Principles of Input/Output (IO)

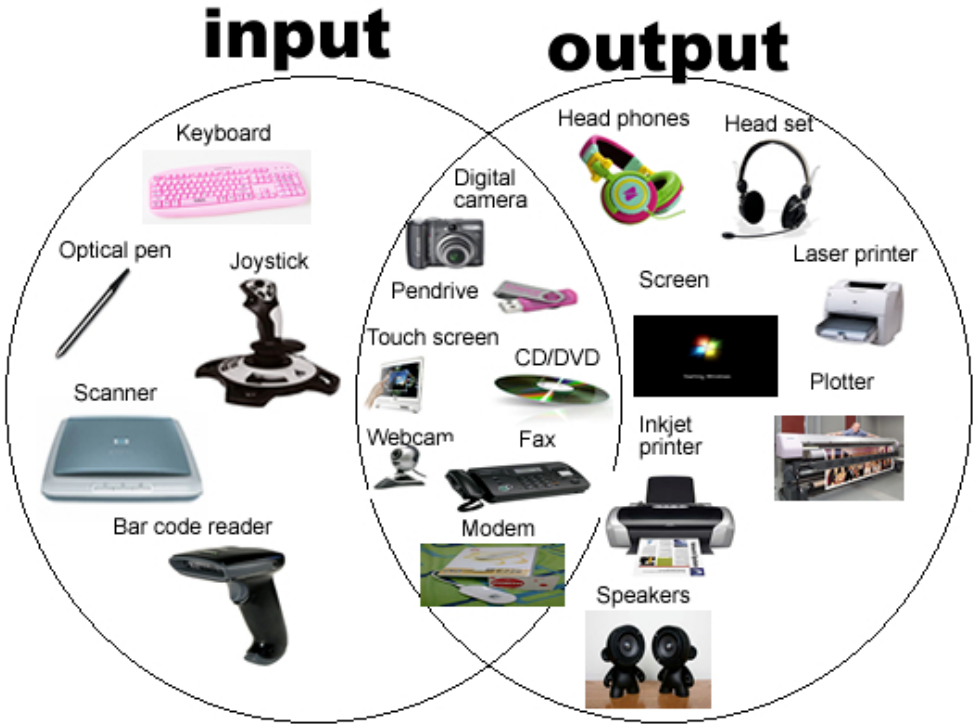
Loïc Guégan

Adapted from I. Raïs @ UiT

UiT The Arctic University of Norway

Spring - 2024

I/O Devices example



Each has:

- Its own speed of execution
- Its own way of working

Large number of device to support that are supposed to work together

Type of I/O devices

Block devices

- Transfers are in units of blocks
- Organize data in fixed-size blocks
- Blocks are addressable independently
  $\Rightarrow$ read or write blocks independently

Examples: hard disks, SSD, USB sticks

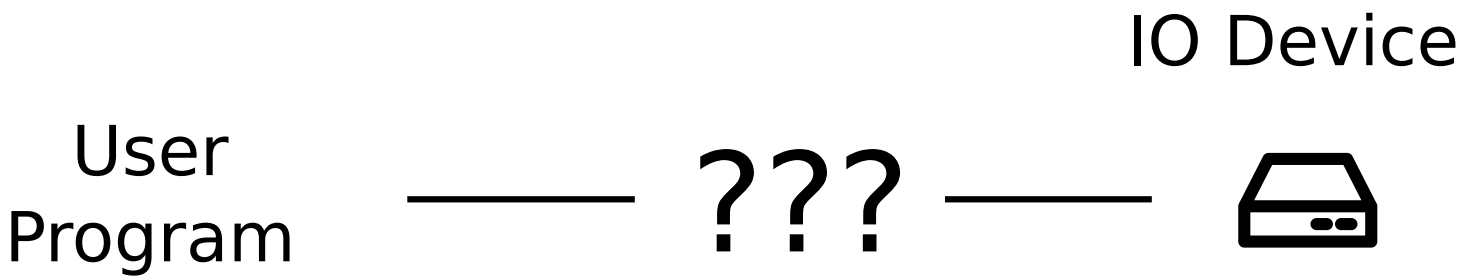
Character devices

- Delivers or accepts a stream of characters,
- No block structure
- Not addressable, no seeks
- Can read from or write to stream

Examples: Printers, network interfaces, terminals

Not a perfect classification! Counter-example: Clocks

Context



In this lecture:

1. What happens between these two ?
   - Components involved and how
2. How does a device interact ?
   - Needed abstractions/architecture
3. What happened from the Process, OS and IO devices point of view ?

Context

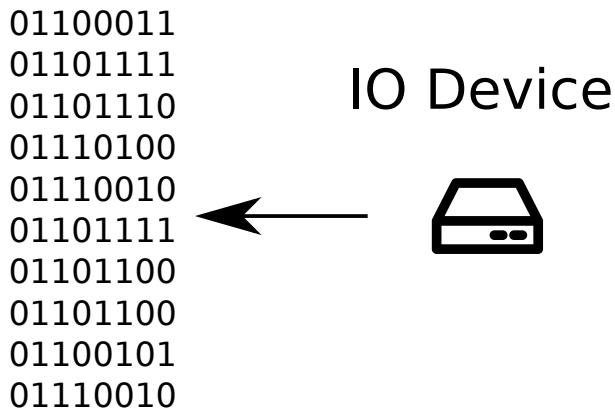
Challenges with I/O devices:

- Different aim: storage, networking, displays, etc.
- Large number of device to support

Goals of the OS:

- Provide a way to access I/O devices
  $\Rightarrow$ convenient, generic, consistent and reliable
- As device-independent as possible
- Without deteriorating the performances

Dealing with data coming from the IO device



Problem:

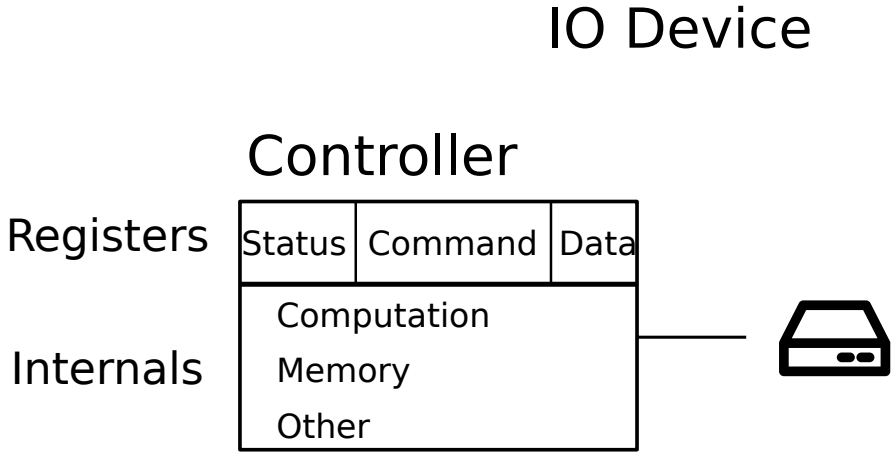
- Up to the programmer to understand the bits coming from the device ?
- Do the same for every (possible) device ?

Which solution to that problem?

The controller

Properties:

- HW device that works as a bridge between hardware and the OS
- Converts serial bit stream to a block of bytes + error corrections
- One per device or family of devices
- Owns a set of registers as interface

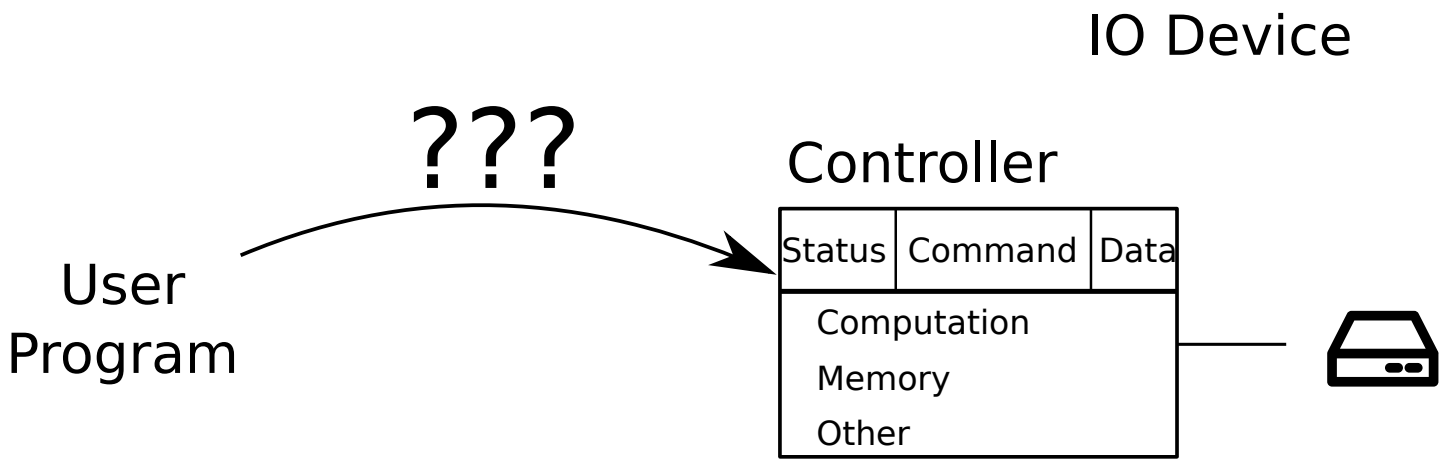


A device controller has a set of device registers:

- command (w-only): to configure and control the device
- status (r-only): provide status information about the device
- data (r/w): read data from or send data to the device

Accessed by a CPU via specific I/O instructions or memory mapping

From the user process point of view

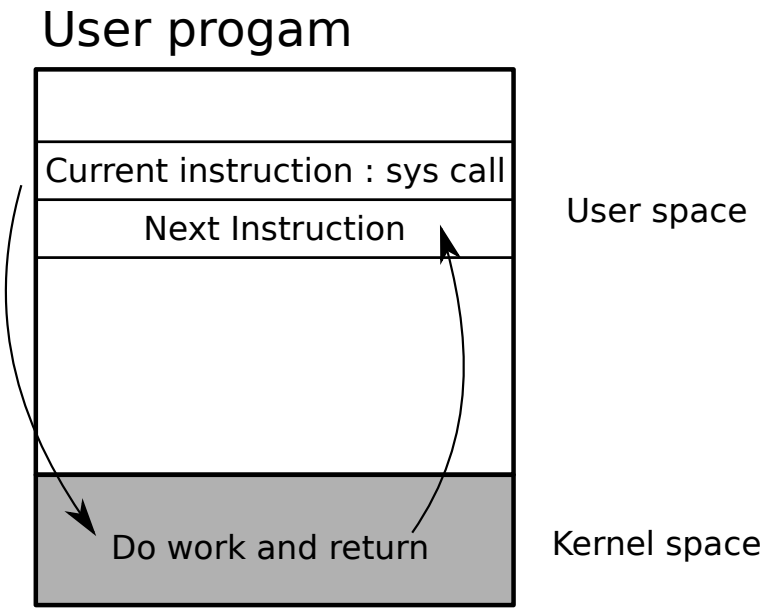


Problems:

- Complicated as controllers are too low level
- Controllers are very different and complex
- Still very low level and not generic at all

Which solution to these problems?

