Lecture 9

# External Memory I/O

Memory hierarchy

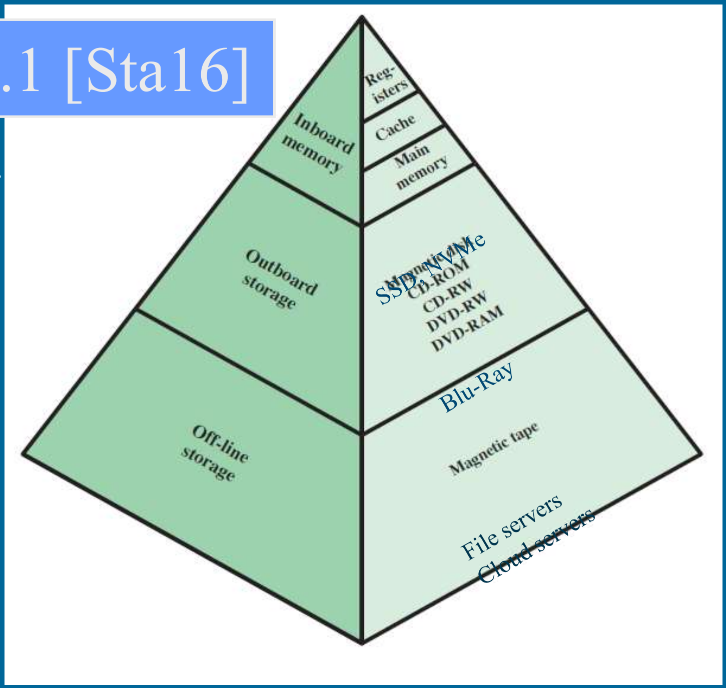
File system

Hard disk (HDD)

SSD, NVMe

I/O implementation with device driver and device controller

Fig 4.1 [Sta16]

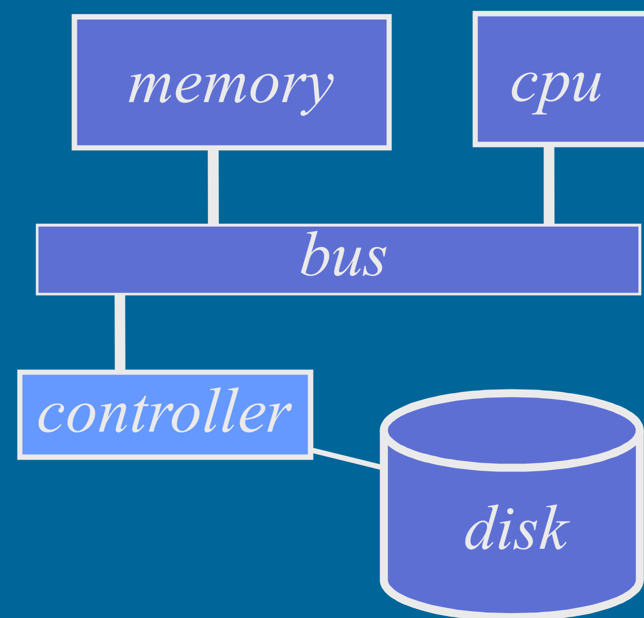



# Memory hierarchy

- External memory (HDD, SSD) is much cheaper per byte
- External memory is much slower than main memory
- Time/space optimization
  - Large data set must be (usually) kept in external memory because of cost
  - Small data set must be kept in main memory for speed
- All referenced data at execution time must be in main memory or even closer to processor
  - Processor cannot wait very long time for referenced data
  - Main memory is (somewhat) ok, external memory is too far
    - Cache makes main memory feel faster than it is

# Virtual Memory

- Part of memory hierarchy
- Answer to problem
  – How to implement memory, that is ”as large” as disk (HDD, SSD, NVMe) and ”as fast” as main memory?
- Two level solution
  – Main memory has all memory areas ”currently needed”
  – Disk has all data
  – Copy when needed
    - Page fault interrupt (referenced data is not in main memory)



