CS 201

Cache Micro-architecture

Gerson Robboy
Portland State University

Topics

- Generic cache memory organization
- Direct mapped caches
- Set associative caches
- What about writing to memory?
- Multiple processors sharing memory

locality - design goals

spatial locality - if you fetch data/instruction X, Xi is likely to be fetched next

true for code in functions

true for data in arrays

we want to load more than X - we want other items nearby (register will only take X) - call this block

the cache line

temporal locality - if you fetch instruction X now, you may do it again soon

true if code is in a loop

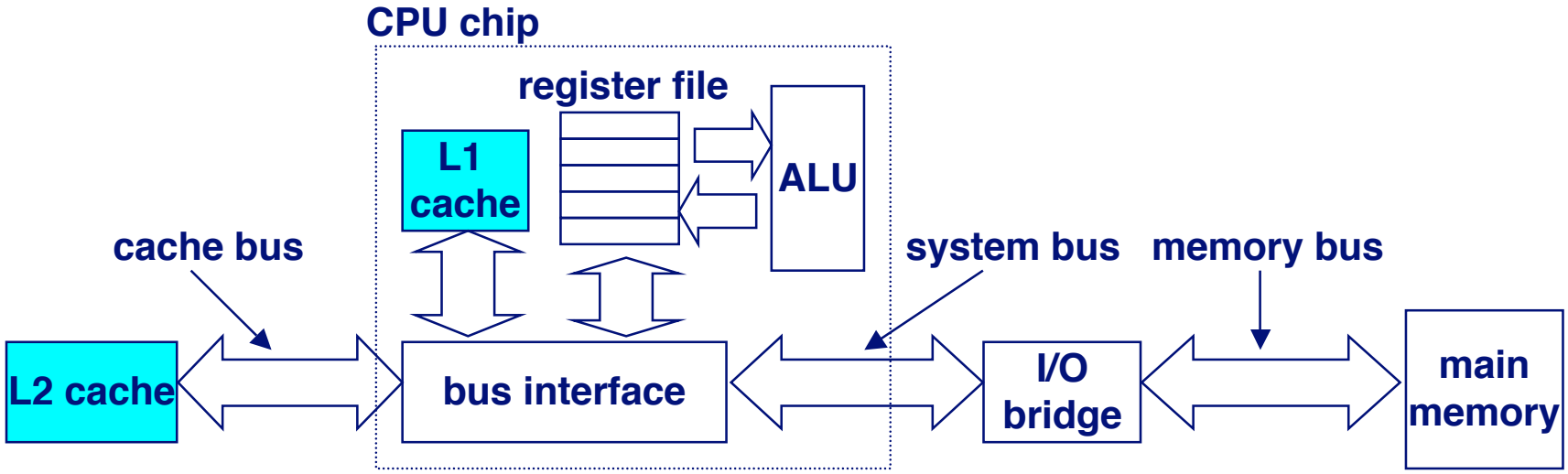
Cache Memories

Cache memories are small, fast SRAM-based memories managed automatically in hardware.

- **Hold frequently accessed blocks of main memory**

CPU looks first for data in L1, then in L2, then in main memory.

Typical bus structure:



On a cache miss…

Data is gotten from memory, stored in both L1 and L2.

The next access will be an L1 hit.

If evicted from L1, an L2 hit is still likely.

This is true for both reads and writes.

On a store, in case of cache miss…

- Pre-read the cache line from memory into both L1 and L2 caches.
- Store the data value into L1.
- The data is written through to L2.

Written back to memory later.

Subsequent stores may write more data into the same line in cache.

The problem with cache design

Given an instruction: movl <address>, %<reg>

Search the L1 cache, find the address, and move the data from L1 to register

- All in one clock cycle

In case of an L1 miss and L2 hit:

- Search L1 cache, find it's not there.
- Search the L2 cache, find the address, and move the data from L2 to register
- Also replicate the cache line in L1
- In at most 10 clock cycles

