

# Fundamentals of Computer Systems

## Caches

Stephen A. Edwards

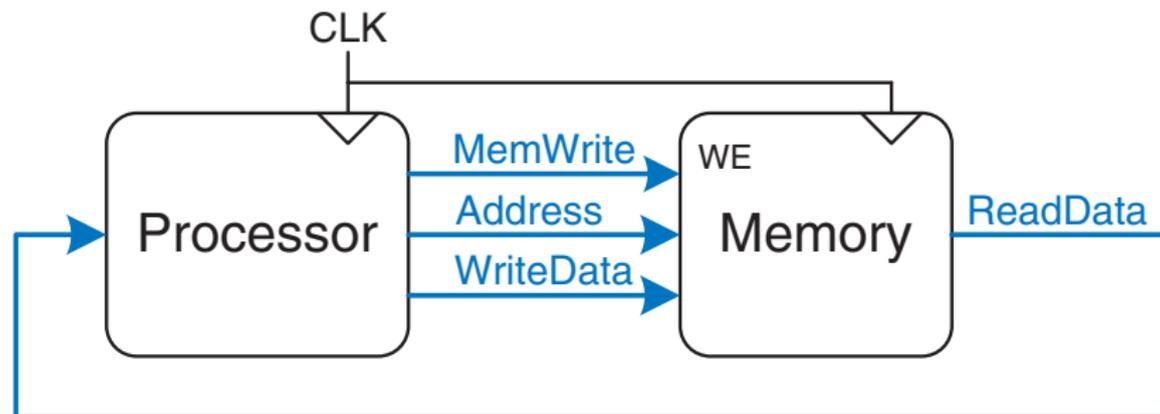
Columbia University

Summer 2015

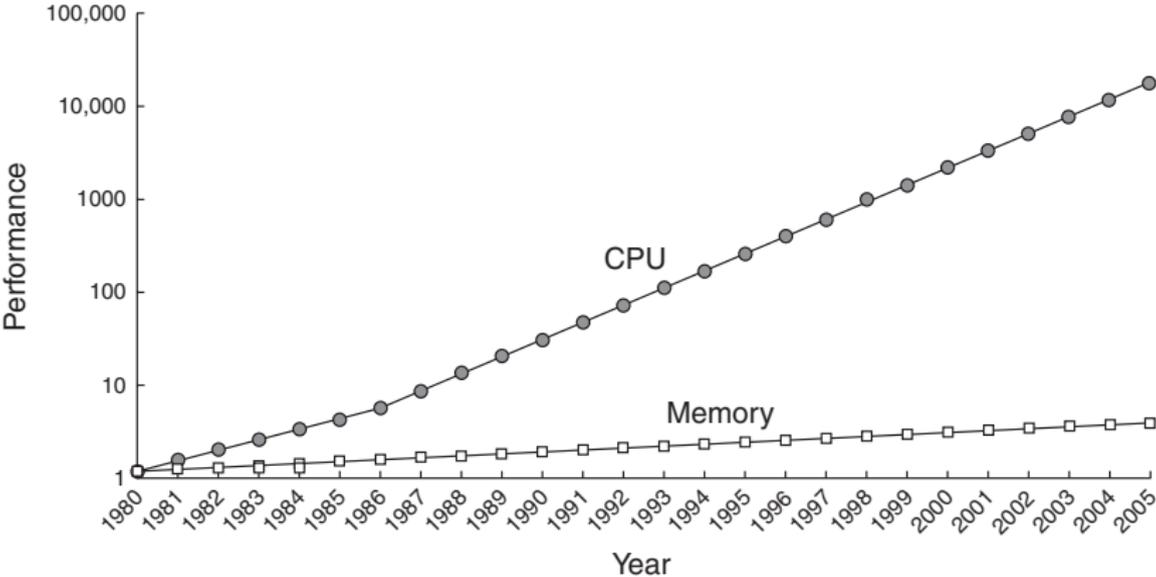
Illustrations Copyright © 2007 Elsevier

# Computer Systems

Performance depends on which is slowest:  
the processor or the memory system



# Memory Speeds Haven't Kept Up



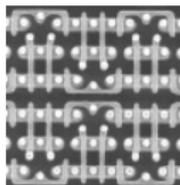
Our single-cycle memory assumption has been wrong since 1980.

