Solid State Drives
Data Reliability and Lifetime

White Paper
Alan R. Olson & Denis J. Langlois
April 7, 2008

Abstract

The explosion of flash memory technology has dramatically increased storage capacity and decreased the cost of non-volatile semiconductor memory. The technology has fueled the proliferation of USB flash drives and is now poised to replace magnetic hard disks in some applications. A solid state drive (SSD) is a non-volatile memory system that emulates a magnetic hard disk drive (HDD). SSDs do not contain any moving parts, however, and depend on flash memory chips to store data. With proper design, an SSD is able to provide high data transfer rates, low access time, improved tolerance to shock and vibration, and reduced power consumption. For some applications, the improved performance and durability outweigh the higher cost of an SSD relative to an HDD.

Using flash memory as a hard disk replacement is not without challenges. The nano-scale of the memory cell is pushing the limits of semiconductor physics. Extremely thin insulating glass layers are necessary for proper operation of the memory cells. These layers are subjected to stressful temperatures and voltages, and their insulating properties deteriorate over time. Quite simply, flash memory can wear out. Fortunately, the wear-out physics are well understood and data management strategies are used to compensate for the limited lifetime of flash memory.

Floating Gate Flash Memory Cells

Flash memory was invented by Dr. Fujio Masuoka while working for Toshiba in 1984. The name "flash" was suggested because the process of erasing the memory contents reminded him of the flash of a camera. Flash memory chips store data in a large array of floating gate metal–oxide–semiconductor (MOS) transistors. Silicon wafers are manufactured with microscopic transistor dimension, now approaching 40 nanometers.

Floating Gate MOS Transistor

A floating gate memory cell is a type of metal-oxide-semiconductor field-effect transistor (MOSFET). Silicon forms the base layer, or substrate, of the transistor array. Areas of the silicon are masked off and infused with different types of impurities in a process called doping. Impurities are carefully added to adjust the electrical properties of the silicon. Some impurities, for example phosphorous, create an excess of electrons in the silicon lattice. Other impurities, for example boron, create an absence of electrons in the lattice. The impurity levels and the proximity of the doped regions are set out in a lithographic manufacturing process. In addition to doped silicon regions, layers of insulating silicon dioxide glass (SiO_2) and conducting layers of polycrystalline silicon and aluminum are deposited to complete the MOSFET structure.

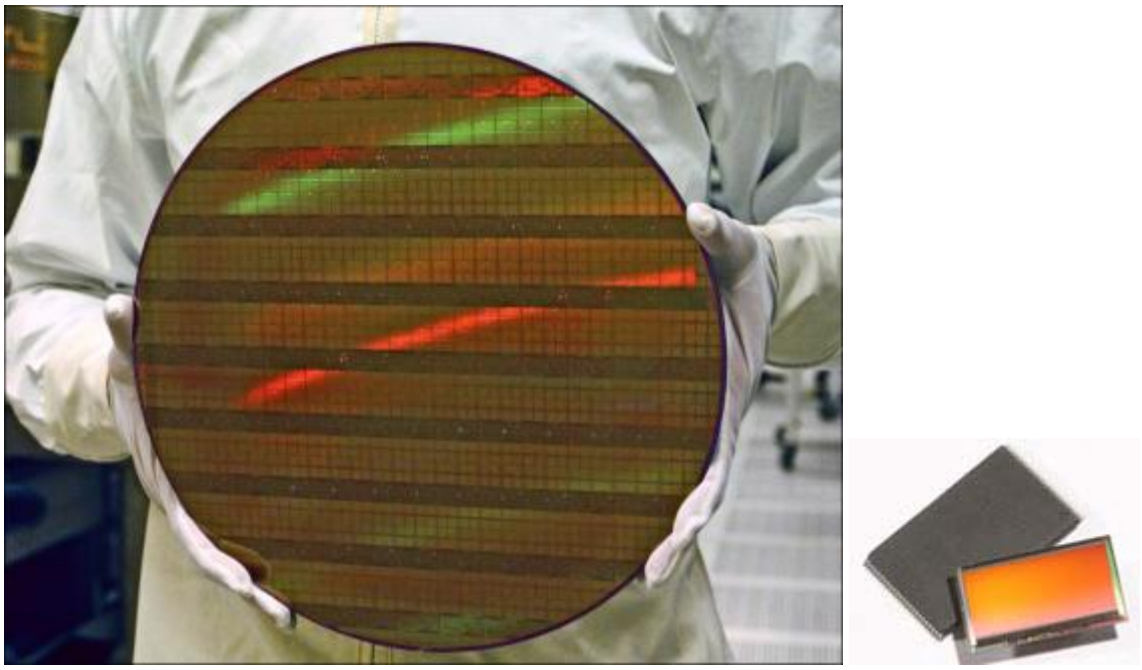

Figure 1 – (Left) A 300 mm silicon wafer containing hundreds of NAND flash memory chips
(Right) A single NAND flash memory chip silicon die measures only 18 mm x 12 mm (not to scale)
Copyright Intel Corporation

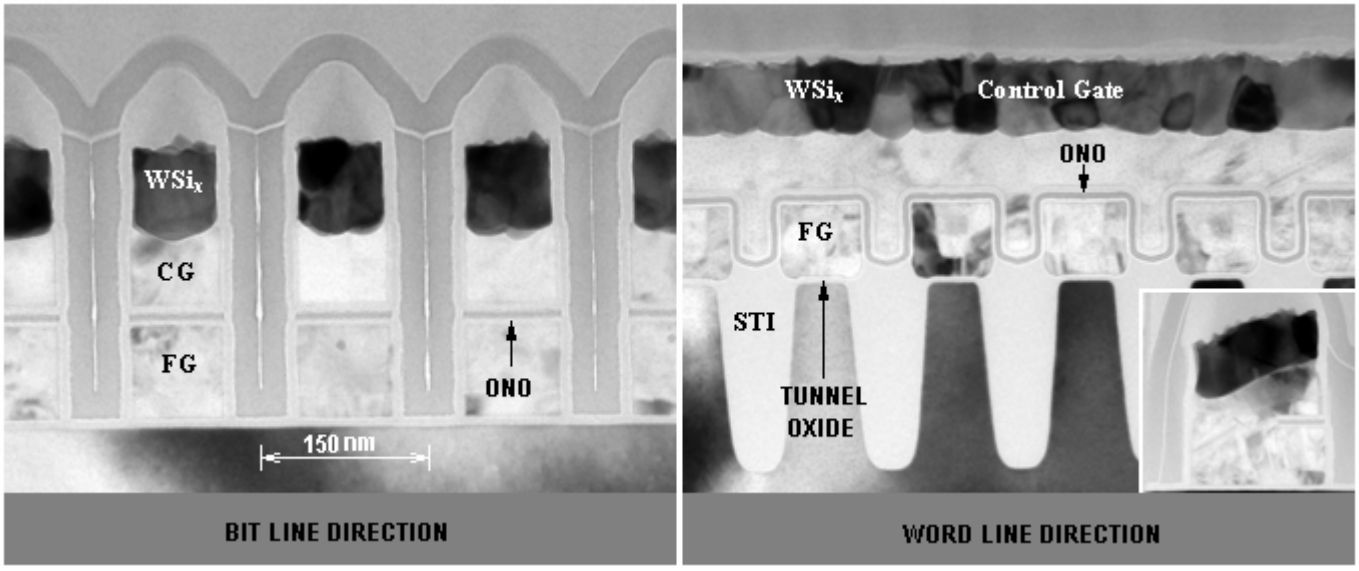

Figure 2 – Scanning electron microscope (SEM) cross-sections of a typical single-level cell (SLC) NAND flash memory cell
Copyright Chipworks

MOS transistors work by forming an electrically conductive channel between the source and drain terminals. When a voltage is applied to the control gate, an electric field causes a thin negatively charged channel to form at the boundary of the SiO_2 and between the source and drain regions. When the N-channel is present, electricity is easily conducted from the source to the drain terminals. When the control voltage is removed, the N-channel disappears and no conduction takes place. The MOSFET operates like a switch, either in the on or off state.

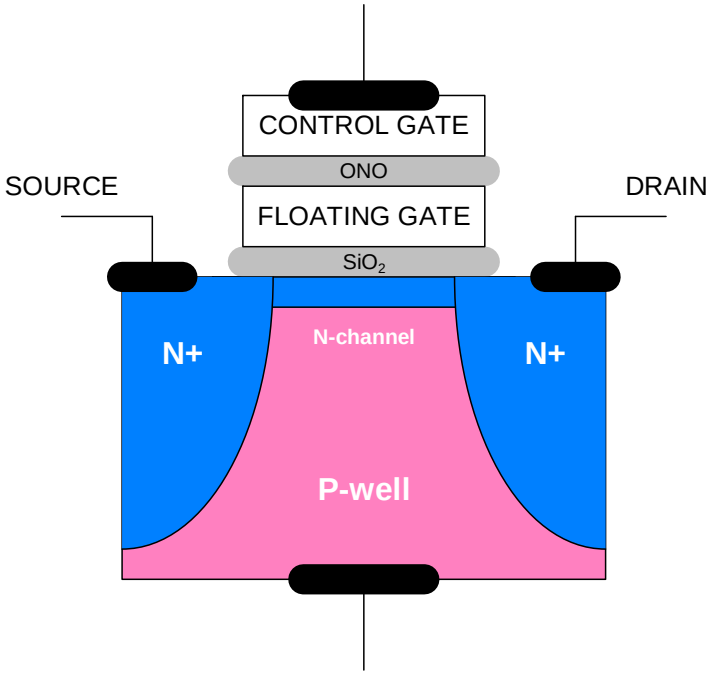

Figure 3 – A cross sectional depiction of a floating gate MOSFET memory cell (not to scale)

In addition to the control gate, there is a secondary floating gate which is not electrically connected to the rest of the transistor. The voltage at the control gate required for N-channel formation can be changed by modifying the charge stored on the floating gate. Even though there is no electrical connection to the floating gate, electric charge can be put in to and taken off of the floating gate. A quantum physical process called Fowler-Nordheim tunneling coaxes electrons through the insulation between the floating gate and the P-well. When electric charge is removed from the floating gate, the cell is considered in an erased state. When electric charge is added to the floating gate, the cell is considered in the programmed state. A charge that has been added to the floating gate will remain for a long period of time. It is this process of adding, removing and storing electric charge on the floating gate that turns the MOSFET into a memory cell.

Erasing the contents of a memory cell is done by placing a high voltage on the silicon substrate while holding the control gate at zero. The electrons stored in the floating gate tunnel through the oxide barrier into the positive substrate. Thousands of memory cells are etched onto a common section of the substrate, forming a single block of memory. All of the memory cells in the block are simultaneously erased when the substrate is “flashed” to a positive voltage. An erased memory cell will allow N-channel formation at a low control gate voltage because all of the charge in the floating gate has been removed. This is referred to as logic level “1” in a single-level cell (SLC) flash memory cell.

