LECTURE 1

Introduction

CLASSES OF COMPUTERS

- A computer is a device that can be instructed to carry out arbitrary sequences of arithmetic or logical operations automatically.

- Computers share a core set of technologies, the implementation and use of these technologies varies with the chosen application.

- In general, there are three classes of applications to consider: *desktop computers*, *servers*, and *embedded computers*.

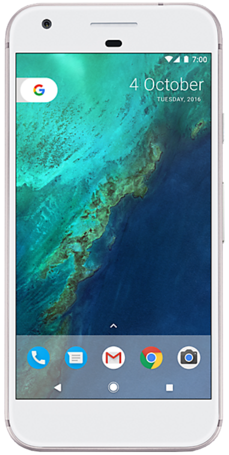
CLASSES OF COMPUTERS

- Desktop Computers (or Personal Computers)
  - Emphasize good performance for a single user at relatively low cost.
  - Mostly execute third-party software.

- Servers
  - Emphasize great performance for a few complex applications.
  - Or emphasize reliable performance for many users at once.
  - Greater computing, storage, or network capacity than personal computers.
- Embedded Computers
  - Largest class and most diverse.
  - Usually specifically manufactured to run a single application reliably.
  - Stringent limitations on cost and power.

PERSONAL MOBILE DEVICES

- PMDs, including small general-purpose devices such as tablets and smartphones
  - generally have the same design requirements as PCs
  - much more stringent efficiency requirements (to preserve battery life and reduce heat emission).
- PMDs are just as powerful as desktop computers but with more sensitive data
- What is the most widely used operating system? Windows, Linux, or MacOS?



GREAT ARCHITECTURE IDEAS

There are 8 great architectural ideas that have been applied in the design of computers for over half a century now.

As we cover the material of this course, we should stop to think every now and then which ideas are in play and how they are being applied in the current context.

GREAT ARCHITECTURE IDEAS

- Design for Moore's law.
  - The number of transistors on a chip doubles every 18-24 months.
  - Architects have to anticipate where technology will be when the design of a system is completed.
- Use abstraction to simplify design.
  - Abstraction is used to represent the design at different levels of representation.
  - Lower-level details can be hidden to provide simpler models at higher levels.
- Make the common case fast.
  - Identify the common case and try to improve it.
  - Most cost efficient method to obtain improvements.
- Improve performance via parallelism.
  - Improve performance by performing operations in parallel.
  - There are many levels of parallelism – instruction-level, process-level, etc.

GREAT ARCHITECTURE IDEAS

- Improve performance via pipelining.
  - Break tasks into stages so that multiple tasks can be simultaneously performed in different stages.
  - Commonly used to improve instruction throughput.
- Improve performance via prediction.
  - Sometime faster to assume a particular result than waiting until the result is known.
  - Known as speculation and is used to guess results of branches.
- Use a hierarchy of memories.
  - Make the fastest, smallest, and most expensive per bit memory the first level accessed and the slowest, largest, and cheapest per bit memory the last level accessed.
  - Allows most of the accesses to be caught at the first level and be able to retain most of the information at the last level.
- Improve dependability via redundancy.
  - Include redundant components that can both detect and often correct failures.
  - Used at many different levels.

WHY LEARN COMPUTER ORGANIZATION?

Improving a program's performance is not as simple as reducing its memory usage. Modern programmers need to have an understanding of the issues "below the program":

- The parallel nature of processors.
  - How might you speed up your application by introducing parallelism via threading or multiprocessing?
  - How will the compiler translate and rearrange your own instruction-level code to perform instructions in parallel?
- The hierarchical nature of memory.
  - How can you rearrange your memory access patterns to more efficiently read data?
  - Ever heard of *page (cache) coloring*, *false sharing*, *side-channel attacks*?
- The translation of high-level languages into hardware instructions
  - What decisions are made by the compiler in the process of generating instruction-level statements?

PROGRAM LEVELS AND TRANSLATION

- The computer actually speaks in terms of electrical signals.
  - > 1V is “on” and 0V is “off”.
- We can represent each signal as a binary digit, or bit.
  - 1 is “on” and 0 is “off”.
- The instructions understood by a computer are simply significant collections of bits.
- Data is also represented as significant collections of bits.

