

# Chapter 8

## ***Digital Design and Computer Architecture: ARM<sup>®</sup> Edition***

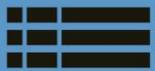
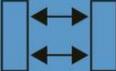
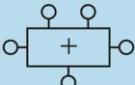
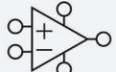
---

Sarah L. Harris and David Money Harris



# Chapter 8 :: Topics

- Introduction
- Memory System Performance Analysis
- Caches
- Virtual Memory
- Memory-Mapped I/O
- Summary

Application Software	<code>&gt;"hello world!"</code>
Operating Systems	
Architecture	
Micro-architecture	
Logic	
Digital Circuits	
Analog Circuits	
Devices	
Physics	

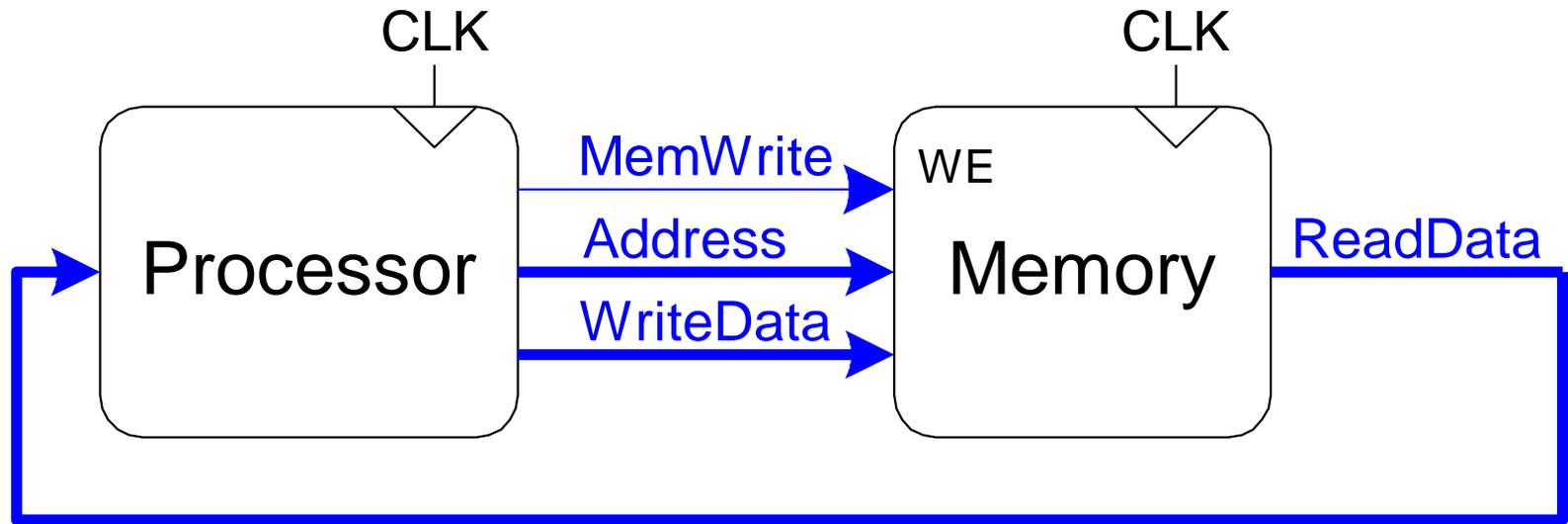


# Introduction

Computer performance depends on:

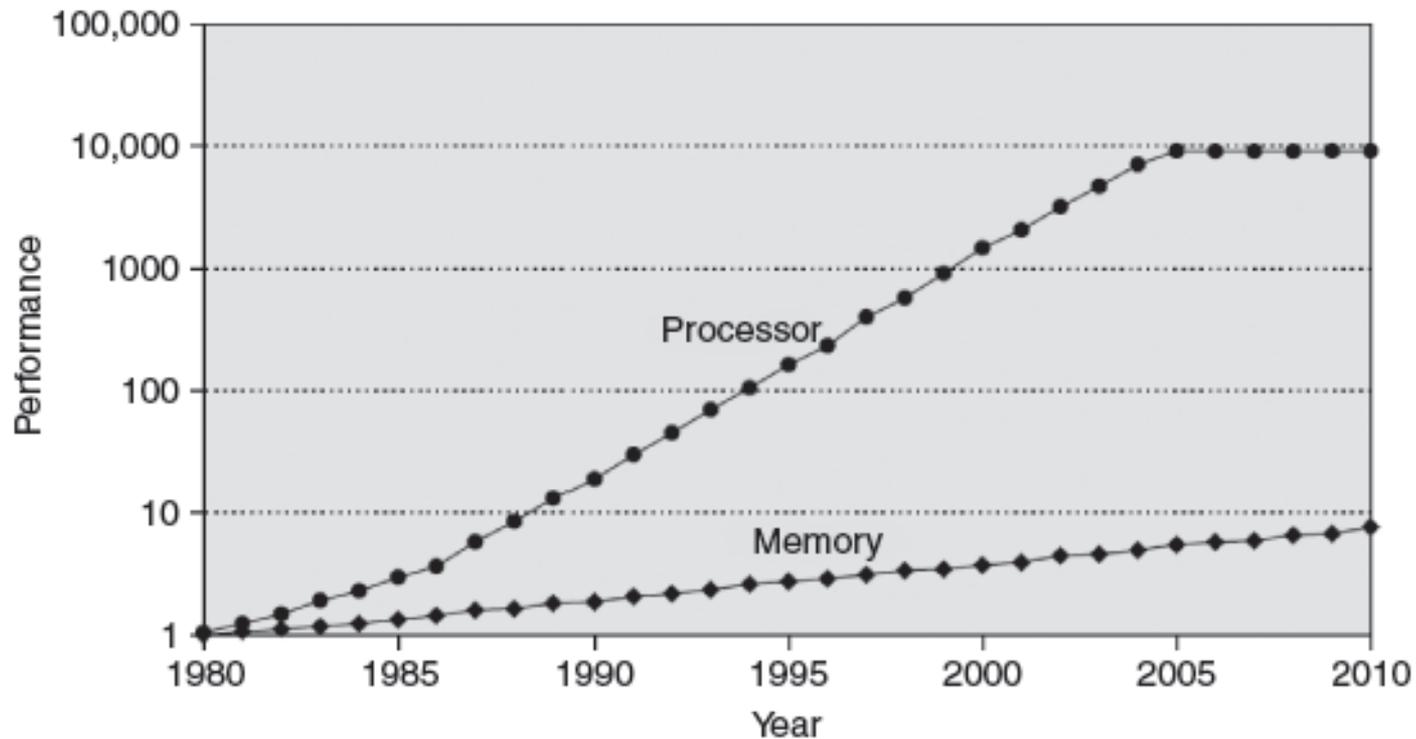
- Processor performance
- Memory system performance

## Memory Interface



# Processor-Memory Gap

In prior chapters, assumed memory access takes 1 clock cycle – but hasn't been true since the 1980's



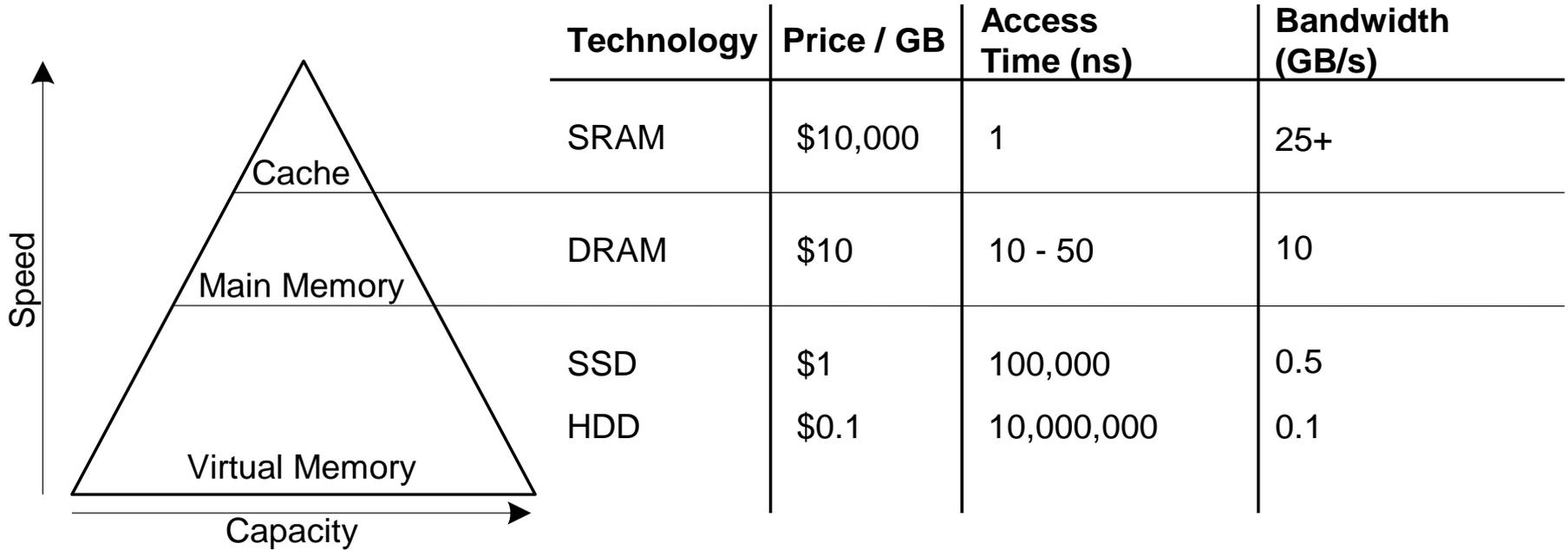
# Memory System Challenge

- Make memory system appear as fast as processor
- Use hierarchy of memories
- Ideal memory:
  - Fast
  - Cheap (inexpensive)
  - Large (capacity)

**But can only choose two!**



# Memory Hierarchy



# Locality

Exploit locality to make memory accesses fast

- **Temporal Locality:**

- Locality in time
- If data 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 used recently, likely to use nearby data soon
- **How to exploit:** when access data, bring nearby data into higher levels of memory hierarchy too



# Memory Performance

- **Hit:** data found in that level of memory hierarchy
- **Miss:** data not found (must go to next level)

**Hit Rate** = # hits / # memory accesses  
= 1 – Miss Rate

**Miss Rate** = # misses / # memory accesses  
= 1 – Hit Rate

- **Average memory access time (AMAT):** average time for processor to access data

$$\mathbf{AMAT} = t_{\text{cache}} + MR_{\text{cache}}[t_{MM} + MR_{MM}(t_{VM})]$$



# Memory Performance Example 1

- A program has 2,000 loads and stores
- 1,250 of these data values in cache
- Rest supplied by other levels of memory hierarchy
- **What are the hit and miss rates for the cache?**



# Memory Performance Example 1

- A program has 2,000 loads and stores
- 1,250 of these data values in cache
- Rest supplied by other levels of memory hierarchy
- **What are the hit and miss rates for the cache?**

$$\text{Hit Rate} = 1250/2000 = \mathbf{0.625}$$

$$\text{Miss Rate} = 750/2000 = \mathbf{0.375} = 1 - \text{Hit Rate}$$



# Memory Performance Example 2

- Suppose processor has 2 levels of hierarchy: cache and main memory
- $t_{\text{cache}} = 1$  cycle,  $t_{MM} = 100$  cycles
- **What is the AMAT of the program from Example 1?**



# Memory Performance Example 2

- Suppose processor has 2 levels of hierarchy: cache and main memory
- $t_{\text{cache}} = 1$  cycle,  $t_{MM} = 100$  cycles
- **What is the AMAT of the program from Example 1?**

$$\begin{aligned}\text{AMAT} &= t_{\text{cache}} + MR_{\text{cache}}(t_{MM}) \\ &= [1 + 0.375(100)] \text{ cycles} \\ &= \mathbf{38.5 \text{ cycles}}\end{aligned}$$



# Gene Amdahl, 1922-

- **Amdahl's Law:** the effort spent increasing the performance of a subsystem is wasted unless the subsystem affects a large percentage of overall performance
- Co-founded 3 companies, including one called Amdahl Corporation in 1970



# Cache

- Highest level in memory hierarchy
- Fast (typically  $\sim 1$  cycle access time)
- Ideally supplies most data to processor
- Usually holds most recently accessed data



# Cache Design Questions

- What data is held in the cache?
- How is data found?
- What data is replaced?