# Virtual Memory Implementation

- Implementation methods
  - Paging, multi-level paging
  - segmentation, segmented paging
- Most of implementation is in OS software
  - Address translation: virtual address → main mem address
    - Many machine instructions and memory references…
  - Copying data: main memory ↔ disk
  - Memory management, which memory areas in use
- Hardware assistance
  - In MMU (memory management unit)
  - Translation Lookaside Buffer (TLB) has most recent address translations (one type of cache)
    - Virtual address to main memory address translation is fast, if it is found from TLB

More info? → Operating Systems

# File System

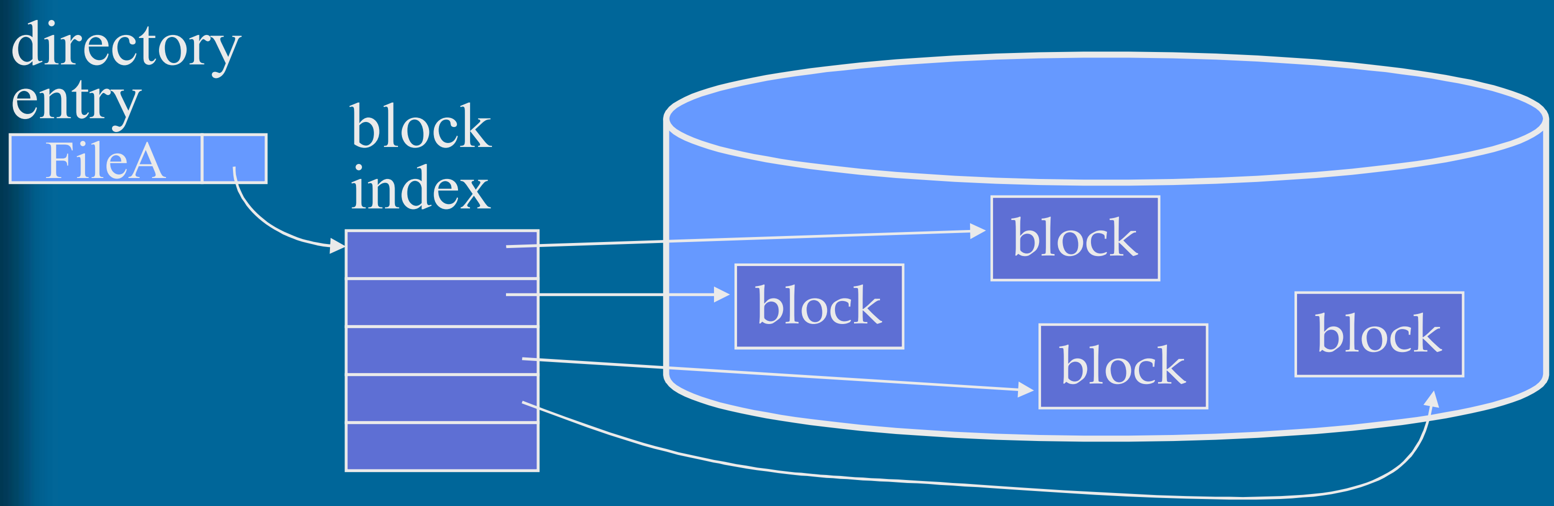
read
write
execute

- Part of OS, manages all files
- Checks for access rights when file opened
- Changes textual file names to physical addresses
- Keeps (OS) data structures, from which you can see which part of which file each process is accessing
- File systems reads and writes files in larger blocks (e.g., 2-8 KB or 2-8 MB)
  – User level processes may have byte access to files, and they do not need to know exact structure of files (OS device driver takes care of it)

