Computer Systems and Storage Media

Goh Weihan

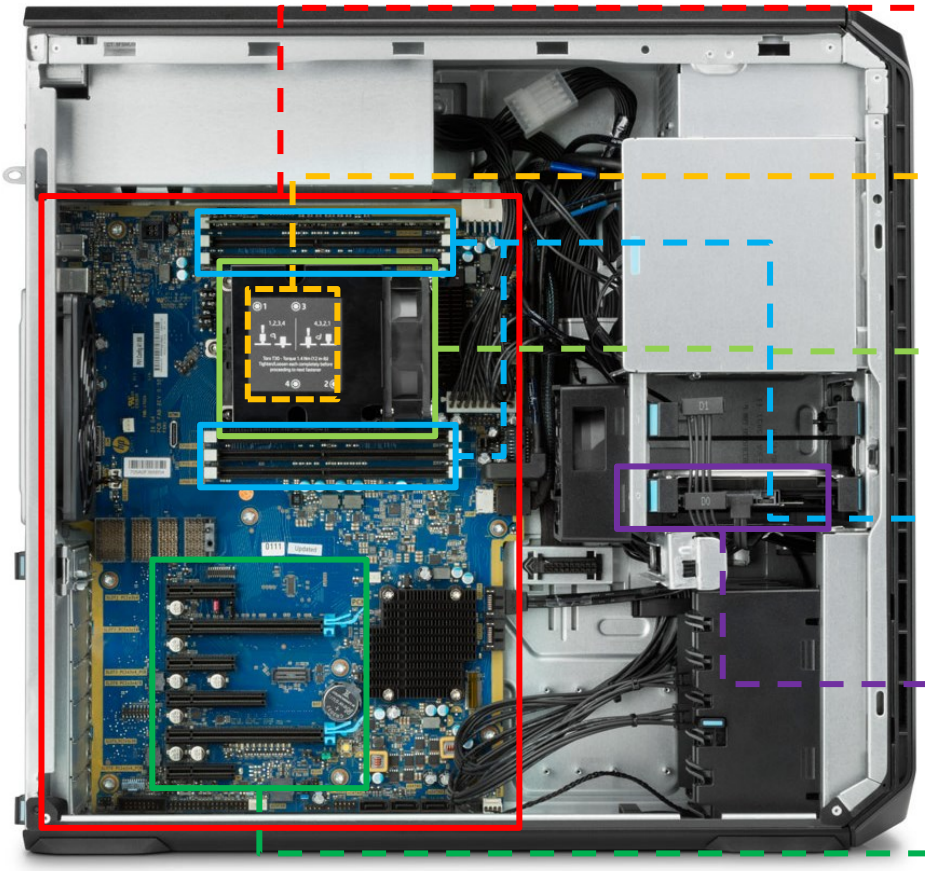
ICT3215 Digital Forensics

BEng (Hons) in Information and Communications Technology (Information Security)

September 2024

The Computer

Inside a Computer



Motherboard

Central processing unit (CPU)

Heatsink assembly

Random-access memory (RAM)

Hard disk

Expansion slots

Components of a Motherboard

CPU socket

RAM slots

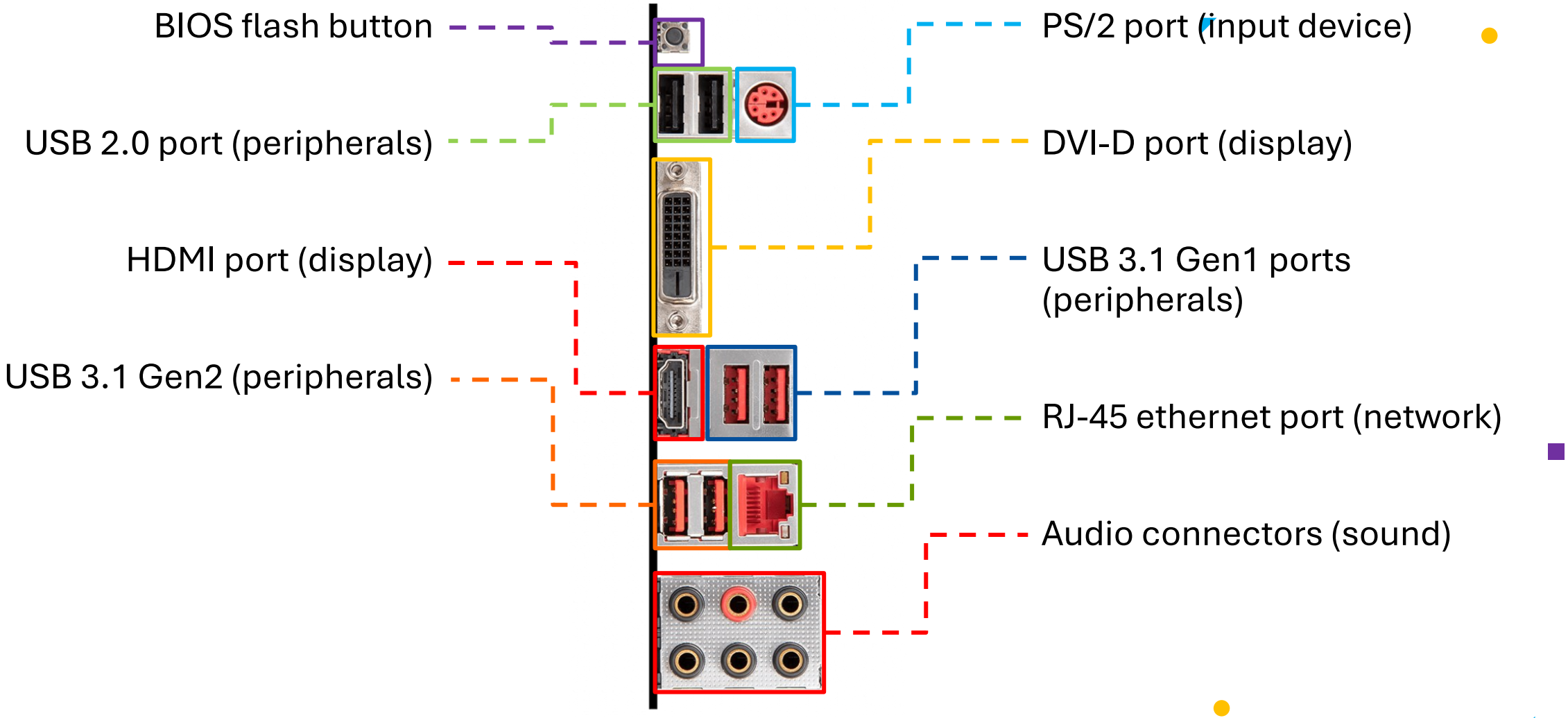
Power connectors

- 24-pin ATX main power connector
- 8-pin ATX 12V power connector

Storage connectors

- M.2 port
- SATA ports

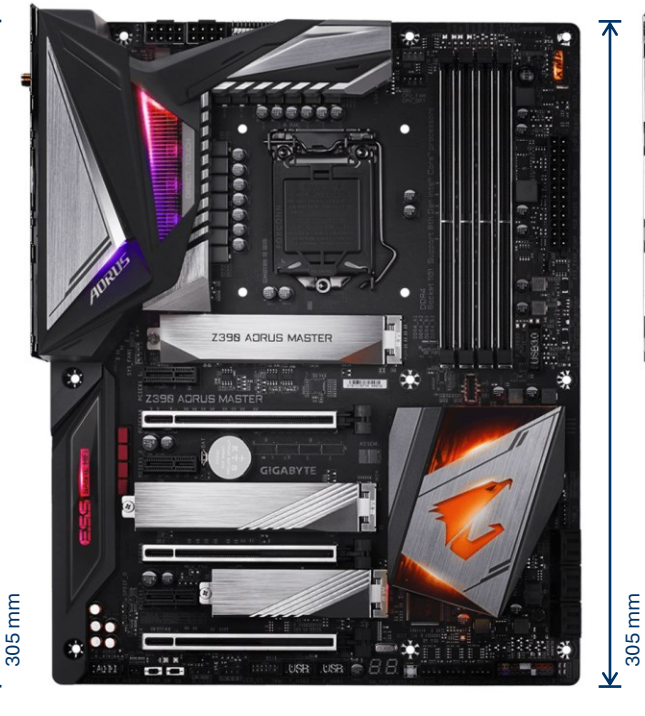
External Ports on a Motherboard



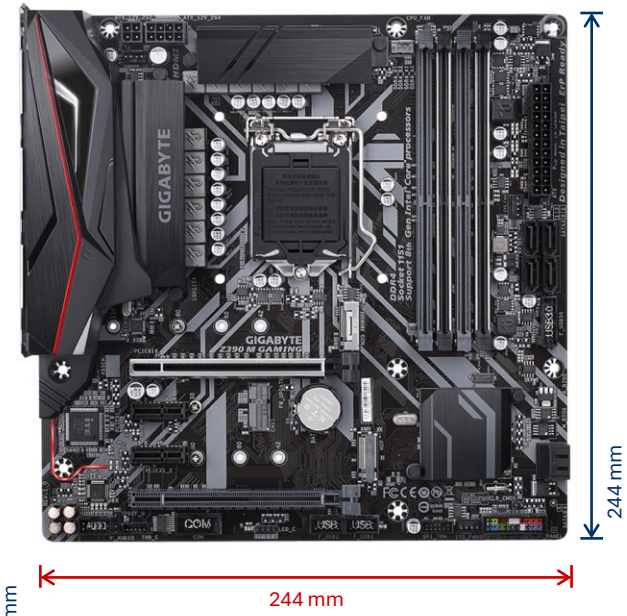
Desktop / Workstation Motherboard Form Factors

Extended ATX (E-ATX)

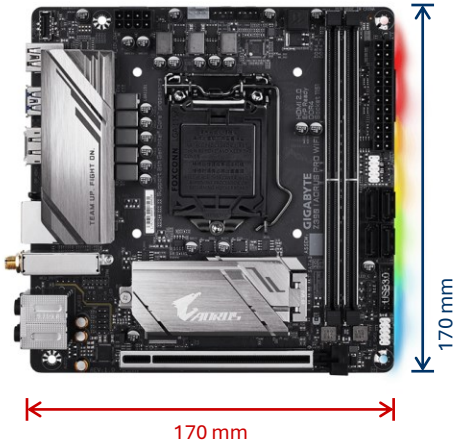
ATX



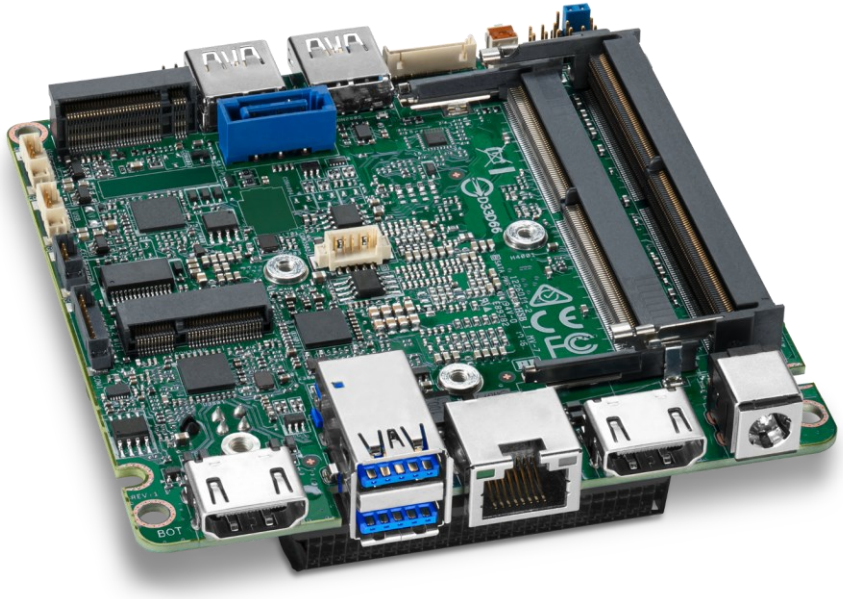
microATX (mATX)

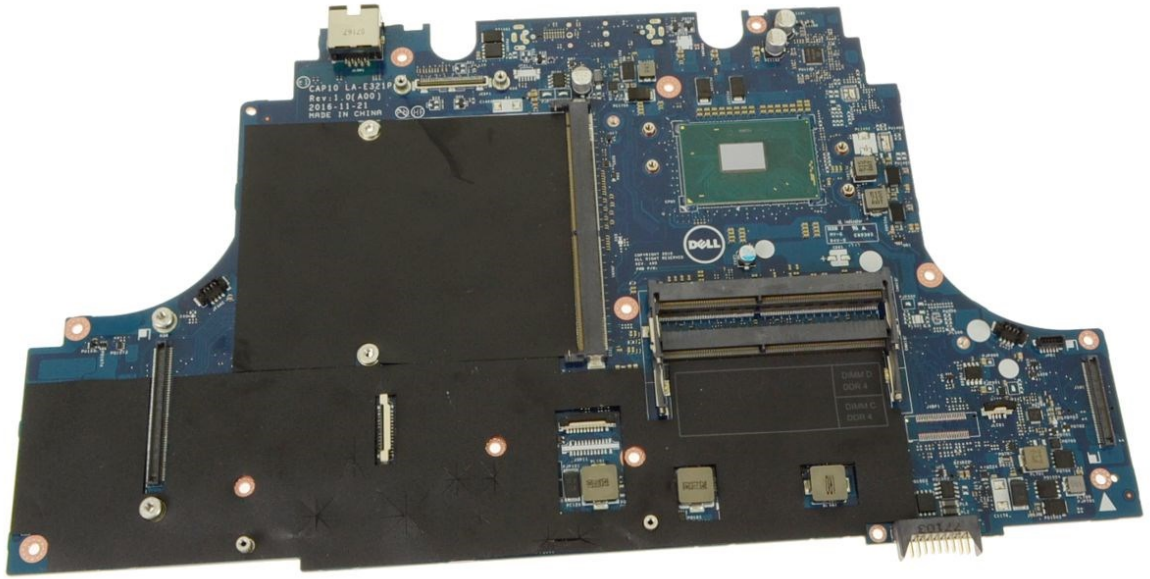


Mini-ITX (ITX)



Other Motherboard Form Factors





The Central Processing Unit (CPU)

Fetch, decode, execute

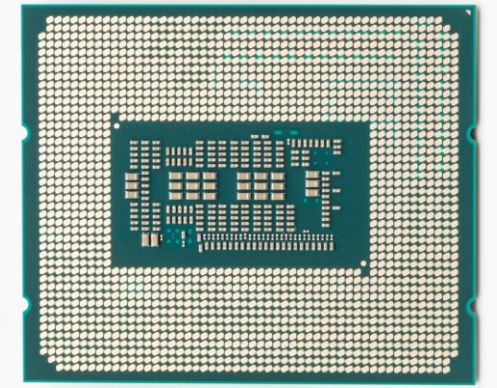
What does it look like?

