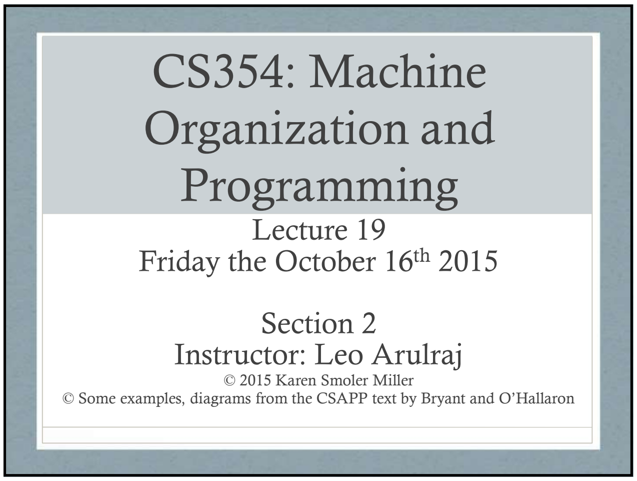
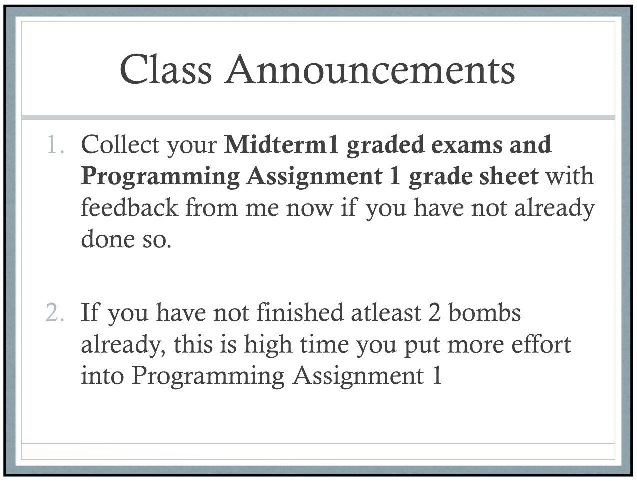


CS354: Machine Organization and Programming

Lecture 19
Friday the October 16th 2015

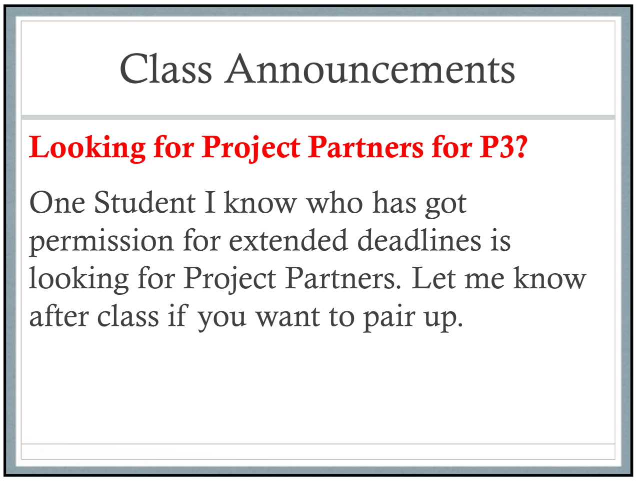
Section 2
Instructor: Leo Arulraj



Class Announcements

1. Collect your **Midterm1 graded exams and Programming Assignment 1 grade sheet** with feedback from me now if you have not already done so.

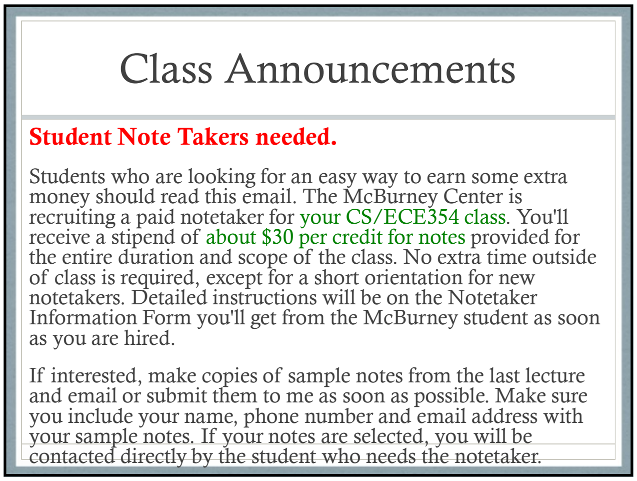
2. If you have not finished atleast 2 bombs already, this is high time you put more effort into Programming Assignment 1



Class Announcements

Looking for Project Partners for P3?