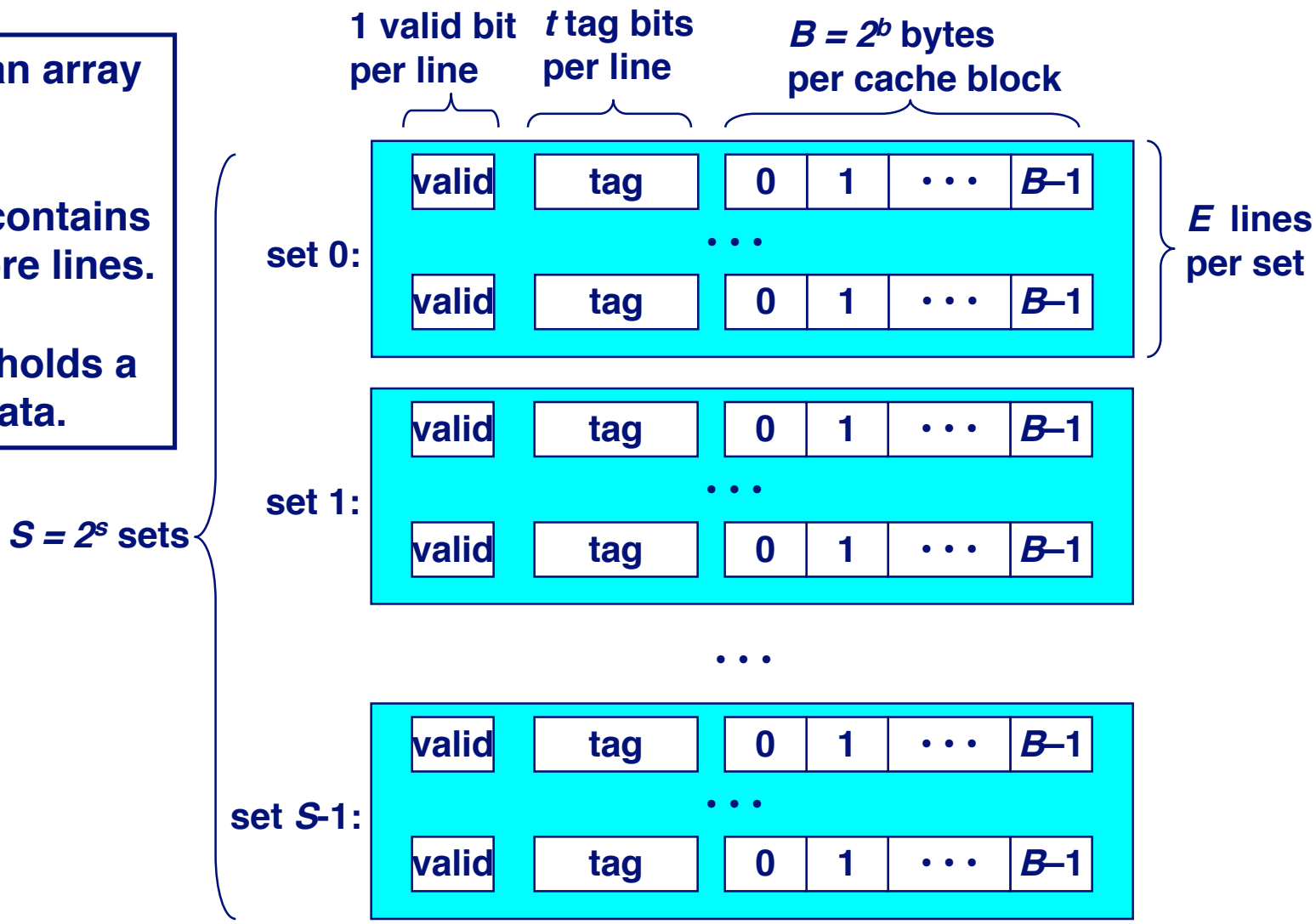
You need more than just fast SRAM to do that.

General Organization of a Cache

Cache is an array of sets.

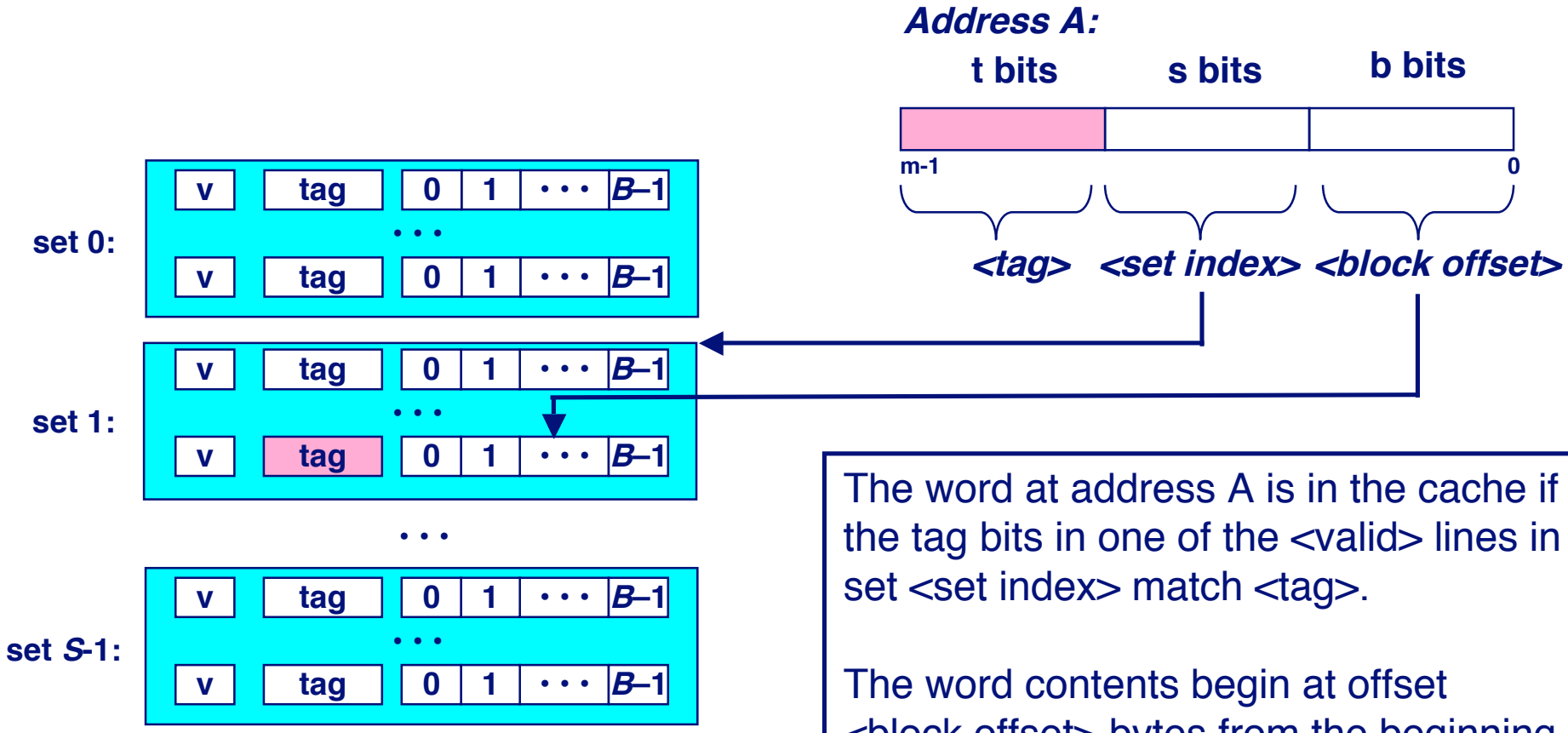
Each set contains one or more lines.

