

Slides 35,36,37 are
the annotations for
the Replacement
Algorithms



Computer Organization and Software Systems

CONTACT SESSION 4

Prof. C R Sarma
WILP.BITS-PILANI



BITS Pilani
Pilani Campus

Today's Session

Contact Hour	List of Topic Title	Text/Ref Book/external resource
7-8	<ul style="list-style-type: none"> • Memory Organization (Contd..) • Cache Memories (Contd..) <ul style="list-style-type: none"> • <u>Set Associative Caches</u> • Issues with Writes • Performance Impact of Cache Parameters • Writing Cache friendly Codes • Replacement Algorithms 	T1, R1

Set Associative Mapping



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

- $m = v * k$

Tag	Set	Word
-----	-----	------

$$i = j \text{ modulo } v$$

where i = cache set number

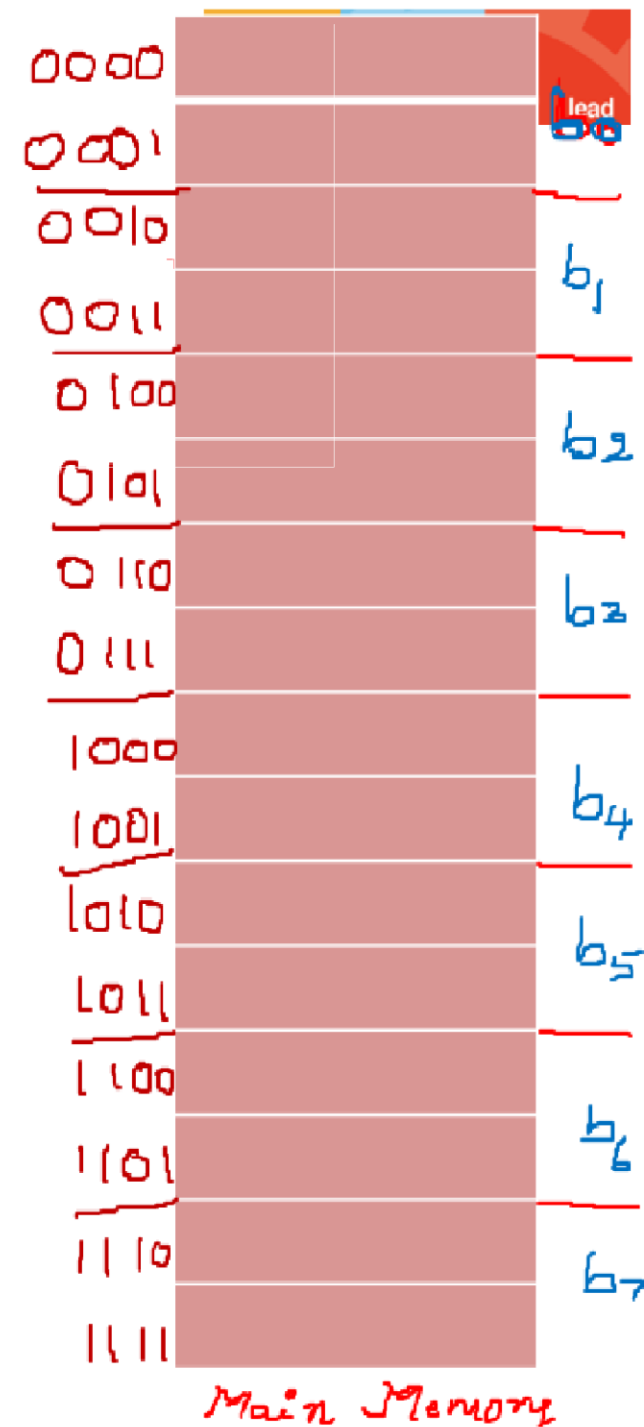
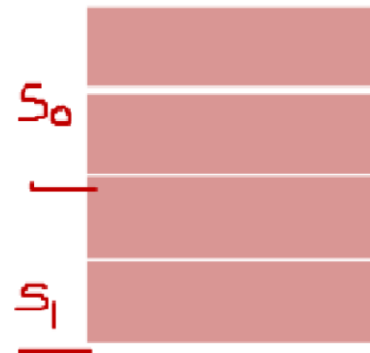
j = main memory block number

