# Course TÖL401G: Stýrikerfi / Operating Systems
# 13. I/O Systems

Mainly based on slides and figures

# Chapter Objectives

- Explore the structure of an operating system's I/O subsystem.
- Discuss the principles and complexities of I/O hardware.
- Explain the performance aspects of I/O hardware and software.

# Contents

# 13.1 Overview Input/Output Systems

- Magnitude of I/O devices (storage, transmission, human-interface):
  - Printer
  - Scanner
  - Keyboard
  - Mouse
  - Game controller
  - Digitiser touch screen/tablet
  - Hard disk
  - Flash memory SSD disk
  - CD/DVD/Blu-ray Reader & Writer
  - Tape drive
  - Graphic card
  - Sound card
  - GPS
  - Accelerometer
  - Wired & Wireless network
  - …
- Magnitude of hardware interfaces:
  - USB
  - (Serial) ATA
  - NVM Express (NVMe)
  - Serial-attached SCSI (SAS)
  - Thunderbolt,
  - PCI Express (PCIe),
  - Legacy interfaces (parallel, serial),
  - Ethernet, …

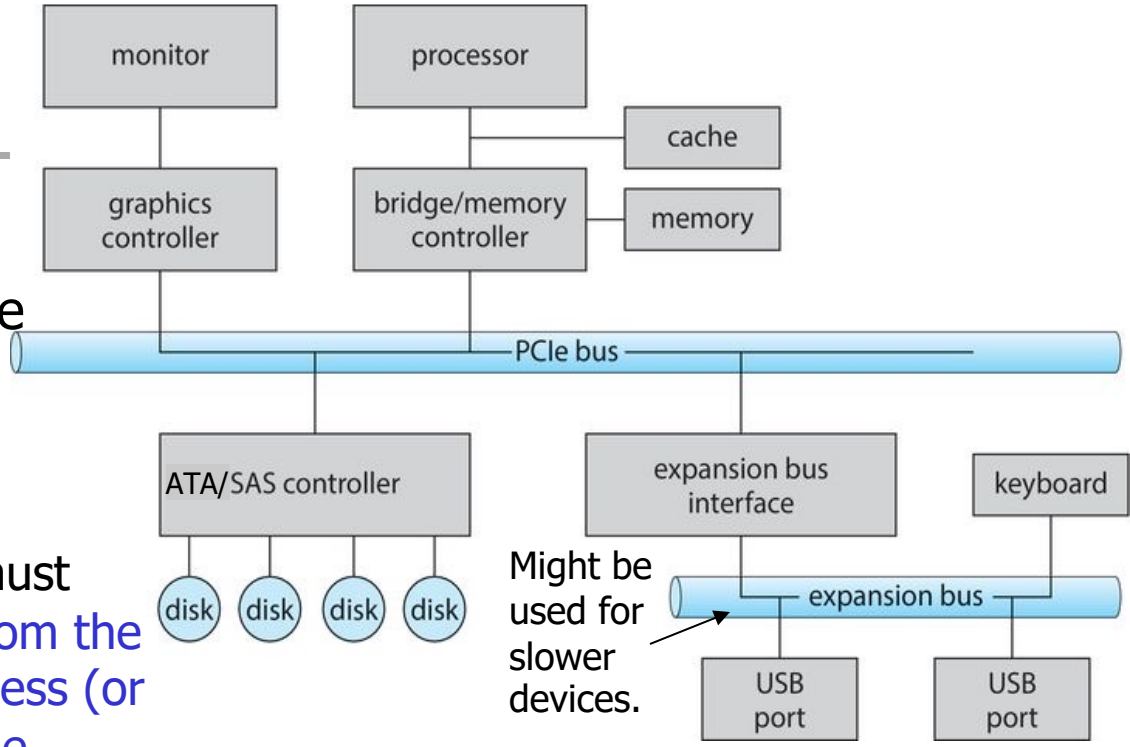
# I/O and the Operating System

- Responsibility of the OS concerning I/O devices:
  - Manage I/O devices (using device drivers):
    - Provide processes/programmer a simple, but controlled access.
    - In multiprogramming systems:
      - Provide a fair access to I/O device used by multiple processes.
    - Minimise I/O performance bottleneck using suitable scheduling.
      - I/O devices are much much slower than CPU and main memory.
        - Avoid that I/O-intensive processes slow down CPU-intensive processes!
  - Cope with magnitude of different devices and interfaces:
    - Application developer should not have to deal with magnitude.