# Storing Files on Disk

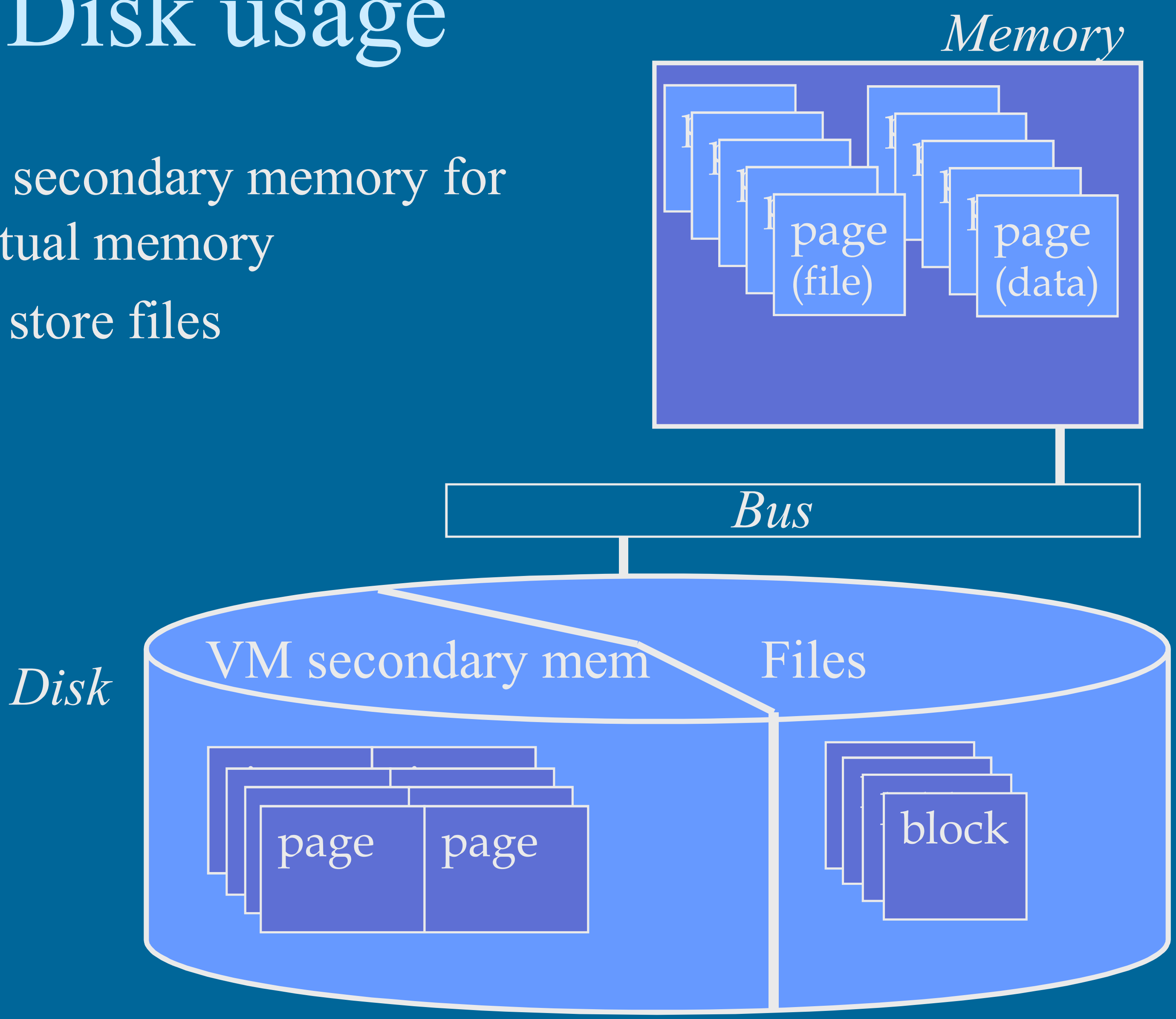
- File consists of many blocks
  - block = 1 or more disk sector
- File's directory entry
  - All blocks for each file
  - Blocks are read in given order

