# Cache Example

Main memory: Byte addressable memory of size 4GB = $2^{32}$ bytes

Cache size: 64KB = $2^{16}$ bytes

Block (line) size : 64 bytes = $2^{6}$ bytes

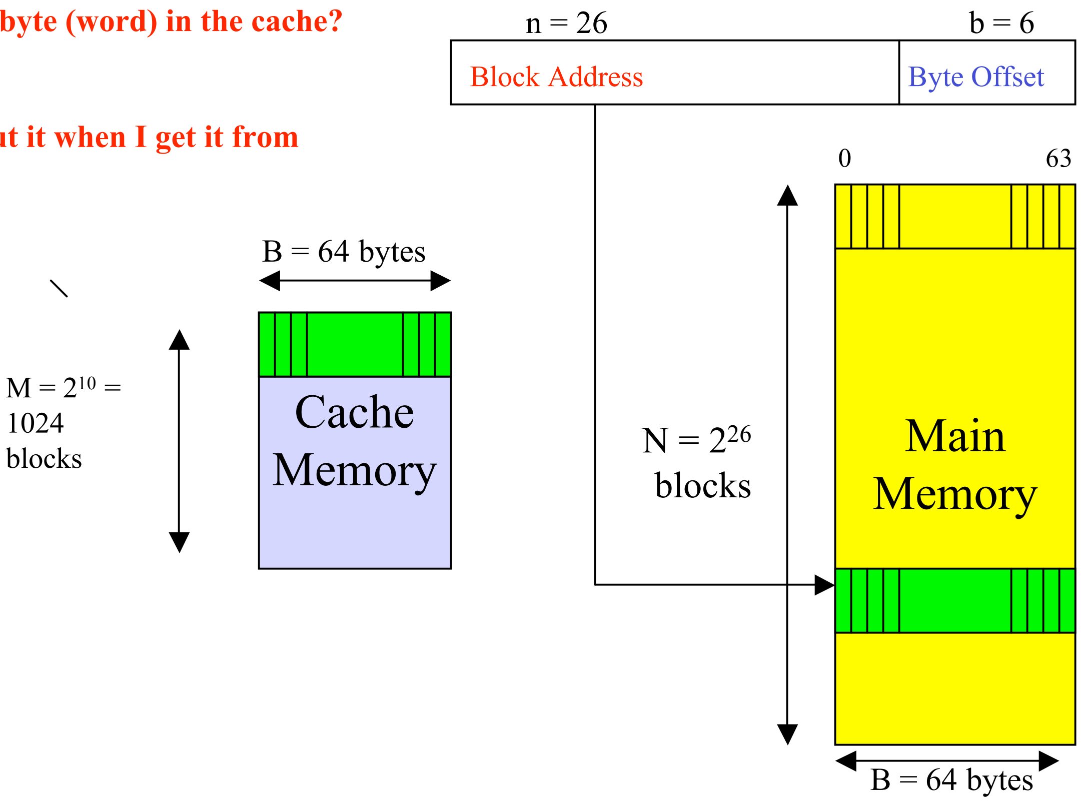
Number of memory blocks = $2^{32} / 2^{6} = 2^{26}$

Number of cache blocks = $2^{16} / 2^{6} = 2^{10}$

**Is the accessed memory byte (word) in the cache?**

**If so where?**

**If not, where should I put it when I get it from main memory?**



## Fully Associative Cache Organization

- Fully-Associative
- Set-Associative
- Direct-Mapped Cache

A cache line can hold any block of main memory

A block in  main memory can be placed in any cache line

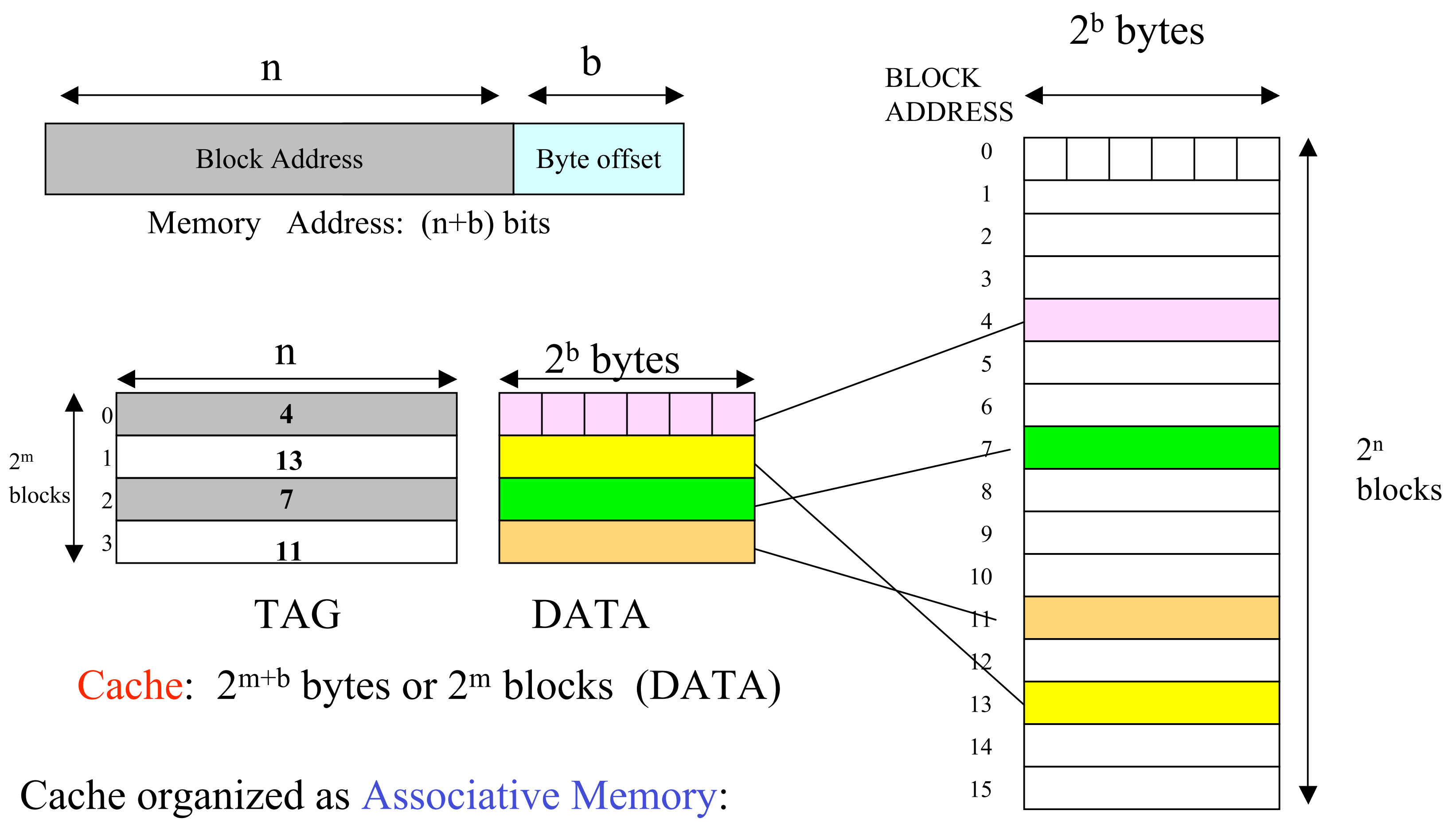
Many- Many mapping

Maintain a directory structure to indicate which block of memory currently occupies a cache block

Directory structure known as the TAG Array

The TAG entry for a cache stores the block number of the  memory block currently in that cache location

# Fully Associative Cache Organization



Cache: $2^{m+b}$ bytes or $2^m$ blocks (DATA)

Memory
$2^{n+b}$ bytes or $2^n$ blocks

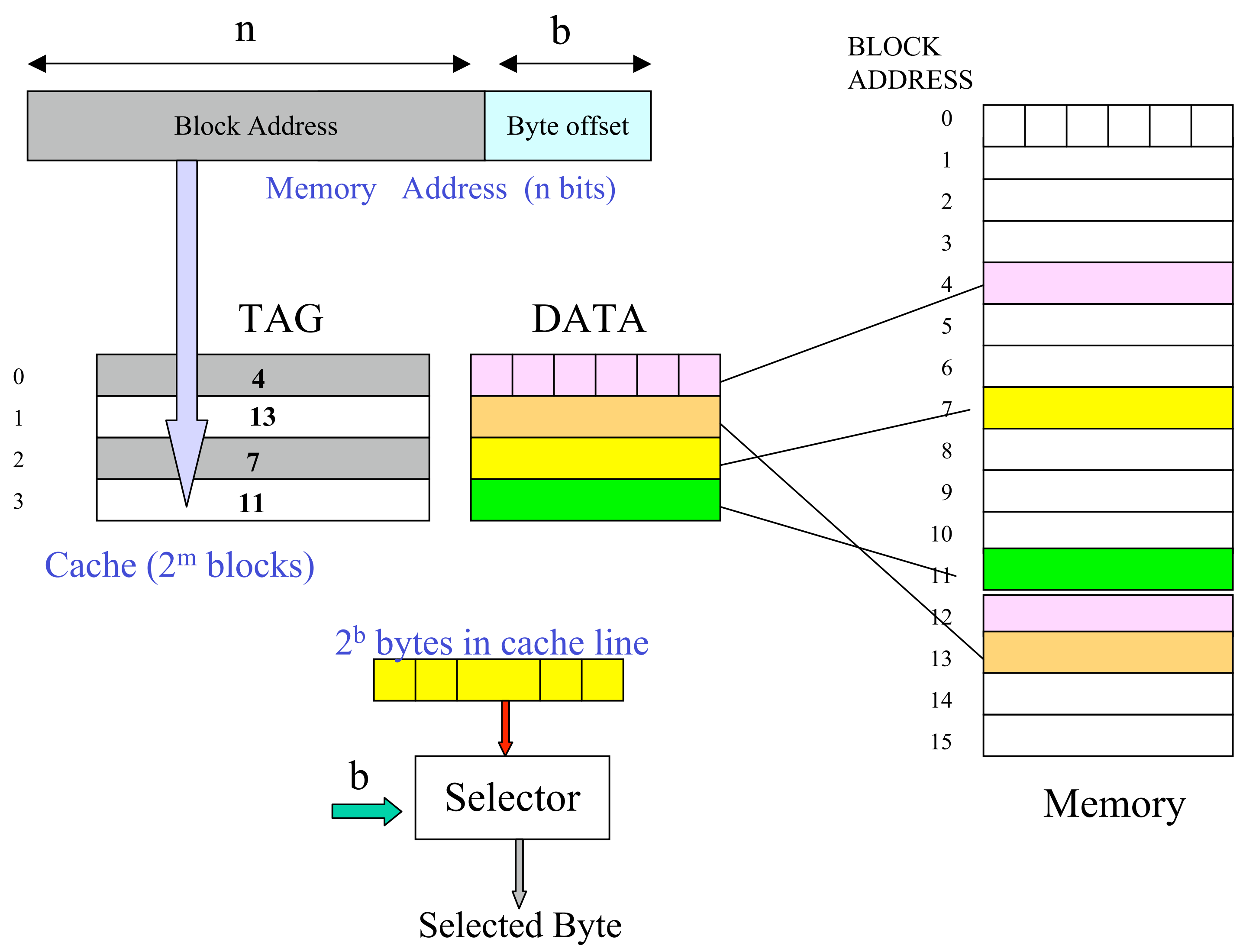
Cache organized as Associative Memory:

