Computer Architecture

CS F342

Introduction

- Rapidly changing field:
  - vacuum tube -> transistor -> IC -> VLSI
  - doubling every 1.5 years:
    - memory capacity
    - processor speed (due to advances in technology <u>and</u> hardware organization)

Moore's Law

IBM POWER5 has 276 million transistors

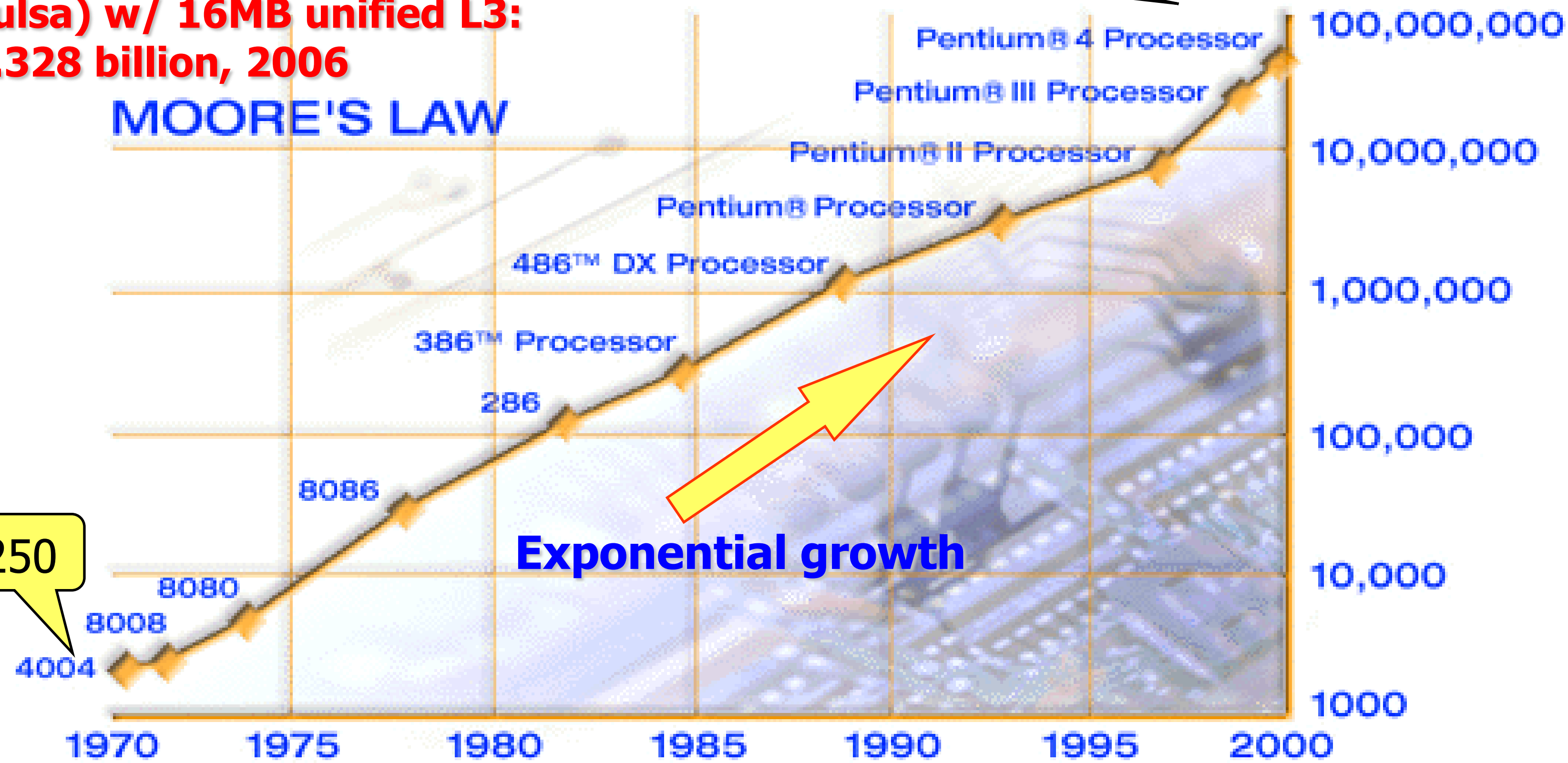
Intel Dual-Core Xeon (P4-based Tulsa) w/ 16MB unified L3: 1.328 billion, 2006

