

# Introduction to CSC 2250

---

**We studied Java (to write SW)**

**In this course, we will study HW & SW interface**

**What is HW & SW interface- why we need to study?**

# Check the answers & Explain why?

## Try in Java

- `System.out.print('0');` → 0
- `System.out.print('0'+ 0);` → 48
- `System.out.print('0' + '0');` → 96
- `System.out.print("0"+ '0');` → 00
- `System.out.print((1+2 == 3));` → true
- `System.out.print((0.1+0.2 == 0.3));` → ??
- `System.out.print(Byte.SIZE);` → 8
- `Byte x = Byte.MAX_VALUE;`
- `System.out.print(x);` → 127
- `System.out.print(++x);` → ??

*Try in JavaScript using alert - try above also (first six)*

- `alert(999999999999999999+1);` → adds 1
- `alert(999999999999999999-1);` → ??

# Topics of course

- Basics of computer system
  - Types, components, memory hierarchy
  - fetch-exec cycle, clock time, performance
- How data represented?
  - Digitization-Unsigned Integers
  - Integer Reps , Real – FLP Reps
- Machine & Assembly Language
  - MIPS- R/I/J Types
  - SPIM

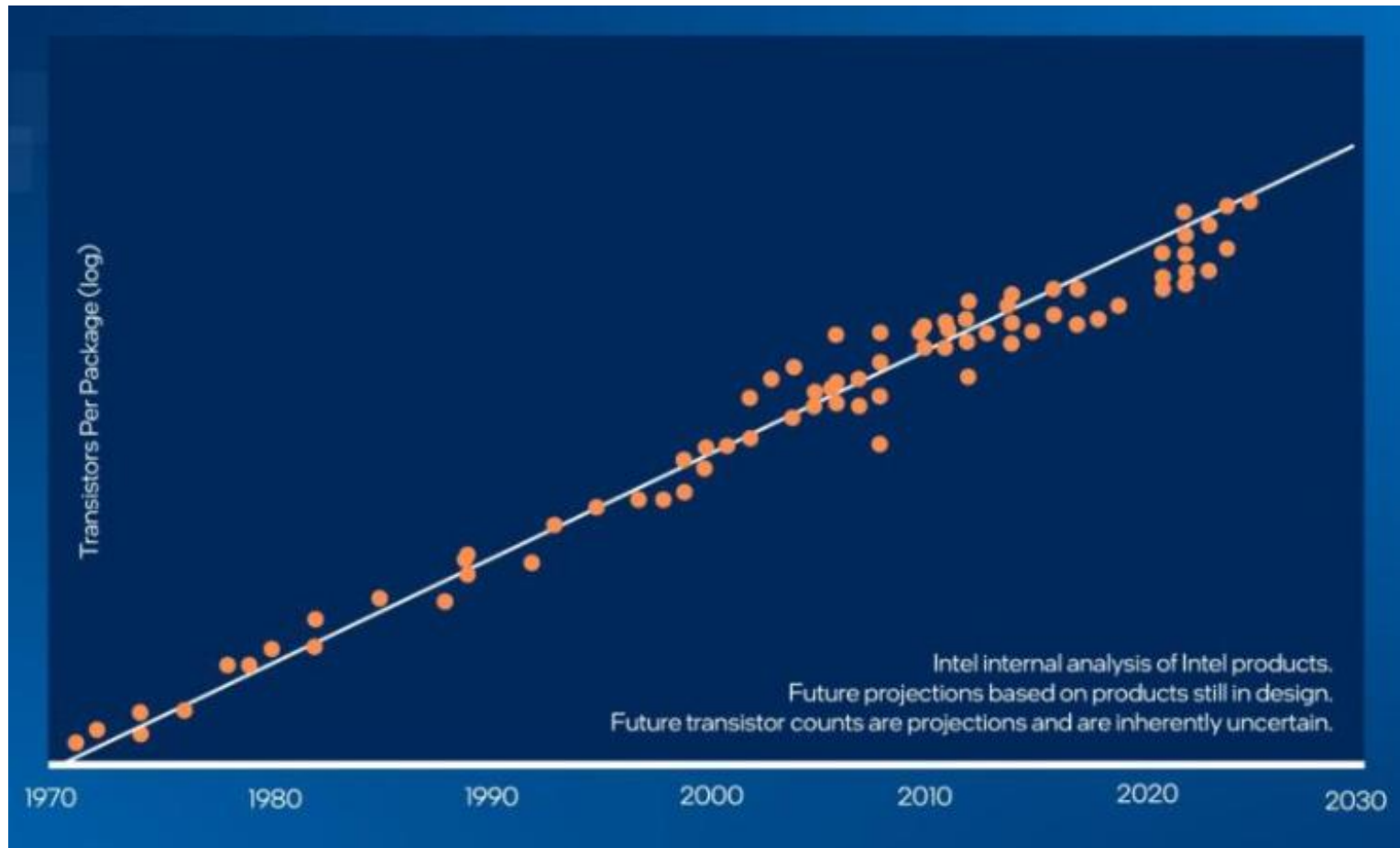
# Topics- Chapter 1

- History, Revolution- Calculator to AI & IoT
- Types of computers: Super to Mobile
- System: HW, SW, Processes, People
- HW: Mother Board, I/O, Storage, Net
- Mother Board: CPU, RAM, ROM,...
- CPU: Cores: ALU, Register, Control, L1-Cache; L2- Cache
- Memory Hierarchy: Cloud-Local Drive-RAM-Cache-Register
- SW: Application SW & System SW (OS, compilers, device drivers,...)
- HW & SW Interface: ISA and Machine Language
- Computer Organization and Architecture
- Fetch-Execute Cycle
- Memory Allocation
- Performance

# The Computer Revolution

- Progress in computer technology
  - Underpinned by Moore's Law
- Makes novel applications feasible
  - Computers in automobiles
  - Cell phones
  - Human genome project
  - World Wide Web
  - Search Engines
- Computers are pervasive

# Moore's Law



# Classes of Computers-1

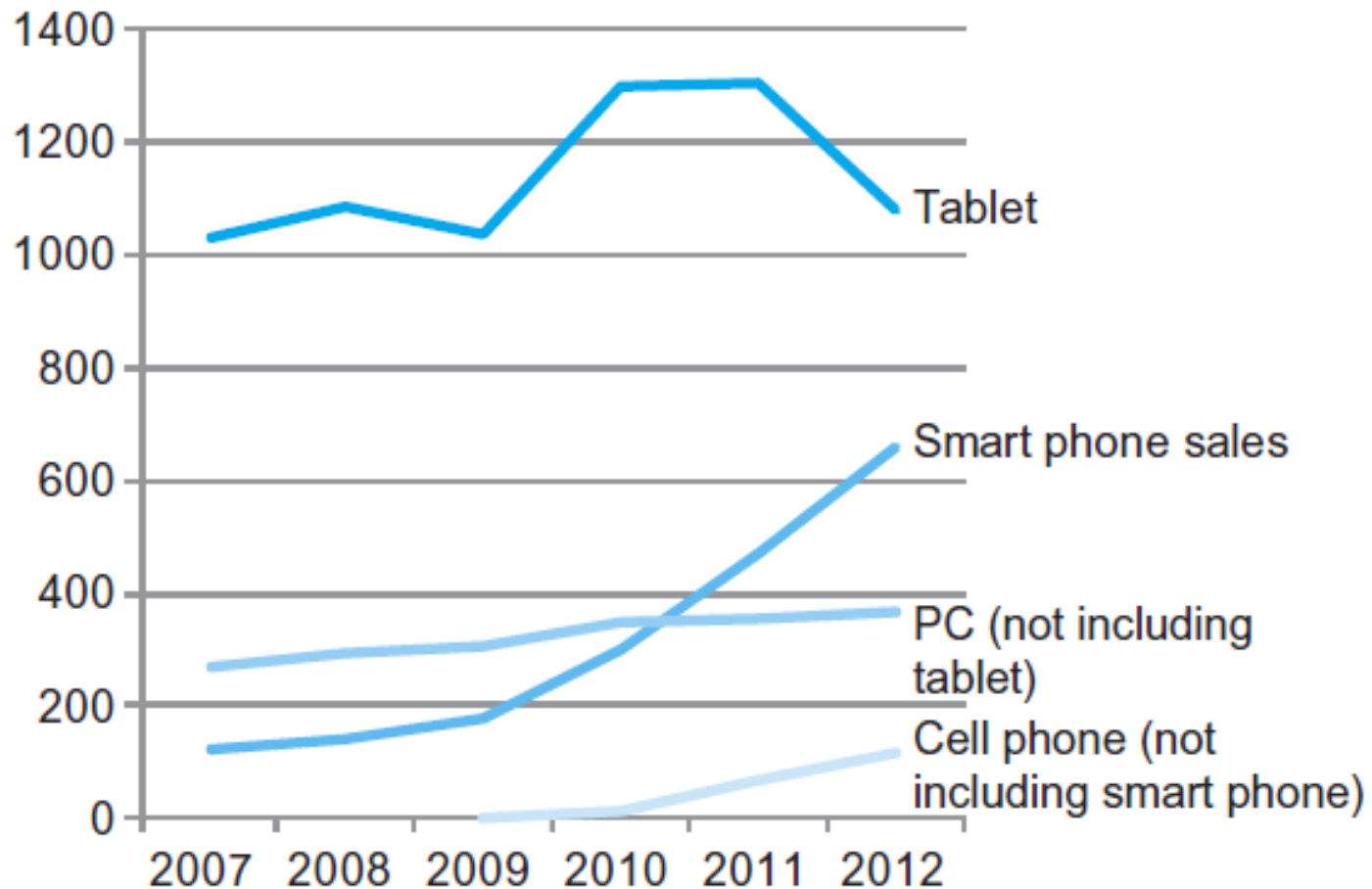
- Personal computers
  - General purpose, variety of software
  - Subject to cost/performance tradeoff
- Server computers
  - Network based
  - High capacity, performance, reliability
  - Range from small servers to building sized

