COMPUTER ARCHITECTURE

9 Memory hierarchy

9 Memory Hierarchy - objectives:

- A basic understanding of:
  - Locality of memory accesses
  - Importance and operation of the memory hierarchy
- Understanding caches:
  - Their effect on the speed of computation
  - A subset of the content of main memory
- Understanding concept of virtual memory

Memory hierarchy - content

9 Memory hierarchy

- □ Locality of memory accesses
- □ Memory hierarchy
- □ Cache
  - Example of cache operation
  - Types of caches regarding restrictions on mapping of blocks
  - Effect of the cache on the operating speed of the CPU
  - Case: Effect of L2 cache on the speed of the CPU
- □ Virtual memory
  - Virtual memory with paging
  - Page fault
  - Strategies and algorithms
  - Speed up address translation

Memory hierarchy - content

- Operation of the memory hierarchy
  - 4-levels memory hierarchy - average access time as seen by the CPU
  - Case: Effect of the miss-rate in main memory to the average access time in three-level memory hierarchy

9.1 Locality of memory accesses

- The principle of locality of memory accesses is one of the most important phenomena, which is observed in the operation of von Neumann‘s computer.

- Programs more commonly use commands and operands, which are close to the memory addresses currently used.

- Programs often use the same commands and operands again and again (more than once).

Locality of memory accesses

- A typical program in 90% of time executes only 10% of instructions.

Types of locality :

- Spatial locality

- Temporal locality

- Locality of memory accesses allows the main memory to be replaced with a memory hierarchy

9.2 Memory Hierarchy

- The desire of programmers:
  - access as fast as possible and
  - memory as big as possible

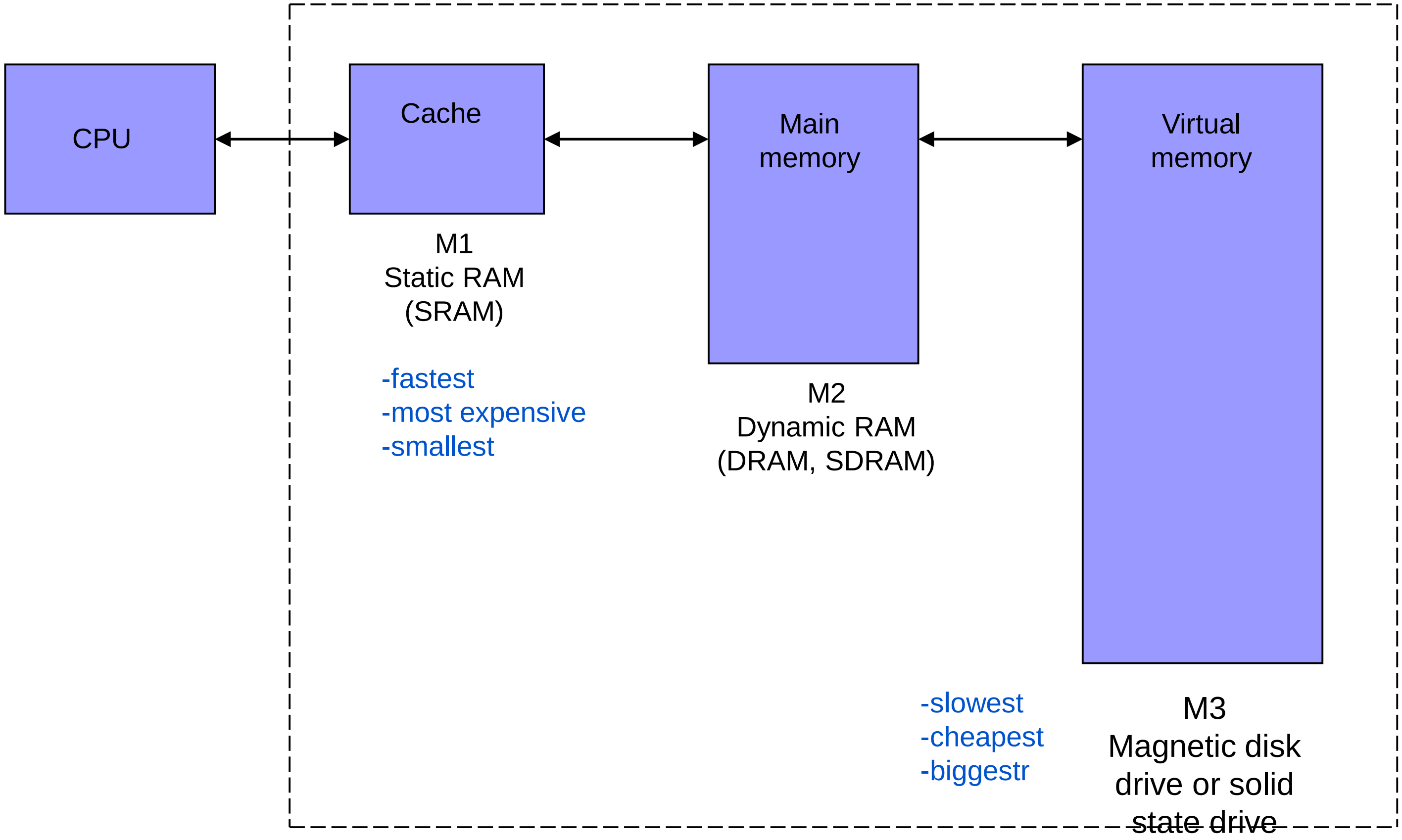
- Memory hierarchy, which consists of several separate memories with different characteristics, allowing the realization of this illusion:
  - Cache (can be in multiple levels)
  - Main memory
  - Virtual memory

- The successful operation of the memory hierarchy is possible due to the already mentioned locality of memory accesses.

Memory hierarchy

- The memory hierarchy therefore consists of several separate storage devices with different characteristics:
  - First in the hierarchy is memory M1 (closest to the CPU), the fastest, most expensive and the smallest.
  - Last in the hierarchy is memory Mn (farthest from the CPU), the cheapest, largest and slowest.
- The aim of the memory hierarchy is that the big, slow and inexpensive memory Mn seems like fast and expensive memory M1.

Case: three-level memory hierarchy



CPU sees memory hierarchy as the Main memory defined in Von Neumann‘s model

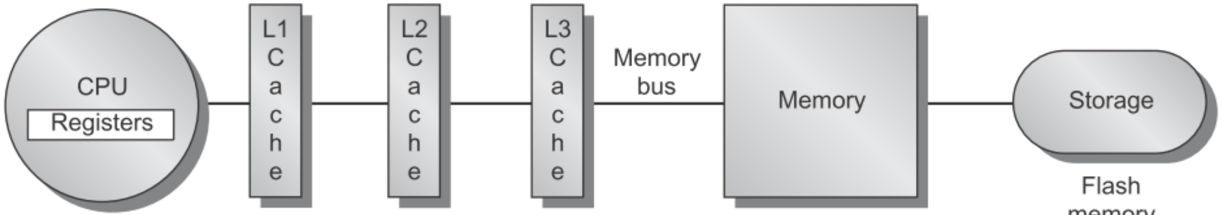
- Operation rule of the hierarchy is that the memory content at level i is a subset of the content at the level i + 1.
- If the information accessed by the CPU is not in M1, it must be transferred from M2 to M1. If it is not in M2, it is transferred first from M3 to M2 and then from M2 to M1.
- Transfers from one level to the next level is carried out automatically, without the involvement of a programmer.

Memory hierarchy

- From CPU, a 3-level memory hierarchy is seen as the size of the main memory M3, with a speed close to the speed of M1.

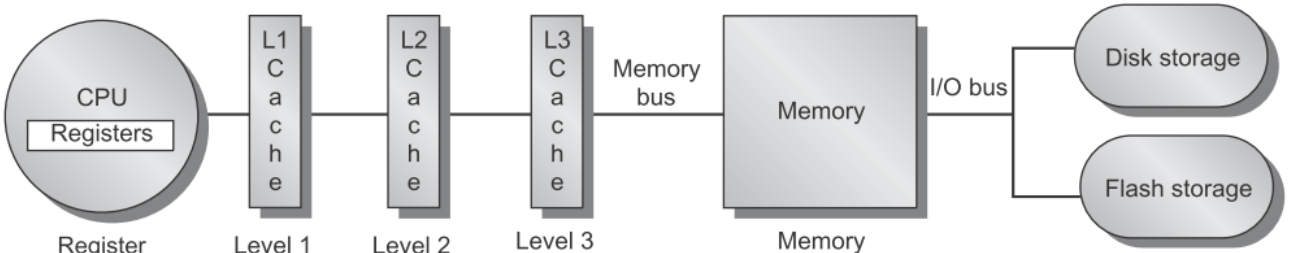
- The memory hierarchy would be useless without locality of memory accesses.

Memories in memory hierarchy [Patt]



		Register reference	Level 1 Cache reference	Level 2 Cache reference	Level 3 Cache reference	Memory reference	Flash memory reference
Laptop	Size:	1000 bytes	64 KB	256 KB	4-8 MB	4–16 GB	256 GB-1 TB
	Speed:	300 ps	1 ns	3–10 ns	10–20 ns	50–100 ns	50-100 uS
Desktop	Size:	2000 bytes	64 KB	256 KB	8-32 MB	8–64 GB	256 GB-2 TB
	Speed:	300 ps	1 ns	3–10 ns	10–20 ns	50–100 ns	50-100 uS

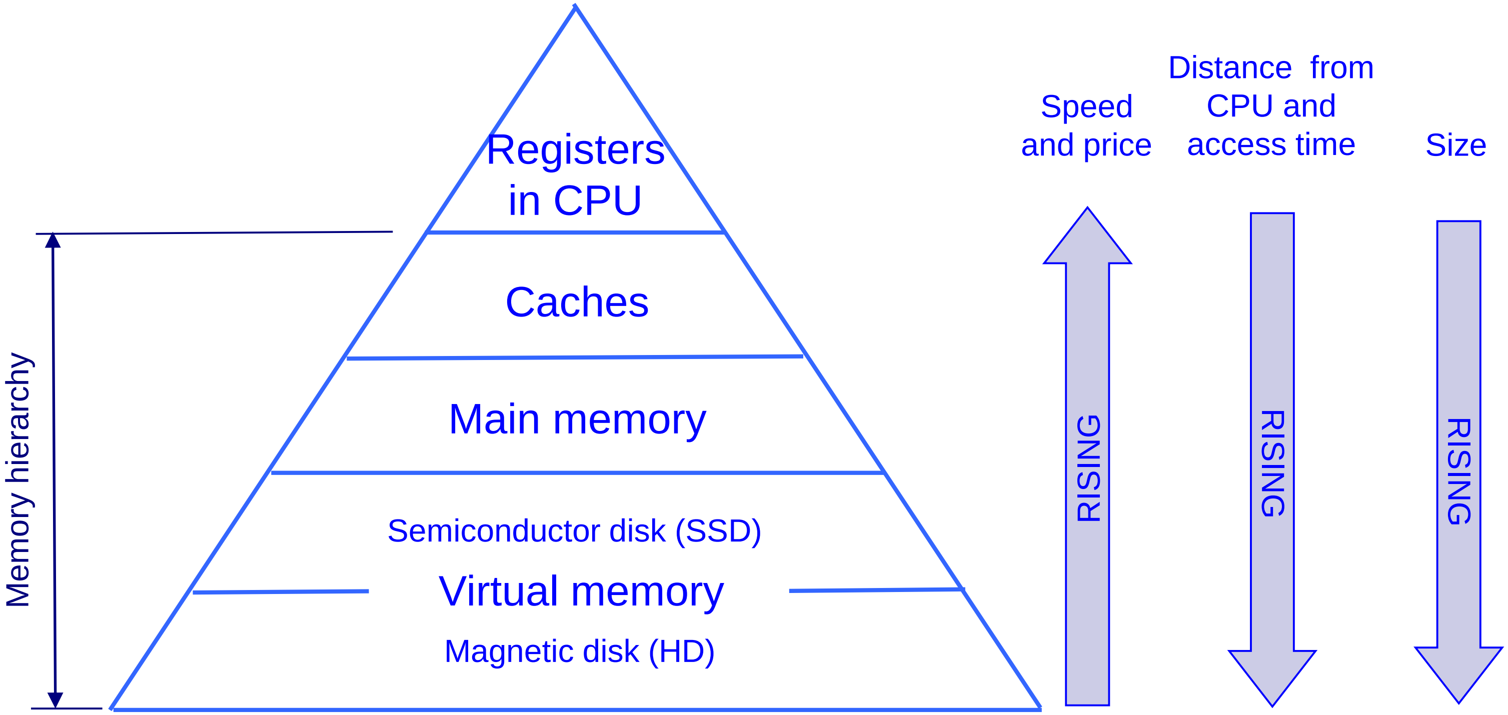
(B) Memory hierarchy for a laptop or a desktop



	Register reference	Level 1 Cache reference	Level 2 Cache reference	Level 3 Cache reference	Memory reference	Disk memory reference	Flash memory reference
Size:	4000 bytes	64 KB	256 KB	16-64 MB	32–256 GB	16–64 TB	1-16 TB
Speed:	200 ps	1 ns	3–10 ns	10–20 ns	50–100 ns	5–10 ms	100-200 us

(C) Memory hierarchy for server

Memories in memory hierarchy



9.3 Cache

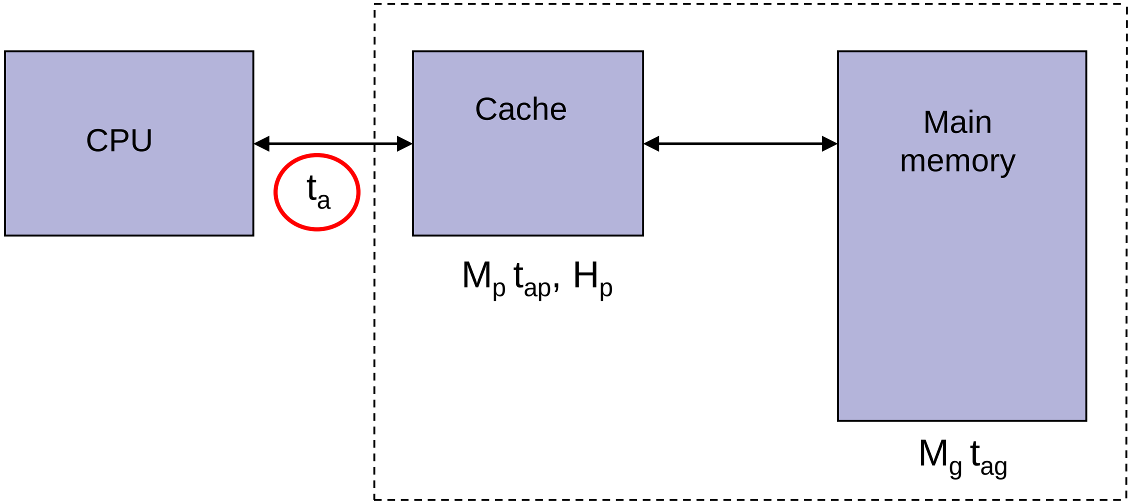
- Cache is a small, fast memory (SRAM) between the CPU and main memory.

- Using cache in the memory hierarchy creates the illusion of fast memory, which is faster than main memory.

- The contents of the cache is a subset of the contents of main memory.

- CPU with memory address always accesses to the cache.

Cache - principles



2-level memory hierarchy

M_p cache
t_{ap} time access to the cache [ns]
H_p hit-rate in cache [%]

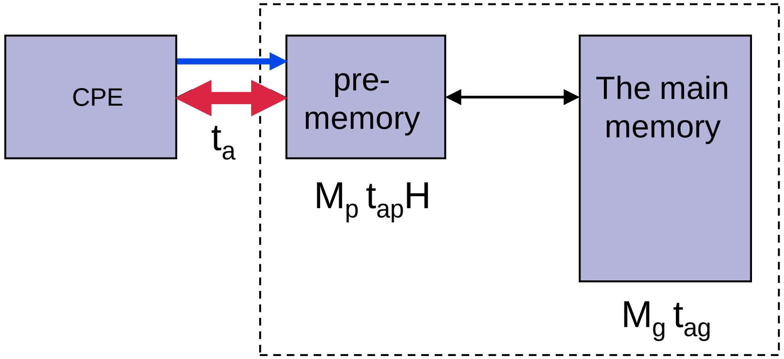
M_g main memory
t_{ag} access time to main memory [ns]
(hit-rate in the main memory is 100%)

t_a average time access the entire hierarchy, as seen by the CPU [ns]

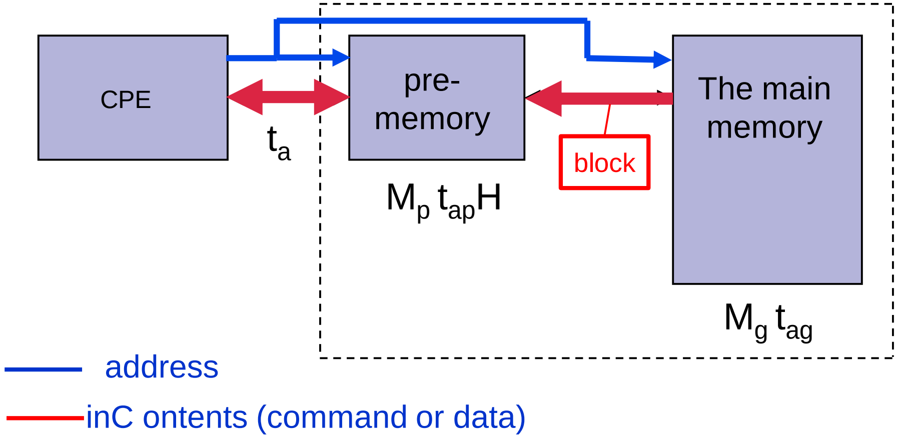
Cache - principles

- When the CPU Access to information (command, operand) cache are two options:
  - □ **The goal** (Hit) if the address (and content from this address) in the cache ⇒ access time is t_{ap}
  - □ **Zgrešitev** (miss) If the address (and content) is not in the cache ⇒ access time is t_{ap} + t_{ag}

The goal of the cache



Zgrešitev cache



- Success of operation of the cache is measured:
  - □ With hit-rate *H*

$$H = \frac{N_p}{N} = \frac{N_p}{N_g + N_p}$$

N - total number of accesses to the cache ($N = N_g + N_p$)

N_p - number of hits (the desired information is stored in the cache)

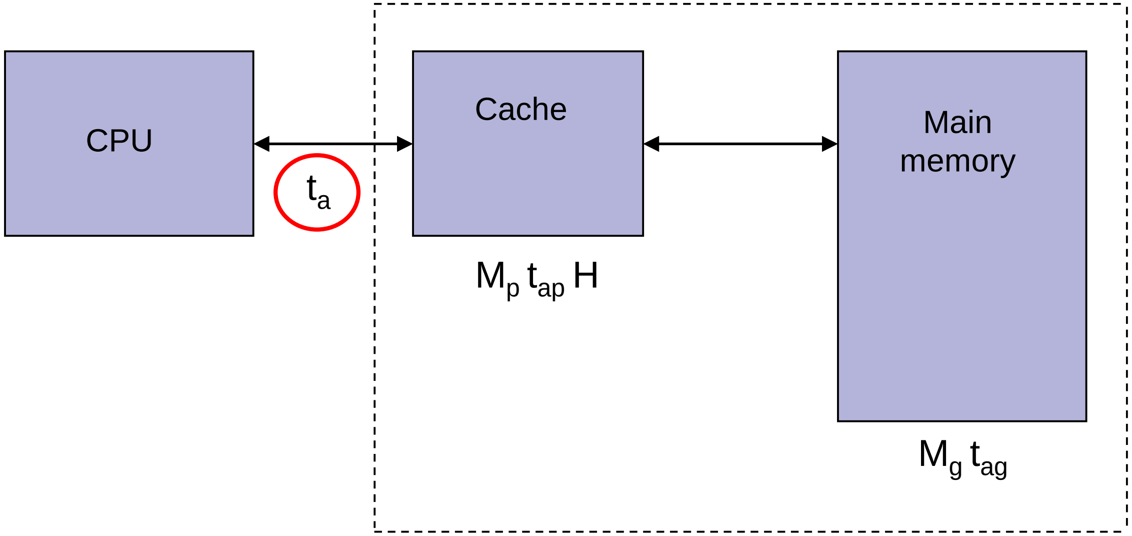
N_g - number of misses (desired information is not in cache,
the transfer of information from the main memory
to the cache is needed)

  - □ Or with the miss rate *1 - H* (we want to minimize it)

- In cache, hit rate is H > 0.9 (90%)

- In case of miss, access to the main memory is necessary.

Cache - principles



- Average memory access time t_a (AMAT) as seen by the CPU is:

$$t_a = H\,t_{ap} + (1-H)(t_{ap} + t_{ag})$$

$$t_a = t_{ap} + (1-H)t_{ag}$$

- When calculating two cache specialties should be considered:
  - Between main memory and cache there is always transfer of **the cache block (cache line)**, that consists of several adjacent memory words (bytes)
  - Time in the computer is usually measured in clock periods
- Access time t_{ap} to the information in the cache level L1 in most computers is from one to several clock periods.
- In the case of miss $\Rightarrow$ time for the access to the main memory and cache block transfer is denoted as **miss penalty t_B.**

Cache - principles

- Miss penalty t_B is time that in case of miss (miss rate is *1 - H*) is added to the access time to the cache.

- Miss penalty is typically between 10 and 100 clock periods.

- If your computer has a cache level L2, then miss penalty is much smaller because the L2 cache is faster than main memory.

Cache - principles

- Average access time t_a including miss penalty, is defined as:

$$t_a = t_{ap} + (1 - H)t_B$$

t_{ap} - access time of cache

$(1 - H)$ - the probability of miss in cache

t_B – miss penalty (access time to the main memory + time for transfer of cache block (line))

- If times t_{ap} and t_B are expressed in clock periods, then also a result t_a is in clock periods.

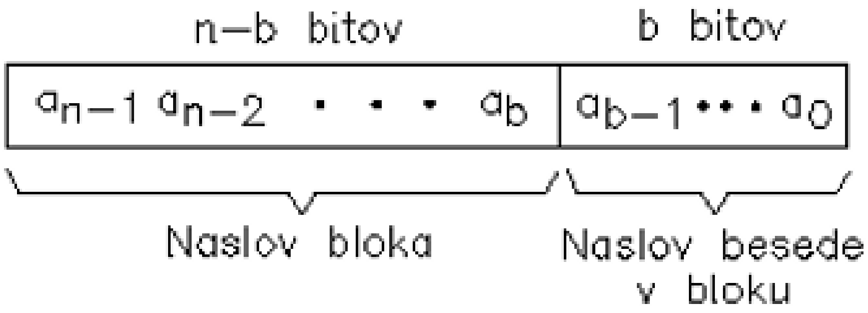
- Average access time in seconds (t_{CPE} is the duration of one clock period in seconds):

$$t_a \text{ [s]} = t_a \text{ [Clock period]} * t_{CPE} \text{ [s]}$$

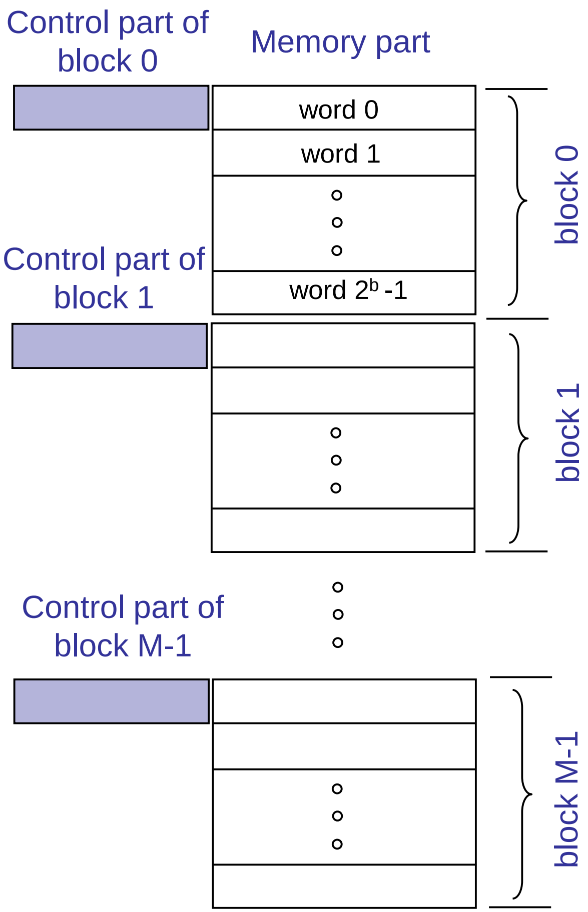
Cache - structure

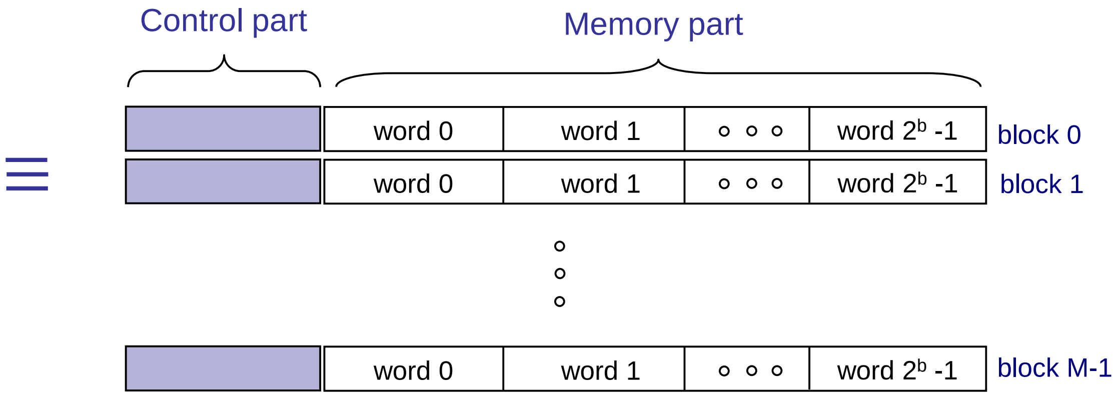
- The content of the cache varies ⇒
  - □ Cache blocks are transferred from the main memory
  - □ and addresses of these blocks (block numbers from the main memory)
- Therefore, each cache consists of two parts:
  - □ **Memory part,** that is divided into blocks or cache lines
  - □ **Control part**, consisting of control words. Each block in the memory part corresponds to certain control word containing the address of a block (number of block in main memory), which is contained in the memory part.

