NAND Flash & SSDs

Angelos Bilas
FORTH-ICS
and University of Crete, Greece
bilas@ics.forth.gr

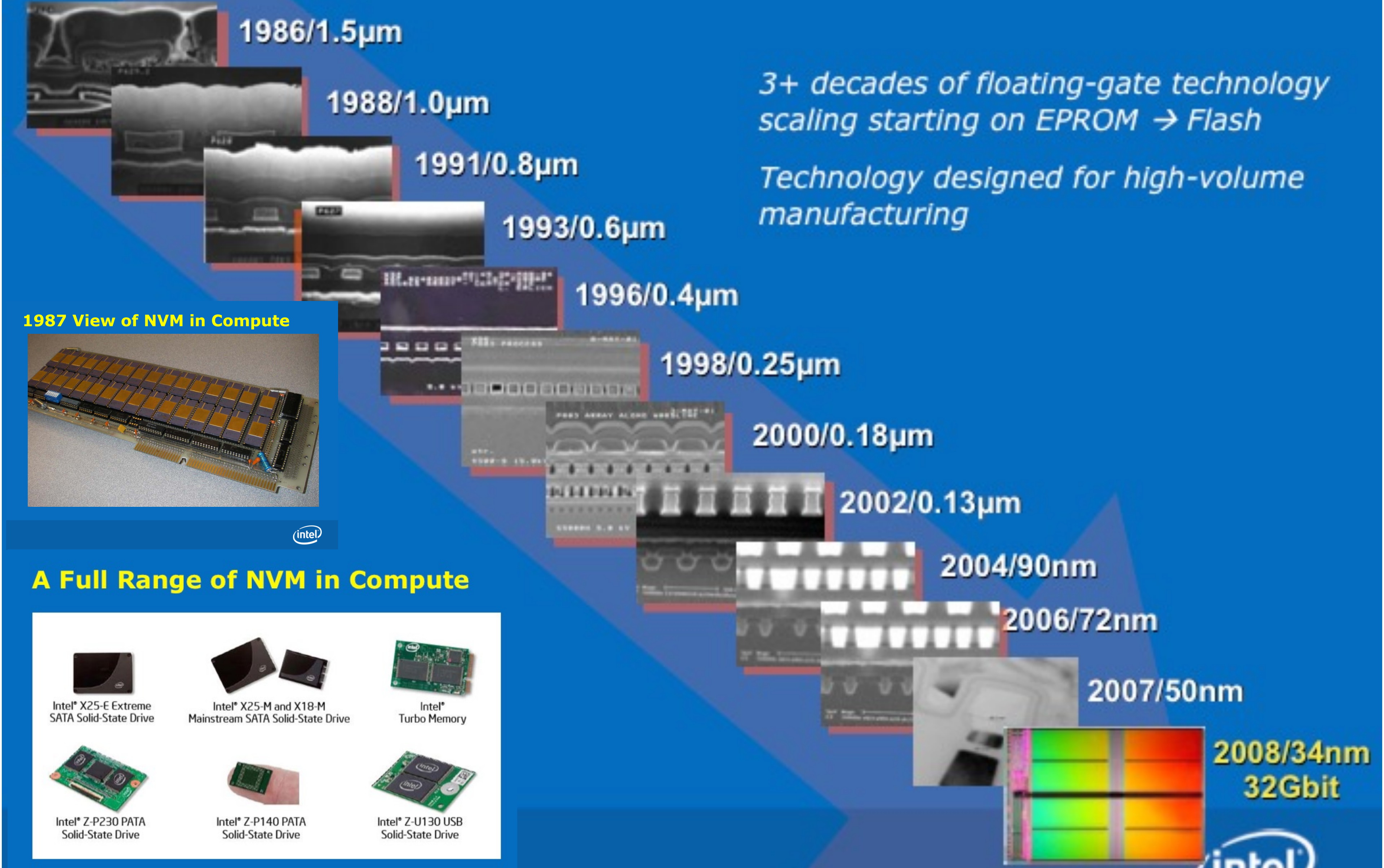
Material

- Slides
  - Toshiba slides (first NAND in 1989)
  - Al Fazio, Intel Fellow, Director, Memory technology Development, November 12, 2008
- Reading
  - (SSDs) Nitin Agrawal et al. Design tradeoffs for SSD performance. In USENIX 2008 Annual Technical Conference (ATC'08).
  - (NVM Intro) Natarajan, S. et al., "Searching for the dream embedded memory," Solid-State Circuits Magazine, IEEE , vol.1, no.3, pp.34,44, Summer 2009.
  - (Flash Intro) Micheloni, R. et al., "Non-Volatile Memories for Removable Media," Proceedings of the IEEE , vol.97, no.1, pp.148,160, Jan. 2009.
  - (edited book) Inside Solid State Drives (SSDs) by R. Micheloni et al., Springer Verlag.

Outline

- NAND Flash
  - How it works
  - Structure
  - Errors
- SSDs
  - Structure
  - Garbage collection
  - Wear leveling
- System-level SSD Storage
  - SSD caching
- Remarks
  - Non-volatile memories (NVM)

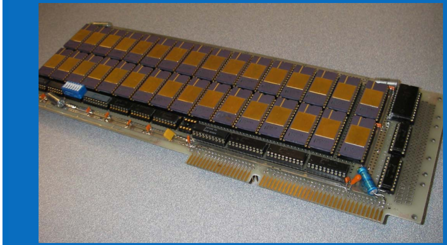
20+ Years Flash Floating Gate Technology



3+ decades of floating-gate technology scaling starting on EPROM → Flash

Technology designed for high-volume manufacturing

1987 View of NVM in Compute



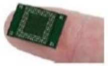
A Full Range of NVM in Compute

Intel® X25-E Extreme
SATA Solid-State Drive

Intel® X25-M and X18-M
Mainstream SATA Solid-State Drive

Intel®
Turbo Memory

Intel® Z-P230 PATA
Solid-State Drive

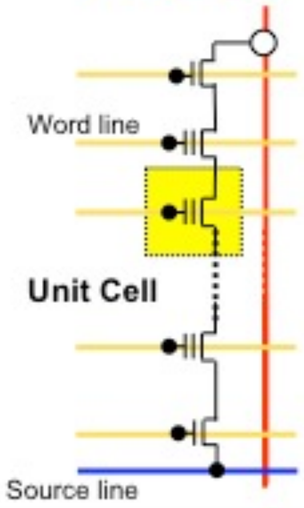
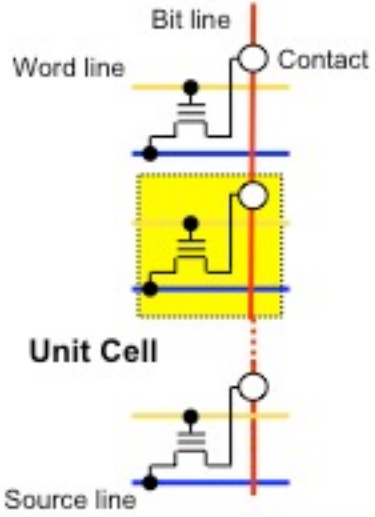
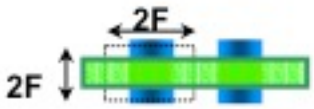
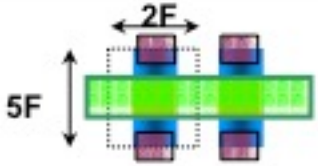
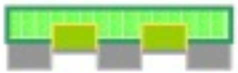
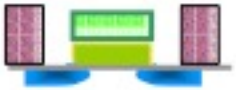


