

# Memory Hierarchies

---

- ◆ **Last Time**
  - Advanced Pipelining
    - » Superpipelining, Superscalar, Dynamic Pipelining
- ◆ **This Time**
  - CAPES?
  - Quiz
  - Memory Hierarchies and Locality
    - » why do they work? How do they work?
  - Cache basics
- ◆ **Reminders/Announcements**
  - Reading is 7.5-7.10
  - Midterm Solutions and HW3 solutions are on the Web
  - Homework #4, Due March 7
  - Graded Midterms returned at the end of class today

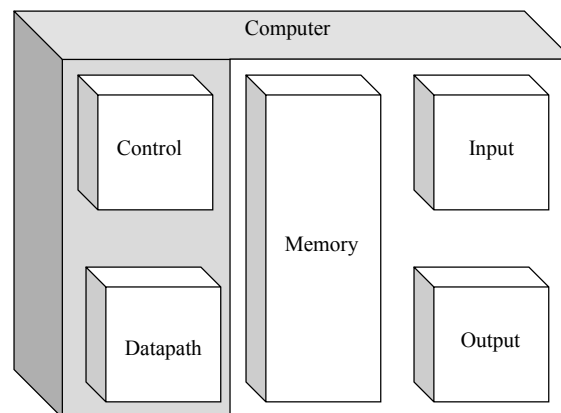
CS 141 Chien

1

March 2, 2000

# Basic Components: the Memory

---



CS 141 Chien

2

March 2, 2000

## Component Speeds

---

### ◆ Processor Speeds

- Intel Pentium III : 800 Mhz =
- Compaq Alpha 21264: 1,000 Mhz =
- Others?

### ◆ Memory Speeds

- Mac/PC/Workstation DRAM: 50 ns
- Disks are even slower....

### ◆ How can we span this “access time” gap?

- 1 instruction fetch per instruction
- 15/100 instructions also do a data read or write (load or store)

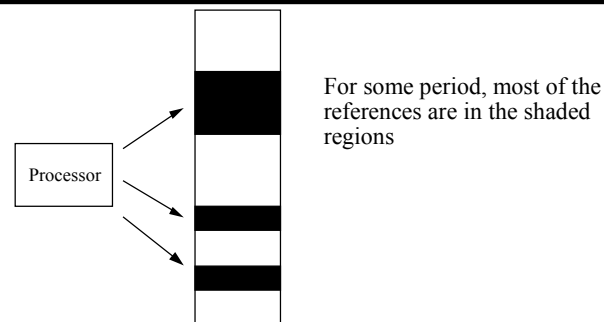
CS 141 Chien

3

March 2, 2000

## Locality

---



- ◆ Property of memory references in “typical programs”
- ◆ Tendency to favor a portion of their address space at any given time
- ◆ A key elements in computer organization for high performance

CS 141 Chien

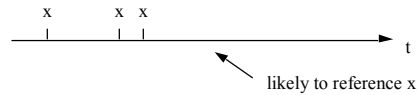
4

March 2, 2000

## Aspects of Locality

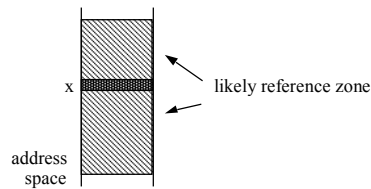
### ◆ Temporal

- Tendency to reference locations recently referenced



### ◆ Spatial

- Tendency to reference locations "near" those recently referenced



CS 141 Chien

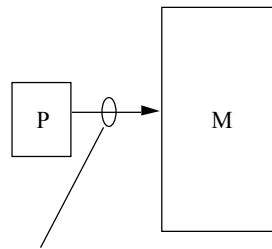
5

March 2, 2000

## Memory References



execute a program



look at what crosses here

- ◆ Memory reference streams / traces
- ◆ Memory requests, addresses, data
- ◆ Focus on the ADDRESSES....

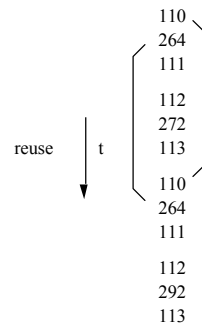
CS 141 Chien

6

March 2, 2000

## Temporal Locality (in time)

### ◆ Program Reference Stream (instruction and memory)



Sources of temporal locality

Instruction fetch (loops)

Locals (loops, repeated invocations)

Data Structures

xxx Instruction fetches (IF)

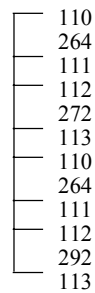
xxx Data Memory Access

CS 141 Chien

7

March 2, 2000

## Spatial Locality (in space)



### ◆ Sources of spatial locality

- Instruction sequence
- Indexing Arrays
- Locals in a stack frame
- Contiguous allocation

xxx Instruction fetch

xxx Data memory access

CS 141 Chien

8

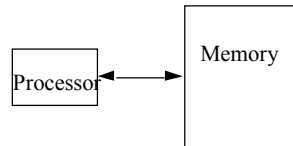
March 2, 2000

## Importance of Locality

---

### ◆ Why does locality matter?

- An opportunity for the computer designer.... make the computer cheaper, faster, etc.



