NATIONAL RESEARCH UNIVERSITY

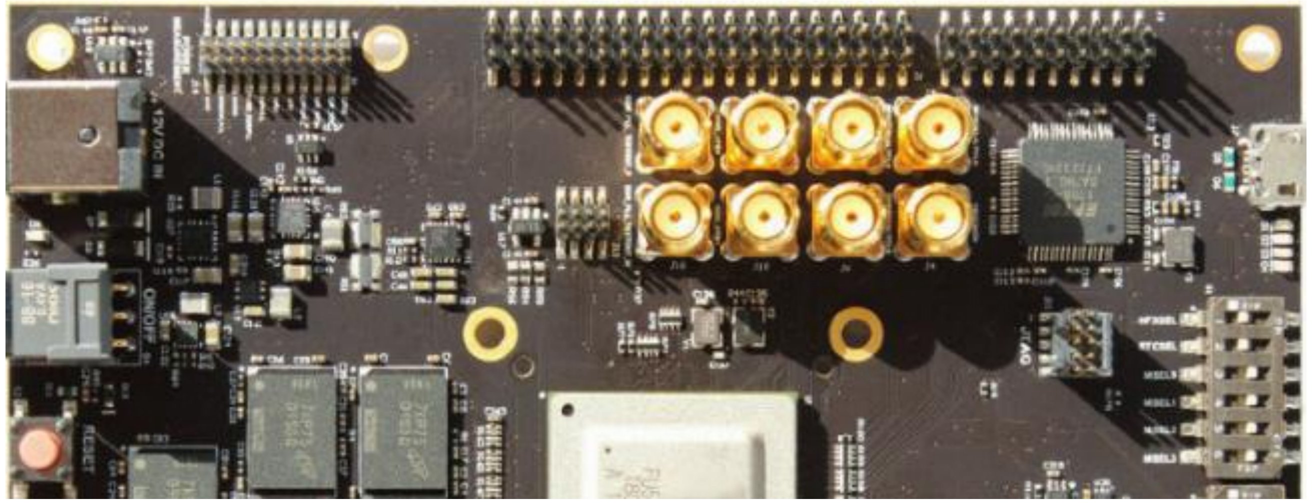
Faculty of Computer science
Higher School of Economics

# Computer Architecture and Operating Systems
# Lecture 1: Introduction

**Andrei Tatarnikov**

atatarnikov@hse.ru
@andrewt0301

# Course Resources



- **Wiki**

http://wiki.cs.hse.ru/ACOS_DSBA_2020/2021

- **Web site**

https://andrewt0301.github.io/hse-acos-course/

- **Telegram channel**

https://t.me/joinchat/AAAAAFDXhCd-WvYYZwBPGQ

## Course Team

### Instructors

**Andrei Tatarnikov**

### Assistants

TODO

## Course Outline

### Syllabus (see the web site for details)

- Module 3: Computer Architecture
  - Computer architecture
  - Assembly language programming (RISC-V)
  - Home works, quizzes, and test
- Module 4: Operating Systems
  - Operating System Architecture (Linux)
  - System programming in C
  - Home works, quizzes, and test
- Final Exam

# Course Motivation

- Increase your computer liretacy
- Have an idea how computers under the hood
- Better understand performance
- Be familiar with system programming
- Be familiar with system tools

## Example: Matrix Multiplication (part 1)

### Python

**Floating-point operations:**

$2 * n^3 = 2 * (2^{10})^3 = 2^{31}$

**Running time:**

503.130450 sec.

**Performance:**

~ 4,27 MFLOPS

```
import random
from time import time

n = 1024

A = [[random.random()
      for row in range(n)]
      for col in range(n)]
B = [[random.random()
      for row in range(n)]
      for col in range(n)]
C = [[0
      for row in range(n)]
      for col in range(n)]

start = time()
for i in range(n):
    for j in range(n):
        for k in range(n):
            C[i][j] += A[i][k] * B[k][j]
end = time()

print('%0.6f' % (end - start))
```

# Example: Matrix Multiplication (part 2)

## Java

**Floating-point operations:**

$2 * n^3 = 2 * (2^{10})^3 = 2^{31}$

**Running time:**

12.946224 sec.

