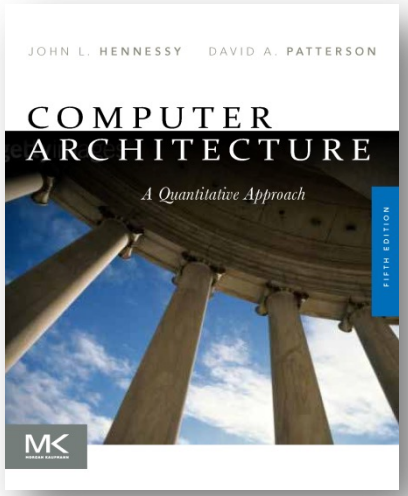
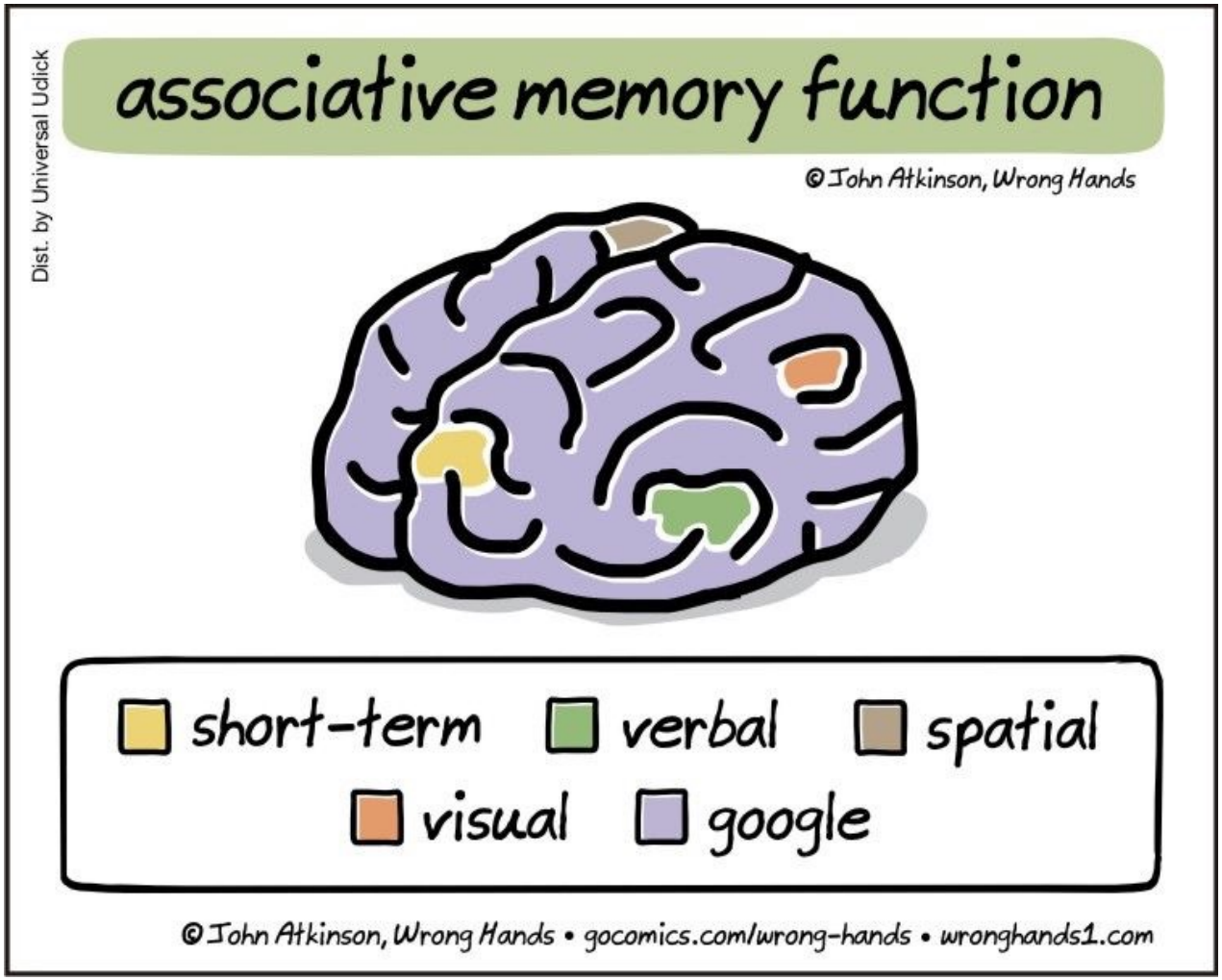


Memory Hierarchy Design and so on.

@carlosjaimebh

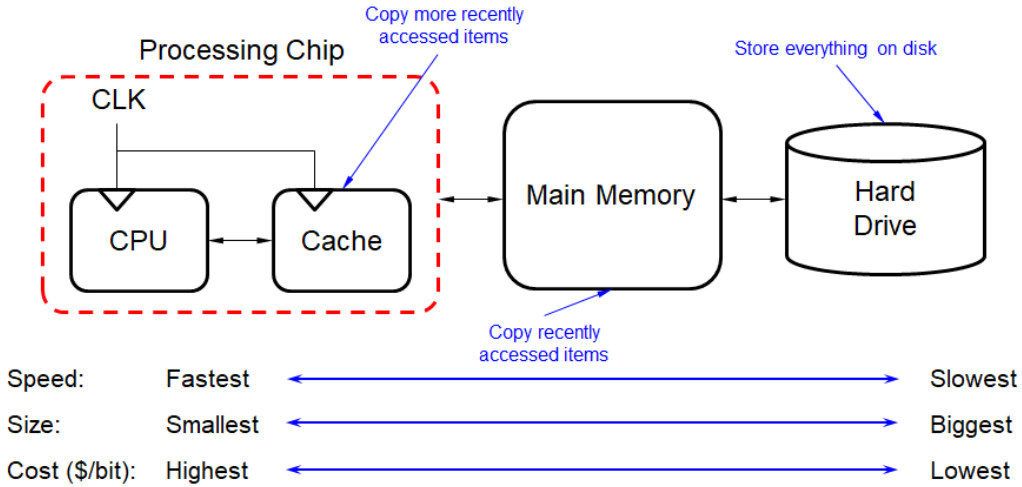
Memory is important to compute



Introduction

- Programmers want unlimited amounts of memory with low latency
- Fast memory technology is more expensive per bit than slower memory
- Solution: organize memory system into a hierarchy
  - Entire addressable memory space available in largest, slowest memory
  - Incrementally smaller and faster memories, each containing a subset of the memory below it, proceed in steps up toward the processor
- Temporal and spatial locality ensure that nearly all references can be found in smaller memories
  - Gives the allusion of a large, fast memory being presented to the processor

Principle of Locality

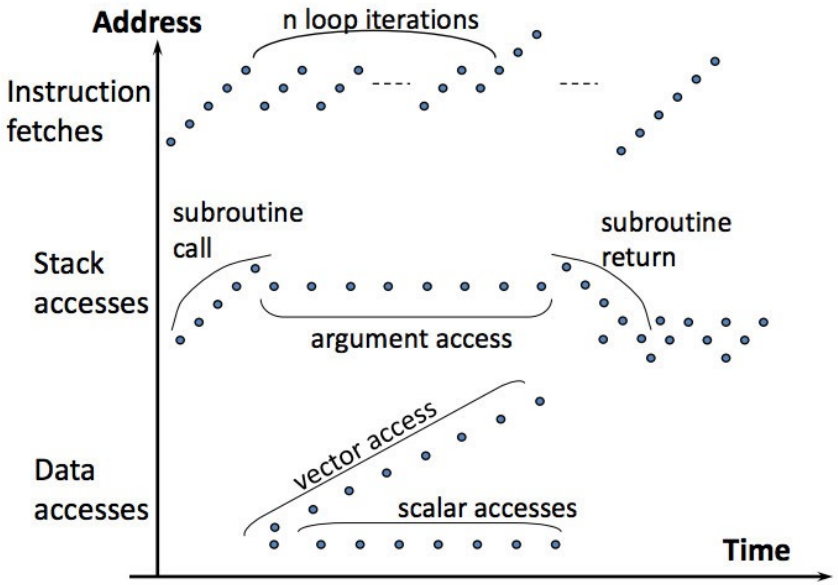


Or principle of reference is the tendency of a processor to access the same set of memory locations repetitively over a short period. Then, programs tend to reuse data and instructions they have used recently. (in other words, most programs do not access all code or data uniformly).

- Temporal Locality states that recently accessed items are likely to be accessed in the near future.
- Spatial Locality refers to using data elements within relatively close storage locations.
  - Sequential locality, a particular case of spatial locality, occurs when data elements are arranged and accessed linearly, such as traversing the elements in a one-dimensional array.

Relevance of Locality (1/2)

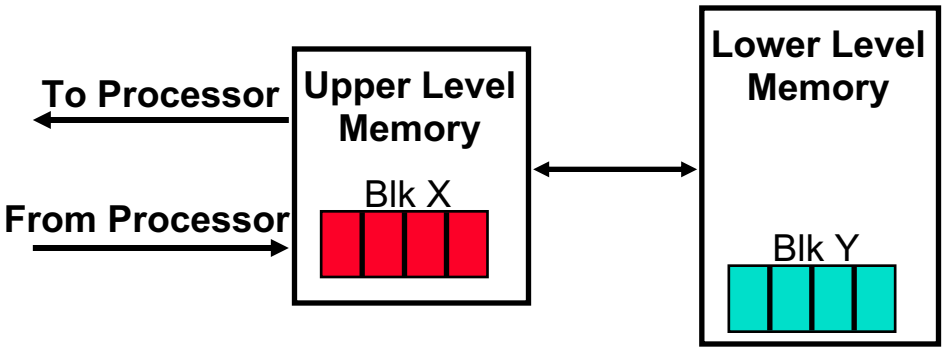
- **Predictability**: Locality is merely one type of predictable behavior in computer systems.
- **Structure of the program**: Locality occurs often because of the way in which computer programs are created, for handling decidable problems. Generally, related data is stored in nearby locations in storage. One common pattern in computing involves the processing of several items, one at a time.



