21 Cache Organization
CSC 230

Department of Computer Science
University of Victoria

M&H: 7.5, insert (p.284) on Intel vs Motorola
Stallings: 4.2, 4.3, 4.5

Cache storage : locality of reference

Cache: a safe place for hiding or storing things (Webster's)

90% of execution time spent on 10% of code

Temporal locality	when the CPU accesses a piece of information, there is a high probability it will be accessed *again soon in time*
Spatial locality	when the CPU accesses a piece of information, there is a high probability that the data *in nearby locations* will be accessed as well

Cache : locality of reference

➔ when data is accessed 1st time, put in cache, ready for next time

↓

Temporal locality

➔ when data is accessed, transfer an adjacent block to cache (and / or use a large block size)

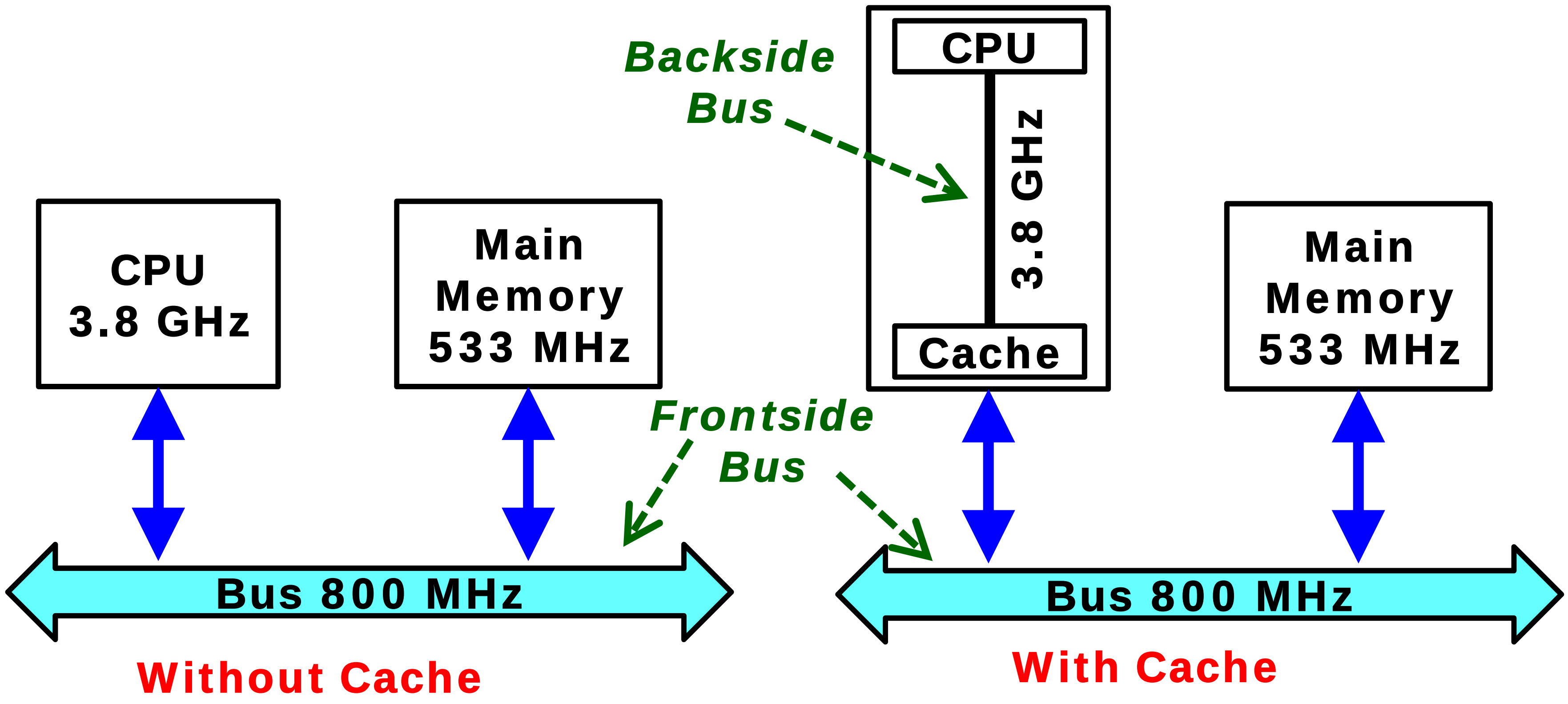
↓

Spatial locality

❑ Put ACTIVE segments of program in fast cache memory

❑ Cache operations are supported by hardware control

Cache : placement and speed



The cache consists of a number of cache lines {slots/blocks}

Each cache line consists of two parts

- a block of data

  (copied from main memory)

- an address tag which specifies the starting memory location of the cache entry

cache hit: when data is found in the cache

cache miss: when data is not found in the cache (next go to memory)

cache full: need to decide how to create space ➔replacement

A few more definitions about cache

Miss rate – the fraction of memory accesses not found in a level of the memory hierarchy (e.g. L1 cache miss because data is in RAM)

Hit time – the time required to access a level of the memory hierarchy, including the time needed to determine whether the access is a hit or a miss

Miss penalty – the time required to fetch a block into a level of the memory hierarchy from the lower level, including:

- the time to access the block,
- transmit it from one level to the other,
- insert it in the level that experienced the miss

How the Cache and the CPU operate together

1. CPU requests contents of memory location
2. Cache is checked first for this data (*hardware-supported search*)
3.

   [3A] If present, get data from cache: HIT (fast)

   [3B.1] If not present, MISS, issue a READ for the required block *from main memory to cache*

   [3B.2] Then copy from cache to CPU
4. Cache includes tags to identify which block of main memory is in each cache slot

Cache design issues

- ✓Size
- ✓Mapping Function

In detail here

- ✓Replacement Algorithm
- ✓Write Policy
- ✓Block Size
- ✓Number of Caches

Definitions only in this course

Write Policy

- ❑ **Write-through protocol**
  - + cache and memory are updated at the same time
  - + simple
  - + memory is consistent when there are two CPUs
  - - extra unnecessary writes

- ❑ **Write-back protocol**
  - - use Dirty/Modified bit to flag cache
  - - can still write too much as it will update whole block
  - - complications if two CPUs access same block
  - + update memory later when cache block is removed

Cache Replacement Policies

Possible algorithms

- **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**

  replace block which has had fewest hits
- **Random**
- **Instructions available to programmers?**

Sizes

Example:

Cache of 64K Byte

Cache block/line of 4 bytes

i.e. cache is 16K (2^{14}) lines of 4 bytes

16 Mb main memory

24 bit address (2^{24}=16M)

Example: a home laptop

Cache of 512K Byte

Cache blocks/lines of 16 bytes

768 Mb main memory

Cache Mapping Strategies: preview summary

Direct

- a memory block always goes to the same cache line
- memory block *j* goes to cache block *(j* mod *size)*
- needs tagging

Associative

- a memory block can go to any cache line
- needs tagging

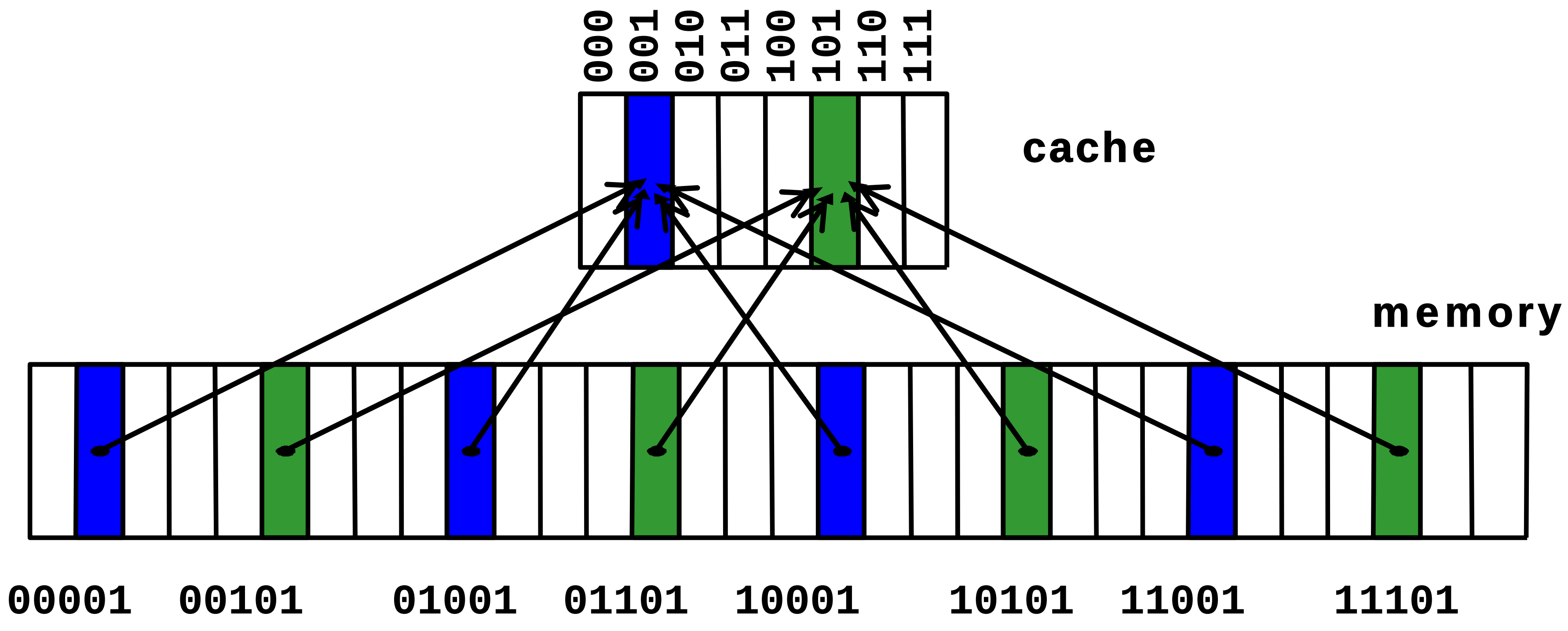
Set-Associative

- cache is divided into a number of sets
- each set contains a number of lines
- block *j* goes to set *(j* mod *#sets)*
  then to *any* line in that set
- needs tagging