**Focus on data loads, but stores follow same principles**



# 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 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:  $B = C/b$
- **Degree of associativity ( $N$ ):**
  - number of blocks in a set
- **Number of sets ( $S = B/N$ ):**
  - each memory address maps to exactly one cache set



# 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
  - **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 ( $B = 8$ )



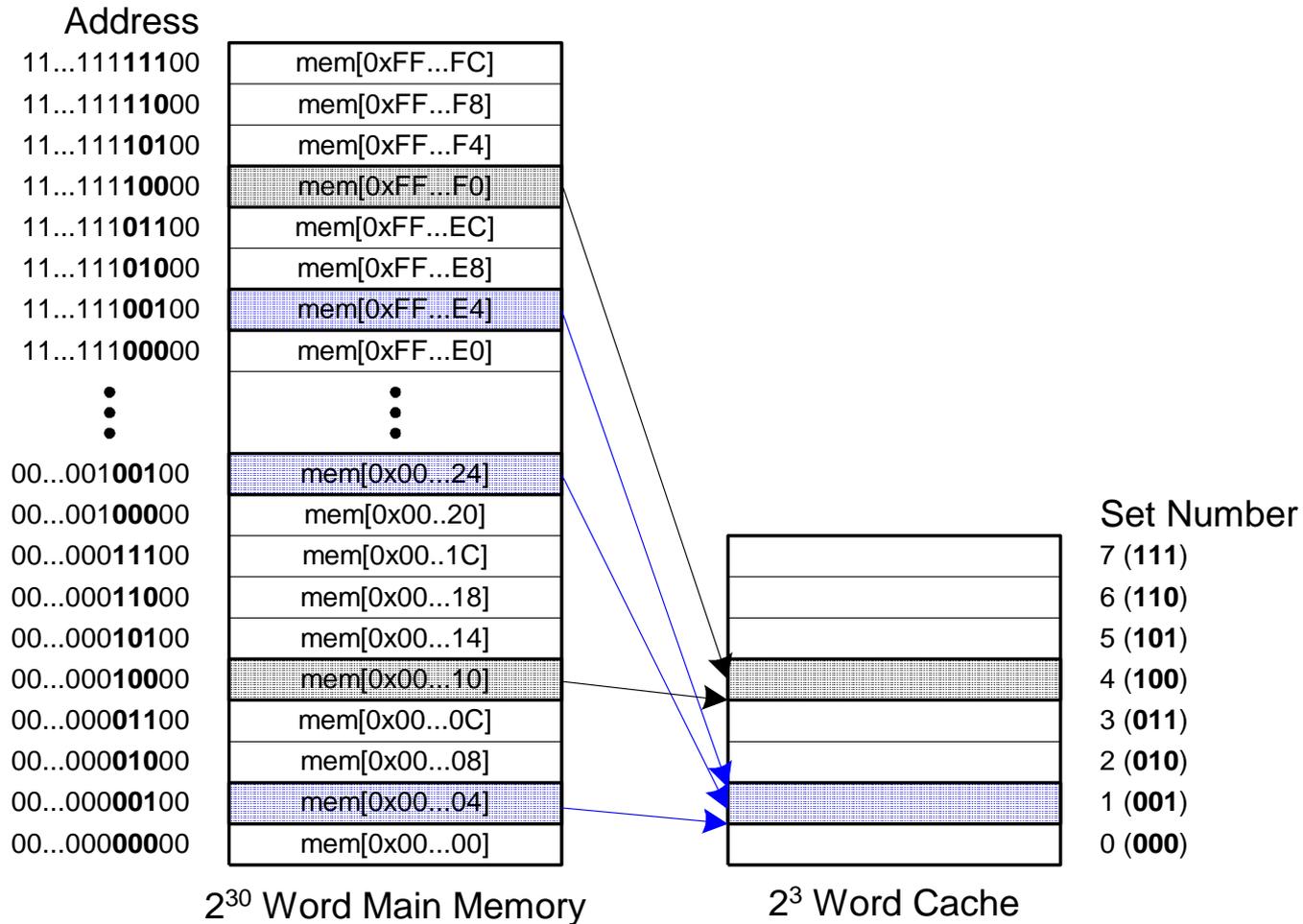
# Example Cache Parameters

- $C = 8$  words (capacity)
- $b = 1$  word (block size)
- So,  $B = 8$  (# of blocks)

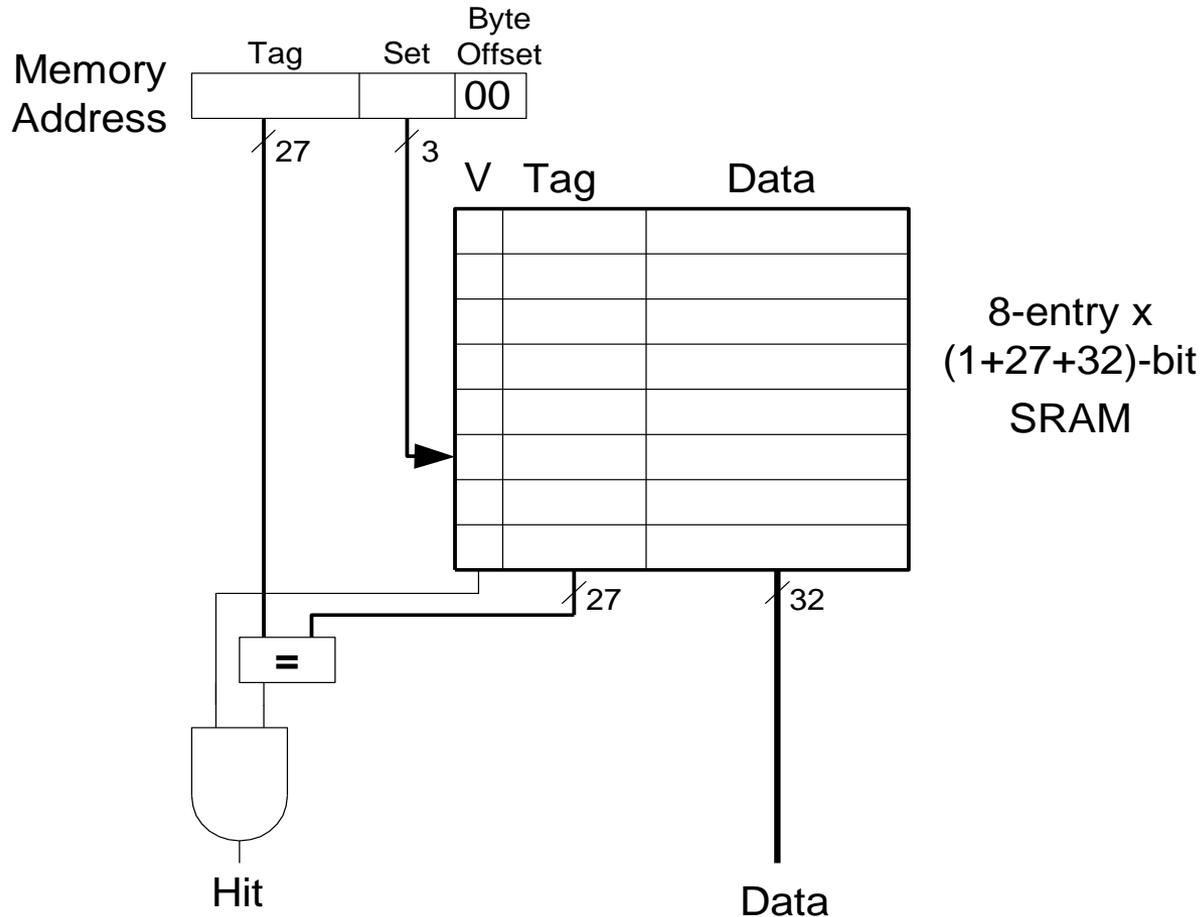
Ridiculously small, but will illustrate organizations



# Direct Mapped Cache



# Direct Mapped Cache Hardware



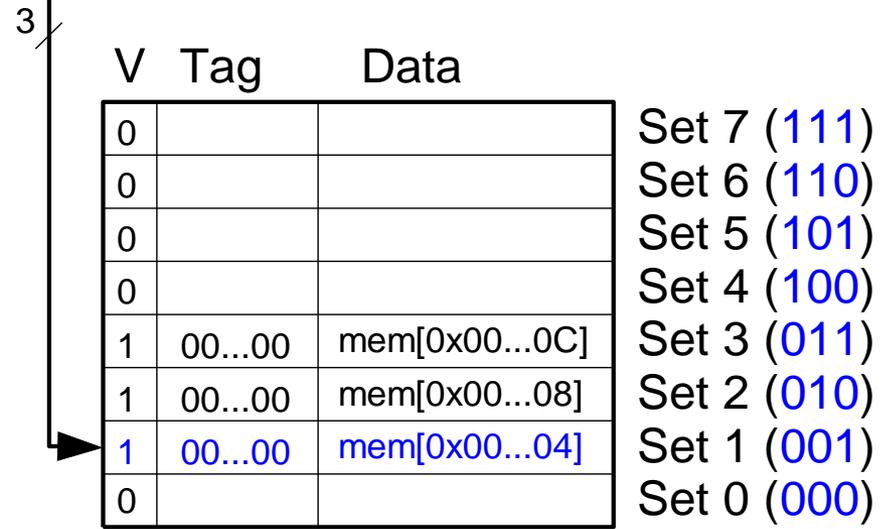
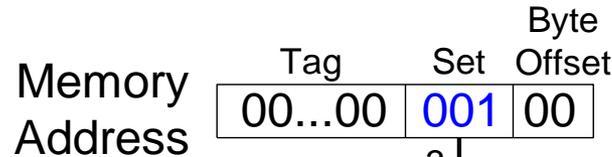
# Direct Mapped Cache Performance

## ARM Assembly Code

```

MOV R0, #5
MOV R1, #0
LOOP  CMP R0, #0
      BEQ DONE
      LDR R2, [R1, #4]
      LDR R3, [R1, #12]
      LDR R4, [R1, #8]
      SUB R0, R0, #1
      B LOOP
DONE

```



**Miss Rate = ?**

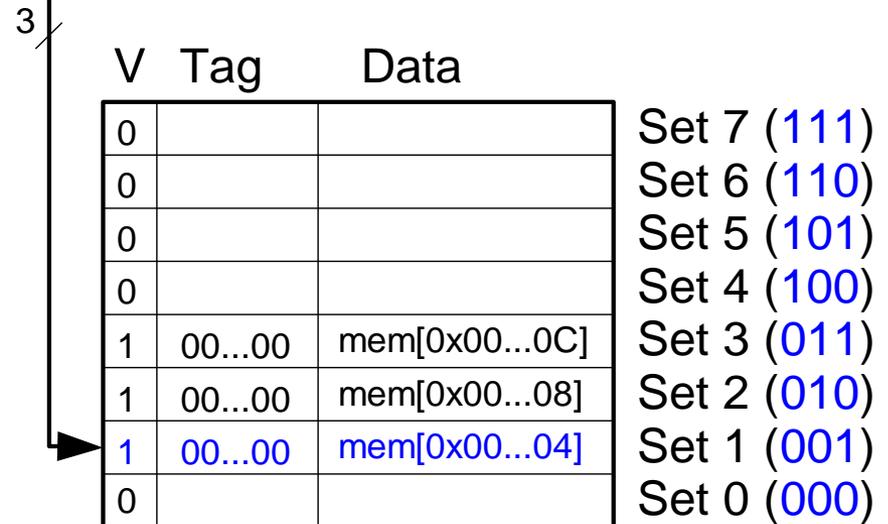
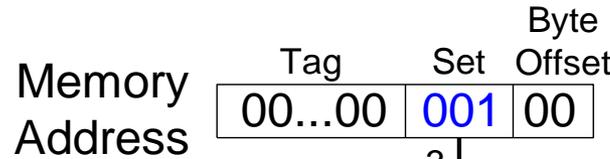


