BCS-29
Advanced Computer Architecture

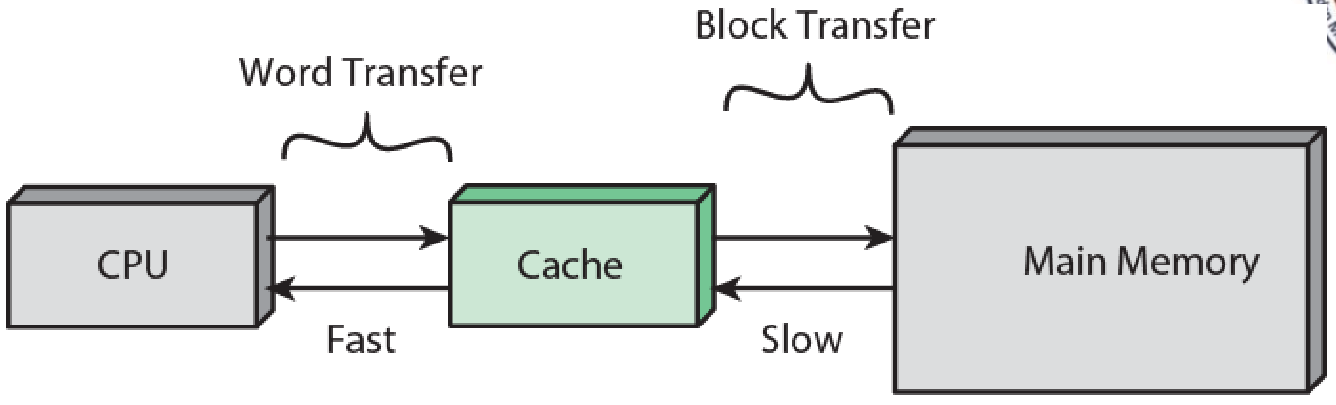
Memory Hierarchy Design

Memory Subsystem

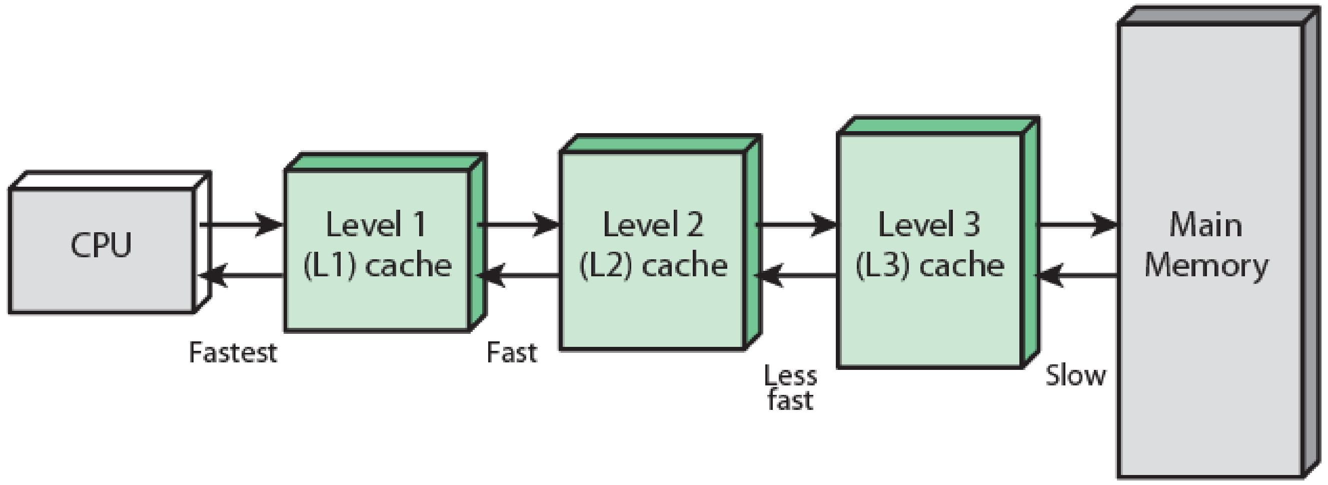
Hierarchy List

- Registers
- L1 Cache
- L2 Cache
- Main memory
- Disk cache
- Disk
- Optical
- Tape