Cache - structure



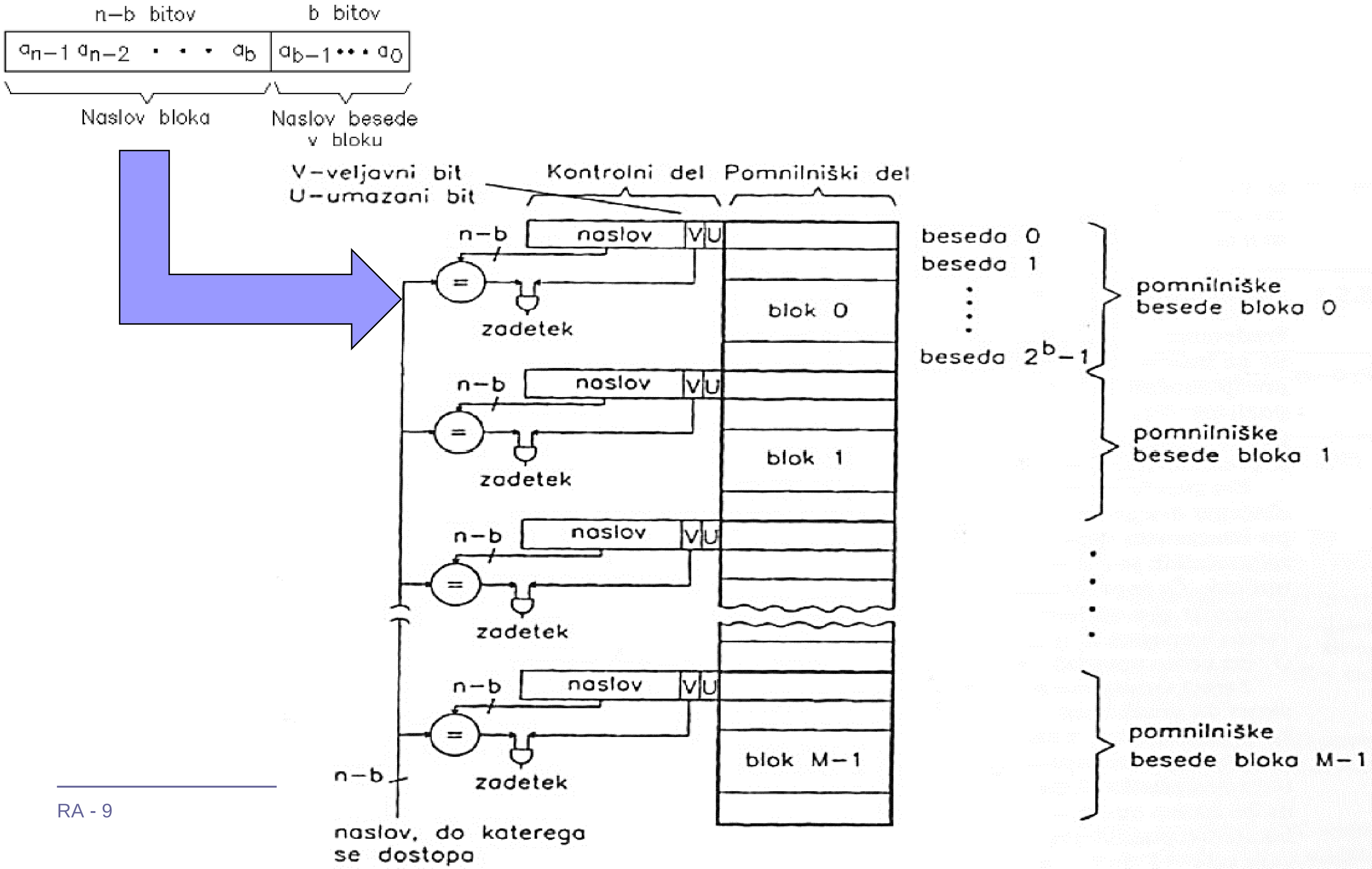
Cache





The block or a cache line = 2^b the words

Structure and operation of cache



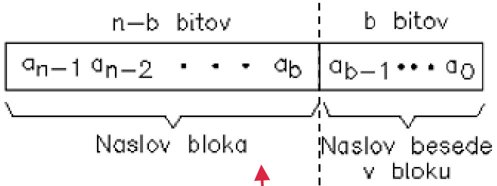
Cache - structure

- Block (or a cache line) consists of a number of consecutive memory words (memory word is usually 1 byte in size).
- Block size ($B = 2^b$) is typically 4 to 512 memory words.
- Remember: Between main memory and the cache, only the **entire block** is transferred.
- When a block from main memory is transferred to the free block frame in the cache :
  - □ Content of the block is transferred to the memory part of the block in cache
  - □ Address (number) of block is transferred to the control part of the block in cache

cache - example

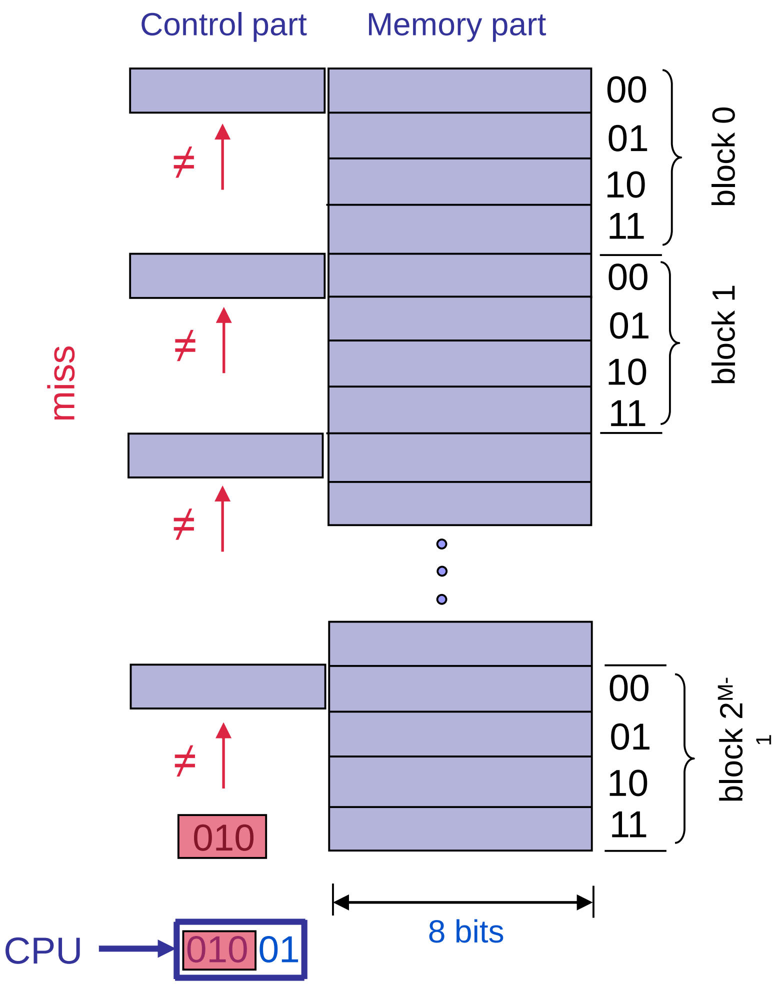
- An example of cache operation:
  - □ Assume:
    - processor accesses the memory words with the following sequence of memory addresses:
    - 9, 10, 11, 2, 3, 9, 10, 11, 2, ...;
    - Cache consists of blocks of 4 bytes (B = 2^2 = 4) and it is initially empty;
    - Memory address is 5 bits long.
    - The top three bits of the memory address specify a block number,

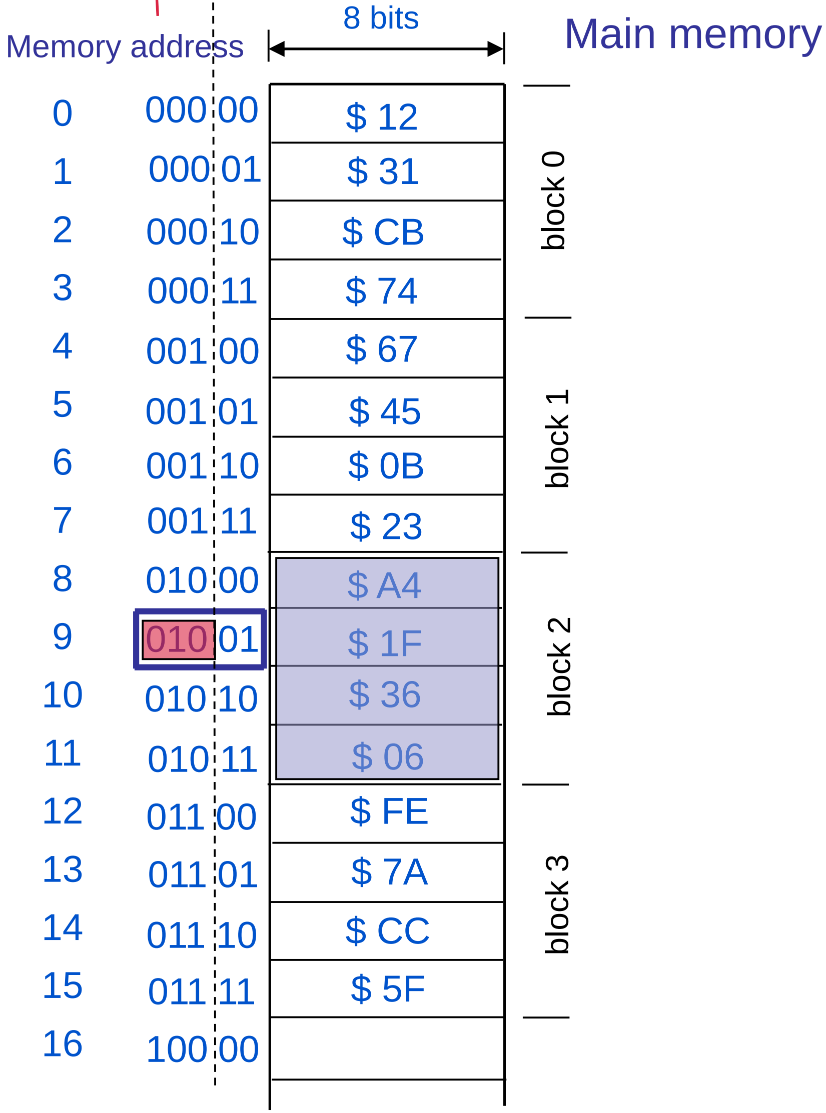
      the lower two bits of the memory address determine the word (byte) in the block (2^2 = 4)



CPU access to the memory address:

9, 10, 11, 2, 3, 9, 10, 11, 12,. . .

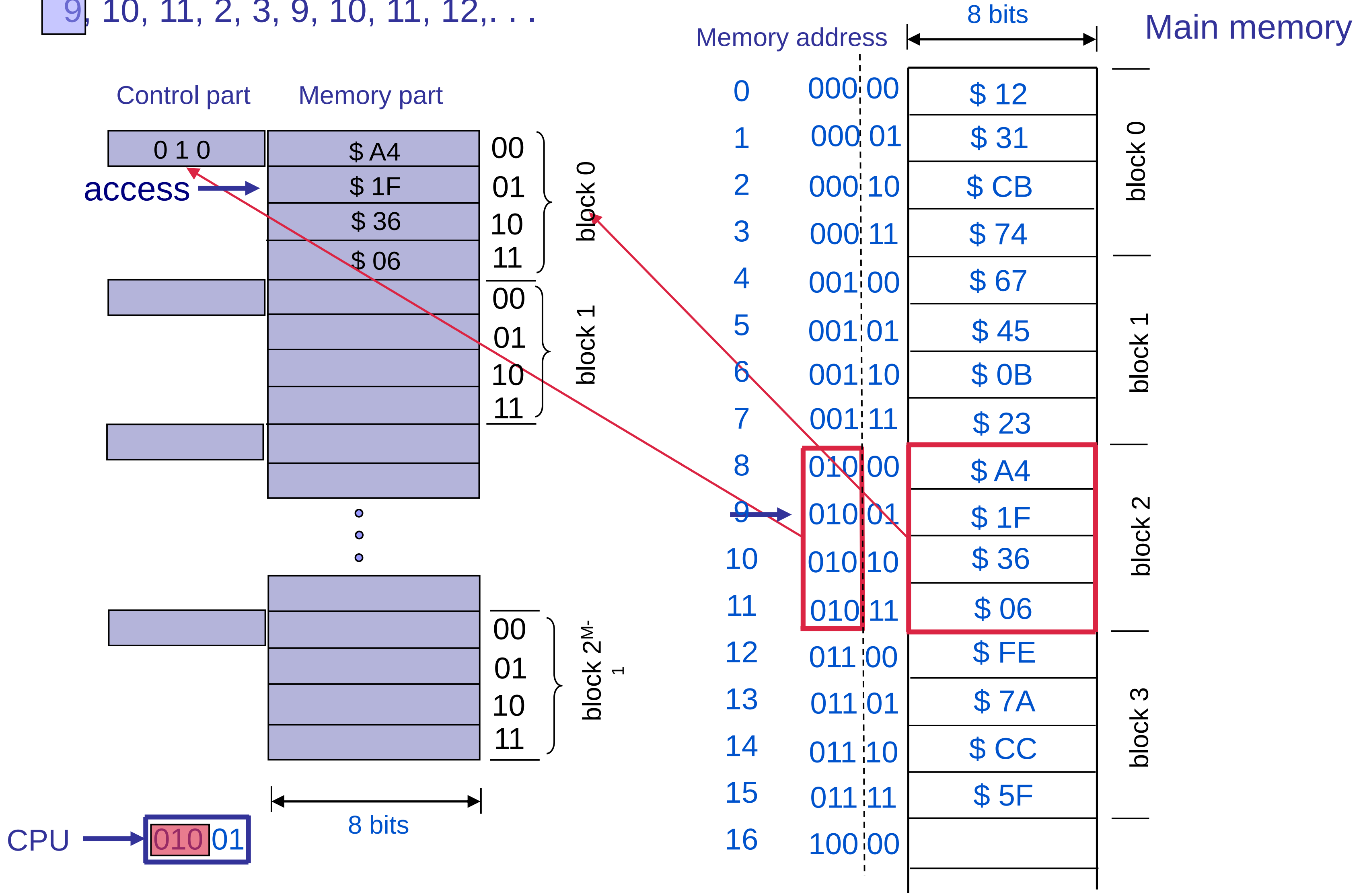




	Memory address		
0	000 00	$ 12	block 0
1	000 01	$ 31	
2	000 10	$ CB	
3	000 11	$ 74	
4	001 00	$ 67	block 1
5	001 01	$ 45	
6	001 10	$ 0B	
7	001 11	$ 23	
8	010 00	$ A4	block 2
9	010 01	$ 1F	
10	010 10	$ 36	
11	010 11	$ 06	
12	011 00	$ FE	block 3
13	011 01	$ 7A	
14	011 10	$ CC	
15	011 11	$ 5F	
16	100 00		

CPU access to the memory address:

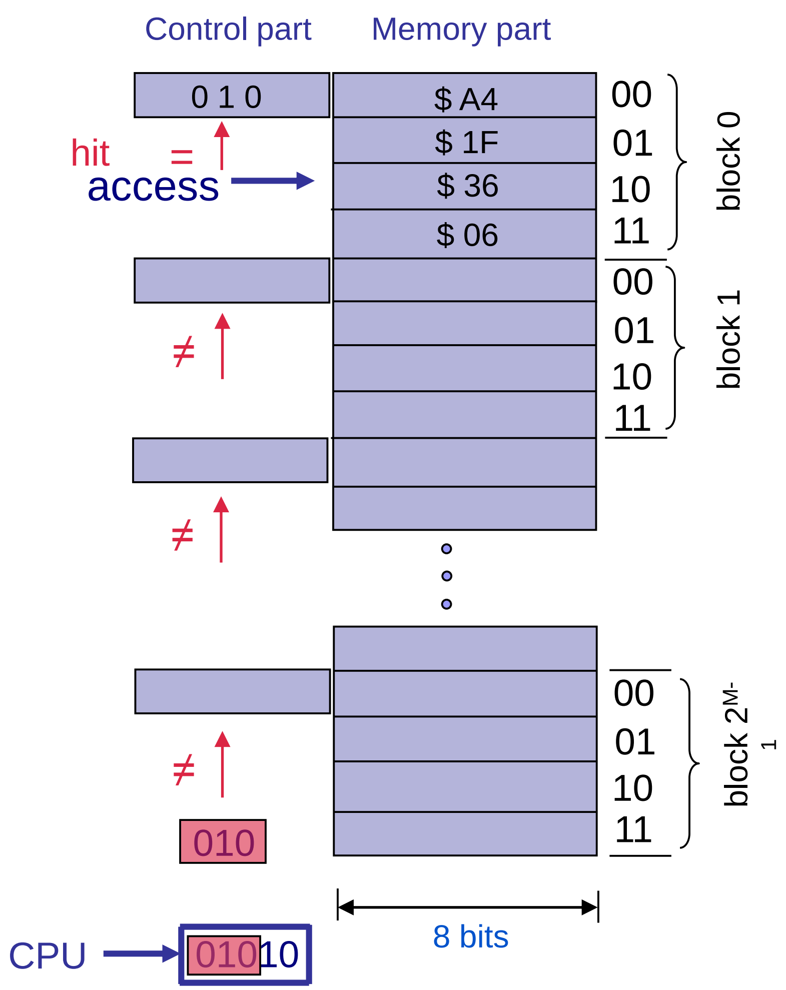
9, 10, 11, 2, 3, 9, 10, 11, 12,. . .

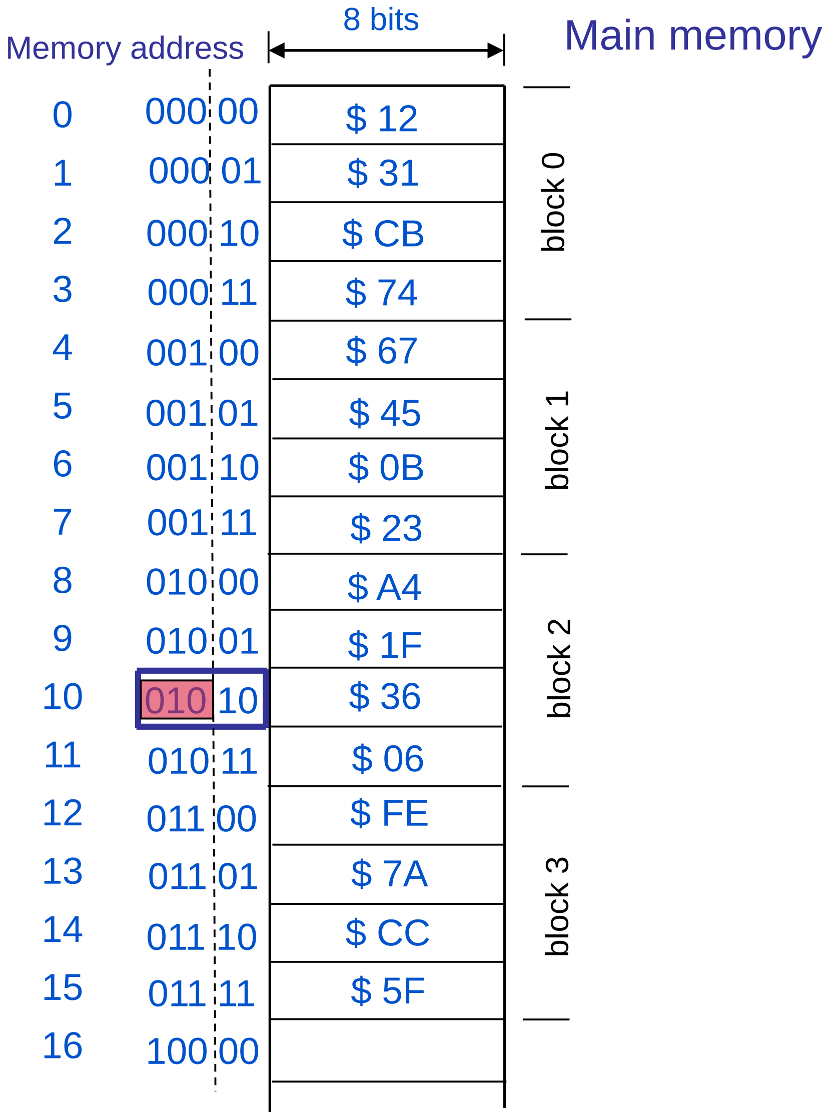


Memory address		Main memory
0	000 00	$ 12
1	000 01	$ 31
2	000 10	$ CB
3	000 11	$ 74
4	001 00	$ 67
5	001 01	$ 45
6	001 10	$ 0B
7	001 11	$ 23
8	010 00	$ A4
9	010 01	$ 1F
10	010 10	$ 36
11	010 11	$ 06
12	011 00	$ FE
13	011 01	$ 7A
14	011 10	$ CC
15	011 11	$ 5F
16	100 00	

CPU access to the memory address:

9, 10, 11, 2, 3, 9, 10, 11, 12,. . .



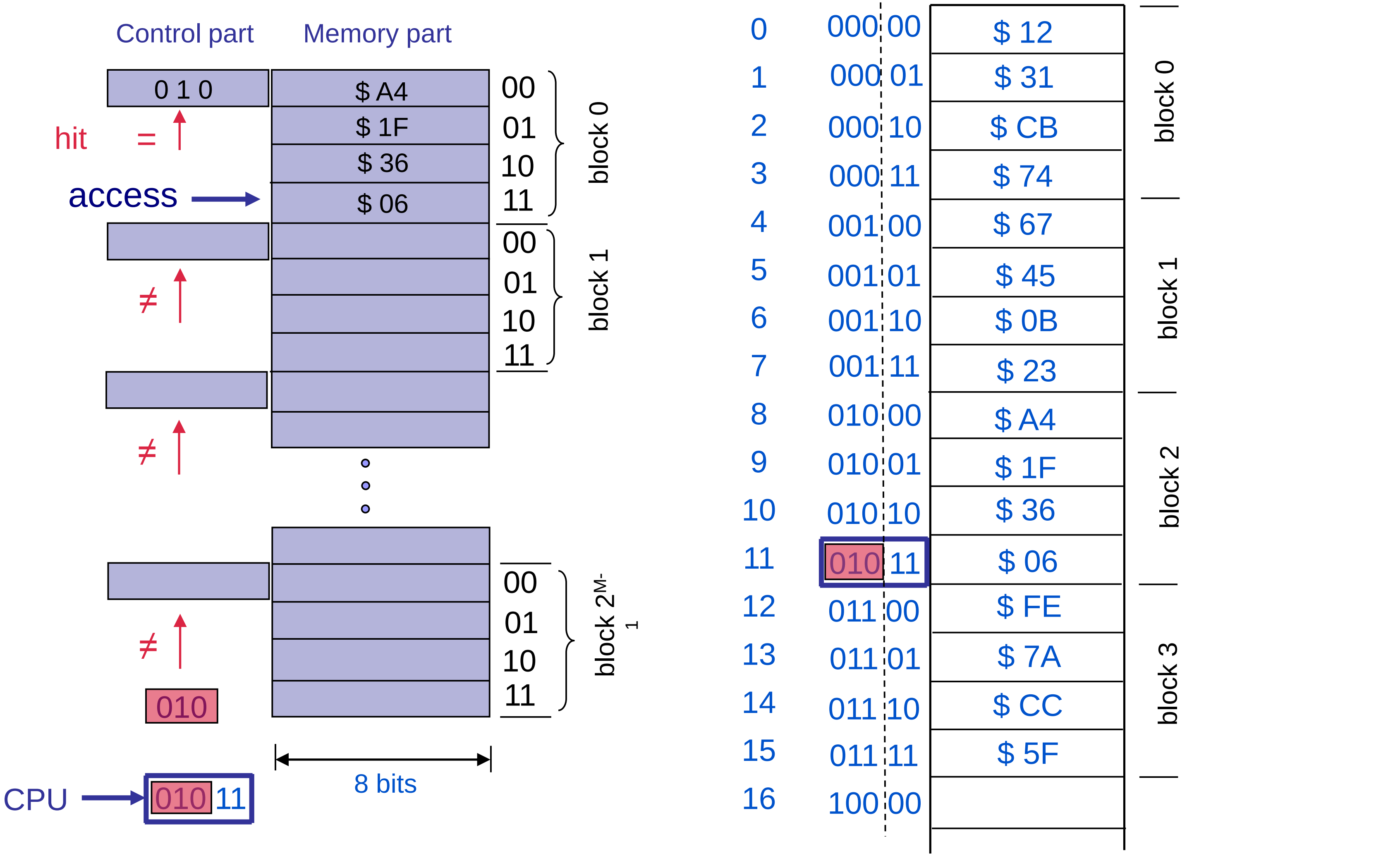


Main memory

Memory address		8 bits	
0	000 00	$ 12	block 0
1	000 01	$ 31	
2	000 10	$ CB	
3	000 11	$ 74	
4	001 00	$ 67	block 1
5	001 01	$ 45	
6	001 10	$ 0B	
7	001 11	$ 23	
8	010 00	$ A4	block 2
9	010 01	$ 1F	
10	010 10	$ 36	
11	010 11	$ 06	
12	011 00	$ FE	block 3
13	011 01	$ 7A	
14	011 10	$ CC	
15	011 11	$ 5F	
16	100 00		

CPU access to the memory address:

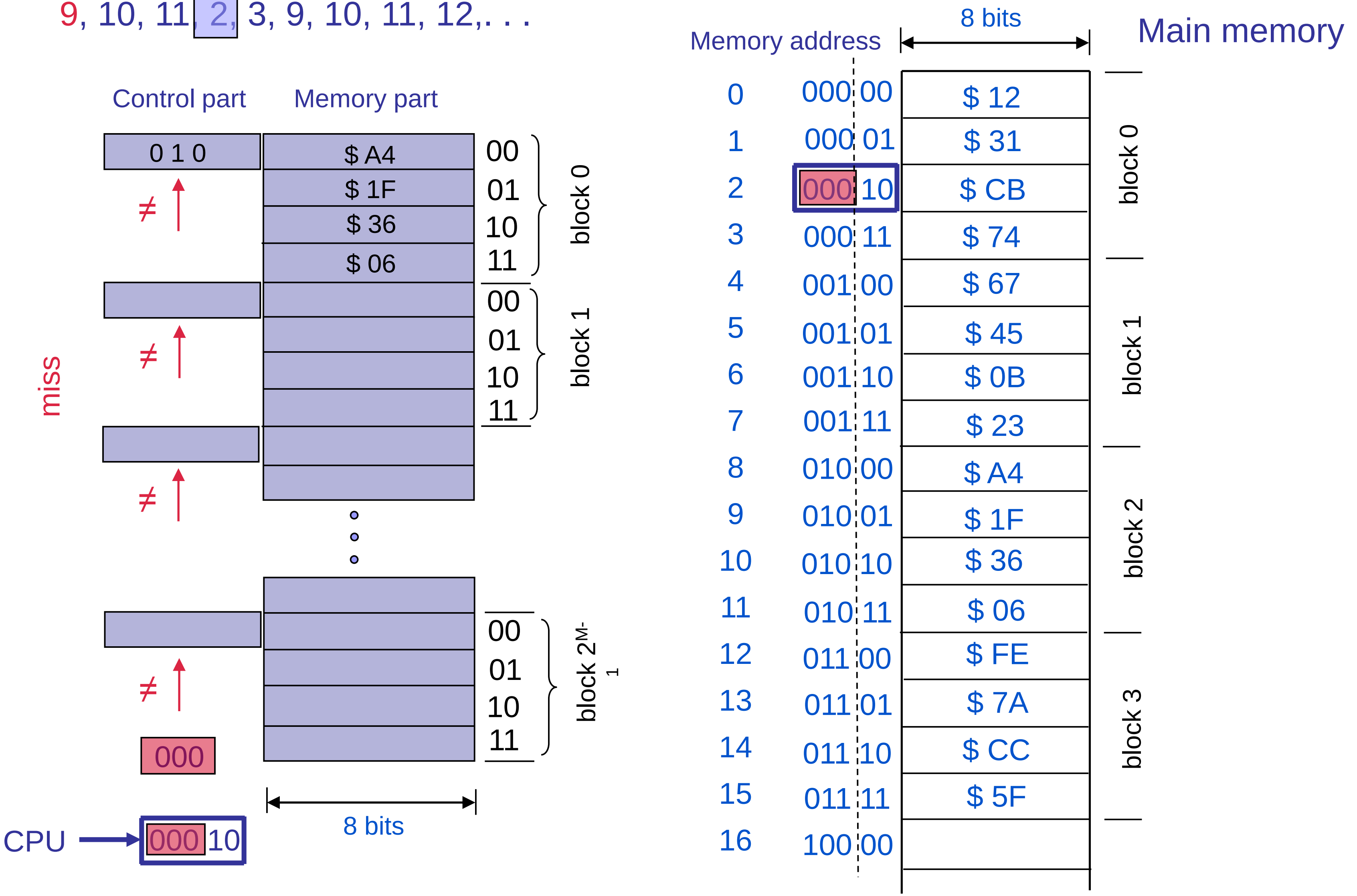
9, 10, 11, 2, 3, 9, 10, 11, 12,. . .