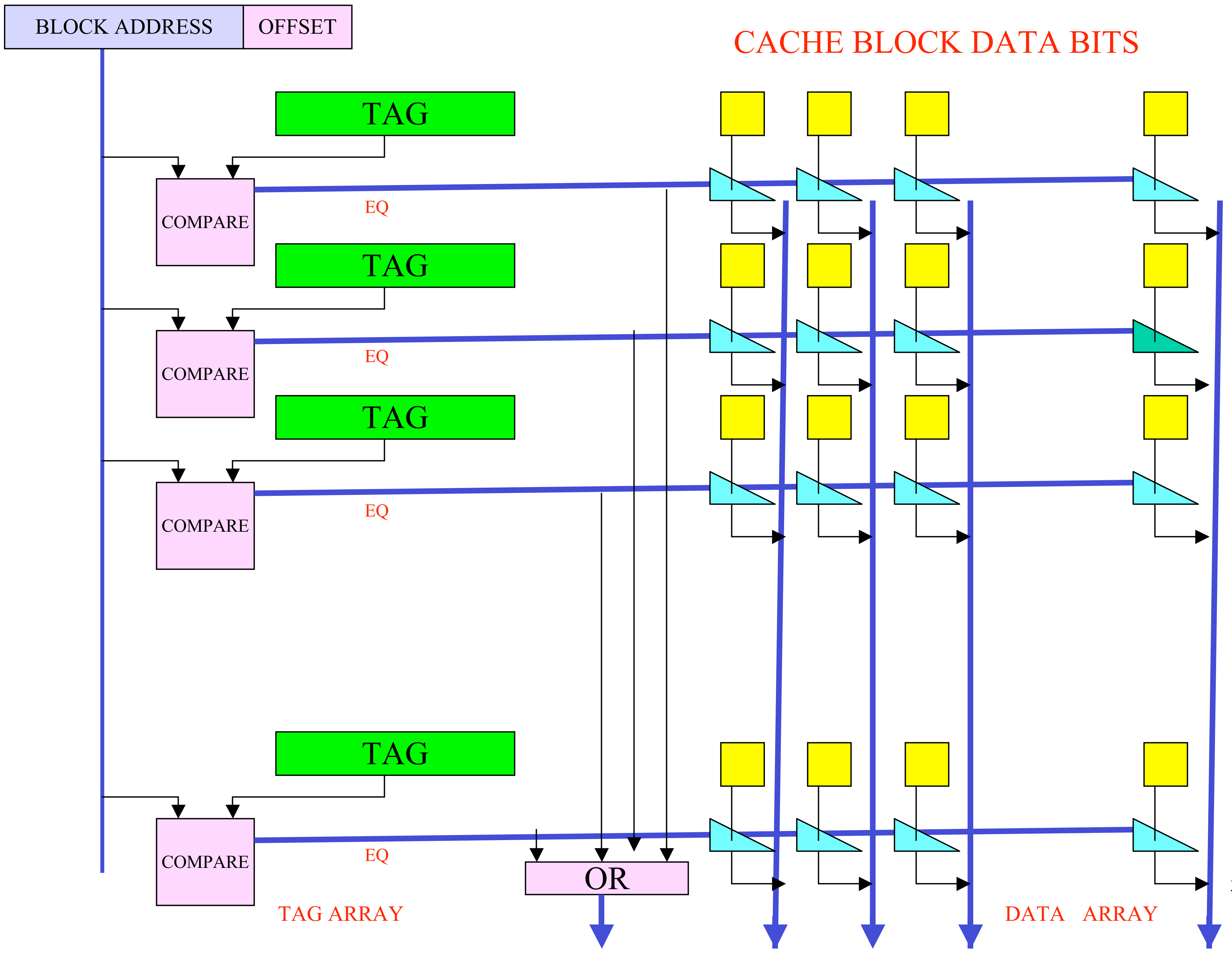
TAG field holds the block address of the memory block stored in the cache line

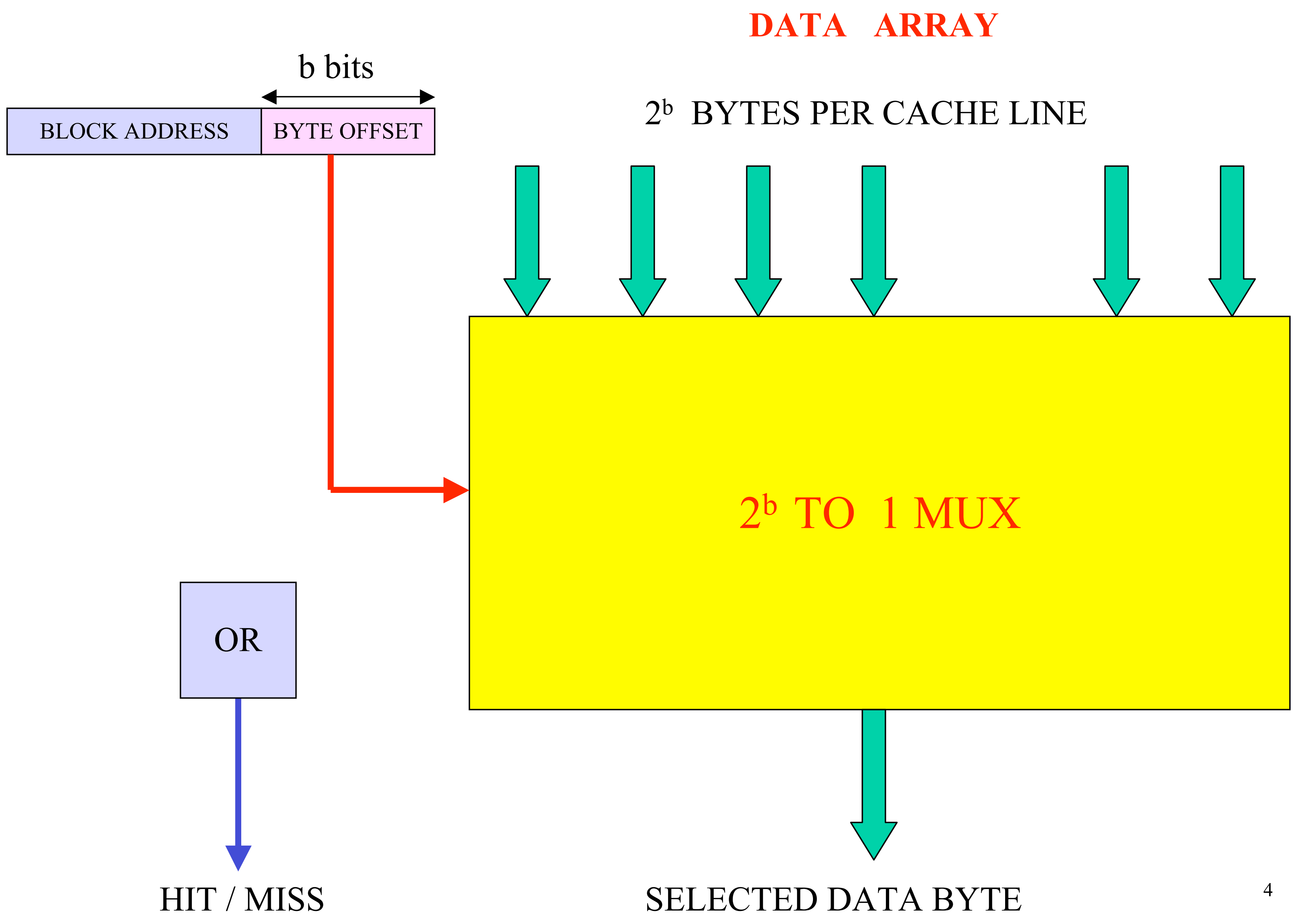
Hardware compares Block Address field of memory address with the TAG fields of each cache block (Associative search -- access by value)