One Student I know who has got permission for extended deadlines is looking for Project Partners. Let me know after class if you want to pair up.

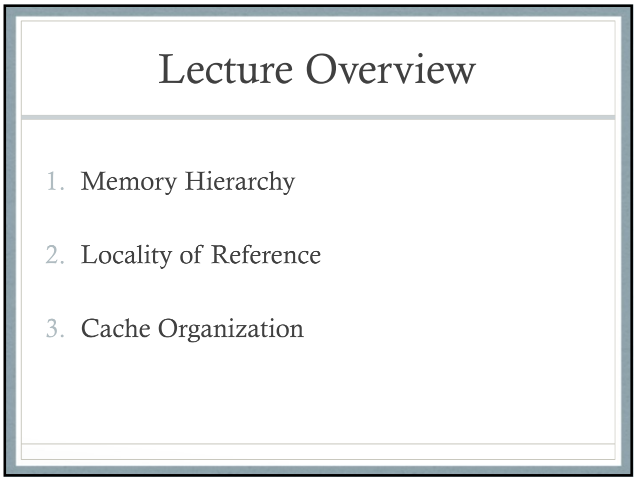


Class Announcements

Student Note Takers needed.

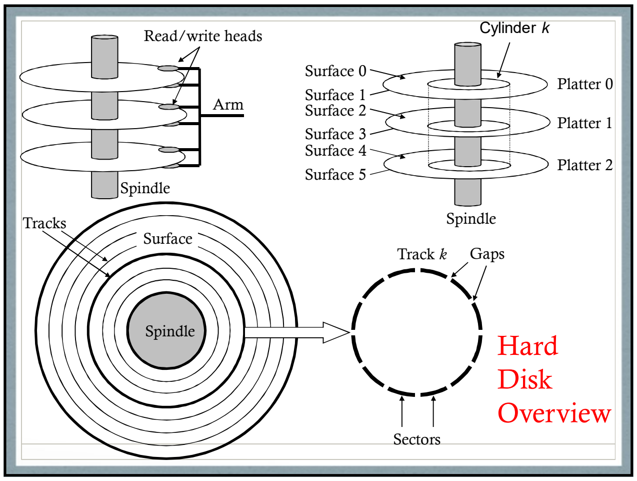
Students who are looking for an easy way to earn some extra money should read this email. The McBurney Center is recruiting a paid notetaker for your CS/ECE354 class. You'll receive a stipend of about $30 per credit for notes provided for the entire duration and scope of the class. No extra time outside of class is required, except for a short orientation for new notetakers. Detailed instructions will be on the Notetaker Information Form you'll get from the McBurney student as soon as you are hired.

If interested, make copies of sample notes from the last lecture and email or submit them to me as soon as possible. Make sure you include your name, phone number and email address with your sample notes. If your notes are selected, you will be contacted directly by the student who needs the notetaker.

