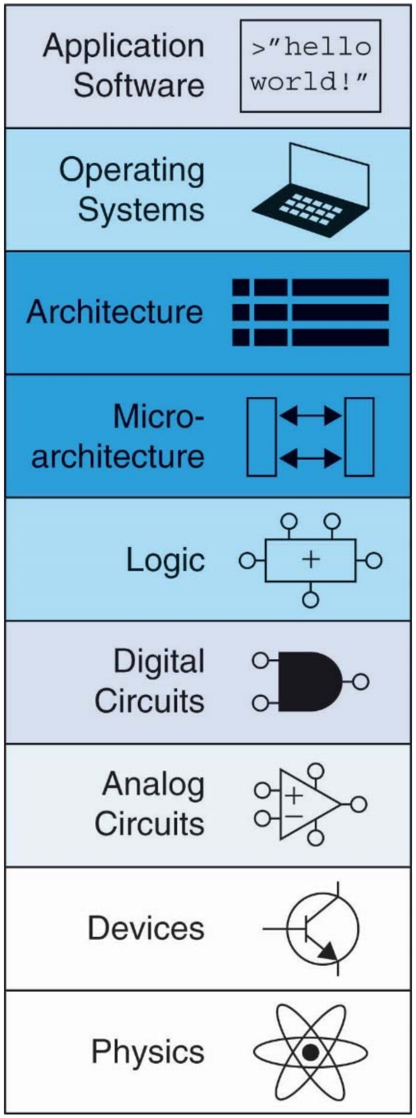
Memory and I/O Systems

Acknowledgments: Slides are adapted from Harris and Harris textbook instructor's material

Chapter 8 :: Topics

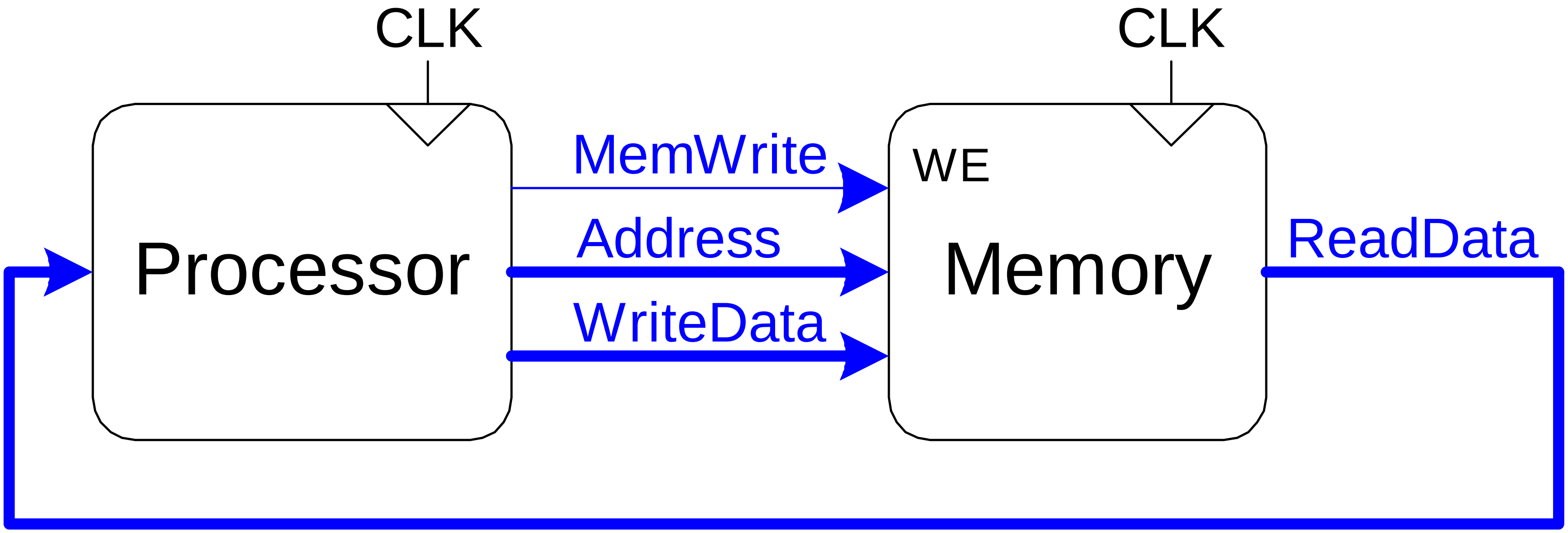
- **Introduction**
- **Memory System Performance Analysis**
- **Caches**
- Virtual Memory
- Memory-Mapped I/O



Introduction

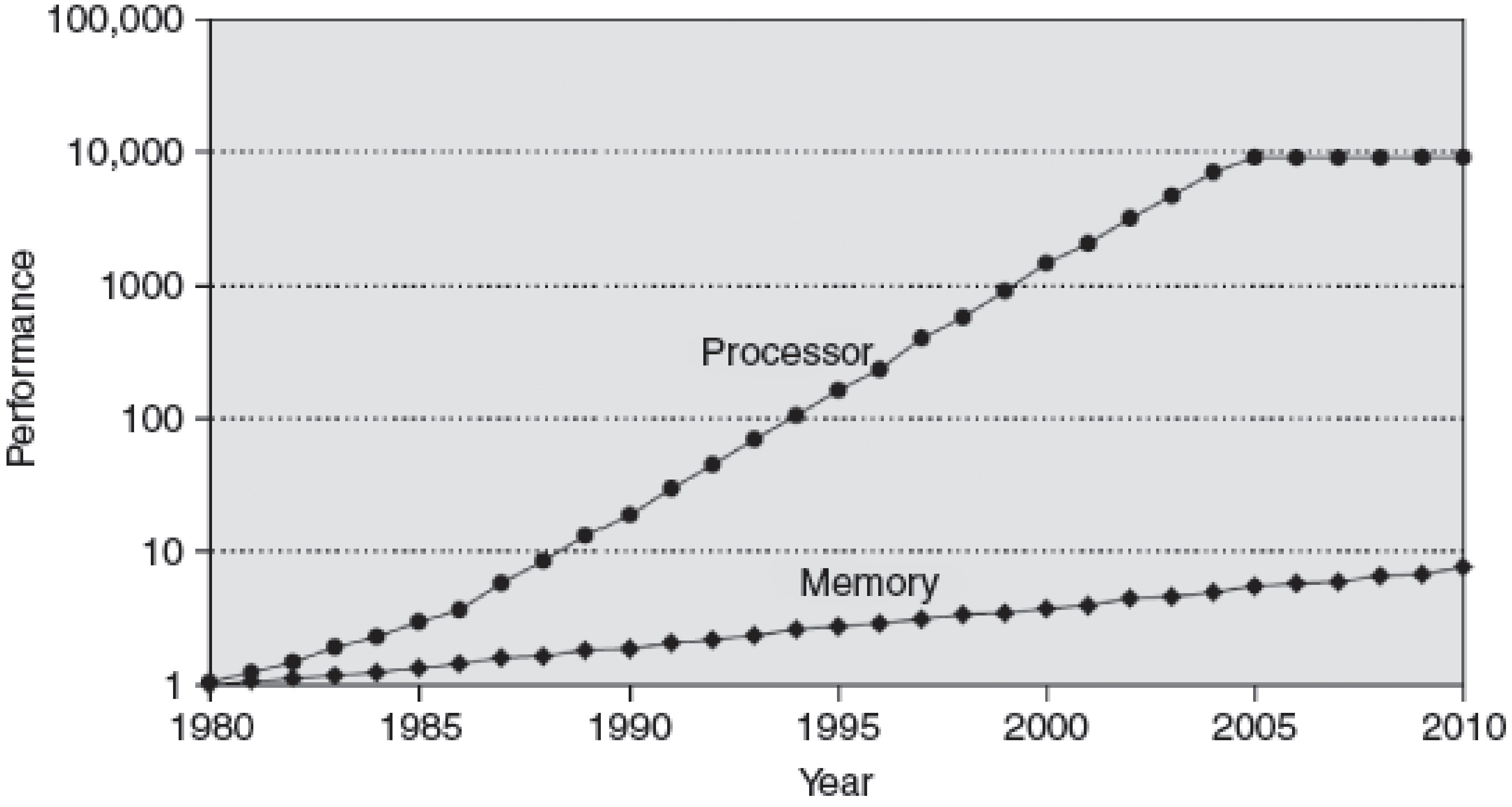
- Computer performance depends on:
  - Processor performance
  - Memory system performance

