

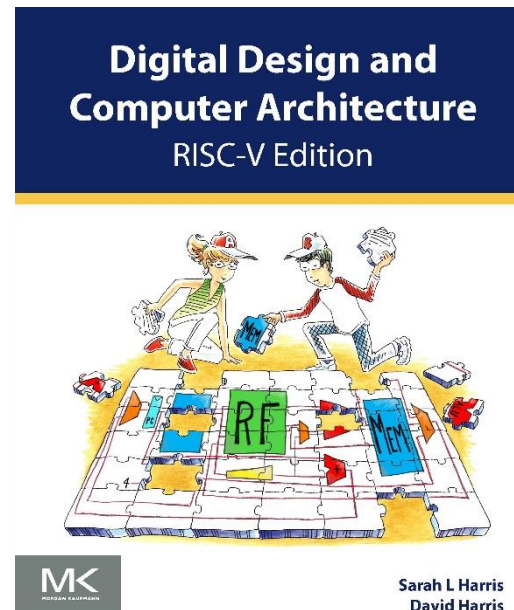


Computer Systems

Memory Systems

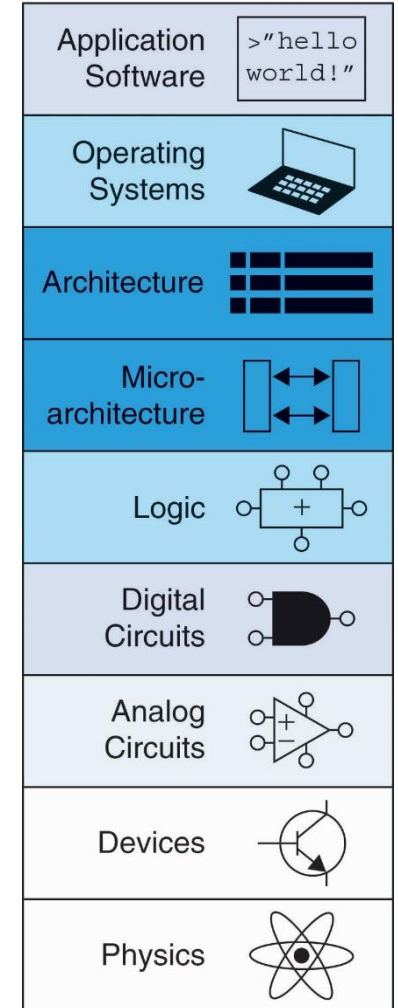
Markus Bader

SS2025



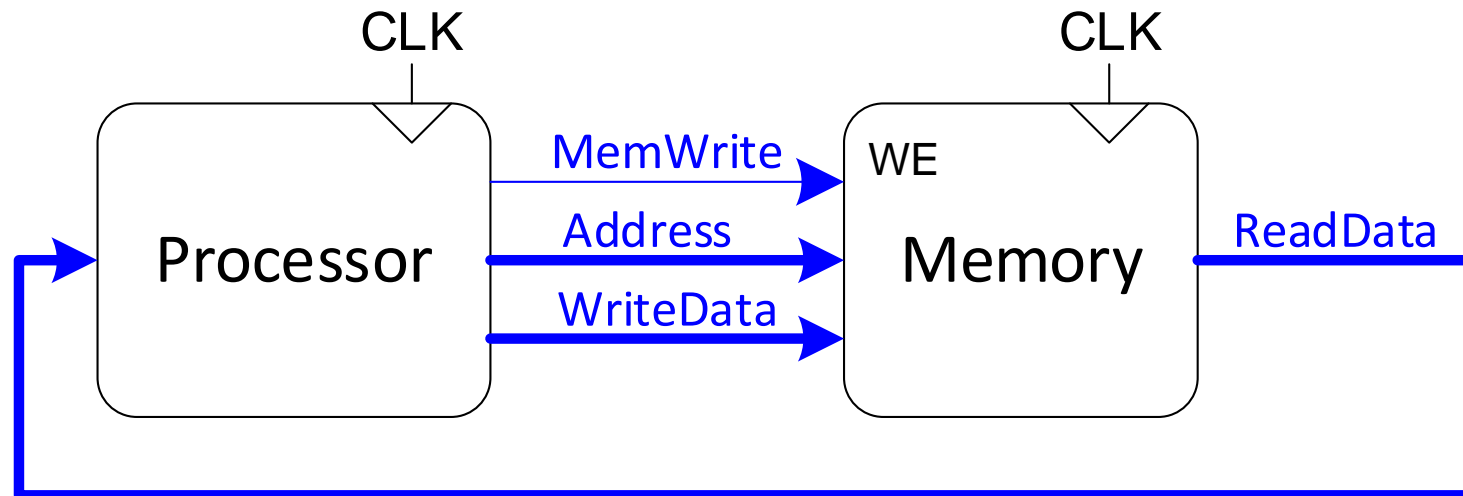
- Literature: „Digital Design and Computer Architecture: RISC-V Edition“, by Sarah L. Harris and David Harris
 - <https://shop.elsevier.com/books/digital-design-and-computer-architecture-risc-v-edition/harris/978-0-12-820064-3>
 - <https://pages.hmc.edu/harris/ddca/ddcarv.html> (Includes resources for students!)
 - They also provide slideshows – the basis for ours! You can investigate extended version at their website.
- Available at TU's library: https://catalogplus.tuwien.at/permalink/f/qknpf/UTW_alma21139903990003336

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



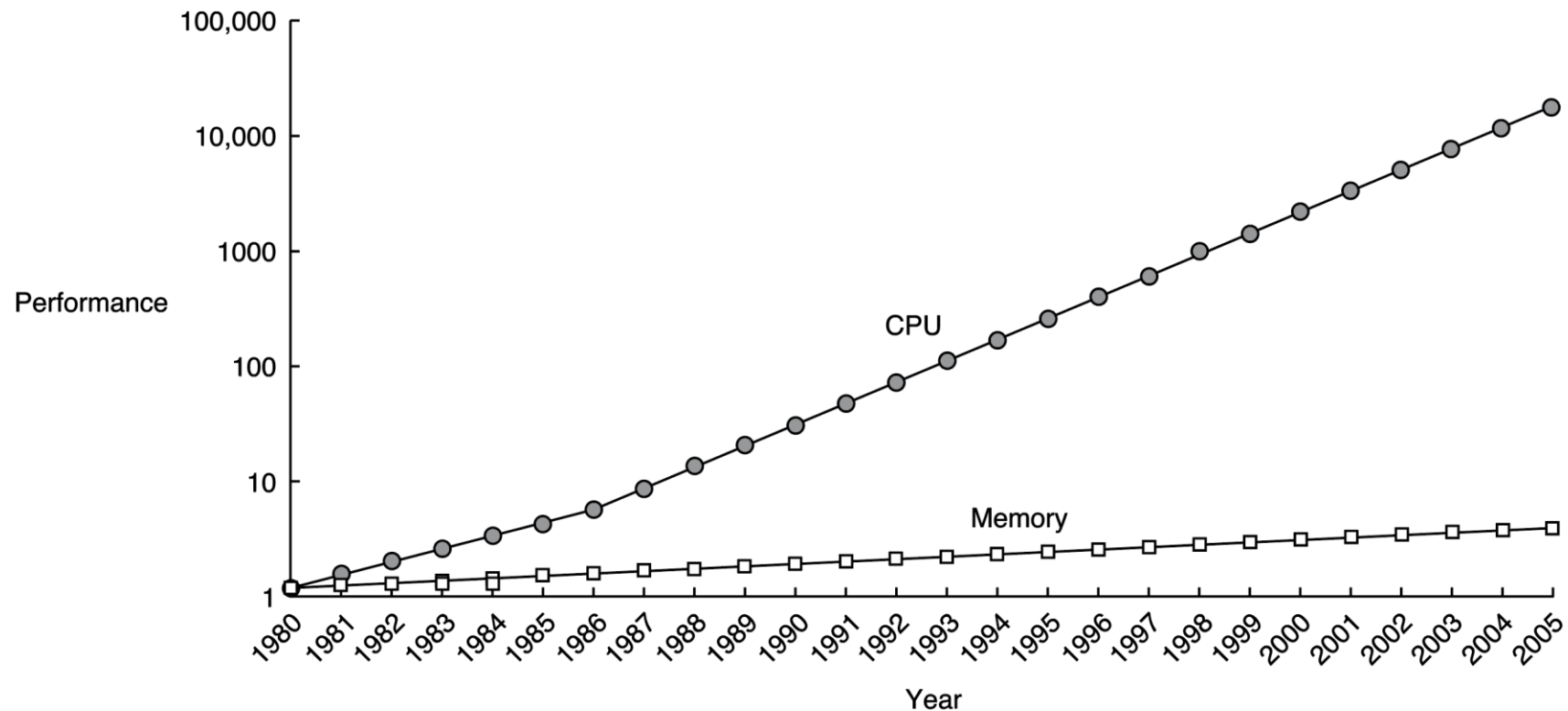
- Computer performance depends on:
 - **Processor** performance
 - **Memory system** performance

Processor / Memory Interface:



Processor-Memory Gap

- In prior chapters, we assumed access memory in 1 clock cycle
- This assumption hasn't been true since the 1980's.

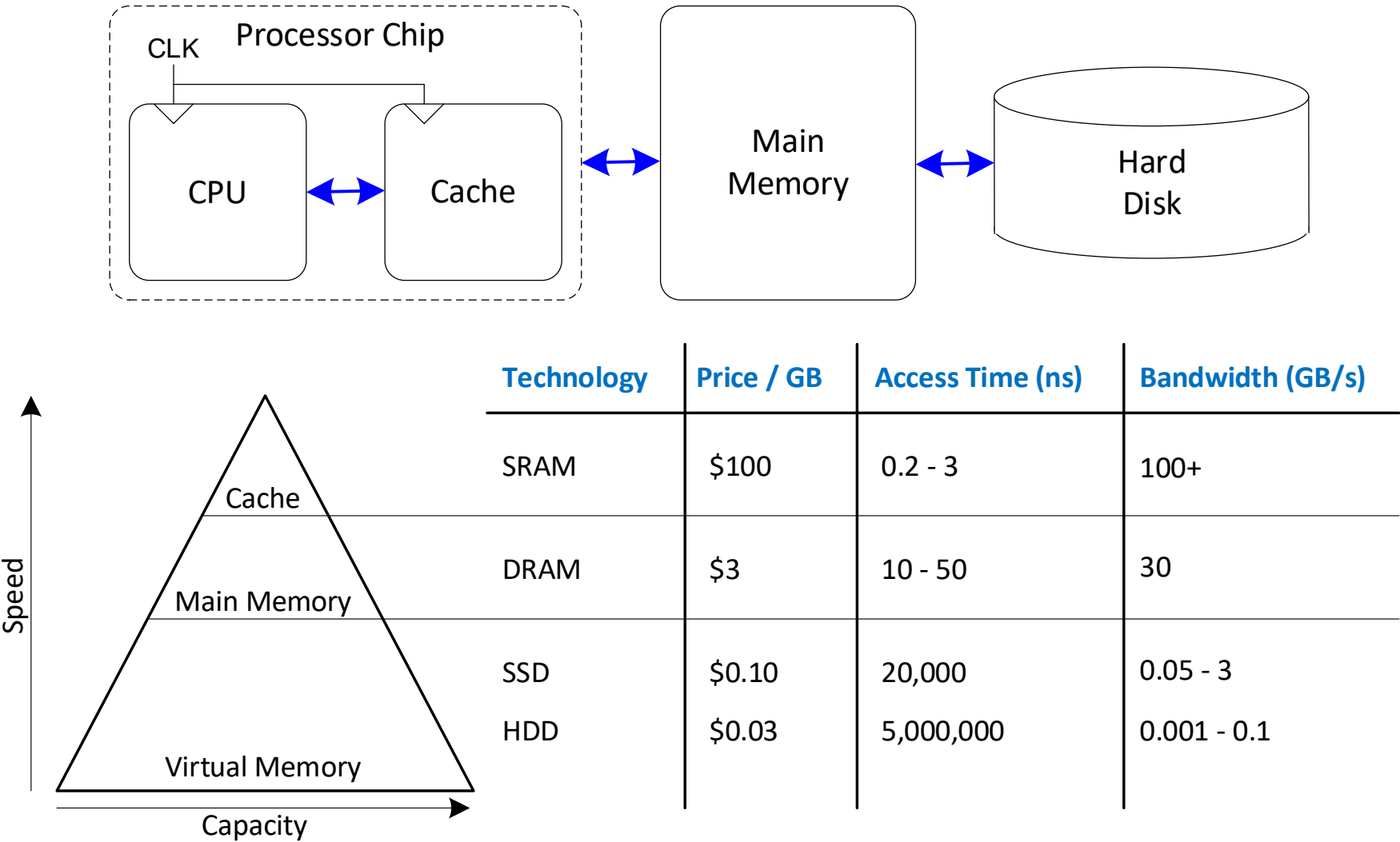


Memory System Challenge

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



Memory Hierarchy



- 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

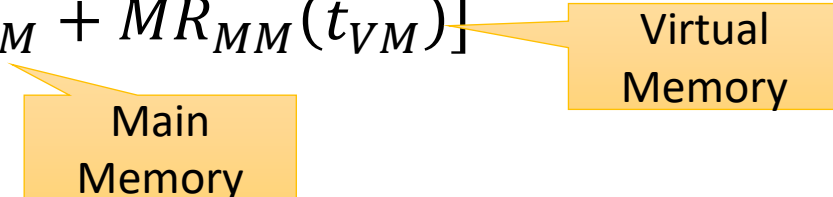
Memory Performance

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

$$\textit{Hit Rate} = \frac{\# \textit{ hits}}{\# \textit{memory accesses}} = 1 - \textit{Miss Rate}$$

$$\textit{Miss Rate} = \frac{\# \textit{ misses}}{\# \textit{memory accesses}} = 1 - \textit{Hit Rate}$$

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

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


The diagram shows two yellow callout boxes. One box labeled "Main Memory" has a pointer to the $t_{\textit{MM}}$ term in the formula. Another box labeled "Virtual Memory" has a pointer to the $t_{\textit{VM}}$ term in the formula.

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 cache hit and miss rates?

$$\text{Hit Rate} = \frac{1250}{2000} = 0.625$$

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

Memory Performance Example 2

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

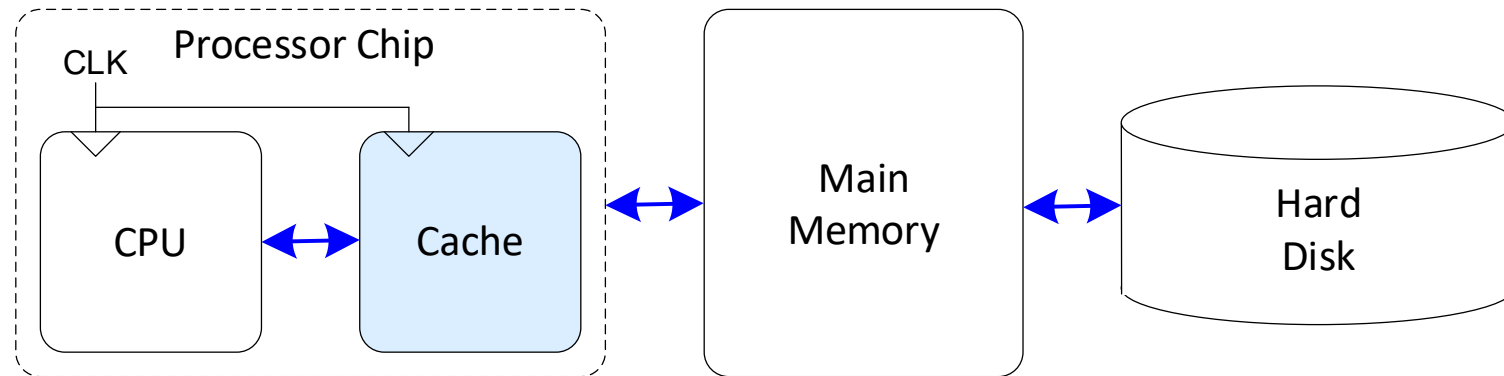
$$AMAT = t_{cache} + MR_{cache}(t_{MM}) = [1 + 0.375(100)] \text{ cycles} = 38.5 \text{ cycles}$$

- **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



Caches

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



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

We focus on data loads, but stores follow the 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

- **Capacity (C)**
 - Number of data bytes in cache
- **Block size (b)**
 - Bytes of data brought into cache at once
- **Number of blocks (B)**
 - Number of blocks in cache
 - $B = \frac{C}{b}$
- **Degree of associativity (N)**
 - Number of blocks in a set
- **Number of sets (S)**
 - Each memory address maps to exactly one cache set
 - $S = \frac{B}{N}$

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 ($B = 8$)

