Performance Parameters (1)

- H - hit ratio
  - percentage of the cases when the requested location was found in the cache
- M - miss ratio
  - percentage of the cases when the requested location was not found in the cache

$$0 \le H, M \le 1$$

$$M = 1 - H$$

Performance Parameters (2)

- T_c - access time of the cache
- T_m - access time of the memory on a cache miss
- T - average access time of the memory (with cache)
- T_p - access time of the main memory (without cache)

Cache Memory Performance (1)

$T = T_c \cdot H + T_m \cdot M$

- if $T < T_p \rightarrow$ speed is improving
- T - must be computed statistically
- extreme cases
  - H=100% (M=0): $T = T_c \rightarrow$ ideally
  - H=0 (M=100%): $T = T_m \rightarrow$ speed is decreasing ($T_m > T_p$)

Cache Memory Performance (2)

Real life - example

- $T_c = 2$ ns
- $T_p = 10$ ns
- $T_m = 11$ ns
- $H = 95\%$
- $T = 2.45$ ns $= 0.245\ T_p \rightarrow$ access speed improves over 4 times

Cache Addressing Issues

- the address in the cache is not the same as the address in the main memory
- but it is the address in the main memory that is known by the processor
- conclusion - the cache must also keep the addresses of the locations in the main memory

Cache Lines

- the cache makes use of temporal locality
- how could we also exploit spatial locality?
- when a location is brought to cache, its neighboring locations are also brought along - *cache line*

Replacement Policy

- the cache is small - quickly becomes full
- new lines brought to cache - older lines must be eliminated
  - elimination - writing back to the main memory
- which lines must be eliminated?
  - the goal - increasing the overall speed
- those lines that will not be accessed in the near future !

Improving Performance

- two items are essential
  - access time of the cache (T_c)
  - hit ratio (H)
- cannot be optimized both at the same time
- influenced by
  - technology
  - replacement policy

