Critical Role of Firmware and Flash Translation Layers in Solid State Drive Design

Robert Sykes

Director of Applications

OCZ Technology

Introduction

- This talk will describe the requirements of the design of an SSD controller and the firmware that implements its Flash Translation Layer (FTL). Data integrity, longevity and performance must be maintained while dealing with host interaction (AHCI SATA and Operating System), garbage collection and wear levelling.

Contents

- Flash Translation Layer (FTL)
- Operating system and associated SATA drivers.
- Wear levelling and Garbage collection
- Performance and Robustness

Contents

- Flash Translation Layer (FTL)
- Operating system and associated SATA drivers.
- Wear levelling and Garbage collection
- Performance and Robustness

Contents

- Flash Translation Layer (FTL)
- Operating system and associated SATA drivers.
- Wear levelling and Garbage collection
- Performance and Robustness

Contents

- Flash Translation Layer (FTL)
- Operating system and associated SATA drivers.
- Wear levelling and Garbage collection
- Performance and Robustness

Flash Translation Layer

FTL

- What is it, and why do we need a Flash Translation Layer (FTL)?

  - SSD’s require a mapping between the LBA and physical media
    - In order to write to an area in the physical media it must be erased on a block basis.
    - Writes to the same LBA will be mapped to different physical locations on the flash media.

  - SSDs need to move data around.
    - To ensure that the SSD is evenly worn to prolong the life of the SSD (Wear Leveling) or Data Stirring.
    - Reclaim blocks previously deleted by the OS so that a new write will not have to do a read / modify / write therefore ensure peak performance is maintained (Garbage Collection).
    - Bad Block Management to handle invalid blocks.
    - Error correction

  - The FTL is responsible for the logical to physical mapping of data!

FTL

- How does it work?
  - Host PC’s expect to access a HDD (LBA device).
    - HDD’s are constructed of sectors.
      - This is usually 512B or 4KB (usually emulated back to 512B)
    - SSD’s are constructed of planes, blocks and pages.
    - FTL therefore translates a sector access into a page or block access.
      - Metadata is used to map the logical to physical address.

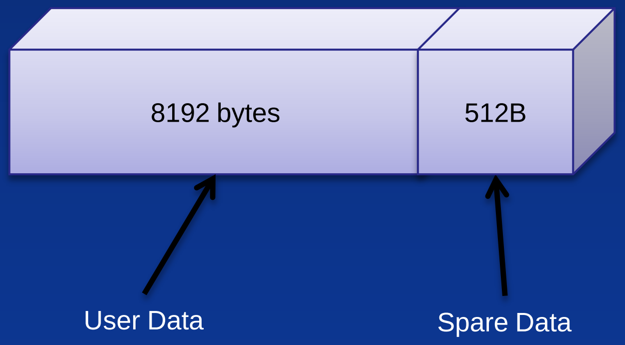
If only it were that simple !

FTL

- HDD and Flash Basics:
  - HDD vs SSD configuration.
    - HDD
      - Made up of sectors.
        - Sector is 512B (maybe 4KB with 512B emulation)
    - SSD
      - 1 Page = (8 KB + 512B)
      - 1 Block = (8 KB + 512B) * 128 pages (Note: block may contain up to 256 pages)
        - = (1 MB + 64 KB)
      - 1 Plane = (1 MB + 64 KB) * 4096 blocks
        - = (4 GB + 256 MB)
      - 1 Die = (4 GB + 256 MB) * 2 Planes
        - = (8 GB + 512 MB
      - 1 Bank = (8 GB + 512 MB) * 2 dies
        - = (16 GB + 1 GB)
      - 1 Device = (16 GB + 1 GB) * 64 banks
        - = (1TB + 64GB)
    - Windows file system Reads / Writes 4KB (i.e. 8 sectors) at a time.
      - If formatting, Windows uses 512B.

FTL

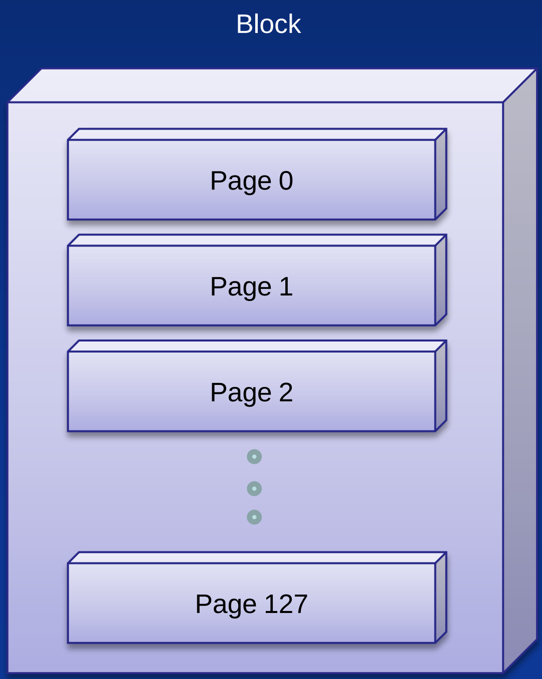
- Pages
  - Smallest addressable unit for read and program operations
  - The ‘Spare Data’ is used for storing ECC parity data and marking the page as bad.



- Note: The size of the spare area is dependant on the device.

FTL

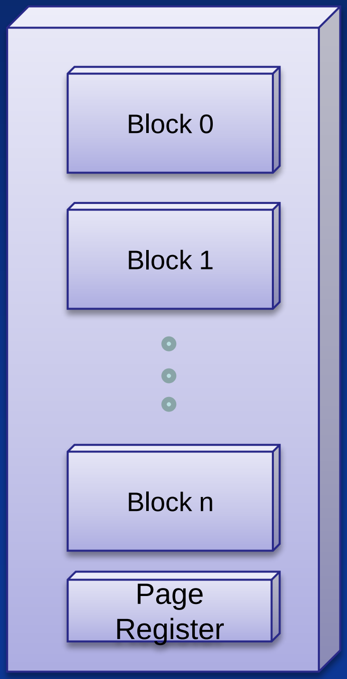
- Blocks
  - Smallest erasable unit of data
  - There are generally 128 or 256 pages per block.
  - Pages within a block must be written sequentially but can be read in any order



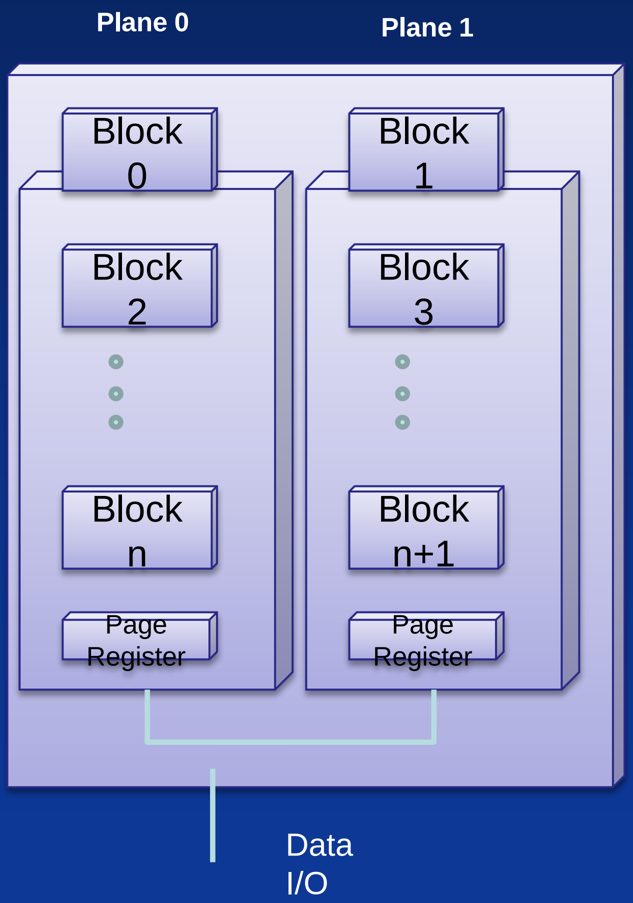
FTL

- Die
  - A die may contain up to 4 planes.
  - Planes have Page Registers and Cache Registers for shifting in and out data.
  - Registers hold data for Flash array operations (read/program)
  - Multiple planes, typically 2 or 4

Plane

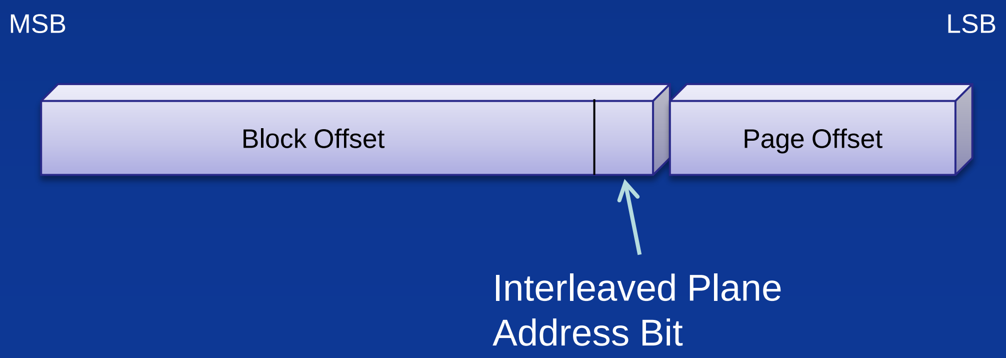


Die



FTL

- Page Addressing
  - Flash controllers use a Row and Column addressing scheme.
  - Column
    - 2 byte field identifying the byte offset within a page.
  - Row
    - 3 byte field identifying the page address.



FTL

- Back to FTL…..
  - Each host file system access to the device must write a minimum of 4Kbytes of data (Assuming Win7).
  - NAND can only write a page or simultaneous pages at a time (e.g. 8K * 2 = 16KB (25nm) and <20nm moves to 16K*2 = 32K
    - Of course this depends on device manufacturer, not all will migrate to larger pages.

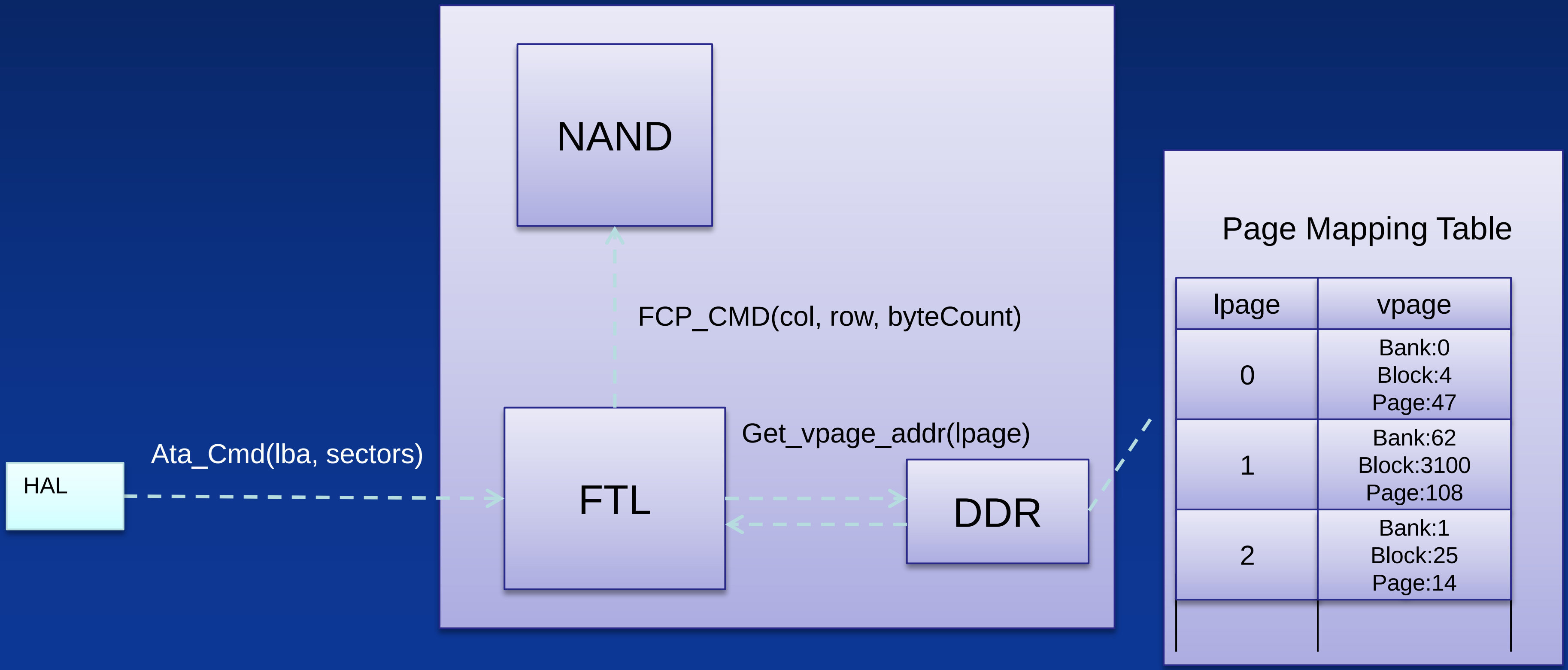
As we are now writing to a page and not a sector we must create a map of the address the host thinks its written to (based on sectors) and the address the NAND controller has actually written the data to.

