SE 230
Computer Organization

Lecture 01: Introduction

Liang Yanyan

澳門科技大學
Macau of University of Science and Technology

General References

- Instructor: Dr. Liang Yanyan (梁延研)
  - Email: yyliang@must.edu.mo
  - Tel: 88971997
  - Office: A213
- TA:
  - To be confirmed

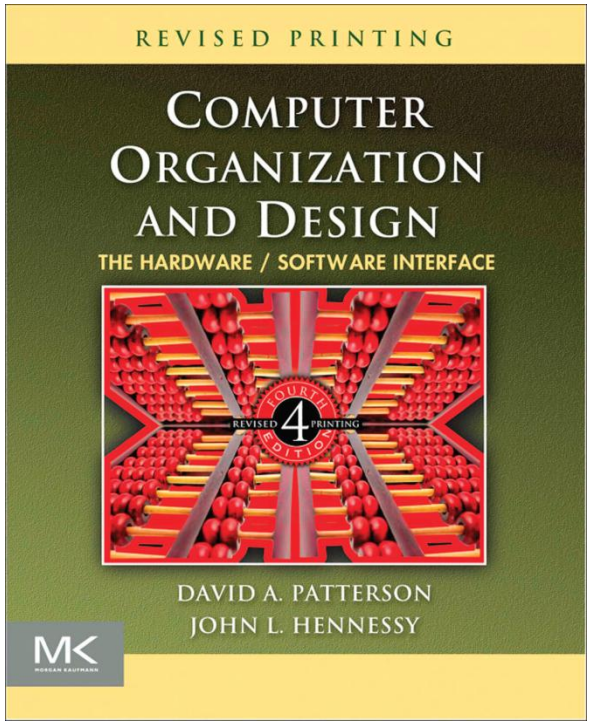
- Please submit your assignments and lab reports to Moodle.

- You are encouraged to ask questions during the lecture or after, or stop by my office.

- Classroom

General References

- Textbook: Computer Organization and Design: the Hardware/Software Interface – 4th Edition, David Patterson and John Hennessy.
- Some Resource in Moodle



Grading Information

- Grade determinates
  - Attendance 5%
  - Assignments 8%
  - Quizzes 15%
  - Labs 10%
  - Project 12%
  - Final Exam 50%

- Late submission per day is subject to 10% of penalty.
- A student must gain at least 40% of the full marks in each part in order to pass the course.

Why Learn This Stuff?

- You want to call yourself a “computer scientist/engineer”.
- You want to build HW/SW system people use.
- You need to make a purchasing decision or offer “expert” advice.

- So need to know the relationship between performance and power.
  - Both hardware and software affect performance/power. Because
    - Algorithm determines number of source-level statements.
    - Language/compiler/architecture determine the number of machine-level instructions.
    - Processor/memory determines how fast and how power-hungry machine-level instructions are executed.

Course Contents

- Introduction to the major components of a computer system, how they function together in executing a program.
- Introduction to CPU datapath and control unit design.
- Introduction to techniques to improve performance and energy efficiency of computer systems.
- Introduction to multiprocessor architecture.

- This course is to learn what determines the capabilities and performance of computer systems,
- and to understand the interactions between the computer's architecture and its software,
- so that
  - future software designers (compiler writers, operating system designers, database programmers, application programmers, …) can achieve the best cost-performance trade-offs,
- and so that
  - future architects understand the effects of their design choices on software.

What You Will Learn

- How programs are translated into the machine language,
  - and how the hardware executes them.
- What determines program performance,
  - and how it can be improved.
- How hardware designers improve performance.

What You Should Already Know

- Electronic circuit and digital logic.
- Knowledge of structured programming languages
  - Create, compile, and run C (C++, Java) programs

Computer Organization

- This course is all about how computers work.
- But what do we mean by a computer?
  - Different types: embedded, laptop, desktop, server.
  - Different uses: automobiles, graphics, finance, genomics…
  - Different manufacturers: Lenovo, Apple, IBM, HP, Sony…
- Analogy: Consider a course on “automotive vehicles”.
  - Many similarities from vehicle to vehicle (e.g., wheels).
  - Huge differences from vehicle to vehicle (e.g., gas vs. electric).
- Best way to learn:
  - Focus on a specific instance and learn how it works,
  - While learning general principles and historical perspectives.

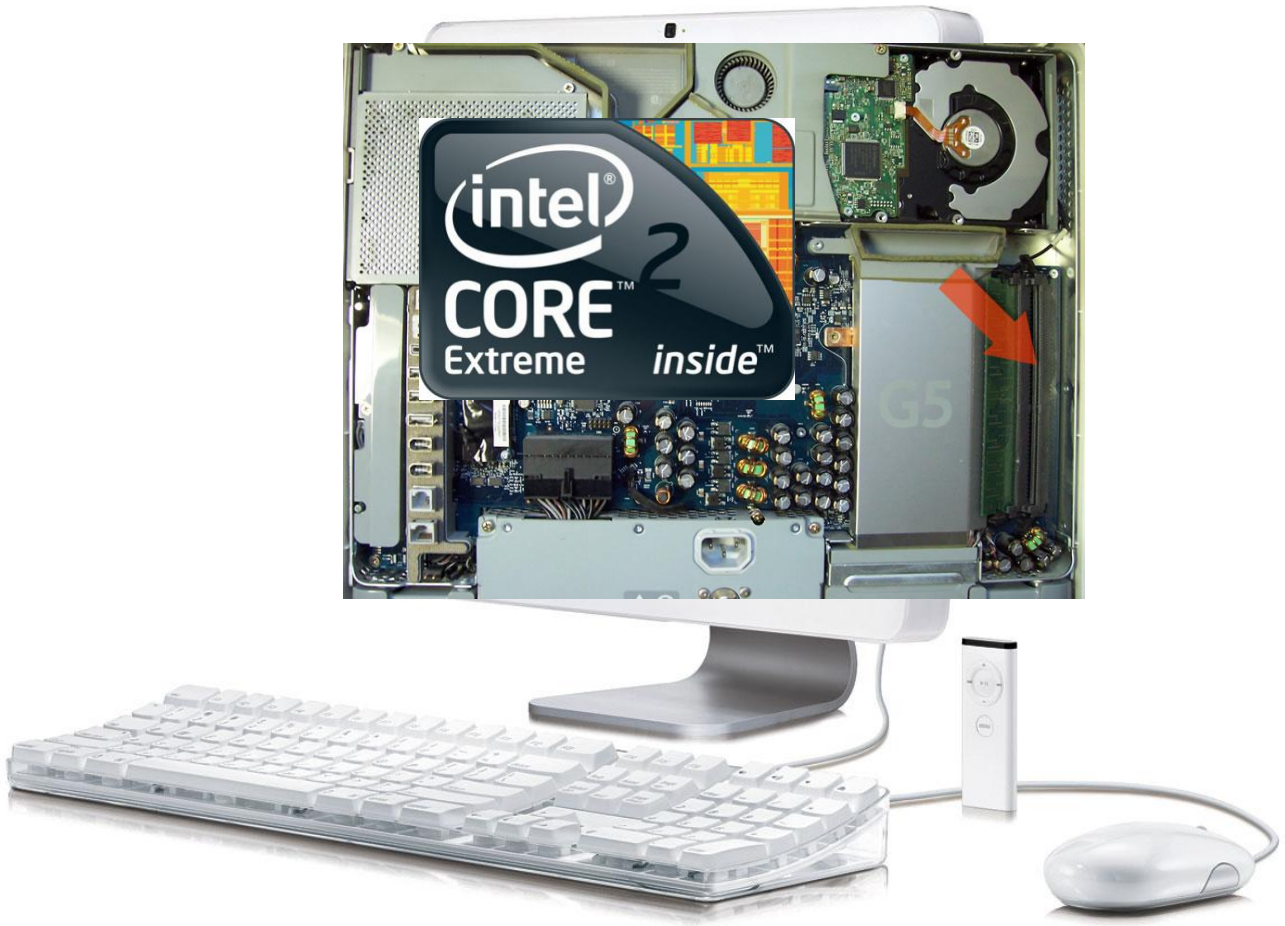
A Computer

Are there other kind of computers?

A Computer

Are there other kind of computers?

A Computer



Are there other kind of computers?

Classes of Computers

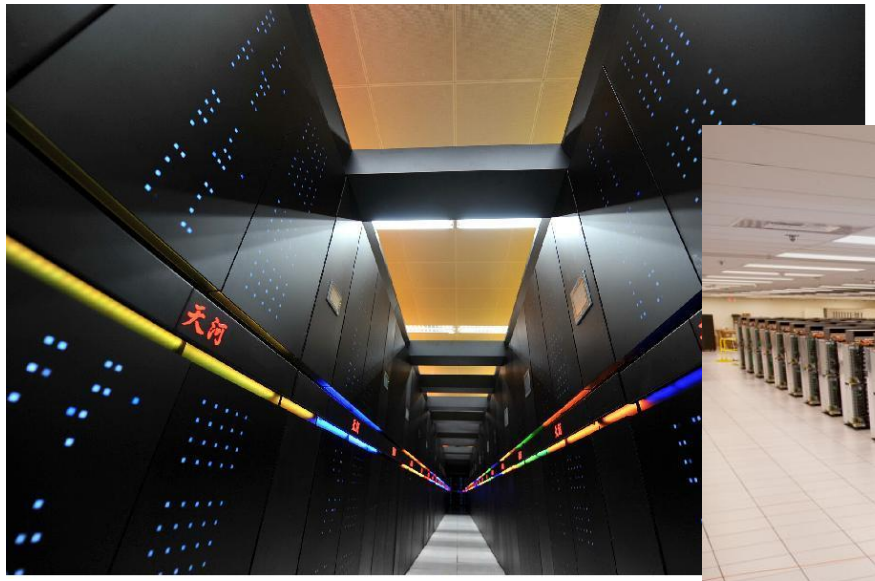
- Desktop computers
  - Designed to deliver good performance to a single user at low cost usually executing 3rd party software, usually incorporating a graphics display, a keyboard, and a mouse.
- Servers
  - Used to run larger programs for multiple, simultaneous users typically accessed only via a network and that places a greater emphasis on dependability and (often) security.

Classes of Computers

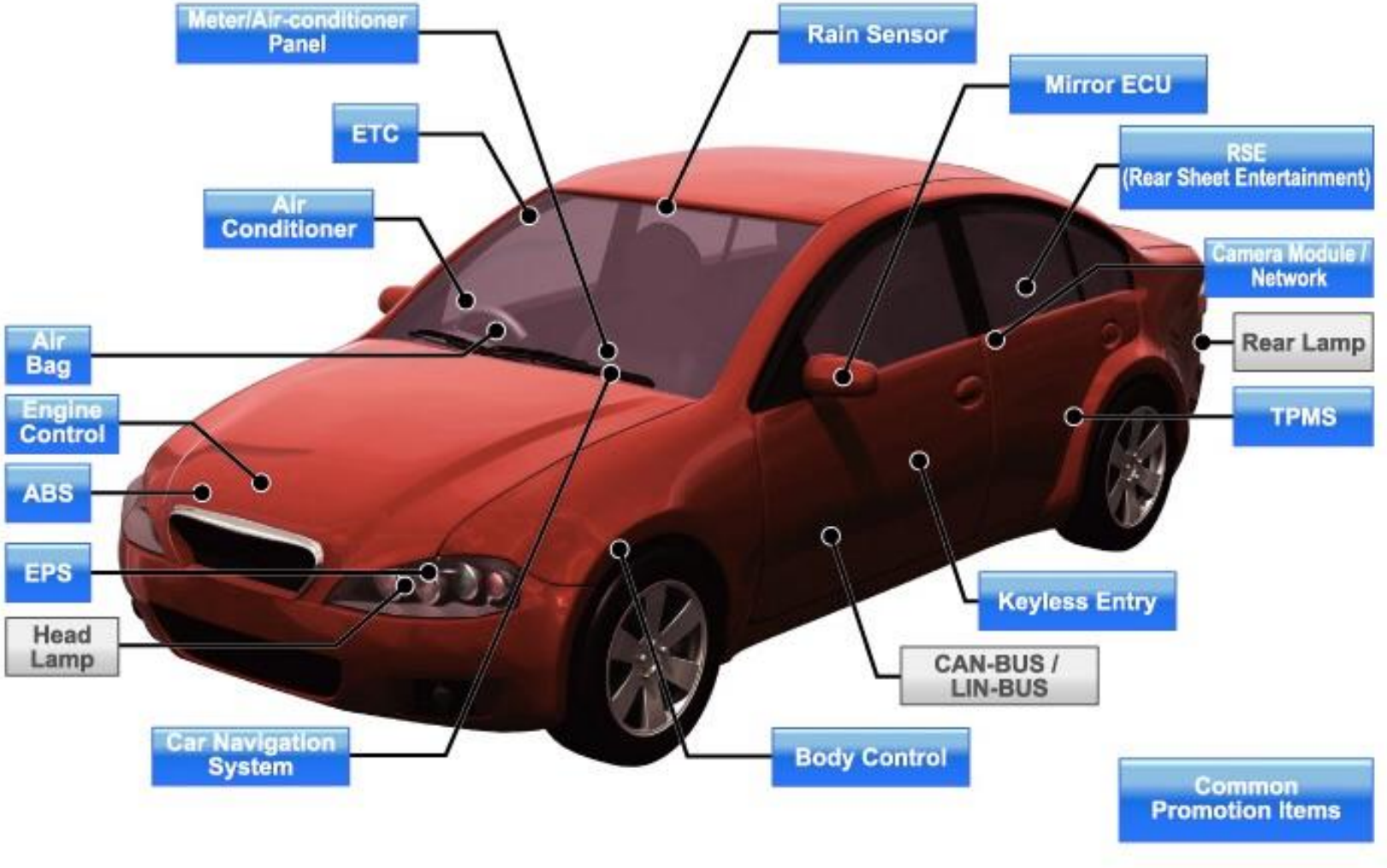
- Supercomputers
  - A high performance, high cost class of servers with hundreds to thousands of processors, terabytes of memory and petabytes of storage that are used for high-end scientific and engineering applications.
- Embedded computers (processors)
  - A computer inside another device used for running one predetermined application.

Supercomputers

- Tianhe-2 (天河-2)
  - Over 3 million cores
  - Power: 17.6 MW (24 MW with cooling)
  - Speed: 33.86 PFLOPS (peta = 10^{15})

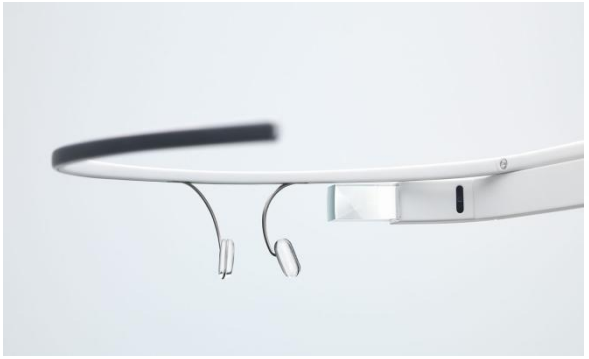


Embedded Computers in You Car



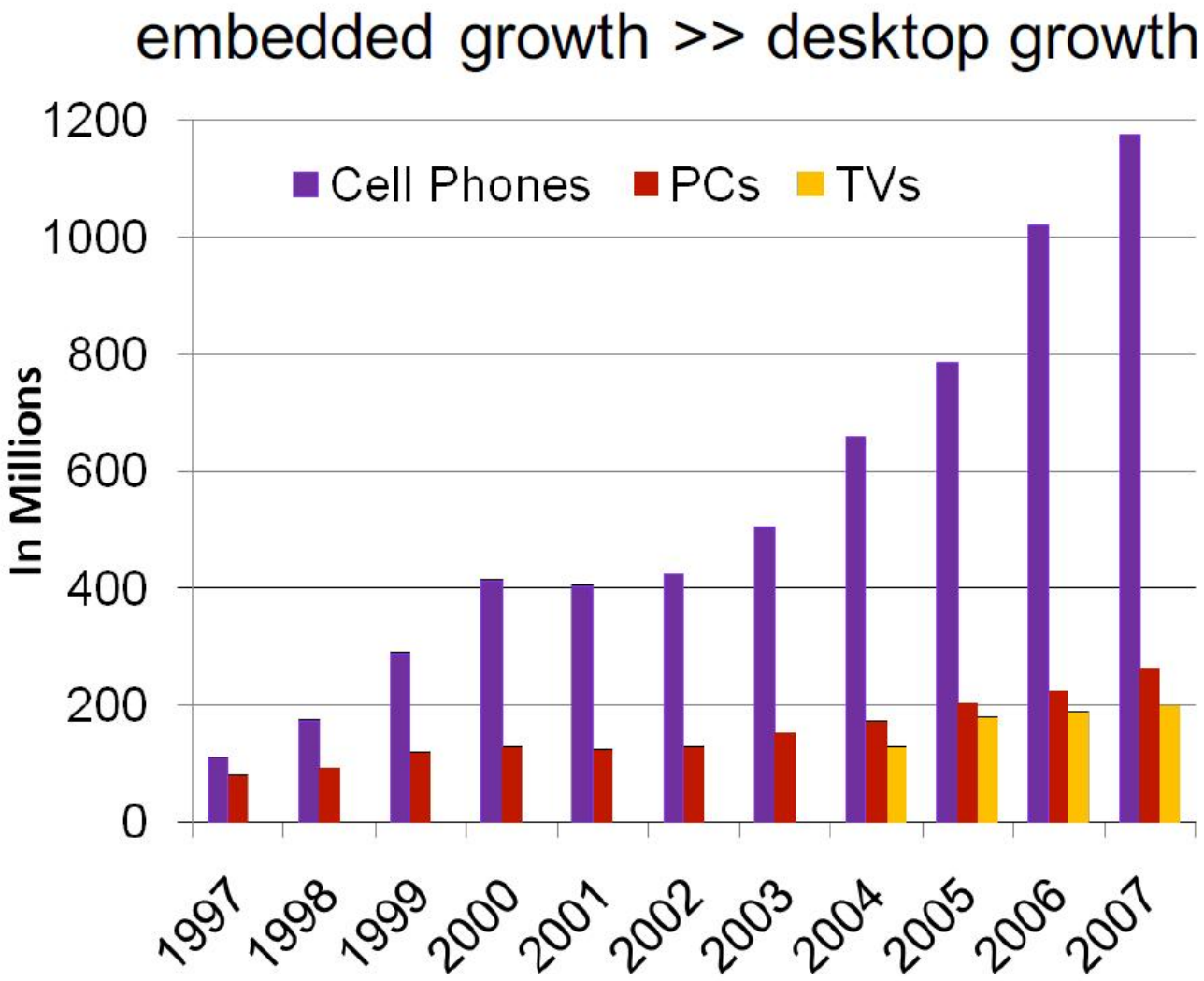
PostPC Era

- Personal Mobile Device (PMD) and wearable devices.