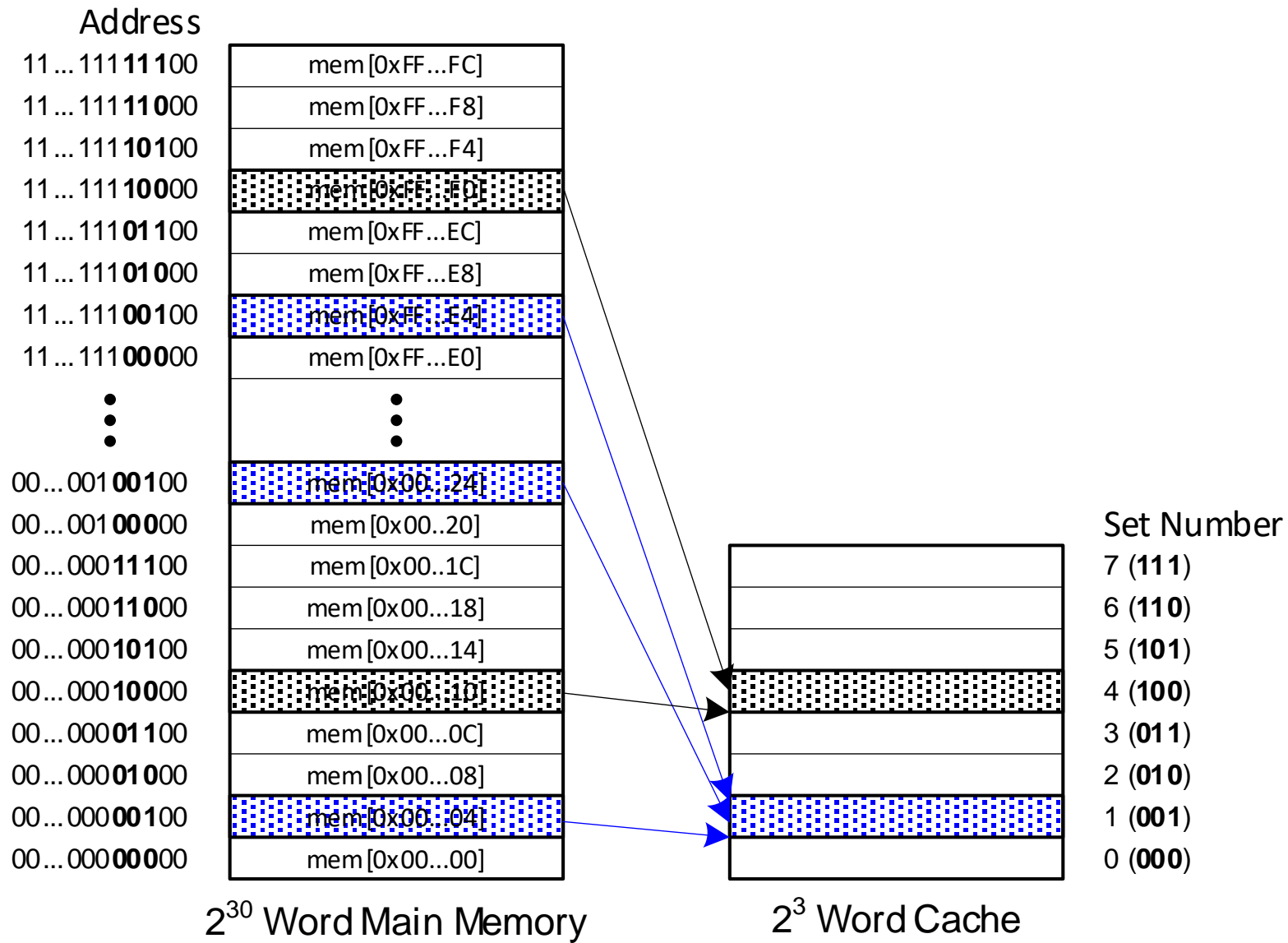
Example of Cache Parameters

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

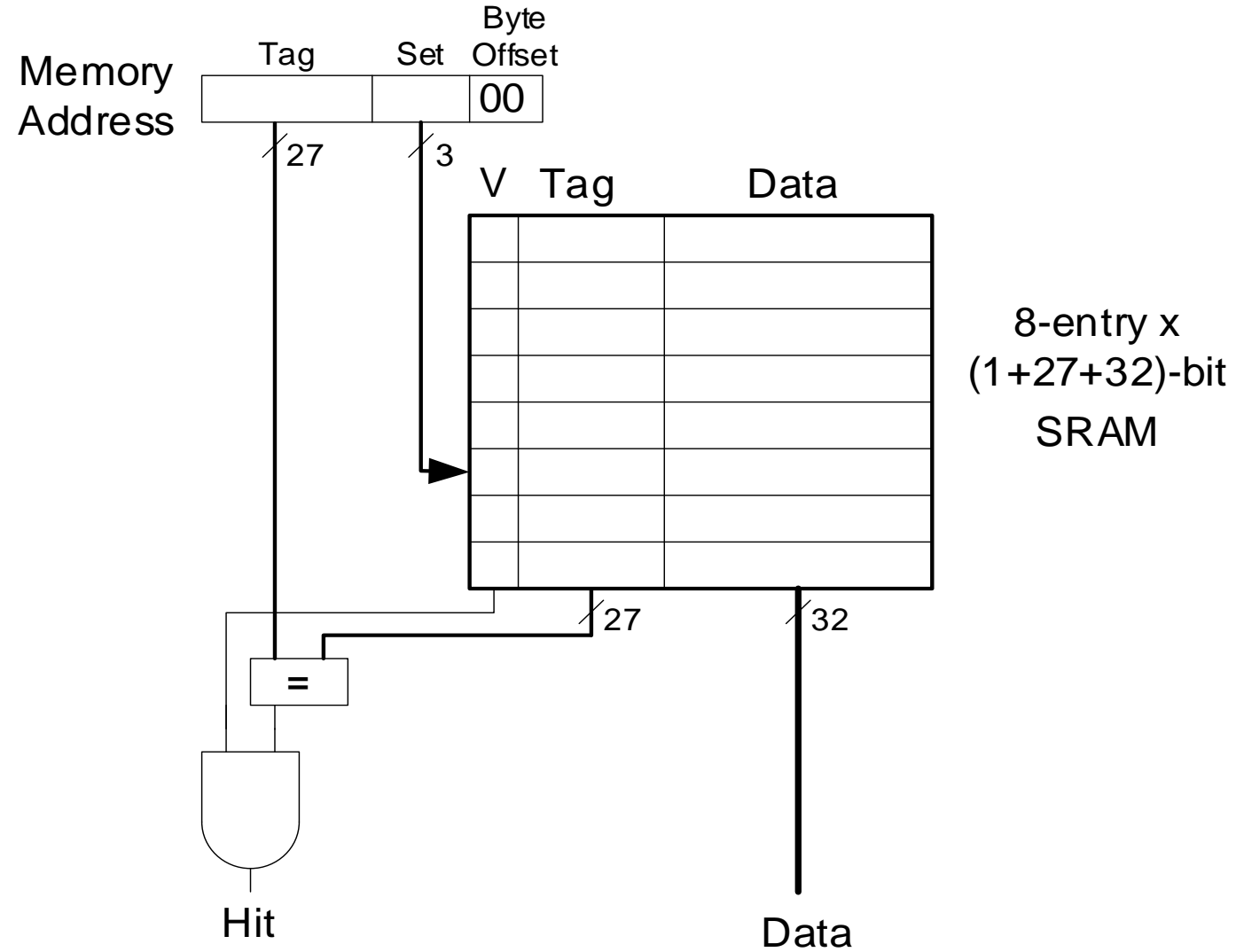
Ridiculously small, but will illustrate organizations

Direct-Mapped Caches

Direct-Mapped Cache



Direct-Mapped Cache Hardware



Direct-Mapped Cache Performance - Compulsory Misses

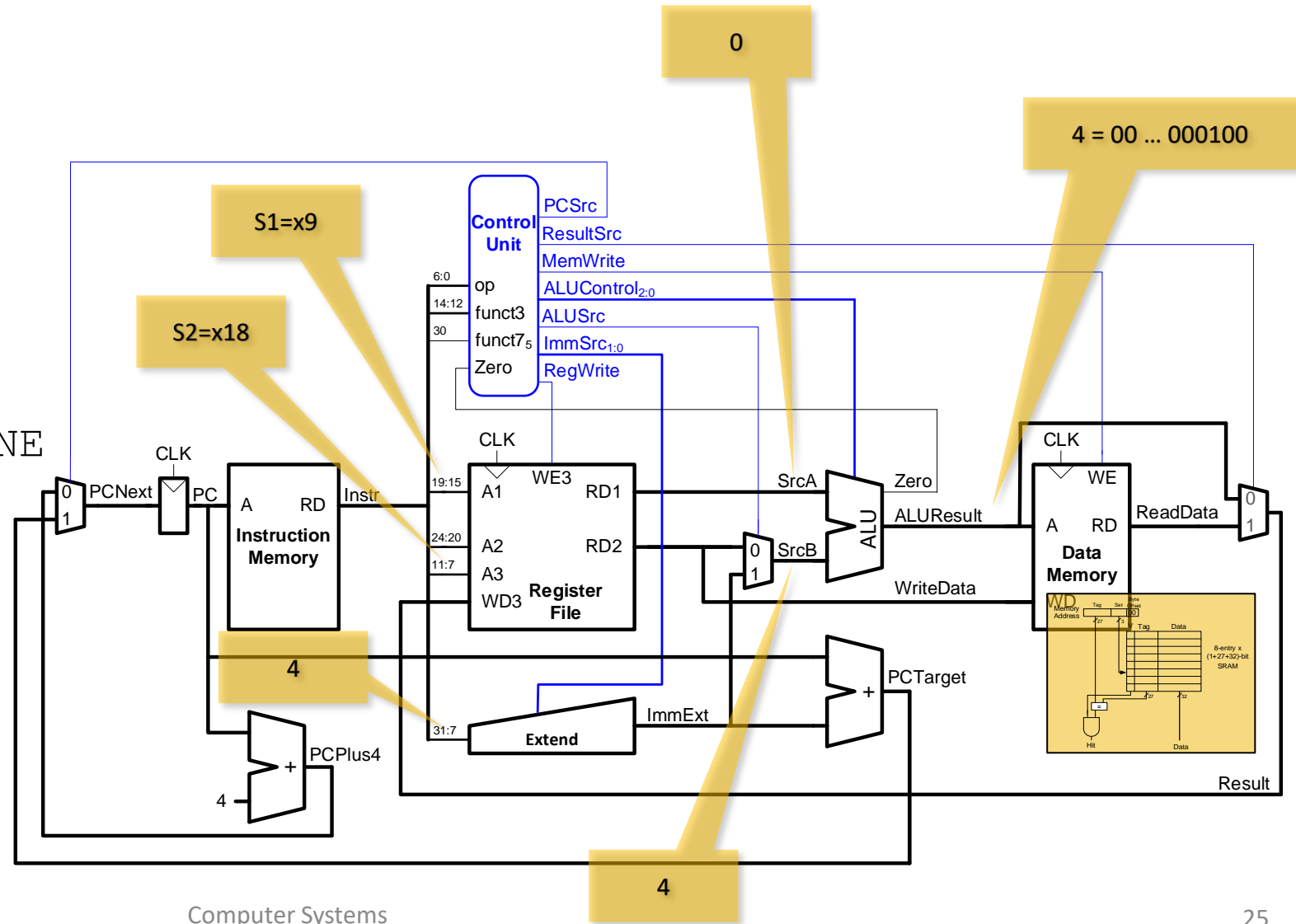
RISC-V assembly code

```
        addi s0, zero, 5
        addi s1, zero, 0
LOOP:   beq  s0, zero, DONE
        lw   s2, 4(s1)
        lw   s3, 12(s1)
        lw   s4, 8(s1)
        addi s0, s0, -1
        j    LOOP
DONE:
```


Direct-Mapped Cache Performance - Compulsory Misses

RISC-V assembly code

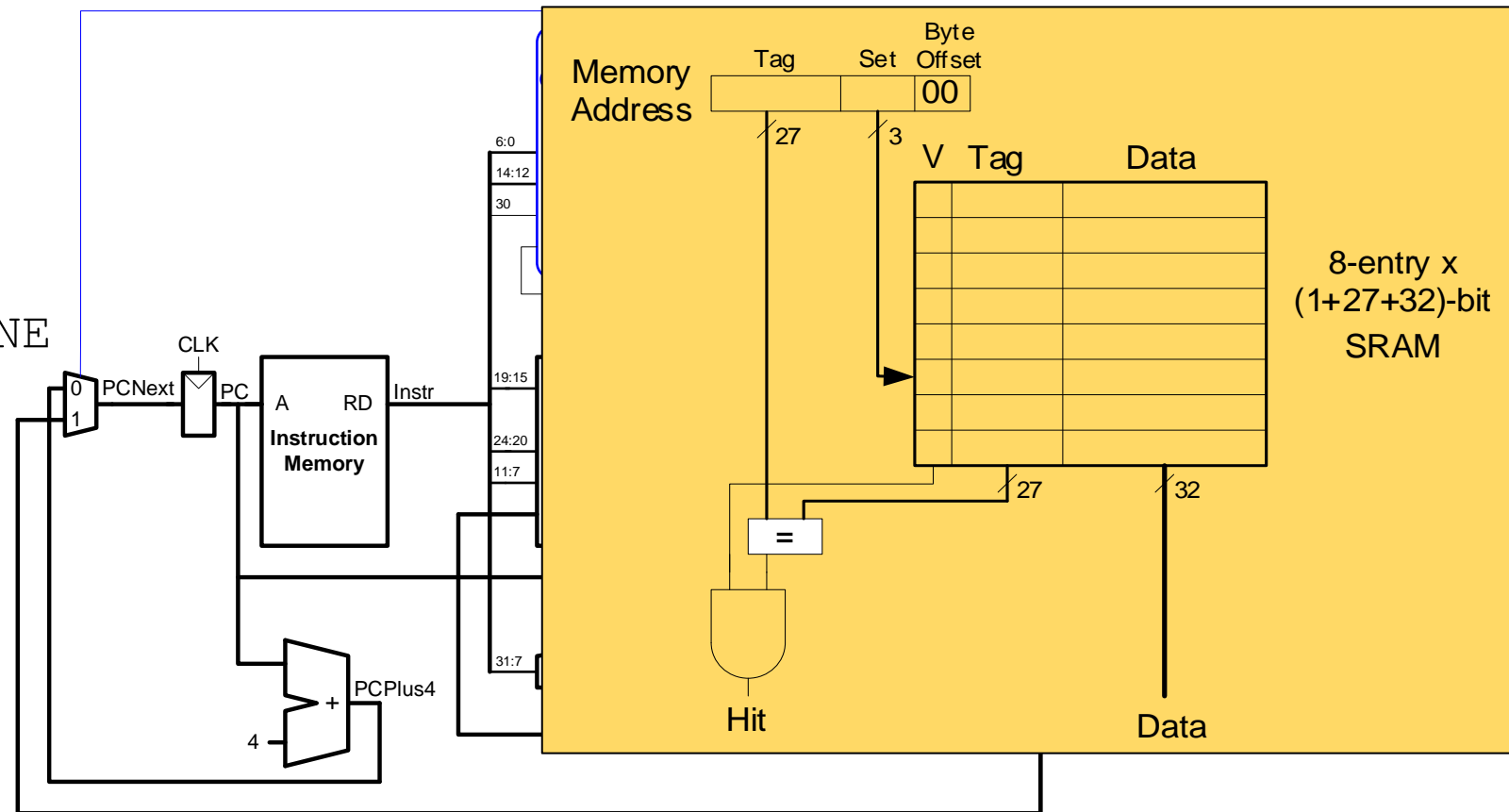
```
addi s0, zero, 5
addi s1, zero, 0
LOOP: beq s0, zero, DONE
      lw  s2, 4(s1)
      lw  s3, 12(s1)
      lw  s4, 8(s1)
      addi s0, s0, -1
      j   LOOP
DONE:
```



Direct-Mapped Cache Performance - Compulsory Misses

RISC-V assembly code

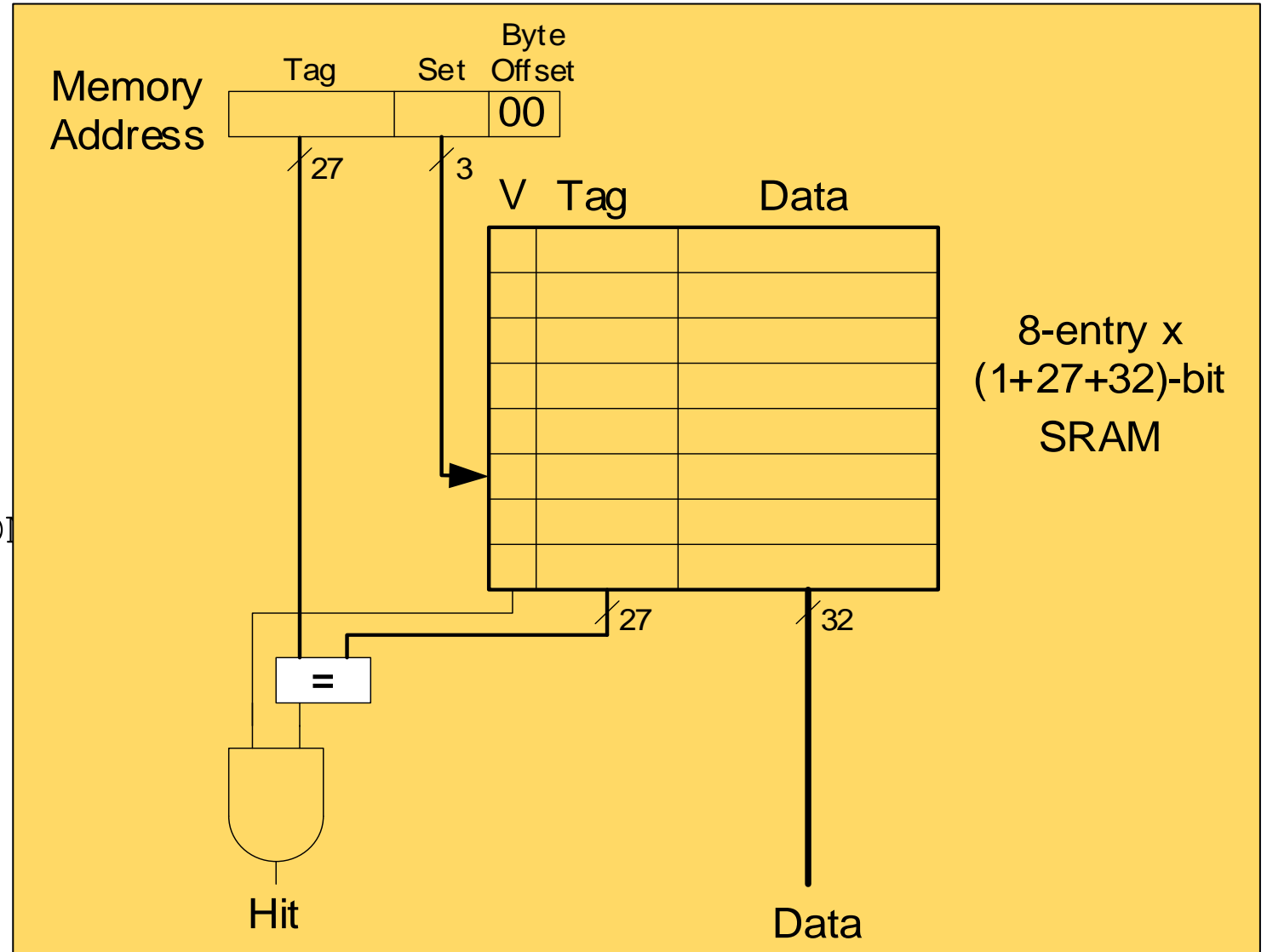
```
addi s0, zero, 5
addi s1, zero, 0
LOOP: beq s0, zero, DONE
      lw  s2, 4(s1)
      lw  s3, 12(s1)
      lw  s4, 8(s1)
      addi s0, s0, -1
      j   LOOP
DONE:
```



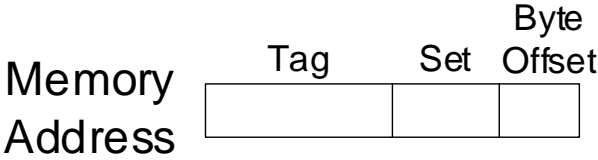
Direct-Mapped Cache Performance - Compulsory Misses

RISC-V assembly code

```
addi s0, zero, 5
addi s1, zero, 0
LOOP: beq s0, zero, DONE
      lw s2, 4(s1)
      lw s3, 12(s1)
      lw s4, 8(s1)
      addi s0, s0, -1
      j LOOP
DONE:
```



Direct-Mapped Cache Performance - Compulsory Misses



RISC-V assembly code

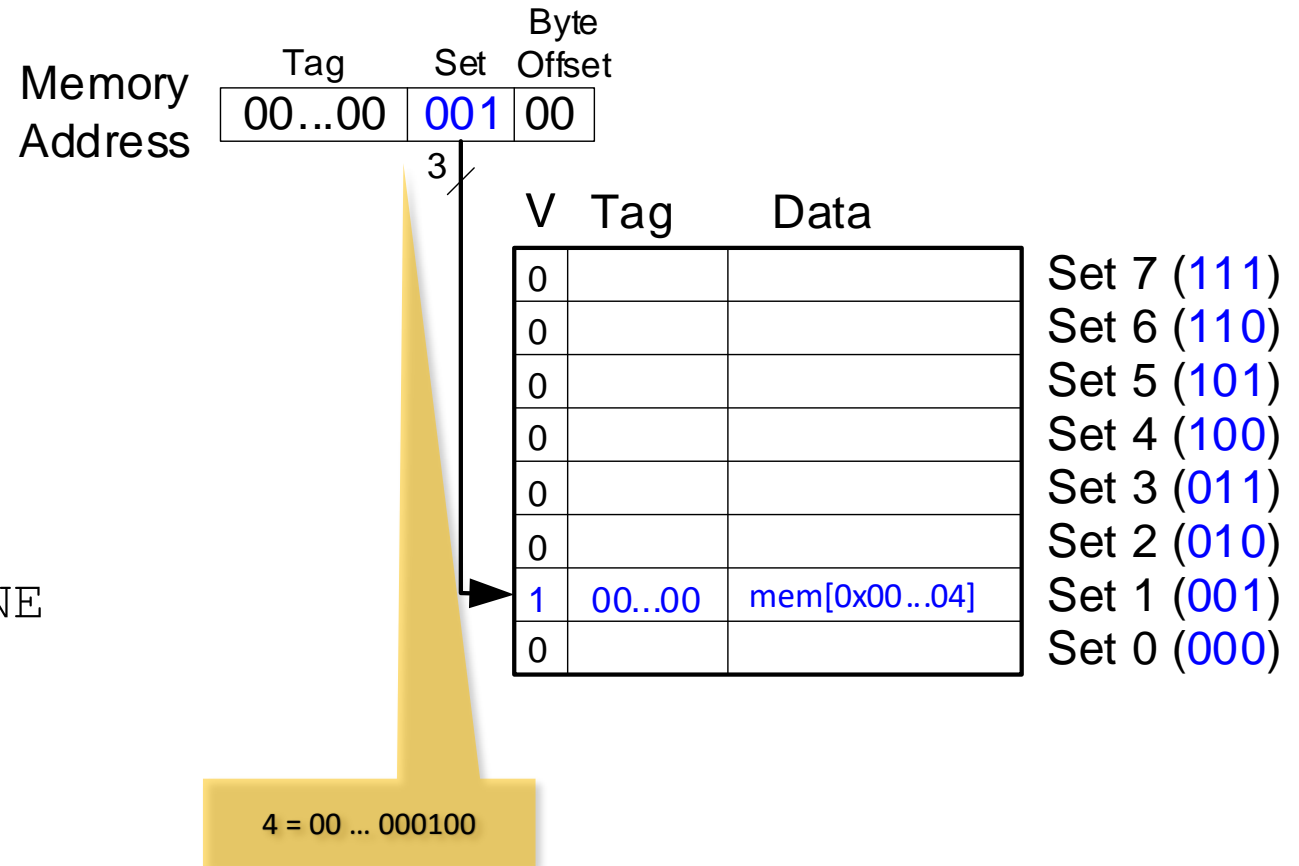
```
    addi s0, zero, 5
    addi s1, zero, 0
LOOP: beq s0, zero, DONE
    lw   s2, 4(s1)
    lw   s3, 12(s1)
    lw   s4, 8(s1)
    addi s0, s0, -1
    j    LOOP
DONE:
```

| 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) |
| 0 | | | Set 1 (001) |
| 0 | | | Set 0 (000) |