P4 Extreme Ed.
178 millions w/ 2MB L3

Core 2 Duo (Conroe)
291 millions, July 2006

42 millions

transistors

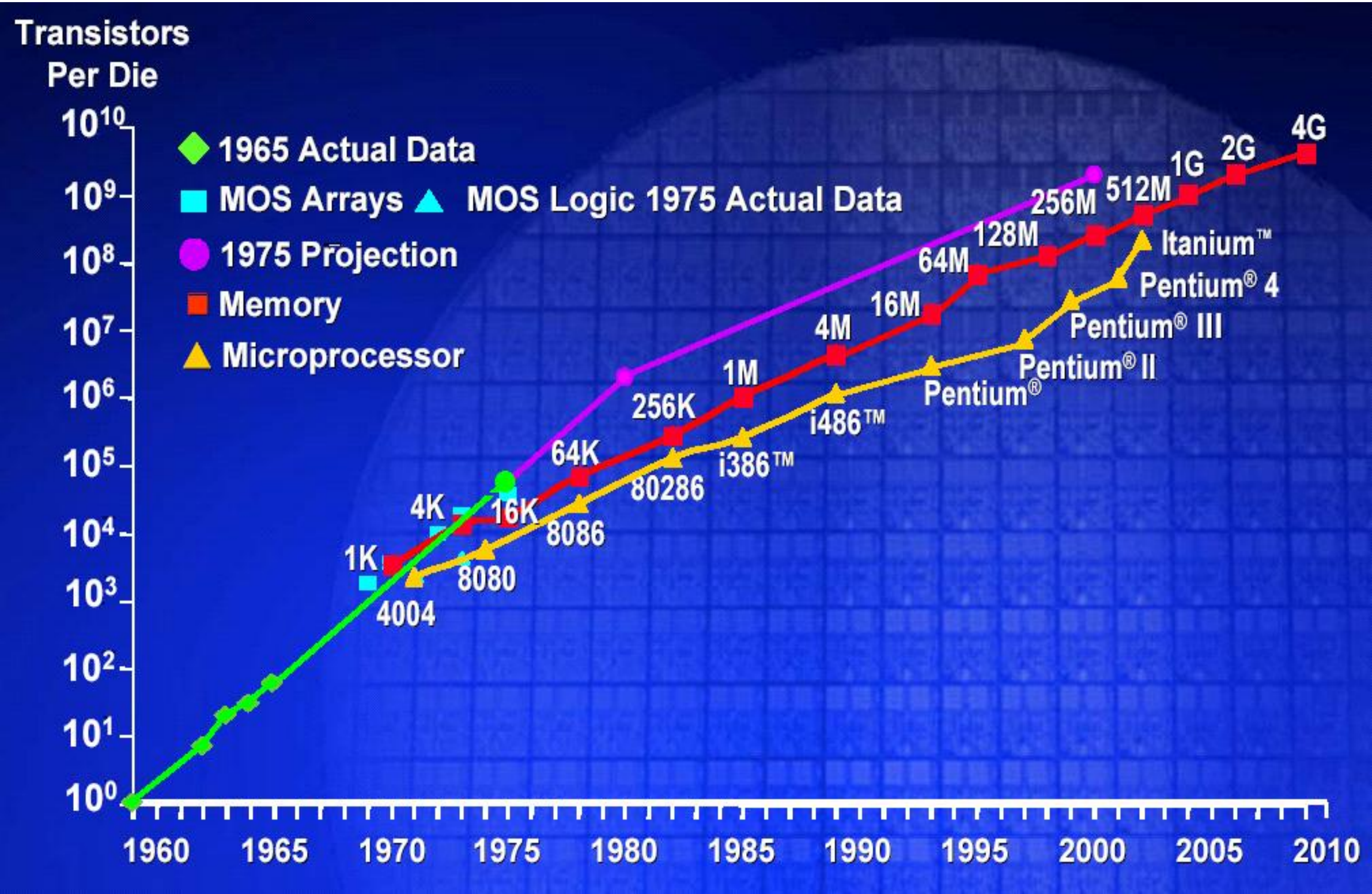


2,250

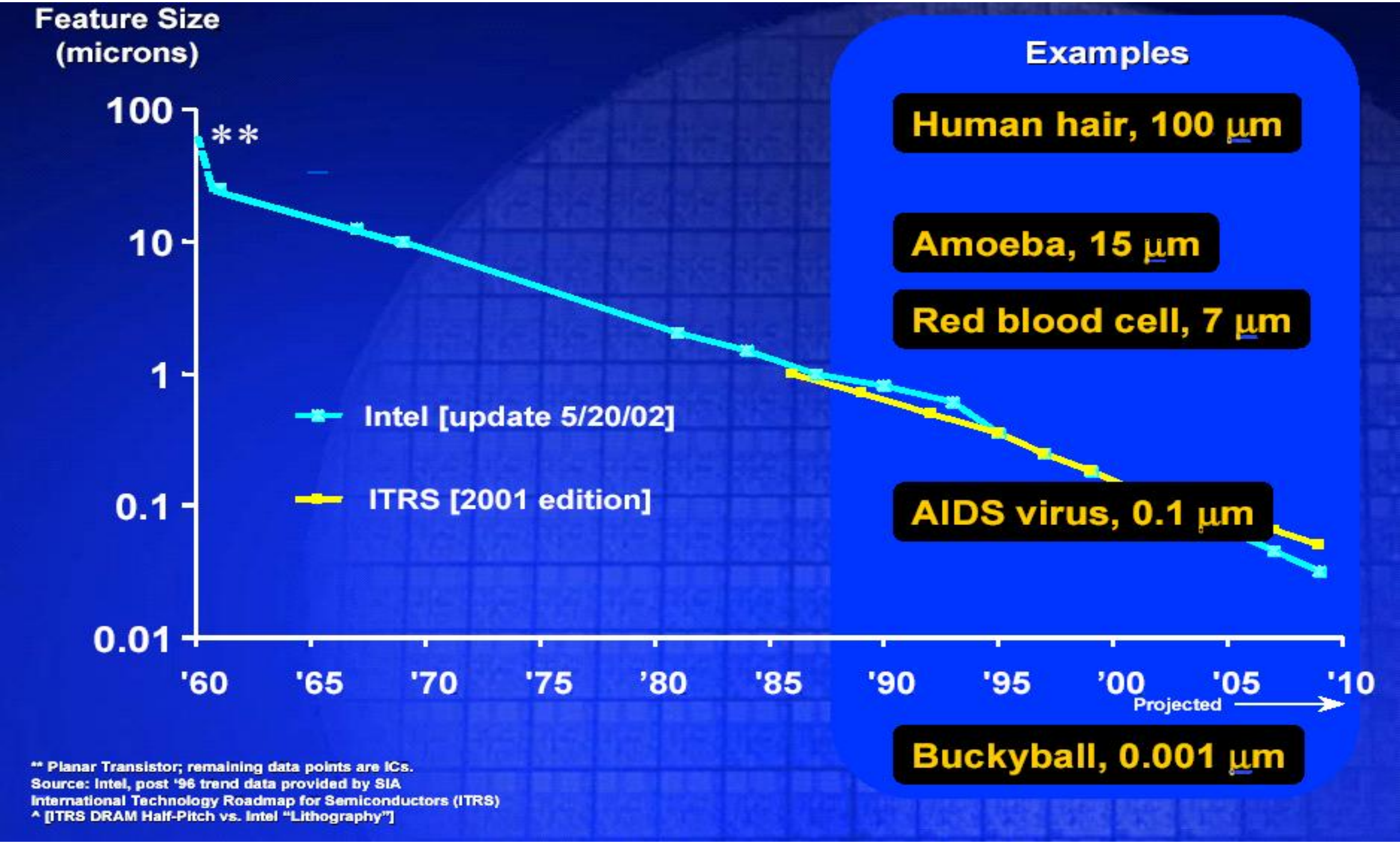
Transistor count will be doubled every 18 months

