BITS Pilani
Pilani Campus

Computer Organization and Software Systems

CONTACT SESSION 4

Pruthvi Kumar K R

Problem 6

Suppose a 1024-byte cache has an access time of 0.1 microseconds and the main memory stores 1 Mbytes with an access time of 1 microsecond. A referenced memory block that is not in cache must be loaded into cache .

Answer the following questions:

a) What is the number of bits needed to address the main memory?

b) If the cache hit ratio is 95%, what is the average access time for a memory reference?

Problem 6

Suppose a 1024-byte cache has an access time of 0.1 microseconds and the main memory stores 1 Mbytes with an access time of 1 microsecond. A referenced memory block that is not in cache must be loaded into cache .

Answer the following questions:

a) What is the number of bits needed to address the main memory?

20 bits

a) If the cache hit ratio is 95%, what is the average access time for a memory reference?

Solution 6

b) If the cache hit ratio is 95%, what is the average access time for a memory reference?

Avg access time = hit ratio * cache access +

(1- hit ratio) * (cache access + memory access)

= .95 * 0.1 microsec + .05 * (1 + 0.1) microsec

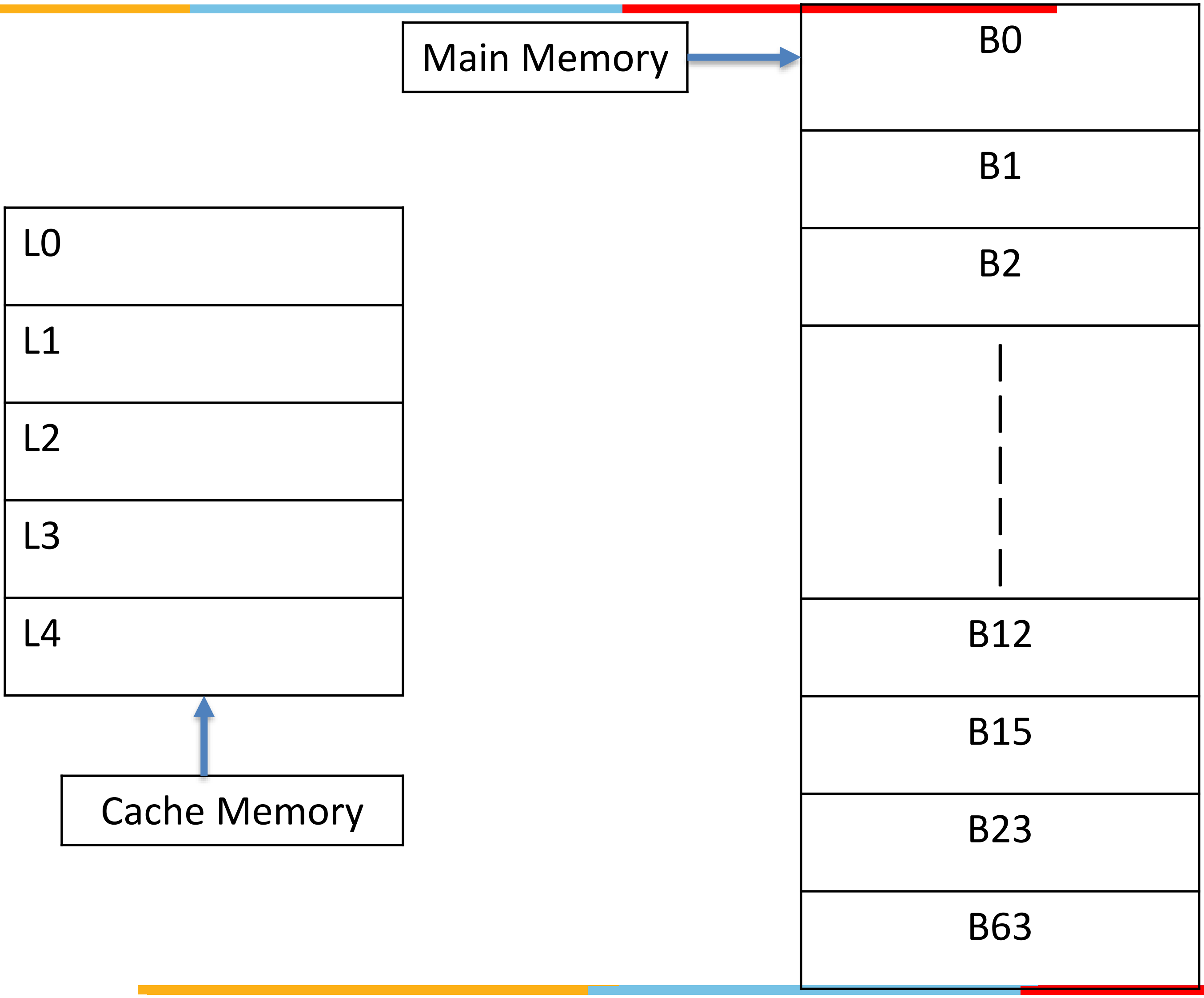
= .095 + .055 microsec

= 0.15 microseconds

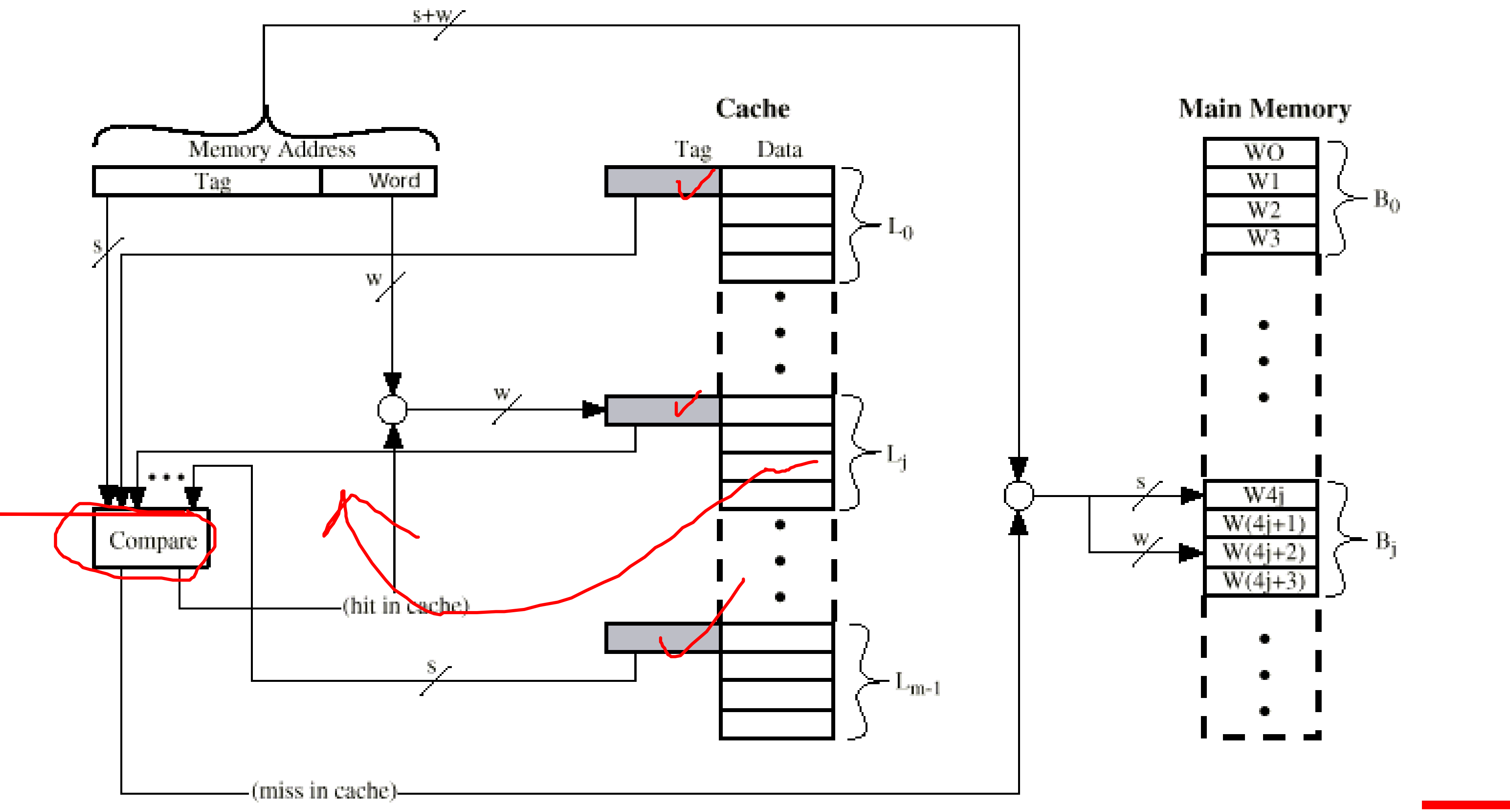
Associative Mapping

- 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

Associative Mapping Cache Organization



Associative Cache Organization



Associative Mapping Summary

- Address length = (s + w) bits
- Number of addressable units = 2^{s+w} words or bytes
- Block size = line size = 2^{w} words or bytes
- Number of blocks in main memory = $2^{s+w}/2^{w} = 2^{s}$
- Number of lines in cache = undetermined
- Size of tag = s bits

Problem 7

Given :

- Cache of 128kByte, Cache block of 8 bytes
- 32 MBytes main memory

Find out

a) Number of bits required to address the memory