Memory Interface



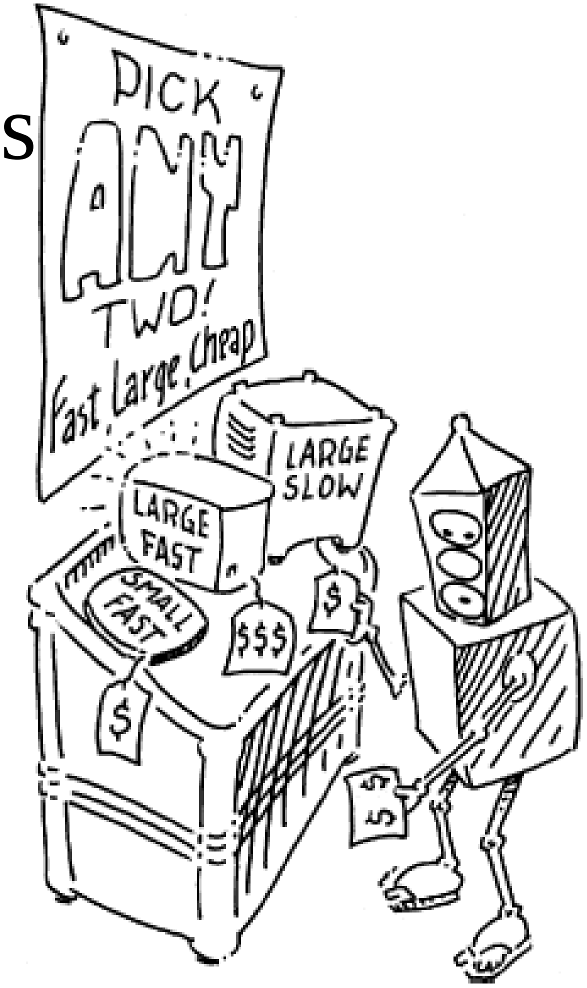
Processor-Memory Gap

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



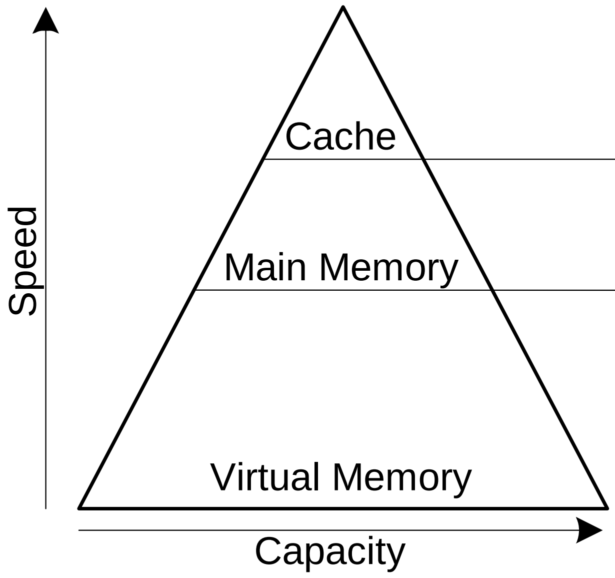
Memory System Challenge

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



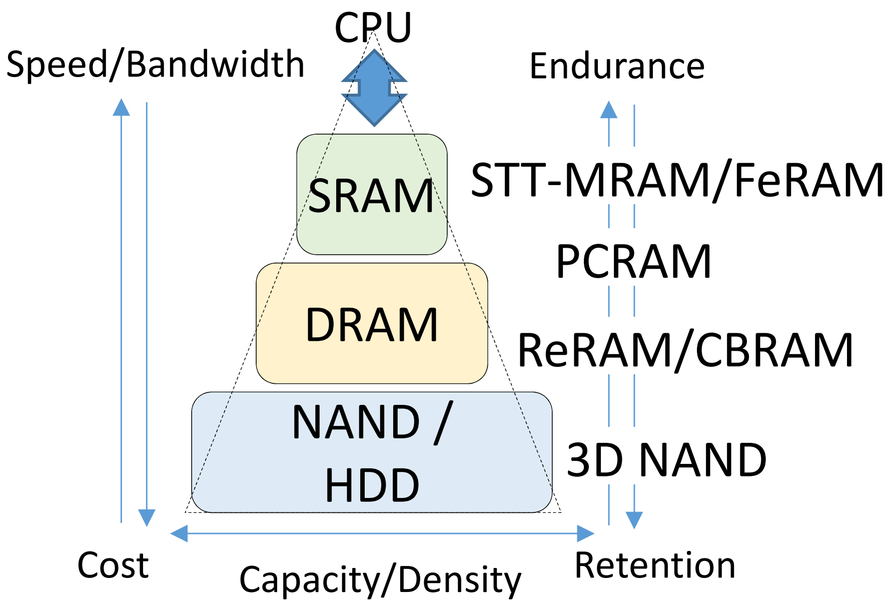
But can only choose two!

Memory Hierarchy



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

Future 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

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?**

Hit Rate = 1250/2000 **= 0.625**

Miss Rate = 750/2000 = **0.375** = 1 – Hit Rate

Memory Performance Example 2

- Suppose 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?**

Memory Performance Example 2

- Suppose 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?**

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

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

$$= \textbf{38.5 cycles}$$

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 ~ 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?

