MODULE 7: Memory Systems

Lecture 7.2
Cache Memory

Prepared By:

- Scott F. Midkiff, PhD
- Luiz A. DaSilva, PhD
- Kendall E. Giles, PhD

Electrical and Computer Engineering

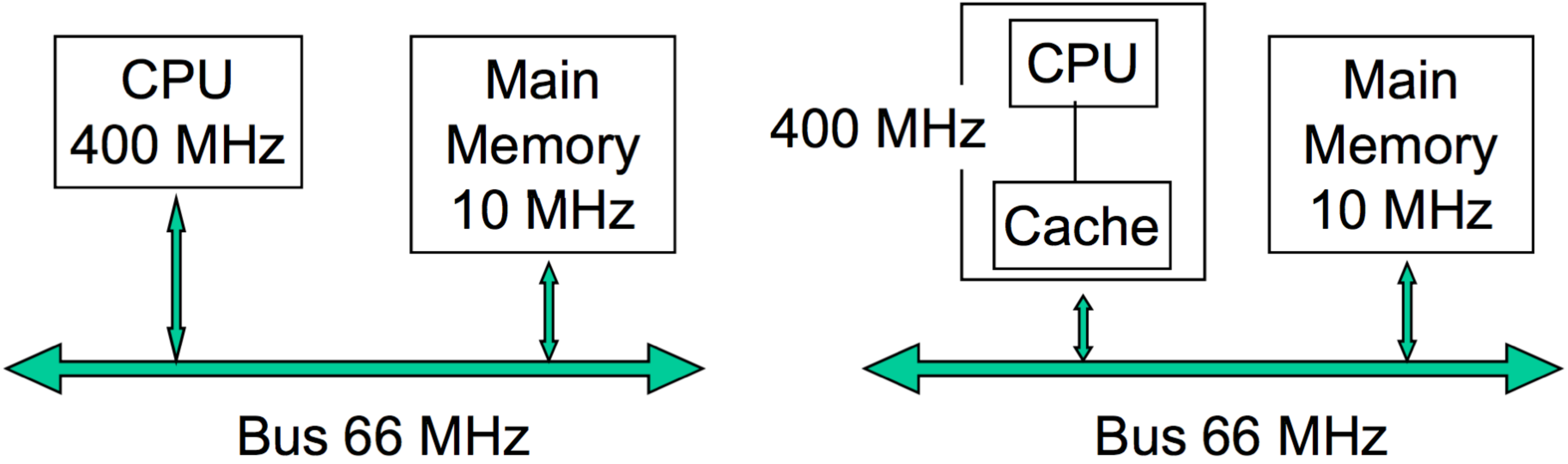
Virginia Tech

Lecture 7.2 Objectives

- Explain the motivation for using cache memory and the concept of locality
- Describe how address translation takes place using associative mapped caches and direct mapped caches
- Calculate hit ratio and effective access time, given a simple sample program
- Discuss cache replacement policies and enhancements to unified cache

Motivation for Cache Memory

- Place a small amount of fast memory between the CPU and main memory
- Take advantage of spatial and temporal locality

