Memory Management

Dr. Igor Ivkovic

iivkovic@uwaterloo.ca

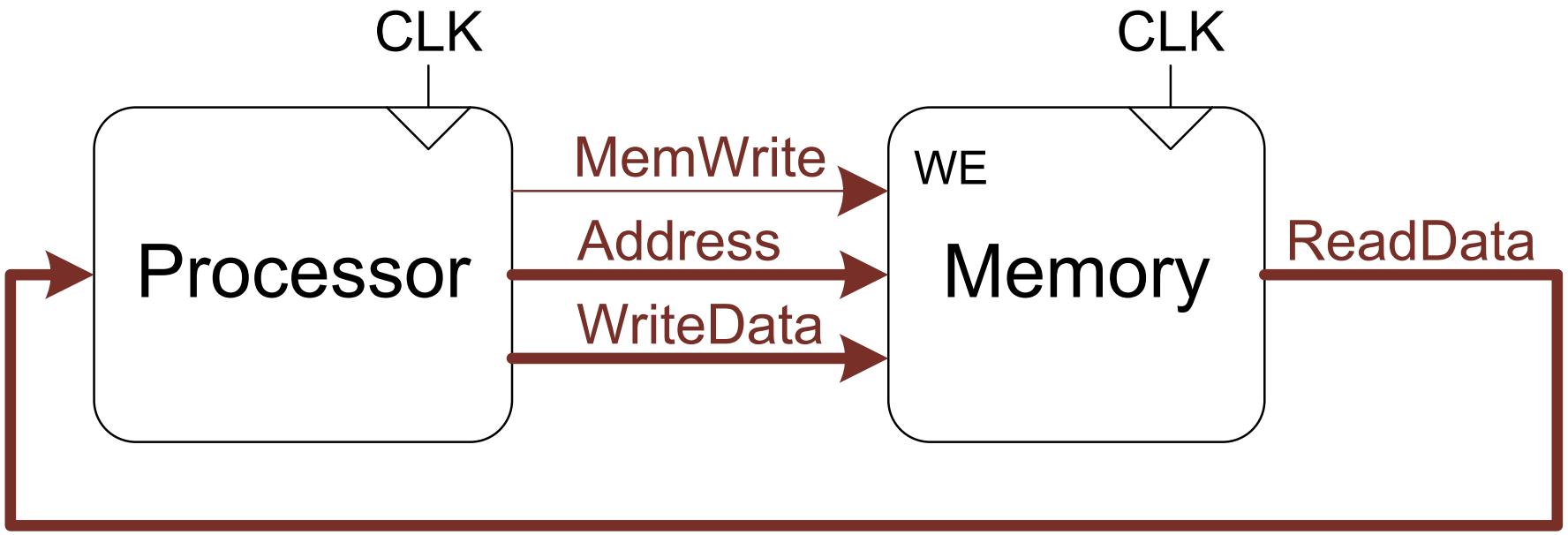
[with material from “Computer Organization and Design” by Patterson and Hennessy, and “Digital Design and Computer Architecture” by Harris and Harris, both published by Morgan Kaufmann]

Objectives

- Introduction to memory management
- Using and managing cache memory
- Introduction to virtual memory

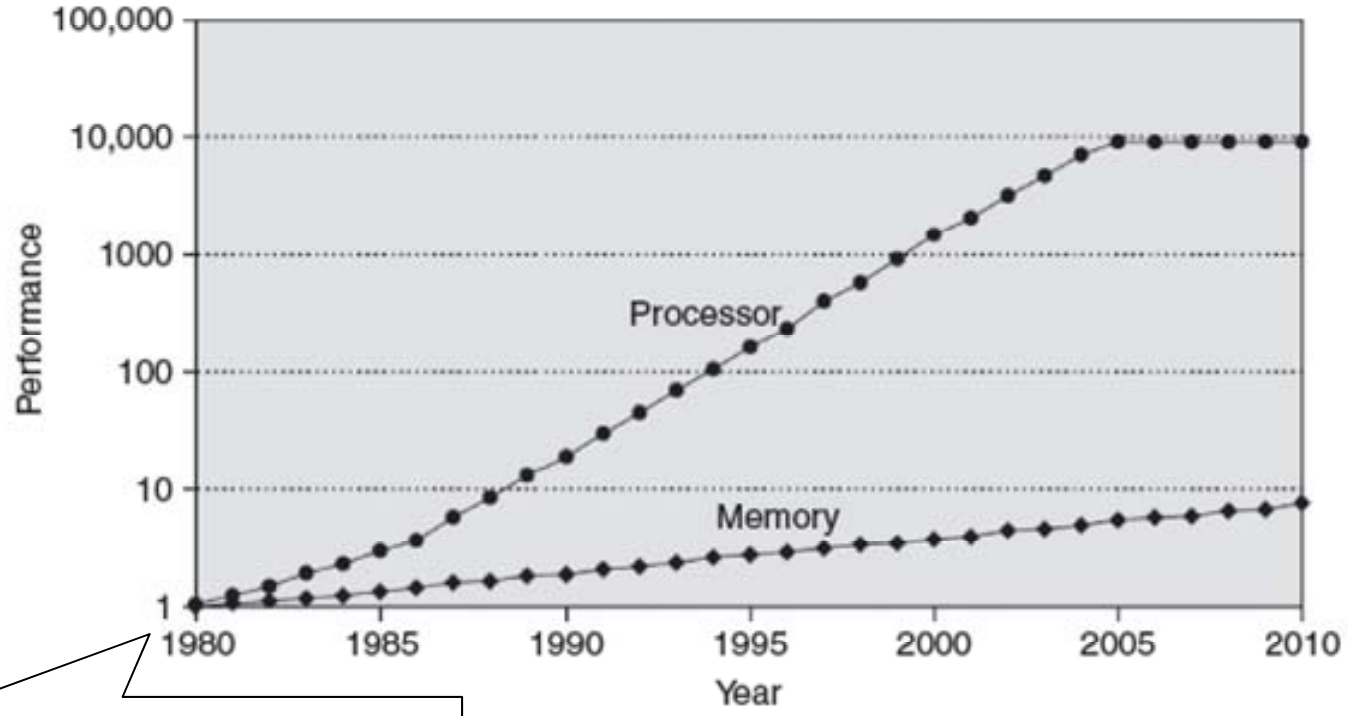
Introduction to Memory Management /1

- **Computer performance depends on:**
  - Processor performance
  - Memory system performance
- **Memory Interface Visualized:**



Introduction to Memory Management /2

- **Processor-Memory Gap:**
  - In prior chapters, we assumed that can access memory in 1 clock cycle
  - In reality, this has not been true since the 1980s



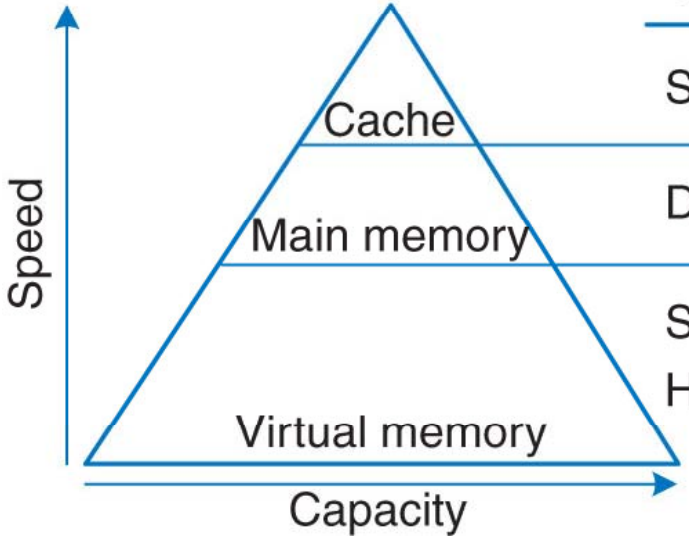
1980 processor speeds as the performance baseline

Introduction to Memory Management /3

- **Memory System Challenge:**
  - Make the memory system appear as fast as the processor
  - Use a hierarchy of memories
  - **Ideal memory system should be:**
    - Fast
    - Large (capacity)
    - Cheap (inexpensive)
  - **However, we can only choose two at a time**

Introduction to Memory Management /4

- **Memory Hierarchy:**
  - Cache – using Static RAM (SRAM)
    - Stores commonly used instructions and data
  - Main memory – using Dynamic RAM (DRAM)
    - Stores other instructions and data that do not fit into the cache
  - Virtual memory – uses external storage, such as HDD or SSD
    - Extends the main memory capacity



Technology	Price / GB	Access Time (ns)	Bandwidth (GB/s)
SRAM	$10,000	1	25+
DRAM	$10	10–50	10
SSD	$1	100,000	0.5
HDD	$0.1	10,000,000	0.1

Approximate values, for comparison only

Introduction to Memory Management /5

- **Memory Locality:**
  - Exploit locality to improve memory access speed
- **Temporal Locality:**
  - Locality in time
  - If data has been used recently, likely to use it again soon
  - How to exploit: Keep recently accessed data in higher levels of memory hierarchy
- **Spatial Locality:**
  - Locality in space
  - If data has been used recently, likely to use nearby data soon
  - How to exploit: When accessing data, bring nearby data into higher levels of memory hierarchy too

Introduction to Memory Management /6

- **Memory Performance:**
  - **Hit:** Data found in that level of memory hierarchy
    - Hit Rate (HR) = # hits / # memory accesses = 1 – Miss Rate
  - **Miss:** Data not found (must go to the next level)
    - Miss Rate (MR) = # misses / # memory accesses = 1 – Hit Rate
- **Average Memory Access Time (AMAT):**
  - Average time for processor to access data
  - **AMAT = $t_{cache} + MR_{cache}[t_{MM} + MR_{MM}(t_{VM})]$**
  - That is, AMAT includes
    - The time it takes to access data from cache (t_{cache}), plus
    - Miss rate of cache (MR_{cache}) x the time it takes to use the main memory (t_{MM}), plus
    - Miss rate of cache (MR_{cache}) x the miss rate of the main memory (MR_{MM}) x the time it takes to access virtual memory

Introduction to Memory Management /7

- **Types of Misses:**
  - **Compulsory:** The first time data accessed
  - **Capacity:** Cache is too small to hold all data of interest
  - **Conflict:** Data of interest maps to same location in cache
- **Miss Penalty:**
  - The time it takes to retrieve a block from a lower level in the memory hierarchy

Introduction to Memory Management /8

- **Memory Performance Example:**
  - A program P has 2,000 loads and stores
  - 1,250 of these values are in cache
  - The rest are supplied by other levels of memory hierarchy
  - What are the hit and miss rates for the cache?
    - **Hit Rate = 1250/2000 = 0.625**
    - **Miss Rate = 750/2000 = 0.375 = 1 – Hit Rate**

Introduction to Memory Management /9

- **Memory Performance Example Continued:**
  - Suppose the processor has 2 levels of memory hierarchy: cache and main memory
  - t_{cache} = 1 cycle, t_{MM} = 100 cycles
  - What is the AMAT of the program P?
    - **AMAT = t_{cache} + MR_{cache}(t_{MM})**

      = [1 + 0.375(100)] cycles

      = 38.5 cycles

Cache Memory Introduced /1

- **Cache Memory:**
  - Highest level in memory hierarchy
  - Fast (typically ~1 cycle access time)
  - Ideally supplies most data to processor
  - Usually holds most recently accessed data
- **Questions to Answer:**
  - What data is held in the cache?
  - How is data found?
  - What data is replaced?

