Computer Organization and Design

Memory Hierarchy

Henry Fuchs

Slides adapted from Anselmo Lastra and from Montek Singh, who adapted them
from Leonard McMillan and from Gary Bishop
Back to McMillan & Chris Terman, MIT 6.004 1999

Thursday, April 9, 2015

Lecture 16

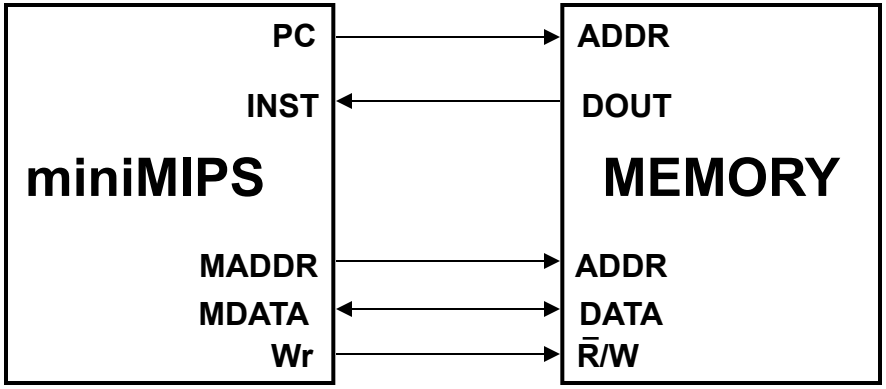
Reading: Ch. 5.1-5.3

Topics

- Memory Types
- Principle of Locality
- Memory Hierarchy: Caches
- Associativity

- Reading: Ch. 5.1-5.3

What Do We Want in a Memory?

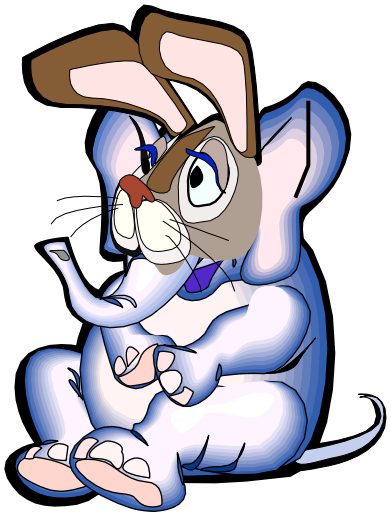


	Capacity	*Latency*	*Cost*
Register	1000s of bits	10 ps	$$$$
SRAM	1-8 MBytes	0.5-1 ns	$2,000/GB
DRAM	1-8 GBytes	50 ns	$20/GB
Hard disk*	100s GBytes	10 ms	$0.20/GB
Want?	Huge	Low (fast)	Cheap!

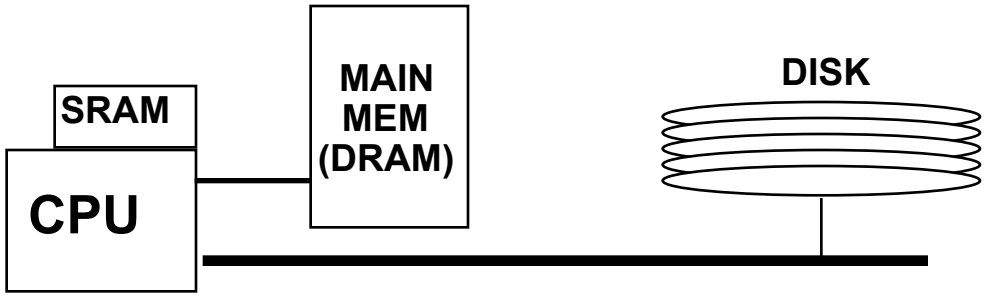
*non-volatile: Typically magnetic disk drive, but also SSD/flash now

Best of Both Worlds

- What we REALLY want: A BIG, FAST memory!
  - Keep everything within instant access



- We'd like to have a memory system that
  - Stores 2-16 GB of data like DRAM would
    - typical SRAM sizes are in MB, not GB
  - Performs fast like SRAM would
    - typical DRAMs are order of magnitude slower than SRAM
  - SURPRISE: We can (nearly) get our wish!
    - Key Idea: Use a hierarchy of memory technologies:



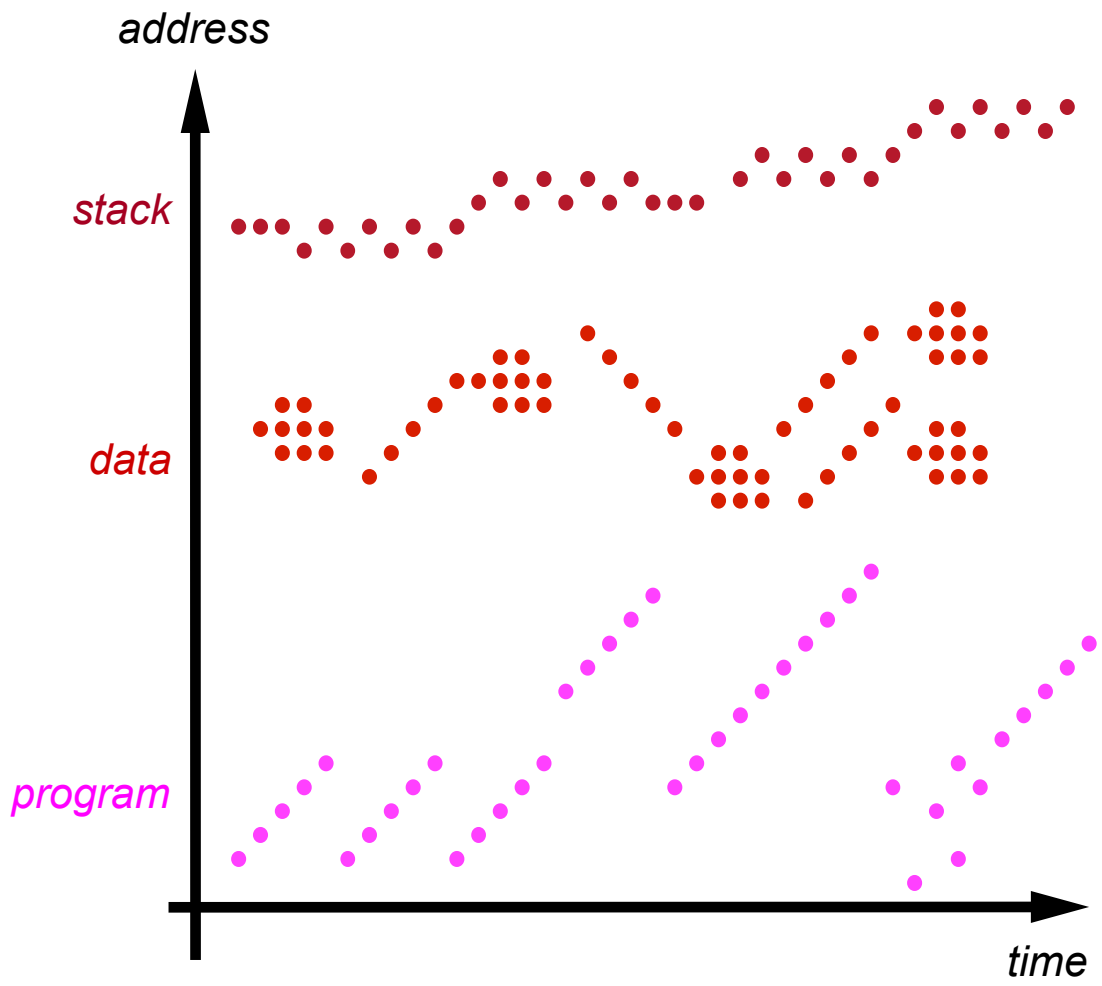
Principle of Locality

- Key Idea: Exploit "Principle of Locality"
  - Keep data used often in a small fast SRAM
    - called "CACHE", often on the same chip as the CPU
  - Keep all data in a bigger but slower DRAM
    - called "main memory", usually separate chip
  - Access Main Memory only rarely, for remaining data
  - The reason this strategy works: LOCALITY
    - if you access something now, you will likely access it again (or its neighbors) soon

Locality of Reference:

Reference to location X at time t implies that reference to location $X+\Delta X$ at time $t+\Delta t$ is likely for small ΔX and Δt.

Typical Memory Reference Patterns



- Memory Trace
  - A temporal sequence of memory references (addresses) from a real program.
- Temporal Locality
  - If an item is referenced, it will tend to be referenced again soon
- Spatial Locality
  - If an item is referenced, nearby items will tend to be referenced soon.