- Where else are embedded processors found?

Growth in Cell Phone Sales (Embedded)



xPUs


我们是谁
xPU!
为神马生我们出来
蹭热点!
谁是老大
CPU!
我们赚不赚钱？
不赚钱！
我们的任务是
干掉他！
谁会买我们？
不知道！

xPUs

- APU
  - Accelerated Processing Unit
    - Designed by AMD
    - Tasks are performed by CPU and GPU
  - Audio Processing Unit
- BPU
  - Biological Processing Unit
    - Simulating the Brain
- CPU
  - Central Processing Unit
- DPU
  - Deep-Learning Processing Unit
- EPU
  - Emotion Processing Unit
    - emotion synthesis

xPUs

- FPU
  - Floating Point Unit
- GPU
  - Graphics Processing Unit
- HPU
  - Holographic Processing Unit
    - for Microsoft Hololens apps.
- IPU
  - Intelligence Processing Unit
    - for graph(network) computing.
- KPU
  - Knowledge Processing Unit
- MPU
  - Micro Processing Unit

xPUs

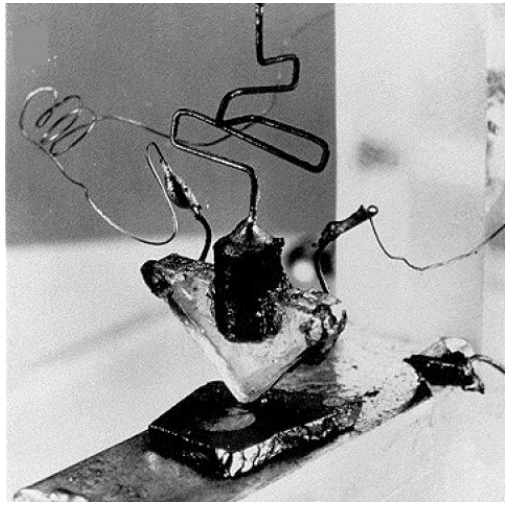
- NPU
  - Neural-Network Processing Unit
- OPU
  - Optical-Flow Processing Unit
- PPU
  - Physical Processing Unit
    - PhysX, have combined in GPU
- QPU
  - Quantum Processing Unit
- RPU
  - Ray-tracing Processing Unit
- SPU
  - Streaming Processing Unit

xPUs

- TPU
  - Tensor Processing Unit
- VPU
  - Vision Processing Unit
- WPU
  - Wearable Processing Unit

The Evolution of Computer Hardware

- When was the first transistor invented?

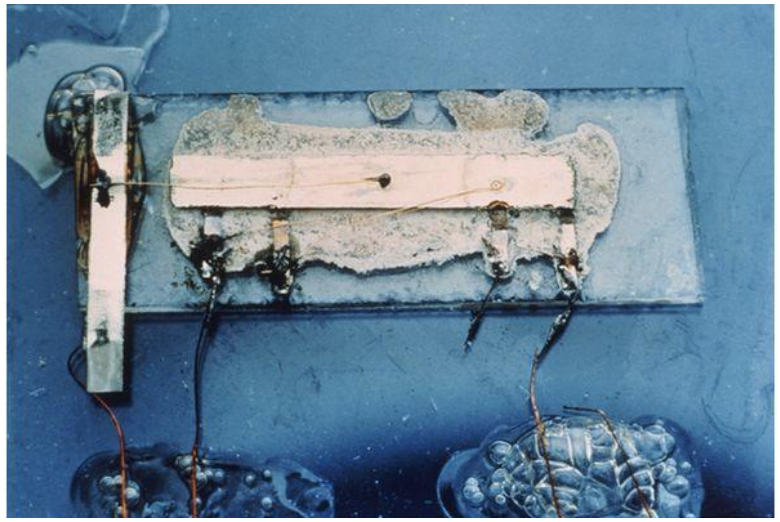


1947 - the bi-polar transistor – by Bardeen *et.al* at Bell Laboratories

UNIVAC I (Universal Automatic Computer) – the first commercial computer in USA

The Evolution of Computer Hardware

- When was the first IC (integrated circuit) invented?

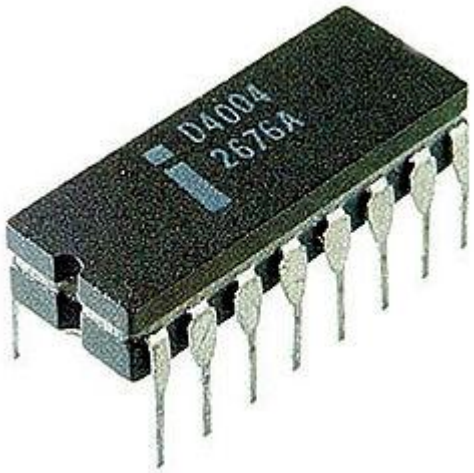


1958, by Jack Kilby@Texas Instruments, by hand, several transistors, resistors and capacitors on a single substrate.

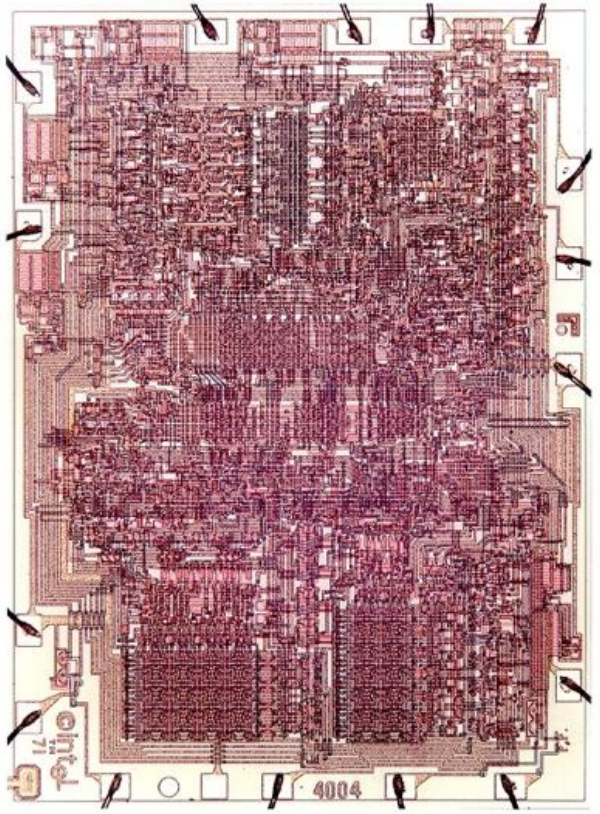
IBM System/360, 2MHz, 128KB ~ 256KB

The Evolution of Computer Hardware

- When was the first Microprocessor?



1971, Intel 4004

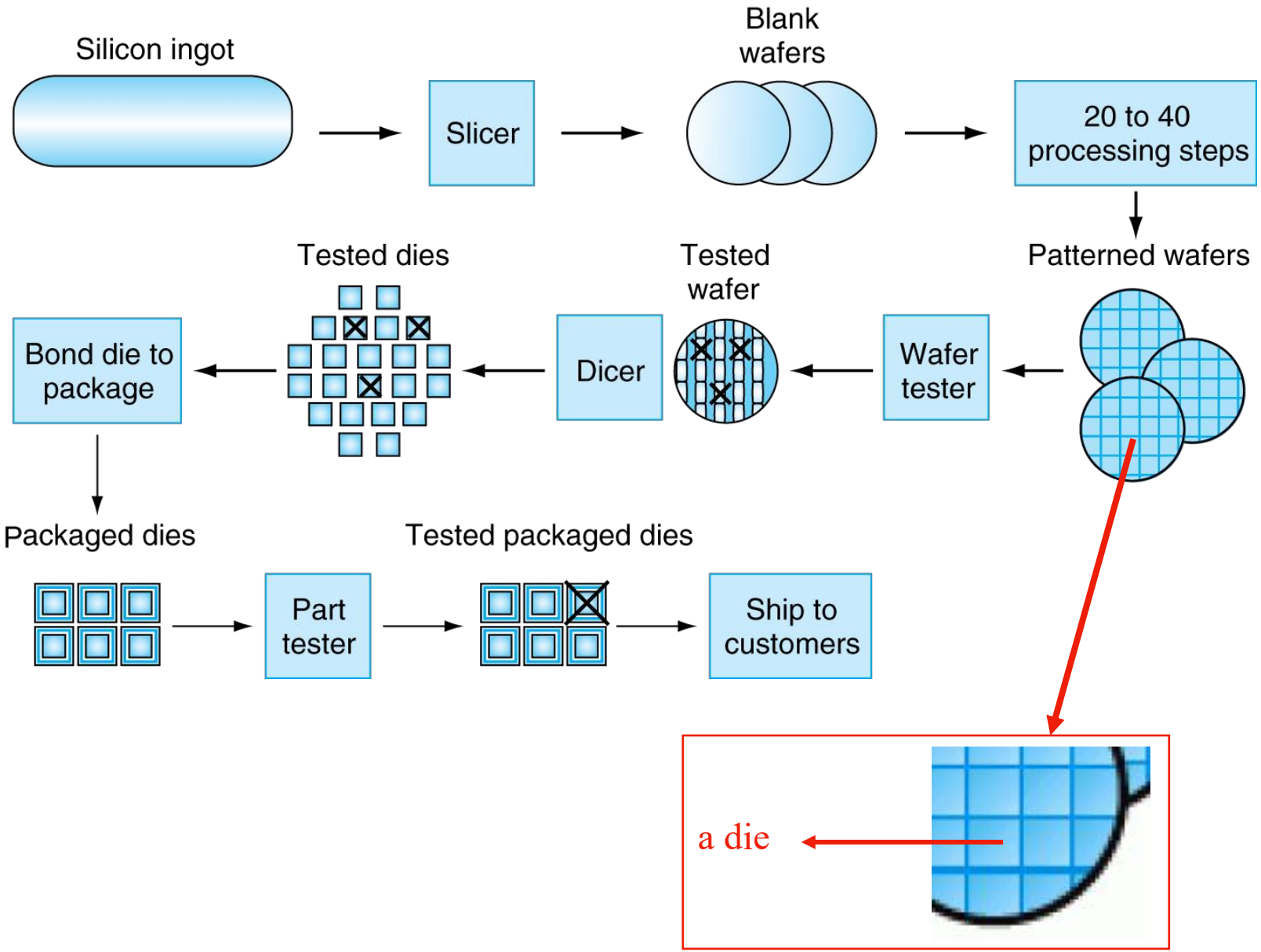


The History of Intel Processors: https://www.youtube.com/watch?v=Qu2njWY3Hjk

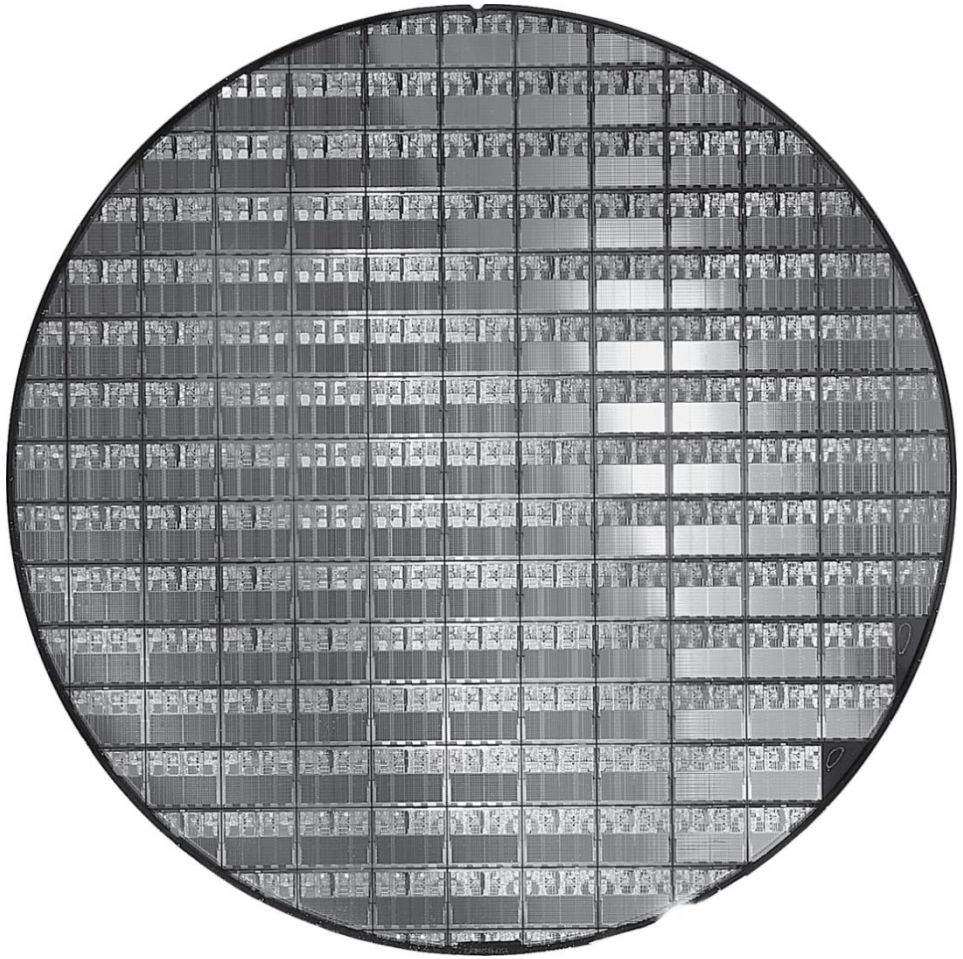
Semiconductor Technology

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

The Chip Manufacturing Process



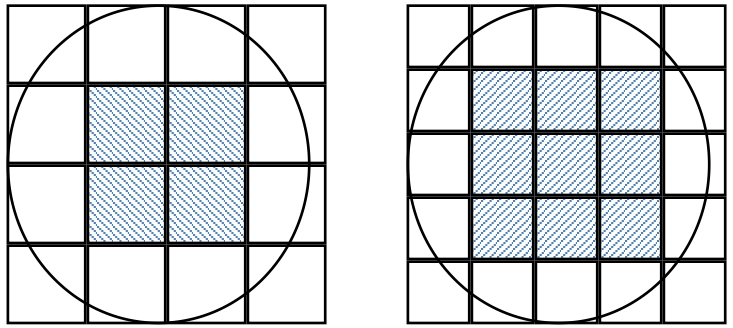
AMD Opteron X2 Wafer



300mm wafer, 117 chips, with 90nm technology

How a CPU is made: https://www.youtube.com/watch?v=qm67wbB5GmI&t=372s

Integrated Circuit Cost



Ideal case:

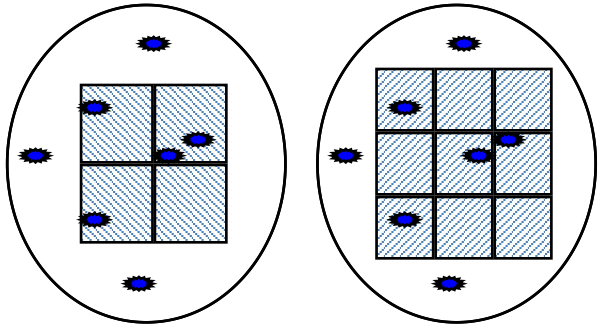
$$Die\ Cost \approx \frac{Cost\ of\ wafer}{Dies\ per\ wafer}$$

$$Dies\ per\ wafer = \frac{\pi \times (Wafer\ diameter / 2)^2}{Die\ Area} - \frac{\pi \times Wafer\ diameter}{\sqrt{2 \times Die\ Area}} - Test\ dies\ per\ wafer$$

No. of testing dies for characteristics testing

No. of dies at the edge
≈ circumference/diagonal of die

Integrated Circuit Cost (Die yield)



$$Die\ Cost = \frac{Cost\ of\ wafer}{Dies\ per\ wafer \times Die\ yield}$$

$$Die\ yield = \{1 + \frac{Defects\ per\ unit\ area \times Die\ area}{\alpha}\}^{-\alpha}$$

Be referred to No. of critical processing steps in the manufacturing process

$$Defects\ per\ unit\ area = \frac{\alpha}{Die\ area}(Die\ yield^{-1/\alpha} - 1)$$

Integrated Circuit Cost (Example)

- What is the approximate cost of a die in the wafer?
  - An 8-inch wafer costs $1000
  - Defect density is 1 per cm^2
  - Die area is 91 mm^2
  - Assume α = 2, test dies per wafer is 10

$$\text{Die yield} = \left(1 + \frac{1 \times 0.91}{2}\right)^{-2} = 0.47$$

$$\text{Dies per wafer} = \frac{\pi \times (8 \times 2.54/2)^2}{0.91} - \frac{\pi \times 8 \times 2.54}{\sqrt{2 \times 0.91}} - 10$$

$$\text{Die Cost} = \frac{1000}{(\text{Dies per wafer} \times 0.47)}$$

Real World Examples

- 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

Chip	*Metal layers*	*Line width*	*Wafer cost*	*Defect /cm²*	*Area mm²*	*Dies/ wafer*	*Yield*	*Die Cost*
386DX	2	0.90	$900	1.0	43	360	71%	$4
486DX2	3	0.80	$1200	1.0	81	181	54%	$12
PowerPC 601	4	0.80	$1700	1.3	121	115	28%	$53
HP PA 7100	3	0.80	$1300	1.0	196	66	27%	$73
DEC Alpha	3	0.70	$1500	1.2	234	53	19%	$149
SuperSPARC	3	0.70	$1700	1.6	256	48	13%	$272
Pentium	3	0.80	$1500	1.5	296	40	9%	$417

From "Estimating IC Manufacturing Costs," by Linley Gwennap, *Microprocessor Report*, August 2, 1993, p. 15

Die cost goes up with the die area.