Smallest data segment in disk to read/write



# Disk usage

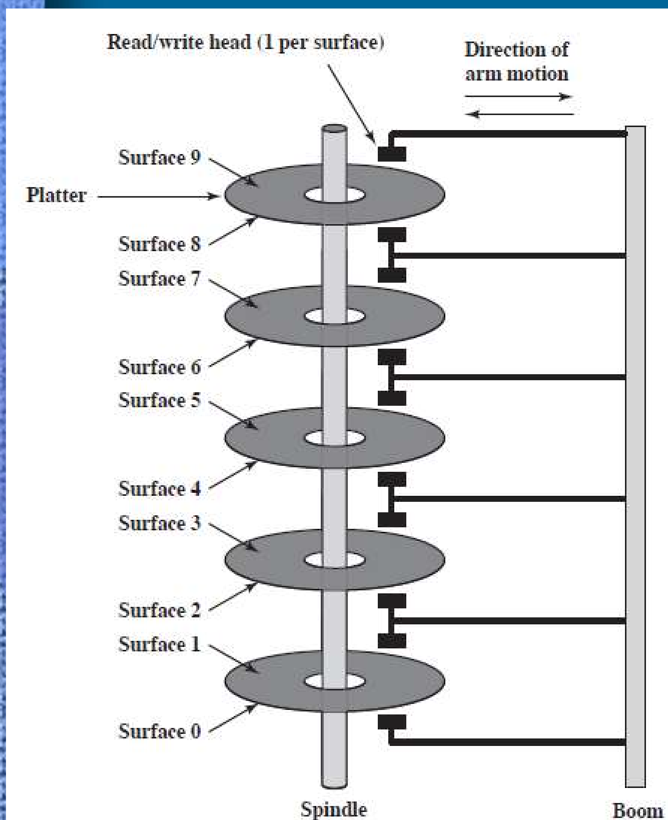
- As secondary memory for virtual memory
- To store files

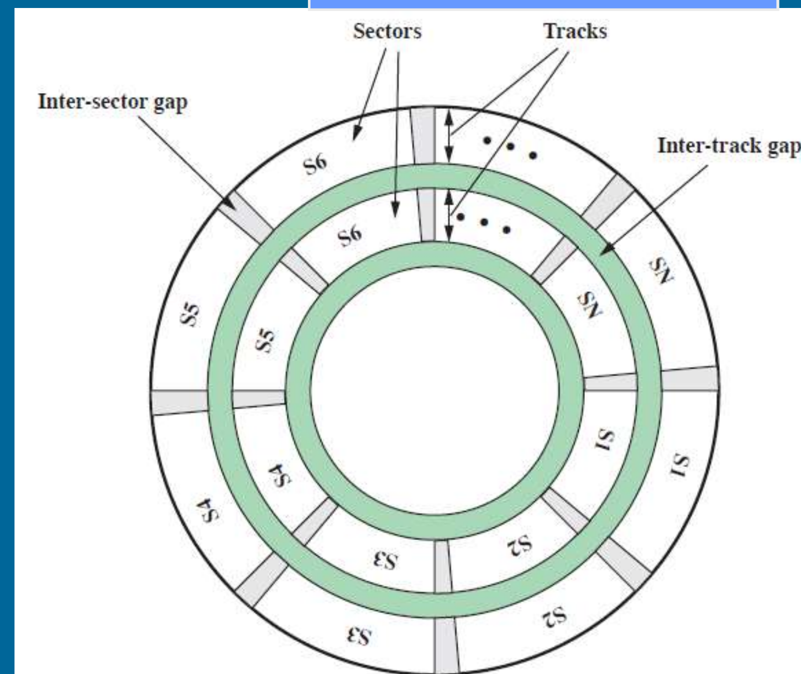


# Hard Disk (HDD) Access Time

- Block address: surface + track + sector
  - Device driver finds from file system OS-tables

Fig 6.2 [Sta16]