Relevance of Locality (2)

- **Linear data structures**: Locality often occurs because code contains loops that tend to reference arrays or other data structures by indices. Sequential locality, a special case of spatial locality, occurs when relevant data elements are arranged and accessed linearly.
- **Efficiency of memory hierarchy use**: Although random-access memory presents the programmer with the ability to read or write anywhere at any time, in practice latency and throughput are affected by the efficiency of the cache, which is improved by increasing the locality of reference. Poor locality of reference results in cache thrashing and cache pollution and to avoid it, data elements with poor locality can be bypassed from cache.

General Usage of Locality



- Increasing the locality of references (generally on the software side)
- Exploiting the locality of references: Generally achieved on the hardware side, hierarchical storage hardware can capitalize temporal and spatial locality.

Example: Matrix Multiplication (1/2)

A common example is matrix multiplication:

```
for i in 0..n
  for j in 0..m
    for k in 0..p
      C[i][j] = C[i][j] + A[i][k] * B[k][j];
```

By switching the looping order for `j` and `k`, the speedup in large matrix multiplications becomes dramatic, at least for languages that put contiguous array elements in the last dimension. This will not change the mathematical result, but it improves efficiency. In this case, "large" means, approximately, more than 100,000 elements in each matrix, or enough addressable memory such that the matrices will not fit in L1 and L2 caches.

```
for i in 0..n
  for k in 0..p
    for j in 0..m
      C[i][j] = C[i][j] + A[i][k] * B[k][j];
```

The reason for this speedup is that in the first case, the reads of `A[i][k]` are in cache (since the `k` index is the contiguous, last dimension), but `B[k][j]` is not, so there is a cache miss penalty on `B[k][j]`. `C[i][j]` is irrelevant, because it can be hoisted out of the inner loop -- the loop variable there is `k`.

```
for i in 0..n
  for j in 0..m
    temp = C[i][j]
    for k in 0..p
      temp = temp + A[i][k] * B[k][j];
    C[i][j] = temp
```

In the second case, the reads and writes of `C[i][j]` are both in cache, the reads of `B[k][j]` are in cache, and the read of `A[i][k]` can be hoisted out of the inner loop.

```
for i in 0..n
  for k in 0..p
    temp = A[i][k]
    for j in 0..m
      C[i][j] = C[i][j] + temp * B[k][j];
```

Thus, the second example has no cache miss penalty in the inner loop while the first example has a cache penalty.

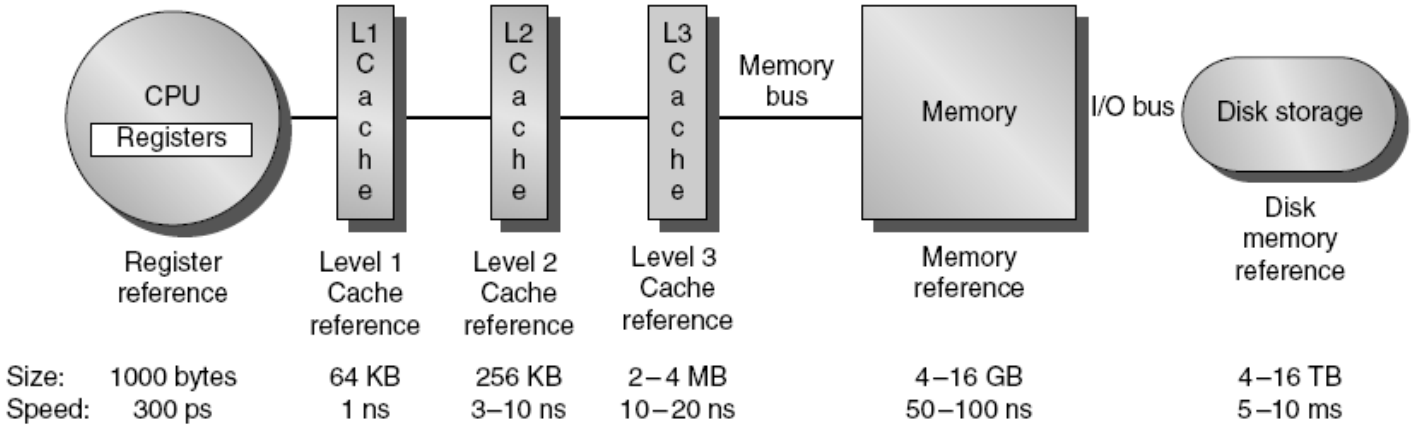
Example: Matrix Multiplication (2/2)

Temporal locality can also be improved in the above example by using a technique called blocking. The larger matrix can be divided into evenly sized sub-matrices, so that the smaller blocks can be referenced (multiplied) several times while in memory. Note that this example works for square matrices of dimensions SIZE x SIZE, but it can easily be extended for arbitrary matrices by substituting SIZE_I, SIZE_J and SIZE_K where appropriate.

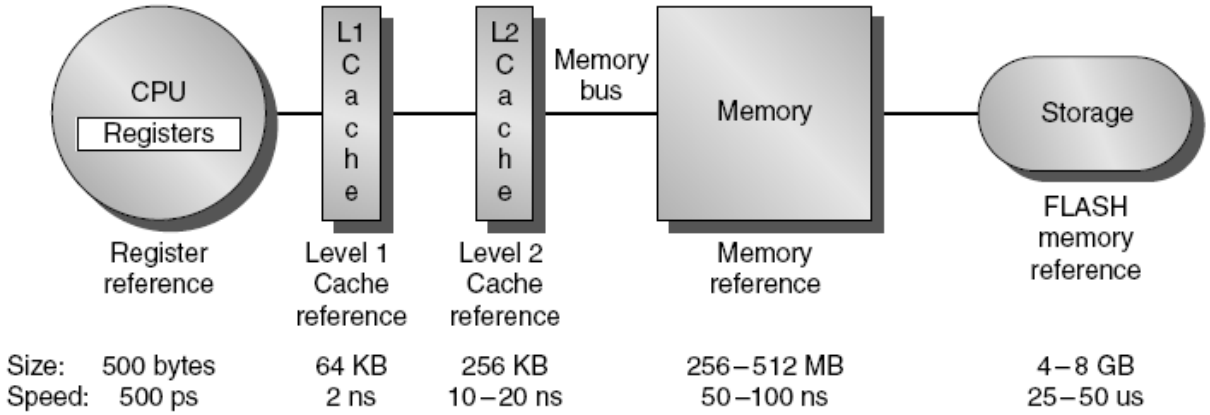
```
1  for (ii = 0; ii < SIZE; ii += BLOCK_SIZE)
2    for (kk = 0; kk < SIZE; kk += BLOCK_SIZE)
3      for (jj = 0; jj < SIZE; jj += BLOCK_SIZE)
4        maxi = min(ii + BLOCK_SIZE, SIZE);
5        for (i = ii; i < maxi; i++)
6          maxk = min(kk + BLOCK_SIZE, SIZE);
7          for (k = kk; k < maxk; k++)
8            maxj = min(jj + BLOCK_SIZE, SIZE);
9            for (j = jj; j < maxj; j++)
10             C[i][j] = C[i][j] + A[i][k] * B[k][j];
```

The temporal locality of the above solution is provided because a block can be used several times before moving on, so that it is moved in and out of memory less often. Spatial locality is improved because elements with consecutive memory addresses tend to be pulled up the memory hierarchy together.

Memory Hierarchy

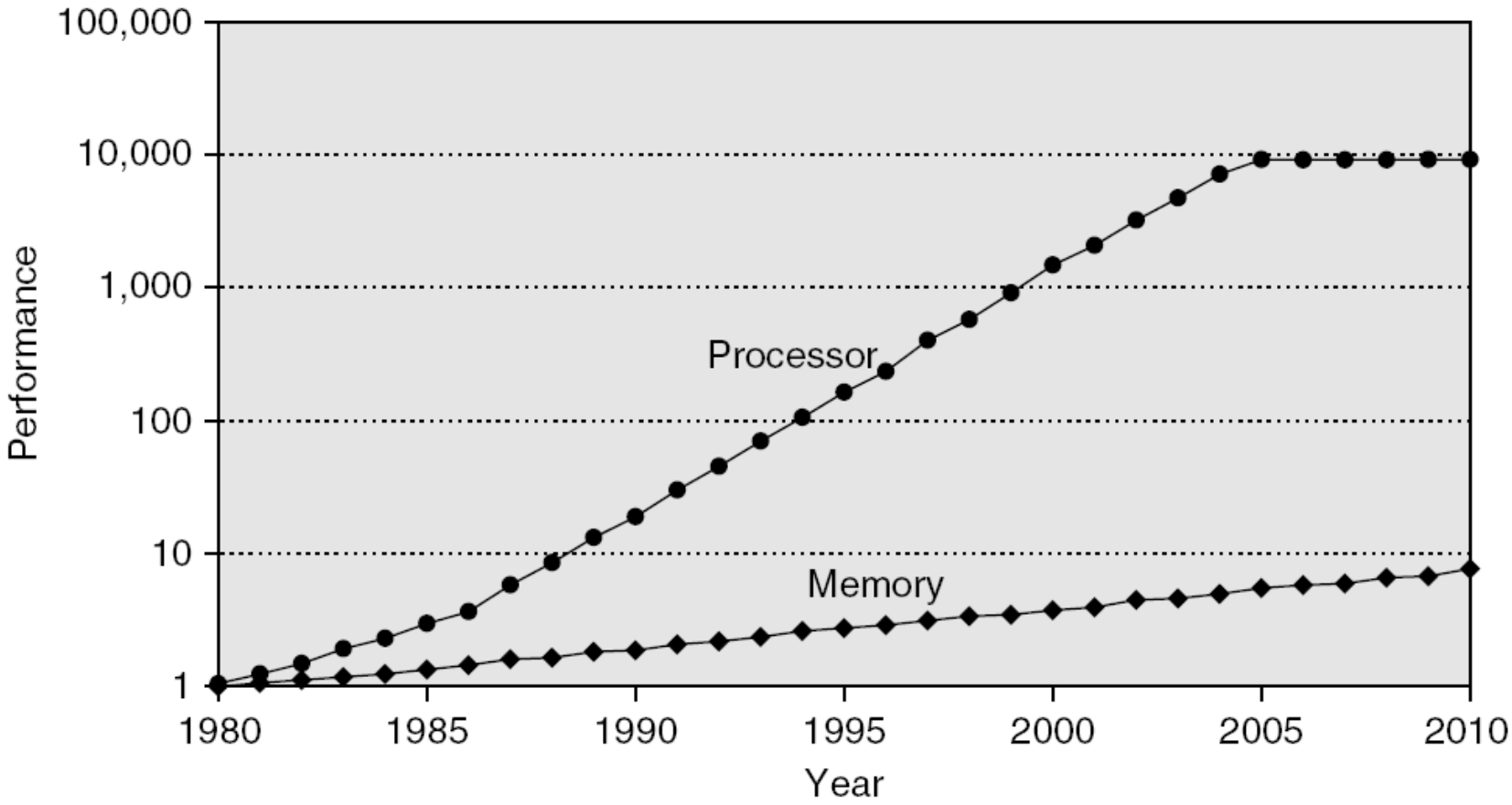


(a) Memory hierarchy for server



(b) Memory hierarchy for a personal mobile device

Memory Performance Gap



Memory Hierarchy Design

- Memory hierarchy design becomes more crucial with recent multi-core processors:
  - Aggregate peak bandwidth grows with # cores:
    - Intel Core i7 can generate two references per core per clock
    - Four cores and 3.2 GHz clock
      - 25.6 billion 64-bit data references/second +
      - 12.8 billion 128-bit instruction references
      - = 409.6 GB/s!
    - DRAM bandwidth is only 6% of this (25 GB/s)
    - Requires:
      - Multi-port, pipelined caches
      - Two levels of cache per core
      - Shared third-level cache on chip