[1]: DIRECT-MAPPED CACHE

- Each memory location is mapped to exactly one cache location using a mapping function
- ***Mapping function:***
  (Block address)* modulo *(Number of cache blocks)

- There is only one choice of what to replace



DIRECT-MAPPED CACHE

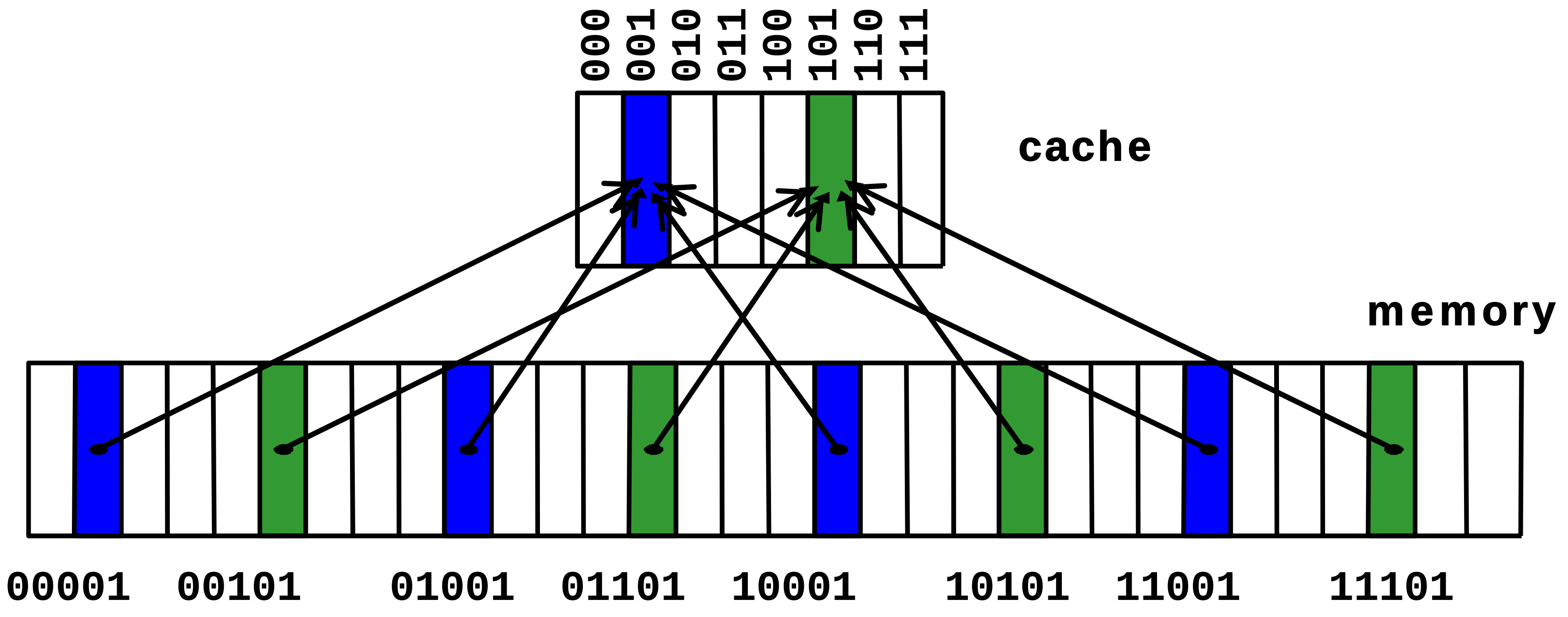
Mapping:

(Block address) modulo (Number of cache blocks)

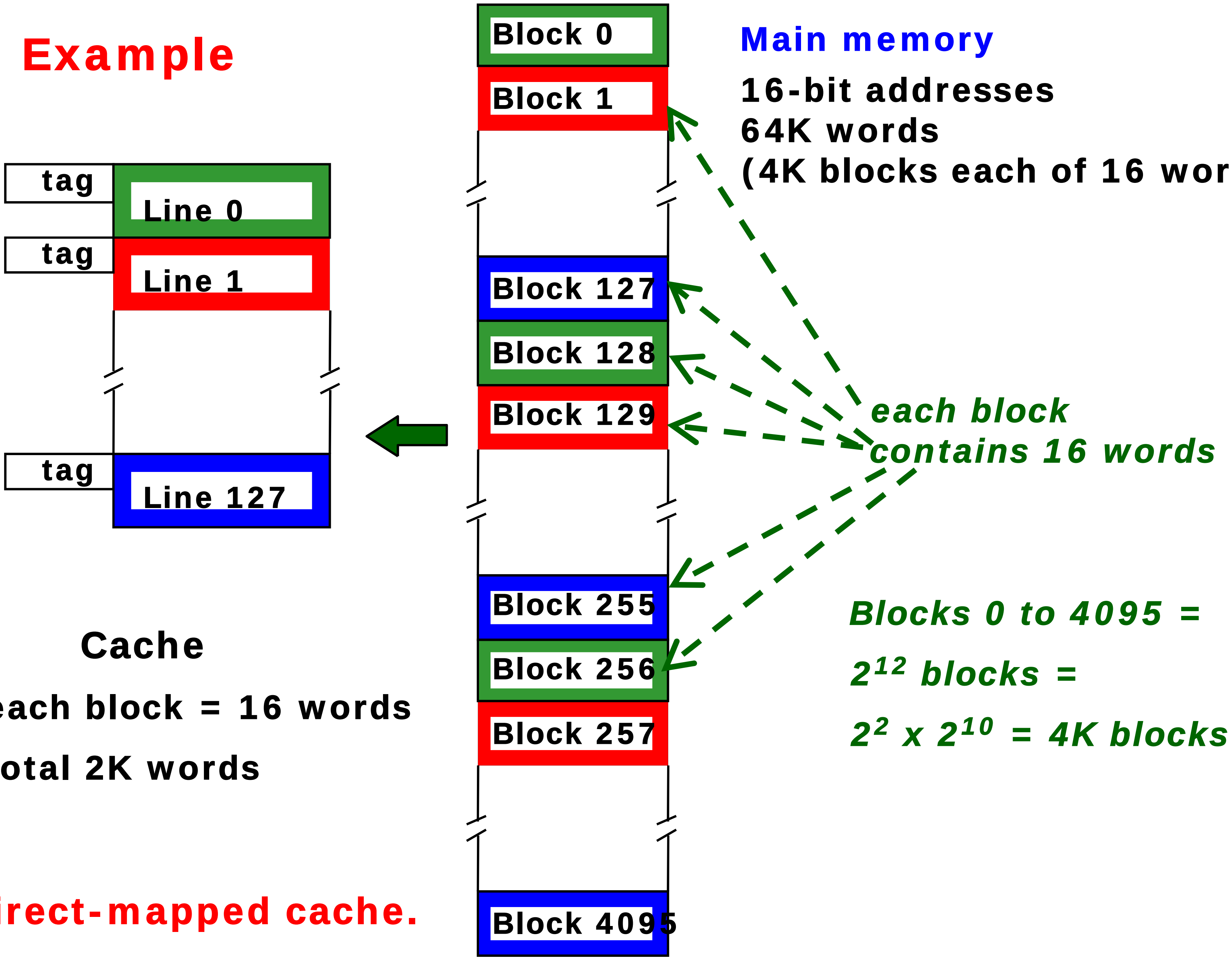
Example below: *(address)* **MOD** *(8)*

e.g. (01101) **MOD** *(8)*

- **If the number of entries in the cache is a power of two, then modulo is given by the low order bits of the address ➔ where the number of bits = $\log_2$ (cache size)**



Example



Main memory

16-bit addresses

64K words

(4K blocks each of 16 words)

Blocks 0 to 4095 =

2^{12} blocks =

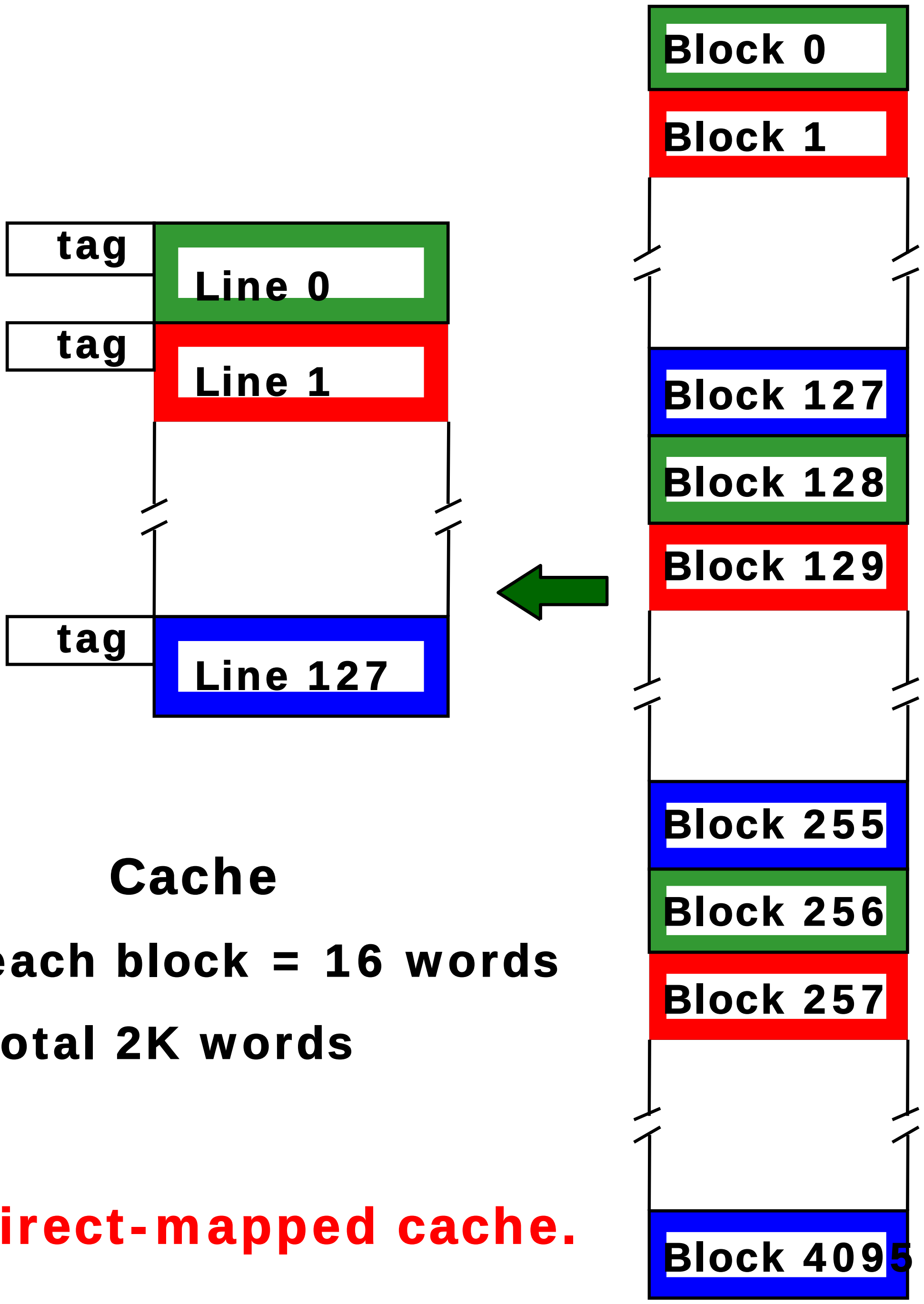
2^2 x 2^{10} = 4K blocks

Cache

each block = 16 words

total 2K words

Direct-mapped cache.



