



Computer Architecture
Faculty of Computer Science & Engineering - HCMUT

Chapter 1

Computer Abstractions and Technology

Binh Tran-Thanh
thanhbinh@hcmut.edu.vn



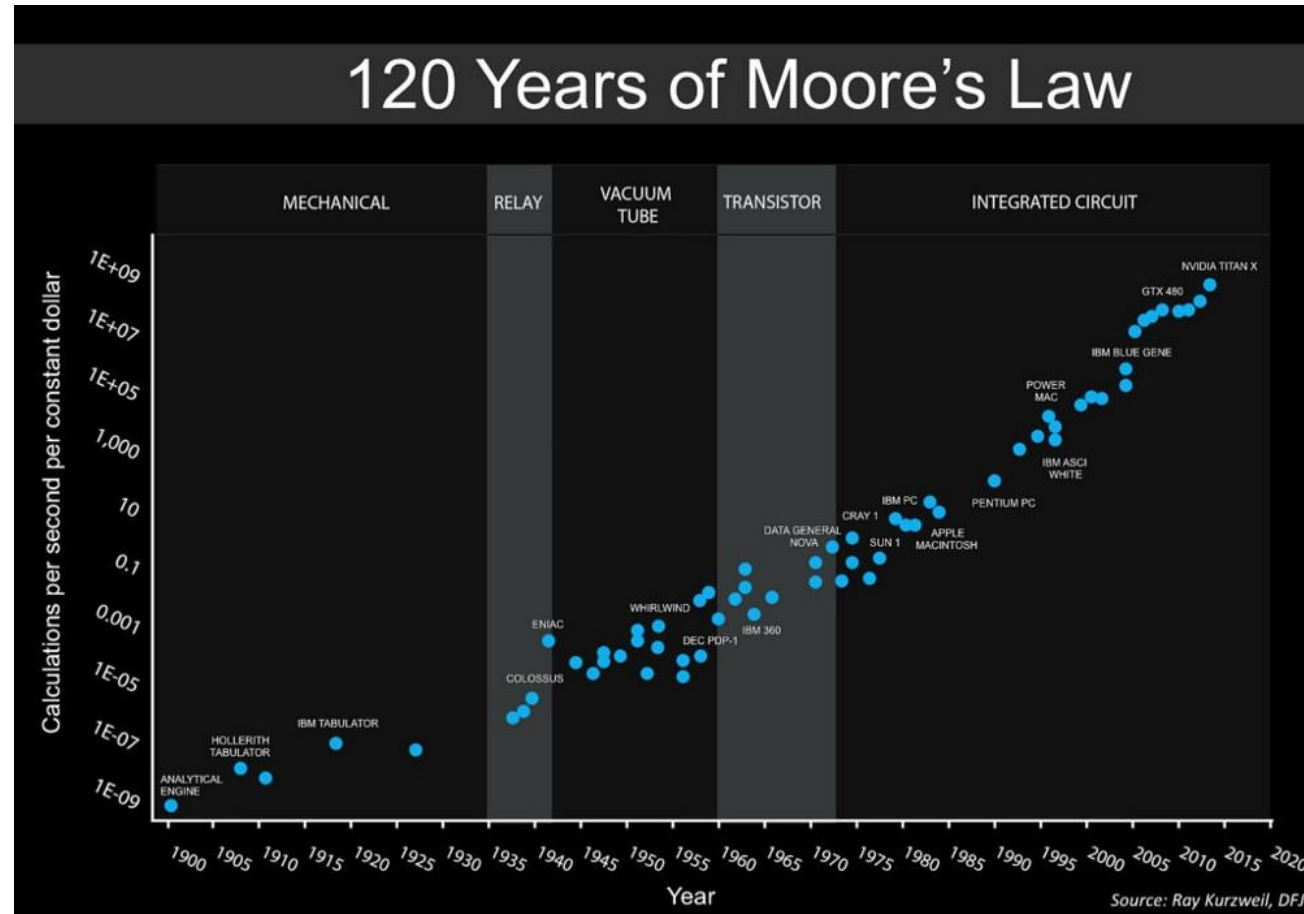
The Computer Revolution

- 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

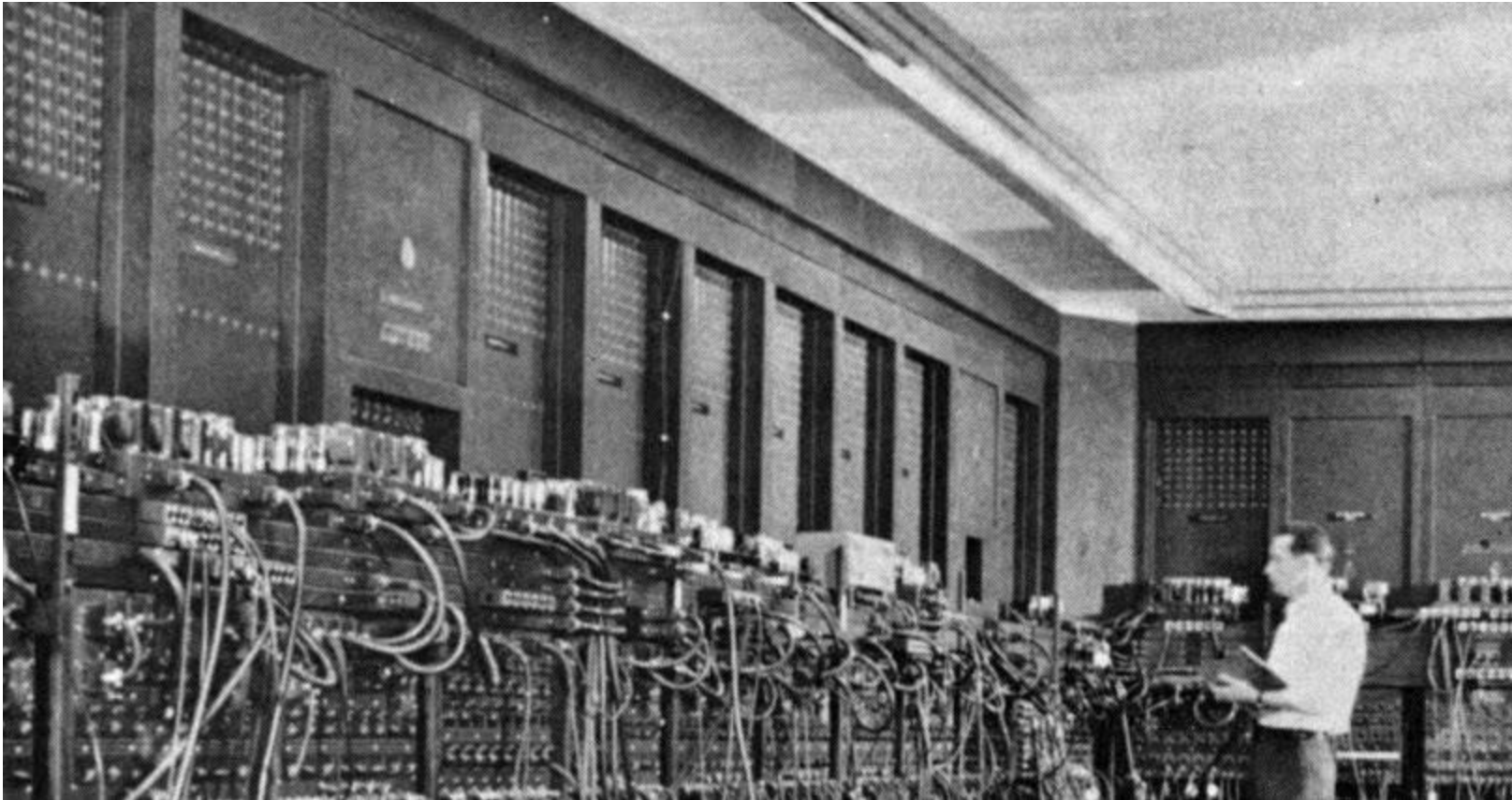
History of Computer Development

- First generation 1945 - 1955
 - vacuum tubes, plug boards
- Second generation 1955 - 1965
 - transistors, batch systems
- Third generation 1965 – 1980
 - ICs and multiprogramming
- Fourth generation 1980 – present
