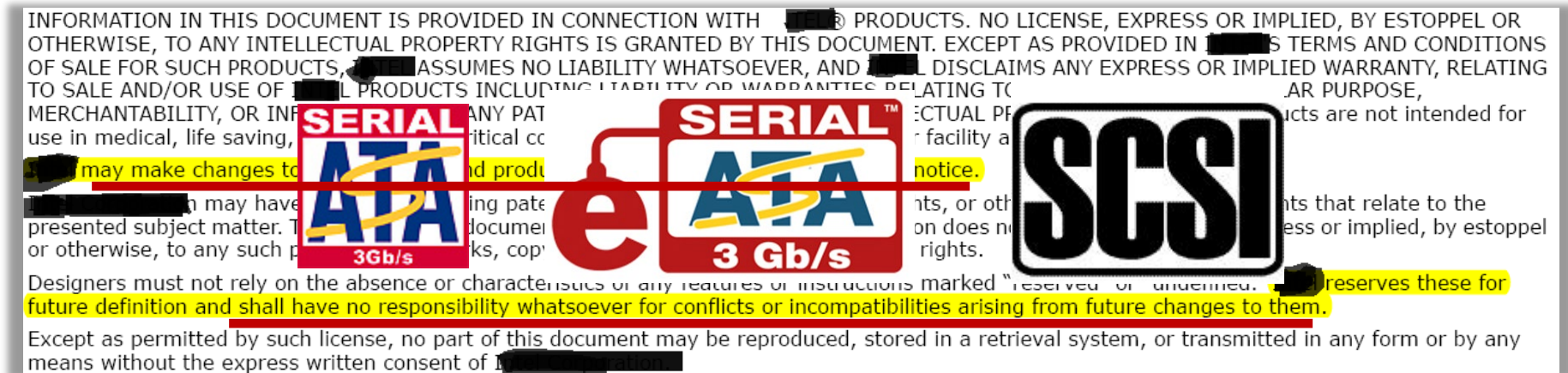
- A mapping table tracks LBA/PBA mappings
- Internal data mapping is **dynamic** on the fly

Read (LBA, size)
Write (LBA, size)

The worst case

Asking for help from SSD manufacturers?

- **Intellectual property** issues
- **Limited unreliable info** in specification
- The strictly defined **standard** interface



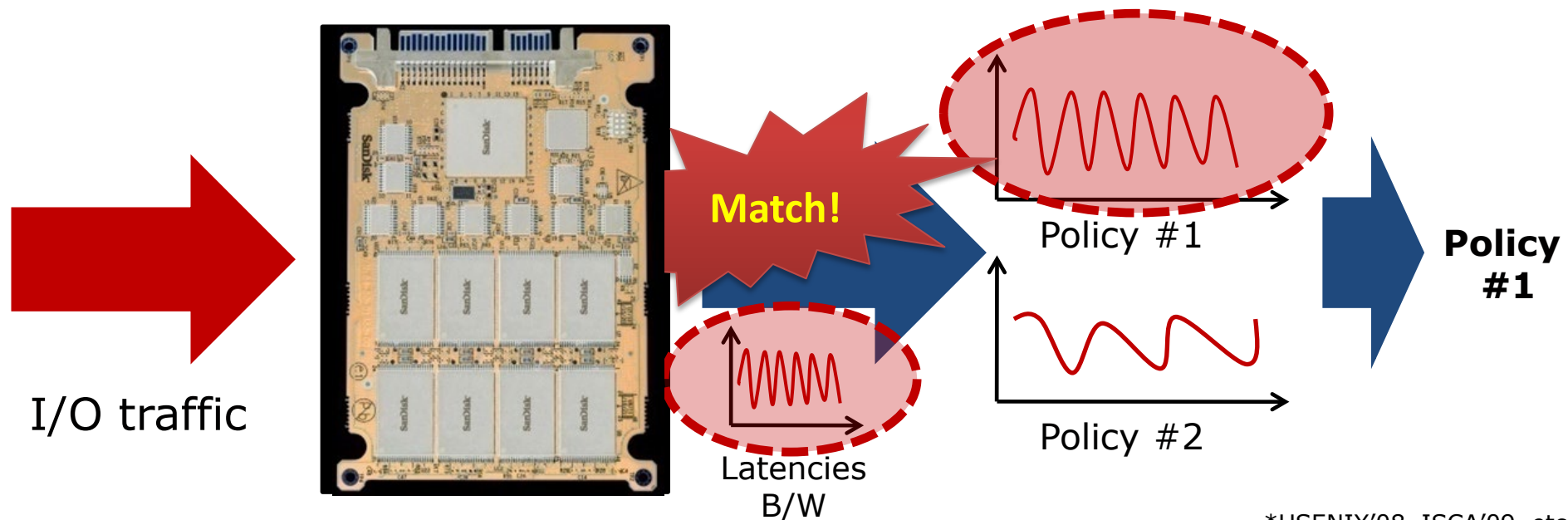
How to get the architectural-level information **without** modifying the interface and hardware?

