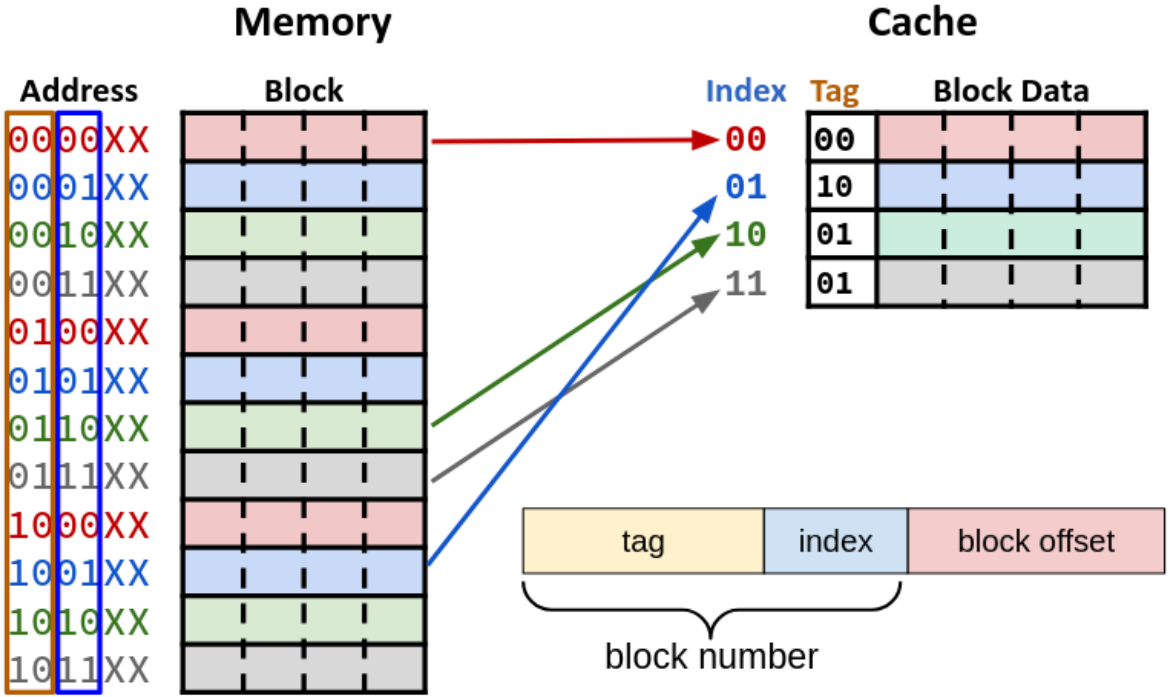
Caches III


Async (📍 Paris Arc 🇫🇷)
@0xAsync
No mom it's not a "messy pile of clothes on my chair" it's an L1 cache for fast random access to my frequently used clothes in O(1) time. It needs to be big to avoid expensive cache misses (looking in my closet). I NEED to be minimizing latency, this is important to me. Please.

