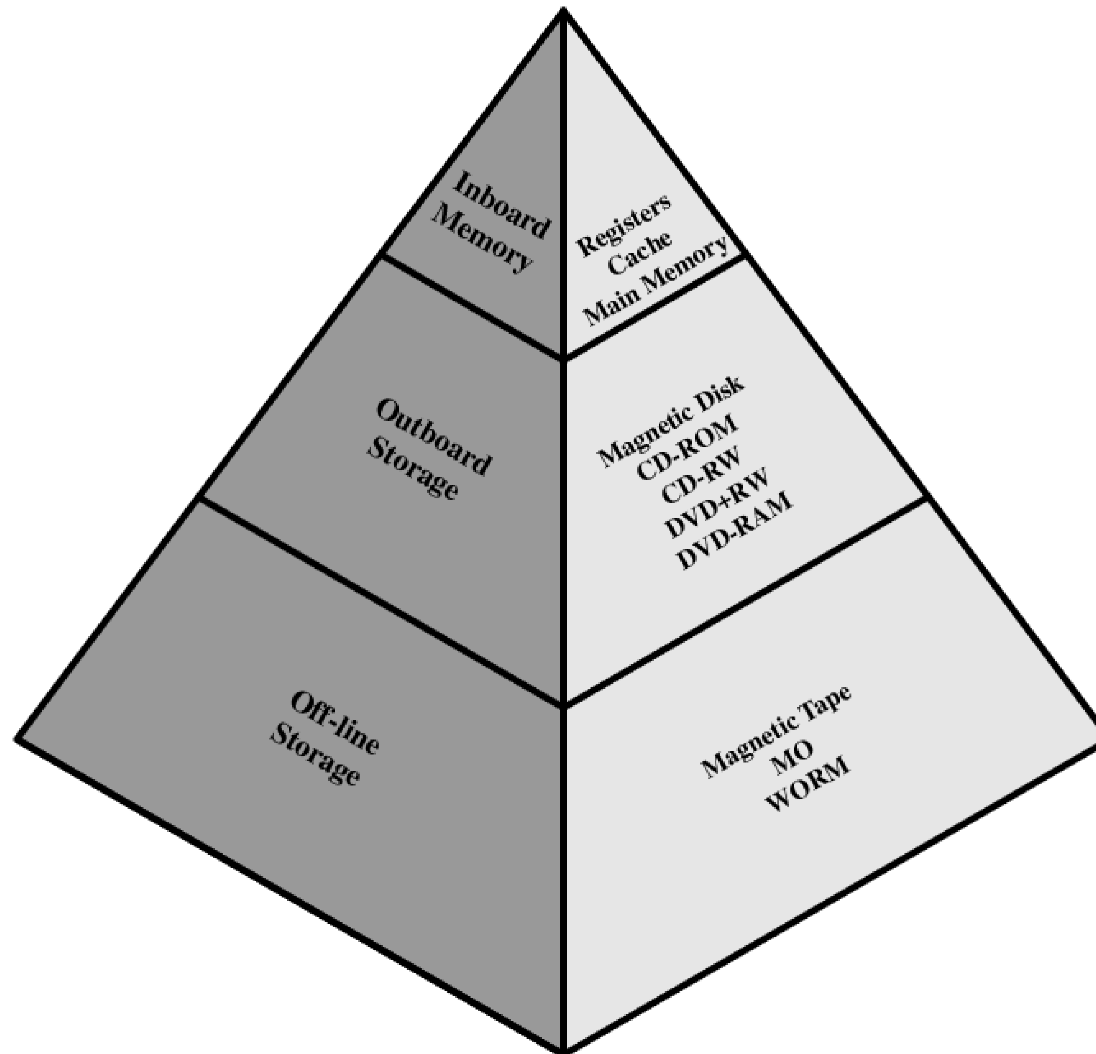
Cache memory

Lecture 4

Principles, structure, mapping

Computer memory overview



Computer memory overview

By analyzing memory hierarchy from top to bottom, the following conclusions can be done:

a. Cost is decreasing

b. Capacity is increasing

c. Access time is increasing

d. Decreasing frequency of access of the memory by the processor

Thus, smaller, more expensive, faster memories are supplemented by larger, cheaper, slower memories.

Locality of reference

The principle of Locality of reference.

During the course of execution of a program, memory references by the processor, for both instructions and data, tend to cluster.

Programs typically contain a number of iterative loops and subroutines. Once a loop or subroutine is entered, there are repeated references to a small set of instructions.

Similarly, operations on tables and arrays involve access to a clustered set of data words. Over a long period of time, the clusters in use change, but over a short period of time, the processor is primarily working with fixed clusters of memory references.

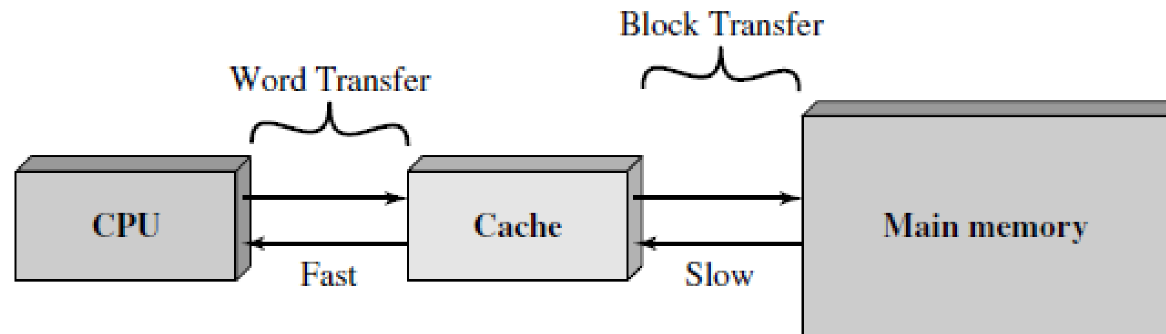
Cache memory

Cache memory is intended to give memory speed approaching that of the fastest memories available, and at the same time provide a large memory size at the price of less expensive types of semiconductor memories.