Intel® Z-P140 PATA
Solid-State Drive

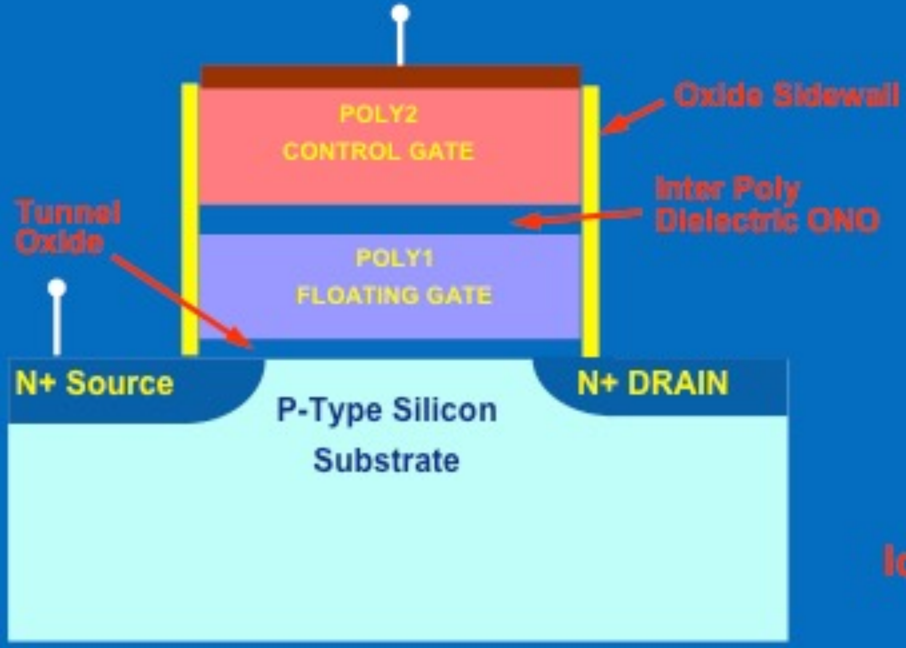
Intel® Z-U130 USB
Solid-State Drive

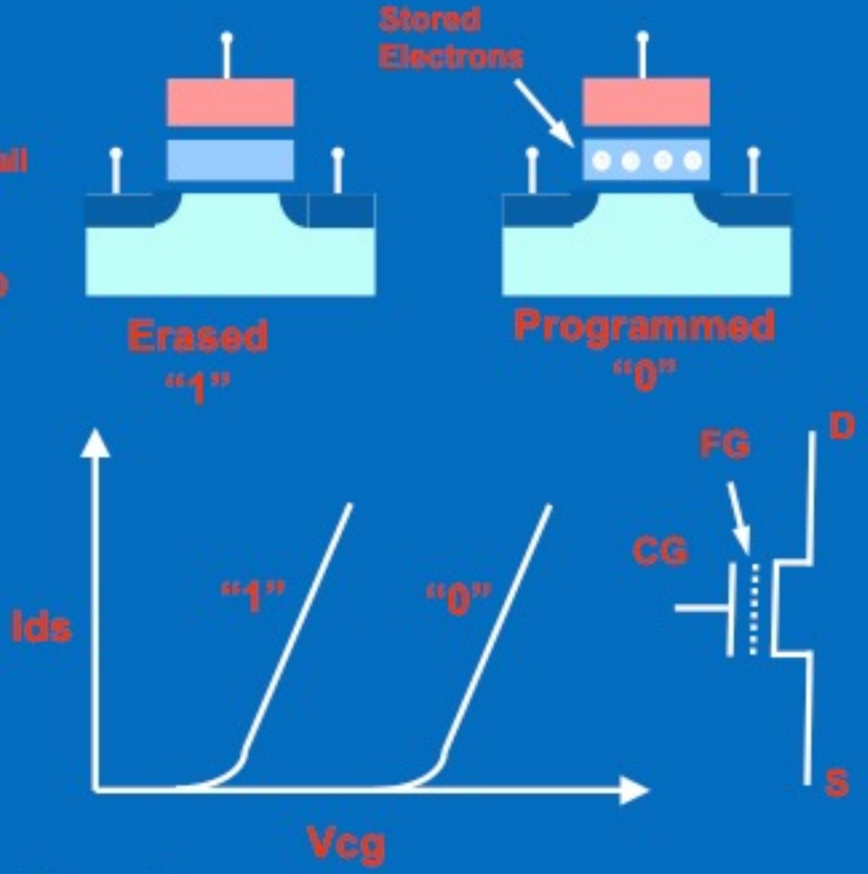
Source: Intel

NAND vs. NOR - Cell Structure

	NAND	NOR
Cell Array		 Bit line Contact Word line Unit Cell Source line
Layout		
Cross-section		
Cell size	$4F^2$	$10F^2$

Flash Cell Layout and Cross-Section

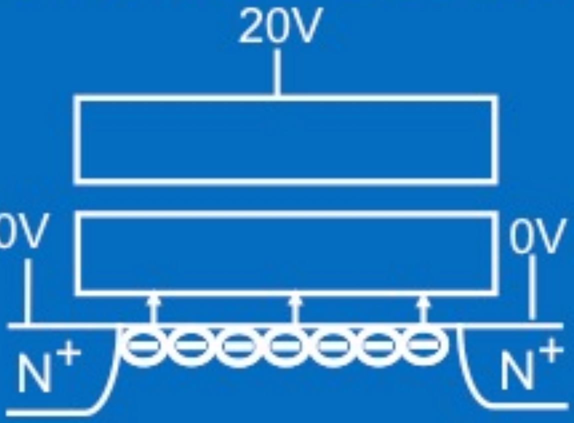




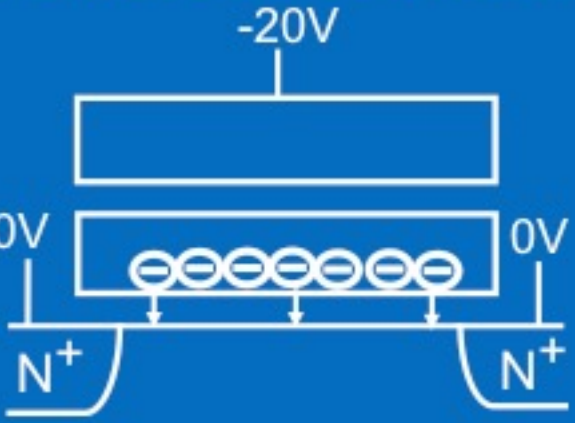
N-Channel MOSFET with a few distinguishing features:

- Isolated floating gate
- Charge storage on Floating gate modulates threshold voltage of underlying MOSFET