PROGRAM LEVELS AND TRANSLATION

The various levels of representation for a program are:

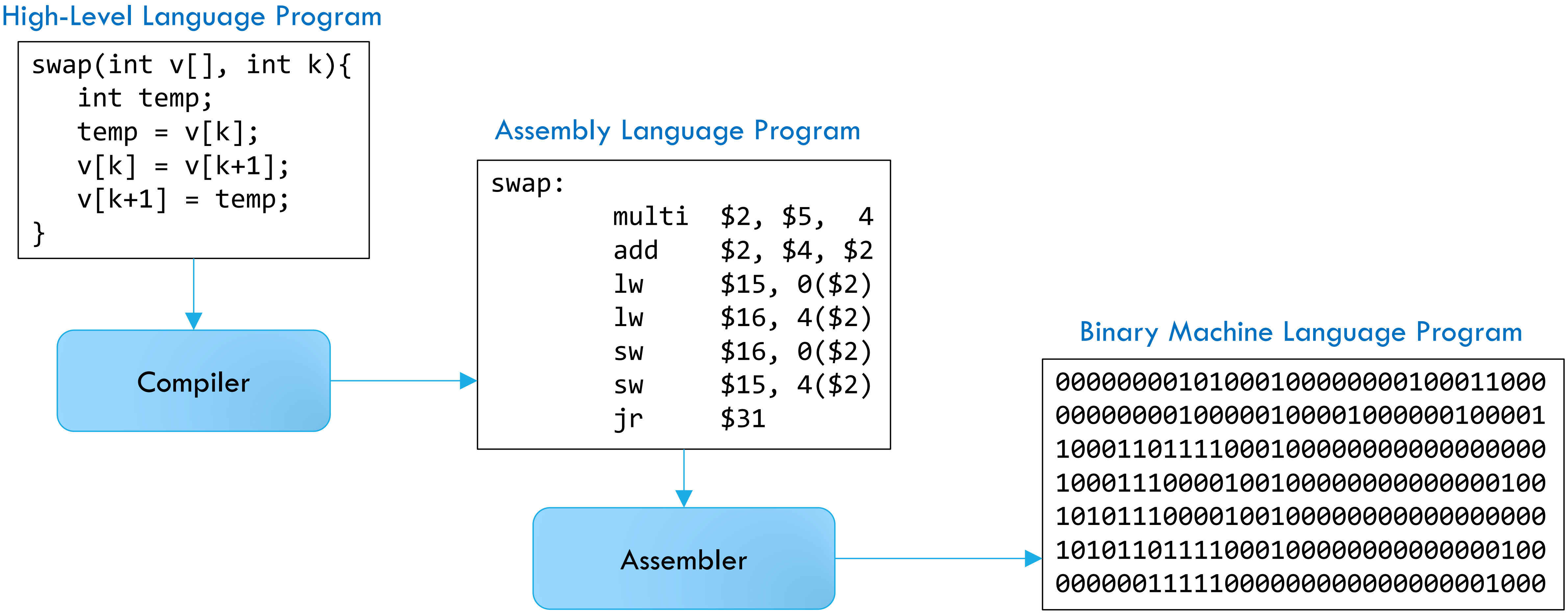
- High-level language: human-readable code at which programmers develop applications.
- Assembly language: symbolic representation of instructions.
- Machine language: binary representation of instructions, understandable by the computer and executable by the processor.

PROGRAM LEVELS AND TRANSLATION

The stages of translation between these program levels are implemented by the following:

- Compiler: high-level language →assembly language.
- Assembler: assembly language → machine language.
- Linker: combines multiple machine language files into a single executable that can be loaded into memory and executed.