Hennessy and Patterson. *Computer Architecture: A Quantitative Approach*. 3rd ed., Morgan Kaufmann, 2003.

# Your Choice of Memories

Fast    Cheap    Large

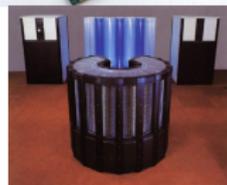
---



On-Chip SRAM



Commodity DRAM



Supercomputer



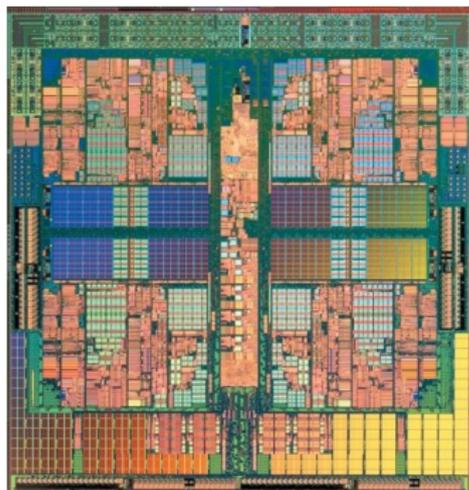
# Memory Hierarchy

An essential trick that makes big memory appear fast

<b>Technology</b>	<b>Cost (\$/Gb)</b>	<b>Access Time (ns)</b>	<b>Density (Gb/cm<sup>2</sup>)</b>
SRAM	30 000	0.5	0.00025
DRAM	10	100	1 – 16
Flash	2	300*	8 – 32
Hard Disk	0.1	10 000 000	500 – 2000

\*Read speed; writing much, much slower

# A Modern Memory Hierarchy



AMD Phenom 9600

Quad-core

2.3 GHz

1.1–1.25 V

95 W

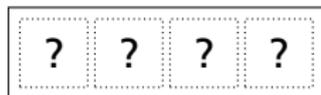
65 nm

A desktop machine:

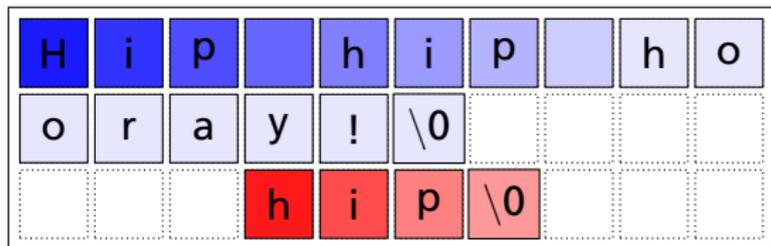
Level	Size	Tech.
L1 Instruction*	64 K	SRAM
L1 Data*	64 K	SRAM
L2*	512 K	SRAM
L3	2 MB	SRAM
Memory	4 GB	DRAM
Disk	500 GB	Magnetic

\*per core

# A Simple Memory Hierarchy



First level: small, fast storage (typically SRAM)



Last level: large, slow storage (typically DRAM)

*Can fit a subset of lower level in upper level, but which subset?*

# Locality Example: Substring Matching

H i p    h i p    h o o r a y ! \0

h i p \0

Addresses accessed over time →  
aka "reference stream"

h H h i h p h    h h i i p p h    h

*temporal locality: if needed X recently, likely to need X again soon*

*spatial locality: if need X, likely also need something near X*

# Cache

Highest levels of memory hierarchy

Fast: level 1 typically 1 cycle access time

With luck, supplies most data

Cache design questions:

What data does it hold? *Recently accessed*

How is data found? *Simple address hash*

What data is replaced? *Often the oldest*



# What Data is Held in the Cache?

Ideal cache: always correctly guesses what you want before you want it.

Real cache: never that smart

## Caches Exploit

### Temporal Locality

Copy newly accessed data into cache, replacing oldest if necessary

### Spatial Locality

Copy nearby data into the cache at the same time

Specifically, always read and write a **block**, also called a **line**, at a time (e.g., 64 bytes), never a single byte.

