Class 06

Shared Folders on Kali Linux

SSDs

Volumes

Partitions

Shared Folders on Kali Linux

Sharing Folders Between RADISH (Windows 10) and your Kali Linux VM

Background

From Last Week

Last lecture, we set up Kali Linux on your RADISH desktop

Be sure you have completed this task by Friday, 30 Sept

During that exercise, we set up shared folders between the RADISH (Windows 10) VM and the Kali Linux VM

We shared three folders (available on RADISH):

E: drive

R:\share

Your documents directory

The shared folders SHOULD be available at the following directory location on Kali Linux:

`/mnt/hgfs`

Sharing Issue Resolution

However, you may find that the `hgfs` folder doesn't exist in your `/mnt` directory

The following slides will walk you through the resolution

The Fix
Page 1 of 4

1. Power up the Kali Linux VM and log in

   login: kali

   password: kali
2. On Workstation Pro, using the *VM > Settings* menu option to open up the Virtual Machine Settings window
3. Click on the *Options* in the Virtual Machine Settings window
4. Click on *Shared Folders* on the left pane
5. Click the *Disabled* radio button the upper right of the window
6. Click *OK* on the lower right

The Fix

Page 2 of 4

7. Open up a Terminal window in your Kali Linux VM
8. Type in the following two commands:

   ```
   sudo apt-get remove open-vm-tools
   sudo apt-get purge open-vm-tools
   ```

9. Reboot Kali Linux
10. Once Kali Linux has finished rebooting, log in
11. Select the following Workstation Pro menu item:
    VM > Install VMWare Tools
12. You will then see a message bar at the bottom of your Kali Linux windows telling you to install VMWare Tools via a CD

The Fix

Page 3 of 4

13. Double-click on the File System icon on your Kali desktop to open the File Manager
14. Click on Computer on the left panel of the File Manager
15. Verify that you see “VMWare Virtual IDE CDROM Drive” displayed on the right
16. Open up a Terminal window in your Kali Linux VM
17. Go to the temp directory by using the following command:
    `cd /tmp`
18. Unzip the VMWare tools package into your tmp directory using the following command:

```
tar zxpf /medi/kali/VMkware\ Tools/VMwareTools-10.3.23-16594550.tar.gz
```

19. Navigate to the tools distribution directory with the following command:
    `cd vmware-tools-distrib`
20. Run the VMware Tools installation script with the following command:
    sudo ./vmware-install.pl
21. Follow the prompts to accept the default values.
22. Ensure the script completes execution

The Fix

Page 4 of 4

23. Shut down Kali Linux
24. Exit Workstation Pro
25. Bring up Workstation Pro
26. Power up Kali Linux
27. Log in to Kali Linux
28. On Workstation Pro, using the *VM > Settings* menu option to open up the Virtual Machine Settings window
29. Click on the *Options* in the Virtual Machine Settings window
30. Click on *Shared Folders* on the left pane
31. Click the *Always enabled* radio button the upper right of the window
32. Click *OK* on the lower right
33. In the Kali VM, double-click on the File System icon on the desktop
34. Verify that you have the **`hgfs`** folder in your **`/mnt`** directory

Flash Memory & SSD File Systems

Overview

Flash/SSD Memory

USB Flash/SSD Drives

Flash/SSD File Systems

Relation to conventional file systems

Flash/SSD Memory

Overview

History

Basic Information

NAND Flash & SSD Memory Organization

Some History

Invented in early 1980s by Toshiba

Called “flash” because erasing the memory reminded scientists of the flash of a camera

Intel offered the first NOR (Not OR) flash memory products in late 1980s

- *Keeps its memory state without needing power*
- *Read/write random access; state can be changed electronically*
- *Code that is stored can be executed directly from NOR flash*
- *Targeted at replacing ROMs (not changeable) and EPROMs (erased using UV light or X-ray)*
- *Useful for electronic configuration info and read-only code such as for BIOSs and TV set-top boxes*

Some History

NAND flash memory announced by Toshiba in mid 1980s

- *Much higher bit density and write speeds than than NOR flash*
- *Could be accessed much like magnetic disk drives*
- *Was thought of as being an eventual replacement for mass storage such as magnetic disk drives*
- *Drawback: Reads & writes had to be done in blocks*

First removable NAND flash memory in mid 1990s

- *Looked like a large SD card*

First NAND USB drive in 2000

Until the advent of smart phones, NAND flash memory was used mostly for USB flash drives and camera SD cards

