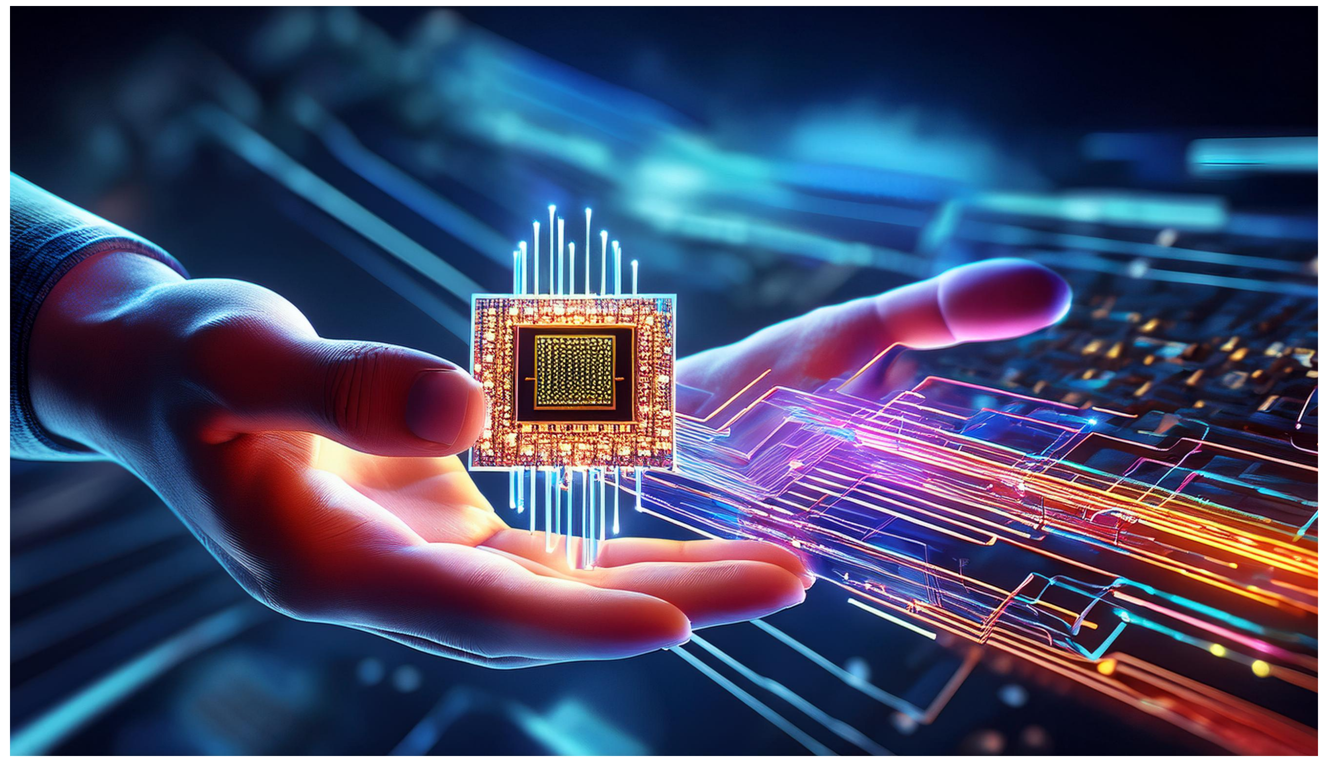
24.10.24 20:51

CSCI 250
Introduction to Computer Organisation
Lecture 3: CPU Architecture IV



Jetic Gū

2024 Fall Semester (S3)

Overview

- Architecture: von Neumann
- Textbook: -
- Core Ideas:
  1. IO Devices
  2. File Systems

Computer IO Devices

von Neumann Architecture

- CPU: Central Processing Unit (Very fast)
  Control Unit, ALU, Instruction Register/Queue/Stack, etc.
- Main Memory (Relatively fast)
- IO Devices (Slow)
  - Storage
  - Display, Keyboard, etc.