Cache Memory Introduced /2

- **What data is held in the cache?**
  - Ideally, cache anticipates needed data and puts it in cache
  - But impossible to predict future
  - Use past to predict future – temporal and spatial locality:
    - **Temporal locality:** copy newly accessed data into cache
    - **Spatial locality:** copy neighboring data into cache too

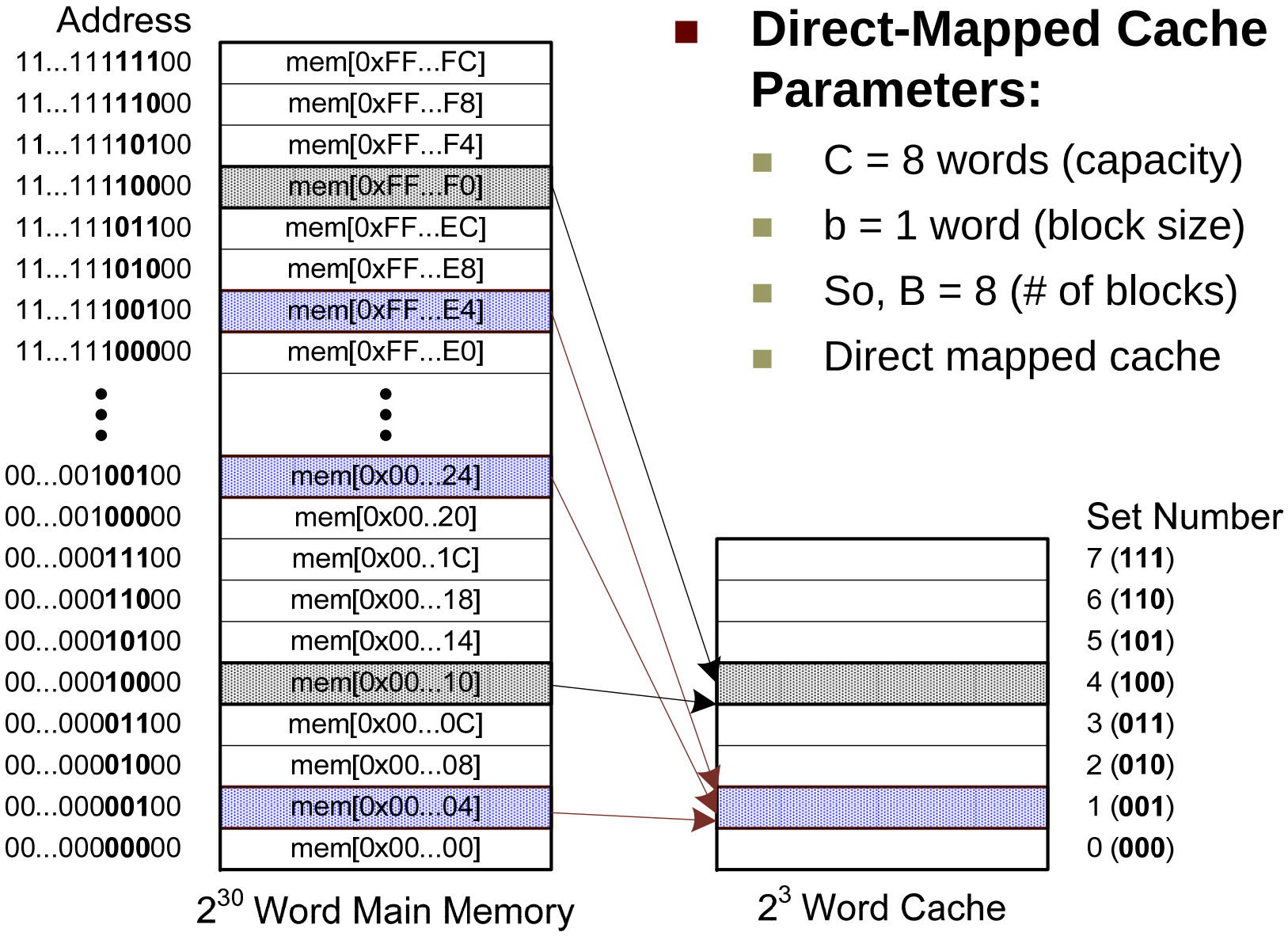
Cache Memory Introduced /3

- **Cache Terminology:**
  - **Capacity (C):**
    - Number of data bytes in cache
  - **Block size (b):**
    - Bytes of data brought into cache at once
  - **Number of blocks (B = C/b):**
    - Number of blocks in cache: B = C/b
- **Mapping:**
  - Relationship between the data address in main memory and the data location in cache
  - **Each memory address maps to exactly one cache set**
  - **Degree of associativity (N):** Number of blocks in a set
  - **Number of sets: S = B/N**

Cache Memory Introduced /4

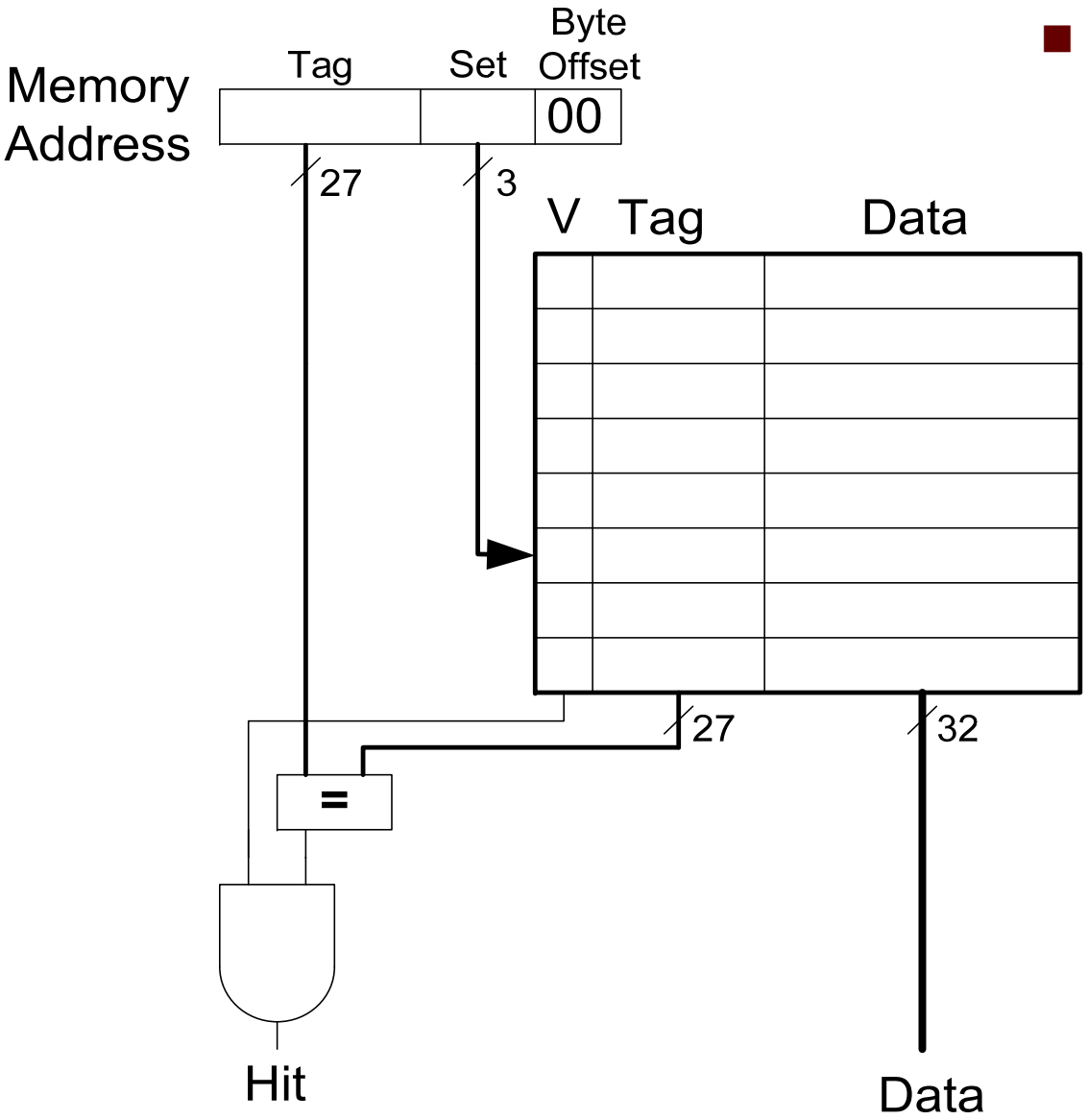
- **How is data found?**
  - Cache organized into S sets
  - Each memory address maps to exactly one set
  - Caches categorized by # of blocks in a set:
    - **Direct mapped:** 1 block per set
    - **N-way set associative:** N blocks per set (data can be mapped to any one of the blocks in the corresponding set)
    - **Fully associative:** All cache blocks in 1 set
  - Examine each organization for a cache with:
    - Capacity (C = 8 words)
    - Block size (b = 1 word)
    - So, number of blocks (B = 8)

Direct-Mapped Cache /1



- **Direct-Mapped Cache Parameters:**
  - C = 8 words (capacity)
  - b = 1 word (block size)
  - So, B = 8 (# of blocks)
  - Direct mapped cache

Direct-Mapped Cache /2

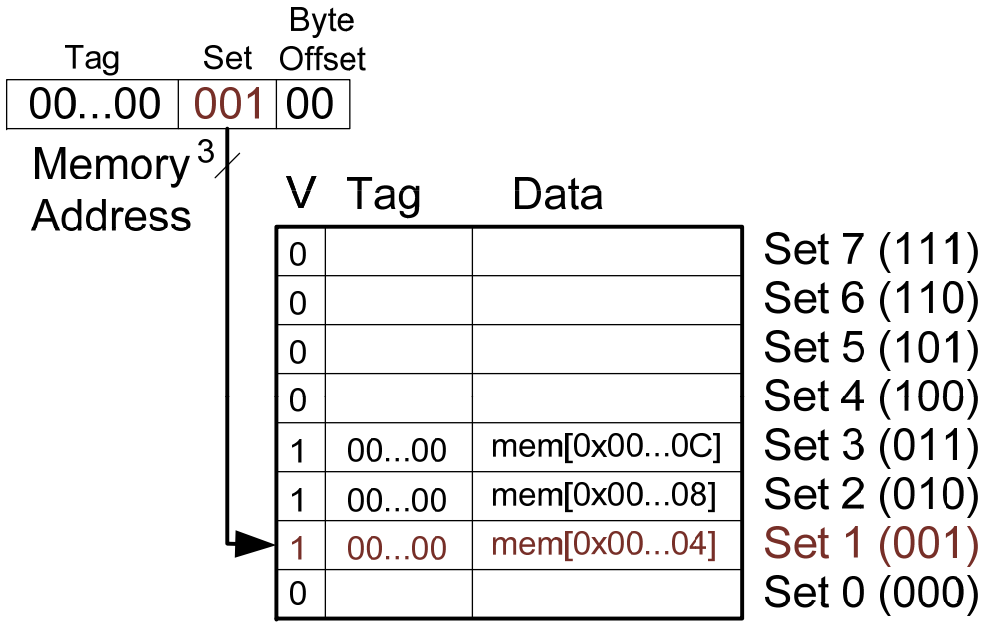


- **Direct-Mapped Cache Example:**
  - **8 entry x (1 + 27 + 32)-bit SRAM**
  - **Valid bit (V):** Indicates if the cache set holds useful data
  - **Tag:** 27 bits of the main memory address
  - **Set bits:** 3 bits representing the set identifier

Direct-Mapped Cache /3

- **Example MIPS code:**

```
addi $t0, $0, 5
loop: beq  $t0, $0, done
      lw   $t1, 0x4($0)
      lw   $t2, 0xC($0)
      lw   $t3, 0x8($0)
      addi $t0, $t0, -1
      j    loop
done:
```



- The code loops 5 times
- On the first loop, data is loaded into cache
- On the remaining loops, data is available in cache

3 misses on the first cycle,
12 hits on the remaining four cycles

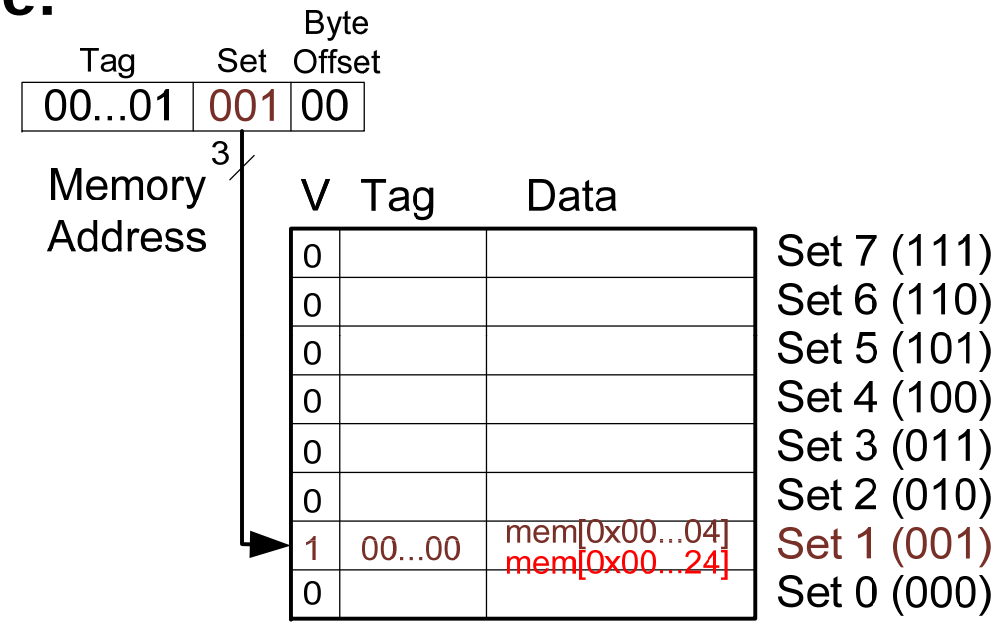
Miss rate = 3/15 = 20%

- **This example utilizes temporal locality, and the misses are compulsory misses**

