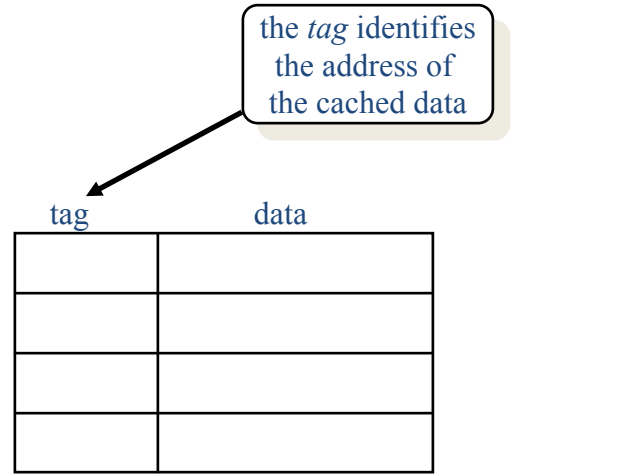
Hardware implications on cache design

- Caches are basically *the* thing that make real workloads fast
- The size of a cache is inversely proportional to its speed
  - Smaller caches are faster
- And every bit counts

- This is why caches use as few **bits** as possible to do their work
  - This makes caches tricky to walk through as a human

A simple cache

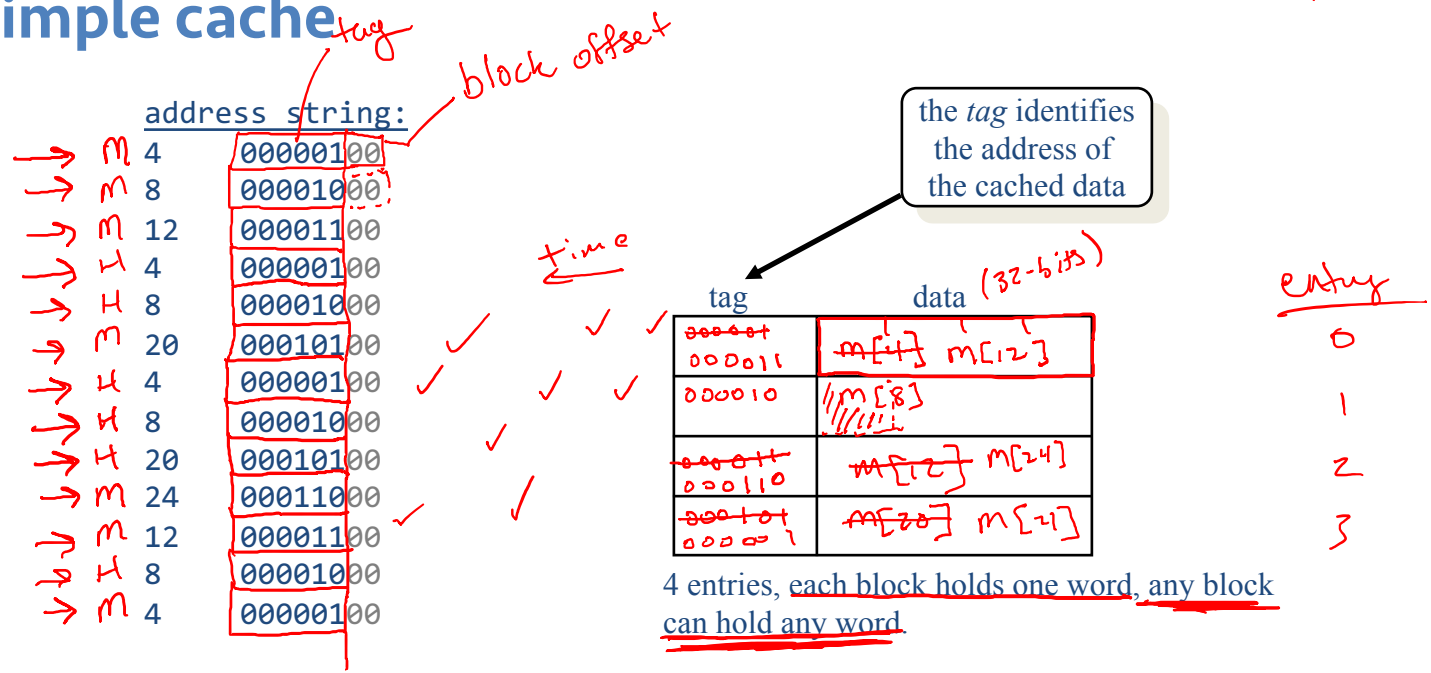
address string:	
4	00000100
8	00001000
12	00001100
4	00000100
8	00001000
20	00010100
4	00000100
8	00001000
20	00010100
24	00011000
12	00001100
8	00001000
4	00000100



