CS354: Machine Organization and Programming

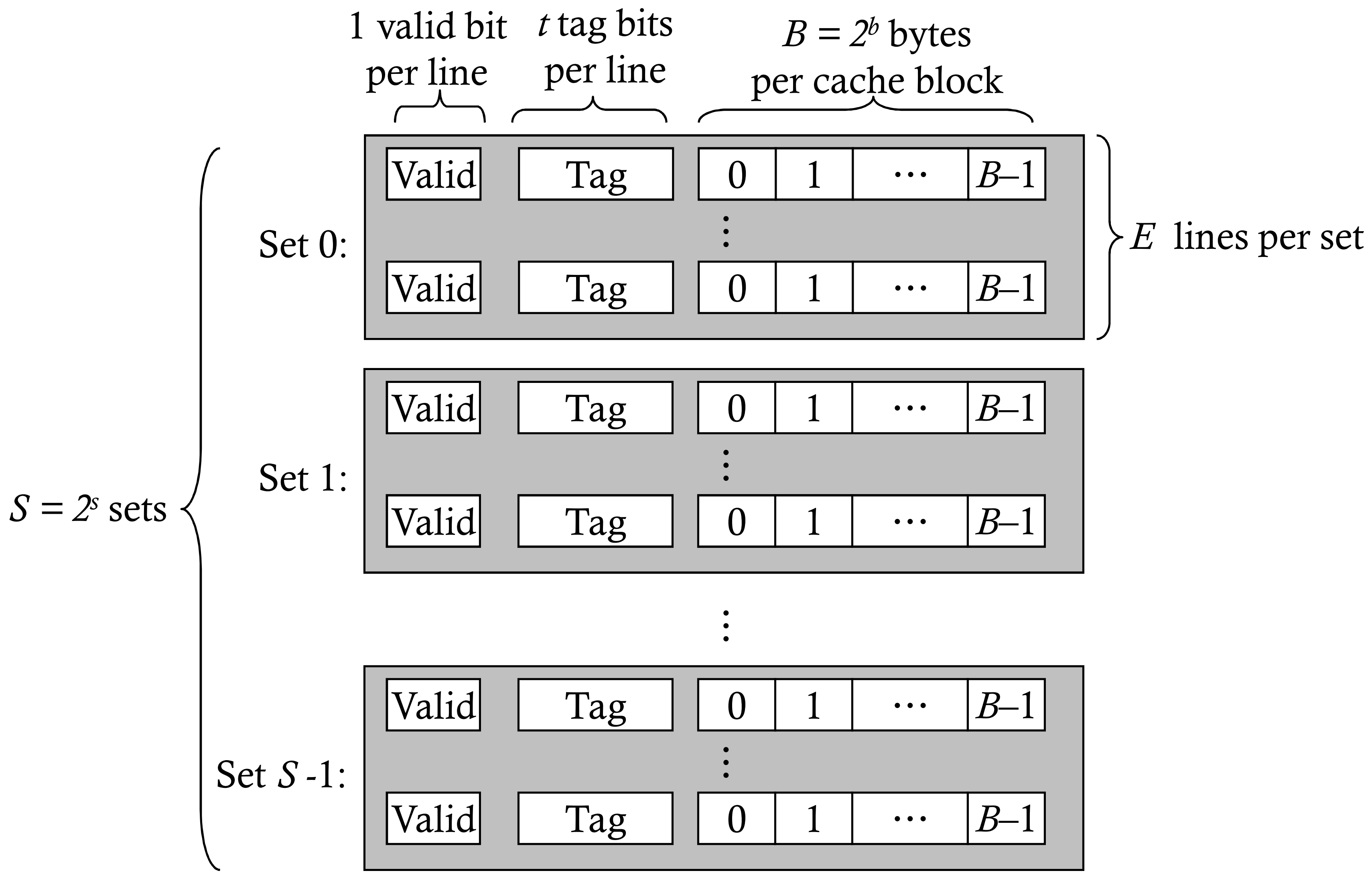
Lecture 23
Monday the October 26th 2015

Section 2
Instructor: Leo Arulraj

Lecture Overview

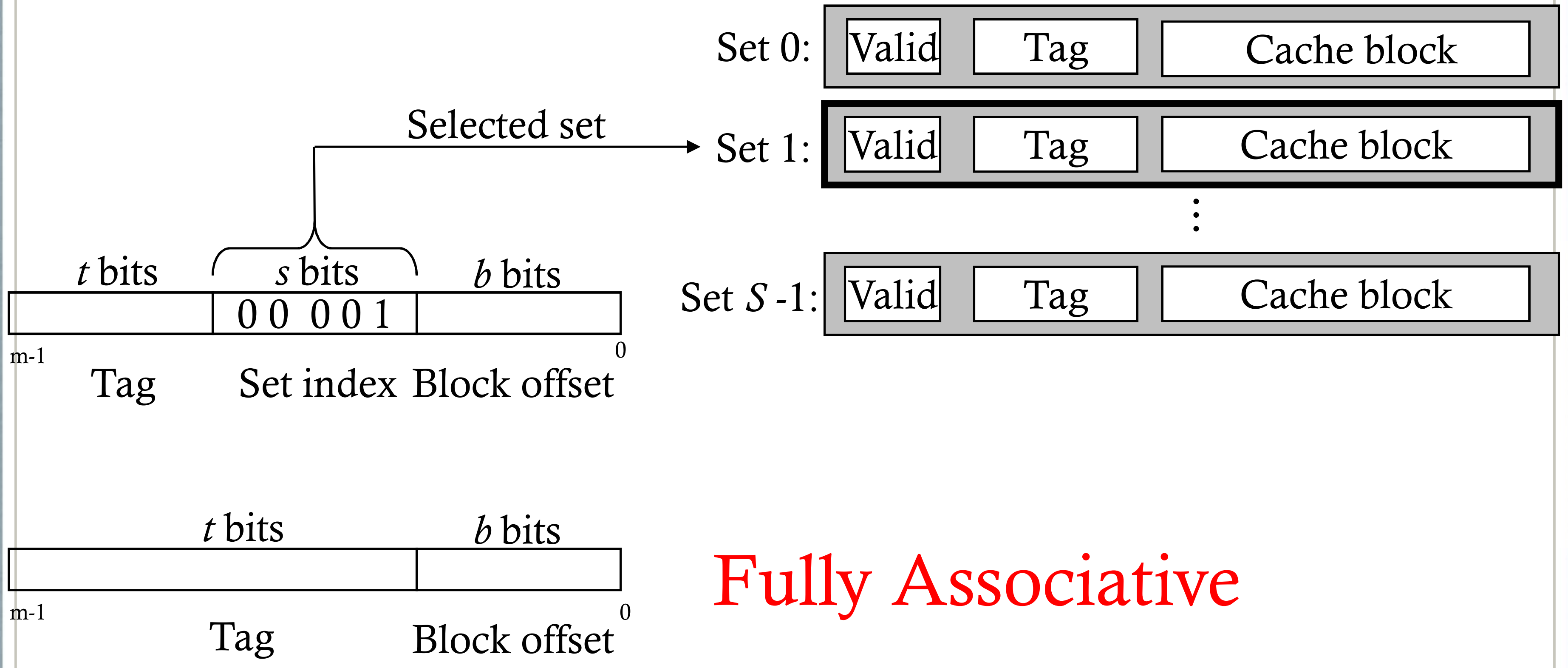
1. Review of Caches
2. Example Problems

Generic Cache Organization



Cache size: $C = B \times E \times S$ data bytes

Looking up a memory address



Fully Associative

Cache Lookup

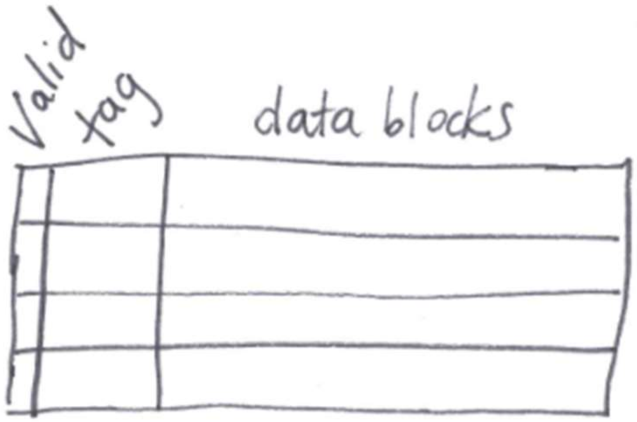
Three steps while determining whether a request is a hit or a miss:

- Set selection: Select the set where the address resides.
- Line matching: Select the cache line within the set.
- Word extraction: Extract the requested word from the right offset.