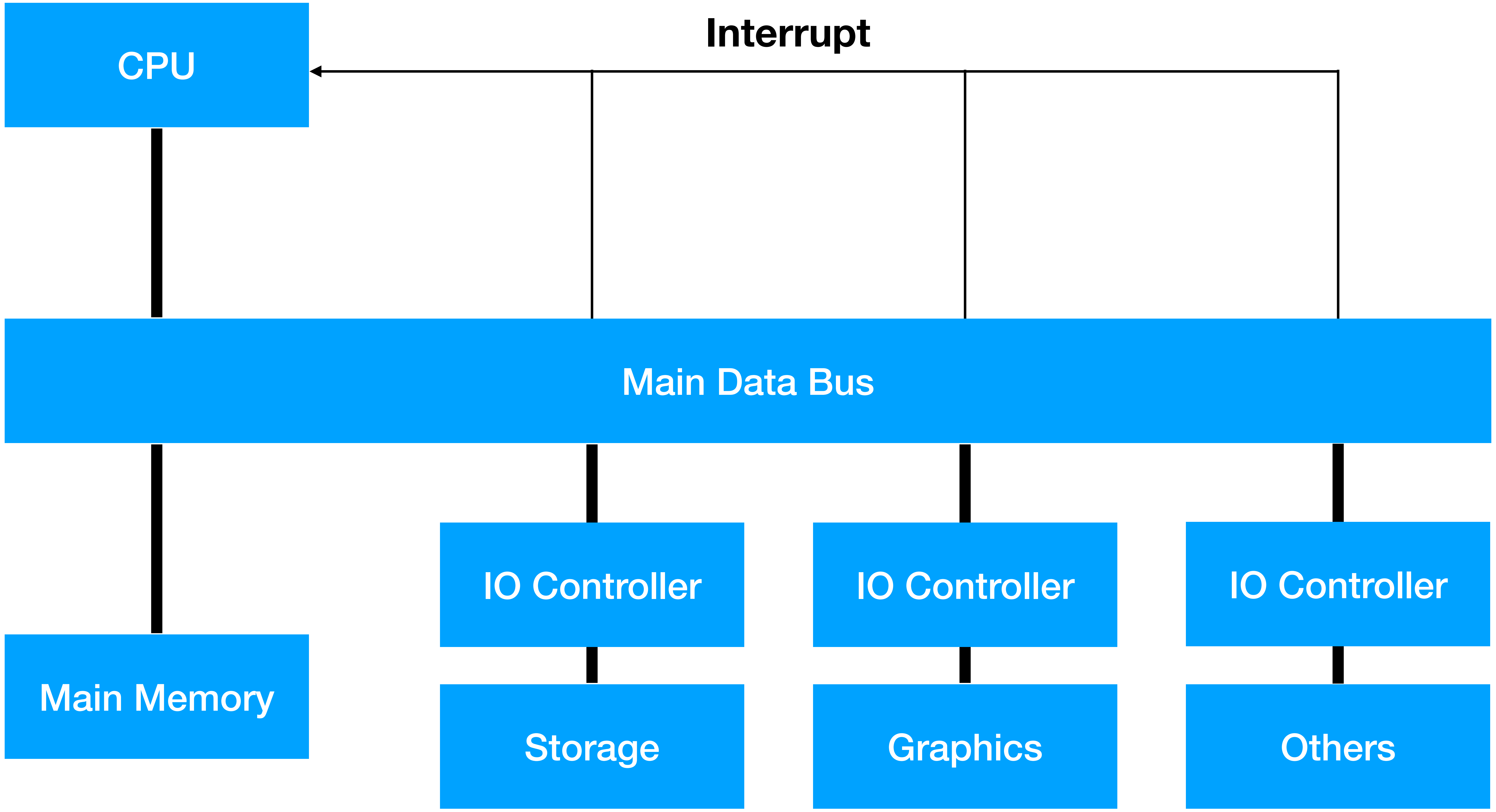
Review

IO Devices

- Controlled by controllers on your motherboard
  - North bridge
    - High Speed Communications
    - Memory, GPU, etc.
  - South bridge
    - Slower I/O Operations
    - HD, SSD, Serial Buses, etc.
  - Information traverses between the CPU and these various controllers through bidirectional buses