**Performance:**

~ 165 MFLOPS

```
public class Matrix {
    static int n = 1024;
    static double[][] A = new double[n][n];
    static double[][] B = new double[n][n];
    static double[][] C = new double[n][n];

    public static void main(String[] args) {
        java.util.Random r = new java.util.Random();
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                A[i][j] = r.nextDouble();
                B[i][j] = r.nextDouble();
                C[i][j] = 0;
            }
        }
        long start = System.nanoTime();
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                for (int k = 0; k < n; k++) {
                    C[i][j] += A[i][k] * B[k][j];
                }
            }
        }
        long stop = System.nanoTime();
        System.out.println((stop - start) * 1e-9);
    }
}
```

## Example: Matrix Multiplication (part 3)

### C Language

**Floating-point operations:**

$2 * n^3 = 2 * (2^{10})^3 = 2^{31}$

**Running time:**

13.714264 sec.

**Performance:**

~ 153 MFLOPS

```c
#include <stdlib.h>
#include <stdio.h>
#include <sys/time.h>

#define n 1024
double A[n][n];
double B[n][n];
double C[n][n];

float tdiff(struct timeval *start, struct timeval *end) {
    return (end->tv_sec - start->tv_sec) + 1e-6*(end->tv_usec - start->tv_usec);
}
int main(int argc, const char *argv[]) {
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            A[i][j] = (double)rand() / (double)RAND_MAX;
            B[i][j] = (double)rand() / (double)RAND_MAX;
            C[i][j] = 0;
        }
    }
    struct timeval start, end;
    gettimeofday(&start, NULL);
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            for (int k = 0; k < n; k++) {
                C[i][j] += A[i][k] * B[k][j];
            }
        }
    }
    gettimeofday(&end, NULL);
    printf("%0.6f\n", tdiff(&start, &end));
    return 0;
}
```

# Example: Matrix Multiplication (part 4)

## C Language: Optimizations

### Loop order: i, j, k

```
for (int i= 0; i < n; i++) {
  for (int j= 0; j < n; j++) {
    for (int k= 0; k < n; k++) {
      C[i][j]+= A[i][k]*B[k][j];
    }
  }
}
```

**Running time:**
13.714264 sec.
**Performance:**
~ 153 MFLOPS

### Loop order: i, k, j

```
for (int i= 0; i < n; i++) {
  for (int k= 0; k < n; k++) {
    for (int j= 0; j < n; j++) {
      C[i][j]+= A[i][k]*B[k][j];
    }
  }
}
```

**Running time:**
2.739385 sec.
**Performance:**
~ 795 MFLOPS

### Loop order: k, j, i

```
for (int i= 0; i < n; i++) {
  for (int j= 0; j < n; j++) {
    for (int k= 0; k < n; k++) {
      C[i][j]+= A[i][k]*B[k][j];
    }
  }
}
```

**Running time:**
19.074106 sec.
**Performance:**
~ 113 MFLOPS

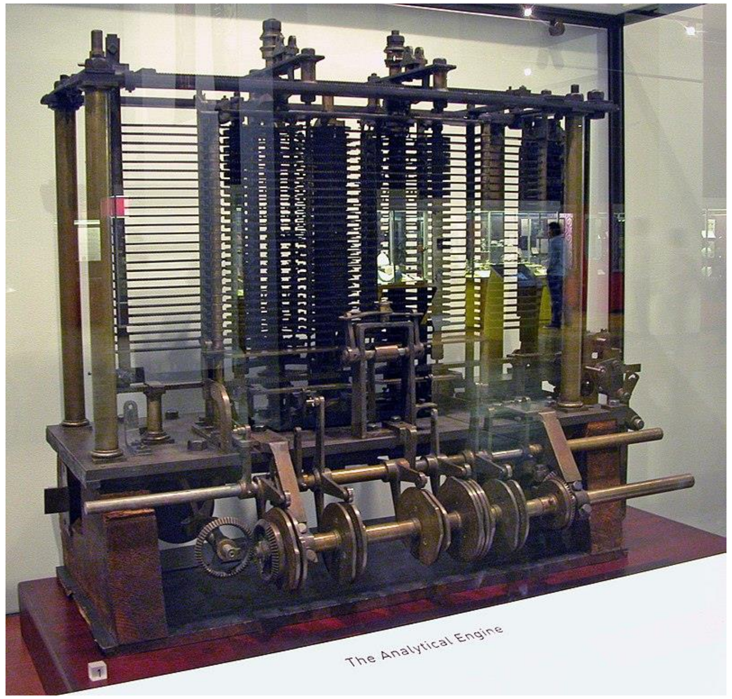
# Example: Matrix Multiplication (part 5)