- Usually square or rectangle in shape
- Has connectors (pins or pads) at the underside

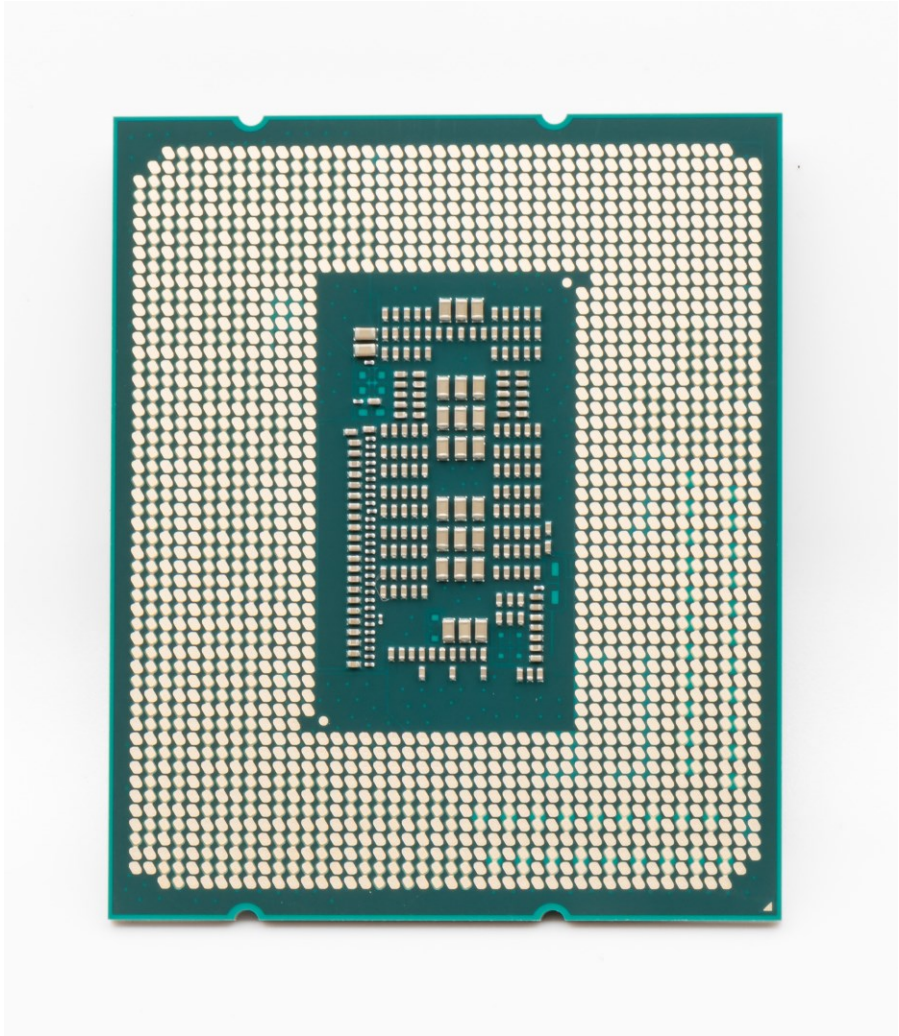
Attaches to a **CPU socket** on the motherboard

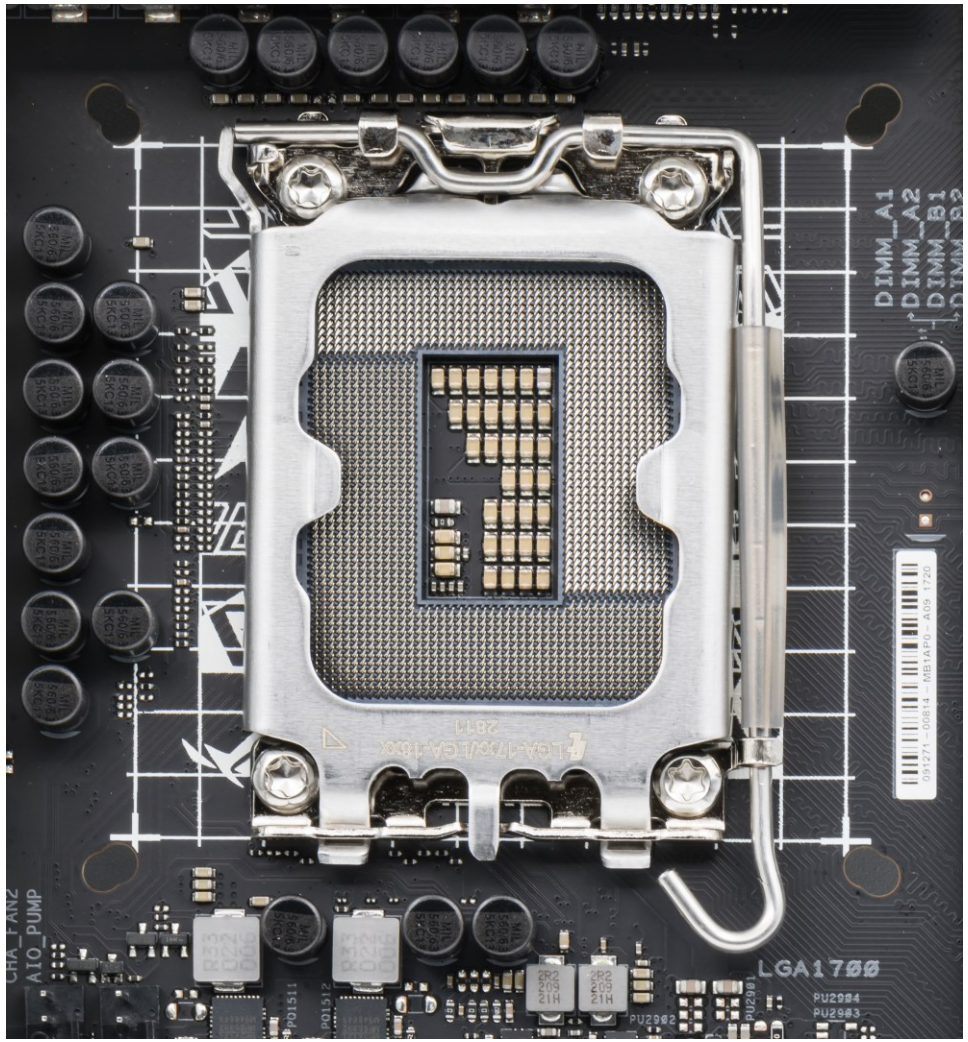
Two major CPU manufacturers, **Intel** and **AMD**

Runs hot, requires cooling (typically a heatsink)



The CPU and CPU Socket





A CPU designed for one socket may not be compatible with another

Memory Devices

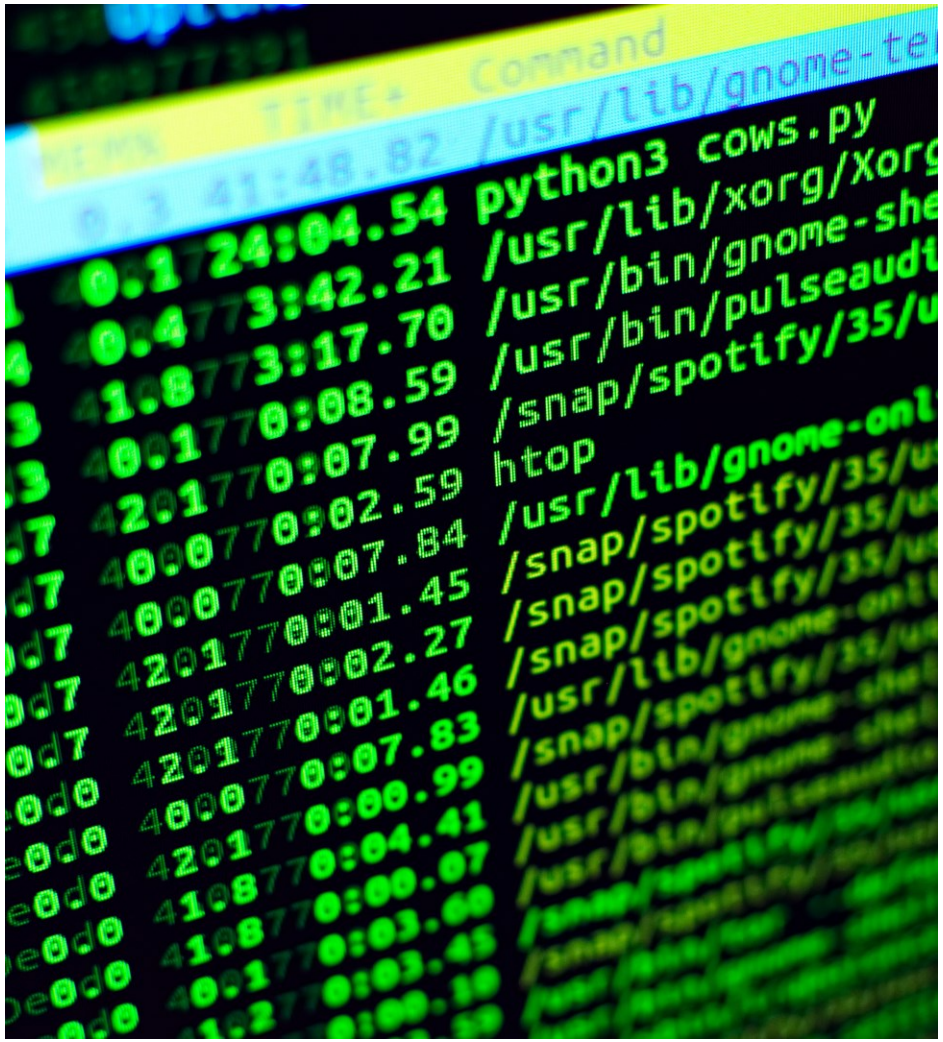
Memory Devices

Digital data are stored in various memory devices within a computer system

These devices have different characteristics

- Data retention capability
- Power requirement
- Performance
- Portability

Can be broadly categorized into two categories - **volatile memory**, and **non-volatile memory**