FTL

- What factors will effect the FTL design?
  - Your FTL architecture should take into account the environment its operating in.
    - What size of commands will the FTL have to cope with.
      - Consider size of metadata?
    - Consider how to ensure system robustness.
      - FTL must cope with power failure
        - » FTL rebuild
      - FTL must cope with FLASH operations outside of host accesses.
        - » Garbage collection, BBM, IPM, PM.

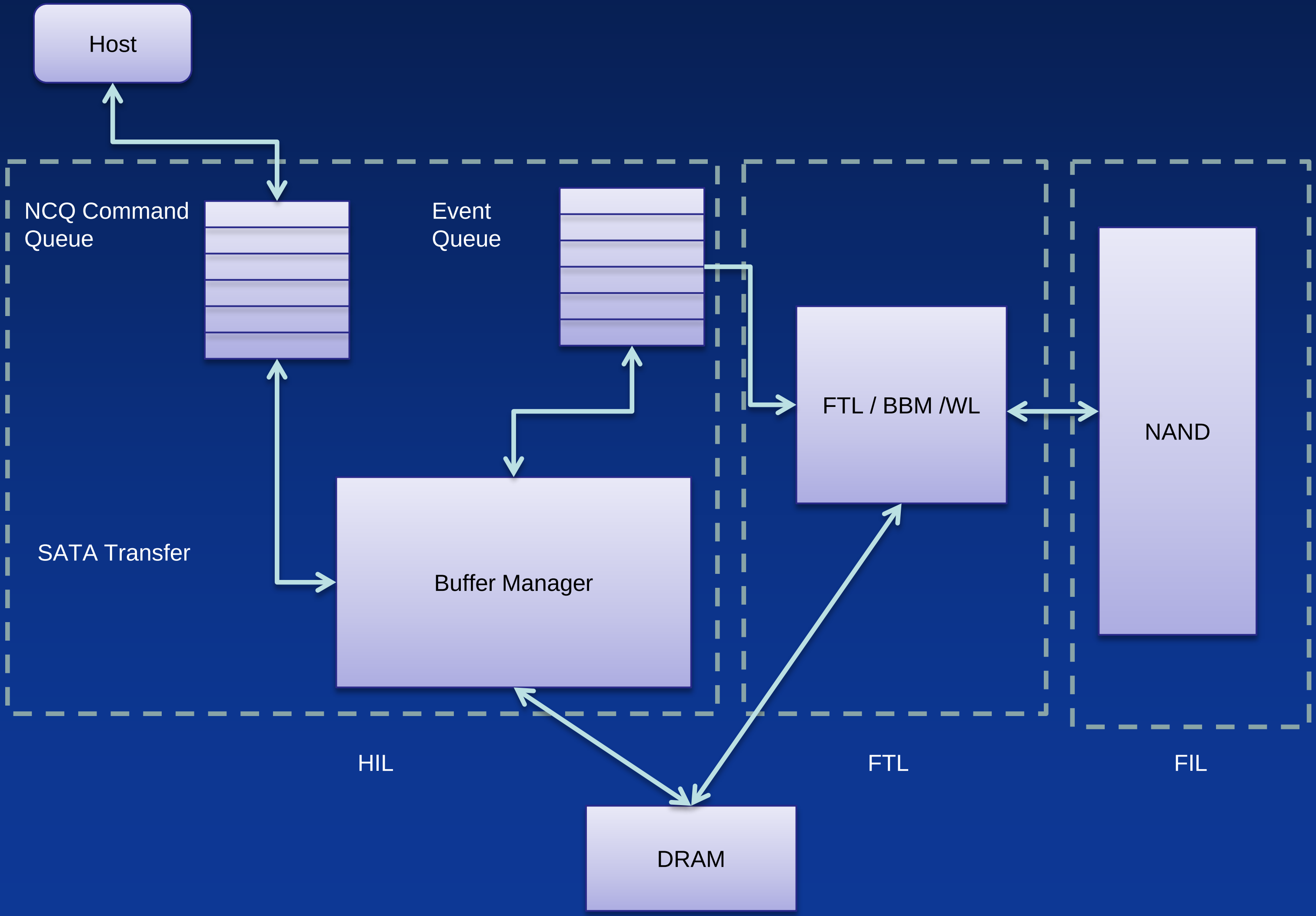
FTL

- Basic FTL Firmware architecture (Jasmine)

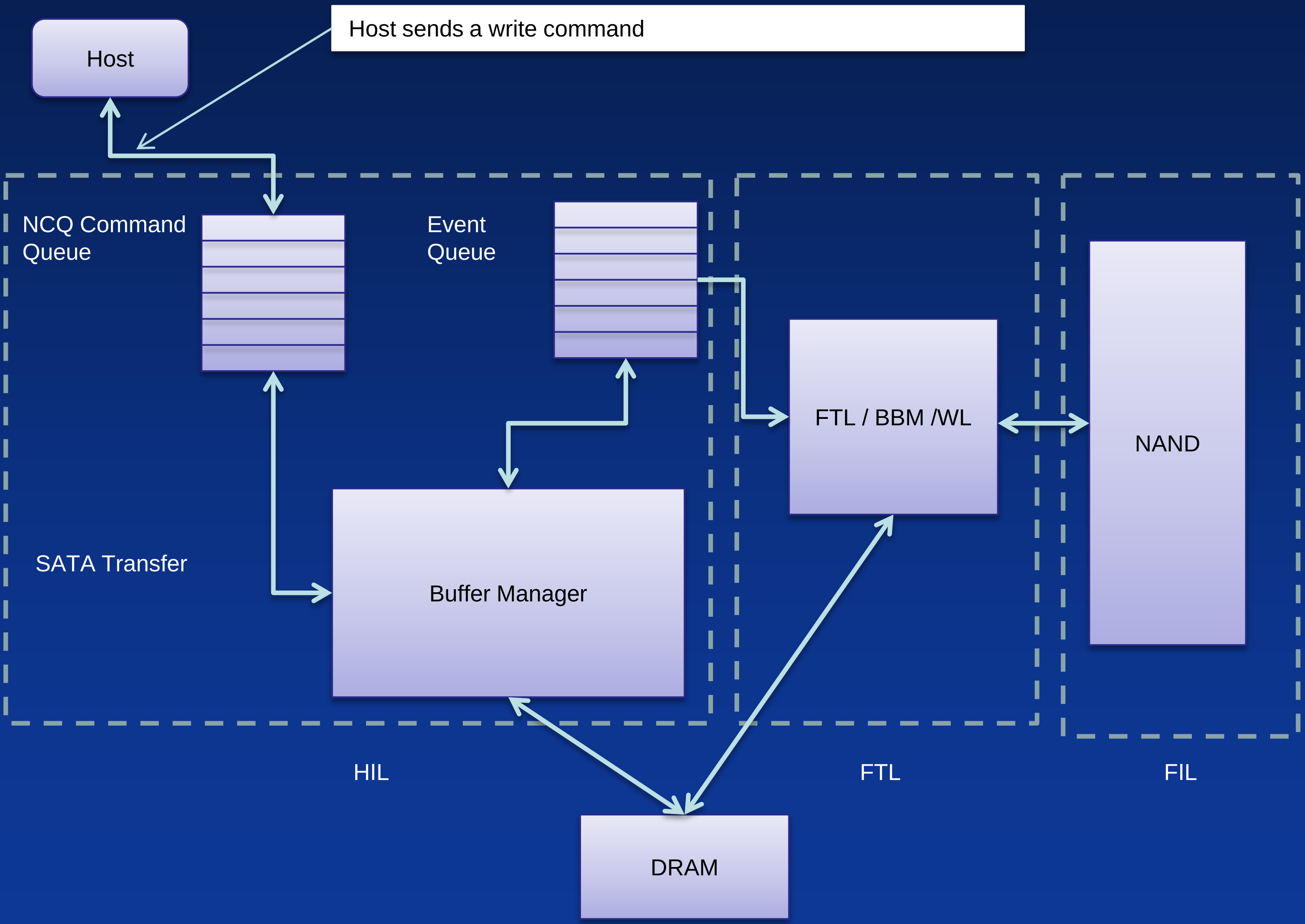


lpage	vpage
0	Bank:0 Block:4 Page:47
1	Bank:62 Block:3100 Page:108
2	Bank:1 Block:25 Page:14

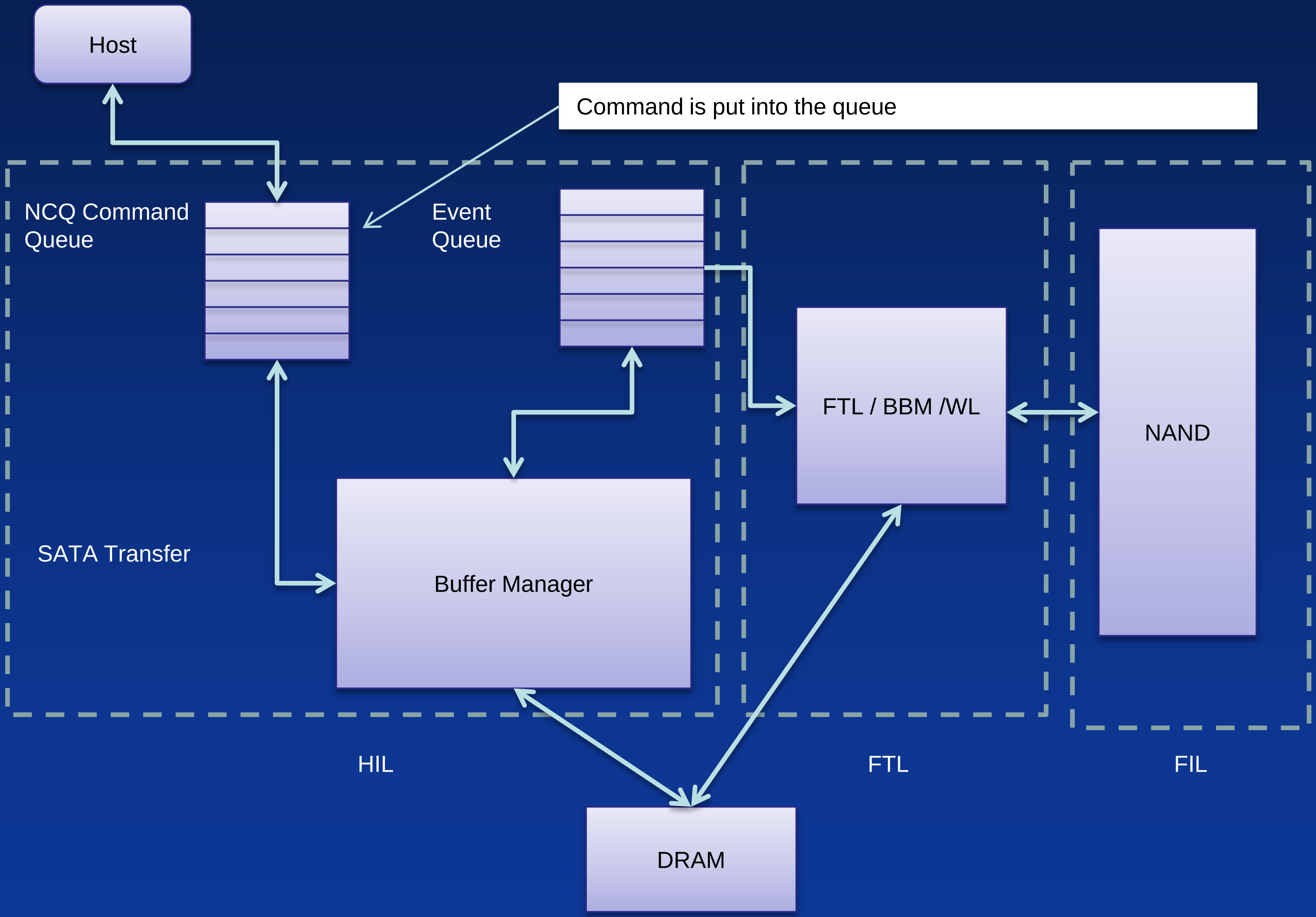
FTL


Host
NCQ Command Queue
Event Queue
FTL / BBM /WL
NAND
SATA Transfer
Buffer Manager
HIL
FTL
FIL
DRAM

