ARCOS Group

uc3m | Universidad Carlos III de Madrid

L5: Memory hierarchy (2)

Computer Structure

Bachelor in Computer Science and Engineering
Bachelor in Applied Mathematics and Computing
Dual Bachelor in Computer Science and Engineering and Business Administration

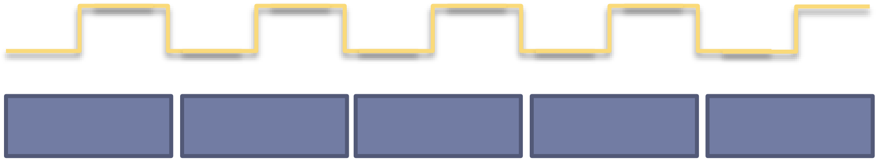
cc BY NC SA

Contents

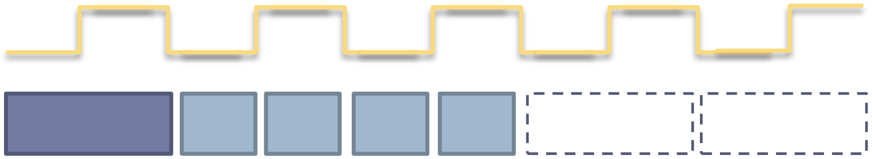
1. Types of memories
2. Memory hierarchy
3. Main memory
4. Cache memory
   1. Introduction
   2. Structure of the cache memory
   3. Cache design and organization
5. Virtual memory

Main memory characteristics

- It is better to access to consecutive words
  - Example 1: access to 5 individuals non-consecutives words



  - Example 2: access to 5 consecutives words



Characteristics of memory accesses

- “Principle of proximity or locality of references”:

  During the execution of a program, references (addresses) to memory tend to be grouped by:

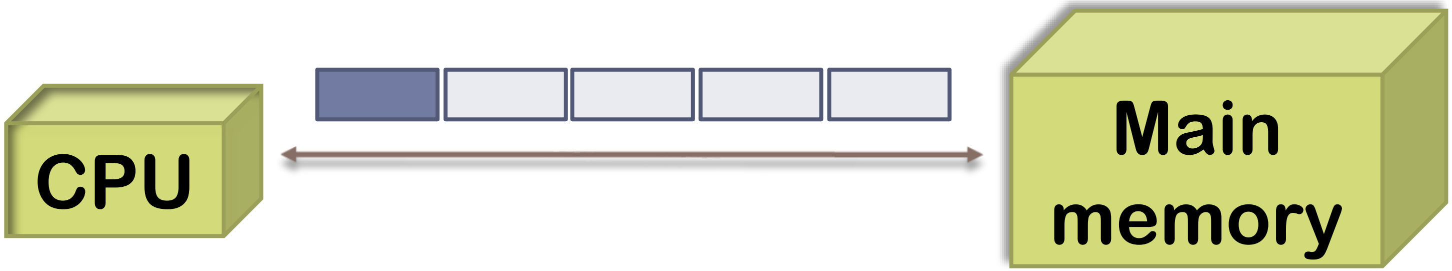
  - Spatial proximity
    - Sequence of instructions
    - Sequential access to arrays
  - Temporal proximity
    - loops

```
.data
  vector: .space 4*1024

.text
 main:      li   t0 0
            la   t1 vector
            li   t3 1024
            li   t4 4
      b2:   bge  t0 t3 fb2
            mul  t2 t0 t4
            add  t2 t1 t2
            sw   t0 0(t2)
            addi t0 t0 1
            j    b2
      fb2:  jr ra
```

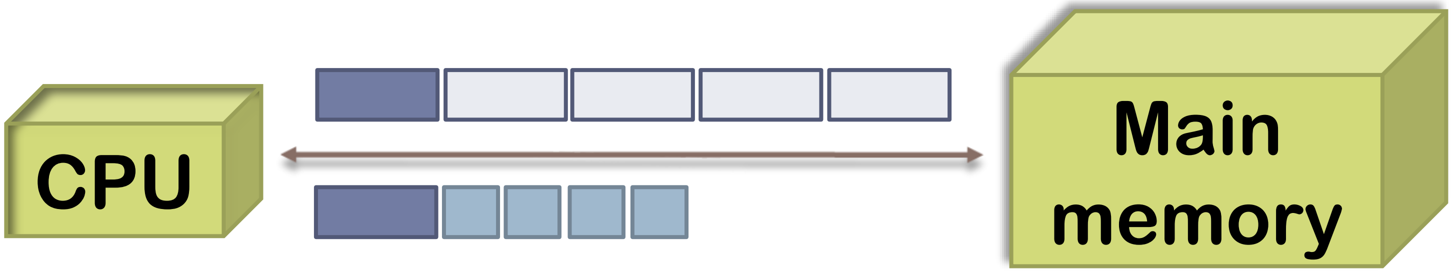
Goal of the cache memory:
to take advantage of contiguous accesses

- If when accessing a memory location only the data of that location is transferred, possible accesses to contiguous data are not taken advantage of.



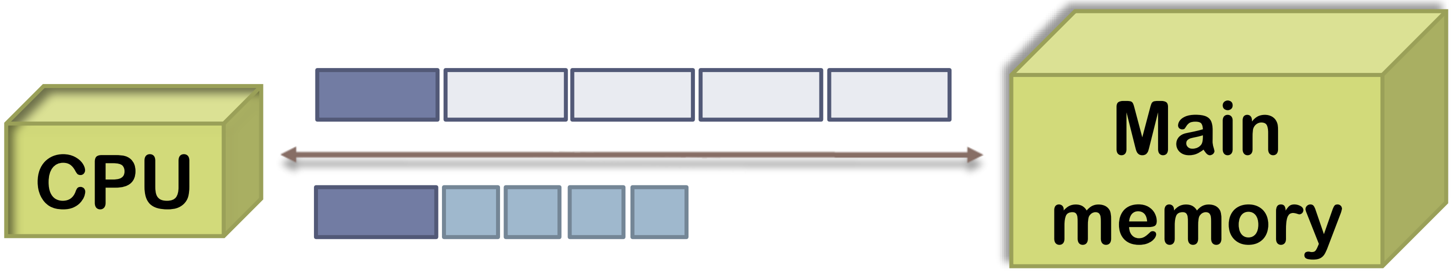
Goal of the cache memory:
to take advantage of contiguous accesses

- If, when accessing a memory location, this data and the contiguous data are transferred, the access to contiguous data is exploited



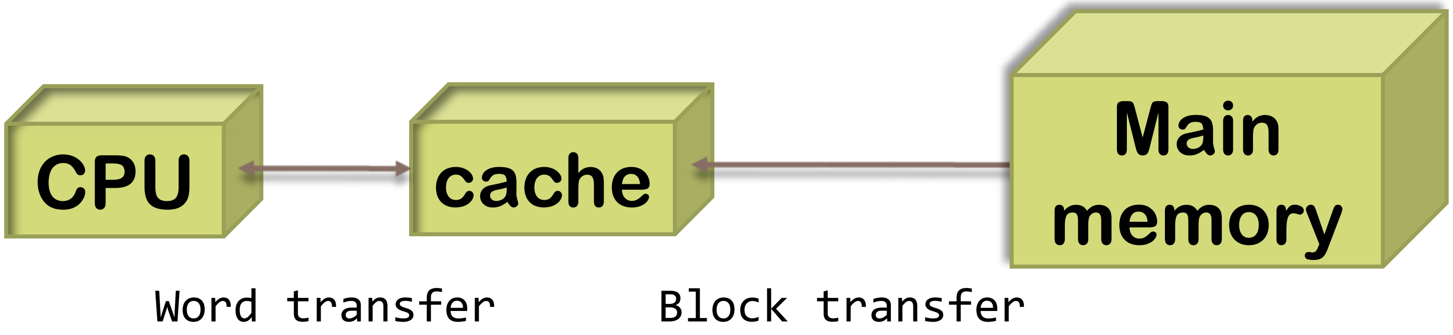
Goal of the cache memory:
to take advantage of contiguous accesses

- If, when accessing a memory location, this data and the contiguous data are transferred, the access to contiguous data is exploited
  - I transfer from the main memory a block of words
  - Where are the words of the block stored?



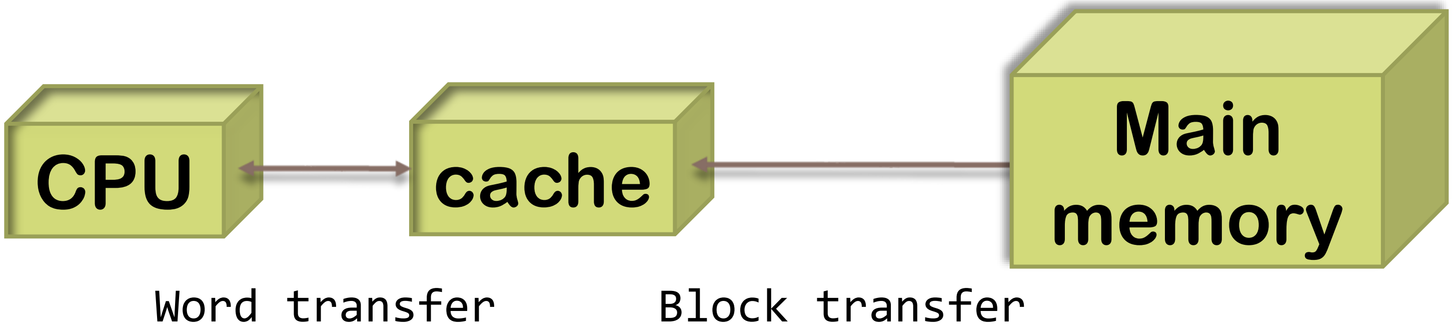
Cache memory

- Small amount of fast SRAM memory
  - Integrated in the Processor itself
  - Faster and more expensive than the DRAM
- Between main memory and processor
- Stores a copy of chunks of the main memory



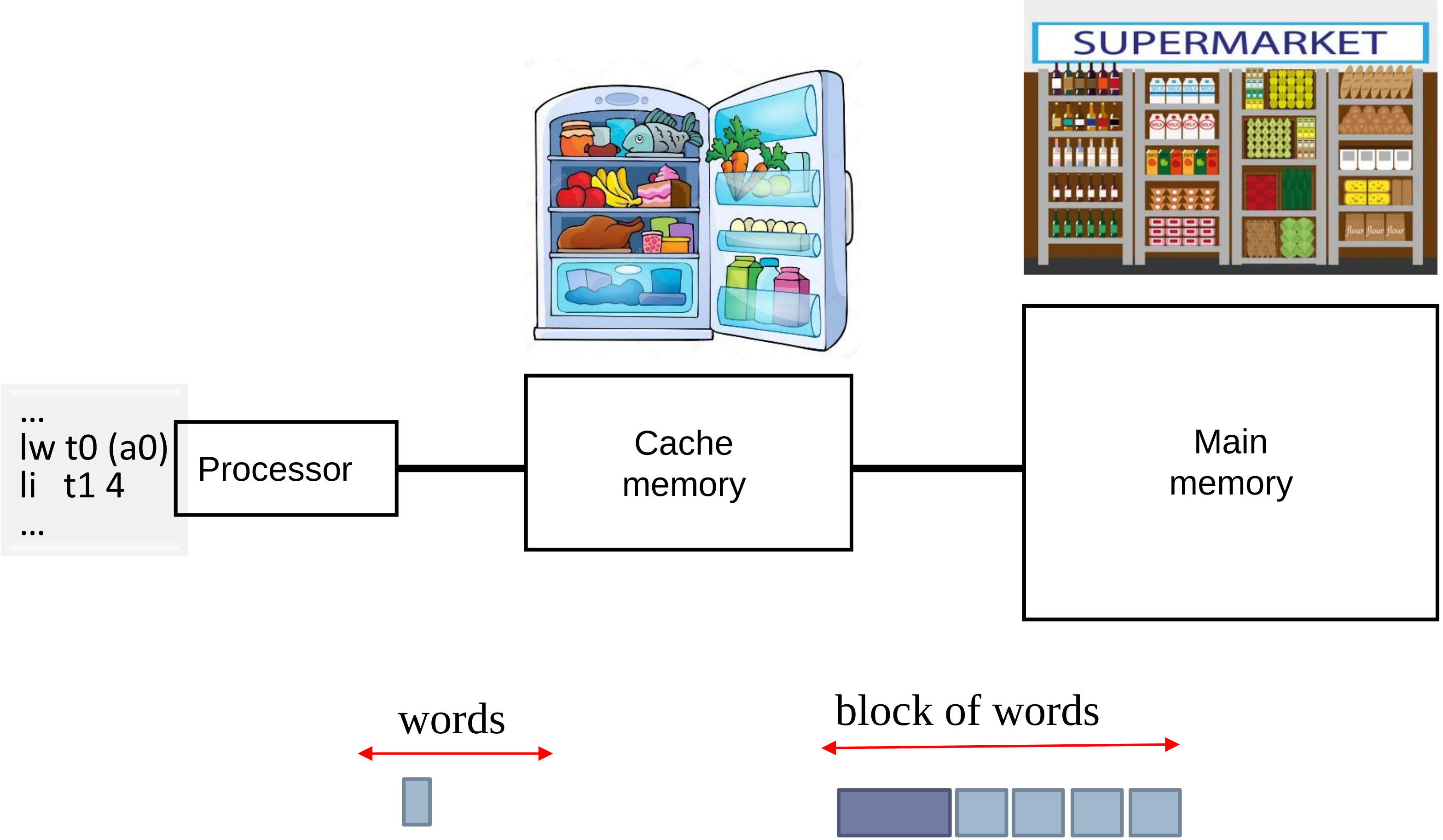
Example of access time

- Main memory (DRAM or similar)
  - Access time: between 20 and 50 ns.
- Cache memory (SRAM or similar)
  - Access time: between 1 and 2.5 ns.



Cache memory

metaphor of supermarket and refrigerator

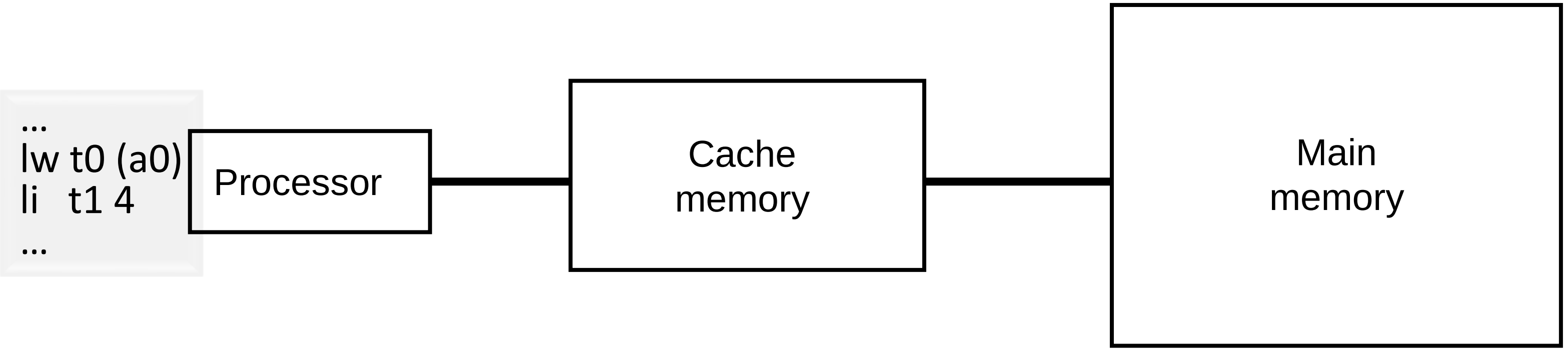


```
…
lw t0 (a0)
li   t1 4
…
```

https://sp.depositphotos.com/55141849/stock-illustration-image-with-fridge-theme-1.html
https://sp.depositphotos.com/77650478/stock-illustration-groccery-store.html

How cache memory works

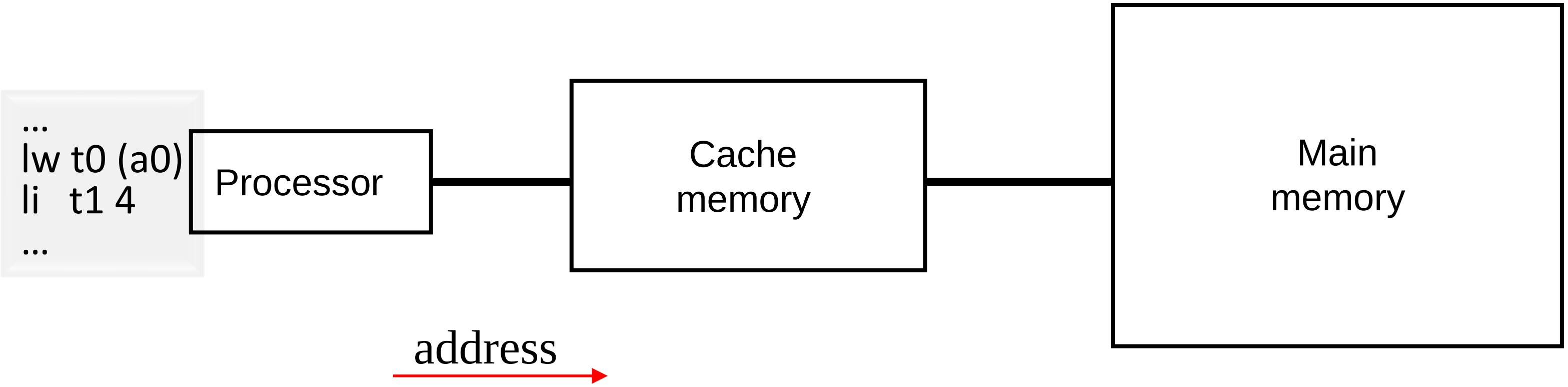
1. The processor requests the contents of a memory location.
2. The cache checks if the data for this position is already there:
   - **IF it is there (HIT)**
     3.A.1 It is served to the Processor from the cache (quickly): Ta.
   - **IF it is not there (MISS)**
     3.B.1 The cache transfers from Main memory the block associated with position: Tf
     3.B.2 The cache then delivers the requested data to the processor: Ta.



```
…
lw t0 (a0)
li   t1 4
…
```

How cache memory works

1. The processor requests the contents of a memory location.
2. The cache checks if the data for this position is already there:
   - IF it is there (HIT)
     - 3.A.1 It is served to the Processor from the cache (quickly): Ta.
   - IF it is not there (MISS)
     - 3.B.1 The cache transfers from Main memory the block associated with position: Tf
     - 3.B.2 The cache then delivers the requested data to the processor: Ta.



```
…
lw t0 (a0)
li   t1 4
…
```

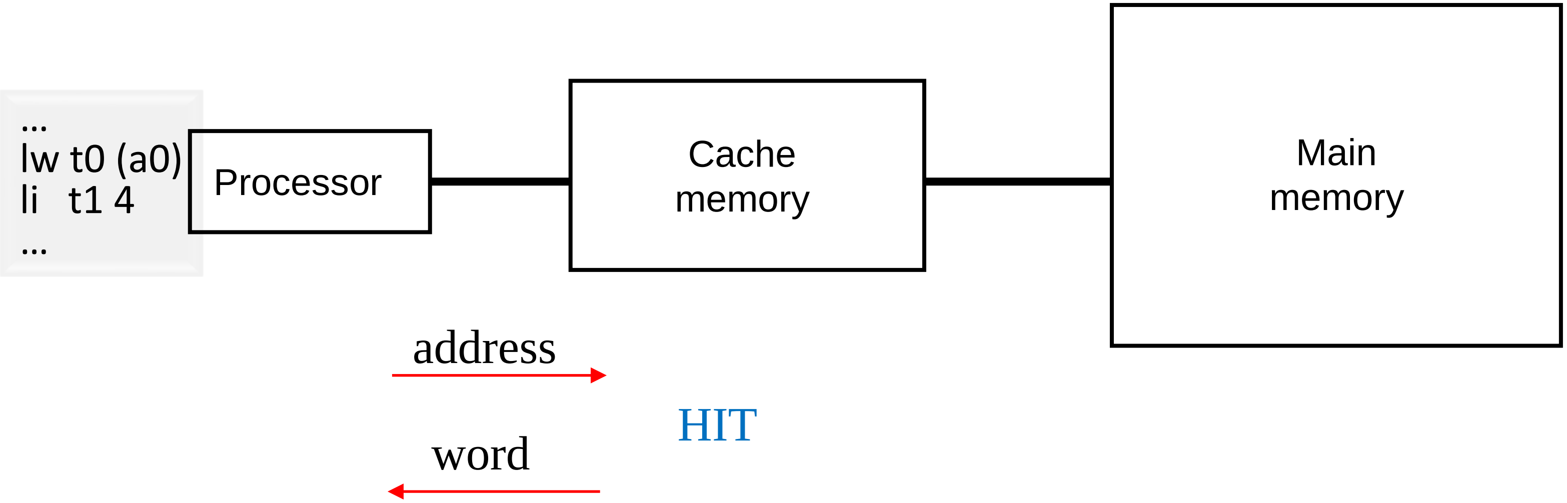
How cache memory works

1. The processor requests the contents of a memory location.
2. The cache checks if the data for this position is already there:
   - **IF it is there (HIT)**

     3.A.1 It is served to the Processor from the cache (quickly): Ta.
   - **IF it is not there (MISS)**

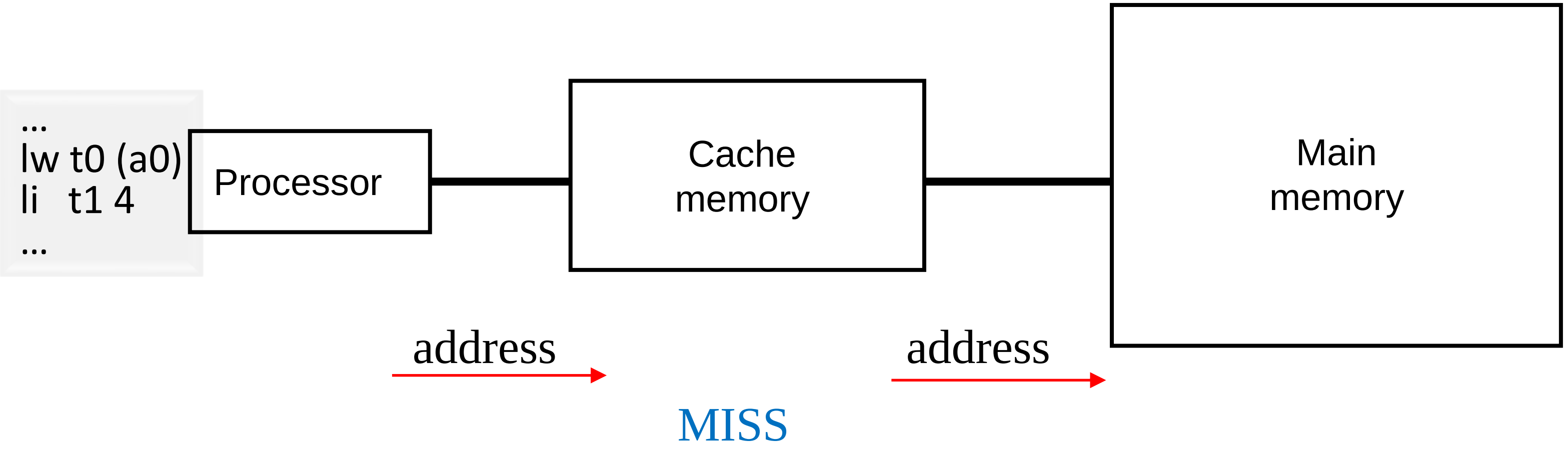
     3.B.1 The cache transfers from Main memory the block associated with position: Tf

     3.B.2 The cache then delivers the requested data to the processor: Ta.



How cache memory works

1. The processor requests the contents of a memory location.
2. The cache checks if the data for this position is already there:
   - IF it is there (HIT)
     3.A.1 It is served to the Processor from the cache (quickly): Ta.
   - **IF it is not there (MISS)**
     3.B.1 The cache transfers from Main memory the block associated with position: Tf
     3.B.2 The cache then delivers the requested data to the processor: Ta.


```
…
lw t0 (a0)
li   t1 4
…
```

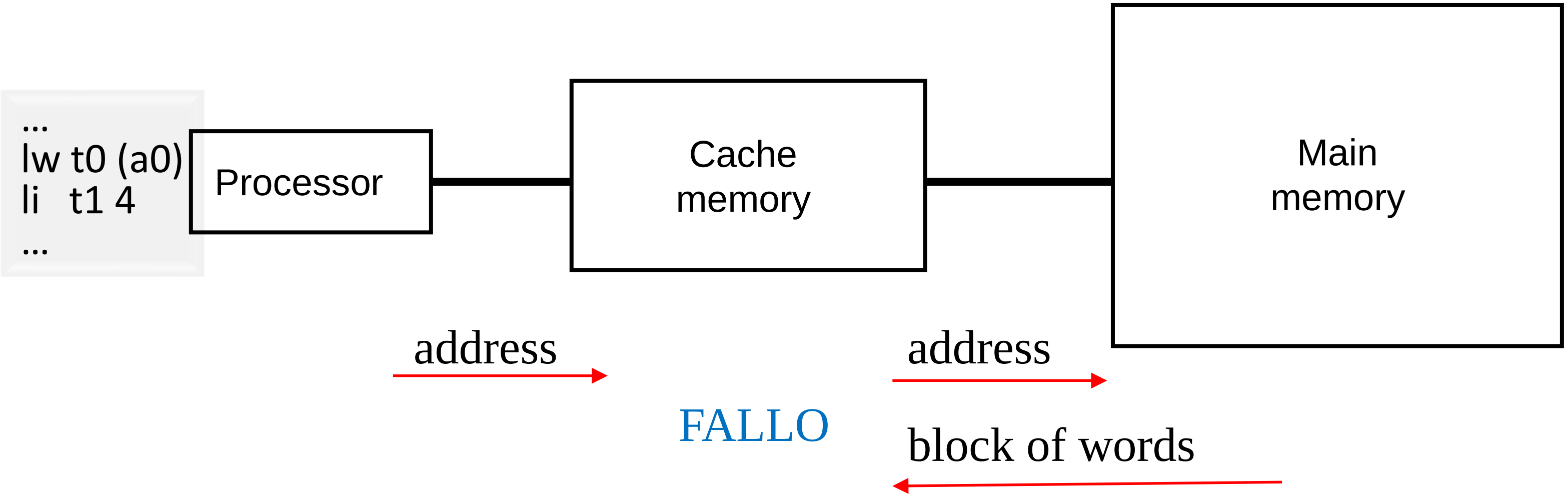
How cache memory works

1. The processor requests the contents of a memory location.
2. The cache checks if the data for this position is already there:
   - IF it is there (HIT)

     3.A.1 It is served to the Processor from the cache (quickly): Ta.
   - **IF it is not there (MISS)**

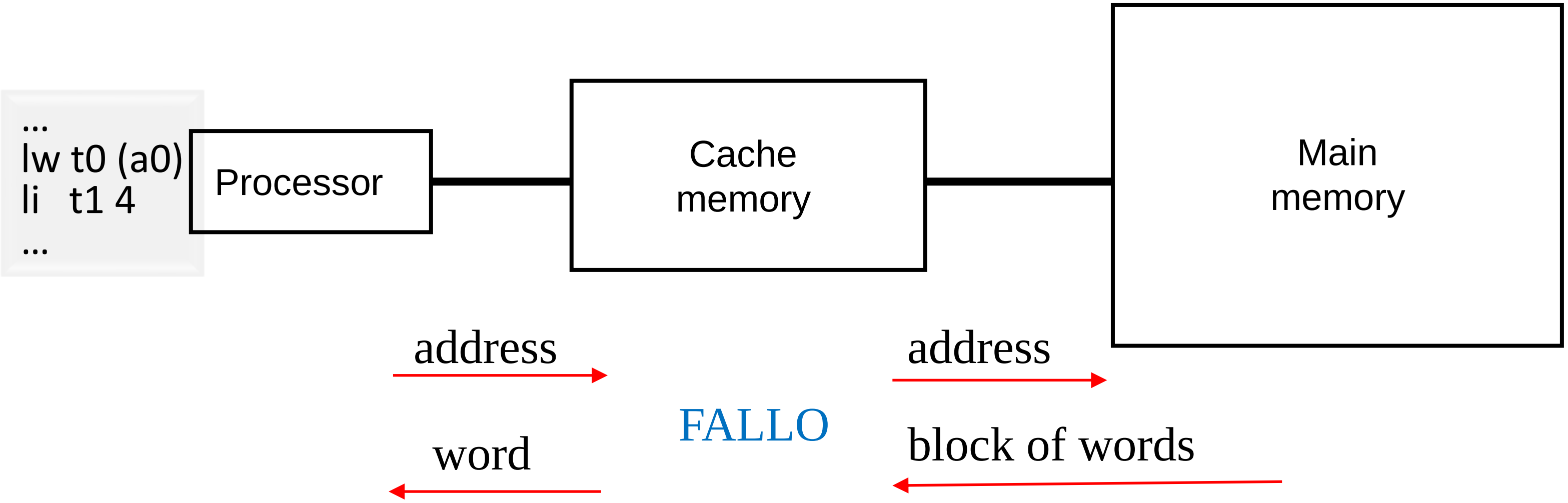
     3.B.1 The cache transfers from Main memory the block associated with position: Tf

     3.B.2 The cache then delivers the requested data to the processor: Ta.



How cache memory works

1. The processor requests the contents of a memory location.
2. The cache checks if the data for this position is already there:
   - IF it is there (HIT)
     3.A.1 It is served to the Processor from the cache (quickly): Ta.
   - **IF it is not there (MISS)**
     3.B.1 The cache transfers from Main memory the block associated with position: Tf
     3.B.2 The cache then delivers the requested data to the processor: Ta.



```
...
lw t0 (a0)
li   t1 4
...
```

Operation of the cache memory

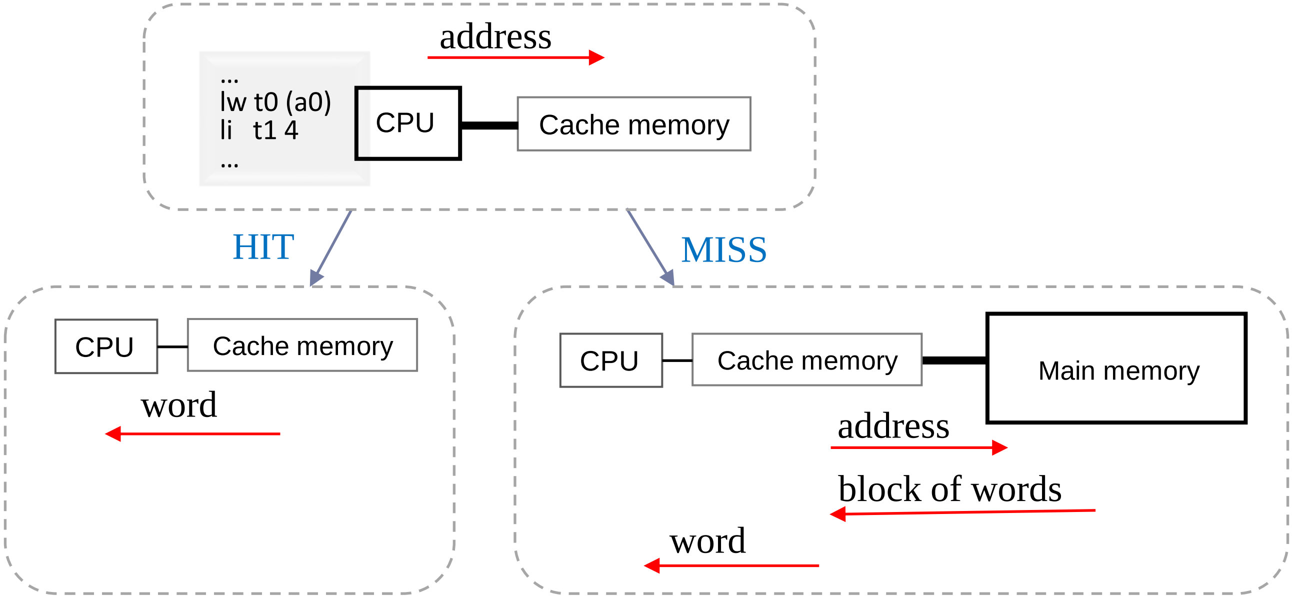
summary

1. The processor requests the contents of a memory location.
2. The cache checks if the data for this position is already there:
   - **IF it is there (HIT)**

     3.A.1 It is served to the Processor from the cache (quickly): Ta.
   - **IF it is not there (MISS)**

     3.B.1 The cache transfers from Main memory the block associated with position: Tf

     3.B.2 The cache then delivers the requested data to the processor: Ta.



```
…
lw t0 (a0)
li   t1 4
…
```

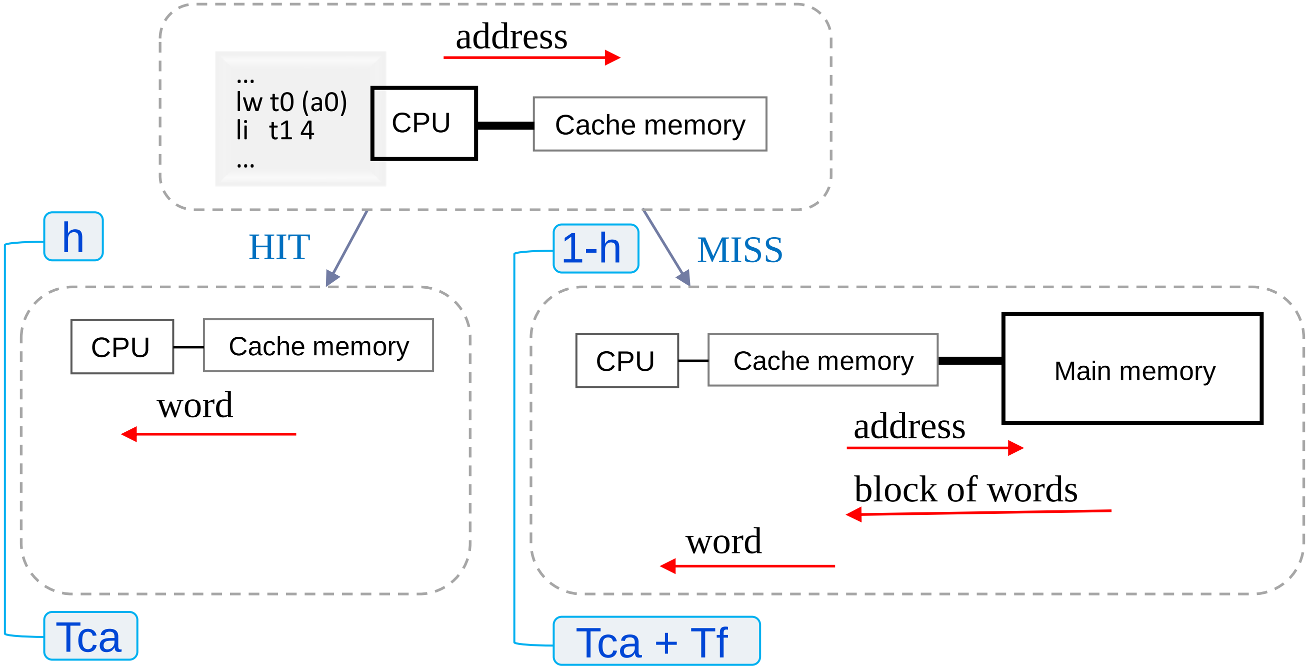
Operation of the cache memory

1. The processor requests the contents of a memory location.
2. The cache checks if the data for this position is already there:
   - **IF it is there (HIT)**

     3.A.1 It is served to the Processor from the cache (quickly): Ta.
   - **IF it is not there (MISS)**

     3.B.1 The cache transfers from Main memory the block associated with position: Tf

     3.B.2 The cache then delivers the requested data to the processor: Ta.