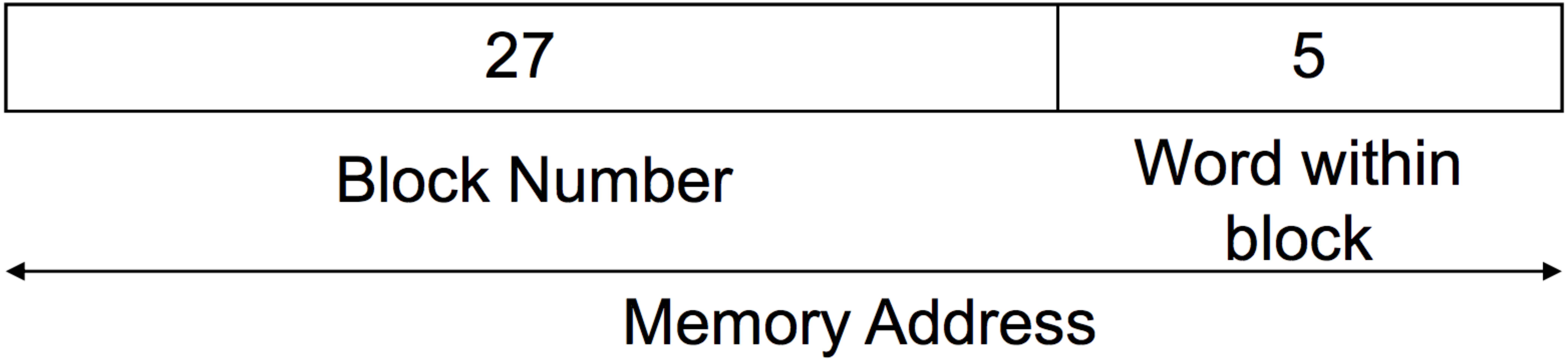


Address Translation

- Must translate between memory addresses and cache addresses
- Application should not need to be aware of the translation
- Different techniques for translation
    - Associative Mapped Caches
    - Direct Mapped Caches

Memory Fields

- Divide memory address into blocks and words within blocks
- In practice, each cache block (cache line) ranges from 8-64 bytes
- In the example below, a 32-bit memory address is partitioned into 2^{27} blocks, each containing $2^5 = 32$ words

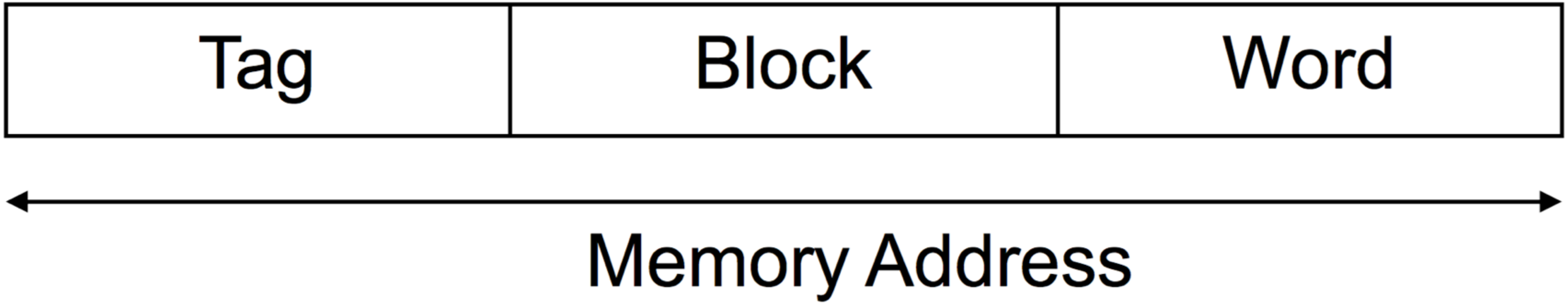


Direct Mapped Cache

- The simplest cache mapping scheme
  - Given memory block can be placed in only one particular place in the cache
- In a direct mapped cache consisting of N blocks of cache, block X of main memory maps to cache block Y = X mod N
  - If we have 10 blocks of cache, block 7 of cache may hold blocks 7, 17, 27, 37, . . . of main memory
- Once a block of memory is copied into its slot in cache, a valid bit is set for the cache block to let the system know that the block contains valid data

Main Memory Address Using Direct Mapped Cache

- Memory address is partitioned into 3 fields
- Given memory block can be placed in only one particular place in the cache
- Block field indicates which of the cache locations the block is to be put in
- Tag field is stored with the block and uniquely identifies this block



