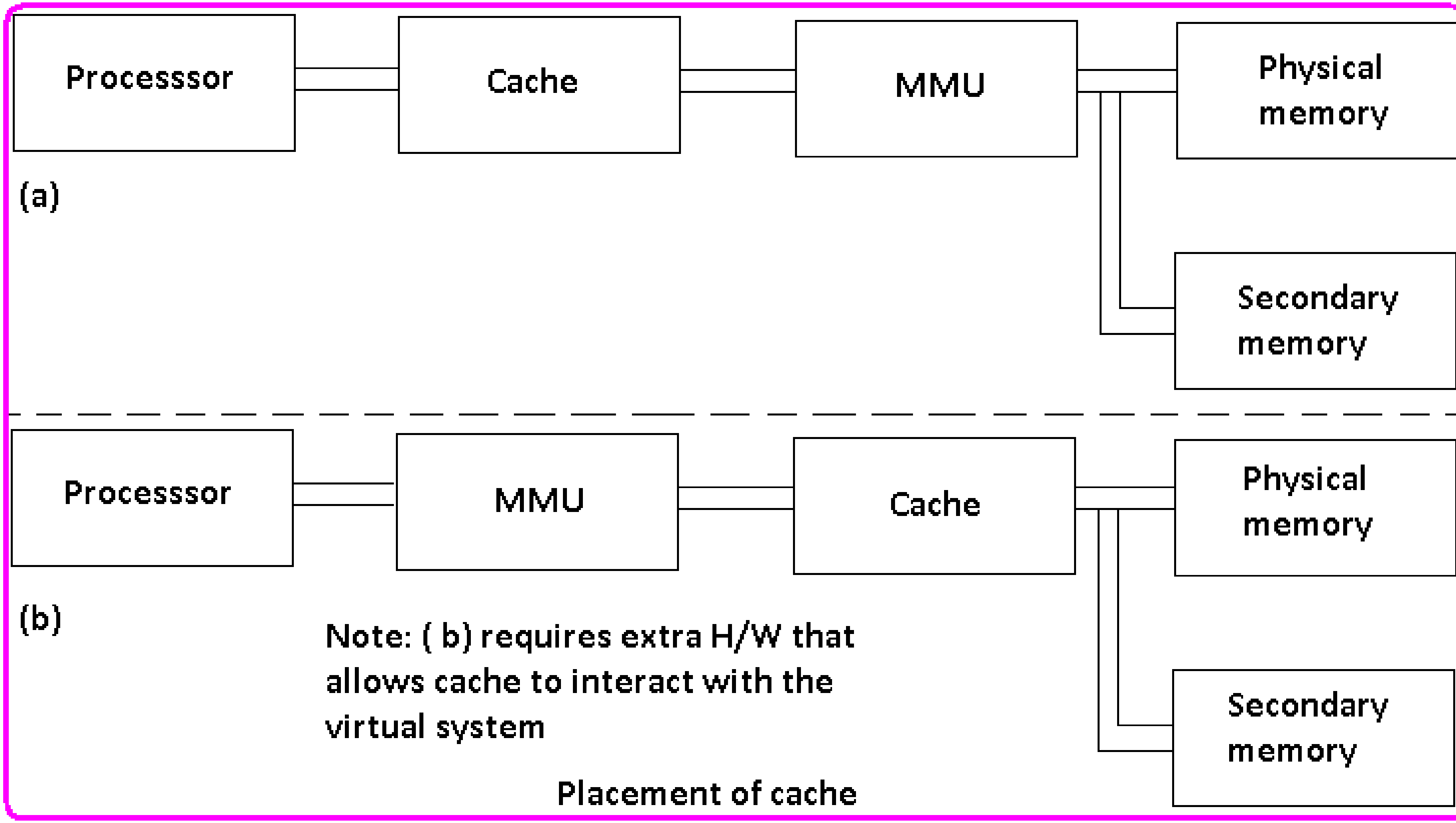
Cache Caching

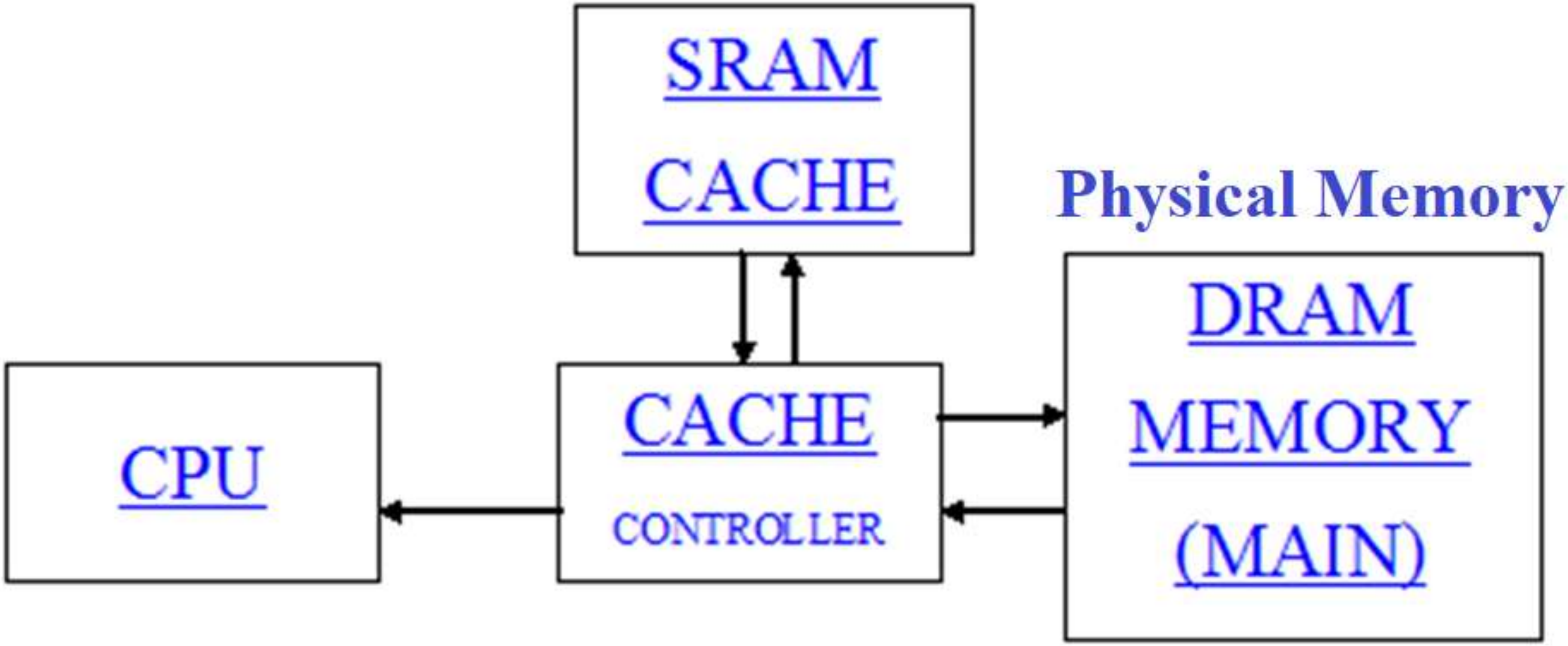
Microprocessors

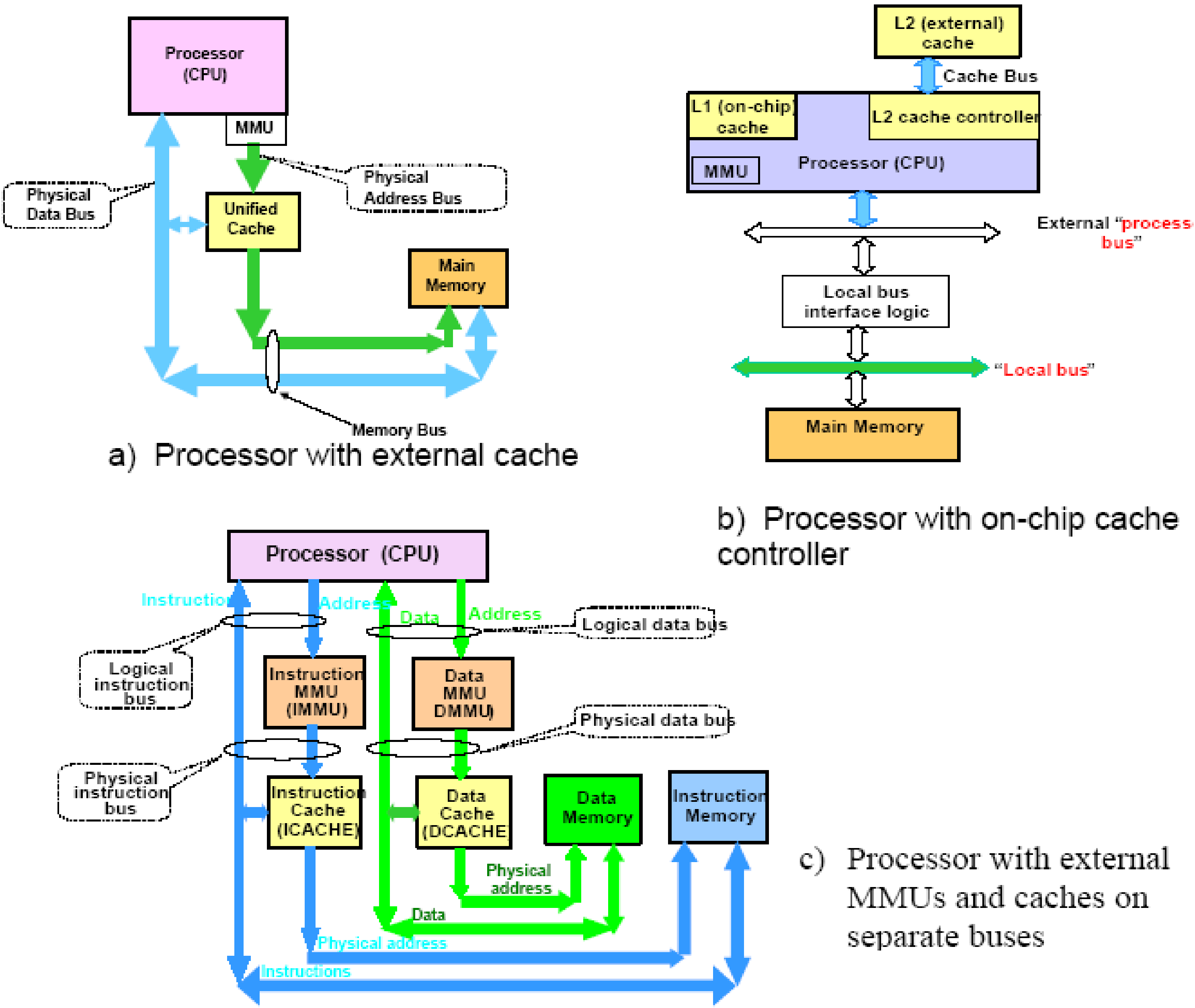
- Caching refers to an important optimization technique used to reduce Von Neumann Bottleneck (time spent performing memory access that can limit overall performance) and improve the performance of any hardware or software system that retrieves information.
- A cache acts as an intermediary.