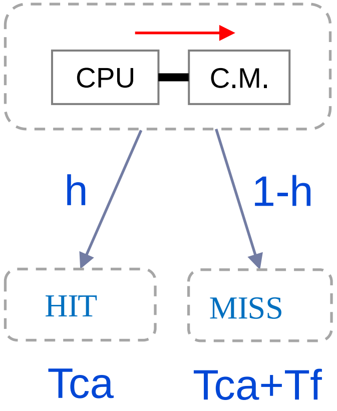


```
...
lw t0 (a0)
li   t1 4
...
```

Average cache access time

- Average access time of a two-level memory system:

$$Tm = h \cdot Ta + (1\text{-}h) \cdot (Ta+Tf)$$
$$= Ta + (1\text{-}h) \cdot Tf$$



  - **Ta**: cache access time
  - **Tf:** time to process a miss
    - It includes time to replace an old block, bring new block from main memory to cache, etc.
  - **h**: hit ratio

Example

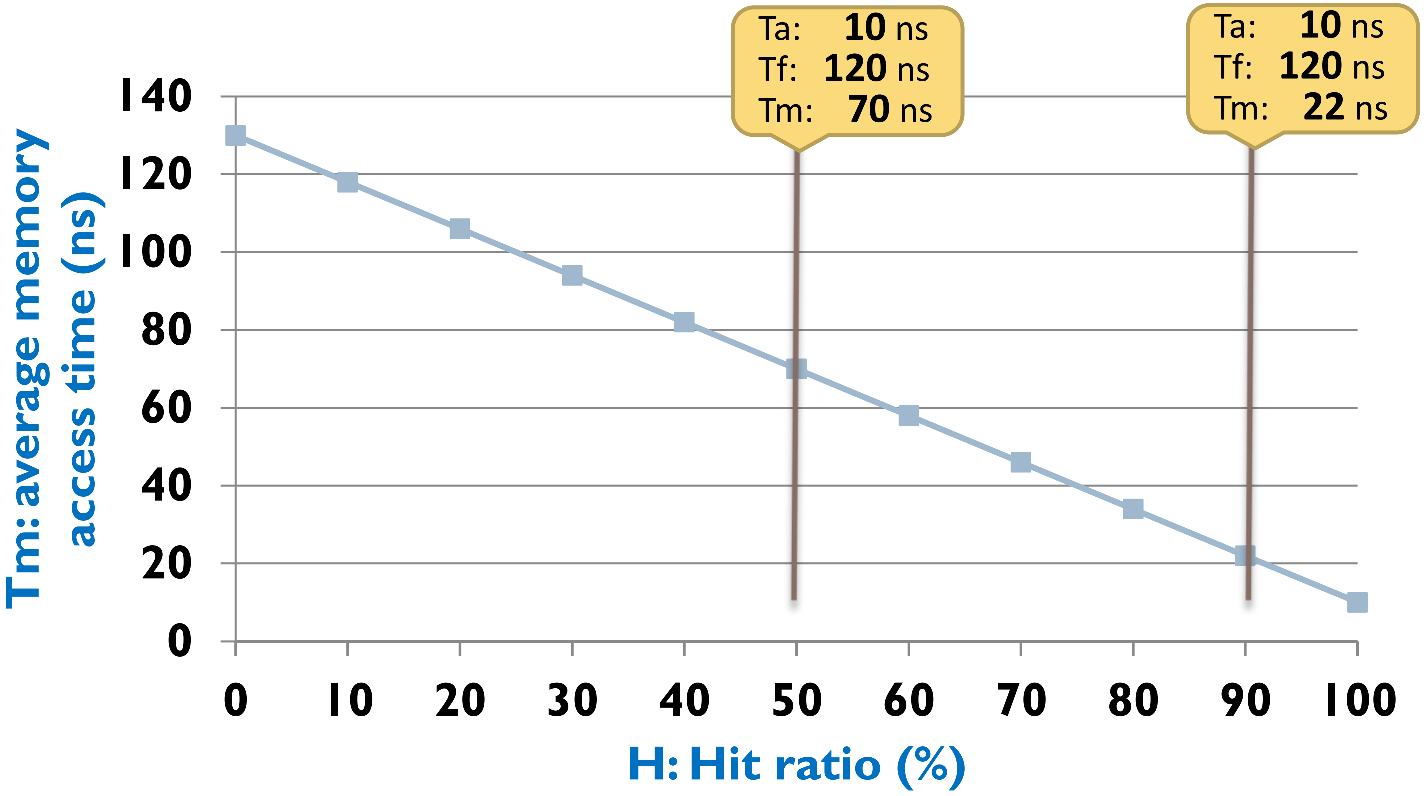
$$
\begin{aligned}
Tm &= h \cdot Ta + (1-h) \cdot (Ta+Tf) \\
&= Ta + (1-h) \cdot Tf
\end{aligned}
$$

1. Ta: Cache access time = **10** ns
2. Tf: Main memory access time = **120** ns
3. h: Hit ratio-> **X = 0.1, 0.2, …, 0.9, 1.0**

   10%, 20%, …, 90%, 100%

Example

$$Tm = h \cdot Ta + (1-h) \cdot (Ta+Tf)$$
$$= Ta + (1-h) \cdot Tf$$



Exercise

- Computer:
  - Cache access time: 4 ns
  - Time to access a block of MM: 120 ns.
- With a hit ratio of 90%, what is the average memory access time?
- What is the hit ratio needed to obtain a memory access time less than 5 ns?

Exercise (solution)

- Computer:
  - Cache access time: 4 ns
  - Time to access a block of MM: 120 ns.
- With a hit ratio of 90%, what is the average memory access time?

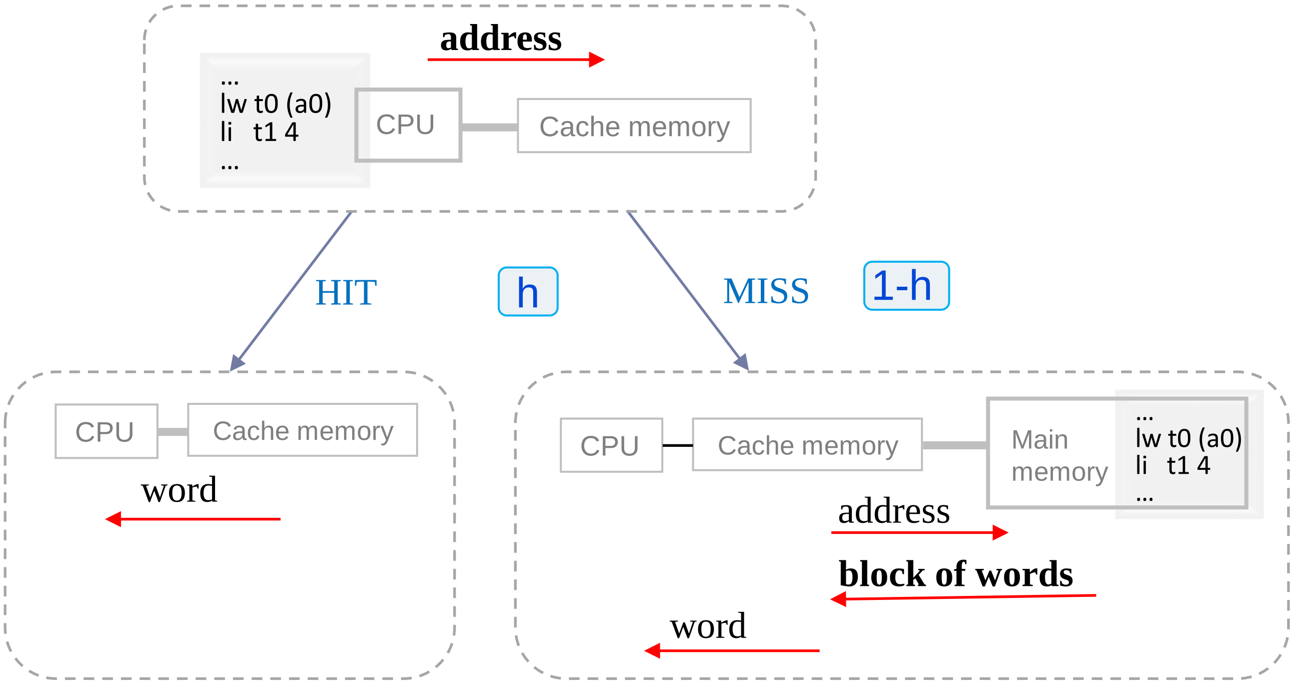
$$T_m = 4 \times 0.9 + (120 + 4) \times 0.1 = 16\ ns$$

- What is the hit ratio needed to obtain a memory access time less than 5 ns?

$$5 = 4 \times h + (120 + 4) \times (1 - h)$$
$$\Rightarrow h > 0.9916$$

Hit ratio of a code fragment

- The hit ratio h depends on:
  - Layout in main memory and cache (affected blocks).
  - Access trace (list of addresses) generated during execution.
  - Cache behavior (lookup time, replacement, etc.)



Example of how it works

```
int i;
int s = 0;
for (i=0; i < 1000; i++)
     s = s + i;
```

```
        li   t0, 0     # s
        li   t1, 0     # i
        li   t2, 1000
bucle:  bge  t1, t2, fin
        add  t0, t0, t1
        addi t1, t1, 1
        j    bucle
fin:    ...
```

- Example:
  - Cache access: 2 ns
  - Main memory Access: 120 ns
  - Cache block: 4 words
  - Transfer a block between main memory and cache: 200 ns

Example of how it works

```
int i;
int s = 0;
for (i=0; i < 1000; i++)
     s = s + i;
```

```
        li   t0, 0     # s
        li   t1, 0     # i
        li   t2, 1000
bucle:  bge  t1, t2, fin
        add  t0, t0, t1
        addi t1, t1, 1
        j    bucle
fin:    …
```

- Without cache memory:
  - Number of memory access = 3 + 4 × 1000 + 1 = 4004 access
  - Total access time = 4004 × 120 = 480480 ns = 0,480 ms

Example of how it works

```
int i;
int s = 0;
for (i=0; i < 1000; i++)
     s = s + i;
```

```
        li    t0, 0     # s
        li    t1, 0     # i
        li    t2, 1000
bucle:  bge   t1, t2, fin
        add   t0, t0, t1
        addi  t1, t1, 1
        j     bucle
fin:    …
```

- With cache memory (blocks of 4 words):
  - Number of accesses = 4004 access
  - Number of blocks = ?
  - Number of misses = ?
  - Access time = ?

Example of how it works

```
int i;
int s = 0;
for (i=0; i < 1000; i++)
     s = s + i;
```

```
        li    t0, 0     # s
        li    t1, 0     # i
        li    t2, 1000
bucle:  bge   t1, t2, fin
        add   t0, t0, t1
        addi  t1, t1, 1
        j     bucle
fin:    …
```

- With cache memory (blocks of 4 words):
  - Number of accesses = 4004 access
  - Number of blocks = 2
  - Number of misses = ?
  - Access time = ?

(1/2) block study:
analysis of affected blocks of data and code

Example of how it works

```
int i;
int s = 0;
for (i=0; i < 1000; i++)
     s = s + i;
```

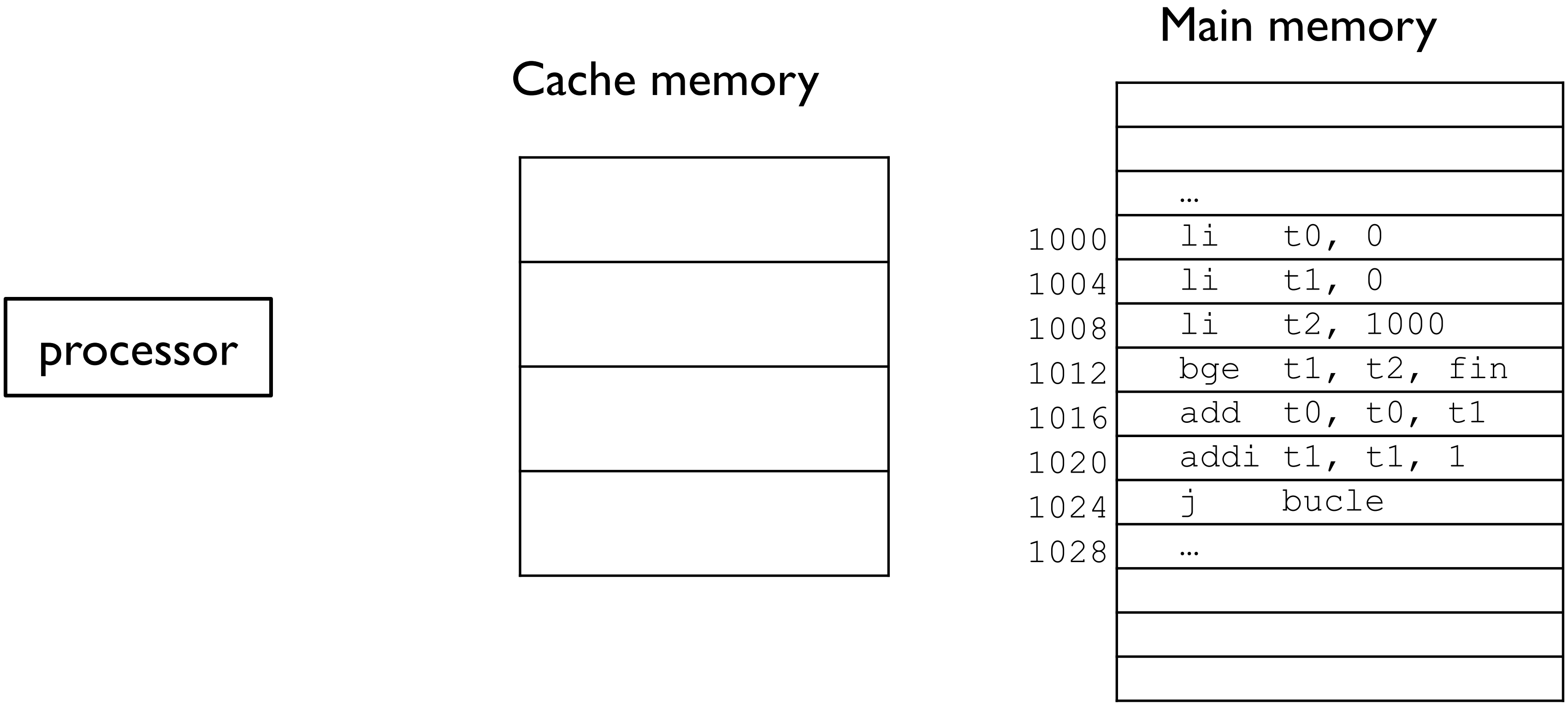
```
        li    t0, 0     # s
        li    t1, 0     # i
        li    t2, 1000
bucle:  bge   t1, t2, fin
        add   t0, t0, t1
        addi  t1, t1, 1
        j     bucle
fin:    …
```

- With cache memory (blocks of 4 words):
  - Number of accesses = 4004 access
  - Number of blocks = 2
  - Number of misses = ?
  - Access time = ?

(2/2) study of references generated by execution: access to code (fetch) and data

Example of how it works

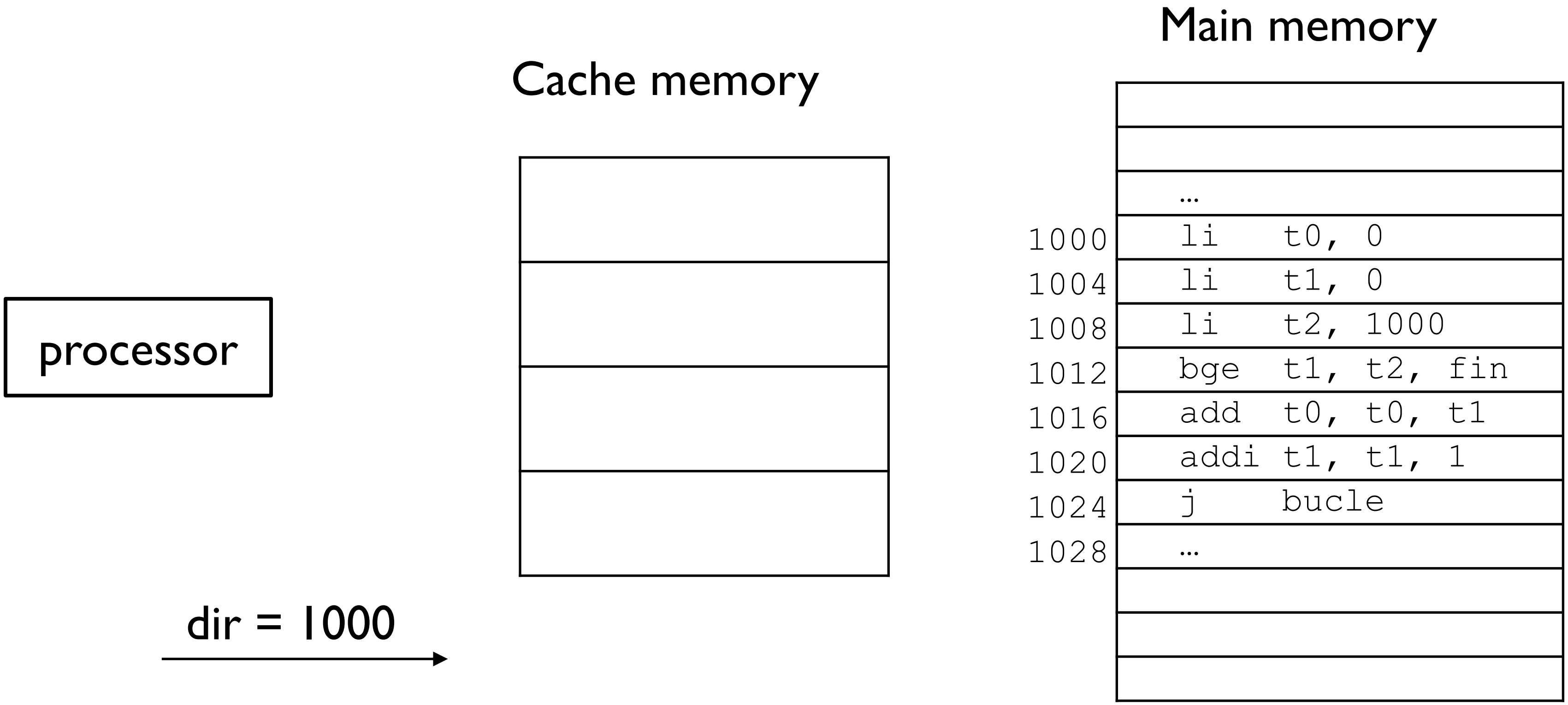
(2/2) study of generated references



```
      …
1000  li   t0, 0
1004  li   t1, 0
1008  li   t2, 1000
1012  bge  t1, t2, fin
1016  add  t0, t0, t1
1020  addi t1, t1, 1
1024  j    bucle
1028  …
```

Example of how it works

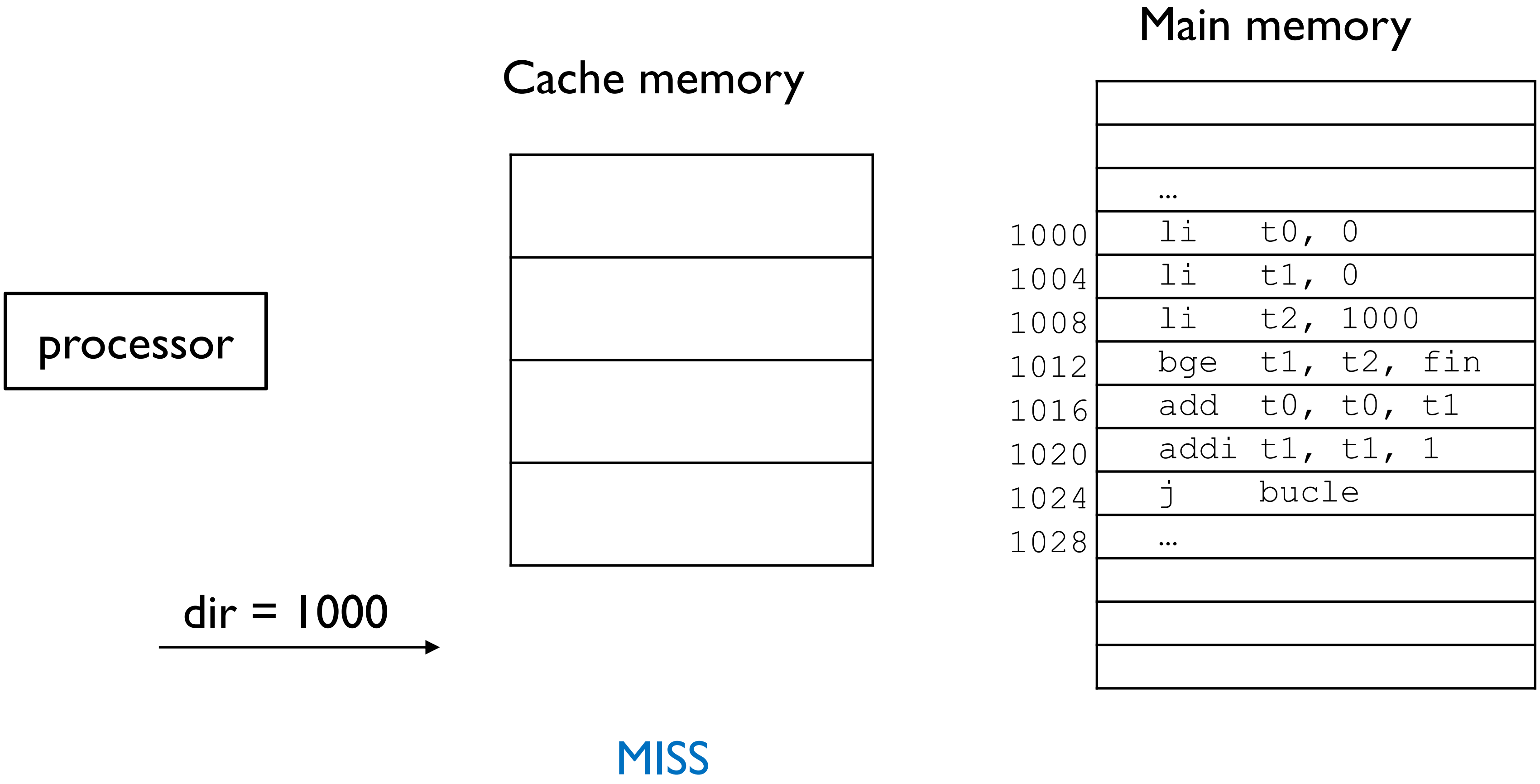
(2/2) study of generated references



```
     …
1000 li   t0, 0
1004 li   t1, 0
1008 li   t2, 1000
1012 bge  t1, t2, fin
1016 add  t0, t0, t1
1020 addi t1, t1, 1
1024 j    bucle
1028 …
```

Example of how it works

(2/2) study of generated references

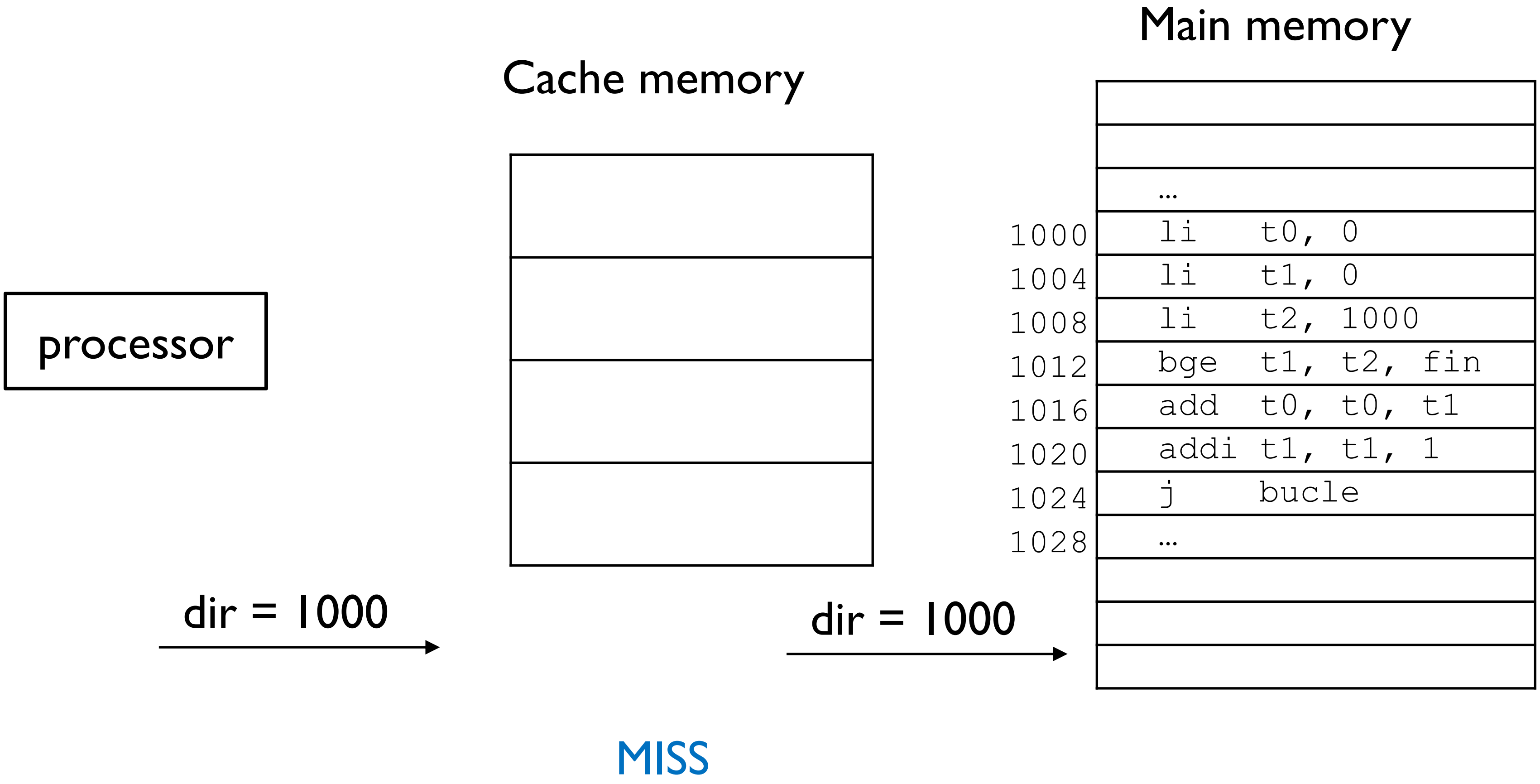


```
     …
1000 li   t0, 0
1004 li   t1, 0
1008 li   t2, 1000
1012 bge  t1, t2, fin
1016 add  t0, t0, t1
1020 addi t1, t1, 1
1024 j    bucle
1028 …
```

MISS

Example of how it works

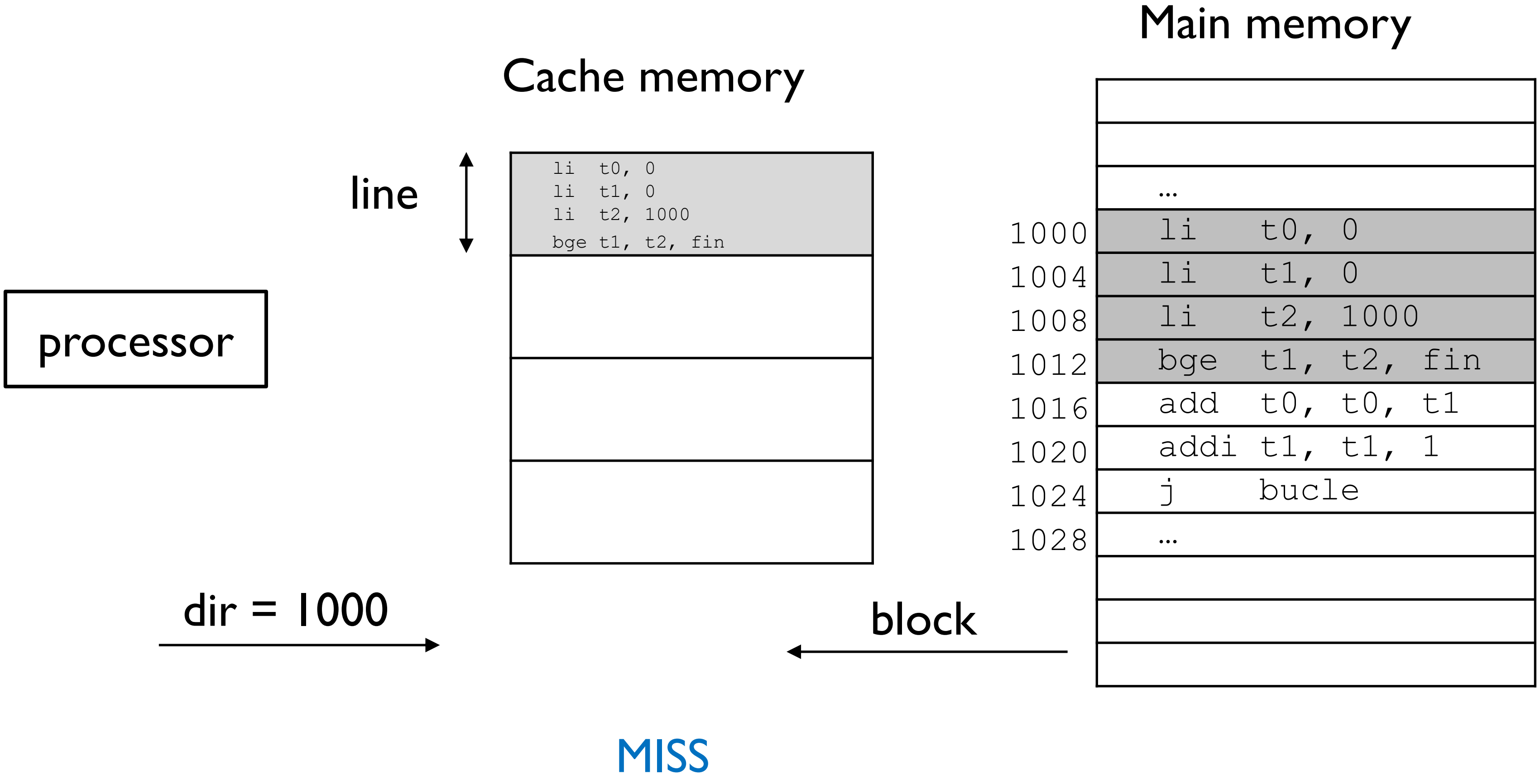
(2/2) study of generated references



```
     …
1000 li   t0, 0
1004 li   t1, 0
1008 li   t2, 1000
1012 bge  t1, t2, fin
1016 add  t0, t0, t1
1020 addi t1, t1, 1
1024 j    bucle
1028 …
```

Example of how it works

(2/2) study of generated references



Cache memory line:

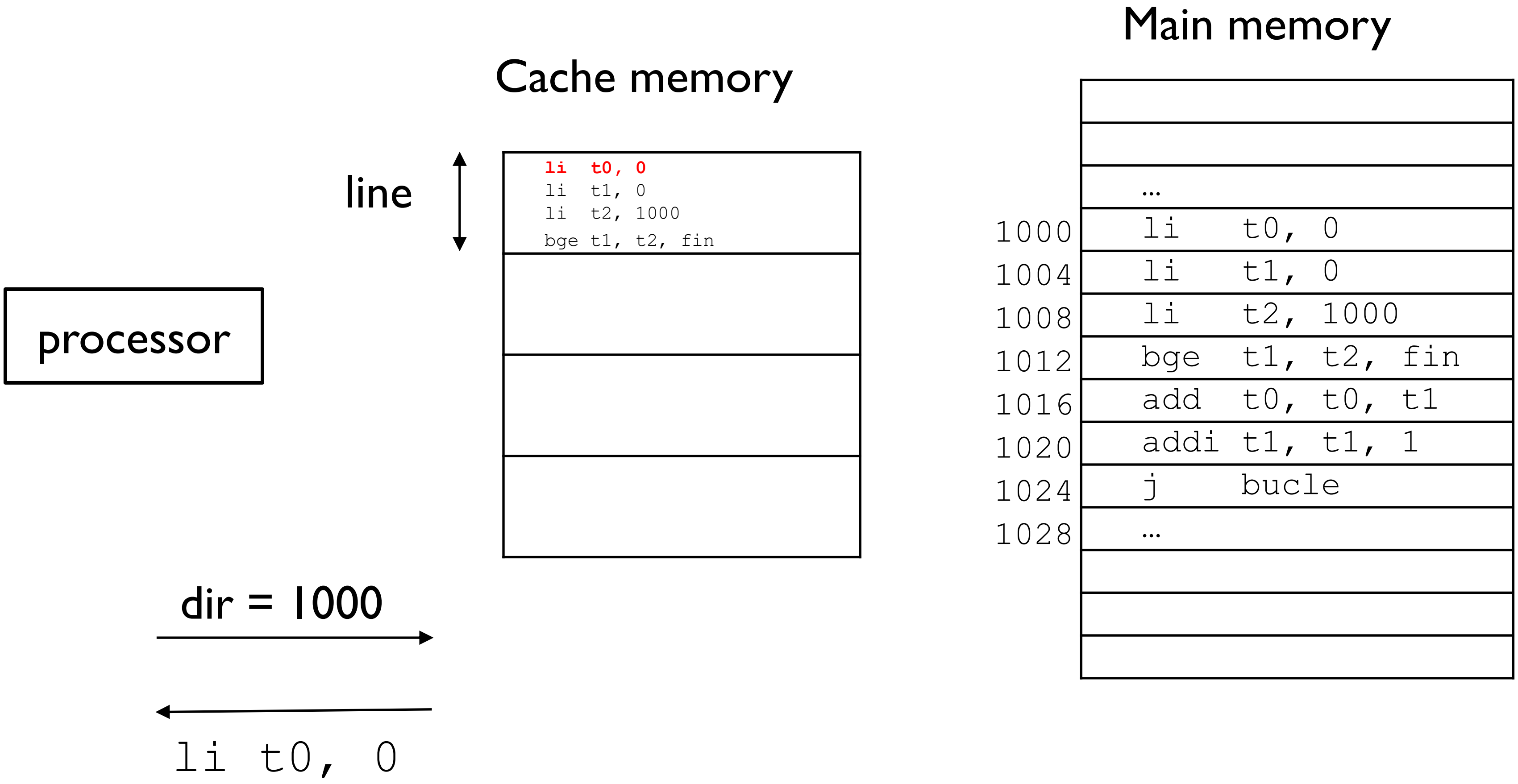
```
li  t0, 0
li  t1, 0
li  t2, 1000
bge t1, t2, fin
```

Main memory:

```
     …
1000 li   t0, 0
1004 li   t1, 0
1008 li   t2, 1000
1012 bge  t1, t2, fin
1016 add  t0, t0, t1
1020 addi t1, t1, 1
1024 j    bucle
1028 …
```

Example of how it works

(2/2) study of generated references



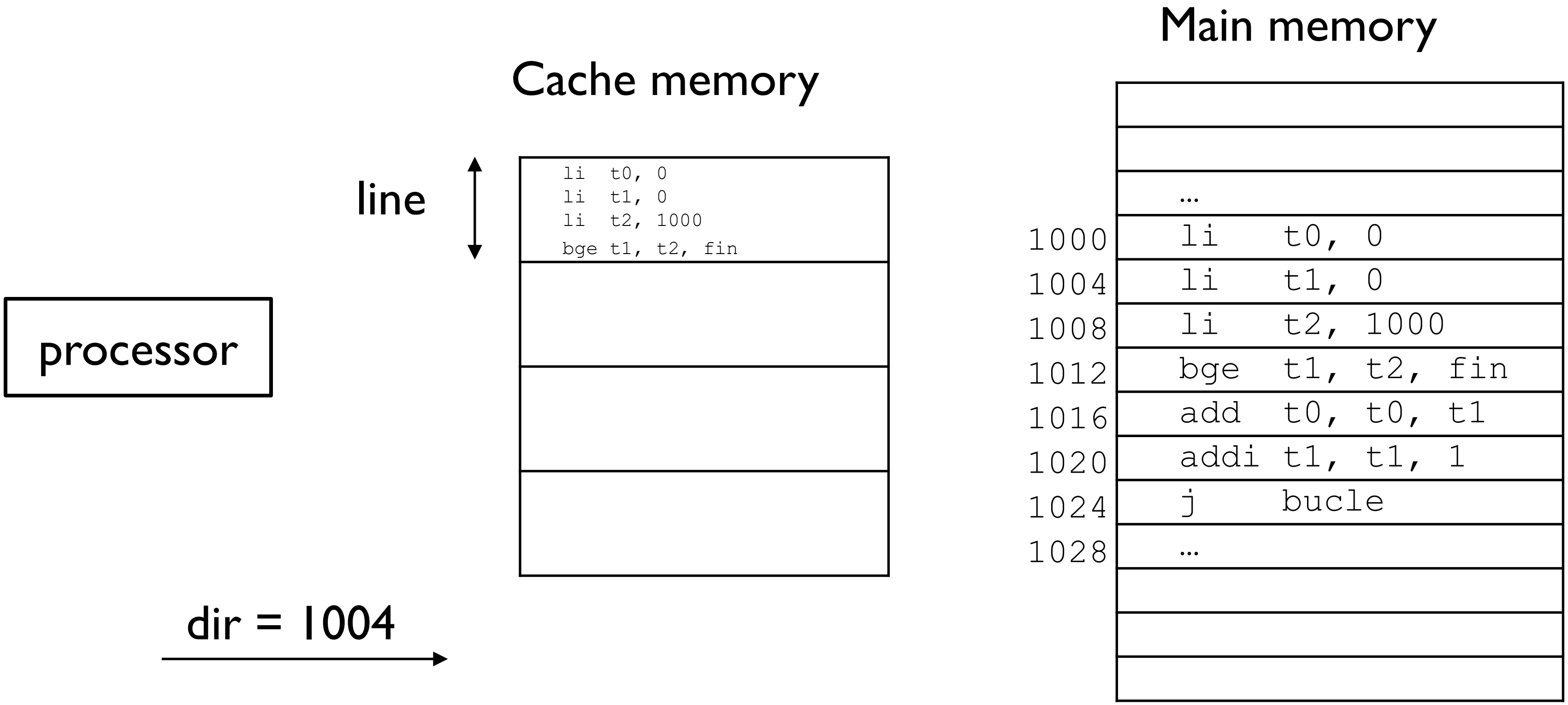
```
li  t0, 0
li  t1, 0
li  t2, 1000
bge t1, t2, fin
```

```
li t0, 0
```

	…
1000	li t0, 0
1004	li t1, 0
1008	li t2, 1000
1012	bge t1, t2, fin
1016	add t0, t0, t1
1020	addi t1, t1, 1
1024	j bucle
1028	…