— Gordon Moore, Intel co-founder

Integrated Circuits Capacity

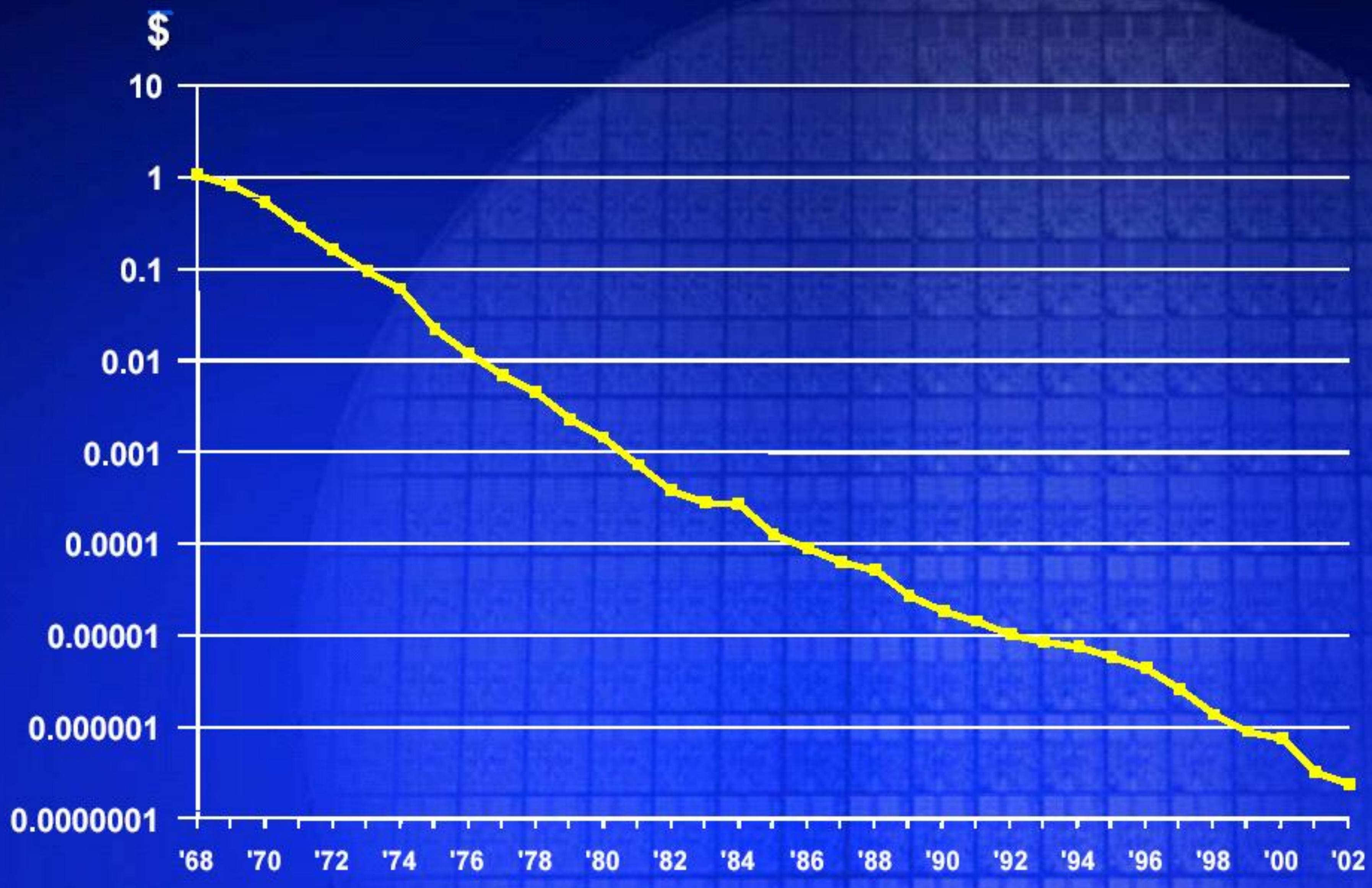


Feature Size

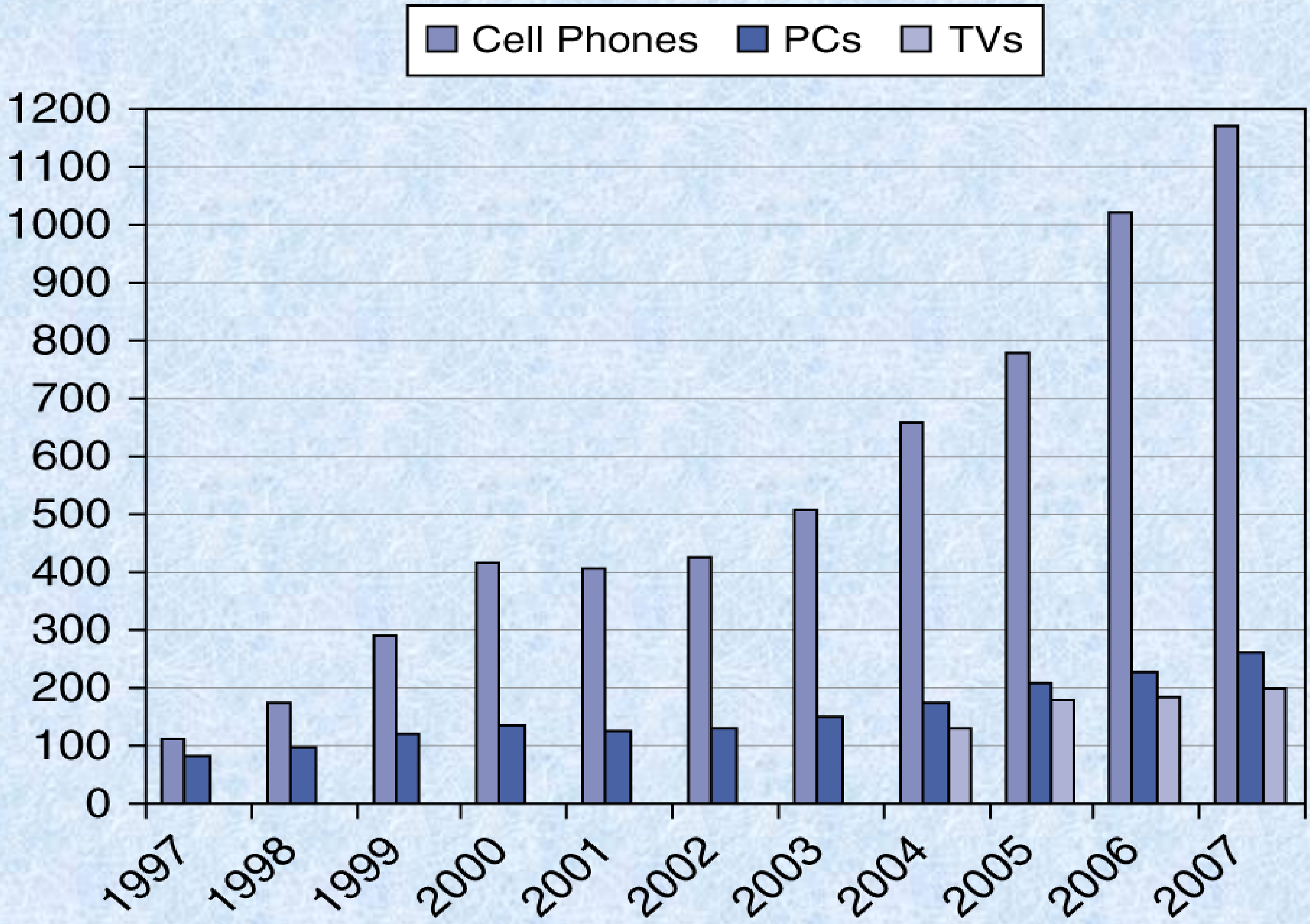


We are currently at 0.014µm

Average Transistor Cost Per Year



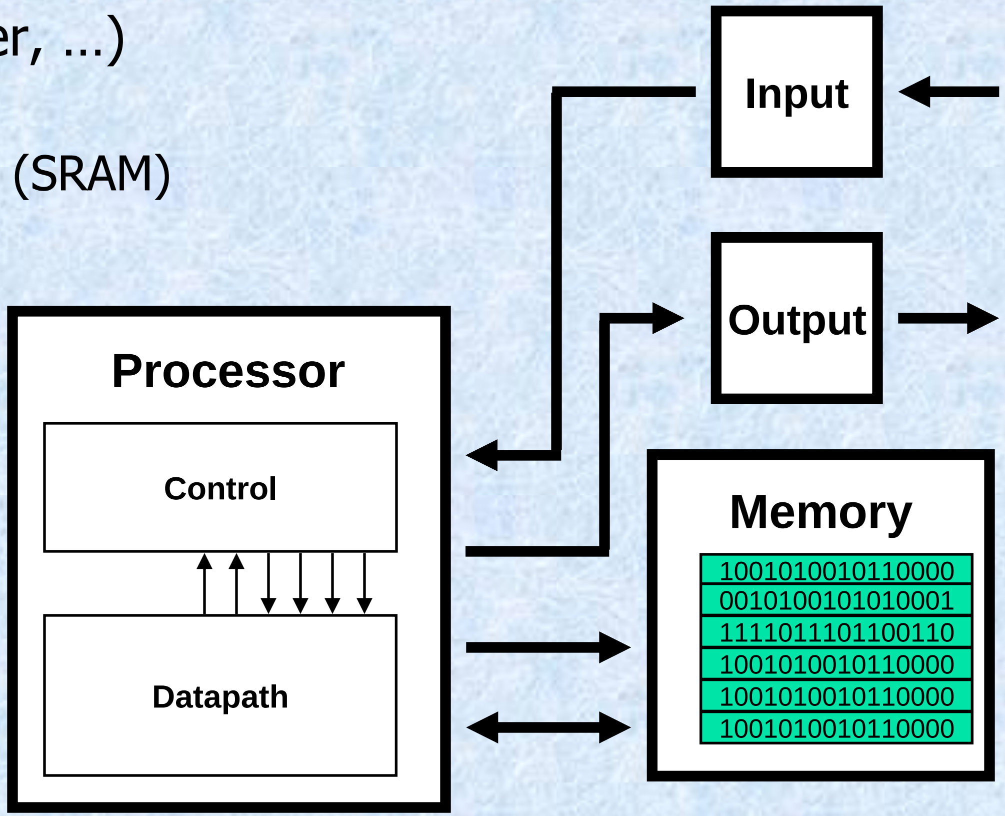
The Processor Market



The Five Classic Components of a Computer

- Input (mouse, keyboard, …)
- Output (display, printer, …)
- Memory
  - main (DRAM), cache (SRAM)
  - secondary (disk, CD, DVD, …)
- Datapath
- Control

Processor (CPU)



Our Primary Focus

- Things we'll be learning:
  - how computers work, what's a good design, what's not
  - how to make them
  - issues affecting modern processors (e.g., caches, pipelines)

- The processor (CPU)…
  - datapath
  - control
- …implemented using millions of transistors
- …impossible to understand by looking at individual transistors
- we need...