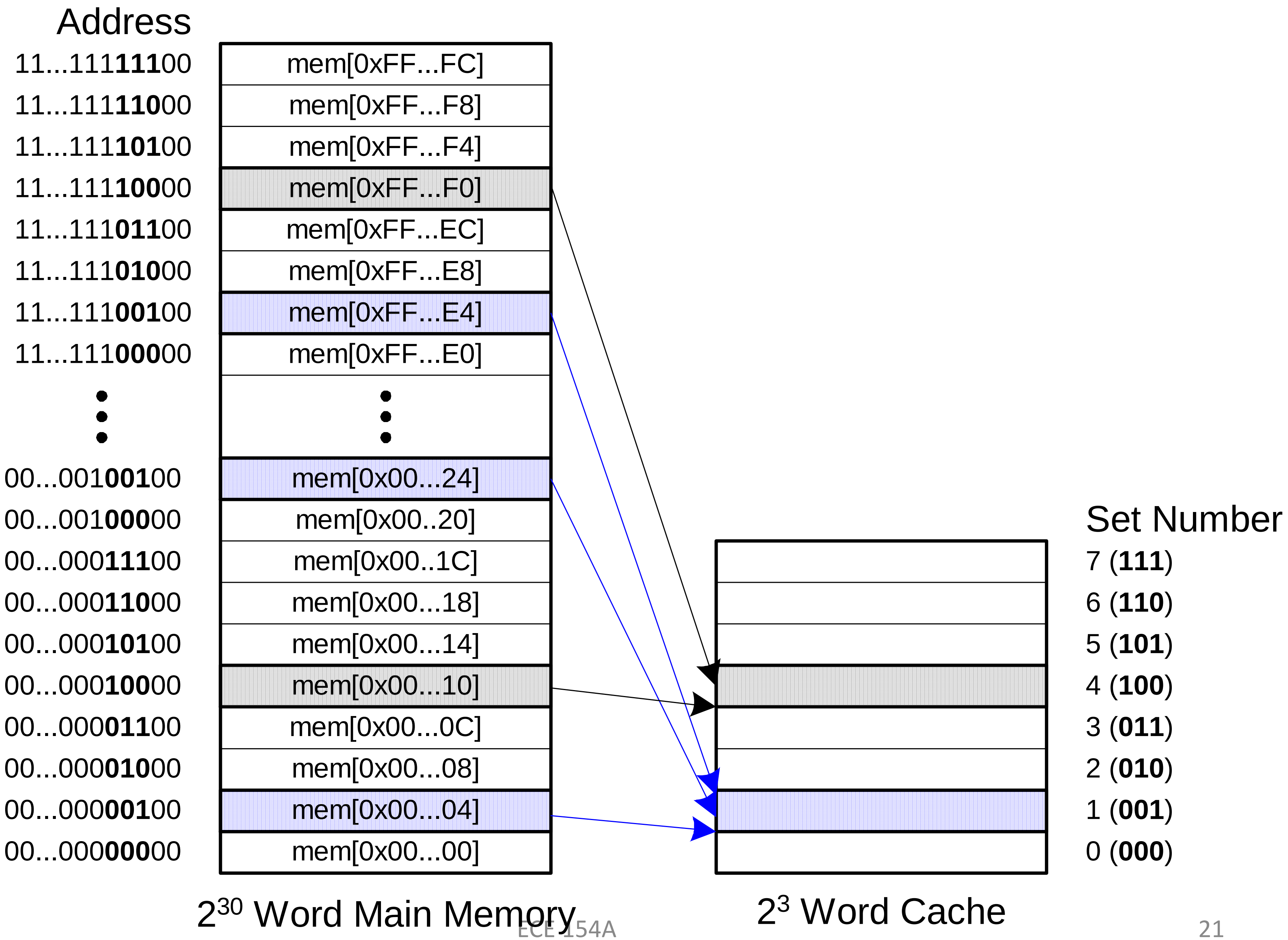
- Cache organized into *S* sets
- Each memory address maps to exactly one set
- Caches categorized by # of blocks in a set:
  - **Direct mapped:** 1 block per set
  - ***N*-way set associative:** *N* blocks per set
  - **Fully associative:** all cache blocks in 1 set
- Examine each organization for a 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



Address	
11...11**111**00	mem[0xFF...FC]
11...11**110**00	mem[0xFF...F8]
11...11**101**00	mem[0xFF...F4]
11...11**100**00	mem[0xFF...F0]
11...11**011**00	mem[0xFF...EC]
11...11**010**00	mem[0xFF...E8]
11...11**001**00	mem[0xFF...E4]
11...11**000**00	mem[0xFF...E0]
⋮	⋮
00...001**001**00	mem[0x00...24]
00...001**000**00	mem[0x00..20]
00...000**111**00	mem[0x00..1C]
00...000**110**00	mem[0x00...18]
00...000**101**00	mem[0x00...14]
00...000**100**00	mem[0x00...10]
00...000**011**00	mem[0x00...0C]
00...000**010**00	mem[0x00...08]
00...000**001**00	mem[0x00...04]
00...000**000**00	mem[0x00...00]

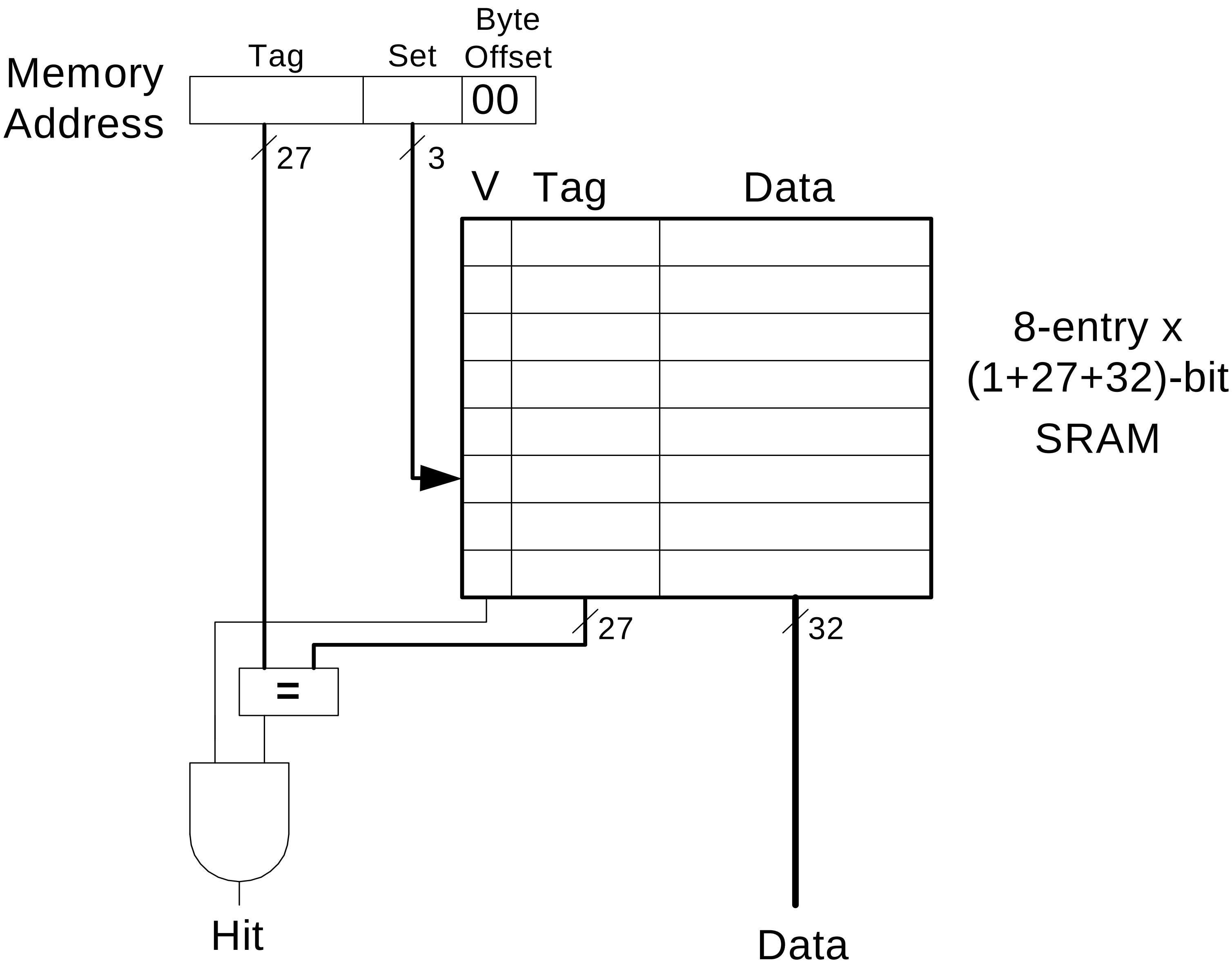
2^{30} Word Main Memory

Set Number

7 (**111**)
6 (**110**)
5 (**101**)
4 (**100**)
3 (**011**)
2 (**010**)
1 (**001**)
0 (**000**)