Direct-Mapped Cache /4

- **Another MIPS example:**

```
addi $t0, $0, 5
loop: beq  $t0, $0, done
      lw   $t1, 0x4($0)
      lw   $t2, 0x24($0)
      addi $t0, $t0, -1
      j    loop
done:
```



- The code loops 5 times
- Both 0x4 and 0x24 map to the same cache set
- On each memory access, the previous data is evicted

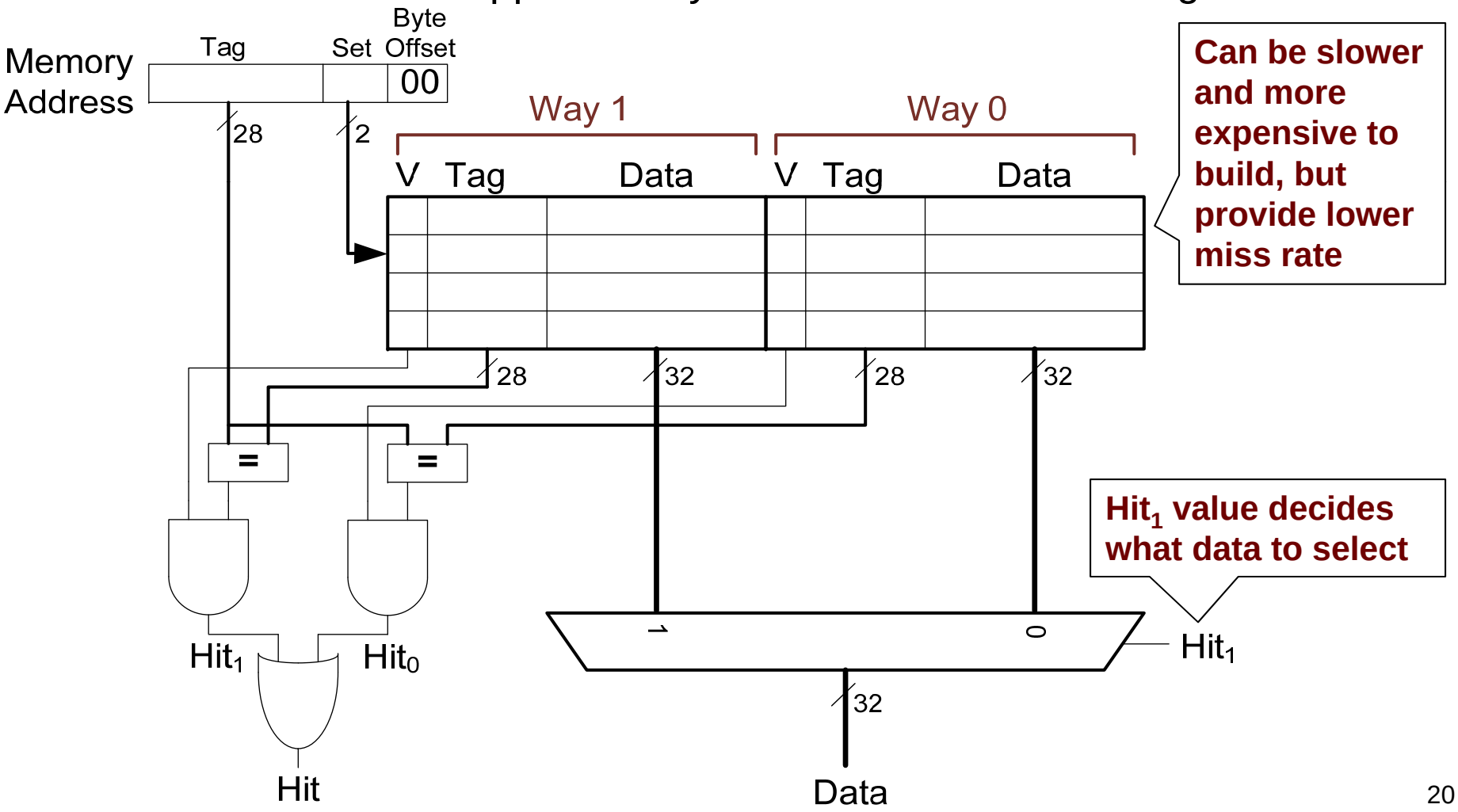
2 misses on the first cycle,
8 misses on the remaining four

Miss rate = 10/10 = 100%

- **This example utilizes temporal locality, but the misses are conflict misses**

N-Way Associative Cache /1

- **N-Way Associative Cache:**
  - Data can be mapped to any one block in the matching set



N-Way Associative Cache /2

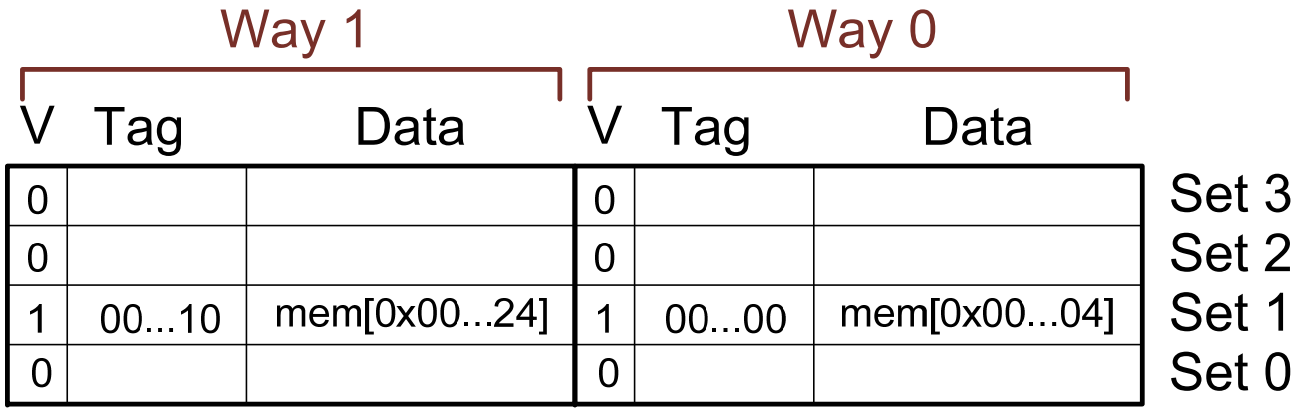
- **MIPS example repeated:**

```
addi $t0, $0, 5
loop: beq  $t0, $0, done
      lw   $t1, 0x4($0)
      lw   $t2, 0x24($0)
      addi $t0, $t0, -1
      j    loop
Done:
```

  - 2-way associative cache (only 4 sets instead of 8)

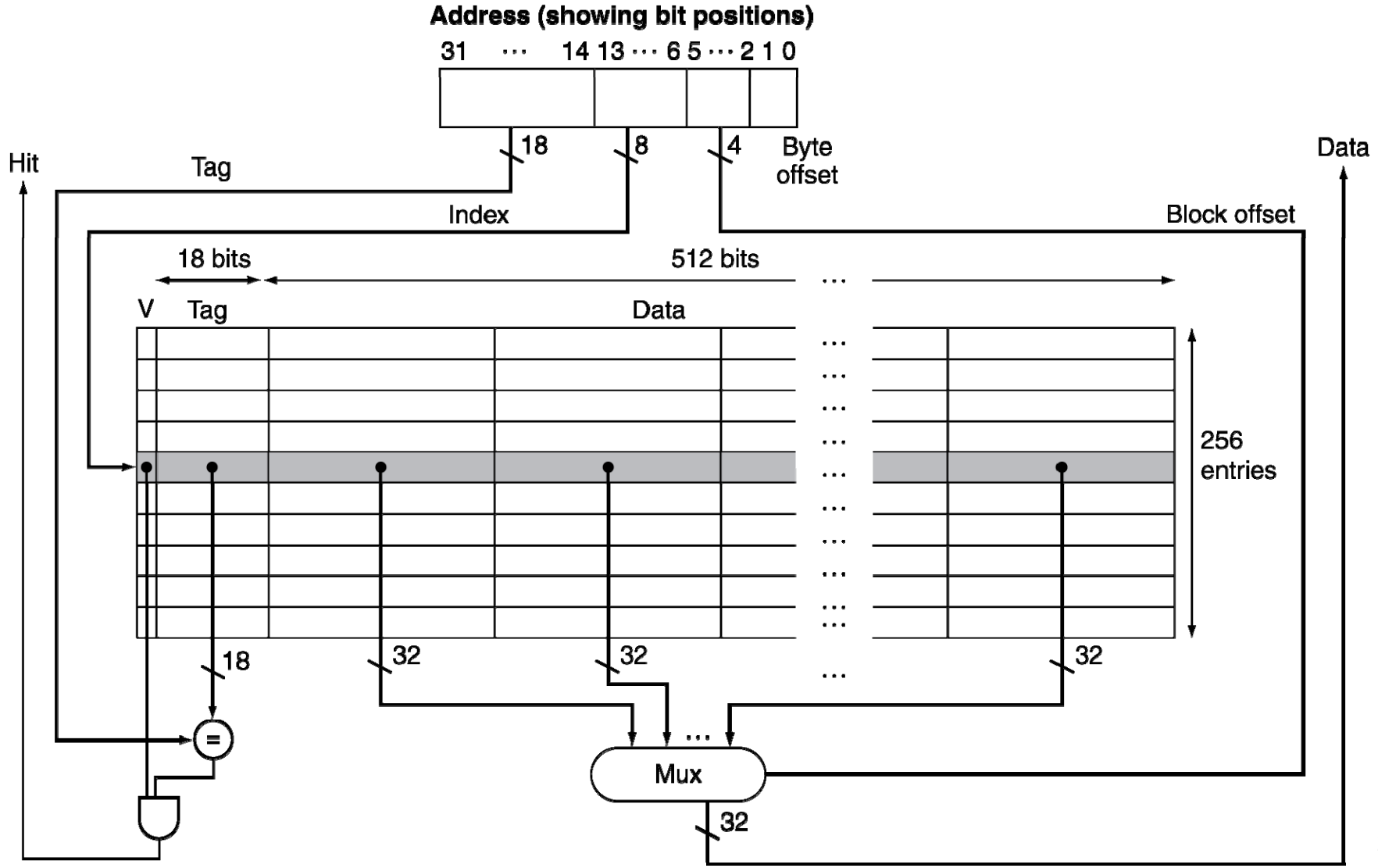
2 misses on the first cycle, 8 hits on the remaining four

Miss rate = 2/10 = 20% (instead of 100% as shown in the previous example)



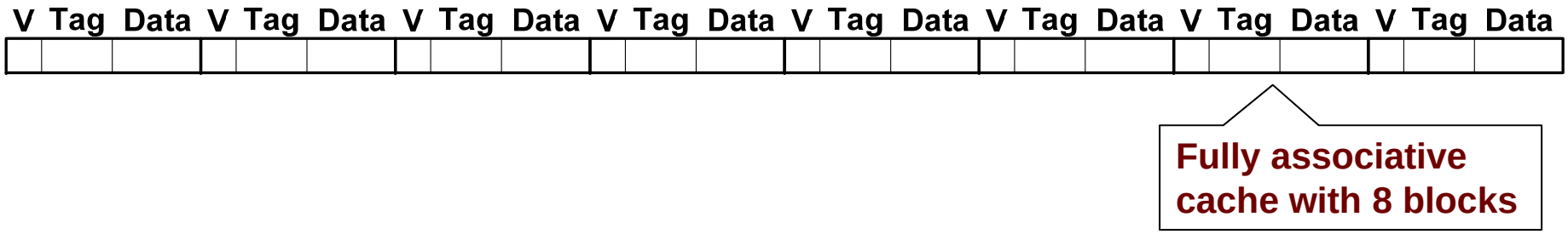
N-Way Associative Cache /3

- **Practical Example: Intrinsity FastMATH memory**



Fully Associative Cache

- **Fully Associative Cache:**
  - All B cache blocks in 1 set
  - Each way/tag is checked if it stores the needed value
  - The B-to-1 multiplexer chooses the matching value
  - Greatly reduces the miss rate
  - But can be significantly expensive to build due to the additional hardware (many comparators and multiplexers)
  - Can be useful for relatively small caches



Cache Misses Emphasized

- **What happens on a cache hit and miss?**
  - On cache hit, CPU proceeds normally
  - On cache miss
    - Stall the CPU pipeline
    - Fetch the block from next level of memory hierarchy
  - Instruction cache miss
    - Restart the instruction fetch
  - Data cache miss
    - Complete the data access

