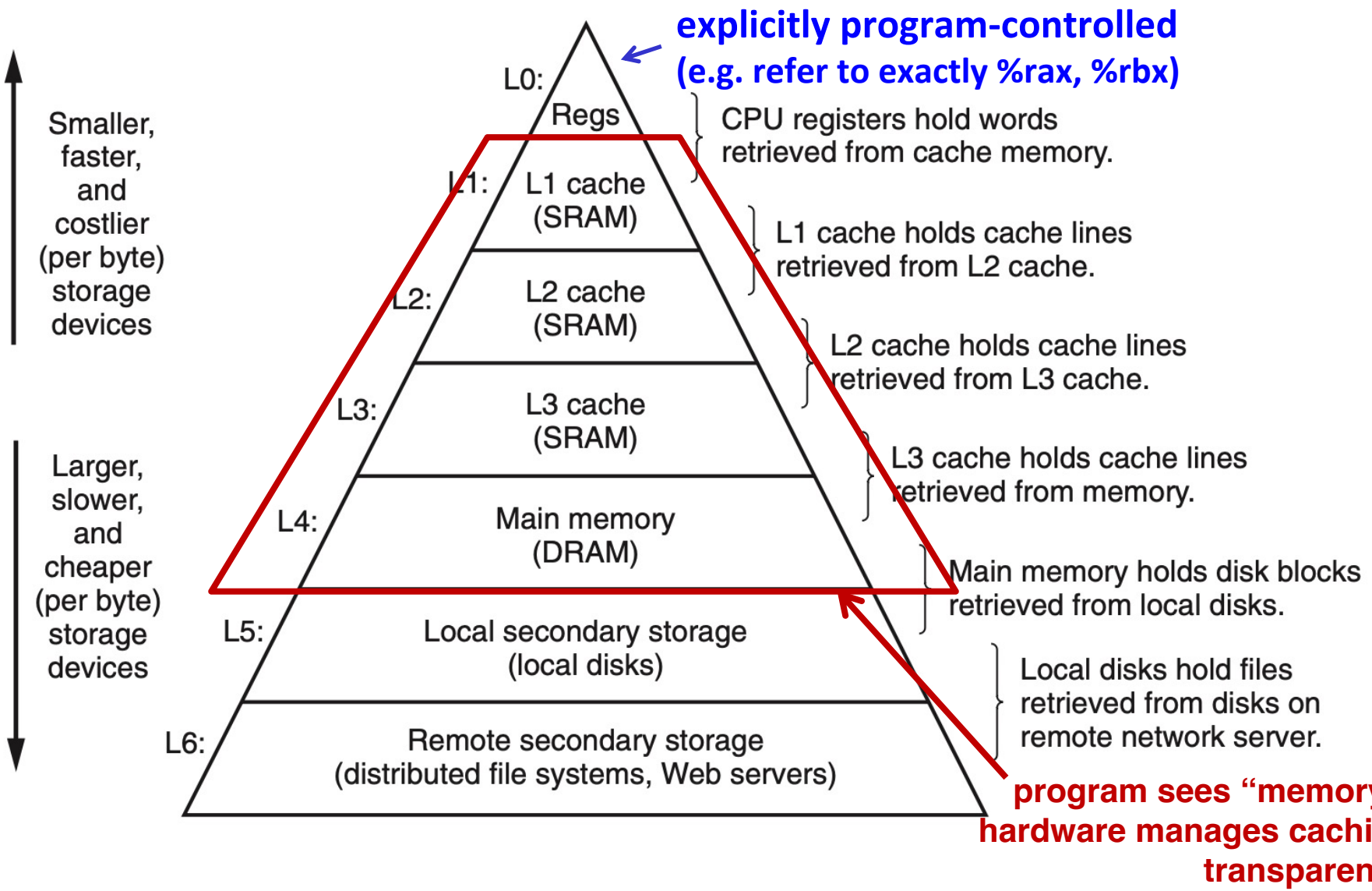
Lecture 09
Caches – part 2

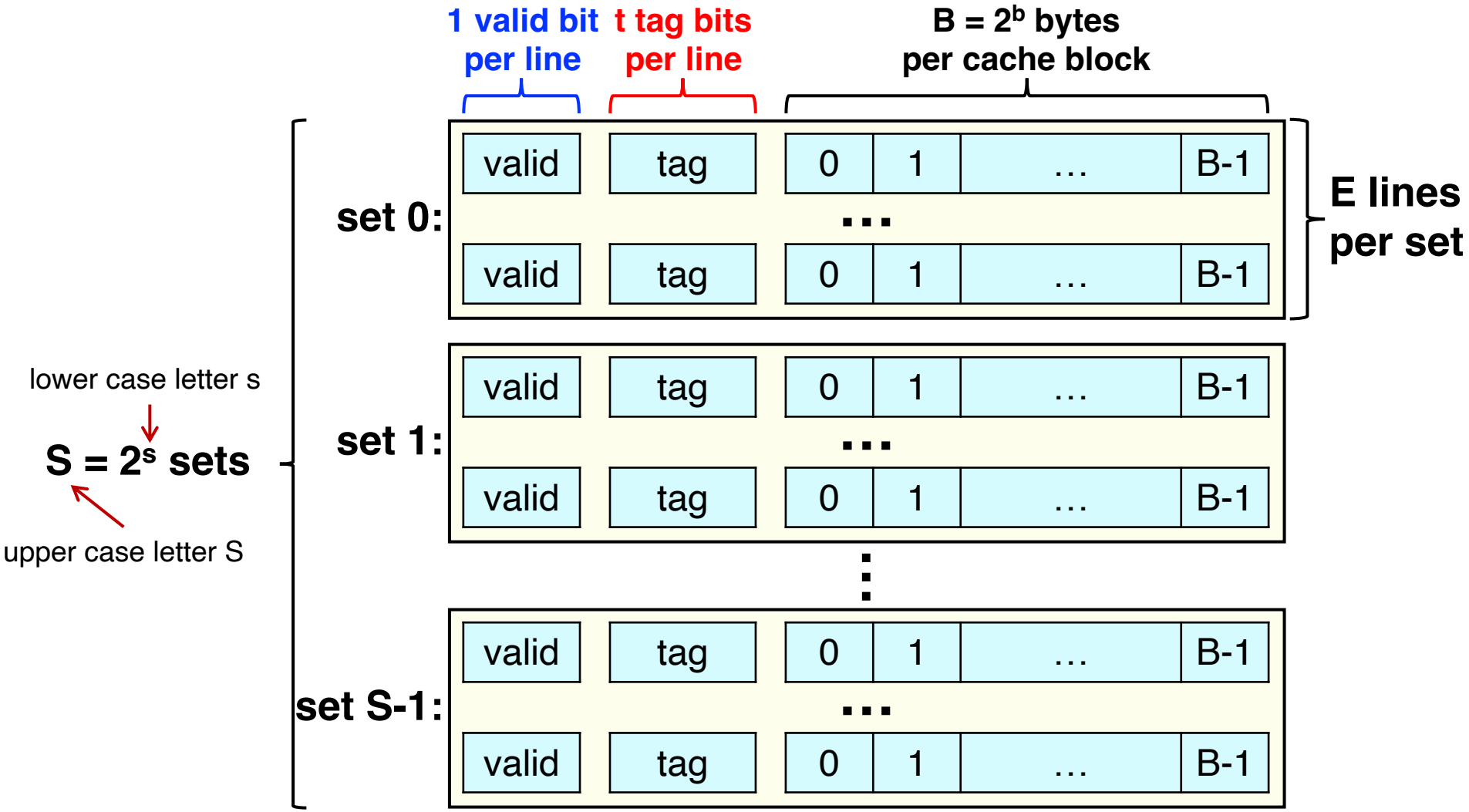
Euhyun Moon, Ph.D.
Machine Learning Systems (MLSys) Lab
Computer Science and Engineering
Sogang University

Slides adapted from Randy Bryant and Dave O'Hallaron: Introduction to Computer Systems, CMU