Concept

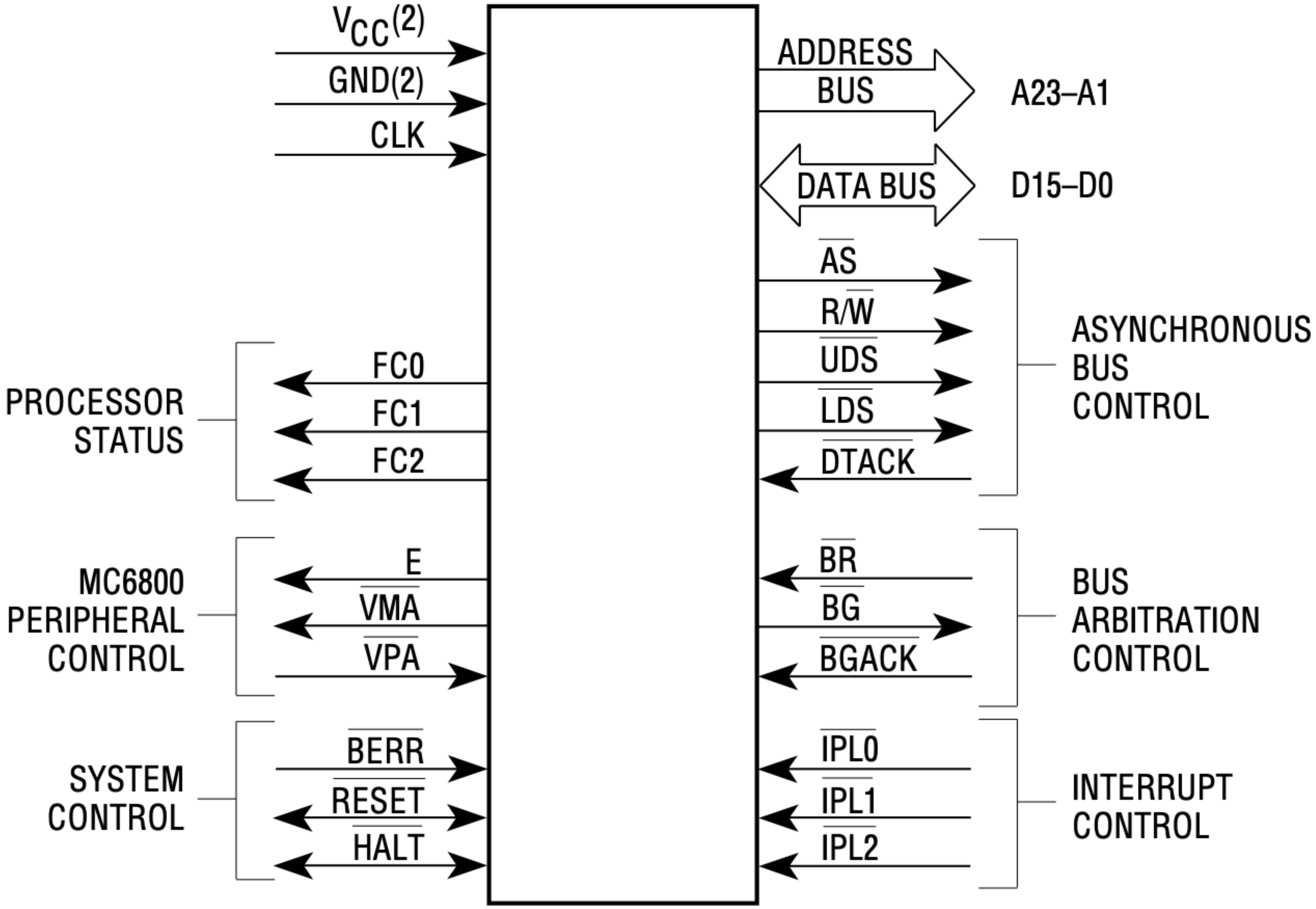
IO Devices



Concept

Example: 68k CPU

- 68000 CPU Pins



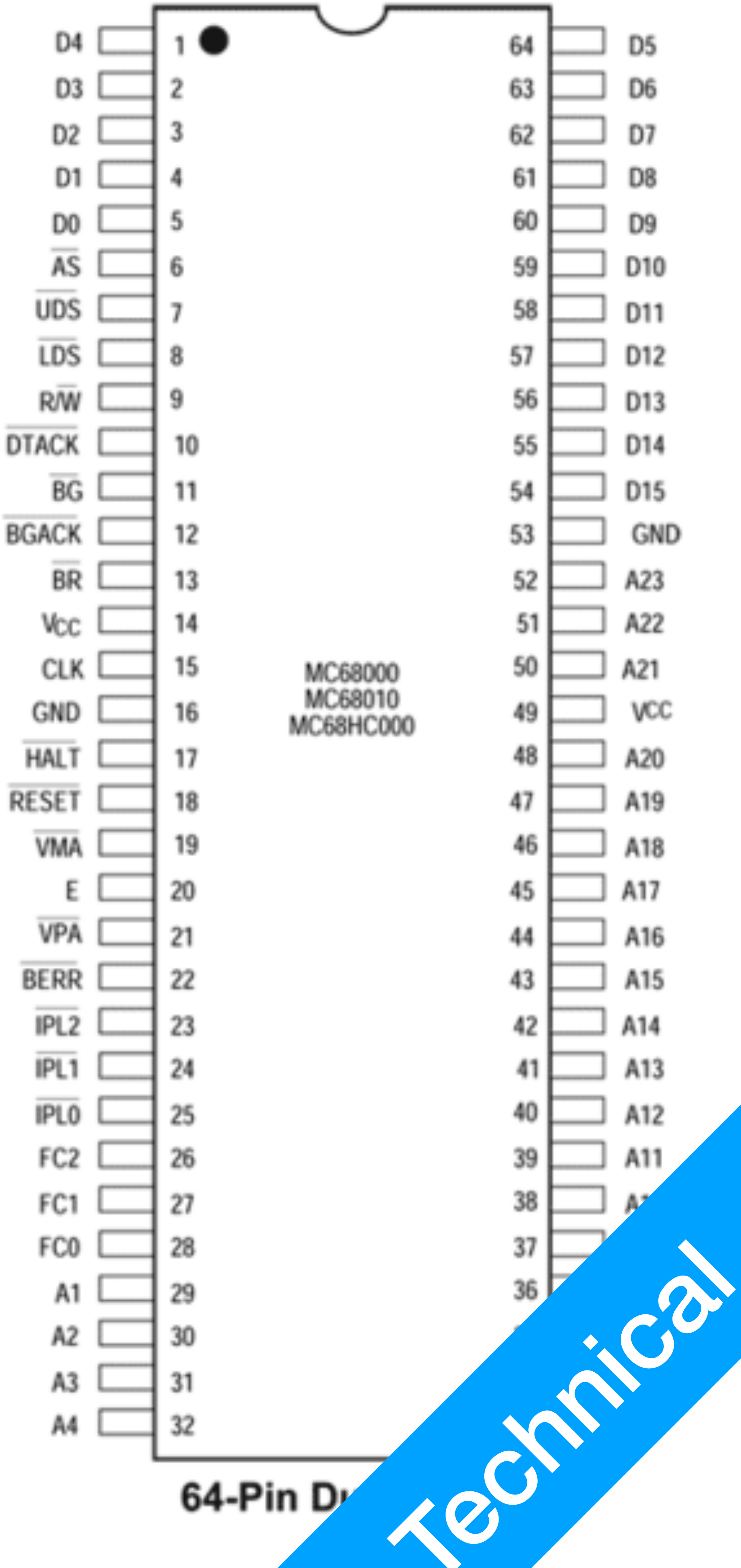
1. https://wiki.console5.com/tw/images/5/5f/M68000.pdf