Volatile memory devices requires power to retain its contents

Non-volatile memory devices can retain its contents when powered down

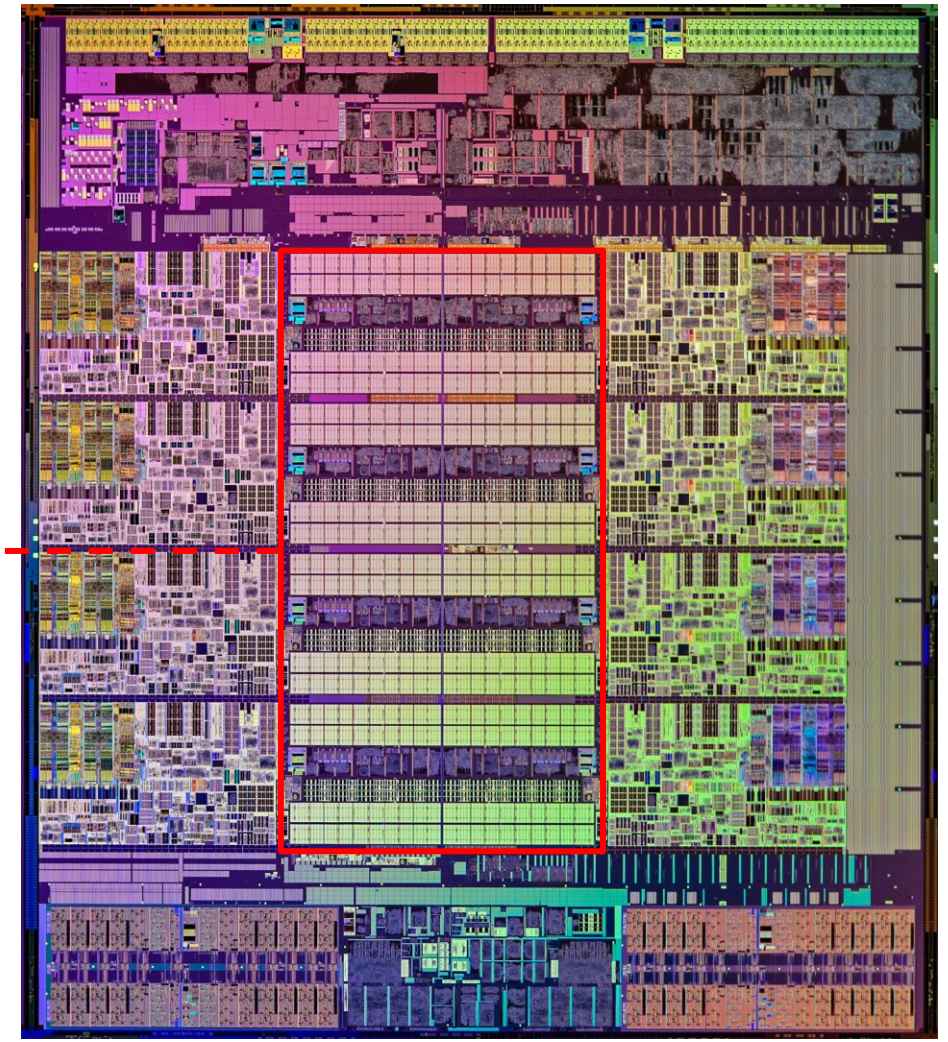
Volatile Memory in Computer Systems

Example of volatile memory devices

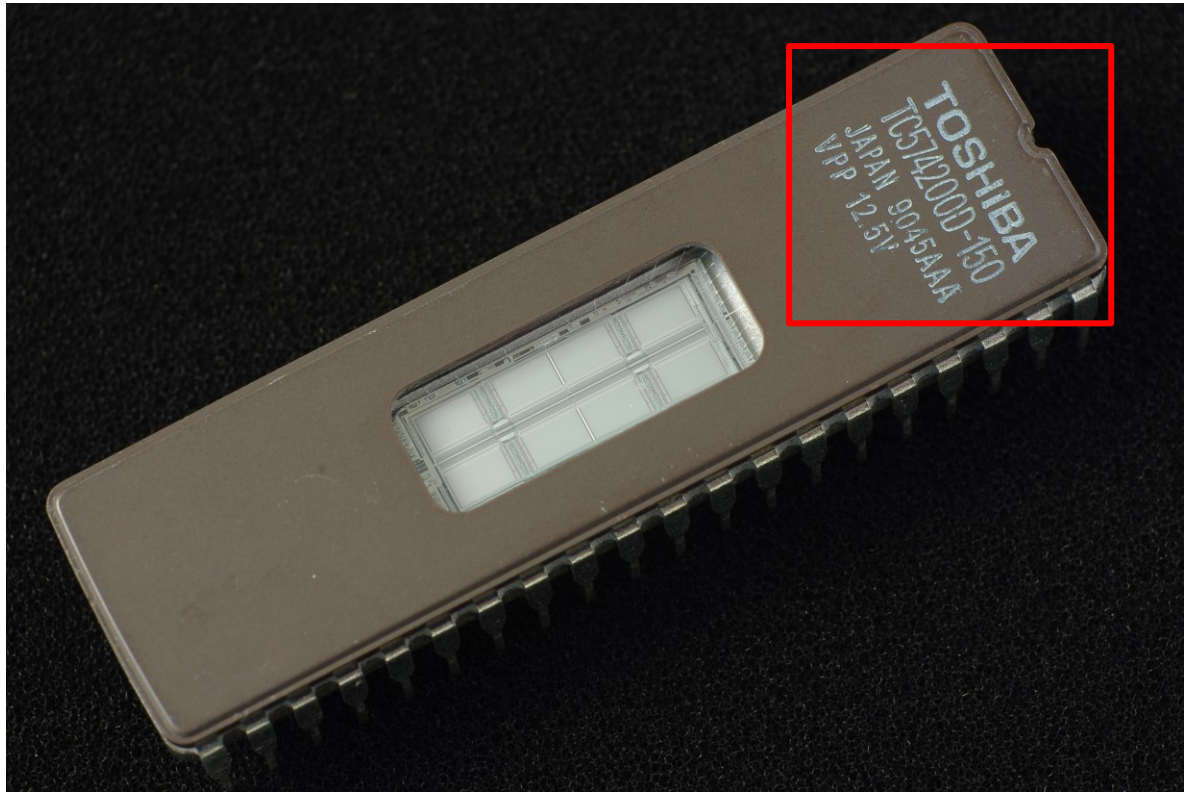
- Registers (in the CPU)
- Cache memory (in the CPU)
- Random-access memory

May exist noticeable delay between power down and total loss of stored data

- Used to be possible to retrieve remnants of data during such window
  - e.g., Cold boot attacks against DDR and DDR2 memory technology (link)



Non-Volatile Memory in Computer Systems



Example of non-volatile memory

- EPROM / EEPROM; (electrically) erasable programmable read-only memory)
- Flash memory (NOR flash, NAND flash)
  - Solid-state drive, USB flash drive
  - Serial flash memory
- Hard disk drive
- Optical disc / Magnetic tape

Data can be electrically accessed or mechanically accessed

So how do we get data off these memory devices?

Random Access Memory

Random-Access Memory (RAM)

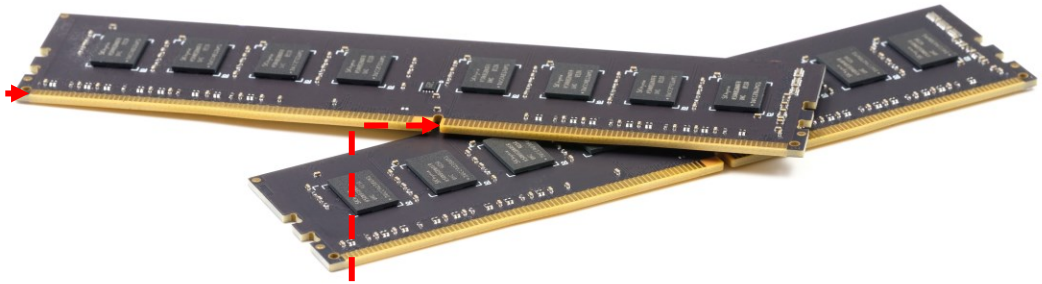
Volatile, cannot store permanent data

- Requires power to retain contents
- Data is lost when powered down

Used to store programs and data in current use

Memory modules are rectangle in shape

- Lined with connectors at the bottom
- Notch to guide installation into a RAM slot



Some Tools for Dumping Memory

Belkasoft Live RAM Capturer (https://belkasoft.com/ram-capturer)

Magnet DumpIt! (https://www.magnetforensics.com/resources/magnet-dumpit-for-windows/)

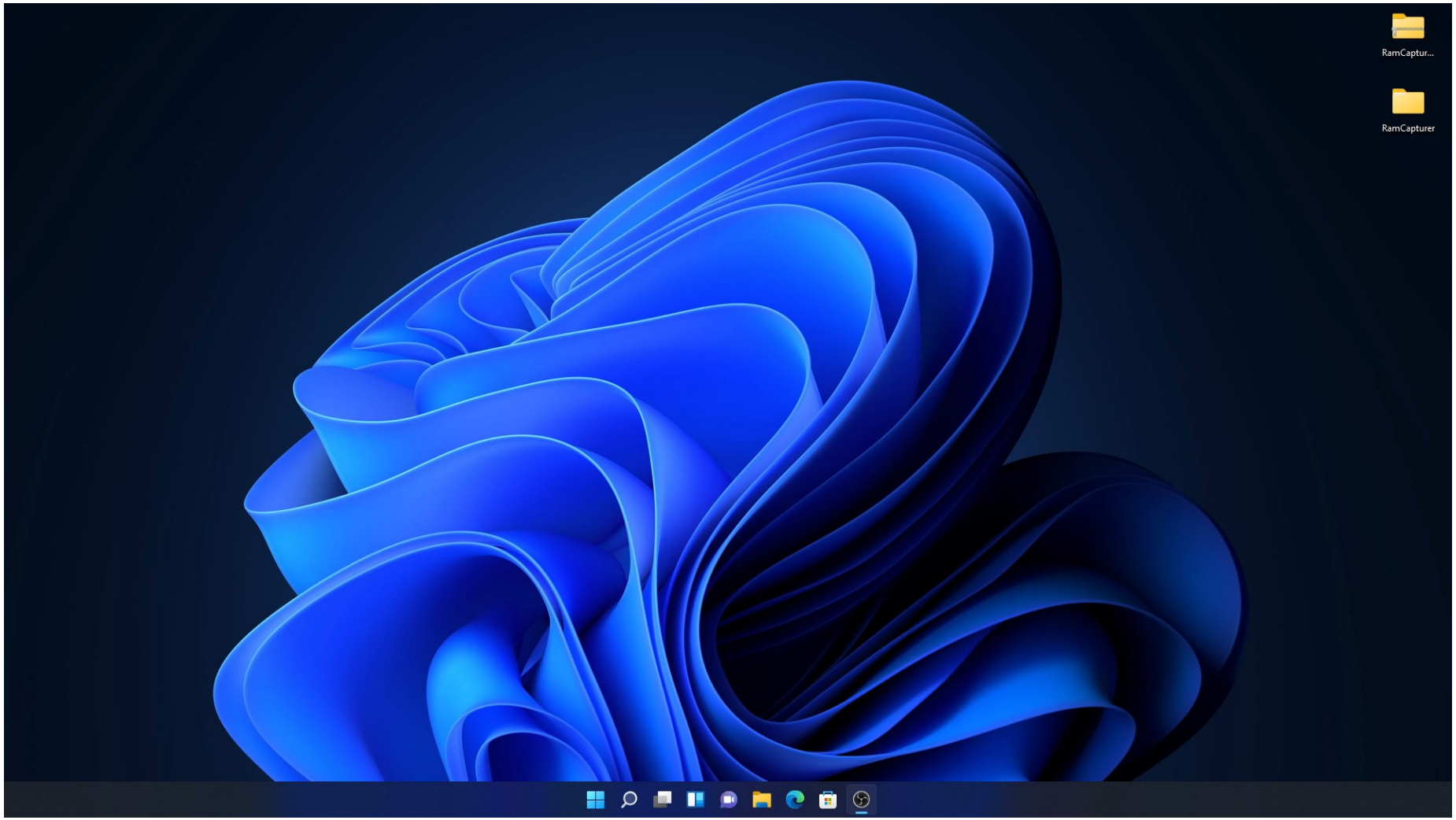
Magnet RAM Capture (https://www.magnetforensics.com/resources/magnet-ram-capture/)

WinPmem, OSXPmem, and LinPmem (https://winpmem.velocidex.com/)

PMDump (https://vidstromlabs.com/freetools/pmdump/); dump memory from process

FTK Imager (https://www.exterro.com/ftk-imager); requires installation

Capturing RAM - Video

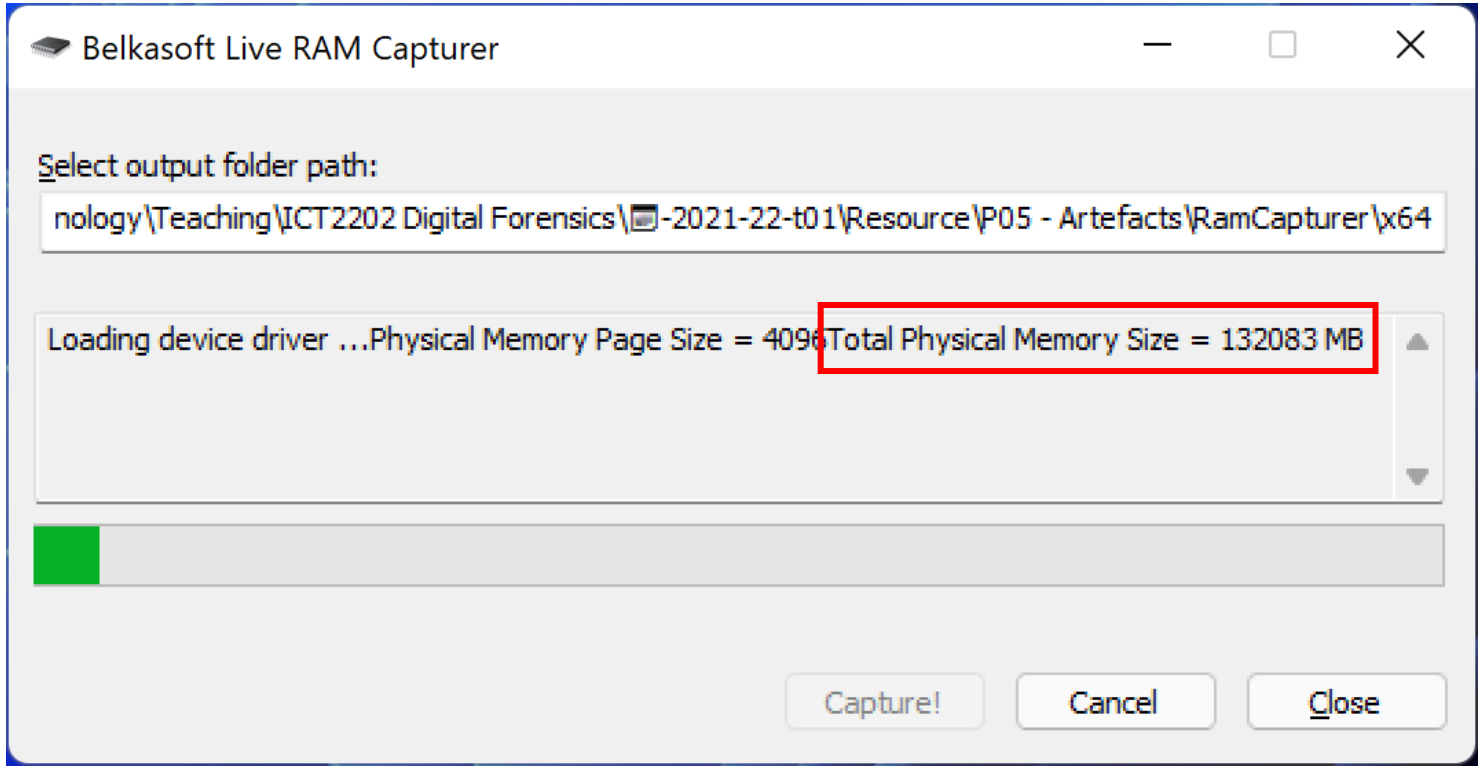


Considerations for Capturing RAM

Be mindful of the size of a RAM dump!

