Cache memory

course “Essentials of computing systems”

Feb.2022

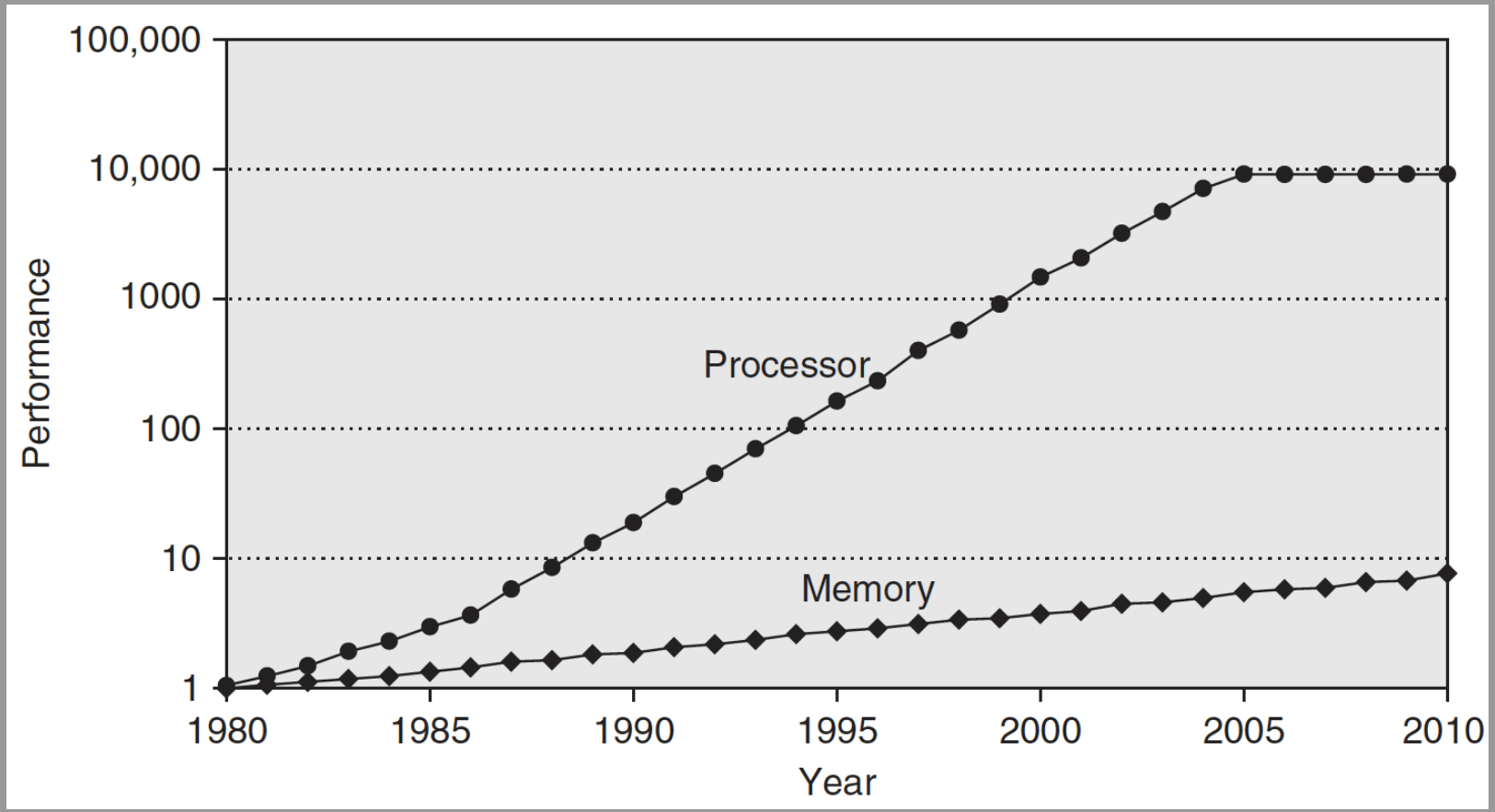
©João M. Fernandes

contents

contents

processor-memory performance gap

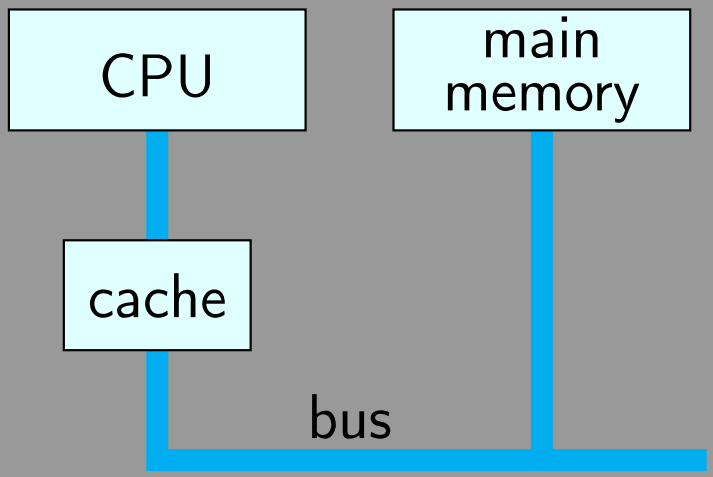
a.k.a. processor-memory bottleneck, memory wall, CPU-memory speed mismatch



Hennessy and Patterson, Computer architecture: A quantitative approach, 5th ed., Morgan Kaufmann, 2012.