Direct-Mapped Cache Performance - Compulsory Misses

RISC-V assembly code

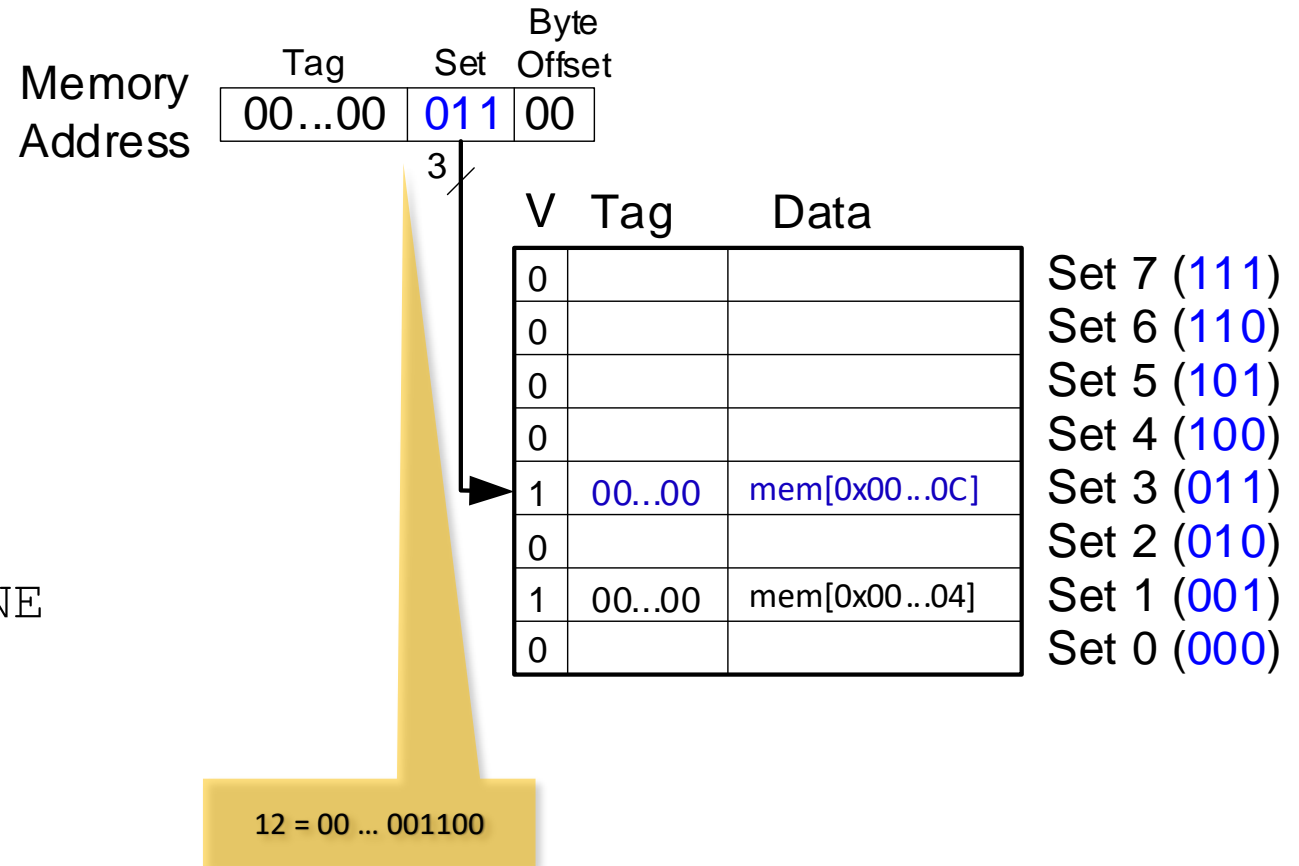
```
    addi s0, zero, 5
    addi s1, zero, 0
LOOP: beq  s0, zero, DONE
      lw   s2, 4(s1)
      lw   s3, 12(s1)
      lw   s4, 8(s1)
      addi s0, s0, -1
      j    LOOP
DONE:
```



Direct-Mapped Cache Performance - Compulsory Misses

RISC-V assembly code

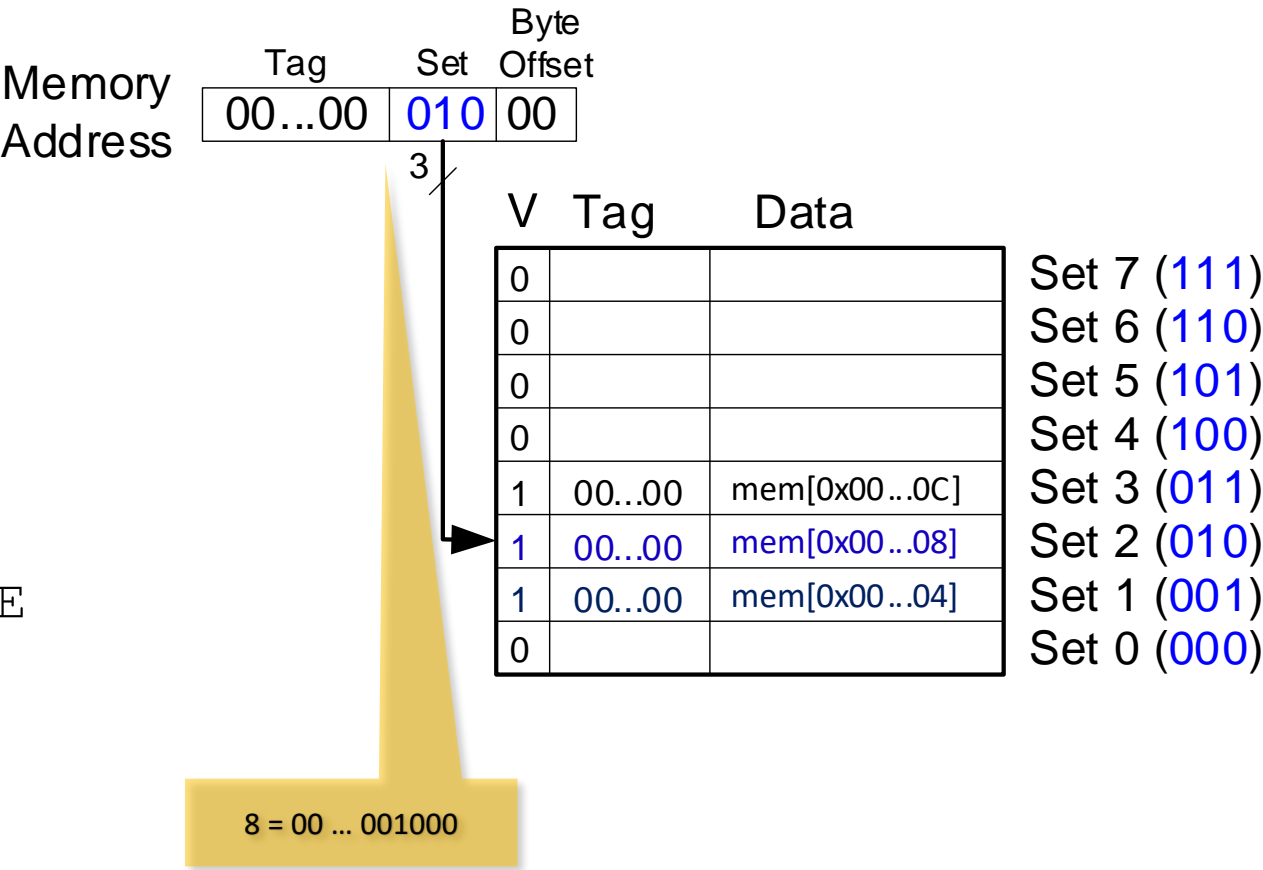
```
    addi s0, zero, 5
    addi s1, zero, 0
LOOP: beq  s0, zero, DONE
      lw   s2, 4(s1)
      lw   s3, 12(s1)
      lw   s4, 8(s1)
      addi s0, s0, -1
      j    LOOP
DONE:
```



Direct-Mapped Cache Performance - Compulsory Misses

RISC-V assembly code

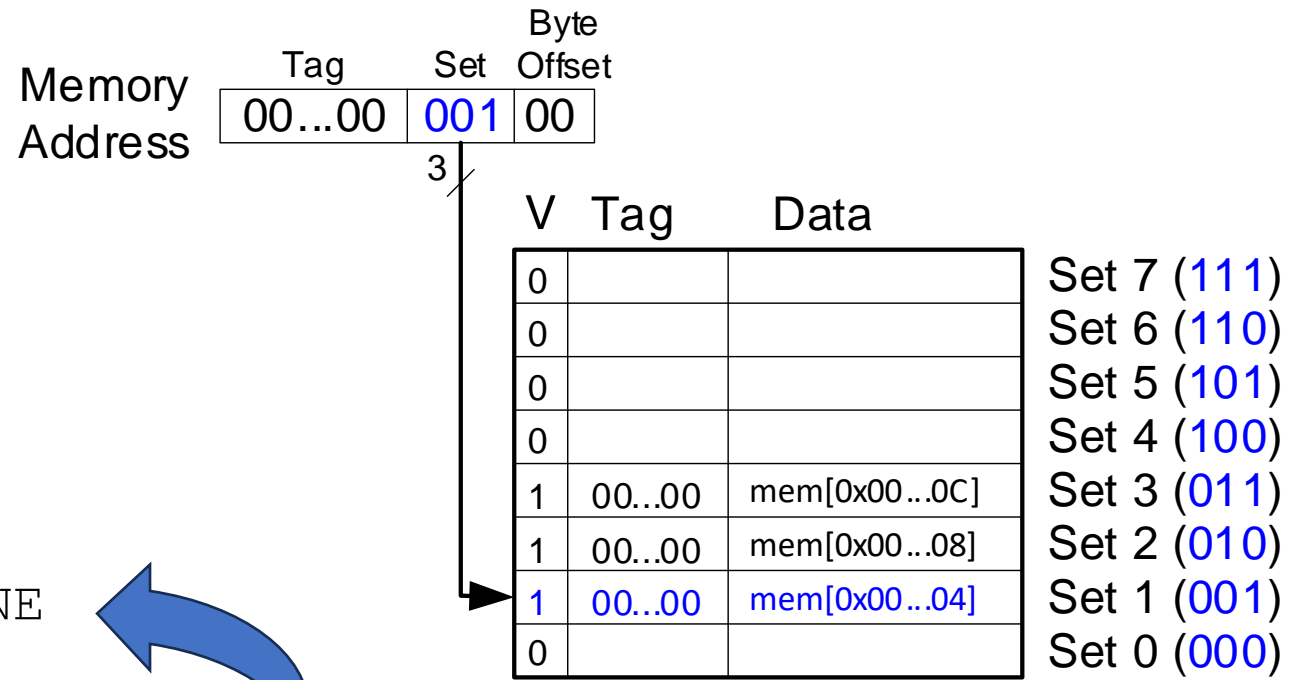
```
    addi s0, zero, 5
    addi s1, zero, 0
LOOP: beq  s0, zero, DONE
      lw   s2, 4(s1)
      lw   s3, 12(s1)
      lw   s4, 8(s1)
      addi s0, s0, -1
      j    LOOP
DONE:
```



Direct-Mapped Cache Performance - Compulsory Misses

RISC-V assembly code

```
    addi s0, zero, 5
    addi s1, zero, 0
LOOP: beq  s0, zero, DONE
      lw   s2, 4(s1)
      lw   s3, 12(s1)
      lw   s4, 8(s1)
      addi s0, s0, -1
      j    LOOP
DONE:
```



$$\text{Miss Rate} = \frac{3}{15} = 20\%$$

Temporal Locality
Compulsory Misses

Direct-Mapped Cache Performance - Conflict Miss

RISC-V assembly code

```
    addi s0, zero, 5
    addi s1, zero, 0
LOOP: beq  s0, zero, DONE
      lw   s2, 0x4(s1)
      lw   s4, 0x24(s1)
      addi s0, s0, -1
      j    LOOP
DONE:
```

0x4 (4) = 00 ... 000100

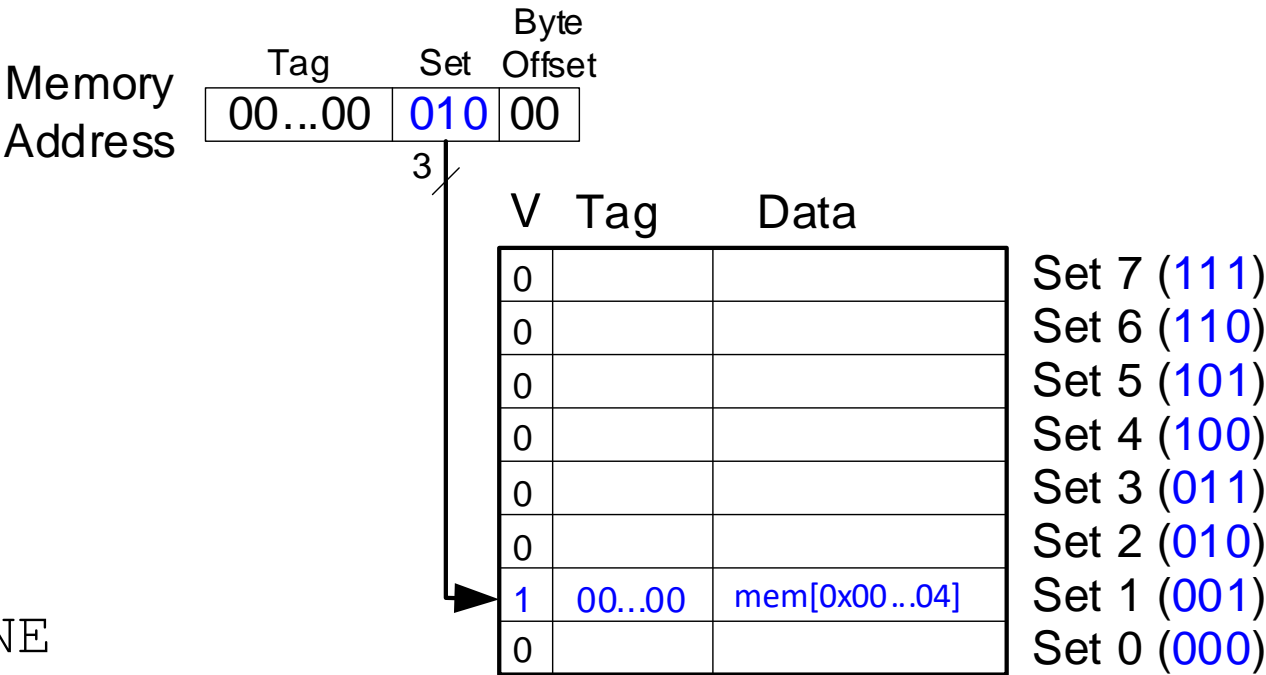
0x24 (36) = 00 ... 100100

Same Set !!!!

Direct-Mapped Cache Performance - Conflict Miss

RISC-V assembly code

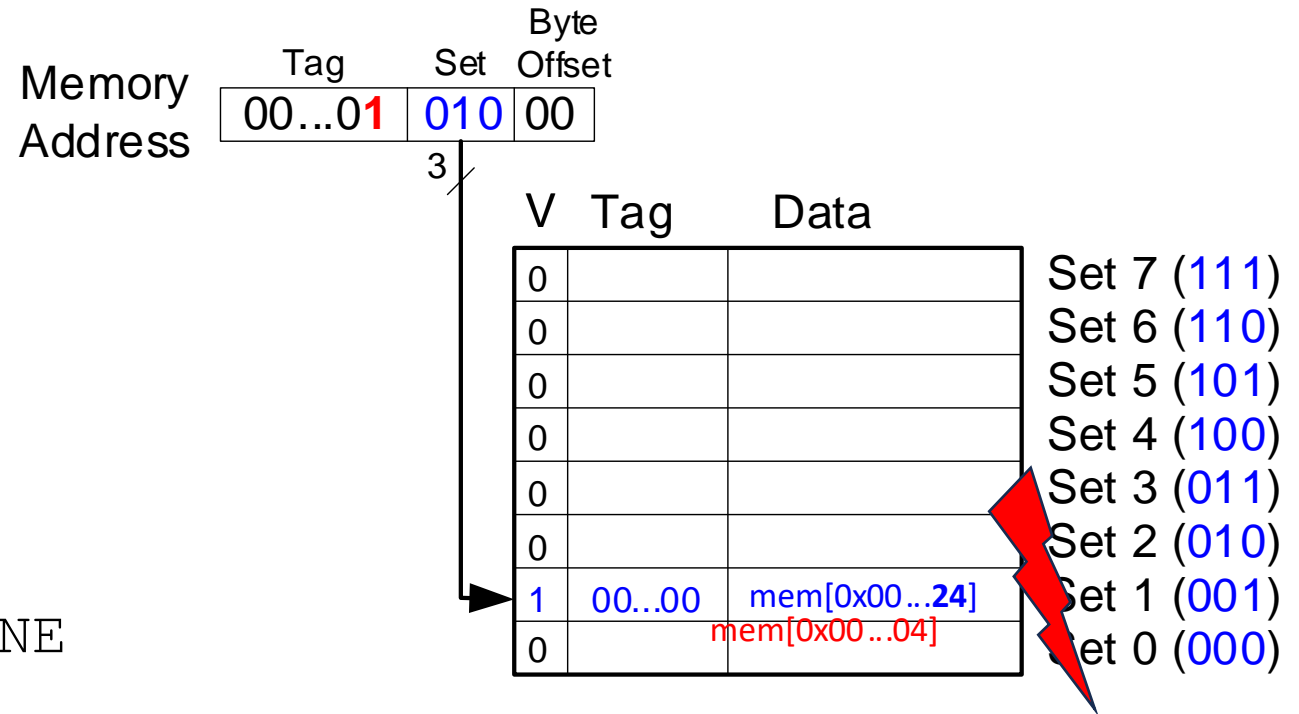
```
    addi s0, zero, 5
    addi s1, zero, 0
LOOP: beq  s0, zero, DONE
      lw    s2, 0x4(s1)
      lw    s4, 0x24(s1)
      addi s0, s0, -1
      j     LOOP
DONE:
```



Direct-Mapped Cache Performance - Conflict Miss

RISC-V assembly code

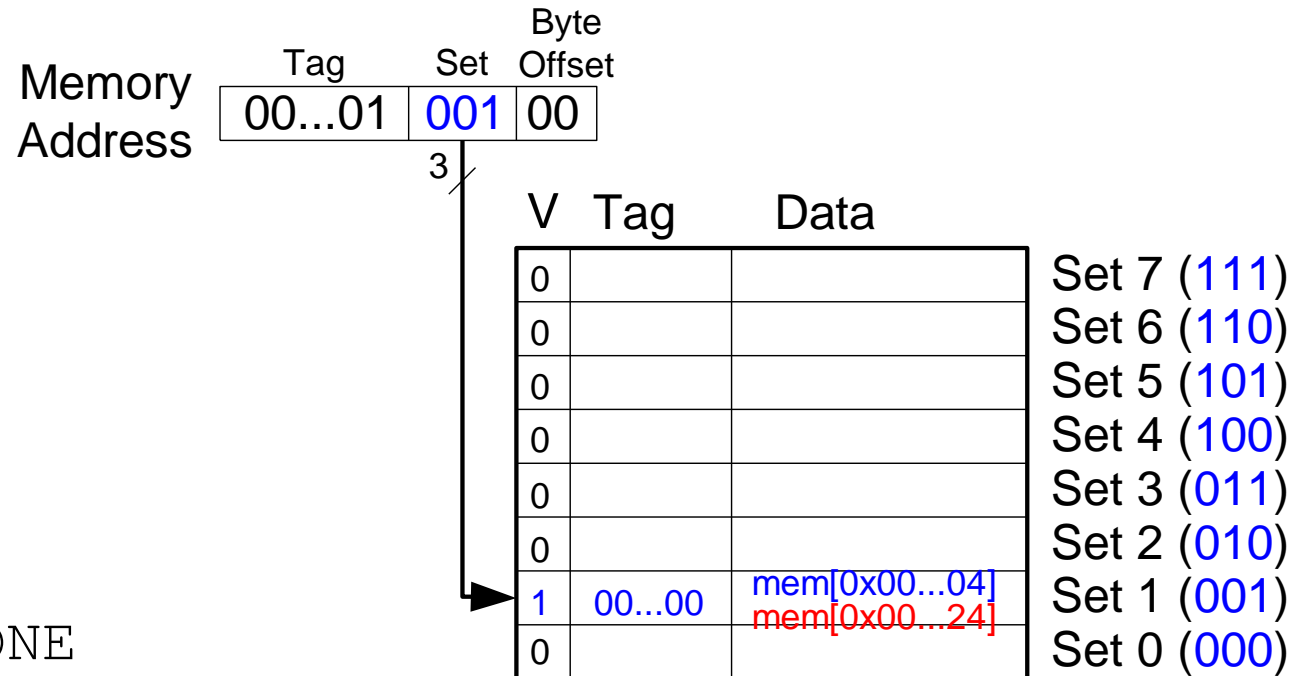
```
    addi s0, zero, 5
    addi s1, zero, 0
LOOP: beq  s0, zero, DONE
      lw   s2, 0x4(s1)
      lw   s4, 0x24(s1)
      addi s0, s0, -1
      j    LOOP
DONE:
```