I/O Interface connecting storage devices

- **Serial ATA (SATA)**
 - Created in 2003
 - Speed: 1.5 Gbits/s, 6.0 Gbits/s and 16 Gbits/s.
 - Widely used in industrial and embedded applications
- **eSATA**
 - A SATA connector accessible from outside the computer to provide signal for external storage devices
- **SCSI (Small Computer System Interface)**
 - One of earliest I/O interfaces
 - Widely used in Windows, Mac, Unix, and Linux
 - Gradually replaced by SATA
- **PCIe (Peripheral Component Interconnect Express)**
 - A high speed I/O interface
 - V1x (4GB/s, 16 links), v2. x (8GB/s. 15 links), v3.0 (15.75 GB/s. 16 links), v4.4 (31.51 GB/s, 16 links)

Our approach

- Treat an SSD as a **black-box**
- Assume a **repeatable but unknown pattern**
- Inject I/O traffic to probe the device
- Observe the reactions of SSD in B/W and latencies
- Enumerate possible policies based on open documents*
- **Speculate** the internal structures



A general model



Intel® X25-E SSD

Domain ✓

- A set of connected chips (how many?)

- 10x domains

Resource sharing:
e.g. channel, ECC, et. al.

- Write unit in a data block (how large?)

- 4KB (the page size)

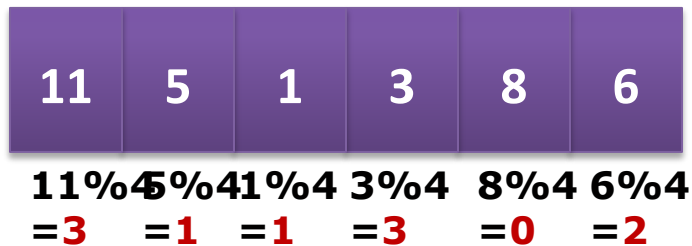
• Mapping ✓

- How chunks are allocated from their LBA to their PBA?

- To be discussed later

LBN-based Mapping

Incoming writes (LBN)



11	5	1	3	8	6
$11\%4$	$5\%4$	$1\%4$	$3\%4$	$8\%4$	$6\%4$
=3	=1	=1	=3	=0	=2

8	Domain #0	
1	5	main #1
6	Domain #2	
3	11	main #3

$N=4$

- A block with a logical block number (**LBN**)
- **N** domains
- The mapping domain (**$LBN \bmod N$**)

Write-order based Mapping

Incoming writes (LBN)



11	5	1	3	8	6
$5\%4$	$4\%4$	$1\%4$	$3\%4$	$8\%4$	$6\%4$
=1	=0	=3	=2	=1	=0

6	5	main #0
8	11	main #1
3	Domain #2	
1	Domain #3	

$N=4$

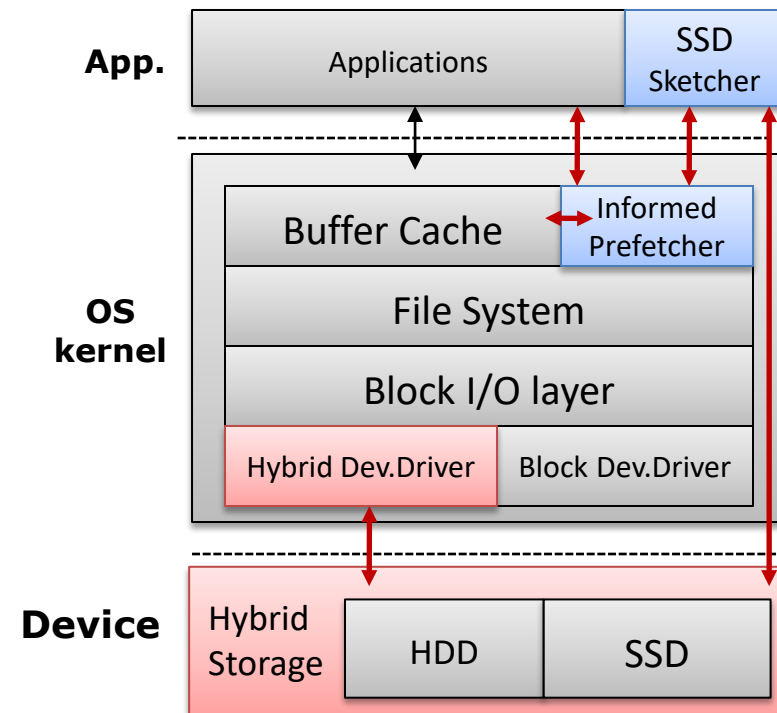
- The T_{th} block being written in a sequence
- N domains
- Domain: $(T \bmod N)$

