# INF2270 — Spring 2010

**Philipp Häfliger**

## Lecture 7: More on Cache, Virtual Memory, I/O

UNIVERSITETET I OSLO

# content

Cache (Continued)

Virtual Memory

Input/Output (I/O)



UNIVERSITETET I OSLO

# content

Cache (Continued)

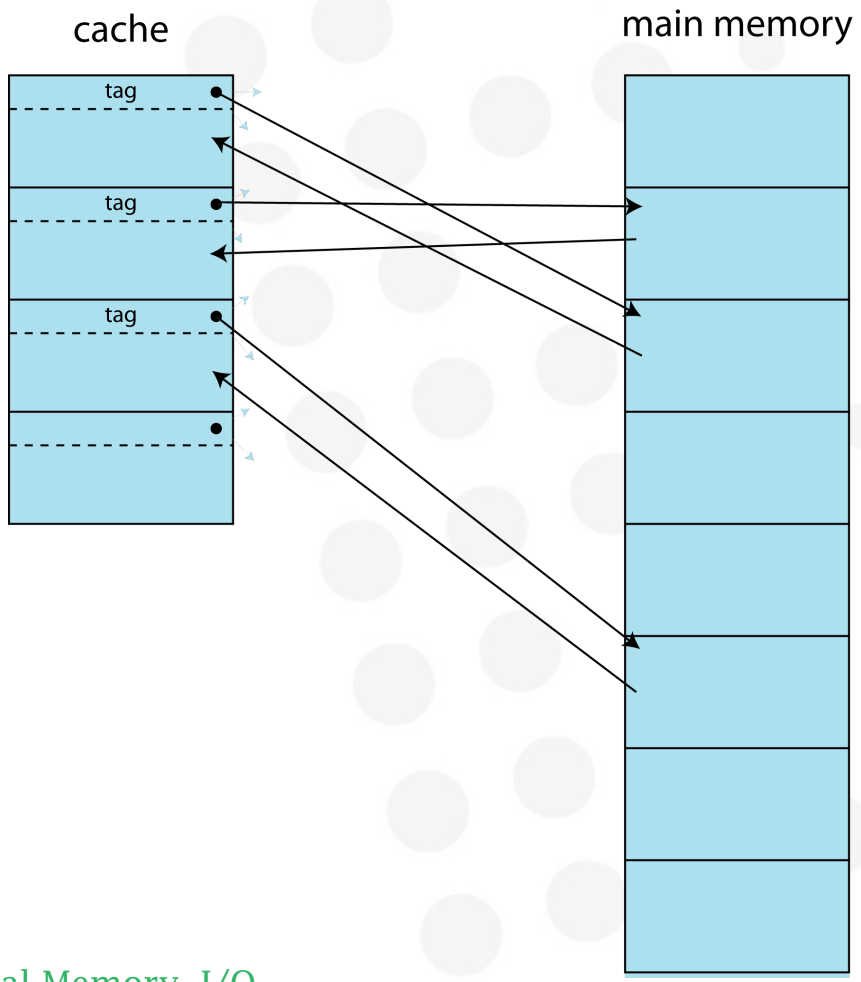
Virtual Memory

Input/Output (I/O)