 - personal computers (Desk, Lap)
 - SuperComp.,
 - DataCenter, Clusters, etc.

The Moore's Law



The History: at the very beginning



ENIAC, 1943, 30 tons, 200KW, ~ 1000 ops/sec
Faculty of Computer Science and Engineering

The History: Now

Typical 2021 laptop
~ 1 kg, 10W, 10 billion ops/sec



Classes of Computers



Source: internet

Classes of Computers

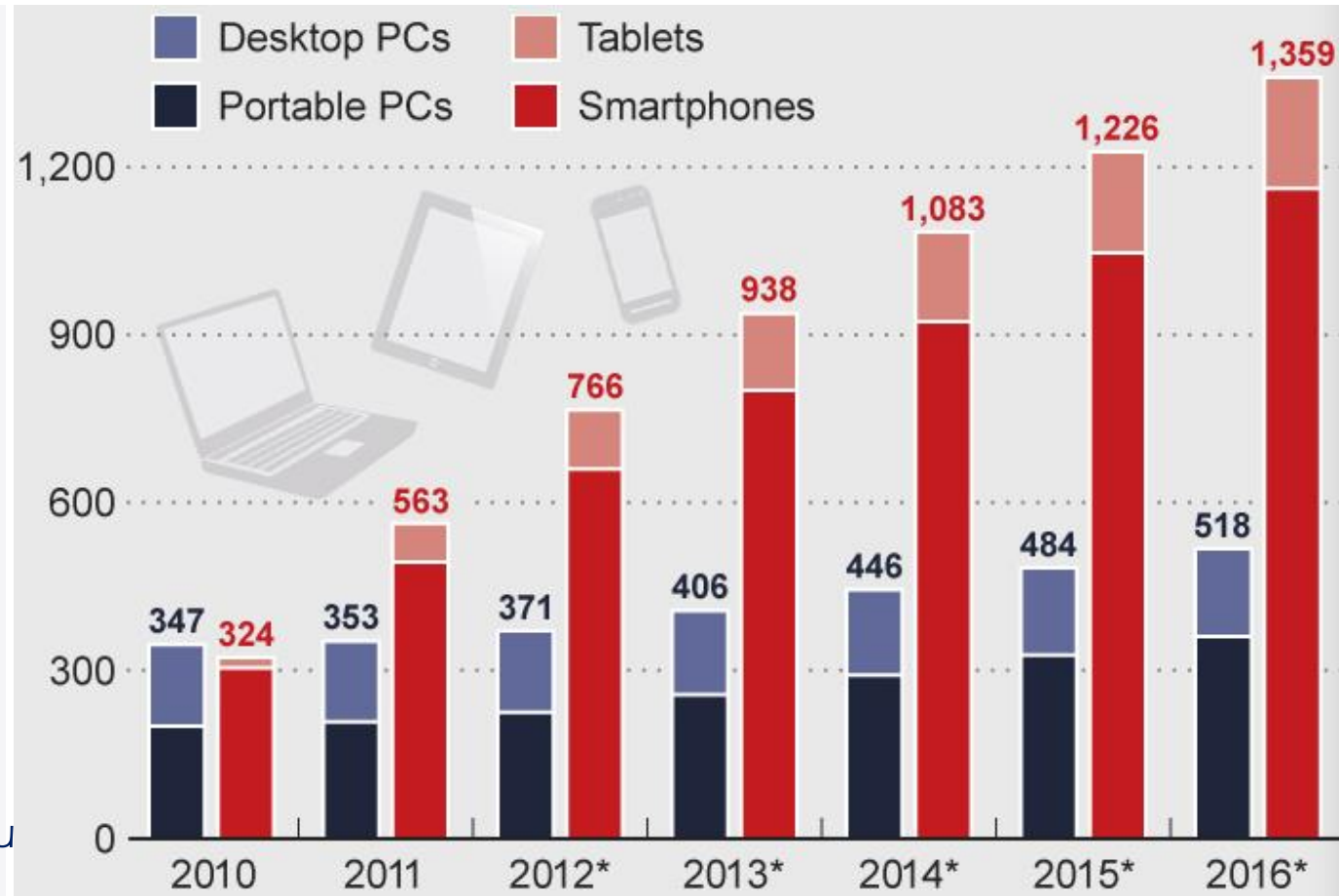
- Personal computers
 - General purpose, variety of software
 - Subject to cost/performance trade-off
- Embedded computers
 - Hidden as components of systems
 - Stringent power/performance/cost constraints