b) Number of blocks in main memory

c) Number of cache lines

d) Number of bits required to identify a word (byte) in a block?

e) Tag, Word

Problem 7

Given :

- Cache of 128kByte, Cache block of 8 bytes
- 32 MBytes main memory

Find out

a) Number of bits required to address the memory=25 bits

b) Number of blocks in main memory

c) Number of cache lines

d) Number of bits required to identify a word (byte) in a block?

e) Tag, Word

Problem 7

Given :

- Cache of 128kByte, Cache block of 8 bytes
- 32 MBytes main memory

Find out

a) Number of bits required to address the memory=25 bits

b) Number of blocks in main memory=$\boxed{2\text{^}22}$ blocks or

4M blocks

c) Number of cache lines

d) Number of bits required to identify a word (byte) in a block?

e)Tag, Word

Problem 7

Given :

- Cache of 128kByte, Cache block of 8 bytes
- 32 MBytes main memory

Find out

a) Number of bits required to address the memory=25 bits

b) Number of blocks in main memory=2^22 blocks or 4M blocks

c)Number of cache lines= 16k Lines

d) Number of bits required to identify a word (byte) in a block?

e)Tag, Word

Problem 7

Given :

- Cache of 128kByte, Cache block of 8 bytes
- 32 MBytes main memory

Find out

a) Number of bits required to address the memory=25 bits

b) Number of blocks in main memory=2^22 blocks or 4M blocks

c)Number of cache lines= 16k Lines

d) Number of bits required to identify a word (byte) in a block?= 3bits

e)Tag, Word

Problem 7

Given :

- Cache of 128kByte, Cache block of 8 bytes
- 32 MBytes main memory

Find out

a) Number of bits required to address the memory=25 bits

b) Number of blocks in main memory=2^22 blocks or 4M blocks

c)Number of cache lines= 16k Lines

d) Number of bits required to identify a word (byte) in a block?= 3bits

e)Tag, Word = 22 bits, 3bits

Problem 8

Cache of 64kByte, Cache block of 4 bytes and 16 M Bytes main memory and associative mapping.

