


COSC175 (Systems I): Computer Organization & Design

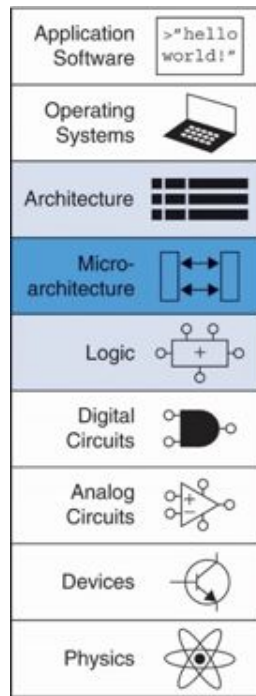


Professor Lillian Pentecost
Fall 2024



Warm-Up December 5

- Where we were
 - Pipelining
- Where we are going
 - Realistic Memory Resources & regaining performance via **Caching**
- Logistics, Reminders
 - TA help 7-9PM on Sundays, Tuesdays, Thursdays in C107
 - Use to start review of whole semester!
 - LP Office hours **DELAYED today 3:10-4PM**
 - **ATTEND FACULTY CANDIDATE TALK TODAY AT 4:30PM in A131, WITH SNACKS @ 4PM in C209**
 - Final Exam announcements @ end of class



Goals for Today

- **What about other tricks that break the programmer's perspective?**
- **What about realistic Memory Resources?**
- **How does cache memory help performance?**

Your microarchitecture plays plenty of tricks to go faster, just maintains the illusion of sequential, accurate program execution

Techniques

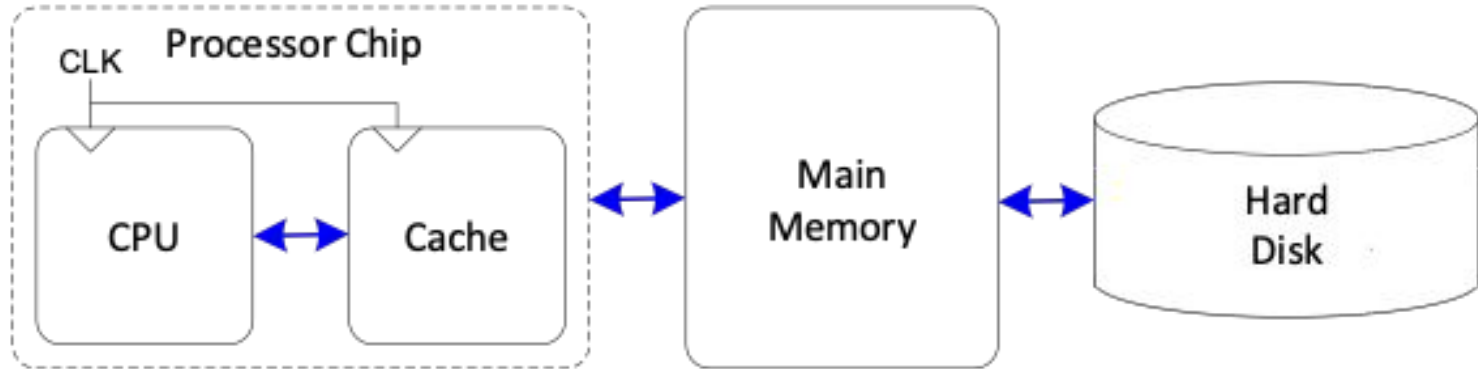
- Deep Pipelining
- Micro-operations
- **Branch Prediction**
- Superscalar Processors
- **Out of Order Processors**
- **Register Renaming**
- SIMD
- Multithreading
- Multiprocessors

Yet another reason to learn more systems: If you know the capabilities and tricks of your machine, you can write better software that takes advantage of these tricks!

Example: let's understand realistic memory resources, and how certain programs make good or bad use of those memory resources

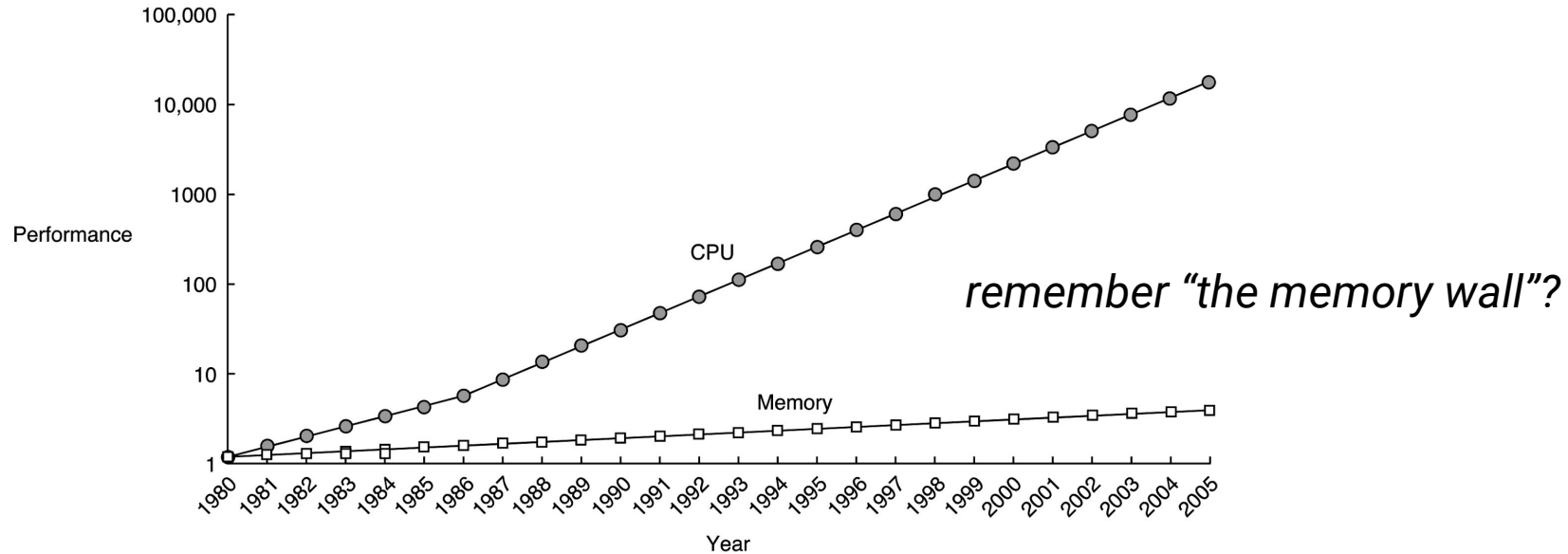
Memory Access Dictates System Performance

