ECEC-355: Computer Organization & Architecture

Dr. Anup K. Das

Electrical and Computer Engineering

Drexel University

Introduction

- **Instructor**
  - Dr. Anup Das
  - Email: anup.das@drexel.edu
- **Teaching Assistant**
  - Shadi Matinizadeh
  - Email: sm4884@drexel.edu

Office Hours

- **TA Hours:**
  - To be announced soon
- **Slack channel will be created**

- **Instructor Office Hours**
  - By appointment: Please email (anup.das@drexel.edu)

- **TA topics: Project related**
- **Instructor topics: Content related**

- **TA Response Time: 24 hours**
- **Instructor Response Time: 48 hours**

Course Logistics

- **Total Grade - 100 points**
  - 6 Quiz assignments (top 5 will contribute) 50
  - 3 C-based Projects 30
  - Midterm 20

- **Quizzes**
  - Each quiz will comprise of approx. 10 questions, 1 point each
  - No negative grading
  - No makeup quiz for missed quizzes

Course Logistics

- **Academic policy for grades***
  - Letter Grade percentage A+ 100–97, A 96.9 – 93, A- 92.9 – 90, B+ 89.9 – 87, B 86.9 – 83, B- 82.9 – 80, C+ 79.9 – 77, C 76.9 – 73, C- 72.9 – 70, D+ 69.9 – 67, D 66.9 – 63, F Below 63

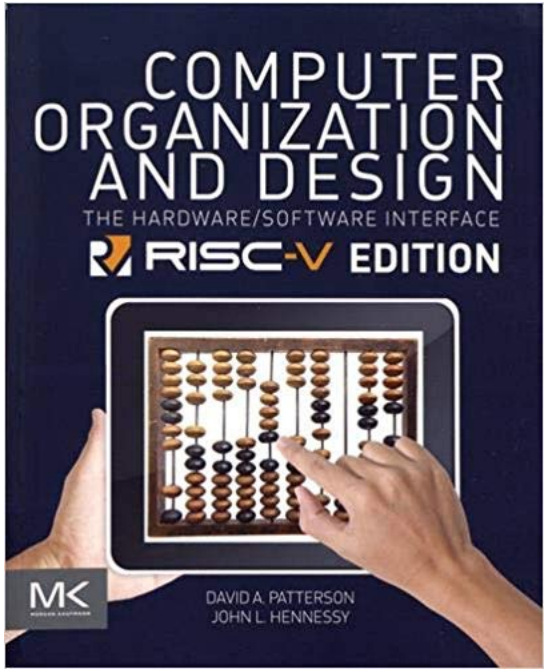
    *** The instructor reserves the right to adjust the grade percentages (e.g. based on the distribution of grades) to accommodate non-standard (low or high) distributions.**

Resources

- **Computer Organization and Design RISC-V Edition: The Hardware Software Interface (The Morgan Kaufmann Series in Computer Architecture and Design)**

- **Slides**

- **Additional resources will be made available during the course**



Practice Problems

- **Content will be based on the reference book Chapters 1, 2, 3, 4, 5, and 6.**
- **Quizzes and Midterm exam problems will be based on some variants of the exercise problems of the book**
  - Some tricks will be there, so practicing the exercise problems will be helpful

Readings

- **Chapter 1**

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

Classes of Computers

- **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

- **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

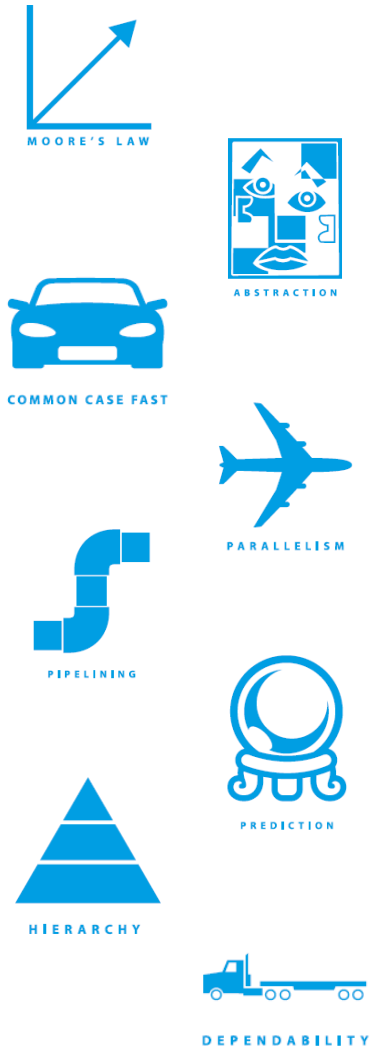
- **Personal Mobile Device (PMD)**
  - Battery operated
  - Connects to the Internet
  - Hundreds of dollars
  - Smart phones, tablets, electronic glasses
- **Cloud computing**
  - Warehouse Scale Computers (WSC)
  - Software as a Service (SaaS)
  - Portion of software run on a PMD and a portion run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

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
- **Who are responsible to improve program performance?**
  - What is parallel processing?
- **What are the emerging performance metrics for computers?**

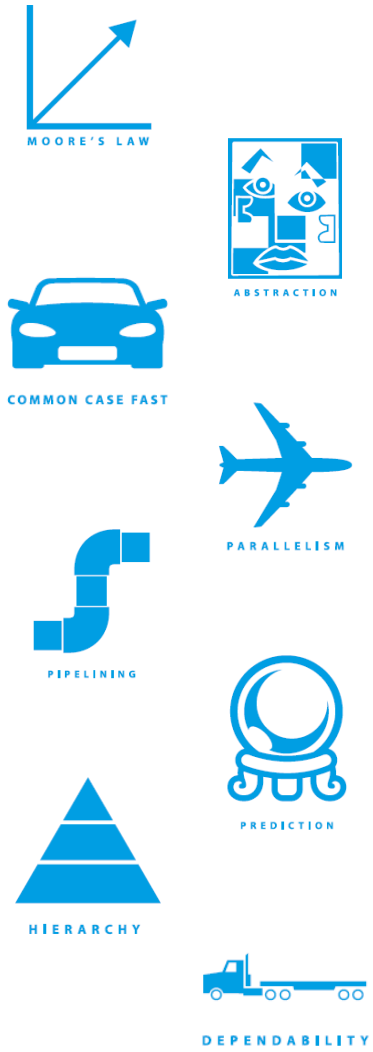
Eight Great Ideas

- 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

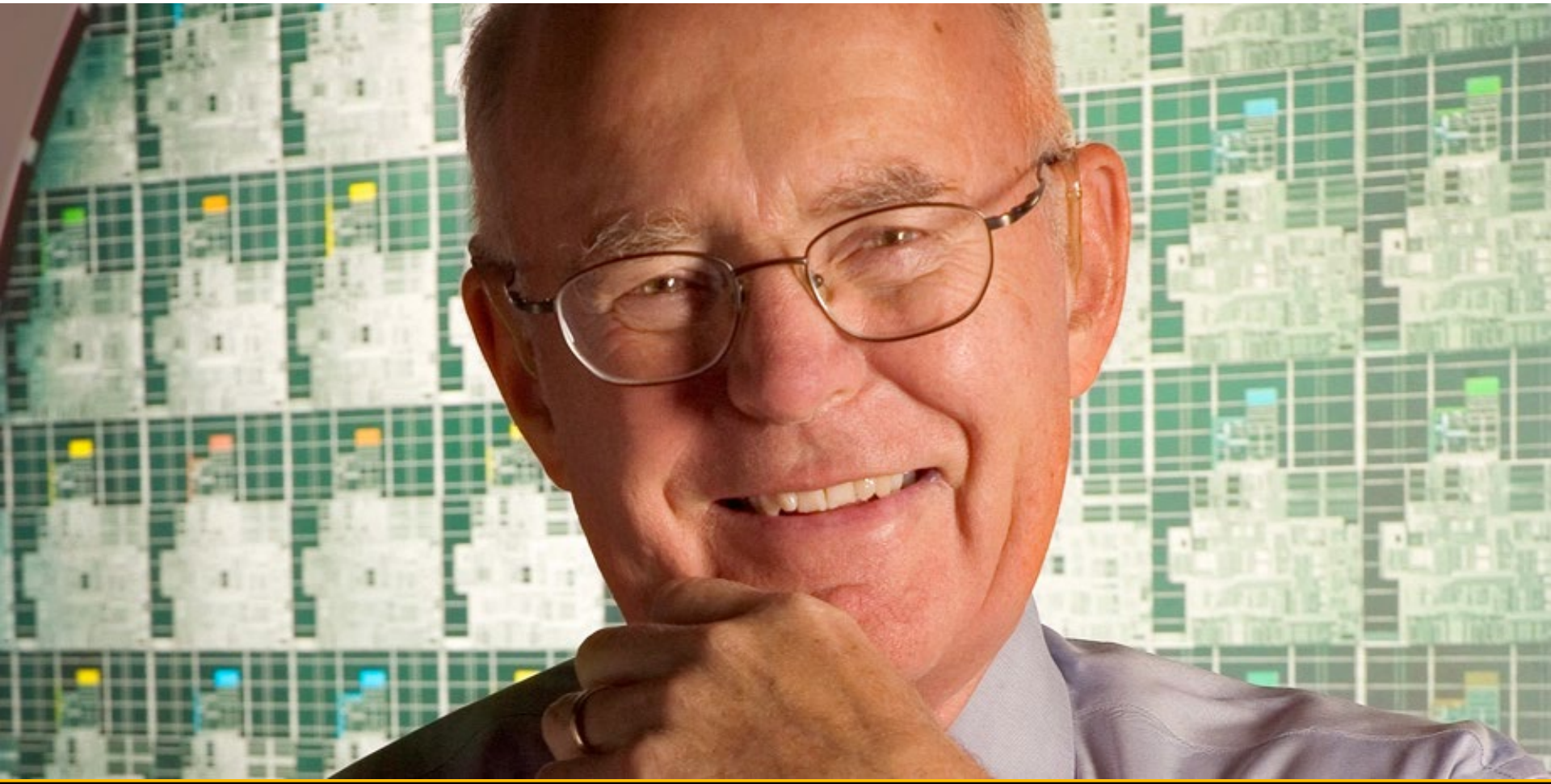


Eight Great Ideas

- 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



A Quick Intro to Gordon Moore



A Quick Intro to Gordon Moore

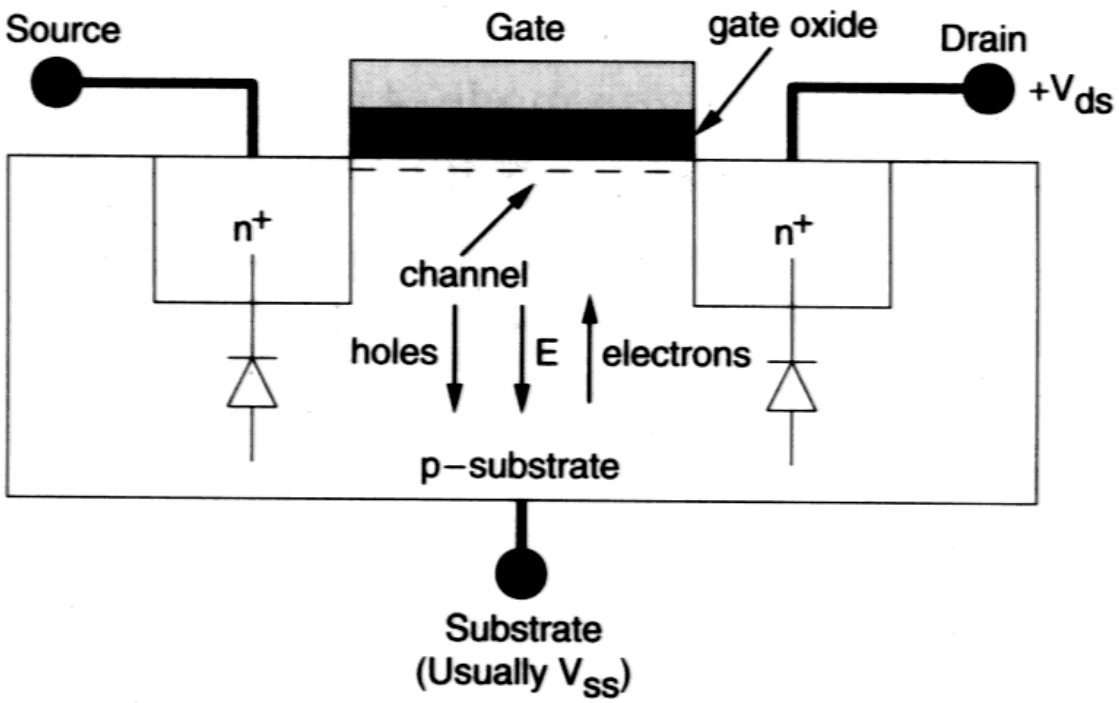
- **Co-founder of Intel**
  - Also known for Moore's Law

Why Are we Interested in Knowing Him?