Lookup algorithm:

(cache receives address)

```
use index to identify frame
if frame is valid
    if frame's tag matches
    address' tag
        HIT
    else
        MISS
else
    MISS
```



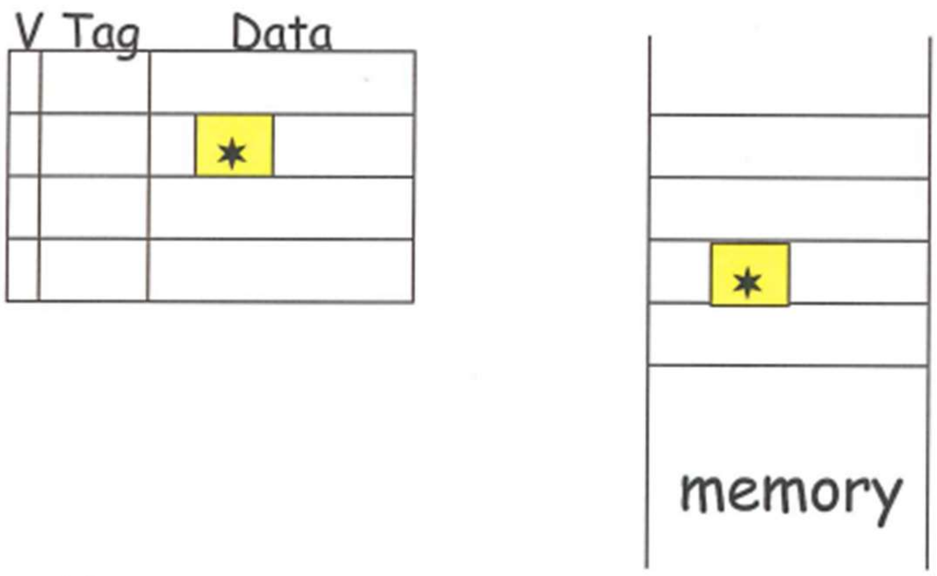
Types of Misses

- **Compulsory or cold misses:** Cache is empty to start with and will miss.
- **Conflict misses:** Cache has space but because objects map to the same cache block they keep missing.
- **Capacity misses:** Cache does not have space because size of the working set exceeds the size of the cache.

Cache Replacement Policies

- Which block to replace or evict to make space for new blocks?
  - Random Replacement Policy: chooses a random victim block.
  - Least Recently Used (LRU) Policy: chooses the block that was last accessed furthest in the past.
  - Least Frequently Used (LFU) Policy: chooses the block that was least frequently accessed in the past.

Implementing <u>writes</u>



1 *write through*

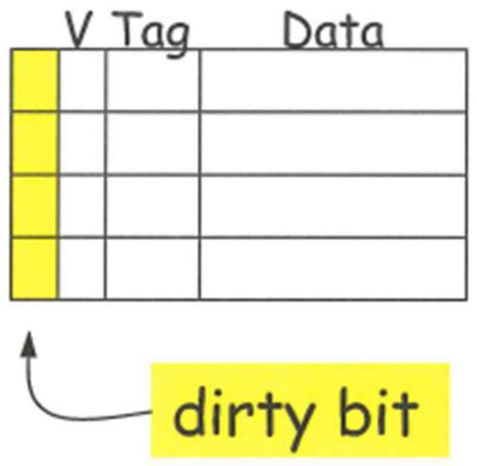
change data in the cache, <u>and</u> send the write to main memory

slow ☹ , but very little circuitry ☺

2 *write back*

- at first, change data in the cache
- write to memory only when necessary

dirty bit is set on a write, to identify blocks to be written back to memory



when a program completes, all dirty blocks must be written to memory. . .