Abstraction

High Level Language

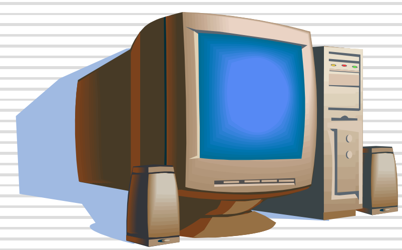
```
main() {
   int  i,b,c,a[10];
   for (i=0; i<10; i++)…
      a[2] = b + c*i;
}
```

Compiler

ISA

```
…
lw     r2, mem[r7]
add  r3, r4, r2
st      r3, mem[r8]
```

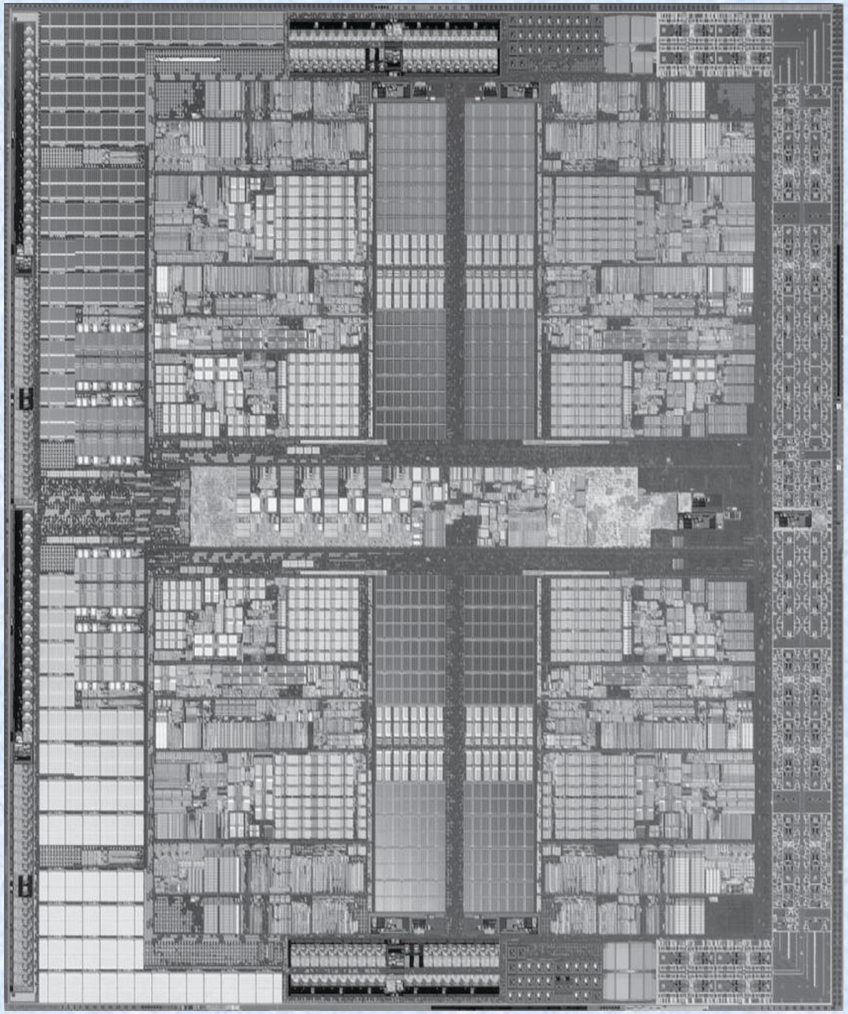
Assembler

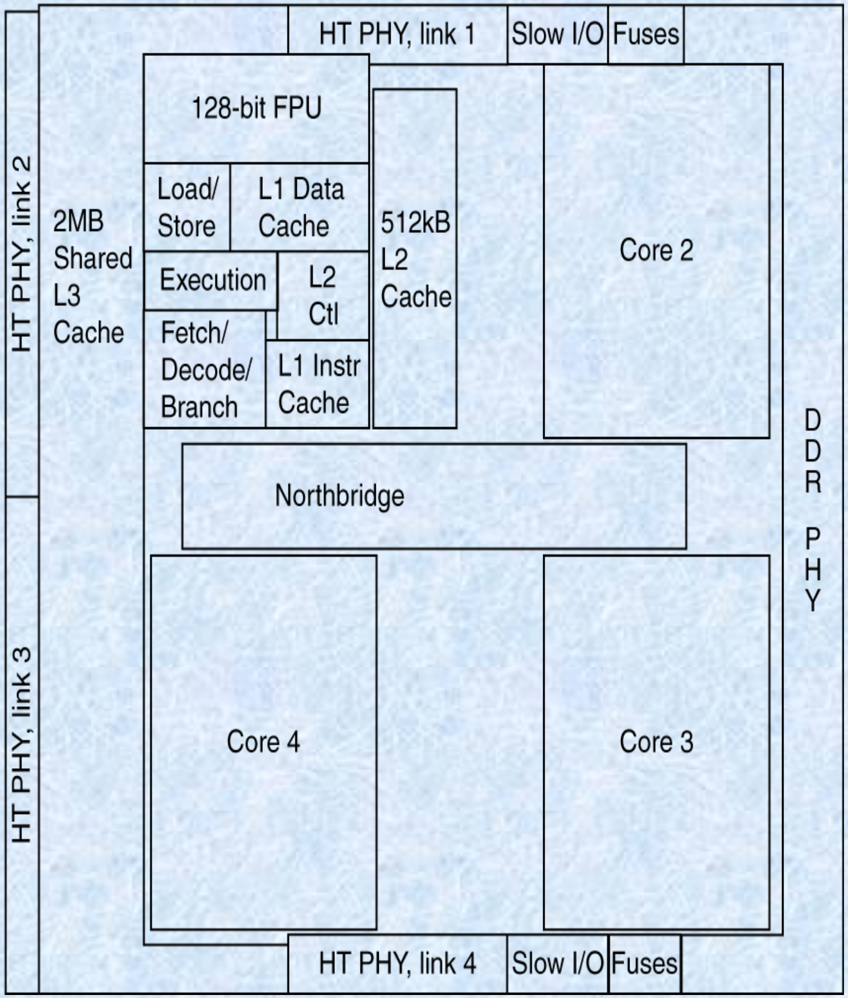


- Delving into the depths reveals more information
- An abstraction omits unneeded detail,
  helps us cope with complexity
- *What are some of the details that appear in these familiar abstractions?*