| Feature | Specifiction |
|---|---|
| Model | MacBook Pro 9,1 |
| Processor Name | Quad-Core Intel Core i7 |
| Processor Speed | 2,3 GHz |
| Number of Processors | 1 |
| Total Number of Cores | 4 |
| Floating-Point Operations per Cycle | 4 |
| L2 Cache (per Core) | 256 KB |
| L3 Cache: | 6 MB |
| Hyper-Threading Technology | Enabled |
| Memory | 8 GB |

**Peak = $(2.3 * 10^9)$ * 1 * 4 * 4 = 36 800 MFLOPS**

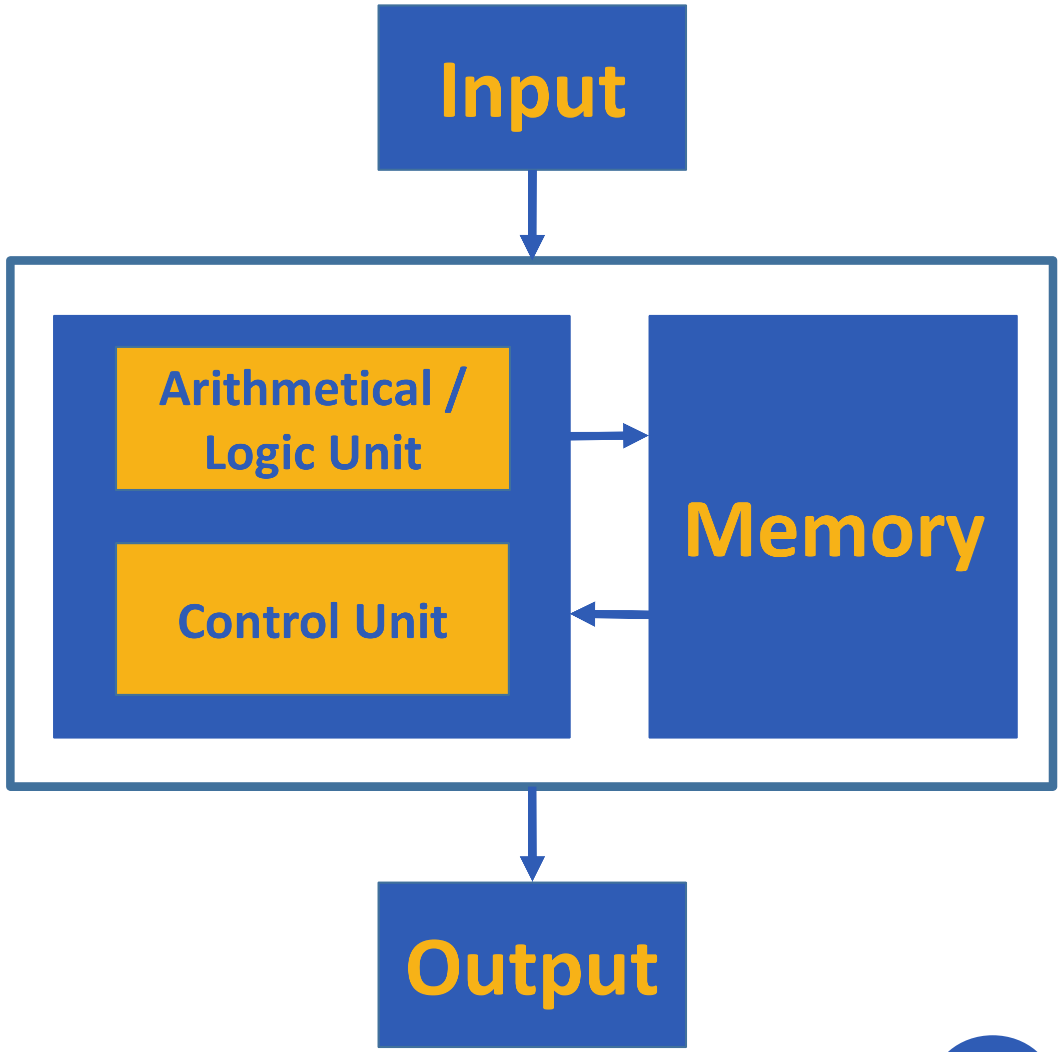
## History: 0th Generation - Analytical Engine