Direct-Mapped Cache Performance - Conflict Miss

RISC-V assembly code

```
    addi s0, zero, 5
    addi s1, zero, 0
LOOP: beq  s0, zero, DONE
      lw   s2, 0x4(s1)
      lw   s4, 0x24(s1)
      addi s0, s0, -1
      j    LOOP
DONE:
```

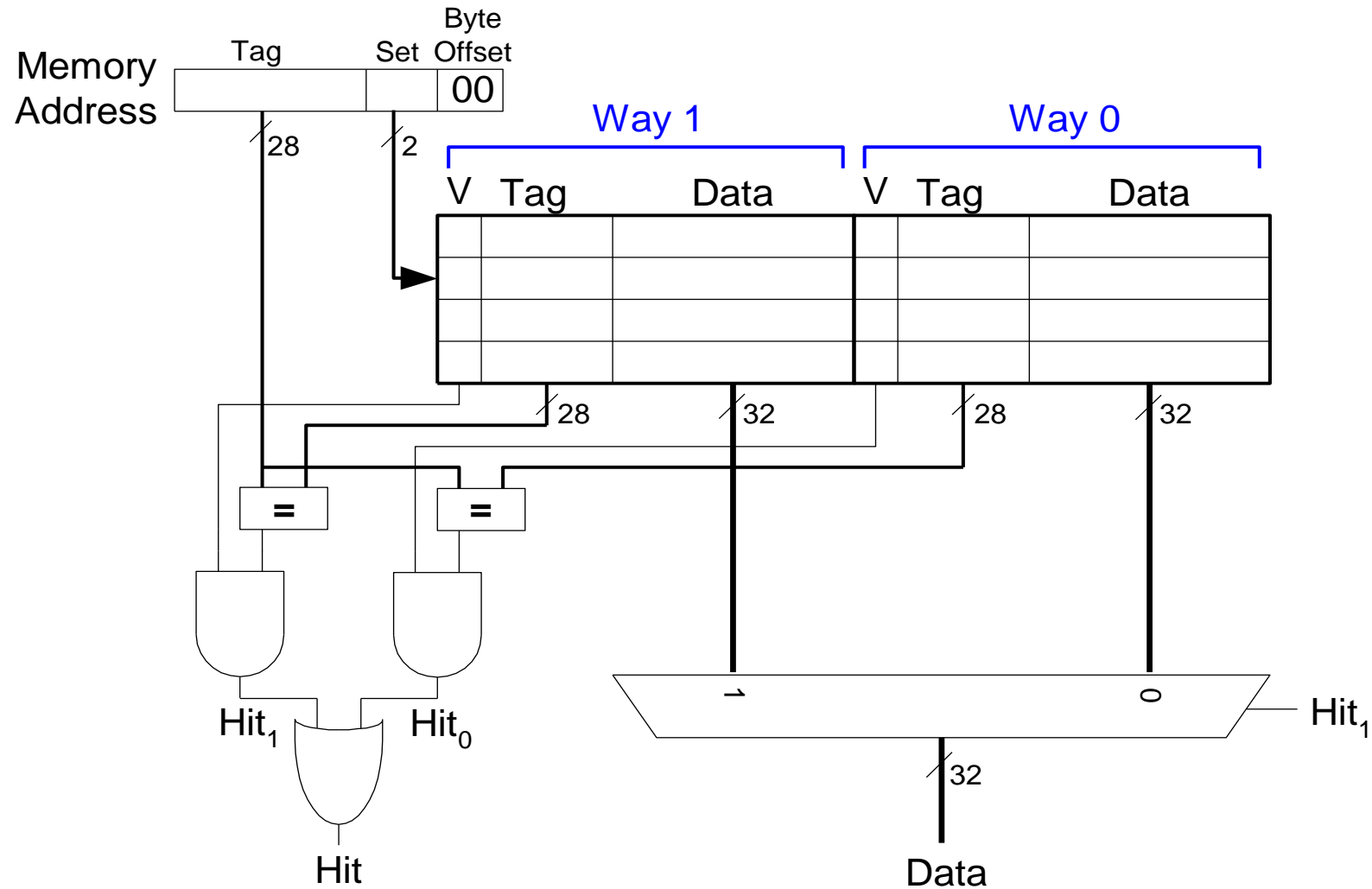


$$\text{Miss Rate} = \frac{10}{10} = 100\%$$

Conflict Misses

Associative Caches

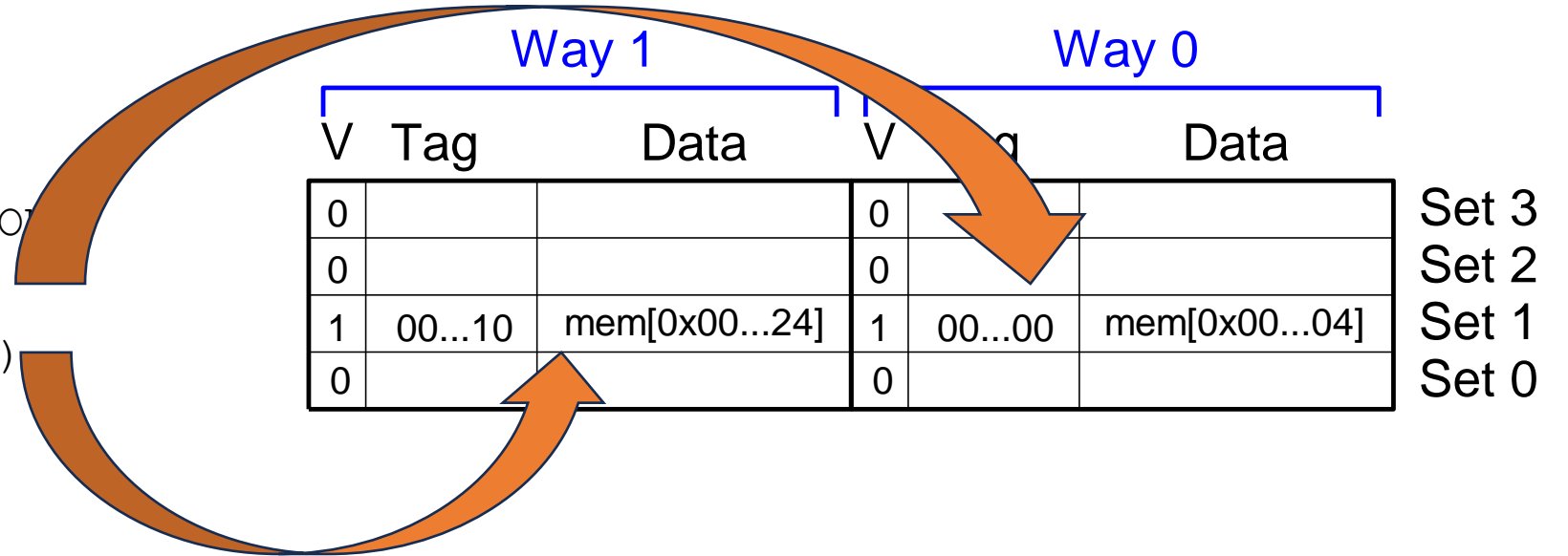
N-Way Set Associative Cache



N-Way Set Associative Cache Performance

RISC-V assembly code

```
    addi s0, zero, 5
    addi s1, zero, 0
LOOP: beq  s0, zero, DONE
      lw   s2, 0x4(s1)
      lw   s4, 0x24(s1)
      addi s0, s0, -1
      j    LOOP
DONE:
```

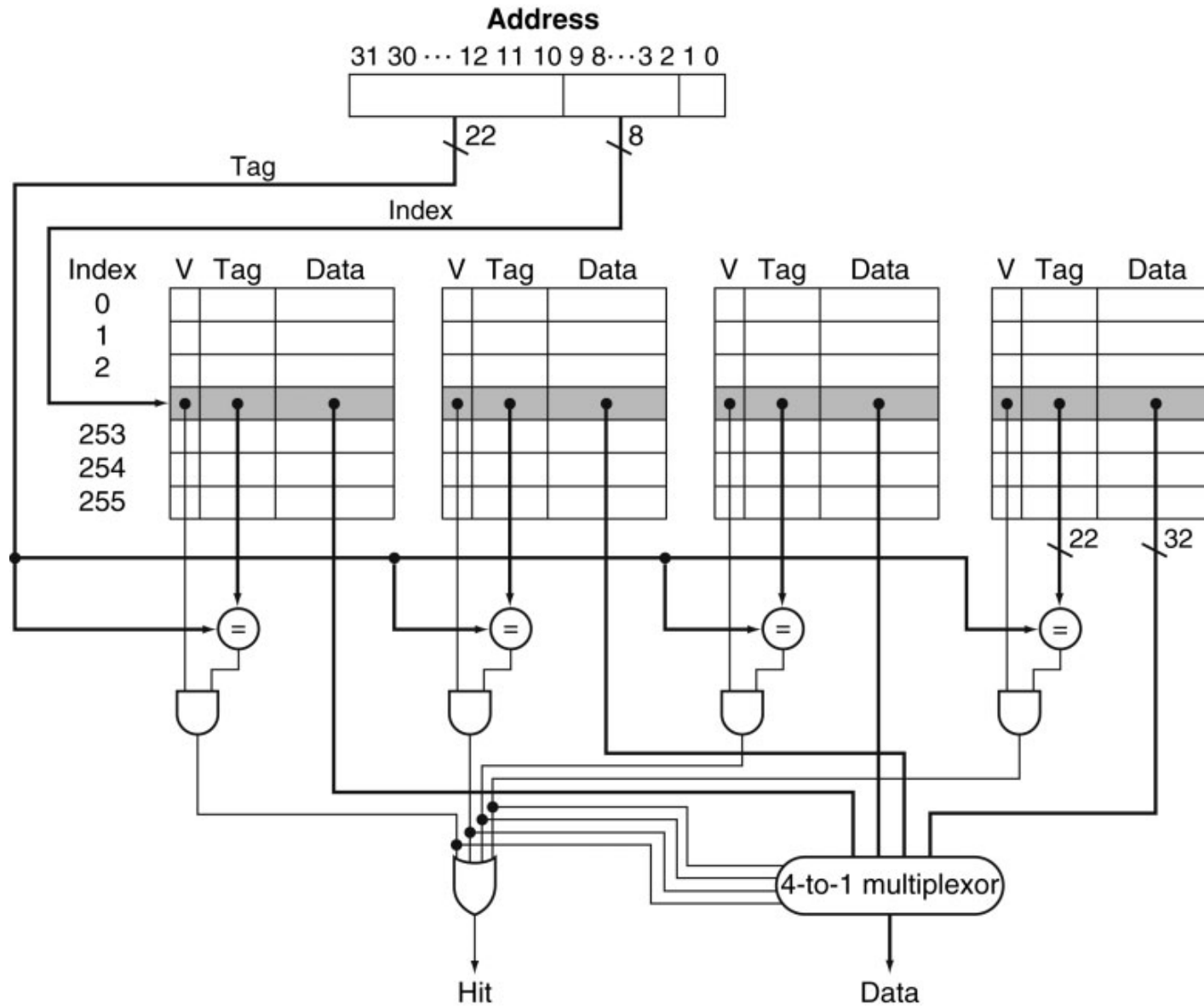


$$\text{Miss Rate} = \frac{2}{10} = 20\%$$

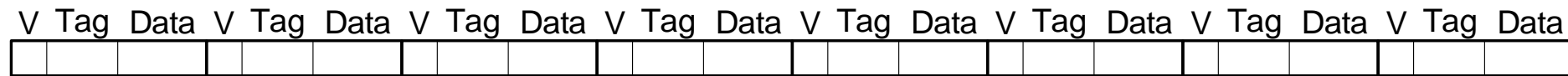
Associativity reduces Conflict Misses

Lower-numbered ways (e.g., Way 0, Way 1) are prioritized before higher-numbered ones.

Set Associative Cache: Aufbau



Fully Associative Cache

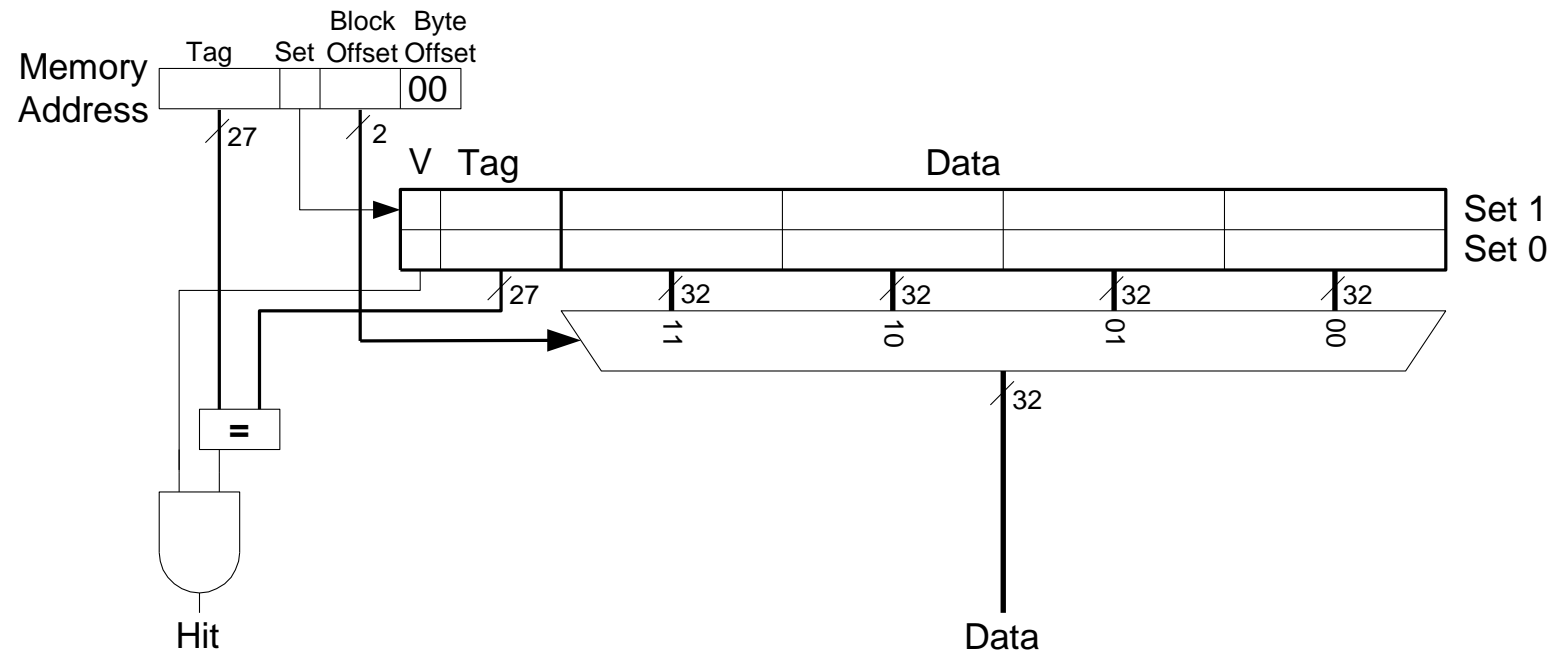


Reduces Conflict Misses but is expensive to build

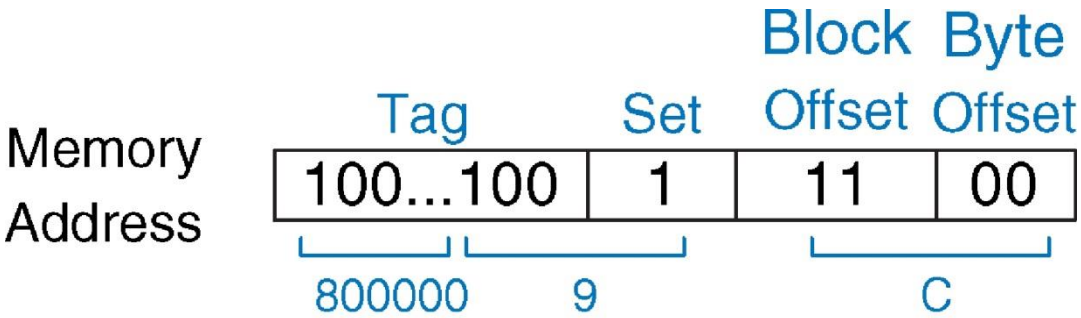
Spatial Locality

Spatial Locality

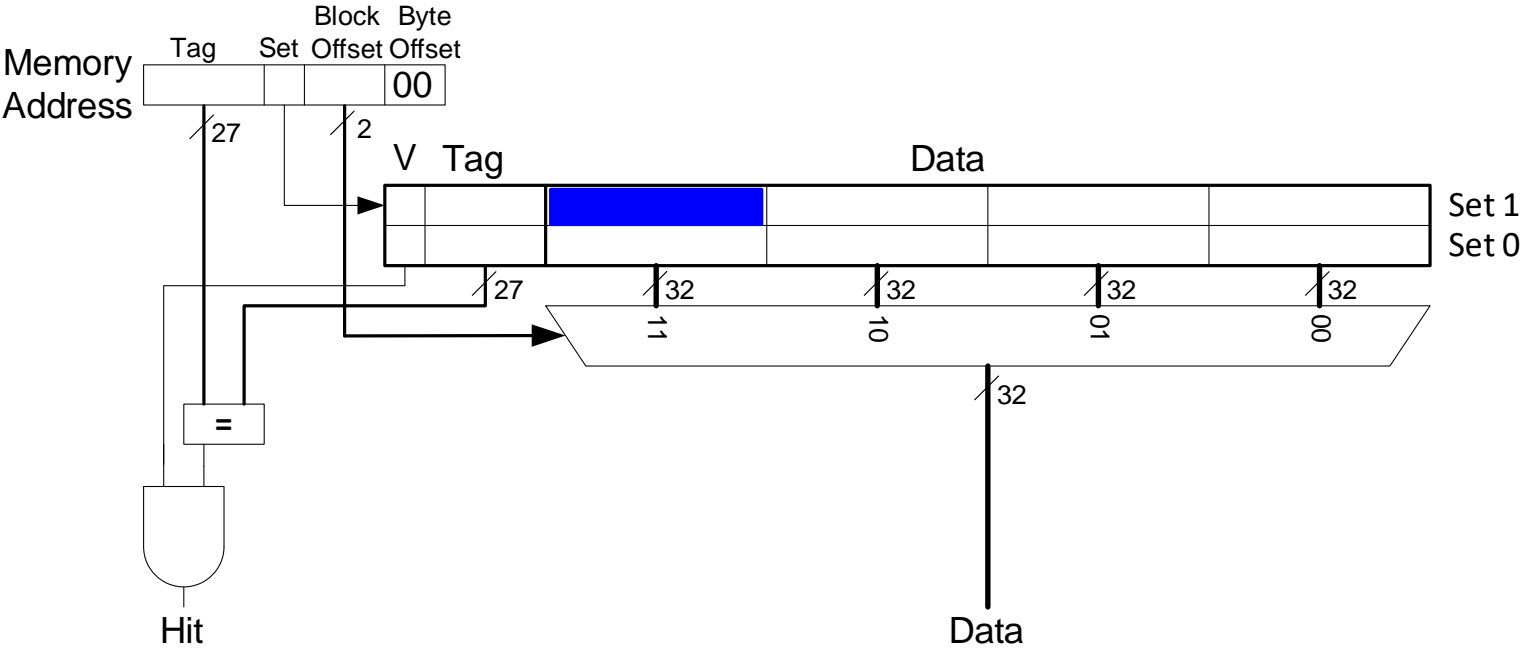
- Increase block size:
 - Block size, $b = 4$ words
 - $C = 8$ words
 - Direct mapped (1 block per set)
 - Number of blocks, $B = 2 \left(\frac{C}{b} = \frac{8}{4} = 2 \right)$