- **He made an observation**
  - Number of transistors per square inch on integrated circuits had doubled every year since their invention
- **He made a prediction**
  - This trend will continue into the foreseeable future
  - Also known as Moore's Law

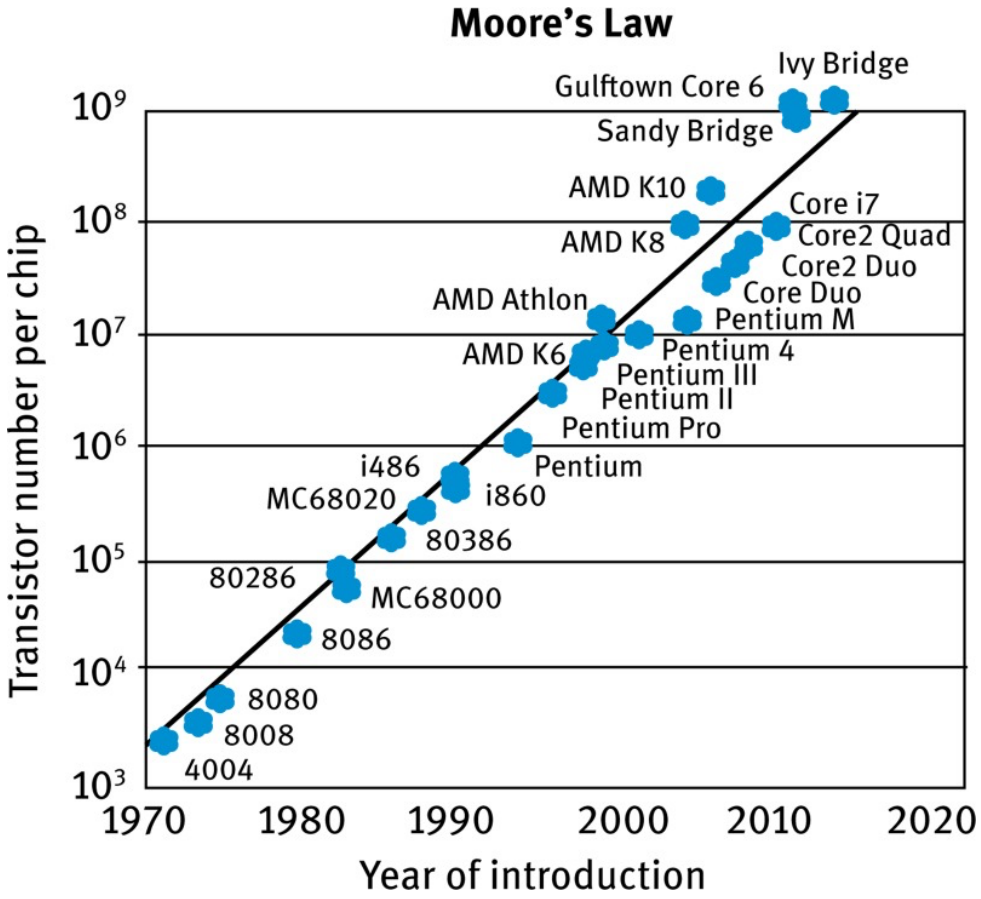
- **We tried to maintain Moore Law by**
  - Going smaller (…,130nm, 90nm, 65nm, 45nm, 22nm,…,5nm)
  - Packing more transistors per chip
  - Delivering double the performance with every new technology node

Technology nodes



Channel length = 65nm etc.

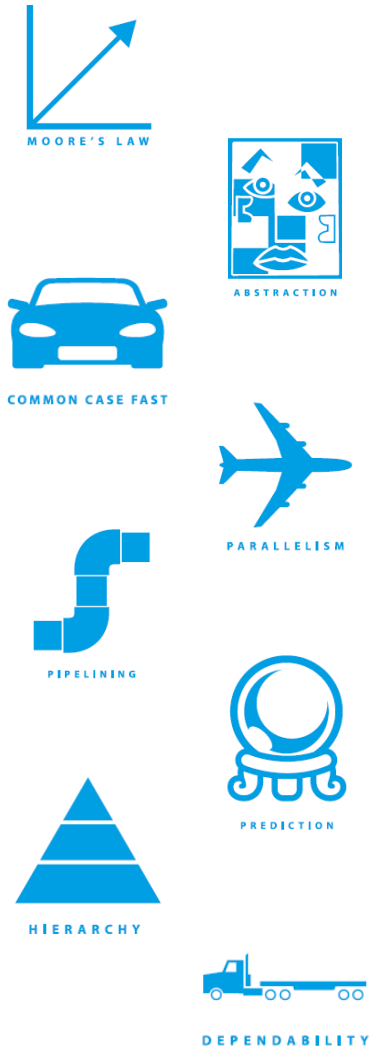
Moore's Law Driving Computer Innovations