Example of how it works

(2/2) study of generated references

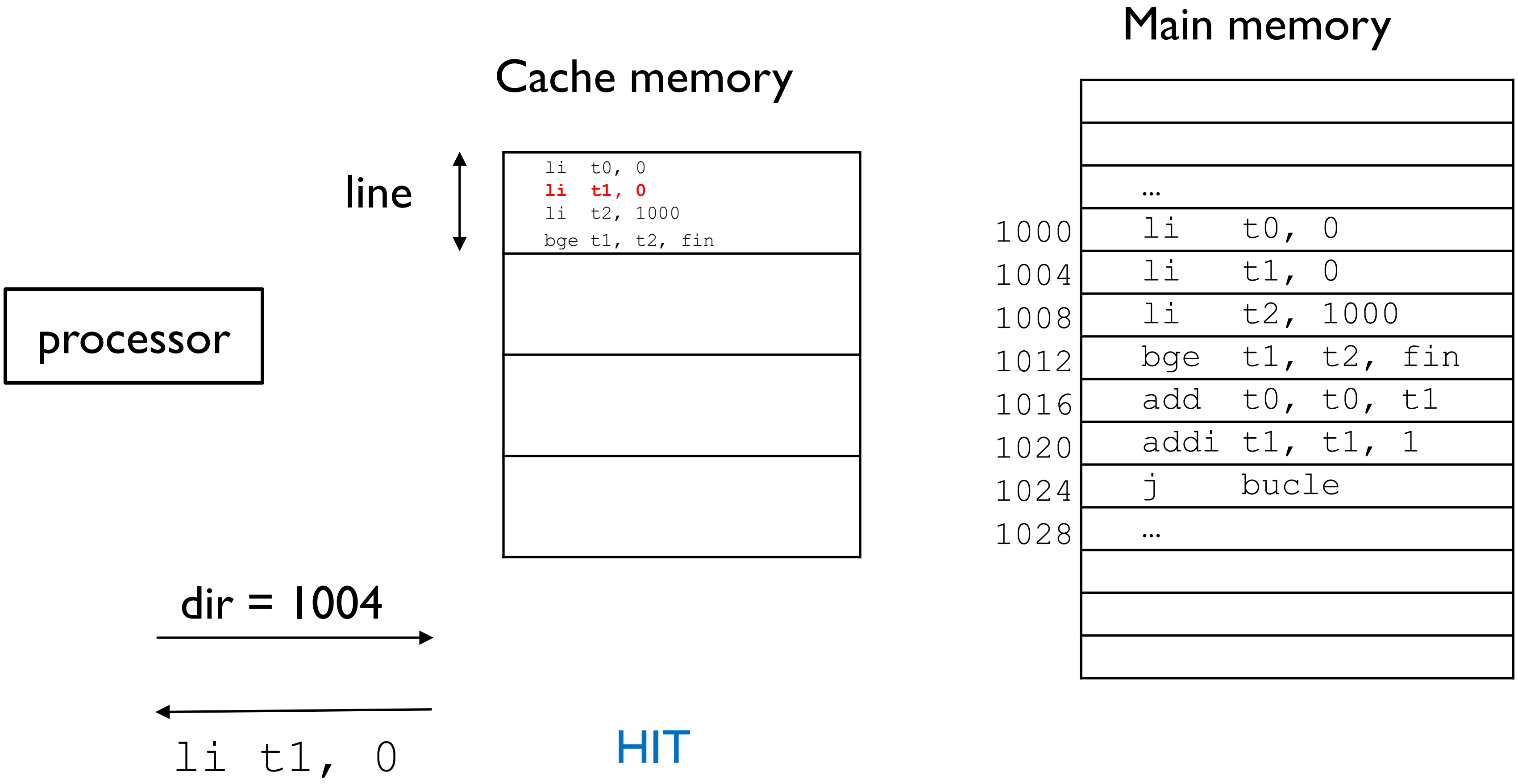


```
li   t0, 0
li   t1, 0
li   t2, 1000
bge t1, t2, fin
```

```
     …
1000 li   t0, 0
1004 li   t1, 0
1008 li   t2, 1000
1012 bge  t1, t2, fin
1016 add  t0, t0, t1
1020 addi t1, t1, 1
1024 j    bucle
1028 …
```

Example of how it works

(2/2) study of generated references



Cache memory line:

```
li  t0, 0
li  t1, 0
li  t2, 1000
bge t1, t2, fin
```

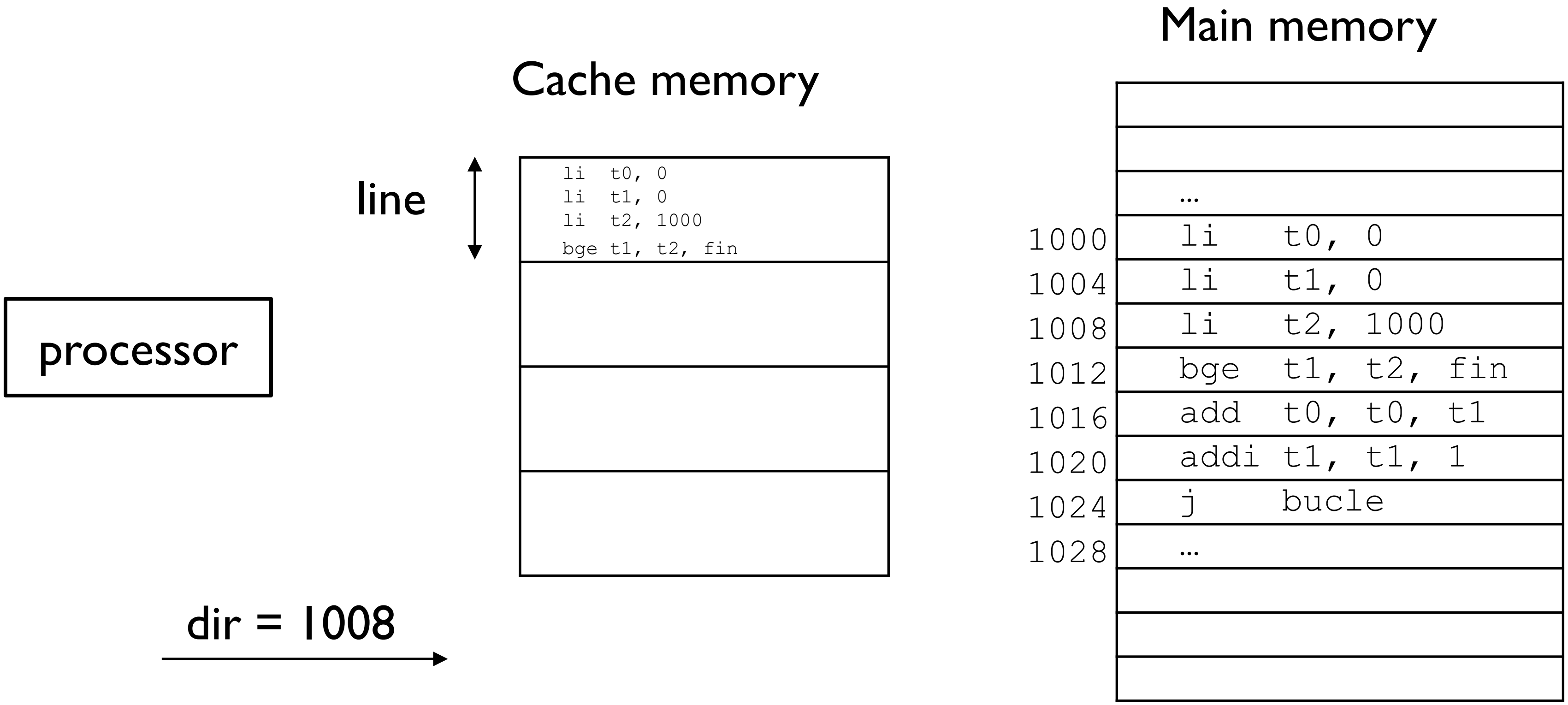
Main memory:

```
     …
1000 li   t0, 0
1004 li   t1, 0
1008 li   t2, 1000
1012 bge  t1, t2, fin
1016 add  t0, t0, t1
1020 addi t1, t1, 1
1024 j    bucle
1028 …
```

```
li t1, 0
```

Example of how it works

(2/2) study of generated references

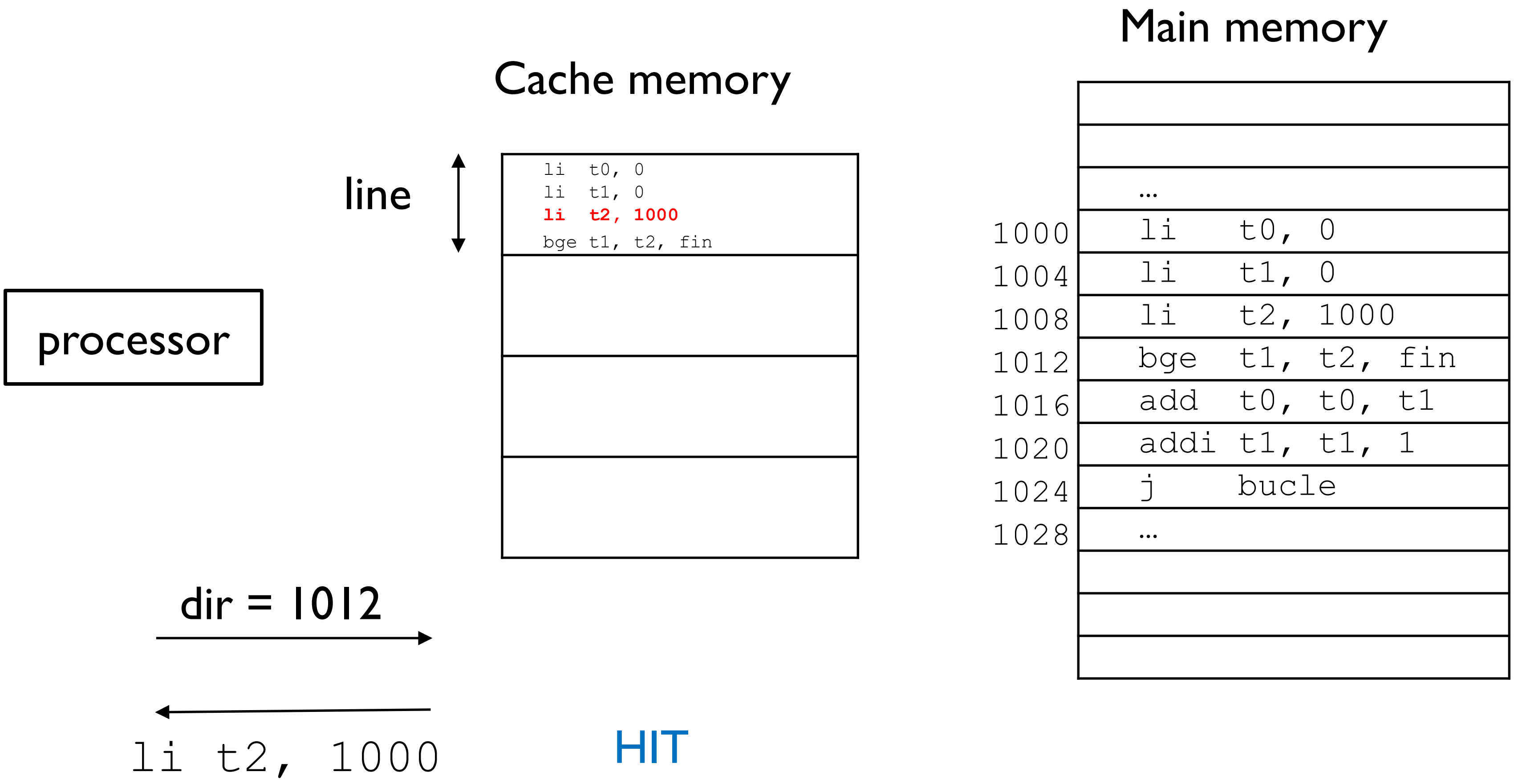


```
li   t0, 0
li   t1, 0
li   t2, 1000
bge t1, t2, fin
```

	…
1000	li t0, 0
1004	li t1, 0
1008	li t2, 1000
1012	bge t1, t2, fin
1016	add t0, t0, t1
1020	addi t1, t1, 1
1024	j bucle
1028	…

Example of how it works

(2/2) study of generated references



Cache memory

line

```
li  t0, 0
li  t1, 0
li  t2, 1000
bge t1, t2, fin
```

processor

dir = 1012

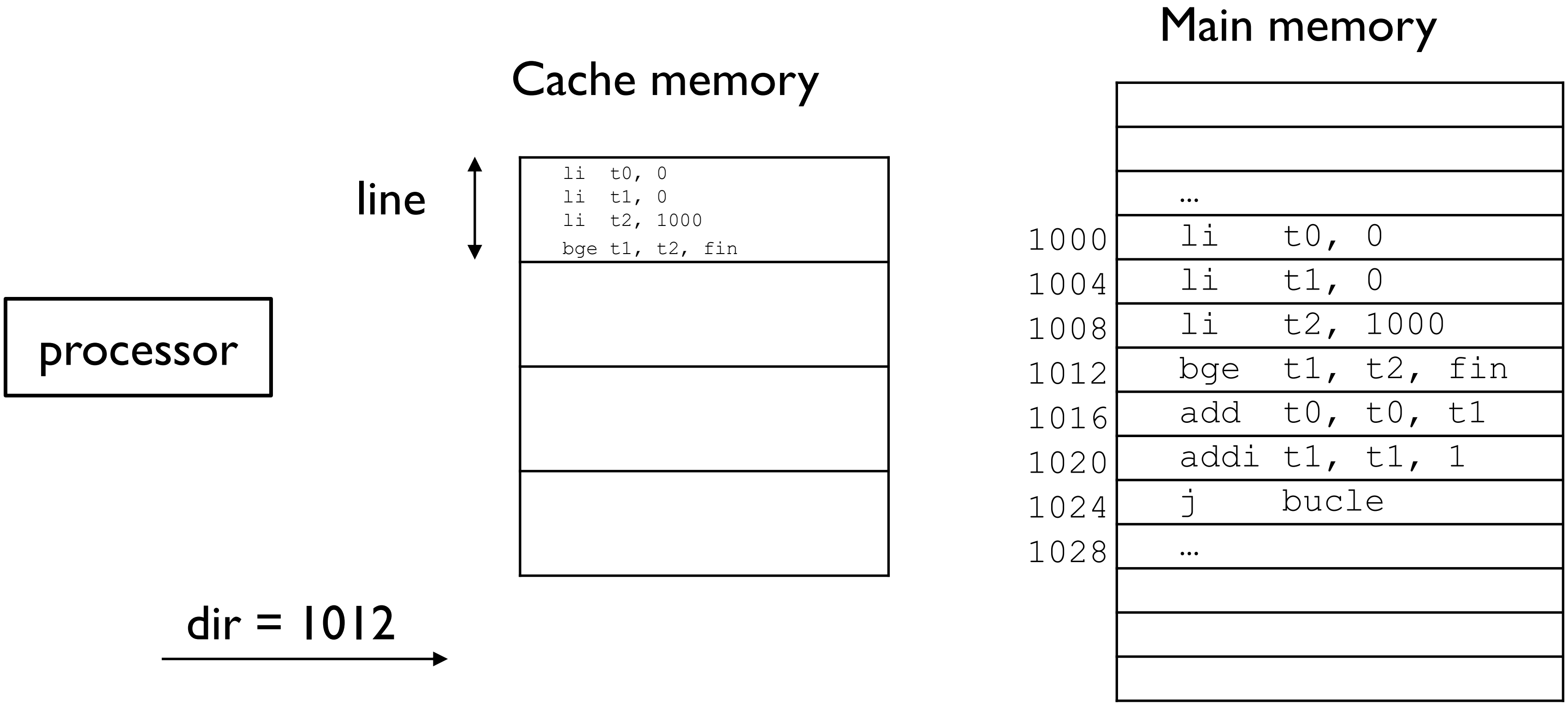
li t2, 1000

HIT

Main memory

	…
1000	li t0, 0
1004	li t1, 0
1008	li t2, 1000
1012	bge t1, t2, fin
1016	add t0, t0, t1
1020	addi t1, t1, 1
1024	j bucle
1028	…

Example of how it works
(2/2) study of generated references



Cache memory line:

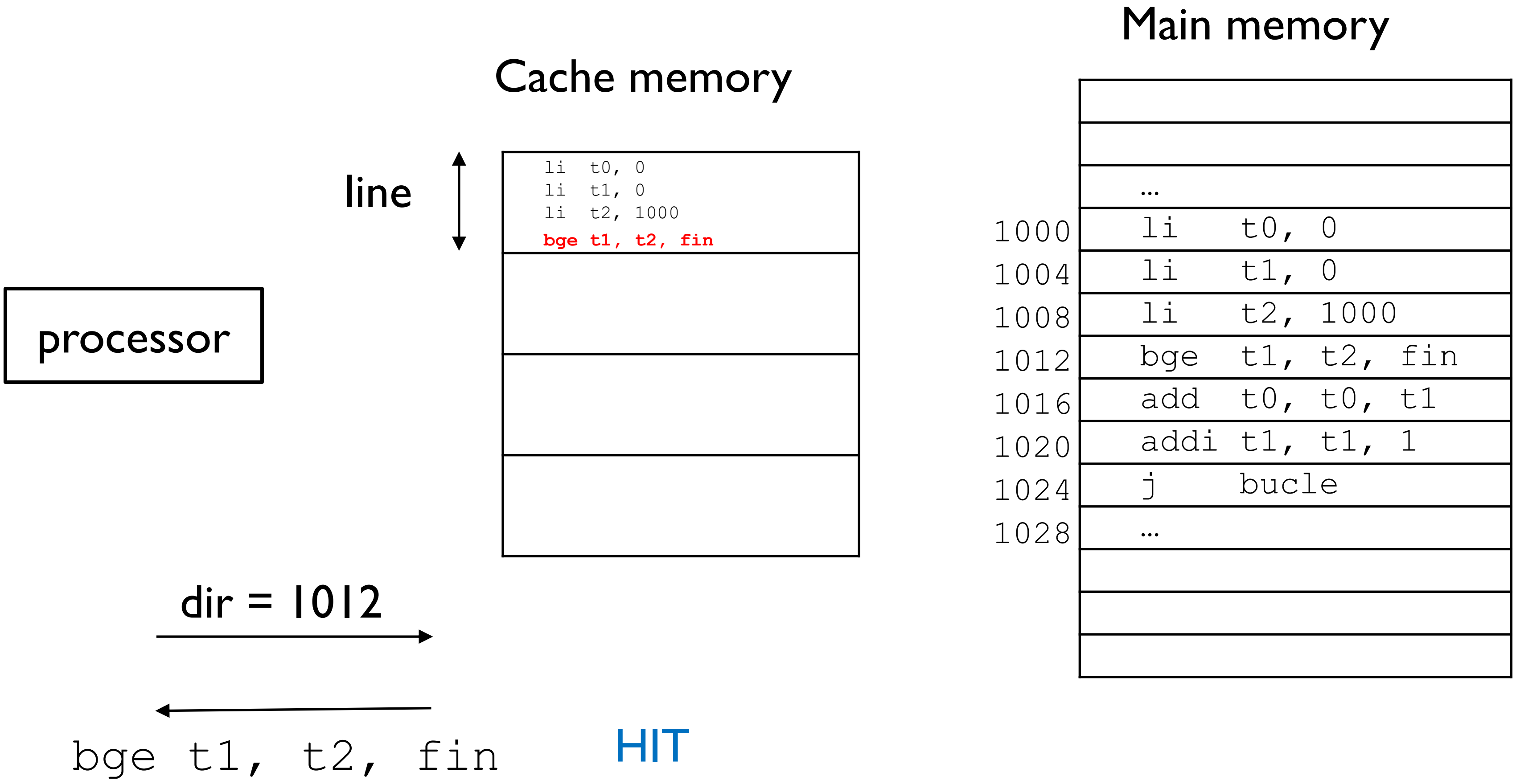
```
li  t0, 0
li  t1, 0
li  t2, 1000
bge t1, t2, fin
```

Main memory:

```
     …
1000 li   t0, 0
1004 li   t1, 0
1008 li   t2, 1000
1012 bge  t1, t2, fin
1016 add  t0, t0, t1
1020 addi t1, t1, 1
1024 j    bucle
1028 …
```

Example of how it works

(2/2) study of generated references



Cache memory

line

```
li  t0, 0
li  t1, 0
li  t2, 1000
bge t1, t2, fin
```

processor

dir = 1012

bge t1, t2, fin

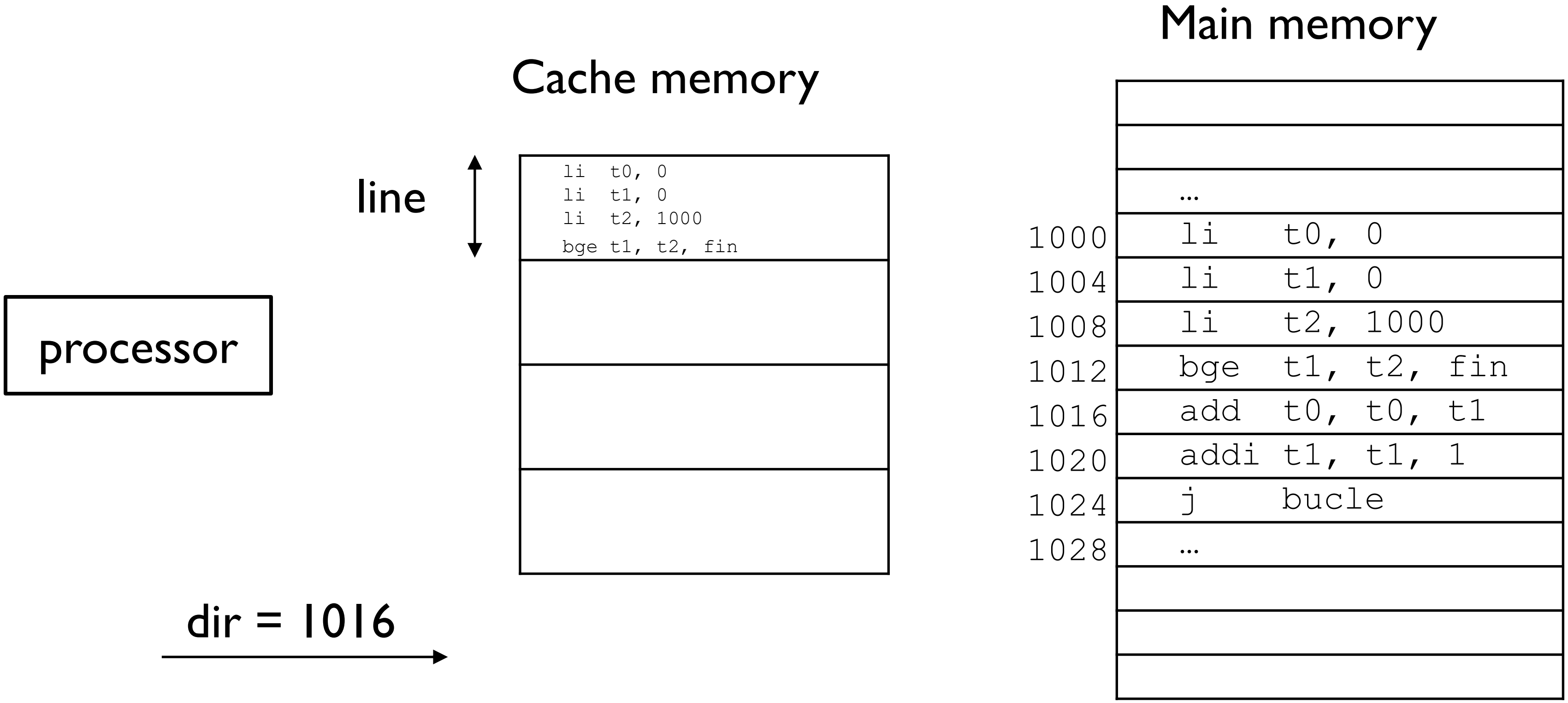
HIT

Main memory

	…
1000	li t0, 0
1004	li t1, 0
1008	li t2, 1000
1012	bge t1, t2, fin
1016	add t0, t0, t1
1020	addi t1, t1, 1
1024	j bucle
1028	…

Example of how it works

(2/2) study of generated references

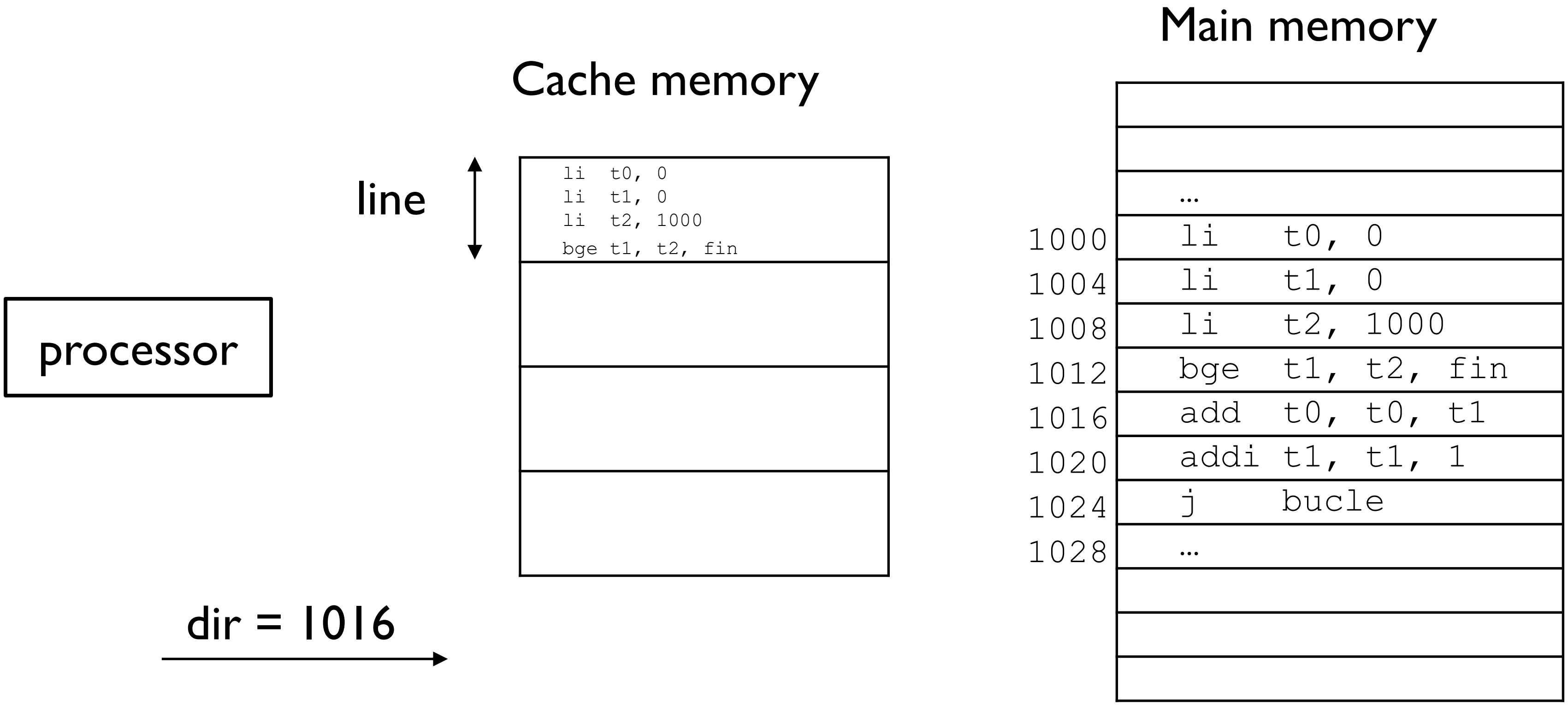


```
li  t0, 0
li  t1, 0
li  t2, 1000
bge t1, t2, fin
```

```
     …
1000 li   t0, 0
1004 li   t1, 0
1008 li   t2, 1000
1012 bge  t1, t2, fin
1016 add  t0, t0, t1
1020 addi t1, t1, 1
1024 j    bucle
1028 …
```

Example of how it works

(2/2) study of generated references



Main memory

	…
1000	li t0, 0
1004	li t1, 0
1008	li t2, 1000
1012	bge t1, t2, fin
1016	add t0, t0, t1
1020	addi t1, t1, 1
1024	j bucle
1028	…

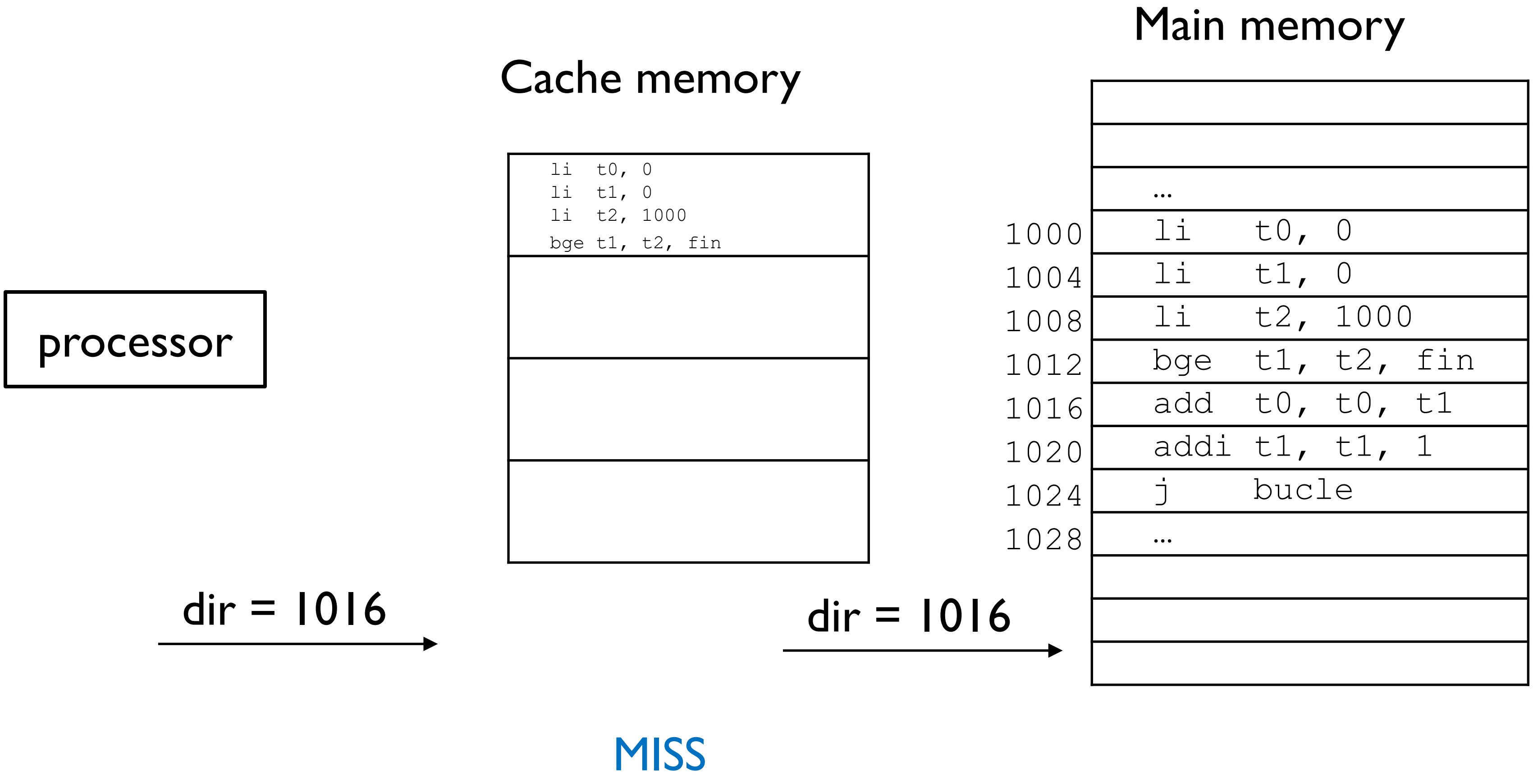
Cache memory

```
li  t0, 0
li  t1, 0
li  t2, 1000
bge t1, t2, fin
```

MISS

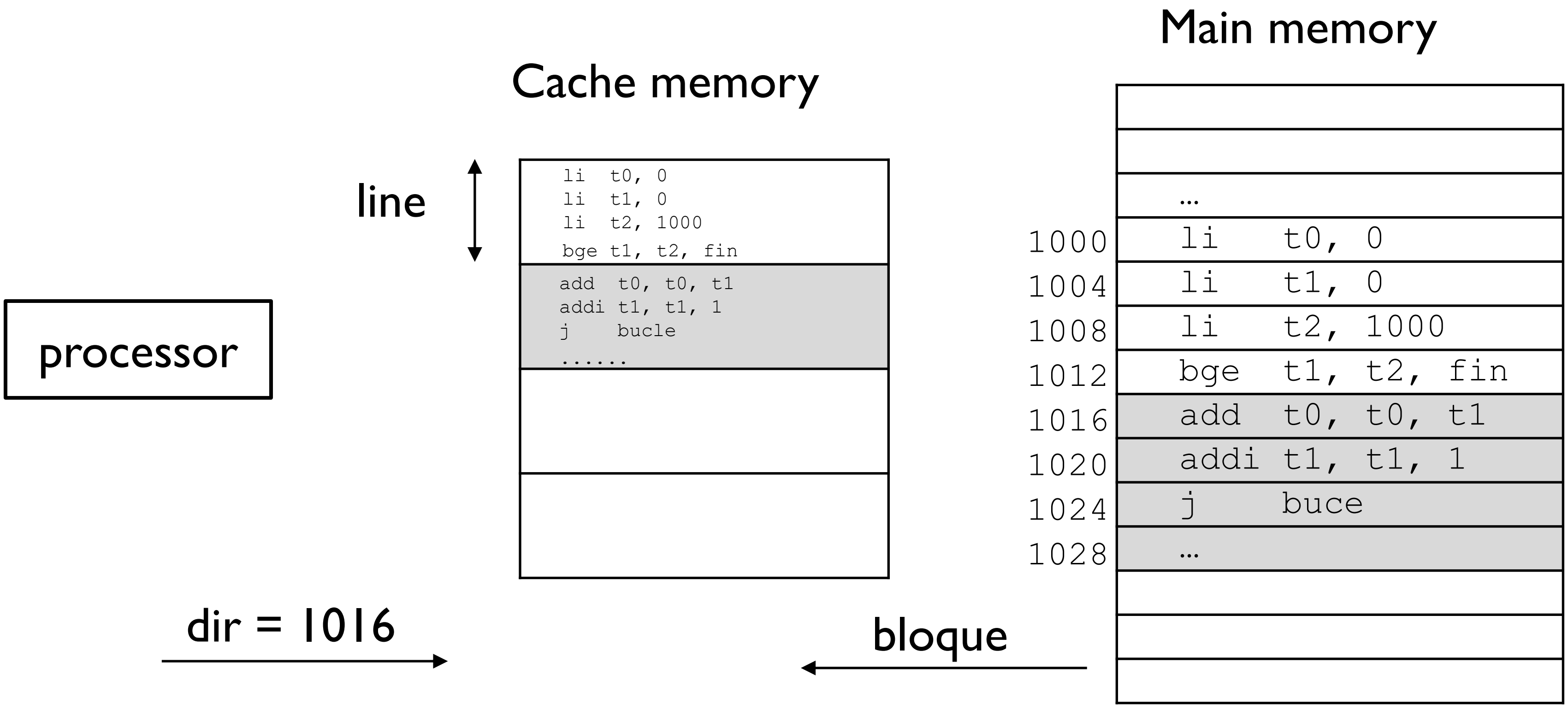
Example of how it works

(2/2) study of generated references



Example of how it works

(2/2) study of generated references



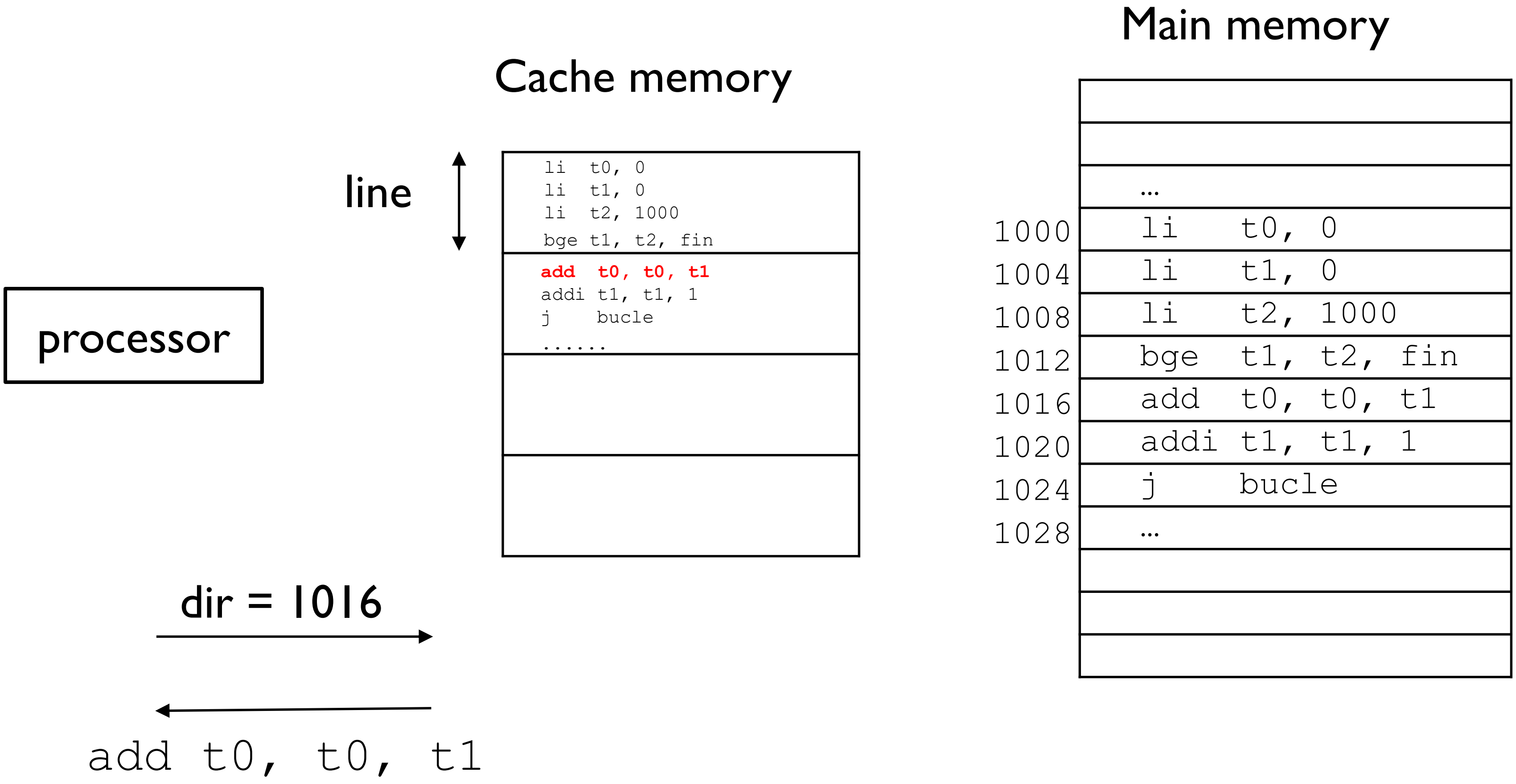
```
li   t0, 0
li   t1, 0
li   t2, 1000
bge t1, t2, fin
add  t0, t0, t1
addi t1, t1, 1
j    bucle
......
```