Charge Storage: Program and Erase



Programming: NAND



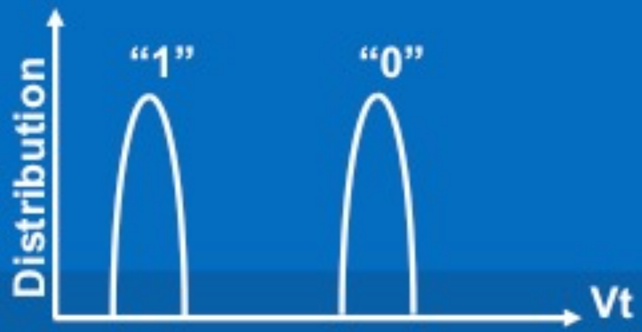
Erase

Programming means injecting electrons to the FG

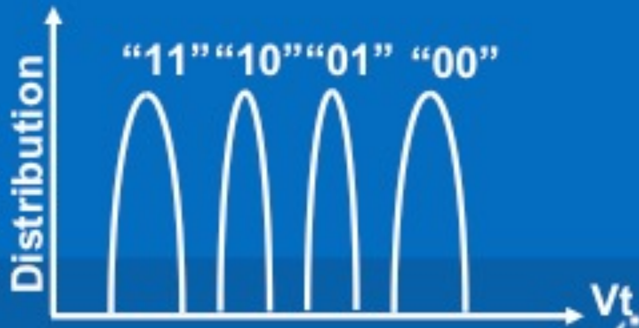
- Fowler-Nordheim Tunneling

Erase: Fowler-Nordheim Tunneling in reverse direction

2 Levels => 1 bit/cell



4 Levels => 2 bit/cell



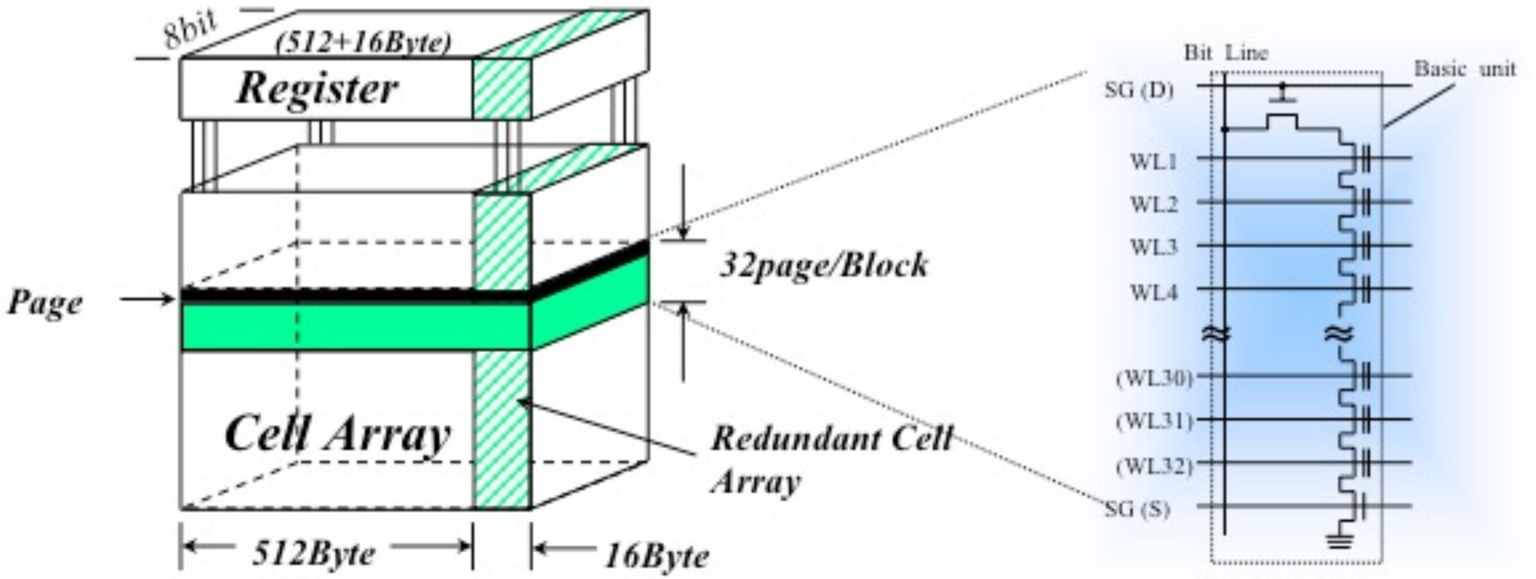
Reliability

- Three main types of problems
- 1. Oxide degradation over time (data retention)
  - No accesses – FLASH on the shelf
  - Oxide degradation leads to bit errors
  - Unused FLASH, typical retention time 10 years
  - BUT depends on how much the material has been used (erase/prog cycles)
- 2. Oxide degradation due to erase/write
  - Main problem so far
  - Between 10,000 and 100,000 erase cycles
  - Wear leveling used to compensate (will discuss)
- 3. Bit errors due to reading neighboring cells
  - Read disturbance
  - Until now smaller problem, but starting to be relevant
  - Depends on how much the material has been used (erase/prog cycles)
- Read/program/erase process is electrically complex
  - Includes various mechanisms
  - Each mechanism with its own sources of error
- (NOR does not suffer from these problems)

TOSHIBA

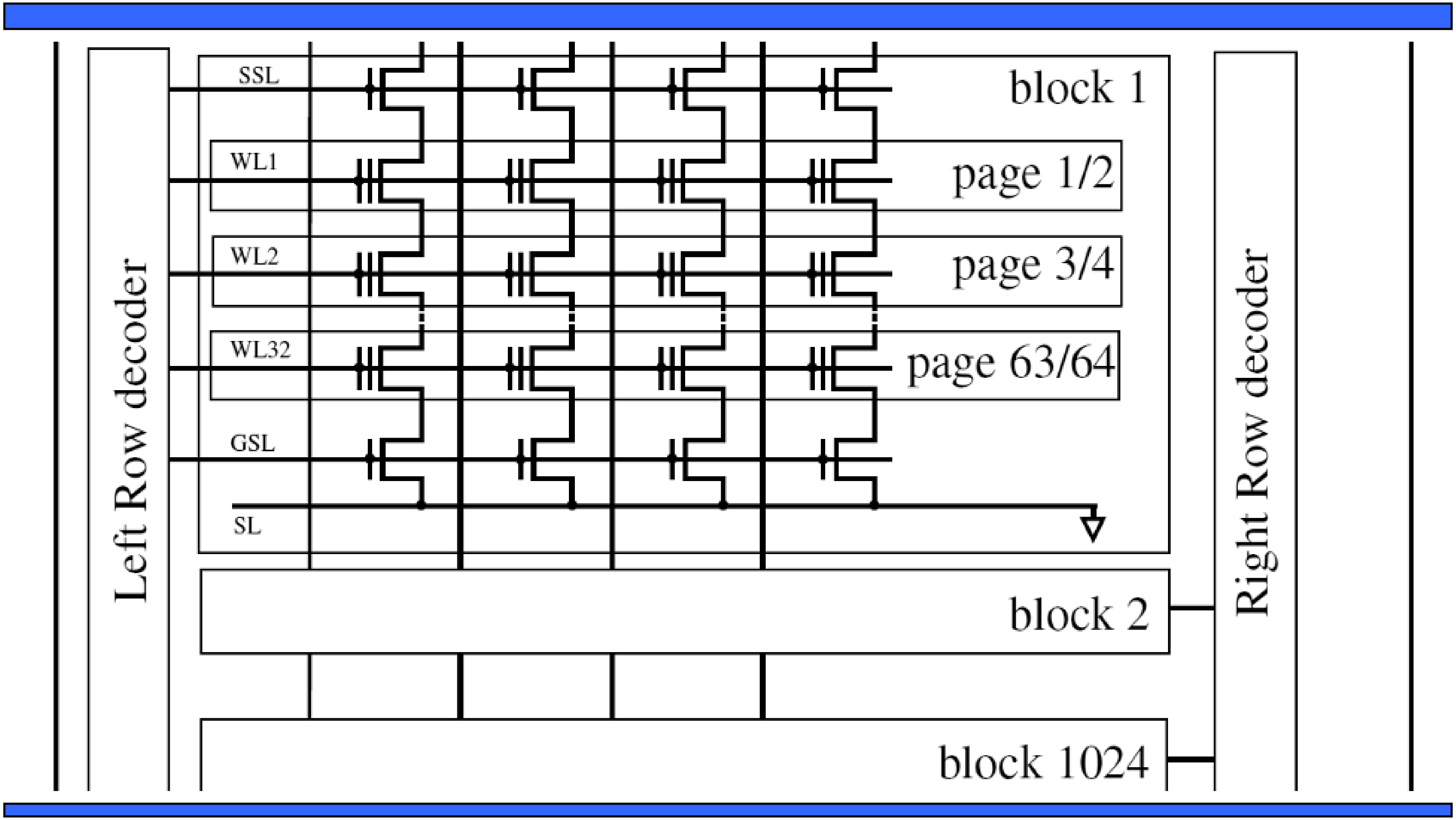
NAND Flash Memory Block Diagram

ex.256Mb NAND Flash Memory



256Mb NAND Flash
- Page Size : 512+16 Bytes
- Block Size : 16KBytes
- # of Blocks : 2048 Blocks

NAND Physical Organization



NAND Operations

- Read
  - Setup time to “reach” a page within block (sequential)
  - Read a full page at a time
  - NOR can be read “truly” randomly
- Program (1 -> 0)
  - Setup time to “reach” a page within block (sequential)
  - Program a full page at a time, set independent cell bits to 0
  - NOR can be programmed randomly but each word separate (therefore a page takes longer to program)
- Erase (0 ->1)
  - Can only erase a full block (physical organization and method)
  - Different methods to erase NAND/NOR

NAND / NOR Characteristics

	NAND	NOR
Capacity	~ 1Gbit (2chips/pkg)	~ 128Mbit
Power Supply	2.7-3.6V	2.3-3.6V
I/O	x8	x8/x16

Access Time	50ns(serial access cycle)
	25µs(random access)
		70ns(30pF, 2.3V)
		65ns(30pF, 2.7V)

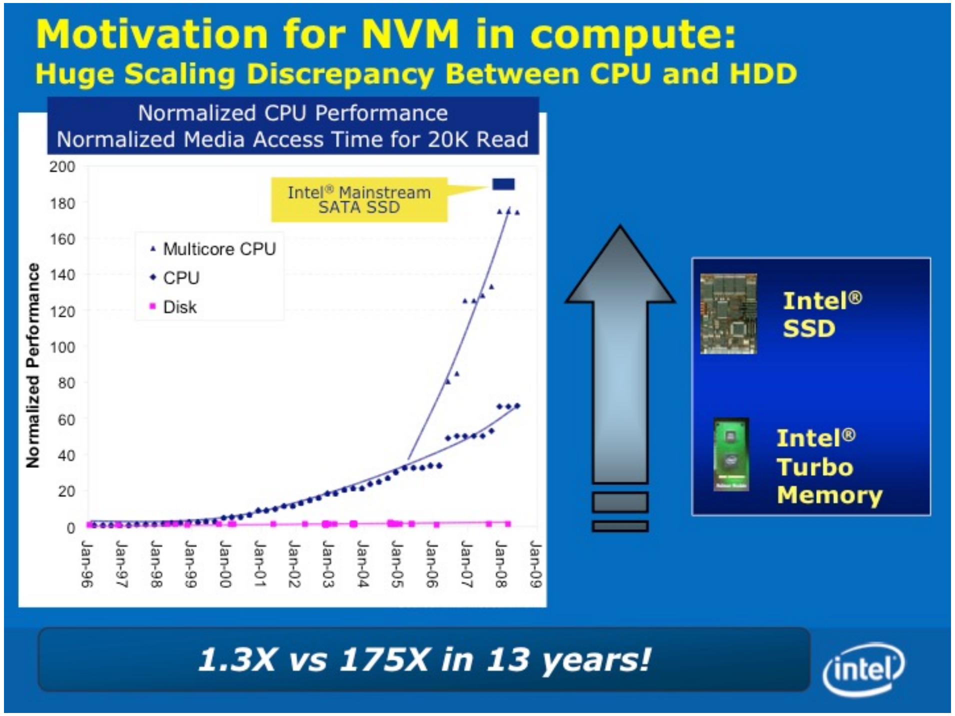
Program Speed (typ.)	——	8µs/Byte
	200µs/512Byte	4.1ms/512Byte
Erase Speed(typ.)	2ms/Block (16KB)	700ms/Block
Prog+Erase(typ.)	33.6ms / 64KB	1.23s/Block (main:64KB)

So What?