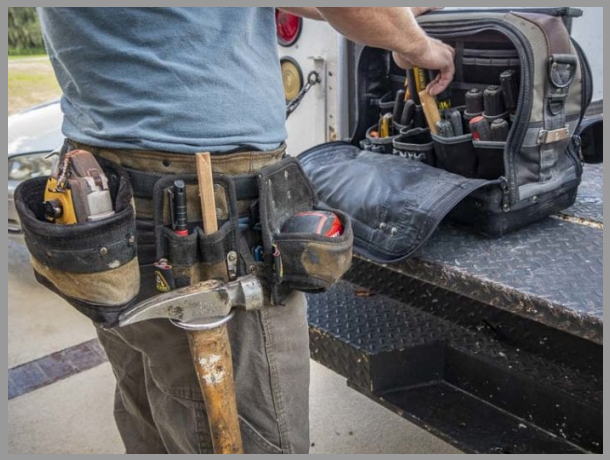
CPUs faster than memories

- Whenever the CPU needs to access the main memory, it must wait for many CPU cycles.
- This degrades the performance of the computer.



- A typical solution to the processor-memory gap is to design a computer to have cache memory.
- Caches are designed to combine the access time of expensive and fast memory with the large size of cheap but slow memory.
- The cache speeds up memory accesses by storing recently used data closer to the CPU, instead of storing it in the memory.
- Cache memory is a small, high-speed type of memory that serves as a buffer for frequently accessed data.

non-computer examples



definition

- cache: high-speed memory used to store frequently accessed or recently accessed data and instructions, so that future requests for that data/instructions can be provided faster.
- Caching is the storage of duplicate copies of digital information in places that are local, fast, and convenient so that the data can be accessed quickly if needed again rather than retrieved from the ultimate source.

cache principle

- main principle: the most frequently requested memory words are stored in the cache.
- Whenever the CPU needs a given word, it first looks in the cache.
- Most of the times, the needed word is expected to be in the cache.
- If this is not the case, then the CPU must access the main memory to get it.
- If a substantial percentage of the words are in the cache, the average access time is greatly reduced.

locality

- The success of the cache in reducing the accesses to the memory depends on what fraction of the words are in the cache.
- The fact that programs do not access the memory completely in an arbitrary way favours the success of caches.
- The use of the memory has some degree of predictability.
- An important observation related to the execution of programs is the principle of locality.
- Programs tend to reuse data and instructions that were recently used.
- Many programs spend 90% of their execution times in 10% of the code.

locality

- One can predict which instructions and pieces of data are to be used in the near future based on the most recent accesses.
- This locality principle has two different types:
  - temporal locality – recently accessed data or instructions are likely to be needed in the near future;
  - spatial locality – items whose memory addresses are close to each other tend to be referenced close together in time.
- These two types of locality are observed during the execution of a loop that accesses all the elements of an array.
- This principle also applies to code, as instructions, by default, are fetched from consecutive positions in the memory.
- Most program execution time is spent in loops, in which the same instructions are repeatedly executed.

