CS 354 - Machine Organization & Programming
Tuesday Oct 22, and Thursday Oct 24, F24

Midterm Exam 2 - Thursday November 7th, 7:30 - 9:30 pm

- **UW ID required**
- **#2 pencils required**
- **closed book, no notes, no electronic devices (e.g., calculators, phones, watches) see "Midterm Exam 2" on course site Assignments for topics**

Project p3B: DUE on or before Friday, Oct 25

Homework hw4: DUE on or before Monday Nov 5th,

Project p4A: DUE on or before Nov 1st,

Project p4AQuestions: DUE on or before Thurs Nov 8th,

Project p4B: DUE on or before Nov 10th,

Learning Objectives

- determine hit or miss given address and cache contents
- determine set number (index) from s-bits
- determine if an address is within a given cache block
- explain the effect of cache configuration on given sequence of address (working set)
- explain difference btw direct mapped, fully associative, and set associative caches
- implement Least Frequently Used replacement policies
- implement Least Recently Used replacement policy
- explain diff btw write-through, write-back, no-write allocate, write allocate caches
- compare cache performance of different cache configurations for working set sequence
- describe the impact of stride and the scales of the memory mountain

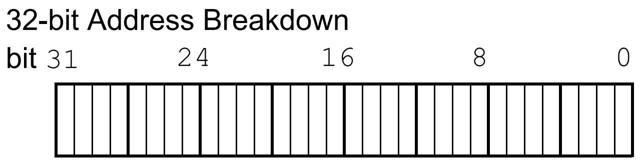
This Week

Finish L14 (bring W7 outline) Direct Mapped Caches - Restrictive Fully Associative Caches - Unrestrictive Set Associative Caches - Sweet! Replacement Policies	Writing to Caches Cache Performance Impact of Stride Memory Mountain C, Assembly, and Mach Code
Next Week: Assembly Language Instr. B&O Chapter 3 Intro 3.1 A Historical Perspective 3.2 Program Encodings 3.3 Data Formats	3.4 Accessing Information 3.5 Arithmetic and Logical Control 3.6 Control

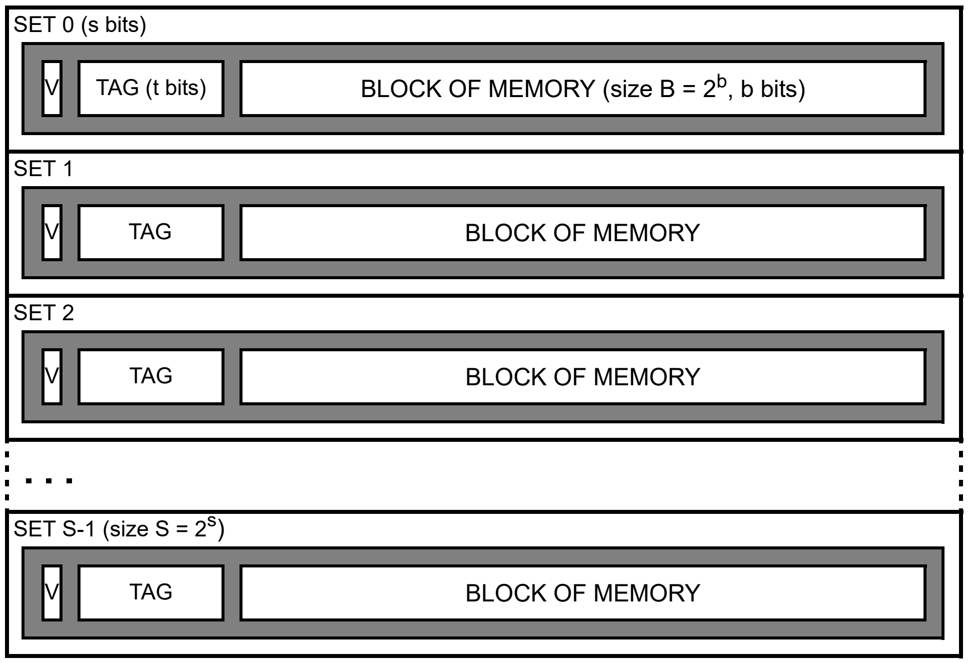
Direct Mapped Caches - Restrictive

Direct Mapped Cache is a cache

→ What is the address breakdown if blocks are 32 bytes and there are 1024 sets?



