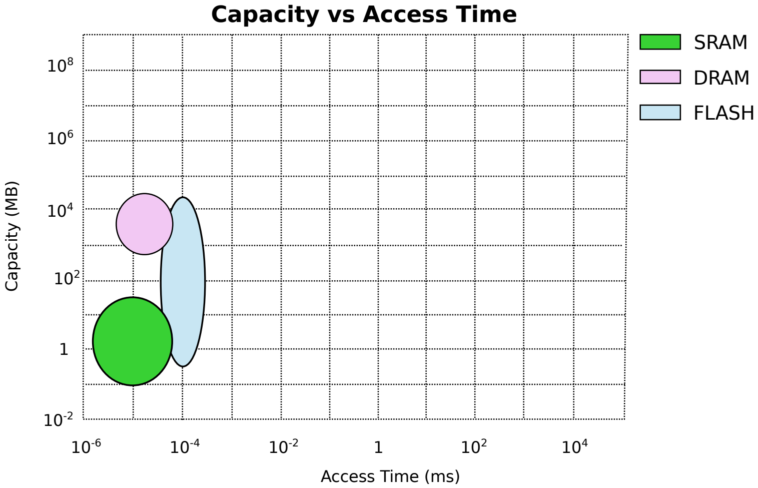
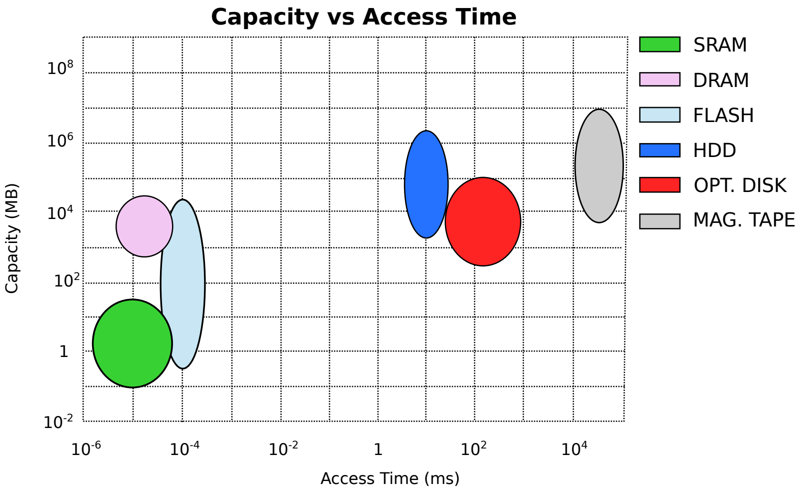
An overview on solid-state-drives architectures and enterprise solutions

Romolo Marotta

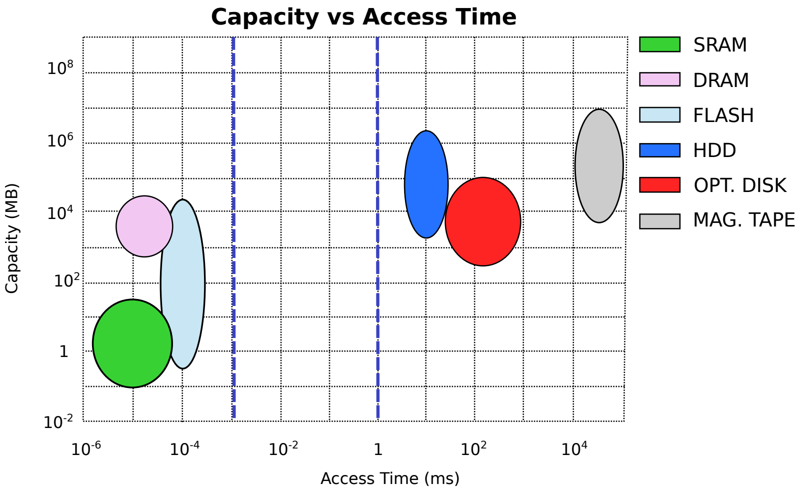
Why are SSDs so attractive?



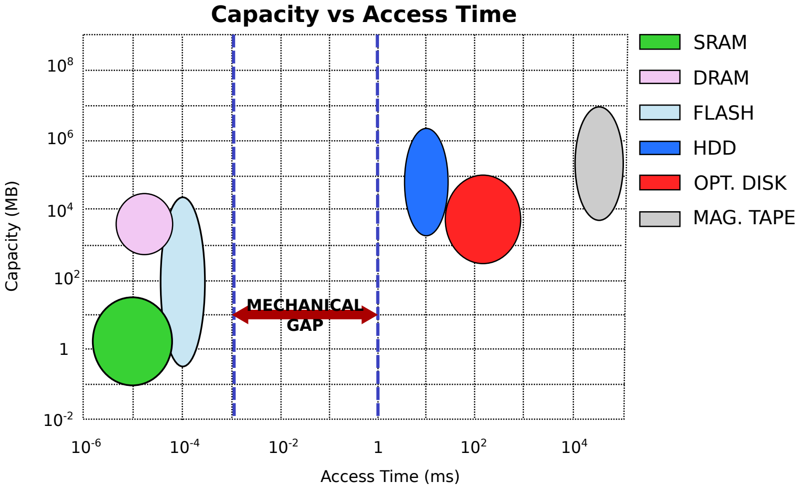
Why are SSDs so attractive?



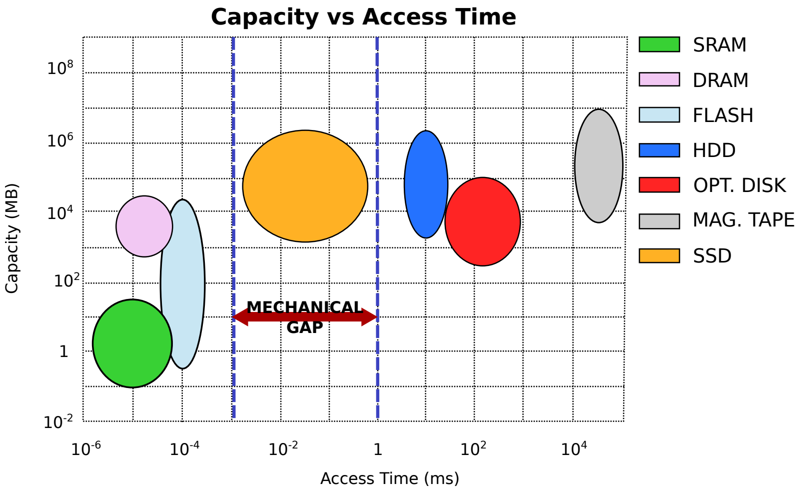
Why are SSDs so attractive?



Why are SSDs so attractive?

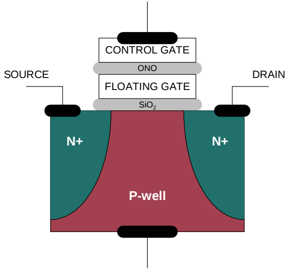


Why are SSDs so attractive?



What is a flash memory cell?

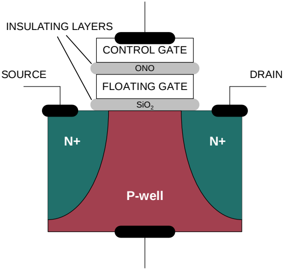
- Flash memory was invented by Dr. Fujio Masuoka in 1984
- The name "Flash" was adopted because the process of erasing the memory contents reminded him of the flash of a camera
- A Flash memory cell is a Floating Gate Metal-Oxide-Semiconductor Field-Effect Transistor (MOSFET)



What is a flash memory cell?