  $\Rightarrow$Provide a standard interface for unified access to devices with comparable characteristics.

# 13.2 I/O Hardware

- How do device drivers (=software) communicate with the actual I/O devices or their device controllers (=hardware)?
  - OS and I/O hardware must support sending data from the address space of a process (or of the OS performing the respective I/O system call) to an I/O device and vice-versa.
  - Devices are either directly connected to the CPU via the data and address bus, or via some peripheral bus, e.g. the PCIe bus (having multiple PCIe "lanes").



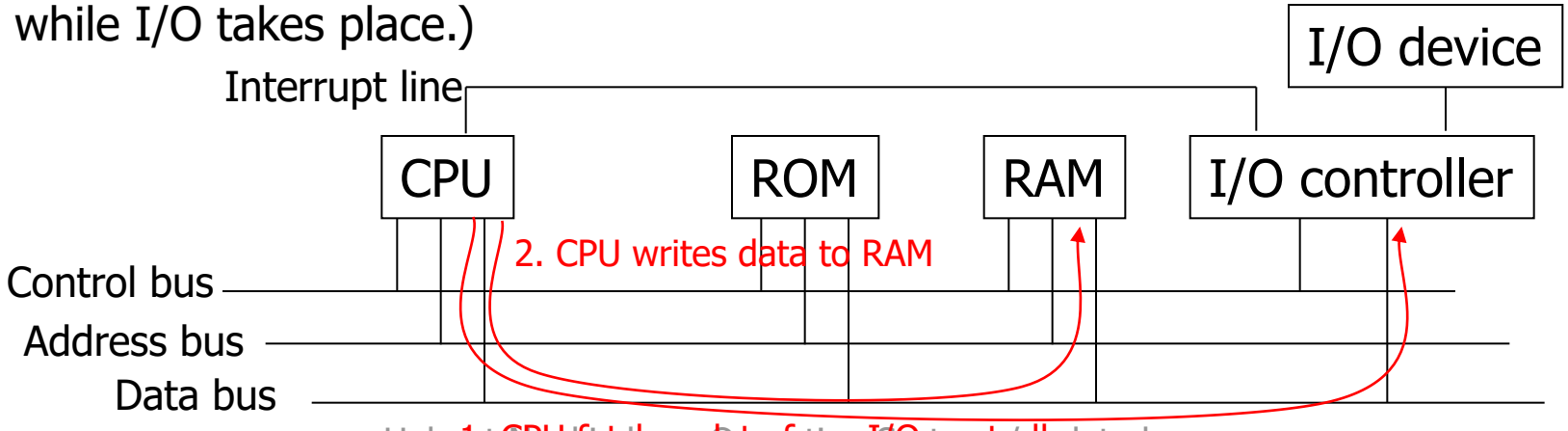

- Note: Nowadays, many controllers are not plugged into a PCIe slot, but integrated into the bridge/memory controller (it's "southbridge" part).
- In fact, since 2011, the bridge/memory controller (former "northbridge") is part of the CPU chip itself, and the device controllers (incl. integrated graphics) are located in the Platform Controller Hub (PCH) chip.
- Starting from 2014 also the PCH is sometimes part of the CPU chip itself.