Fill in the blanks:

Number of bits in main memory address = ______

Number of lines in the cache memory = _______

Word bits = ___________

Tag bits = __________

Problem 8

Cache of 64kByte, Cache block of 4 bytes and 16 M Bytes main memory and associative mapping.

Fill in the blanks:

Number of bits in main memory address = __24bits___

Number of lines in the cache memory = __16K lines____

Word bits = ___2 bits______

Tag bits = ____22 bits____

Problem 9

- 16 Bytes main memory, Memory block size is 4 bytes, Cache of 8 Byte (cache is 2 lines of 4 bytes each) and associative mapping
- Block access sequence :

$\boxed{0}$ 2 0 2 2 0 0 2 0 0 0 2 1

X X ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ ✓ X

5 0

1 2

- Find out hit ratio.

Set Associative Mapping

- m line Cache is divided into a number of sets (v sets each with k lines)
- m = v * k

  i = j modulo $\boxed{v}$

  where i = cache set number

  j = main memory block number

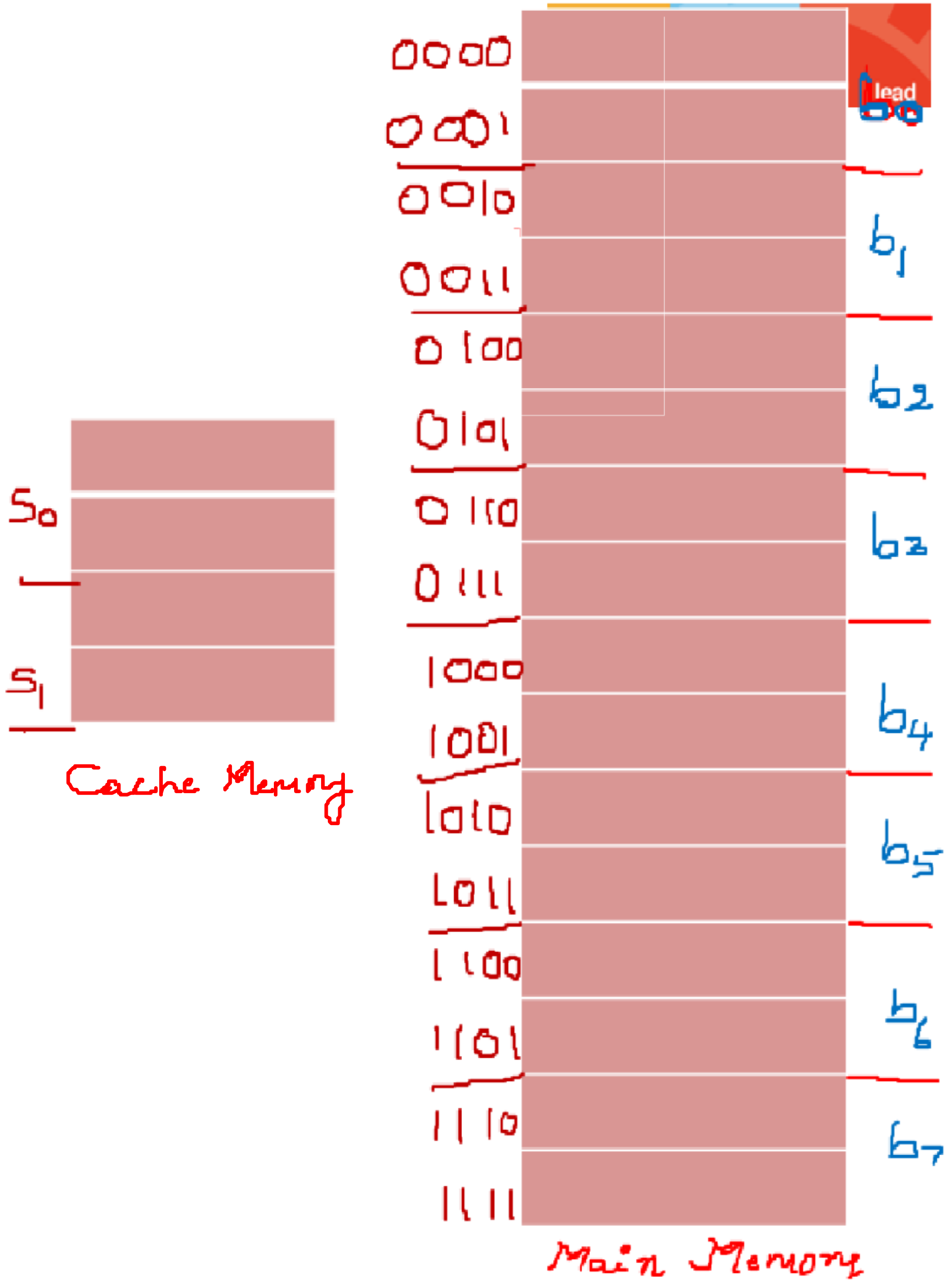
  m = number of lines in the cache

Tag	Set	Word

- Each set contains 'k' number of lines
- A given block maps to any line in a given set
  - e.g. Block B can be in any line of set i
- e.g. 2 lines per set
  - 2 way set associative mapping
  - A given block can be in one of 2 lines in only one set