Memory Writes Emphasized /1

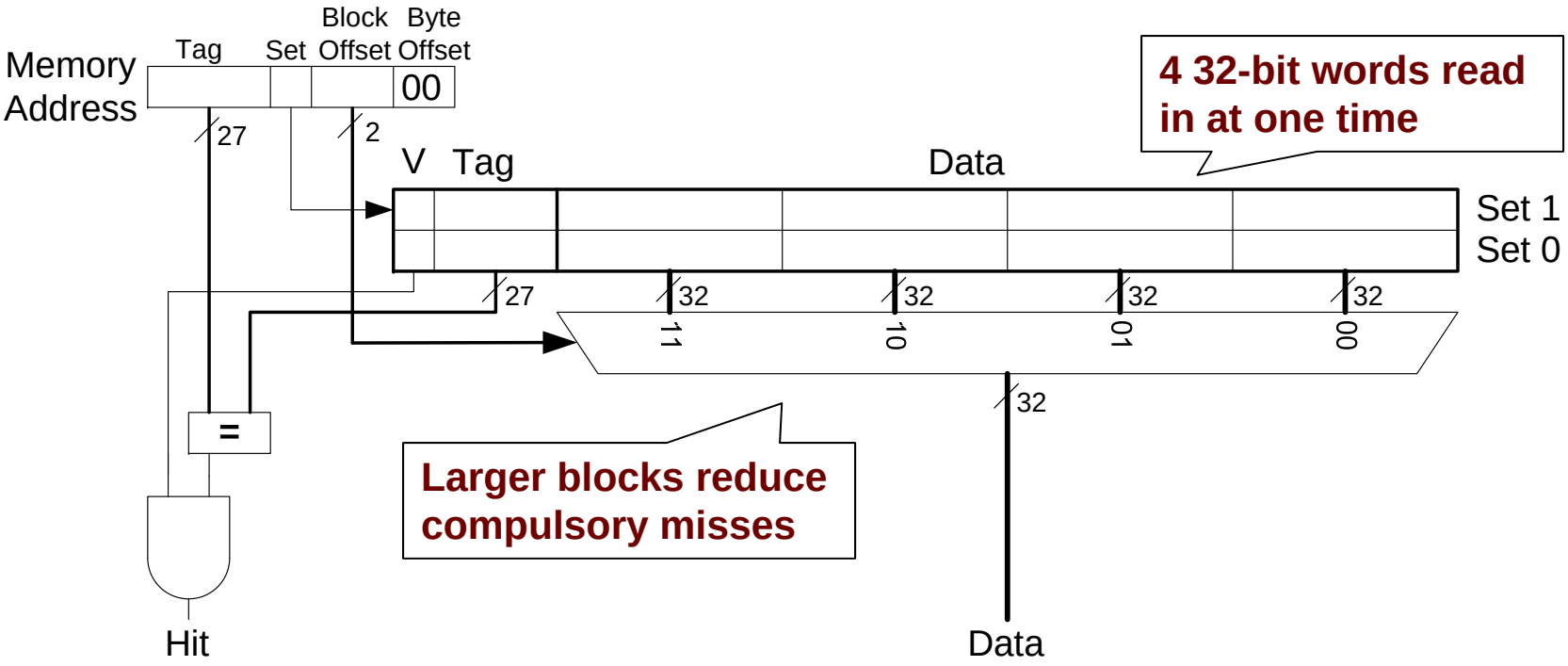
- On data writes, could just update the block in cache
  - But then cache and memory would be inconsistent
  - **Write-hit:** Needed main memory block is already in cache
  - **Write-miss:** Needed main memory block is not in cache
- **Write-through policy on write-hit:**
  - Need to also update main memory
  - However, this makes the writes take longer
  - Example: If base CPI = 1, 10% of instructions are stores, write to memory takes 100 cycles
    - Effective CPI = 1 + 0.1 × 100 = 11
  - **Solution: Write buffer**
    - Holds data waiting to be written to memory
    - CPU continues immediately
    - Only stalls on write if write buffer is already full

Memory Writes Emphasized /2

- **Write-back policy on write-hit:**
  - On data write, just update the block in cache
  - Keep track of whether each block is dirty
  - **When a dirty block is replaced:**
    - Write it back to memory
    - Can use a write buffer to allow replacing block to be read first
- What should happen on a write-miss?
  - **Allocate on miss:** Fetch the block to which we need to write, and store the data into it
    - Typically paired with write-back policy on write-hit
  - **Write-around policy:** Do not fetch the block, and just write directly to the main memory
    - Programs often write a whole block before reading it (e.g., initialization), so the most recently-written data not needed
    - Typically paired with write-through policy on write-hit

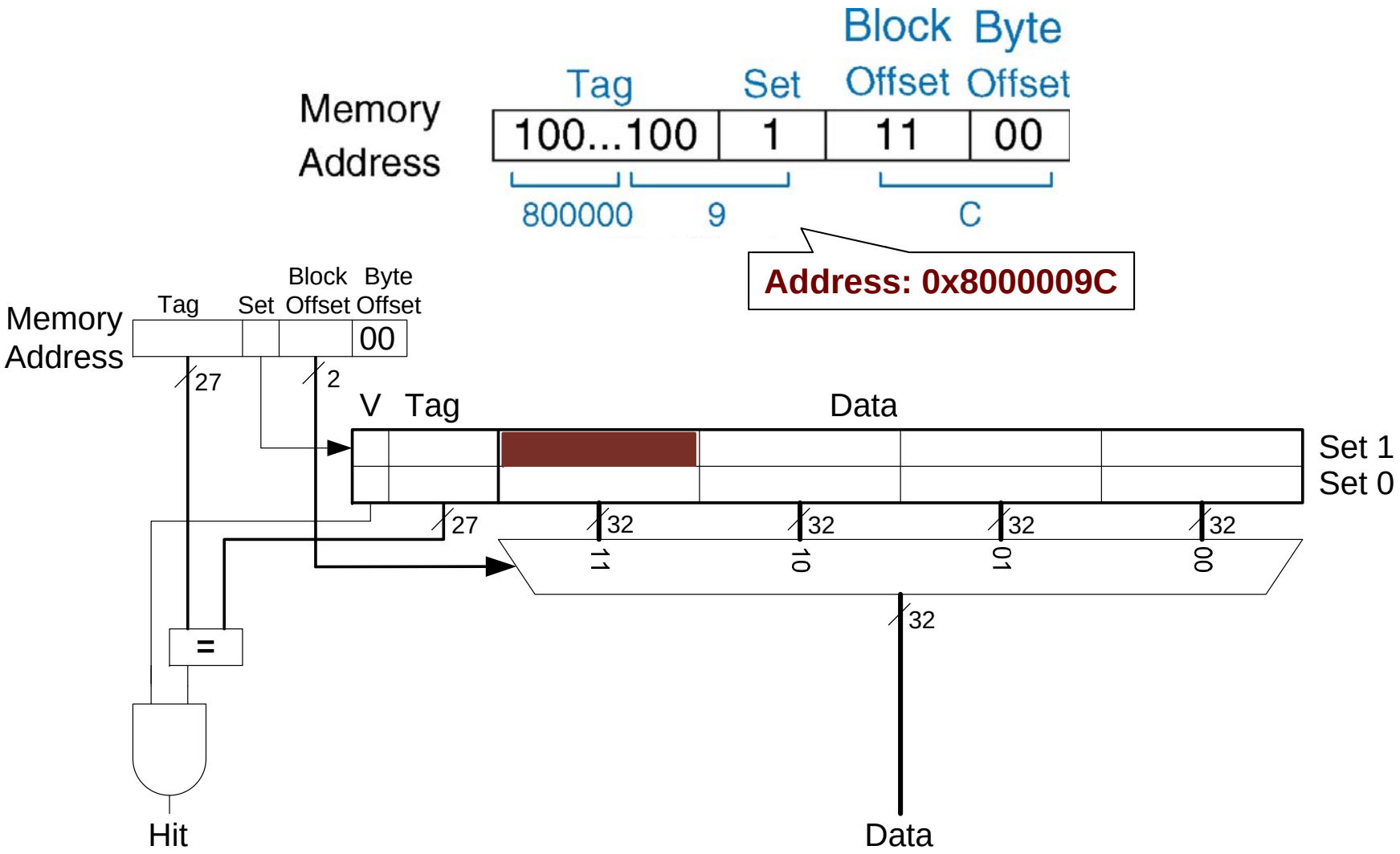
Cache with Larger Block Size /1

- **Cache with larger block size (using spatial locality):**
  - Capacity C – number of data bytes in cache – 8 words
  - Block size b – bytes of data brought in at once – 4 words
  - Direct-mapped cache – 1 block per set
  - Number of blocks in cache – B = 2 (C/b = 8/4 = 2)