Technical

P1
IO Devices

Example: 68k CPU

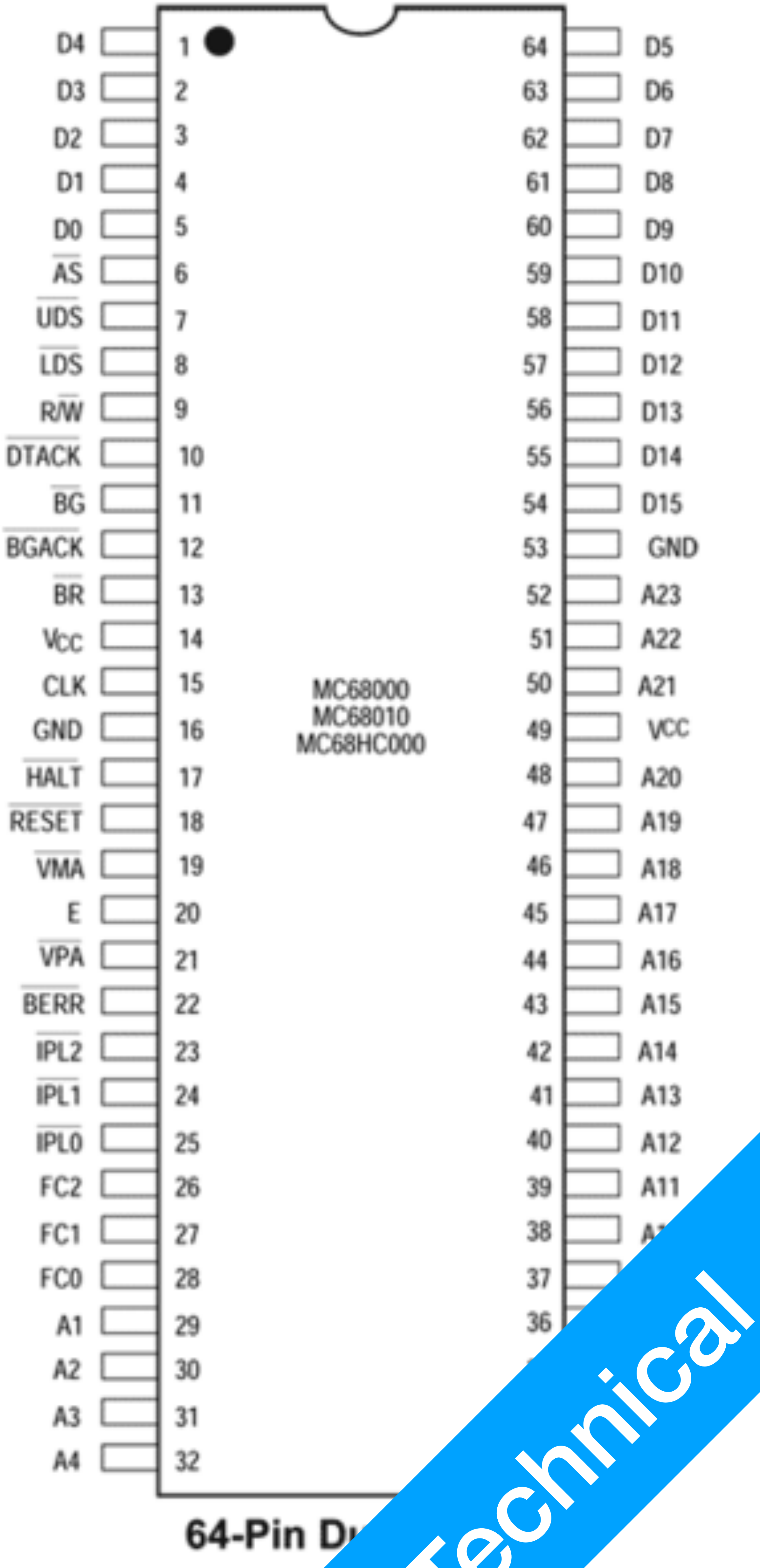
- 68000 CPU
  - 64 total pins
  - 16pins for data bus, bidirectional
  - 24pins for memory address
  - Various status and control input, like interrupt



1. https://wiki.console5.com/tw/images/5/5f/M68000.pdf

Example: 68k CPU

- 68000, Interrupts, Input to the CPU
  - Generated by various controllers
    Floppy Controller + SCSI Controller + Mouse Controller + Keyboard Controller
  - Interrupt: devices want to tell the CPU something
    - CPU listens, when new things happen, interrupts are generated to inform the CPU
      - e.g. new disk inserted, mouse movement, keyboard press, etc.
    - CPU will drop whatever its doing to handle interrupts. Once handled, CPU will enter into **Interrupt Acknowledgment Cycle**, confirm that an intercept has been handled. (using address pins)

