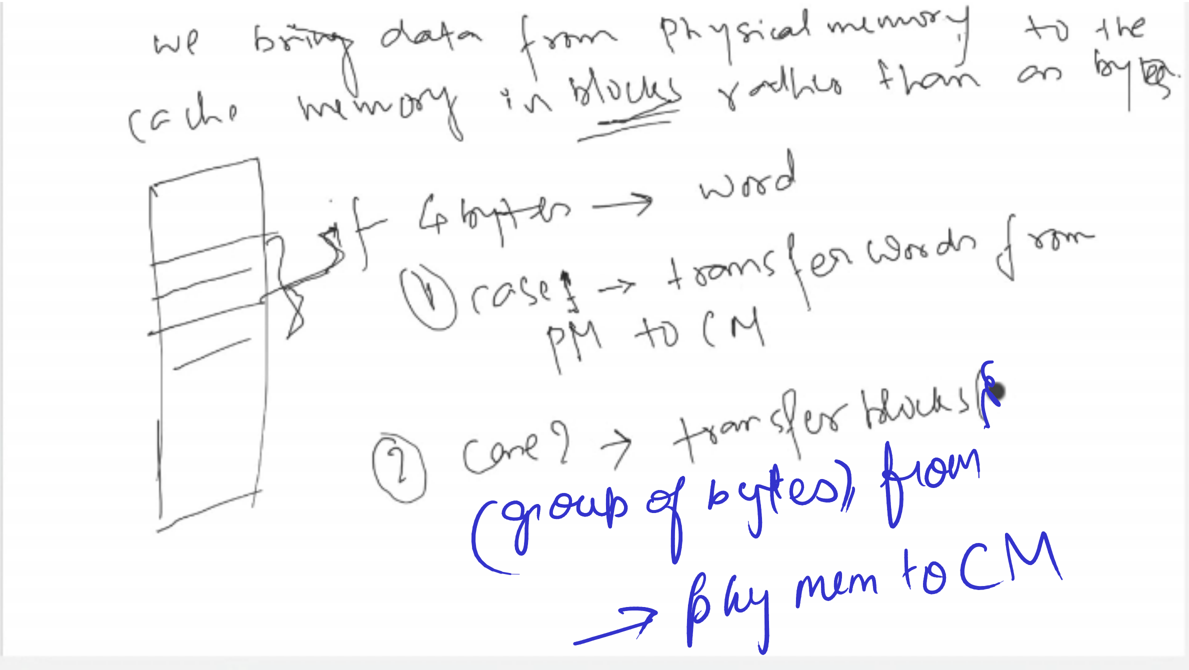
Line

=

Block



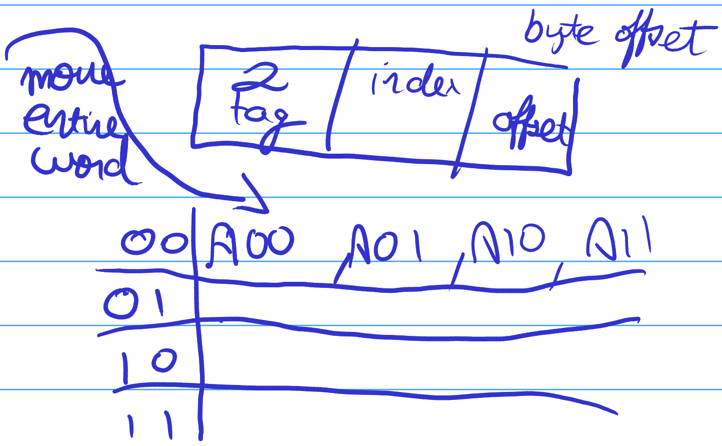
we bring data from physical memory to the cache memory in blocks rather than as bytes.