# Direct Mapped Cache Performance

## ARM Assembly Code

```

MOV R0, #5
MOV R1, #0
LOOP  CMP R0, #0
      BEQ DONE
      LDR R2, [R1, #4]
      LDR R3, [R1, #12]
      LDR R4, [R1, #8]
      SUB R0, R0, #1
      B  LOOP
DONE
    
```



**Miss Rate = 3/15  
= 20%**

**Temporal Locality  
Compulsory Misses**

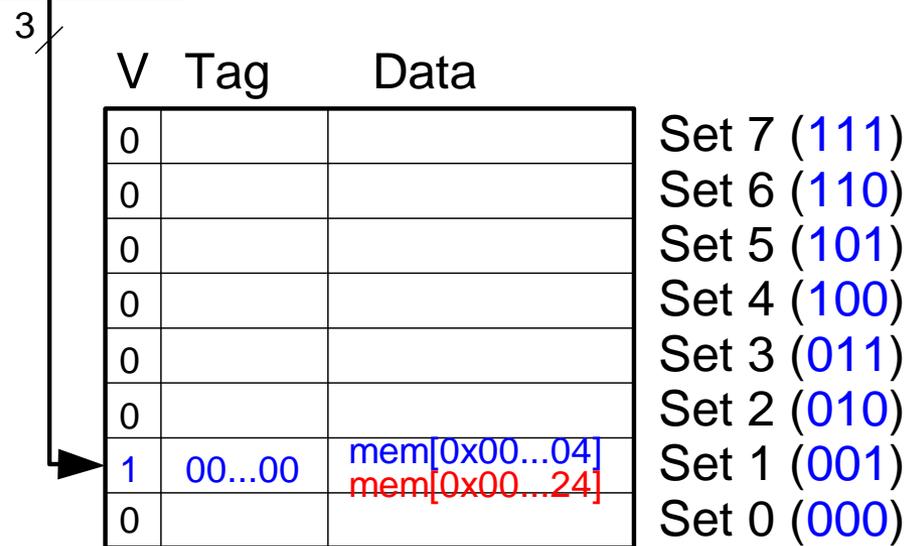
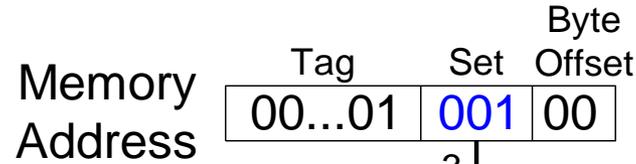


# Direct Mapped Cache: Conflict

## ARM Assembly Code

```

MOV R0, #5
MOV R1, #0
LOOP  CMP R0, #0
      BEQ DONE
      LDR R2, [R1, #0x4]
      LDR R3, [R1, #0x24]
      SUB R0, R0, #1
      B LOOP
DONE
    
```



**Miss Rate = ?**

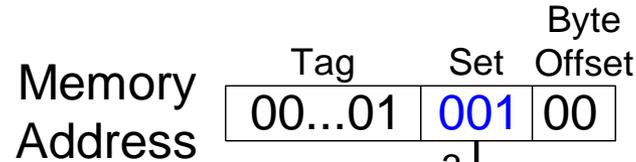


# Direct Mapped Cache: Conflict

## ARM Assembly Code

```

MOV R0, #5
MOV R1, #0
LOOP  CMP R0, #0
      BEQ DONE
      LDR R2, [R1, #0x4]
      LDR R3, [R1, #0x24]
      SUB R0, R0, #1
      B LOOP
DONE
    
```



3

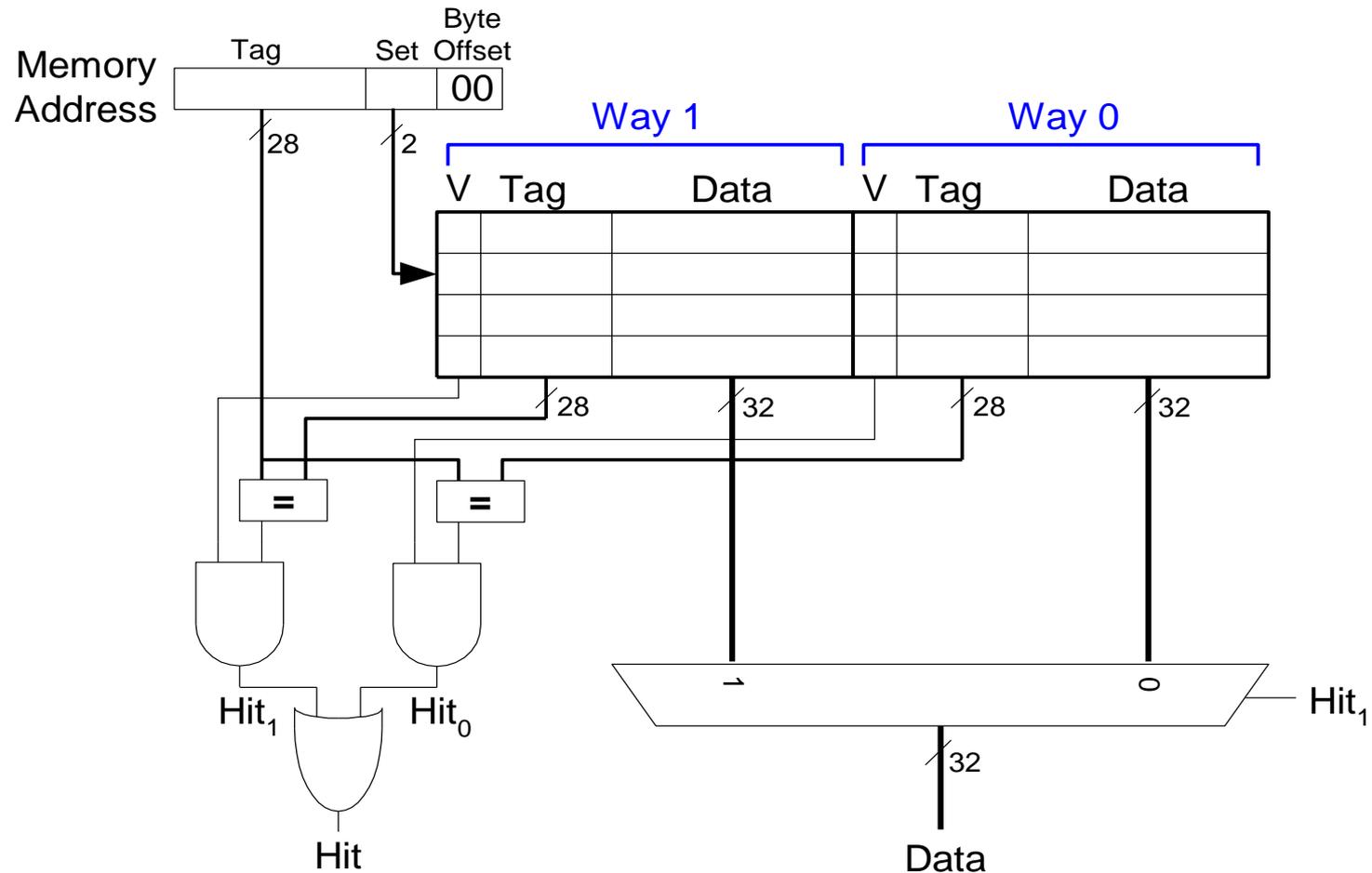
V	Tag	Data	
0			Set 7 (111)
0			Set 6 (110)
0			Set 5 (101)
0			Set 4 (100)
0			Set 3 (011)
0			Set 2 (010)
1	00...00	mem[0x00...04] mem[0x00...24]	Set 1 (001)
0			Set 0 (000)

**Miss Rate = 10/10**  
**= 100%**

**Conflict Misses**



# N-Way Set Associative Cache



# N-Way Set Associative Performance

## ARM Assembly Code

```
MOV R0, #5
MOV R1, #0
LOOP  CMP R0, 0
      BEQ DONE
      LDR R2, [R1, #0x4]
      LDR R3, [R1, #0x24]
      SUB R0, R0, #1
      B LOOP
DONE
```

**Miss Rate = ?**

Way 1			Way 0			
V	Tag	Data	V	Tag	Data	
0			0			Set 3
0			0			Set 2
0			0			Set 1
0			0			Set 0



# N-Way Set Associative Performance

## ARM Assembly Code

```

MOV R0, #5
MOV R1, #0
LOOP  CMP R0, 0
      BEQ DONE
      LDR R2, [R1, #0x4]
      LDR R3, [R1, #0x24]
      SUB R0, R0, #1
      B LOOP
DONE
    
```

**Miss Rate = 2/10  
= 20%**

**Associativity reduces  
conflict misses**

Way 1			Way 0		
V	Tag	Data	V	Tag	Data
0			0		
0			0		
1	00...10	mem[0x00...24]	1	00...00	mem[0x00...04]
0			0		