	…
1000	li t0, 0
1004	li t1, 0
1008	li t2, 1000
1012	bge t1, t2, fin
1016	add t0, t0, t1
1020	addi t1, t1, 1
1024	j buce
1028	…

MISS

Example of how it works

(2/2) study of generated references



Cache memory

line

```
li   t0, 0
li   t1, 0
li   t2, 1000
bge t1, t2, fin
```

```
add  t0, t0, t1
addi t1, t1, 1
j    bucle
......
```

processor

dir = 1016

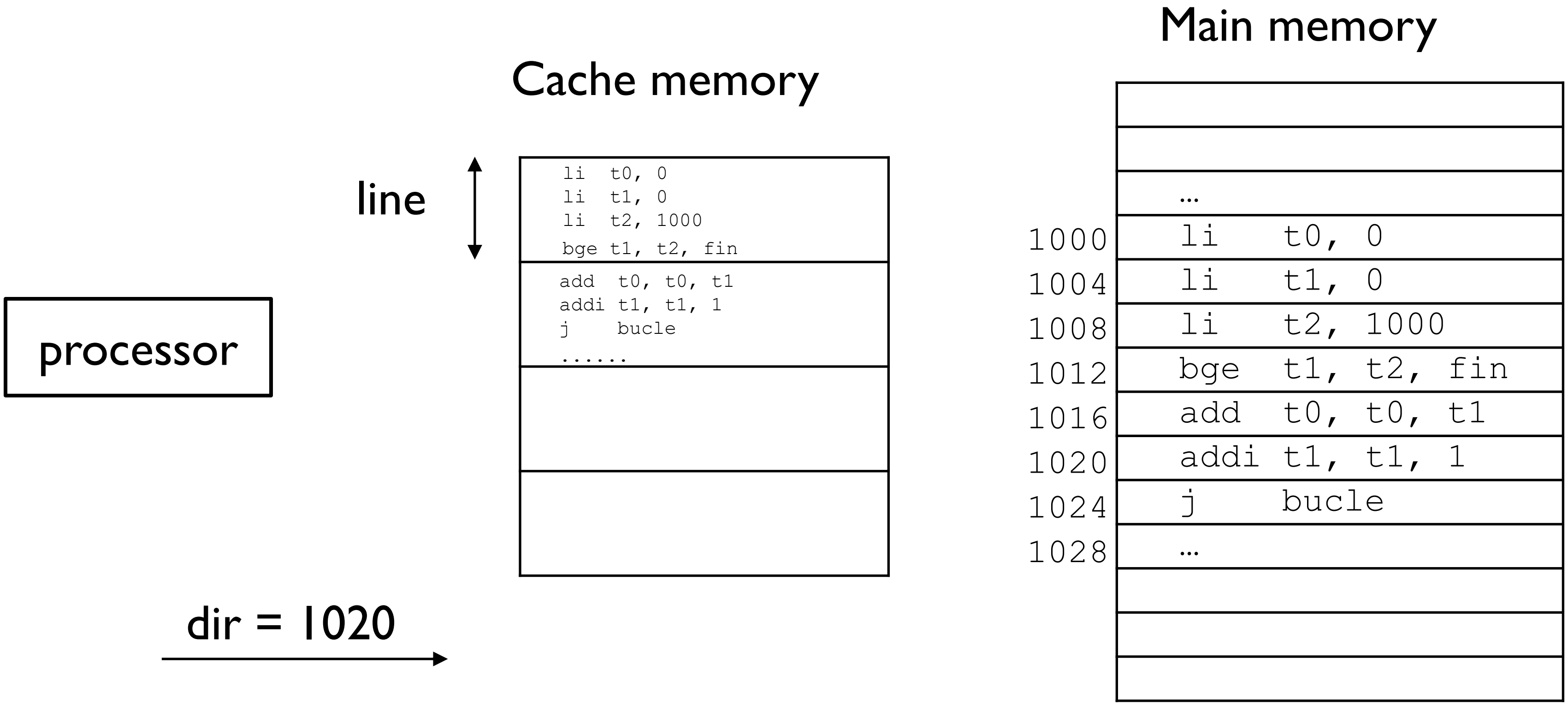
```
add t0, t0, t1
```

Main memory

	…
1000	`li   t0, 0`
1004	`li   t1, 0`
1008	`li   t2, 1000`
1012	`bge  t1, t2, fin`
1016	`add  t0, t0, t1`
1020	`addi t1, t1, 1`
1024	`j    bucle`
1028	…

Example of how it works

(2/2) study of generated references



Cache memory:

```
li   t0, 0
li   t1, 0
li   t2, 1000
bge t1, t2, fin
```

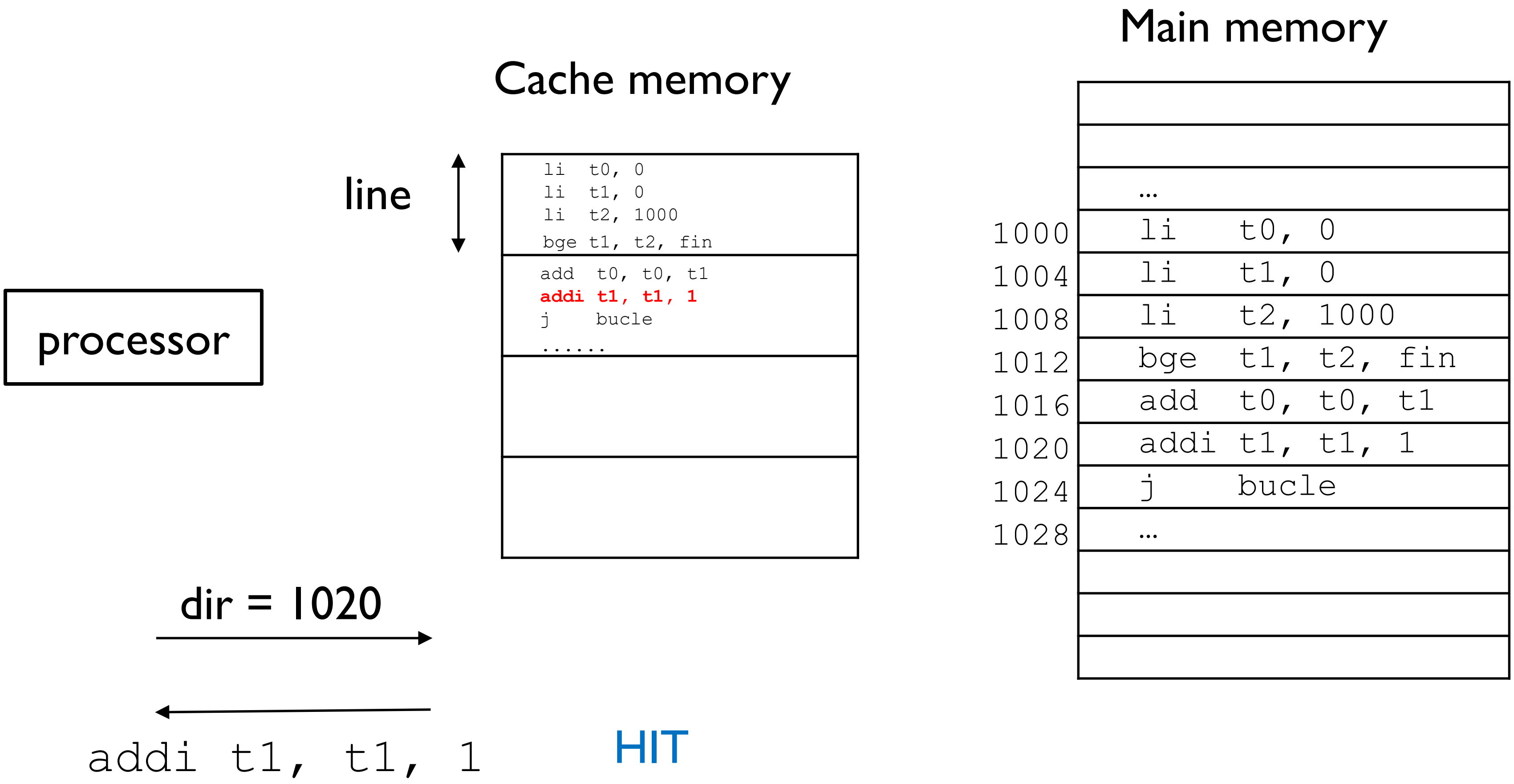
```
add  t0, t0, t1
addi t1, t1, 1
j    bucle
......
```

Main memory:

	…
1000	li t0, 0
1004	li t1, 0
1008	li t2, 1000
1012	bge t1, t2, fin
1016	add t0, t0, t1
1020	addi t1, t1, 1
1024	j bucle
1028	…

Example of how it works

(2/2) study of generated references



Cache memory

line

processor

```
li   t0, 0
li   t1, 0
li   t2, 1000
bge t1, t2, fin
```

```
add  t0, t0, t1
addi t1, t1, 1
j    bucle
......
```

Main memory

	…
1000	li t0, 0
1004	li t1, 0
1008	li t2, 1000
1012	bge t1, t2, fin
1016	add t0, t0, t1
1020	addi t1, t1, 1
1024	j bucle
1028	…

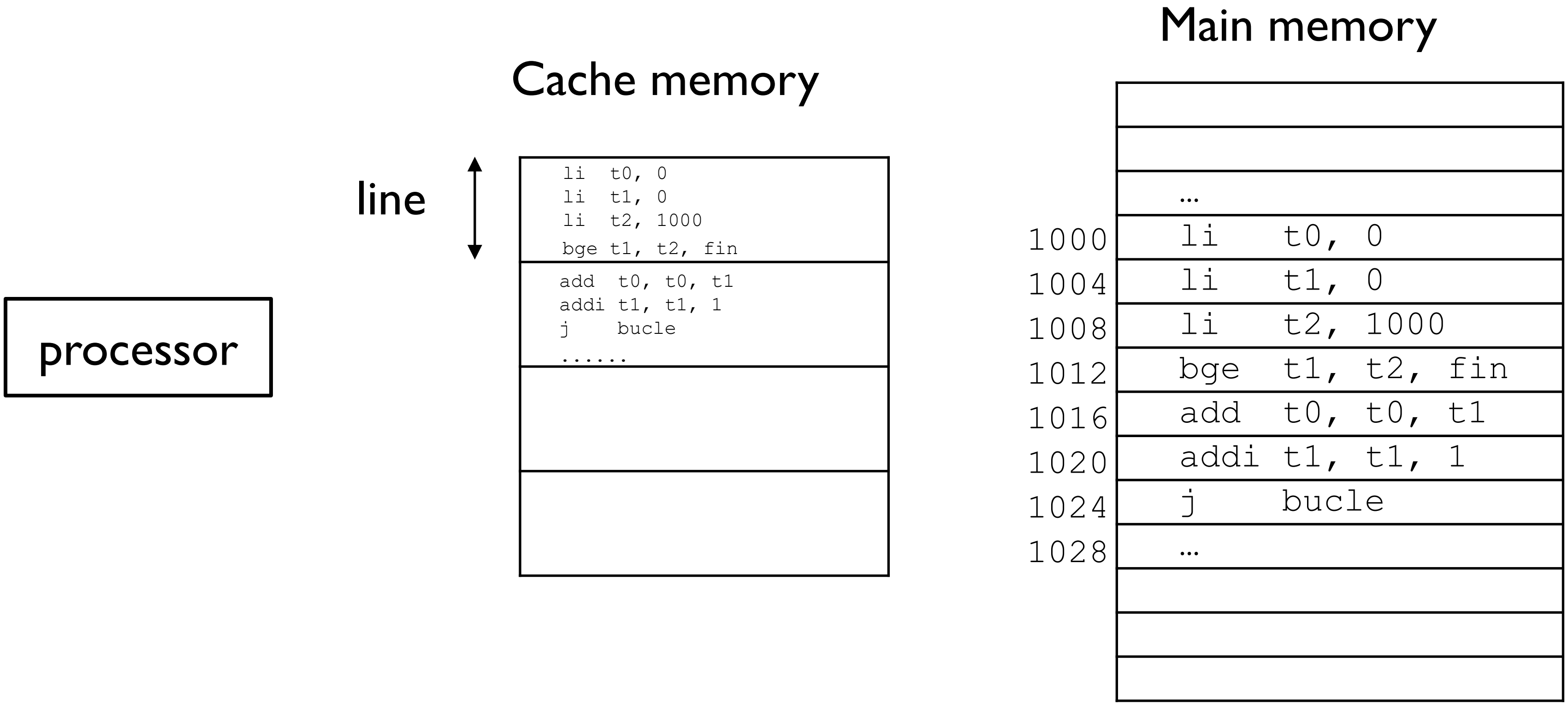
dir = 1020

`addi t1, t1, 1`

HIT

Example of how it works

(2/2) study of generated references



Cache memory:

```
li   t0, 0
li   t1, 0
li   t2, 1000
bge t1, t2, fin
```

```
add  t0, t0, t1
addi t1, t1, 1
j    bucle
......
```

Main memory:

	…
1000	`li   t0, 0`
1004	`li   t1, 0`
1008	`li   t2, 1000`
1012	`bge  t1, t2, fin`
1016	`add  t0, t0, t1`
1020	`addi t1, t1, 1`
1024	`j    bucle`
1028	…

All other accesses (in this code) are HITS

Example of how it works

```
int i;
int s = 0;
for (i=0; i < 1000; i++)
     s = s + i;
```

```
        li    t0, 0      # s
        li    t1, 0      # i
        li    t2, 1000
bucle:  bge   t1, t2, fin
        add   t0, t0, t1
        addi  t1, t1, 1
        j     bucle
fin:    …
```

- With cache memory (blocks of 4 words):
  - Number of accesses = 4004 access
  - Number of blocks = 2
  - Number of misses = 2
  - Access time = ?

(2/2) study of generated references by execution: access to code (fetch) and data

Example of how it works

```
int i;
int s = 0;
for (i=0; i < 1000; i++)
     s = s + i;
```

```
        li    t0, 0      # s
        li    t1, 0      # i
        li    t2, 1000
bucle:  bge   t1, t2, fin
        add   t0, t0, t1
        addi  t1, t1, 1
        j     bucle
fin:    …
```

- With cache memory (blocks of 4 words):
  - Number of accesses = 4004 access
  - Number of blocks = 2
  - Number of misses = 2
  - **Access time = 8408 ns**
    - Time to transfer 2 blocks = 200 × 2 = 400 ns
    - Cache access time = 4004 × 2 = 8008 ns

- **Cache M. access time: 2 ns**
- **Main M. access time: 120 ns**
- **Cache block (line): 4 words**
- **Block transfer between m. memory and cache: 200 ns**

Example of how it works

```
int i;
int s = 0;
for (i=0; i < 1000; i++)
     s = s + i;
```

```
        li    t0, 0     # s
        li    t1, 0     # i
        li    t2, 1000
bucle:  bge   t1, t2, fin
        add   t0, t0, t1
        addi  t1, t1, 1
        j     bucle
fin:    …
```

- With cache memory = 480480 ns
- Without cache memory = 8408 ns
- Hit ratio of cache = 4002 / 4004 => 99,95 %

¡ x57 !

Exercise
Calcular tasa de aciertos

```
int v[1000]; // global
...
int i;
int s;
for (i=0; i < 1000; i++)
     s = s + v[i];
```

- Example:
  - Acceso a caché: 2 ns
  - Acceso a MP: 120 ns
  - Block de MP: 4 palabras
  - Transferencia de un bloque entre memoria principal y caché: 200 ns

```
.data
       v: .space 4000 # 4*1000

.text
main:
       li   t0, 0    # i
       li   t1, 0    # i de v
       li   t2, 1000 # num. eltos.
       li   t3, 0    # s
bucle: bge  t0, t2, fin
       lw   t4, v(t1)
       add  t3, t3, t4
       addi t0, t0, 1
       addi t1, t1, 4
       j    bucle
fin:   …
```

Why does the cache work?

- Cache access time is much shorter than main memory access time.
- Main memory is accessed by blocks.

- When a program accesses an address, it is likely to access it again in the near future.
  - Temporary locality.
- When a program accesses an address, it is likely to access nearby positions in the near future.
  - Spatial location.

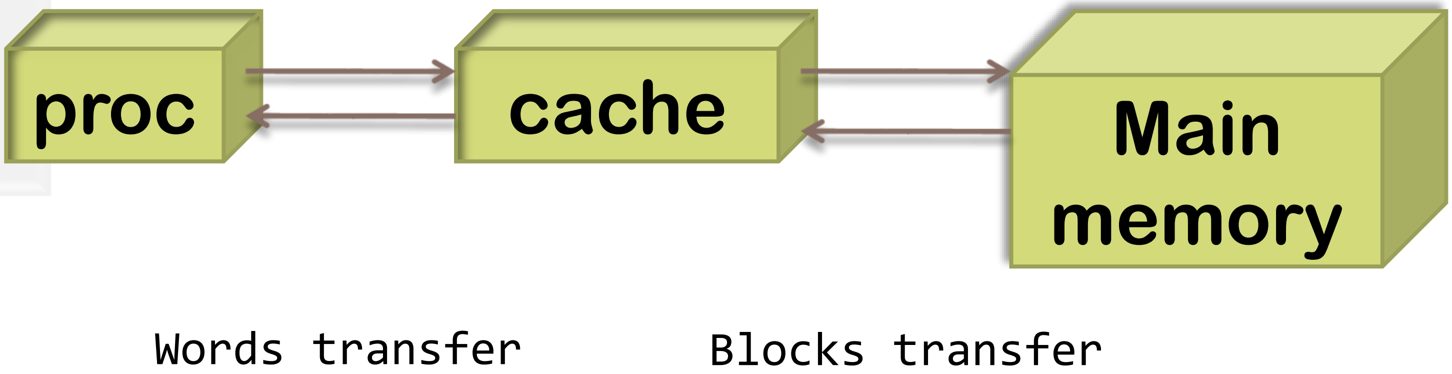
- Hit ratio: probability that an accessed data is in cache

High hit ratio

General operation

summary

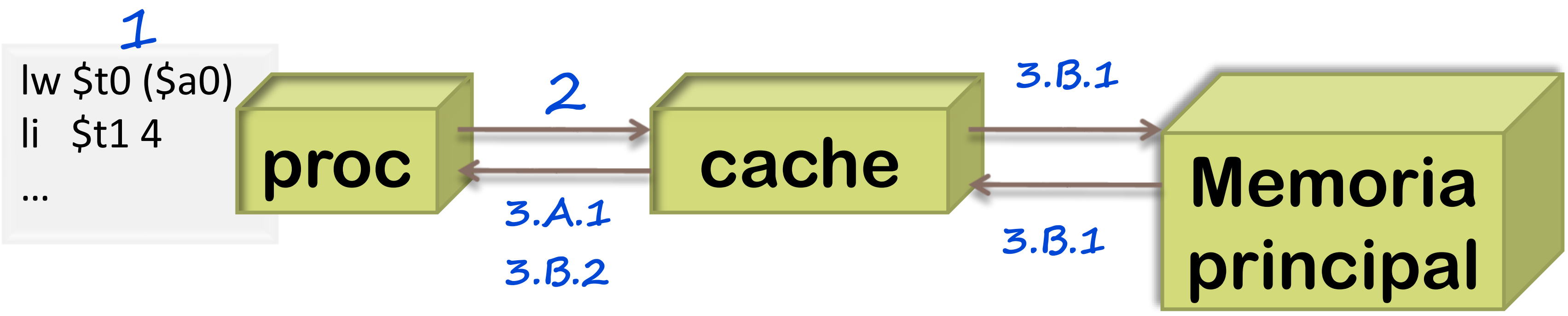
```
lw $t0 ($a0)
li   $t1 4
…
```



- Small amount of fast SRAM memory
  - Integrated in the Processor itself
  - Faster and more expensive than the DRAM
- Between main memory and processor
- Stores a copy of chunks of the main memory

General operation

summary



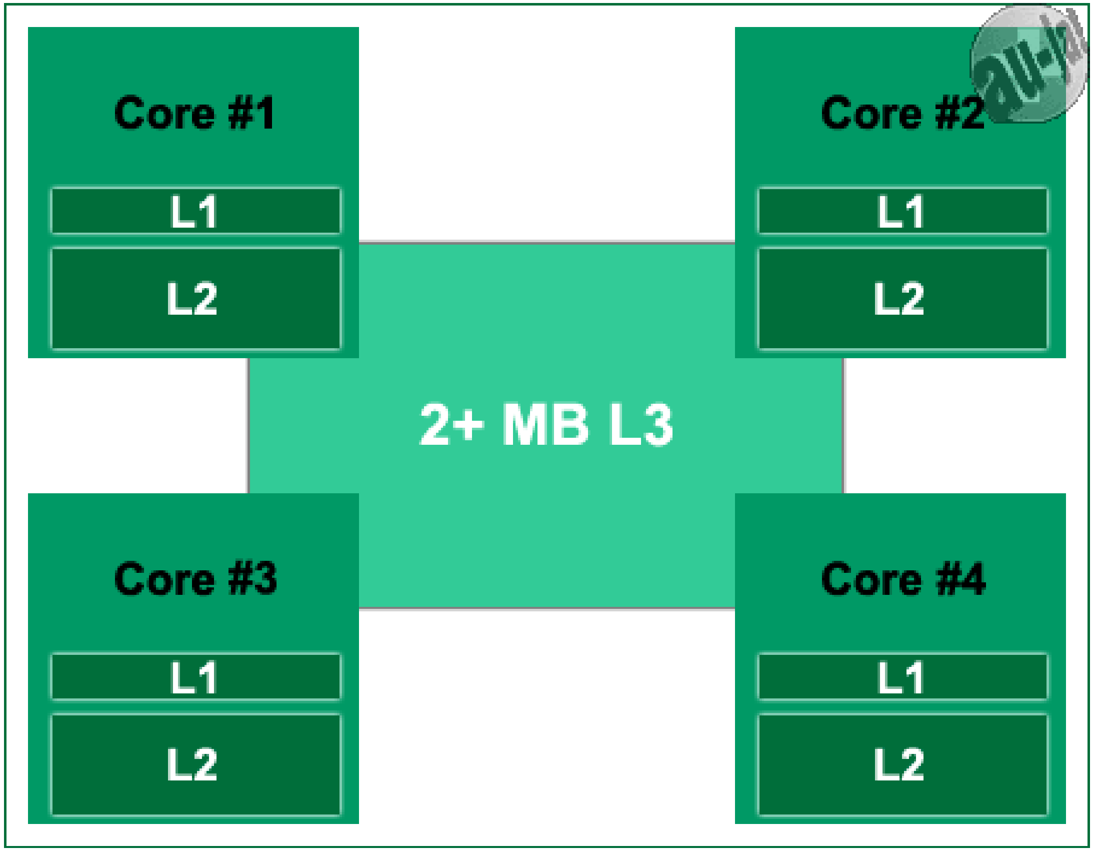
1. The Processor performs an access memory (cache intercepts).
2. The cache checks if the data for this position is already there:

- **If it is there (HIT),**
  - **3.A.1** It is served to the Processor from the cache (quickly): Ta
- **If it is not there (MISS),**
  - **3.B.1** The cache transfers from Main memory the block associated with position: Tf
  - **3.B.2** The cache then delivers the requested data to the processor: Ta

Cache memory levels

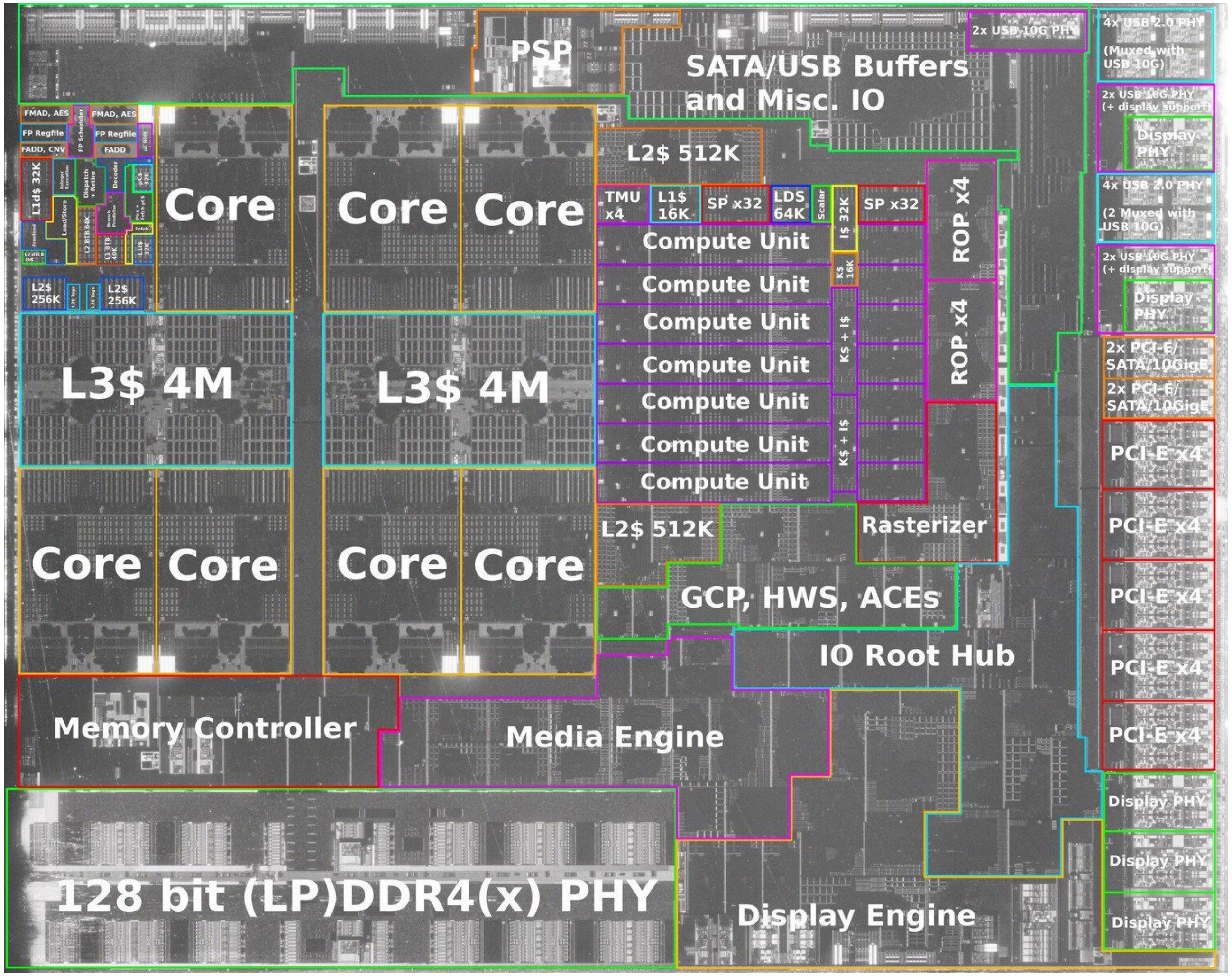
- It is common to find three levels:
  - L1 or level 1:
    - Internal cache: closest to the Processor
    - Small size (8KB-128KB) and maximum speed
    - Can be split for instructions and data
  - L2 or level 2:
    - Internal cache
    - Between L1 and L3 (or between L1 and main memory)
    - Medium size (256KB - 4MB) and lower speed than L1
  - L3 or level 3:
    - Typically, last level before main memory
    - Larger size and slower speed than L2
    - Internal or external to the processor

Example: AMD quad-core



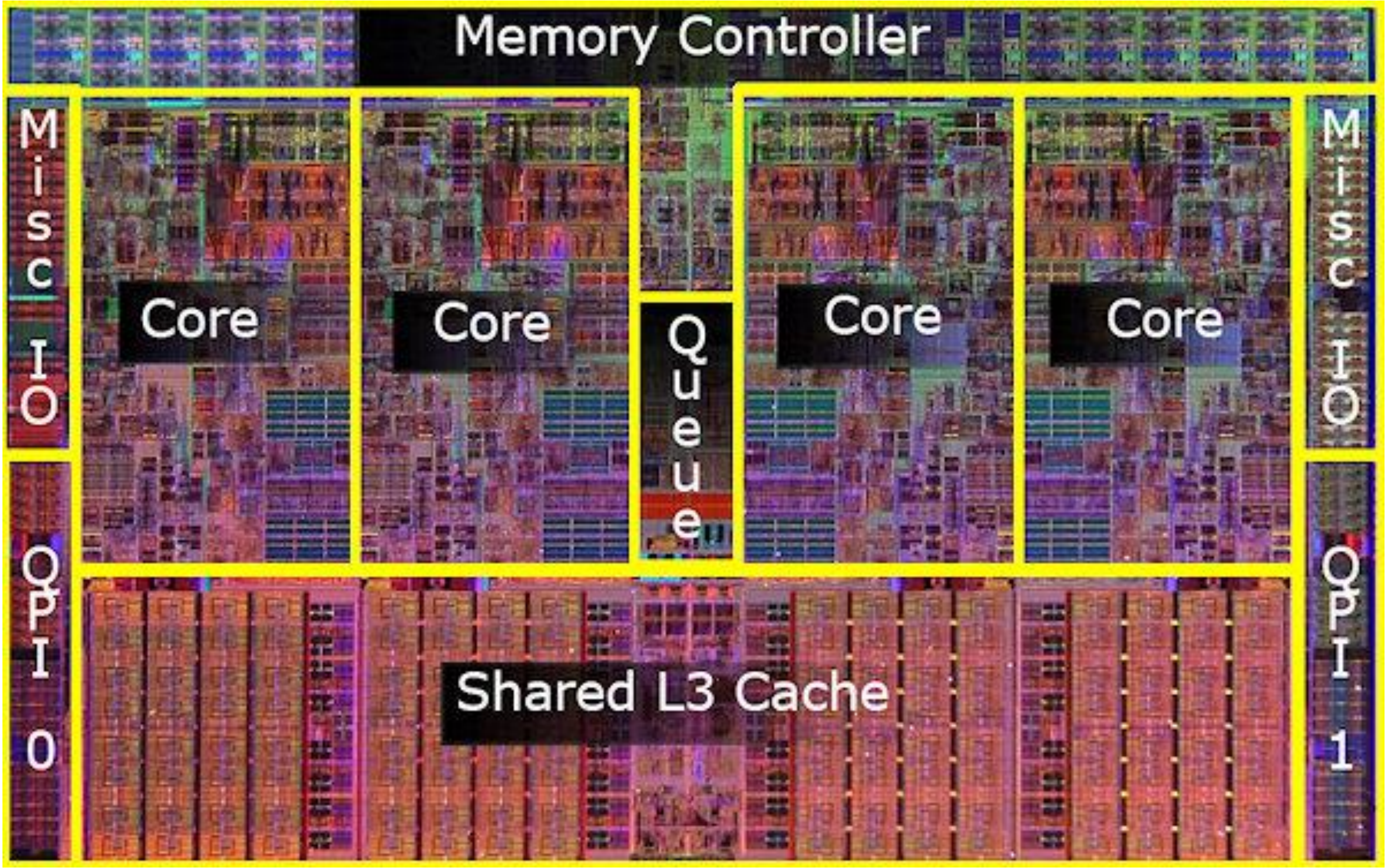
Quad-Core CPU mit gemeinsamen L3-Cache

Example: AMD Ryzen 4000



https://www.profesionalreview.com/2020/06/21/amd-ryzen-4000-apu-peg-x16/

Example: Intel Core i7



https://www.geeknetic.es/Review/607/AMD-Phenom-2-X4-980-Black-Edition.html

Contents

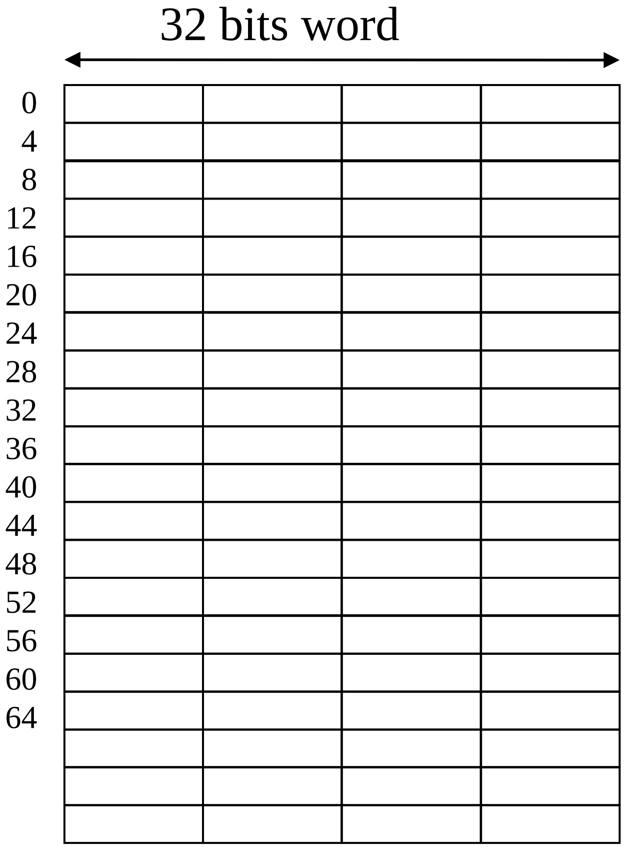
1. Types of memories
2. Memory hierarchy
3. Main memory
4. Cache memory
    1. Introduction
    2. Structure of the cache memory
    3. Cache design and organization
5. Virtual memory

Remainder

Main memory access

- Example:
  - 32-bit computer
  - Byte addressed memory
  - Main memory accessed by word
  - How to access to this address?