Some History

2000: USB 2.0 standard specified 1.5, 12 or 480 Mbps

But NAND flash could do only 100 Mbps reads

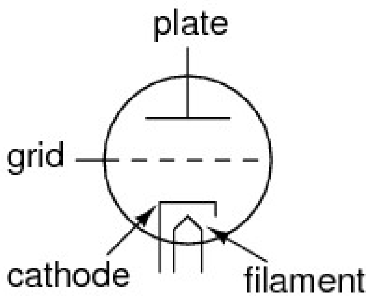
A lot less for writes

2008: USB 3.0 standard specified up to 5 Gbps

Even today's faster NAND flash memory can't come close

Some Basic Flash/SSD Info

Electronic Evolution*



Vacuum tube triode

Thermion amplifier

Invented: 1906 by Lee de Forest

Later on IIT's Armour College physics faculty

Originally used

Phone systems

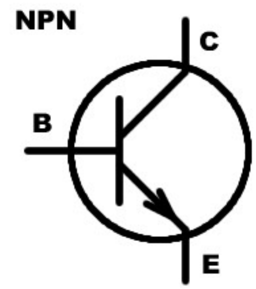
Radio & TV

Very early computers

e.g., AVIDAC*

Uses today

Power RF transmitters



Junction transistor

Created: 1947

W.Brattan, J.Bardeen, W.Schockley at Bell Labs

Nobel laureates

Used

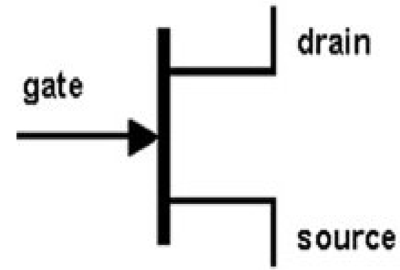
Phone systems

Radio & TV

Computers

Uses today

Electronic power devices



Field-Effect Transistor

MOSFET

Created: 1959

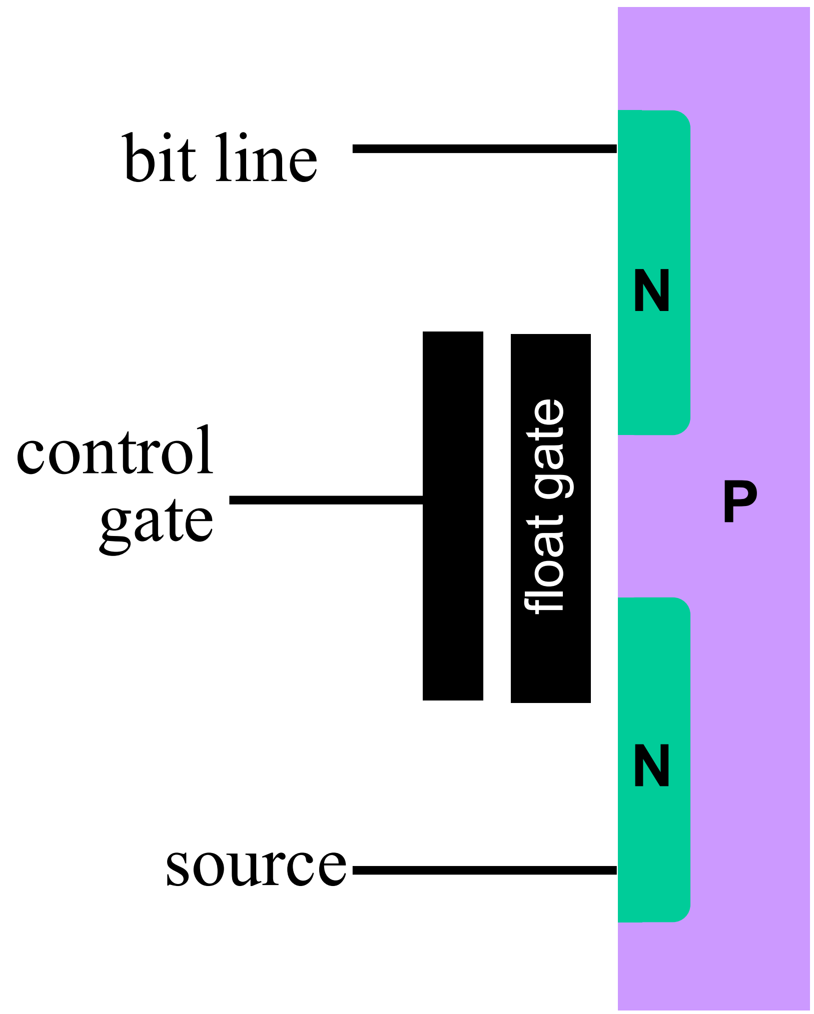
D.Kahng, M.Atalla at Bell Labs

Uses today

Almost all electronic devices and systems

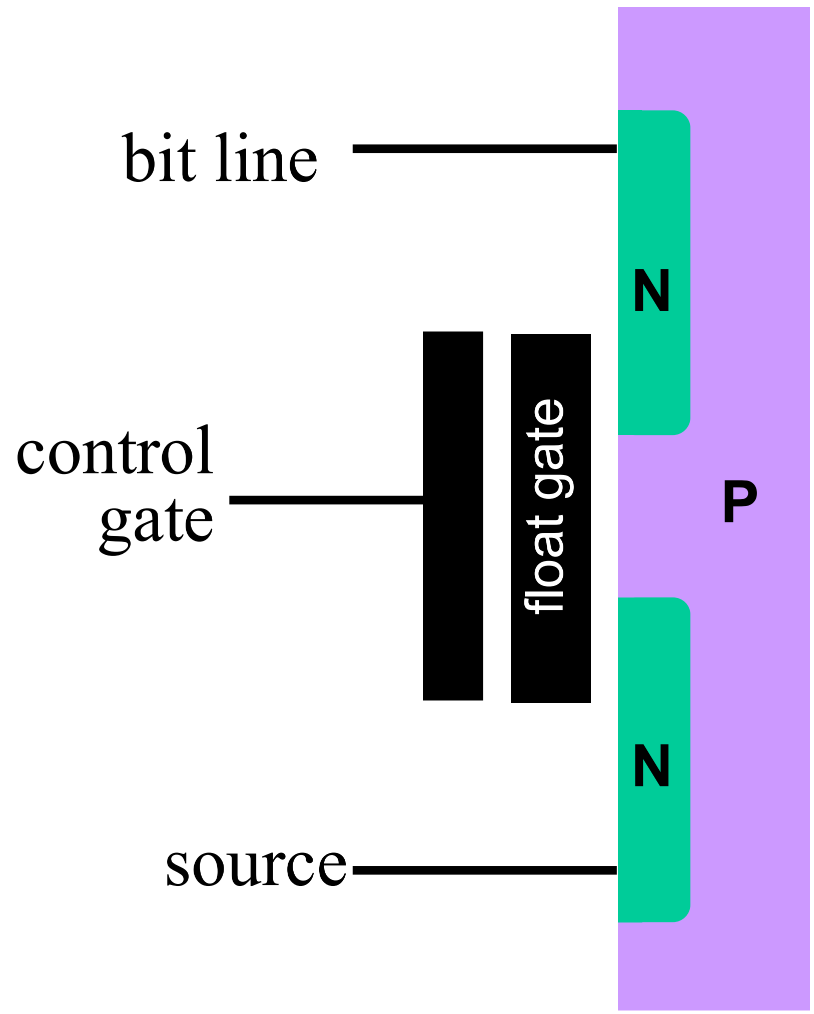
Flash Bit Cell

- *float gate* (FG) completely insulated by metal oxide
- Electrons put on *FG* can stay there for years
- The electron charge on the *FG* affects the field imposed by the *CG*
- Reading the bit
  - *If there is a negative charge on the FG, less current will flow in the bit line for a given CG-to-source voltage*
  - *And the converse if there is no FG charge*