- Designed by Charles Babbage from 1834 – 1871
- Built from mechanical gears, where each gear represented a discrete value (0-9)
- Programs provided as punched cards
- Never finished due to technological restrictions



## History: 1st Generation - Vacuum Tubes

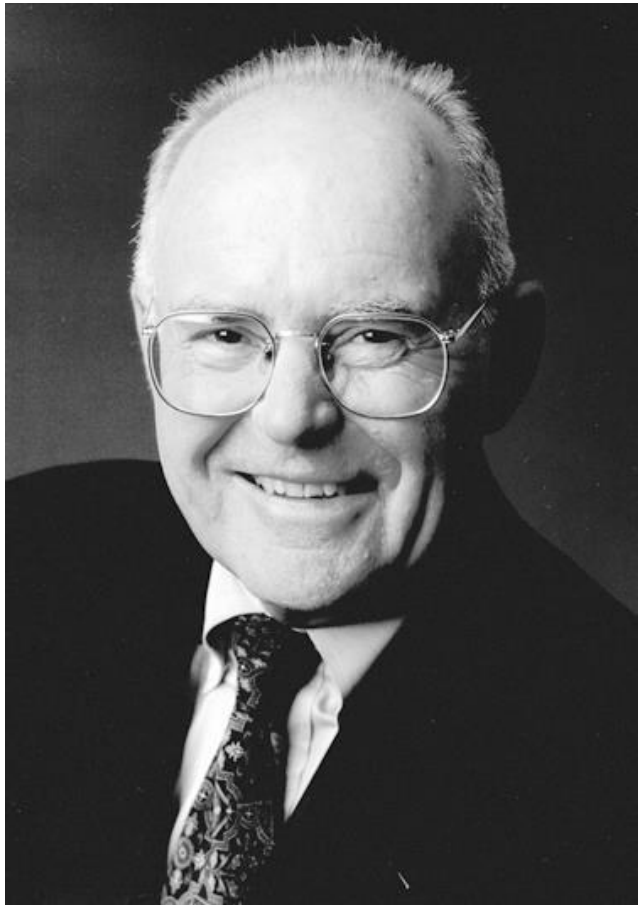
- In 1945–55, first machines were created: Atanasoff–Berry computer, Z3, Colossus, ENIAC
- All programming was done in machine language by connecting boards and wires
- Stored program concept was formulated



