Cache Architectures

Design of Digital Circuits 2017
Srdjan Capkun
Onur Mutlu

http://www.syssec.ethz.ch/education/Digitaltechnik_17

What Will We Learn ?

- Brief review of how data can be stored
- Memory System Performance Analysis
- Caches

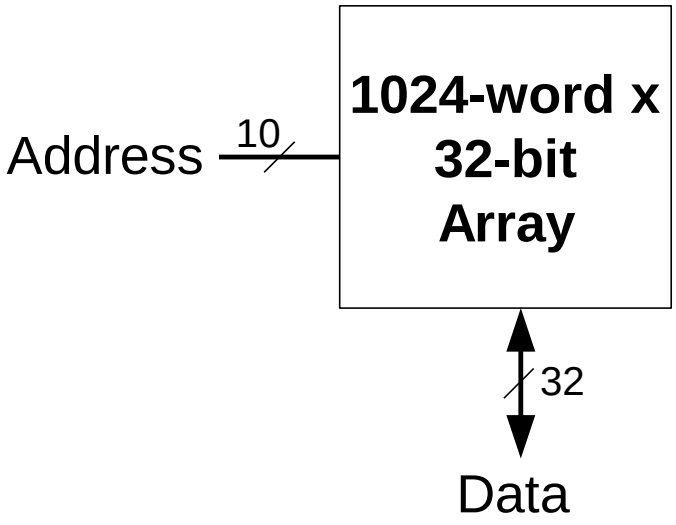
Review: What were Memory Elements ?

- **Memories are large blocks**
  - A significant portion of a modern circuit is memory.
- **Memories are practical tools for system design**
  - Programmability, reconfigurability all require memory
- **Allows you to store data and work on stored data**
  - Not all algorithms are designed to process data as it comes, some require data to be stored.
  - Data type determines required storage
    - SMS: 160 bytes
    - 1 second normal audio: 64 kbytes
    - 1 HD picture: 7.32 Mbytes

How Can We Store Data

- **Flip-Flops (or Latches)**
  - Very fast, parallel access
  - Expensive (one bit costs 20+ transistors)
- **Static RAM**
  - Relatively fast, only one data word at a time
  - Less expensive (one bit costs 6 transistors)
- **Dynamic RAM**
  - Slower, reading destroys content (refresh), one data word at a time, needs special process
  - Cheaper (one bit is only a transistor)
- **Other storage technology (hard disk, flash)**
  - Much slower, access takes a long time, non-volatile
  - Per bit cost is lower (no transistors directly involved)

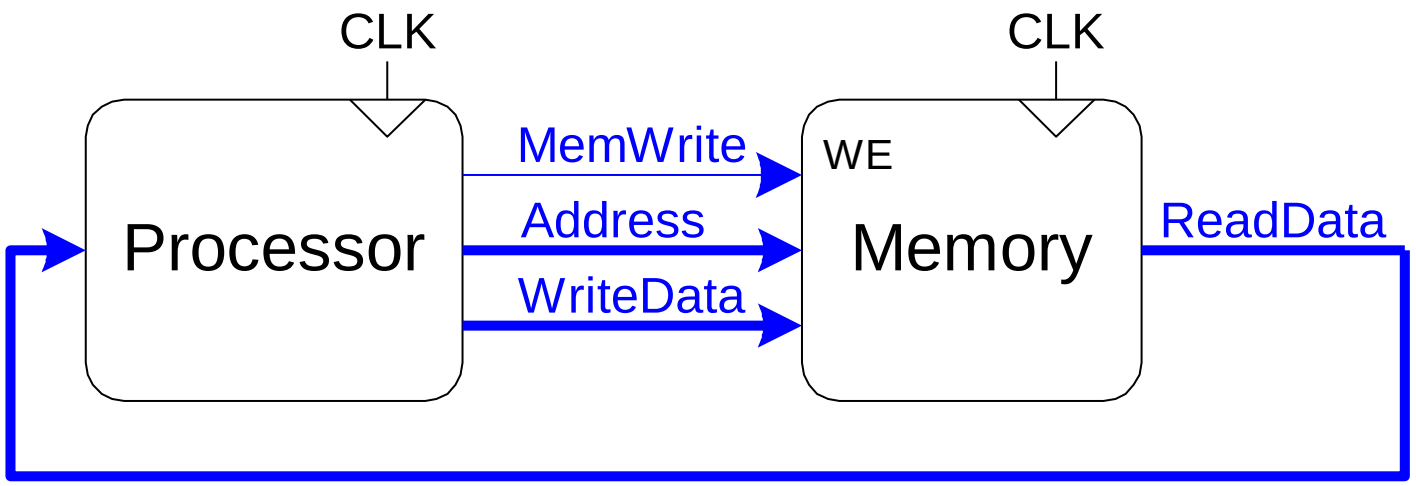
Array Organization of Memories



- **Efficiently store large amounts of data**
  - Consists of a memory ***array to store data***
  - The ***address selects one row*** of the array
  - The data in the row is read out
- **An M-bit value can be read or written at each unique N-bit address**
  - All values in array can be accessed
  - … but only M-bits at a time

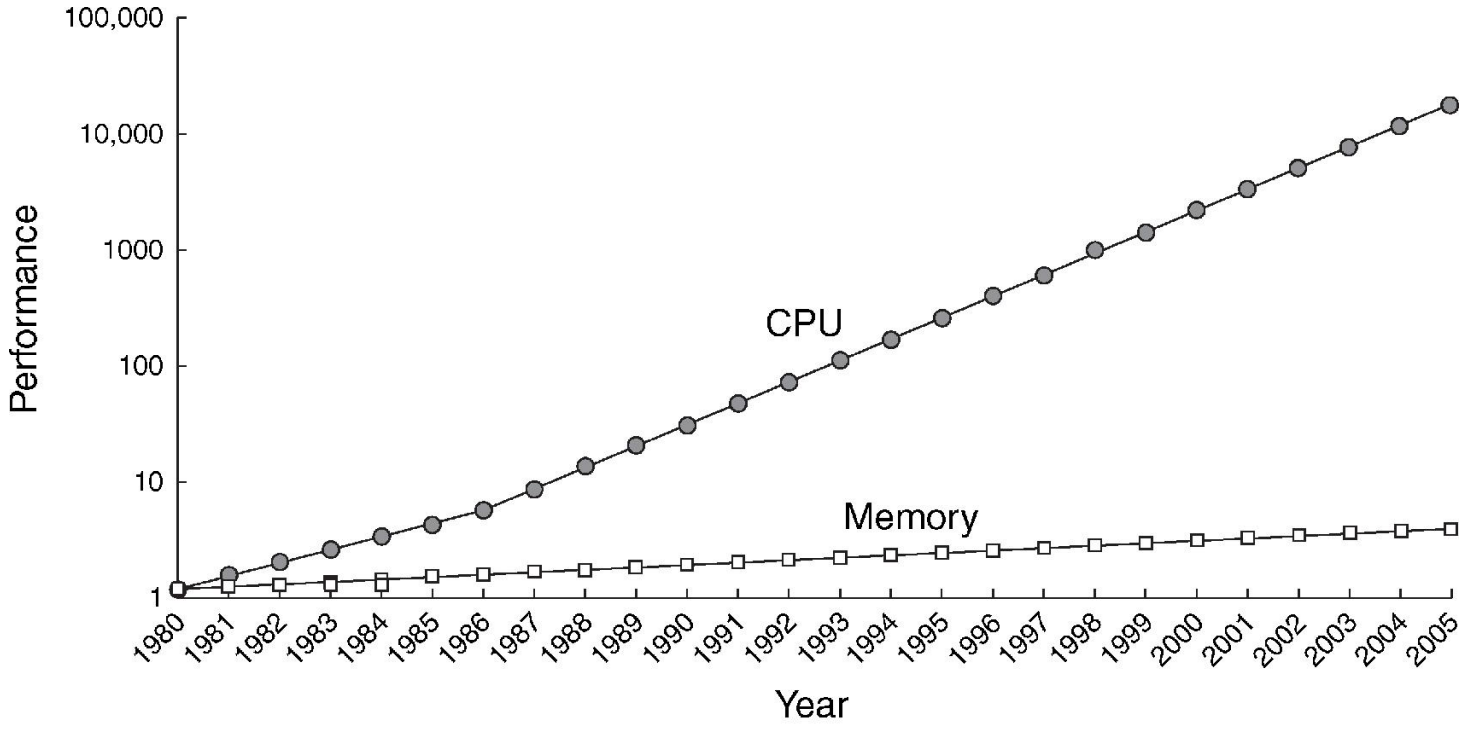
Introduction

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



Introduction

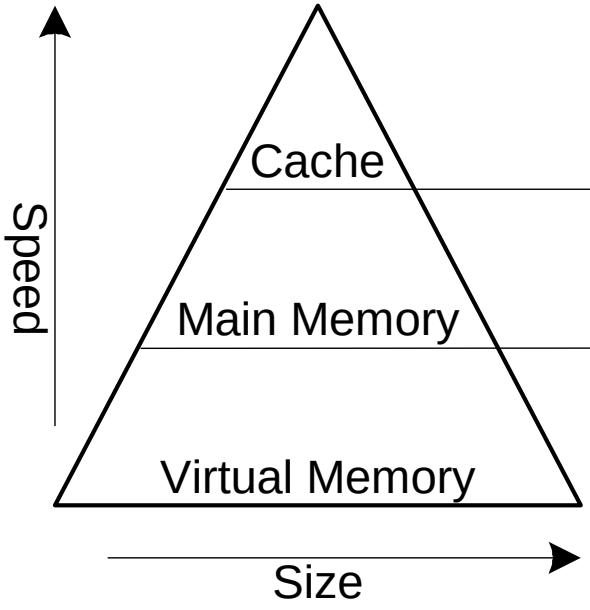
- Up until now, *assumed* memory could be accessed in 1 clock cycle
- But that hasn't been true since the 1980's



Memory System Challenge

- **Make memory system appear as fast as processor**
- **Use a hierarchy of memories**
- **Ideal memory:**
  - Fast
  - Cheap (inexpensive)
  - Large (capacity)
- **But we can only choose two!**

Memory Hierarchy



Technology	cost / GB	Access time
SRAM	~ $10,000	~ 1 ns
DRAM	~ $100	~ 100 ns
Hard Disk	~ $1	~ 10,000,000 ns

Locality

- **Exploit locality to make memory accesses fast**

- **Temporal Locality:**
  - Locality in time (e.g., if looked at a Web page recently, likely to look at it again soon)
  - 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 (e.g., if read one page of book recently, likely to read nearby pages soon)
  - If data used recently, likely to use nearby data soon
  - ***How to exploit:*** when accessing data, bring nearby data into higher levels of memory hierarchy too

Memory Performance

- ***Hit:*** **is found in that level of memory hierarchy**
- ***Miss:*** **is 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 it takes for processor to access data**

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

Memory Performance Example 1

A program has 2,000 load and store instructions. 1,250 of these data values found in cache. The rest are supplied by other levels of memory hierarchy

What are the hit and miss rates for the cache?

Hit Rate **=**

Miss Rate **=**

Memory Performance Example 1

A program has 2,000 load and store instructions. 1,250 of these data values found 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**

Memory Performance Example 2

Suppose a processor has 2 levels of hierarchy: cache and main memory:

- t_{cache} = 1 cycle,
- t_{MM} = 100 cycles

What is the AMAT of the program from Example 1?

AMAT **=**

Memory Performance Example 2

Suppose a processor has 2 levels of hierarchy: cache and main memory:

- t_{cache} = 1 cycle,
- t_{MM} = 100 cycles

What is the AMAT of the program from Example 1?

$$AMAT = t_{cache} + MR_{cache}(t_{MM})$$

$$= [1 + 0.375(100)] \text{ cycles}$$

$$= 38.5 \text{ cycles}$$

Cache

A safe place to hide things

- Highest level in memory hierarchy
- Fast (typically ~ 1 cycle access time)
- Ideally supplies most of the data to the 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?

We'll focus on data loads, but stores follow same principles

What data is held in the cache?

- **Ideally, cache anticipates data needed by processor and holds it in cache**
- **But impossible to predict future**
- **So, use past to predict future – temporal and spatial locality:**
  - ***Temporal locality:*** copy newly accessed data into cache. Next time it's accessed, it's available in cache.
  - ***Spatial locality:*** copy neighboring data into cache too. Block size = number of bytes copied into cache at once.