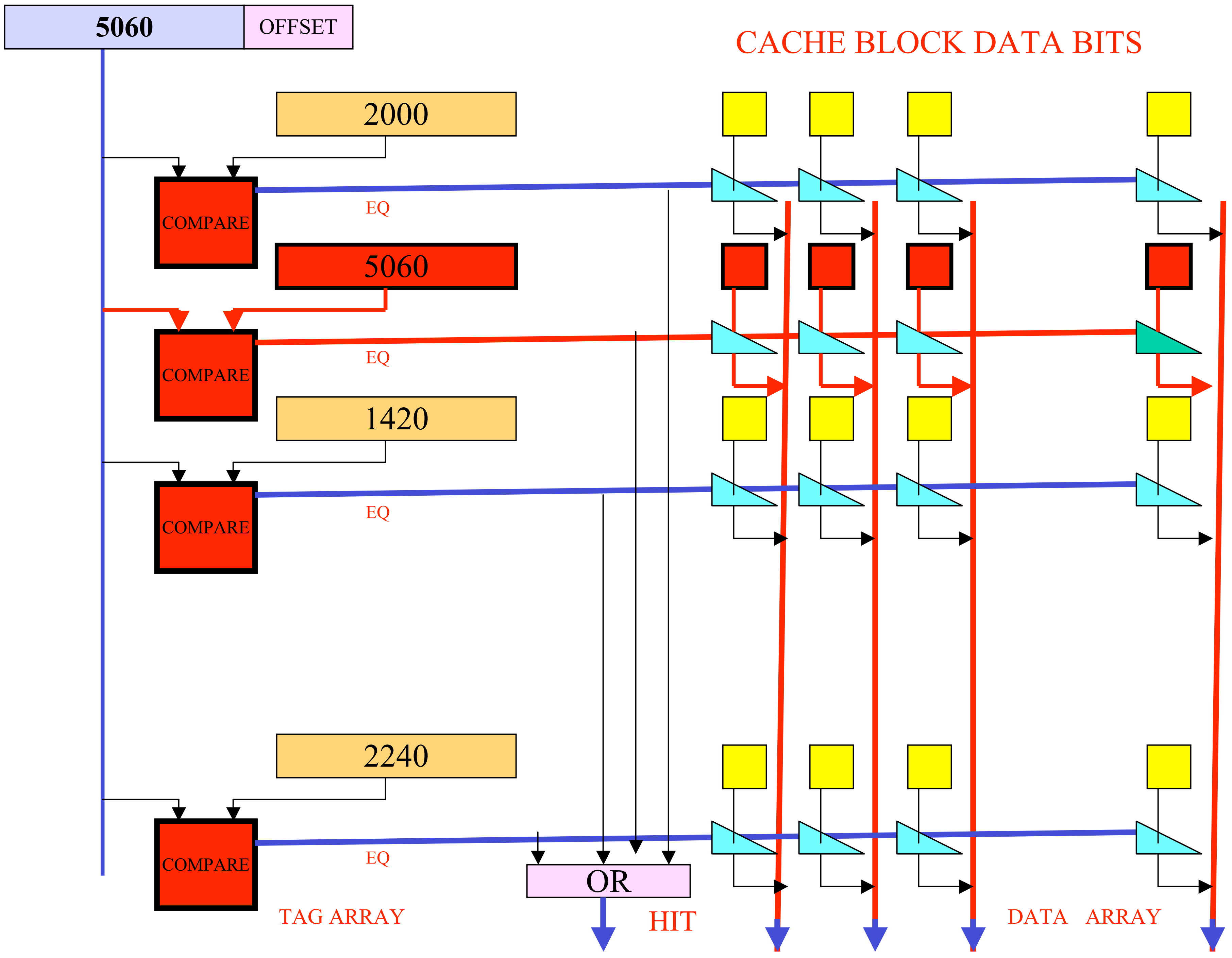
TAG field is n bits

# Fully Associative Cache Organization

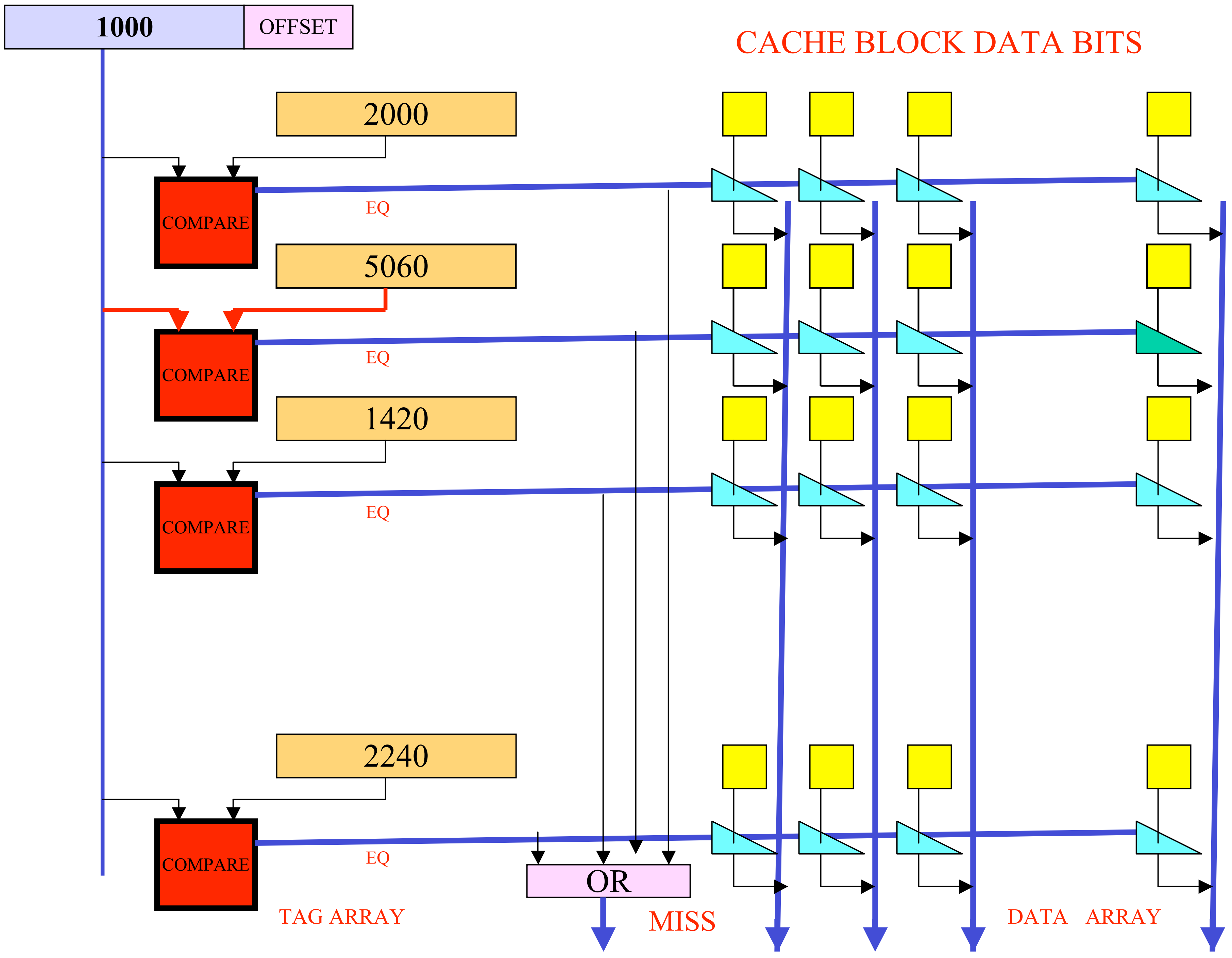



BLOCK ADDRESS
OFFSET
CACHE BLOCK DATA BITS
TAG
COMPARE
EQ
TAG
COMPARE
EQ
TAG
COMPARE
EQ
TAG
COMPARE
EQ
OR
TAG ARRAY
DATA ARRAY

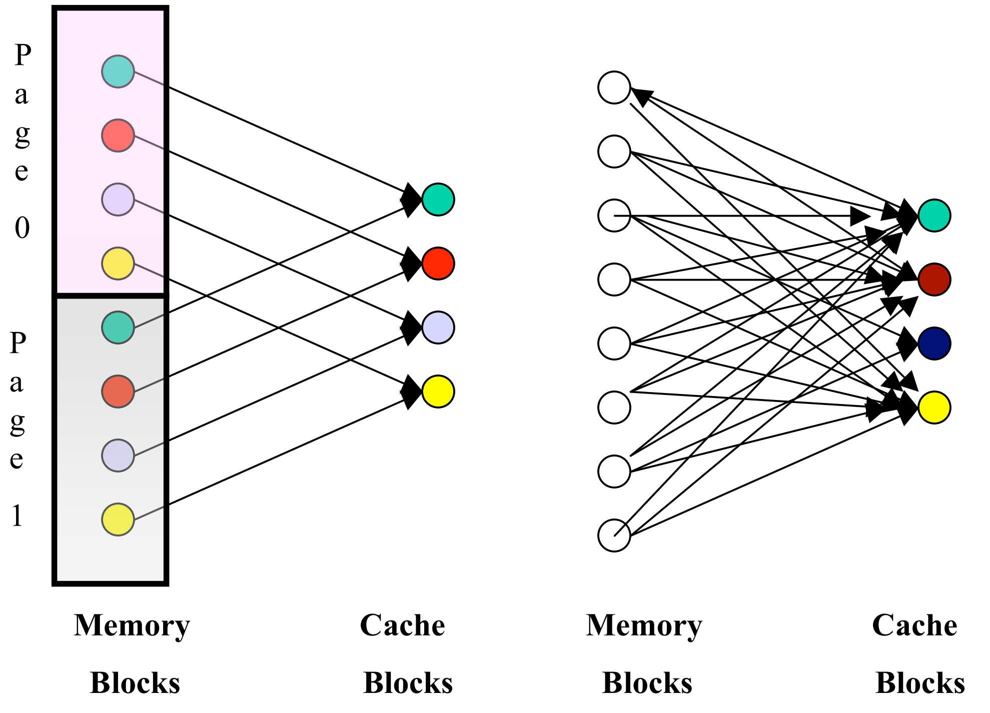

DATA ARRAY
b bits
BLOCK ADDRESS
BYTE OFFSET
$2^b$ BYTES PER CACHE LINE
$2^b$ TO 1 MUX
OR
HIT / MISS
SELECTED DATA BYTE



5060
OFFSET
CACHE BLOCK DATA BITS
2000
COMPARE
EQ
5060
COMPARE
EQ
1420
COMPARE
EQ
2240
COMPARE
EQ
OR
HIT
TAG ARRAY
DATA ARRAY


1000
OFFSET
CACHE BLOCK DATA BITS
2000
COMPARE
EQ
5060
COMPARE
EQ
1420
COMPARE
EQ
2240
COMPARE
EQ
OR
TAG ARRAY
MISS
DATA ARRAY

# Direct Mapped and Fully Associative Cache Organizations



**Direct-Mapped Cache mapping**

**Fully Associative mapping**

- All cache blocks have different colors
- Memory blocks in each page cycle through the same colors in order
- A memory block can be placed only in a cache block of matching color