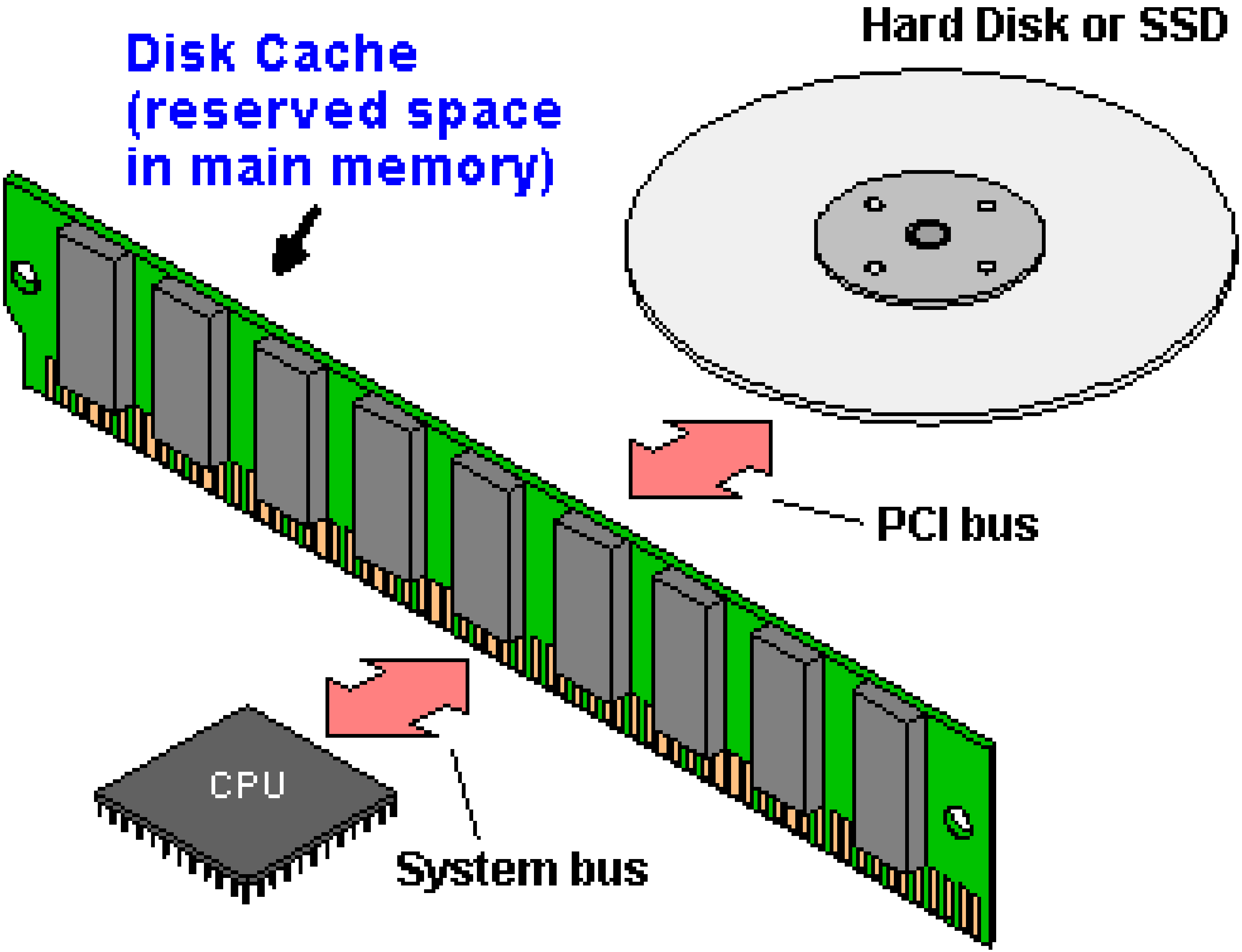
Note: (b) requires extra H/W that allows cache to interact with the virtual system

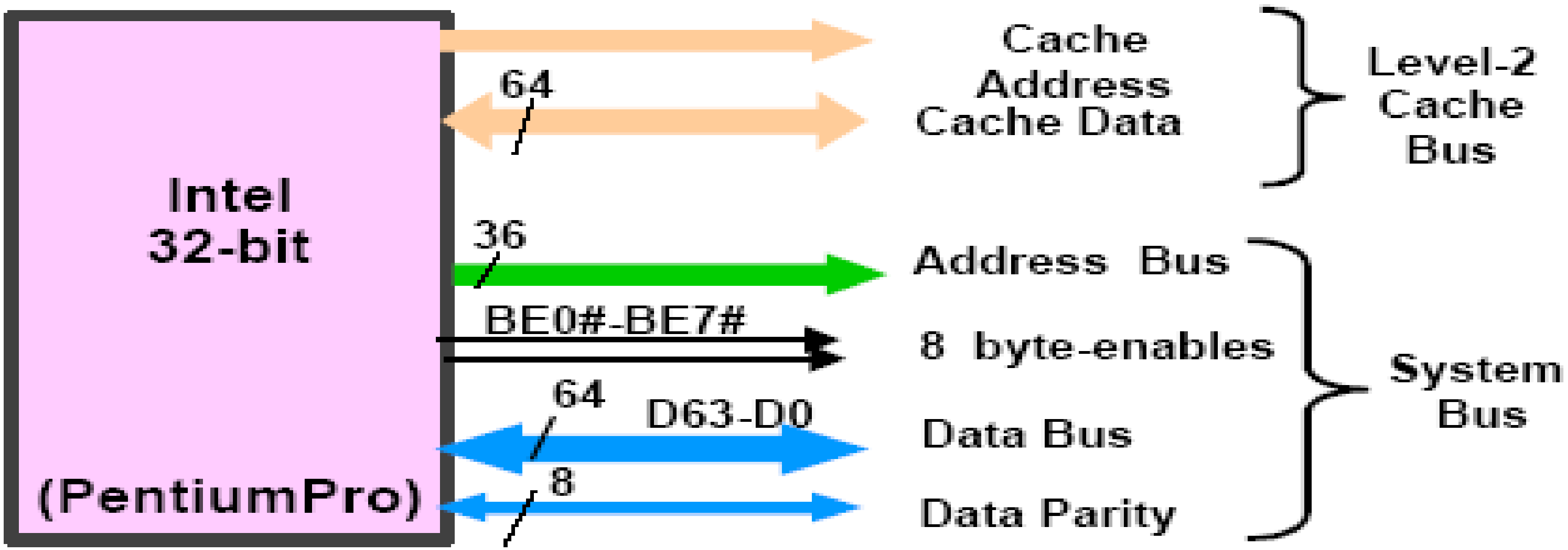
Placement of cache


SRAM
CACHE
Physical Memory
DRAM
MEMORY
(MAIN)
CPU
CACHE
CONTROLLER



Example processors with external MMUs and caches


Disk Cache
(reserved space
in main memory)
Hard Disk or SSD
PCI bus
CPU
System bus



(8 byte-enables)

- 40 MHZ – 25 ns
- DRAM chips – access time 60 – 100 ns
- SRAM – access time 15 – 25 ns
- SRAM (ECL) – access time 12 ns BUT expensive
- Assume aircraft moving at 850 km/h
- Distance moved in 12 ns=1/10 of diameter of hair
- Cache – attempts adv of quick SRAM with cheapness of DRAMs. to achieve the most effective memory system.

Characteristics of Cache

Small, active, transparent and automatic

Small Most caches are 10% of the main memory size and hold equal percentage of data

Active Has active mechanism that examines each request and decides how to respond Available or not available. If not available, to retrieve a copy of item from data store (main memory)

Decides which item to keep in cache

Transparent	A cache can be inserted without making changes to the request or data store. Interface cache presents to requester the same interface as it does to data storage and vice versa
Automatic	Cache mechanism does not retrieve instruction on how to act or which data items to store in the cache storage. Instead it implements an algorithm that examines the sequence of requests and uses the request to determine how to manage the cache.

Flexibility as in usage

- Hardware, software and combination of the two
- Small, medium and large data items
- Generic data items
- Application type of data
- Textual and non textual
- Variety of computers
- Systems designed to retrieve data(web) or those that store (physical memories)

Cache terminologies

There are many terminologies depending on application

Memory system	Backing store
Cache web pages	Browser
Server	origin server
Database lookups	Client request for database servers (system that handles requests)

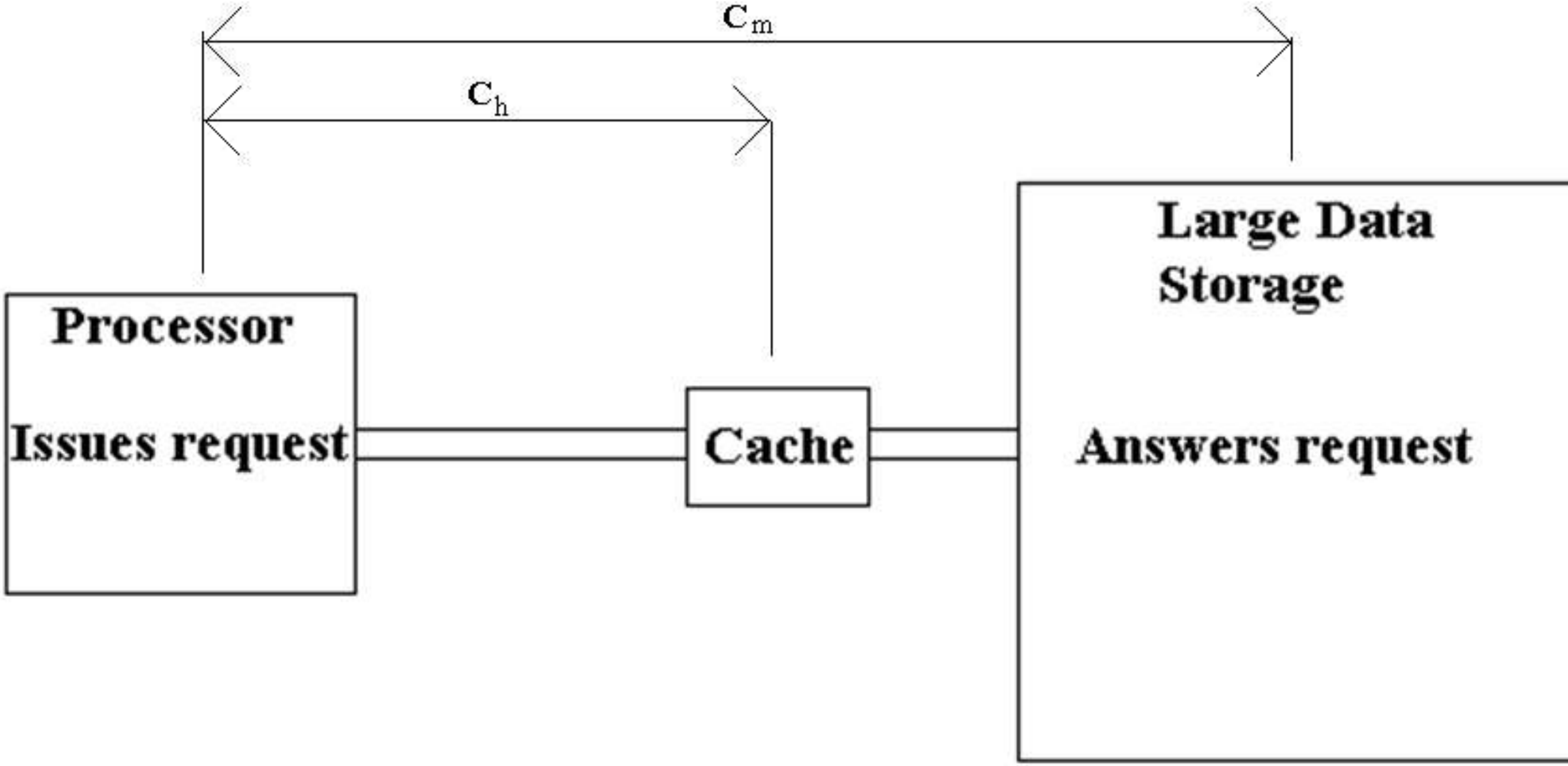
Hit Request that can be satisfied without any need to access the underlying data store

Miss Request that cannot be satisfied

High locality of reference

sequence containing repetitions of the same request

$$Hit\ Ratio\left(r\right) = \frac{number\ of\ request\ that\ are\ hits}{Total\ number\ of\ requests}$$



cache system

Miss ratio =1-hit ratio

$$Cost = rC_h + (1-r)C_m$$

where C_h and C_m are costs of accessing cache and store respectively

Multi level cache

$$Cost = r_1 C_{h1} + r_2 C_{h2} + (1 - r_1 - r_2) C_m$$

Pre-fetch related data

- If a processor accesses a byte of memory, the cache fetches 64 bytes. Thus if the processor fetches the next byte, the value will come from the cache. Modern computer systems employ multiple caches (L1, L2, L3). Caching is used with both virtual and physical memory as well as secondary memory.
- Translation Lookaside Buffer (TLB) contains digital circuits that move values into a Content Addressable memory (CAM) at high speed, i.e. Cache line in any of the ways available (set Associative).