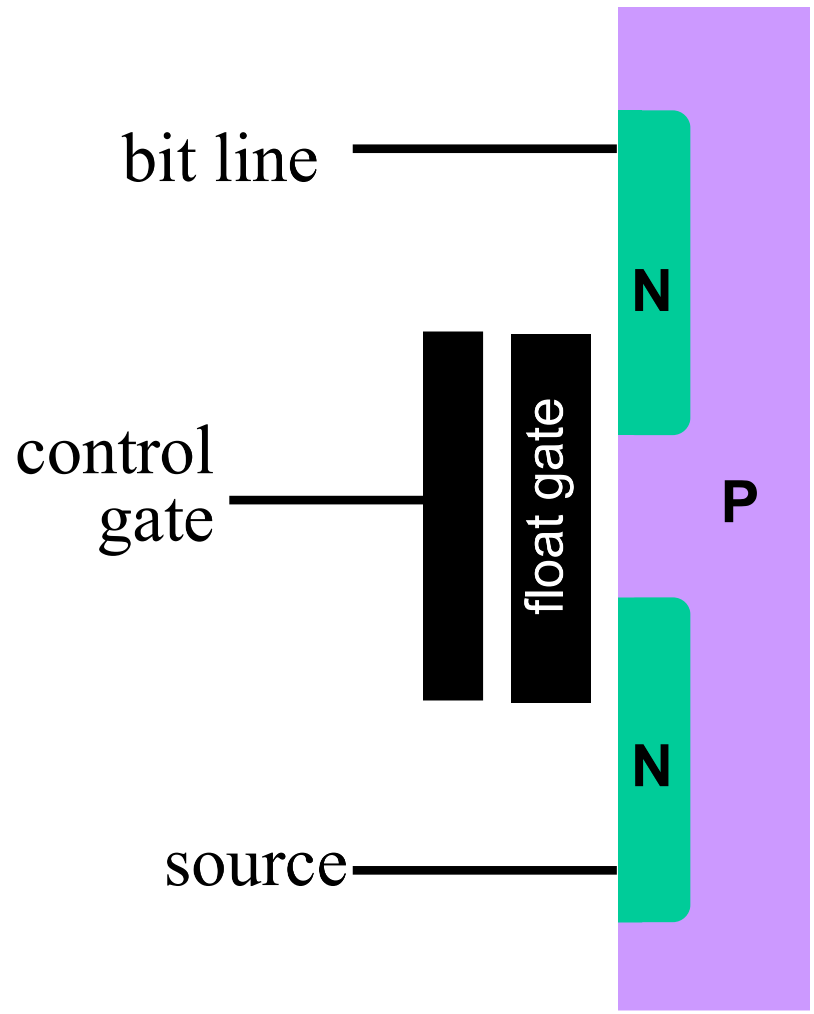
Putting Charge on the FG

- The normal charge on the FG is 0
- To put a negative charge on the FG:
  - *Apply a high + V_{GS} voltage to the CG with the bit line also at a high voltage*
  - *Because there is no charge on the FG, many electrons will flow from the source to the bit line*
  - *Some electrons will have sufficient energy to tunnel through the insulating layer to the FG*
- Called ***hot injection***



Removing Charge from the FG

- The source is disconnected
- A large + voltage is applied to the bit line
- A large negative voltage relative to the bit line is applied to the CG
- This causes electrons to be attracted to the N-doped silicon connected to the bit line
- If the voltage is high enough, electrons will leave the FG by tunneling through the insulation



Bit Wear

- Reading the bit that is stored on the float gate is done by simply sensing the bit line current
  - *No charge is added or removed from the float gate*
  - *Reading can be done an almost infinite number of times with no degradation*
- But changing the charge on the float gate involves hot injection
  - *This causes wear on the float gate insulation*
  - *After a number of program/erase (P/E) cycles, the metal oxide surrounding the FG will become degraded*
  - *Then the FG is unable to reliably hold a charge*

NOR Flash

Called NOR because the transistors are arranged as a NOR gate

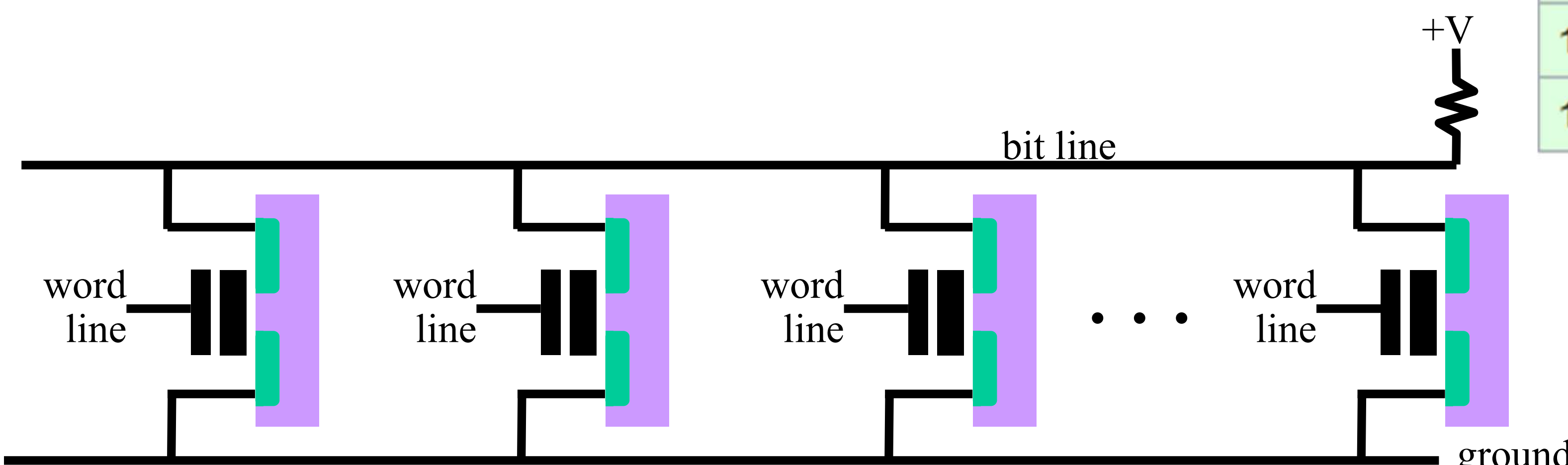
If any word line is high (1), it's flash cell conducts

This causes the bit line voltage to be close to ground, i.e., low (0)

A high output (1) is produced only if all inputs are low (0)

Here is a NOR truth table for two inputs (words)

Input		Output
A	B	A NOR B
0	0	1
0	1	0
1	0	0
1	1	0



NAND Flash

Here the FETs are arranged as a NAND gate

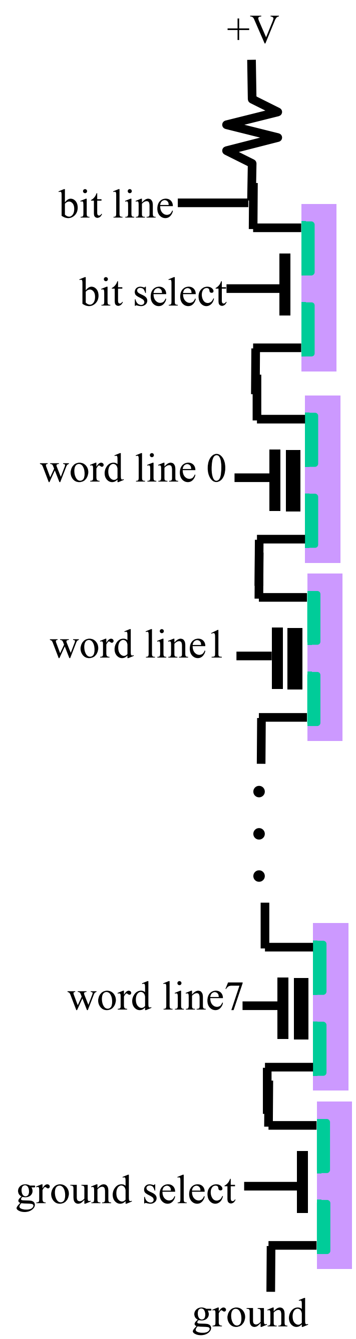
AND: WL0 & WL1 & … WL7 = 1

NAND: WL0 & WL1 & … WL7 = 0

All word lines, the bit select and the ground select must be high in order for the bit line to be low