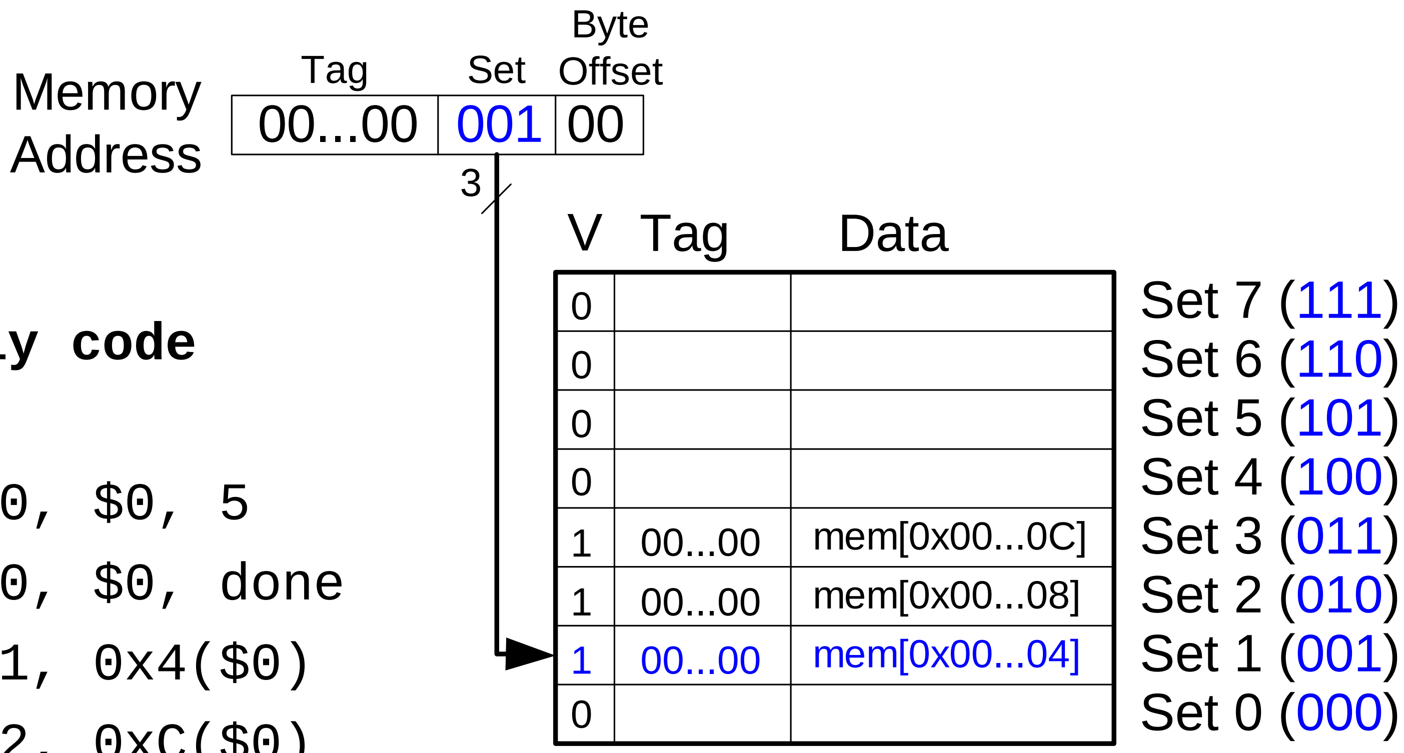
2^{3} Word Cache

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:
```



Memory Address

Tag	Set	Byte Offset
00...00	001	00

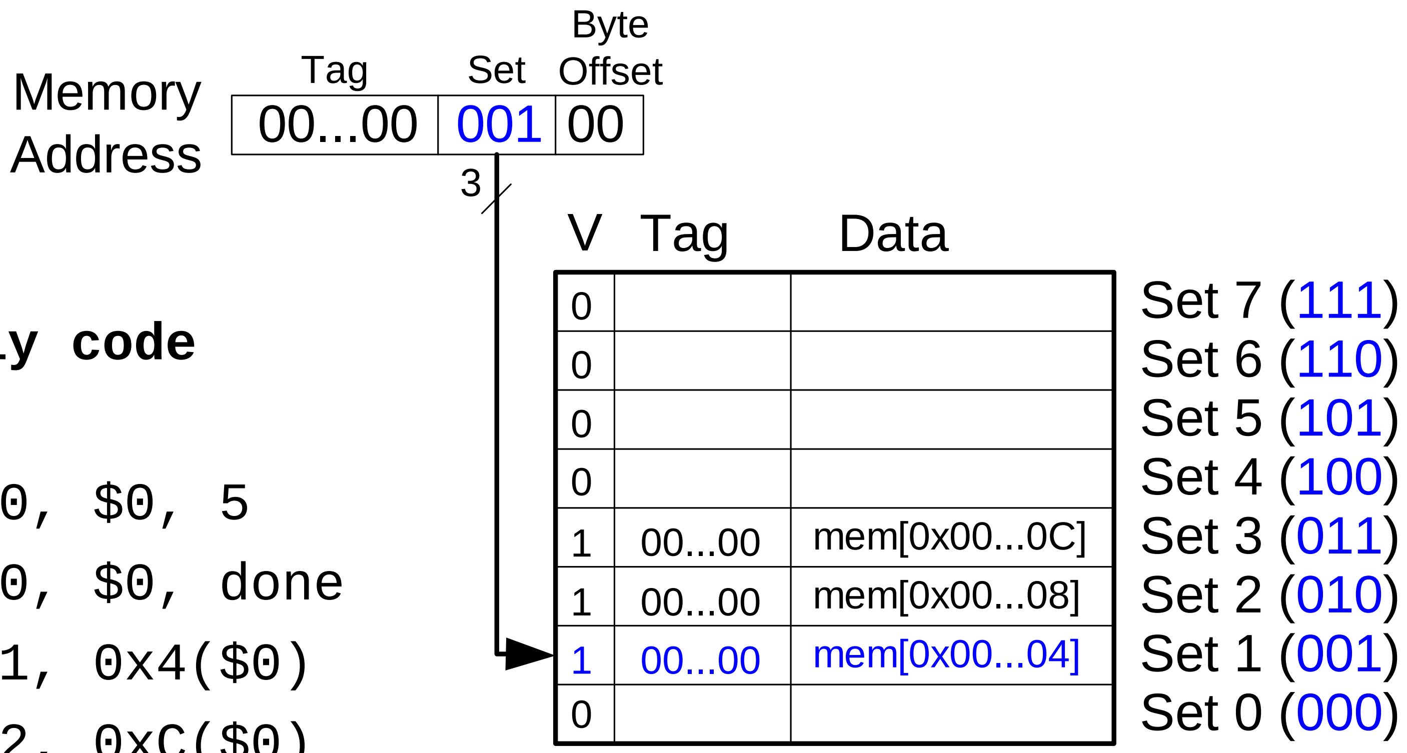
3

V	Tag	Data	
0			Set 7 (111)
0			Set 6 (110)
0			Set 5 (101)
0			Set 4 (100)
1	00...00	mem[0x00...0C]	Set 3 (011)
1	00...00	mem[0x00...08]	Set 2 (010)
1	00...00	mem[0x00...04]	Set 1 (001)
0			Set 0 (000)

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:
```



V	Tag	Data	
0			Set 7 (111)
0			Set 6 (110)
0			Set 5 (101)
0			Set 4 (100)
1	00...00	mem[0x00...0C]	Set 3 (011)
1	00...00	mem[0x00...08]	Set 2 (010)
1	00...00	mem[0x00...04]	Set 1 (001)
0			Set 0 (000)

