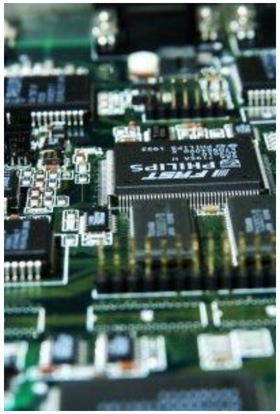
CSE 321a

Computer Organization (1)

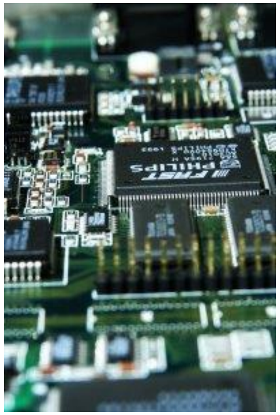
تنظيم الحاسبات (1)



3rd year, Computer Engineering

Fall 2016

Lecture #5



Dr. Hazem Ibrahim Shehata

Dept. of Computer & Systems Engineering

Credits to Dr. Ahmed Abdul-Monem Ahmed for the slides

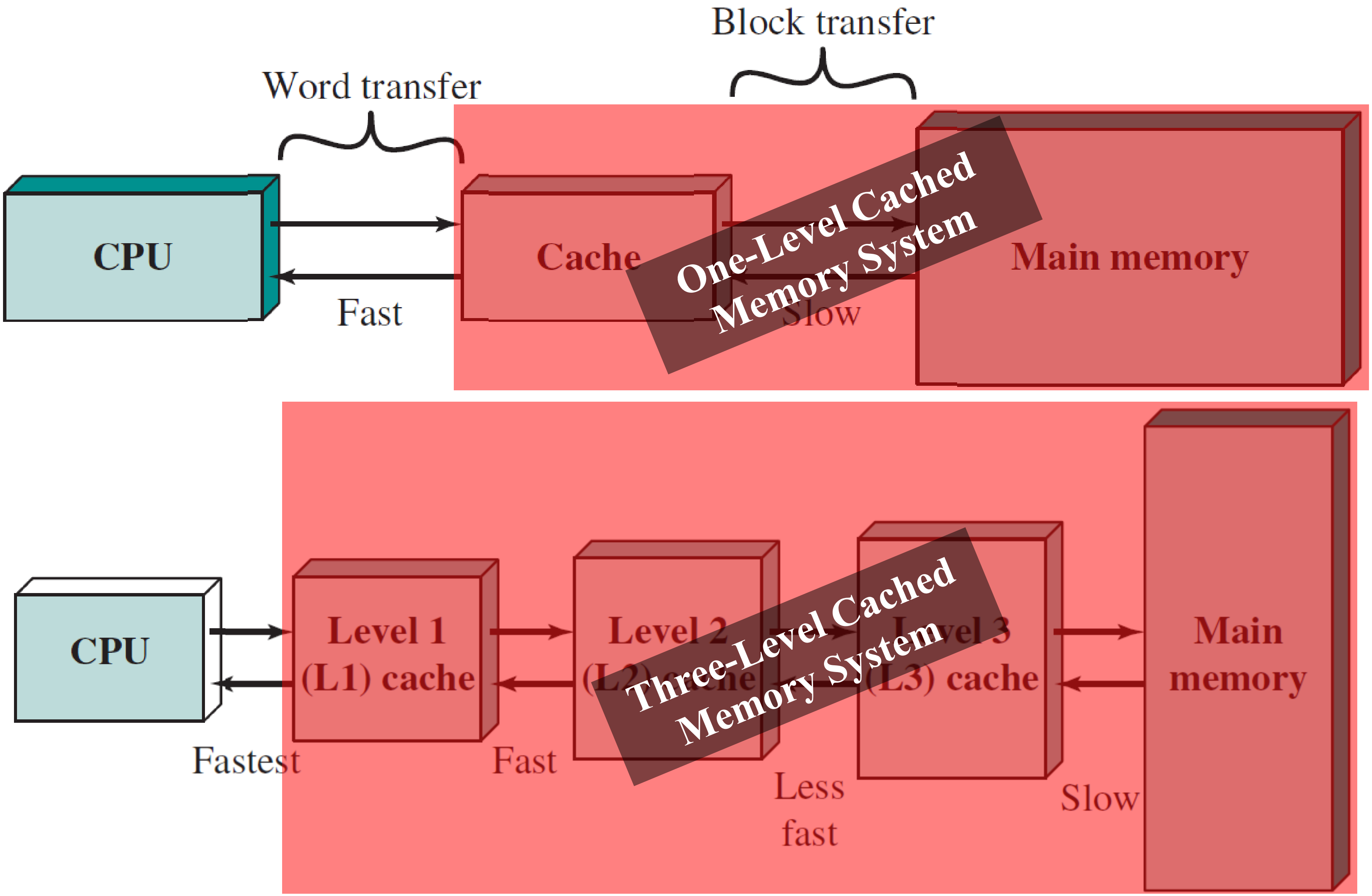
Administrivia

- Lecture:
  - Will get back to regular schedule starting next week!
  - Day/Time: Tuesday, 10:15am – 12:45pm
- Assignment #1:
  - Solution will posted on Thursday.

Website: http://hshehata.github.io/courses/zu/cse321a
Office hours: Sunday 12:00pm-1:00pm

Chapter 4. Cache Memory (*cont.*)

