CSN08601 – Computer Systems

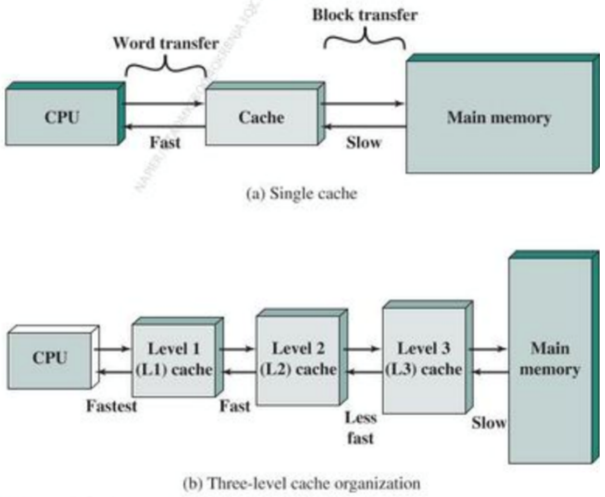
Cache

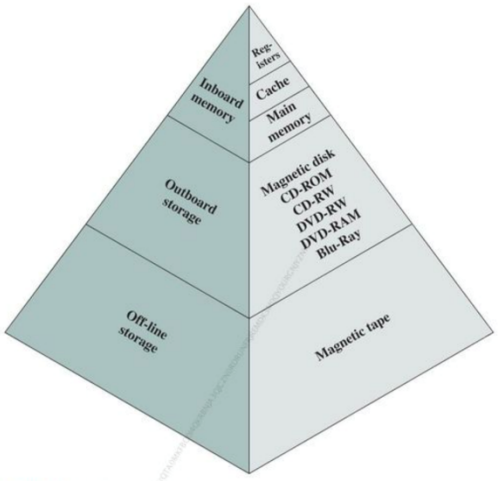
Cache Memory Principles

* It is designed to combine the memory access time of expensive, high speed memory with the large memory size of less expensive, lower speed memory
* In general: larger and slow main memory together with a smaller, faster cache memory

1- What happen when the processor attempt to read a word of memory?

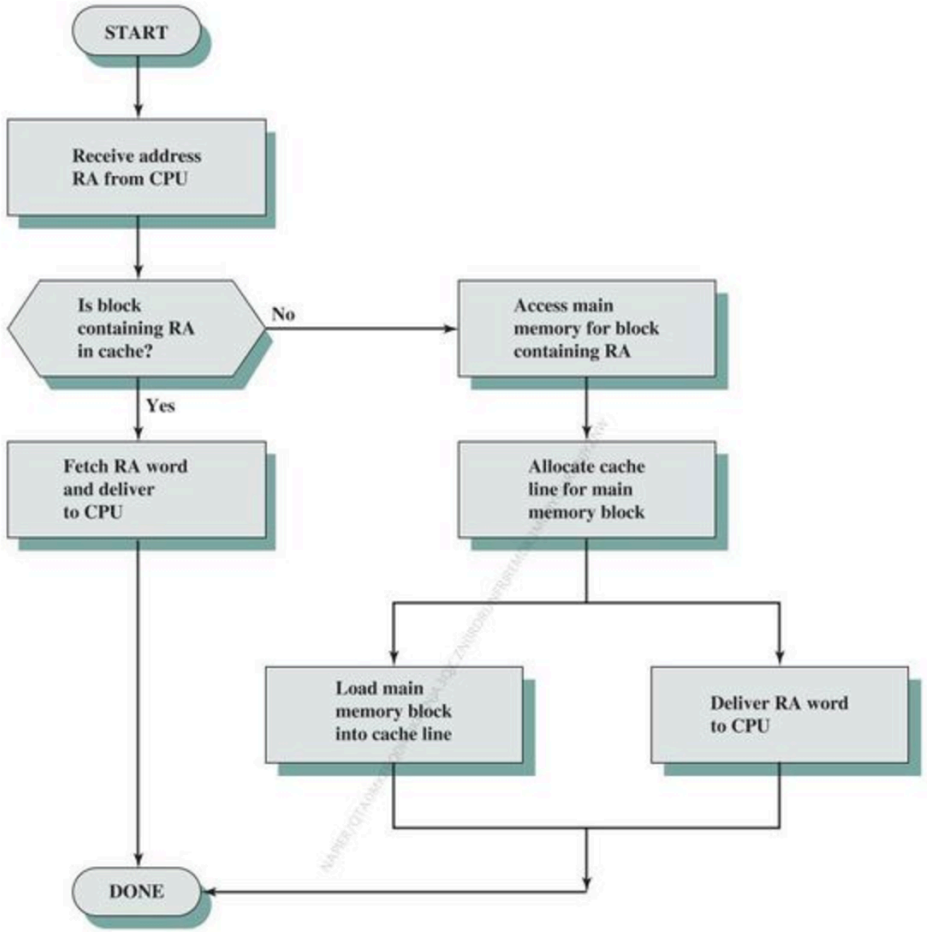
2- Why reading a block of memory rather than a word from memory ?
Hint: Phenomenon of locality of reference