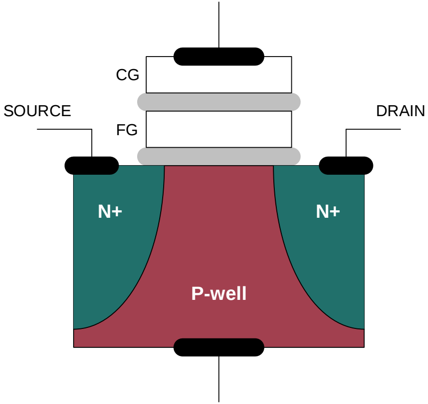
Anatomy

- The N+ region is a silicon lattice with phosphorus impurities, creating an **excess of electrons**
- The P- region is a silicon lattice with boron impurities, creating an **absence of electrons**
- The floating gate is surrounded by insulating layers



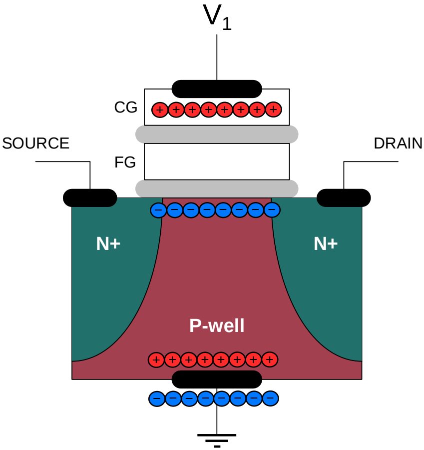
How does a flash memory cell work?

- The Source (S) and Drain (D) are disconnected, thus a current cannot flow between them



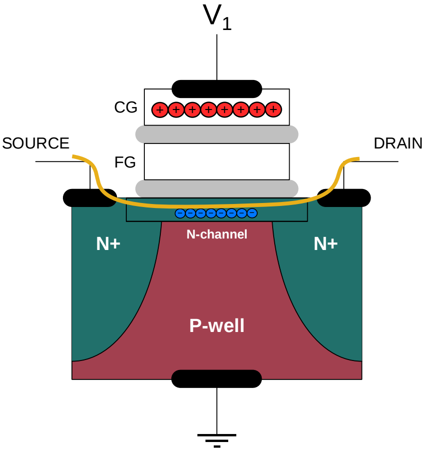
How does a flash memory cell work?

- The Source (S) and Drain (D) are disconnected, thus a current cannot flow between them
- Applying a voltage between Control Gate (CG) and Body (P-well) creates a concentration of electrons between S and D



How does a flash memory cell work?

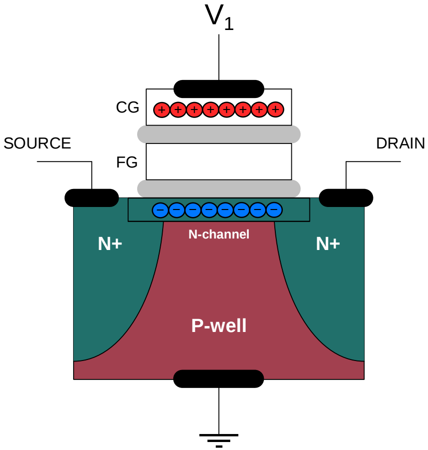
- The Source (S) and Drain (D) are disconnected, thus a current cannot flow between them
- Applying a voltage between Control Gate (CG) and Body (P-well) creates a concentration of electrons between S and D
- If the voltage if high enough (V_1), it creates a channel between S and D which allows the current I_D to flow between them.



How does a flash memory cell work?

Program operation

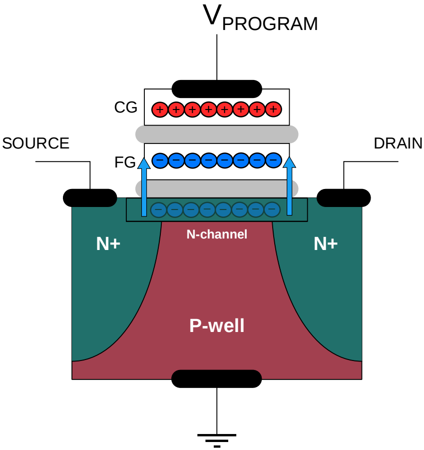
- Applying an appropriate voltage ($V_{PROGRAM}$) to CG, the electrons will be trapped in FG



How does a flash memory cell work?

Program operation

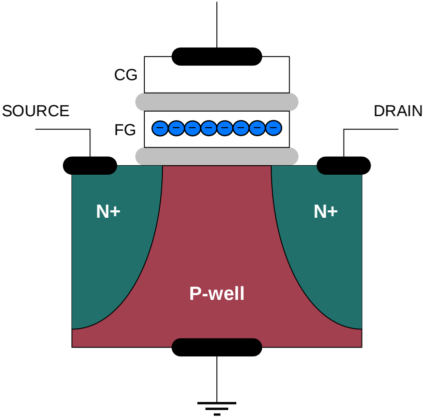
- Applying an appropriate voltage ($V_{PROGRAM}$) to CG, the electrons will be trapped in FG



How does a flash memory cell work?

Program operation

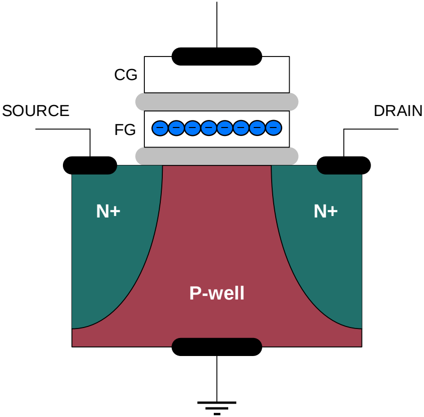
- Applying an appropriate voltage ($V_{PROGRAM}$) to CG, the electrons will be trapped in FG
- those electrons are kept in FG, although there is no tension on CG



How does a flash memory cell work?

Program operation

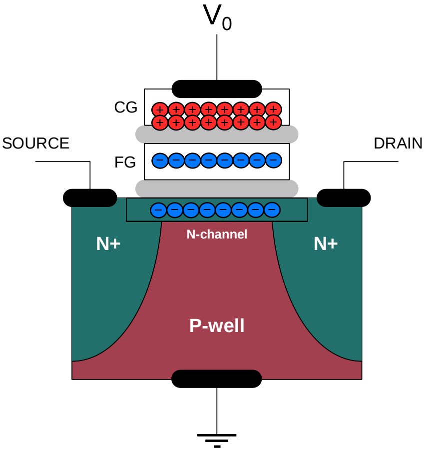
- Applying an appropriate voltage ($V_{PROGRAM}$) to CG, the electrons will be trapped in FG
- those electrons are kept in FG, although there is no tension on CG
- we call this state “0”



How does a flash memory cell work?

Program operation

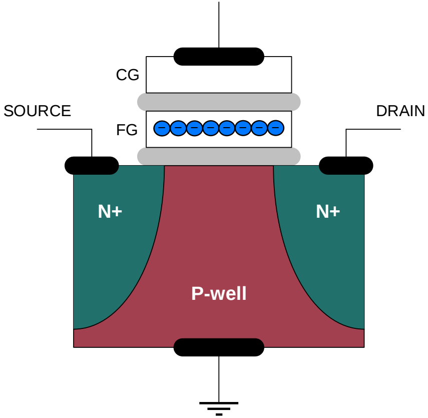
- Applying an appropriate voltage ($V_{PROGRAM}$) to CG, the electrons will be trapped in FG
- those electrons are kept in FG, although there is no tension on CG
- we call this state "0"
- In state 0, a voltage $V_0 > V_1$ is required in order to establish the N-channel



How does a flash memory cell work?

Erase operation

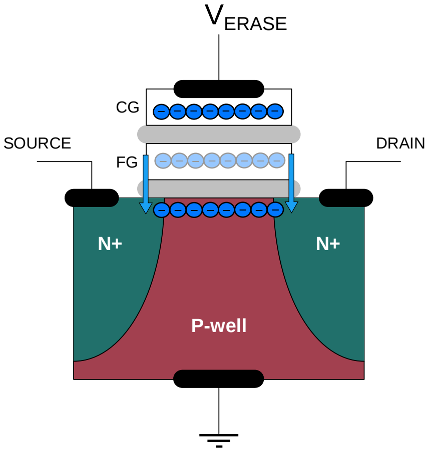
- In state 0, electrons are stored in FG



How does a flash memory cell work?