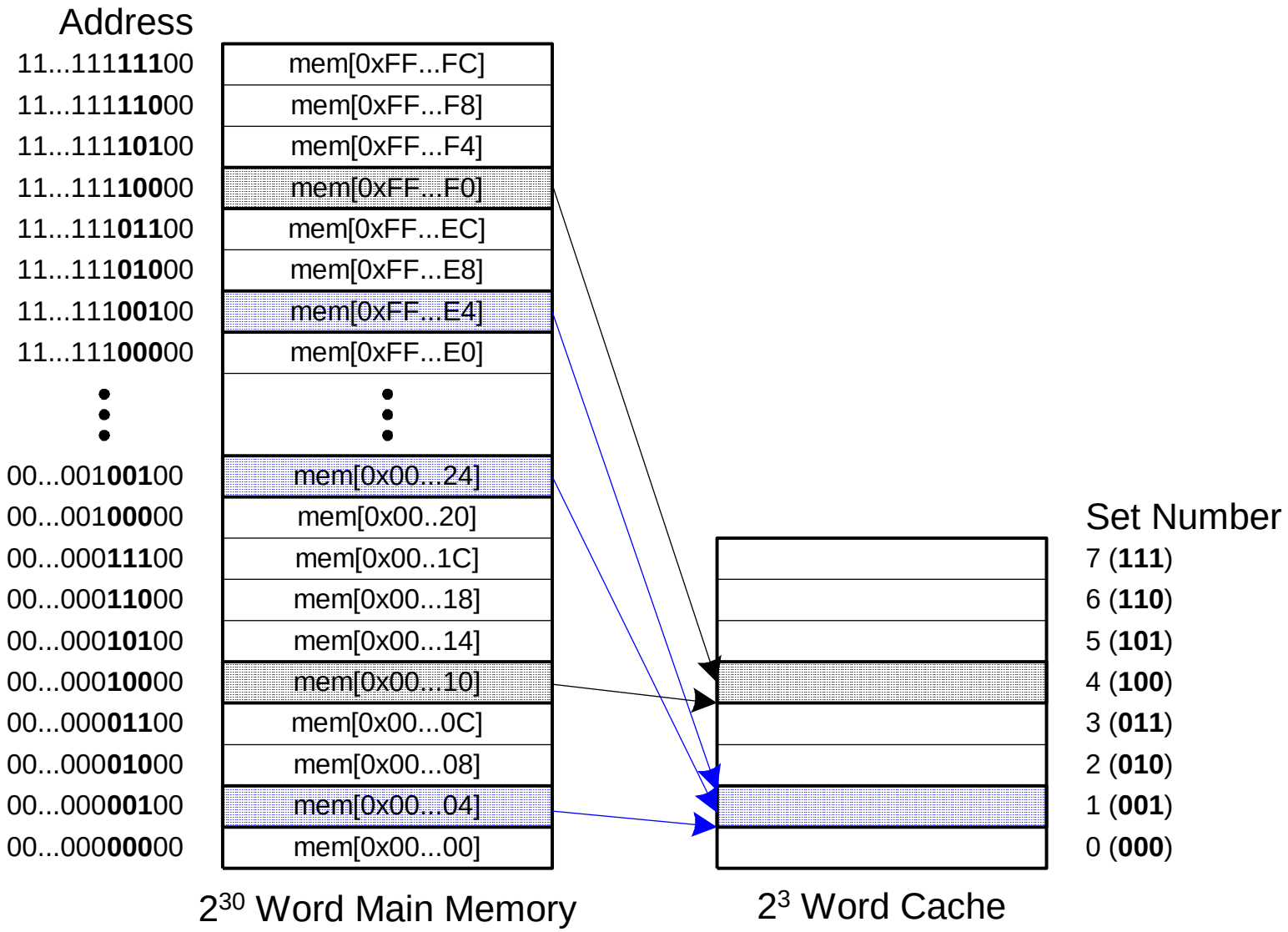
Cache Terminology

- **Capacity (*C*):**
  - the number of data bytes a cache stores
- **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

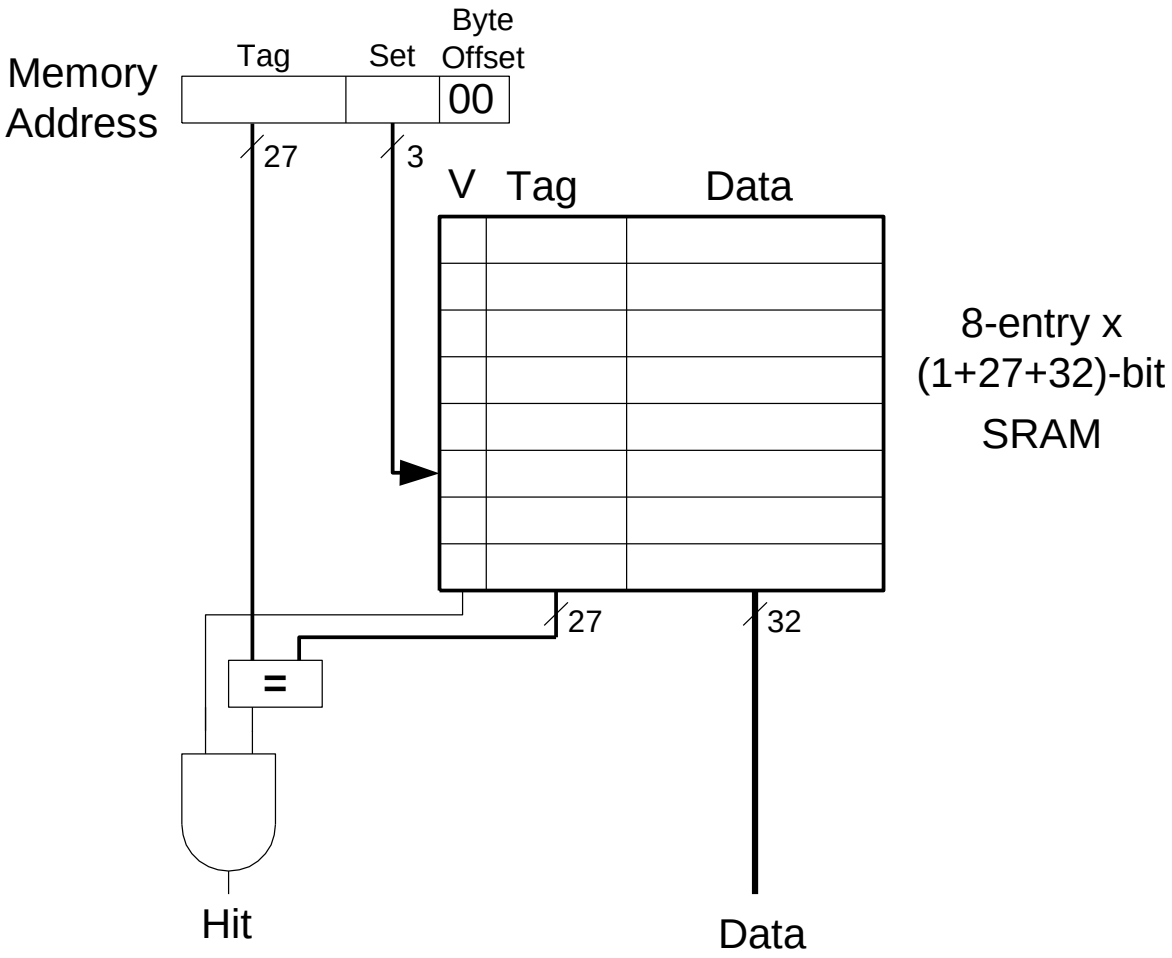
How is data found?

- **Cache organized into *S* sets**
- **Each memory address maps to exactly one set**
- **Caches categorized by number of blocks in a set:**
  - ***Direct mapped:*** 1 block per set
  - ***N-way set associative:*** N blocks per set
  - ***Fully associative:*** all cache blocks are in a single set
- **Examine each organization for a cache with:**
  - Capacity (C = 8 words)
  - Block size (b = 1 word)
  - So, number of blocks (B = 8)

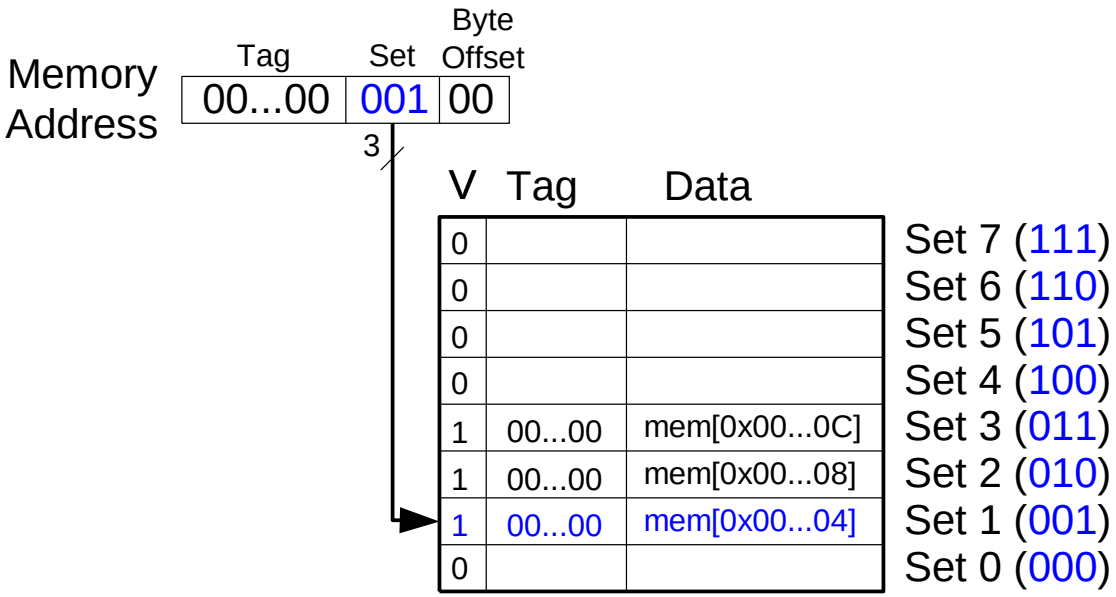
Direct Mapped Cache



Direct Mapped Cache Hardware



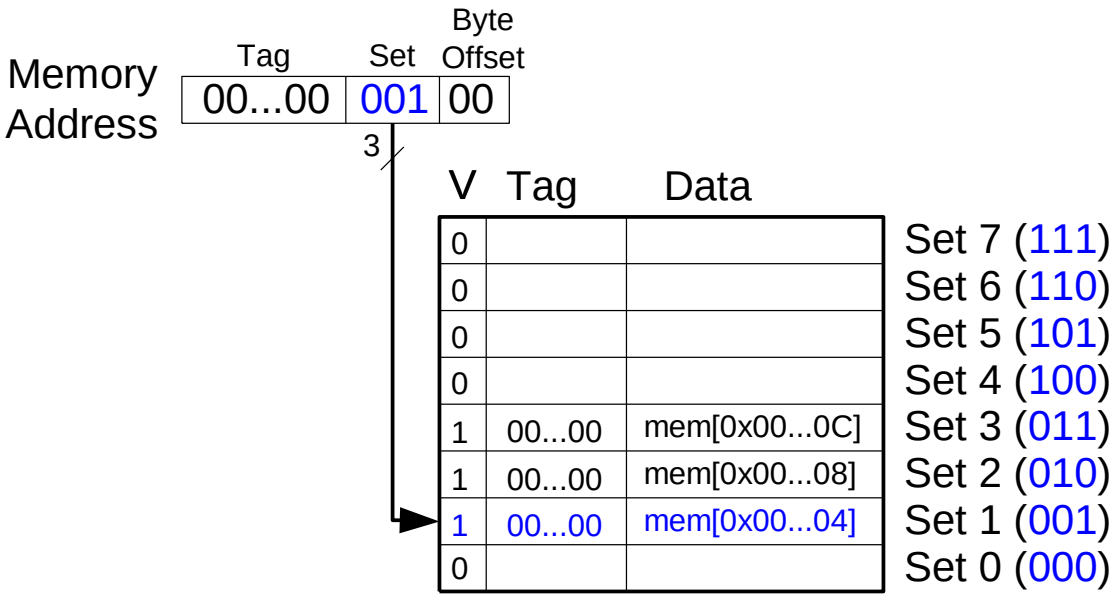
Direct Mapped Cache Performance




```
# MIPS assembly 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:
```

Miss Rate =

Direct Mapped Cache Performance




```
# MIPS assembly 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:
```

