Performance:

- Latency (or response time or execution time): time to complete one task
- Bandwidth (or throughput): tasks completed per unit time
- Time = seconds / program = (instructions / program) * (clock cycles / instruction) * (seconds / clock cycle)
- Workload: Set of programs run on a computer; specifies both programs and relative frequencies
- Benchmark: Program selected for use in comparing computer performance
  - Benchmarks form a workload
  - Usually standardized so that many use them
- Amdahl's Law: Speedup = time without enhancement / time with enhancement = $1/((1 - F) + (F / S_E))$

Floating Point:

- Sign: determines the sign of the number (0 for positive, 1 for negative)
- Exponent: in biased notation, bias of 127 (smallest is 0)
- Significand: fraction part of number; 0 < significand < 1 (for normalized #s)
- Note: 0 has no leading 1, so reserve exponent value 0 just for number 0

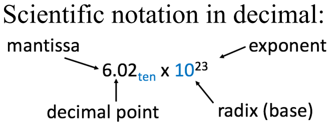
Sign	Exponent	Significand
1 bit	8 bits	23 bits

For normalized floats:
Value = $(-1)^{Sign} \times 2^{(Exponent - Bias)} \times 1.significand_2$

For denormalized floats:
Value = $(-1)^{Sign} \times 2^{(Exponent - Bias + 1)} \times 0.significand_2$

Exponent	Significand	Meaning
0	Anything	Denorm
1-254	Anything	Normal
255	0	Infinity
255	Nonzero	NaN

- Largest finite positive value that can be stored using a single precision float: 0x7F7FFFFF = $(2 - 2^{-23}) \times 2^{127}$
- Smallest positive value that can be stored using a single precision float: 0x00000001 = $2^{-23} \times 2^{-126}$
- Smallest positive normalized value that can be stored w/ a single precision float: 0x00800000 = 2^{-126}
- Largest single precision real can represent: $1.11…11 \times 2^{+127}$
- With denorms, can represent #s as small as $2.0_{ten} \times 10^{-38}$ to as large as $2.0_{ten} \times 10^{38}$
- Scientific notation in decimal:



  - Normalized: no leadings 0s (exactly one digit to left of decimal point)
  - Ex: normalized: 1.0×10^{-9}, not normalized: 0.1×10^{-8}, 10.0×10^{-10}
- Scientific notation in binary:
  - Computer arithmetic that supports it is called floating point, because it represents numbers where the binary point is not fixed, as it is for integers
  - Declare such variable in C as float (double for double precision)

- "Binary Point" signifies boundary between integer and fractional parts:
  - Example 6-bit representation:
  - $10.1010_{two} = 1 \times 2^1 + 1 \times 2^{-1} + 1 \times 2^{-3} = 2.625_{ten}$
  - Fixed binary point range of 6-bit representations: 0 to 3.9375 (~4)

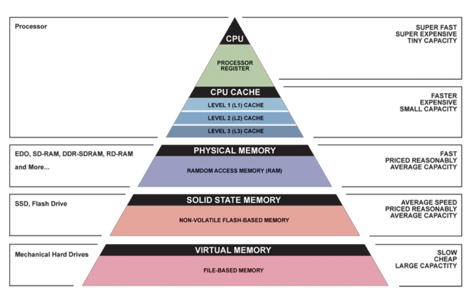
Fractional Powers of 2

i	2^{-i}	
0	1.0	1
1	0.5	1/2
2	0.25	1/4
3	0.125	1/8
4	0.0625	1/16
5	0.03125	1/32
6	0.015625	
7	0.0078125	
8	0.00390625	
9	0.001953125	
10	0.0009765625	
11	0.00048828125	
12	0.000244140625	
13	0.0001220703125	
14	0.00006103515625	
15	0.000030517578125	

- Underflow/overflow when exponent is too large or two small (negative) to fit in 8 bits
- Dividing by 0 produces ± ∞, not overflow except 0/0
- NaN examples: sqrt(-4.0)or 0/0
- Double precision floating point:
  - 1 bit for sign (s)
  - 11 bits for exponent (E)
  - 52 bits for fraction (F)
  - Bias of 1023
  - 1 extra bit of precision if leading 1 is implicit
  - (-1)s * (1 + F) * 2E
  - Range 2.0×10^{-308} to 2.0×10^{308}

How it is interpreted / How it is encoded

	Decimal Exponent	signed 2's complement	Biased Notation	Decimal Value of Biased Notation
∞, NaN	For infinities		11111111	255
Getting closer to zero ↓	127	01111111	11111110	254
	…	…	…	…
	2	00000010	10000001	129
	1	00000001	10000000	128
	0	00000000	01111111	127
	-1	11111111	01111110	126
	-2	11111110	01111101	125
	…	…	…	…
	-126	10000010	00000001	1
Zero	For Denorms	10000001	00000000	0