- A memory block can be placed in any cache block

## Direct Mapped Cache Organization

- **Direct-Mapped Cache**
- Fully-Associative
- Set-Associative

- Restrict possible placements of a memory block in the cache

- A block in main memory can be placed in exactly one location in the cache

- A cache line can be target of only a subset of possible memory blocks

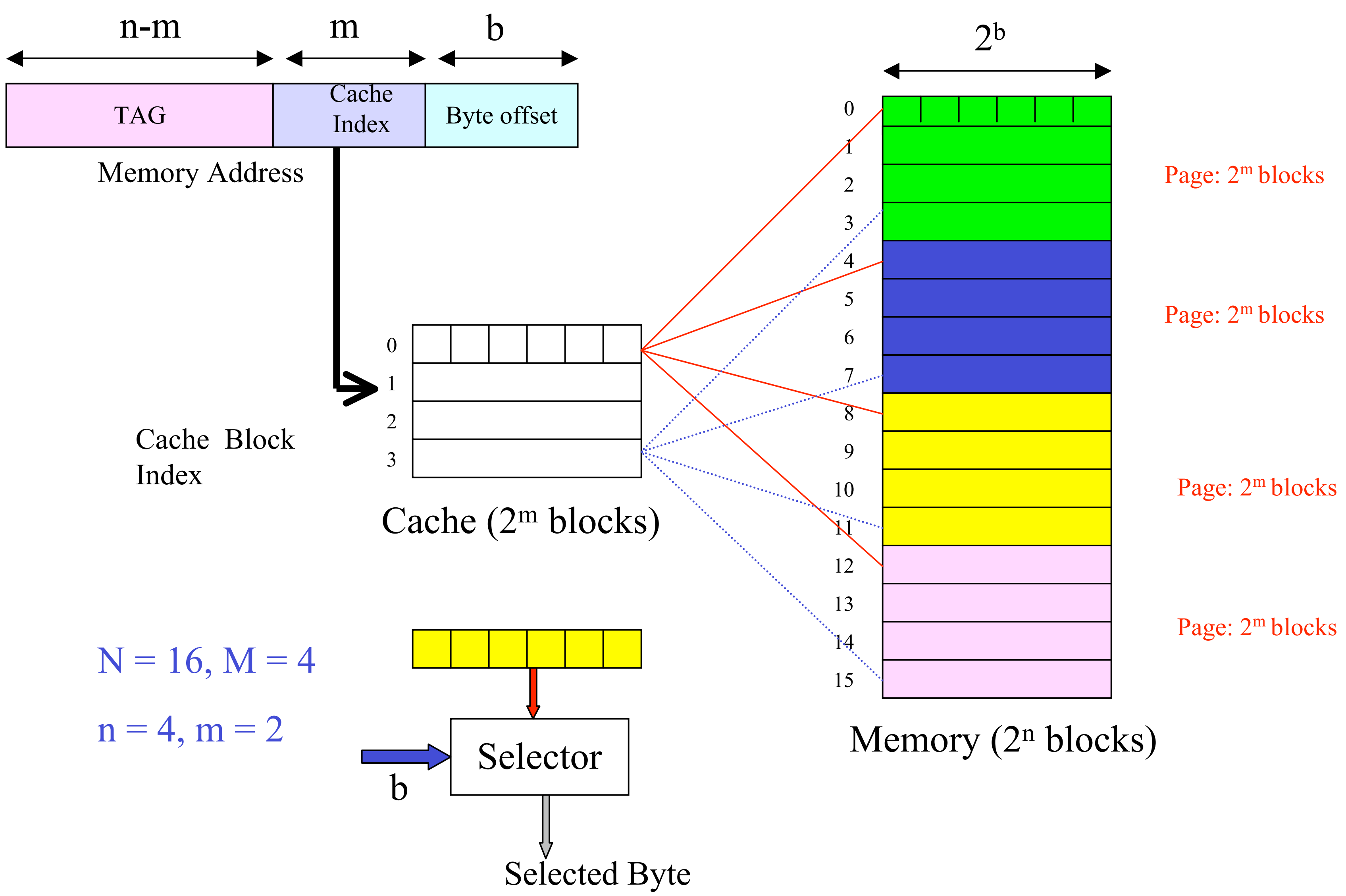
- Many - 1 relation from memory blocks to cache lines

- Useful to think of memory divided into pages of contiguous blocks
  - **Do not confuse this use of memory page with that used in Virtual Memory**

- Size of a page is the size of the Direct Mapped Cache

- The k$^{th}$ block in any page can be mapped only to the k$^{th}$ cache line

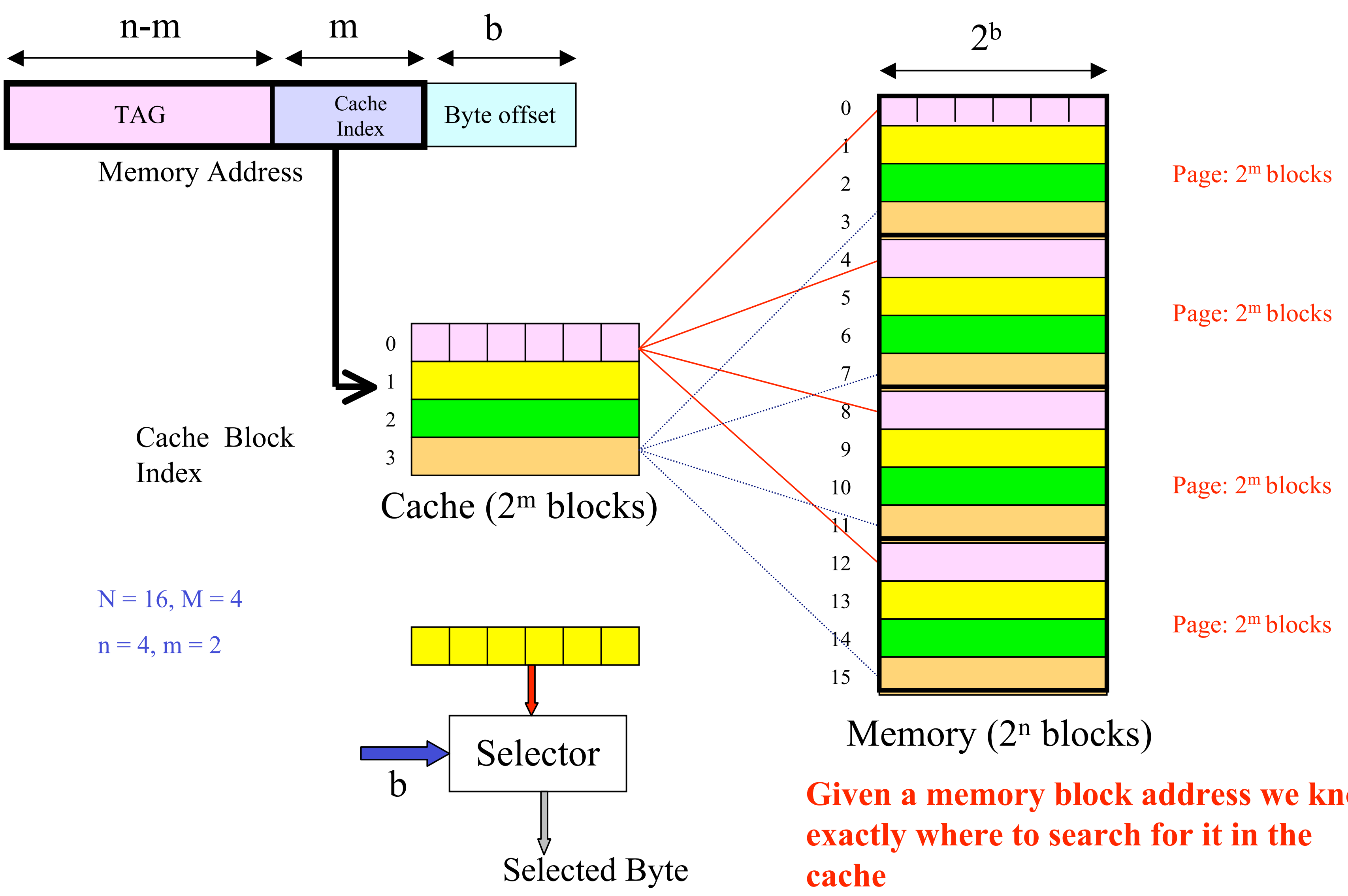
## Direct-Mapped Cache Organization



N = 16, M = 4

n = 4, m = 2

# Direct-Mapped Cache Organization



N = 16, M = 4

n = 4, m = 2

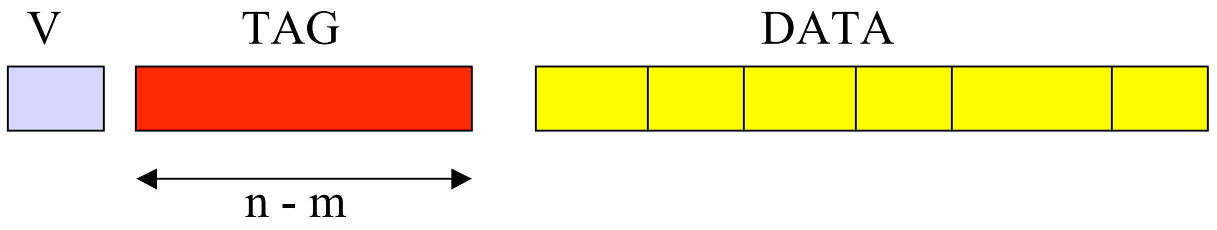
**Given a memory block address we know exactly where to search for it in the cache**

# Direct-Mapped Cache Organization

How does one identify which of the $2^{n-m}$ possible memory blocks is actually stored in a given cache block?

From which page does the block in that cache line come form?

Cache Line Entry:



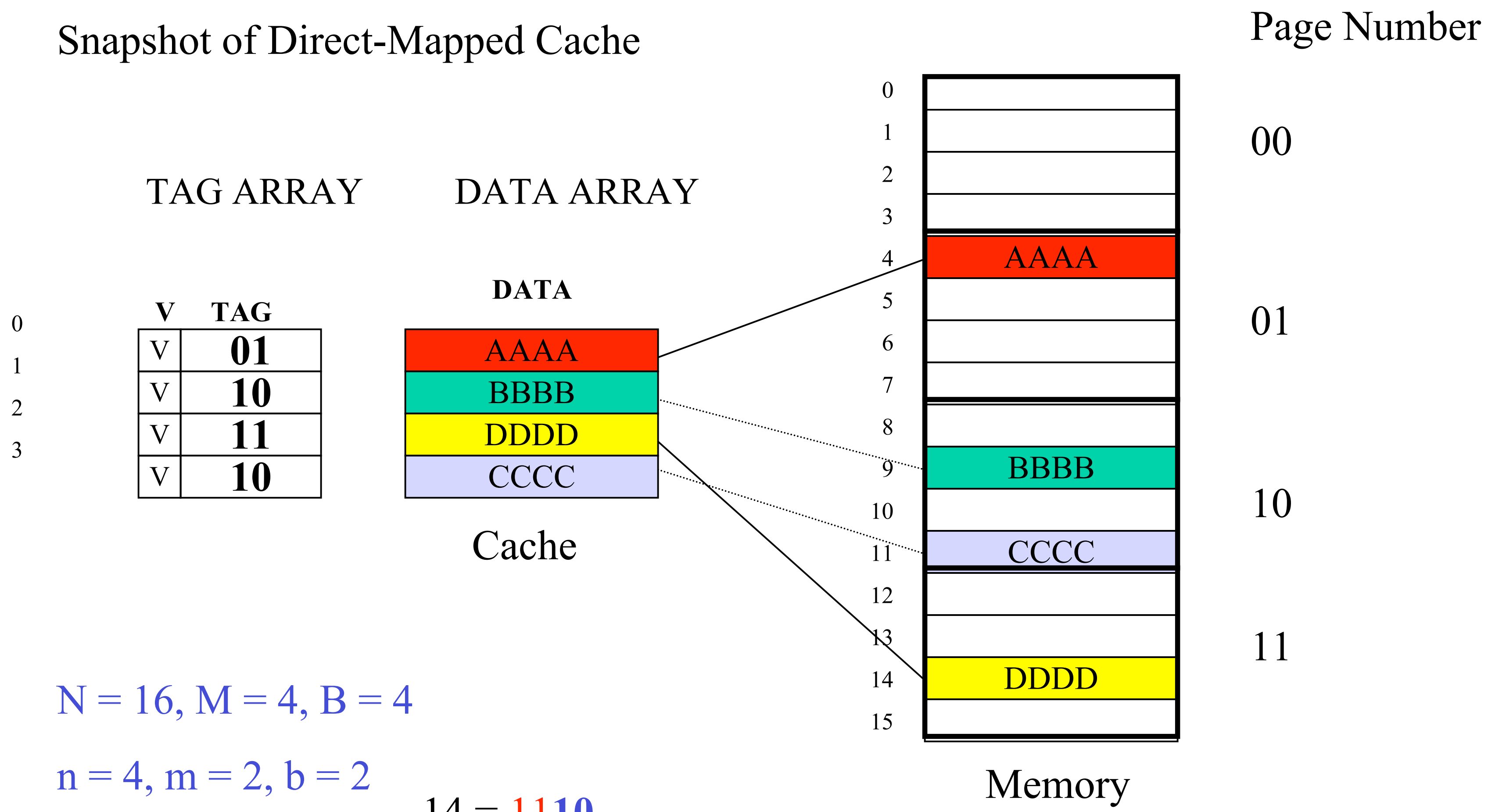
Maintain meta data (directory information) in the form of a TAG field with each cache line