EXAMPLE OF TRANSLATING A C PROGRAM



High-Level Language Program

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

Compiler

Assembly Language Program

```
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 Program

```
00000000101000100000000100011000
00000000100000100001000000100001
10001101111000100000000000000000
10001110000100100000000000000100
10101110000100100000000000000000
10101101111000100000000000000100
00000011111000000000000000001000
```

BENEFITS OF ABSTRACTION

There are several important benefits to the layers of abstraction created by the high-level programming language to machine language translation steps.

- Allows programmers to think in more natural terms
  - Using English words and algebraic notation.
  - Languages can also be tailor-made for a particular domain.
- Improved programmer productivity. Conciseness is key.
  - Java is a verbose language, C++ is too complicated, …
- The most important advantage is portability.
  - Programs are independent of the machine because compilers and assemblers can take a universal program and translate it for a particular machine.
  - Java is a portable language (Java VM)

PERFORMANCE

Being able to gauge the relative performance of a computer is an important but tricky task. There are a lot of factors that can affect performance.

- Architecture
- Hardware implementation of the architecture
- Compiler for the architecture
- Operating system

Furthermore, we need to be able to define a measure of performance.

- Single users on a PC → a minimization of response time.
- Large data → a maximization of throughput

PERFORMANCE

To discuss performance, we need to be familiar with two terms:

- *Latency* (response time) is the time between the start and completion of an event.
- *Throughput* (bandwidth) is the total amount of work done in a given period of time.

In discussing the performance of computers, we will be primarily concerned with program latency.

PERFORMANCE