- Access time
  - Search + rotation + data transfer

Discuss

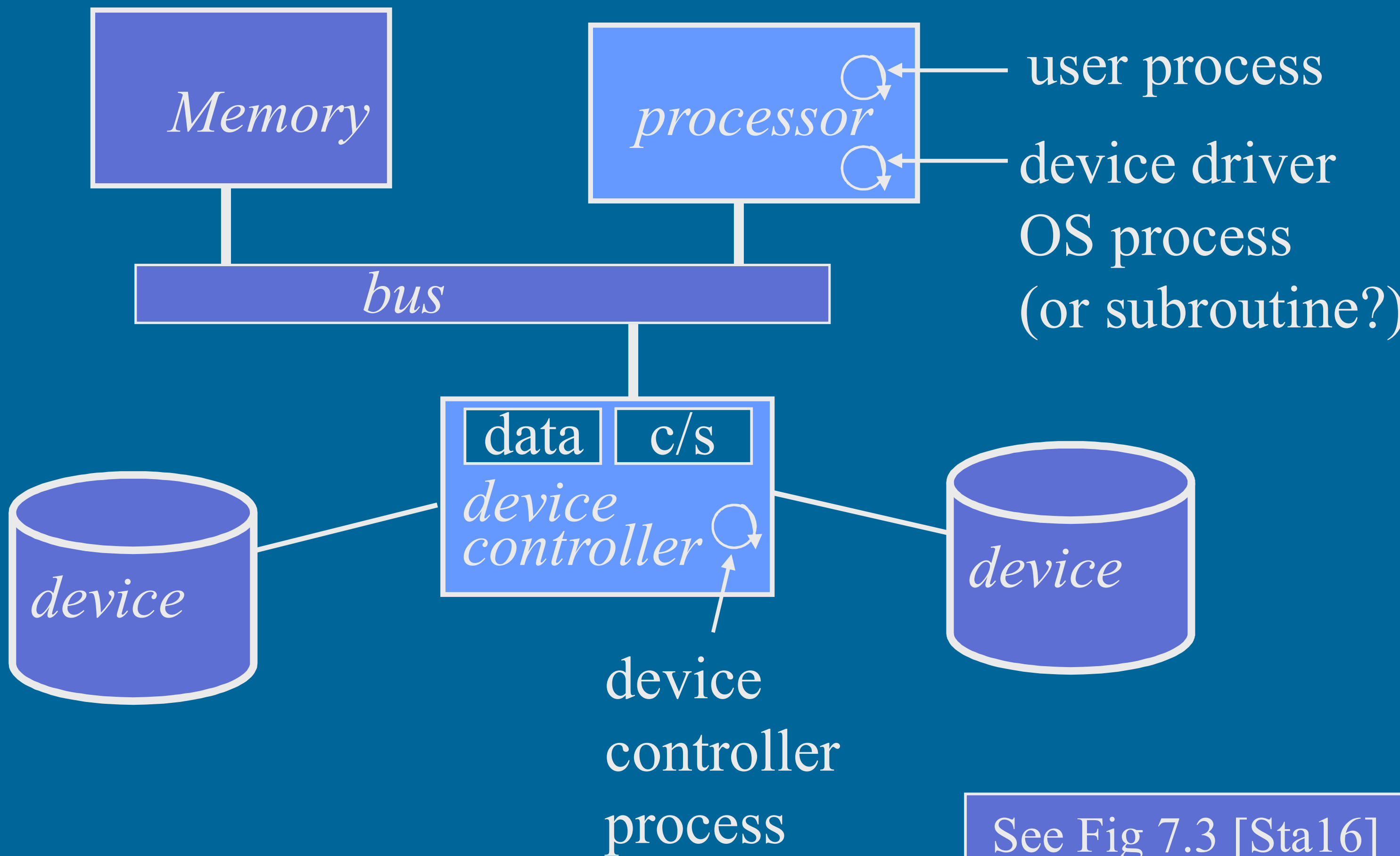
# SSD and NVMe

Solid State Disk

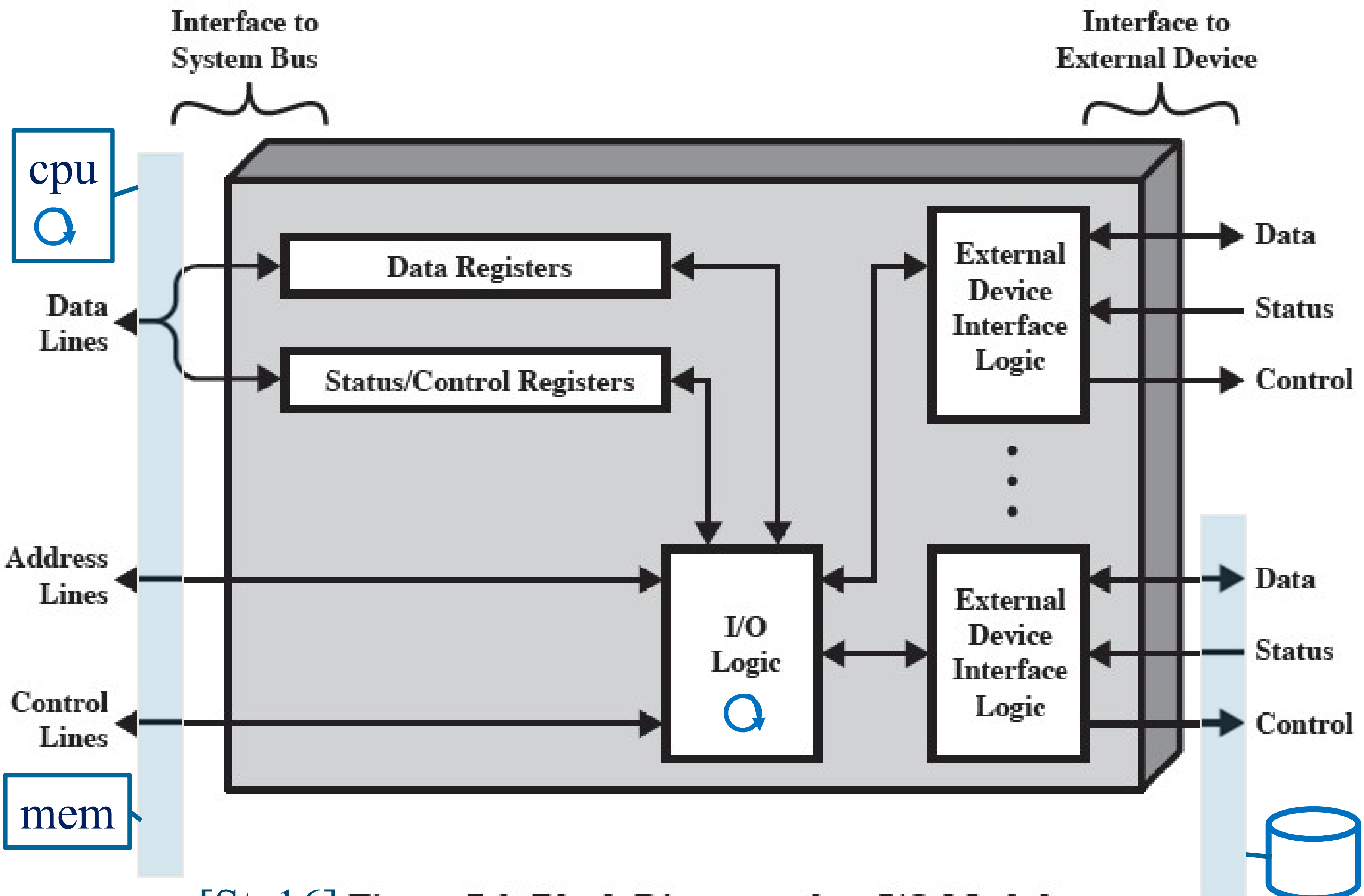
Non-Volatile Memory Express (NVMe, NVM Express)

- SSD
  - Some kind of flash circuit (e.g.)
  - Usually: OS sees it as another hard disk
- NVMe: OS sees it as flash memory
  - Faster, OS can utilize concurrency within implementation
- Blocks and pages
  - Files (e.g.) as 4 KB pages
  - Read/write (e.g.) as 512 KB blocks
    - Whole block must be written at a time
    - Writing may be to another new block
    - Each hw-block could have limit on nr of writes(e.g., 100K)
      - Spare blocks on circuit?

# Device Controller (I/O Module)




See Fig 7.3 [Sta16]




[Sta16] **Figure 7.3 Block Diagram of an I/O Module**

# I/O Implementation with I/O Instructions: Reference Device Controller Registers with Specific Machine Instructions

I/O konekäskyt

- I/O operation recognized from the opcode
  - I/O devices have their own machine instructions
  - I/O instructions have their own address space, they do not reference main memory
- I/O instructions have device controller id and device register nr (own I/O address space)
- Hard to extend use to new devices that may have different device registers (device memory referenced via bus)
- Machine instructions cannot be modified (in general)

x86: IN, OUT
INS, OUTS

Ttk-91:
IN, OUT

# I/O Implementation with Memory-Mapped I/O: Reference Device Controller Registers with Ordinary Read/Write Memory Instructions