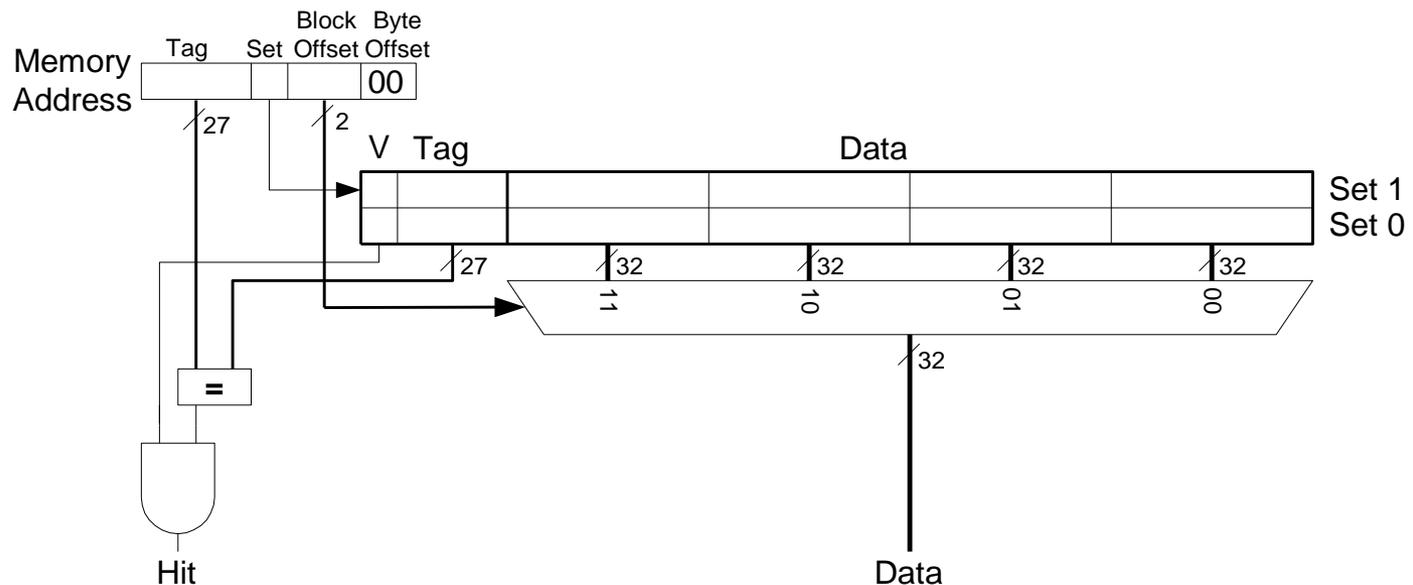
Set 3  
Set 2  
Set 1  
Set 0



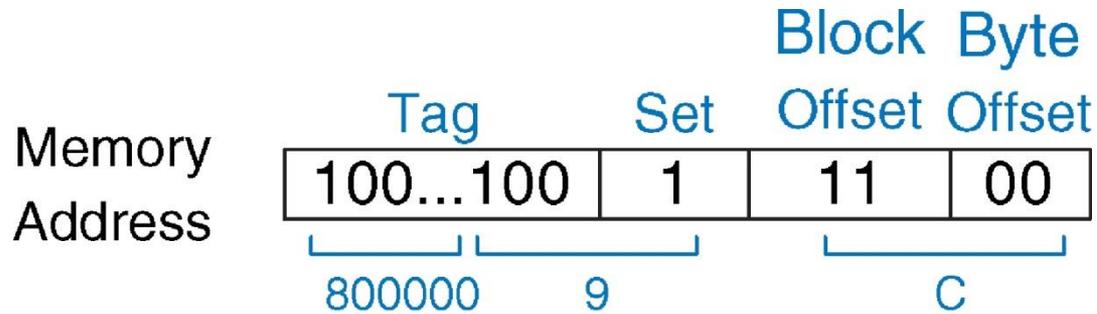


# Spatial Locality?

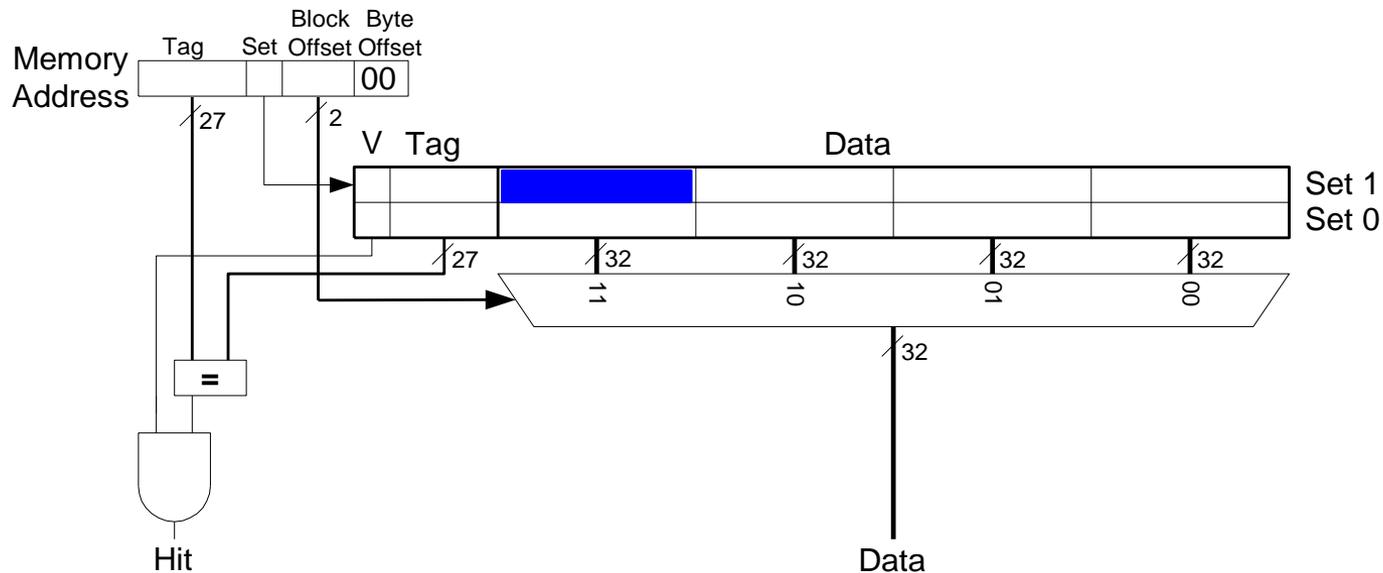
- Increase block size:
  - Block size,  $b = 4$  words
  - $C = 8$  words
  - Direct mapped (1 block per set)
  - Number of blocks,  $B = 2$  ( $C/b = 8/4 = 2$ )



# Cache with Larger Block Size



© 2007 Elsevier, Inc. All rights reserved



# Direct Mapped Cache Performance

## ARM assembly code

```
MOV R0, #5
MOV R1, #0
LOOP  CMP R0, 0
      BEQ DONE
      LDR R2, [R1, #4]
      LDR R3, [R1, #12]
      LDR R4, [R1, #8]
      SUB R0, R0, #1
      B   LOOP
DONE
```

**Miss Rate = ?**



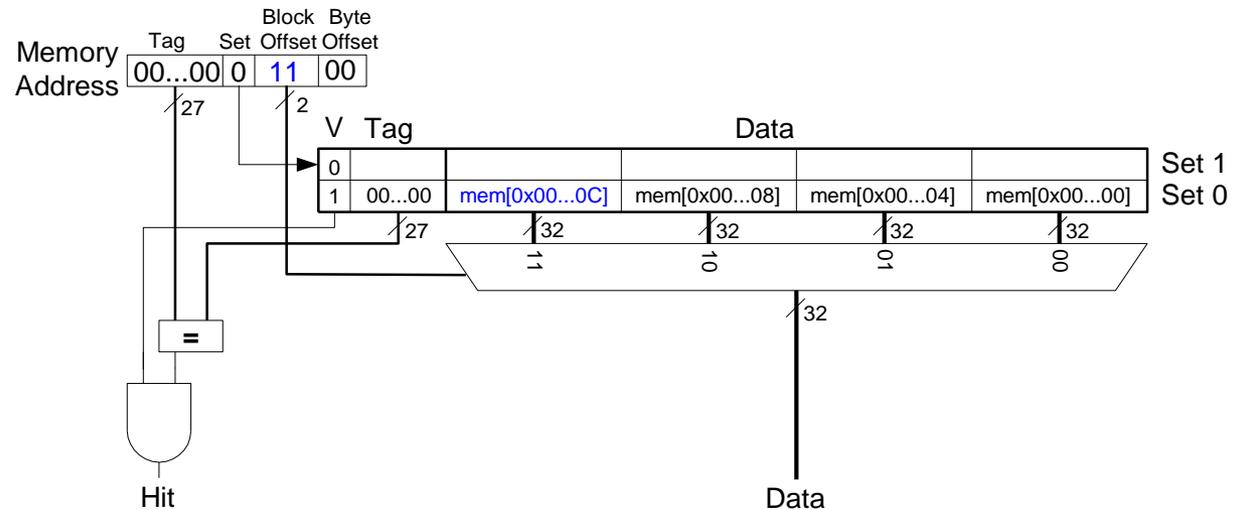
# Direct Mapped Cache Performance

## ARM assembly code

```
MOV R0, #5
MOV R1, #0
LOOP  CMP R0, 0
      BEQ DONE
      LDR R2, [R1, #4]
      LDR R3, [R1, #12]
      LDR R4, [R1, #8]
      SUB R0, R0, #1
      B LOOP
DONE
```

**Miss Rate = 1/15**  
**= 6.67%**

**Larger blocks**  
**reduce compulsory misses**  
**through spatial locality**



# Cache Organization Recap

- Capacity:  $C$
- Block size:  $b$
- Number of blocks in cache:  $B = C/b$
- Number of blocks in a set:  $N$
- 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



# Capacity Misses

- Cache is too small to hold all data of interest at once
- If cache full: program accesses data X & evicts data Y
- **Capacity miss** when access Y again
- How to choose Y to minimize chance of needing it again?
- **Least recently used (LRU) replacement:** the least recently used block in a set evicted



# Types of Misses

- **Compulsory:** first time data accessed
- **Capacity:** cache too small to hold all data of interest
- **Conflict:** data of interest maps to same location in cache

**Miss penalty:** time it takes to retrieve a block from lower level of hierarchy



# LRU Replacement

## ARM Assembly Code

```
MOV R0, #0
LDR R1, [R0, #4]
LDR R2, [R0, #0x24]
LDR R3, [R0, #0x54]
```

Way 1				Way 0			
V	U	Tag	Data	V	Tag	Data	
0	0			0			Set 3 (11)
0	0			0			Set 2 (10)
0	0			0			Set 1 (01)
0	0			0			Set 0 (00)



# LRU Replacement

## ARM Assembly Code

```
MOV R0, #0
LDR R1, [R0, #4]
LDR R2, [R0, #0x24]
LDR R3, [R0, #0x54]
```

Way 1				Way 0			
V	U	Tag	Data	V	Tag	Data	
0	0			0			Set 3 (11)
0	0			0			Set 2 (10)
1	0	00...010	mem[0x00...24]	1	00...000	mem[0x00...04]	Set 1 (01)
0	0			0			Set 0 (00)

(a)

Way 1				Way 0			
V	U	Tag	Data	V	Tag	Data	
0	0			0			Set 3 (11)
0	0			0			Set 2 (10)
1	1	00...010	mem[0x00...24]	1	00...101	mem[0x00...54]	Set 1 (01)
0	0			0			Set 0 (00)

(b)



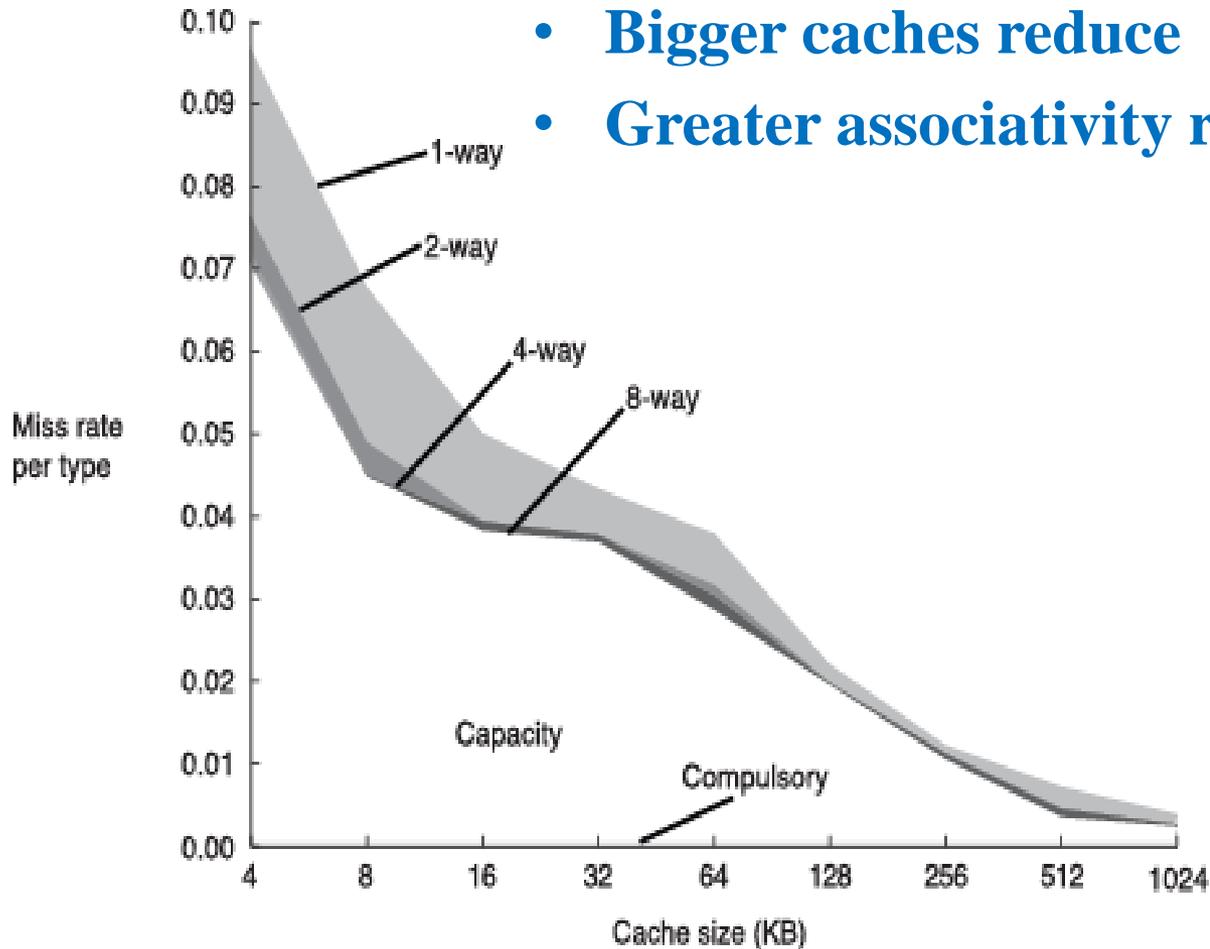
# Cache Summary

- **What data is held in the cache?**
  - Recently used data (temporal locality)
  - Nearby data (spatial locality)
- **How is data found?**
  - Set is determined by address of data
  - Word within block also determined by address
  - In associative caches, data could be in one of several ways
- **What data is replaced?**
  - Least-recently used way in the set