Cache

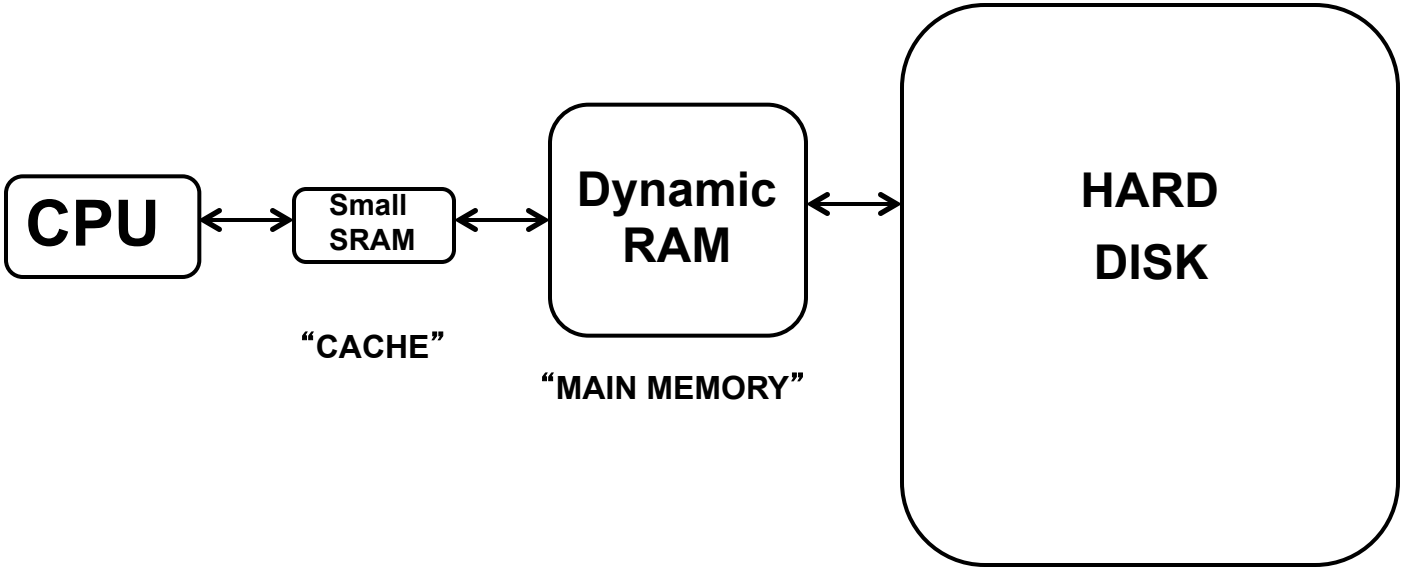
- cache (ˈkash)
  n.
  - A hiding place used especially for storing provisions.
  - A place for concealment and safekeeping, as of valuables.
  - The store of goods or valuables concealed in a hiding place.
  - *Computer Science.* A fast storage buffer in the central processing unit of a computer. In this sense, also called cache memory.
- *v. tr.* cached, cachˑing, cachˑes.
  - To hide or store in a cache.

Cache Analogy

- You are writing a term paper for your history class at a table in the library
  - As you work you realize you need a book
  - You stop writing, fetch the reference, continue writing
  - You don't immediately return the book, maybe you'll need it again
  - Soon you have a few books at your table, and you can work smoothly without needing to fetch more books from the shelves
  - The table is a CACHE for the rest of the library
- Now you switch to doing your biology homework
  - You need to fetch your biology textbook from the shelf
  - If your table is full, you need to return one of the history books back to the shelf to make room for the biology book