Cache Read Operation

Read Address (RA)

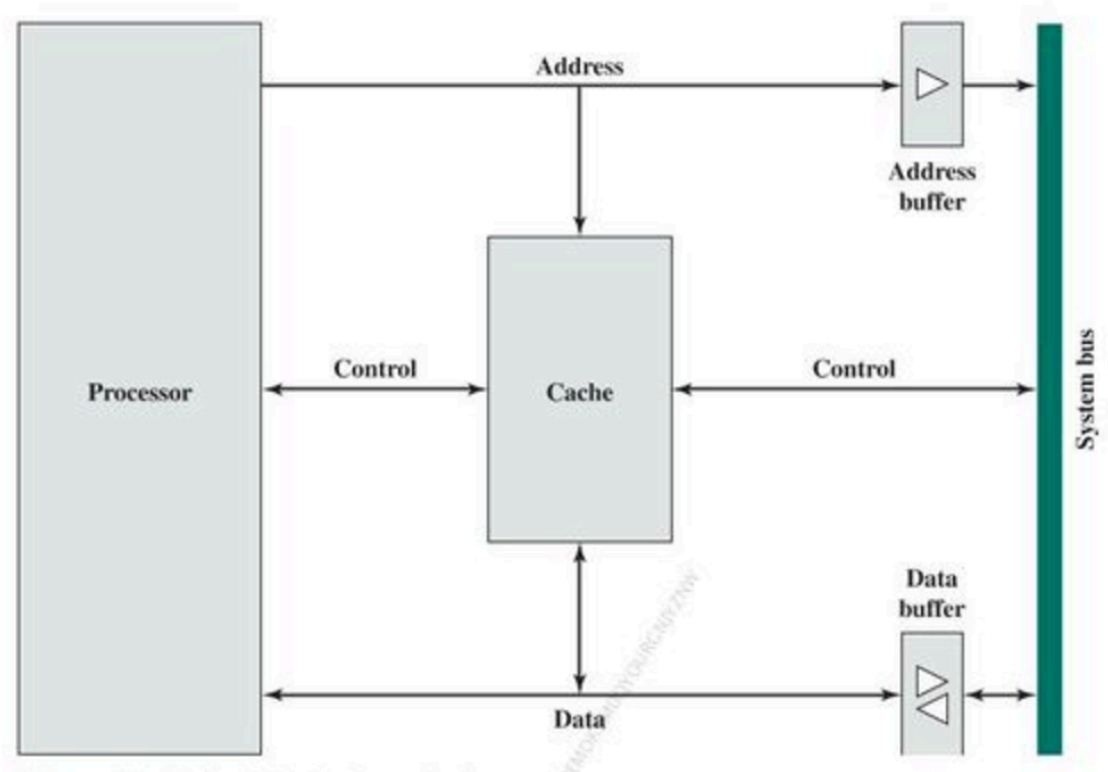


Contemporary Cash Organisations

* Connection via x3 buses
* Each then to buffers
* Then to system bus
* Cache hit
* Cache miss

1- What happen when the cache hit happens?

2- What happen when the cache miss happens?



Elements of Cache Design

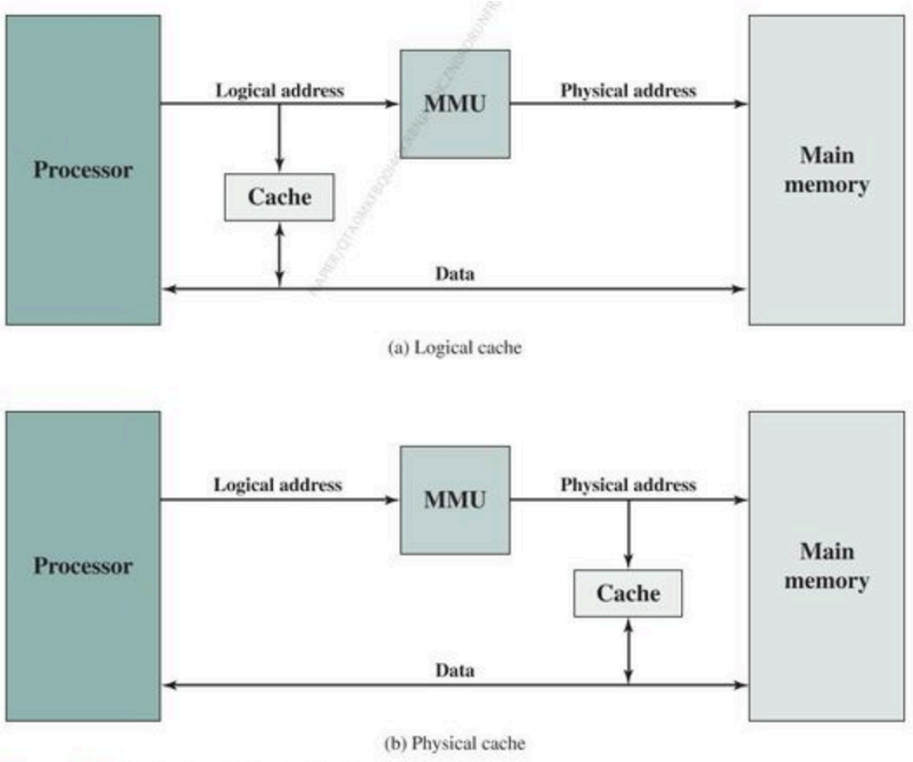
* There are a large number of cache implementations, there are a few basic design elements that serve to classify and differentiate cache architectures