SSD Sketcher

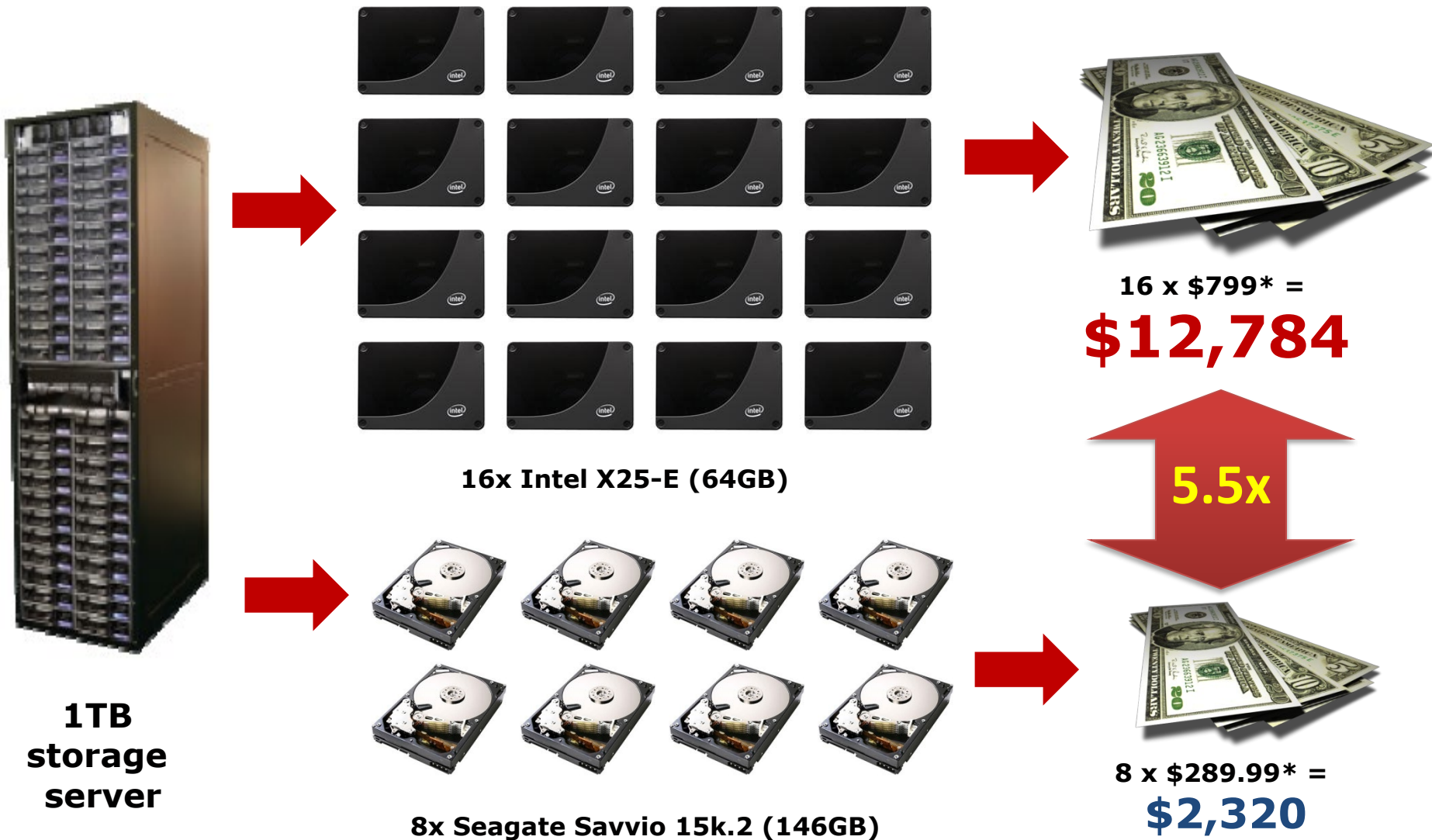
- **A software tool for users to detect the unknowns of SSD**
- **The Intel OS group would not know the IP secrets of their products.**

Outline

- Introduction
- Sketching SSD internals
- **Hystor: A hybrid storage system**
- Exploiting Internal Parallelism
- Conclusion
- Future Work