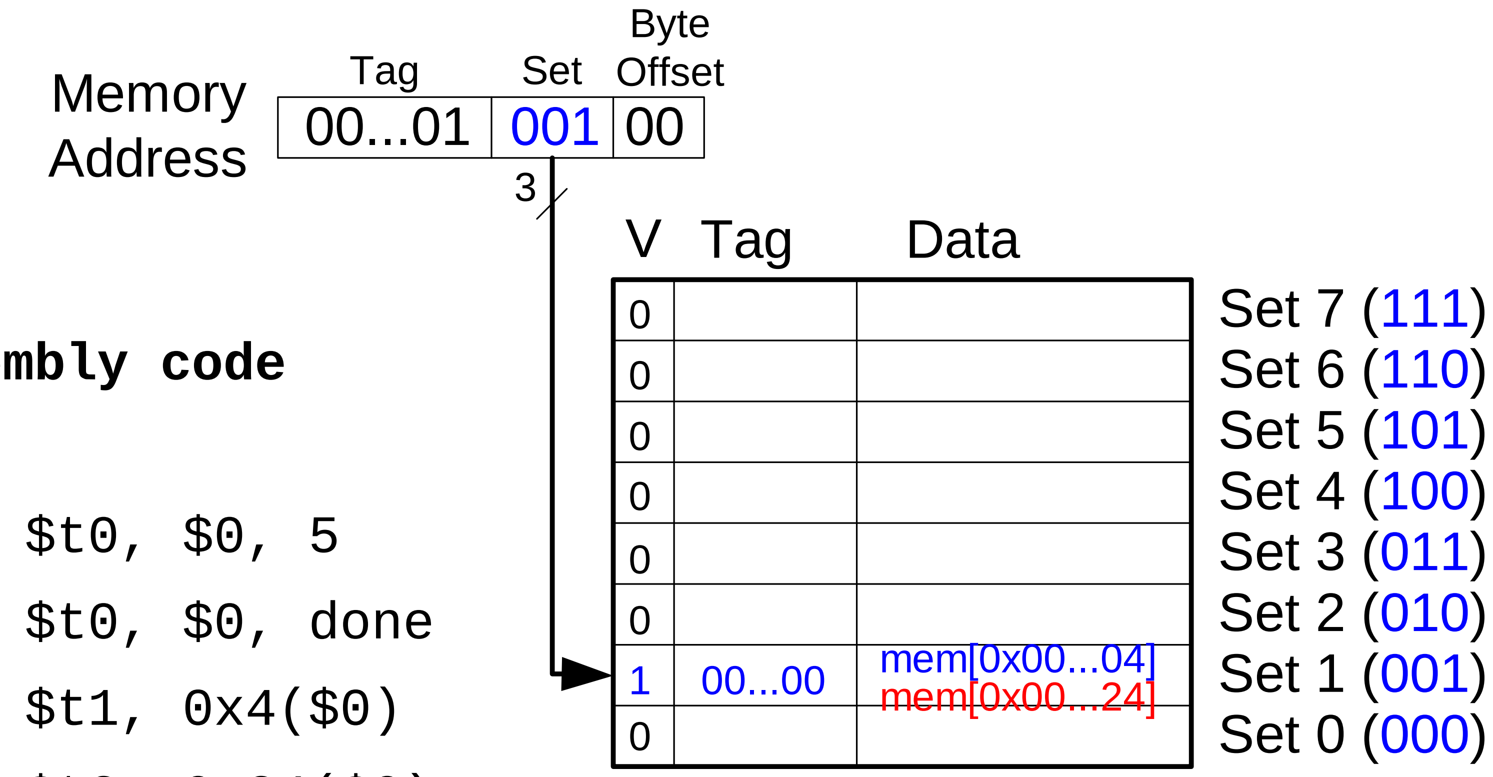
Miss Rate = 3/15

= 20%

Temporal Locality

Compulsory Misses

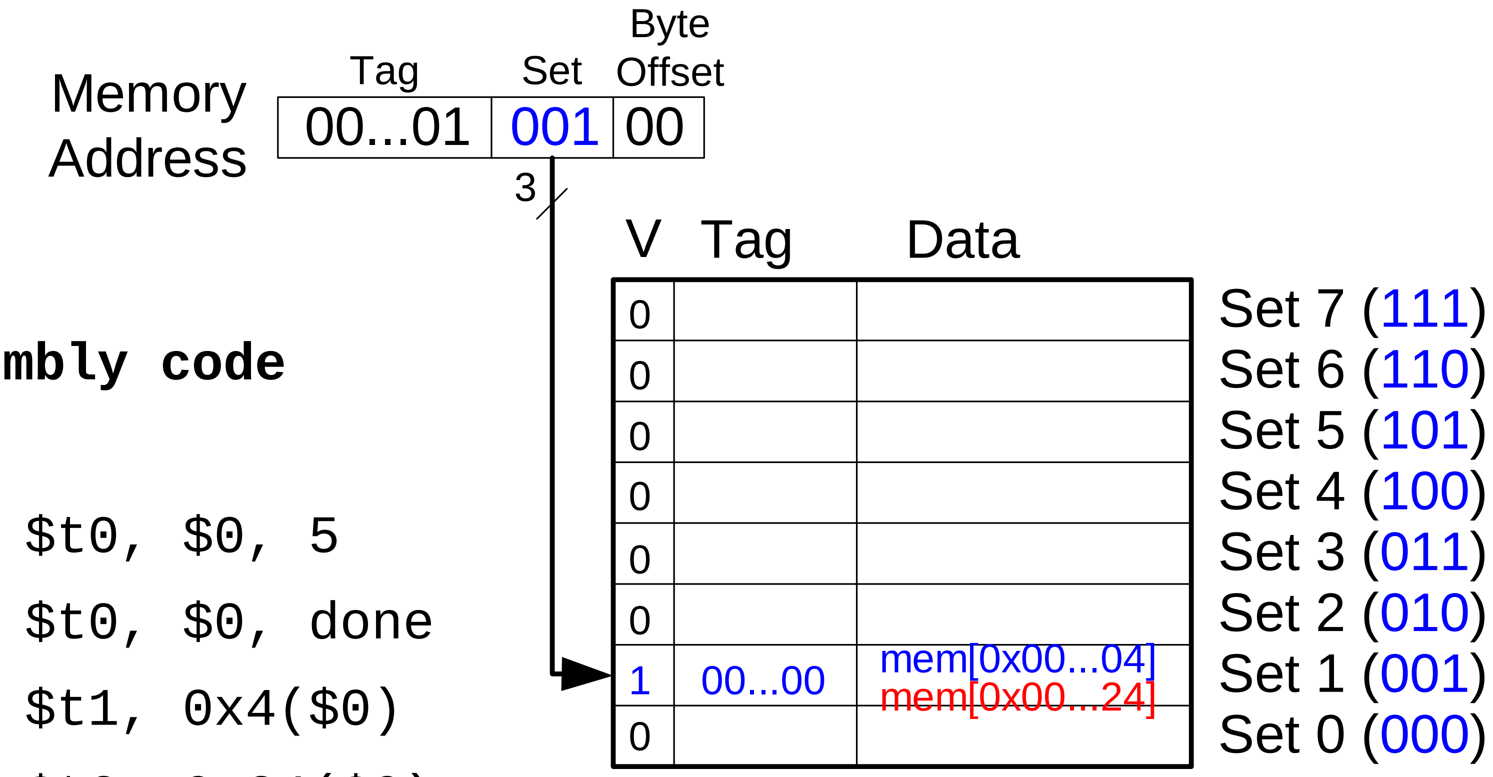
Direct Mapped Cache: Conflict



```
# 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



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)

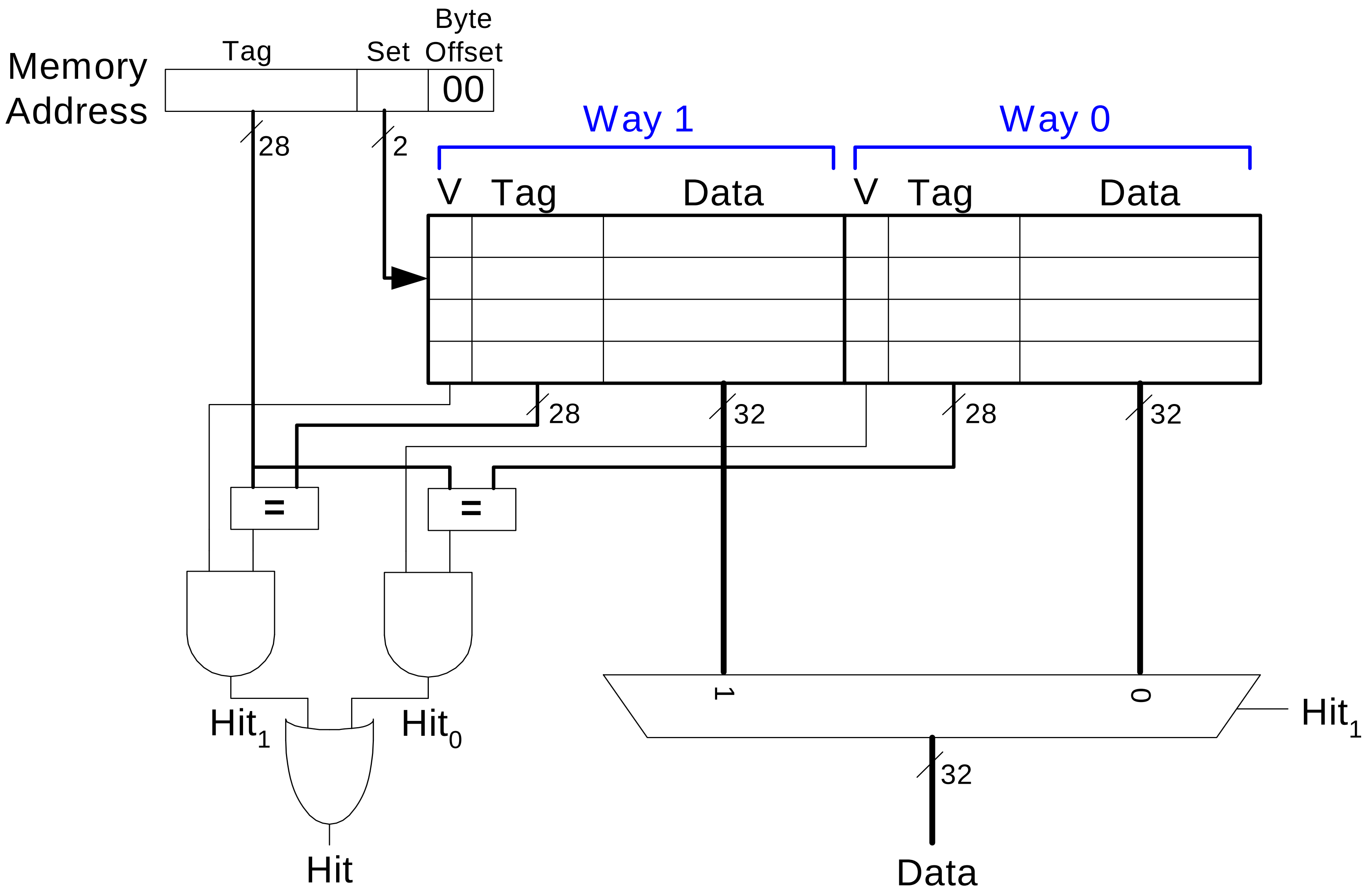
```
# MIPS assembly code