Cache and Main Memory

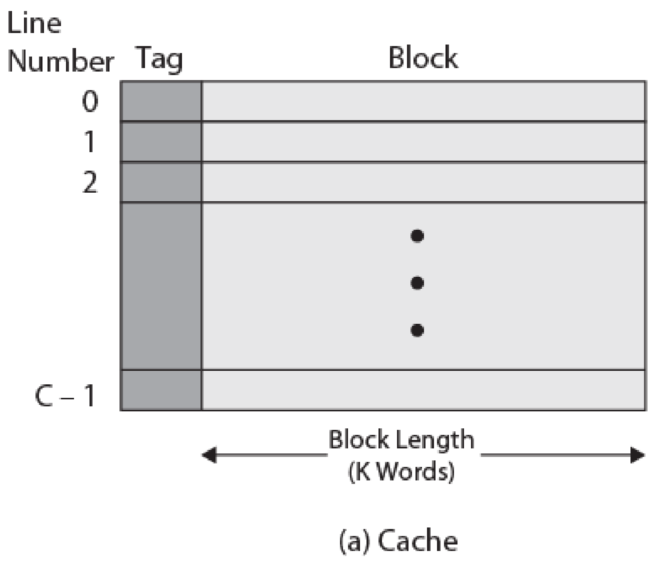


(a) Single cache

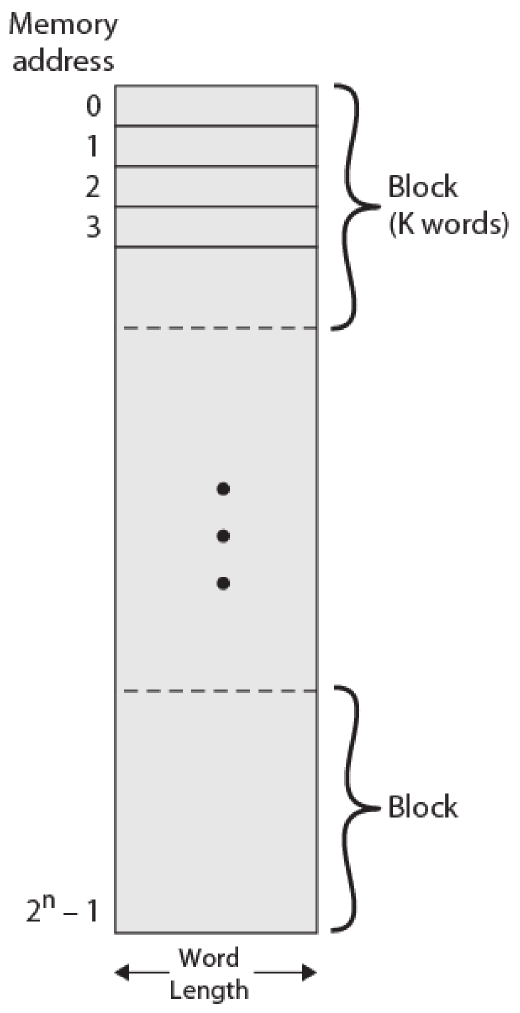


(b) Three-level cache organization

Cache/Main Memory Structure



(a) Cache

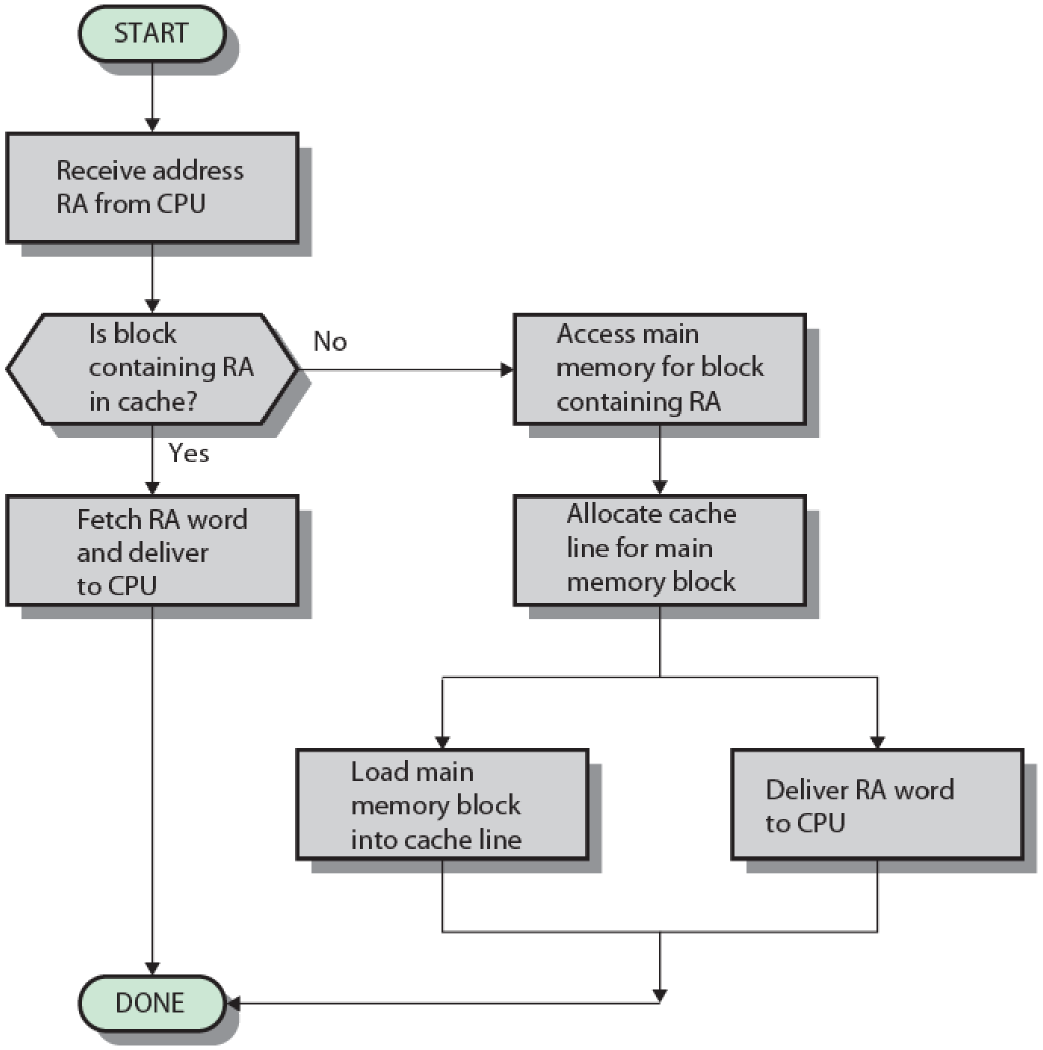


(b) Main memory

Cache operation – overview

- CPU requests contents of memory location
- Check cache for this data
- If present, get from cache (fast)
- If not present, read required block from main memory to cache
- Then deliver from cache to CPU
- Cache includes tags to identify which block of main memory is in each cache slot

Cache Read Operation - Flowchart


START
Receive address RA from CPU
Is block containing RA in cache?
No
Yes
Access main memory for block containing RA
Fetch RA word and deliver to CPU
Allocate cache line for main memory block
Load main memory block into cache line
Deliver RA word to CPU
DONE

How does cache memory work?

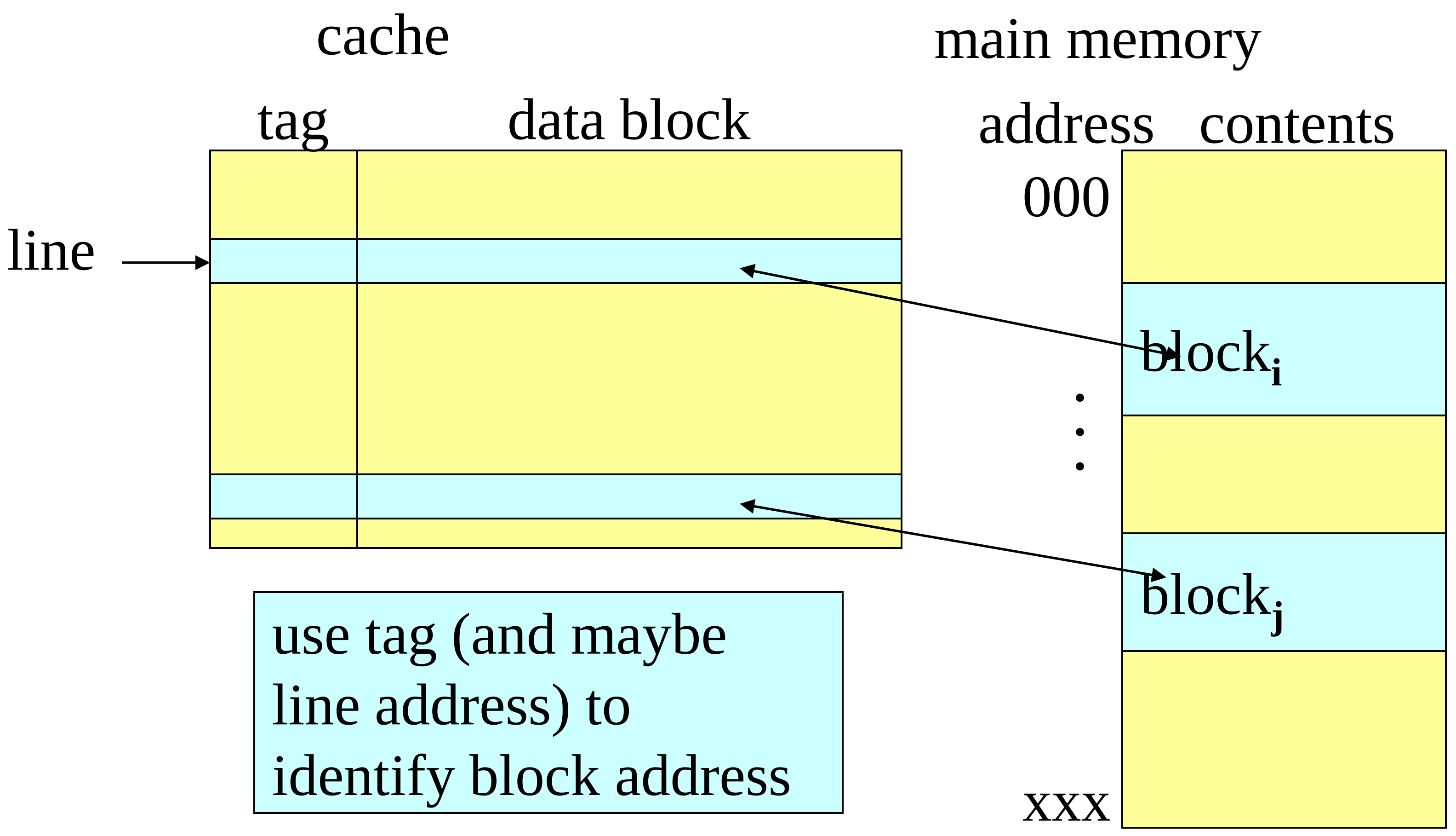
- Main memory consists of up to 2^n addressable words, with each word having a unique *n*-bit address. For mapping purposes, this memory is considered to consist of a number of fixed-length blocks of *K* words each. Thus, there are $M = 2^n/K$ blocks.
- The cache is split into *C lines* of *K* words each, Figure with number of lines considerably smaller than the number of main memory blocks ($C << M$).
- At any time, only a subset of the blocks of main memory resides in the lines in the cache.
- If a word in a block of memory is read, that block is transferred to one of the lines of the cache.