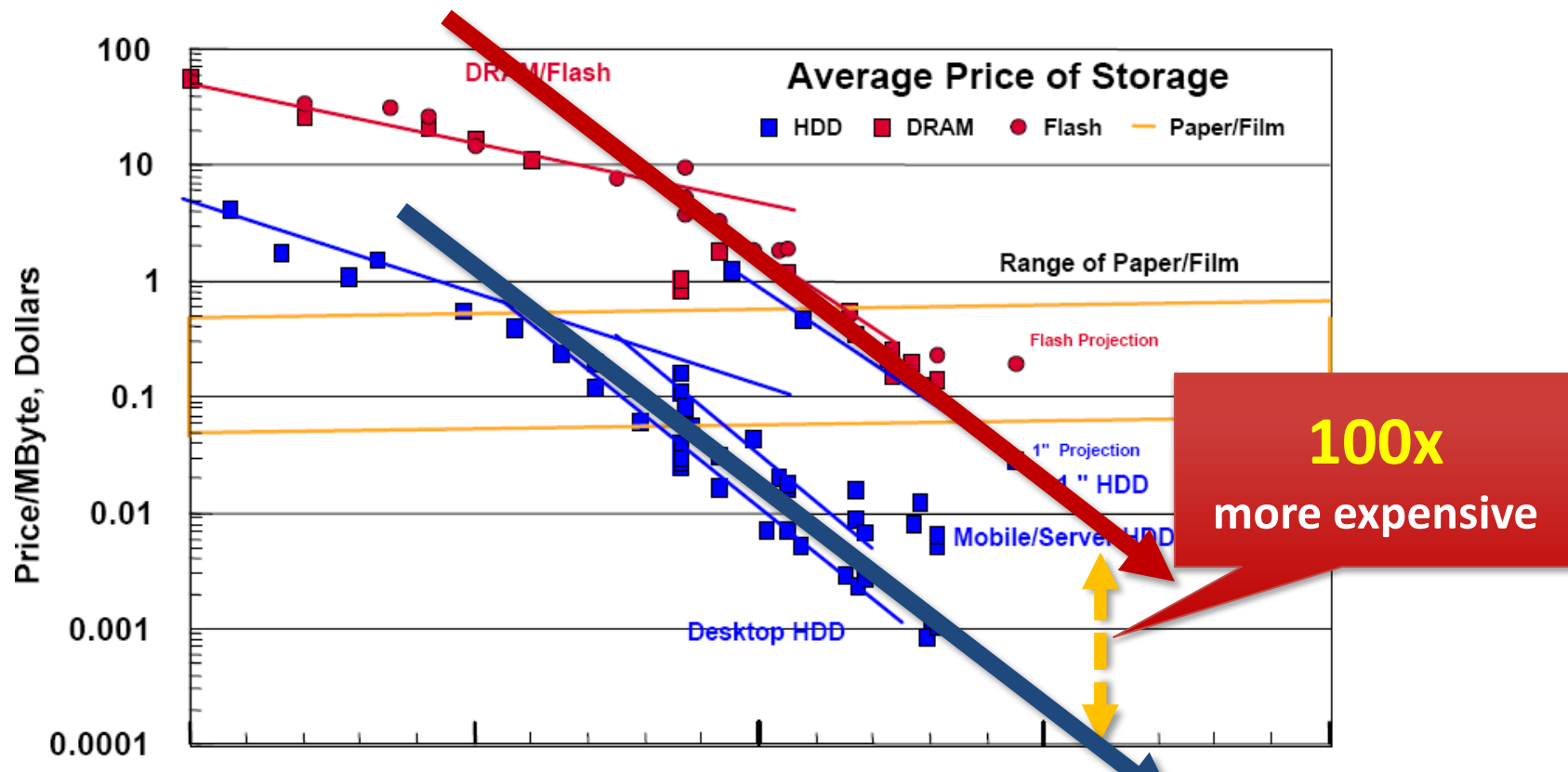


Cost-sensitive commercial systems



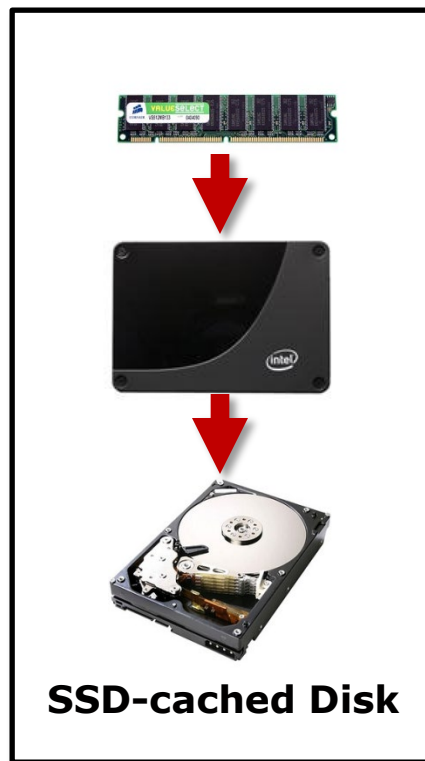
Price gap between SSD and HDD

- Flash is about **100x** more expensive than disks



*We need to find a middle ground between SSD and HDD and strike a **right balance** between performance and cost.*