4 entries, each block holds one word, any block can hold any word.

- A cache that can put a line of data anywhere is called _____
- The most popular replacement strategy is *LRU* ( ).

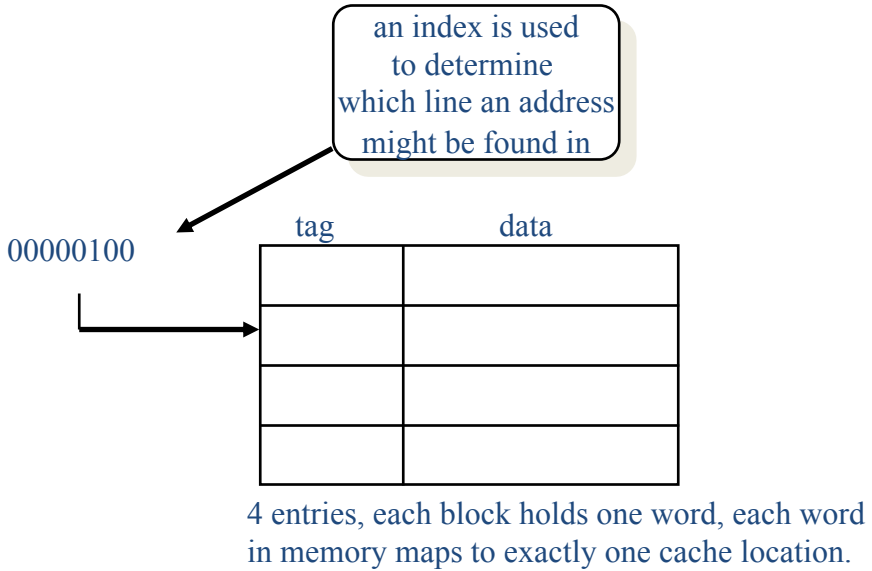
A simple cache



- A cache that can put a line of data anywhere is called **Fully Associative**
- The most popular replacement strategy is *LRU* (**Least Recently Used**).