Cache with Larger Block Size



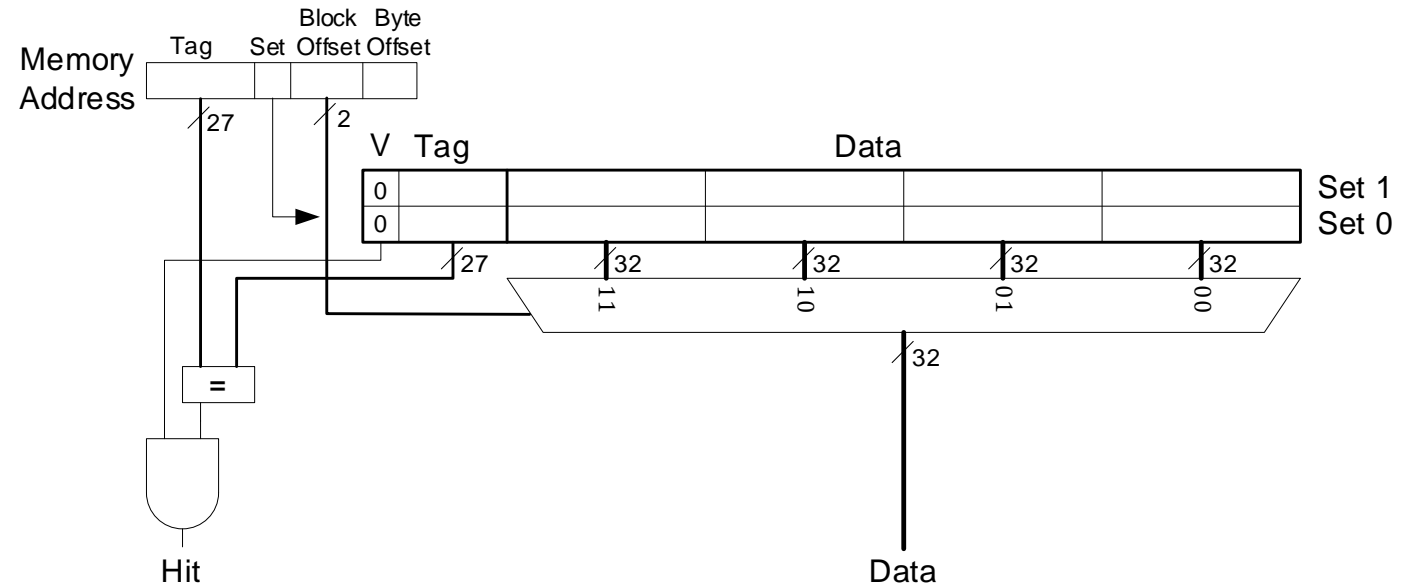
© 2007 Elsevier, Inc. All rights reserved



Cache Performance with Spatial Locality

RISC-V assembly code

```
    addi s0, zero, 5
    addi s1, zero, 0
LOOP: beq  s0, zero, DONE
      lw   s2, 4(s1)
      lw   s3, 12(s1)
      lw   s4, 8(s1)
      addi s0, s0, -1
      j    LOOP
DONE:
```

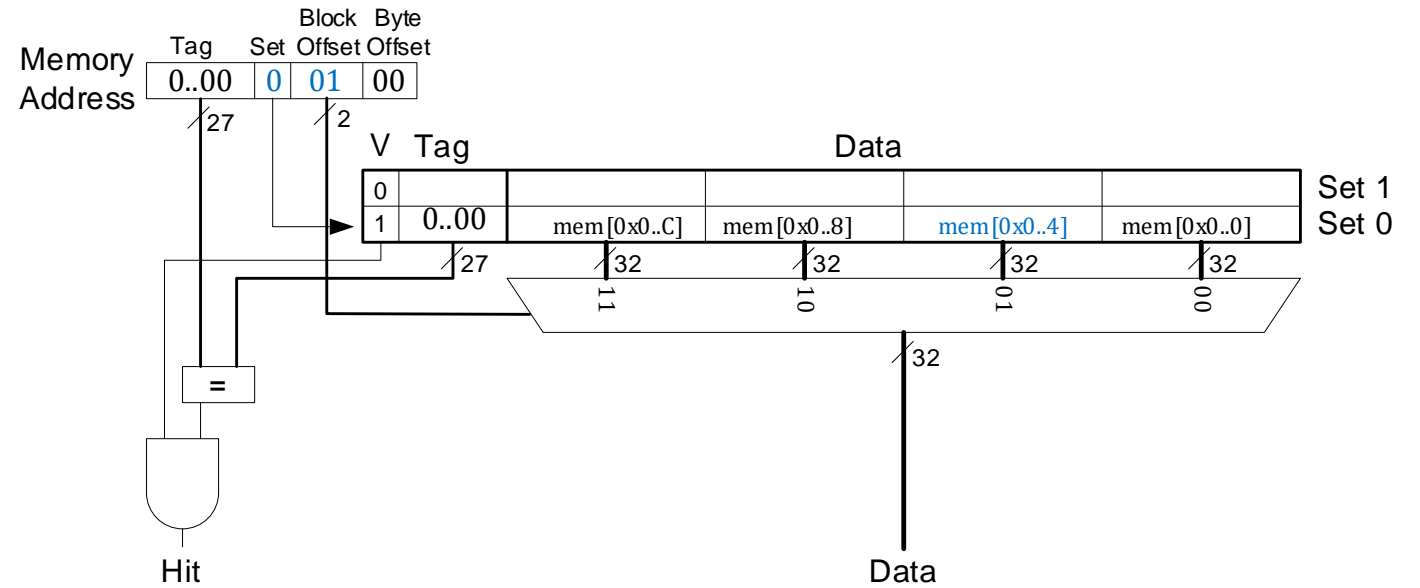


Cache is empty

Cache Performance with Spatial Locality

RISC-V assembly code

```
    addi s0, zero, 5
    addi s1, zero, 0
LOOP: beq  s0, zero, DONE
      lw   s2, 4(s1)
      lw   s3, 12(s1)
      lw   s4, 8(s1)
      addi s0, s0, -1
      j    LOOP
DONE:
```

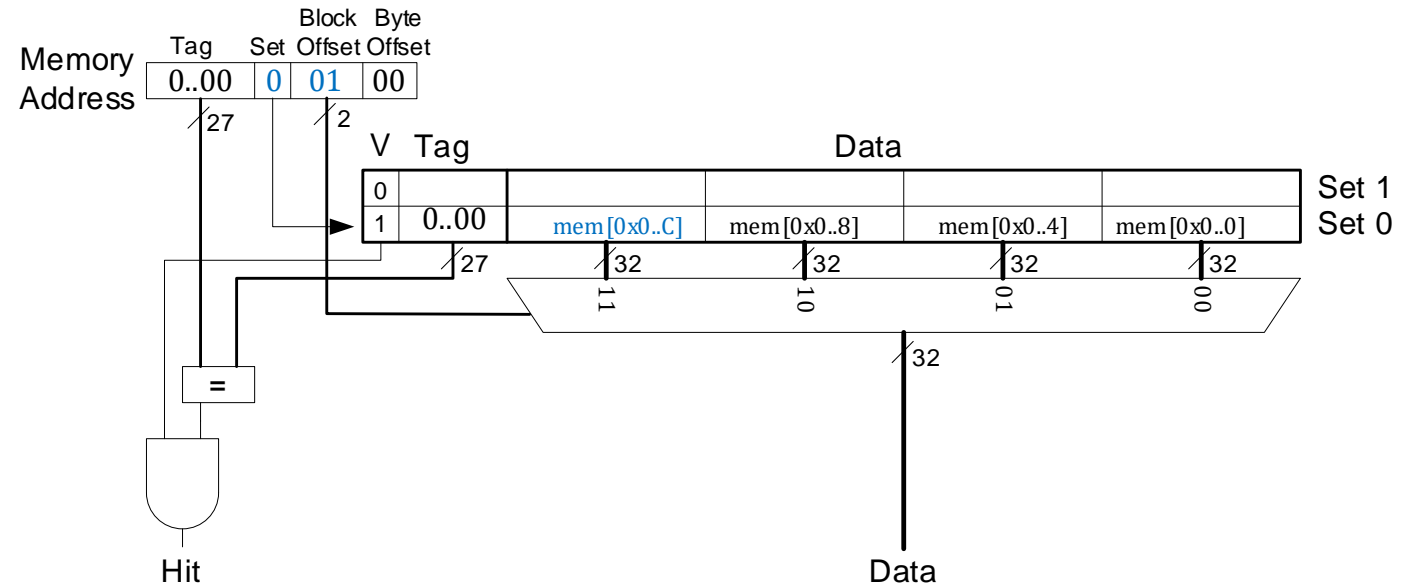


Whole block is loaded under once

Cache Performance with Spatial Locality

RISC-V assembly code

```
    addi s0, zero, 5
    addi s1, zero, 0
LOOP: beq  s0, zero, DONE
      lw   s2, 4(s1)
      lw   s3, 12(s1)
      lw   s4, 8(s1)
      addi s0, s0, -1
      j    LOOP
DONE:
```

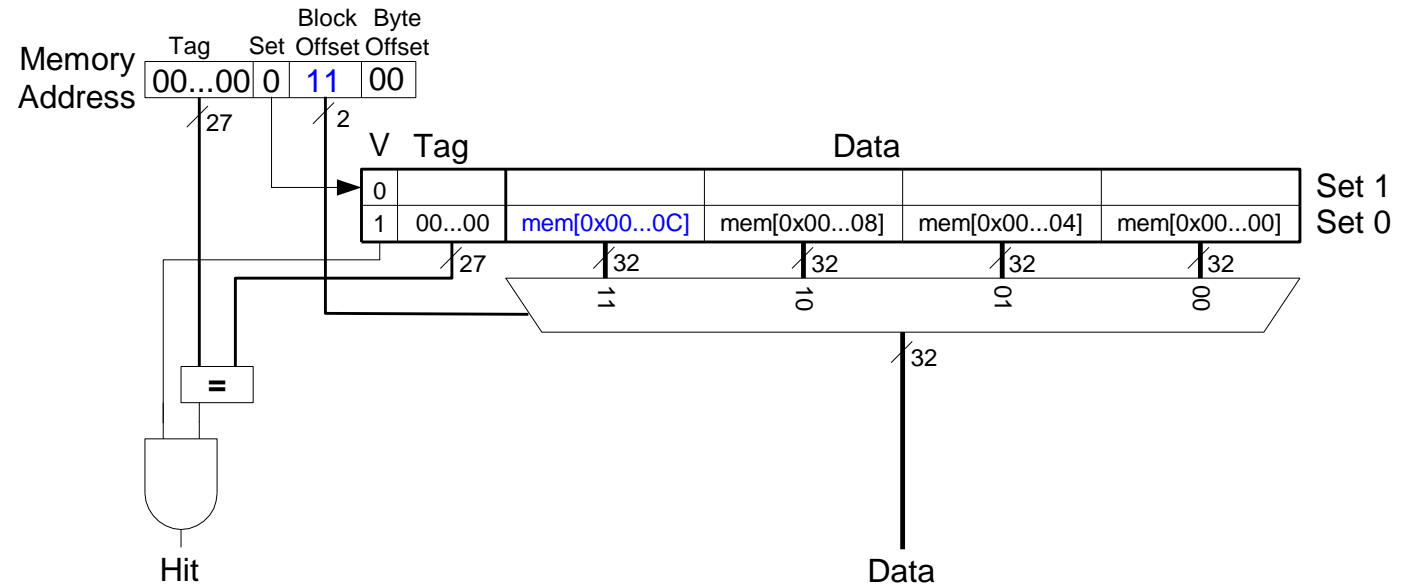


Data is already in cache

Cache Performance with Spatial Locality

RISC-V assembly code

```
    addi s0, zero, 5
    addi s1, zero, 0
LOOP: beq  s0, zero, DONE
      lw   s2, 4(s1)
      lw   s3, 12(s1)
      lw   s4, 8(s1)
      addi s0, s0, -1
      j    LOOP
DONE:
```



$$\text{Miss Rate} = \frac{1}{15} = 6.67\%$$

Larger blocks reduce compulsory misses through spatial locality

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

Cache Organization Recap

- Capacity: C
- Block size: b
- Number of blocks in cache: $B = \frac{C}{b}$
- Number of blocks in a set: N
- Number of sets: $S = \frac{B}{N}$

| Organization | Number of Ways (N) | Number of Sets (S) |
|-----------------------|------------------------|------------------------|
| Direct Mapped | 1 | B |
| N-Way Set Associative | $1 < N < B$ | $\frac{B}{N}$ |
| Fully Associative | B | 1 |

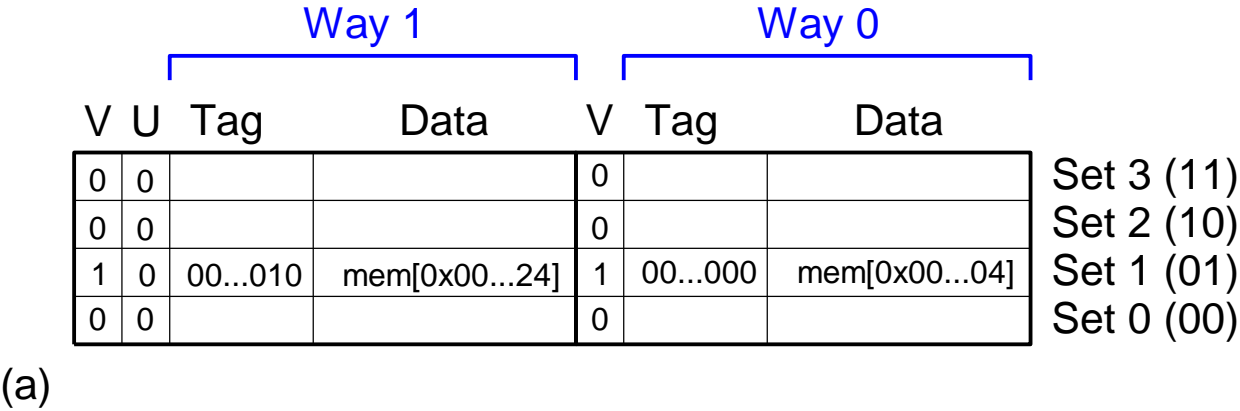
Cache Replacement Policy

Replacement Policy

- Cache is too small to hold all data of interest at once
- If cache full: program accesses data X and 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
 - „Am längsten nicht verwendet“
 - Complex to implement
 - **Least Frequently Used (LFU)**: the least used used block in a set evicted
 - “Am wenigsten verwendet”
 - A frequently referenced page is only replaced after many misses and thus blocks the cache
 - **Reference Bit**: it is a approximated versions of Least Recently Used (LRU)
 - A bit is set on access and cleared in time intervals
 - **Random**
 - **FIFO (first-in first-out)**

RISC-V assembly

```
lw s1, 0x04(zero)
lw s1, 0x04(zero)
lw s2, 0x24(zero)
lw s3, 0x54(zero)
```



Least recently used - LRU Replacement

RISC-V assembly

```
lw s1, 0x04(zero)
lw s1, 0x04(zero)
lw s2, 0x24(zero)
lw s3, 0x54(zero)
```

Way 1Way 0

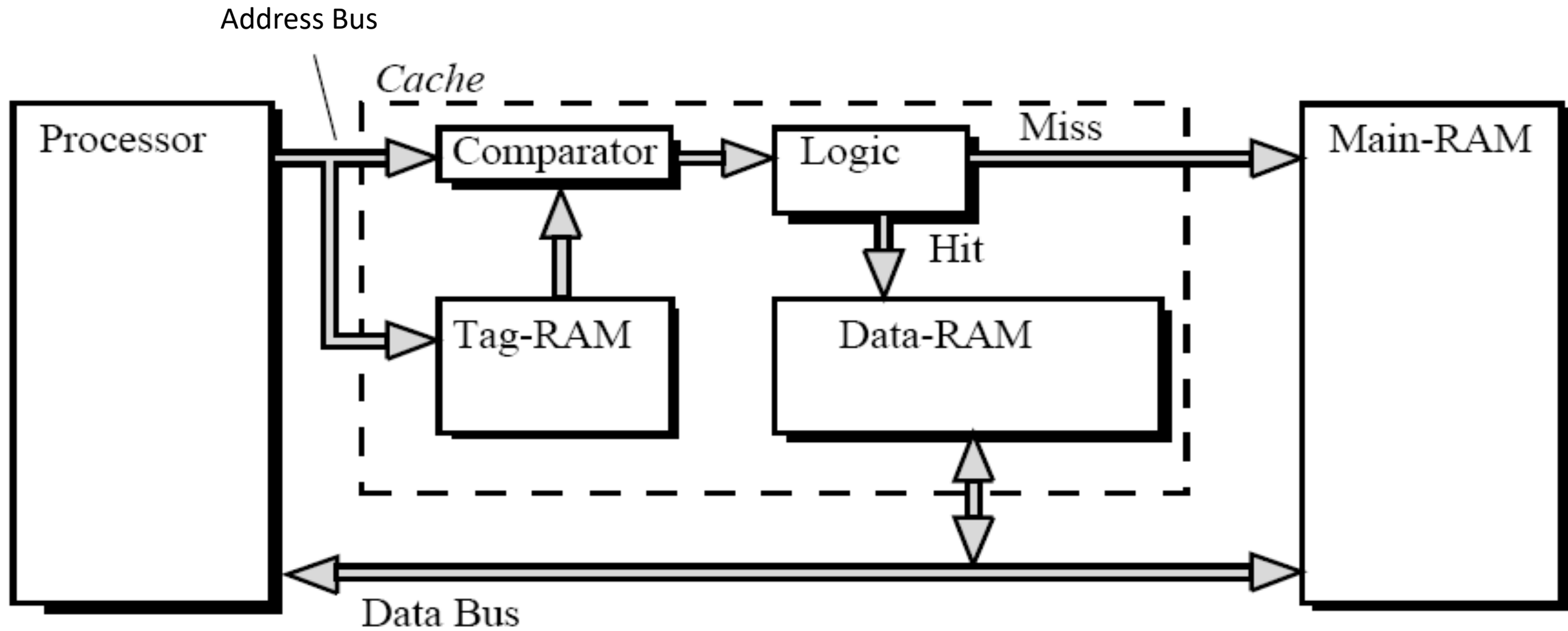
| V | U | Tag | Data | V | Tag | Data | |
|---|---|----------|----------------|---|----------|----------------|------------|
| 0 | 0 | | | 0 | | | Set 3 (11) |
| 0 | 0 | | | 0 | | | Set 2 (10) |
| 1 | 0 | 00...010 | mem[0x00...24] | 1 | 00...000 | mem[0x00...04] | Set 1 (01) |
| 0 | 0 | | | 0 | | | Set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

(a)

Way 1Way 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)



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

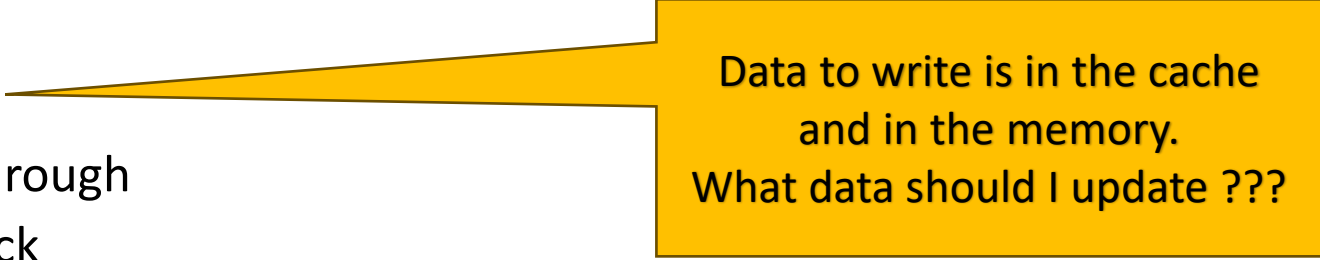
Load

- Load Hit: Valid bit is set and tag matches. Data is found in cache
- Load Miss: Data is not found in cache
 - Pipeline needs to be stalled
 - Slower memory must be asked to deliver the data

Store

- Write Hit

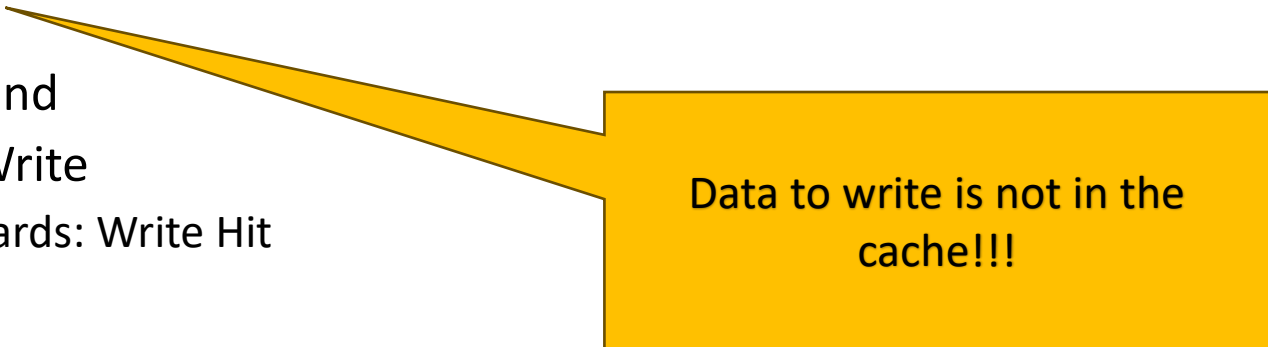
- Write-Through
- Copy-Back
- Write-Buffer



Data to write is in the cache
and in the memory.
What data should I update ???

- Write Miss

- Write-Around
- Fetch-on-Write
 - Afterwards: Write Hit



Data to write is not in the
cache!!!