Memory address		8 bits
0	000 00	$ 12
1	000 01	$ 31
2	000 10	$ CB
3	000 11	$ 74
4	001 00	$ 67
5	001 01	$ 45
6	001 10	$ 0B
7	001 11	$ 23
8	010 00	$ A4
9	010 01	$ 1F
10	010 10	$ 36
11	010 11	$ 06
12	011 00	$ FE
13	011 01	$ 7A
14	011 10	$ CC
15	011 11	$ 5F
16	100 00	

CPU access to the memory address:

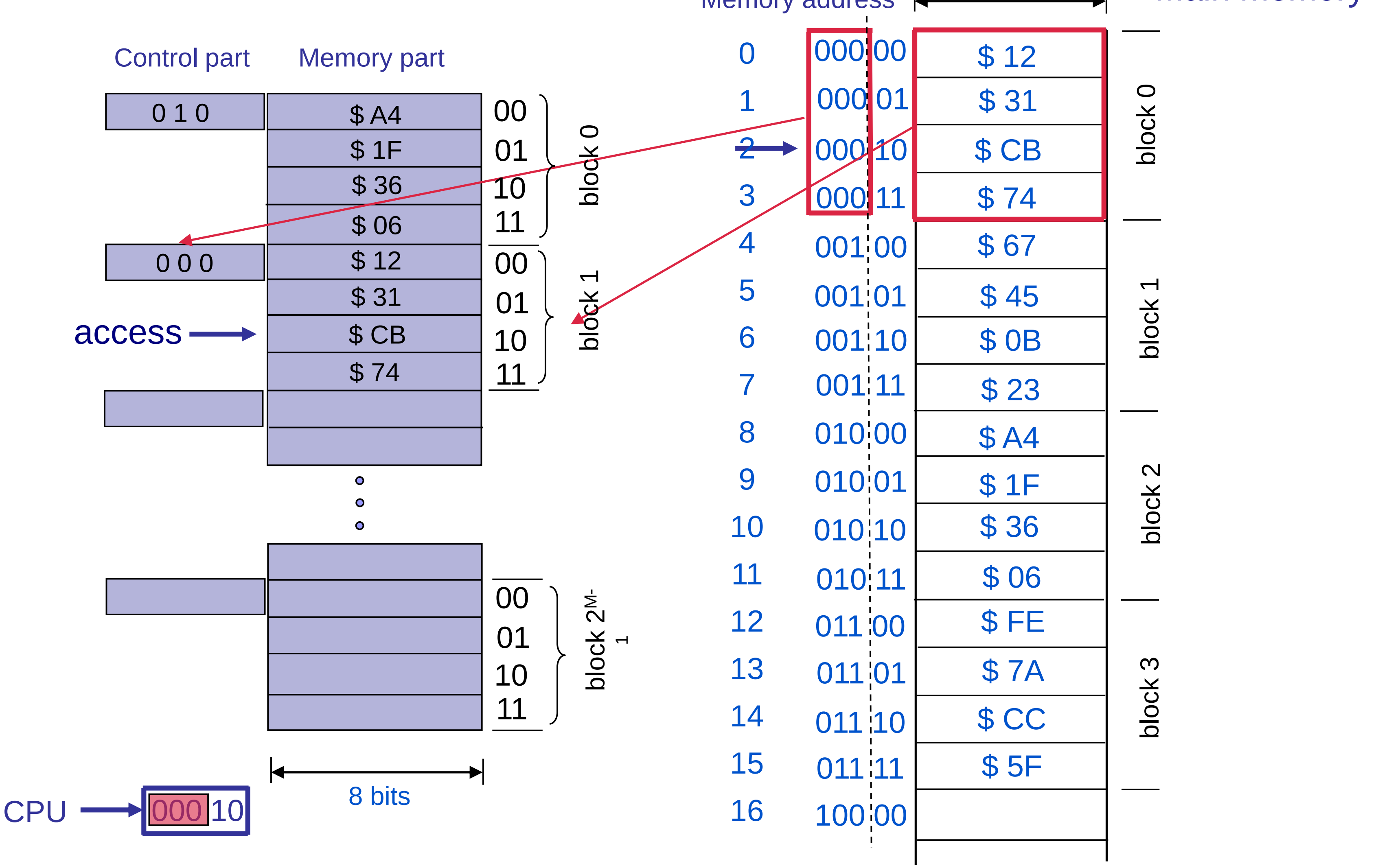
9, 10, 11, 2, 3, 9, 10, 11, 12,. . .



Memory address		Main memory
0	000 00	$ 12
1	000 01	$ 31
2	000 10	$ CB
3	000 11	$ 74
4	001 00	$ 67
5	001 01	$ 45
6	001 10	$ 0B
7	001 11	$ 23
8	010 00	$ A4
9	010 01	$ 1F
10	010 10	$ 36
11	010 11	$ 06
12	011 00	$ FE
13	011 01	$ 7A
14	011 10	$ CC
15	011 11	$ 5F
16	100 00	

CPU access to the memory address:

9, 10, 11, 2, 3, 9, 10, 11, 12,. . .

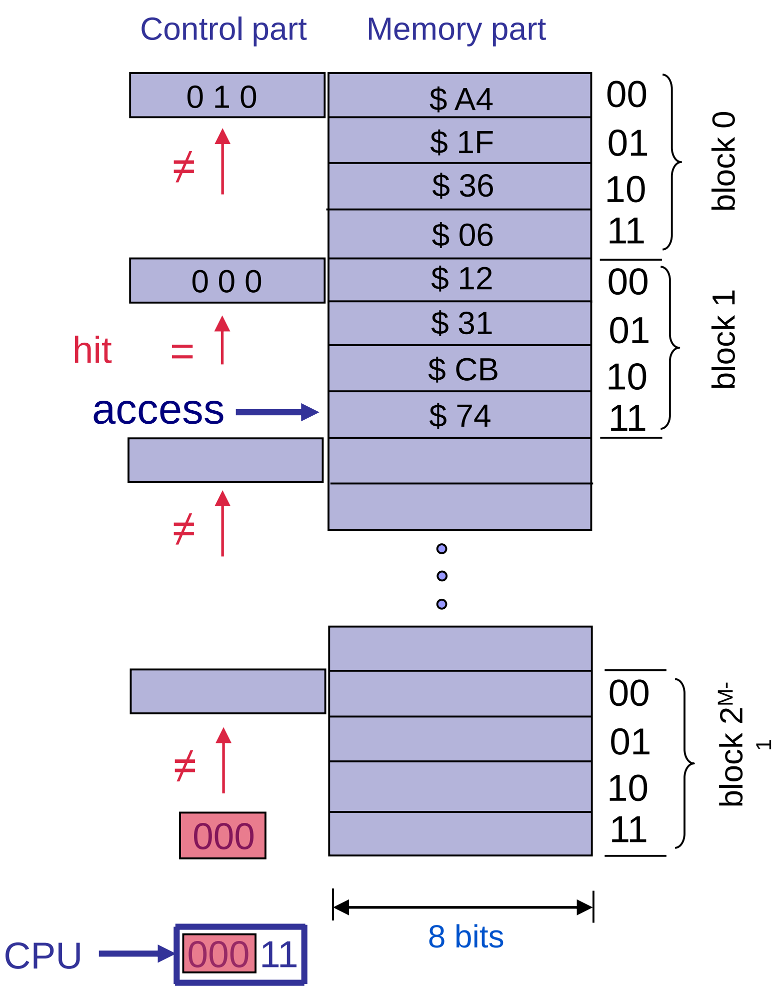


Control part	Memory part		
0 1 0	$ A4	00	block 0
	$ 1F	01	
	$ 36	10	
	$ 06	11	
0 0 0	$ 12	00	block 1
	$ 31	01	
	$ CB	10	
	$ 74	11	
		00	block 2^{M-1}
		01	
		10	
		11	

Memory address		Main memory	
0	000 00	$ 12	block 0
1	000 01	$ 31	
2	000 10	$ CB	
3	000 11	$ 74	
4	001 00	$ 67	block 1
5	001 01	$ 45	
6	001 10	$ 0B	
7	001 11	$ 23	
8	010 00	$ A4	block 2
9	010 01	$ 1F	
10	010 10	$ 36	
11	010 11	$ 06	
12	011 00	$ FE	block 3
13	011 01	$ 7A	
14	011 10	$ CC	
15	011 11	$ 5F	
16	100 00		

CPU access to the memory address:

9, 10, 11, 2, 3, 9, 10, 11, 12,. . .



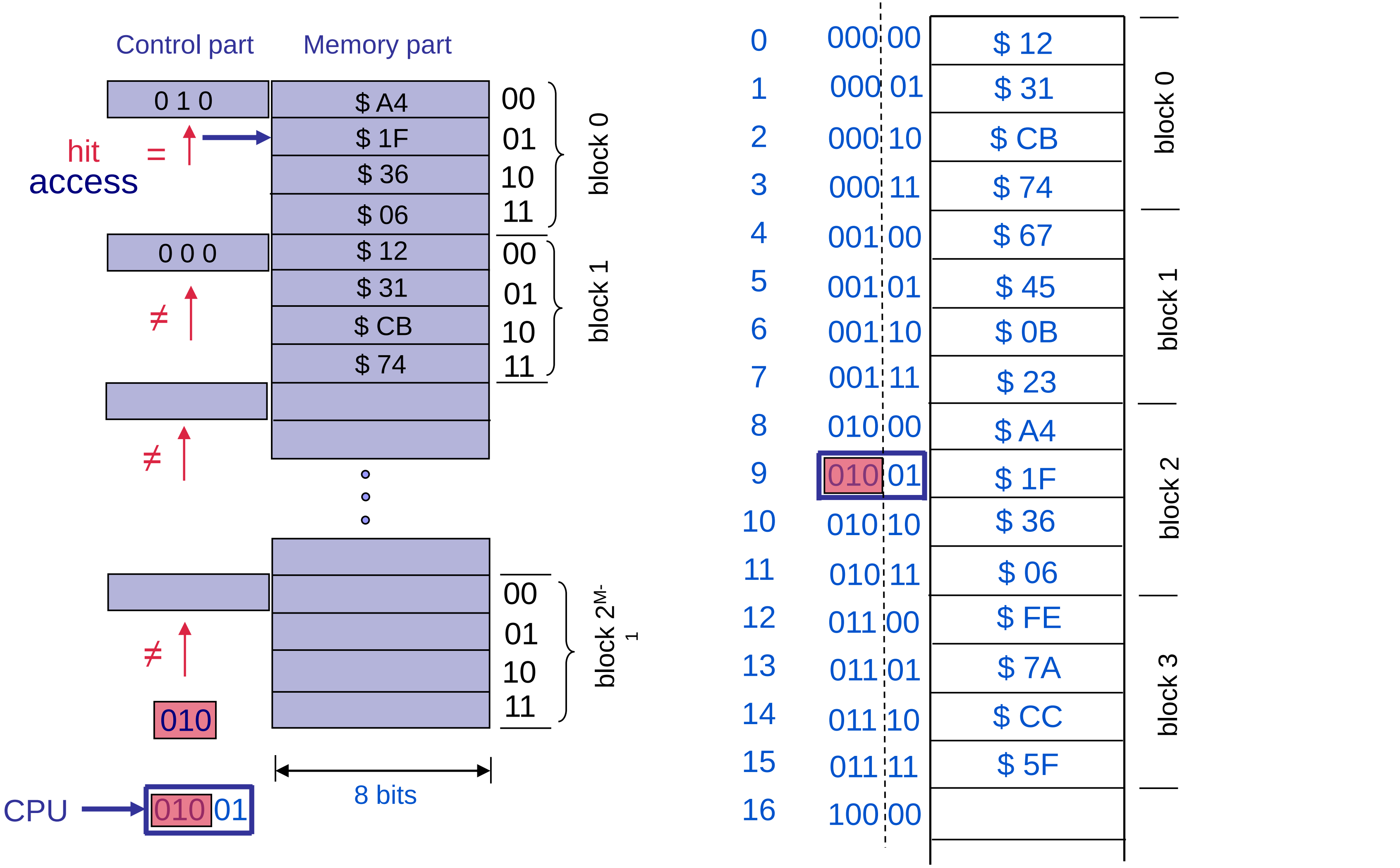
Main memory

8 bits

Memory address			
0	000 00	$ 12	block 0
1	000 01	$ 31	
2	000 10	$ CB	
3	000 11	$ 74	
4	001 00	$ 67	block 1
5	001 01	$ 45	
6	001 10	$ 0B	
7	001 11	$ 23	
8	010 00	$ A4	block 2
9	010 01	$ 1F	
10	010 10	$ 36	
11	010 11	$ 06	
12	011 00	$ FE	block 3
13	011 01	$ 7A	
14	011 10	$ CC	
15	011 11	$ 5F	
16	100 00		

CPU access to the memory address:

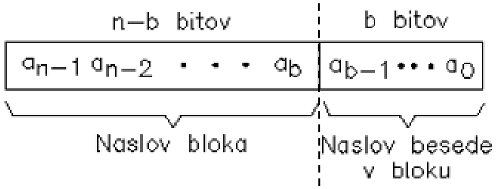
9, 10, 11, 2, 3, 9, 10, 11, 12,. . .



Control part	Memory part		
0 1 0	$ A4	00	block 0
	$ 1F	01	
	$ 36	10	
	$ 06	11	
0 0 0	$ 12	00	block 1
	$ 31	01	
	$ CB	10	
	$ 74	11	
		00	block 2^{M-1}
		01	
		10	
		11	

Main memory

Memory address		8 bits	
0	000 00	$ 12	block 0
1	000 01	$ 31	
2	000 10	$ CB	
3	000 11	$ 74	
4	001 00	$ 67	block 1
5	001 01	$ 45	
6	001 10	$ 0B	
7	001 11	$ 23	
8	010 00	$ A4	block 2
9	010 01	$ 1F	
10	010 10	$ 36	
11	010 11	$ 06	
12	011 00	$ FE	block 3
13	011 01	$ 7A	
14	011 10	$ CC	
15	011 11	$ 5F	
16	100 00		



CPU access to the memory address:

9, 10, 11, 2, 3, 9, 10, 11, 2,. . .

↑ of missing ↑

Control part	Memory part		
0 1 0	$ A4	00	block 0
	$ 1F	01	
	$ 36	10	
	$ 06	11	
0 0 0	$ 12	00	block 1
	$ 31	01	
	$ CB	10	
	$ 74	11	
	⋮		
		00	block 2^{M-1}
		01	
		10	
		11	

8 bits

Main memory

Memory address		8 bits	
0	000 00	$ 12	block 0
1	000 01	$ 31	
2	000 10	$ CB	
3	000 11	$ 74	
4	001 00	$ 67	block 1
5	001 01	$ 45	
6	001 10	$ 0B	
7	001 11	$ 23	
8	010 00	$ A4	block 2
9	010 01	$ 1F	
10	010 10	$ 36	
11	010 11	$ 06	
12	011 00	$ FE	block 3
13	011 01	$ 7A	
14	011 10	$ CC	
15	011 11	$ 5F	
16	100 00		

Cache - principles

Access to an operand in memory:

- HIT in cache (probability H):
  - CPU accesses to the operand in cache (read or write)
- MISS in cache (probability of 1-H) :
  - Transfer of the block from main memory to cache or
  - Block replacement - if the cache is full, one of the blocks stored in the cache is saved back to main memory (is this always necessary?), on its location a new block from main memory is transferred.
  - CPU accesses to the operand in cache

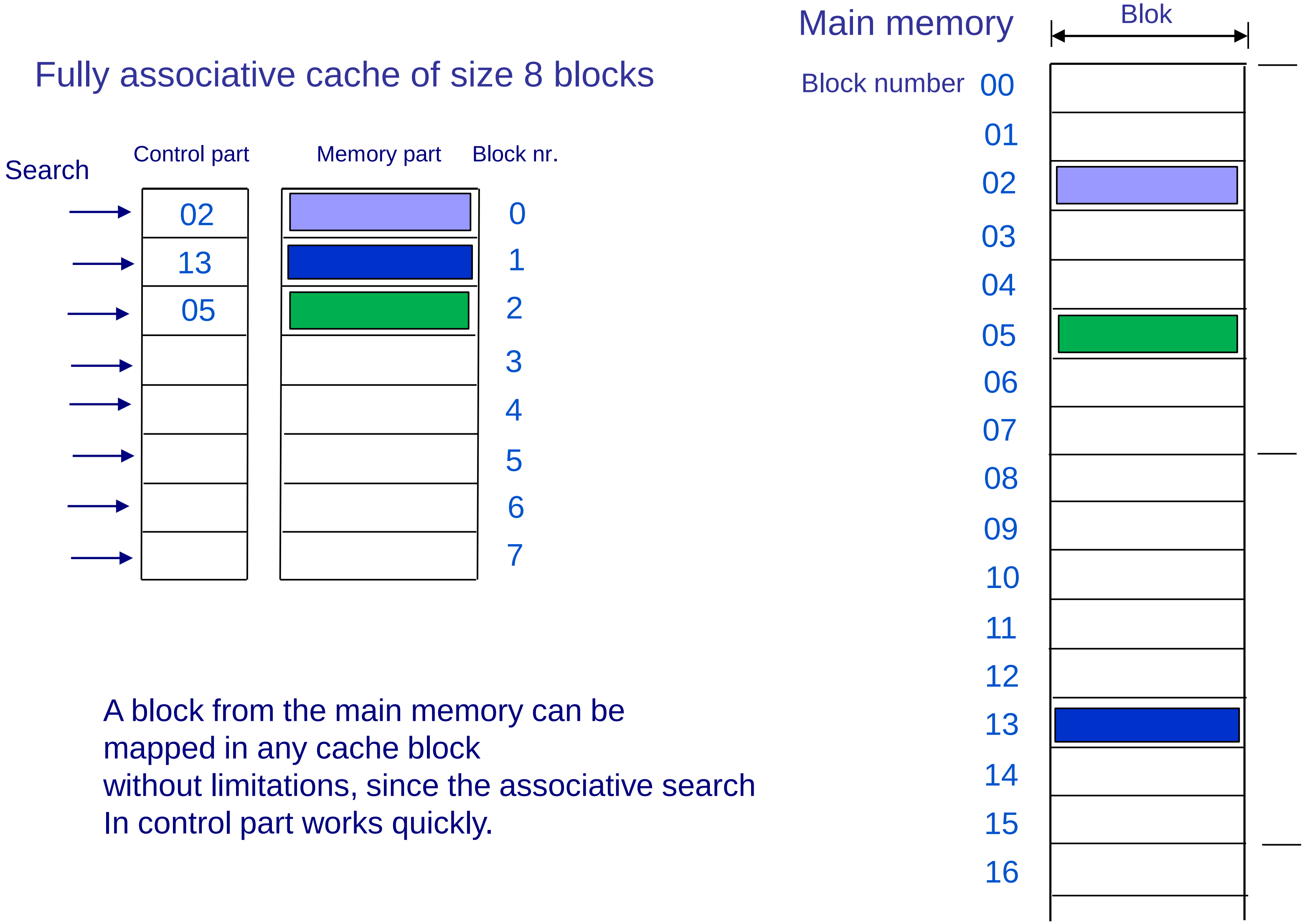
Types of caches according to restrictions on the mapping of blocks

- Search for the block in cache must be fast.
- If this is not possible, it is necessary to introduce restrictions on mapping a block from main memory to cache.
- Depending on the severity of restrictions on the mapping, we distinguish three types of caches:
  - Associative cache
    - (No restrictions on the mapping of blocks in the cache)
  - Set-associative cache
    - (Block can be mapped only to a specific set, but within the set without limitation)
  - Direct cache
    - (Block can be mapped only to a specific block frame)

- Fully associative cache
  - Memory part of the cache is a static RAM (true for all three types of caches)
  - Control part of the cache is associative memory, which allows fast search for a block number across whole control part of the cache
  - Since the search is fast across whole cache, the block can be mapped to cache anywhere in any block frame.
  - Because of today's technology associative memory size is limited, fully associative caches are rare and large only a few 100 blocks.
  - If we want a large cache, the solution is another type of cache - set-associative or direct.

Mapping block in fully associative cache

Fully associative cache of size 8 blocks

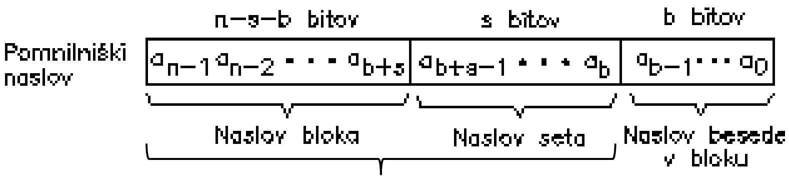


A block from the main memory can be mapped in any cache block without limitations, since the associative search In control part works quickly.

Set-associative cache

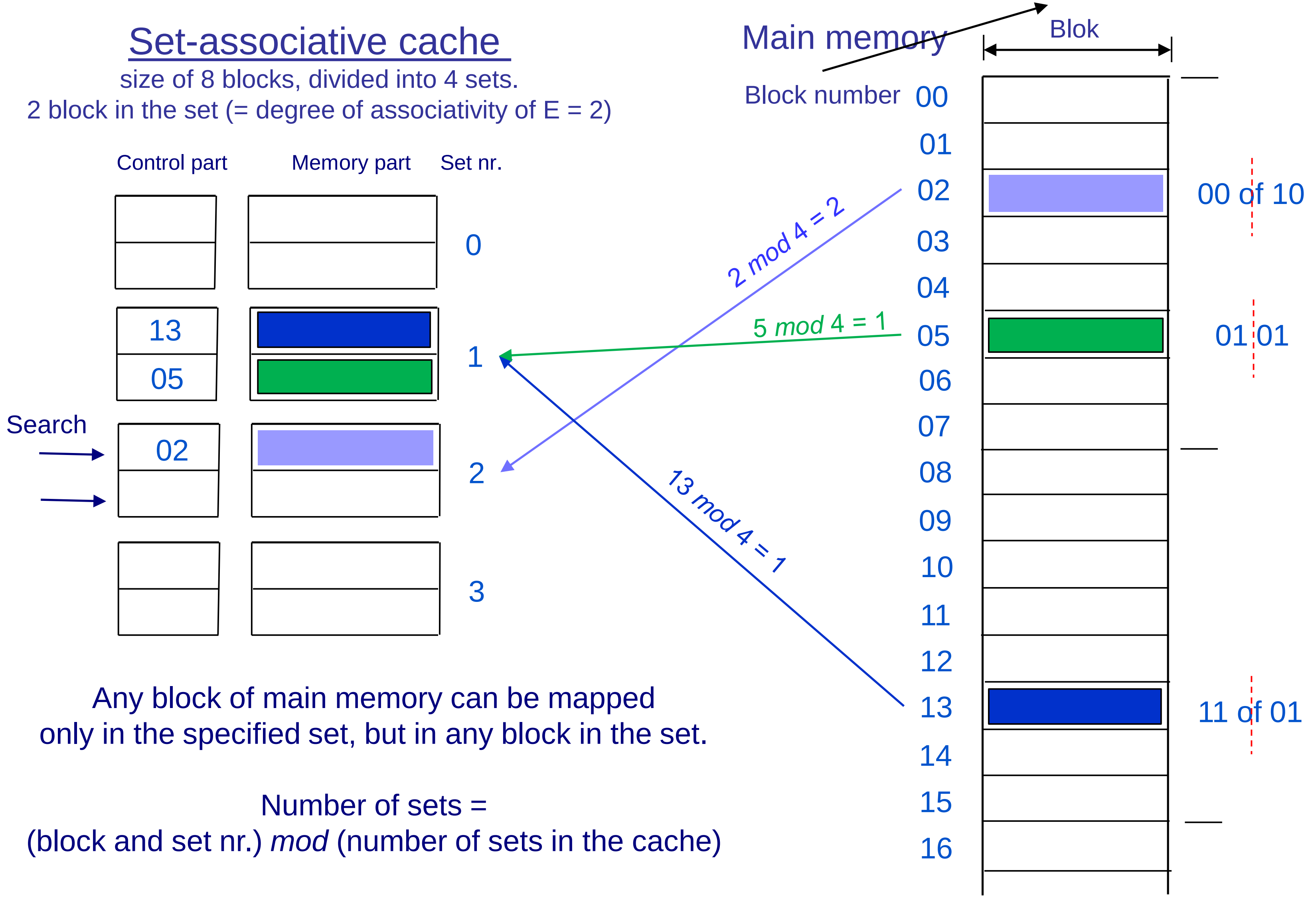
- The entire cache is divided into several parts - called sets.
- Each set has a smaller associative cache.
- Search for block within the set is fast (associative control part), search for the block in different sets is much slower
- Therefore, a certain block of main memory can be mapped only in the specified set (it is not necessary to search between sets), but within the set can be mapped anywhere.
- The number of blocks in the set is called the associativity E.
- The higher the degree of associativity, the higher is hit rate.