cache miss

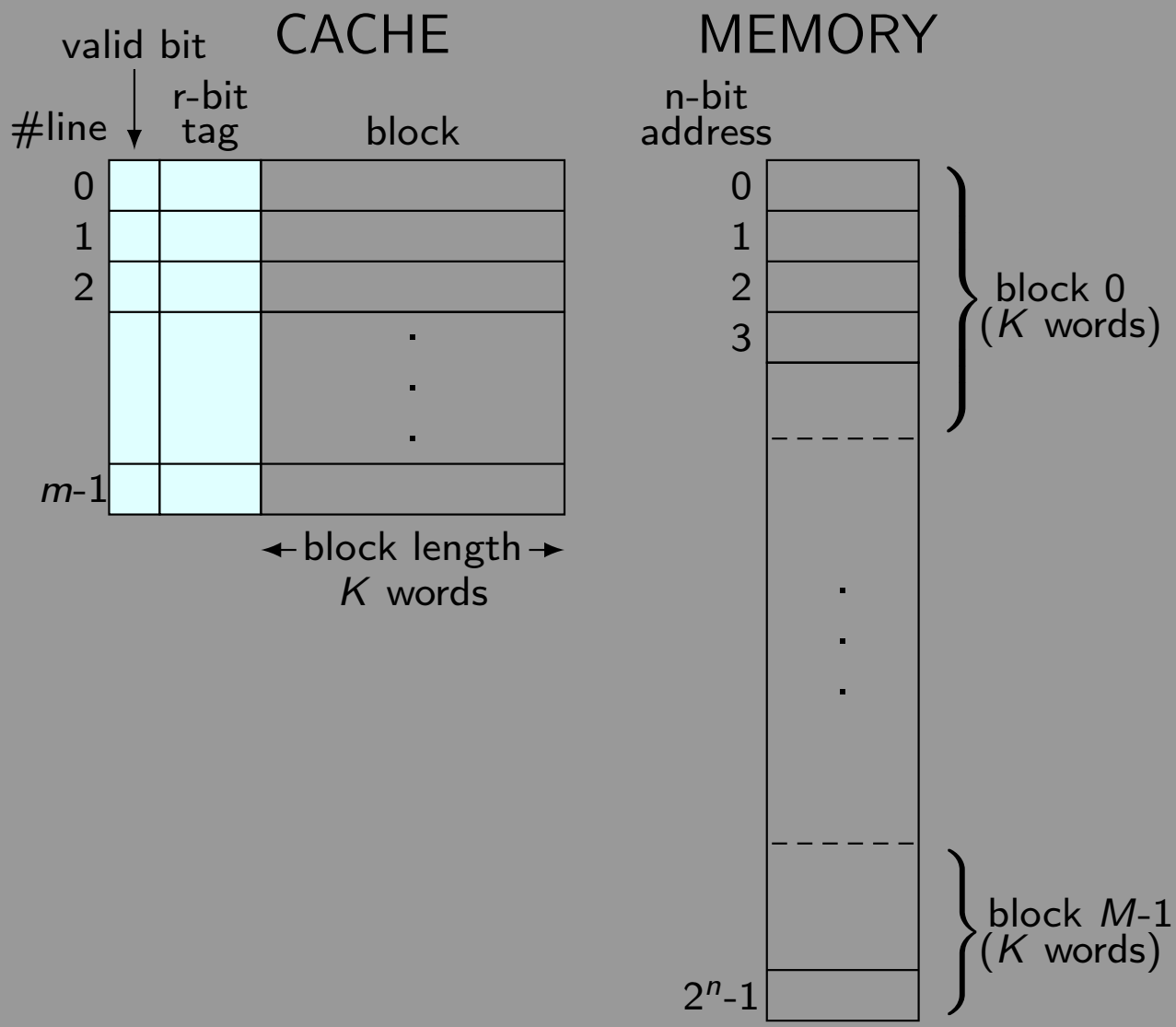
- Whenever the processor fails to obtain a given word from the cache (cache miss), the word must be read from memory.
- To take advantage of locality, the word and its neighbours are brought from the main memory into the cache.
- It is likely that many of the neighbours will be needed shortly.
- This approach is more efficient than fetching individual words, as it is faster to fetch n words at once than one word n times.

hit rate and miss rate

- A cache hit occurs when the requested data is located in the cache.
- With the number of hits and misses, it is possible to calculate the hit rate and the miss rate.
- They are the percentages of memory accesses found and not found, respectively, in the cache, during the execution of a program.
- The time required to fetch the requested data in the cache is designated as hit time.
- The miss penalty is the time required to process a miss, which includes substituting a block in the cache and the additional time to deliver the requested data to the processor.
- The time to process a hit is significantly smaller than the time to process a miss.

cache block and line

- A cache block is the minimum unit of transfer between the cache and the main memory.
- This term also refers to the physical location in the main memory and in the cache.
- A cache line is the portion of the cache memory that can store one block.
- A line contains control information, including:
  - a tag that identifies the memory address of the block stored in that line;
  - a valid bit to signal whether the data in the line is valid or not.
- When the system is booted, all cache lines are obviously marked as not valid.