m = number of lines in the cache

- 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

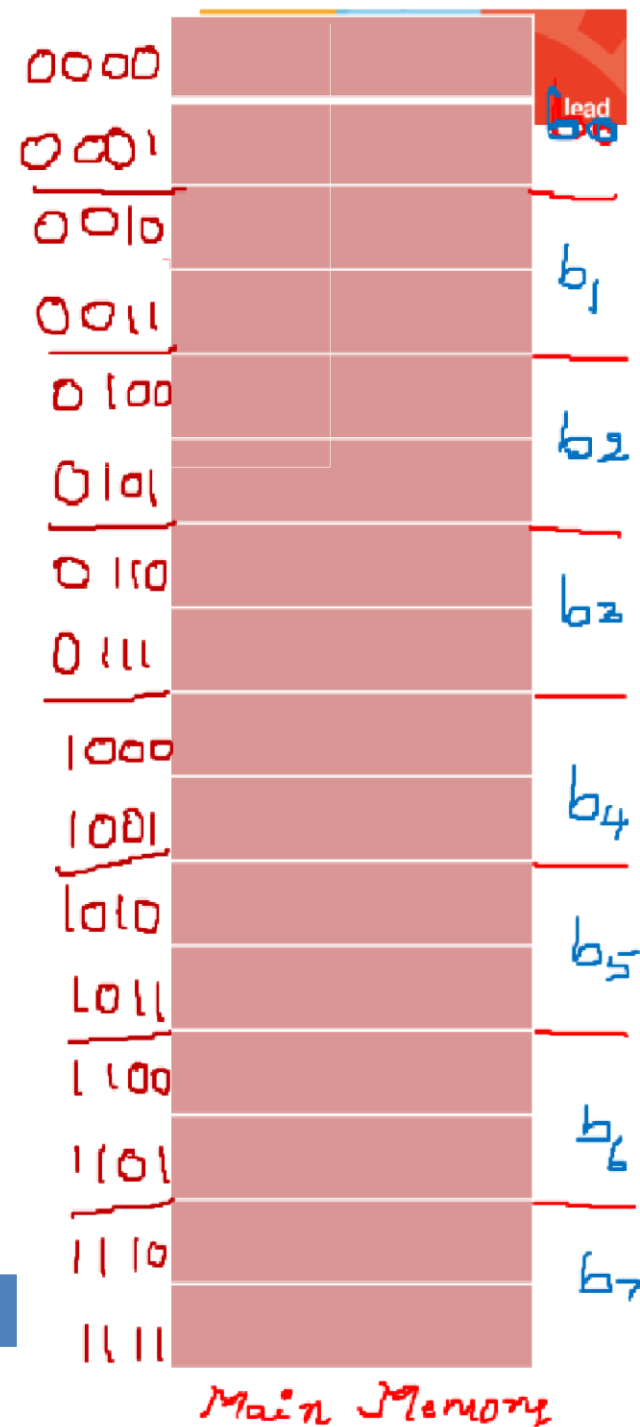
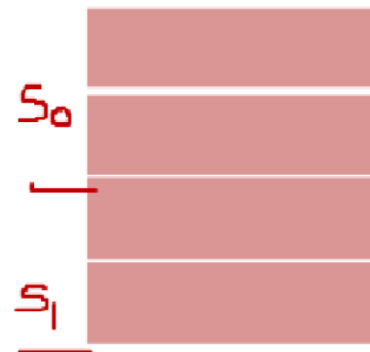
Example

- 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

$i = j \text{ modulo } v$	Set #
$0\%2$	
$1\%2$	
$2\%2$	
$3\%2$	
$4\%2$	
$5\%2$	
$6\%2$	
$7\%2$	