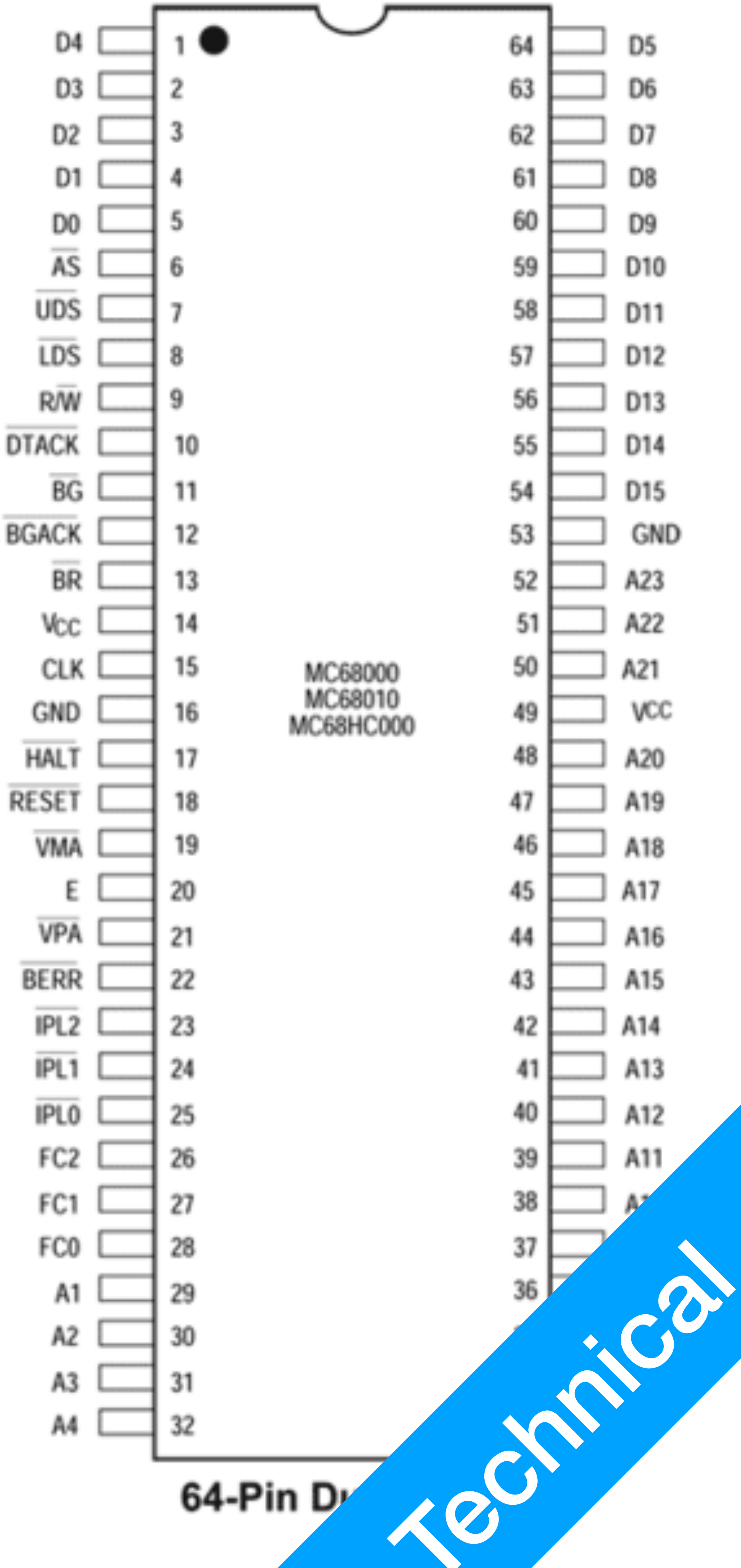


Technical

1. https://wiki.console5.com/tw/images/5/5f/M68000.pdf

Example: 68k CPU

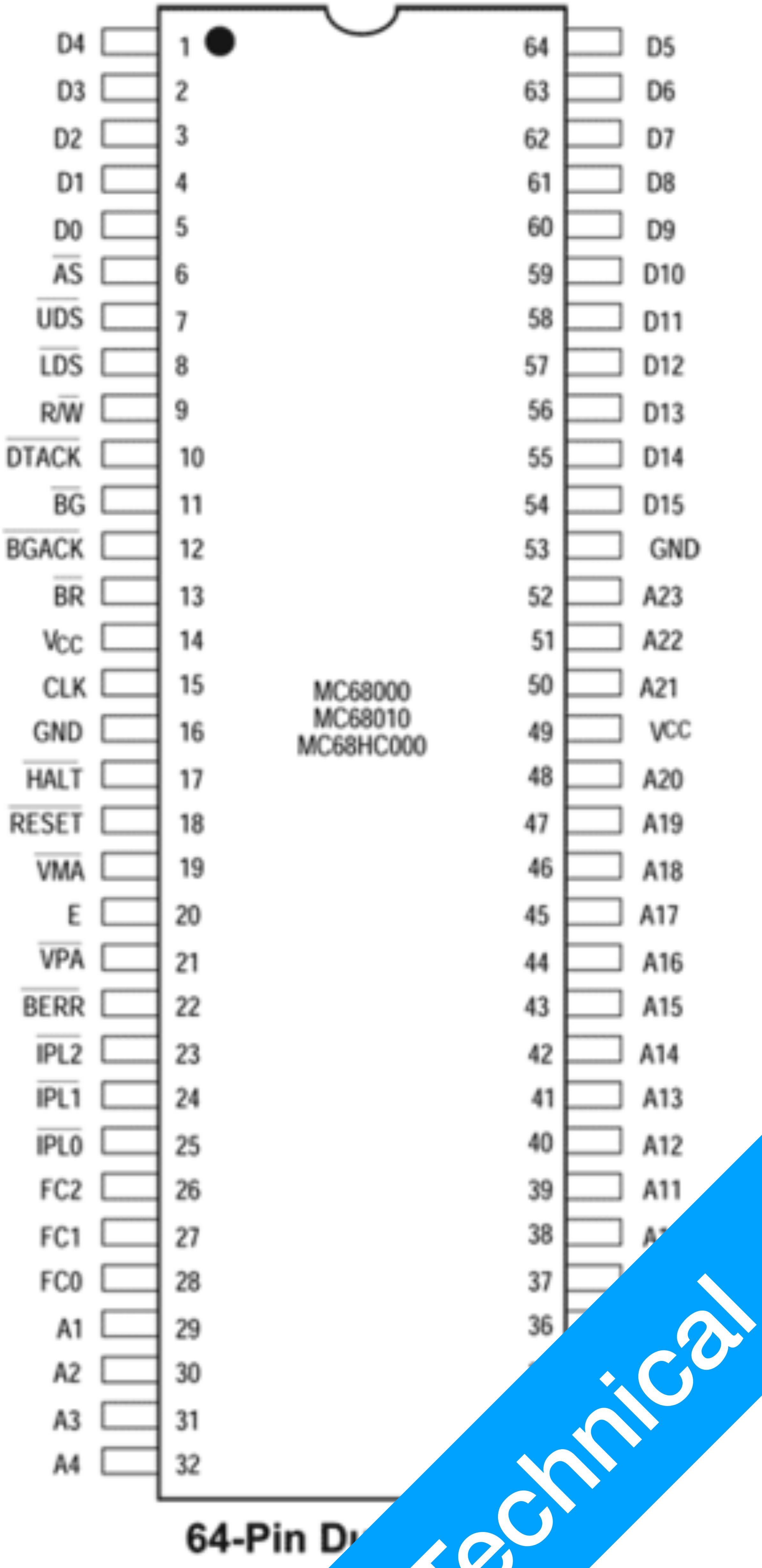
- 68000, the CPU wants to talk to other controllers
  - CPU requests for data bus write permission
  - CPU transmits data to be processed (16bits)
  - When controllers received and processed the data, it will generate a response to the CPU for confirmation
  - CPU moves on to transmit other part of the data, or just go do something else