    0x00000064

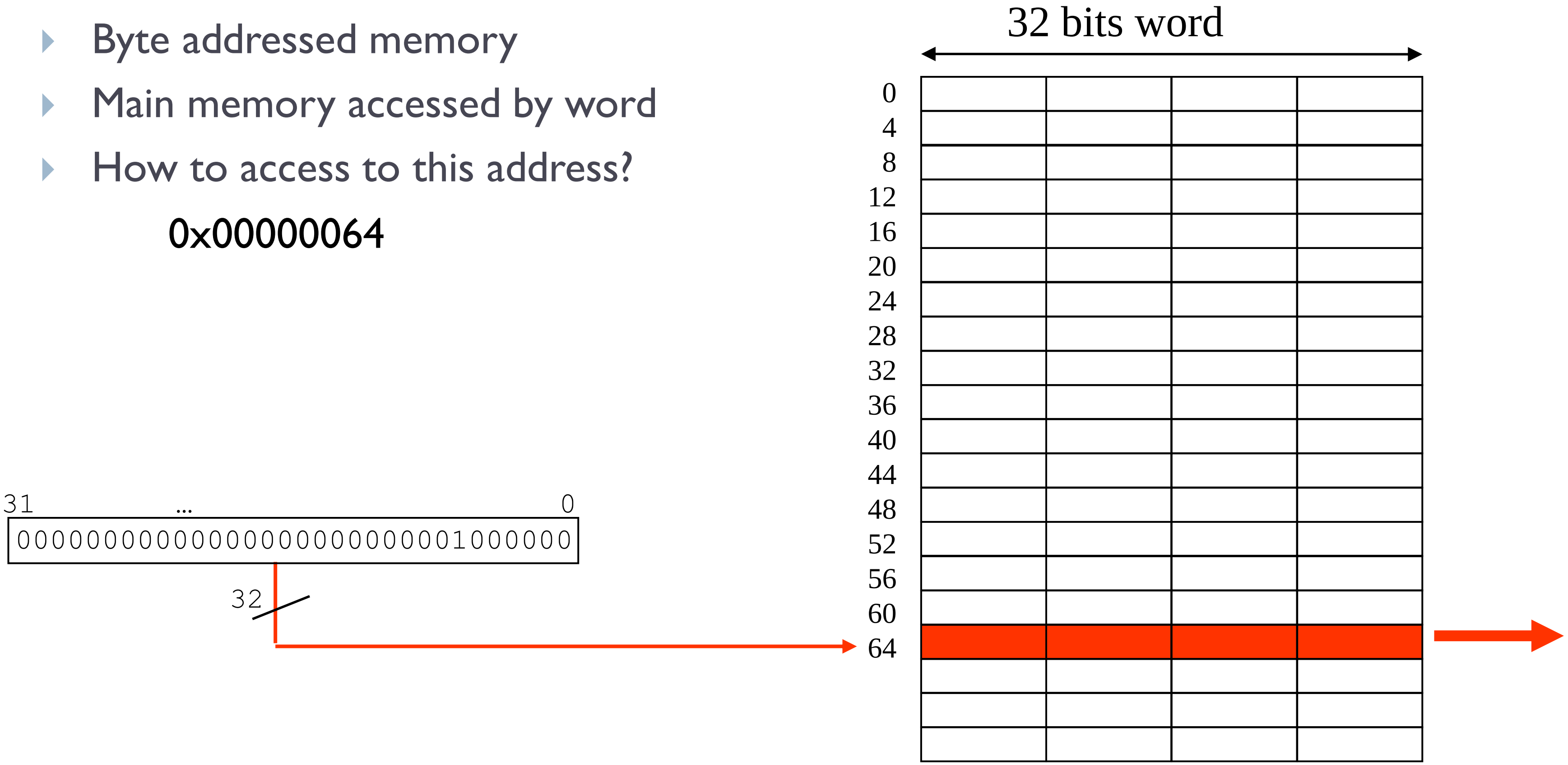


Main memory access

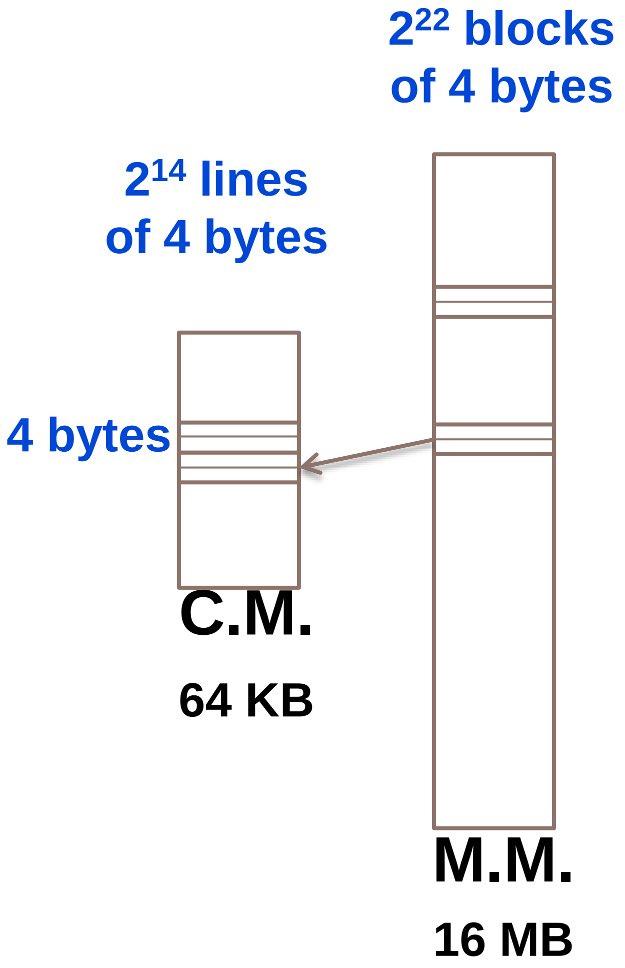
Remainder

- Example:
  - 32-bit computer
  - Byte addressed memory
  - Main memory accessed by word
  - How to access to this address?

    0x00000064

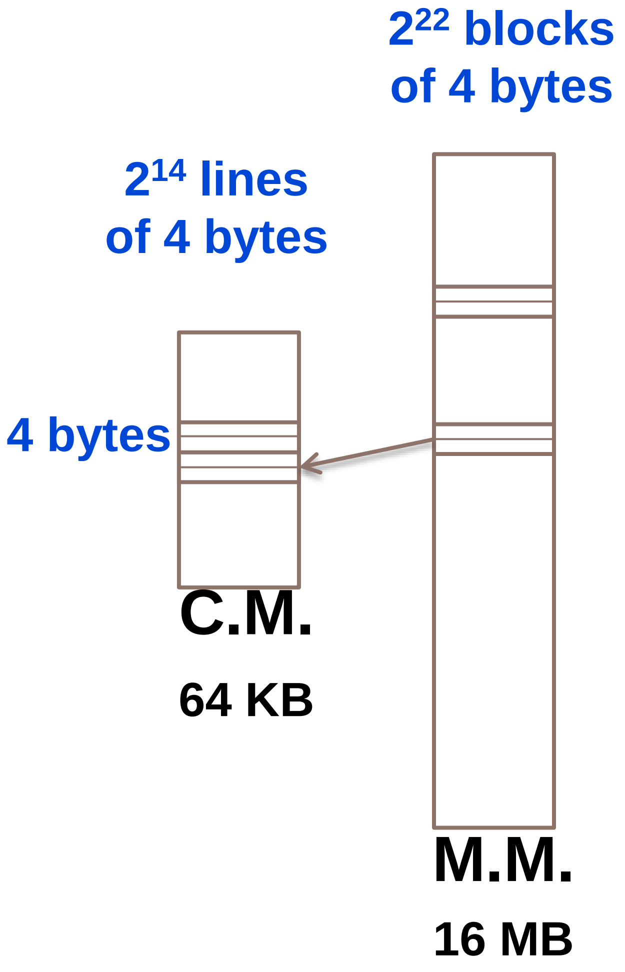


Structure of the cache memory



- M.M. and C.M. are divided into blocks of equal size.
  - The block in cache is call line
- Each M.M. block will have a corresponding C.M. line (block in cache)
- The size of the C.M. is smaller:
  - The number of blocks in the cache is small.

Structure of the cache memory



2^{22} blocks of 4 bytes

2^{14} lines of 4 bytes

4 bytes

C.M.

64 KB

M.M.

16 MB

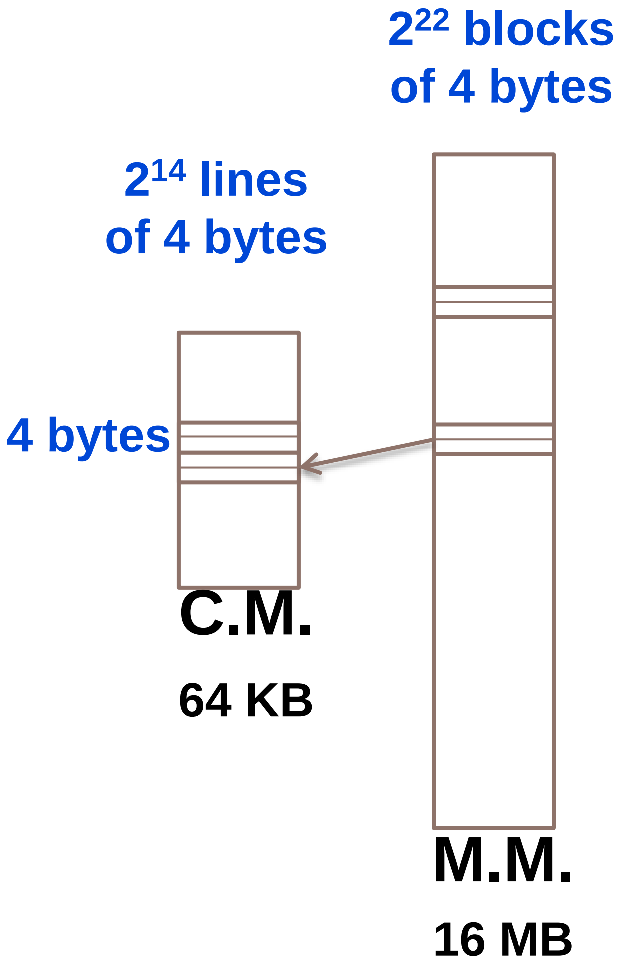
- M.M. and C.M. are divided into blocks of equal size.
  - The block in cache is call line
- Each M.M. block will have a corresponding C.M. line (block in cache)
- The size of the C.M. is smaller:
  - The number of blocks in the cache is small.

How many blocks of 4 words can fit in a 64 KB cache memory in a 32-bits CPU?

- Solución:

  $2^6 \cdot 2^{10}$ bytes / 2^4 bytes = 2^{11} blocks = 2048 blocks = 2048 lines

Structure of the cache memory



- M.M. and C.M. are divided into blocks of equal size.
  - The block in cache is call line
- Each M.M. block will have a corresponding C.M. line (block in cache)
- The size of the C.M. is smaller:
  - The number of blocks in the cache is small.

How many blocks of 4 words can fit in a 1 GiB memory in a 32-bits CPU?

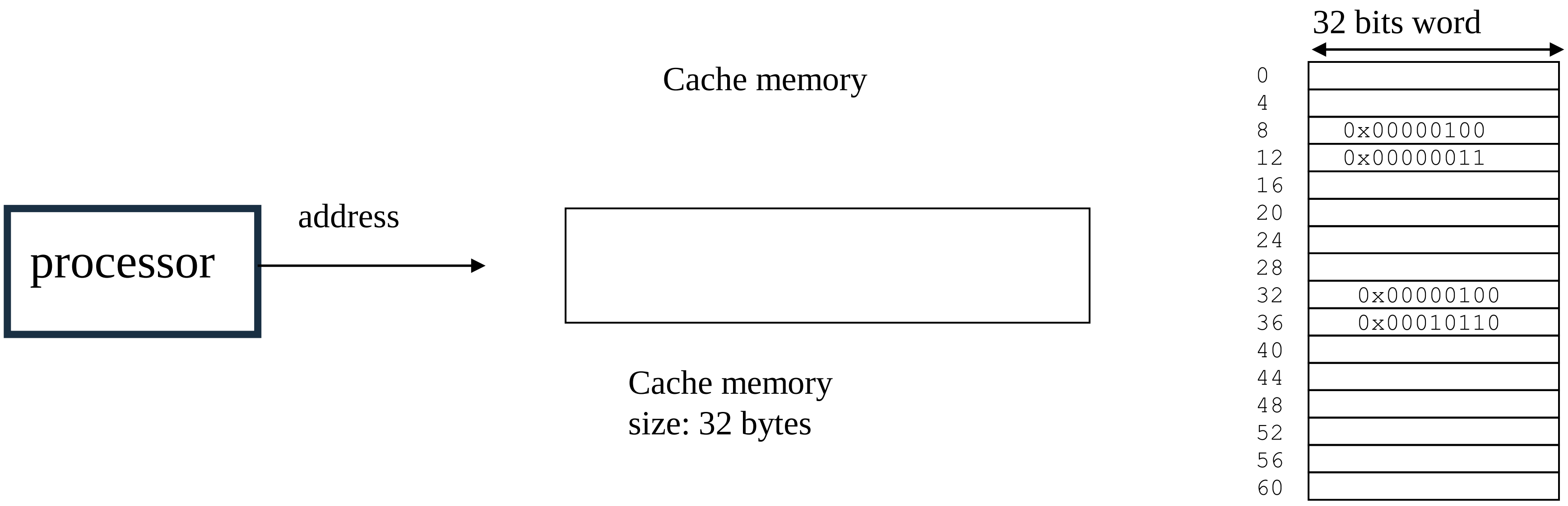
- Solución:

2^{30} b / 2^{4} b = 2^{30-4} b = 2^{26} b =
64 megablocks = ~64 millions

Locating a word in the cache

Example:

With blocks of 2 words
How many lines does the cache have?

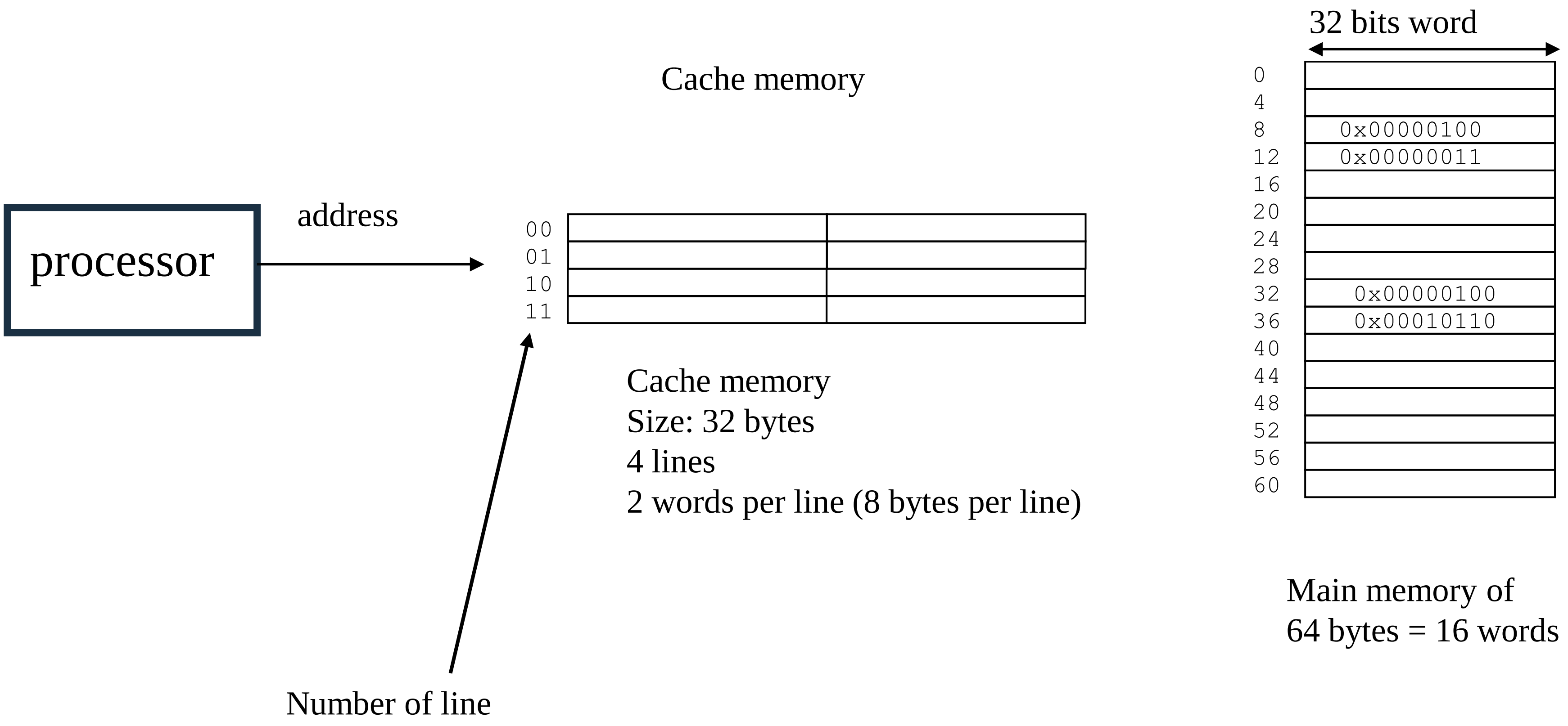


0	
4	
8	0x00000100
12	0x00000011
16	
20	
24	
28	
32	0x00000100
36	0x00010110
40	
44	
48	
52	
56	
60	

Cache memory
size: 32 bytes

Main memory of
64 bytes = 16 words

Locating a word in the cache

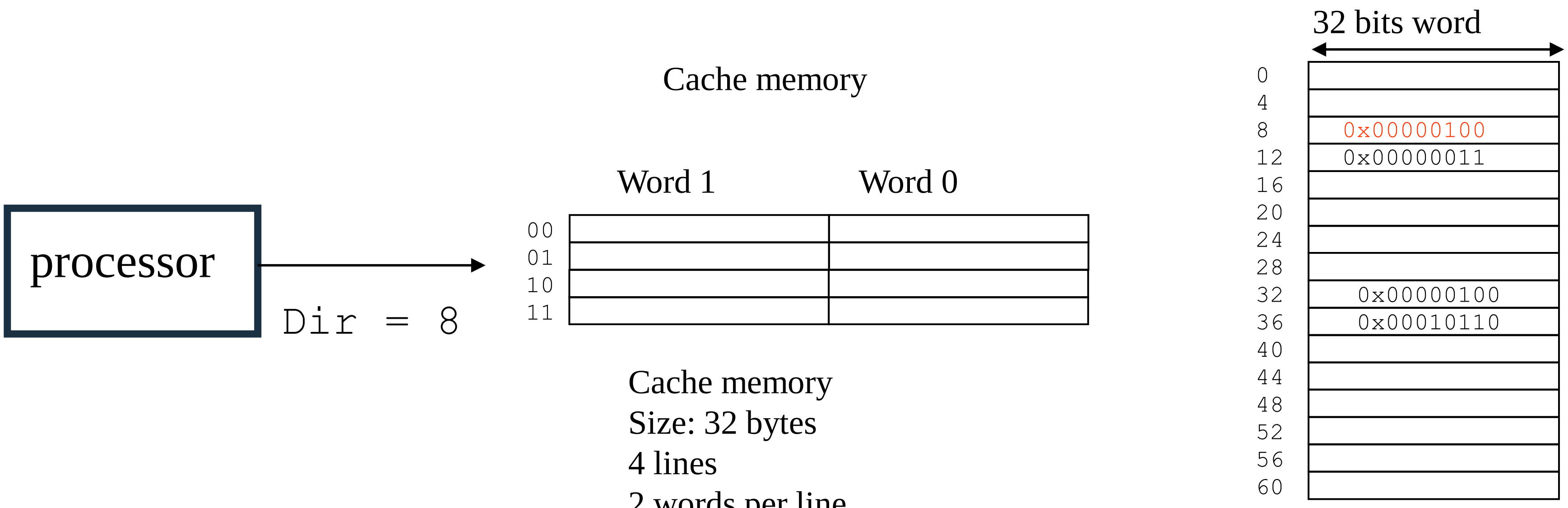


Cache memory
Size: 32 bytes
4 lines
2 words per line (8 bytes per line)

0	
4	
8	0x00000100
12	0x00000011
16	
20	
24	
28	
32	0x00000100
36	0x00010110
40	
44	
48	
52	
56	
60	

Main memory of
64 bytes = 16 words

Locating a word in the cache

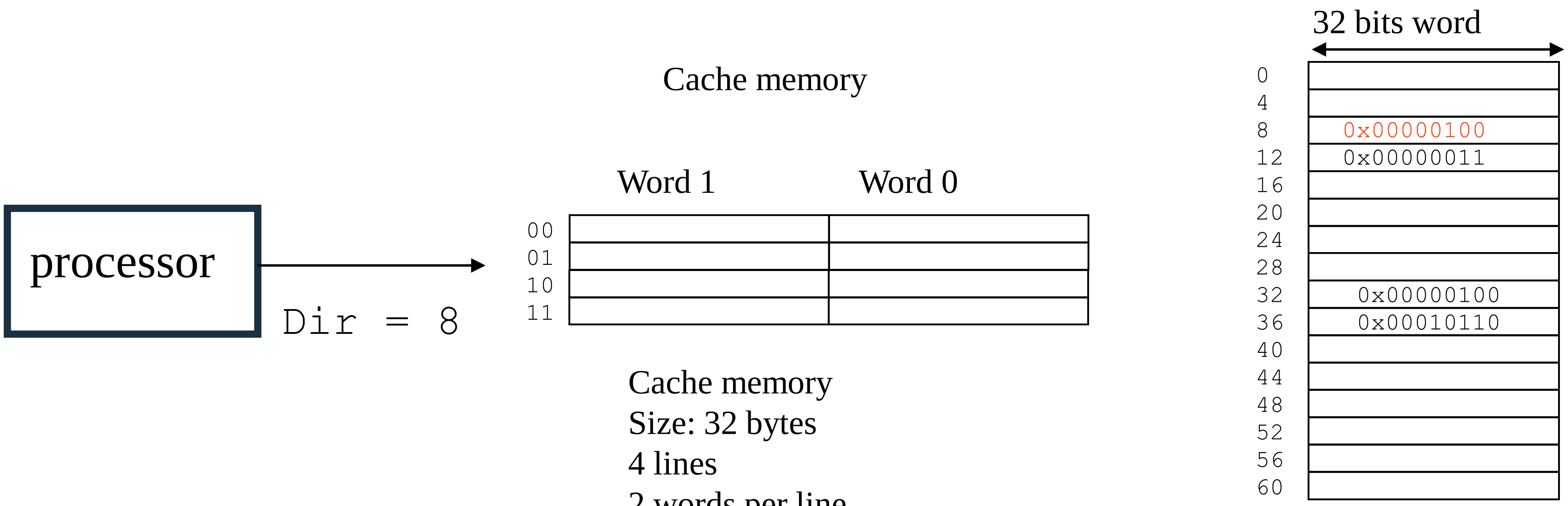


0	
4	
8	0x00000100
12	0x00000011
16	
20	
24	
28	
32	0x00000100
36	0x00010110
40	
44	
48	
52	
56	
60	

Cache memory
Size: 32 bytes
4 lines
2 words per line

Main memory of
64 bytes = 16 words

Locating a word in the cache



	Word 1	Word 0
00		
01		
10		
11		

Cache memory
Size: 32 bytes
4 lines
2 words per line

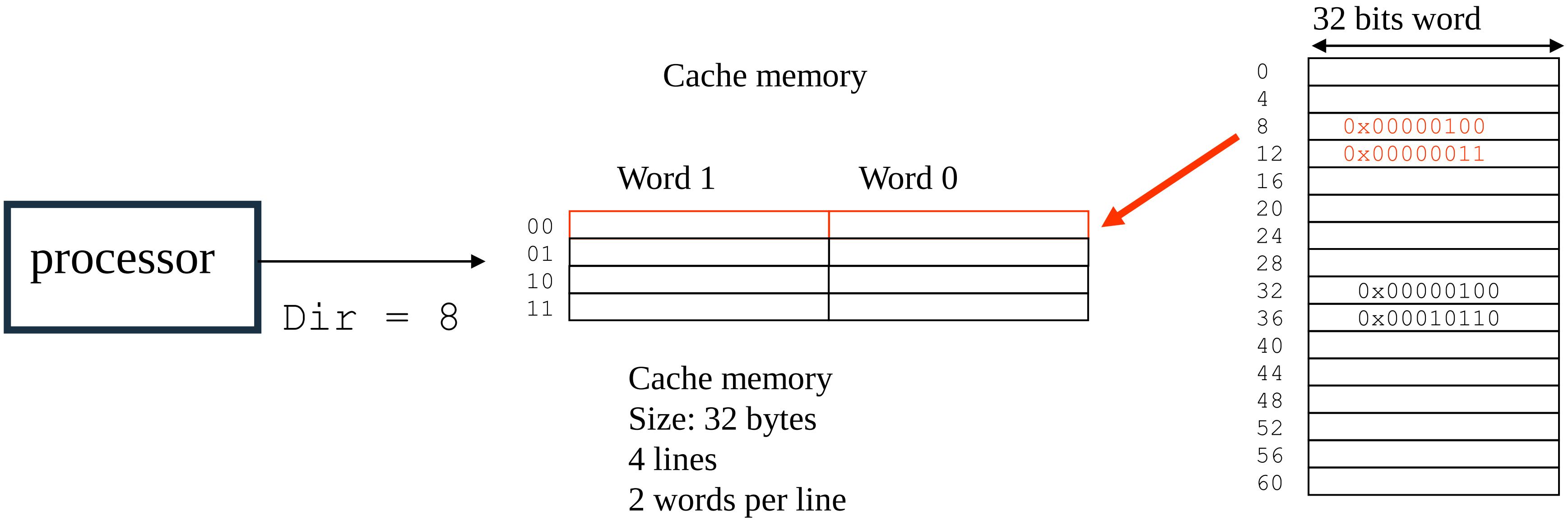
0	
4	
8	0x00000100
12	0x00000011
16	
20	
24	
28	
32	0x00000100
36	0x00010110
40	
44	
48	
52	
56	
60	

Main memory of
64 bytes = 16 words

MISS

How do you know?

Locating a word in the cache

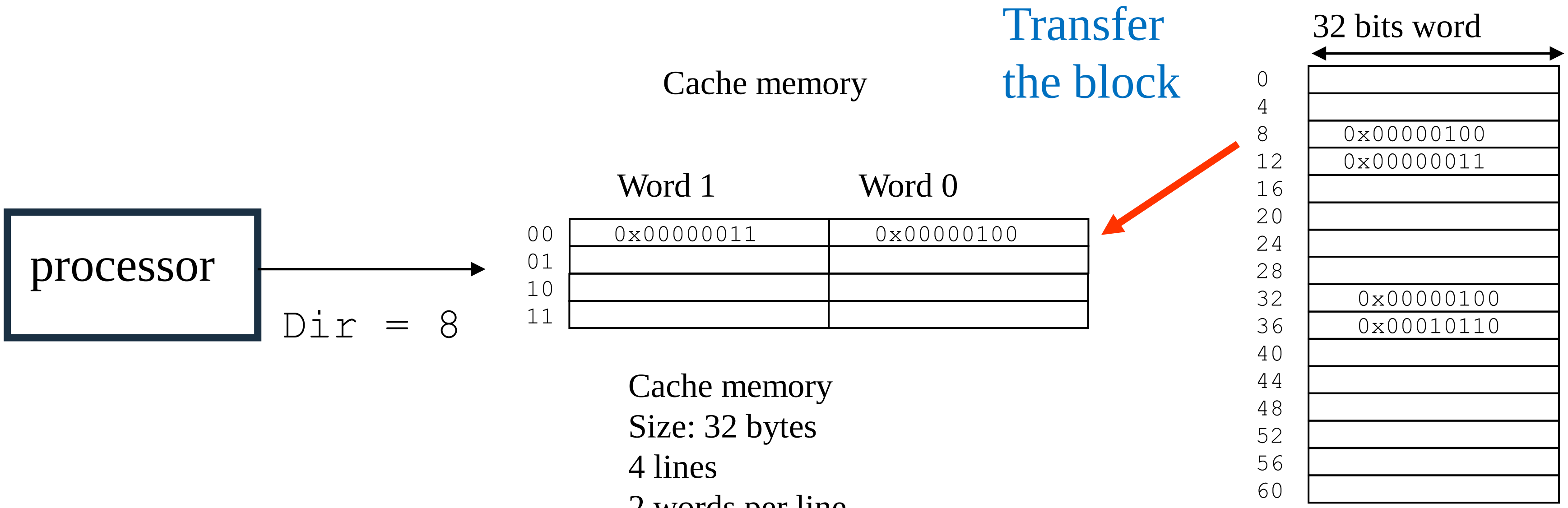


Cache memory
Size: 32 bytes
4 lines
2 words per line

Main memory of
64 bytes = 16 words

A line is selected in the cache
Which line?

Locating a word in the cache



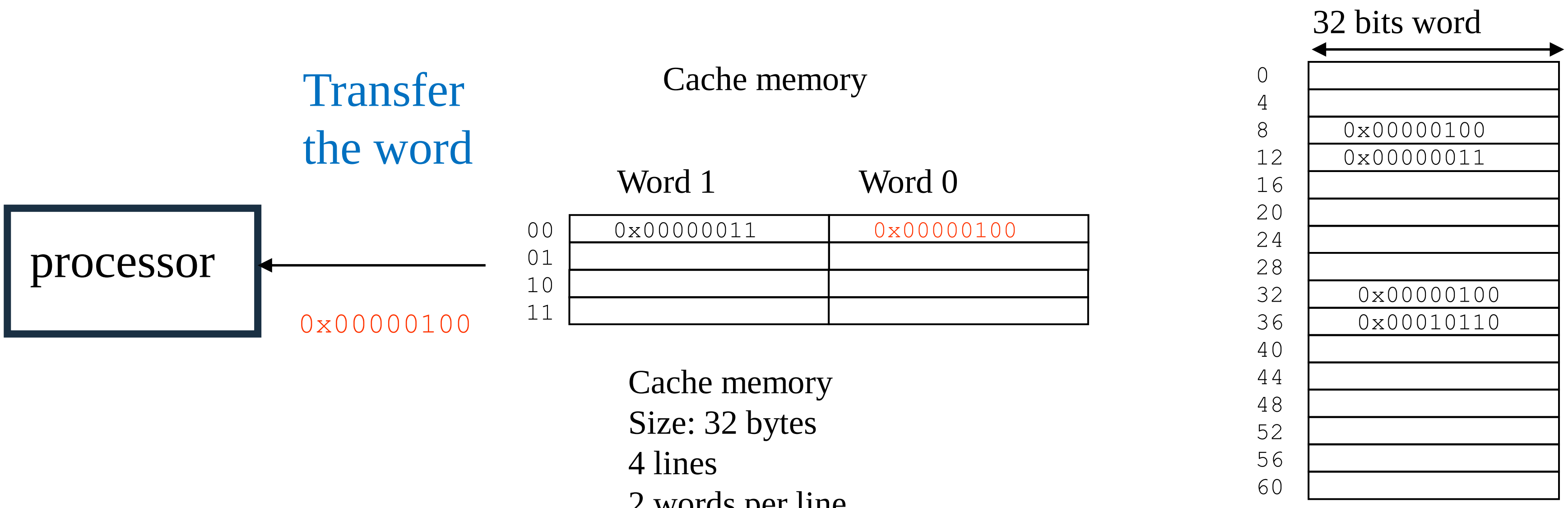
	Word 1	Word 0
00	0x00000011	0x00000100
01		
10		
11		

Cache memory
Size: 32 bytes
4 lines
2 words per line

0	
4	
8	0x00000100
12	0x00000011
16	
20	
24	
28	
32	0x00000100
36	0x00010110
40	
44	
48	
52	
56	
60	

Main memory of
64 bytes = 16 words

Locating a word in the cache



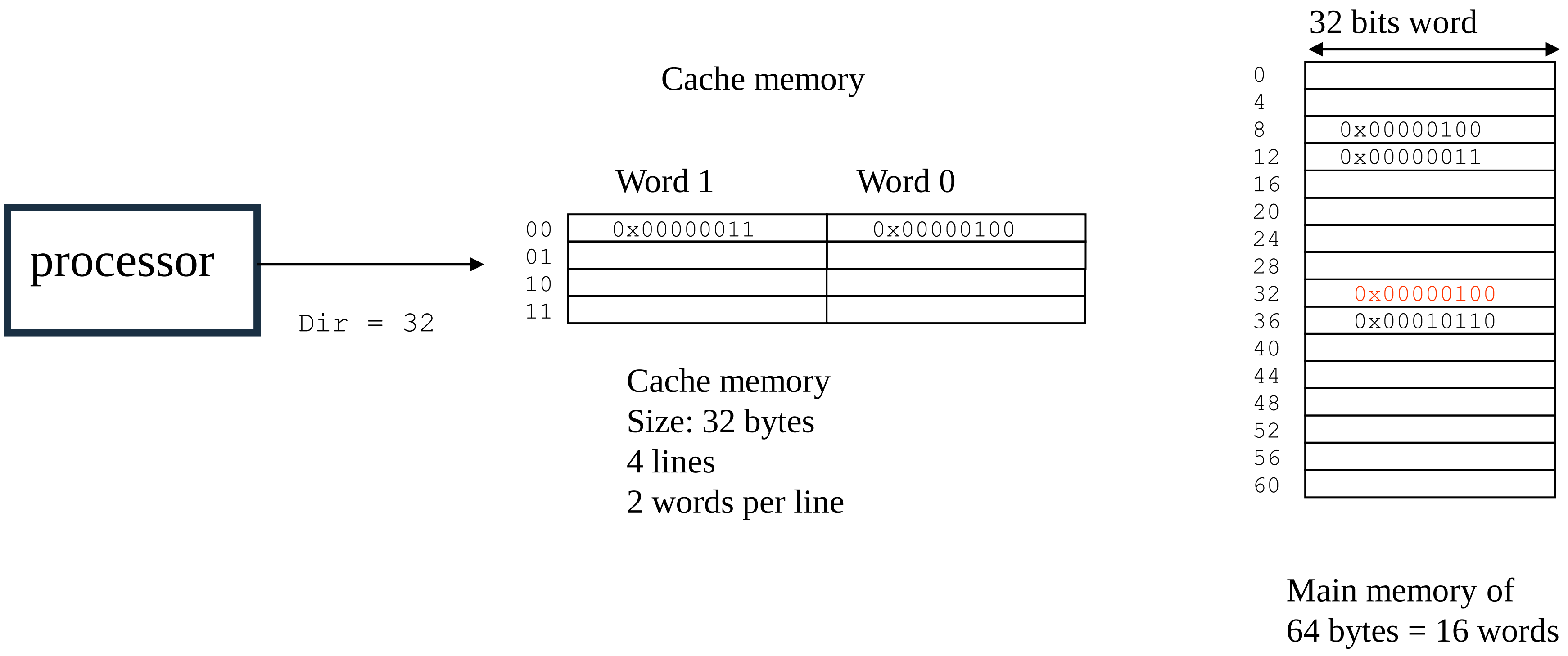
	Word 1	Word 0
00	0x00000011	0x00000100
01		
10		
11		

0	
4	
8	0x00000100
12	0x00000011
16	
20	
24	
28	
32	0x00000100
36	0x00010110
40	
44	
48	
52	
56	
60	

Cache memory
Size: 32 bytes
4 lines
2 words per line

Main memory of
64 bytes = 16 words

Locating a word in the cache



Cache memory

	Word 1	Word 0
00	0x00000011	0x00000100
01		
10		
11		

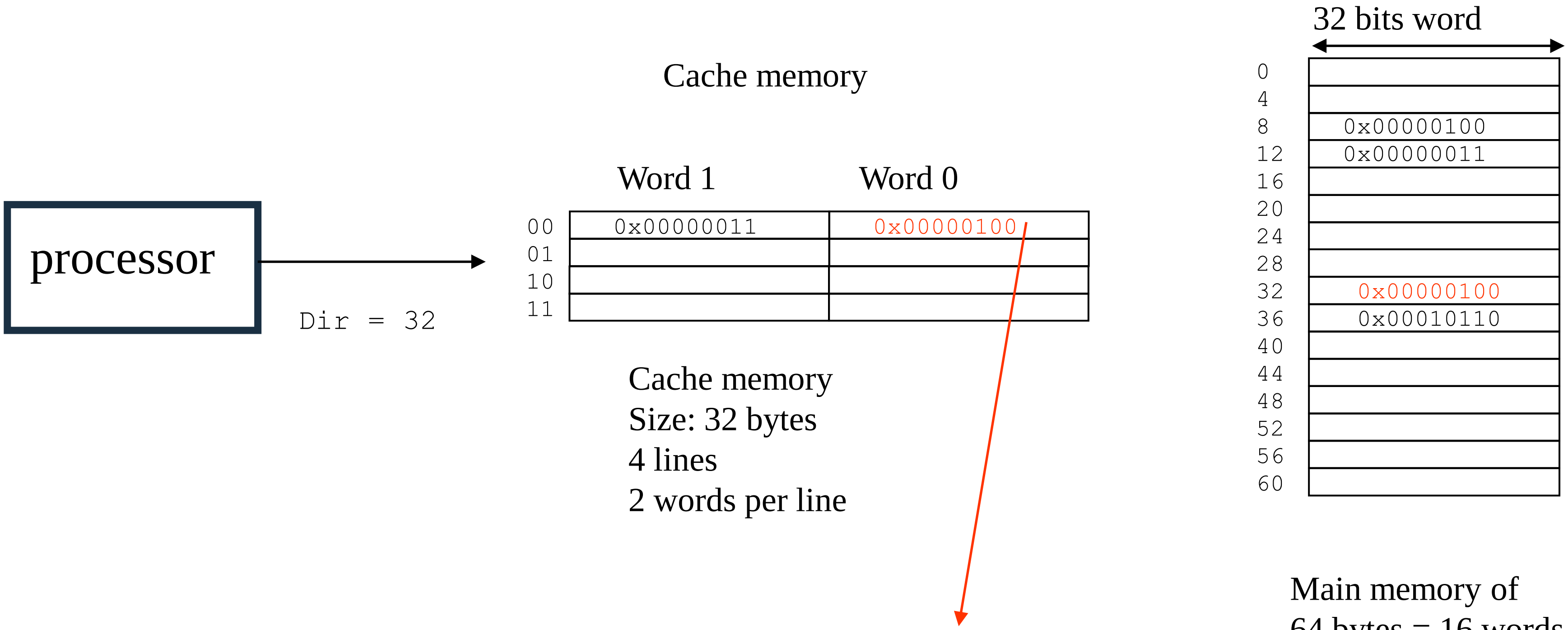
Cache memory
Size: 32 bytes
4 lines
2 words per line

	32 bits word
0	
4	
8	0x00000100
12	0x00000011
16	
20	
24	
28	
32	0x00000100
36	0x00010110
40	
44	
48	
52	
56	
60	

Main memory of
64 bytes = 16 words

How to know if it is in the cache?