Cache with Larger Block Size /2

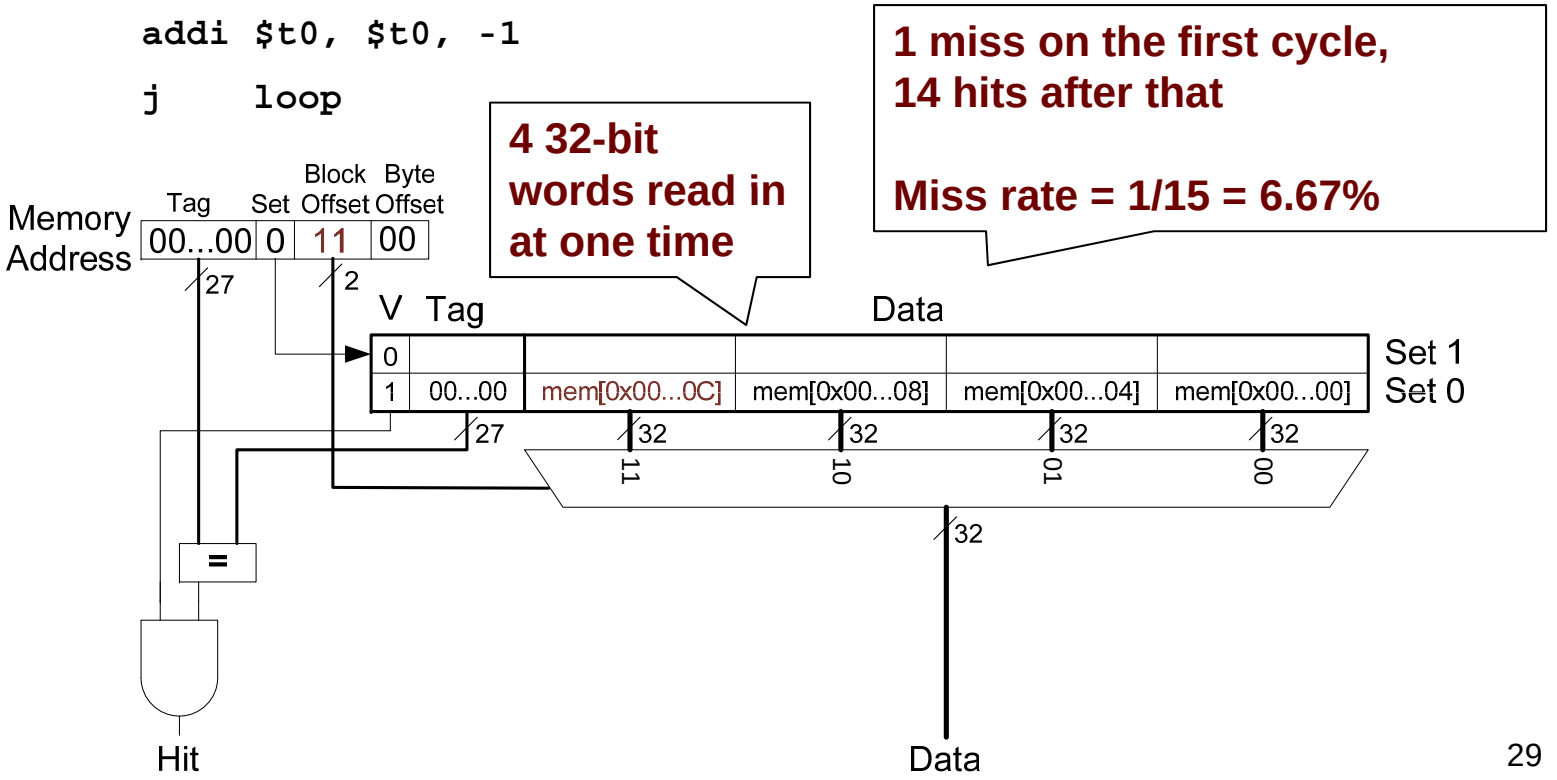
- **Cache with larger block size:**



Cache with Larger Block Size /3

- **MIPS example repeated (using spatial locality):**

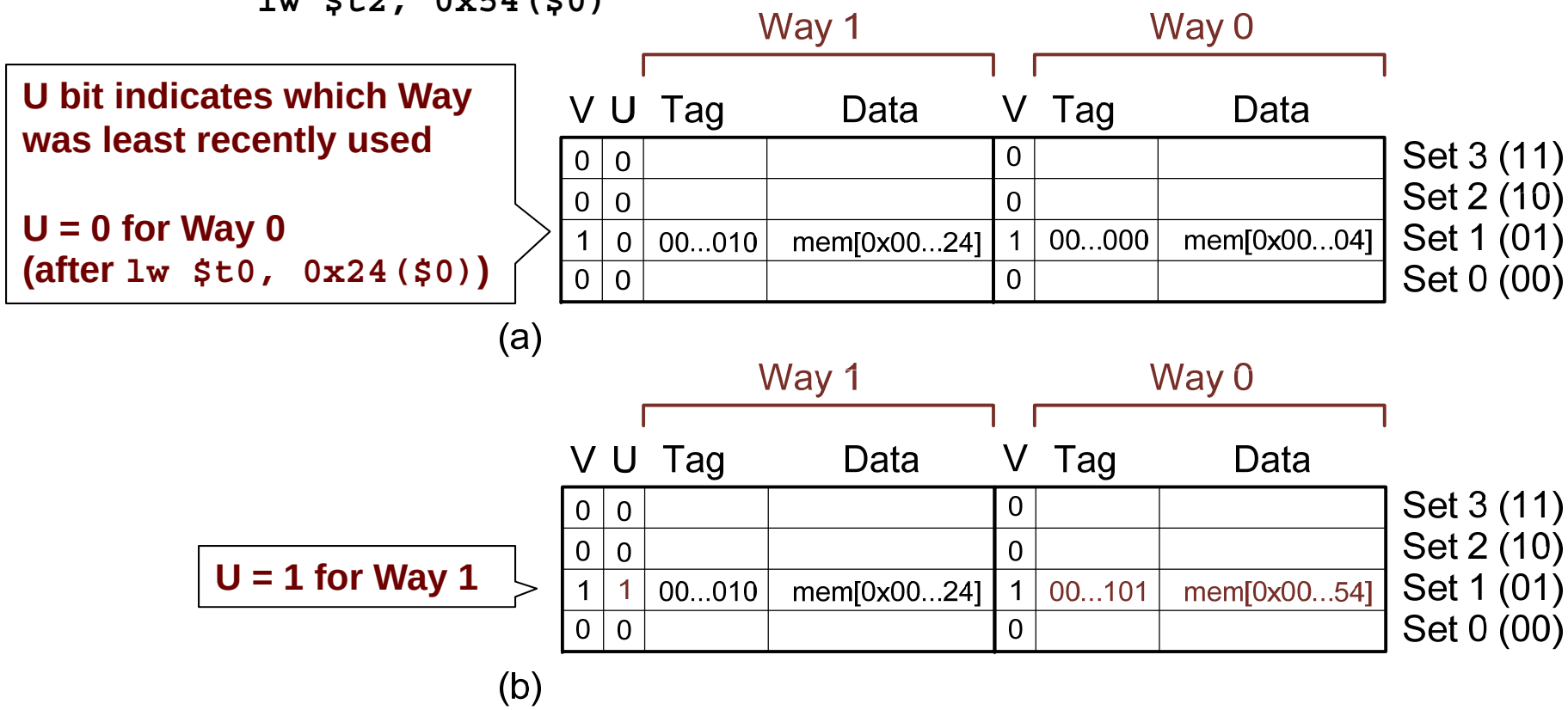
```
addi $t0, $0, 5
loop:       beq  $t0, $0, done
            lw   $t1, 0x4($0)
            lw   $t2, 0xC($0)
            lw   $t3, 0x8($0)
            addi $t0, $t0, -1
            j    loop
Done:
```



LRU Replacement /1

- **Least recently used (LRU) replacement:**
  - The least recently used block in a set evicted

```
lw $t0, 0x04($0)
lw $t1, 0x24($0)
lw $t2, 0x54($0)
```



LRU Replacement /2

- **Least recently used (LRU) replacement:**
  - For more than two Ways, divide the blocks into groups, and use U to select the group that was least recently used
  - Within the selected group, randomly select a block
  - **Such a policy is called pseudo-LRU replacement**

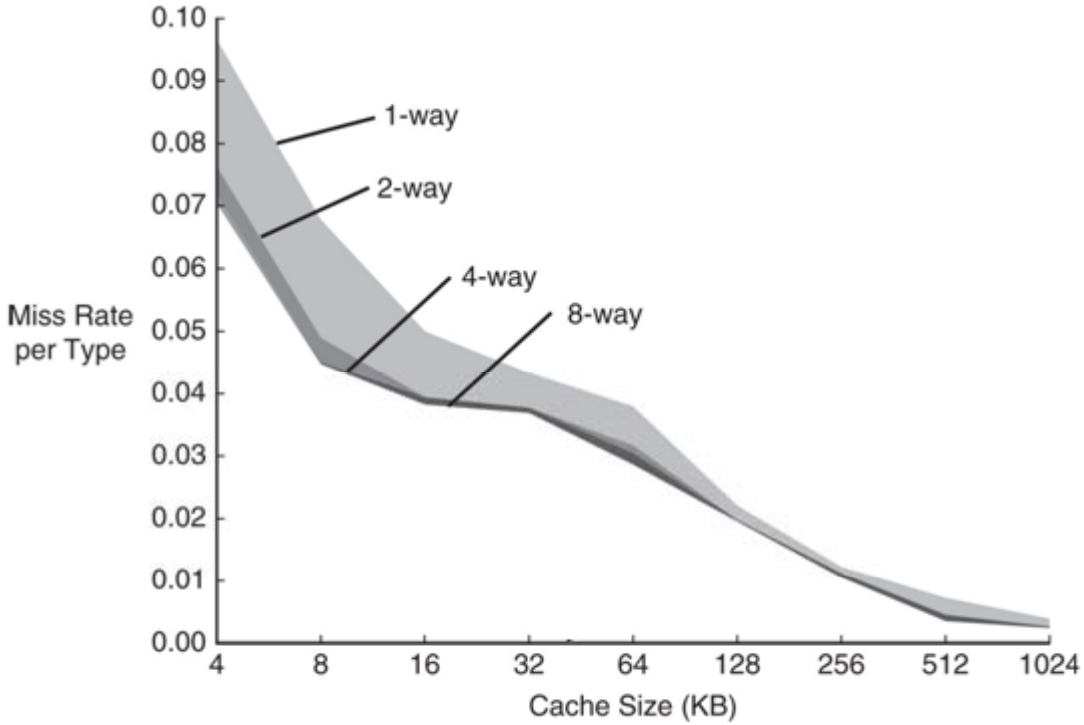
Cache Organization Summary /1

- **Cache Organization Summary:**
  - **Capacity (C):**
    - Number of data bytes in cache
  - **Block size (b):**
    - Bytes of data brought into cache at once
  - **Number of blocks (B = C/b):**
    - Number of blocks in cache: B = C/b
  - **Number of sets: S = B/N**

Organization	Number of Ways (*N*)	Number of Sets (*S = B/N*)
Direct Mapped	1	*B*
N-Way Set Associative	1 < *N* < *B*	*B* / *N*
Fully Associative	*B*	1

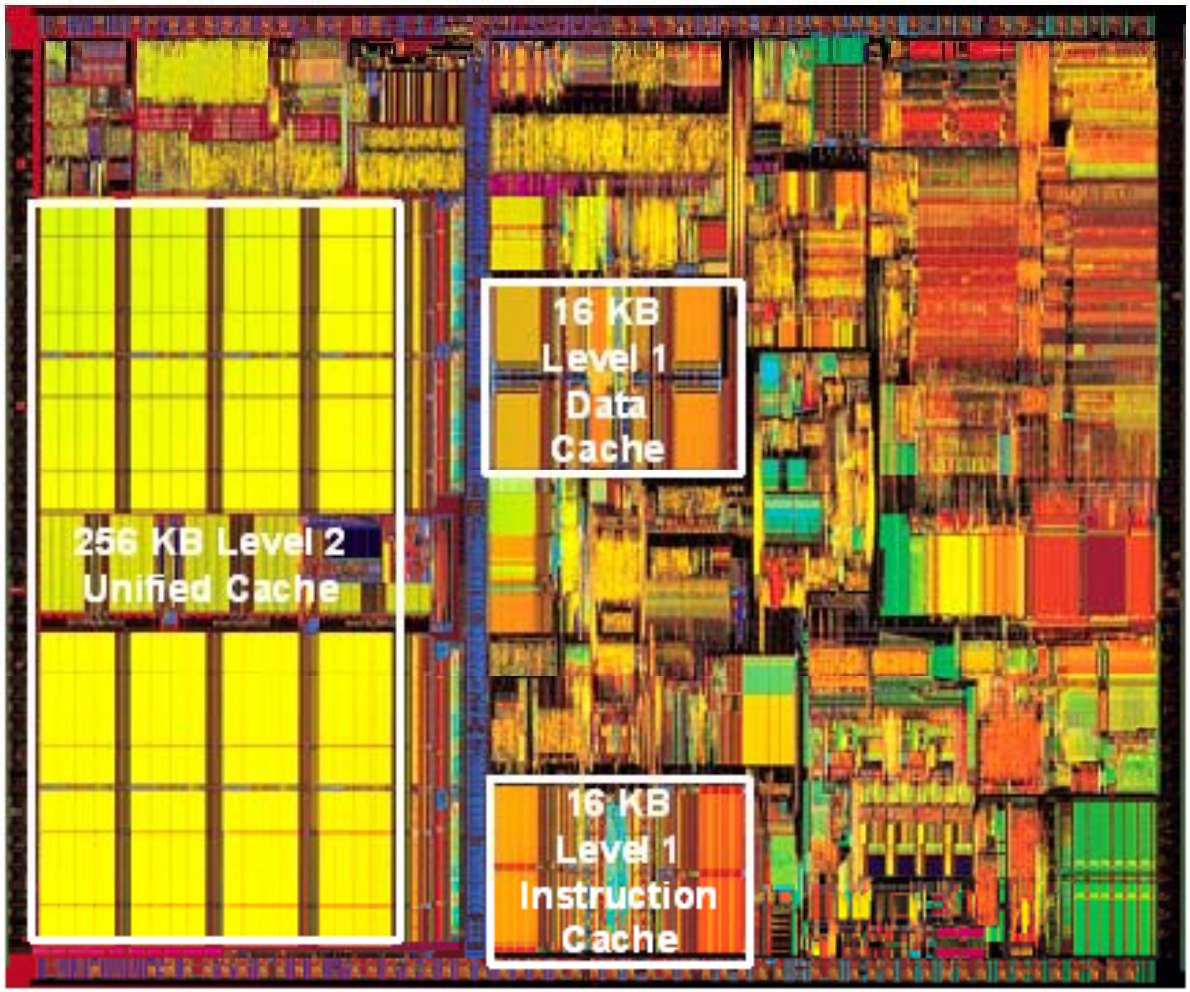
Cache Organization Summary /2

- **Cache Organization Improvements:**
  - Bigger caches reduce capacity misses
  - Greater associativity reduces conflict misses
  - Bigger blocks reduce compulsory misses
  - Bigger blocks increase conflict misses



Cache Organization Summary /3

- **Pentium III Memory:**



Multilevel Caches /1

- **Primary cache attached to CPU:**
  - Larger caches have lower miss rates, longer access times
  - Expand memory hierarchy to multiple levels of caches
  - Level-1 cache: small and fast (e.g., 16 KB, 1 cycle)
  - Level-2 cache services misses from primary cache
    - Larger and slower (e.g. 256 KB, 2-6 cycles)
    - Main memory services L-2 cache misses
  - Some high-end systems include L-3 cache too

Multilevel Caches /2

- **Multilevel Cache Example:**
  - Example details:
    - CPU base CPI = 1, clock rate = 4 GHz
    - Miss rate/instruction = 2%
    - Main memory access time = 100ns
  - **With just primary cache:**
    - Miss penalty = 100ns / 0.25ns = 400 cycles
    - CPI = 1 + 0.02 × 400 = 9
  - **Now add L-2 cache:**
    - Access time = 5ns
    - Compound miss rate to main memory = 0.5%
  - **Primary cache miss with L-2 hit:**
    - Penalty = 5ns / 0.25ns = 20 cycles
  - **Primary cache miss with L-2 miss:**
    - CPI = 1 + 0.02 × 20 + 0.005 × 400 = 3.4
    - Performance ratio improvrement = 9 / 3.4 = 2.6

Multilevel Caches /3

- **Multilevel Cache Elaboration:**
  - **Primary cache**
    - Focus on minimal hit time
  - **L-2 cache**
    - Focus on low miss rate to avoid main memory access
    - Hit time has less overall impact
  - **As a result:**
    - L-1 block size smaller than L-2 block size