Do the following changes to a computer system increase throughput, decrease latency, or both?

- Replacing the processor in a computer with a faster processor.
- Adding additional processors to a system that uses processors for separate tasks.

PERFORMANCE

Answers to previous slide:

- Throughput increases and latency decreases (i.e. both improve).
- Only throughput increases.

PERFORMANCE

Performance has an inverse relationship to execution time.

$$Performance = \frac{1}{Execution\ Time}$$

Comparing the performance of two machines can be accomplished by comparing execution times.

$$Performance_X > \ Performance_Y$$

$$\longrightarrow \quad \frac{1}{Execution_X} > \frac{1}{Execution_Y}$$

$$\longrightarrow \quad Execution_Y > Execution_X$$

RELATIVE PERFORMANCE

Often people state that a machine *X* is *n* times faster than a machine *Y*. What does this mean?

$$\frac{Performance_X}{Performance_Y} = \frac{Execution_Y}{Execution_X} = n$$

If machine *X* takes 20 seconds to perform a task and machine *Y* takes 2 minutes to perform the same task, then machine *X* is how many times faster than machine *Y*?

RELATIVE PERFORMANCE

Answer to previous slide: Machine X is 6 times faster than Machine Y.

Computer C's performance is 4 times faster than the performance of computer B, which runs a given application in 28 seconds. How long will computer C take to run the application?

RELATIVE PERFORMANCE

Answer to previous slide: 7 seconds.

MEASURING PERFORMANCE

There are several ways to measure the execution time on a machine.

- Elapsed time: total wall clock time needed to complete a task (including I/O, etc).
- CPU time: time CPU spends actually working on behalf of the program. This does not include waiting for I/O or other running programs.
- User CPU time: CPU time spent in the program itself.
- System CPU time: CPU time spent in the OS, performing tasks on behalf of the program.

MEASURING PERFORMANCE