Cache Memory - Concept



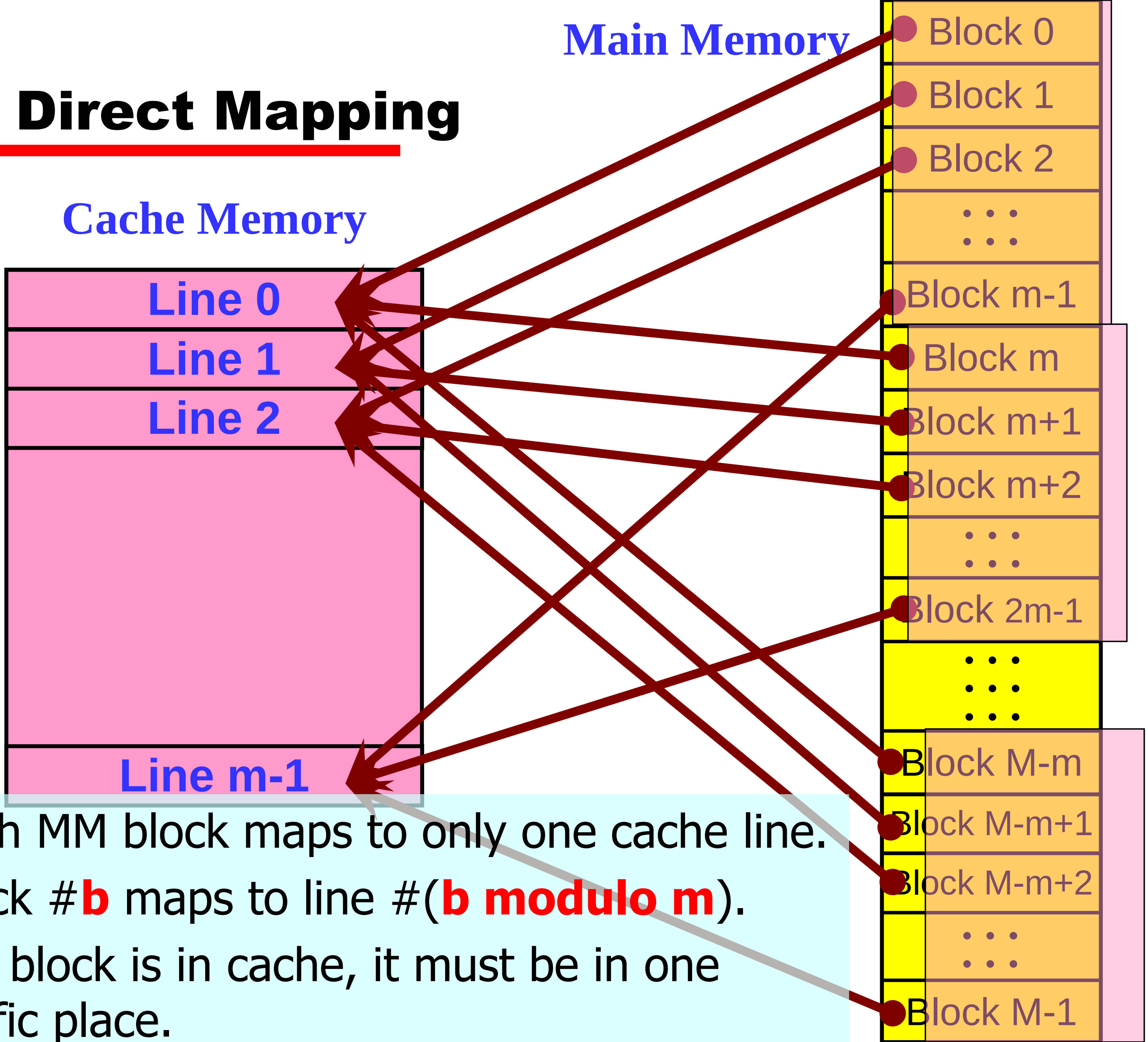
Cache Memory – Design

1. Mapping function
2. Replacement algorithm
3. Read/Write policies
4. Number of caches
5. Addresses
6. Size
7. Block/line size

Cache Memory – Design

1. Mapping function
2. Replacement algorithm
3. Read/Write policies
4. Number of caches
5. Addresses
6. Size
7. Block/line size

(I) Direct Mapping



- Each MM block maps to only one cache line.
- Block #**b** maps to line #(**b modulo m**).
- If a block is in cache, it must be in one specific place.

(I) Direct Mapping
Cache Line Table

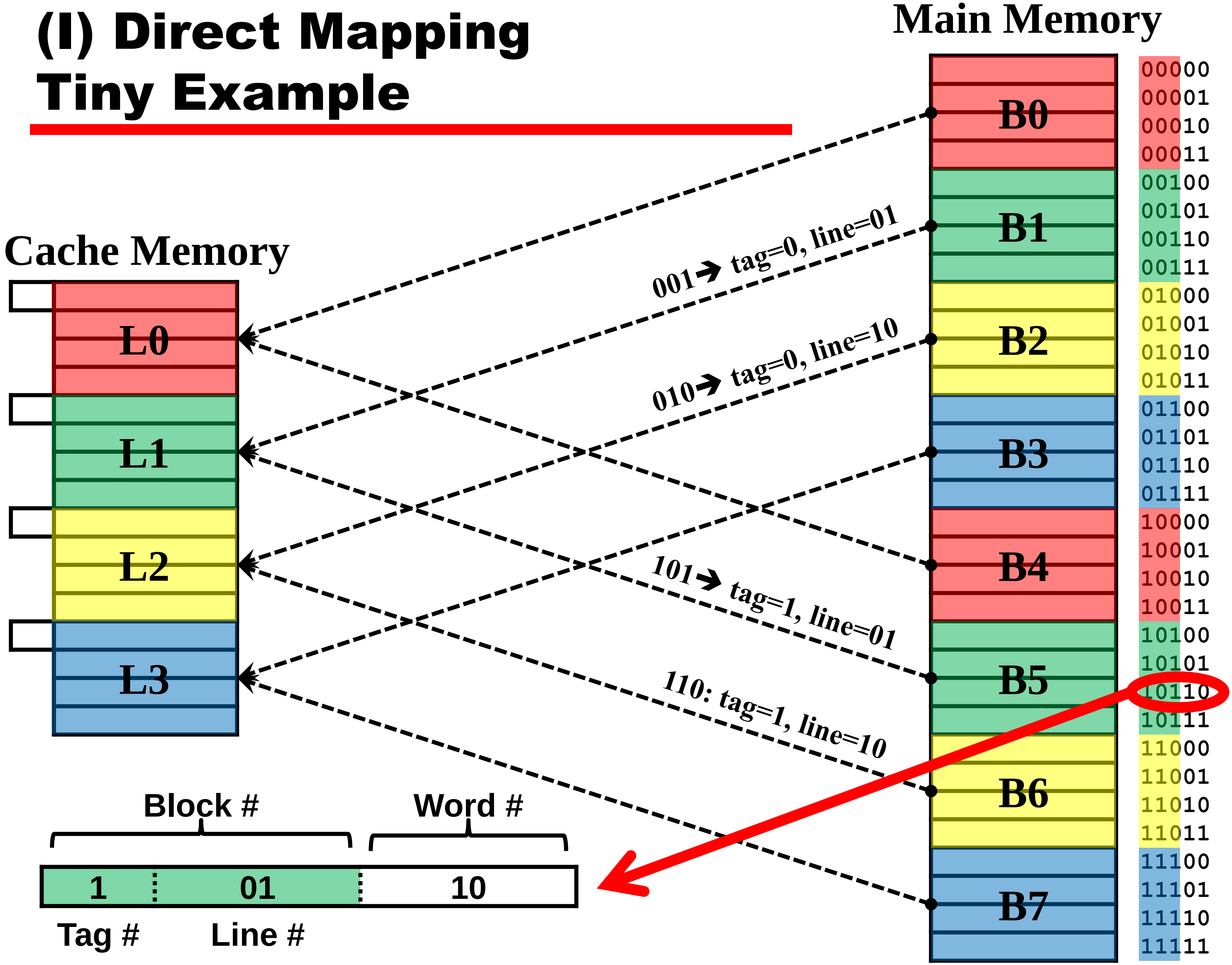
- Number of MM blocks = M
- Number of cache lines = m

Cache line # (b modulo m)	MM block # (b)
0	← 0, m, 2m, 3m, …, M-m
1	← 1,m+1, 2m+1, …, M-m+1
2	← 2,m+2, 2m+2, …, M-m+2
……	← ………………………………
m-1	← m-1, 2m-1, 3m-1, …, M-1

(I) Direct Mapping
Tiny Example

- Main memory:
  - 1 Location ≡ 1 word ≡ 1 byte
  - 32 byte
    - Length of address = $\log_2 32 = 5$
  - 4-byte **blocks**
    - # of blocks (M) = 32 / 4 = 8
- Cache:
  - 16 byte
  - 4-bytes **lines**
    - # of lines (m) = 16 / 4 = 4

(I) Direct Mapping
Tiny Example



(I) Direct Mapping
Address Format

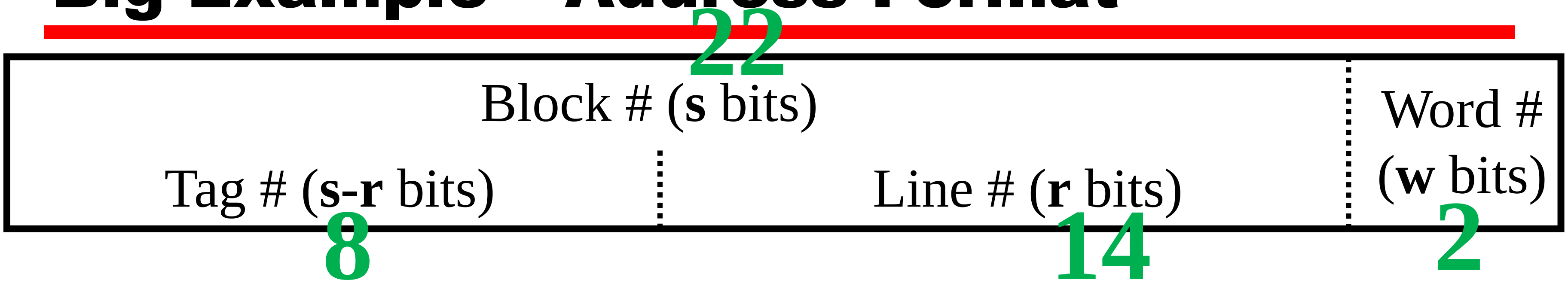
- Memory address is split (based on block size) into:
  1. Least significant **w** bits → identify **word** in block.
     - w = $\log_2$ (# of words in block).
  2. Most significant **s** bits → identify **block** in MM.
     - s = $\log_2$ (# of blocks in MM).
     - The **s** bits are split (based on cache size) into:
       1. Least significant **r** bits → identify cache **line**.
          - r = $\log_2$ (# of lines in cache).
       2. Most significant **s – r** bits → **tag** (identify group).
- In the "tiny example": w=2, s=3, r=2, s-r=1.

