Systems I

Cache Organization

Topics

- Generic cache memory organization
- Direct mapped caches
- Set associative caches
- Impact of caches on programming

Cache Vocabulary

Capacity

Cache block (aka cache line)

Associativity

Cache set

Index

Tag

Hit rate

Miss rate

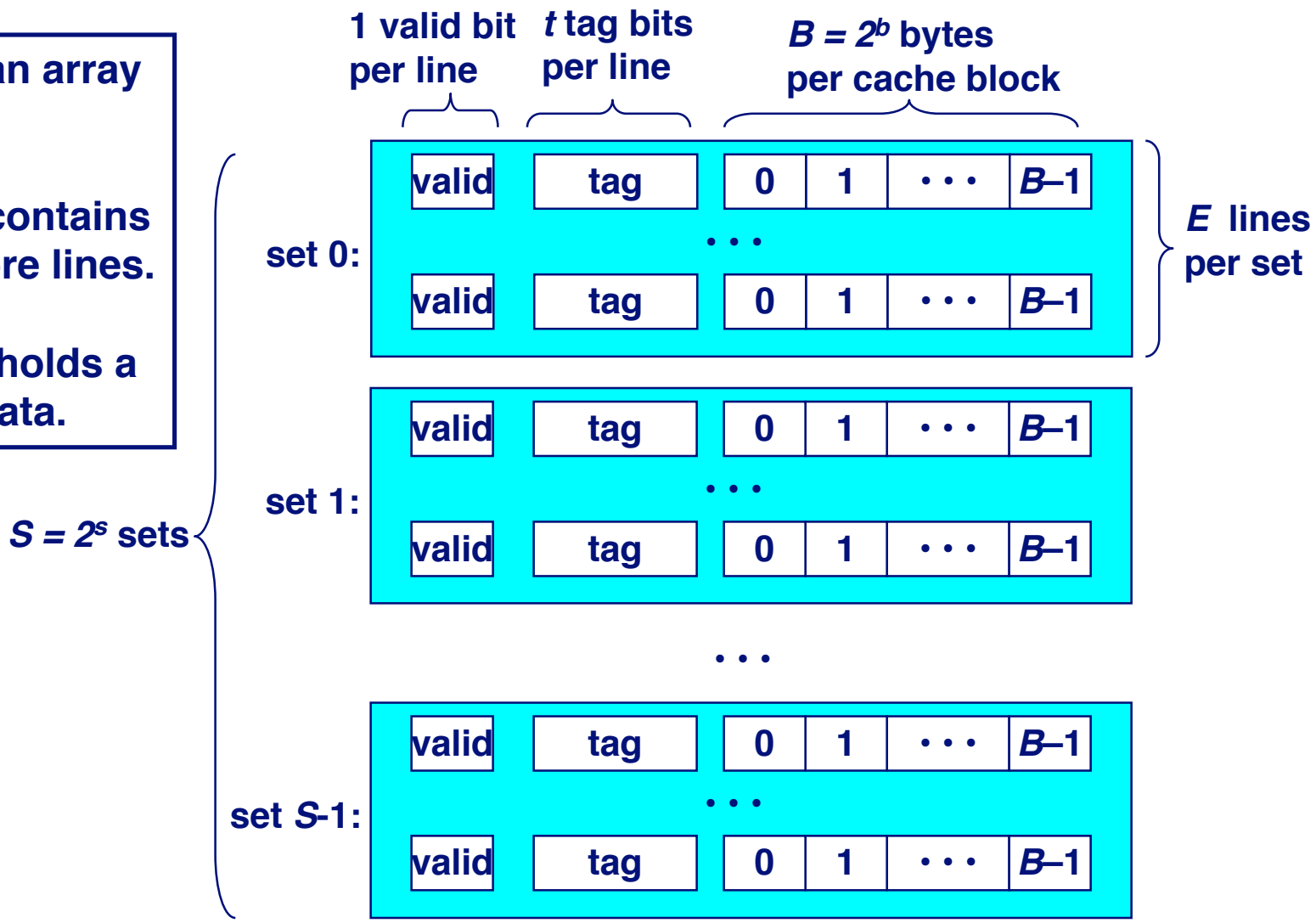
Replacement policy

General Org of a Cache Memory

Cache is an array of sets.