Main memory

16-bit addresses
64K words
(4K blocks each of 16 words)

Block J → *line (J mod 128)*

block number in memory

line number in cache

e.g. 212 → 84

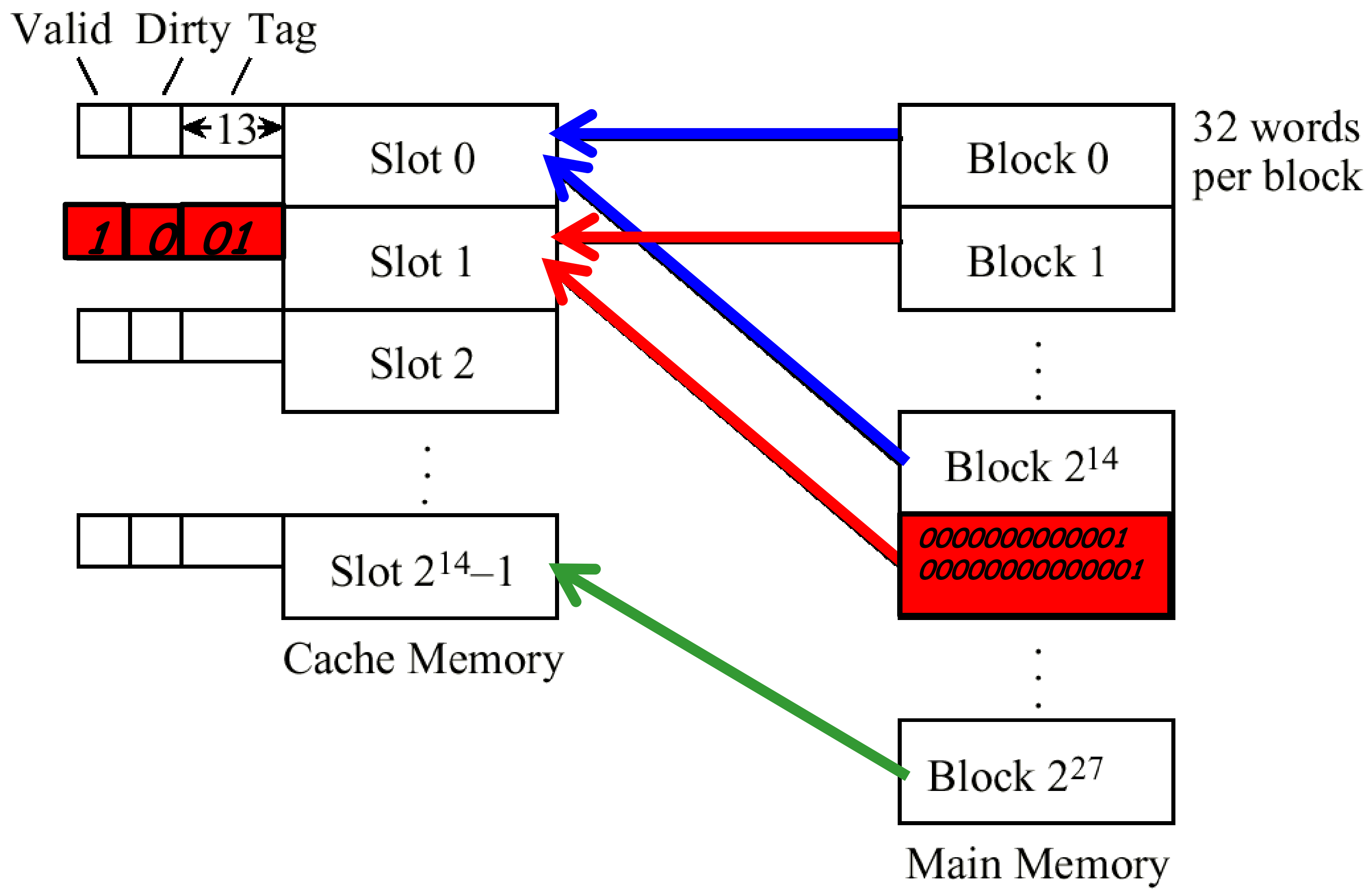
Cache

each block = 16 words

total 2K words

Direct-mapped cache.

DIRECT-MAPPED CACHE (from textbook) – read example

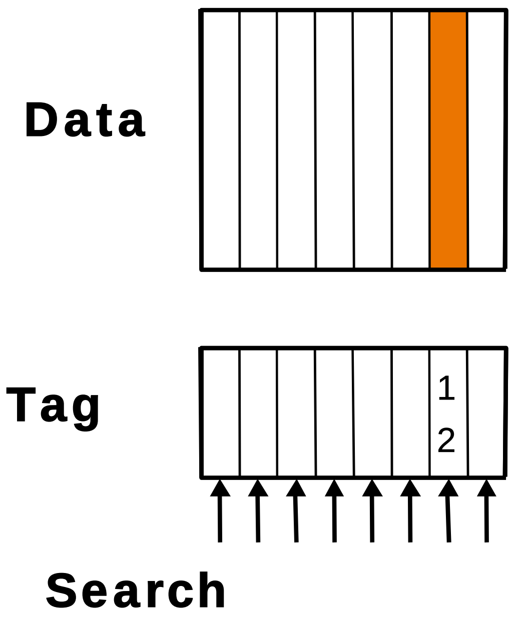


[2]: ASSOCIATIVE CACHE

***Fully associative*:** A block is placed in *any* location in cache
No cache index is needed
Reduces miss ratio

Searching:

- To find a given block, *all the entries* in the cache must be searched
- Search is done *in parallel in hardware*
- Practical only for caches with small number of blocks (else hardware too expensive)



➔ ***Most flexible and most expensive***

[3]: SET-ASSOCIATIVE CACHE

Combination of direct-mapped and fully associative

- ❑ **Blocks of the cache are grouped into sets**

A. Mapping of a block from memory to a specific set
***B. and then* from within that set into any position**

associative

direct-mapping

Given a memory block J:

1. The set is given by:
(*block number J*) modulo (*Number of sets in the cache*)

2. Then any line number within that set

SET-ASSOCIATIVE CACHE: notes

- *A Direct-mapped cache* is simply a one-way set-associative cache

  In direct-mapped, each cache entry holds 1 line and forms 1 set with 1 element

- A *fully associative* cache with *m* entries is simply an *m-way* set-associative cache

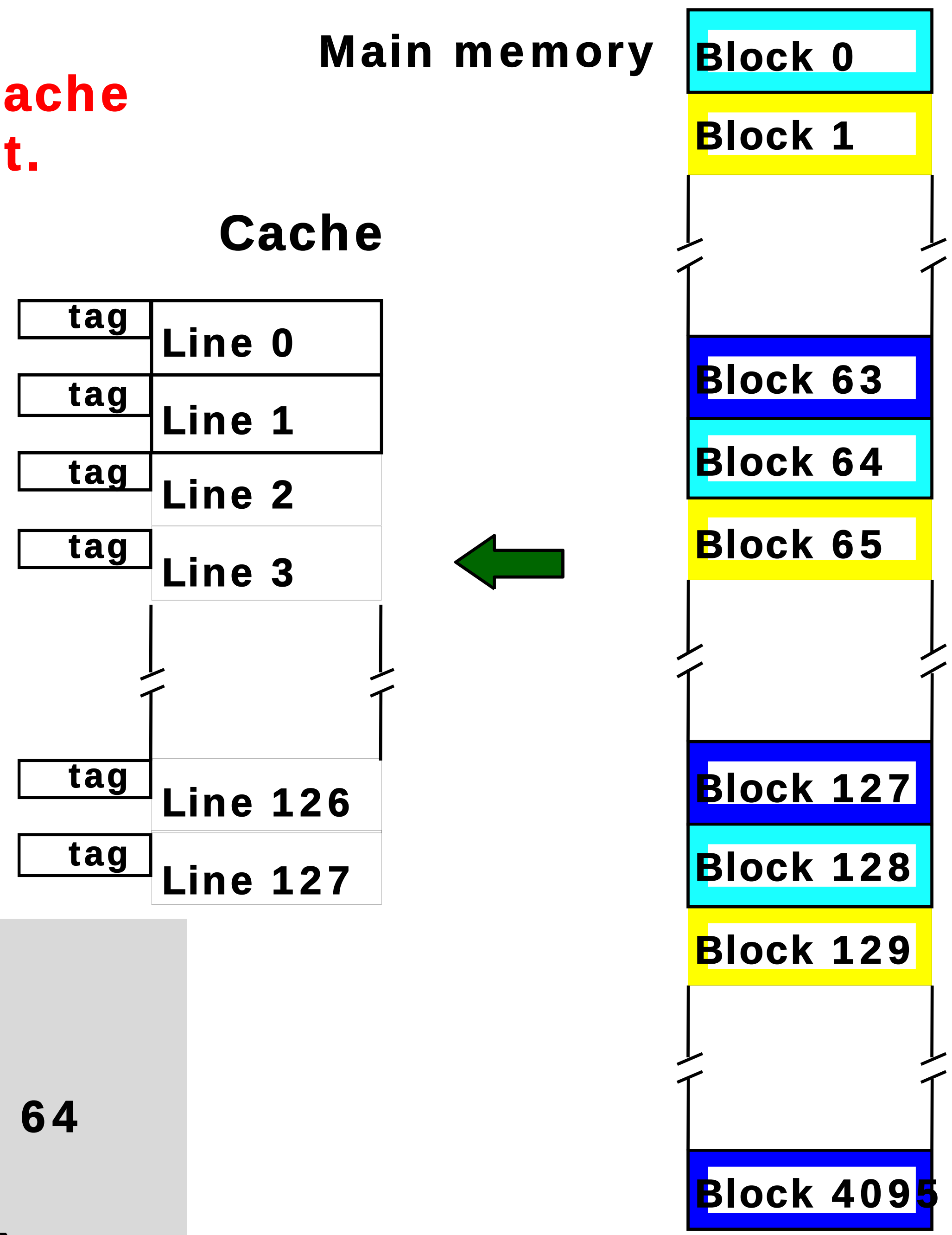
  It has *one* set with *m* lines

In *set-associative*, an entry can reside in any line within a set

Example. Set-associative-mapped cache with 2 lines per set.