Performance and Power

- High-end microprocessors have >10 MB on-chip cache
  - Consumes large amount of area and power budget

Memory Hierarchy Basics

- When a word is not found in the cache, a *miss* occurs:
  - Fetch word from lower level in hierarchy, requiring a higher latency reference
  - Lower level may be another cache or the main memory
  - Also fetch the other words contained within the *block*
    - Takes advantage of spatial locality
  - Place block into cache in any location within its *set*, determined by address
    - block address MOD number of sets

Memory Hierarchy Basics

- *n* sets => *n-way set associative*
  - *Direct-mapped cache* => one block per set
  - *Fully associative* => one set

- Writing to cache: two strategies
  - *Write-through*
    - Immediately update lower levels of hierarchy
  - *Write-back*
    - Only update lower levels of hierarchy when an updated block is replaced
  - Both strategies use *write buffer* to make writes asynchronous

Memory Hierarchy Basics

- Miss rate
  - Fraction of cache access that result in a miss
- Causes of misses
  - Compulsory
    - First reference to a block
  - Capacity
    - Blocks discarded and later retrieved
  - Conflict
    - Program makes repeated references to multiple addresses from different blocks that map to the same location in the cache

Memory Hierarchy Basics

$$\frac{\text{Misses}}{\text{Instruction}} = \frac{\text{Miss rate} \times \text{Memory accesses}}{\text{Instruction count}} = \text{Miss rate} \times \frac{\text{Memory accesses}}{\text{Instruction}}$$

$$\text{Average memory access time} = \text{Hit time} + \text{Miss rate} \times \text{Miss penalty}$$

- Note that speculative and multithreaded processors may execute other instructions during a miss
  - Reduces performance impact of misses

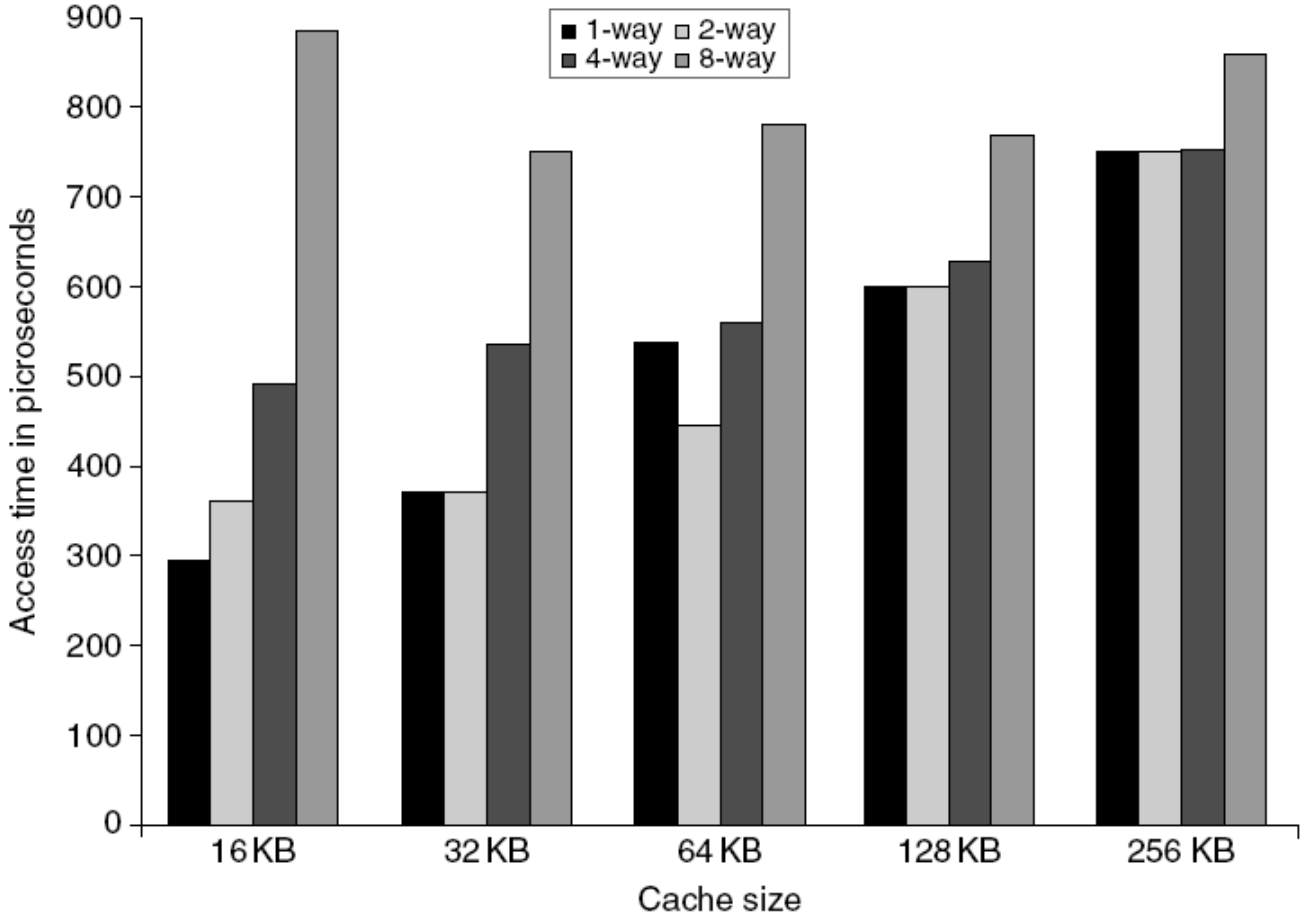
Memory Hierarchy Basics

- Six basic cache optimizations:
  - Larger block size
    - Reduces compulsory misses
    - Increases capacity and conflict misses, increases miss penalty
  - Larger total cache capacity to reduce miss rate
    - Increases hit time, increases power consumption
  - Higher associativity
    - Reduces conflict misses
    - Increases hit time, increases power consumption
  - Higher number of cache levels
    - Reduces overall memory access time
  - Giving priority to read misses over writes
    - Reduces miss penalty
  - Avoiding address translation in cache indexing
    - Reduces hit time

Ten Advanced Optimizations

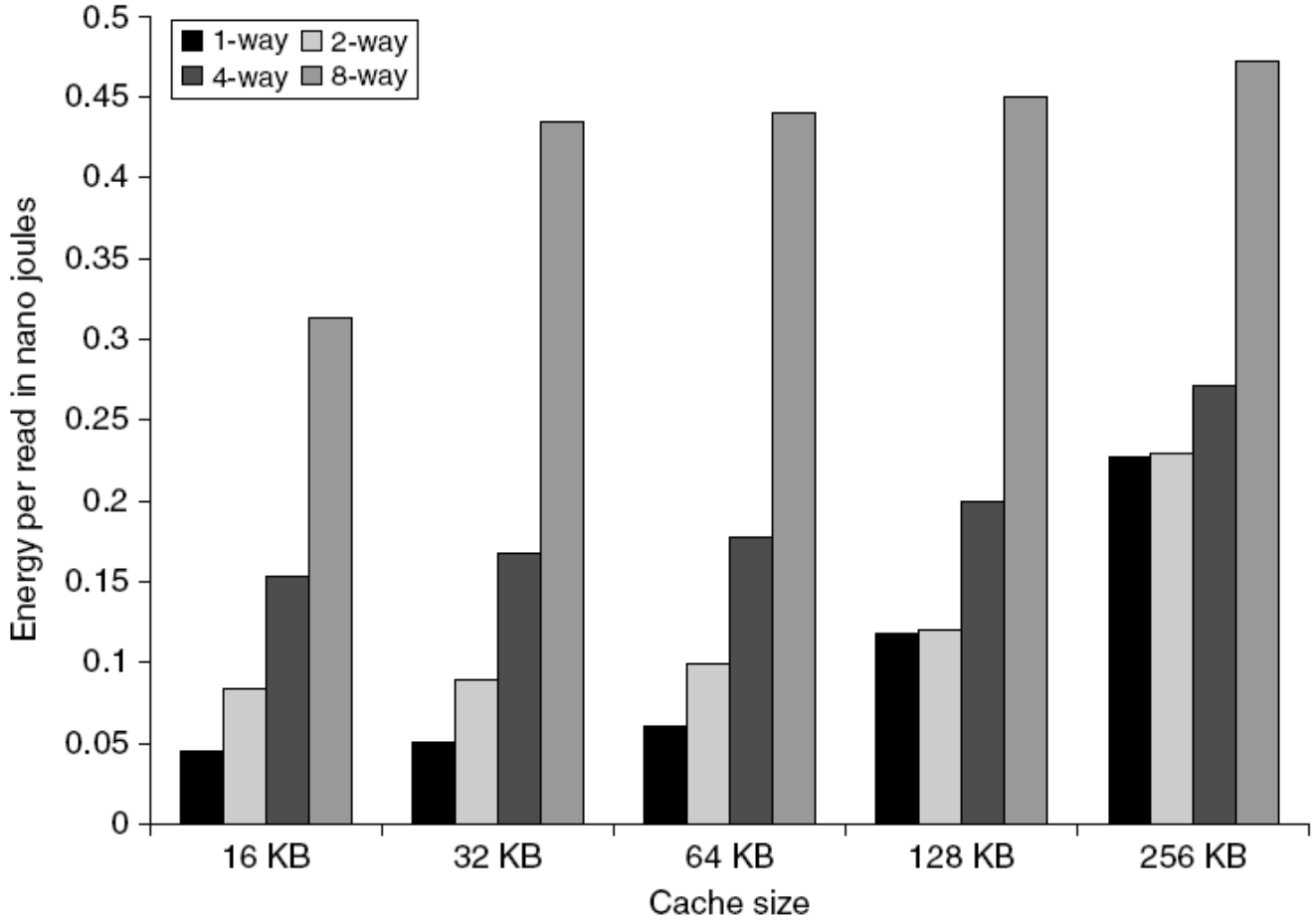
- Small and simple first level caches
  - Critical timing path:
    - addressing tag memory, then
    - comparing tags, then
    - selecting correct set
  - Direct-mapped caches can overlap tag compare and transmission of data
  - Lower associativity reduces power because fewer cache lines are accessed