# Classes of Computers-2

- Supercomputers
  - High-end scientific and engineering calculations
  - Highest capability but represent a small fraction of the overall computer market
- Embedded computers
  - Hidden as components of systems
  - Stringent power/performance/cost constraints



# The PostPC Era

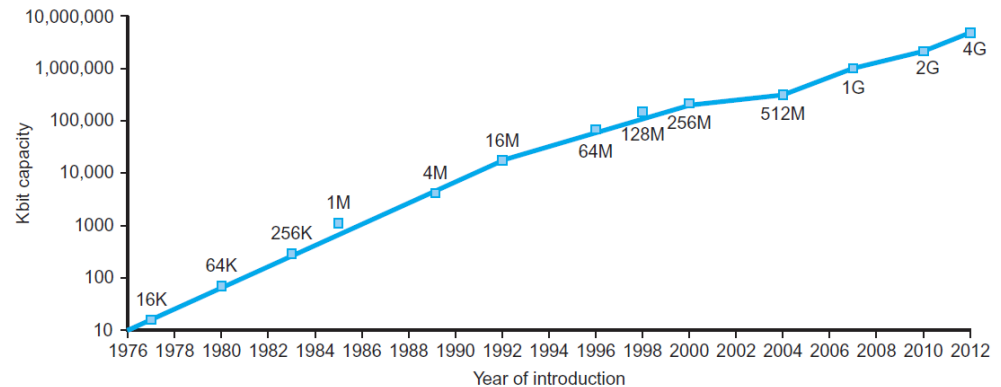


# The PostPC Era

- Personal Mobile Device (PMD)
  - Battery operated
  - Connects to the Internet
  - Hundreds of dollars
  - Smart phones, tablets, electronic glasses
- Cloud computing
  - Warehouse Scale Computers (WSC)
  - Software as a Service (SaaS)
  - Portion of software run on a PMD and a portion run in the Cloud
  - Amazon and Google

# Technology Trends

- Electronics technology continues to evolve
  - Increased capacity and performance
  - Reduced cost

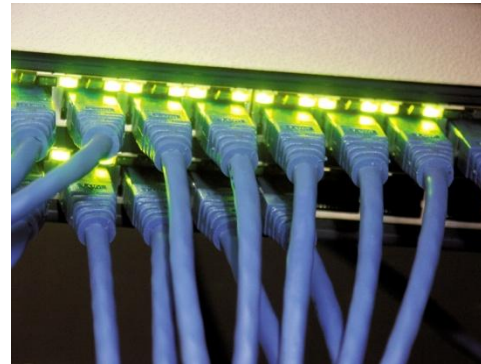


DRAM capacity

Year	Technology	Relative performance/cost
1951	Vacuum tube	1
1965	Transistor	35
1975	Integrated circuit (IC)	900
1995	Very large scale IC (VLSI)	2,400,000
2013	Ultra large scale IC	250,000,000,000

# Networks

- Communication, resource sharing, nonlocal access
- Local area network (LAN): Ethernet
- Wide area network (WAN): the Internet
- Wireless network: WiFi, Bluetooth



# What You Will Learn

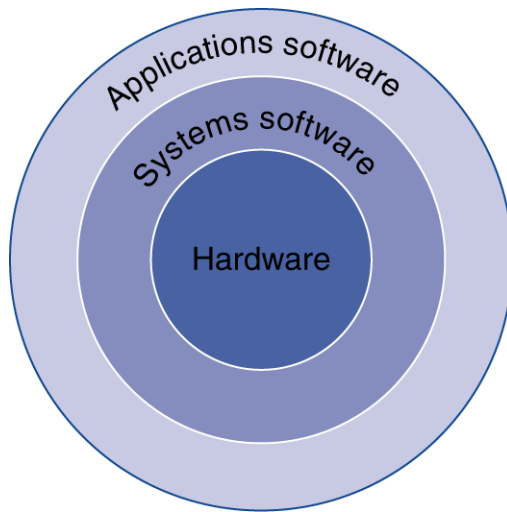
- How programs are translated into the machine language
  - And how the hardware executes them
- The hardware/software interface
- What determines program performance
  - And how it can be improved
- How hardware designers improve performance
- What is parallel processing

# Design Tradeoffs- Eight Great Ideas

- Design for **Moore's Law**
- Use **abstraction** to simplify design
- Make the **common case fast**
- Performance via **parallelism**
- Performance via **pipelining**
- Performance via **prediction**
- **Hierarchy** of memories
- **Dependability** via redundancy



# Below Your Program



- Application software
  - Written in high-level language
- System software
  - Compiler: translates HLL code to machine code
  - Operating System: service code
    - Handling input/output
    - Managing memory and storage
    - Scheduling tasks & sharing resources
- Hardware
  - Processor, memory, I/O controllers

# Levels of Program Code

- High-level language
  - Level of abstraction closer to problem domain
  - Provides for productivity and portability
- Assembly language
  - Textual representation of instructions
- Hardware representation
  - Binary digits (bits)
  - Encoded instructions and data

High-level  
language  
program  
(in C)

```
swap(int v[], int k)
{int temp;
  temp = v[k];
  v[k] = v[k+1];
  v[k+1] = temp;
}
```

Compiler

Assembly  
language  
program  
(for MIPS)

```
swap:
  muli $2, $5, 4
  add  $2, $4, $2
  lw   $15, 0($2)
  lw   $16, 4($2)
  sw   $16, 0($2)
  sw   $15, 4($2)
  jr   $31
```

Assembler

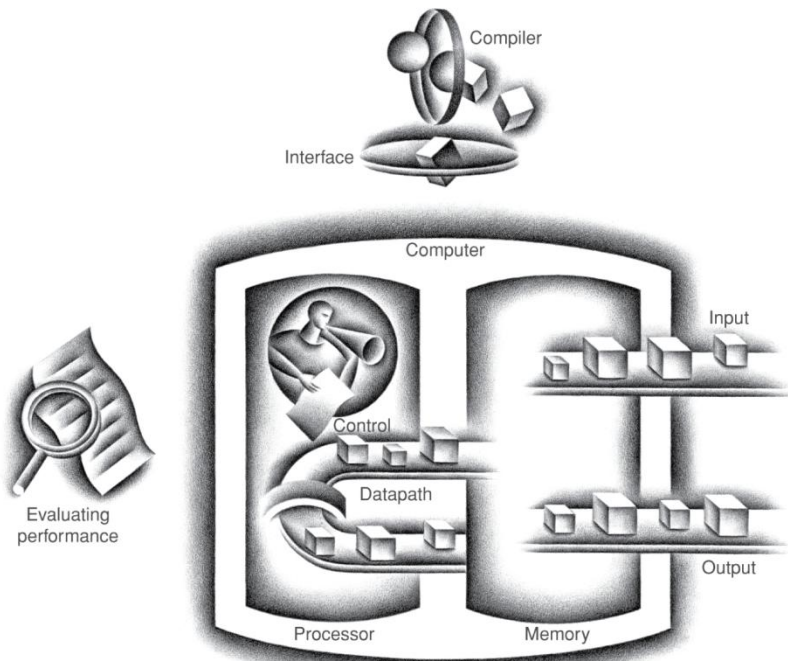
Binary machine  
language  
program  
(for MIPS)

```
000000001010000100000000000011000
000000000000110000001100000100001
100011000110001000000000000000000
100011001111001000000000000000100
101011001111001000000000000000000
101011000110001000000000000000100
00000011111000000000000000001000
```



