Improving System Performance: Caches

December 04

CSC201 Section 002

Fall, 2000

A Motivating Example

- Application: making a (mechanical) clock
- dozens of tools and pages of instructions, hundreds of parts / materials
- Current work: what's in hand
- Temporary storage: workbench surface
- Large-scale storage: the garage
- Storage of last resort: mail-order catalog warehouse

A Motivating Example

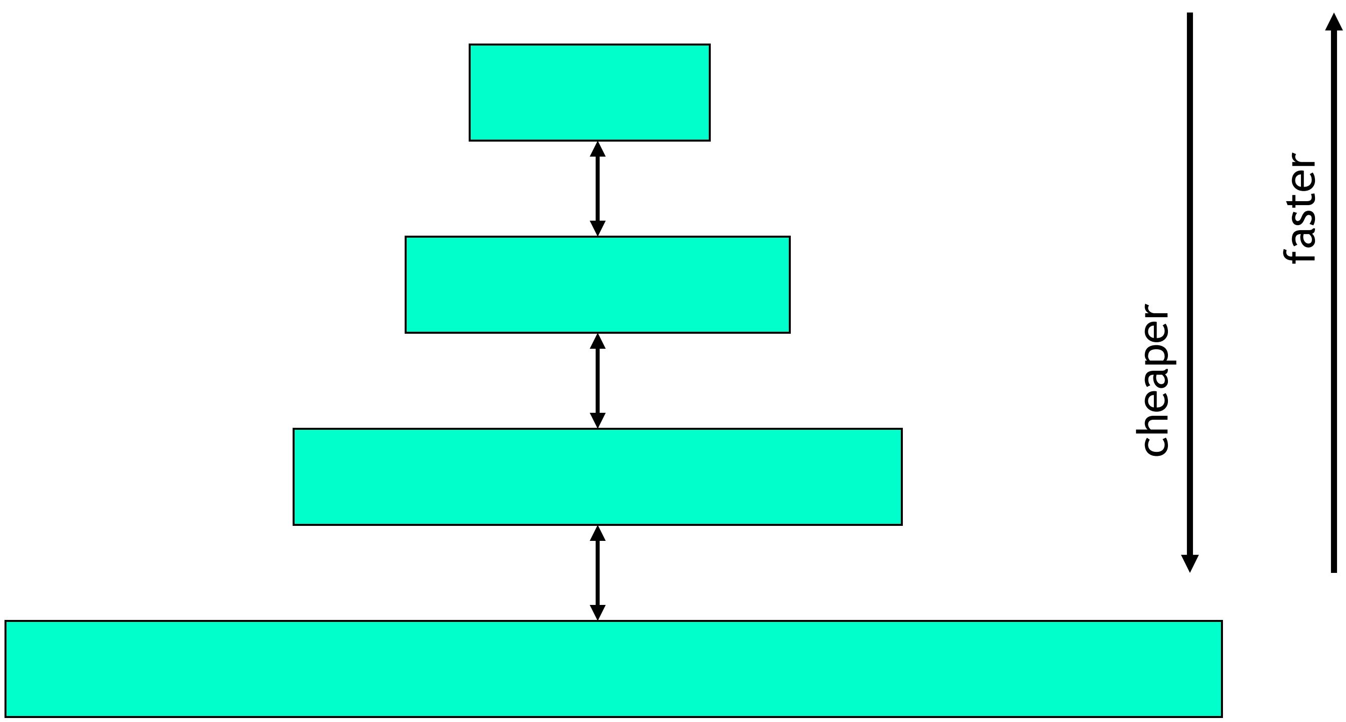
- What happens when you need a tool or part?
  - Check workbench.
  - Not found? check garage.
  - Not found? Order from catalog (and go do something else).
- Performance
  - How often is it found on workbench, in garage, or catalog?
  - How much time does it take to access from workbench, garage, or catalog?
- Victim: something has to go to make room for the new part or tool. Who?
- Other improvements?
  - prefetch?
  - cabinet between workbench and garage?

The Memory Hierarchy

- Different memory technologies: semiconductors vs. magnetic disks, static RAM vs. dynamic RAM, on-chip vs. off-chip, ...
- Memory stores instructions and variables
  - we'll assume the unit of access is a doubleword
- Tradeoff: faster vs. cheaper
  - faster: access time (time to read or write a doubleword)
  - cheaper: cost per bit (can afford more memory if cheaper)

The Memory Hierarchy

Level	Memory	Speed (ns)	$ per Mb
1	Registers	1-2	100.00
2	On-chip cache	2-4	20.00
3	Off-chip cache	4-10	10.00
4	Main memory	20-100	.20
5	Disk	10^6	.001