→ Is the cache operation fast O(1) or slow O(S) where S is the number of sets?



→ What happens when two different memory blocks map to the same set?

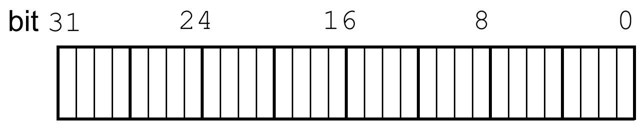
✵ *Appropriate for*

Fully Associative Caches - Unrestrictive

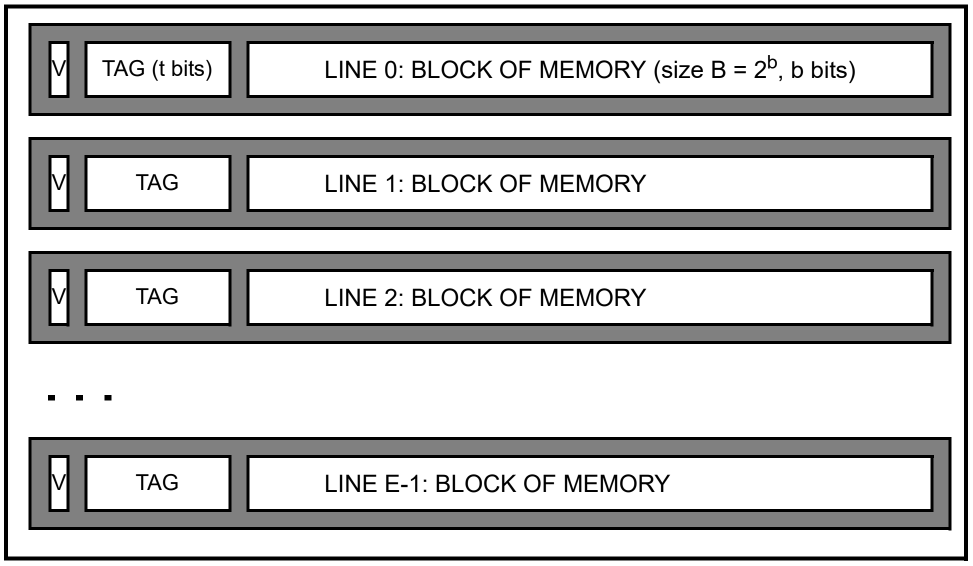
Fully Associative Cache is a cache

→ What is the address breakdown if blocks are 32 bytes and there are 1024 sets?

32-bit Address Breakdown



→ Is the cache operation fast O(1) or slow O(E) where E is the number of lines?



→ What happens when two different memory blocks map to the same set?

→ How many lines should a fully associative cache have?