muistiinkuvattu I/O

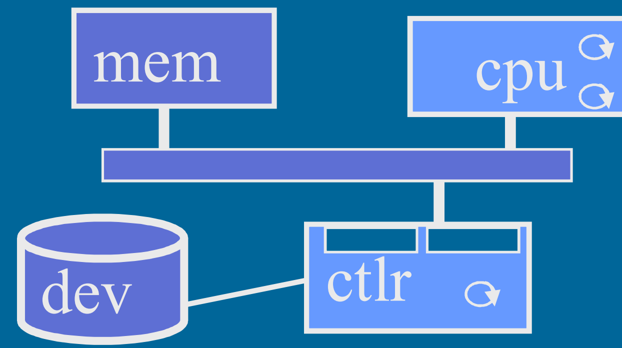
- I/O-operation is recognized from memory address that was used (and not from opcode)
- Device driver reads/writes device registers (data, status, control) on device controller with ordinary memory read/write operations
  - No need for separate I/O instructions
  - Device registers in device controller are similar memory as "normal" main memory
  - 1st bits in memory address determine, whether normal memory or device registers in some device controller is referenced
    - Part (half?) of memory space is reserved to I/O devices

```
load  R1, =1   ; read
store R1, @ptrCtr
```

```
ptrCtr   DC  0x80000001 ; control register address
ptrStat  DC  0x80000002 ; status register address
ptrData  DC  0x80000003 ; data register address
```

# I/O types

suora I/O




- Programmed I/O  Direct I/O
  - Device driver is active all the time
  - Device driver waits in busy-wait loop for the device controller
  - Data travels via CPU register, one word at a time
- Interrupt driven I/O  Indirect I/O
  - Device driver can do I/O interrupts
  - Device driver waits in suspended state for the device controller
  - Data travels via CPU register, one word at a time

epäsuora I/O
keskeyttävä I/O

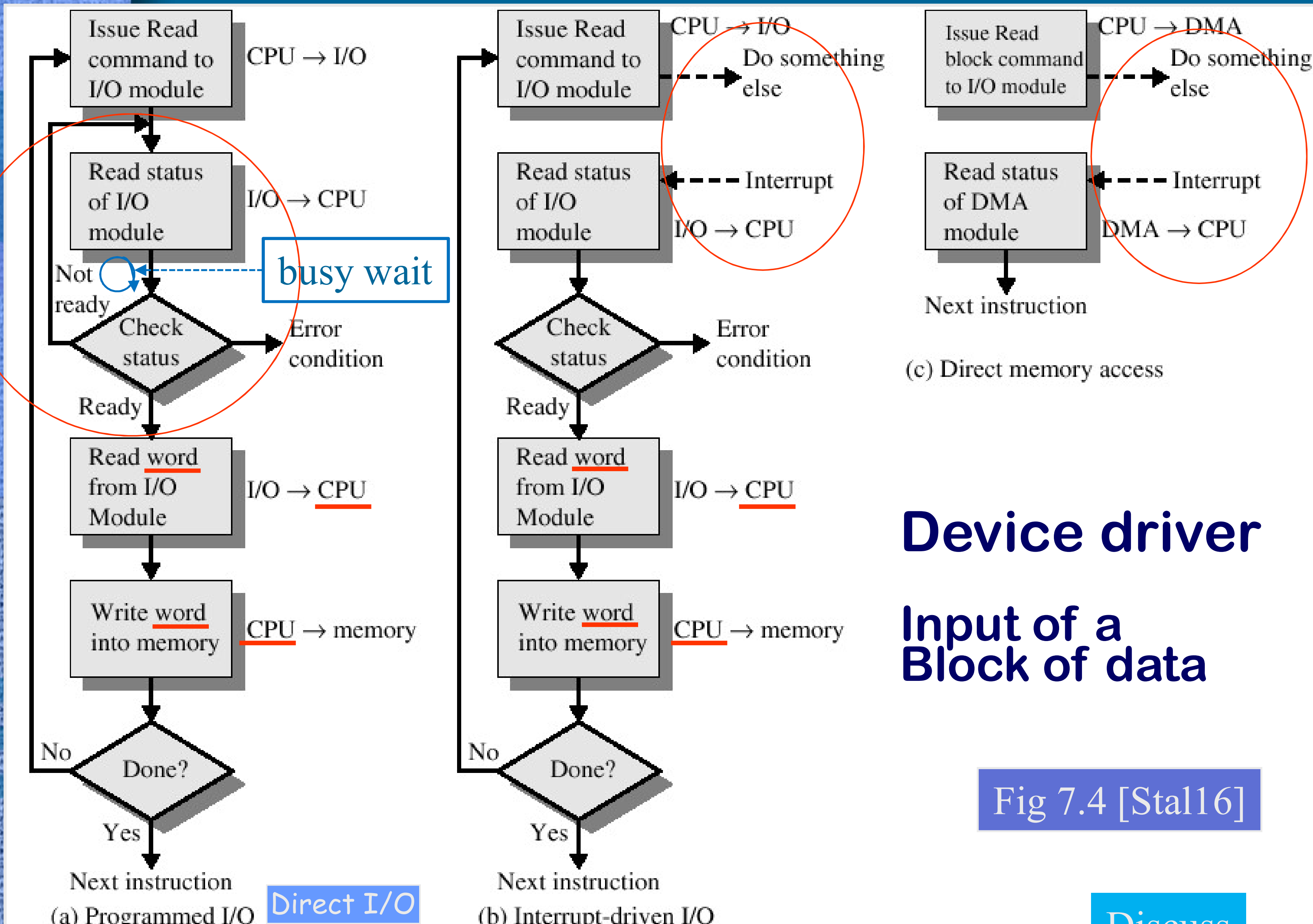
- DMA I/O  Direct Memory Access I/O
  - Device driver can also reference main memory
  - Device driver waits in suspended state for the device controller
  - Data travels via memory bus only once on its way from memory to/from device controller data register
  - Tasks given to device controller are larger