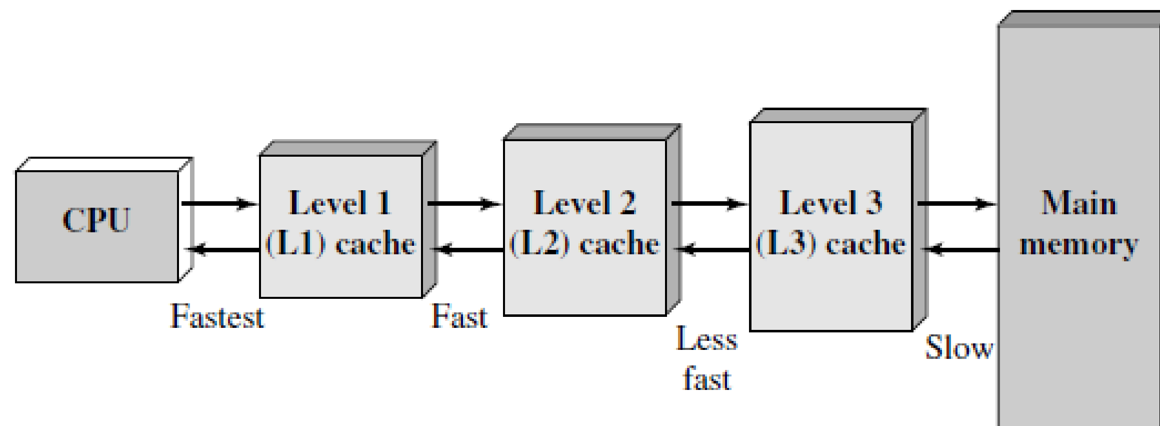
A relatively large and slow main memory together with a smaller, faster cache memory is used to increase performance. The cache contains a copy of portions of main memory.

When the processor attempts to read a word of memory, a check is made to determine if the word is in the cache. If so, the word is delivered to the processor. If not, a block of main memory, consisting of some fixed number of words, is read into the cache and then the word is delivered to the processor.

Cache and Main Memory



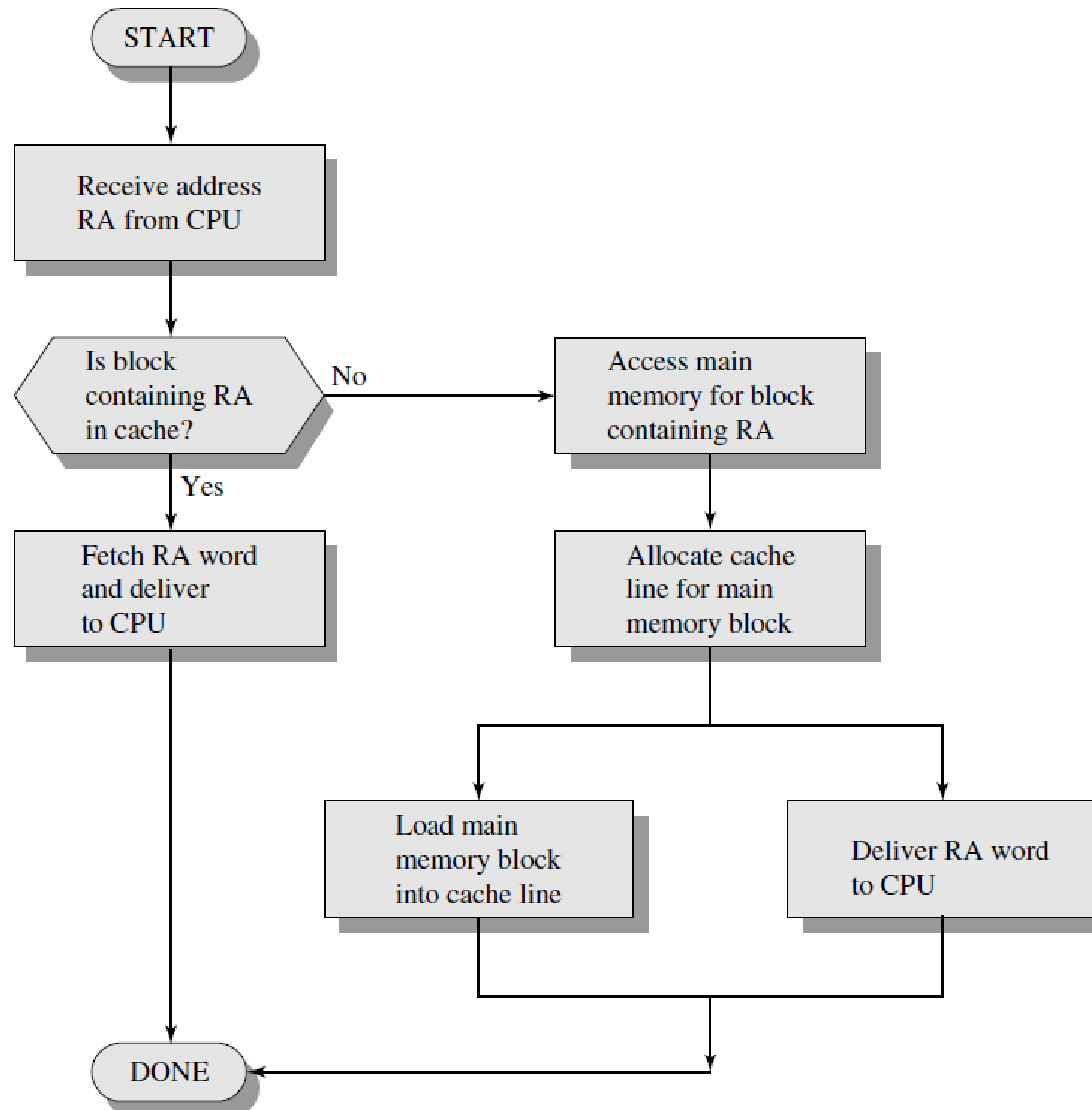
(a) Single cache



(b) Three-level cache organization

Figure depicts the use of multiple levels of cache.
The L2 cache is slower and typically larger than the L1 cache, and the L3 cache is slower and typically larger than the L2 cache.

