Operating Systems:
Chap 5: I/O devices and management
CS400

Week 12: 31st March

Spring 2020

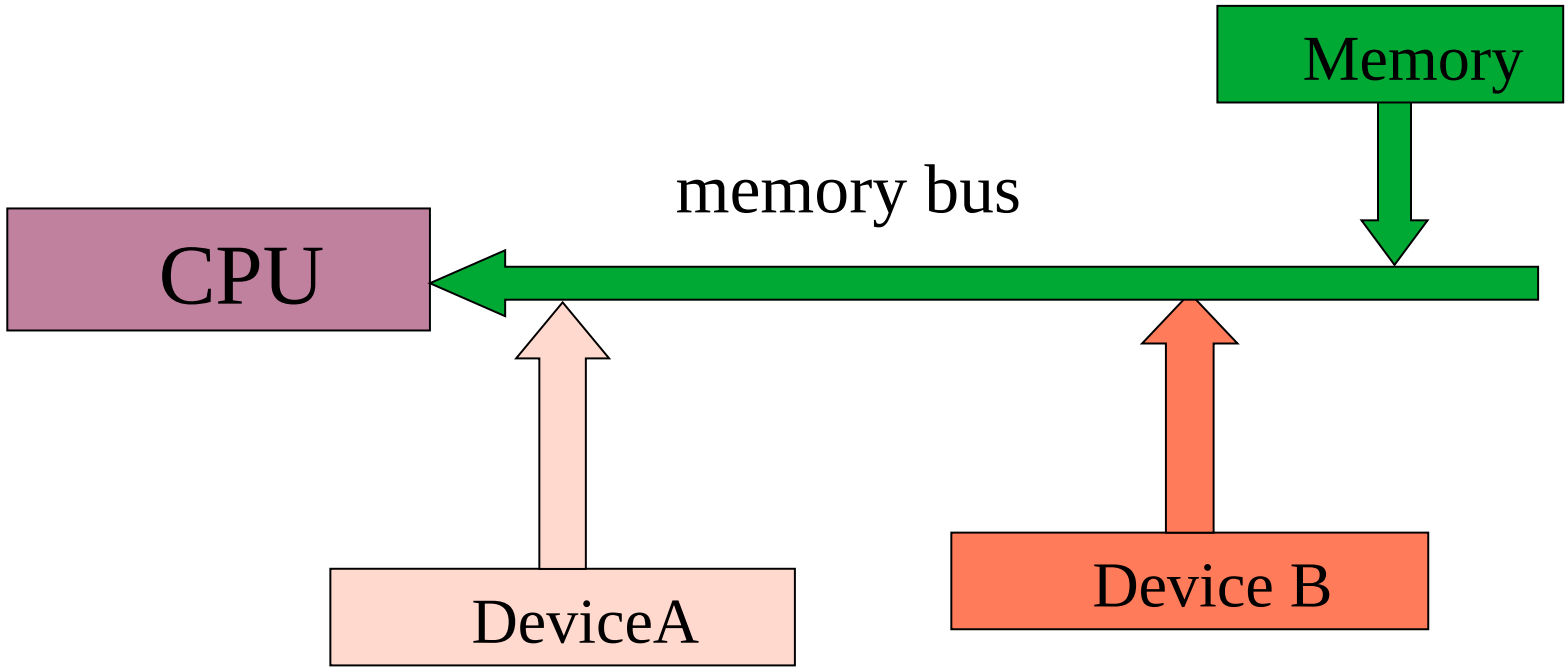
Oliver BONHAM-CARTER

The I/O Subsystem

- The largest, most complex subsystem in OS
- Most lines of code
- Highest rate of code changes
- Where OS engineers most likely to work
- Difficult to test thoroughly