Most systems nowadays have ~8 to 16GB RAM

Server systems can go up to hundreds of GB

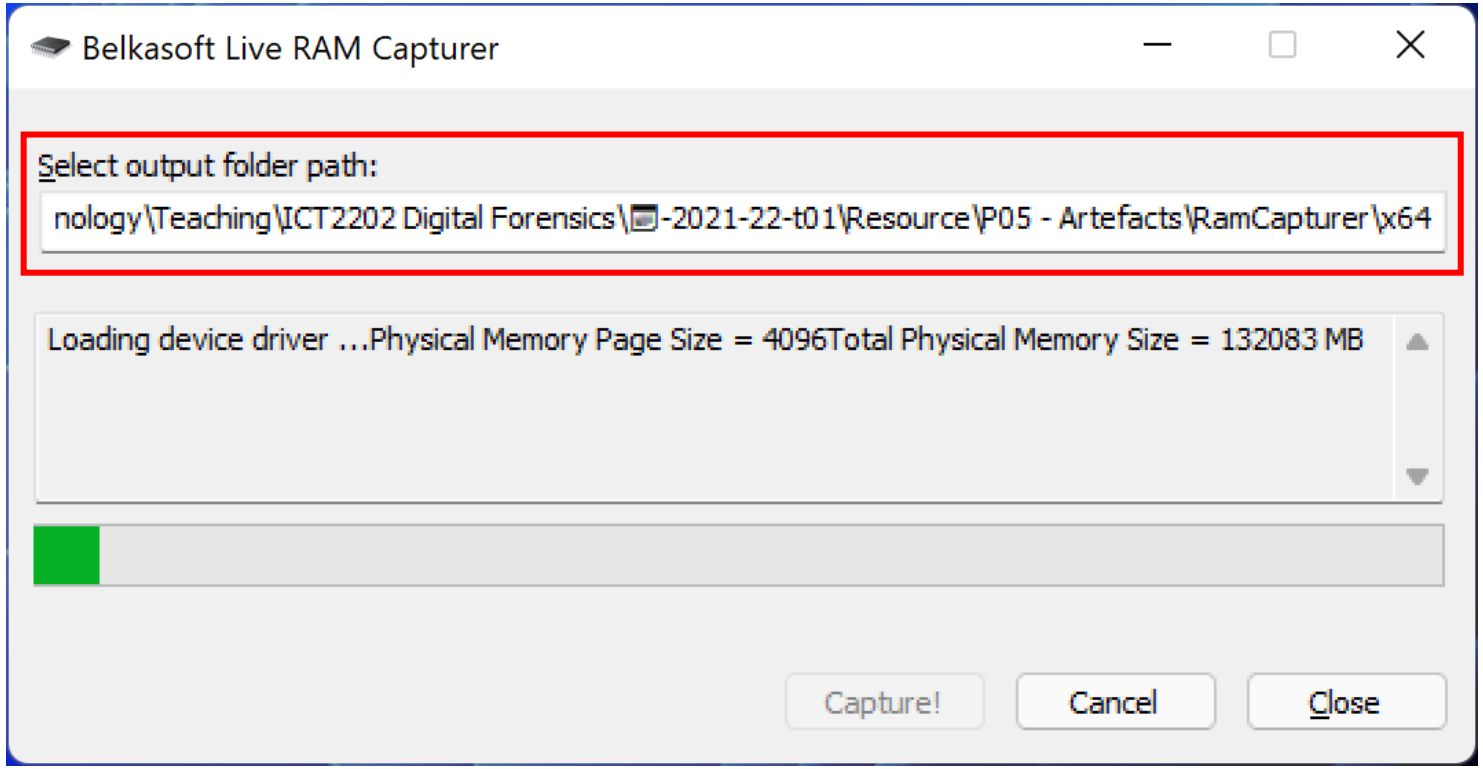


Considerations for Capturing RAM

Be mindful of where you are dumping your RAM to!

Dumping RAM to the hard disk or SSD of the computer may overwrite evidence present on those storage devices

It is suggested that you dump RAM to an external storage device that you control

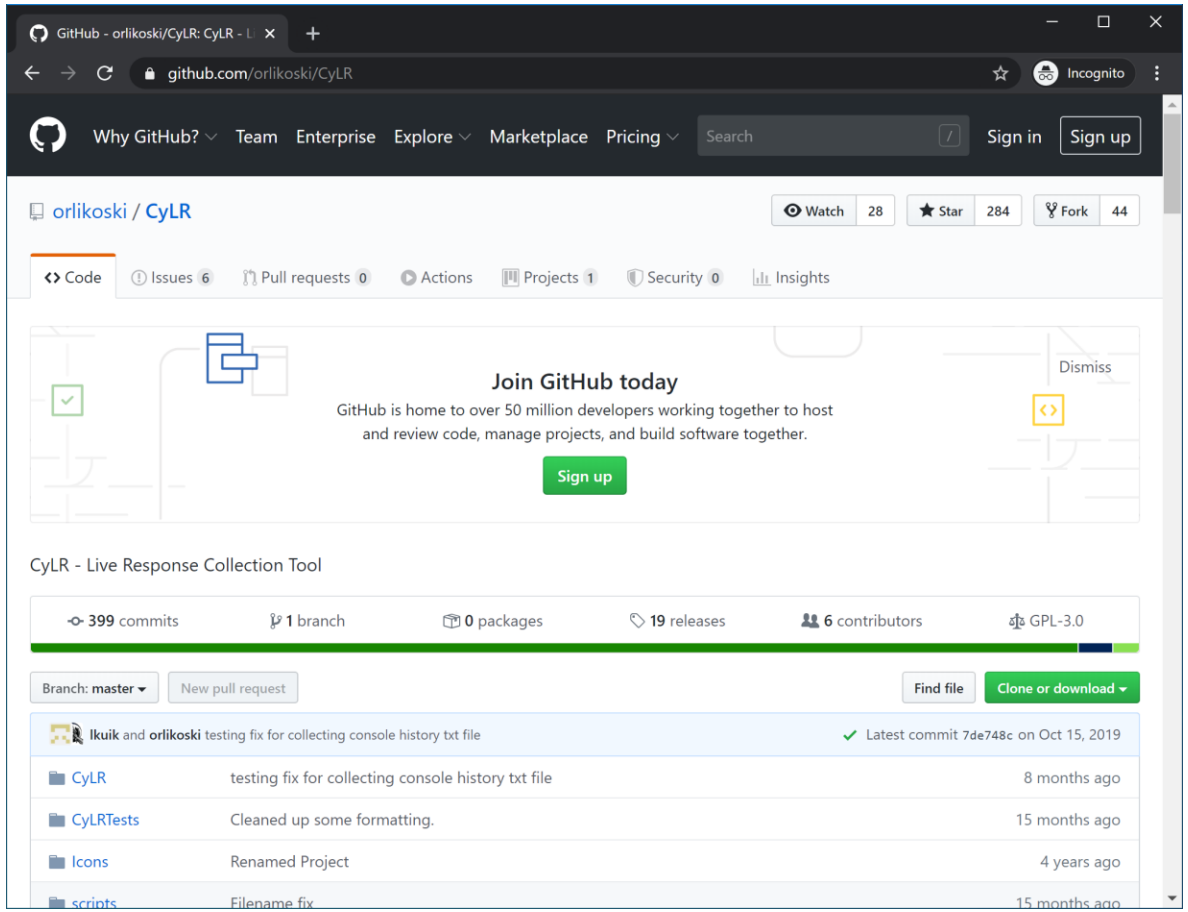


Quick Note on Dumping Artifacts from a Live System

Tools for dumping artifacts from live systems are usually light, e.g., click-and-run, minimal options

- e.g., Kroll Artifact Parser and Extractor (KAPE) and Belkasoft Live RAM Capturer are designed for standalone execution
- e.g., CyLR will collect a default set of artifacts when run without options

What happens *after* we dumped the artifacts?

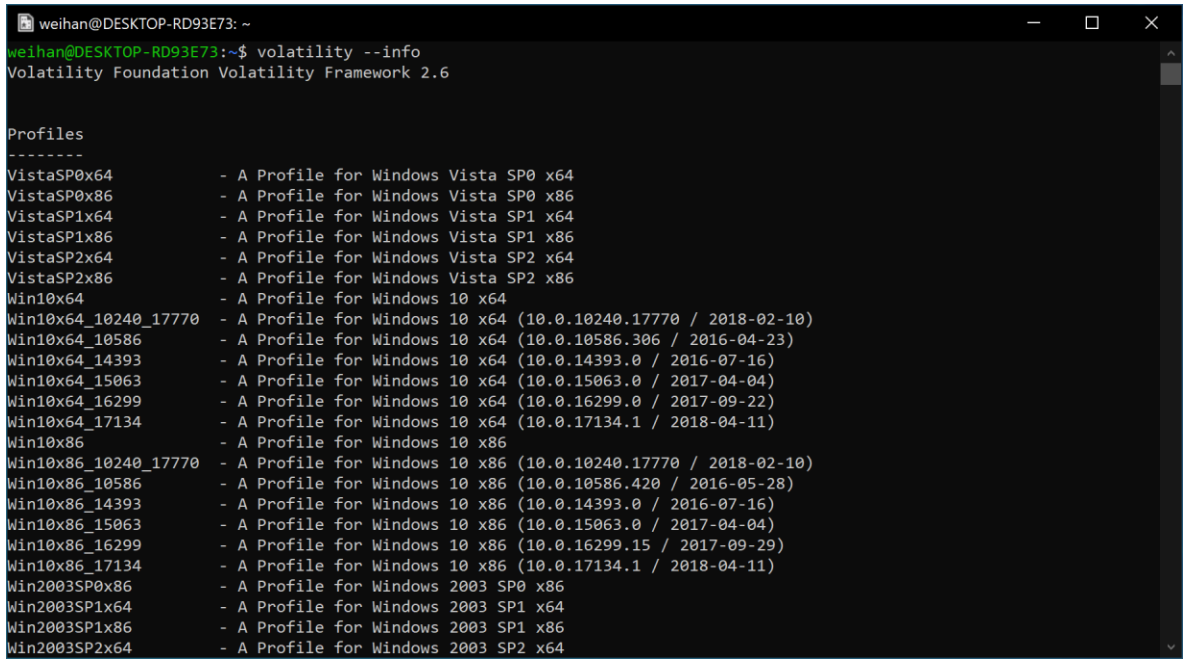


A Note About Volatility

Volatility is a memory analysis framework; is the primary workhorse for memory forensics

Kali Linux no longer provides Volatility (even if it does, the version would likely be obsolete)

- Will need to manually clone the latest Volatility repository from the official GitHub page
  - **Volatility 3:** https://github.com/volatilityfoundation/volatility3
  - **Volatility 2:** https://github.com/volatilityfoundation/volatility



Memory Analysis - Video

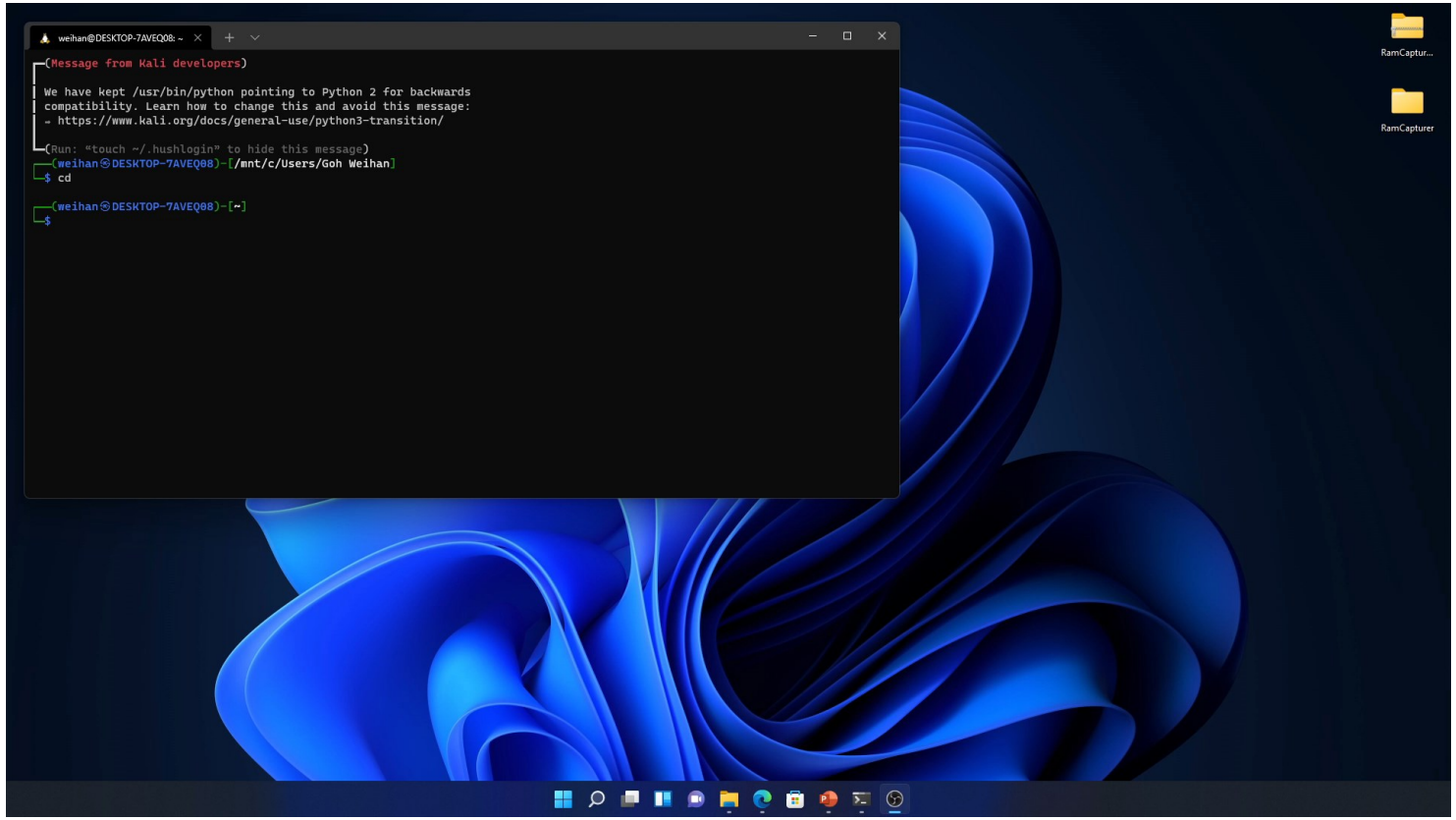
Memory dumps may contain information of interest, captured at a particular **state of time** on the machine

List of running processes

List of network connections

Registry values

etc.