Cache Addresses	**Write Policy**
Logical	Write through
Physical	Write back
Cache Size	**Line Size**
Mapping Function	**Number of Caches**
Direct	Single or two level
Associative	Unified or split
Set associative	
Replacement Algorithm	
Least recently used (LRU)	
First in first out (FIFO)	
Least frequently used (LFU)	
Random	

Cache Addresses - *Elements of Cache Design*

This can be logical (virtual) and/or physical

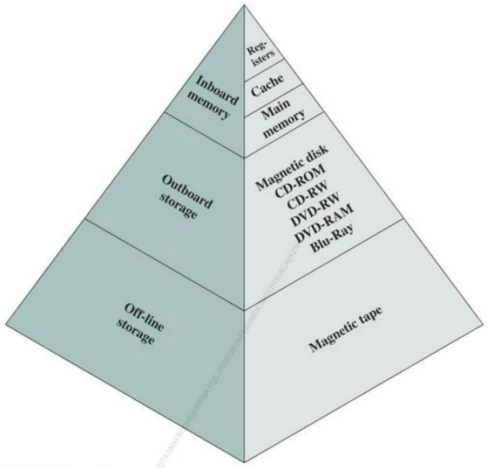
1- Logical cache vs. physical cache? E.g. cache access speed. Same VM address space for two applications



For reads and writes from main memory, a hardware Memory Management Unit (MMU) translates each virtual address into a physical address in main memory

Cache Size – *Elements of Cache Design*

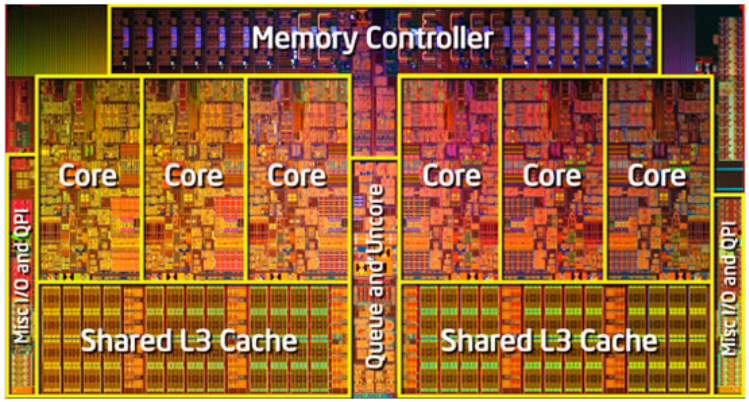
* Large cache vs. small cache – what limits the cache size?
  * Large cache tend to be slightly slower than small ones
* The available chip and board area also limits cache size
* The performance of the cache is very sensitive to the nature of the workload therefore it is impossible to arrive at a single "optimum" cache size



Cache Size con.

1- What does "/" indicate in e.g. PowerPC 620 processor?