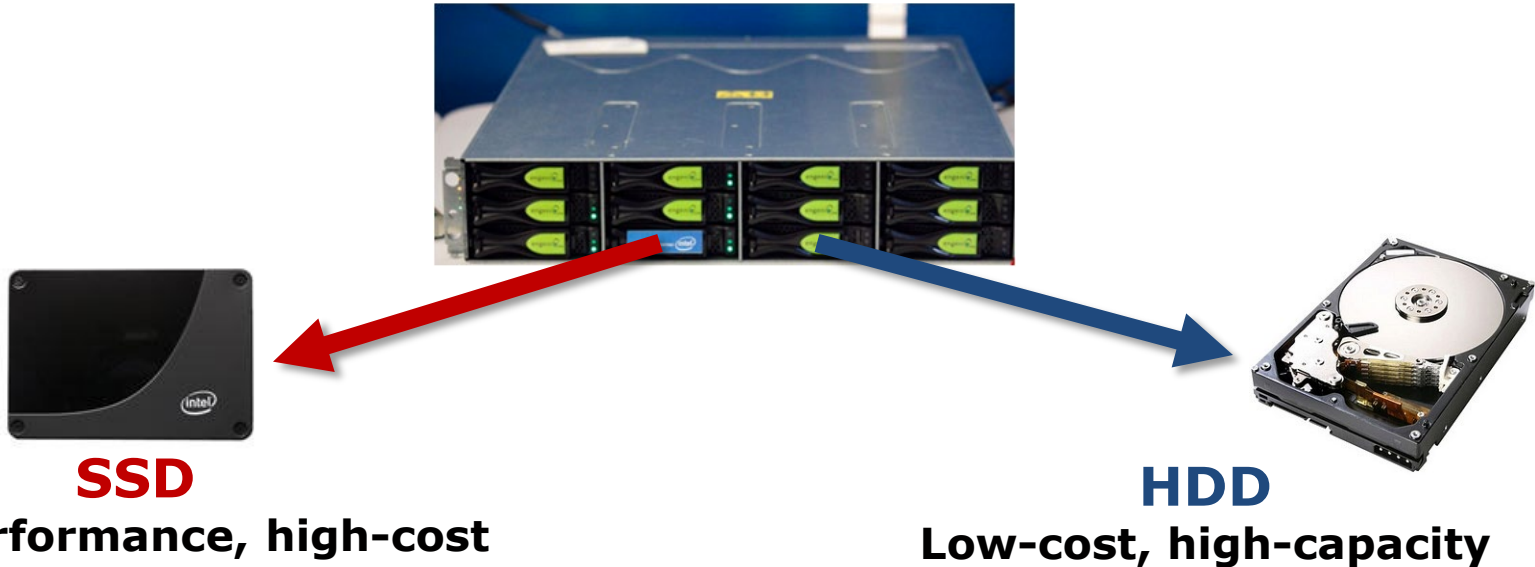
Integrating SSD and HDD together



- Cache-based solutions
 - SSD – a secondary-level cache
 - HDD – the permanent storage
 - Cache replacement policy
- Limitations
 - Weak locality memory misses
 - Intensive write traffic
 - Non-trivial system changes
 - High-cost on-line replacement
 - Frequent on-access updates
 - 10-20x Larger SSD space

- Conquest [USENIX'02]
- SmartSaver [ISLPED'06]
- ReadyBoost [MS'06]
- TurboMemory [ToS'08]
- L2ARC [CACM'08]
- FlashCache [ISCA'08]
- other ...

Hystor: A cost-efficient hybrid storage*



- A small data set
 - **Semantically critical** – F/S metadata blocks
 - **Performance critical** – High-cost data blocks