- Free lunch
  - Use NAND for large, sequential operations
    - Small, random READs are great too
  - Use NAND for workloads with few writes
  - Mobile is a good match: few updates, media files
- When FLASH is good, it's very good!
- Can we use it elsewhere?
  - General purpose computing and servers have an I/O problem
- No free lunch there
  - Small random writes + many writes
  - Reliability

Potential

- A disk is about 100 IOPS and 100 Mbytes/s
- An SSD can do 100,000 IOPS and 1000 MBy/s
- Does it matter?
- NO: use many disks
- How many?
  - 100K IOPS/core
  - 100 MBy/s/core
- A server with 128 cores is not easy to feed from disks



Motivation for NVM in compute:

Huge Scaling Discrepancy Between CPU and HDD

Normalized CPU Performance
Normalized Media Access Time for 20K Read

1.3X vs 175X in 13 years!

Flash in Storage Systems

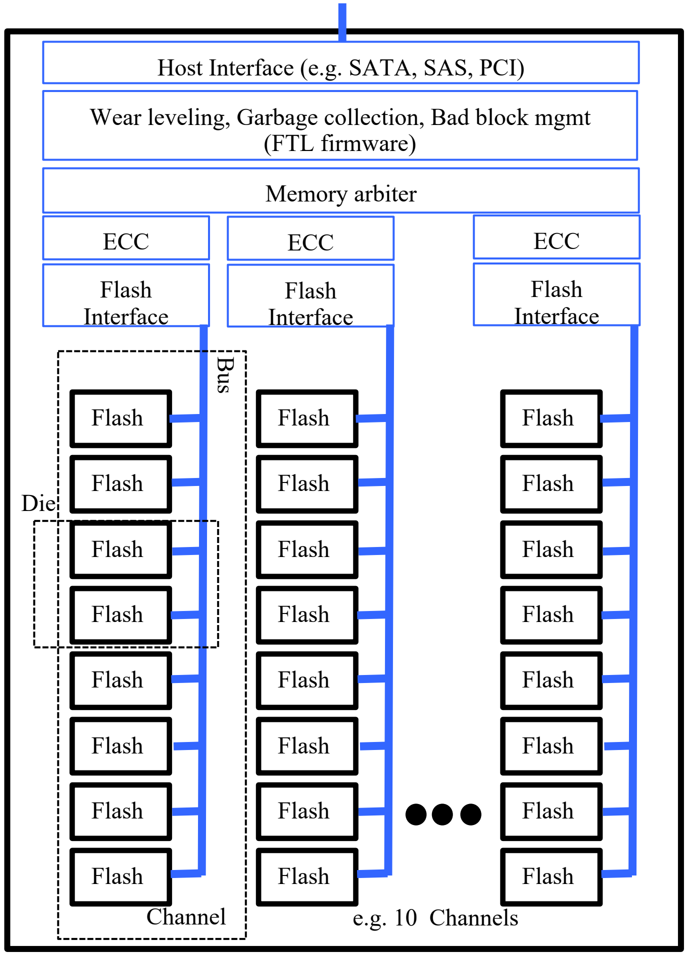
- How do we use Flash to improve storage I/O?
- Solid state drives (SSDs)

Outline

- NAND Flash
  - How it works
  - Structure
  - Errors
- SSDs
  - I/O Concurrency
  - Garbage collection
  - Wear leveling
- System-level SSD Storage
  - SSD caching
- Remarks
  - Non-volatile memories (NVM)

SSD Organization

- 10 channels
- 2 components per channel
- 4 dies per component
- Multiple outstanding commands per channel
- Error detection/correction per channel



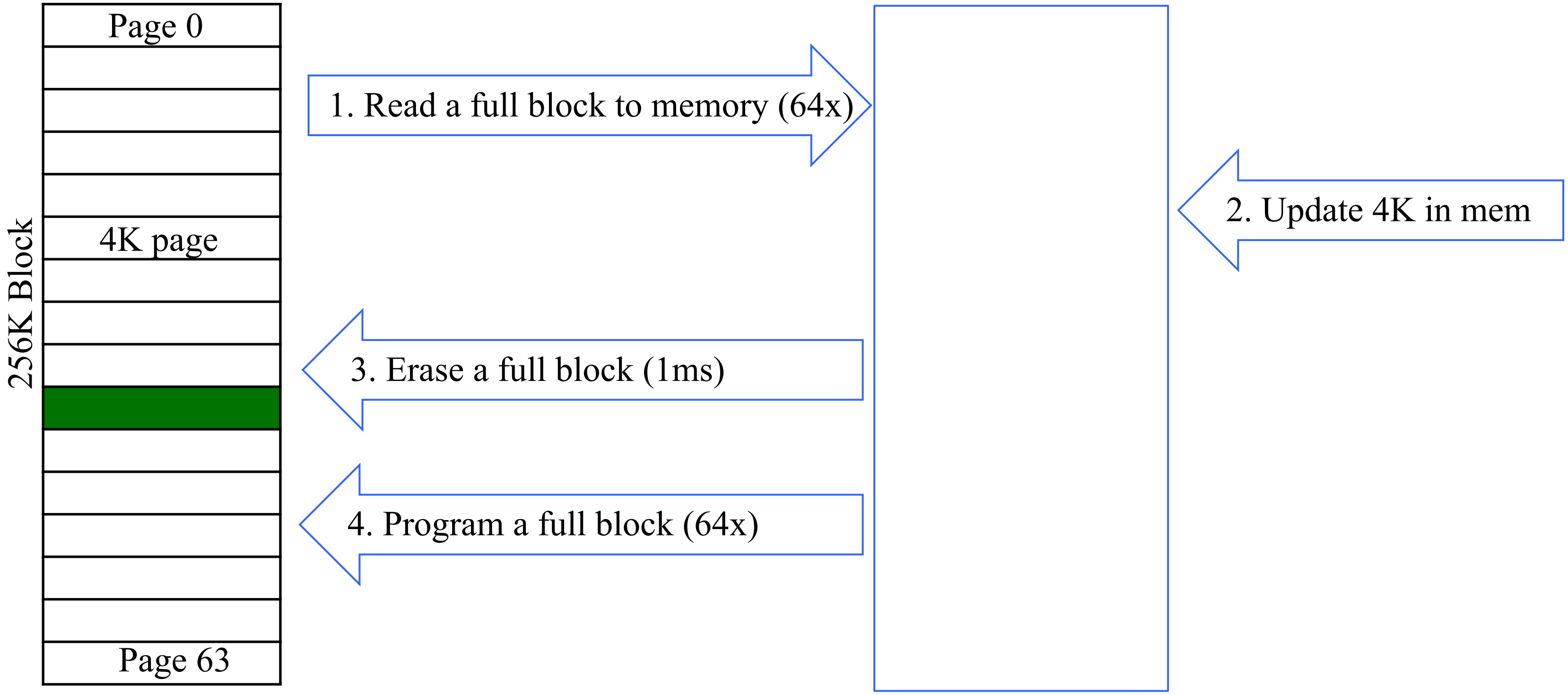
Today

- A lot of concurrency internally
- Placement in internal Flash chips to improve I/O concurrency
  - Spatial parallelism
- Optimize concurrent internal operations
  - Eg spread out mutually blocking operations
- Use lower-cost operations when possible
  - Eg some types of copying/moving faster than others

I/O writes in SSDs

- Let’s try to do a small write – where small = 4K
- Let’s say this is an update (not first write)
- What could we do?
- Find physical 4K page containing block
- Cannot write page directly – need to erase block first
- Read block to memory (read)
- Update page (modify)
- Erase block and program full block (write)
- This is a lot of overhead!

I/O writes in SSDs


Page 0
4K page
Page 63
256K Block
1. Read a full block to memory (64x)
2. Update 4K in mem
3. Erase a full block (1ms)
4. Program a full block (64x)

- Traffic 128x more
- Erase on critical path
- Read-modify-write does not work (write amplification)