1. https://wiki.console5.com/tw/images/5/5f/M68000.pdf

P1
IO Devices

Example: 68k CPU



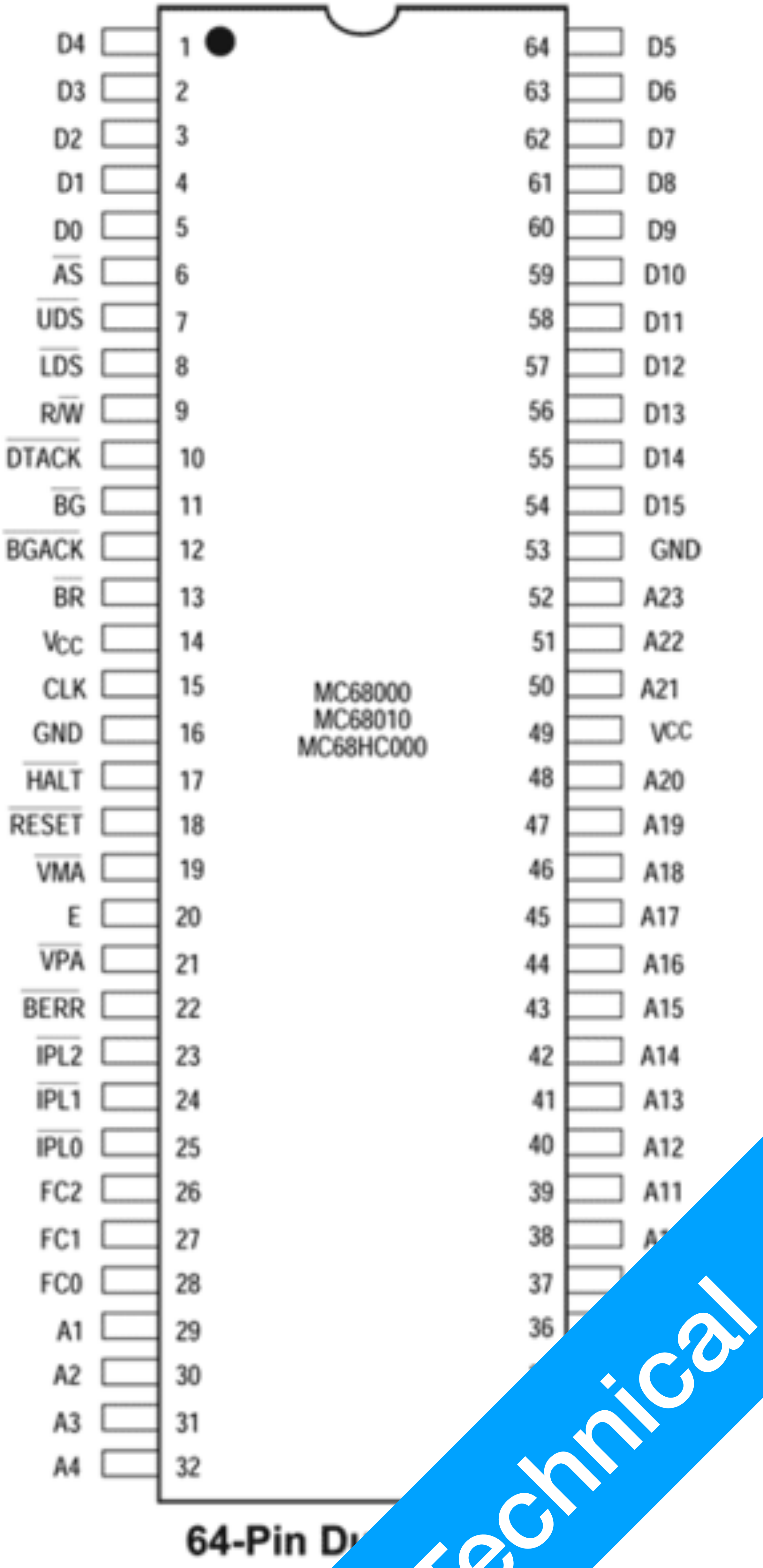
- 68000, the CPU wants other controllers to provide it with data
    - CPU transmits request, then awaits for data bus to provide data
    - Data comes in, CPU gets signal confirming the wait is over, then gets busy processing the data

Technical

1. https://wiki.console5.com/tw/images/5/5f/M68000.pdf

Example: 68k CPU

- Data bus usage
  - R/W (O): Read/Write
    CPU controls whether it reads from or writes to the data bus
  - DTACK (I): Data Acknowledgment
    indicates the completion of data transfer
  - Arbitration Control
    - BUS Request (I); BUS Grant (O); BUS Grant Acknowledge (I)
    - Other devices negotiate with the CPU when they want to transfer data



Technical

1. https://wiki.console5.com/tw/images/5/5f/M68000.pdf

P1
IO Devices

Example: 68k CPU (in MacIntosh Plus)

- The computer has 1MB main memory, shared by the CPU and video controller
- What you see on the 512 x 342 pixel display, each pixel can be Black or White, 1bit of storage required
- This is a total of 22KB, from memory address `#0FA700`

1. https://www.osdata.com/system/physical/memmap.htm#MacPlus
2. Modern computers don't work like this

Technical

Example: 68k CPU (in MacIntosh Plus)

- Memory Address reserved for controllers
  - These are NOT valid main memory addresses, but for when the CPU wants direct access to other controllers/peripherals
  - `#400000 - #41FFFF`: ROM, proprietary subroutines
  - `#580000 - #5FFFFF`: SCSI controller
  - `#900000 - #9FFFFF`: SCSI Read
  - `#B00000 - #BFFFFF`: SCSI Read
- When the CPU requests access to these addresses, it is in fact interacting directly with controllers. Modern computers don't do this normally.