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.

note: tag in set may be different

therefore

1. we have tag, set, offset (break memory 32 bits up into 3 parts), set should give us the set,
2. the offset gives us a specific word (register) in a block (set of words)
3. tag must match (all parts must match actually). In some cases we may have multiple “lines” in a set.

Exercise

For each cache in the table, m=number of address bits, C=cache size in bytes, B=block size, and E=lines/set.

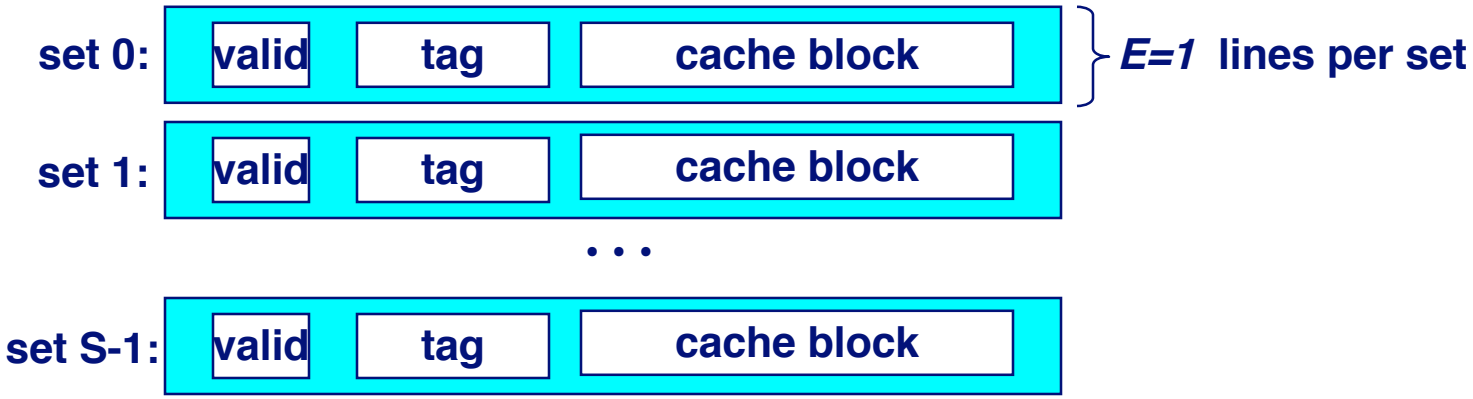
Determine the number of cache sets (S), tag bits (t), set index bits (s), and block offset bits (b).

m	C	B	E	S	t	s	b
32	1024	4	1				
32	1024	8	4				
32	1024	32	32				

Direct-Mapped Cache

Simplest kind of cache

Characterized by exactly one line per set.



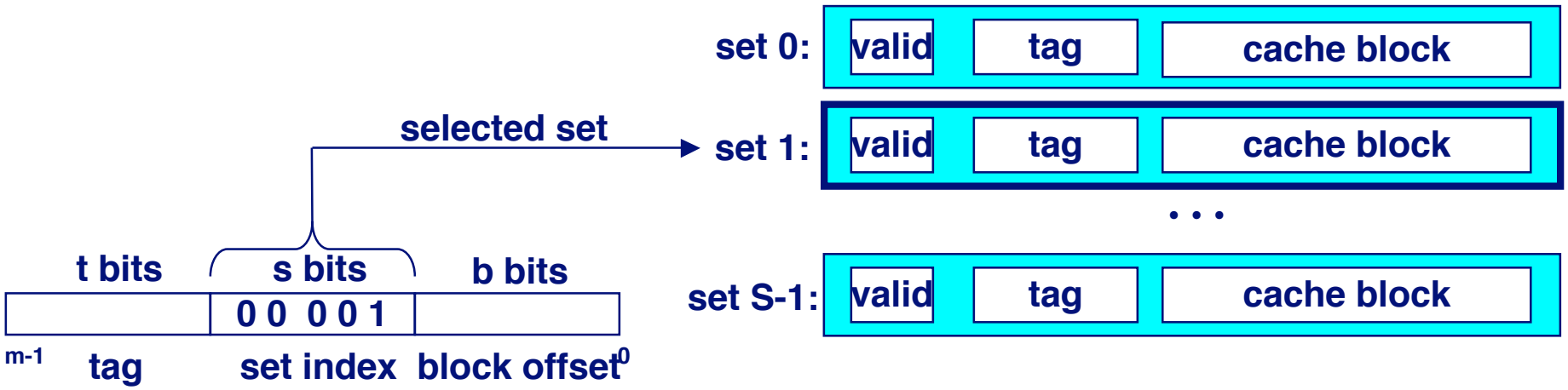
parking lot analogy (thanks to umd)