<table>
<tr><td colspan="2">Block # (<b>s</b> bits)</td><td rowspan="2">Word # (<b>w</b> bits)</td></tr>
<tr><td>Tag # (<b>s-r</b> bits)</td><td>Line # (<b>r</b> bits)</td></tr>
</table>

(I) Direct Mapping
Big Example

- Main memory:
  - 1 Location ≡ 1 word ≡ 1 byte
  - 16M byte
    - Length of address = $\log_2$ 16M = 24
  - 4-byte **blocks**
    - # of blocks (M) = 16M / 4 = 4M
- Cache:
  - 64K byte
  - 4-bytes **lines**
    - # of lines (m) = 64K / 4 = 16K

(I) Direct Mapping
Big Example – Address Format



22

Block # (**s** bits)

Word # (**w** bits)

Tag # (**s**-**r** bits)

Line # (**r** bits)

8

14

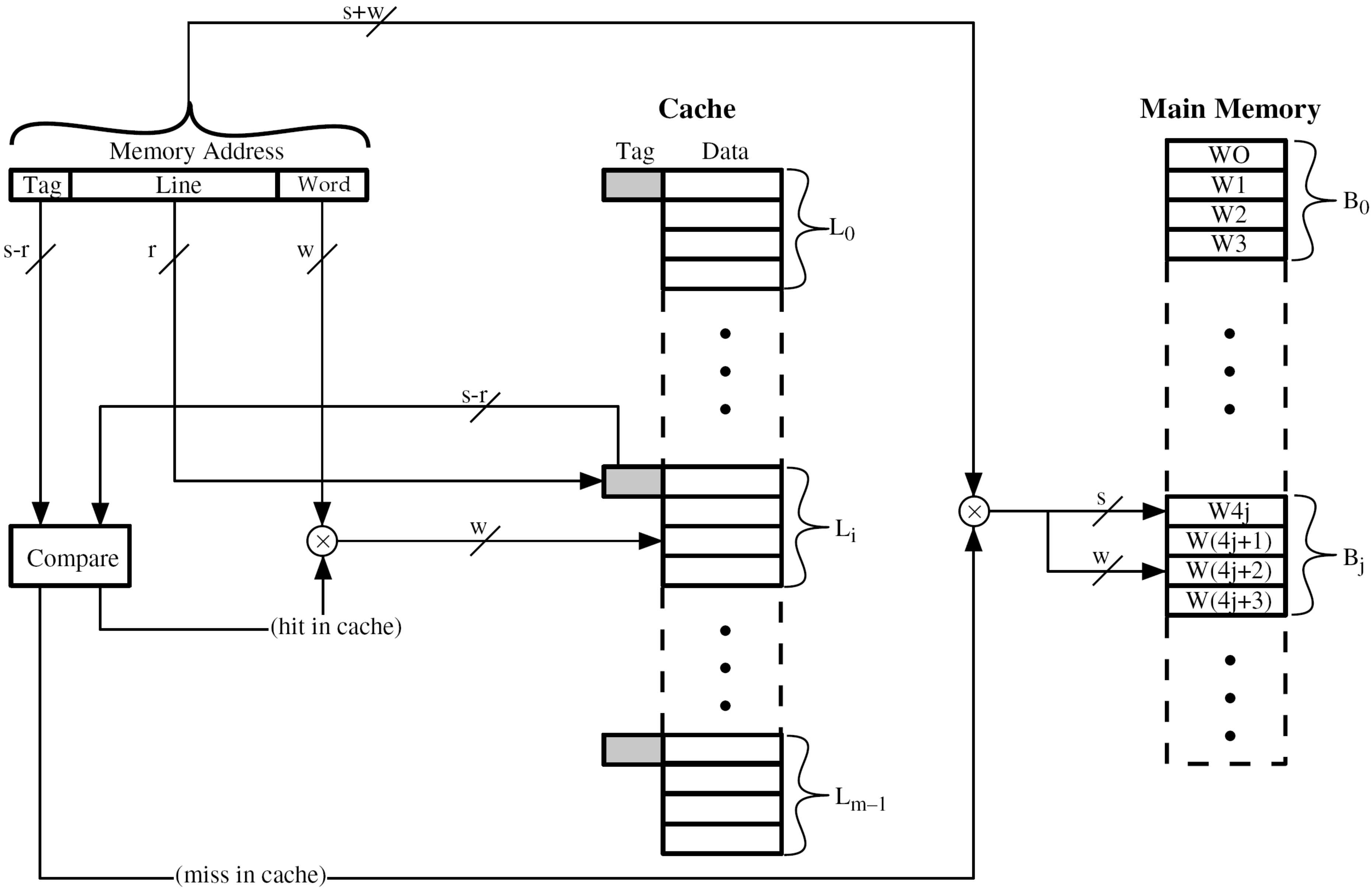
2

- 24-bit address
  - s + w = (s-r) + r + w = 24
- 4-byte block ➔ 4M-block MM
  - w = $\log_2 4$ = 2 bits
  - s = 24 – w = 22 bits (Another way: s = $\log_2$ 4M = 22 bits)
- 64K-byte Cache ➔ 16K-line Cache
  - r = $\log_2$ 16K = 14 bits
  - s-r = 22 – 14 = 8 bits
- Notes:
  - No 2 blocks that map to the same line have the same tag field
  - Check contents of cache by finding line and checking tag

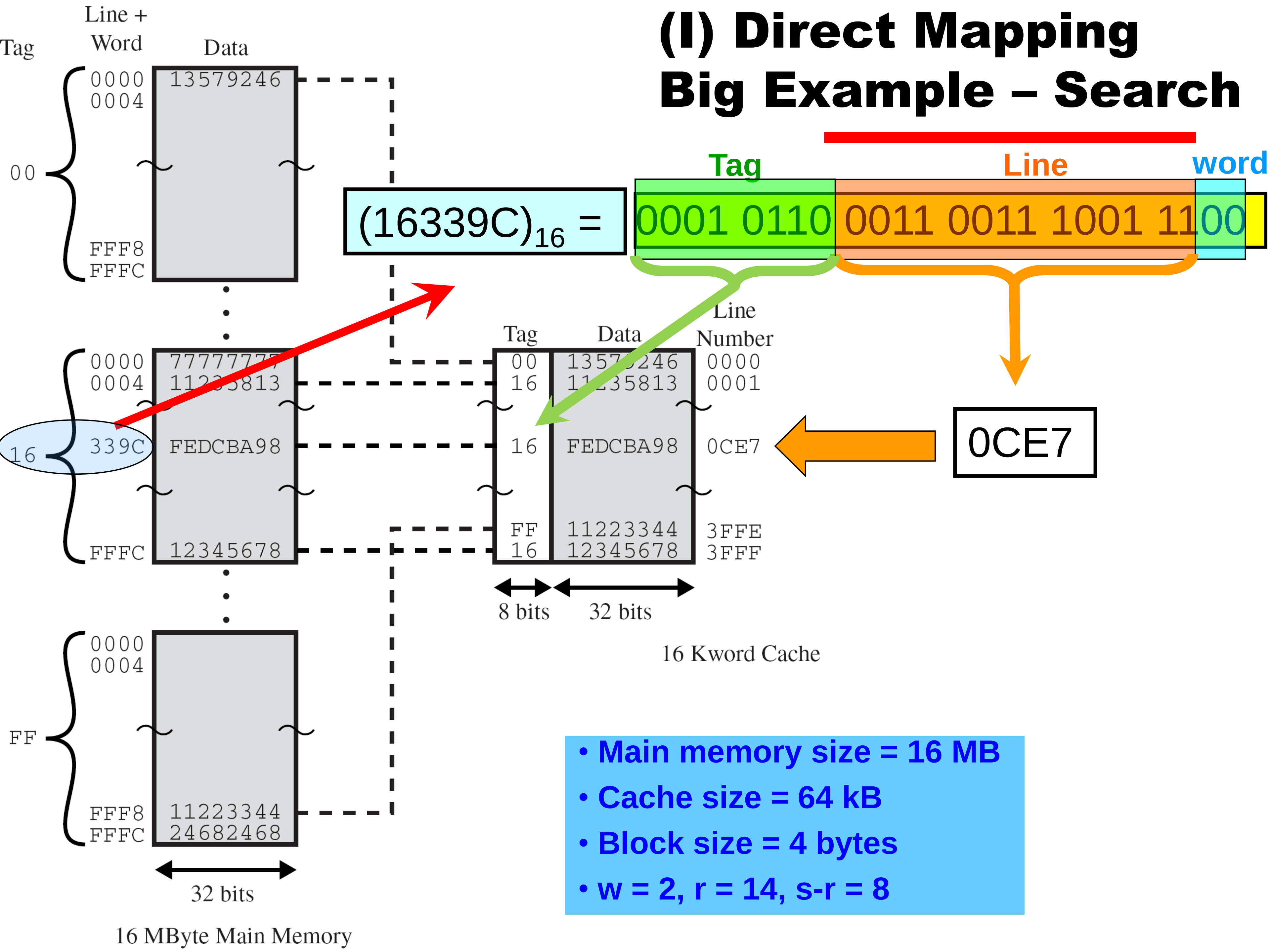
(I) Direct Mapping Address Format Summary