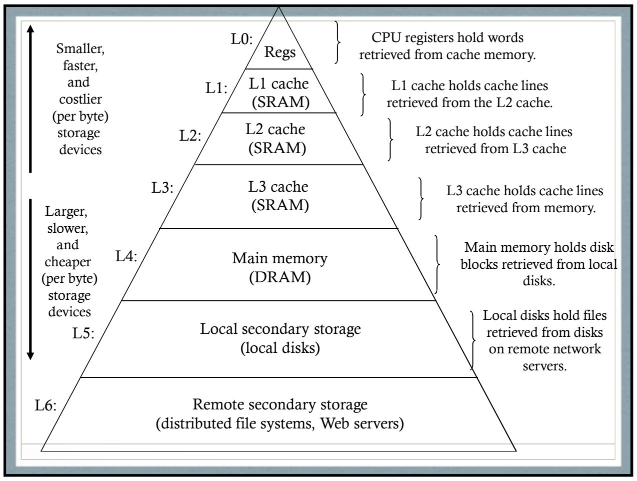


Lecture Overview

1. Memory Hierarchy
2. Locality of Reference
3. Cache Organization



Hard Disk Overview



Smaller, faster, and costlier (per byte) storage devices

Larger, slower, and cheaper (per byte) storage devices

L0: Regs — CPU registers hold words retrieved from cache memory.

L1: L1 cache (SRAM) — L1 cache holds cache lines retrieved from the L2 cache.

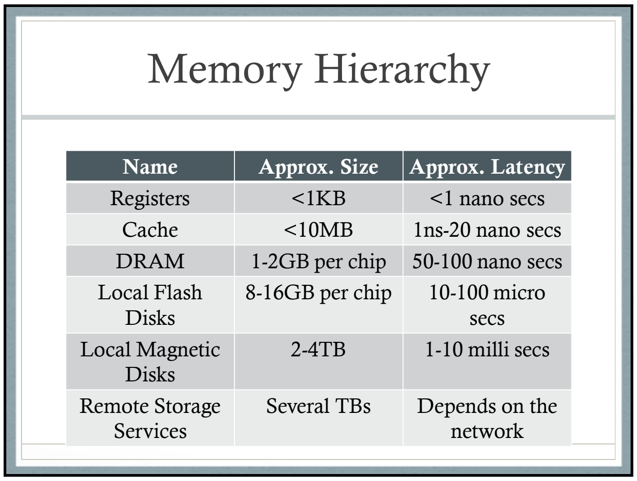
L2: L2 cache (SRAM) — L2 cache holds cache lines retrieved from L3 cache

L3: L3 cache (SRAM) — L3 cache holds cache lines retrieved from memory.

L4: Main memory (DRAM) — Main memory holds disk blocks retrieved from local disks.

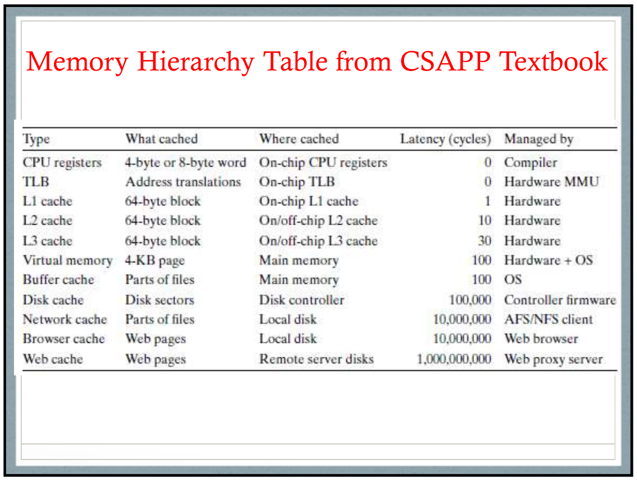
L5: Local secondary storage (local disks) — Local disks hold files retrieved from disks on remote network servers.

L6: Remote secondary storage (distributed file systems, Web servers)