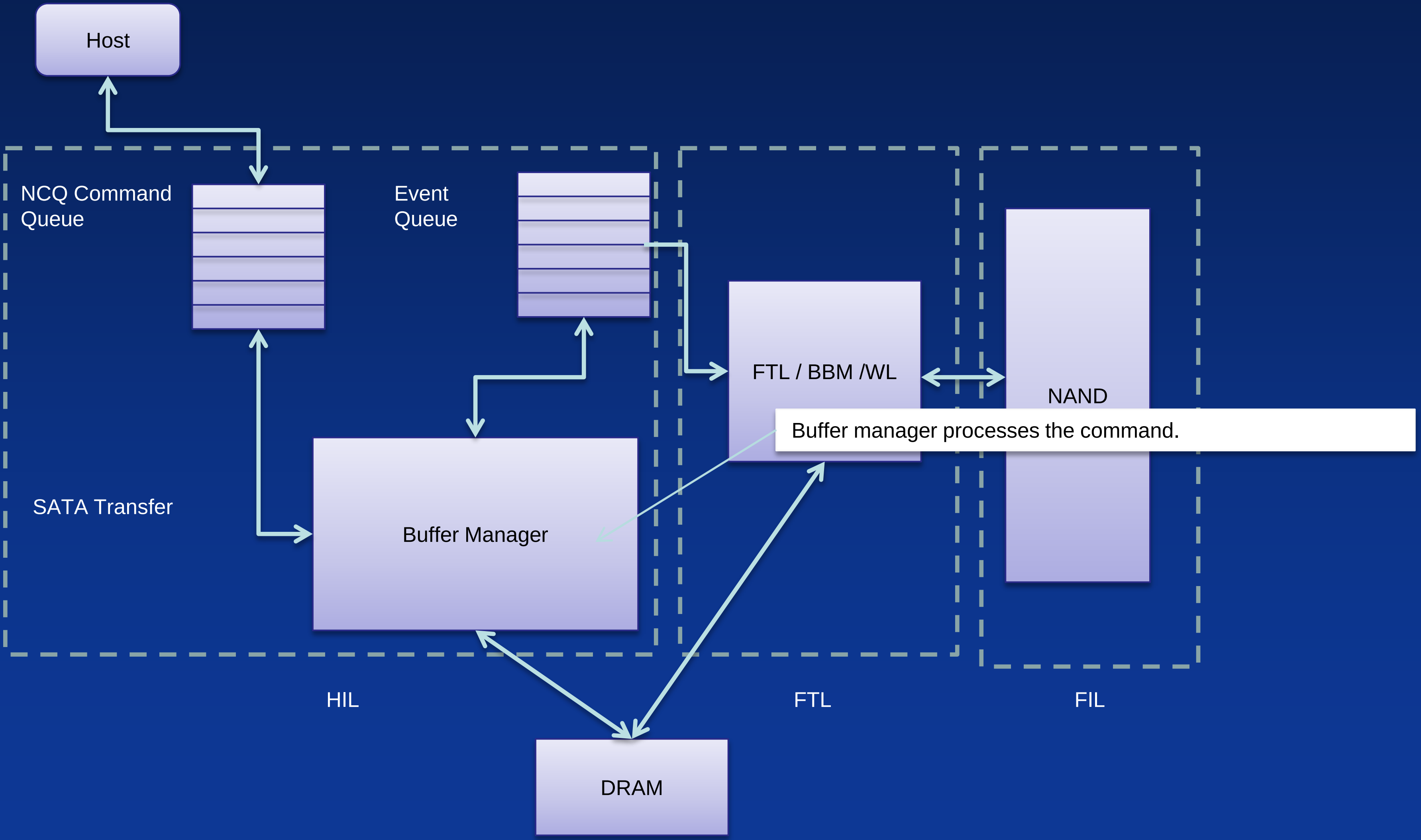
FTL


Host sends a write command
Host
NCQ Command Queue
Event Queue
SATA Transfer
Buffer Manager
FTL / BBM /WL
NAND
HIL
FTL
FIL
DRAM

FTL

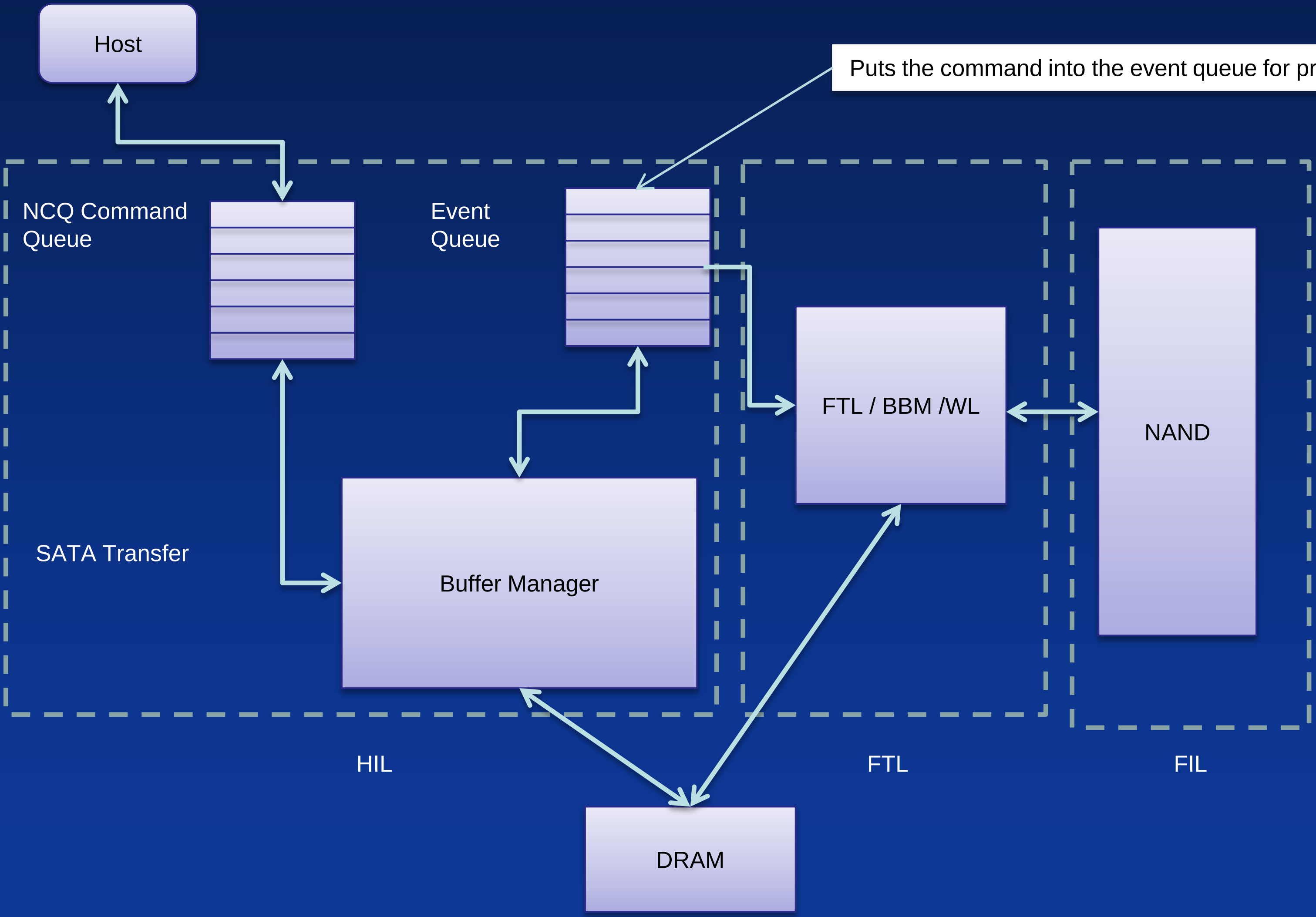

Host
Command is put into the queue
NCQ Command Queue
Event Queue
SATA Transfer
Buffer Manager
FTL / BBM /WL
NAND
HIL
FTL
FIL
DRAM

FTL



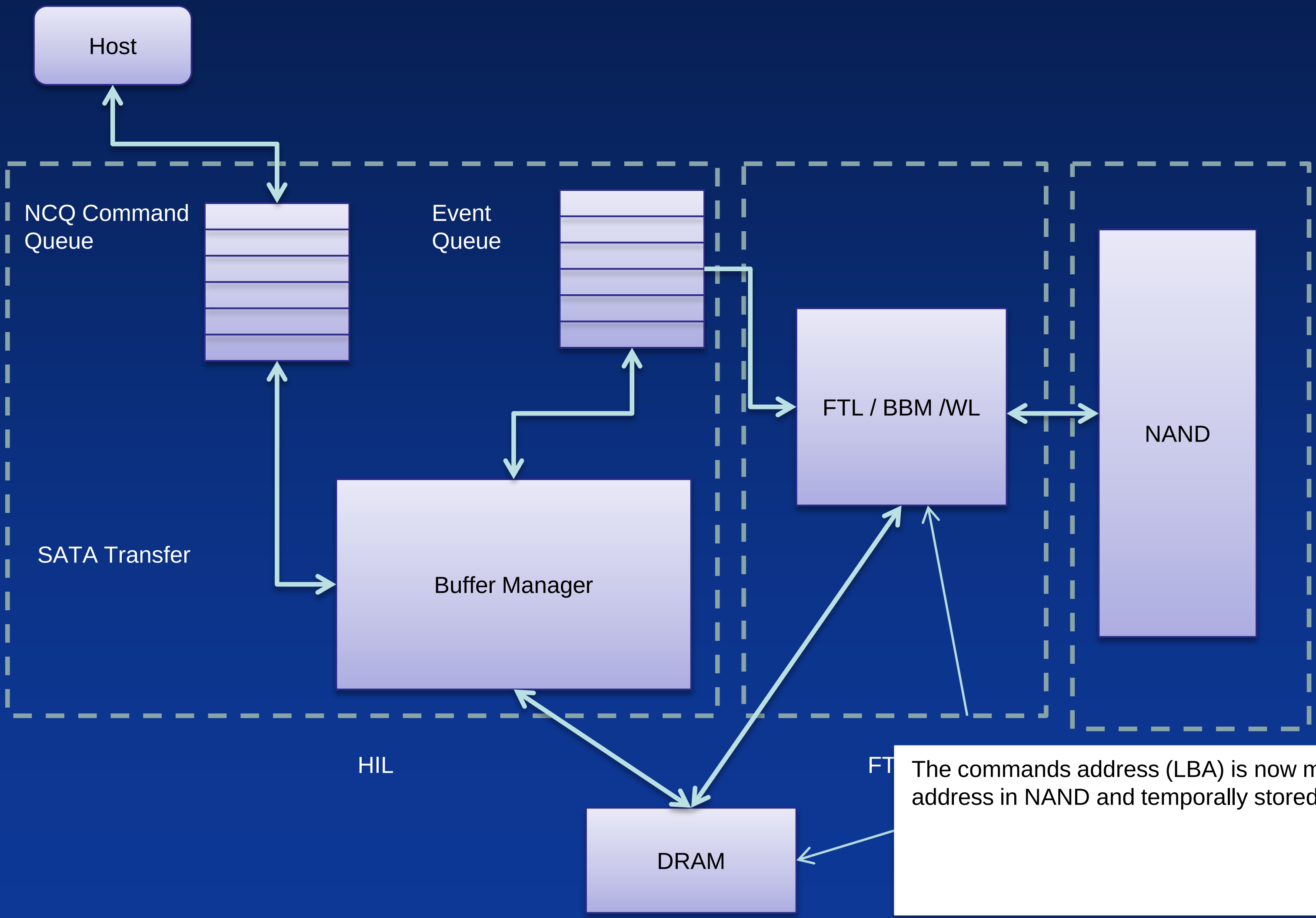
Buffer manager processes the command.

FTL



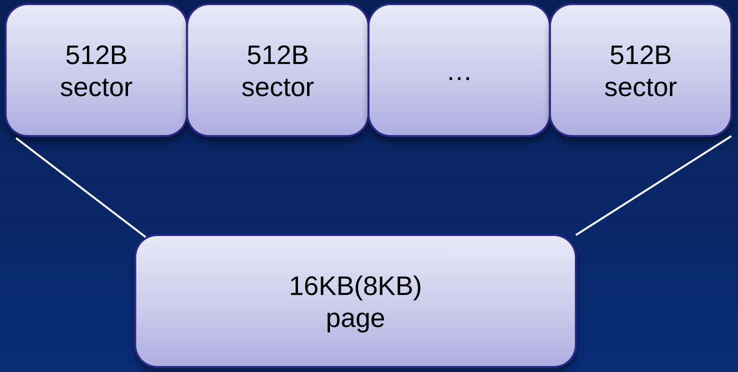
Puts the command into the event queue for processing.

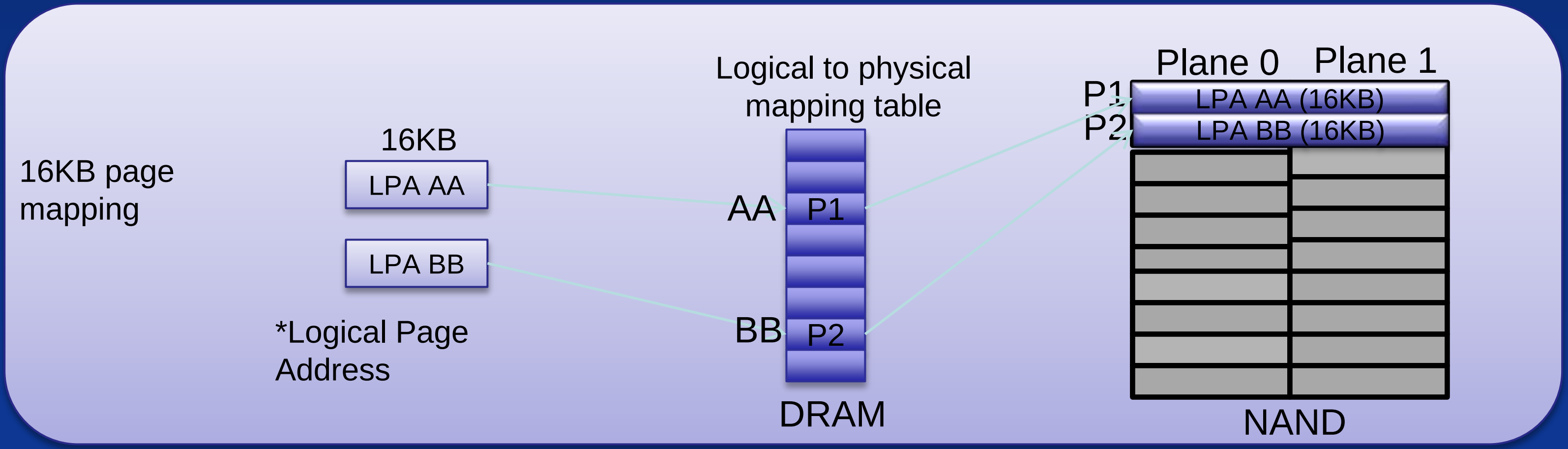
FTL



The commands address (LBA) is now mapped to the physical address in NAND and temporally stored in DRAM.