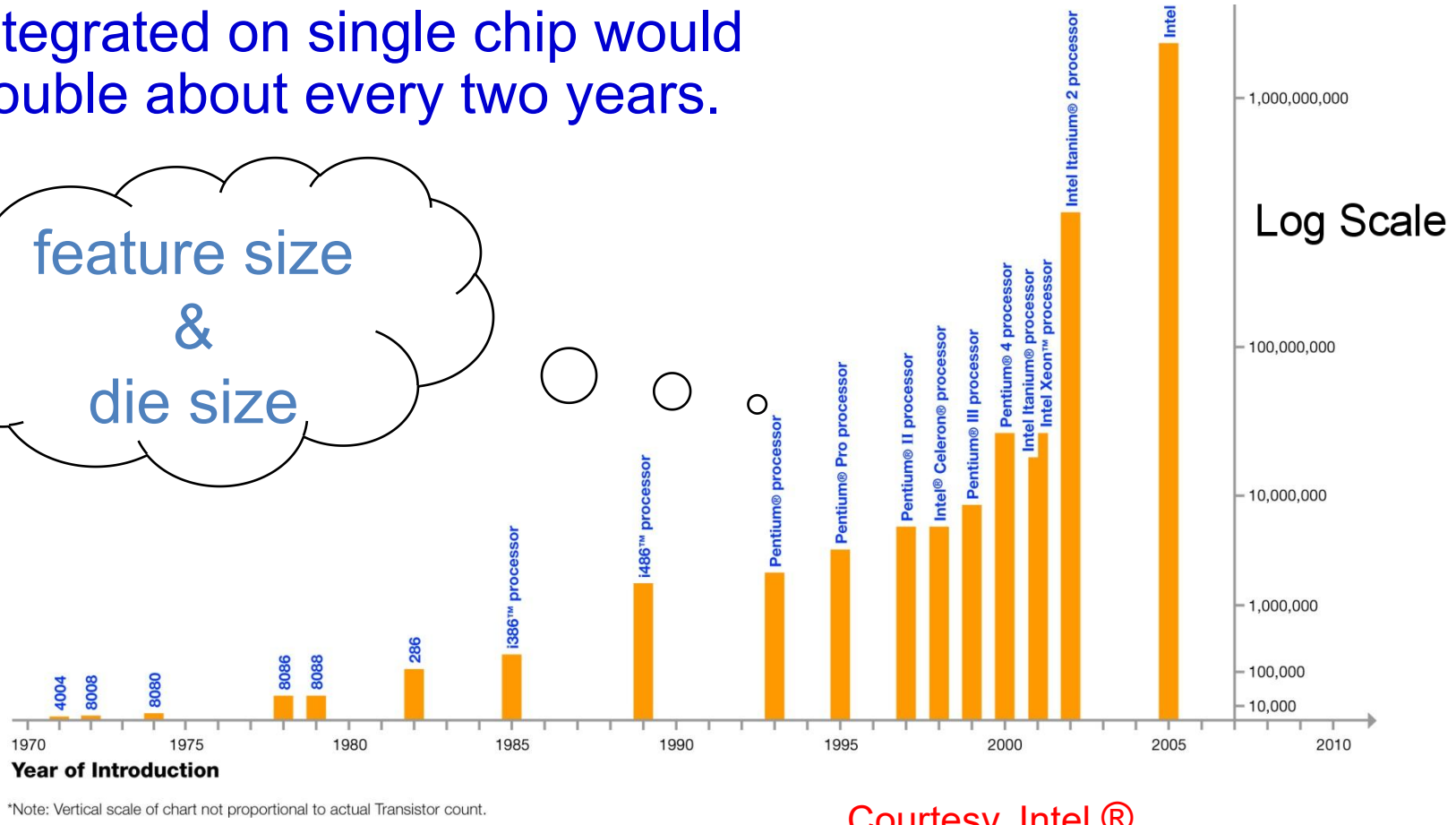
Impacts of Advancing Technology

- Processor
  - logic capacity: increases about 30% per year
  - performance: increases 2x every 1.5 years
- Memory
  - DRAM capacity: increases 4x every 3 years, about 60% per year
  - memory speed: increases 1.5x every 10 years
  - cost per bit: decreases about 25% per year
- Disk
  - capacity: increases about 60% per year

Moore's Law

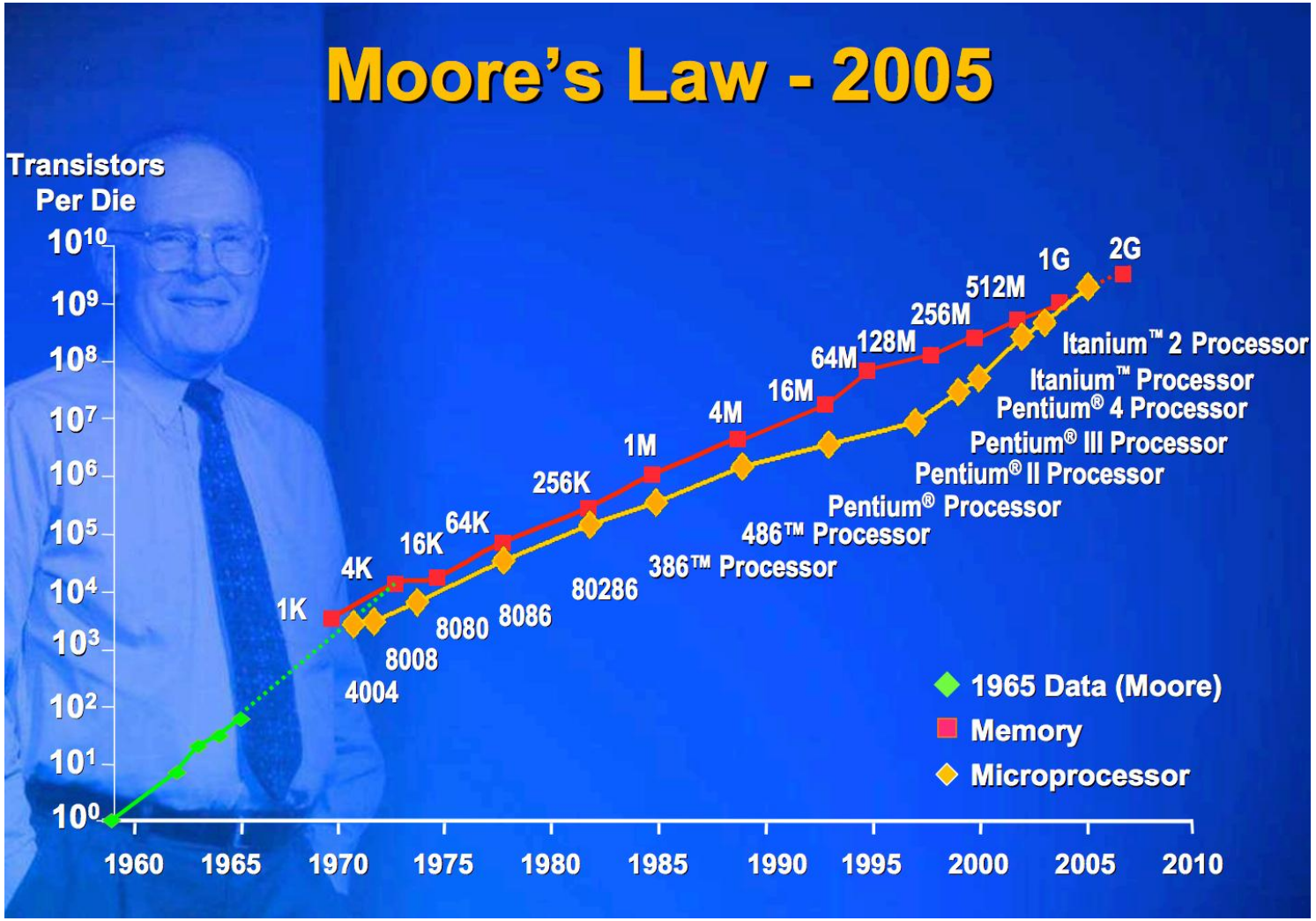
- In 1965, Intel's Gordon Moore predicted that the number of transistors that can be integrated on single chip would double about every two years.

feature size
&
die size



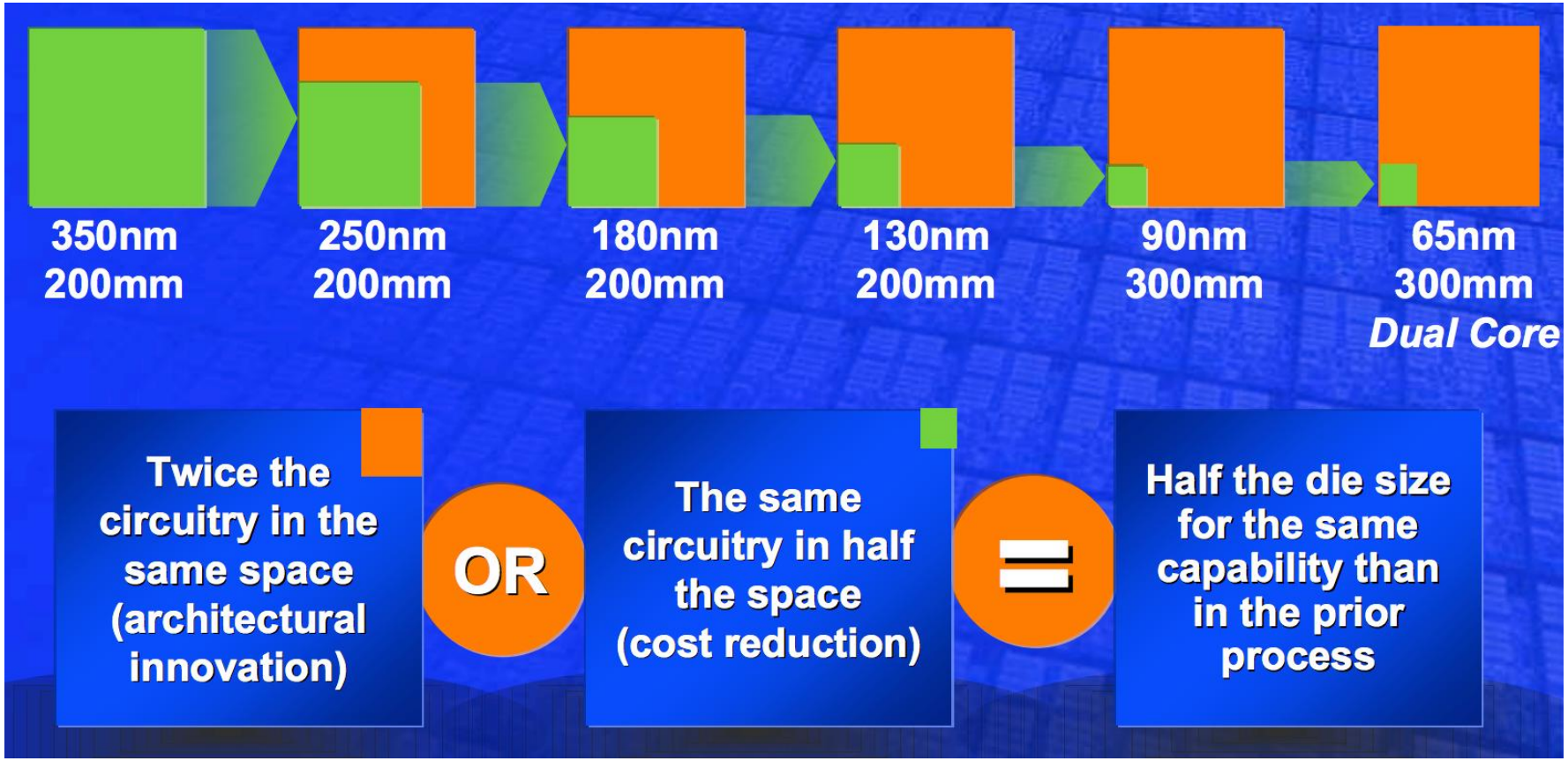
Courtesy, Intel ®

Moore's Law for CPUs and DRAMs



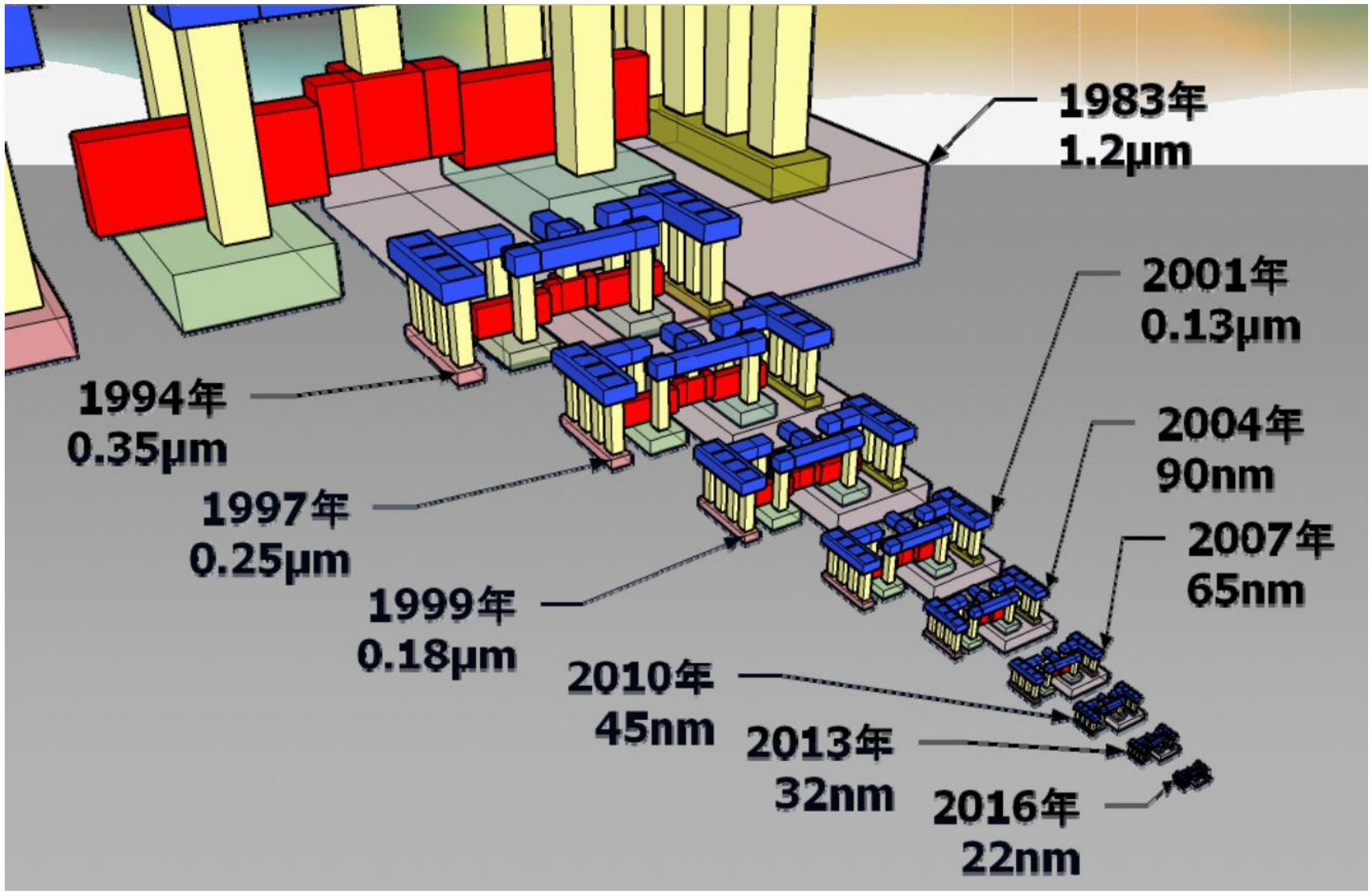
From: "Facing the Hot Chips Challenge Again", Bill Holt, Intel, presented at Hot Chips 17, 2005.