Caching: Additional Considerations

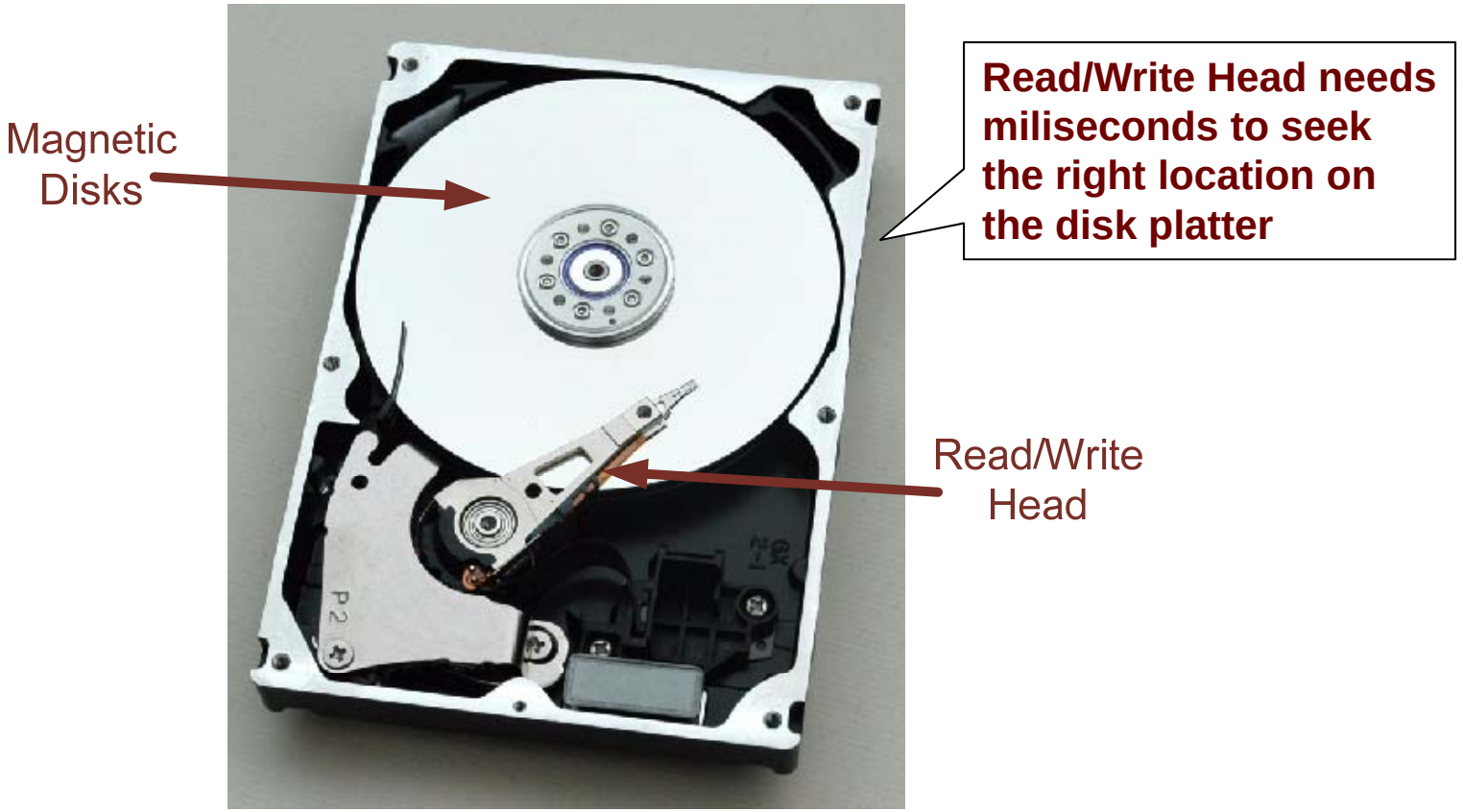
- **Interactions with Advanced CPUs:**
  - Out-of-order CPUs can execute instructions during cache miss
    - Pending data write stays in a load/store unit
    - Dependent instructions wait in the reservation stations
    - Independent instructions continue
  - Effect of miss depends on the program data flow
    - Can be much harder to analyze
    - Typically need to use system simulation to analyze
- **Interactions with Software:**
  - Misses depend on memory access patterns, which include
    - Algorithm behavior
    - Compiler optimization for memory access

Introduction to Virtual Memory /1

- **Virtual Memory (VM):**
  - Gives the illusion of bigger memory
  - Main memory (DRAM) acts as cache for hard disk
  - Managed jointly by the processor and the OS
  - Virtual memory is stored on a hard drive so, in comparison to DRAM and SRAM, it is Slow, Large, and Cheap
- **For example:**
  - A computer with 128 MB DRAM (called Physical Memory) could have 2 GB of available memory (called Virtual Memory)
  - The virtual memory is effectively stored on the hard disk, and physical memory stores a subset of the virtual memory

Introduction to Virtual Memory /2

- **Virtual Memory (VM) Visualized:**
  - Accessing virtual memory on a hard drive takes milliseconds, which are needed to seek the correct location on a disk

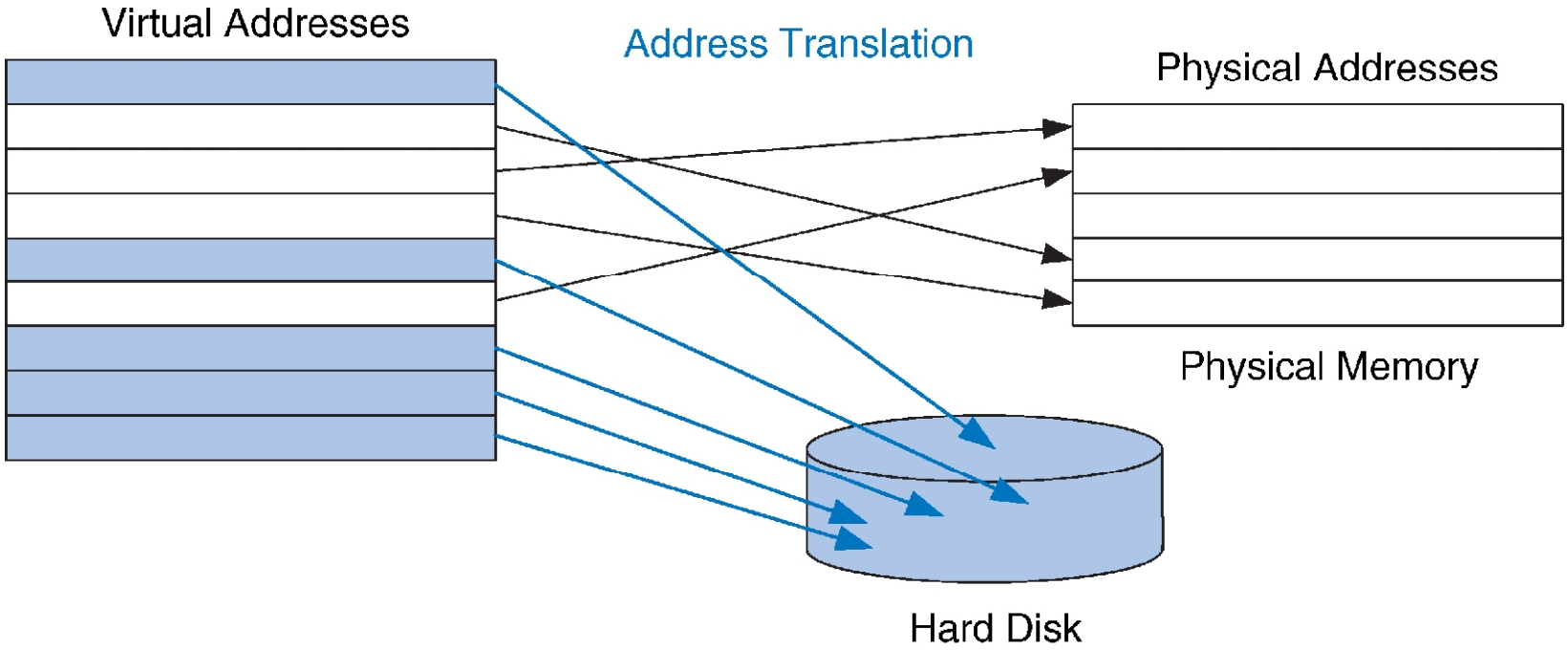


Introduction to Virtual Memory /3

- **Virtual addresses:**
  - Each program gets a private virtual address space to hold its frequently used code and data
  - Program's virtual address space is stored on a hard drive
  - Subset of the virtual address data is loaded in DRAM
  - CPU translates virtual addresses into physical addresses (DRAM addresses)
  - **Data not in DRAM is fetched from the hard drive**
    - **This is called page fault, and it is a very expensive but unavoidable operation to perform**
    - We can minimize the number of page faults by using specific placement and replacement algorithms

Introduction to Virtual Memory /4

- **Virtual to Physical Memory Address Translation:**



  - Most memory accesses hit in physical memory
  - But programs use the larger capacity of virtual memory, so some memory accesses end up in page faults

Introduction to Virtual Memory /5

- **Virtual Memory Protection:**
  - Each program has its own virtual to physical mapping
  - Two programs can use the same virtual address for different data
  - Programs do not need to be aware that the other programs are running at the same time

Cache	Virtual Memory
Block	Page
Block Size	Page Size
Block Offset	Page Offset
Miss	Page Fault
Tag	Virtual Page Number

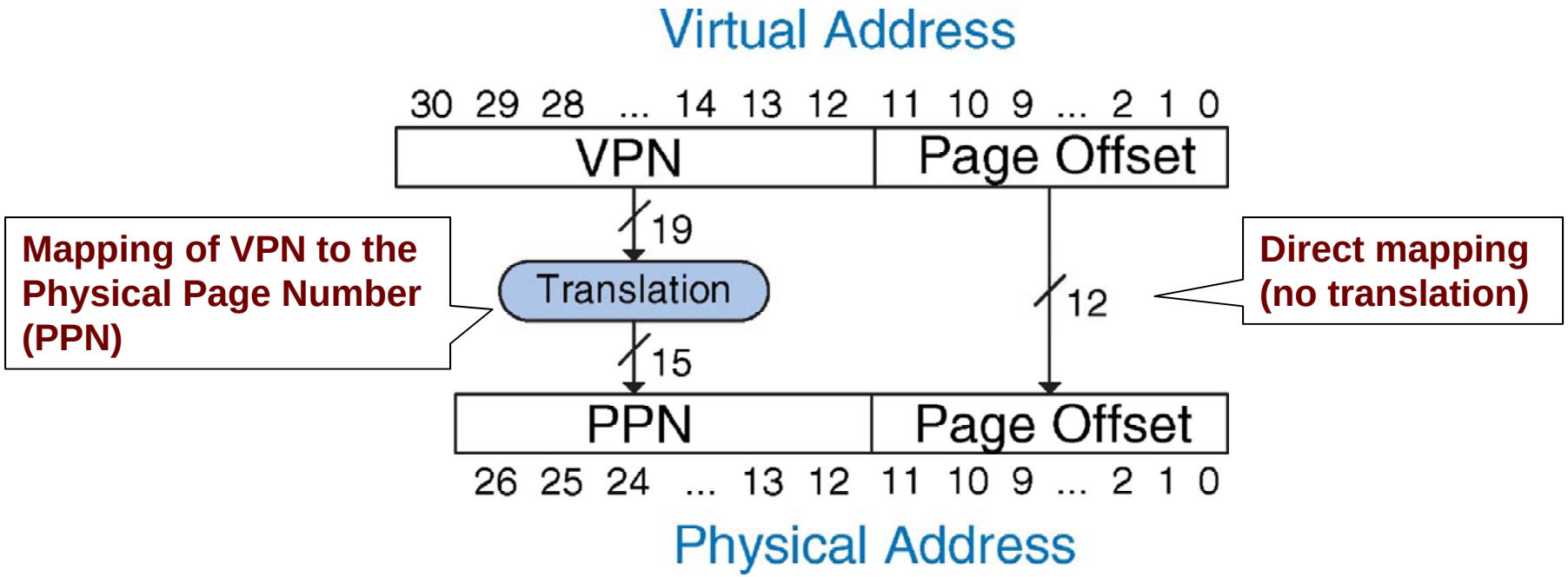
Typically 4K in size

Introduction to Virtual Memory /6

- **Virtual Memory Definitions Elaborated:**
  - **Page size:**
    - Amount of memory transferred from hard disk to DRAM at once
  - **Address translation:**
    - Determining physical address from virtual address
    - To minimize page faults (and avoid page faults caused by conflicts), any virtual page can map to any physical page
    - **That is, physical memory serves as a "fully-associative cache" for virtual memory**
    - Recall fully-associative cache: One set for all blocks in cache
  - **Page table:**
    - Lookup table used to translate virtual addresses to physical addresses
    - **A practical virtual memory has too many pages to provide a comparator for each page**
    - A table is used to map the addresses instead

Introduction to Virtual Memory /7

- **Address Translation:**
  - Each virtual address divided into two parts
  - **The top (most-significant) bits represent the Virtual Page Number (VPN)**
  - **The bottom (least-significant) bits represent the word within that page, and are called the Page Offset**