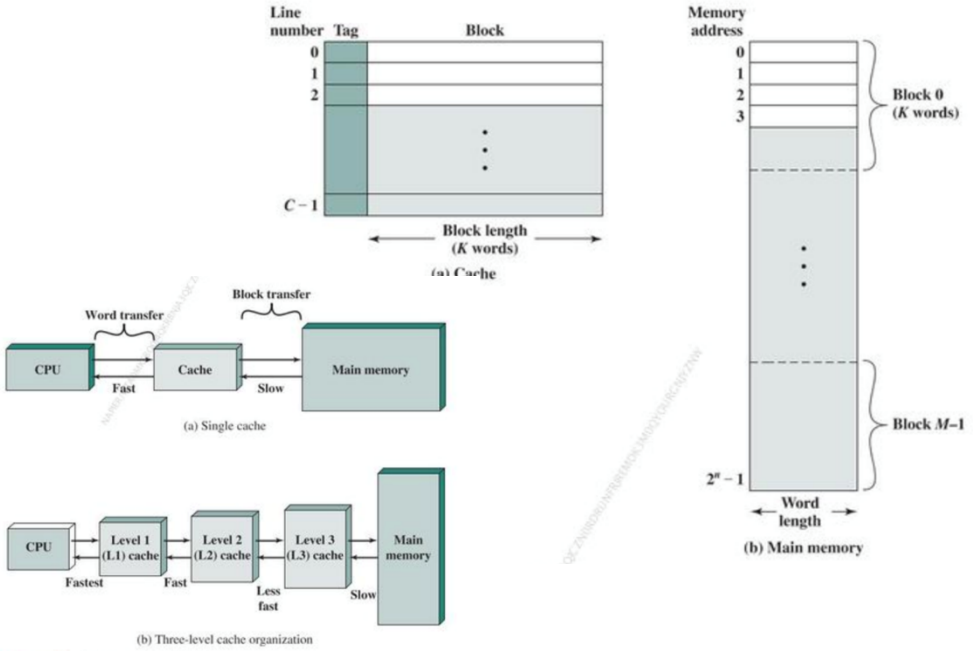
intel core i7 EE 990 processor



Processor	Type	Year of Introduction	L1 Cache[a]	L2 Cache	L3 Cache
IBM 360/85	Mainframe	1968	16–32 kB	—	—
PDP-11/70	Minicomputer	1975	1 kB	—	—
VAX 11/780	Minicomputer	1978	16 kB	—	—
IBM 3033	Mainframe	1978	64 kB	—	—
IBM 3090	Mainframe	1985	128–256 kB	—	—
Intel 80486	PC	1989	8 kB	—	—
Pentium	PC	1993	8 kB/8 kB	256–512 kB	—
PowerPC 601	PC	1993	32 kB	—	—
PowerPC 620	PC	1996	32 kB/32 kB	—	—
PowerPC G4	PC/server	1999	32 kB/32 kB	256 kB to 1 MB	2 MB
IBM S/390 G6	Mainframe	1999	256 kB	8 MB	—
Pentium 4	PC/server	2000	8 kB/8 kB	256 kB	—
IBM SP	High-end server/ supercomputer	2000	64 kB/32 kB	8 MB	—
CRAY MTA[b]	Supercomputer	2000	8 kB	2 MB	—
Itanium	PC/server	2001	16 kB/16 kB	96 kB	4 MB
Itanium 2	PC/server	2002	32 kB	256 kB	6 MB
IBM POWER5	High-end server	2003	64 kB	1.9 MB	36 MB
CRAY XD-1	Supercomputer	2004	64 kB/64 kB	1 MB	—
IBM POWER6	PC/server	2007	64 kB/64 kB	4 MB	32 MB
IBM z10	Mainframe	2008	64 kB/128 kB	3 MB	24–48 MB
Intel Core i7 EE 990	Workstation/ server	2011	6 × 32 kB/ 32 kB	1.5 MB	12 MB
IBM zEnterprise 196	Mainframe/ server	2011	24 × 64 kB/ 128 kB	24 × 1.5 MB	24 MB L3 192 MB L4

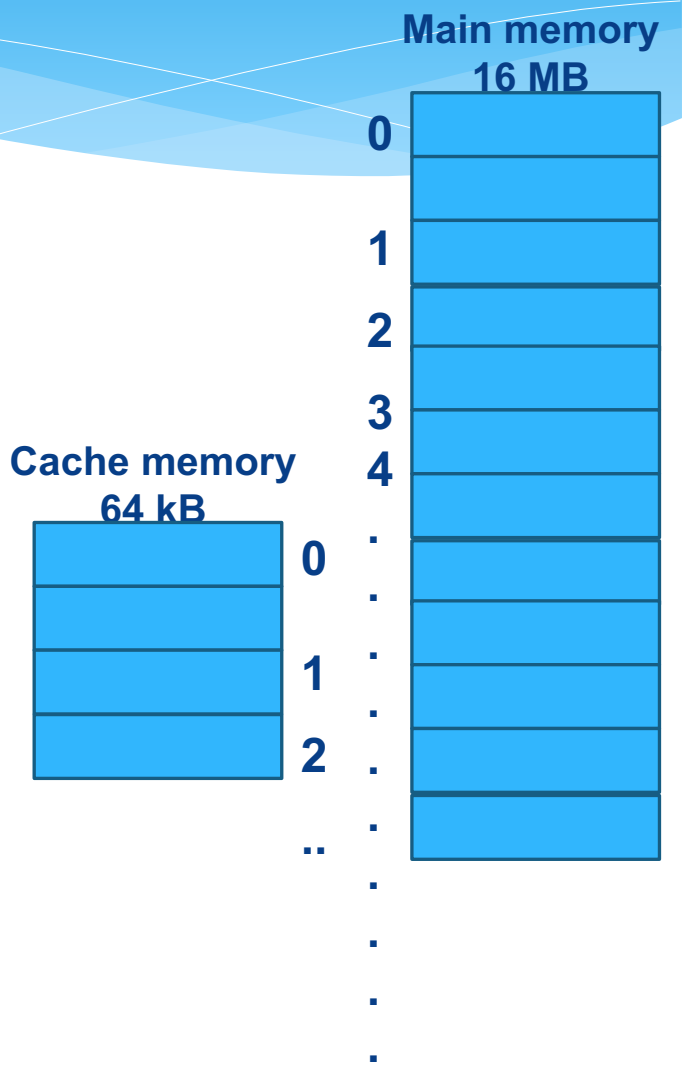
Mapping Function - *Elements of Cache Design*