      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 = ?

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

```
# 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:
```

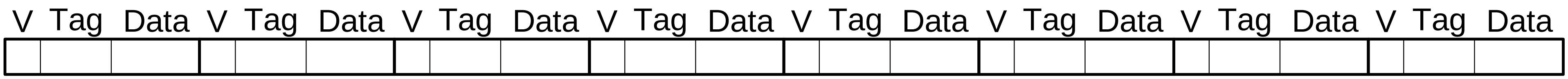
Miss Rate = 2/10

= 20%

Associativity reduces conflict misses

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



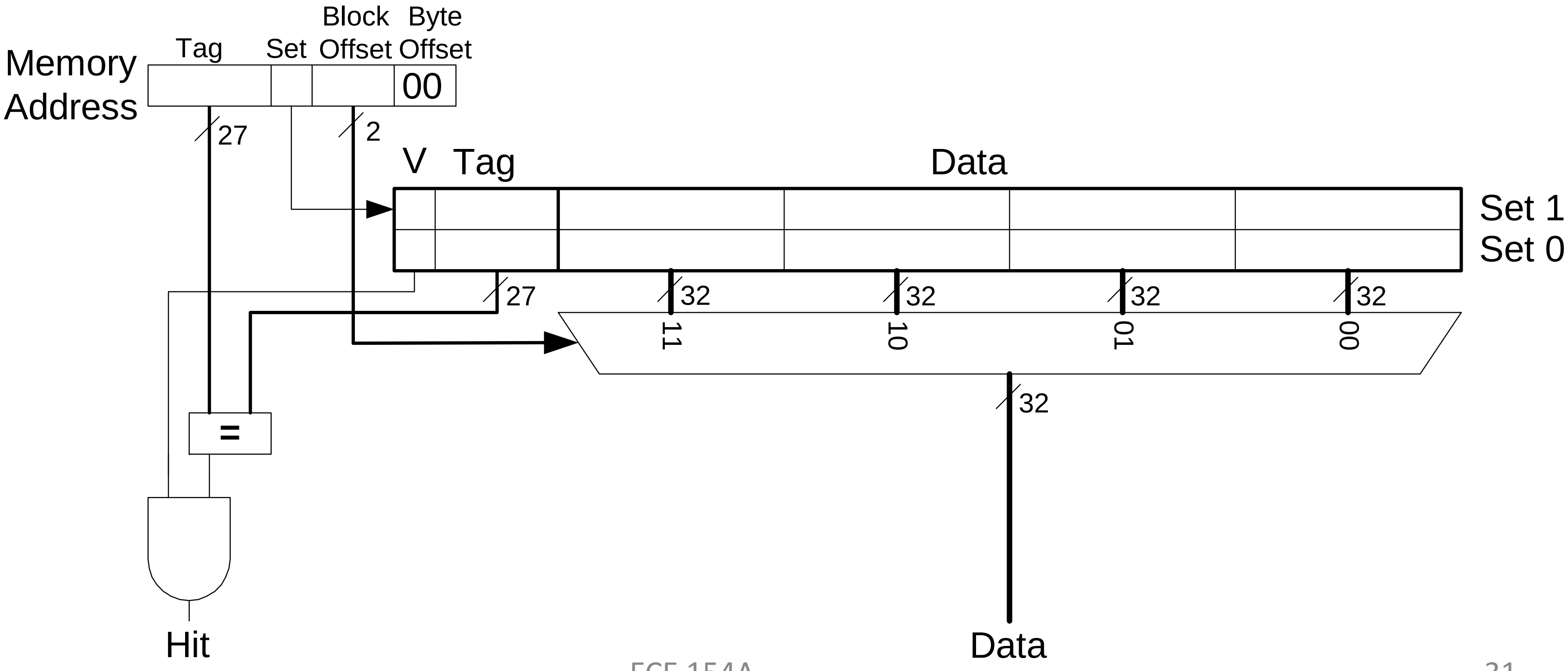
V Tag Data V Tag Data V Tag Data V Tag Data V Tag Data V Tag Data V Tag Data V Tag Data

Reduces conflict misses

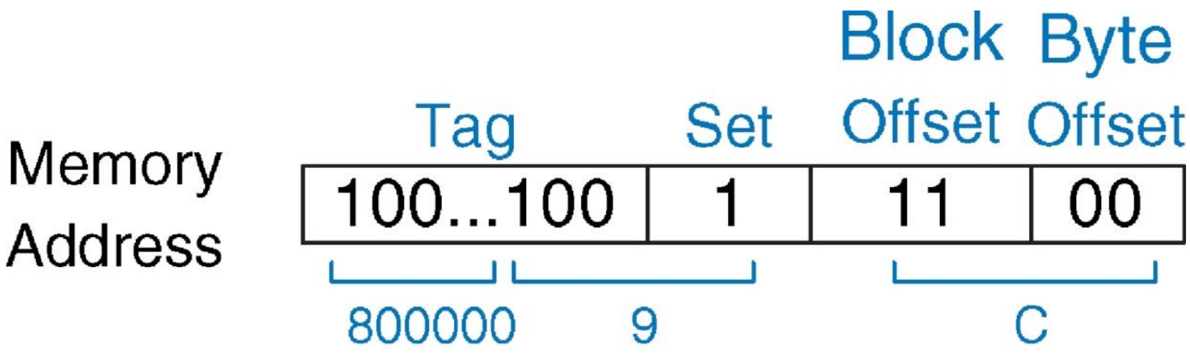
Expensive to build

Spatial Locality?