CHECK POINT

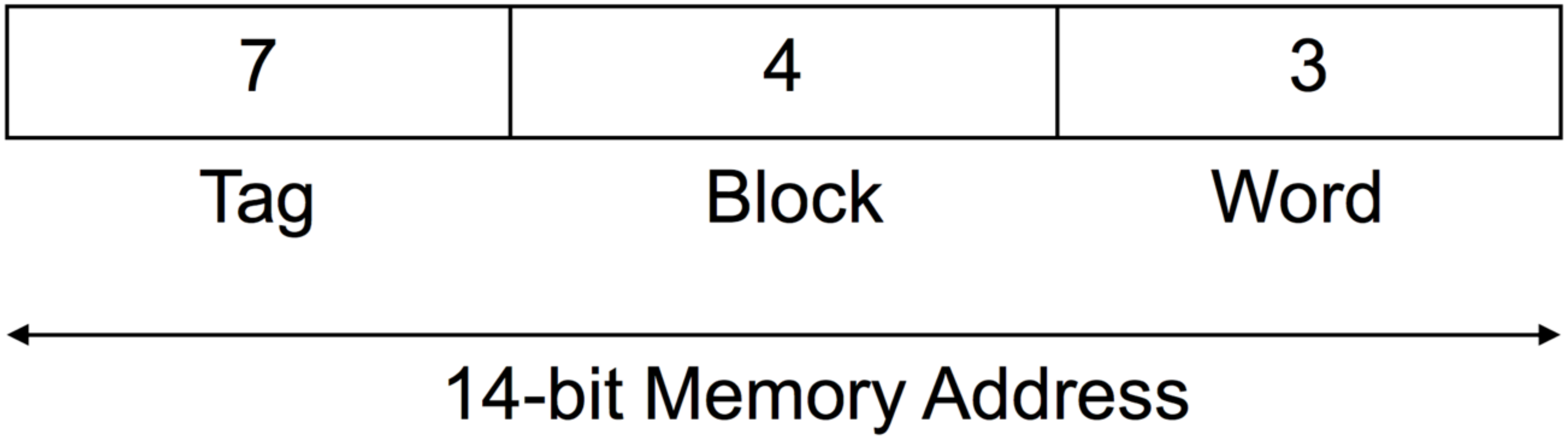
As a checkpoint of your understanding, please pause the video and make sure you can do the following:

- Explain the motivation for using cache memory and the concept of locality

If you have any difficulties, please review the lecture video before continuing.

Direct Mapped Example (1)

- Suppose 14-bit memory addresses and a direct mapped cache that can hold 16 8-word blocks
    - Need 3 bits for word addresses
    - Need 4 bits for block addresses
    - Have 7 bits for tag addresses
- Note that main memory is divided into $2^{14}/2^{3} = 2^{11}$ blocks



Direct Mapped Example (2)