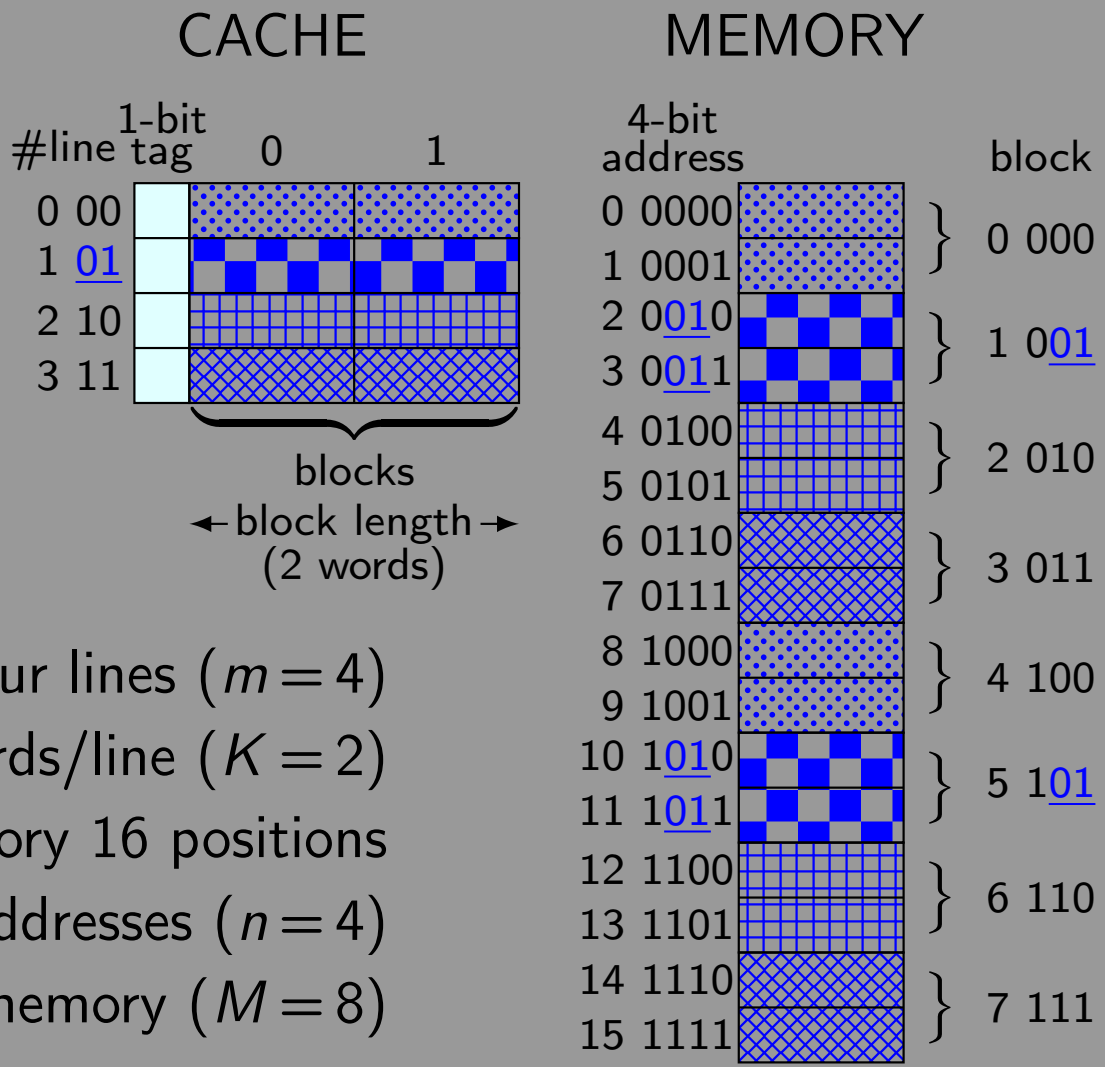

CACHE
valid bit
#line
r-bit tag
block
0
1
2
m-1
block length
K words
MEMORY
n-bit address
0
1
2
3
block 0
(K words)
block M-1
(K words)
2^n-1

contents

content-addressable memory

- Caches are not accessed by address, but instead by content.
- Cache is often called content-addressable memory (CAM).
- Both (part of) the address and the contents are stored.
- When a given memory address is being requested, all cache entries must be searched to check if it is stored there.
- When the address matches, its content is fetched from the cache memory.
- One needs a mechanism for determining which specific main memory block currently occupies a cache line.
- The selection of the mapping function determines how the cache is logically organised.
- Three techniques are usually adopted: direct, associative, and set-associative.

direct mapping



four lines ($m = 4$)

two words/line ($K = 2$)

memory 16 positions

4-bit addresses ($n = 4$)

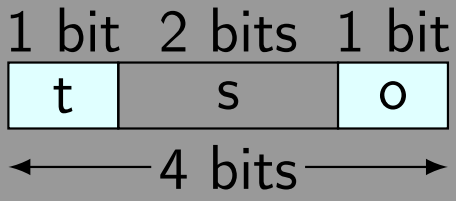
eight blocks in memory ($M = 8$)

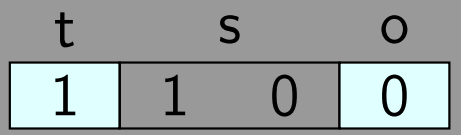
direct mapping

- Direct mapping maps each block of main memory into only one possible cache line.
- The number of the cache line $i = j \bmod m$, where j is the memory block number.
- The memory block number is given by the memory address but ignoring the $log_2 K$ LSBs.
- The mapping is expressed as $i = j \bmod 4$.
- Blocks 0 and 4 map to line 0, blocks 1 and 5 to line 1, ...
- The two last bits of the block number indicate the cache line.
- The MSB of the block number is stored in the tag of the cache line, to identify which block is stored there.
- Blocks 0, m, $2 \cdot m$, ... map to line 0.
- Blocks 1, $m+1$, $2 \cdot m+1$, ... map to line 1, ...