Example

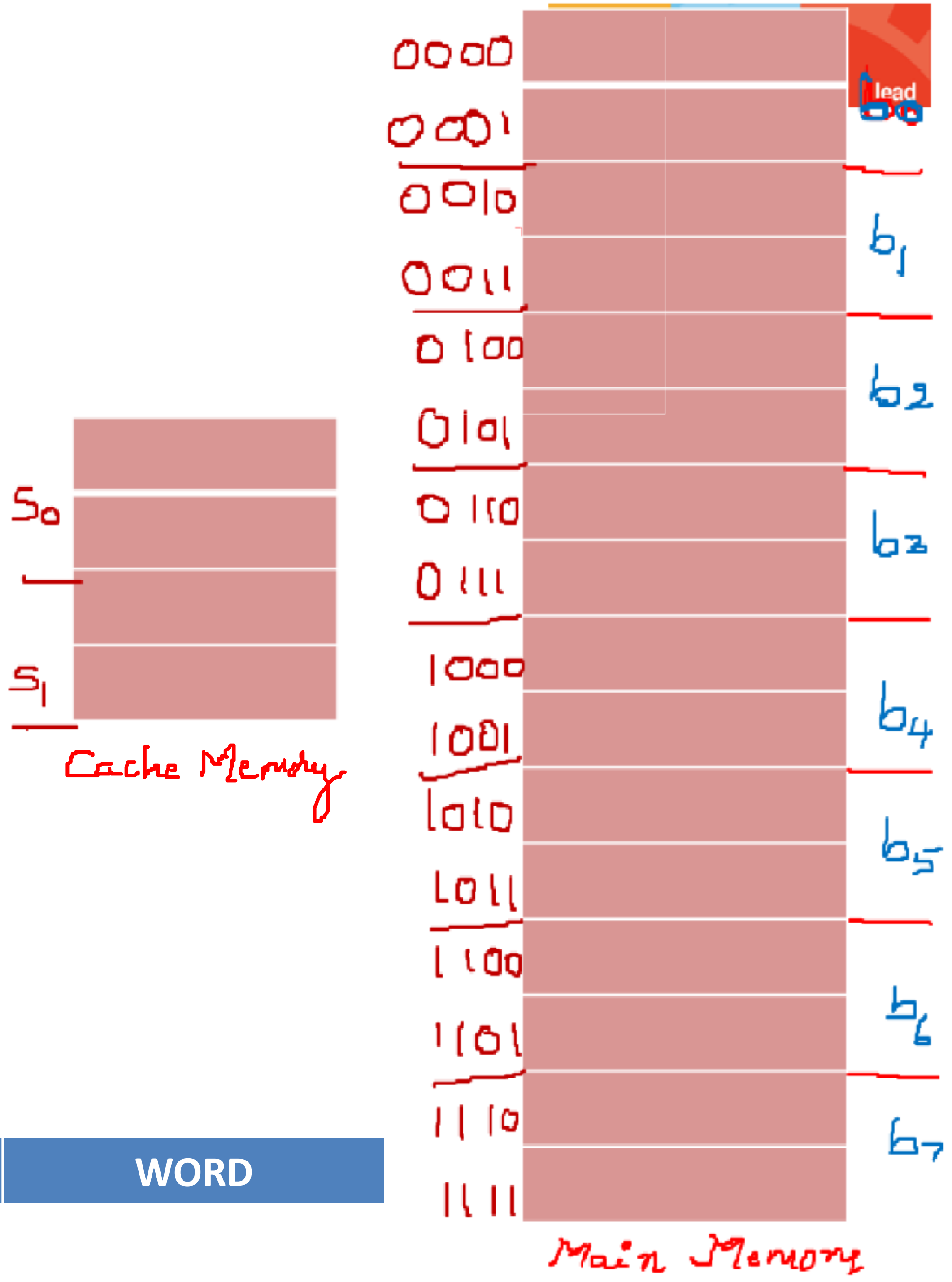
- 16 Bytes main memory, Block Size is 2 Bytes,
- Cache of 8 Bytes, 2 way set associative cache
  - # address bits
  - Cache line size
  - # main memory blocks
  - # Number of cache lines
  - # lines per set
  - # of sets