Mapping block in set-associative cache



Set-associative cache

size of 8 blocks, divided into 4 sets.
2 block in the set (= degree of associativity of E = 2)



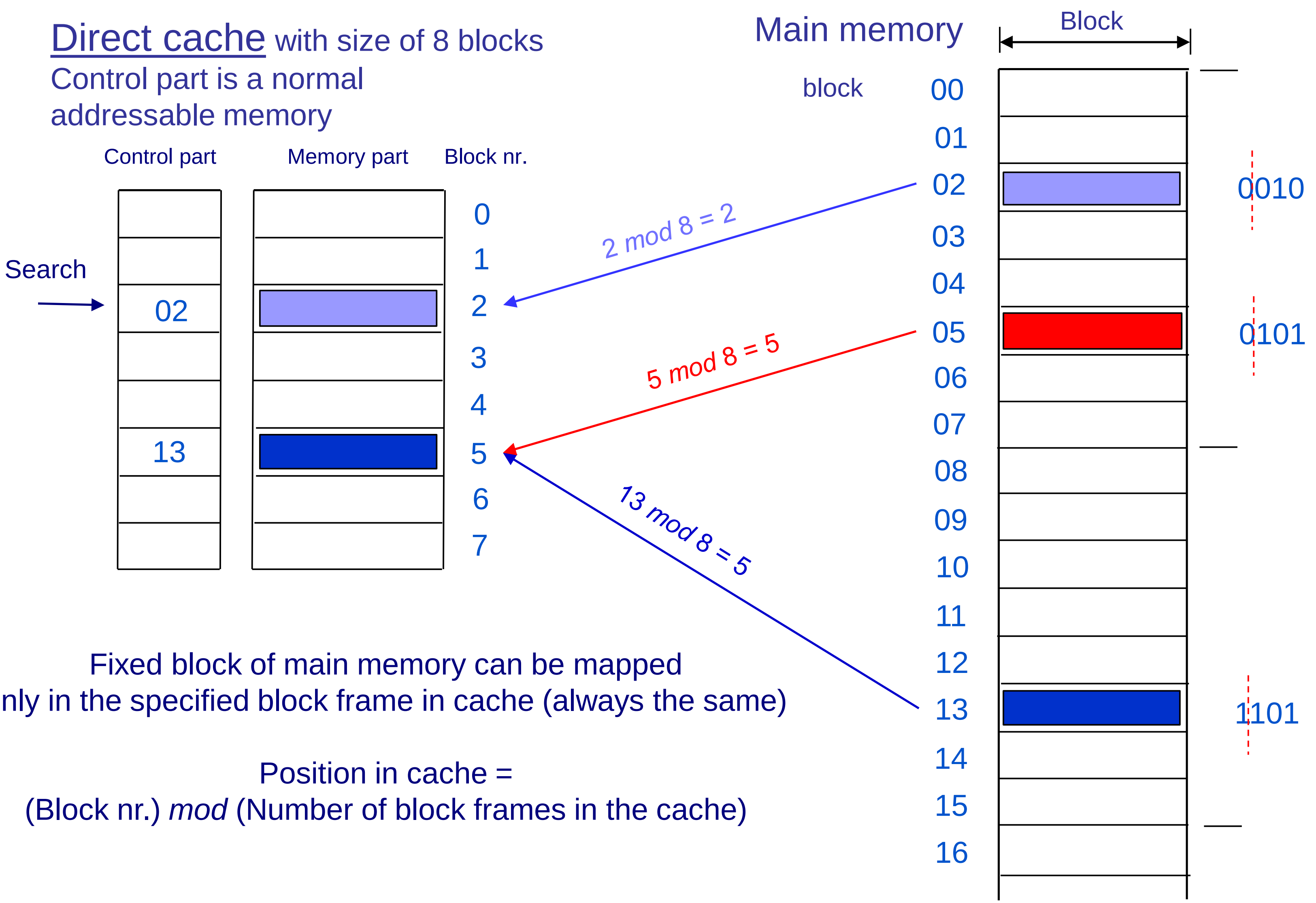
Any block of main memory can be mapped
only in the specified set, but in any block in the set.

Number of sets =
(block and set nr.) *mod* (number of sets in the cache)

Cache - principles

- Direct cache
  - The entire control part of the cache is usual addressable memory - static RAM
  - Therefore, it is impossible to do a fast block search (it would be too slow).
  - Certain block of main memory can therefore be mapped only in the specific block frame in cache (so search is not necessary anymore)
  - If the block frame, into which a new block from memory must be mapped, is full, it is necessary to replace the block.
  - Hit rate is therefore in direct cache compared with the set-associative cache of same size, much smaller ..

Mapping block in direct cache

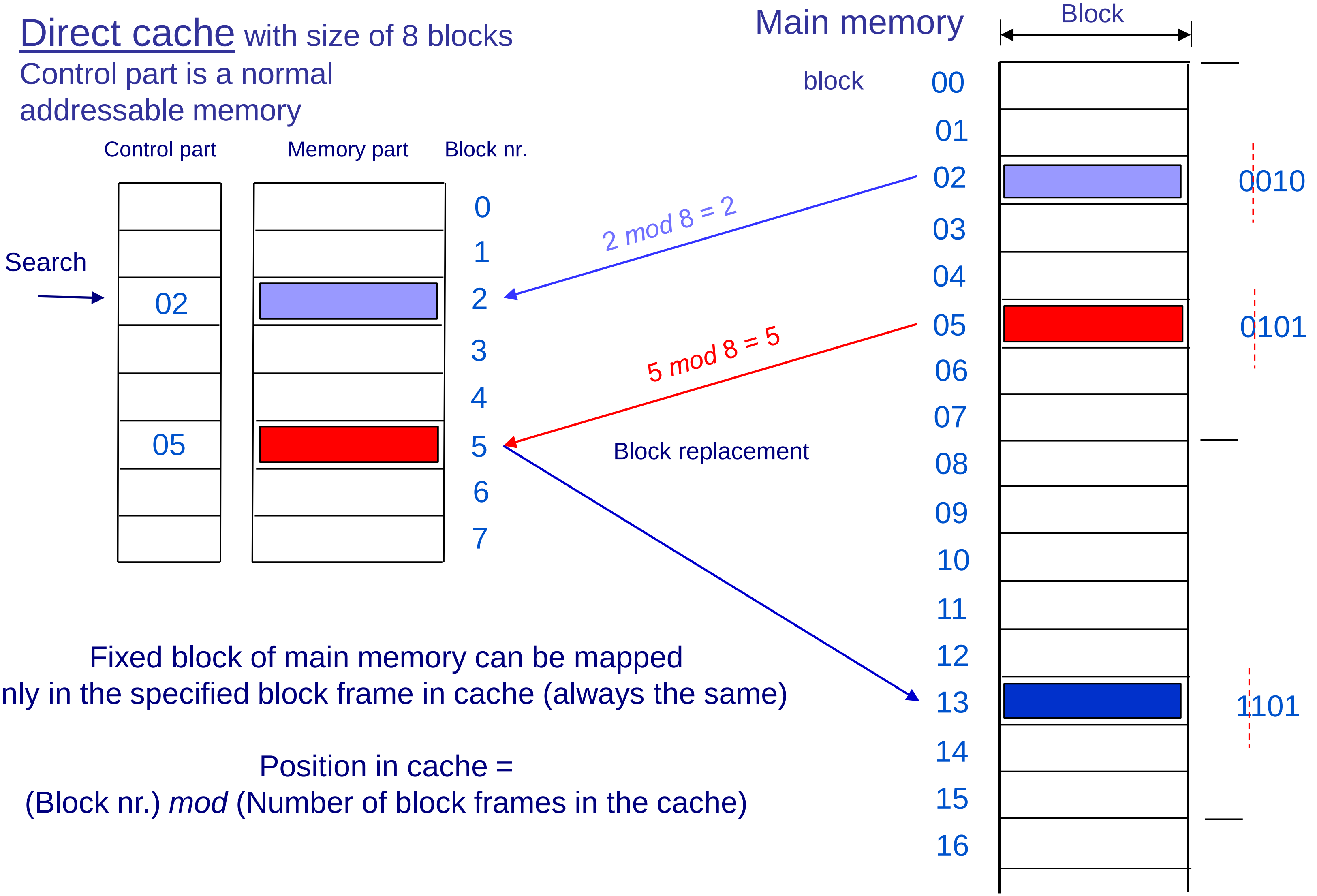


Fixed block of main memory can be mapped
only in the specified block frame in cache (always the same)

Position in cache =
(Block nr.) *mod* (Number of block frames in the cache)

Mapping block in direct cache

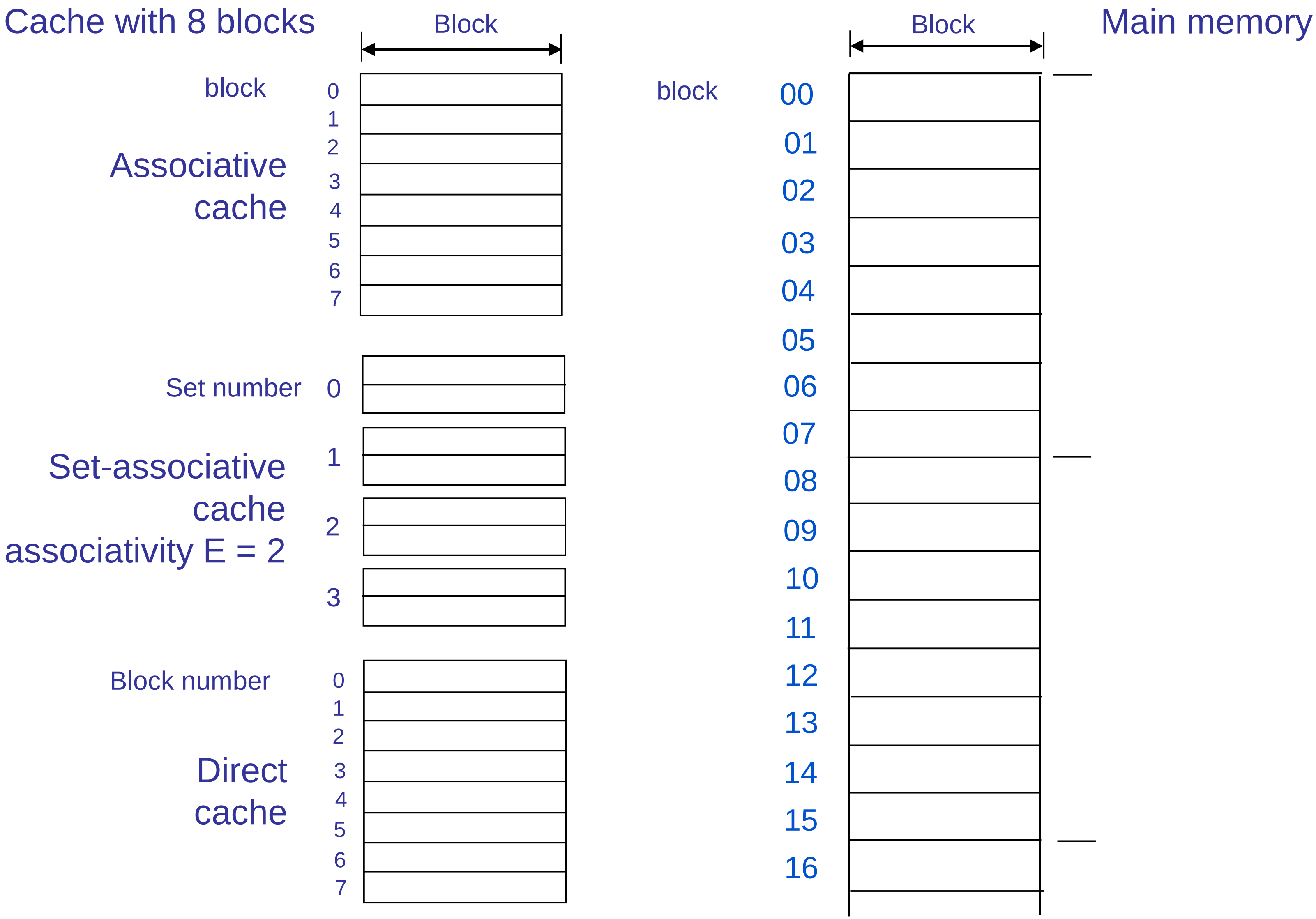
Direct cache with size of 8 blocks
Control part is a normal addressable memory



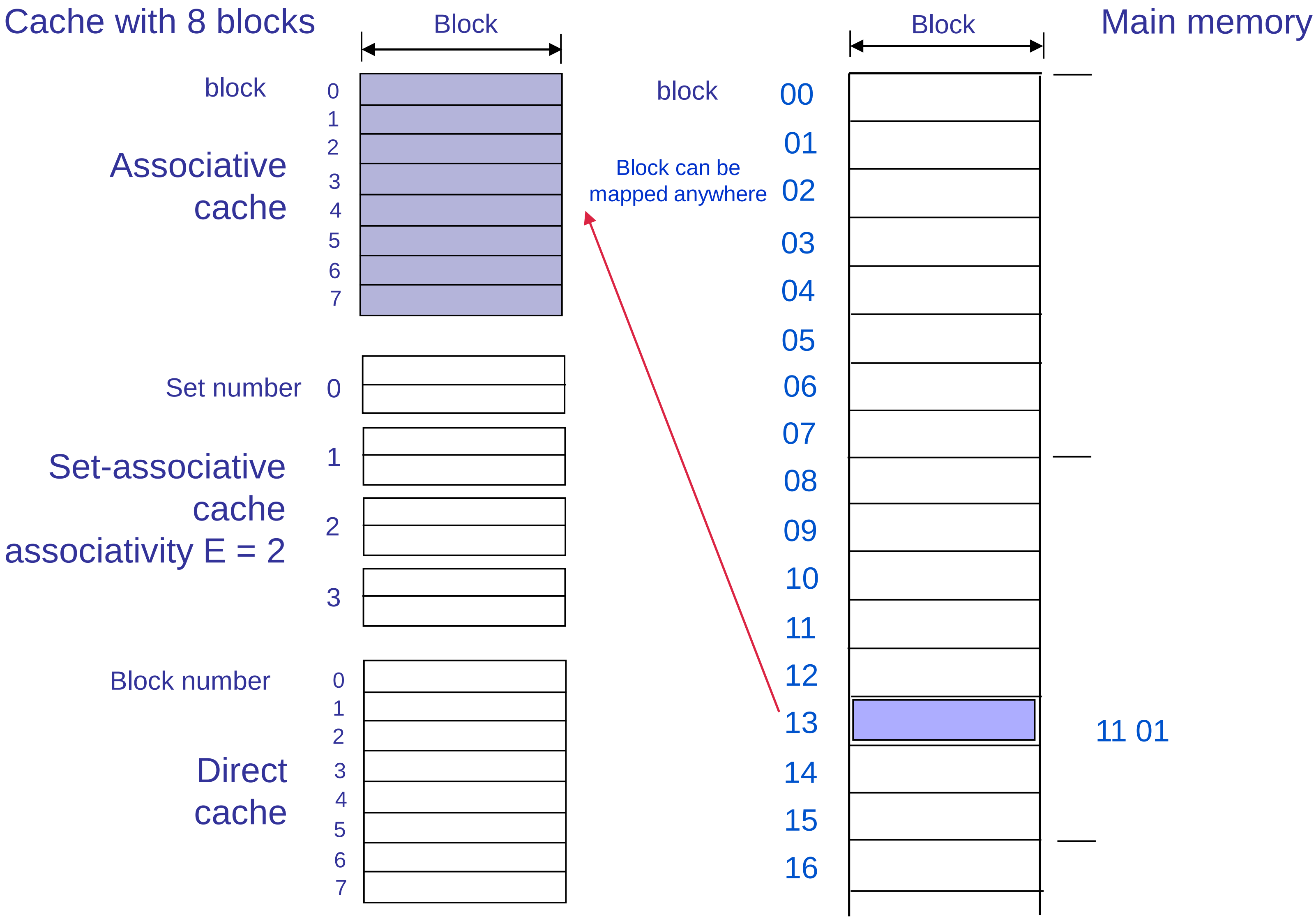
Fixed block of main memory can be mapped
only in the specified block frame in cache (always the same)

Position in cache =
(Block nr.) *mod* (Number of block frames in the cache)

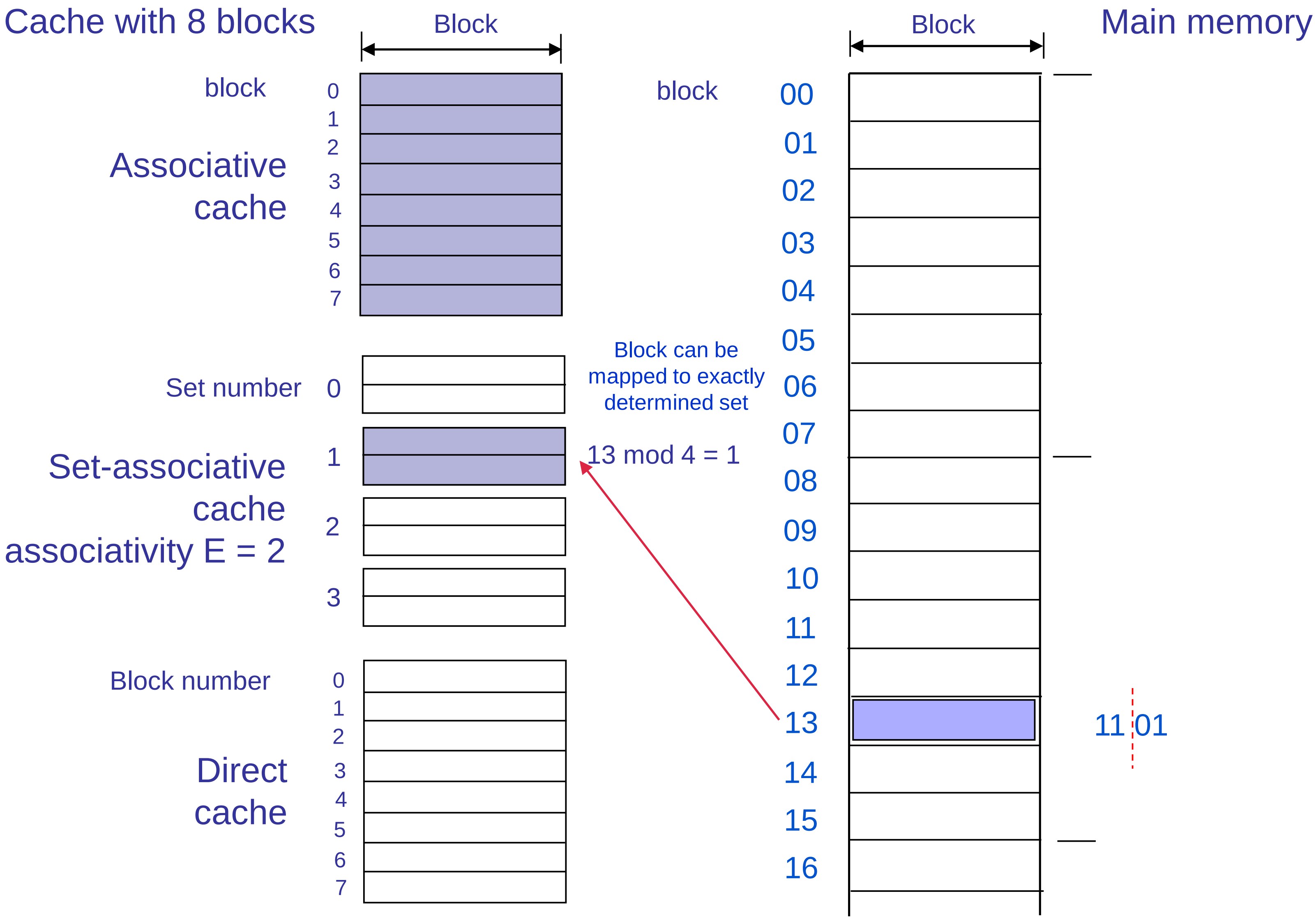
Cache - mapping block in the cache for different types of caches



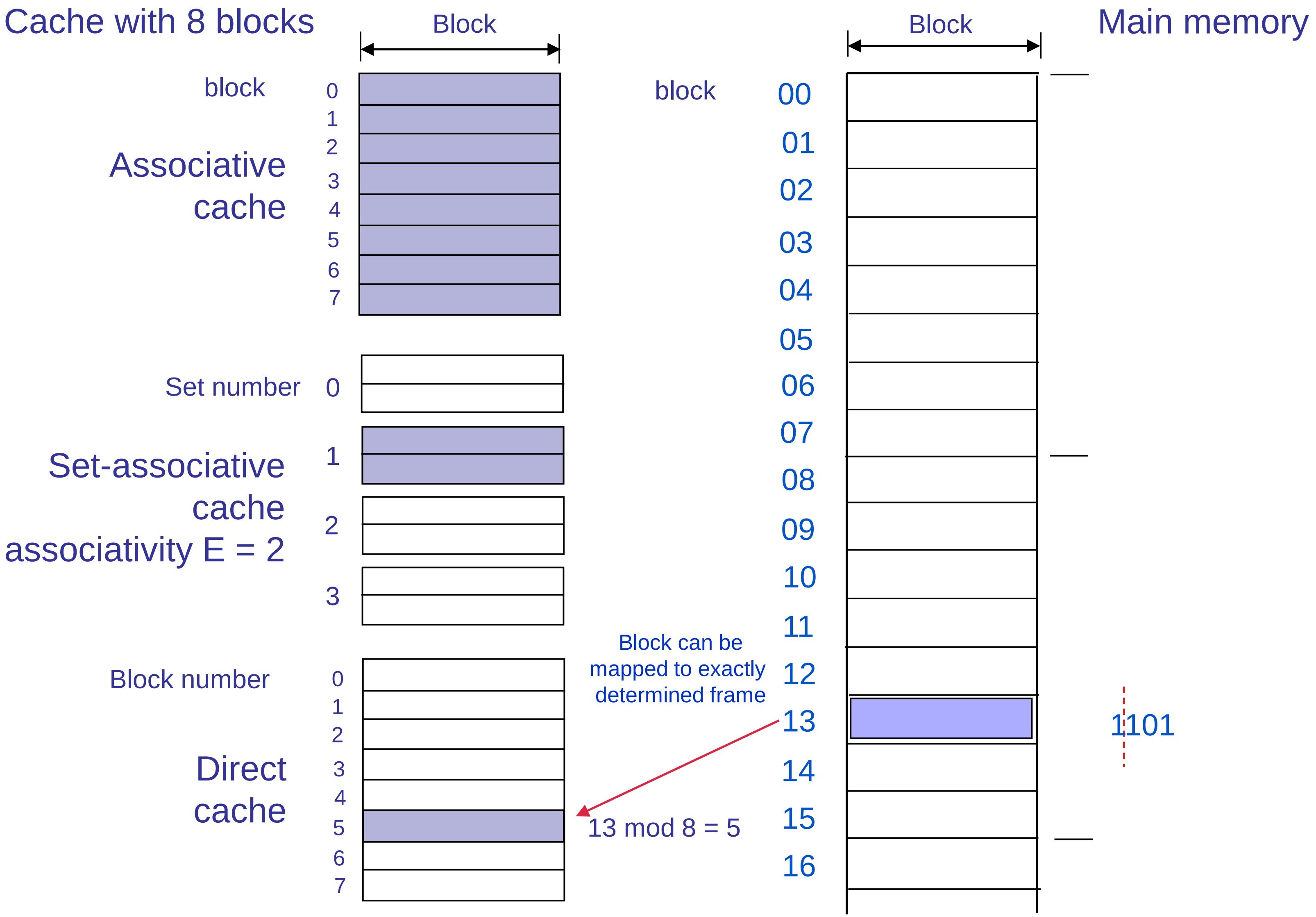
Cache - mapping block in the cache for different types of caches