* Why do we need algorithms for mapping main memory blocks into cache lines?
* The choice of the mapping function dictates how the cache is organised
  * Direct mapping
  * Associative mapping
  * Set-Associative mapping



Cache mapping techniques

* Direct Cache Mapping
  * Each block has only one place it can appear in the cache
* Set Associative Cache Mapping
  * Each block can be placed in a restricted set of places in the cache
* Fully Associative Cache Mapping
  * Each block can be placed anywhere in the cache

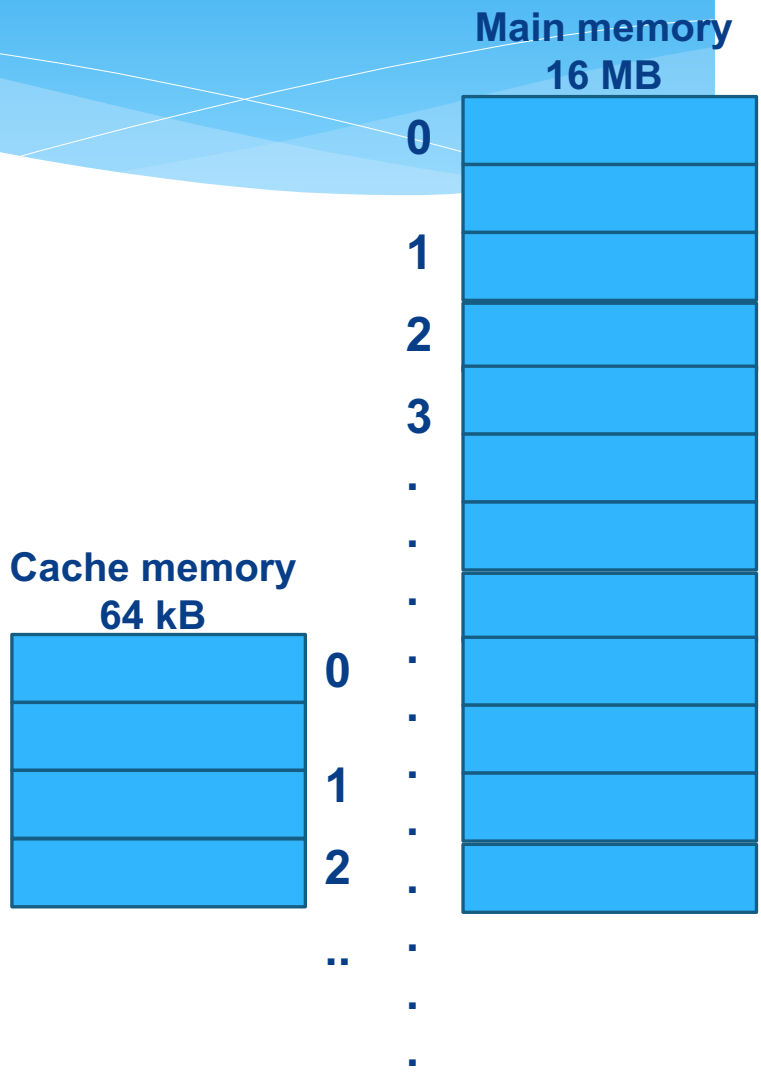


Mapping Function Con.

* For the mapping functions, imagine the following elements
  * The cache size is: 64kB
  * Data can transfer between the main memory and cache in blocks of 4 bytes each
  * The main memory size is: 16 MB

1- How many lines are in cache?
16 k lines of 4 bytes each

2- How many blocks in the main memory?
4 M blocks of 4 bytes each



Replacement Algorithms

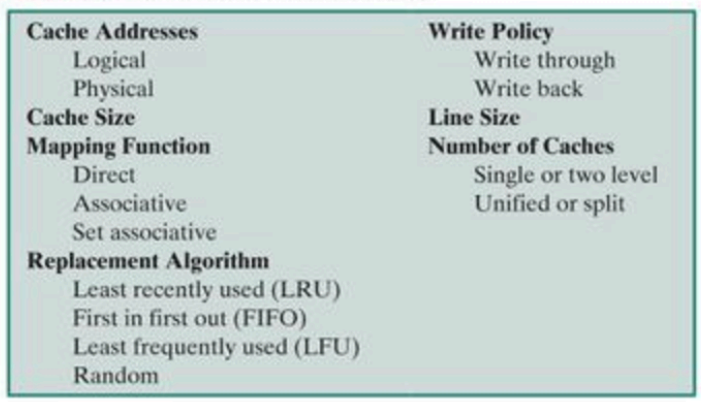
* Once the cache has been filled, when a new block is brought into the cache, one of the existing blocks must be replaced
* Direct Mapping
  * No choice is possible given that there is only one possible line for any particular block
  * The particular line will be replaced
* Associative and set-associative techniques
  * A replacement algorithm is needed
  * The algorithm needs to be implemented in hardware – but why?