- **Difficult to sustain the growth**
  - We will learn why

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

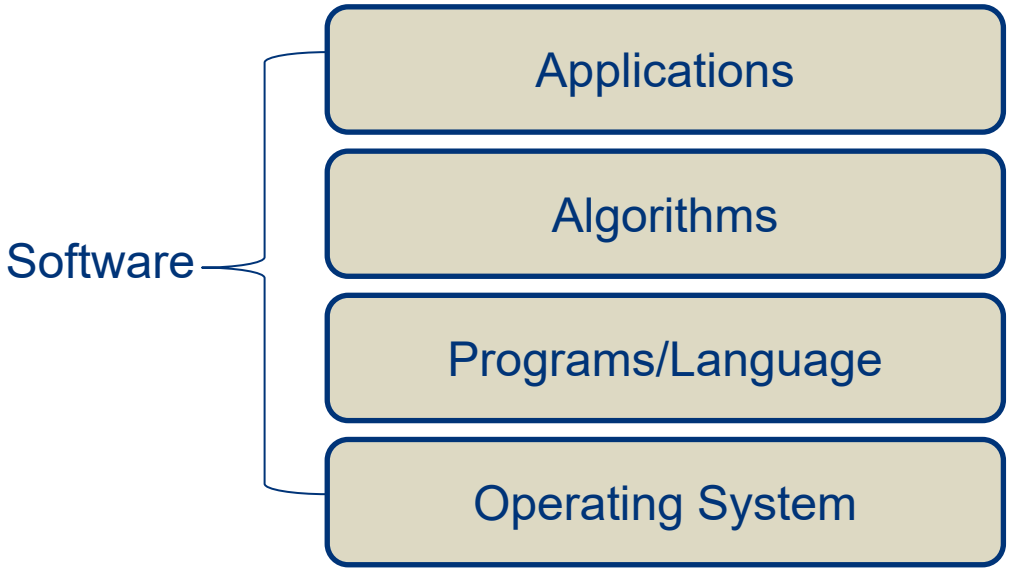


Abstraction

- **Abstraction helps us deal with complexity**
  - Hide lower-level detail

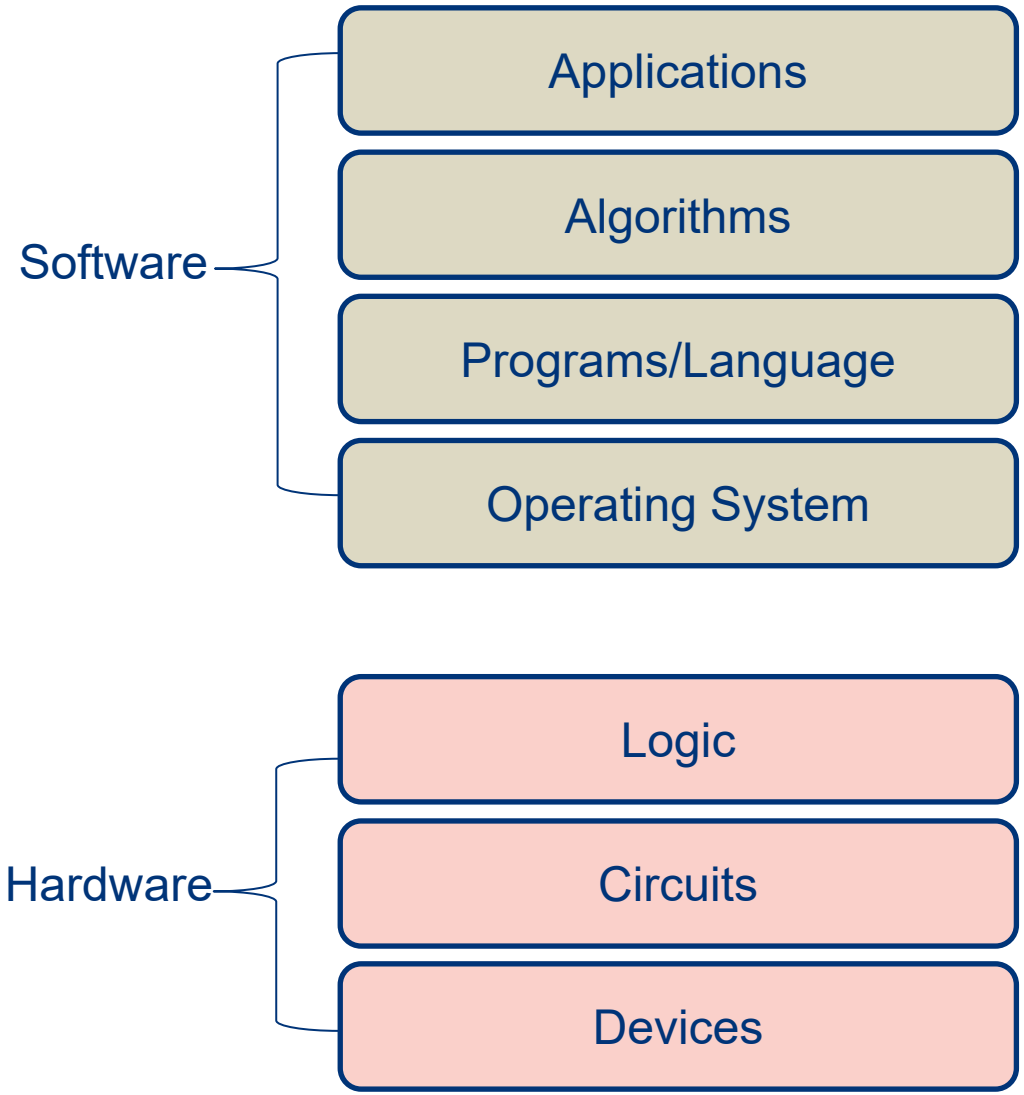
Computing Stacks

- **What are the stacks of a computing system?**



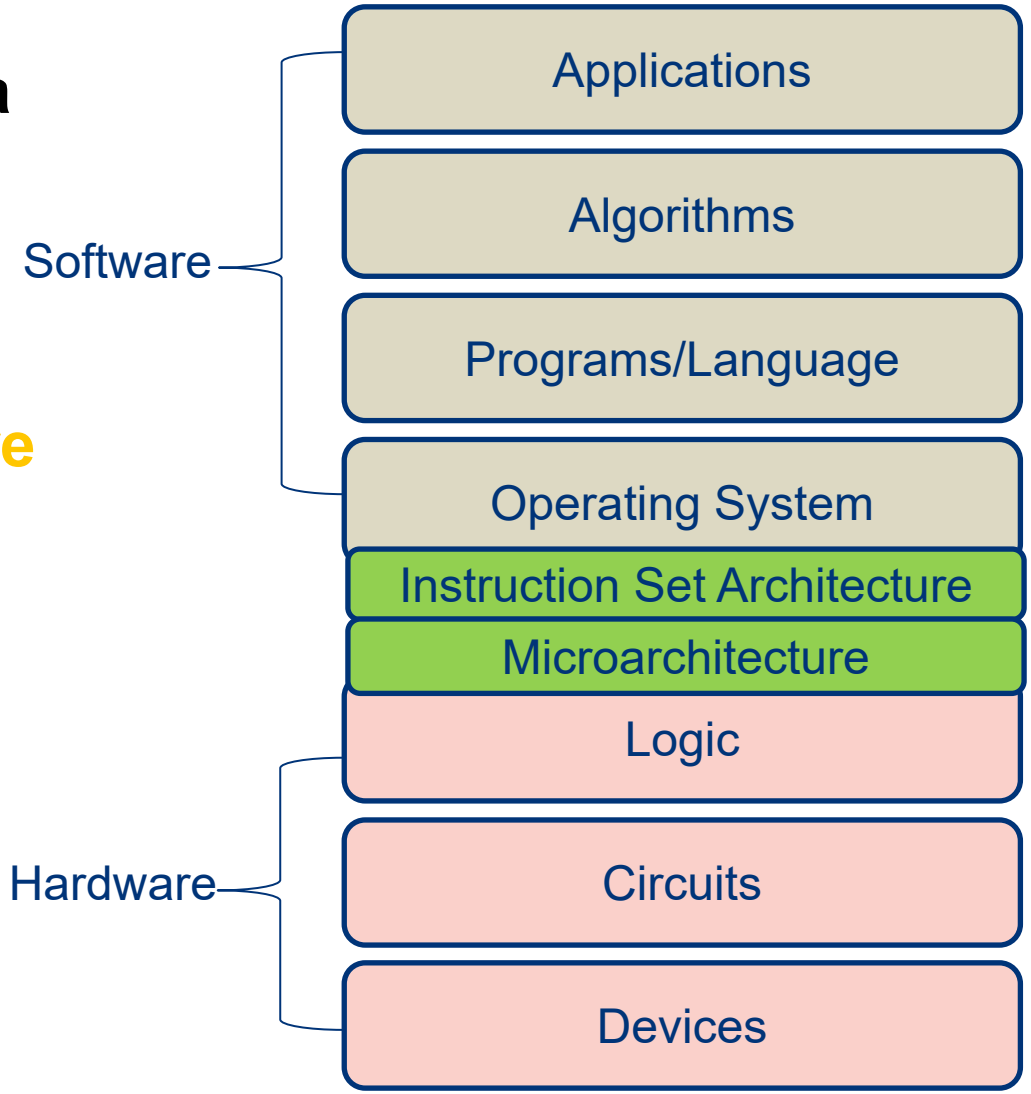
Computing Stacks

- **What are the stacks of a computing system?**



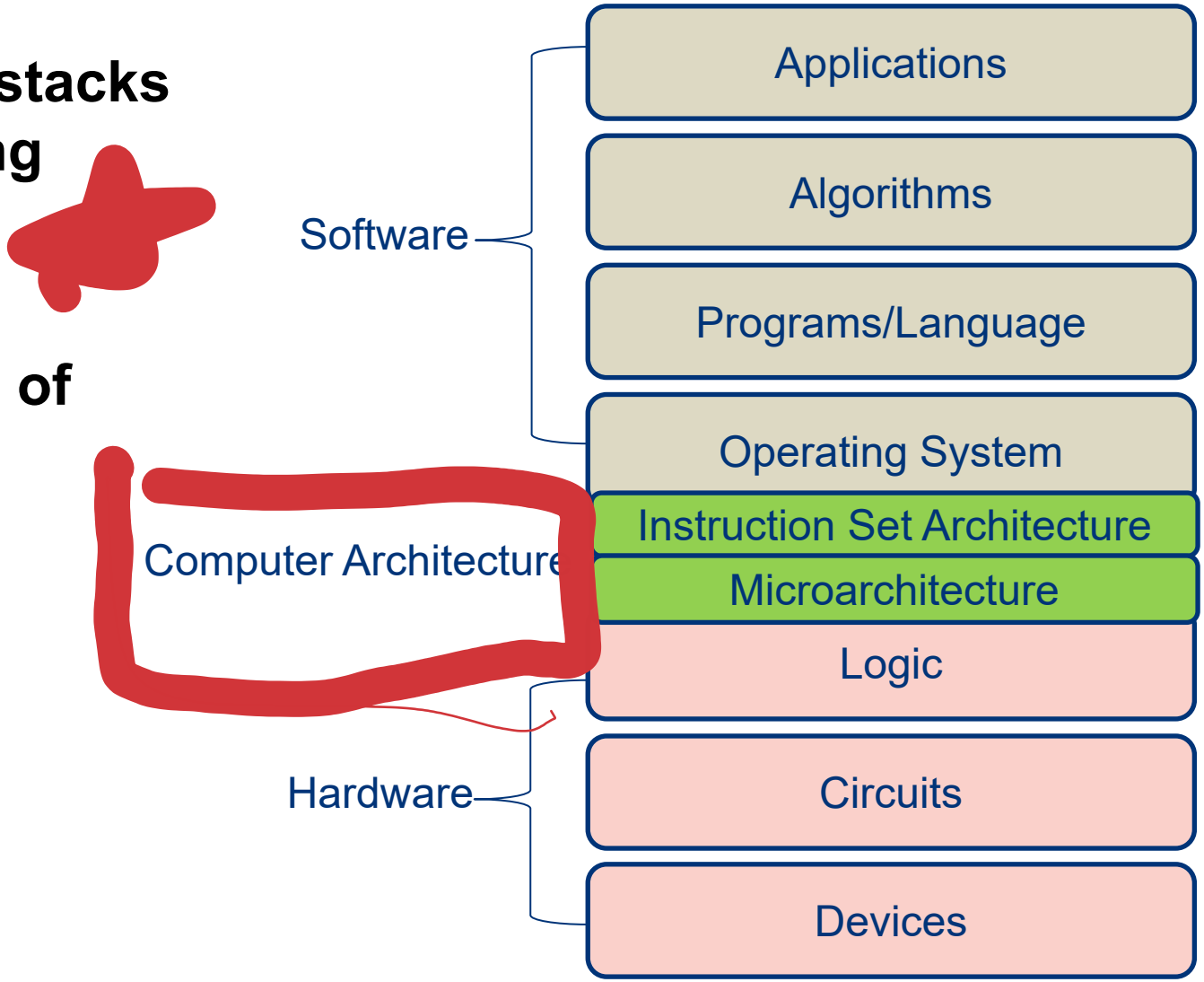
Computing Stacks

- **What are the stacks of a computing system?**
- **Instruction Set: Software Specifications of the computing system**
- **Microarchitecture: Hardware implementation of the computing system**



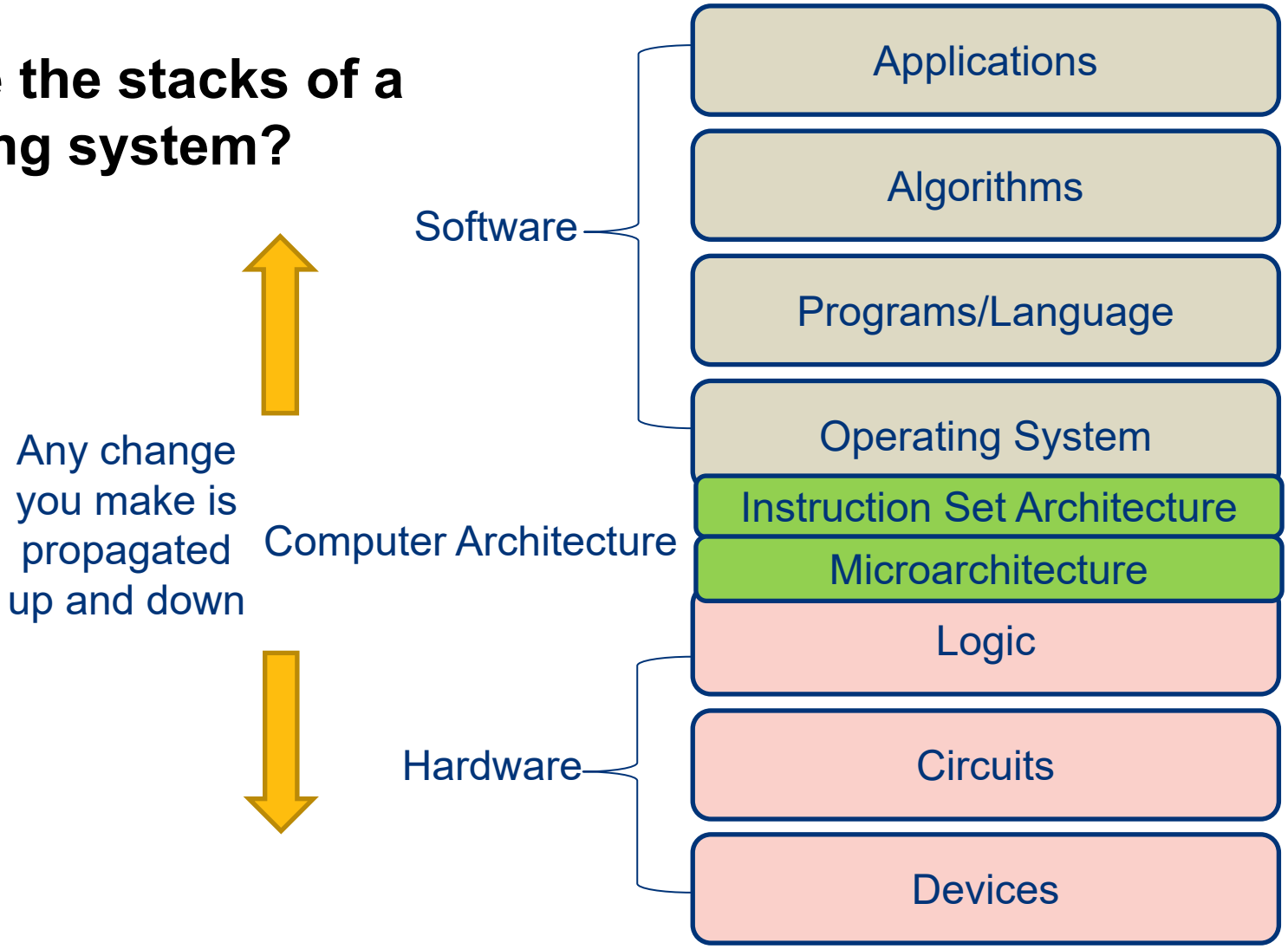
Computing Stacks

- **What are the stacks of a computing system?**
- **Old definition of computer architecture**



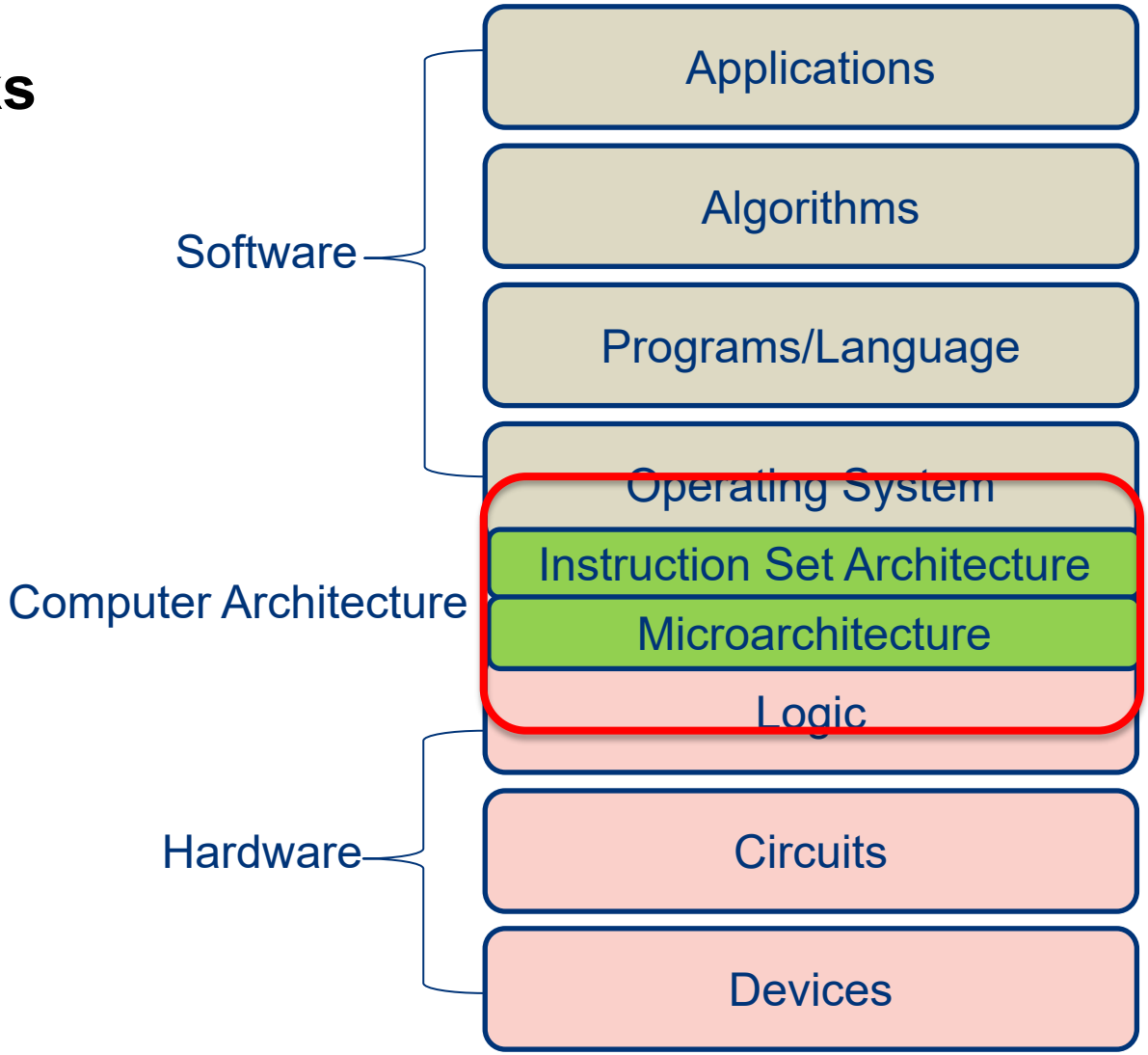
Computing Stacks

- **What are the stacks of a computing system?**



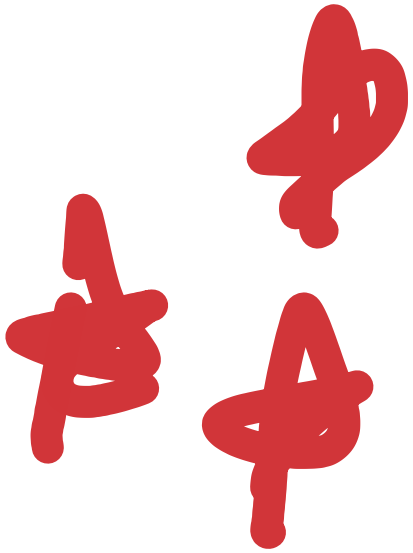
Computing Stacks

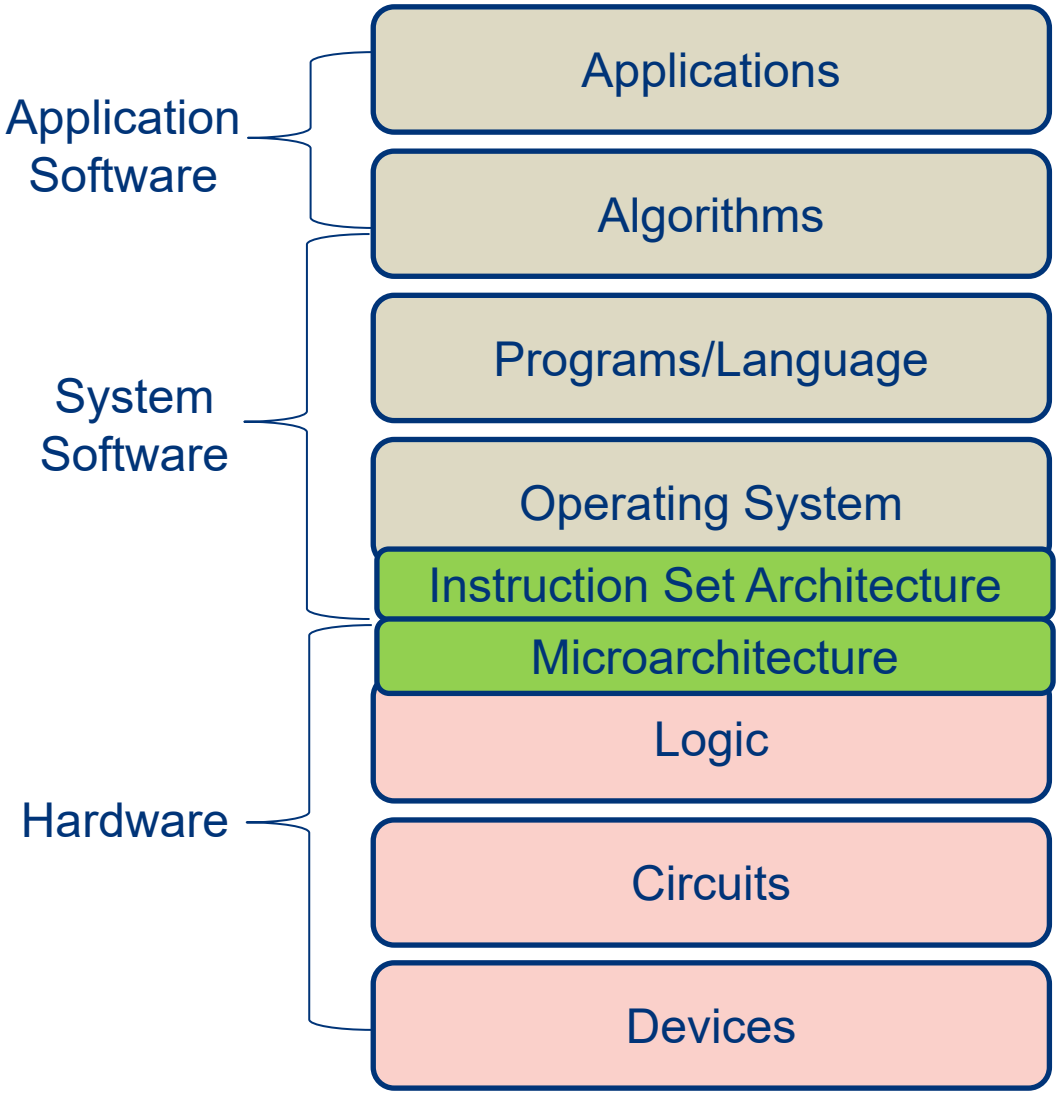
- **What are the stacks of a computing system?**
- **New definition of computer architecture**