Main memory (same as before)

16-bit addresses
64K words
(4K lines each of 16 words)



Cache

each line = 16 words

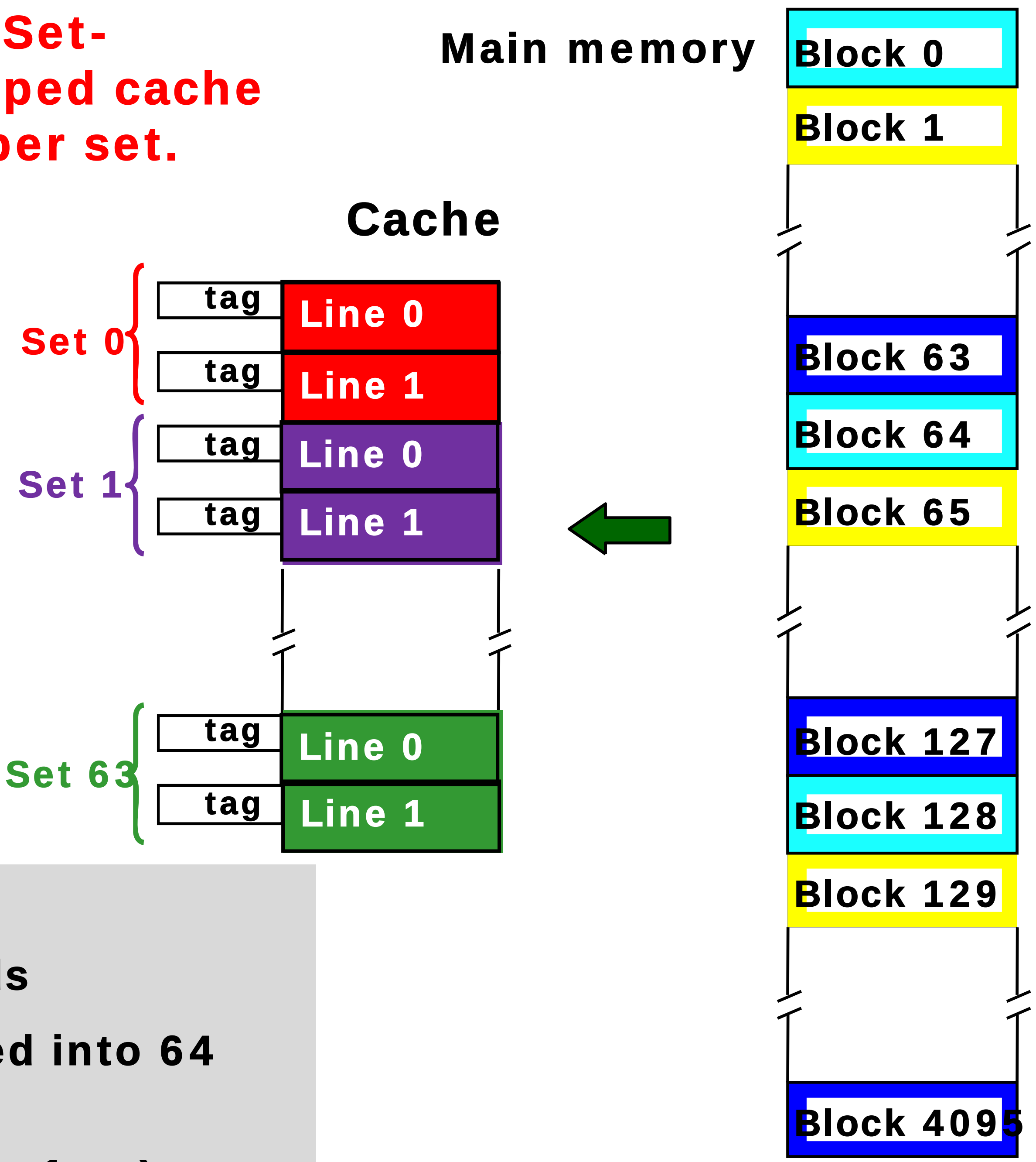
128 blocks organized into 64 sets of 2 lines each

(total 2K words as before)

Example. Set-associative-mapped cache with 2 lines per set.

Main memory (same as before)

16-bit addresses
64K words
(4K lines each of 16 words)



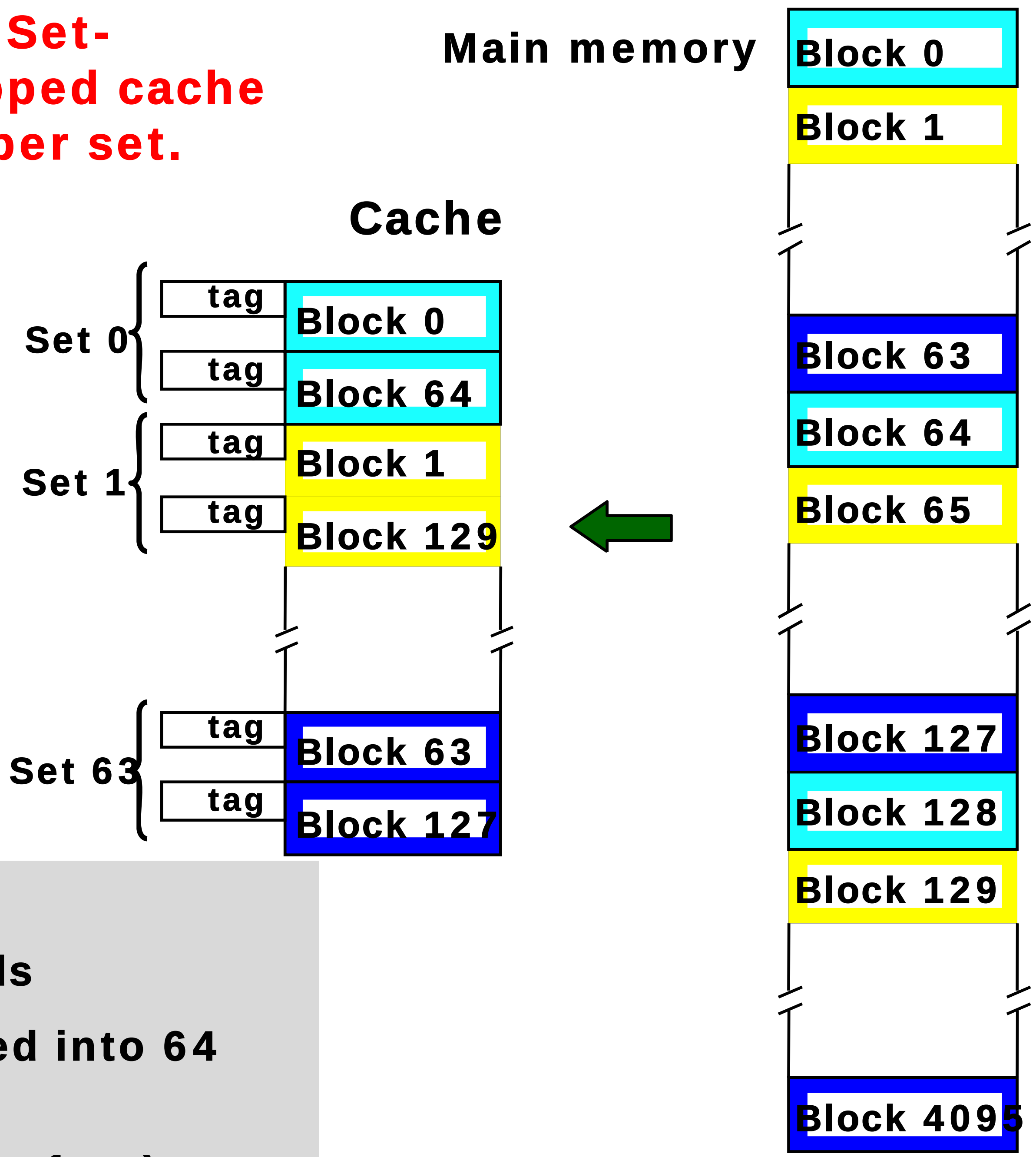
Cache

each line = 16 words

128 blocks organized into 64 sets of 2 lines each

(total 2K words as before)

Example. Set-associative-mapped cache with 2 lines per set.