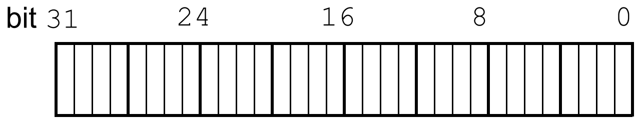
✵ *Appropriate for*

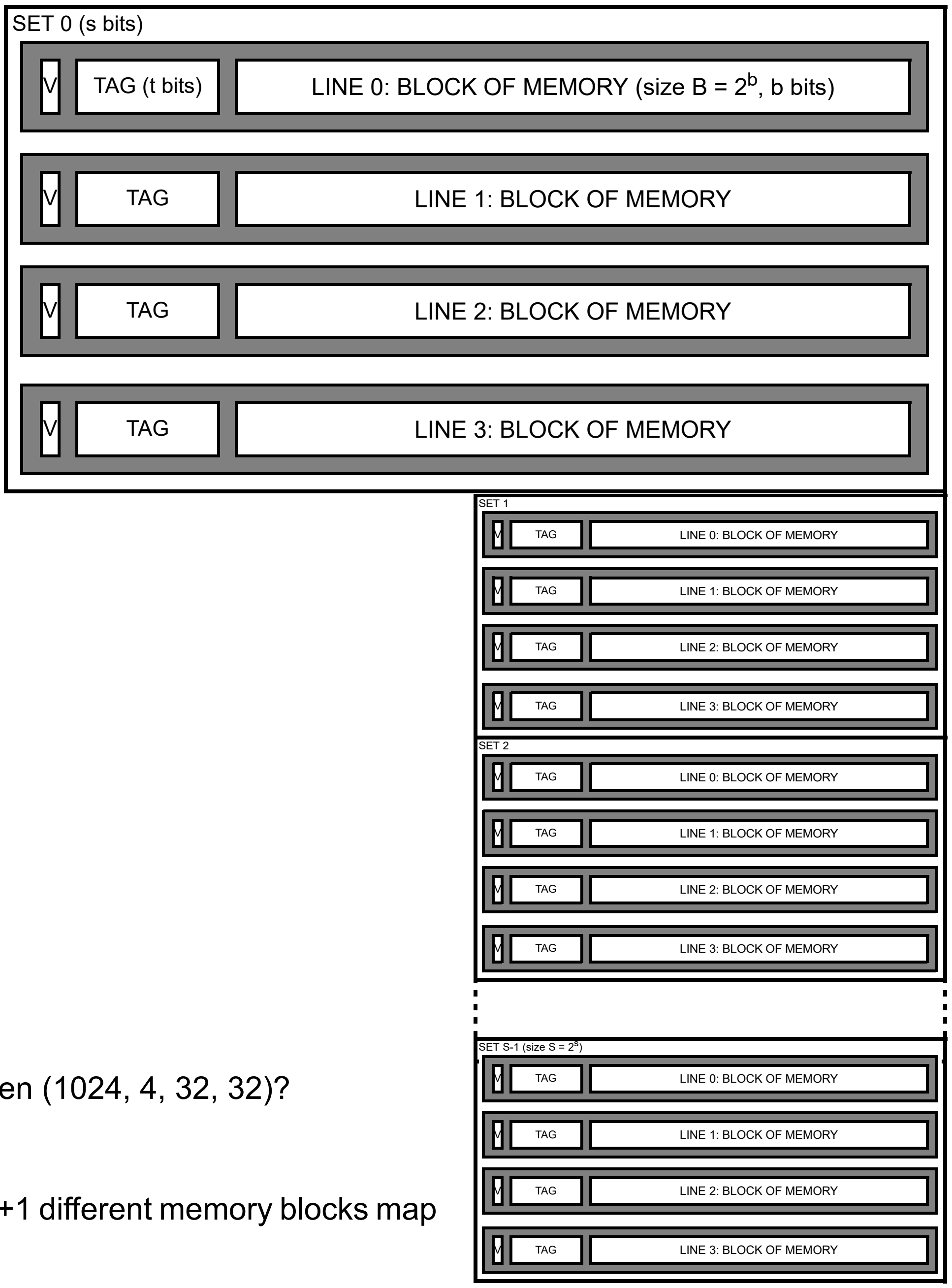
Set Associative Caches - Sweet!

Set Associative Cache is a cache commonly used today

→ What is the address breakdown if blocks are 32 bytes and there are 1024 sets?

32-bit Address Breakdown





Let E be

E = 4 is

E = 1 is

✵ ***C = (S, E, B, m)***

Let C be

→ How big is a cache given (1024, 4, 32, 32)?

→ What happens when E+1 different memory blocks map to the same set?

Replacement Policies

Assume the following sequence of memory blocks

are fetched into <u>the same set</u> of a 4-way associative cache that is initially empty:
`b1, b2, b3, b1, b3, b4, b4, b7, b1, b8, b4, b9, b1, b9, b9, b2, b8, b1`

1. *Random Replacement*

→ Which of the following four outcomes is possible after the sequence finishes?
Assume the initial placement is random.

```
   L0 L1 L2 L3
1. b9 b1 b8 b2
2. b1 b2 -- b8
3. b1 b4 b7 b3
4. b1 b2 b8 b1
```

2. *Least Recently Used* (LRU)

→ What is the outcome after the sequence finishes?
Assume the initial placement is in ascending line order (left to right below).

L0 L1 L2 L3

3. *Least Frequently Used* (LFU)

→ Which blocks will remain in the cache after the sequence finishes?