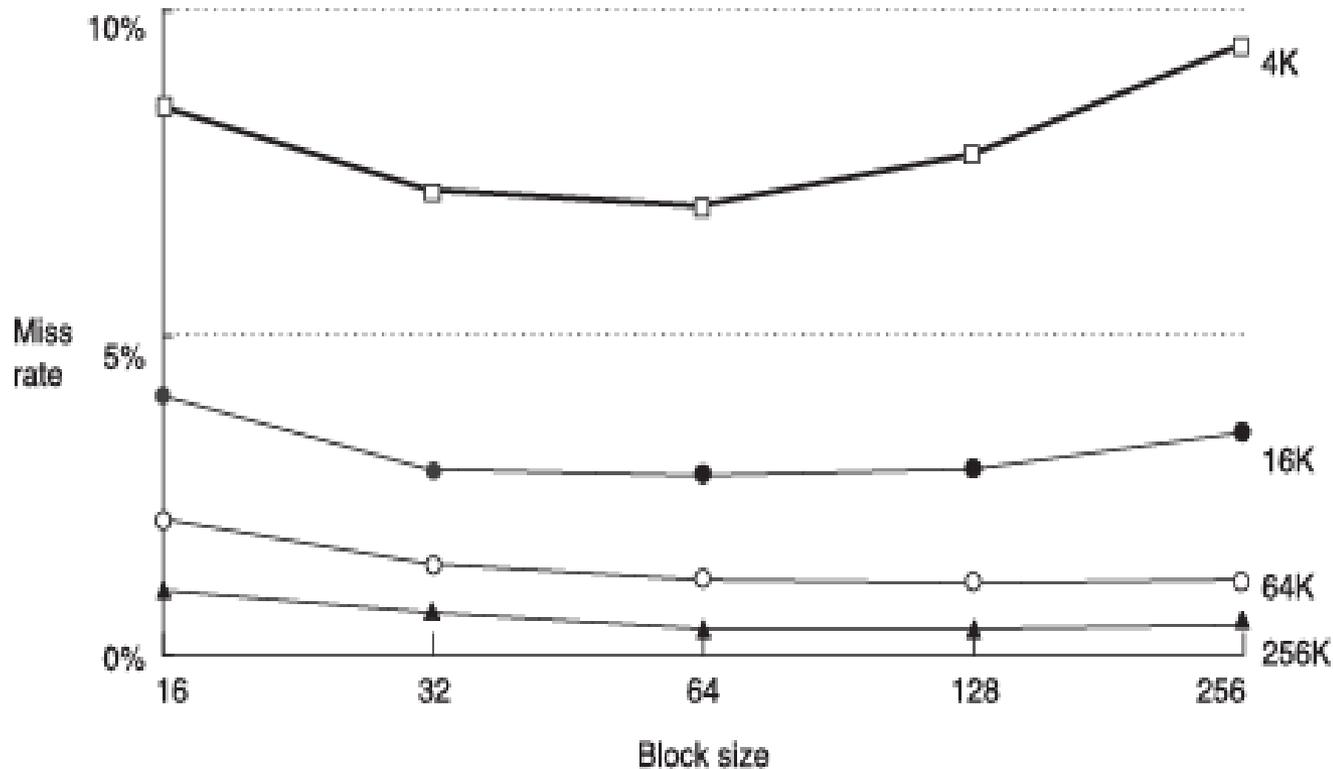


# Miss Rate Trends

- **Bigger caches reduce capacity misses**
- **Greater associativity reduces conflict misses**



# Miss Rate Trends



- **Bigger blocks reduce compulsory misses**
- **Bigger blocks increase conflict misses**

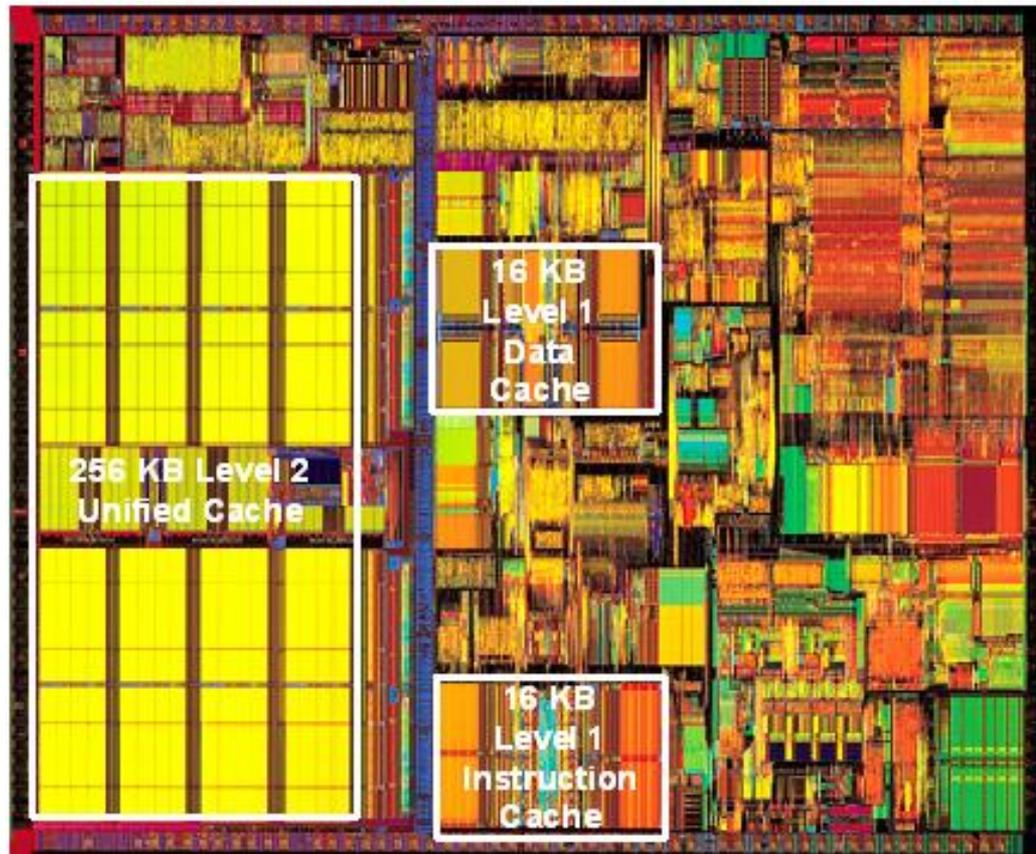


# Multilevel Caches

- Larger caches have lower miss rates, longer access times
- Expand memory hierarchy to multiple levels of caches
  - Level 1: small and fast (e.g. 16 KB, 1 cycle)
  - Level 2: larger and slower (e.g. 256 KB, 2-6 cycles)
- Most modern PCs have L1, L2, and L3 cache



# Intel Pentium III Die

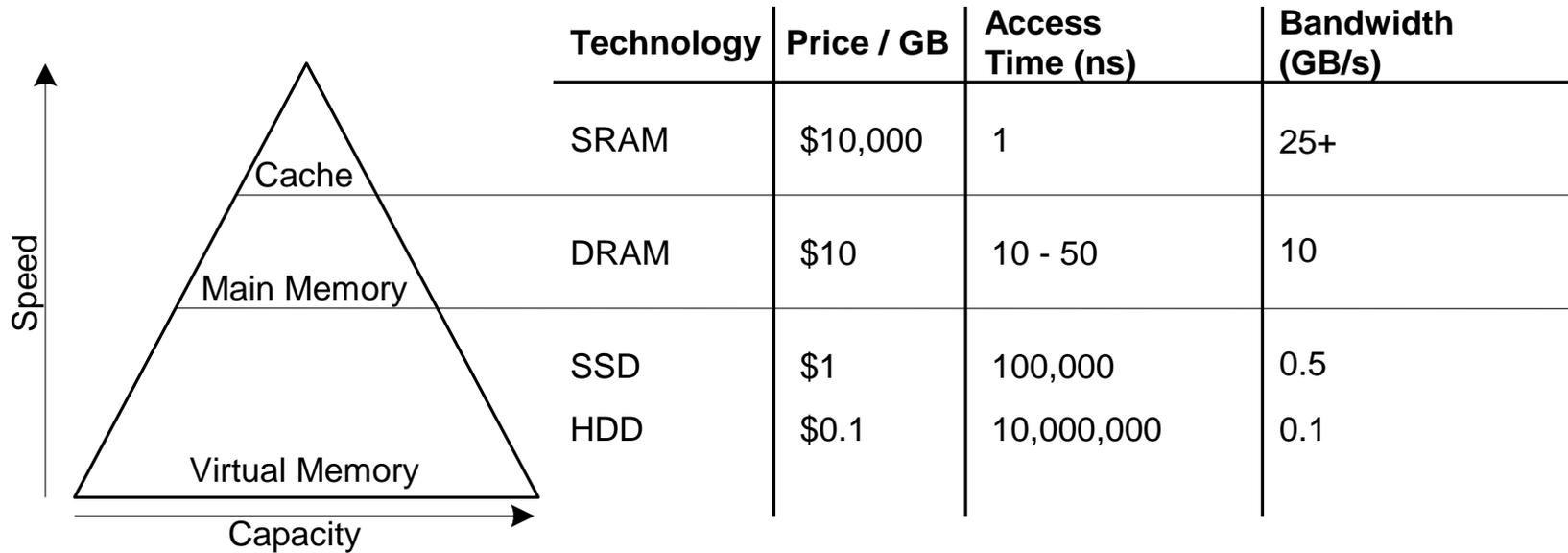


# Virtual Memory

- Gives the illusion of bigger memory
- Main memory (DRAM) acts as cache for hard disk