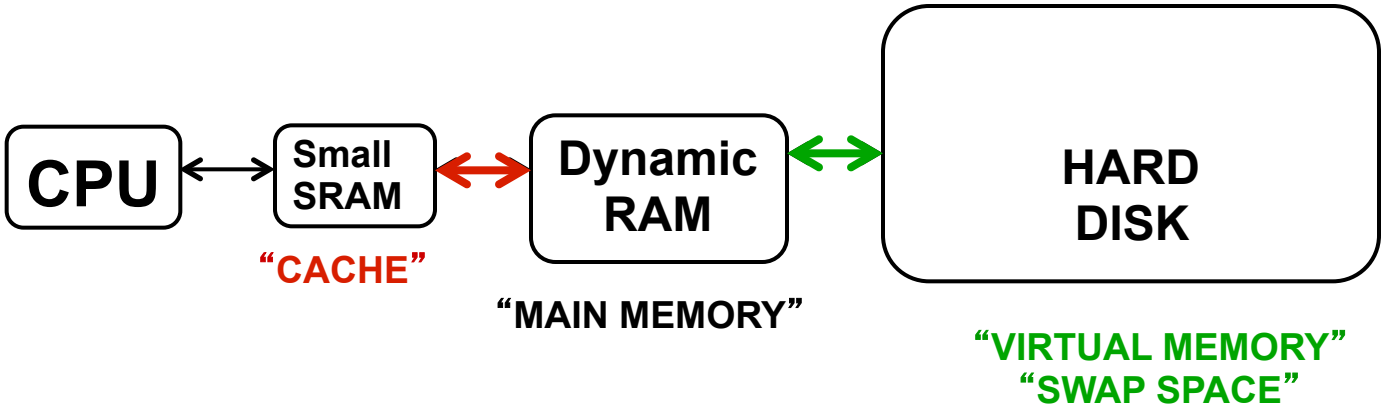
Exploiting the Memory Hierarchy

- Memory Hierarchy is hidden from programmer!
  - Programming model: SINGLE kind of memory, single address space.
  - Transparent to programmer: Machine AUTOMATICALLY assigns locations, depending on runtime usage patterns.
    - programmer doesn't (***cannot***) know where the data is actually stored!



Exploiting the Memory Hierarchy

- CPU speed is dominated by memory performance
  - More significant than: ISA, circuit optimization, pipelining, etc.

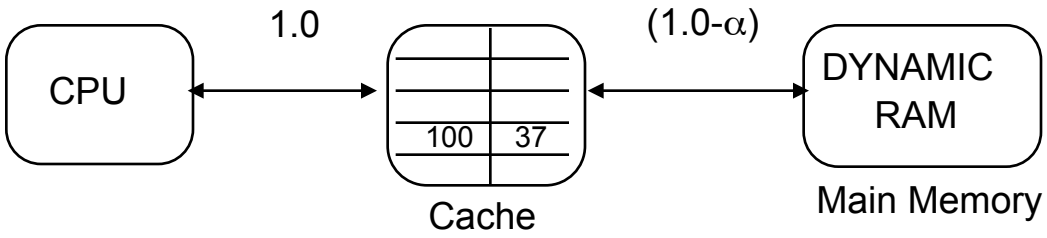


  - TRICK #1: Make slow MAIN MEMORY appear faster
    - Technique: CACHING
  - TRICK #2: Make small MAIN MEMORY appear bigger
    - Technique: VIRTUAL MEMORY

The Cache Idea

Program-Transparent Memory Hierarchy:

- Cache contains TEMPORARY COPIES of selected main memory locations…
  - e.g. Mem[100] = 37



✱ Two Goals:

- Improve the average memory access time
  - HIT RATIO (α): Fraction of refs found in cache
  - MISS RATIO (1-α): Remaining references
  - average total access time depends on these parameters (and time to access cache and memory)

$$t_{ave} = \alpha t_c + (1-\alpha)(t_c + t_m) = t_c + (1-\alpha)t_m$$

Challenge:
To make the hit ratio as high as possible.

- Transparency (compatibility, programming ease)

How High of a Hit Ratio?

- Suppose we can easily build an on-chip SRAM with a 0.8 ns access time, but the fastest DRAM we can buy for main memory has access time of 10 ns. How high of a hit rate do we need to sustain an average total access time of 1 ns?

$$t_{ave} = \alpha t_c + (1-\alpha)(t_c + t_m) = t_c + (1-\alpha)t_m$$

$$\alpha = 1 - \frac{t_{ave} - t_c}{t_m} = 1 - \frac{1 - 0.8}{10} = 98\%$$

Wow, a cache really needs to be good!