❅ *Exploiting replacement policies*

Writing to a Cache

- ✵ *Reading data copies a block of memory into the cache levels*
- ✵ *Writing data requires that these copies must be kept consistent*

Write Hits

occur when writing to a block **that IS in this cache**

→ When should a block be updated in lower memory levels?

1. ***Write Through***:

2. ***Write Back***:

Write Misses

occur when writing to a block **that IS NOT in this cache**

→ Should space be allocated in this cache for the block being changed?

1. ***No Write Allocate***:

2. ***Write Allocate***:

Typical Designs

1. **Write Through** paired with **No Write Allocate**
2. **Write Back paired** with **Write Allocate**

→ Which best exploits locality?

Cache Performance

Metrics

hit rate

hit time

miss penalty,

Larger B̲locks (S and E unchanged)

hit rate

hit time

miss penalty

THEREFORE

More S̲ets (B and E unchanged)

hit rate

hit time

miss penalty

THEREFORE

More Lines E̲ per Set (B and S unchanged)

hit rate

hit time

miss penalty

THEREFORE

Intel Quad Core i7 Cache (gen 7)

all: 64 byte blocks, use pseudo LRU, write back

L1: 32KB, 4-way Instruction & 32KB 8-way Data, no write allocate
L2: 256KB, 8-way, write allocate
L3: 8MB, 16-way (2MB/Core shared), write allocate

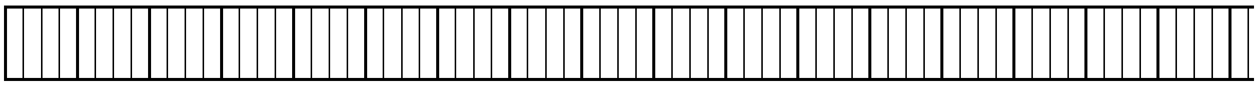
Impact of Stride

Stride Misses

Example:

```
int initArray(int a[][8], int rows) {
   for (int i = 0; i < rows; i++)
      for(int j = 0; j < 8; j++)
         a[i][j] = i * j;
}
```

→ Draw a diagram of the memory layout of the first two rows of `a`:



recall C uses row-major order

Assume: `a` is aligned with cache blocks
is too big to fit entirely into the cache
words are 4 bytes, block size is 16 bytes
direct-mapped cache is initially empty, write allocate used

→ Indicate the order elements are accessed in the table below and mark H for hit or M for miss:

a[i][j]	j = 0	1	2	3	4	5	6	7
i = 0								
1								
...								

→ Now exchange the `i` and `j` loops mark the table again:

a[i][j]	j = 0	1	2	3	4	5	6	7
i = 0								
1								
...								