Cache Design

- Addressing
- Size
- Mapping Function
- Replacement Algorithm
- Write Policy
- Block Size
- Number of Caches

Mapping Function

- how does cache contents map to main memory contents?



Cache Basics

- cache line vs. main memory location
  - same concept – avoid confusion (?)
  - line has address and contents
- contents of cache line divided into tag and data fields
  - fixed width
  - fields used differently !
  - data field holds contents of a block of main memory
  - tag field helps identify the start address of the block of memory that is in the data field

Mapping Function Example

- cache of 64 KByte ← holds up to 64 Kbytes of main memory contents
  - 16 K (2^{14}) lines – each line is 5 bytes wide = 40 bits

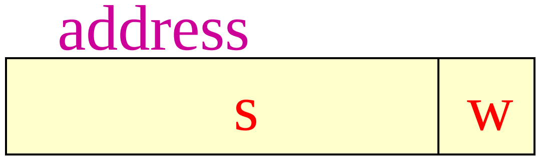
tag field: 1 byte

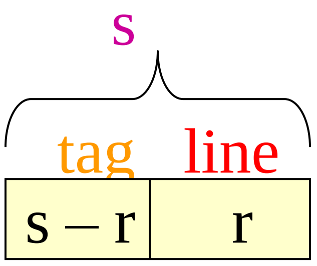
data field: 4 bytes ← 4 byte blocks of main memory

- 16 MBytes main memory
- 24 bit address
  - 2^{24} = 16 M
- will consider DIRECT and ASSOCIATIVE mappings

Direct Mapping

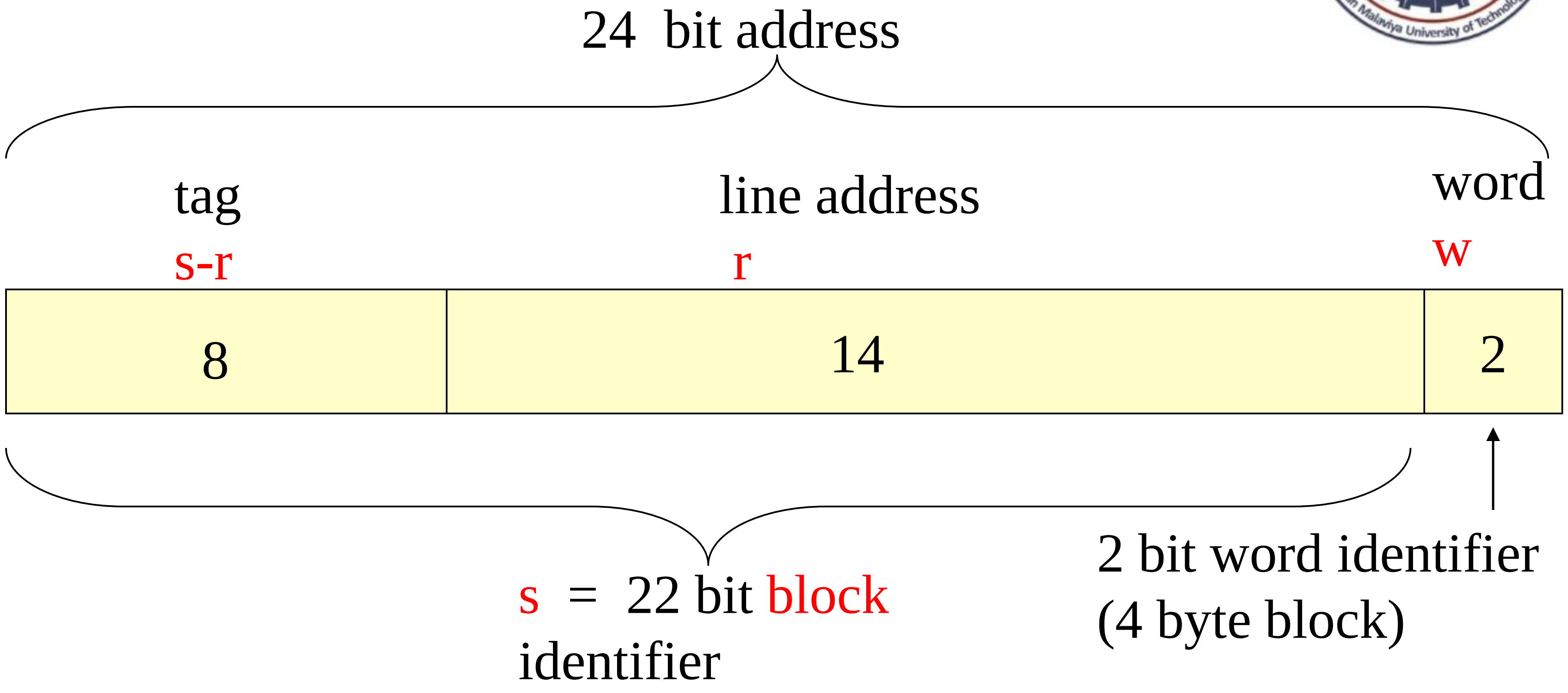
- each block of main memory maps to only one cache line
  - i.e. if a block is in cache, it must be in one specific place – based on address!
- split address into two parts
  - least significant w bits identify unique word in block
  - most significant s bits specify one memory block
    - split s bits into:
      - cache line address field r bits
      - tag field of s-r most significant bits





line field identifies line containing block !