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



Processor-Memory Gap

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



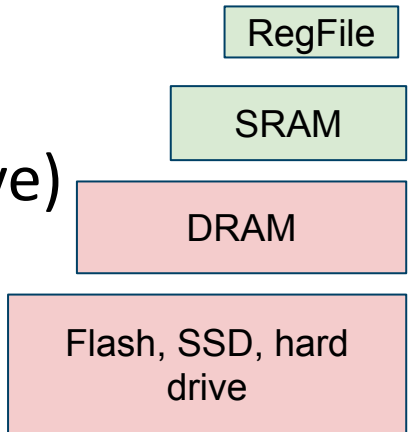
Memory System Challenge

- Make memory system appear as fast as processor

- Use hierarchy of memories

- Ideal memory:

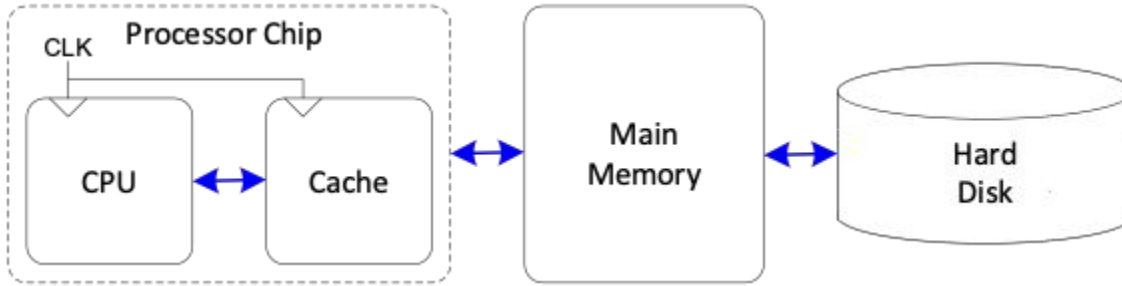
- **Fast**
- **Cheap** (inexpensive)
- **Large** (capacity)



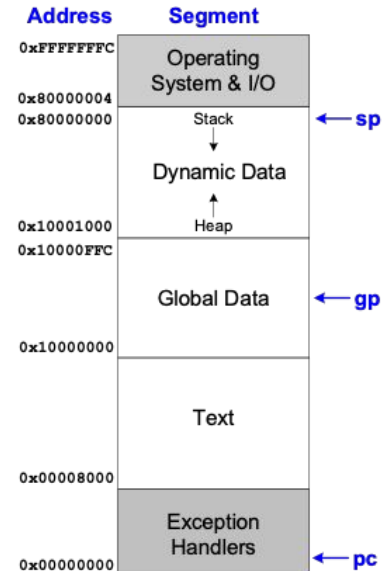
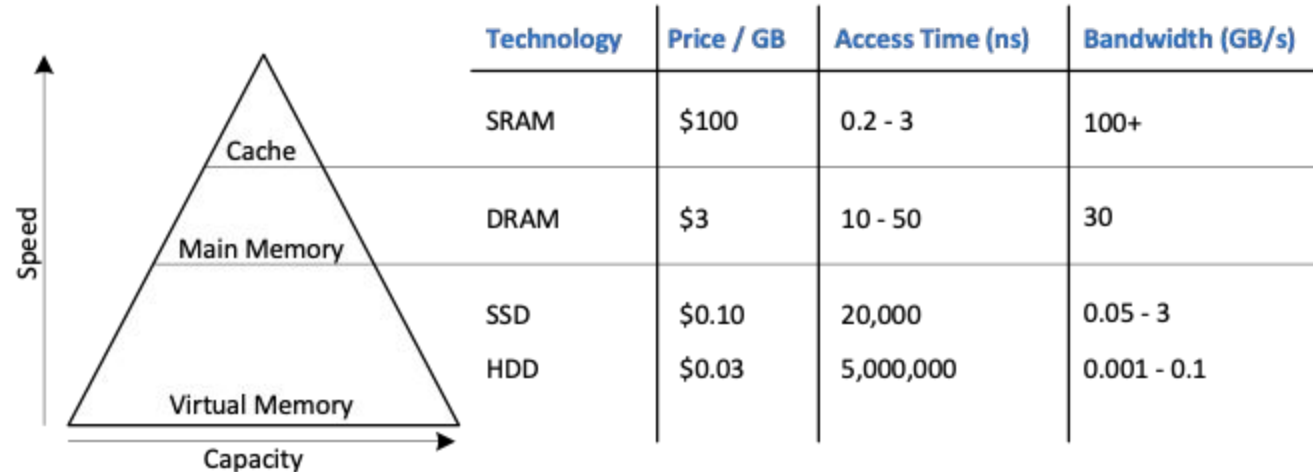
	Capacity	Read Latency (approx)
RegFile	< 1KB	10-100ps
SRAM	1-64MB	1-10ns
DRAM	4-64GB	~100ns (maybe more)
Flash, SSD, hard drive	256-1024GB	~100us

- **But can only choose two!**

Memory Hierarchy



How to determine what information (i.e., which instructions, which data) should be kept closer to the CPU?



Program behavior dictates systems choices → we rely on **locality**

- Programs are unique, but tend to exhibit **locality** of reference; this is our guiding principle for identifying + refining the policies and designs in computer systems
- **Temporal Locality:**
 - if a program accesses a certain address once, it is likely to access the same address again
- **Spatial Locality:**
 - If a program accesses a certain address, it is likely to access addresses close to that certain address too.