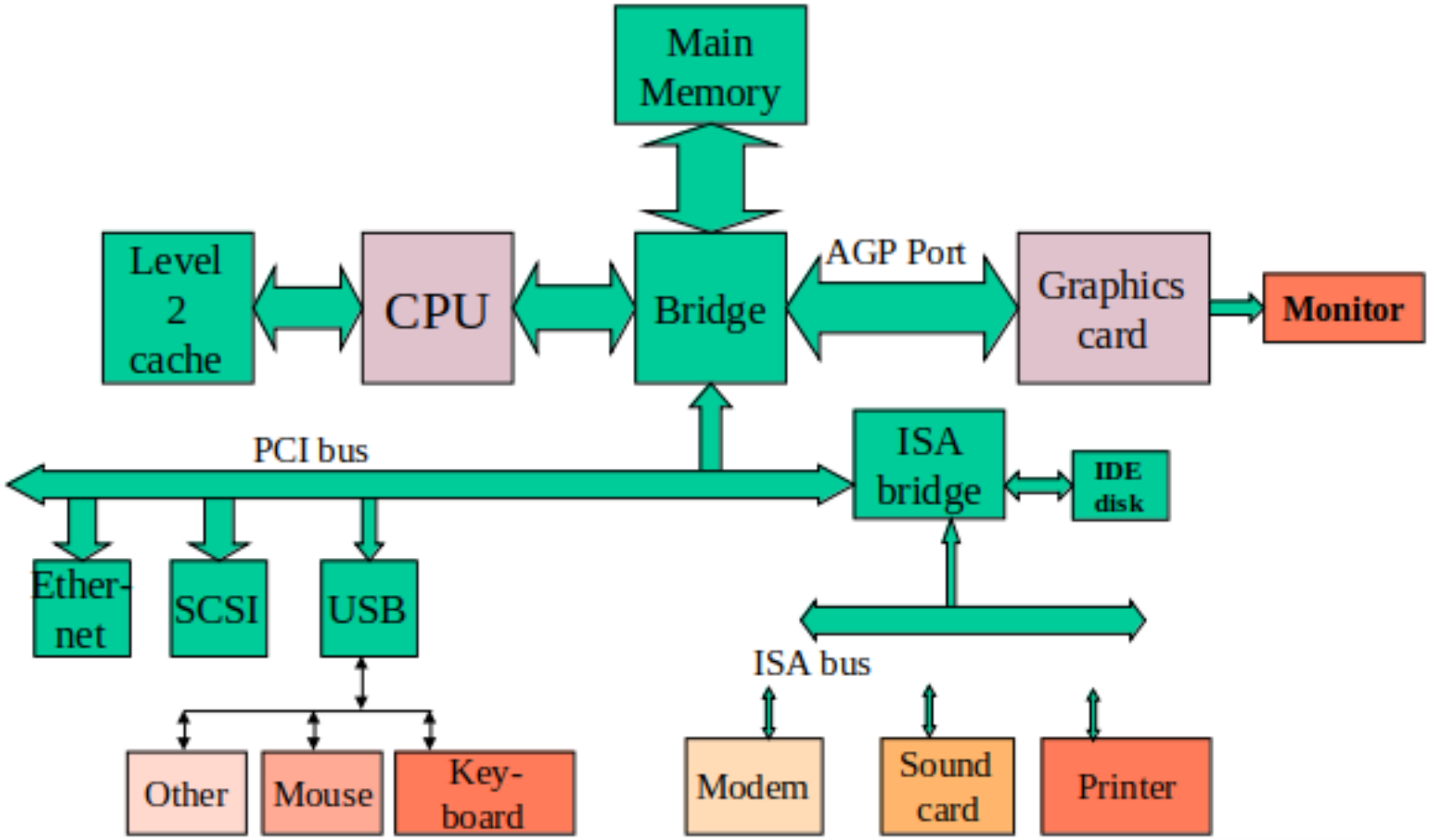
- Make-or-break issue for any system
  - Big impact on performance and perception
  - Bigger impact on acceptability in market

How Does I/O Move?



Simple Hardware Organization

How Does I/O Move?



Complex Hardware Organization

So, What Uses I/O?

- Character (and sub-character) devices
  - Mouse, character terminal, joy stick, some keyboards
- Block transfer
  - Disk, tape, CD, DVD
  - Network
- Clocks
  - Internal, external
- Graphics
  - GUI, games
- Multimedia
  - Audio, video
- Other
  - Sensors, controllers

Anything that plugs into a computer and has to work with the CPU will need some kind of I/O communication.

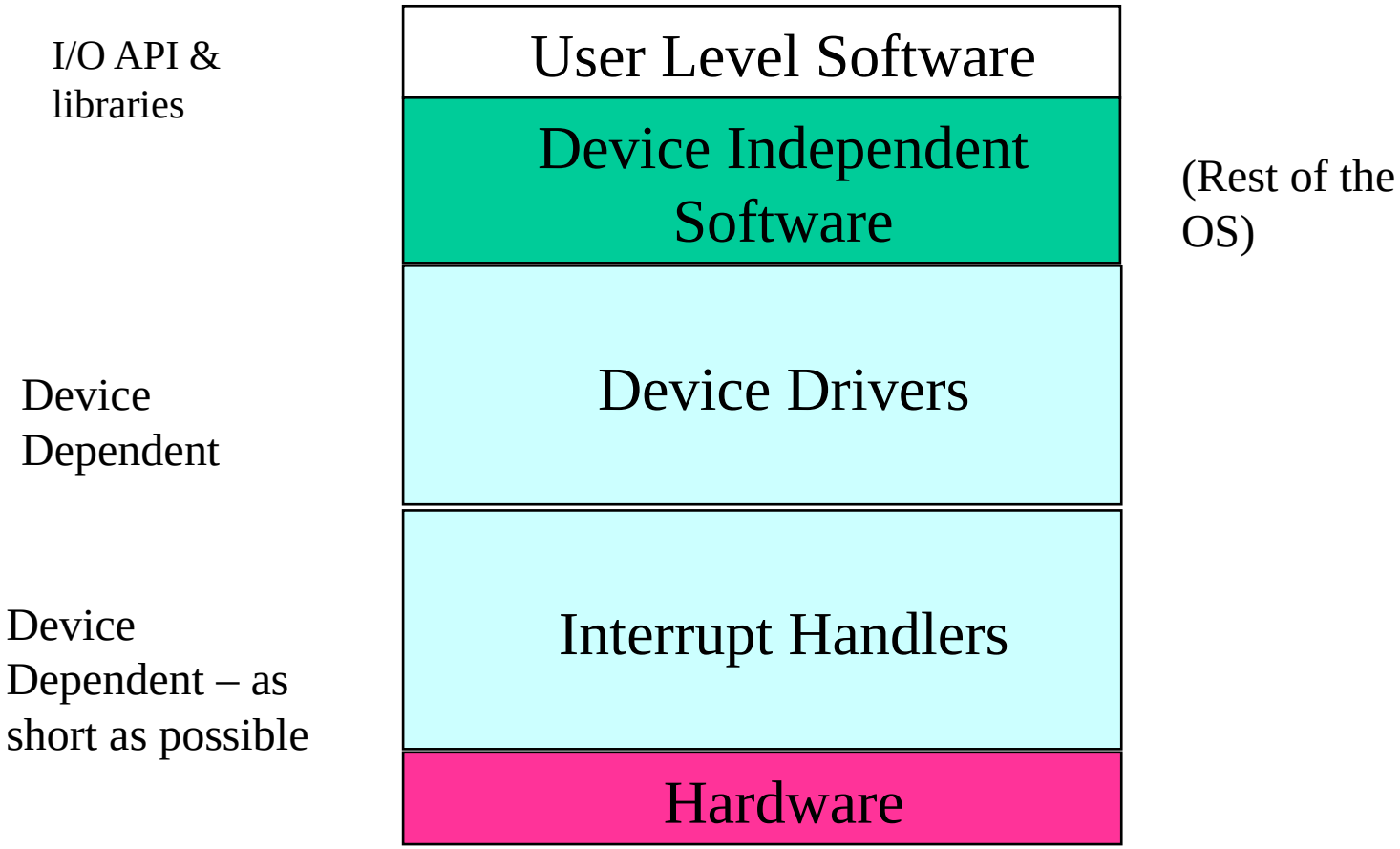
What Should I/O *Software* Do?