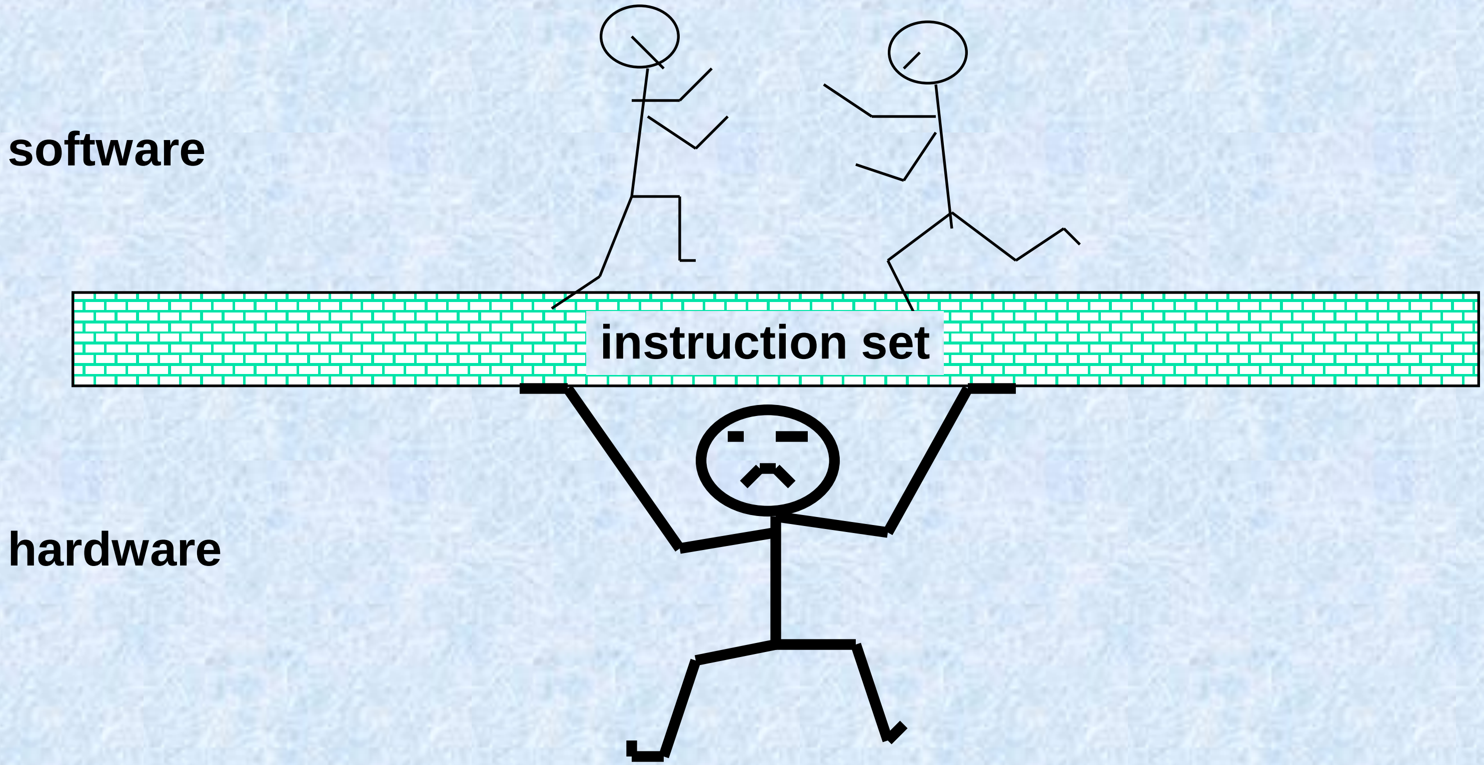
Inside the Processor

- AMD Barcelona: 4 processor cores





The Instruction Set: a Critical Interface



Instruction Set Architecture

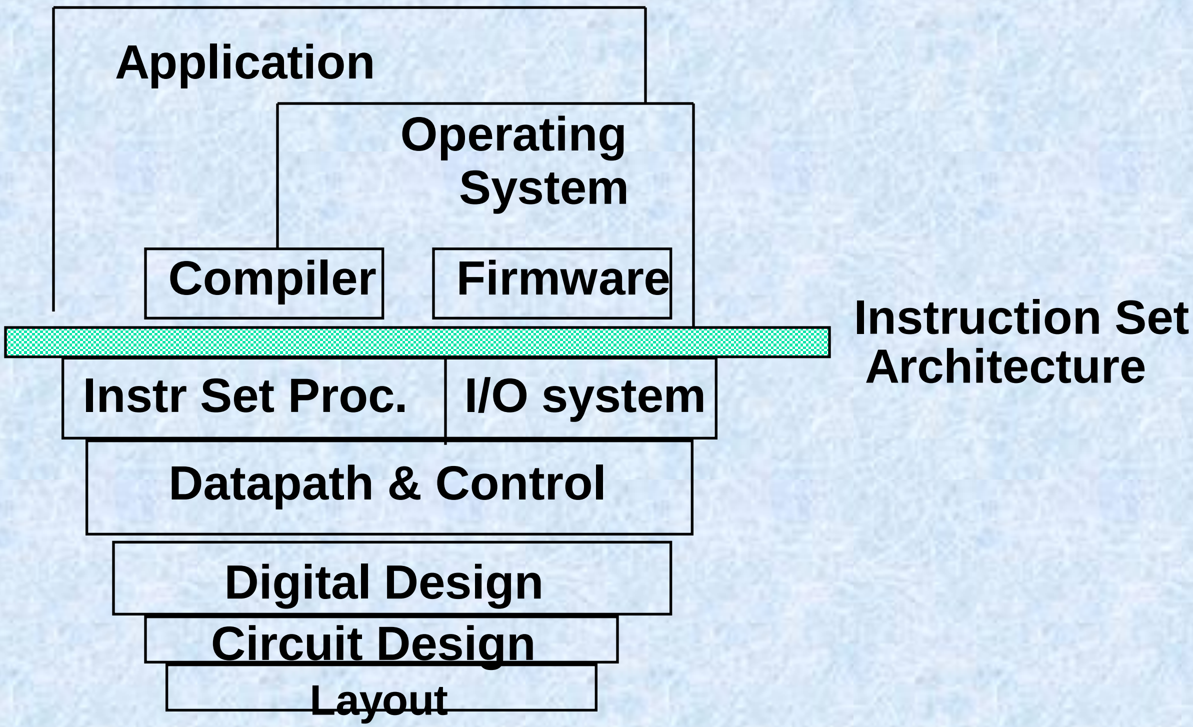
- A very important abstraction:
  - *interface* between hardware and low-level software
  - *standardizes* instructions, machine language bit patterns, etc.
  - advantage: *allows different implementations of the same architecture*
  - disadvantage: *sometimes prevents adding new innovations*

- Modern instruction set architectures:
  - 80x86/Pentium/K6, PowerPC, DEC Alpha, MIPS, SPARC, HP

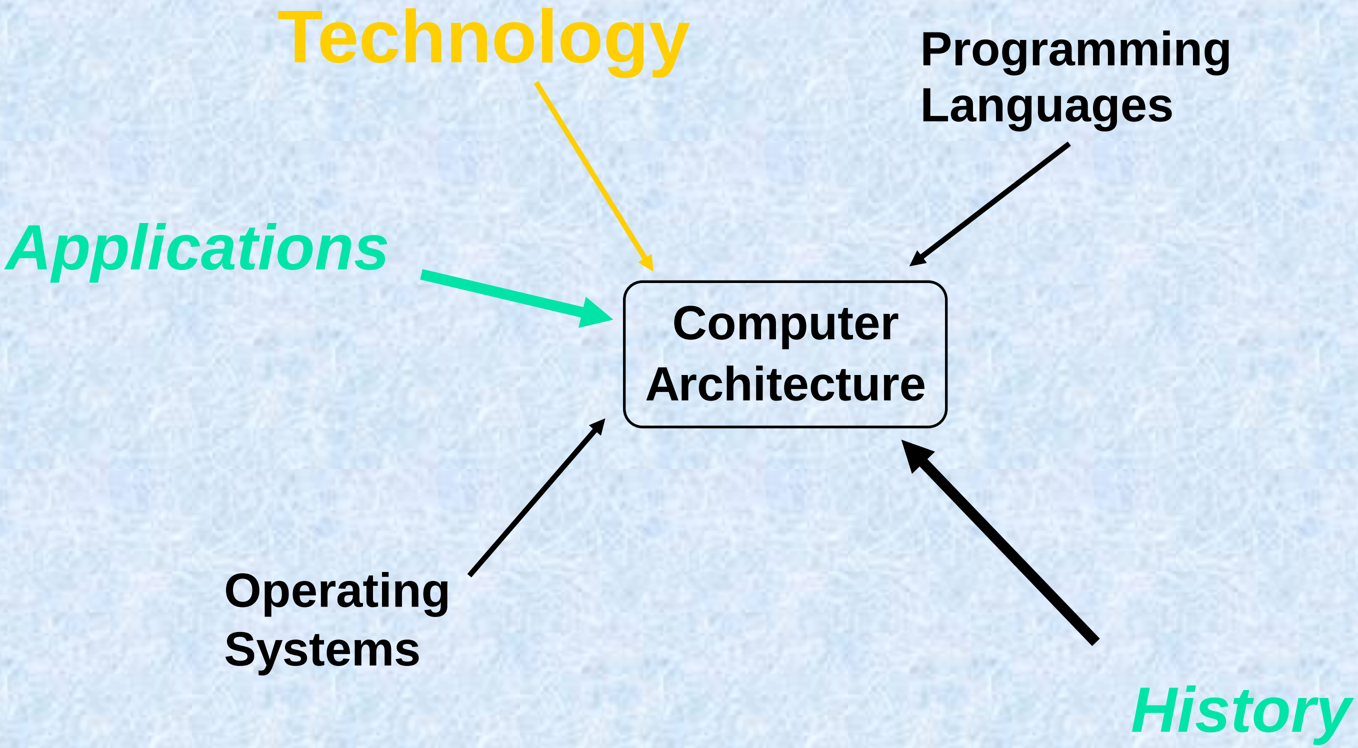
What is Computer Architecture? Easy Answer