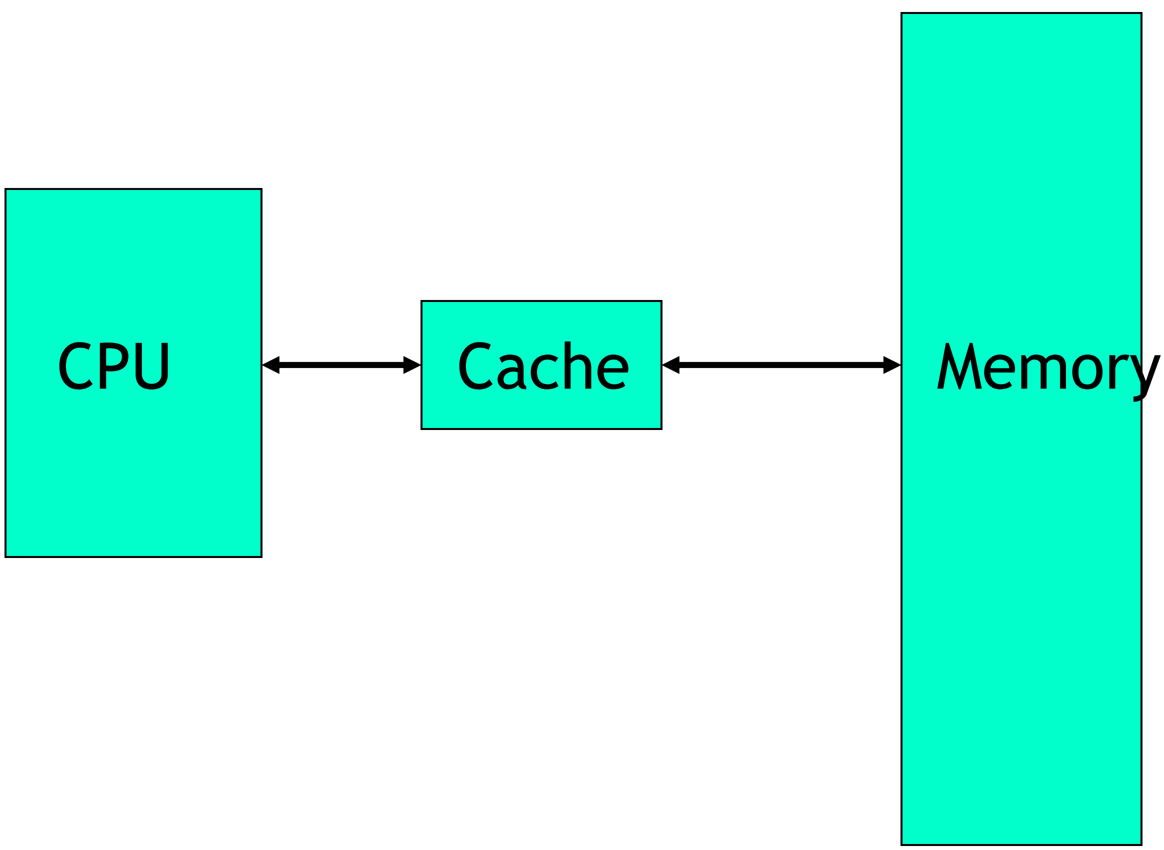


Locality of Memory References

- Some variables and instructions are fetched from memory repeatedly
  - loops
  - important subroutines
  - counters
  - important parameters
  - flags
- Some variables and instructions are fetched in a predictable way
  - sequential execution of a program
  - sequential processing of an array
- Goal of caching: exploit locality!
- For our motivating example....?

Caches and Their Effect on Performance

- Inclusion: level i+1 contains everything found at level i, and more
- Searching:
  - Check cache (level 1)
  - Not found? Check memory (level 2)



Caches and Their Effect on Performance

- Finding a doubleword in cache = "hit"
  - The opposite of a hit is a "miss"
- Critical factors for performance
  - Hit (or miss) rate at each level
  - Access time at each level
- Formula for average memory access time (2 levels)

$$T_{avg} = t_{cache} + M_{cache} * t_{mem}$$

- Example
  - T_{cache} = 1ns
  - Tmem = 10ns
  - M_{cache} = .1 (10% miss rate)