- **Efficiency** – Do not allow I/O operations to become system bottleneck
  - Especially slow devices
- **Device independence** – isolate OS and application programs from device specific details and peculiarities
- **Uniform naming** – support a way of naming devices that is scalable and consistent
- **Error handling** – isolate the impact of device errors, retry where possible, provide uniform error codes
  - Errors are abundant in I/O

What Should I/O *Software* Do?

- **Buffering** – provide uniform methods for storing and copying data between physical memory and the devices
- **Uniform data transfer modes** – synchronous and asynchronous, read, write, ..
- **Controlled device access** – sharing and transfer modes
- **Uniform driver support** – specify interfaces and protocols that drivers must adhere to

The I/O *Software* Stack



ALLEGHENY COLLEGE

To Control or Interact with I/O?

- A function of host CPU architecture
  - One of the most important parts of running a machine
- Special I/O instructions
  - Opcode to stop, start, query, etc.
  - Separate I/O address space
  - Kernel mode only
- Memory-mapped I/O control registers
  - Each register has a physical memory address
  - Writing to data register is output
  - Reading from data register is input
  - Writing to control register causes action
  - Can be mapped to user-level virtual memory

Ways to Access (1)

- Memory-Mapped I/O (MM)
  - If there are special instructions to read or write the device control registers (i.e., to tell it to switch states), MM I/O is able to access addresses in memory as variables.
  - Commands may be coded directly in C.
- IO Ports
  - Special read instructions must be sent with assembly code (system calls) to connect to device registers.
  - Overhead and slow-down to seek command.

Ways to Access (2)

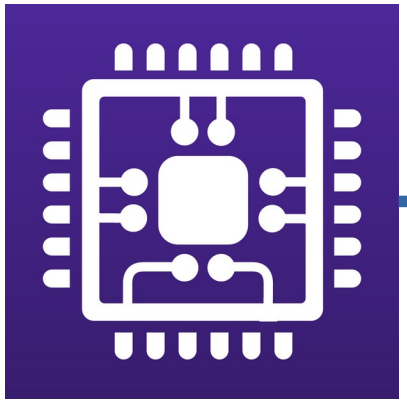
- Memory-Mapped I/O
  - No special protection to prevent user processes from performing I/O: OS hides the memory space where these “variables” are held.
  - Selective access: OS can hide memory space for some devices but not others.
  - A driver cannot interfere with *non-self* devices
- I/O Ports
  - Harder to control which devices the user can access: all registers are “available”.

Ways to Access (3)

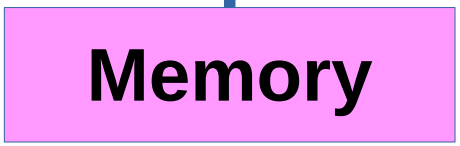
- Memory-Mapped I/O
  - Every instruction that can reference memory can also reference control registers
  - Control registers may be saved in memory and used to signal availability. Returning values stored in memory are these signals.
  - Different drivers do not conflict with each other by accident: coded to find specific entries at memory addresses.
- I/O Ports
  - Hard to prevent drivers from accessing the wrong devices. Resulting errors are hard to interpret.

How Do You Move Data?