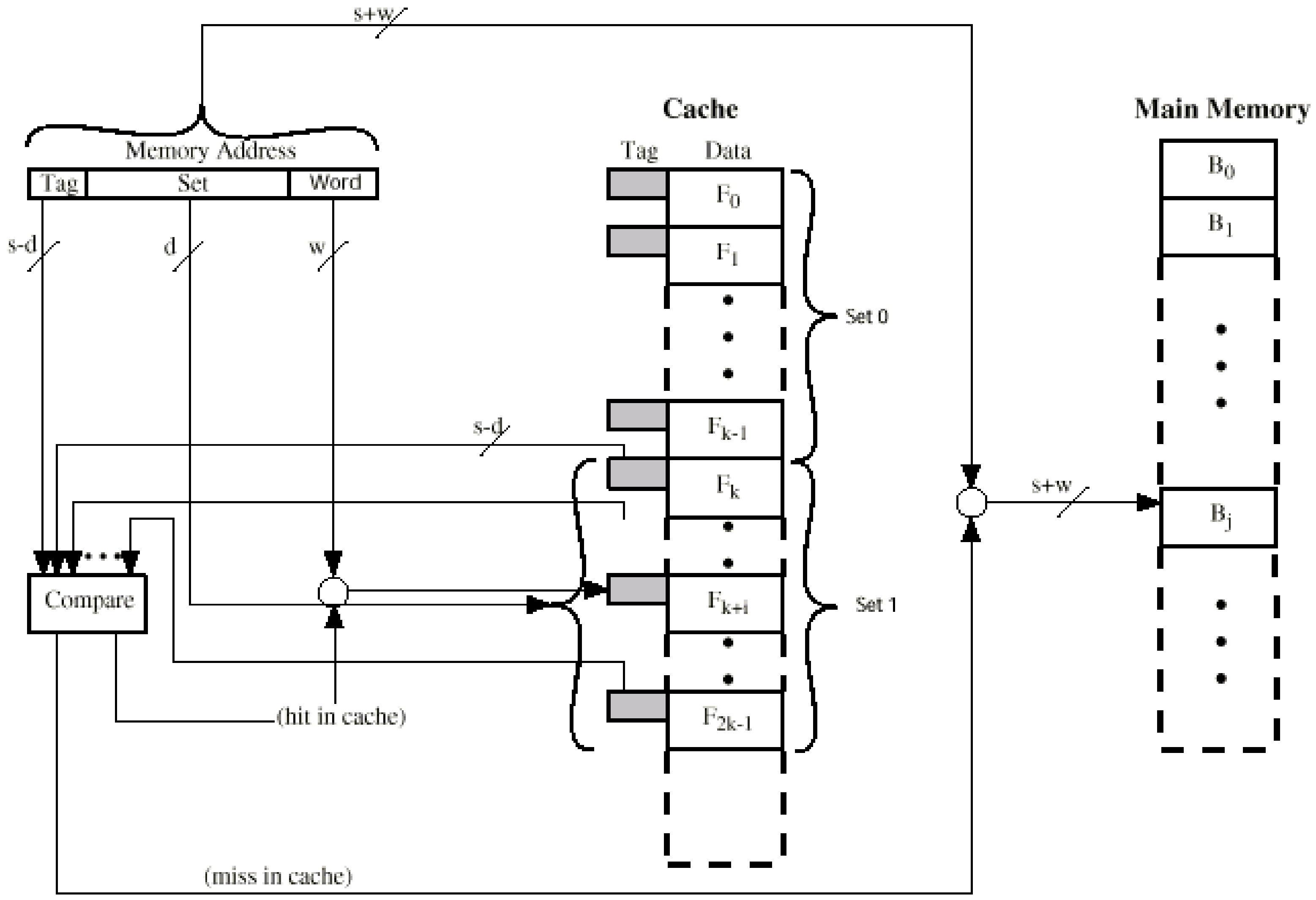
Example – Mapping Function

i = j modulo v	Set #
0%2	
1%2	
2%2	
3%2	
4%2	
5%2	
6%2	
7%2	

TAG	SET	WORD



Set Associative Cache Organization



Set Associative Mapping Summary

Address length = $(s + w)$ bits

Number of addressable units = 2^{s+w} words or bytes

Block size = line size = 2^w words or bytes

Number of blocks in main memory = 2^d

Number of lines in set = k

Number of sets = $v = 2^d$

Number of lines in cache = $kv = k * 2^d$

Size of tag = $(s - d)$ bits

Problem 1

A set-associative cache consists of 64 lines, or slots, divided into four-line sets. Main memory contains 4K blocks of 128 bytes each. Show the format of main memory addresses. Find out

- Total main memory capacity
- Total cache memory capacity
- Total number of sets in the cache
- Number of bits for TAG, SET and word

$\boxed{19}$

$4K \times 128 = 2^{19} = 512K$

$64 \times 128 = 2^{13} = 8KB$

8, 4, 7

Problem 2(Byte Addressable)

A 4-way set-associative cache memory unit with a capacity of 16 KB is built using a block size of 8 words. The word length is 32 bits. The size of the physical address space is 4 GB. Find out address format.

$4 \times 2^{30} = 2^{32}$

$32 = 2^{5}$

Lines $= 16KB / 32 = 2^{14} / 2^{5} = 2^{9}$

Sets $= 2^{9}/4 = 2^{7}$ sets

Problem 2(Word Addressable) HomeWork

A 4-way set-associative cache memory unit with a capacity of 16 KB is built using a block size of 8 words. The word length is 32 bits. The size of the physical address space is 4 GB. Find out address format.

Replacement Algorithms (1/3)

Direct mapped cache

- No choice
- Each block maps to one line and replace that line

Replacement Algorithms (2/3)

- Needed in Associative & Set Associative mapped cache
- Hardware implemented algorithm (speed)
- Methods:
  - Least Recently Used (LRU)
  - Least Frequently Used (LFU)
  - First In First Out (FIFO)
  - Random

Replacement Algorithms (3/3)

- Least Recently used (LRU): Replace the block in the set that has been in the cache longest with no reference to it
  - e.g. 2 way set associative
  - Uses "USE" bits
  - Most effective method
- Least frequently used: Replace block which has had fewest hits
  - Uses counter with each line
- First in first out (FIFO): Replace block that has been in cache longest
  - Round robin or circular buffer technique
- Random

Problem 2

Consider a reference pattern that accesses the sequence of blocks 0, 4, 0, 2, 1, 8, 0, 1, 2, 3, 0, 4. Assuming that the cache uses associative mapping, find the hit ratio with a cache of four lines

a) LRU

b) LFU

c) FIFO

Problem 2 - LRU

Ref	0	4	0	2	1	8	0	1	2	3	0	4
time	0	1	2	3	4	5	6	7	8	9	10	11
L0												
L1												
L2												
L3												
H/M												