Temporal Locality

- *Definition:* if a program accesses a certain address once, it is likely to access the same address again
- *Example:* Literally any loop ever
 - Repeatedly accesses the same addresses to access the **instructions** over and over again
 - Repeatedly access the same **data** within a loop
 - overwrite a value, update a counter that is at a memory location

// a snippet of C; s and i both exhibit temporal locality

```
int s = 0;
for (int i=0; i<100; i++){
    s = s + i;
}
```

Spatial Locality

- *Definition:* If a program accesses a certain address, it is likely to access addresses close to that certain address too.
- *Example:* Still literally any loop, but for different reasons
 - **Instructions** tend to follow one-after-another, or often jump within page-sized blocks
 - **Data** in arrays, fields in a struct, etc. are stored contiguously in the address space

```
// a snippet of C; data exhibits spatial locality
```

```
int s = 0;
int data[100];
for (int i=0; i<100; i++){
    s = s + i;
    data[i] = s;
}
```

Program behavior dictates systems choices → we rely on **locality**

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

- **Temporal Locality:**

- 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:**

- 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

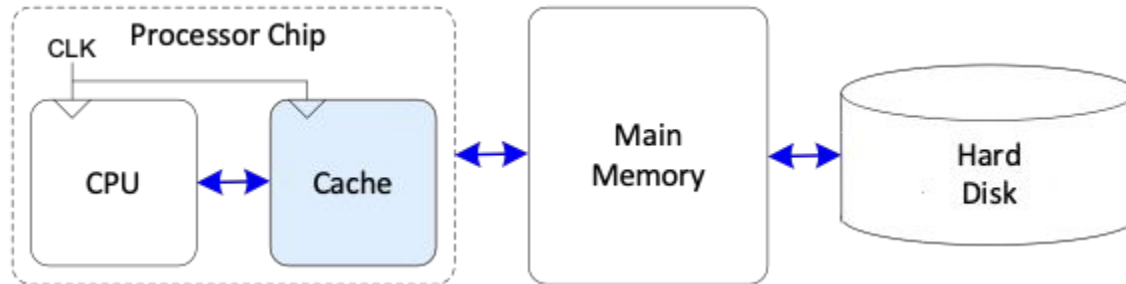
Check-In Activity: Exploring Locality



(see activity worksheet)

Cache

- Highest level in memory hierarchy
- Fast (typically ~ 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?

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

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 ($B = 8$)