Main memory
(same as before)
16-bit addresses
64K words
(4K lines each
of 16 words)

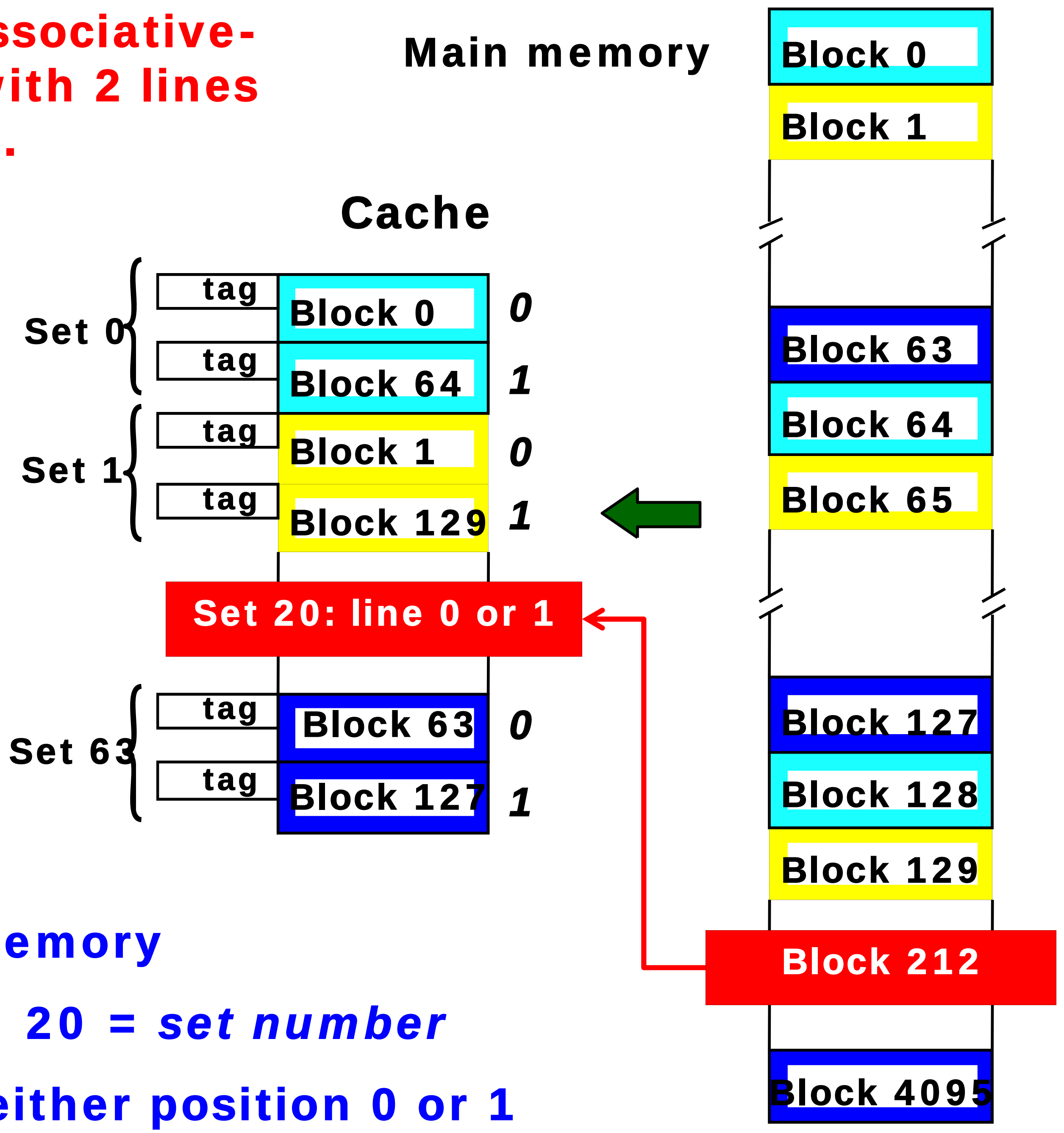
Cache

each line = 16 words

128 blocks organized into 64 sets of 2 lines each

(total 2K words as before)

Example. Set-associative-mapped cache with 2 lines per set.



Example:

Block 212 from memory

➔(212) mod 64 = 20 = *set number*

➔ within set 20, either position 0 or 1

To find a block in a set-associative cache

Decompose the address: index + tag + offset

- Index is used to select the set
- Tag is used to choose the block by parallel comparison within the selected set
- Block offset is the address of the desired data within a block

Advantages:

- Decreases the miss rate as it reduces the misses that compete for the same location

Disadvantages:

- Requires *parallel* comparators
- Requires more tag bits per cache block
- Extra delay for data, increasing hit time

➔ *Do not need to become competent at computing tags/index/offset here*

MULTI-LEVEL CACHE

Often primary cache is on the same chip as the processor

Add 2nd SRAMs as second cache (on chip or on board)

First cache closes the speed gap between the CPU and memory

Multilevel cache reduces the miss penalty

- Miss penalty goes down if data is in 2nd level cache
- Primary cache can be smaller and have a higher miss rate

If the 2nd level cache contains the desired data, the miss penalty = access time of the 2nd level cache

If the 2nd level cache does not contain the data, a main memory access is required, ***a larger miss penalty*** is incurred

➔ Primary cache is often 10 or more times faster than secondary

Comparing single and multi-level cache

Primary cache of a multilevel cache:

- often smaller
- often uses a smaller block size
- tries to optimize the hit time for a shorter clock cycle
- many recent systems use *direct-mapped primary caches* because of their advantage in access time and simplicity

Secondary cache:

- usually larger
- access time less critical
- often uses larger block size
- Tries to optimize the miss rate to reduce the penalty of long memory access time

On-chip caches have associativity of two to four, while off-chip caches rarely have associativity greater than two

Another Design Concept: Split cache

Two independent caches which operate in parallel

➔ **instruction cache (I cache)**

➔ **data cache (D cache)**

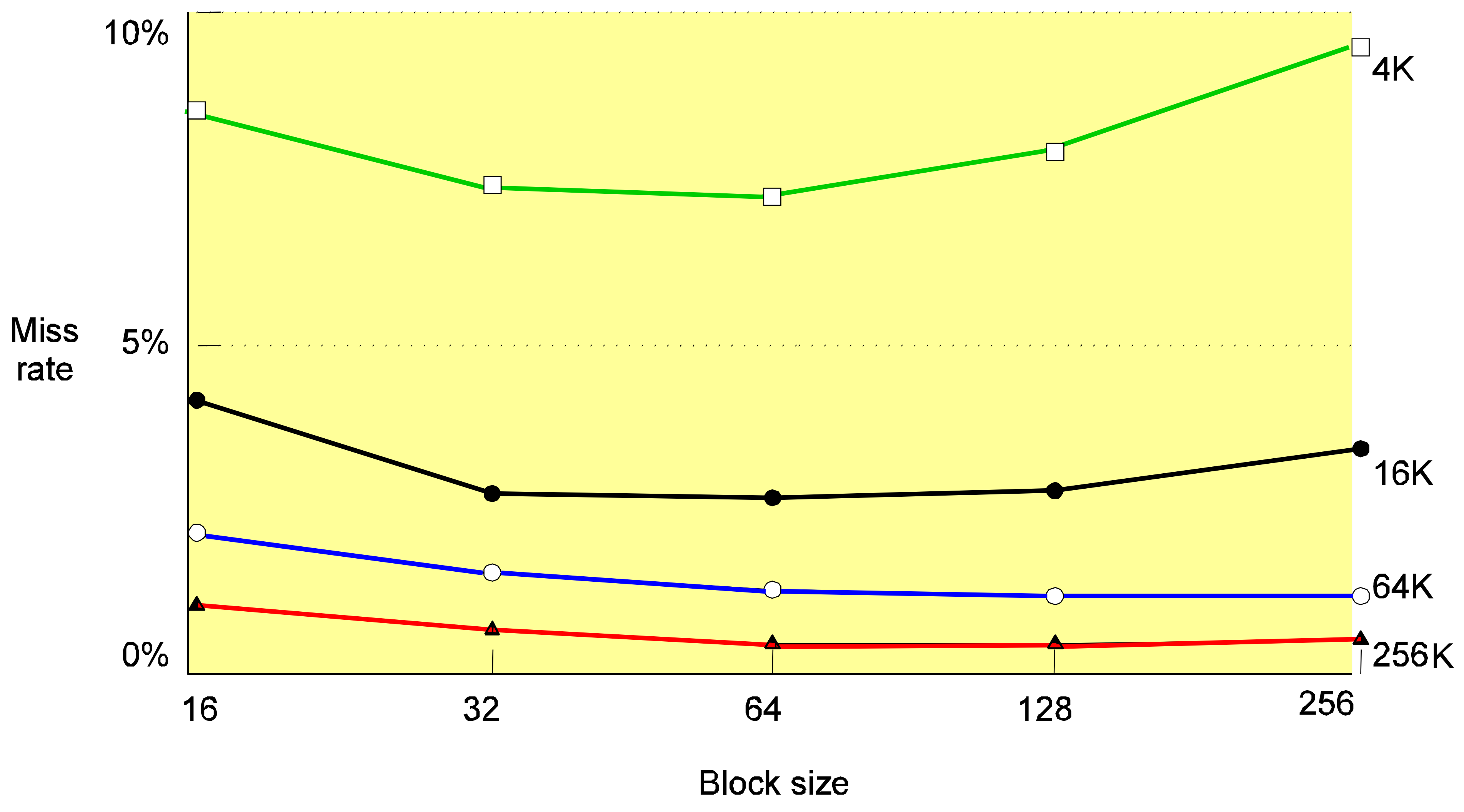
- ❑ **To minimize collisions, separate fetches of data and instructions**
  - **Then instructions only have collisions with instructions, data with data**
  - **The two have different behaviour patterns**
  - **Goal is to decrease the miss ratio from a combined cache**
- ❑ **The miss ratio does not necessarily improve!**

(do not confuse split and multi-level cache)

Instruction cache miss steps

1. **Send original address to memory**
2. **Send “READ” command to memory and wait for access cycle**
3. **Write the cache entry, plus tag and V bit**
4. ***RESTART*** **instruction cycle from fetch**