TAG : identifies which of the $2^{n-m}$ memory blocks stored in cache block

V (Valid) bit: Indicates that the cache entry contains valid data

DATA : Copy of the memory block stored in this cache block

# Direct-Mapped Cache Organization



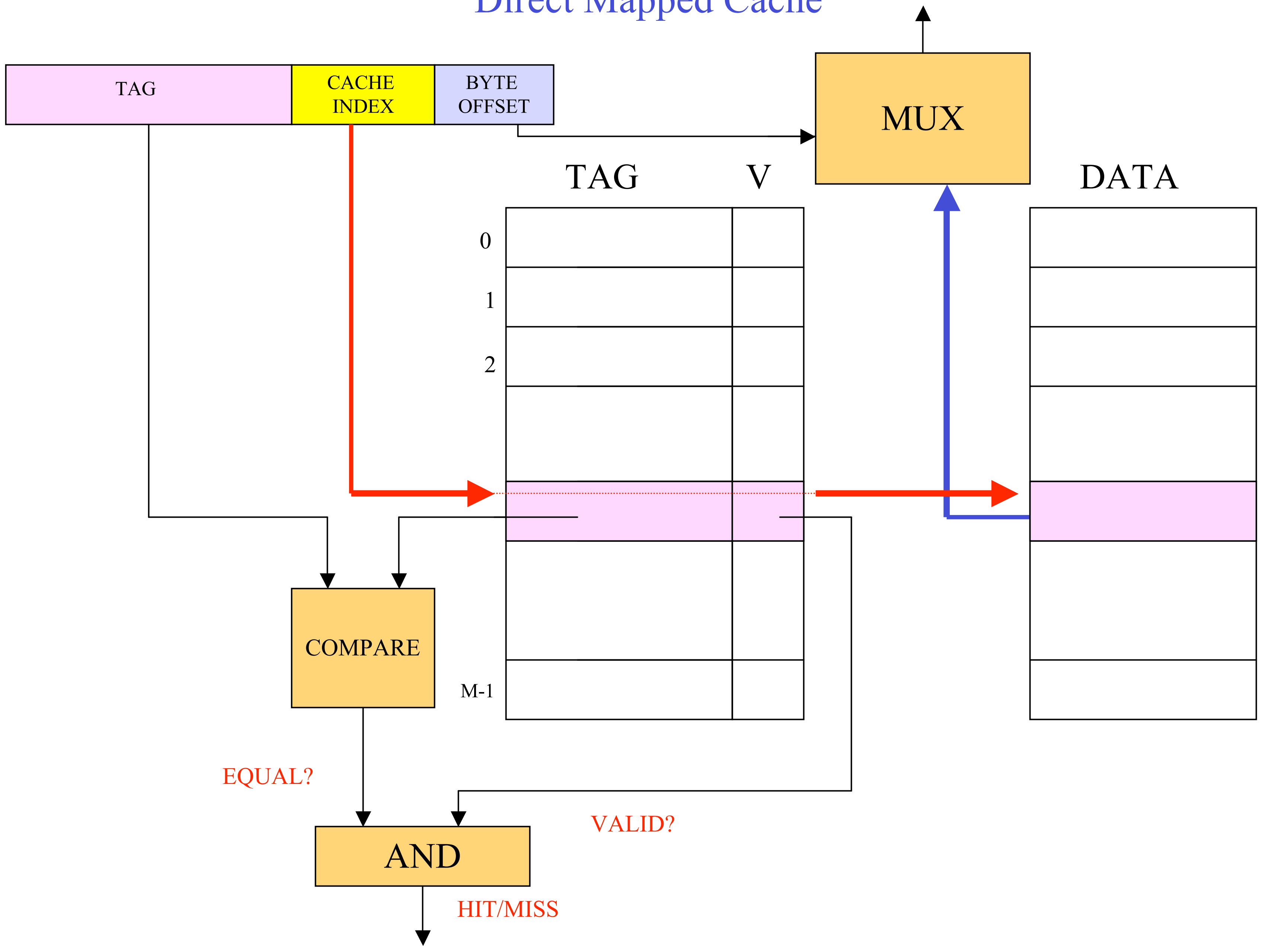
N = 16, M = 4, B = 4

n = 4, m = 2, b = 2

14 = 1110

**Index = 2, TAG = 3**

## Direct Mapped Cache



## Direct-Mapped Cache Summary

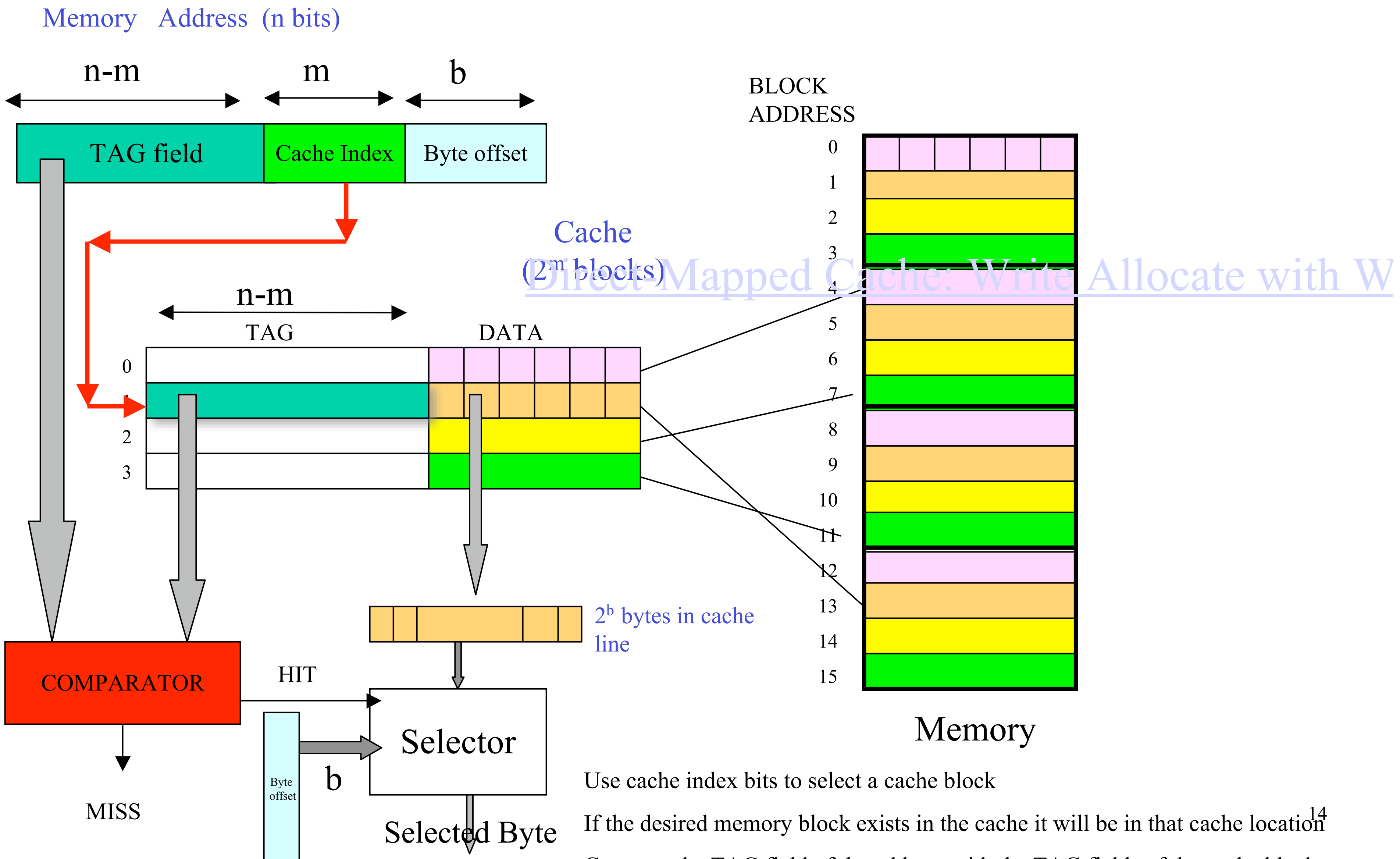
Each memory block has a unique location it can be present in the cache

Main memory size: $N = 2^n$ blocks. Block addresses: $0, 1, \ldots, 2^n - 1$

Cache size : $M = 2^m$ blocks. Block addresses: $0, 1, \ldots, 2^m - 1$

- Memory block with address $\mu$ is mapped to the *unique* cache block: $\mu \bmod M$
- Cache index = $\mu \bmod M$ computed as m LSBs of the binary representation of $\mu$
- The cache index is the address in the cache where a memory block is placed
- $2^{n-m}$ memory blocks (differing in the n-m MSBs) have the same cache index
- A cache block can hold any one of the $2^{n-m}$ memory blocks with the same cache index (i.e. that agree on the m LSBs)
- Disambiguation is done associatively
  - Each cache block has a TAG field of n-m bits
  - Tag holds the n-m MSBs of the memory block that is currently stored in that cache location

# Direct Mapped Cache Organization



Use cache index bits to select a cache block

If the desired memory block exists in the cache it will be in that cache location

Compare the TAG field of the address with the TAG fields of the cache block

## Direct Mapped Cache Operation

Memory Read Protocol Assume all memory references are reads

Input: n+b-bit memory word address $[x]_{n-m}\,[w]_m\,[d]_b$

Block Address $A = [x]_{n-m}\,[w]_m$

```
Compute cache index w = A mod M
Read block at cache[w] (both TAG and DATA fields)

if  (cache[w].V is TRUE  and cache[w].TAG = x)   /*  Cache Hit
      Select word[d]  from block cache[w].DATA and transfer to processor

else  /*  Cache Miss  */

      1.        Stall processor till block brought into cache
      2.        Read memory block at address A and load to cache[w].DATA
      3.        Update cache[w].TAG to x and cache[w].V to TRUE
      4.        Restart processor from start of cycle
```

## Direct-Mapped Cache Replacement

### Replacement Strategy

- No choice in replacements for direct-mapped cache
- The current block at cache[w] is replaced by the new reference that maps to cache[w].