- assume 1000 parking lot spaces
- each space numbered from 000..999
- your parking spot based on 1st 3 digits of SSN
- there can only be one space
- simple to find - however there may be many collisions
- not very efficient
- drawbacks:
- poor hash function
- don’t use slots very well (don’t use a free slot)

Accessing Direct-Mapped Caches

Set selection

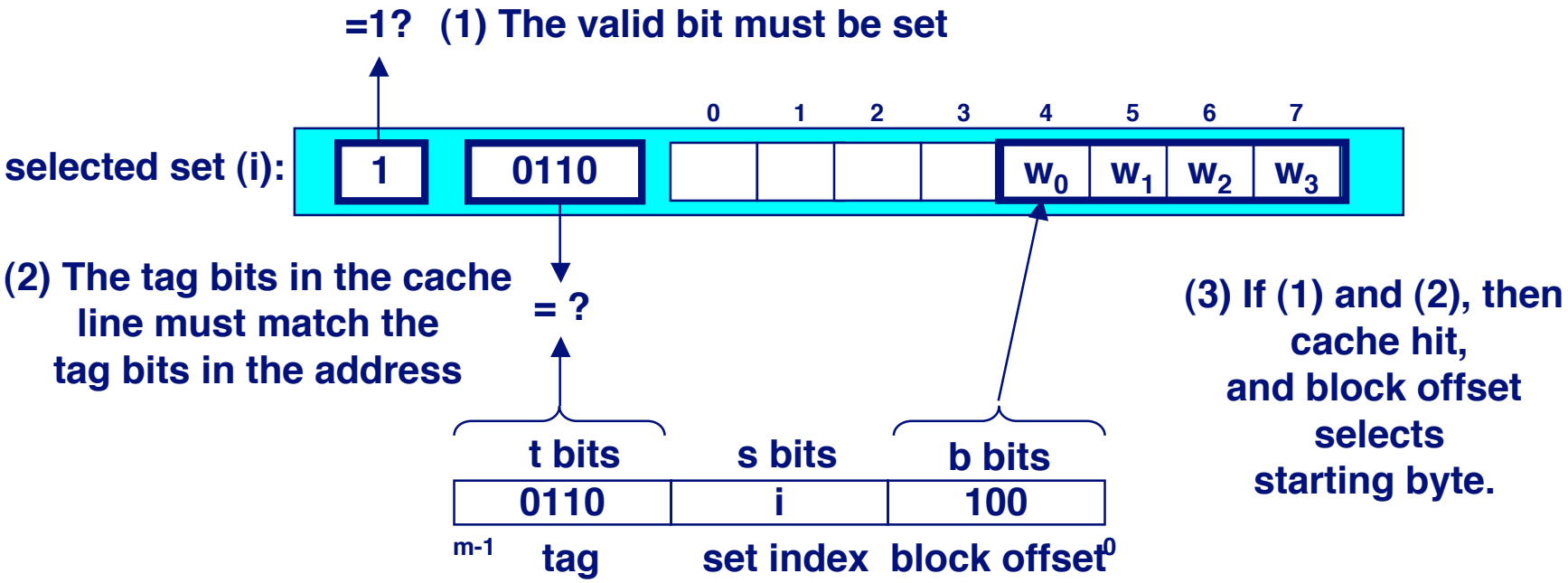
- Use the set index bits to determine the set of interest.



Accessing Direct-Mapped Caches

Line matching and word selection

- **Line matching**: Find a valid line in the selected set with a matching tag (actually look up set, tag makes sure it is right)
- **Word selection**: Then extract the word



Direct-Mapped Cache Simulation

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

m=4 bits/address, M=16 byte address space,
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)

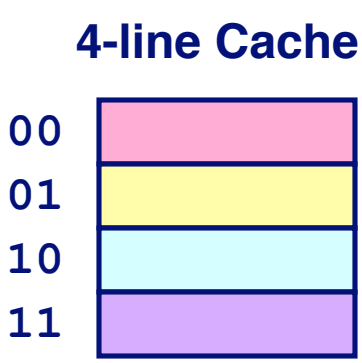
v	tag	data
1	1	M[8-9]
1	1	M[12-13]

0 [0000_2] *(miss)*

(5)

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

Why Use Middle Bits as Index?