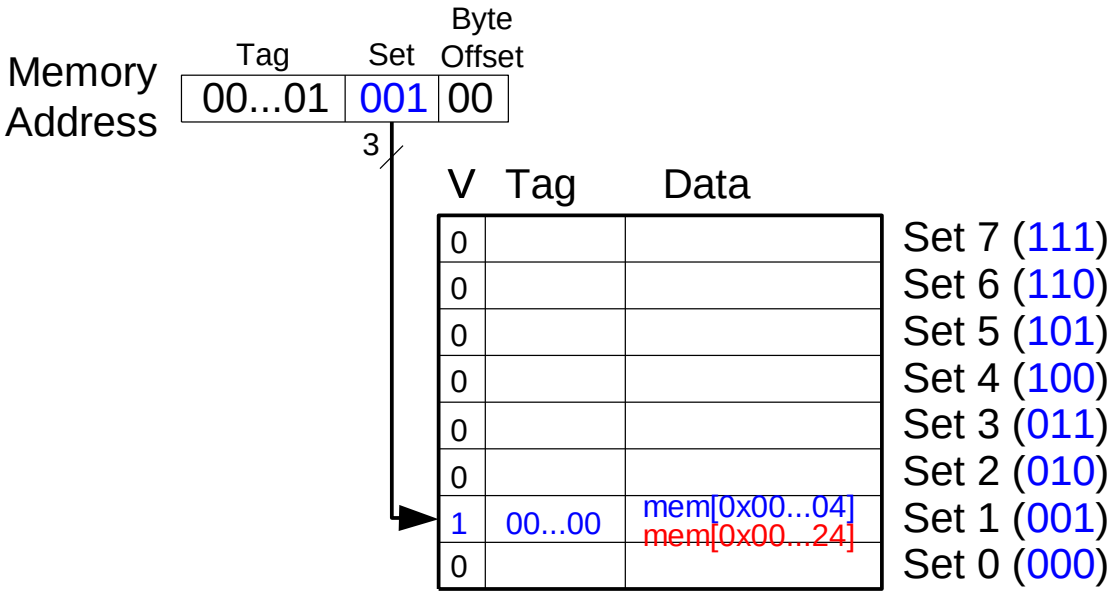
Miss Rate = 3/15

= 20%

Temporal Locality

Compulsory Misses

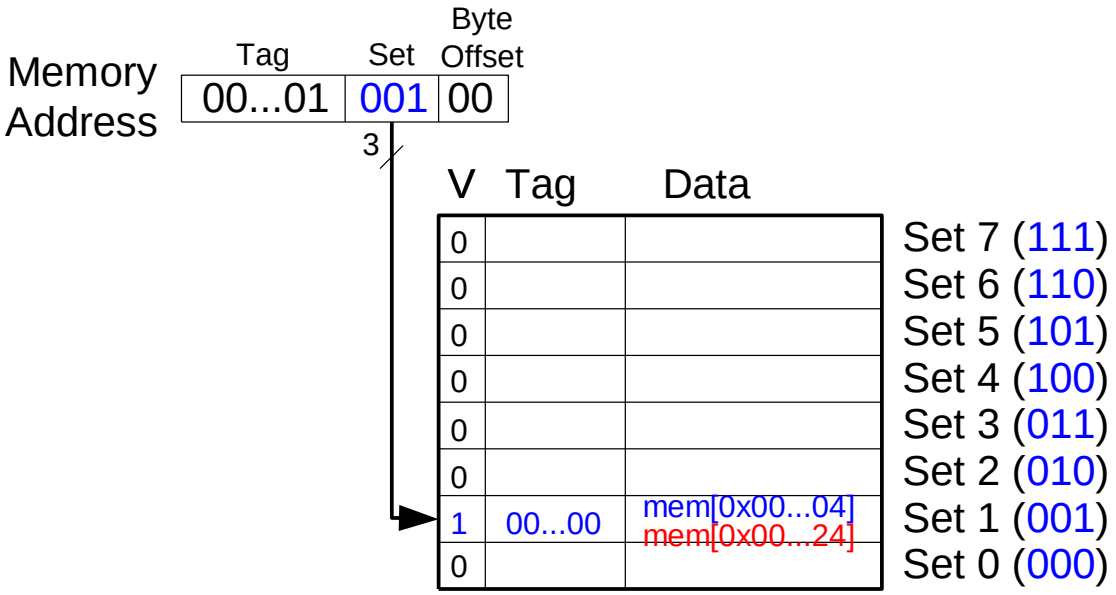
Direct Mapped Cache: Conflict




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

Miss Rate =

Direct Mapped Cache: Conflict



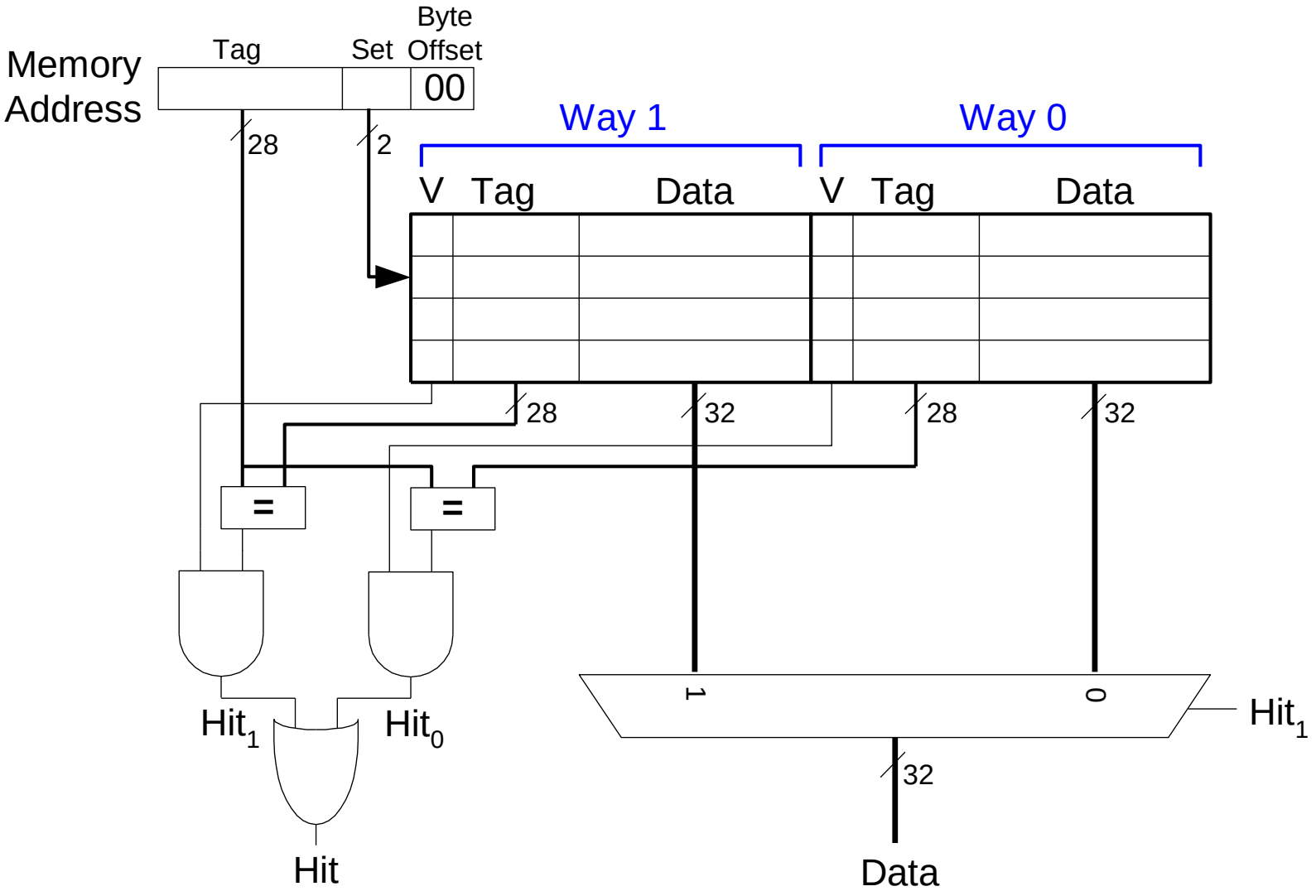

```
# MIPS assembly code
        addi $t0, $0, 5
loop:   beq  $t0, $0, done
        lw   $t1, 0x4($0)
        lw   $t2, 0x24($0)
        addi $t0, $t0, -1
        j    loop
done:
```

Miss Rate = 10/10

= 100%

Conflict Misses

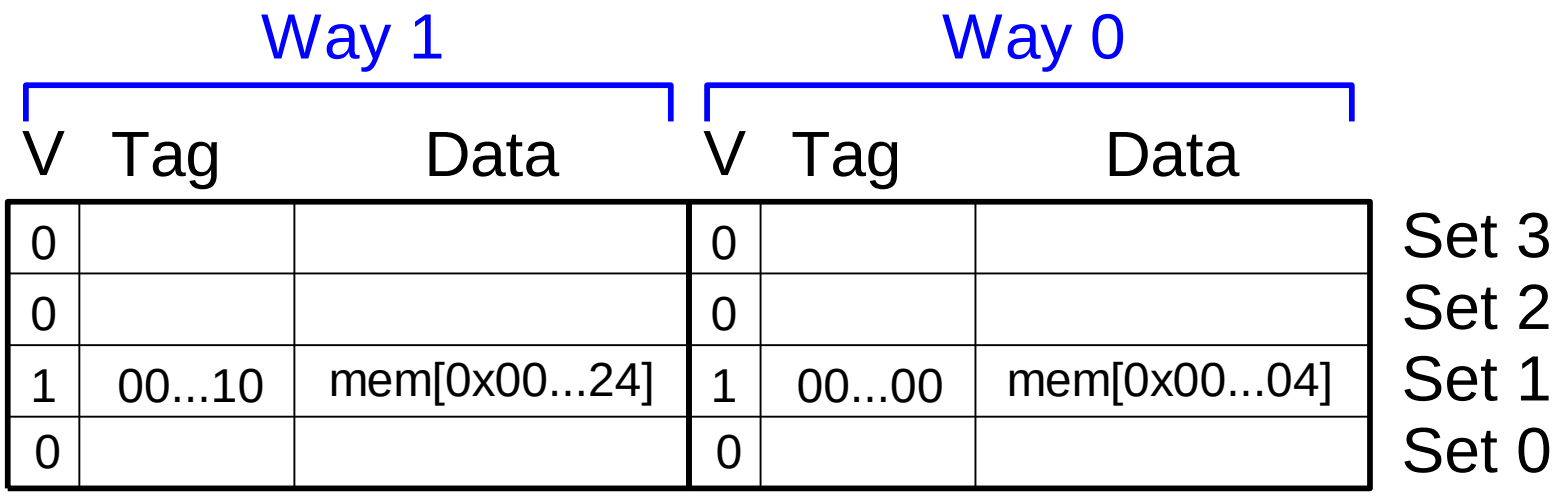
N-Way Set Associative Cache



N-way Set Associative Performance

```
# MIPS assembly code

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

Miss Rate =



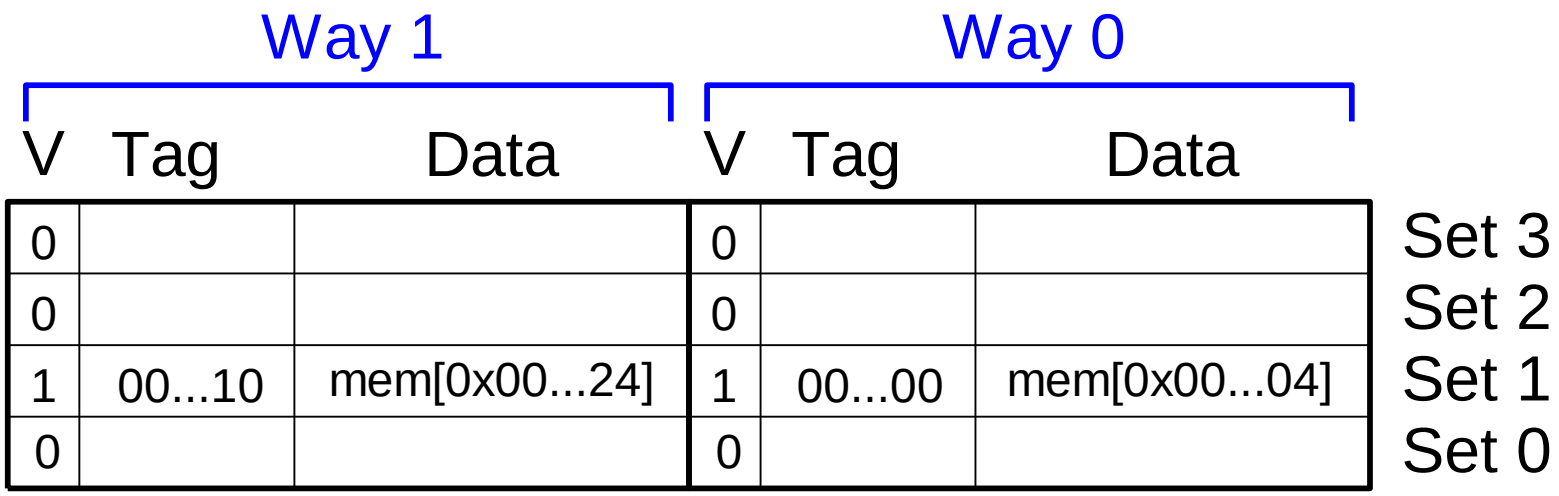
N-way Set Associative Performance