Classes of Computers

- Server computers
 - Network based
 - High capacity, performance, reliability
 - Range from small servers to building sized
- Supercomputers
 - High end scientific and engineering calculations
 - Highest capability but represent a small fraction of the overall computer market

The PostPC Era has arrived

- Your next computer is not a computer (apple)



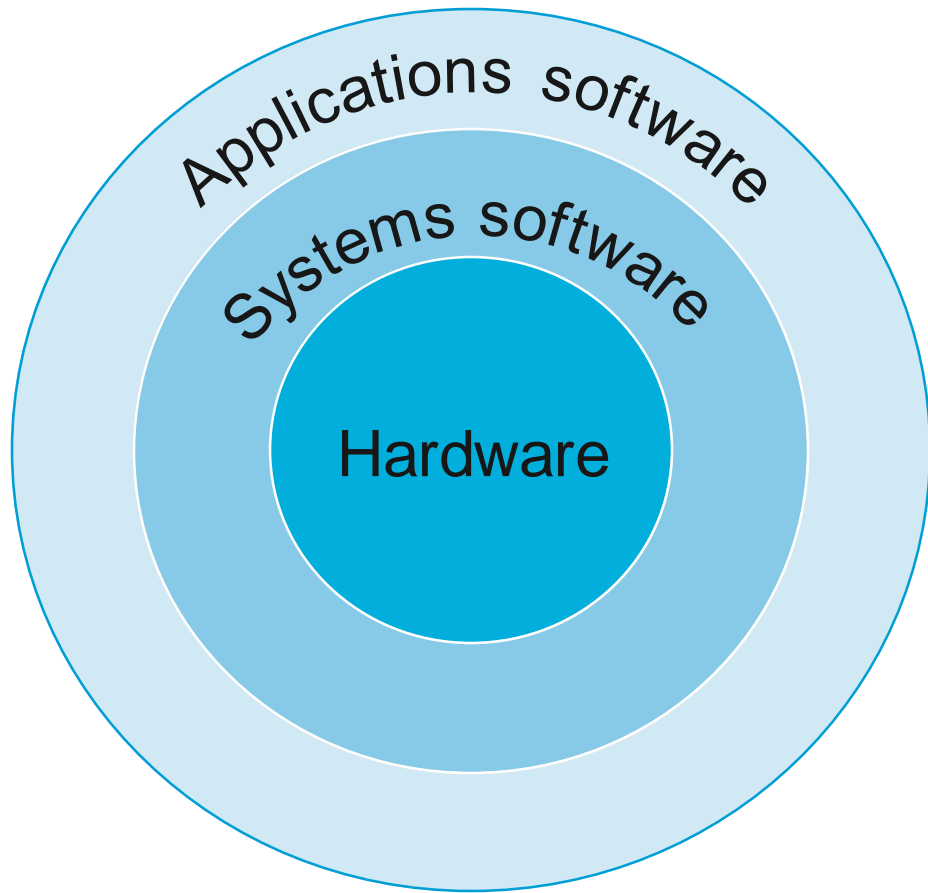
The PostPC Era

- 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
- Personal Mobile Device (PMD)
 - Battery operated
 - Connects to the Internet
 - Hundreds of dollars
 - Smart phones, tablets, electronic glasses

Understanding Performance

- 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

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
- Which layer represents for program.exe/.asm/.c?

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)

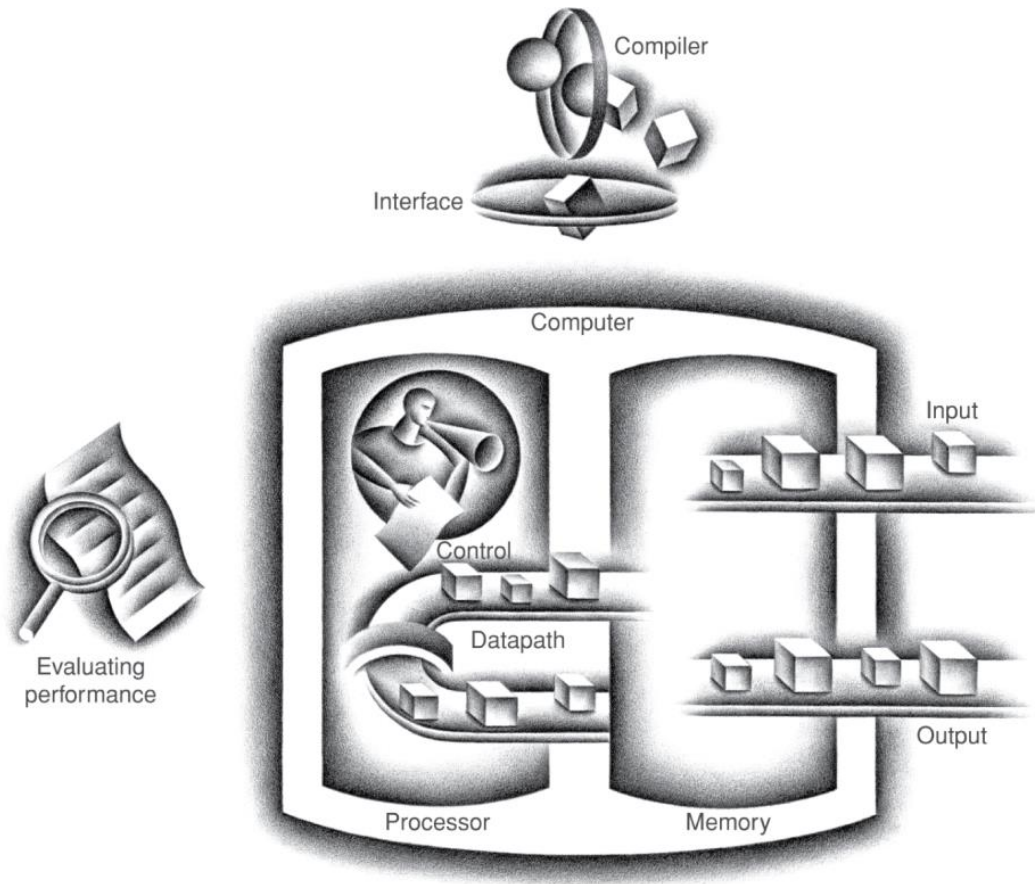
```
swap: multi $2, $5, 4  
      add  $2, $4, $2  
      lw   $15, 0($2)  
      lw   $16, 4($2)  
      sw   $16, 0($2)  
      sw   $15, 4($2)  
      jr   $31
```