Reminder: system call

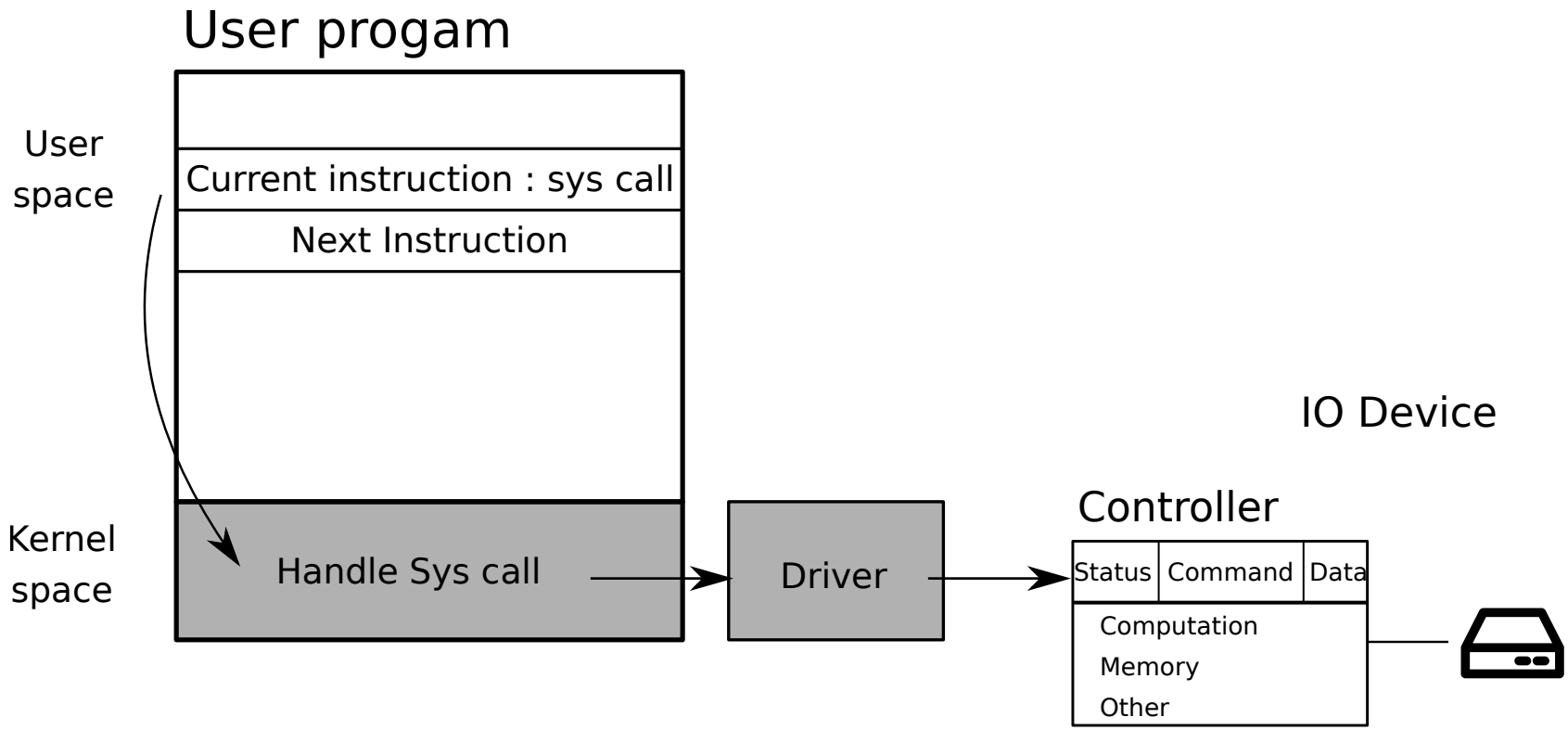


Syscall at the process level, simplified:

- To get a service from the OS, user program makes a system call
- Syscall is trapped into the kernel. Switch from user to kernel mode
- At this stage, user program is waiting for return of syscall
- When kernel is done, return at next instruction of user program
- Still, too many devices to deal with, can't have sys call per device

Which solution to these problems?

System calls, driver and controller



Driver to controllers interaction

- To get a service from a device, the OS delegates the discussion to a driver process
- Usually implemented by device vendors

Responsibilities of the device driver

In summary:

- Isolate the users and OS from the hardware
- Interface that provides standard interfaces to the OS and hardware
- Permits machine independent operations from the kernel

Responsibilities:

- Initialize devices
- Interpreting commands from OS
- Schedule multiple requests
- Manage data transfers
- Accept and process interrupts

Device driver, simplified behaviour

- Check input parameters for validity, and translate them to device-specific language
- Check if device is free (wait or block if not)
- Issue commands to control device
  - Write commands into device controller's registers
  - Check after each if device is ready for next (wait or block if not)
- Block or wait for controller to execute work
- Check for errors, and pass data to device-independent software
- Return status information
- Process next queued request, or block waiting for next

Challenges:

- Must be re-entrant (can be called by an interrupt while running)
- Handle hot-pluggable devices and device removal while running
- Complex and many of them: can crash system if buggy

Different types of Device drivers

Kernel-level drivers

- Critical devices, must keep running (e.g. disks, display drivers)
- Strong interaction with the kernel
- Must be resilient and reliable
- Major effort in developing such drivers

User-level drivers

- Non Critical devices (e.g. USB devices)
- Does not crash the system if it is buggy or if it fails

Zoom on the previous driver to controller interaction

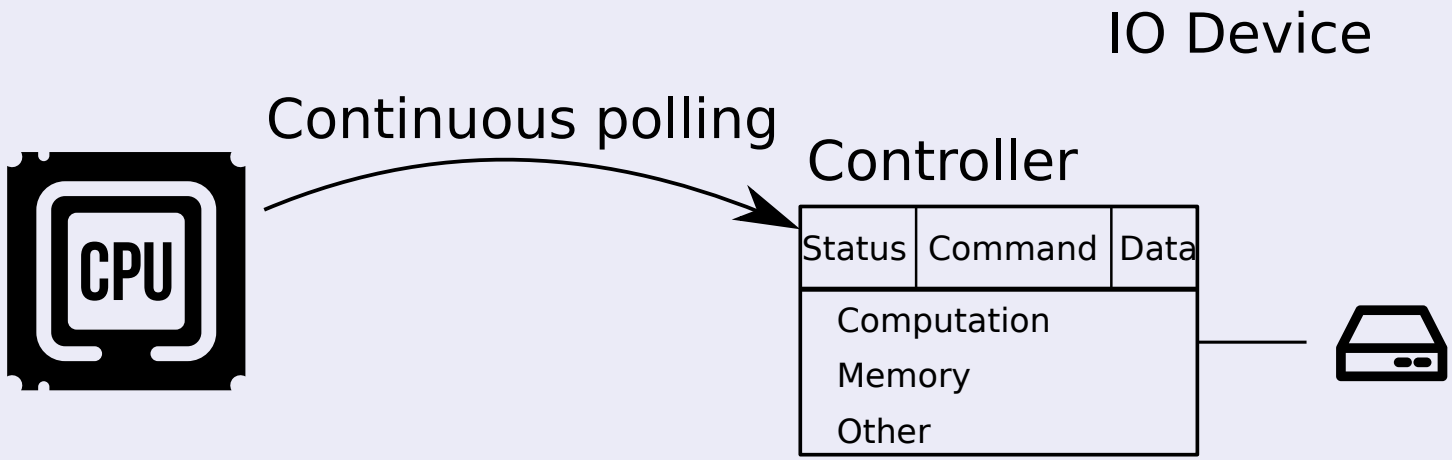


Figure: The CPU is executing the driver call. It checks, periodically, if the status of the device has changed: polling

Zoom on the previous driver to controller interaction

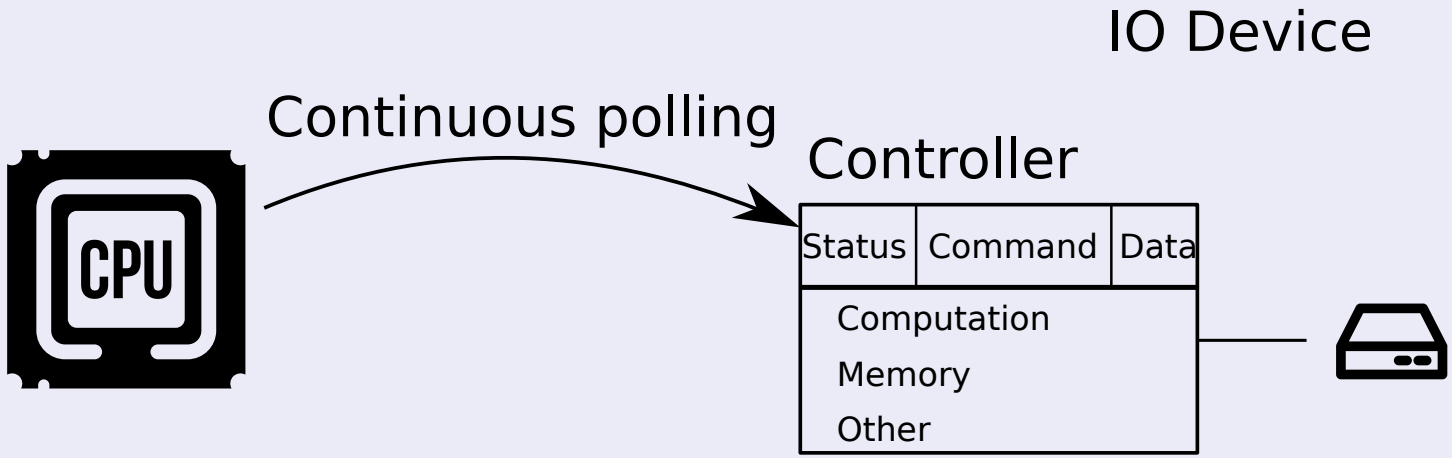
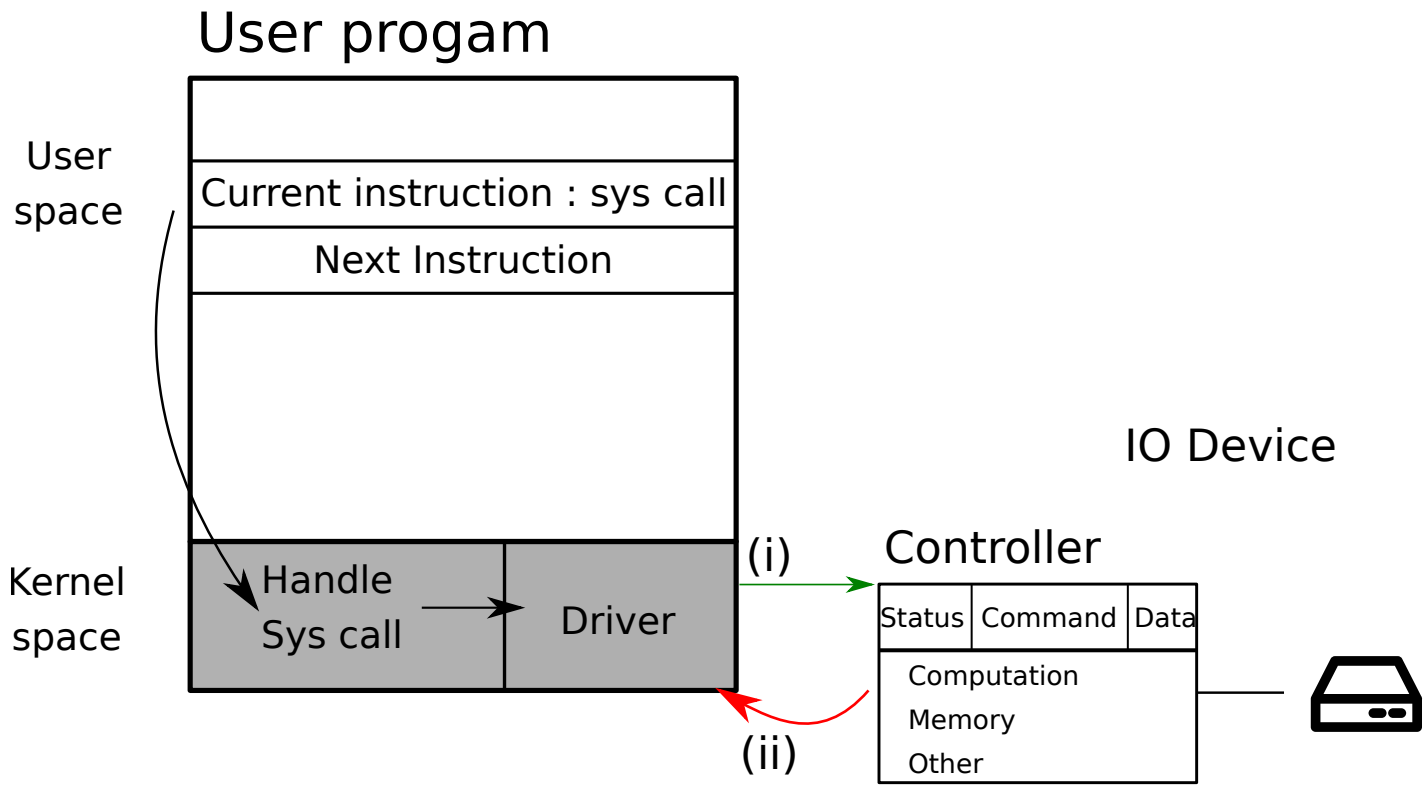