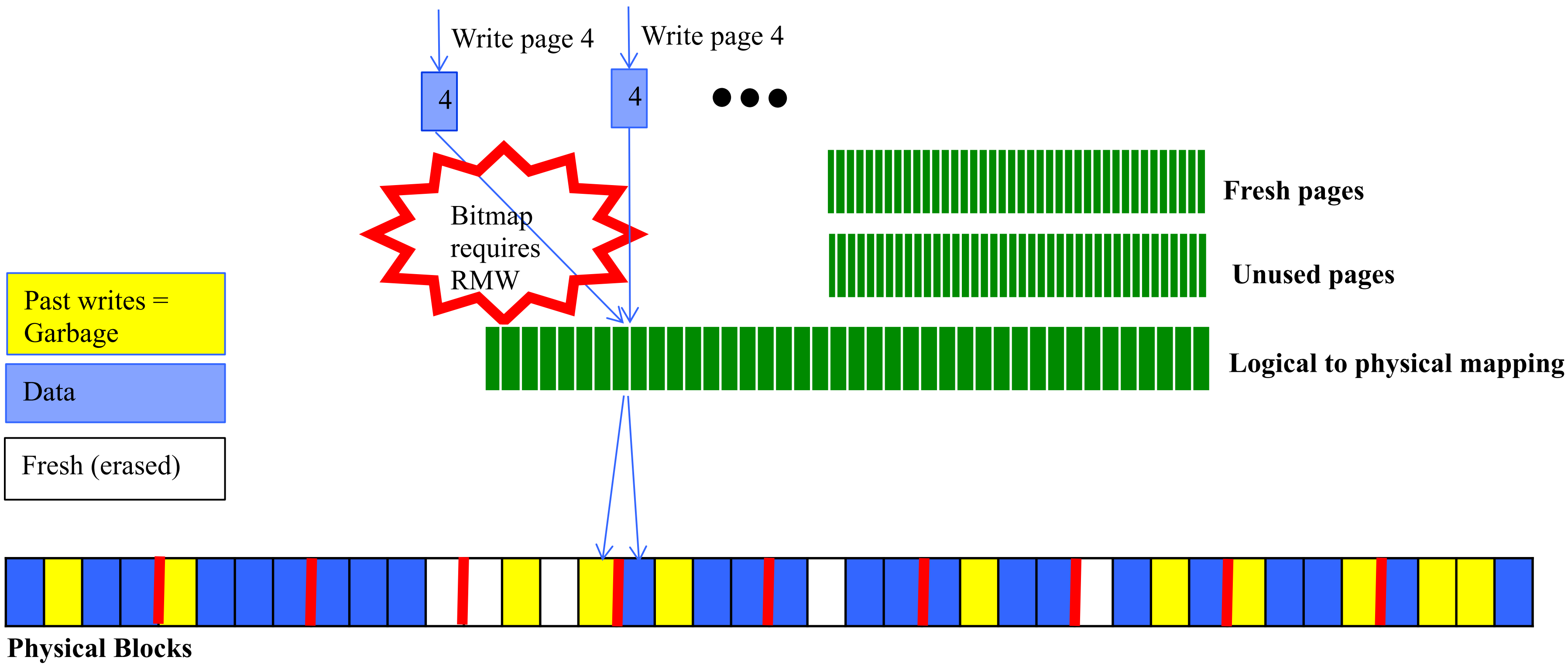
The problem

- Different erase block vs. page sizes
  - E.g. erase block size 256K - erases
  - E.g. page size 4K – updates
- Why? Large blocks => higher density for NAND

Let’s rethink

- We cannot overwrite the same page, unless we erase
- Fine, let’s find another page in a block that has been fresh erased
- Write 4K to this page
- Remap page to new physical page
- Free old page
- Essentially, introduce a logical-physical page mapping
  - First time ever in a storage device for all blocks
  - Requires a lot and non-trivial metadata – like a file system

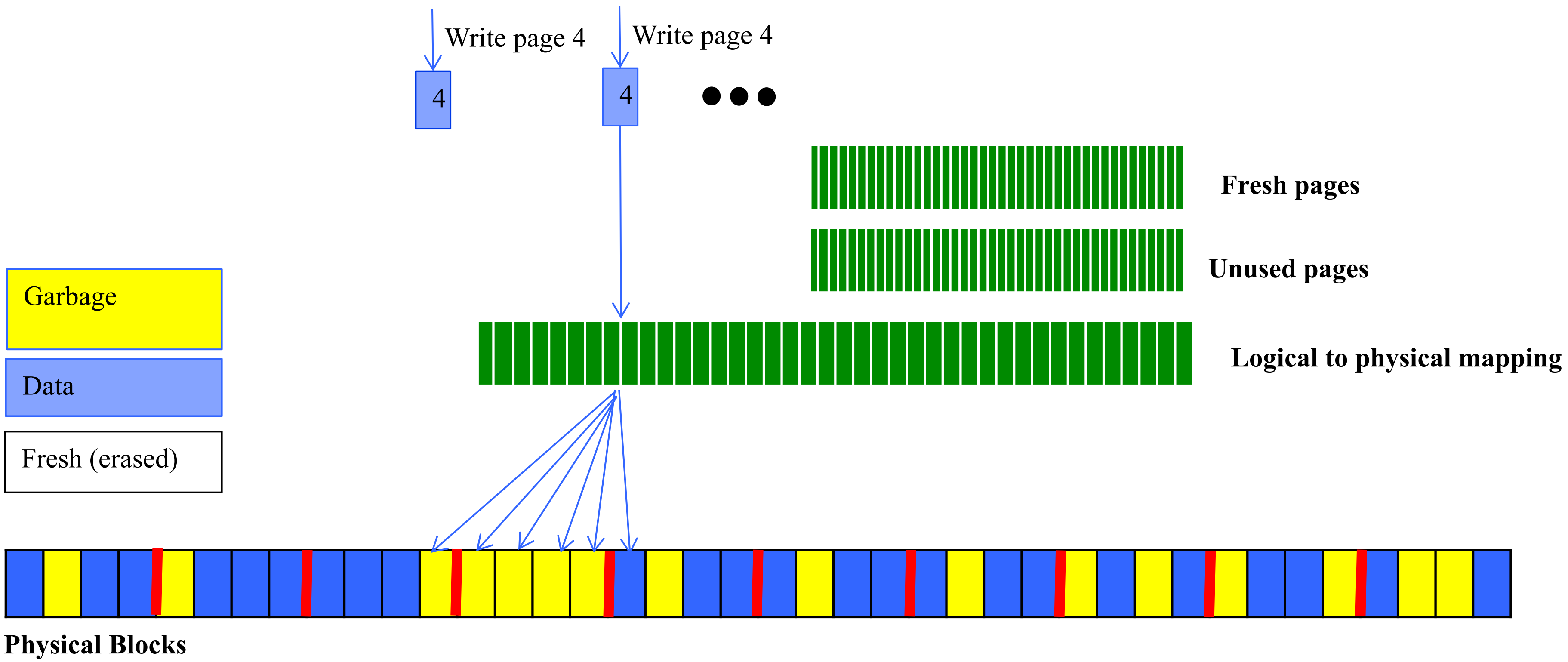
Logical to physical mapping



After a while…

- We end up with:
- Erased blocks that were empty will fill up
- Pages that were re-written will be free leaving holes in various blocks here and there
- At a next write, there will be no location to put the data
  - Cannot write a “hole”, need to erase first
  - Cannot erase a “hole”, need to erase the full block

No “fresh” pages



Write page 4

4

Write page 4

4

●●●

Fresh pages

Unused pages

Logical to physical mapping

Garbage

Data

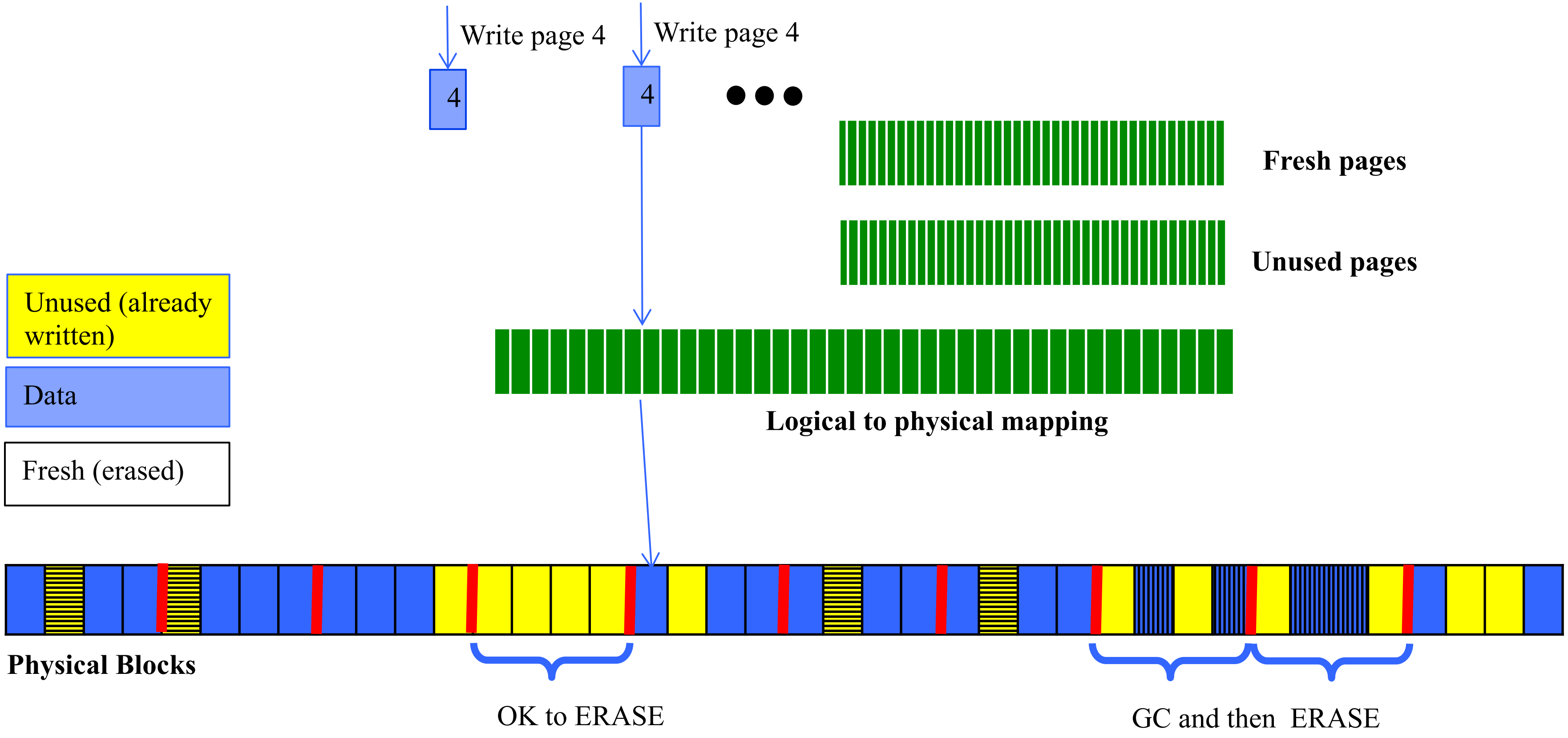
Fresh (erased)