```
# MIPS assembly code

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

Miss Rate = 2/10

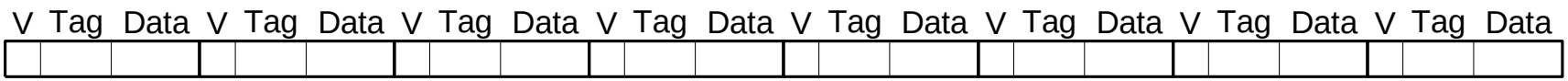
= 20%

Associativity reduces conflict misses



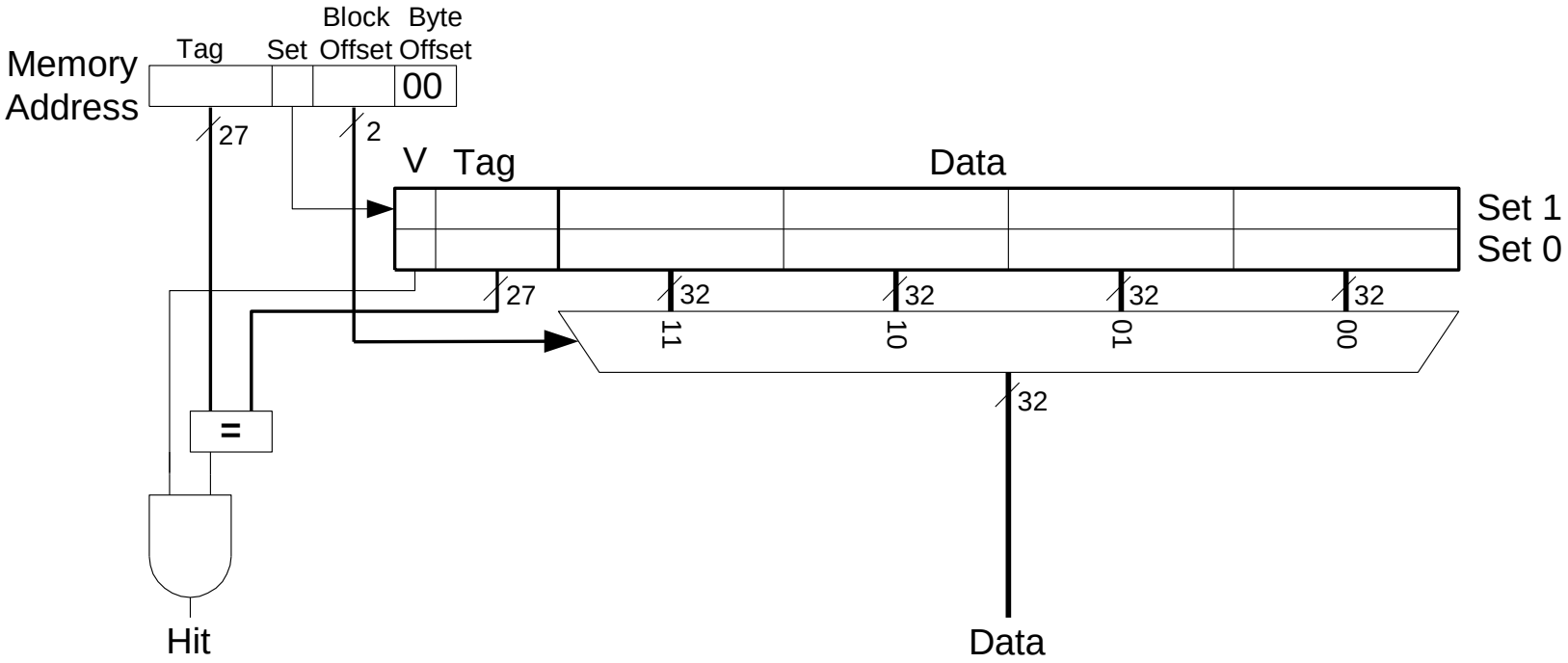
Fully Associative Cache

- **No conflict misses**
- **Expensive to build**



Spatial Locality?

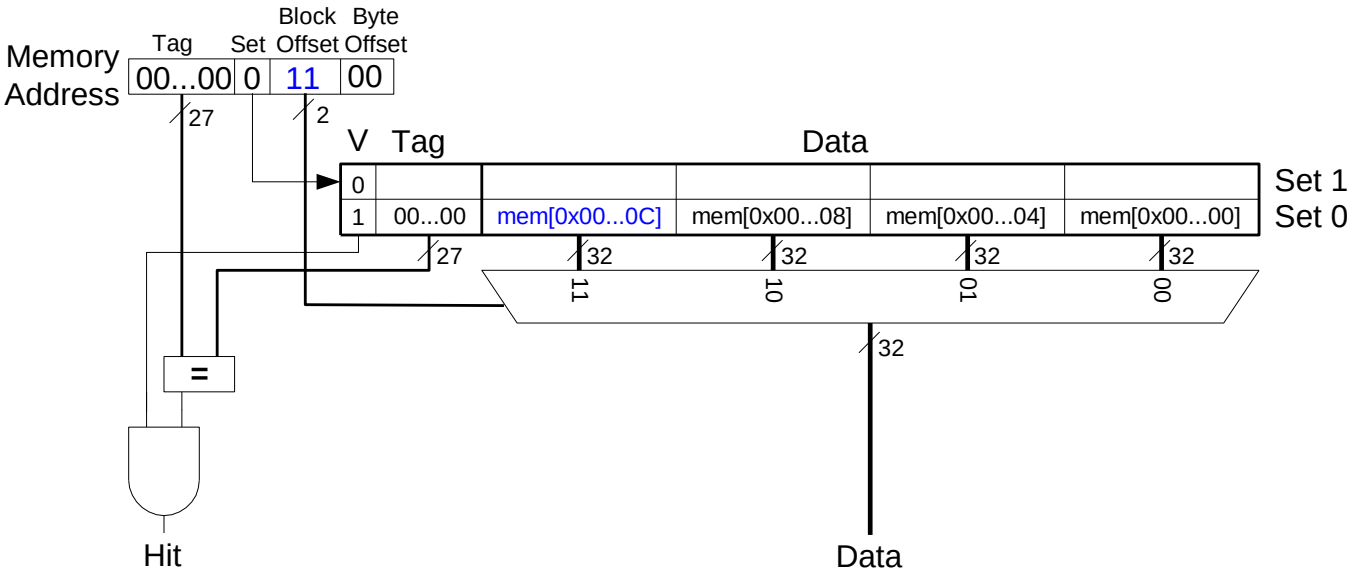
- **Increase block size:**
  - Block size, ***b* = 4** words
  - *C* = 8 words
  - Direct mapped (1 block per set)
  - Number of blocks, $B = C/b = 8/4 = 2$



Direct Mapped Cache Performance

```
        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:
```

Miss Rate =



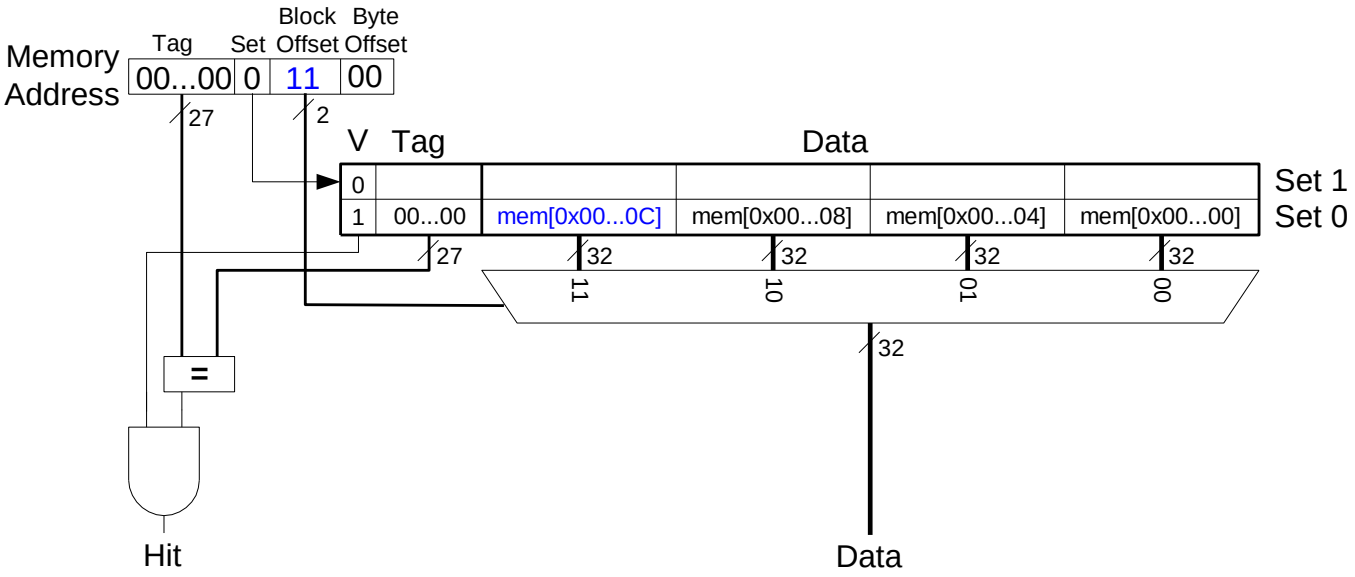
Direct Mapped Cache Performance

```
        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:
```

Miss Rate = 1/15

= 6.67%

Larger blocks reduce compulsory misses through spatial locality



Cache Organization Recap

- **Main Parameters**
  - 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 one time**
  - If the cache is full and program tries to access data X that is not in cache, cache must evict data Y to make room for X
  - ***Capacity miss*** occurs if program then tries to access Y again
  - X will be placed in a particular set based on its address
- **In a *direct mapped* cache, there is only one place to put X**
- **In an *associative cache*, there are multiple ways where X could go in the set.**
- **How to choose Y to minimize chance of needing it again?**
  - Least recently used (LRU) replacement: the least recently used block in a set is evicted when the cache is full.