Write-Hit: Write-Through

- update the cache **AND**
- update the main memory immediately

- Pros
 - Data consistency with main memory guaranteed (I/O, multiprocessor)
 - simple
- Cons
 - Frequent accesses to the main memory
 - Loss of performance

Write-Hit: Copy-Back

- refresh the cache **AND** marks the block "**dirty**"
- only update the main memory later when the block is removed from the cache
 - often also referred to as *write-back*
- Pros
 - Write hit is much faster
 - Less frequent accesses to the main memory
- Cons
 - Data inconsistency with the main memory
 - Read miss is slower (due to copy-back)
 - A dirty block needs to be synced before replacing

- for data consistency and fast write operations
 - advantages of Write-Through and Copy-Back
- Write-Buffer (Buffered Write-Through)
 - new value is entered in the cache and second fast cache
 - Processor can continue with further processing
 - if buffer is full, processor must wait

Write-Miss: Write-Around

- Ignore the cache AND write directly to memory
- Mostly in combination with Write-Through

- Replace the current content of the cache and update Tag
- If block size > 1 word, load the remaining data belonging to the block from the main memory after
 - Read access to the memory and
 - then write hit depending on the strategy
- This is the most frequently used method

Management data/bits on a N-Way set associative

- Management data/bits
 - Tag
 - Valid Bit
 - (Dirty Bit)
 - Bits for Replacement Policy
- User data
- Number of blocks = Number of Sets * Ways

| N-way set associative | | | | | | | | | | | | | | |
|-----------------------|-------|------|-------|------|-----|--|-----|--|-----|--|-----|--|---------|------|
| | Way 0 | | Way 1 | | ... | | ... | | ... | | ... | | Way N-1 | |
| Set | Verw. | Data | Verw. | Data | | | | | | | | | Verw. | Data |
| 0 | | | | | | | | | | | | | | |
| 1 | | | | | | | | | | | | | | |
| ... | | | | | | | | | | | | | | |
| m | | | | | | | | | | | | | | |

Management data/bits on a One-Way set associative (direct mapped)

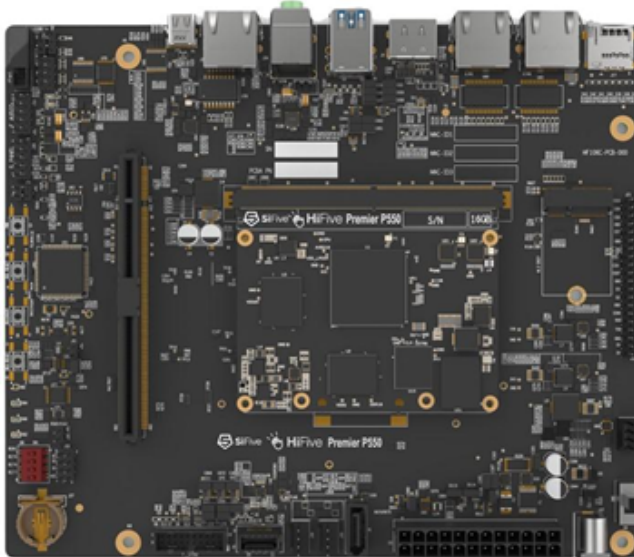
- Management data/bits
 - Tag
 - Valid Bit
 - (Dirty Bit)
- User data
- Number of blocks = Number of Sets

| One-way set associative (direct mapped) | | |
|---|-------|------|
| | Way 0 | |
| Set | Verw. | Data |
| 0 | | |
| 1 | | |
| 2 | | |
| 3 | | |
| 4 | | |
| 5 | | |
| ... | | |
| m | | |

(Aktuelle) Prozessoren

| | ARM Cortex-A8 (Raspberry v1) | Intel i7 Nehalem |
|------------------------|---------------------------------------|---|
| L1 Organisation | Geteilt Instruktionen / Daten | Geteilt Instruktionen / Daten |
| L1 Size | 32KiB per Instruktion/Data | 32KiB per Instruktion/Data per Core |
| L1 Associative | 4-way (I), 4-way (D) Set Associative | 4-way (I), 8-way (D) Set Associative |
| L1 replacement | Random | Approximated LRU |
| L1 block size | 64 byte | 64 byte |
| L1 write policy | Write-back, Write-allocate (?) | Write-back, No-write-allocate |
| L1 hit time (load-use) | 1 clock cycle | 4 clock cycles, pipelined |
| | | |
| L2 Organisation | Ein Cache für Instruktionen und Daten | Ein Cache für Instruktionen und Daten, per Core |
| L2 Size | 128KiB to 1MiB | 256KiB, per Core |
| L2 Associative | 8-way Set Associative | 8-way Set Associative |
| L2 replacement | Random (?) | Approximated LRU |
| L2 block size | 64 bytes | 64 byte |
| L2 write policy | Write-back, Write-allocate (?) | Write-back, Write-allocate |
| L2 hit time | 11 clock cycles | 10 clock cycles, pipelined |
| | | |
| L3 Organisation | | Ein Cache für Instruktionen und Daten |
| L3 Size | | 8 MiB, shared |
| L3 Associative | | 16-way Set Associative |
| L3 replacement | | Approximated LRU |
| L3 block size | | 64 byte |
| L3 write policy | | Write-back, Write-allocate |
| L3 hit time | | 35 clock cycles, pipelined |

HiFive™ Premier P550



The HF106 is a RISC-V platform board with carrier board + SOM, using ESWIN-developed EIC7700X SOC. The HF106 features 16GB/32GB of 64-bit LPDDR5 memory up to 6400MHz and 128GB eMMC memory. High-speed interconnect with PCIE Gen3, and external connectors with Gigabit Ethernet and USB3.2 Gen1. Supports storage expansion with SATA and Micro SD Card.

Technical Information

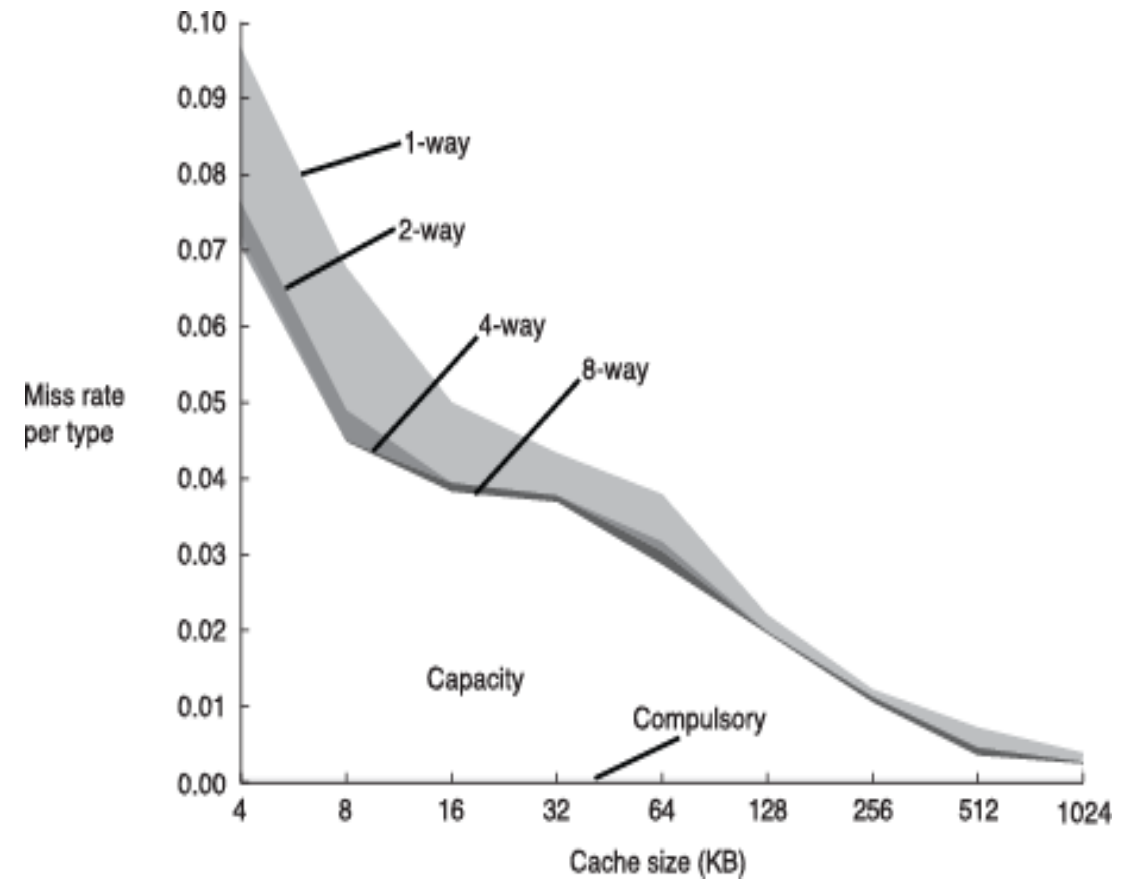
| | |
|----------------------------------|--|
| CPU | ESWIN EIC7700X <ul style="list-style-type: none">Operating Frequency of 1.4 GHzTriple-issue Out-of-order 64-bit execution pipelineQuad-core 64-bit SiFive P550<ul style="list-style-type: none">RV64GCL1-Cache 32 KB(I) + 32KB(D)(private)L2-Cache 256KB(private)L3-Cache 4MB(shared)Cache Supports ECC (Support SECDED) |
| System Memory and Storage | On-board 16GB/32GB 64-bit LPDDR5 @ 6400MT/s On-board 16 MB SPI NOR Flash On-board 128 GB eMMC 5.1 Flash Micro SD Card 2Kbit EEPROM (Manufacturing data) |
| Networking | 1x 100 Mbps Ethernet 2x 1000 Mbps Ethernet |

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

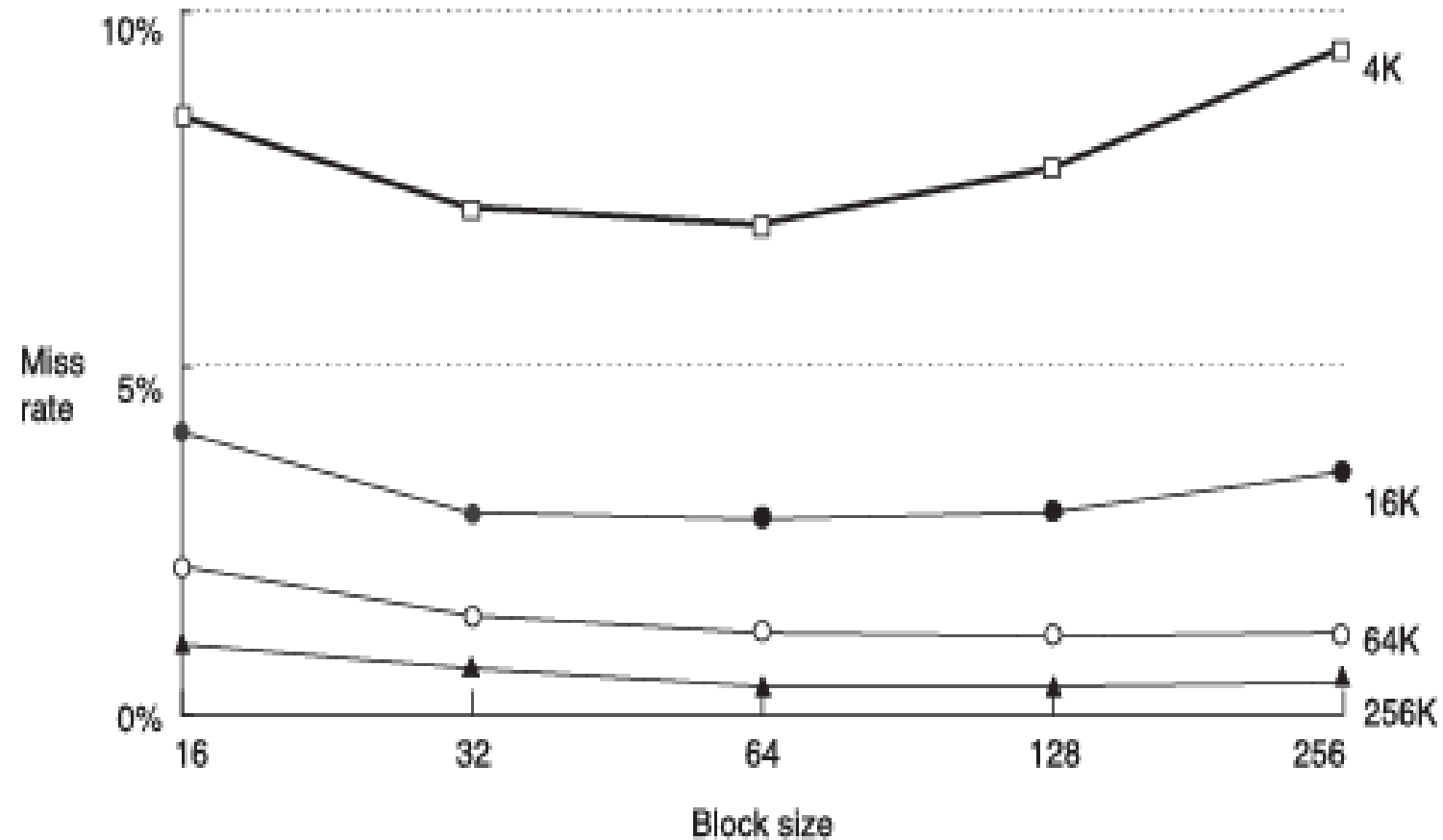
Miss Rate Trends

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



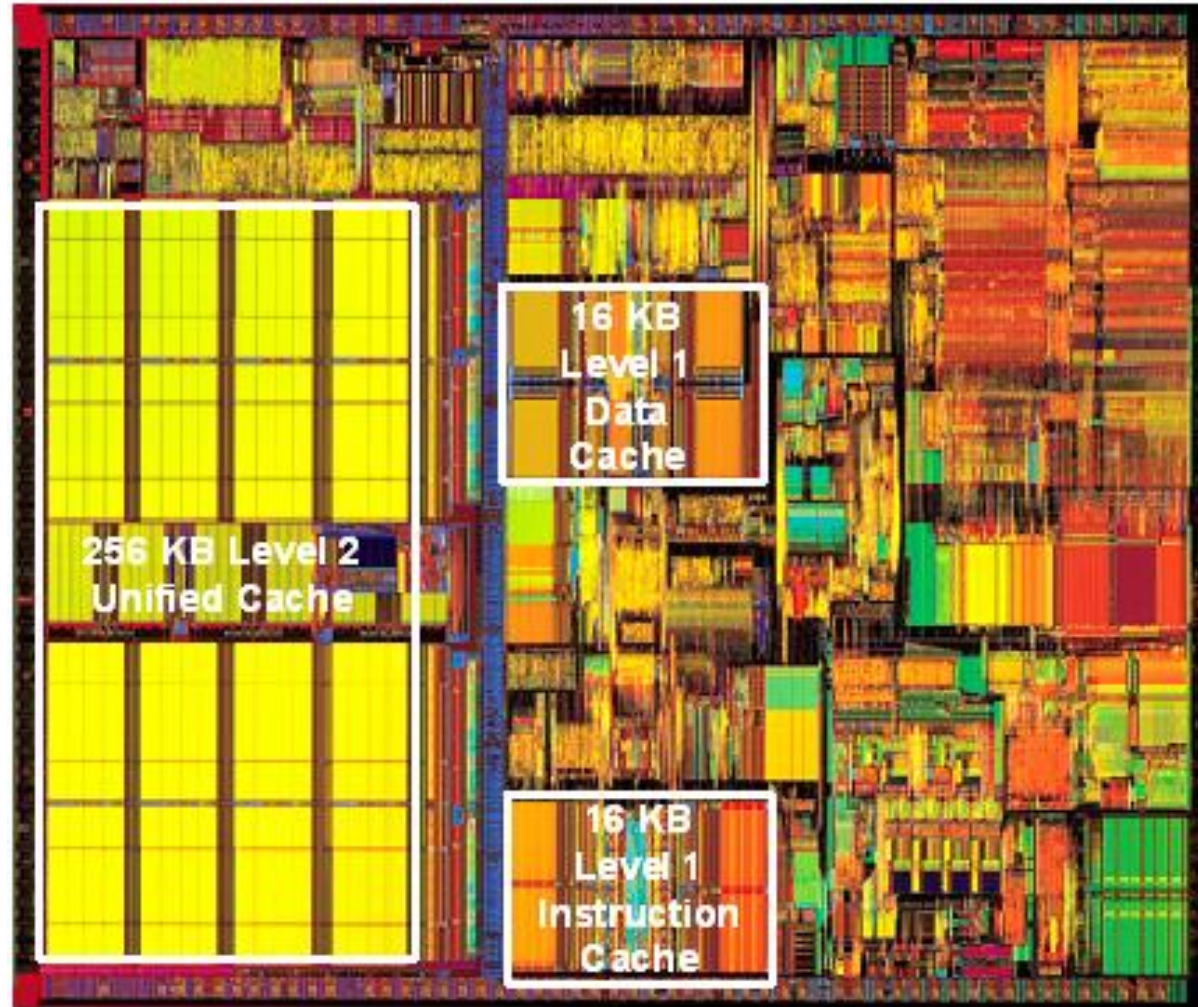
Adapted from Patterson & Hennessy, *Computer Architecture: A Quantitative Approach*, 2011

Miss Rate Trends



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

Intel Pentium III Die

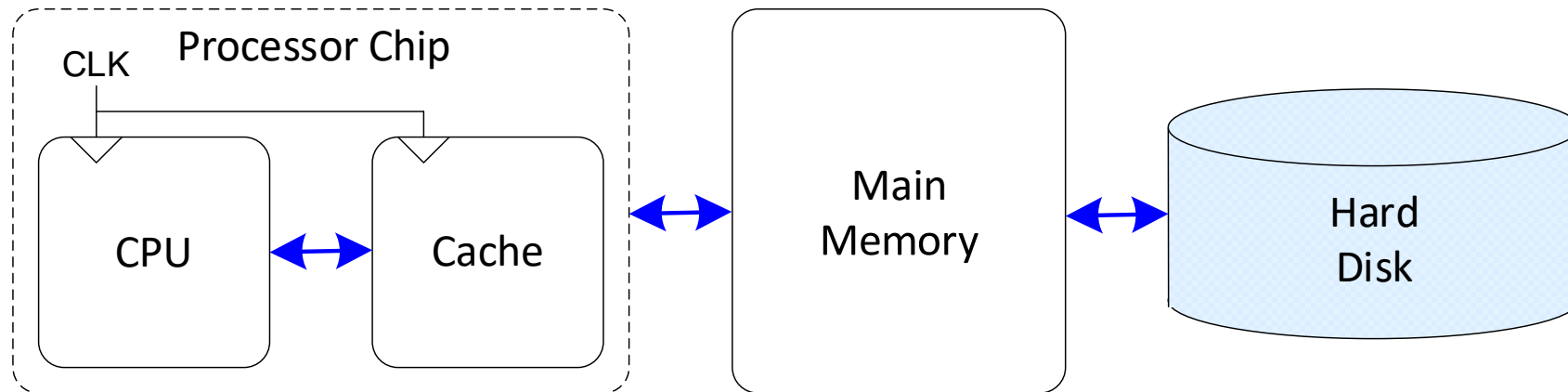


© Intel Corp.