Main driver: device scaling ...

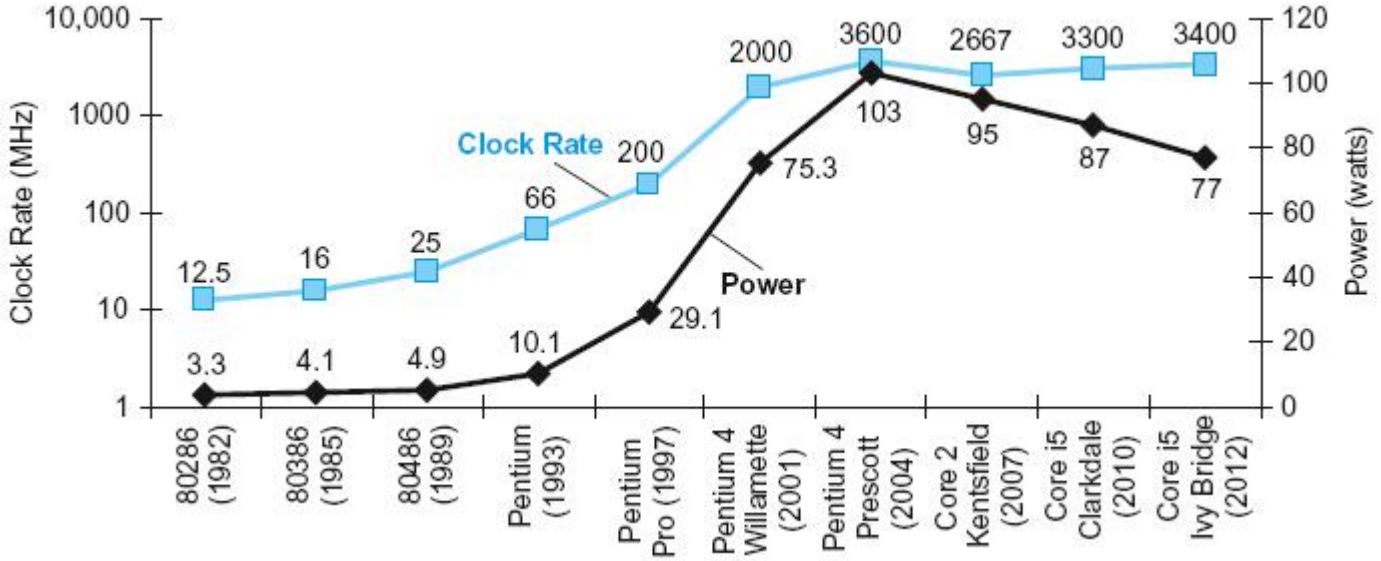


From: “Facing the Hot Chips Challenge Again”, Bill Holt, Intel, presented at Hot Chips 17, 2005.

Main driver: device scaling ...



Highest Clock Rate of Intel Processors



- In CMOS (Complementary Metal-Oxide-Semiconductor) IC technology

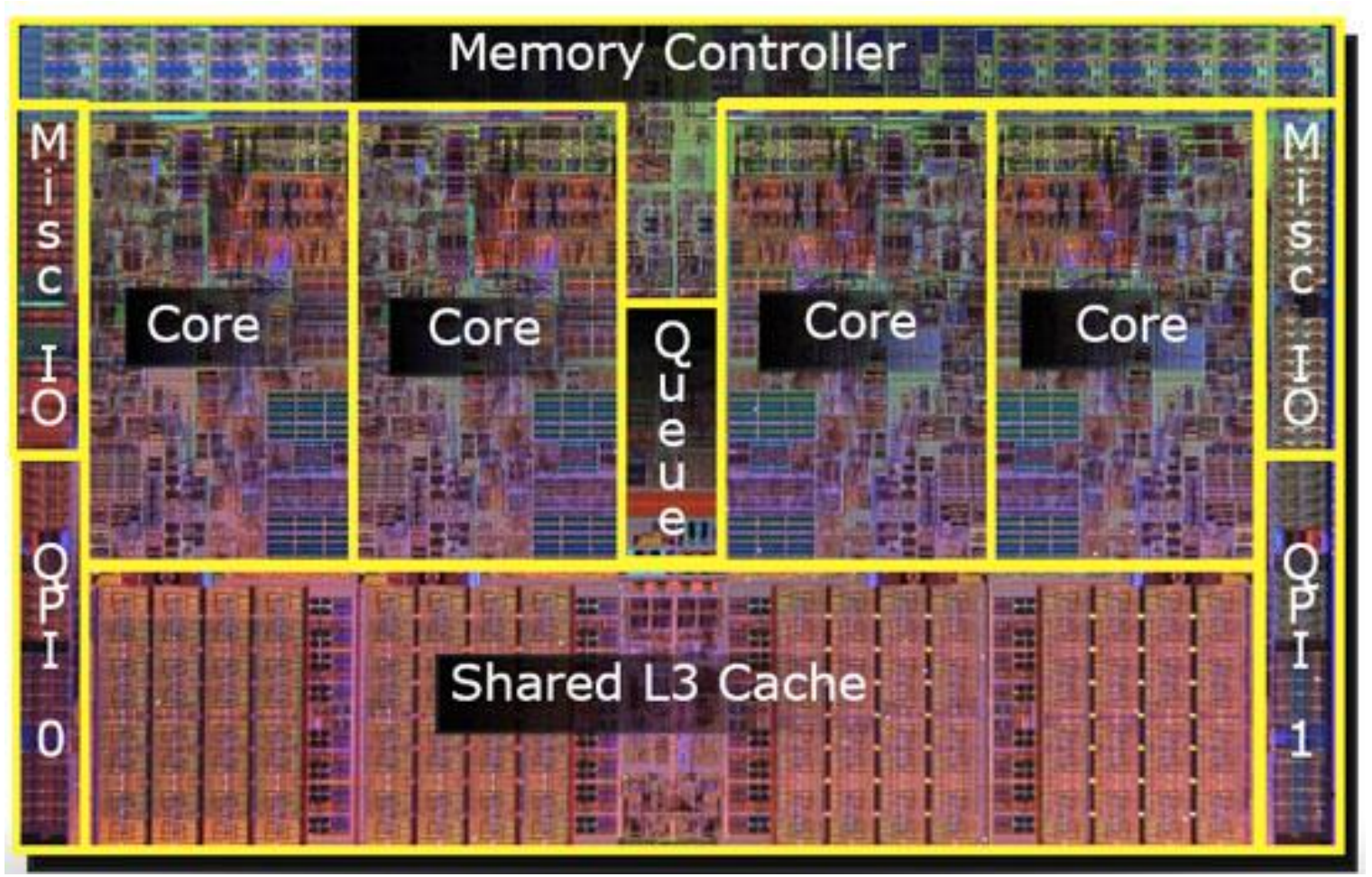
$$\text{Power} = \text{Capacitive load} \times \text{Voltage}^2 \times \text{Frequency}$$

$\times 30$ | $5\text{V} \rightarrow 1\text{V}$ | $\times 1000$

A Sea Change is at Hand

- The power challenge has forced a change in the design of microprocessors.
  - Since 2002 the rate of improvement in the response time of programs on desktop computers has slowed from a factor of 1.5 per year to less than a factor of 1.2 per year.
- As of 2006 all desktop and server companies are shipping microprocessors with multiple processors per chip.
- Plan of record is to add two cores per chip per generation (about every two years).
  - Pentium 4, 2 cores, 2002-2005
  - Core 2 Duo, 2-4 cores, 2006-2009
  - Core i7, 4-8 cores, 2010-now
  - Xeon, 1-15 cores, 1998-now

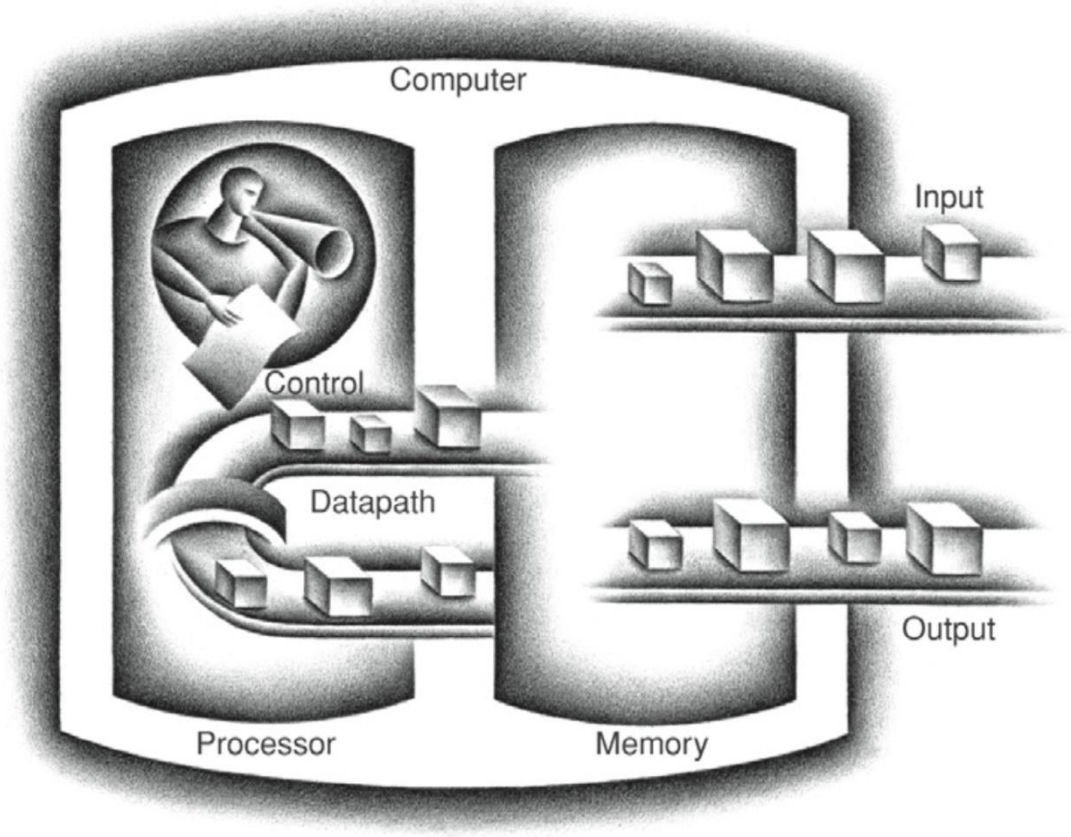
Intel Core i7 Processor



45nm technology, 18.9mm x 13.6mm, 0.73billion transistors, 2008

What is a Computer?

- Components:
  - processor (datapath, control)
  - input (mouse, keyboard)
  - output (display, printer)
  - memory (cache (SRAM), main memory (DRAM), disk drive, CD/DVD)



Four Issues about Machine Organization

- Capabilities and performance characteristics of the principal Functional Units (FUs).
  - Functional Unit: a hardware component that can perform specific operations (functions). For example, Adders, Registers, ALU, Shifters, Logic Units.
- The ways in which these FUs are interconnected.
  - e.g., buses.
- Information flows between components.
  - e.g., the data flow is fetched from memory and transferred to processor.
- Logic and means by which such information flow is controlled.

Our Primary Focus

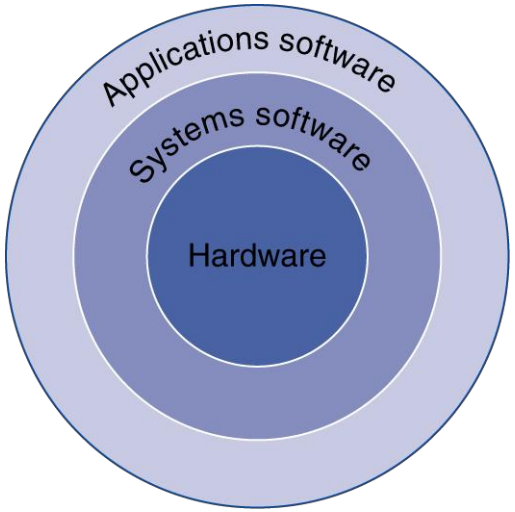
- Our primary focus: the processor (datapath and control) and its interaction with memory systems.
  - Implemented using tens/hundreds of millions of transistors.
  - Impossible to understand by looking at each transistor.
  - We need abstraction!

Processor Organization

- Control unit needs to have circuitry to
  - Decide which is the next instruction and input it from memory.
  - Decode the instruction.
  - Issue signals that control the way information flows between datapath components.
  - Control what operations the datapath's functional units perform.

- Datapath needs to have circuitry to
  - Execute instructions - functional units (e.g., adder) and storage locations (e.g., register file).
  - Interconnect the functional units so that the instructions can be executed as required.
  - Load data from and store data to memory.

Below the Program



- Application software
  - Written in high-level language, e.g. C, C++, java…
- System software
  - Operating system – supervising program that interfaces the user's program with the hardware (e.g., Linux, iOS, Windows).
    - Handles basic input and output operations.
    - Allocates storage and memory.
    - Provides for protected sharing among multiple applications.
  - Compiler – translates programs written in a high-level language (e.g., C, Java) into instructions that the hardware can execute.

Why We use Higher-Level Languages?

- Higher-Level Languages
  - Allow the programmer to think in a more natural language and for their intended use (Fortran for scientific computation, Cobol for business programming, Lisp for symbol manipulation, Java for web programming, …).
  - Improve programmer productivity – more understandable code that is easier to debug and validate.
  - Improve program maintainability.
  - Allow programs to be independent of the computer on which they are developed (compilers and assemblers can translate high-level language programs to the binary instructions of any machine).
- Emergence of optimizing compilers that produce very efficient assembly code optimized for the target machine.
  - As a result, very little programming is done today at the assembler level.

You can become programmers programming programs that program programs!
-- using AI!