Caches in multiprocessors

Write through and write back

Write through

This is the method of writing to memory where the cache keeps a copy and forwards the write operation to the underlying memory.

Write back (WB) scheme

Cache keeps data item locally and only writes the value to memory if necessary. This is the case if value reaches end of LRU list and must be replaced. To determine whether value is to be written back, a bit termed dirty bit is kept by cache.

Cache Coherence

Performance can be optimized by using write back scheme than write through scheme. The performance can also be optimized by giving each processor its own cache.

Conflicts occur

To avoid conflicts, all devices that access memory must follow a cache coherence protocol that coordinates the values.

Each processor must inform the other processor of its operation so that the addressing is not confused.

Physical memory cache

Demand paging as a form of cache

- Cache behaves like physical memory and data storage as external memory
- Page replacement policy as cache replacement policy

Example READ

- Cache performs two tasks, passing the request simultaneously to physical (Main) and searches locally (cache)
- If answer is local, cancel memory operation

- If no local answer, wait for underlying memory operation to complete
- Answer arrives, save copy, transfer answer to processor

Instructions and Data caches

- **Should all memory references pass through a single cache?** To understand the question, imagine instructions being executed and data being accessed.
- Instruction fetch tends to behave with highly locality

- Data fetch may be at random and hence not necessarily adjacent in the memory address.
- The overall performance of the cache is reduced. Architects vary in choice from different caches and one large cache that can allow intermixing (L3 for (i7) itanium processors).

Virtual memory caching and cache flush

- When the OS is running a program, the addresses given are always the same, ie starting from zero.
- If the OS changes the program, it must also change that information in the cache

Multiple caches

- The cache must have a way to resolve these multiple application address location

1. Cache flush operation

   The cache is flushed whenever the OS changes to a new virtual space.

2. Disambiguation

- ○ Use extra bits that identify the address space
- ○ Processor contains an extra hardware register that contain an address space **ID**
- ○ Each program is allocated a unique number

- Any application swap, the OS loads the application **ID** into the address space **ID** register.
- Processor creates artificially longer addresses before passing an address to the cache containing the **ID**

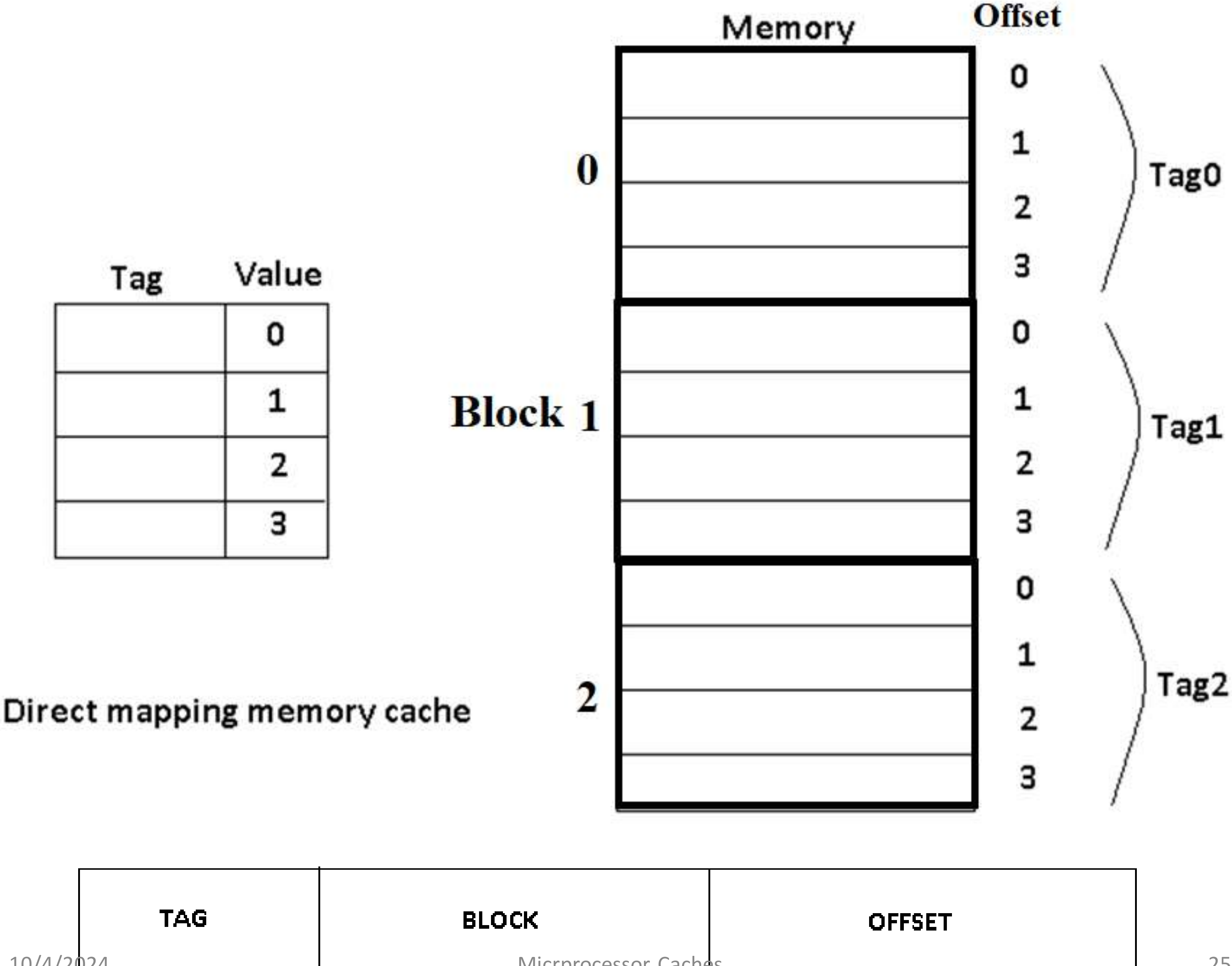
Implementation of memory cache

Originally the memory cache contained two values of entry: memory address and the content found in that address. New methods are:

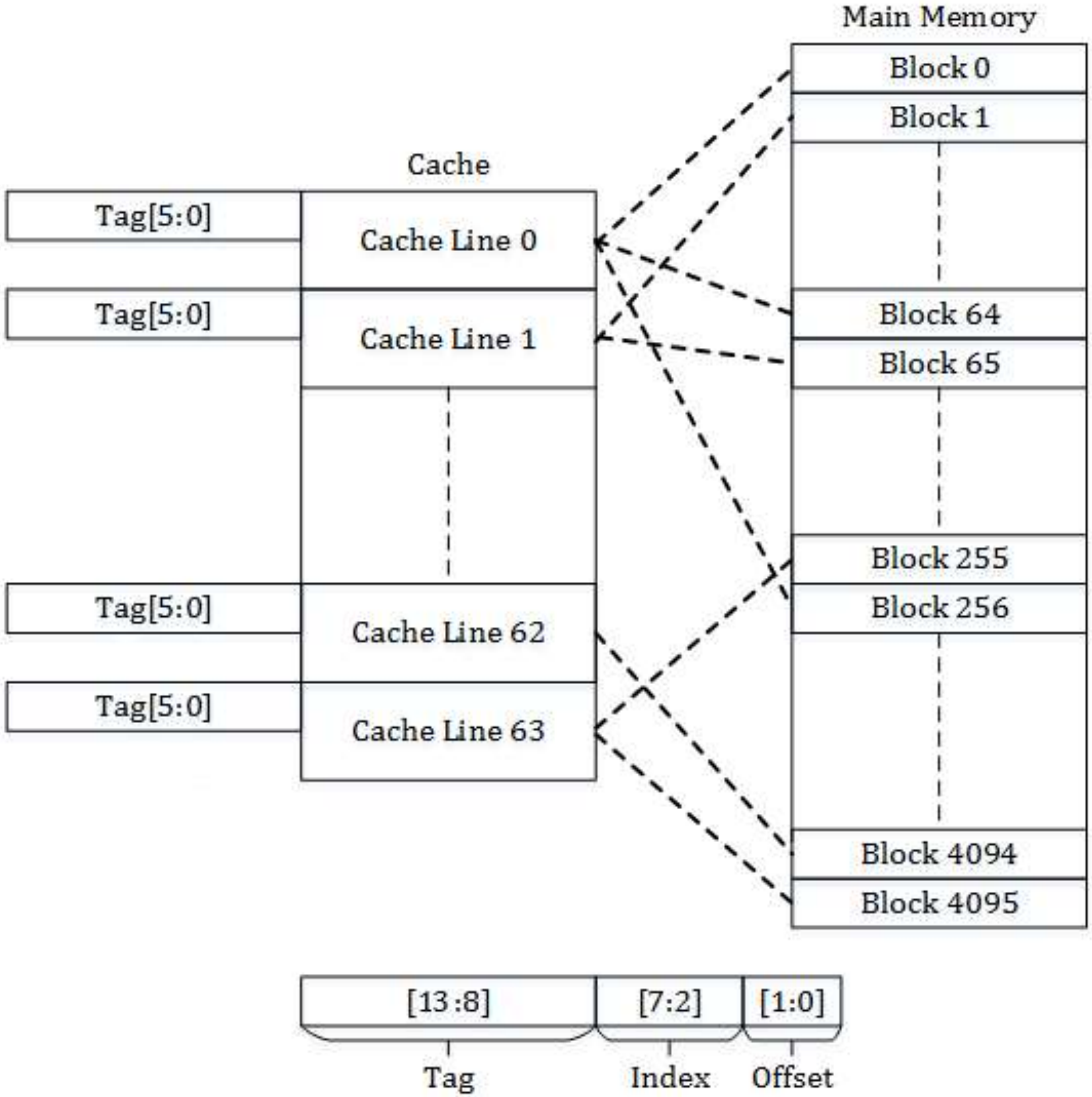
1. Direct mapping cache
2. Set associative memory cache

Direct Mapping

- Cache controller divides memory and cache into blocks where the block is in powers of two
- To distinguish blocks, a unique tag value is assigned to each group of the blocks
- Tags are used to identify a large group of bytes than single byte
- Slide 25 illustrates tag 2 occupying block 2 in cache
- Cache look-up becomes extremely efficient