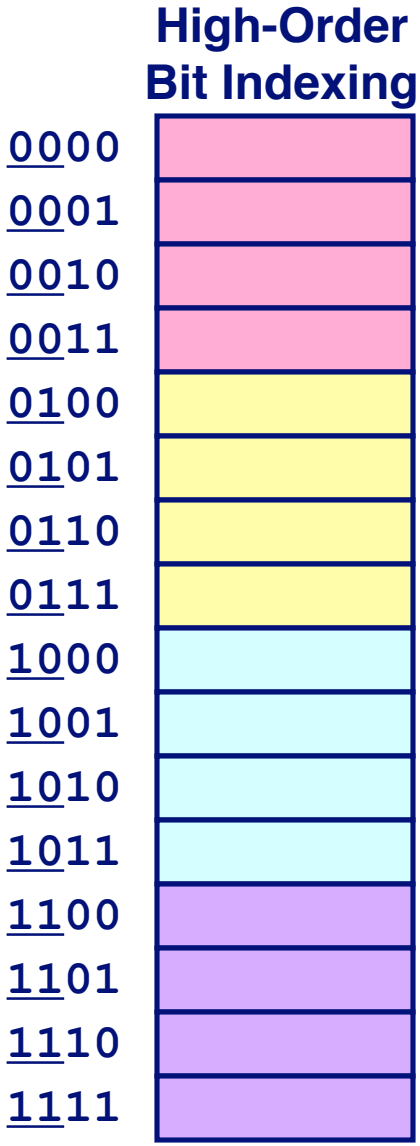


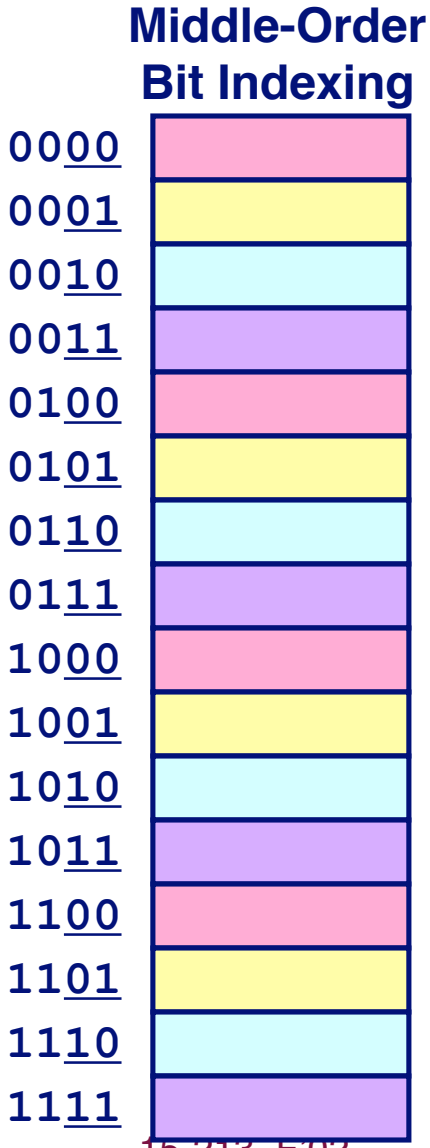
High-Order Bit Indexing

- 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





Exercise

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

m=4 bits/address, M=16 byte address space,
B=2 bytes/block, S=4 sets, E=1 entry/set

Address trace (reads):
4 [0100_2], 1 [0001_2], 13 [1101_2], 8 [1000_2], 0 [0000_2],
5 [0101]

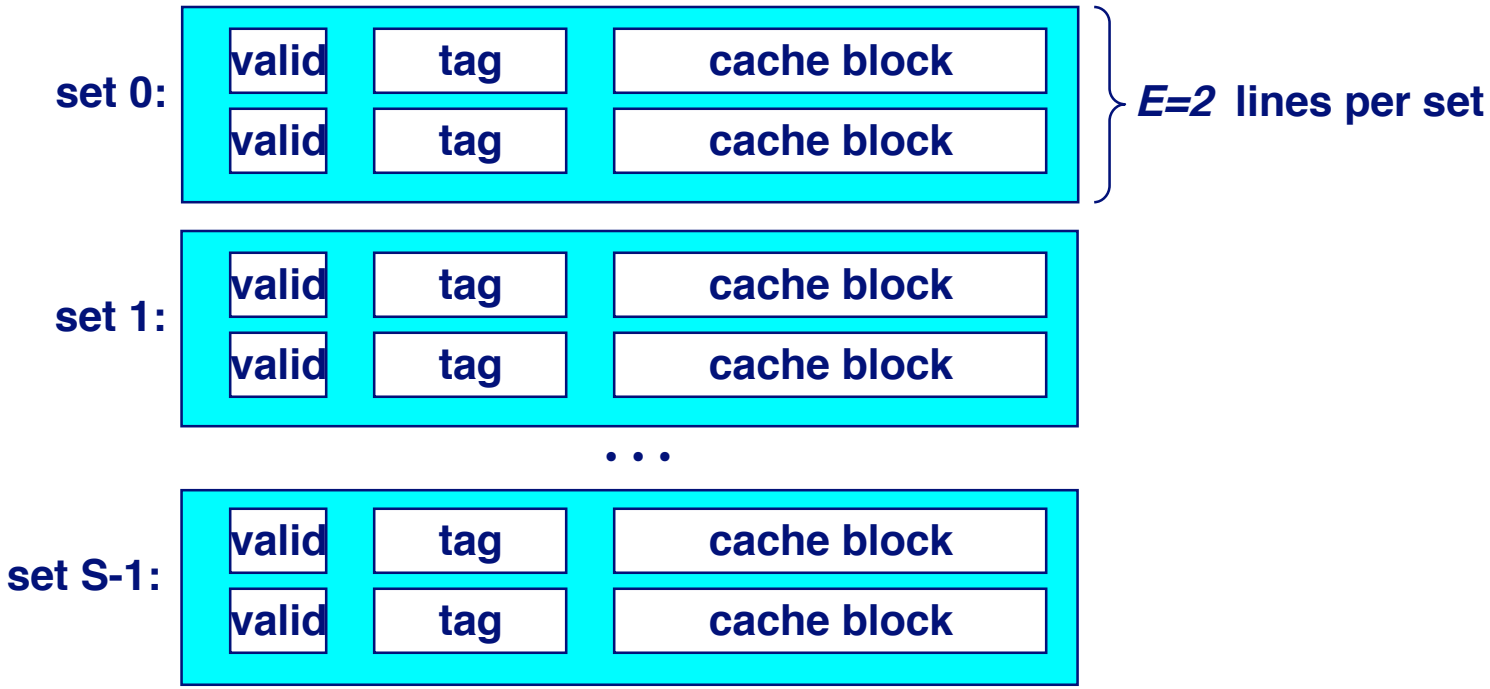
Draw a chart showing the cache misses and hits.