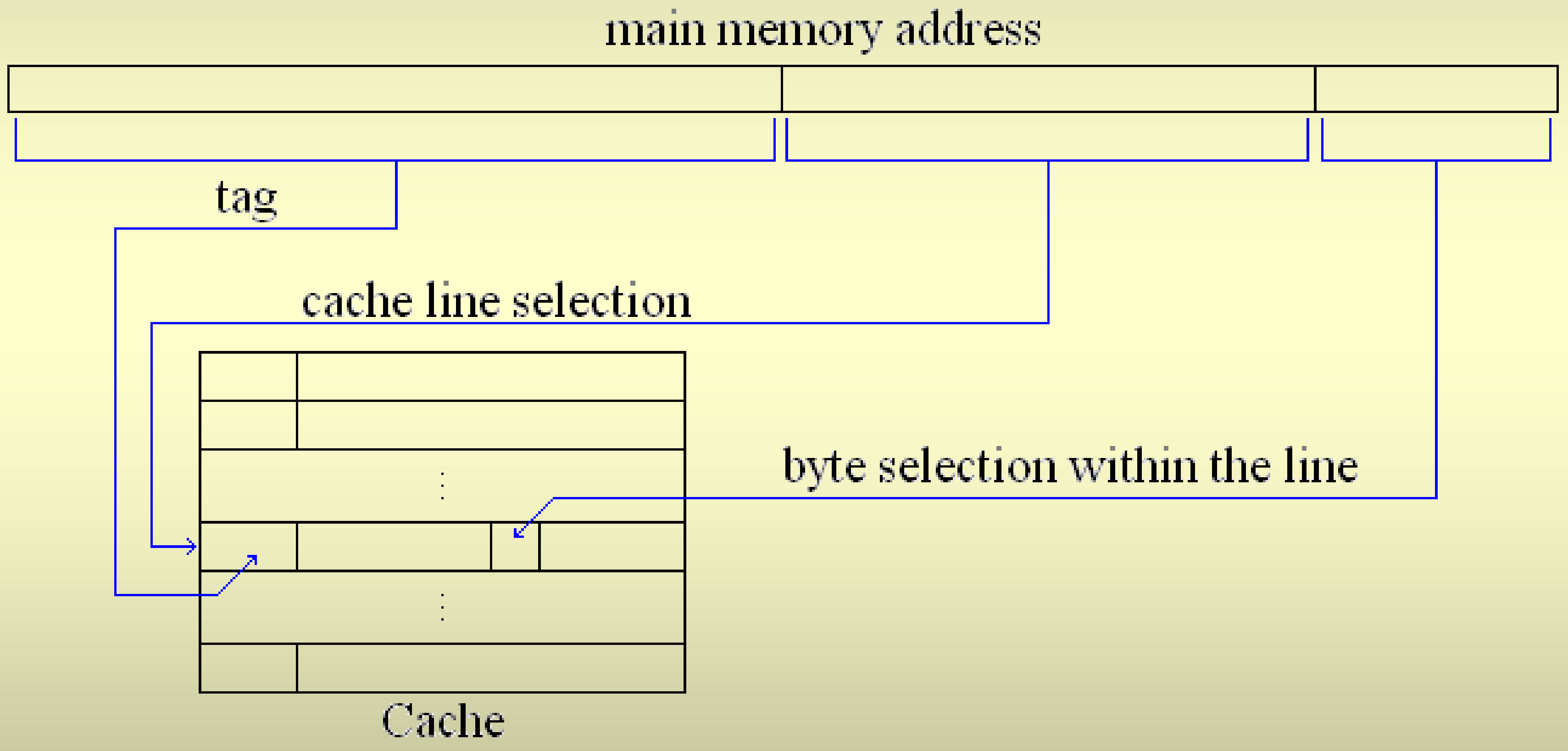
Cache - Construction Types

- direct mapped cache
- fully associative cache
- set associative cache

Direct Mapped Cache (1)

- placing a location in the cache
  - the cache line is always the same
  - depends on the address in the main memory
- the address in the main memory - 3 parts
  - the tag - stored in the cache
  - the cache line selector
  - the byte selector within the line

Direct Mapped Cache (2)



Direct Mapped Cache (3)

Example

- the address in the main memory - 32 bits
- cache size: 2^{11} lines × 2^{5} bytes/line
- the address in the main memory is split into
  - the cache line selector - 11 bits
  - the byte selector within the line - 5 bits
  - the tag - 16 bits (= 32 – 11 – 5)

Direct Mapped Cache (4)

Example

- address in the main memory: $45097373_{(10)}$

$00000010101100000010000110011101_{(2)}$

- the tag: $0000001010110000_{(2)}=688_{(10)}$
- the cache line : $0010000110 0_{(2)}=268_{(10)}$
- the byte within the line: $11101_{(2)}=29_{(10)}$

Direct Mapped Cache (5)

Example

- which addresses from the main memory are brought in the cache line?

$000000101011000000100001100xxxxx_{(2)}$

$00000010101100000010000110000000 \div$

$0000010101100000010000110011111_{(2)}$

$45097344 \div 45097375_{(10)}$

Direct Mapped Cache (6)

The contents of a cache line

- a bit indicating whether the line contains valid data
  - at first, all lines are empty, so they are invalid
- the tag
- the data bytes, brought from the main memory

Direct Mapped Cache (7)

Advantages

- simple implementation
- short access time (T_c)

Drawbacks

- lacks flexibility
- inefficient replacement policy - lower hit ratio (H)

Direct Mapped Cache (8)

Example

```
for(i=0;i<1000;i++) a=a+i;
```

- addresses: `i`$\rightarrow$3806240, `a` $\rightarrow$1756566571
- both stored in the cache in line 161
- alternative accesses $\rightarrow$ frequent replacements in the cache $\rightarrow$ lots of misses

Fully Associative Cache (1)

- built with associative memory circuits
  - "classic" memory - access a location based on its address
  - associative memory - also allows finding the location based on its contents
  - implementation - the requested value is compared in parallel to all locations
    - why in parallel?

Fully Associative Cache (2)

Advantages

- placing the data from the main memory - in any cache line
- can choose conveniently the addresses brought into the cache line
- can implement efficient replacement policies - high hit ratio

Fully Associative Cache (3)