- Whether or not a CPU has a memory-mapped I/O, it needs to address device controllers to exchange data with them

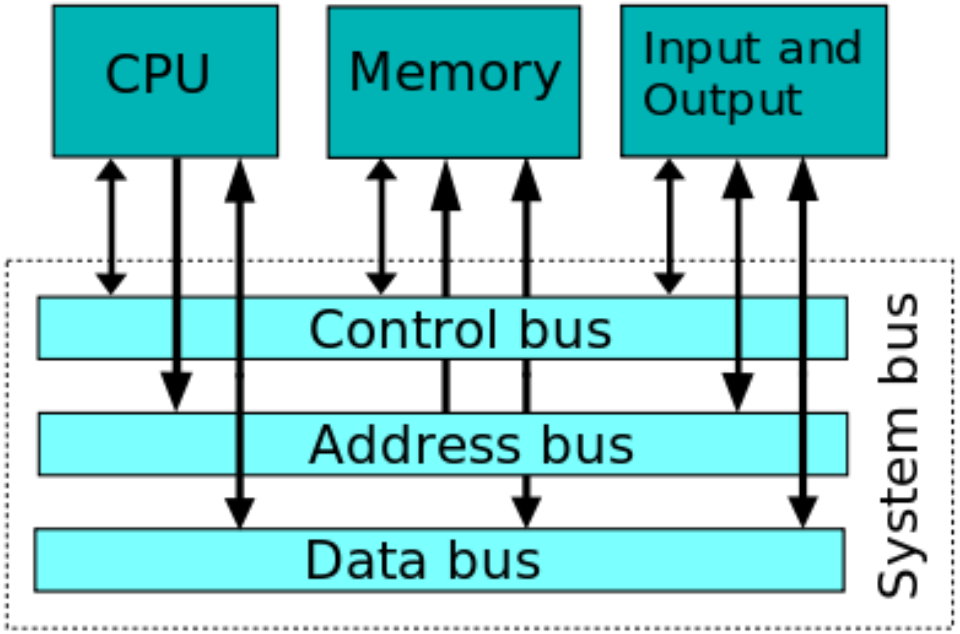


The CPU connects to all devices, memory and etc via the bus



Data Moves Via the Bus

- In computer architecture, a **bus** is a communication system that transfers data between components inside a computer, or between computers. Data moved between hardware components (wire, optical fiber, etc.) and software, including communication protocols.