➔ it will cause a cache hit

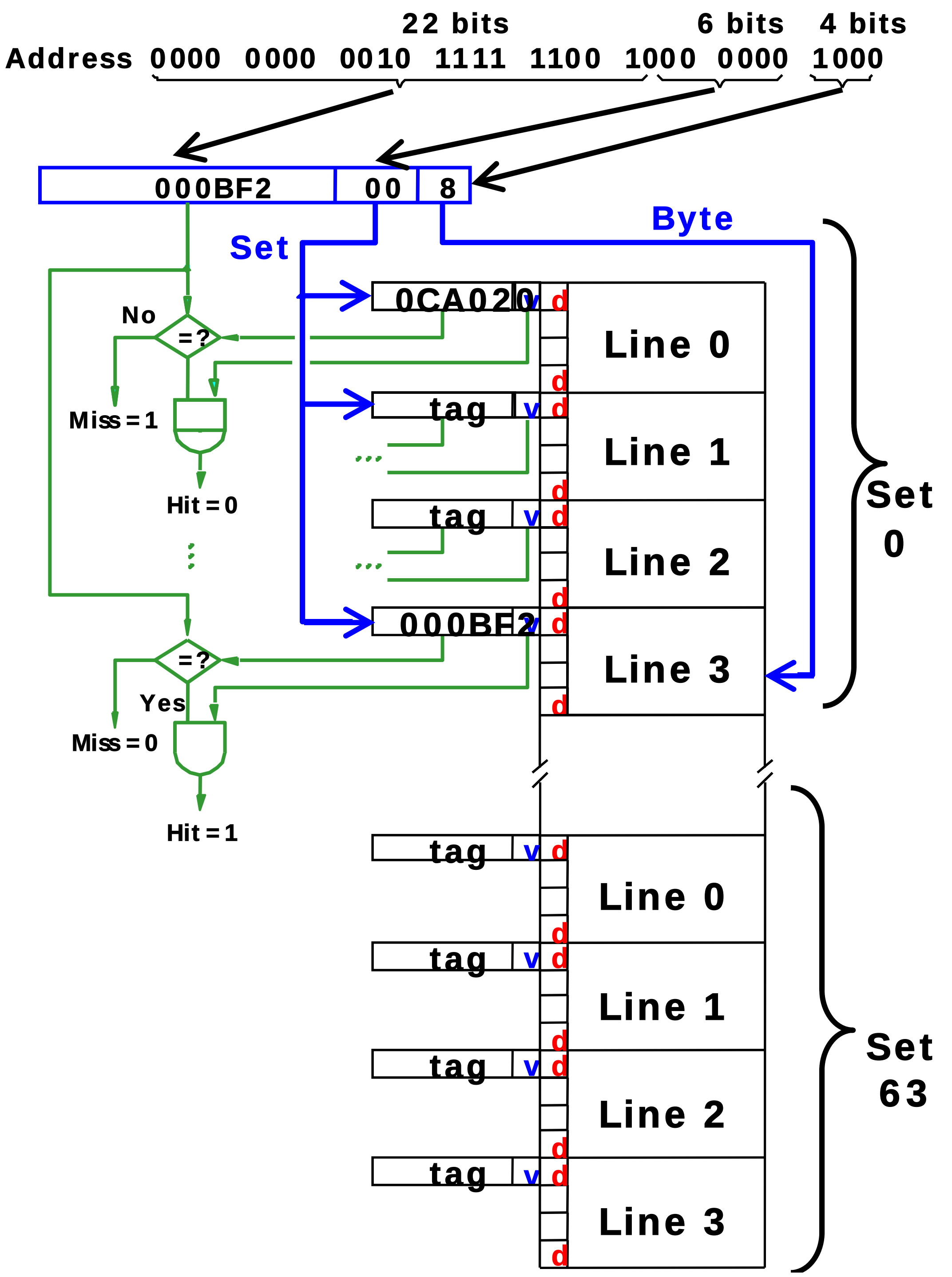


Miss rate goes UP if block size is too large relative to cache size

CACHE DESIGN: what the designer must consider

- ❑ Cache size
- ❑ Block size
- ❑ Associativity
- ❑ Replacement policy
- ❑ Write-hit policy (write-through, write-back)

Design change	Effect on miss rate	Possible negative performance effect
Increase size	decrease misses	may increase access time
Increase associativity	decreases miss rate due to conflict misses	may increase access time
Increase block size	decreases miss rate for a wide range of block sizes	may increase miss penalty



Data cache organization in 68040 microprocessor.

Two caches:

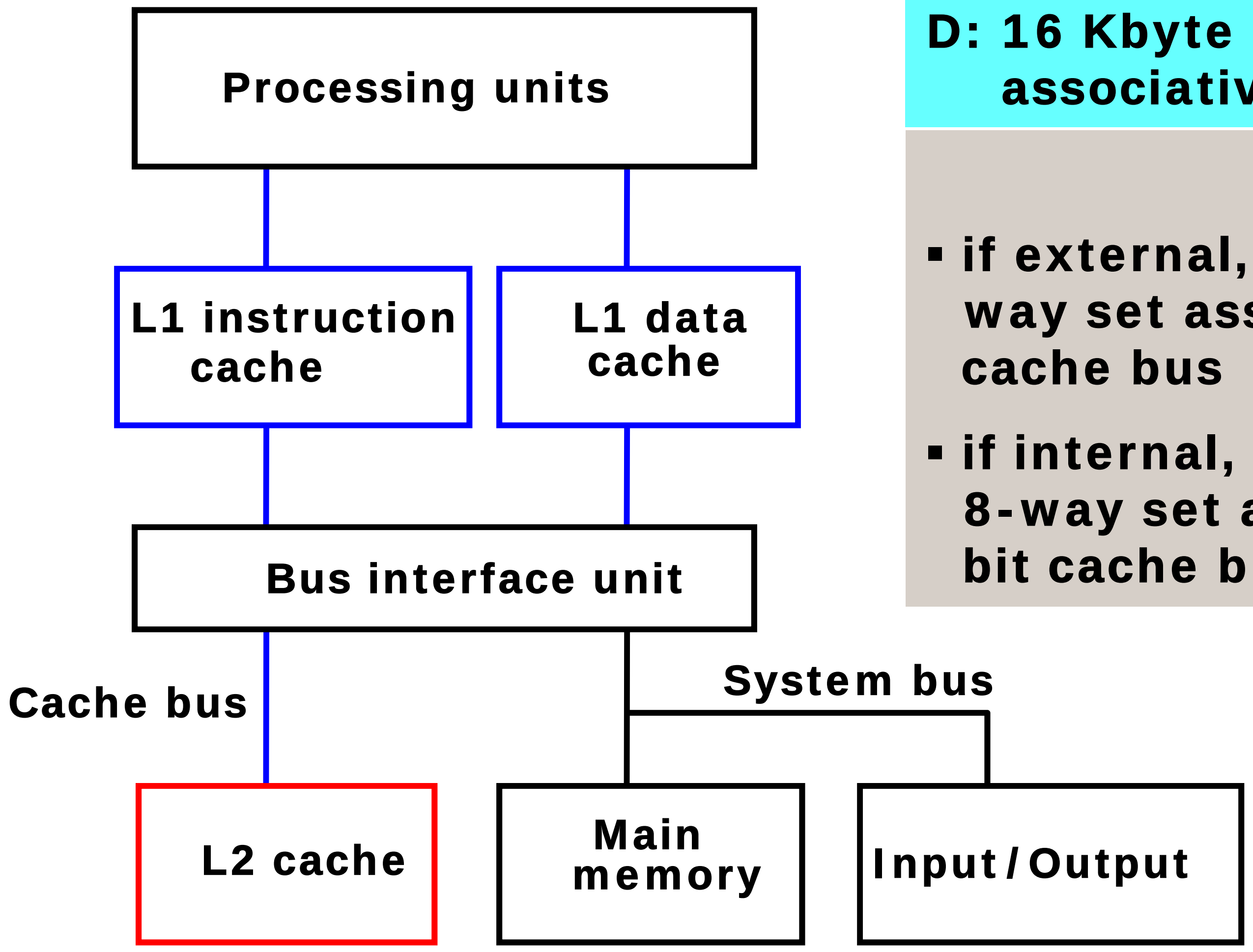
- instructions and data, each of 4K bytes
- 4-way set associative
- 64 sets
- each set holds 4 lines
- each line holds 4 32-bit words

ARM Cache

- One cache only
- similar organization to 68040
- 4-way set associative
- 64 sets
- each set with 4 lines
- each block has 16 bytes = 4 32-bit words

- write-through protocol and random replacement

  ➔ simplicity



L1

I: 16 Kbyte instruction 2-way set associative

D: 16 Kbyte data, 4-way set associative

L2

- if external, 512 Kbyte, 4-way set associative, 64 bit cache bus
- if internal, 256 Kbyte data, 8-way set associative, 256 bit cache bus

Caches and external connections in Pentium III processor

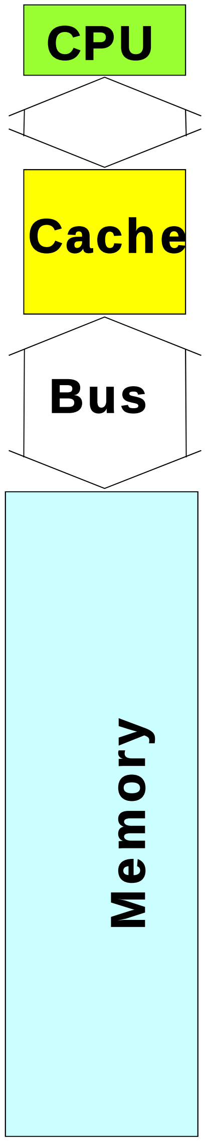
Examples

Time to transfer a block of data from memory to cache after miss

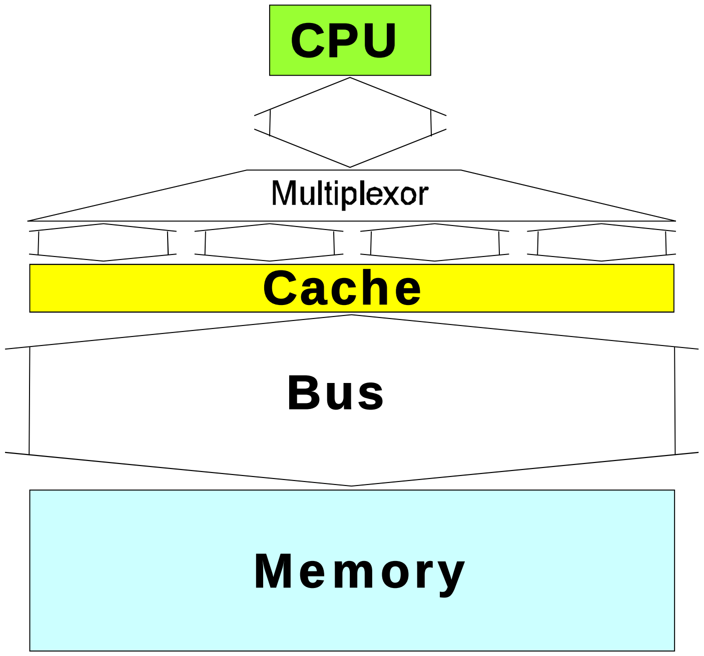
- cache with 8-word blocks
- 1 clock cycle to send address to memory
- in DRAM, 1st word accessed in 8 cycles, following words in 4 cycles
- 1 clock cycle to send 1 word data back to cache