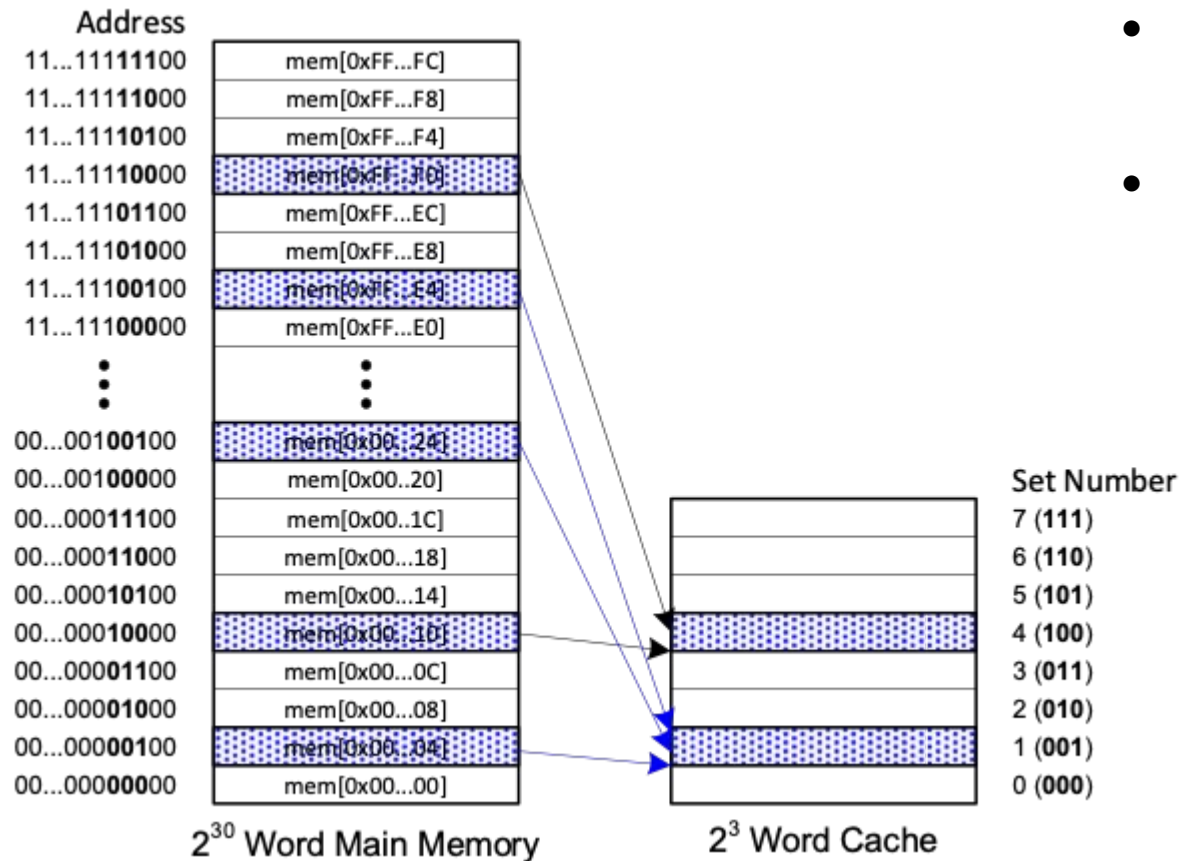
Example Cache Parameters

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

Ridiculously small, but will illustrate organizations

Even a small cache can have a big performance impact

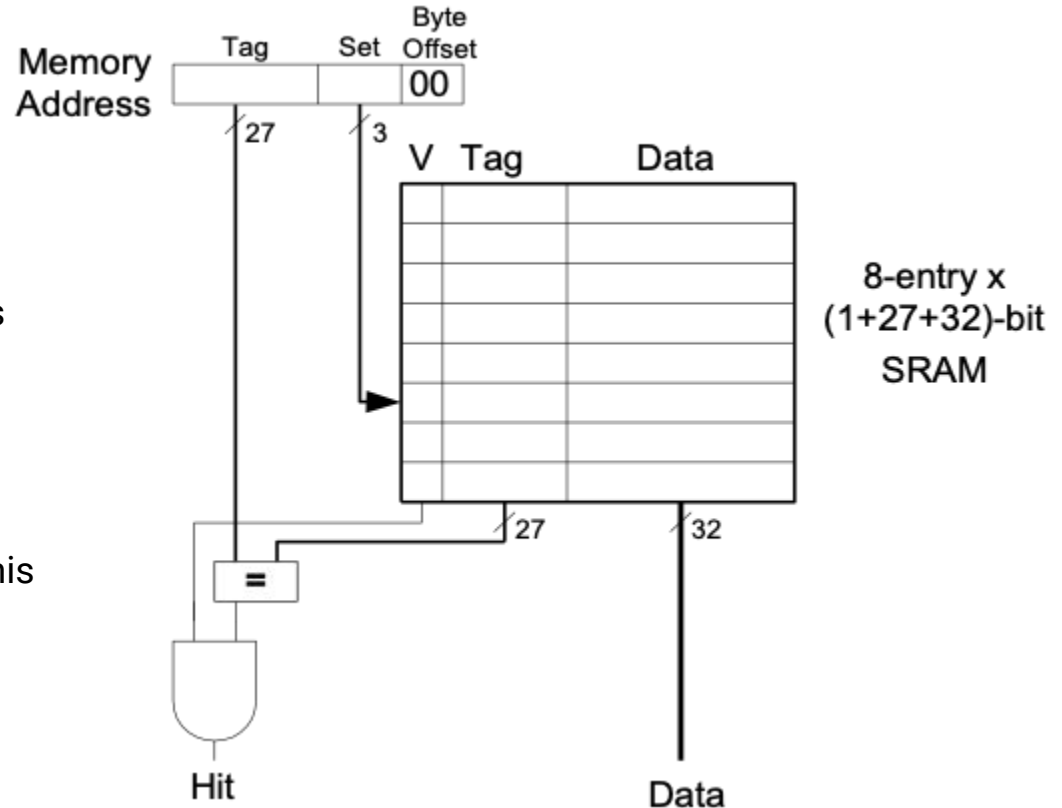
Direct Mapped Cache



- Each main memory location maps to a particular cache set, using the least significant bits as indicator/index
- There are multiple potential data blocks possibly held at a given cache index → these are **collisions**
- **How do we distinguish between them?**
 - Using some **metadata** to uniquely identify which memory address this data corresponds to
 - “tag” with higher address bits

Direct Mapped Cache Hardware

- Each main memory location maps to a particular cache line number, using the least significant bits as indicator/index
- There are multiple potential data blocks possibly held at a given cache index → these are **collisions**
- **How do we distinguish between them?**
 - Using some **metadata** to uniquely identify which memory address this data corresponds to
 - “tag” with higher address bits



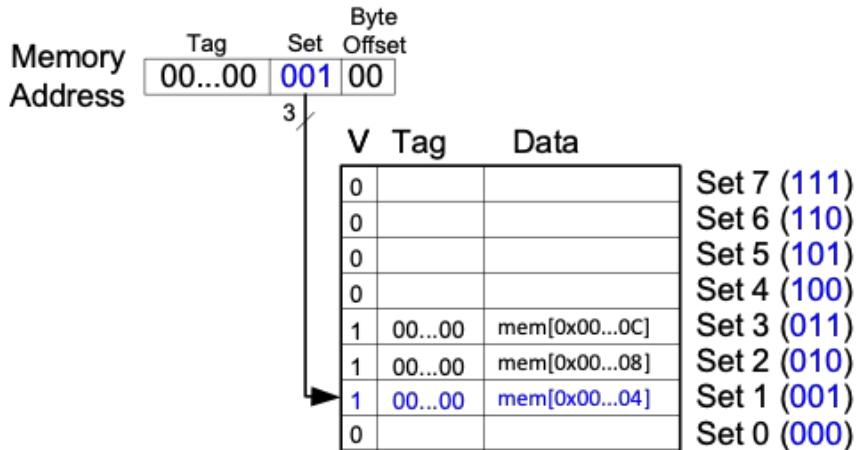
Direct Mapped Cache Performance

Cache starts empty:

first three accesses are 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:
```



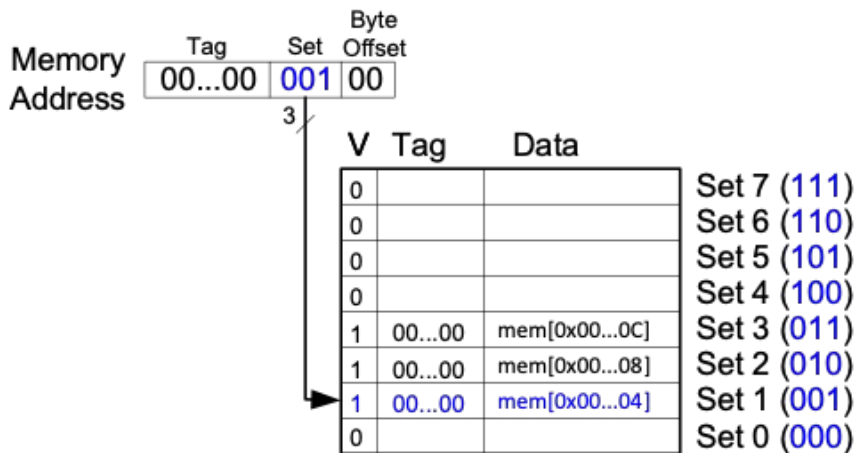
Direct Mapped Cache Performance

Cache starts empty:

first three accesses are 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:
```