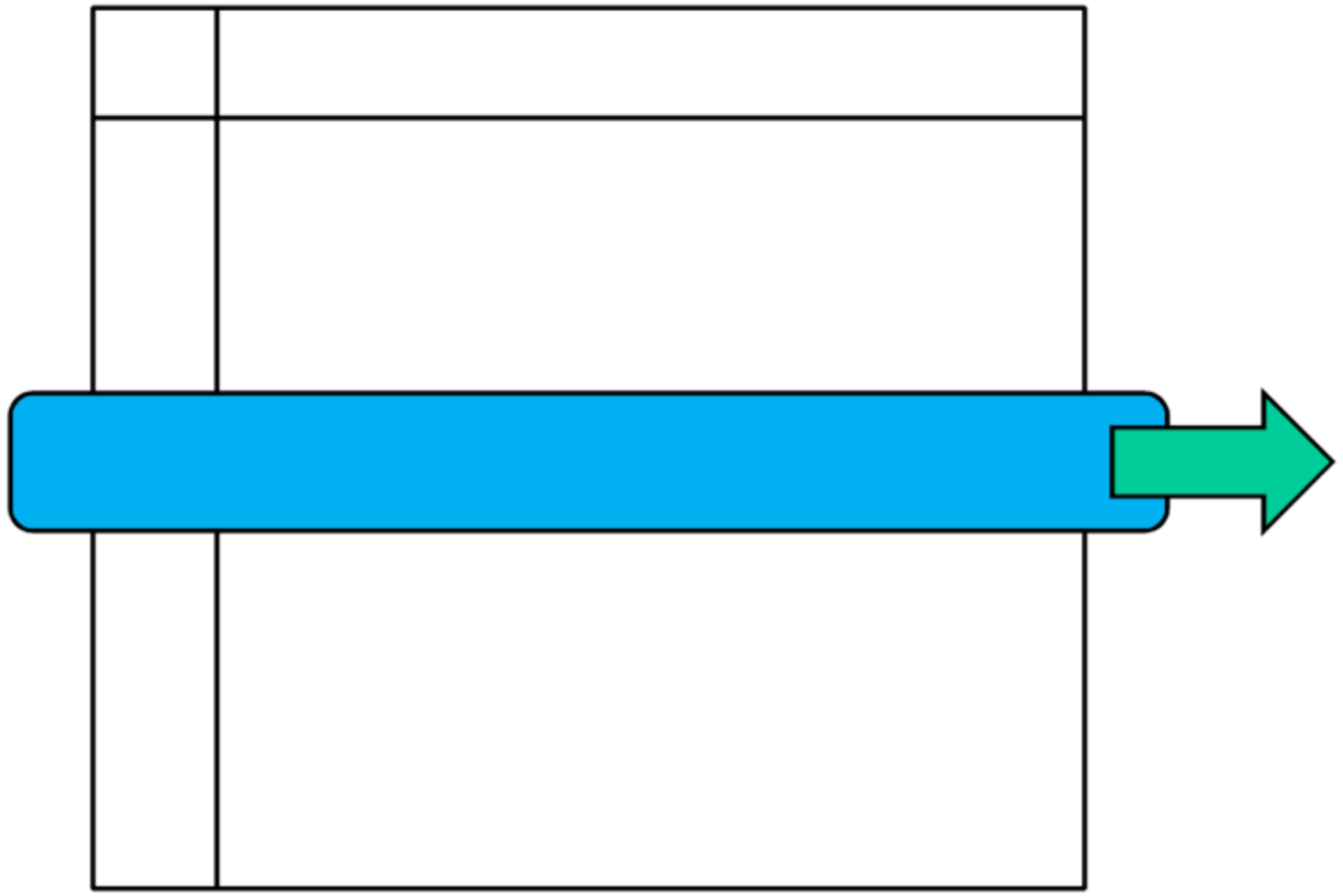
- Suppose a program generates the 14-bit address 0x01AA
  - In binary: 00000110101010
- All 8 words in that block in main memory are moved to cache block 0101

0000011	0101	010
Tag	Block	Word

14-bit Memory Address

Direct Mapped Example (3)

Main Memory



Cache

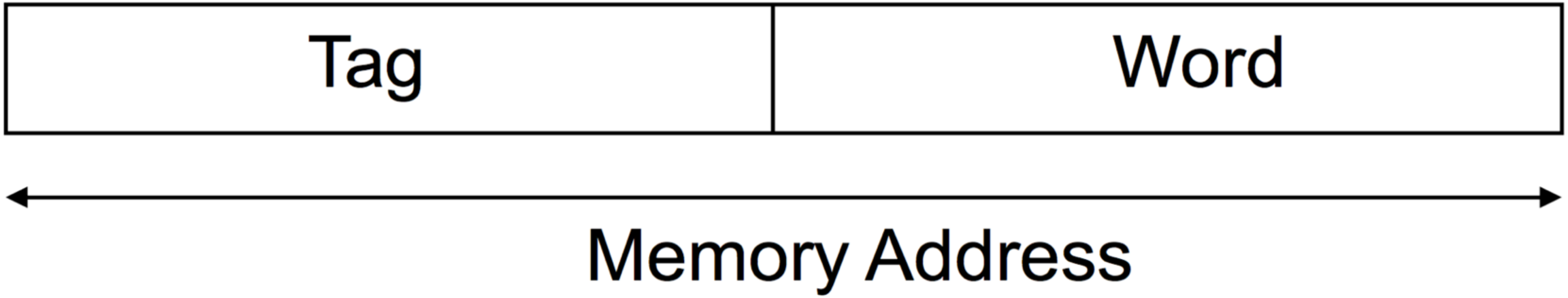
Block	Tag	Data	Valid
0000			
(…)			
0101	0000011	Words 000 through 111	1
(…)			
1111			

Direct Mapped Example (4)