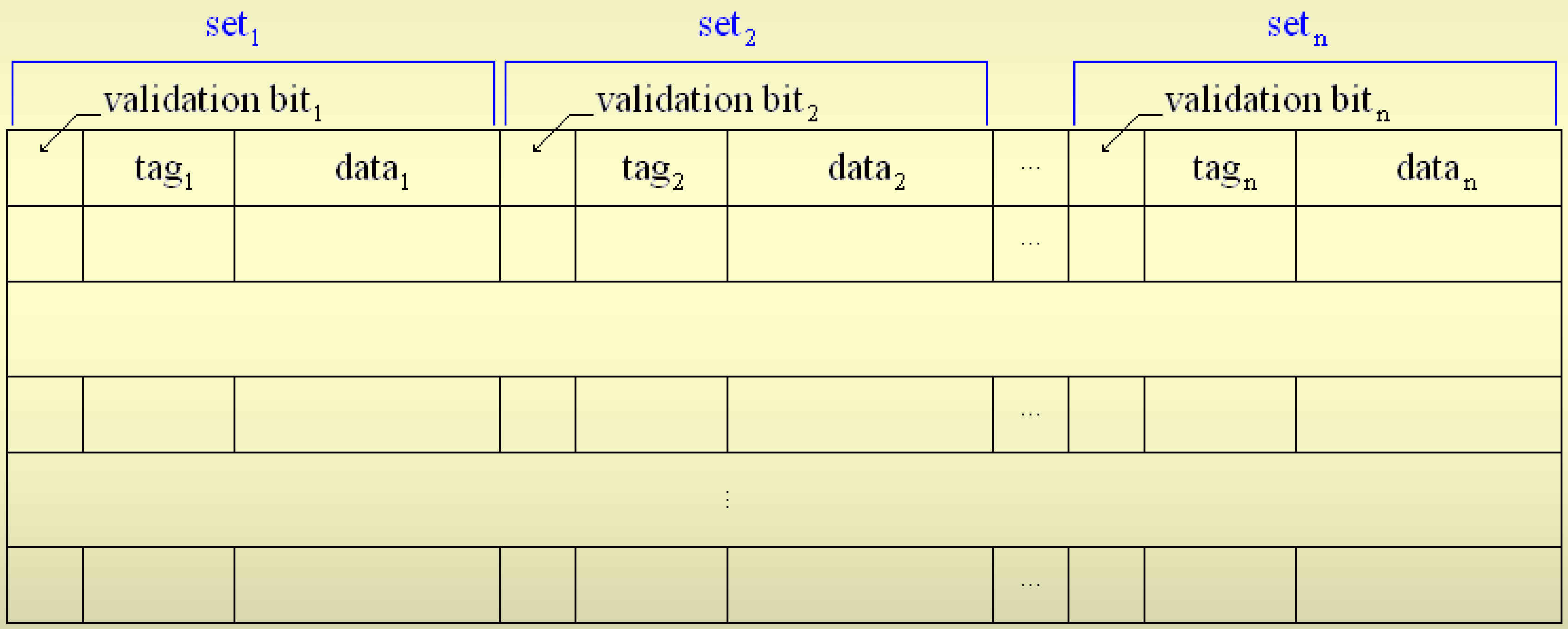
Drawbacks

- long access time
  - associative memories - slow
  - complex replacement algorithms - additional time consumed
- complex hardware necessary for the associative memories and for the replacement algorithms

Set Associative Cache (1)

- derived from the direct mapped cache
- each cache line contains several data sets (4, 8, 16, ...)
- the structure of such a set
  - validation bit
  - tag
  - data

Set Associative Cache (2)



Set Associative Cache (3)

Access time (T_c)

- a little longer than the direct mapped cache
  - all *n* sets must be checked

Hit ratio (H)

- high
  - eliminates the overlaps that impair the performance of the direct mapped cache

Writing to the Cache (1)

- writing to a location which is currently not stored in the cache
- where to write?
- variants
  - to the main memory only - not a good idea (why?)
  - to the cache only (*write-back*)
  - both to the cache and to the main memory (*write-through*)

Writing to the Cache (2)

Write-back cache

- write to the cache only
- data is written to the main memory only when the line is eliminated from the cache
- high speed
- raises problems in multi-processor systems

Writing to the Cache (3)

Write-through cache

- write both to the cache and to the main memory
- slower
  - because of the access to the main memory
- both types of cache are widely used

Cache - Extending the Concept

- can be applied not only to processors
- class of problems: communication to a slow, high-capacity entity (information source)
- solution: use a lower-capacity, higher-speed entity
  - placed before the main entity
  - keeps the last data that has been accessed

Where Else Can We Use the Idea?

Applicable

- wherever the locality laws work
- hardware
- software

Examples (1)

Disk caches

- 2 directions
  - hardware - memory circuit integrated inside the controller
  - software - a part of the system memory
- bigger, slower entity - the disk
- smaller, faster entity - the memory

Examples (2)

Web browsers

- the last pages that have been accessed are stored on the local disk
  - only temporal locality - why?
- bigger, slower entity - the network (Internet)
- smaller, faster entity - the disk