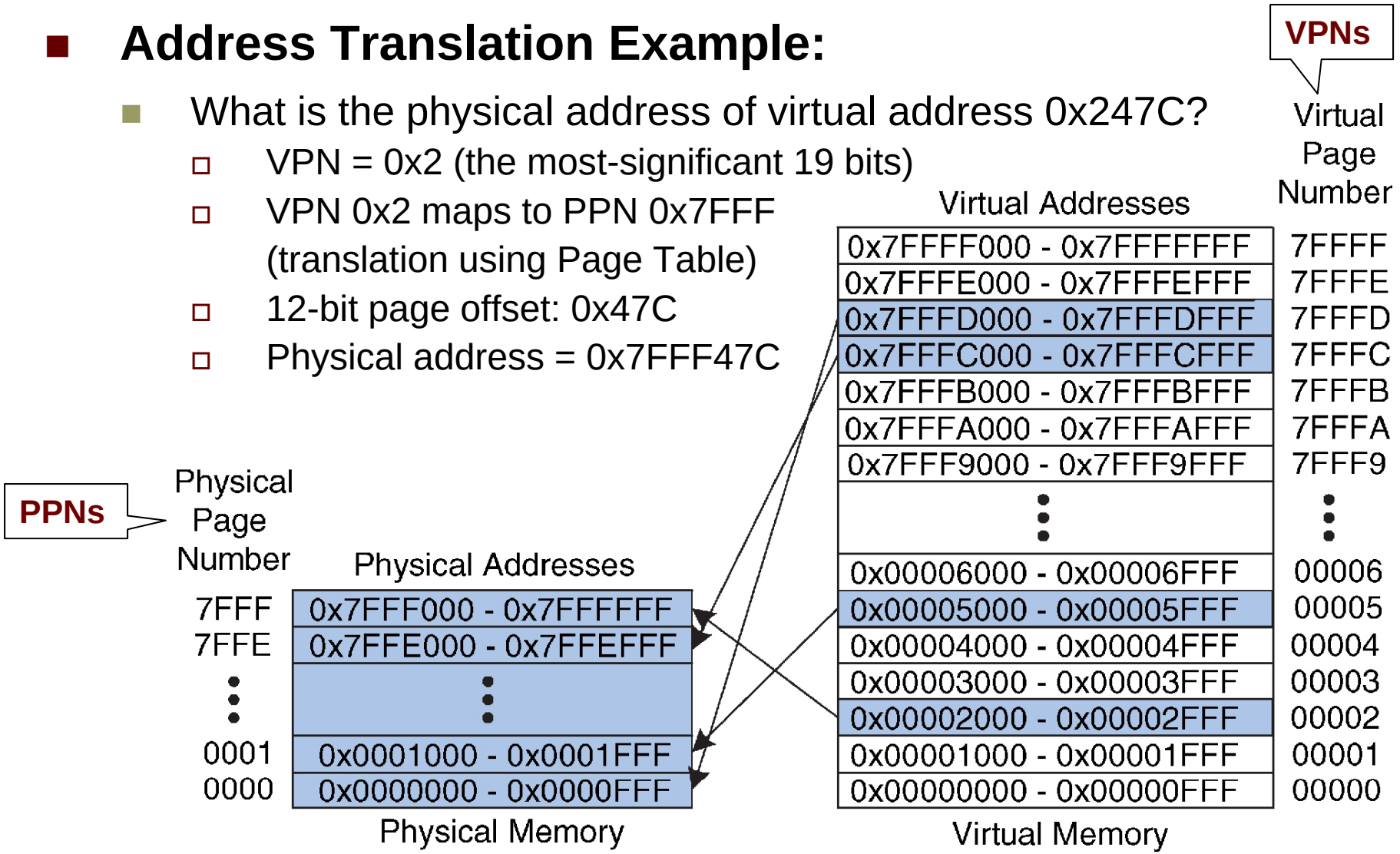


Introduction to Virtual Memory /8

- **Address Translation Example:**
  - Virtual memory size: 2 GB = 2^{31} bytes (2,147,483,648 bytes)
    - In MIPS, the least-significant 31 bits are used; 32nd bit is 0
  - Physical memory size: 128 MB = 2^{27} bytes (134,217,728 bytes)
    - In MIPS, the least-significant 27 bits are used; other 5 are 0s
  - Page size: 4 KB = 2^{12} bytes (4096 bytes)
  - **System organization based on the above:**
    - Virtual address: **31** bits
    - Physical address: **27** bits
    - Page offset: **12** bits
    - Number of Virtual pages = $2^{31}/2^{12}$ = $\mathbf{2^{19}}$ (VPN = 19 bits)
    - Number of Physical pages = $2^{27}/2^{12}$ = $\mathbf{2^{15}}$ (PPN = 15 bits)
    - Hence, physical memory can only hold **1/16th** of virtual memory at one time; the remainder of the virtual memory is on the hard disk

Introduction to Virtual Memory /9

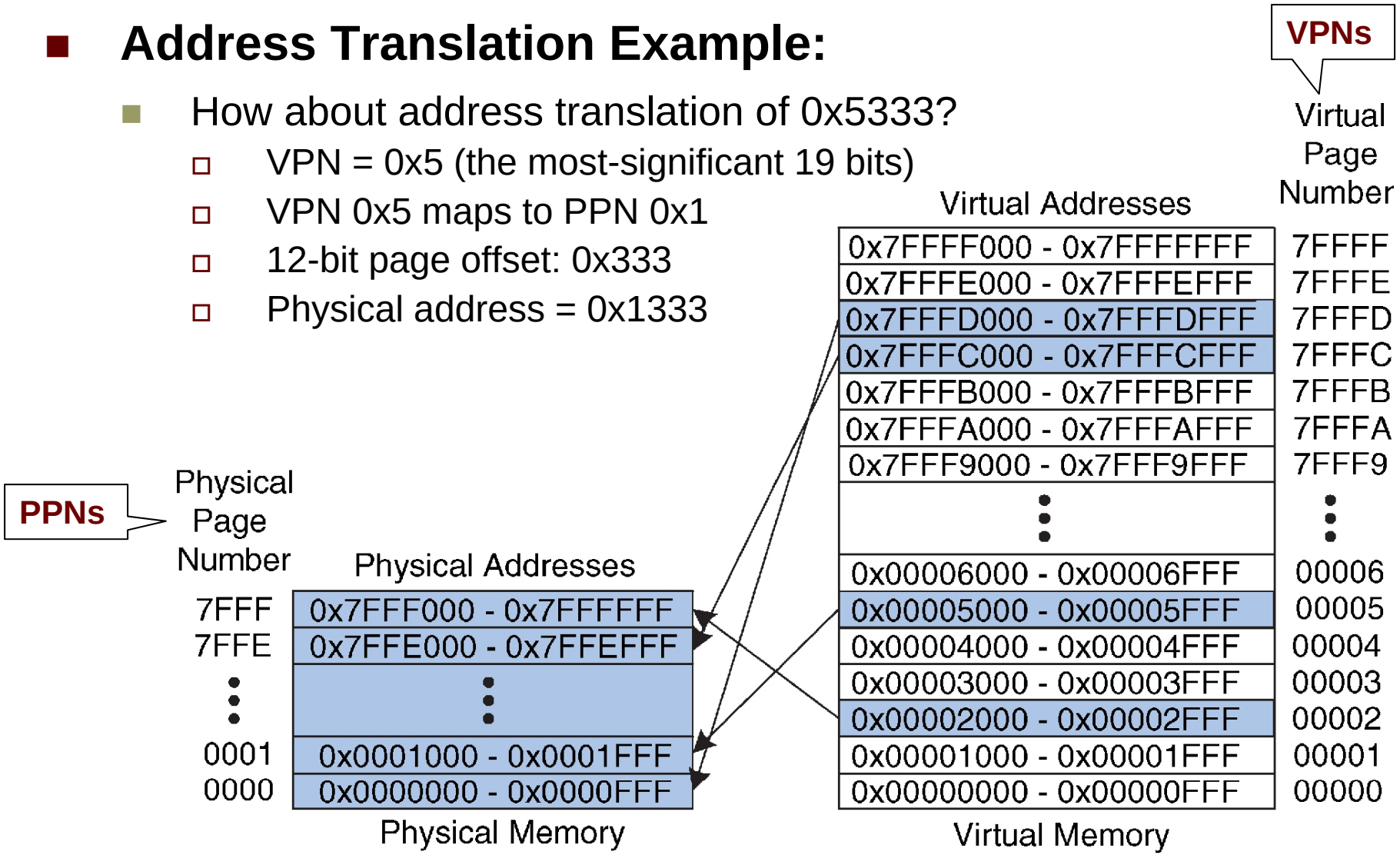
- **Address Translation Example:**
  - What is the physical address of virtual address 0x247C?
    - VPN = 0x2 (the most-significant 19 bits)
    - VPN 0x2 maps to PPN 0x7FFF (translation using Page Table)
    - 12-bit page offset: 0x47C
    - Physical address = 0x7FFF47C



(taken from the Harris and Harris textbook; see Chapter 8 for details)

Introduction to Virtual Memory /10

- **Address Translation Example:**
  - How about address translation of 0x5333?
    - VPN = 0x5 (the most-significant 19 bits)
    - VPN 0x5 maps to PPN 0x1
    - 12-bit page offset: 0x333
    - Physical address = 0x1333



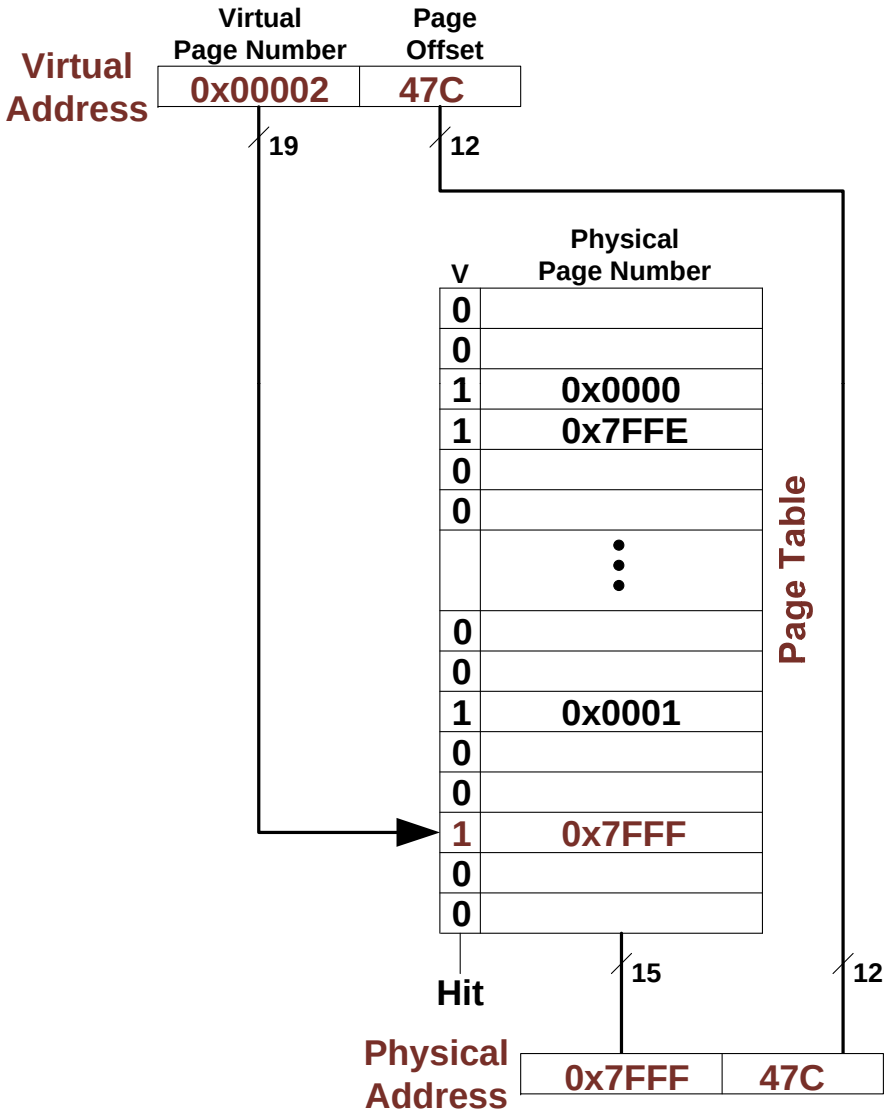
(taken from the Harris and Harris textbook; see Chapter 8 for details)

Virtual Memory Address Translation /1

- **Page Table:**
  - Used to translate VPN to PPN during virtual memory access
  - **One entry for each virtual page**
  - **Entry fields:**
    - **Valid bit:** 1 if the page is in physical memory; if 0, a page fault occurs and the block is loaded from hard disk
    - **Physical page number:** Where the page is located
  - **Each load or store instruction requires one page table access and one physical memory access**
    - We shall use a buffer (called Translation Lookaside Buffer (TLB)) to speedup these operations