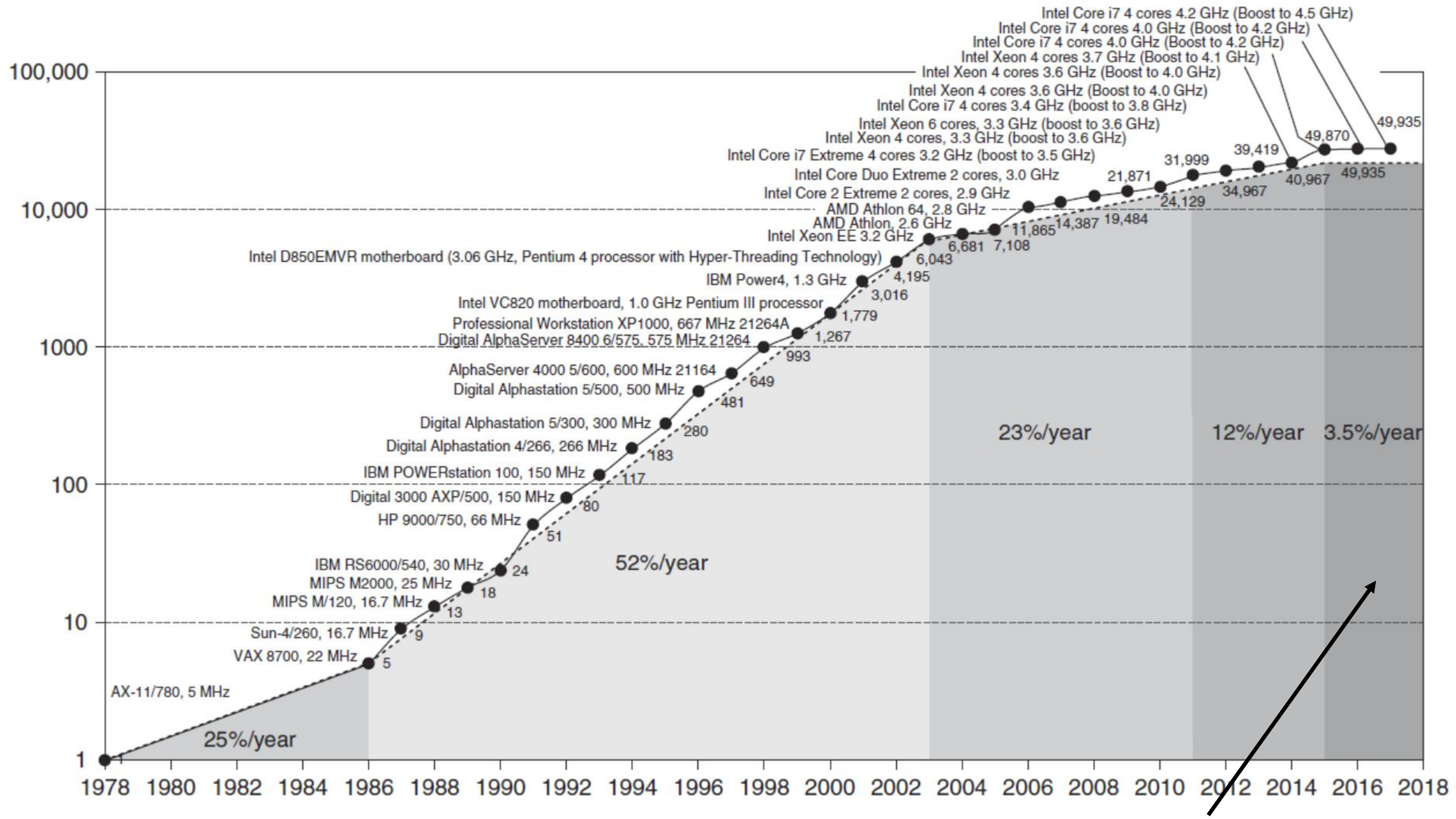
- 1955–65: 2nd Generation. Transistors and Batch Systems.
- 1965–1980: 3rd Generation. ICs and Multiprogramming.
- 1980–Present: 4th Generation. Personal Computers.
- 1990–Present: 5th Generation. Mobile Computers.

## Gordon Moore

- Cofounded Intel in 1968 with Robert Noyce.
- **Moore's Law:** number of transistors on a computer chip doubles every year (observed in 1965)
- Since 1975, transistor counts have doubled every two years.