EX 1: Single memory module

$$\underset{\substack{\uparrow \\ \textit{send address}}}{1} + \underset{\substack{\uparrow \\ \textit{Word 1}}}{8} + \underset{\substack{\uparrow \\ \textit{Next 7 words}}}{(7\ \mathrm{x}4)} + \underset{\substack{\uparrow \\ \textit{Write to cache}}}{1} = 38 \text{ cycles}$$

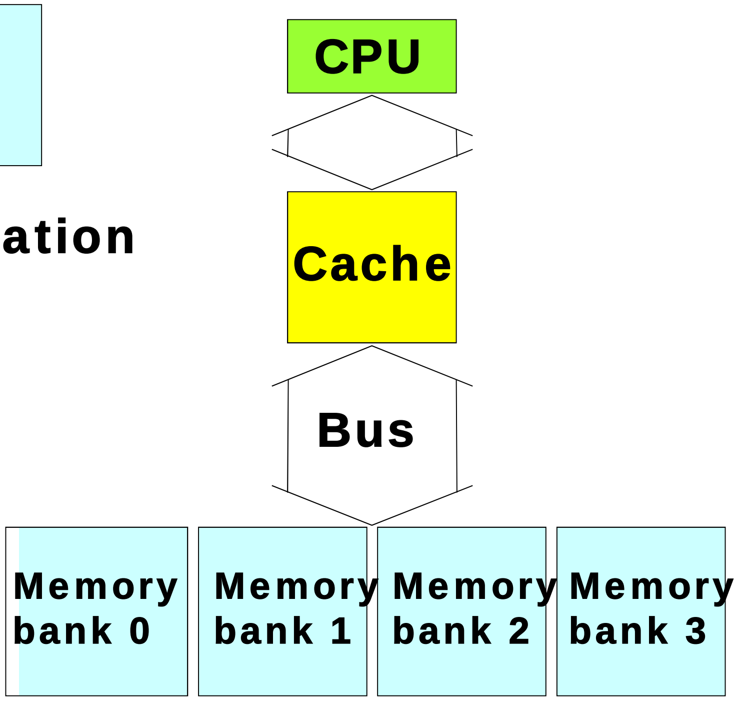


a. One-word wide memory organization



b. Wide memory organization

Read only and be aware!



c. Interleaved memory organization

Examples

Time to transfer a block of data from memory to cache after miss

- cache with 8-word blocks
- 1 clock cycle to send address to memory
- in DRAM, 1st word accessed in 8 cycles, following words in 4 cycles
- 1 clock cycle to send 1 word data back to cache

EX 1: Single memory module

1 + 8 + (7 x4) + 1 = 38 cycles

EX 2: Four interleaved memory modules

1 + 8 + 4 + 4 = 17 cycles

first 4 words accessed in parallel

first 4 words to cache in parallel

plus 4 more words accessed in parallel

last 4 words to cache in parallel

[1] Consider memory access time with different memory widths: one-word wide

Read only and be aware!

If cache block = 4 words,

DRAM is 1 word wide,

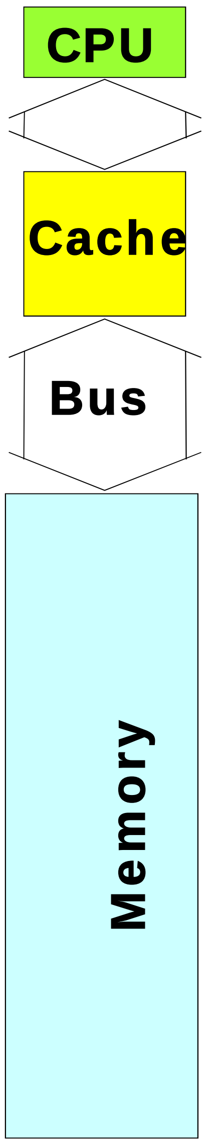
➔ then miss penalty:

1 + (4 x 15) + (4 x 1) = 65 memory bus clock cycles

Bytes transferred per bus clock cycle for a miss is:

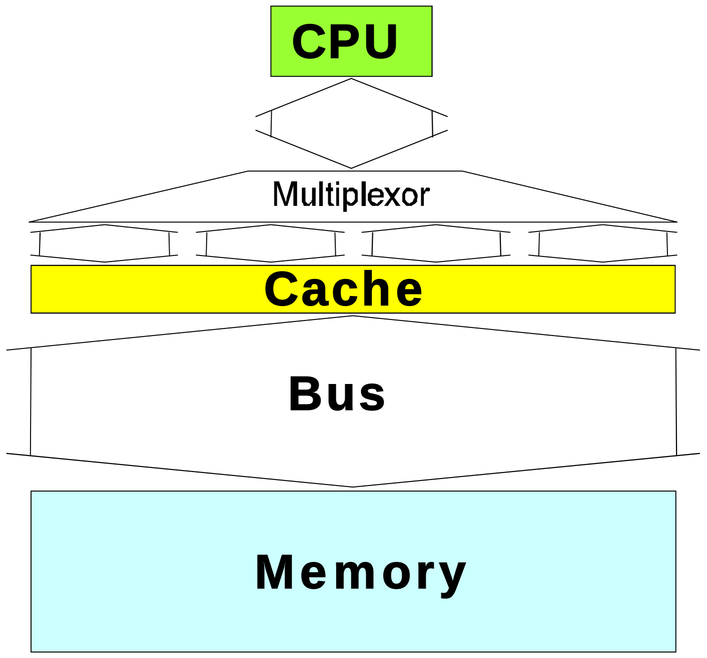
(4 x 4) / 65 = 0.25

➔ effective bandwidth



a. One-word wide memory organization

[2] Consider memory access time with different memory widths: 2-word wide



b. Wide memory organization

If cache block = 4 words,

DRAM is 2 words wide,

➔ then miss penalty:

1 + (2 x 15) + (2 x 1) = 33 memory bus clock cycles

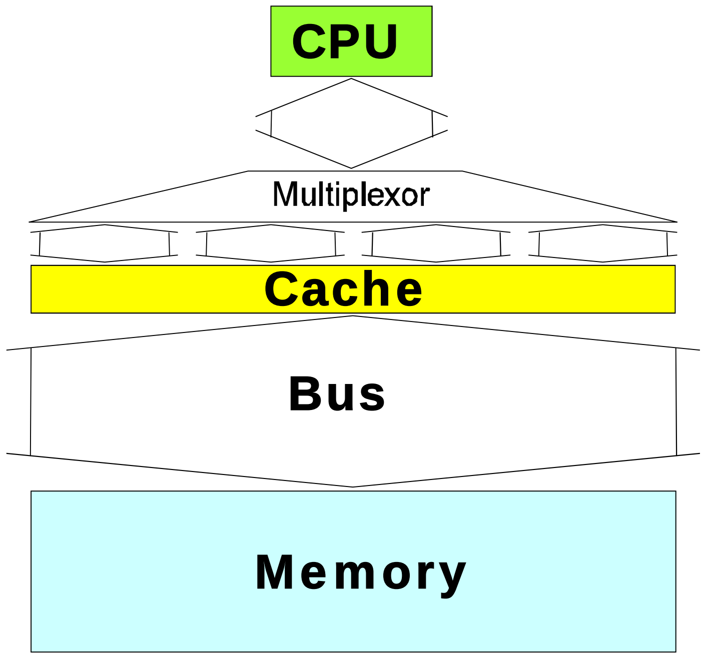
Read only and be aware!

Bytes transferred per bus clock cycle for a miss is:

(4 x 4) / 33 = 0.48

➔ effective bandwidth

[3] Consider memory access time with different memory widths: 4-word wide



b. Wide memory organization

If DRAM is 4 words wide,

➔ then miss penalty:

1 + (1 x 15) + (1 x 1) = 17 memory bus clock cycles

Bytes transferred per bus clock cycle for a miss is:

(4 x 4) / 17 = 0.94

➔ effective bandwidth

Read only and be aware!

In general, average access time from CPU

$$T\ (\text{average}) = \underbrace{h}_{\textit{hit rate}} \times \underbrace{C}_{\textit{cache access time}} + \underbrace{(1 - h)}_{\textit{miss = no hit}} \times \underbrace{M}_{\textit{miss penalty}}$$

Look at another example (follow up)

$$T(\text{ave}) = h \times C + (1 - h) \times M$$

- 10 cycles for DRAM access, no cache
- cache with 8-word blocks, and interleaved memory
- 30% of all instructions need a Read or Write

  ➔ i.e. 130 memory accesses every 100 instructions executed (fetch!)

GIVEN:

- hit rate in instruction cache = 0.95,
- hit rate in data cache = 0.90
- need 17 cycles to load a block in cache (see earlier example)

Calculate the speedup rate

Calculate the ratio between a "no cache" and a "with cache" system

$$\frac{\text{no cache}}{\text{with cache}} =$$

$$\text{no cache} = \underbrace{130}_{\text{Accesses to memory for 100 instructions + 30 data}} \times \underbrace{10}_{\text{Given time for each Memory access}}$$

Accesses to memory for 100 instructions + 30 data

Given time for each Memory access

$$\text{with cache} = 100\ \boxed{(0.95 \times 1} + \boxed{0.05 \times 17)}$$

Accesses to memory for 100 instructions

$$+\ 30\ \boxed{(0.9 \times 1 +}\ \boxed{0.1 \times 17)}$$

Accesses to memory for 30 data

$$\boxed{\frac{130 \times 10}{100\ (0.95 \times 1 + 0.05 \times 17) + 30\ (0.9 \times 1 + 0.1 \times 17)} = 5.04}$$

$$T\,(\text{ave}) = h \times C + (1 - h) \times M$$

(1) 10 cycles for DRAM access, no cache

(2) cache with 8-word blocks, and interleaved memory

30% instructions need Read or Write

i.e. 130 memory accesses every 100 instructions executed (fetch!)

hit rate in instruction cache = 0.95, and for data = 0.90

need 17 cycles to load block in cache (see earlier)

$$\frac{\text{no cache}}{\text{with cache}} =$$

$$\frac{130 \times 10}{100\,(0.95 \times 1 + 0.05 \times 17) + 30\,(0.9 \times 1 + 0.1 \times 17)} = 5.04$$

Direct Mapped Cache Example

- Compute hit ratio and effective access time for a program that executes from memory locations 48 to 95, and then loops 10 times from 15 to 31.