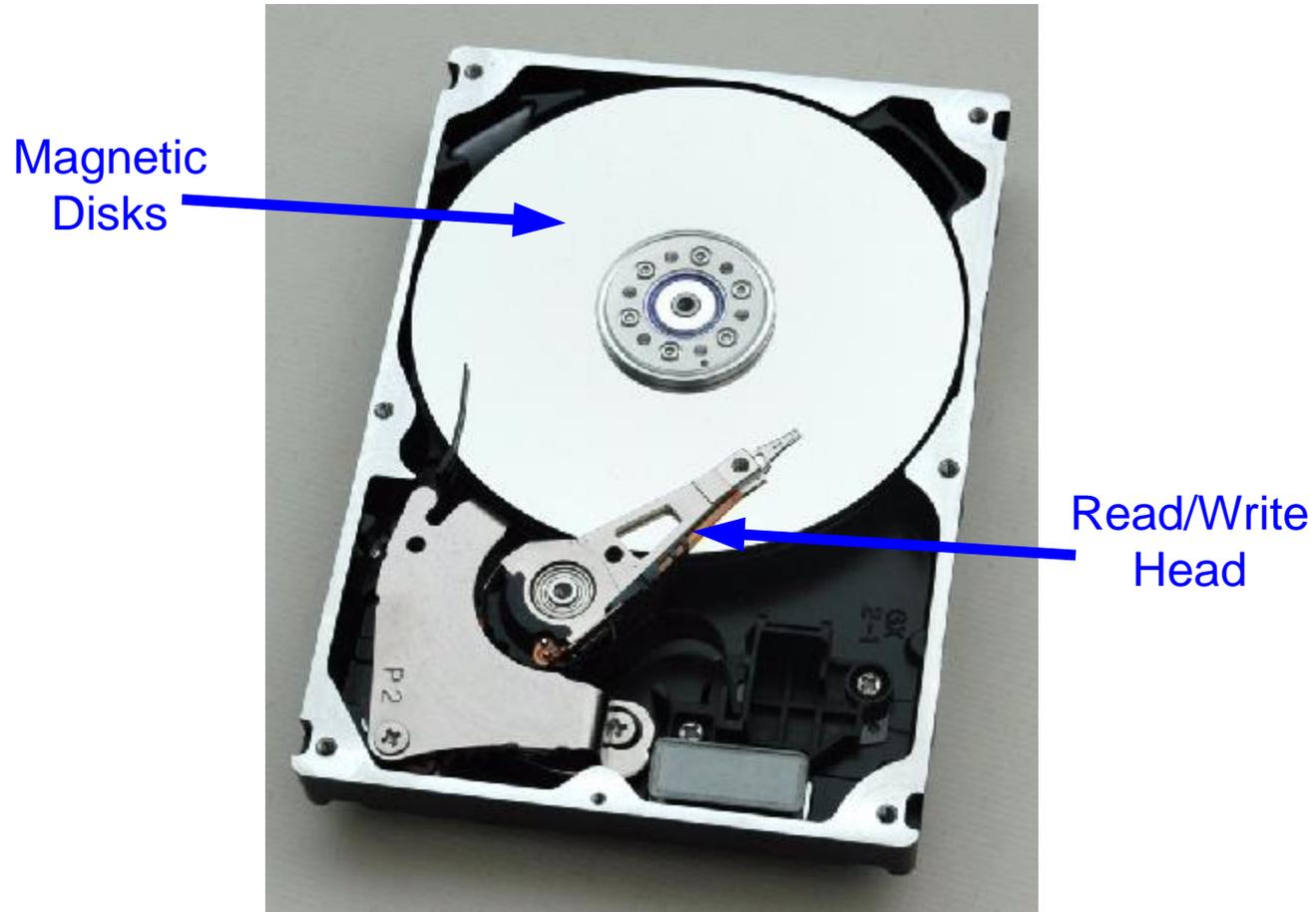
# Memory Hierarchy



- **Physical Memory:** DRAM (Main Memory)
- **Virtual Memory:** Hard drive
  - Slow, Large, Cheap



# Hard Disk



**Takes milliseconds to *seek* correct location on disk**



# Virtual Memory

- **Virtual addresses**

- Programs use virtual addresses
- Entire virtual address space stored on a hard drive
- Subset of virtual address data in DRAM
- CPU translates virtual addresses into ***physical addresses*** (DRAM addresses)
- Data not in DRAM fetched from hard drive

- **Memory Protection**

- Each program has own virtual to physical mapping
- Two programs can use same virtual address for different data
- Programs don't need to be aware others are running
- One program (or virus) can't corrupt memory used by another



# Cache/Virtual Memory Analogues

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

**Physical memory acts as cache for virtual memory**

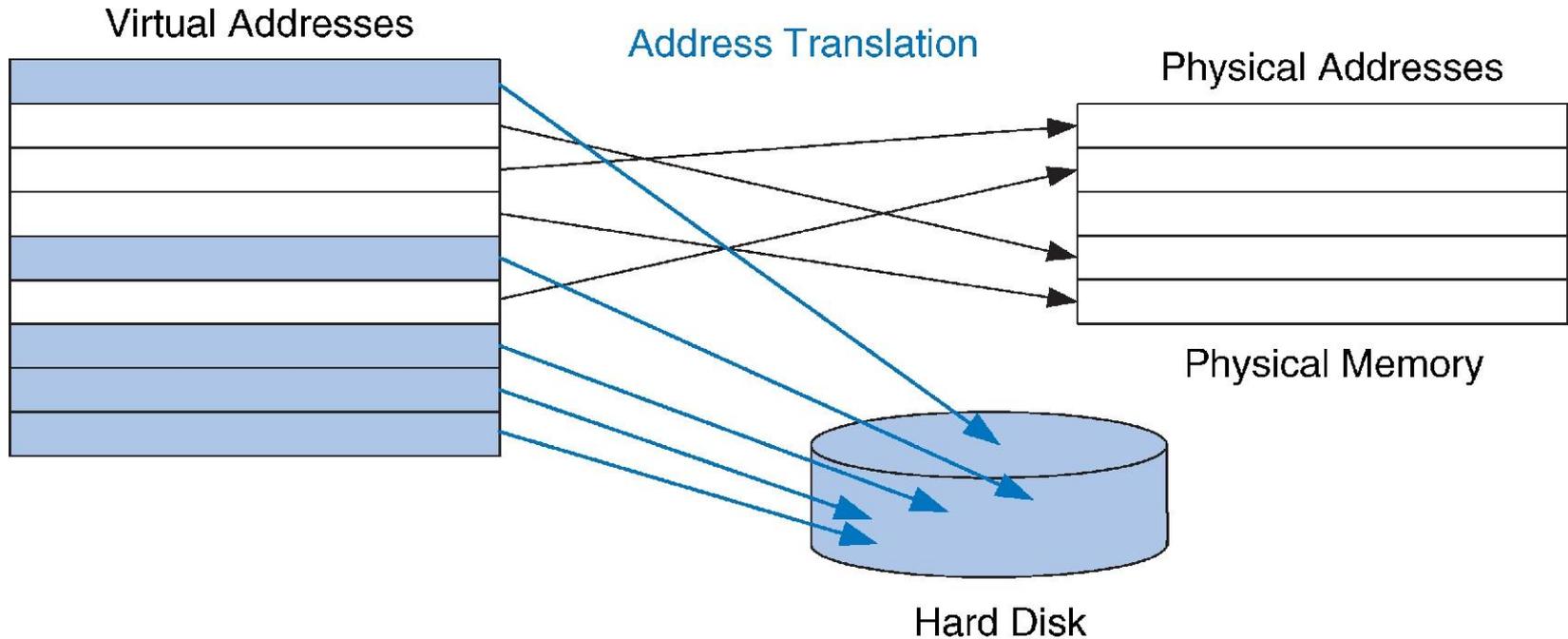


# Virtual Memory Definitions

- **Page size:** amount of memory transferred from hard disk to DRAM at once
- **Address translation:** determining physical address from virtual address
- **Page table:** lookup table used to translate virtual addresses to physical addresses



# Virtual & Physical Addresses

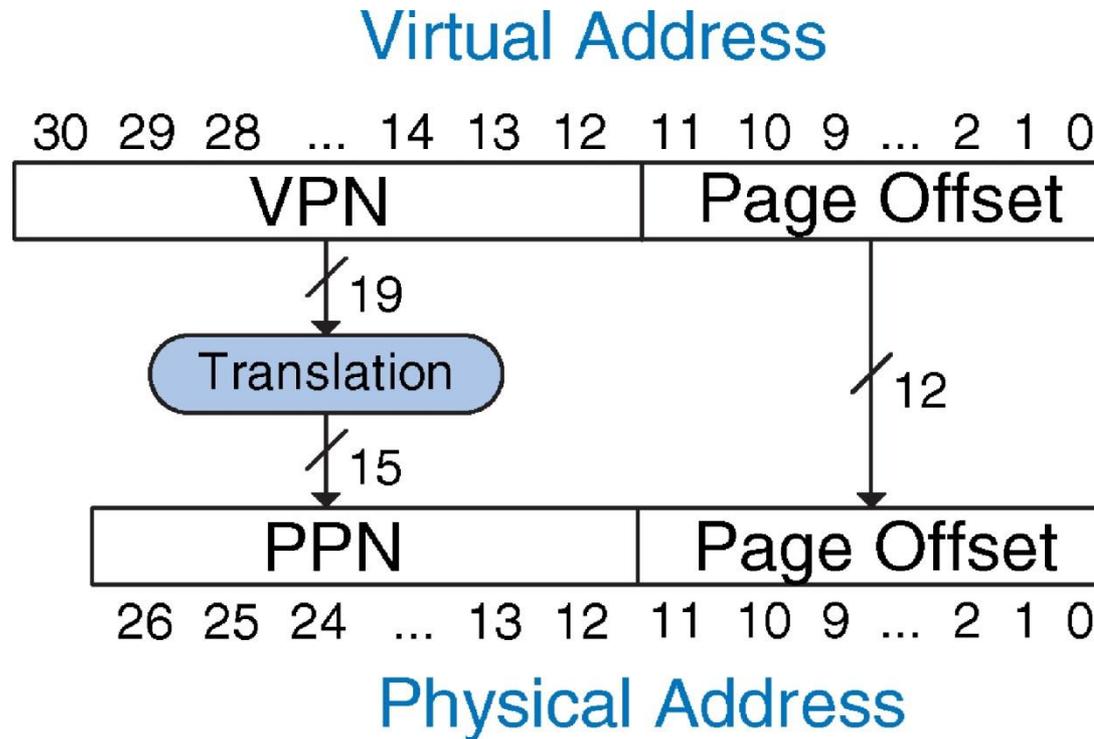


**Most accesses hit in physical memory**

**But programs have the large capacity of virtual memory**



# Address Translation



# Virtual Memory Example

- **System:**

- Virtual memory size: 2 GB =  $2^{31}$  bytes
- Physical memory size: 128 MB =  $2^{27}$  bytes
- Page size: 4 KB =  $2^{12}$  bytes



# Virtual Memory Example

- **System:**

- Virtual memory size: 2 GB =  $2^{31}$  bytes
- Physical memory size: 128 MB =  $2^{27}$  bytes
- Page size: 4 KB =  $2^{12}$  bytes