Assembler

Binary
Machine
Language
(for MIPS)

```
000000001010001000000000100011000  
00000000100000100001000000100001  
10001101111000100000000000000000  
100011100001001000000000000000100  
101011100001001000000000000000000  
101011011110001000000000000000100  
00000011111000000000000000001000
```

Components of a Computer



- 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 Processor (CPU)

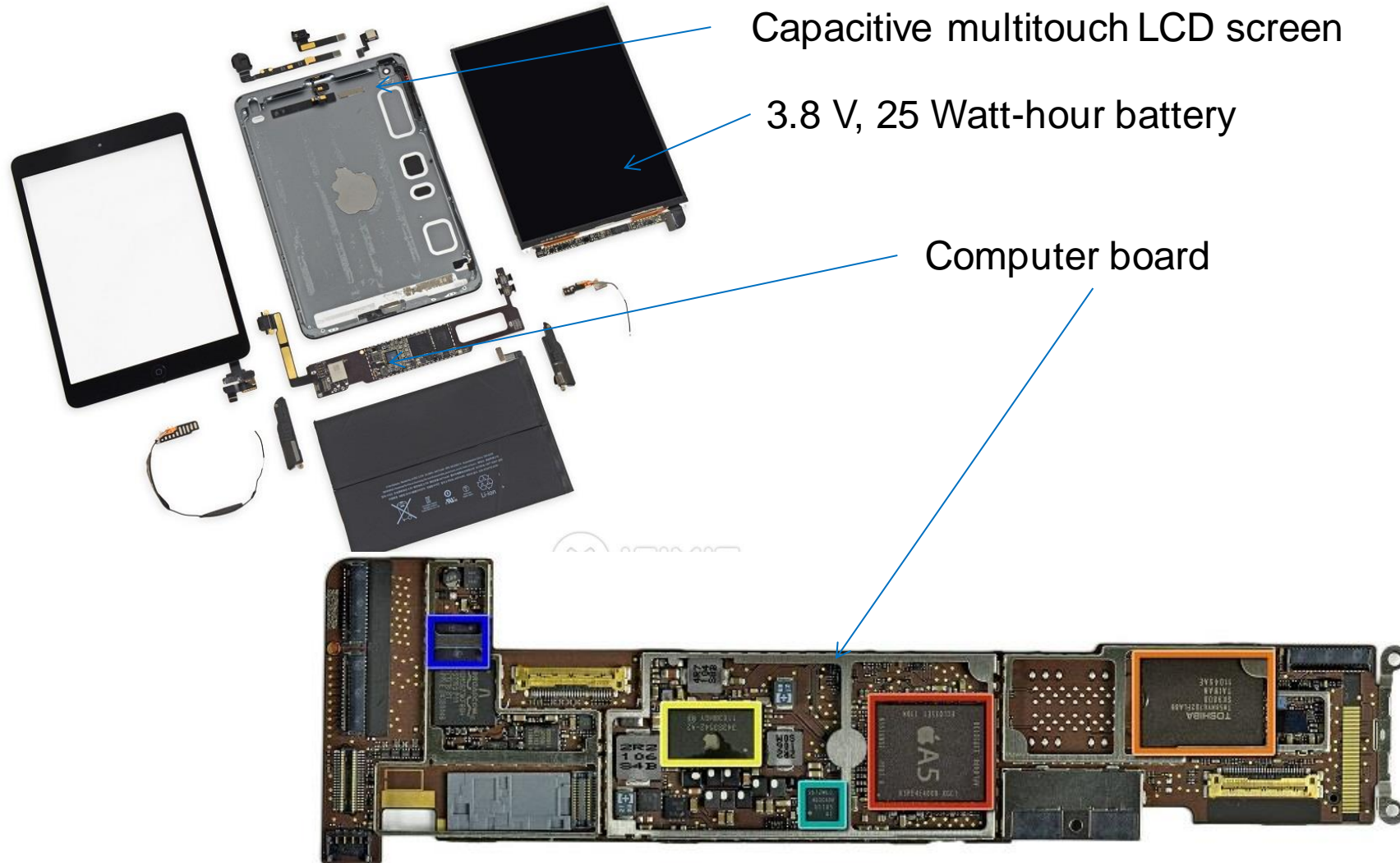
- Datapath: performs operations on data
- Control: sequences Datapath, memory, ...
- Cache memory
 - Small fast SRAM memory for immediate access to data