L1 Size and Associativity



Access time vs. size and associativity

L1 Size and Associativity



Energy per read vs. size and associativity

Way Prediction

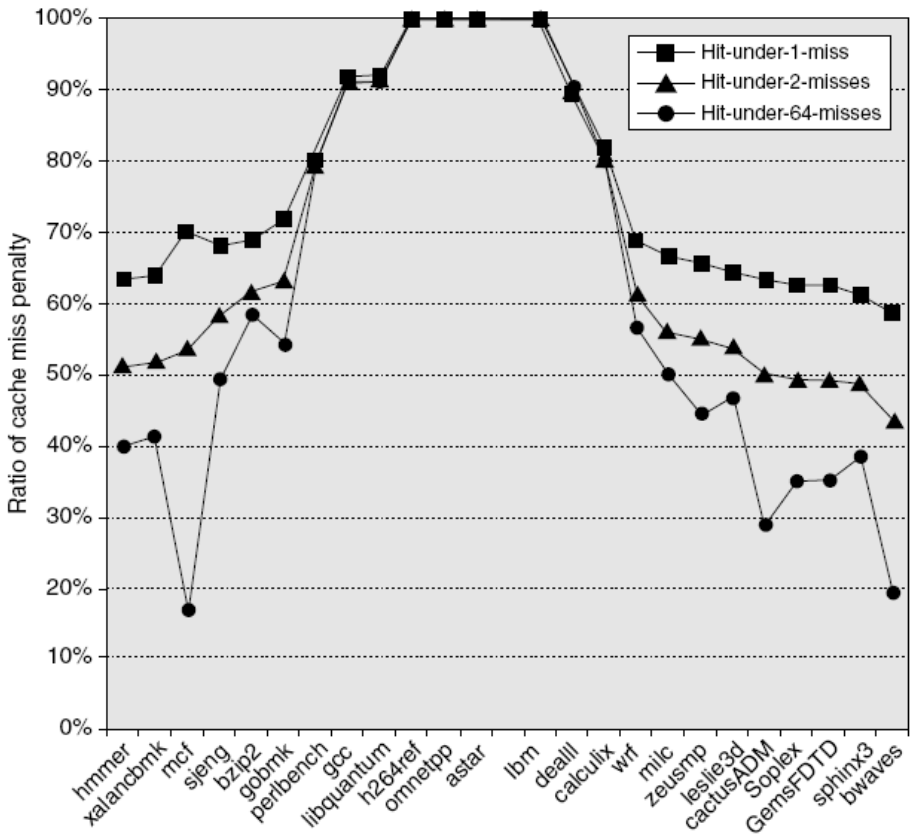
- To improve hit time, predict the way to pre-set mux
  - Mis-prediction gives longer hit time
  - Prediction accuracy
    - > 90% for two-way
    - > 80% for four-way
    - I-cache has better accuracy than D-cache
  - First used on MIPS R10000 in mid-90s
  - Used on ARM Cortex-A8
- Extend to predict block as well
  - “Way selection”
  - Increases mis-prediction penalty

Pipelining Cache

- Pipeline cache access to improve bandwidth
  - Examples:
    - Pentium: 1 cycle
    - Pentium Pro – Pentium III: 2 cycles
    - Pentium 4 – Core i7: 4 cycles
- Increases branch mis-prediction penalty
- Makes it easier to increase associativity

Nonblocking Caches

- Allow hits before previous misses complete
  - "Hit under miss"
  - "Hit under multiple miss"
- L2 must support this
- In general, processors can hide L1 miss penalty but not L2 miss penalty



Multibanked Caches

- Organize cache as independent banks to support simultaneous access
  - ARM Cortex-A8 supports 1-4 banks for L2
  - Intel i7 supports 4 banks for L1 and 8 banks for L2

- Interleave banks according to block address

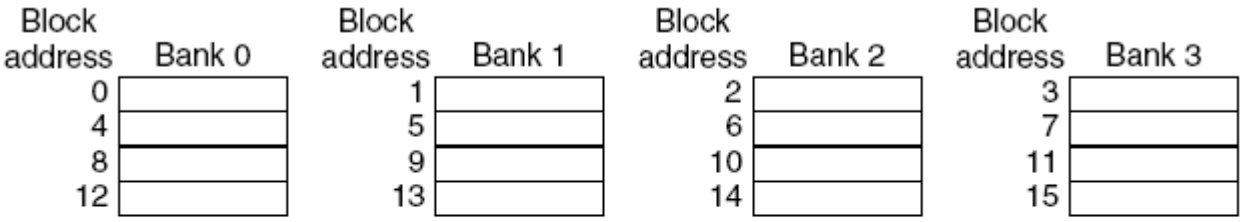


Figure 2.6 Four-way interleaved cache banks using block addressing. Assuming 64 bytes per blocks, each of these addresses would be multiplied by 64 to get byte addressing.

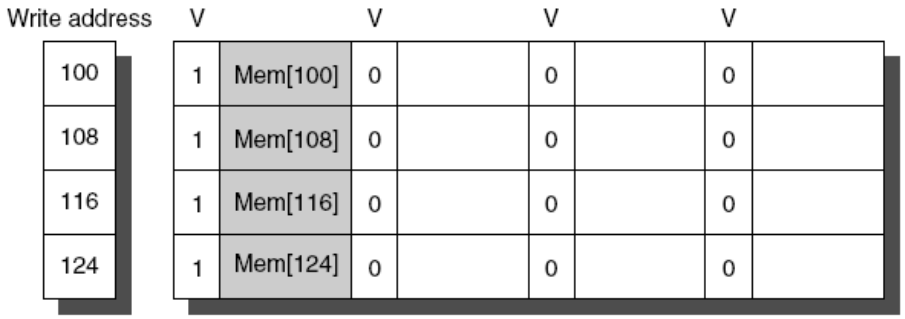
Critical Word First, Early Restart

- Critical word first
  - Request missed word from memory first
  - Send it to the processor as soon as it arrives
- Early restart
  - Request words in normal order
  - Send missed work to the processor as soon as it arrives

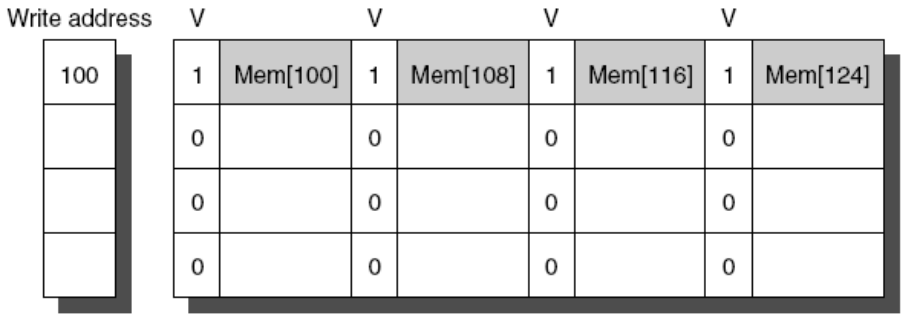
- Effectiveness of these strategies depends on block size and likelihood of another access to the portion of the block that has not yet been fetched

Merging Write Buffer

- When storing to a block that is already pending in the write buffer, update write buffer
- Reduces stalls due to full write buffer
- Do not apply to I/O addresses



No write buffering

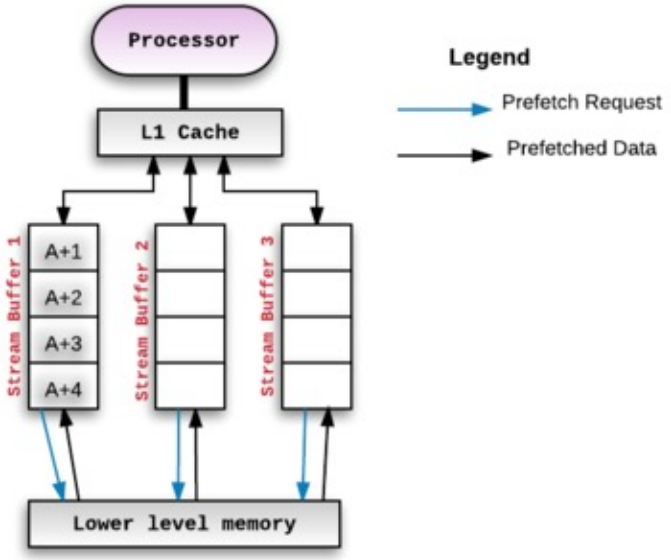


Write buffering

Compiler Optimizations

- Loop Interchange
  - Swap nested loops to access memory in sequential order
- Blocking
  - Instead of accessing entire rows or columns, subdivide matrices into blocks
  - Requires more memory accesses but improves locality of accesses