Here is the NAND truth table for two inputs (words)

Input		Output
A	B	A NAND B
0	0	1
0	1	1
1	0	1
1	1	0

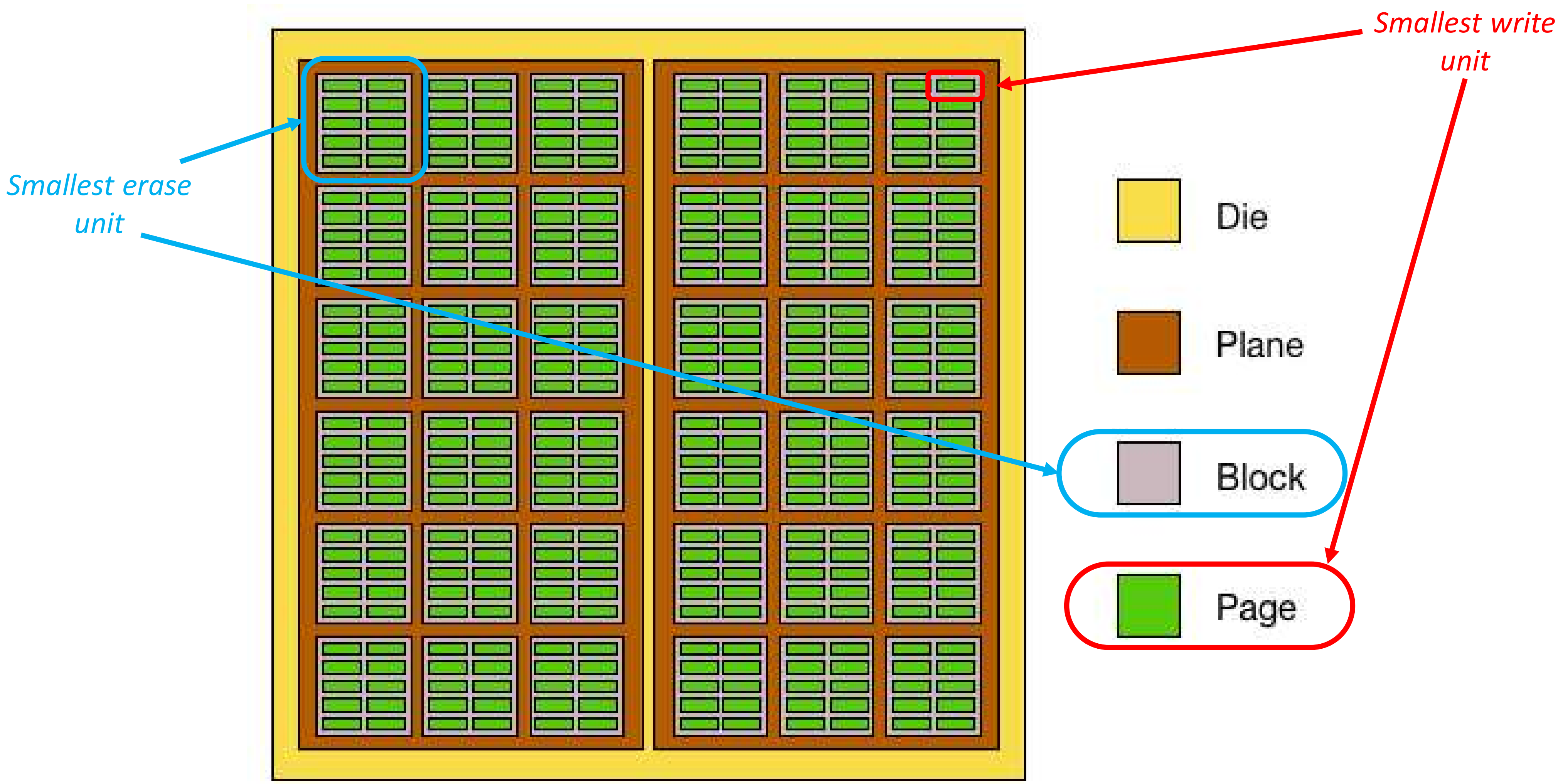


NAND Flash

NAND flash memory density can be made much greater than NOR flash. How?

- *Reduced the number of wires relative to NOR flash*
  - Only the word lines and a single ground and bit line are needed to access a bit
- *Multi-level cells*
- *NAND flash is arranged so as to not access individual bytes*
- *Taking advantage of NAND structure*
  - NAND flash is arranged so that
    - *It can be written as a group of bytes (e.g., a* **page***) at a time*
    - *But it can only be erased a* **block** *at a time*
    - **Blocks** >> **pages**

NAND Flash
Blocks and Pages



Multi-level NAND Flash Cells

There are currently 4 types of NAND flash cells

SLC (single-level cell):	*1 bit per cell*
MLC (multi-level cell):	*2 bits per cell*
TLC (tri-level cell):	*3 bits per cell*
QLC (quad-level cell):	*4 bits per cell*

How can we get more than 1 bit per cell?

Multi-level NAND Flash Cells

How?

We can put different charges on the FG

SLC: 1 bit per cell => 2 state of charge on the FG

MLC: 2 bits per cell => 4 states of charge on the FG

TLC: 3 bits per cell => 8 states of charge on the FG

QLC: 4 bits per cell => 16 states of charge on the FG

We then can use different voltages on the CG to differentiate these charges

This is really an ***analog*** way of storing bits in memory

Density goes up

But so does ***block*** size, error rate, wear out and access time

Why?