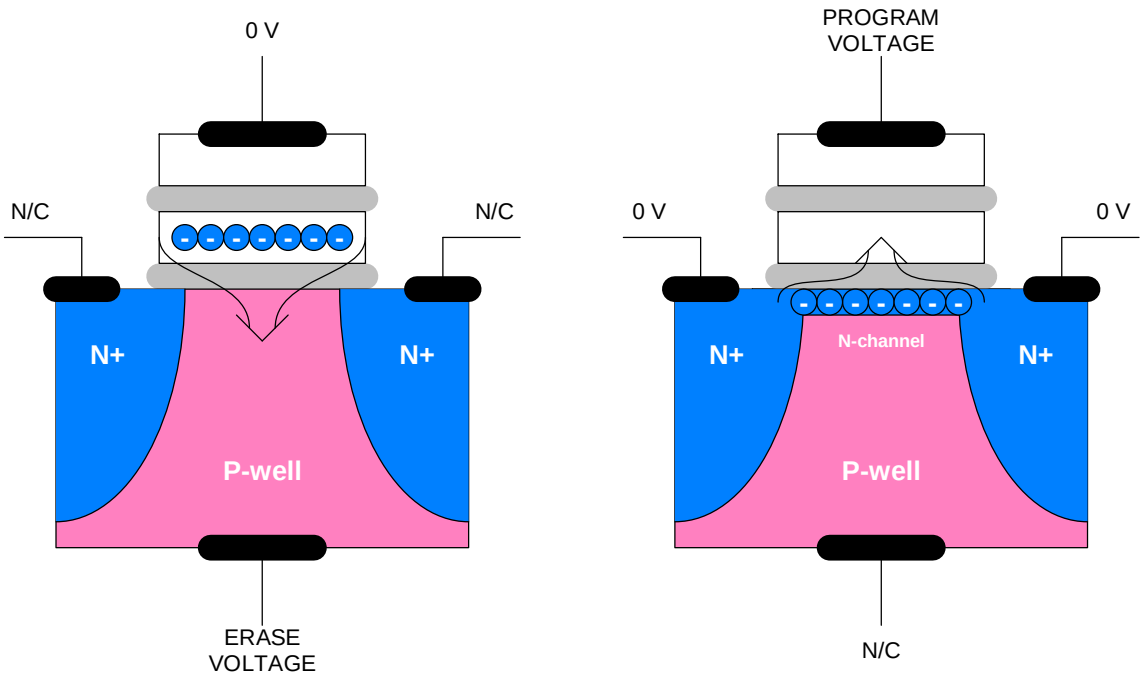

Figure 4 – Erasing and programming the contents of a flash memory cell via Fowler-Nordheim tunneling

The cell is programmed by placing a high voltage on the control gate while holding the source and drain regions at zero. The high electric field causes the N-channel to form and allows electrons to tunnel through the oxide barrier into the floating gate. Programming the memory cells is performed one word at a time (i.e., cell by cell) and usually an entire page (e.g., 2048 bytes) is programmed in a single operation. A programmed memory cell inhibits the control gate from forming an N-channel at normal voltages because of the negative charge stored on the floating gate. To form the N-channel in the substrate, the control gate voltage must be raised to a higher level. This is referred to as logic level “0” in an SLC flash memory cell.

Single vs. Multiple Level Cells

The control gate voltage necessary to form the N-channel is controlled by the charge on the floating gate. The required voltage is called the gate threshold voltage and is labeled V_{th}.

With SLC flash memory, there is only one programmed state in addition to the erased state. The total of two states allows a single data bit to be stored in the memory cell.

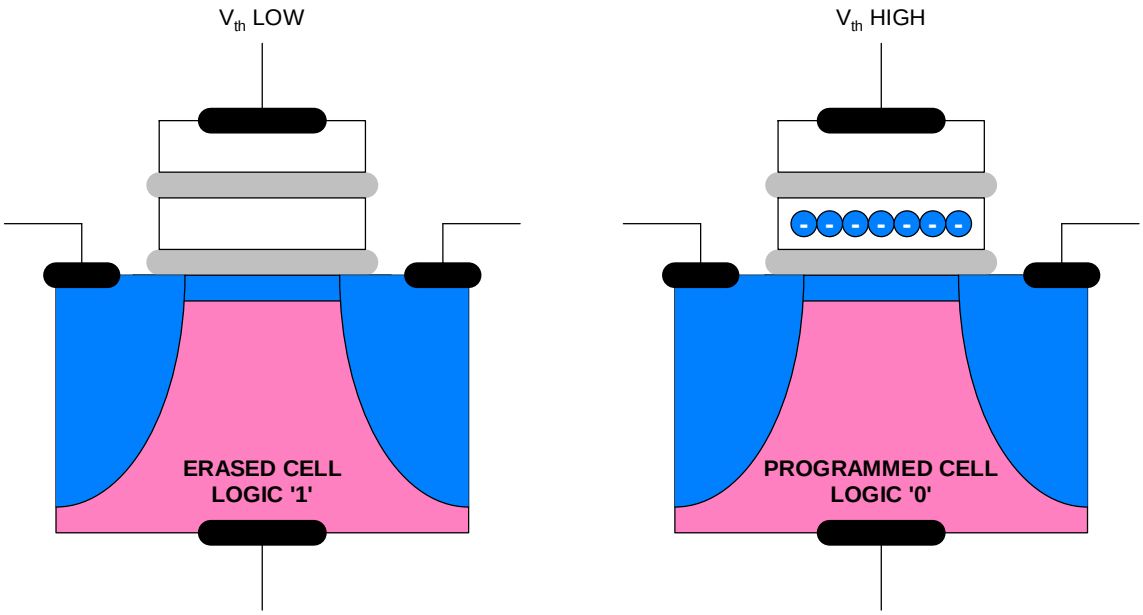

Figure 5 – The erased and programmed states of an SLC memory cell

Because there are only two states of charge on the floating gate, there are only two threshold voltages required at the control gate. It is important to note that the control gate threshold voltage varies from one cell to the next. This is a result of the normal manufacturing process variations. The threshold voltages are usually depicted as bell shaped distributions.

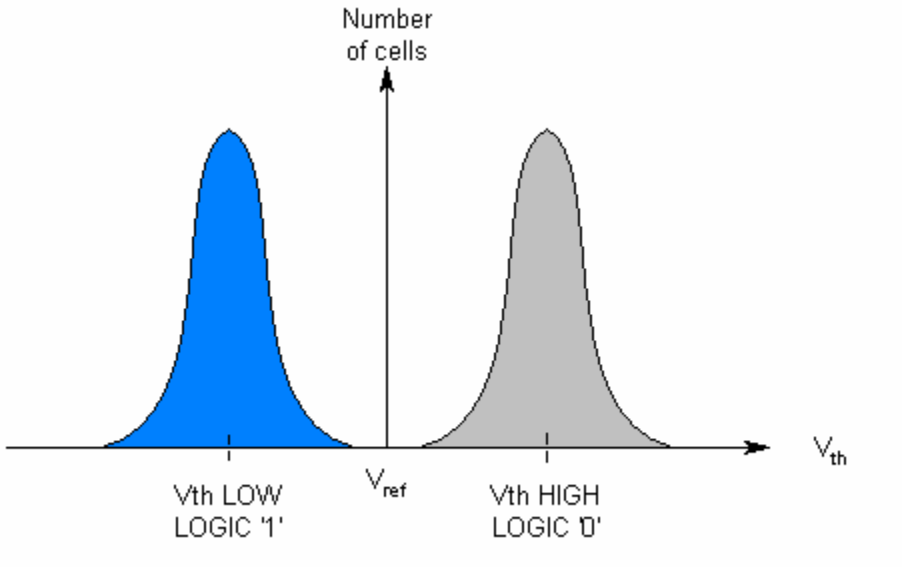

Figure 6 – The distribution of control gate threshold voltages for a large number of SLC memory cells

With multi-level cell (MLC) flash memory, there are multiple programmed states in addition to the erased state. Typically, there are three programmed states resulting in a total of four states. This allows two data bits to be stored in the memory cell. The additional two programmed states result from partially charging the floating gate. A partially charged floating gate will result in an intermediate value for the control gate threshold voltage. In MLC devices, the

partial charging of the floating gate is carefully monitored and the resulting distribution of the intermediate threshold voltages tends to be tighter.

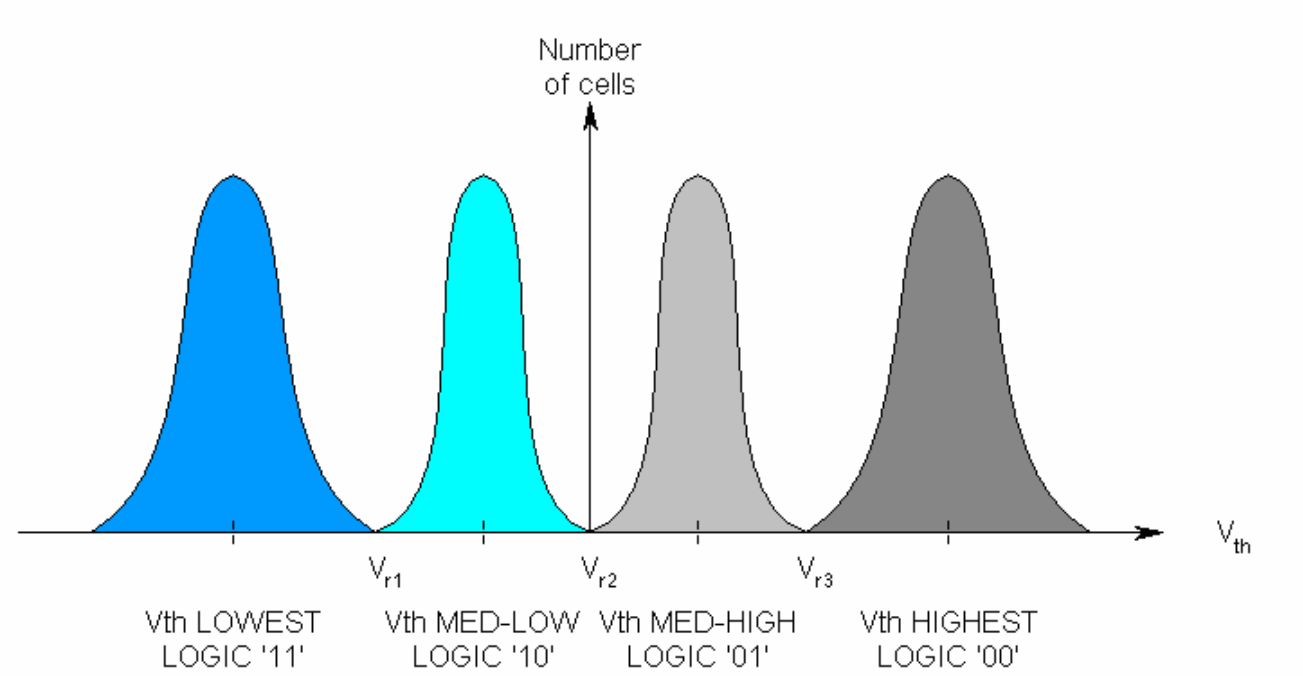

Figure 7 – The distribution of control gate threshold voltages for a large number of MLC memory cells

MLC memory cells are erased in the same way as SLC memory cells and the time required to erase a block is similar. MLC memory cells take longer to program than SLC memory cells, however. The charge state of the floating gate must be carefully monitored during programming to ensure that the resulting control gate threshold voltages are unambiguous.

MLC memory cells tend to wear out faster than SLC memory cells because they are more sensitive to physical changes in the insulating SiO_2 layer. MLC memory cells also experience higher levels of read errors due to variations in control gate threshold voltage and disturbances from neighboring memory cells.

Flash Memory Array Architecture - NAND vs. NOR

Flash memory cells are organized into a hierarchy of bytes, pages, blocks and planes. The organization of one type of 4 Gigabit (Gb) SLC NAND flash memory chip is shown below.