- process needs 256MB
- processor can run at 500MHz
- 2ns memory?
  - expense
  - feasibility

What does locality buy us?

## Memory Locality

---

- ◆ Memory hierarchies take advantage of *memory locality*.
- ◆ *Memory locality* is the principle that future memory accesses are *near* past accesses.
- ◆ Memories take advantage of two types of locality
  - *Temporal locality* -- near in time => we will often access the same data again very soon
  - *Spatial locality* -- near in space/distance => our next access is often very close to our last access (or recent accesses).

## Locality and caching

- ◆ **Memory hierarchies exploit locality by *cacheing* (keeping close to the processor) data likely to be used again.**
- ◆ **This is done because we can build large, slow memories and small, fast memories, but we can't build large, fast memories.**
- ◆ **If it works, we get the illusion of SRAM access time with disk capacity**

SRAM access times are 2 - 25ns at cost of \$500 to \$125 per Mbyte.

DRAM access times are 40-80ns at cost of \$1 to \$4 per Mbyte.

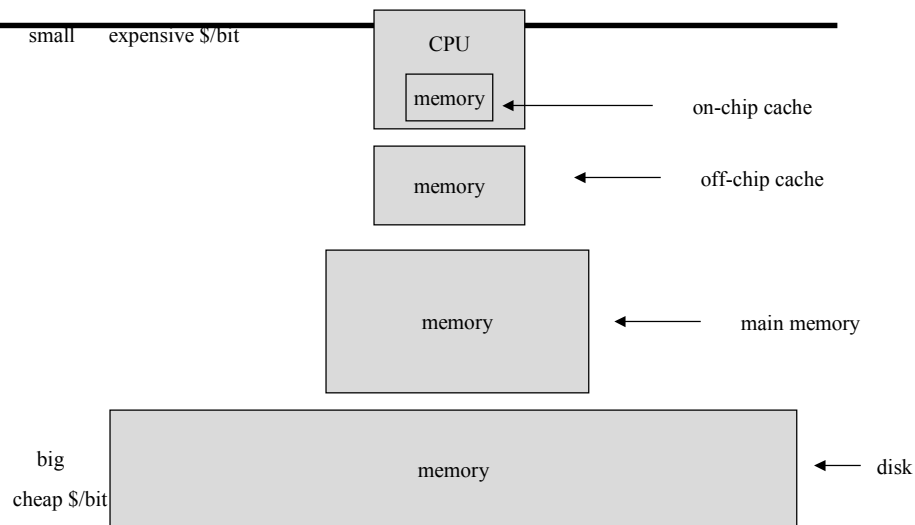
Disk access times are 10 to 20 million ns at cost of <\$.01 per Mbyte.

CS 141 Chien

11

March 2, 2000

## A typical memory hierarchy



CS 141 Chien

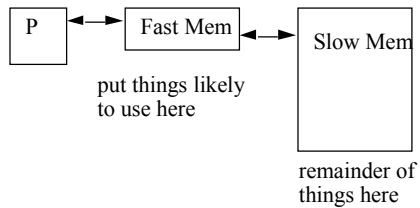
•so then where is my program and data??

March 2, 2000

## How to exploit locality?

---

**Q: How do we use a memory hierarchy to exploit locality?**



**How do we decide?**

- **explicit control**      Registers
- **implicit control**      Caches; what we'll talk about.

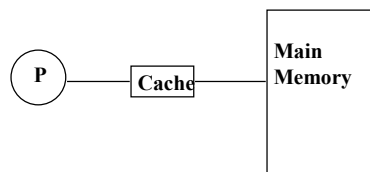
CS 141 Chien

13

March 2, 2000

## Caches: The Idea

---



- ◆ **Processor accesses “Memory hierarchy”**
- ◆ **Accesses first check small, fast “cache”**
- ◆ **If “miss” in cache, retrieve data from main memory**
- ◆ **Cache exploits locality to provide high performance**

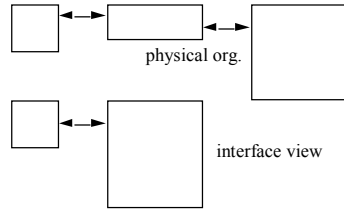
CS 141 Chien

14

March 2, 2000

# Implicit Control

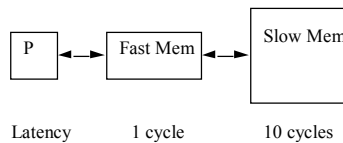
```
LD R0,#00FF
ST R1,#5F6F
```



- ◆ Hardware controlled (or OS) data movement
- ◆ Generic Memory/Address operation interface
- ◆ Examples:
  - Caches -- implicitly managed memory hierarchies
  - design HW to exploit locality
  - fast, concurrent movement
  - no need for addressing changes
  - "performance visible only!"