Physical Blocks

Garbage Collection

- So at a next write
- We need to erase a block and create space
- To erase a block we need to consolidate “holes” in a single erase block
- This consolidation process is a form of garbage collection
- Requires
  - Assume a few spare erase blocks
  - Read many pages from various erase blocks
  - Write these pages in fewer erase blocks
  - Update metadata (mappings) and worry about failures
- All this while performing an I/O write
  - High overhead
  - Unpredictable

Create “fresh” pages

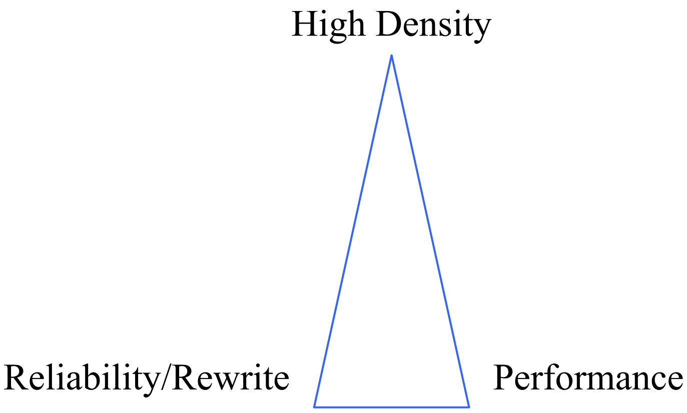


Today

- Perform garbage collection asynchronously
- Reduce overhead of GC
  - Eg segment SSD space between read-mostly and write-mostly data
  - Read-mostly will not be updated so erase blocks do not have “holes”
  - Write-mostly are updated frequently creating many “holes” => empty erase blocks (ready to be erased)
- Generally, today
  - a complex process
  - high overhead
  - still unpredictable behavior

Go back one step

- Can we avoid this problem?
- (1) pages / erase block (# metal contacts) – hurts performance
  - E.g. Erase block size = Page size
  - Erase block size depends on technology and affects density
  - Density critical – probably THE driving force behind market size
- (2) bits / cell (multi-level) – hurts reliability
- (3) page size not as fundamental – important for parallelism
  - No parallelism => expensive read/program
- Maybe there is room for different configurations



Wear Leveling

- Each block can be erased a number of times only
  - After that not possible to update/read
- If we are unlucky (and naive)
  - One block gets picked all the time to erase
  - After a number of erase cycles it goes bad
  - The SSD needs to reduce size
  - The filesystem (usually) does not handle this
  - Device is not usable
- What can we do
  - (1) Replace blocks
  - (2) Spread erases across blocks

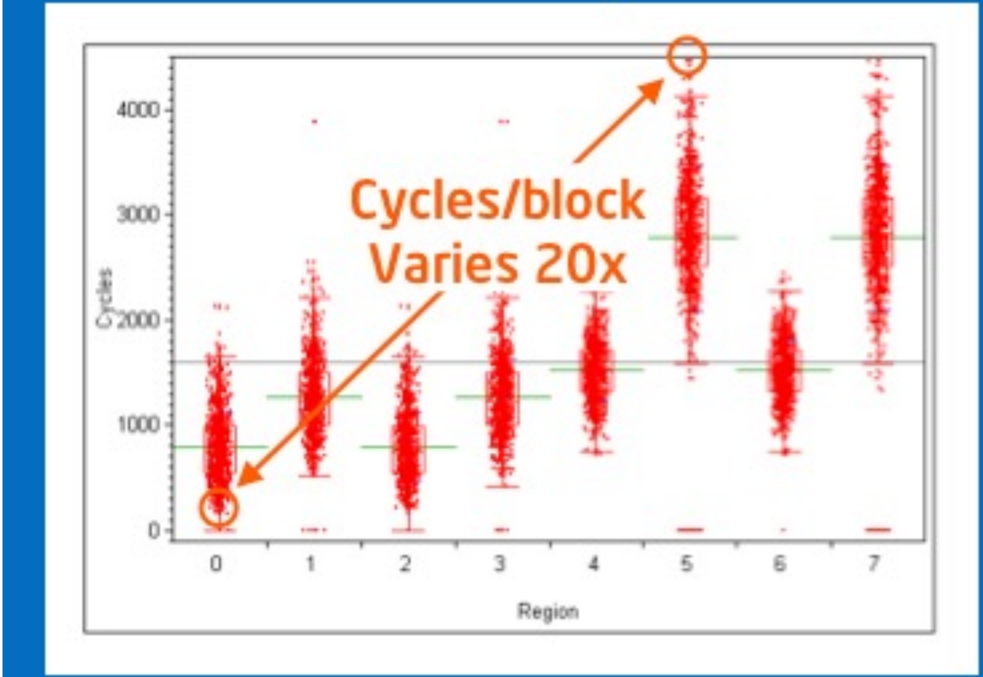
Replace blocks

- No matter what, blocks will go bad at some point
- SSDs have some spare blocks internally (non-negligible percentage)
- When a block goes bad, the SSD detects this
- It remaps logical block to a new physical block – SSD size remains the same
- This is straight-forward (not a common path operation)

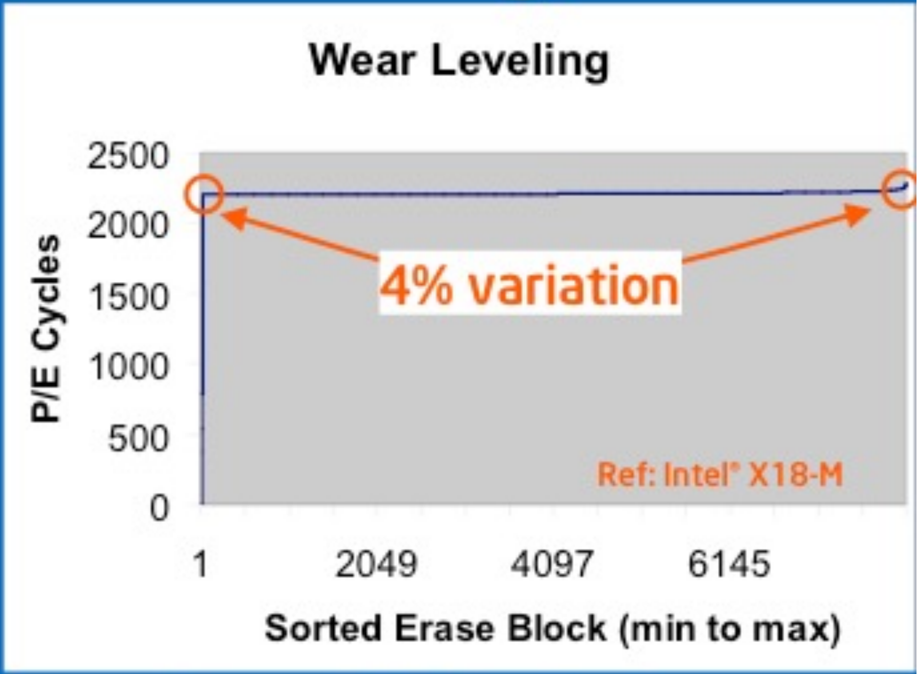
Spread erase operations

- Ensure that each block is erased about the same number of times
  - “Wear-leveling”
- Each block has metadata on how many times it has been erased
- When selecting a block to erase
  - (we said) consult the “fullness” of the block – less full block means less work for GC
  - In addition now consider the erase count
- Elaborate policies
  - E.g. segmentation of space for reads/writes does not help wear leveling

Variability in Wear Leveling



Unsophisticated regioned scheme



More sophisticated scheme

Controllers vary in in wear-leveling effectiveness

Poor wear leveling can have high impact

20x in cycles can be 10x or more in RBER

10x in RBER is 10^{ECC+1} in ECC failure rate: 100,000x for 4-bit ECC

Putting it together: SSD Reliability Metrics

SSD UBER values can be << 10-15

UBER $\propto$ usage: program/erase/read & subsequent retention

Intel® X18-M and X25-M Mainstream SATA SSD (80GB)