Cache

- Sits between CPU and main memory
- Very fast table that stores a TAG and DATA
  - TAG is the memory address
  - DATA is a copy of memory contents at the address given by TAG

Tag	Data
1000	17
1040	1
1032	97
1008	11

Cache

Main Memory

1000	17
1004	23
1008	11
1012	5
1016	29
1020	38
1024	44
1028	99
1032	97
1036	25
1040	1
1044	4

Cache Access

- On load (lw) we look in the TAG entries for the address we're loading
  - Found → a HIT, return the DATA
  - Not Found → a MISS, go to memory for the data and put it and the address (TAG) in the cache

Tag	Data
1000	17
1040	1
1032	97
1008	11

Cache

Main Memory

1000	17
1004	23
1008	11
1012	5
1016	29
1020	38
1024	44
1028	99
1032	97
1036	25
1040	1
1044	4

Cache Lines

- Usually get more data than requested
  - a LINE is the unit of memory stored in the cache
  - usually much bigger than 1 word, 32 bytes per line is common
  - bigger LINE means fewer misses because of spatial locality
  - but bigger LINE means longer time on miss

Tag	Data	
1000	17	23
1040	1	4
1032	97	25
1008	11	5

Cache

Main Memory

1000	17
1004	23
1008	11
1012	5
1016	29
1020	38
1024	44
1028	99
1032	97
1036	25
1040	1
1044	4

Finding the **Tag** (address) in the Cache

- Fully Associative:
  - Requested data could be anywhere in the cache
  - Fully Associative cache uses hardware to compare the address to the tags in parallel but it is expensive
    - 1 MB is thus unlikely, typically smaller
- Direct Mapped Cache:
  - Directly computes the cache entry from the address
    - multiple addresses will map to the same cache line
    - use TAG to determine if right
  - Choose some bits from the address to determine cache entry:
    Cache size: 32 bytes in each line; 32k lines
    - low 5 bits determine which byte within the line of 32 bytes
    - we need 15 bits to determine which of the 32k different lines has the data
    - which of the 32 – 5 = 27 remaining bits should we use?

Direct-Mapping Example

- Suppose: 2 words/line, 4 lines, bytes are being read
  - With 8 byte lines, bottom 3 bits determine byte within line
  - With 4 cache lines, next 2 bits determine which line to use
    - 1024d = 10000**00**000b → line = 00b = 0d
    - 1000d = 01111**01**000b → line = 01b = 1d
    - 1040d = 10000**10**000b → line = 10b = 2d

Adr	Tag	Data	
00	1024	44	99
01	1000	17	23
10	1040	1	4
11	1016	29	38

Cache

Main memory

1000	17
1004	23
1008	11
1012	5
1016	29
1020	38
1024	44
1028	99
1032	97
1036	25
1040	1
1044	4

Direct Mapping Miss

- What happens when we now ask for address 1008?
  - 1008d = 0111111**10**000b → line = 10b = 2d
- …but earlier we put 1040d there
  - 1040d = 1000001**10**000b → line = 10b = 2d
- …so evict 1040d, put 1008d in that entry

Adr	Tag	Data	
00	1024	44	99
01	1000	17	23
10	1008	11	5
11	1016	29	38

Cache

Memory

1000	17
1004	23
1008	11
1012	5
1016	29
1020	38
1024	44
1028	99
1032	97
1036	25
1040	1
1044	4

Miss Penalty and Rate

How much time do you lose on a Miss?

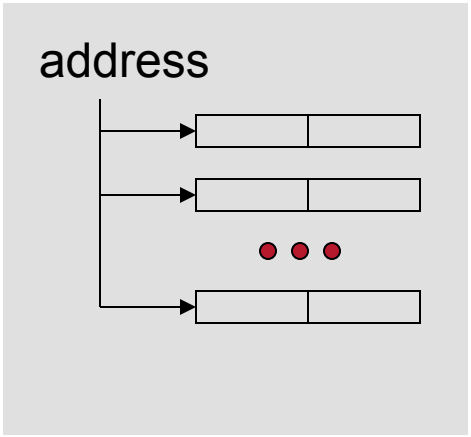
- MISS PENALTY is the time it takes to read the main memory if data was not found in the cache
  - 50 to 100 clock cycles is common
- MISS RATE is the fraction of accesses which MISS
- HIT RATE is the fraction of accesses which HIT
- MISS RATE + HIT RATE = 1