Figure: The CPU is executing the driver call. It checks, periodically, if the status of the device has changed: polling

Programmed I/O

- Each byte is transferred via CPU load / store
- Problem : busy waiting
- The CPU is actively waiting for an event to happen. Results in low utilization !

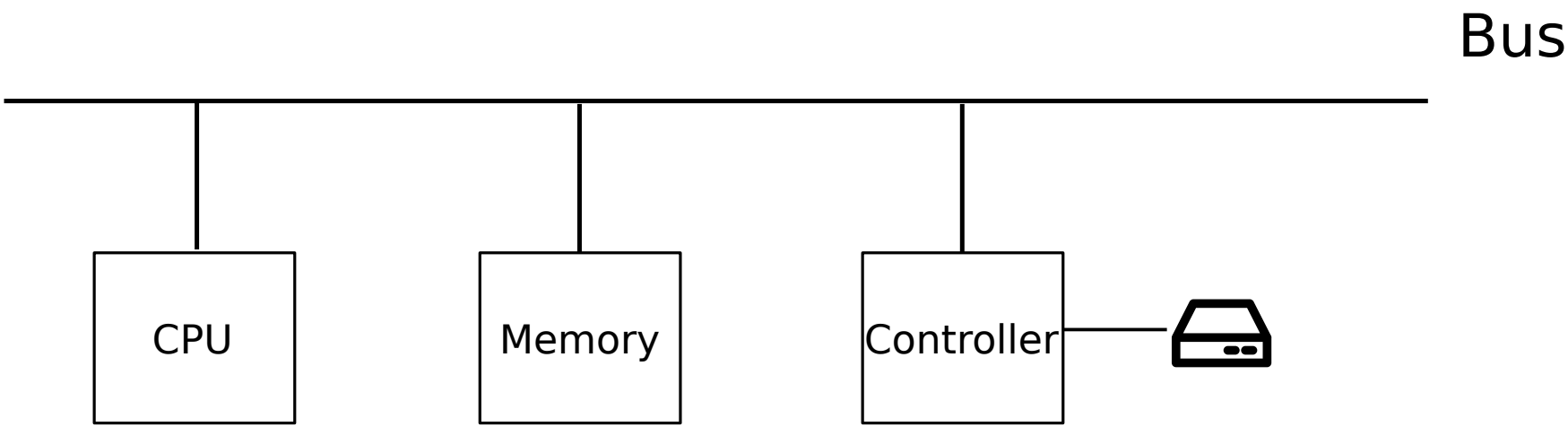
Interrupt



Driver to controllers interaction, with interrupt

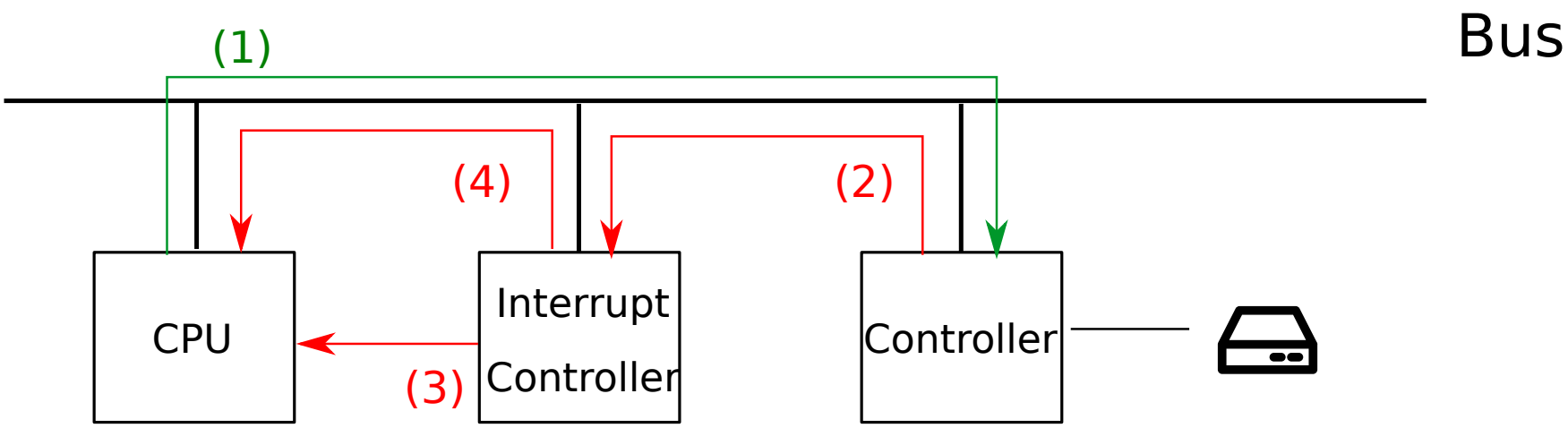
- (i) driver asks for controller to do work and to give an interrupt when asked task is done. The driver then returns
- (ii) When work is done on I/O device, controller generates interrupt

Reminder: simple layout of components connected in a computer



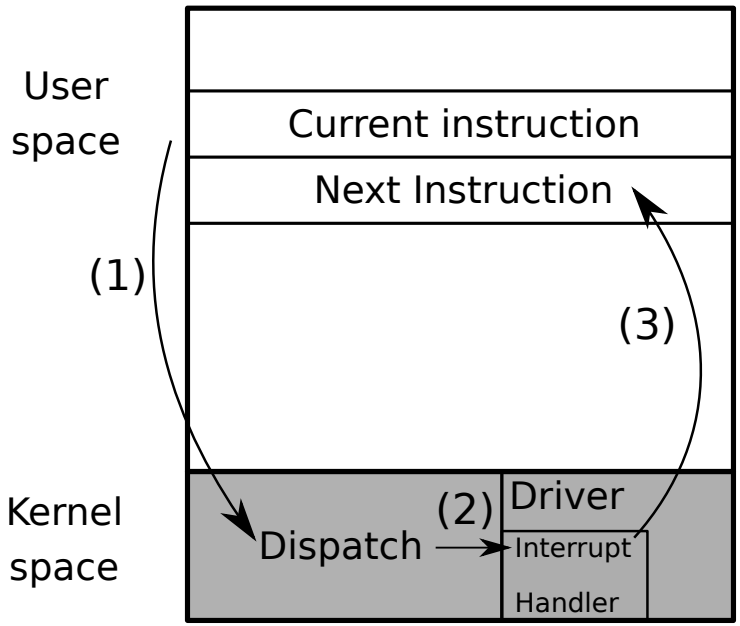
- All components connected by a system bus
- Communication from one to the other is done through it

Interrupt, hardware perspective



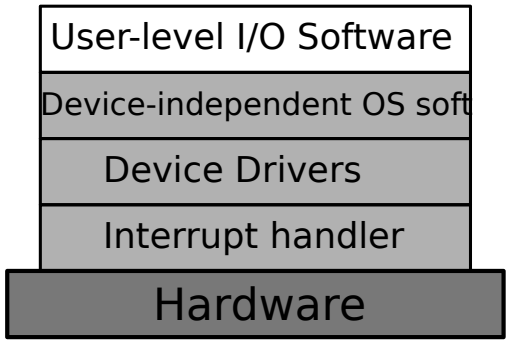
- 1) Driver process asks for a specific action by modification of controller registers
- 2) Interrupt is raised by the "Controller"
- 3) When "Interrupt Controller" has handled (2), it asserts the CPU that an interrupt is pending
- 4) "Interrupt Controller" points to the devices that issued the interrupt

Interrupt, CPU and user program perspective



- 1) When CPU decides to take the interrupt, saves the context and switch to kernel mode. ACK to the interrupt controller.
- 2) Kernel dispatches to the correct driver, thanks to the interrupt vector. The driver starts its interrupt handler
- 3) When handler is done, returns to the next instruction of user program

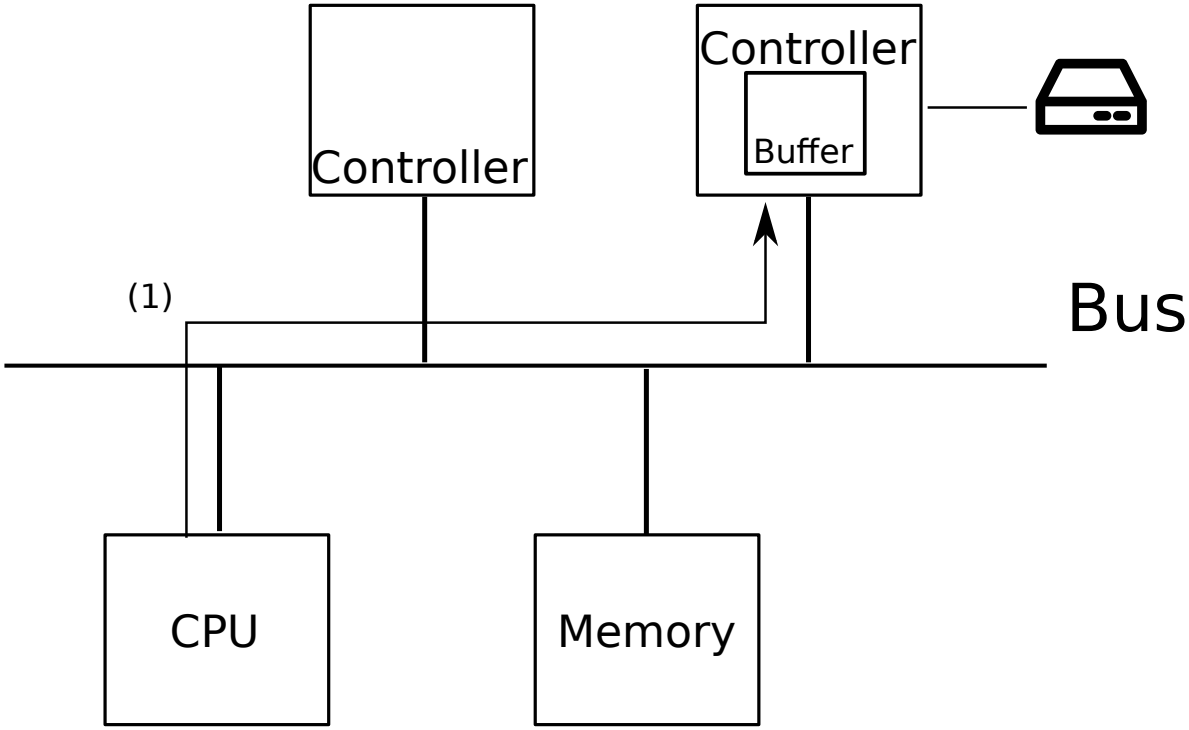
I/O Software Layers



Each layer has

- A well defined function to perform
- A well defined interface to the adjacent layers