# HW- Components of a Computer

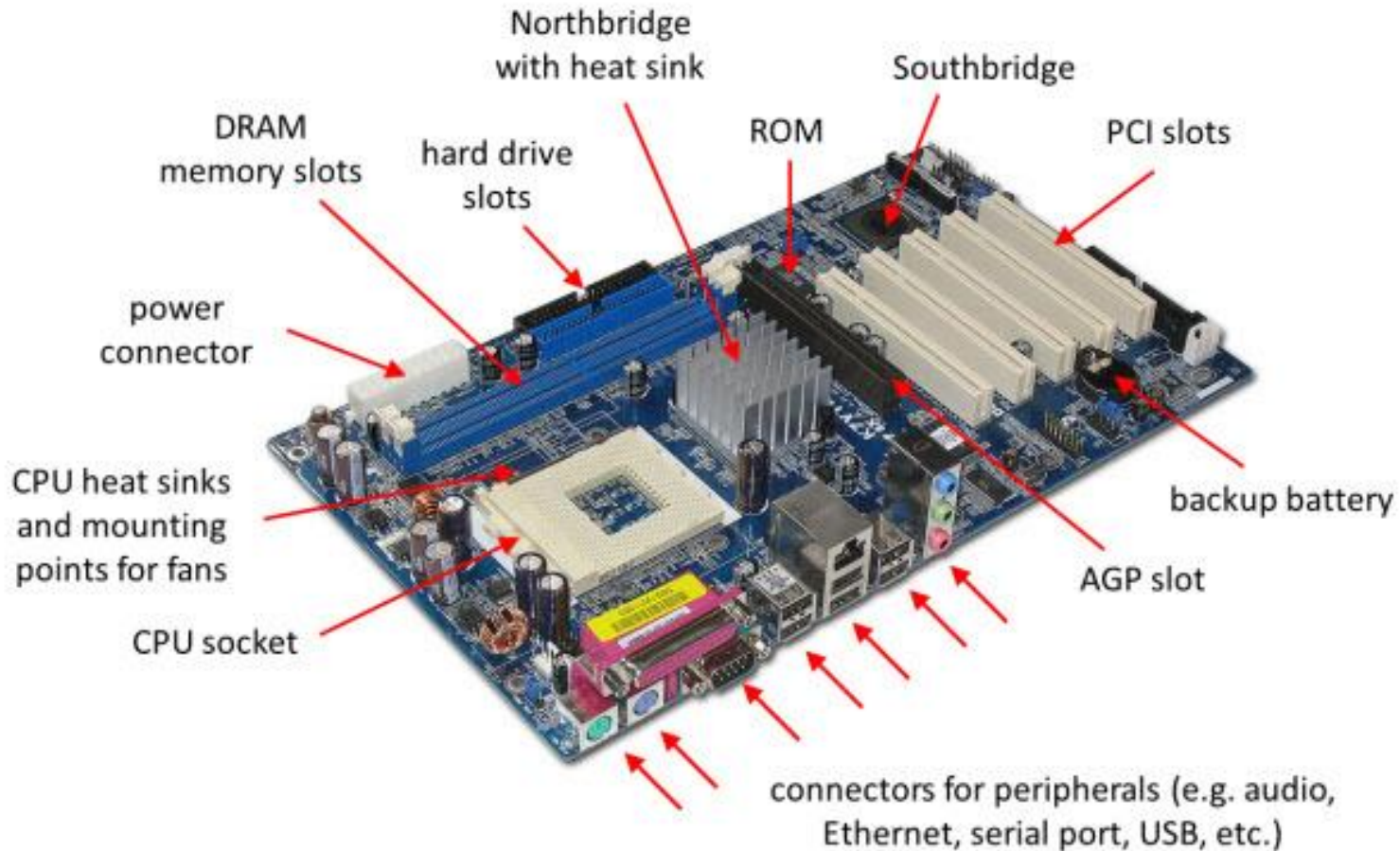
## The BIG Picture



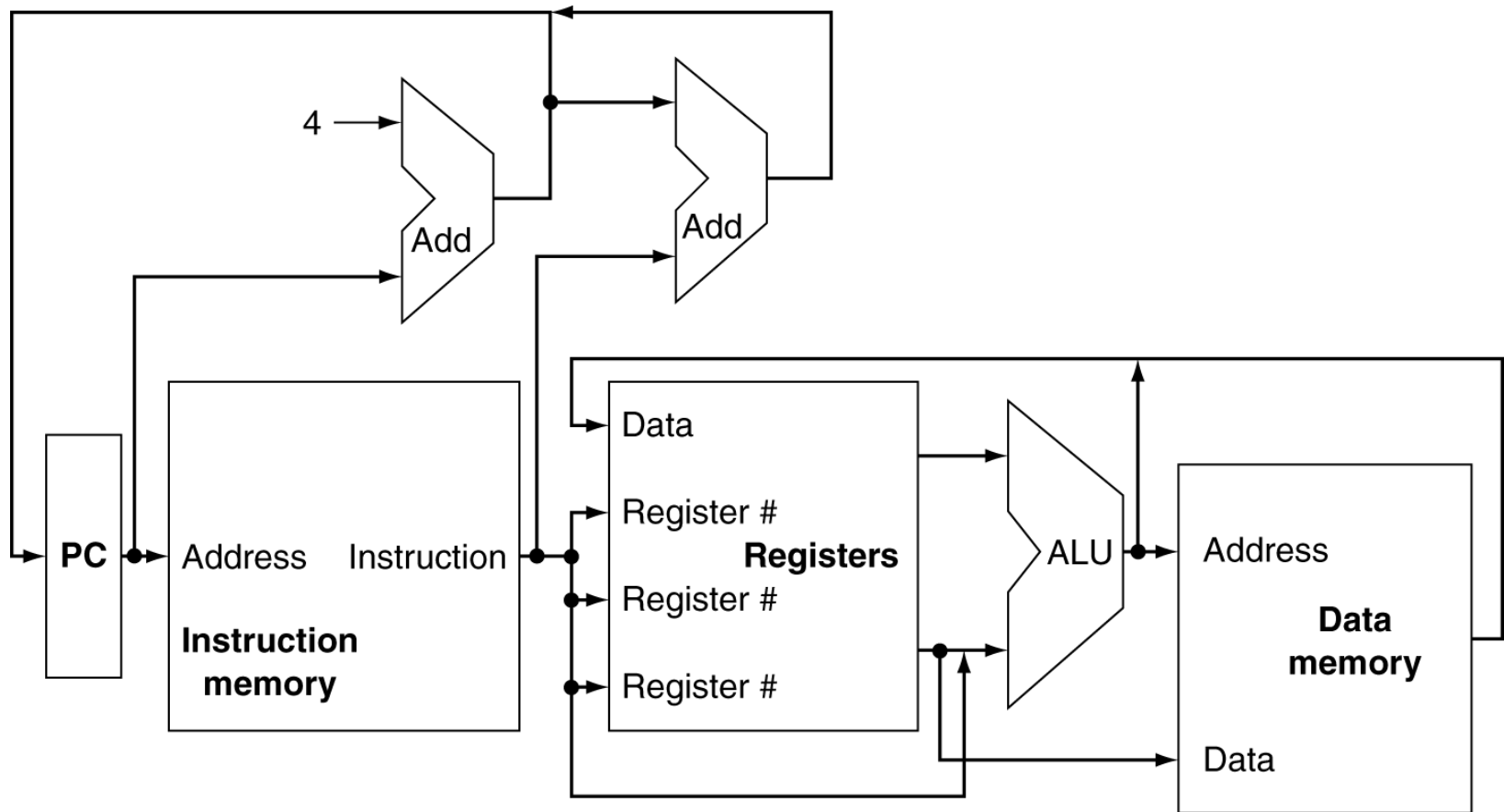
- Same components for all kinds of computer
  - Desktop, server, embedded
- Input/output includes
  - User-interface devices
    - Display, keyboard, mouse
  - Storage devices
    - Hard disk, CD/DVD, flash
  - Network adapters
    - For communicating with other computers

# Inside the Box (Mother Board)

- <https://study.com/academy/lesson/what-is-a-motherboard-definition-function-diagram.html>