Below the Program

- High-level language program (in C)

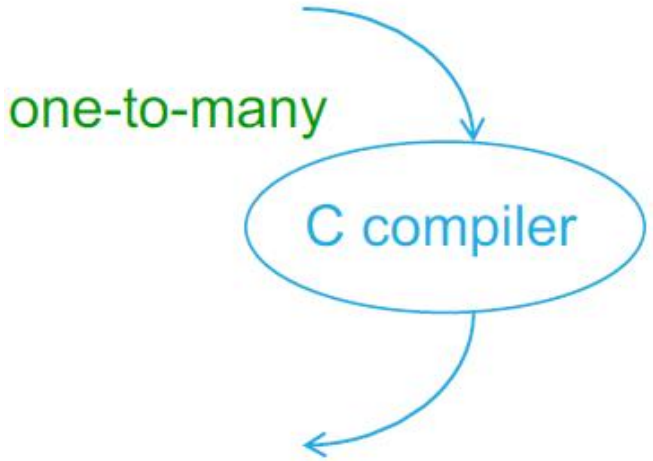
```
swap (int v[], int k)
(
    int temp;
    temp = v[k];
    v[k] = v[k+1];
    v[k+1] = temp;
)
```

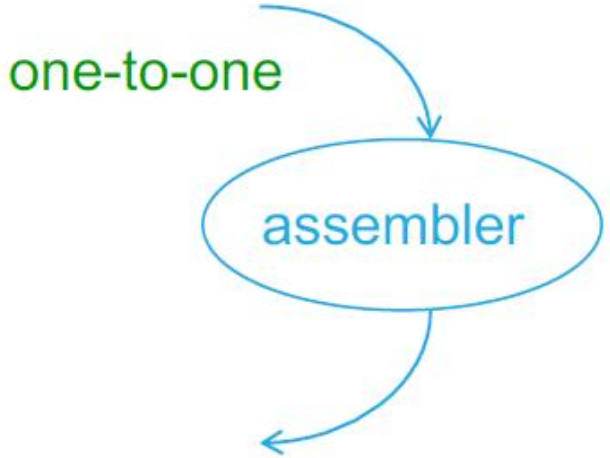
- Assembly language program (for MIPS)

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

- Machine (object) code (for MIPS)

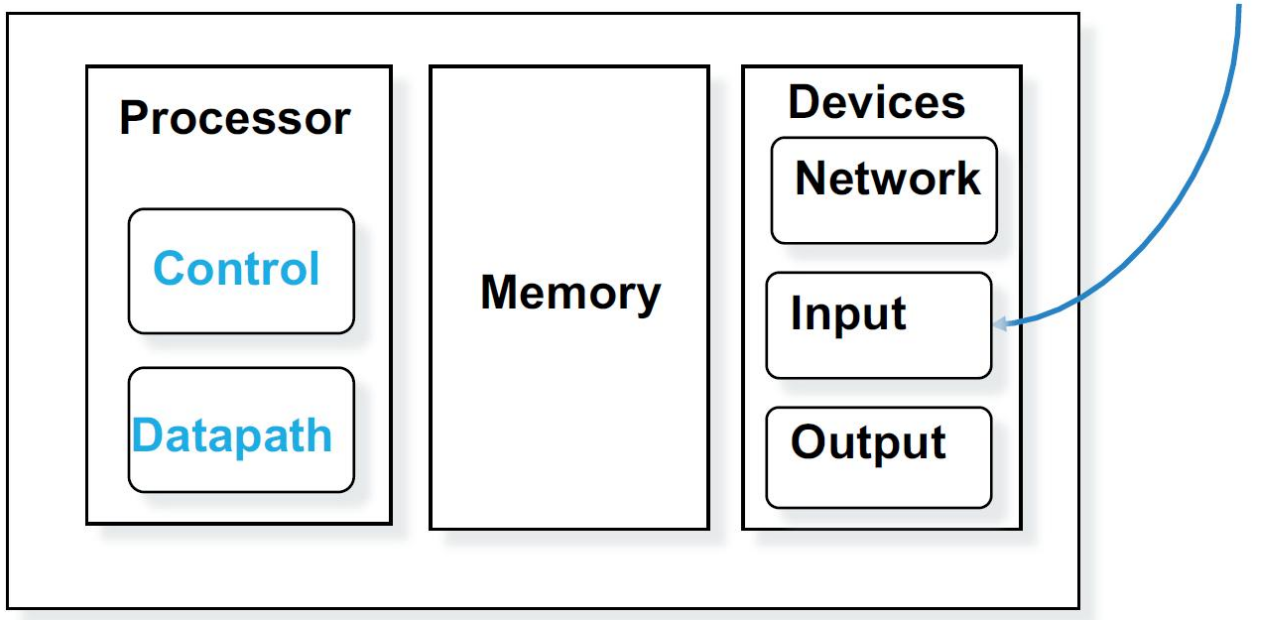
```
000000 00000 00101 0001000010000000
000000 00100 00010 0001000000100000
. . .
```



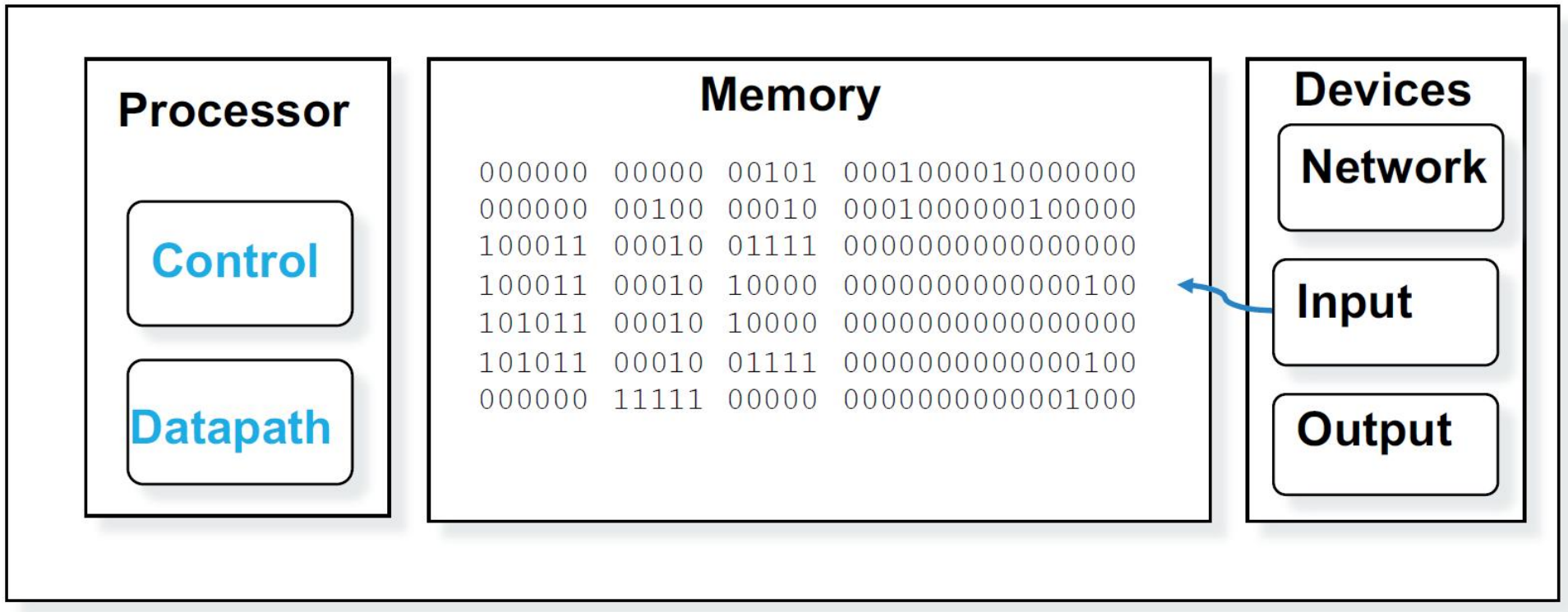


Input Device Inputs Object Code

```
000000 00000 00101 0001000010000000
000000 00100 00010 0001000000100000
100011 00010 01111 0000000000000000
100011 00010 10000 0000000000000100
101011 00010 10000 0000000000000000
101011 00010 01111 0000000000000100
000000 11111 00000 0000000000001000
```

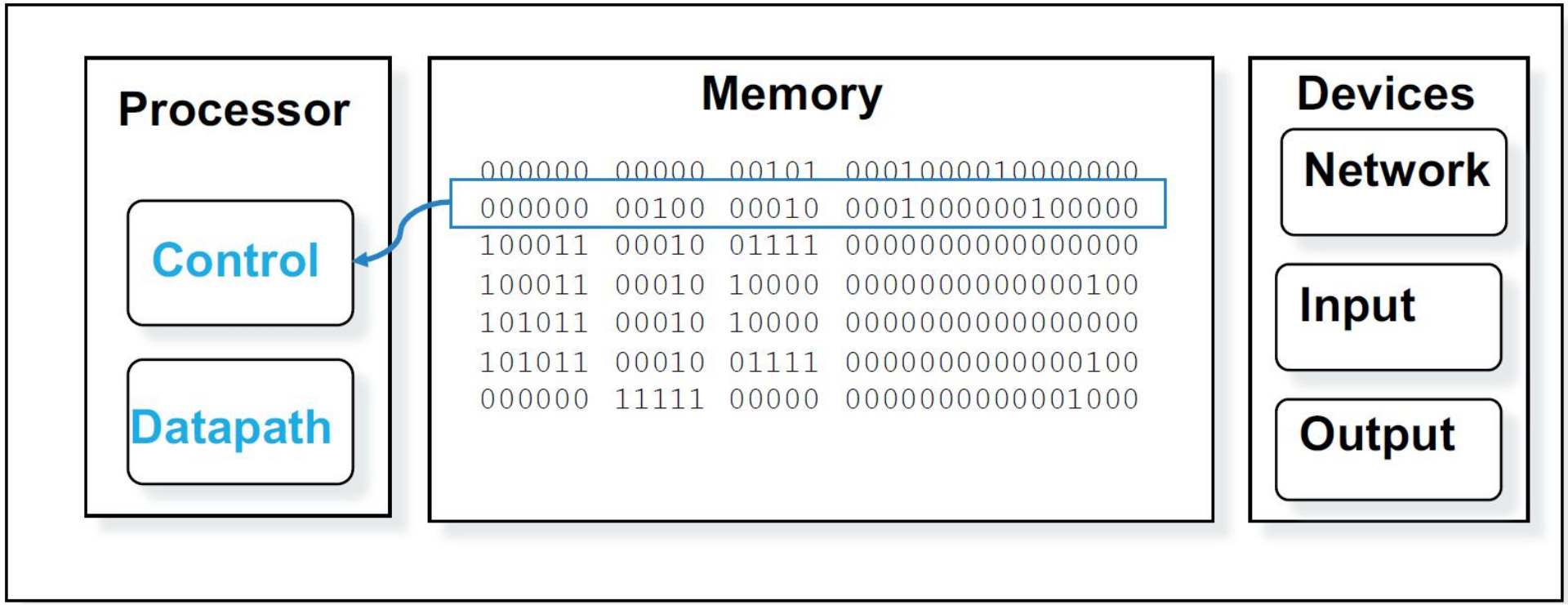


Object Code Stored in Memory



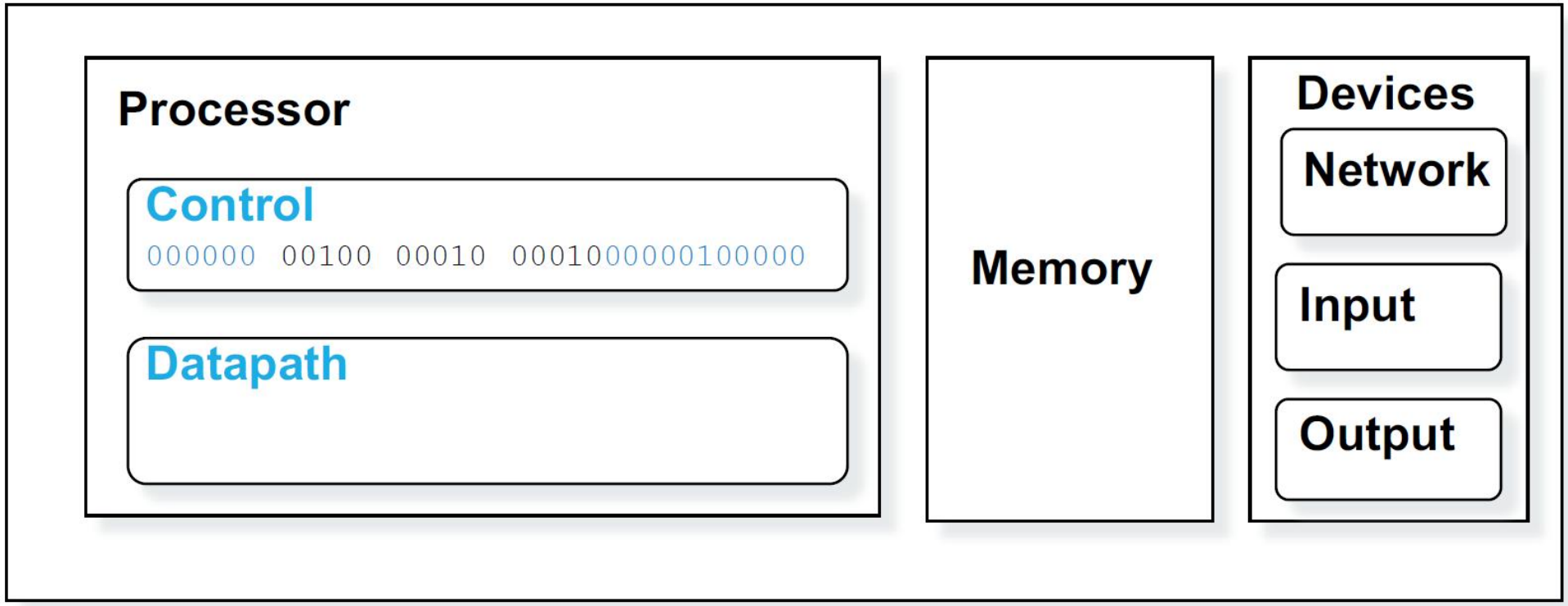
Processor Fetches an Instruction

- Processor fetches an instruction from memory.



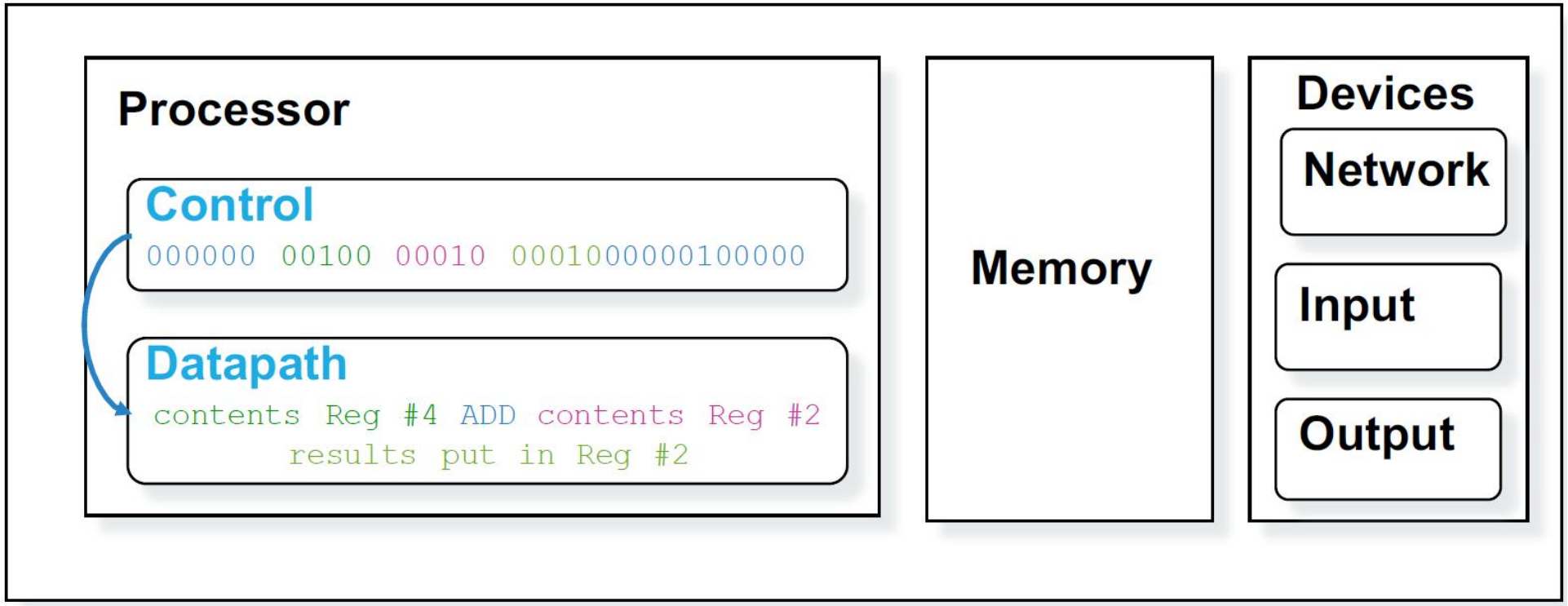
Control Decodes the Instruction

- Control decodes the instruction to determine what to execute.



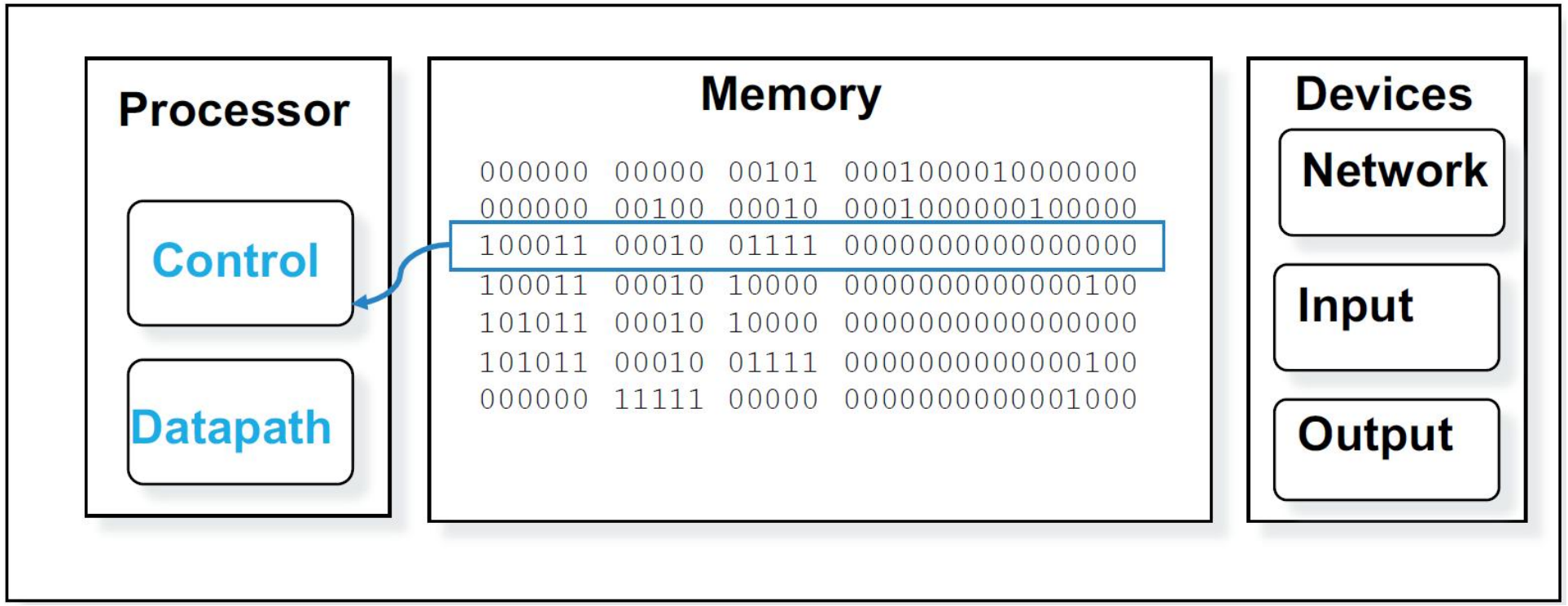
Datapath Executes the Instruction

- Datapath executes the instruction as directed by control.