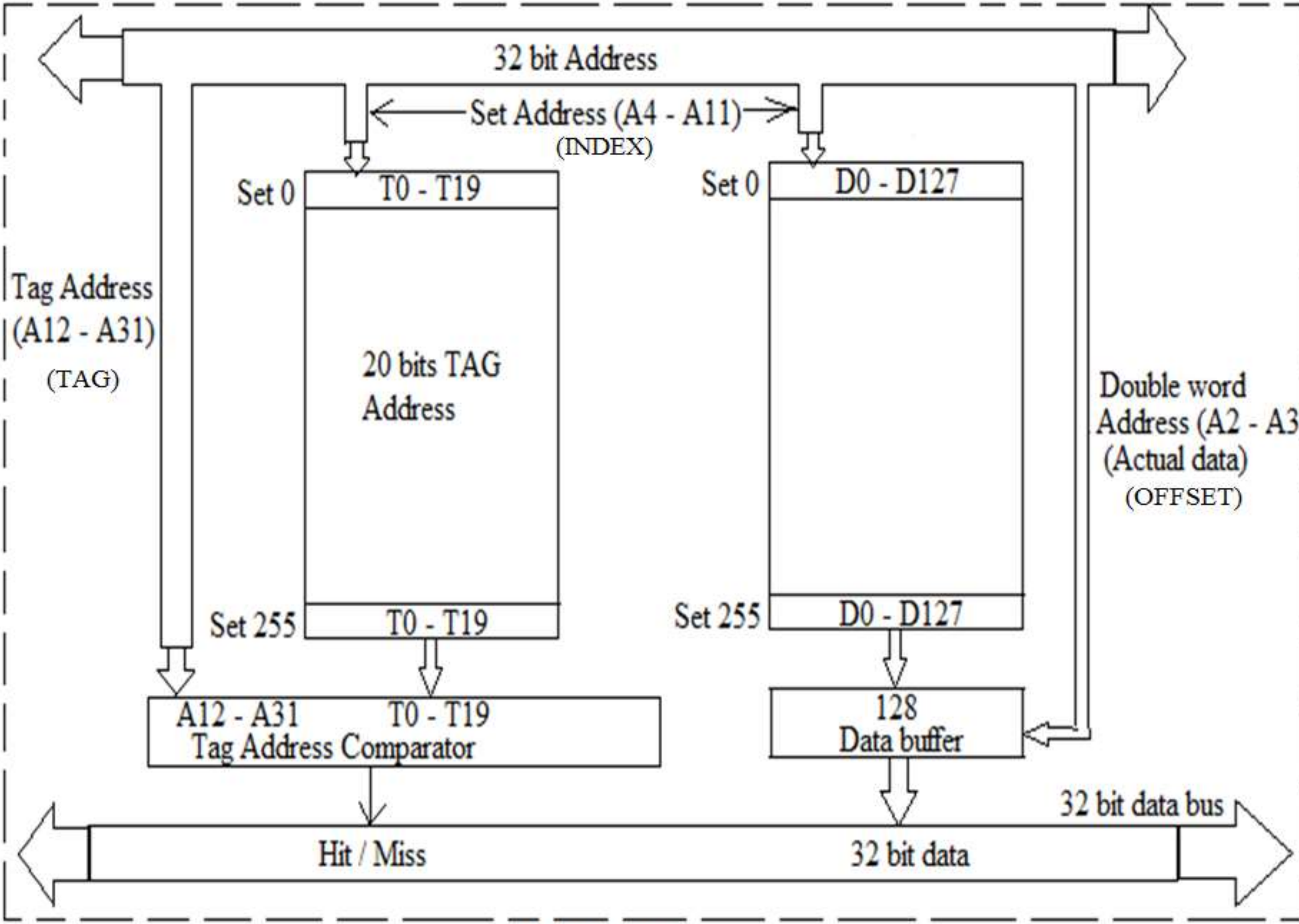
Direct mapping cache



Memory Size = 16Kbytes
Memory Block Size = 4 bytes
Cache Size = 256 bytes
Block Size = 4 bytes
Associativity = 1
Number of Sets = 64

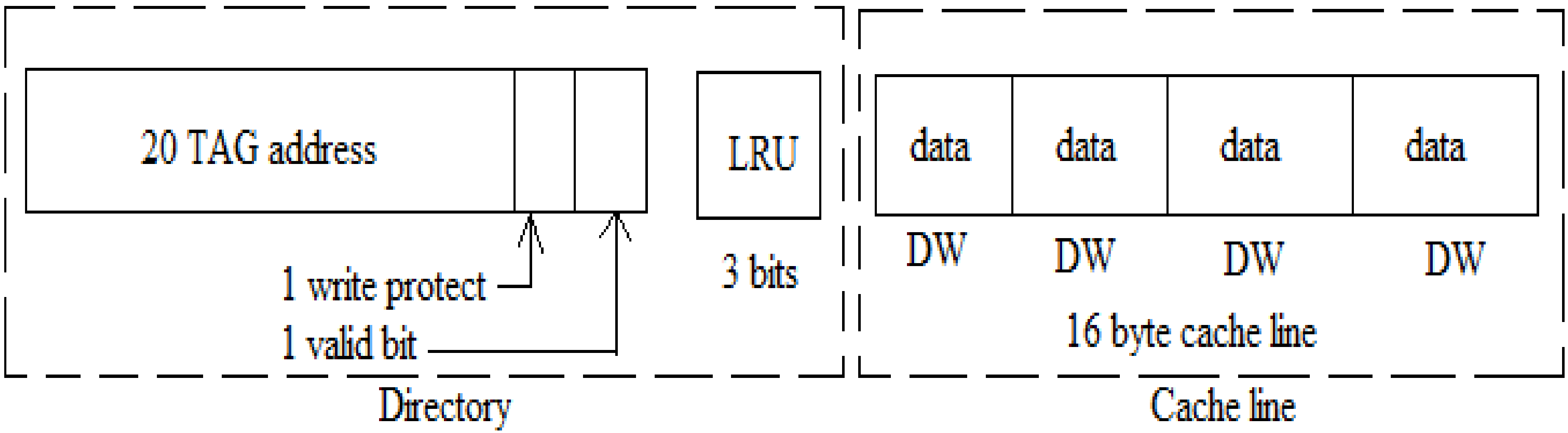
Direct-Mapping Cache

- In a direct-mapped cache structure, the cache is organized into multiple sets with a single cache line per set
- The incoming address to the cache is divided into bits for Offset, Index and Tag.
  - Index — Defines number of sets
  - Tag — Defines the bits to be used for identifying the cache line (cache line identifier)
  - Offset — Bytes to be transparent for information transfer


32 bit Address
Set Address (A4 - A11)
(INDEX)
Set 0
T0 - T19
Set 0
D0 - D127
Tag Address
(A12 - A31)
(TAG)
20 bits TAG
Address
Double word
Address (A2 - A3)
(Actual data)
(OFFSET)
Set 255
T0 - T19
Set 255
D0 - D127
A12 - A31
T0 - T19
Tag Address Comparator
128
Data buffer
32 bit data bus
Hit / Miss
32 bit data

WAY 0

cache directory



- TAG: Element of cache directory
  Determines whether a hit or miss
- Valid bit: implies valid cache line
- Flush: reset valid bits of the cache line
- Write protect: No overwrite
- Cache memory: Stores actual data

- In a direct-mapped cache structure, the cache is organized into multiple sets with a single cache line per set. Based on the address of the memory block, it can only occupy a single cache line. The cache can be framed as a (n*1) column matrix.

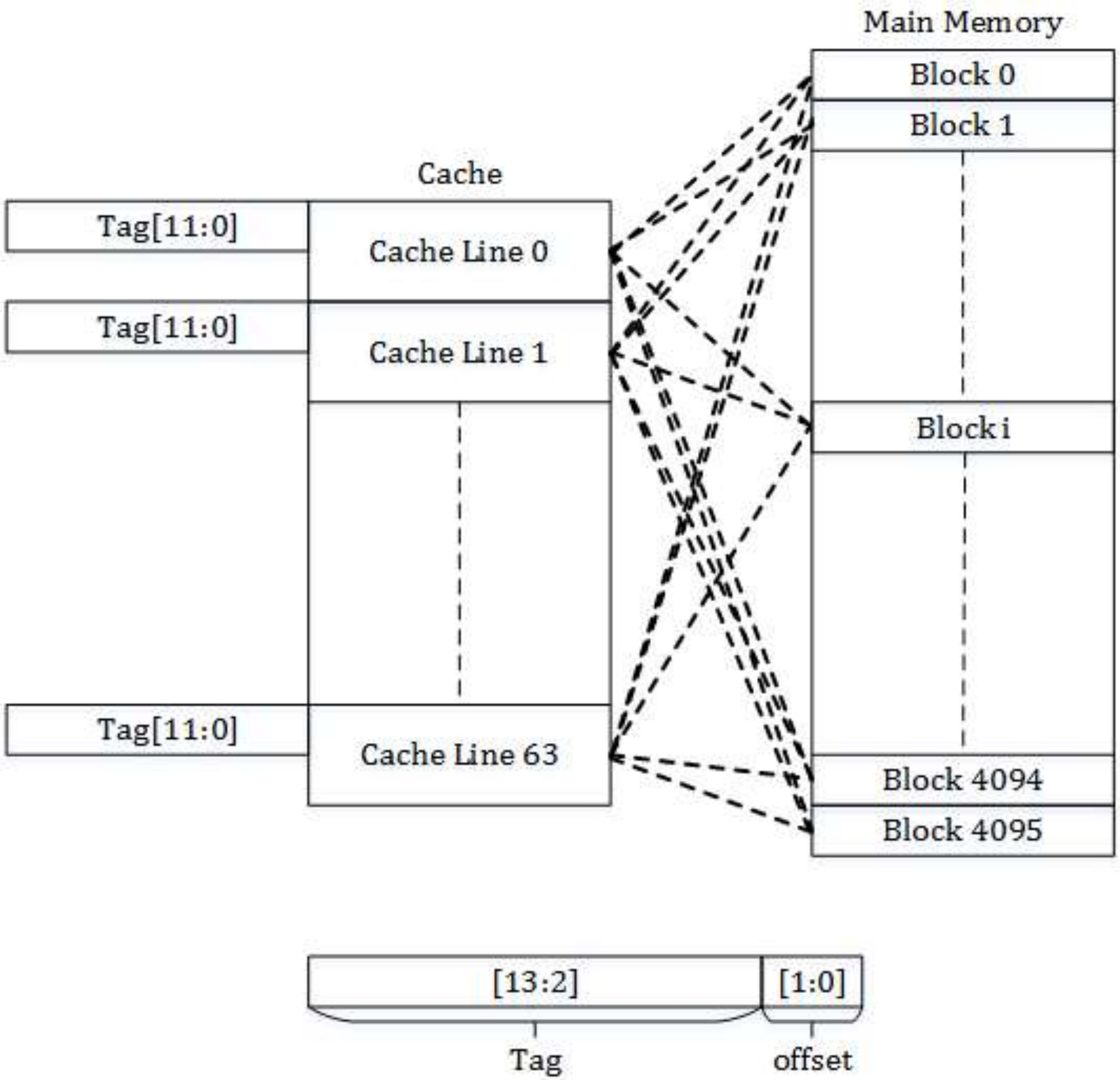
Associative Cache

Associative:

Cache line can be anywhere within a four way structure. Overwrite can be avoided. 2 way would be faster than a 4 way cache.