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

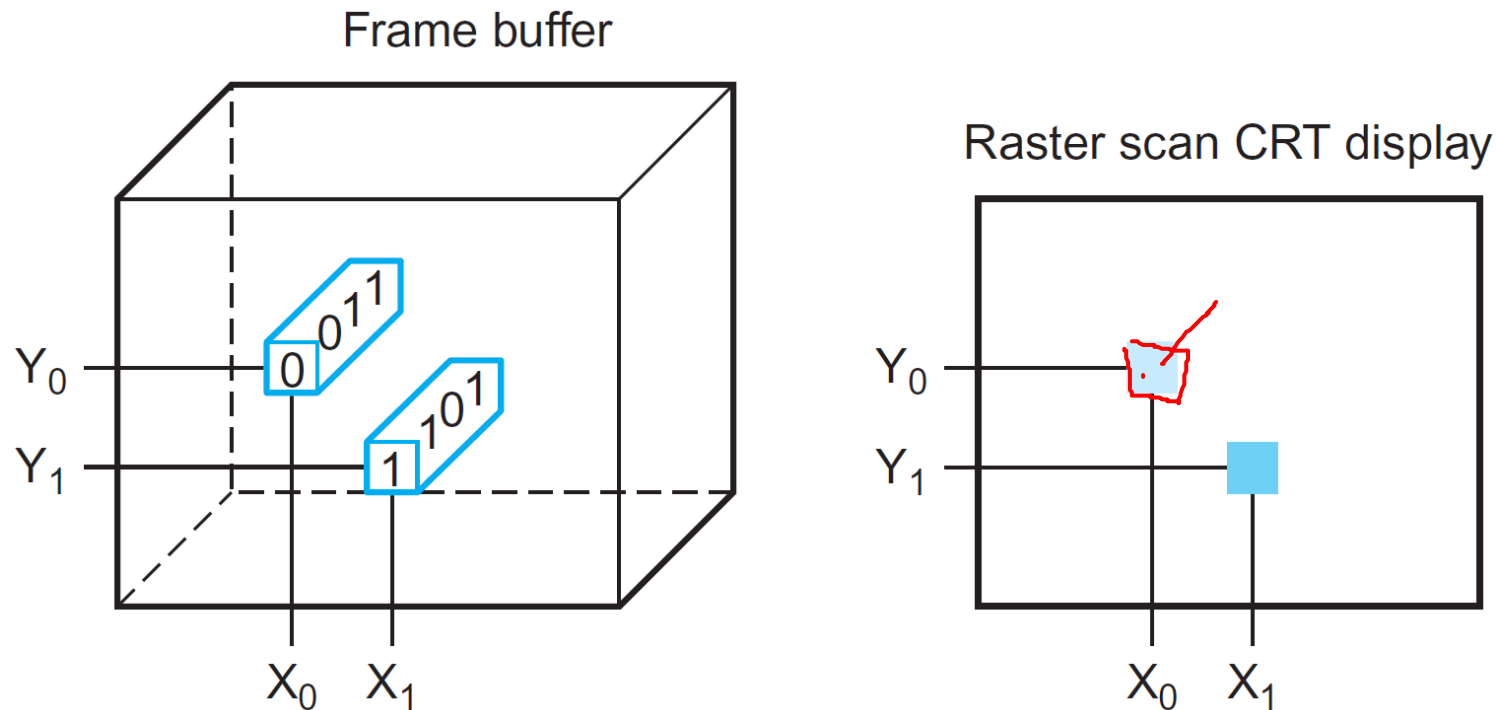


Opening the Box



Through the Looking Glass

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



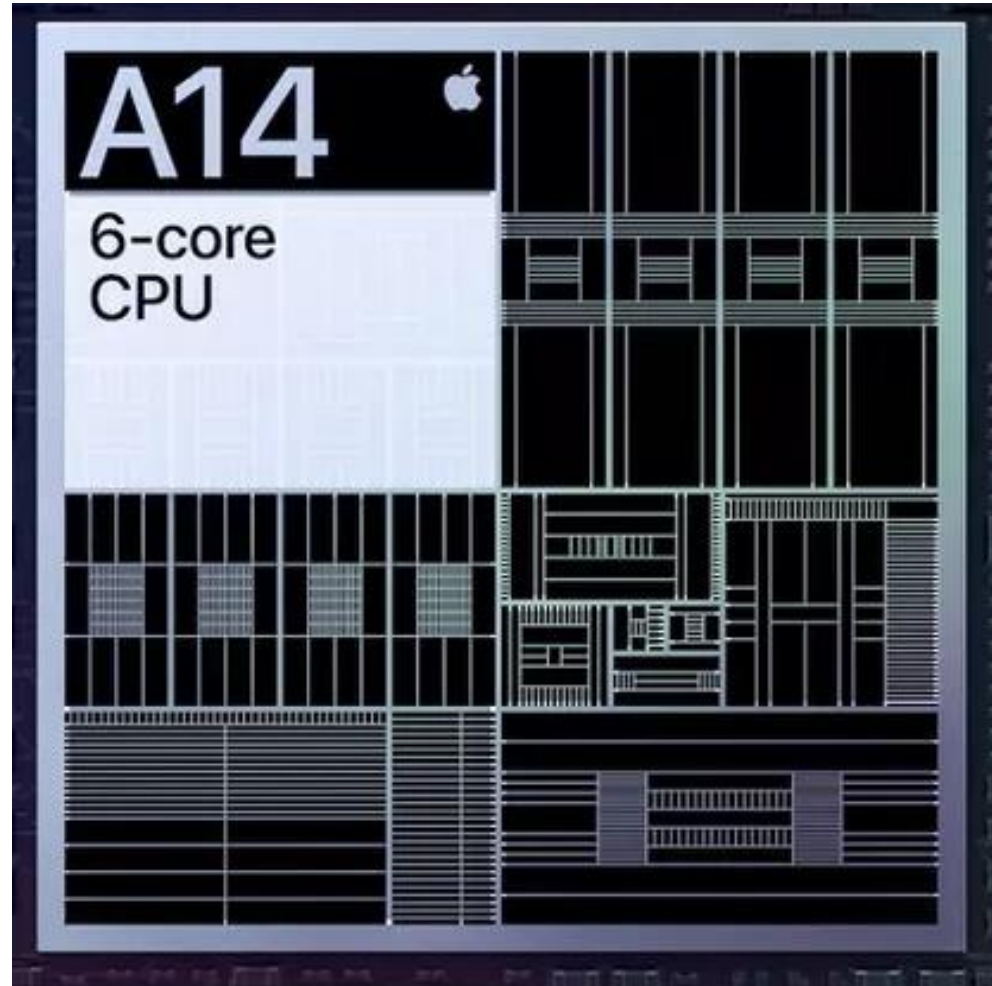
Touchscreen

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



Inside the Processor

- Apple A14



Abstractions

- 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

A Safe Place for Data

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



[This Photo](#) by Unknown Author is licensed under [CC BY](#)



Author is licensed under [CC](#)