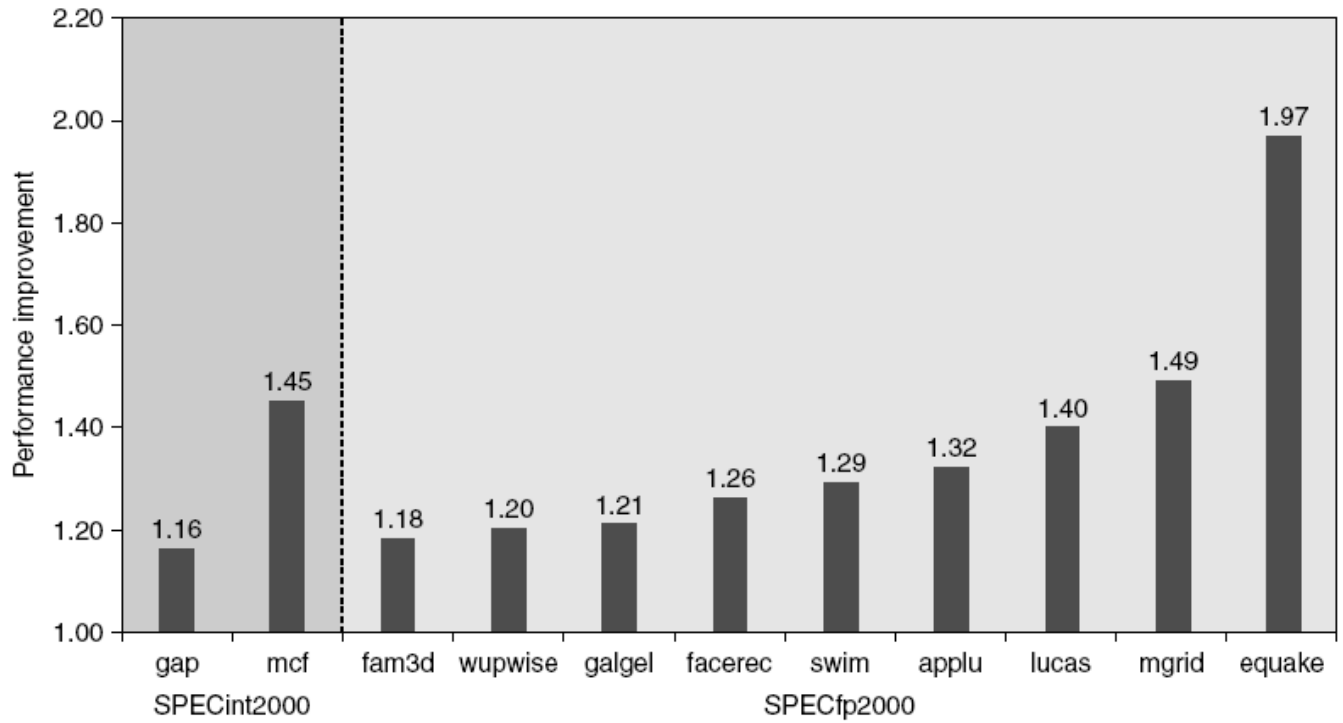
Prefetching



Cache prefetching is a technique used by computer processors to boost execution performance by fetching instructions or data from their original storage in slower memory to a faster local memory before it is actually needed (hence the term 'prefetch')

Hardware Prefetching

- Fetch two blocks on miss (include next sequential block)



Pentium 4 Pre-fetching

Compiler Prefetching

- Insert prefetch instructions before data is needed
- Non-faulting: prefetch doesn't cause exceptions

- Register prefetch
  - Loads data into register
- Cache prefetch
  - Loads data into cache

- Combine with loop unrolling and software pipelining

Summary

Technique	Hit time	Band-width	Miss penalty	Miss rate	Power consumption	Hardware cost/ complexity	Comment
Small and simple caches	+			–	+	0	Trivial; widely used
Way-predicting caches	+				+	1	Used in Pentium 4
Pipelined cache access	–	+				1	Widely used
Nonblocking caches		+	+			3	Widely used
Banked caches		+			+	1	Used in L2 of both i7 and Cortex-A8
Critical word first and early restart			+			2	Widely used
Merging write buffer			+			1	Widely used with write through
Compiler techniques to reduce cache misses				+		0	Software is a challenge, but many compilers handle common linear algebra calculations
Hardware prefetching of instructions and data			+	+	–	2 instr., 3 data	Most provide prefetch instructions; modern high-end processors also automatically prefetch in hardware.
Compiler-controlled prefetching			+	+		3	Needs nonblocking cache; possible instruction overhead; in many CPUs

Figure 2.11 Summary of 10 advanced cache optimizations showing impact on cache performance, power consumption, and complexity. Although generally a technique helps only one factor, prefetching can reduce misses if done sufficiently early; if not, it can reduce miss penalty. + means that the technique improves the factor, – means it hurts that factor, and blank means it has no impact. The complexity measure is subjective, with 0 being the easiest and 3 being a challenge.

Memory Technology

- Performance metrics
  - Latency is concern of cache
  - Bandwidth is concern of multiprocessors and I/O
  - Access time
    - Time between read request and when desired word arrives
  - Cycle time
    - Minimum time between unrelated requests to memory
- DRAM used for main memory, SRAM used for cache

Memory Technology

- SRAM
  - Requires low power to retain bit
  - Requires 6 transistors/bit
- DRAM
  - Must be re-written after being read
  - Must also be periodically refeshed
    - Every ~ 8 ms
    - Each row can be refreshed simultaneously
  - One transistor/bit
  - Address lines are multiplexed:
    - Upper half of address: row access strobe (RAS)
    - Lower half of address: column access strobe (CAS)

Memory Technology

- Amdahl:
  - Memory capacity should grow linearly with processor speed
  - Unfortunately, memory capacity and speed has not kept pace with processors
- Some optimizations:
  - Multiple accesses to same row
  - Synchronous DRAM
    - Added clock to DRAM interface
    - Burst mode with critical word first
  - Wider interfaces
  - Double data rate (DDR)
  - Multiple banks on each DRAM device

Memory Optimizations

Production year	Chip size	DRAM Type	Row access strobe (RAS)		Column access strobe (CAS)/ data transfer time (ns)	Cycle time (ns)
			Slowest DRAM (ns)	Fastest DRAM (ns)		
1980	64K bit	DRAM	180	150	75	250
1983	256K bit	DRAM	150	120	50	220
1986	1M bit	DRAM	120	100	25	190
1989	4M bit	DRAM	100	80	20	165
1992	16M bit	DRAM	80	60	15	120
1996	64M bit	SDRAM	70	50	12	110
1998	128M bit	SDRAM	70	50	10	100
2000	256M bit	DDR1	65	45	7	90
2002	512M bit	DDR1	60	40	5	80
2004	1G bit	DDR2	55	35	5	70
2006	2G bit	DDR2	50	30	2.5	60
2010	4G bit	DDR3	36	28	1	37
2012	8G bit	DDR3	30	24	0.5	31

Figure 2.13 Times of fast and slow DRAMs vary with each generation. (Cycle time is defined on page 95.) Performance improvement of row access time is about 5% per year. The improvement by a factor of 2 in column access in 1986 accompanied the switch from NMOS DRAMs to CMOS DRAMs. The introduction of various burst transfer modes in the mid-1990s and SDRAMs in the late 1990s has significantly complicated the calculation of access time for blocks of data; we discuss this later in this section when we talk about SDRAM access time and power. The DDR4 designs are due for introduction in mid- to late 2012. We discuss these various forms of DRAMs in the next few pages.

Memory Optimizations

Standard	Clock rate (MHz)	M transfers per second	DRAM name	MB/sec /DIMM	DIMM name
DDR	133	266	DDR266	2128	PC2100
DDR	150	300	DDR300	2400	PC2400
DDR	200	400	DDR400	3200	PC3200
DDR2	266	533	DDR2-533	4264	PC4300
DDR2	333	667	DDR2-667	5336	PC5300
DDR2	400	800	DDR2-800	6400	PC6400
DDR3	533	1066	DDR3-1066	8528	PC8500
DDR3	666	1333	DDR3-1333	10,664	PC10700
DDR3	800	1600	DDR3-1600	12,800	PC12800
DDR4	1066–1600	2133–3200	DDR4-3200	17,056–25,600	PC25600

Figure 2.14 Clock rates, bandwidth, and names of DDR DRAMS and DIMMs in 2010. Note the numerical relationship between the columns. The third column is twice the second, and the fourth uses the number from the third column in the name of the DRAM chip. The fifth column is eight times the third column, and a rounded version of this number is used in the name of the DIMM. Although not shown in this figure, DDRs also specify latency in clock cycles as four numbers, which are specified by the DDR standard. For example, DDR3-2000 CL 9 has latencies of 9-9-9-28. What does this mean? With a 1 ns clock (clock cycle is one-half the transfer rate), this indicate 9 ns for row to columns address (RAS time), 9 ns for column access to data (CAS time), and a minimum read time of 28 ns. Closing the row takes 9 ns for precharge but happens only when the reads from that row are finished. In burst mode, transfers occur on every clock on both edges, when the first RAS and CAS times have elapsed. Furthermore, the precharge in not needed until the entire row is read. DDR4 will be produced in 2012 and is expected to reach clock rates of 1600 MHz in 2014, when DDR5 is expected to take over. The exercises explore these details further.