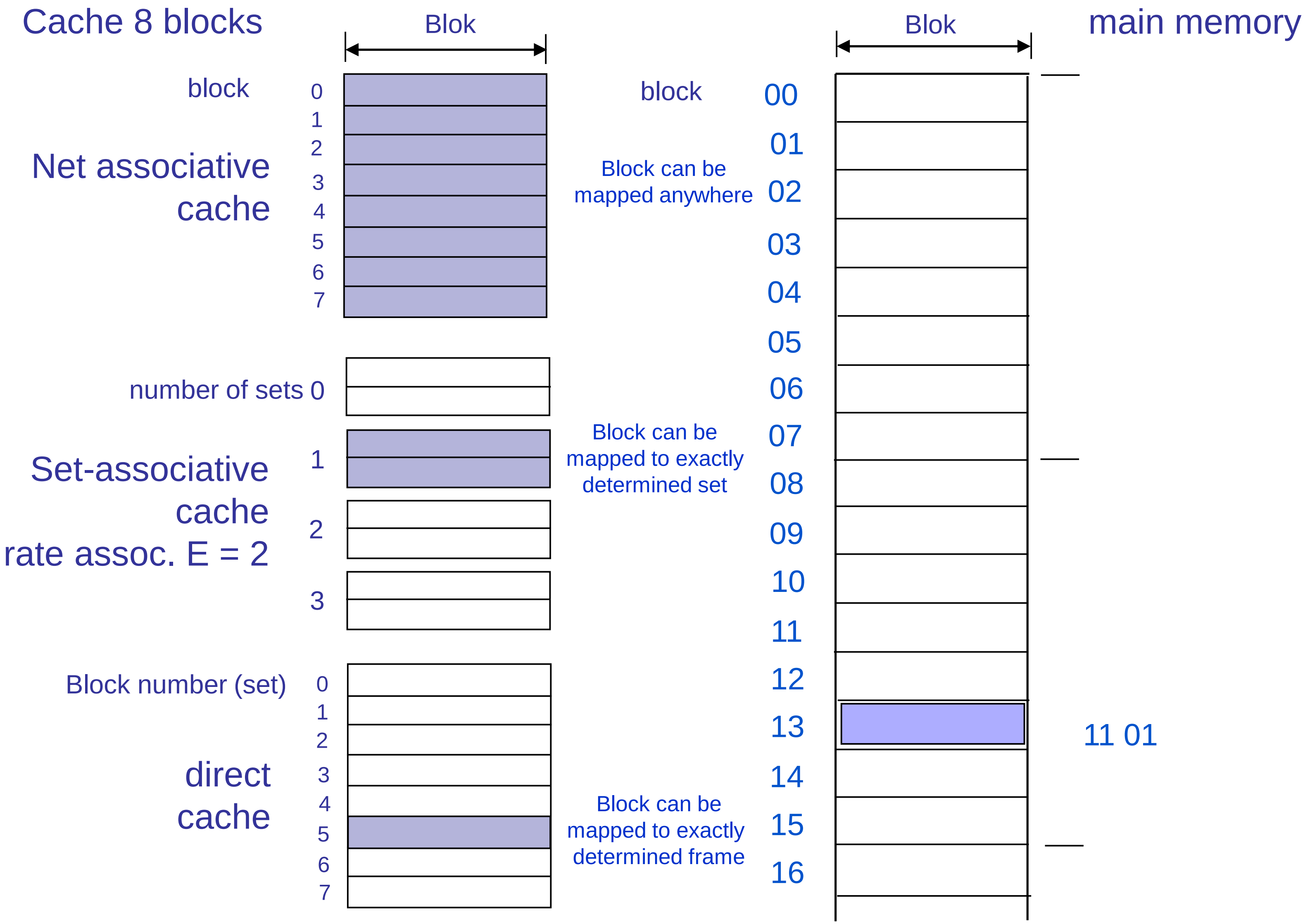
Cache - mapping block in the cache for different types of caches



Cache - mapping block in the cache for different types of caches



cache - restrictions on the mapping block in the cache



Cache 8 blocks

Blok

block 0 1 2 3 4 5 6 7

Net associative cache

Block can be mapped anywhere

number of sets 0 1 2 3

Set-associative cache
rate assoc. E = 2

Block can be mapped to exactly determined set

Block number (set) 0 1 2 3 4 5 6 7

direct cache

Block can be mapped to exactly determined frame

Blok

main memory

block 00 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15 16

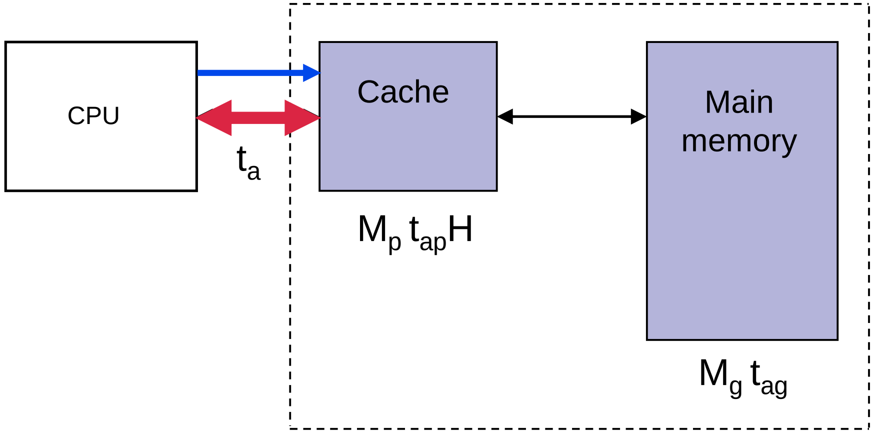
11 01

Impact of cache to the speed of CPU

- Access to the cache:
  - HIT:
    - read - usually 1 clock period,
    - write - read block,

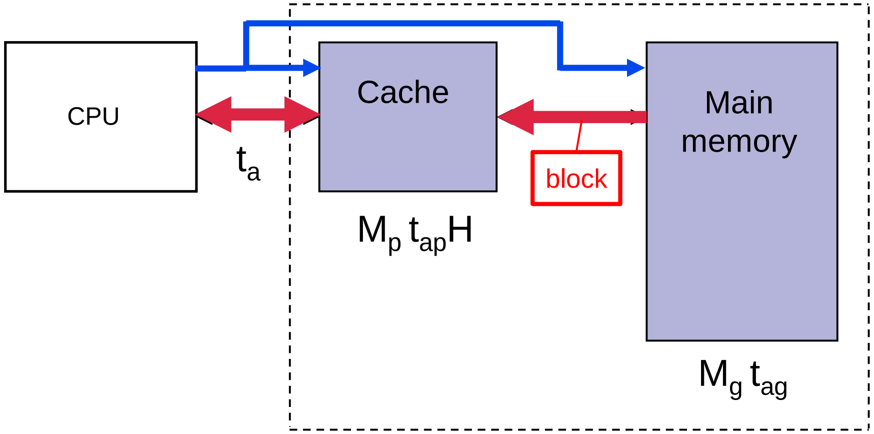
      - change the content,

      - write block back - typically a clock period more.



Cache - Cache impact on CPU speed

- Zgrešitev:
  - access to the main memory,
  - transfer of block to the cache,
  - write block in the cache,
  - followed by reading or writing as in case of hit,
  - if the cache is full, it is necessary to replace the block.



For all these operations in case of miss, it takes from 10 to 100 clock periods (miss penalty).

Cache - Cache impact on CPU speed

- Misses in cache reduce the operating speed of the CPU, i.e. they increase the CPI.
- Ideal CPI (CPI_I) – disregarding misses in cache
- Real CPI (CPI_R) – including misses in cache

Cache - Cache impact on CPU speed

- Real CPI with respect zgrešitev in the cache:

$$CPI_R = CPI_I + M_I(1 - H) * Miss\_penalty$$

CPI_R - real CPI

CPI_I - ideal CPI (excluding misses in cache)

M_I - average number of memory accesses per instruction

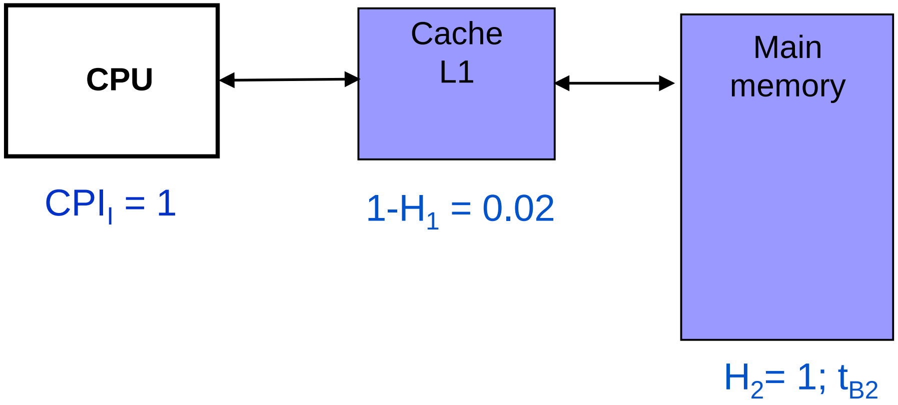
- Real time of execution of the program with *N* instructions is:

$$CPU_{time} = N * CPI_R * t_{CPE}$$

Example: Effect of L2 cache to the CPU speed

- Processor has ideal CPI_I = 1, there are no misses in instruction cache L1
- Clock frequency of the processor f_{CPE} = 4 GHz
- Probability of miss in the L1 cache is 2%
- Miss penalty is 100 ns (time to transfer the block from the main memory)
- If we add L2 cache to a hierarchy with miss penalty of 5 ns (time for the transfer of the block t_{B2}), a global probability of miss in L2 is 0.5% (general probability of the access to the main memory)
- How faster is the operation of CPU, if we add L2 cache to the memory hierarchy?

2-level memory hierarchy (no L2)



$CPI_I = 1$

$1-H_1 = 0.02$

$H_2 = 1;\ t_{B2}$

miss penalty t_{B2} (time of transfer a block from the main memory to the cache)

t_{B2}= 100 ns = 400 [cp]
[cp] - clock periods

$$t_{CPE} = \frac{1}{f_{CPE}} = \frac{1}{4 \cdot 10^9}[s] = 0{,}25 \cdot 10^{-9}[s] = 0{,}25[ns]$$

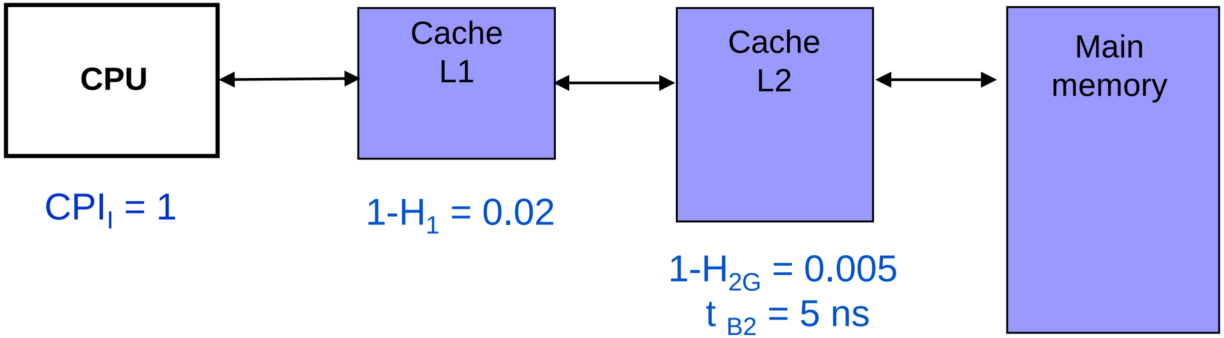
The duration of one clock period

$$t_B = \frac{100[ns]}{0{,}25[\frac{ns}{cp}]} = 400[cp]$$

$$CPI_R(L1) = CPI_I + (1 - H_1) \cdot t_{B2} = 1[cp] + 0{,}02 \cdot 400[cp] = 9\ [cp]$$

Due to the misses in the cache, the CPI increases from 1 to 9 clock periods

3-level memory hierarchy



$CPI_I = 1$

$1\text{-}H_1 = 0.02$

$1\text{-}H_{2G} = 0.005$
$t_{B2} = 5$ ns

$H_3 = 1$; t_{B3}

miss penalty t_{B2} (time of transfer a block from the main memory to the cache)

$t_{B2} = 100$ ns = 400 [cp]
[cp] - clock periods

$1\text{-}H_{2G}$ represents a global probability of miss in relation to all memory accesses and includes local probabilities of miss in L1 and L2 (Main memory is accessed only when both misses happen)

$$t_{B2} = \frac{5[ns]}{0{,}25[\frac{ns}{cp}]} = 20[cp]$$

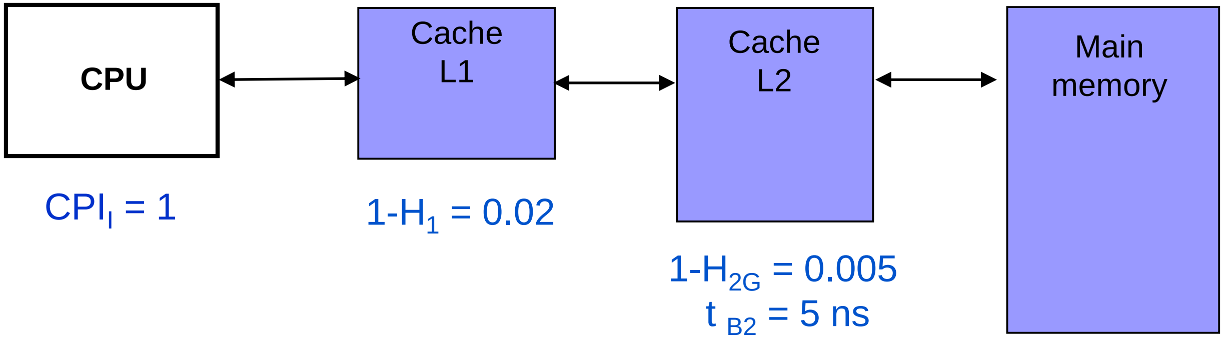
Time to transfer a block from L2 to L1

$$CPI_R(L1,L2) = CPI_I + (1-H_1)\cdot t_{B2} + (1-H_{2G})\cdot t_{B3} =$$
$$= 1[cp] + 0{,}02\cdot 20[cp] + 0{,}005\cdot 400[cp] = 1 + 0{,}4 + 2 = 3{,}4\ [cp]$$

$$Speedup = \frac{CPI_R(L1)}{CPI_R(L1,L2)} = \frac{9}{3{,}4} = 2{,}6$$

If we add L2 cache, the speed increase Is 2.6-fold

3-level memory hierarchy



$CPI_I = 1$

$1\text{-}H_1 = 0.02$

$1\text{-}H_{2G} = 0.005$
$t_{B2} = 5$ ns

$H_3 = 1$; t_{B3}

miss penalty t_{B2} (time of transfer a block from the main memory to the cache)

t_{B2}= 100 ns = 400 [cp]
[cp] - clock periods

$1\text{-}H_{2G}$ represents a global probability of miss in relation to all memory accesses and includes local probabilities of miss in L1 and L2
(Main memory is accessed only when both misses happen)

Comparing calculations using local or global probabilities:

$$CPI_R(L1,L2) = CPI_I + (1-H_1)\cdot(t_{B2} + (1-H_{2L})\cdot t_{B3}) =$$

$$= CPI_I + (1-H_1)\cdot t_{B2} + (1-H_1)(1-H_{2L})\cdot t_{B3}$$

$$= CPI_I + (1-H_1)\cdot t_{B2} + (1-H_{2G})\cdot t_{B3}$$

$$= 1[cp] + 0{,}02\cdot 20[cp] + 0{,}005\cdot 400[cp] = 1 + 0{,}4 + 2 = 3{,}4\ [cp]$$

In L1 cache local and global probabilities of miss are the same, because all memory accesses come in L1 cache.

In L2 cache a local probability of miss is $1\text{-}H_{2L}$ expressed in relation to local accesses only (in L2), while global probability is related to all memory accesses .

In multilevel hierarchies, global probabilities tend to be more useful, as they include also the impact of previous levels (local ones refer only to a certain level)

Example: Effect of L2 cache to the CPU speed: local and global probabilities

Global :

$$CPI_R = CPI_I + (1 – H_1) * t_{BL2} + (1 – H_{2G}) * t_{BG}$$

$$CPI_R = 1 + 0.02*20 + 0.005*400 = 3.4\ t_{CPE}$$

Local :

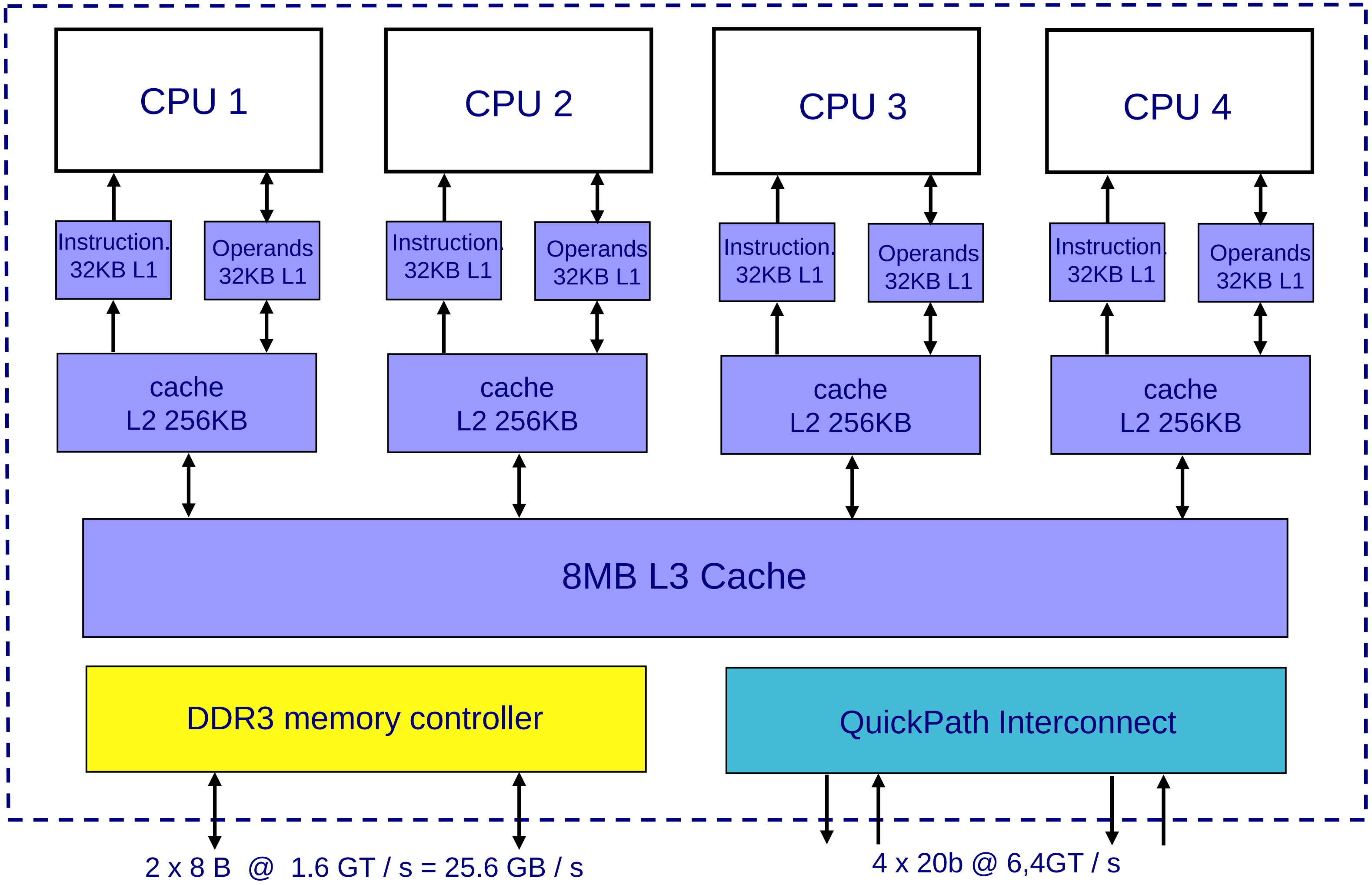
$H_{1L} = H_1$;

$(1-H_{2L})=(1-H_{2G})/(1-H_1)=0.005/0.02 = 0.25$

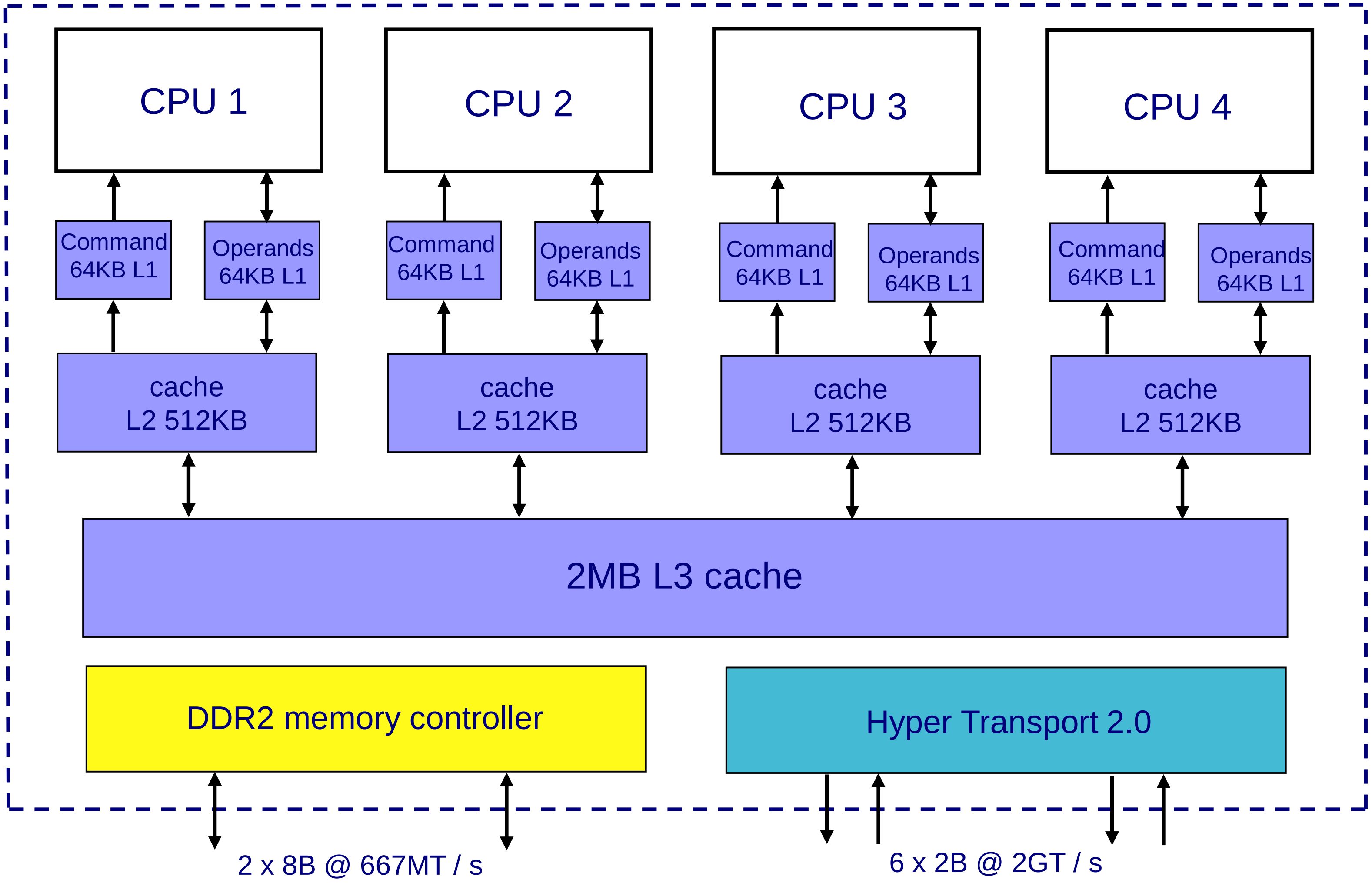
$$CPI_R = CPI_I +(1 – H_1) * (t_{BL2} + (1 – H_{2L}) * t_{BG})$$

$$CPI_R = 1 + 0.02* (20 + 0.25*400) = 3.4\ t_{CPE}$$

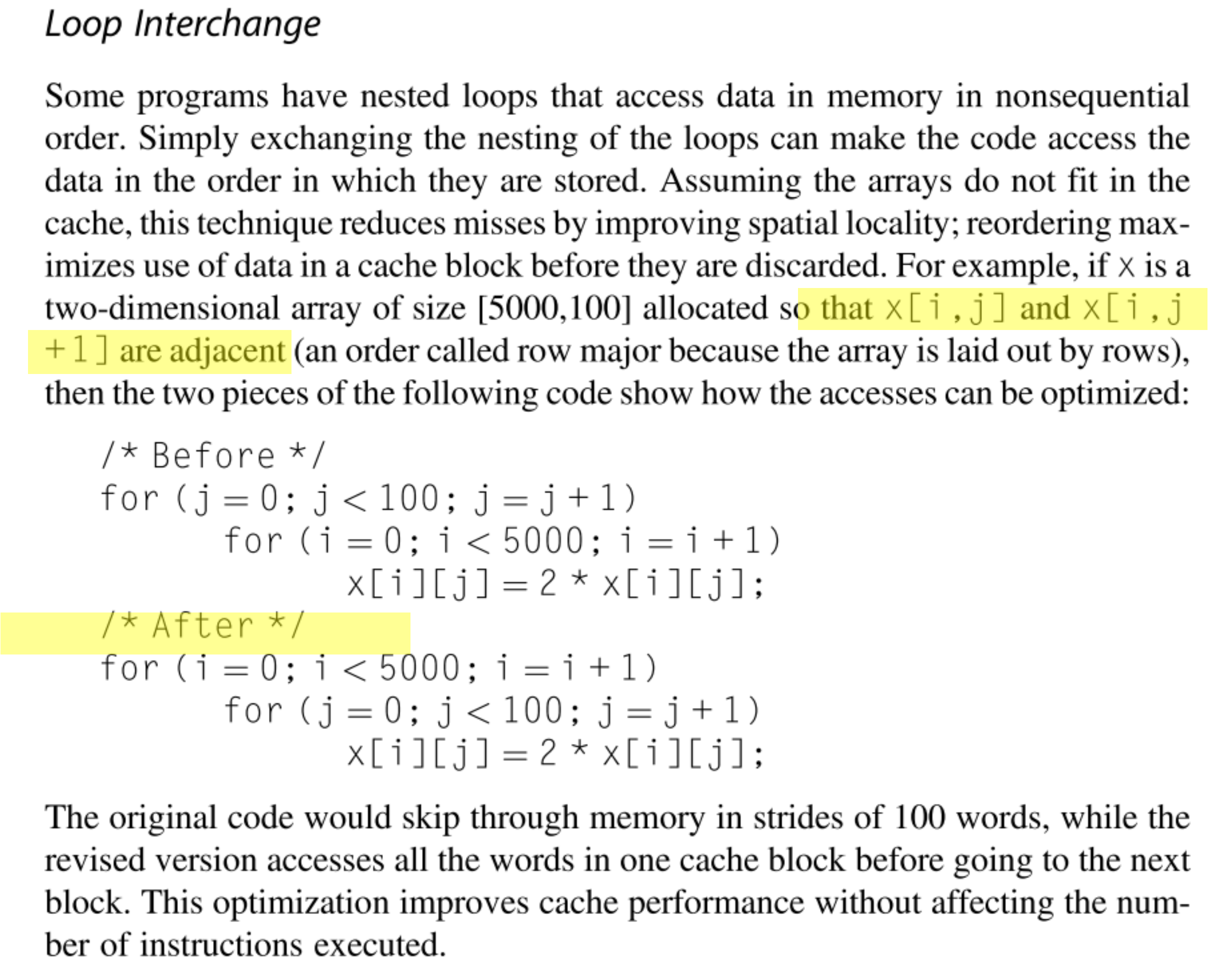
Structure of 4-core processor Intel Core i7 (Haswell)



Structure 4-core processor AMD Opteron (Barcelona)



Caches – Effect of a program on execution speed in memory hierarchy



Loop Interchange

Some programs have nested loops that access data in memory in nonsequential order. Simply exchanging the nesting of the loops can make the code access the data in the order in which they are stored. Assuming the arrays do not fit in the cache, this technique reduces misses by improving spatial locality; reordering maximizes use of data in a cache block before they are discarded. For example, if x is a two-dimensional array of size [5000,100] allocated so that x[i,j] and x[i,j+1] are adjacent (an order called row major because the array is laid out by rows), then the two pieces of the following code show how the accesses can be optimized:

```
/* Before */
for (j = 0; j < 100; j = j + 1)
        for (i = 0; i < 5000; i = i + 1)
                x[i][j] = 2 * x[i][j];
/* After */
for (i = 0; i < 5000; i = i + 1)
        for (j = 0; j < 100; j = j + 1)
                x[i][j] = 2 * x[i][j];
```

The original code would skip through memory in strides of 100 words, while the revised version accesses all the words in one cache block before going to the next block. This optimization improves cache performance without affecting the number of instructions executed.