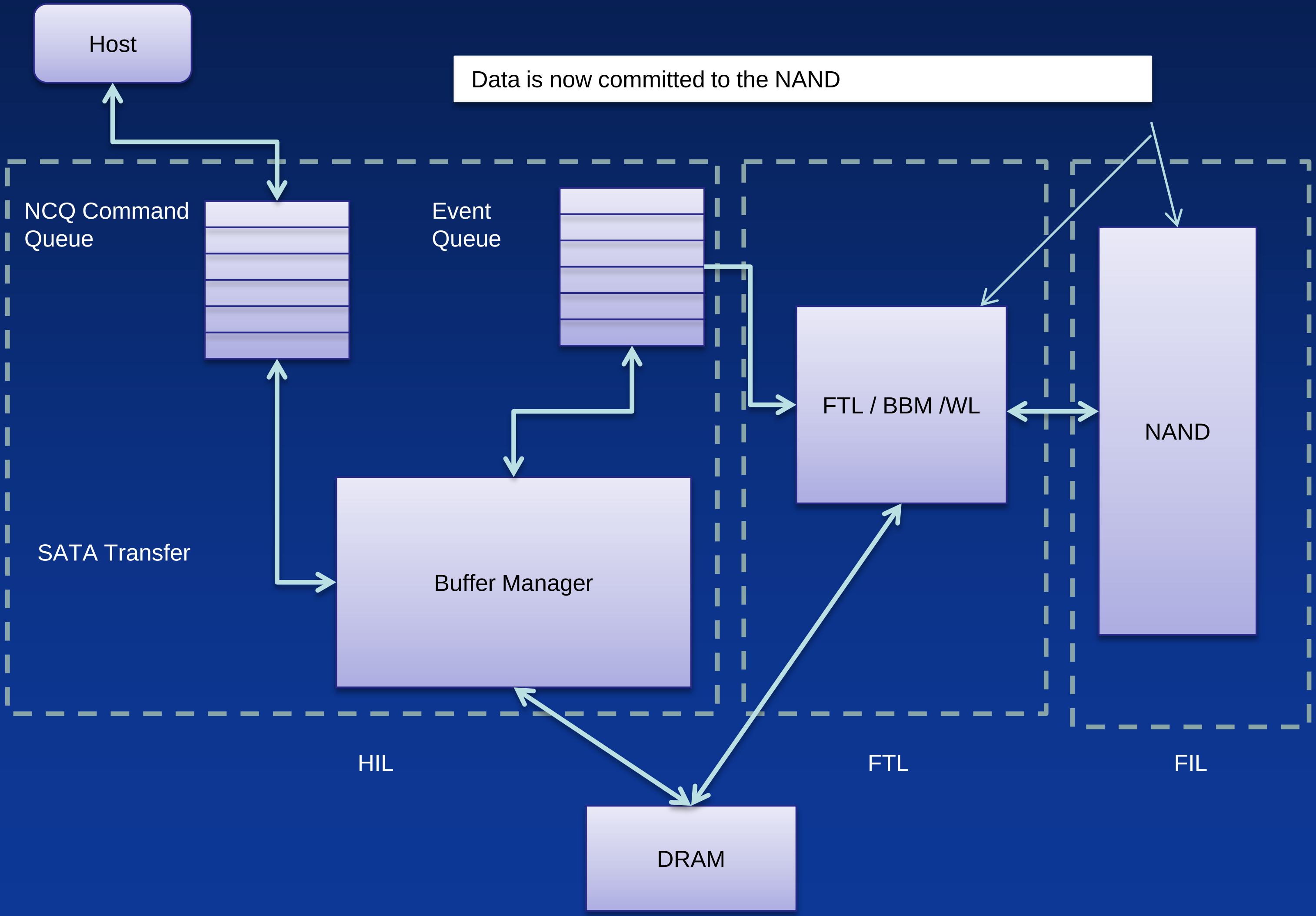
FTL


512B
sector
512B
sector
…
512B
sector
16KB(8KB)
page


16KB page
mapping
16KB
LPA AA
LPA BB
*Logical Page
Address
Logical to physical
mapping table
AA
P1
BB
P2
DRAM
Plane 0
Plane 1
P1
P2
LPA AA (16KB)
LPA BB (16KB)
NAND

FTL

Data is now committed to the NAND



FTL

- Well…… we didn’t really just commit the data to the NAND!
  - We actually did some wear leveling first.
  - We need to do this so that the data written to the NAND is spread across the NANDs so that we don’t inadvertently write to the same bank over and over and inadvertently wear out a bank relative to others.
  - The FTL shouldn’t just map logical to physical. It has to send data to different parts of the NAND and of course know where the data is at all times.

Firmware

- Not done quite yet!
  - We cant just write wear leveled data to the NAND.
  - To ensure the longevity of the NAND the data must first be scrambled (Scrambler is at the hardware level).
    - Scrambler can be either on NAND or part of the controller logic.
      - Scrambler is used to ensure data written has an even bit distribution.
        - » Retention errors
        - » Adjacent cells may have been disturbed from weak charge on floating gate.

Firmware

- Now we have committed data to the NAND we still have some work to do.
  - We need to write the updated metadata to the NAND.
    - If we don’t, and power is lost, the metadata mapping table is lost and thus the data written on the NAND will not be accessible to the OS.
    - We don’t want to write the complete metadata table at the end of each command. This would hamper performance. Better would be to write the change in metadata at the end of a command and commit the full table at predefined intervals.

FTL

- Factors effecting the FTL
  - Host interaction (AHCI SATA, Operating System and drivers)
  - Garbage collection and Wear Levelling
    - Significantly stress the SSD device, therefore the overall design of the SSD firmware and the FTL architecture play an import role in ensuring data integrity and the longevity of the device whilst still maintaining performance.
      - If data is moved around the NAND devices the FTL needs to track it in the metadata.

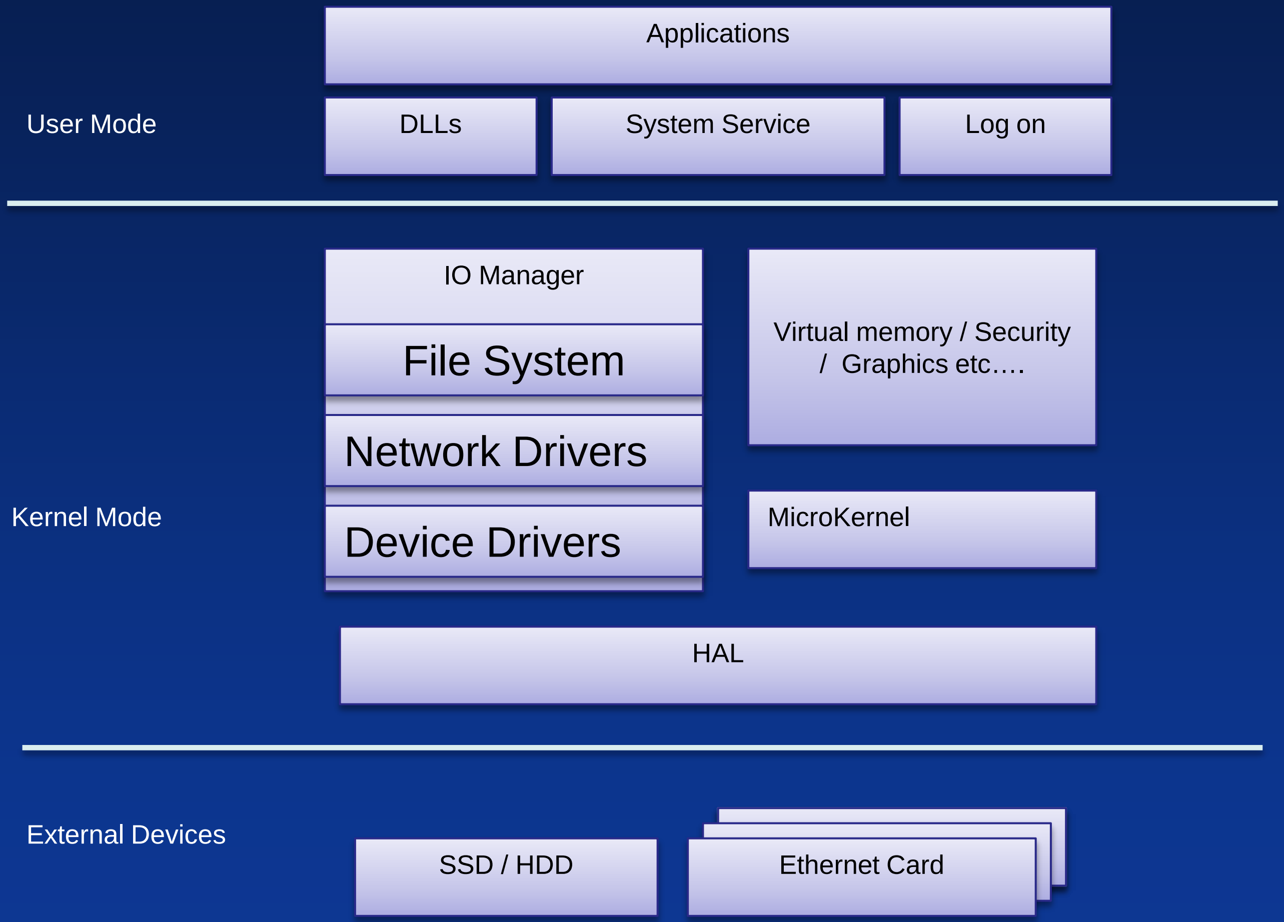
- Operating System and Associated Driver

OS & Driver

- First we need to consider the environment we are working in.
  - Laptop / Desktop environment.
    - The SSD makes up part of the system and most likely connects to one of the onboard host SATA ports.
    - Data across the SATA bus will be dependant on the OS file system and device driver.