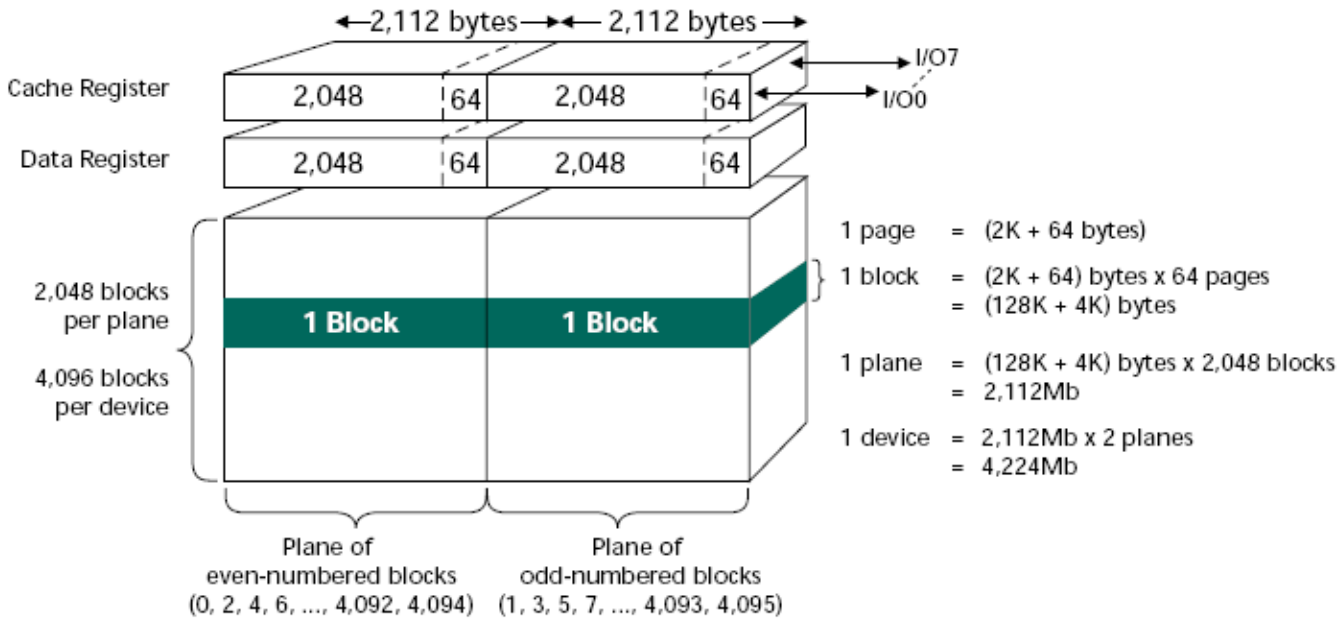

Figure 8 - Organization of MT29F4G08AAA 4Gb SLC NAND flash memory chip
Copyright Micron

In this example, the 4Gb chip consists of 4096 blocks, divided into even and odd numbered blocks. Each block contains 64 pages. The memory page contains 2048 bytes of user data and 64 bytes of non-user data. The additional 64 bytes per page are used by the flash memory controller for data management. The flash memory controller may additionally divide the 2048 bytes per page into four sectors of 512 bytes, a typical size for HDDs.

NAND flash memory chips arrange the memory cells in a logical “not-and” (NAND) configuration. This arrangement strings together all of the cells for a common input / output (I/O) bit across all memory pages.

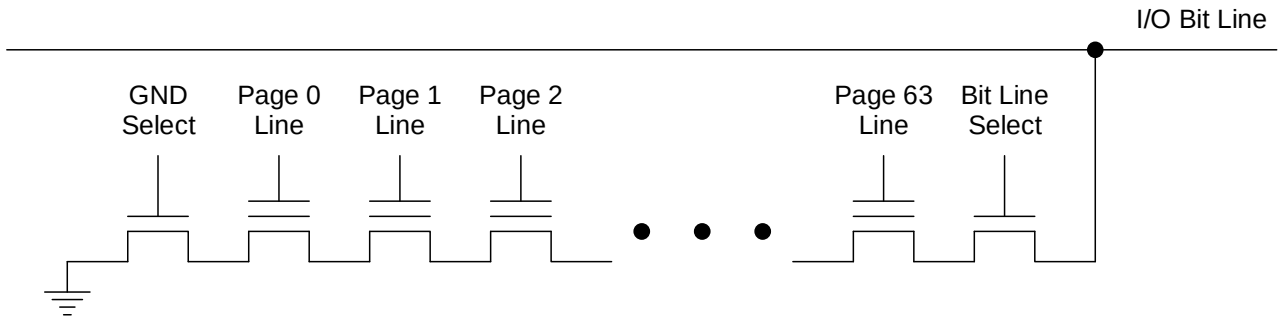

Figure 9 – NAND flash memory cell arrangement

Because of this arrangement, it is not possible to directly access individual data bytes within a memory page. The flash memory controller must read an entire page of memory from the device. Also, an entire page must usually be programmed at once, although some devices permit partial page programming. That makes NAND flash unsuitable for most random byte-access memory applications.

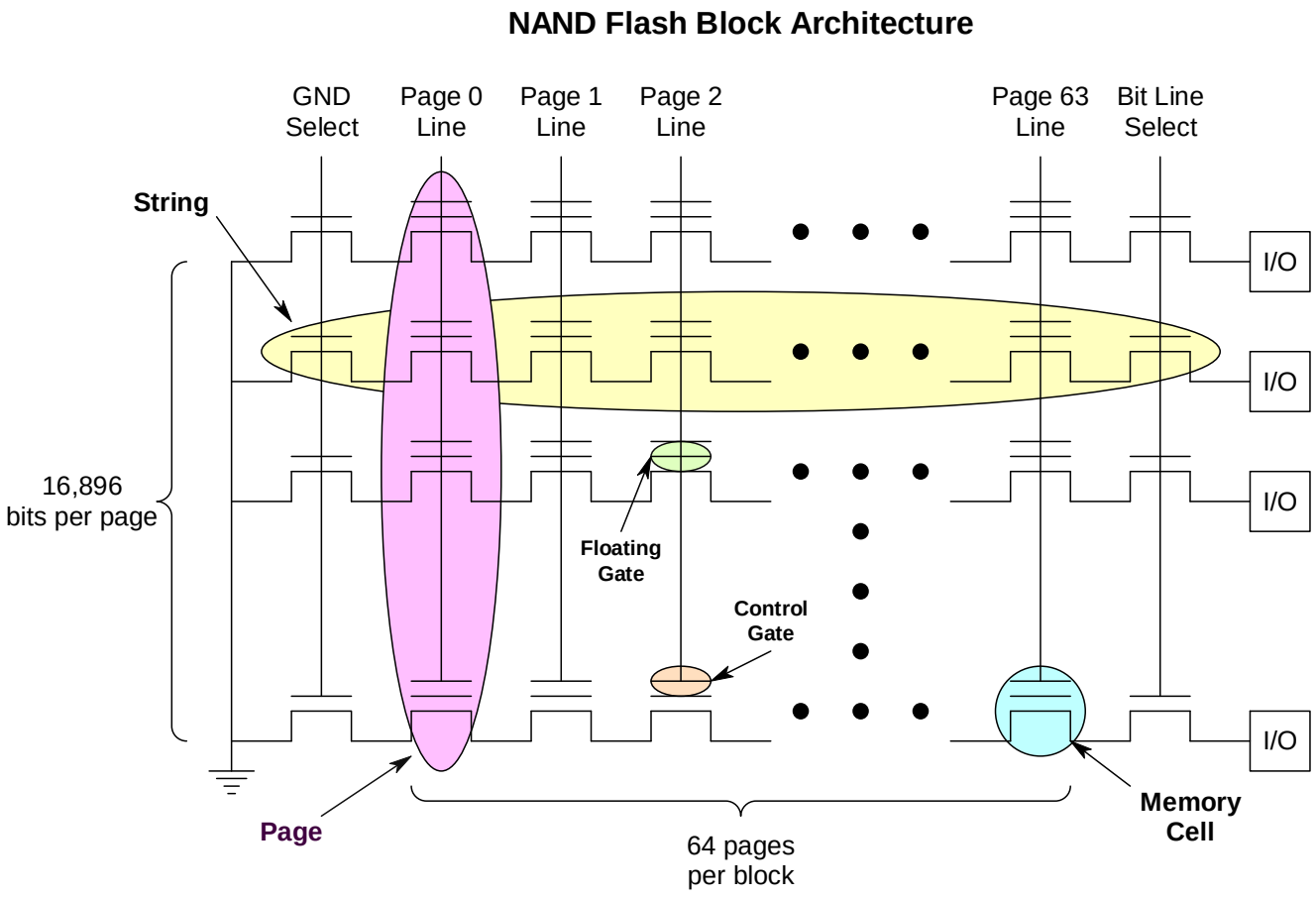

Figure 10 – Block architecture of MT29F4G08AAA 4Gb SLC NAND flash memory chip

A different type of arrangement combines cells into a “not-or” (NOR) logic configuration. NOR flash memory allows each memory cell to be individually addressed and is suitable for random byte-access memory application.

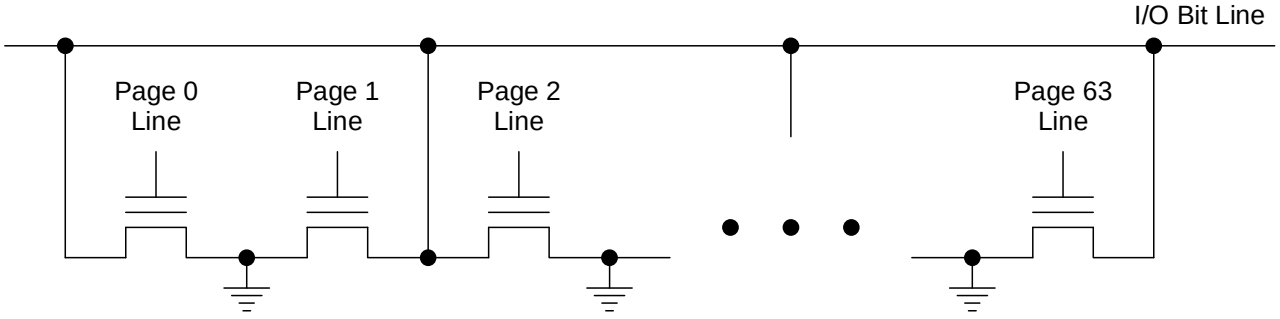

Figure 11 – NOR flash memory cell arrangement

NOR flash memory is less densely packed on the silicon due to the space required for all of the I/O bit lines and ground connections. NOR flash memory has a corresponding higher cost per bit than NAND flash memory. Because NOR flash memory allows random byte-access, it can be used as program storage for microcontrollers.

NAND flash memory is more densely packed on the silicon due to the space saved by reducing the number of I/O bit lines and ground connections. NAND flash memory has a lower cost per bit than NOR flash memory. NAND flash memory is more suitable for data file storage where random byte-access is not essential. NAND flash memory cells are more susceptible to disturbances from reading and programming neighboring pages due to the high density of memory cells on the silicon wafer.

Sources of Data Errors with Flash Memory Cells

Floating gate memory cells are carefully designed and insulated and can retain a programming charge for more than 10 years under favorable conditions. While this is a long time, consideration must be given to the less than ideal conditions these devices may experience.

SLC memory cells can survive approximately 100,000 erase / program cycles before the tunnel oxide begins to wear out. MLC memory cells have a shorter lifetime of approximately 10,000 erase / program cycles. Erasure and programming subjects the tunnel oxide to stress from large electric fields. The tunnel oxide insulation between the silicon substrate and the floating gate is very thin, less than 10 nm.