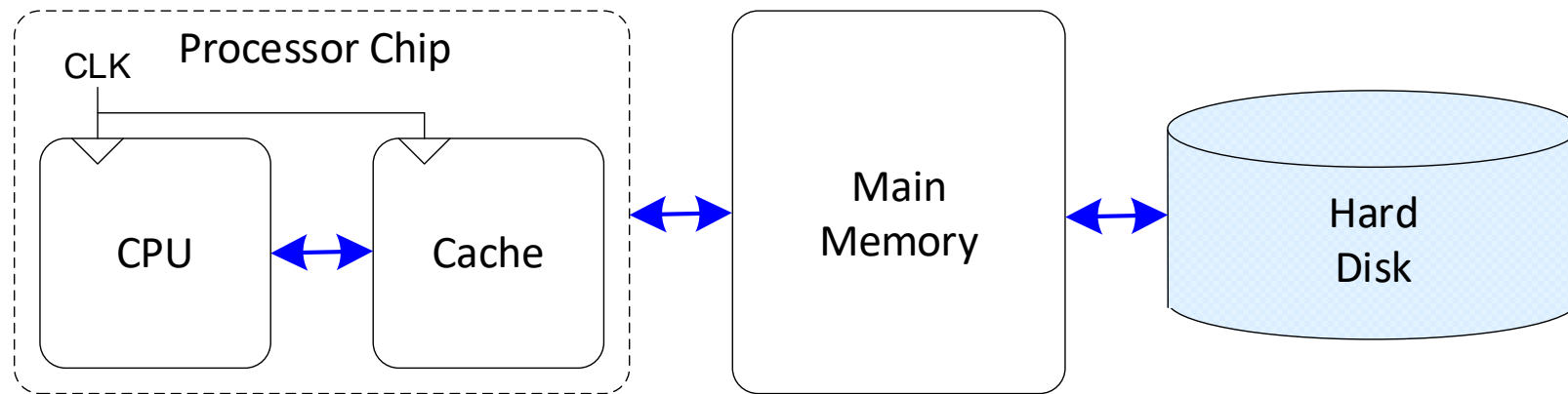
Virtual Memory

Virtual Memory

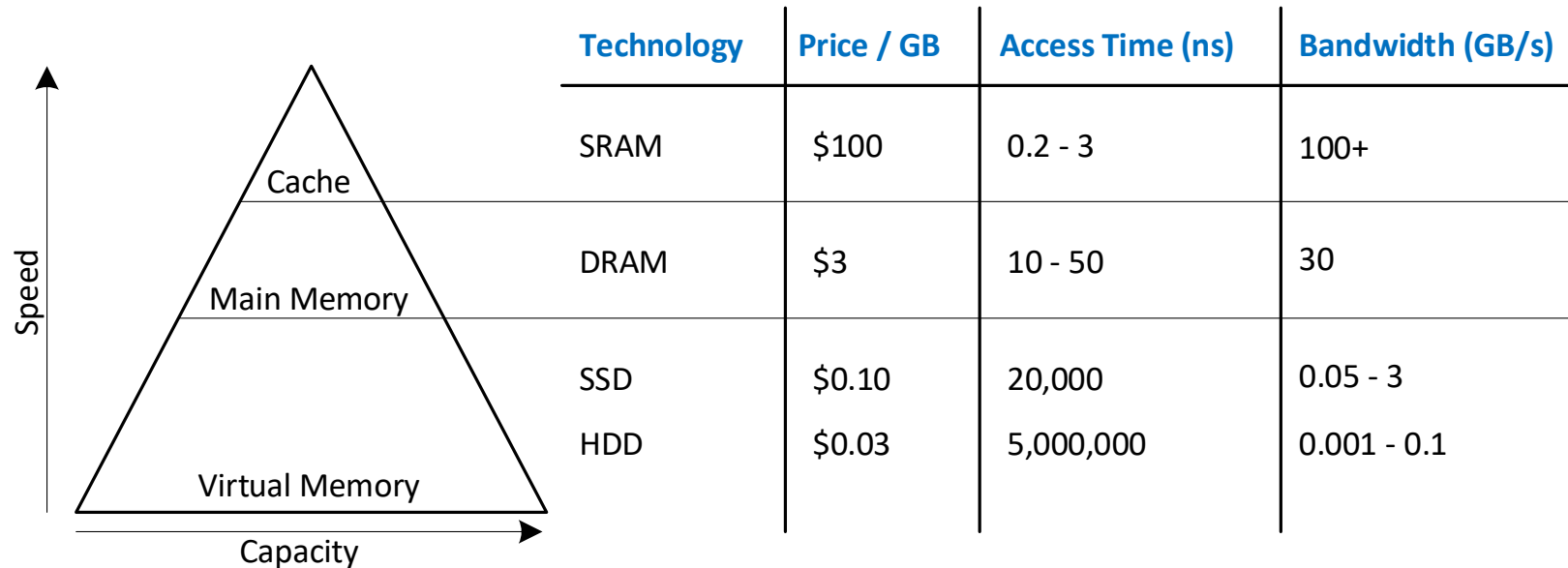
- Gives the illusion of bigger memory
- Main memory (DRAM) acts as cache for hard disk



Memory Hierarchy



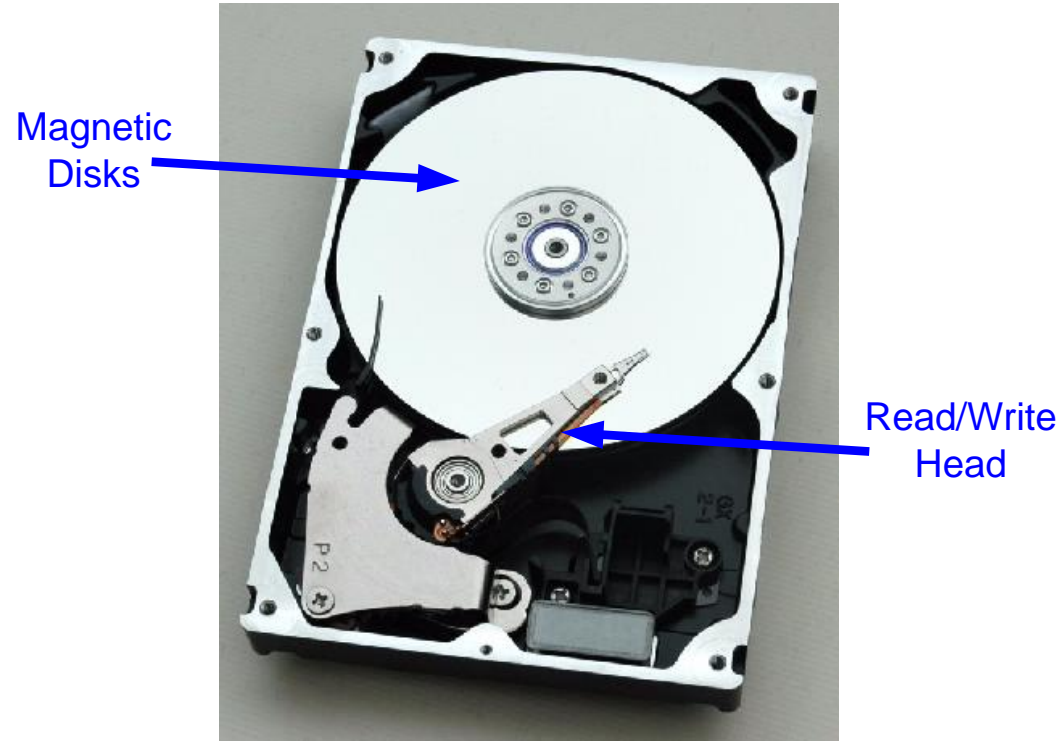
Memory Hierarchy



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

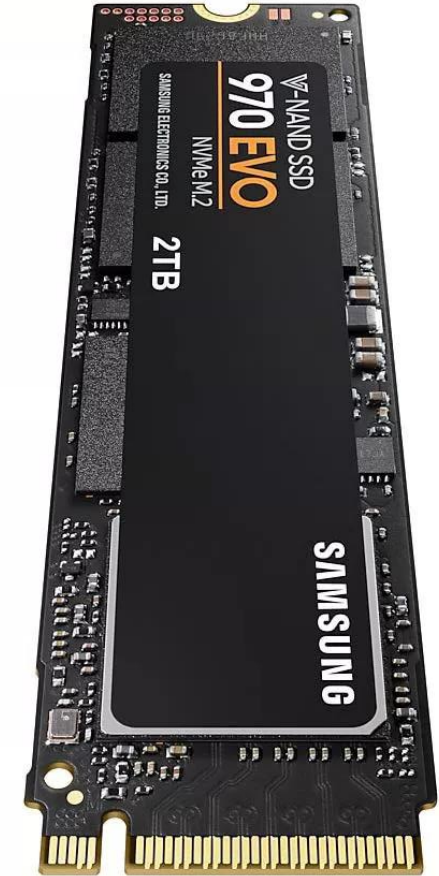
Memory Hierarchy

Hard Disk Drive



Takes milliseconds to seek correct location on disk

Solid State Drive



Arshane88 / CC BY-SA 4.0 / Wikimedia Commons

- **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

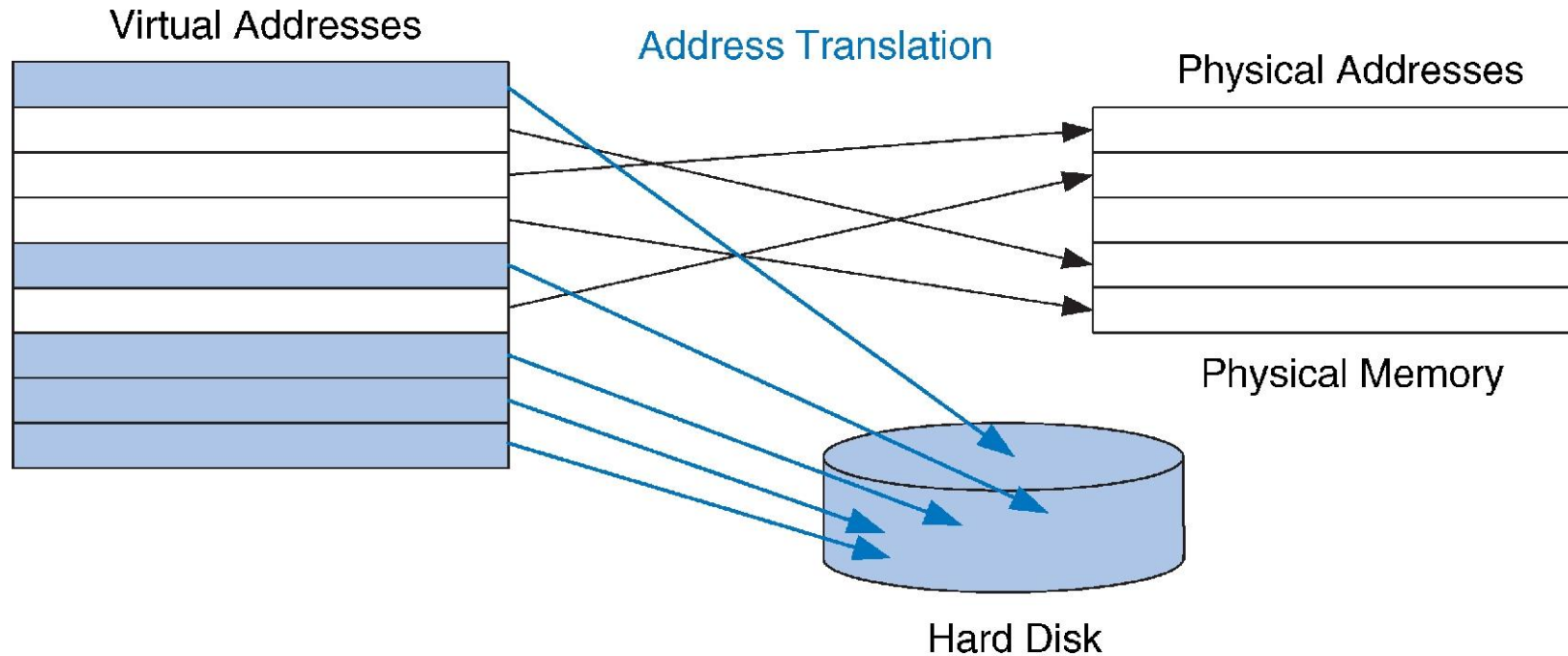
| Cache | Virtual Memory |
|--------------|---------------------|
| Block | Page |
| Block Size | Page Size |
| Block Offset | Page Offset |
| Miss | Page Fault |
| Tag | Virtual Page Number |

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 Memory Definitions



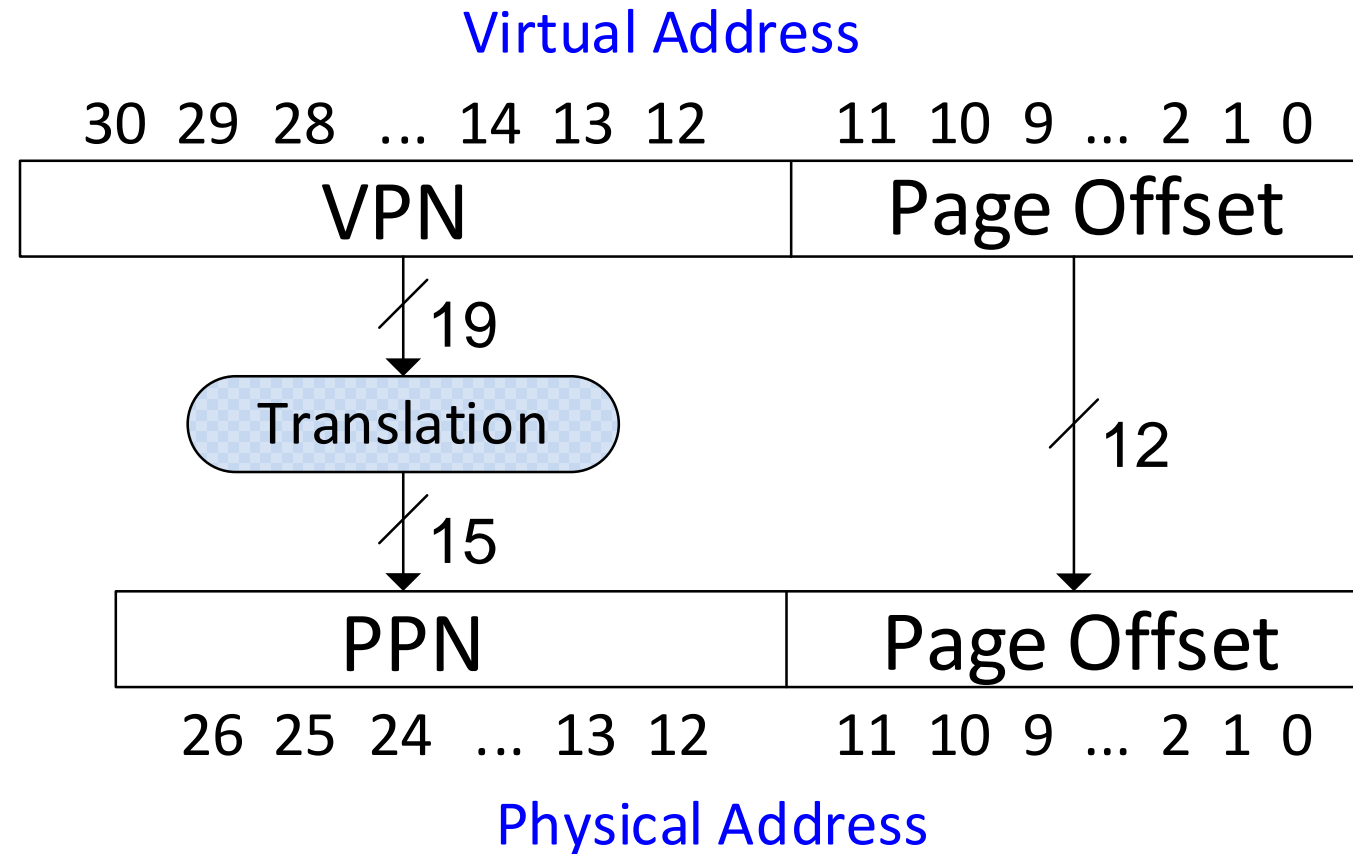
© 2007 Elsevier, Inc. All rights reserved

Most accesses hit in physical memory

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

Address Translation

Address Translation



Virtual Memory Example

- **System:**

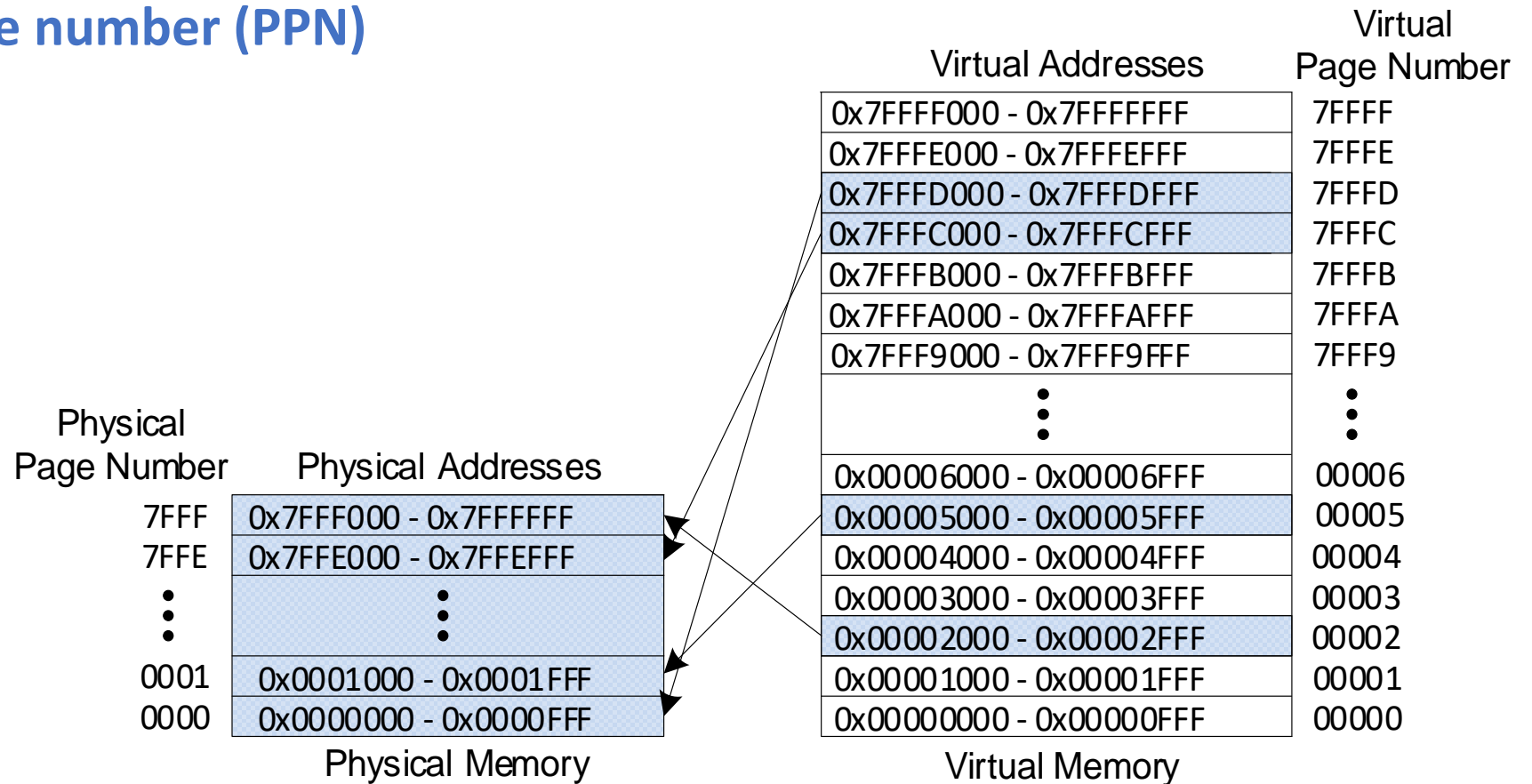
- Virtual memory size: $2\text{ GB} = 2^{31}\text{bytes}$
- Physical memory size: $128\text{ MB} = 2^{27}\text{bytes}$
- Page size: $4\text{ KB} = 2^{12}\text{ bytes}$

- **Organization:**

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

Virtual Memory Example

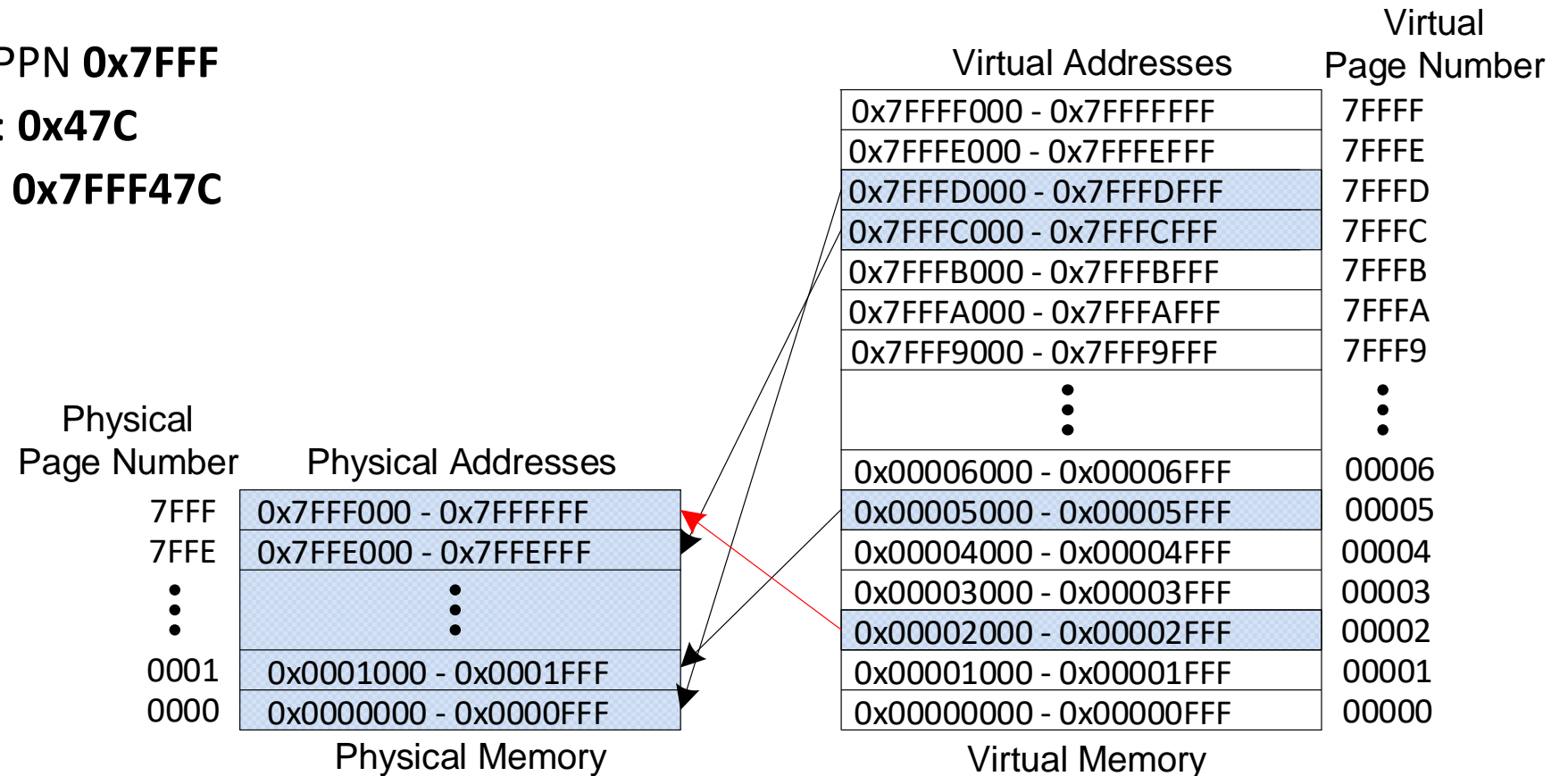
- 19-bit virtual page number (VPN)
- 15-bit physical page number (PPN)