2 *write back (continued)*

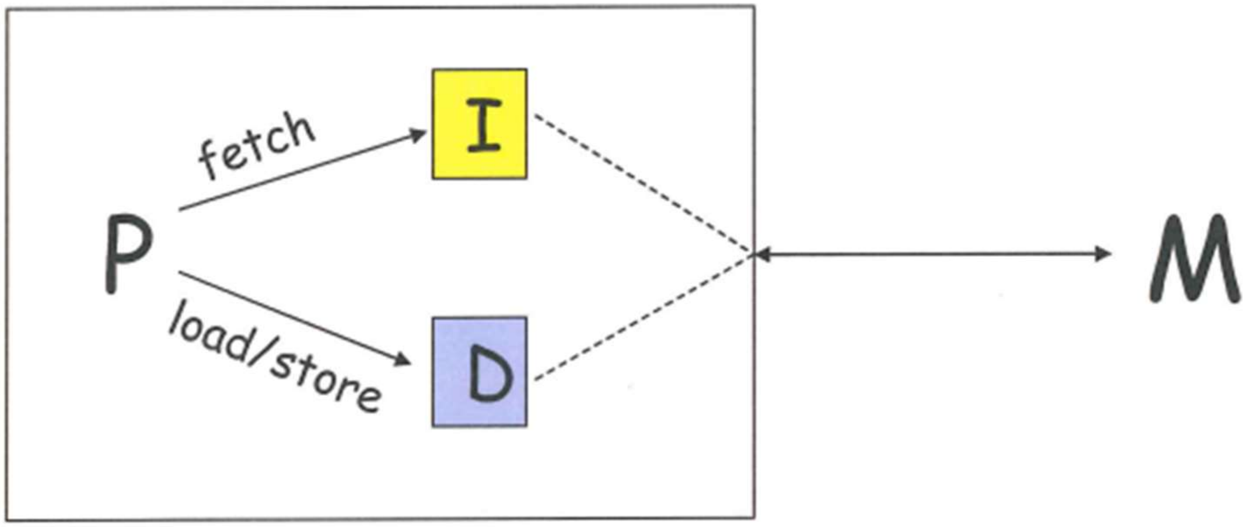
- faster ☺
  multiple stores to the same location result in only 1 main memory access
- more circuitry ☹
  - must maintain the dirty bit
  - *dirty miss* : a miss caused by a read or write to a block not in the cache, but the required block frame has its dirty bit set. So, there is a write of the dirty block, followed by a read of the requested block.

Writing during cache miss: (Two approaches)

- Write Alloc: Load block in cache and update word (often used along with Write back)
- Write No-Alloc (a.k.a.Write around): Just update memory (often used along with Write through)

We can send memory accesses to the 2 caches independently. . .

☺ (increased parallelism)



Suppose we have a system with the following properties:

- The memory is byte addressable.
- Memory accesses are to **1-byte words** (not to 4-byte words).
- Addresses are 12 bits wide.
- The cache is two-way set associative ($E = 2$), with a 4-byte block size ($B = 4$) and four sets ($S = 4$).

The contents of the cache are as follows, with all addresses, tags, and values given in hexadecimal notation:

Set index	Tag	Valid	Byte 0	Byte 1	Byte 2	Byte 3
0	00	1	40	41	42	43
	83	1	FE	97	CC	D0
1	00	1	44	45	46	47
	83	0	—	—	—	—
2	00	1	48	49	4A	4B
	40	0	—	—	—	—
3	FF	1	9A	C0	03	FF
	00	0	—	—	—	—