Problem 2 - LFU

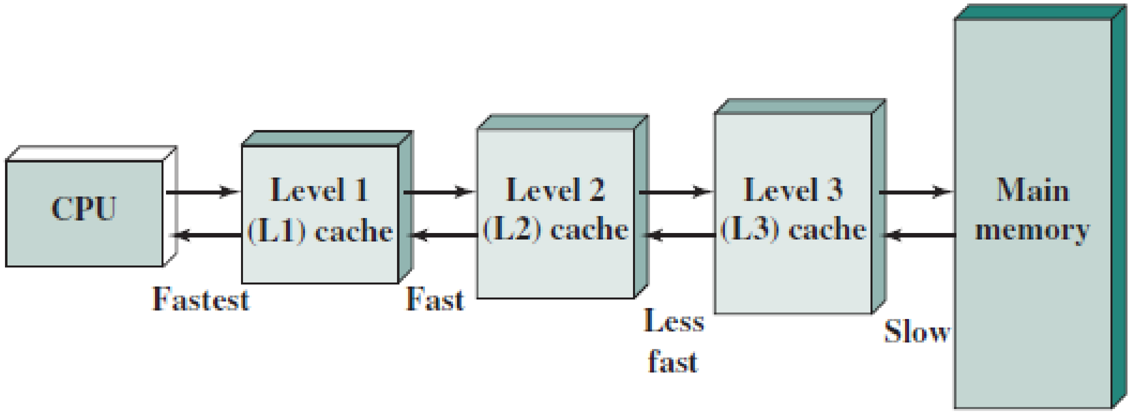
Ref	0	4	0	2	1	8	0	1	2	3	0	4
L0												
L1												
L2												
L3												
H/M												

Problem 2 - FIFO

Ref	0	4	0	2	1	8	0	1	2	3	0	4
time	0	1	2	3	4	5	6	7	8	9	10	11
L0												
L1												
L2												
L3												
H/M												

Issues with Writes

- Multiple copies of data exist:
  - L1, L2, L3, Main Memory, Disk
- What to do on a write-hit?
  - Write-through (write immediately to memory)
  - Write-back (defer write to memory until replacement of line)
    - Need a dirty bit (line different from memory or not)