- The direct mapped cache has four 16-word slots, a hit time of 80 ns, and a miss time of 2500 ns. Load-through is used. The cache is initially empty.

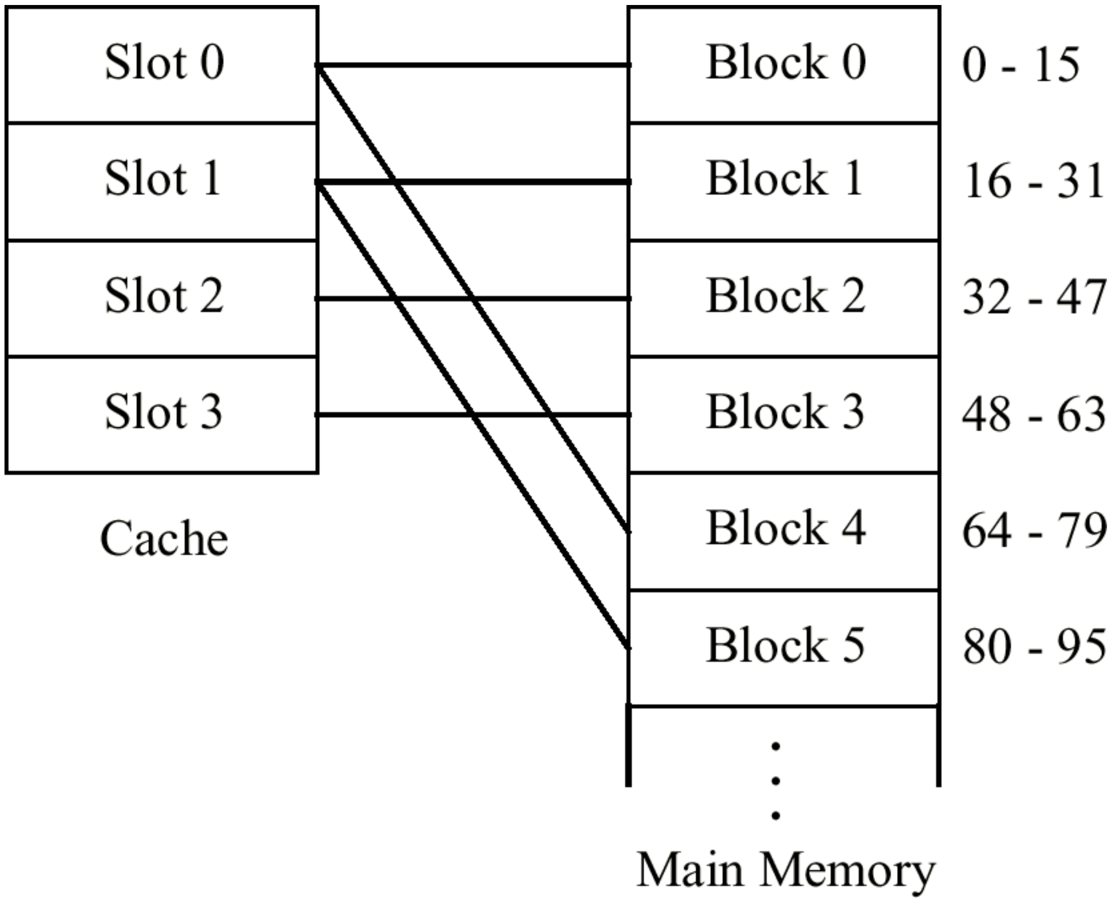
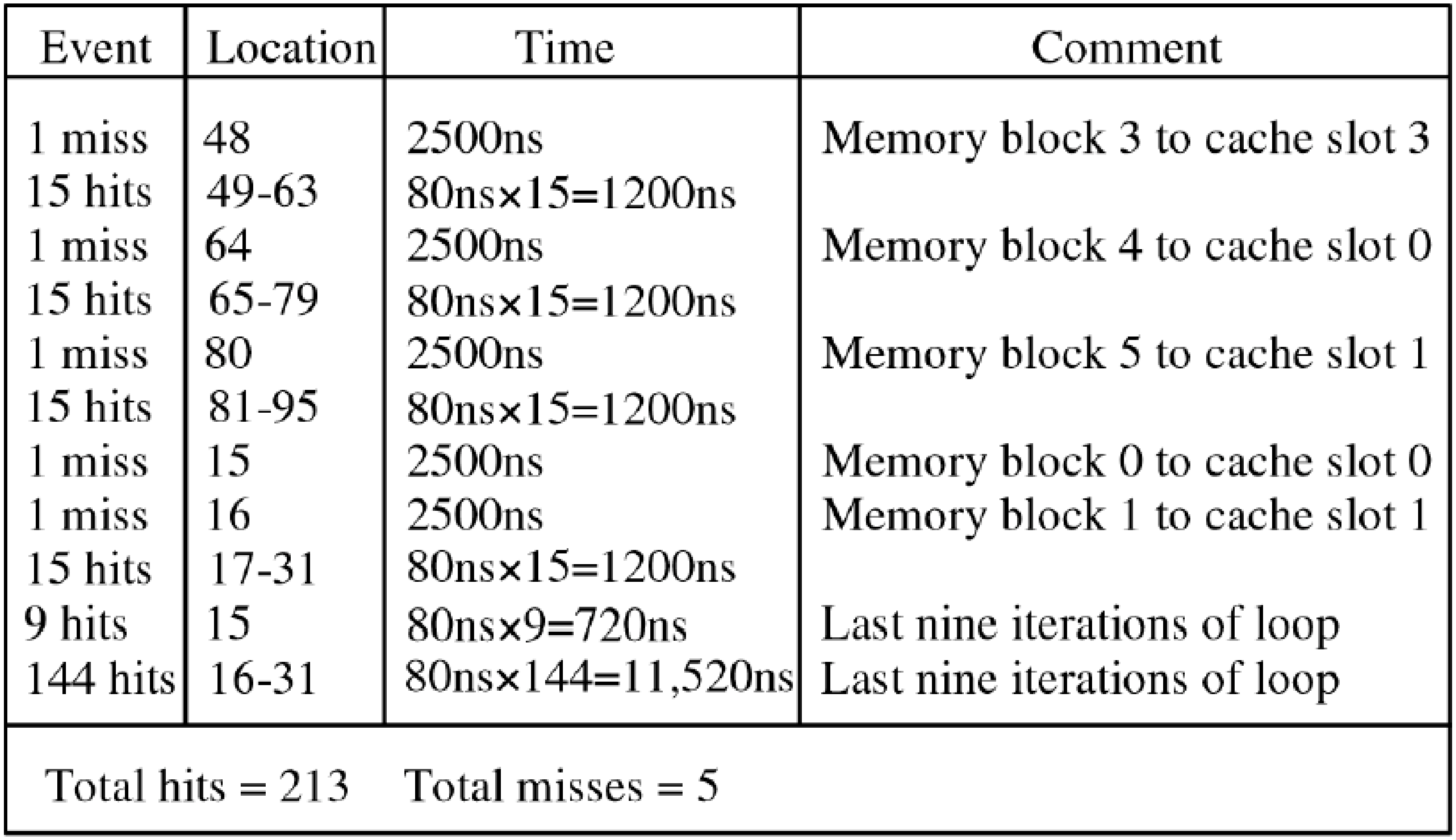


Table of Events for Example Program



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=11,520ns	Last nine iterations of loop

Total hits = 213 Total misses = 5

Calculation of Hit Ratio and Effective Access Time for Example Program

$$Hit\ ratio = \frac{213}{218} = 97.7\%$$

$$EffectiveAccessTime = \frac{(213)(80ns) + (5)(2500ns)}{218} = 136ns$$

Can the hit rate be improved to almost optimal?

- Larger cache ➔ increased costs
- Increase block size, same total cache size ➔ ok only up to certain relative ratio between sizes
- BEST: block sizes not too small, not too large

  usually 16 to 128 bytes

❑ Speed is improved if cache is on processor chip

  ➔ however space on processor is at premium

❑ Split caches ➔ instruction and data

❑ Multi-level ➔ L1, L2 or even L3

❑ Prefetching

❑ Lockup free cache

Multi-level Cache: Example

As an example, consider a two-level cache in which the L1 hit time is 5 ns, the L2 hit time is 20 ns, and the L2 miss time is 100 ns. There are 10,000 memory references of which 10 cause L2 misses and 90 cause L1 misses. Compute the hit ratios of the L1 and L2 caches and the overall effective access time.

H_1 is the ratio of the number of times the accessed word is in the L1 cache to the total number of memory accesses. There are a total of 85 (L1) and 15 (L2) misses, and so:

$$H_1 = \frac{(10{,}000 - 10 - 90)\ hits}{10{,}000\ accesses} = 99\%$$

(Continued on next slide.)

Multi-level Cache: Example (cont.)

H_2 is the ratio of the number of times the accessed word is in the L2 cache to the number of times the L2 cache is accessed, and so:

$$H_2 = \frac{(100 - 10)\ hits}{100\ accesses} = 90\%$$

The effective access time is then:

$$\begin{aligned} T_{EFF} = {} & (10,000 - 10 - 90\ L1\ hits)(5\ ns\ per\ L1\ hit) + \\ & (100 - 10\ L2\ hits)(20\ ns\ per\ L2\ hit) + \\ & (10\ L2\ cache\ misses)(100\ ns\ per\ L2\ cache\ miss) \\ & /10{,}000\ accesses \\ = {} & 5.23\ \text{ns per access} \end{aligned}$$

Designing memory to support cache more efficiently

- ❑ DRAM for main memory designed for density rather than access time
- ❑ Increase bandwidth from memory to cache to reduce miss penalty?
- ❑ Processor connected to memory over bus ➔ the bus is at least 10 times slower than CPU clock cycle

Consider memory access time (example):

- ✓ 1 memory bus clock cycle to send the address
- ✓ 15 memory bus clock cycles for each DRAM access
- ✓ 1 memory bus clock cycle to send a word of data

If cache block = 4 words, and DRAM is 1 word wide, then miss penalty:

1 + (4 x 15) + (4 x 1) = 65 memory bus clock cycles

Bytes transferred per bus clock cycle for a miss is:

(4 x 4) / 65 = 0.25 ➔ bandwidth