Computer Architecture = Instruction Set Architecture + Machine Organization

What is Computer Architecture? Better (More Detailed) Answer



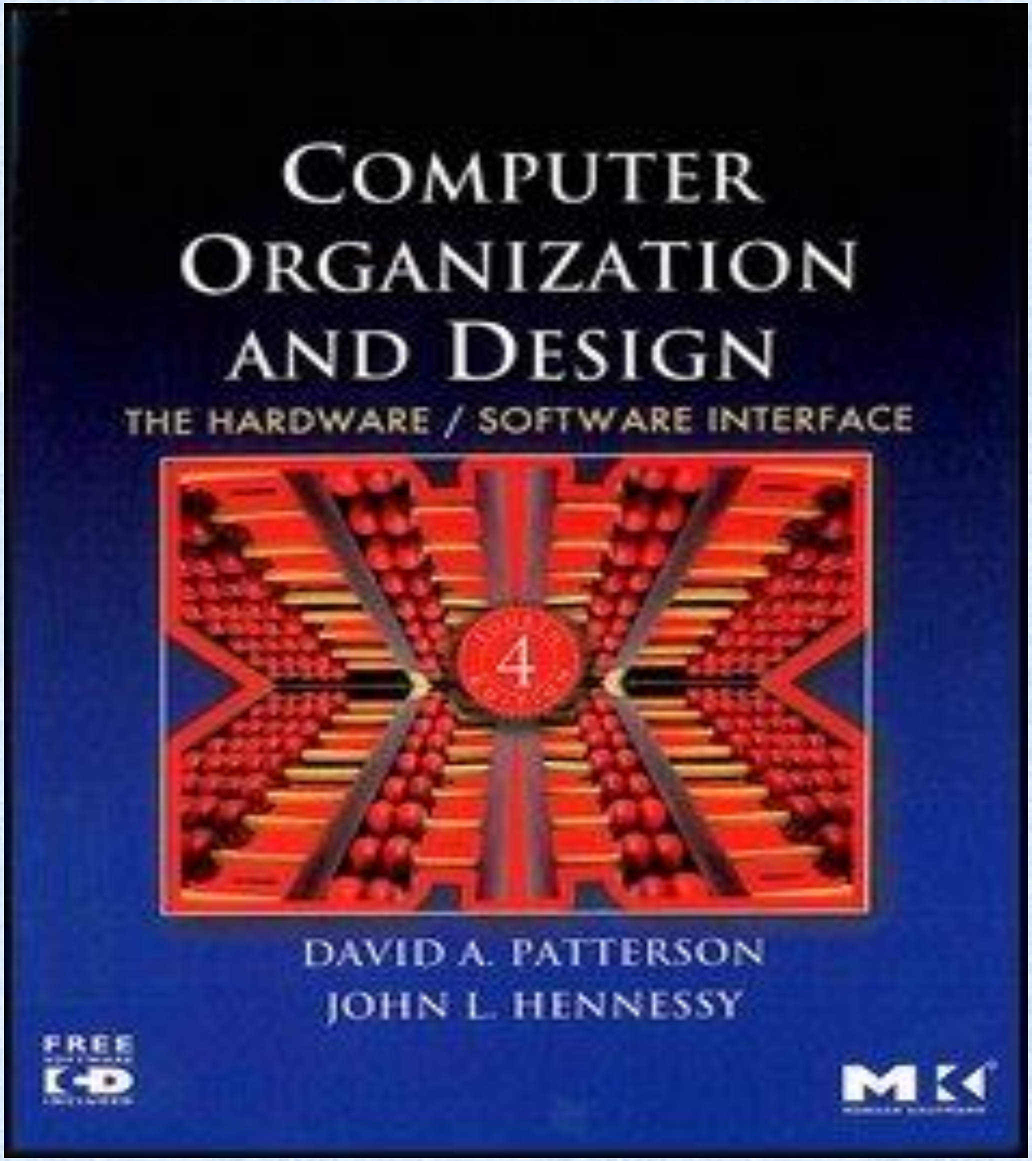
Forces on Computer Architecture



Where we are headed

- Performance issues
- A specific instruction set architecture
- Arithmetic and how to build an ALU
- Constructing a processor to execute our instructions
  Pipelining to improve performance
- Memory: caches and virtual memory
- I/O

Patterson & Hennessy book



Components

Mid Sem Test	:75
Lab (Reg+Test)	:45
Comprehensive	:80

The Role of Performance

Performance

- *Performance is the key to understanding underlying motivation for the hardware and its organization*
- Measure, report, and summarize performance to enable users to
  - make intelligent choices
  - see through the marketing hype!

- *Why is some hardware better than others for different programs?*
- *What factors of system performance are hardware related?*
  (e.g., do we need a new machine, or a new operating system?)
- *How does the machine's instruction set affect performance?*

What do we measure? Define performance….

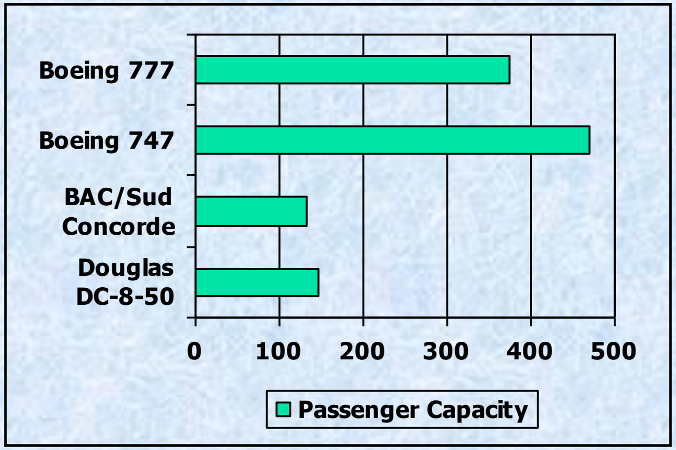
Airplane	Passengers	Range (mi)	Speed (mph)
Boeing 737-100	101	630	598
Boeing 747	470	4150	610
BAC/Sud Concorde	132	4000	1350
Douglas DC-8-50	146	8720	544

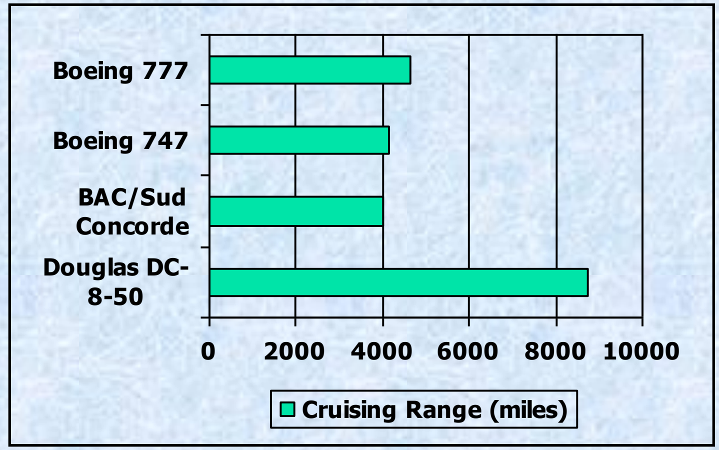
- How much faster is the Concorde compared to the 747?
- How much bigger is the Boeing 747 than the Douglas DC-8?