Caches

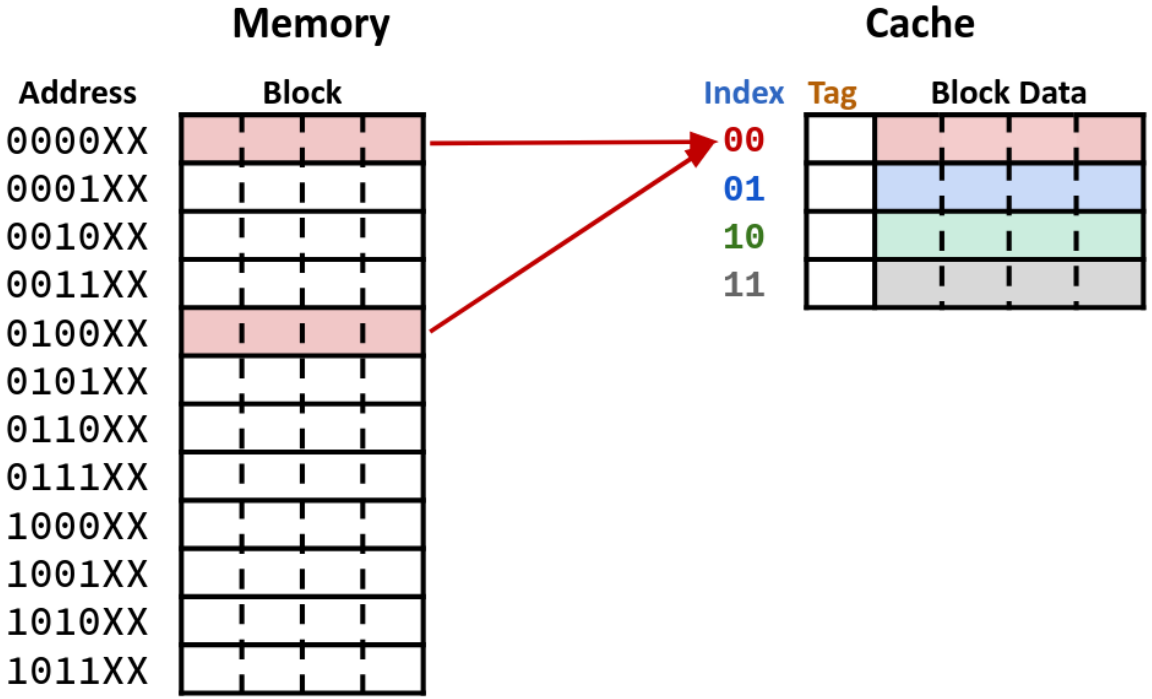
- Cache basics
- Principle of locality
- Memory hierarchies
- **Cache organization**
  - Direct-mapped (sets; index + tag)
  - **Associativity (ways)**
  - **Replacement policy**
  - Handling writes
- Program optimizations that consider caches

Recap: Direct-Mapped Cache



- Each block in memory can only go into one set
  - Fast and simple
- Hash function
  - *(Block #)* mod (*# of sets)*

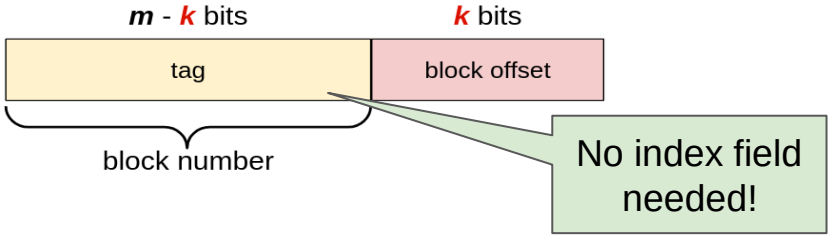
Direct-Mapped: a Problem!

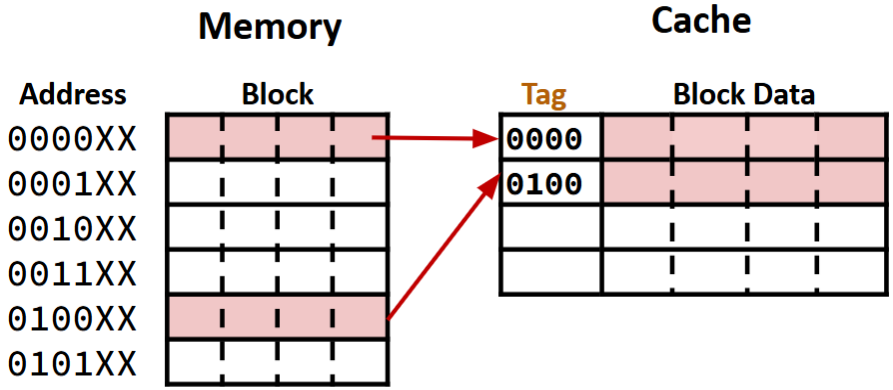


- What if we have the following access pattern:
  - 0, 16, 0, 16, …
  - Each access will miss
    - **Thrashing!**
  - Rest of the cache unused