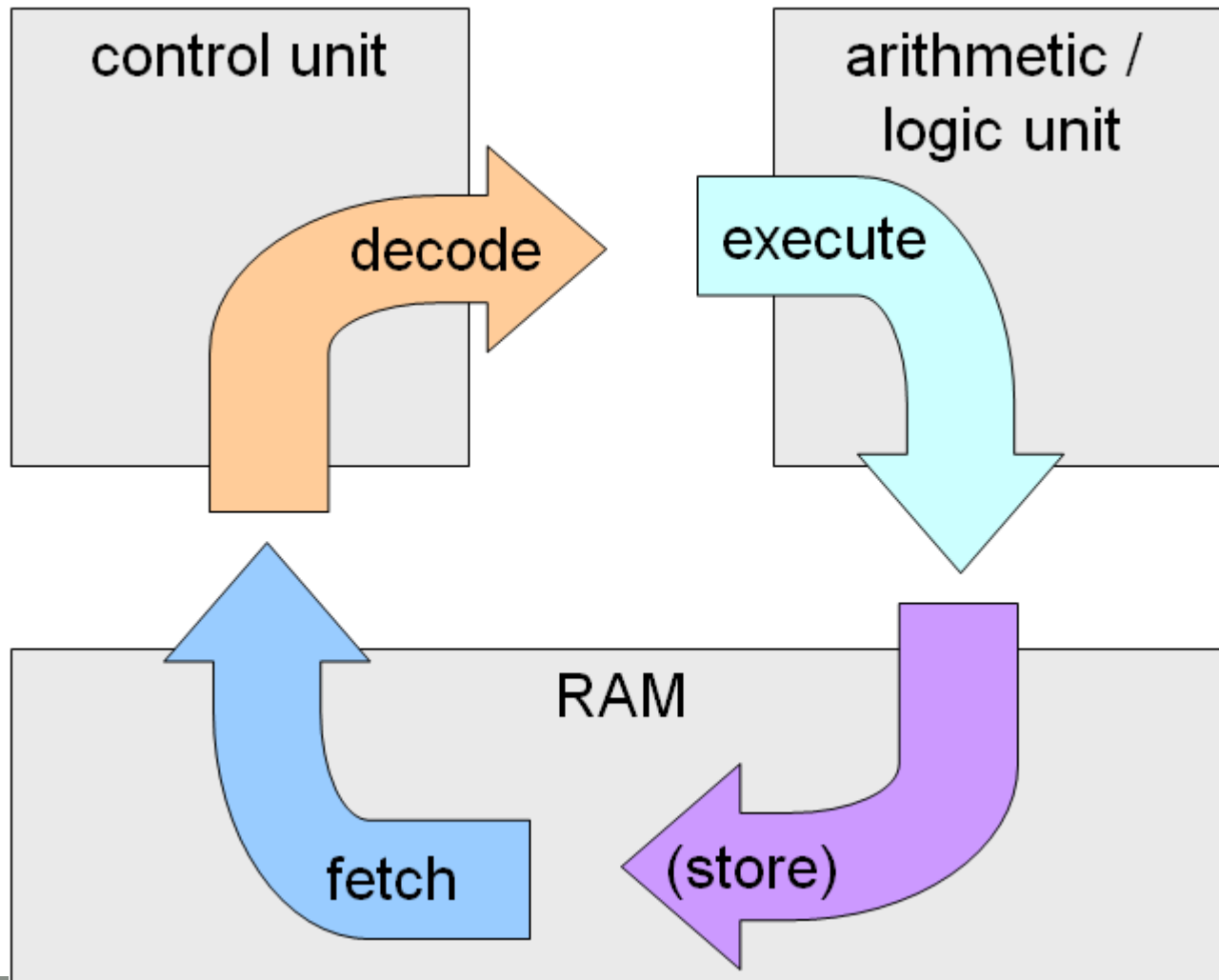


# CPU & ideal RAM Overview



**NOTE: RAM and CPU are on the mother board connected by FSB. There is no separate RAM for Data & Instruction.**

# FETCH-EXECUTE CYCLE

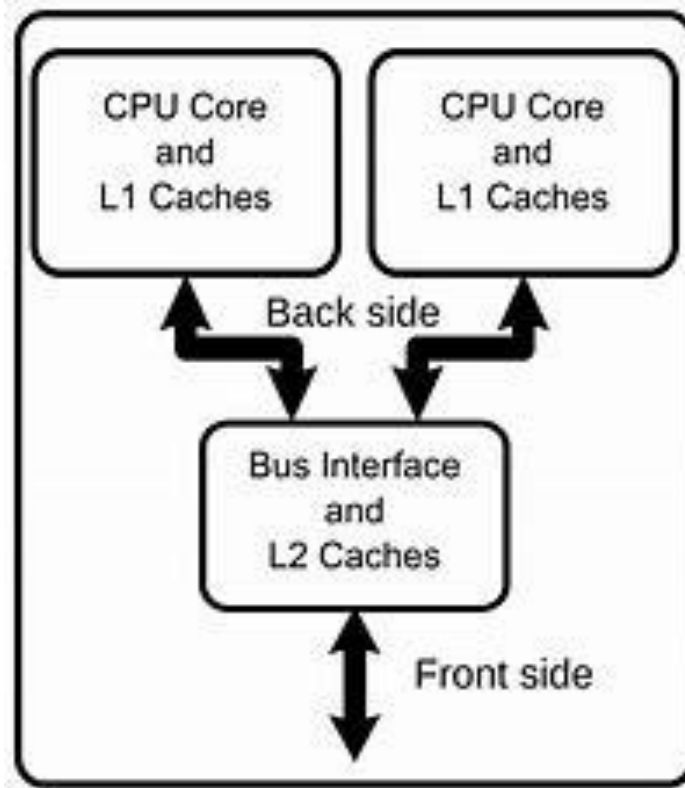


# Inside the Processor (CPU)

- Datapath: performs operations on data
- Control: sequences datapath, memory, ...
- Registrar
- Cache memory
  - Small fast SRAM memory for immediate access to data

# Inside the Processor (CPU)

- Dual Core



# Inside the Processor (CPU)

- Apple A5

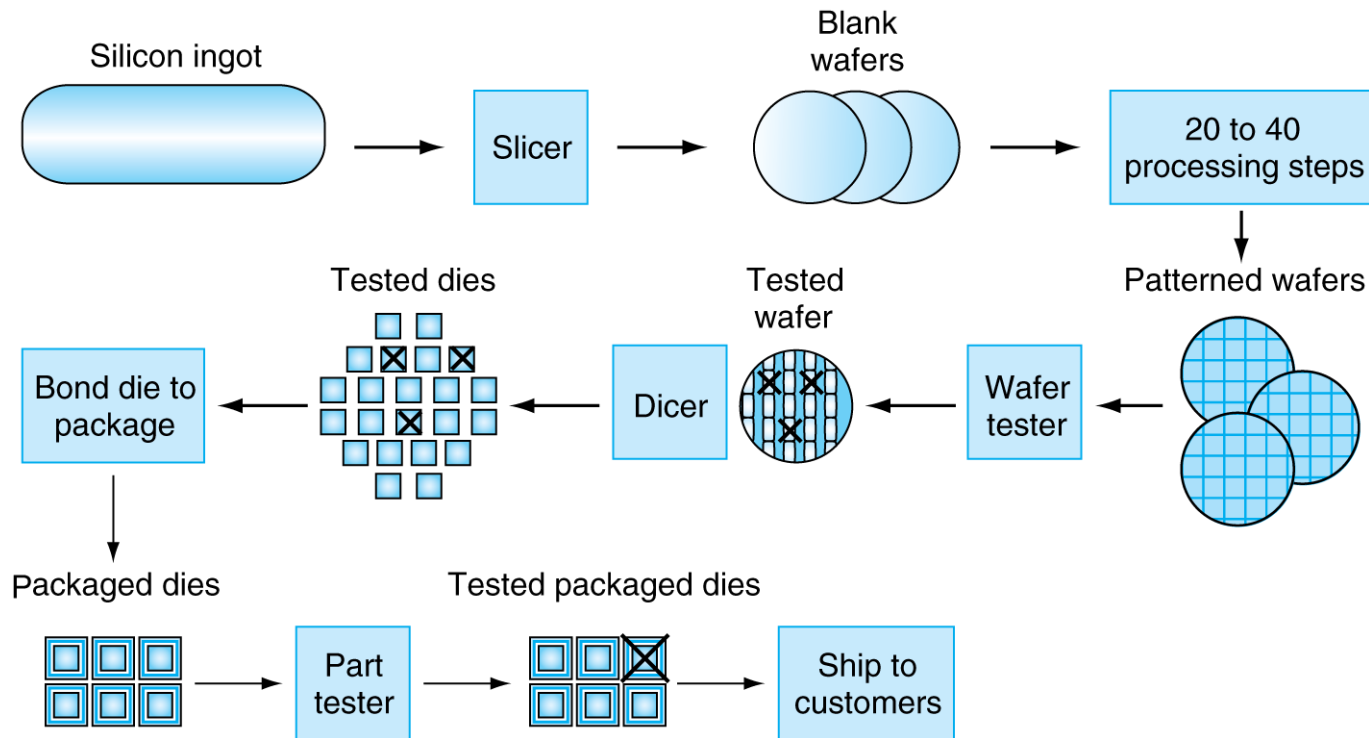


# Semiconductor Technology

- Silicon: semiconductor
- Add materials to transform properties:
  - Conductors
  - Insulators
  - Switch