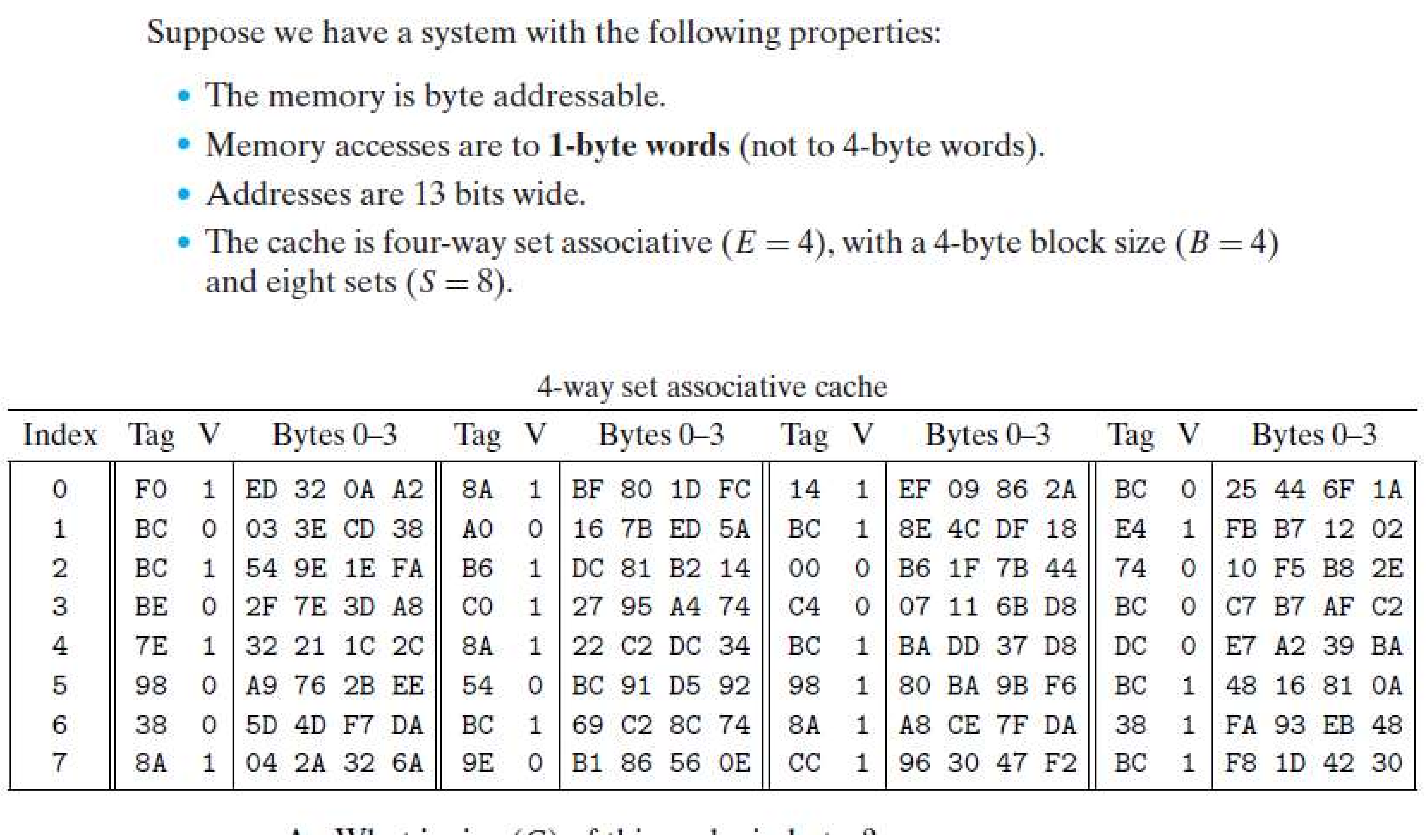
A. The following diagram shows the format of an address (one bit per box). Indicate (by labeling the diagram) the fields that would be used to determine the following:

CO	The cache block offset
CI	The cache set index
CT	The cache tag

11	10	9	8	7	6	5	4	3	2	1	0

B. For each of the following memory accesses indicate if it will be a cache hit or miss when **carried out in sequence** as listed. Also give the value of a read if it can be inferred from the information in the cache.

Operation	Address	Hit?	Read value (or unknown)
Read	0x834		
Write	0x836		
Read	0xFFD		



Suppose we have a system with the following properties:

- The memory is byte addressable.
- Memory accesses are to **1-byte words** (not to 4-byte words).
- Addresses are 13 bits wide.
- The cache is four-way set associative ($E = 4$), with a 4-byte block size ($B = 4$) and eight sets ($S = 8$).