A simpler cache

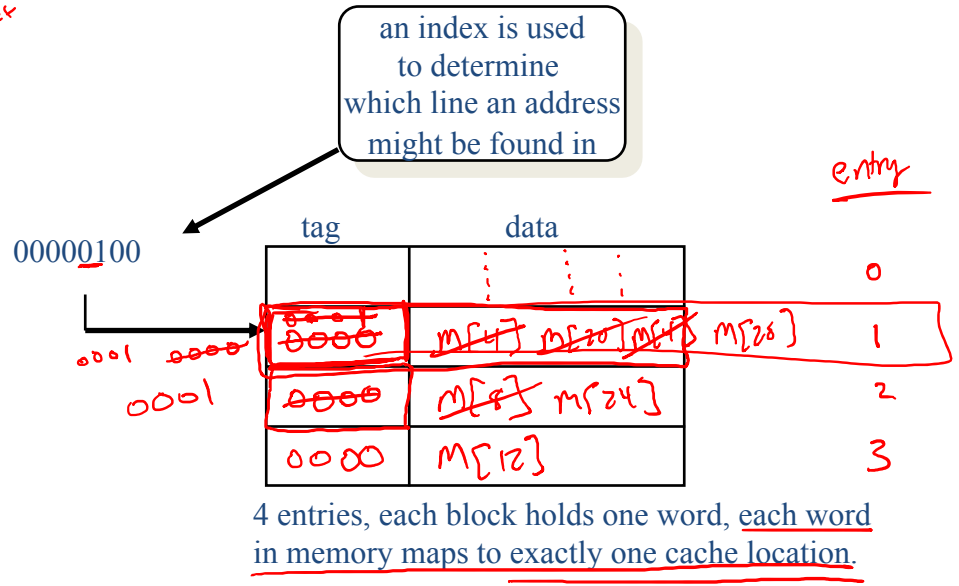
address string:	
4	00000100
8	00001000
12	00001100
4	00000100
8	00001000
20	00010100
4	00000100
8	00001000
20	00010100
24	00011000
12	00001100
8	00001000
4	00000100



- A cache that can put a line of data in exactly one place is called _____.
- Advantages/disadvantages vs. fully-associative?

A simpler cache

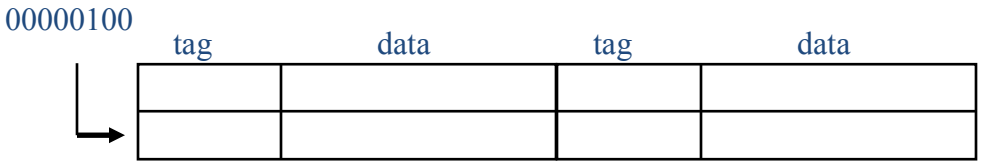
	address	string:
→ M	4	00000100
→ M	8	00001000
→ M	12	00001100
→ H	4	00000100
→ H	8	00001000
→ M	20	00010100
→ M	4	00000100
→ H	8	00001000
→ M	20	00010100
→ M	24	00011000
	12	00001100
	8	00001000
	4	00000100



- A cache that can put a line of data in exactly one place is called **direct mapped**
- Advantages/disadvantages vs. fully-associative?

A set-associative cache

address string:	
4	00000100
8	00001000
12	00001100
4	00000100
8	00001000
20	00010100
4	00000100
8	00001000
20	00010100
24	00011000
12	00001100
8	00001000
4	00000100