Networks

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



Technology Trends

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

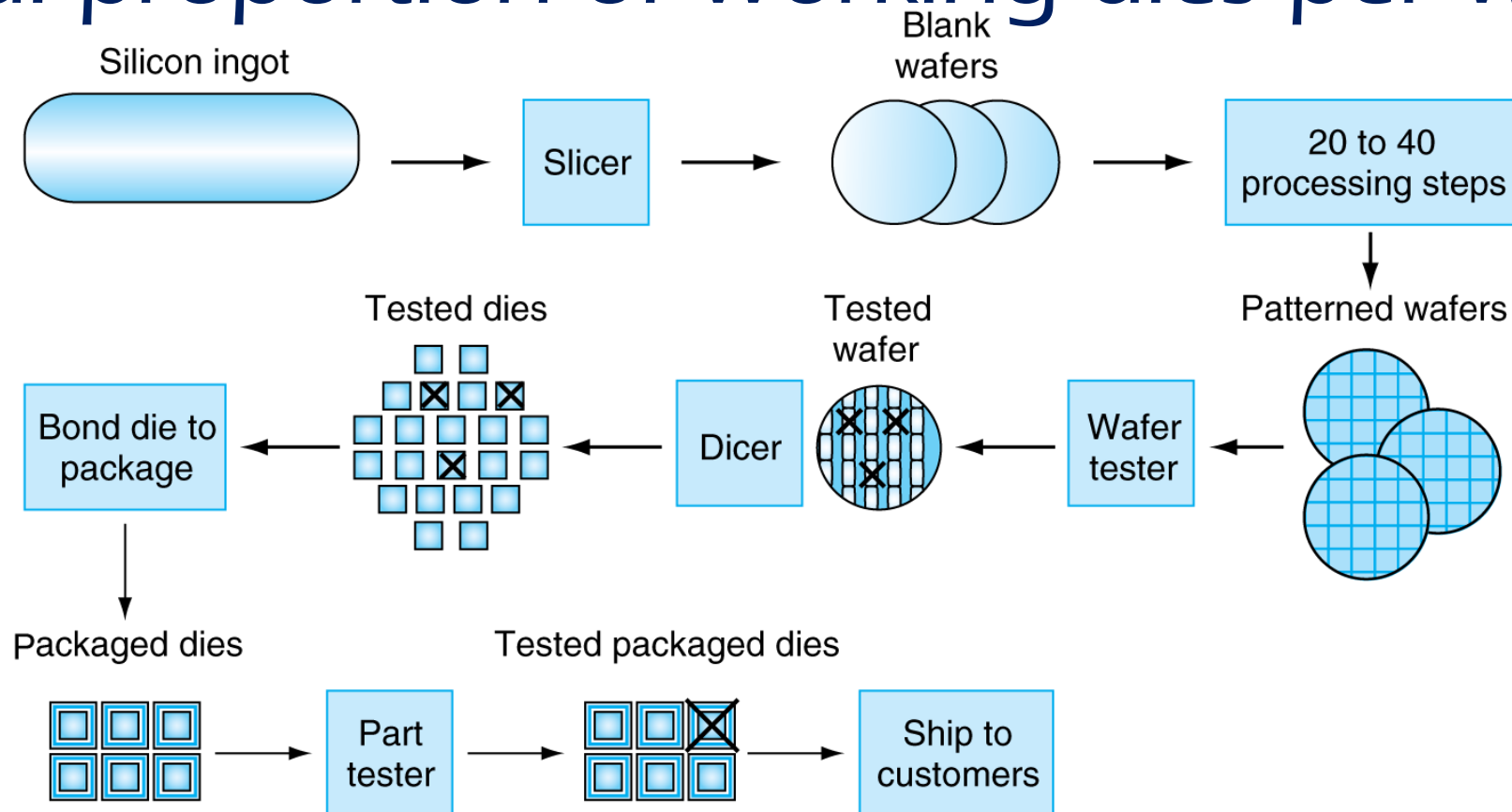
Year	Technology	Relative performance/cost
1951	Vacuum tube	1
1965	Transistor	35
1975	Integrated circuit (IC)	900
1995	Very large scale IC (VLSI)	2,400,000
2013	Ultra large scale IC	250,000,000,000