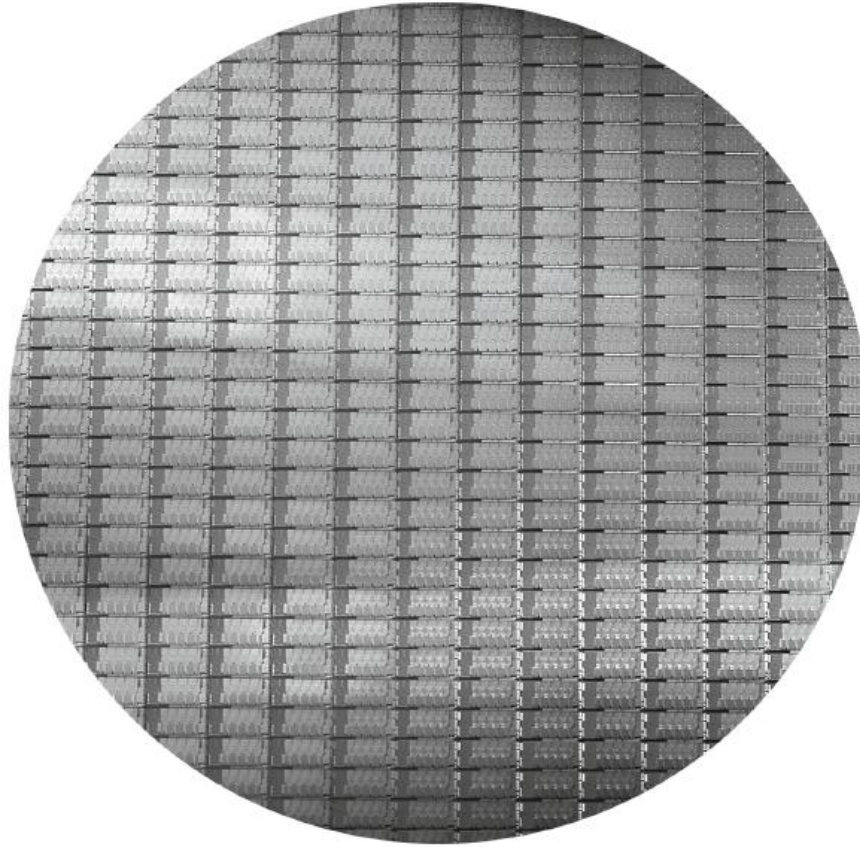


# Manufacturing ICs



- **Yield:** proportion of working dies per wafer

# Intel Core i7 Wafer



- 300mm wafer, 280 chips, 32nm technology
- Each chip is 20.7 x 10.5 mm

# Integrated Circuit Cost

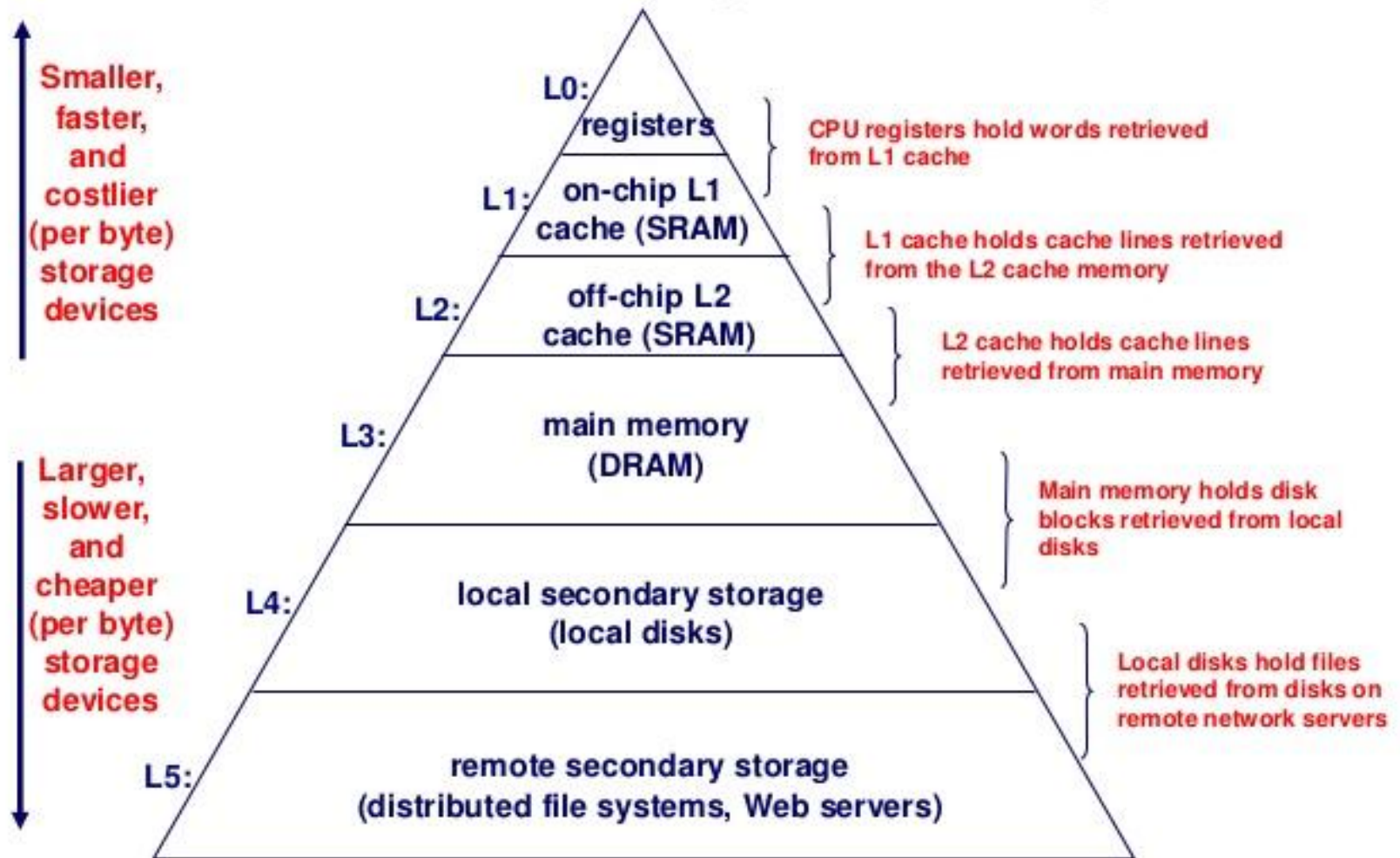
$$\text{Cost per die} = \frac{\text{Cost per wafer}}{\text{Dies per wafer} \times \text{Yield}}$$

$$\text{Dies per wafer} \approx \text{Wafer area} / \text{Die area}$$

$$\text{Yield} = \frac{1}{(1 + (\text{Defects per area} \times \text{Die area} / 2))^2}$$

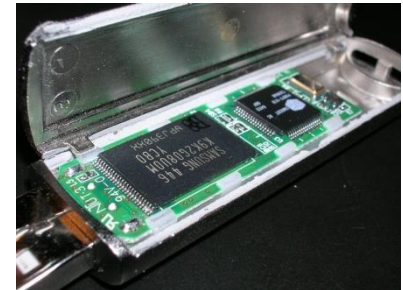
- Nonlinear relation to area and defect rate
  - Wafer cost and area are fixed
  - Defect rate determined by manufacturing process
  - Die area determined by architecture and circuit design

# An Example Memory Hierarchy



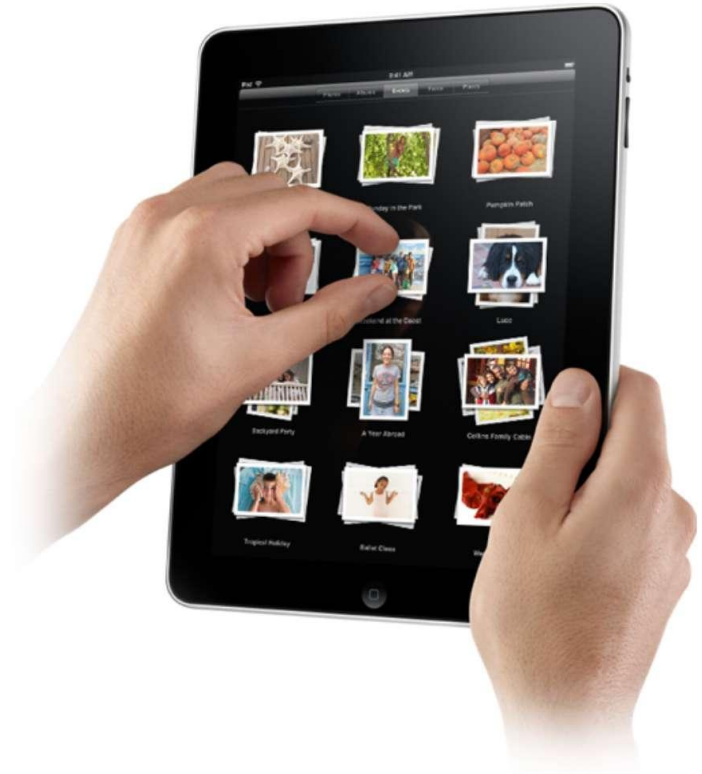
# A Safe Place for Data

- Volatile main memory
  - Loses instructions and data when power off
- Non-volatile secondary memory
  - Magnetic disk
  - Flash memory
  - Optical disk (CDROM, DVD)



# Touchscreen

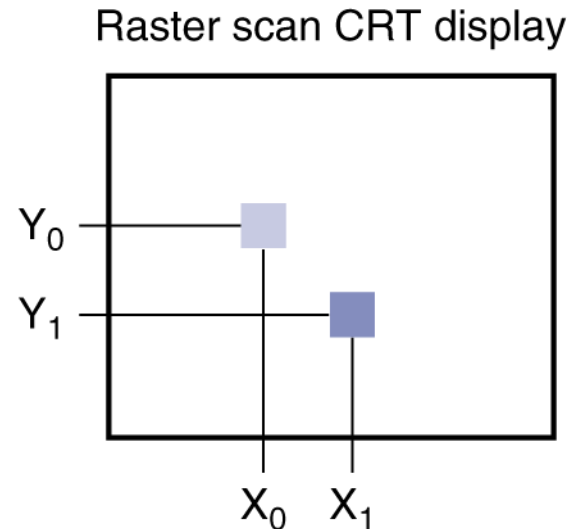
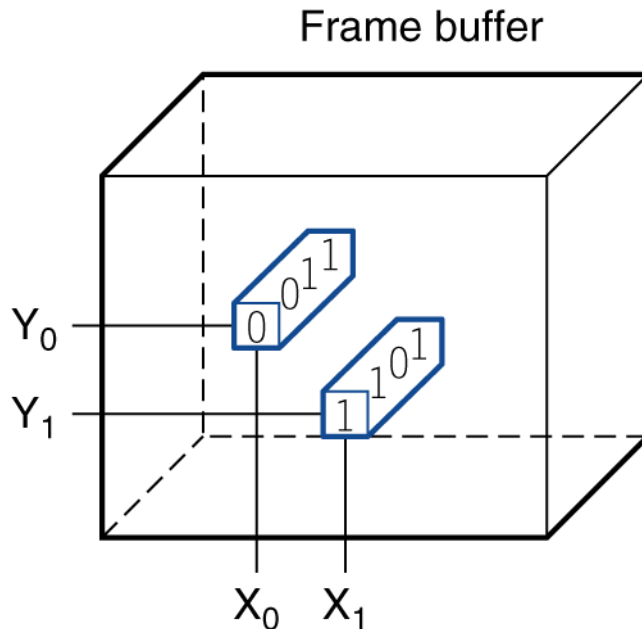
- PostPC device
- Supersedes keyboard and mouse
- Resistive and Capacitive types
  - Most tablets, smart phones use capacitive
  - Capacitive allows multiple touches simultaneously