# Memory Performance

Hit: Data is found in the level of memory hierarchy

Miss: Data not found; will look in next level

$$\text{Hit Rate} = \frac{\text{Number of hits}}{\text{Number of accesses}}$$

$$\text{Miss Rate} = \frac{\text{Number of misses}}{\text{Number of accesses}}$$

$$\text{Hit Rate} + \text{Miss Rate} = 1$$

The expected access time  $E_L$  for a memory level  $L$  with latency  $t_L$  and miss rate  $M_L$ :

$$E_L = t_L + M_L \cdot E_{L+1}$$



## Memory Performance Example

Two-level hierarchy: Cache and main memory

Program executes 1000 loads & stores

750 of these are found in the cache

*What's the cache hit and miss rate?*

## Memory Performance Example

Two-level hierarchy: Cache and main memory

Program executes 1000 loads & stores

750 of these are found in the cache

*What's the cache hit and miss rate?*

$$\text{Hit Rate} = \frac{750}{1000} = 75\%$$

$$\text{Miss Rate} = 1 - 0.75 = 25\%$$

---

If the cache takes 1 cycle and the main memory 100,

*What's the expected access time?*

## Memory Performance Example

Two-level hierarchy: Cache and main memory

Program executes 1000 loads & stores

750 of these are found in the cache

*What's the cache hit and miss rate?*

$$\text{Hit Rate} = \frac{750}{1000} = 75\%$$

$$\text{Miss Rate} = 1 - 0.75 = 25\%$$

---

If the cache takes 1 cycle and the main memory 100,

*What's the expected access time?*

Expected access time of main memory:  $E_1 = 100$  cycles

Access time for the cache:  $t_0 = 1$  cycle

Cache miss rate:  $M_0 = 0.25$

$$E_0 = t_0 + M_0 \cdot E_1 = 1 + 0.25 \cdot 100 = 26 \text{ cycles}$$

# A Direct-Mapped Cache

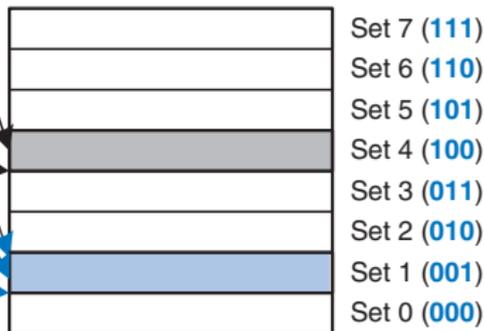
Address	Data
11...111 <b>111</b> 00	mem[0xFFFFFFFFC]
11...111 <b>110</b> 00	mem[0xFFFFFFFF8]
11...111 <b>101</b> 00	mem[0xFFFFFFFF4]
11...111 <b>100</b> 00	mem[0xFFFFFFFF0]
11...111 <b>011</b> 00	mem[0xFFFFFEC]
11...111 <b>010</b> 00	mem[0xFFFFFE8]
11...111 <b>001</b> 00	mem[0xFFFFFE4]
11...111 <b>000</b> 00	mem[0xFFFFFE0]
⋮	⋮
00...001 <b>001</b> 00	mem[0x00000024]
00...001 <b>000</b> 00	mem[0x00000020]
00...000 <b>111</b> 00	mem[0x0000001C]
00...000 <b>110</b> 00	mem[0x00000018]
00...000 <b>101</b> 00	mem[0x00000014]
00...000 <b>100</b> 00	mem[0x00000010]
00...000 <b>011</b> 00	mem[0x0000000C]
00...000 <b>010</b> 00	mem[0x00000008]
00...000 <b>001</b> 00	mem[0x00000004]
00...000 <b>000</b> 00	mem[0x00000000]