if 4 bytes → word

① case 1 → transfer words from PM to CM

② case 2 → transfer blocks (group of bytes) from → phy mem to CM

00	00	00
00	00	01
00	00	10
00	00	11
00	01	00
00	01	01
00	01	10
00	01	11



generalized location

k-n-2	n	2

for k bits

Case 2

Cache line → block of memory location in PM that is in CM

4 words / 1 block

index

word offset

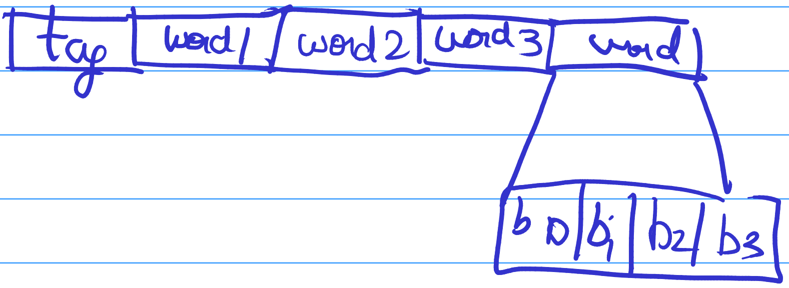
k-n-4	n	2	2

byte offset

Line = Block

→ set associative

→ direct mapped cache



Cache and Main Memory Structure

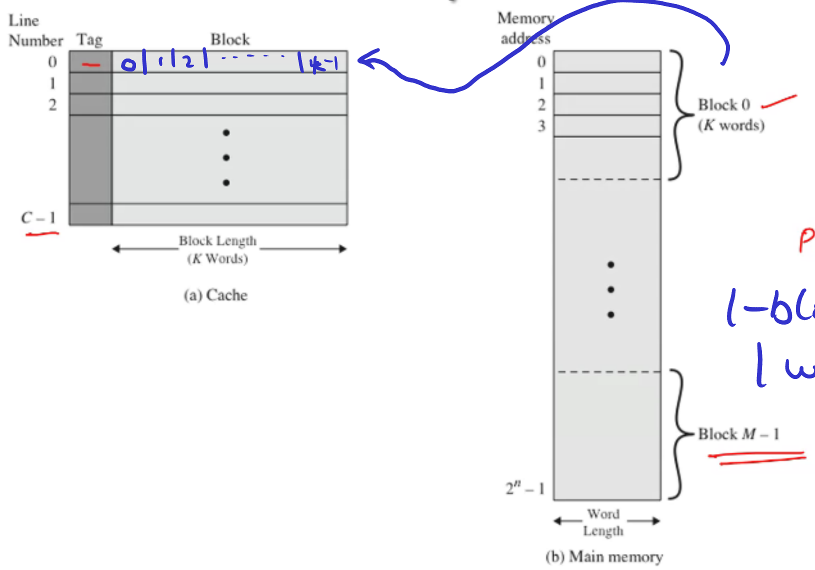


Figure 4.4 Cache/Main-Memory Structure

PM — M blocks

1-block - k words

1 word - 4 bytes

2^n addressable words

$2^n/K$ blocks

Cache contains C lines or slots C << M

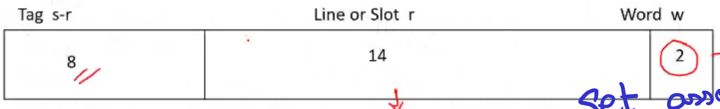
Cache Line 0 ← Block 0 eg 4k words in each block

Line 1 ← Block 1

...

Line C-1 Block($2^n/k$ - 1)

Direct Mapping



byte offset

index

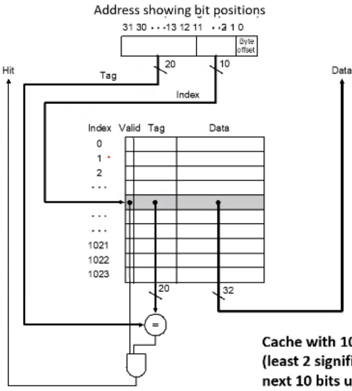
Set associative mapping

- 24 bit address PM
- 2 bit word identifier (4 byte block)
- 22 bit block identifier
  - 8 bit tag (=22-14)
  - 14 bit slot or line
- No two blocks in the same line have the same Tag field
- Check contents of cache by finding line and checking Tag

different tags cannot map into same place

32 bits , 2 bits are byte offset LSB

- MIPS style:



30 bits

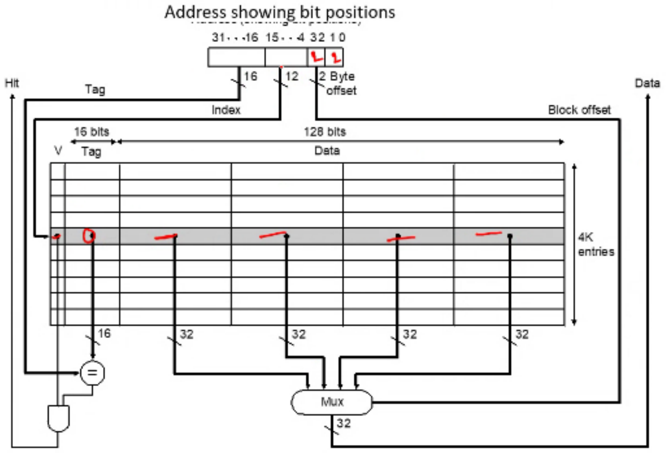
are used for mapping

Cache with 1024 1-word blocks: *byte offset* (least 2 significant bits) is ignored and next 10 bits used to index into cache

1 word per cache line

Direct Mapped Cache: Taking Advantage of Spatial Locality

- Taking advantage of spatial locality with *larger* blocks:



Cache with 4K 4-word blocks: *byte offset* (least 2 significant bits) is ignored, next 2 bits are *block offset*, and the next 12 bits are used to index into cache

Spatial Locality may cause dependence b/w words in block

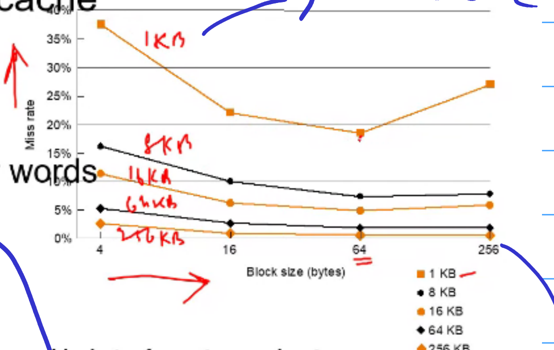
⇒

Direct Mapped Cache: Taking Advantage of Spatial Locality

- *Cache replacement* in large (multiword) blocks:
  - word *read miss*: read entire block from main memory
  - word *write miss*: *cannot* simply write word and tag! *Why*?! ?!
  - writing in a *write-through* cache:
    - if *write hit*, i.e., tag of requested address and and cache entry are equal, continue as for 1-word blocks by replacing word and writing block to both cache and memory
    - if *write miss*, i.e., tags are unequal, fetch block from memory, replace word that caused miss, and write block to both cache and memory
    - therefore, unlike case of 1-word blocks, a write miss with a multiword block causes a memory read

- Miss rate falls at first with increasing block size as expected, but, as block size becomes a large fraction of total cache size, miss rate may go up because
  - there are few blocks
  - competition for blocks increases
  - blocks get ejected before most of their words are accessed (*thrashing* in cache)

cache size



Miss rate vs. block size for various cache sizes

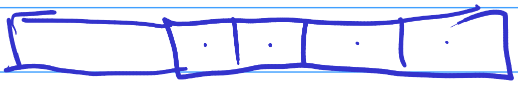
Too many block replacements due to temporal locality

multiprogramming

high cache
=> power consumption ↑
cost ↑
space is small

128 kB cache data
1 w block size
32 bits space

total bits?



32 k words $= 2^{15}$

$$\overset{\text{entry data}}{32} + \overset{\text{tag}}{(32-15-2)} + \overset{\text{valid}}{1} = 48 \text{ bits}$$

$$48 \times 2^{15} = (1.5 \times 32) \times 2^{15}$$

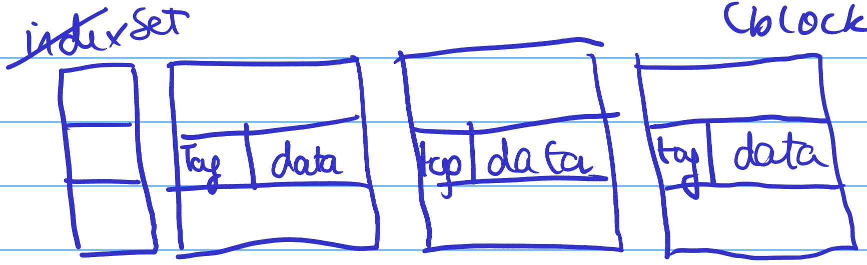
$$= 1.5 \times 2^{20} \text{ bits}$$

data bits = 1 Mbits

total cache size / actual cache data = 1.5

Set associative mapping

In set associative mapped cache, same indices in the cache can store > 1 line (block)