NAND Flash Memory P/E Cycles

Here are some P/E cycles for NAND flash before it becomes unreliable:

SLC NAND: ~10^5 P/E cycles

MLC NAND: ~10^4 P/E cycles

TLC NAND: ~10^3 P/E cycles

QLC NAND: ~500 P/E cycles

Recently these P/E cycles have been experimentally increased 10 to 50 times by local thermal annealing during block erasure

P/E cycles

- *Reading and writing 0s are done a* **page** *at a time*
  - Cannot write 1s to a page
- *Erasing (writing 1s) must be done a* **block** *at a time*
  - 1s are written to an entire block

More on ***pages*** and ***blocks*** in a few slides

NAND Flash Latency

As a function of bits/cell

	SLC	MLC	TLC	HDD	RAM
P/E cycles	100k	10k	5k	*	*
Bits per cell	1	2	3	*	*
Seek latency (µs)	*	*	*	9000	*
Read latency (µs)	25	50	100	2000-7000	0.04-0.1
Write latency (µs)	250	900	1500	2000-7000	0.04-0.1
Erase latency (µs)	1500	3000	5000	*	*
Notes	* metric is not applicable for that type of memory				

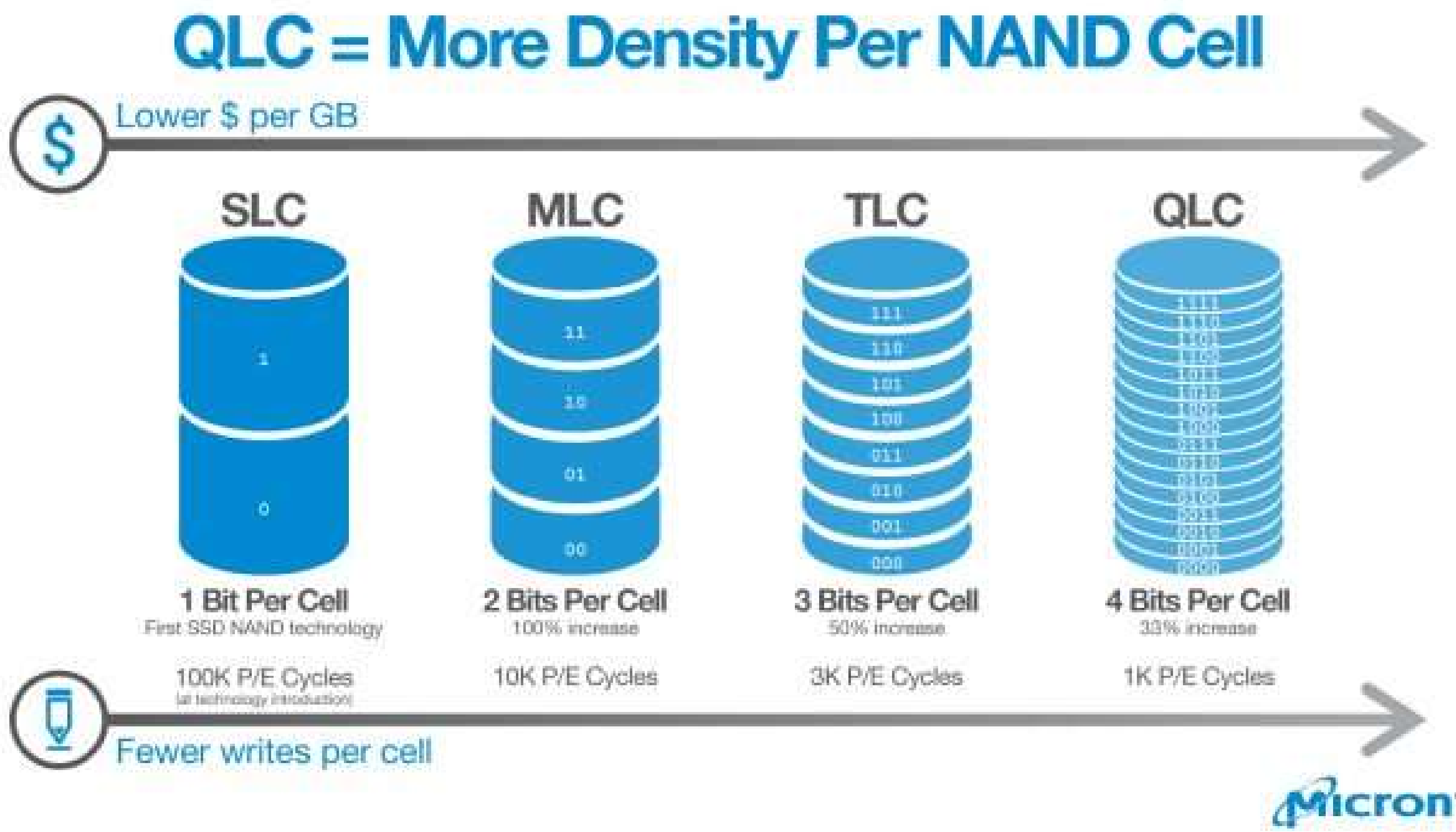
Note that latencies of NAND is far greater than HDD, but less than RAM

Note also that latencies increase as cell density increases

The SSD controller has to be more discriminating regarding voltages coming out of the NAND. Higher densities requires the memory controller to use more precise voltage measurements, increasing latencies

NAND Density Effects

P/E Cycles & Cost / GB



Read Thy Neighbor

Reading NAND flash memory can cause both the cells being read and also adjacent cells to change over several hundred thousand read cycles

Because erasing requires a high voltage

The problem is mitigated by erasing at the block level rather than the page level

Controllers must keep track of cell read counts and rewrite cells when the count reaches a certain level

Rewrite means erasing (setting entire block to 1s) and then restoring an entire **bloc**

Compared to Magnetic Memory

Magnetic memory has none of the wear problems associated with flash memory

Actually they do, but

- It's not really "wear"
- The equivalent numbers of cycles is much m**uch** larger
- As areal density increases, magnetic dipole stability decreases

Because of wear and P/E cycles, SSD file systems must function differently than disk-based file systems

NAND Flash Memory Organization

Make It Look Like Mass Storage

NAND flash was targeted at replacing mass storage such as CDs, DVDs and magnetic disk drives

Groups of cells are arranged into ***pages***

Each ***page*** contains a number of bytes

- *e.g., 512, 1024, 2048 or 4096 data bytes*
- *Plus bytes for ECC (error correcting)*

So a flash ***page*** is sort of like a disk sector

Sort of emulates a rotating magnetic disk drive

- *But differs because only* **blocks** *of* ***pages*** *can be erased (all bits = 1)*
- *By comparison, for rotating disks each sector can be changed and bits can change from 0 → 1 or 1 → 0 without any issues*

Pages are arranged into ***blocks***

- *Each* **block** *usually contains between 32 and 128* **pages**

SSD Controllers

To make an SSD behave like an HDD, a significant number of functions are integrated into the SSD Controller.