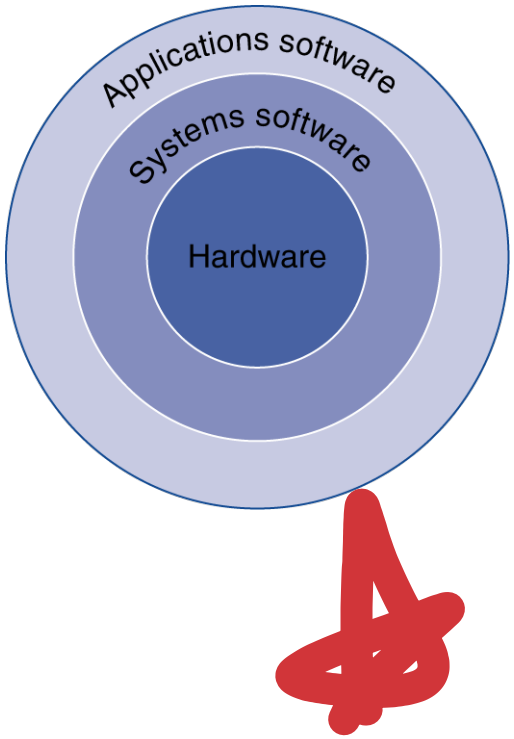
Computing Stacks

- **Translation to other definitions**





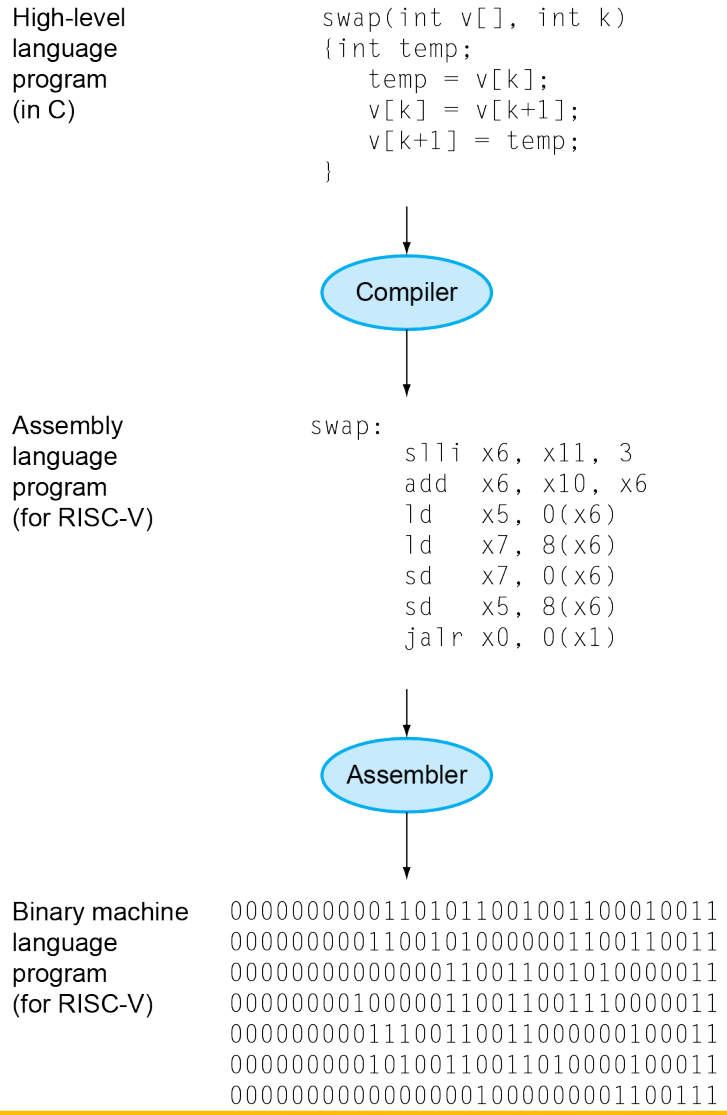
Below Your Program



- **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

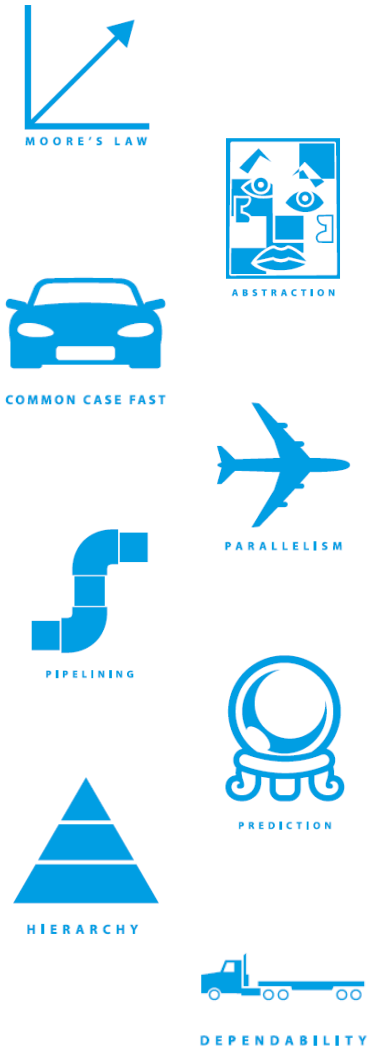
Levels of Program Code

- **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



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

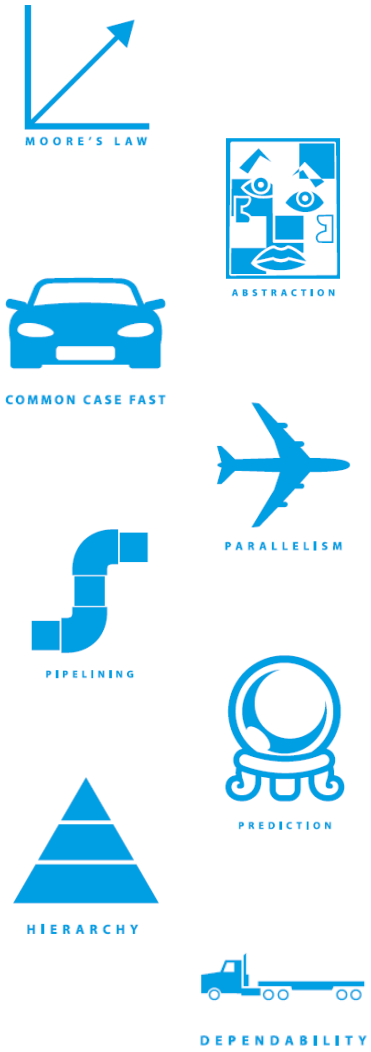


Understanding Performance

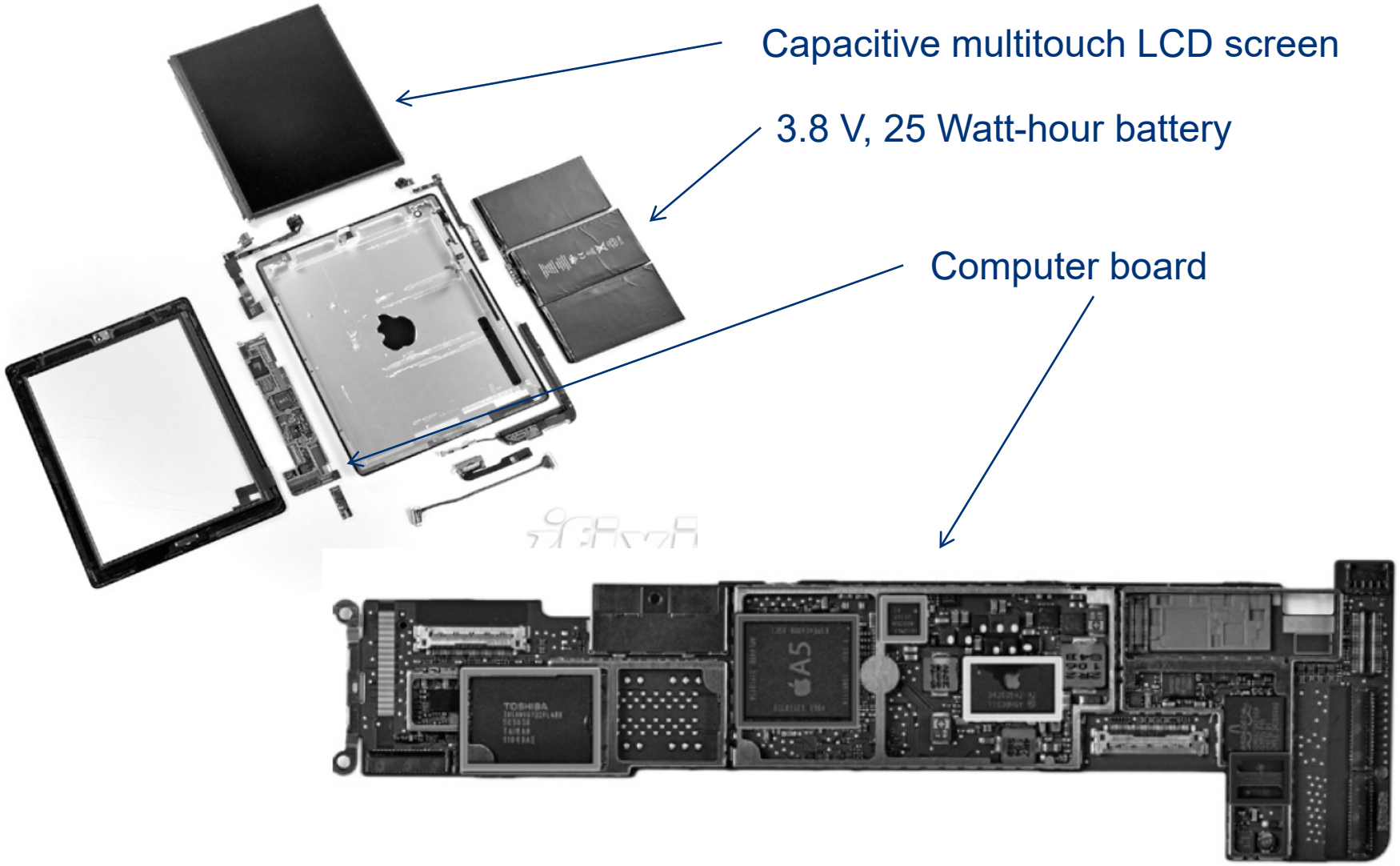
- **Algorithm**
  - Determines number of operations executed
- **Programming language, compiler, architecture**
  - Determine number of machine instructions executed per operation
- **Processor and memory system**
  - Determine how fast instructions are executed
- **I/O system (including OS)**
  - Determines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