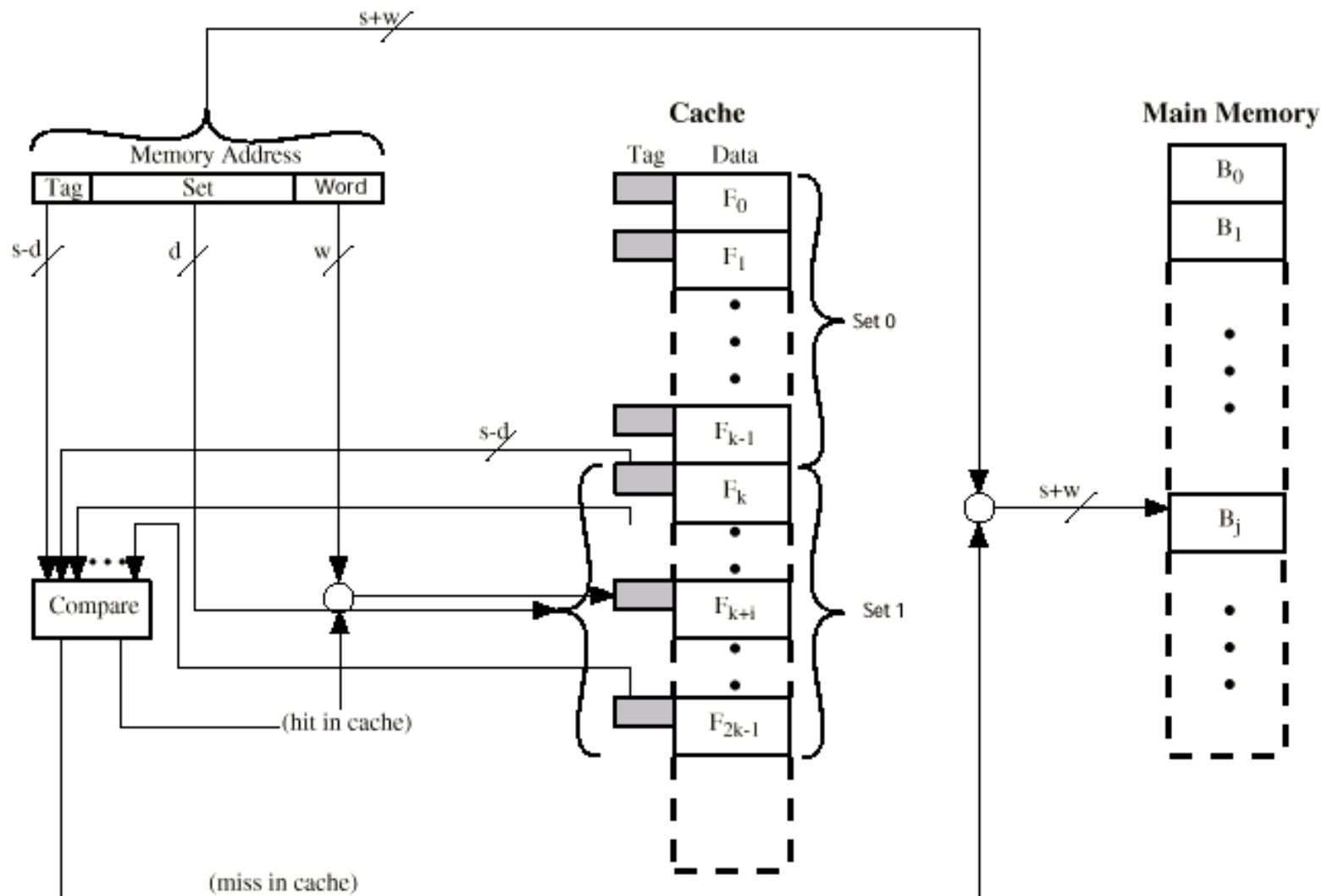
TAG	SET	WORD
-----	-----	------

Set Associative Cache Organization



[Direct Mapping link](#)

[Associative Mapping Link](#)



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



White Board

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

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.

Problem 2(Word 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.