Sometimes, it is more useful to think about performance in metrics other than time. In particular, it is common to discuss performance in terms of how fast a computer can perform basic functions.

- Clock cycle: the basic discrete time intervals of a processor clock, which runs at a constant rate (mostly).
- Clock period: the length of each clock cycle.
- Clock rate: inverse of the clock period.

MEASURING PERFORMANCE

Some common prefixes for clock period and clock rate:

clock periods

- millisecond (ms) - 10^{-3} s
- microsecond (μs) - 10^{-6} s
- nanosecond (ns) - 10^{-9} s
- picosecond (ps) - 10^{-12} s
- femtosecond (fs) - 10^{-15} s

clock rates

- kilohertz (KHz) - 10^{3} cycles per second
- megahertz (MHz) - 10^{6} cycles per second
- gigahertz (GHz) - 10^{9} cycles per second
- terahertz (THz) - 10^{12} cycles per second
- petahertz (PHz) - 10^{15} cycles per second

MEASURING DATA SIZE

- bit - Binary digit
- nibble - four bits
- byte - eight bits
- word - four bytes (32 bits) on many embedded/mobile processors and eight bytes (64 bits) on many desktops and servers.
- kibibyte (KiB) [kilobyte (KB)] - 2^{10} (1,024) bytes
- mebibyte (MiB) [megabyte (MB)] - 2^{20} (1,048,576) bytes
- gibibyte (GiB) [gigabyte (GB)] - 2^{30} (1,073,741,824) bytes
- tebibyte (TiB) [terabyte (TB)] - 2^{40} (1,099,511,627,776) bytes
- pebibyte (PiB) [petabyte (PB)] - 2^{50} (1,125,899,906,842,624) bytes

CPU PERFORMANCE

In order to determine the effect of a design change on the performance experienced by the user, we can use the following relation:

$$CPU\ Execution\ Time = CPU\ Clock\ Cycles \times Clock\ Period$$

Alternatively,

$$CPU\ Execution\ Time = \frac{CPU\ Clock\ Cycles}{Clock\ Rate}$$

Clearly, we can reduce the execution time of a program by either reducing the number of clock cycles required or the length of each clock cycle.

CPU PERFORMANCE