- 10 Channels Architecture with 50nm MLC ONFI 1.0 NAND
- 5 years usage, 1000G, 1.2million hrs MTBF
- GB/day client workload @ 1e-15 UBER → >>100GB/day, 5 years

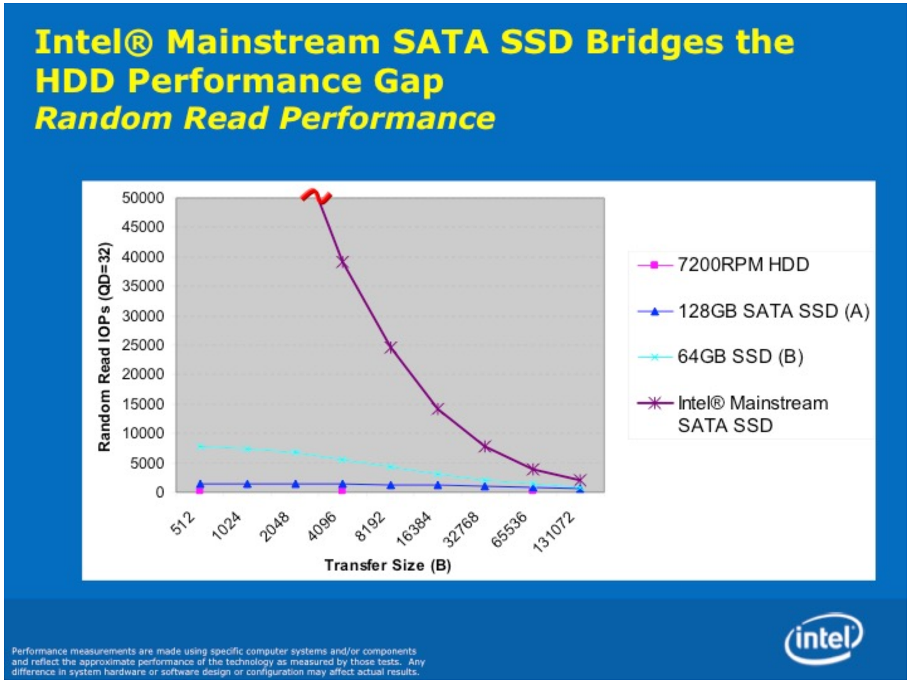
Intel® X25-M and X18-M Mainstream SATA SSDs deliver >5X accepted requirement for clients (20GB/day)

Intel® X25-E Extreme SATA SSD (32GB)

- 10 Channels Architecture with 50nm SLC ONFI 1.0 NAND
- 1000G, 2Million hrs MTBF
- Intel SLC SSD support > 7000 8K 2:1 R/W Random IOPs 24/7, 5 years

Intel X25-E SLC SSDs support the endurance required to replace many 15K RPM HDDs for IOPS applications

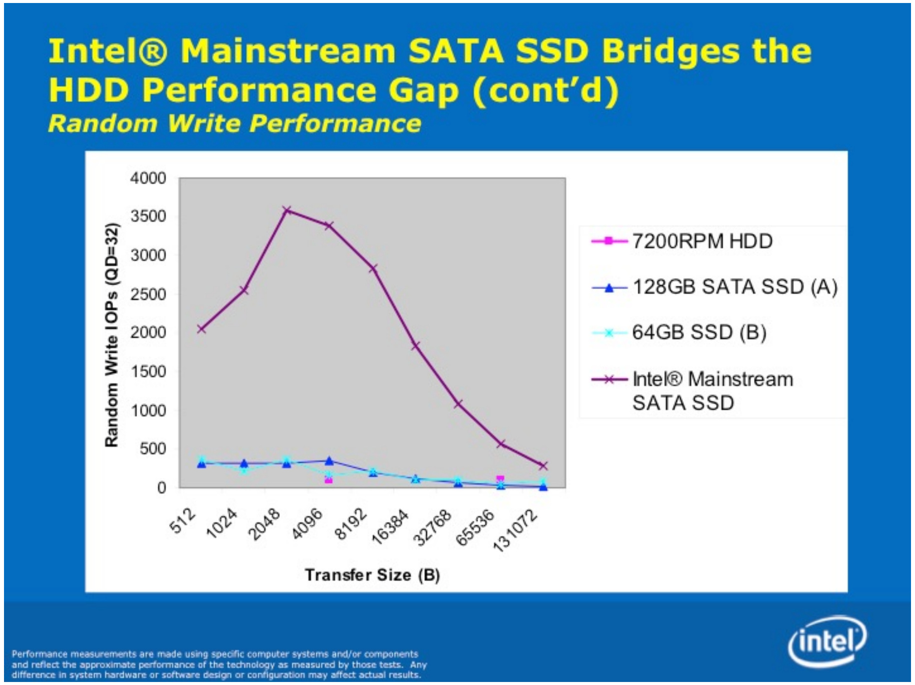
SSD Performance



Intel® Mainstream SATA SSD Bridges the HDD Performance Gap

Random Read Performance

Performance measurements are made using specific computer systems and/or components and reflect the approximate performance of the technology as measured by those tests. Any difference in system hardware or software design or configuration may affect actual results.



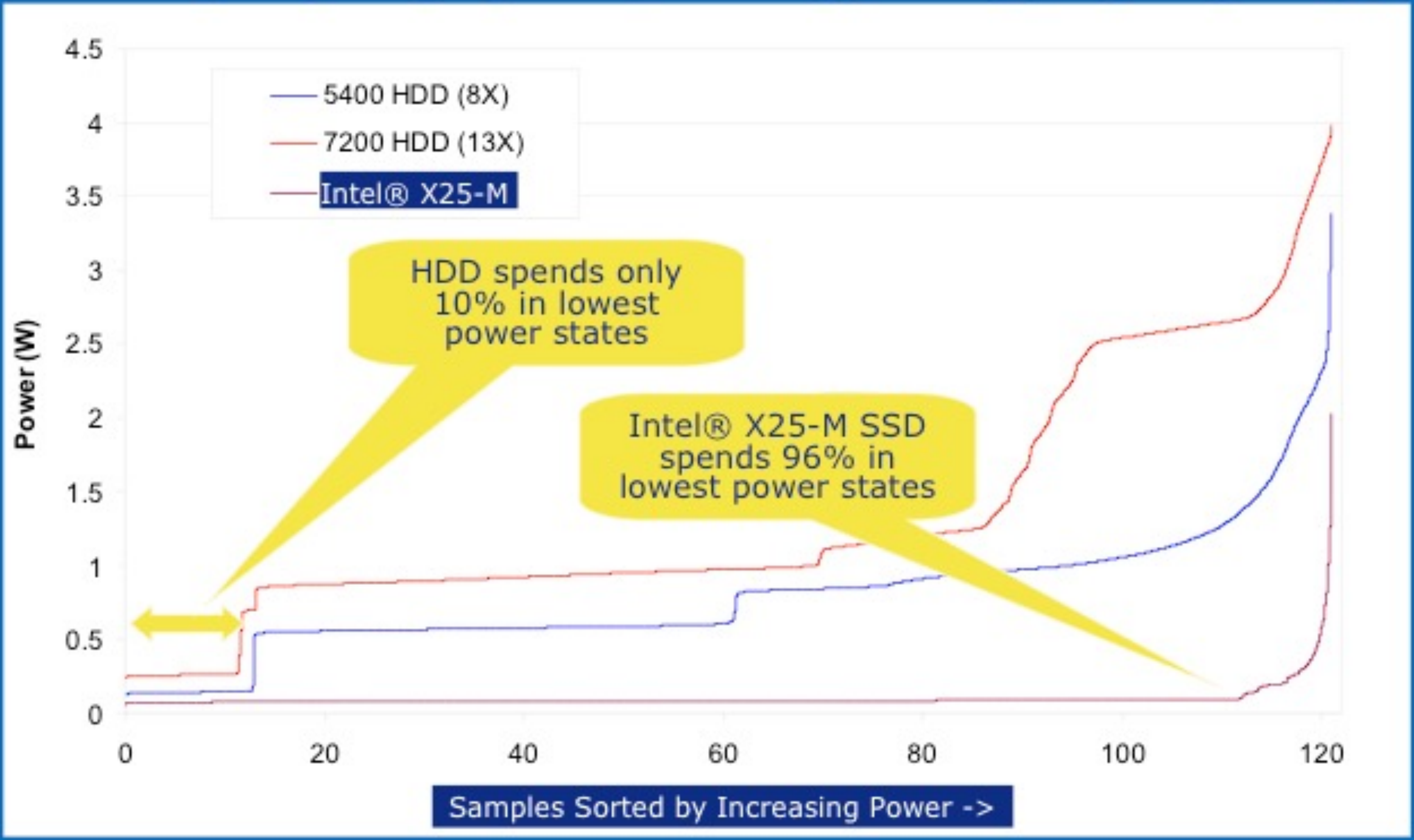
Intel® Mainstream SATA SSD Bridges the HDD Performance Gap (cont'd)

Random Write Performance

Performance measurements are made using specific computer systems and/or components and reflect the approximate performance of the technology as measured by those tests. Any difference in system hardware or software design or configuration may affect actual results.