$2^{30}$ -Word Main Memory

This simple cache has

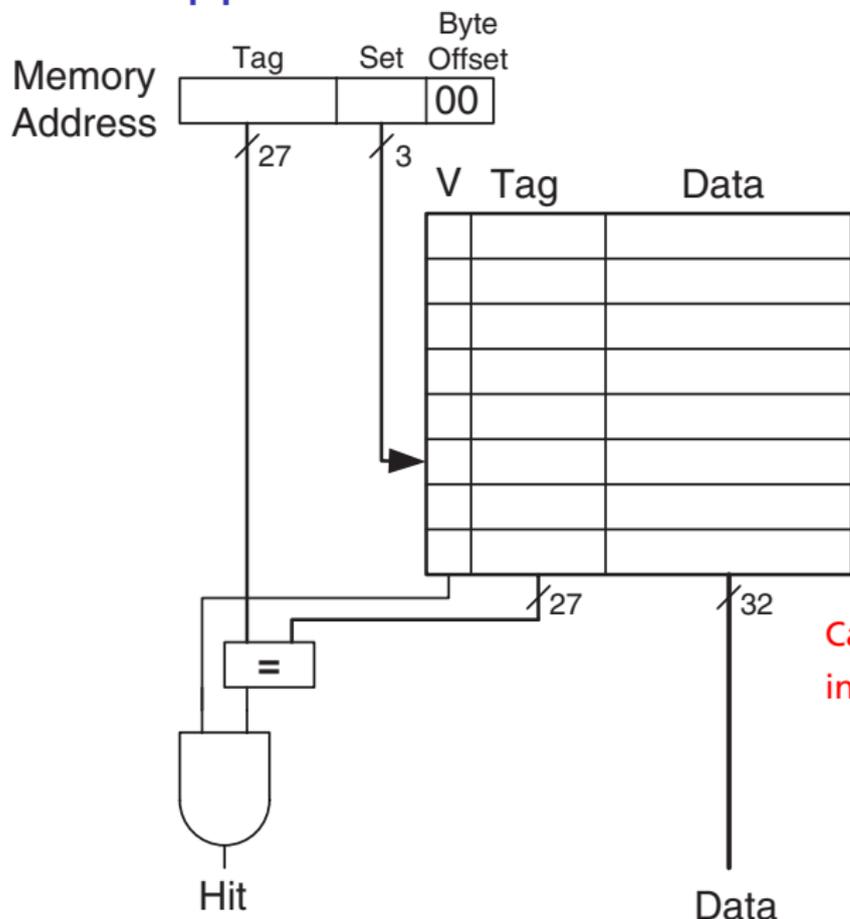
- ▶ 8 sets
- ▶ 1 *block* per set
- ▶ 4 bytes per block

To simplify answering “is this data in the cache?,” each byte is mapped to exactly one set.



$2^3$ -Word Cache

# Direct-Mapped Cache Hardware



Address bits:

0–1: byte within block

2–4: set number

5–31: block "tag"

Set 7  
Set 6  
Set 5  
Set 4  
Set 3  
Set 2  
Set 1  
Set 0

8-entry x  
(1+27+32)-bit  
SRAM

Cache hit if  
in the set of the address,

- ▶ block is valid (V=1)
- ▶ tag (address bits 5–31) matches

# Direct-Mapped Cache Behavior

A dumb loop:

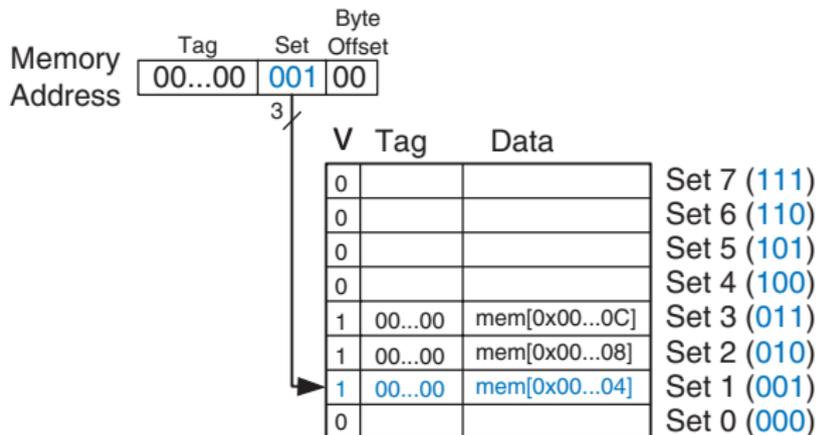
repeat 5 times

load from 0x4;

load from 0xC;

load from 0x8.

```
li    $t0, 5
l1:  beq    $t0, $0, done
     lw     $t1, 0x4($0)
     lw     $t2, 0xC($0)
     lw     $t3, 0x8($0)
     addiu  $t0, $t0, -1
     j     l1
done:
```