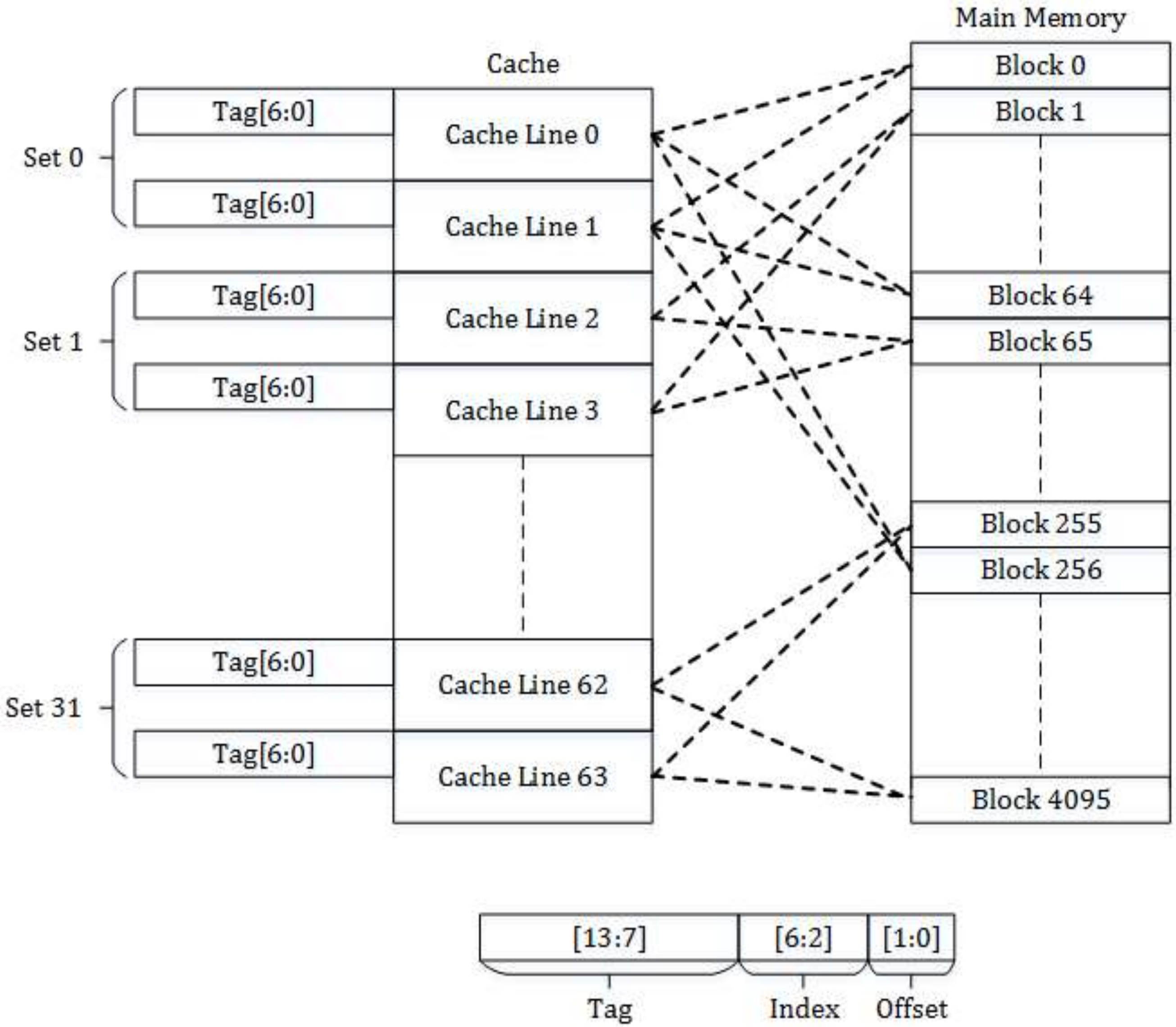
Associative memory concept is also known as Content Addressable Memory (CAM).

- Fully Associative
  - In a fully associative cache, the cache is organized into a single cache set with multiple cache lines.
  - A memory block can occupy any of the cache lines. The cache organization can be framed as (1*m) row matrix.
- Set Associative
  - The cache is divided into ‘n’ sets and each set contains ‘m’ cache lines. A memory block is first mapped onto a set and then placed into any cache line of the set.



Memory Size = 16Kbytes
Memory Block Size = 4 bytes
Cache Size = 256 bytes
Block Size = 4 bytes
Number of Cache Lines = 64