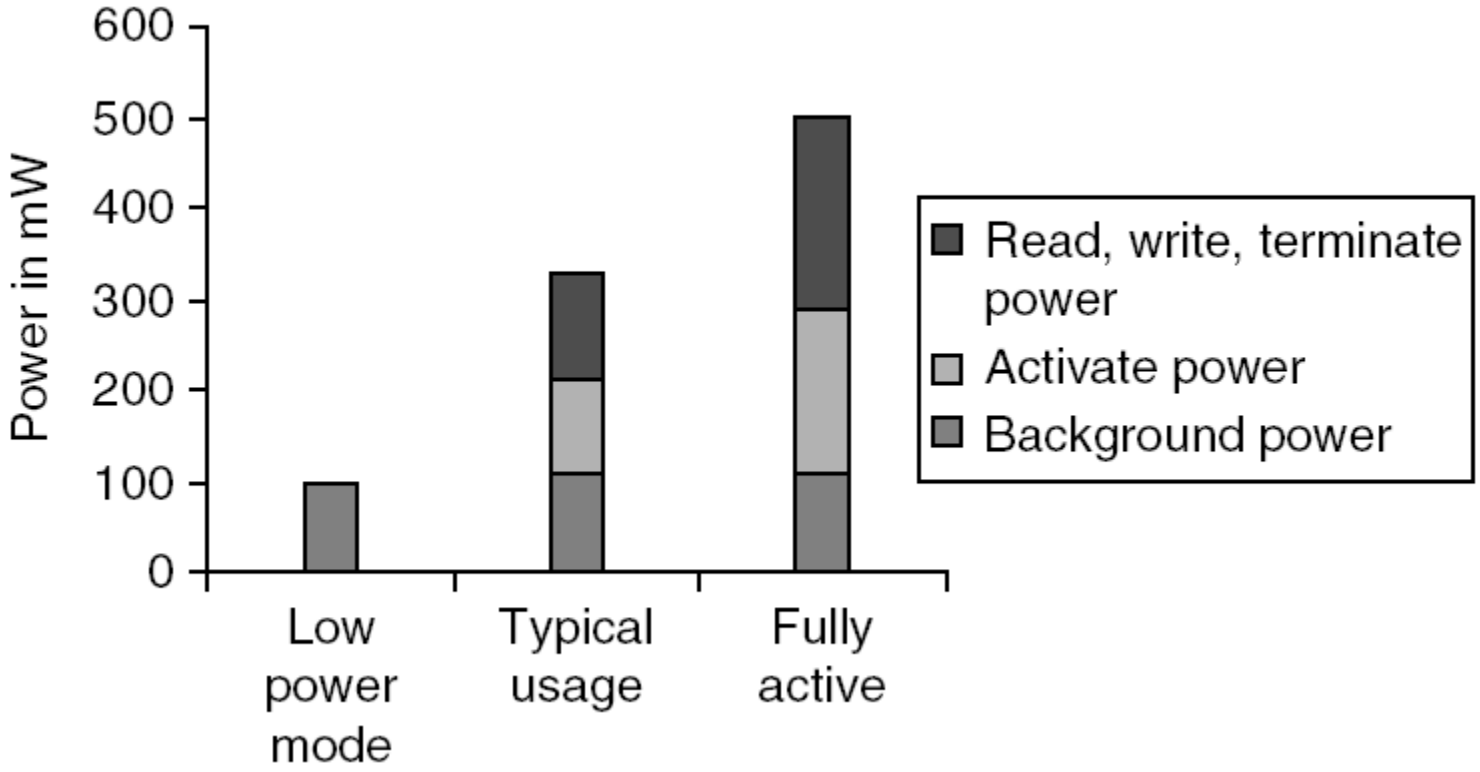
Memory Optimizations

- DDR:
  - DDR2
    - Lower power (2.5 V -> 1.8 V)
    - Higher clock rates (266 MHz, 333 MHz, 400 MHz)
  - DDR3
    - 1.5 V
    - 800 MHz
  - DDR4
    - 1-1.2 V
    - 1600 MHz
- GDDR5 is graphics memory based on DDR3

Memory Optimizations

- Graphics memory:
  - Achieve 2-5 X bandwidth per DRAM vs. DDR3
    - Wider interfaces (32 vs. 16 bit)
    - Higher clock rate
      - Possible because they are attached via soldering instead of socketted DIMM modules
- Reducing power in SDRAMs:
  - Lower voltage
  - Low power mode (ignores clock, continues to refresh)

Memory Power Consumption



Flash Memory

- Type of EEPROM
- Must be erased (in blocks) before being overwritten
- Non volatile
- Limited number of write cycles
- Cheaper than SDRAM, more expensive than disk
- Slower than SRAM, faster than disk

Memory Dependability

- Memory is susceptible to cosmic rays
- *Soft errors*: dynamic errors
  - Detected and fixed by error correcting codes (ECC)
- *Hard errors*: permanent errors
  - Use sparse rows to replace defective rows

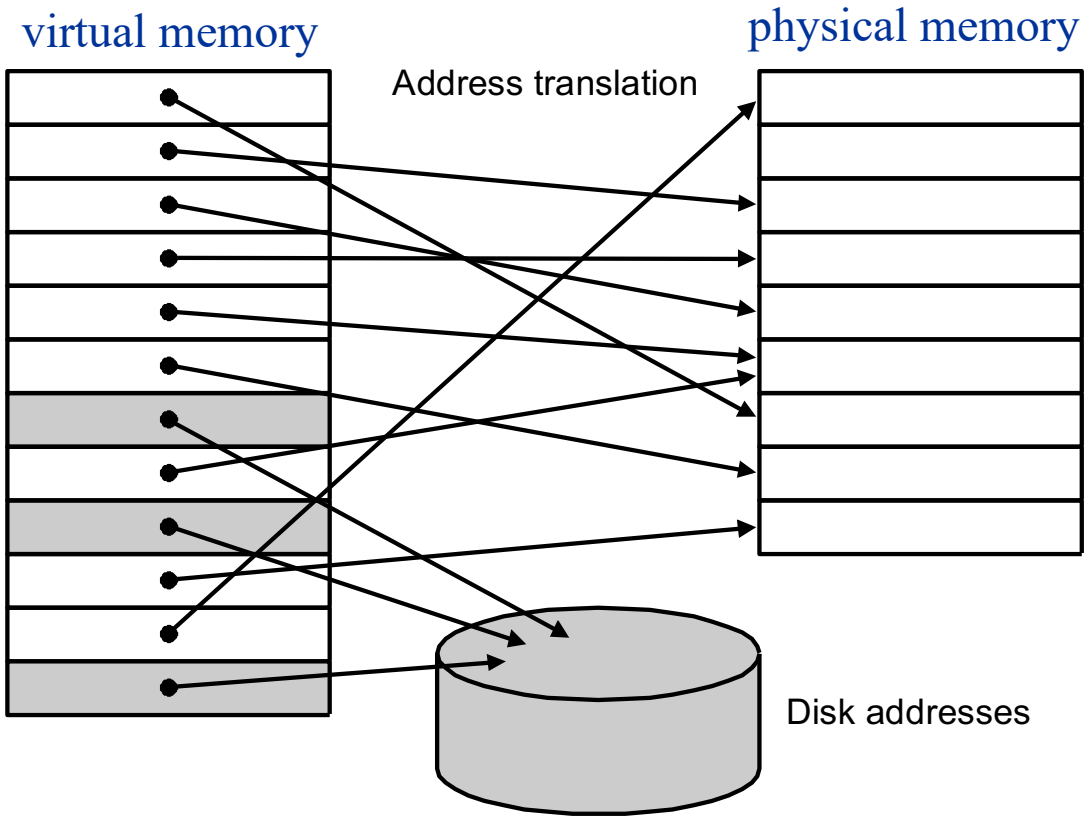
- Chipkill: a RAID-like error recovery technique

Virtual Memory

- Protection via virtual memory
  - Keeps processes in their own memory space
- Role of architecture:
  - Provide user mode and supervisor mode
  - Protect certain aspects of CPU state
  - Provide mechanisms for switching between user mode and supervisor mode
  - Provide mechanisms to limit memory accesses
  - Provide TLB to translate addresses

Virtual Memory at Glance

- Main memory can act as a cache for the secondary storage (disk)



Virtual Machines

- Supports isolation and security
- Sharing a computer among many unrelated users
- Enabled by raw speed of processors, making the overhead more acceptable

- Allows different ISAs and operating systems to be presented to user programs
  - "System Virtual Machines"
  - SVM software is called "virtual machine monitor" or "hypervisor"
  - Individual virtual machines run under the monitor are called "guest VMs"

Impact of VMs on Virtual Memory

- Each guest OS maintains its own set of page tables
  - VMM adds a level of memory between physical and virtual memory called “real memory”
  - VMM maintains shadow page table that maps guest virtual addresses to physical addresses
    - Requires VMM to detect guest’s changes to its own page table
    - Occurs naturally if accessing the page table pointer is a privileged operation

Memory Organization

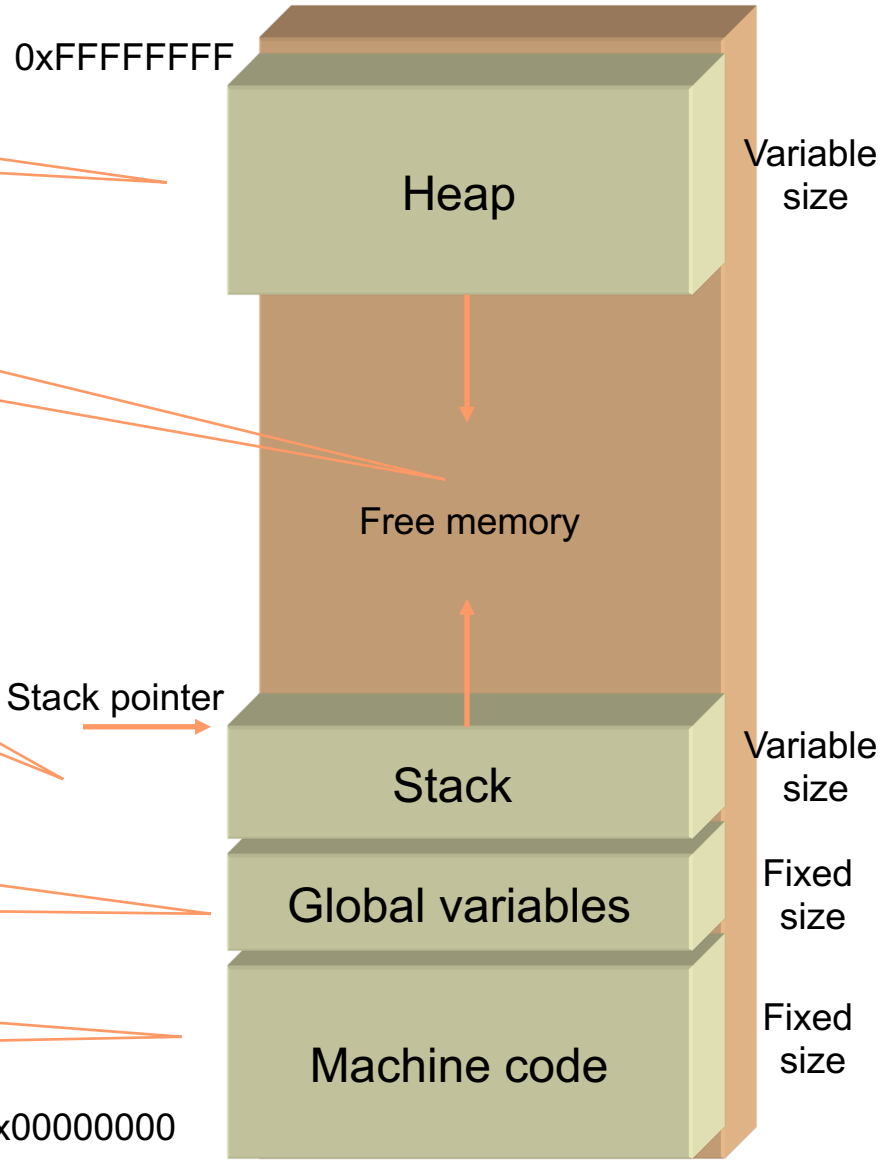
The memory that is reserved by the memory manager

If the heap and the stack collide, we're out of memory

The local variables in the routines. With each routine call, a new set of variables if put in the stack.