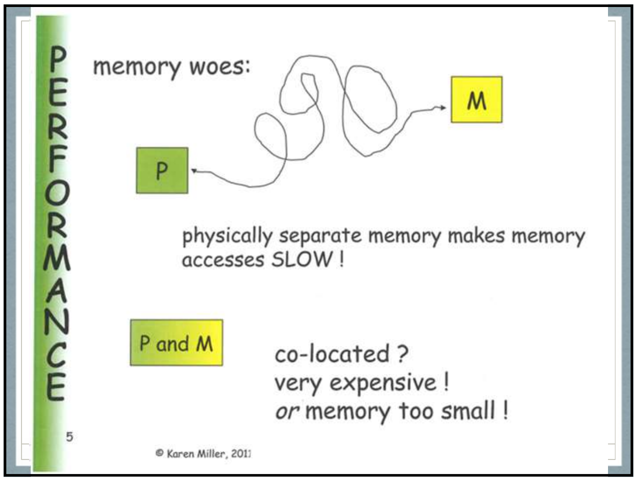
Memory Hierarchy

Name	Approx. Size	Approx. Latency
Registers	<1KB	<1 nano secs
Cache	<10MB	1ns-20 nano secs
DRAM	1-2GB per chip	50-100 nano secs
Local Flash Disks	8-16GB per chip	10-100 micro secs
Local Magnetic Disks	2-4TB	1-10 milli secs
Remote Storage Services	Several TBs	Depends on the network



Memory Hierarchy Table from CSAPP Textbook

Type	What cached	Where cached	Latency (cycles)	Managed by
CPU registers	4-byte or 8-byte word	On-chip CPU registers	0	Compiler
TLB	Address translations	On-chip TLB	0	Hardware MMU
L1 cache	64-byte block	On-chip L1 cache	1	Hardware
L2 cache	64-byte block	On/off-chip L2 cache	10	Hardware
L3 cache	64-byte block	On/off-chip L3 cache	30	Hardware
Virtual memory	4-KB page	Main memory	100	Hardware + OS
Buffer cache	Parts of files	Main memory	100	OS
Disk cache	Disk sectors	Disk controller	100,000	Controller firmware
Network cache	Parts of files	Local disk	10,000,000	AFS/NFS client
Browser cache	Web pages	Local disk	10,000,000	Web browser
Web cache	Web pages	Remote server disks	1,000,000,000	Web proxy server



PERFORMANCE

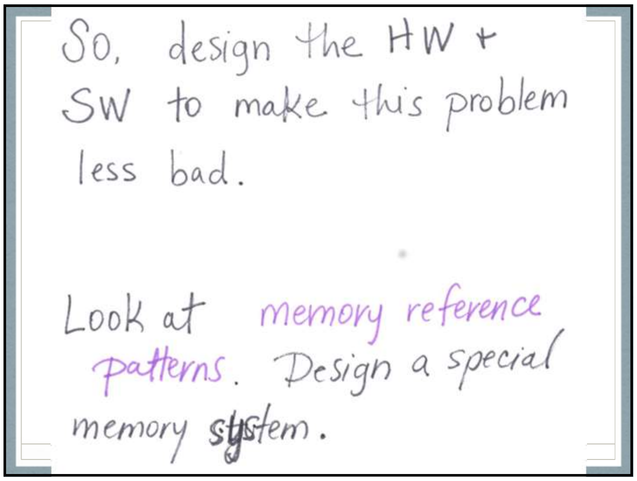
memory woes:

physically separate memory makes memory accesses SLOW !

co-located ?
very expensive !
or memory too small !

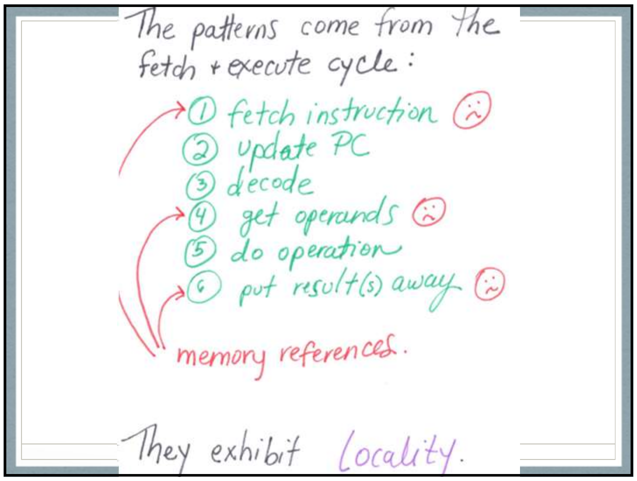
5

© Karen Miller, 2011



So, design the HW + SW to make this problem less bad.

Look at memory reference patterns. Design a special memory system.