OS & Driver

- Overview: Operating System



Applications

User Mode

DLLs

System Service

Log on

IO Manager

File System

Network Drivers

Device Drivers

Virtual memory / Security / Graphics etc.…

Kernel Mode

MicroKernel

HAL

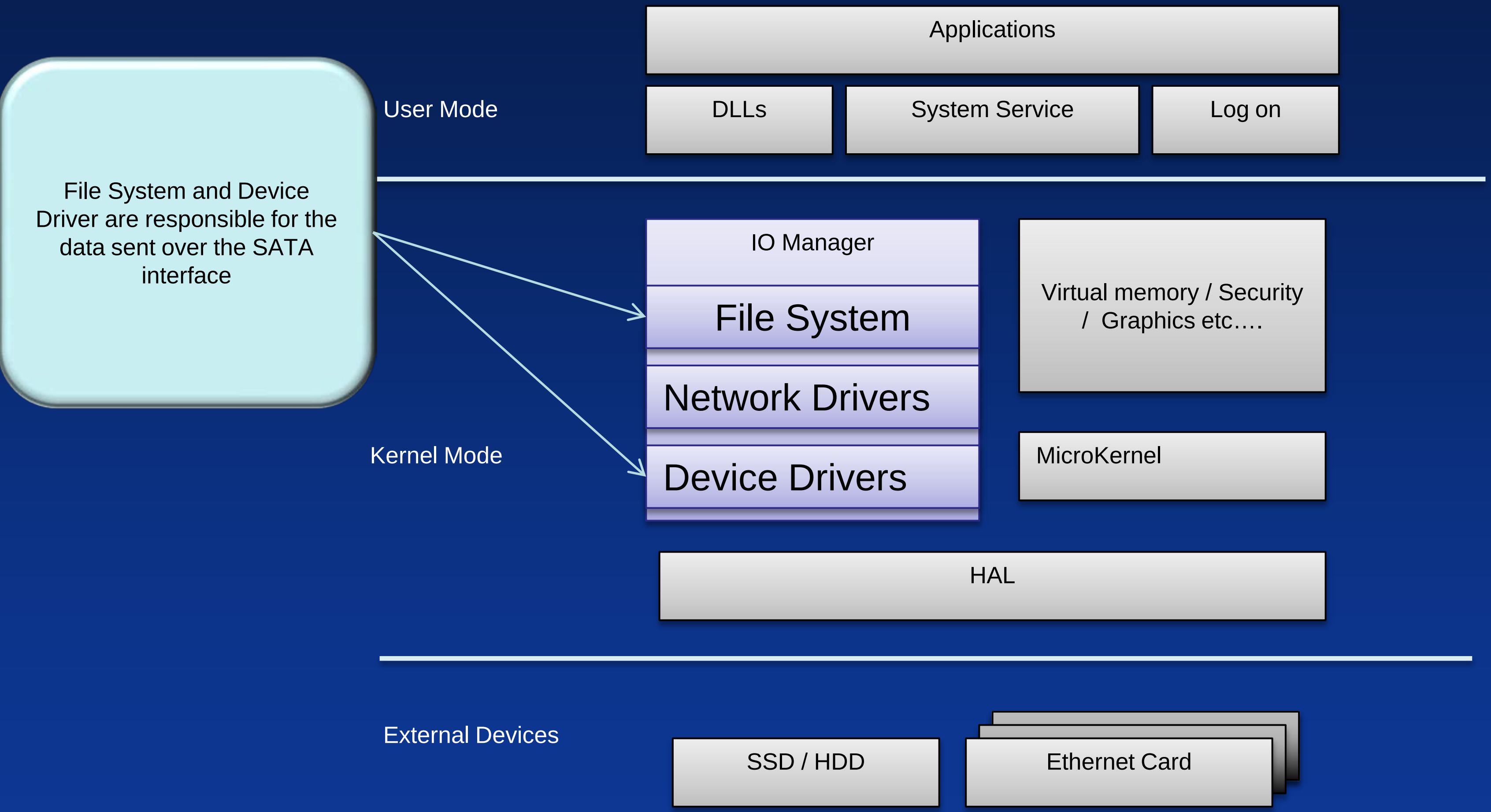
External Devices

SSD / HDD

Ethernet Card

OS & Driver

- Overview: Operating System



OS & Driver

- File System
  - Windows 7 file system smallest command is 4KB.
    - When formatting a disk 512B commands will be used.
  - Windows XP file system smallest command is 512B.
- Drivers
  - Different drivers will send different command sizes.
    - E.g. if the host is to send a continuous 10MB command, some drivers may break that down into 1MB chunks, others will use 128KB chunks.
    - All this has an impact on the FTL and thus the SSDs performance and reliability capability as the combination of hardware and software controlled buffers along with the FTL have to be capable of dealing with different file sizes.

OS & Driver

- Why is the OS and associated drivers important to consider when designing a SSD controller and its firmware?

  - Different command sizes impact how the firmware buffers and then sends data to the NAND. Most of the NAND today is either 16K (2*8K (2 plane mode)) or single 8K page (latest NANDs 32K) , we can therefore make decisions in firmware as to how we want to commit data to the NAND based on the commands size from the host and the page size we are writing to.
    - This will impact the overall performance of the system!

OS & Driver

- In the old days of NAND, each page was sector aligned (512B) so the process of writing to the NAND was significantly simplified. As NAND technology has progressed, NAND page sizes and total densities have increased and as a result so has the page size.

- Ideally we would have an FTL that was still sector aligned (512B) for best performance with small commands but this would cause problems with storing the metadata and would most likely require DDR at a size that would be cost prohibitive.
  - 512B page mapped FTL for a 128G disk would require 512MB DDR, 512GB disk would require 2G of DDR.

OS & Driver

- The Calculations:
  - If Page mapped FTL is used then:
    - For 16K page there would be one entry for each 16K
      - One entry is ~4B. I.e. for each 16K page written 4B of metadata must be stored in DDR.
      - For a 256GB drive with 16K page NAND:
        - » (256G / 16K) *4B = 64MB
      - For a 256GB drive with 8K page NAND:
        - » (256G / 8K) *4B = 128MB

OS & Driver

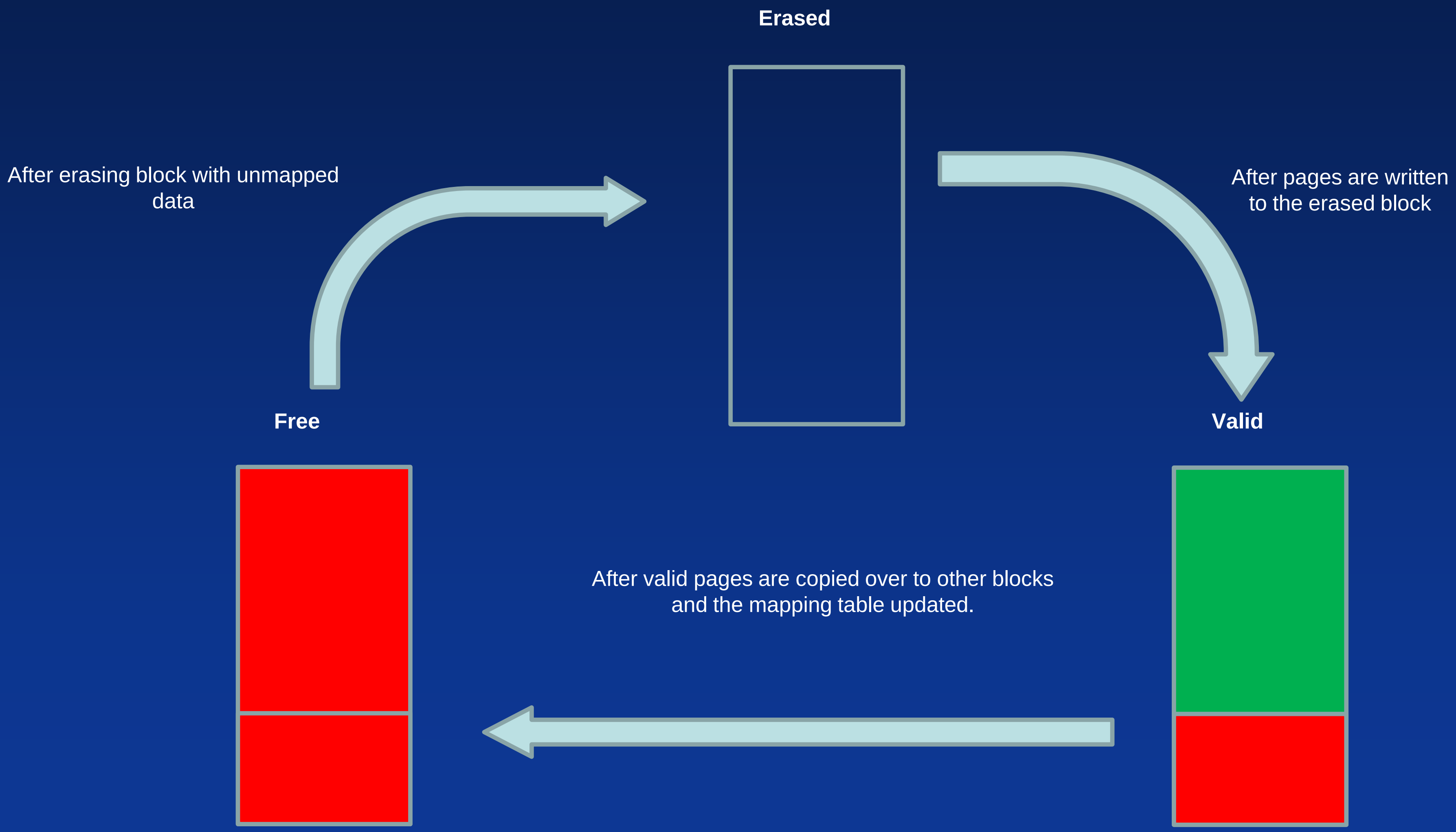
- Performance.
  - We now have some complex algorithms.
    - FTL
    - Garbage Collection
    - Wear Levelling
  - How is the best in class performance achieved.
    - With a very intimate knowledge of the system.

Garbage Collection

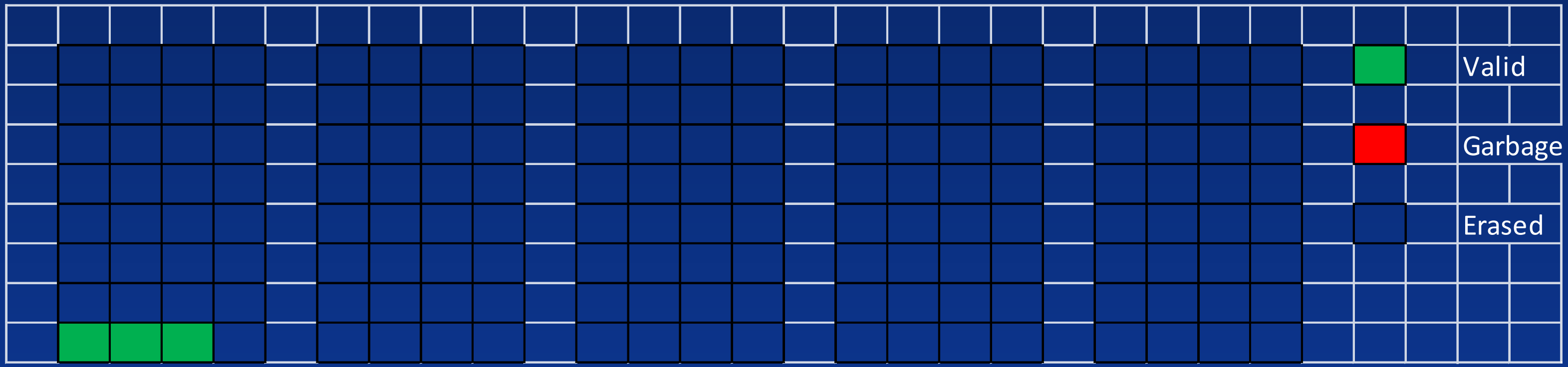
Garbage Collection

- What is Garbage Collection?
  - Garbage Collection is the process of freeing up partially filled blocks to make room for more data.

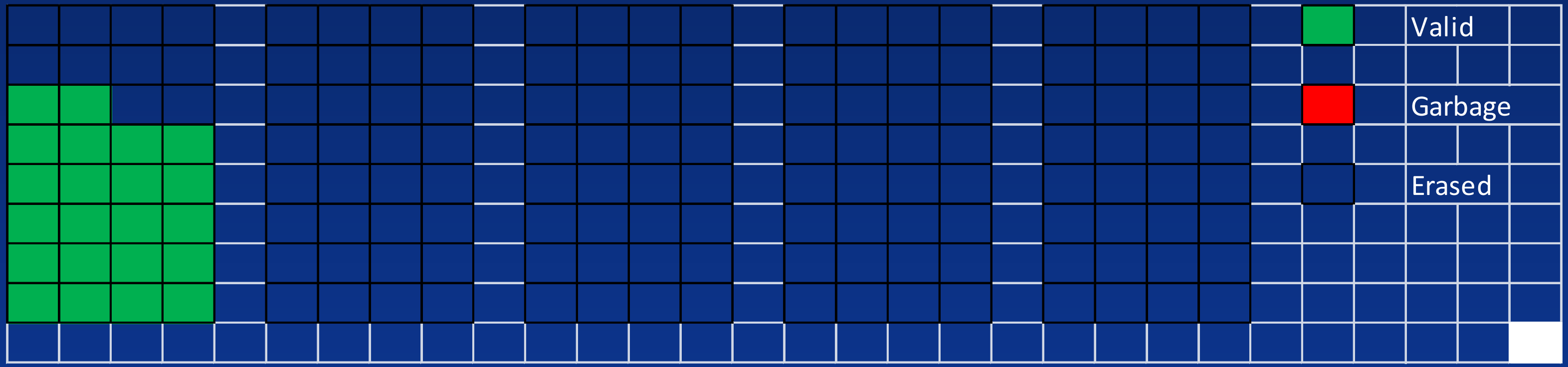
Garbage Collection


Erased
After pages are written to the erased block
Valid
After valid pages are copied over to other blocks and the mapping table updated.
Free
After erasing block with unmapped data