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

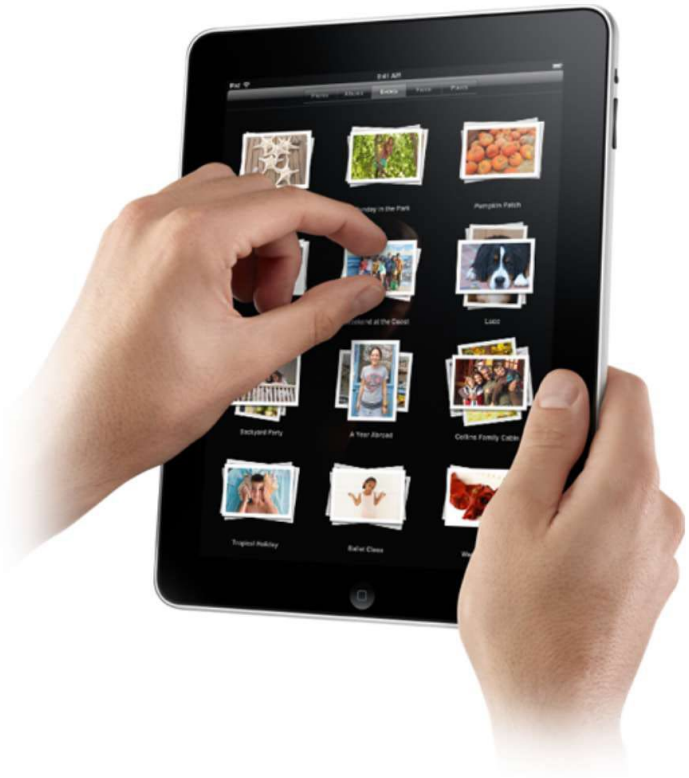


Components of a computer

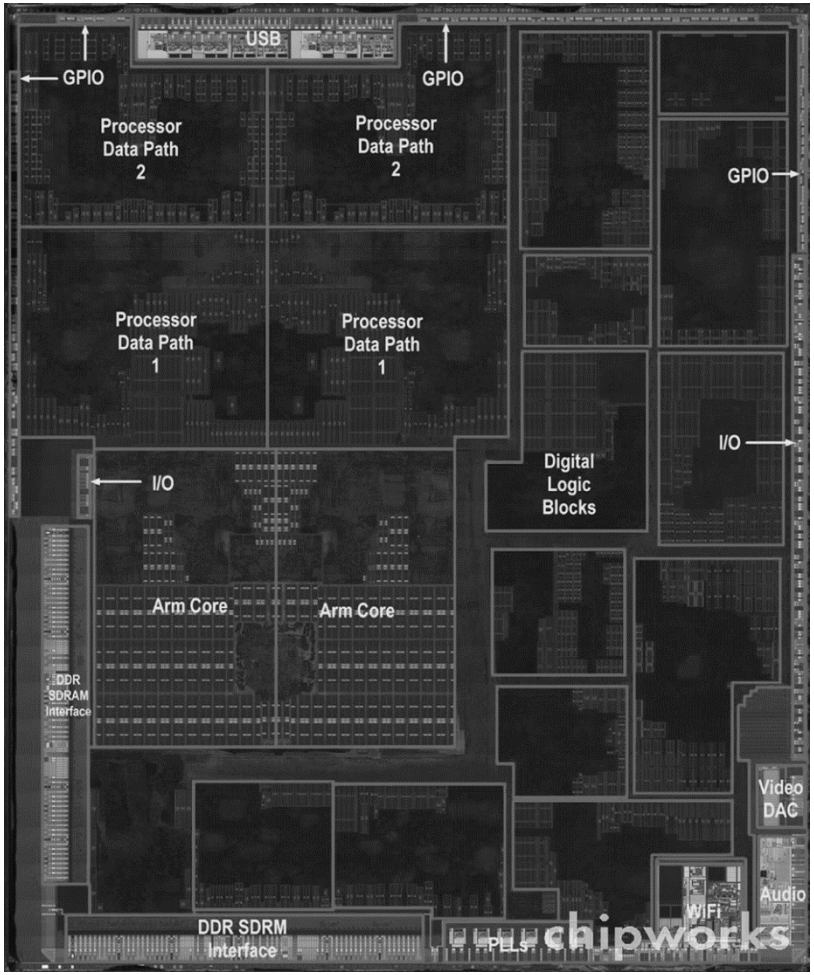
- **Same components for all kinds of computer**
  - Datapath: performs operations on data
  - Control: sequences datapath, memory, ...
  - Register: for performing operations
  - Cache memory
    - Small fast memory for immediate access to data
- **Input/output includes**
  - User-interface devices
    - Display, keyboard, mouse
  - Storage devices
    - Hard disk, CD/DVD, flash
  - Network adapters
    - For communicating with other computers

Touchscreen

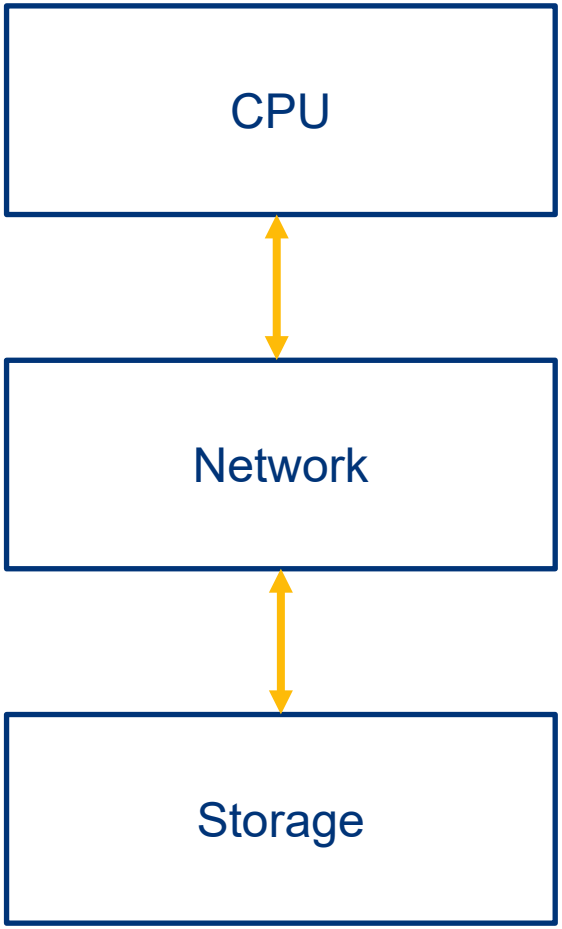
- **PostPC device**
- **Supersedes keyboard and mouse**
- **Resistive and Capacitive types**
  - Most tablets, smart phones use capacitive
  - Capacitive allows multiple touches simultaneously



Inside the Apple A5

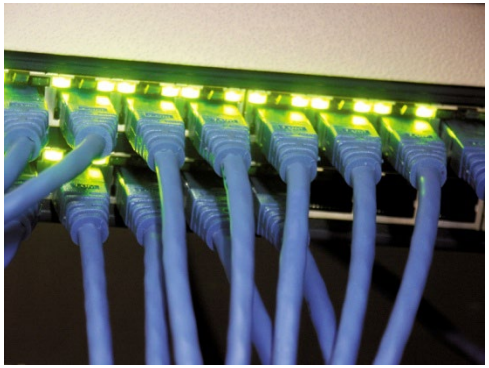


Computer: High level



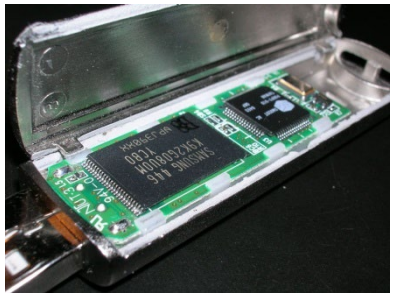
Networks

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

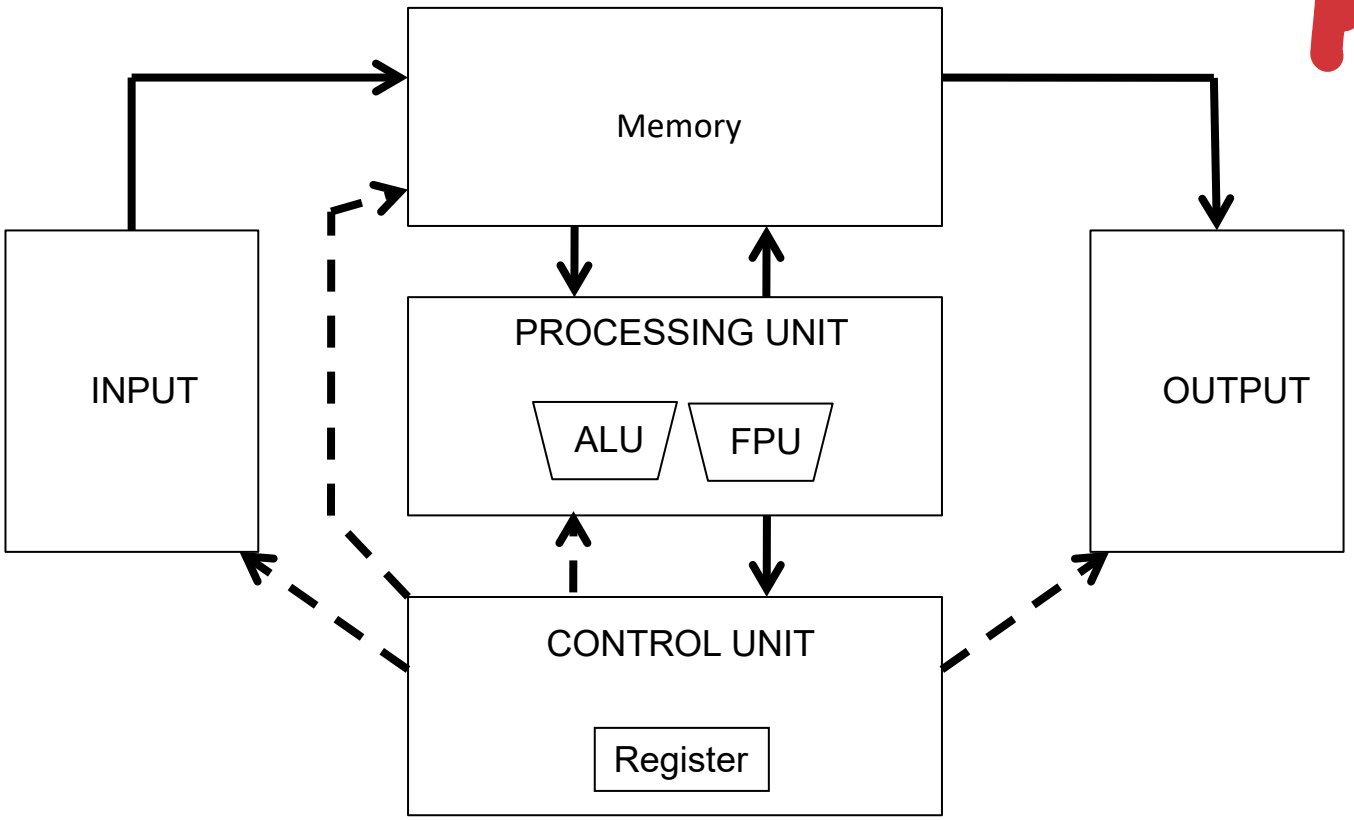


A Safe Place for Data

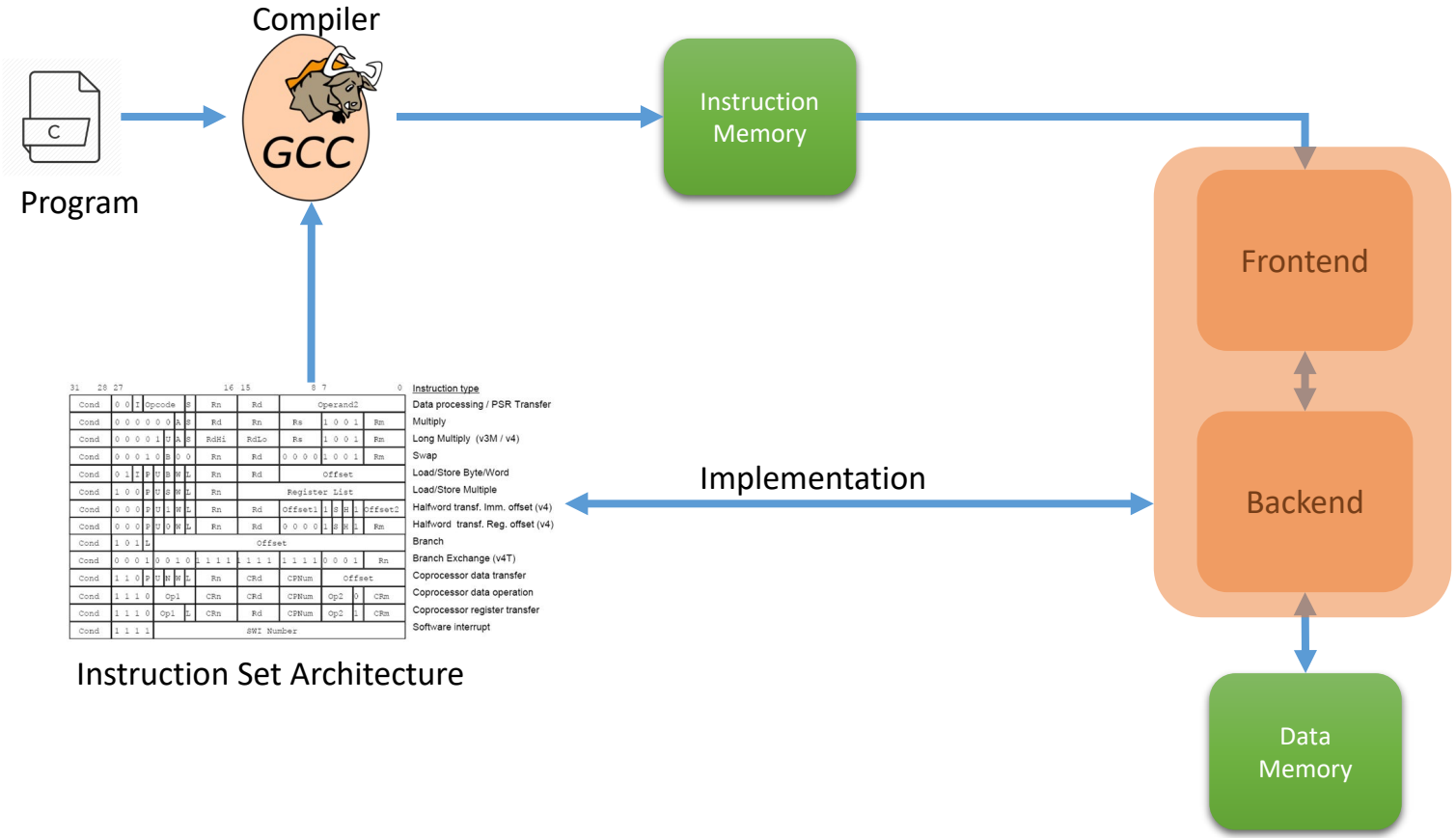
- **Volatile main memory**
  - Loses instructions and data when power off
- **Non-volatile secondary memory**
  - Magnetic disk
  - Flash memory
  - Optical disk (CDROM, DVD)



Control flow Architecture (Von-Neumann)



Context



Instruction Set Architecture

Content

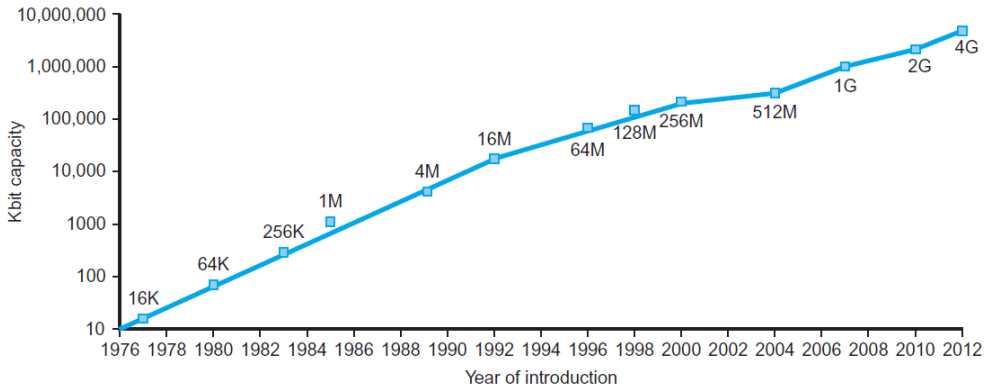
- More Insight into IC manufacturing
- Performance