Miss Rate = 3/15

= 20%

Temporal Locality

Compulsory Misses

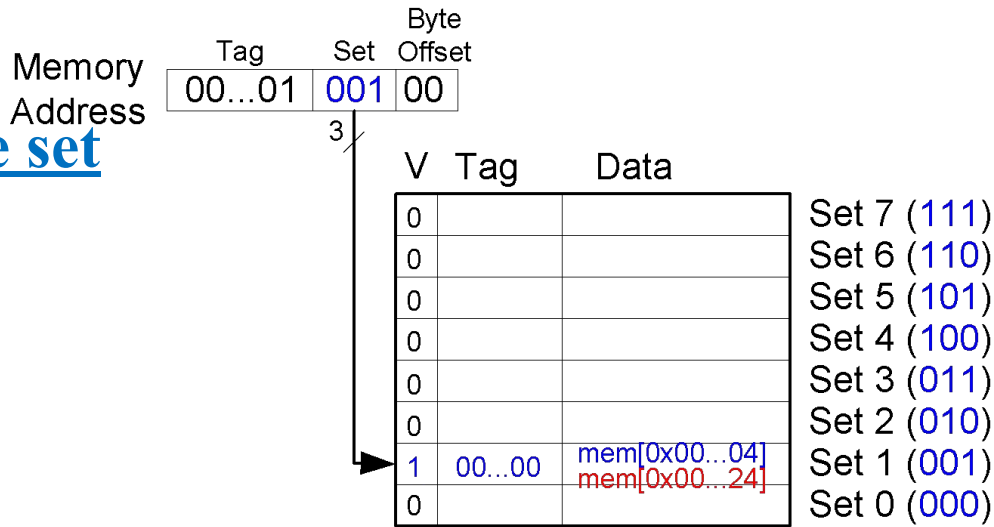
Direct Mapped Cache: Conflict Miss

Cache starts empty:

0x4 and 0x24 map to same set

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:
```



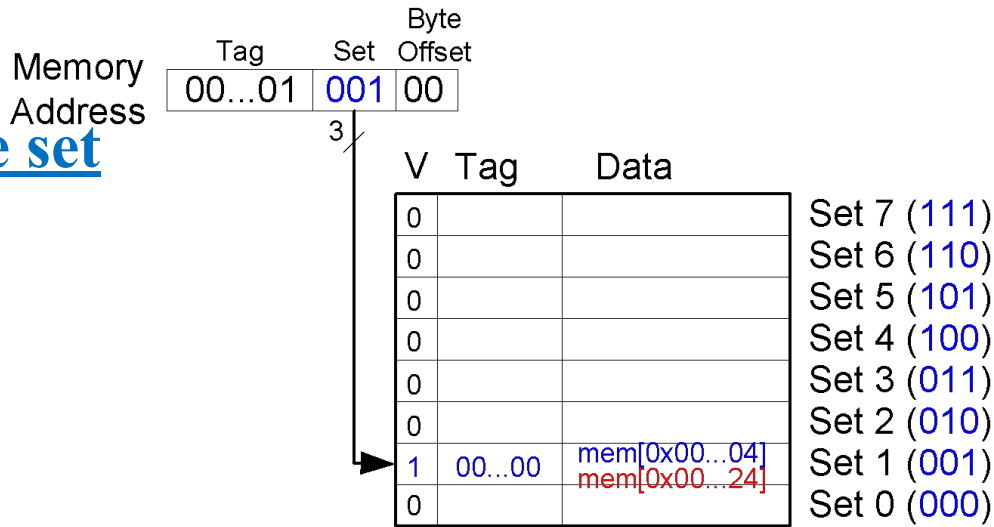
Direct Mapped Cache: Conflict Miss

Cache starts empty:

0x4 and 0x24 map to same set

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:
```