Direct Mapping: Address Structure Example



- two blocks may have the same r value, but then always have different tag value !

Direct Mapping Cache Line Table

each block = 4 bytes

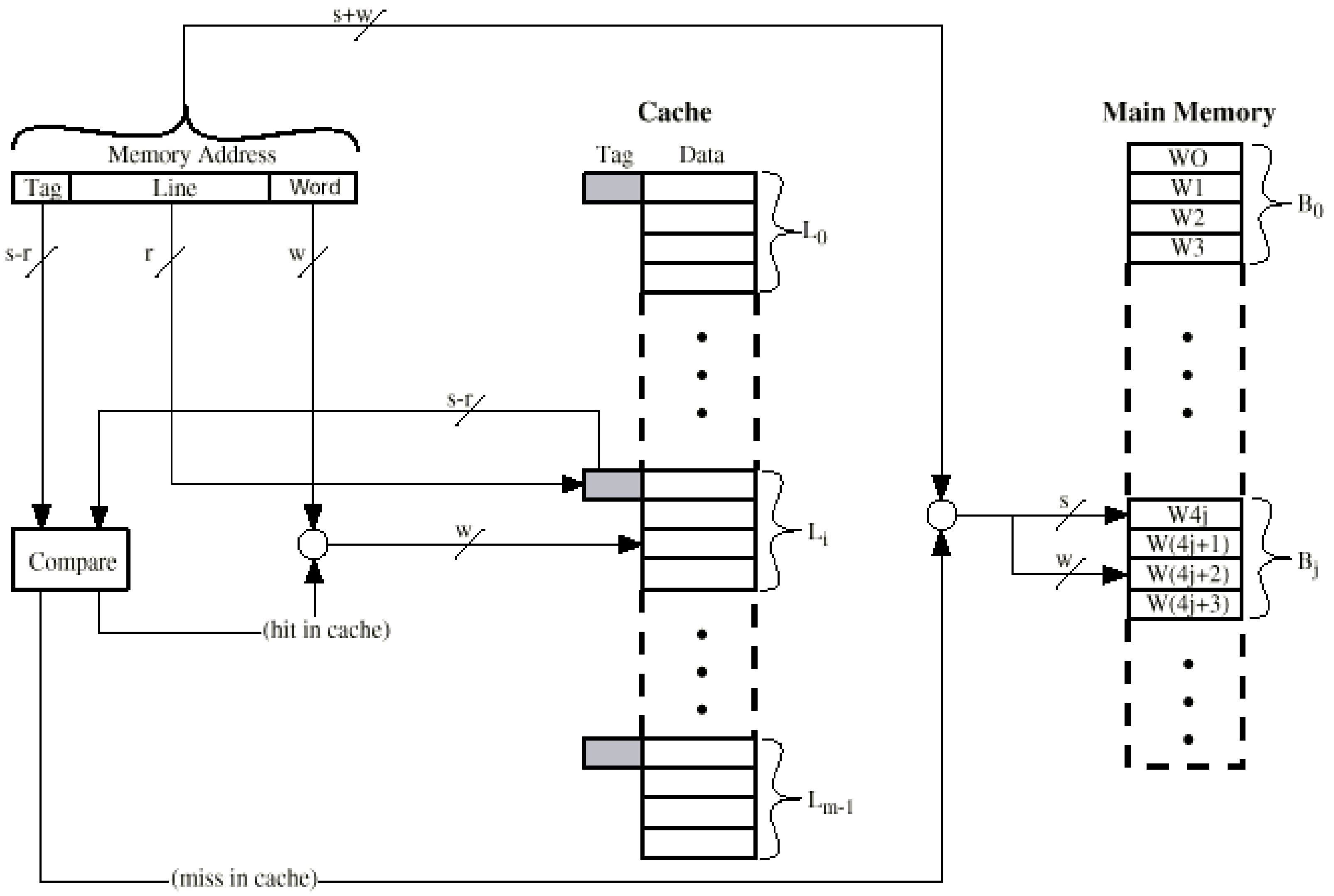
cache line		main memory blocks held
0	$\longleftrightarrow$	0, m, 2m, 3m, … 2^s-m
1	$\longleftrightarrow$	1, m+1, 2m+1, … 2^s-m+1
⋮		⋮
m-1	$\longleftrightarrow$	m-1, 2m-1,3m-1, … 2^s-1

s=22

m=2^{14}

But…a line can contain only one of these at a time!

Direct Mapping Cache Organization



Direct Mapping Summary

- Address length = (s + w) bits
- Number of addressable units = 2^{s+w} words or bytes
- Block size = line size – tag size = 2^{w} words or bytes
- Number of blocks in main memory = $2^{s+w}/2^{w} = 2^{s}$
- Number of lines in cache = m = 2^{r}
- Size of tag = (s – r) bits

Direct Mapping pros & cons

- Simple
- Inexpensive
- Fixed location for given block
  - If a program accesses 2 blocks that map to the same line repeatedly, cache misses are very high