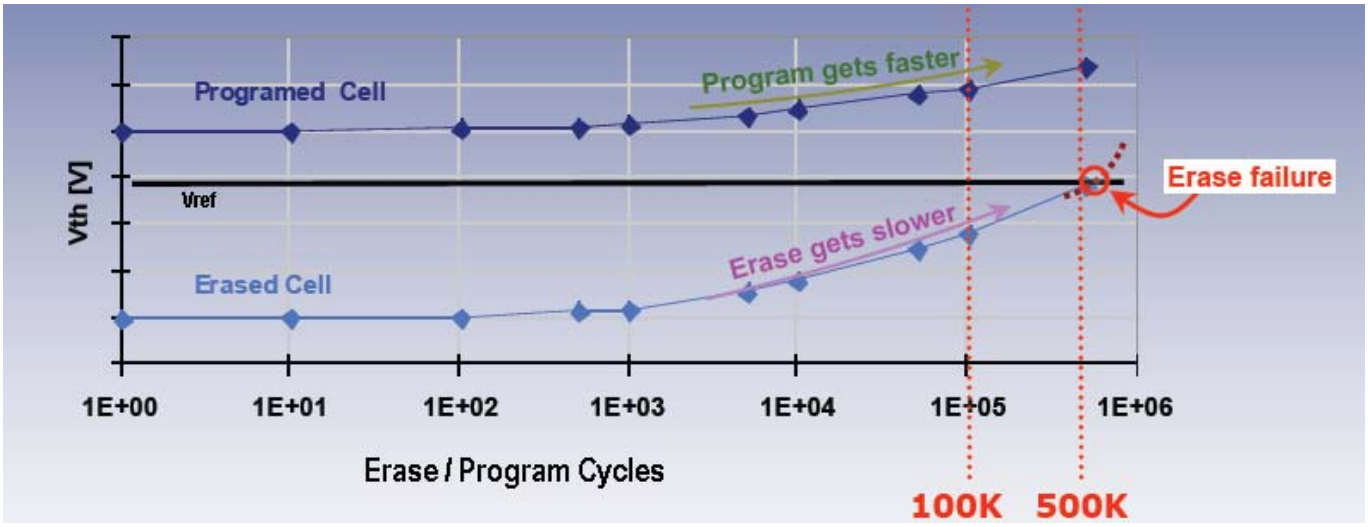

Figure 12 – SLC flash memory cell endurance characteristics
Copyright Samsung

Over time, the tunnel oxide will lose its insulating properties leading to the inability to erase or program the memory cell. As this occurs, the flash memory controller will retire blocks from use and replace them with reserve blocks. The deterioration of the tunnel oxide is compensated for by using a “wear-leveling” algorithm. The wear-leveling algorithm evenly spreads the erase / program cycles over all the memory blocks. Spreading the erase / program cycles over the whole device prevents data “hot spots” from prematurely retiring the device. Manufacturing defects can also cause the tunnel oxide to lose its insulating properties. However, memory cells that are inoperative after manufacturing are marked as unusable and don’t necessarily prevent the flash chip from being used.

Stressful conditions in the tunnel oxide also can lead to data retention problems. There also is the risk of data corruption from reading and programming neighboring memory pages. Data retention and corruption issues are compensated for by using powerful error correction codes.

Stress Induced Leakage Current

During erasure and programming the electric field strength across the tunnel oxide is very high -- several million volts per centimeter. Such high field strength can lead to structural changes in the SiO_2 layer. These structural changes can lead to defects that trap electrons in the oxide layer. The structural defects behave like tiny “cracks” in the insulation. These defects permit the charge on the floating gate to leak out into the substrate. Over time, more and more defects arise, which lead to a total breakdown of the oxide layer.

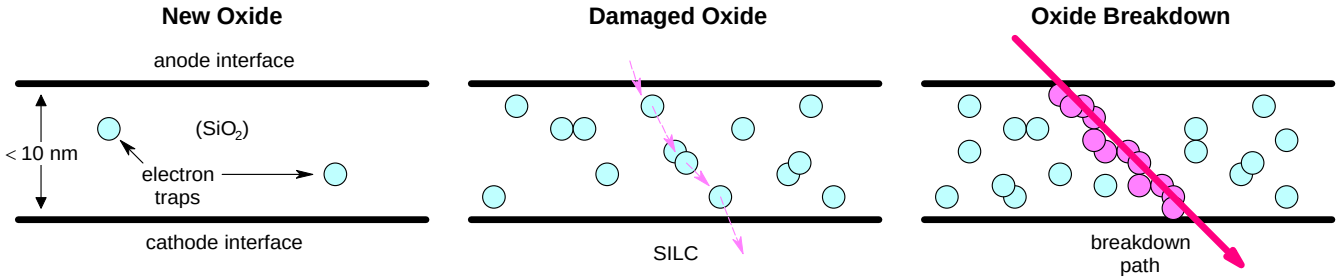

Figure 13 – A percolation model for stress-induced leakage current (SILC) and eventual oxide breakdown

Disturb Faults

Programming and reading memory pages can cause charge disruptions within adjacent memory pages. Given the high density of memory cells, voltage changes are capacitively coupled between memory cells in adjacent pages. The coupling can lead to random bit errors in the stored data.

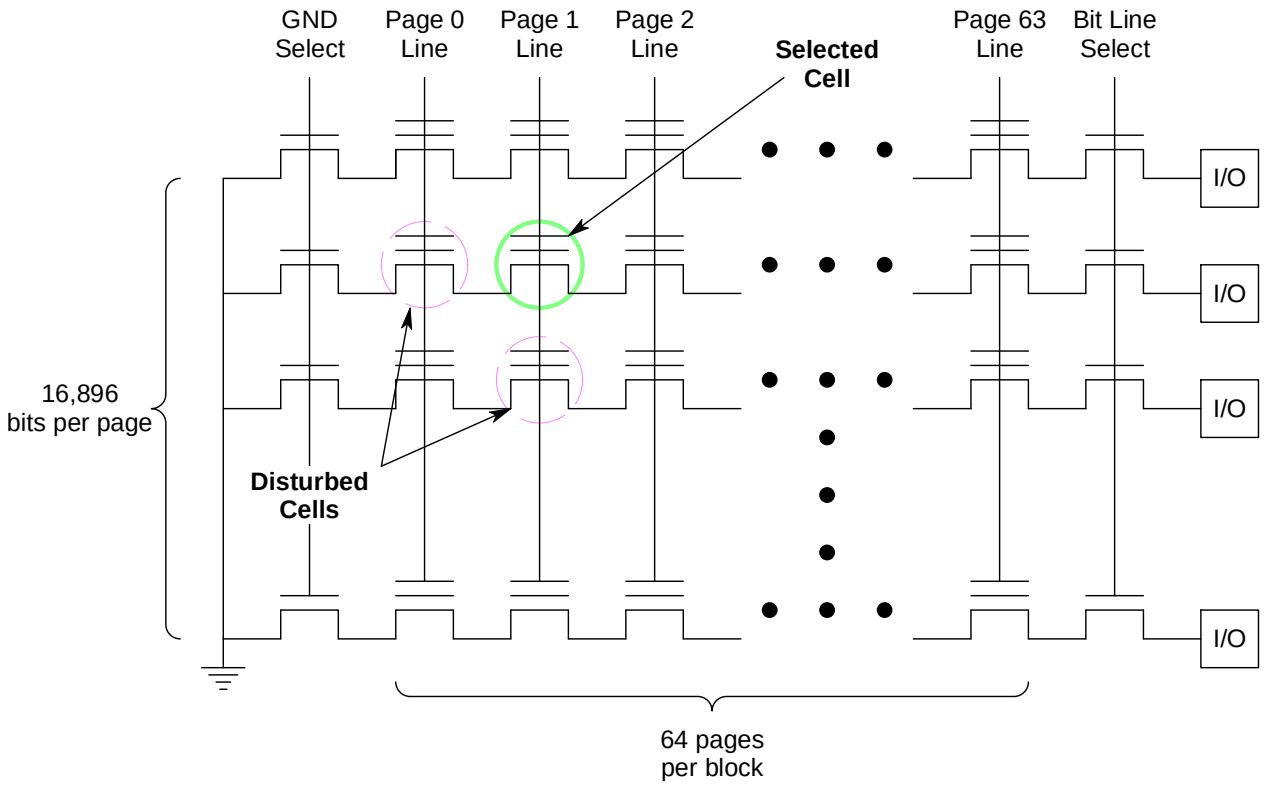

Figure 14 – Program and read activity can disturb the contents of adjacent cells

Error Correction Codes

The deterioration of the tunnel oxide over time and the disruptions from neighboring memory pages can lead to random bit errors in the stored data. While the chances of any given data bit being corrupted is quite small, the vast number of data bits in a storage system makes the likelihood of data corruption a very real possibility.

Error detection and correction codes are used in flash memory storage systems to protect the data from corruption. The construction and application of error control codes is a branch of mathematics called abstract algebra. Claude Shannon laid the theoretical foundation for error control coding and information theory in his ground breaking 1948 paper, “A Mathematical Theory of Communications.” All types of error control codes add redundancy to the information being transmitted through a communication channel or a storage system. The redundant data is calculated when the data is transmitted (or written) and checked when the data is received (or read). The redundant information is calculated so that a balance of correction power and efficiency is achieved.

Flash memory systems use a class of error correction codes (ECC) called block codes. With block codes, the redundant data bits are calculated for a fixed size block of user data, (e.g., 512 bytes). The addition of redundant ECC bits to the user data bits forms a larger set of bits called a code word. The code word space is designed to be sparsely populated. That is to say that not every possible code word is valid. ECCs rely on the concept of binary Hamming distance to calculate the difference between a received code word and a valid code word. ECCs use “maximum-likelihood” decoding algorithms to estimate a valid code word from a received code word that contains an error.