DMA I/O

Programmed I/O

Interrupt driven I/O

DMA I/O




# Device driver

## Input of a Block of data

Fig 7.4 [Stal16]

Discuss

# Example: Printer Device Driver for ttk-91

- You can print integer  numbers one at a time
- Memory mapped I/O, direct I/O
- Device port
  - Control register      memory location 1048576 = 0x80000
  - State register      memory location 1048577 = 0x80001
  - Data register      memory location 1048578 = 0x80002
- Device driver Print operates in privileged mode
  - Can reference registers in device port
- Call:

```
PUSH  SP, =0        ; space for return value
PUSH  SP, X         ; parameter to print
SVC   SP, =Print    ; returns Success/Failure
POP   SP, R1
JNZER R1, TakeCareOfTrouble
```

# Device Driver Print Implementation (12)

## Solution with no timeout

```
ptrCtr    DC  1048576 ; control register address
ptrStat   DC  1048577 ; status register address
ptrData   DC  1048578 ; data register address
retVal    EQU  -3
parData   EQU  -2
```

Assume: SVC & IRET implem. ”same way” as CALL and EXIT


```
Print    PUSHR  SP                    ;save regs
         LOAD   R1, parData(FP)
         STORE  R1, @ptrData ; data to print
         ------------------------------------------
         LOAD   R1, =0
         STORE  R1, @ptrStat  ; init (clear) state register
         ------------------------------------------
         LOAD   R1, =1
         STORE  R1, @ptrCtr  ; give command to print
         ------------------------------------------

Wait     LOAD   R1, @PtrStat ; check state register
         JNZER  R1, Done
         JUMP   Wait               ; wait until I/O done
         ------------------------------------------

Done     LOAD   R1, =0             ; return ”Success”
         ------------------------------------------
         STORE  R1, retVal(FP)
         POPR   SP                 ; recover regs
         IRET   SP, =1
```

See: driver.k91

Discuss

http://www.cs.helsinki.fi/group/nodes/kurssit/tito/esimerkit/driver.k91

# -- End--

- Ferrite ring (core) technology
  - 1952, Jay Forrester & Bob Everett, MIT (Whirlwind)
  - Data sustained without power
  - Not disturbed by radiation (space and military technology)
  - 1955, conquers main memory markets from Williams Tube
  - Used still in 1970's
  - Now only the name (”core”) remains and is still in use