Associative Mapping

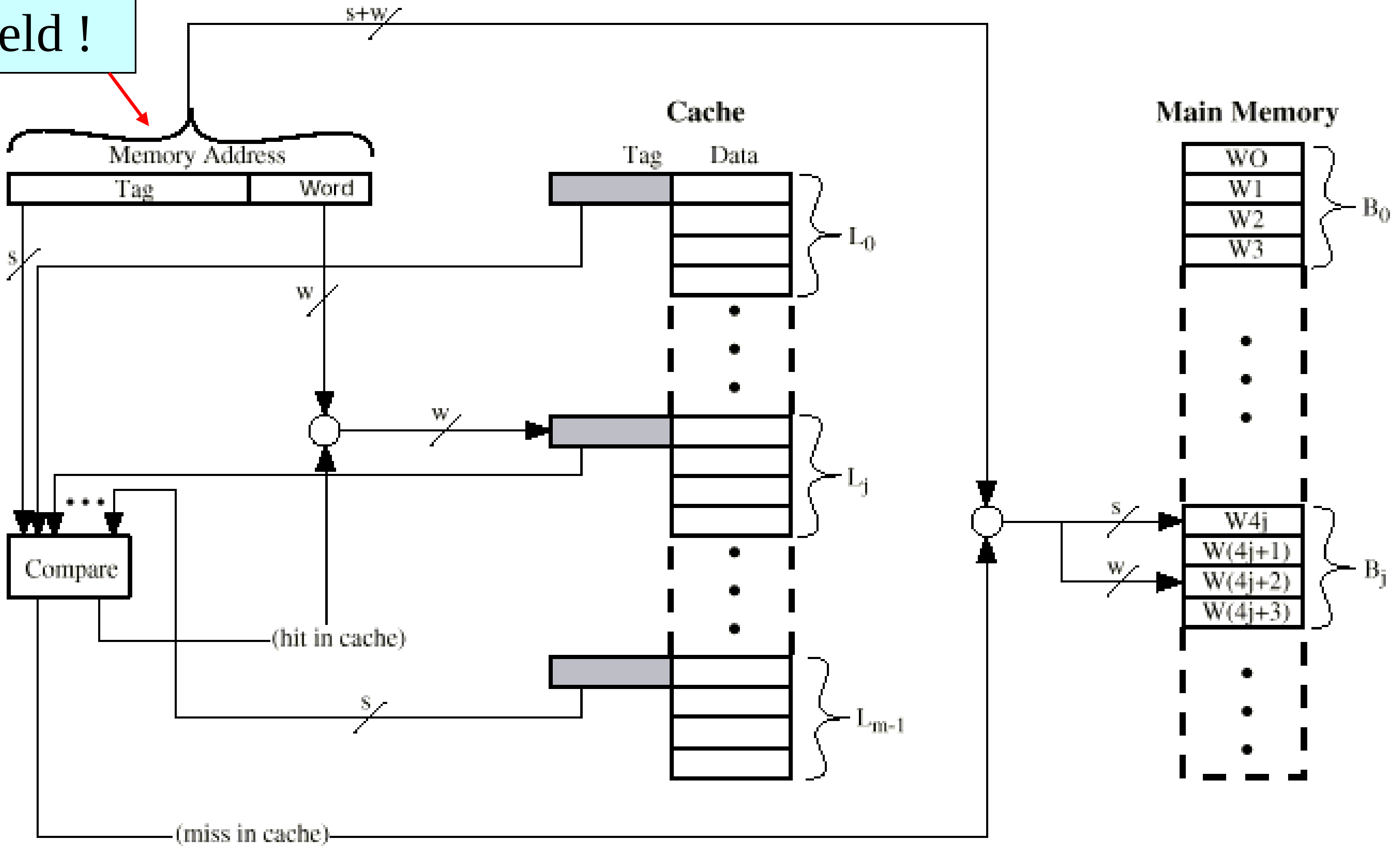
- main memory block can load into any line of cache
- memory address is interpreted as tag and word select in block

s = tag → does not use line address !

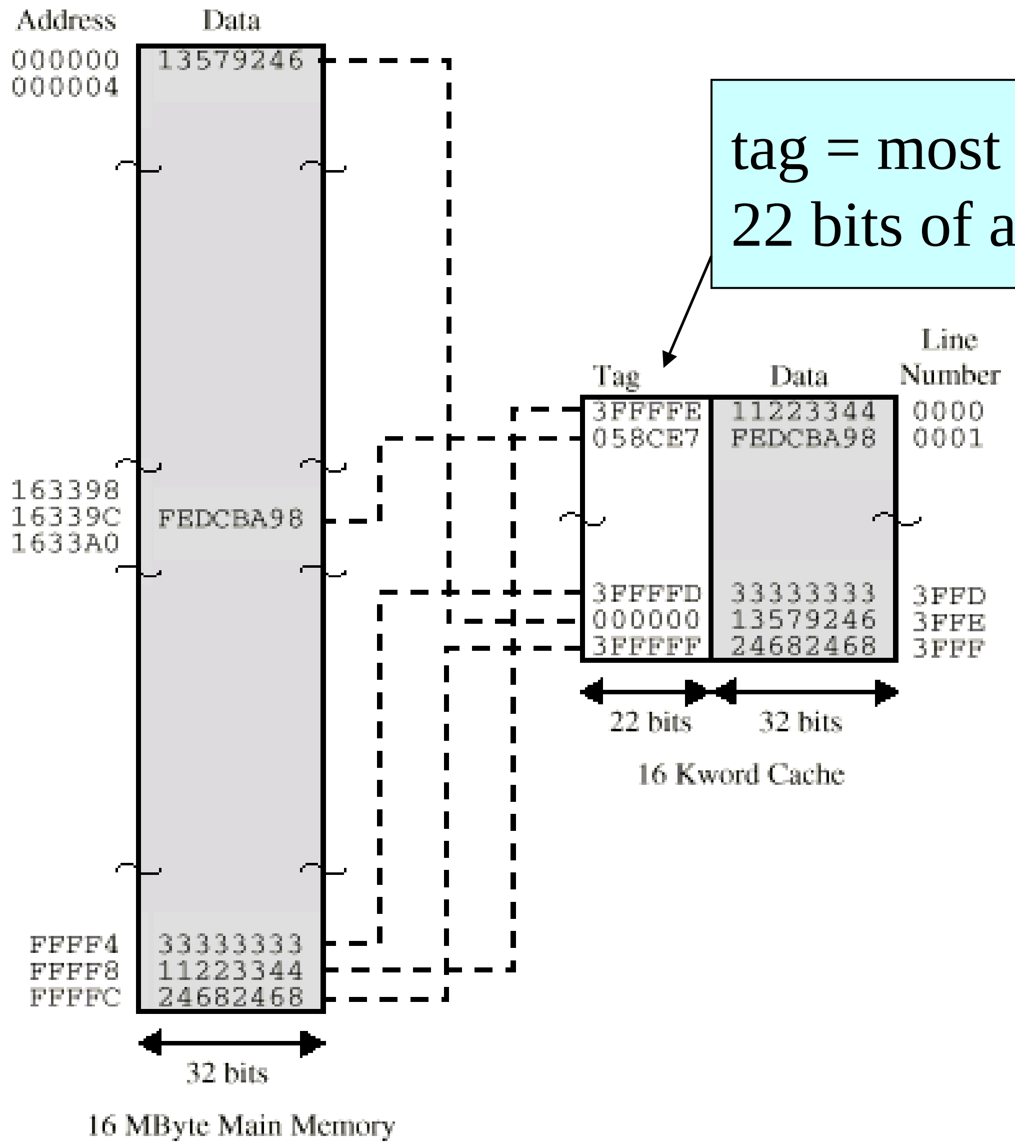
- tag uniquely identifies block of memory !
- every line's tag is examined for a match
- cache searching gets expensive

Fully Associative Cache Organization

no line field !



Associative Mapping Example



tag = most signif. 22 bits of address

Typo- leading F missing!

Associative Mapping (Address Structure)

Tag 22 bit	Word 2 bit

- 22 bit tag stored with each 32 bit block of data
- Compare tag field with tag entry in cache to check for hit
- Least significant 2 bits of address identify which 8 bit word is required from 32 bit data block
- e.g.