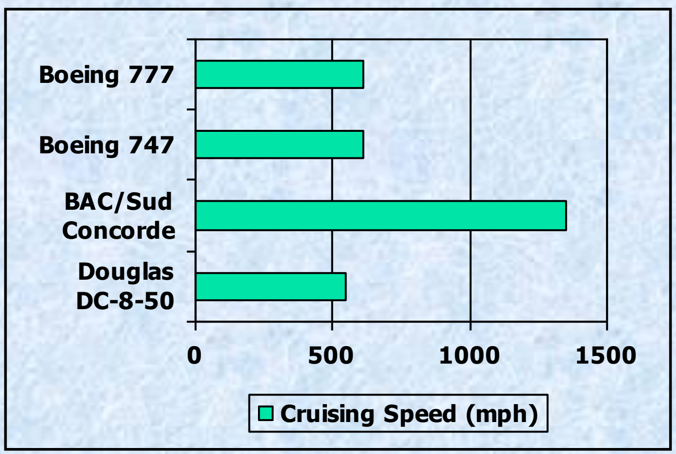
- *So which of these airplanes has the best performance?!*

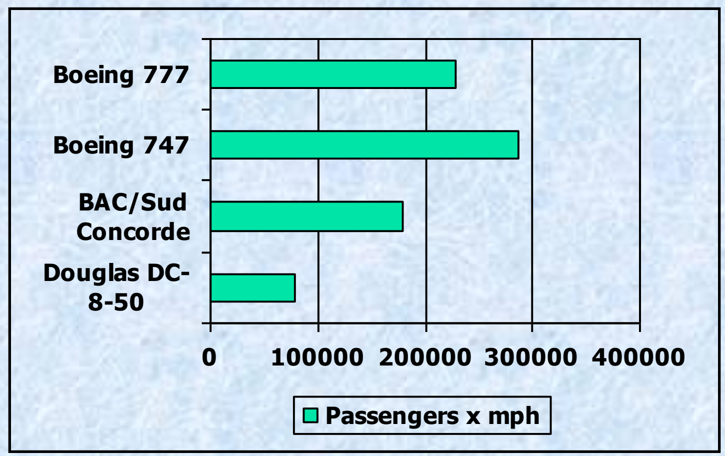
Defining Performance

- Which airplane has the best performance?









Computer Performance: TIME, TIME, TIME!!!

- *Response Time* (*elapsed time, latency*):
  - how long does it take for *my* job to run?
  - how long does it take to execute (start to finish) *my* job?
  - how long must *I* wait for the database query?

Individual user concerns…

- *Throughput*:
  - how *many* jobs can the machine run at once?
  - what is the *average* execution rate?
  - how *much* work is getting done?

Systems manager concerns…

Execution Time

- ***Elapsed Time***
  - counts everything (*disk and memory accesses, waiting for I/O, running other programs, etc.*) from start to finish
  - a useful number, but often not good for comparison purposes

  elapsed time = CPU time + wait time (I/O, other programs, etc.)

- ***CPU time***
  - doesn't count waiting for I/O or time spent running other programs
  - can be divided into *user CPU time* and *system CPU time* (OS calls)

  CPU time = user CPU time + system CPU time

  $\Rightarrow$ elapsed time = user CPU time + system CPU time + wait time

- Our focus: *user CPU time* (*CPU execution time* or, simply, *execution time*)
  - time spent executing the lines of code that are *in our program*

Definition of Performance

- For some program running on machine X:

$$\text{Performance}_X = 1 \,/\, \text{Execution time}_X$$

- *X is n times faster than Y* means:

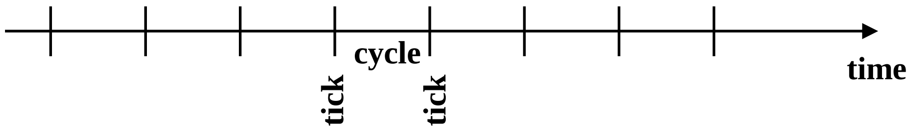
$$\text{Performance}_X \,/\, \text{Performance}_Y = n$$

Clock Cycles

- Instead of reporting execution time in seconds, we often use *cycles*. In modern computers hardware events progress cycle by cycle: in other words, each event, e.g., multiplication, addition, etc., is a sequence of cycles

$$\frac{\text{seconds}}{\text{program}} = \frac{\text{cycles}}{\text{program}} \times \frac{\text{seconds}}{\text{cycle}}$$

- *Clock ticks* indicate start and end of cycles:



- *cycle time* = time between ticks = seconds per cycle
- *clock rate* (*frequency*) = cycles per second (1 Hz. = 1 cycle/sec, 1 MHz. = 10^6 cycles/sec)
- *Example*: A 200 Mhz. clock has a $\frac{1}{200 \times 10^6} \times 10^9 = 5 \text{ nanoseconds}$ cycle time

Performance Equation I

$$\frac{\text{seconds}}{\text{program}} = \frac{\text{cycles}}{\text{program}} \times \frac{\text{seconds}}{\text{cycle}}$$

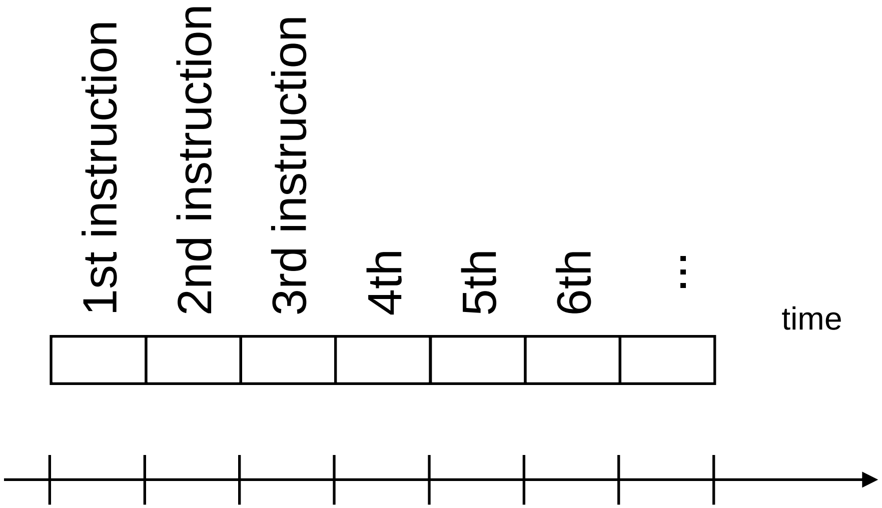
equivalently

$$\begin{array}{l}\text{CPU execution time}\\ \text{for a program}\end{array} = \begin{array}{l}\text{CPU clock cycles}\\ \text{for a program}\end{array} \times \text{Clock cycle time}$$

- So, to improve performance one can either:
  - reduce the number of cycles for a program, or
  - reduce the clock cycle time, or, equivalently,
  - increase the clock rate

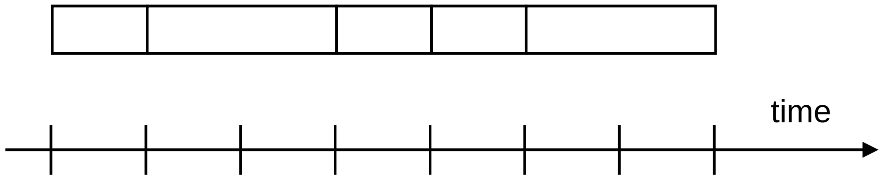
How many cycles are required for a program?

- Could assume that # of cycles = # of instructions



- *This assumption is incorrect!* Because:
  - Different instructions take different amounts of time (cycles)

How many cycles are required for a program?



- Multiplication takes more time than addition
- Floating point operations take longer than integer ones
- Accessing memory takes more time than accessing registers
- *Important point*: changing the cycle time often changes the number of cycles required for various instructions because it means changing the hardware design.

Example

- Our favorite program runs in 10 seconds on computer A, which has a 4GHz. clock.
- We are trying to help a computer designer build a new machine B, that will run this program in 6 seconds. The designer can use new (or perhaps more expensive) technology to substantially increase the clock rate, but has informed us that this increase will affect the rest of the CPU design, causing machine B to require 1.2 times as many clock cycles as machine A for the same program.