Review: Example Memory Hierarchy

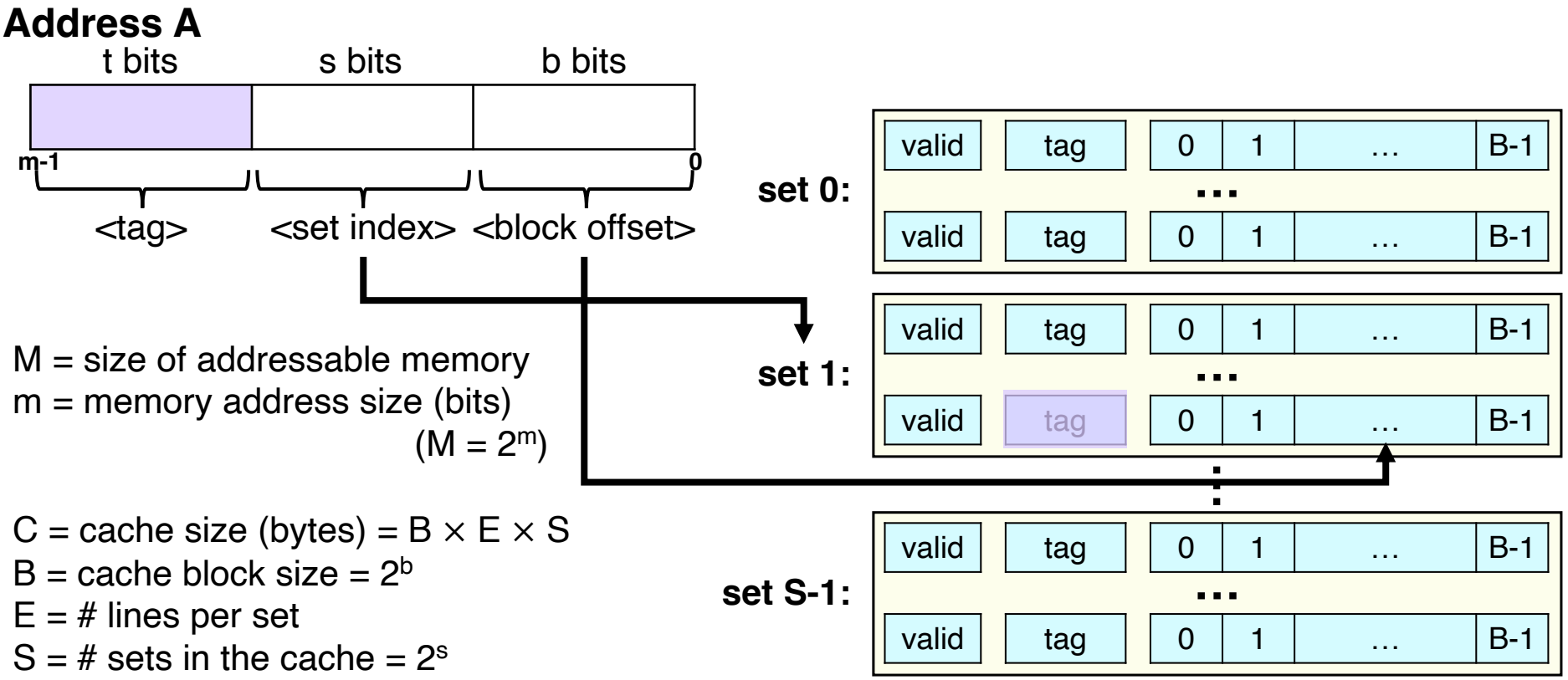


Review: Cache Structure



Cache size: C = B × E × S data bytes

Review: Cache Structure



M = size of addressable memory
m = memory address size (bits)
($M = 2^m$)

C = cache size (bytes) = B × E × S
B = cache block size = 2^b
E = # lines per set
S = # sets in the cache = 2^s

s = # bits for set offset
t = # tag bits
b = # bits for block offset

m = t + s + b

Making Memory Accesses Fast!

- Cache basics
- Principle of locality
- Memory hierarchies
- Cache structure
- **Cache mappings**
  - **Direct-mapped cache**
  - Set associative cache
  - Fully associative cache
- Cache performance metrics
- Cache-friendly code