Erase operation

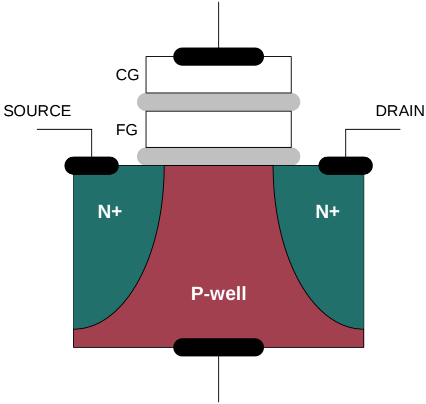
- In state 0, electrons are stored in FG
- a voltage (V_{ERASE}) is required in order to remove them from FG



How does a flash memory cell work?

Erase operation

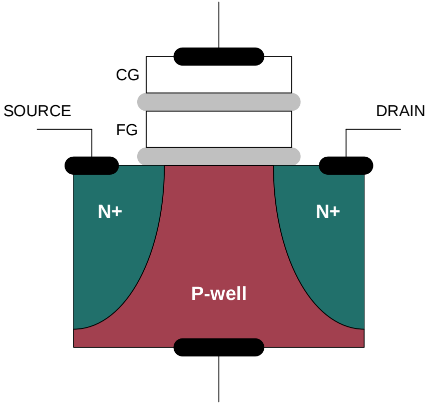
- In state 0, electrons are stored in FG
- a voltage (V_{ERASE}) is required in order to remove them from FG
- at this point no charges are on FG



How does a flash memory cell work?

Erase operation

- In state 0, electrons are stored in FG
- a voltage (V_{ERASE}) is required in order to remove them from FG
- at this point no charges are on FG
- we call this state "1"



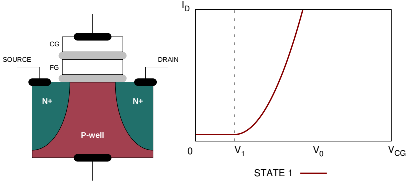
How does a flash memory cell work?

States of a Single-Level Cell

- The two states allows a cell to store a bit

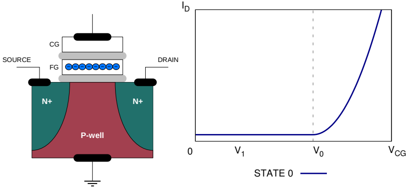
STATE 1:

- No charges in FG
- required a $V_{CG} > V_1$ in order to set-up the N-Channel $\Rightarrow I_D > 0$



STATE 0:

- Charges are in FG
- required a $V_{CG} > V_0 > V_1$ in order to set-up the N-Channel $\Rightarrow I_D > 0$

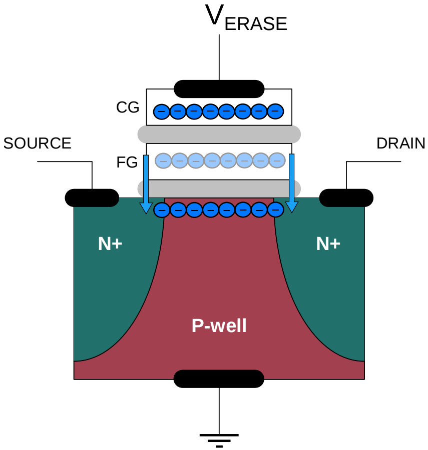


Since the cell stores ONE bit, it is called Single-Level Cell (SLC)

How does a flash memory cell work?

How to write a cell?

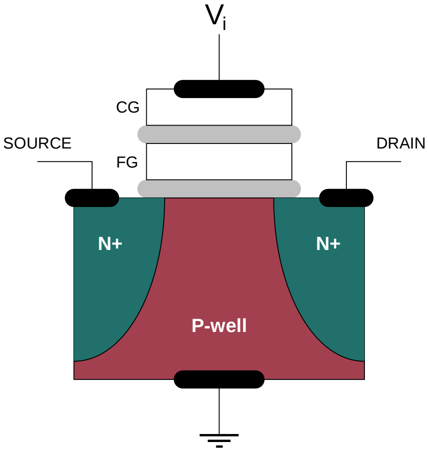
- Erase the cell



How does a flash memory cell work?

How to write a cell?

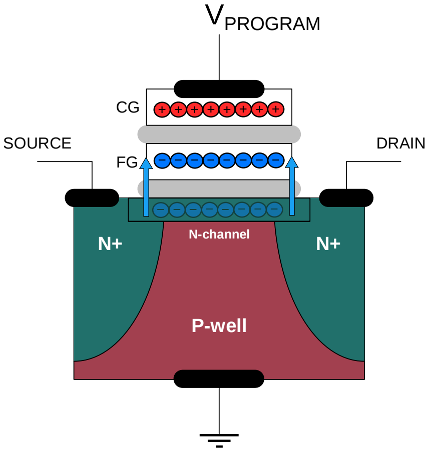
- Erase the cell
- To write 1: it is done



How does a flash memory cell work?

How to write a cell?

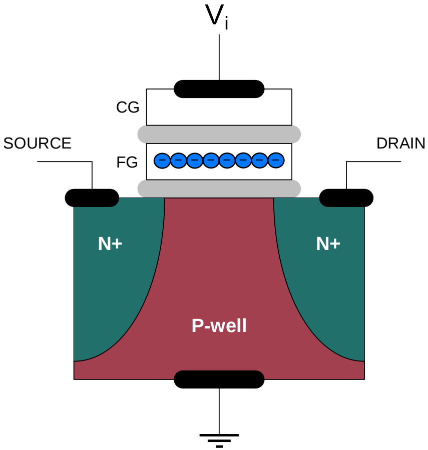
- Erase the cell
- To write 1: it is done
- To write 0: program the cell



How does a flash memory cell work?

How to write a cell?

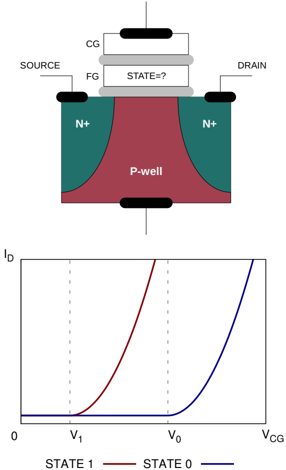
- Erase the cell
- To write 1: it is done
- To write 0: program the cell



How does a flash memory cell work?

How to read a cell?

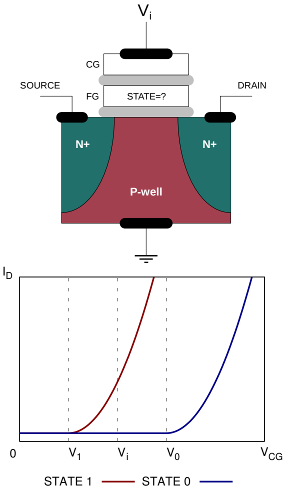
- Reading a cell consists in inferring on the cell state



How does a flash memory cell work?

How to read a cell?

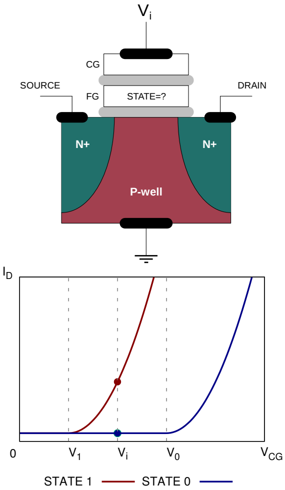
- Reading a cell consists in inferring on the cell state
- Apply an intermediate voltage $V_1 < V_i < V_0$ on CG



How does a flash memory cell work?

How to read a cell?

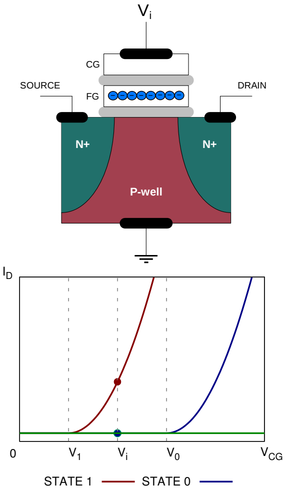
- Reading a cell consists in inferring on the cell state
- Apply an intermediate voltage $V_1 < V_i < V_0$ on CG
- Read the actual value I_D^* of the current I_D



How does a flash memory cell work?

How to read a cell?