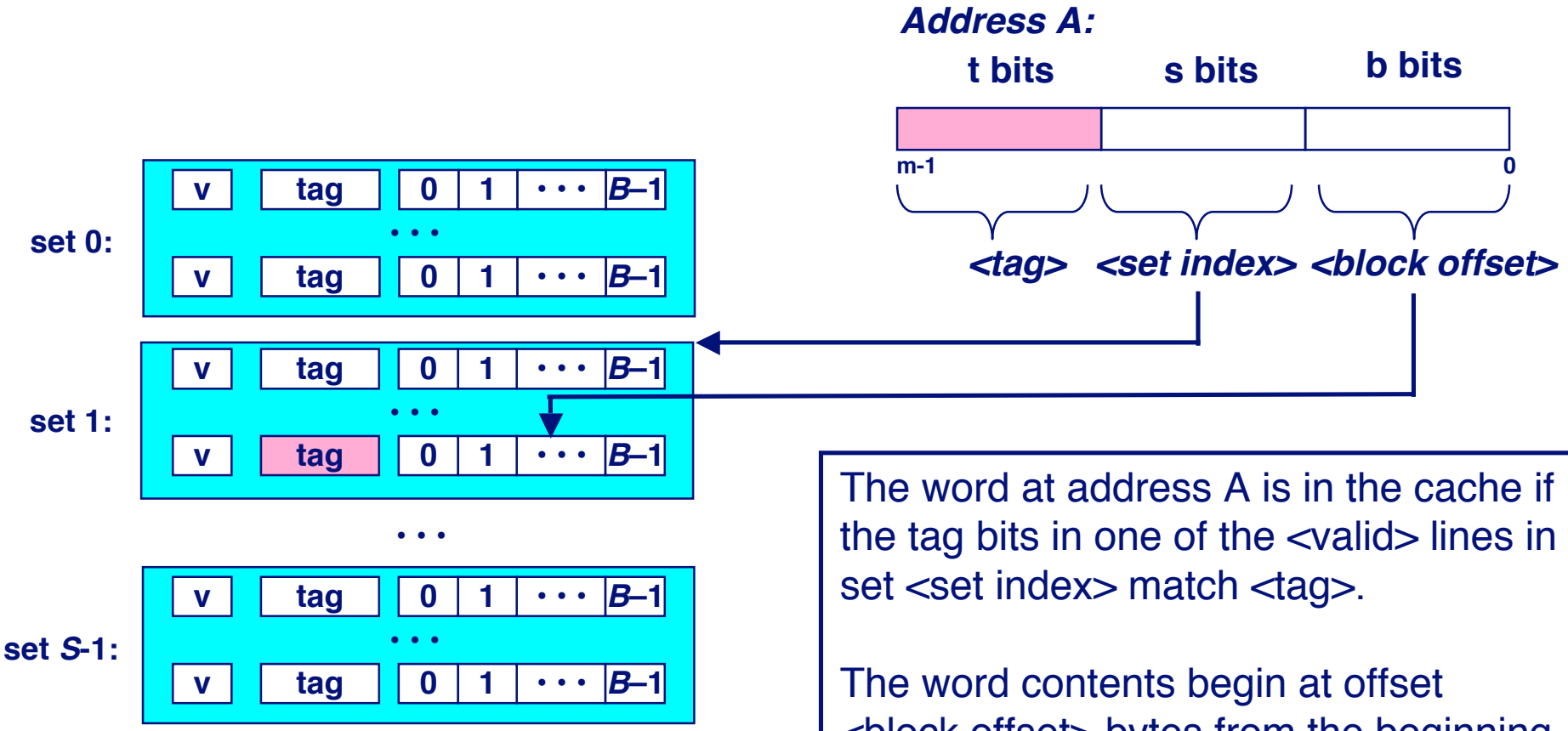
Each set contains one or more lines.

Each line holds a block of data.