Cache Mappings

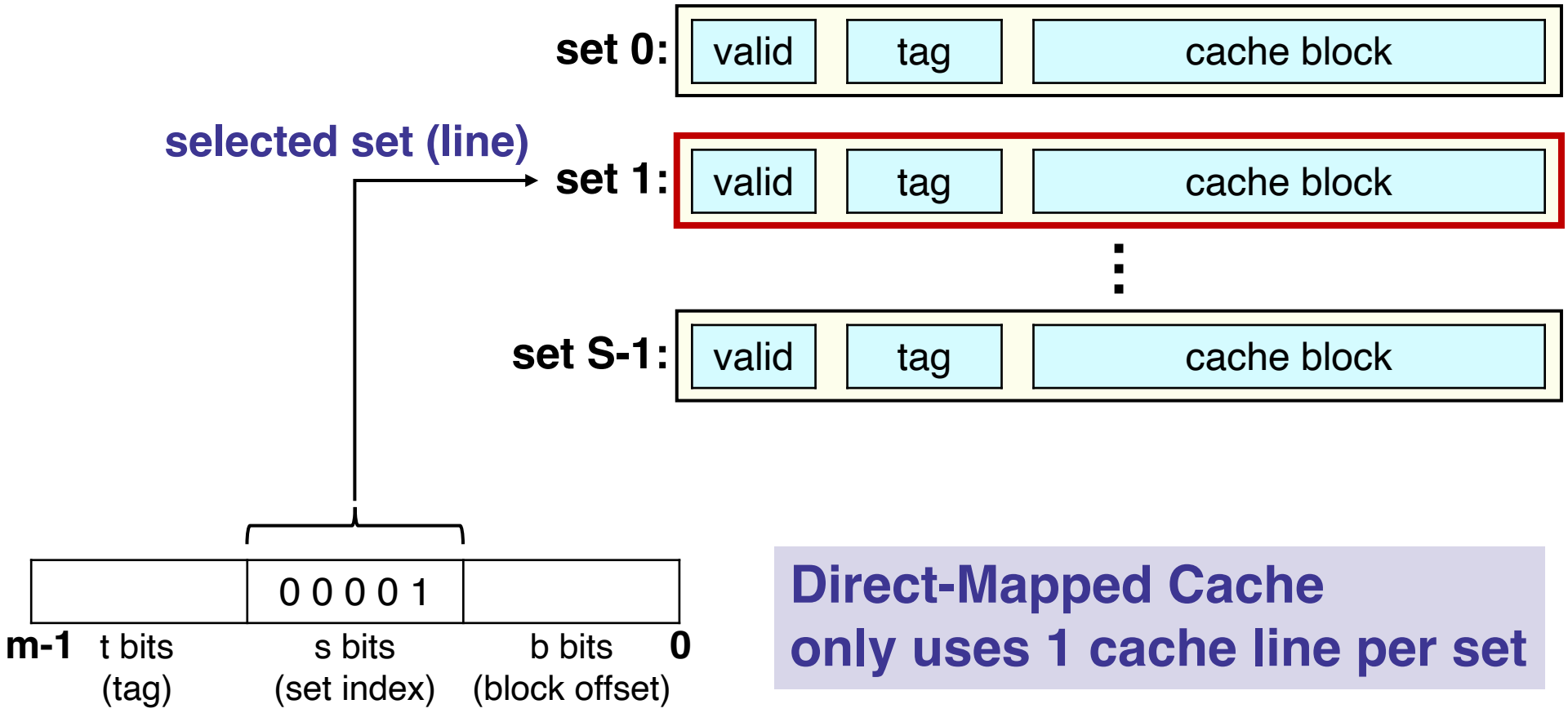
- Direct-mapped cache
- n-way set associative cache
- Fully associative cache

Direct-Mapped Cache

example	m	C	B	E	S	t	s	b
A	32	4096	8	1				
B	64	1024	8	2				
C	32	4096	64	32				
D	64	4096	128	16				