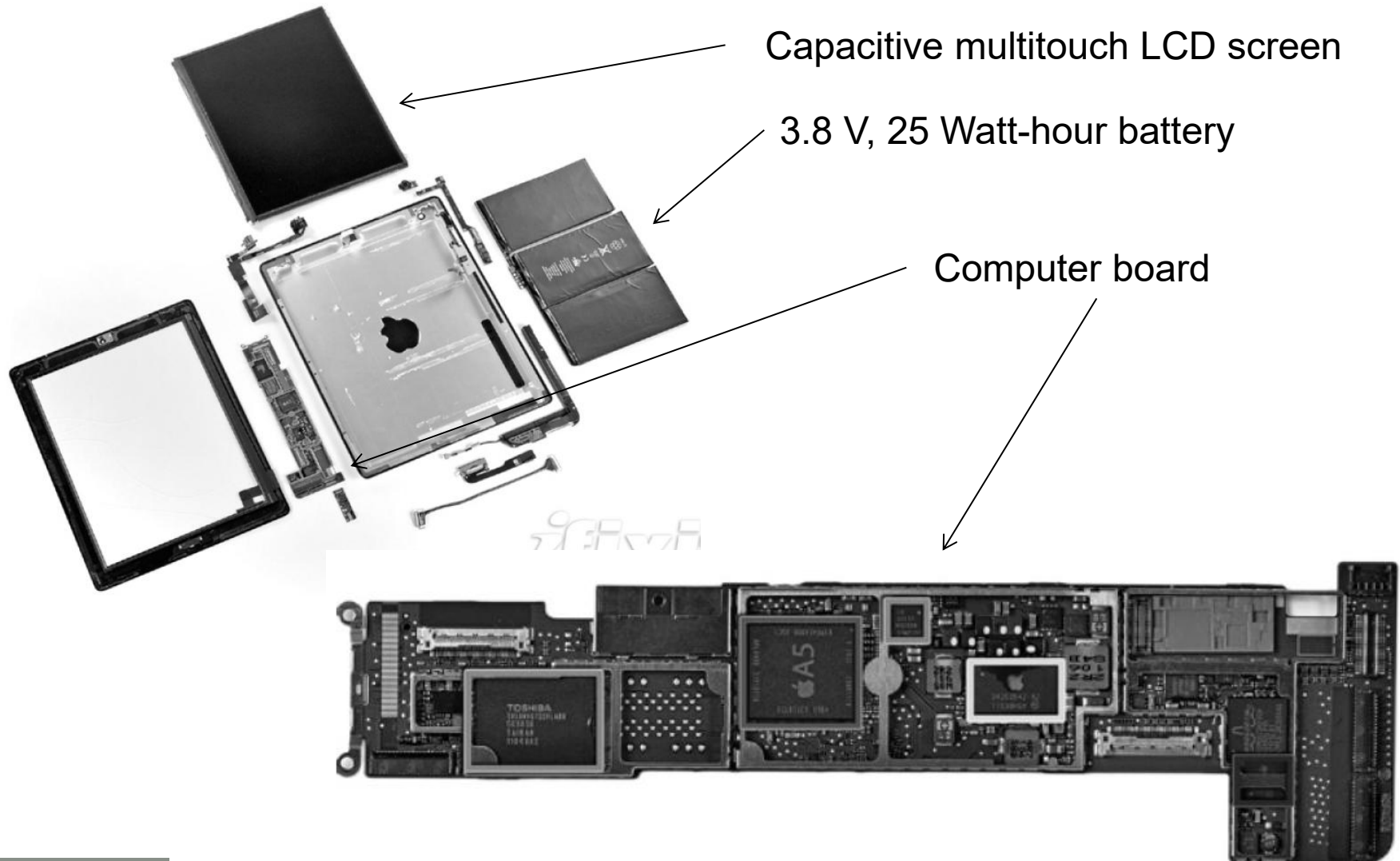


# Through the Looking Glass

- LCD screen: picture elements (pixels)
  - Mirrors content of frame buffer memory



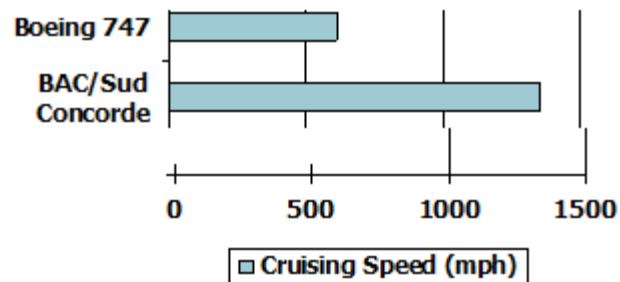
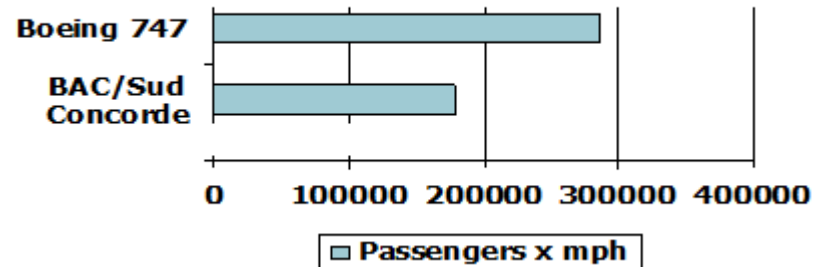
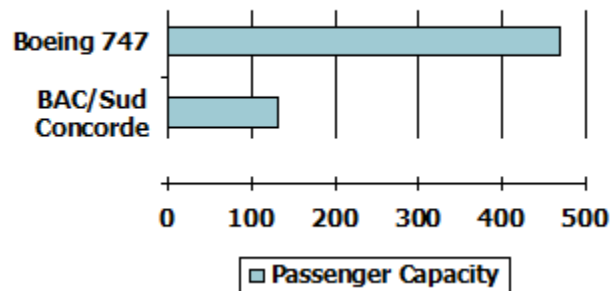
# Opening the Box





# Performance (Speed Vs Tput)

- Which airplane has the best performance?



- Which is better?  
Server or Workstation ?

# Response Time Vs Throughput

- Response time [ Time / Task ]
  - How long it takes to do a task
- Throughput [ Task / Time ]
  - Total work done per unit time
    - e.g., tasks/transactions/... per hour
- How response time and throughput affected by
  - Replacing the processor with a faster version?
  - Adding more processors (if not parallel) ?

**We'll focus on response time (speed) for now...**

# Performance – Response Time/ Speed

- Algorithm
  - Determines number of operations executed
- Programming language, compiler, architecture
  - Determine number of machine instructions executed per operation
- Processor and memory system
  - Determine how fast instructions are executed
- I/O system (including OS)
  - Determines how fast I/O operations are executed