UNIVERSITETET I OSLO

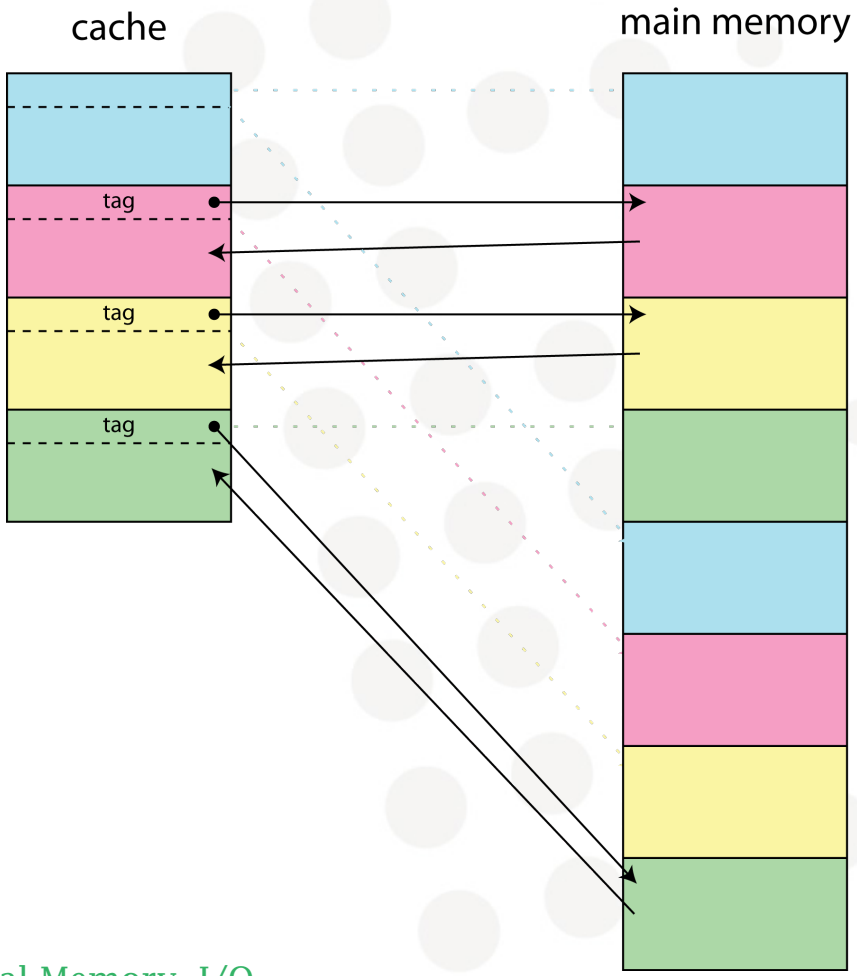
# Mapping Strategy: Associative (rep.)






UNIVERSITETET I OSLO

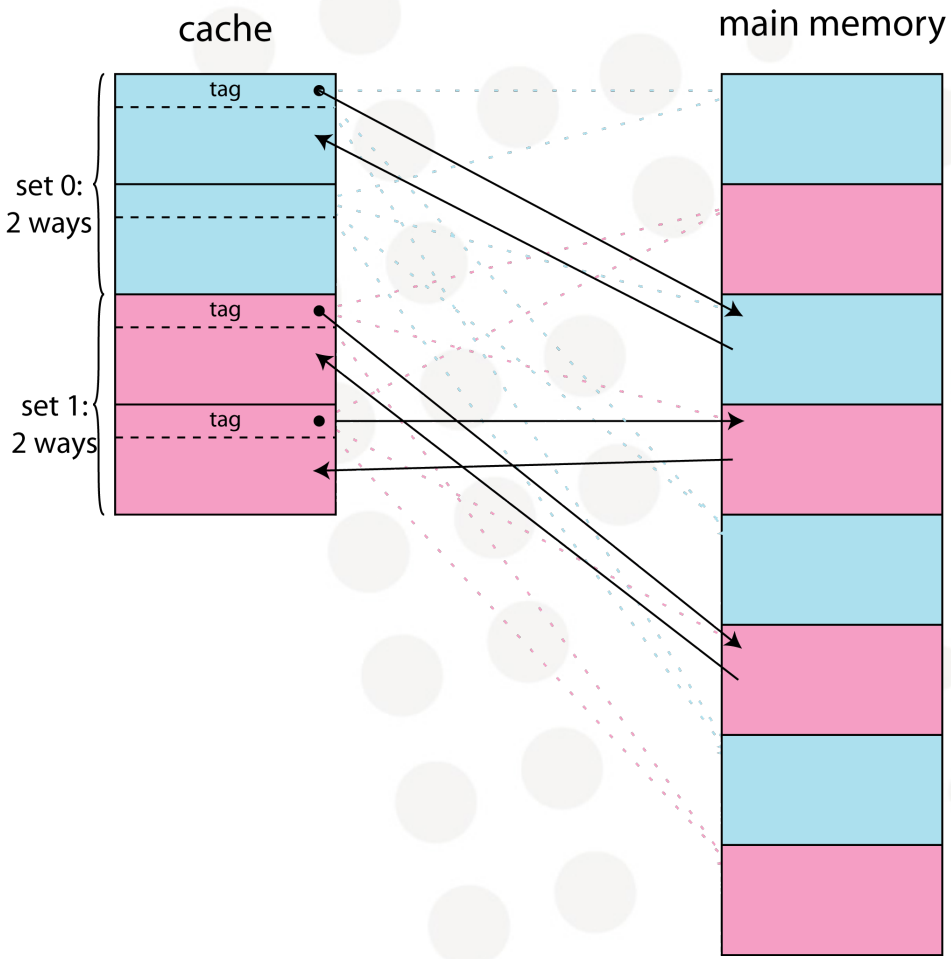
# Mapping Strategy: Direct (rep.)