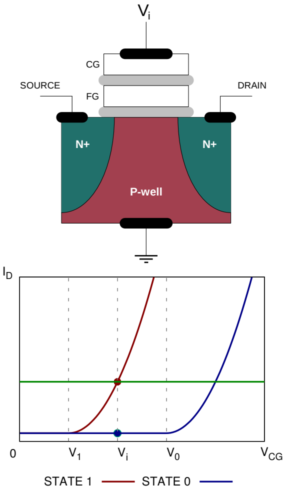
- Reading a cell consists in inferring on the cell state
- Apply an intermediate voltage $V_1 < V_i < V_0$ on CG
- Read the actual value I_D^* of the current I_D
- If $I_D^* = 0$ the bit value is 0



How does a flash memory cell work?

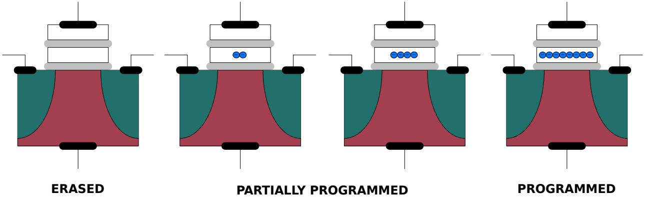
How to read a cell?

- Reading a cell consists in inferring on the cell state
- Apply an intermediate voltage $V_1 < V_i < V_0$ on CG
- Read the actual value I_D^* of the current I_D
- If $I_D^* = 0$ the bit value is 0
- If $I_D^* \neq 0$ the bit value is 1



Multi-Level Cell

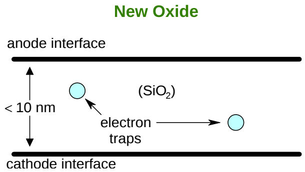
- If we partially charge FG, we need a lower threshold voltage for creating a channel
- We store 2 bits by using 1 programmed state, 2 partially programmed states and 1 erased state
- A flash cell storing multiple bits is a Multi-Level Cell (MLC)
- A Triple-Level Cell (TLC) stores 3 bits



Why does a flash cell deteriorate in time?

- writing a flash cell involves an erase and a program

⇒ electrons move from/into FG



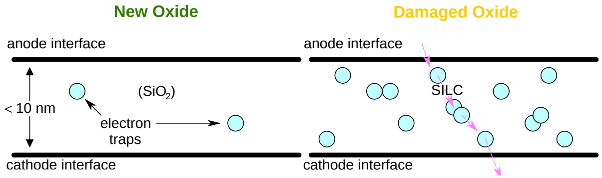
Why does a flash cell deteriorate in time?

- writing a flash cell involves an erase and a program

⇒ electrons move from/into FG

- electrons collide with and damage the insulating layer creating traps

⇒ a Stress Induced Leakage Current (SILC) can flow through these traps



Why does a flash cell deteriorate in time?

- writing a flash cell involves an erase and a program

⇒ electrons move from/into FG

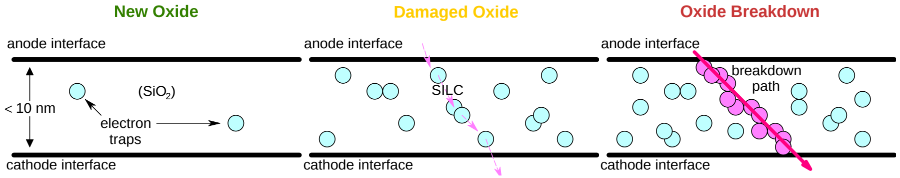
- electrons collide with and damage the insulating layer creating traps

⇒ a Stress Induced Leakage Current (SILC) can flow through these traps

- a lot of traps can build a path from the body to FG

⇒ electrons can flow through that path ⇒impossibility to program

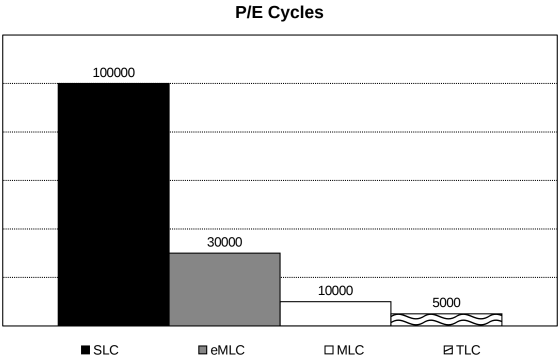
⇒ the flash cell is unusable



Why does a flash cell deteriorate in time?

PE-Cycles

- A flash cell can be programmed and erased a limited number of times before a breakdown ⇒ This number is called P/E-Cycles
- Vendors design firmware capable of recompute the voltage thresholds for read/write operations ⇒ enterprise-MLC (eMLC)



MLC vs SLC

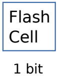
SLC:

- lower density
- higher cost
- faster write
- faster read
- higher endurance

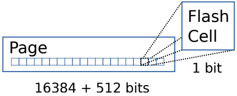
MLC:

- higher density
- lower cost
- erase time is similar to SLC
- the level of charges in FG has to be set carefully $\Rightarrow$ slower program $\Rightarrow$ slower write
- state is not $0/1$ $\Rightarrow$ slower read
- eMLC has 3x shorter endurance
- MLC has 10x shorter endurance
- TLC has 20x shorter endurance

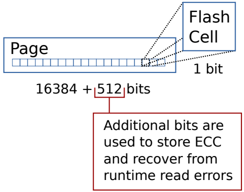
How are flash chips organized?



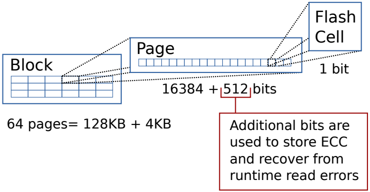
How are flash chips organized?



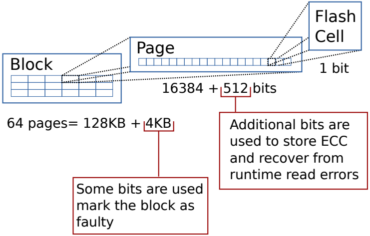
How are flash chips organized?



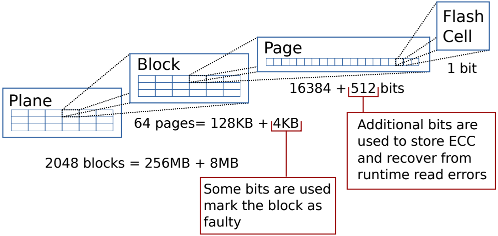
How are flash chips organized?



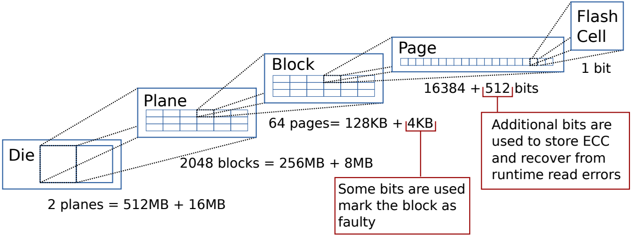
How are flash chips organized?



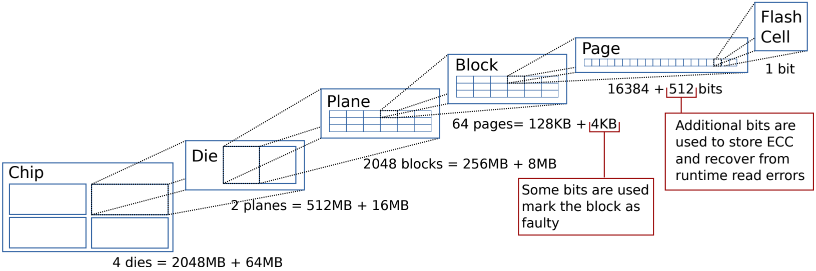
How are flash chips organized?



How are flash chips organized?



How are flash chips organized?

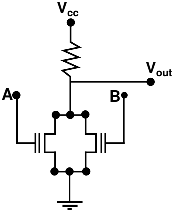


How are flash cells organized?