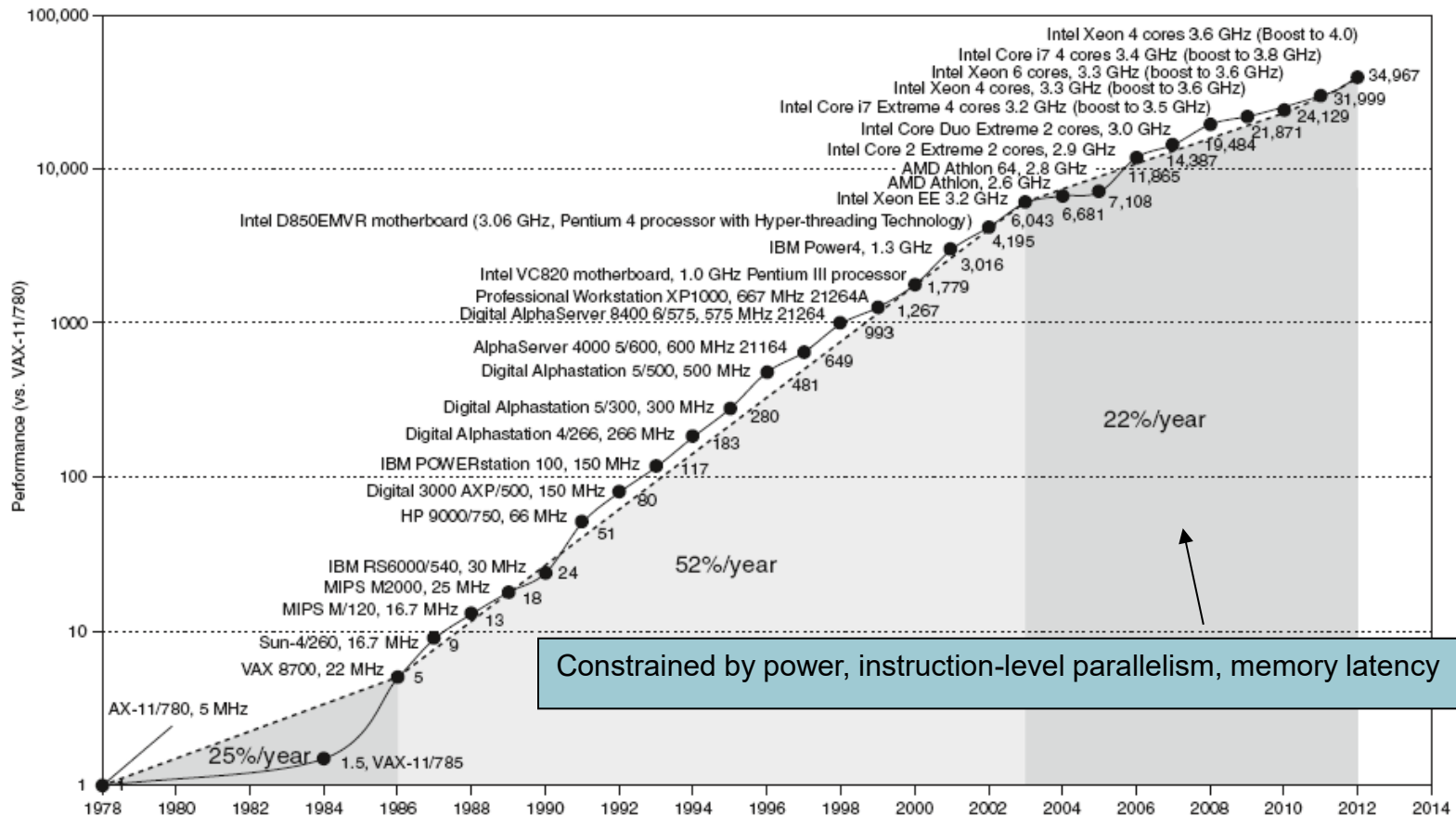
# Performance and Exec Time

- Which is better A or B? time to run program
  - 10s on A, 15s on B
  - $\text{Execution Time}_B / \text{Execution Time}_A$   
 $= 15\text{s} / 10\text{s} = 1.5$
  - So A is 1.5 times faster than B

$$\begin{aligned} \text{Performance}_x / \text{Performance}_y \\ = \text{Execution time}_y / \text{Execution time}_x = n \end{aligned}$$

- Define Performance =  $1/\text{Execution Time}$
- “X is  $n$  time faster than Y”

# Uniprocessor Performance

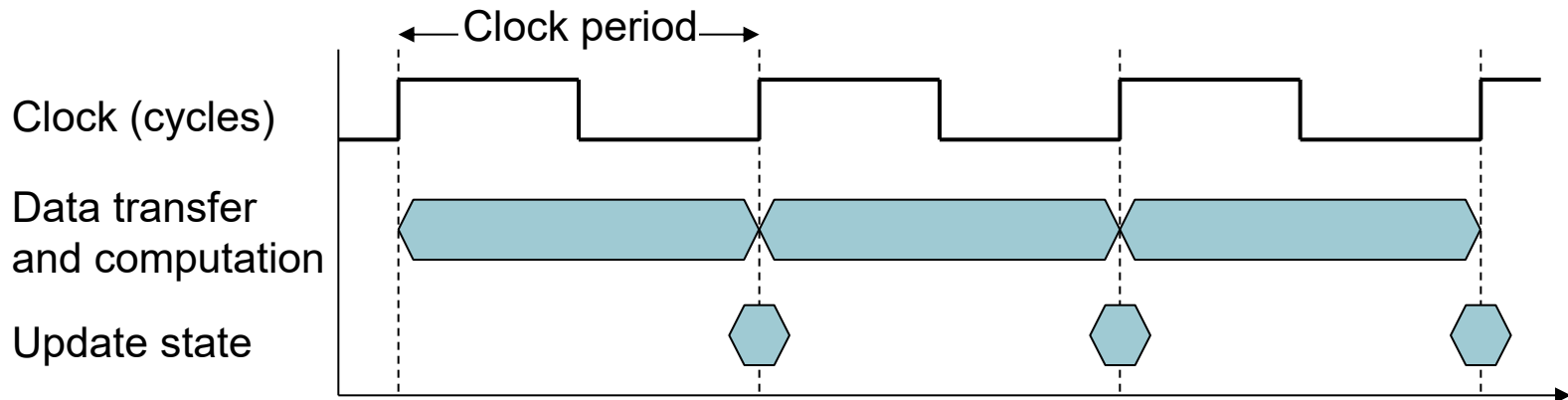


# Measuring Execution Time

- Elapsed time
  - Total response time, including all aspects
    - Processing, I/O, OS overhead, idle time
  - Determines system performance
- **We focus on CPU time**
  - Time spent processing a given job
    - Discounts I/O time, other jobs' shares
  - Comprises user & system CPU time
  - Different programs are affected differently by CPU and system performance

# CPU Clocking

- Operation of digital hardware governed by a constant-rate clock



- Clock frequency (rate): cycles per second
  - e.g.,  $4.0\text{GHz} = 4000\text{MHz} = 4.0 \times 10^9\text{Hz}$
- Clock period: duration of a clock cycle
  - e.g.,  $250\text{ps} = 0.25\text{ns} = 250 \times 10^{-12}\text{s}$

# CPU Time

$$\begin{aligned}\text{CPU Time} &= \text{CPU Clock Cycles} \times \text{Clock Cycle Time} \\ &= \frac{\text{CPU Clock Cycles}}{\text{Clock Rate}}\end{aligned}$$

- Performance improved by
  - Reducing number of clock cycles
  - Increasing clock rate
  - Hardware designer must often trade off clock rate against cycle count



# Compare

Which computer is faster?

i7 - 2.5 GHz

Atom – 3 GHz

Price

Performance

# Compare

A program P1 has 1000 instructions,  
and P2 has 2000 instructions.

Which is faster?

$x = 2 * x + 1;$

$A[1] = 1;$

Usually different instruction classes take  
different numbers of cycles.

For example,

Adding – 1 cycle

Multi

For the multi-cycle MIPS,  
there are five types of instructions:

- Load (5 cycles)
- Store (4 cycles)
- R-type (4 cycles)
- Branch (3 cycles)
- Jump (3 cycles)

If a program has:

50% load instructions, 25% store instructions

15% R-type instructions, 8% branch instructions

2% jump instructions, then, the CPI is:

# Instruction Count and CPI

Clock Cycles = Instruction Count  $\times$  Cycles per Instruction

CPU Time = Instruction Count  $\times$  CPI  $\times$  Clock Cycle Time

$$= \frac{\text{Instruction Count} \times \text{CPI}}{\text{Clock Rate}}$$

- Instruction Count for a program
  - Determined by program, ISA and compiler
- **Average CPI**
  - Determined by CPU hardware
  - If different instructions have different CPI
    - Average CPI affected by instruction mix

# CPI Example

- Computer A: Cycle Time = 250ps, CPI = 2.0
- Computer B: Cycle Time = 500ps, CPI = 1.2
- Same ISA
- Which is faster (same program) , by how much?

$$\begin{aligned}\text{CPU Time}_A &= \text{Instruction Count} \times \text{CPI}_A \times \text{Cycle Time}_A \\ &= 1 \times 2.0 \times 250\text{ps} = 1 \times 500\text{ps} \end{aligned}$$

A is faster...

$$\begin{aligned}\text{CPU Time}_B &= \text{Instruction Count} \times \text{CPI}_B \times \text{Cycle Time}_B \\ &= 1 \times 1.2 \times 500\text{ps} = 1 \times 600\text{ps}\end{aligned}$$

$$\frac{\text{CPU Time}_B}{\text{CPU Time}_A} = \frac{1 \times 600\text{ps}}{1 \times 500\text{ps}} = 1.2$$

...by this much

# CPI in More Detail

- If different instruction classes take different numbers of cycles

$$\text{Clock Cycles} = \sum_{i=1}^n (\text{CPI}_i \times \text{Instruction Count}_i)$$

- Weighted average CPI

$$\text{CPI} = \frac{\text{Clock Cycles}}{\text{Instruction Count}} = \sum_{i=1}^n \left( \text{CPI}_i \times \frac{\text{Instruction Count}_i}{\text{Instruction Count}} \right)$$

Relative frequency

# CPI Example

- Alternative compiled code sequences using instructions in classes A, B, C

Class	A	B	C
CPI for class	1	2	3
IC in sequence 1	2	1	2
IC in sequence 2	4	1	1

- Sequence 1: IC = 5

- Clock Cycles  
 $= 2 \times 1 + 1 \times 2 + 2 \times 3$   
 $= 10$
- Avg. CPI =  $10/5 = 2.0$

- Sequence 2: IC = 6

- Clock Cycles  
 $= 4 \times 1 + 1 \times 2 + 1 \times 3$   
 $= 9$
- Avg. CPI =  $9/6 = 1.5$

# Performance Summary

## The BIG Picture

$$\text{CPU Time} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Clock cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Clock cycle}}$$

- Performance depends on
  - Algorithm: affects IC, possibly CPI
  - Programming language: affects IC, CPI
  - Compiler: affects IC, CPI
  - Instruction set architecture: affects IC, CPI,  $T_c$



## Consider the following Narrative:

There are two machines, M1 and M2 with different ISAs. Three classes of instructions A, B, and C are included in both instruction sets. M1 has a clock rate of 2.25 GHz and M2 has a clock rate of 3 GHz. The number of cycles required for each instruction class in each machines are given below. A typical program is compiled (by similar compilers) into 1 million and 2 million instructions in M1 and M2 respectively. The frequencies of instructions in each instruction class in each machine are also given below.