Content

- More Insight into IC manufacturing
- Performance

Technology Trends

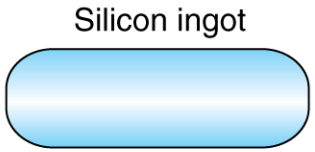
- Electronics technology continues to evolve
  - Increased capacity and performance
  - Reduced cost



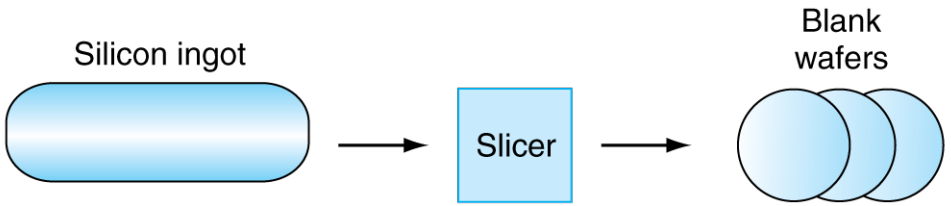
DRAM capacity

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

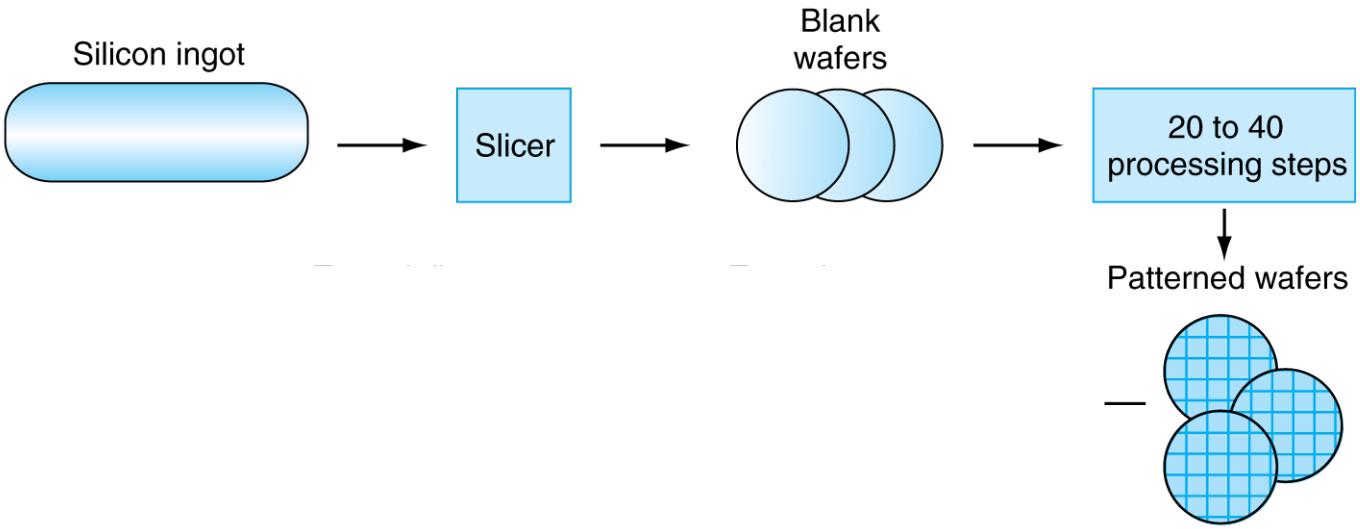
Manufacturing ICs



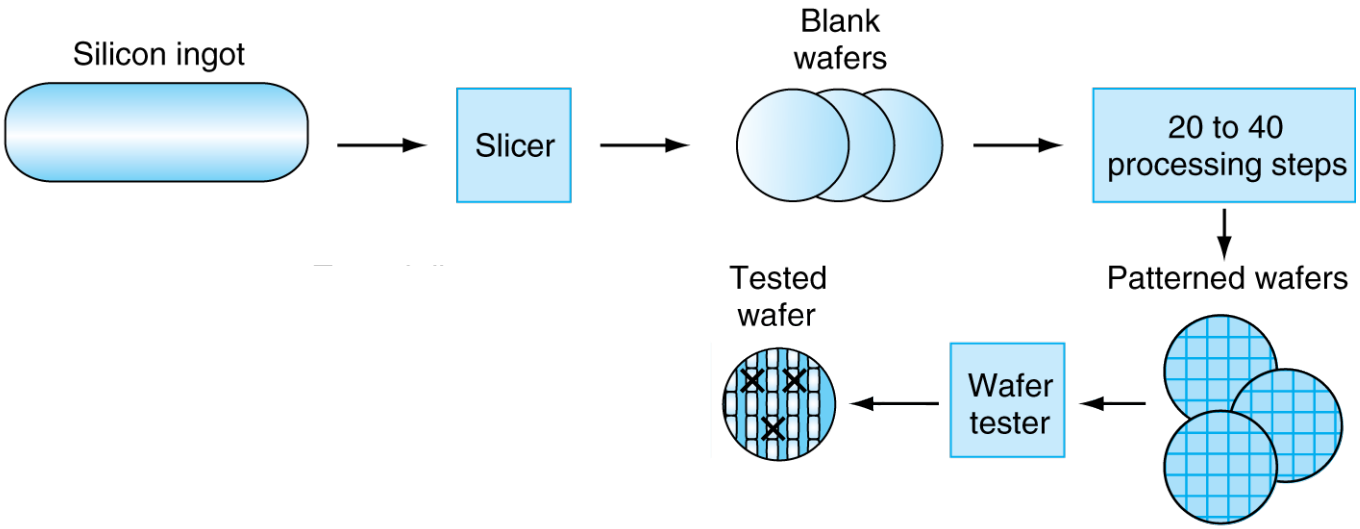
Manufacturing ICs



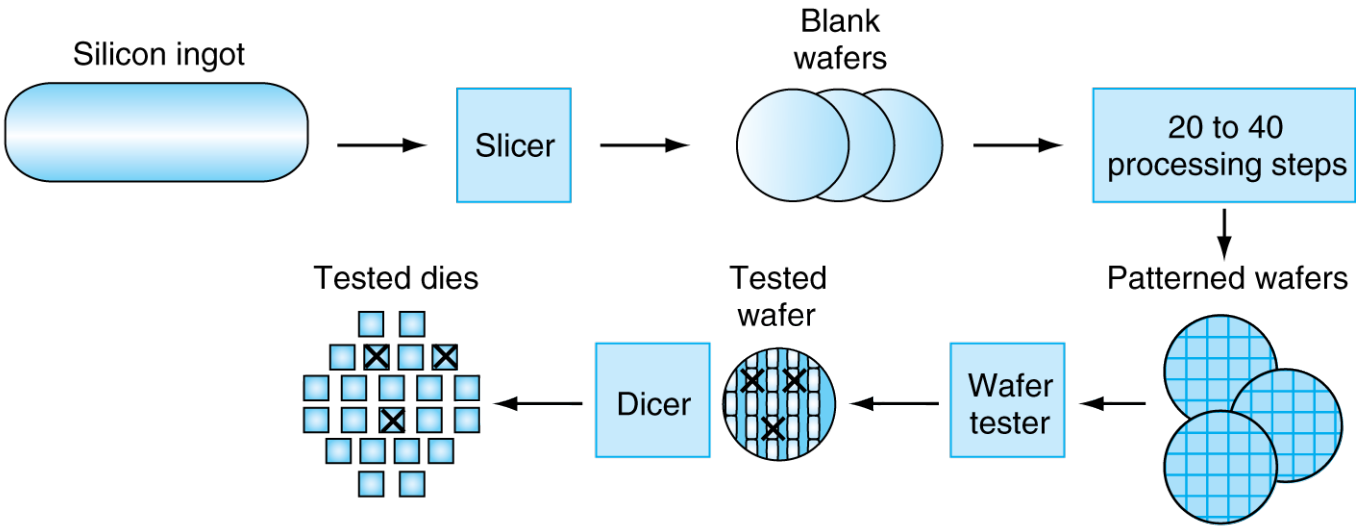
Manufacturing ICs



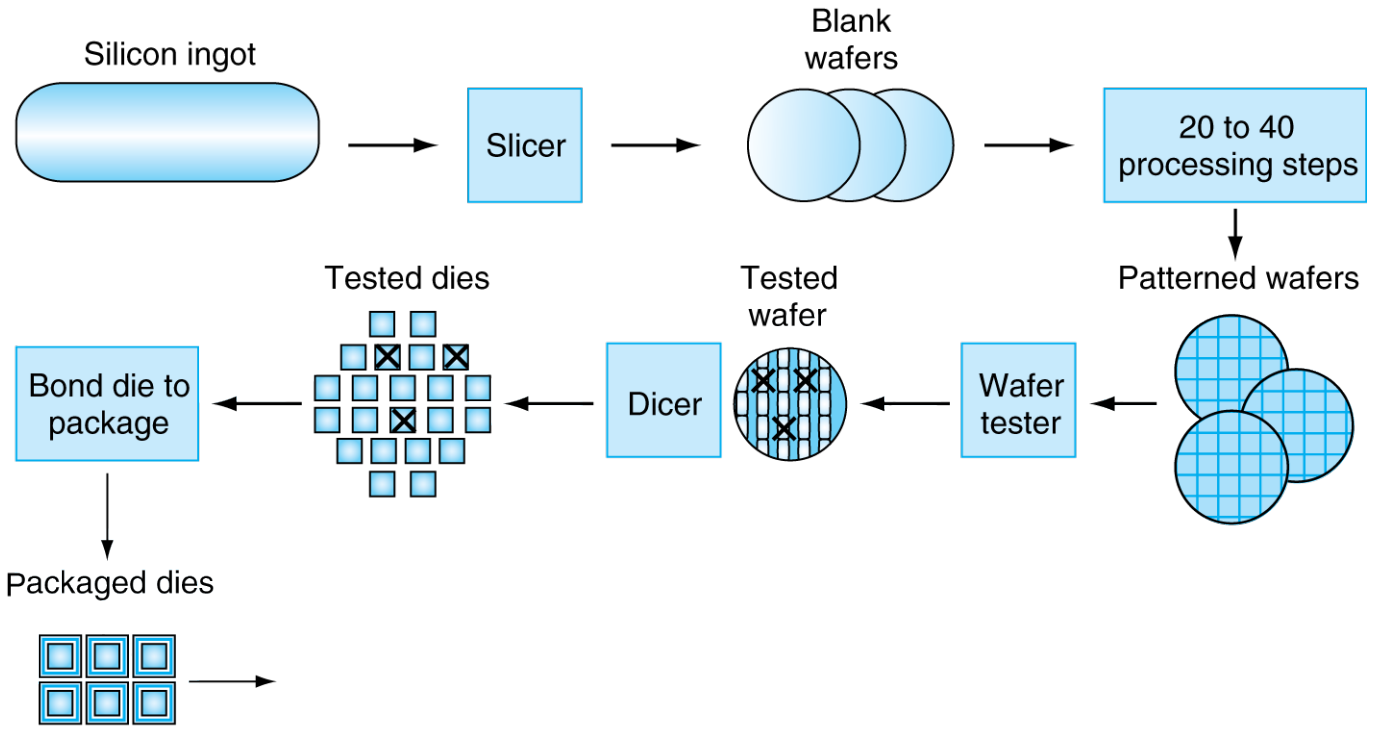
Manufacturing ICs



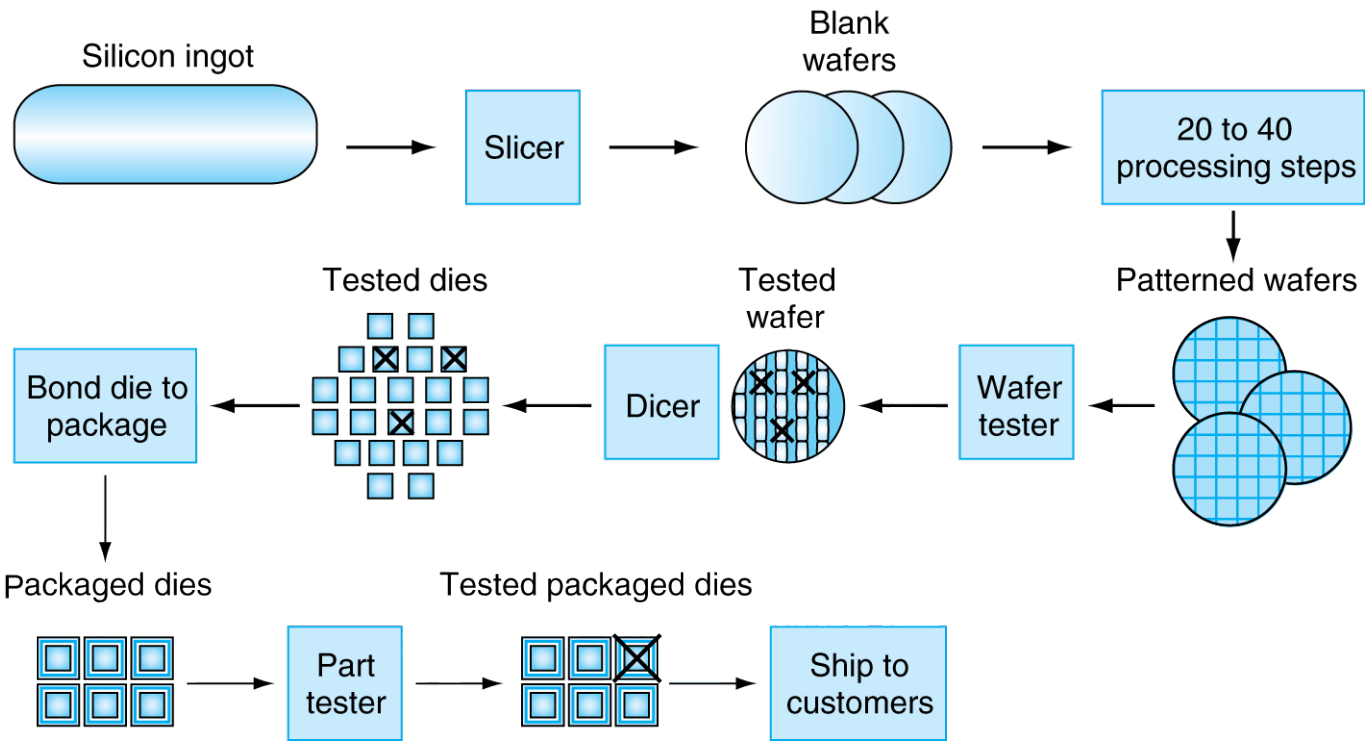
Manufacturing ICs



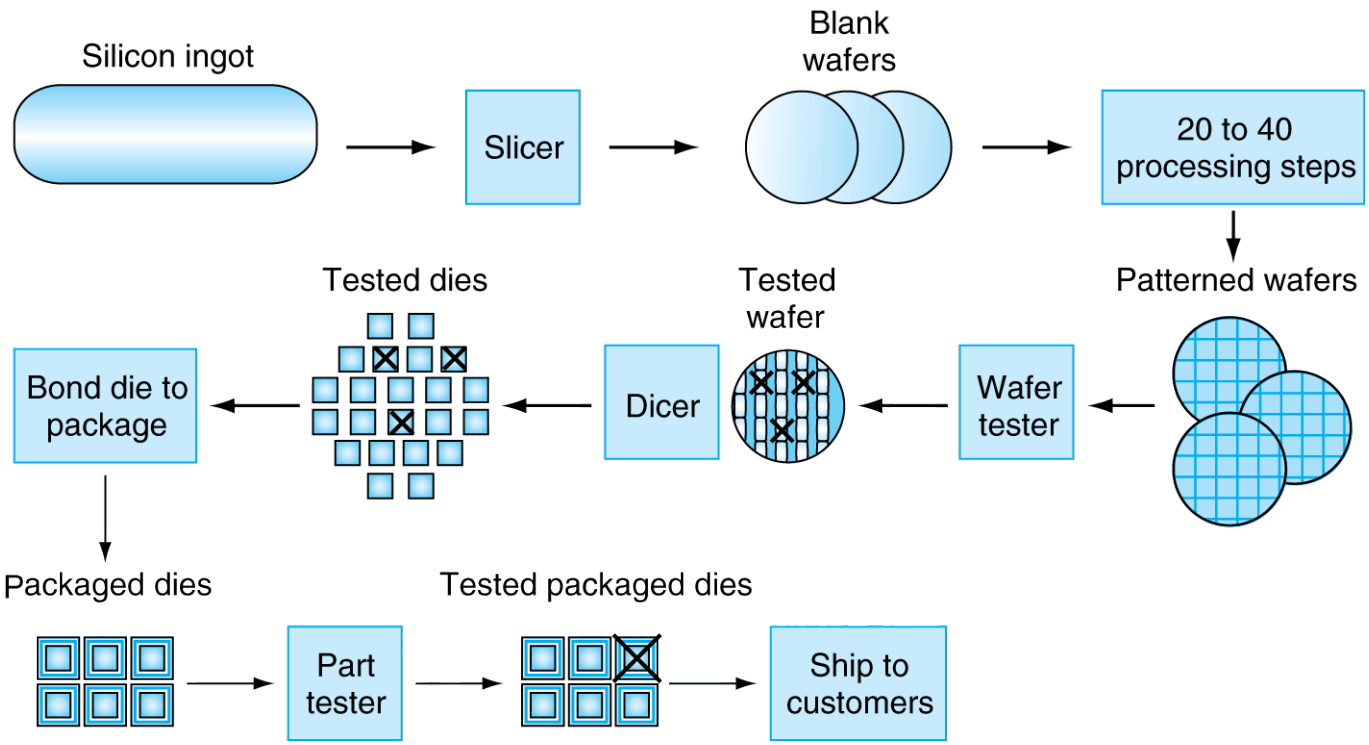
Manufacturing ICs



Manufacturing ICs



Manufacturing ICs

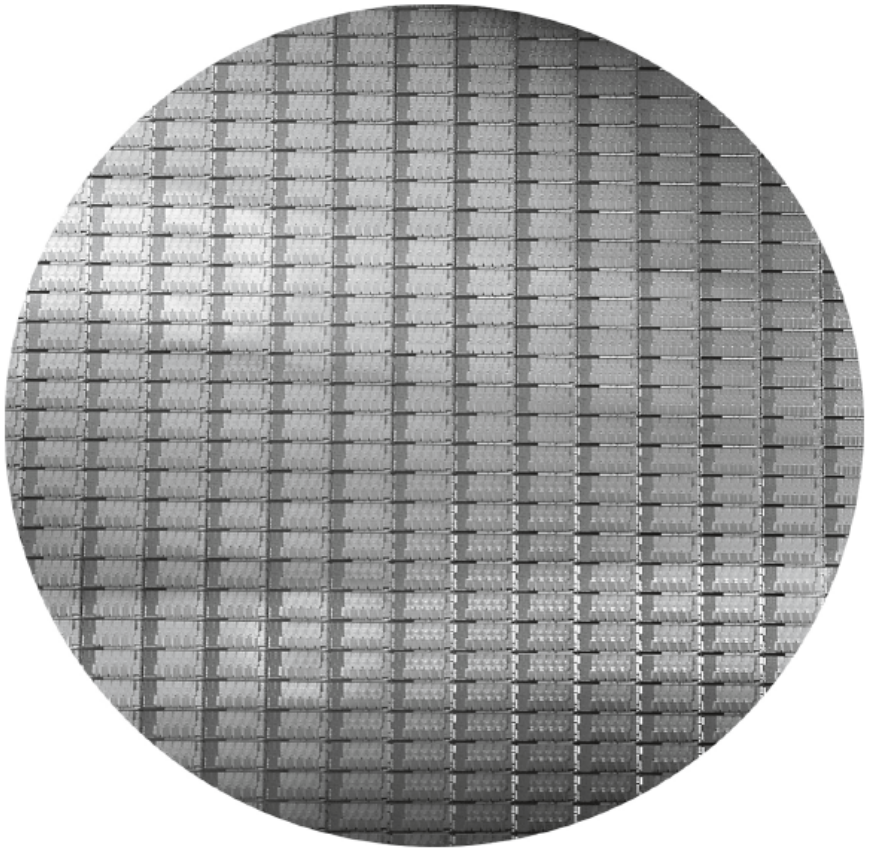


- **Yield: proportion of working dies per wafer**

Yield

- **Fabs are secretive about yields**
  - Can be as low as 30%
  - 70% of the dies have defects
  - Smaller the technology node, lower is the yield
- **Intel**
  - Has its own fab but also uses external fabs for lower technology
- **ARM**
  - Fabless, uses TSMC
- **NVIDIA**
  - Fabless, uses TSMC
- **TSMC ➔ IC fabrication is their sole business**

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

Content

- More Insight into IC manufacturing
- Performance

Defining 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

Defining Performance

- **We introduce two terms**
  - Execution time
    - How long it takes to do a task
  - Throughput
    - Total work done per unit time
      - e.g., tasks/transactions/… per hour

- **How are execution time and throughput affected by**
  - Replacing the processor with a faster version?
  - Adding more processors?

Defining Performance

- **Define Performance = 1/Execution Time**
- **"Processor X is n time faster than processor Y"**

$$\text{Performance}_X / \text{Performance}_Y$$
$$= \text{Execution time}_Y / \text{Execution time}_X = n$$

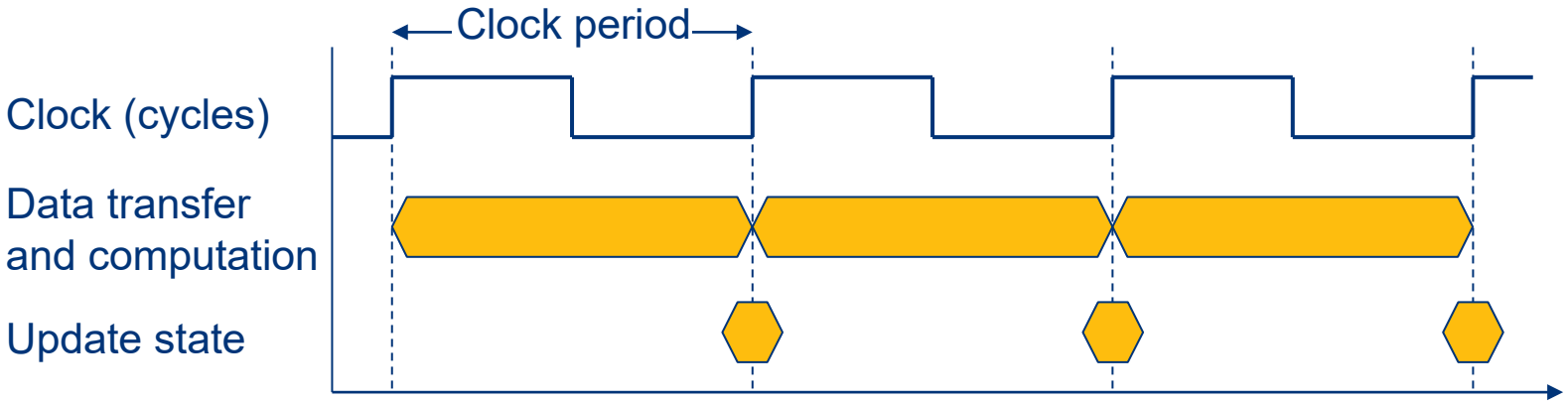
- **Example: time taken to run a program**
  - 10s on A, 15s on B
  - Execution Time_B / Execution Time_A = 15s / 10s = 1.5
  - So A is 1.5 times faster than B

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
  - Different programs are affected differently by CPU and system performance

CPU Clocking

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



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

CPU Time

$$\text{CPU Time} = \text{CPU Clock Cycles} \times \text{Clock Cycle Time}$$

$$= \frac{\text{CPU Clock Cycles}}{\text{Clock Rate}}$$

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

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 × clock cycles
- **How fast must Computer B clock be?**

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

$$\text{Clock Cycles}_A = \text{CPU Time}_A \times \text{Clock Rate}_A$$

$$= 10\text{s} \times 2\text{GHz} = 20 \times 10^9$$