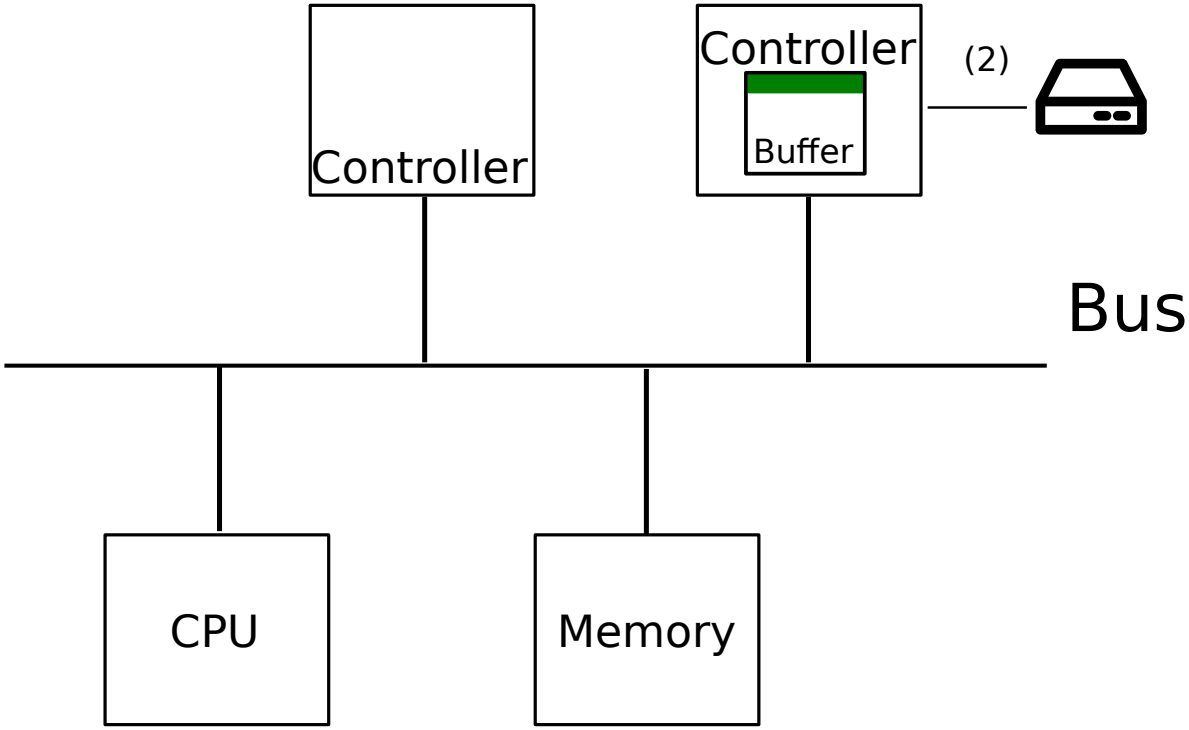
How disk reads occur, interrupt-driven IO



Interrupt-driven IO

- (1) CPU command disk controller to locally buffer data

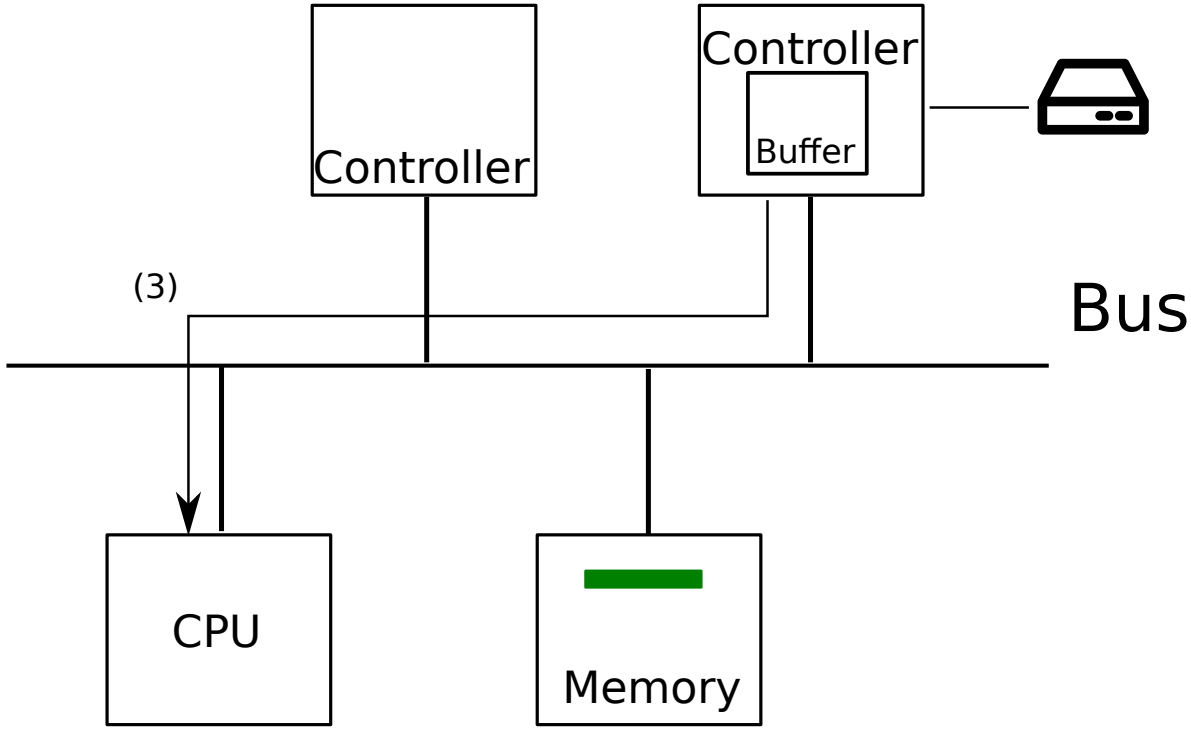
How disk reads occur, interrupt-driven IO



Interrupt-driven IO

- (1) CPU command disk controller to locally buffer data
- (2) Disk controller reads the block until the entire block is in controller's buffer

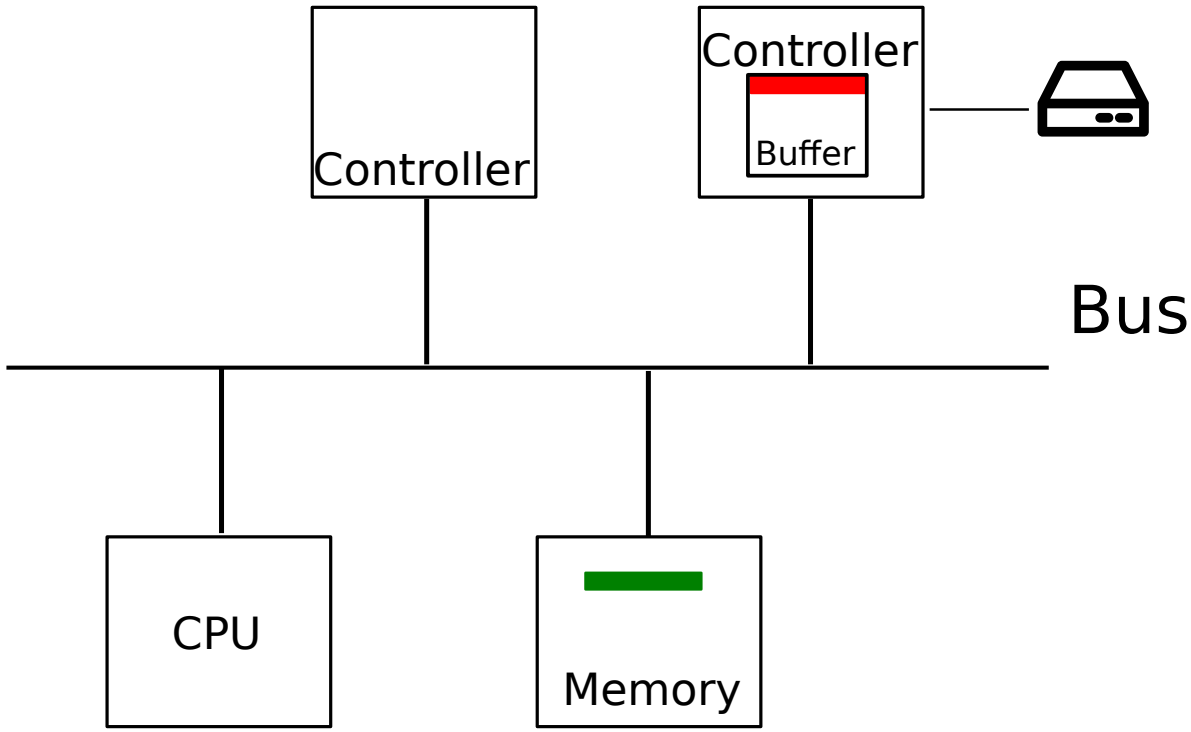
How disk reads occur, interrupt-driven IO



Interrupt-driven IO

- (1) CPU command disk controller to locally buffer data
- (2) Disk controller reads the block until the entire block is in controller's buffer
- Controller verifies that no errors have occurred (like check-sum) and (3) causes an interrupt

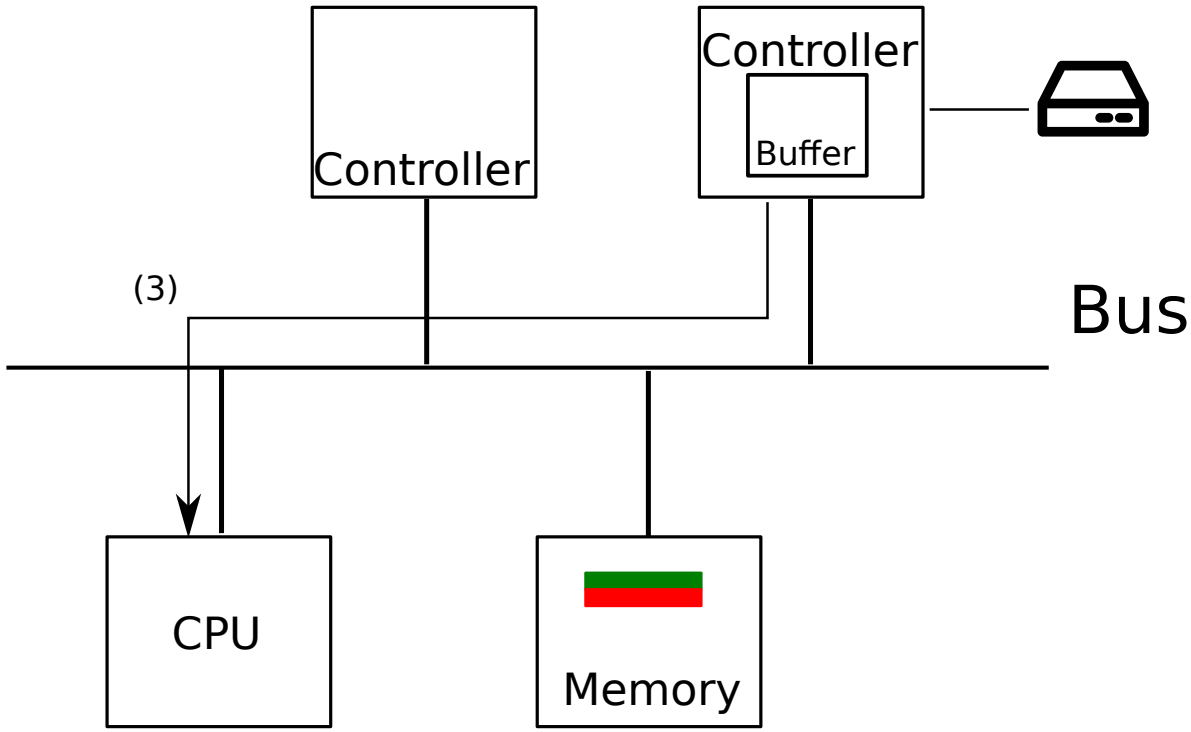
How disk reads occur, interrupt-driven IO