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

Addressing Caches



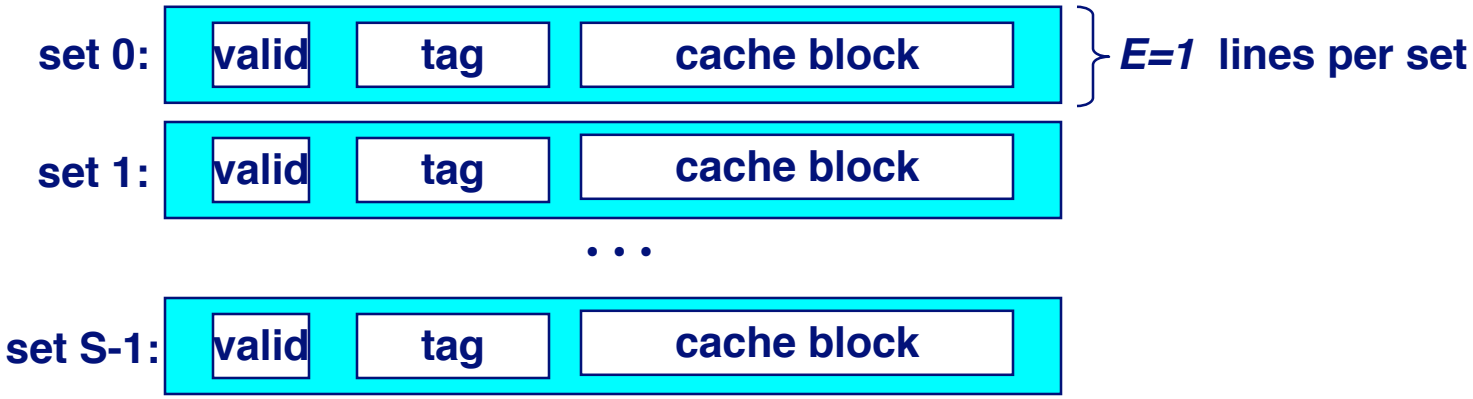
The word at address A is in the cache if the tag bits in one of the <valid> lines in set <set index> match <tag>.

The word contents begin at offset <block offset> bytes from the beginning of the block.

Direct-Mapped Cache

Simplest kind of cache

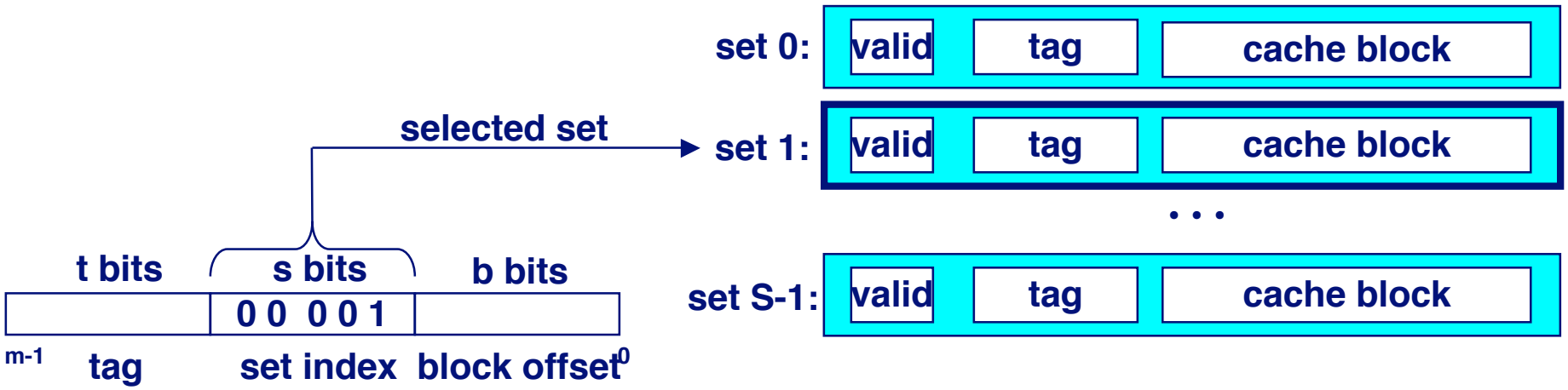
Characterized by exactly one line per set.