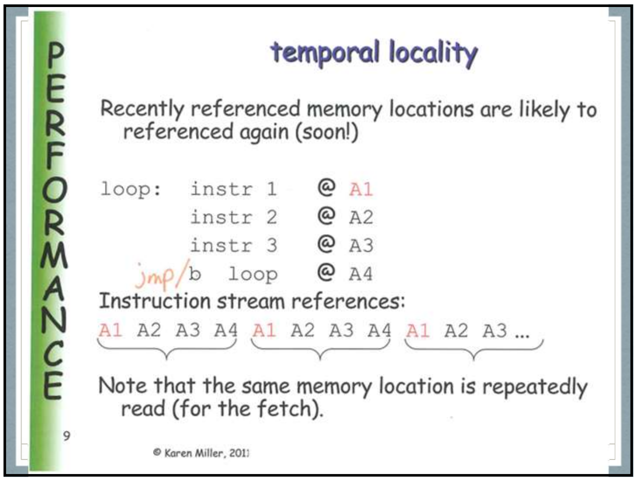


The patterns come from the fetch + execute cycle:

1. fetch instruction ☹
2. update PC
3. decode
4. get operands ☹
5. do operation
6. put result(s) away ☹

memory references.

They exhibit locality.



PERFORMANCE

temporal locality

Recently referenced memory locations are likely to referenced again (soon!)

```
loop:  instr 1    @ A1
       instr 2    @ A2
       instr 3    @ A3
   jmp/b  loop    @ A4
```

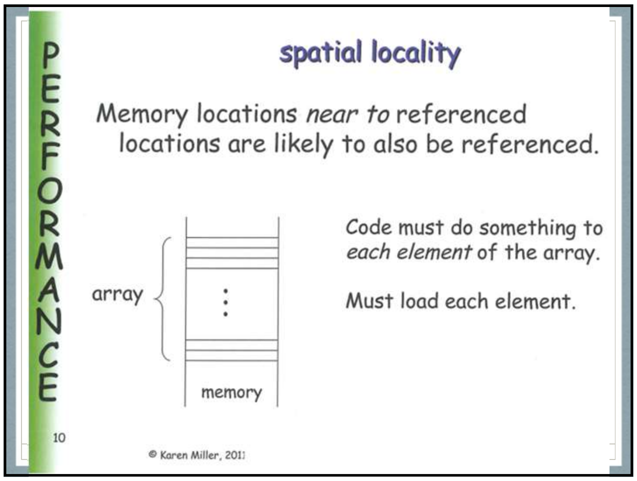
Instruction stream references:

A1 A2 A3 A4 A1 A2 A3 A4 A1 A2 A3 ...

Note that the same memory location is repeatedly read (for the fetch).

9

© Karen Miller, 2011



PERFORMANCE

spatial locality

Memory locations *near to* referenced locations are likely to also be referenced.

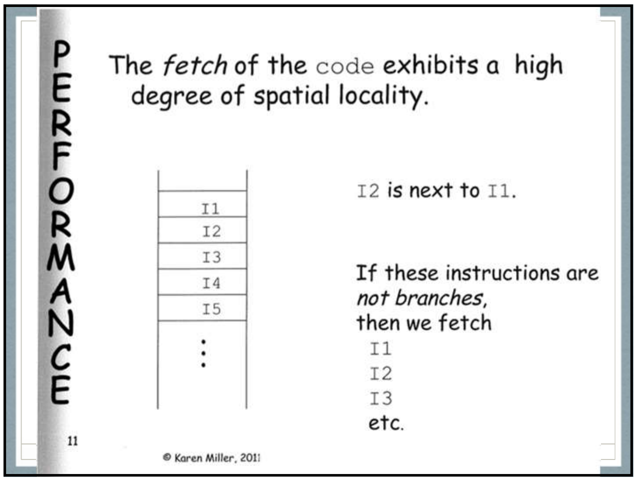
array
memory

Code must do something to *each element* of the array.

Must load each element.

10

© Karen Miller, 2011



PERFORMANCE

The *fetch* of the code exhibits a high degree of spatial locality.