- *Wear leveling*
- *Garbage collection*
- *Trim*
- *SLC cache management*

Many of these capabilities have analogs in the HDD world

SSD manufacturers keep their implementations of the functions under wraps

- *Affects SSD performance*
- *Can give then competitive edge*

Wear Leveling
Error and Wear

- Error correcting
  - *Claude Shannon (Bell Labs) developed the error correcting algorithms still used today*
  - *Flash drives usually correct for 1 or 3 bit errors per page*
  - *But some MLCs (multi-level cells) require 5 or more*
- Wear leveling – typical scheme
  - *Each time a file is changed and then saved, it is saved to a different location*
    - The old location of the file is marked as having been rewritten but available
  - *A memory location is never written a 2nd time until all the virgin or erased memory has been exhausted*

Wear leveling

Dynamic vs. Static Wear Leveling

SSD Garbage Collection
Key Characteristics of SSDs

SSDs using NAND technology
- *Fast reads*
- *Slow writes*
- *Very slow erases*

Pages are made up of cells

Blocks are made up of pages

Smallest unit of information for READ and WRITE operations
- *Page*

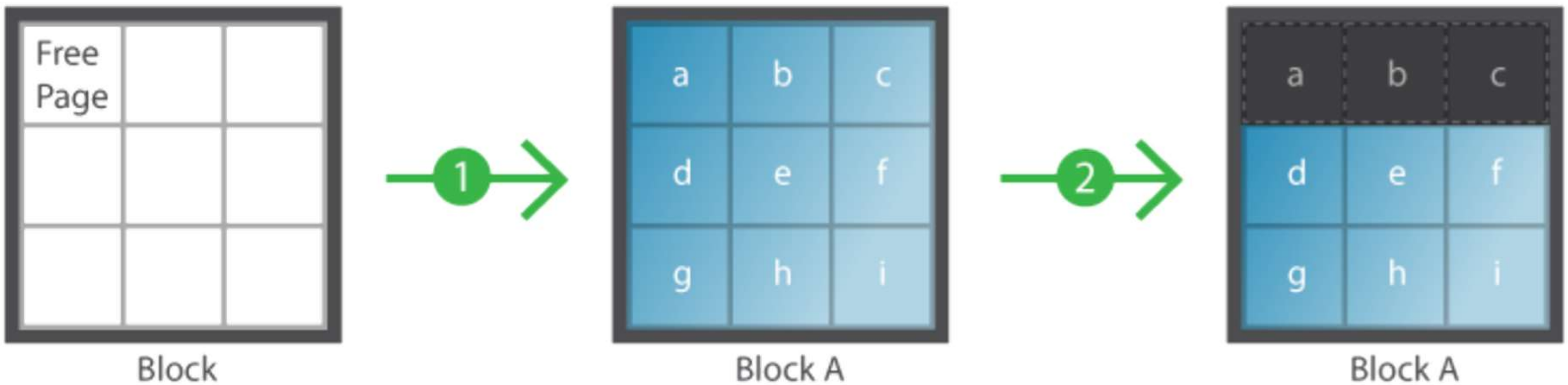
Smallest unit of information for ERASE operation
- *Block*

SSD Garbage Collection
Key Characteristics of SSDs

To re-write a cell that already contains data, you must first erase the cell.

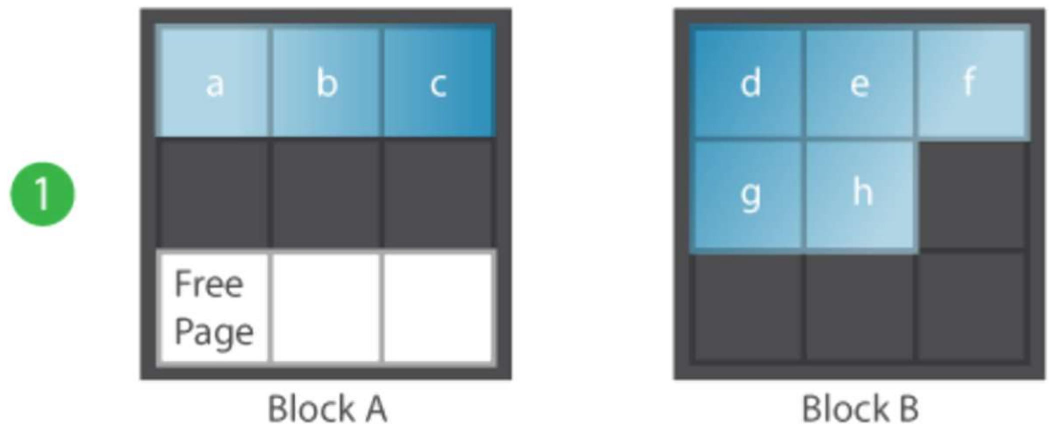
- *But the smallest unit for a write is a page*
  - And a page must be erased before it can be written again
  - And the smallest unit for an erase is a block
- *So, you must erase an entire block to re-write a NAND cell*
- *Recall that erases are the slowest operation*
  - For fast operations, you need to keep a ready pool of ready-to-write blocks
- *This is the reason for Garbage Collection*

SSD Garbage Collection



- Data are written to the 9 pages of Block A.
- After the write operation is done, Block A’s 9 pages are full.
- Pages a-c’s data are deleted, but pages cannot be individually erased
  - *They are marked as unreadable, but cannot be written to again.*

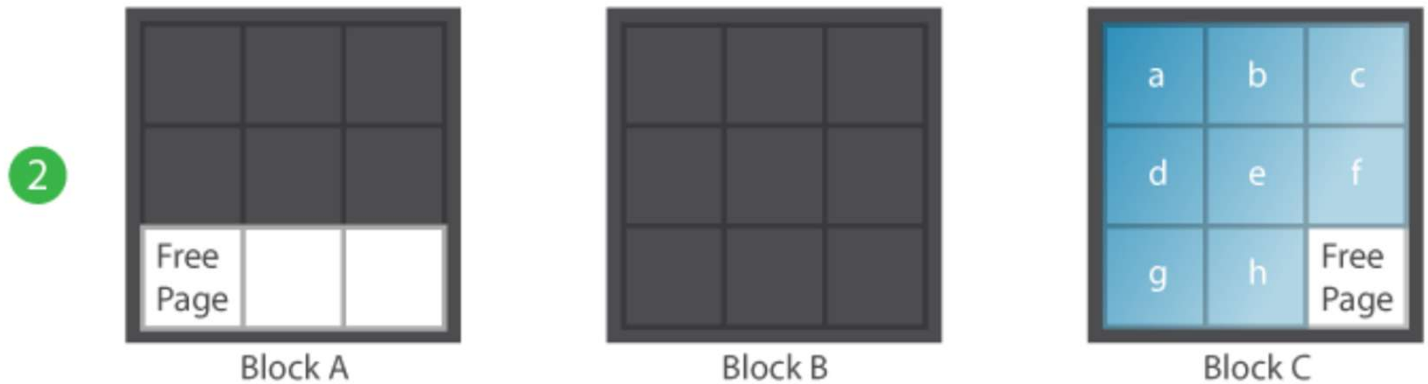
SSD Garbage Collection



Block A and Block B both have invalid pages (unreadable)