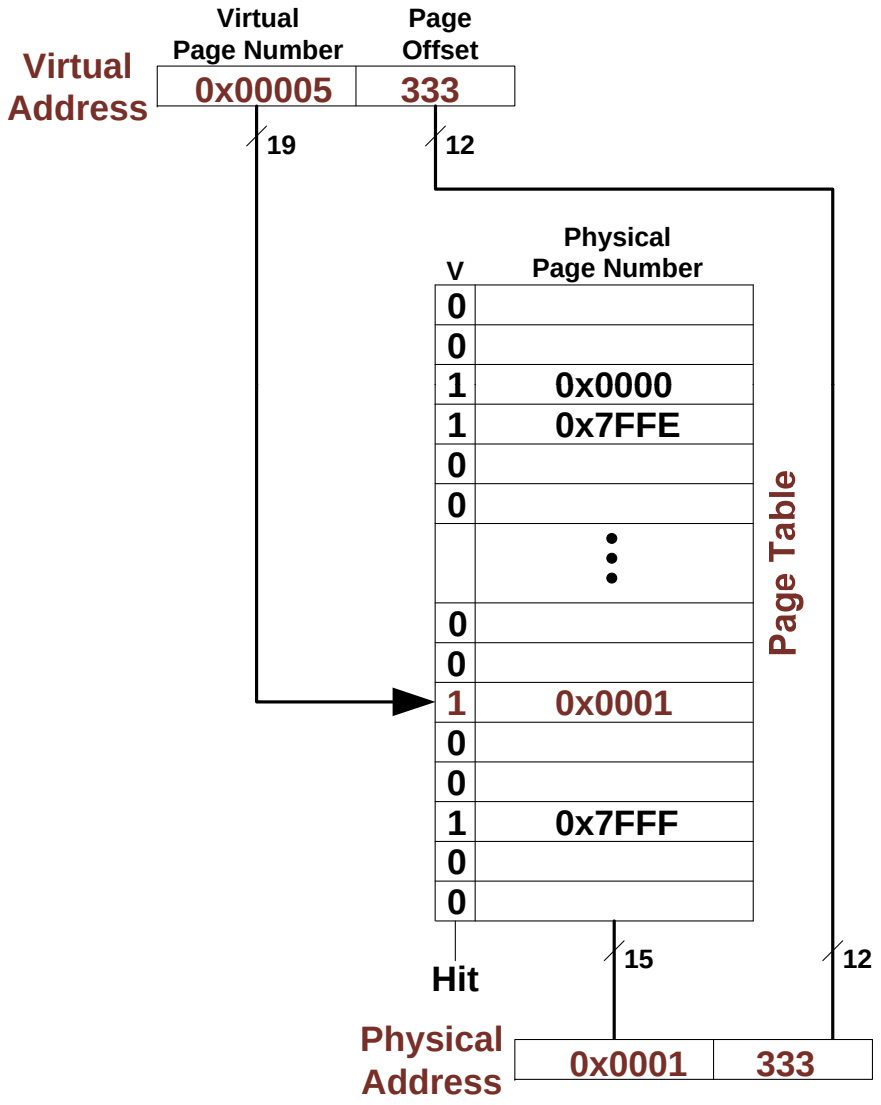
Virtual Memory Address Translation /2

- **Page Table Visualized:**
  - Virtual Page Number (VPN) is an index of the entry in the Page Table
  - VPN indicates the location inside the Page Table
    - In the given example, it is the 3nd location counted from the bottom
  - If the Hit = 1, then the indexed location stores the PPN; if not, a page fault occurs and the location has to be paged from disk
    - In the given example, PPN is 0x7FFF
  - **The translated physical address is the stored PPN and the Page Offset of the VM address**



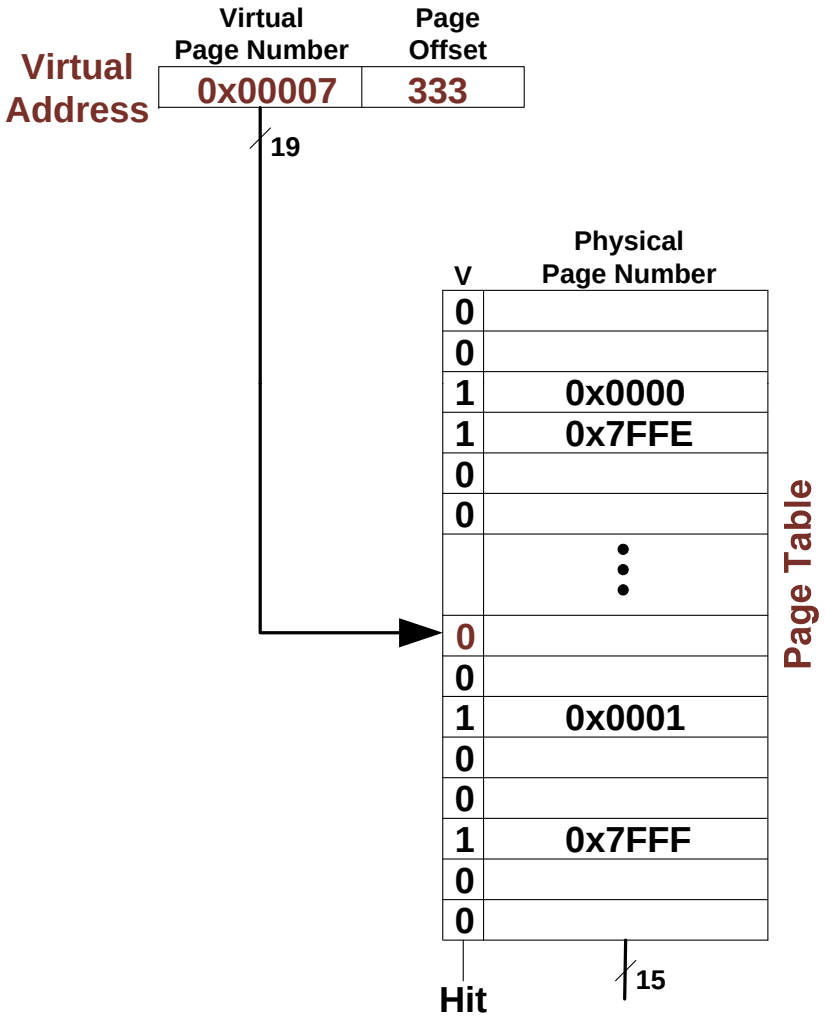
Virtual Memory Address Translation /3

- **Page Table Example:**
  - Translate 0x5333 virtual address into the matching physical address
  - VPN value of 0x5 is used as the index to retrieve the 6th record (counted from 0) from Page Table
  - That location stores PPN value of 0x1, which is then used as the base of the physical address
  - The final physical address is 0x1333, where Page Offset of 333 is appended



Virtual Memory Address Translation /4

- **Page Table Example:**
  - Translate 0x7333 virtual address into the matching physical address
  - VPN value of 0x7 is used as the index to retrieve the 8th record (counted from 0) from Page Table
  - That location has the Hit value of 0, so there is no valid PPN address stored in there
  - **Hence, a page fault occurs and this virtual memory page has to be page from disk into physical memory**



Virtual Memory Address Translation /5

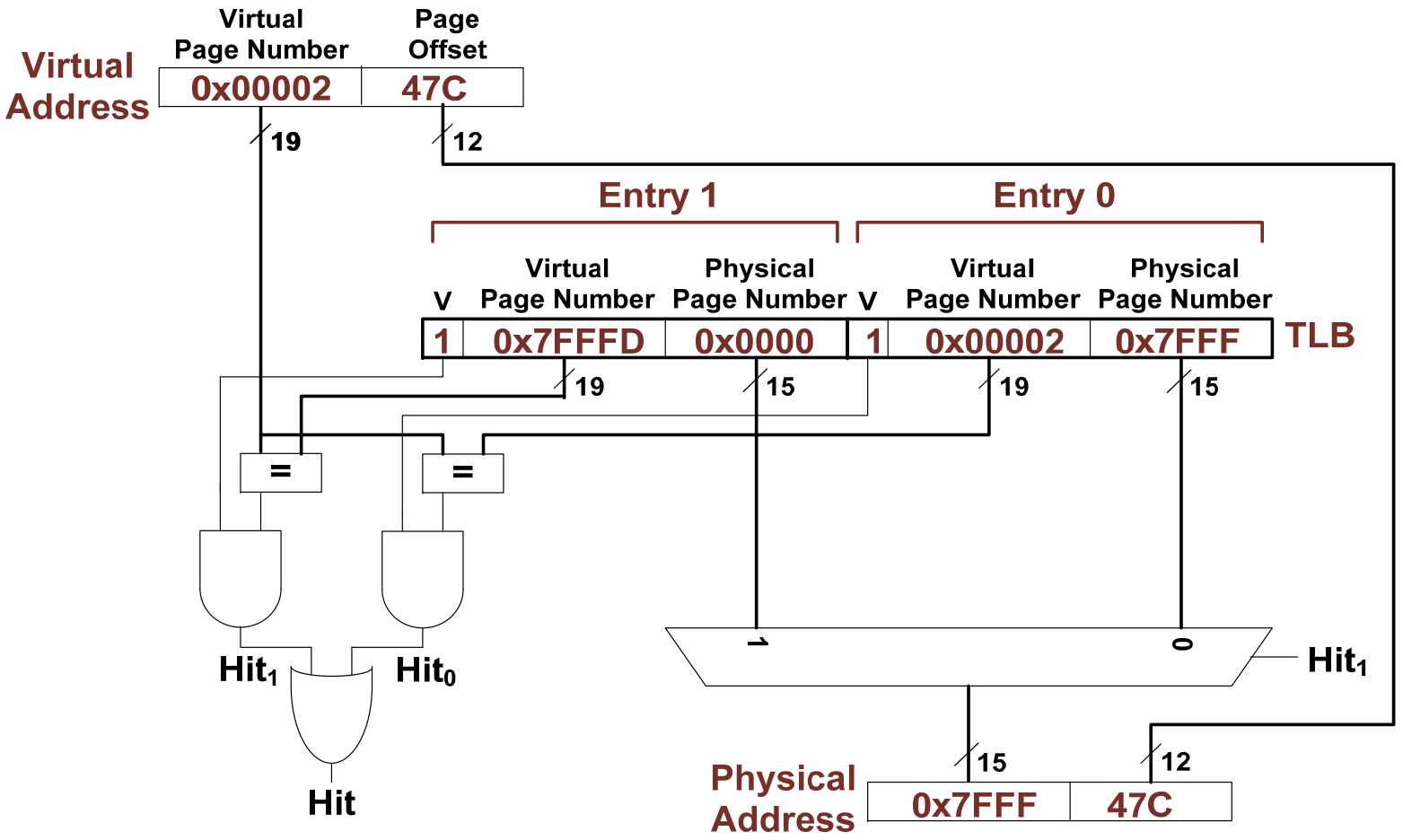
- **Page Table Elaborated:**
  - Page Table is large and usually located in physical memory
  - OS uses a special register, called Page Table Register, to store the base address of a Page Table for a program
  - Each program uses its own Page Table, and only those physical addresses that are given to it by the operating system
  - To load or store data from memory, this requires
    - One memory access for address translation (page table read)
    - One memory access to access data (after translation)
- **Translation Lookaside Buffer (TLB):**
  - Small cache of the most recently run translations
  - **The main goal of TLB is to reduces the number of memory accesses for <u>most</u> load and store operations from 2 to 1**

Virtual Memory Address Translation /6

- **Translation Lookaside Buffer (TLB):**
  - Small cache of the most recently run translations
  - Exploits the temporal and spatial locality of page accesses
  - Each page is relatively large in size, so consecutive load and store operations are likely to access the same page
  - TLB is small in size, as it typically includes 16-512 entries
  - It is fully associative, with hit rates greater than 99%
  - It is also accessed in less than 1 cycle
  - **The main goal of TLB is to reduces the number of memory accesses for <u>most</u> load and store operations from 2 to 1**

Virtual Memory Address Translation /7

- **Translation Lookaside Buffer (TLB) Visualized:**
  - Uses a cache-like structure, where the VPN value is used as the tag value, and the PPN value is used as the data value



Virtual Memory Summary

- Virtual memory increases memory capacity
- A subset of virtual memory pages is in physical memory
- Page table maps virtual pages to physical pages using address translation
- Different page tables for different programs provides memory protection
- A TLB speeds up address translation

Food for Thought

- **Download and Read Assignment #4 Specifications**
- **Read:**
  - Chapter 7 from the Course Notes
    - Review the material discussed in the lecture notes in more detail
    - Our course schedule follows the material in the Course Notes
  - Recommended: Chapter 8 from the Harris and Harris textbook

  OR
  - Recommended: Chapter 5 from the course textbook