Accessing Direct-Mapped Caches

Set selection

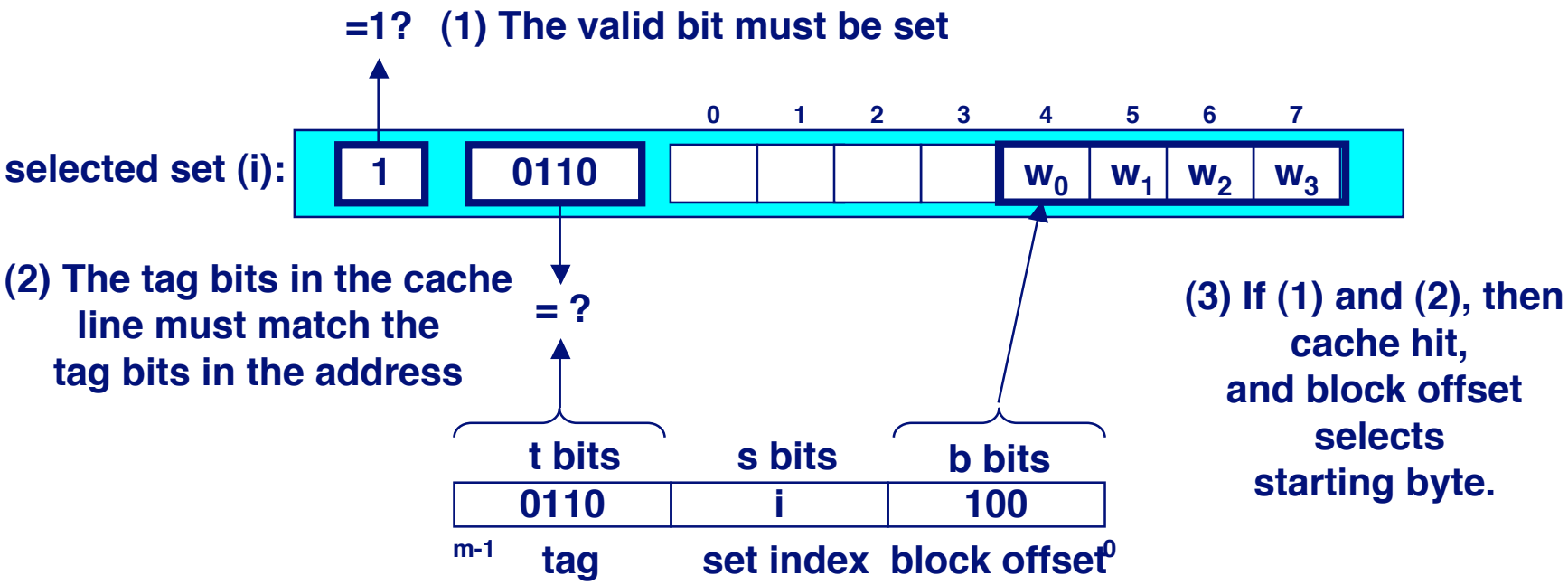
- Use the set index bits to determine the set of interest.



Accessing Direct-Mapped Caches

Line matching and word selection

- **Line matching**: Find a valid line in the selected set with a matching tag
- **Word selection**: Then extract the word



Direct-Mapped Cache Simulation

t=1	s=2	b=1
x	xx	x

M=16 byte addresses, B=2 bytes/block,
S=4 sets, E=1 entry/set

Address trace (reads):
0 [0000_2], 1 [0001_2], 13 [1101_2], 8 [1000_2], 0 [0000_2]

0 [0000_2] *(miss)*

(1)

v	tag	data
1	0	M[0-1]

13 [1101_2] *(miss)*

(3)

v	tag	data
1	0	M[0-1]
1	1	M[12-13]

8 [1000_2] *(miss)*

(4)

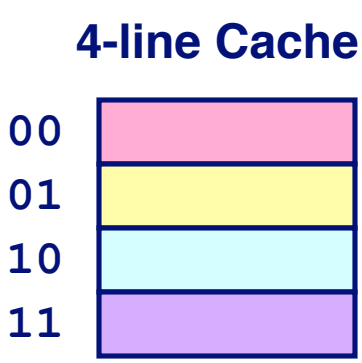
v	tag	data
1	1	M[8-9]
1	1	M[12-13]

0 [0000_2] *(miss)*

(5)

v	tag	data
1	0	M[0-1]
1	1	M[12-13]

Why Use Middle Bits as Index?