Assuming the cache starts empty, what's the miss rate?

# Direct-Mapped Cache Behavior

A dumb loop:

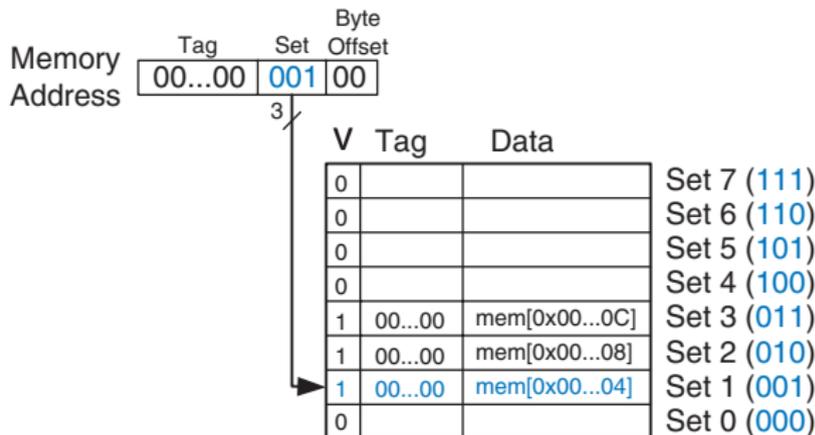
repeat 5 times

load from 0x4;

load from 0xC;

load from 0x8.

```
li    $t0, 5
l1:  beq    $t0, $0, done
     lw     $t1, 0x4($0)
     lw     $t2, 0xC($0)
     lw     $t3, 0x8($0)
     addiu  $t0, $t0, -1
     j     l1
done:
```



Assuming the cache starts empty, what's the miss rate?

4 C 8 4 C 8 4 C 8 4 C 8 4 C 8

M M M H H H H H H H H H H H H

$$3/15 = 0.2 = 20\%$$

# Direct-Mapped Cache: Conflict

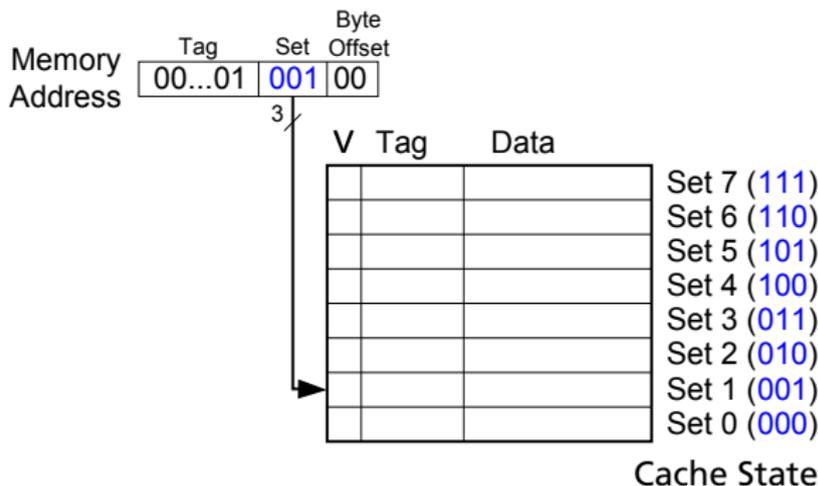
A dumber loop:

repeat 5 times

load from 0x4;

load from 0x24

```
li    $t0, 5
l1:  beq    $t0, $0, done
     lw     $t1, 0x4($0)
     lw     $t2, 0x24($0)
     addiu  $t0, $t0, -1
     j     l1
done:
```



Assuming the cache starts empty, what's the miss rate?