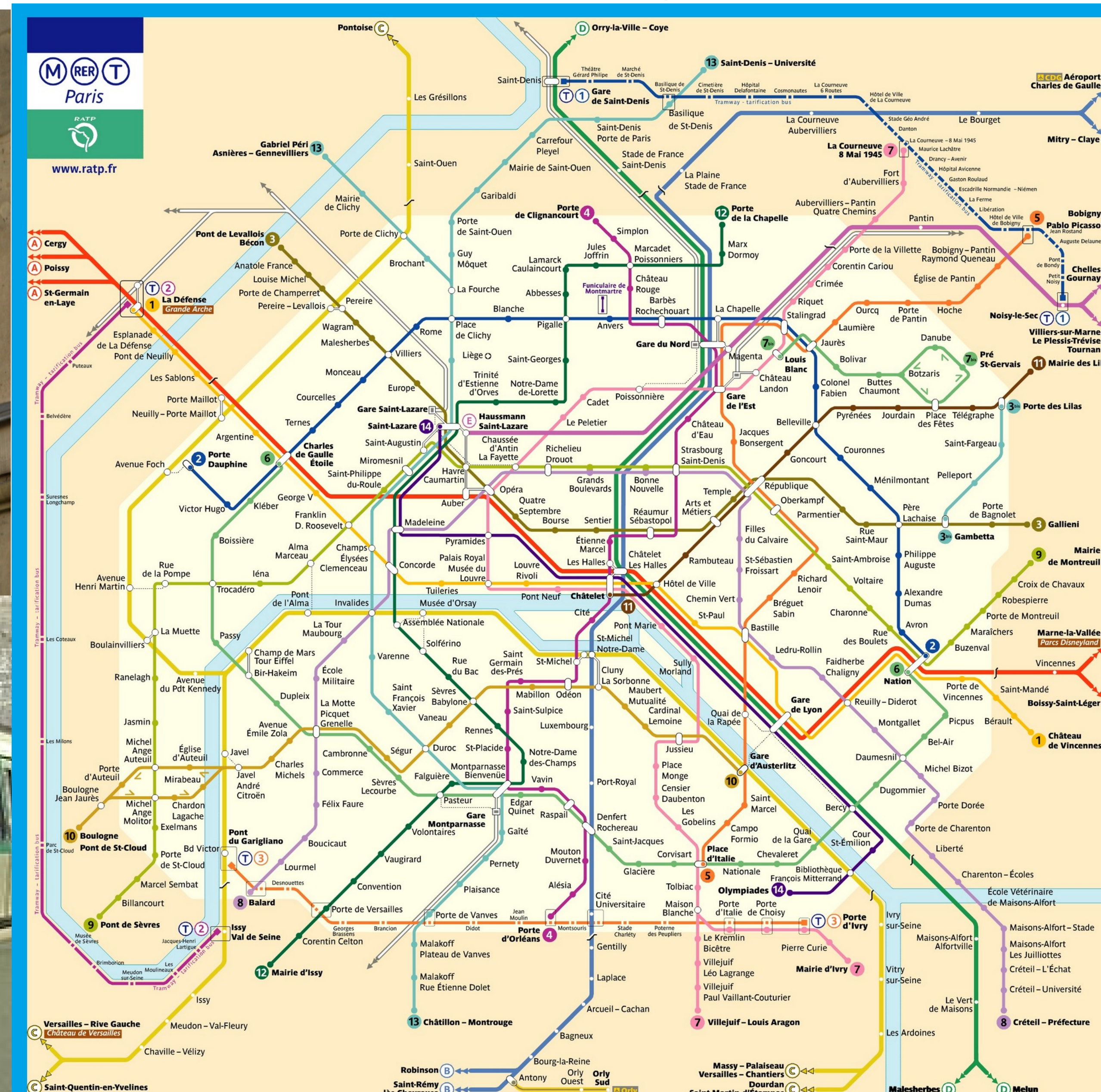
Channels of the Bus

- **Data bus**: connects all the internal components of a computer, such as CPU and memory, to the motherboard.
- **Address bus** (a series of lines connecting two or more devices) is used to specify a physical address.
  - When a processor or DMA-enabled device needs to read or write to a memory location, it specifies that memory location on the address bus (the value to be read or written is sent on the data bus)

Channels of the Bus

- **Control bus:** is used by CPUs for communicating with other devices within the computer.
  - The address bus carries the information about the device with which the CPU is communicating and the **data bus** carries the actual data being processed,
  - The **control bus** carries commands from the CPU and returns status signals from the devices.
  - For example, if the data is being read or written to the device the appropriate line (read or write) will be active (logic one).

The Real-World Bus System

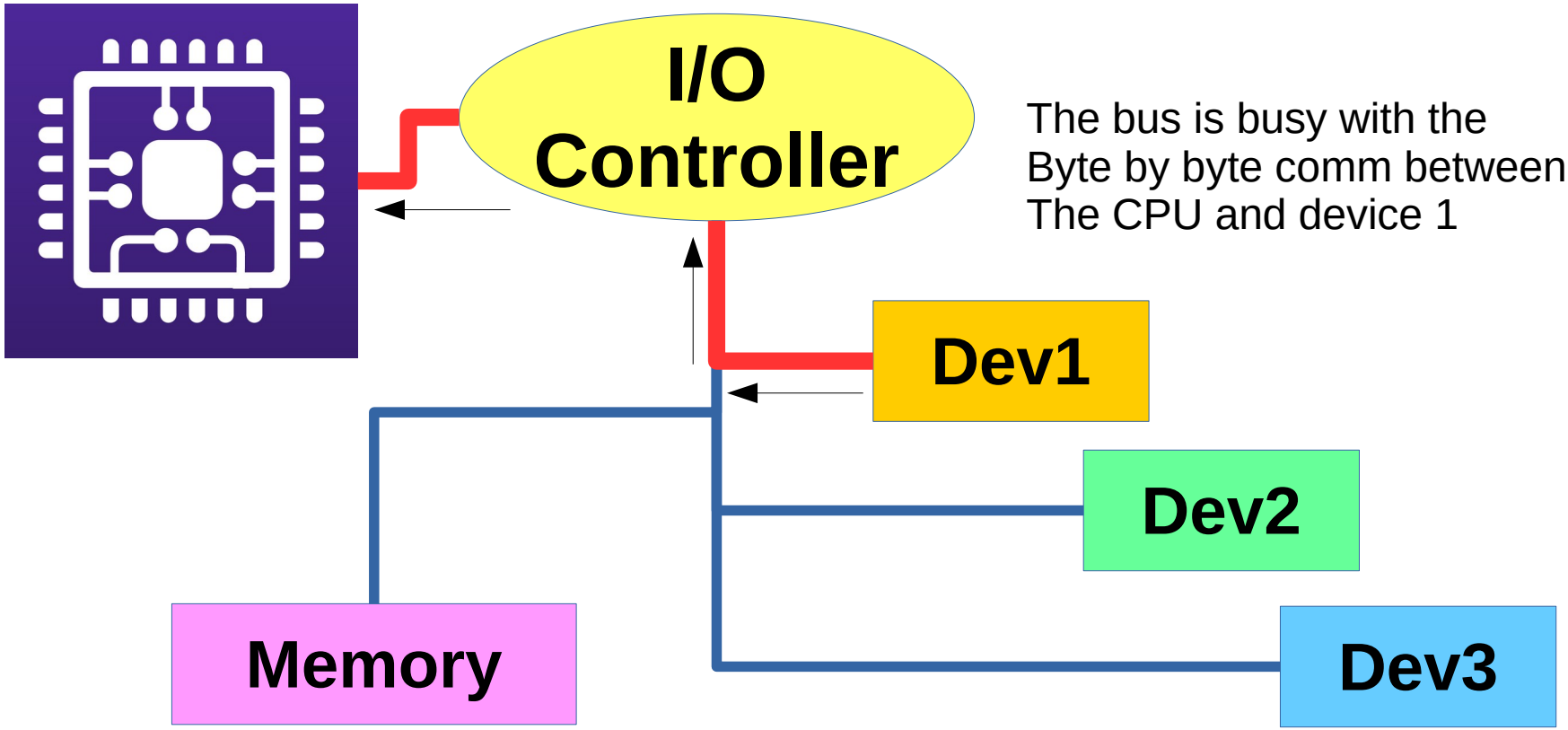


Bus Systems Get Clogged

- Real world bus systems can be over-worked and cannot run efficiently when:
  - Too many passengers to move
  - Too few lines of transport
- Slow-down in computer comm lines (bus systems)
  - Too much data
  - Too few lines of transport
- Slow down in transfer