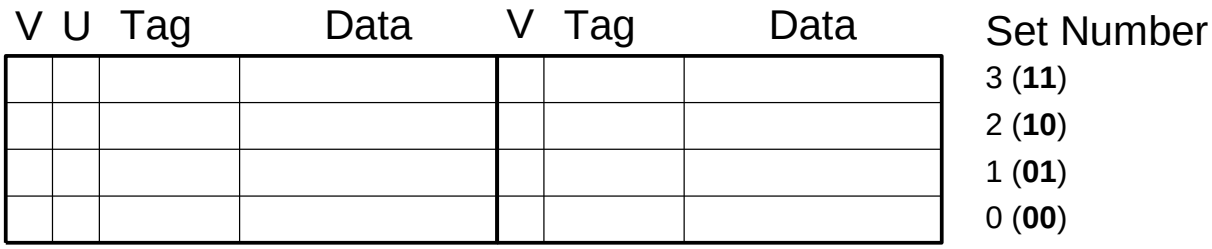
Types of Misses

- *Compulsory:* **first time data is 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**

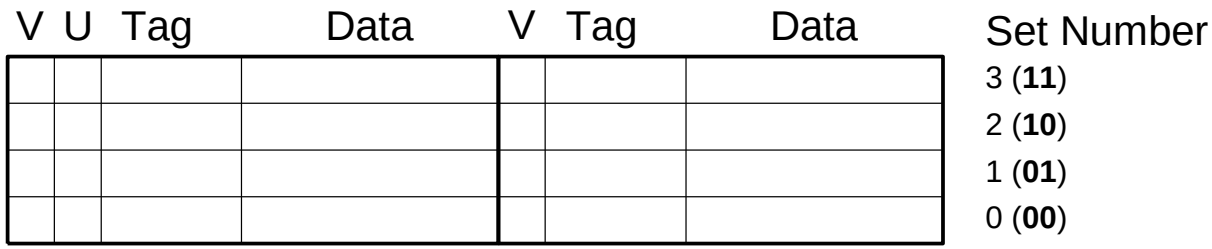
LRU Replacement

```
# MIPS assembly

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

(a)

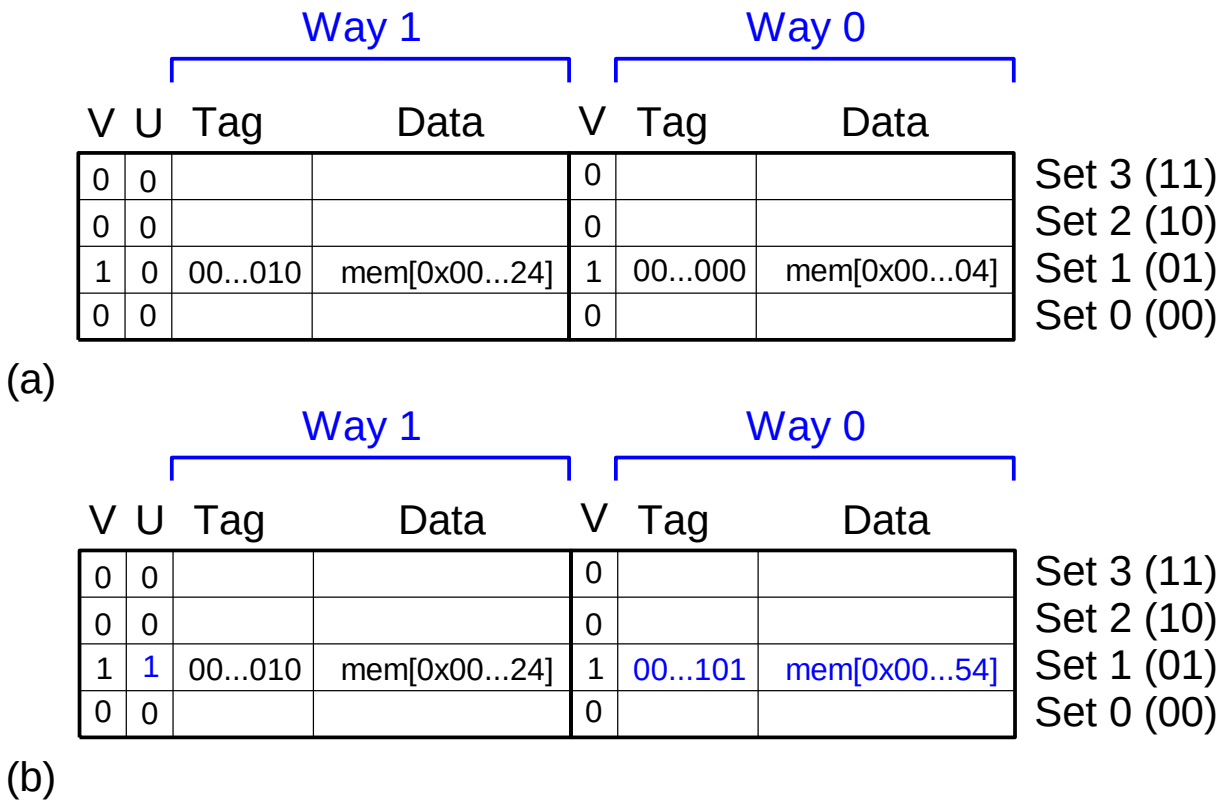


(b)



LRU Replacement

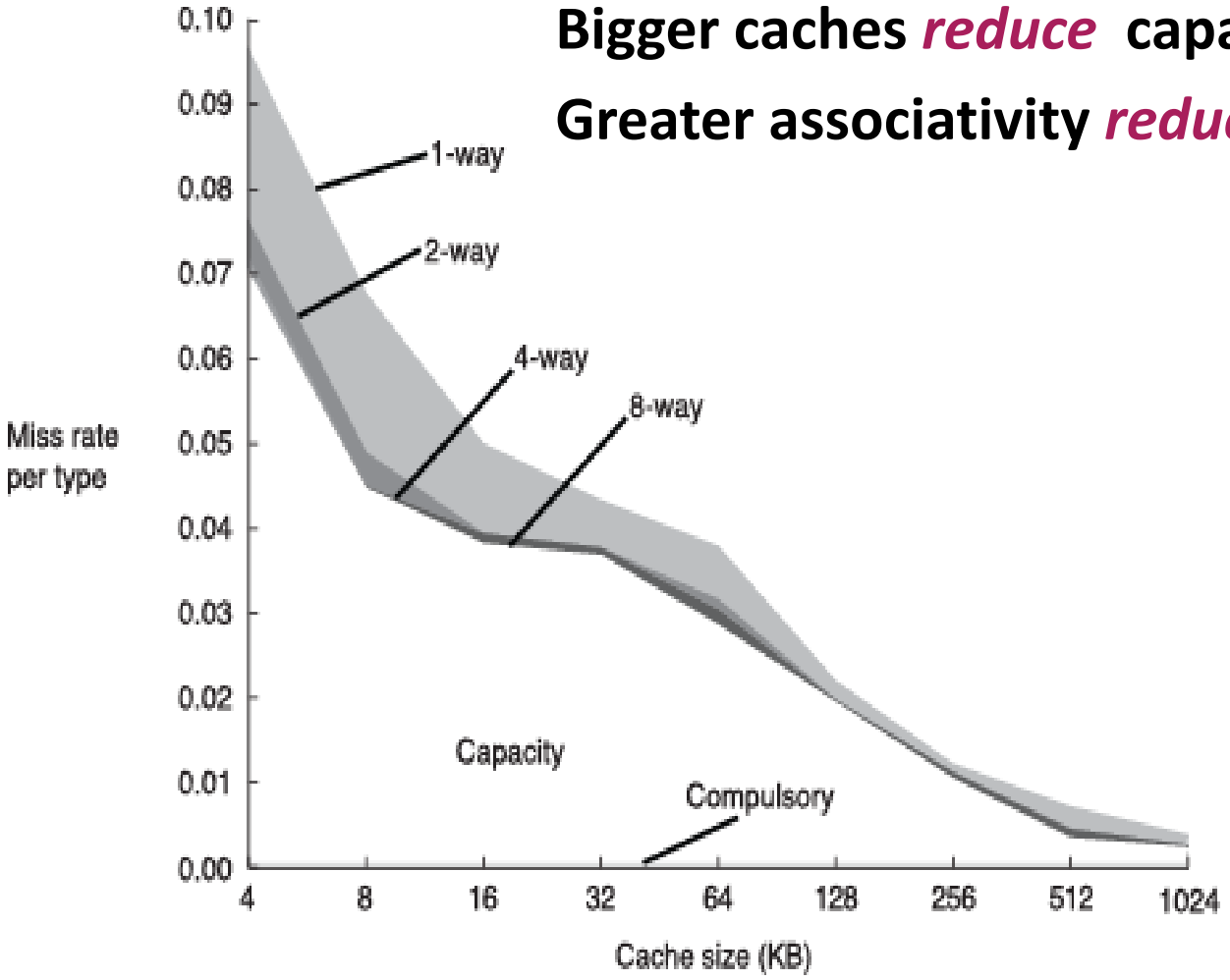
```
# MIPS assembly

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



Caching Summary

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

Miss Rate Data

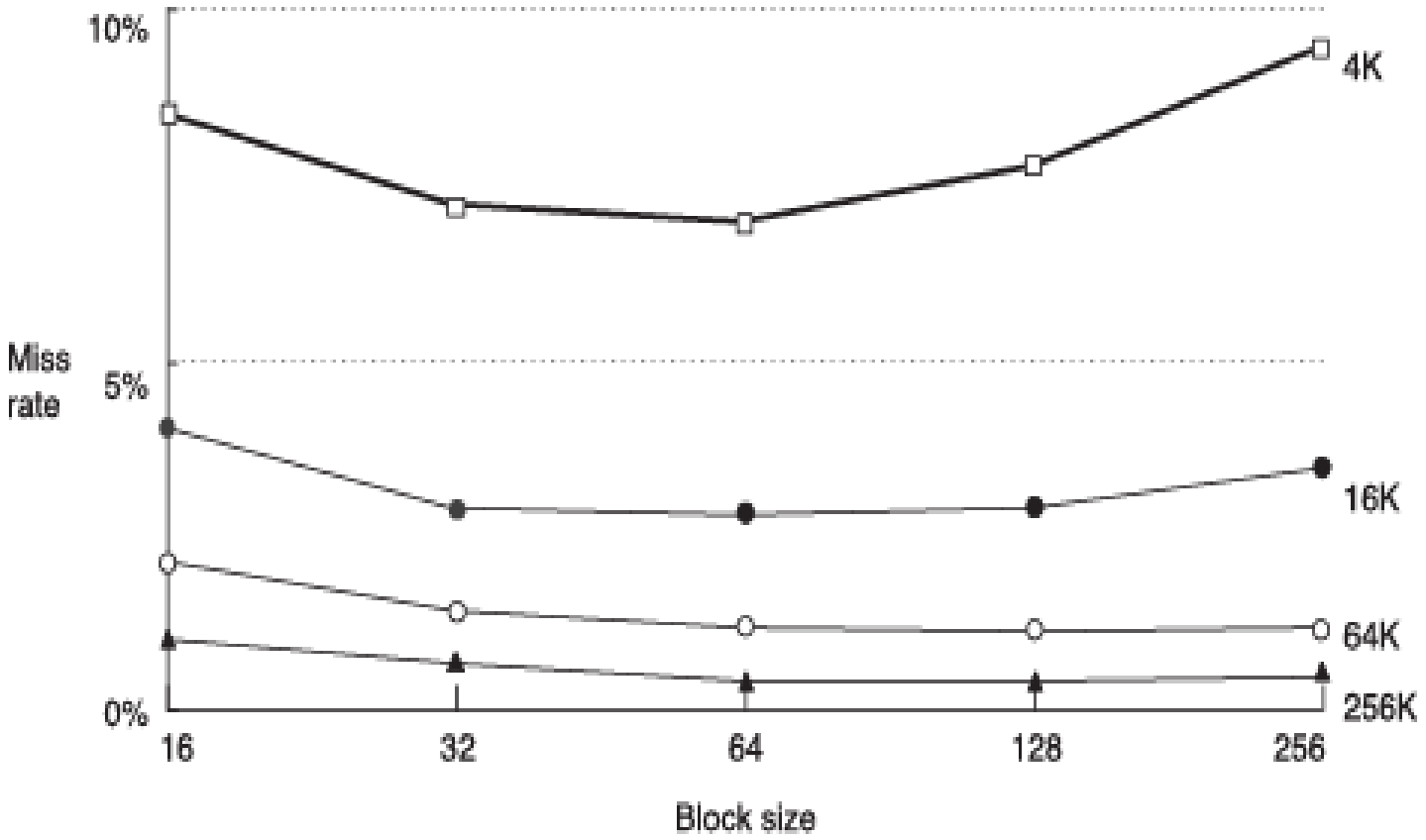


Bigger caches *reduce* capacity misses

Greater associativity *reduces* conflict misses

Adapted from Patterson & Hennessy, *Computer Architecture: A Quantitative Approach*