Locating a word in the cache



	Word 1	Word 0
00	0x00000011	0x00000100
01		
10		
11		

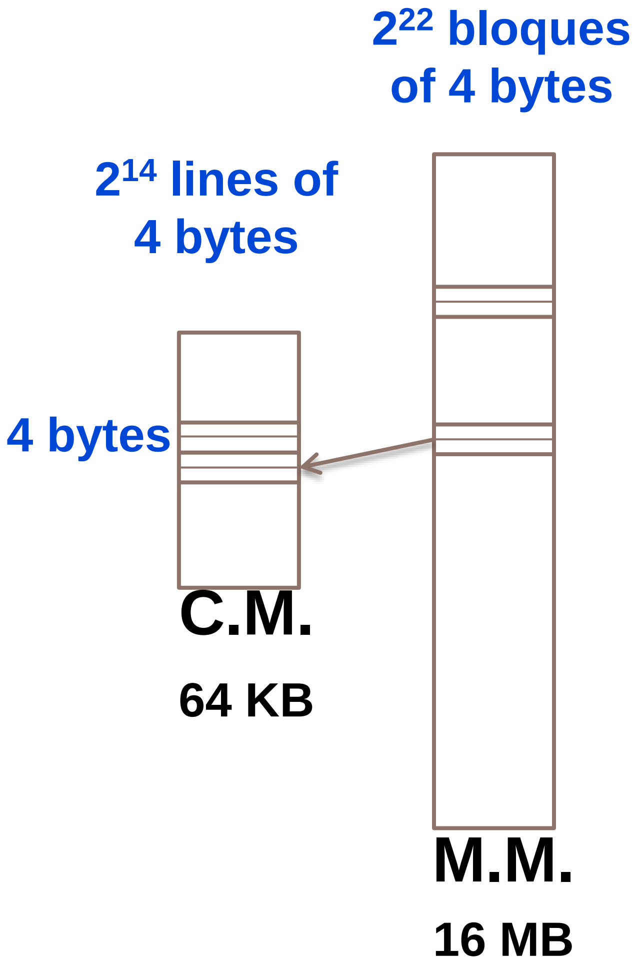
Cache memory
Size: 32 bytes
4 lines
2 words per line

0	
4	
8	0x00000100
12	0x00000011
16	
20	
24	
28	
32	0x00000100
36	0x00010110
40	
44	
48	
52	
56	
60	

Main memory of
64 bytes = 16 words

Reminder: the content is from address 8 not 32.

Structure of the cache memory



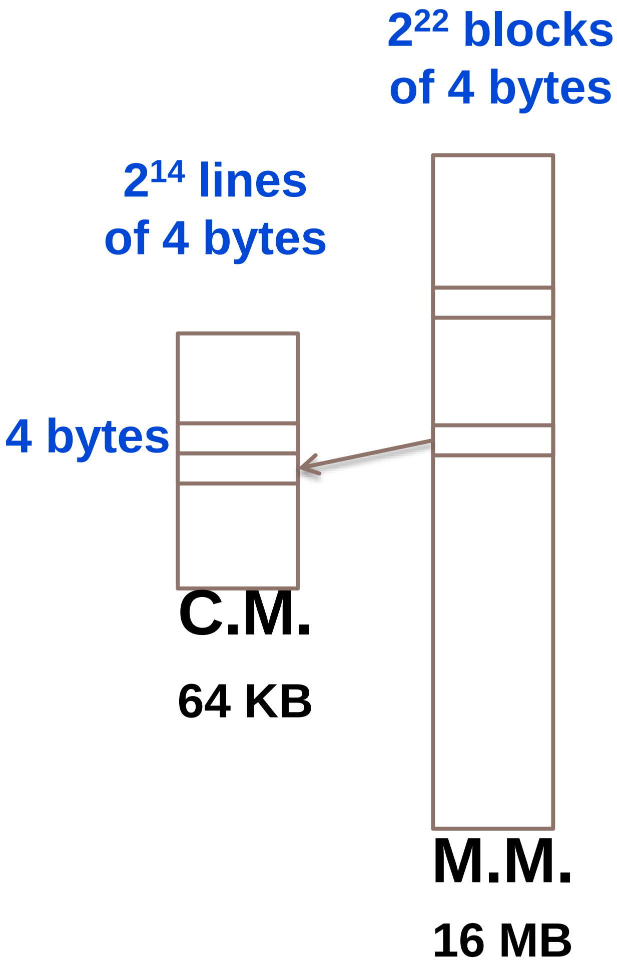
- M.M. and C.M. are divided into blocks of equal size.
  - The block in cache is call line
- Each M.M. block will have a corresponding C.M. line (block in cache)
- The size of the C.M. is smaller:
  - The number of blocks in the cache is small.

1. Where is an M.M. block located?
2. How is an M.M. block identified?
3. In case of miss and C.M. full…Which block should be replaced?
4. In case of write…What should be updated…C.M.? M.M. and C.M.?

Contents

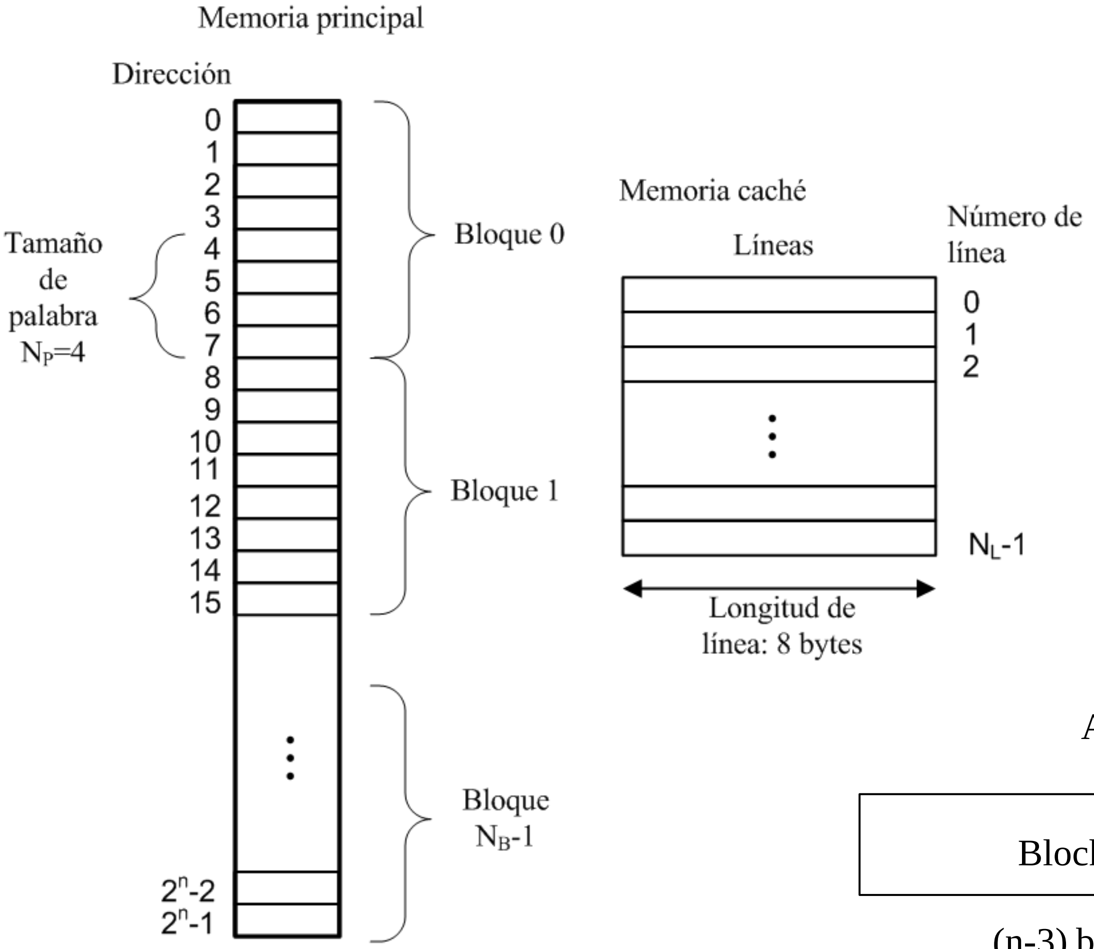
1. Types of memories
2. Memory hierarchy
3. Main memory
4. Cache memory
    1. Introduction
    2. Structure of the cache memory
    3. Cache design and organization
5. Virtual memory

Cache structure and design



- M.M. and C.M. are divided into blocks of equal size.
- Each M.M. block will have a corresponding C.M. line (block in cache)

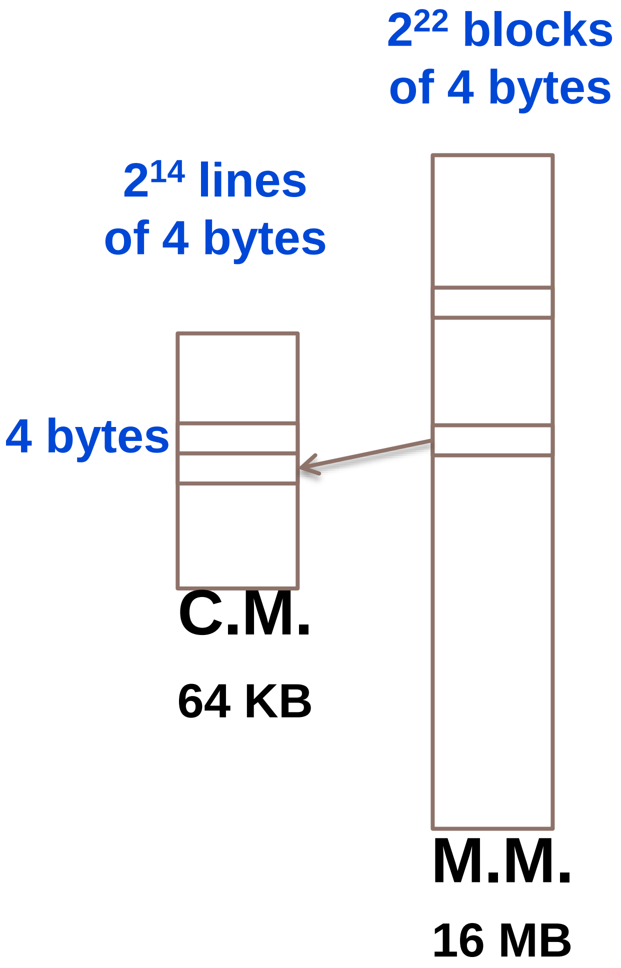
Example of cache organization



Address of n bits

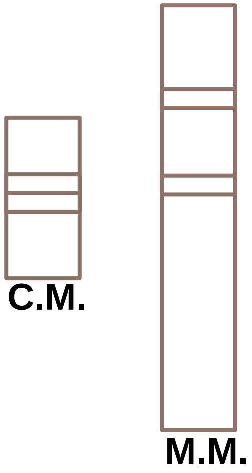
Block Id. (# block)	Byte in block
(n-3) bits	3 bits

Cache structure and design



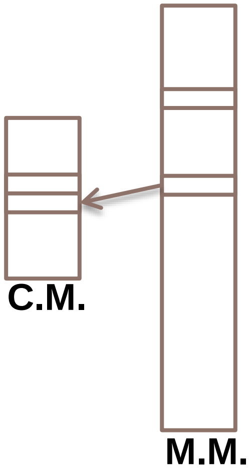
- M.M. and C.M. are divided into blocks of equal size.
- Each M.M. block will have a corresponding C.M. line (block in cache)
- The design determines:
  - Size
  - Mapping function
  - Replacement Algorithm
  - Write policy
- Different designs for L1, L2, ... are common.

Cache size



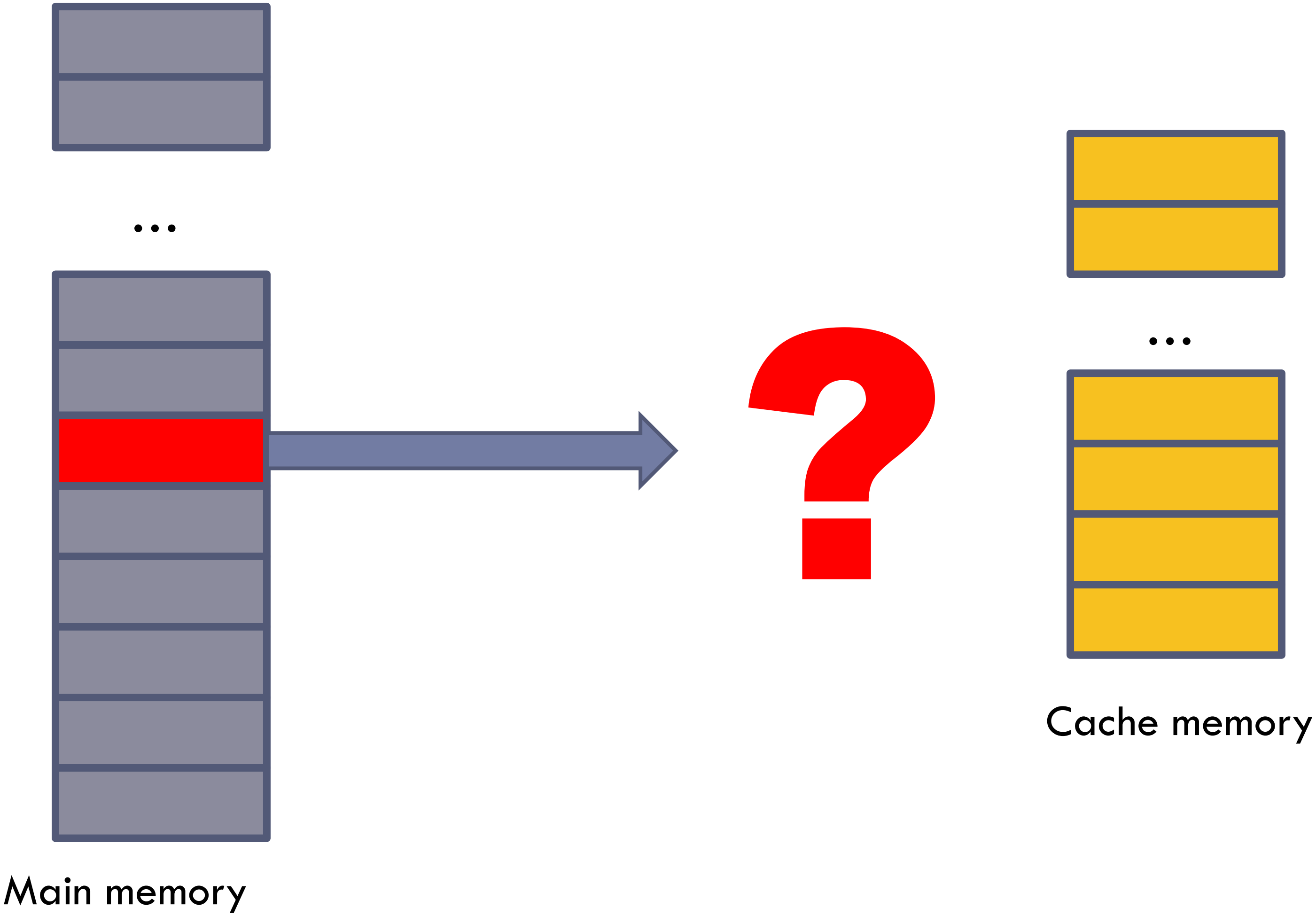
- Sizes:
  - The total cache memory size.
  - The size of the lines into which it is organized.
- Determined by studies on widely used codes

Mapping function

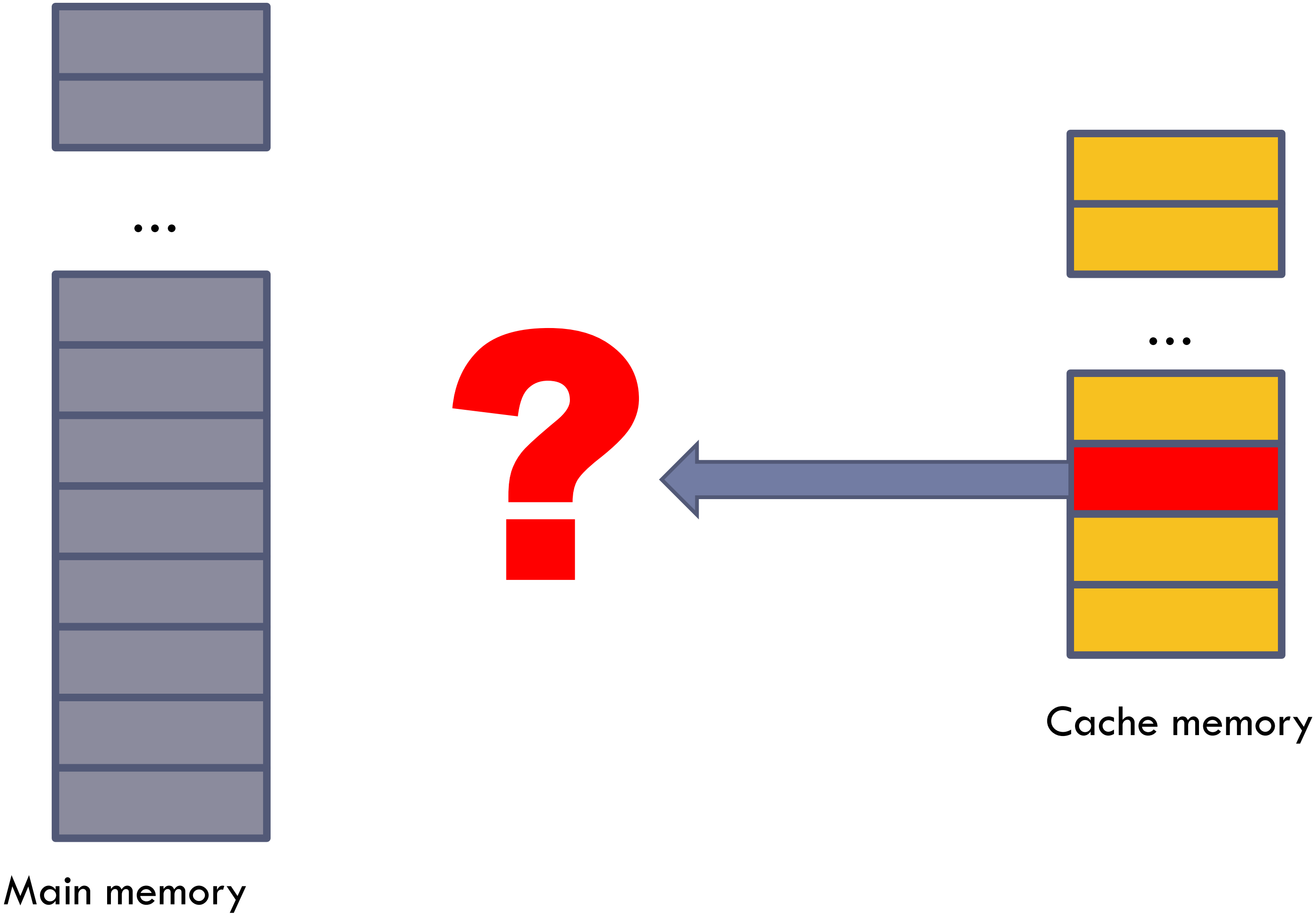


- Algorithm that determines where in the cache memory a specific block of the main memory can be stored.
- A mechanism that allows to know which specific block of main memory is in a line of the cache memory (or if it is free).
  - Labels are associated with the lines.
  - The labels are based on the starting address of the line.

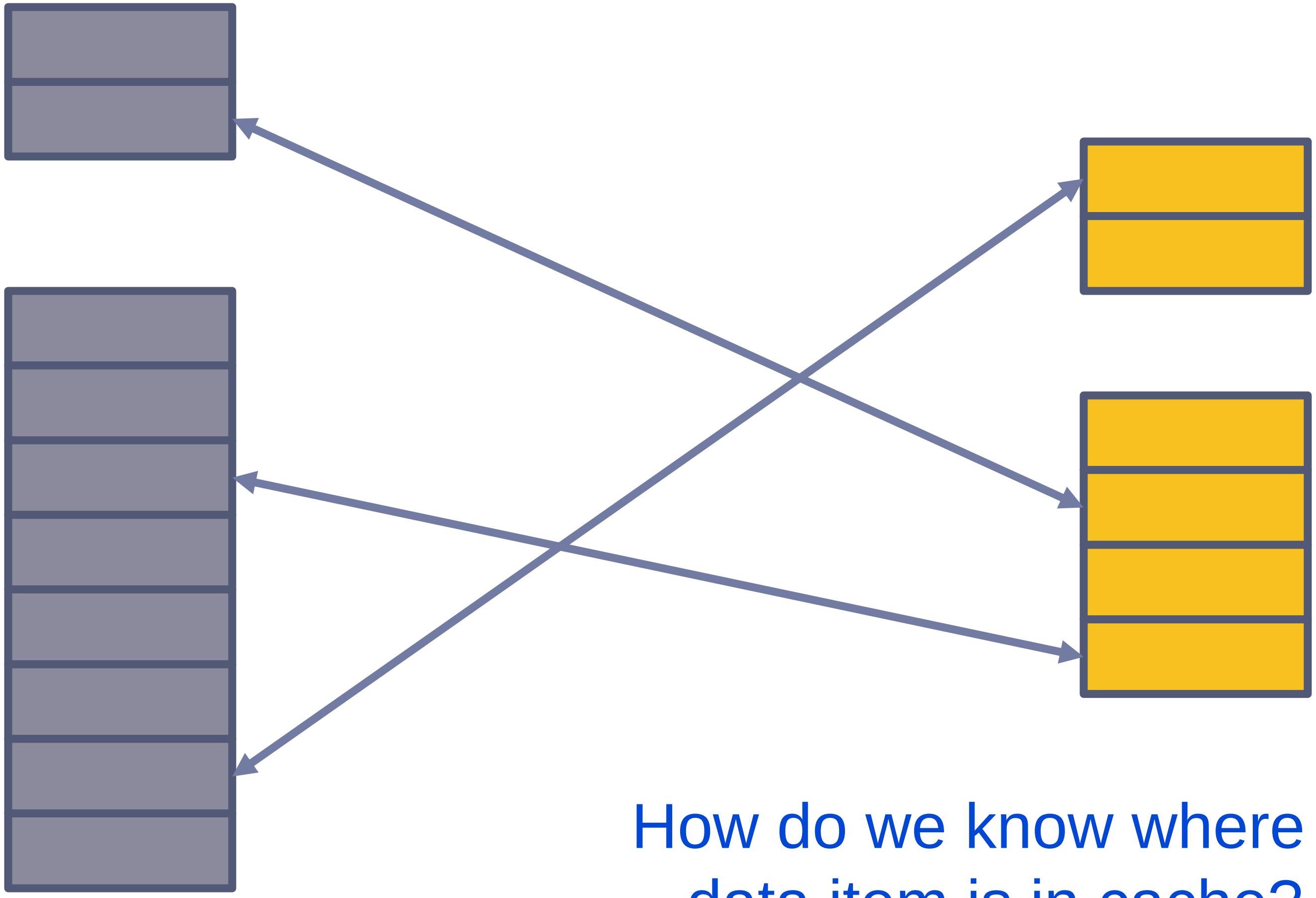
Location in cache memory



Location in cache memory



Mapping function

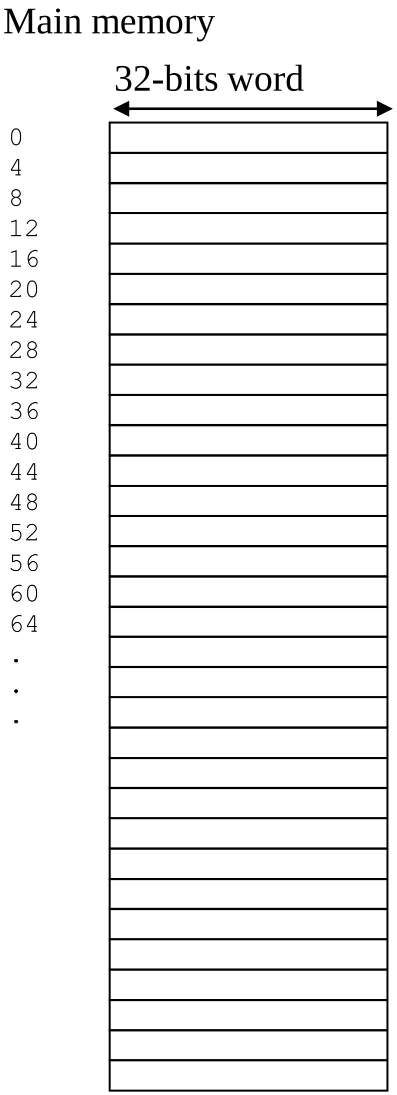


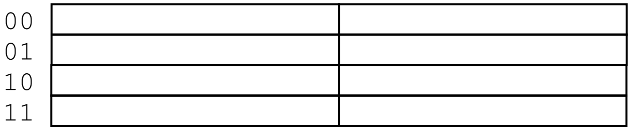
How do we know where a data item is in cache?

Mapping functions

- **Direct mapping** function
- **Associative** mapping function
- **Set associative** mapping function

Direct mapped

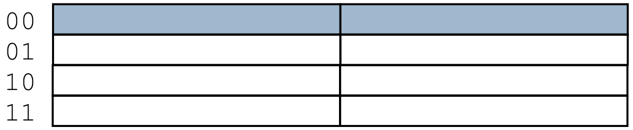




Cache memory
Size: 32 bytes
4 lines
2 words per line

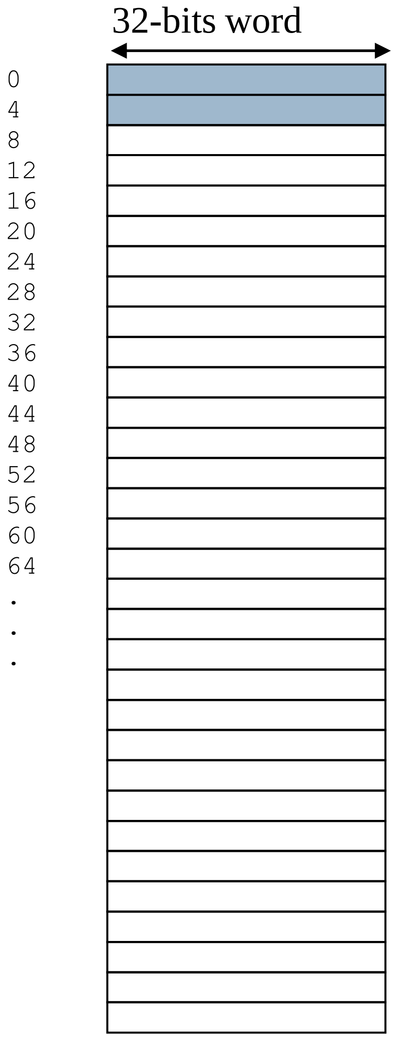
Direct mapped

Block 0 - line 0

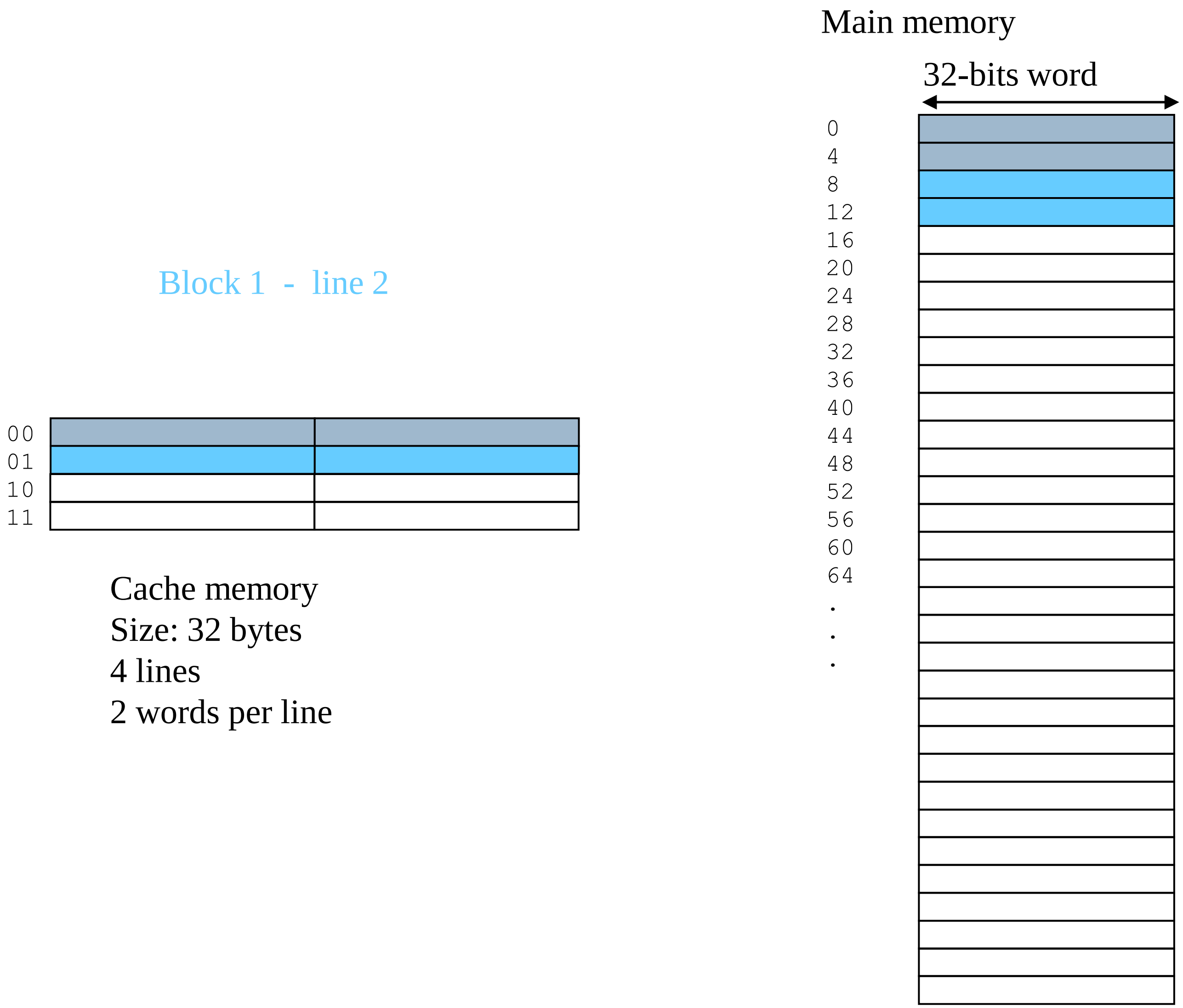


Cache memory
Size: 32 bytes
4 lines
2 words per line

Main memory



Direct mapped



Main memory

32-bits word

0
4
8
12
16
20
24
28
32
36
40
44
48
52
56
60
64
.
.
.

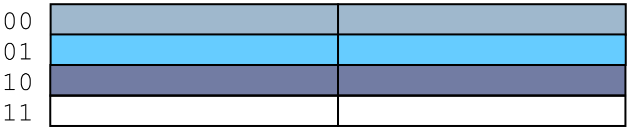
Block 1 - line 2

00
01
10
11

Cache memory
Size: 32 bytes
4 lines
2 words per line

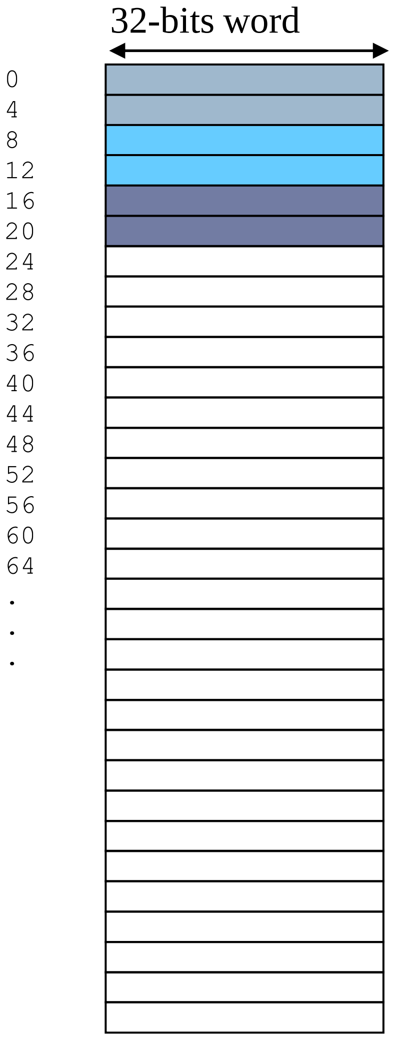
Direct mapped

Block 2 - line 2



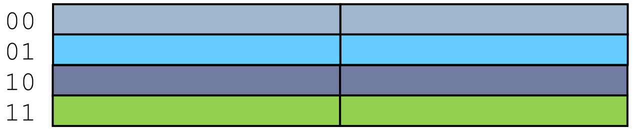
Cache memory
Size: 32 bytes
4 lines
2 words per line

Main memory



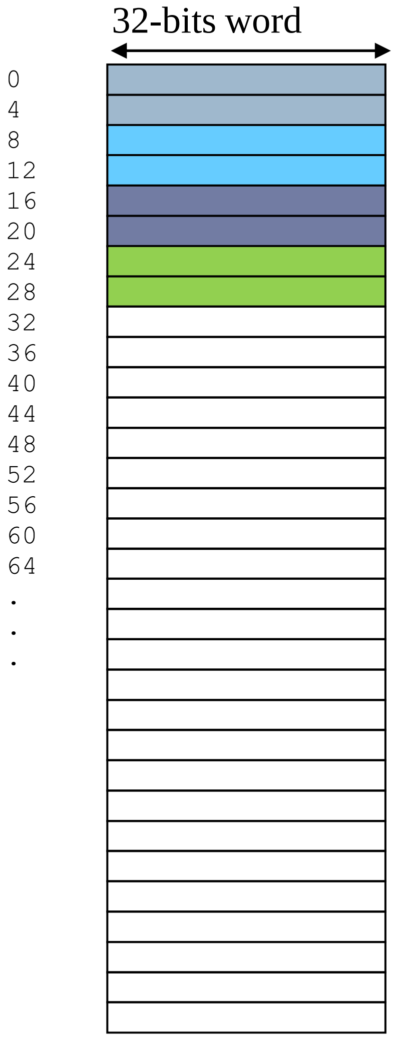
Direct mapped

Block 3 - line 3



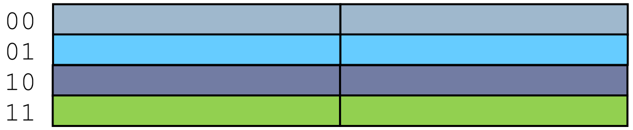
Cache memory
Size: 32 bytes
4 lines
2 words per line

Main memory



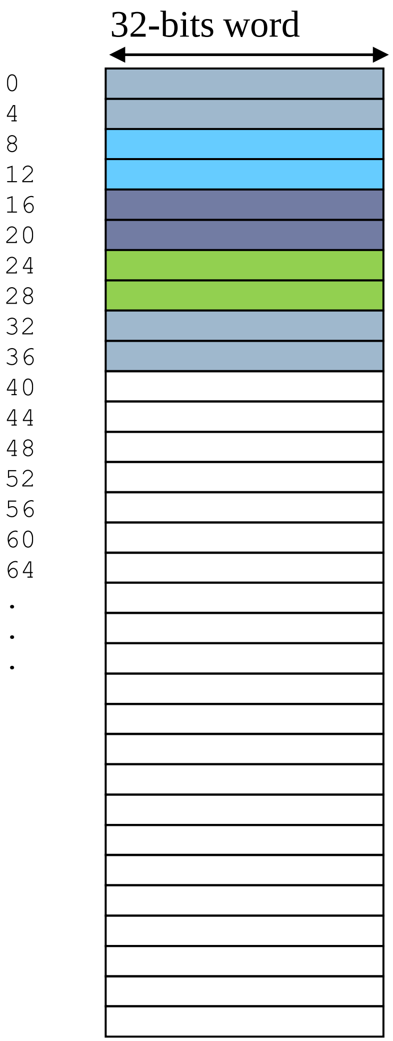
Direct mapped

Block 4 - line 0

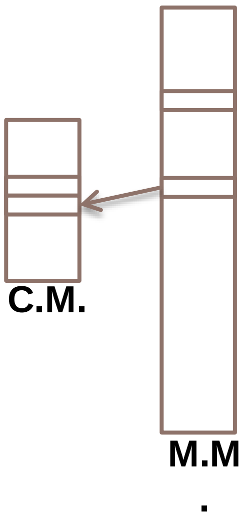


Cache memory
Size: 32 bytes
4 lines
2 words per line

Main memory



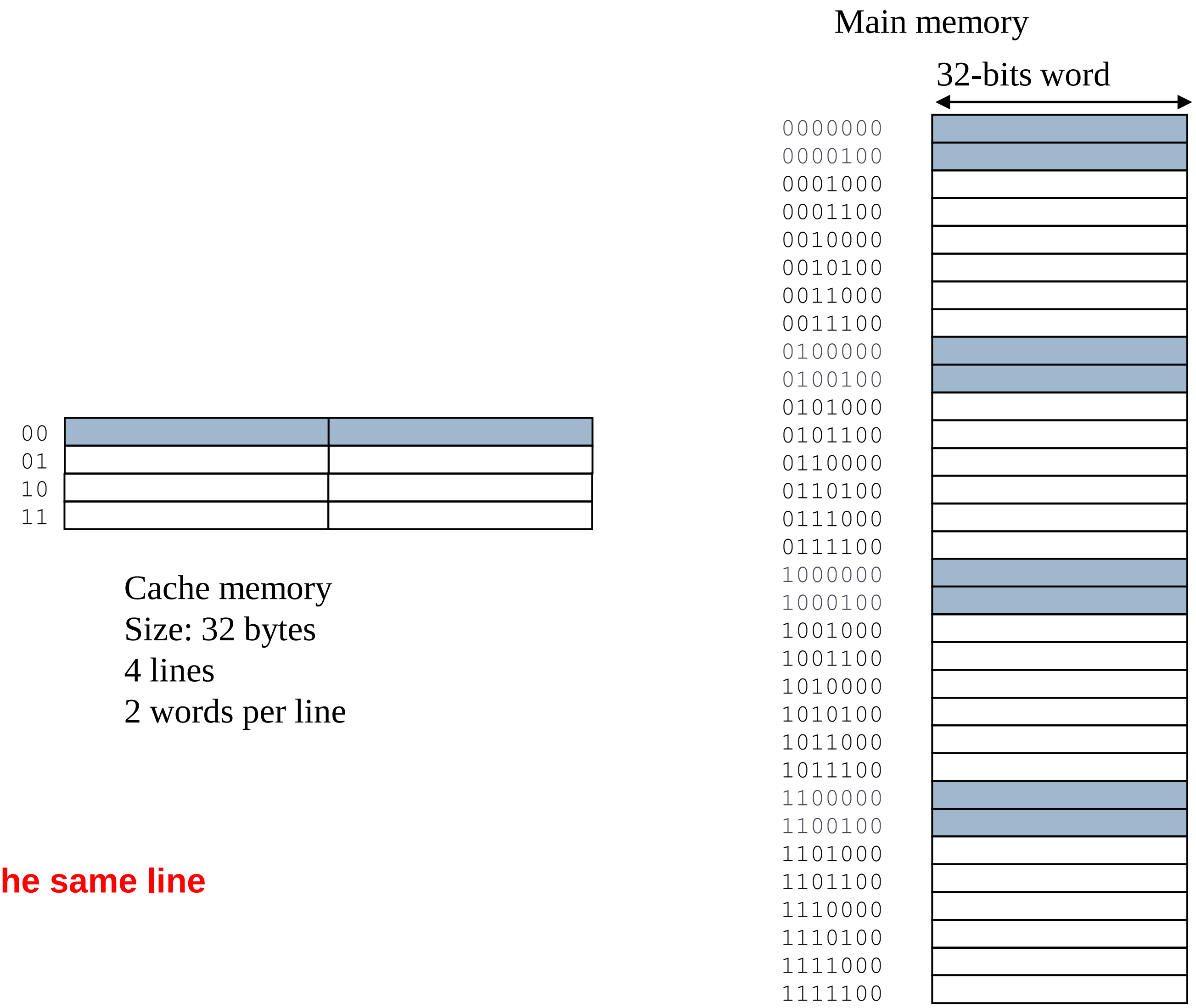
Direct mapped



- In general:
  - In a cache memory with NL number of lines, the memory block K is stored in the line:

K mod NL

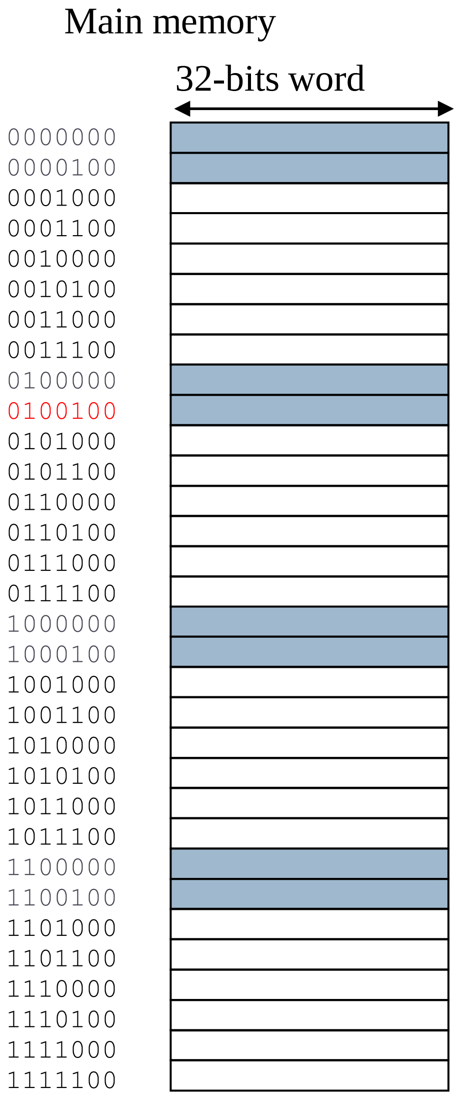
Direct mapped

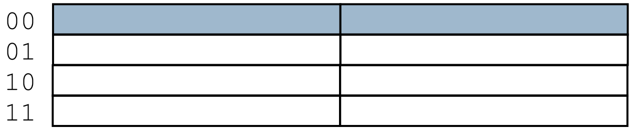


Cache memory
Size: 32 bytes
4 lines
2 words per line

Several blocks in the same line

Direct mapped

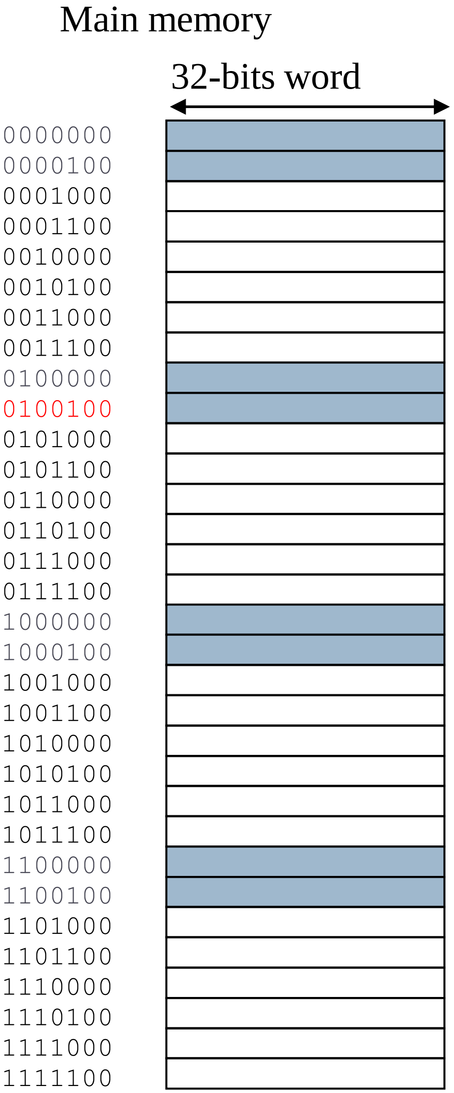


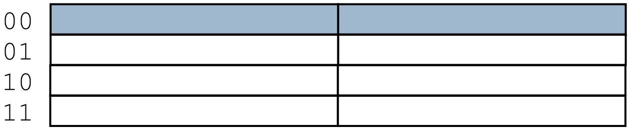


Cache memory
Size: 32 bytes
4 lines
2 words per line

How do we know which memory block is stored in a line?
Example: the address 0100100

Direct mapped





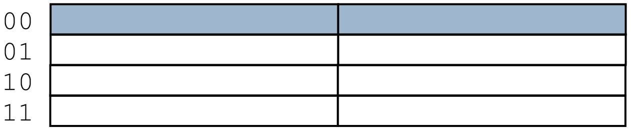
Cache memory
Size: 32 bytes
4 lines
2 words per line

How do we know which memory block is stored in a line?
Example: the address 0100100
A label is added to each line

Direct mapped
Idea

which byte inside the line?

8-byte lines

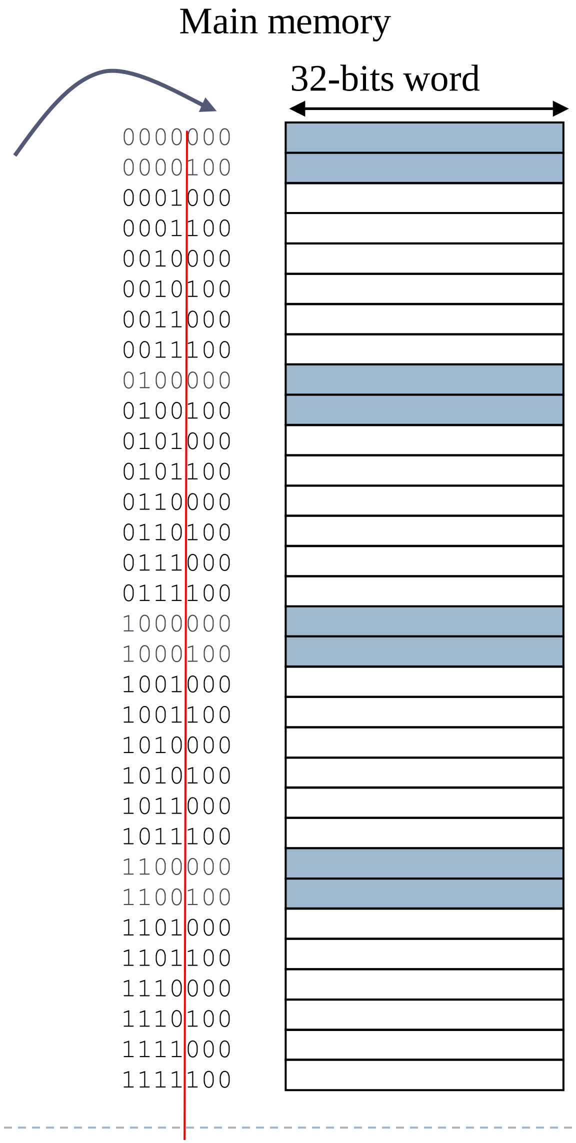


Cache memory

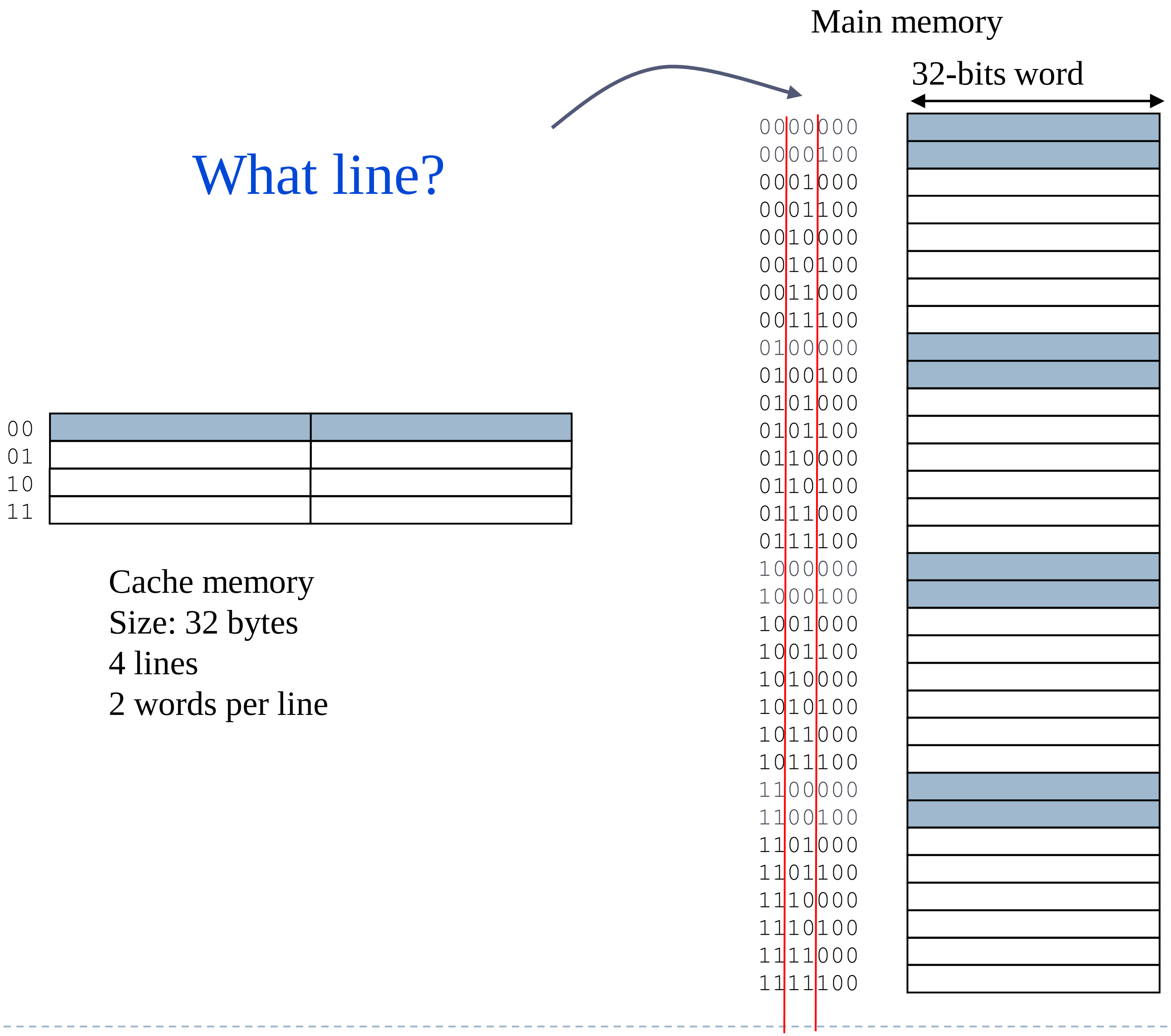
Size: 32 bytes

4 lines

2 words per line



Direct mapped
Idea



What line?

Main memory

32-bits word

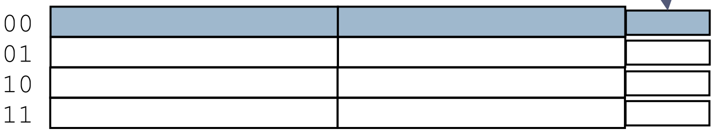
0000000
0000100
0001000
0001100
0010000
0010100
0011000
0011100
0100000
0100100
0101000
0101100
0110000
0110100
0111000
0111100
1000000
1000100
1001000
1001100
1010000
1010100
1011000
1011100
1100000
1100100
1101000
1101100
1110000
1110100
1111000
1111100

00
01
10
11

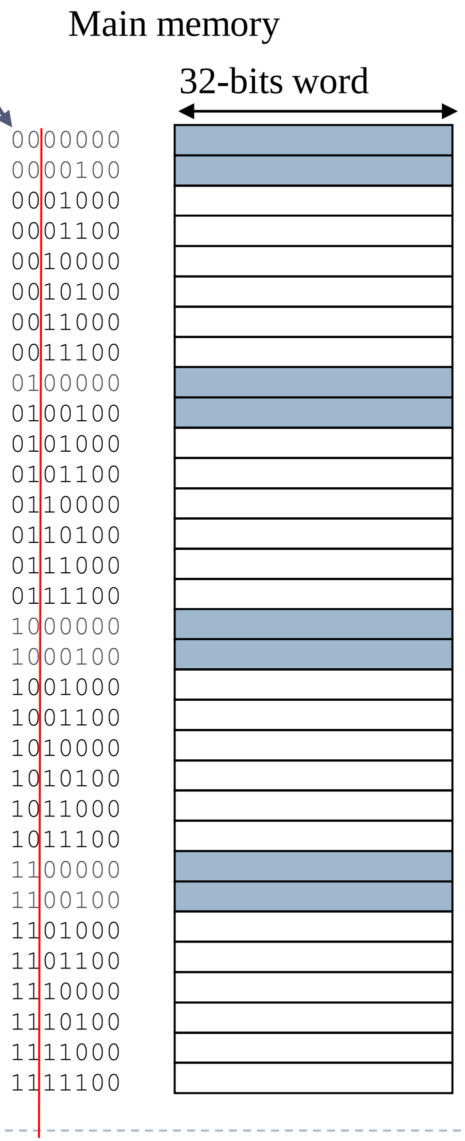
Cache memory
Size: 32 bytes
4 lines
2 words per line

Direct mapped
Idea

label associated to the line that differentiates the blocks that go to the same line

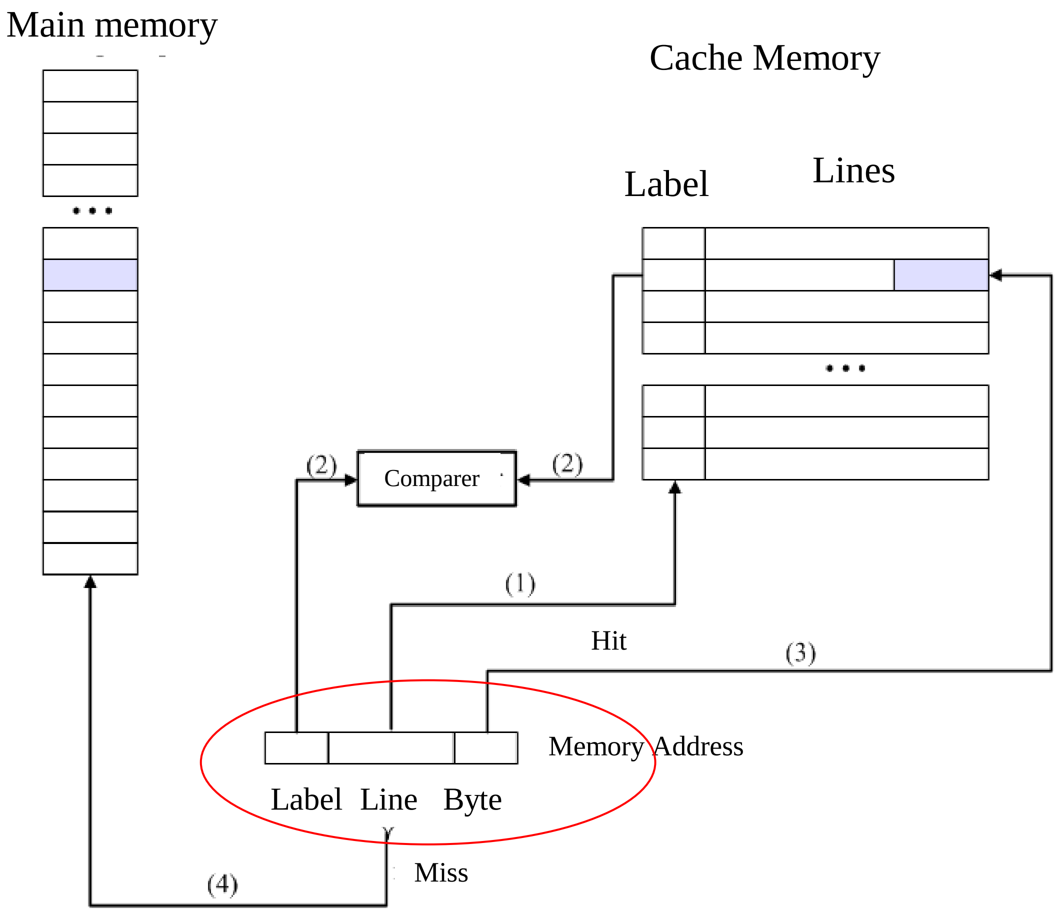


Cache memory
Size: 32 bytes
4 lines
2 words per line



The tag (upper bits of the address) is also stored in the cache memory.

Scheme for a direct mapped cache memory



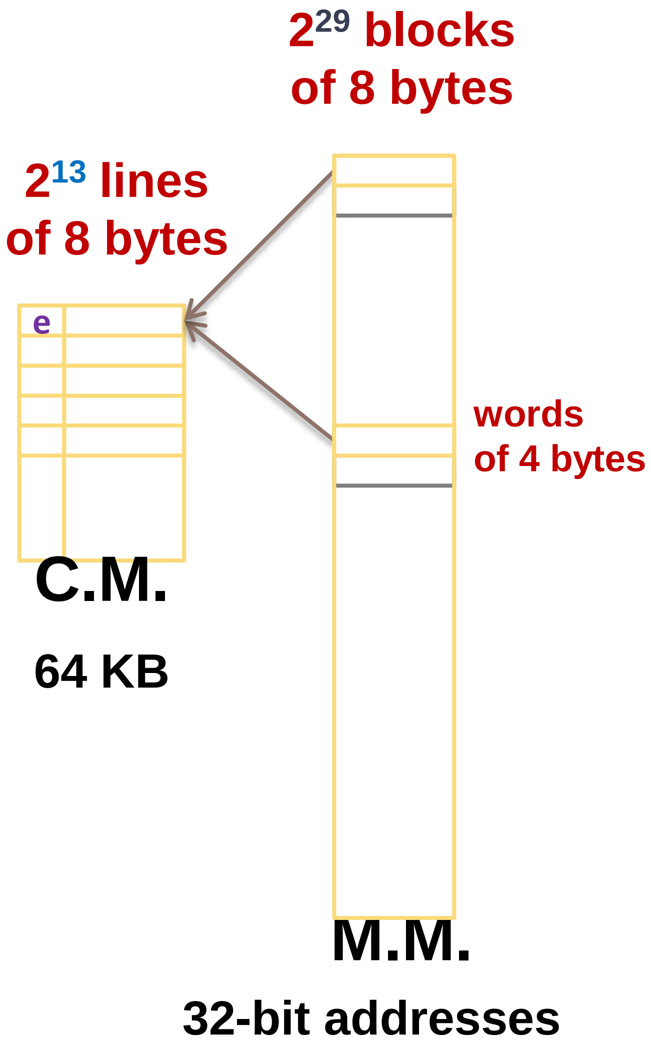
Direct mapping function

Example

- Each block of main memory is associated with a single cache line (always the same)
- Main memory address is interpreted as :

32-16	13	3
label	line	byte

- If in '*line*' there is '*label*', then block is cache (HIT)
- Simple, inexpensive, but can cause many misses depending on access pattern

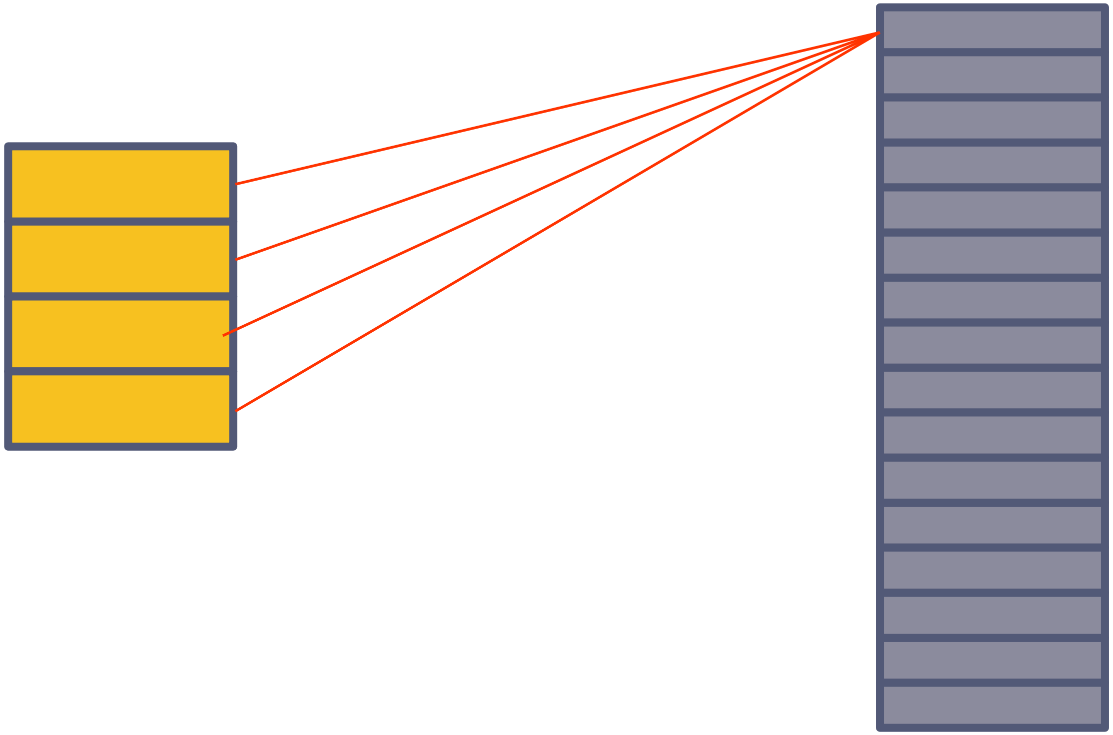


Exercise

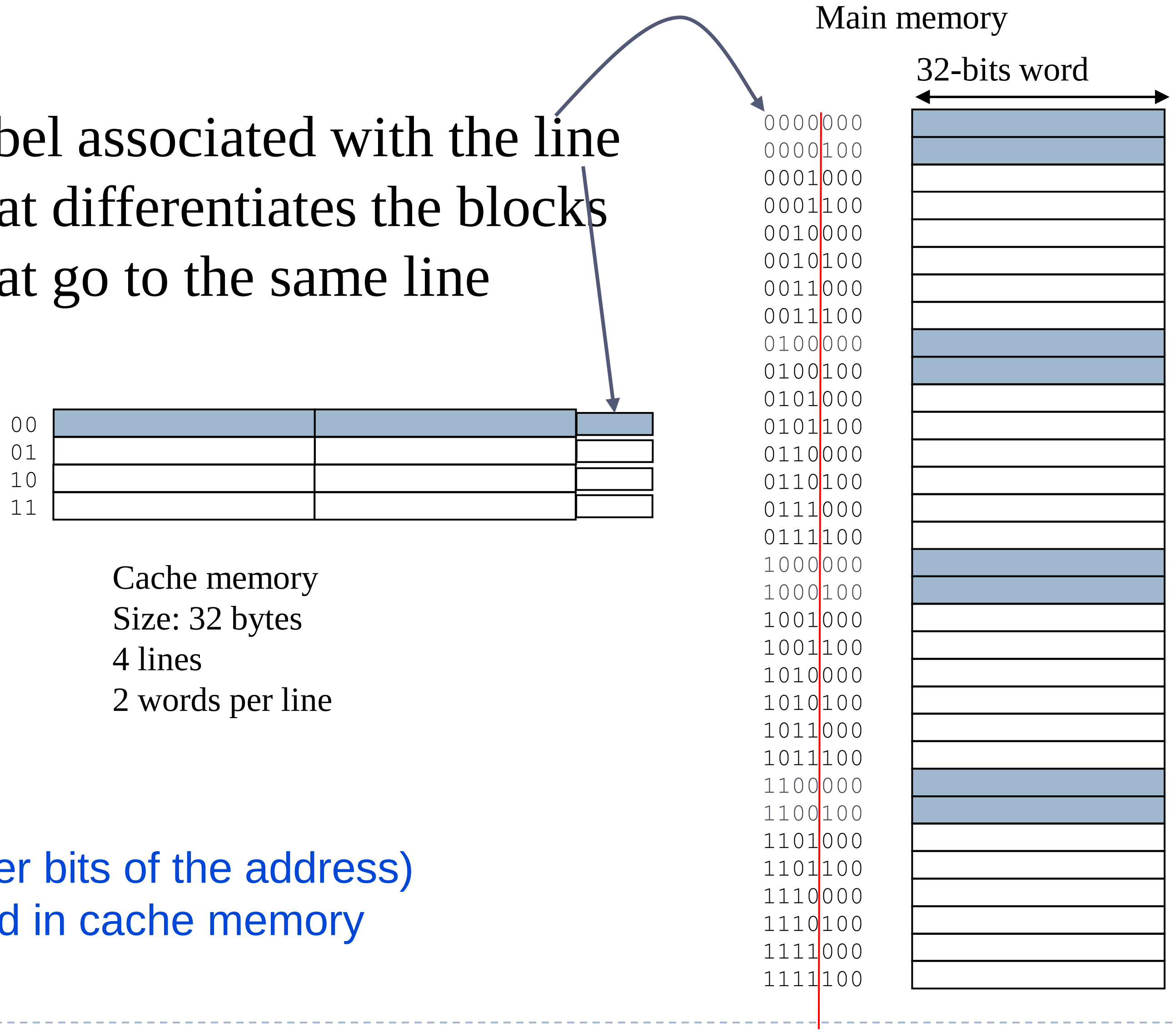
- Given a 32-bit computer with a 64 KB cache memory and 32-byte blocks. If direct matching is used:
  - On which line of the cache memory is the word for address 0x0000408A stored?
  - How can it be fetched quickly?
  - On which line of the cache memory is the word for address 0x1000408A stored?
  - How does the cache know if the word stored on that line corresponds to the word at address 0x0000408A or to the word at address 0x1000408A?

Associative mapping

- Each block of main memory can be stored in any cache line.



Associative mapping
Idea

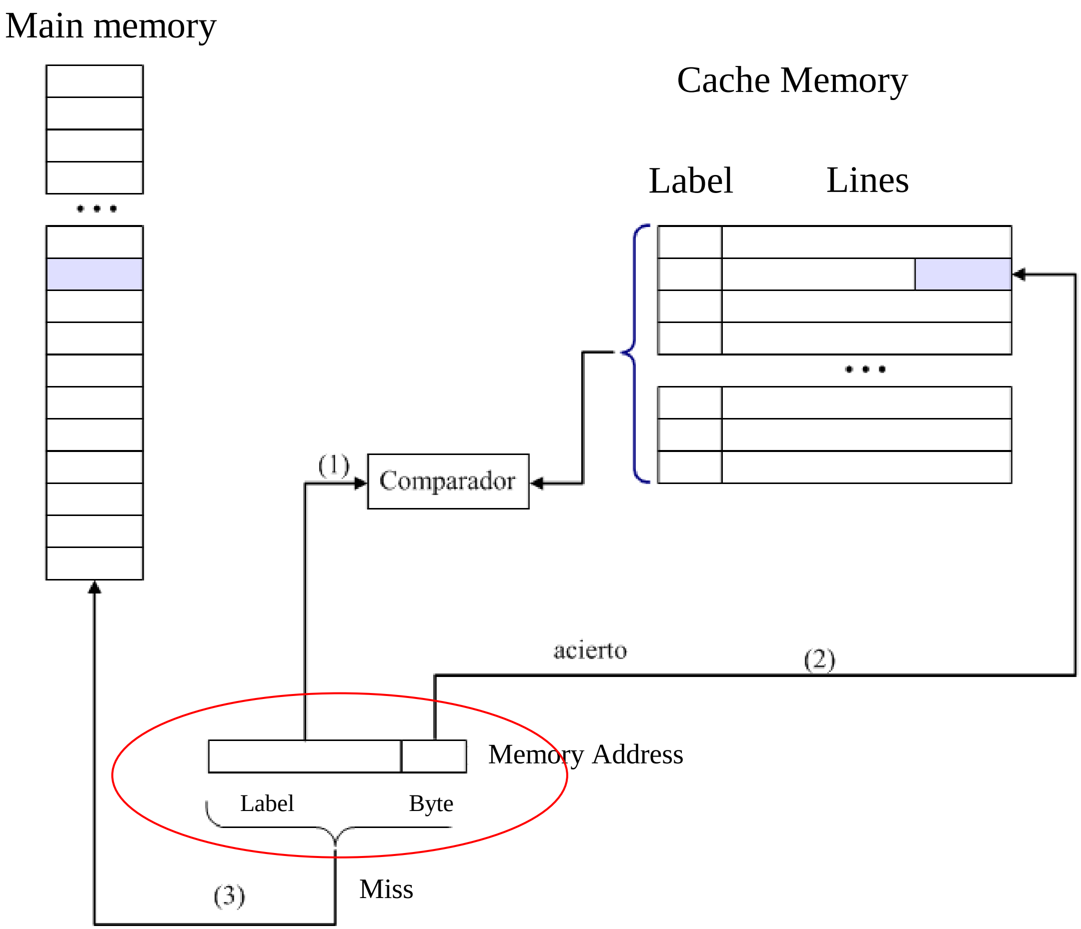


label associated with the line that differentiates the blocks that go to the same line

Cache memory
Size: 32 bytes
4 lines
2 words per line

The label (upper bits of the address) is also stored in cache memory

Organization of a cache memory with associative mapping



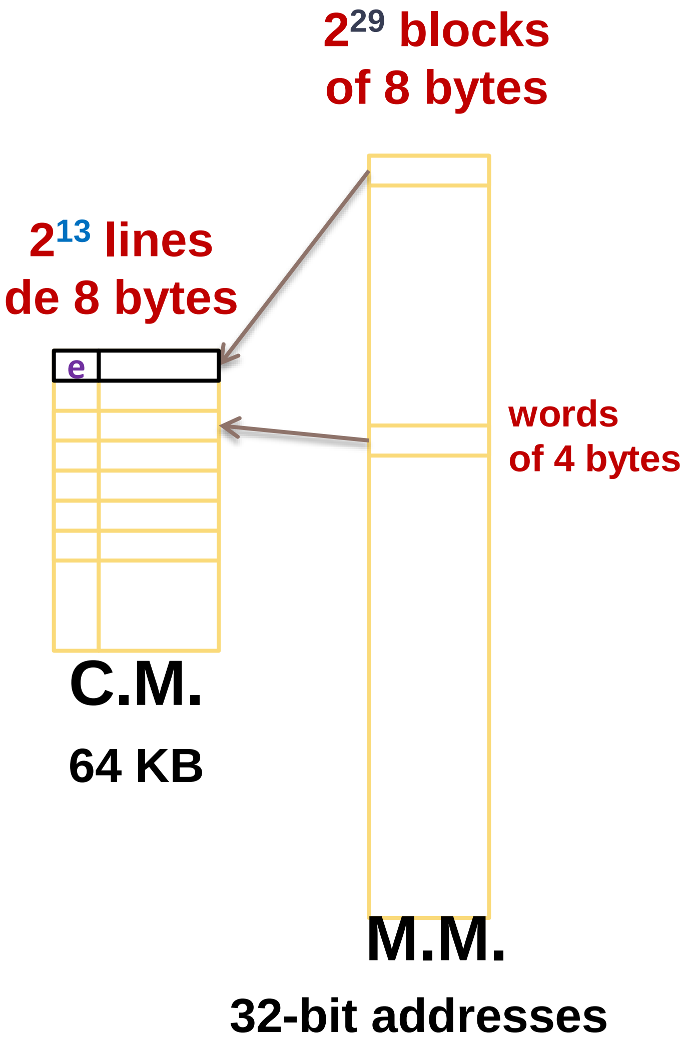
Associative mapping function

Example

- A main memory block can be placed in any cache line
- Main memory address is interpreted as:

32-3	3
label	byte

- If there is a line with '*label*' within cache, then block is there
- Access pattern independent, expensive search
- Larger tags: larger caches



2^{29} blocks of 8 bytes

2^{13} lines de 8 bytes

words of 4 bytes

C.M.

64 KB

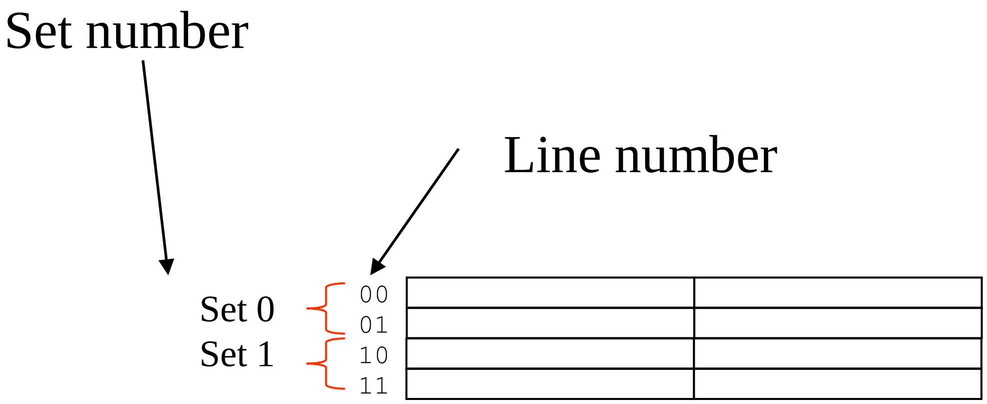
M.M.

32-bit addresses

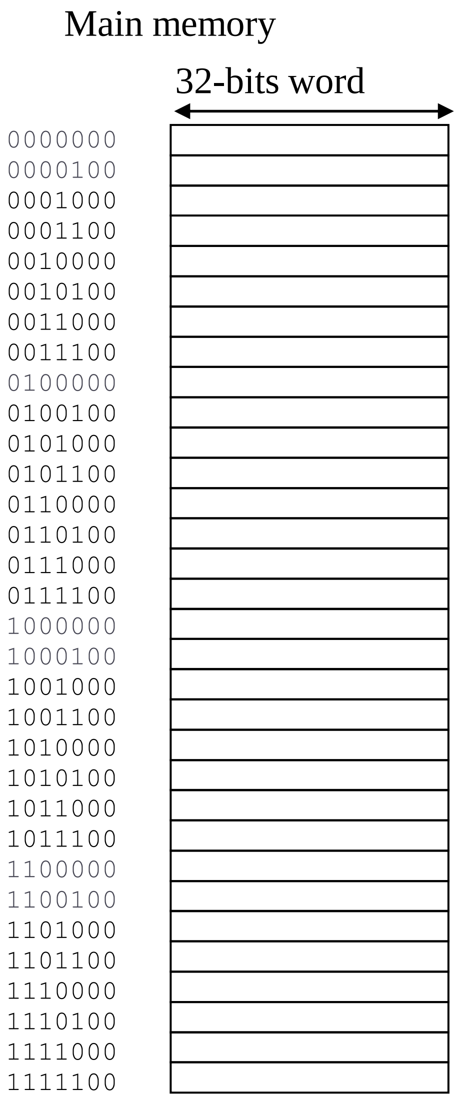
Set associative mapping

- Memory is organized into sets of lines
  - The lines in a set are referred to as ways.
- One set-associative memory cache of k-ways:
  - Each set stores K lines
- Each block is always stored in the same set…
  - Block B is stored in set:
    - B **mod** <number of sets>
- …Within a set the block can be stored in any of the lines of that set.