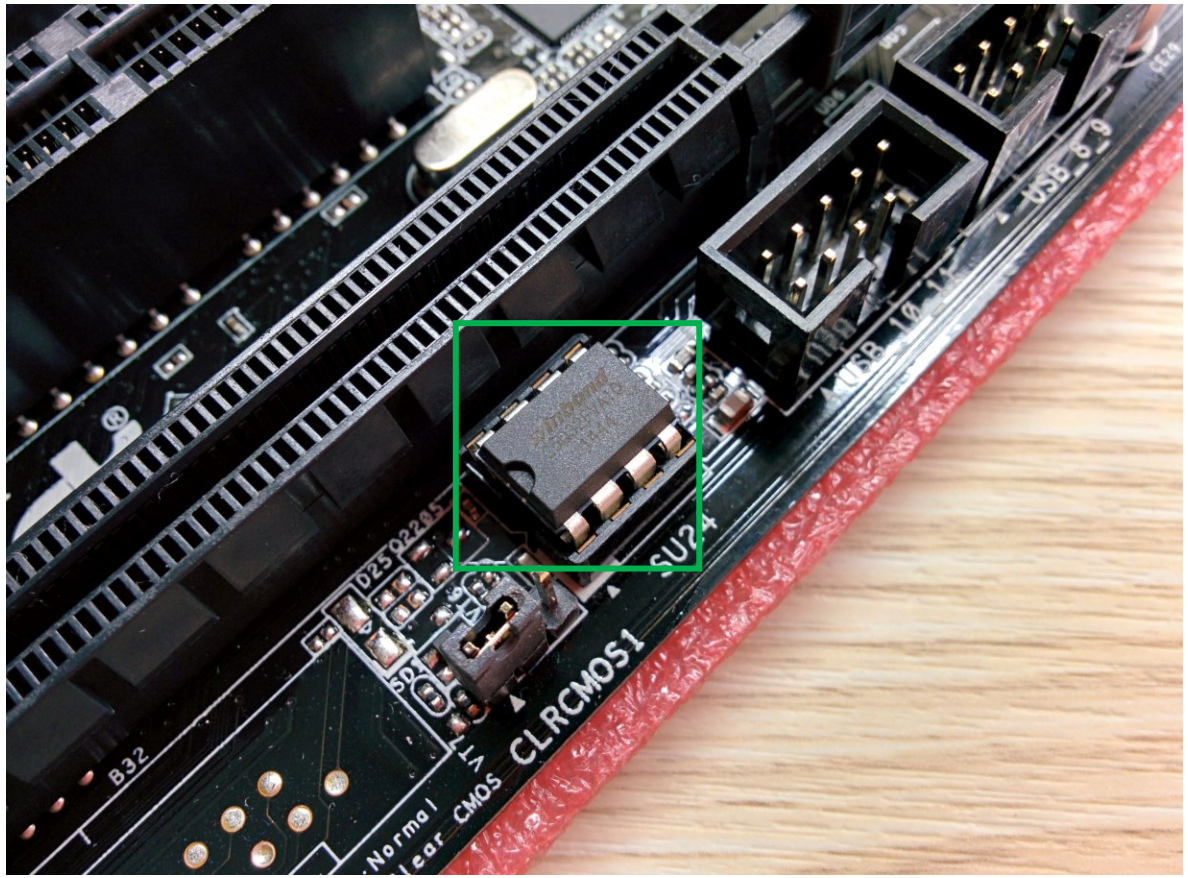


Firmware

Firmware in Computers

Computer motherboards

- Has a chip that stores firmware (e.g., BIOS, UEFI, etc.) required to start the computer
- Hosts the BIOS (Basic Input / Output System) or UEFI (Unified Extensible Firmware Interface)
- The firmware itself is non-volatile / persistent across reboots
  - Typically stored on EEPROM / flash memory (EPROM had been used in the past)



Firmware in Computers

Maintains information like real-time clock in a separate, volatile memory

- Backed by battery
  - Typically, a 3V, CR2032 cell battery
  - Old RTC chips may have their battery integrated with the chip

Such volatile memory may be a

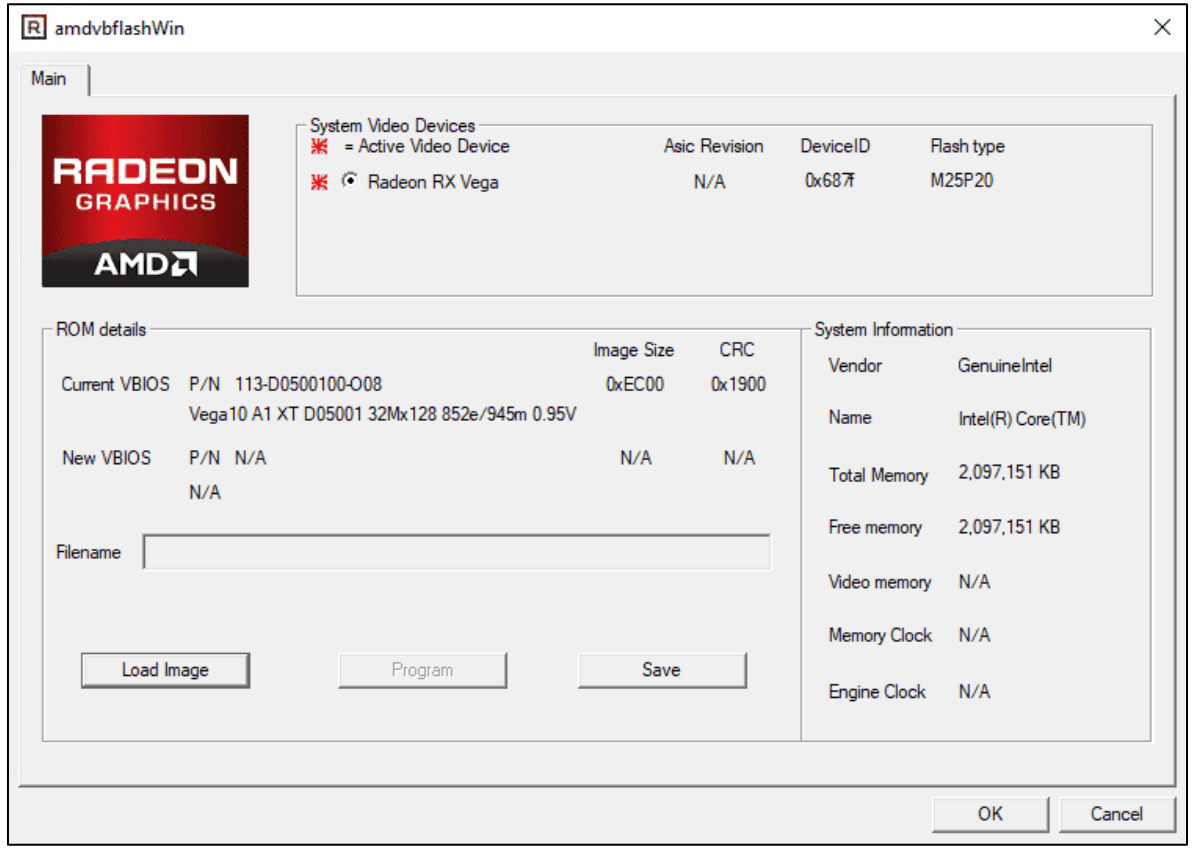
- Separate chip on older computers
- Integrated into the motherboard chipset on more modern systems
  - e.g., Intel 100-series PCH integrates a Motorola MC146818B-compatible RTC with 256 bytes of battery-backed RAM

Firmware in Computers

Other sources of firmware

- Graphics card / integrated graphics controller
- Storage controller
- Network controller

Firmware may be retrievable via tools or memory map approaches, or through sysfs filesystem, or chip-off



Why Interest in Firmware?

Additional data may be inserted into the firmware by external entities

- e.g., Inserting an SLIC table to activate Windows 7 by means of a manufacturer key

Can be a platform for viruses / malware

- e.g., CIH virus (a.k.a. Chernobyl), in the wild; corrupts BIOS
- e.g., Persistent BIOS infection

Useful in supply chain attacks

Storage Media

Hard Disk Drive

Mechanical data storage device

- Stores data on rotating **platter(s)** within the disk body
- Read / write performed by a **head** on a platter
- The head is connected to an **arm**, which is moved by an **actuator**
- Movement of the arm is not done using motor, instead using a **voice coil**

Hard Disk Terminology

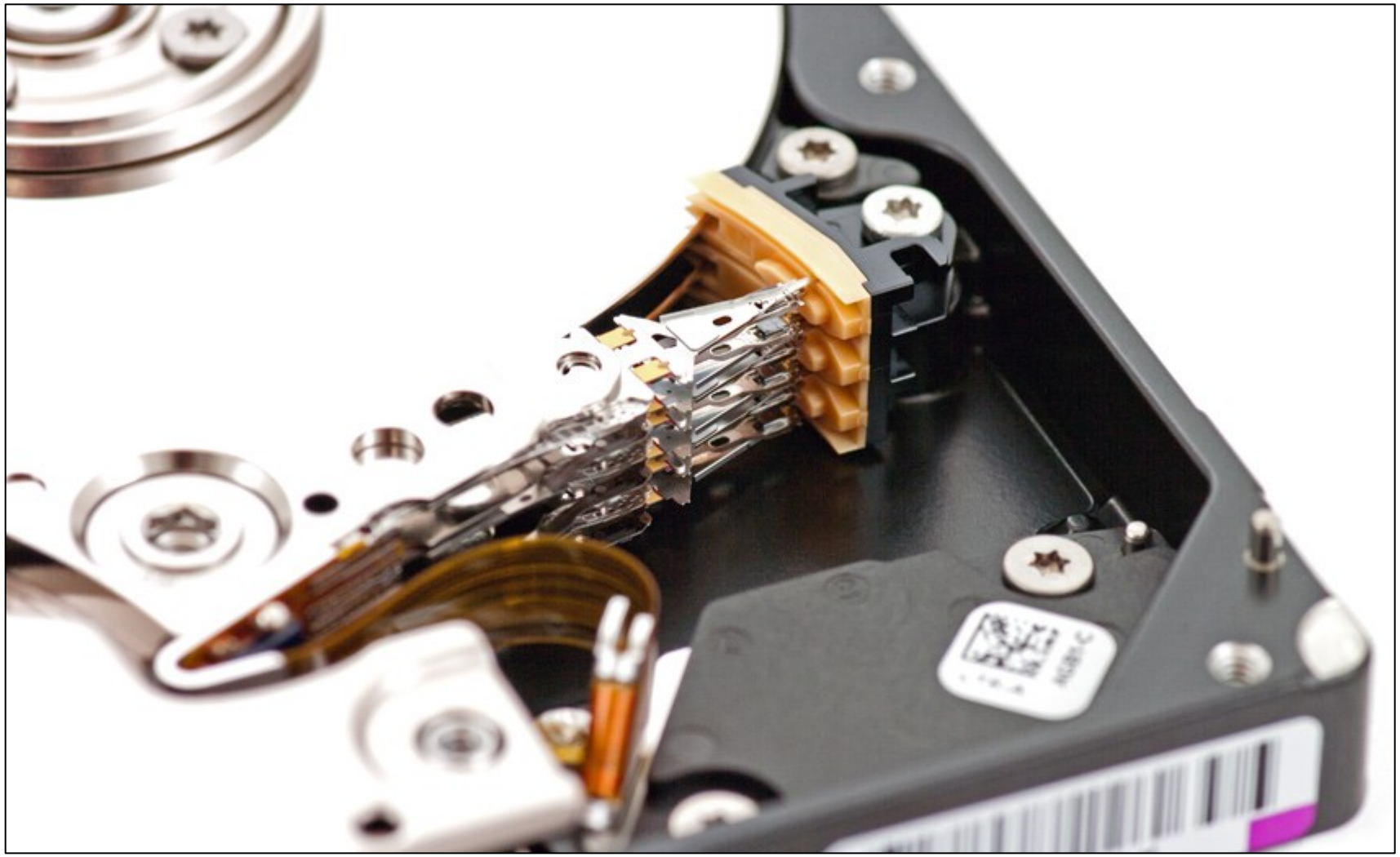
Platter: Circular disc used to store data

Spindle: Component holding all the platters in the disk

Actuator: A non-motor component that moves the head(s) across the platters

Hard Disk Terminology

Head: Performs read / write by converting the platter's magnetic field into current (read), or current into magnetic field (write)



Hard disk drive heads, parked

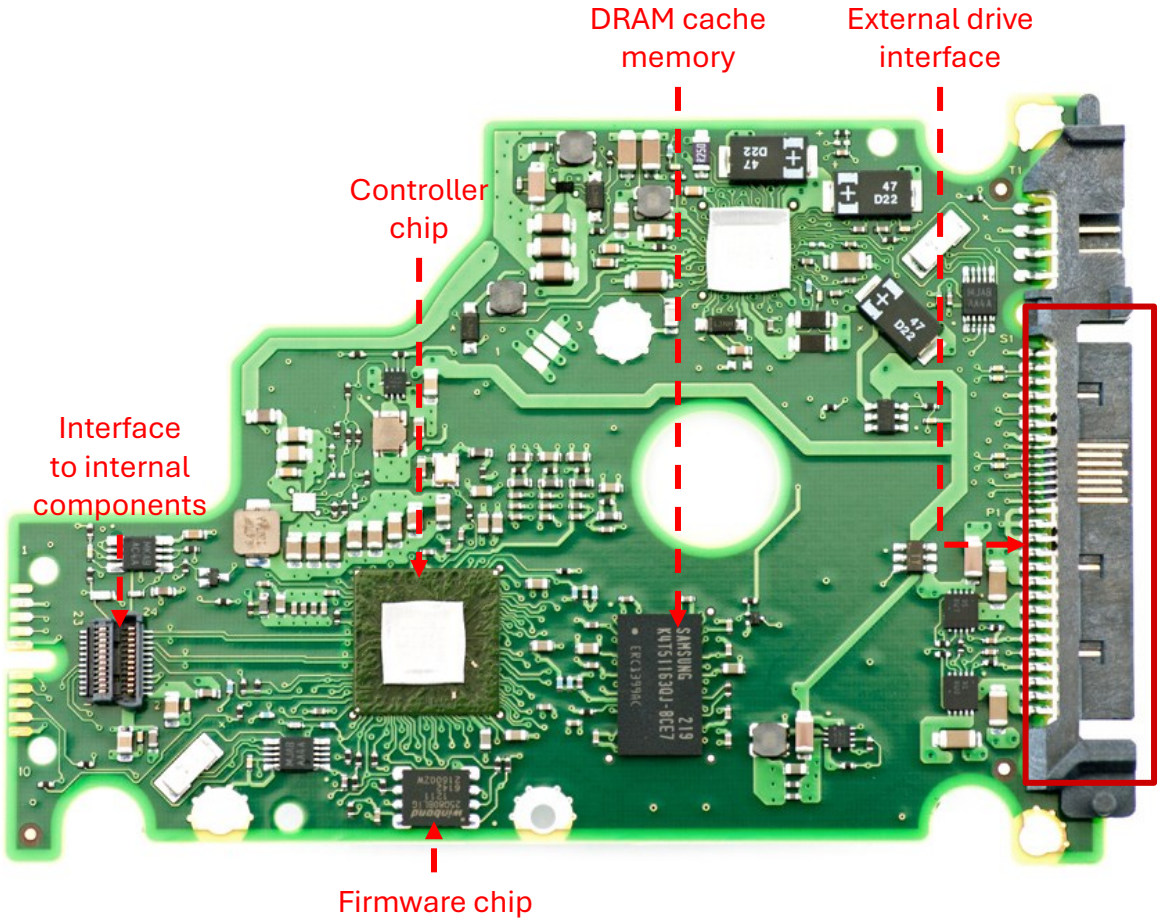
Hard Disk Controller Board

Activity managed by a controller board on the disk

- The controller also manages the drive's interface

Controller board contains among other things

- The disk controller chip
- Firmware chip
- Cache memory
- External drive interface (connects to motherboard)
- Interface to drive internals