UNIVERSITETET I OSLO

# Mapping Strategy: Set Associative (rep.)






UNIVERSITETET
I OSLO

# Replacement Strategy (1/3)

As a consequence of a cache miss a new block needs to be loaded into the cache. In associative and set associative cache it might happen that all available slots are already occupied and a choice needs to be made, which block in the cache that will be replaced by the new block. There are different strategies for this choice, most prominently:

- first in first out (FIFO)
- least recently used (LRU)
- random
- hybrid solutions

(Note that in direct mapping cache there is no choice as to which block to replace).



UNIVERSITETET I OSLO

# Replacement Strategy (2/3)

LRU seems intuitively quite reasonable but requires a good deal of administrative processing (causing delay): Usually a 'used' flag is set per block when it is accessed. This flag is reset in fixed intervals and a time tag is updated for all blocks that have been used. These time tags have either to be searched before replacing a block or a queue can be maintained and updated whenever the time tags are updated.

FIFO is simpler. The cache blocks are simply organized in a queue (ring buffer)



UNIVERSITETET I OSLO

# Replacement Strategy (3/3)

random Both LRU and FIFO are in trouble if a program works several times sequentially through a portion of memory that is bigger than the cache: the block that is cast out will very soon be needed again. A random choice will do better here

hybrid solutions, e.g. using FIFO within a set of blocks that is randomly chosen are also used in an attempt to combine the positive properties of the approaches



UNIVERSITETET I OSLO

# Cache Architectures

- look-through
- look-aside

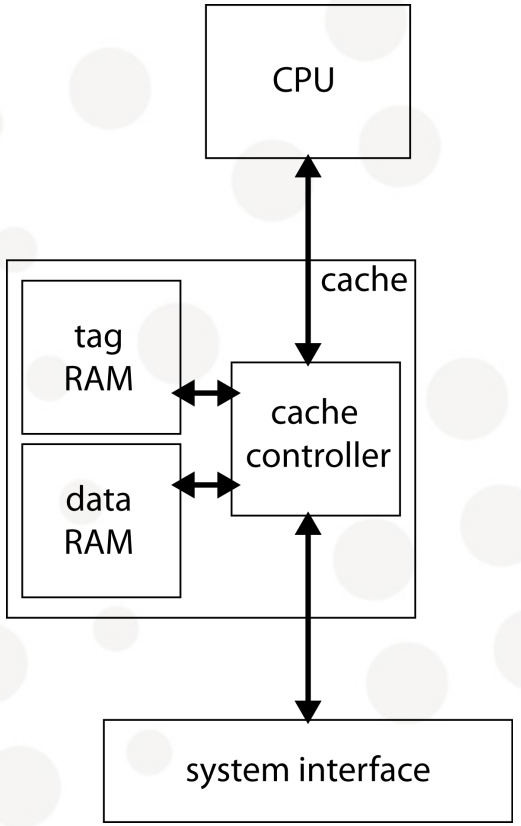


UNIVERSITETET I OSLO

# Look-Through Architecture

The cache is physically placed between CPU and memory (system interface)

- memory access is initiated *after* a cache miss is determined (i.e. with a delay)
- only if a cache miss is determined, is a memory access initiated
- CPU can use cache while memory is in use by other units