Accessing Direct-Mapped Caches

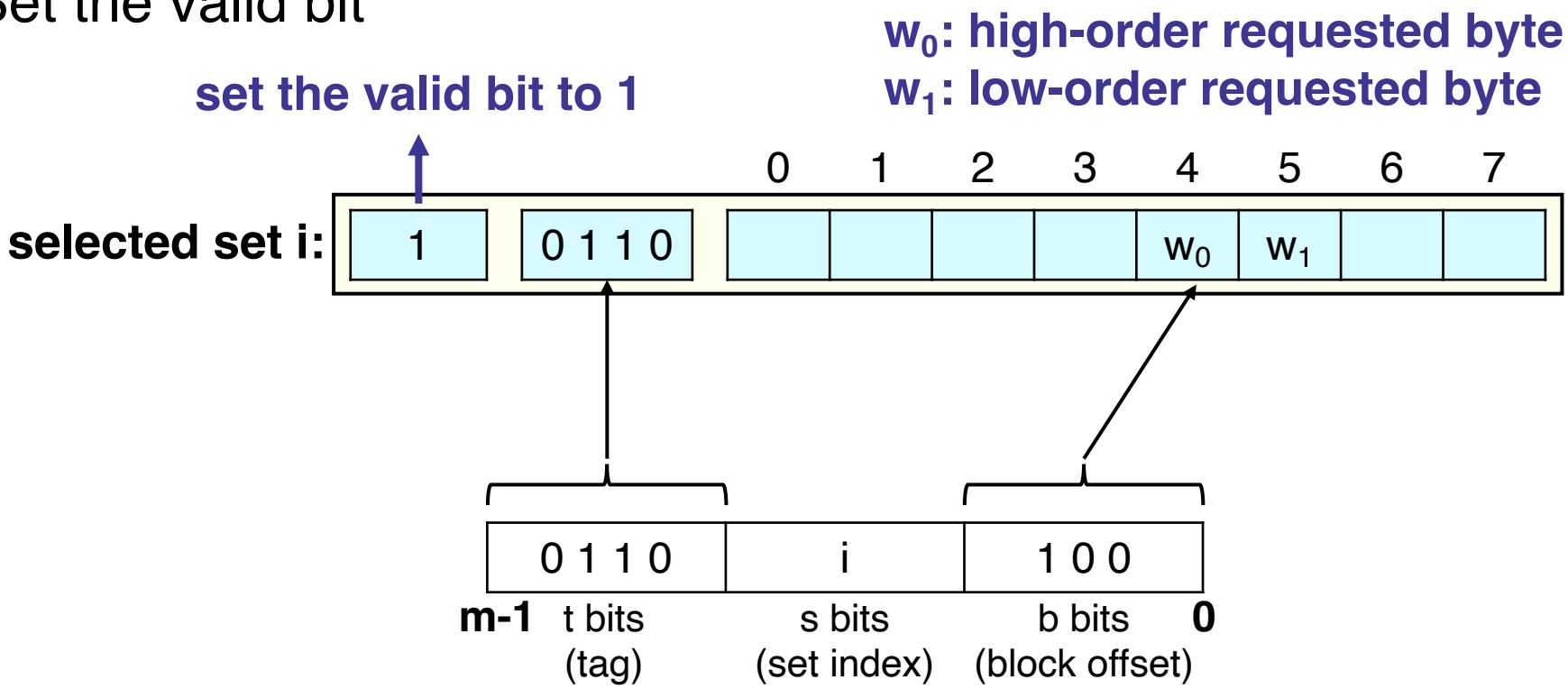
- Set selection
  - Use the set (= line) index bits to determine the set of interest