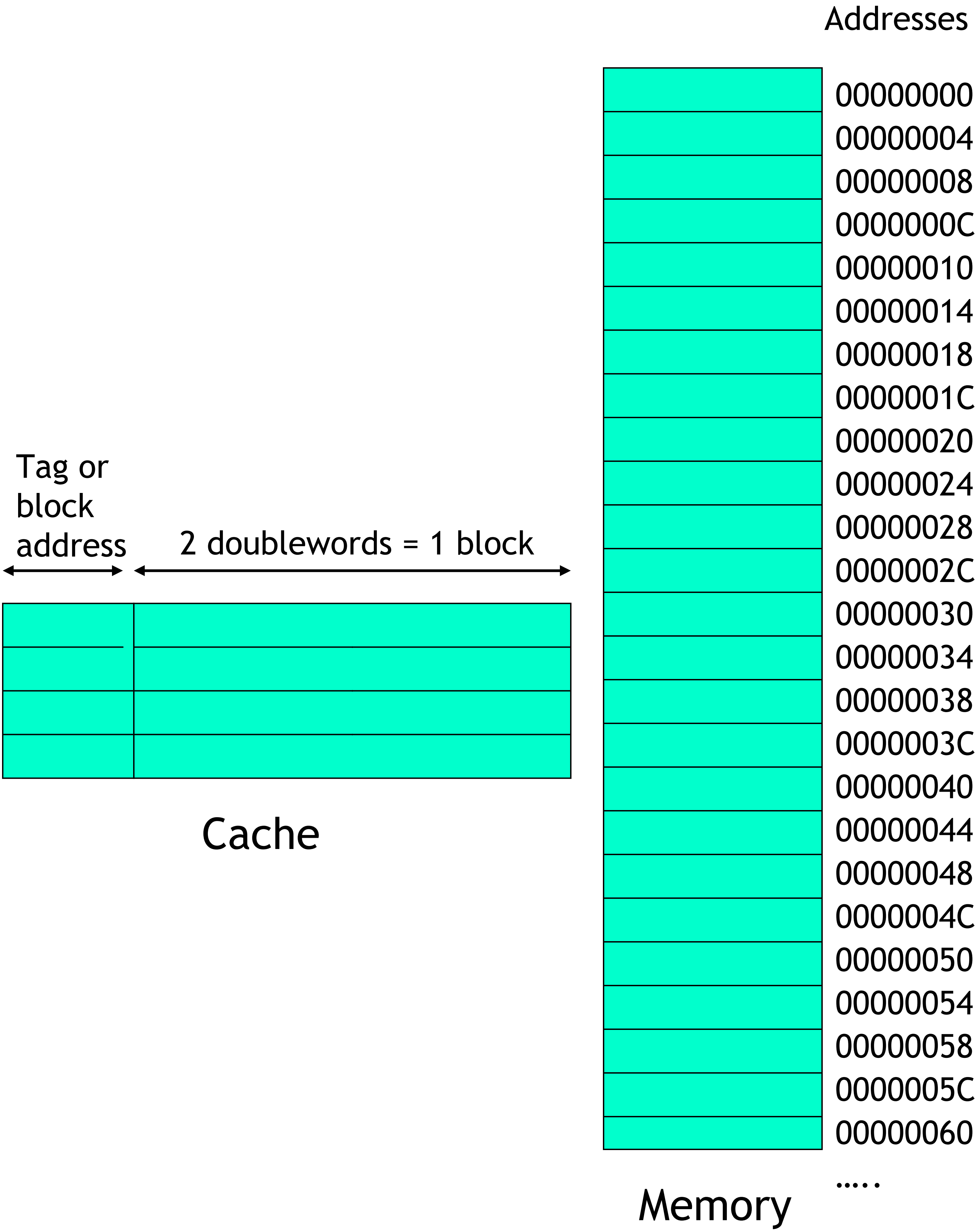
$$T_{avg} = 1 + .1 * 10 = 2ns$$

Caches and Their Effect on Performance

- In reality, hit rates range from 95-99%
- Improving memory system performance
  - Reduce access times!
  - Increase hit rates!
  - Add another level to the hierarchy!

How Caches Are Built

- Read a "block" of data from memory
  - 1 block = 2-16 doublewords
- Store the block as a "line" in the cache
- Every line in the cache has a "valid" bit
  - Initially, all lines are invalid
  - Becomes valid when you store a memory block in it
- How do you identify where a cache line came from?
  - Store the block address with the line


Addresses
00000000
00000004
00000008
0000000C
00000010
00000014
00000018
0000001C
00000020
00000024
00000028
0000002C
00000030
00000034
00000038
0000003C
00000040
00000044
00000048
0000004C
00000050
00000054
00000058
0000005C
00000060
.....
Tag or block address
2 doublewords = 1 block
Cache
Memory

How Caches Are Built

- How do you find the address you're looking for?
  - Search the block addresses one at a time (sequentially)
  - Search the block addresses all at the same time (in parallel)
- Memory that can be searched in parallel = "associative" memory
  - Fast searching, but expensive to build

Placement Policy

- Where in the cache do you put a block when you bring it from memory?
  - In any line (cache is "associative")
  - Always in one specific line (cache is "direct")
  - In one of a few lines (cache is "set associative")