- Flash cells are connected forming an array called string
- According to the strategies used to connect multiple cells, we can distinguish at least two kind of configuration:

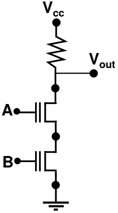
NOR

Flash cells are connected in parallel, resembling a NOR gate



NAND

Flash cells are connected in series, resembling a NAND gate



How are flash cells organized?

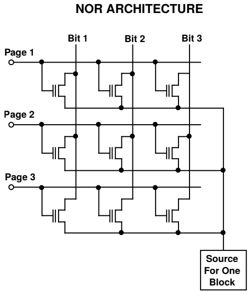
- Let F be the CG side length

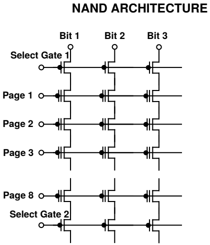
NOR:

- occupies area $10F^2$
- read/write a **single cell**

NAND:

- occupies area $4F^2$
- read/write a **single page**
- erase a **single block**





NOR vs NAND

NOR:

- fast random-byte read
- slower page read
- slower write
- lower density

⇒ good for source code

NAND:

- no random-byte read
- slow partial page read when supported
- faster page read
- faster page write
- higher density

⇒ good for storage

We focus on NAND flash technology

How is a page written?

- Write-in-place strategy:
  1. read the block
  2. erase the block
  3. program the block with the updated page

How is a page written?

- Write-in-place strategy:
  1. read the block
  2. erase the block
  3. program the block with the updated page
- 1 page write = N page read + 1 block erase + N page write (N = number of pages in a block)

⇒ very slow write

How is a page written?

- Write-in-place strategy:
  1. read the block
  2. erase the block
  3. program the block with the updated page
- 1 page write = N page read + 1 block erase + N page write (N = number of pages in a block)

⇒ very slow write

⇒ If we update the page 40 times per second (every 25ms), the block is completely broken in:

- $SLC = \frac{PECycles}{UpdateRate} = \frac{10^5}{40ps} \approx 2500s \approx 40m$
- $MLC = \frac{10^4}{40ps} \approx 4m$
- $TLC = \frac{5 \cdot 10^3}{40ps} \approx 2m$

How is a page written?

- Write-in-place strategy:
  1. read the block
  2. erase the block
  3. program the block with the updated page
- 1 page write = N page read + 1 block erase + N page write (N = number of pages in a block)

⇒ very slow write

⇒ If we update the page 40 times per second (every 25ms), the block is completely broken in:

  - $SLC = \frac{PECycles}{UpdateRate} = \frac{10^5}{40ps} \approx 2500s \approx 40m$
  - $MLC = \frac{10^4}{40ps} \approx 4m$
  - $TLC = \frac{5 \cdot 10^3}{40ps} \approx 2m$

ALERT!

In our example the write rate is $80KBps$

Write Amplification

- Write amplification occurs when 1 user page write leads to multiple flash writes
- Write amplification make flash blocks deteriorate faster
- Let F be the number of flash writes corresponding to U user writes

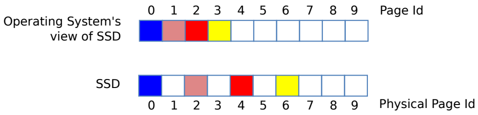
⇒ The write amplification A is:

$$A = \frac{F + U}{U} = 1 + \frac{F}{U} = 1 + A_f$$

where A_f is the write amplification factor

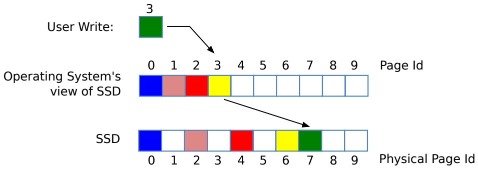
How is a page written? Relocation-on-write

- Write-in-place is inadequate in terms of reliability and performance ($A_f \approx \#$ number of pages in a block)



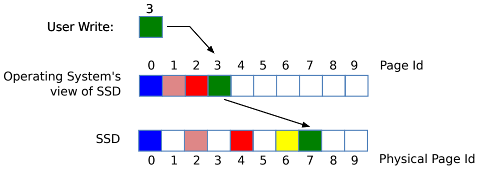
How is a page written? Relocation-on-write

- Write-in-place is inadequate in terms of reliability and performance ($A_f \approx \#$ number of pages in a block)
- Updated pages are re-written on new locations



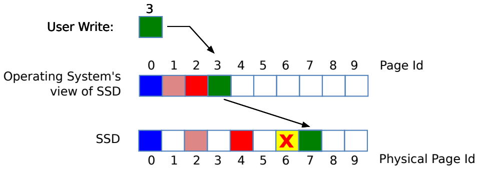
How is a page written? Relocation-on-write

- Write-in-place is inadequate in terms of reliability and performance ($A_f \approx \#$ number of pages in a block)
- Updated pages are re-written on new locations
- The logical address of the update page is mapped to a different physical page



How is a page written? Relocation-on-write

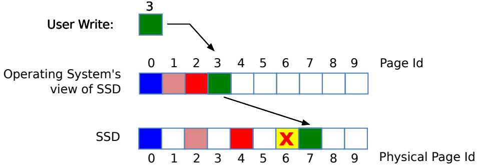
- Write-in-place is inadequate in terms of reliability and performance ($A_f \approx \#$ number of pages in a block)
- Updated pages are re-written on new locations
- The logical address of the update page is mapped to a different physical page
- Previous pages are invalidated



How is a page written? Relocation-on-write

- Write-in-place is inadequate in terms of reliability and performance ($A_f \approx \#$ number of pages in a block)
- Updated pages are re-written on new locations
- The logical address of the update page is mapped to a different physical page
- Previous pages are invalidated

$\Rightarrow$ 1 user page write = 1 page read (obtain an empty page) + 2 page write (update data + invalidate page) $\Rightarrow$ faster write



Flash Translation Layer

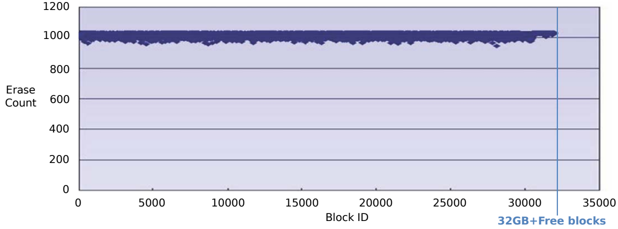
- Assign Logical Addresses to pages
- Store the association between physical and logical addresses in a Translation Mapping Table
- Store the number of erase operation performed on physical pages in a Erase Count Table
- Tables are:
  - maintained in SRAM (high efficient) at runtime
  - stored on flash during shutdown to ensure durability
  - loaded at boot-up

Wear-Leveling

- Free pages for relocation can be retrieved from the whole SSD
- Wear-leveling guarantees that the number of PE-Cycles is uniformly distributed among all blocks

⇒ Wear-leveling extends the time to live of each block and the whole SSD

- Thanks to wear-leveling all blocks break at the same time



Wear-Leveling

- In order to guarantee that enough free pages are available for write relocation, wear-leveling needs:
  - **Over-provisioning** - keep free a percentage of raw capacity
  - **Garbage collection** - keep invalid pages in the same block
  - **DRAM buffers** - keep valid pages in a buffer in order to write full blocks and reduce fragmentation
- We can distinguish at least two kind of wear-leveling algorithms:
  - Dynamic wear-leveling
  - Static wear-leveling

Wear-Leveling

Dynamic Algorithm

- It is called dynamic, because it is executed every time the OS replace a block of data
- A small percentage (e.g. 2%) of raw capacity is reserved as free-block pool
- It chooses from the free pool the block with minimum erase count the buffer is flushed
- The replaced block is erased and added to the free pool

$\Rightarrow$ Only frequently-updated blocks are consumed

Wear-Leveling

Static Algorithm

- Periodically scan the metadata of each block
- Individuate inactive data blocks with lower erase count than free blocks
- Copy their content into free-blocks and exchange them

$\Rightarrow$ this guarantees that static blocks participate to wear leveling

Wear-Leveling

Impact on reliability

At first approximation, wear-leveling eliminate write amplification generated by different sizes of erase and write units