Interrupt-driven IO

- (1) CPU command disk controller to locally buffer data
- (2) Disk controller reads the block until the entire block is in controller's buffer
- Controller verifies that no errors have occurred (like check-sum) and (3) causes an interrupt
- OS reads from the controller's buffer, a byte (or a word) at a time from a device register, to store it in main memory

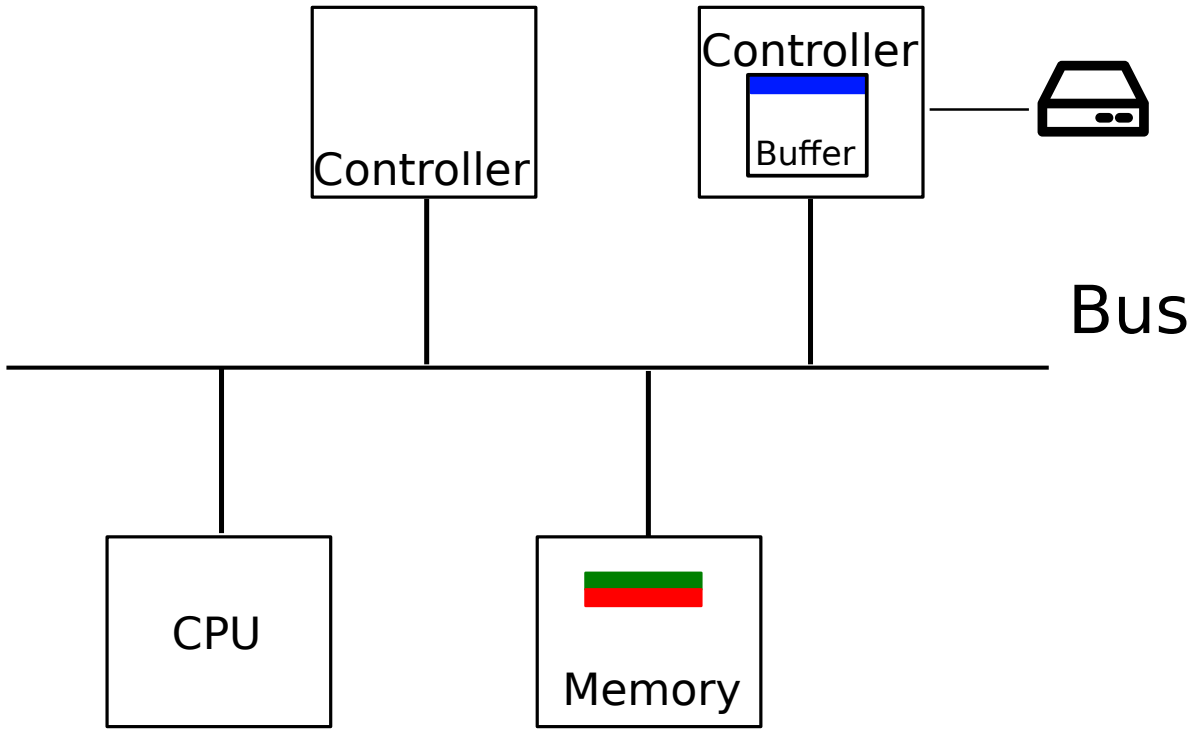
How disk reads occur, interrupt-driven IO



Interrupt-driven IO

- (1) CPU command disk controller to locally buffer data
- (2) Disk controller reads the block until the entire block is in controller's buffer
- Controller verifies that no errors have occurred (like check-sum) and (3) causes an interrupt
- OS reads from the controller's buffer, a byte (or a word) at a time from a device register, to store it in main memory

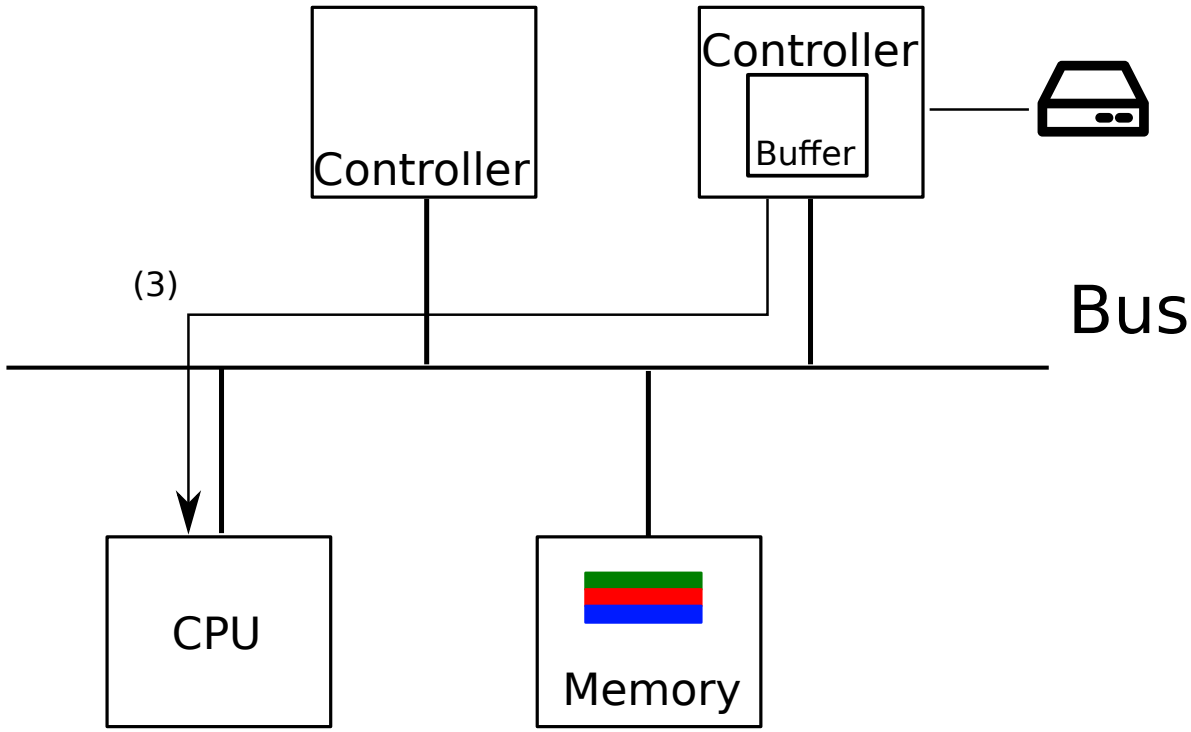
How disk reads occur, interrupt-driven IO



Interrupt-driven IO

- (1) CPU command disk controller to locally buffer data
- (2) Disk controller reads the block until the entire block is in controller's buffer
- Controller verifies that no errors have occurred (like check-sum) and (3) causes an interrupt
- OS reads from the controller's buffer, a byte (or a word) at a time from a device register, to store it in main memory

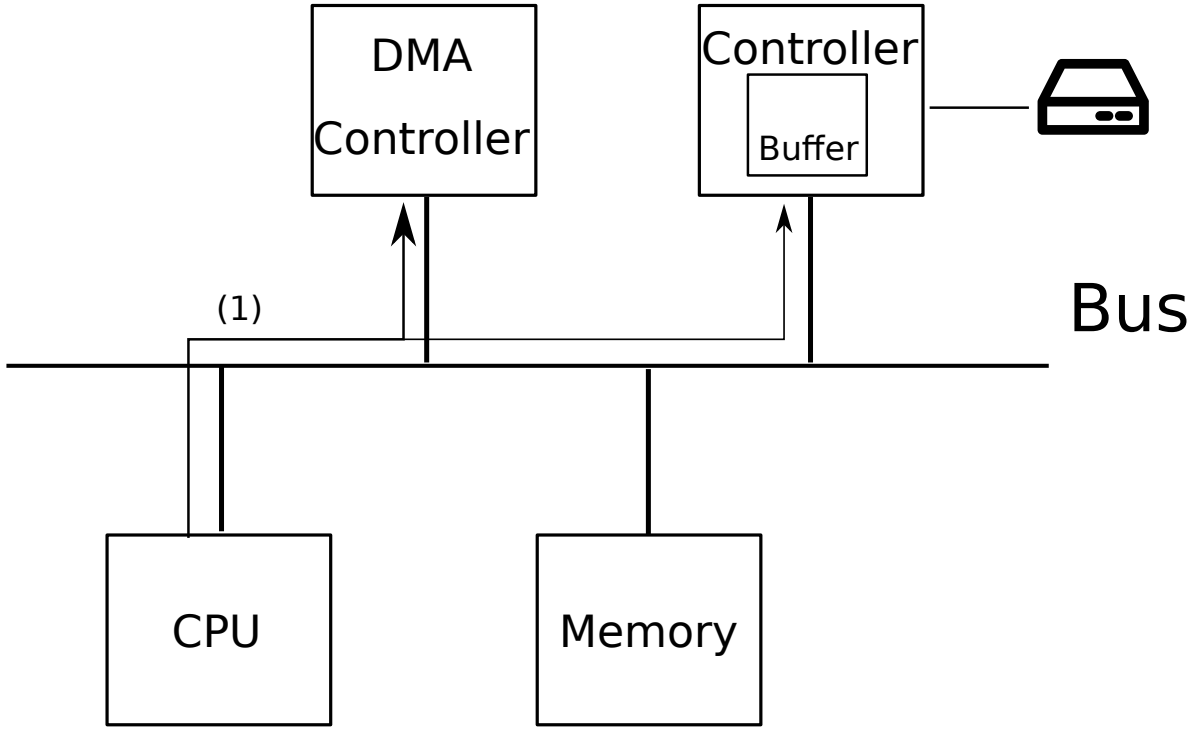
How disk reads occur, interrupt-driven IO



