INF2270 — Spring 2011

Lecture 6: Memory Hierarchy

UNIVERSITETET I OSLO

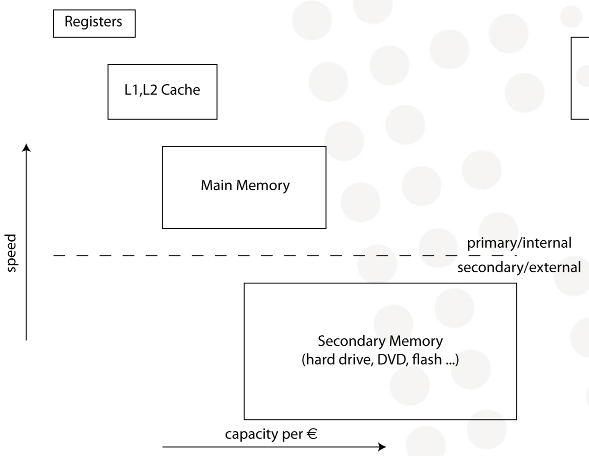
content

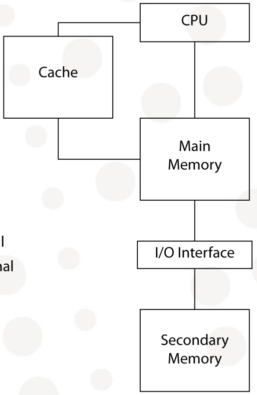
Cache

- Mapping Strategy
- Write Strategy
- Replacement Strategy
- Architecture

Virtual Memory

Memory Hierarchy





Memory Hierarchy access speeds

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

content

Cache

- Mapping Strategy
- Write Strategy
- Replacement Strategy
- Architecture

Virtual Memory

Cache

Cache is used to ameliorate the von Neumann memory access bottleneck. Cache refers to a small high speed RAM integrated into the CPU or close to the CPU. Access time to cache memory is considerably faster than to the main memory. Cache is small to reduce cost, but also because there is always a trade off between access speed and memory size. Thus, modern architectures include also several hierarchical levels of cache (L1, L2, L3 ...).

Locality of Code and Data

Cache uses the principle of *locality* of code and data of a program, i.e. that code/data that is used close in time is often also close in space (memory address). Thus, instead of only fetching a single word from the main memory, a whole block around that single word is fetched and stored in the cache. Any subsequent load or write instructions that fall within that block (a cache *hit*,) will not access the main memory but only the cache. If an access is attempted to a word that is not yet in the cache (a cache *miss*) a new block is fetched into the cache (paying a penalty of longer access time).

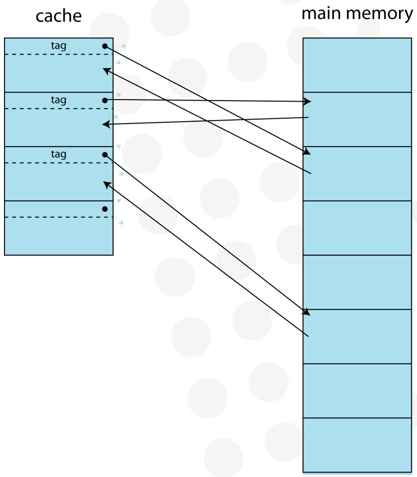
content

Checking for Hits

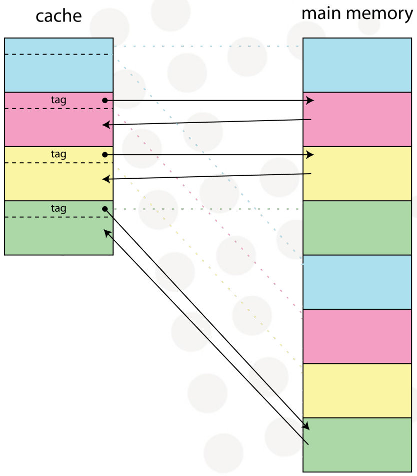
Checking for hits or misses quickly is a prerequisite for the usefulness of cache memory.

- associative cache
  Parallel search (extremely specialized HW) among memory block tags in the cache.
- direct mapped cache
  A hash-function assigns each memory block to only one cache slot, only one tag needs to be checked
- set-associative cache
  A combination of the previous two: each memory block is hashed to one block-set in the cache. Quick search for the tag needs only to be conducted within the set.

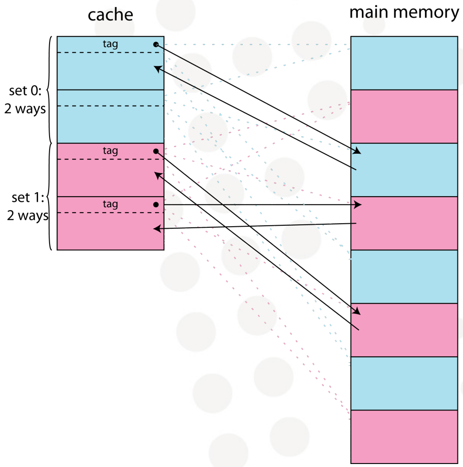
Associative Cache



Direct Mapping



Set Associative Cache



content

Cache

- Mapping Strategy
- Write Strategy
- Replacement Strategy
- Architecture

Virtual Memory

Cache Coherency

A write operation will lead to a temporary inconsistency between the content of the cache and the main memory. Several strategies are used in different designs to correct this inconsistency with varying delay. Major strategies:

write through	: a simple policy where each write to the cache is followed by a write to the main memory. Thus, the write operations do not really profit from the cache.
write back	: delayed write back where a block that has been written to in the cache is marked as *dirty*. Only when dirty blocks are reused for another memory block will they be written back into the main memory.

content

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).