⇒ The block time to fault is:

$$Block_{TTF} \approx \frac{N_{die} \cdot N_{planes} \cdot N_{blocks} \cdot N \cdot PECycles}{PageWriteRate}$$

Wear-Leveling

Impact on reliability

At first approximation, wear-leveling eliminate write amplification generated by different sizes of erase and write units

⇒ The block time to fault is:

$$Block_{TTF} \approx \frac{N_{die} \cdot N_{planes} \cdot N_{blocks} \cdot N \cdot PECycles}{PageWriteRate}$$

$$Block_{TTF} \approx \frac{N_{die} \cdot N_{planes} \cdot N_{blocks} \cdot N \cdot PageSize \cdot PECycles}{PageWriteRate \cdot PageSize}$$

Wear-Leveling

Impact on reliability

At first approximation, wear-leveling eliminate write amplification generated by different sizes of erase and write units

⇒ The block time to fault is:

$$Block_{TTF} \approx \frac{N_{die} \cdot N_{planes} \cdot N_{blocks} \cdot N \cdot PECycles}{PageWriteRate}$$

$$Block_{TTF} \approx \frac{N_{die} \cdot N_{planes} \cdot N_{blocks} \cdot N \cdot PageSize \cdot PECycles}{PageWriteRate \cdot PageSize}$$

$$Block_{TTF} \approx \frac{Capacity_{SSD} \cdot PECycles}{WriteRate}$$

Wear-Leveling

Impact on reliability

At first approximation, wear-leveling eliminate write amplification generated by different sizes of erase and write units

⇒ The block time to fault is:

$$Block_{TTF} \approx \frac{N_{die} \cdot N_{planes} \cdot N_{blocks} \cdot N \cdot PECycles}{PageWriteRate}$$

$$Block_{TTF} \approx \frac{N_{die} \cdot N_{planes} \cdot N_{blocks} \cdot N \cdot PageSize \cdot PECycles}{PageWriteRate \cdot PageSize}$$

$$Block_{TTF} \approx \frac{Capacity_{SSD} \cdot PECycles}{WriteRate}$$

- Blocks deteriorate uniformly, thus:

$$Block_{TTF} \approx SSD_{TTF}$$

Wear-Leveling

Example

- Take a SSD with capacity C and a write rate W
- According to the flash cells used, we have different time to fault:

Wear-Leveling

Example

- Take a SSD with capacity C and a write rate W
- According to the flash cells used, we have different time to fault:

  - $C = 4GB, W = 80KBps$
    - SLC $\Rightarrow SSD_{TTF} = \frac{C \cdot PECycles_{SLC}}{W} = \frac{4GB \cdot 10^5}{80KBps} \approx 158 years$
    - MLC $\Rightarrow SSD_{TTF} = \frac{C \cdot PECycles_{MLC}}{W} \approx 15.8 years$
    - TLC $\Rightarrow SSD_{TTF} = \frac{C \cdot PECycles_{TLC}}{W} \approx 7.9 years$

Wear-Leveling

Example

- Take a SSD with capacity C and a write rate W
- According to the flash cells used, we have different time to fault:

- $C = 4GB, W = 80KBps$
  - SLC $\Rightarrow SSD_{TTF} = \frac{C \cdot PECycles_{SLC}}{W} = \frac{4GB \cdot 10^5}{80KBps} \approx 158years$
  - MLC $\Rightarrow SSD_{TTF} = \frac{C \cdot PECycles_{MLC}}{W} \approx 15.8years$
  - TLC $\Rightarrow SSD_{TTF} = \frac{C \cdot PECycles_{TLC}}{W} \approx 7.9years$

- $C = 128GB, W = 4MBps$
  - SLC $\Rightarrow SSD_{TTF} = \frac{C \cdot PECycles_{SLC}}{W} = \frac{128GB \cdot 10^5}{4MBps} \approx 101years$
  - MLC $\Rightarrow SSD_{TTF} = \frac{C \cdot PECycles_{MLC}}{W} \approx 10years$
  - TLC $\Rightarrow SSD_{TTF} = \frac{C \cdot PECycles_{TLC}}{W} \approx 5years$

Wear-Leveling

Impact on Reliability 2

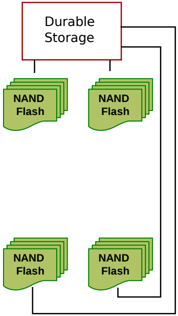
As said before wear leveling make flash blocks deteriorate uniformly. Anyhow

- Garbage collection increase the number of flash write
- Static Wear-leveling increases the number of flash write

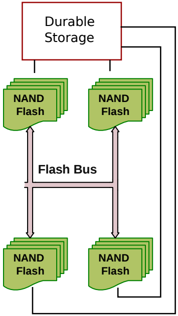
$\Rightarrow$ re-introduce write amplification factor