direct mapping

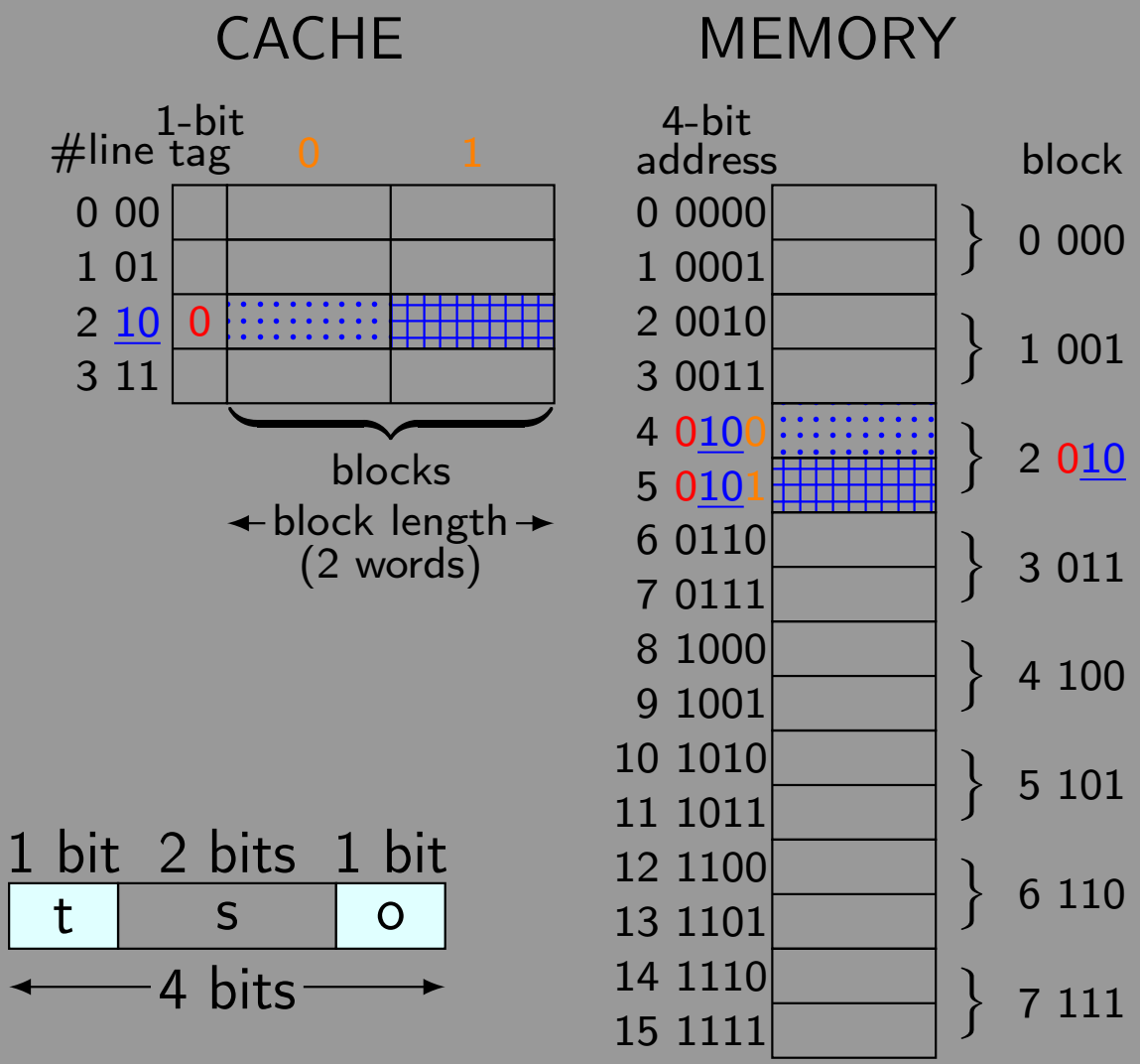
- The direct mapping function can be implemented using the main memory address.
- For the purpose of cache access, each main memory address can be viewed as consisting of three fields:
  1. The o LSBs identify a unique word or byte within a memory block ($o = \log_2 K$).
  2. The next s bits specify part of the block number ($s = \log_2 m$).
  3. The remaining t bits are saved in the tag ($t = n\text{-}s\text{-}o$).
- $o = \log_2 2 = 1$, $s = \log_2 4 = 2$, and $t = 4\text{-}2\text{-}1 = 1$.