Cache read operation



Cache/Main Memory Structure

Cache Addresses
- Logical
- Physical

Cache Size

Mapping Function
- Direct
- Associative
- Set Associative

Replacement Algorithm
- Least recently used (LRU)
- First in first out (FIFO)
- Least frequently used (LFU)
- Random

Write Policy
- Write through
- Write back
- Write once

Line Size

Number of caches
- Single or two level
- Unified or split

Cache read/write architecture

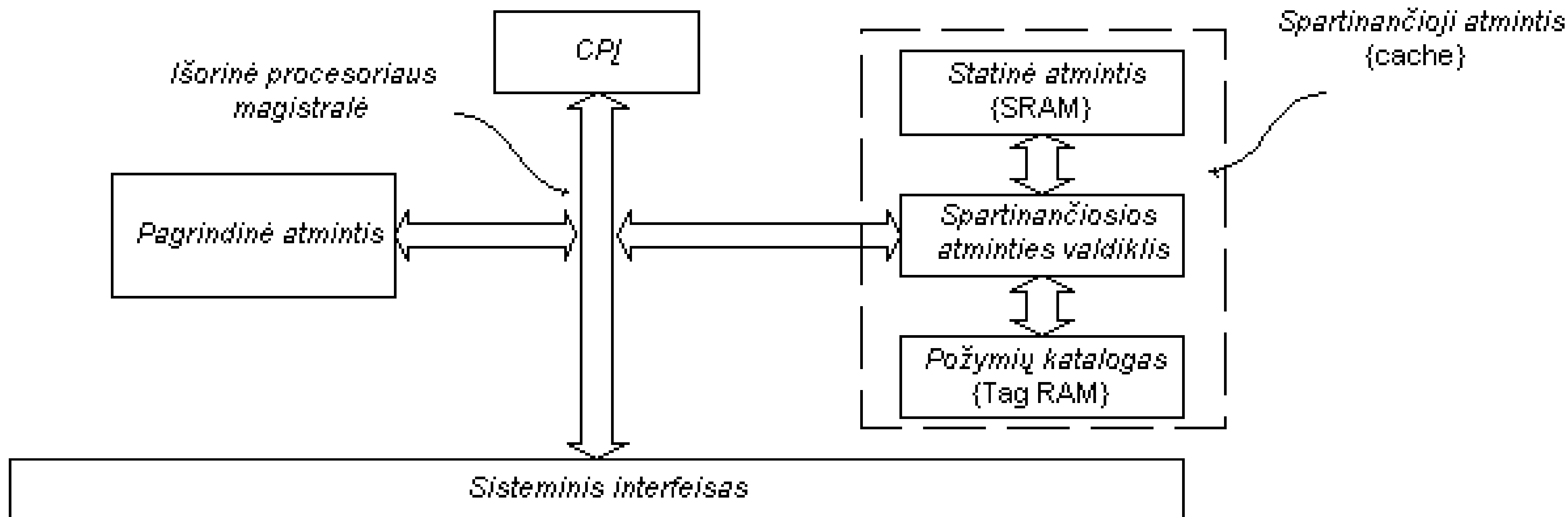
Read architecture:

- Look Aside
- Look Through

Write architecture:

- Write Back
- Write Through.

Look Aside architecture



Look Aside architecture

Cache memory operates in parallel to the main memory.

Main memory and cache know about access cycles to the main memory at the same time.