- Address length = (s + w) bits.
- Number of addressable units = 2^{s+w} words.
- Block size = line size = 2^{w} words.
  - w = $\log_2$ (# of words in block).
- Number of blocks in MM = M = $2^{s+w}/2^{w} = 2^{s}$.
  - s = $\log_2$ (# of words in MM / # of words in block)
- Number of lines in cache = m = 2^{r}.
  - r = $\log_2$ (# of words in cache / # of words in line).
- Size of tag = (s – r) bits.
  - (s-r) = $\log_2$ (# of words in MM / # of words in cache)

(I) Direct Mapping Cache Organization



(I) Direct Mapping Big Example – Search



- **Main memory size = 16 MB**
- **Cache size = 64 kB**
- **Block size = 4 bytes**
- **w = 2, r = 14, s-r = 8**

(I) Direct Mapping
Pros & Cons

- Simple.
- Inexpensive.
- Fixed location for given block.
  - If a program accesses 2 blocks that map to the same line repeatedly, cache misses are very high.

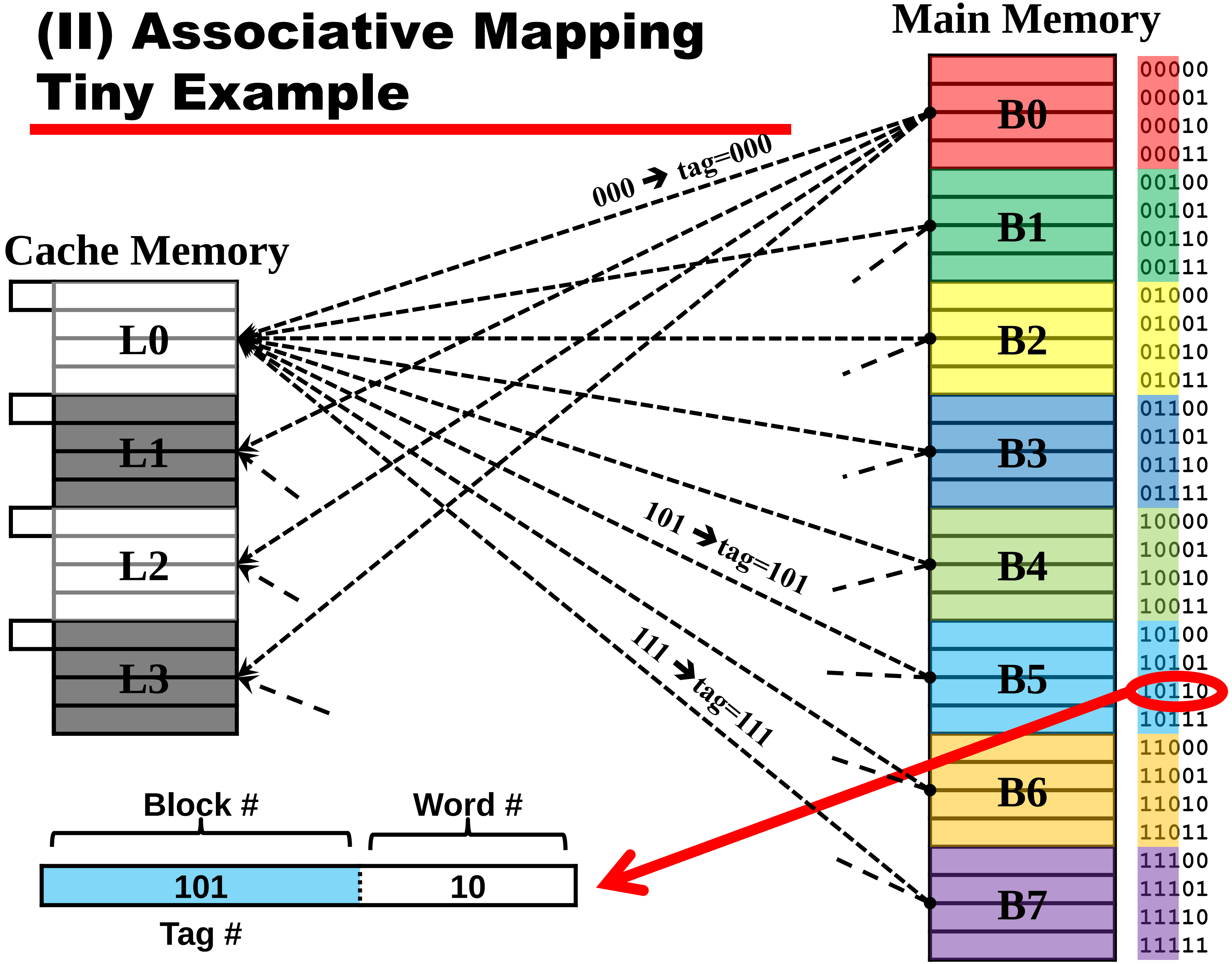
(II) (Fully-)Associative Mapping

- A main memory block can load into any line of cache.
- Memory address is interpreted as tag and word.
- Tag uniquely identifies block of memory.
- Every line’s tag is examined for a match.
- Cache searching gets expensive.