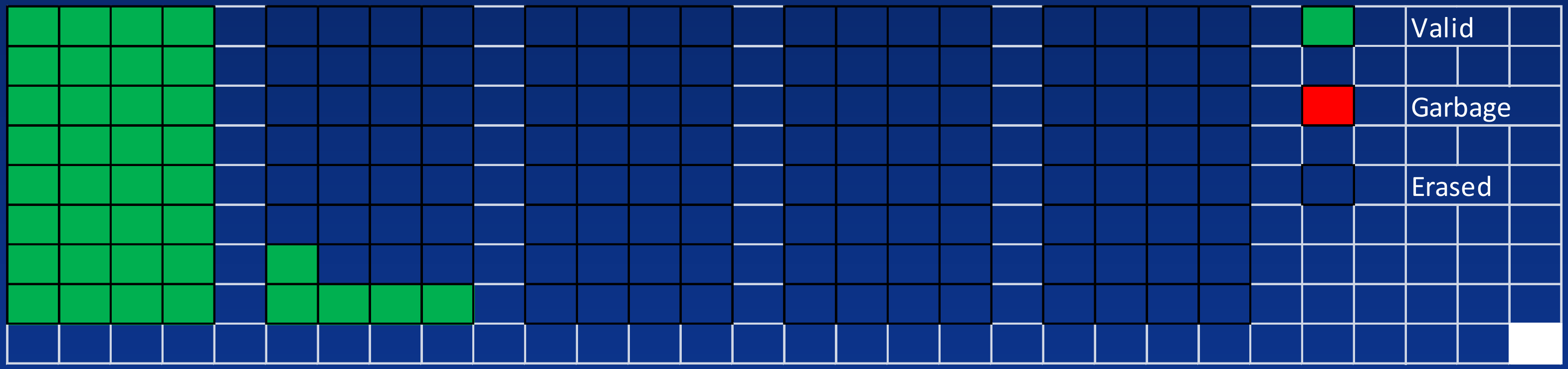
Garbage Collection



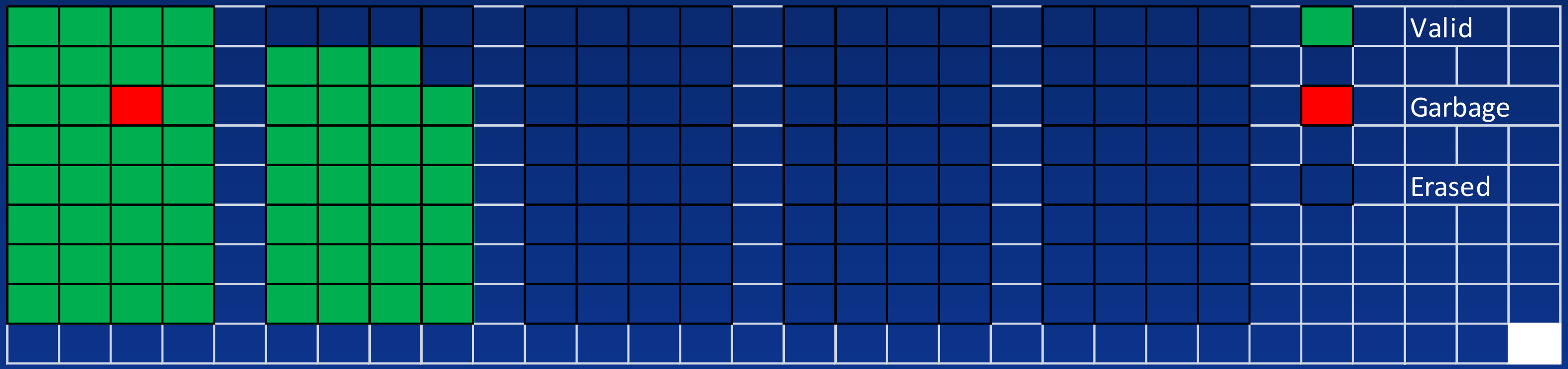
Garbage Collection



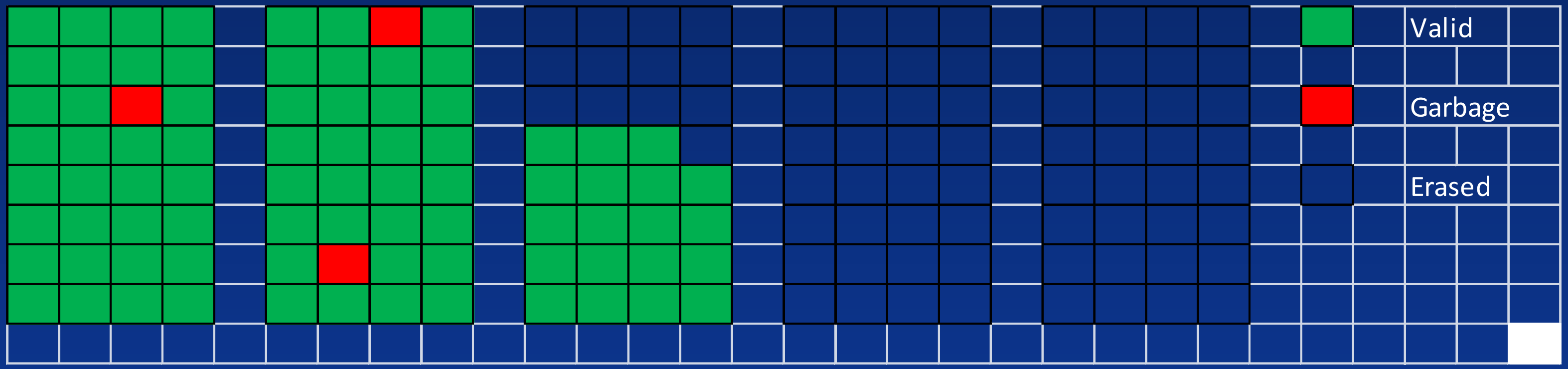
Garbage Collection



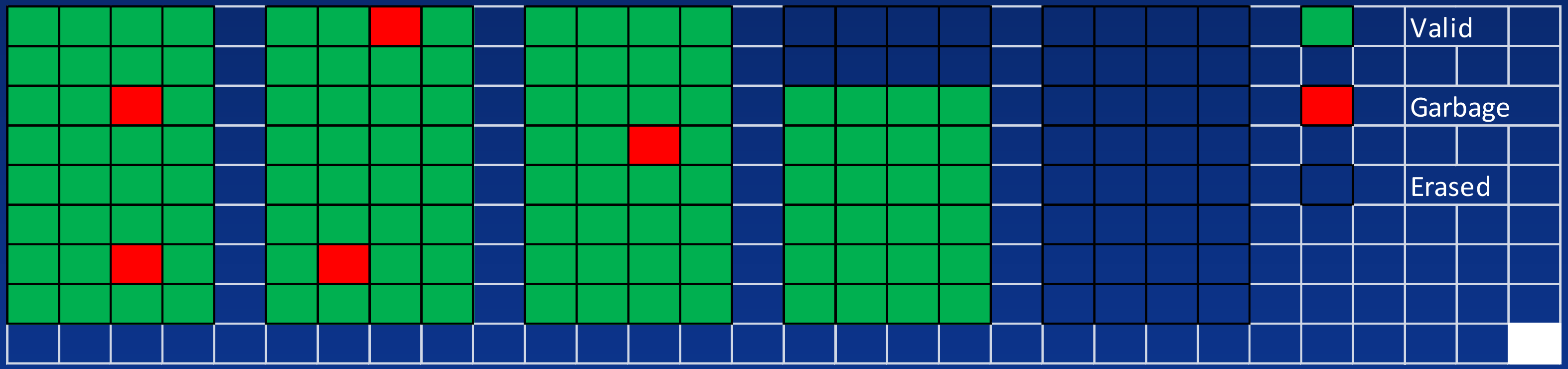
Garbage Collection



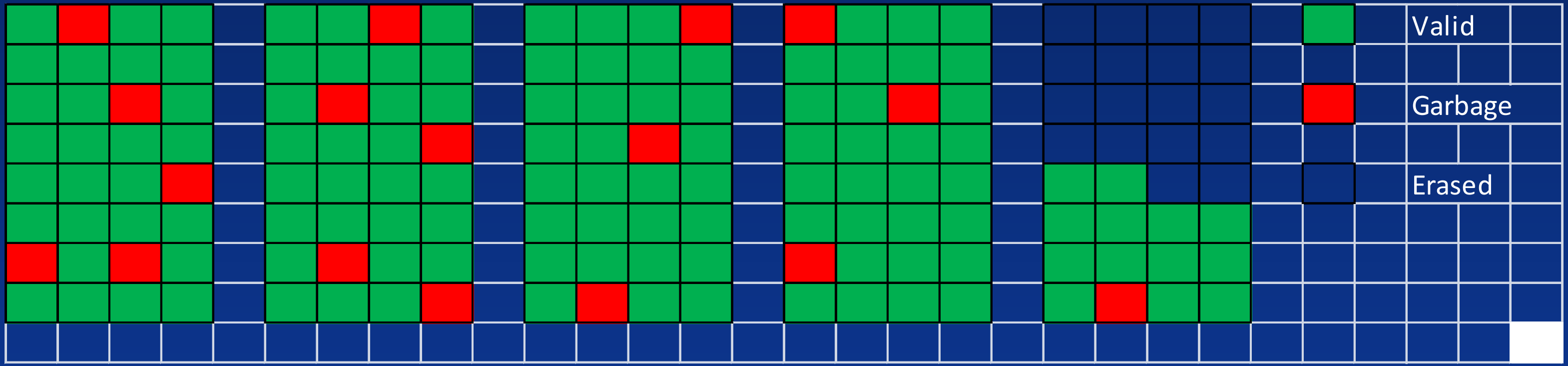
Garbage Collection



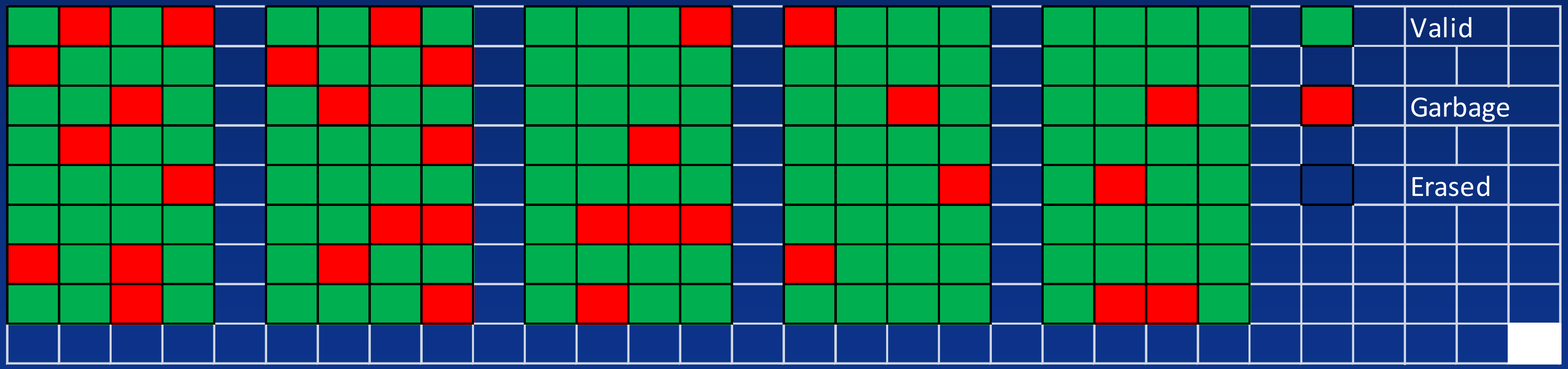
Garbage Collection



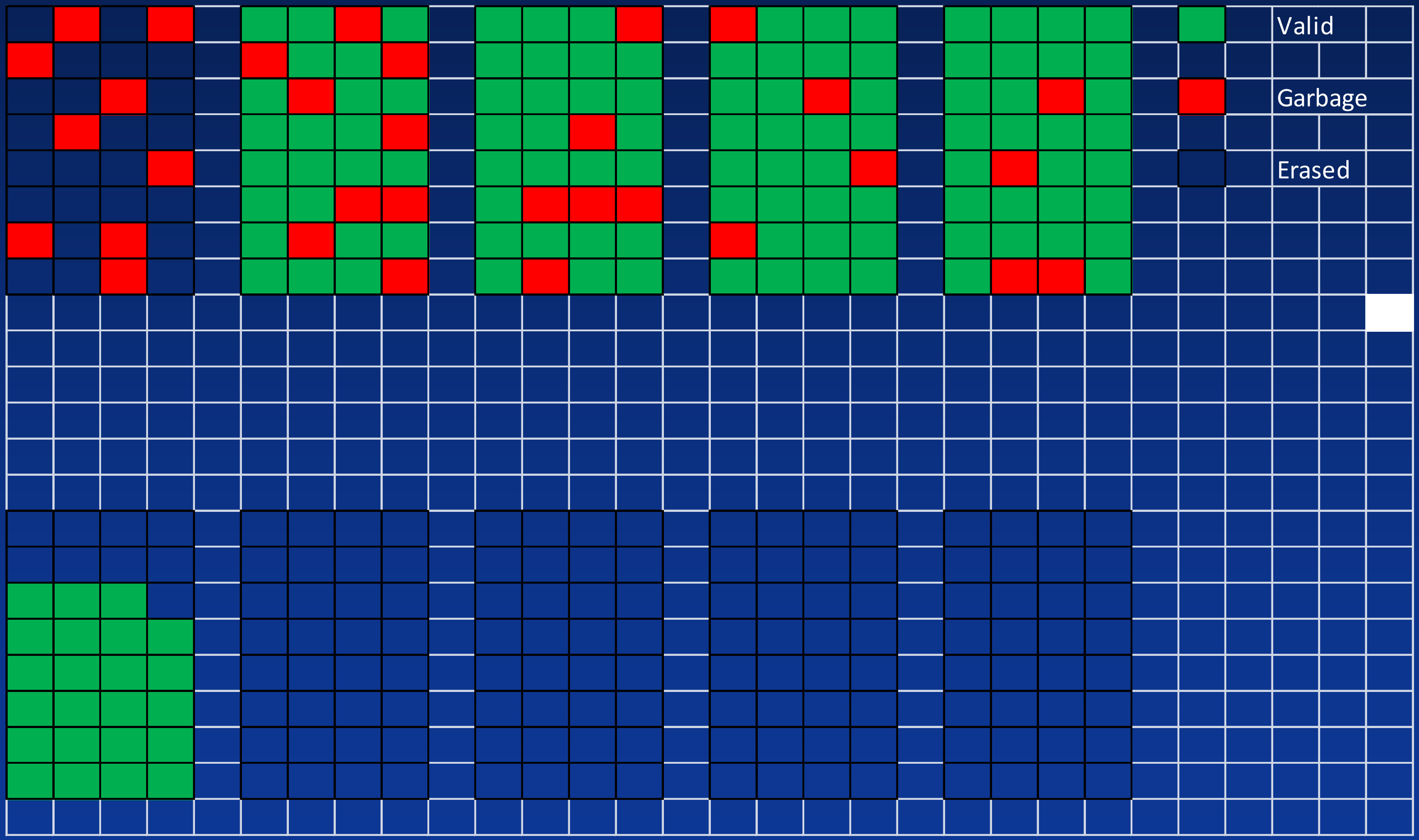
Garbage Collection



Garbage Collection



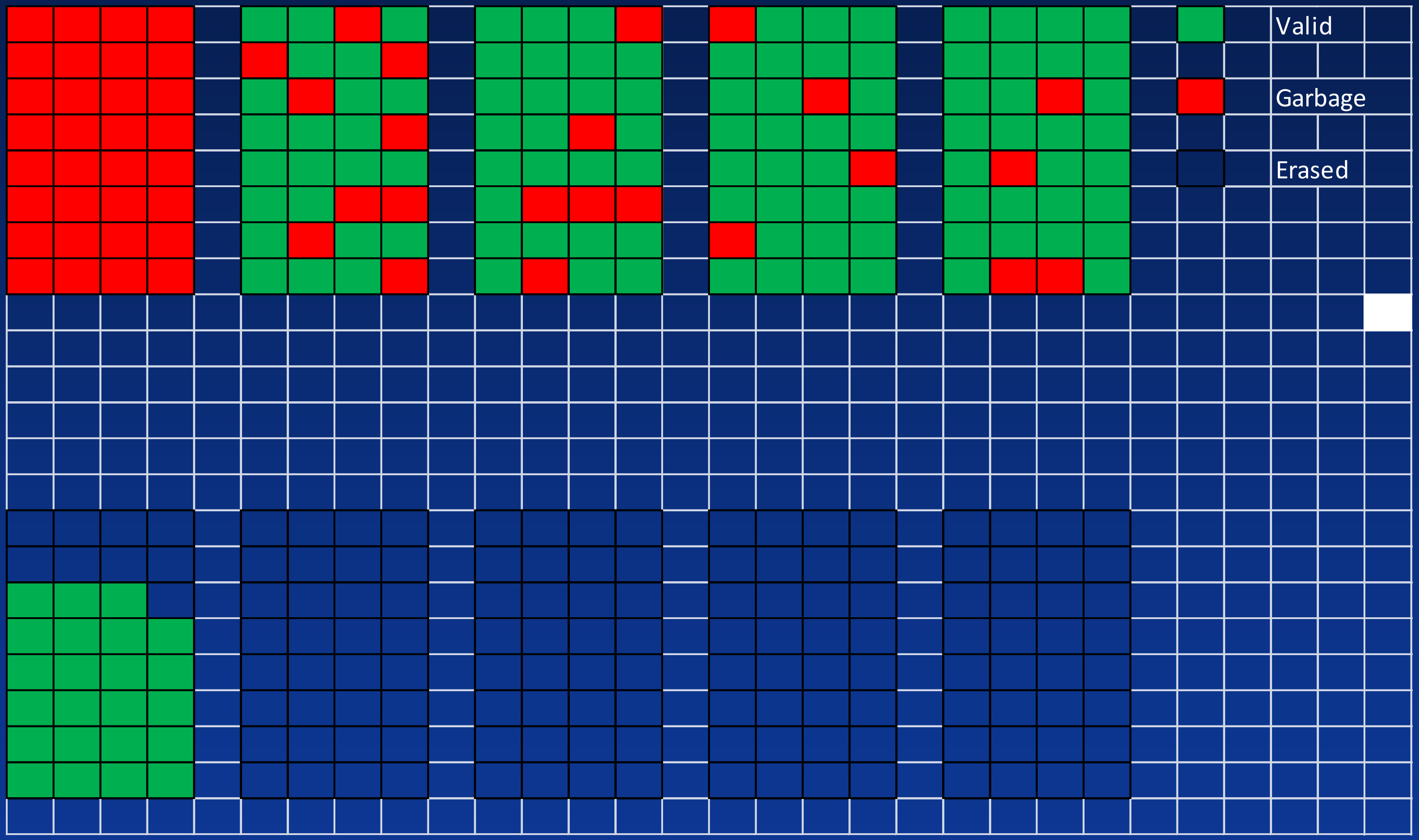
Garbage Collection



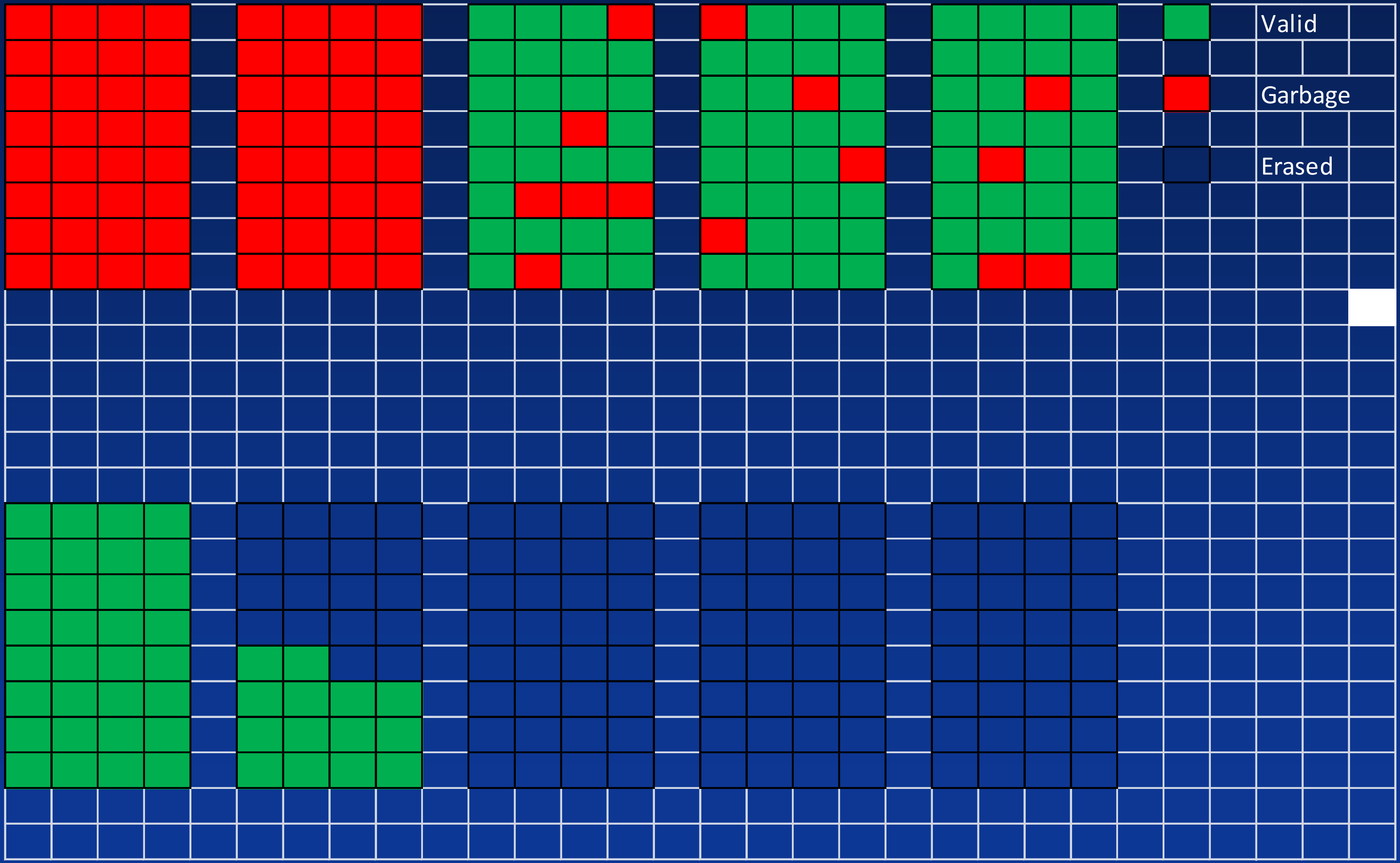
🟩 Valid

🟥 Garbage

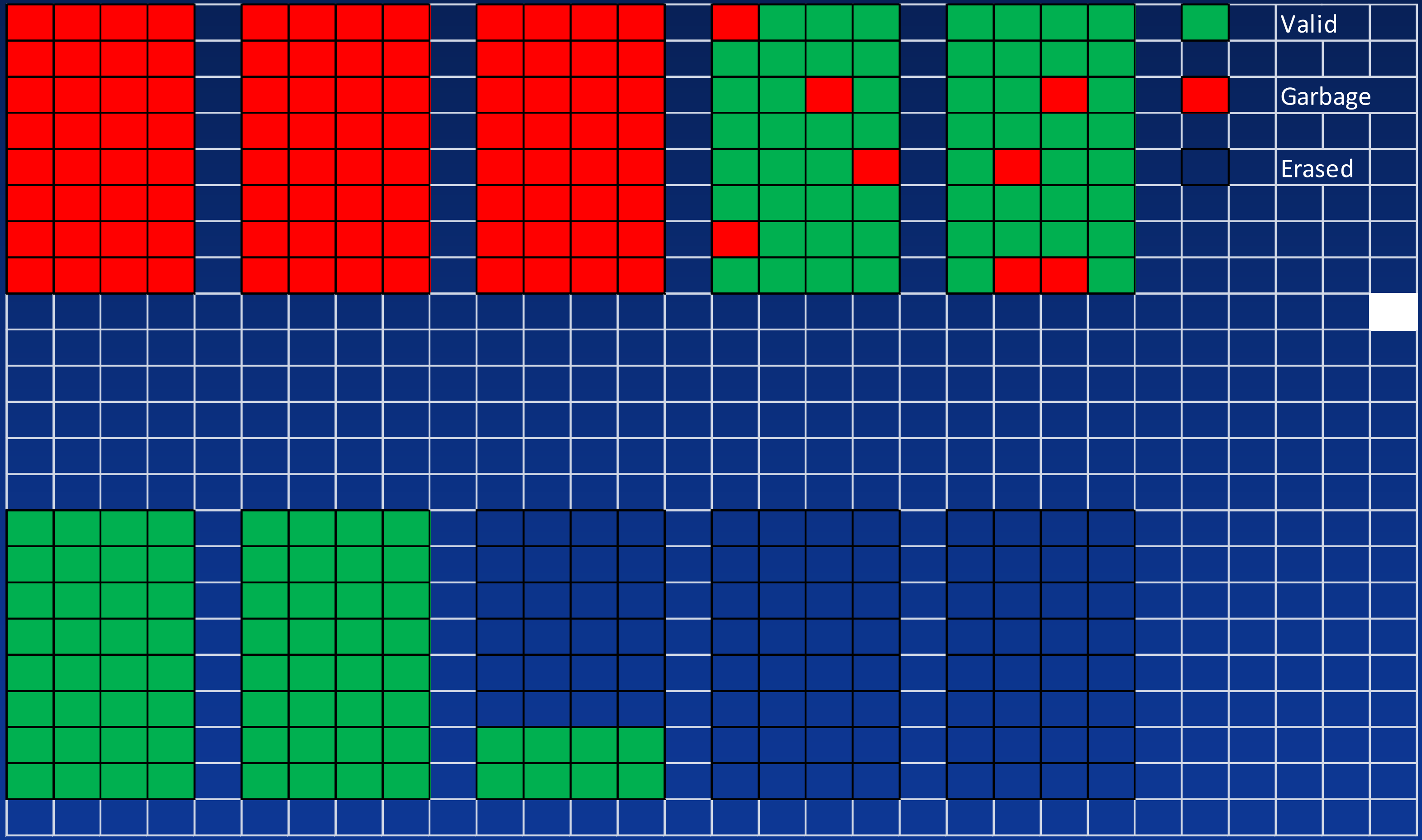
■ Erased


🟩 Valid
🟥 Garbage
Erased

Garbage Collection



Garbage Collection

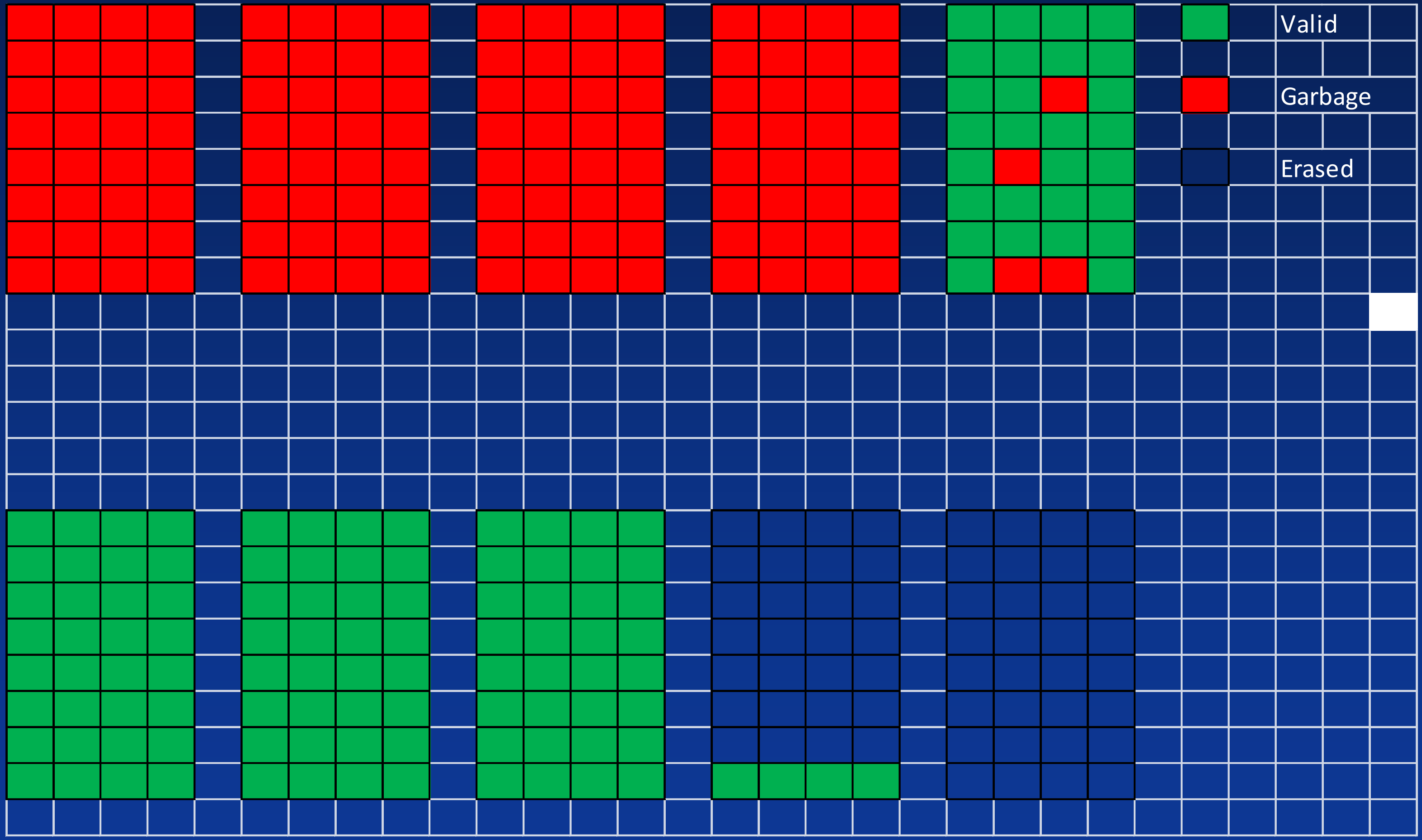


🟩 Valid

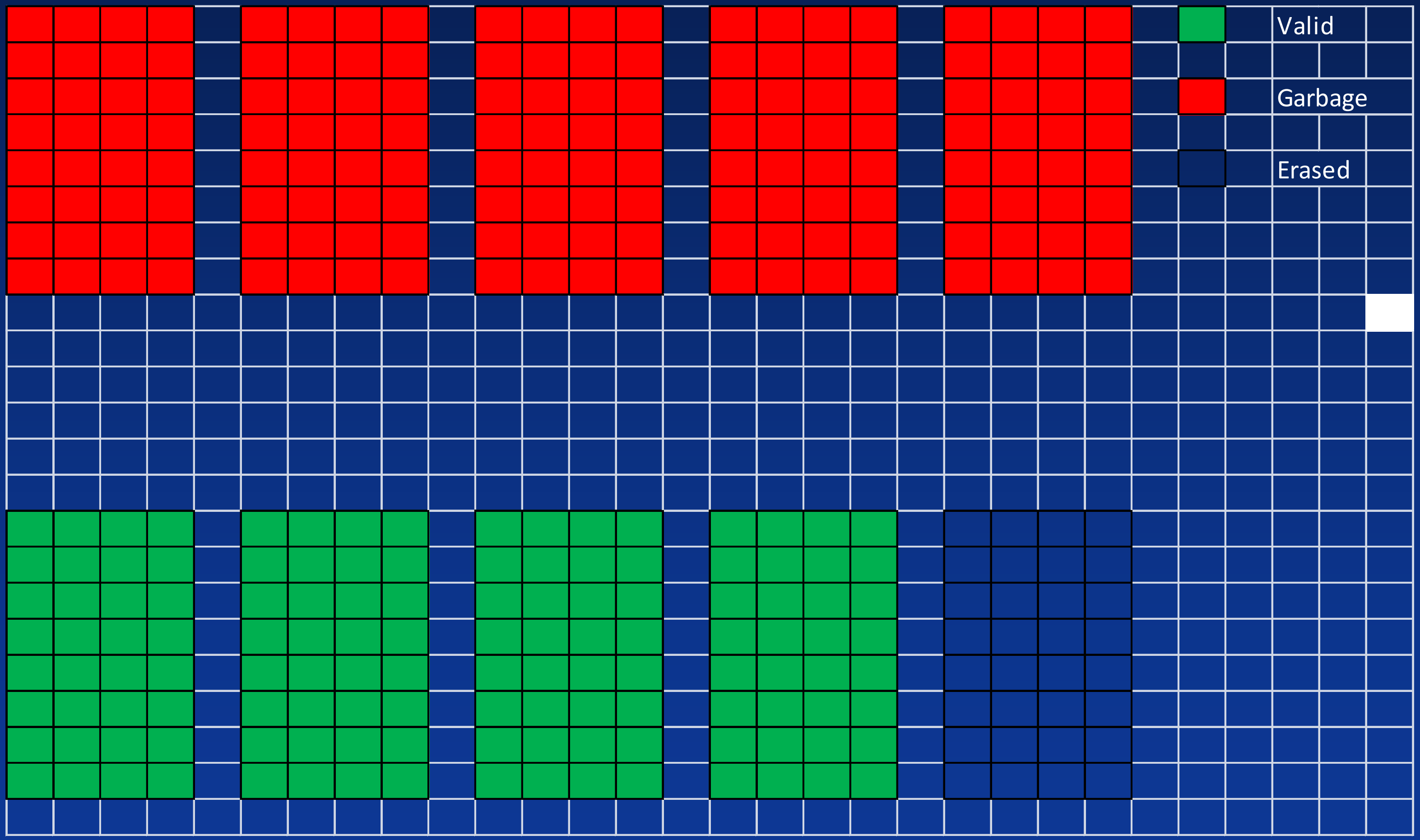
🟥 Garbage

■ Erased

Garbage Collection



Garbage Collection

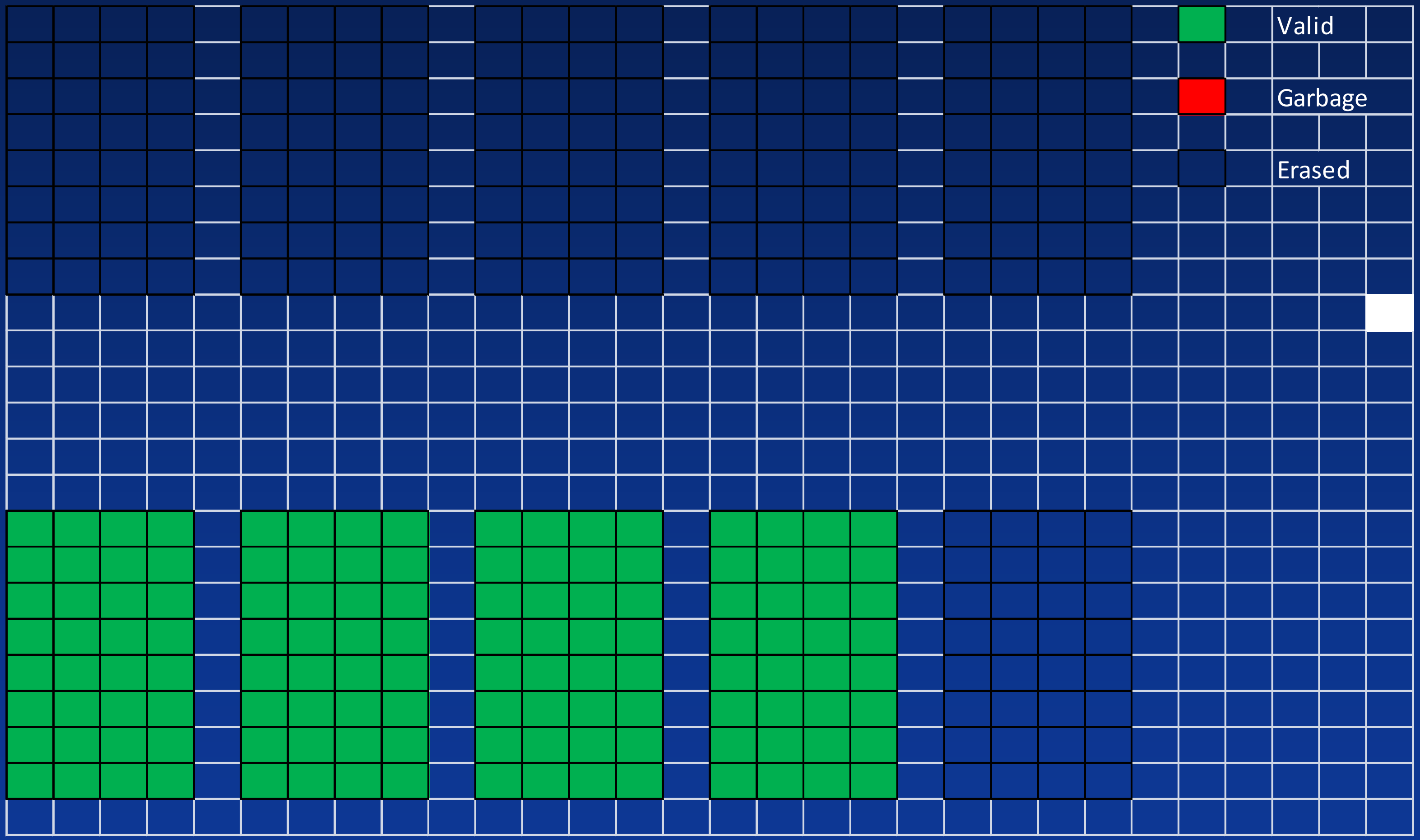


🟩 Valid

🟥 Garbage

▨ Erased

Garbage Collection



🟩 Valid

🟥 Garbage

Erased

Garbage Collection

- Is there a faster way to move the data?
  - Inter-flash chip copy
    - Copy back may be problematic as you can’t use this feature for multi die operations.
    - Read out the data from the flash and program it into flash through ECC HW.

Wear Levelling

Wear Levelling