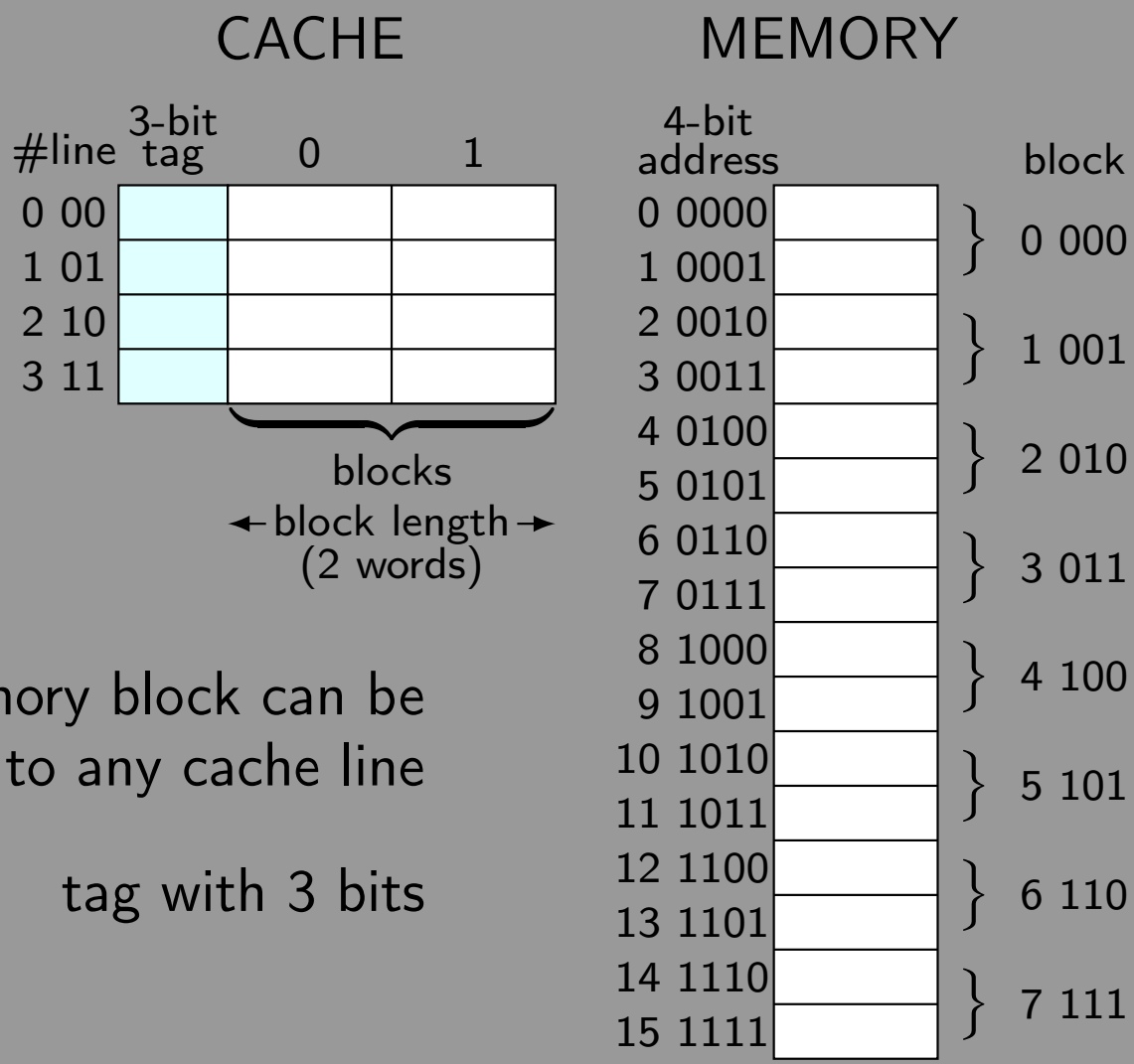


mapped to cache line 10_2
tag is 1

direct mapping



fully associative mapping

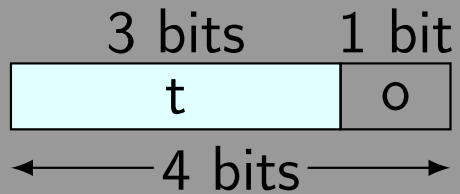


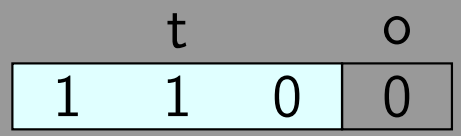
each memory block can be mapped to any cache line