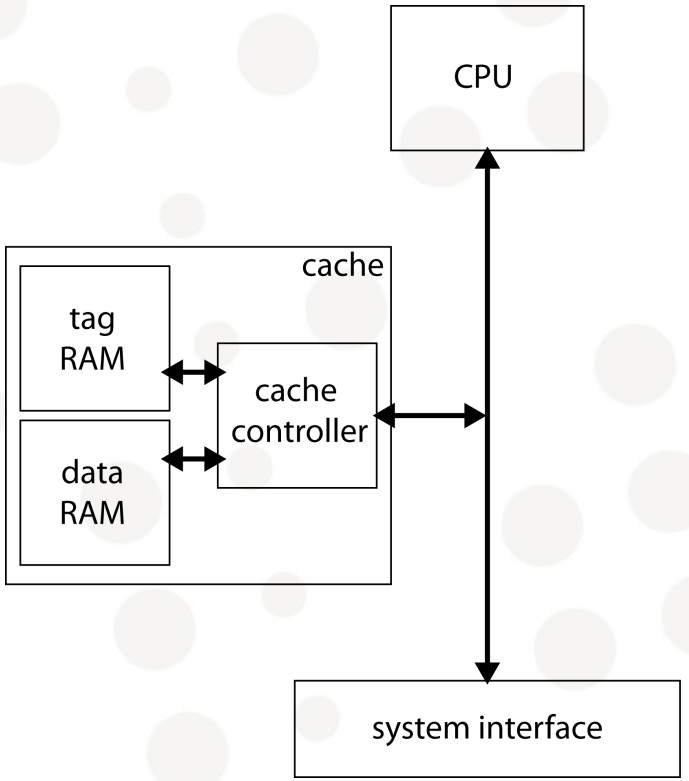



UNIVERSITETET I OSLO

# Look-Aside Architecture

The cache shares the bus between CPU and memory (system interface)

- memory access is initiated *before* a cache miss is determined (i.e. with no delay)
- with a miss the cache just listens in 'snarfs' the data
- only if a cache hit is determined, does the cache takes over
- CPU cannot use cache while other units access memory






UNIVERSITETET I OSLO

# Cache Summary

- Mapping Strategy
  - associative
  - direct
  - set associative
- Write Strategy
  - write through
  - write back
- Replacement Strategy
  - least recently used (LRU)
  - FIFO
  - random
- Architecture
  - look aside
  - look through



UNIVERSITETET I OSLO

# content





UNIVERSITETET I OSLO

# Virtual Memory

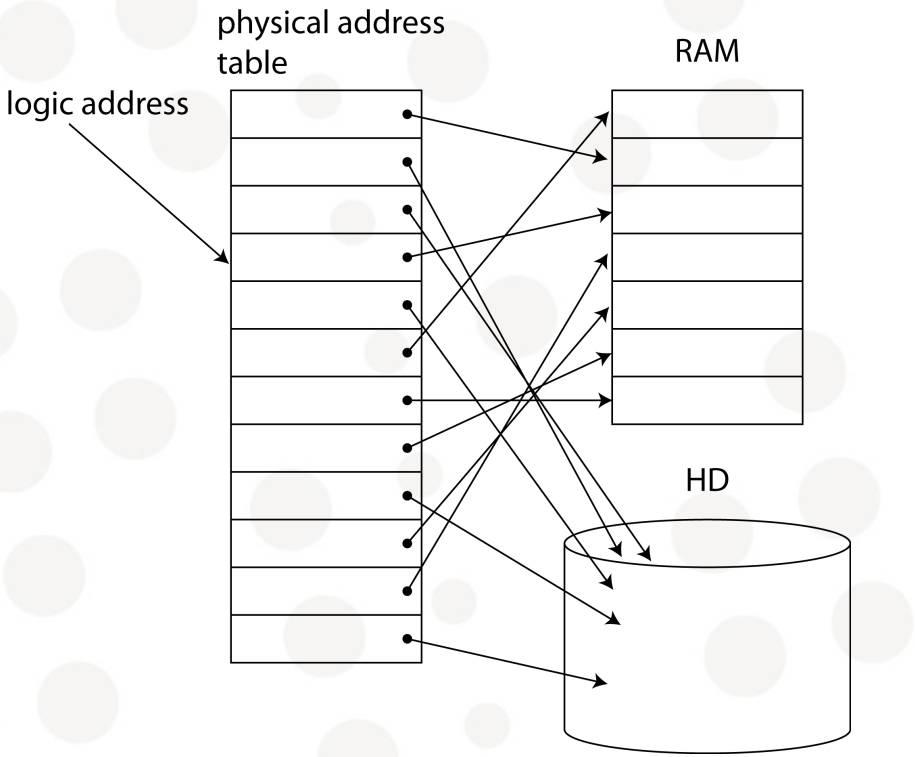
*Virtual* memory extends the amount of main memory as seen by programs/processes beyond the capacity of the *physical* memory. Additional space on the hard drive (swap space) is used to store a part of the virtual memory that is, at present, not in use. The task of the virtual memory controller is quite similar to a cache controller: it distributes data between a slow and fast storage medium. A virtual memory controller may simply be part of the operating system rather than a hardware component, but most often there is a HW memory management unit (MMU) using a translation look-aside buffer (TLB) that supports virtual memory.