Interrupt-driven IO

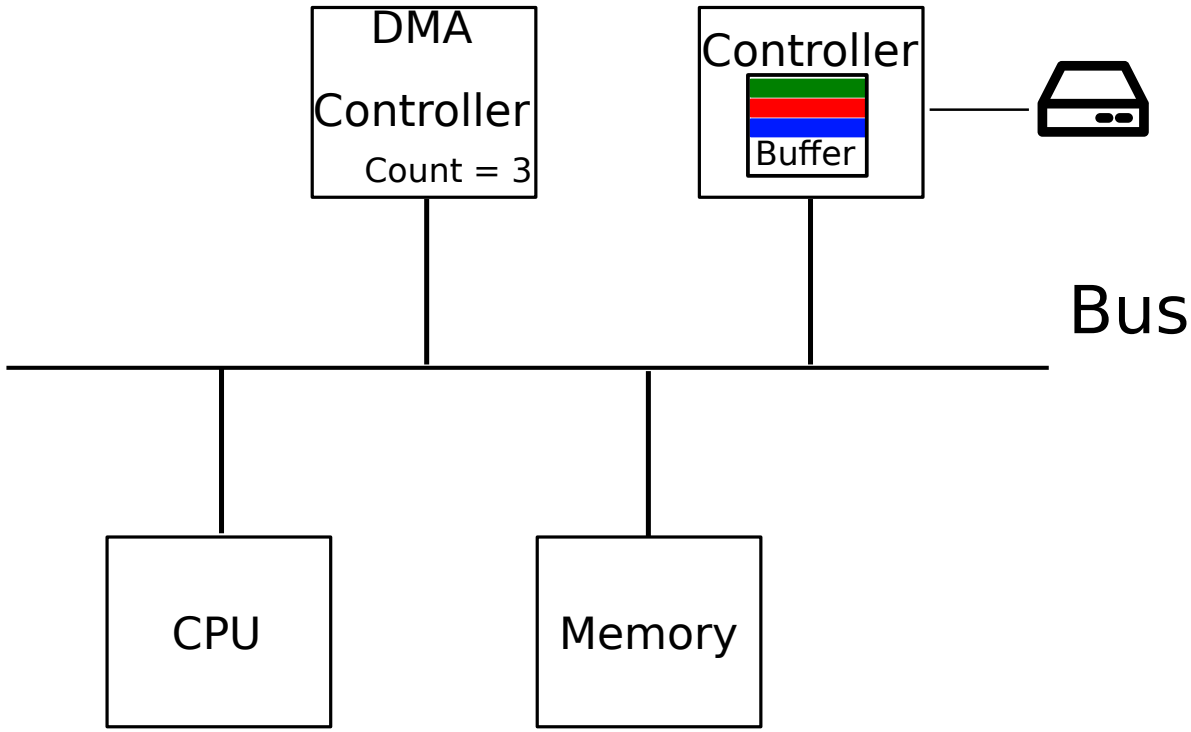
- (1) CPU command disk controller to locally buffer data
- (2) Disk controller reads the block until the entire block is in controller's buffer
- Controller verifies that no errors have occurred (like check-sum) and (3) causes an interrupt
- OS reads from the controller's buffer, a byte (or a word) at a time from a device register, to store it in main memory

How disk reads occur, Direct Memory Access



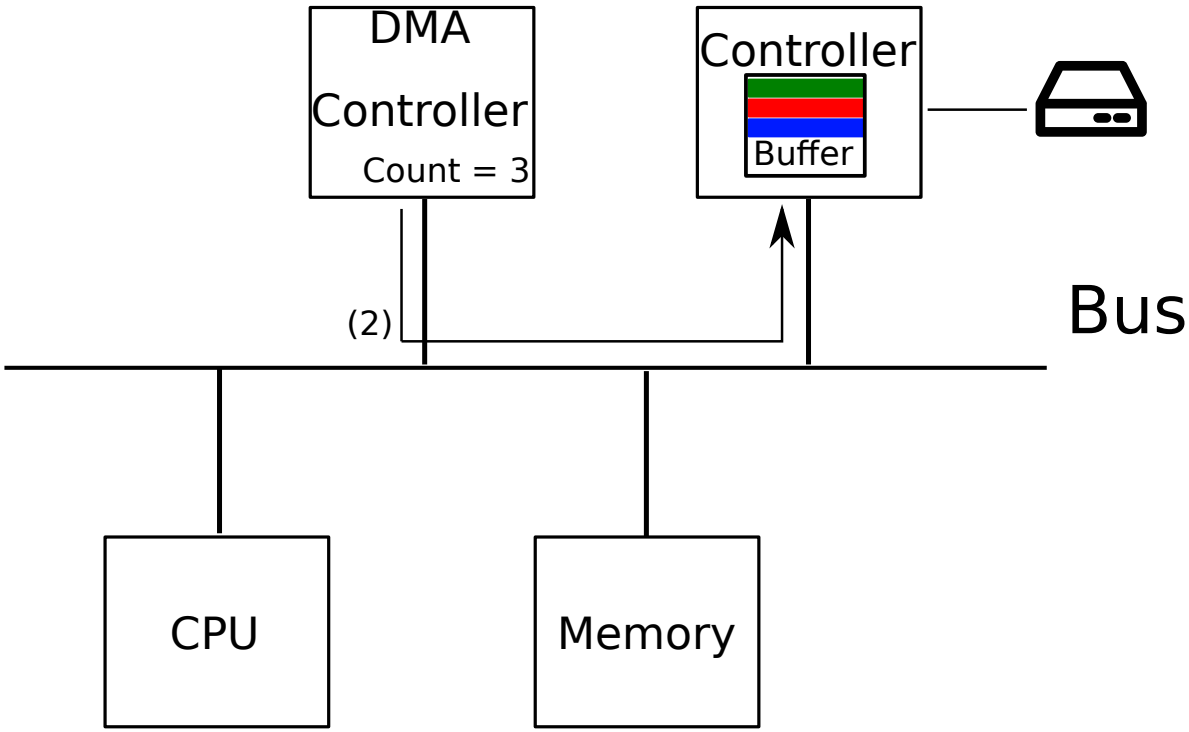
- (1) CPU programs the DMA controller (what and where to transfer) + command to the disk controller (from disk to internal controller's buffer)

How disk reads occur, Direct Memory Access



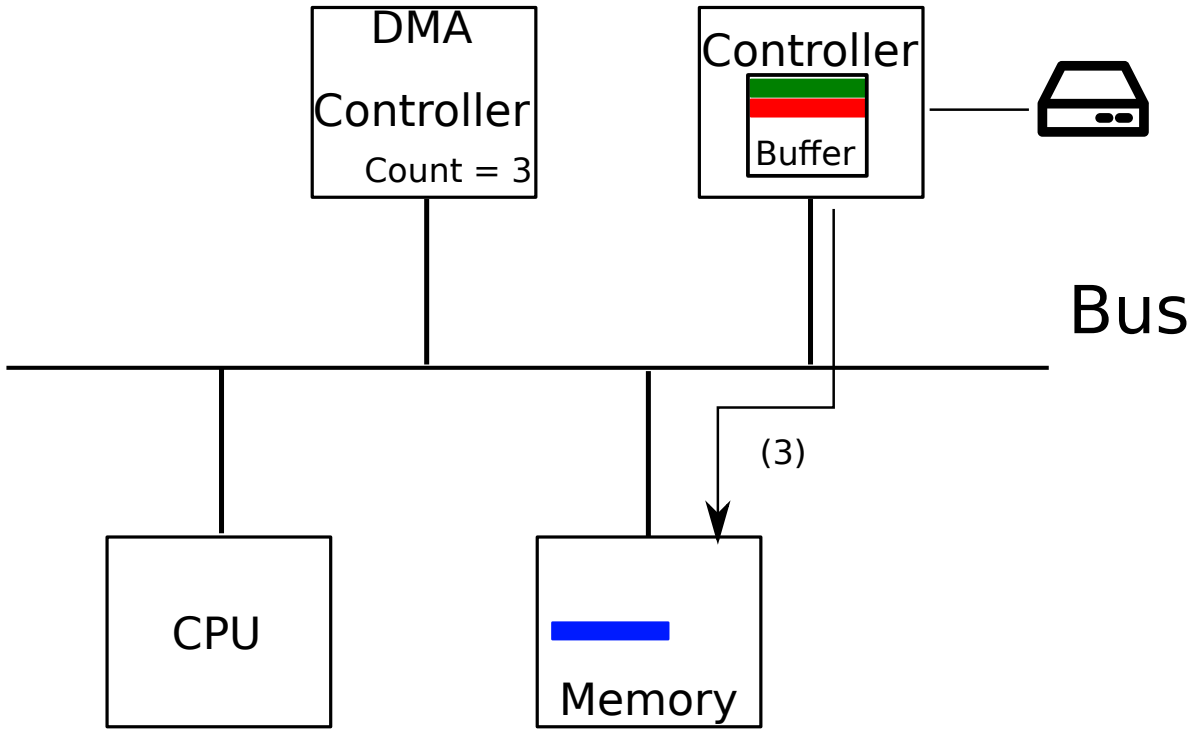
- (1) CPU programs the DMA controller (what and where to transfer) + command to the disk controller (from disk to internal controller's buffer)
- (2) DMA Controller issues read request to disk controller to transfer to main memory

How disk reads occur, Direct Memory Access



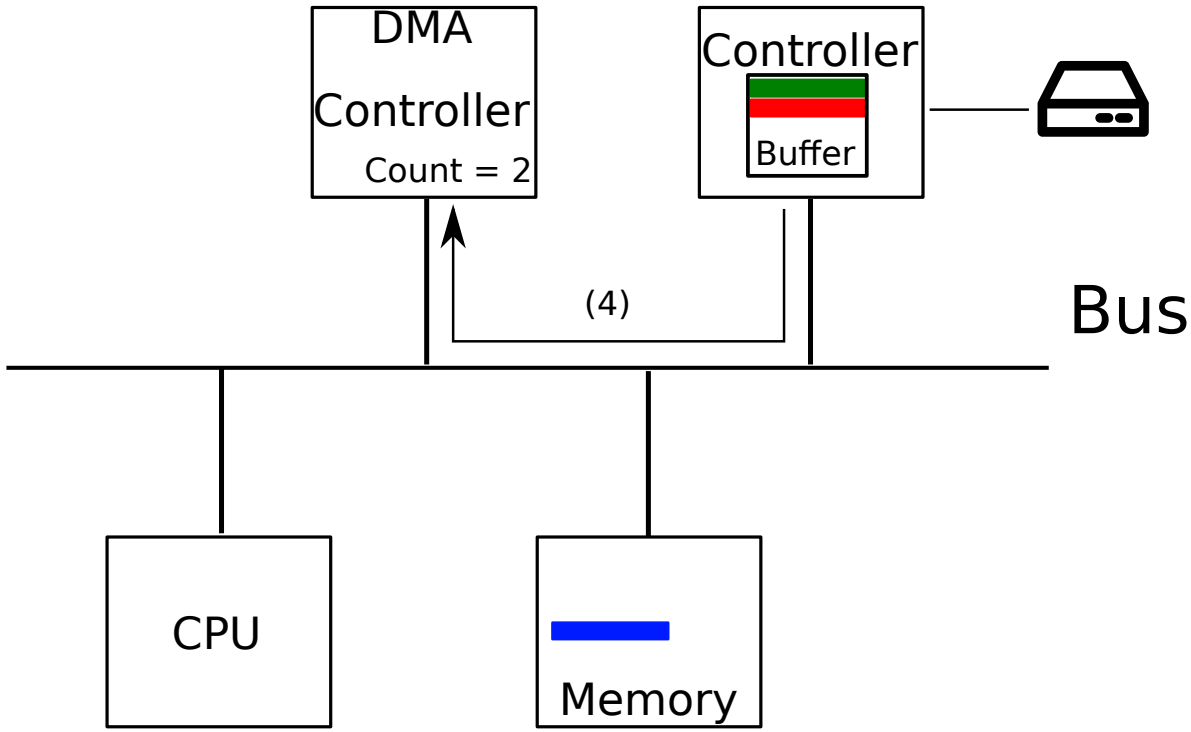
- (1) CPU programs the DMA controller (what and where to transfer) + command to the disk controller (from disk to internal controller's buffer)
- (2) DMA Controller issues read request to disk controller to transfer to main memory