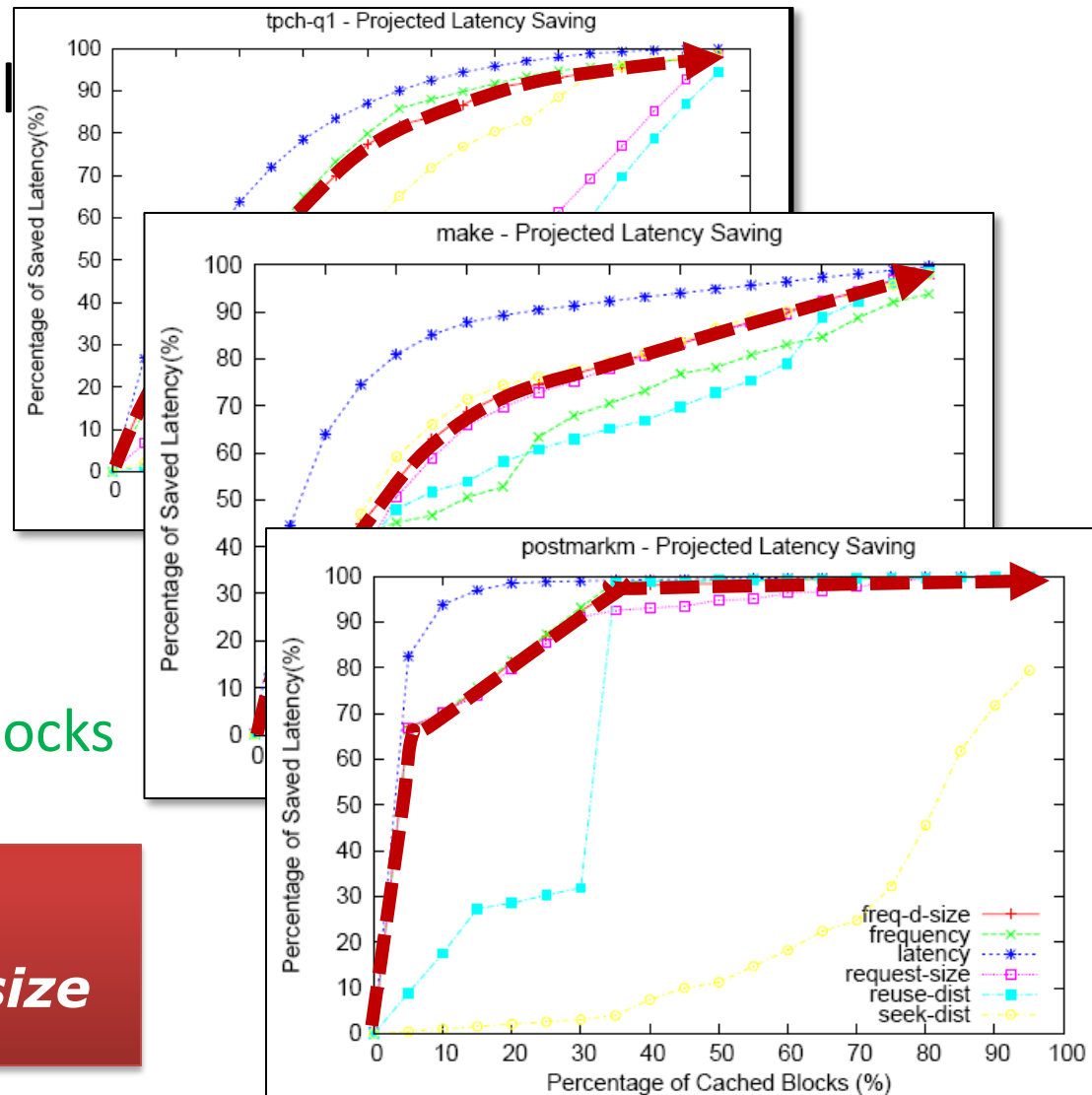
- A large data set
 - Low-priority data (e.g. movie files)

A prototype system at Intel® Labs for future storage system solution.

Identifying the high-cost data blocks

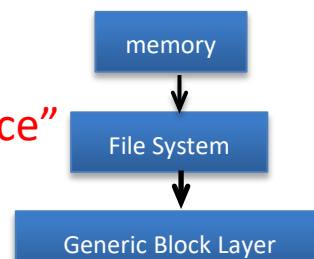
- A metric highly correlated with latency
 - Latency (**optimal**)
 - Frequency
 - Request size
 - Reuse distance
 - Seek distance
 - combinations
- Frequently used small blocks

The best metric:
Frequency/Request size



The prototype system of **Hystor***

- Implementation
 - **Kernel module** in the kernel 2.6.25.8
 - Core code: 2,500 lines
 - Kernel-level monitor: 4,800 lines
 - Merge SSD/HDD as a **single “block device”**
 - 50+ lines in stock kernel
- A pseudo device driver
 - /dev/mapper/hybrid
- Inline Tracer
 - Intercepts I/O operations
- Monitor
 - Updates the block table
- Data Mover
 - Reorganizes data layout