- BUT performance variation a big problem, even today
- More work needed
- SSD controllers and host side become more and more complex

Intel® Mainstream SATA SSDs Save Power: SATA Power Rails With 2 Hour Mobile Workload



HDD spends only 10% in lowest power states

Intel® X25-M SSD spends 96% in lowest power states

Samples Sorted by Increasing Power ->

Performance measurements are made using specific computer systems and/or components and reflect the approximate performance of the technology as measured by those tests. Any difference in system hardware or software design or configuration may affect actual results.

What makes a good SSD

- SSD performance not always superior to hard disks (HDDs)
  - Request size matters
  - Request type matters
  - Access pattern matters
  - A HDD raid can be faster than an SSD at the same price
- A good SSD
  - I/O concurrency
  - Garbage Collection
  - Wear leveling
- Still room for improvement

Outline

- NAND Flash
  - How it works
  - Structure
  - Errors
- SSDs
  - High concurrency
  - Efficient garbage collection
  - Efficient wear leveling
- System-level SSD Storage
  - SSD caching
- Remarks
  - Non-volatile memories (NVM)

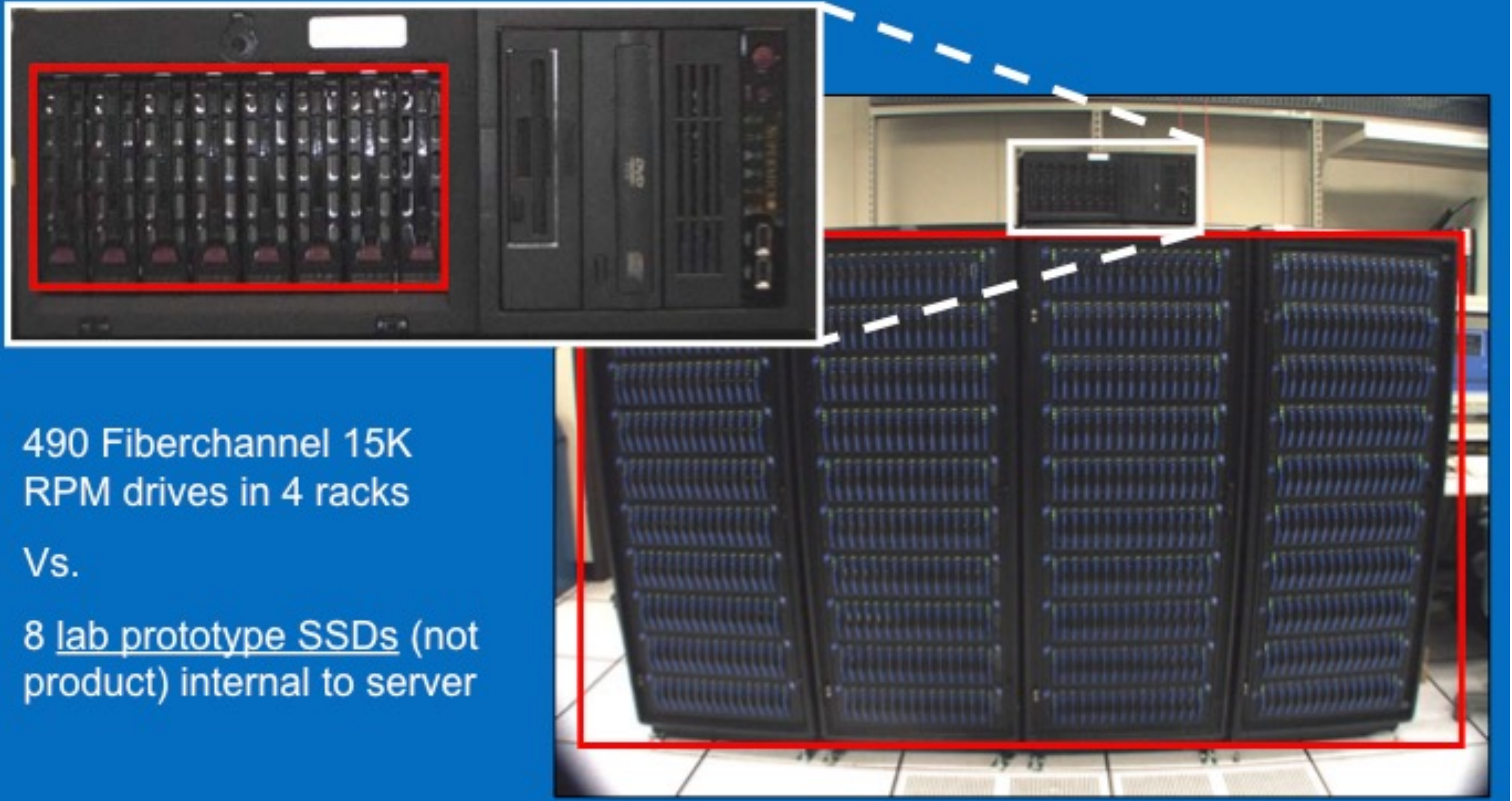
Background

- Increased need for high-performance storage I/O
    1. Larger file-set sizes ⇒ more I/O time
    2. Server virtualization and consolidation ⇒ more I/O pressure
- SSDs can mitigate I/O penalties

	SSD	HDD
Throughput (R/W) (MB/s)	277/202	100/90
Response time (ms)	0.17	12.6
IOPS (R/W)	30,000/3,500	150/150
Price/capacity ($/GB)	$3	$0.3
Capacity per device	32 – 120 GB	Up to 3TB

- Mixed SSD and HDD environments are necessary
- Cost-effectiveness: deploy SSDs as HDDs caches

Similar I/O Performance For IOPS Intensive Workload



490 Fiberchannel 15K RPM drives in 4 racks

Vs.

8 lab prototype SSDs (not product) internal to server

Unlikely to happen…

- Flash will not replace disks
  - Disk lowest cost/Gbyte
  - Trends not clear – not necessarily in favor of Flash
  - Manufacturing capacity for disks high
  - Would require replacing disk manufacturing capacity with Flash manufacturing capacity – not viable economically
- More likely
  - Integrate both SSDs and HDDs in storage system
  - E.g. SSD caching

Outline

- NAND Flash
  - How it works
  - Structure
  - Errors
- SSDs
  - Structure
  - Garbage collection
  - Wear leveling
- System-level SSD Storage
  - SSD caching
- Remarks
  - Non-volatile memories (NVM)

Non-volatile memory (NVM)

- NAND Flash is a type of NVM
- But
  - Still block addressable (similar to HDDs)
  - Accessible via the same interfaces as HDDs
  - Recent NVMe is FLASH on PCI
- Other types of NVM are emerging
  - NVM refers to byte-addressable technologies
- Most prominent
  - Phase change memory (PCM)
  - Magnetic RAM, variant Spin Transfer Torque (STT-RAM)
- Not clear what will happen, but something will happen

Situation 1: More I/O throughput

- Consider a server in the (near) future
  - 1000 cores
  - 1 Gbyte/s/core memory throughput => 1 Tbyte/s
  - 1 Gbit/s/core I/O throughput => 1 Tbit/s
- Assume all I/O throughput is eventually storage
  - This implies 10,000 spindles/server (assuming 100 MBytes/s/spindle)
  - Or 100,000 spindles/server (assuming 10 MBytes/s/spindle)
- SSDs/NVMe/NVM is the only way to achieve this compute density
  - Memory-level throughput for persistent storage

Situation 2: Less I/O throughput

- Alternative:
- Let’s reduce required I/O by keeping more stuff (data) in memory
- DRAM is very energy hungry
  - Need to refresh all the time, even if not accessing data
  - Some claim up to 50% of total energy in data-centre
- Fast persistent storage, and especially NVM (byte-addressable) is currently our best way to address this
  - Practically 0 idle energy consumption

The future memory hierarchy

- One memory to rule them all - Unlikely
- NVM to replace DRAM
  - DRAM is faster for writes
  - All NVM memories need to change state in a “permanent” manner for writes and this costs
- NVM/SSD to replace magnetic disks
  - Disk has high areal density which makes it low cost
  - Manufacturing capacity for NVM not adequate to replace disks
- Eventually, complex memory hierarchy
  - DRAM ⇔ NVM ⇔ Flash ⇔ HDDs

How?

- Nobody knows yet ☺
- Not an easy problem
  - SSDs have been around for almost 10 years now
  - Still not well integrated in storage hierarchy
- NVM will have issues at multiple layers
- Devices and packaging
  - What (write) update time will each technology allow?
- Architectural level
  - What to place in DRAM and what in NVM?
- Systems software level
  - Can we afford the overhead of the traditional I/O stack?
- Application level
  - Can we reduce recovery overheads?
- There is potential for significant impact

Questions?

Angelos Bilas
FORTH-ICS
and University of Crete, Greece
bilas@ics.forth.gr