Semiconductor Technology

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

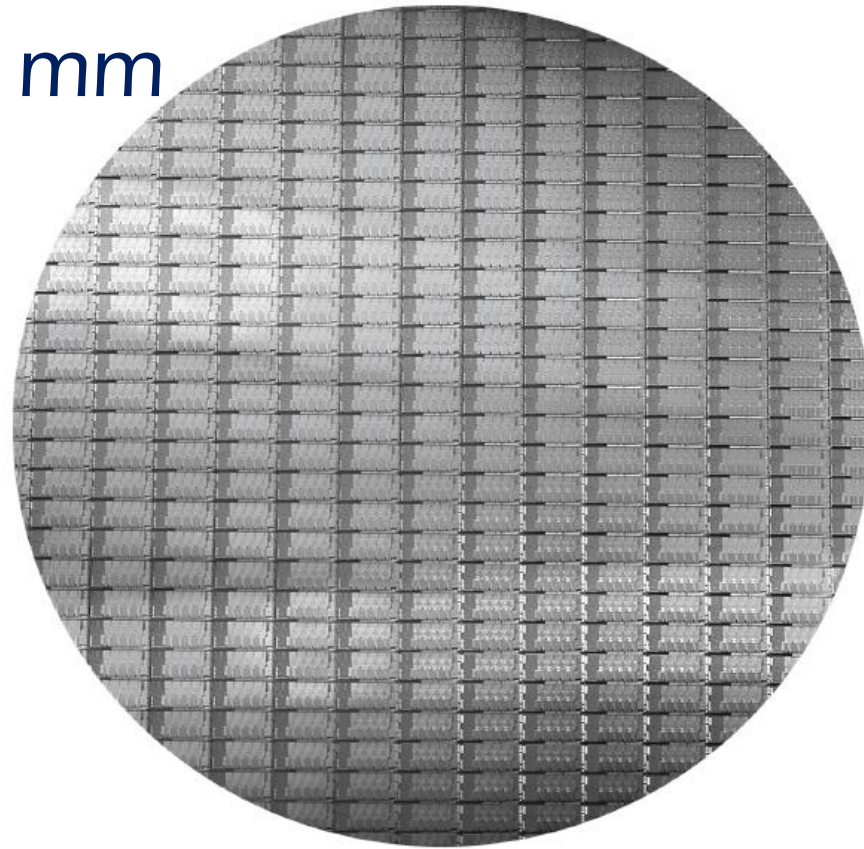
Manufacturing ICs

- 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

- 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

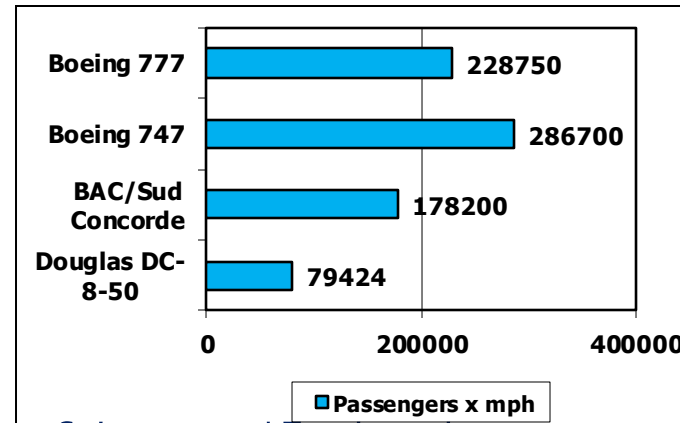
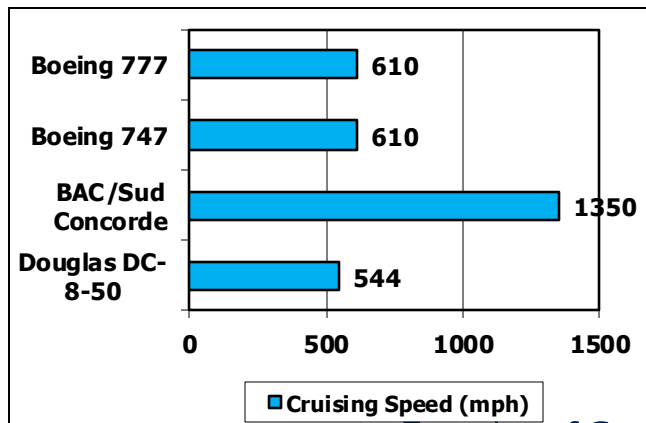
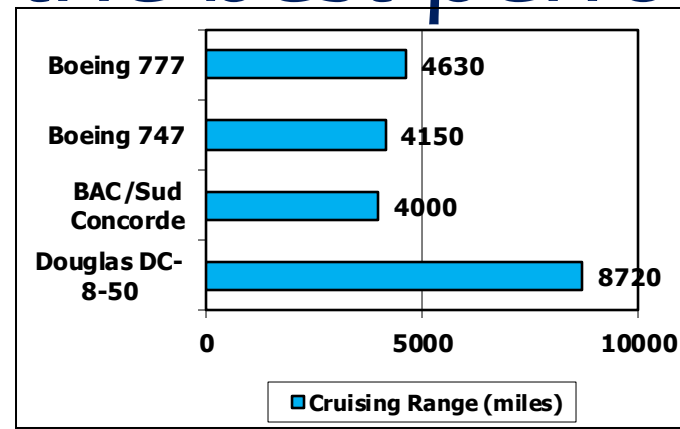
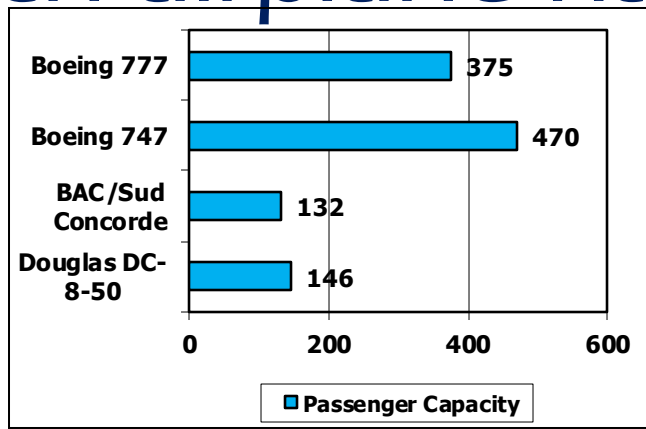
$$\text{Cost per die} = \frac{\text{Cost per wafer}}{\text{Dies per wafer} \times \text{Yield}}$$

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

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

Defining Performance

- Which airplane has the best performance?



Response Time and Throughput

- Response time
 - How long it takes to do a task
- Throughput
 - Total work done per unit time
 - e.g., tasks/transactions/... per hour
- How are response time and throughput affected by
 - Replacing the processor with a faster version?
 - Adding more processors?
- We'll focus on response time for now...

Relative Performance

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

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

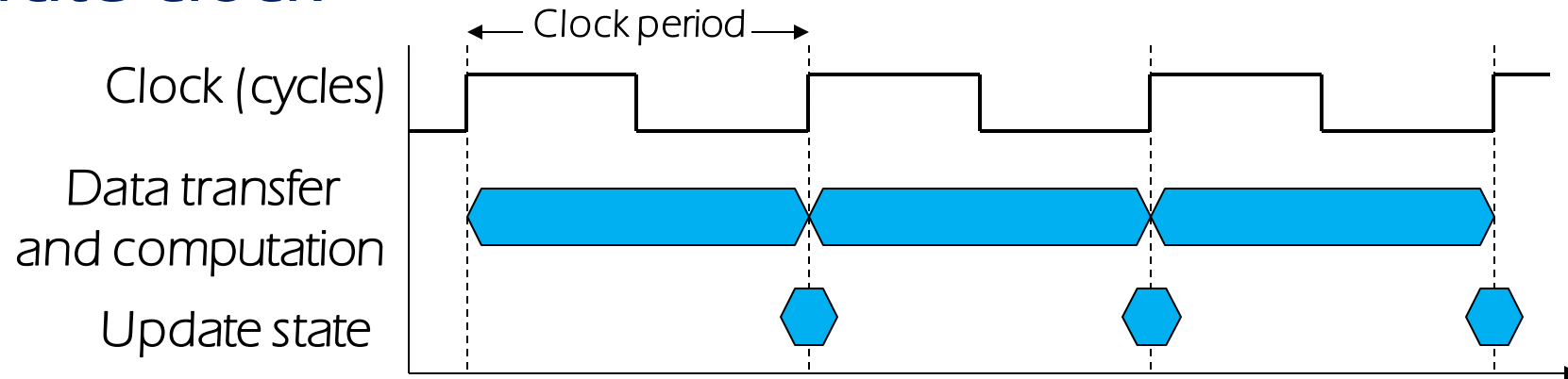
- Example: time taken to run a program
 - 10s on A, 15s on B
 - $\text{Execution Time}_B / \text{Execution Time}_A$
 $= 15s / 10s = 1.5$
 - So, A is 1.5 times faster than B