4-way set associative cache

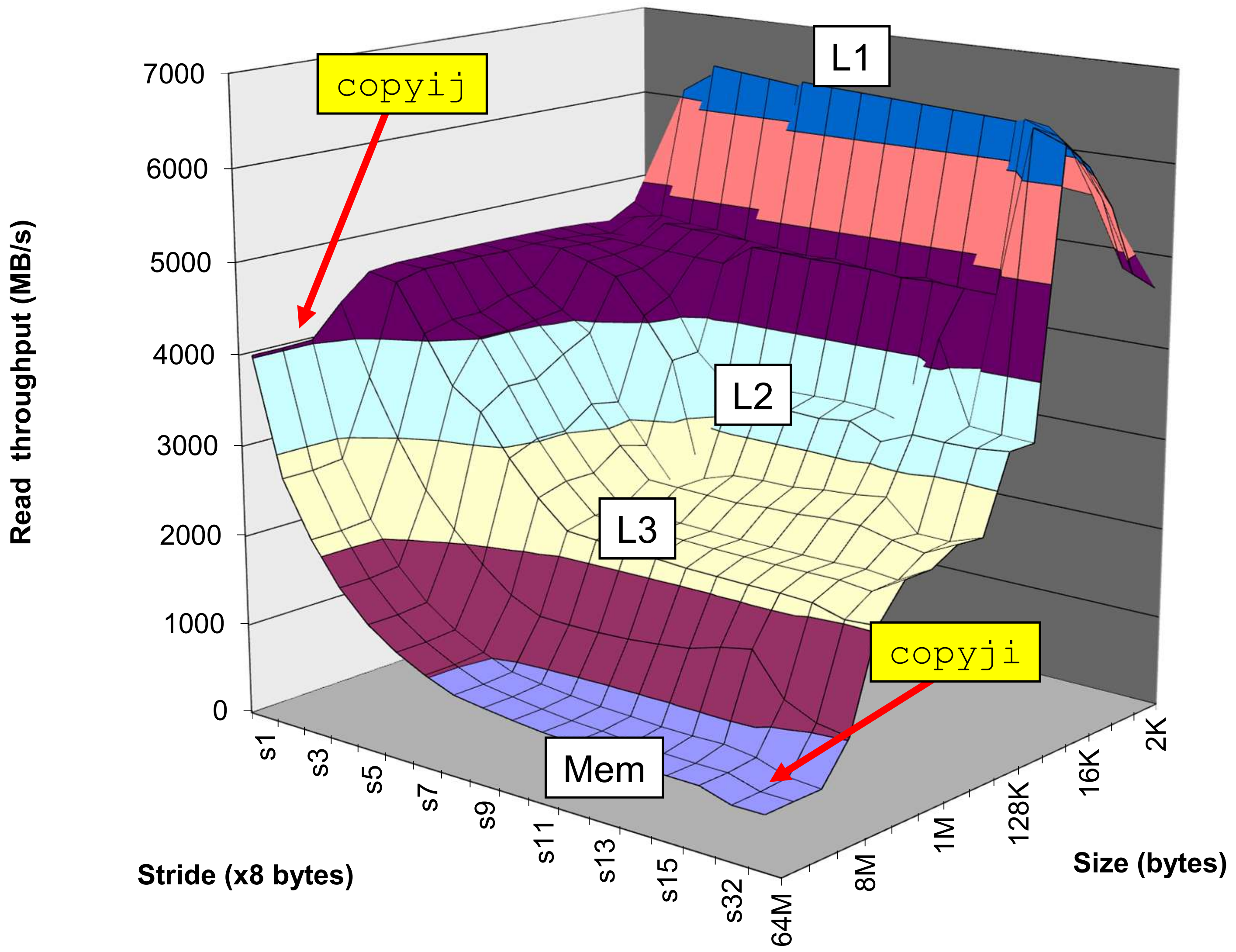
Index	Tag	V	Bytes 0–3	Tag	V	Bytes 0–3	Tag	V	Bytes 0–3	Tag	V	Bytes 0–3
0	F0	1	ED 32 0A A2	8A	1	BF 80 1D FC	14	1	EF 09 86 2A	BC	0	25 44 6F 1A
1	BC	0	03 3E CD 38	A0	0	16 7B ED 5A	BC	1	8E 4C DF 18	E4	1	FB B7 12 02
2	BC	1	54 9E 1E FA	B6	1	DC 81 B2 14	00	0	B6 1F 7B 44	74	0	10 F5 B8 2E
3	BE	0	2F 7E 3D A8	C0	1	27 95 A4 74	C4	0	07 11 6B D8	BC	0	C7 B7 AF C2
4	7E	1	32 21 1C 2C	8A	1	22 C2 DC 34	BC	1	BA DD 37 D8	DC	0	E7 A2 39 BA
5	98	0	A9 76 2B EE	54	0	BC 91 D5 92	98	1	80 BA 9B F6	BC	1	48 16 81 0A
6	38	0	5D 4D F7 DA	BC	1	69 C2 8C 74	8A	1	A8 CE 7F DA	38	1	FA 93 EB 48
7	8A	1	04 2A 32 6A	9E	0	B1 86 56 0E	CC	1	96 30 47 F2	BC	1	F8 1D 42 30

Analyze memory references 0x071A and 0x16E8

Strided Access Patterns

```
int i, j, sum =0;
for(i=0;i<16;i++)
    for(j=0;j<16;j++)
        sum += a[i][j]
```

What if: sum += a[j][i] ?



Memory Mountain from the CSAPP textbook