Solid-State Drives

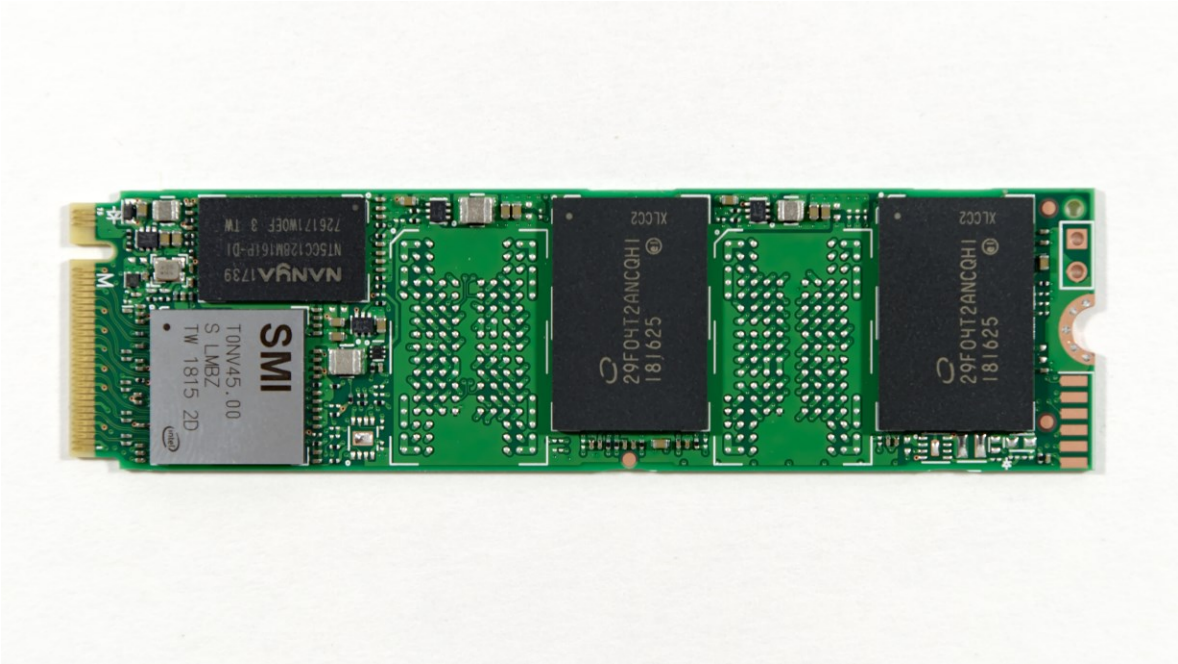
Entire drive is electronic, with **no mechanical components**

Much faster than a hard disk

Typically utilize **NAND flash memory** to store contents

Flash memory is managed by the **SSD controller**, which also manages the drive's interface

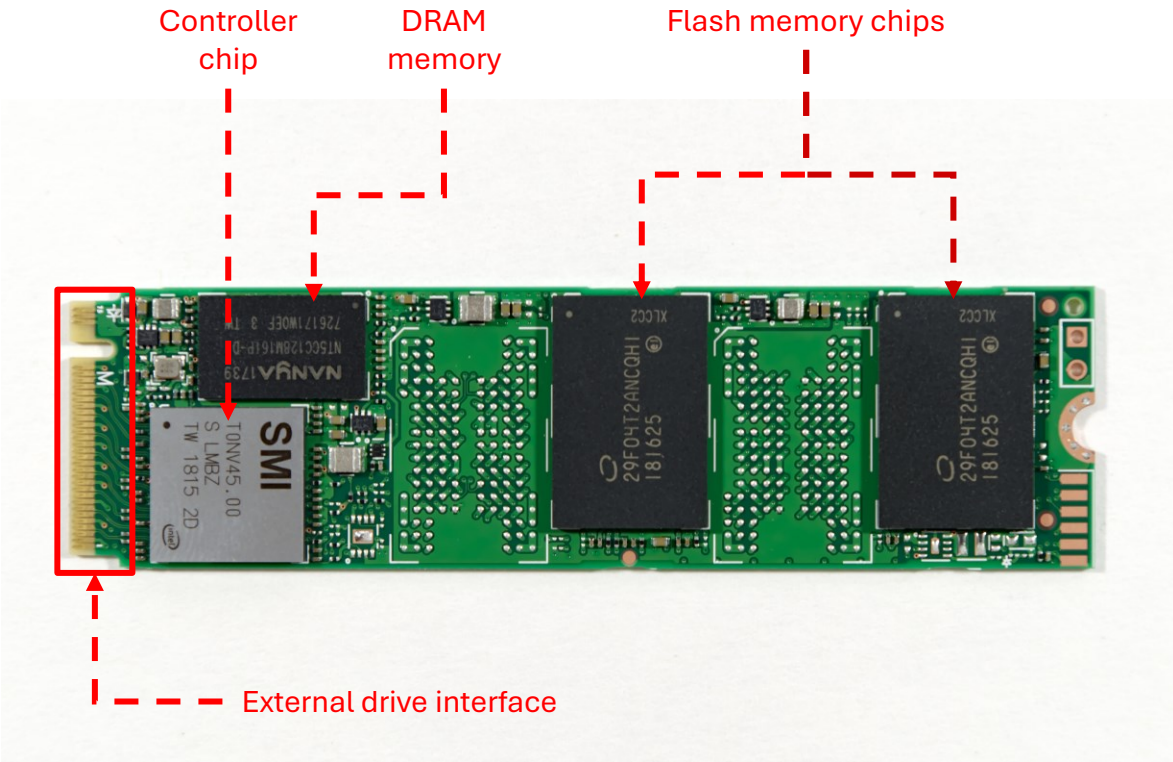
Some SSDs have **DRAM** to improve performance



Solid-State Drives

Components on an SSD

- NAND flash memory chips
- SSD controller chip
- DRAM chip
- Drive interface (connects to motherboard)



Common Hard Disk Drive Physical Interfaces

AT Attachment (ATA) or Integrated Drive Electronics (IDE)

Sometimes called Parallel ATA

A device can be configured as **master** or **slave**

Each IDE port on the motherboard supports **one master and one slave device**

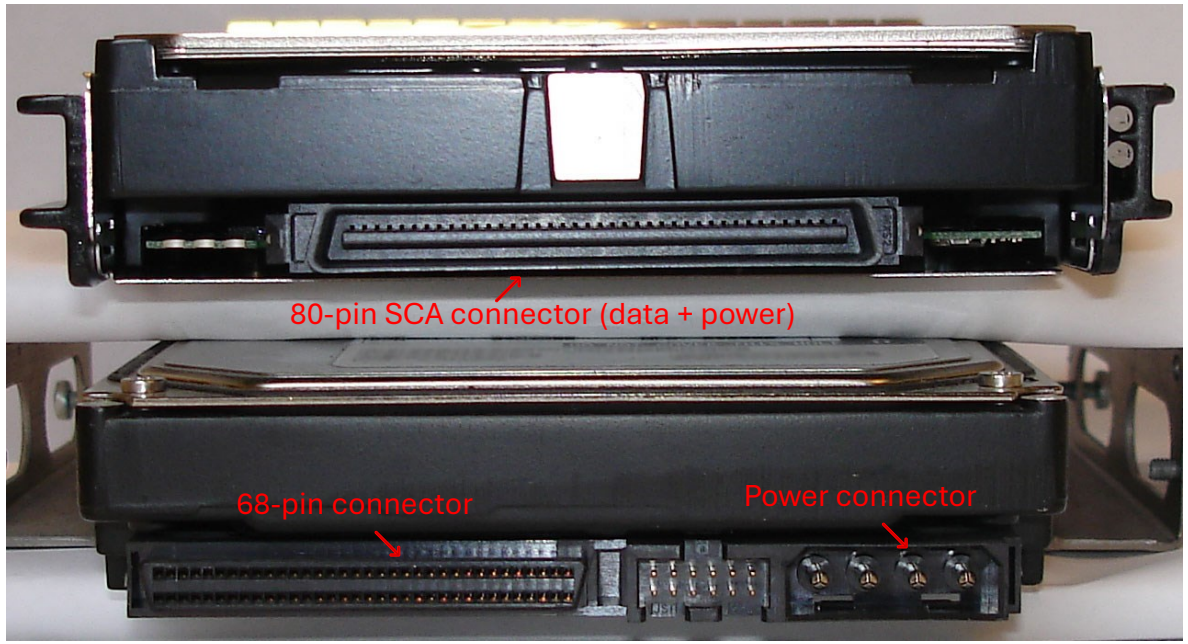
Common Hard Disk Drive Physical Interfaces

Serial AT Attachment (SATA)

Each SATA cable connects **one device to one SATA port** on the motherboard

Common Hard Disk Drive Physical Interfaces

Small Computer System Interface (SCSI), 68-pin and 80-pin

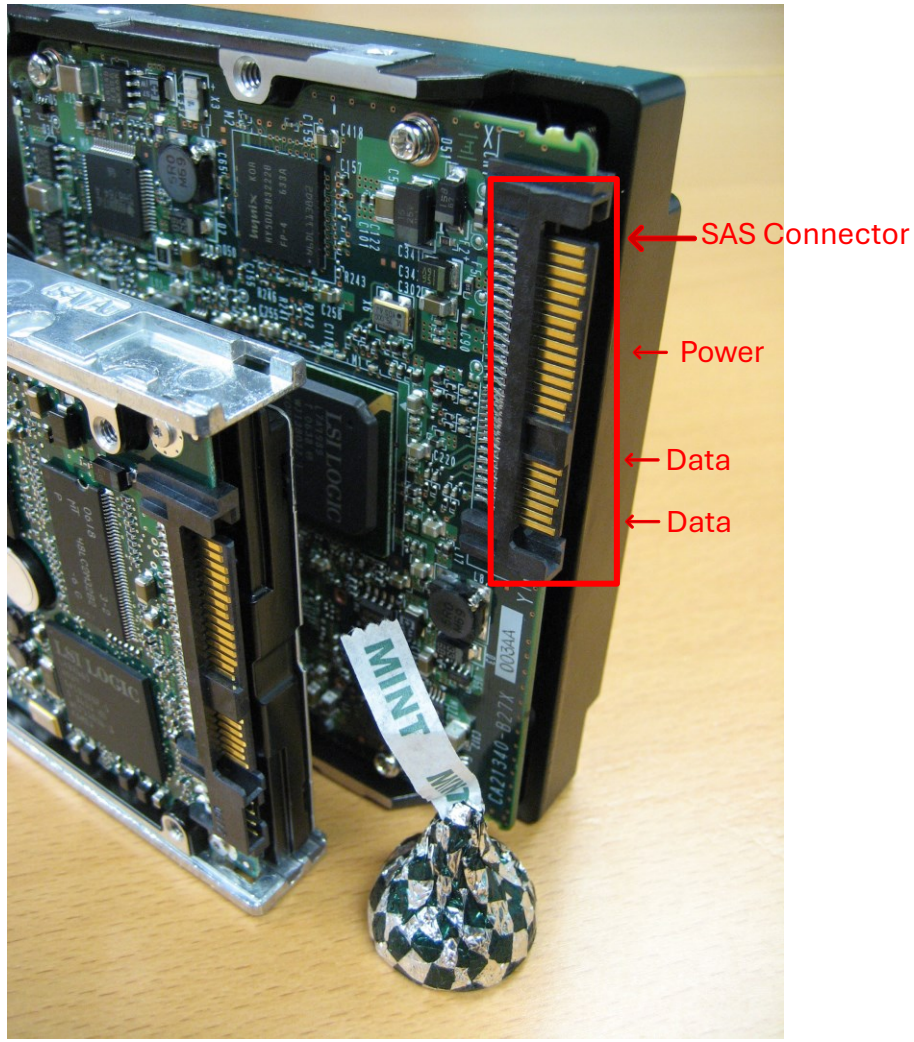


Common Hard Disk Drive Physical Interfaces

Serial Attached SCSI (SAS)