Our favorite program runs in 10 seconds on computer A, which has a 2 GHz clock. We are trying to help a computer designer build computer B, which will run this program in 6 seconds. The designer has determined that a substantial increase in the clock rate is possible, but it will affect the rest of the CPU design, causing computer B to require 1.2 times as many clock cycles as computer A for this program. What clock rate should we tell the designer to target?

CPU PERFORMANCE

Answer to previous slide: To run the program in 6 seconds, B must have twice the clock rate of A.

CPU PERFORMANCE

Another way to think about program execution time is in terms of instruction performance. Generally, execution time is equal to the number of instructions executed multiplied by the average time per instruction.

$$CPU\ Clock\ Cycles = Instructions\ for\ a\ Program \times Average\ Clock\ Cycles\ Per\ Instruction$$

The average number of clock cycles per instruction is often abbreviated as CPI. The above equation can be rearranged to give the following:

$$CPI = \frac{CPU\ Clock\ Cycles}{Instruction\ Count}$$

CPU PERFORMANCE

Suppose we have two implementations of the same instruction set architecture. Computer A has a clock cycle time of 250 ps and a CPI of 2.0 for some program, and computer B has a clock cycle time of 500 ps and a CPI of 1.2 for the same program. Which computer is faster for this program and by how much?

CPU PERFORMANCE

Answer to previous slide: Computer A is 1.2 times as fast as Computer B.

CLASSIC CPU PERFORMANCE EQUATION

We can now write the basic equation in terms of instruction count, CPI, and clock cycle time.

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

Alternatively,

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

COMPONENTS OF PERFORMANCE

The basic components of performance and how each is measured.

Component	**Units of Measure**
CPU Execution Time for a Program	Seconds for the Program
Instruction Count	Instructions Executed for the Program
Clock Cycles per Instruction	Average Number of Clock Cycles per Instruction
Clock Cycle Time (Clock Period)	Seconds per Clock Cycle