- Suppose subsequently the program generates address 0x01AB
  - In binary: 0000000110101011
- The contents of this address can already be found in the cache
  - Cache location 0101 is valid, and the tag matches
- If, on the other hand, the program generates address 0x03AB (in binary, 0000001110101011), block 0101 will be evicted from the cache and a new block will be brought in
  - Cache location 0101 is valid but the tag doesn't match

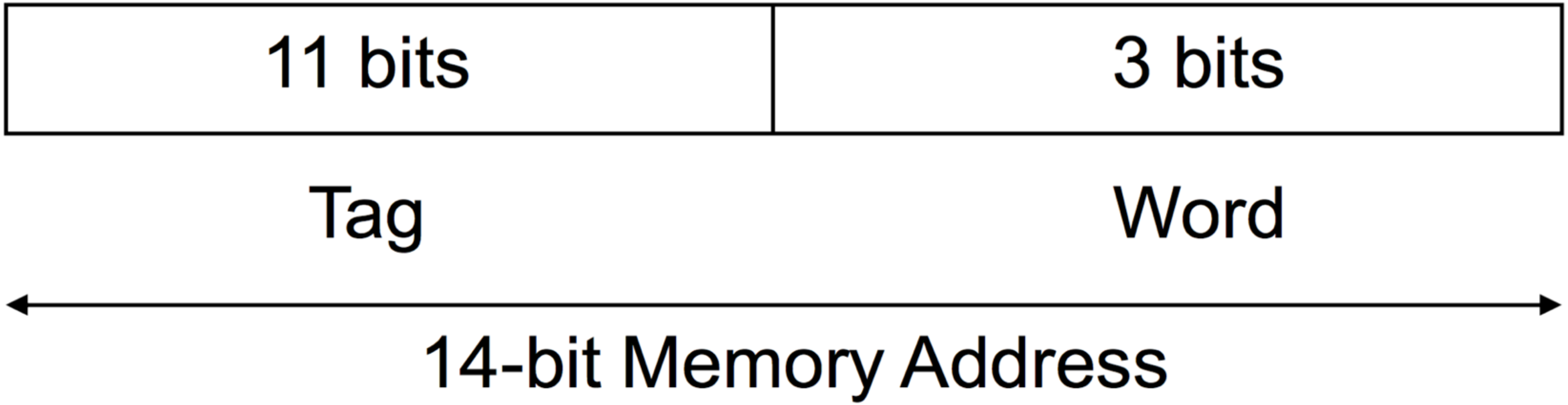
Fully Associative Cache

- Any block from main memory can be placed anywhere in the cache
  - In this way, cache would have to fill up before any blocks are evicted
- A memory address is partitioned into only two fields: the tag and the word
  - When the cache is searched, all tags are searched in parallel to retrieve the data (costly implementation)



Fully Associative Cache Example

- Suppose, as before, we have 14-bit memory addresses and a cache with 16 blocks, each block of size 8
- The field format of a memory reference is:



CHECK POINT

As a checkpoint of your understanding, please pause the video and make sure you can do the following:

- Describe how address translation takes place using fully associative mapped caches and direct mapped caches

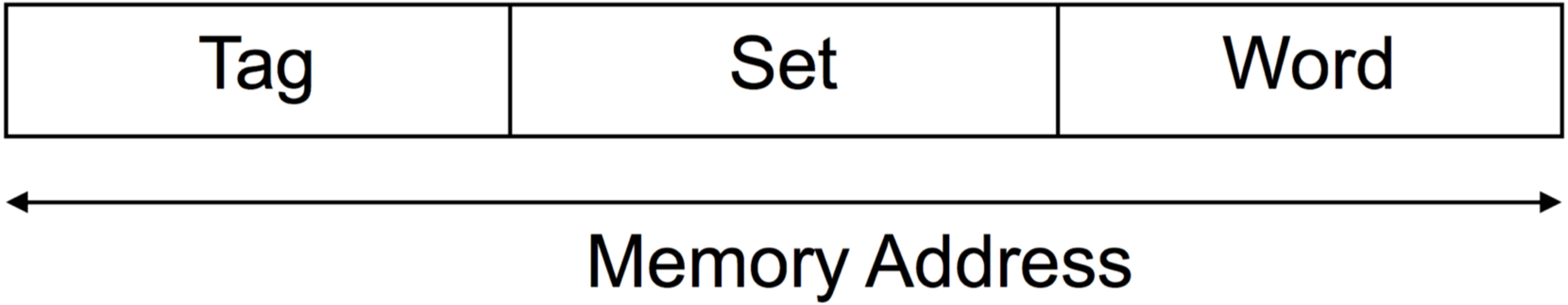
If you have any difficulties, please review the lecture video before continuing.