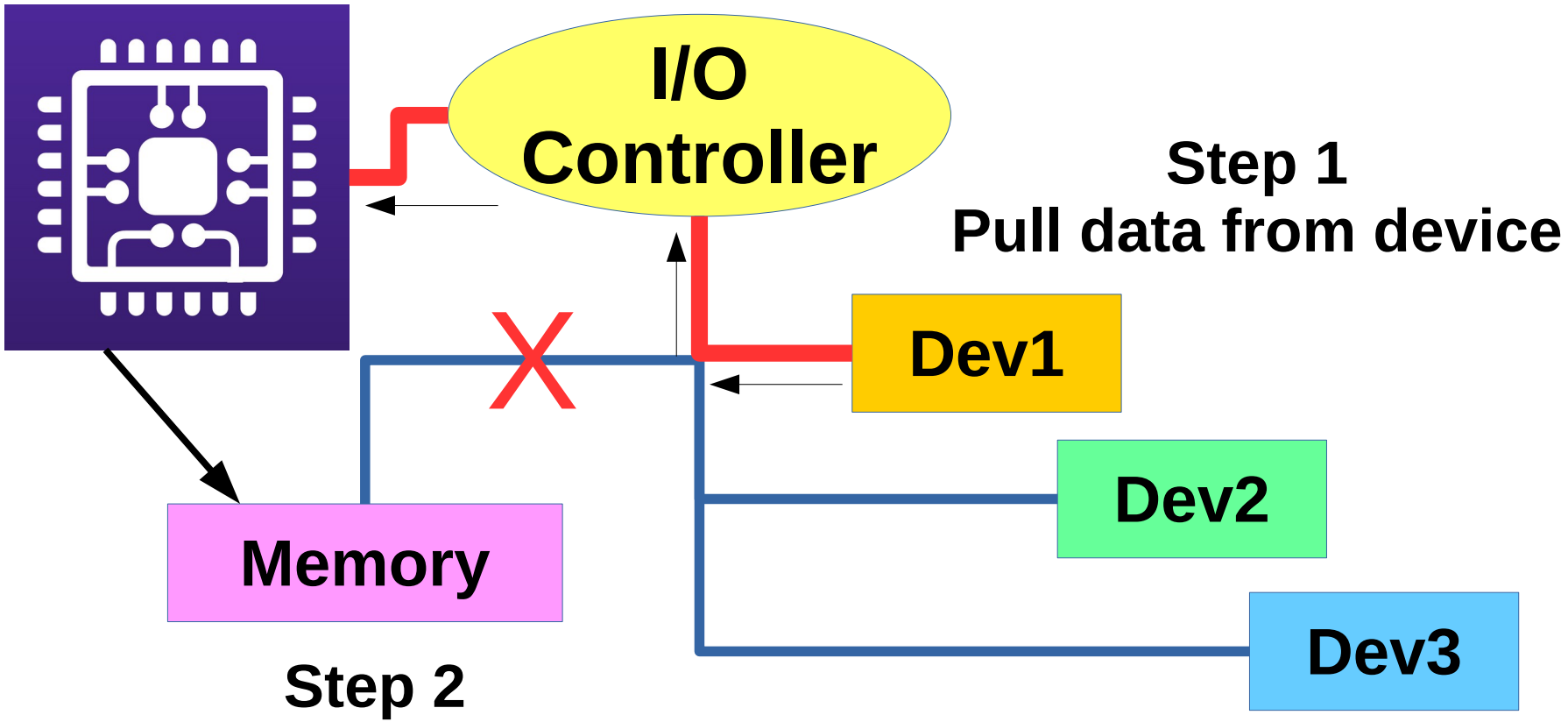
CPU Data Request From Dev1

- The CPU could request one byte at a time from a device, but that would monopolize the bus and be inefficient.



Get Data into Memory?