One Solution: Fully Associative

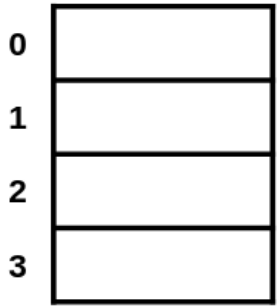
- Can store any data block anywhere in the cache
  - When loading in a new block, store it in the first unused space
  - Only evict old blocks when the entire cache is full!
- Problems with this?
  - Requires complicated hardware to check each set
    - More power consumed, slower





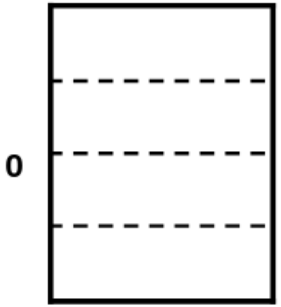
Full Solution: Associativity

- Rather than just one block per set, we can have multiple
  - **Cache line** = block + relevant metadata (i.e. tag)
  - **Associativity** (***E***) = number of lines in a set
    - Cache is "***E***-way set associative"
- If any block can go in any set, the cache is **Fully Associative**



1-way (1 block per set, 4 sets)
Direct-Mapped

2-way (2 blocks per set, 2 sets)



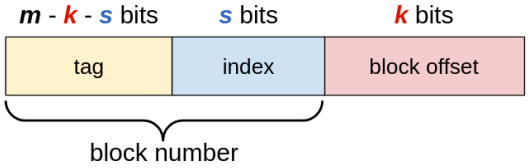
4-way (4 blocks per set, 1 set)
Fully-Associative

Cache Organization: Associativity

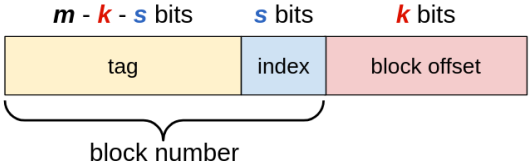
K = block size (in bytes), ***C*** = cache size (in bytes), ***E*** = associativity

- ***S*** = number of sets = ***C*÷*K*÷*E***
- Use lowest ***s*** = $\log_2$(***S***) bits of block number to find the set
  - Direct-Mapped: ***E*** = 1, so ***s*** = $\log_2$(***C*÷*K***), same as we saw previously
  - Fully Associative: ***E*** = ***C*÷*K***, so $\log_2(1) = 0$

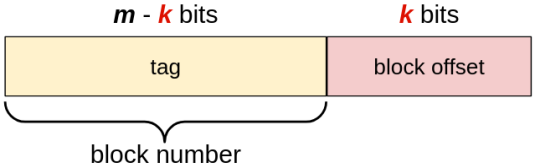
Direct-Mapped



n-way Set Associative



Fully Associative

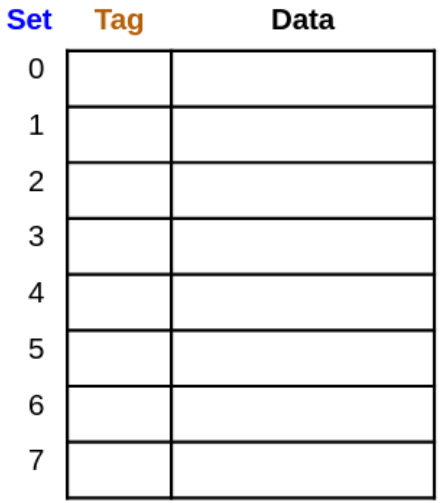


Example Placement #1

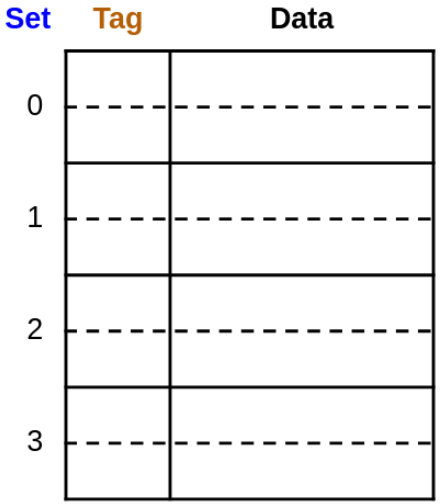
Block size K: 16 B
Capacity C/K: 8 blocks
Address m: 16 bits