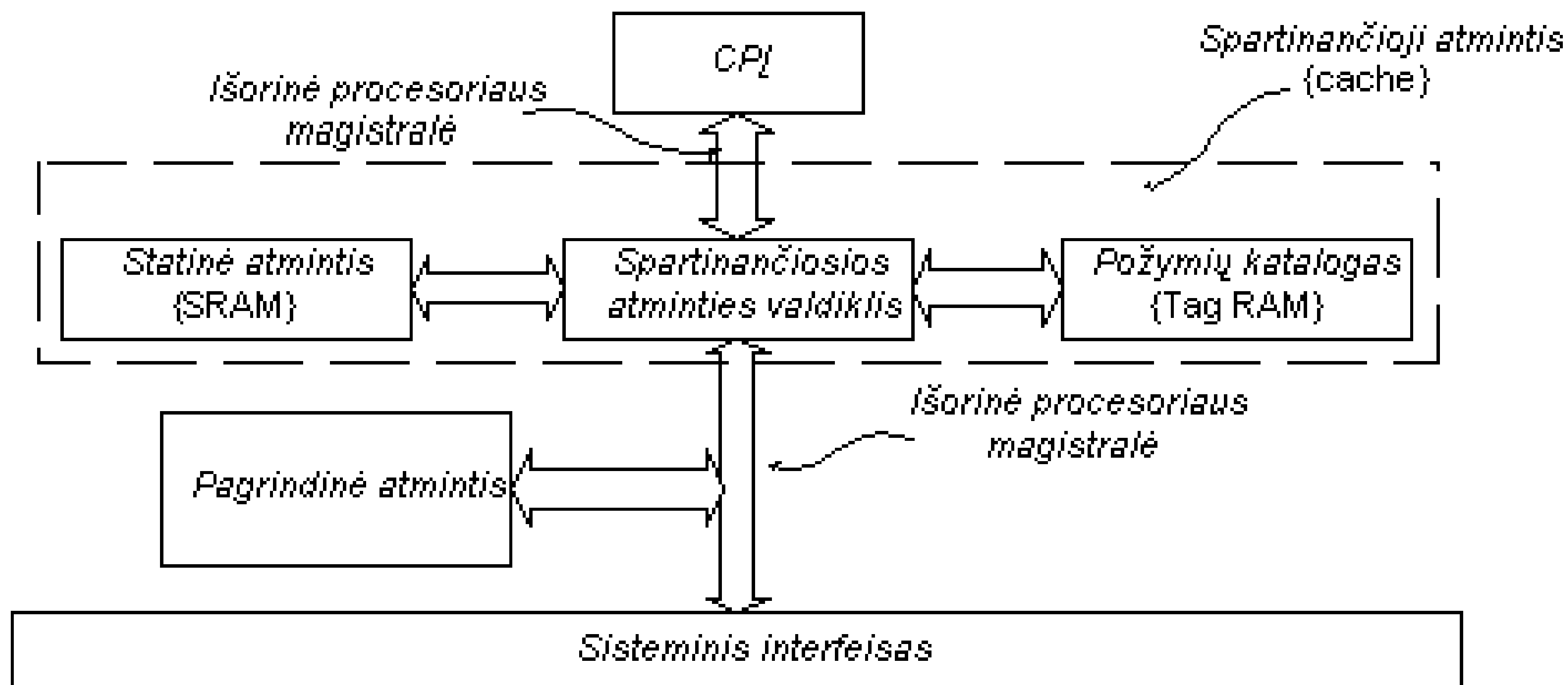
Advantages

This principle works faster in cache miss cases.

Disadvantages

Processor can’t access cache when another controller is working with main memory.

Look Through architecture



Look Through architecture

Look Through cache is located between processor and main memory. All requests and data is moving through cache before they access main memory.

Advantages

Processor can access cache when another controller is working with main memory.

Disadvantage

An access to the main memory is slower because first of all data are searched in the cache and after that in main memory.

Write-through cache

Write-through cache directs write I/O onto cache and through to underlying permanent storage before confirming I/O completion to the host.

This ensures data updates are safely stored on, for example, a shared storage array, but has the disadvantage that I/O still experiences latency based on writing to that storage.

Write-through cache is good for applications that write and then re-read data frequently as data is stored in cache and results in low read latency.

Write-back cache

Write-back cache is where write I/O is directed to cache and completion is immediately confirmed to the host.

This results in low latency and high throughput for write-intensive applications, but there is data availability exposure risk because the only copy of the written data is in cache.

Suppliers have added resiliency with products that duplicate writes. Users need to consider whether write-back cache solutions offer enough protection as data is exposed until it is staged to external storage.

Write-back cache is the best performing solution for mixed workloads as both read and write I/O have similar response time levels.

Write-around cache

Write-around cache is a similar technique to write-through cache, but write I/O is written directly to permanent storage, bypassing the cache.

This can reduce the cache being flooded with write I/O that will not subsequently be re-read, but has the disadvantage is that a read request for recently written data will create a “cache miss” and have to be read from slower bulk storage and experience higher latency.

Cache/Main Memory Structure

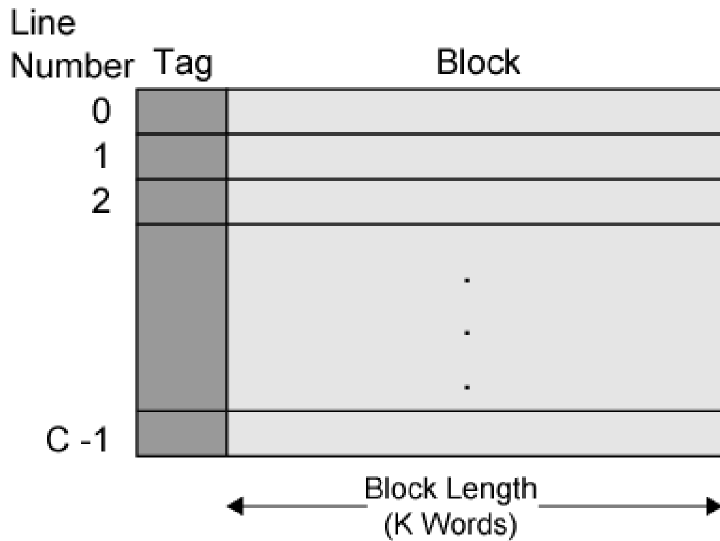
Main memory consists of up to 2^n addressable words, with each word having a unique n-bit address. For mapping purposes, this memory is considered to consist of a number of fixed length blocks of K words each.

That is, there are $M = 2^n / K$ blocks in main memory.

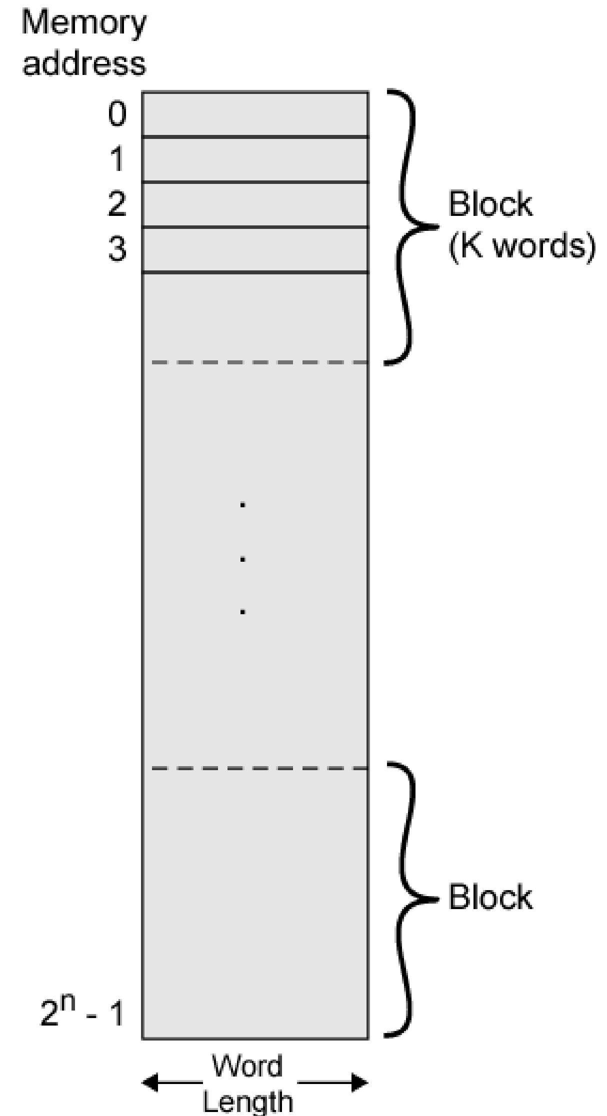
The cache consists of m blocks, called **lines**. Each line contains K words, plus a tag of a few bits. Each line also includes control bits (not shown), such as a bit to indicate whether the line has been modified since being loaded into the cache.The length of a line, not including tag and control bits, is the **line size.**The line size may be as small as 32 bits, with each “word” being a single byte; in this case the line size is 4 bytes.