(II) Associative Mapping Tiny Example

- Main memory:
  - 1 Location ≡ 1 word ≡ 1 byte
  - 32 byte
    - Length of address = $\log_2 32 = 5$
  - 4-byte **blocks**
    - # of blocks (M) = 32 / 4 = 8
- Cache:
  - 16 byte
  - 4-bytes **lines**
    - # of lines (m) = 16 / 4 = 4

(II) Associative Mapping
Tiny Example



(II) Associative Mapping Address Format

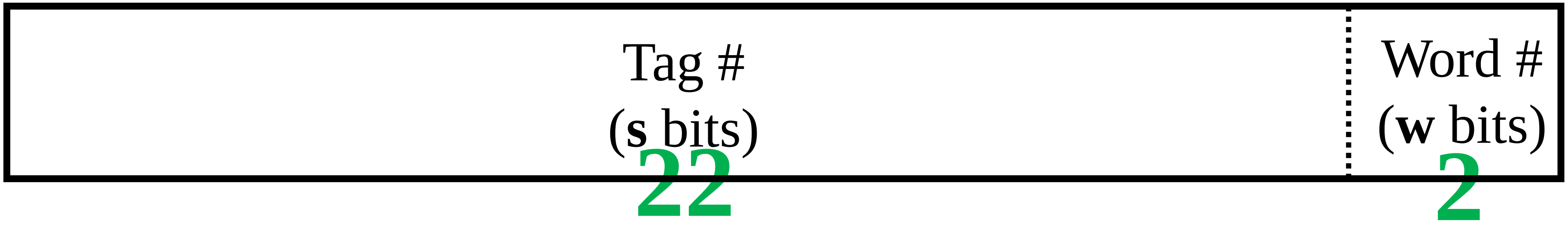
- Memory address is split (based on block size) into:
  1. Least significant **w** bits → identify **word** in block.
     - w = $\log_2$ (# of words in block).
  2. Most significant **s** bits → identify **block** in MM, and used as a **tag**.
     - s = $\log_2$ (# of blocks in MM).
- In the "tiny example": w=2, s=3.

Tag # (**s** bits)	Word # (**w** bits)

(II) Associative Mapping Big Example

- Main memory:
  - 1 Location ≡ 1 word ≡ 1 byte
  - 16M byte
    - Length of address = $\log_2$ 16M = 24
  - 4-byte **blocks**
    - # of blocks (M) = 16M / 4 = 4M
- Cache:
  - 64K byte
  - 4-bytes **lines**
    - # of lines (m) = 64K / 4 = 16K

(II) Associative Mapping
Big Example – Address Format



Tag #
(**s** bits)
22

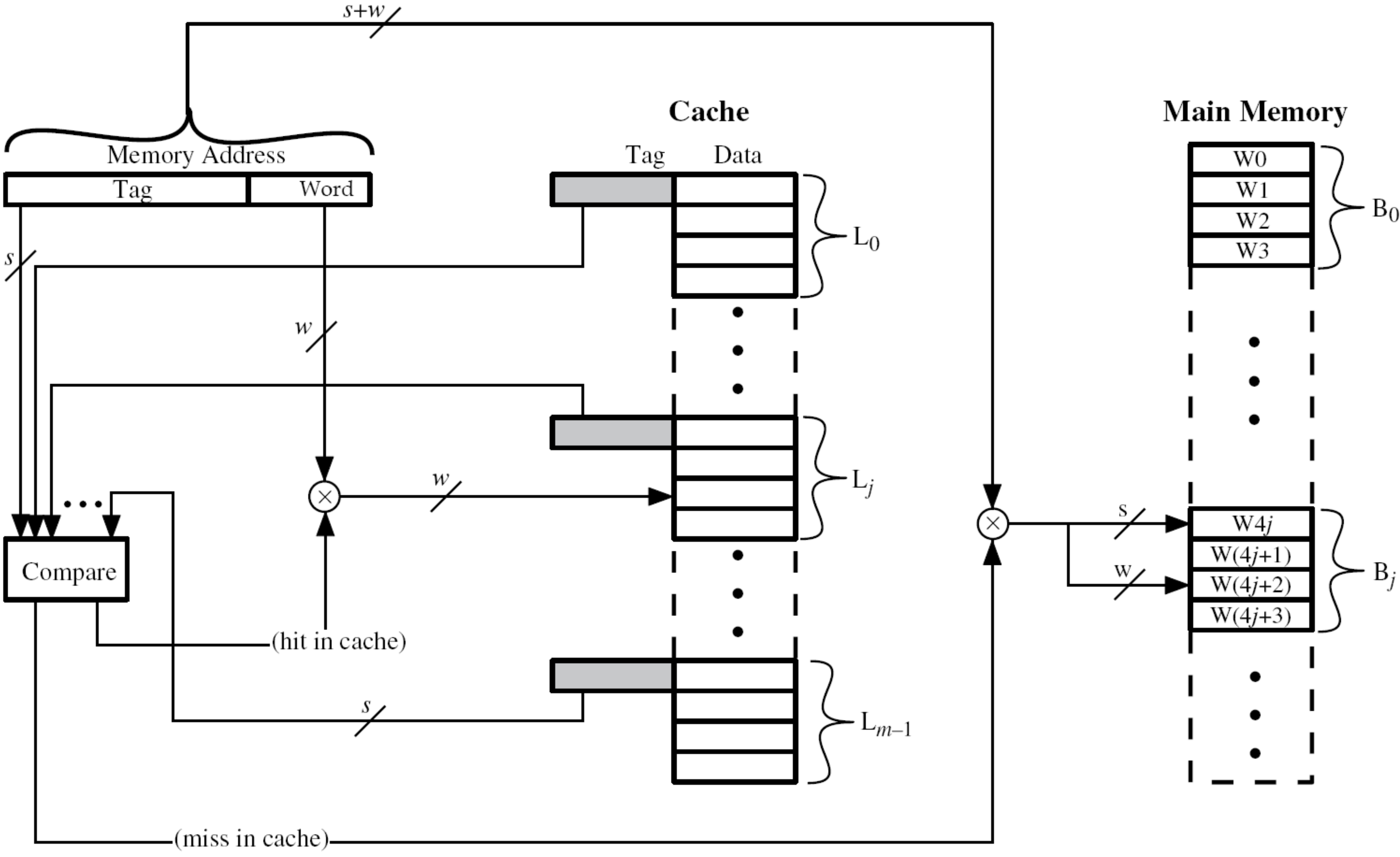
Word #
(**w** bits)
2

- 24-bit address
  - s + w = (s-r) + r + w = 24
- 4-byte block ➔ 4M-block cache
  - w = $\log_2 4$ = 2 bits
  - s = 24 – w = 22 bits (Another way: s = $\log_2$ 4M = 22 bits)
- Note:
  - Each cache line can store a 4-byte block and a 22-bit tag (identifying that block).
  - To find a block in cache, compare the address tag field (s) against all cache line tags until hit!!

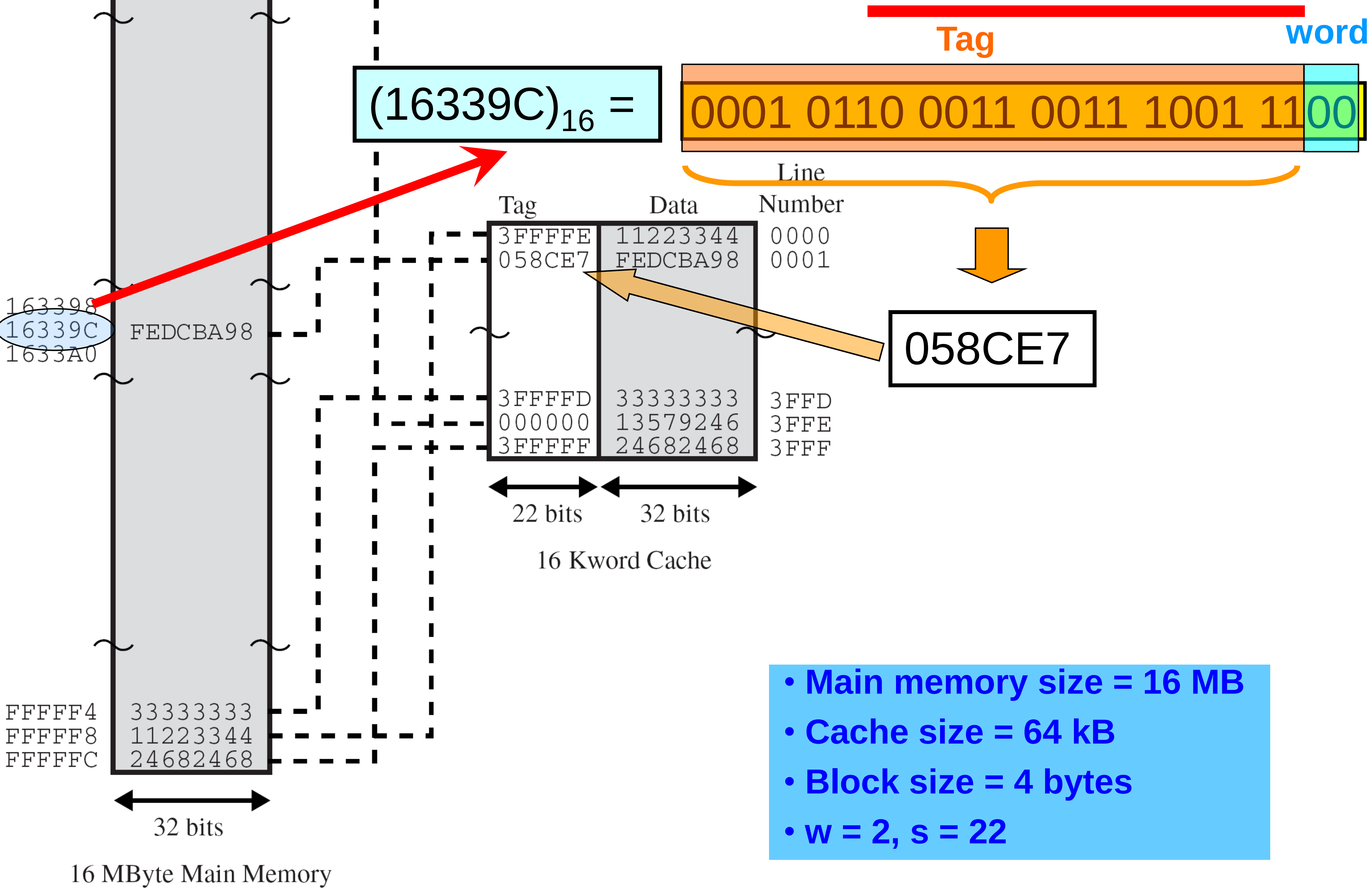
(II) Associative Mapping Address Format Summary