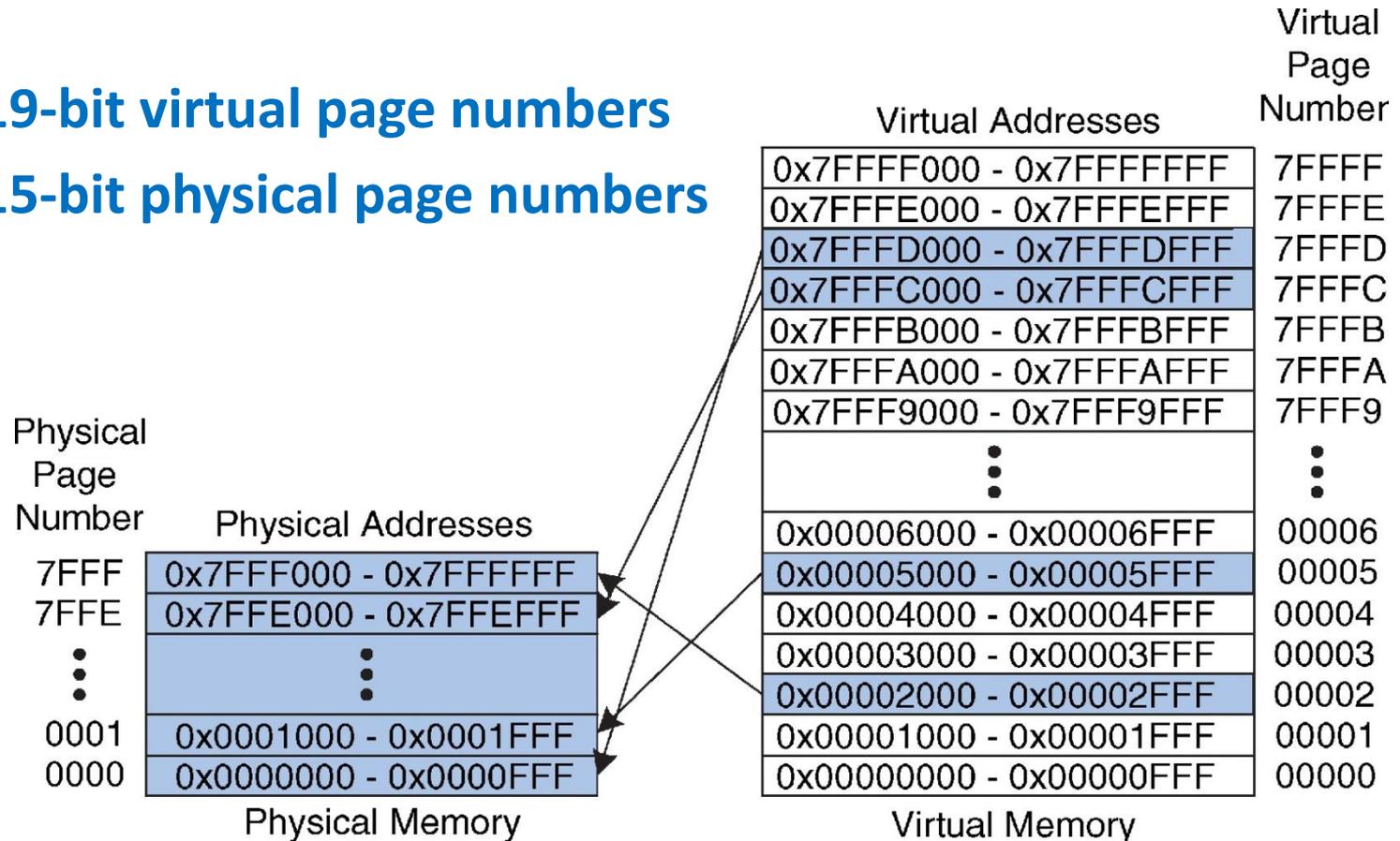
- **Organization:**

- Virtual address: **31** bits
- Physical address: **27** bits
- Page offset: **12** bits
- # Virtual pages =  $2^{31}/2^{12} = 2^{19}$  (VPN = **19** bits)
- # Physical pages =  $2^{27}/2^{12} = 2^{15}$  (PPN = **15** bits)



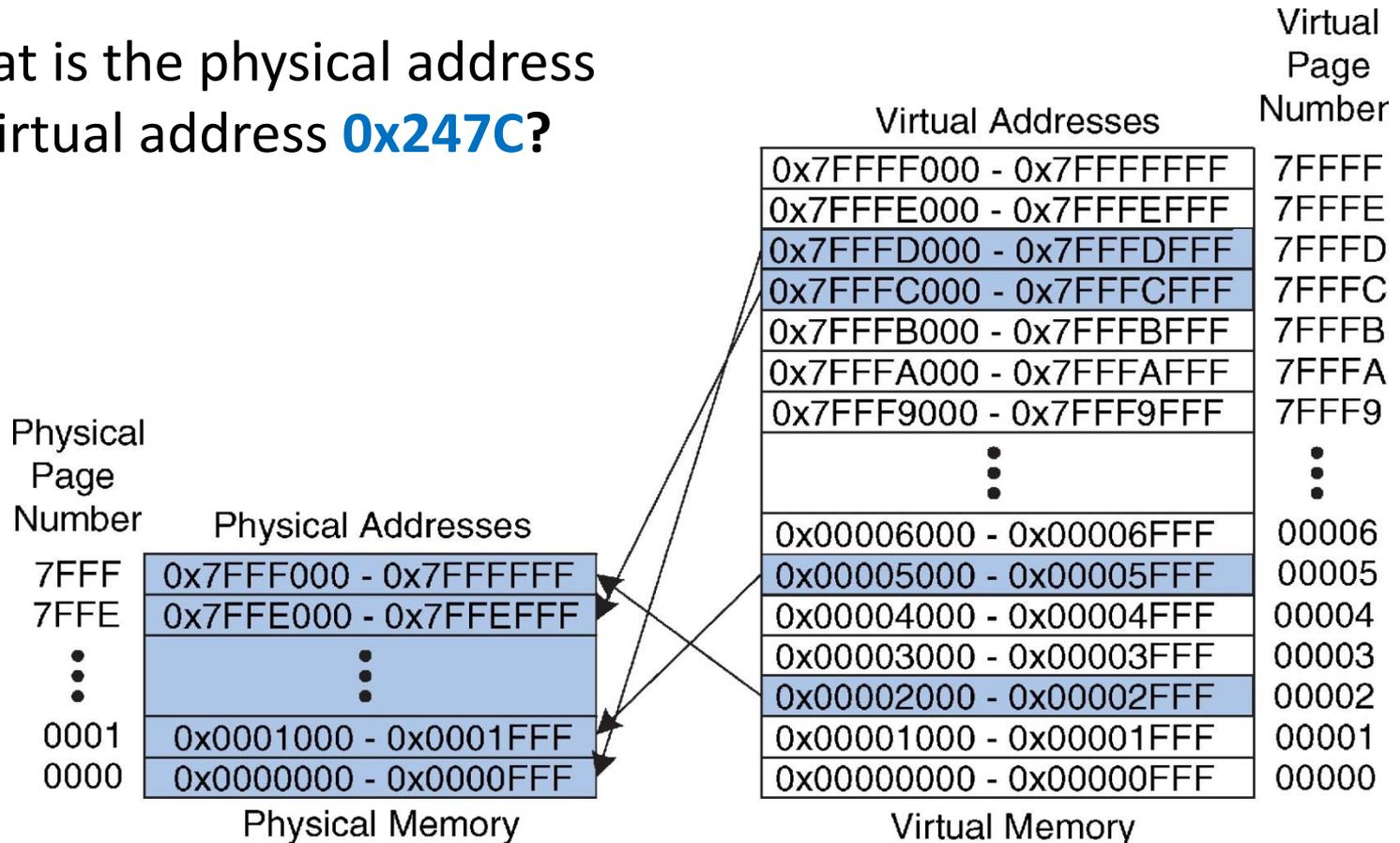
# Virtual Memory Example

- 19-bit virtual page numbers
- 15-bit physical page numbers



# Virtual Memory Example

What is the physical address of virtual address **0x247C**?



# Virtual Memory Example

What is the physical address of virtual address **0x247C**?

- VPN = **0x2**
- VPN 0x2 maps to PPN **0x7FFF**
- 12-bit page offset: **0x47C**
- Physical address = **0x7FFF47C**

Physical  
Page  
Number

7FFF  
7FFE  
⋮  
0001  
0000

Physical Addresses

0x7FFF000 - 0x7FFFFFFF
0x7FFE000 - 0x7FFEFF
⋮
0x0001000 - 0x0001FFF
0x0000000 - 0x0000FFF

Physical Memory

Virtual Addresses

0x7FFFF000 - 0x7FFFFFFF	7FFFF
0x7FFFE000 - 0x7FFFEFFF	7FFFE
0x7FFFD000 - 0x7FFFDFFF	7FFFD
0x7FFFC000 - 0x7FFFCFFF	7FFFC
0x7FFFB000 - 0x7FFFBFFF	7FFFB
0x7FFFA000 - 0x7FFFAFFF	7FFFA
0x7FFF9000 - 0x7FFF9FFF	7FFF9
⋮	⋮
0x00006000 - 0x00006FFF	00006
0x00005000 - 0x00005FFF	00005
0x00004000 - 0x00004FFF	00004
0x00003000 - 0x00003FFF	00003
0x00002000 - 0x00002FFF	00002
0x00001000 - 0x00001FFF	00001
0x00000000 - 0x00000FFF	00000

Virtual Memory

Virtual  
Page  
Number



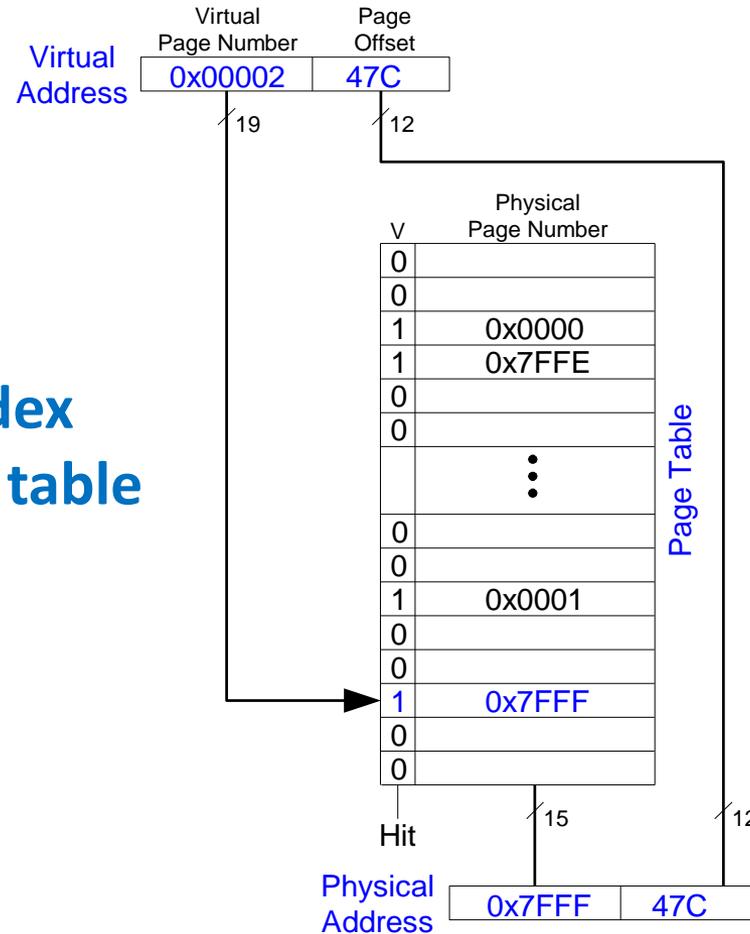
# How to perform translation?

- **Page table**
  - Entry for each virtual page
  - Entry fields:
    - **Valid bit:** 1 if page in physical memory
    - **Physical page number:** where the page is located



# Page Table Example

VPN is index  
into page table



# Page Table Example 1

What is the physical address of virtual address **0x5F20**?