Address	Tag	Data	Cache line
FFFFFC	3FFFFF	24682468	any, e.g. 3FFF

Associative Mapping Summary

- Address length = (s + w) bits
- Number of addressable units = 2^{s+w} words or bytes
- Block size = line size – tag size = 2^{w} words or bytes
- Number of blocks in main memory = $2^{s+ w}/2^{w} = 2^{s}$
- Number of lines in cache = undetermined
- Size of tag = s bits

Set Associative Mapping

- Cache is divided into a number of sets
- Each set contains k lines → k – way associative
- A given block maps to any line in a given set
  - e.g. Block B can be in any line of set i
- e.g. 2 lines per set
  - 2 – way associative mapping
  - A given block can be in one of 2 lines in only one set

Set Associative Mapping Address Structure

Tag 9 bit	Set 13 bit	Word 2 bit

E.g. Given our 64Kb cache, with a line size of 4 bytes, we have 16384 lines. Say that we decide to create 8192 sets, where each set contains 2 lines. Then we need 13 bits to identify a set (2^{13}=8192)

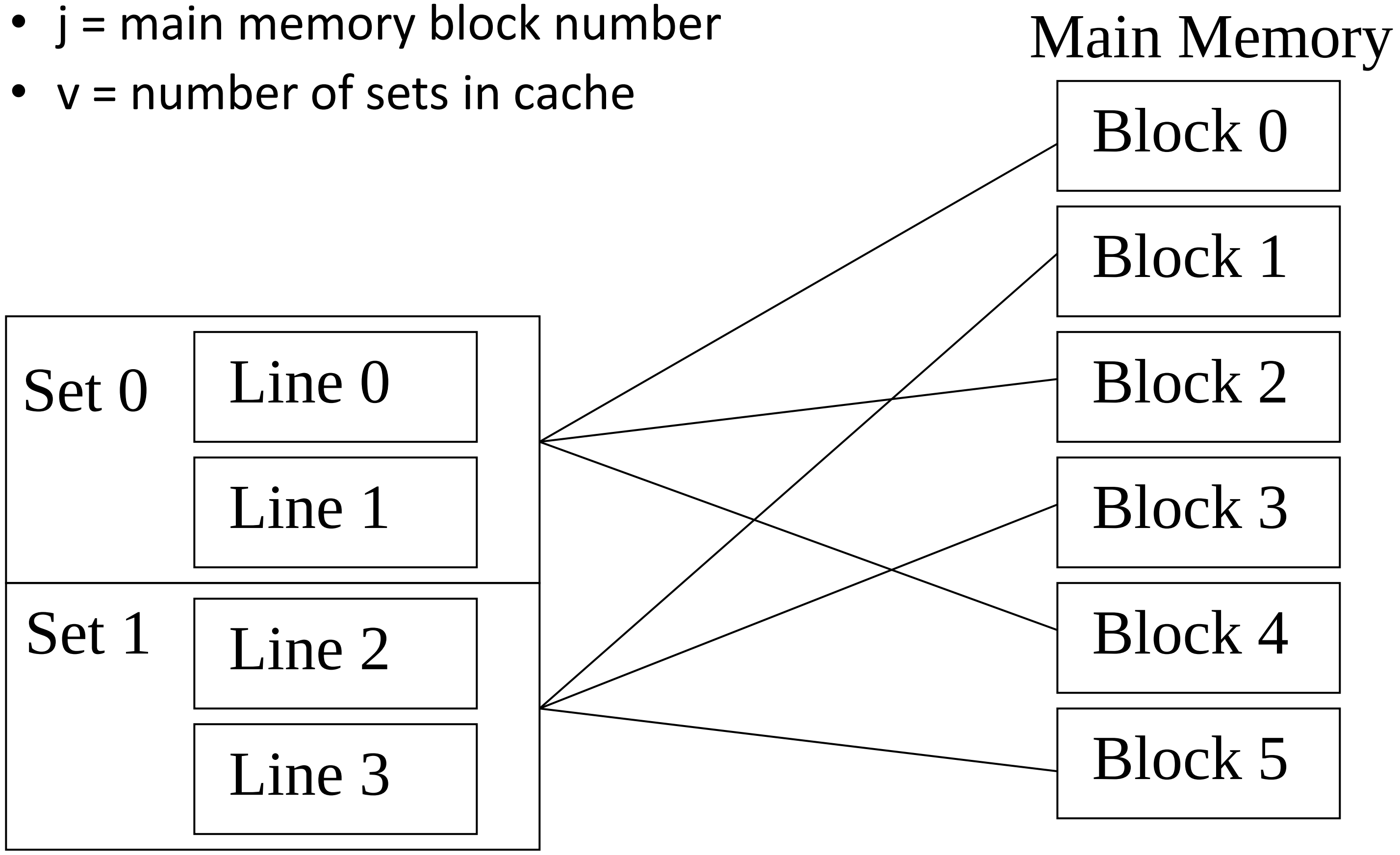
Use set field to determine cache set to look in

Compare tag field of all slots in the set to see if we have a hit, e.g.:

- Address = 16339C = 0001 0110 0011 0011 1001 1100
  - Tag = 0 0010 1100 = 02C
  - Set = 0 1100 1110 0111 = 0CE7
  - Word = 00 = 0
- Address = 008004 = 0000 0000 1000 0000 0000 0100
  - Tag = 0 0000 0001 = 001
  - Set = 0 0000 0000 0001 = 0001
  - Word = 00 = 0

Set Associative Mapping

- To compute cache set number:
  - SetNum = j mod v
    - j = main memory block number
    - v = number of sets in cache