tag with 3 bits

fully associative mapping

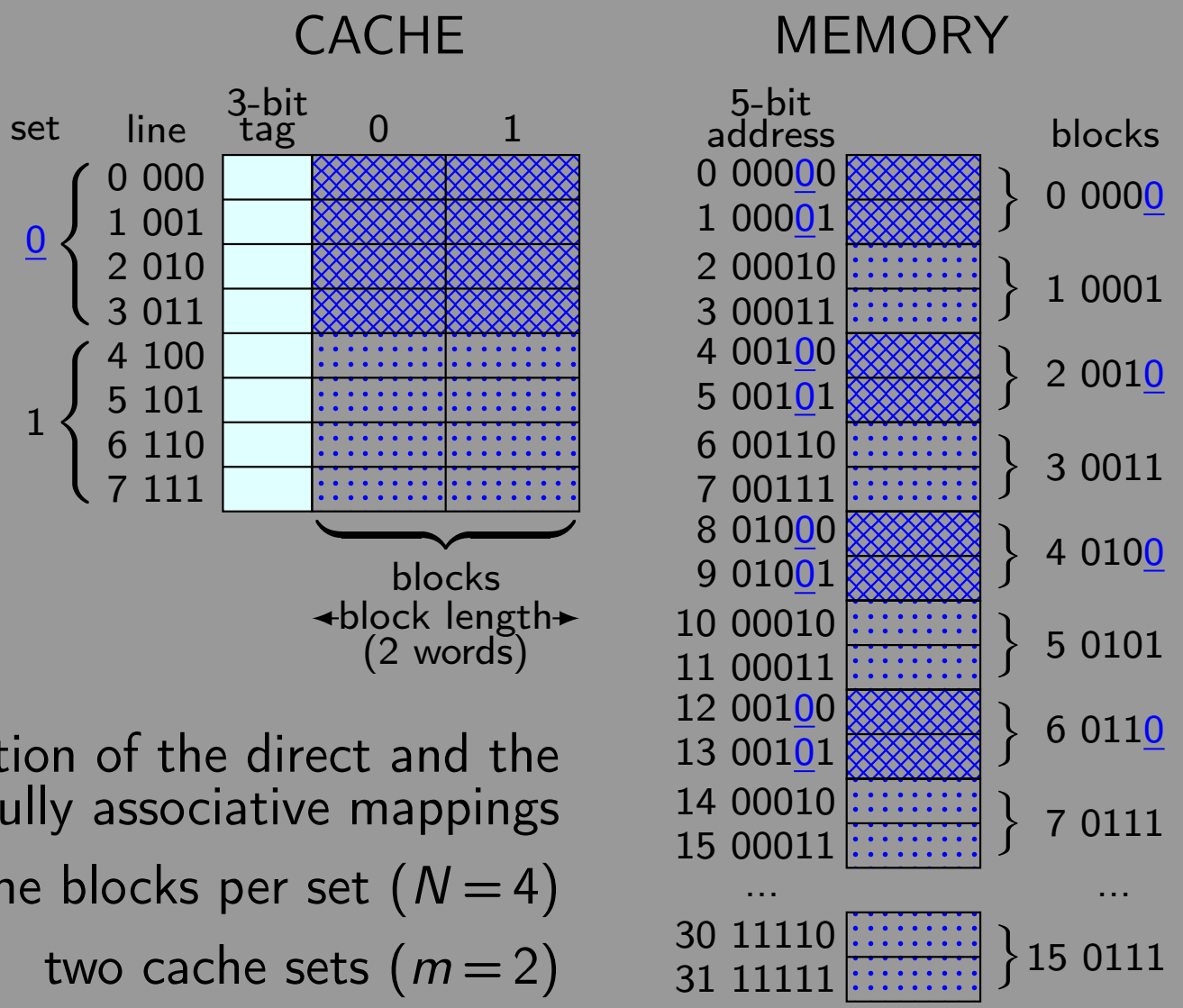
- For the purpose of cache access, each main memory address can be viewed as consisting of two fields:
  1. The o LSBs identify a unique word or byte within a memory block ($o = \log_2 K$).
  2. The remaining t bits are saved in the tag ($t = n\text{-}o$).
- $o = \log_2 2 = 1$ and $t = 4\text{-}1 = 3$.





mapped to any cache line
tag is 110_2

set associative mapping



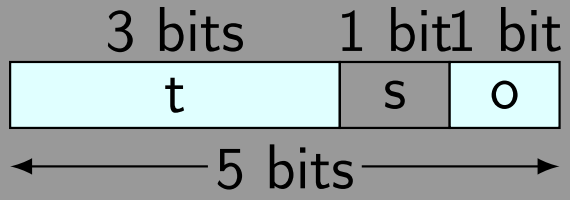
Combination of the direct and the fully associative mappings

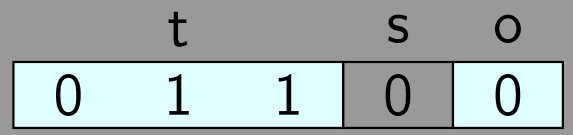
four cache blocks per set ($N = 4$)

two cache sets ($m = 2$)