The number of lines is considerably less than the number of main memory blocks (m<< M). At any time, some subset of the blocks of memory resides in lines in the cache. If a word in a block of memory is read, that block is transferred to one of the lines of the cache. Because there are more blocks than lines, an individual line cannot
be uniquely and permanently dedicated to a particular block. Thus, each line includes a **tag** that identifies which particular block is currently being stored. The tag is usually a portion of the main memory address, as described later in this section

Cache/Main Memory Structure

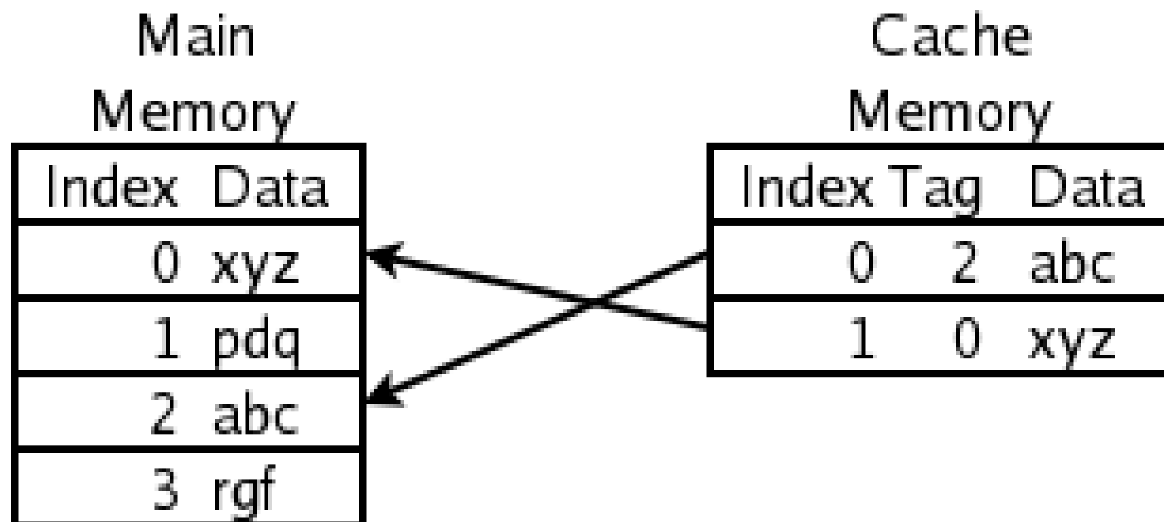


(a) Cache



(b) Main memory

Cache/Main Memory Structure



Cache line consists of:

- Index - block address of the main memory
- Data
- Tag (valid, not valid).

Mapping Function

Because there are fewer cache lines than main memory blocks, an algorithm is needed for mapping main memory blocks into cache lines.

Further, a means is needed for determining which main memory block currently occupies a cache line.

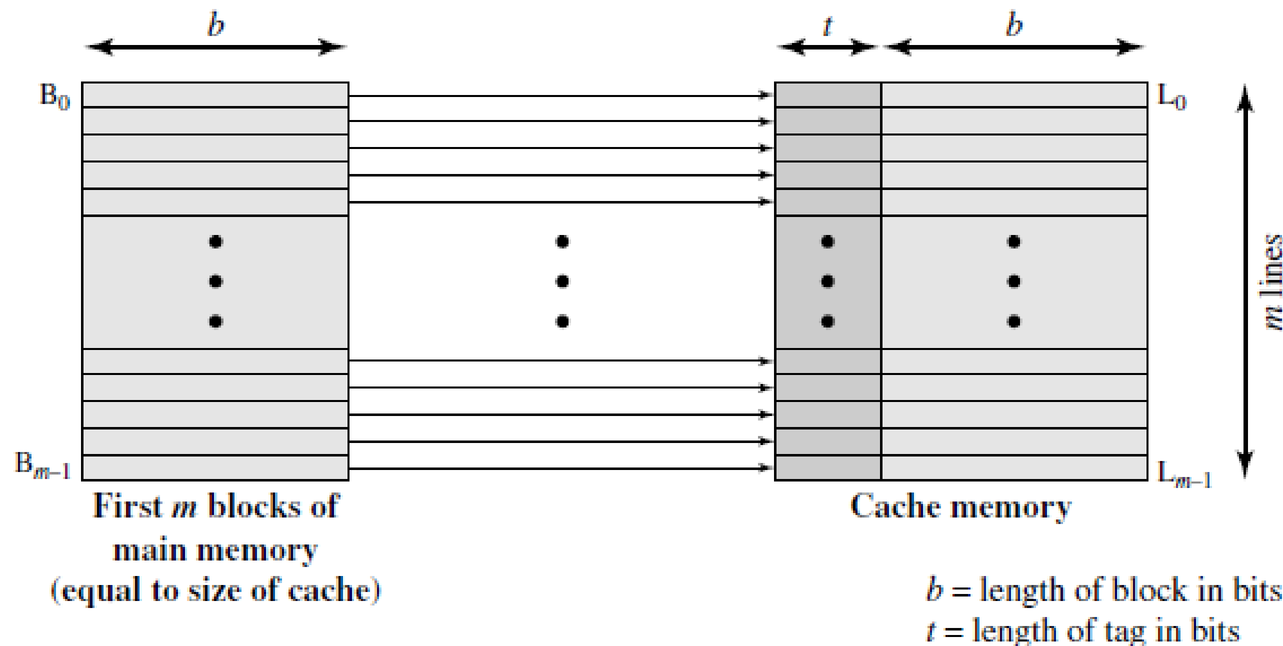
The choice of the mapping function dictates how the cache is organized.