$$SSD_{TTF} \approx \frac{Capacity_{SSD} \cdot PECycles}{(1 + A_f) WriteRate}$$

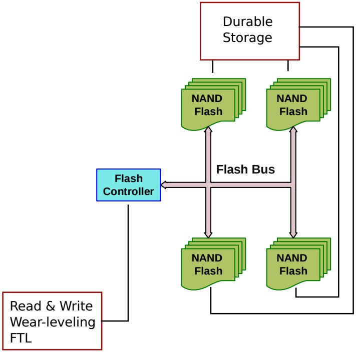
SSD Architecture



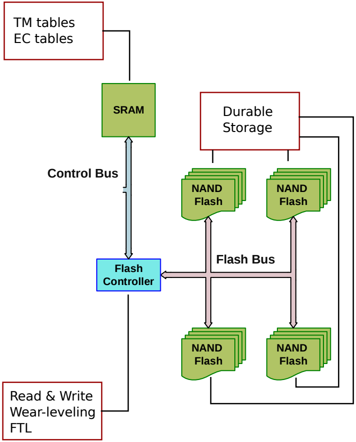
SSD Architecture



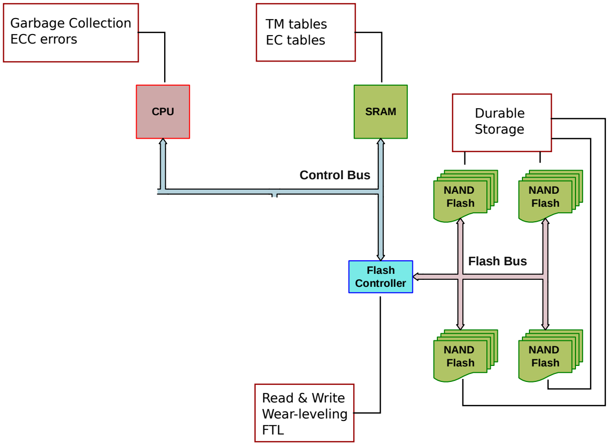
SSD Architecture



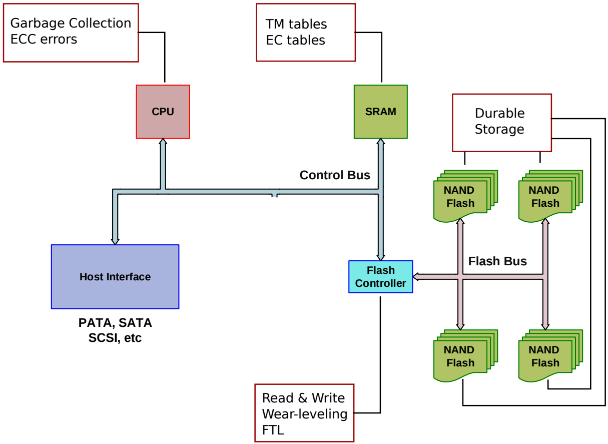
SSD Architecture



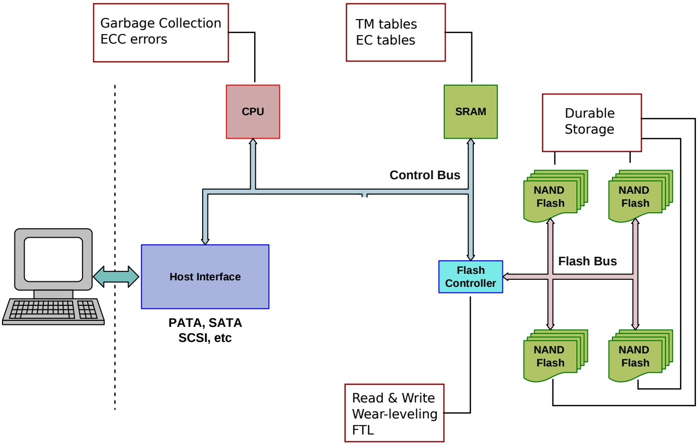
SSD Architecture



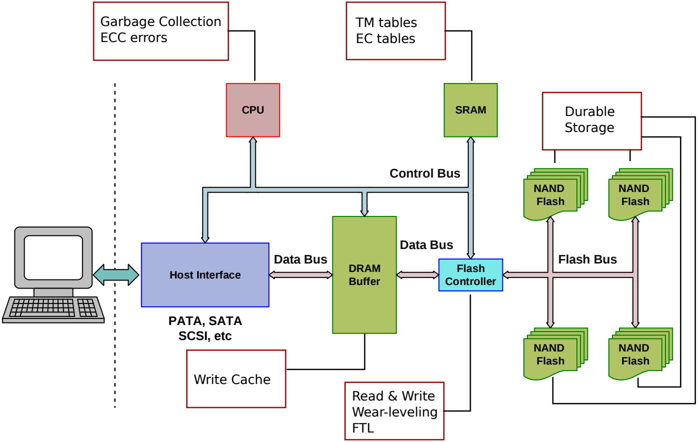
SSD Architecture



SSD Architecture



SSD Architecture



Data Reduction

- Reducing the amount of user data effectively stored in flash chips allows to reduce the write rate and increase the life of flash drives
- Data reduction techniques are:
  - Compression
  - Deduplication

Data Reduction

Data Compression

- It consists in reducing the number of bits needed to store data.
- Lossless compression allows to restore data to its original state
- Lossy compression permanently eliminates bits of data that are redundant, unimportant or imperceptible
- $CompressionRatio = \frac{UncompressedSize}{CompressedSize}$

⇒ Data reduction is $DR_c = \frac{1}{CompressionRatio}$

$$SSD_{TTF} \approx \frac{Capacity_{SSD} \cdot PECycles}{WriteRate \cdot (1 + A_f) \cdot DR_c}$$

Data Reduction

Data Deduplication

- It looks for redundancy of sequences of bytes across very large comparison windows.
- Sequences of data are compared to the history of other such sequences.
- The first uniquely stored version of a sequence is referenced rather than stored again
- Let *DD* the average percentage of deduplicable data

$$SSD_{TTF} \approx \frac{Capacity_{SSD} \cdot PECycles}{WriteRate \cdot (1 + A_f) \cdot DR_c \cdot (1 - DD)}$$

RAID Solutions on flash technology

- RAID uses redundancy (e.g. a parity code) to increase reliability
- Any RAID solution increase the amount of data physically written on disks (RAID Overhead)

⇒ when adopting a RAID solution with flash technology we are reducing the lifetime of the whole storage system by a factor at most equal to the RAID overhead

RAID Solutions on flash technology

Example

- N flash disks of capacity C and cells supporting L P/E-cycles.
- Write load rate equal to W.

RAID Solutions on flash technology

Example

- N flash disks of capacity C and cells supporting L P/E-cycles.
- Write load rate equal to W.

RAID0:

- stripes data
- no fault tolerance
- W is uniformly distributed on disks (thanks to striping)

RAID Solutions on flash technology

Example

- N flash disks of capacity C and cells supporting L P/E-cycles.
- Write load rate equal to W.

RAID0:

- stripes data
- no fault tolerance
- W is uniformly distributed on disks (thanks to striping)

$\Rightarrow TTL_{RAID0} = \frac{N \cdot C \cdot L}{W}$

RAID Solutions on flash technology

Example

- N flash disks of capacity C and cells supporting L P/E-cycles.
- Write load rate equal to W.

RAID0:

- stripes data
- no fault tolerance

RAID10:

- stripes data
- replicates each disk

- W is uniformly distributed on disks (thanks to striping)

$\Rightarrow \; TTL_{RAID0} = \frac{N \cdot C \cdot L}{W}$

RAID Solutions on flash technology

Example

- N flash disks of capacity C and cells supporting L P/E-cycles.
- Write load rate equal to W.

RAID0:

- stripes data
- no fault tolerance

RAID10:

- stripes data
- replicates each disk

- W is uniformly distributed on disks (thanks to striping)

$\Rightarrow TTL_{RAID0} = \frac{N \cdot C \cdot L}{W}$

$\Rightarrow TTL_{RAID10} = \frac{N \cdot C \cdot L}{2W}$

RAID Solutions on flash technology

Example

- N flash disks of capacity C and cells supporting L P/E-cycles.
- Write load rate equal to W.

RAID0:

- stripes data
- no fault tolerance

RAID10:

- stripes data
- replicates each disk

- W is uniformly distributed on disks (thanks to striping)

$\Rightarrow \; TTL_{RAID0} = \frac{N \cdot C \cdot L}{W}$

$\Rightarrow \; TTL_{RAID10} = \frac{N \cdot C \cdot L}{2W}$

Alert!

In order to increase reliability we **half** the time to live of flash cells

Modeling SSD endurance in a complex system

$$SSD_{TTF} \approx \frac{Capacity_{SSD} \cdot PECycles}{WriteRate \cdot (1 + A_f) \cdot DR_c \cdot (1 - DD) \cdot Overhead_{RAID}}$$

Modeling SSD endurance in a complex system

SSDs

$$SSD_{TTF} \approx \frac{Capacity_{SSD} \cdot PECycles}{WriteRate \cdot (1 + A_f) \cdot DR_c \cdot (1 - DD) \cdot Overhead_{RAID}}$$

Modeling SSD endurance in a complex system

SSDs

$$SSD_{TTF} \approx \frac{Capacity_{SSD} \cdot PECycles}{WriteRate \cdot (1 + A_f) \cdot DR_c \cdot (1 - DD) \cdot Overhead_{RAID}}$$

System Workload

Modeling SSD endurance in a complex system

SSDs

$$SSD_{TTF} \approx \frac{Capacity_{SSD} \cdot PECycles}{WriteRate \cdot (1 + A_f) \cdot DR_c \cdot (1 - DD) \cdot Overhead_{RAID}}$$

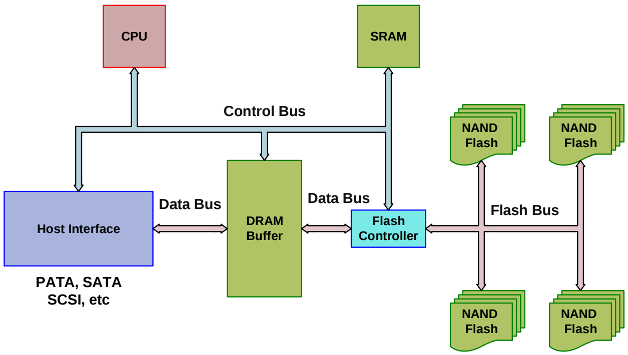
System Workload

SSD System

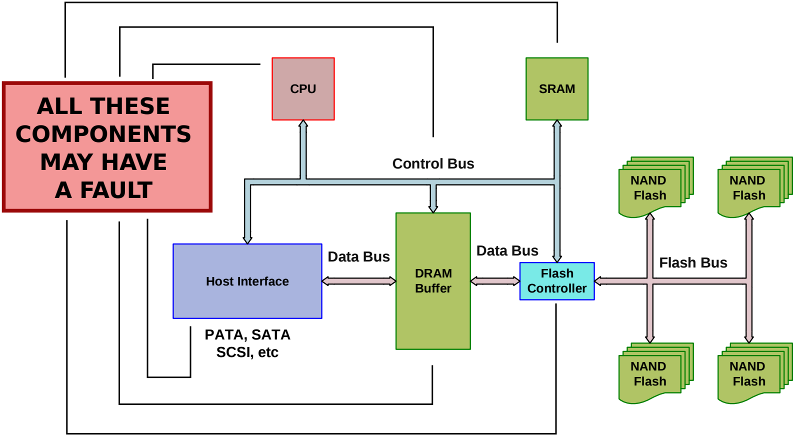
Does it still make sense to use RAID?