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

CPU Time

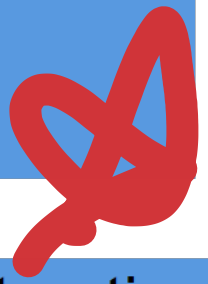
$$\text{CPU Time} = \text{CPU Clock Cycles} \times \text{Clock Cycle Time}$$

$$= \frac{\text{CPU Clock Cycles}}{\text{Clock Rate}}$$

Instruction Count and CPI

$$\text{CPU Time} = \text{CPU Clock Cycles} \times \text{Clock Cycle Time}$$

$$= \frac{\text{CPU Clock Cycles}}{\text{Clock Rate}}$$



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

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

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

Instruction Count and CPI

$$\text{CPU Time} = \text{CPU Clock Cycles} \times \text{Clock Cycle Time}$$

$$= \frac{\text{CPU Clock Cycles}}{\text{Clock Rate}}$$

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

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

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

Instruction Count and CPI

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

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

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

- **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

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?**

$$\text{CPU Time}_A = \text{Instruction Count} \times \text{CPI}_A \times \text{Cycle Time}_A$$
$$= I \times 2.0 \times 250\text{ps} = I \times 500\text{ps}$$

A is faster…

$$\text{CPU Time}_B = \text{Instruction Count} \times \text{CPI}_B \times \text{Cycle Time}_B$$
$$= I \times 1.2 \times 500\text{ps} = I \times 600\text{ps}$$

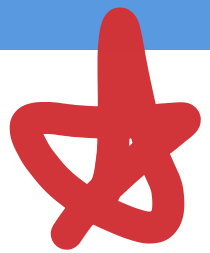
$$\frac{\text{CPU Time}_B}{\text{CPU Time}_A} = \frac{I \times 600\text{ps}}{I \times 500\text{ps}} = 1.2$$

…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**

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

CPI Example

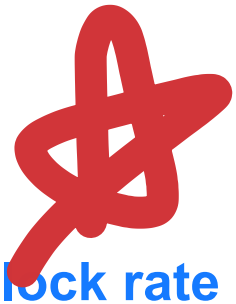
Class	A	B	C
CPI for class	1	2	3
instructions in sequence 1	2	1	2
instructions in sequence 2	4	1	1

- Sequence 1: IC = 5
  - Clock Cycles
    = 2×1 + 1×2 + 2×3
    = 10
  - Avg. CPI = 10/5 = 2.0
- Sequence 2: IC = 6
  - Clock Cycles
    = 4×1 + 1×2 + 1×3
    = 9
  - Avg. CPI = 9/6 = 1.5

Performance Summary

- CPU Time = time taken to execute a program

- A program has N instructions
- A program requires (N * CPI) CPU cycles to execute
- CPU Time = CPU cycles / clock rate



- CPU Time = (N * CPI) / clock rate
- CPU Time = ((instructions / program) * CPI) / clock rate
- CPU Time = ((instructions / program) * CPI) / (cycles/ second)
- CPU Time = ((instructions / program) * CPI) * (second/ cycle)

Performance Summary

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

CPI

- **Performance depends on**
  - Programmer: affects instruction count
  - Algorithm: affects instruction count, possibly CPI
  - Programming language: affects instruction count, CPI
  - Compiler: affects instruction count, CPI
  - Instruction set architecture: affects instruction count, CPI, clock cycle