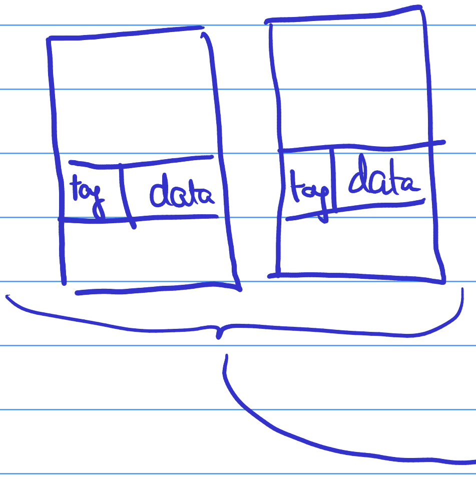


watch lecs

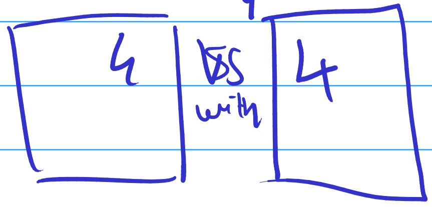
Main memory - 128 bytes. locations are byte addressable

CM - 32 bytes
BS - 4 bytes
Set - 2 way set associative

$\frac{128}{4} = \boxed{32}$



Lines: $\frac{32}{4} = 8$ lines



no. of sets = $\frac{\text{no. of lines}}{2 \text{ no. of line/set}} = 4$

BS -> size: 2 bits for Block offset

tag	set no.	bytes ~~block~~ offset
3	2	2

} index

associated

Now, to which # in same set do we place

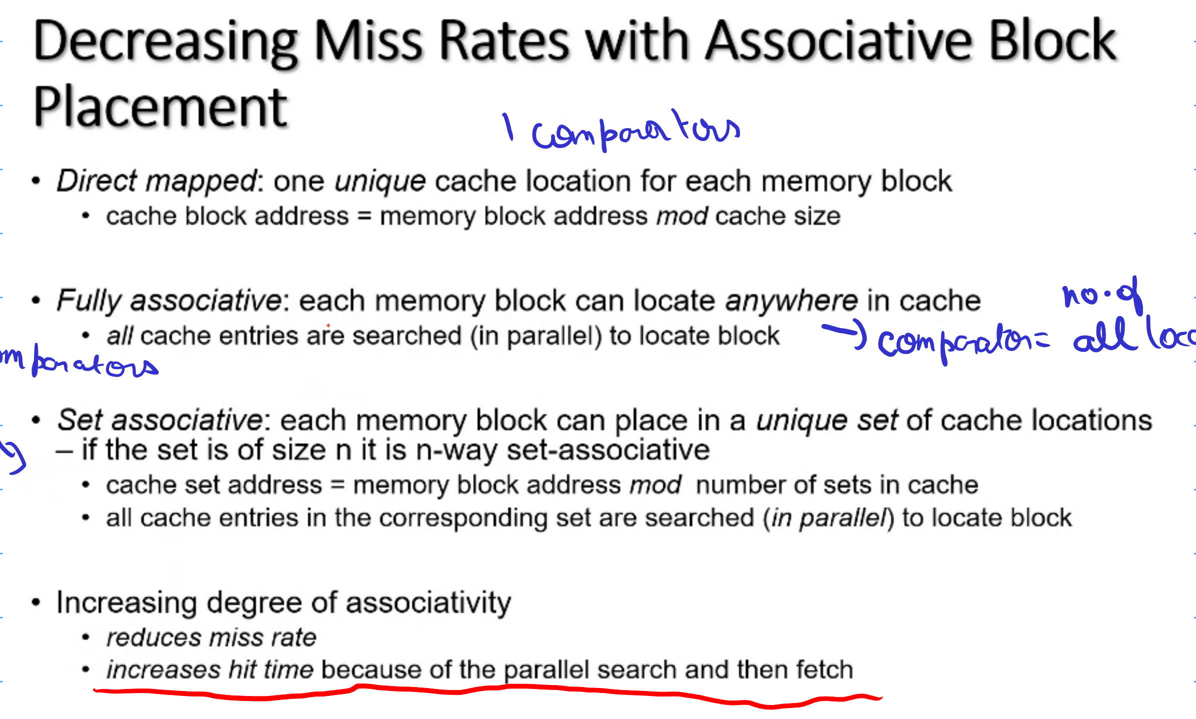
Step 1 find the set no.