- Where would data from address `0x1833` be placed?
  - `0b0001 1000 0011 0011`

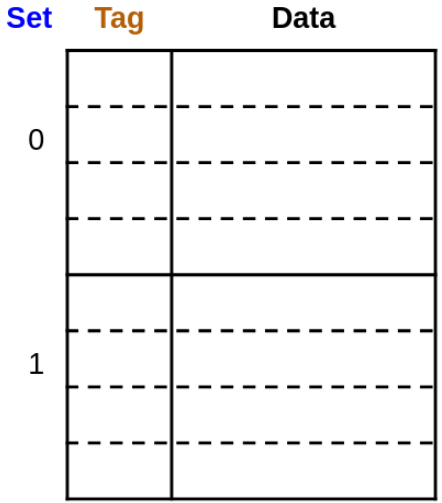
S = _____



S = _____



S = _____



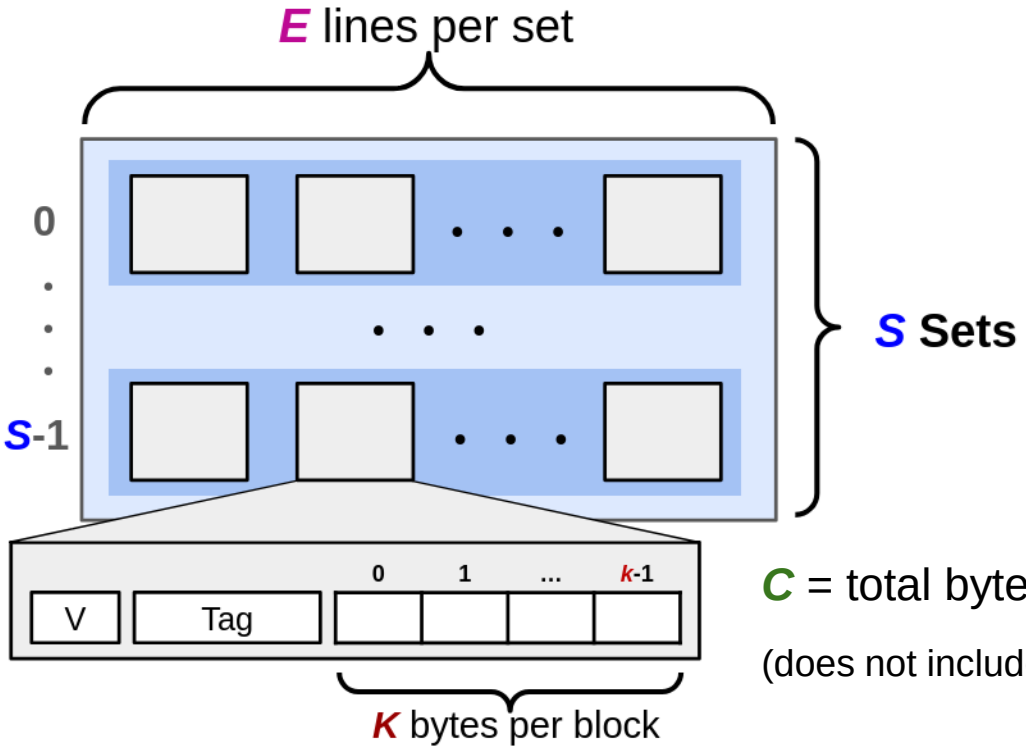
Block Placement and Replacement

- If there is an empty line in the set, store data there
  - How to tell if the line is empty? **Valid bit** (0 = empty, 1 = used)
    - Stored in cache line
- If there are no empty lines, which one do we replace?
  - No choice for direct-mapped, easy!
  - For other caches, need a **replacement policy**
    - Ideal is **least recently used** (**LRU**)
    - Most real caches *approximate* LRU