Replacement Algorithms - LRU

* Least Recently Used (LRU)- the most effective and the most popular
  * Replace that block in the set that has been in the cache longest with no reference to it
  * It is a simple algorithm to implement

* For set associative, this is easily implemented
  * Each line includes a bit
  * When a line is referenced, the bit is set to 1
  * For replacement, the line whose this bit is 0 is used

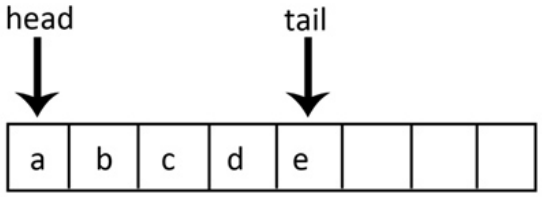
* For a fully associative cache
  * They maintain a separate list of indexes to all the lines in the cache
  * When a line is referenced, it moves to the front of the list
  * For replacement, the line at the back of the list is used

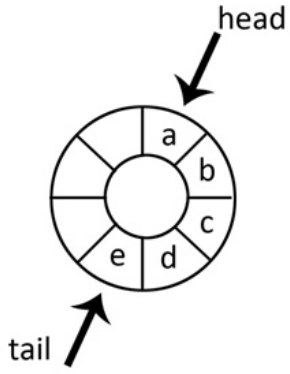


Cache Addresses	**Write Policy**
Logical	Write through
Physical	Write back
Cache Size	**Line Size**
Mapping Function	**Number of Caches**
Direct	Single or two level
Associative	Unified or split
Set associative	
Replacement Algorithm	
Least recently used (LRU)	
First in first out (FIFO)	
Least frequently used (LFU)	
Random	

Replacement Algorithms - FIFO

* First-In-First-Out (FIFO)
  * Replace that block in the set that has been in the cache longest
  * It is simple to implement as a round-robin or circular buffer technique



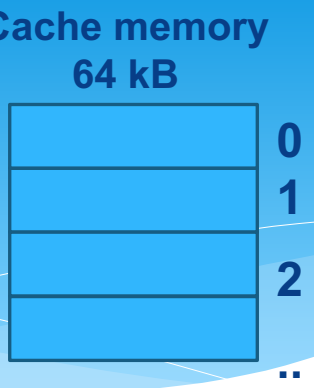


Replacement Algorithms – LFU & Random

* Least Frequently Used (LFU)
  * Replace that block in the set that has experienced the fewest references
  * This could be implemented by associating a counter with each line

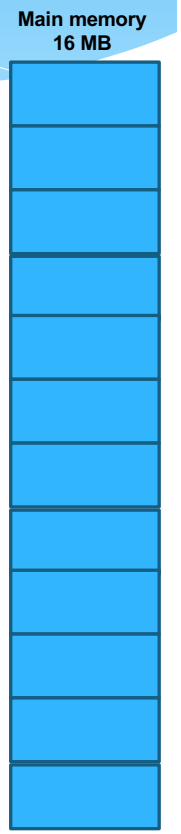
* Random technique
  * A technique not based on usage (i.e., not LRU, LFU, FIFO, or some variant)
  * It picks a line at random from among the candidate lines
  * Based on simulation studies
    * Random replacement provides only slightly lower performance comparing with the usage-based algorithms

Write Policy



* There are two cases to consider when a block that is resident in the cache is to be replaced
  * The old block in the cache has not been altered
    * In this case no need to write out the old block
  * At least one write operation has been performed on a word in that line of the cache
    * In this case then main memory must be updated by writing the line of cache out to the block of memory before bringing in the new block

Write Policy Con.



* There are variety of write policies with performance and economic trade-offs
  * Write through
  * Write back

* In general there are two problems to consider:
* First, more than one device may have access to main memory
  * For example, an I/O module
  * If a word has been altered only in the cache, then the corresponding memory word is invalid
  * Further, if the I/O device has altered main memory, then the cache word is invalid
* A more complex problem
  * Multiple processors are attached to the same bus and each processor has its own local cache
  * Then, if a word is altered in one cache, it could conceivably invalidate a word in other caches

Write Through Policy

* It is a simplest technique
* All write operations are made to main memory AND to the cache
  * This is to ensure that main memory is always valid
* Any other processor–cache module can monitor traffic to main memory to maintain consistency within its own cache

* It generates substantial memory traffic and may create a bottleneck

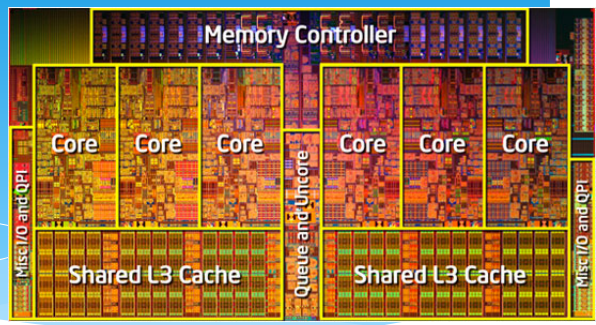
Write Back Policy

* This policy minimizes memory writes
* With write back, updates are made only in the cache
* When an update occurs, a dirty bit, or use bit, associated with the line is set. Then, when a block is replaced, it is written back to main memory if and only if the dirty bit is set