# Amortized Access Cost (Avg)

- ◆ Measure set of access costs
- ◆ Weight by frequency

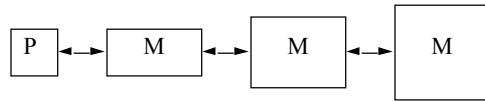


Assumptions:

- 90% accesses to 10% of addresses
- magic 10% in fast memory
- remainder in slow memory

$$\begin{aligned} \text{Avg. Access Cost} &= 0.9 * 1 + 0.1 * 10 = 0.9 + 1.0 = 1.9 \\ &= (\text{frac. of fast mem refs}) * (\text{fast mem access cost}) + \\ &\quad (\text{frac. of slow mem refs}) * (\text{slow mem access cost}) \end{aligned}$$

## Terminology (implicitly managed)



Hit - find desired data at desired level

Miss - don't find data at desired level

- typically search from processor outward (speed)

Hit time - time to access, if data found (w.r.t. a particular level)

Miss Penalty - time to bring desired data to this level (following a miss) and deliver the data to the CPU.

Access time + transfer time  
(mem latency, next level) (multiple transfers for larger data sizes)

### ◆ Hit, Miss

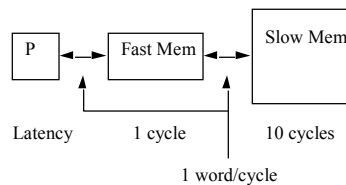
### ◆ Hit Time, Miss penalty (time)

CS 141 Chien

17

March 2, 2000

## Example



100 references --> 85 cache hits  
15 misses

Total time = (# hits) \* (hit time) + (# misses) \* (miss penalty)  
(MM access time + transfer time)

$$= 85 * 1 + 15 * (10 + 3) = 85 + 195$$

$$= 280 \text{ --> Average access time} = 2.8 \text{ cycles}$$

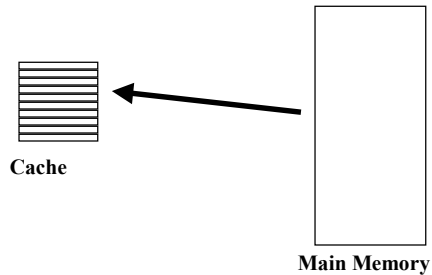
CS 141 Chien

18

March 2, 2000

## Cache Design Issues

---



- ◆ Data is copied into the cache.
- ◆ Where to put the data? (which location)
- ◆ Which data to replace? (later)

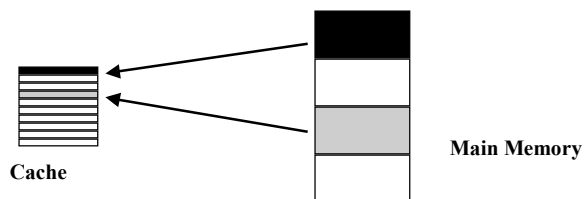
CS 141 Chien

19

March 2, 2000

## Cache Organization (Basic)

---



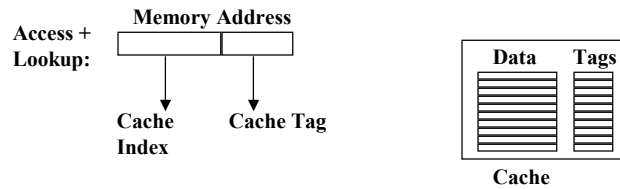
- ◆ Map each region of address space to a single cache location
  - Direct Mapped Cache
  - Mapping: Discard some address bits
    - Discard lowest (shown above)
    - Discard highest is more typical (interleaved)

CS 141 Chien

20

March 2, 2000

## Finding Cache Data



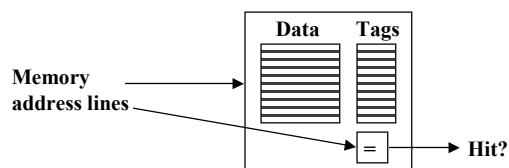
- ◆ For fast access, must be able to find data quickly
- ◆ Lookup: discard a few address bits, access the fast memory
- ◆ Tags indicate the data that's really there
- ◆ Why does this work?
  - Memory location + tag -> complete address

CS 141 Chien

21

March 2, 2000

## Cache Access



- ◆ Part of Memory Address applied to cache
- ◆ Tags checked against remaining address bits
- ◆ If match -> Hit!, use data
- ◆ No match -> Miss, retrieve data from memory
- ◆ This works pretty well... but there are some complications...

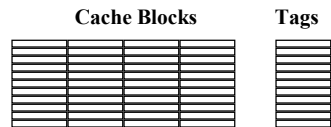
CS 141 Chien

22

March 2, 2000

## Multiword Cache Entries

---



- ◆ Cache entries are several contiguous words of memory
- ◆ Shared tag, involves only contiguous addresses
  - reduces tag overhead
  - exploits spatial locality
- ◆ A few address lines select the word from the line on a “hit”.... Which ones?

## Summary

---

- ◆ Memory time access gap
- ◆ Locality: structure in typical address references
- ◆ Caches: automatically managed fast storage, get some of the performance of the fast, at price & capacity of the slow
  - Cache basics: amortized access cost, finding the data
- ◆ Next time: cache organizations and management