# I/O Hardware

- Possible ways for connecting an I/O controller to a CPU or main memory (more on next slides):
  - Programmed I/O (PIO),
  - Memory-mapped I/O,
  - Direct Memory Access (DMA).
- Knowing the different characteristics of these hardware connections may help minimising the I/O performance bottleneck.
  - Optimal: While I/O device is working (corresponding process is blocked), CPU can be assigned to other (ready) process.

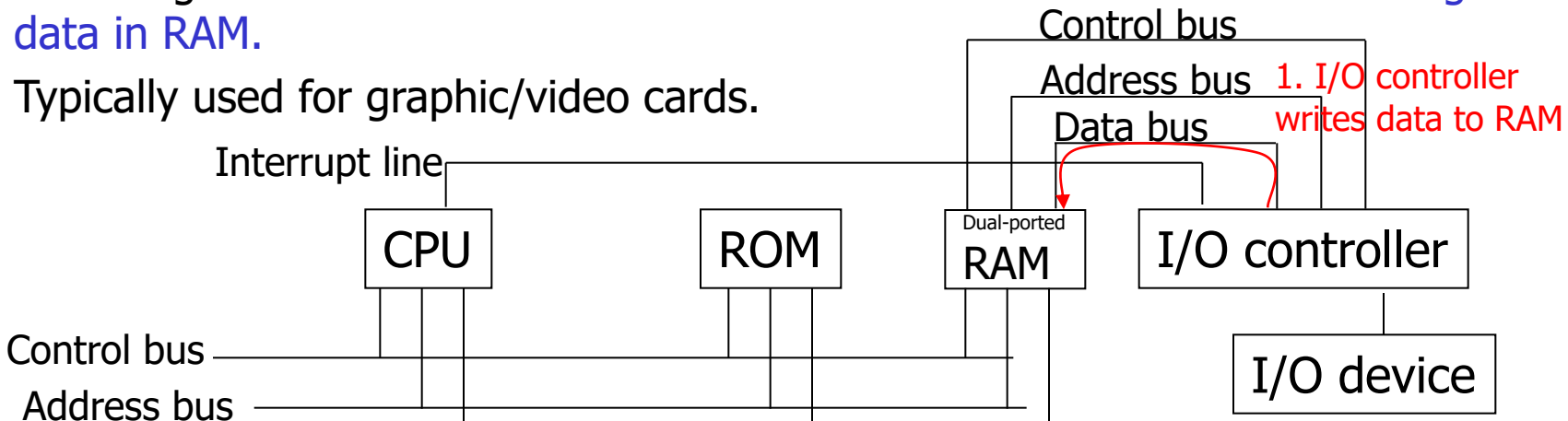
# Programmed I/O (PIO)

- CPU is connected to a data register ("I/O port") of the controller, i.e. device driver or CPU respectively have to read/write data byte-wise from/to controller.
  - Polling: Device driver (=CPU) has to read periodically status register of controller whether data needs to be transferred from/to controller. (Disadvantage: busy waiting)
  - Interrupt: Controller raises interrupt if data must be transferred. Interrupt handler dispatches to device driver that transfers actual data from/to controller. (Disadvantage: interrupt handling is slow due to involved context switch.)
- Disadvantage of PIO: not suitable for multiprogramming, as byte-wise transfer of data is very CPU intensive! (I.e. CPU cannot be assigned to some ready process while I/O takes place.)