* Portions of main memory are invalid, and hence accesses by I/O modules can be allowed only through the cache
* This makes for complex circuitry and a potential bottleneck

Write Policy Con.

intel core i7 EE 990 processor



* In a bus organization, if data in one cache are altered, this invalidates not only the corresponding word in main memory, but also that same word in other caches (if any other cache happens to have that same word)
* Even if a write-through policy is used, the other caches may contain invalid data
* Cache coherency is a solution to this problem – this is an active field of research

Cache coherency

* Cache coherency is a solution for memory invalid problem – this is an active field of research
  * Bus watching with write through
  * Hardware transparency
  * Non-cacheable memory

Write Policy Con.

* Bus watching with write through policy
  * This strategy depends on the use of a write-through policy by all cache controllers
  * Each cache controller monitors the address lines to detect write operations to memory by other bus masters
  * If another master writes to a location in shared memory that also resides in the cache memory, the cache controller invalidates that cache entry

Write Policy Con.

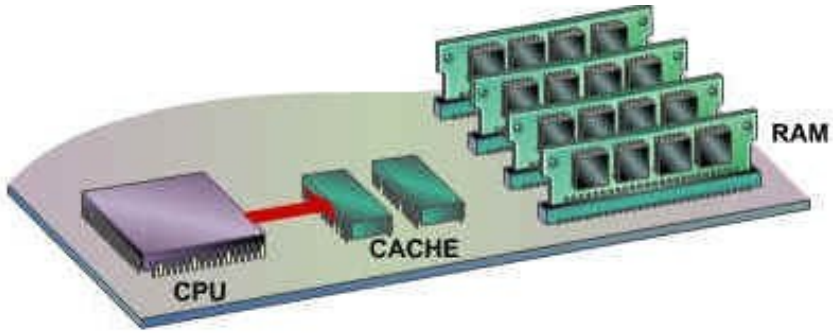
* Hardware transparency
  * Additional hardware is used to ensure that all updates to main memory via cache are reflected in all caches
  * Thus, if one processor modifies a word in its cache, this update is written to main memory
  * In addition, any matching words in other caches are similarly updated

Write Policy Con.

* Non cacheable memory
  * Only a portion of main memory is shared by more than one processor, and this is designated as non cacheable
  * In such a system, all accesses to shared memory are cache misses, because the shared memory is never copied into the cache
  * The non-cacheable memory can be identified using e.g. high-address bits

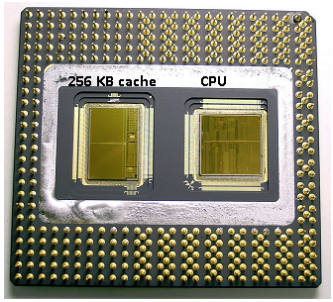
Number of Caches

* When caches were originally introduced, the typical system had a single cache
* Then the use of multiple caches has become the norm
* The aspects of using multiple caches concern about
  * The number of levels of caches
  * The use of unified versus split caches



Multilevel Caches

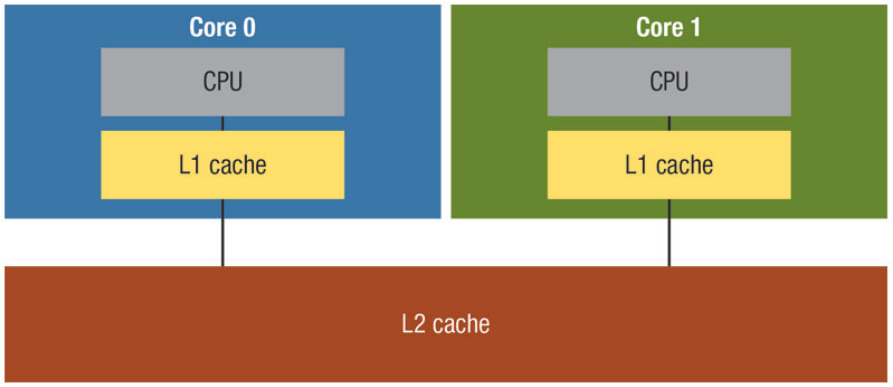
* On-chip cache
  * Cache on the same chip as the processor
    * Speeds up execution times
    * Increases overall system performance
    * Reduces the processor's external bus activity
    * Bus is free to support other transfers