UNIVERSITETET I OSLO

# Virtual Memory Principle

The principle of virtual memory is that each *logic* address is translated into a *physical* address, either in the main memory or on the hard drive. processes running on the CPU only see the logic addresses and a coherent the virtual memory.

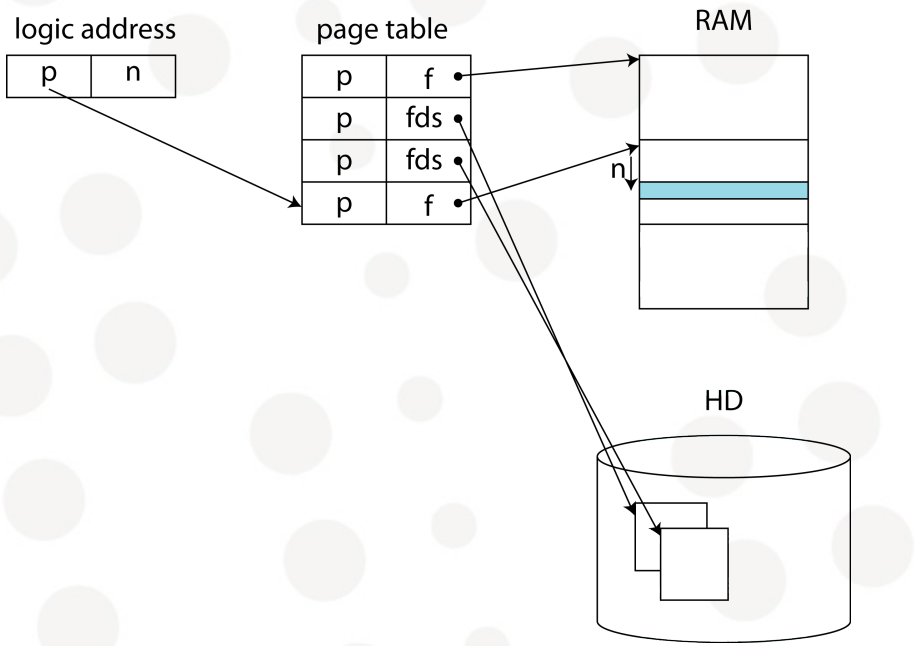





UNIVERSITETET I OSLO

# Virtual Memory Paging

A pointer for each individual logic address would require as much space as the entire virtual memory. Thus, a translation table is mapping memory blocks (called *pages* (fixed size) and *segment* (variable size)). A logic address can, thus, be divided into a page number and a page offset. A location in memory that holds a page is called page *frame*.