What is the problem with one entry per set?

Set Associative Caches

Characterized by more than one line per set

- This example is a two-way set associative cache



parking lot analogy for N-way set

1000 parking spaces

2-digit assigned number. 00-99

10 slots numbered 00, 01, 02, … 99

thus 10 possible places to park

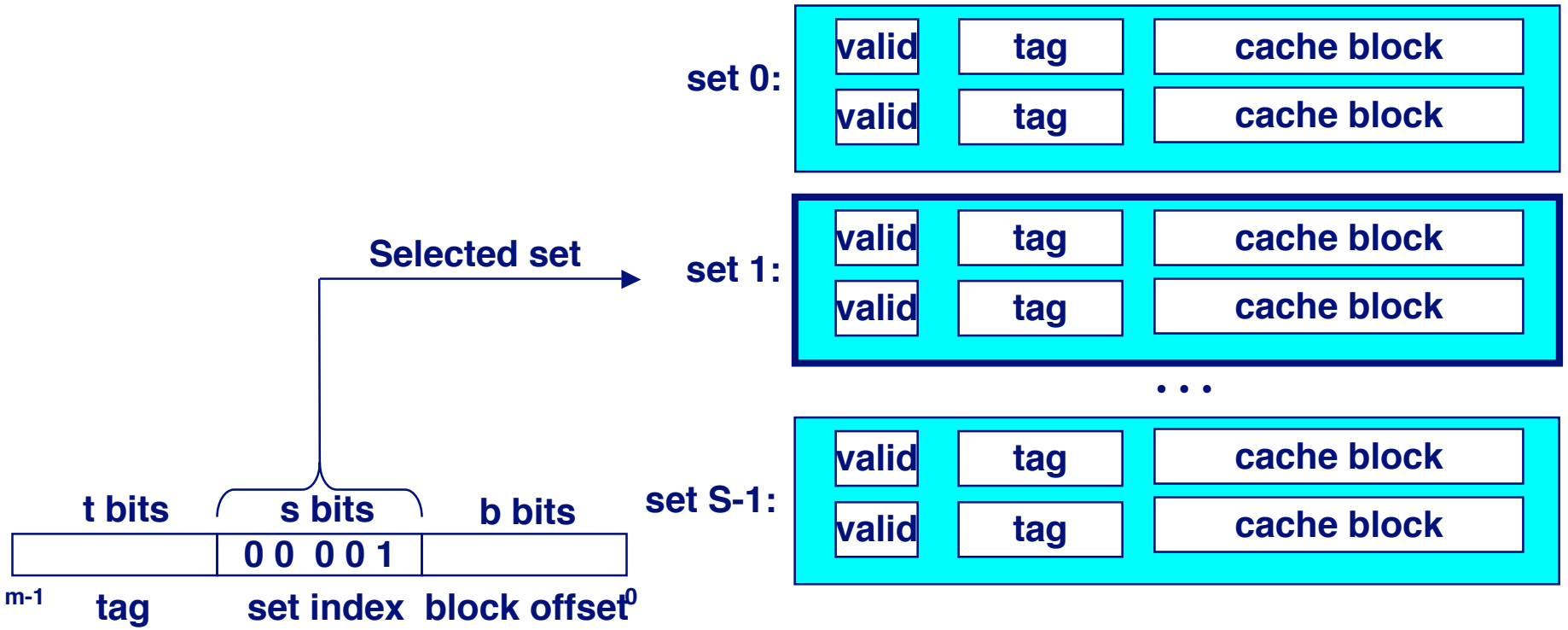
in reality you might be assigned lot A, with 100 places to park, but you can't park in lot B