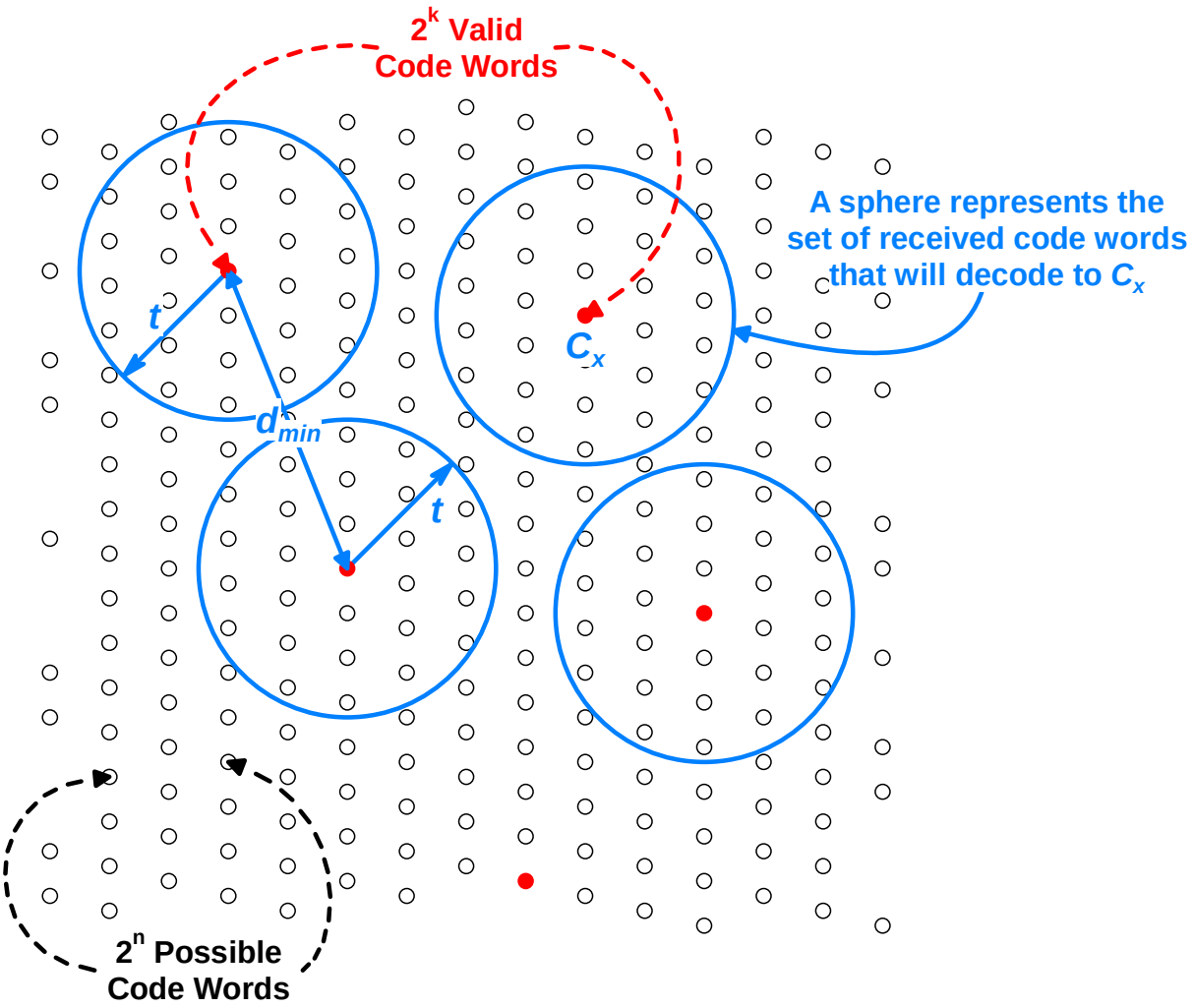

Figure 15 – Depiction of ECC words and maximum-likelihood decoding

Block codes are designed using mathematical parameters n, k, t and d_{min}. The size of the code word is n bits, yielding 2^n possible code words. The size of the user data is k bits, yielding 2^k valid code words.

The correction power of the code is t bits and can be thought of as the radius of a sphere surrounding each valid code word. The minimum Hamming distance between valid code words is d_{min} bits, where $d_{min} \geq 2t + 1$, insuring that the spheres do not overlap. The greater the Hamming distance between valid code words, the greater the correction power of the code. The correction power of the code is the maximum number of received bits in error which can be corrected. It is not the number of redundant ECC bits in the code word, however. The number of ECC bits stored with the data is a function of both the correction power of the code (the t parameter) and the size of the code word (the n parameter).

The process of encoding the ECC bits is done when data is written to the storage system. Circuits for calculating the ECC bits are fast and efficient and do not significantly penalize system performance. In a flash memory system, the ECC bits are stored in the additional 64 bytes associated with each memory page. The process of decoding the ECC bits is done when data is read from the storage system. Correcting for errors in the received data is more complicated, however. The first step is to determine if there are any bits in error in the received data. This operation

uses the same algorithm and circuits as the ECC encoding process. As a result, checking for the presence of a data error is fast. If the received data is determined to contain an error, a second algebraic manipulation is necessary to determine which valid code word was most likely sent. The procedure finds the most likely of the 2^k valid code words using calculated Hamming distances. This step is more time consuming, but only needs to be performed when a received data error is detected, which is inherently unlikely. The final step, once the most likely valid code word is found, is to calculate which received data bits are in error and to correct their values. It must be noted that the ECC decoding process can fail if a valid code word cannot be located within the minimum Hamming distance or if the decoder erroneously selects an incorrect code word. ECC failure can occur if the number of bits in error exceeds the designed capabilities of the code. The ECC system must be properly designed so that the probability of ECC failure is virtually zero.

Many modern flash memory system controllers use a block code invented by Hocquenghem, Bose and Ray-Chaudhuri. The acronym BCH comprises the initials of these inventors' names. The binary BCH codes are designed to correct for multiple random bit errors and are relatively efficient in terms of overhead and computational complexity. A t-bit correcting BCH code will detect and correct up to t error bits per code word. The code will detect, but not correct, at least t+1 error bits per code word.

The probability of data corruption can be calculated by summing the probabilities of receiving all error patterns of at least t+1 error bits:

$$P_{corrupt} = \sum_{i=t+1}^{n} P_{fail}(n,i) \qquad \textit{(eqn 1)}$$

The probability term, $P_{fail}(n,i)$, is the probability of receiving "i" bits in error in a code word of "n" bits. The total probability of corrupt data can be approximated by the largest term, the probability of receiving exactly t+1 error bits. Given the ECC parameters (n,k,t), and the probability of a random bit error within the flash memory, P_{raw}, the probability of data corruption can be approximated by:

$$P_{corrupt} \cong P_{fail}(n,t+1) = \binom{n}{t+1} P_{raw}^{\,t+1} \left(1 - P_{raw}\right)^{n-(t+1)} \qquad \textit{(eqn 2)}$$

The formula is for the binomial probability density function, which is used to describe a succession of experiments, each of which has two possible outcomes. Each experiment is independent and has the same probability of outcomes. This function applies to flash memory data corruption when each data bit read is the "experiment," and the outcome is whether the data bit is "good" or "bad." The probability of receiving a bad data bit is the term, P_{raw}, and the parenthetical term is the binomial coefficient, which is the number of ways to arrange "t+1" bad data bits out of "n" bits read from the flash memory.

The probability of corrupted data is expressed as the number of errors per bit read, and is called the application bit error rate. The random bit error probability of a flash memory cell is similarly expressed as the raw bit error rate. Typical raw bit error rates for SLC NAND flash memory range from $1x10^{-9}$ to $1x10^{-11}$ errors per bit read, while typical raw bit error rates for MLC NAND flash memory range from $1x10^{-5}$ to $1x10^{-7}$ errors per bit read. MLC raw bit error rates are higher than SLC raw bit error rates because of the multiple control gate voltage thresholds that must be programmed and detected.

Binary BCH ECCs are becoming widely used in flash memory systems. A critical design issue is the strength of the ECC or how many bits in error can be corrected. Some flash memory systems, for example USB flash drives, do not use a very strong ECC. A typical USB flash drive might employ a simple 1- or 2-bit correcting code. These devices are often used as temporary storage, where the added expense of a powerful ECC system is not justified. SSDs are intended for primary storage, however, and require a powerful error correction system.

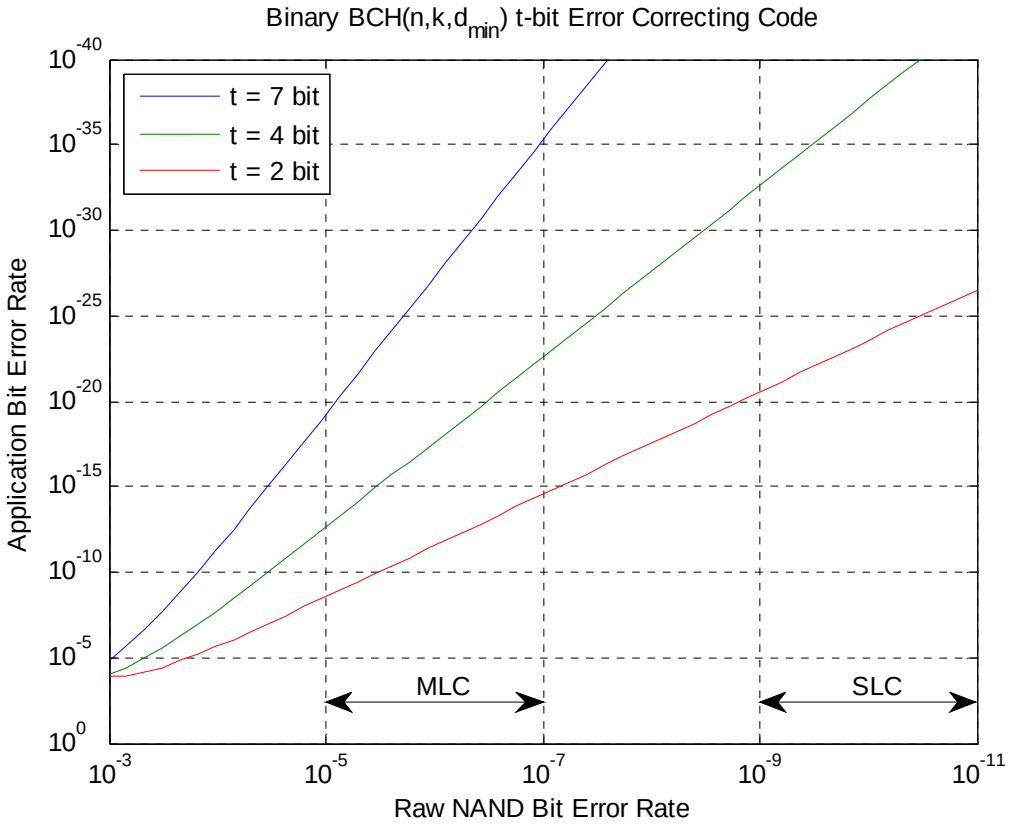

Figure 16 – Application bit error rates vs. raw NAND cell bit error rates
for Binary BCH codes over 512 byte user data sectors
BCH($k = 4096$, $n = k + 13t$, $d_{min} = 2t + 1$)

The ECC parameters for an SSD must be chosen so that the application bit error rate is at least as small as the HDD it is replacing. Modern hard drives typically use a different kind of ECC system, the Reed-Solomon codes. The properties of Reed-Solomon codes make them especially well-suited to applications where errors occur in bursts. Physical defects associated with rotating magnetic media often manifest as streaks making the Reed-Solomon codes a good choice. Typical application bit error rates for hard drives are specified as 1×10^{-15} errors per bit read. A binary BCH code capable of correcting 7 bits in error would deliver outstanding performance for an SSD using either SLC or MLC NAND flash memory.

Wear Leveling

NAND flash memory is susceptible to wear due to repeated program and erased cycles that are commonly done in data storage applications. Constantly programming and erasing to the same memory location eventually wears that portion of memory out and makes it invalid. As an example, systems using File Allocation Tables (FAT) require frequent updating during data write operations, which means many program erase cycles would be required to the same memory location. As a result, the NAND flash would have limited lifetime. To prevent scenarios such as these from occurring, special algorithms are deployed within the SSD called wear leveling. As the terms suggests, wear leveling provides a method for distributing program and erase cycles uniformly throughout all of the memory within the SSD. This prevents continuous program and erase cycles to the same memory location, resulting in greater extended life to the overall NAND flash memory.

Memory Layout

To understand how wear leveling is performed on memory within the SSD flash drive, it is important to understand the architecture layout of the NAND flash. An SSD is made up of many individual NAND flash chips. Each individual NAND flash chip is made up of an array of blocks. Within each block is an array of memory cells called pages (or sectors).