Other Cache Policies

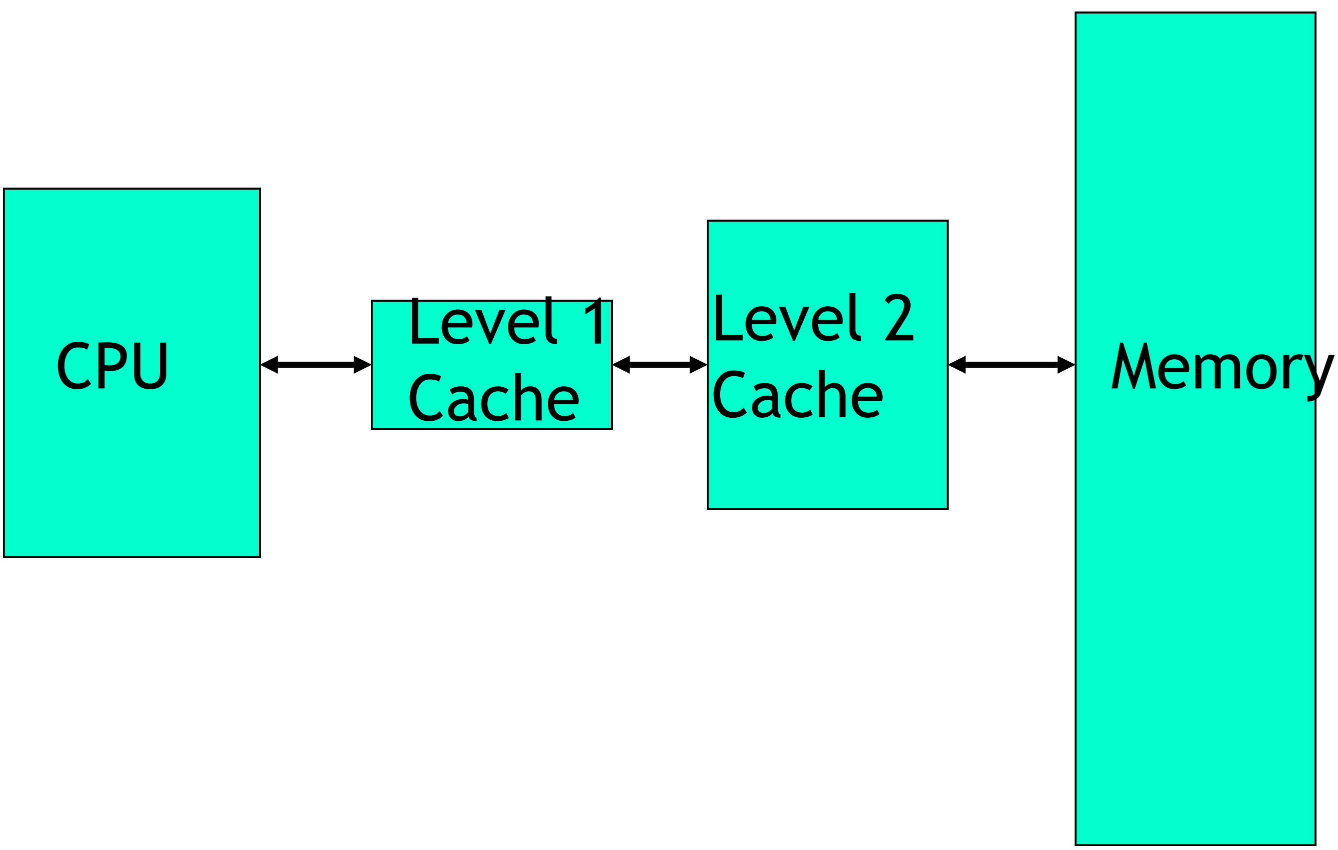
1. Prefetch (blocks from memory)
   - If you can predict what will be needed
2. Replacement (choosing a victim)
   - "Least recently used" or "least frequently used" are good candidates
3. Let programmer provide hints about what to cache
   - Used for prefetching, and for replacing

Other Cache Policies

4. Writing a new value to memory
   - "Write-through": update cache and memory at the same time
   - "Write-back": update memory when the block is replaced in the cache

Cache Improvements

- Use *two* caches: one for data, and one for code (instructions)
  - Access both at same time
  - Optimize them independently
- Use multiple levels of cache
  - Most processors today have at least a first and second level cache



Programming for Improved Cache Performance

- Programmer inserts hints in the program
  - E.g., "This is an important variable; prefetch it, and keep it in the cache"
- Programmer writes the program to maximize the cache hit rate
  - Design code to improve locality
  - Increase frequency of access?
  - Increase predictability of access?