Direct-Mapped Cache only uses 1 cache line per set

Accessing Direct-Mapped Caches

- Update the tag bits
- Load the cache block
- Set the valid bit

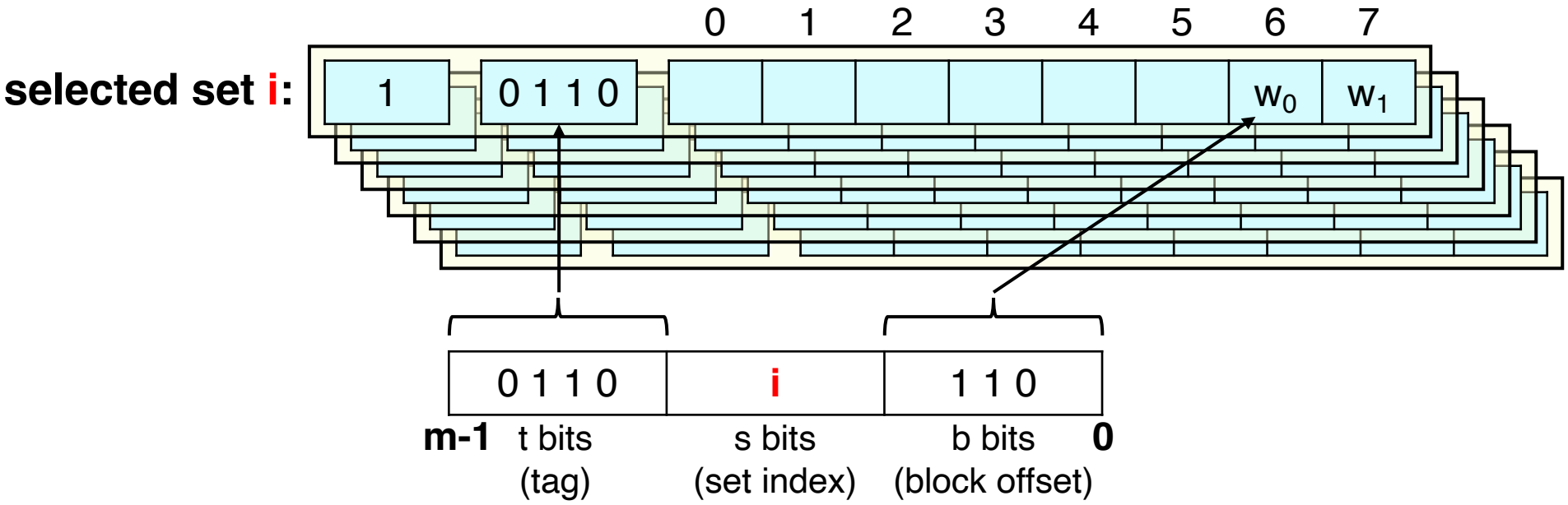


Cache Performance Metrics

- Huge difference between a cache hit and a cache miss
  - Could be 100x speed difference between accessing cache and main memory (measured in *clock cycles*)
- **Hit Rate**
  - Fraction of memory references found in cache
    - **cache hits / number of references**
- **Miss Rate**
  - Fraction of memory references not found in cache
    - **cache misses / number of references = 1 - Hit Rate**
  - Typical numbers:
    - 3-10% for L1
    - can be quite small (e.g., < 1%) for L2, depending on size, etc.
- **Hit Time**
  - Time to deliver a line (block) in the cache to the processor
    - includes time to determine whether the line is in the cache
  - Typical numbers:
    - 1 clock cycle for L1
    - 3-8 clock cycles for L2
- **Miss Penalty**
  - Additional time required because of a miss
    - Typically 25-100 cycles for main memory

Direct-Mapped Cache – cache hit

- Starting with the complete address for the next memory access:
  - Decode s bits to find set
  - Check valid bit
  - Match tag (now a hit)
  - Use block offset to locate data



Notations

- For some problems, the address and organization of data is important
  - For example:

sizeof(int) = 4	Specified the size in bytes of the given data type
int value_1 = 30	Specifies a variable of the given type along with its initial value. Note that the address of this variable is not specified
@value_1 = 0x8800200C	Specified the address, in main memory, of the given variable
@data_arr[512] = 0xAAAA0000	Specifies both the dimension of the given data structure, along with its base address

Hexadecimal Representation

- Hexadecimal
  - Frequently used in computer science applications because of the ease at which bytes (8 binary digits) can be encoded
  - Any byte value can be represented by two hexadecimal digits
  - Hexadecimal also has the convenience that each hex digit represent a count of some power of two

Hexadecimal	Decimal	Binary
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
A	10	1010
B	11	1011
C	12	1100
D	13	1101
E	14	1110
F	15	1111

$9_{16} = 9 \times 2^0$,
$30_{16} = 3 \times 2^4$,
$1376_{16} = 1\times2^{12} + 3\times2^8 + 7\times2^4 + 6\times2^0$
Kilobyte (1024 bytes = 400_{16})
Megabyte (1,048580 bytes = $100{,}000_{16}$)

Hexadecimal	Binary string
AA	1010 1010
123	0001 0010 0011
FA71	1111 1010 0111 0001

Class Activity – 9.2

problem	m	C	B	E	S	t	s	b
1	32	4096	8	1	512	20	9	3

sizeof(int)	= 4
data path width	= B
int value_1	= 30
@value_1	= 0x8800200C

Assuming a cache miss,
what is loaded into cache, and where, when value_1 is accessed?

Class Activity – 9.3

problem	m	C	B	E	S	t	s	b
1	32	4096	8	1	512	20	9	3

sizeof(int)	= 4
data path width	= B
int valueA[256]	= 30
@valueA[256]	= 0x88002008

```
for ( i = 0; i < 256; i++ )
{
      sum += valueA[i];
}
```

Assume that sum is a register variable.
What is loaded into cache, and where, as the elements of valueA[] are accessed?

Class Activity – 9.4

problem	m	C	B	E	S	t	s	b
1	32	4096	16	1	256	20	8	4

sizeof(int)	= 4
data path width	= B
int valueA[2048]	= whatever
@valueA[2048]	= 0x88002008

```
for ( i = 0; i < 1024; i++ )
{
      sum += valueA[i];
}
```

Assume that sum is a register variable.
What is loaded into cache, and where, as the elements of valueA[] are accessed?
How many main memory accesses are performed?