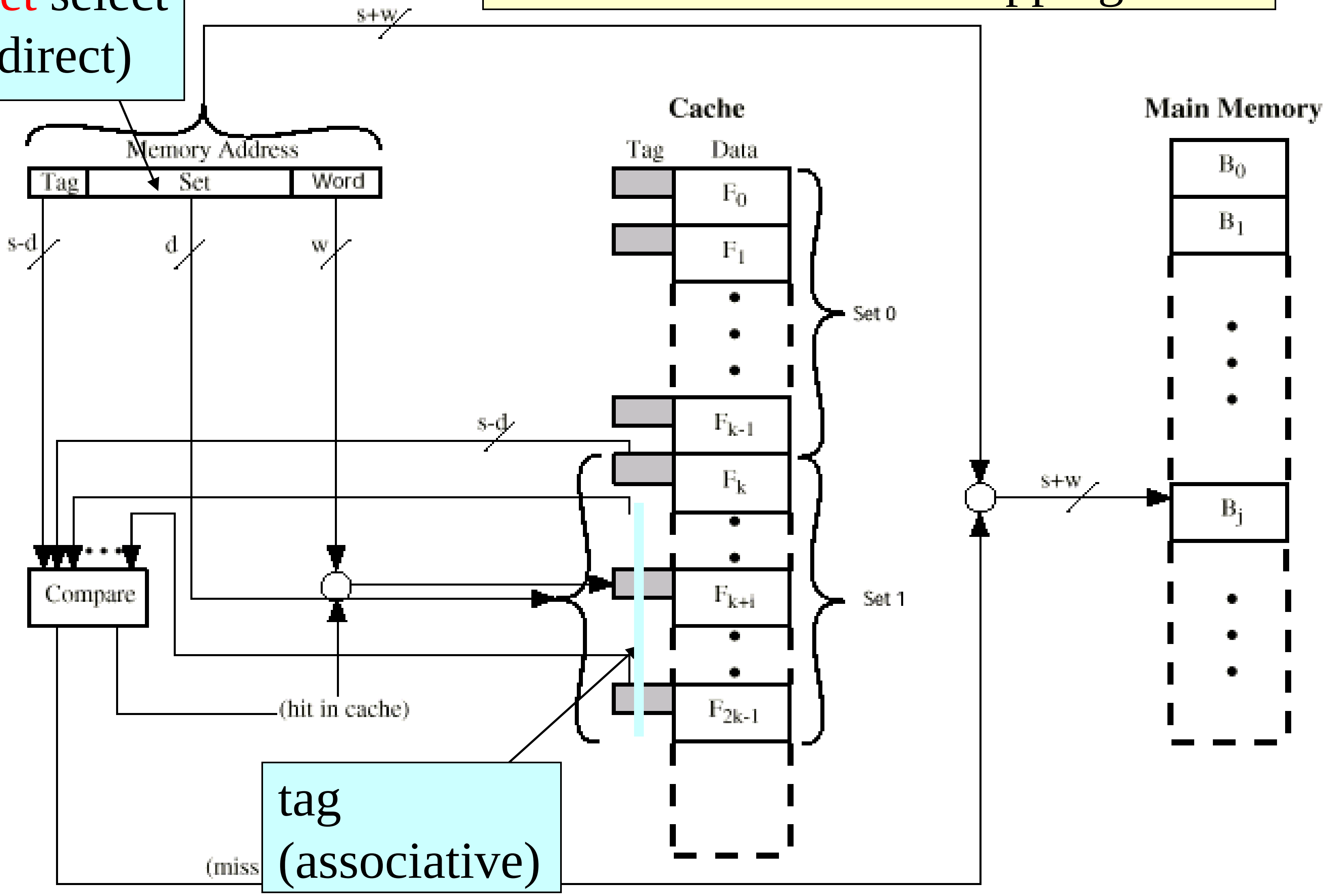


K-Way Set Associative Cache Organization

Direct + Associative Mapping

set select (direct)

tag (associative)



Breaking into Tag, Set, Word

- Given Tag=9 bits, Set=13 bits, Word=2 bits
- Given address FFFFFD_{16}
- What are values of Tag, Set, Word?
    - First 9 bits are Tag, next 13 are Set, next 2 are Word
    - Rewrite address in base 2: 1111 1111 1111 1111 1111 1101
    - Group each field in groups of 4 bits starting at right
    - Add *zero bits* as necessary to leftmost group of bits
- *000*1 1111 1111 *000*1 1111 1111 1111 *000*1
- → 1FF 1FFF 1 (Tag, Set, Word)

Replacement Algorithms Direct Mapping

- what if bringing in a new block, but no line available in cache?
- must replace (overwrite) a line – which one?

- direct → no choice
  - each block only maps to one line
- replace that line

Replacement Algorithms (Associative & Set Associative)

- hardware implemented algorithm (speed)
- Least Recently Used (LRU)
- e.g. in 2-way set associative
  - which of the 2 blocks is LRU?
- First In first Out (FIFO)
  - replace block that has been in cache longest
- Least Frequently Used (LFU)
  - replace block which has had fewest hits
- Random

Write Policy

- must not overwrite a cache block unless main memory is up to date
- Complication: Multiple CPUs may have individual caches!!
- Complication: I/O may address main memory too (read and write)!!
- N.B. 15% of memory references are writes

Write Through Method

- all writes go to main memory as well as cache
- Each of multiple CPUs can monitor main memory traffic to keep its own local cache up to date
- lots of traffic → slows down writes

Write Back Method

- updates initially made in cache only
- update (dirty) bit for cache slot is set when update occurs
- if block is to be replaced, write to main memory only if update bit is set
- Other caches get out of sync
- I/O must access main memory through cache

Example-1

- M1 : 16K word 50 ns access time
- M2: 1M words 400 ns Access time
- If 8 word cache line and set size is 256 words with set associative mapping
- Give Mapping between M2 and M1
- Calculate Effective memory access time (Cache hit ratio =0.95)

Cache Performance

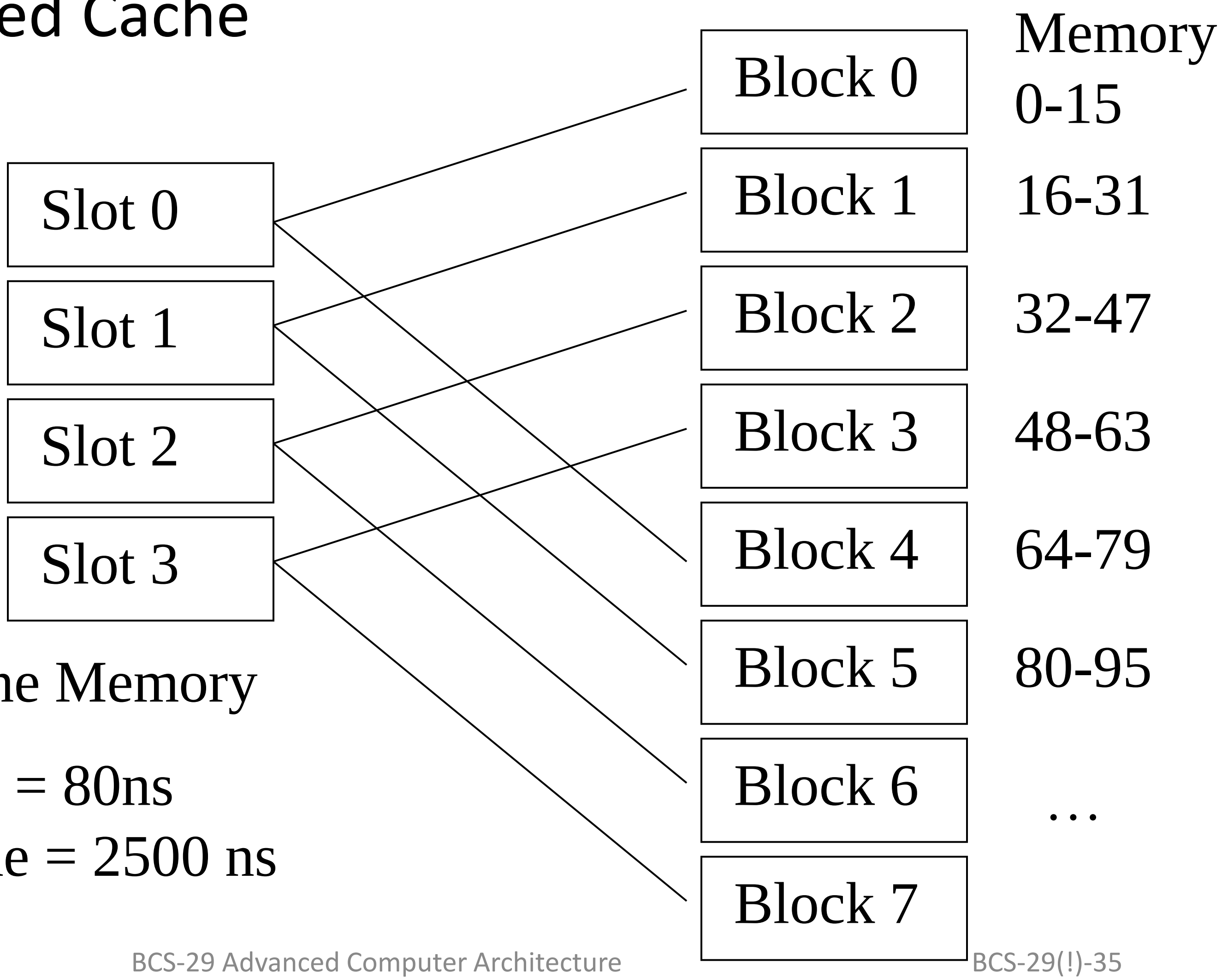
- Two measures that characterize the performance of a cache are the **hit ratio** and the **effective access time**