Step 2 Compare the tag (valid bit first)

n way set associative

= n comparators

each comparator has no of bits in tag



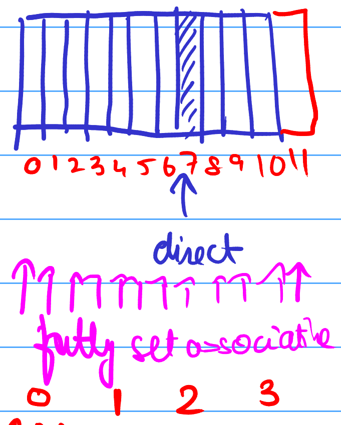
Decreasing Miss Rates with Associative Block Placement

1 comparators

- *Direct mapped*: one *unique* cache location for each memory block
  - cache block address = memory block address *mod* cache size
- *Fully associative*: each memory block can locate *anywhere* in cache
  - *all* cache entries are searched (in parallel) to locate block → no. of comparator= all locations
- *Set associative*: each memory block can place in a *unique set* of cache locations – if the set is of size n it is n-way set-associative
  - cache set address = memory block address *mod* number of sets in cache
  - all cache entries in the corresponding set are searched (*in parallel*) to locate block

n-comparators

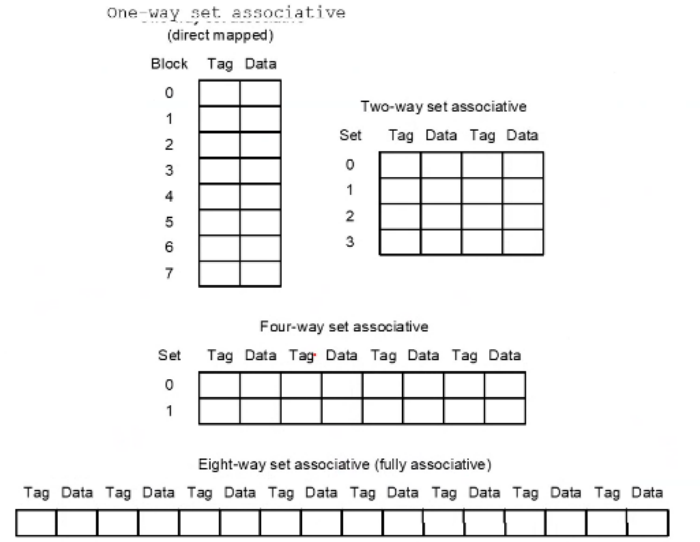
- Increasing degree of associativity
  - *reduces miss rate*
  - *increases hit time* because of the parallel search and then fetch



↑↑↑ → n-way associative (Compromise)

it's like small fully associative cache

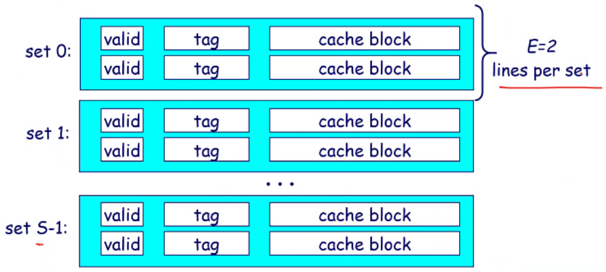
Decreasing Miss Rates with Associative Block Placement