UNIVERSITETET I OSLO

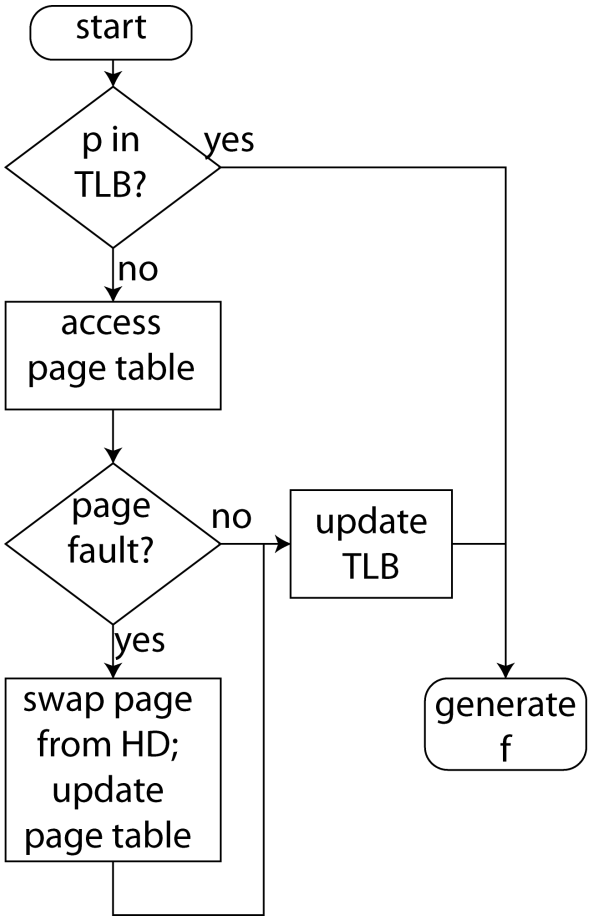
## Translation Look-Aside Buffer (TLB)

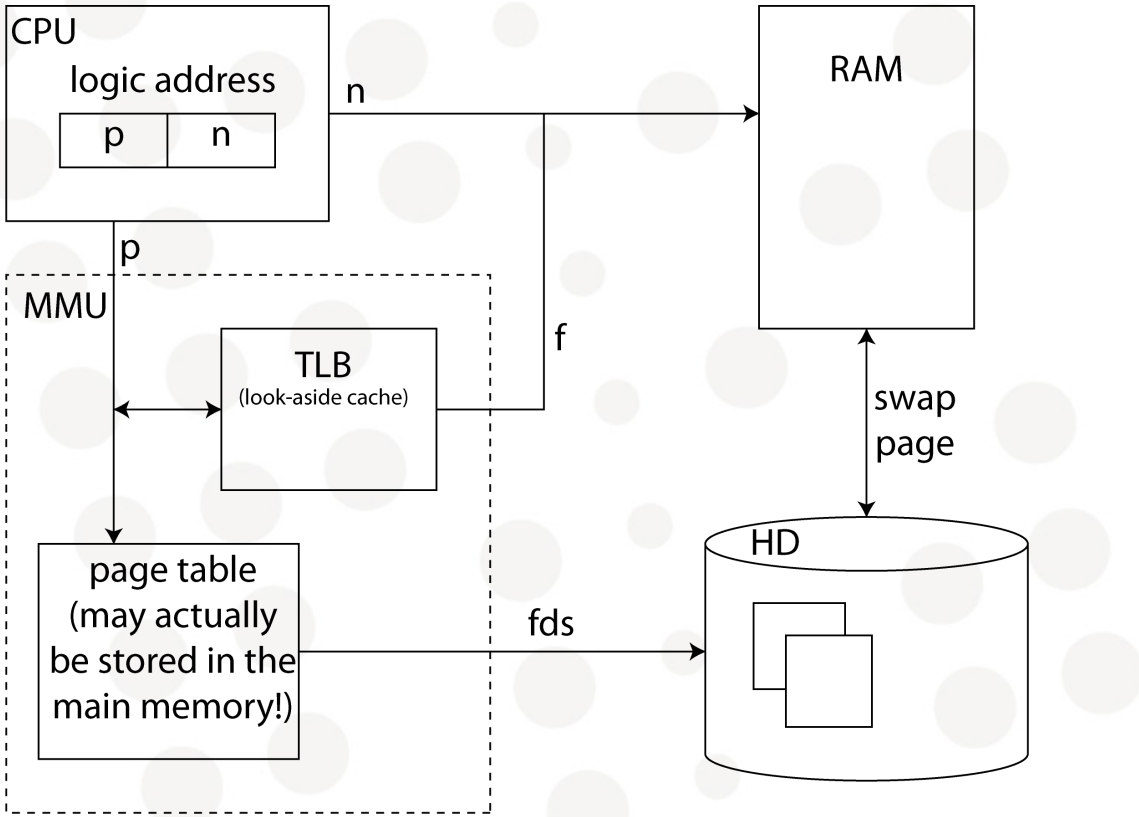
A translation look-aside buffer is a cache for the page table, accelerating the translation of logic to physical address by the MMU.