These are *conflict misses*

# Direct-Mapped Cache: Conflict

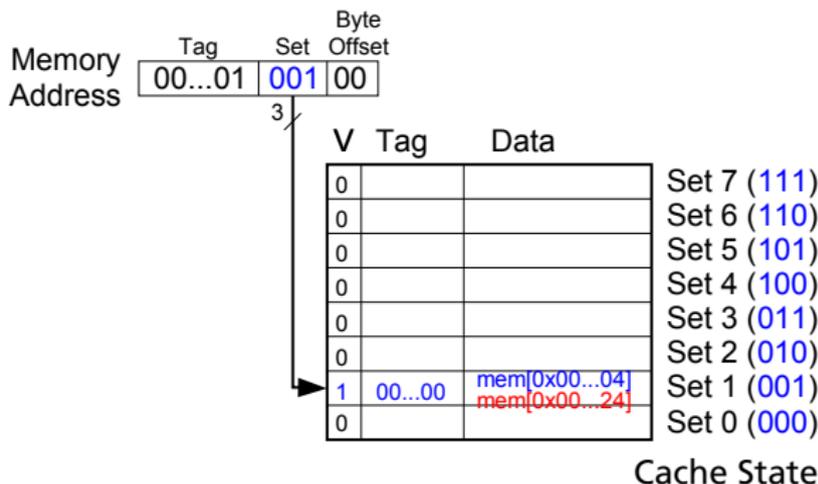
A dumber loop:

repeat 5 times

load from 0x4;

load from 0x24

```
li    $t0, 5
l1:  beq    $t0, $0, done
      lw     $t1, 0x4($0)
      lw     $t2, 0x24($0)
      addiu $t0, $t0, -1
      j     l1
done:
```



Assuming the cache starts empty, what's the miss rate?