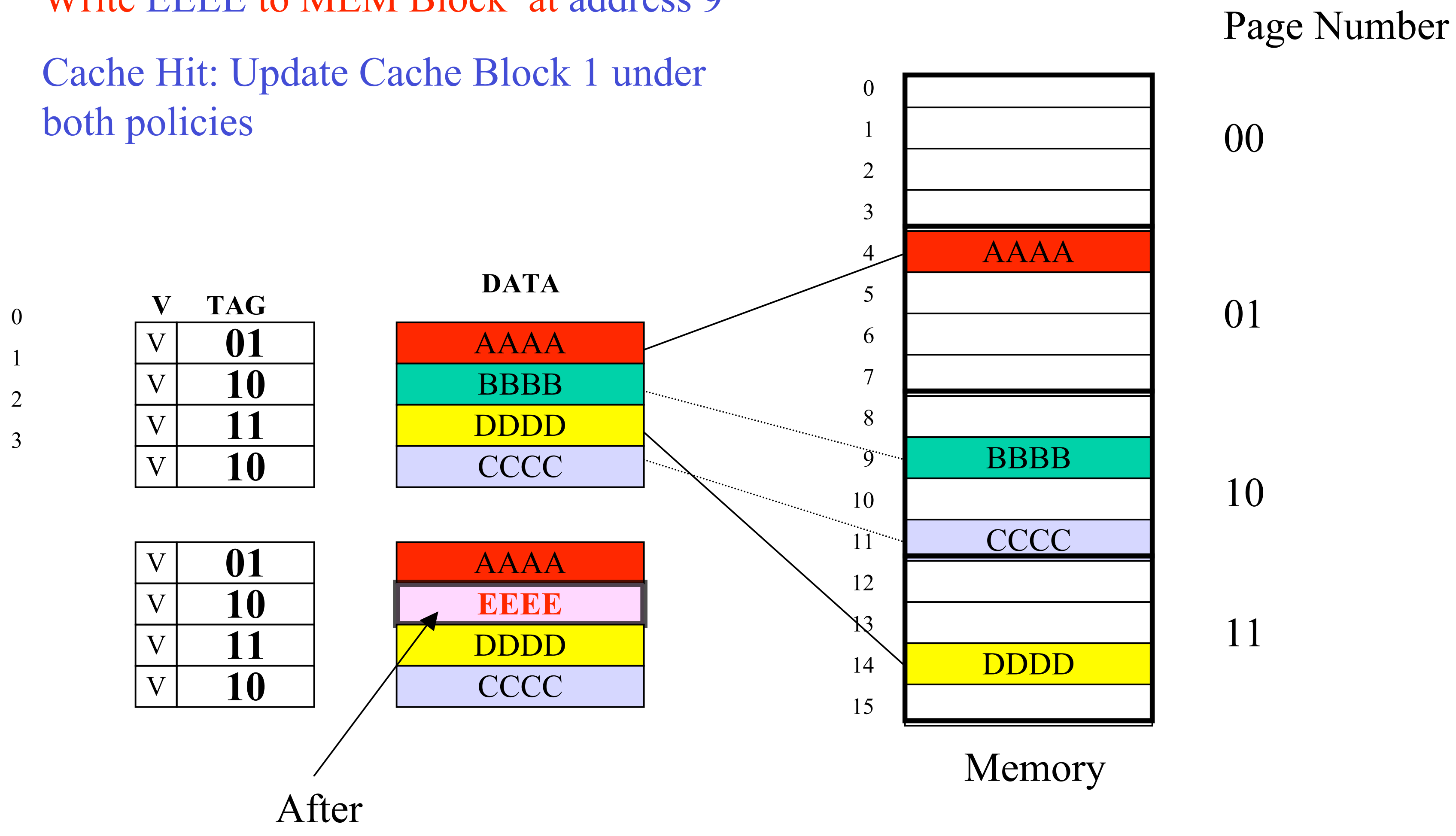
### Handling Writes

1. Write Allocate: Treat  a write to a word that is not in the cache as a cache miss. Read the missing block into cache and update it.
2. No Write Allocate: A write to a word that is not in the cache updates only main memory without disturbing the cache.

## Write Allocate and No Allocate Policies

Write EEEE to MEM Block at address 9

Cache Hit: Update Cache Block 1 under both policies



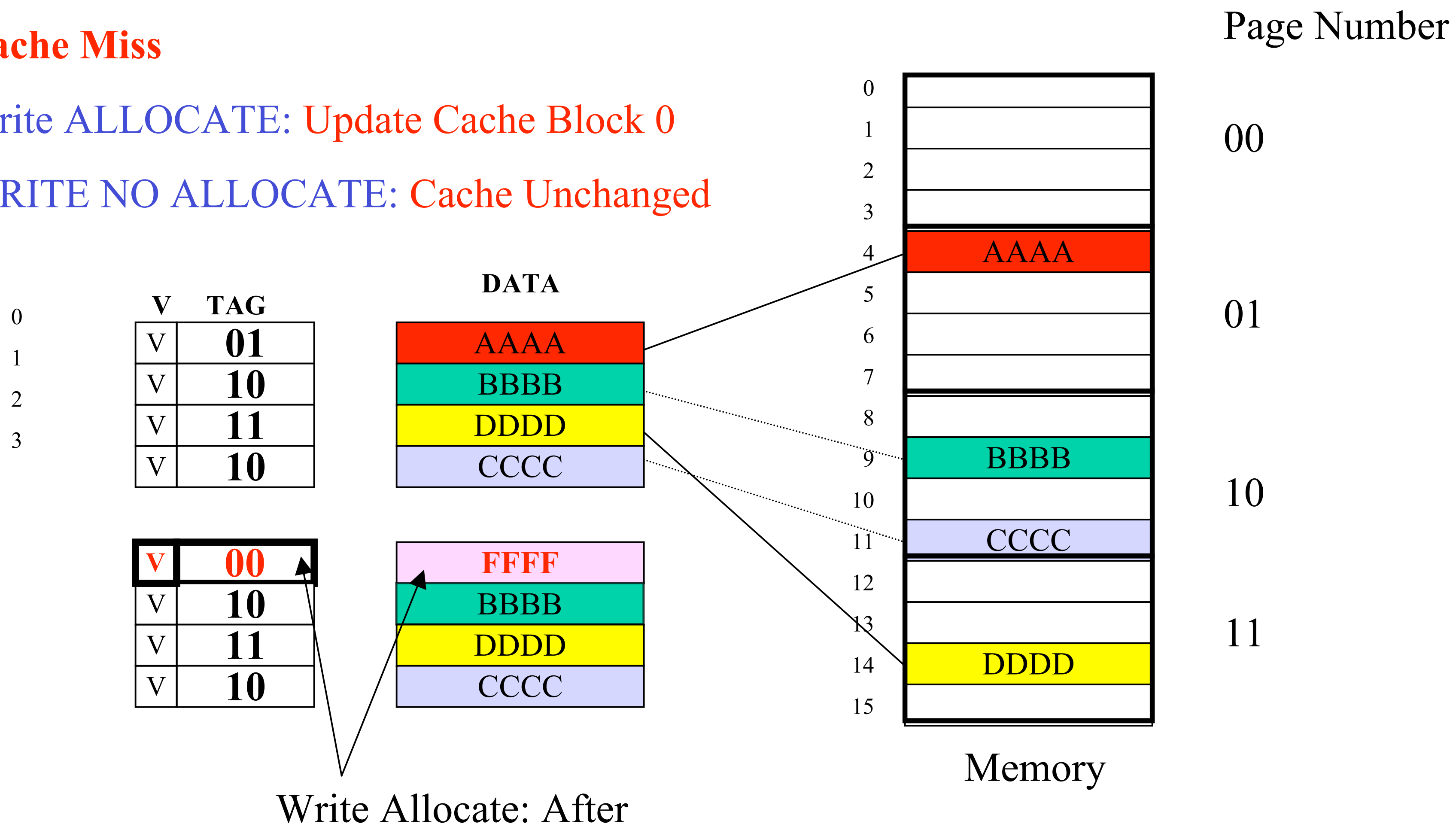
## Write Allocate and No Allocate Policies

Write FFFF to MEM Block at address 0

**Cache Miss**

Write ALLOCATE: Update Cache Block 0

WRITE NO ALLOCATE: Cache Unchanged



## Write Through and Write Back Policies

### Handling Writes

1. Write Through: A write updates both main memory and cache locations for the block (eager write)

2. Write Back: A write updates only the cache location; main memory is updated only when the corresponding cache block is replaced (lazy update)