Processor Fetches the Next Instruction

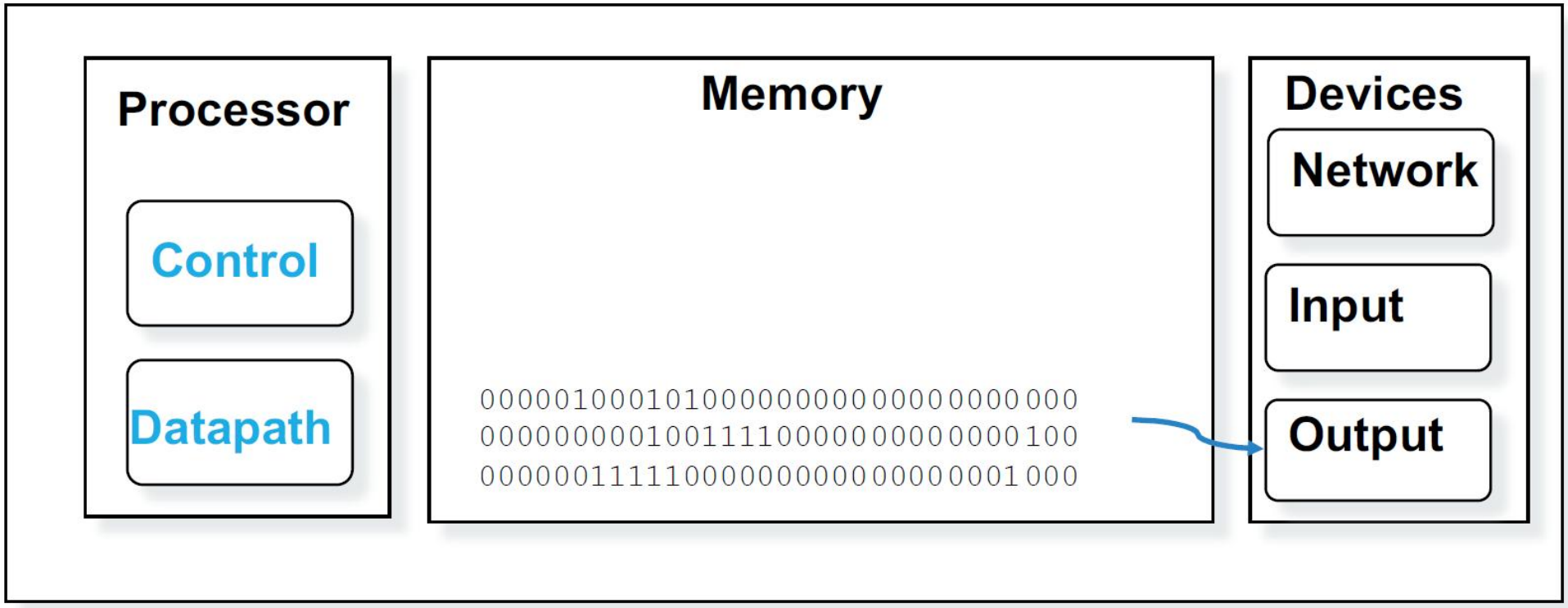
- Processor fetches the *next* instruction from memory.



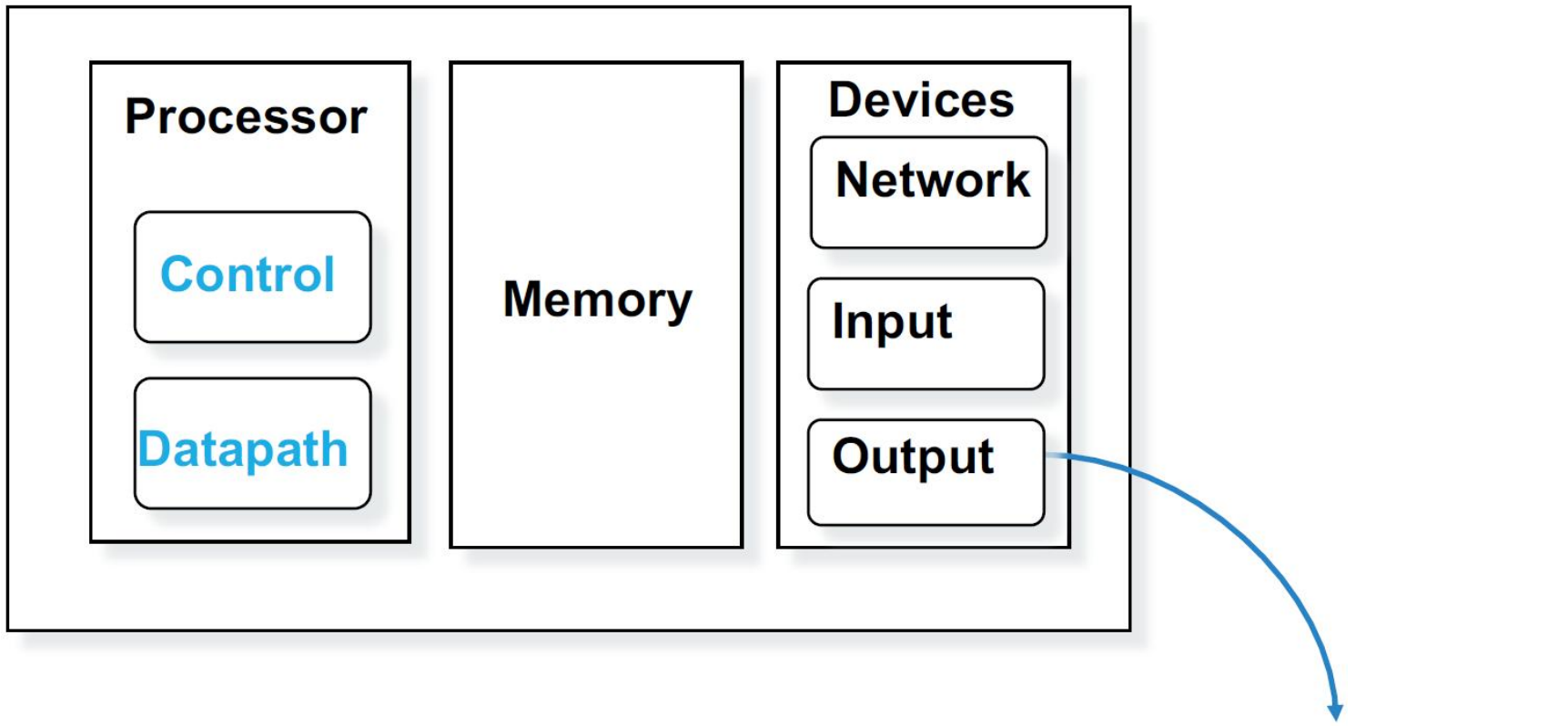
How does it know which location in memory to fetch from next?

Output Data Stored in Memory

- At program completion the data to be output resides in memory.



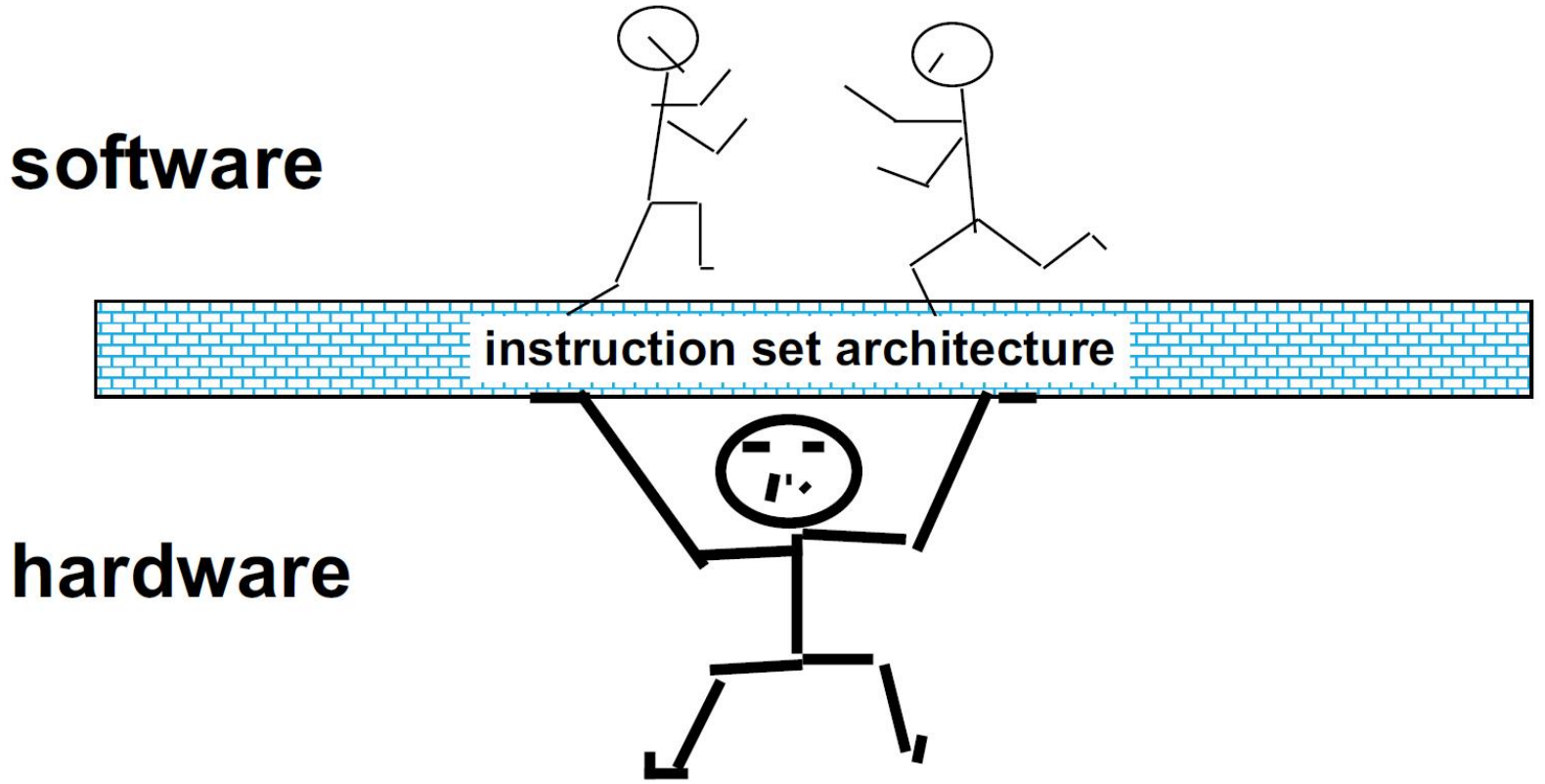
Output Device Outputs Data




```
00000100010100000000000000000000
00000000010011110000000000000100
00000011111000000000000000001000
```

The Instruction Set Architecture (ISA)

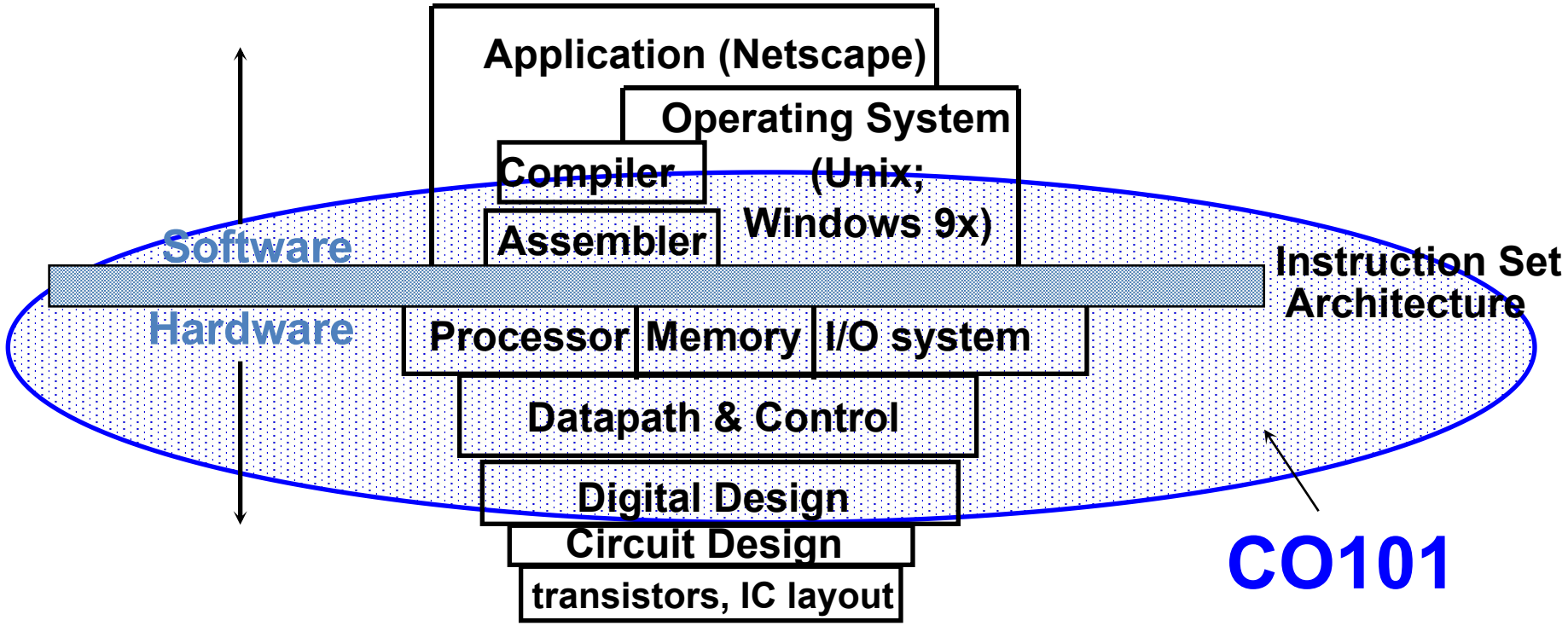
- The instruction set is a critical interface.
- The interface description is separating the software and hardware.



Instruction Set Architecture (ISA)

- ISA – the abstract interface between the hardware and the lowest level software that includes all the information necessary to write a machine language program, including instructions, registers, memory access, I/O, …
  - Enables implementations of varying cost and performance to run identical software.
- The combination of the basic instruction set (the ISA) and the operating system interface is called the application binary interface (ABI).
  - ABI – The user portion of the instruction set plus the operating system interfaces used by application programmers. Defines a standard for binary portability across computers.

How Do the Pieces Fit Together?