How disk reads occur, Direct Memory Access



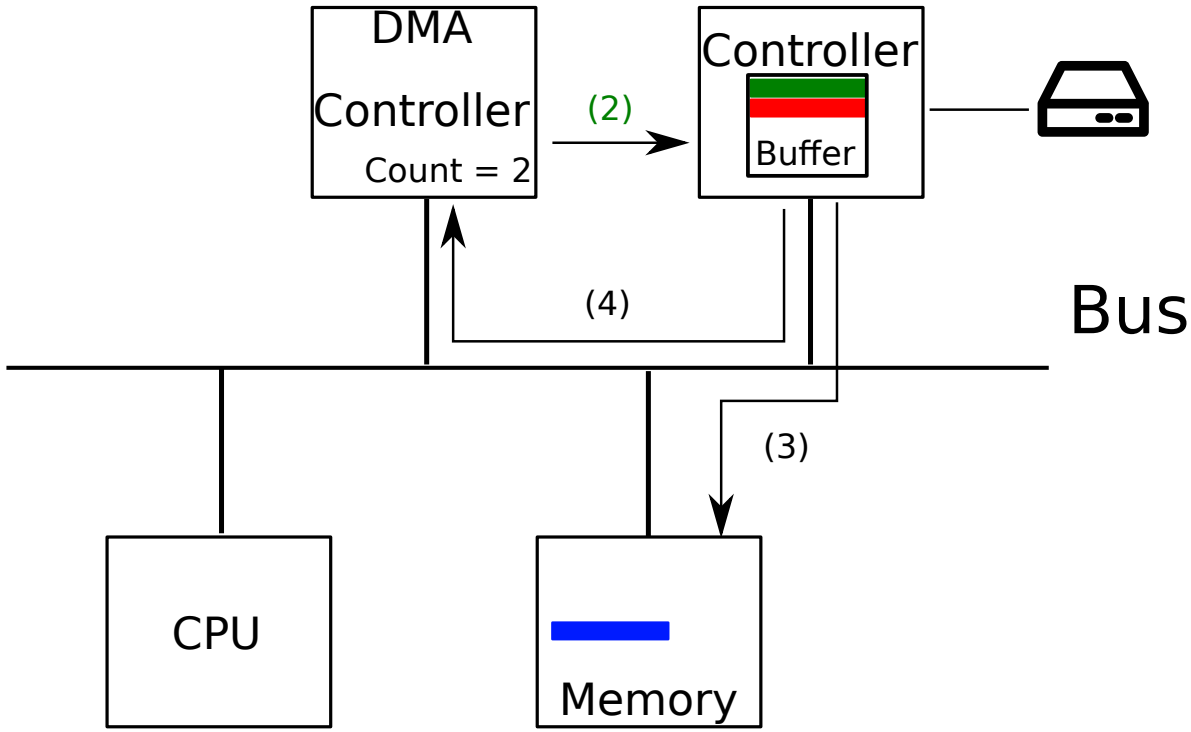
- (1) CPU programs the DMA controller (what and where to transfer) + command to the disk controller (from disk to internal controller's buffer)
- (2) DMA Controller issues read request to disk controller to transfer to main memory
- When (3) Disk controller has written a word in main memory, (4) ACK to DMA controller, *count -= 1;* on DMA

How disk reads occur, Direct Memory Access



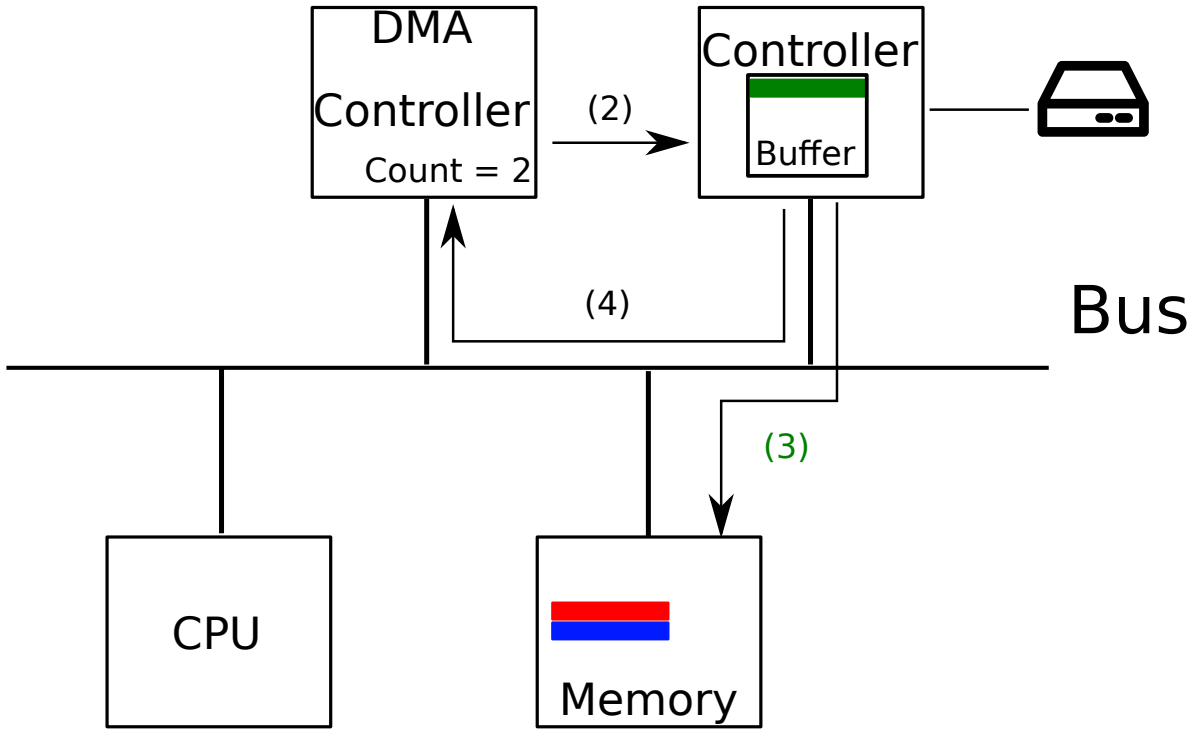
- (1) CPU programs the DMA controller (what and where to transfer) + command to the disk controller (from disk to internal controller's buffer)
- (2) DMA Controller issues read request to disk controller to transfer to main memory
- When (3) Disk controller has written a word in main memory, (4) ACK to DMA controller, *count -= 1;* on DMA

How disk reads occur, Direct Memory Access



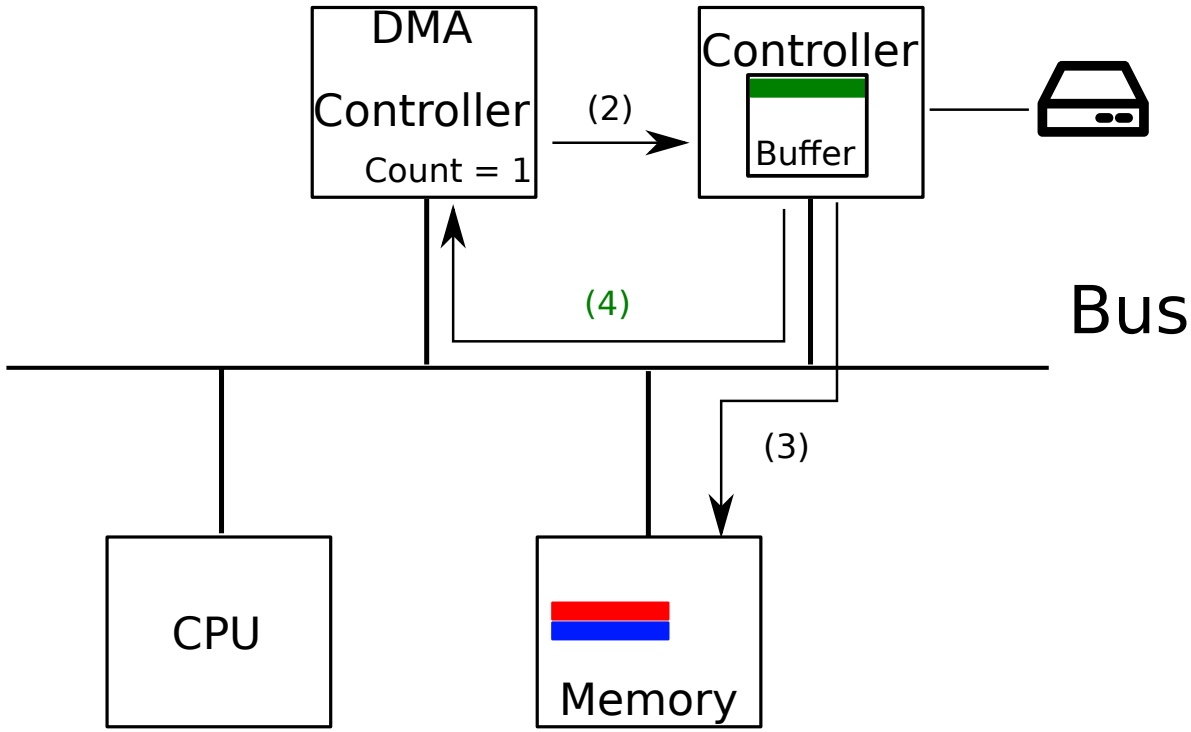
- (1) CPU programs the DMA controller (what and where to transfer) + command to the disk controller (from disk to internal controller's buffer)
- (2) DMA Controller issues read request to disk controller to transfer to main memory
- When (3) Disk controller has written a word in main memory, (4) ACK to DMA controller, *count -= 1;* on DMA

How disk reads occur, Direct Memory Access



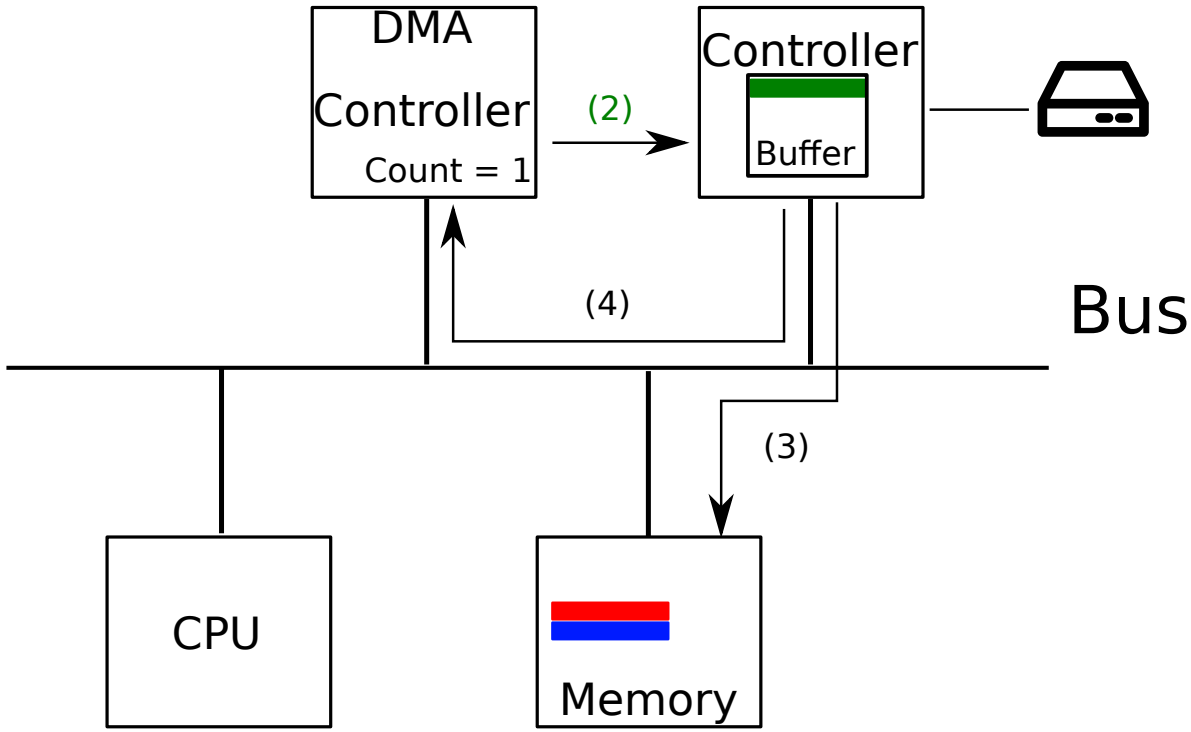
- (1) CPU programs the DMA controller (what and where to transfer) + command to the disk controller (from disk to internal controller's buffer)
- (2) DMA Controller issues read request to disk controller to transfer to main memory
- When (3) Disk controller has written a word in main memory, (4) ACK to DMA controller, *count -= 1;* on DMA