UNIVERSITETET I OSLO

# MMU Flow Chart and Block Diagram






UNIVERSITETET I OSLO

# Memory Hierarchy Summary

| | | |
|---|---|---|
| registers | ~ 1ns | ~ 100B |
| L1 (on CPU) cache | $\sim\geq$ 1ns | ~ 10kB |
| L2,L3 (off CPU) cache | ~ 2ns-10ns | ~ 1MB |
| main memory (DRAM) | ~ 20ns-100ns | ~ 1GB |
| SSD/flash | ~ 100ns-1$\mu$s | ~ 10-100GB |
| hard disc | ~ 1ms | ~ 0.1-1TB |



UNIVERSITETET I OSLO

# content

Cache (Continued)

Virtual Memory

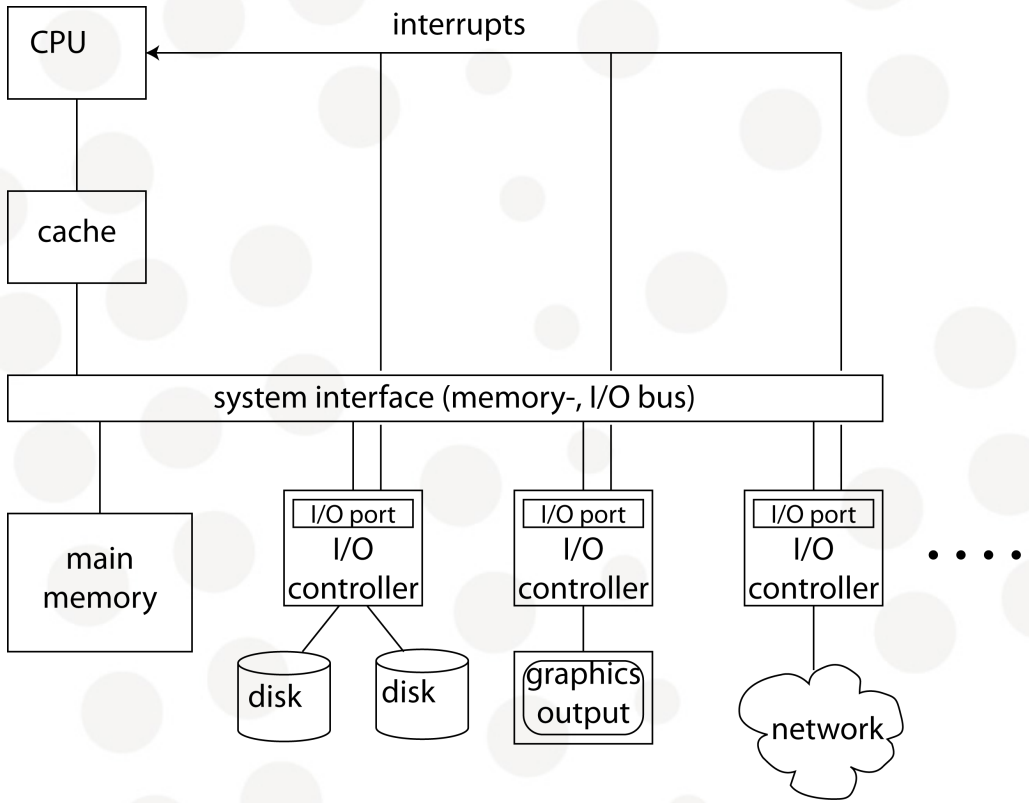
Input/Output (I/O)



UNIVERSITETET I OSLO

# I/O Block Diagram

A computer is connected to various devices transferring data to and from the main memory. This is referred to as Input/output (I/O). Examples: Keyboard, Graphics, Mouse, Network (Ethernet, Bluetooth ...), USB, Firewire, PCI, PCI-express, SATA ...