$$\text{Hit Ratio} = \frac{\text{(Num times referenced words are in cache)}}{\text{(Total number of memory accesses)}}$$

$$\text{Eff. Access Time} = \frac{\text{(\# hits)(TimePerHit)+(\# misses) (TimePerMiss)}}{\text{(Total number of memory accesses)}}$$

Cache Performance Example

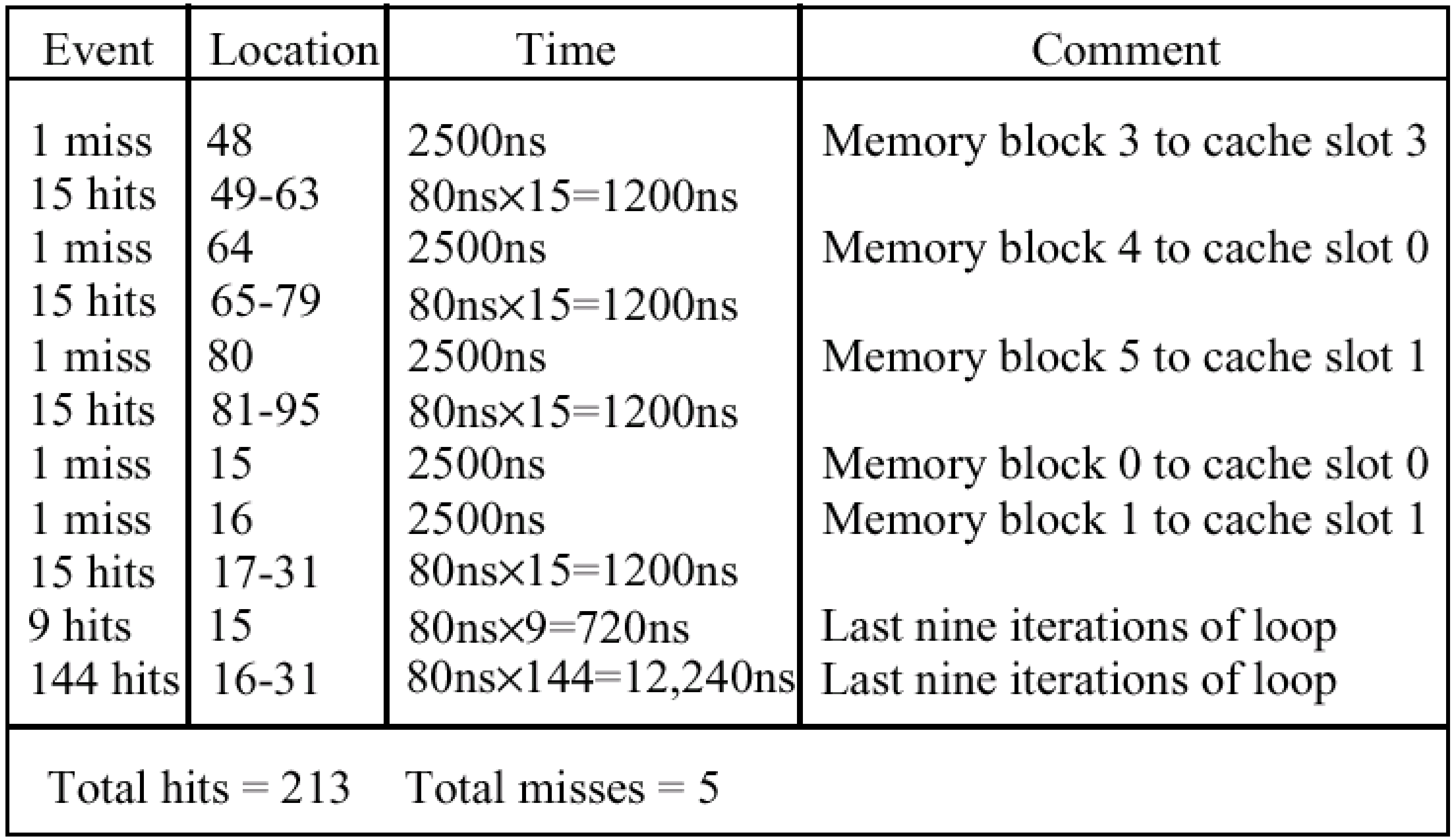
- Direct-Mapped Cache



Cache access time = 80ns
Main Memory time = 2500 ns

Cache Performance Example

- Sample program executes from memory location 48-95 once. Then it executes from 15-31 in a loop ten times before exiting.



Event	Location	Time	Comment
1 miss	48	2500ns	Memory block 3 to cache slot 3
15 hits	49-63	80ns×15=1200ns	
1 miss	64	2500ns	Memory block 4 to cache slot 0
15 hits	65-79	80ns×15=1200ns	
1 miss	80	2500ns	Memory block 5 to cache slot 1
15 hits	81-95	80ns×15=1200ns	
1 miss	15	2500ns	Memory block 0 to cache slot 0
1 miss	16	2500ns	Memory block 1 to cache slot 1
15 hits	17-31	80ns×15=1200ns	
9 hits	15	80ns×9=720ns	Last nine iterations of loop
144 hits	16-31	80ns×144=12,240ns	Last nine iterations of loop
Total hits = 213	Total misses = 5		

Cache Performance Example

- Hit Ratio: 213 / 218 = 97.7%
- Effective Access Time: ((213)*(80ns)+(5)(2500ns)) / 218 = 136 ns

- Although the hit ratio is high, the effective access time in this example is 75% longer than the cache access time due to the large amount of time spent during a cache miss
- What sequence of main memory block accesses would result in much worse performance?

Cache Performance Example

- Consider Cache and Main Memory hierarchy
- Cache targeted to maintain a hit ratio of 0.9.
- A cache access on read-hit takes 20 ns; that on a write-hit takes 60 ns with a write-back scheme, and with a write-through scheme it needs 400 ns.
- The probability of a cache block is to be replaced i.e. dirty is estimated as 0.1.
- An average block transfer time between the cache and shared memory via the bus is 400 ns.
- Consider the read and write accesses are assumed equally probable.
- Derive the effective memory-access times per instruction for the write-through and write-back caches separately.