- Address length = (s + w) bits.
- Number of addressable units = 2^{s+w} words.
- Block size = line size = 2^w words.
  - ➢ w = $\log_2$ (# of words in block).
- Number of blocks in MM = M = $2^{s+w}/2^w = 2^s$.
  - ➢ s = $\log_2$ (# of words in MM / # of words in block)
- Number of lines in cache = m ➔ unknown!
- Size of tag = s.

(II) Associative Mapping Cache Organization



(II) Associative Mapping Big Example – Search



Address	Data
000000	13579246
000004	
163398	
16339C	FEDCBA98
1633A0	
FFFFF4	33333333
FFFFF8	11223344
FFFFFC	24682468

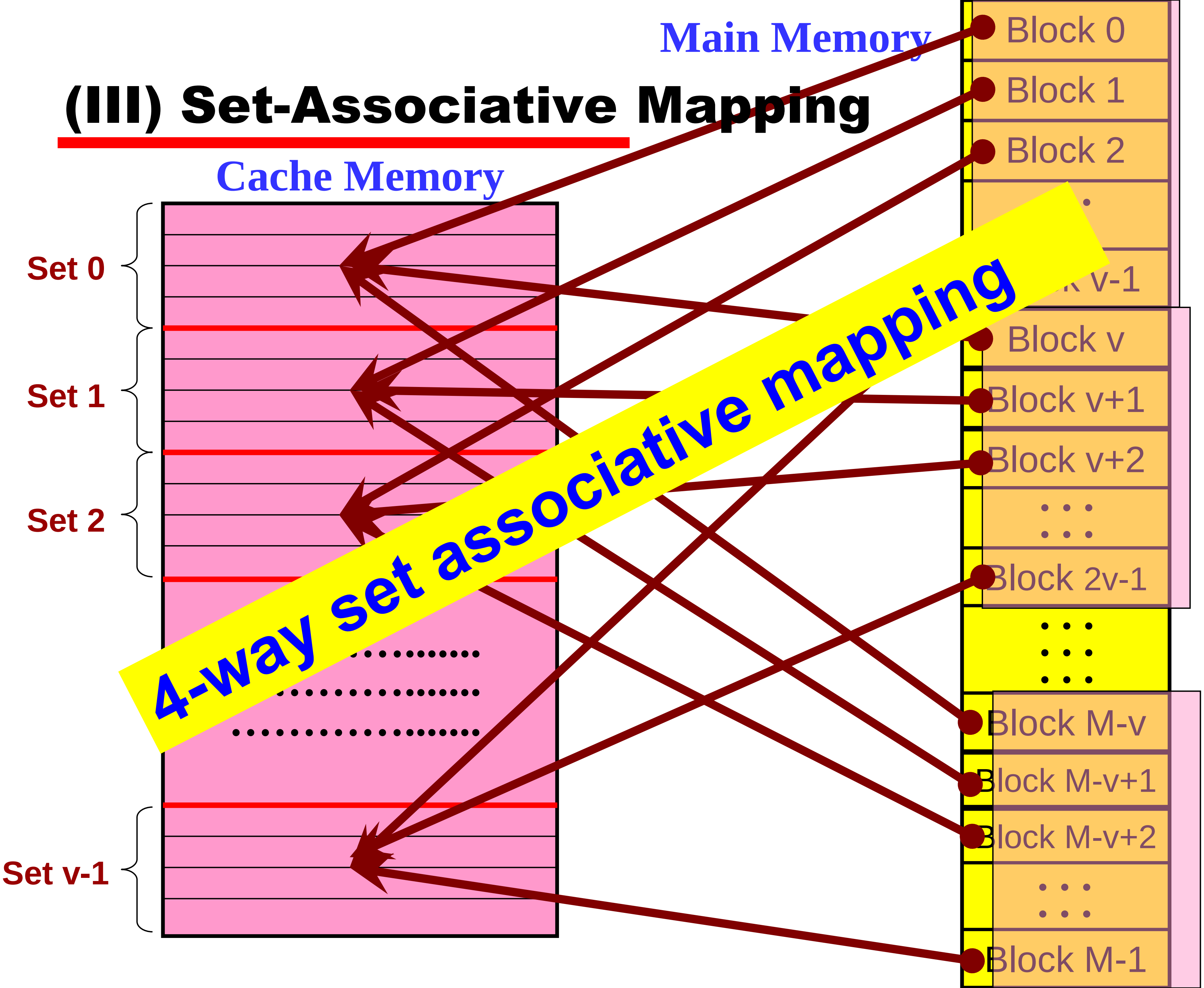
Tag	Data	Line Number
3FFFFE	11223344	0000
058CE7	FEDCBA98	0001
3FFFFD	33333333	3FFD
000000	13579246	3FFE
3FFFFF	24682468	3FFF

- Main memory size = 16 MB
- Cache size = 64 kB
- Block size = 4 bytes
- w = 2, s = 22

(III) Set-Associative Mapping

- Cache is divided into a number of **sets** (v) of equal size.
- Each set contains a number of lines (k).
  - **k-way** set-associative mapping!
- A block b could map to **any line** in a set i if and only if i = b modulo v.