Polling Questions

1. We have a cache of size 2 KiB with block size of 128 B. If our cache has 2 sets, what is its associativity?
   A) 2
   B) 4
   C) 8
   D) 16
2. If addresses are 16 bits wide, how wide is the Tag field?

General Cache Layout Review



C = total bytes = *K***E***S*

(does not include tag or valid bit)

Notation Review

Parameter	Variable	Formulas
Block size	K (B in book)	$C = K*E*S \leftrightarrow S=C/K/E$ $k = \log_2(K) \leftrightarrow K = 2^k$ $s = \log_2(S) \leftrightarrow S = 2^s$ $m = \log_2(M) \leftrightarrow M = 2^m$ $m = t + s + k$
Cache size	C	
Associativity	E	
Number of sets	S	
Address space	M	
Address width	m	
Offset field width	k (b in book)	
Index field width	s	
Tag field width	t	

Example Cache Problem

1KiB address space, 125 cycles to go to memory. Fill in the following table:

Cache size (*C*)	64B
Block size (*K*)	8B
Associativity (*E*)	2-way
Hit Time	3 cycles
Miss Rate	20%
Address width (*m*)	
Offset bits (*k*)	
Index bits (*s*)	
Tag bits (*t*)	
AMAT	

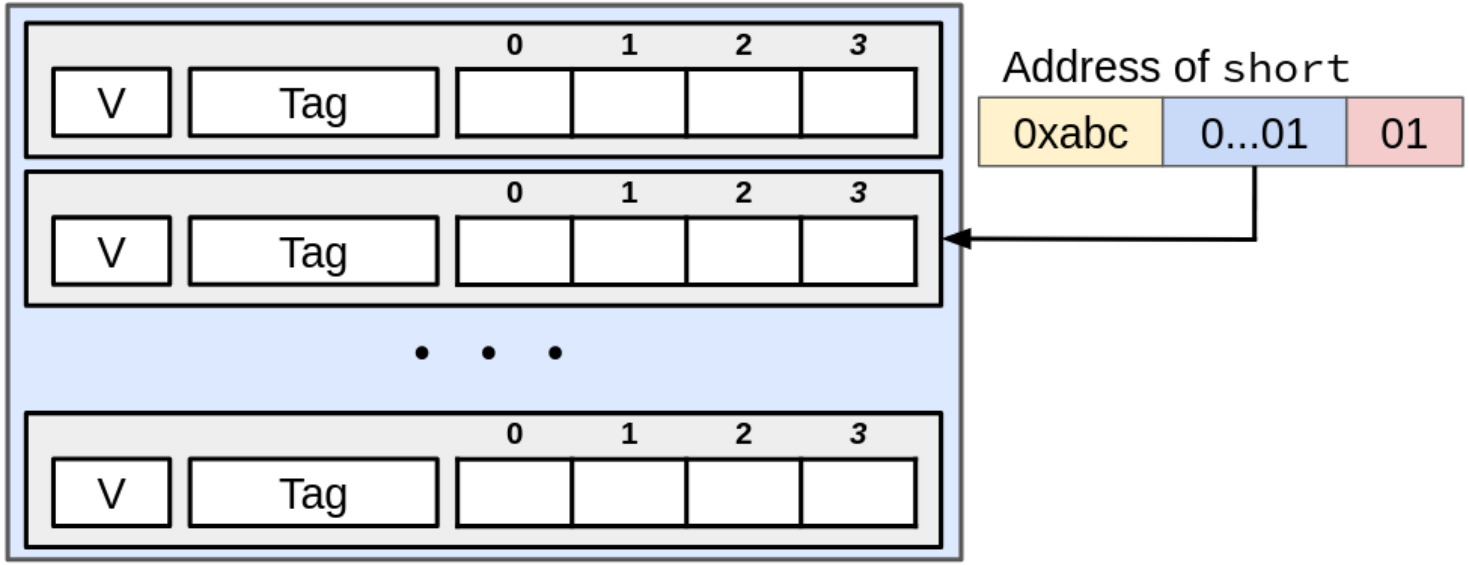
Read: General Steps

1. Break up address into **offset**, **index**, and **tag**
2. Locate the set using **index**
3. Check if *any line* in the set has our data
    a. **Valid bit** = 1 *and* **tag** matches
    b. If yes - hit!
        i. Otherwise, load in from memory
4. Read out data from block starting at **offset**

Read: Direct-Mapped Cache

One line per set (***E***=1), ***K***=4B

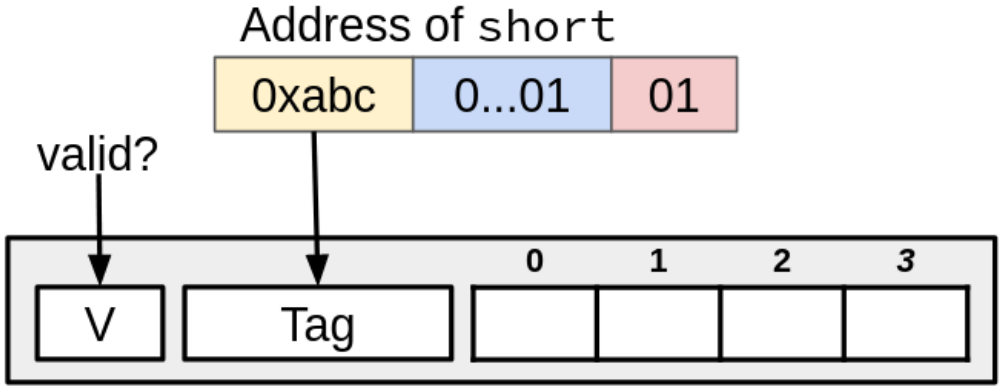
1. Locate set



Read: Direct-Mapped Cache (pt 2)

One line per set (***E***=1), ***K***=4B

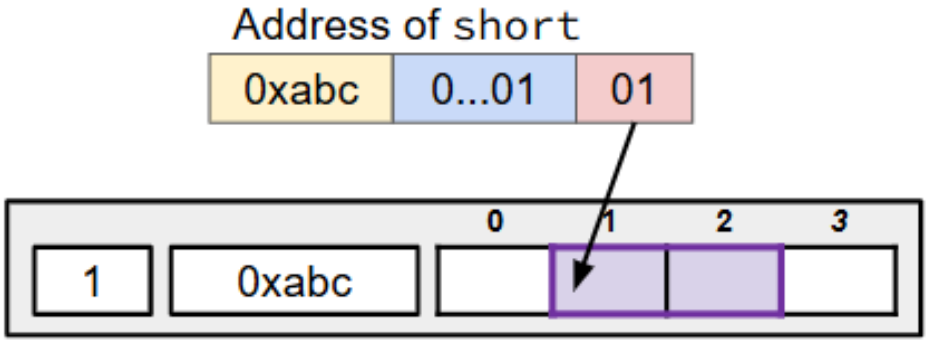
1. Locate set
2. Check if valid and compare tag
    a. No match? Old block gets evicted, replaced with new one



Read: Direct-Mapped Cache (pt 3)