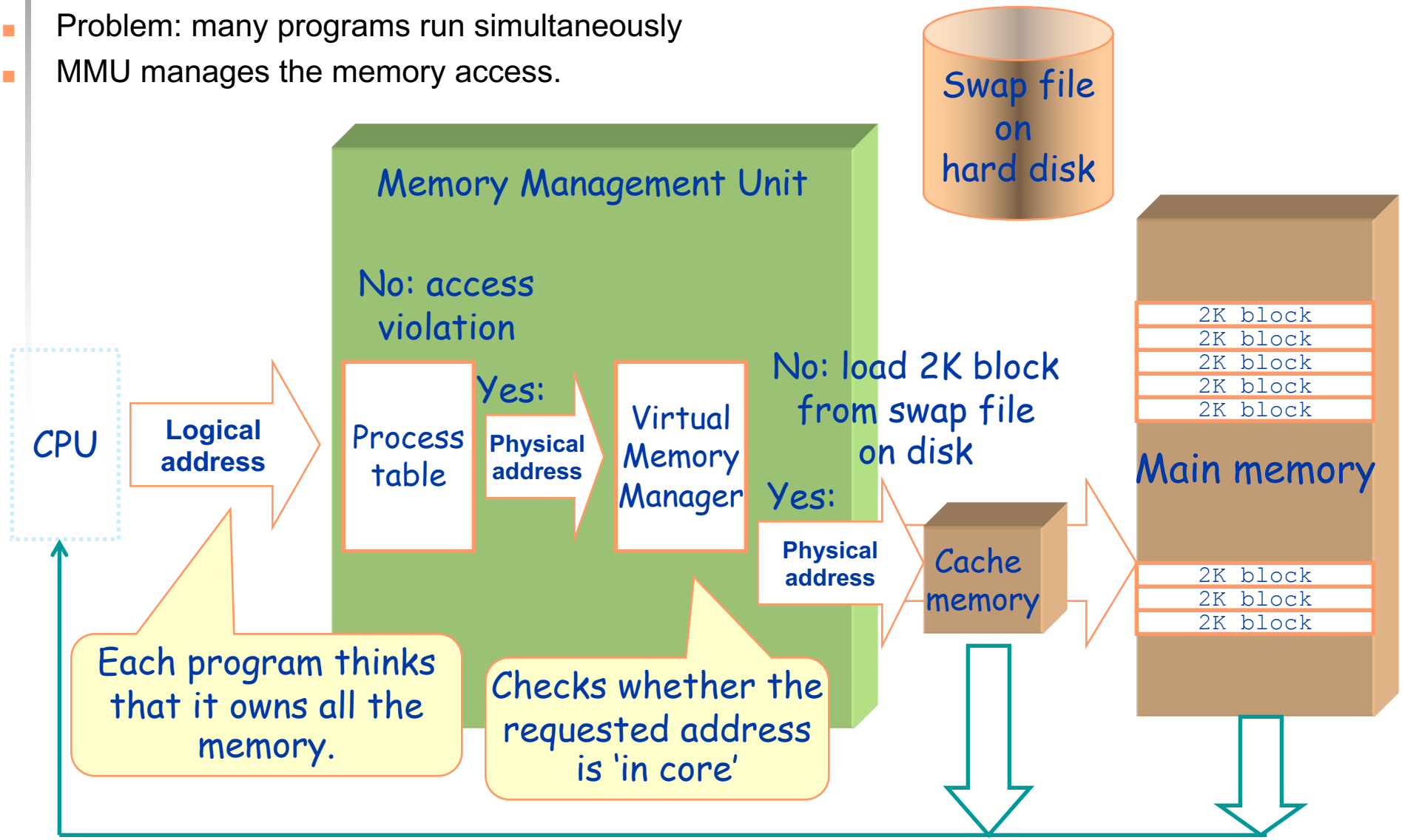
Before the first line of the program is run, all global variables and constants are initialized.

The program itself: a set of machine instructions. This is in the .exe



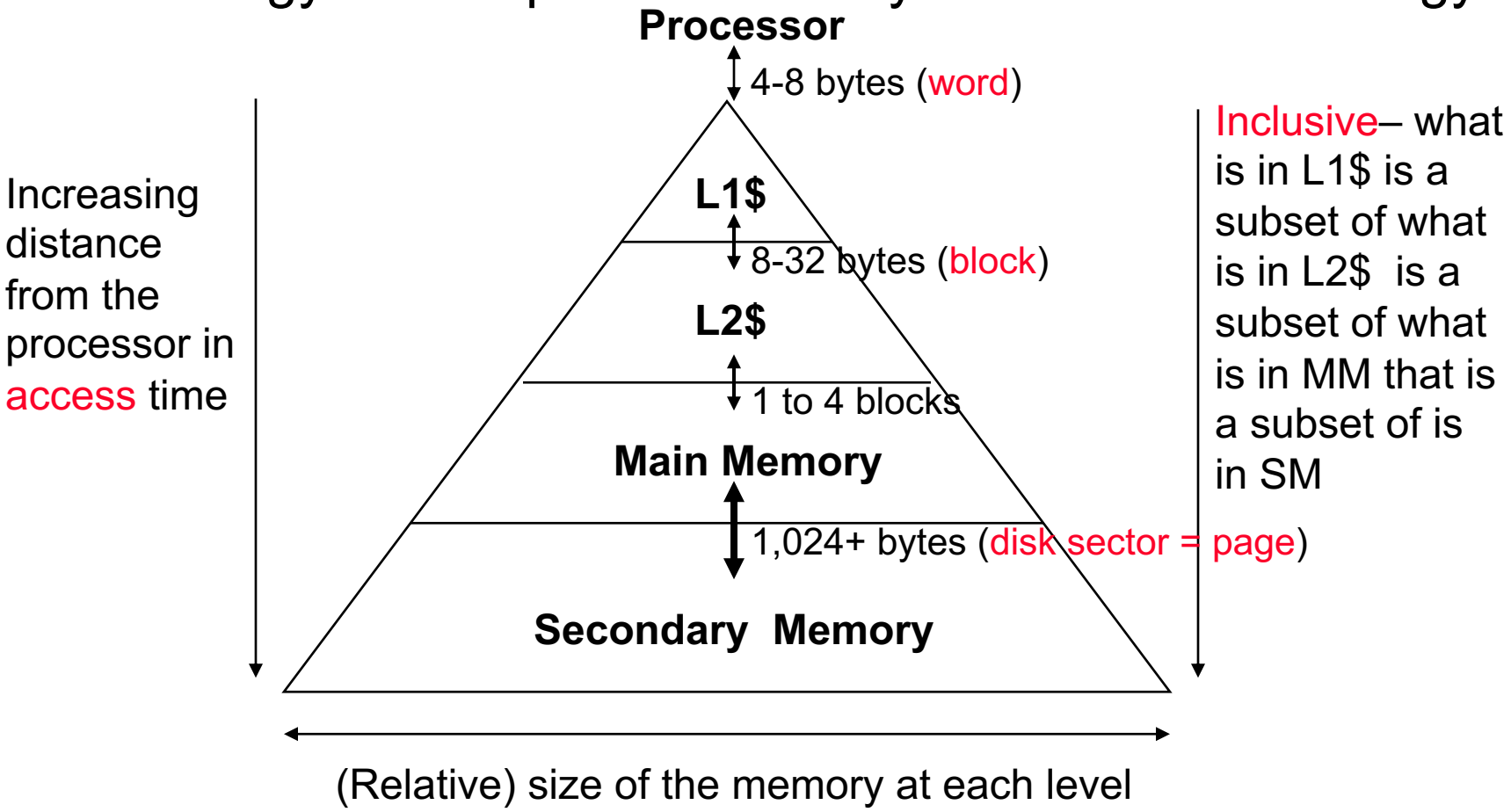
Memory Management

- Problem: many programs run simultaneously
- MMU manages the memory access.