V	Physical Page Number
0	
0	
1	0x0000
1	0x7FFE
0	
0	
	⋮
0	
0	
1	0x0001
0	
0	
1	0x7FFF
0	
0	

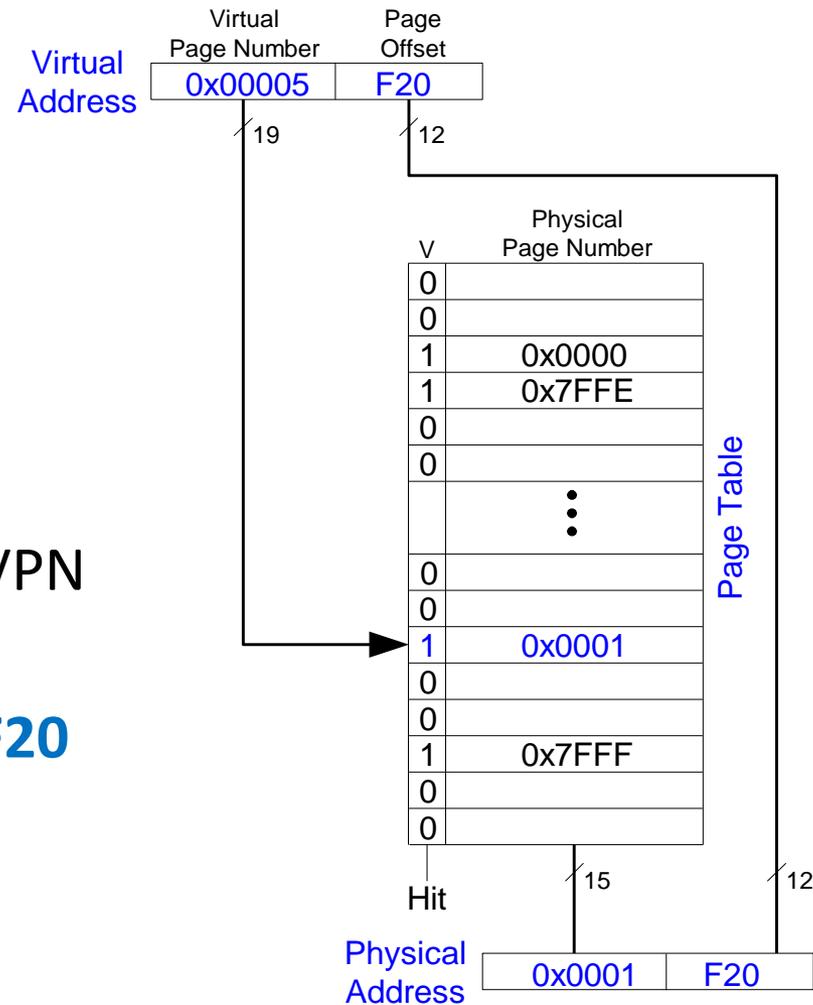
Page Table



# Page Table Example 1

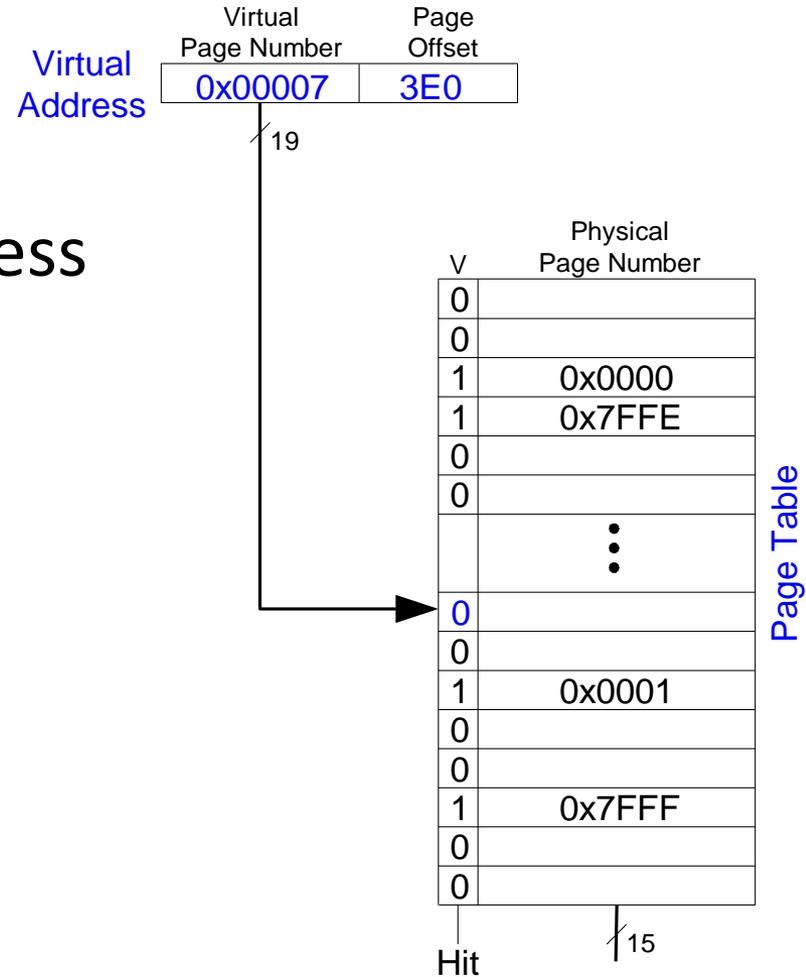
What is the physical address of virtual address **0x5F20**?

- VPN = **5**
- Entry 5 in page table VPN 5 => physical page **1**
- Physical address: **0x1F20**



# Page Table Example 2

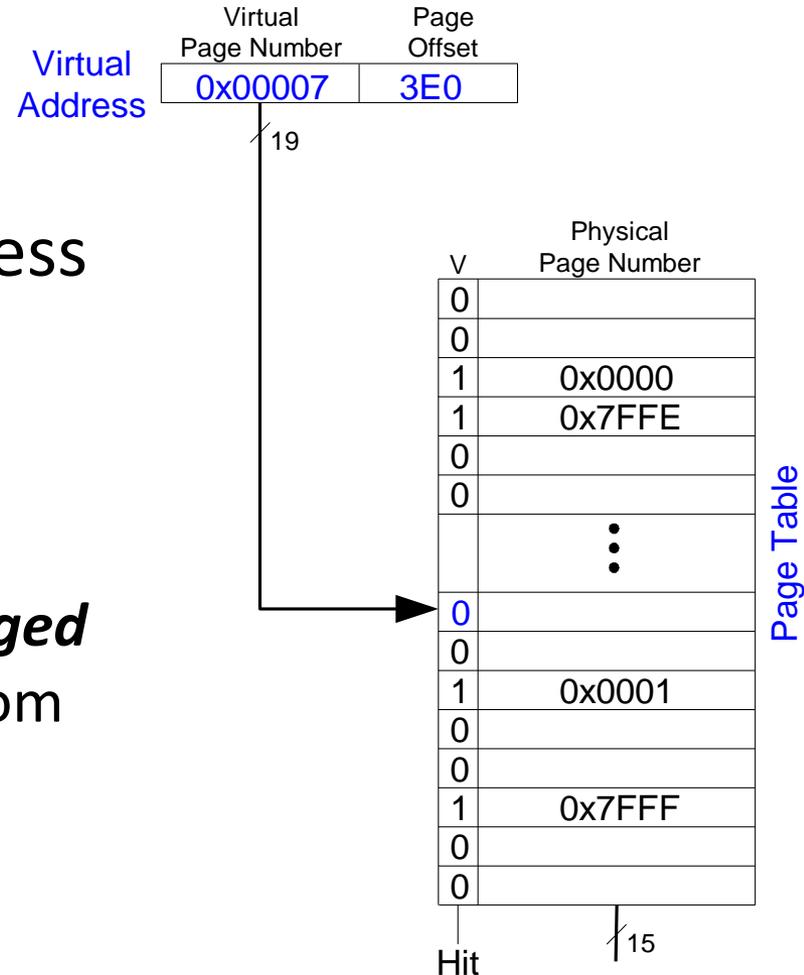
What is the physical address of virtual address **0x73E0**?



# Page Table Example 2

What is the physical address of virtual address **0x73E0**?

- VPN = **7**
- Entry 7 is invalid
- Virtual page must be *paged* into physical memory from disk



# Page Table Challenges

- **Page table is large**
  - usually located in physical memory
- Load/store requires 2 main memory accesses:
  - one for translation (page table read)
  - one to access data (after translation)
- Cuts memory performance in half
  - *Unless we get clever...*



# Translation Lookaside Buffer (TLB)

- Small cache of most recent translations
- Reduces # of memory accesses for *most* loads/stores from 2 to 1

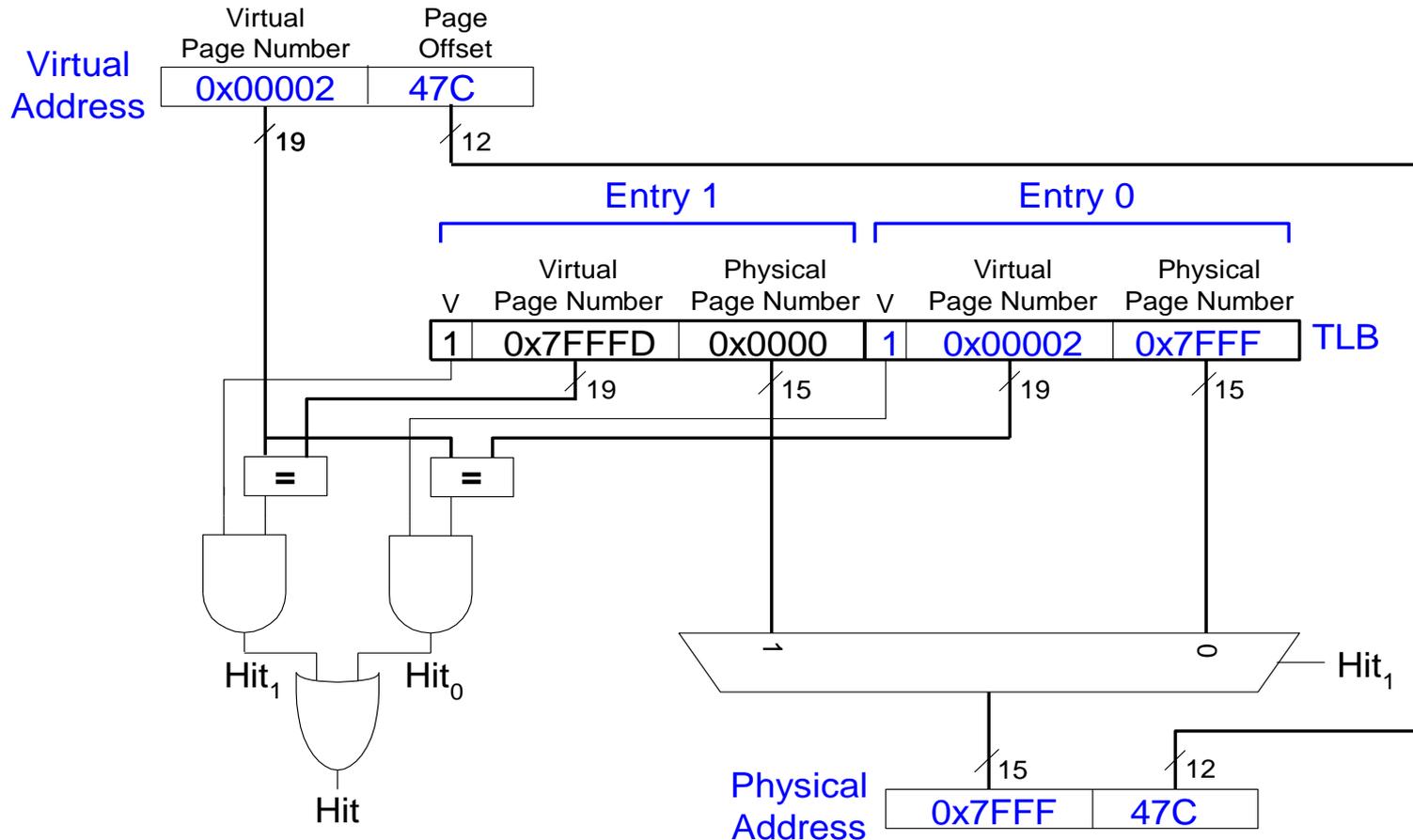


# TLB

- Page table accesses: high temporal locality
  - Large page size, so consecutive loads/stores likely to access same page
- TLB
  - Small: accessed in  $< 1$  cycle
  - Typically 16 - 512 entries
  - Fully associative
  - $> 99\%$  hit rates typical
  - Reduces # of memory accesses for most loads/stores from 2 to 1



# Example 2-Entry TLB



# Memory Protection

- Multiple processes (programs) run at once
- Each process has its own page table
- Each process can use entire virtual address space
- A process can only access physical pages mapped in its own page table



# Virtual Memory Summary

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