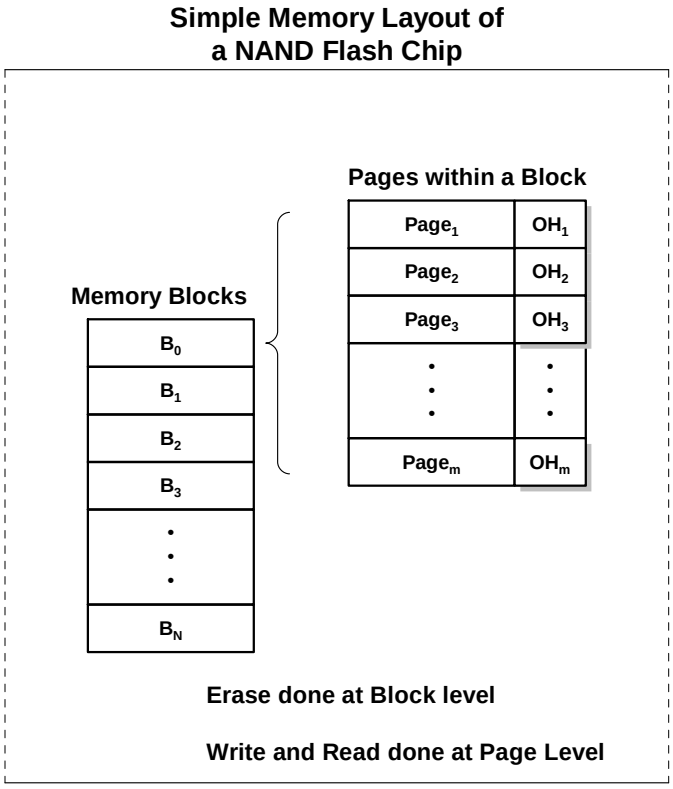

Figure 17 - Simple NAND flash memory layout

Pages are typically about 512 bytes, but can be as large as 2048 bytes. Each page incorporates additional overhead bytes to handle such things as ECC and indexing.

To write data to a portion of memory within the NAND flash, a memory location must first be erased prior to being programmed. The smallest unit that can be erased is a block and the smallest unit that can be programmed or read is a page (or sector).

SSD System Architecture

Before entering a discussion on wear leveling, the basic system architecture for an SSD will be described to help visualize the main components that assist in the wear-leveling algorithm. Below is a simple block diagram of an SSD. Data transferred to and from the SSD passes through a Host interface chip that is configured for different interfaces (PATA, SATA, SCSI, SAS, etc). The host interface is connected to two buses, a system bus used for addressing and control, and a data bus that provides the data path to the NAND flash. On the control bus is the CPU, Flash controller and static random access memory (SRAM). The SRAM is used for tables, CPU scratch pad computing and logical block to physical block address mapping. Since the SRAM is volatile memory, pertinent information, such as tables and logical to physical address mapping, are continually backed up to NAND flash. The CPU is the main controller for the SSD. It provides coordination of writing and reading to and from the flash memory. It also executes and monitors the wear-leveling algorithms used on the flash memory. The flash controller performs the intimate control of addressing, programming, erasing and reading of the flash memory.

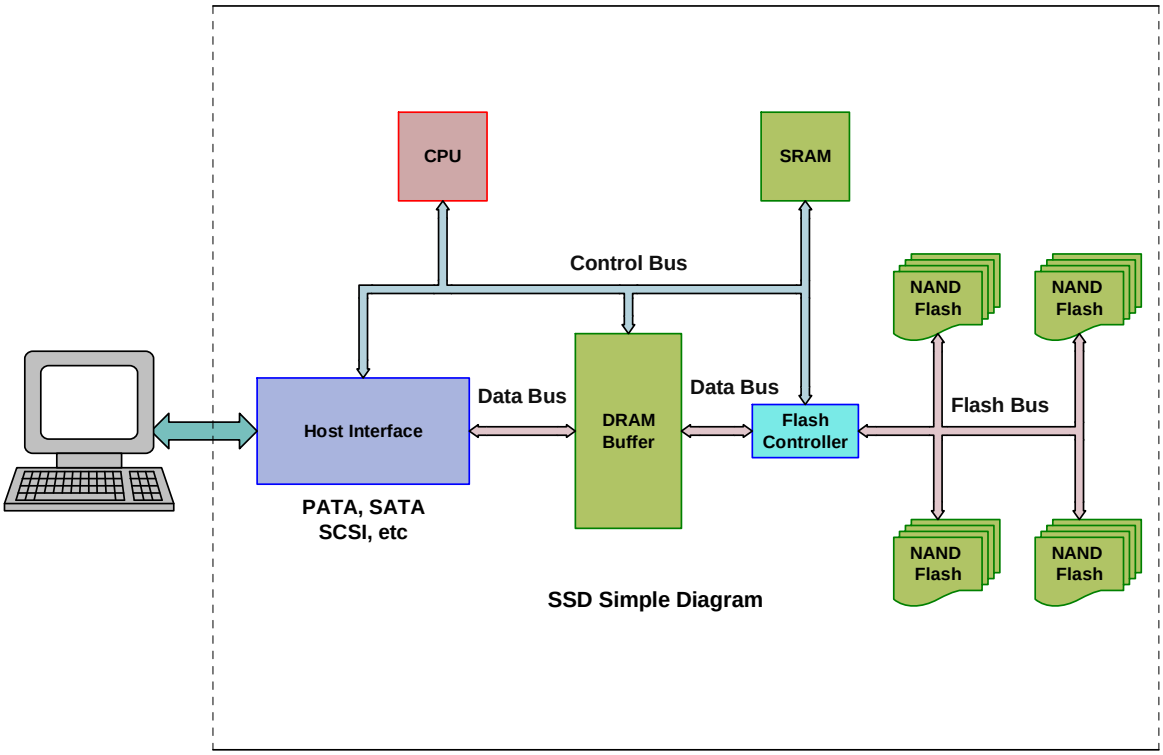

Figure 18 - Simple block diagram of SSD architecture

On the data bus, data received is transferred from the host interface into a synchronous dynamic random access memory (SDRAM) buffer. From there, data is flushed and routed to the flash controller, which writes the data into flash memory. Similarly, a read command along with the logical address is sent to the CPU via the host interface chip. The CPU determines the physical address of memory from the mapping table and sends the information to the Flash controller chip so that the data can be accessed and sent to the host interface. From there the data is sent to the host.

Block Configuration

Blocks are pooled into two categories for wear leveling: data blocks and free blocks. The majority of blocks in memory are data blocks and are subjected to wear leveling and data storage. Free blocks, which are made up of 2 percent of the entire number of blocks in memory, are used to buffer the wear-leveling algorithm. Meaning that blocks will be swapped between the data block pool and the free block pool based on the wear-leveling criteria.

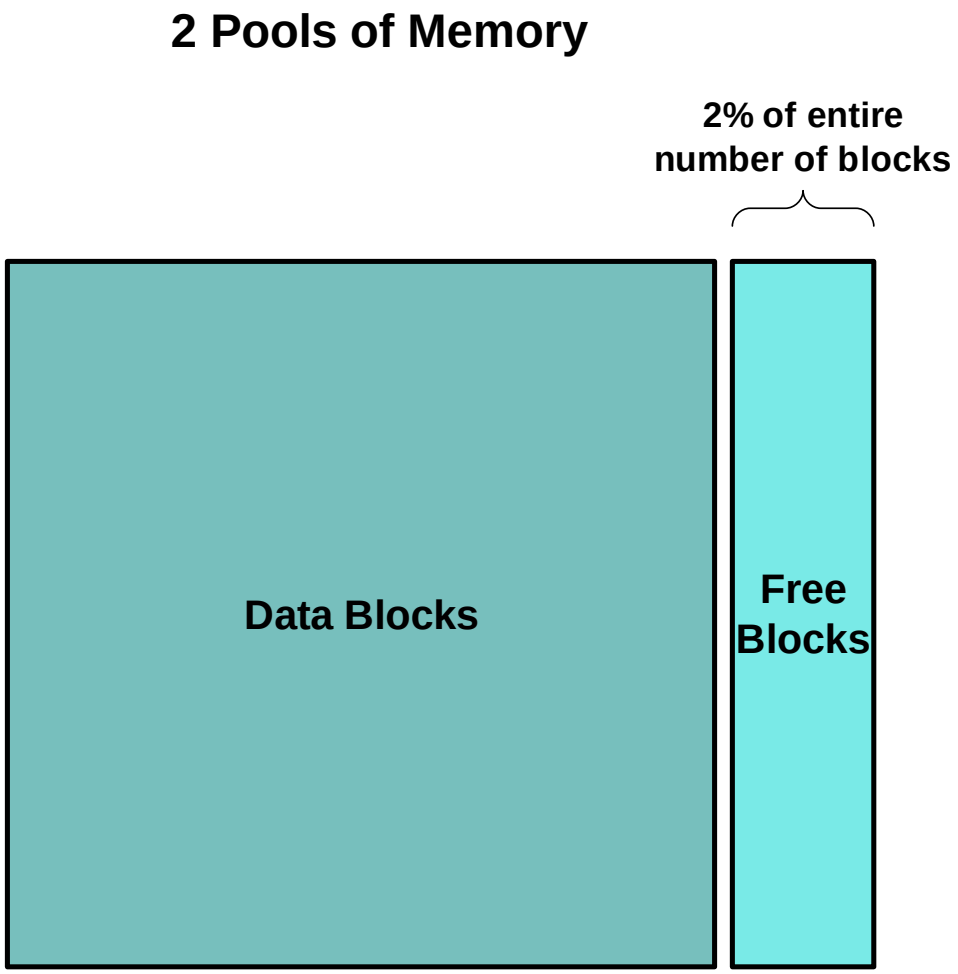

Figure 19 - Blocks are pooled into two categories

Flash Translation Layer (FTL) Management

The FTL provides management of logical to physical address mapping of sector addresses, erase count management for the flash memory and invalid block mapping. Erase count management of each block must be maintained to effectively execute the wear-leveling algorithms. Erase count of every block is maintained in a table called the erase count table (ECT). Each time a block is erased, the ECT is updated by incrementing the erase count value for that block. The wear-leveling algorithms must scan this table to determine which blocks to swap between the data block pool and the free block pool. Because SSDs are typically 32GB or higher, significant time would be required to scan each individual erase count value in the ECT. To reduce this time, the ECT is expanded to a second tier made up of a group of blocks. A total group erase count sum (eqn 3) is calculated for each group. As indicated in eqn 3, the group sum is a summation of the entire individual block erase count values assigned within a group. The size of a group is determined by the number of blocks assigned within a group. The scan process of the ECT scans

the group's total sum erase count, then scans the block erase counts within the group. This method greatly reduces the time required for scanning the ECT.

$$Group\_Sum = \sum_{0}^{N} Block\_ec_n \qquad \textit{(eqn 3)}$$

Where ***N*** = size of Group

When blocks are swapped between the data block pool and the free block pool, the translation table manages the reallocation of logical block addresses to physical block addresses and regroups the new block addresses and erase count values into the their new groups. Once this is complete, the group sum must subtract the departing blocks' erase count value and add in the arriving blocks' erase count value (eqn 4).

$$Group\_Sum = Group\_Sum + \left(Arrival\_Block\_ec - Depart\_Block\_ec\right) \qquad \textit{(eqn 4)}$$

The illustration below shows a simple block diagram of the ECT and translation mapping table. It should be noted that in typical operation the ECT and translation mapping table are managed in SRAM.