Caches – Effect of a program on execution speed in memory hierarchy

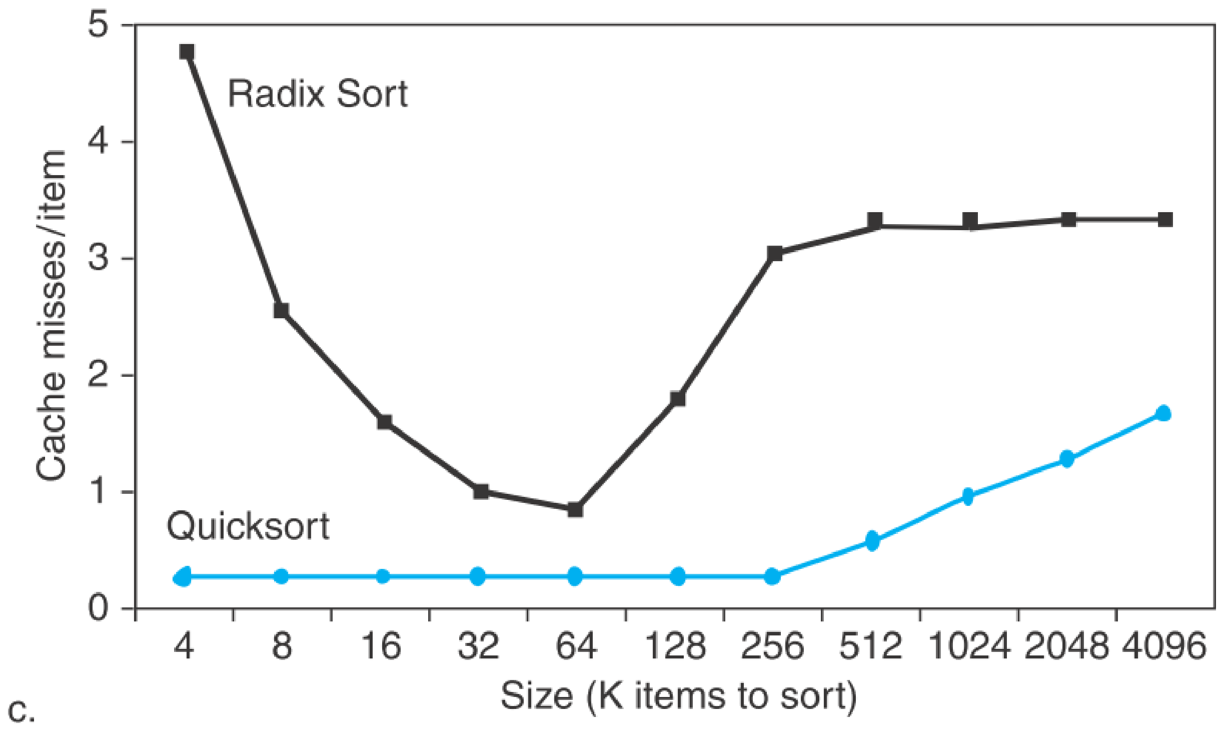


FIGURE 5.19 Comparing Quicksort and Radix Sort by (a) instructions executed per item sorted, (b) time per item sorted, and (c) cache misses per item sorted. These data are from a paper by LaMarca and Ladner [1996]. Due to such results, new versions of Radix Sort have been invented that take memory hierarchy into account, to regain its algorithmic advantages (see Section 5.15). The basic idea of cache optimizations is to use all the data in a block repeatedly before they are replaced on a miss.

9.4 Virtual Memory

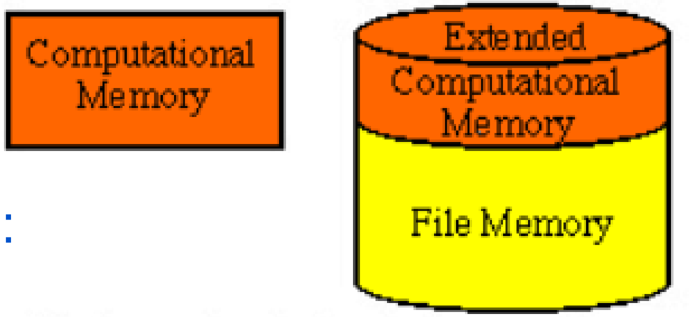
- Virtual memory (virtual memory) ⇒ space in secondary memory (SSD or magnetic disk), which is from the user viewpoint seen as the main memory.

- Access to the auxiliary (secondary) memory is implemented with the I/O commands or I/O programs.

- Transfers between the main and virtual memories are invisible to the user (⇒ virtual memory)

- The additional logic in the CPU and software is needed

- Virtual memory is in most today computers, the reason is not only size of the main memory as was years ago, but also:
  - □ Much lower cost of secondary memory.
  - □ Simple solution for positional independence of programs.
  - □ Memory protection.



(ii) Conventional virtual memory systems

- Space in secondary memory (e.g. HDD):
  - □ Space for virtual memory.
  - □ Storage for files (typically much larger part).

- The access time and the transfer of the information (= miss penalty) from the auxiliary memory to the main memory is very long.

- Solutions to reduce the impact of very large miss penalties for virtual memory:

  - □ The blocks must be large (4KB, 8KB, up to 64KB or more)

  - □ Each block can be mapped to an arbitrary block of main memory (no restrictions)

  - □ Blocks replacements are done by the software and not the hardware as in the case of cache

Virtual memory

- Memory address from CPU = **virtual address** (as it relates to virtual memory).
- In conjunction with the virtual memory, we denote main memory as **physical memory.**
- Address that refers to the main memory = **physical address**

Virtual memory

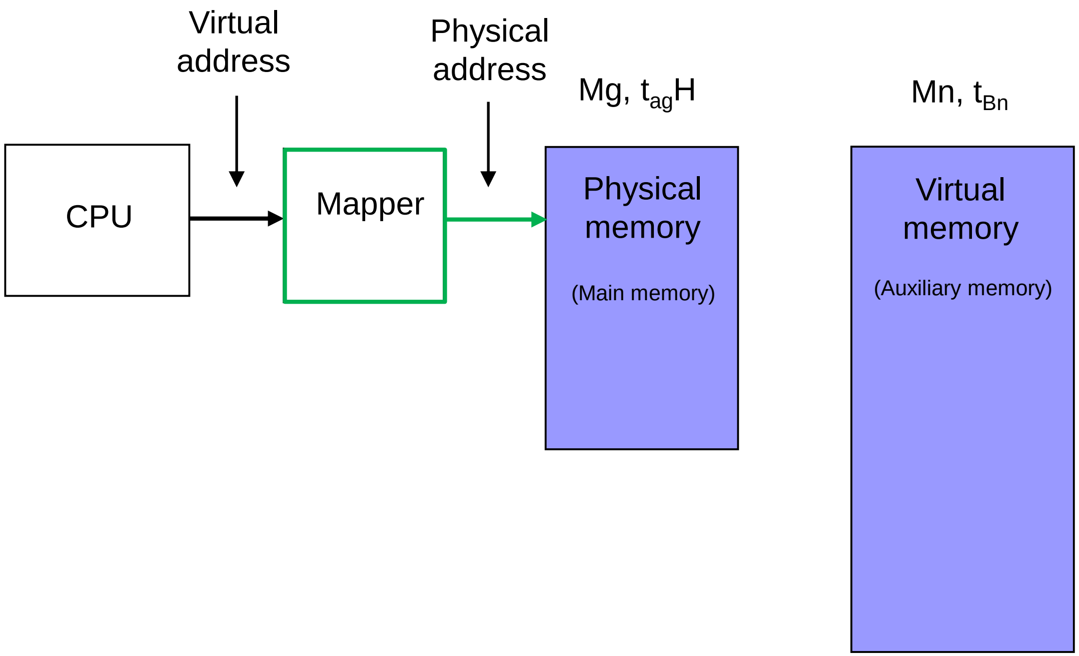
- For each memory access:
  virtual address → mapping → physical address

- Physical address exists, if there is a hit in main (physical) memory.

- For most computers, the physical address (not virtual) is used to access the caches.

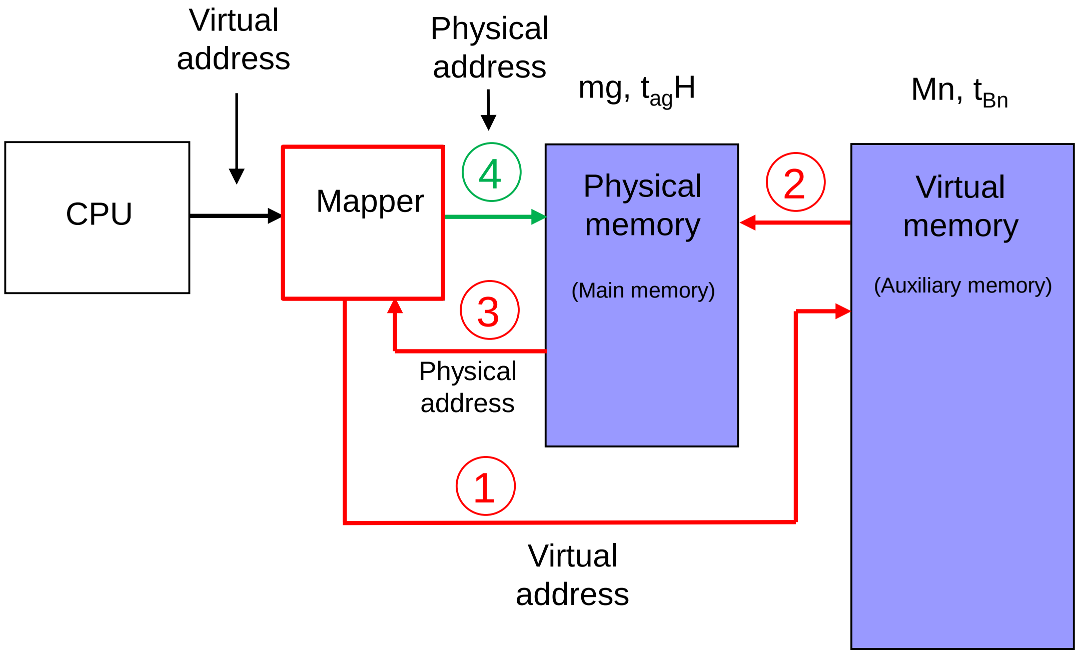
Mapping of virtual addresses

Addressed information is in physical memory - hit
Probability of hit *H*



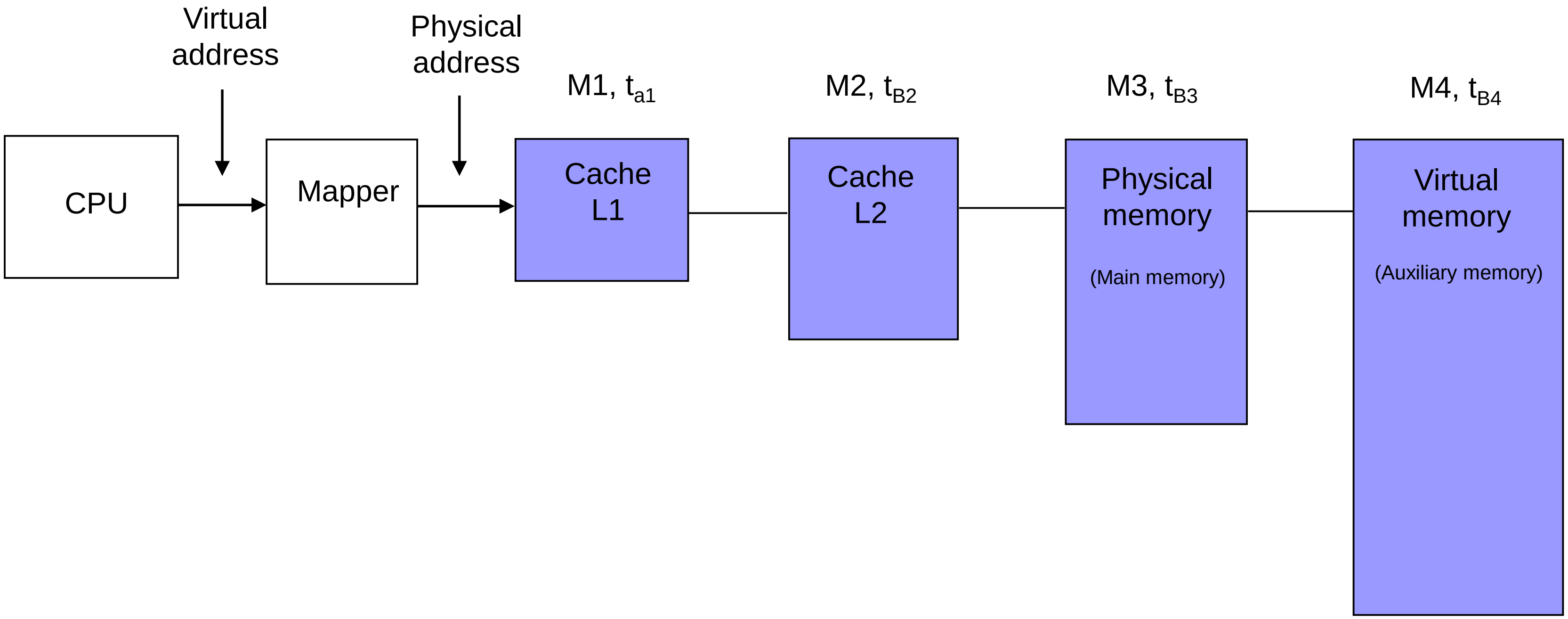
Mapping of virtual addresses

Addressed information is not in physical memory - miss
Probability of miss *1-H*



Mapping of virtual addresses

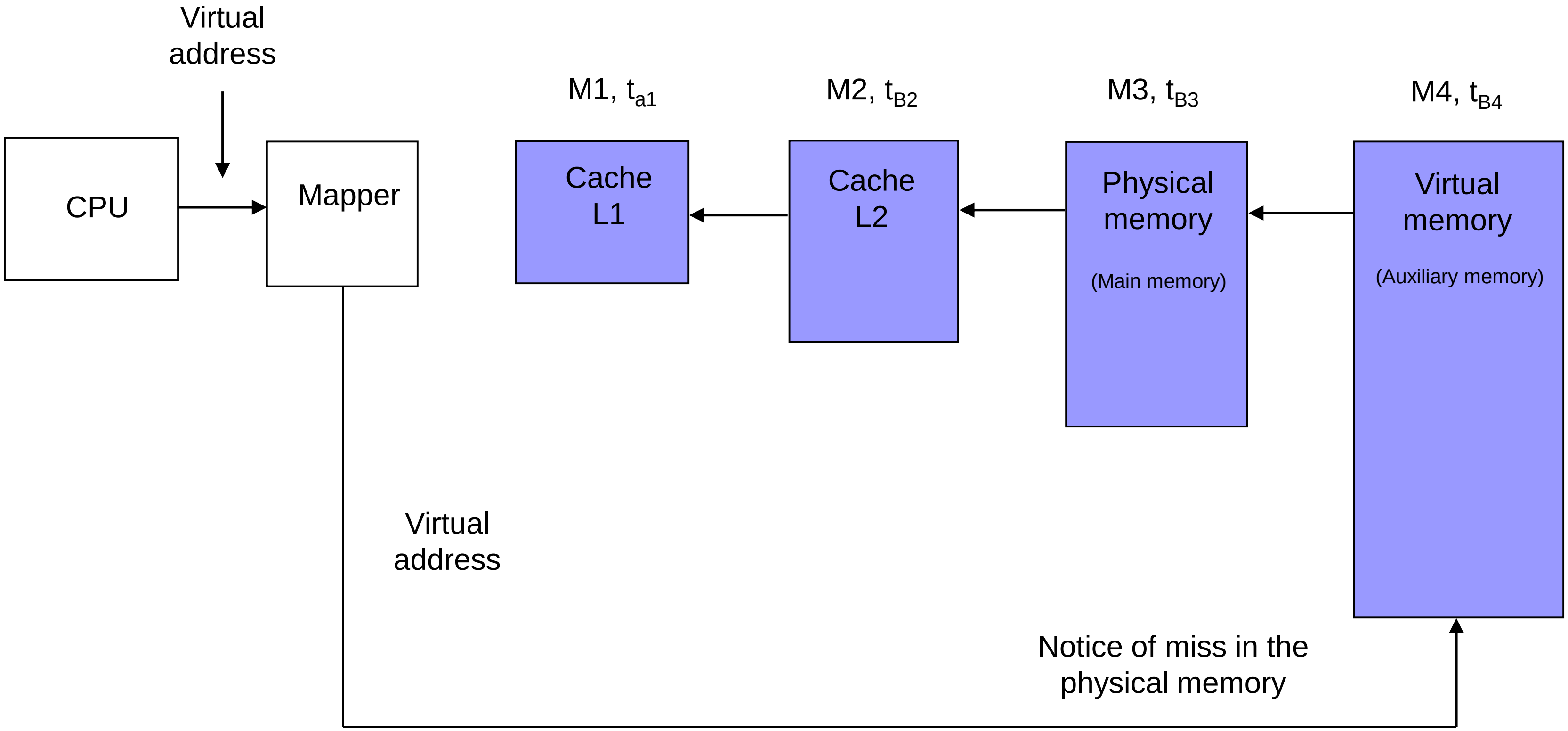
Entire hierarchy
Addressed Information is in physical memory - hit



Mapping of virtual addresses

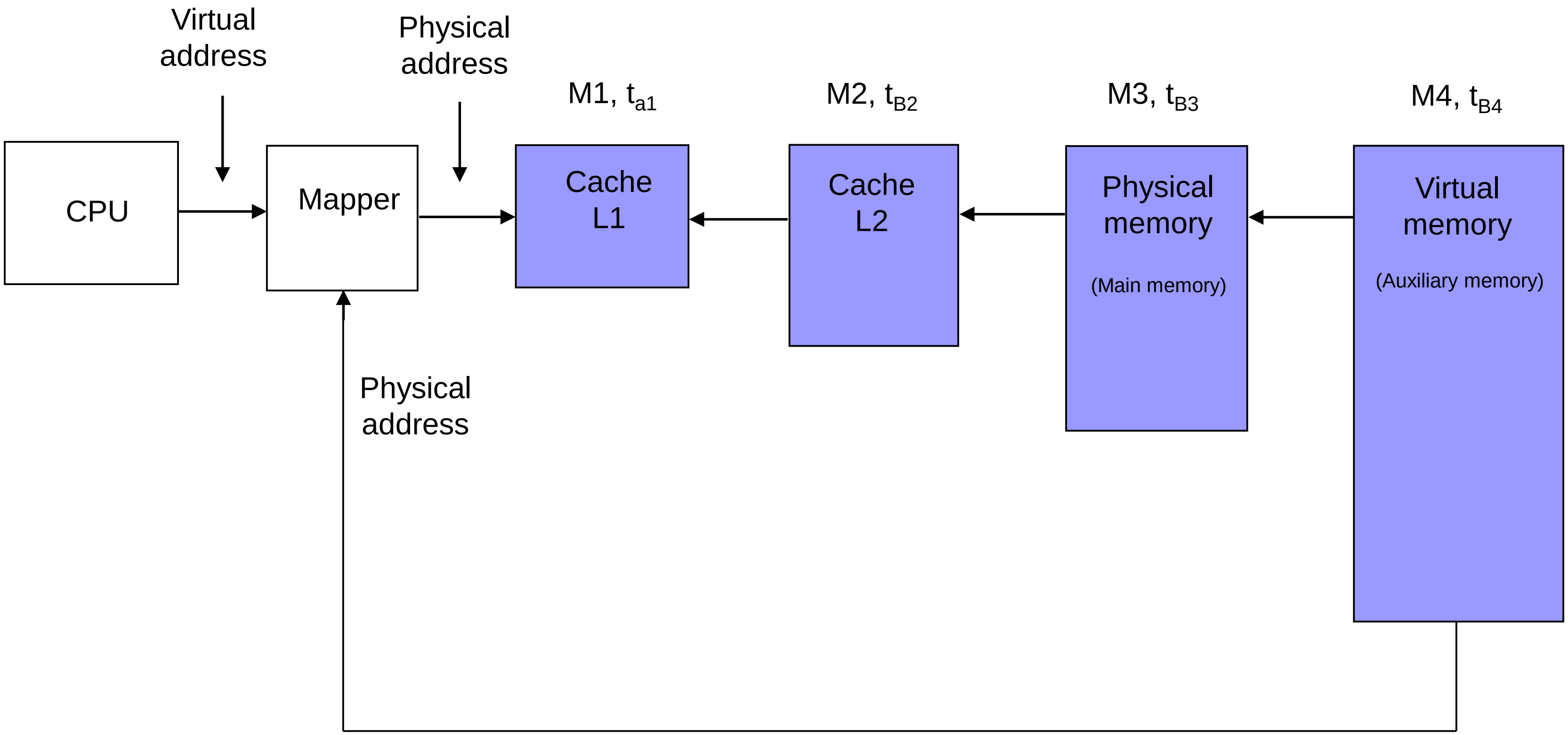
Entire hierarchy

Addressed information is not in physical memory - miss



Mapping of virtual addresses

Entire hierarchy
Addressed information is not in physical memory - miss



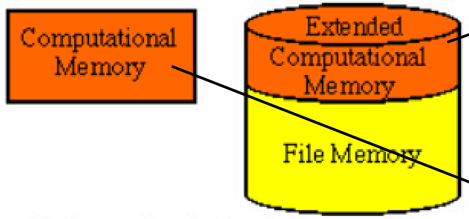
Mapping of virtual addresses

- mapping function is established in software (operating system)
- When you turn on your computer, the mapping of virtual addresses into physical must be switched off (because it does not yet work).
- Mapping can be switched off at any time, in this case :

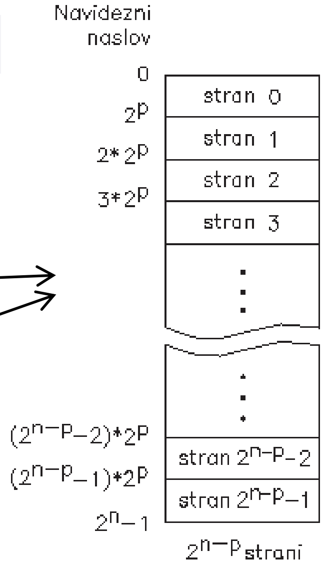
  virtual address = physical address

Virtual memory by paging

- Auxiliary memory ⇒ divided into pages:
  - □ pages ⇒ blocks of equal size.