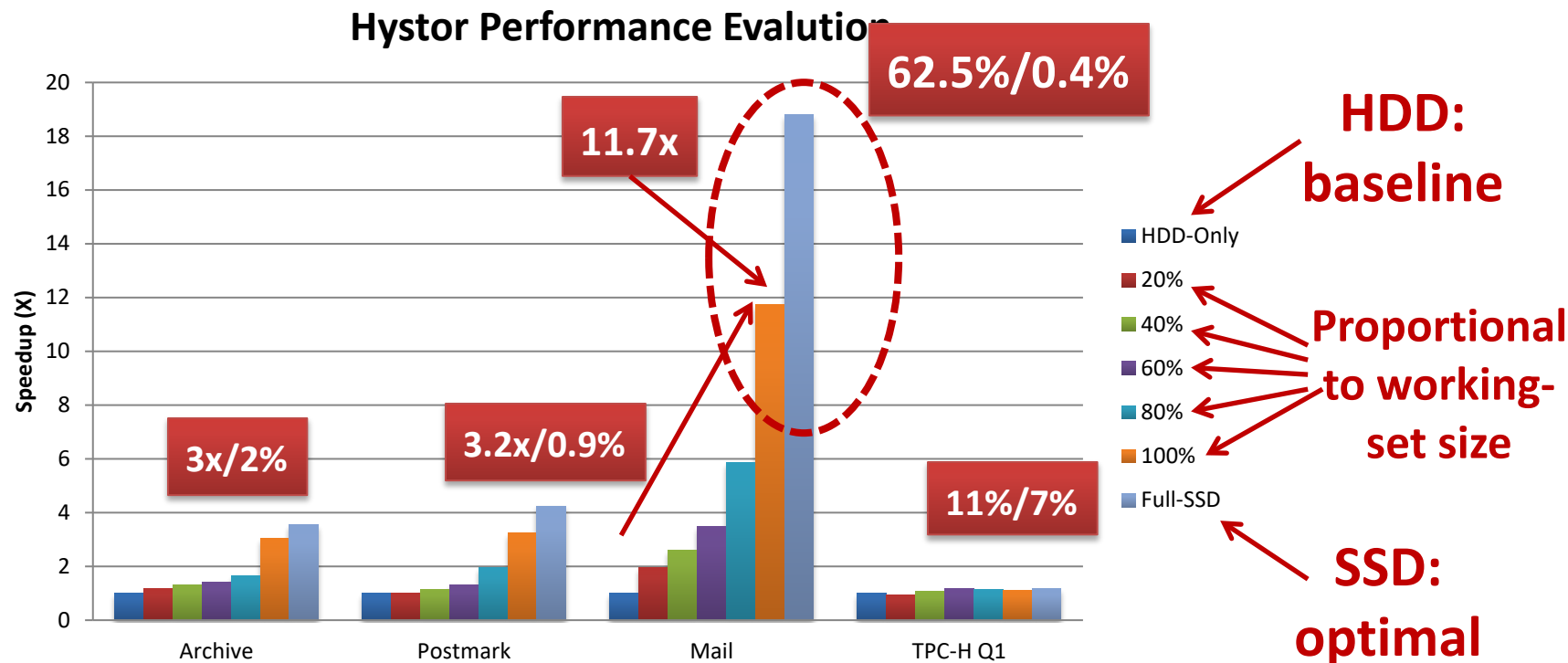
Future plan: Prototyping a hardware hybrid storage system

Performance Evaluation

- Measurement System
 - Intel® D975BX, 2.66GHz Intel® Core™ 2 Quad, 4GB Memory
 - LSI® MegaRAID 8704 SAS Card, Seagate® 15k.5 SAS HDD, Intel X25-E SSD
 - Fedora Core 8 Linux, Linux Kernel 2.6.25.8
- Experimental Results
 - **Archive**: compare two Linux source code tree 2.6.22.5/2.6.22.6 and tar one
 - **Postmark**: a file system benchmark (100 dir., 20,000 files, 100,000 trans.)
 - **Mail**: E-mail server, U Michigan benchmark (500 dir., 500 files, 5,000 trans.)
 - **Database workloads**: TPC-H Q1 on PostgreSQL 8.1.4 (scans LINEITEM table)

Performance Evaluation

- Measurement System
 - Intel® D975BX, 2.66GHz Intel® Core™ 2 Quad, 4GB Memory
 - LSI® MegaRAID 8704 SAS Card, Seagate® 15k.5 SAS HDD, Intel X25-E SSD
 - Fedora Core 8 Linux, Linux Kernel 2.6.25.8
- Experimental Results