One line per set (***E***=1), ***K***=4B

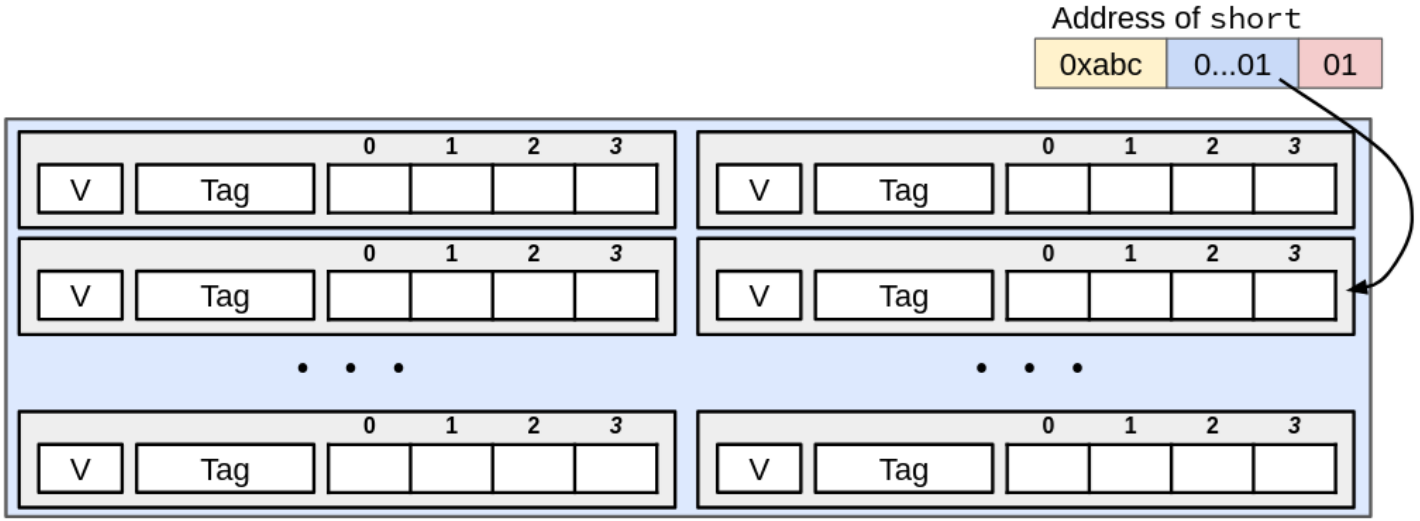
1. Locate set
2. Check if valid and compare tag
3. Get data starting at offset



Read: Set Associative Cache

Two lines per set (***E***=2), ***K***=4B

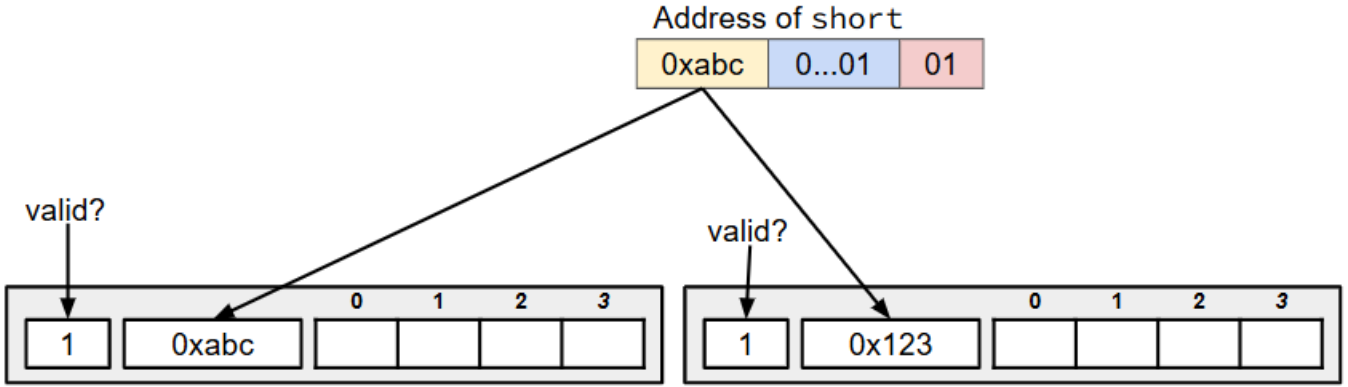
1. Locate set



Read: Set Associative Cache (pt 2)