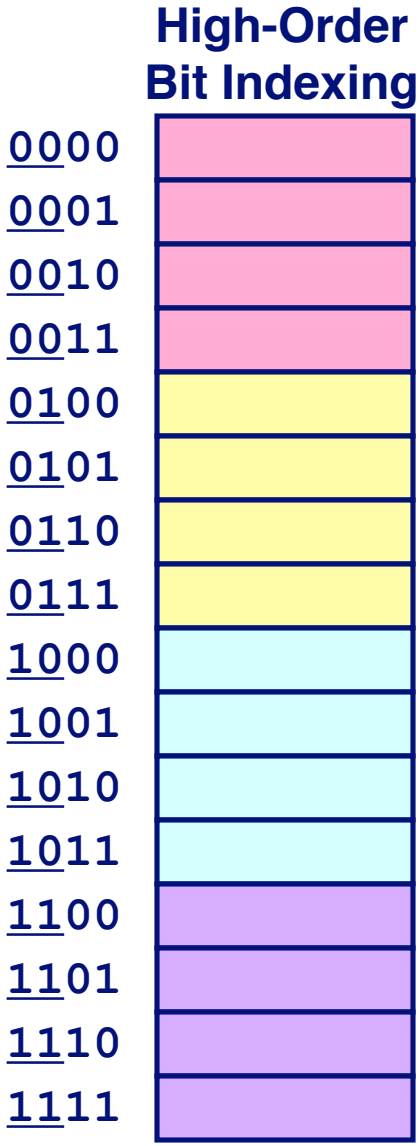


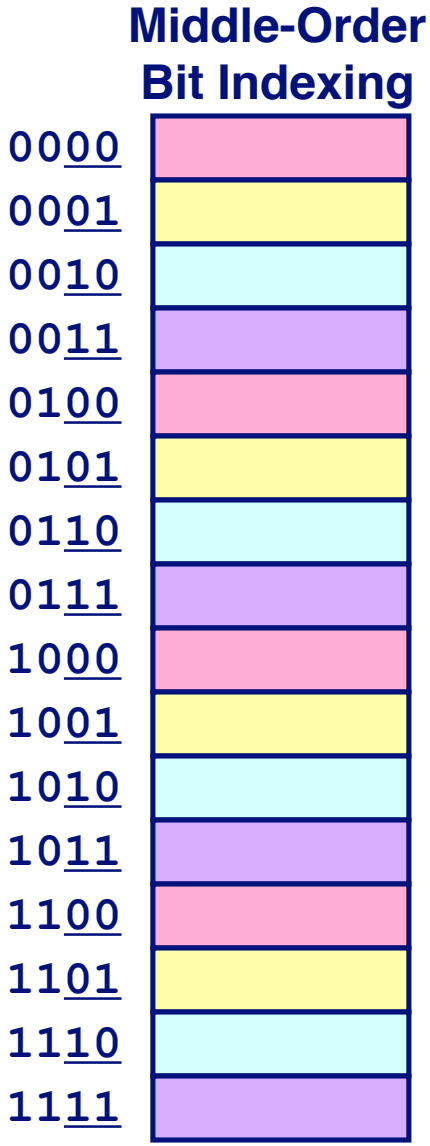
High-Order Bit Indexing

- Adjacent memory lines would map to same cache entry
- Poor use of spatial locality