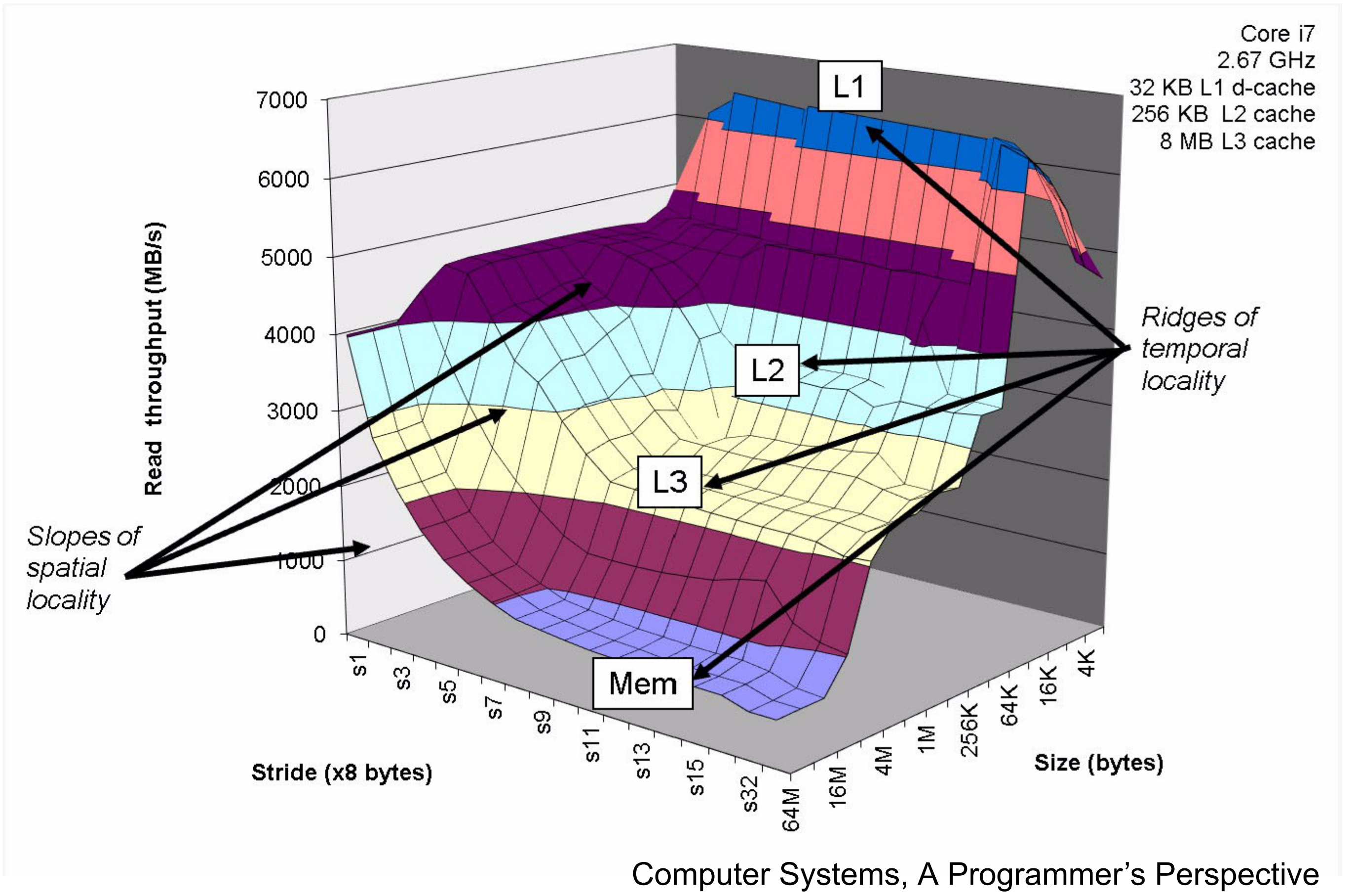
Memory Mountain

Independent Variables

stride - 1 to 16 double words step size used to scan through array
size - 2K to 64 MB arraysize

Dependent Variable

read throughput - 0 to 7000 MB/s



Computer Systems, A Programmer's Perspective
Second Edition, Bryant and O'Hallaron

Temporal Locality Impacts

factor of ~10 between L1 (6000MB/s) and MM (600MB/s)

Spatial Locality Impacts

factor of ~7 from top (4000MB/s) to bottom stride slope (600MB/s)

✵ *Memory access speed is not characterized by a single value. It's a landscape that can be exploited through the use of spatial and temporal locality.*

C, Assembly, & Machine Code

C Function / **Assembly (AT&T)** / **Machine (hex)**

```
int accum = 0;
int sum(int x, int y)     sum:
{                            pushl %ebp                55
                             movl %esp, %ebp           89 e5
                             movl 12(%ebp), %eax       8b 45 0C
   int t = x + y;            addl 8(%ebp), %eax        03 45 08
   accum += t;               addl %eax, accum          01 05 ?? ?? ?? ??
   return t;                 popl %ebp                 5D
}                            ret                       C3
```

C

- **is HLL (high level language) that enable us to be more productive coders**
- **helps us write correct code with syntax and type checking**
- **can be compiled and run on different architectures (portable)**

→ What aspects of the machine does C hide from us?
low-level machine details

Assembly (ASM)

- **is human readable representation of MC**
- **is very machine dependent**

→ What ISA (Instruction Set Architecture) are we studying?

→ What does assembly remove from C source? **HLL constructs**

→ Why Learn Assembly?
1. **better understand the stack**
2. **identify code inefficiencies and vulnerabilities**
3. **understand compiler optimization options**

Machine Code (MC) **is**

- **elementary cpu instructions and data in binary (typically generated by assembler)**
- **the unique encodings that a particular machine understands and can execute**

→ How many bytes long is an IA-32 instructions? **1 - 15 bytes**