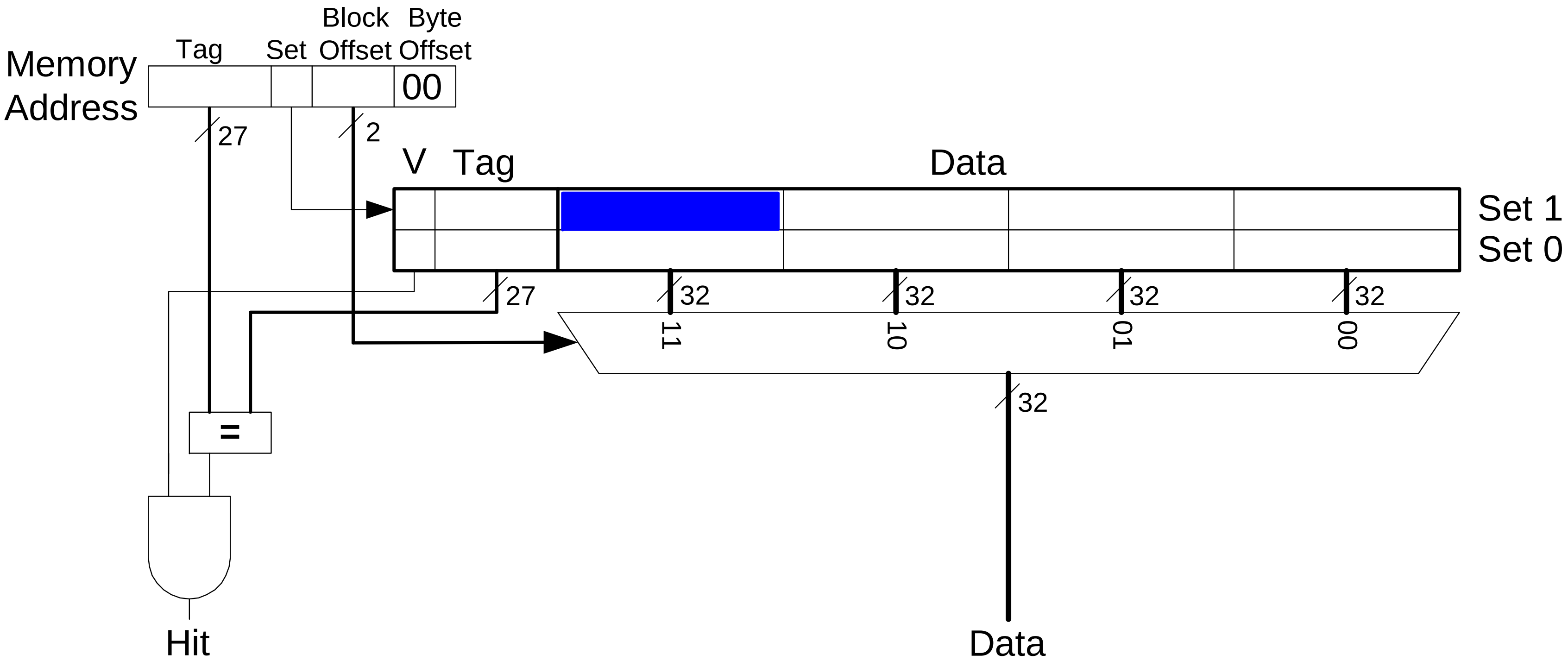
- Increase block size:
  - Block size, ***b* = 4 words**
  - *C* = 8 words
  - Direct mapped (1 block per set)
  - Number of blocks, ***B* = 2** (*C*/*b* = 8/4 = 2)



Cache with Larger Block Size



© 2007 Elsevier, Inc. All rights reserved



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 = ?

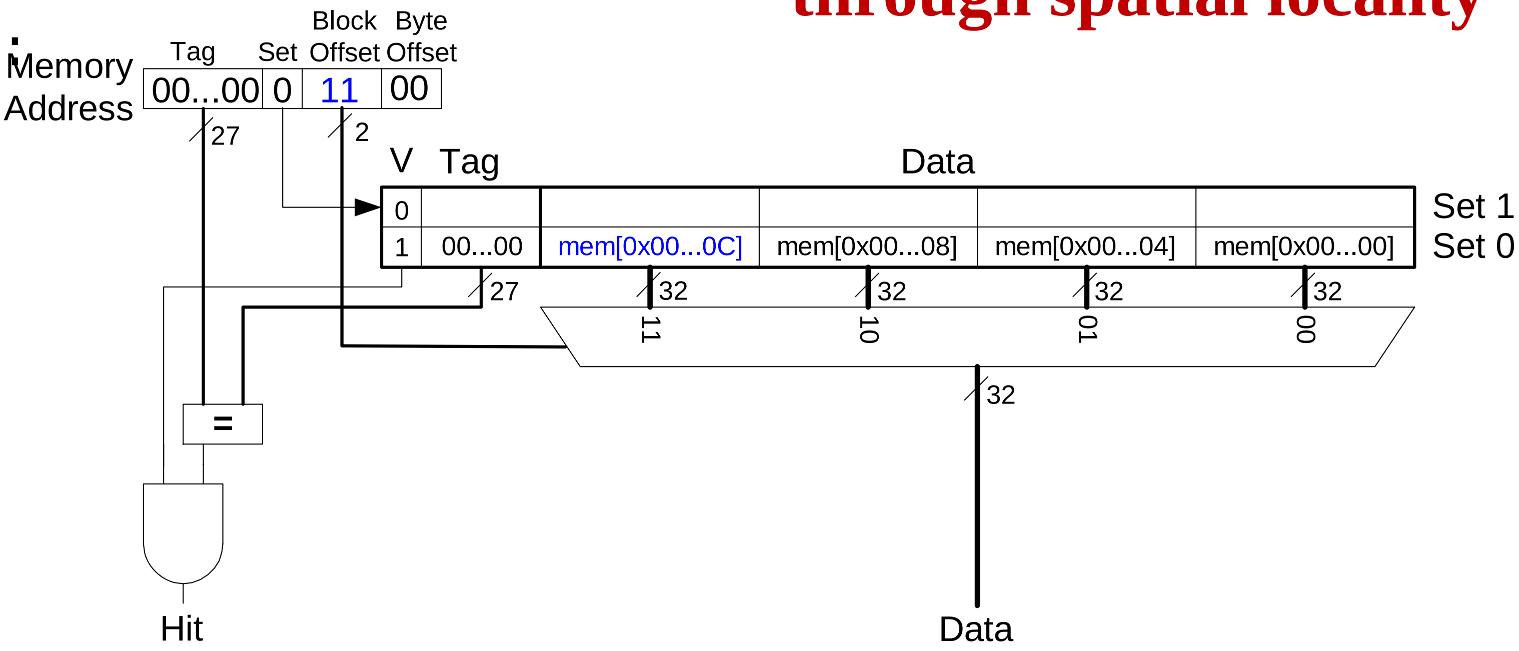
Direct Mapped Cache Performance

```
       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



V	Tag	Data				
0						Set 1
1	00...00	mem[0x00...0C]	mem[0x00...08]	mem[0x00...04]	mem[0x00...00]	Set 0

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