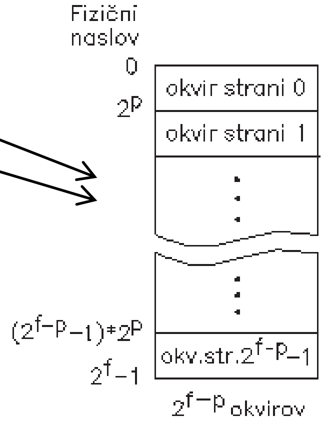


(ii) Conventional virtual memory systems



a) Navidezni pomnilnik velikosti 2^n besed

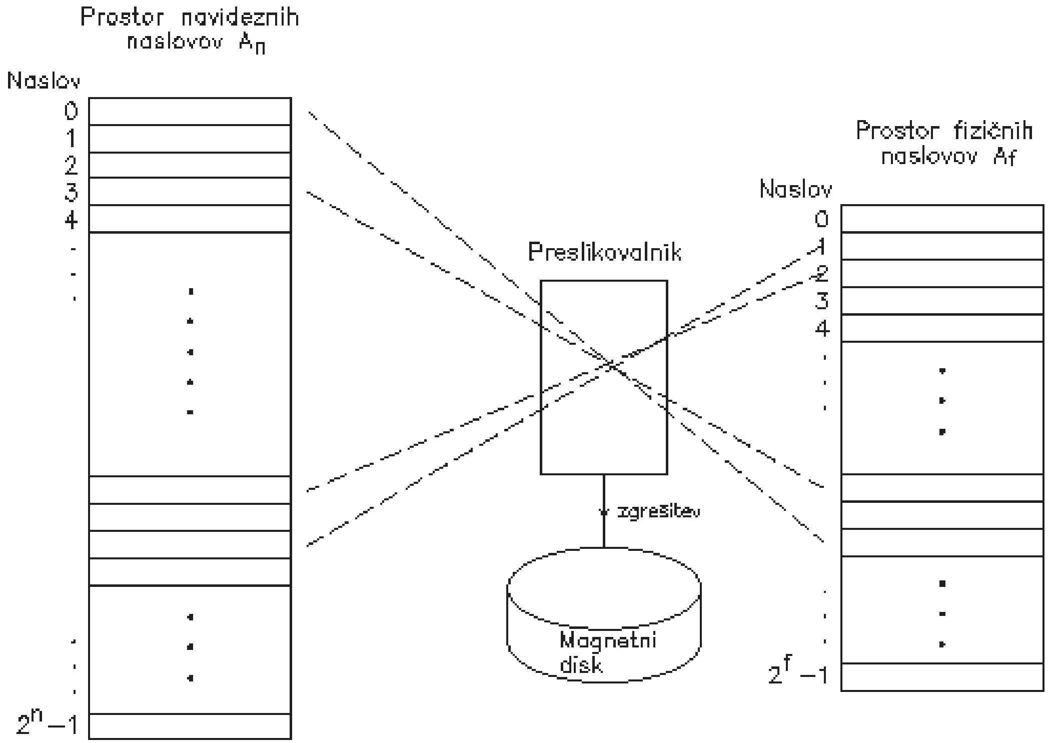
- Main memory ⇒ divided into page frames:
  - □ page frames ⇒ slots of the same size as in the secondary (auixiliary) memory.



- Number of pages in the virtual memory is usually much larger as the number of frames in the main memory:
  - □ illusion of practically unlimited large memory.

b) Fizični pomnilnik velikosti 2^f besed

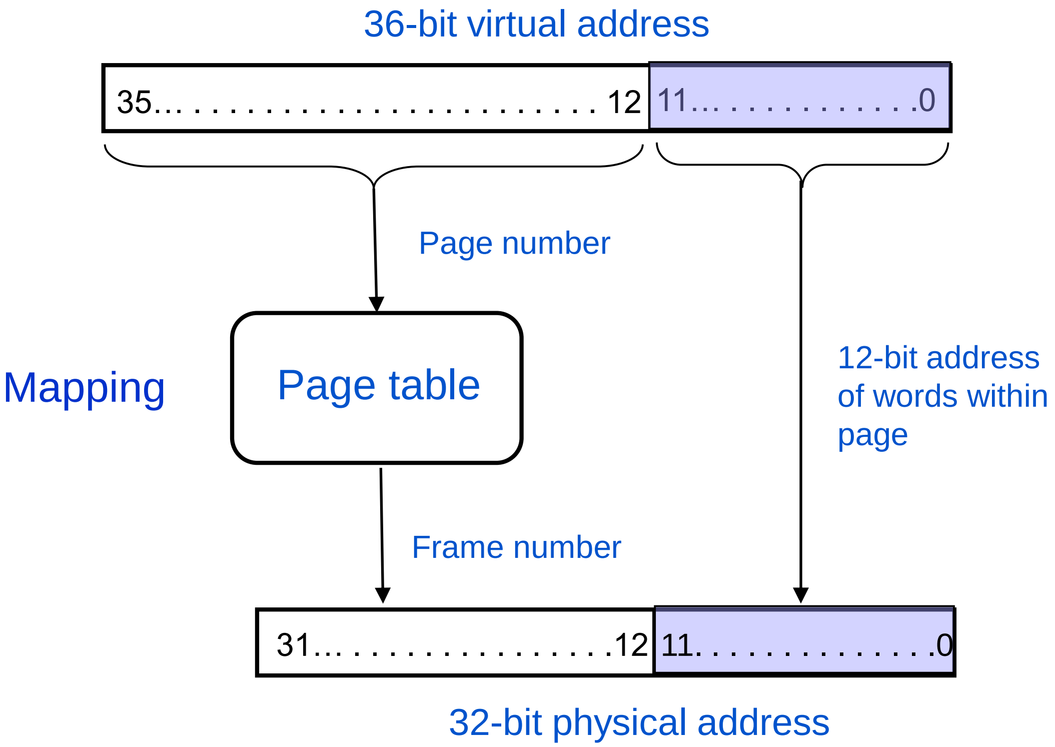
- Each page from the virtual memory can be downloaded in any frame in physical memory.
- to the user, the division of the memory space to pages is invisible.



- Mapping of virtual addresses (page address) to a physical addresses (frames) is through the page table ->

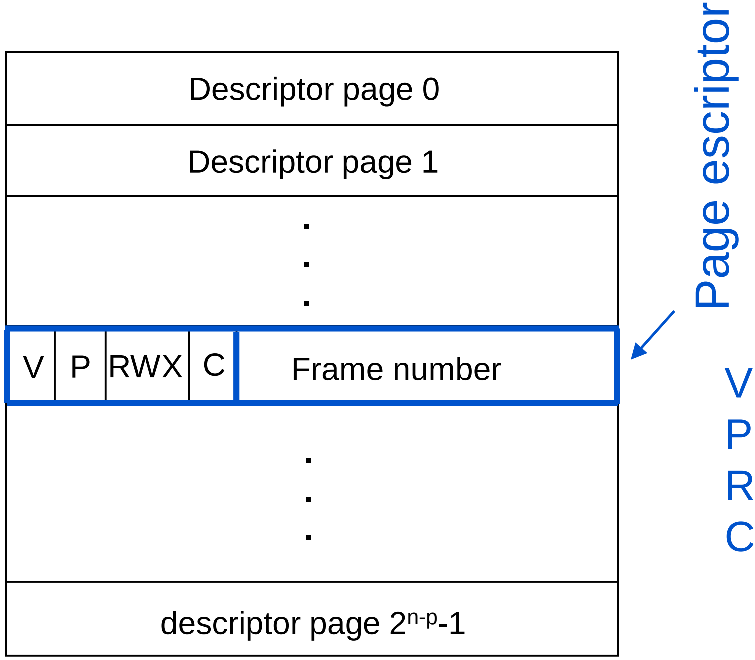
Case: Mapping of virtual addresses into physical in case of paging:

Page size (and frame) 4 KB (⇒ 2^{12} B)
Virtual address of 36 bits (⇒ Virtual memory max 2^{36} B = 64 GB)
Physical address of 32 bits (⇒ Physical memory max 2^{32} B = 4 GB)



building tables page

The size of the virtual memory 2^n Bytes (where n = 36 ⇒ virtual memory = 64 GB)
Size page 2^p Bytes (at p = 12 ⇒ page size = 4K)
Number of pages in virtual memory = 2^{n-p} ($2^{36-12} = 2^{24}$ = 16 M pages (M = 2^{20}))
Number of descriptors in Page table = Number of pages = 16 M



Descriptor page 0

Descriptor page 1

V | P | RWX | C | Frame number

descriptor page 2^{n-p}-1

Page escriptor

V - valid bit (Valid)
P – presence bit (Present)
RWX - protection key (Read, Write, eXecute)
C - dirty bit (Change)

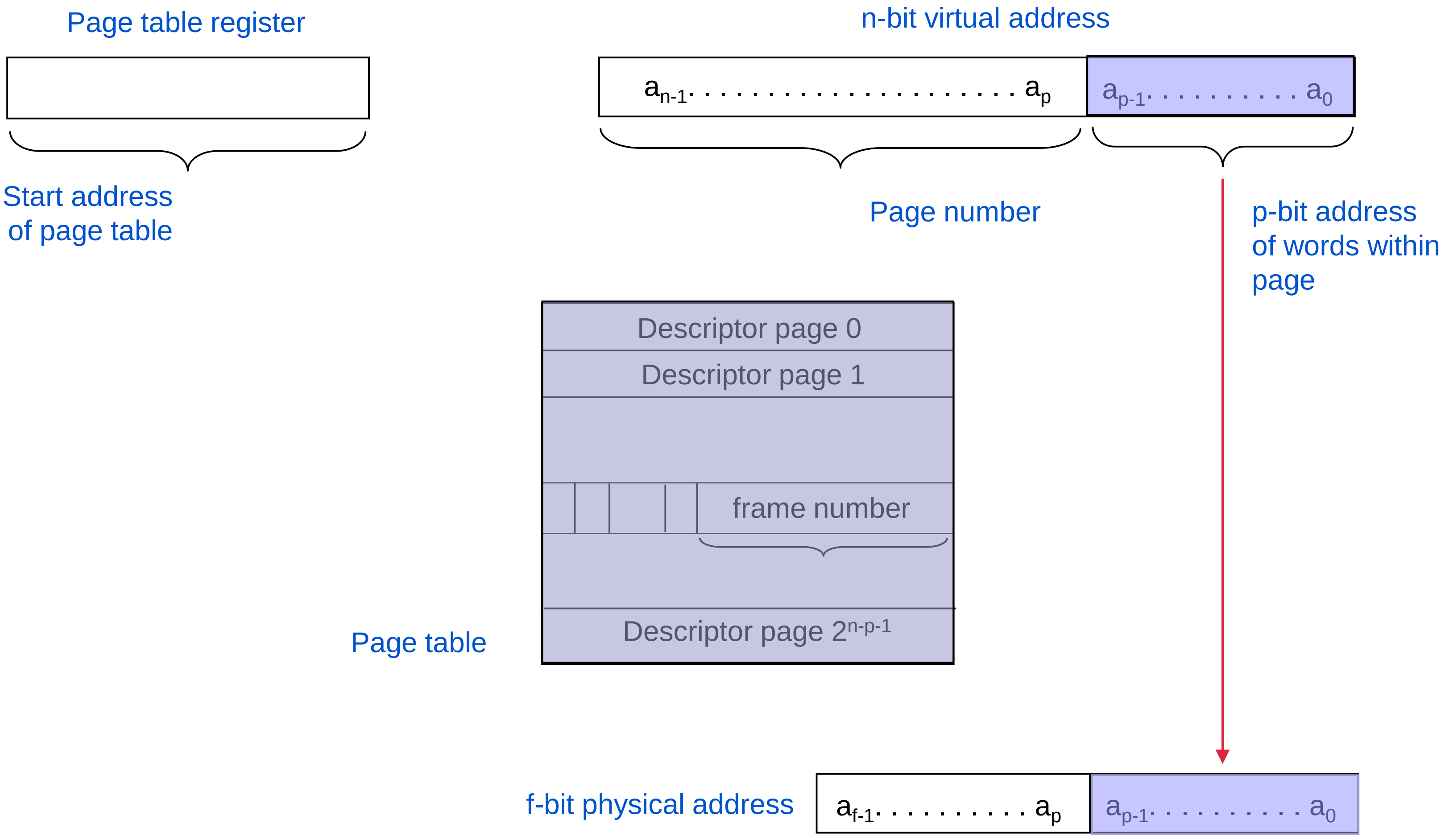
Virtual memory - paging

- Page descriptor ⇒ field in the page table, that describes a particular page.

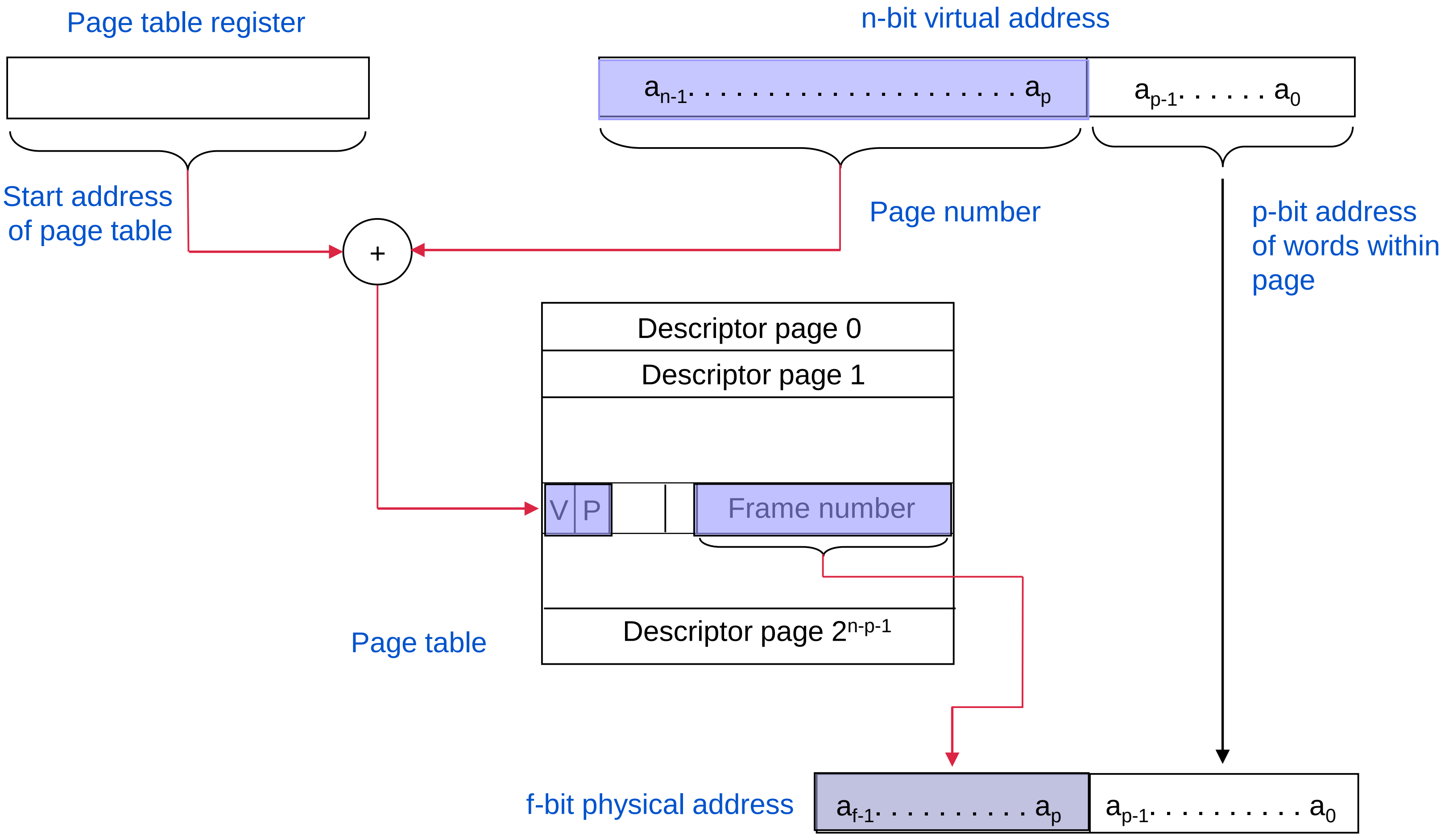
- Number of descriptors in the page table is equal to the number of pages in the virtual memory.

- Table page is usually located in the main memory.

Mapping virtual addresses into physical with paging


Page table register
n-bit virtual address
a_{n-1}. . . . . a_p
a_{p-1}. . . . . a_0
Start address of page table
Page number
p-bit address of words within page
Descriptor page 0
Descriptor page 1
frame number
Page table
Descriptor page 2^{n-p-1}
f-bit physical address
a_{f-1}. . . . . a_p
a_{p-1}. . . . . a_0

Mapping virtual addresses into physical with paging


Page table register
n-bit virtual address
a_{n-1}. . . . . a_p
a_{p-1}. . . . . . a_0
Start address of page table
Page number
p-bit address of words within page
+
Descriptor page 0
Descriptor page 1
V
P
Frame number
Page table
Descriptor page 2^{n-p-1}
f-bit physical address
a_{f-1}. . . . . a_p
a_{p-1}. . . . . a_0

Virtual memory - paging

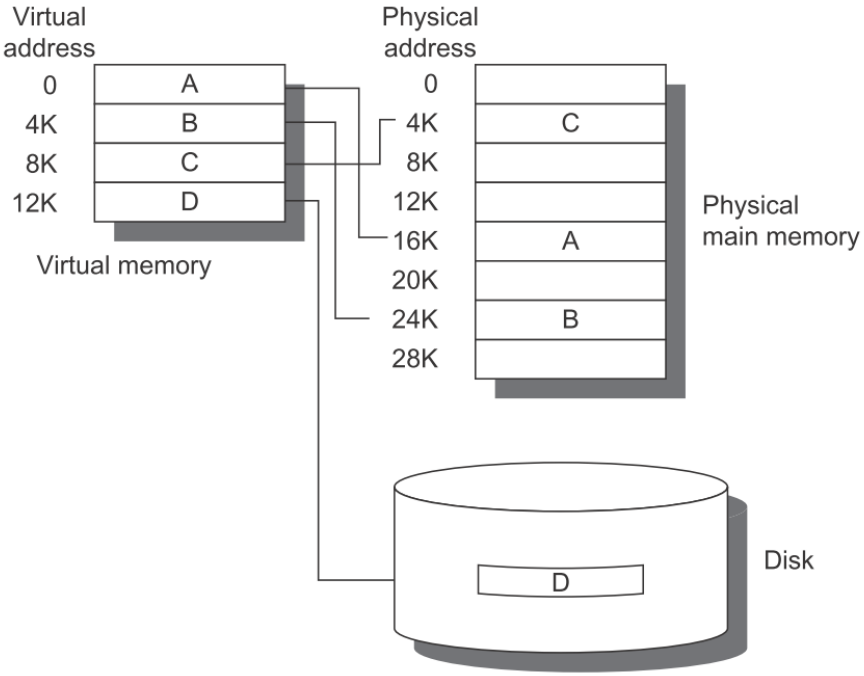
- Linear mapping - a virtual address space is linear. Mapping the virtual addresses has no restrictions, as if we did not have virtual memory.

- Division of storage space ton pages is invisible to the user - normal programmers do not need to know of the existence of the pages.

- A single Page table ⇒ One-level mapping

Virtual memory - paging

- Operating system for each program establishes its page table. When you switch to another program to replace the contents of the register, which points to the page table



- Program state is defined by the page table, program counter, and registers (= process).

- Page table determines the address space, that can be used by the process (program).

Virtual memory - paging

- Page tables obviously take up a lot of space in memory
- Page table can be divided into multiple levels ⇒ multi-level mapping
- Advantage: reduces the space occupied by a page tables in main memory.
- Mostly, two or three-level mapping over two or three levels of page table is used.

Virtual memory - paging

- The operating system allocates main (physical) memory to processes and is responsible for updating the page table.

- Virtual memory allows the use of main memory to multiple processes so that:

    - a memory space of one process is protected from other processes.

Page faults

- Page fault: if the virtual page is not in any of the frames in the main memory (P-bit of page descriptor = 0), it triggers an exception for the page fault.

- Page fault exception ⇒ starts a service program that:
  - □ finds a page in the virtual memory (on disk);
  - □ determines the frame in main memory, where a page will be mapped and transferred,
  - □ updates descriptor of this page in page table.

Virtual memory - paging

- When the operating system creates a process, usually creates space for all process‘ pages (swap space).

- At the same time, it creates a data structure that for each page contains information, where it is stored on disk.

Comparison of virtual memory realizations

	Intel Core i7 (Nehalem)	ARM Cortex-A8 (32-bit)	ARM Cortex-A53 (64-bit)
Virtual address	48 bits	32 bits	48 bits
Physical address	44 bits	32 bits	44 bits
Page size	4 KB, 2 MB, 4 MB	4, 16, 64 KB; 1,16 MB	4, 16, 64 KB; 1, 2 MB; 1 GB

Strategies and algorithms

- Operation of virtual memory is controlled by operating system, with the aim of achieving maximum utilization of the computer.

- As a large utilization, it is generally considered that the given set of programs is executed in the shortest possible time.

- The utilization of computer is influenced by the choice of rules that determine:
  - How many page frames in the main memory are assigned to a program.
  - When, where and how many pages should be transferred from the auxiliary (secondary) to the main memory.
  - Which pages should be transferred from the main memory back to the auxiliary memory.

Virtual memory - strategies and algorithms

- These rules are called **assignment, filling and replacement strategies**.

- When virtual memory strategies are realized in a program, on the other hand in caches it is realized by hardware.

- All three strategies are implemented with algorithms collectively denoted as **memory management**.

Speed up of mapping

- When mapping a virtual address into a physical address
  - □ it requires access to the page table
  - □ tables are stored in main memory or even in virtual memory

- Any access to the memory therefore requires two accesses to the main memory (if the mapping is single-level):
  - □ 1. access to the page descriptor in the page table in main memory
  - □ 2. access to the desired word in the physical address in main memory

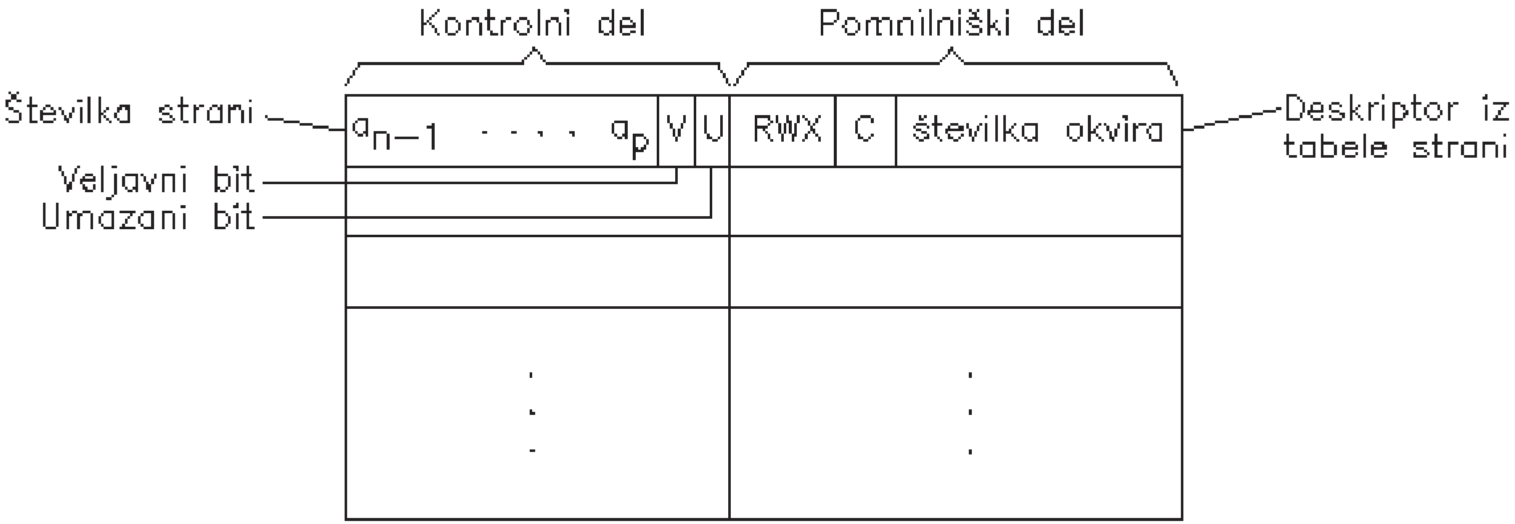
- In multi-level mapping, number of accesses is increased upto 3 to 4 accesses to the main memory.
- Too slow!
- Solution: Special cache in the CPU, that contains some of recently used page descriptors (never operands or instructions).

Mapping cache (translation cache)

- **TLB** (Translation Lookaside Buffer)

- The length of the block in the cache is the same as the length of the page descriptor. In the control part of the cache we have the page number, to which descriptor belongs.

- A high probability of hit (99% to 99.9%) can be achieved with just a few descriptors, therefore TLB cache may be small and fully associative.



Virtual memory - Cache mappings

- With the hit in mapping cache (TLB), access to the page table in main memory is not necessary.

- Harvard architecture (separate instruction and operand cache), requires two mappings caches (instruction and operand - ITLB and DTLB).

9.5 Operation of the memory hierarchy

- The memory hierarchy from the CPU looks like a single memory:
  - With a speed that is close to the speed of cache (memory that is closest to the CPU).
  - The size of the virtual memory on auxiliary memory (last in the memory hierarchy).