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 3



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												

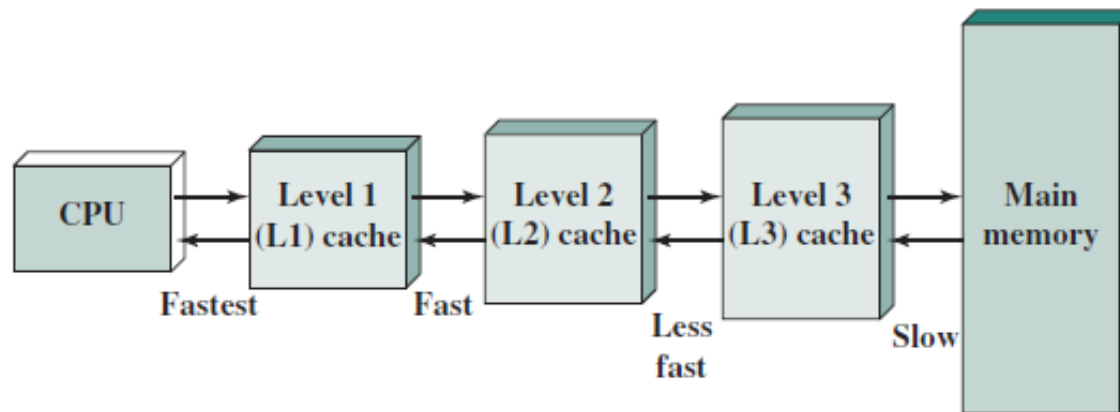
//

|

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

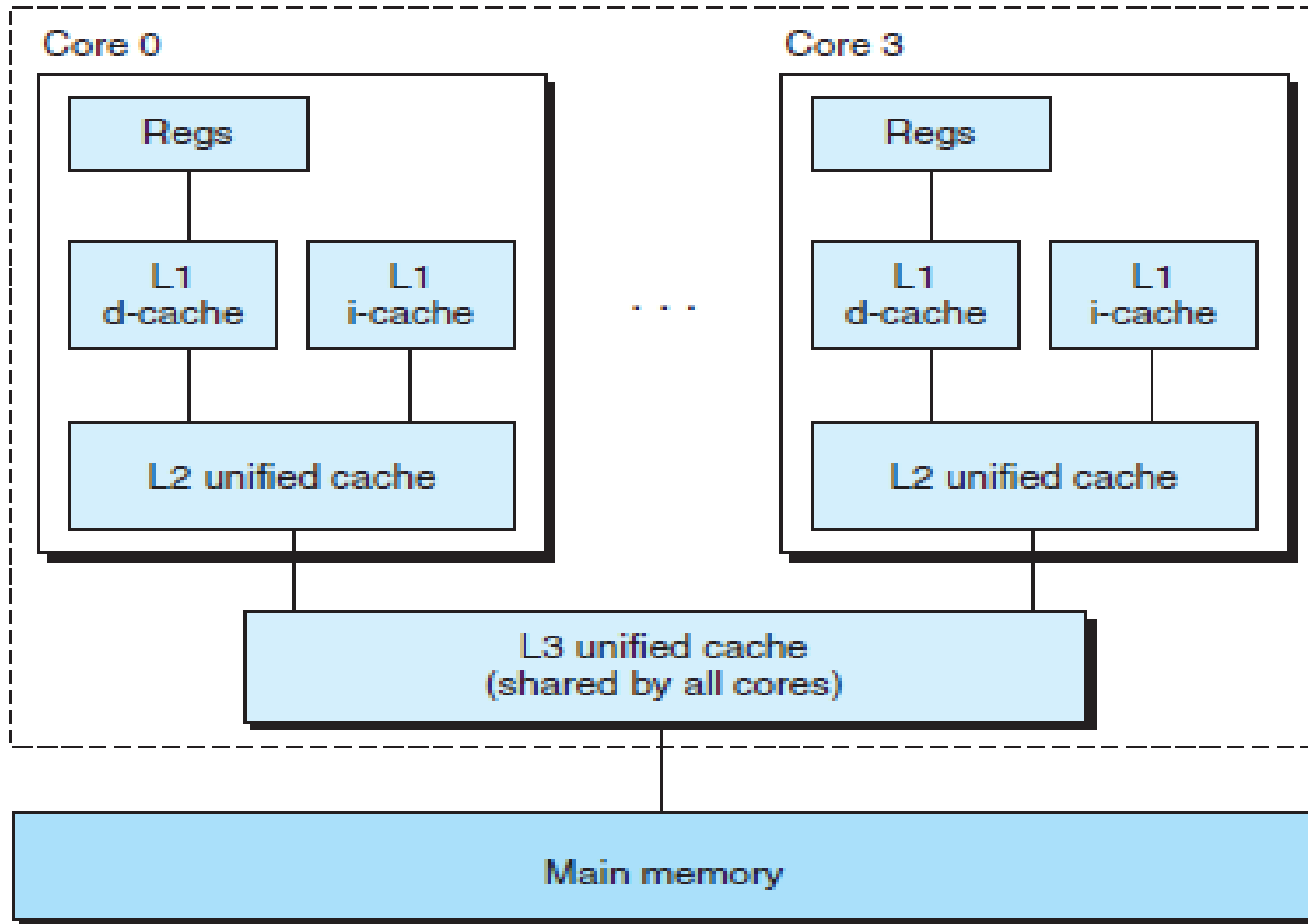


- What to do on a write-miss?
 - **Write-allocate** (load into cache, update line in cache)
 - Good if more writes to the location follow
 - **No-write-allocate** (writes straight to memory, does not load into cache)
- Typical
 - Write-through + No-write-allocate
 - **Write-back + Write-allocate**

Intel Core i7 Cache Hierarchy



Processor package



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

```

1  int sumvec(int v[N])
2  {
3      int i, sum = 0;
4
5      for (i = 0; i < N; i++)
6          sum += v[i];
7      return sum;
8  }

```

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

innovate

achieve

lead

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

Address difference
Stride=-----
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

```

1  int sumarrayrows(int a[M][N])
2  {
3      int i, j, sum = 0;
4
5      for (i = 0; i < M; i++)
6          for (j = 0; j < N; j++)
7              sum += a[i][j];
8      return sum;
9  }