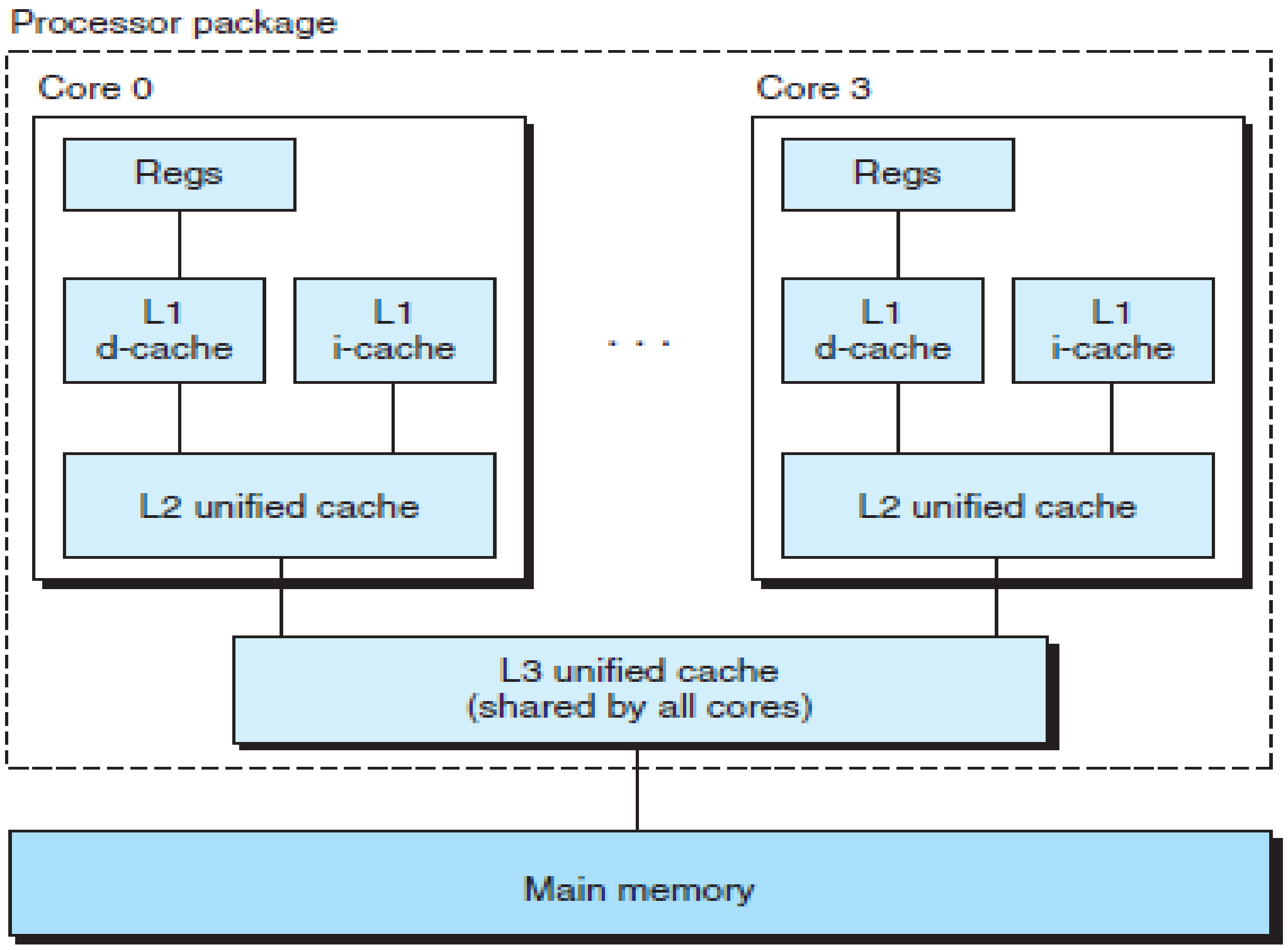
(b) Three-level cache organization

Cache and Main Memory

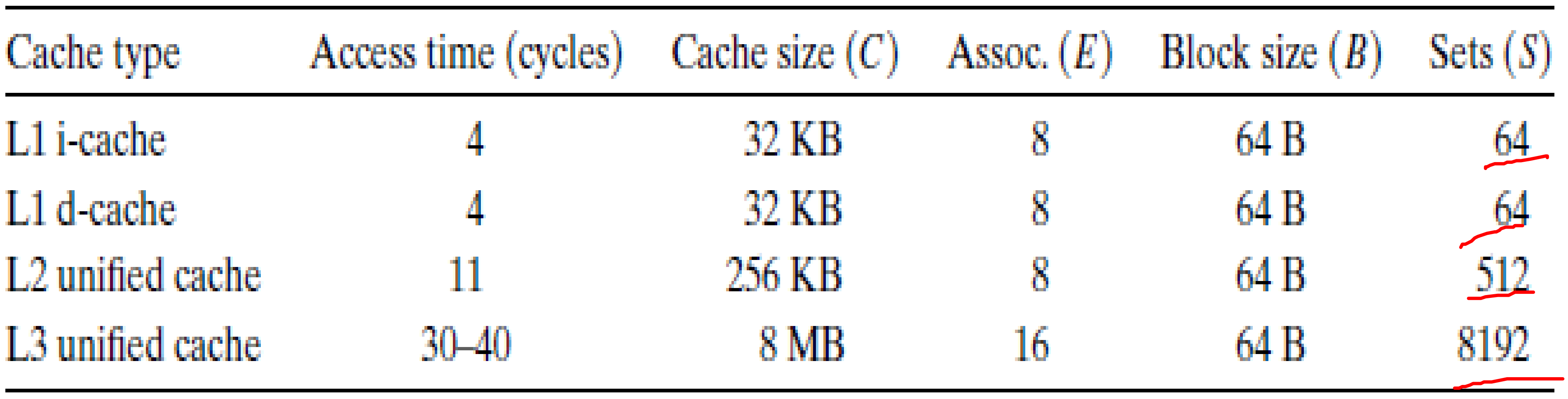
Issues with Writes

- What to do on a write-miss?
  - Write-allocate (load into cache, update line in cache)
    - Good if more writes to the location follow
  - No-write-allocate (writes straight to memory, does not load into cache)
- Typical
  - Write-through + No-write-allocate
  - **Write-back + Write-allocate**

Intel Core i7 Cache Hierarchy



Intel Core i7 Cache Hierarchy



Cache type	Access time (cycles)	Cache size (C)	Assoc. (E)	Block size (B)	Sets (S)
L1 i-cache	4	32 KB	8	64 B	64
L1 d-cache	4	32 KB	8	64 B	64
L2 unified cache	11	256 KB	8	64 B	512
L3 unified cache	30–40	8 MB	16	64 B	8192

Characteristics of the Intel Core i7 cache hierarchy.

Performance Impact of Cache Parameters

- Associativity :
  - higher associativity ➔ more complex hardware
  - Higher Associativity ➔ Lower miss rate
  - Higher Associativity ➔ reduces average memory access time (AMAT)
- Cache Size
  - Larger the cache size ➔ Lower miss rate
  - Larger the cache size ➔ reduces average memory access time (AMAT)
- Block Size:
  - Smaller blocks do not take maximum advantage of spatial locality.

Revisiting Locality of reference

```
int sumvec(int v[N])
{
    int i, sum = 0;

    for (i = 0; i < N; i++)
        sum += v[i];
    return sum;
}
```

Does this function have good locality?

N=8								
Address	0	1	2	3	4	5	6	7
Contents	v0	v1	v2	v3	v4	v5	v6	V7
Access Order	1	2	3	4	5	6	7	8

Stride k – reference pattern

Byte Addressable memory and word length is 1 byte

i	Address
0	0000
1	0001
2	0002
3	0003
4	0004
5	0005
6	0006
7	0007
8	0008
9	0009
10	000A
11	000B
12	000C

Stride 1

$$\text{Stride} = \frac{\text{Address difference}}{\text{Word Length}}$$

i	Address
0	0000
1	0001
2	0002
3	0003
4	0004
5	0005
6	0006
7	0007
8	0008
9	0009
10	000A
11	000B
12	000C

Stride 2

Stride k – reference pattern

Byte Addressable memory and word length is 2 bytes

i	Address
0	0000
1	0002
2	0004
3	0006
4	0008
5	000A
6	000C
7	000E
8	0010
9	0012
10	0014
11	0016
12	0018

Stride 1

Address difference
Stride=---------------------
Word Length

i	Address
0	0000
1	0002
2	0004
3	0006
4	0008
5	000A
6	000C
7	000E
8	0010
9	0012
10	0014
11	0016
12	0018

Stride 2

Revisiting Locality of reference

```
int sumarrayrows(int a[M][N])
{
    int i, j, sum = 0;

    for (i = 0; i < M; i++)
        for (j = 0; j < N; j++)
            sum += a[i][j];
    return sum;
}
```

Does this function have good locality?

M = 2, N=3						
Address	0	1	2	3	4	5
Contents	a00	a01	a02	a10	a11	a12
Access Order	1	2	3	4	5	6

Revisiting Locality of reference

```
int sumarraycols(int a[M][N])
{
    int i, j, sum = 0;

    for (j = 0; j < N; j++)
        for (i = 0; i < M; i++)
            sum += a[i][j];
    return sum;
}
```

Does this function have good locality?