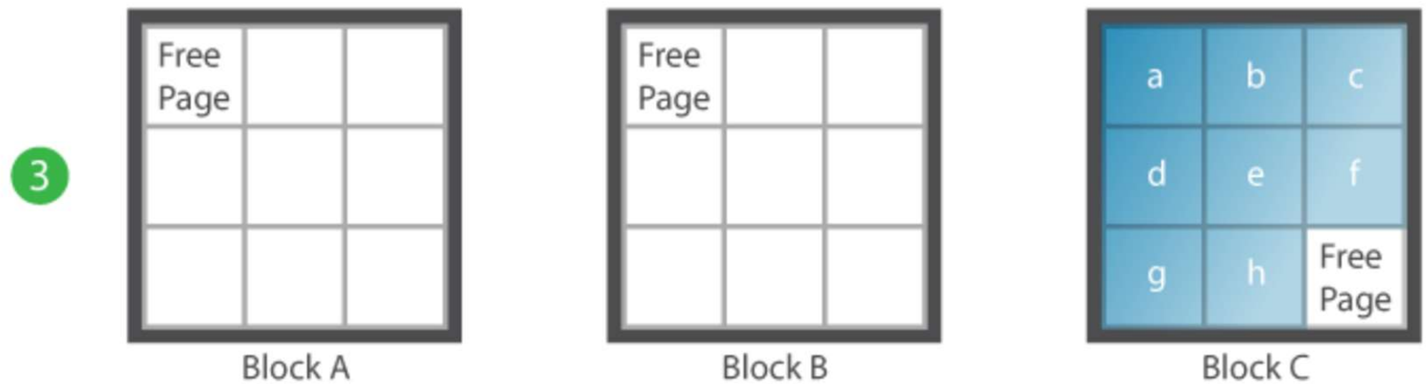
Designed as dark grey areas

SSD Garbage Collection



The system will read Pages a-h as containing valid information and transfer them to Block C

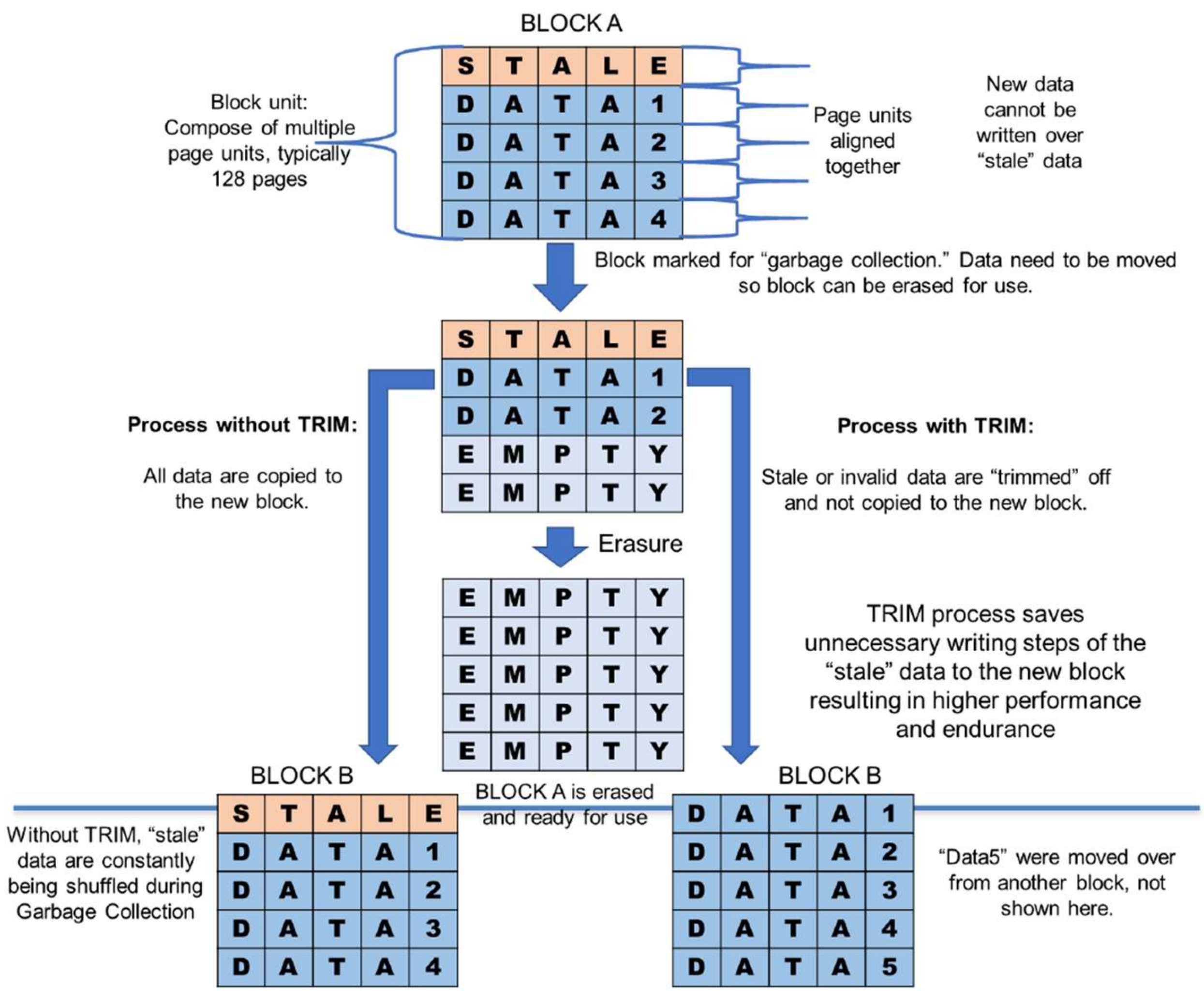
SSD Garbage Collection



Garbage collection then erases Blocks A and B

They become available for future writes

SSD Trim



BLOCK A
Block unit:
Compose of multiple
page units, typically
128 pages
S T A L E
D A T A 1
D A T A 2
D A T A 3
D A T A 4
Page units
aligned
together
New data
cannot be
written over
"stale" data
Block marked for "garbage collection." Data need to be moved
so block can be erased for use.
S T A L E
D A T A 1
D A T A 2
E M P T Y
E M P T Y
Process without TRIM:
All data are copied to
the new block.
Process with TRIM:
Stale or invalid data are "trimmed" off
and not copied to the new block.
Erasure
E M P T Y
E M P T Y
E M P T Y
E M P T Y
E M P T Y
TRIM process saves
unnecessary writing steps of the
"stale" data to the new block
resulting in higher performance
and endurance
BLOCK B
S T A L E
D A T A 1
D A T A 2
D A T A 3
D A T A 4
Without TRIM, "stale"
data are constantly
being shuffled during
Garbage Collection
BLOCK A is erased
and ready for use
BLOCK B
D A T A 1
D A T A 2
D A T A 3
D A T A 4
D A T A 5
"Data5" were moved over
from another block, not
shown here.