- We have learned that any redundancy reduces the maximum time to live of all SSDs
- The answer is **YES**, but why?

Does it still make sense to use RAID?YES! Why?

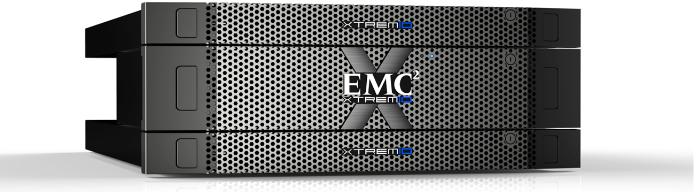


Does it still make sense to use RAID?YES! Why?



EMC XtremIO

- The system building block is called XBrick:
  - 25 800GB eMLC SSDs
  - Two 1U Storage Controllers (redundant storage processors)
- The scale up is guaranteed by adding more XBricks (up to six in a rack) that will be connected through InfiniBand ports.
- The system performs inline data reduction by:
  - deduplication
  - compression



EMC XtremIO

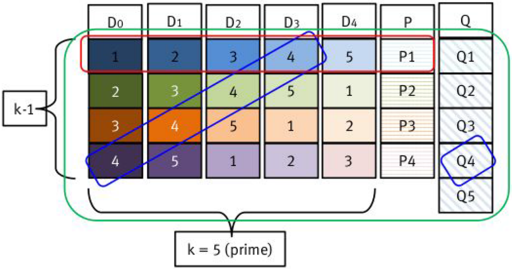
Deduplication

- The system checks for deduplicated data:
  1. subdivide the write stream in 4KB blocks
  2. for each block in the stream
     - 2.1 compute a digest
     - 2.2 check in a shared mapping table the presence of the block
     - 2.3 if present update a reference counter
     - 2.4 else use the digest to determine the location of the block and send the block to the respective controller node
- The addressing of blocks should uniformly distribute the data on all nodes

EMC XtremIO

XtremIO Data Protection

- The XtremIO system implements a proprietary data protection algorithm called XtremIO Data Protection (XDP)
- Disks in a node are arranged in 23+2 columns
- 1 row parity and 1 diagonal party
- Each stripe is subdivided in 28 rows and 29 diagonals



- In order to compute efficiently the diagonal parity and to spread writes on all disks, XDP waits to fill in memory the emptiest stripe
- When the stripe is full, commit it on disks
- The emptiest stripe selection implies that free space is linearly distributed on stripes
- XDP can:
  - overcome 2 concurrent failures (2 parities)
  - have a write overhead smaller than other RAID solutions

Suppose a system that is 80% full:

- The emptiest stripe is 40% full (due to the emptiest selection)
- A stripe can handle $28 \cdot 23 = 644$ writes
- The emptiest stripe can handle $644 \cdot 40\% \approx 257$

$$\#parities = 28(rows) + 29(diagonal) = 57$$

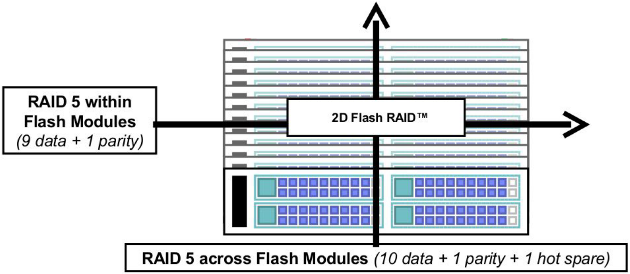
$$RAIDoverhead = \frac{\#writes}{\#userwrites}$$

$$RAIDoverhead = \frac{257 + 57}{257} 1.22$$

IBM FlashDrive V840

- The system building block is made of:
    - One Storage Enclosure of 12 4TB eMLC SSDs
    - Two Control Enclosures (redundant storage processors) with 8-core Intel Xeon and 32GB of RAM
- The system performs inline data reduction by:
    - compression with two dedicated hardware accelerators

- FlashDrive V840 offers two levels of RAID protection:
  - RAID5 in configuration 10 +1 Parity +1 Spare among disks
  - RAID5 in configuration 9+1 among chips in a disk
  - RAID overhead = 4



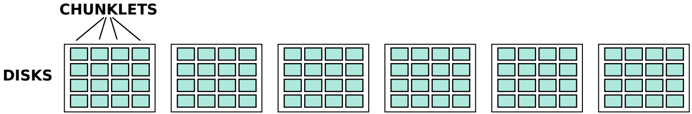
HP 3PAR STORE 7450

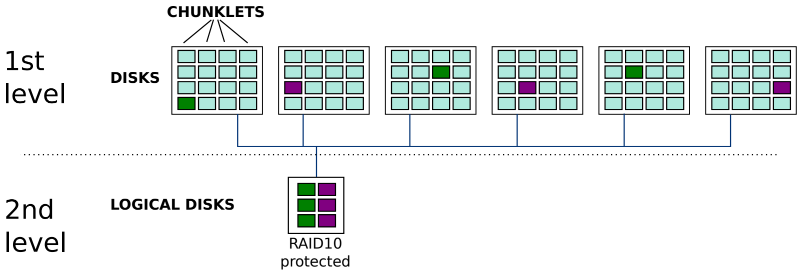
- One storage enclosure equipped with:
  - 2 controller nodes with 2 Intel eight-core processors and 32GB of RAM
  - 24 SSD drives
  - according to the type of flash cells, drives capacity is: 1920GB (MLC), 400GB (eMLC), 200GB (SLC);
- The system performs inline data reduction by:
  - deduplication which uses a hashing engine capability built into ASICs

HP 3PAR STORE 7450

Data Protection

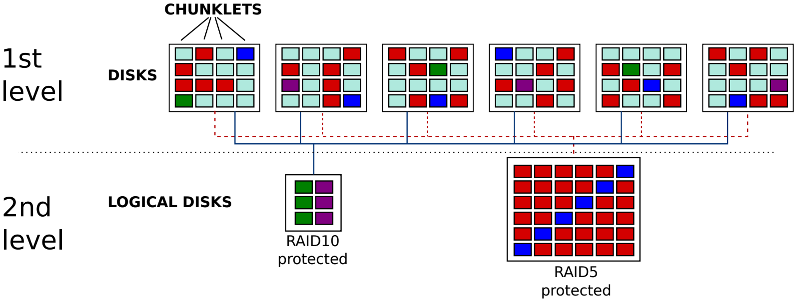
1st
level




CHUNKLETS
1st level
DISKS
2nd level
LOGICAL DISKS
RAID10 protected

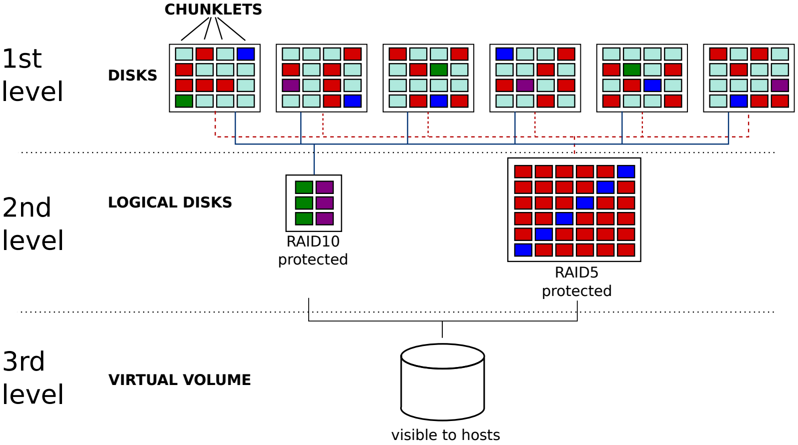
HP 3PAR STORE 7450

Data Protection



HP 3PAR STORE 7450

Data Protection



Additional Material

More detailed info can be found in the main references:

- `http://www.csee.umbc.edu/~squire/images/ssd1.pdf`
- XtremIO, FlashDrive v840, HP 3PAR white papers

If you want to play, there is an interesting tool by Intel:

- `http://estimator.intel.com/ssdendurance`

Quick look to the (not-far) future