- Key Idea: *levels of abstraction*

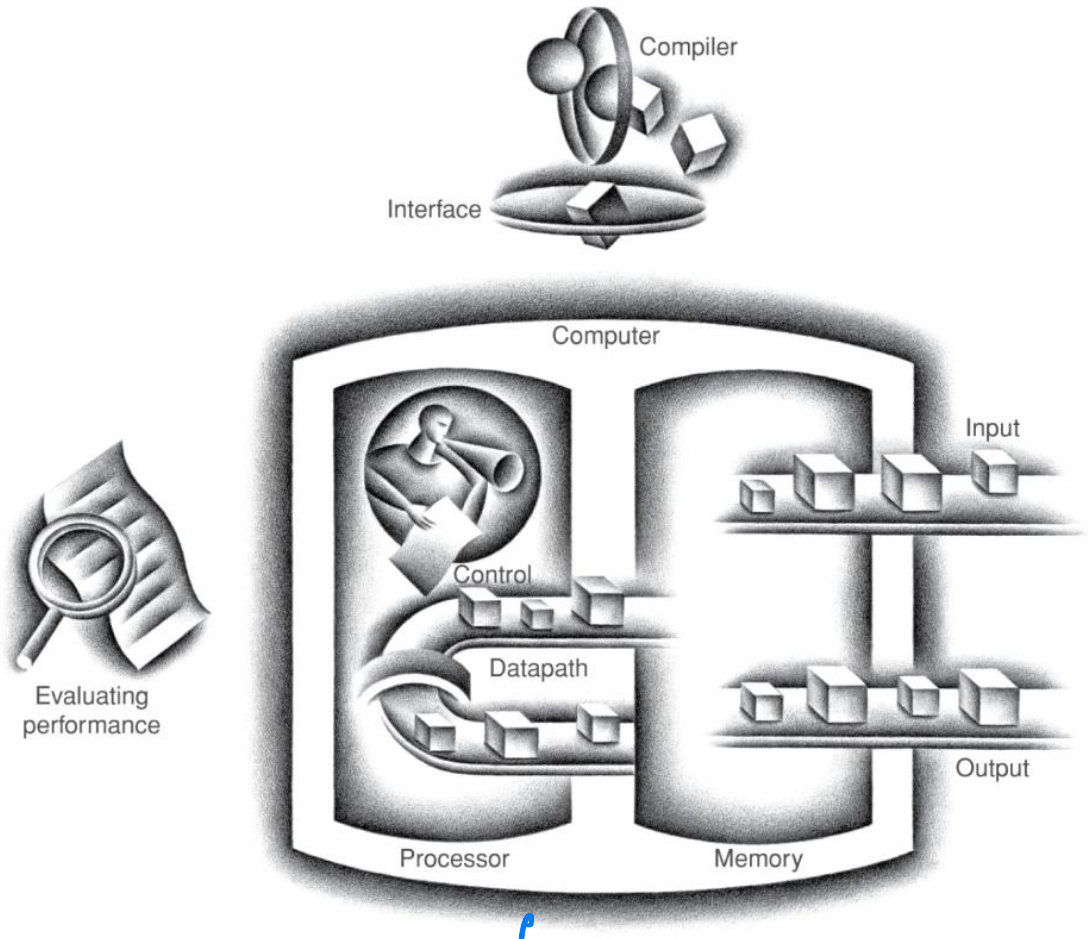
Abstractions

- Abstraction helps us deal with complexity of real systems, as it hides unnecessary lower-level implementation details.
  - Both hardware and software consist of hierarchical layers, with each lower layer hiding details from the level above.
- One key interface between the levels of abstraction is the *instruction set architecture* - the interface between the hardware and low-level software.
  - This abstract interface enables many *implementations* of varying cost and performance to run identical software.
- An instruction set architecture allows computer designers to talk about functions independently from the hardware that performs them.
  - Computer designers distinguish architecture from an *implementation* of an architecture along the same lines: an implementation is hardware that obeys the architecture abstraction.

How a CPU Works: https://www.youtube.com/watch?v=cNN_tTXABUA&t=517s

Understanding Performance

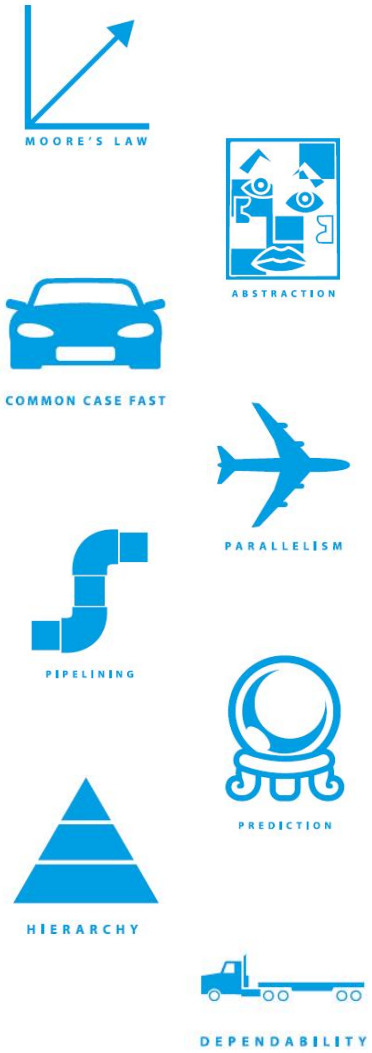
- Algorithm
  - Determines number of operations executed
- Programming language, compiler, architecture
  - Determine number of machine instructions executed per operation
- Processor and memory system
  - Determine how fast instructions are executed
- I/O system (including OS)
  - Determines how fast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



Separation of storage and computation

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



Measure Performance

- **Why We Need to Measure Performance?**
- Performance is an important attribute when choosing computers.
- Salespeople may show you the best light of a computer. However, is the "best light" accurately reflect the performance?
- Understanding how best to measure performance and the limitations of a particular performance measure is critical in choosing computers.
- This lecture → different metrics of performance measure.

Two notions of “performance”

Plane	DC to Paris	Speed (mph)	Passengers	Throughput (pmph)
Boeing 747	6.5 hours	610 mph	470	286,700
Concorde	3 hours	1350 mph	132	178,200

Which has higher performance?

pmph: passengers miles per hour

Example

- Time of Concorde vs. Boeing 747?
  - Concord is 1350 mph / 610 mph = 6.5 hours / 3 hours
    = 2.2 "times faster"

- pmph of Concorde vs. Boeing 747 ?
  - Boeing is 286,700 pmph / 178,200 pmph = 1.6 "times faster"

- Boeing is 1.6 times faster in terms of pmph
- Concord is 2.2 times faster in terms of flying time

- A problem: which plane is better?
  - Need a performance measure when we say someone's performance is better.

Computer Performance Metrics

- Purchasing perspective
  - given a collection of machines, which has the
    - best performance?
    - least cost?
    - best ratio of cost-performance?
- Design perspective
  - Faced with design options, which has the
    - best performance improvement?
    - least cost?
    - best ratio of cost-performance?
- Both require
  - basis for comparison
  - metric for evaluation
- Our goal is to understand what factors in the architecture contribute to overall system performance and the relative importance (and cost) of these factors

Computer Performance

- Execution time (response time)
  - The time between the start and completion of a task (program).
- Throughput
  - The total amount of work done in a given time.
  - The number of tasks finished in a time period.

- If we upgrade a machine with a faster processor, what do we increase?
- If we add a new machine to the lab, what do we increase?

Computer Performance

- Individual computer user: interested in execution time.
  - Care about how long to execute my job?
- Data center manager: interested in throughput
  - Care about how many tasks can be done in a time period?

- We will need different performance metrics as well as a different set of applications to benchmark embedded and desktop computers, which are more focused on execution time, versus servers, which are more focused on throughput.

Defining (Speed) Performance

- In discussing the (speed) performance of computers, we are primarily concerned with execution time. To maximize performance, need to minimize execution time.
- For some programs running on computer X,

$$\text{performance}_X = 1 / \text{execution\_time}_X$$

- Relative performance: If X is n times faster than Y, then

$$\text{performance}_X / \text{performance}_Y = n$$

Relative Performance Example

- If computer A runs a program in 10 seconds and computer B runs the same program in 15 seconds, how much faster is A than B?
- We know that A is n times faster than B if

$$\frac{\text{performance}_A}{\text{performance}_B} = \frac{\text{excution\_time}_B}{\text{excution\_time}_A} = n$$

- The performance ratio is 15/10 = 1.5.
- So A is 1.5 times faster than B.

Performance Factors

- CPU execution time (CPU time): time the CPU spends working on a task.
  - Does not include time waiting for I/O or running other programs.

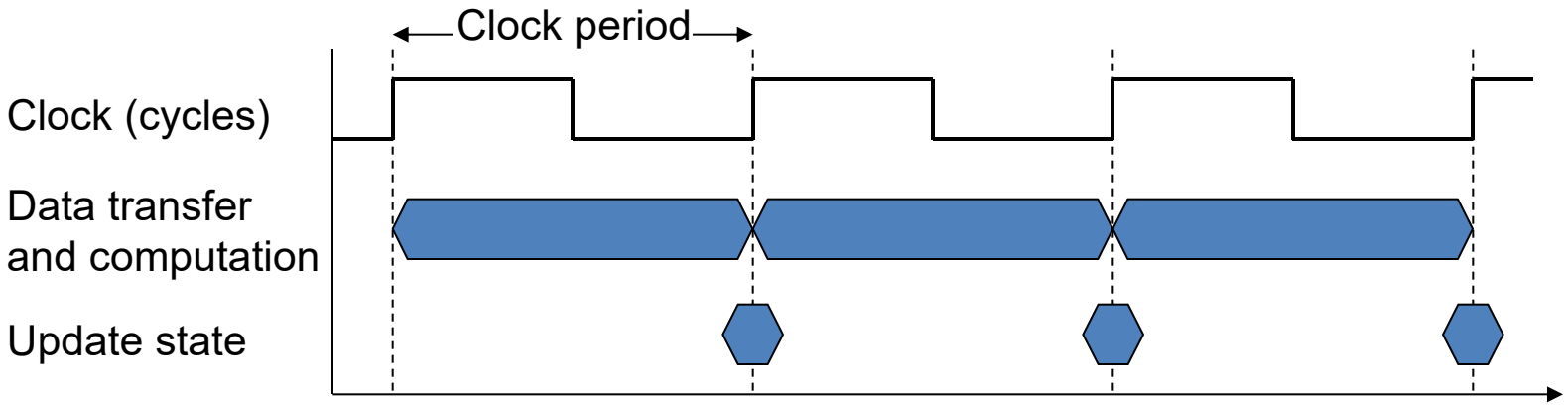
$$\text{CPU execution time for a program} = \#\text{ CPU clock cycles for a program} \times \text{clock cycle time}$$

$$= \frac{\#\text{CPU clock cycles for a program}}{\text{clock rate}}$$

- We can improve performance by
  - reducing the number of clock cycles required for a program;
  - reducing the clock cycle time or Increasing clock rate.

CPU Clocking

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



  - Clock period: duration of a clock cycle
    - e.g., 250ps = 0.25ns = 250×10^{-12}s
  - Clock frequency (rate): 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?

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

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

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

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

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

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

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
    = 2×1 + 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

The BIG Picture

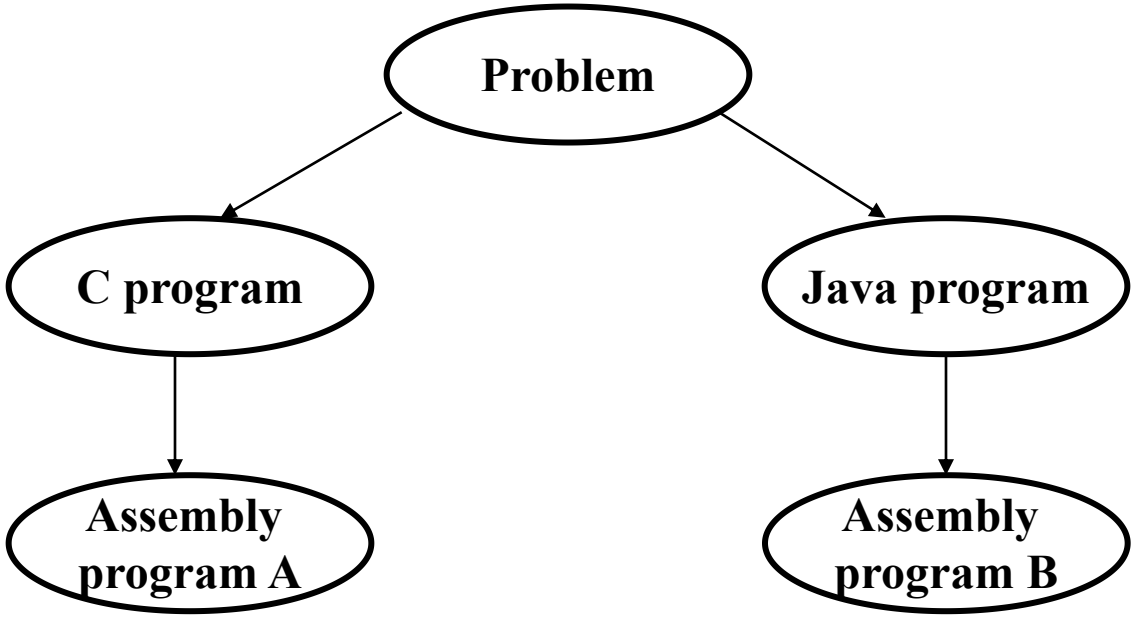
$$\text{CPU Time} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Clock cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Clock cycle}}$$

- 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

Determinates of CPU Performance

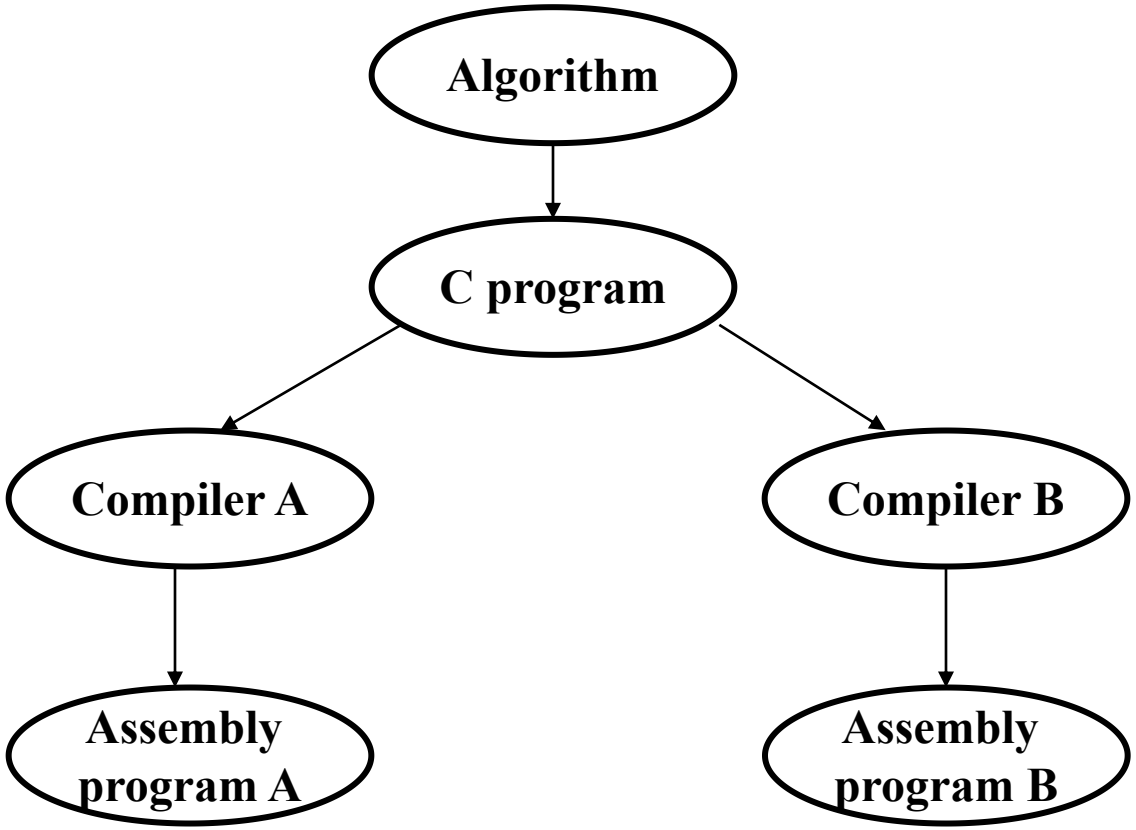
	Instruction count	CPI	Clock rate
Programming language	x	x	
Compiler	x	x	
ISA	x	x	x
Organization		x	x
Technology			x

Determinates of CPU Performance



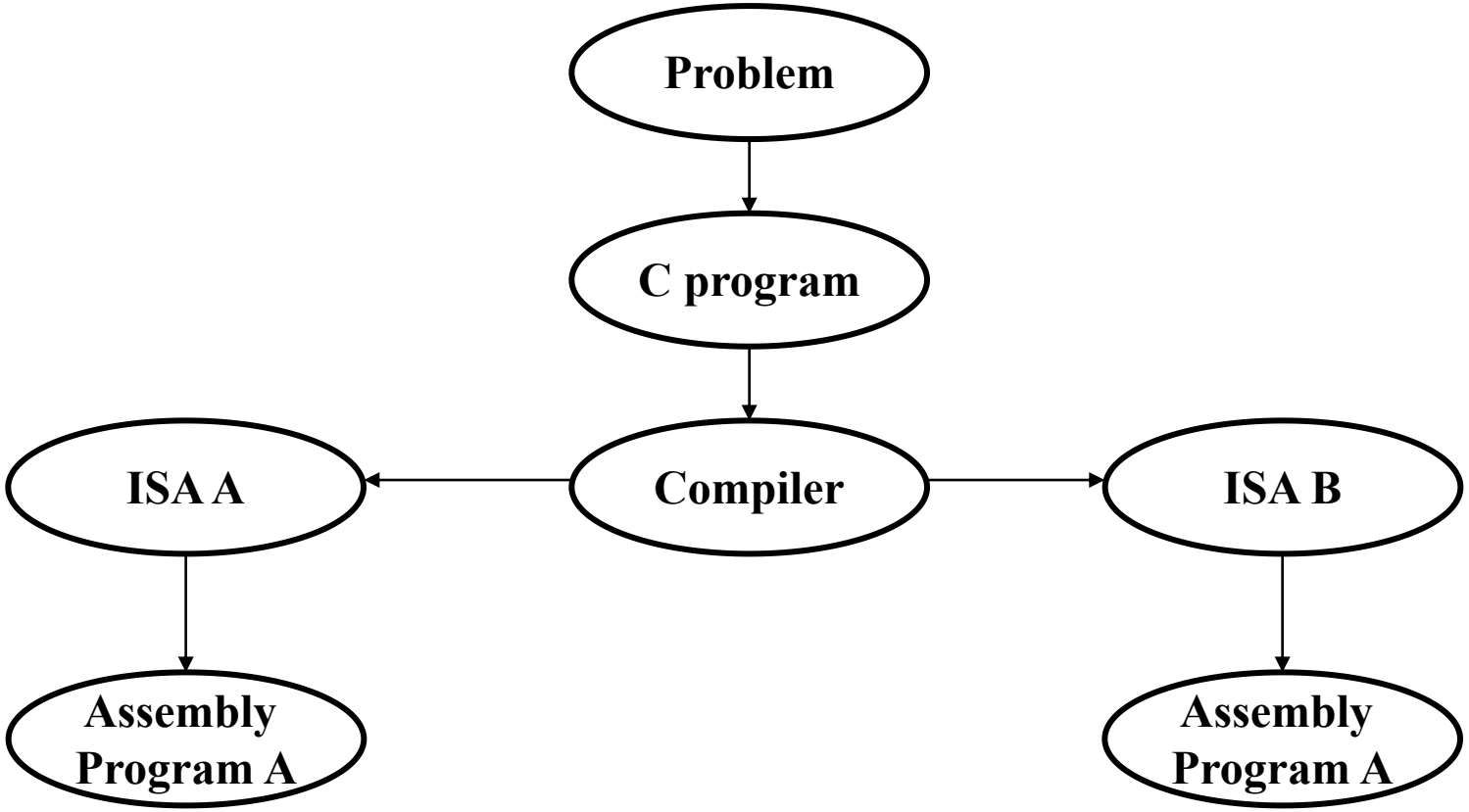
- Language : C vs Java
  - A same algorithm but written by different languages should have different programs, assemble programs and different machine instructions.
- So program language determines instruction count and CPI.