Three techniques can be used:

- direct,
- associative,
- set associative.

Direct mapping

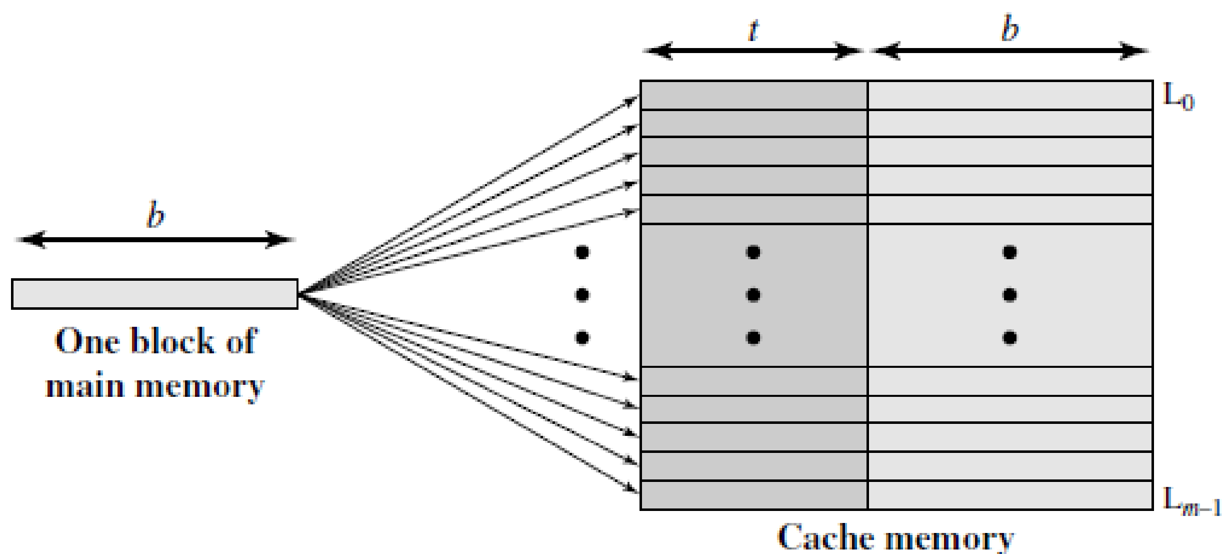
The simplest technique, known as **direct mapping**, maps each block of main memory into only one possible cache line.



Main **disadvantage** is that there is a fixed cache location for any given block. Thus, if a program happens to reference words repeatedly from two different blocks that map into the same line, then the blocks will be continually swapped in the cache, and the hit ratio will be low.

Associative mapping

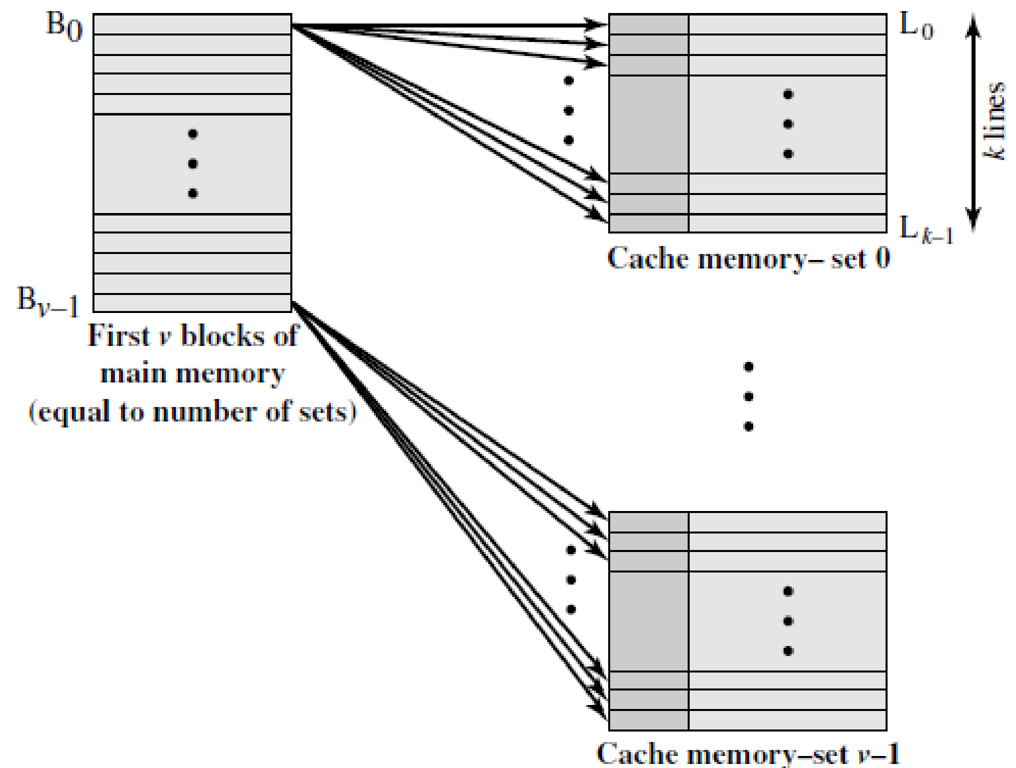
Associative mapping maps each block of main memory into any cache line. In this case, the cache control logic interprets a memory address simply as a Tag and a Word field. The Tag field uniquely identifies a block of main memory. To determine whether a block is in the cache, the cache control logic must simultaneously examine every line's tag for a match.



Set - Associative mapping

Set-associative mapping is a compromise that exhibits the strengths of both the direct and associative approaches while reducing their disadvantages.

In this case, the cache consists of a number sets, each of which consists of a number of lines.
With set-associative mapping, block B_j can be mapped into any of the lines of set j.