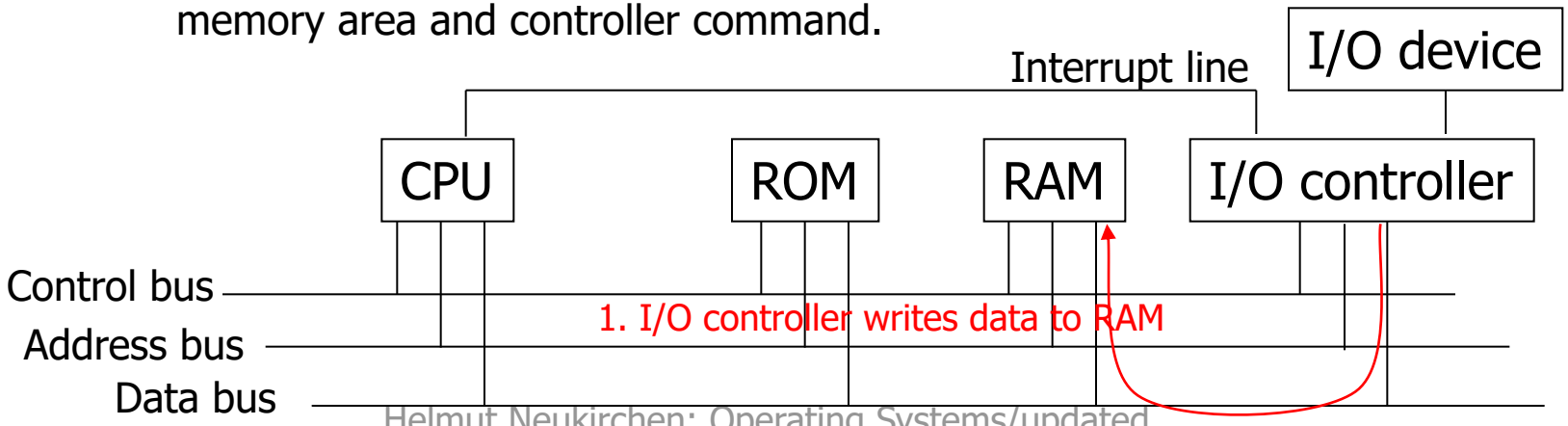
# Memory-Mapped I/O

- Controller (e.g. graphic card) has its own RAM memory that is mapped into the physical address space of the CPU.
- Disadvantage: RAM memory that can by used by two parties at the same time (i.e. by controller & CPU), is either more expensive (dual-ported RAM) or slower (shared Memory: I/O controller steals bus memory cycles from CPU to access ordinary (=single ported) RAM – however no problem if CPU cache contains all data needed by CPU) than ordinary RAM.
- Advantage: Less CPU intensive. CPU can work while controller is accessing data in RAM.
- Typically used for graphic/video cards.