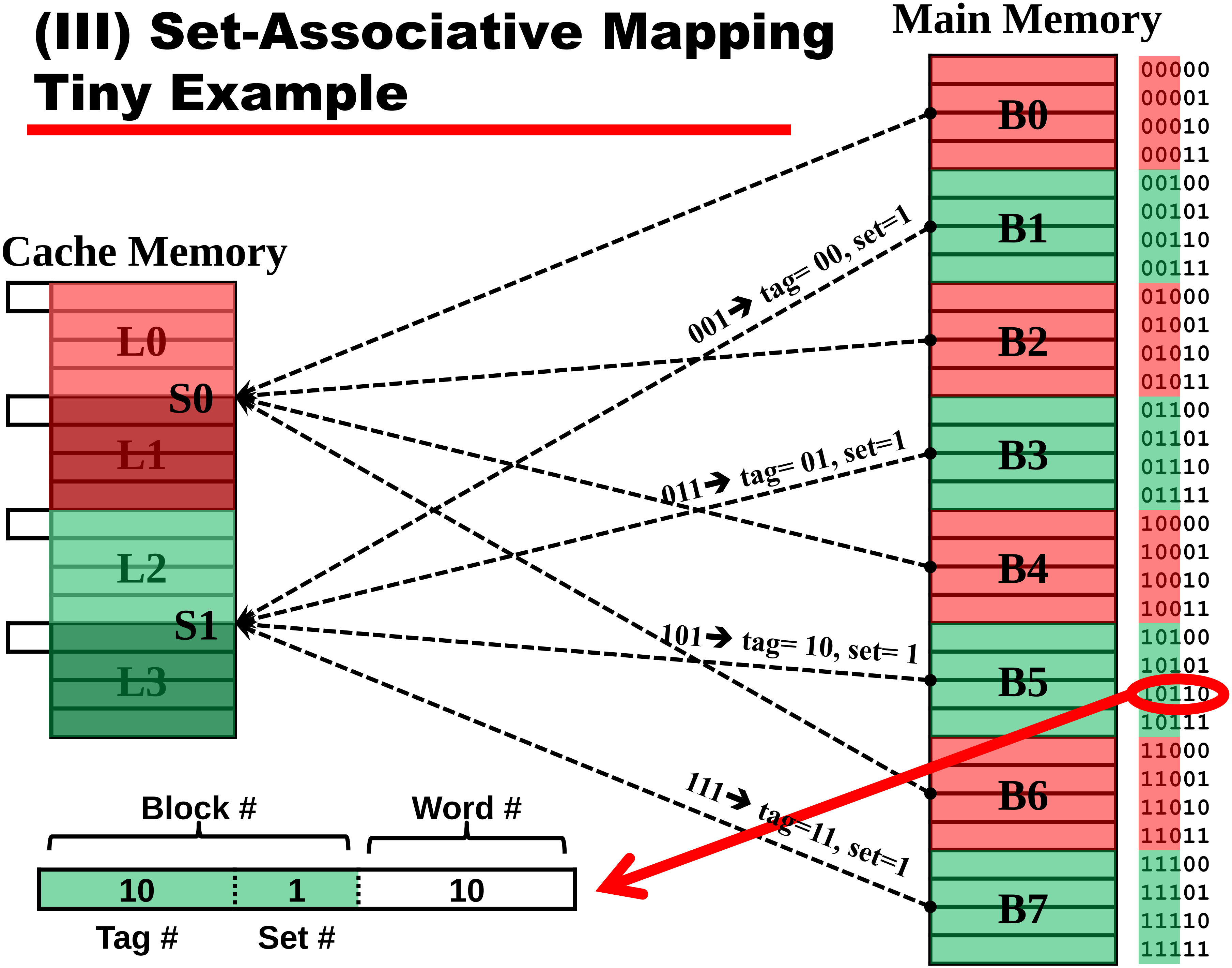
(III) Set-Associative Mapping



(III) Set-Associative Mapping Tiny Example

- Main memory:
  - 1 Location ≡ 1 word ≡ 1 byte
  - 32 byte
    - Length of address = $\log_2 32 = 5$
  - 4-byte **blocks**
    - # of blocks (M) = 32 / 4 = 8
- Cache:
  - 16 byte
  - 4-bytes **lines**
    - # of lines (m) = 16 / 4 = 4
  - 2-line **sets** ➔ k=2 ➔ 2-way set-associative
    - # of sets (v) = 4 / 2 = 2

(III) Set-Associative Mapping Tiny Example



(III) Set-Associative Mapping Address Format

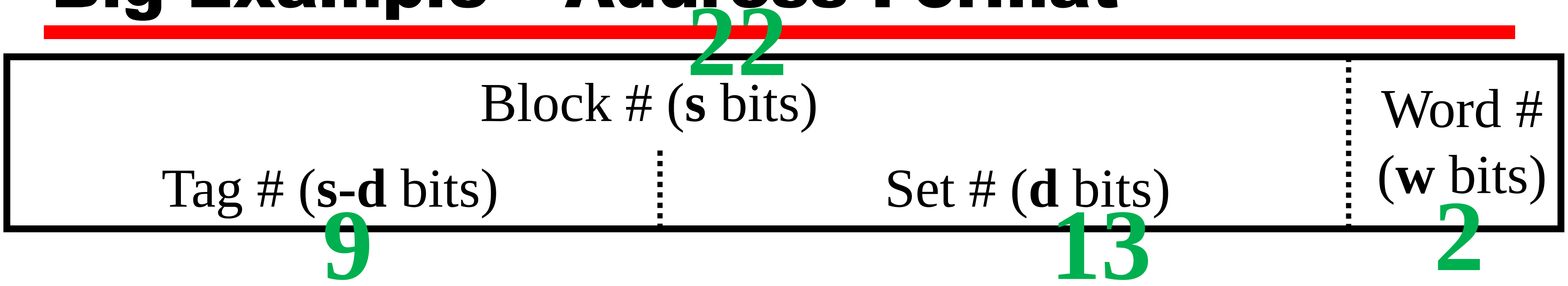
- Memory address is split (based on block size) into:
  1. Least significant **w** bits → identify **word** in block.
     - w = $\log_2$ (# of words in block).
  2. Most significant **s** bits → identify **block** in MM.
     - s = $\log_2$ (# of blocks in MM).
     - The **s** bits are split (based on cache size) into:
       1. Least significant **d** bits → identify cache **set**.
          - d= $\log_2$ (# of sets in cache).
       2. Most significant **s – d** bits → **tag** (identify group).
- In the "tiny example": w=2, s=3, d=1, s-d=2.

<table>
<tr><td colspan="2">Block # (<b>s</b> bits)</td><td rowspan="2">Word # (<b>w</b> bits)</td></tr>
<tr><td>Tag # (<b>s-d</b> bits)</td><td>Set # (<b>d</b> bits)</td></tr>
</table>

(III) Set-Associative Mapping Big Example

- Main memory:
  - —1 Location ≡ 1 word ≡ 1 byte
  - —16M byte
    - ➢ Length of address = $\log_2 16M = 24$
  - —4-byte **blocks**
    - ➢ # of blocks (M) = 16M / 4 = 4M
- Cache:
  - —64K byte
  - —4-bytes **lines**
    - ➢ # of lines (m) = 64K / 4 = 16K
  - —2-line **sets** ➔ k=2 ➔ 2-way set-associative
    - ➢ # of sets (v) = 16K / 2 = 8K

(III) Set-Associative Mapping Big Example – Address Format



22

Block # (**s** bits)

Word # (**w** bits)

Tag # (**s-d** bits)

Set # (**d** bits)