Determinates of CPU Performance



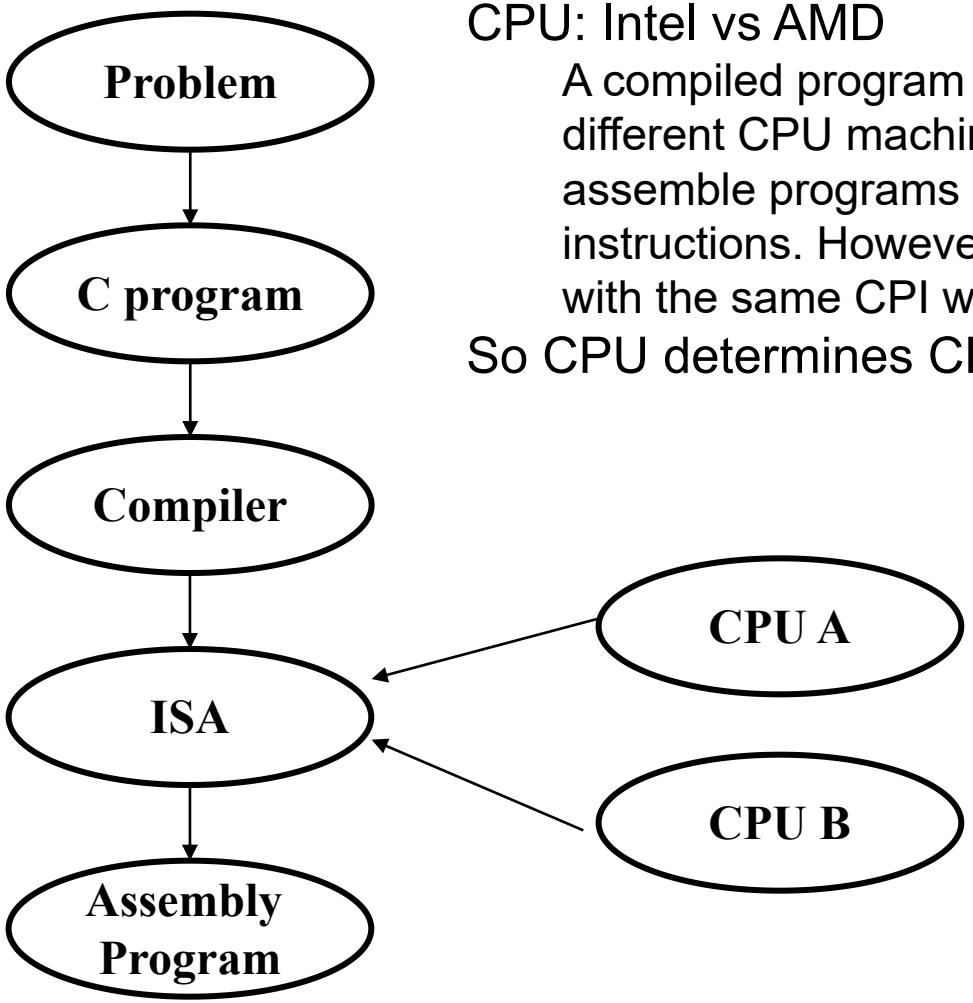
- Compiler: Visual studio C++ vs GCC
  - A same program but assembled by different compilers should have different assemble programs and different machine instructions.
- So compiler determines instruction count and CPI.

Determinates of CPU Performance



- ISA: Intel vs Mac
  - A program used a same compiler but running at different ISA machines should have different assemble programs and different machine instructions.
  - Different ISAs have different implementations (CPUs).
- So ISA determines instruction count, CPI and clock rate.

Determinates of CPU Performance

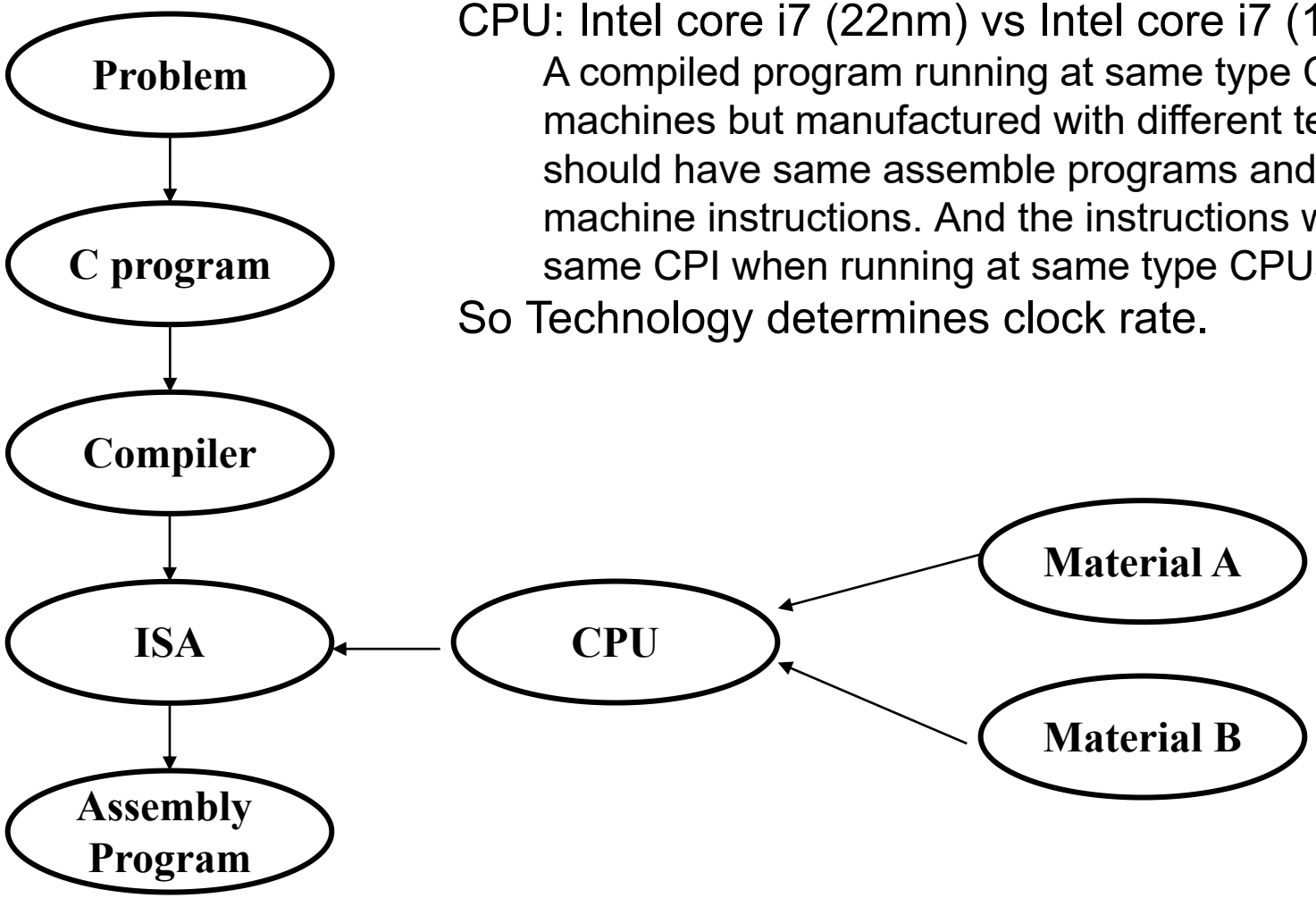


CPU: Intel vs AMD

A compiled program running at same ISA but different CPU machines should have same assemble programs and same machine instructions. However, the instructions won't be with the same CPI when running at different CPUs.

So CPU determines CPI and clock rate.

Determinates of CPU Performance



CPU: Intel core i7 (22nm) vs Intel core i7 (14nm)

A compiled program running at same type CPU machines but manufactured with different technologies should have same assemble programs and same machine instructions. And the instructions will be the same CPI when running at same type CPUs.

So Technology determines clock rate.

Summary

- For a given architecture performance increases come from:
  - use better material to increase clock rate (without increase CPI);
  - improvements in processor organization that reduce the CPI;
  - compiler enhancements that reduce the CPI and/or instruction count.

Amdahl's Law

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

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

- Example:

"Suppose a program runs in 100 seconds on a machine, with multiply responsible for 80 seconds of this time. How much do we have to improve the speed of multiplication if we want the program to run 4 times faster?“

Exe. Time affected = 80 seconds

Exe. Time unaffected = 100 – 80 = 20 seconds

4 times faster means 100/4=25 seconds, as a result:

25 = 80/Improvement + 20 → Improvement = 16 times

Workloads and Benchmarks

- Performance best determined by running a real application
  - Use programs typical of expected workload
  - Or, typical of expected class of applications

    e.g., compilers/editors, scientific applications, graphics, etc.
- Benchmarks – a set of programs that form a “workload” specifically chosen to measure performance.
- SPEC (System Performance Evaluation Cooperative)
  - Companies have agreed on a set of real program and inputs.
  - SPEC creates standard sets of benchmarks starting with SPEC89. The latest is SPEC CPU2006 which consists of 12 integer benchmarks (CINT2006) and 17 floating-point benchmarks (CFP2006 www.spec.org).
  - There are also benchmark collections for power workloads (SPECpower_ssj2008), for mail workloads (SPECmail2008), for multimedia workloads (mediabench), and so on.

SPEC Benchmarks

Integer benchmarks		FP benchmarks	
gzip	compression	wupwise	Quantum chromodynamics
vpr	FPGA place & route	swim	Shallow water model
gcc	GNU C compiler	mgrid	Multigrid solver in 3D fields
mcf	Combinatorial optimization	applu	Parabolic/elliptic pde
crafty	Chess program	mesa	3D graphics library
parser	Word processing program	galgel	Computational fluid dynamics
eon	Computer visualization	art	Image recognition (NN)
perlbmk	perl application	equake	Seismic wave propagation simulation
gap	Group theory interpreter	facerec	Facial image recognition
vortex	Object oriented database	ammp	Computational chemistry
bzip2	compression	lucas	Primality testing
twolf	Circuit place & route	fma3d	Crash simulation fem
		sixtrack	Nuclear physics accel
		apsi	Pollutant distribution

Comparing and Summarizing Performance

- How do we summarize the performance for benchmark set with a single number?
  - First the execution times are normalized given the “SPEC ratio” (bigger is faster, i.e., SPEC ratio is the in.verse of execution time)
  - The SPEC ratios are then “averaged” using the geometric mean (GM).

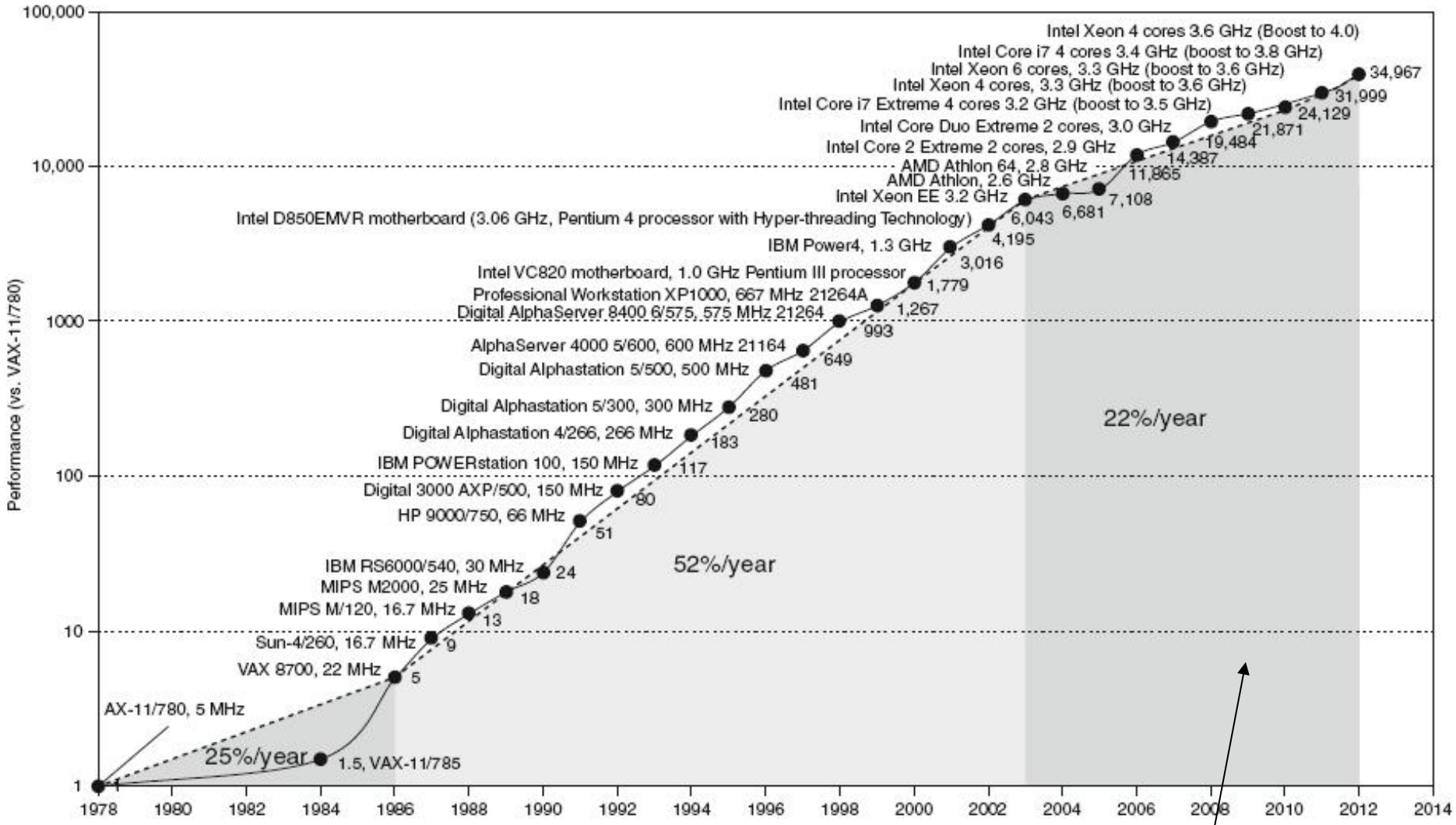
$$GM = \sqrt[n]{\prod_{i=1}^{n} \text{SPEC ratio}_i}$$

- Guiding principle in reporting performance measurements is reproducibility – list everything another experimenter would need to duplicate the experiment (version of the operating system, compiler settings, input set used, specific computer configuration (clock rate, cache sizes and speed, memory size and speed, etc.))

SPEC CINT2006 on AMD Barcelona

Name	ICx10^9	CPI	ExTime	RefTime	SPEC ratio
perl	2,1118	0.75	637	9,770	15.3
bzip2	2,389	0.85	817	9,650	11.8
gcc	1,050	1.72	724	8,050	11.1
mcf	336	10.00	1,345	9,120	6.8
go	1,658	1.09	721	10,490	14.6
hmmer	2,783	0.80	890	9,330	10.5
sjeng	2,176	0.96	837	12,100	14.5
libquantum	1,623	1.61	1,047	20,720	19.8
h264avc	3,102	0.80	993	22,130	22.3
omnetpp	587	2.94	690	6,250	9.1
astar	1,082	1.79	773	7,020	9.1
xalancbmk	1,058	2.70	1,143	6,900	6.0
Geometric Mean					11.7

Uniprocessor Performance



Constrained by power, instruction-level parallelism, memory latency