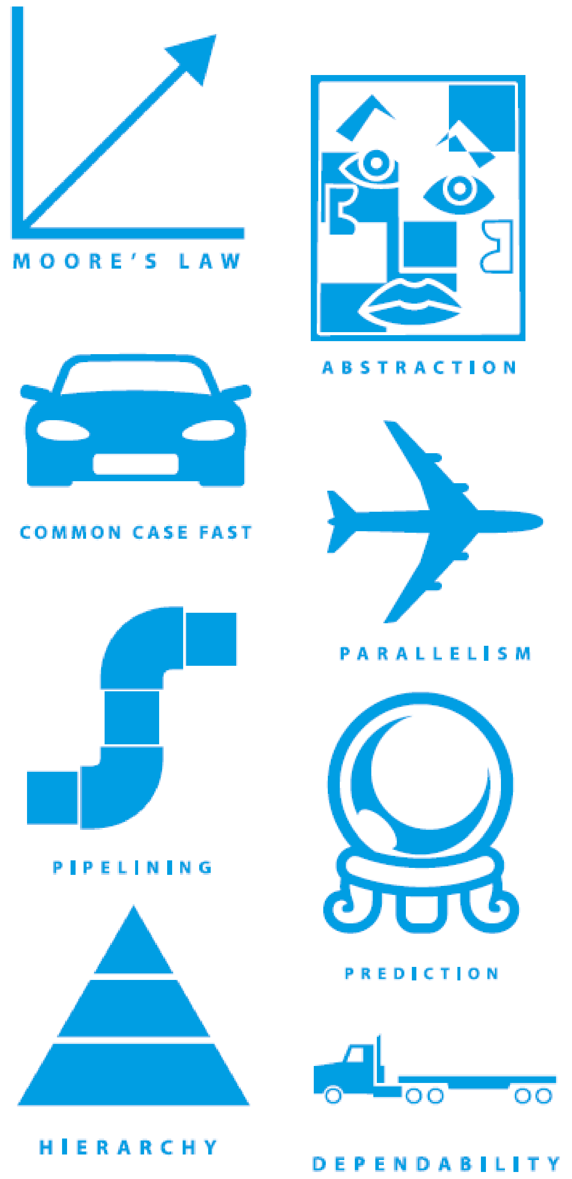
# Single Core Performance



Constrained by power, instruction-level parallelism, memory latency

# Eight Great Ideas

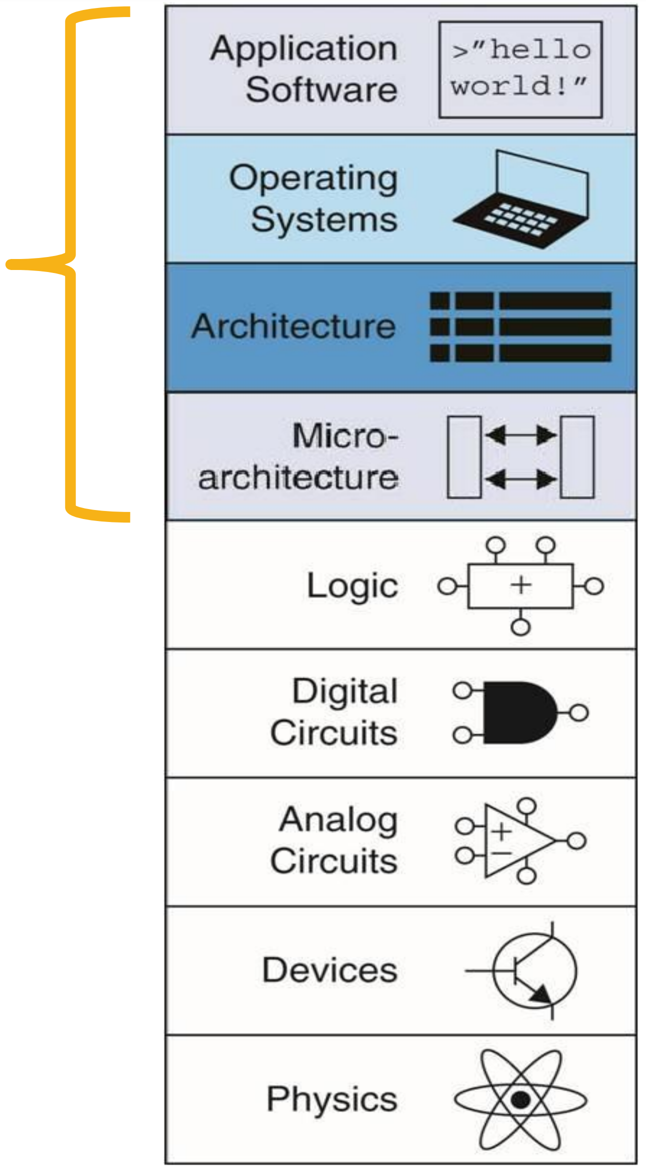
- Design for ***Moore's Law***
- Use ***abstraction*** to simplify design
- Make the ***common case fast***
- Performance *via* ***parallelism***
- Performance *via* ***pipelining***
- Performance *via* ***prediction***
- ***Hierarchy*** of memories
- ***Dependability*** *via* redundancy



# Abstraction

- Hiding details when they are not important

Focus of this course



# Any Questions?

```
           .text
__start:   addi t1, zero, 0x18
           addi t2, zero, 0x21
cycle:     beq t1, t2, done
           slt t0, t1, t2
           bne t0, zero, if_less
           nop
           sub t1, t1, t2
           j cycle
           nop
if_less:   sub t2, t2, t1
           j cycle
done:      add t3, t1, zero
```