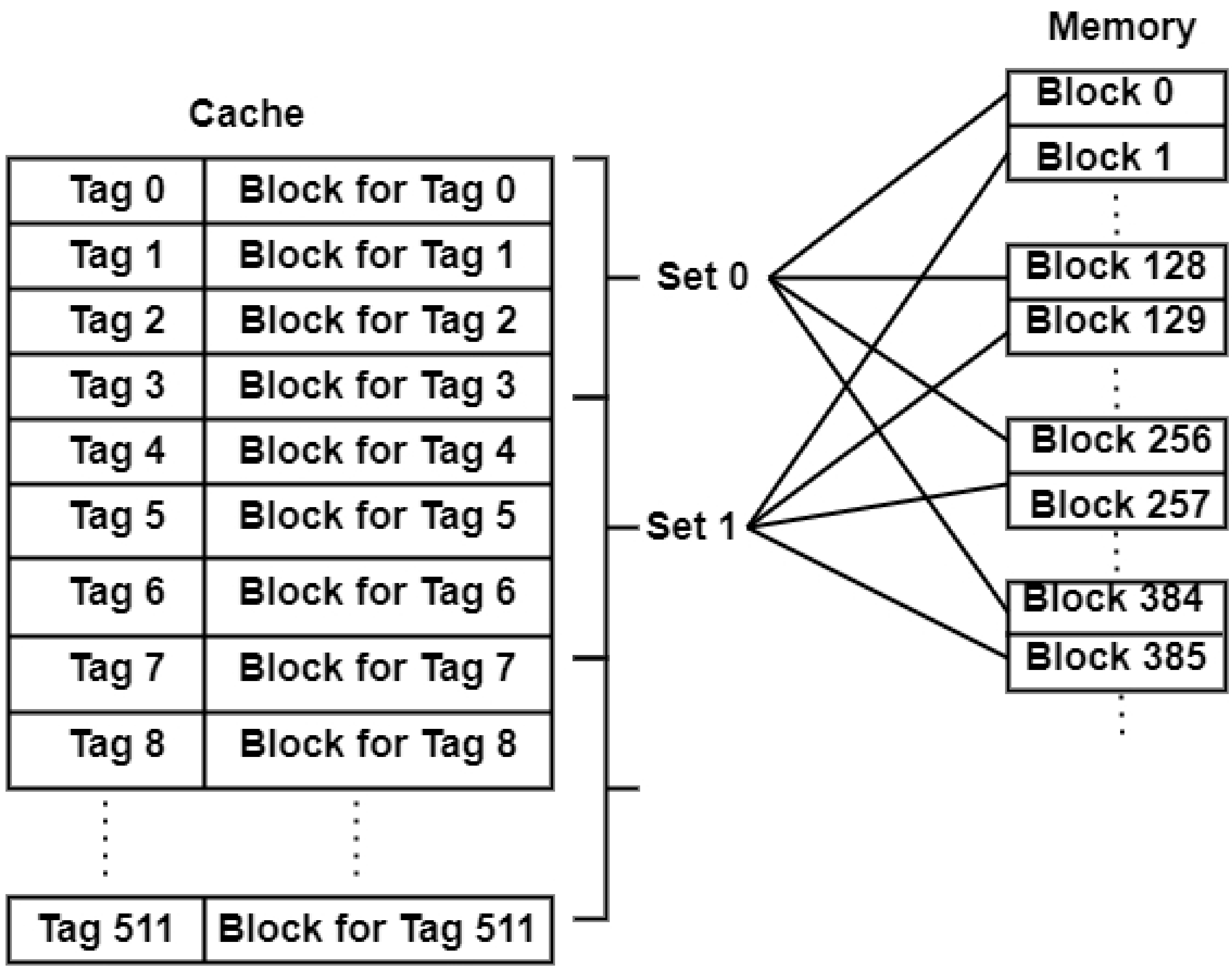
Fully Associative Cache

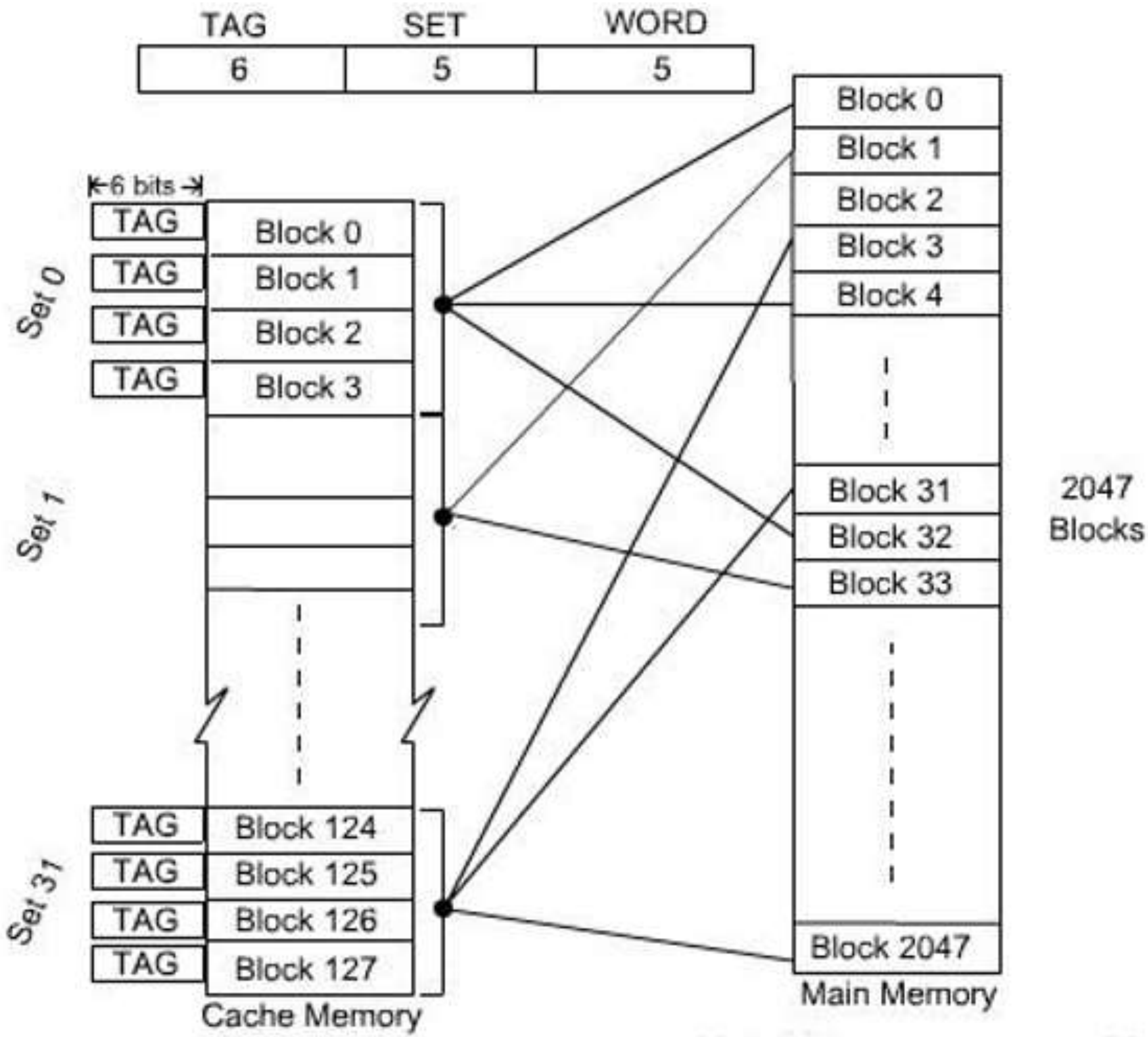


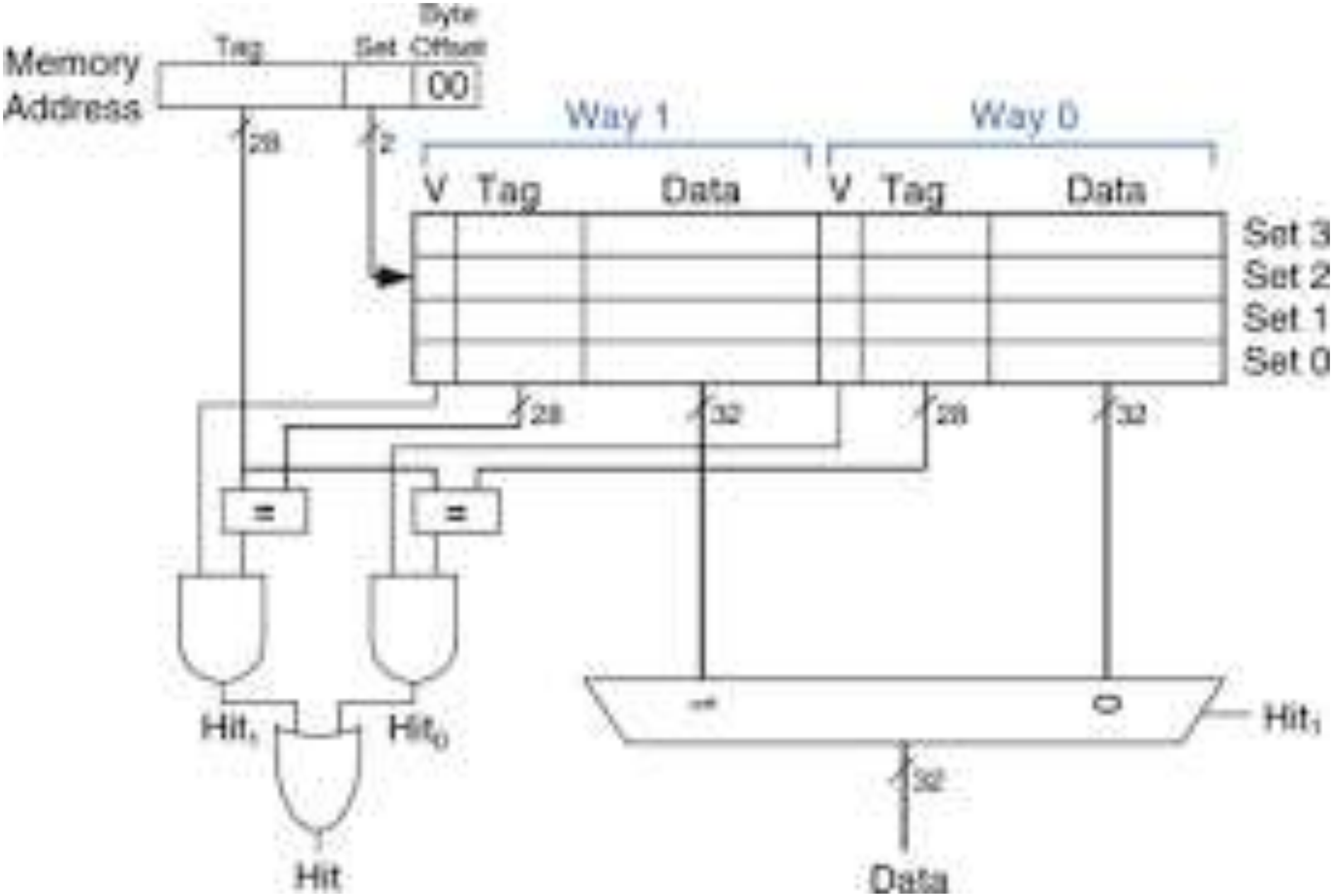
Memory Size = 16Kbytes
Memory Block Size = 4 bytes
Cache Size = 256 bytes
Block Size = 4 bytes
Associativity = 2
Number of Sets = 32

Set Associative Cache

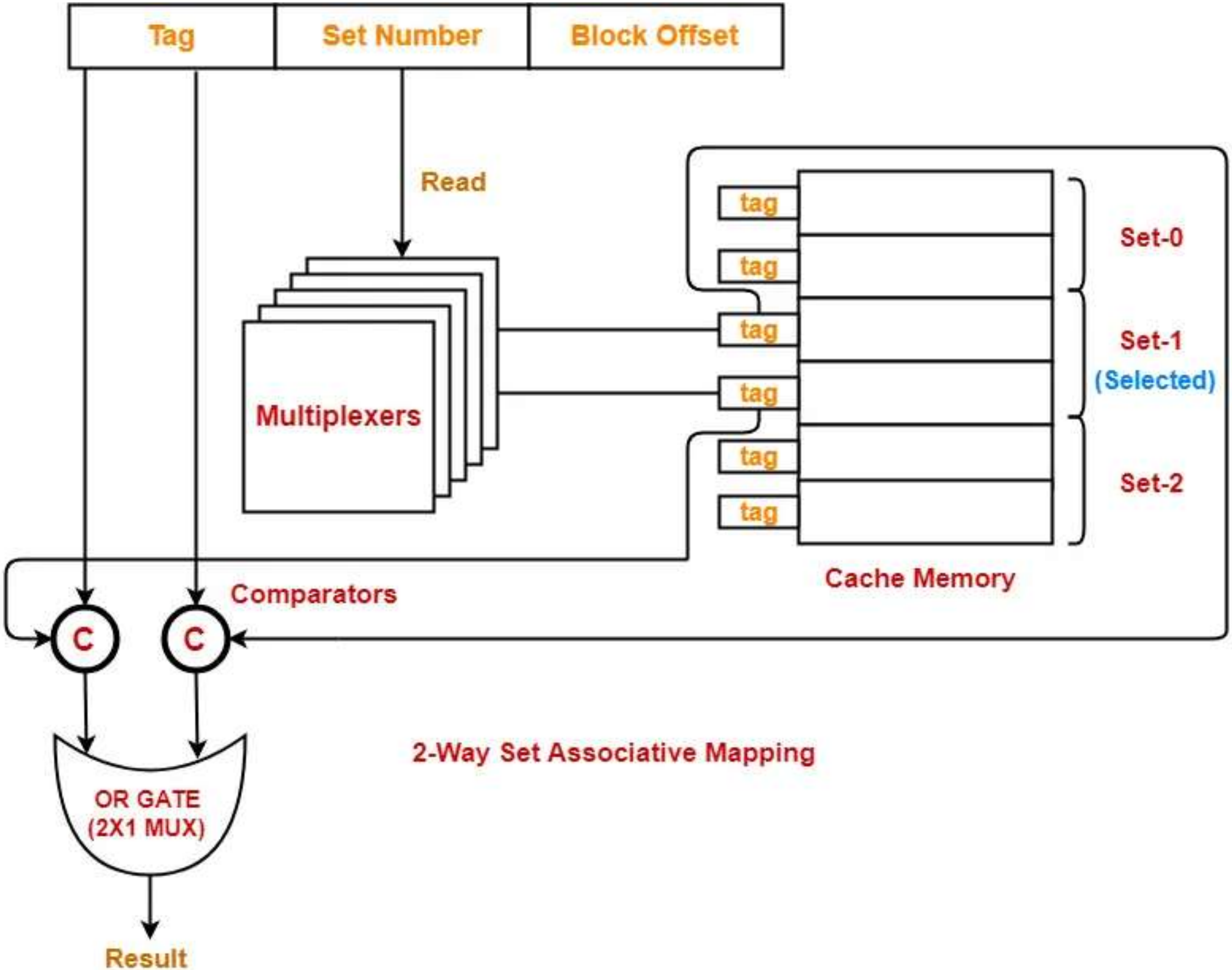
Set Associative Mapping of Main Memory to Cache




TAG
SET
WORD
6
5
5
6 bits
TAG
Block 0
Block 1
Block 2
Block 3
Set 0
Set 1
Set 31
Block 124
Block 125
Block 126
Block 127
Cache Memory
Block 4
Block 31
Block 32
Block 33
Block 2047
Main Memory
2047 Blocks



Set Associative Cache

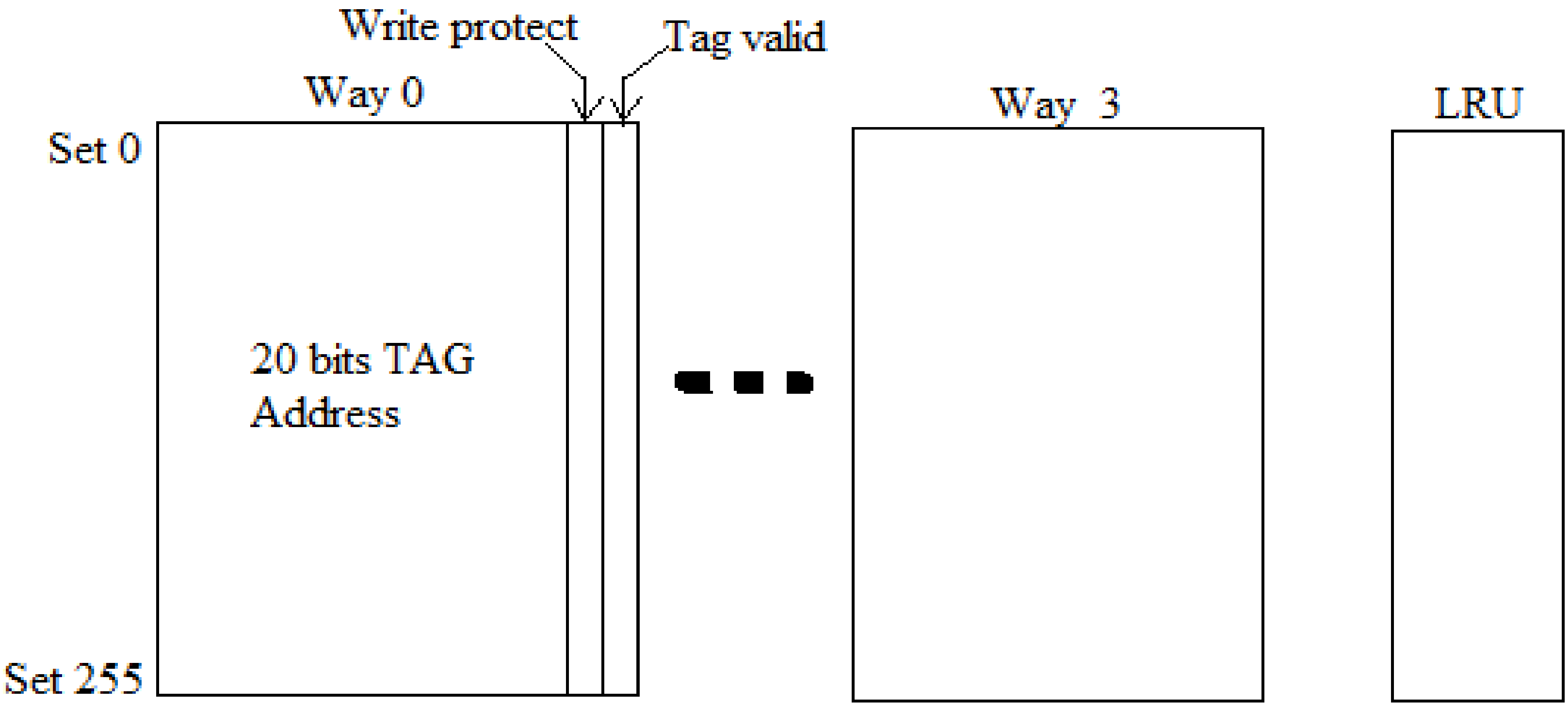


2-Way Set Associative Mapping

- In a k-way set associative mapping,
- Cache lines are grouped into sets where each set contains k number of lines.
- A particular block of main memory can map to only one particular set of the cache.
- However, within that set, the memory block can map to any freely available cache line.
- The set of the cache to which a particular block of the main memory can map is given by-
  - (Main Memory Block Address) Modulo (Number of sets in Cache)

4-Way Cache

Set Associative


Write protect
Tag valid
Way 0
Way 3
LRU
Set 0
Set 255
20 bits TAG
Address

SET

- Every tag of corresponding cache line are elements of the set.
- Way For a given set address, the tag address of all ways are simultaneously compared with the tag part of the address given out of the CPU for a hit/miss criterion.
- Capacity 4 way X set X cache line size = 16Kb
- A Miss will check LRU for replacement.