Set Associative Cache

- Set associative cache combines the ideas of direct mapped cache and fully associative cache
- An N-way set associative cache mapping is like direct mapped cache in that a memory reference maps to a particular location in cache
- Unlike direct mapped cache, a memory reference maps to a set of several cache blocks, similar to the way in which fully associative cache works
- Instead of mapping anywhere in the entire cache, a memory reference can map only to the subset of cache slots

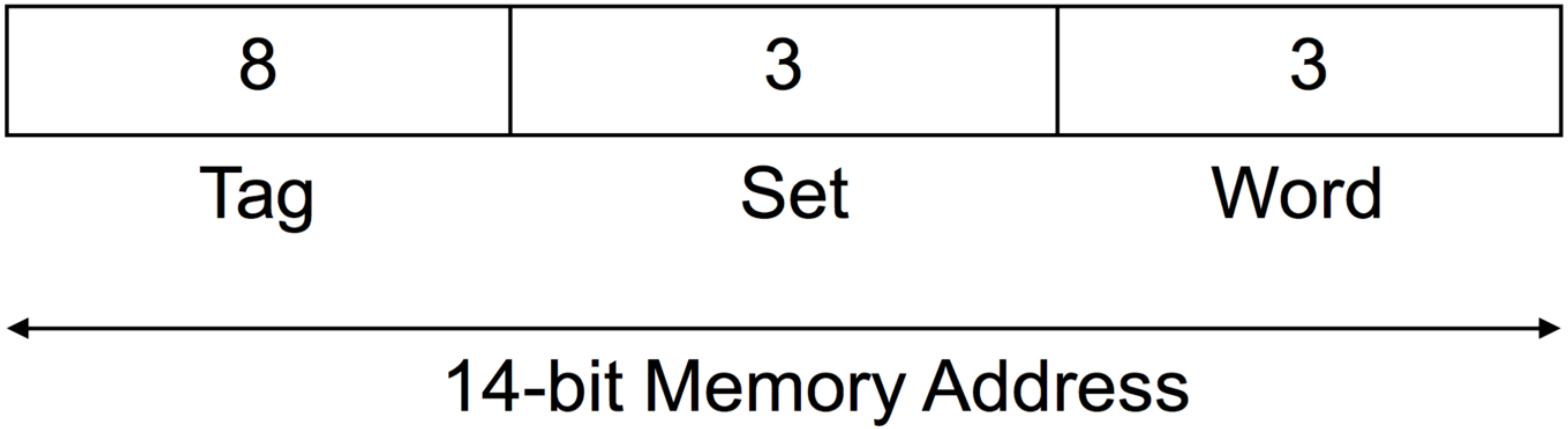
Main Memory Address Using Set Associative Cache

- Memory address is partitioned into 3 fields
- As with direct-mapped cache, the word field chooses the word within the cache block, and the tag field uniquely identifies the memory address
- The set field determines the set to which the memory block maps



Set Associative Example

- Suppose 14-bit memory addresses and a 2-way set associative cache that can hold 16 8-word blocks
  - 2-way means each set contains 2 blocks (so, 8 sets)
  - Need 3 bits for word addresses
  - Need 3 bits for set addresses
  - Have 8 bits for tag addresses



Replacement Policies

- With fully associative and set associative cache, a replacement policy is invoked when it becomes necessary to evict a block from cache
    - The block to be evicted is called the victim block
- An optimal replacement policy would be able to look into the future to see which blocks won't be needed for the longest period of time
    - Although it is impossible to implement an optimal replacement algorithm, it is instructive to use it as a benchmark for assessing the efficiency of a feasible replacement scheme