Instruction Class	Machine M1		Machine M2	
	Cycles/Instructions	Frequency	Cycles/Instructions	Frequency
A	1	60%	1	50%
B	2	30%	3	30%
C	3	10%	3	20%
CLOCK RATE	2.25 GHZ		3 GHZ	
# of ML Instructions	1 million		2 million	
Average CPI				
MIPS				
Execution Time				

Find Average CPIs, MIPS ratio & Exec time ratio?

Which is faster, M1 or M2?

# Multiprocessors

- Multicore microprocessors
  - More than one processor per chip
- Requires explicitly parallel programming
  - Compare with instruction level parallelism
    - Hardware executes multiple instructions at once
    - Hidden from the programmer
  - Hard to do
    - Programming for performance
    - Load balancing
    - Optimizing communication and synchronization

# SPEC CPU Benchmark

- Programs used to measure performance
  - Supposedly typical of actual workload
- Standard Performance Evaluation Corp (SPEC)
  - Develops benchmarks for CPU, I/O, Web, ...
- SPEC CPU2006
  - Elapsed time to execute a selection of programs
    - Negligible I/O, so focuses on CPU performance
  - Normalize relative to reference machine
  - Summarize as geometric mean of performance ratios
    - CINT2006 (integer) and CFP2006 (floating-point)

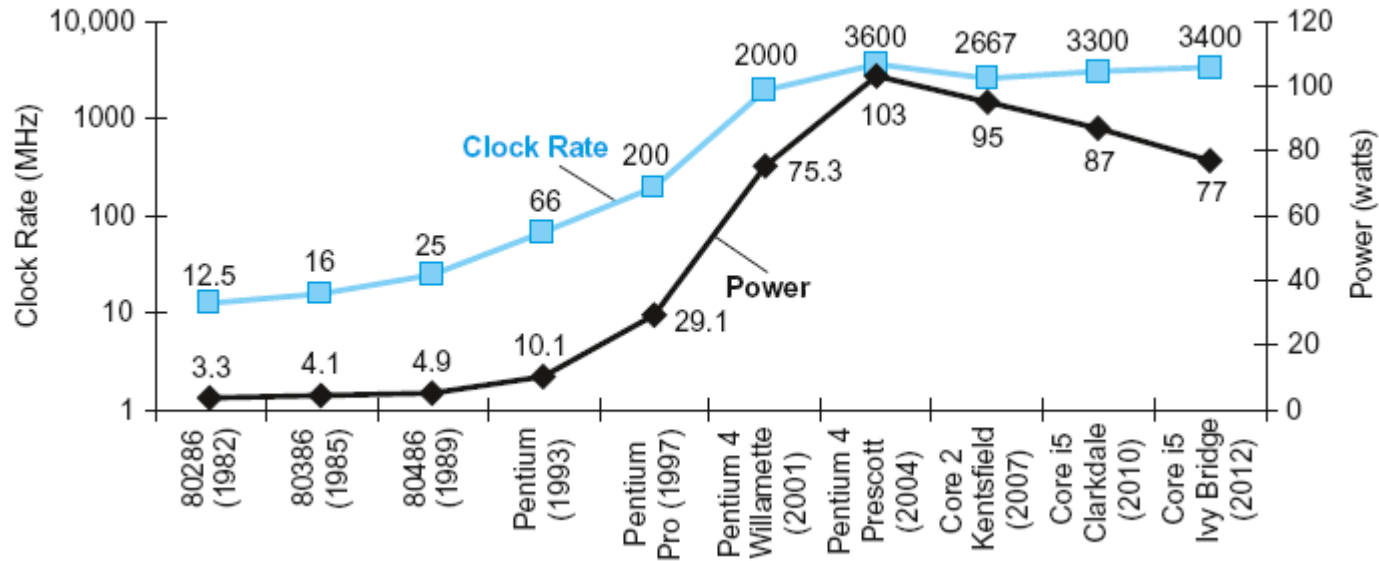
$$\sqrt[n]{\prod_{i=1}^n \text{Execution time ratio}_i}$$

# CINT2006 for Intel Core i7 920

Description	Name	Instruction Count x 10 <sup>9</sup>	CPI	Clock cycle time (seconds x 10 <sup>-9</sup> )	Execution Time (seconds)	Reference Time (seconds)	SPECratio
Interpreted string processing	perl	2252	0.60	0.376	508	9770	19.2
Block-sorting compression	bzip2	2390	0.70	0.376	629	9650	15.4
GNU C compiler	gcc	794	1.20	0.376	358	8050	22.5
Combinatorial optimization	mcf	221	2.66	0.376	221	9120	41.2
Go game (AI)	go	1274	1.10	0.376	527	10490	19.9
Search gene sequence	hmmer	2616	0.60	0.376	590	9330	15.8
Chess game (AI)	sjeng	1948	0.80	0.376	586	12100	20.7
Quantum computer simulation	libquantum	659	0.44	0.376	109	20720	190.0
Video compression	h264avc	3793	0.50	0.376	713	22130	31.0
Discrete event simulation library	omnetpp	367	2.10	0.376	290	6250	21.5
Games/path finding	astar	1250	1.00	0.376	470	7020	14.9
XML parsing	xalancbmk	1045	0.70	0.376	275	6900	25.1
Geometric mean	–	–	–	–	–	–	25.7

<https://www.spec.org/cpu2006/CINT2006/>

# Power Trends



- In CMOS IC technology

$$\text{Power} = \text{Capacitive load} \times \text{Voltage}^2 \times \text{Frequency}$$

×30

5V → 1V

×1000

# Reducing Power

- Suppose a new CPU has
  - 85% of capacitive load of old CPU
  - 15% voltage and 15% frequency reduction

$$\frac{P_{\text{new}}}{P_{\text{old}}} = \frac{C_{\text{old}} \times 0.85 \times (V_{\text{old}} \times 0.85)^2 \times F_{\text{old}} \times 0.85}{C_{\text{old}} \times V_{\text{old}}^2 \times F_{\text{old}}} = 0.85^4 = 0.52$$

- The power wall
  - We can't reduce voltage further
  - We can't remove more heat
- How else can we improve performance?

# SPEC Power Benchmark

- Power consumption of server at different workload levels
  - Performance: ssj\_ops/sec
  - Power: Watts (Joules/sec)

$$\text{Overall ssj_ops per Watt} = \left( \sum_{i=0}^{10} \text{ssj\_ops}_i \right) / \left( \sum_{i=0}^{10} \text{power}_i \right)$$

# SPECpower\_ssj2008 for Xeon X5650

Target Load %	Performance (ssj_ops)	Average Power (Watts)
100%	865,618	258
90%	786,688	242
80%	698,051	224
70%	607,826	204
60%	521,391	185
50%	436,757	170
40%	345,919	157
30%	262,071	146
20%	176,061	135
10%	86,784	121
0%	0	80
Overall Sum	4,787,166	1,922
$\Sigma \text{ssj\_ops} / \Sigma \text{power} =$		2,490



# Pitfall: Amdahl's Law

- Improving an aspect of a computer and expecting a proportional improvement in overall performance

$$T_{\text{improved}} = \frac{T_{\text{affected}}}{\text{improvement factor}} + T_{\text{unaffected}}$$

- Example: multiply accounts for 80s/100s
  - How much improvement in multiply performance to get 5× overall?

$$20 = \frac{80}{n} + 20 \quad \text{■ Can't be done!}$$

- Corollary: make the common case fast

# Fallacy: Low Power at Idle

- Look back at i7 power benchmark
  - At 100% load: 258W
  - At 50% load: 170W (66%)
  - At 10% load: 121W (47%)
- Google data center
  - Mostly operates at 10% – 50% load
  - At 100% load less than 1% of the time
- Consider designing processors to make power proportional to load

# Pitfall: MIPS as a Performance Metric

- MIPS: Millions of Instructions Per Second
  - Doesn't account for
    - Differences in ISAs between computers
    - Differences in complexity between instructions

$$\begin{aligned}\text{MIPS} &= \frac{\text{Instruction count}}{\text{Execution time} \times 10^6} \\ &= \frac{\text{Instruction count}}{\frac{\text{Instruction count} \times \text{CPI}}{\text{Clock rate}} \times 10^6} = \frac{\text{Clock rate}}{\text{CPI} \times 10^6}\end{aligned}$$

- CPI varies between programs on a given CPU

# Next – Abstractions & ISA

## The BIG Picture

- Abstraction helps us deal with complexity
  - Hide lower-level detail
- Instruction set architecture (ISA)
  - The hardware/software interface
- Application binary interface
  - The ISA plus system software interface
- Implementation
  - The details underlying and interface

# Concluding Remarks

- Cost/performance is improving
  - Due to underlying technology development
- Hierarchical layers of abstraction
  - In both hardware and software
- Instruction set architecture
  - The hardware/software interface
- Execution time: the best performance measure
- Power is a limiting factor
  - Use parallelism to improve performance

# Compare ADD Vs LW