Middle-Order Bit Indexing

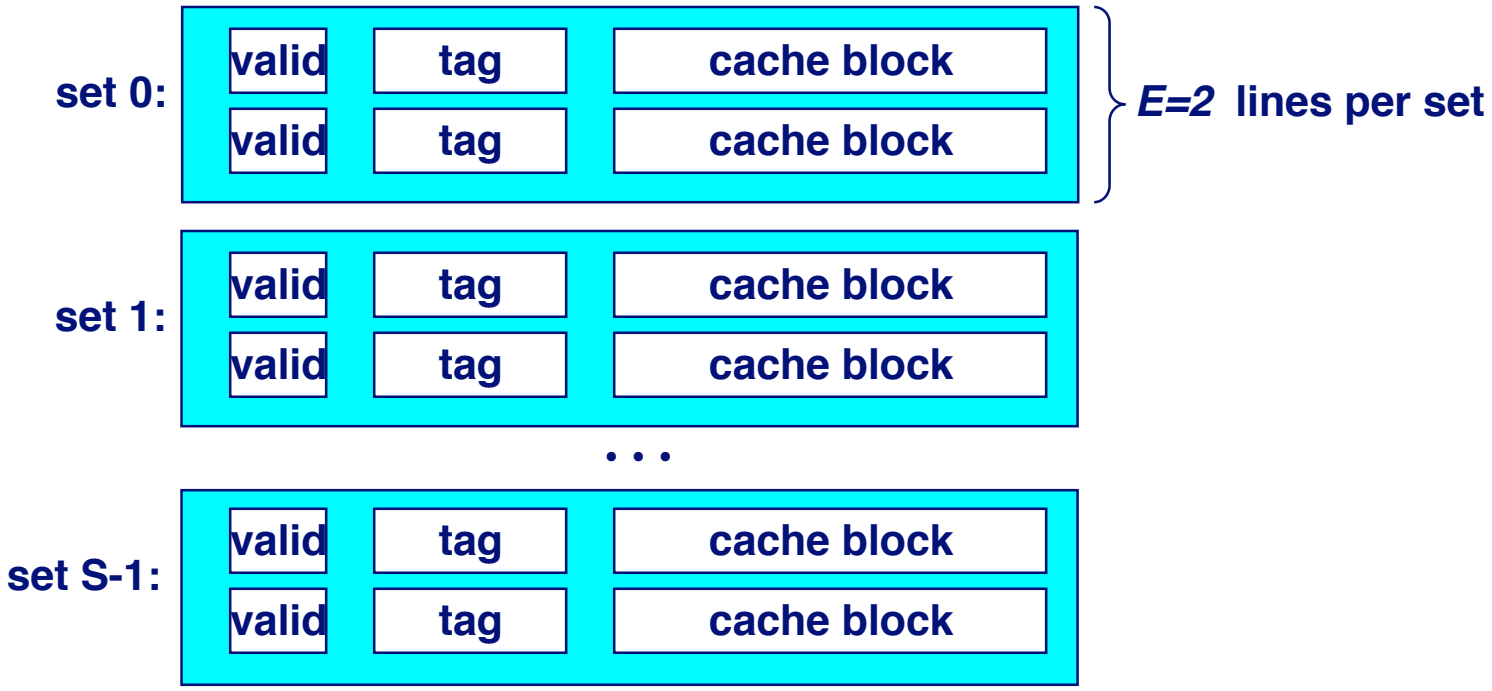
- Consecutive memory lines map to different cache lines
- Can hold C-byte region of address space in cache at one time