SAS disks **are not physically compatible** with SATA ports

One cannot simply connect a SAS disk to a SATA port



Common Hard Disk Drive Physical Interfaces

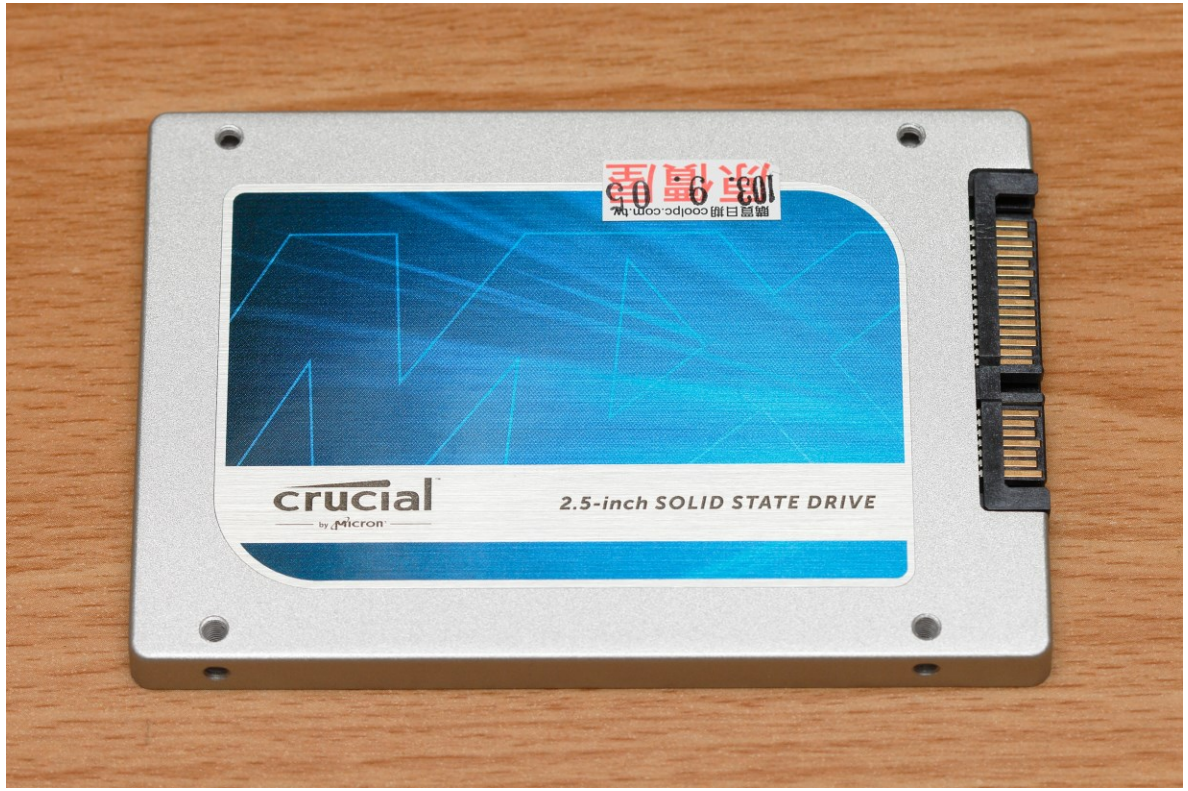
Universal Serial Bus (USB)

Many different types of USB connectors, e.g., micro-USB, micro-USB 3.0, USB-C, etc.

Common Solid-State Drive Physical Interfaces

Some disk interfaces are also used for solid-state drives (SSD), such as

- Serial ATA (SATA)
- Universal Serial Bus (USB)
- Serial Attached SCSI (SAS)
- AT Attachment (ATA) or Integrated Drive Electronics (IDE)
  - Rare, but it exist

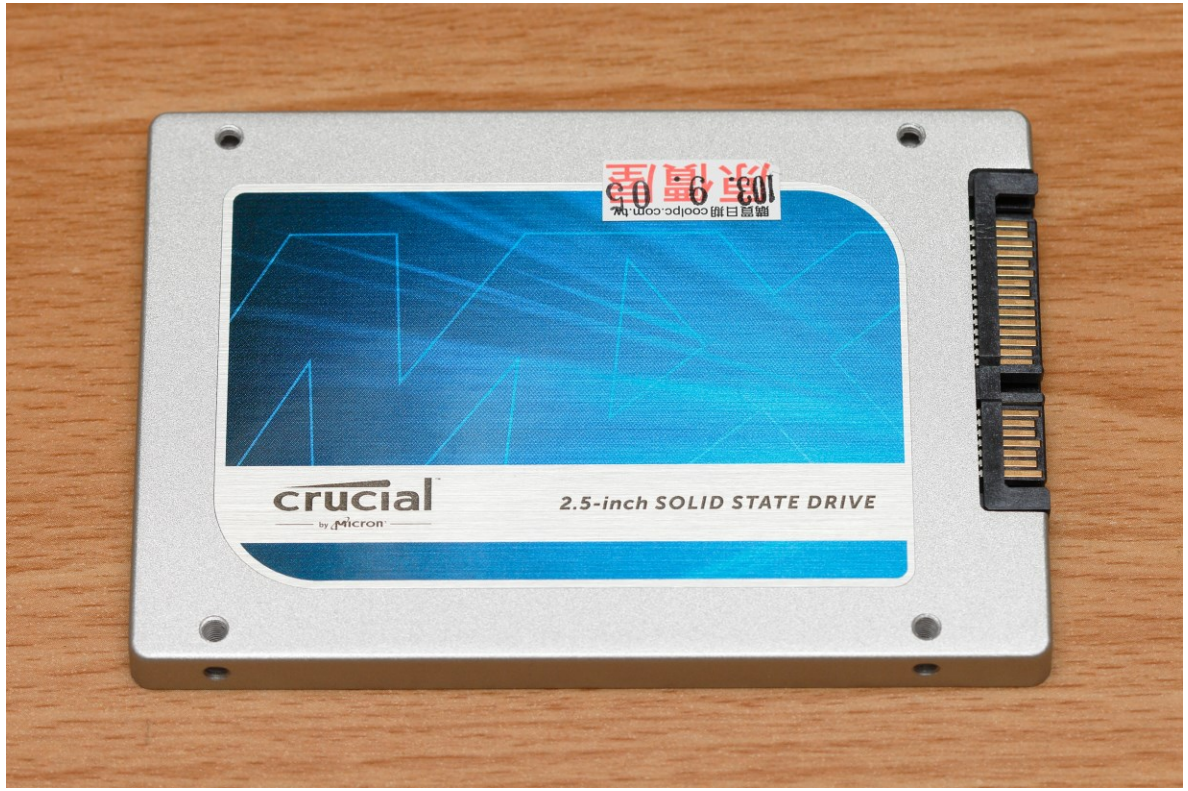


Common Solid-State Drive Form Factors

2.5" Serial ATA

One of the most common form factor for SSD; uses the SATA connector

Form factor and connector type is similar to 2.5" SATA hard disks commonly used in laptops



Common Solid-State Drive Form Factors

mSATA

Small factor SSDs about 1/8 the size of a standard 2.5-inch drives

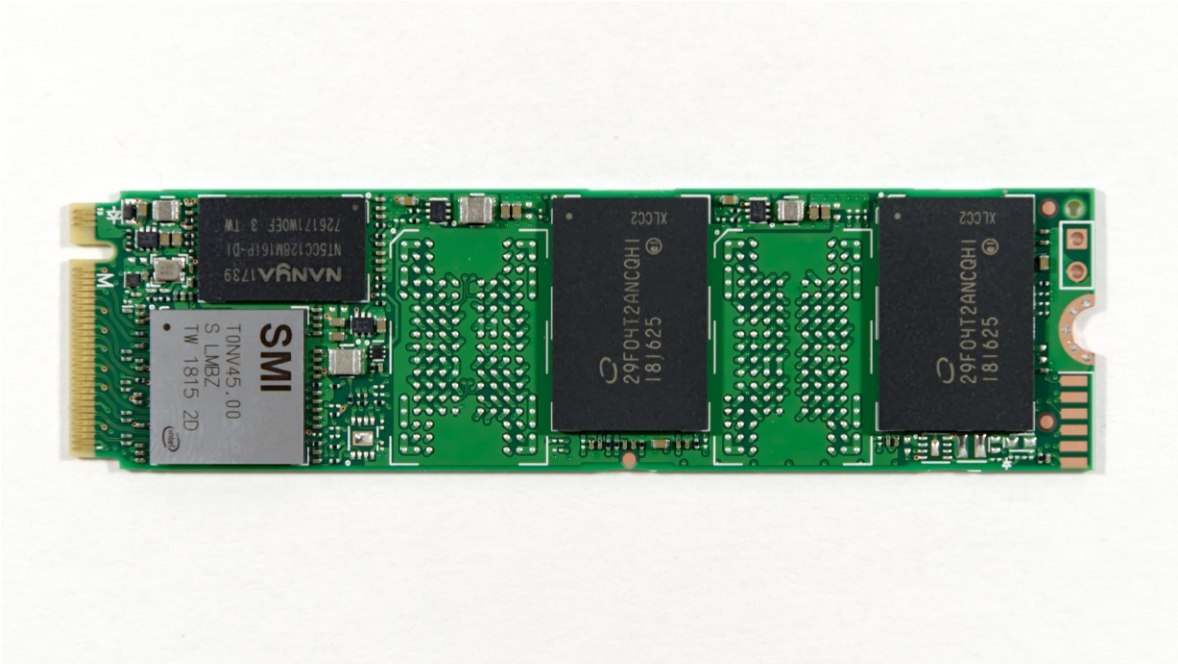
Uses the SATA protocol

Common Solid-State Drive Form Factors

M.2 (formerly NGFF, or *Next-Generation Form Factor*)

M.2 defines the drive physical interface

The drive protocol may be **SATA** or **NVMe**; can be identified via the notch configuration on the physical interface

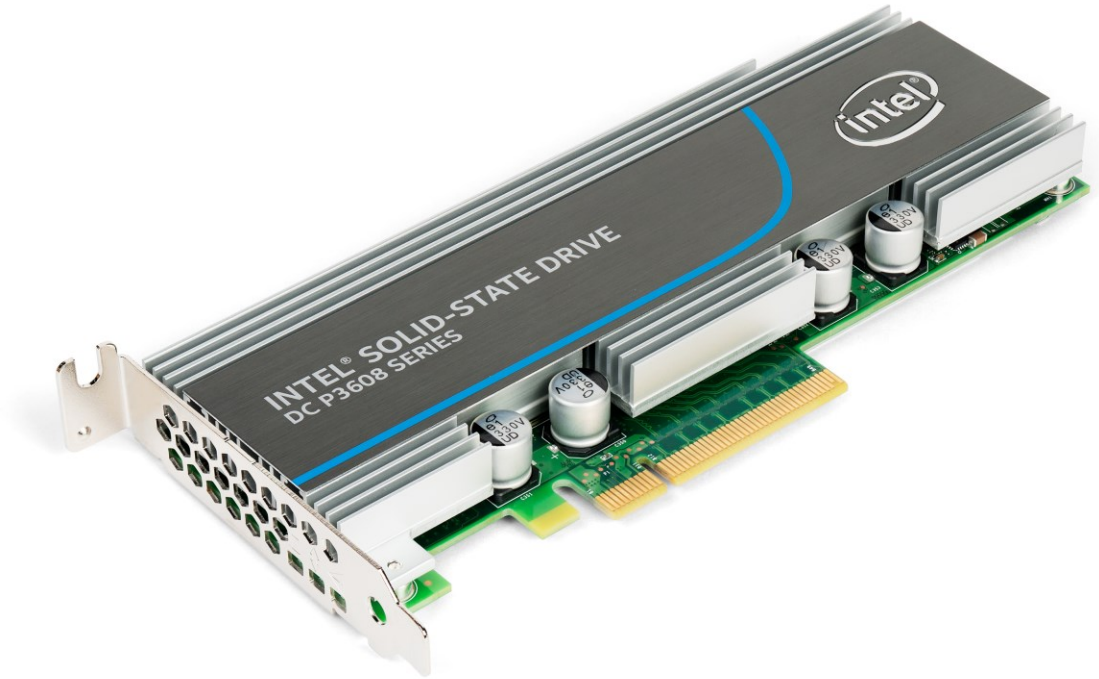


Common Solid-State Drive Form Factors

PCI Express (PCIe)

NVMe drive protocol typically, but need to check with documentation for certainty

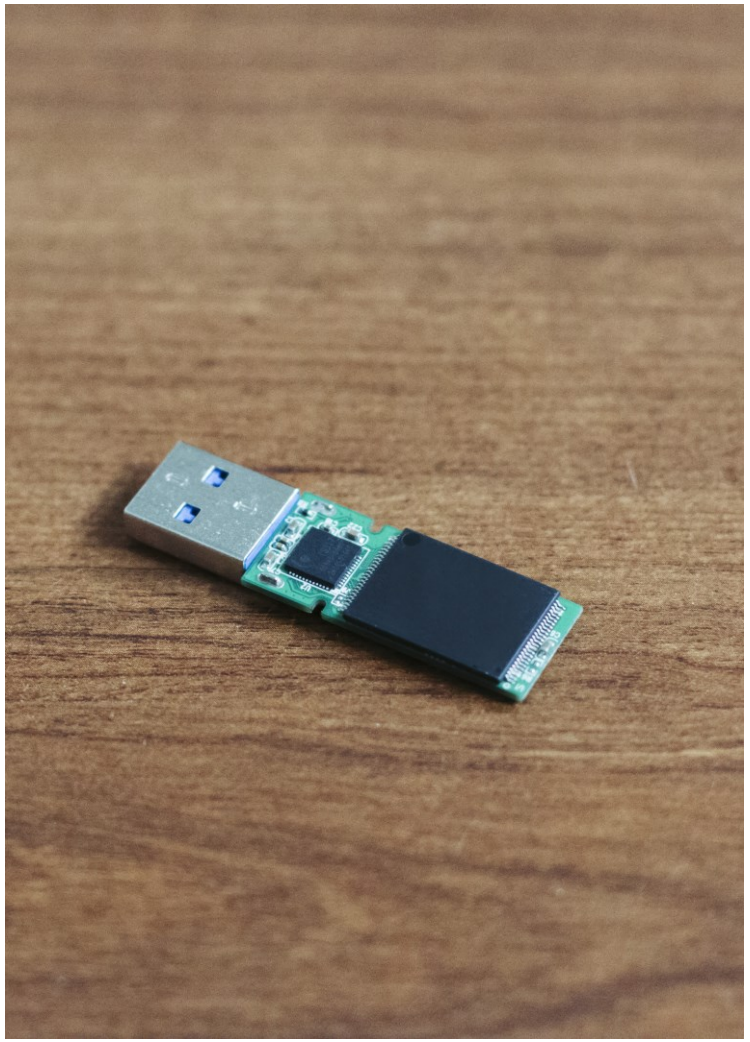
There are PCIe SSDs that do not conform to the NVMe drive protocol



USB Flash Drive

Like SSDs, they use NAND flash memory for storage

- Has a controller to manage memory and interface
- Typically, much slower than SSD, but more ubiquitous
- Small, cheap, and readily available



Speed of data transfer may impact acquisition time

- Limited by speed of the USB interface, though typically (very) much slower in practice
  - **USB 2.0** theoretical maximum 480Mbps
  - **USB 3.0 / USB 3.1 Gen 1 / USB 3.2 Gen 1x1** theoretical maximum 5Gbps
  - **USB 3.1 / USB 3.1 Gen 2 / USB 3.2 Gen 2x1 / USB 3.2 Gen 1x2** theoretical maximum 10Gbps
  - **USB 3.2 Gen 2x2** theoretical maximum 20Gbps
  - **USB4 Gen 3x2** theoretical maximum 40Gbps

Data Acquisition from Storage Media

Preparing to Handle Storage Media

Hardware equipment consideration

- Media duplicators for your target storage media interfaces
- Write blockers for your target storage media interfaces
- Live bootable disks (e.g., Kali Linux, SIFT Workstation, etc.)
- Data cables (e.g., SATA to USB, IDE to USB, etc.)