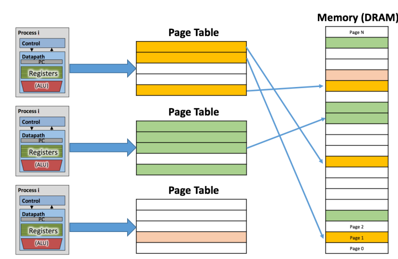
▲ Simplified Computer Memory Hierarchy
Illustration: Ryan J. Leng

Physical Memory:

- Visible to kernel (& firmware)
- Main/internal memory
  - Fast, but expensive
  - Loses data on power loss
  - Directly accessible by CPU
  - Ex: registers, cache, DRAM
- Auxiliary/external memory
  - Cheap, but slow
  - Retains data on power loss
  - Not directly accessible by CPU
  - Ex: hard drive, SSD, flash drive

Virtual Memory:

- Primary memory visible to your programs
  - Hides physical memory from general programs
  - Hardware-accelerated (most systems)
  - Cannot be disabled (most modern systems)
  - Nothing to do with hard drive or SSD
- Note: "swap"/"page" file (C:\pagefile.sys)
  - Secondary mem. used when primary mem. full
  - Can be disabled
  - Doesn't need VM per se (can be emulated), but only practical (fast) with hardware-accelerated VM
  - HW accelerator: "memory-management unit" (MMU)



Each process has a dedicated page table. Physical memory non-consecutive.

Virtual Memory Implementation:

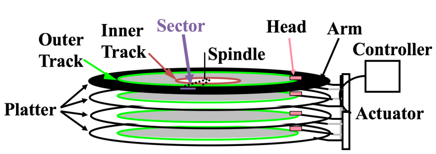
- Goal: separate programs' memory spaces
- Efficiency vs. flexibility tradeoff
  - Doesn't have to be linearly mapped
  - Hardware-accelerated or software emulated?
- Two common (but orthogonal) approaches:
  - Segmentation: split mem. into segment base + offset (less popular nowadays)
  - Paging: split mem. into conveniently-sized blocks (focus in 61C)

Virtual Memory Paging:

- Divide mem space into pages (4KiB)
- Treat entire page as a single unit of mem (attributes uniform within each page)

- Goal: find an efficient & practical way to represent attributes and permissions
- CPU uses these page tables in memory for address translation
- Page table base register = address in physical memory?
- Translation Lookaside Buffer (TLB)
- Reading page tables from DRAM slow
- Dedicate cache for page table entries
- Usually fully-associative; usually small
- Problem: wasteful. Why?
    - 4GiB RAM / 4KiB pages ≈ 1M pages
    - Even 4 bytes of information / page uses 4MiB of memory / process
    - 256 processes use 1GiB of RAM just for page tables!
- Better idea? Hierarchy (add indirection)
    - Sub-divide each "large page" into "smaller pages" when necessary
    - Separate page tables for each level
    - Massive space improvement
    - Small time penalty
- Equations:
    - VPN bits = log(VA size / page size)
    - PPN bits = log(PA size / page size)
    - Page offset = log(page size)
    - Bits per row of PT: PPN bits + valid + dirty + R + W
    - Size of page table = # of pages = size of VA space / size of a page
    - Page table base register = address in physical memory?
    - Size of a page table:
        - (Size of VM / size of a page) * size of page table entry
    - TLB Reach = TLB size * page size

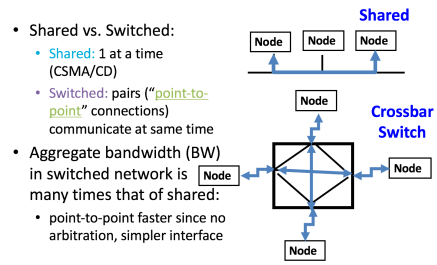
<u>Disks:</u>



- Disk Access Time = Seek Time + Rotation Time + Transfer Time + Controller Overhead
- Seek Time = time to move head to correct track/cylinder; average time ((total # of tracks)/3) * time to move across one track
- Rotation time = time for the disk to rotate to the correct sector to read from/ write to; average rotations to get to correct location is ½
- Transfer Time = time taken by the sectors of the block and any gaps between them to rotate past the head; time to get data on/off the disk
- Modern disks have on-disk caches, hidden from the outside world
- Generally, what limits real performance is the on-disk cache access time

<u>Networks:</u>

- Shared vs. switch-based networks



- What makes them work:
    - Links connecting switches and/or routers to each other/devices
    - Ability to route packets from source to destination
    - Layering, redundancy, protocols, and encapsulation as means of abstraction
- SW Send steps
    1. App. copies data to OS buffer
    2. OS calculates checksum, starts timer
    3. OS sends data to network interface HW and says start
- SW Receive steps
    3. OS copies data from network interface HW to OS buffer
    2. OS calculates checksum, if OK, send ACK; if not, delete message (sender resends when timer expires)
    1. If OK, OS copies data to user address space & signals application to continue
- Hierarchy network layers:
    - Application (chat client, game, etc.)
    - Transport (TCP, UDP)
    - Network (IP)
    - Data Link Layer (ethernet)
    - Physical Link (copper, wireless, ...)