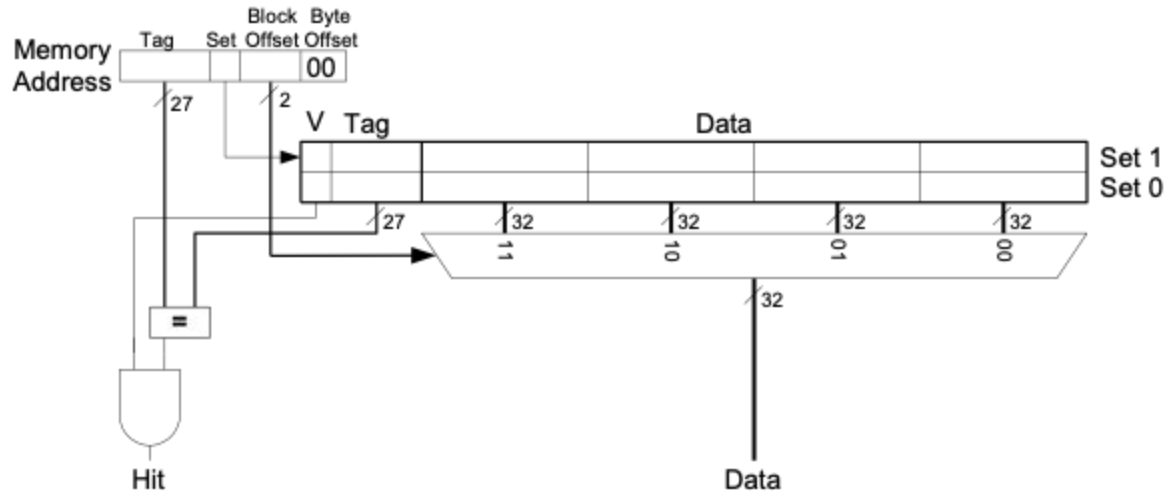
Miss Rate = 10/10

= 100%

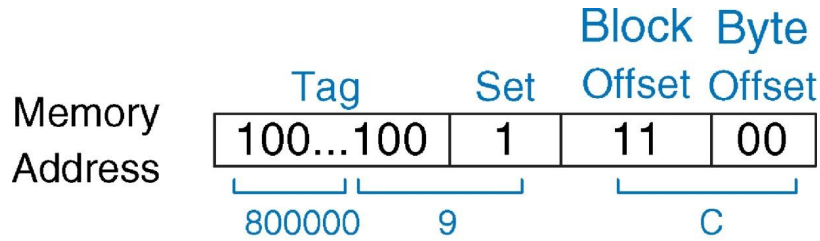
Conflict Misses

What about spatial locality

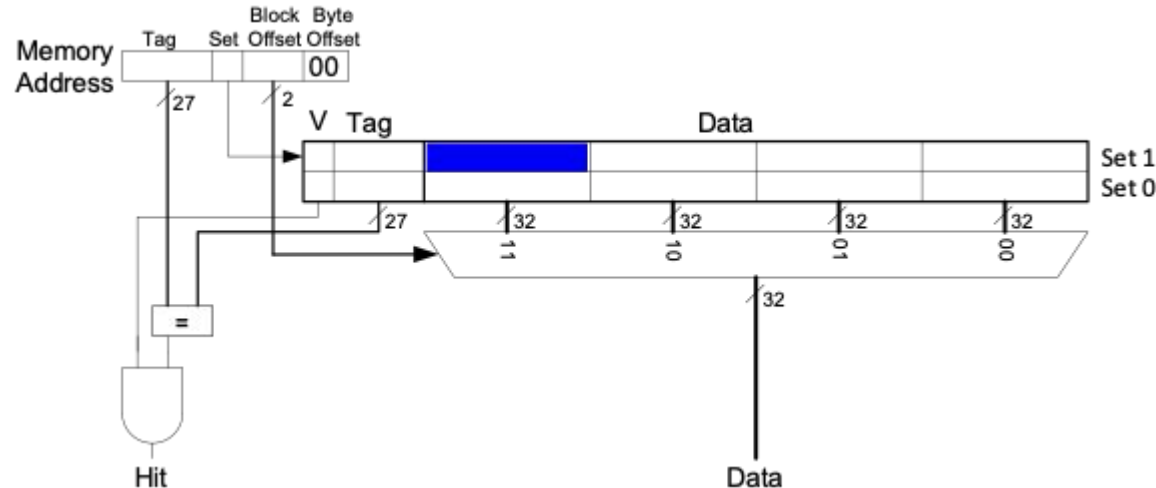
- Increase block size:
 - Block size, **$b = 4$ words**
 - $C = 8$ words
 - Direct mapped (1 block per set)
 - Number of blocks, **$B = 2$** ($C/b = 8/4 = 2$)



Cache with Larger Block Size



© 2007 Elsevier, Inc. All rights reserved



Cache Perf. with Spatial Locality

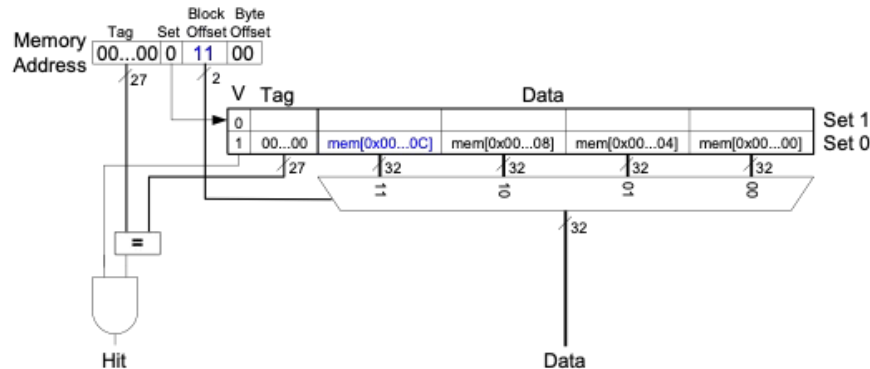
```
      addi s0, zero, 5
      addi s1, zero, 0
LOOP: beq  s0, zero, DONE
      lw   s2, 4(s1)
      lw   s3, 12(s1)
      lw   s4, 8(s1)
      addi s0, s0, -1
      j    LOOP
```

DONE :

Miss Rate = 1/15

= 6.67%

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



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

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

Memory Performance

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

$$\begin{aligned}\text{Hit Rate} &= \# \text{ hits} / \# \text{ memory accesses} \\ &= 1 - \text{Miss Rate}\end{aligned}$$

$$\begin{aligned}\text{Miss Rate} &= \# \text{ misses} / \# \text{ memory accesses} \\ &= 1 - \text{Hit Rate}\end{aligned}$$

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

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

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 **cache hit and miss rates?**

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

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

Memory Performance Example 2

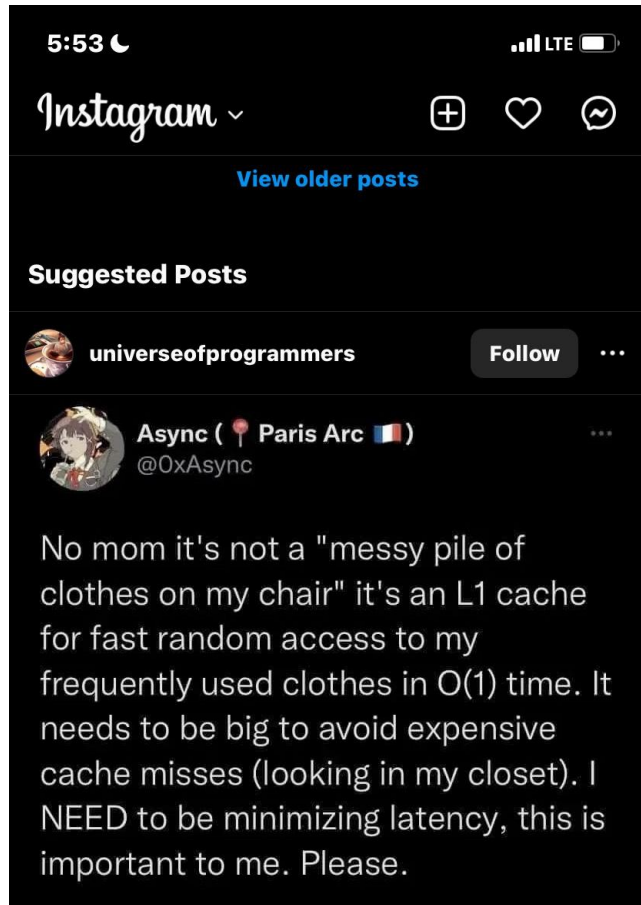
- Suppose processor has 2 levels of hierarchy: cache and main memory
- $t_{\text{cache}} = 1$ cycle, $t_{MM} = 100$ cycles
- What is the **AMAT** (average memory access time) of the program from Example 1?