Direct-Mapped Cache

example	m	C	B	E	S	t	s	b
1	32	1024	4	1	256	22	8	2

main memory address:	AAAA0000
tag:	1010 1010 1010 1010 0000 00
set index:	0000 0000
block offset:	00

This will map to set 0 with offset 0.
This will load addresses AAAA0000 – AAAA0003 into cache

Cache Lines

example	m	C	B	E	S	t	s	b
1	32	1024	4	1	256	22	8	2

addresses within the block range exist on same cache line

access one byte:

main memory address:	AAAA0000
tag:	1010 1010 1010 1010 0000 00
set index:	0000 0000
block offset:	00

this will map to set 0 with offset 0
this will load addresses AAAA0000 – AAAA0003 into cache

access one byte:

main memory address:	AAAA0001
tag:	1010 1010 1010 1010 0000 00
set index:	0000 0000
block offset:	01

this will map to set 0 with offset 1
this will load addresses AAAA0000 – AAAA0003 into cache

this also holds for addresses AAAA0002 and AAAA0003

Set Bit Operation

example	m	C	B	E	S	t	s	b
1	32	1024	4	1	256	22	8	2

as set bits change, cache line moves to next set

main memory address:	AAAA0000	tag:	1010 1010 1010 1010 0000 00
		set index:	0000 0000
		block offset:	00
	set 0 with offset 0		

main memory address:	AAAA0001-3	tag:	1010 1010 1010 1010 0000 00
		set index:	0000 0000
		block offset:	01 – 11
	set 0 with offset 1 – 3		

main memory address:	AAAA0004	tag:	1010 1010 1010 1010 0000 00
		set index:	0000 0001
		block offset:	00
	set 1 with offset 0		

this is good for spatial locality

Set Warp-Around

example	m	C	B	E	S	t	s	b
1	32	1024	4	1	256	22	8	2

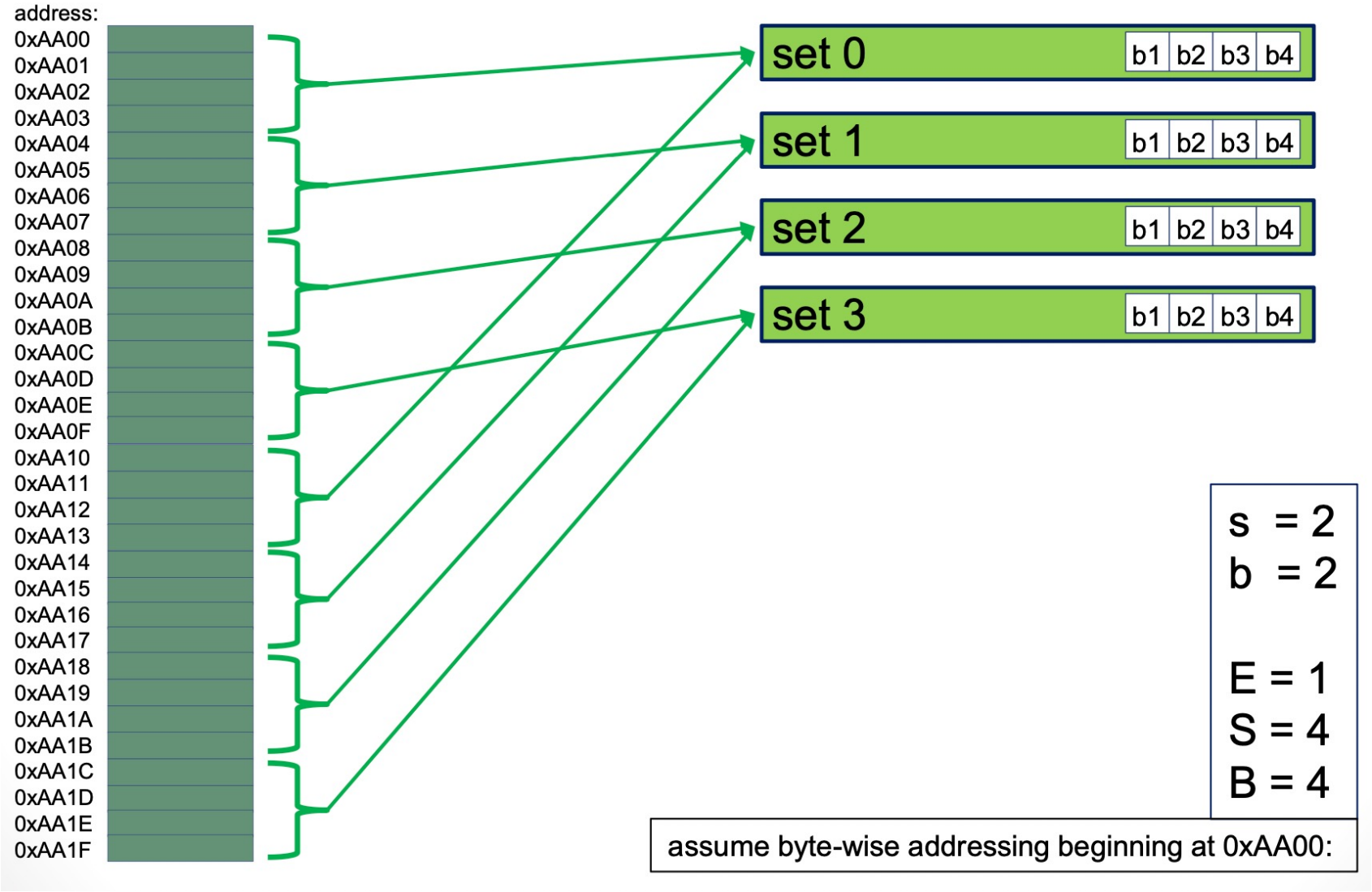
addresses which share same set bits will over-write each other