## Write Back Policy

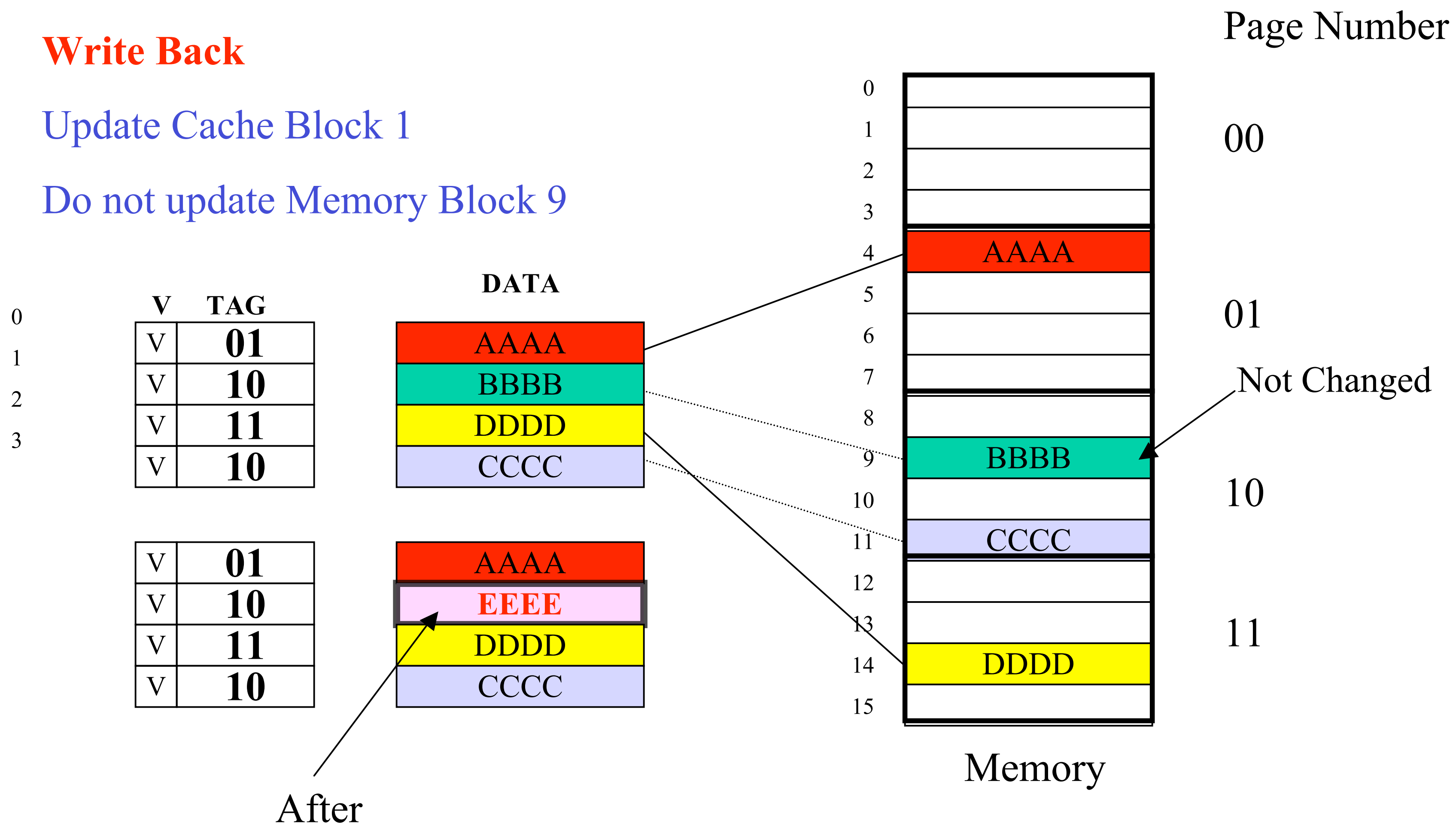
Write EEEE to MEM Block at address 9

**Cache Hit**

**Write Back**

Update Cache Block 1

Do not update Memory Block 9



# Write Through Policy

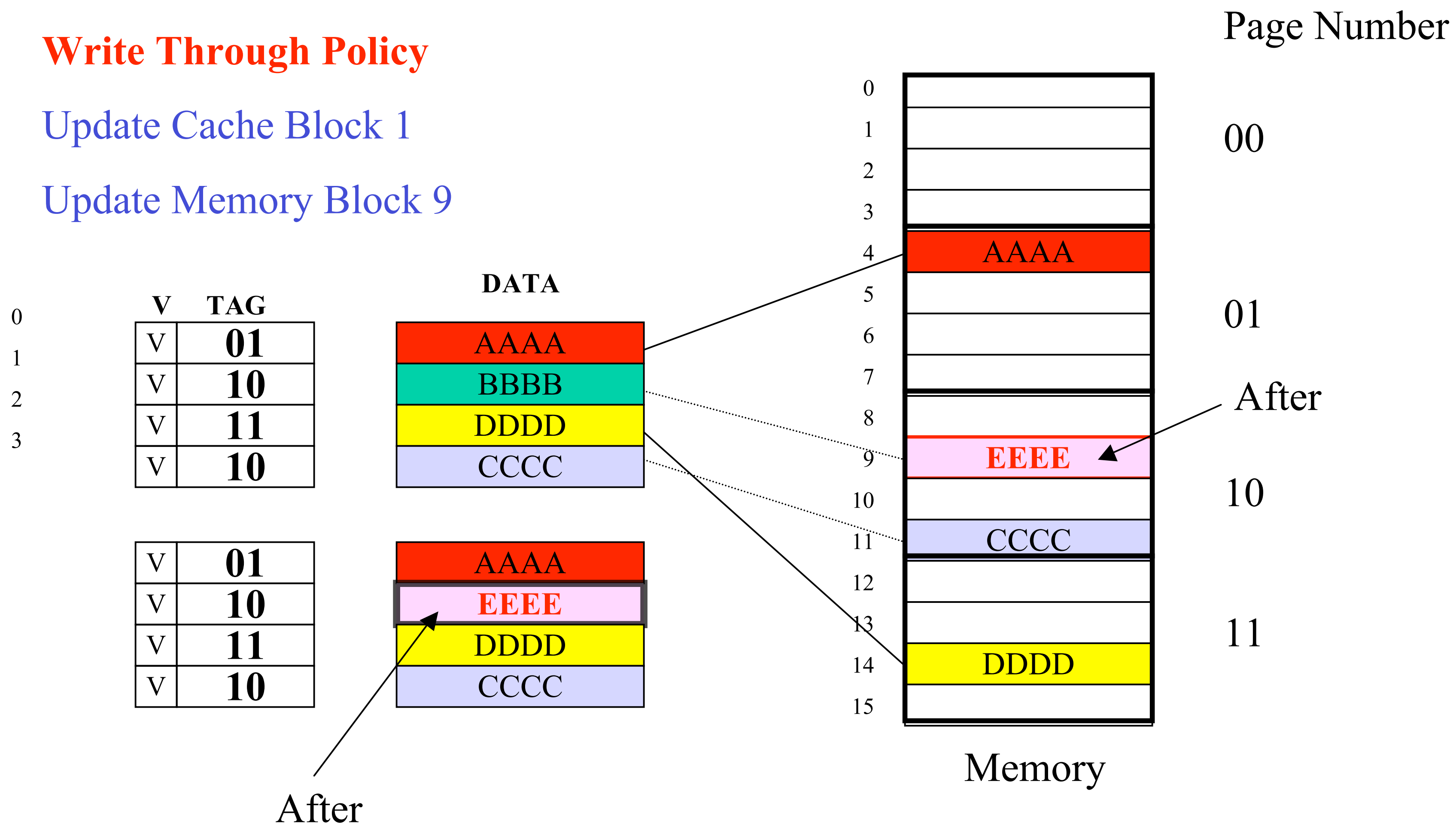
Write EEEE to MEM Block at address 9

**Cache Hit**

**Write Through Policy**

Update Cache Block 1

Update Memory Block 9

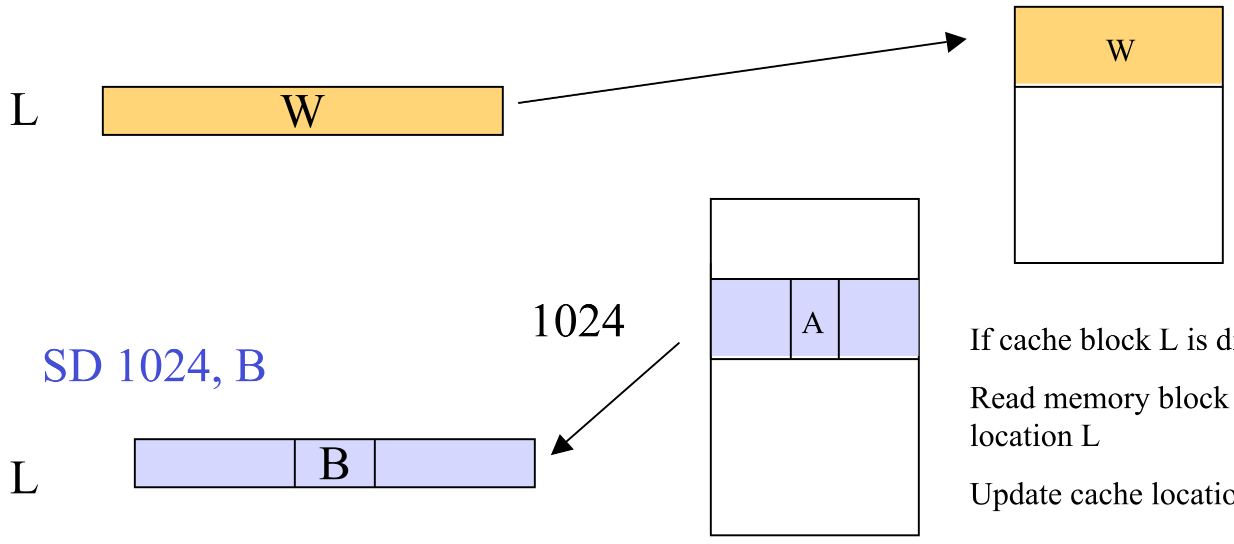


## Direct-Mapped Cache: Write Allocate with Write-Back

### Write Allocate and Write-Back Protocol

- On a write only cache block is written with updated value
- Memory is updated (write back) only when cache block is replaced
- Main memory and cache are inconsistent till write-back

- Additional bit (D) in cache entry: Dirty/Clean Bit
  - Set to TRUE when that cache entry is updated
- Replaced block needs to be written to memory only if its D bit is TRUE



If cache block L is dirty write it to memory

Read memory block that includes address 1024 into cache location L

Update cache locations corresponding to 1024 to B

## Direct-Mapped Cache: Write Allocate with Write-Back

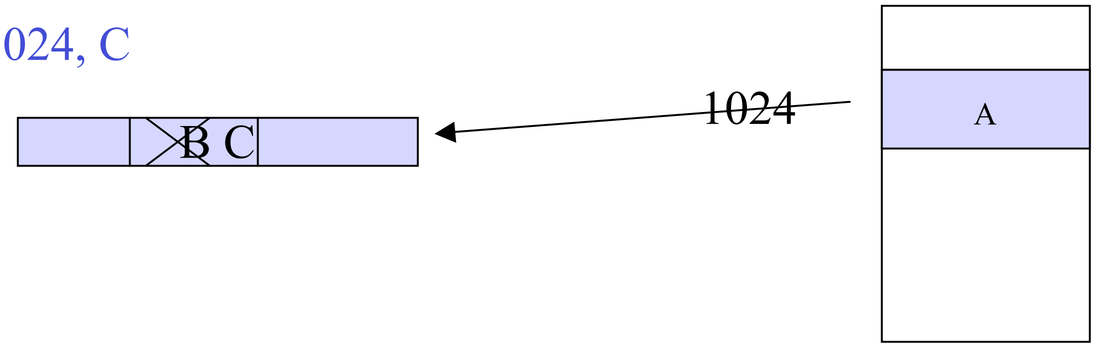
Write Allocate and Write-Back Protocol

- On a write only cache block is written with updated value
- Memory is updated (write back) only when cache block is replaced
- Main memory and cache are inconsistent till write-back

- Additional bit (D) in cache entry: Dirty/Clean Bit
  - Set to TRUE when that cache entry is updated
- Replaced block needs to be written to memory only if its D bit is TRUE

SD 1024, B

SD 1024, C

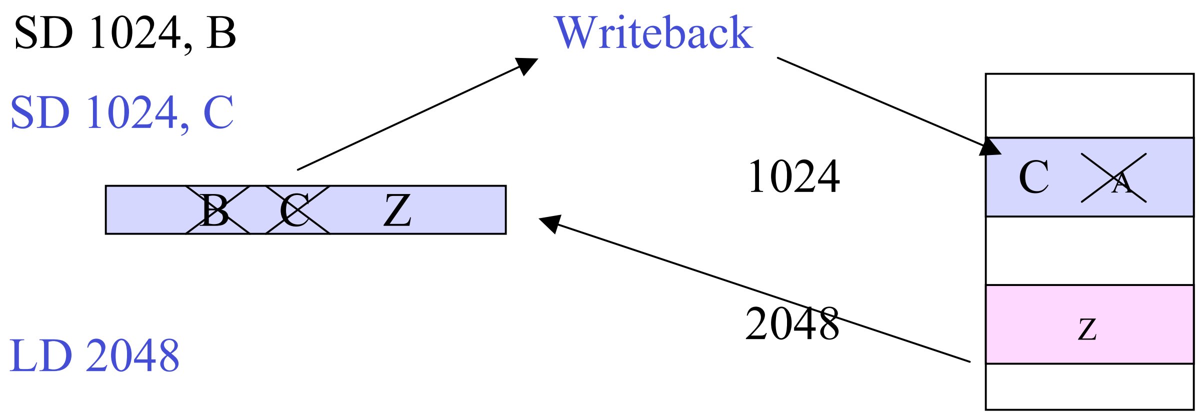


## Direct-Mapped Cache: Write Allocate with Write-Back

### Write Allocate and Write-Back Protocol

- On a write only cache block is written with updated value
- Memory is updated (write back) only when cache block is replaced
- Main memory and cache are inconsistent till write-back