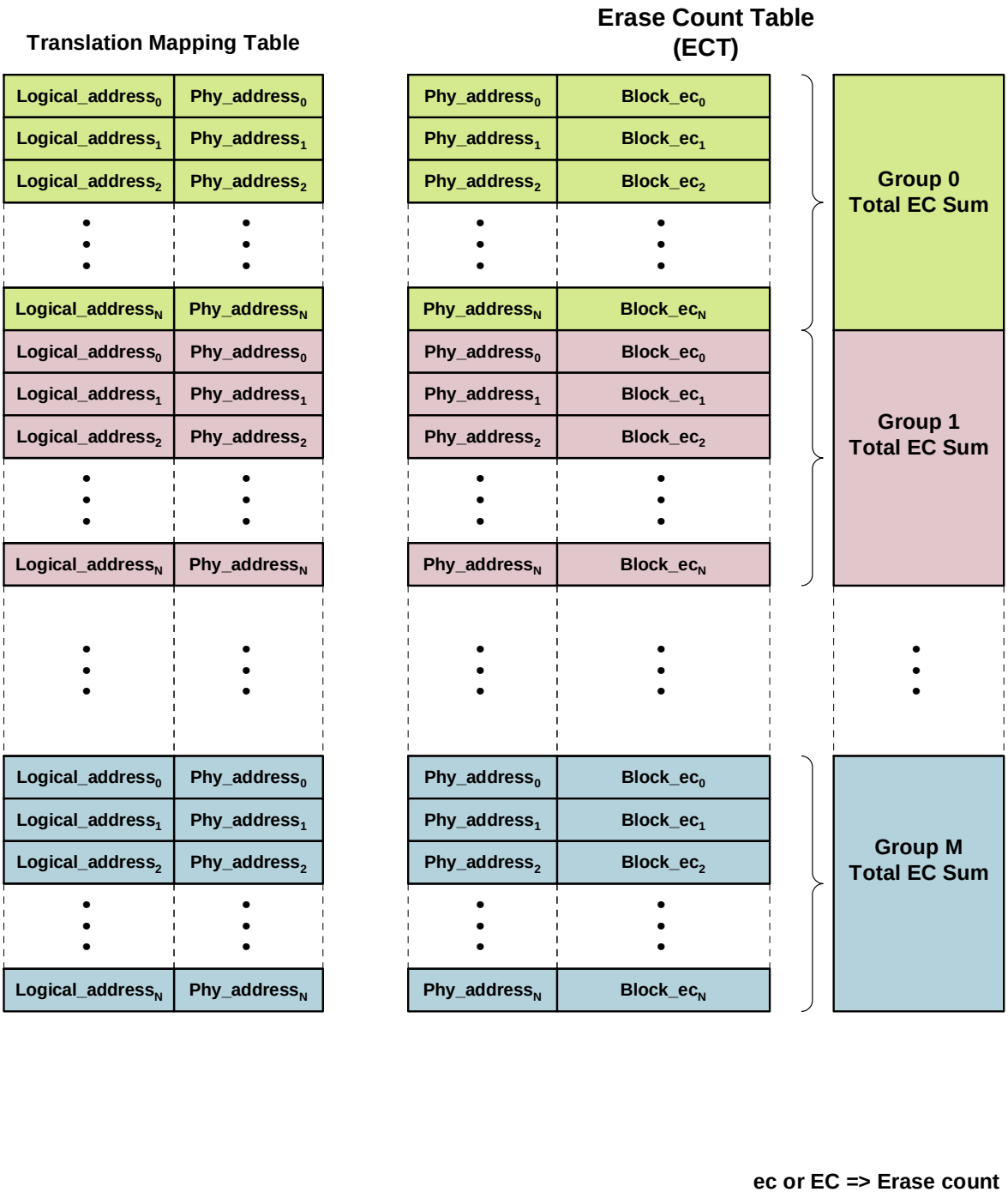

Figure 20 - Erase count table (ECT) and associated translation mapping table

However, SRAM is a volatile memory, so the table must be backed up in NAND flash. On power-up, the tables stored in NAND flash are retrieved and placed into SRAM, where they are managed and updated. At periodic times of low activity or at power-down, the tables are backed-up into NAND flash.

Also within the FTL is a translation table that holds the physical address of each page and its related logical page address. The FTL manages the physical address mapping when data passes from the host interface with a logical

address. The combination of the FTL and an aggressive write buffering scheme using external dynamic random access memory (DRAM) allows the reordering of write requests in the write buffer. This makes it possible to process high volumes of data at one time and augment or block level map the data in the buffer; therefore, data is only committed to flash memory in near block packet sizes. Thus, data transaction time is improved and the need for garbage collection is unnecessary, since aging and fragmentation hardly ever occurs.

When data is requested by the host interface with a logical address, the FTL locates the physical address where the data is located in flash memory. The data is then transferred from flash memory through the read data path of the foreground unit to the host interface.

Dynamic Wear Leveling

There are two levels of wear leveling, dynamic and static. The dynamic wear algorithm guarantees that data program and erase cycles will be evenly distributed throughout all the blocks within the NAND flash. The algorithm is dynamic because it is executed every time the data in the buffer is flushed and written to flash memory. As will be shown, the algorithm eliminates situations where the application repeatedly writes to the same location over and over again until wear out occurs.

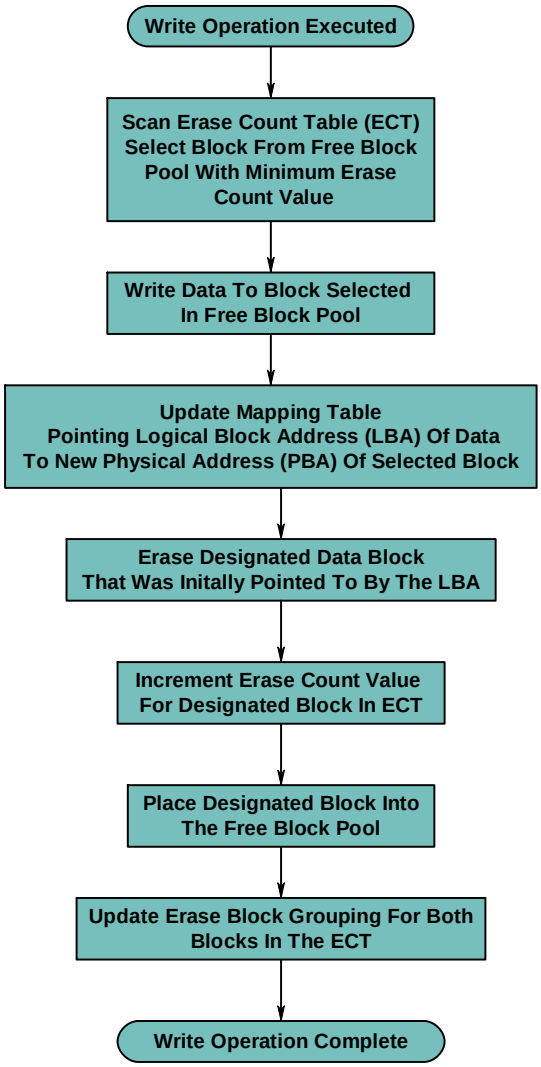

Figure 21 - Simple flow diagram for dynamic wear leveling

When a write command is initiated, the dynamic wear algorithm is executed. The algorithm first scans the ECT within the free block pool, searching for a block with the minimum number of erase counts. Once located, the data is written to that block and is pooled into the data block pool. Since the host writes to a logical block address, the new block's physical address must be mapped to the logical block address within the translation table. This is why continuous writes to the same logical address are never written to the same physical block. The next step is to erase the data block initially mapped with the logical block address and to update the ECT block erase count for that block. Finally, this block is now pooled into the free block pool and the final grouping of the ECT table is updated.

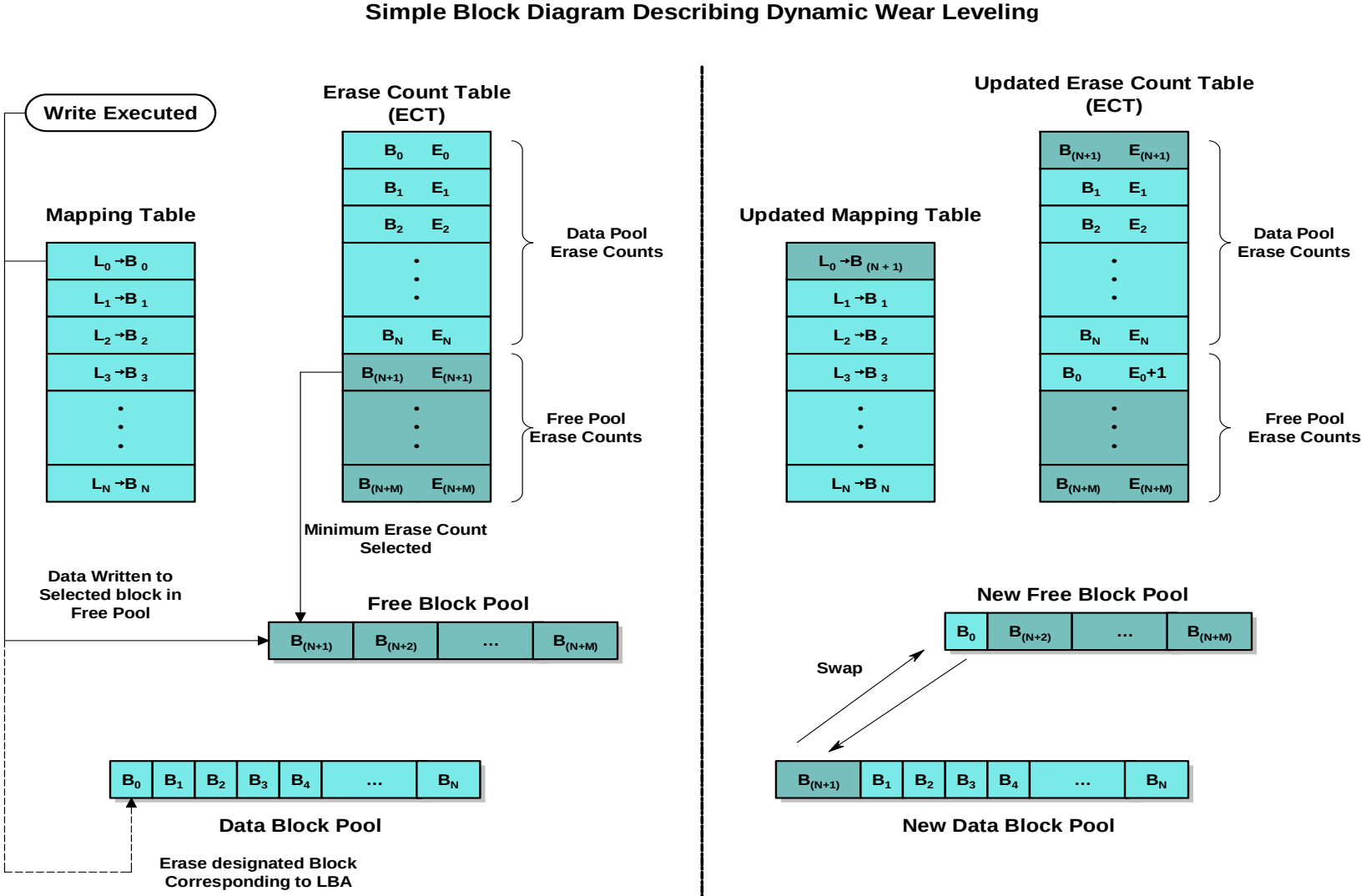

Figure 22 – Dynamic wear leveling block diagram

Static Wear Leveling

Dynamic wear leveling alone cannot insure that all blocks are being wear-leveled at the same rate. There is also the special case when data is written and stored in flash for long periods of time or indefinitely. While other blocks are actively being swapped, erased and pooled, these blocks remain inactive in the wear-leveling process.

To insure that all blocks are being wear-leveled at the same rate, a secondary wear-leveling algorithm called static wear leveling is deployed. Static wear leveling addresses the blocks that are inactive and have data stored in them. Unlike dynamic wear leveling, which is evaluated each time a write flush buffer command is executed, static wear leveling has two trigger mechanisms that are periodically evaluated. The first trigger condition evaluates the idle stage period of inactive blocks. If this period is greater than the set threshold, then a scan of the ECT is initiated.