M = 2, N=3						
Address	0	1	2	3	4	5
Contents	a00	a01	a02	a10	a11	a12
Access Order	1	3	5	2	4	6

Writing Cache Friendly Code

- Make the common case go fast
  - Focus on the inner loops of the core functions

- Minimize the misses in the inner loops
  - Repeated references to variables are good (temporal locality)
  - Stride-1 reference patterns are good (spatial locality)

Example 1

```
int sumarrayrows(int a[4][4])
{
 int i, j, sum = 0;
 for (i = 0; i < 4; i++)
    for (j = 0; j < 4; j++)
     sum += a[i][j];
 return sum;
}
```

Assumption:

- The cache has a block size of 4 words each, 2 cache lines
- Word size 4 bytes.
- C stores arrays in row-major order

Example 1(Contd..)

```
int sumarrayrows(int a[4][4])
{
 int i, j, sum = 0;
 for (i = 0; i < 4; i++)
    for (j = 0; j < 4; j++)
     sum += a[i][j];
 return sum;
}
```

A[i][j]	J = 0	J = 1	J = 2	J = 3
i = 0				
i = 1				
i = 2				
I = 3				

a[0][0]	w_0
a[0][1]	w_1
a[0][2]	w_2
a[0][3]	w_3
a[1][0]	w_4
a[1][1]	w_5
a[1][2]	w_6
a[1][3]	w_7
a[2][0]	w_8
a[2][1]	w_9
a[2][2]	w_{10}
a[2][3]	w_{11}
a[3][0]	w_{12}
a[3][1]	w_{13}
a[3][2]	w_{14}
a[3][3]	w_{15}

Example 2:

```
int sum_array(int a[4][4])
{
 int i, j, sum = 0;
 for (j = 0; j < 4; j++)
   for (i = 0; i < 4; i++)
   sum += a[i][j];
 return sum;
}
```

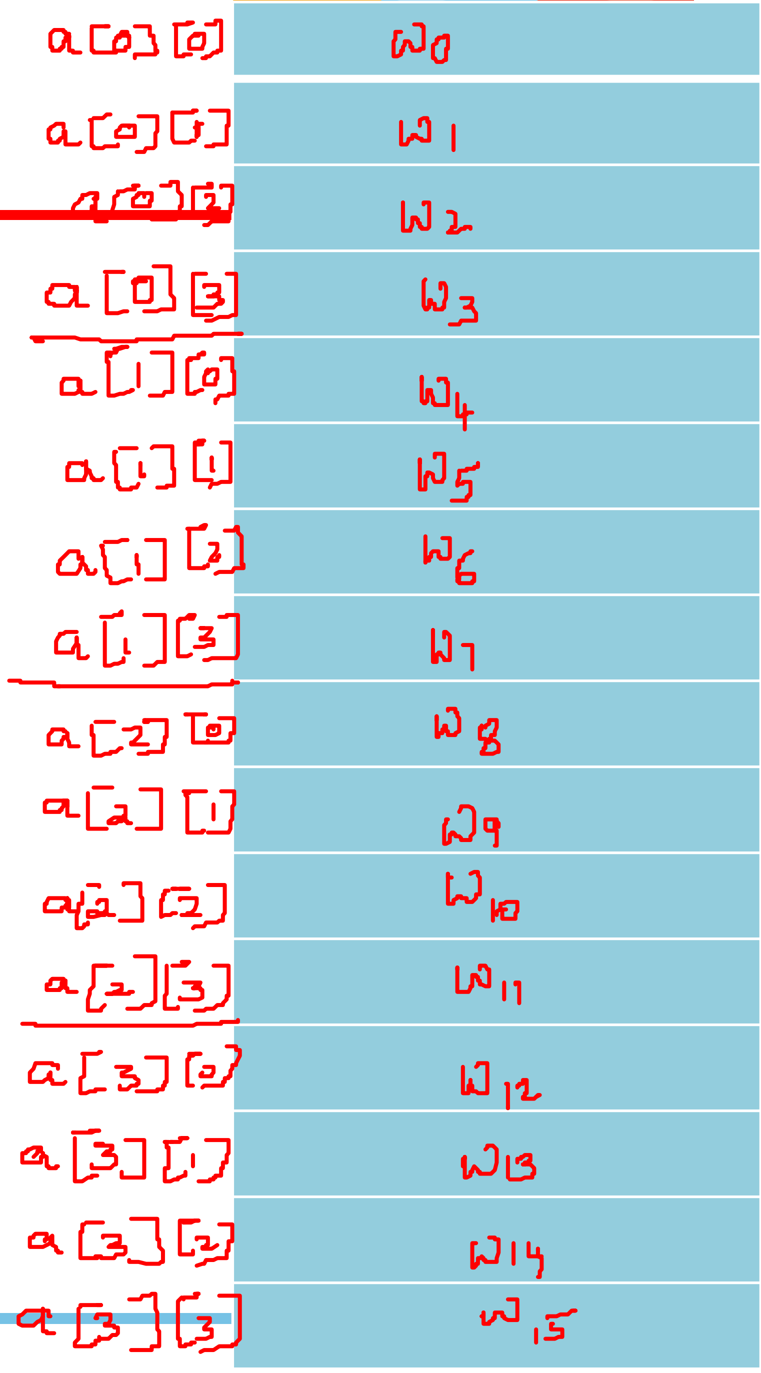
Assumption:

- The cache has a block size of 4 words each, 2 cache lines
- Word size 4 bytes.
- C stores arrays in row-major order

Example 1(Contd..)

```
int sum_array(int a[4][4])
{
 int i, j, sum = 0;
 for (j = 0; j < 4; j++)
   for (i = 0; i < 4; i++)
   sum += a[i][j];
 return sum;
}
```

A[i][j]	J = 0	J = 1	J = 2	J = 3
i = 0				
i = 1				
i = 2				
I = 3				



Home Work – Which one is better ?

Program 1:

```
for (int i = 0; i < n; i++) {
    z[i] = x[i] - y[i];
    z[i] = z[i] * z[i];
}
```

Program 2:

```
for (int i = 0; i < n; i++) {
        z[i] = x[i] - y[i];
}
for (int i = 0; i < n; i++) {
        z[i] = z[i] * z[i];
}
```