- A bottleneck in the bus



ALLEGHENY COLLEGE

Basic Dev Data Requests

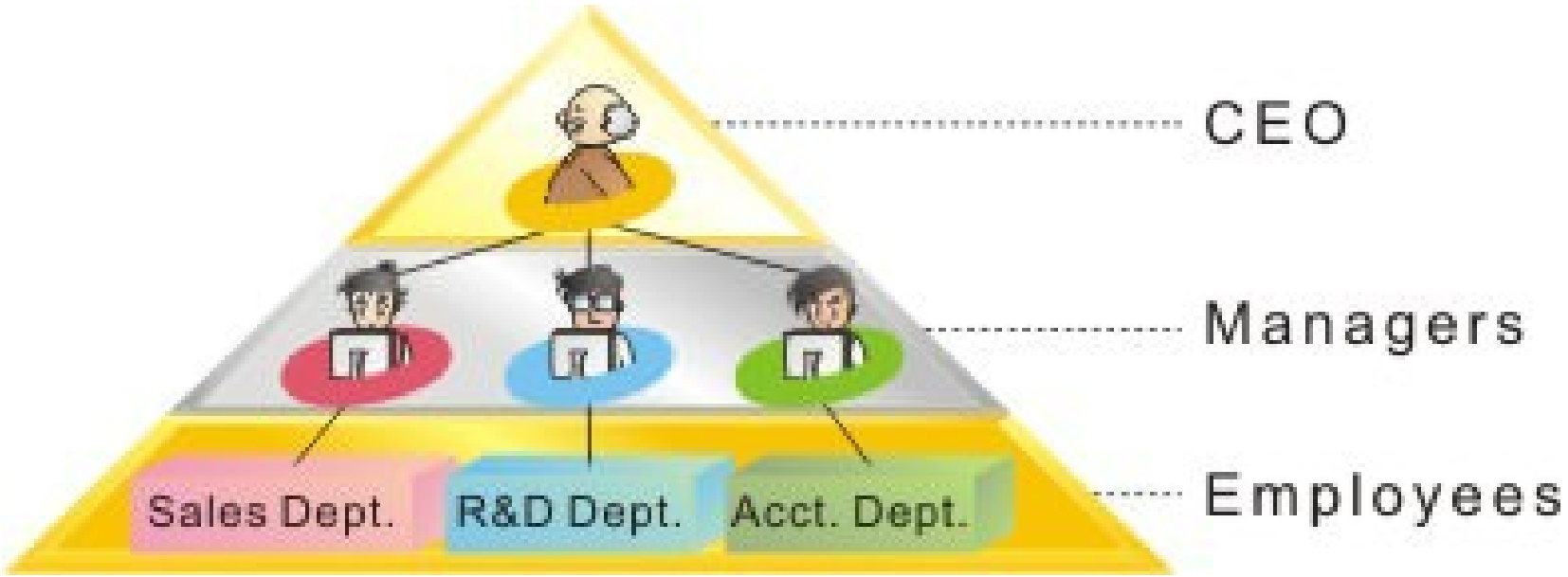
- CPU → Controller: *get me the data*
- Controller reads data blocks from hard drive device (bit by bit)
- Controller's buffer fills
- Controller computes checksum to determine if any read-errors have been made (re-send blocks?)
- Controller causes an interrupt: controller's buffer is sent to CPU, while (data_exists) do, loop.
- CPU copies this data into memory addresses

ALLEGHENY COLLEGE

Basic Dev Data Requests

- The CPU soon becomes monopolized
- A slowdown: pulling data and pushing data around the bus can cause bottlenecks
- How can the system be changed to take away some of the bus' traffic?
- **Seriously, how do you think the bus' data transferring could be optimized?**
- Remember that the CPU is also being used to copy data from devices to memory

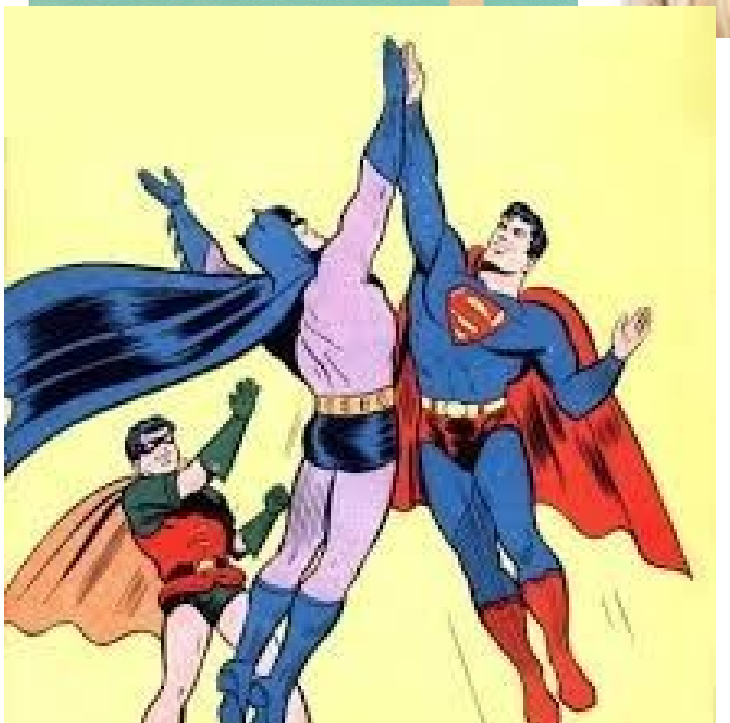
A Management System



- Management of devices by CPU is done by a “local manager” like system. CEO is free to do other work.
- Direct Memory Access: devices are controlled by a manager. CPU is free to do other work.

ALLEGHENY COLLEGE

Use Direct Memory Access!!