1. https://www.osdata.com/system/physical/memmap.htm#MacPlus
2. Modern computers don't work like this

Technical

Computer File Systems

Storage Device History

- Before the invention of PC, computers commonly use paper tapes like punch cards

- Remember some of our instructions? They are done on punch cards/tapes and fed to the machine directly, as computer processes each instruction, the feed machine feeds in the next bit

- Later on, PCs used stuff like cassette tapes to store binary signals
  Yes, the same cassette tapes as your mum's walkman

Concept

Storage Device History

- Entire cassette's content needed to be loaded into memory before anything can be used
- Apple I, Commodore, etc.
  - Put the cassette in, type a load command, play the tape to the computer, then entire programmes are loaded into the main memory
  - A hardware controller is used to transfer information directly to memory, this is called: Direct Memory Access
  - CPU has limited capability to control these devices, it requires special instruction designed specifically for the task

Concept

Common CPU Capabilities

- Entire cassette's content needed to be loaded into memory before anything can be used
- Apple I, Commodore, etc.
  - Put the cassette in, type a load command, play the tape to the computer, then entire programmes are loaded into the main memory
  - A hardware controller is used to transfer information directly to memory, this is called: Direct Memory Access
  - CPU has limited capability to control these devices, it requires special instruction designed specifically for the task

Concept

Storage Device History

- Floppy Disk
  - A major breakthrough
  - 5.25 Floppy
    - Initially, 80KB per disk
    - The use of file systems: multiple files (including programmes) are stored within a single disk

Technical

Major File Systems

- 1977 FAT (File Allocation Table)
    - Modern version of which: exFAT, still used today
- Modern File Systems
    - M$: NTFS; Linux: ext4; apple: APFS

Concept

Major File Systems

- Features
  - Multiple Files
  - Folders
  - Journaling*
  - Encryption*

Concept

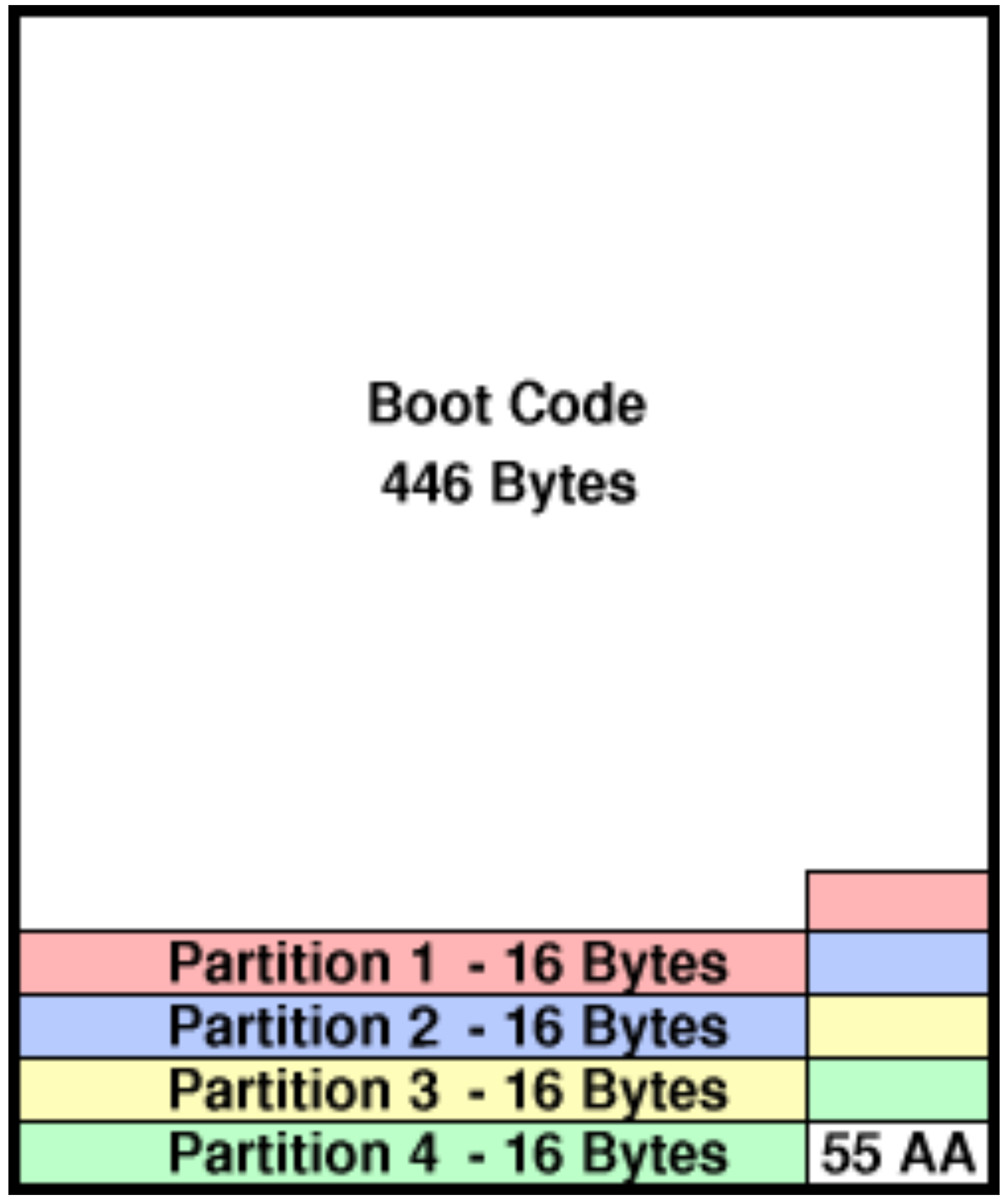
Physical Hard Drive/SSD/SD Cards

- Partition Maps
  - GUID Partition Map (Current)
    - Used by Windows/Linux/OSX
  - Master Boot Record (Old)
  - Partition Maps store how physical storage devices are divided into software drives
    - Windows: C drive, D drive, etc.