Algorithms

- Direct Mapping Cache line in one position
- Associative
    - Cache line can be anywhere within the four ways. Overwrite can be avoided.
    - 2 way would be faster than a 4 way cache. Associative memory concept is also known as Content Addressable Memory (CAM).

Replacement Policy

Replacement policy

Need To increase the hit ratio:

1. The policy should retain those items that will be referenced most frequently
2. Should be inexpensive
3. LRU method preferred

Pre-loading Caches

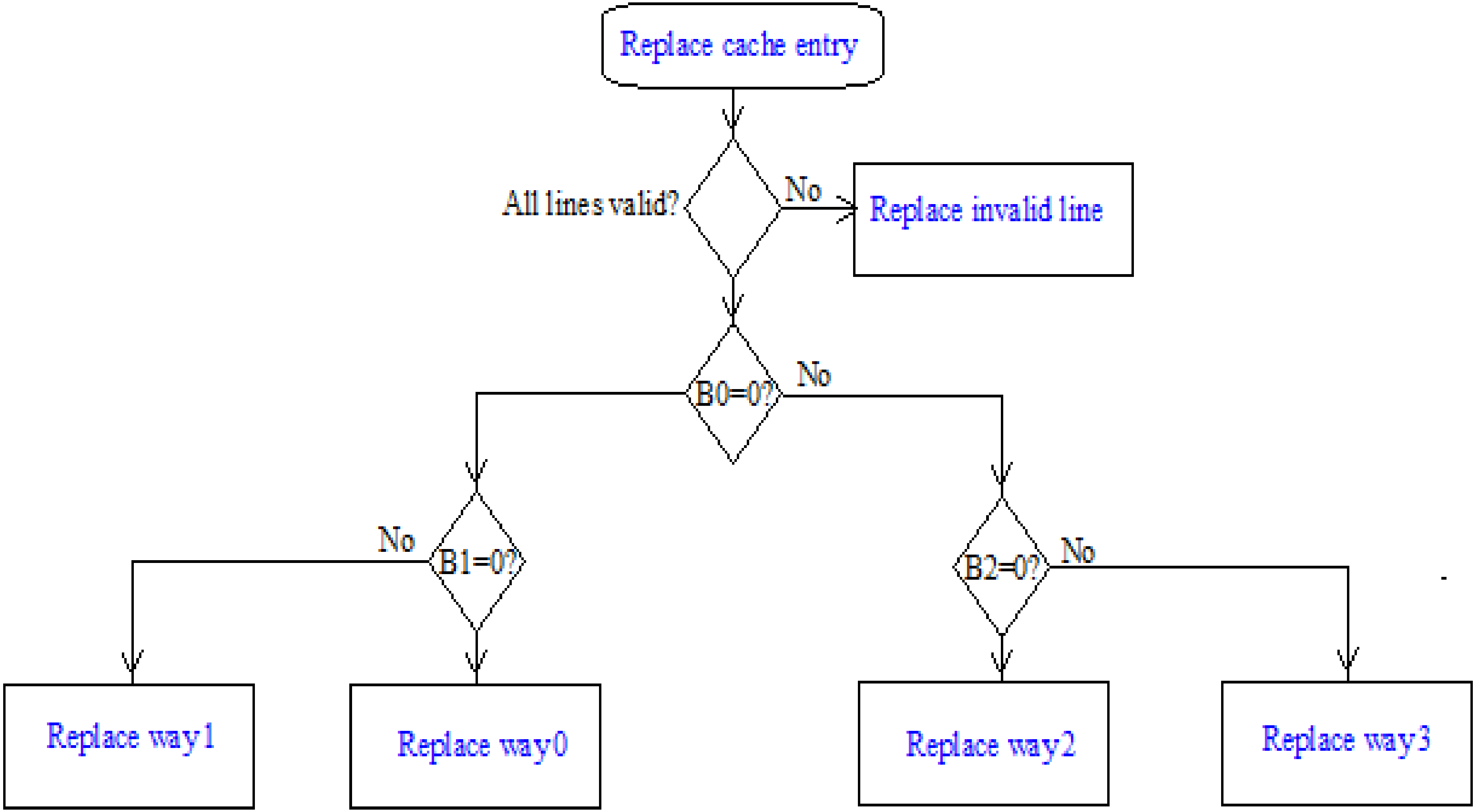
During start-up the hit ratio is very low since it has to fetch items from the store. This can be improved by preloading the cache.

- Using anticipation of requests (repeated)
- Frequently used pages

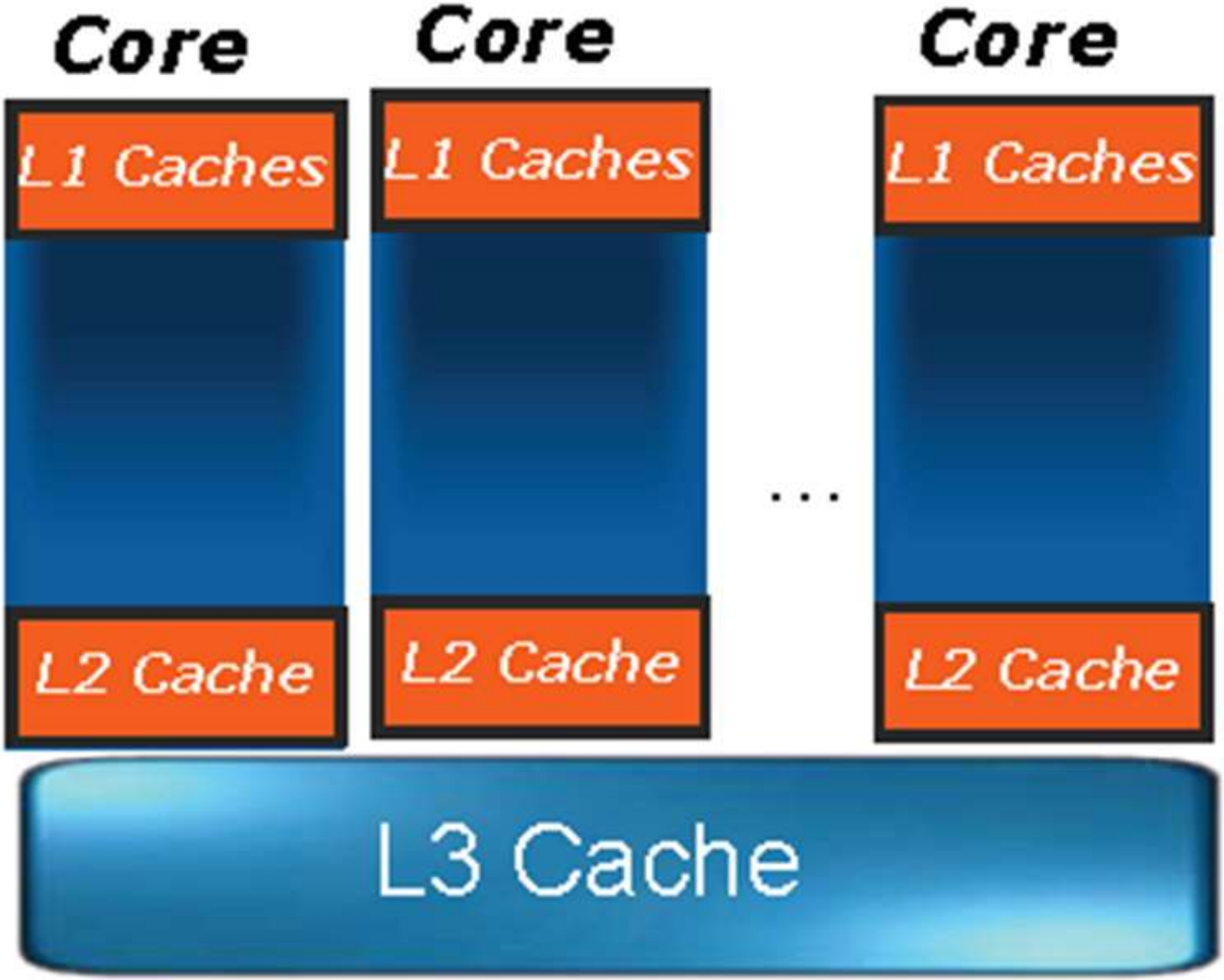
Access and Addressing

- If last access was way 0 or 1, controller sets LRU bit B0. For access to way 0, bit B1 is set; for addressing way 1, B1 is cleared.
- If last access was way 2 or 3, the controller clears B0. If way 2 last accessed, B2 is set; for addressing way 3, B2 is cleared.
- LRU bits are updated upon every access and cleared upon each cache reset or cache flush.
- Random replacement is also possible.
- Replacement policy will solely rely on the cache designer.