SLC Cache Management

Many SSD Controllers have SLC Caches

These caches are implemented as a portion of the drive

- *It acts as an SLC with better performance*

Two primary types of SLC Caches

- *Static*
    - Assigned area on SSD is fixed
    - Simpler scheme, but area can wear out more quickly
    - Cache size is guaranteed
- *Dynamic*
    - Area on SSD not fixed
    - Allows for longer lifetime (via wear leveling)
    - Cache size isn't guaranteed

SSD Challenges to Forensic Examiner

- To summarize
  - *SSDs have sophisticated controllers to emulate HDDs and maximize performance, cost and device lifetime.*
- This has the following implications for the forensic examiner
  - *On HDDs, when a file is deleted*
    - The space is marked as free
    - The data will remain there indefinitely until the space is needed for another write operation
  - *On SSDs, when a file is deleted*
    - The space taken by the file is also marked as free
    - Garbage collection will eventually erase the deleted data

Familiar File Systems

A Quick Overview

Some Common File Systems

FATfs (*File Allocation Table*)

- *Several variants*
  - FAT12, FAT16, FAT32, VFAT…
- *Used in rotating mass storage*
- *Also used in SD cards, USB devices*
  - Really flash memory that is made to appear as a FATfs

NTFS (*New Technology File System*)

- *Developed by Microsoft*
- *Used in all Windows OSs*
- *Several variants with small differences*

Some Common File Systems

EXTfs *(Extended File System)*

Several variants (EXT2, EXT3, EXT4)

Used as default in most Linux systems

Flash File Systems

NAND Flash Needs

Because of the special issues surrounding NAND flash memory, file systems tailored to NAND flash are needed

Some issues

- *Flash memory* **blocks** *must be erased before they are written*
  - To minimize ***block*** erasures, some schemes cleverly try to write only where existing 1s can be changed to 0s
    - e.g., 0101 -> 0100 -> 0000
      - *The changes require only changes from 1 -> 0*
- *Rotating disks are not random access. Flash is!*
- *Wear leveling is highly desirable.*

Some Well Known Flash File Systems

TrueFFS

The first FFS; 1992

FFS2

Late 1992. Designed for use with MSDOS

Did some wear leveling

FTL

1994

Mapped LBA sector numbers to NAND flash pages

Wear leveling

Multiple proposed enhancements over the years

Some Well Known Flash File Systems

LogFS

Similar to JFFS2, but better at handling large memories in the multi-gigabyte range

We'll discuss JFFS2 next

JFFS2

JFFS2 released in 2001

First Linux FFS

Included in Linux kernels since v. 2.4.10

JFFS2 blocks that are the same size as the NAND flash ***blocks***

The smallest number of bits that can be erased (set to 1s)

JFFS2 “nodes” are contained in NAND ***blocks***

Nodes contain items such as files and directories

Each node has an inode containing metadata about the node

Nodes consist of one or more pages

Blocks ⊃ nodes ⊃ pages

JFFS2

Allocation

As files are updated, a new copy of the file is located in heretofore unused nodes or nodes that are part of cleaned blocks

Journaling

Log of JFFS2 "nodes"

States of JFFS2 **nodes**

Valid: When, as yet, unused or previously erased (all 1s)

Obsolete: Happens when a newer version of the contents of the node is located somewhere else

JFFS2

Blocks are filled with nodes from low numbered blocks to high numbered blocks

Block States

- ***Clean***: *All nodes are valid*
- ***Dirty***: *Block contains at least one obsolete node*
- ***Free***: *Has no nodes, valid or obsolete*

Garbage Collector

- *Background job*
- *Erases dirty blocks, converting them into free blocks*
- *If dirty block has both obsolete and valid nodes, it copies the valid nodes to another location before erasing*

JFFS2

Wear leveling

- *Copying updated files to a valid node affects wear leveling*
- *Also, as needed, contents of clean blocks containing fixed but heavily read info are copied to another block*

YAFFS2

Yet Another Flash File System

YAFFS has many similarities to JFFS

Accommodates newer NAND flash needs

YAFFS1 didn't need to write only once before erasing

Note: YAFFS 1 & 2 refer to NAND pages as "chunks" and obsolete pages as "*stale*"

YAFFS2 keeps a database in RAM of the physical location of each chunk (page) and their state

Sort of FAT's FAT or Ext's Block Bitmap

Stale chunks (pages) are marked by an indicator in a reserved byte of the chunk (page)

YAFFS2

As files are updated, a new copy of the file is located in heretofore unused chunk (page)

As each new block is used, it is given a sequence number

The block sequence number combined with the location in the block yields the chunk (page) sequence number

So if files have the same metadata, the latest one can be determined. The earlier ones are stale

Deleted files by unlinking it in the RAM database

YAFFS2 Issues

The file system index is stored in RAM

Means that it must be rebuilt upon each boot

This requires scanning the entire NAND flash memory

Takes a lot of time for multi-gigabyte memories

Write-thru caches

Saved files are written to NAND flash

This means that a number of small changes might consume a lot of NAND flash memory

Newer FFSs use write-back caching

Some issues exist respect to recovery from abrupt power outages

Beyond YAFFS2

The latest FFSs claim to overcome some of the issues with JFFS2 and YAFFS2

ExtremeFFS SanDisk

Better write efficiency and speed for MLC memory

TargetFFS-NAND

Targets embedding in backing stores

SSD Wrap Up

So now I hope that:

- *You have an idea of the complexity of flash file systems (FFSs)*
- *How FFSs differ from rotating disc drive file systems*
- *How much more is needed in a FFS*

And we haven't included

- *ECC* (Error Correction Codes)
- Refreshing

Wrap-Up

Because of

Tunneling wear

Asymmetry of program and erase

And also adjacent read wear

Flash mass storage and file systems are quite a bit different than rotating disk magnetic mass storage and their file systems

This lecture attempts to give you an understanding of what flash file systems do and why

Wrap-Up

- But today all the P/E cycle, wear leveling, error managing, etc. is handled in an SSD with on-board electronics
  - *It's "invisible" and almost inaccessible to the computer or software*
- So today's SSD looks like a rotating disk drive to the computer
  - *It follows the ATA command set standard – usually SATA*
  - *File systems such as FAT, EXT and NTFS along with MBRs and PBRs are what the BIOS and operating system "sees".*