Two lines per set (***E***=2), ***K***=4B

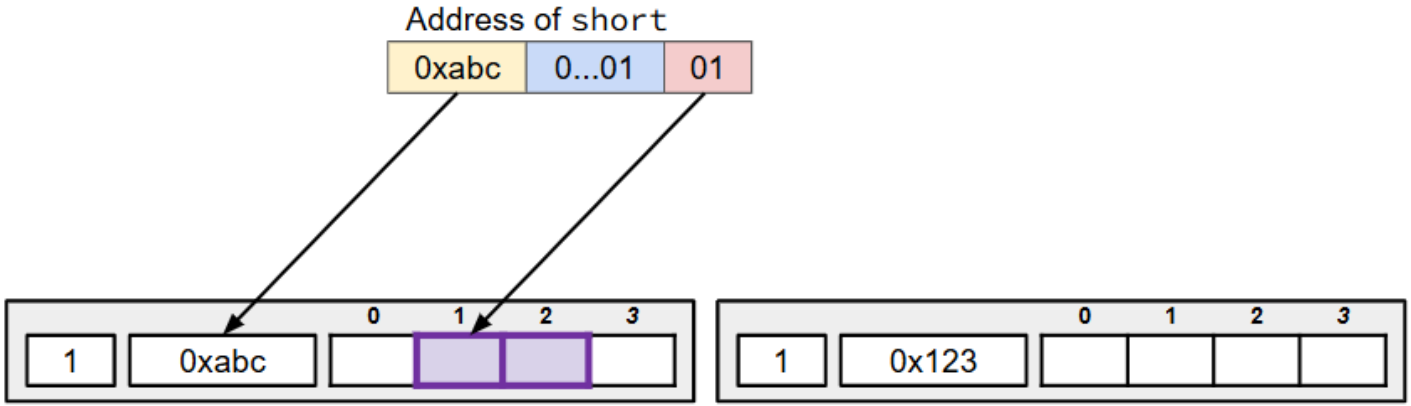
1. Locate set
2. Check if valid and compare tag for **every line** in the set
    a. No match? One line is selected to get evicted and replaced by new one



Read: Set Associative Cache (pt 3)

Two lines per set (***E***=2), ***K***=4B

1. Locate set
2. Check if valid and compare tag for **every line** in the set
3. Get data starting at offset



Types of Cache Misses: 3 C’s

- **Compulsory** (cold-start) miss
  - Occurs on the first access to a block
  - Smaller block size = more compulsory misses
- **Conflict** miss
  - Occurs when cache is large enough to hold multiple data blocks, but they cannot be in the cache at the same time because they conflict
  - Lower associativity = more conflict misses
    - *Does not occur in fully associative caches*
- **Capacity** miss
  - Occurs when the set of active blocks (the **working set**) is too big to fit in the cache
  - Smaller cache size = more capacity misses

Code Analysis

- Assuming cache starts **cold** (i.e. all blocks invalid), and `sum`, `i`, and `j` are all stored in registers, calculate the **miss rate**.
  - ***m*** = 10 bits, ***C*** = 64B, ***K*** = 8B, ***E*** = 2

```
#define SIZE 8
short ar[SIZE][SIZE], sum = 0; // &ar=0x200
for (int i = 0; i < SIZE; i++) {
    for (int j = 0; j < SIZE; j++)
        sum += ar[j][i];
}
```

Summary

- **Associativity**: number of cache lines in a set
  - Direct-Mapped caches have associativity 1
  - **Fully Associative** caches have associativity equal to # of blocks (all in one set)
  - Most caches are somewhere in between
- 3 types of misses
  - **Compulsory**: first time accessing block
  - **Conflict**: block was evicted by another when the cache was not full
  - **Capacity**: block was evicted because the cache was full