Miss Rate Data

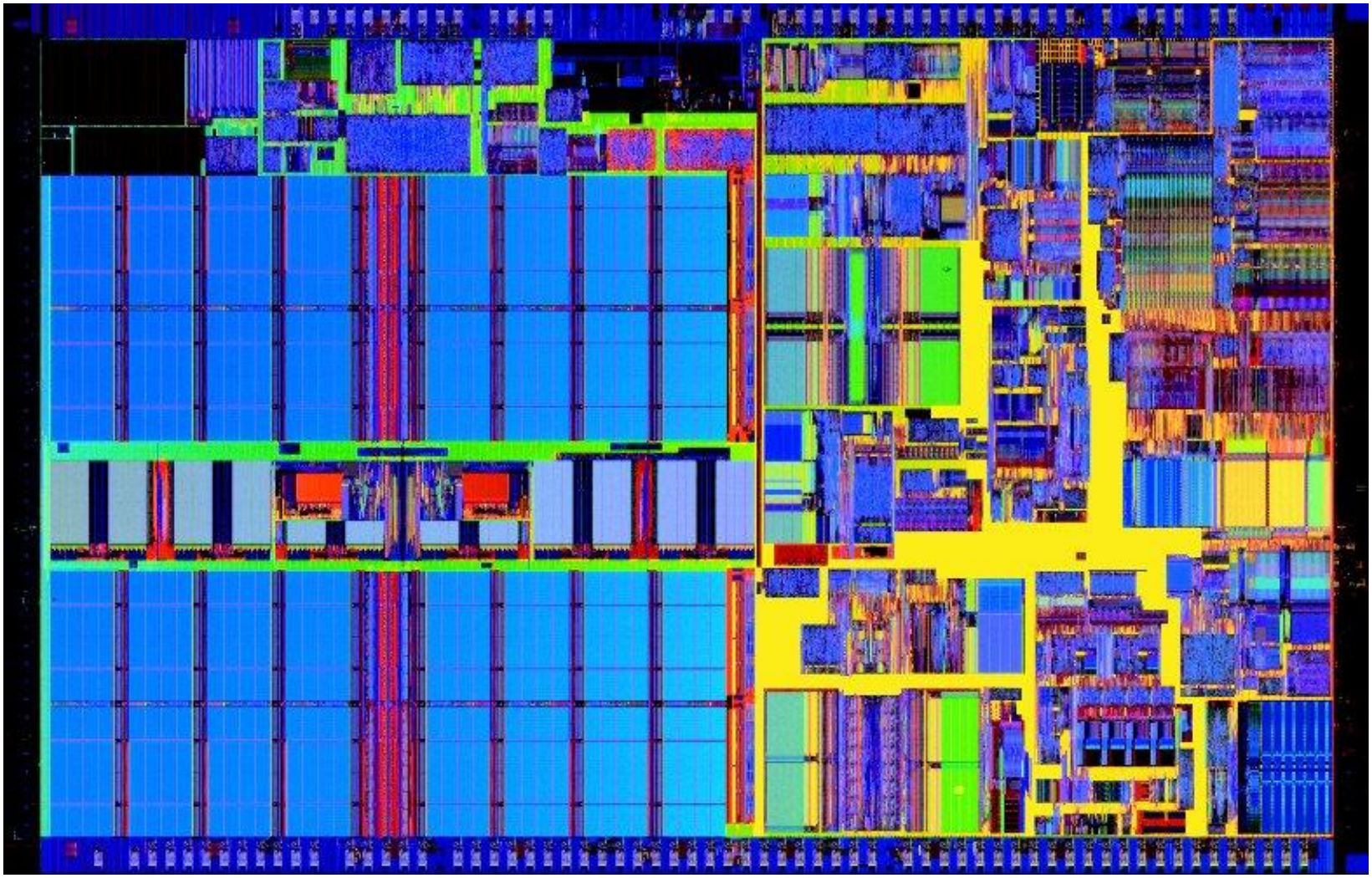


- Bigger block size *reduces* compulsory misses
- Bigger block size *increases* conflict misses

Multilevel Caches

- **Larger caches have lower miss rates, longer access times**
- **Expand the 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)
- **Even more levels are possible**

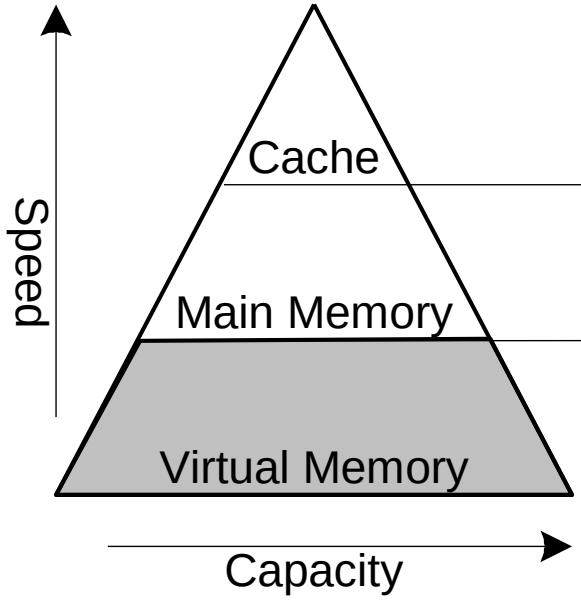
Intel Pentium III Die



Virtual Memory

- **Gives the illusion of a bigger memory without the high cost of DRAM**
- **Main memory (DRAM) acts as cache for the hard disk**

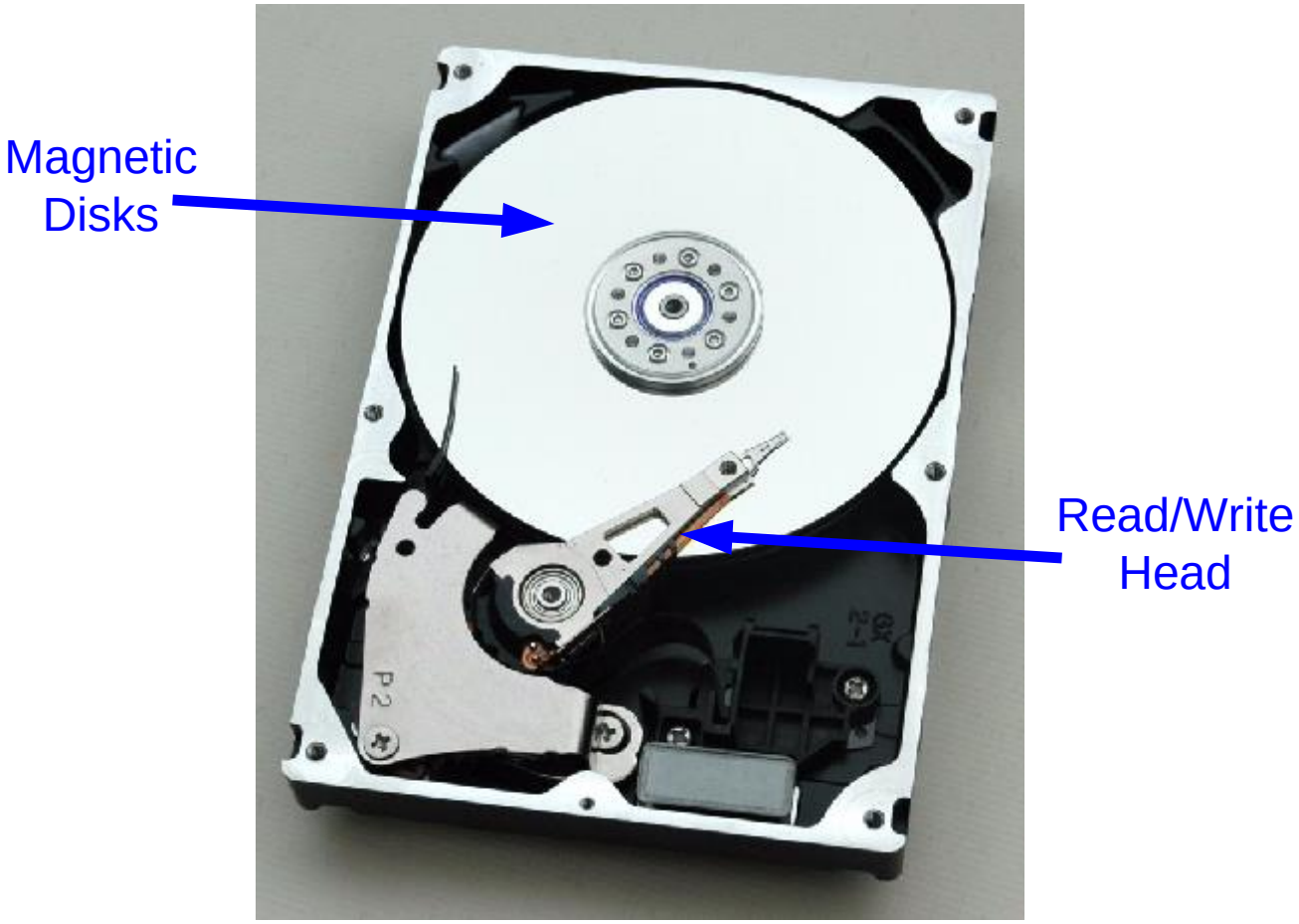
The Memory Hierarchy



Technology	cost / GB	Access time
SRAM	~ $10,000	~ 1 ns
DRAM	~ $100	~ 100 ns
Hard Disk	~ $1	~ 10,000,000 ns

- **Physical Memory: DRAM (Main Memory)**
  - Faster, Not so large, Expensive
- **Virtual Memory: Hard disk**
  - Slow, Large, Cheap

The Hard Disk



Virtual Memory

- **Each program uses *virtual addresses***
  - Entire virtual address space stored on a hard disk.
  - Subset of virtual address data in DRAM
  - CPU translates virtual addresses into ***physical addresses***
  - Data not in DRAM is fetched from the hard disk