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

Wrap-Up December 5

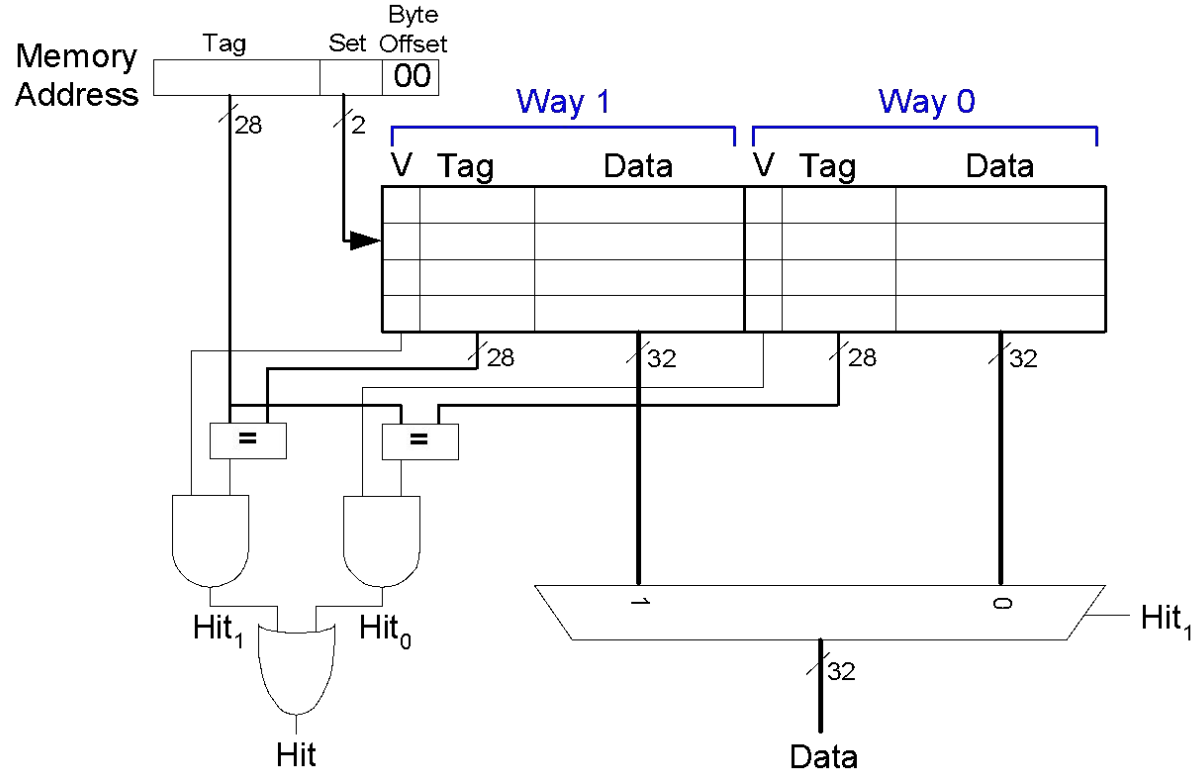


- Coming up next!
 - Looking ahead to COSC275; advanced topic
- Logistics, Reminders
 - TA help 7-9PM on Sundays, Tuesdays, Thursdays in C107
 - LP Office hours **DELAYED today 3:10-4PM**
 - Final Exam special review session Wednesday, Dec. 18 6-8PM
 - Final Exam 9AM-Noon on Friday, December 20 in Paino Lecture Hall (Beneski)
 - After receiving additional requests, weekly exercise solutions will be posted next week (Dec 8) to help with final exam review
 - Final Lab Report (all stages) due Monday, Dec. 9 10PM
 - **ATTEND FACULTY CANDIDATE TALK TODAY AT 4:30PM in A131, WITH SNACKS @ 4PM in C209**
 - *If you attend at least 3 candidate talks, then send me an email including 1 thing you learned from each talk, I'll give you 5% extra credit on any previous lab report*
 - **Course Evaluations are available!! Please fill out at your leisure!!**
- FEEDBACK
 - <https://forms.gle/5Aafcm3iJthX78jx6>



(Courtesy of Michael Xu '25)

N-Way Set Associative Cache



N-Way Set Assoc. 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:
```

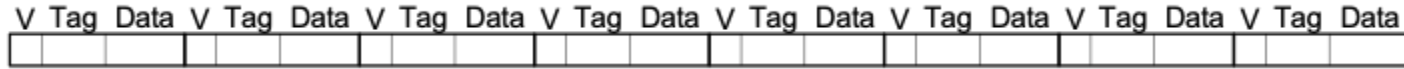
Miss Rate = 2/10

= 20%

**Associativity reduces
conflict misses**

Way 1			Way 0			
V	Tag	Data	V	Tag	Data	
0			0			Set 3
0			0			Set 2
1	00...10	mem[0x00...24]	1	00...00	mem[0x00...04]	Set 1
0			0			Set 0

Fully Associative Cache



Compare against every tag

Reduces conflict misses

Expensive to build

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:** $B = C/b$
- **Number of blocks in a set:** 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

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

LRU Replacement

RISC-V assembly