Set Associative Caches

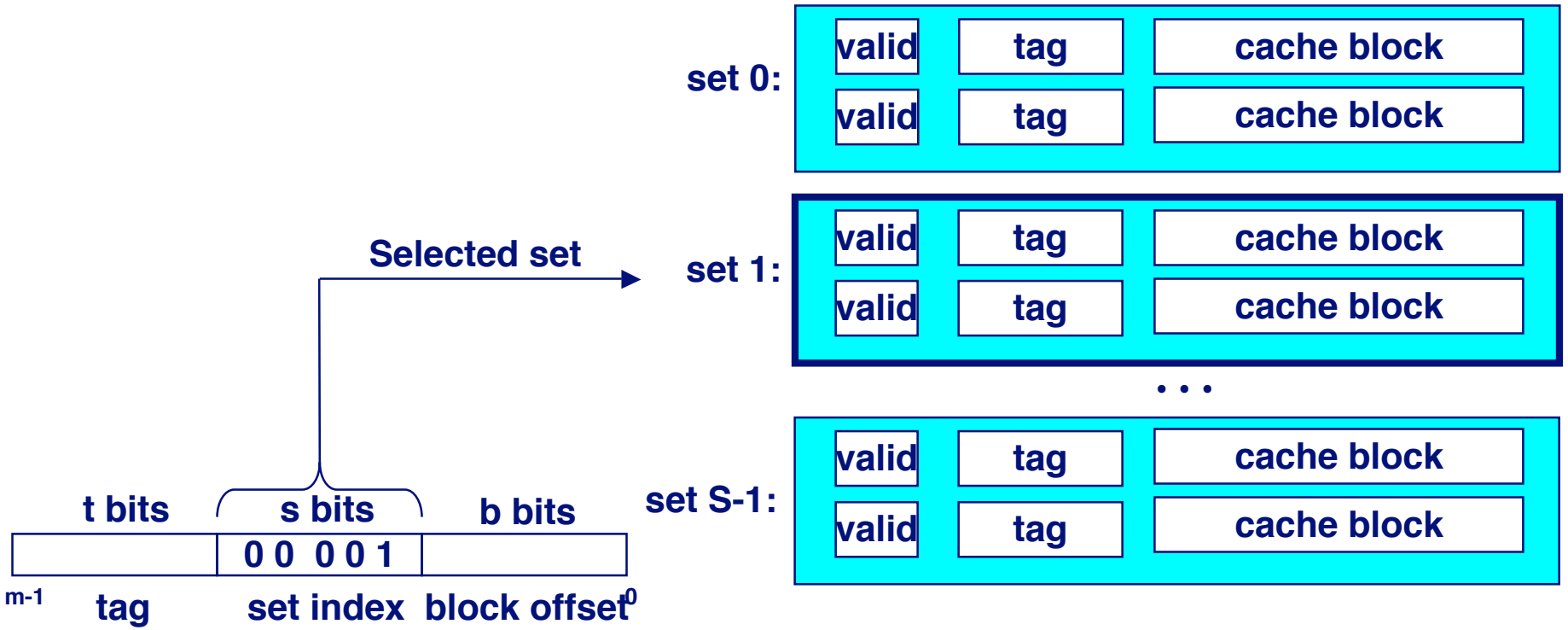
Characterized by more than one line per set