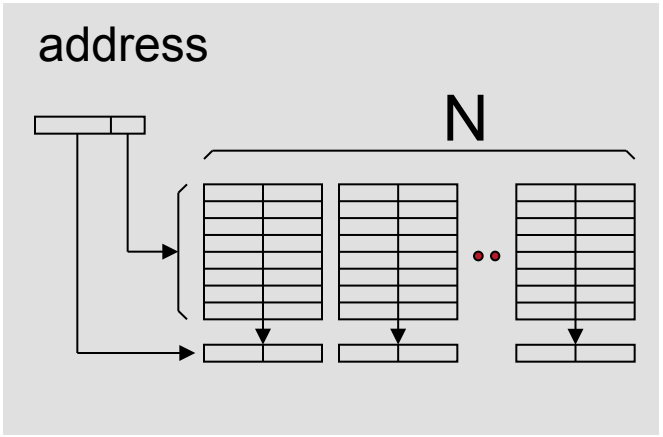
Example

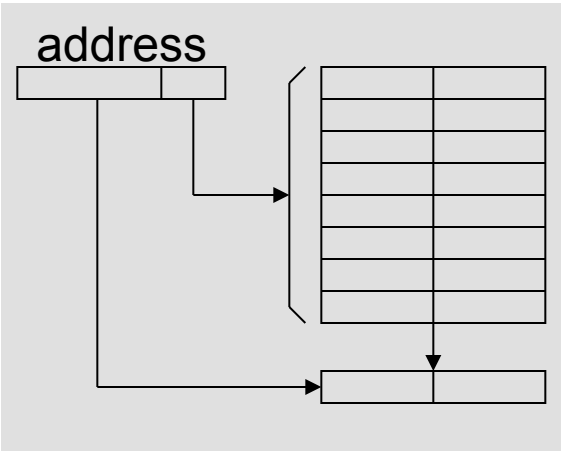
- Suppose a particular cache has a MISS PENALTY of 100 cycles and a HIT RATE of 95%. The CPI for load on HIT is 5 but on a MISS it is 105. What is the average CPI for load?
  - Average CPI = 5 * 0.95 + 105 * 0.05 = 10
- What if MISS PENALTY = 120 cycles?
  - Average CPI = 5 * 0.95 + 120 * 0.05 = 11

Continuum of Associativity

Fully associative | N-way set associative | Direct-mapped







Compares addr with ALL tags simultaneously location A can be stored in any cache line.

Compares addr with N tags simultaneously. Data can be stored in any of the N cache lines belonging to a "set" like N direct-mapped caches.

Compare addr with only ONE tag. Location A can be stored in exactly one cache line.

What happens on a MISS?

Finds one entry out of entire cache to evict

Finds one entry out of N in a particular row to evict

There is only one place it can go

Three Replacement Strategies

- When an entry has to be evicted, how to pick the victim?
  - LRU (Least-recently used)
    - replaces the item that has gone UNACCESSED the LONGEST
    - favors the most recently accessed data
  - FIFO/LRR (first-in, first-out/least-recently replaced)
    - replaces the OLDEST item in cache
    - favors recently loaded items over older STALE items
  - Random
    - replace some item at RANDOM
    - no favoritism – uniform distribution
    - no "pathological" reference streams causing worst-case results
    - use pseudo-random generator to get reproducible behavior

Handling WRITES

- Observation: Most (80+%) of memory accesses are reads, but writes are essential. How should we handle writes?
  - Two different policies:
    - WRITE-THROUGH: CPU writes are cached, but also written to main memory (stalling the CPU until write is completed). Memory always holds the latest values.
    - WRITE-BACK: CPU writes are cached, but not immediately written to main memory. Main memory contents can become "stale". Only when a value has to be evicted from the cache, and only if it had been modified (i.e., is "dirty"), only then it is written to main memory.
  - Pros and Cons?
    - WRITE-BACK typically has higher performance
    - WRITE-THROUGH typically causes fewer consistency problems

Memory Hierarchy Summary

- Give the illusion of fast, big memory
  - small fast cache makes the entire memory appear fast
  - large main memory provides ample storage
  - even larger hard drive provides huge virtual memory (TB)
- Various design decisions affect caching
  - total cache size, line size, replacement strategy, write policy
- Performance
  - Put your money on bigger caches, them bigger main memory.
  - Don't put your money on CPU clock speed (GHz) because memory is usually the culprit!