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

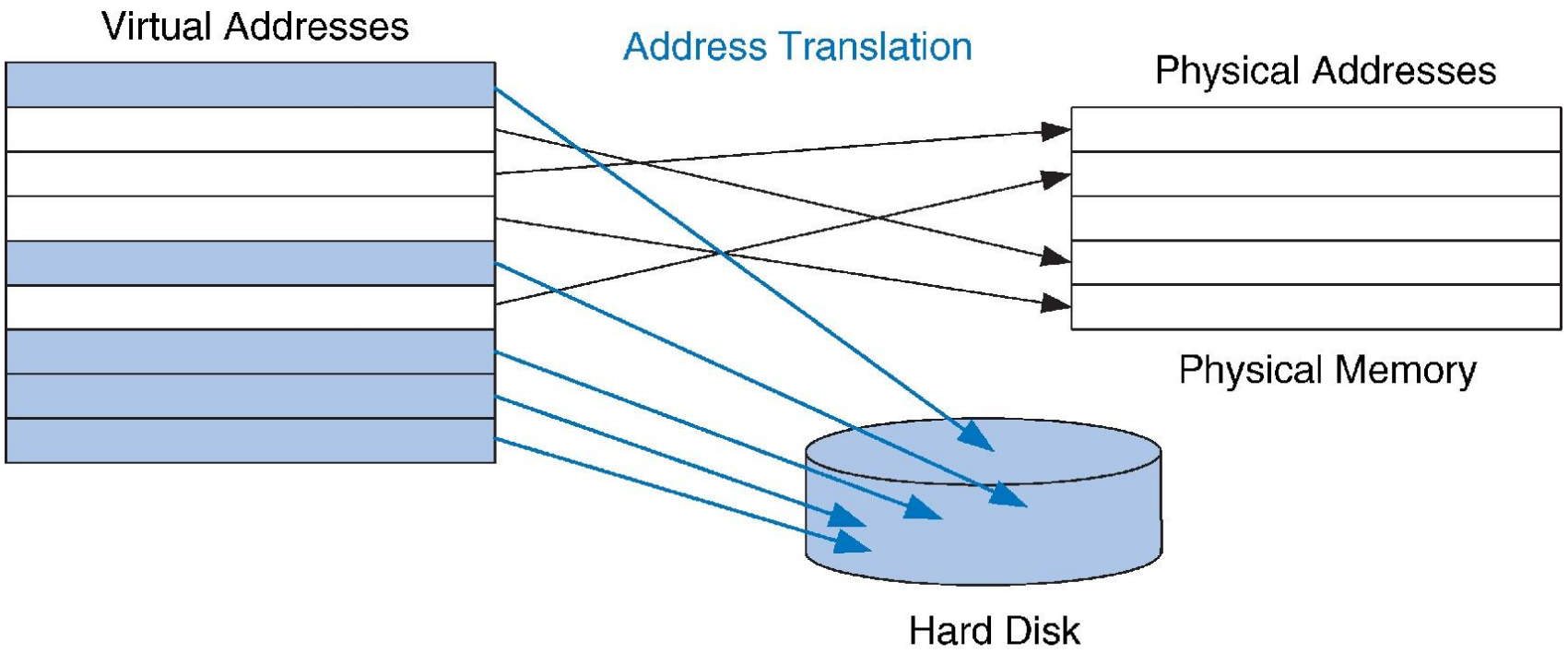
Cache/Virtual Memory Analogues

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

Virtual Memory Definitions

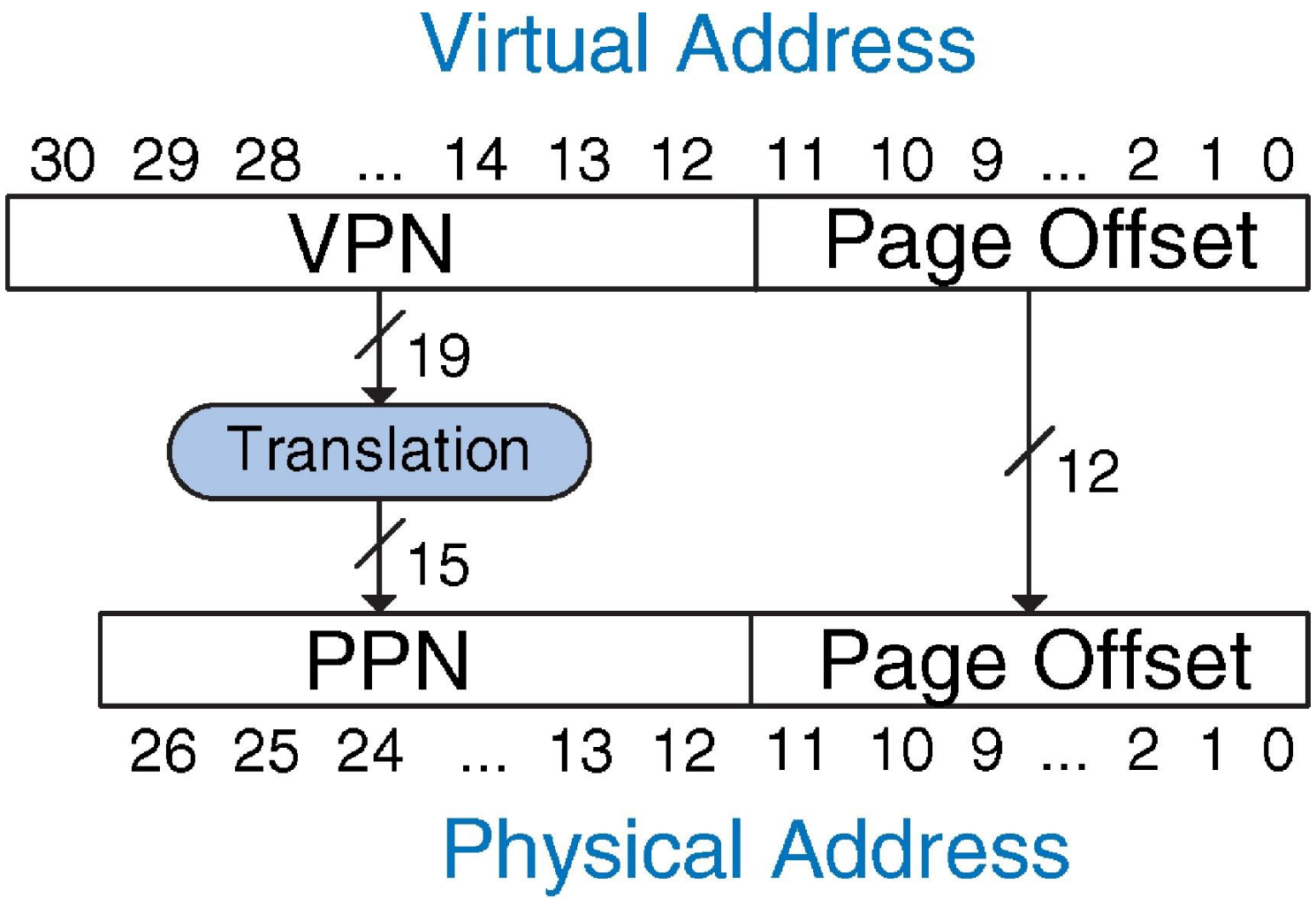
- ***Page size:*** **amount of memory transferred from hard disk to DRAM at once**
- ***Address translation:*** **determining the physical address from the virtual address**
- ***Page table:*** **lookup table used to translate virtual addresses to physical addresses**

Virtual and Physical Addresses



- **Most accesses hit in physical memory**
- **But programs have the large capacity of virtual memory**

Address Translation



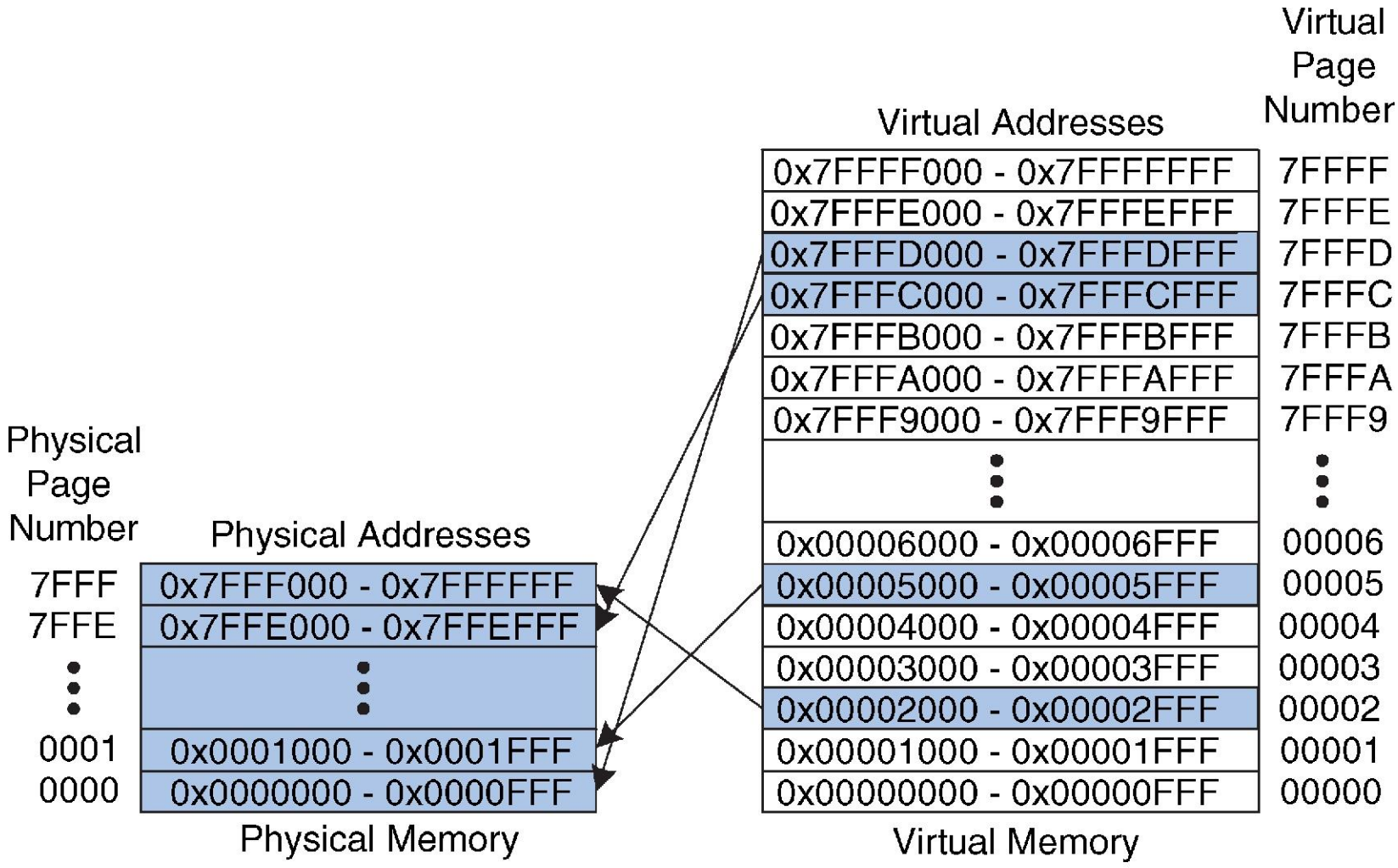
Virtual Memory Example

- **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 = $2^{31}/2^{12}$ = $\mathbf{2^{19}}$ (VPN = 19 bits)
  - # Physical pages = $2^{27}/2^{12}$ = $\mathbf{2^{15}}$ (PPN = 15 bits)

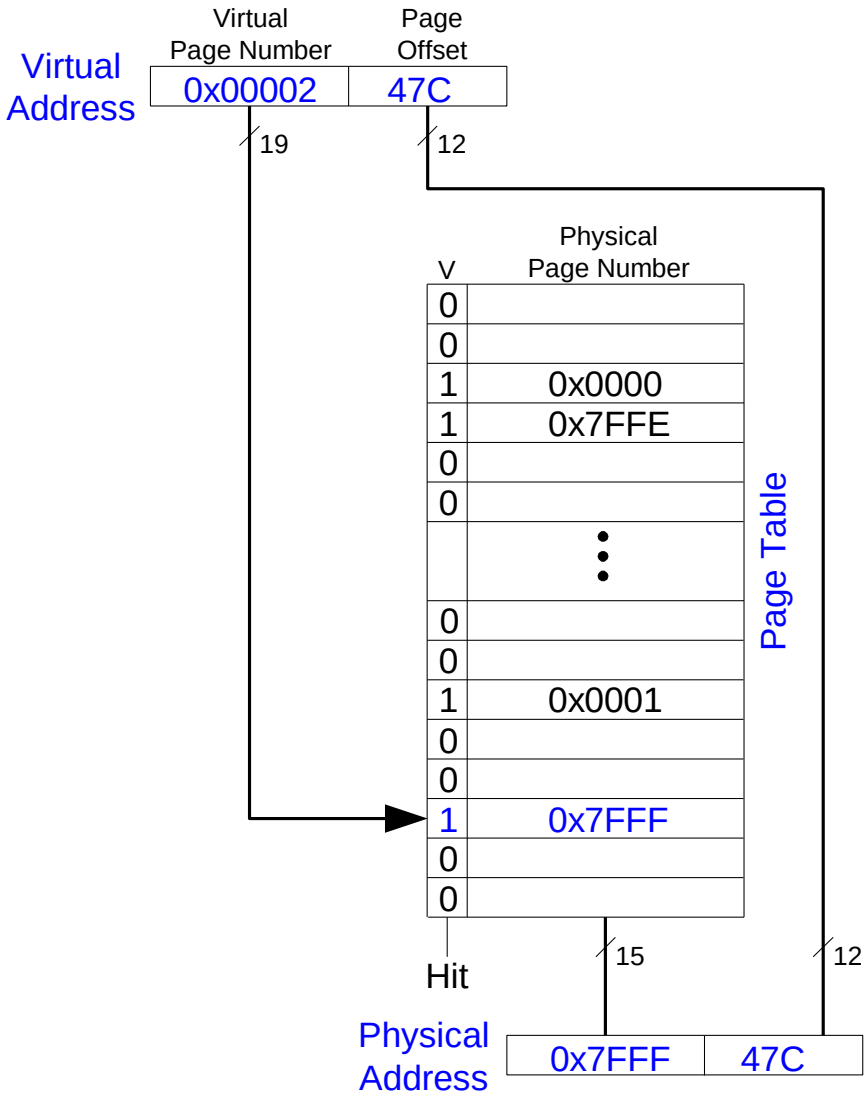
Virtual Memory Example



How do we translate addresses?