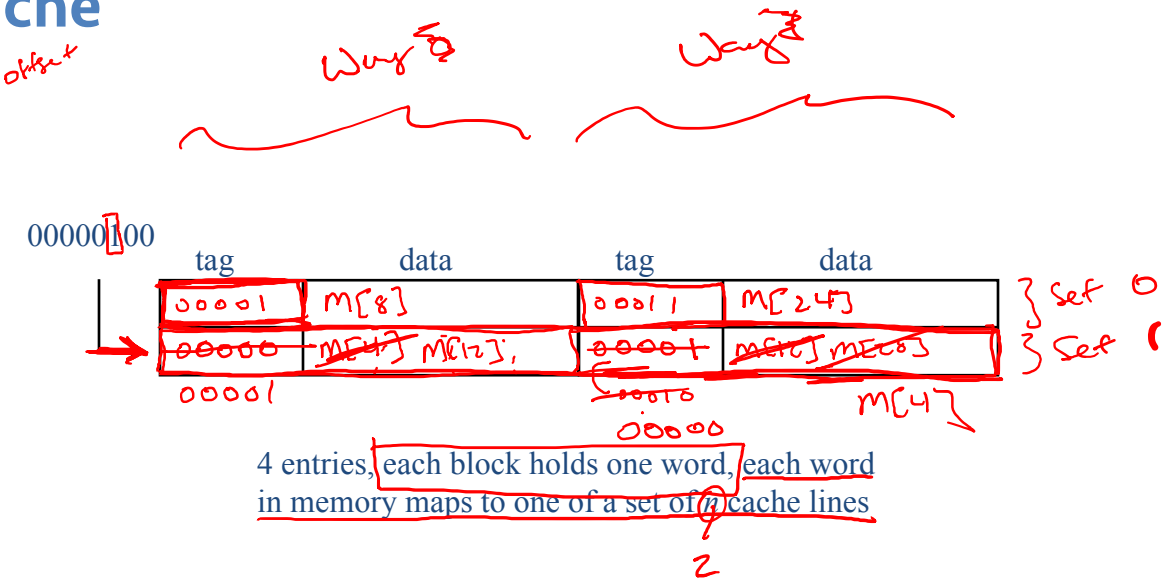


4 entries, each block holds one word, each word in memory maps to one of a set of *n* cache lines

- A cache that can put a line of data in exactly *n* places is called *n-way* _____.
- The cache lines/blocks that share the same index are a cache _____.

A set-associative cache

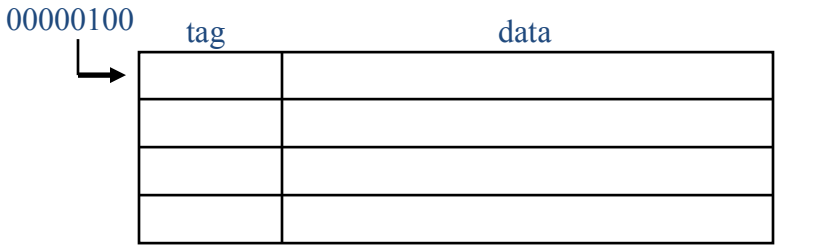
		address	string:
→	M	4	00000100
→	M	8	00001000
→	M	12	00001100
→	H	4	00000100
→	H	8	00001000
→	M	20	00010100
→	H	4	00000100
→	H	8	00001000
→	H	20	00010100
→	M	24	00011000
→	M	12	00001100
→	H	8	00001000
→	M	4	00000100



- A cache that can put a line of data in exactly *n* places is called ***n-way set-associative***.
- The cache lines/blocks that share the same index are a cache **set**.

Longer Cache Blocks

address	string:
4	00000100
8	00001000
12	00001100
4	00000100
8	00001000
20	00010100
4	00000100
8	00001000
20	00010100
24	00011000
12	00001100
8	00001000
4	00000100

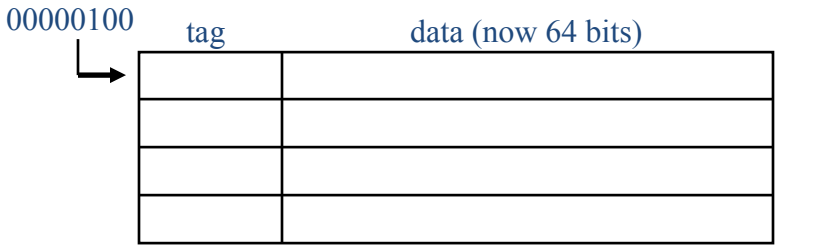


4 entries, each block holds two words, each word in memory maps to exactly one cache location (this cache is twice the total size of the prior caches).

- Large cache blocks take advantage of *spatial locality*.
- Too large of a block size can waste cache space.
- Longer cache blocks require less tag space

Longer Cache Blocks

address	string:
4	00000100
8	00001000
12	00001100
4	00000100
8	00001000
20	00010100
4	00000100
8	00001000
20	00010100
24	00011000
12	00001100
8	00001000
4	00000100



4 entries, each block holds two words, each word in memory maps to exactly one cache location (this cache is twice the total size of the prior caches).

- Large cache blocks take advantage of ***spatial locality***.
- Too large of a block size can waste cache space.
- Longer cache blocks require less tag space