point: odds are higher that you can find an acceptable slot nearby

Accessing Set Associative Caches

Set selection

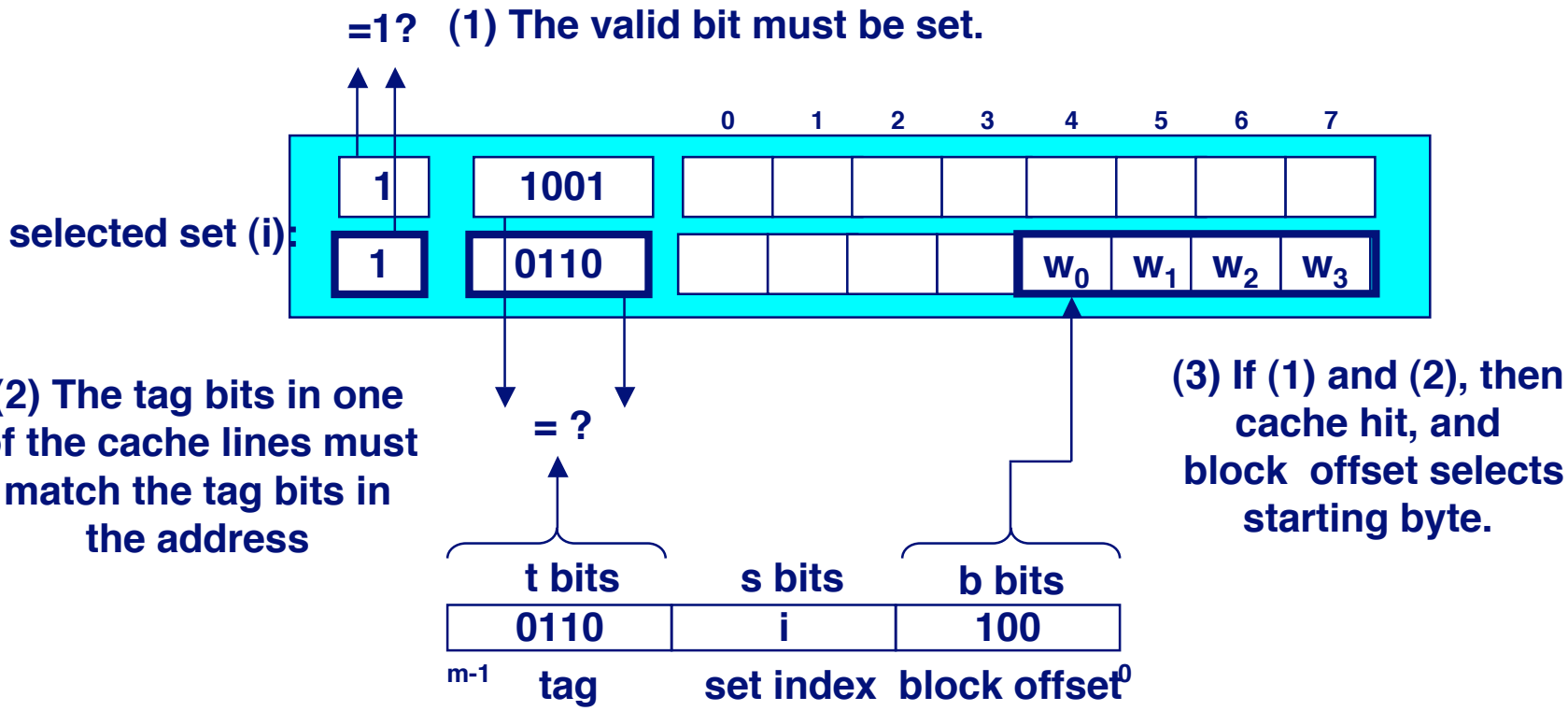
- identical to direct-mapped cache



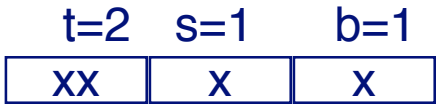
Accessing Set Associative Caches

Line matching and word selection

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



Exercise



m=4 bits/address, M=16 byte address space,
B=2 bytes/block, S=2 sets, E=2 entries/set

Address trace (reads):
4 [0100_2], 2 [0010_2], 13 [1101_2], 3 [0011_2], 5 [0101],
0 [0000]

Draw a chart showing the cache misses and hits.

Associative Caches

A cache with N lines per set is called an *N-way set associative cache*

For example, a typical microprocessor might have a 4 way set associative cache

Exercise

A processor has 32 bit addresses. The L2 cache is 256 K-bytes in size, 8-way set associative, with a block size of 64 bytes.

How many block offset bits are there, how many set index bits, and how many tag bits?

Given an address 0x30004a5c

What is the block offset, what is the set index, and what is the tag?