Replacement Policy Examples

- Least recently used (LRU) algorithm: keeps track of the last time that a block was accessed and evicts the block that has been unused for the longest period of time
  - High complexity: LRU has to maintain an access history for each block
- First-in, first-out (FIFO): evict the block that has been in the cache the longest, regardless of when it was last used
- Random replacement policy: evicts a block at random and replaces it with a new block

CHECK POINT

As a checkpoint of your understanding, please pause the video and make sure you can do the following:

- Describe how address translation takes place using set associative mapped caches
- Explain the concept of a cache replacement policy

If you have any difficulties, please review the lecture video before continuing.

Performance Metrics

- Hit Ratio

$$H = \frac{\#\ of\ times\ referenced\ word\ is\ in\ cache}{\#\ of\ memory\ accesses}$$

- Effective Access Time

$$EAT = H \times Access_C + (1 - H) \times Access_{MM}$$

- Access_C and Access_MM are access times for the cache and main memory, respectively

Hit Ratio Example (1)

- Direct-mapped cache with four 16-word blocks
- Cache initially empty
- Cache access time is 80 ns
- Time for transferring memory block to cache is 2500 ns
- Program executes from memory locations 48-95, then loops 10 times from 15-31 before halting

Hit Ratio Example (2)

- Initially, 1 miss, since block of addresses 48-63 must be loaded into the cache
- 15 hits follow (for locations 49 through 63)
- Same for blocks 64-79 and 80-95
- Two additional misses for blocks of addresses 0-15 and 16- 31
- 15 hits follow (for locations 17-31)
- An additional 9 x 17 = 153 hits as the program loops 9 more times through addresses 15-31
- Total number of accesses = 48 + 10 x 17 = 218
- Hit ratio = (218 – 5)/218 = 97.7%

EAT Example

- Using the parameters in the previous example:
  - $H = 0.977$
  - $Access_C = 80$ ns
  - $Access_{MM} = 2500$ ns
- EAT = 0.977 x 80 + (1-0.977) x 2500 = 135.7 ns

Dirty Blocks

- Dirty blocks are blocks that have been updated while they were in the cache
- Write policies
    - Write through: updates cache and main memory simultaneously on every write
    - Write back (also called copy back): updates memory only when the block is selected for replacement

Enhancements to Integrated Cache

- Integrated (or unified) cache: holds both data and instructions
  - The cache we have been discussing thus far
- Harvard cache: separate caches for data and instructions
  - Better locality, higher complexity
- Victim cache: a separate cache to hold blocks that have been evicted recently
- Trace cache: a variant of an instruction cache that holds decoded instructions for program branches
  - Provides the illusion that noncontiguous instructions are really contiguous

Cache Hierarchies

- Multilevel cache hierarchies: the levels of cache form their own small memory hierarchy
- Level 1 cache (8KB to 64KB) is on the processor itself
  - Access time ~ 4 ns
- Level 2 cache (64KB to 2MB) may be on the motherboard, or on an expansion card
  - Access time ~ 10 - 20 ns
- Level 3 cache (2MB to 256MB) refers to cache that is situated between the processor and main memory

CHECK POINT

As a checkpoint of your understanding, please pause the video and make sure you can do the following:

- Review the calculations for the hit ratio and effective access time for the examples given on the previous slides
- Describe example enhancements to unified cache

If you have any difficulties, please review the lecture video before continuing.

Summary

- The cache consists of a small amount of fast memory between the CPU and main memory
  - It takes advantage of spatial and temporal locality
- Direct Mapped and Associative Mapped caches are different approaches for mapping between main memory addresses and cache locations
- Cache replacement policies refer to the decision of which current block in the cache to replace
- Hit ratio and effective access time are performance metrics for the cache
- Systems often implement a multi-level cache hierarchy

MODULE 7: Memory Systems

Lecture 7.2
Cache Memory

Prepared By:

- Scott F. Midkiff, PhD
- Luiz A. DaSilva, PhD
- Kendall E. Giles, PhD

Electrical and Computer Engineering

Virginia Tech