```
# MIPS assembly

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

		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

```
# MIPS assembly

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

		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)

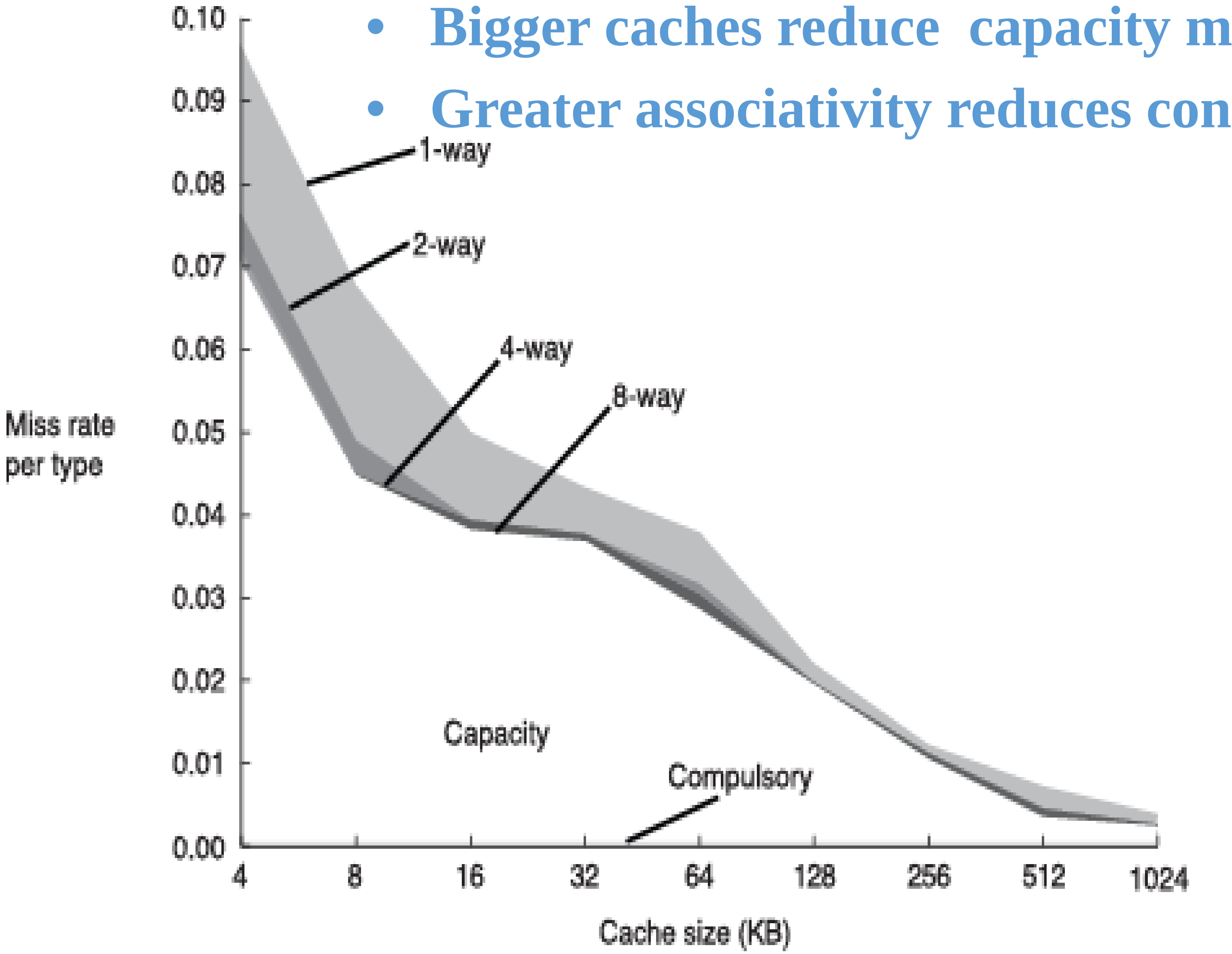
(b)

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

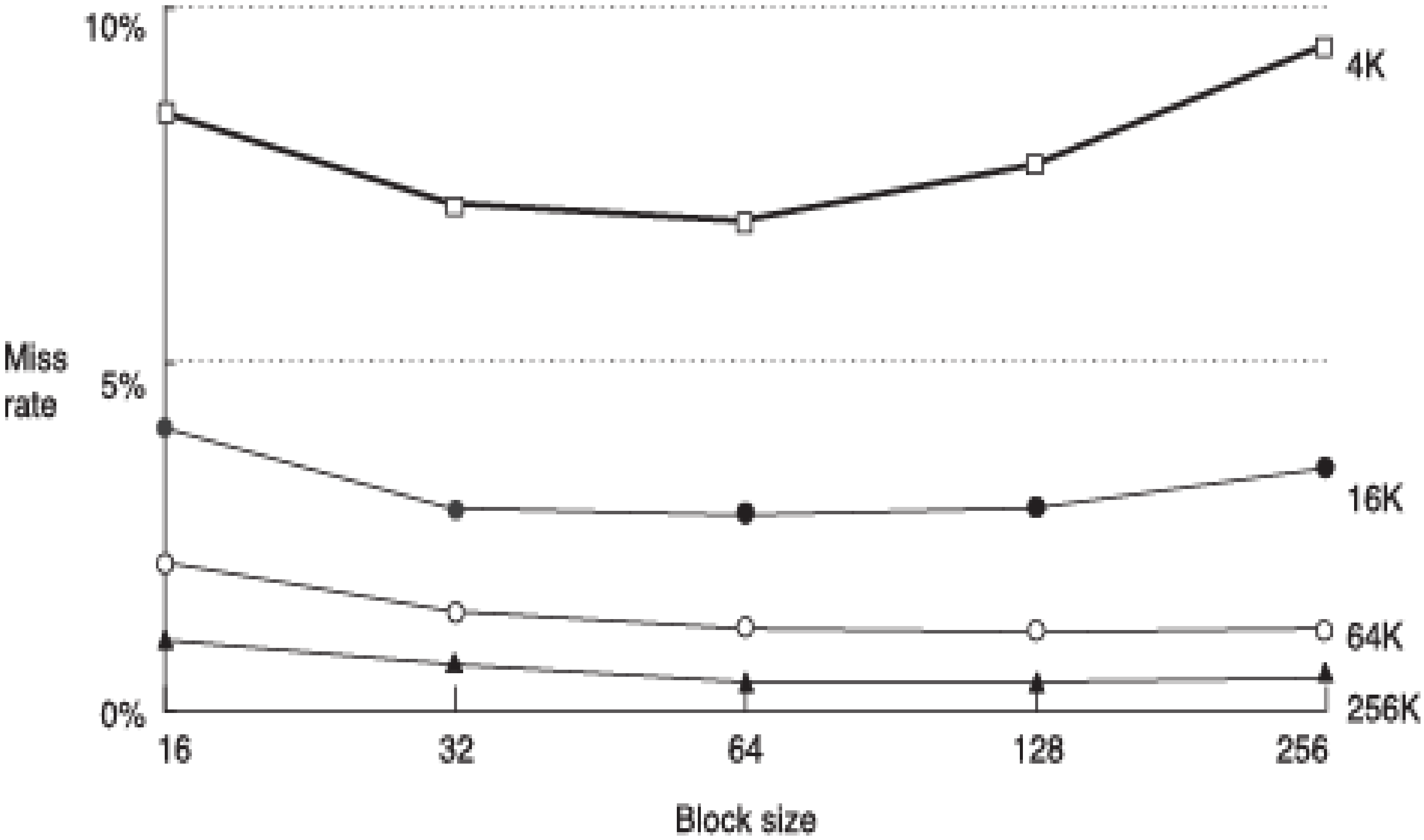
Miss Rate Trends

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



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

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

Stores

- Store are similar to loads
- On miss fetch the block into cache (and then hit, i.e. replace the corresponding word)
- On hit
  - update both cache and memory for **write-through** caches
    - Don't have to wait till request to main memory is finished (the next request to that word will be served by cache so it is okay if memory not updated yet)
  - update only cache for **write-back** cache

Write-through vs. write back

- Write-through:
  - update both cache and memory for stores
  - through away evicted block from cache when miss occurs
  - Cache has always the same data as main memory
  - simple to implement
- Write-back:
  - update cache only for stores
  - when the block is evicted from cache (whether due to store or load) main memory is updated with data in the evicted block
  - Values in cache and main memory might be different
  - Reduce traffic (bandwidth)
    - Dirty bit (reduce even further)

Intel Pentium III Die