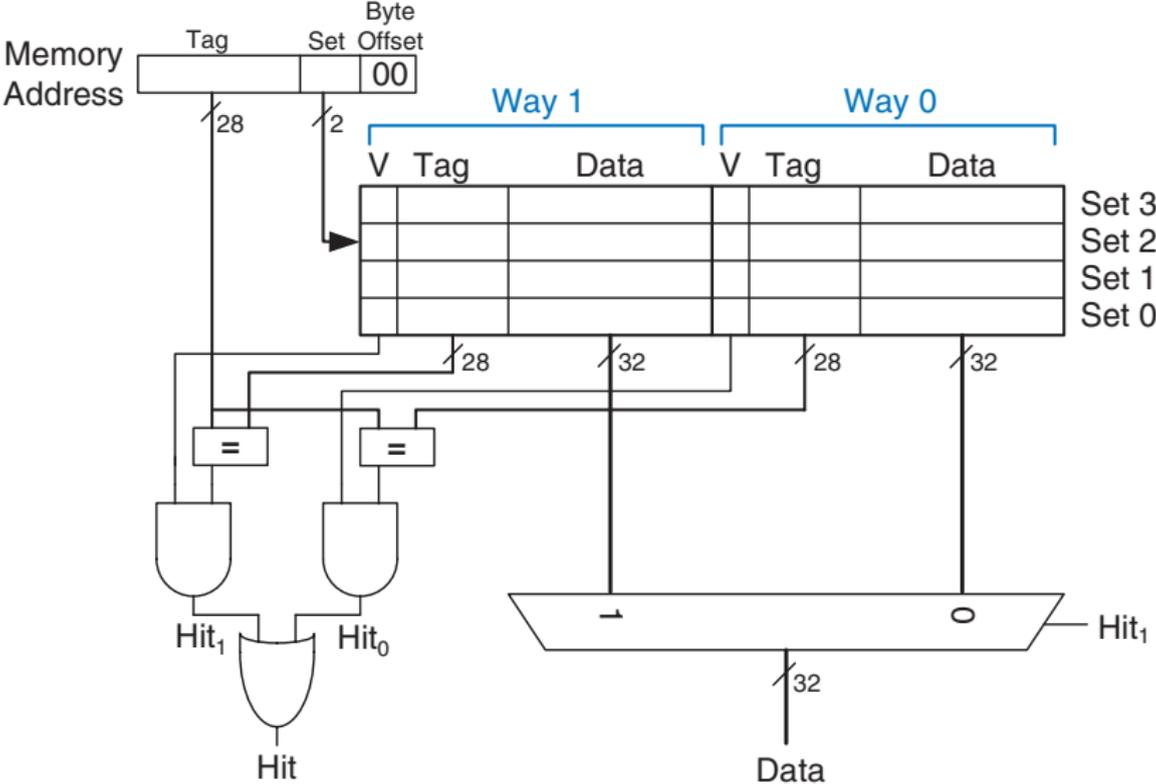
4 24 4 24 4 24 4 24 4 24

M M M M M M M M M M

10/10 = 1 = 100% Oops

These are *conflict misses*

# No Way! Yes Way! 2-Way Set Associative Cache



## 2-Way Set Associative Behavior

```
li    $t0, 5
l1:  beq  $t0, $0, done
      lw   $t1, 0x4($0)
      lw   $t2, 0x24($0)
      addiu $t0, $t0, -1
      j    l1
done:
```

Assuming the cache starts empty, what's the miss rate?

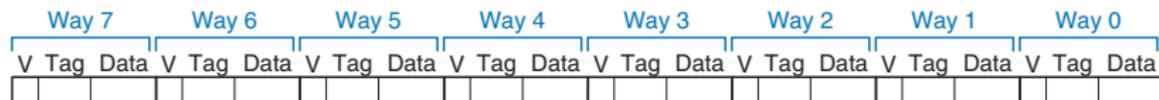
4 24 4 24 4 24 4 24 4 24  
M M H H H H H H H H

$$2/10 = 0.2 = 20\%$$

*Associativity reduces conflict misses*

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

# An Eight-way Fully Associative Cache



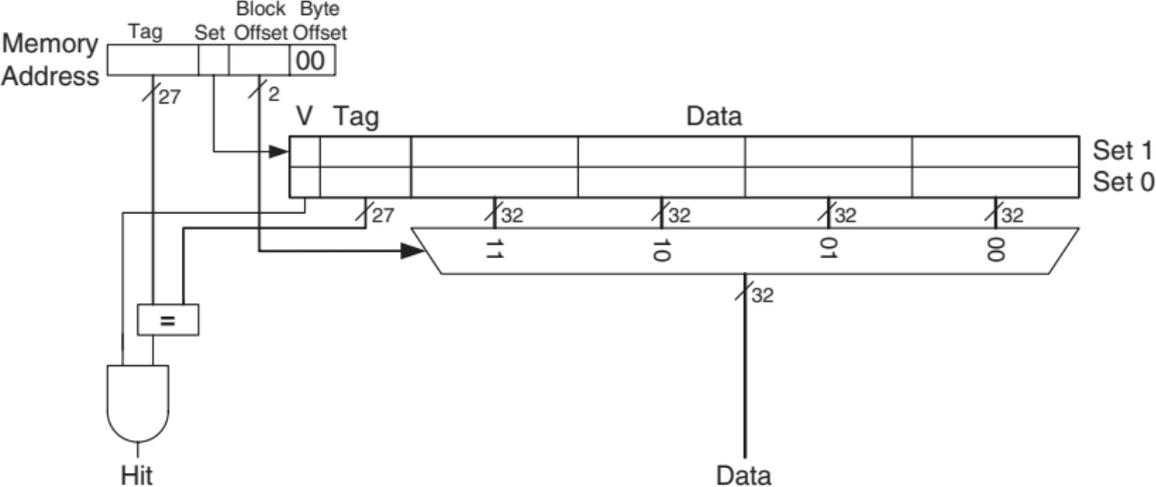
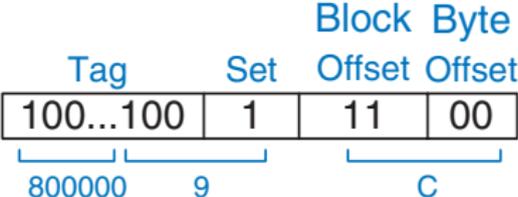
No conflict misses: only compulsory or capacity misses