What if you do not have access to a media duplicator or write blocker? 💾

Preparing to Handle Storage Media

What if you do not have access to a media duplicator or write blocker?

Configure operating system not to mount storage media automatically

- May not offer true write blocking
- Settings may not be persistent
  - Always check with a "dummy" disk before connecting evidence disk
- May not be possible with some operating systems

Kali Linux in *Forensics Mode* is an option, but always verify before use

Disk Access

Disk access is done in units called *sectors*

Due to how physical media is designed, a sector can mean either

- **Physical sector**, i.e., unit for which the physical disk operates, and is the smallest unit the disk reads and writes in
- **Logical sector**, i.e., the unit that is used when the computer addresses the media, and is the smallest unit the disk can be addressed

To read from or write to a disk, one specifies the position on the disk by using a *logical sector number*

Disk Access

A sector (both physical and logical) on a disk used to be 512 bytes, until Advanced Format hard disks introduced 4096-byte physical sectors

Some kind of transparent translation required to support legacy systems

Similar translation required for other storage media such as SSD, USB disks, etc.

How?

- The disk exposes 512-byte logical sectors to the computer via emulation
- Computer accesses disk as though it has 512-byte logical sectors
- In the disk, operations are performed on 4096-byte physical sectors

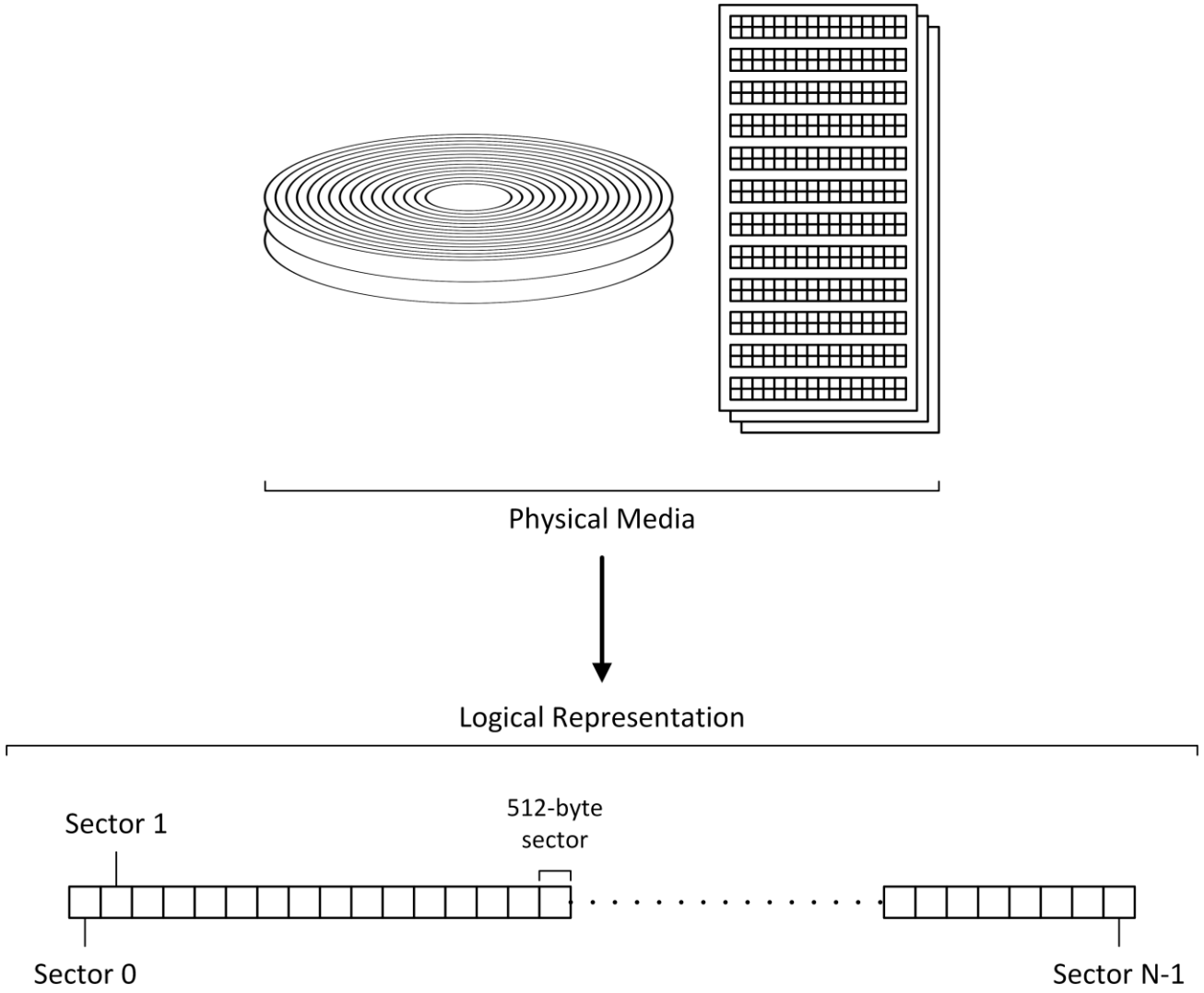
Logical Block Addressing

Modern media are addressed via a linear addressing scheme called Logical Block Addressing

Each LBA address points to a single logical sector on the media

The media appears as an array of *N* LBA sectors, starting from LBA 0 and ending at LBA N-1

When we refer to sectors, we refer to LBA sectors



Acquiring a Disk Image from Storage Media

Create a bit-for-bit copy of the media, resulting in a disk image file containing

- Partition table
- Partitions and filesystems
- Slack and unused space
- Deleted data
- Unallocated and unusable space

Disk image files are files

Disks are also files in Linux

- **Specifically, disks are *block special files* / *block devices* in Linux**

Working with original data should be avoided, even if possible 💾

Instead, work on forensically sound *working copies* 💾

How? 

Safeguarding Forensic Soundness

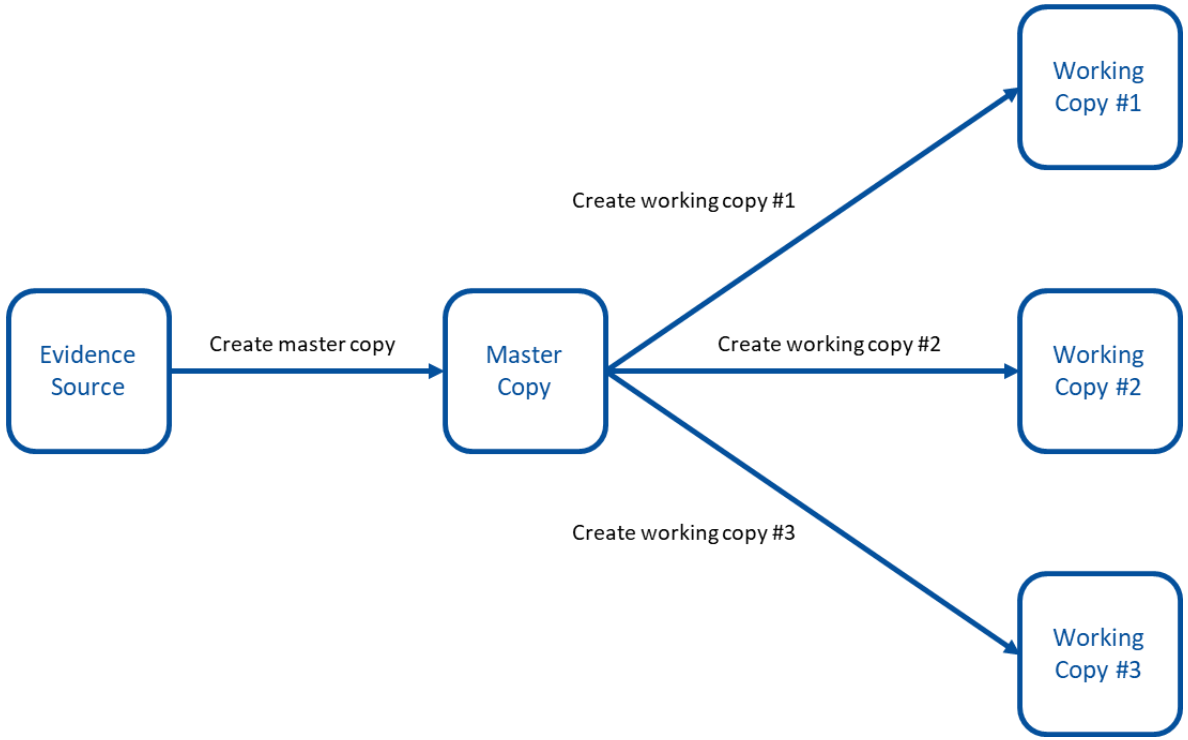
Create a forensically sound copy from the evidence source, and designate this as the *master copy*

Ensure that the hash value of the *master copy* matches that of the evidence source

- If not the case, document the reasons for it

Create *working copies* from the master copy

- Ensure hash of working copies matches the master copy / evidence source
- Ensure each copy is tagged and can be identified



Safeguarding Forensic Soundness

Why working copies?

- Fallback mechanism; if something goes wrong, can use another working copy (desirable) or create another working copy from master copy (not so desirable)
- Applying a technique may introduce computationally irreversible changes; can use another working copy to attempt other techniques
- Investigation does not end due to loss of working copy

Verifying Master and Working Copies

Verify master and working copies against the evidence source by comparing their hash values

Recap: If two files have the same hash value, it implies that their contents are identical 🙂

Example

- Hash value of `/dev/sda` (i.e., the evidence disk) is
  - **`fa3a9aa91b1b7ddf30a6159dd78f04f7428e349d`**
- Hash value of `sda.img`, a master copy created from `/dev/sda` is
  - **`fa3a9aa91b1b7ddf30a6159dd78f04f7428e349d`**
- We can therefore conclude that the contents of `sda.img` is bit-for-bit identical to `/dev/sda`
  - Bit-for-bit identical contents
  - Both are of the same size

Verifying Master and Working Copies

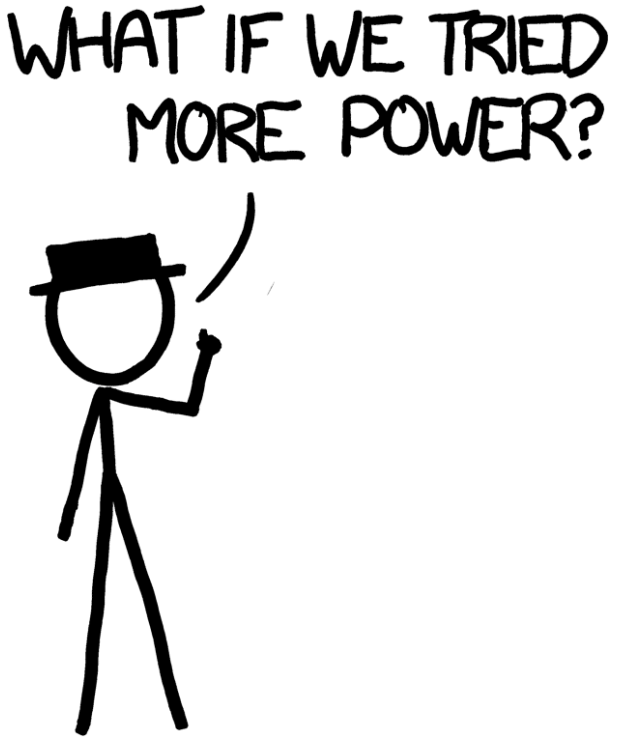
Some notes about hash functions

- Cryptographic hash functions such as those in the **SHA-2 family** are suitable candidates
  - e.g., SHA-256, SHA-512
  - **In Linux commands:** `sha256sum`, `sha512sum`, `shasum`
- Try not to use SHA-1 standalone unless necessary; its use is superseded by SHA-2
- Although it is possible to encounter the use of MD5, do not use it standalone due to existing cryptographic weaknesses
  - The use of MD5 could be due to legacy

Acquisition from Large Storage Media

Media storage size is getting larger

- Samsung PM1633a (16TB; SSD)
- HGST (WD) Ultrastar DC HC520 (12TB; HDD)
- Seagate Exos X16 (16TB; HDD)
- WD Red Pro (22TB; HDD)



Acquisition from Large Storage Media

Media storage size is getting larger

- Not practical to image
- Yet need to maintain forensic soundness
- How?

Acquisition from Large Storage Media

Media storage size is getting larger

- **Not practical to image**
- **Yet need to maintain forensic soundness**
- **How?**

Maintain forensic soundness

- Hence, should not introduce changes, i.e., no writing to media

Not practical to image

- Hence, need to work directly with evidence disk

Solution(?)

- Read directly from media, but redirect all writes to another overlay file; have a software that processes the overlay file

What technology allows this? QEMU Copy-on-Write

Acquisition from Live Storage Media

Cloud storage and systems are becoming more prevalent, e.g., AWS, Azure, Alibaba Cloud, DigitalOcean instances

May not be possible to power off live systems hosted remotely to perform imaging

How to perform acquisition?

Considerations

- Need to connect to the server
  - How to minimize overwriting?
- Need to dump data from media to somewhere?
  - How to retrieve the data from that somewhere?

Possible solution(?)

- SSH into the server, then run dd and pipe output through the SSH connection to the connecting host?

👋 Questions?
Thank You!

📨 Weihan.Goh {at} Singaporetech.edu.sg

📌 https://www.singaporetech.edu.sg/directory/faculty/weihan-goh

📌 https://sg.linkedin.com/in/weihan-goh