main memory address:	AAAA0000	tag:	1010 1010 1010 1010 0000 00
		set:	0000 0000
		offset:	00
set 0 with offset 0			

⋮

main memory address:	AAAA**0400**	tag:	1010 1010 1010 1010 0000 01
		set:	0000 0000
		offset:	00
set 0 with offset 0			

Example: Set Warp-Around



Hit Rate Example

example	m	C	B	E	S	t	s	b
1	32	1024	4	1	256	22	8	2

read address:	AAAA0000	cold miss	will load AAAA0000 – AAAA0003
read address:	AAAA0001	cache hit	
read address:	AAAA0002	cache hit	
read address:	AAAA0003	cache hit	
read address:	AAAA0004	cold miss	will load AAAA0004 – AAAA0007

⋮

"hit rate" will be $\frac{3}{4}$ = 75%

Direct-Mapped Cache

example	m	C	B	E	S	t	s	b
1	32	1024	4	1	256	22	8	2

```
for ( i = 0; i < 1000000; i++ )
{
      sum += my_array[i];
}
```

Assume that

sizeof(int) = 2 (bytes) ← size of data items

@my_array[1000000] = AAAA0000

sum is a register variable

What percentage of memory accesses are cache hits?

Direct-Mapped Cache

example	m	C	B	E	S	t	s	b
1	32	1024	4	1	256	22	8	2

```
for ( i = 0; i < 1000000; i++ )
{
      sum += my_array[i];
}
```

Assume that

sizeof(int) = 2 (bytes) ← size of data items

@my_array[1000000] = AAAA0000

sum is a register variable

What percentage of memory accesses are cache hits?

50% ← affected cache performance

Direct-Mapped Cache

example	m	C	B	E	S	t	s	b
2	32	1024	4	1	256	22	8	2

```
for ( i = 0; i < 1000000; i += 2 )
{
      sum += my_array[i];
}
```

very slight change
should be less work, right?

What percentage of memory accesses are cache hits?

Direct-Mapped Cache

example	m	C	B	E	S	t	s	b
2	32	1024	4	1	256	22	8	2

```
for ( i = 0; i < 1000000; i += 2 )
{
      sum += my_array[i];
}
```

very slight change
should be less work, right?

What percentage of memory accesses are cache hits?

0%

Class Activity – 9.5

```
for ( i = 999999; i >= 0; i-- )
{
      sum += my_array[i];
}
```

Assume that

@my_array[1000000] = AAAA0000

sum is a register variable

problem	m	C	B	E	S	t	s	b
1	32	4096	16	1	?	?	?	?

what are S, t, s and b?
let sizeof(int) = 2 (bytes), what percentage of memory accesses are cache hits?
what was the effect of going backward?

problem	m	C	B	E	S	t	s	b
2	64	4096	16	1	?	?	?	?

what are S, t, s and b?
let sizeof(int) = 4 (bytes), what percentage of memory accesses are cache hits?
what was the impact of the change to 64 bits?
what was the primary cause of the change in hit rate?

Class Activity – 9.6

problem	m	C	B	E	S	t	s	b
1	32	1024	4	1	256	22	8	2

```
for ( i = 0; i < 1024; i++ )
{
      sum += my_array[ i ][ 0 ];
}
```

Assume that

sizeof(short int) = 1 (bytes)

@my_array[1024][1024] = AAAA0000 (assume contiguous)

sum is a register variable

What percentage of memory accesses are cache hits?