Instruction Count, CPI, and Clock Period combine to form the three important components for determining CPU execution time. *Just analyzing one is not enough!* Performance between two machines can be determined by examining non-identical components.

AMDAHL'S LAW

Amdahl's Law states that the performance improvement to be gained from using some faster mode of execution is limited by the fraction of the time the faster mode can be used.

Amdahl's Law depends on two factors:

- The fraction of the time the enhancement can be exploited.
- The improvement gained by the enhancement while it is exploited.

$$Improved\ Execution\ Time = \frac{Affected\ Execution\ Time}{Amount\ of\ Improvement} + Unaffected\ Execution\ Time$$

Make the common case fast!

AMDAHL'S LAW

If the speed of arithmetic operations is improved by a factor of 5 and these operations constitute 40% of a program's old execution time, then what is the overall speedup?

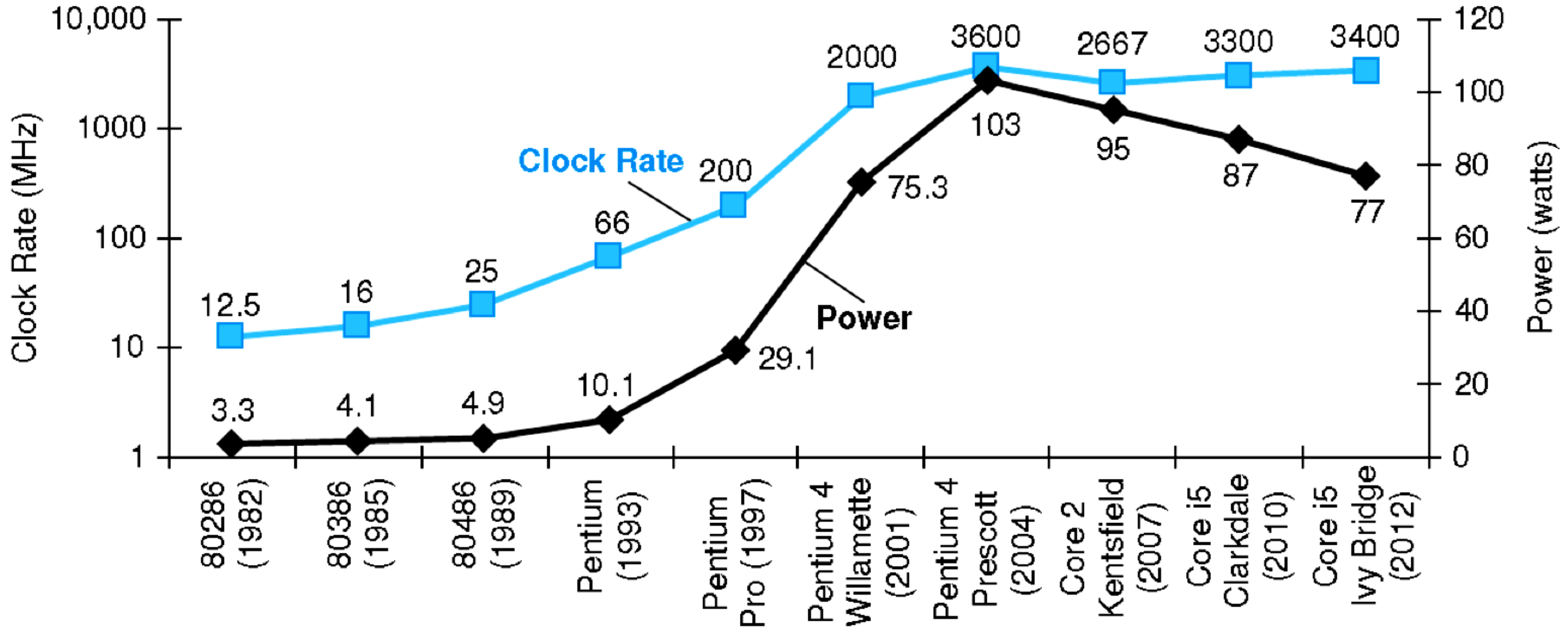
AMDAHL'S LAW

If the speed of arithmetic operations is improved by a factor of 5 and these operations constitute 40% of a program's old execution time, then what is the overall speedup?

Answer: improved execution time is 1.47 times faster.

ENERGY EFFICIENT PROCESSORS

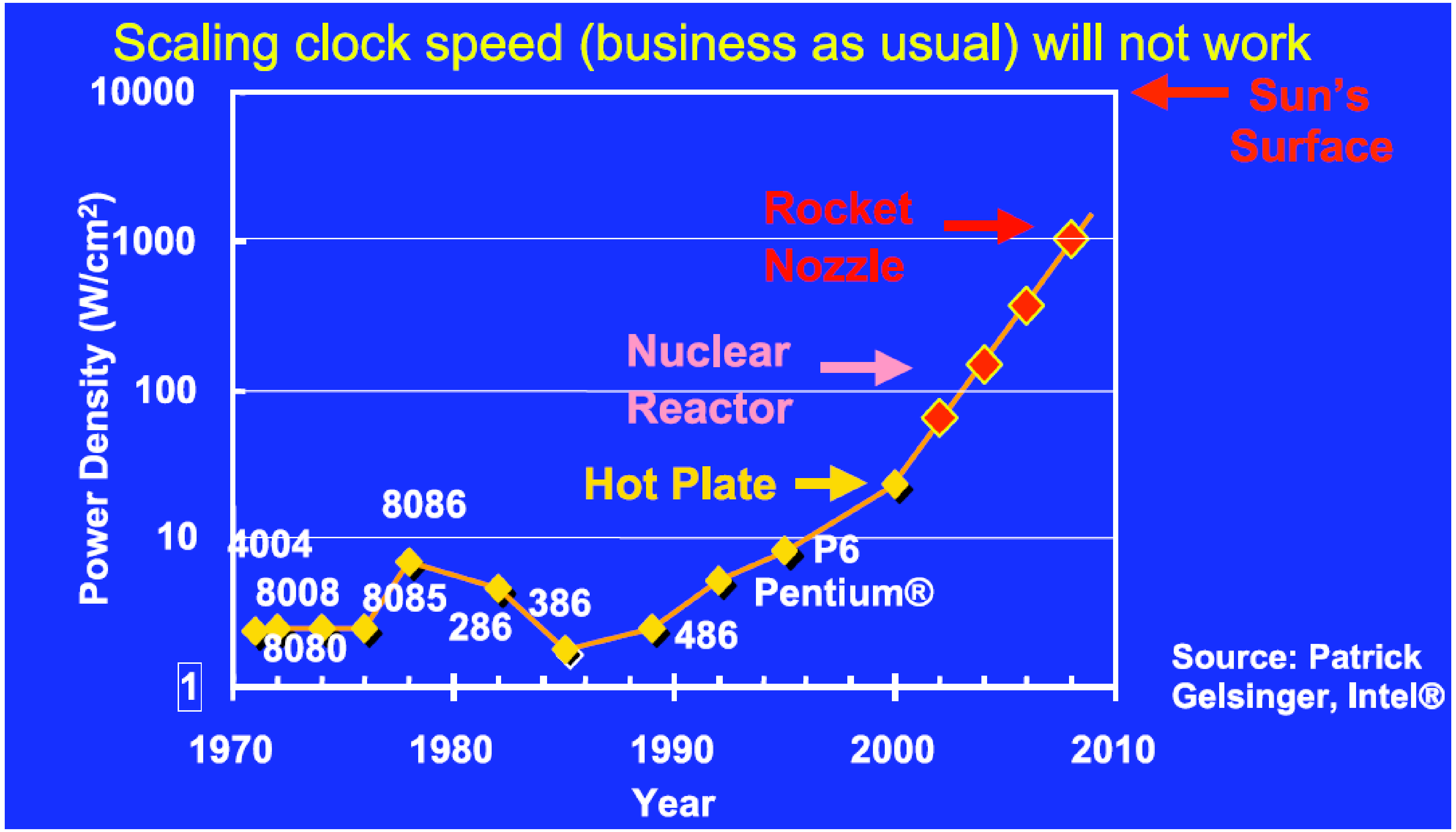
- Extend battery life for mobile systems.
- Reduce heat dissipation for general-purpose processors.
- Energy cost for computing is increasing, especially for server farms.



THE POWER WALL

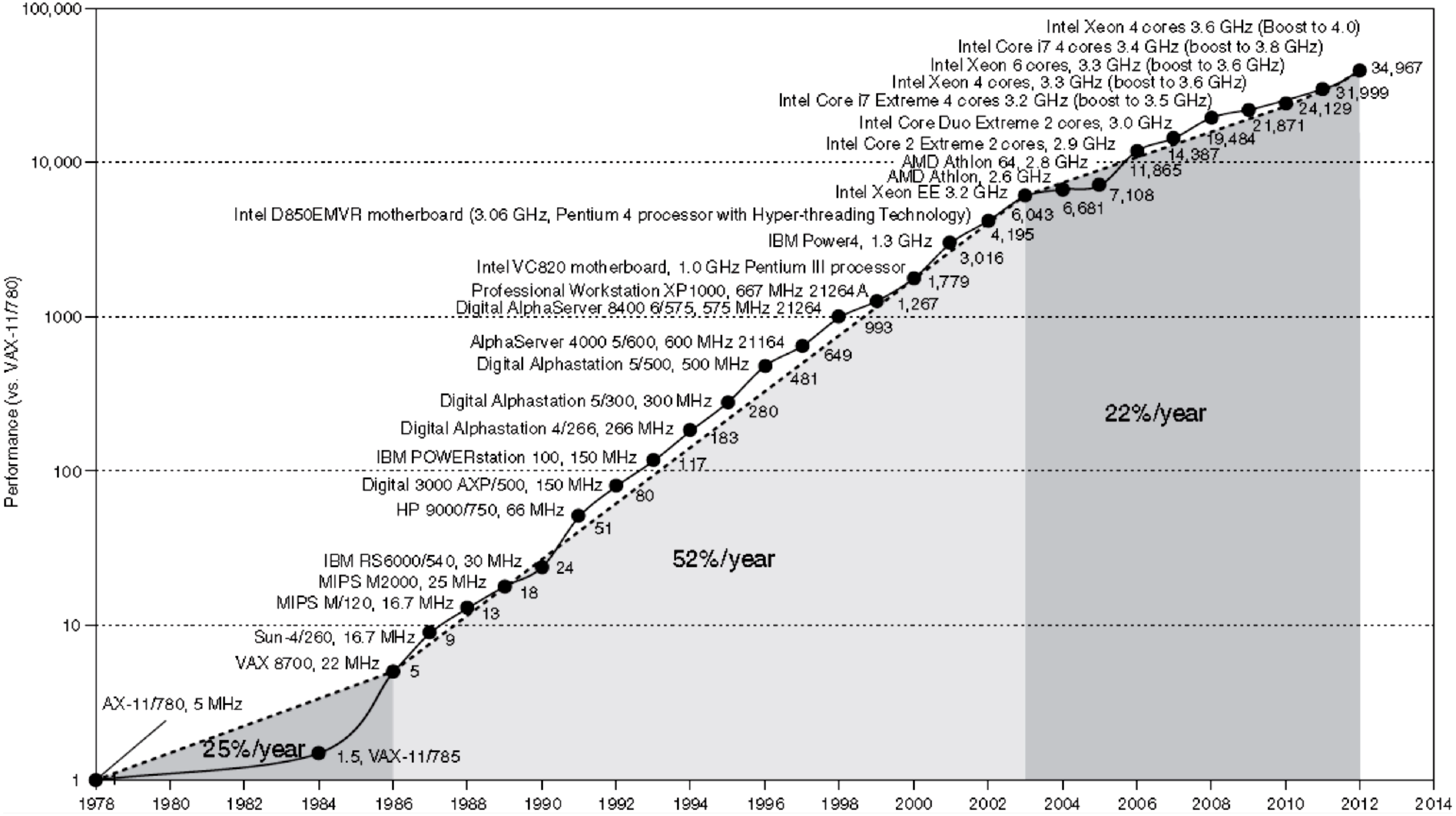
The previous graph has shown that although clock rate and power have increased dramatically over the past few decades, they have flattened recently.

The power wall refers to the issue that clock rates are not able to increase further due to thermal constraints.


Scaling clock speed (business as usual) will not work

Source: Patrick Gelsinger, Intel®

THE POWER WALL



THE POWER WALL

- **Dennard scaling:** as transistors get smaller their power density stays constant
  - the power use stays in proportion with area.