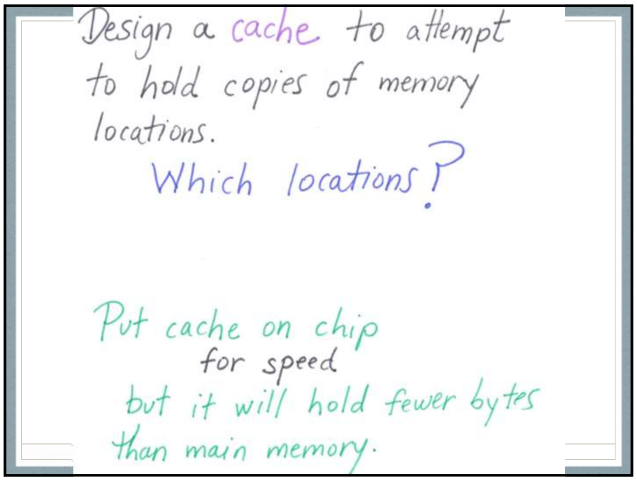
I1
I2
I3
I4
I5

I2 is next to I1.

If these instructions are *not branches*, then we fetch
I1
I2
I3
etc.

11

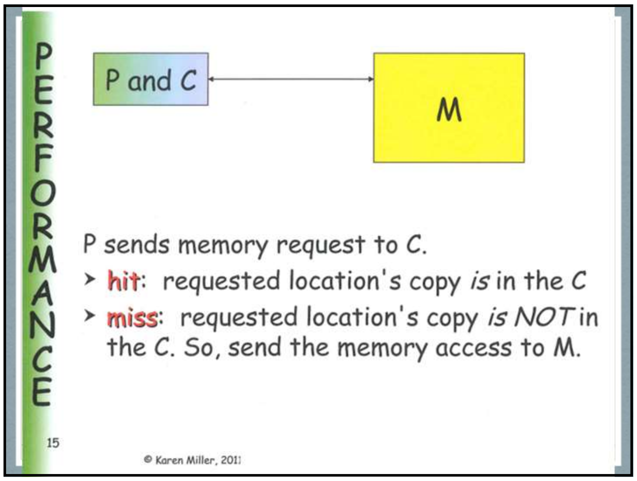
© Karen Miller, 2011



Design a cache to attempt to hold copies of memory locations.

Which locations?

Put cache on chip
for speed
but it will hold fewer bytes than main memory.



PERFORMANCE

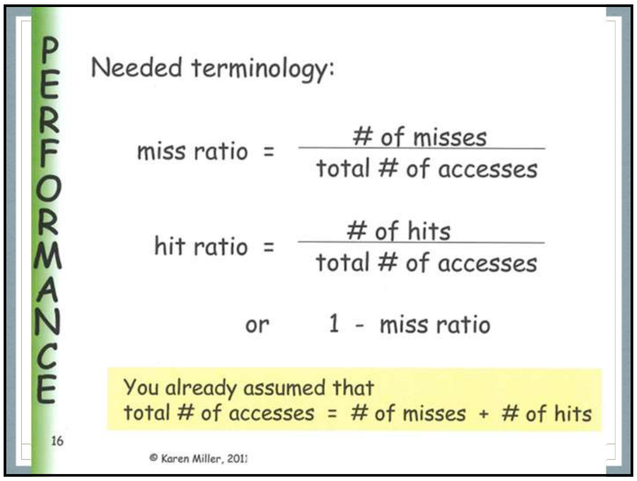
P and C ↔ M

P sends memory request to C.

- **hit**: requested location's copy *is* in the C
- **miss**: requested location's copy *is NOT* in the C. So, send the memory access to M.