```

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

```

1  int sumarraycols(int a[M][N])
2  {
3      int i, j, sum = 0;
4
5      for (j = 0; j < N; j++)
6          for (i = 0; i < M; i++)
7              sum += a[i][j];
8      return sum;
9  }

```



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				

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				

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

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];  
}
```

That's all
For Now
Folks

Problem 2 - LRU

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

HR = 5/12

0 3 2
1 1 1

Problem 2 - L~~LFU~~

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

HR = 5
12

0, 4, 2

Problem 2 - FIFO

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

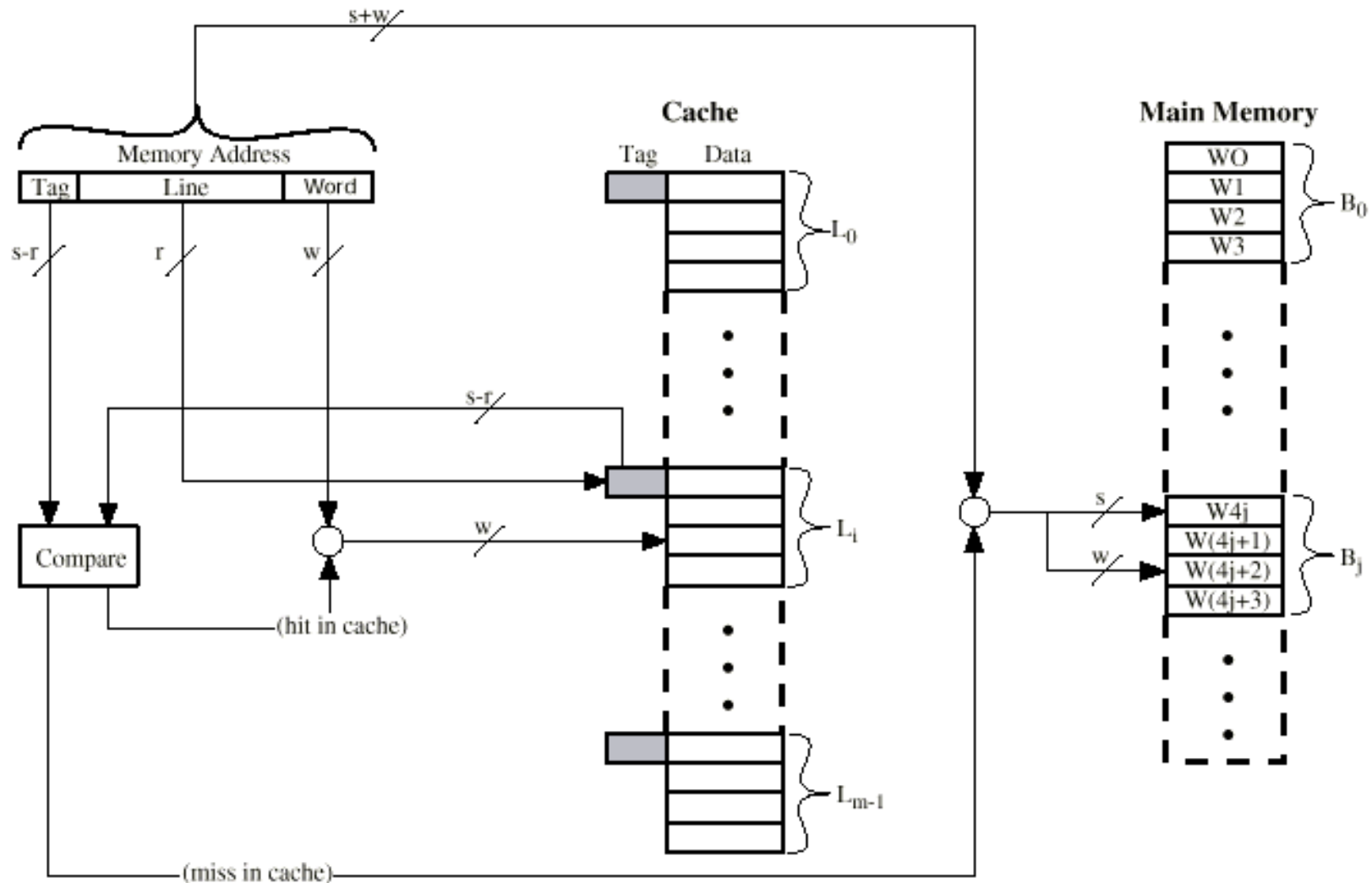
$$HR = \frac{4}{12}$$

18	10	3	1
----	----	---	---



Direct Mapping Cache Organization

[Back](#)





Associative Cache Organization

[Back](#)