The scan searches for the minimum erase count block in the data pool and the maximum erase count block in the free pool. Once the scan is complete, the second level of triggering is checked by taking the difference between the maximum erase count block found in the free pool and the minimum erase count block found in the data pool, and checking if that result is greater than a set wear-level threshold. If it is greater, then a block swap is initiated by first writing the content of the minimum erase count block found in the data pool to the maximum erase count block found in the free pool.

Next, each block is re-associated in the opposite pool. The minimum erase count block found in the data pool is erased and placed in the free pool, and the maximum erase count block, which now has the contents of the other block's data, is now associated in the data block pool. With the block swap complete, the re-mapping of the logical block address to the new physical block address is completed in the FTL. Finally, the ECT is updated by associating each block to its new groups. A simple block diagram illustrating static wear-leveling algorithm is shown in Figure 23.

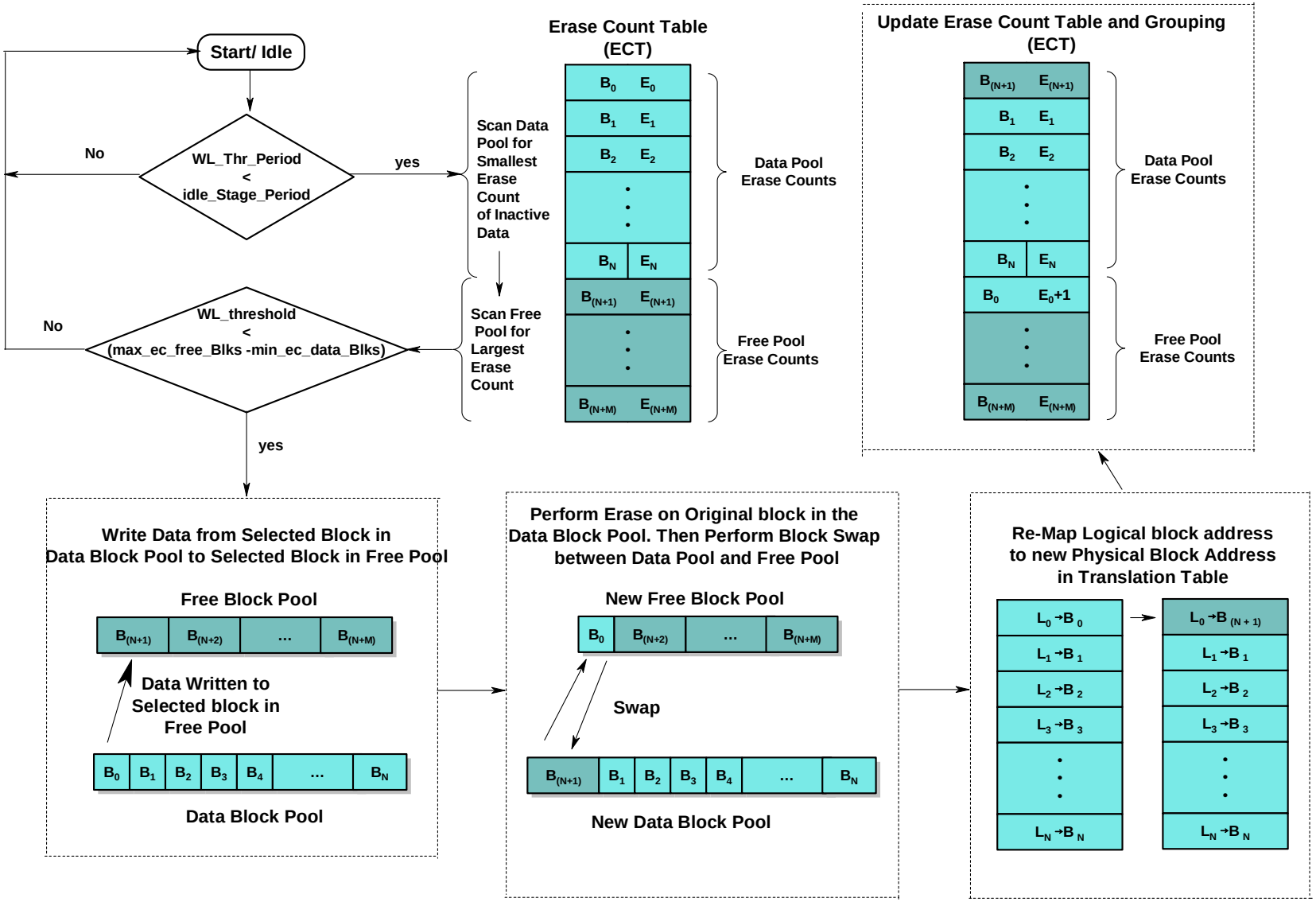

Figure 23 – Static wear leveling block diagram

Invalid Block Handling and Mapping

Invalid blocks are blocks that contain one or more invalid bits whose reliability can no longer be guaranteed. Bad blocks can occur during the manufacturing process or during runtime. NAND flash vendors indicate which blocks are invalid from the manufacturing process by typically providing block status in the first byte of the spare area and writing “00” in the first or second page of each invalid block at the last address column. When the SSD is initially booted up, the drive scans the blocks and maps which blocks are marked invalid in the invalid block table (IBT) in the FTL.

At runtime, errors can occur during erasing and programming. They are detected by monitoring the status register in the NAND flash chip during an erase and program operation. If a hard error occurs during one of these operations, a bit is set in the status register, indicating that the block is no longer reliable and should be marked invalid. When a bad block is detected, the FTL re-writes the block from the buffer to a block with the least amount of erase counts from the free pool and re-maps the logical address to the new physical address of the new block. The block with the detected error is mapped invalid in the IBT and excluded from the free pool. The amount of flash memory in the free pool is determined by the following equation:

Total # of valid flash memory blocks - # of blocks allocated for disk capacity *(eqn 5)*

Typically, this value is usually more than two percent of the total number of valid flash memory blocks. As the number of invalid blocks increases, the number of valid flash memory decreases, reducing the amount of available blocks in the free pool. This technique enhances the usage rate of the overall flash memory as compared to the usual bad block replacement algorithm found in other SSDs, where reserved unused blocks are swapped with invalid blocks.

Life Time Estimation

Program and erase cycles (P/E) of the flash memory are the limiting factor for the overall lifetime of the SSD. Efficient use of the wear-leveling algorithm guarantees equally distributed P/E cycles on each block of flash memory.

Depending on the type of NAND flash memory (SLC or MLC), and the predicted usage per day, one can estimate the overall lifetime of the SSD (sometimes called the write endurance of the SSD). However, it is particularly important to emphasize usage per day. Embedded in the word usage and often overlooked is the type of usage, such as sequentially written, random written or mixed written (random and sequential). Each scenario affects performance (IOPS, transfer rate, block size, etc.) and the overhead written along with the data. These parameters, when accounted for properly, can effect the resulting life time estimation, even when the total perceived usage per day is the same. This is why, when life time estimation is quoted, the vendor indicates the type of written strategy executed, which is usually sequential write. Sequentially written data provides an ideal scenario for estimating lifetime since the IOPS, transfer rate and overhead are constant factors that can be used in a simple formula, as opposed to random or mixed written data, which can vary the overhead size. The calculated lifetime estimation example used in this paper will exercise the sequential written usage type, which is a reasonable estimation for the SSD lifetime, and will give the reader a base of understanding for the predicted value. Keep in mind, the true written usage will vary depending on the type of write strategy used.

Recall for the SLC NAND flash, which is the present technology used today, P/E cycle endurance is approximately 100,000 cycles, as opposed to MLC which is only 10,000 cycles.

In all cases the equation for lifetime is based on the simple product of SSD capacity and the write endurance of the NAND flash memory. This is because the wear-leveling algorithm insures the wear rate for each block of flash

memory to be essentially equal throughout the entire lifetime of the drive. The plot below shows the wear leveling result after constantly writing to a 32GB Imation SSD for 170 hours. Note each block has about the same number of erase counts.

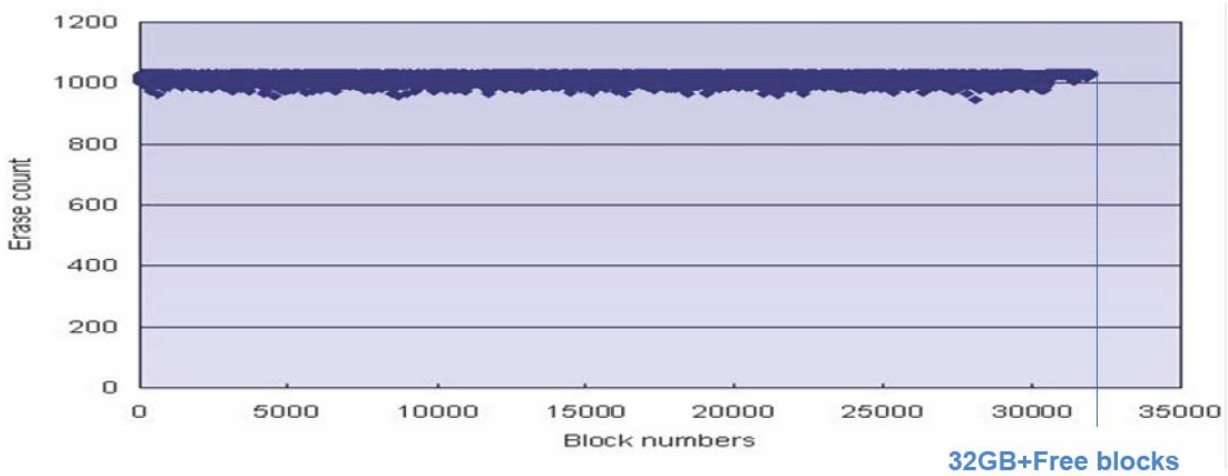

Figure 24 – Effects of wear leveling algorithms

The wear-leveling algorithm allows the lifetime calculation to be simply described as the product of the SSD capacity and the P/E cycle endurance of the flash memory divided by the usage per day. Additional factors such as percent utilization can be used as a safety net to indicate when wear out is imminent, and a capacitor rate factor is added to the equation to indicate the written overhead accompanying the data. The capacity rate factor, as described earlier, is constant for the sequentially written usage strategy, but can vary when alternative write strategies are used, causing the overhead to vary in size.

Parameters:

- NAND flash P/E cycle endurance
  - SLC 100,000 cycles, MLC 10,000 cycles
- NAND flash P/E utilization
  - The allowable amount of P/E utilization before flash memory is considered worn out – 95%
- SSD capacity
  - Total capacity of the SSD
- Capacity rate factor – The amount of written overhead accompanying the data
  - Sequential write has a constant value of 1.1
  - Mixed written data has a value that varies depending on the percent mixture of sequential and random written data

$$Lifetime(years) = \frac{(SSD\_Capacity(GB))(P/E)(Percent\_Utilization)}{(Usage/Day)(Capacity\_Rate)(365\,days/year)} \qquad \textit{(eqn 6)}$$

The table below shows the resulting lifetime of a 32GB drive using sequentially written data.

Usage (GB/day)	Lifetime of a Sequential Written data Years
3.2	2366.1
10	757.2
32	236.6
100	75.7
320	23.7

Table 25 – 32GB SSD lifetime

Alan Olson is an electrical engineering specialist at Imation's New Product Development Laboratory; Denis Langlois is senior design specialist at Imation's Advanced Tape and Systems Laboratory. Both laboratories are located at Imation's worldwide headquarters in Oakdale, Minn.