15

© Karen Miller, 2011



PERFORMANCE

Needed terminology:

$$\text{miss ratio} = \frac{\#\text{ of misses}}{\text{total }\#\text{ of accesses}}$$

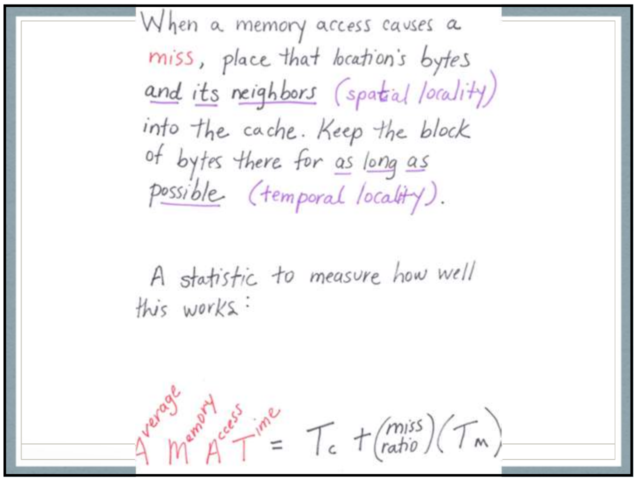
$$\text{hit ratio} = \frac{\#\text{ of hits}}{\text{total }\#\text{ of accesses}}$$

or 1 - miss ratio

You already assumed that
total # of accesses = # of misses + # of hits

16

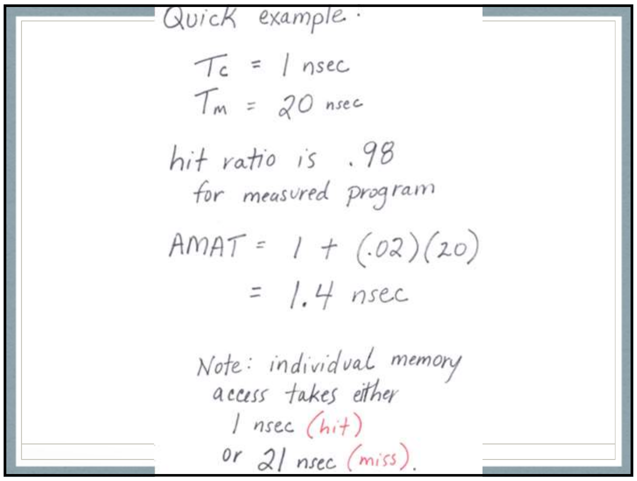
© Karen Miller, 2011



When a memory access causes a miss, place that location's bytes and its neighbors (spatial locality) into the cache. Keep the block of bytes there for as long as possible (temporal locality).

A statistic to measure how well this works:

$$A^{\text{verage}}M^{\text{emory}}A^{\text{ccess}}T^{\text{ime}} = T_c + \left(\begin{smallmatrix}\text{miss}\\ \text{ratio}\end{smallmatrix}\right)(T_M)$$



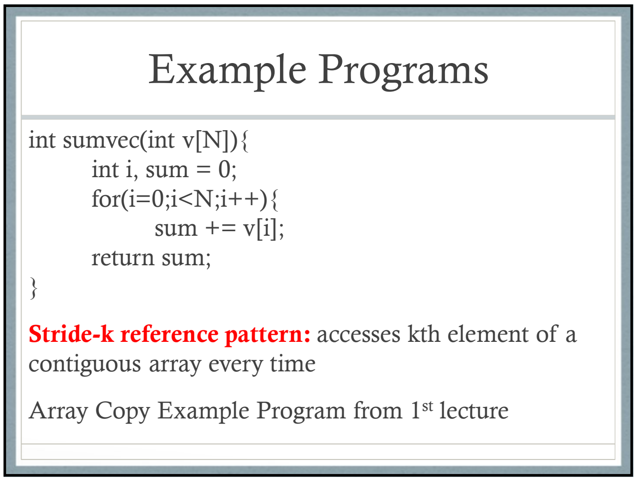
Quick example:

$T_c = 1$ nsec
$T_M = 20$ nsec

hit ratio is .98
for measured program

$$\text{AMAT} = 1 + (.02)(20)$$
$$= 1.4 \text{ nsec}$$

Note: individual memory access takes either
1 nsec (hit)
or 21 nsec (miss).



Example Programs

```
int sumvec(int v[N]){
        int i, sum = 0;
        for(i=0;i<N;i++){
                sum += v[i];
        return sum;
}
```

Stride-k reference pattern: accesses kth element of a contiguous array every time

Array Copy Example Program from 1st lecture