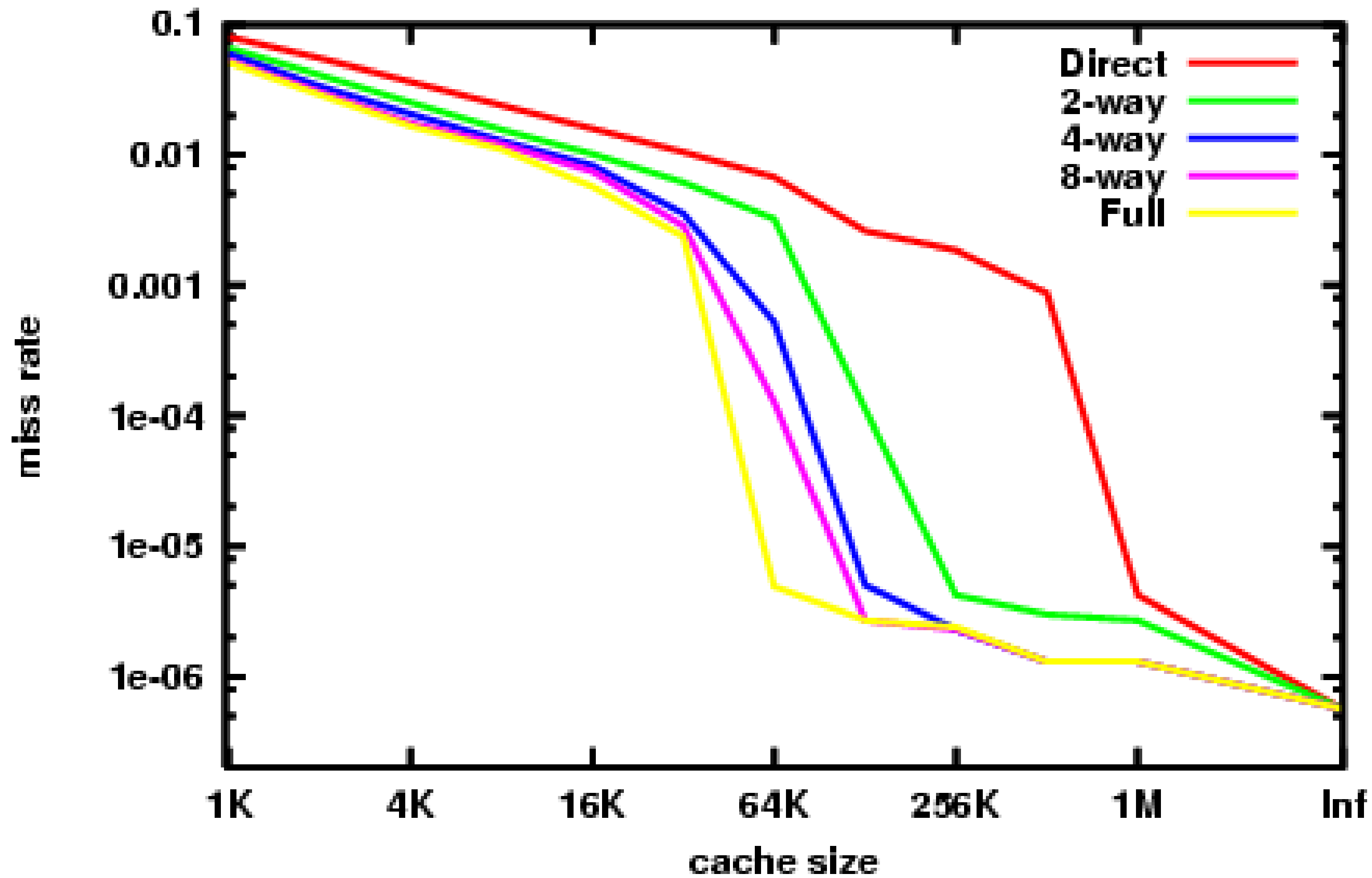


Cache characteristics

Cache characteristics

- cache hit
- cache miss
- hit ratio
- miss ratio = 1 – hit ratio
- hit time
- miss penalty

Cache efficiency



Cache examples

Processor	Type	Year of Introduction	L1 Cache[a]	L2 Cache	L3 Cache
IBM 360/85	Mainframe	1968	16 to 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 to 256 kB	—	—
Intel 80486	PC	1989	8 kB	—	—
Pentium	PC	1993	8 kB/8 kB	256 to 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 G4	Mainframe	1997	32 kB	256 KB	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
SGI Origin 2001	High-end server	2001	32 kB/32 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

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.
For direct mapping, there is only one possible line for any particular block, and no choice is possible.
For the associative and set-associative techniques, a replacement algorithm is needed.

A number of algorithms have been tried. Four of the most common are:

- Random
- LRU (Least-recently used)
- LFU (Least frequently used)
- FIFO (First in First out)

Replacement Algorithms

The most effective is **least recently used** (**LRU**): Replace that block in the set that has been in the cache longest with no reference to it.

First-in-First-out (FIFO): Replace that block in the set that has been in the cache longest. FIFO is easily implemented as a round-robin or circular buffer technique.

Least frequently used (LFU): Replace that block in the set that has experienced the fewest references. LFU could be implemented by associating a counter with each line.

Random algorithm pick random line from among the candidate lines. Simulation studies have shown that random replacement provides only slightly inferior performance to an algorithm based on usage.