  - Historically, Dennard scaling allows CPU manufactures to drastically raise clock frequency without significantly increasing circuit power consumption.

- Around 2006, Dennard scaling breaks down as current leakage and thermal runaway worsen.
  - Megahertz war → core war
  - AMD Threadripper 1950x TDP 180 Watt

TRENDS IN IMPLEMENTATION TECHNOLOGY

- Transistor count on a chip is increasing by about 40% to 55% a year, or doubling every 18 to 24 months (Moore's law).
- DRAM capacity per chip is increasing by about 25% to 40% a year, doubling every two to three years.
- Flash capacity per chip is increasing by about 50% to 60% a year, doubling recently about every 1.5 years. Flash memory is 15 to 20 times cheaper per byte than DRAM.
- Disk density is increasing about 40% per year, doubling every two years. Disks per byte are 15 to 25 times cheaper than flash.

TRENDS IN IMPLEMENTATION TECHNOLOGY

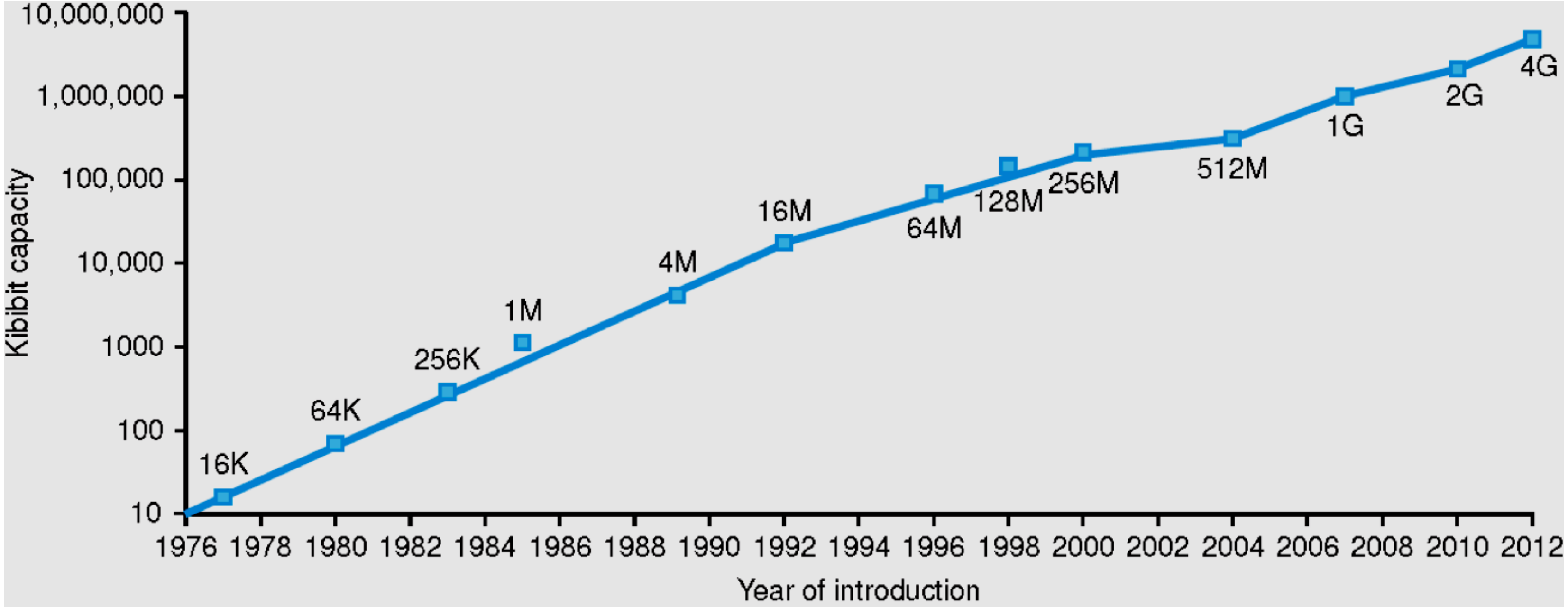
- Increasing the number of transistors per chip has benefits.
  - Reduces chip manufacturing cost since less material is being used and it improves yield as die sizes decrease.
  - Improves performance since there is less distance for electricity to travel, which means the rate of executing machine instructions can increase.

Year	**Technology**	**Relative Performance/Unit Cost**
1951	Vacuum Tube	1
1965	Transistor	35
1975	Integrated Circuit	900
1995	VLS Integrated Circuit	2,400,000
2013	ULS Integrated Circuit	250,000,000,000

TRENDS IN IMPLEMENTATION TECHNOLOGY

DRAM chips are also made of transistors.

Increasing the number of transistors on a DRAM chip directly improves DRAM capacity as shown in the figure below.



EFFECTS OF DRAMATIC GROWTH

- Enhanced capability available to users.
- Led to new classes of computers.
- Led to dominance of microprocessor based computers.
- Allows programmers to trade performance for productivity.
- Nature of applications are also changing.

- Sometimes might reduce reliability, e.g., in HDD (hard disk)