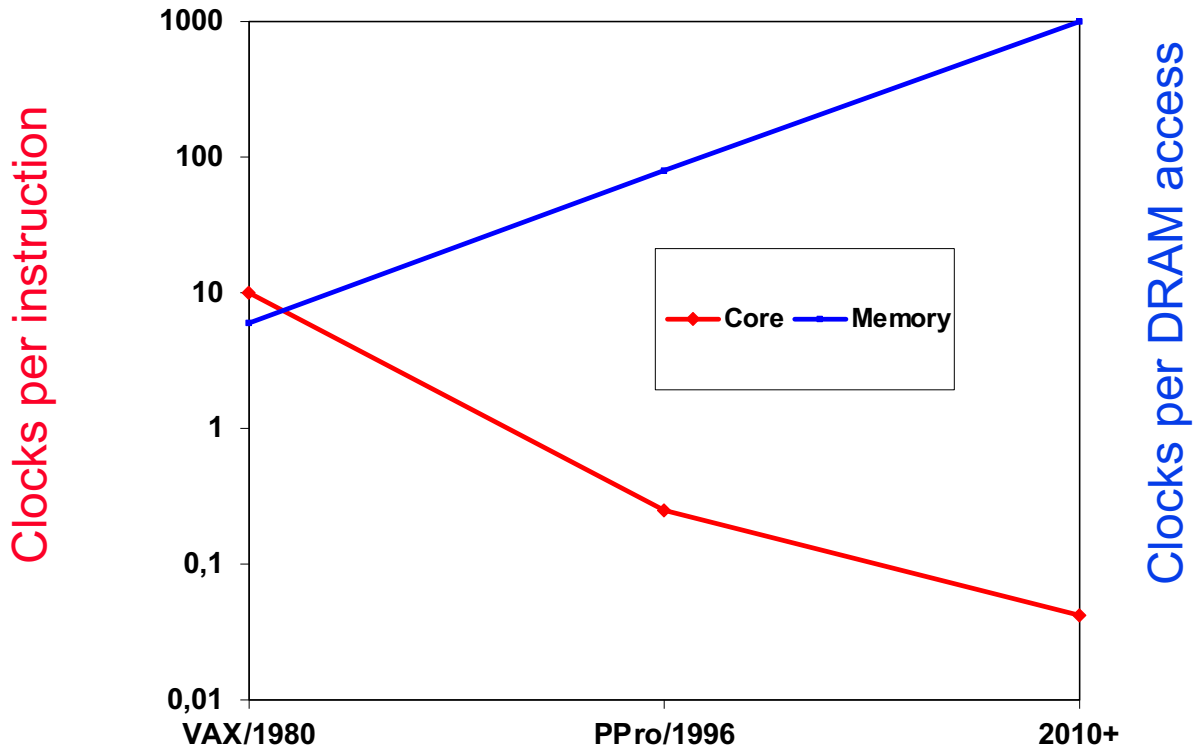
Review: Hierarchy of Memory

- Take advantage of the principle of locality to present the user with as much memory as is available in the cheapest technology at the speed offered by the fastest technology

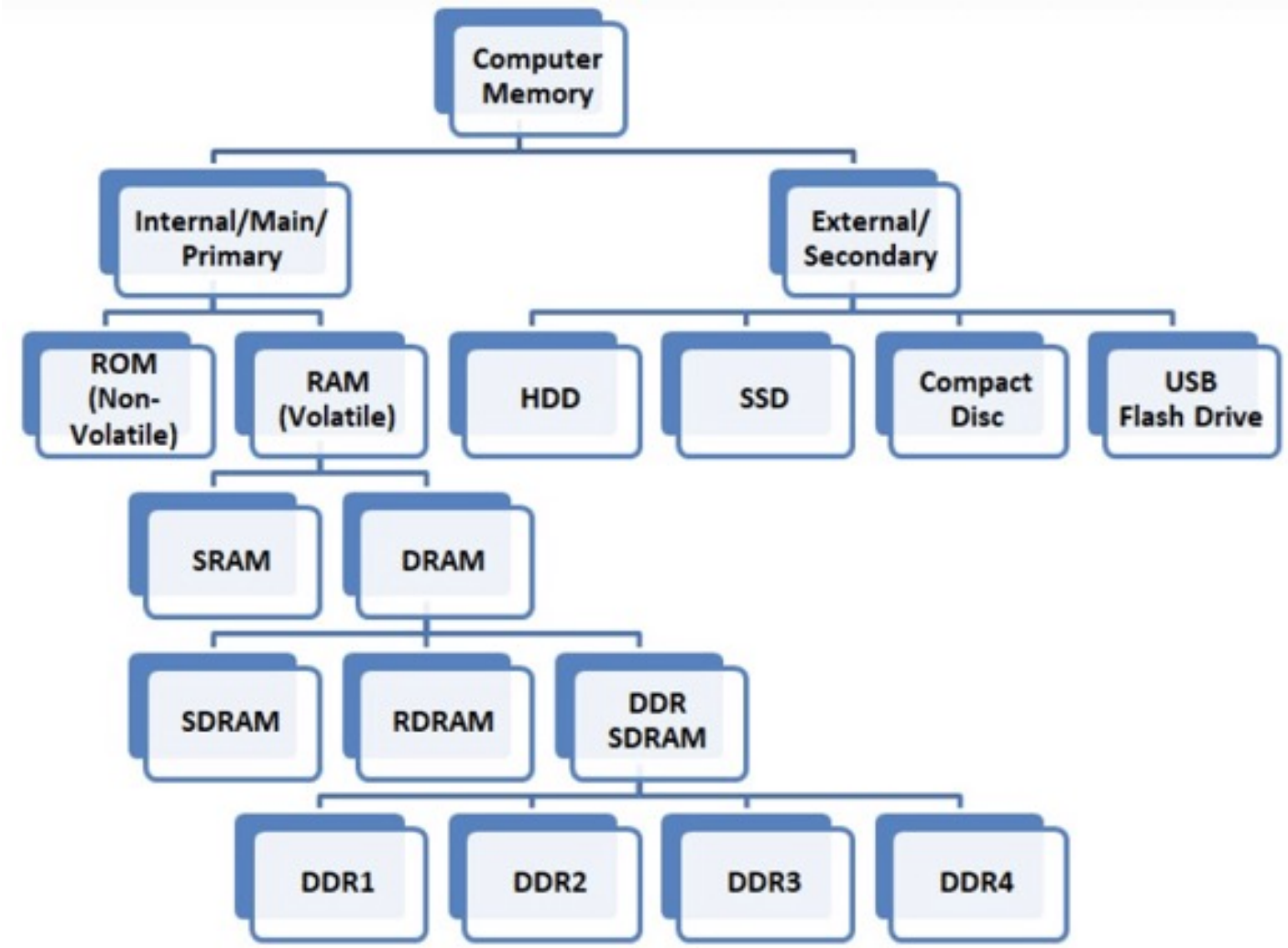


Review: The Memory Wall

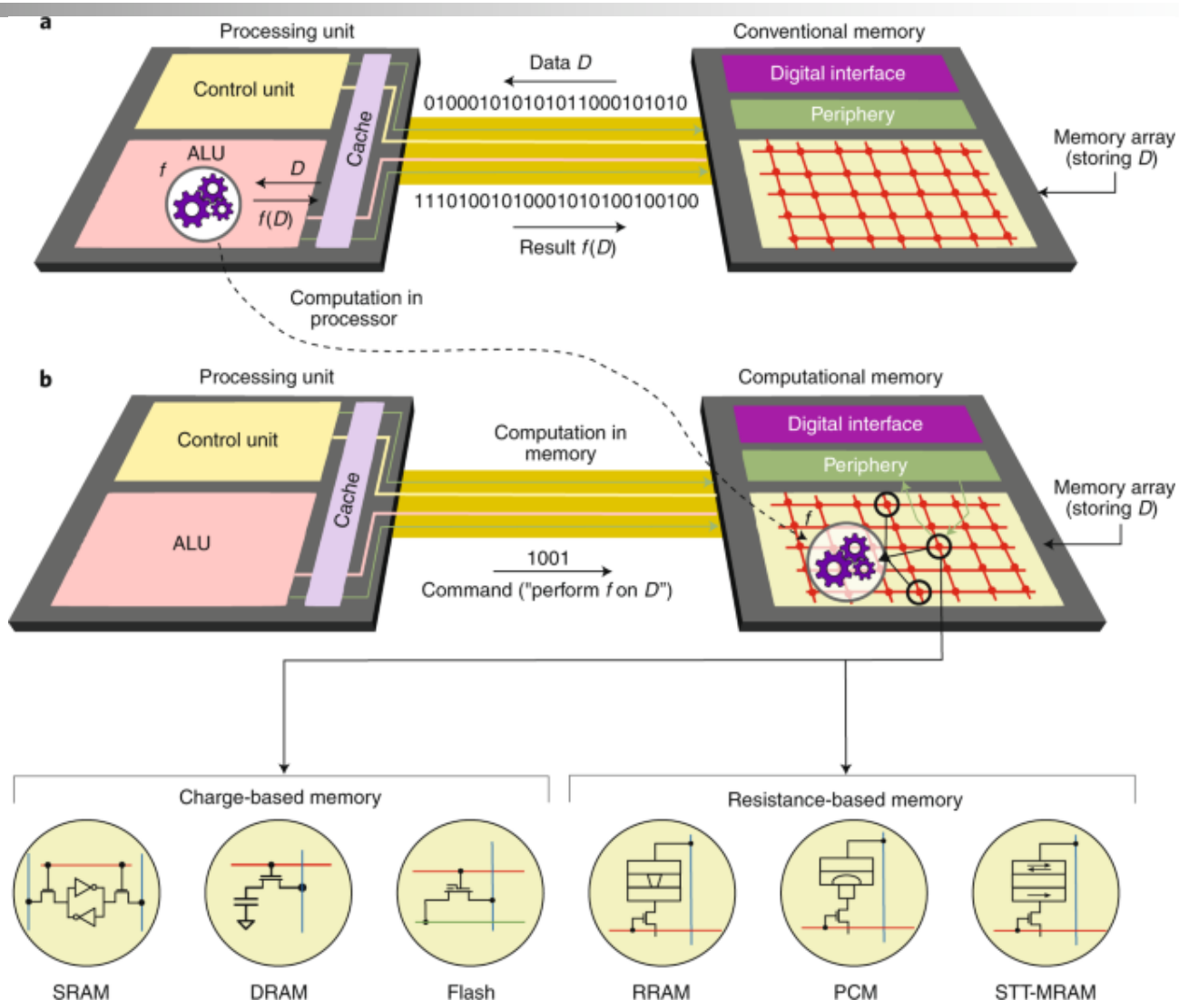
- Logic vs DRAM speed gap continues to grow



Review: Types of Memory Computing



From Conventional Memory to Computational Memory



https://www.nature.com/articles/s41565-020-0655-z

Memory Computing (Data Grid) for Big Data

Advances in Hardware

Multi-Core Architecture
(8 x 10core CPU per blade)

Parallel scaling across blades

One blade ~$50.000 = 1
Enterprise Class Server

64 bit address space – 2TB in current server boards

25GB/s data throughput

Cost-performance ratio rapidly declining

Advances in Software

Row and Column Store

Compression

Partitioning

No Aggregate Tables

Insert Only

On-the-fly extensibility

https://natishalom.typepad.com/nati_shaloms_blog/2013/01/in-memory-computing-data-grid-for-big-data.html