Fully associative caches

A cache with exactly one set is called a *fully associative cache*

- All lines in the cache are in that one set
- Each line is uniquely identified by the tag bits alone

The problem with fully associative caches

- There are many cache lines in one set
- The CPU must find a matching tag very fast (less than one clock cycle)
- You need logic to compare all the tags in parallel
- This can get expensive

parking lot analogy

1. more parking permits than slots
2. student can park in any space
3. in reality hardware has to do a parallel search based on the valid bit AND the tag

Review

A direct mapped cache has one line per set

- There are as many sets as there are cache lines in the cache
- Likelihood of contention
  - "conflict misses"
  - Cache entries get evicted when the cache isn't full

A fully associative cache has exactly one set

- All lines in the cache belong to that one set
- Problem of quick search for a matching tag

A set associative cache has several lines per set

- An N way associative cache has N lines per set

What about writing to memory?

Written data is also cached.

On a cache miss, pre-read the cache line into the cache.

Then write the data into the cache line.

Subsequent reads or writes will have a cache hit.

Writing to memory

How does the data get from the cache to memory?

***Write-through cache*: Data is written through to memory at the time of the "store" instruction**

***Write-back cache*: New data is stored in the cache and written to memory later.**

What is the problem with a write-through cache?

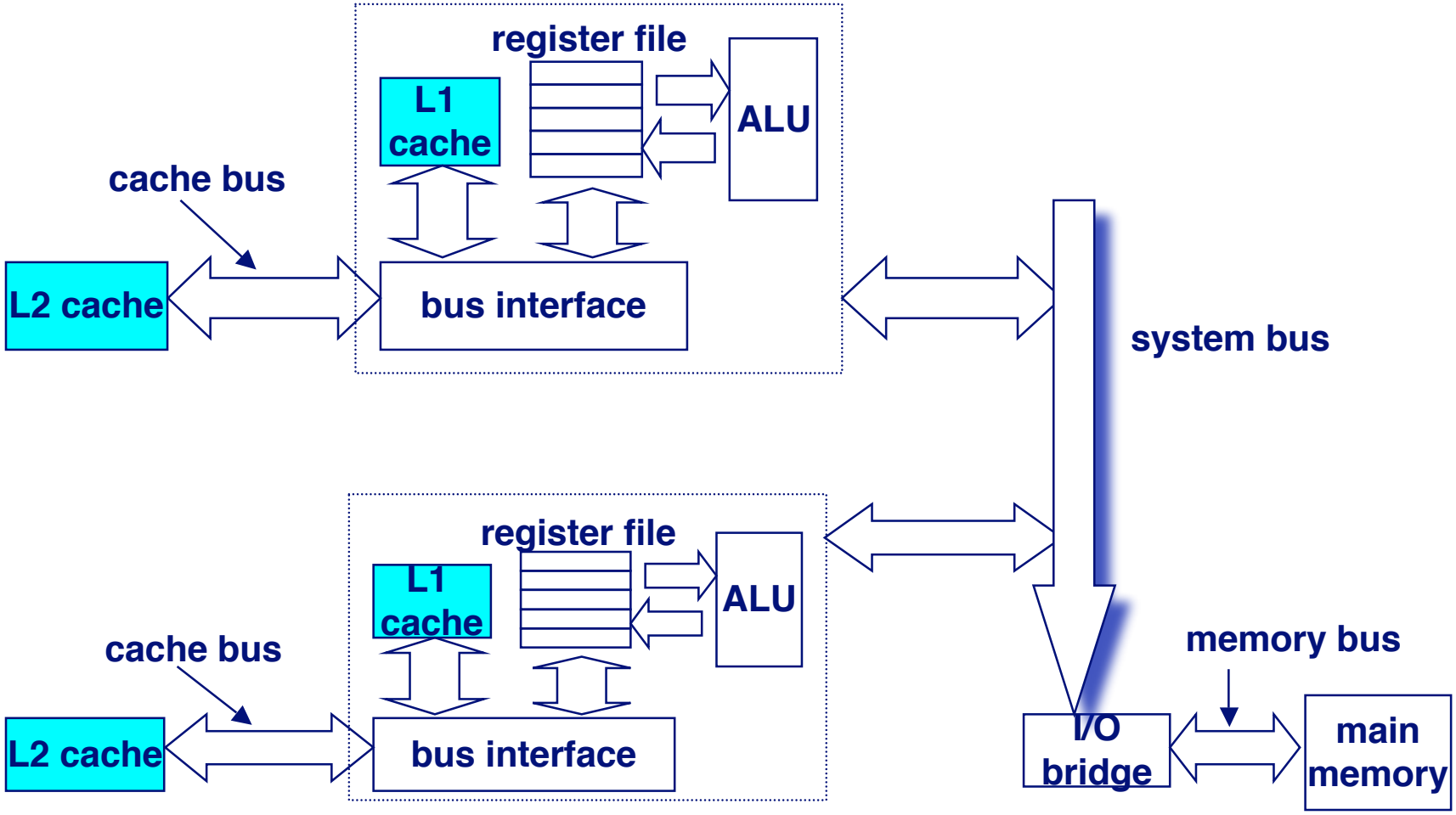
What is the problem with a write-back cache?

may have 2 write data twice (evict dirty and write new)

With write-back, how does the CPU decide when to write the data to memory?

What about multi-processors?

What's the problem with write-back caches?



design rule of thumb for caches

if you double the associativity, that is about the same as doubling the cache size itself

so 2-way > direct

4-way > 2-way

beyond 4-way, not so good