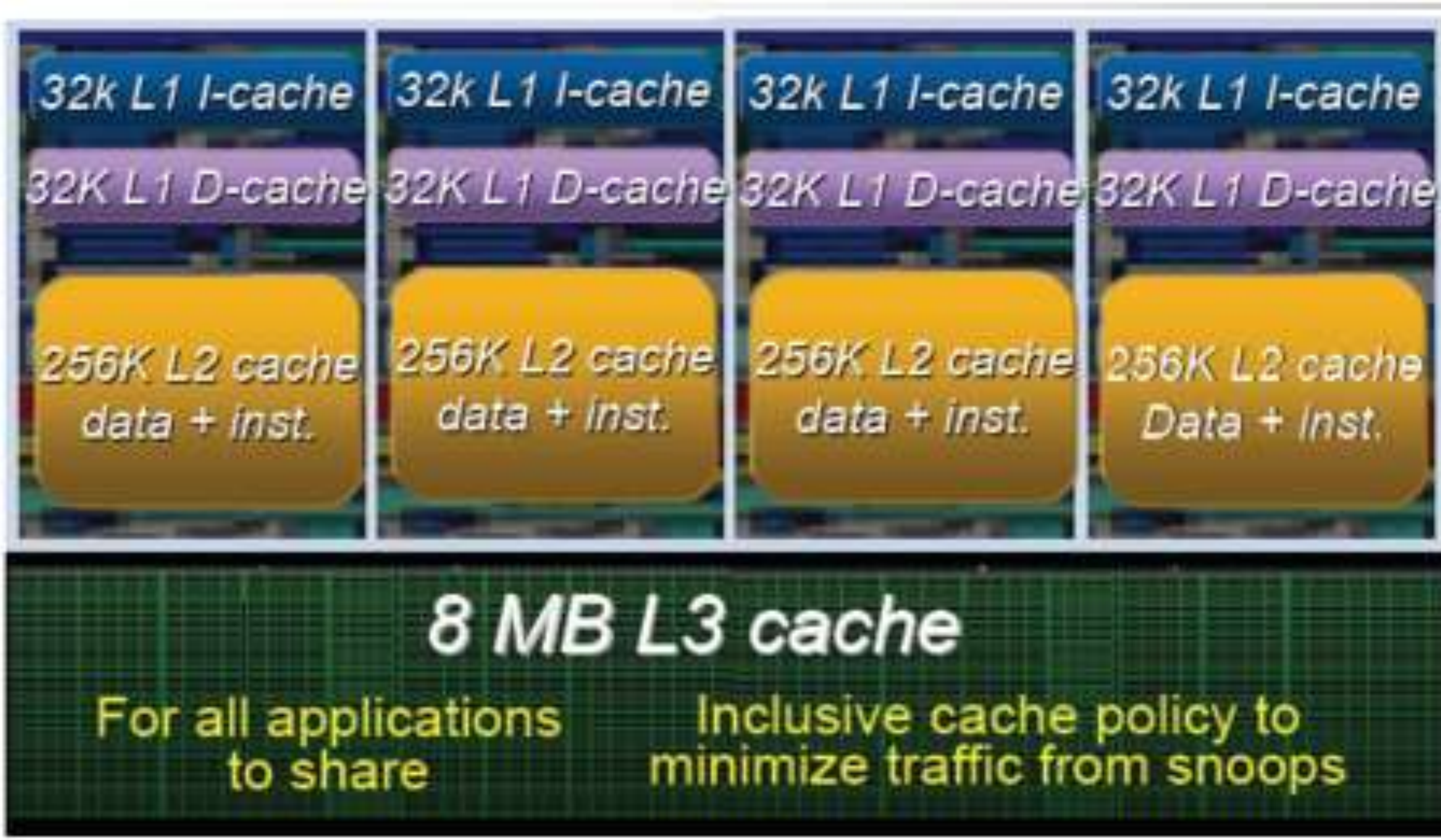
Replacement Policy



Cache Applications



Three level cache



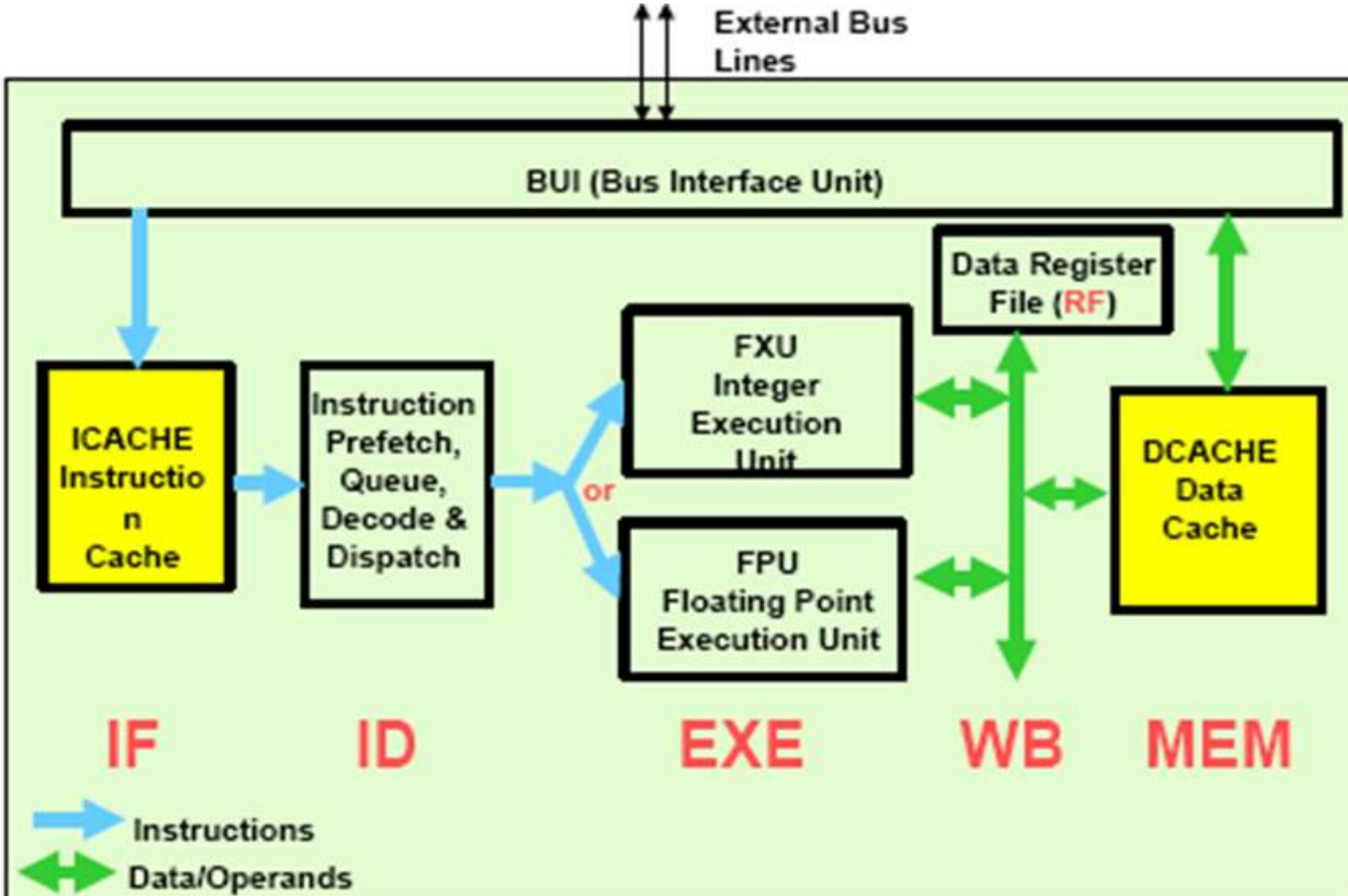
Intel core i7 (2008) 3-level cache

µP Bus	FETCH	Decode 1	Execute 1	FETCH 2	Decode 2	Execute 2	FETCH 3	Decode 3	Execute 3
	BUSY	IDLE	BUSY	BUSY	IDLE	BUSY	BUSY	IDLE	BUSY

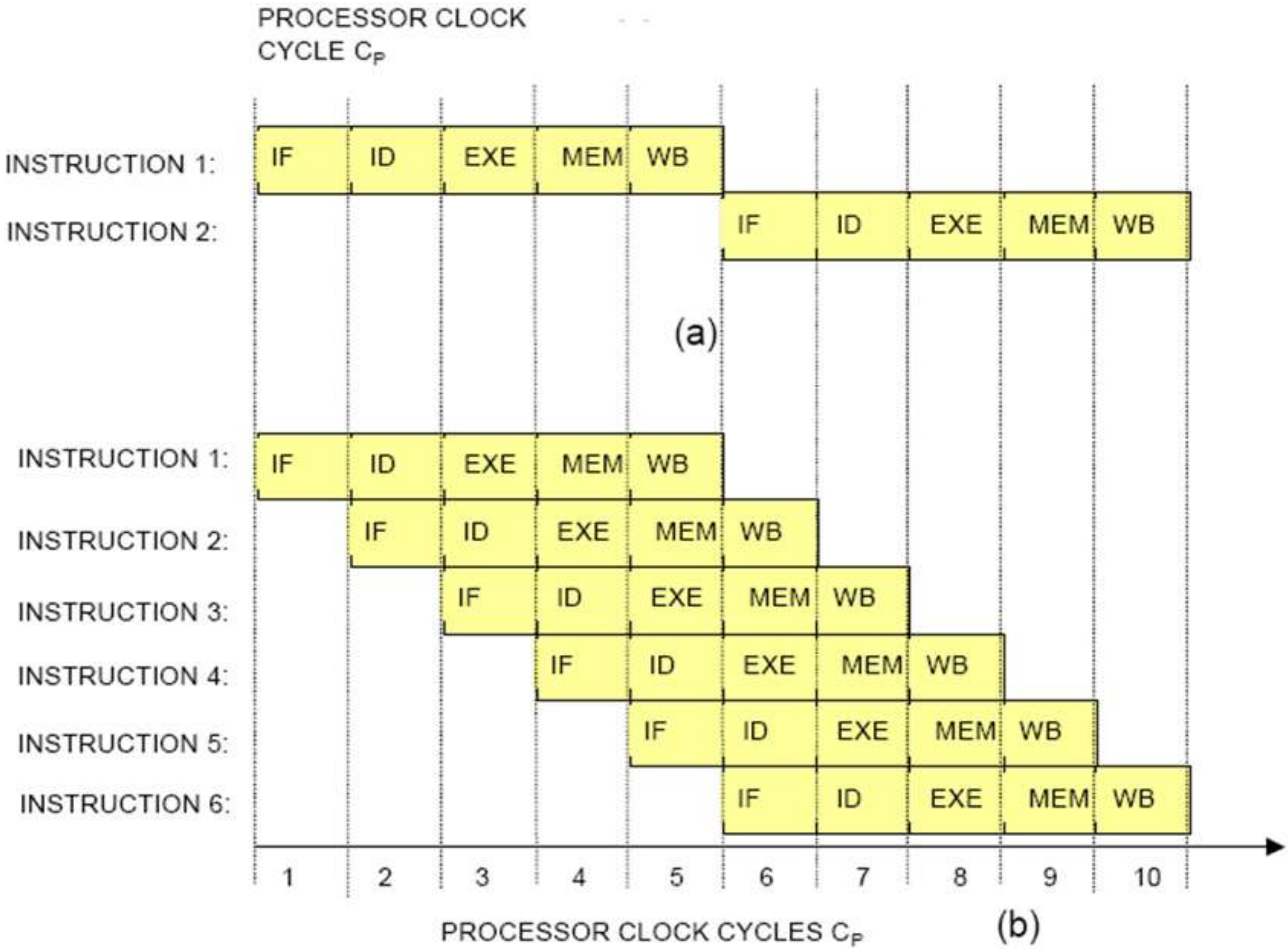
Non-Pipelined Execution (8085)

Bus Unit	FETCH 1	FETCH 2	FETCH 3	FETCH 4	STORE1	FETCH 5	FETCH 6	READ	FETCH 7
Instruction Unit		Decode 1	Decode 2	Decode 3	Decode 4	IDLE	Decode 5	Decode 6	IDLE
Execution Unit			Execute 1	Execute 2	Execute 3	Execute 4	IDLE	Execute 5	Execute 6
Address Unit				Generate Address 1			Generate Address 2		

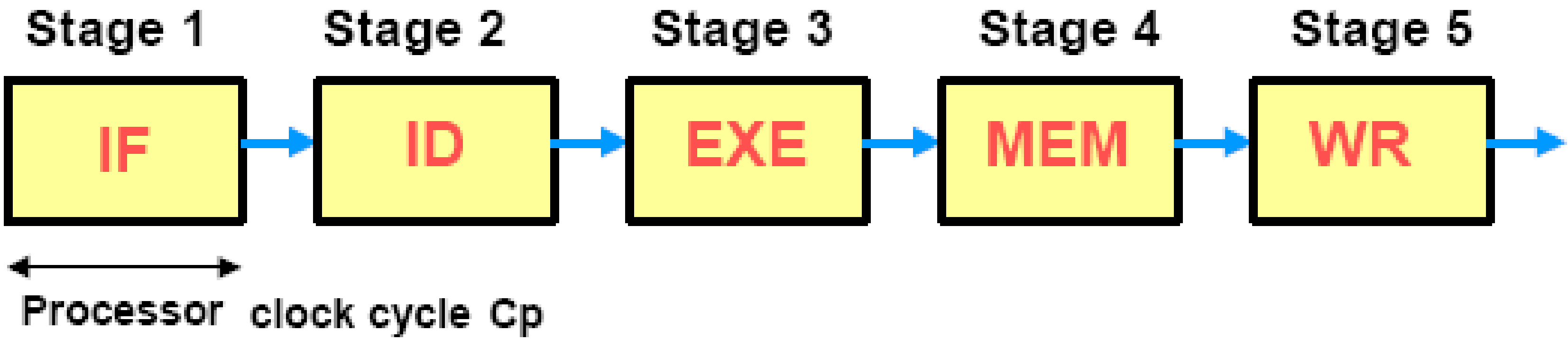
Pipelining of instructions



The model of a 5-stage <u>scalar pipelined</u> processor.
(One instruction executed per processor clock cycle).



Comparing a nonpipelined with a 5-stage scalar pipelined operation



5 independent functional units (FUs) of the pipeline:

- ICACHE: instruction memory for the instr fetch stage
- Decoder and register file's (RF) read ports
- ALU for the execution stage
- DCACHE: data memory for the MEM stage
- Register file's (RF) write port for the (write register) WR stage