How disk reads occur, Direct Memory Access



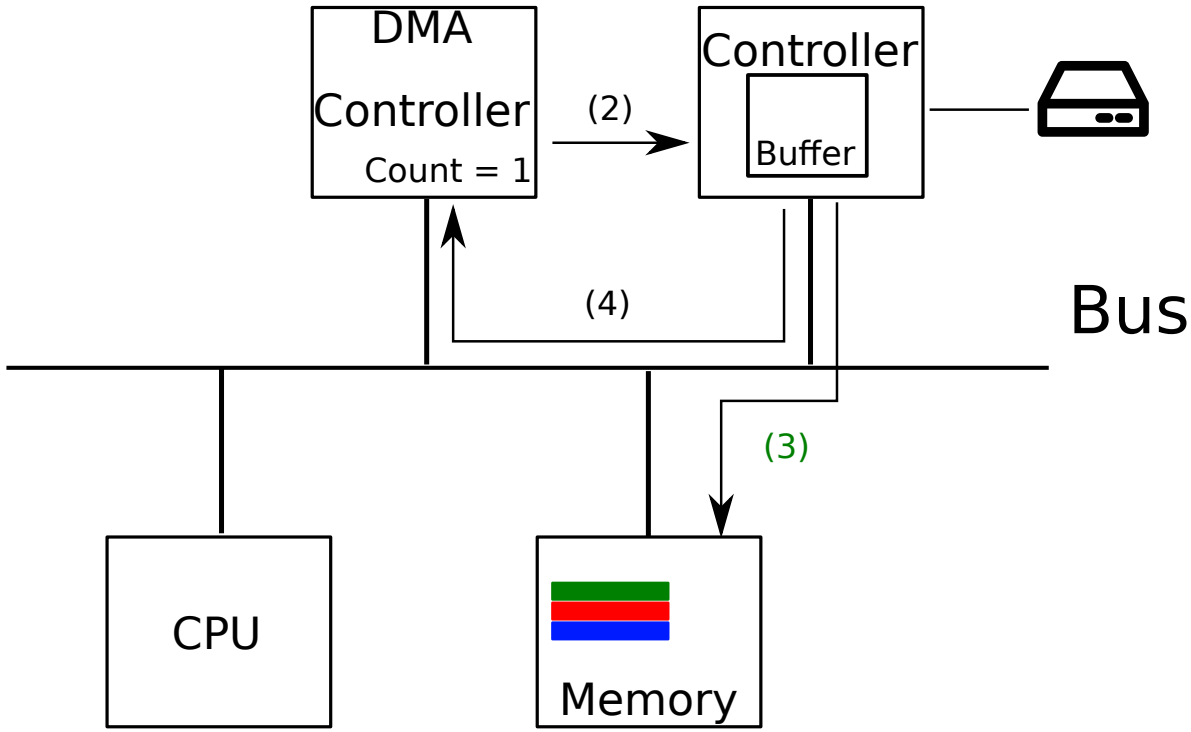
- (1) CPU programs the DMA controller (what and where to transfer) + command to the disk controller (from disk to internal controller's buffer)
- (2) DMA Controller issues read request to disk controller to transfer to main memory
- When (3) Disk controller has written a word in main memory, (4) ACK to DMA controller, *count -= 1;* on DMA

How disk reads occur, Direct Memory Access



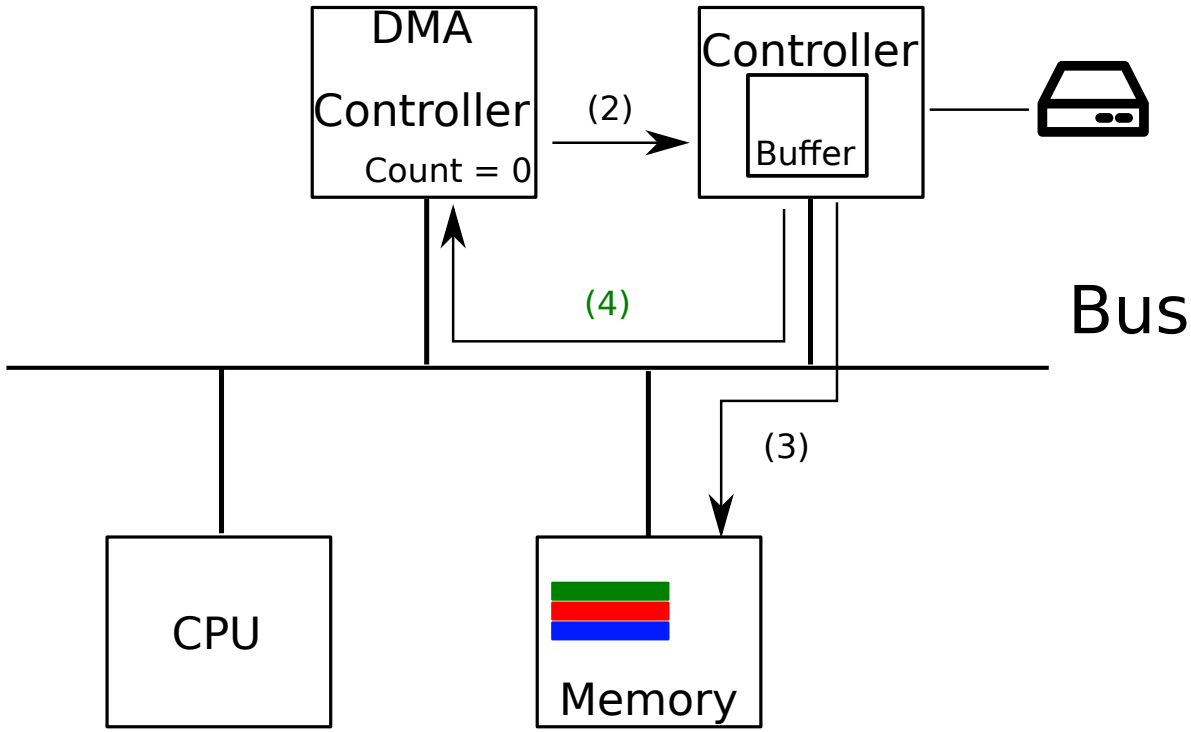
- (1) CPU programs the DMA controller (what and where to transfer) + command to the disk controller (from disk to internal controller's buffer)
- (2) DMA Controller issues read request to disk controller to transfer to main memory
- When (3) Disk controller has written a word in main memory, (4) ACK to DMA controller, *count -= 1;* on DMA

How disk reads occur, Direct Memory Access



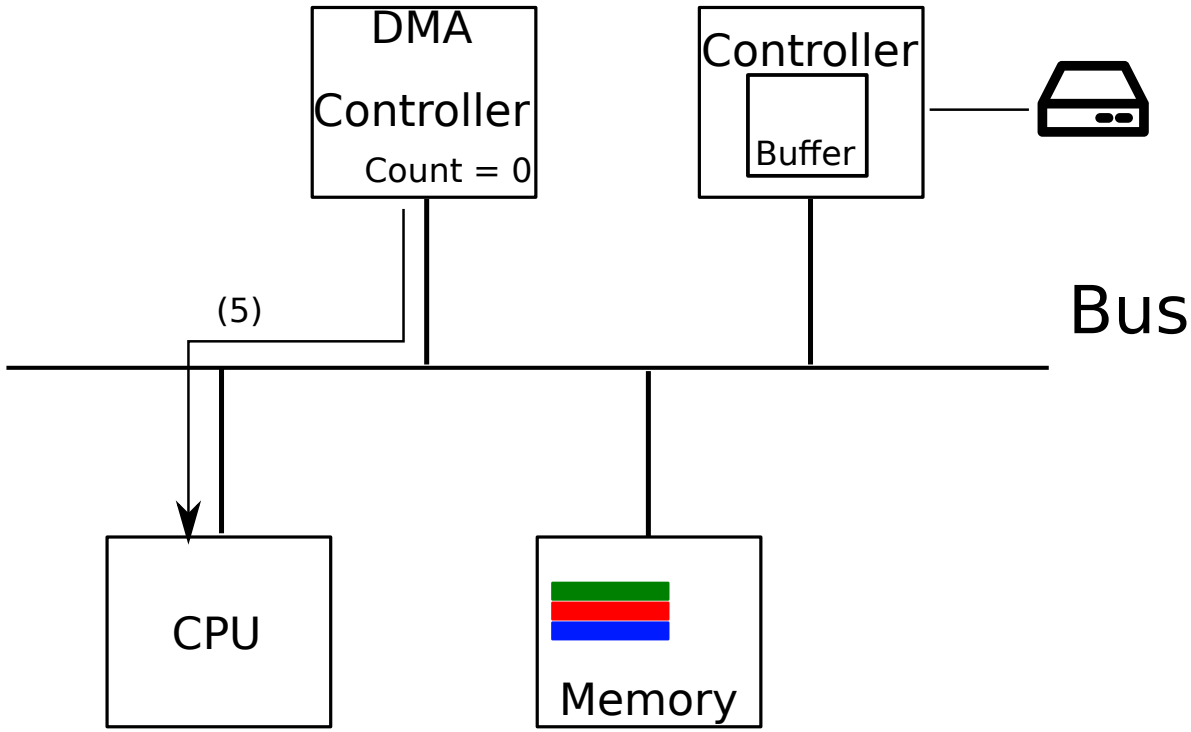
- (1) CPU programs the DMA controller (what and where to transfer) + command to the disk controller (from disk to internal controller's buffer)
- (2) DMA Controller issues read request to disk controller to transfer to main memory
- When (3) Disk controller has written a word in main memory, (4) ACK to DMA controller, *count -= 1;* on DMA

How disk reads occur, Direct Memory Access



- (1) CPU programs the DMA controller (what and where to transfer) + command to the disk controller (from disk to internal controller's buffer)
- (2) DMA Controller issues read request to disk controller to transfer to main memory
- When (3) Disk controller has written a word in main memory, (4) ACK to DMA controller, *count -= 1;* on DMA

How disk reads occur, Direct Memory Access



- (1) CPU programs the DMA controller (what and where to transfer) + command to the disk controller (from disk to internal controller's buffer)
- (2) DMA Controller issues read request to disk controller to transfer to main memory
- When (3) Disk controller has written a word in main memory, (4) ACK to DMA controller, *count -= 1;* on DMA
- When count == 0, (5) DMA controller interrupts the CPU
- Key difference: CPU almost not involved in the data transfer

Summary

I/O operations are important

- A big part of the OS
- Can have a huge impact on performance

Done in 3 ways

- Programmed I/O
- Interrupt-driven I/O
- DMA

References / Check out

Check-out

- Chapter 10 in textbook : Linux description
- Chapter 5 : Input/Output

References

- Textbook : Chapter 5, 1, 10
- "Operating Systems: Three Easy Pieces" : chapter 36, I/O Devices
- Linux Device Drivers, 4th Edition