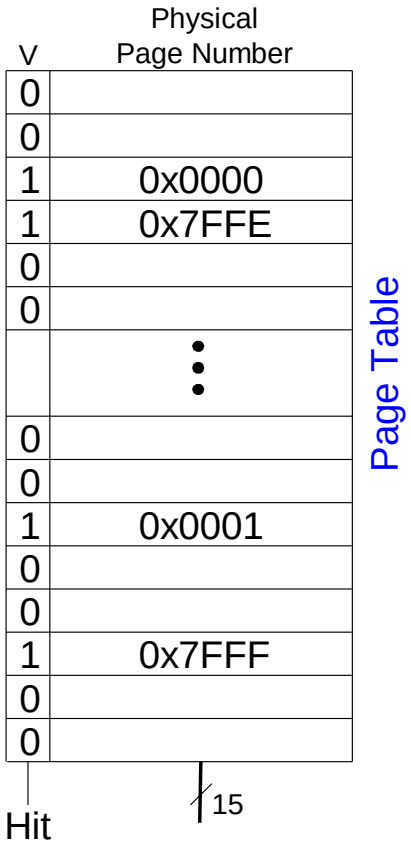
- **Page table**
  - Has entry for each virtual page
  - Each entry has:
    - ***Valid bit:*** whether the virtual page is located in physical memory (if not, it must be fetched from the hard disk)
    - ***Physical page number:*** where the page is located

Page Table Example



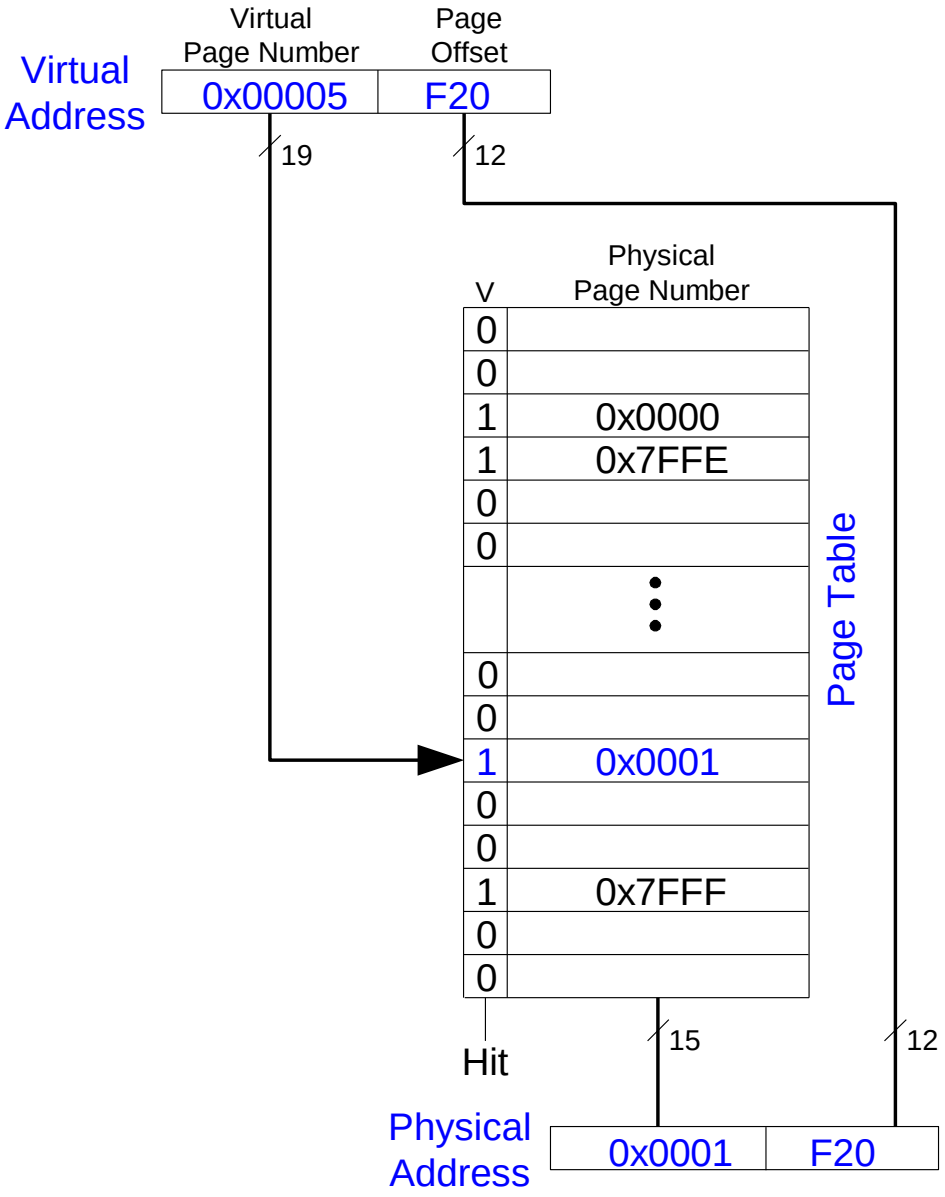
Page Table Example 1

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



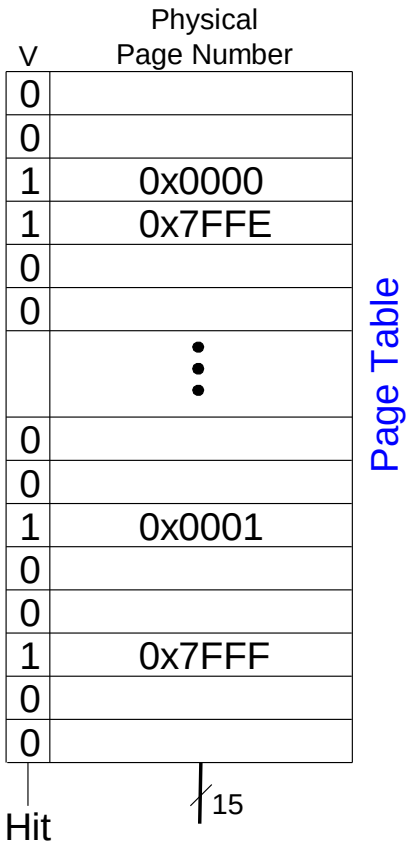
Page Table Example 1

- **What is the physical address of virtual address `0x5F20`?**
  - VPN = 5
  - Entry 5 in page table indicates VPN 5 is in physical page 1
  - Physical address is `0x1F20`



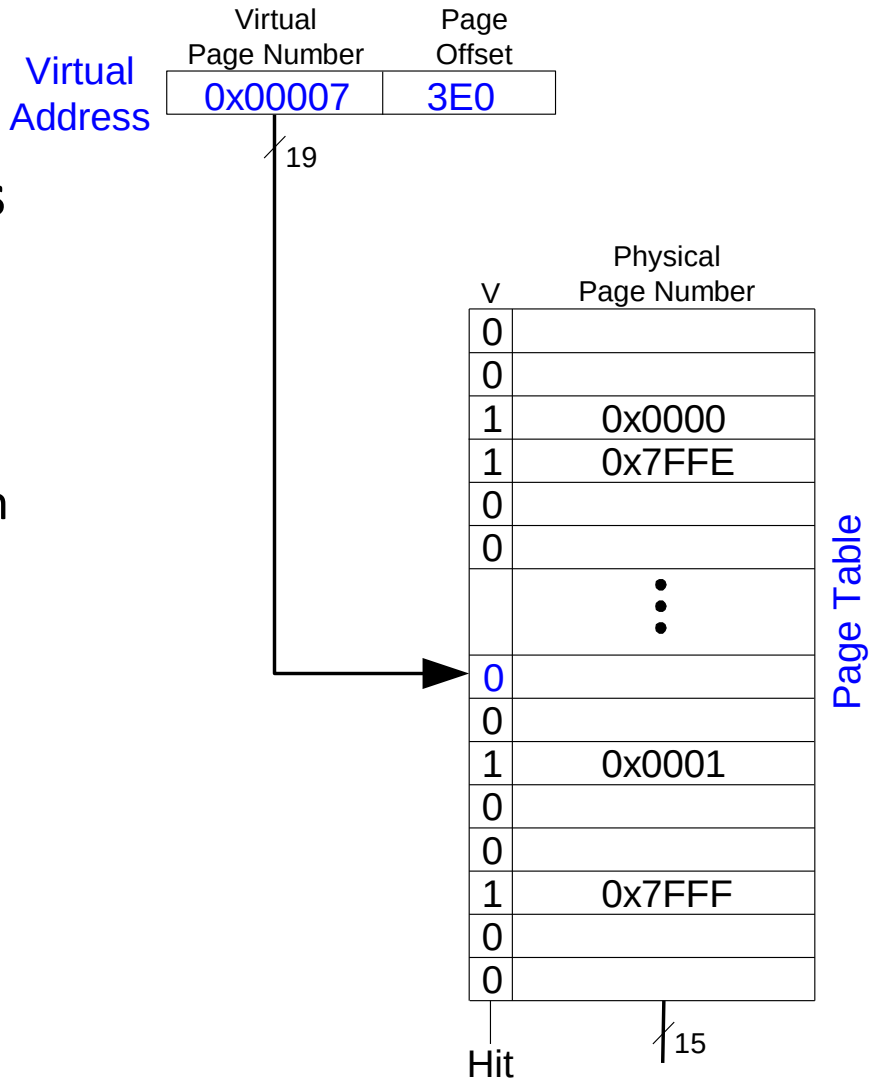
Page Table Example 2

- **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 in page table is invalid, so the page is not in physical memory
  - The virtual page must be swapped into physical memory from disk



Page Table Challenges

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

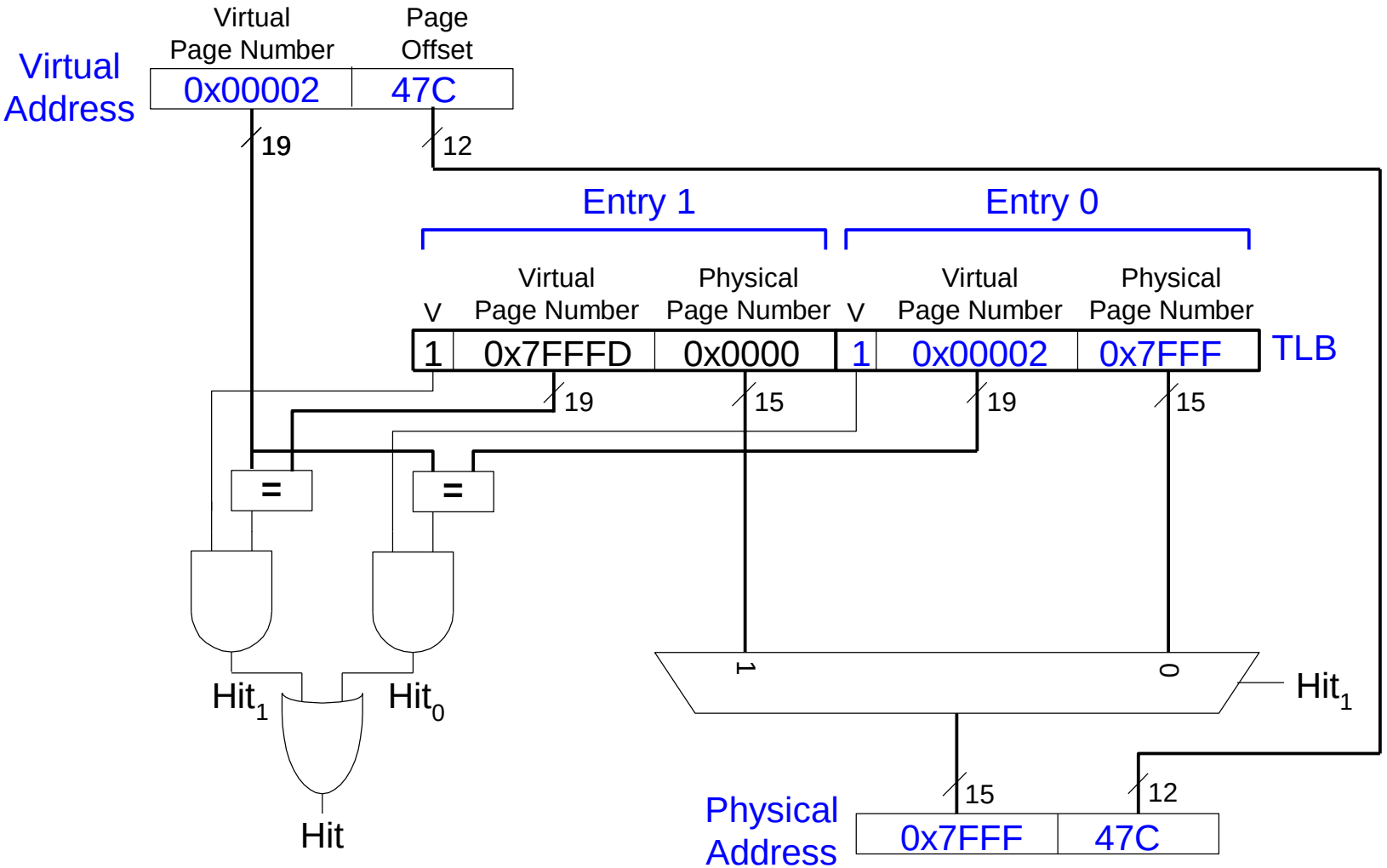
Translation Lookaside Buffer (TLB)

- **Use a *translation lookaside buffer* (TLB)**
  - Small cache of most recent translations
  - Reduces number of memory accesses required for *most* loads/stores from two to one

Translation Lookaside Buffer (TLB)

- **Page table accesses have a lot of temporal locality**
  - Data accesses have temporal and spatial 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 and stores from 2 to 1

Example Two-Entry TLB



Memory Protection

- **Multiple programs (*processes*) run at once**
  - Each process has its own page table
  - Each process can use entire virtual address space without worrying about where other programs are
- **A process can only access physical pages mapped in its page table – can't overwrite memory from another process**

Virtual Memory Summary

- Virtual memory increases *capacity*
- A subset of virtual pages are located in physical memory
- A *page table* maps virtual pages to physical pages – this is called address translation
- A *TLB* speeds up address translation
- Using different page tables for different programs provides *memory protection*