set associative mapping

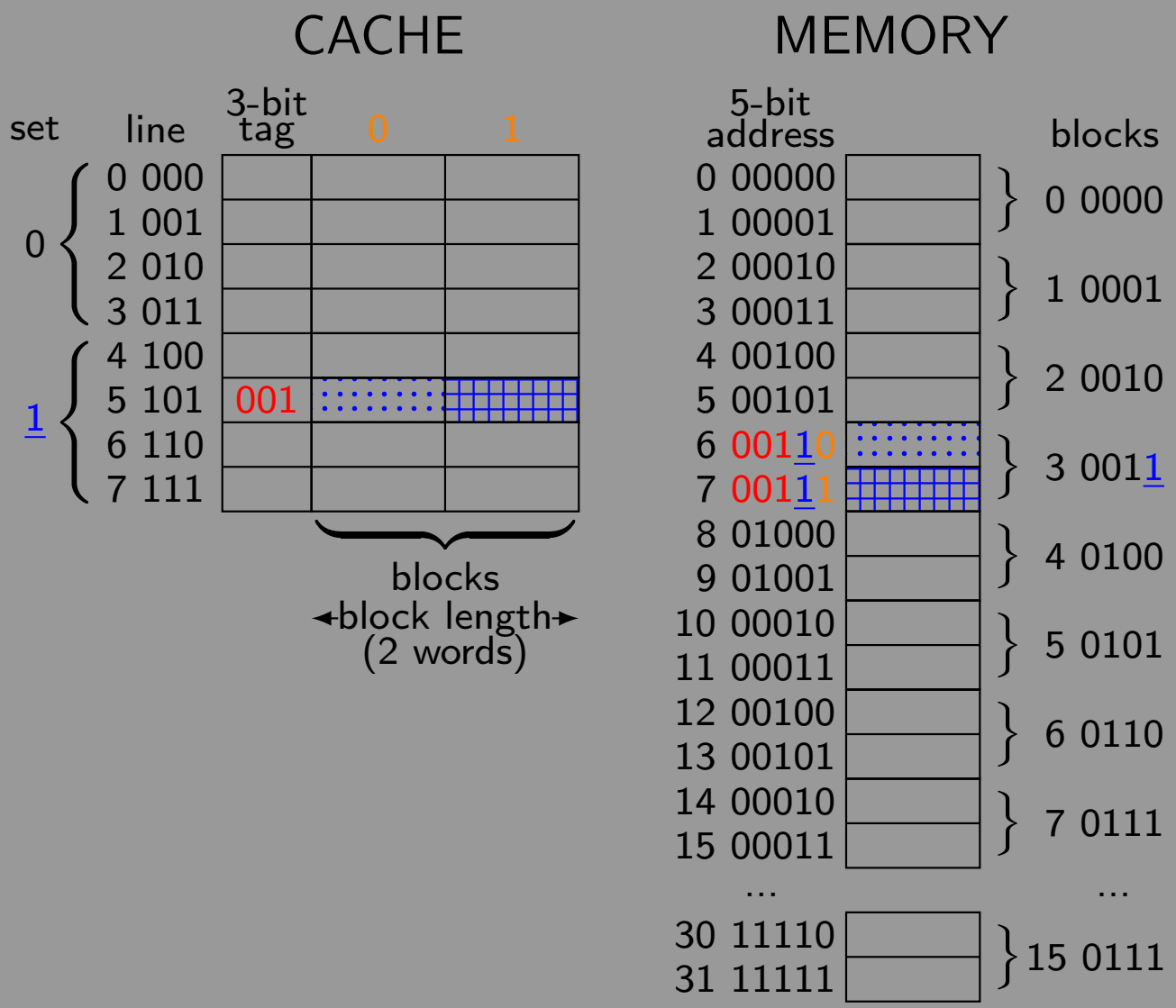
- Direct mapped cache can be seen as a special case of N-way set associative cache mapping where the set size is one.
- The 2-way set associative mapping function can be implemented using the main memory address.
- Each memory address can be viewed as consisting of three fields:
  1. The o LSBs identify a unique word or byte within a memory block ($o = \log_2 K$).
  2. The next s bits specify the set number ($s = \log_2 m$).
  3. The remaining t bits are saved in the tag ($t = n\text{-}s\text{-}o$).
- $o = \log_2 2 = 1$, $s = \log_2 2 = 1$, and $t = 5\text{-}1\text{-}1 = 3$.





mapped to set 0
tag is 011

set associative mapping



contents

replacement

- In a direct-mapped cache, if there is contention for a cache block, there is only one possible action.
- The existing block is removed from the cache to make room for the new block.
- This process is called replacement.
- With direct mapping, there is no need for a sophisticated replacement algorithm.
- With direct mapping, if a block is about to be replaced, it should be updated in the main memory if it has been modified after being loaded into the cache.

replacement algorithm

- For associative mappings, whenever the cache is full and a new block needs be stored, a replacement algorithm is needed.
- It decides in which cache line to store the new block.
- Many alternatives exist for this algorithm.
- If temporal locality is considered, it is likely that any value that has not been used recently will not be needed again soon.
- If one registers the time a cache line is accessed, one can select for replacement the line that has been used least recently.
- This is the least recently used (LRU) algorithm, which requires to keep a history of accesses for every cache line.

replacement algorithm

- Another possible approach is the first in first out (FIFO) strategy.
- The block that has been in cache the longest is selected as the one to be replaced by the new block.
- Another alternative is to randomly select the cache line to be replaced.
- A possible option is the not most-recently used (NMRU) policy that randomly chooses among all blocks but the one most recently referenced.
- The least frequently used (LFU) policy replaces the block that has been referenced the fewest number of times.
- The algorithm selected by the computer architects often depends on how the system is to be used.
- This selection is not trivial as no single algorithm is the best for all contexts.