One-way set associative
(direct mapped)
Block Tag Data
0 1 2 3 4 5 6 7
Two-way set associative
Set Tag Data Tag Data
0 1 2 3
Four-way set associative
Set Tag Data Tag Data Tag Data Tag Data
0 1
Eight-way set associative (fully associative)
Tag Data Tag Data Tag Data Tag Data Tag Data Tag Data Tag Data Tag Data

Configurations of an 8-block cache with different degrees of associativity

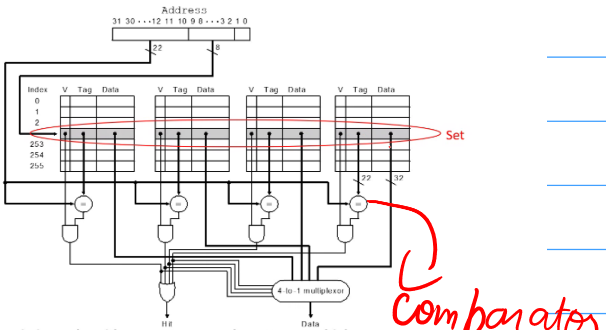
Example: Set Associative Cache

Characterized by more than one line per set



E-way associative cache

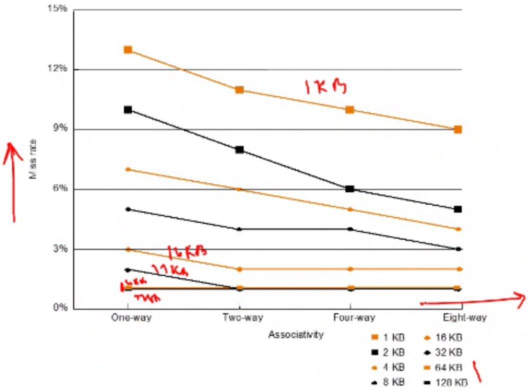
Implementation of a Set-Associative Cache



4-way set-associative cache with 4 comparators and one 4-to-1 multiplexor: size of cache is 1K blocks = 256 sets * 4-block set size

Comparator

Performance with Set-Associative Caches

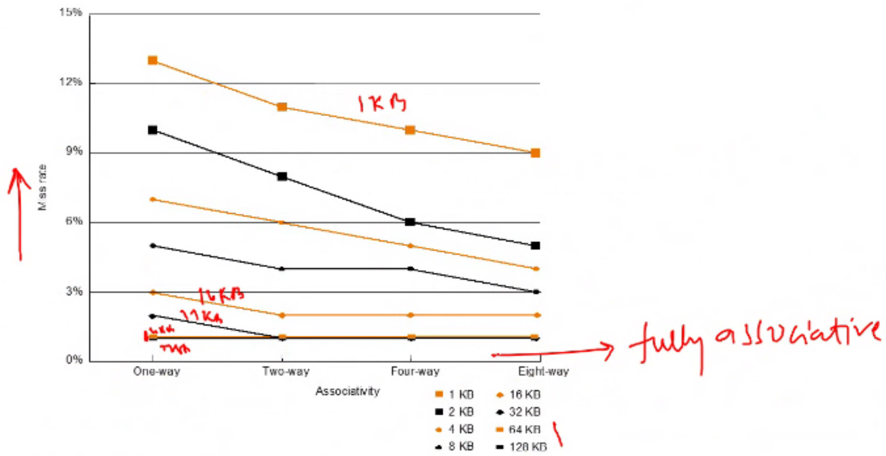


fully associative

Miss rates for each of eight cache sizes with increasing associativity: data generated from SPEC92 benchmarks with 32 byte block size for all caches

08068

Performance with Set-Associative Caches



Miss rates for each of eight cache sizes with increasing associativity:
data generated from SPEC92 benchmarks with 32 byte block size for all caches

Example Problems

- *Find the number of misses for a cache with four 1-word blocks given the following sequence of memory block accesses:* *0, 8, 0, 6, 8,* ← mem. references.
  for each of the following cache configurations
  1. *direct mapped*
  2. *2-way set associative (use LRU replacement policy)* Least recently used
  3. *fully associative*

- Note about LRU replacement
  - in a 2-way set associative cache LRU replacement can be implemented with one bit at each set whose value indicates the mostly recently referenced block