- Flashes have a limited life span which is measured by its Program / Erase Cycle count. Beyond this limit, the data integrity starts degrading.

- Wear Leveling ensures that the firmware evenly distributes the writes throughout the flash.

- Wear Leveling techniques extends the life of SSDs.

Wear Levelling

- **Wear Levelling Techniques:**
  - **Dynamic Wear Leveling**
    - Involves monitoring the erase counts of blocks from a pool of ‘data free’ blocks and selecting the block with the lowest number of erases for the next write operation.
  - **Static Wear Leveling**
    - Identifies partially filled blocks whose data has remained static for some time and whose erase count is substantially lower compared to other blocks.
    - This block is freed up of its static data so that this block is available for future write.

Performance and Robustness

Performance and Robustness

- Performance
  - Factors effecting performance:
    - Inadequate FTL architecture.
      - FTL and associated buffers are not adequately designed for the amount of data coming from the host.
    - Adding garbage collection can dramatically reduce performance. If garbage collection is executed during a command sequence this will reduce performance.
      - Running garbage collection during idle periods maybe the easiest way of maintaining performance during busy periods.
    - System inadequacies.
      - Firmware Bottlenecks in command processing.
      - Inadequate buffer allocation.
      - Processor speed.
  - Understanding these constraints and having an architecture that reduces and works around these system inadequacies will greatly improve performance.

Performance and Robustness

- Robustness.
  - We have gone through several firmware paths that are susceptible to error.
    - FTL
      - Garbage Collection
      - Wear Levelling
      - Scrambling
    - OS interaction
      - Different command sizes
      - IPM (Interface Power Management) (HIPM & DIPM)
      - Device Power Management

Performance and Robustness

- Ensuring stability across these areas requires a test strategy that can fully stress the system, focusing on the areas above.
  - Easy tests to run.
    - Long soaks. Will the firmware run for days without error.
    - Testing to specification. Does the SSD meet ATA specifications / ONFI specifications.
  - Complex testing
    - Can only be done with an intermit understanding of the system.
      - Looking for bugs wont always work, sometime specific equipment is required to uncover the most difficult issues.

Thank You!