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)

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

content

Cache

- Mapping Strategy
- Write Strategy
- Replacement Strategy
- Architecture

Virtual Memory

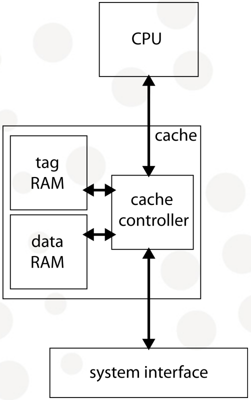
Cache Architectures

- look-through
- look-aside

Look-Through

The cache is physically placed between CPU and memory

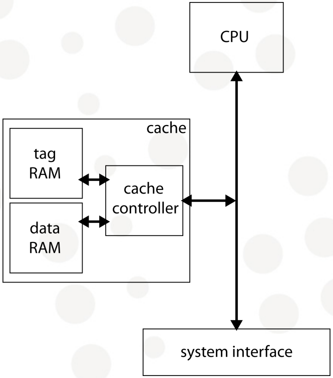
- memory access is initiated *after* a cache miss is determined (i.e. with a delay)
- only if a cache miss is determined, is a memory access initiated
- CPU can use cache while memory is in use by other units



Look-Aside

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 take over
- CPU cannot use cache while other units access memory



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

content

Cache

- Mapping Strategy
- Write Strategy
- Replacement Strategy
- Architecture

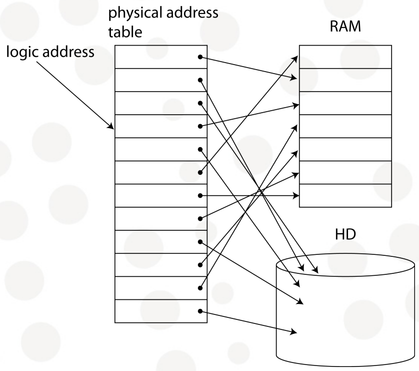
Virtual Memory

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.

Virtual Memory Principle

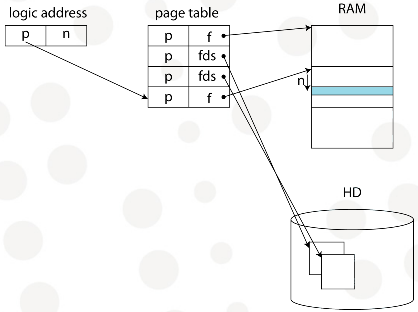
The principle of virtual memory is that each *logic* address is translated into a *physical* address, either in the main memory or on the hard drive. processes running on the CPU only see the logic addresses and a coherent the virtual memory.



UNIVERSITETET I OSLO

Virtual Memory Paging

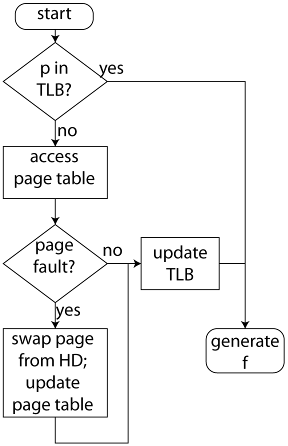
A pointer for each individual logic address would require as much space as the entire virtual memory. Thus, a translation table is mapping memory blocks (called *pages* (fixed size) and *segment* (variable size)). A logic address can, thus, be divided into a page number and a page offset. A location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

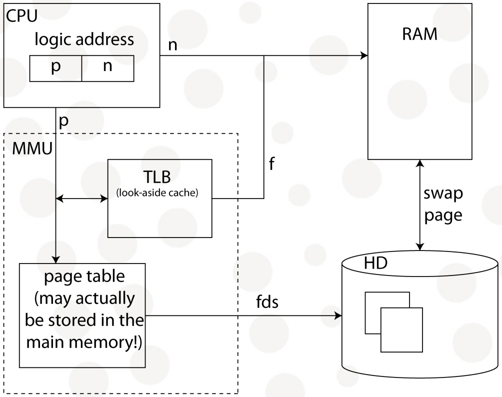


Translation Look-Aside Buffer (TLB)

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

MMU Flow Chart and Block Diagram


start
p in TLB?
yes
no
access page table
page fault?
no
update TLB
yes
swap page from HD; update page table
generate f


CPU
logic address
p
n
n
RAM
p
MMU
TLB
(look-aside cache)
f
swap page
HD
page table (may actually be stored in the main memory!)
fds