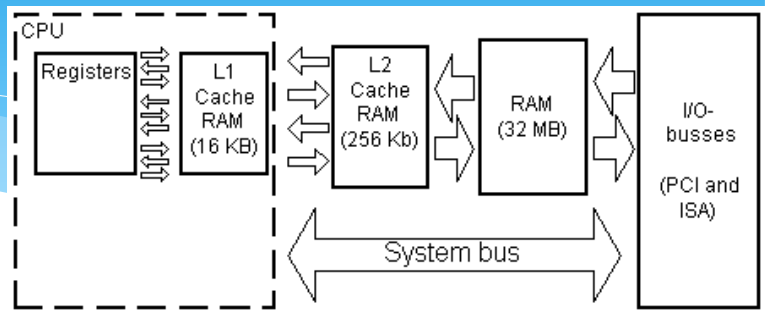
* External cache
  * Cache reachable via an external bus

Multilevel Caches Con

* The question is whether an off chip or external cache is still desirable?
  * Most modern designs include both on-chip and external caches
* This organization is known as a two-level cache
  * The internal cache: level 1 (L1)
  * The external cache: level 2 (L2)

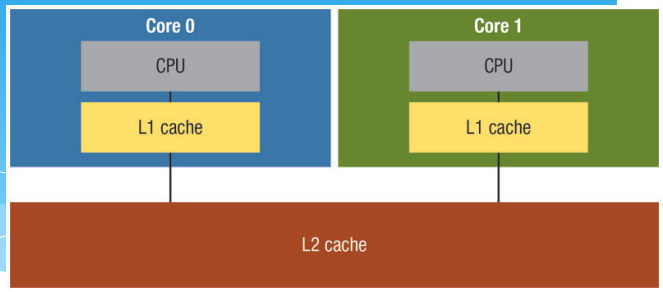


Multilevel Caches Con



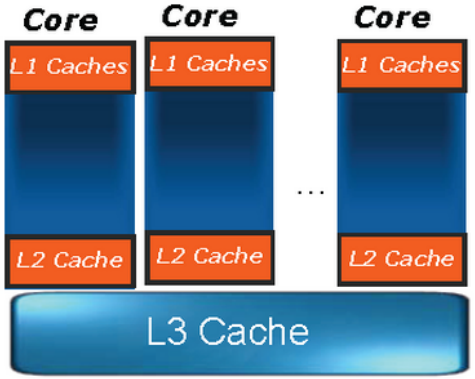
* But why having L2 cache?
* If there is no L2 cache, the processor must access the memory across the bus
  * Slow bus speed and slow memory access time are the problems; this reduces the performance
* If an L2 cache is used, then frequently the missing information can be quickly retrieved
  * If the L2 cache is fast enough to match the bus speed, then the data can be accessed using a zero-wait state transaction, the fastest type of bus transfer

Multilevel Caches Con



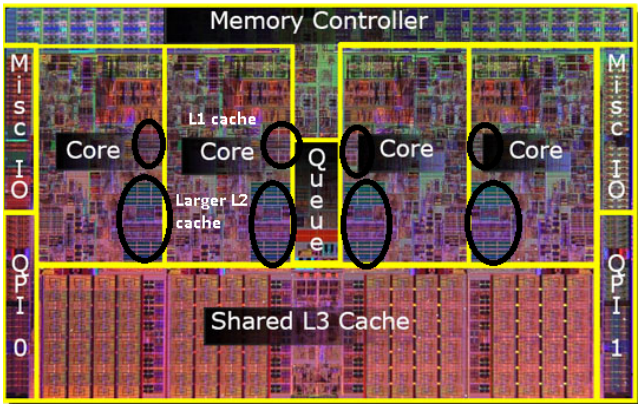
* Two features of modern cache design for multilevel caches
  * For an off-chip L2 cache, many designs use a separate data path, so as to reduce the load on the system bus
  * A number of processors incorporate the L2 cache on the processor chip, improving performance

With the increasing availability of on-chip area available for cache, most modern microprocessors have moved the L2 cache onto the processor chip and added an L3 cache

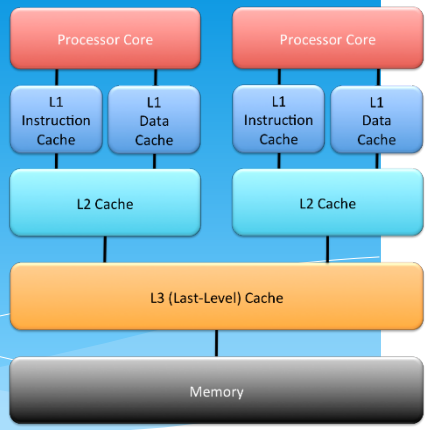


Multilevel Caches Con

* Originally, the L3 cache was accessible over the external bus
* More recently, most microprocessors have incorporated an on-chip L3 cache
* In either case, there appears to be a performance advantage to adding the third level



Unified and Split Caches



* It has become common to split the cache into two - exist at the same level, typically L1 caches
  * One dedicated to instructions
  * One dedicated to data
  * They called it split caches

* Advantages of a unified cache
  * It has a higher hit rate than split caches
  * Simpler to design: only one cache needs to be designed and implemented
  * The trend is toward split caches at the L1 and unified cashes for higher levels

* The key advantage of the split cache design is that it eliminates contention for the cache between the instruction fetch/decode unit and the execution unit – this design suits pipelining

Questions?