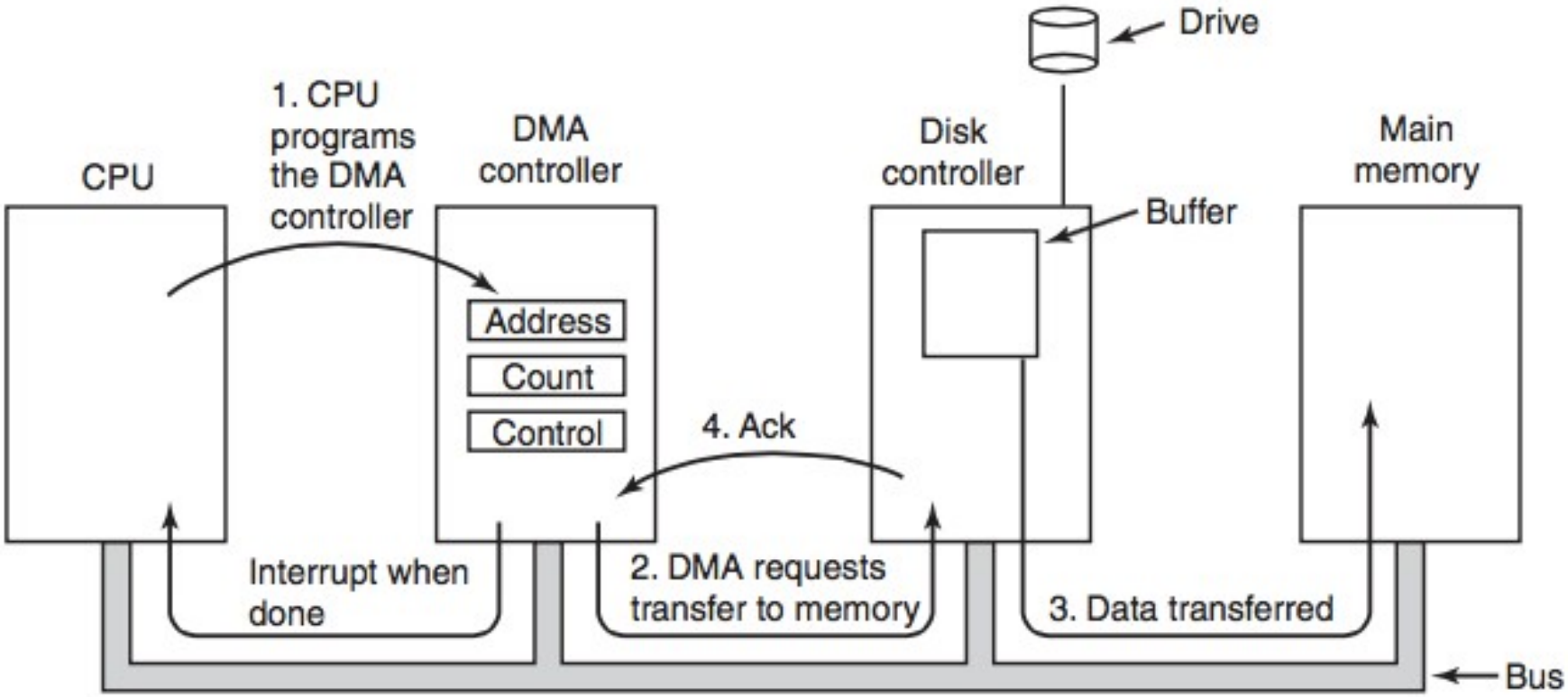


ALLEGHENY COLLEGE

Direct Memory Access (DMA)

- A system to help move information to the memory and not tie-up the system bus
- DMA has independent access to system bus, different from CPU's access to system bus.
- Mechanism: The DMA acts like a postal system
  - Data pulled from devices, stored in DMA controller and then sent directly to memory locations
  - CPU is not busy with data traffic process and is free to use bus for other tasks.
  - DMA system to find *opportune* times to populate memory.

Direct Memory Access Systems



DMA Controllers

- CPU has access to all devices on bus: sends instructions
- Streaming devices have own DMA controller to handle device data
- The CPU requests device controller to pull data from its device
- CPU command to controller contains:
  - Command: read / write
  - Dev address
  - Memory address
  - Length of data to copy
  - Status: active /inactive
- CPU is now free to work on other processes

ALLEGHENY COLLEGE

Four Steps

- **Step 1**: CPU sends request to DMA controller.
  - Registers are set: metadata sent about which device addresses and memory addresses to use
  - Read command given, permission to use controller's buffer, command verify checksums
- **Step 2**: The controller's buffer is now full
  - Controller sends data to memory addresses
- **Step 3**: Loop to keep writing
  - While buffer is not empty, send to memory addresses
- **Step 4**: Wait for the "all completed" signal
  - An acknowledgement signal from device (hard drive) will confirm that data transfer is complete

Slight Delays Beat Canceled Service

- Cycle Stealing and bus sharing: The device controller “sneaks” bandwidth from the bus when the buffer is full. Here, CPU must wait.
- Although there may be a slight CPU delay, generally, the bus traffic with DMA than if the CPU had pull the device data directly and then place it in memory alone.
- More advanced controllers may use the bus more efficiently by breaking up their transfers over more time or by sending data at *quieter* times in CPU.