Replacement Algorithms

- Random
- LRU (Least-recently used)
- LFU (Least frequently used)
- FIFO (First in First out)

The cache miss benchmarking result based on replacement algorithms (*Cache misses per 1000 instructions*)

	Associativity								
	Two-way			Four-way			Eight-way		
Size	LRU	Random	FIFO	LRU	Random	FIFO	LRU	Random	FIFO
16 KB	114.1	117.3	115.5	111.7	115.1	113.3	109.0	111.8	110.4
64 KB	103.4	104.3	103.9	102.4	102.3	103.1	99.7	100.5	100.3
256 KB	92.2	92.1	92.5	92.1	92.1	92.5	92.1	92.1	92.5

Cache optimization

Cache optimization strategies:

- **Reducing cache miss ratio**
  - Larger block size
  - Larger cache memory size
  - Larger association level
- **Reducing cache miss penalty**
  - Multilevel caches
  - Read before write
- **Reducing hit time**

Increment of block size

	Cache size			
Block size	4K	16K	64K	256K
16	8.57%	3.94%	2.04%	1.09%
32	7.24%	2.87%	1.35%	0.70%
64	7.00%	2.64%	1.06%	0.51%
128	7.78%	2.77%	1.02%	0.49%
256	9.51%	3.29%	1.15%	0.49%

Cache miss ratio is decresing when block size is increasing.

Percents mean miss ration

Inrement of the block row size

Block size	Miss penalty	Cache size			
		4K	16K	64K	256K
16	82	8.027	4.231	2.673	1.894
32	84	**7.082**	3.411	2.134	1.588
64	88	7.160	**3.323**	**1.933**	**1.449**
128	96	8.469	3.659	1.979	1.470
256	112	11.651	4.685	2.288	1.549

Results show memory access time in cycles and ns.

Multilevel caches

As logic density has increased, it has become possible to have a cache on the same chip as the processor: the **on-chip cache**.

Compared with a cache reachable via an external bus, the on-chip cache reduces the processor's external bus activity and therefore speeds up execution times and increases overall system performance.

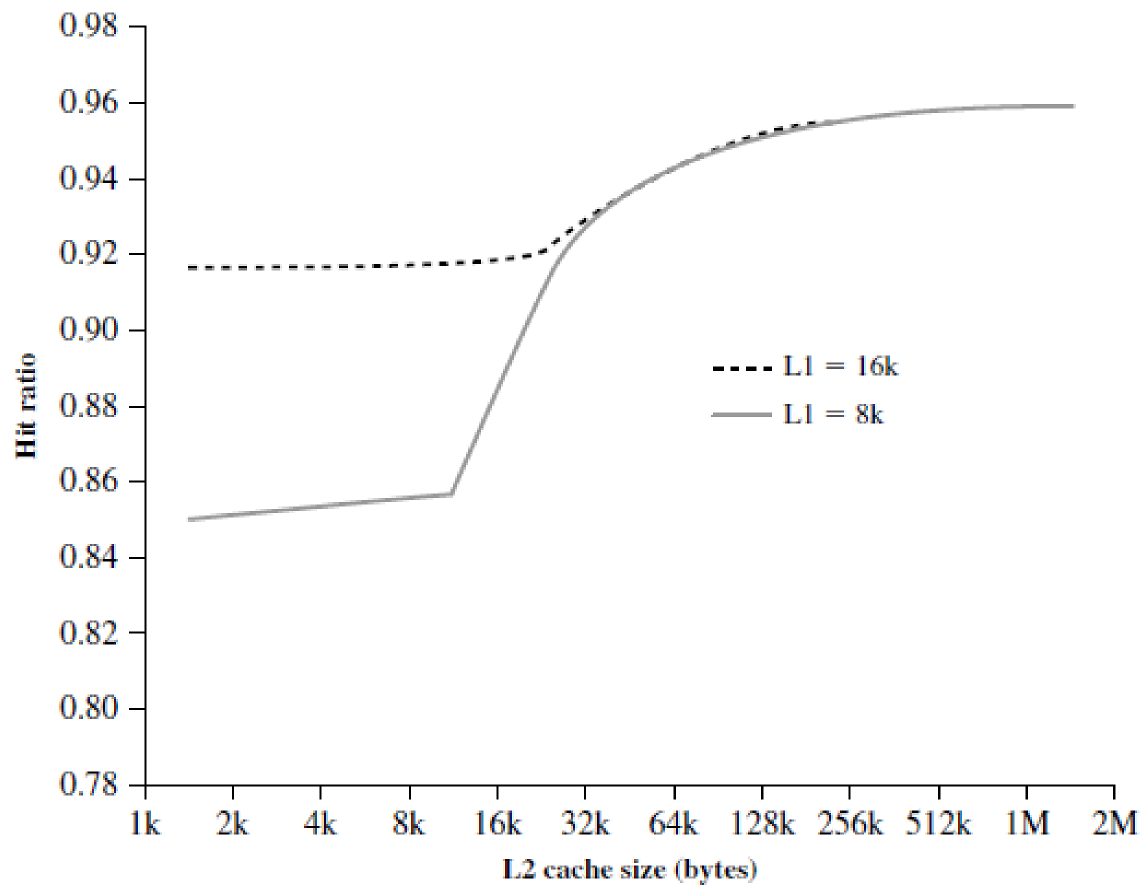
The inclusion of an **on-chip cache** leaves open the question of whether an offchip, or external, cache is still desirable. Typically, the answer is yes, and most contemporary designs include both on-chip and external caches.

Multilevel caches

The simplest such organization is known as a two-level cache, with the internal cache designated as level 1 (L1) and the external cache designated as level 2 (L2) and level 3 (L3).

The reason for including an L2 cache is the following: If there is no L2 cache and the processor makes an access request for a memory location not in the L1 cache, then the processor must access DRAM or ROM memory across the bus. Due to the typically slow bus speed and slow memory access time, this results in poor performance. On the other hand, if an L2 SRAM (static RAM) cache is used, then frequently the missing information can be quickly retrieved.

Hit ratio of multilevel cache



Cache optimization

Technique	Hit time	Miss penalty	Miss rate	Hardware complexity	Comment
Larger block size		–	+	0	Trivial; Pentium 4 L2 uses 128 bytes
Larger cache size	–		+	1	Widely used, especially for L2 caches
Higher associativity	–		+	1	Widely used
Multilevel caches		+		2	Costly hardware; harder if L1 block size ≠ L2 block size; widely used
Read priority over writes		+		1	Widely used
Avoiding address translation during cache indexing	+			1	Widely used