Technical

Physical Hard Drive/SSD/SD Cards

- Master Boot Record
  - 4 Logical Partitions per device
  - First 512 bytes (sector 0) of a device is used for the MBR partition table



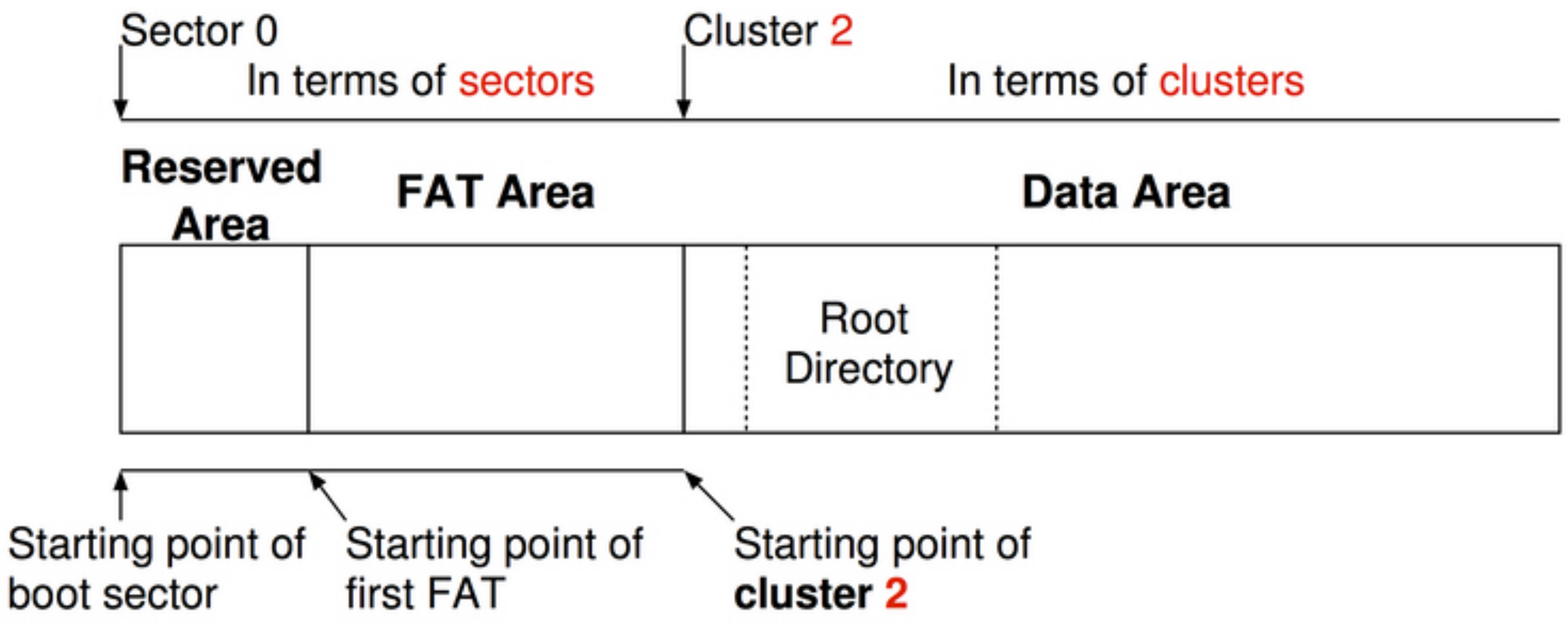
1. https://www.pjrc.com/tech/8051/ide/fat32.html

FAT32

- An FAT32 partition (also called volumes) is divided into **equally sized clusters** (small blocks of contiguous spaces)
- **Each file** may occupy **1 or more clusters** depending on its size. These clusters are not necessarily adjacent to one another, which may lead to fragmentation
  - This is implemented using a **Linked List** data structure.
  - Bytes per sector: 512 Bytes
    this is more a reference to hard drives, but still exist when discussing SSDs etc.
  - Sectors per cluster: 1-128
    depends on device's actual size, partition's size, etc.
- Every partition volume has 2 **File Allocation Tables**, storing the general folder/file structure of the entire partition, as well as where each file is located.

Technical

FAT32



- Directories are represented as a special type of files
- The FAT area in the beginning part of a partition:
  - Two **File Allocation Tables**, for redundancy. This provides the maps of data region, indicating which clusters are used by files and directories
- Boot Sector: used for starting an Operating System, stores the instructions for OS startup

1. https://eric-lo.gitbook.io/lab9-filesystem/overview-of-fat32

Technical

FAT32

- File/Directory Entry in the FAT table
  - `0x00-0x07`: Short file name (8bytes)
    Long filenames require extensions, boring subject we won't cover it in class
  - `0x08-0x0A`: Short file extension (3bytes)
  - `0x0B`: File Attribute (1byte)
    Read-Only, Hidden, System Flag, etc.
  - `0x0C-0x15`: Misc
  - `0x16-0x19`: Last modified time and date (4bytes)
  - `0x1A-0x1B`: First file cluster location (2bytes)
  - `0x1C-0x1F`: File size in bytes (4bytes)

1. https://eric-lo.gitbook.io/lab9-filesystem/overview-of-fat32

Technical

Example: a hard drive with a single file

- File name: `boo.txt`
- File content: `Hello World!`
    - HEX: `48 65 6c 6c 6f 0a`
    - `0a` is ASCII for End-of-File

- Your hard drive
    - MBR sector; followed by
    - Boot Sector of partition 1; followed by
    - FAT area with 2 FATs, inside only one entry, the rest empty; followed by
    - Actual file: `48 65 6c 6c 6f 0a`

1. https://eric-lo.gitbook.io/lab9-filesystem/overview-of-fat32

Technical