Accessing Set Associative Caches

Set selection

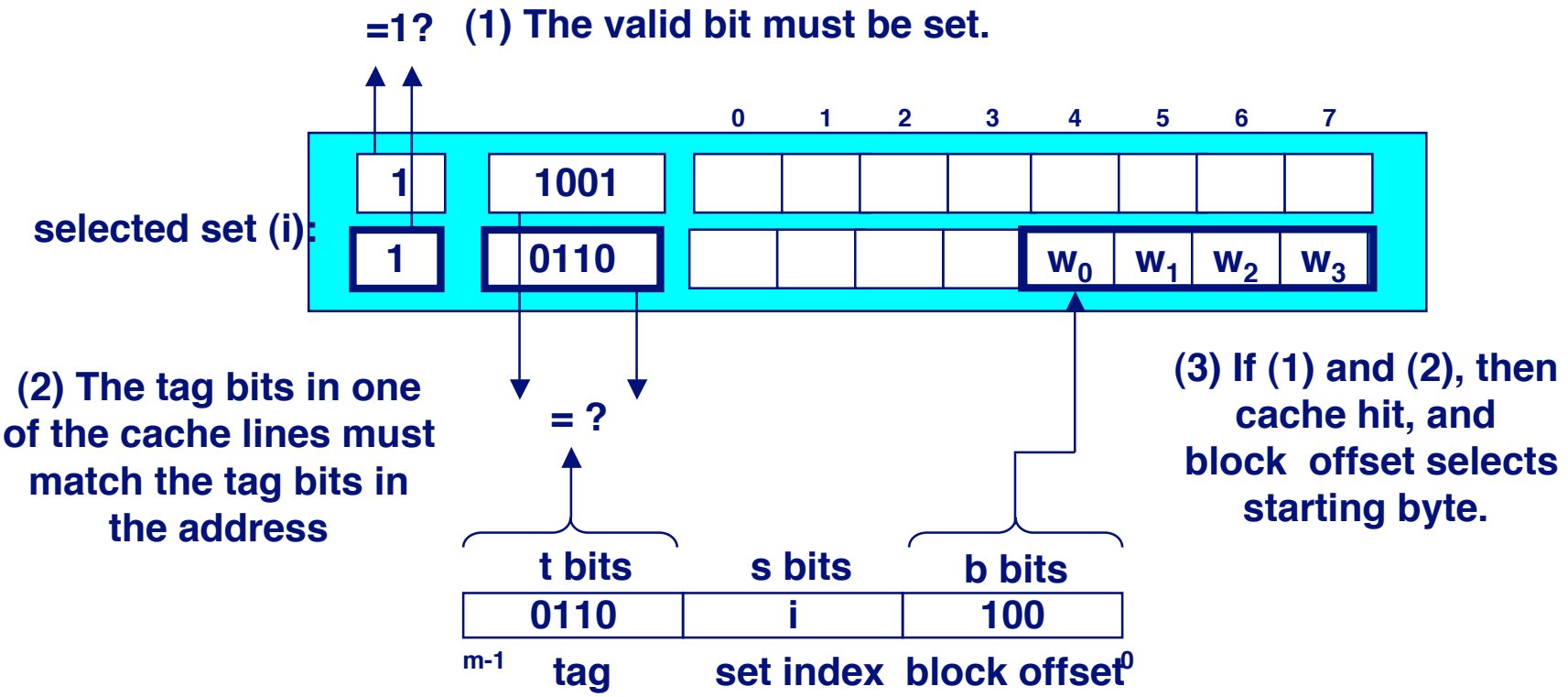
- identical to direct-mapped cache



Accessing Set Associative Caches

Line matching and word selection

- must compare the tag in each valid line in the selected set.



Cache Performance Metrics

Miss Rate

- Fraction of memory references not found in cache (misses/references)
- 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 in the cache to the processor (includes time to determine whether the line is in the cache)
- Typical numbers:
  - 1-3 clock cycle for L1
  - 5-12 clock cycles for L2

Miss Penalty

- Additional time required because of a miss
  - Typically 100-300 cycles for main memory

Memory System Performance

Average Memory Access Time (AMAT)

$$T_{access} = (1 - p_{miss})t_{hit} + p_{miss}t_{miss}$$
$$t_{miss} = t_{hit} + t_{penalty}$$

Assume 1-level cache, 90% hit rate, 1 cycle hit time, 200 cycle miss penalty

AMAT = 21 cycles!!! - even though 90% only take one cycle

Memory System Performance - II

How does AMAT affect overall performance?

Recall the CPI equation (pipeline efficiency)

$$CPI = 1.0 + lp + mp + rp$$

- load/use penalty (lp) assumed memory access of 1 cycle
- Further - we assumed that all load instructions were 1 cycle
- More realistic AMAT (20+ cycles), really hurts CPI and overall performance

Cause	Name	Instruction Frequency	Condition Frequency	Stalls	Product
Load	lp	0.30	0.7	21	4.41
Load/Use	lp	0.30	0.3	21+1	1.98
Mispredict	mp	0.20	0.4	2	0.16
Return	rp	0.02	1.0	3	0.06
Total penalty					6.61

Memory System Performance - III

$$T_{access} = (1 - p_{miss})t_{hit} + p_{miss}t_{miss}$$
$$t_{miss} = t_{hit} + t_{penalty}$$

How to reduce AMAT?

- **Reduce miss rate**
- **Reduce miss penalty**
- **Reduce hit time**

There have been numerous inventions targeting each of these

Writing Cache Friendly Code

Can write code to improve miss rate

Repeated references to variables are good (temporal locality)

Stride-1 reference patterns are good (spatial locality)

Examples:

- cold cache, 4-byte words, 4-word cache blocks

```
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;
}
```

Miss rate = 1/4 = 25%

```
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;
}
```

Miss rate = 100%

Questions to think about

- What happens when there is a miss and the cache has no free lines?
  - What do we evict?
- What happen on a store miss?
- What if we have a multicore chip where the processing cores share the L2 cache but have private L1 caches?
  - What are some bad things that could happen?

Concluding Observations

Programmer can optimize for cache performance

- How data structures are organized
- How data are accessed
  - Nested loop structure
  - Blocking is a general technique

All systems favor “cache friendly code”

- Getting absolute optimum performance is very platform specific
  - Cache sizes, line sizes, associativities, etc.
- Can get most of the advantage with generic code
  - Keep working set reasonably small (temporal locality)
  - Use small strides (spatial locality)