Impact of Hystor

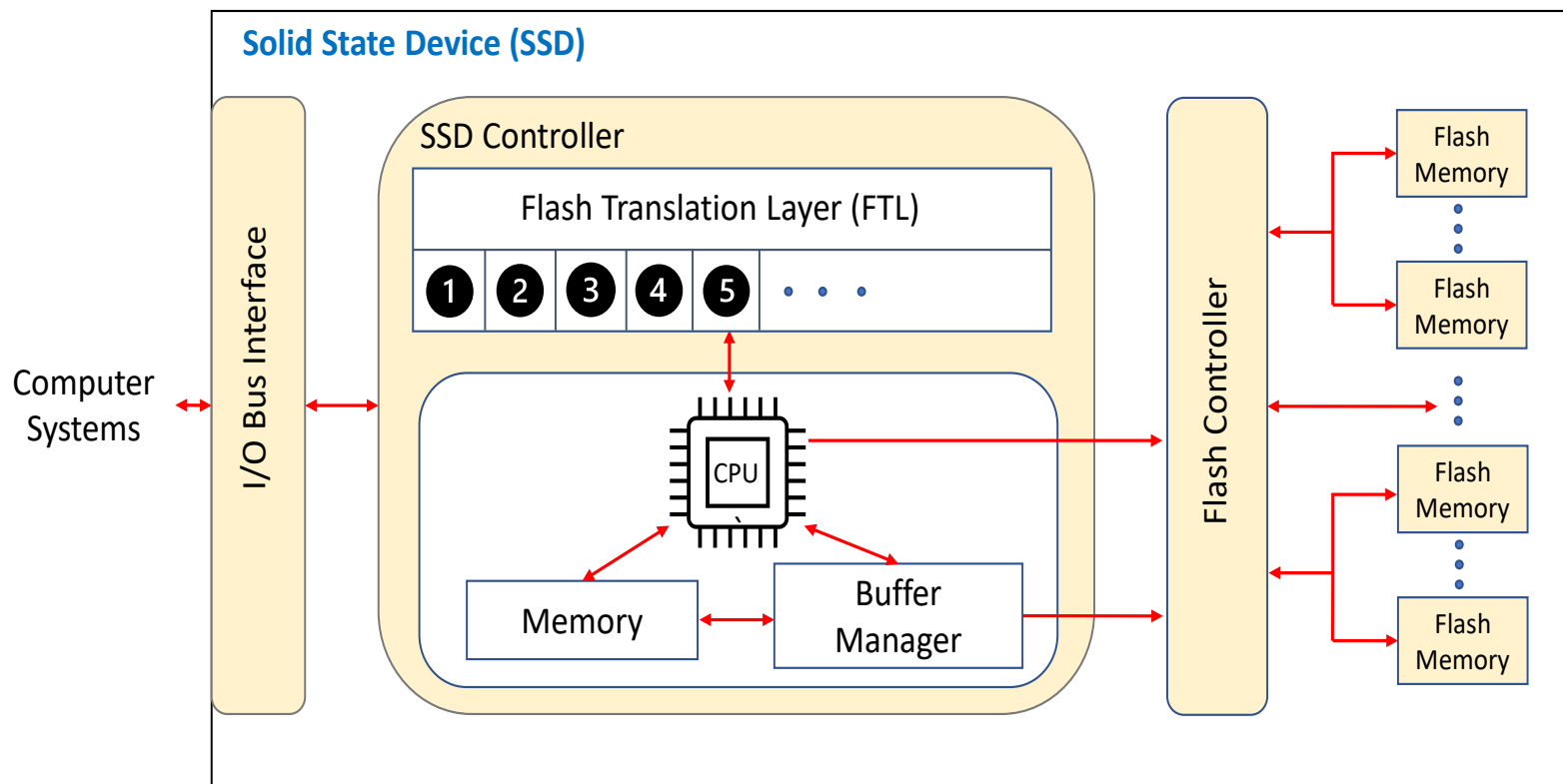
- **Hystor** received the Best Paper Award in 2011 ACM International Conference on Supercomputing (ICS 2011)
- **Hystor** has made impact on Apple hybrid storage product **Fusion Drive**



- A new storage product on the market since October 23, 2012
- Consisting of a small SSD and a large hard drive
- The hybrid storage is managed by OS in a single space

- Comments on **Hystor** from Apple:
 - Hystor is a well-designed system, and its paper discussed several key systems trade-offs in details. The Apple software engineers had carefully and systematically evaluated Hystor. This work had a significant influence in the design of Apple's Fusion Drive. Some design elements and algorithms in Hystor have been directly used in Apple's Fusion Drive.*
- **Steve Jobs'** Philosophy on technology transfer for Apple:
 - Picasso had a saying "Good artists copy, great artists steal". We have always been shameless about stealing great ideas ...*

A Summary SSD Architecture



- 1 Logical-Physical Address Mapping
- 2 Wear-Leveling
- 3 Garbage Collection
- 4 SSD Parallelism Management
- 5 Error Control Code (ECC)