Measuring Execution Time

- Elapsed time
 - Total response time, including all aspects
 - Processing, I/O, OS overhead, idle time
 - Determines system performance
- CPU time
 - Time spent processing a given job
 - Discounts I/O time, other jobs' shares
 - Comprises user CPU time and 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 period: duration of a clock cycle
 - e.g., $250\text{ps} = 0.25\text{ns} = 250 \times 10^{-12}\text{s}$
- Clock frequency (rate): cycles per second
 - e.g., $4.0\text{GHz} = 4000\text{MHz} = 4.0 \times 10^9\text{Hz}$

CPU Time

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

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

CPU Time Example

- Computer A: 2GHz clock, 10s CPU time
- Designing Computer B
 - Aim for 6s CPU time
 - Can do faster clock, but causes $1.2 \times$ clock cycles
- How fast must Computer B clock be?

$$\text{Clock Rate}_B = \frac{\text{Clock Cycles}_B}{\text{CPU Time}_B} = \frac{1.2 \times \text{Clock Cycles}_A}{6s}$$

$$\begin{aligned}\text{Clock Cycles}_A &= \text{CPU Time}_A \times \text{Clock Rate}_A \\ &= 10s \times 2\text{GHz} = 20 \times 10^9\end{aligned}$$

$$\text{Clock Rate}_B = \frac{1.2 \times 20 \times 10^9}{6s} = \frac{24 \times 10^9}{6s} = 4\text{GHz}$$

Instruction Count and CPI

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

$\text{Clock Cycles} = \text{Instruction Count} \times \text{Cycles per Instruction}$

$\text{CPU Time} = \text{Instruction Count} \times \text{CPI} \times \text{Clock Cycle Time}$

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

CPI Example

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

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

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

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

A is faster...

...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

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

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 = $9/6 = 1.5$

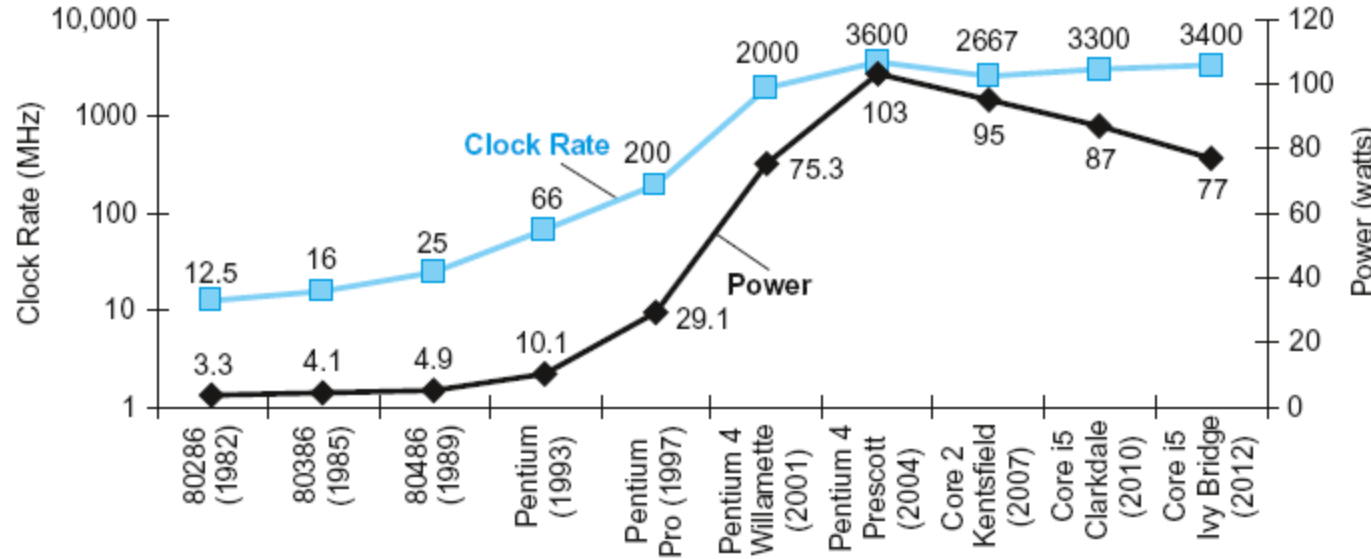
Performance Summary

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

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

Power Trends

- In CMOS IC technology



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

×30

5V → 1V

×1000

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)

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

SPEC Power Benchmark

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

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

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?

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

- Corollary: make the common case fast

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

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?

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

Exercise

- Given a program X in detail as below.

Instruction class	A	B	C	D
CPI	2	4	3	2.5
# instruction	1000	2000	3000	4000

- What is CPU time of X where it is run at 2Ghz ?
- Improving performance by reducing #instruction of B by a half. What is the speed up?
- To improve performance by changing only 1 class of instruction. What is the limitation of speed up?

Exercise

- Given a program X in detail as below.

Instruction class	A	B	C	D
CPI	2	4	3	2.5
# instruction	1000	2000	3000	4000

- What is CPU time of X where it is run at 2Ghz ? (14.5 μ sec)
- Improving performance by reducing #instruction of B by a half. What is the speed up? (1.16)
- To improve performance by changing only 1 class of instruction. What is the limitation of speed up? (1.53)

Question?