- Additional bit (D) in cache entry: Dirty/Clean Bit
    - Set to TRUE when that cache entry is updated
- Replaced block needs to be written to memory only if its D bit is TRUE



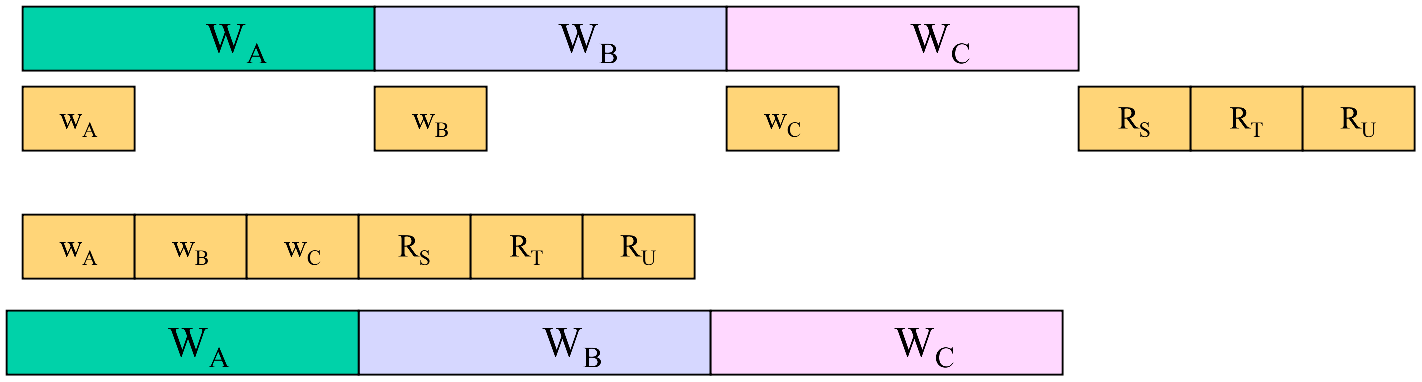
## Direct-Mapped Cache: Write Allocate with Write-Through

Write Allocate and Write-Through Protocol**:** write data to address $[x]_{n-m}\,[w]_m[d]_b$

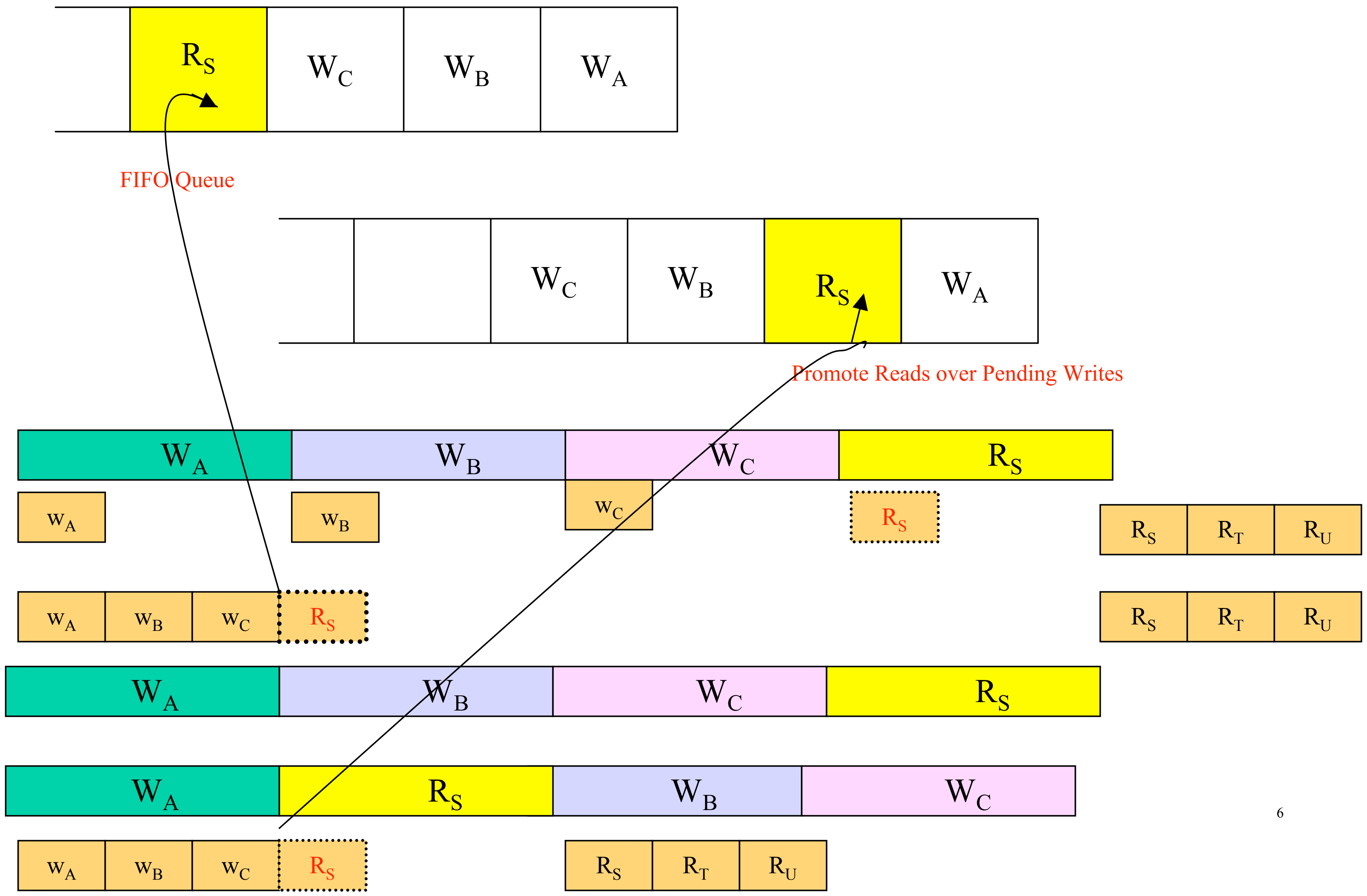
Block Address $A = [x]_{n-m}\,[w]_m$

- Synchronous Writes
- Writes proceed at the speed of main memory not at speed of cache

$W_A \quad W_B \quad W_C \quad R_P \quad R_S \quad R_T \quad R_U$



## Direct-Mapped Cache: Write Allocate with Write-Through



## Direct-Mapped Cache: Write Allocate with Write-Through

Write Allocate and Write-Through Protocol**:** write data to address $[x]_{n-m} [w]_m [d]_b$

Block Address $A = [x]_{n-m} [w]_m$

- Writes proceed at the speed of main memory not at speed of cache
- To speed up writes use asynchronous writes:
  - Write into cache and simultaneously into a write buffer
  - Execution continues concurrently with memory write from buffer
  - Write buffer should be deep enough to buffer burst of writes
  - If write buffer full on write then stall processor till buffer frees up
  - Write buffer served in FCFS order : simple protocol
  - Allow (later) reads to overtake pending writes
    - Read protocol modified appropriately (Can it happen?)
      - On memory read check write buffer for a write in transit

## Writes Summary

1. In a write allocate scheme with a write through policy:
   - Write Hit: Update both cache and main memory (1W)
   - Write Miss: Read in block to cache. Update cache and main memory (1R + 1W)

2. In a write allocate scheme with a write back policy:
   - Write Hit: Update cache only
   - Write Miss: Read in block to cache. Write evicted block if dirty. Update cache. (1R + 1W if dirty block being replaced)

3. In a no write allocate scheme with a write through policy:
   - Write Hit: Update both cache and main memory (1W)
   - Write Miss: Update main memory only (1W)

4. In a no write allocate scheme with a write back policy:
   - Write Hit: Update cache only
   - Write Miss: Update main memory only (1W)

## Direct-Mapped Cache: Write Allocate with Write-Through Protocol

WRITE data to address $[x]_{n-m}\,[w]_m[d]_b$

Block Address $A = [x]_{n-m}\,[w]_m$

```
Compute cache index w = A mod M

 if (Cache Hit)
      1.     Write data  into word d of cache[w].DATA
      2.     Store data into memory address [x]_{n-m} [w]_m[d]_b

if (Cache Miss)

      1.   Load block at memory block address A into cache[w].DATA
      2.   Update cache[w].TAG to x   ;cache[w].V = TRUE
      3.   Retry cache access
```

READ from address $[x]_{n-m}\,[w]_m[d]_b$

Cache Hit: Replace step 1 with Read word from the cache line and omit step 2

## Direct-Mapped Cache: Write Allocate and Write Back

Write Allocate and Write-Back Protocol : write data to address $[x]_{n-m}\,[w]_m\,[d]_b$

Block Address $A = [x]_{n-m}\,[w]_m$

If cache hit update DATA and D fields of cache entry

If cache miss

replace current block writing it to main memory if dirty

update cache block with new data and V, D, TAG fields

```
Compute cache index w = A mod M
if  Cache Hit
        Write data  into cache[w].DATA
        Set cache[w].D to  TRUE
else  /* Cache Miss */
        Stall Processor
        if   cache block is dirty          /* cache[w].D = TRUE */
         Store cache[w].DATA into memory block at address [TAG][w]
        Load memory block at address [x][w]
        Update cache[w].TAG to x, cache[w].V = TRUE and cache[w].D to  FALSE
                Retry cache Access
```

## Direct-Mapped Cache: Reads in a Write Back Cache

Write-Back Protocol : read address $[x]_{n-m}$ $[w]_m$ $[d]_b$

If cache hit read data field of cache entry

If cache miss

replace current block writing it to memory if dirty

read in new block from memory and install in cache

```
Compute cache index w = A mod M
if  Cache Hit
            Read block cache[w].DATA; select word d of block
else   /* Cache Miss */
            Stall processor
            if   cache block is dirty   /* cache[w].D = TRUE  */
                        Store cache[w].DATA into memory at address [TAG][w]
            Read block at memory address A into cache[w].DATA
            Update cache[w].TAG to x, cache[w].V to TRUE, cache[w].D to FALSE
            Retry cache access
```