9

13

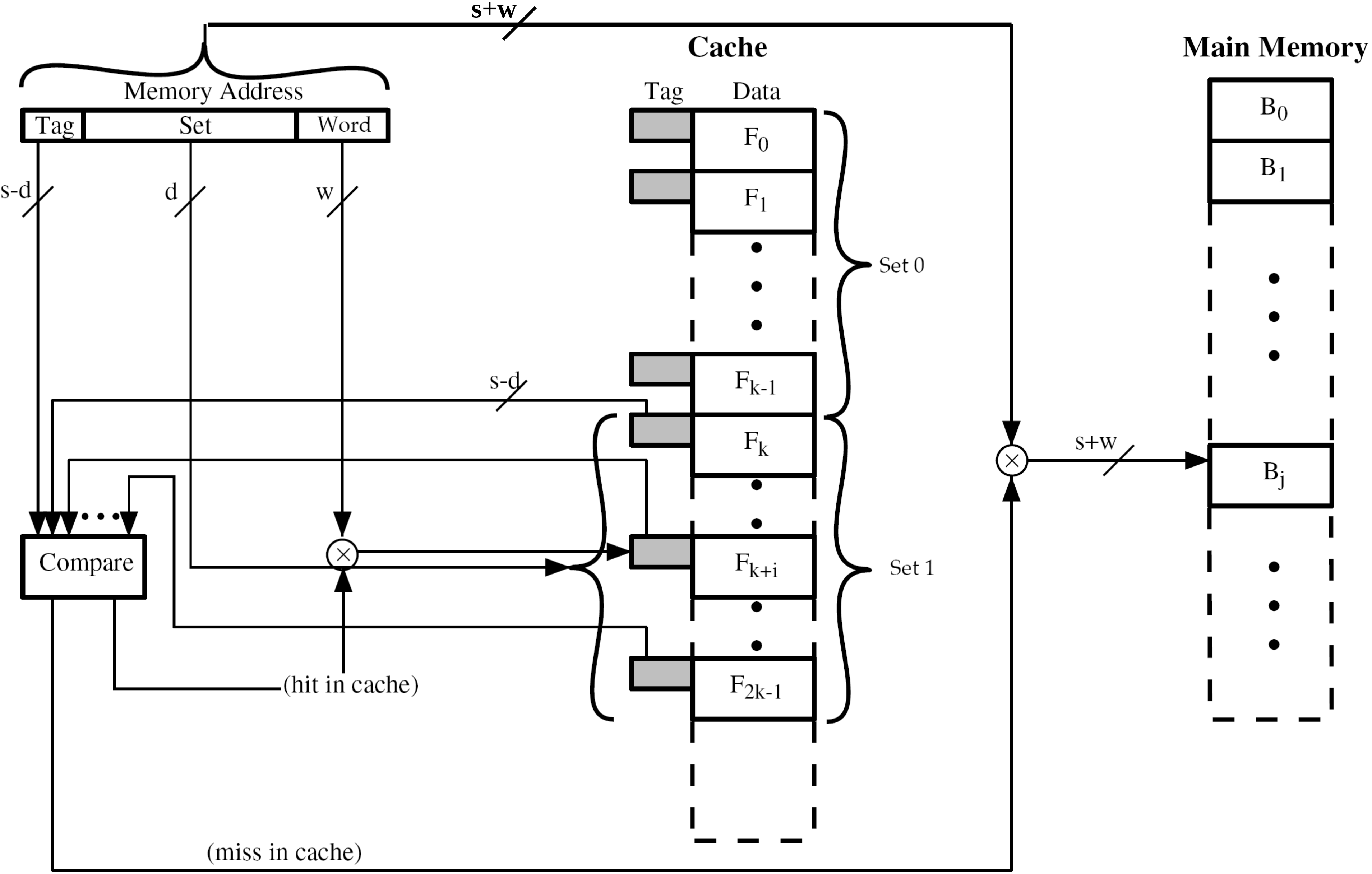
2

- 24-bit address
  - $s + w = (s-r) + r + w = 24$
- 4-byte block
  - $w = \log_2 4 = 2$ bits
  - $s = 24 – w = 22$ bits
- 64K-byte Cache ➔ 16K-line Cache ➔ 8K-set Cache
  - $d = \log_2 8K = 13$ bits
  - $s-r = 22 – 13 = 9$ bits

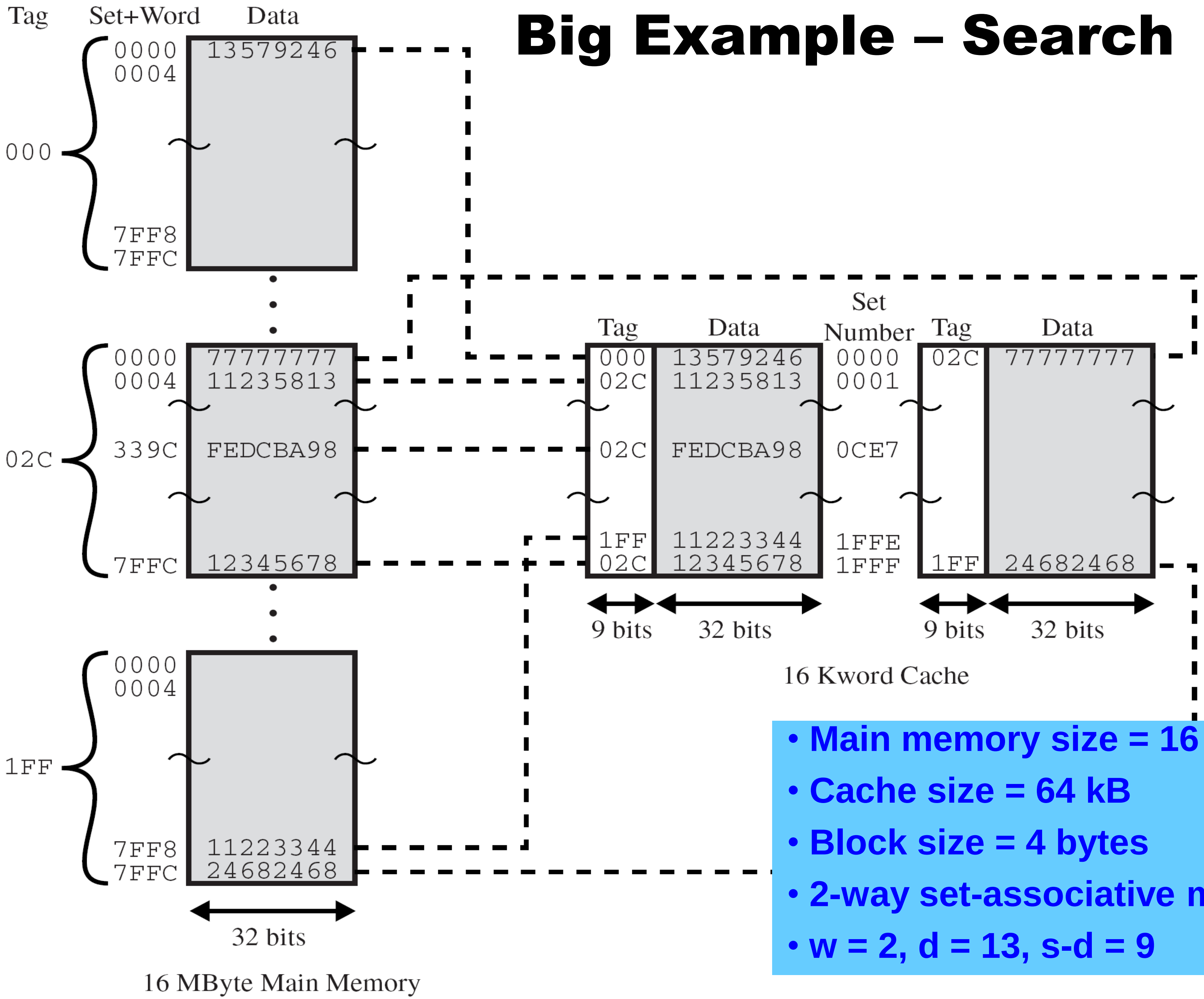
(III) Set-Associative Mapping Address Format Summary

- Address length = $(s + w)$ bits.
- Number of addressable units = 2^{s+w} words.
- Block size = line size = 2^w words.
  - ➢ $w = \log_2$ (# of words in block).
- Number of blocks in MM = $M = 2^{s+w}/2^w = 2^s$.
  - ➢ $s = \log_2$ (# of words in MM / # of words in block)
- Number of lines in set = k ➔ **k-way set-associative**
- Number of sets = $v = 2^d$.
- Number of lines in cache = $m = k * v = k * 2^d$.
  - ➢ $d = \log_2$ (# of words in cache / # of words in set).
- Size of tag = $(s – d)$ bits.
  - ➢ $(s\text{-}d) = \log_2$ (# of blocks in MM / # of sets in cache).

(III) Set-Associative Mapping Cache Organization (k-way)



(III) Set-Associative Mapping Big Example – Search



- Main memory size = 16 MB
- Cache size = 64 kB
- Block size = 4 bytes
- 2-way set-associative mapping
- w = 2, d = 13, s-d = 9

Reading Material

- Stallings, Chapter 4:
  - Pages 123 – 136