Performance Analysis

- **CPU Time = (N * CPI) / clock rate**

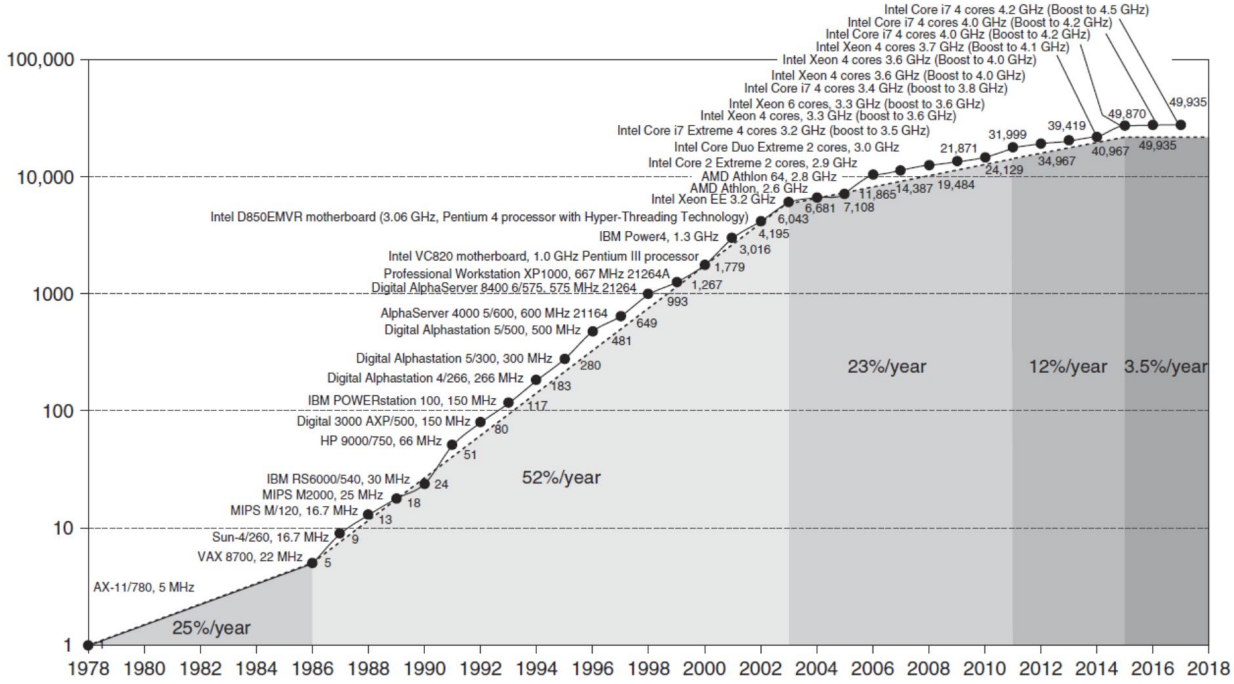
- **N is fixed ➔ programmer code**
- **CPI is fixed ➔ underlying ISA of the hardware**

- **High performance ➔ low CPU Time ➔ high clock rate**
- **Performance can be increased by increasing the clock rate (clock frequency)**
  - Performance (P) ∝ Frequency (F)

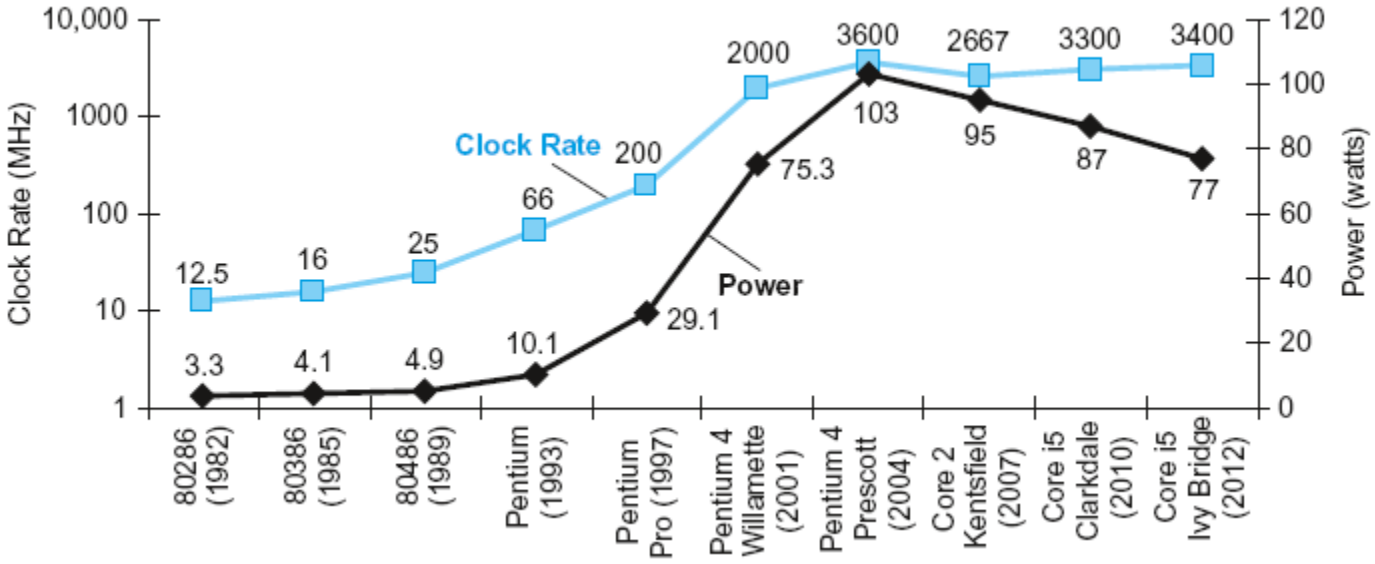
- **High performance computer is high frequency computer**

High frequency computer architecture

- Do you see some trends here?
- Observe the steady growth of frequency until 2003
- Saturation around 2012 onwards



Pitfall: Power Trends



- **In CMOS IC technology**

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

Power Consumption

- Constraints specific to processors: $\mathrm{F} \propto V$
  - Voltage and frequency scales together (dynamic voltage and frequency scaling, DVFS)
- Power: $\mathrm{P} \propto F^3$

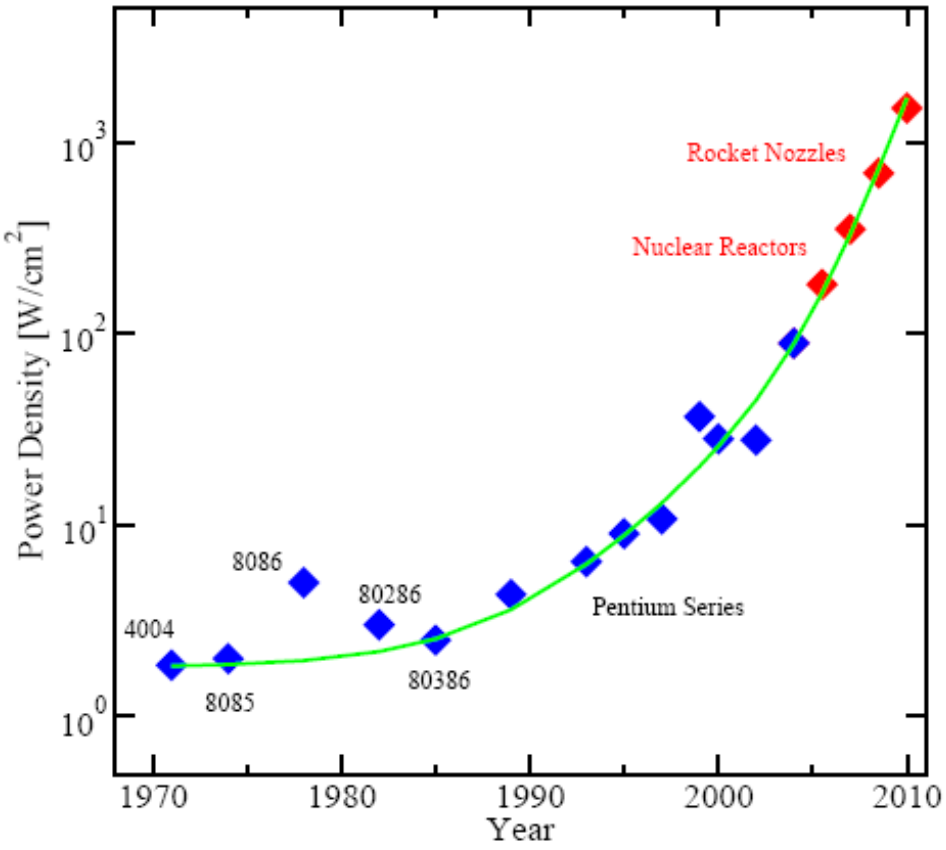
Pitfall: Power Trends

- **Increase of frequency F**
  - Increase of performance ($\propto F$)
  - Increase of power ($\propto F^3$)

- **What does increase of power implies?**
  - Energy consumption = Power * execution time
  - Increase of frequency = increase of energy consumption
    ➔ A battery (energy storage) will drain faster
  - Increase of frequency = reduced battery-life

Pitfall: Power Trends

- **Increase of frequency F**
  - Increase of performance
  - Increase of power
- **What does increase of power implies?**
  - Temperature = f(Power)
  - Increase of frequency = increase of temperature
    - ➔ potentially burn your chip
  - Increase of frequency = reduced reliability



Reducing Power

- **Suppose a new CPU has**
  - 85% of capacitive load of old CPU
  - 15% voltage and 15% frequency reduction (i.e., 15% performance reduction)

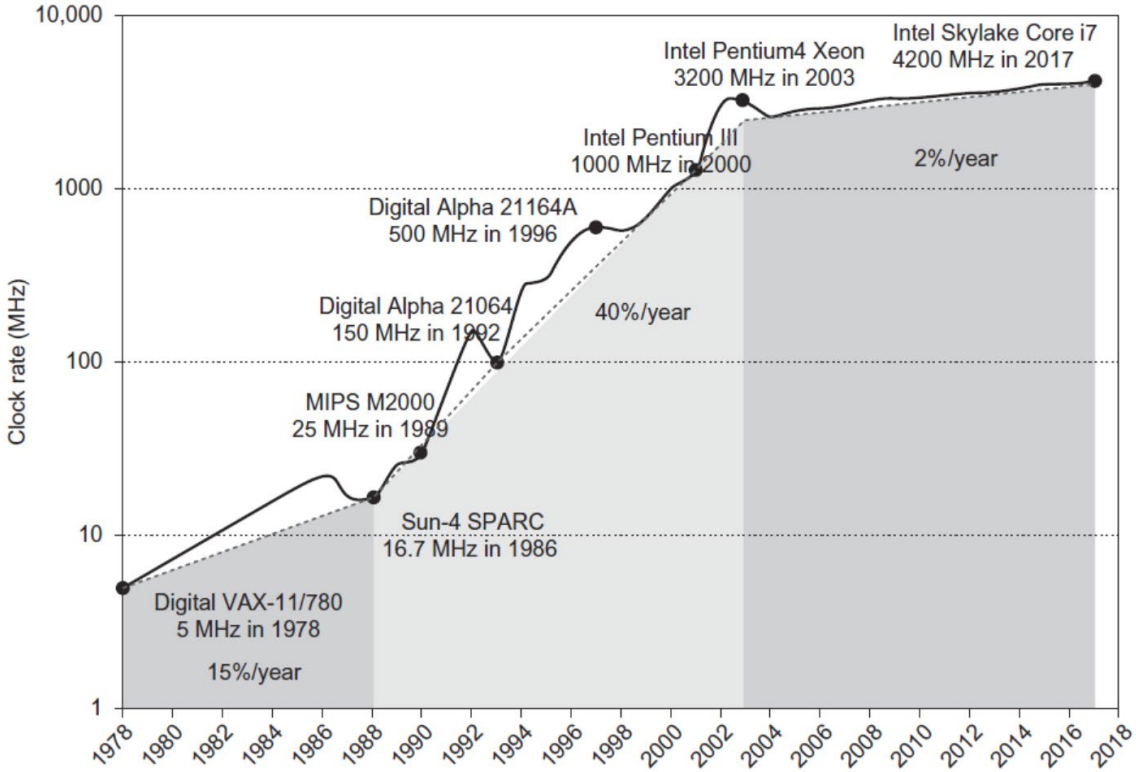
$$\frac{P_{new}}{P_{old}} = \frac{C_{old} \times 0.85 \times (V_{old} \times 0.85)^2 \times F_{old} \times 0.85}{C_{old} \times V_{old}^{\ 2} \times F_{old}} = 0.85^4 = 0.52$$

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

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

Multi-core Evolution



Pitfall: MIPS as a Performance Metric

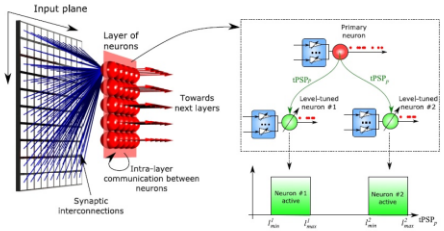
- **MIPS: Millions of Instructions Per Second**
  - Doesn't account for
    - Differences in ISAs between computers
    - Differences in complexity between instructions

$$\text{MIPS} = \frac{\text{Instruction count}}{\text{Execution time} \times 10^6}$$

$$= \frac{\text{Instruction count}}{\frac{\text{Instruction count} \times \text{CPI}}{\text{Clock rate}} \times 10^6} = \frac{\text{Clock rate}}{\text{CPI} \times 10^6}$$

- **CPI varies between programs on a given CPU**

The computing landscape

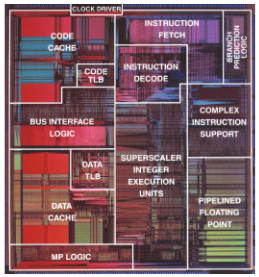


Neuromorphic Computing

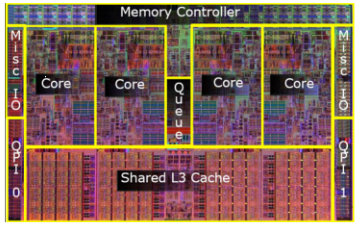
Quantum Computing
Probabilistic Computing
In-Memory Computing

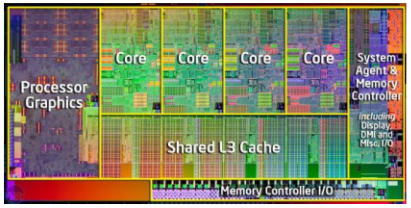
Approximate Computing

Single Core



Multi Core





Source: Intel

1971 2004 2015

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