Memory hierarchy

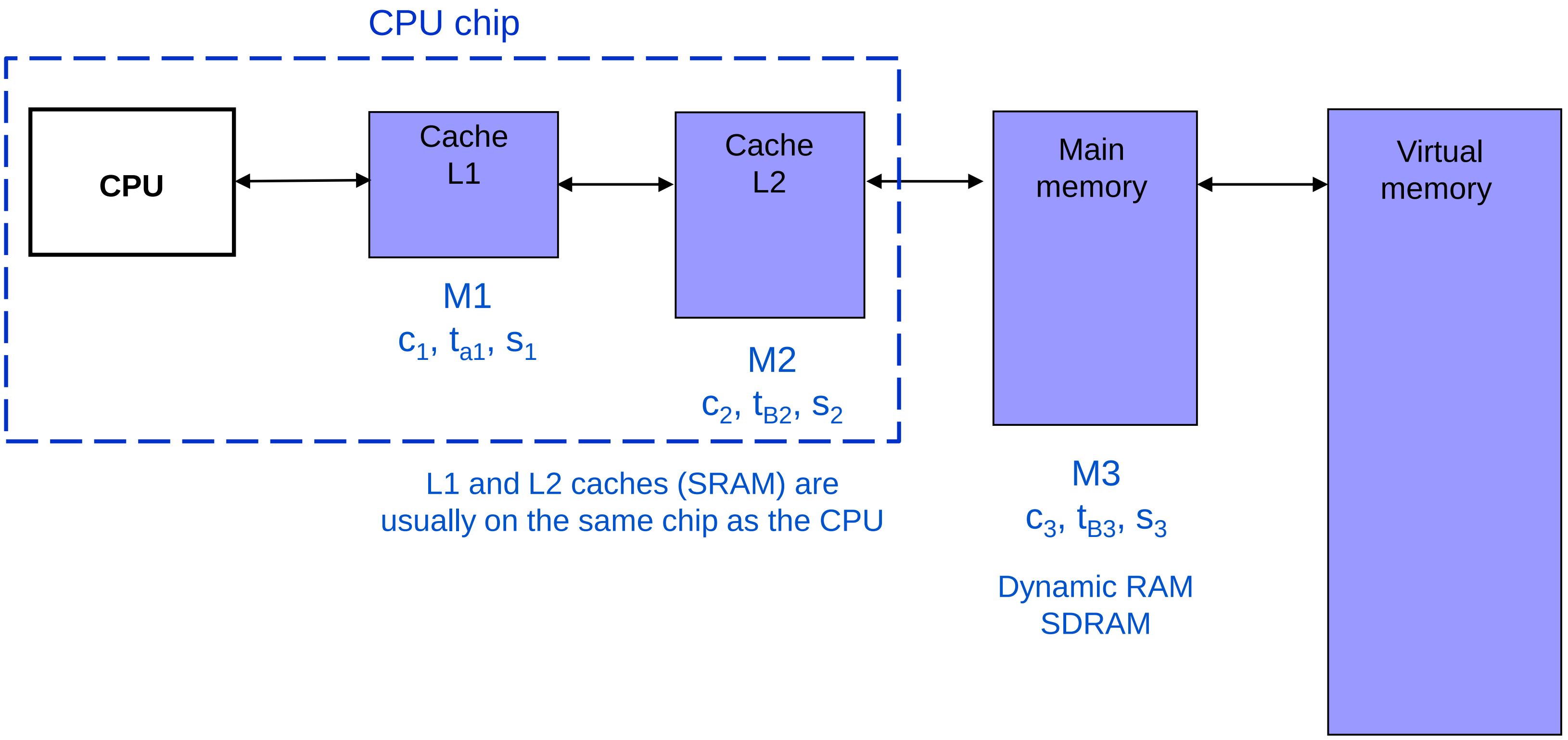
- The memory hierarchy differs from single-level memory in following characteristics:
  - □ The access time is not the same for all memory addresses, it depends on the level of a memory in which currently searched memory word is located.
  - □ For certain memory access we can not predict its duration, we can only calculate statistically determined average value of the access time.

- CPU sends to the memory hierarchy always address that refers to memory M_n (last in the hierarchy), but this does not mean that access is in fact carried out to M_n.
  - □ if the information wanted by the CPU, is in M_1 (⇒ hit), then access to M_1 is executed.
  - □ if the information is not in M_1 (⇒ miss), it is transferred from the M_2 to M_1
  - □ if the information is not even in M_2it is transferred from the M_3 to M_2
  - □ . . .
  - □ For any access requested, information is always in the memory M_n on the last level

Comparison of cache and virtual memory with paging

	Cache	Virtual memory
Access	Cache line (block)	Page (page frame)
Block	16B to 128B	4KB to 16KB (also a few MB)
Miss probability (1-H)	0.1% to 10% of L1	<0.0001% (for main memory.)
Hit	few clock periods	~ 10 to 100 clock periods
Miss penalty	~ 10 to 100 clock periods	~ 10M clock periods
Block replacement	hardware	Software

4-level memory hierarchy



CPU chip

CPU

Cache L1

Cache L2

Main memory

Virtual memory

M1

c_1, t_{a1}, s_1

M2

c_2, t_{B2}, s_2

M3

c_3, t_{B3}, s_3

Dynamic RAM
SDRAM

L1 and L2 caches (SRAM) are
usually on the same chip as the CPU

M4

c_4, t_{B4}, s_4

Auxiliary memory
- magnetic disk
or SSD

c_i – cost / bit level i

t_{a1} – access time cache L1

t_{Bi} - time to access and transfer the block
from level i to level i-1

s_i – size of the memory level i

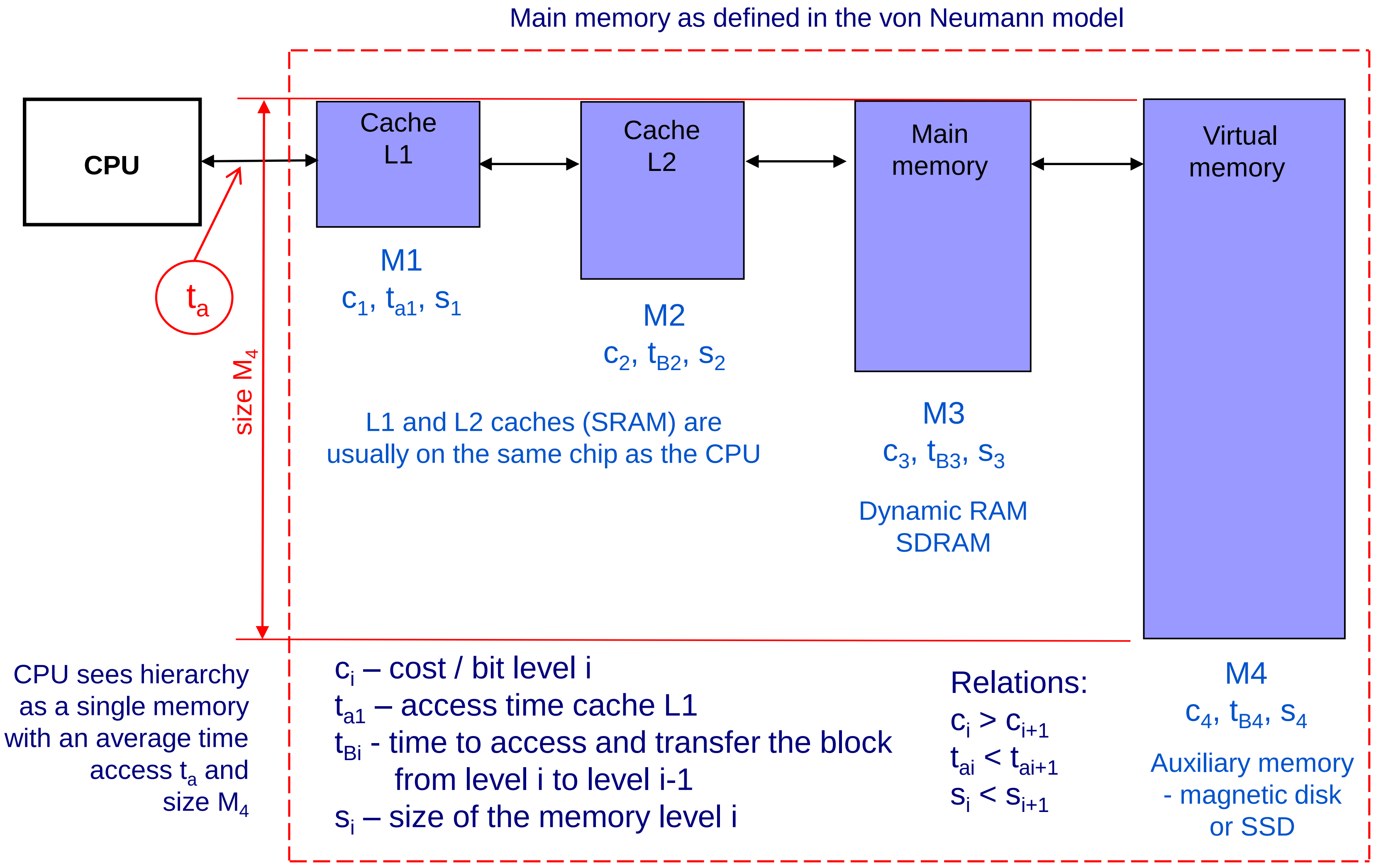
Relations:

$c_i > c_{i+1}$

$t_{ai} < t_{ai+1}$

$s_i < s_{i+1}$

4-level memory hierarchy



Main memory as defined in the von Neumann model

CPU

Cache L1

Cache L2

Main memory

Virtual memory

t_a

size M_4

M1

c_1, t_{a1}, s_1

M2

c_2, t_{B2}, s_2

M3

c_3, t_{B3}, s_3

M4

c_4, t_{B4}, s_4

L1 and L2 caches (SRAM) are usually on the same chip as the CPU

Dynamic RAM SDRAM

Auxiliary memory - magnetic disk or SSD

CPU sees hierarchy as a single memory with an average time access t_a and size M_4

c_i – cost / bit level i

t_{a1} – access time cache L1

t_{Bi} - time to access and transfer the block from level i to level i-1

s_i – size of the memory level i

Relations:

$c_i > c_{i+1}$

$t_{ai} < t_{ai+1}$

$s_i < s_{i+1}$

Rule: If the content is in level *i* , it is certainly also in level (*i + 1).*

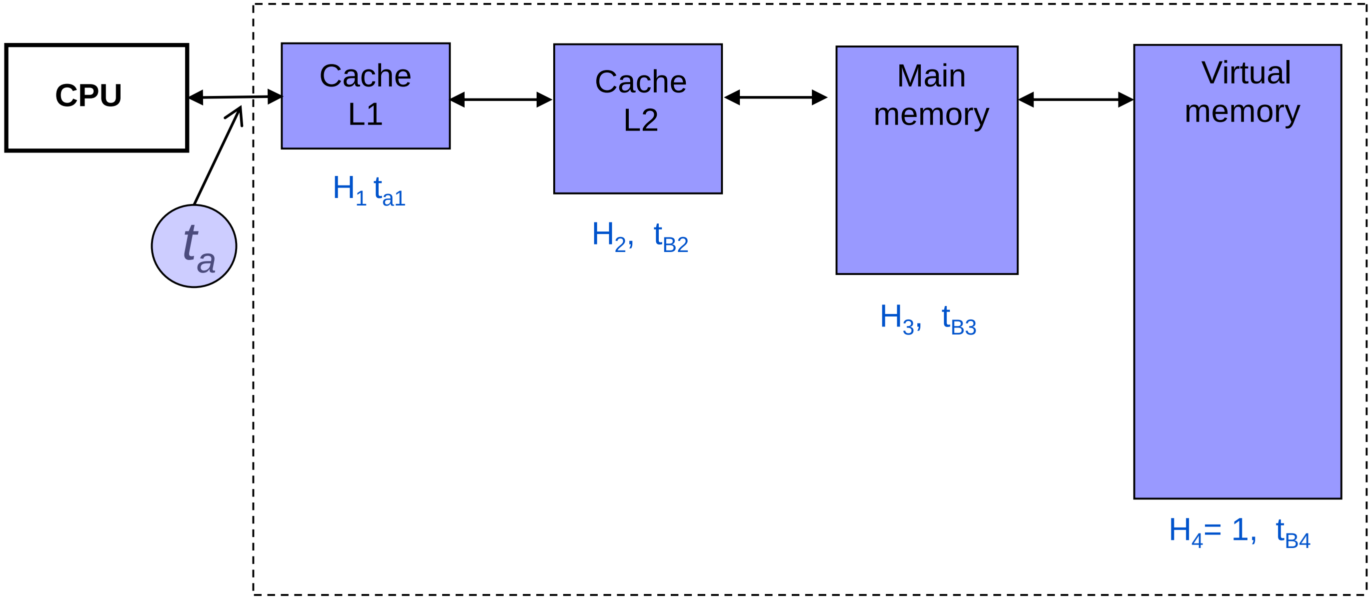
- H_i ⇒ (global) probability that for any access to the memory hierarchy, the content is in the layer *i.*

- $(1 - H_i)$ ⇒ (global) probability that for any access to the memory hierarchy, the content is not in the layer *i.*

- average access time t_a to *n*-level memory hierarchy, as seen by the CPU is:

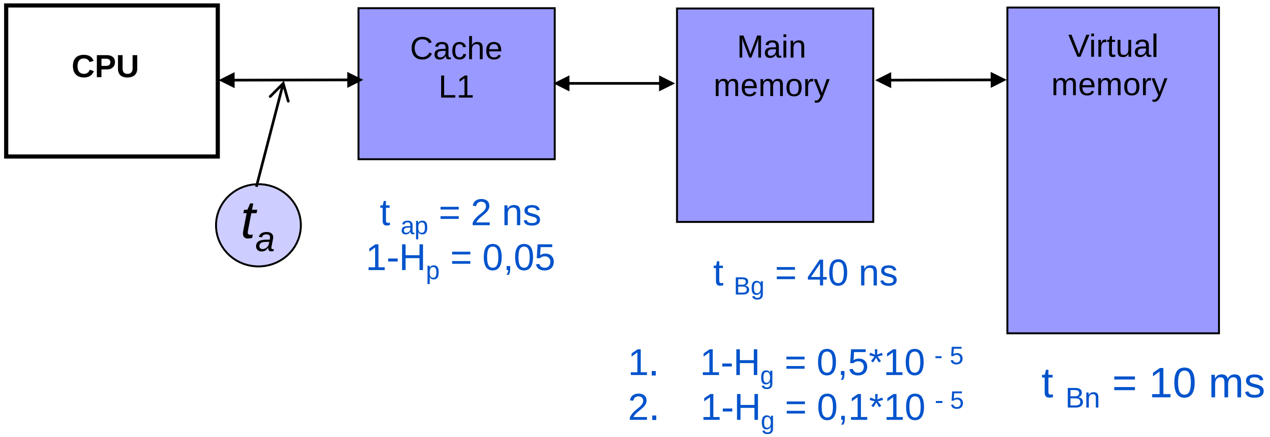
$$t_a = t_{a1} + (1 - H_1)t_{B2} + \ldots + (1 - H_{i-1})t_{Bi} + \ldots + (1 - H_{n-1})t_{Bn}$$

- 4-level memory hierarchy :



$$\boxed{t_a} = t_{a1} + (1 - H_1)t_{B2} + (1 - H_2)t_{B3} + (1 - H_3)t_{B4}$$

Case: Impact of the miss probability in the main memory to the average access time in 3-level hierarchy



t_{ap} - access time of L1 cache
H_p - probability of cache hit in L1 ($1\text{-}H_p$ - probability of miss in L1)
t_{Bg} – access time to the main memory and the transfer of a block from main memory into L1
H_g - probability of hit in main memory ($1\text{-}H_g$ - probability of miss in the main memory)
t_{Bn} – access time to virtual memory and transfer of block from virtual memory to main memory
t_a - average access time of the entire hierarchy as seen by the CPU

Memory hierarchy – case: 3-level memory hierarchy

1. Let the probability of hit in main memory be H_g = 0.999995 = 99.9995%, probability of miss in the main memory is 1-H_g = 1 - 0.999995 = = 0.000005 = 0.0005% or 1-H_g = 0.5 * 10^{-5}

   t_{ap} = 2 ns; 1-H_p = 0.05; t_{Bg} = 40 ns; t_{Bn} = 10 ms

$$t_a = t_{ap} + (1 - H_p) \cdot t_{Bg} + (1 - H_g) \cdot t_{Bn} =$$
$$= 2 \cdot 10^{-9}[s] + 0,05 \cdot 40 \cdot 10^{-9}[s] + 0,5 \cdot 10^{-5} \cdot 10 \cdot 10^{-3}[s] =$$
$$= 2 \cdot 10^{-9}[s] + 2 \cdot 10^{-9}[s] + 5 \cdot 10^{-8}[s] =$$
$$= 2 \cdot 10^{-9}[s] + 2 \cdot 10^{-9}[s] + 50 \cdot 10^{-9}[s] = 54 \cdot 10^{-9}[ns] = 54[ns]$$

Memory hierarchy – case: 3-level memory hierarchy

1. Let the probability of hit in main memory be $H_g = 0.999995 = 99.9995\%$, probability of miss in the main memory is $1\text{-}H_g = 1 - 0.999995 =$ $= 0.000005 = 0.0005\%$ or $1\text{-}H_g = 0.5 * 10^{-5}$

   $t_{ap} = 2$ ns; $1\text{-}H_p = 0.05$; $t_{Bg} = 40$ ns; $t_{Bn} = 10$ ms

$$t_a = t_{ap} + (1 - H_p) \cdot t_{Bg} + (1 - H_g) \cdot t_{Bn} =$$
$$= 2 \cdot 10^{-9}[s] + 0{,}05 \cdot 40 \cdot 10^{-9}[s] + 0{,}5 \cdot 10^{-5} \cdot 10 \cdot 10^{-3}[s] =$$
$$= 2 \cdot 10^{-9}[s] + 2 \cdot 10^{-9}[s] + 5 \cdot 10^{-8}[s] =$$
$$= 2 \cdot 10^{-9}[s] + 2 \cdot 10^{-9}[s] + 50 \cdot 10^{-9}[s] = 54 \cdot 10^{-9}[ns] = 54[ns]$$

The average access time of the entire hierarchy in this case is 54 ns, which is worse than the access time of main memory (40 ns). Such a memory hierarchy is solving the problem of its storage capacity, but it deteriorates the access time and therefore completely useless. The solution is to increase the probability of a hit in main memory.

Memory hierarchy – case: 3-level memory hierarchy

2. If the probability of hit in the main memory increases from 99.9995% to 99.9999% ⇒ $H_g = 0.999999 = 99.9999\%$

   So the probability of miss in the main memory is $1\text{-}H_g = 0.000001 = 0.0001\%$ or $1\text{-}H_g = 0.1 * 10^{-5}$ (in previous example $1\text{-}H_g = 0.5 * 10^{-5}$)

   while other data remain unchanged:
   $t_{ap} = 2$ ns; $1\text{-}H_p = 0.05$; $t_{Bg} = 40$ ns; $t_{Bn} = 10$ ms

$$t_a = t_{ap} + (1 - H_p) \cdot t_{Bg} + (1 - H_g) \cdot t_{Bn} =$$
$$= 2 \cdot 10^{-9}[s] + 0{,}05 \cdot 40 \cdot 10^{-9}[s] + 0{,}1 \cdot 10^{-5} \cdot 10 \cdot 10^{-3}[s] =$$
$$= 2 \cdot 10^{-9}[s] + 2 \cdot 10^{-9}[s] + 1 \cdot 10^{-8}[s] =$$
$$= 2 \cdot 10^{-9}[s] + 2 \cdot 10^{-9}[s] + 10 \cdot 10^{-9}[s] = 14 \cdot 10^{-9}[ns] = 14[ns]$$

Memory hierarchy – case: 3-level memory hierarchy

2. If the probability of hit in the main memory increases from 99.9995% to 99.9999% $\Rightarrow$ H_g = 0.999999 = 99.9999%

   So the probability of miss in the main memory is $1-H_g$ = 0.000001 = 0.0001% or $1-H_g = 0.1 * 10^{-5}$ (in previous example $1-H_g = 0.5 * 10^{-5}$)

   while other data remain unchanged:
   t_{ap} = 2 ns; $1-H_p$ = 0.05; t_{Bg} = 40 ns; t_{Bn} = 10 ms

$$t_a = t_{ap} + (1 - H_p)\cdot t_{Bg} + (1 - H_g)\cdot t_{Bn} =$$

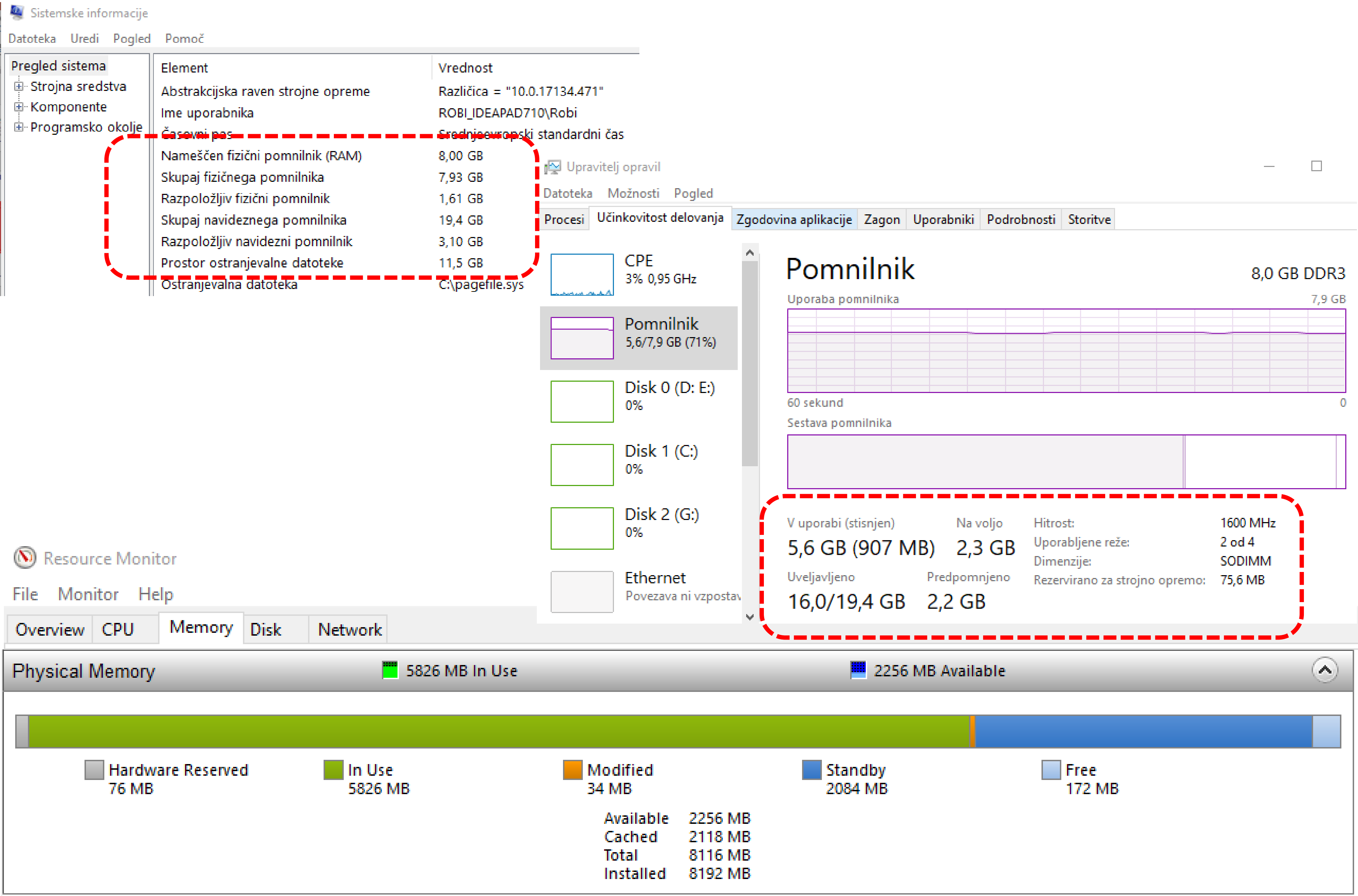
$$= 2\cdot 10^{-9}[s] + 0{,}05\cdot 40\cdot 10^{-9}[s] + 0{,}1\cdot 10^{-5}\cdot 10\cdot 10^{-3}[s] =$$

$$= 2\cdot 10^{-9}[s] + 2\cdot 10^{-9}[s] + 1\cdot 10^{-8}[s] =$$

$$= 2\cdot 10^{-9}[s] + 2\cdot 10^{-9}[s] + 10\cdot 10^{-9}[s] = 14\cdot 10^{-9}[ns] = 14[ns]$$

If the probability of miss in the main memory is reduced from $0.5 * 10^{-5}$ to $0.1 * 10^{-5}$ (probability of hit is increased), the average access time is reduced from 54 ns to 14 ns.

Case: Virtual memory in Win10 (SLO)



Sistemske informacije

Element	Vrednost
Abstrakcijska raven strojne opreme	Različica = "10.0.17134.471"
Ime uporabnika	ROBI_IDEAPAD710\Robi
Časovni pas	Srednjeevropski standardni čas
Nameščen fizični pomnilnik (RAM)	8,00 GB
Skupaj fizičnega pomnilnika	7,93 GB
Razpoložljiv fizični pomnilnik	1,61 GB
Skupaj navideznega pomnilnika	19,4 GB
Razpoložljiv navidezni pomnilnik	3,10 GB
Prostor ostranjevalne datoteke	11,5 GB
Ostranjevalna datoteka	C:\pagefile.sys

Upravitelj opravil

Datoteka Možnosti Pogled
Procesi | Učinkovitost delovanja | Zgodovina aplikacije | Zagon | Uporabniki | Podrobnosti | Storitve

CPE 3% 0,95 GHz
Pomnilnik 5,6/7,9 GB (71%)
Disk 0 (D: E:) 0%
Disk 1 (C:) 0%
Disk 2 (G:) 0%
Ethernet Povezava ni vzpostav

Pomnilnik — 8,0 GB DDR3

V uporabi (stisnjen)	Na voljo
5,6 GB (907 MB)	2,3 GB

Uveljavljeno	Predpomnjeno
16,0/19,4 GB	2,2 GB

Hitrost:	1600 MHz
Uporabljene reže:	2 od 4
Dimenzije:	SODIMM
Rezervirano za strojno opremo:	75,6 MB

Resource Monitor

Physical Memory — 🟩 5826 MB In Use — 🟦 2256 MB Available

Hardware Reserved	In Use	Modified	Standby	Free
76 MB	5826 MB	34 MB	2084 MB	172 MB

Available	2256 MB
Cached	2118 MB
Total	8116 MB
Installed	8192 MB

- Thanks for attention and best wishes for the exams !
- Web pages: http://ucilnica.fri.uni-lj.si
  http://www.fri.uni-lj.si/
- Email: rozman@fri.uni-lj.si
- Literature:
  - □ Dušan Kodek: ARHITEKTURA IN ORGANIZACIJA RAČUNALNIŠKIH SISTEMOV, Bi-TIM, 2008
  - □ David A. Patterson, John L. Hennesy: COMPUTER ORGANIZATION AND DESIGN, ARM Edition, Morgan Kaufmann, Elsevier, 2017
  - □ Andrew S. Tanenbaum: STRUCTURED COMPUTER ORGANIZATION, Sixth Edition, Pearson Prentice Hall, 2013
  - □ Slides on http://ucilnica.fri.uni-lj.si