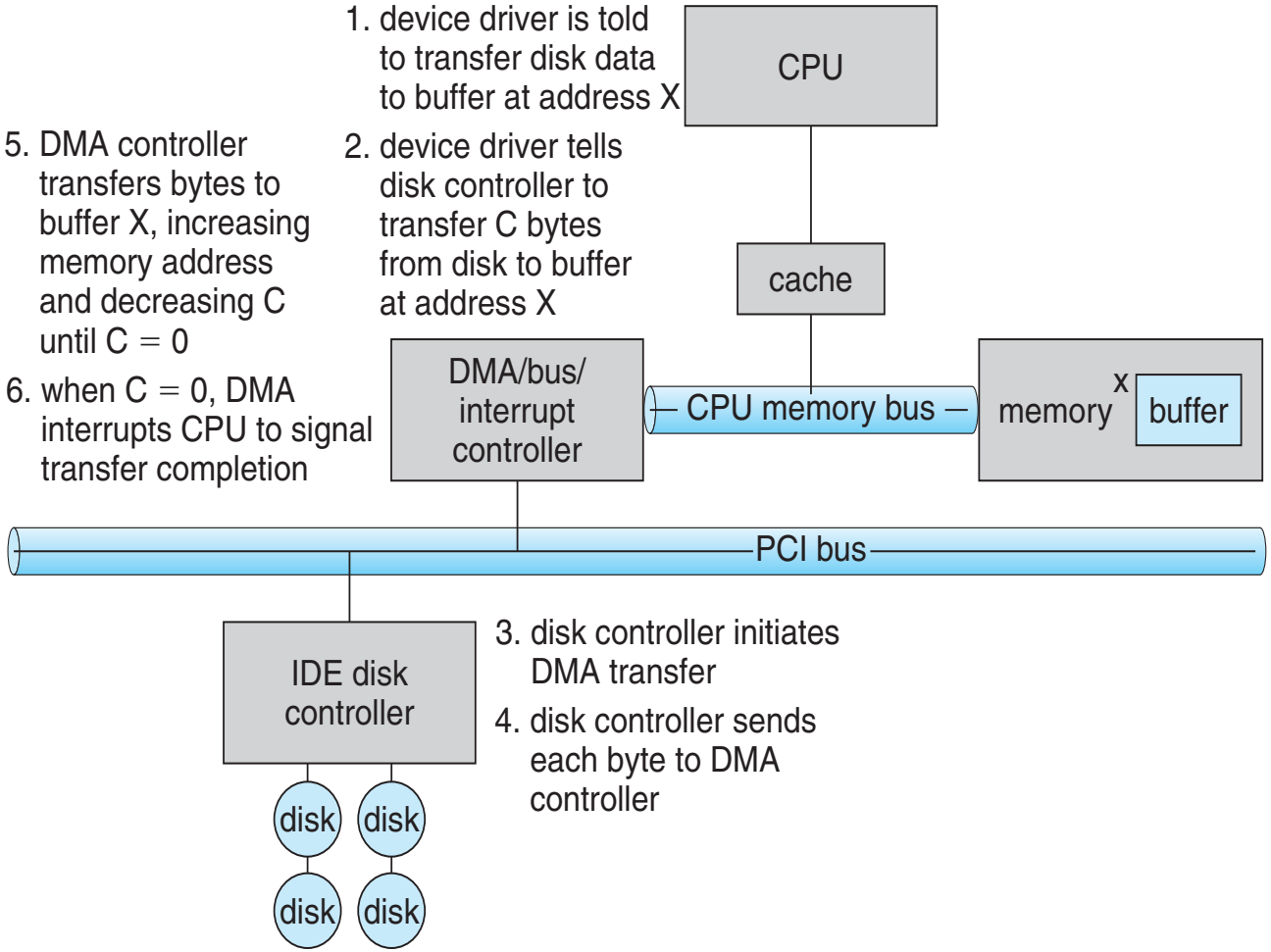


# Direct Memory Access (DMA) (1)

- Controller (e.g. disk controller) is able to write from/to arbitrary addresses of the main memory without intervention of the CPU.
  - CPU just configures DMA controller once (using PIO) for each transfer:
    1. command to be executed by controller (e.g. sector to be read from disk & number of consecutive sectors to be read),
    2. start of memory address to read from/write to via DMA.
  - DMA controller raises interrupt when DMA transfer is finished.
    - CPU can then, e.g., issue another command.
    - Each DMA transfer comprises exactly one consecutive memory area and controller command.



# Direct Memory Access (DMA) (2) Detailed Steps



# Direct Memory Access (DMA) (3) Discussion

- Advantage: Less CPU intensive. CPU can work while controller accesses data in RAM.
  - Ideal for virtual memory using paging: While DMA controller is loading pages, CPU may still execute those processes that are ready and in physical memory.
    - However, frames that are currently subject of DMA transfer must not be chosen as victim for replacement while DMA transfer takes place!
  - Still, DMA transfer slows down access of CPU to RAM ("memory cycle stealing" by DMA controller to access RAM in parallel to CPU just like shared-memory approach of memory-mapped I/O).
    - But again: no slow down if data needed by CPU is in CPU cache.
- All major device controllers that transfer huge amounts of data are nowadays DMA-enabled.
  - E.g. hard disk/SSD controller, USB controller.

# 13.7 Summary

- DMA attachment of mass storage devices allows CPU to compute in parallel to I/O.
- OS provides a unified interface for accessing I/O devices.
- Layered approach allows to exchange low-level device drivers while keeping higher-level device-independent drivers and file system implementations.
- Kernel I/O subsystem may help to reduce I/O bottleneck.