Either very expensive or slow because of all the associativity

# Exploiting Spatial Locality: Larger Blocks

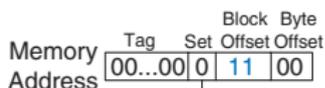
0x8000 0009C:

Memory  
Address



- 2 sets
- 1 block per set (Direct Mapped)
- 4 words per block

# Direct-Mapped Cache Behavior w/ 4-word block



Cache when reading 0xC

The dumb loop:

repeat 5 times

load from 0x4;

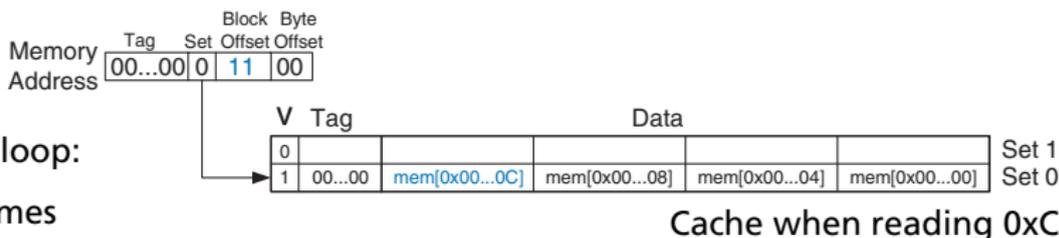
load from 0xC;

load from 0x8.

```
li    $t0, 5
l1:  beq    $t0, $0, done
     lw     $t1, 0x4($0)
     lw     $t2, 0xC($0)
     lw     $t3, 0x8($0)
     addiu  $t0, $t0, -1
     j     l1
done:
```

Assuming the cache starts empty, what's the miss rate?

# Direct-Mapped Cache Behavior w/ 4-word block



The dumb loop:

repeat 5 times

load from 0x4;

load from 0xC;

load from 0x8.

```

li    $t0, 5
l1:  beq    $t0, $0, done
      lw    $t1, 0x4($0)
      lw    $t2, 0xC($0)
      lw    $t3, 0x8($0)
      addiu $t0, $t0, -1
      j    l1
done:
    
```

Assuming the cache starts empty, what's the miss rate?

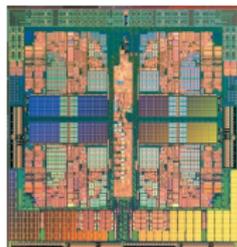
4 C 8 4 C 8 4 C 8 4 C 8 4 C 8

M H H H H H H H H H H H H H H H

$1/15 = 0.0666 = 6.7\%$

*Larger blocks reduce compulsory misses by exploiting spatial locality*

# Stephen's Desktop Machine Revisited



AMD Phenom  
9600  
Quad-core  
2.3 GHz  
1.1–1.25 V  
95 W  
65 nm

On-chip caches:

Cache	Size	Sets	Ways	Block
L1I*	64 K	512	2-way	64-byte
L1D*	64 K	512	2-way	64-byte
L2*	512 K	512	16-way	64-byte
L3	2 MB	1024	32-way	64-byte

\* per core

# Intel On-Chip Caches

Chip	Year	Freq. (MHz)	L1		L2	
			Data	Instr		
	80386	1985	16–25	off-chip		none
	80486	1989	25–100	8K unified		off-chip
	Pentium	1993	60–300	8K	8K	off-chip
	Pentium Pro	1995	150–200	8K	8K	256K–1M (MCM)
	Pentium II	1997	233–450	16K	16K	256K–512K (Cartridge)
	Pentium III	1999	450–1400	16K	16K	256K–512K
	Pentium 4	2001	1400–3730	8–16K	12k op trace cache	256K–2M
	Pentium M	2003	900–2130	32K	32K	1M–2M
	Core 2 Duo	2005	1500–3000	32K per core	32K per core	2M–6M