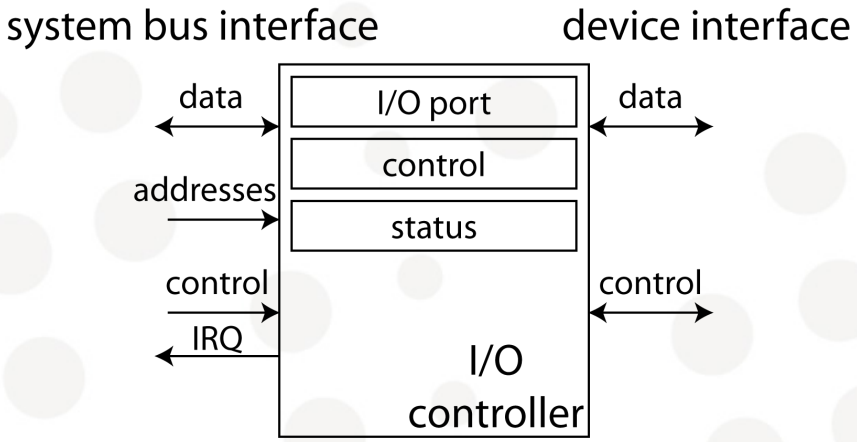





UNIVERSITETET I OSLO

# I/O Controller Principle

An I/O controller translates and synchronizes a peripheral device *protocol* (communication language) for the system bus. It normally has at least one data buffer referred to as *I/O port*, a control register that allows some SW configuration and a status register with information for the CPU.






UNIVERSITETET I OSLO

# I/O Addressing (1/2)

Memory mapped I/O is to access I/O ports and I/O control- and status registers (each with its own address) with the same functions as the memory. Thus, in older systems, the system interface might simply have been a single shared I/O and memory bus. A disadvantage is that the use of memory addresses may interfere with the expansion of the main memory.



UNIVERSITETET I OSLO

# I/O Addressing (2/2)

Isolated I/O (as in the 80x86 family) means that separate instructions accessing an I/O specific address space are used for I/O access. An advantage can be that these functions can be made *privileged*, i.e. only available in certain modes of operation, e.g. only to the operating system.



UNIVERSITETET I OSLO

# Modes of Transfer(1/3)

**Programmed/Polled:** The processor is in full control of all aspects of the transfer. It *polls* the I/O status register in a loop to check if the controller has data to be collected from the port or is ready to receive data to the port. Polling uses up some CPU time and prevents the CPU from being used for other purposes while waiting for I/O.



UNIVERSITETET I OSLO

# Modes of Transfer(2/3)

**Interrupt Driven:** The I/O controller signals with a dedicated 1bit data line (*interrupt request* (IRQ)) to the CPU that it needs servicing. The CPU is free to run other processes while waiting for I/O. If the interrupt is not *masked* in the corresponding CPU status register, the current instruction cycle is completed, the processor status is saved (PC and flags pushed onto stack) and the PC is loaded with the starting address of an *interrupt handler*. the start address is found, either at a fixed memory location specific to the interrupt priority (*autovectored*) or stored in the controller and received by the CPU after having sent an *interrupt acknowledge* control signal to the device (*vectored*)



UNIVERSITETET I OSLO

# Modes of Transfer(3/3)

**Direct Memory Access (DMA):** The processor is not involved, but the transfer is negotiated directly with the memory, avoiding copying to CPU registers first and the subroutine call to the interrupt handler. DMA is used for maximum speed usually by devices that write whole blocks of data to memory (e.g. disk controllers). The CPU often requests the transfer but then relinquishes control of the system bus to the I/O controller, which only at the completion of the block transfer notifies the CPU with an interrupt.
(DMA poses another challenge to the cache as data can now become *stale*, i.e. invalid in the cache)



UNIVERSITETET I OSLO