```
lw s1, 0x04(zero)
```

```
lw s2, 0x24(zero)
```

```
lw s3, 0x54(zero)
```

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)

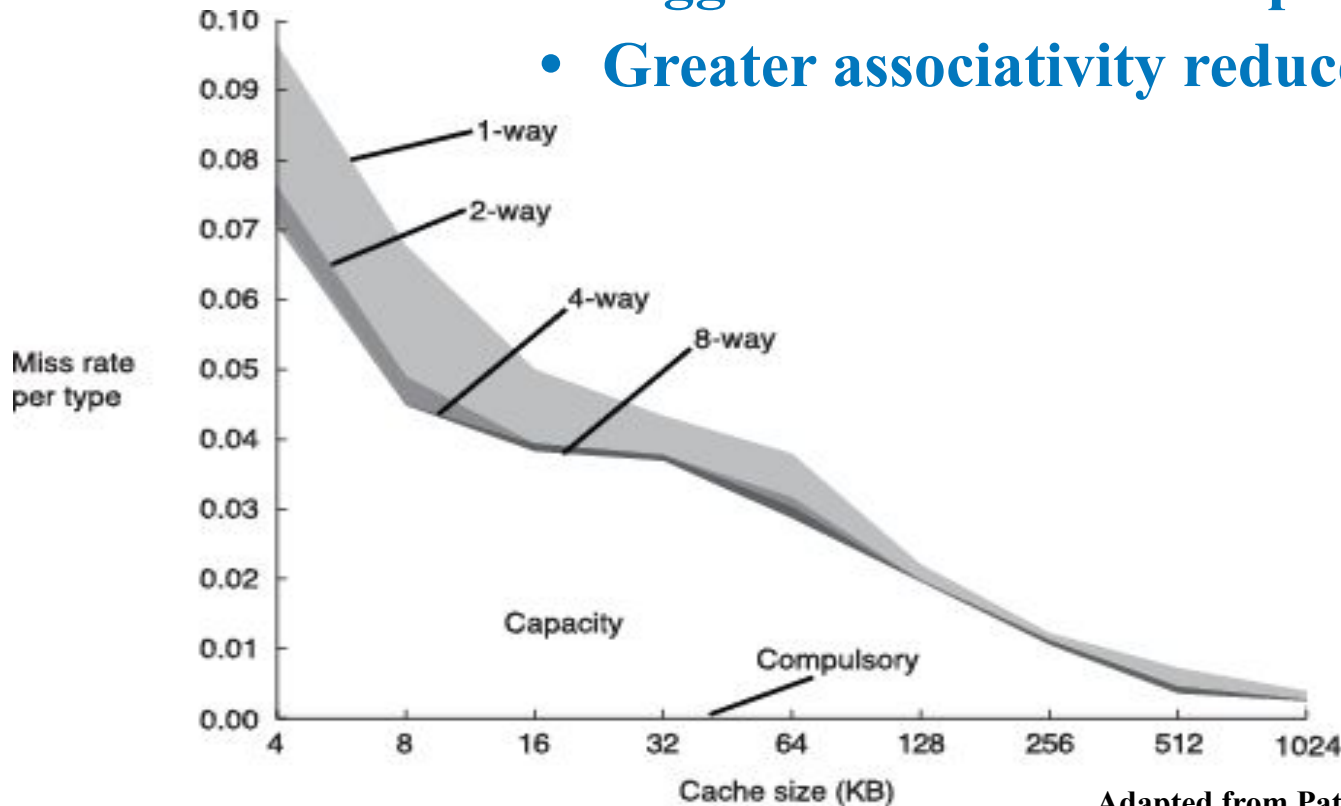
(a)

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)

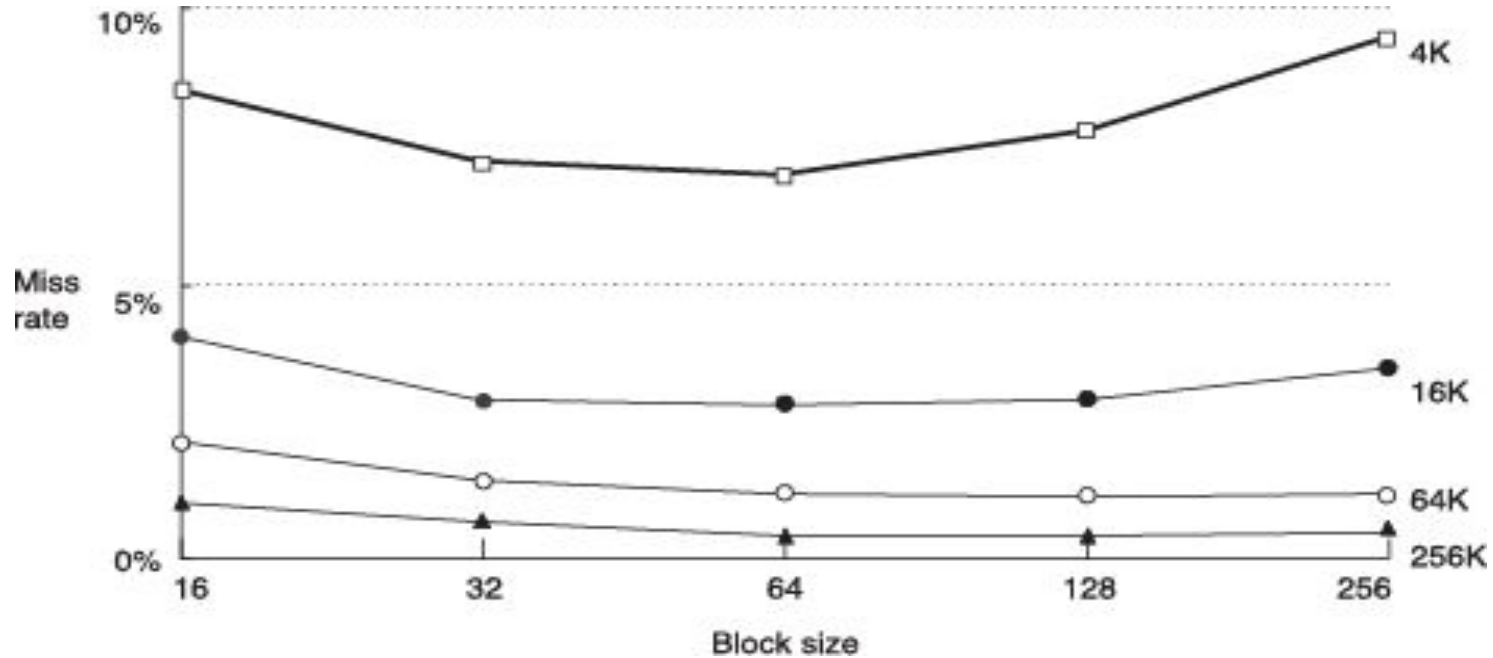
Miss Rate Trends

- 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