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**



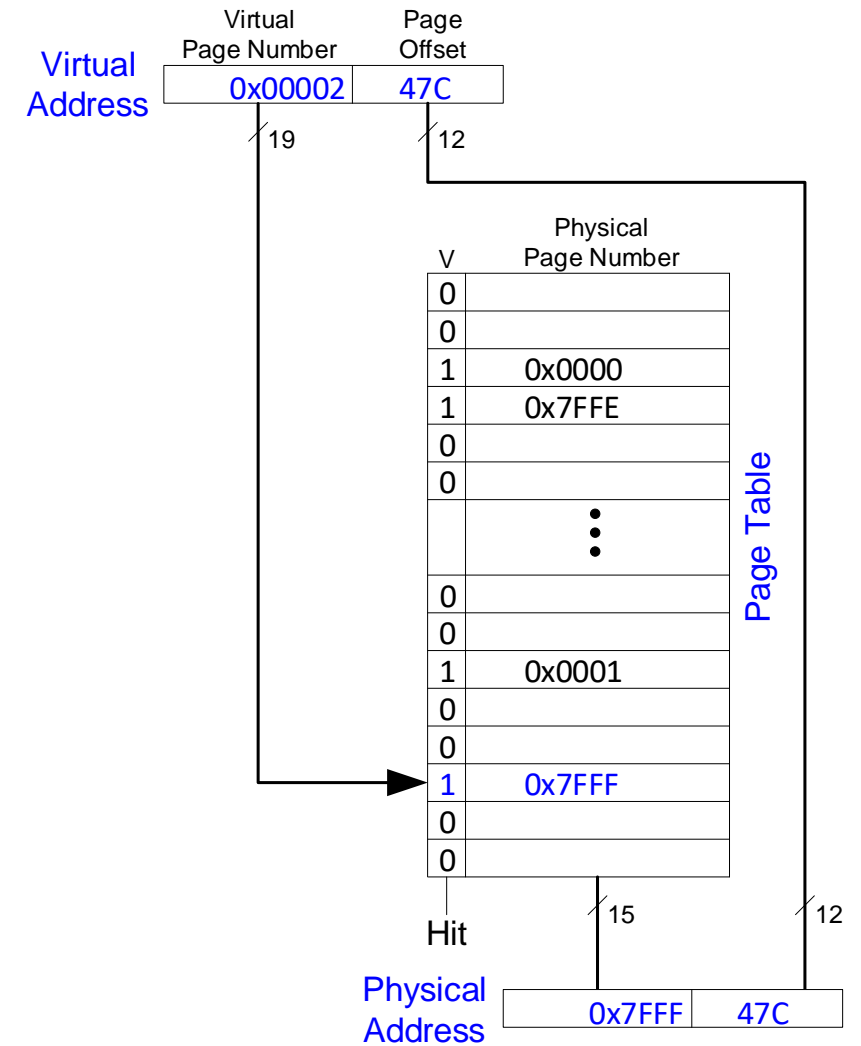
Page Table

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

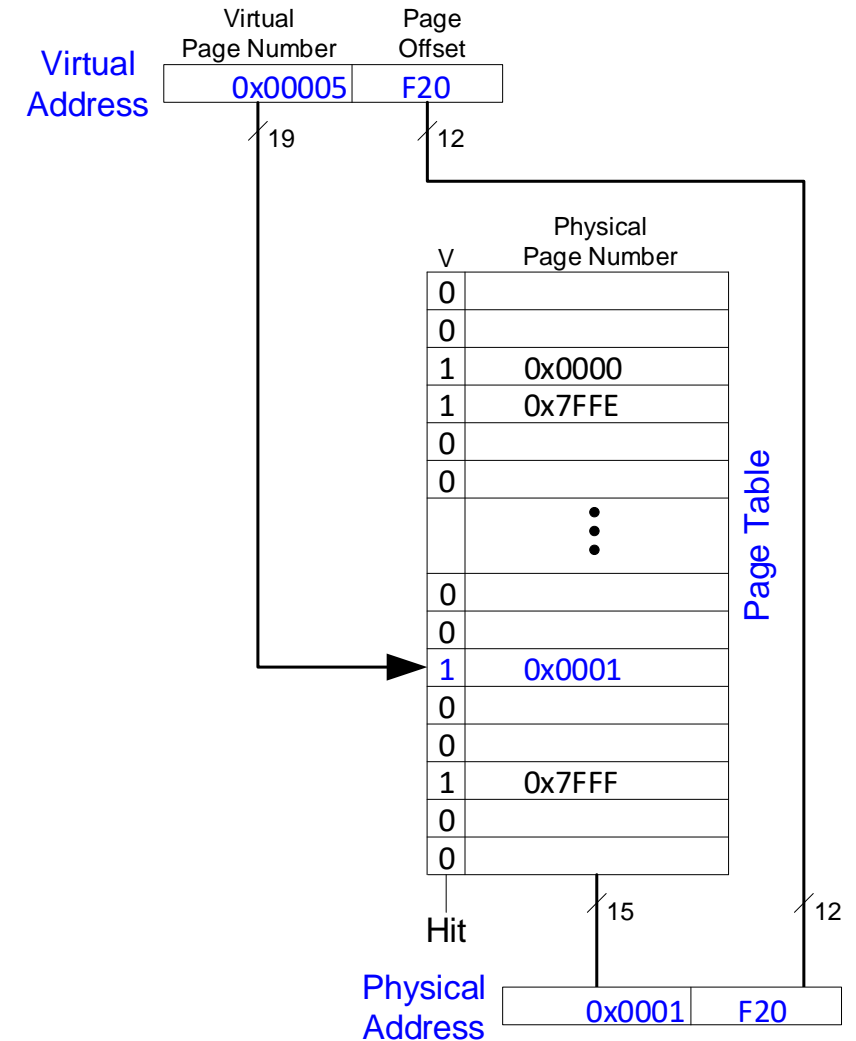
Page Table Example

VPN is **index** into page table



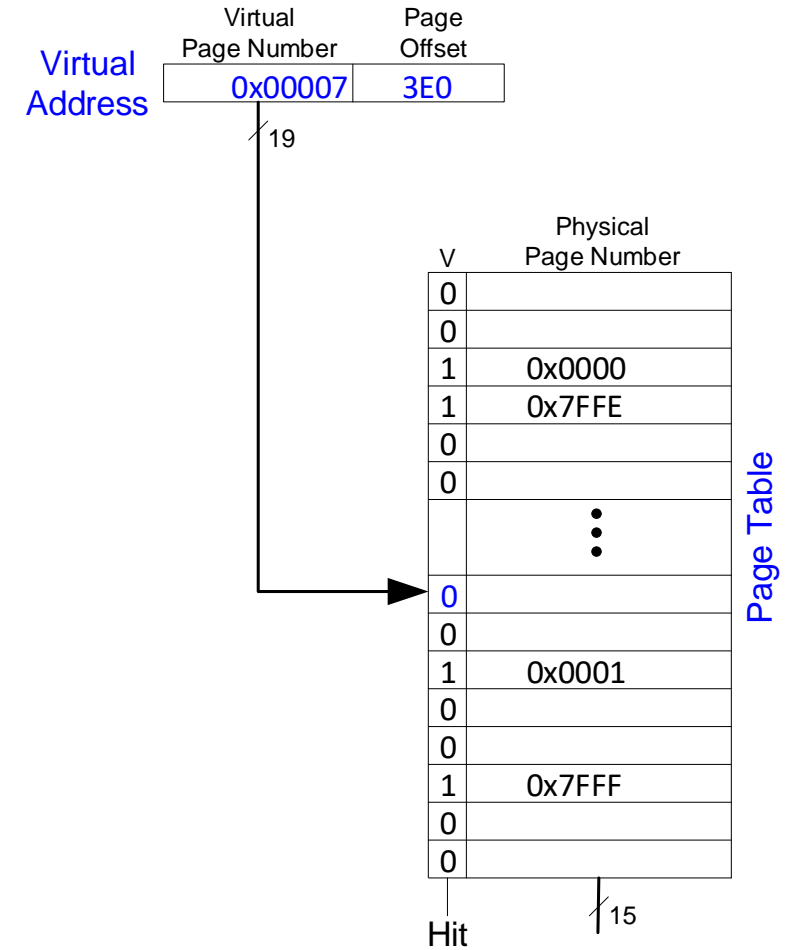
Page Table Example 1

- What is the physical address of virtual address **0x5F20**?
 - VPN = **5**
 - Entry 5 in page table VPN 5 => physical page **1**
 - Physical address: 0x**1**F20



Page Table Example 2

- What is the physical address of virtual address **0x73E4**?
 - VPN = **7**
 - Entry 7 in page table is invalid
 - Virtual page must be **paged** into physical memory from disk



Page Table Challenges

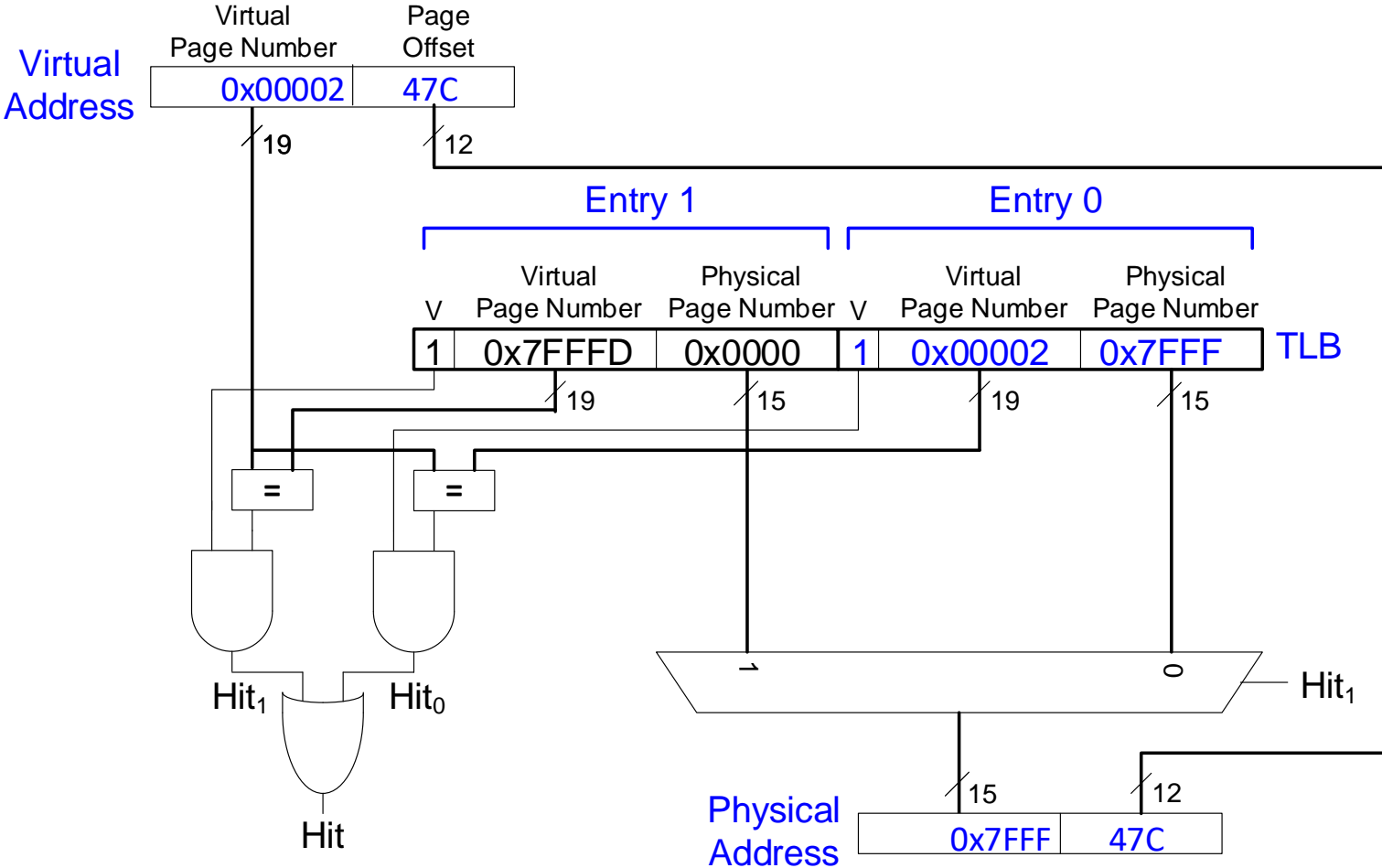
- Page table is large
 - Usually located in physical memory
- Load/store requires 2 main memory accesses:
 - One for translation (page table read)
 - One to access data (after translation)
- Cuts memory performance in half
 - Unless we get clever... by using a Translation Lookaside Buffer (TLB)

Translation Lookaside Buffer (TLB)

Translation Lookaside Buffer (TLB)

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

Example: 2-entry TLB



Virtual Memory Summary

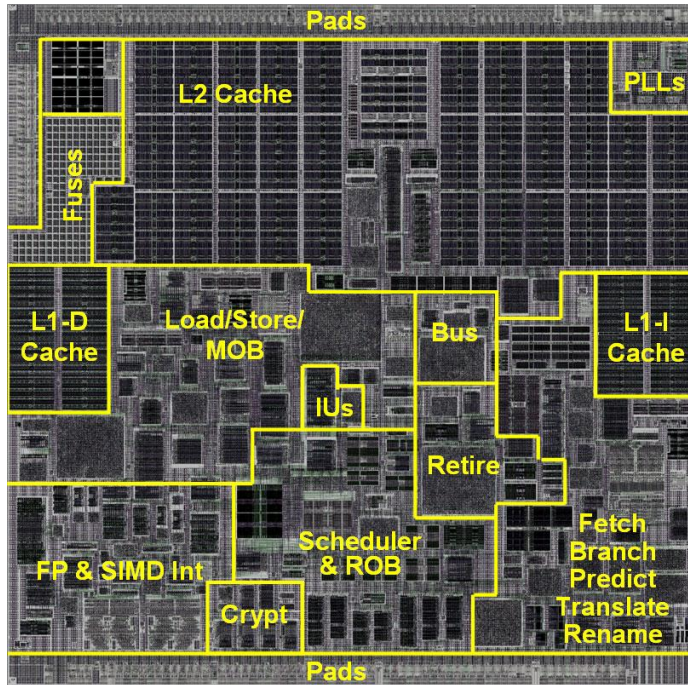
- **Multiple processes** (programs) run at once
- Each process has its **own page table**
- Each process can use **entire virtual address space**
- A process can only access a **subset of physical pages**: those mapped in its own page table

Virtual Memory Summary

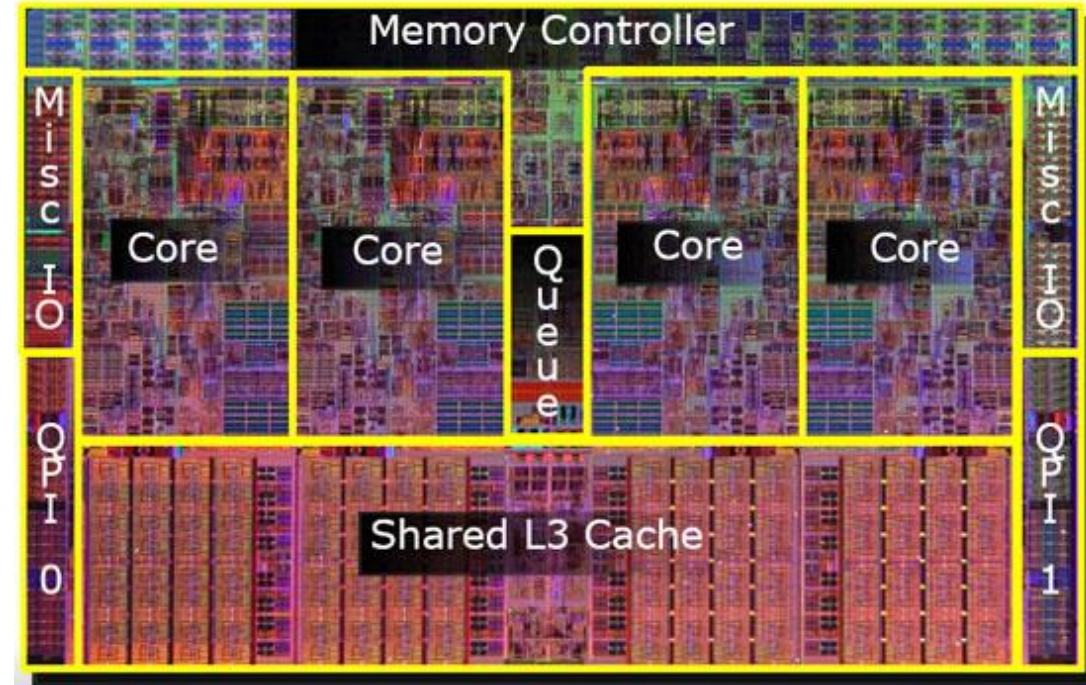
- Virtual memory increases **capacity**
- A subset of virtual pages in physical memory
- **Page table** maps virtual pages to physical pages – address translation
- A **TLB** speeds up address translation
- Different page tables for different programs provides **memory protection**

More

Aktuelle Prozessoren



Via Nano 1,8 GHz (z.B. in eeePCs)



Intel Core i7 950

Bsp.: i7 Nehalem

| | ARM Cortex-A8 (Raspberry v1) | Intel i7 Nehalem |
|------------------------|---------------------------------------|---|
| L1 Organisation | Geteilt Instruktionen / Daten | Geteilt Instruktionen / Daten |
| L1 Size | 32KiB per Instruktion/Data | 32KiB per Instruktion/Data per Core |
| L1 Associative | 4-way (I), 4-way (D) Set Associative | 4-way (I), 8-way (D) Set Associative |
| L1 replacement | Random | Approximated LRU |
| L1 block size | 64 byte | 64 byte |
| L1 write policy | Write-back, Write-allocate (?) | Write-back, No-write-allocate |
| L1 hit time (load-use) | 1 clock cycle | 4 clock cycles, pipelined |
| | | |
| L2 Organisation | Ein Cache für Instruktionen und Daten | Ein Cache für Instruktionen und Daten, per Core |
| L2 Size | 128KiB to 1MiB | 256KiB, per Core |
| L2 Associative | 8-way Set Associative | 8-way Set Associative |
| L2 replacement | Random (?) | Approximated LRU |
| L2 block size | 64 bytes | 64 byte |
| L2 write policy | Write-back, Write-allocate (?) | Write-back, Write-allocate |
| L2 hit time | 11 clock cycles | 10 clock cycles, pipelined |
| | | |
| L3 Organisation | | Ein Cache für Instruktionen und Daten |
| L3 Size | | 8 MiB, shared |
| L3 Associative | | 16-way Set Associative |
| L3 replacement | | Approximated LRU |
| L3 block size | | 64 byte |
| L3 write policy | | Write-back, Write-allocate |
| L3 hit time | | 35 clock cycles, pipelined |