Set associative mapping

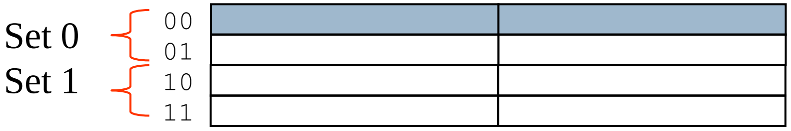


Cache memory
Size: 32 bytes
Set associative of 2 ways
2 lines per set
2 words per line

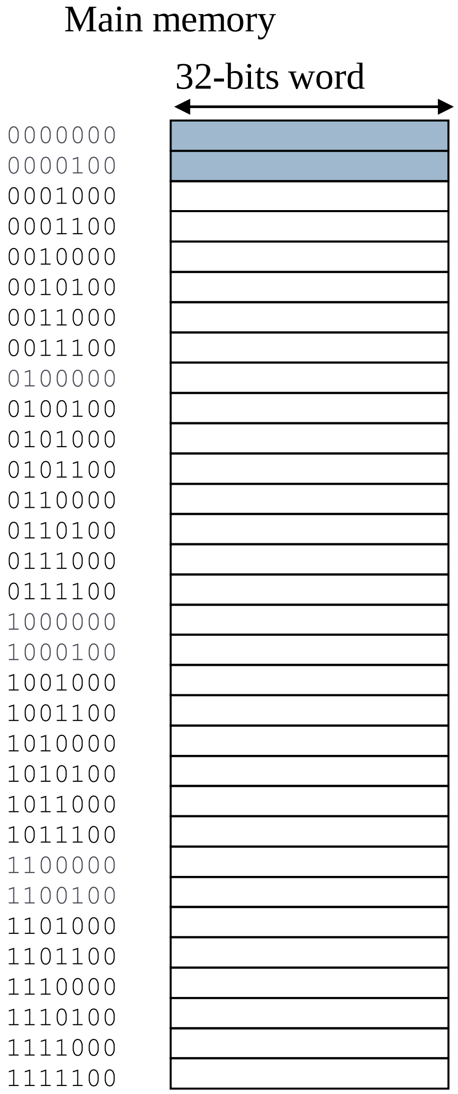


Set associative mapping

Block 0 - Set 0

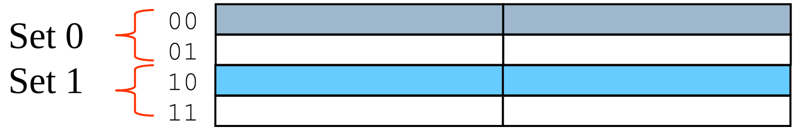


Cache memory
Size: 32 bytes
Set associative of 2 ways
2 lines per set
2 words per line

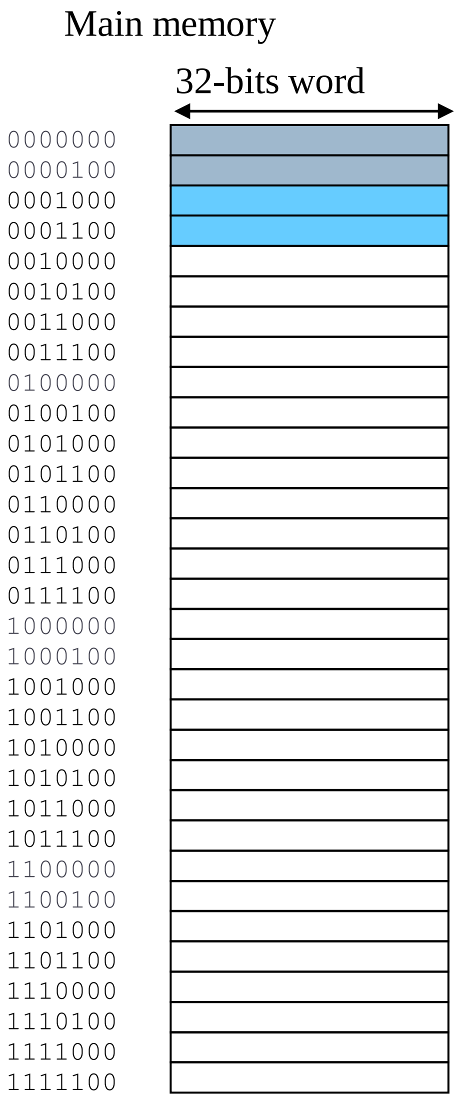


Set associative mapping

Block 1 - set 1

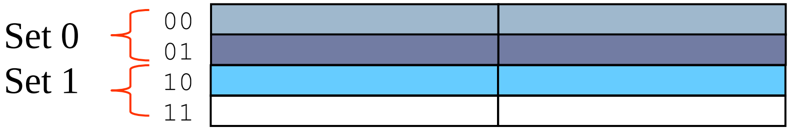


Cache memory
Size: 32 bytes
Set associative of 2 ways
2 lines per set
2 words per line

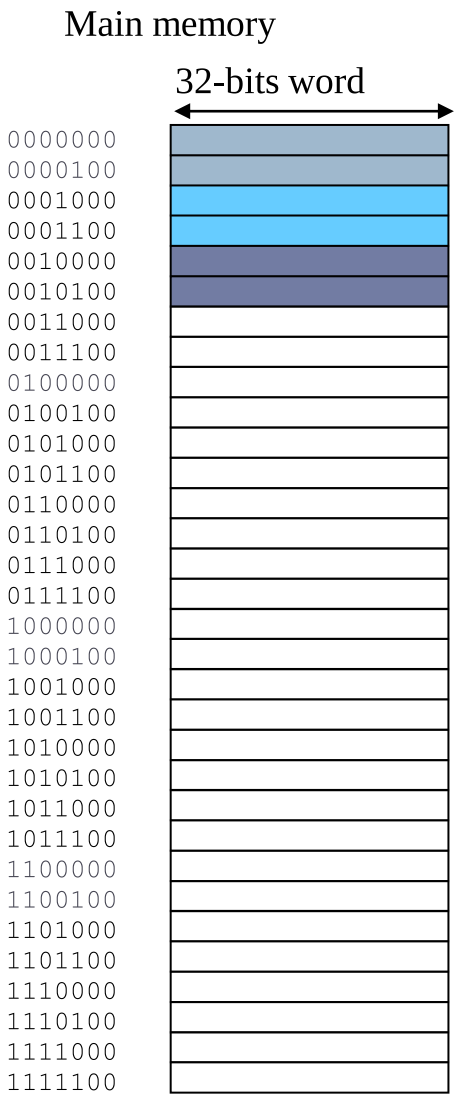


Set associative mapping

Block 2 - set 0

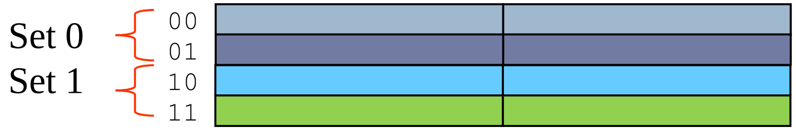


Cache memory
Size: 32 bytes
Set associative of 2 ways
2 lines per set
2 words per line

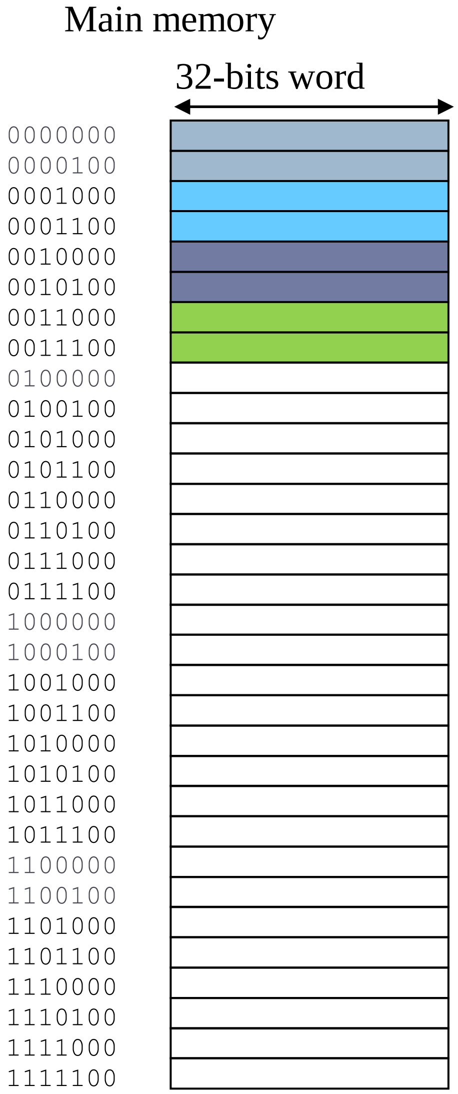


Set associative mapping

Block 3 - set 1

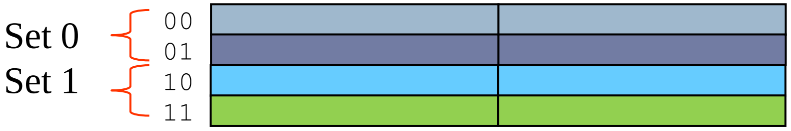


Cache memory
Size: 32 bytes
Set associative of 2 ways
2 lines per set
2 words per line



Set associative mapping

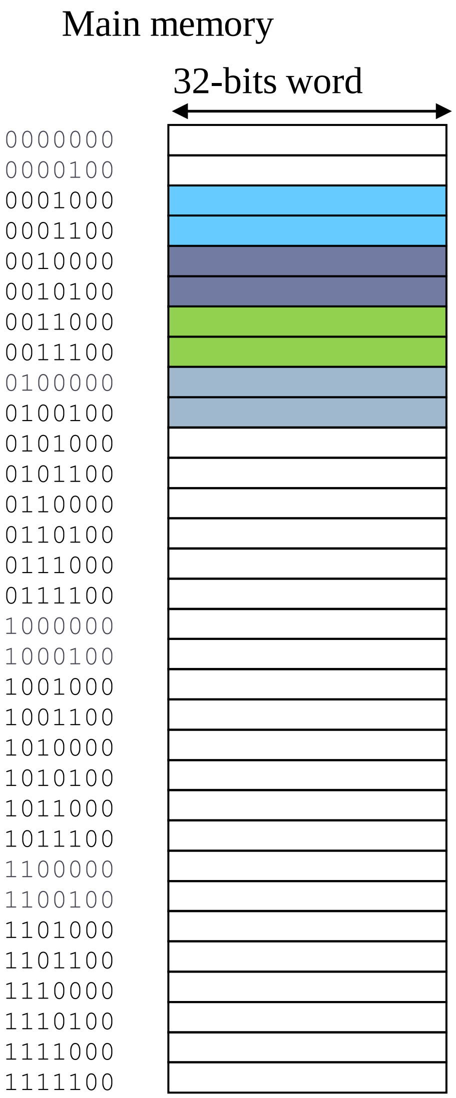
Block 4 - set 0



Set 0
Set 1
00
01
10
11

Cache memory
Size: 32 bytes
Set associative of 2 ways
2 lines per set
2 words per line

We have to discard the line stored before

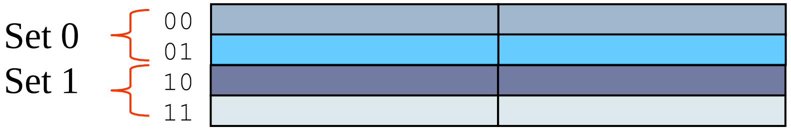


Main memory
32-bits word

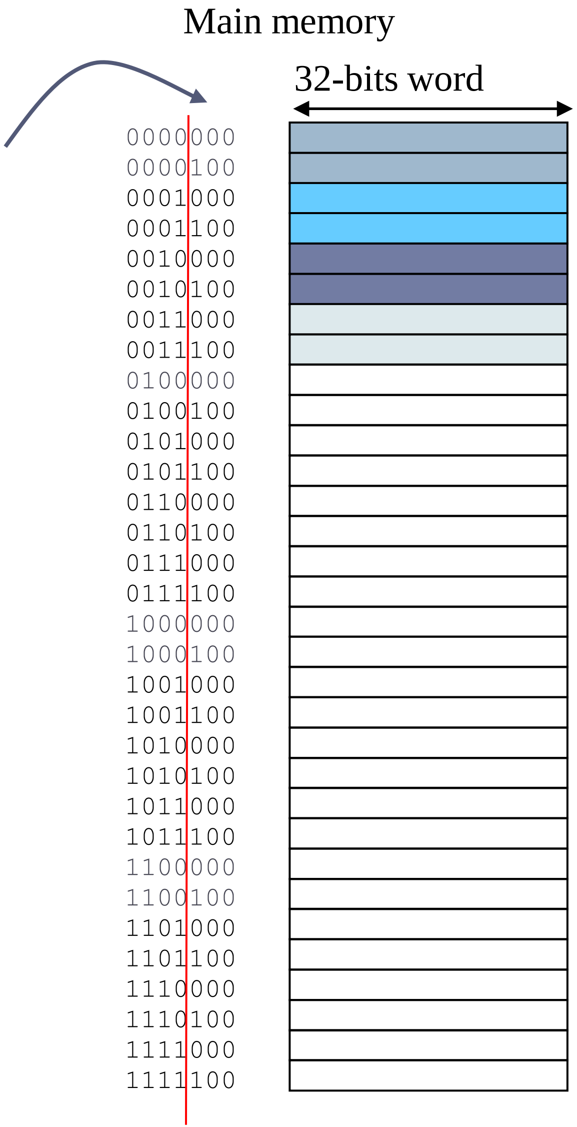
0000000
0000100
0001000
0001100
0010000
0010100
0011000
0011100
0100000
0100100
0101000
0101100
0110000
0110100
0111000
0111100
1000000
1000100
1001000
1001100
1010000
1010100
1011000
1011100
1100000
1100100
1101000
1101100
1110000
1110100
1111000
1111100

Set associative mapping

which byte inside the line?
8-byte lines



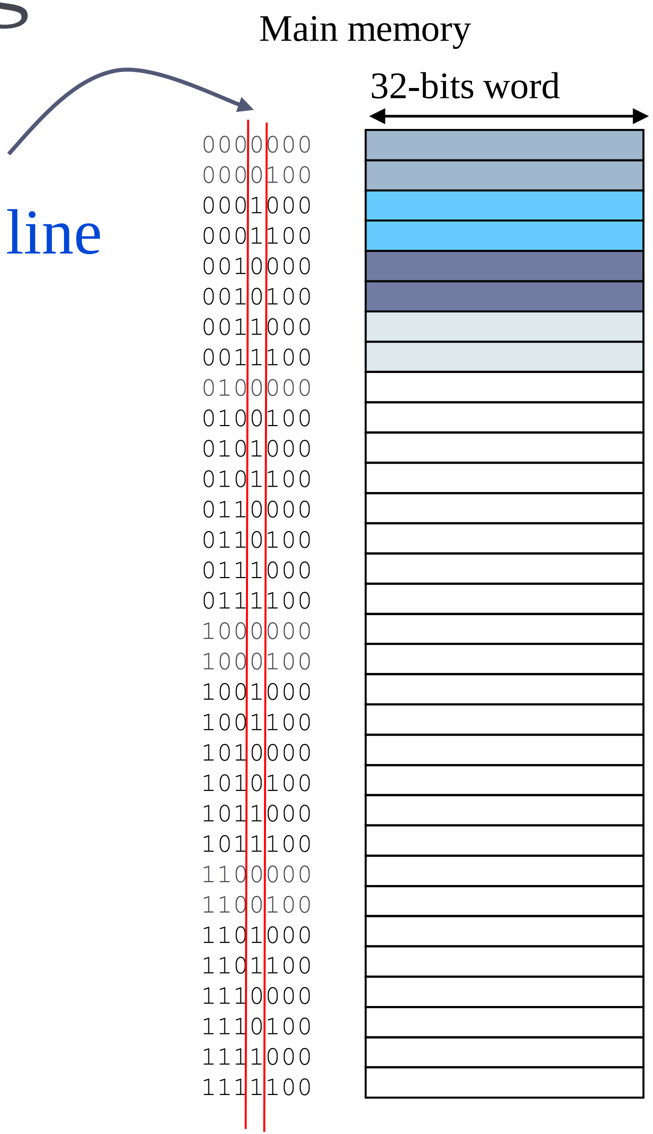
Cache memory
Size: 32 bytes
Set associative of 2 ways
2 lines per set
2 words per line

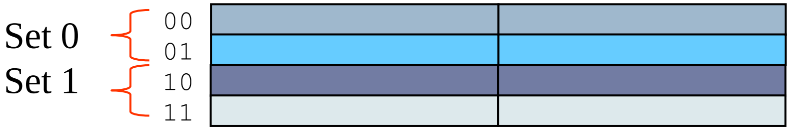


Set associative mapping

which set?

within the set to any line

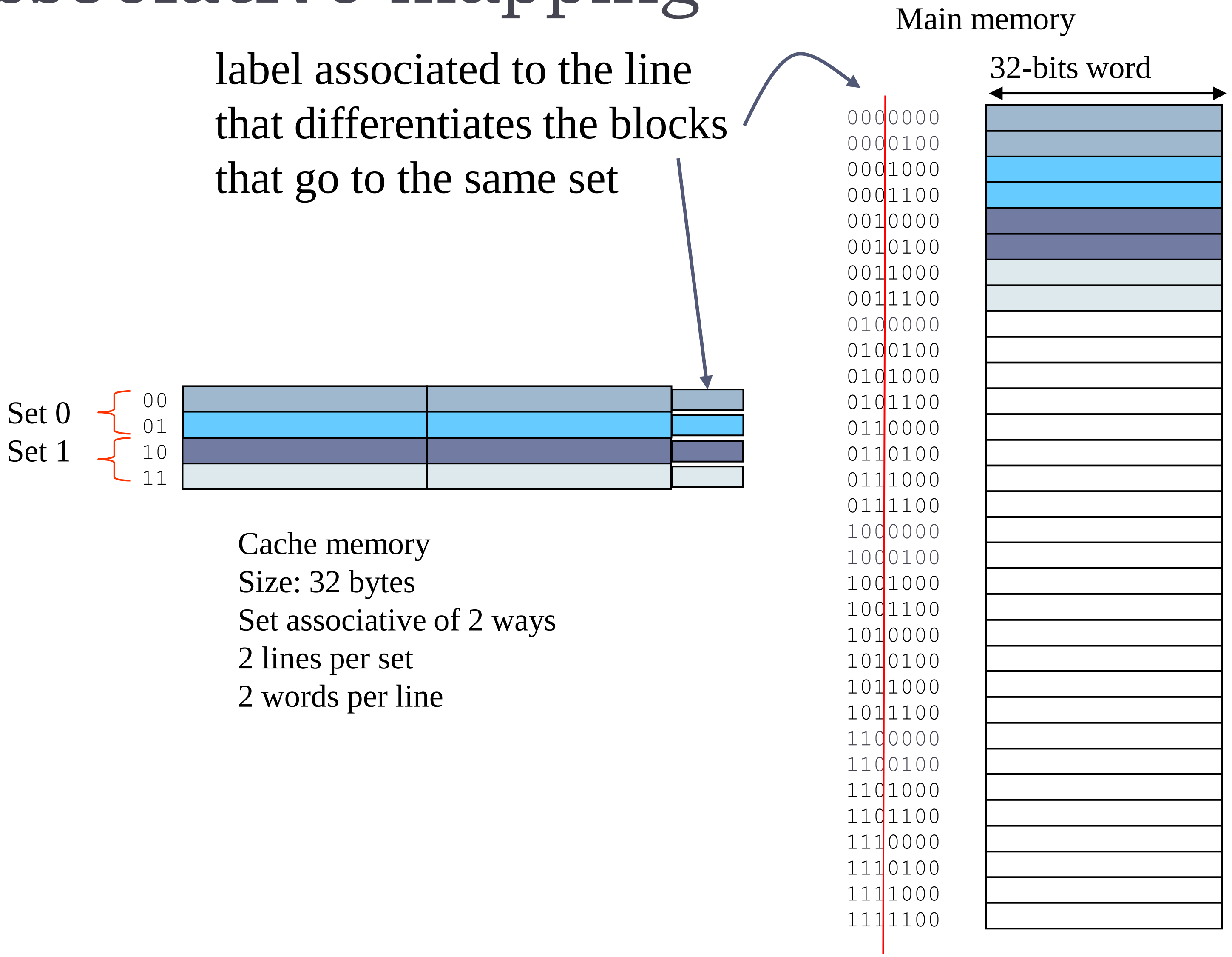




Cache memory
Size: 32 bytes
Set associative of 2 ways
2 lines per set
2 words per line

Set associative mapping

label associated to the line that differentiates the blocks that go to the same set

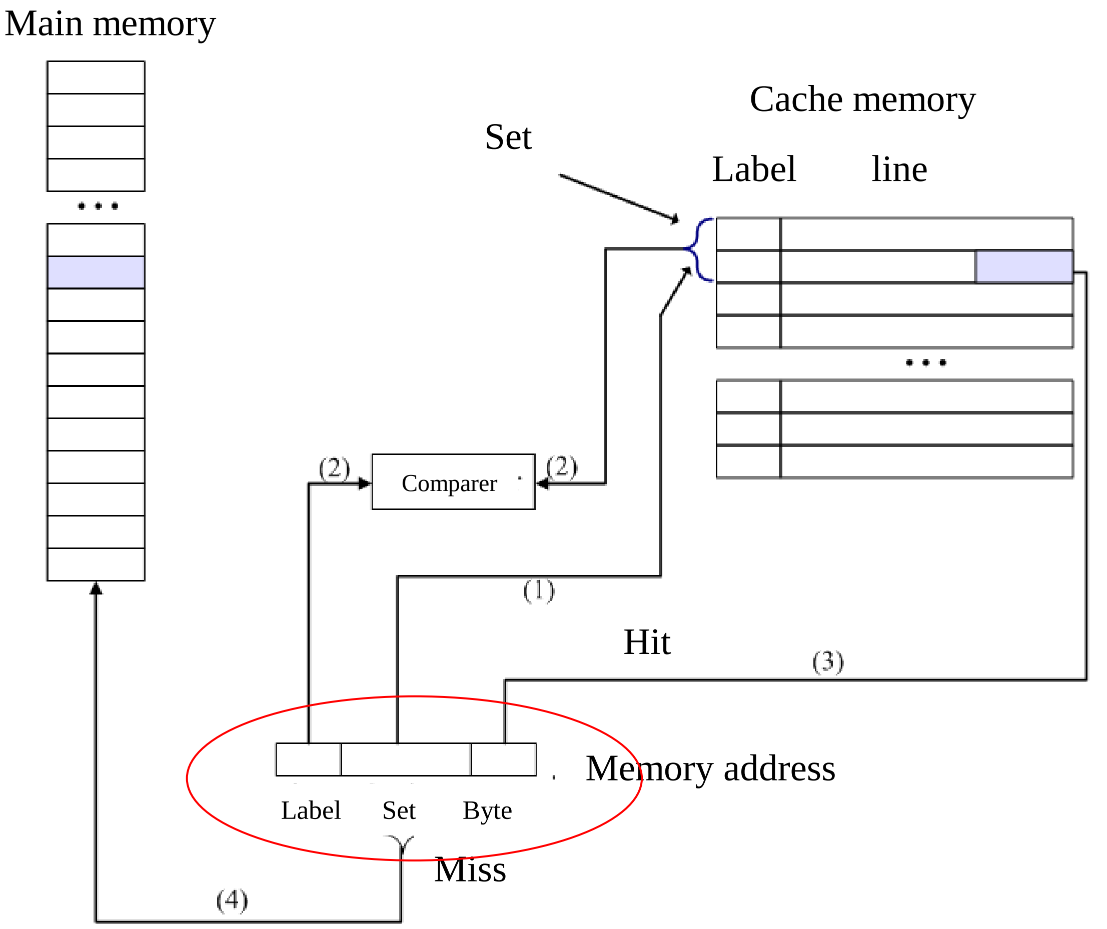


Cache memory
Size: 32 bytes
Set associative of 2 ways
2 lines per set
2 words per line

Set associative mapping

- Establishes a compromise between flexibility and cost:
  - It is more flexible than direct correspondence.
  - It is less expensive than associative matching.

Scheme for a set associative cache memory

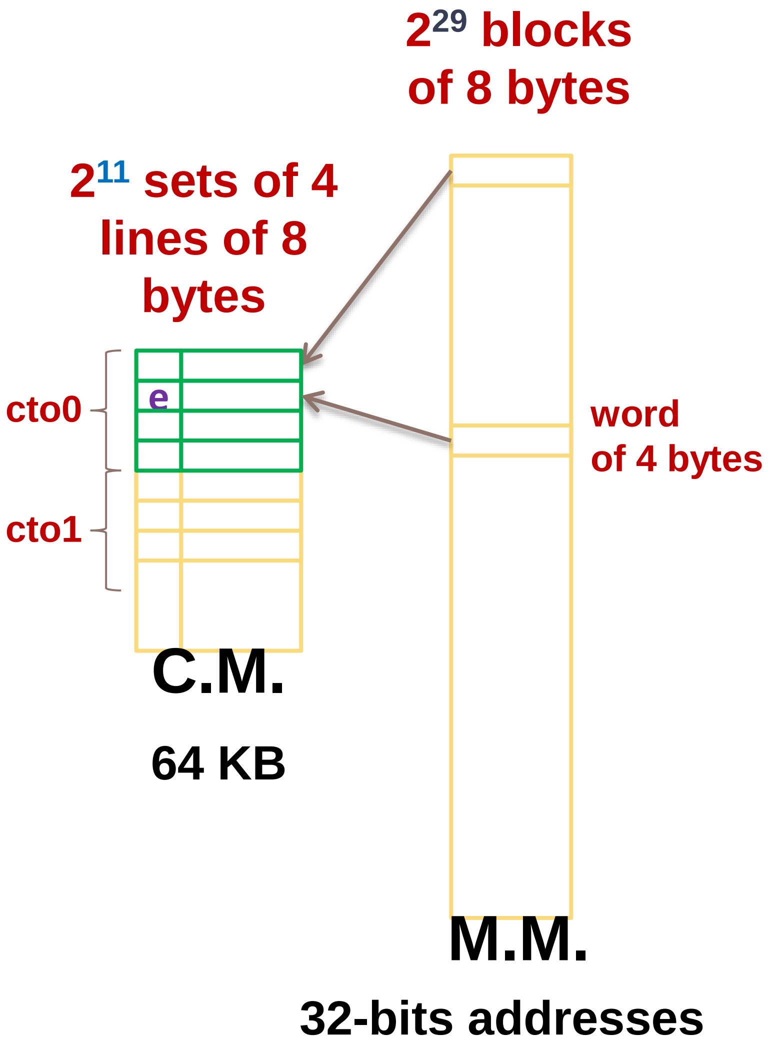


Set-associative mapping (example)

- Set-associative:
  - A block of main memory can be placed in any cache line (way) of a given set.
  - Main memory address is interpreted as:

18	11	3
label	set	word

  - If there is a line with '*label*' in the set '*set*', then there is the block in cache
  - [A] the best of direct and associative
    [D] (less) expensive search


2^{29} blocks of 8 bytes
2^{11} sets of 4 lines of 8 bytes
cto0
cto1
e
word of 4 bytes
C.M.
64 KB
M.M.
32-bits addresses

Block replacement

- When all cache entries contain blocks from main memory (MM):
  - It is necessary to select a line to be left free in order to bring a block from the MM.
    - Direct: no possible choice
    - Associative: select a line from the cache.
    - Set-associative: select a line from the selected set.
  - There are several algorithms for selecting the cache line to be released.

Replacement algorithms

- FIFO
  - *First-in-first-out*
  - Replaces the line that has been in the cache the longest.

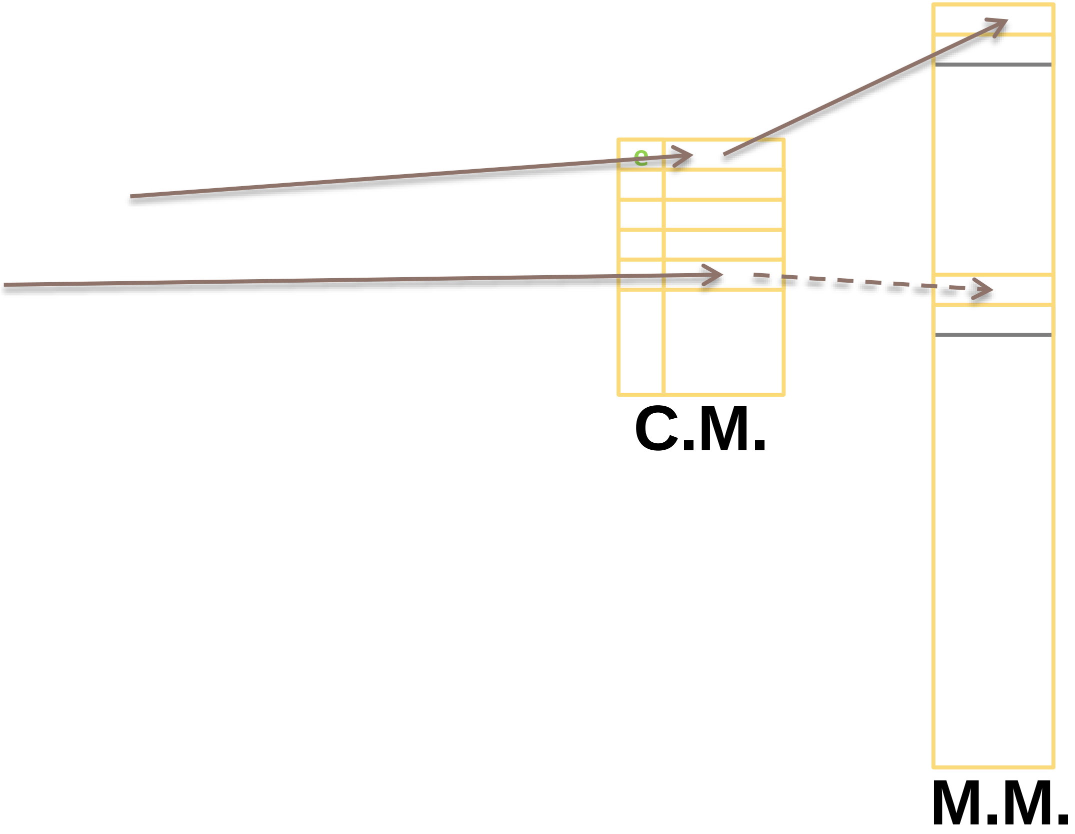
- LRU:
  - *Least Recently Used*
  - Replaces the line that has not been used for the longest time with no reference to it.

- LFU:
  - *Least Frequently Used*
  - This replaces the candidate line in the cache that has had the fewest references.

Write policy

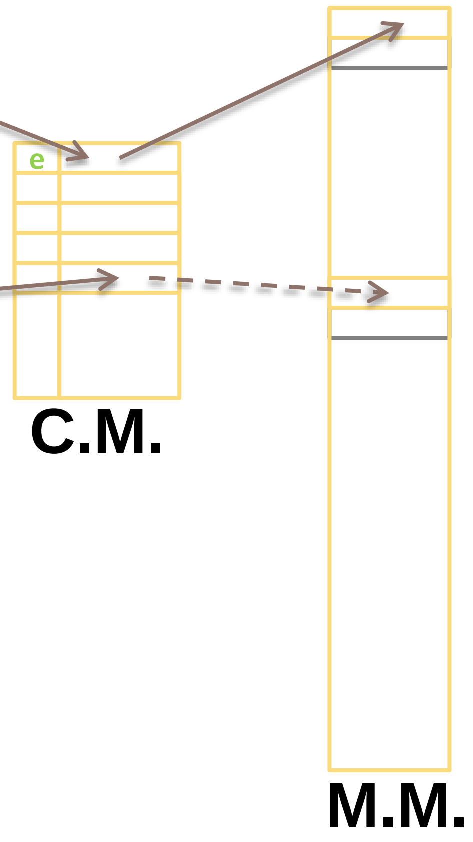
- When a data is modified in Cache memory, the Main memory must be updated at some point.

- Techniques:
  - Write through.
  - Write back.



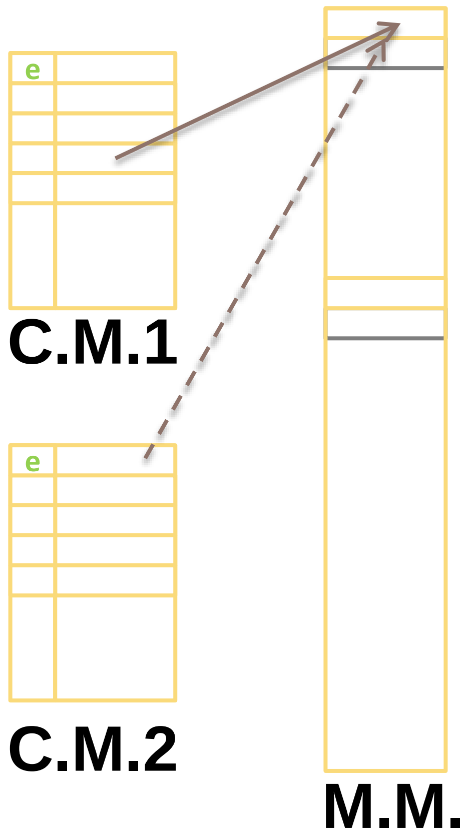
Write policy

- Write through:
  - Writing is done in both M.M. and cache.
  - [A] Coherence
  - [D] Heavy traffic
  - [D] Slow writing
- Write back:
  - The write is only done in the cache, indicating in a bit that the line is not flushed in M.M.
  - When substituting the block (or when ↓ traffic with M.M.) it is written in M.M.
  - [A] Speed
  - [D] Coherence + inconsistency

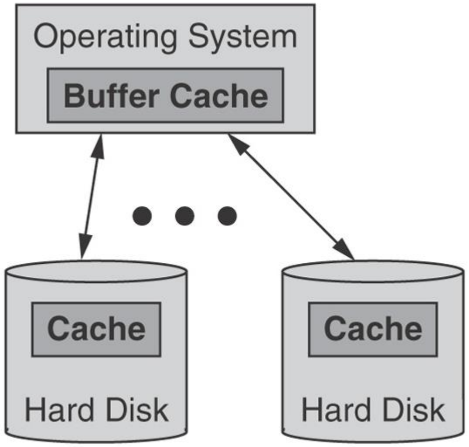


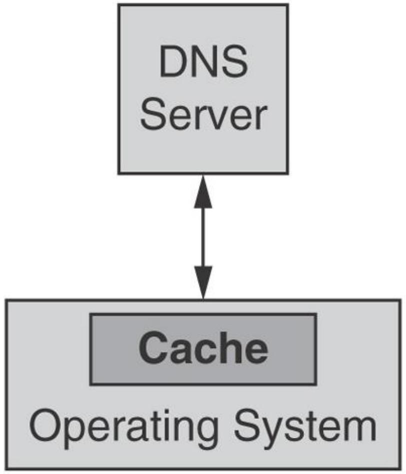
Write policy

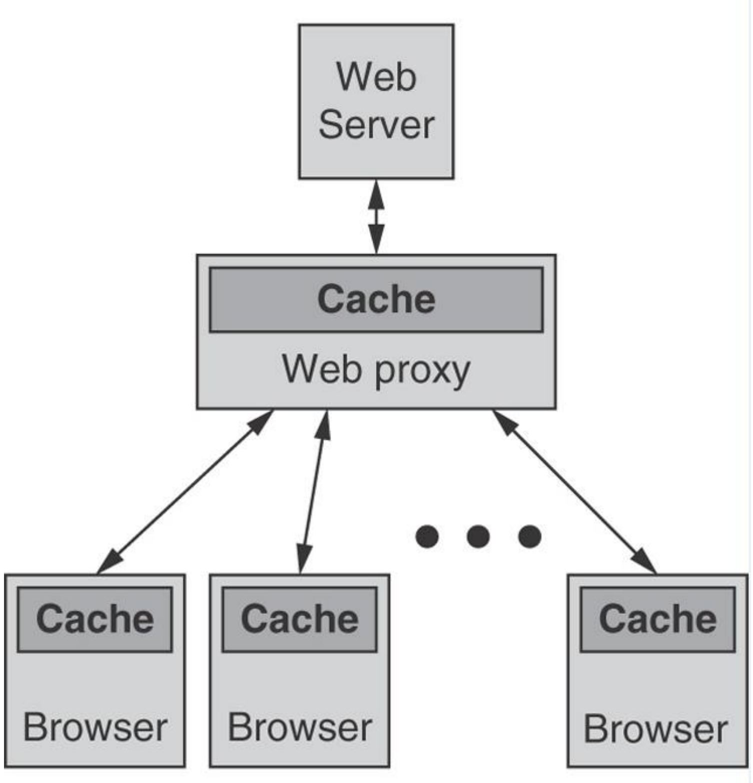
- E.g.: Multicore CPU with per-core cache
  - Cached writes are only seen by one core
  - If each core writes to the same word, what is the final result?
- E.g.: I/O module with DM.
  - Updates by DMA (direct memory access) cannot be coherent
- Percentage of references to memory for writing in the order of 15% (some studies).



Example of cache in other systems







Memory Systems
Cache, DRAM, Disk
Bruce Jacob, Spencer Ng, David Wang
Elsevier

ARCOS Group

uc3m | Universidad Carlos III de Madrid

L5: Memory hierarchy (2)

Computer Structure

Bachelor in Computer Science and Engineering
Bachelor in Applied Mathematics and Computing
Dual Bachelor in Computer Science and Engineering and Business Administration

cc BY NC SA