- *What clock rate should we tell the designer to target?*

Clock cycles(A)=40×10^9

$6 = 1.2 \times 40 \times 10^9 / \text{Clock rate(B)}$

➔Clock rate (B)=8GHz

Terminology

- A given program will require:
  - some number of instructions (machine instructions)
  - some number of cycles
  - some number of seconds
- We have a vocabulary that relates these quantities:
  - *cycle time* (seconds per cycle)
  - *clock rate* (cycles per second)
  - (*average*) *CPI* (cycles per instruction)
    - a floating point intensive application might have a higher average CPI
  - *MIPS* (millions of instructions per second)
    - this would be higher for a program using simple instructions

Performance Measure

- *Performance is determined by execution time*

- Do any of these other variables equal performance?
  - # of cycles to execute program?
  - # of instructions in program?
  - # of cycles per second?
  - average # of cycles per instruction?
  - average # of instructions per second?

- *Common pitfall* : thinking one of the variables is indicative of performance when it really isn't

Performance Equation II

$$\begin{array}{l}\text{CPU execution time} \\ \text{for a program}\end{array} = \begin{array}{l}\text{Instruction count} \\ \text{for a program}\end{array} \times \text{average CPI} \times \text{Clock cycle time}$$

- *Derive the above equation from Performance Equation I*

$$\frac{\text{seconds}}{\text{program}} = \frac{\text{cycles}}{\text{program}} \times \frac{\text{seconds}}{\text{cycle}}$$

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 $= 2\times1 + 1\times2 + 2\times3 = 10$
  - Avg. CPI = 10/5 = 2.0
- Sequence 2: IC = 6
  - Clock Cycles $= 4\times1 + 1\times2 + 1\times3 = 9$
  - Avg. CPI = 9/6 = 1.5

MIPS Example

- Two different compilers are being tested for a 4 GHz. machine with three different classes of instructions: Class A, Class B, and Class C, which require 1, 2 and 3 cycles (respectively). Both compilers are used to produce code for a large piece of software.
- Compiler 1 generates code with 5 billion Class A instructions, 1 billion Class B instructions, and 1 billion Class C instructions.
- Compiler 2 generates code with 10 billion Class A instructions, 1 billion Class B instructions, and 1 billion Class C instructions.

- *Which sequence will be faster according to MIPS?*
- *Which sequence will be faster according to execution time?*

Performance Summary

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

- 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

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.

- Benchmark suites
  - Perfect Club: set of application codes
  - Livermore Loops: 24 loop kernels
  - Linpack: linear algebra package
  - SPEC: mix of code from industry organization

SPEC (System Performance Evaluation Corporation)

- Sponsored by industry but independent and self-managed – trusted by code developers and machine vendors
- Clear guides for testing, see www.spec.org
- Regular updates (benchmarks are dropped and new ones added periodically according to relevance)
- Specialized benchmarks for particular classes of applications

SPEC History

- First Round: SPEC CPU89
  - 10 programs yielding a single number
- Second Round: SPEC CPU92
  - SPEC CINT92 (6 integer programs) and SPEC CFP92 (14 floating point programs)
  - compiler flags can be set differently for different programs
- Third Round: SPEC CPU95
  - new set of programs: SPEC CINT95 (8 integer programs) and SPEC CFP95 (10 floating point)
  - single flag setting for all programs
- Fourth Round: SPEC CPU2000
  - new set of programs: SPEC CINT2000 (12 integer programs) and SPEC CFP2000 (14 floating point)
  - single flag setting for all programs
  - programs in C, C++, Fortran 77, and Fortran 90

CINT2000 (Integer component of SPEC CPU2000)

Program	Language	What It Is
164.gzip	C	Compression
175.vpr	C	FPGA Circuit Placement and Routing
176.gcc	C	C Programming Language Compiler
181.mcf	C	Combinatorial Optimization
186.crafty	C	Game Playing: Chess
197.parser	C	Word Processing
252.eon	C++	Computer Visualization
253.perlbmk	C	PERL Programming Language
254.gap	C	Group Theory, Interpreter
255.vortex	C	Object-oriented Database
256.bzip2	C	Compression
300.twolf	C	Place and Route Simulator

CFP2000 (Floating point component of SPEC CPU2000)

Program	Language	What It Is
168.wupwise	Fortran 77	Physics / Quantum Chromodynamics
171.swim	Fortran 77	Shallow Water Modeling
172.mgrid	Fortran 77	Multi-grid Solver: 3D Potential Field
173.applu	Fortran 77	Parabolic / Elliptic Differential Equations
177.mesa	C	3-D Graphics Library
178.galgel	Fortran 90	Computational Fluid Dynamics
179.art	C	Image Recognition / Neural Networks
183.equake	C	Seismic Wave Propagation Simulation
187.facerec	Fortran 90	Image Processing: Face Recognition
188.ammp	C	Computational Chemistry
189.lucas	Fortran 90	Number Theory / Primality Testing
191.fma3d	Fortran 90	Finite-element Crash Simulation
200.sixtrack	Fortran 77	High Energy Physics Accelerator Design
301.apsi	Fortran 77	Meteorology: Pollutant Distribution

SPEC CPU2000 reporting

- Refer SPEC website www.spec.org for documentation
- Single number result – geometric mean of normalized ratios for each code in the suite
- Report precise description of machine
- Report compiler flag setting

SPEC CPU Benchmark

- SPEC CPU2006
  - Elapsed time to execute a selection of programs
    - Negligible I/O, so focuses on CPU performance
  - Normalize relative to reference machine
  - Summarize as geometric mean of performance ratios
    - CINT2006 (integer) and CFP2006 (floating-point)

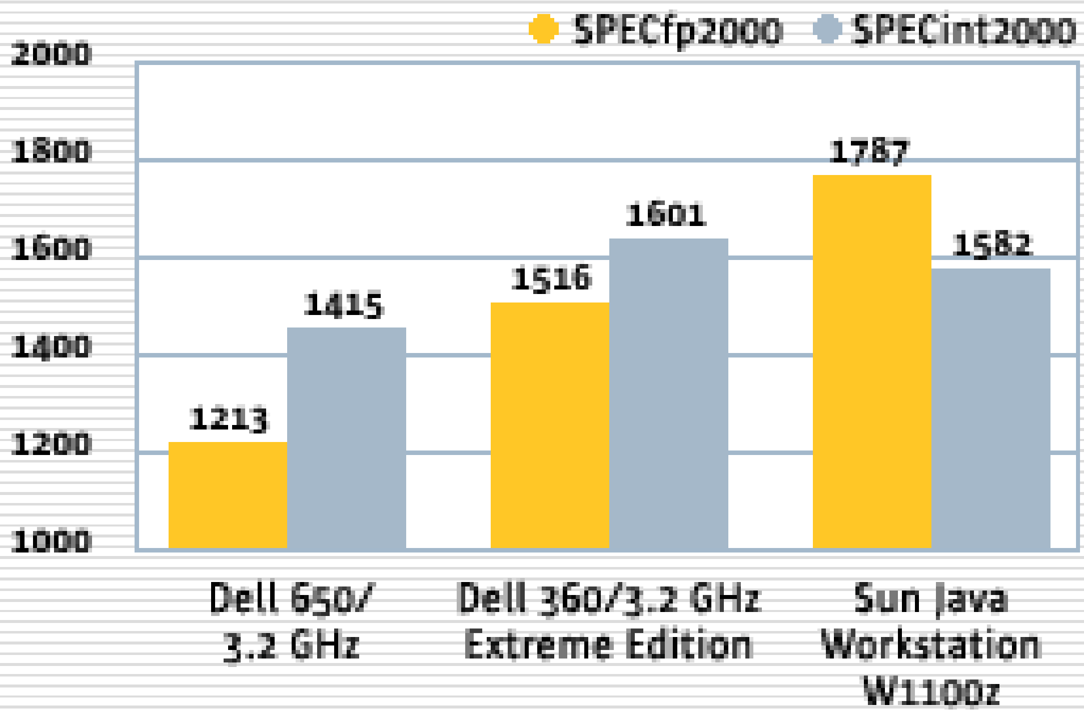
$$\sqrt[n]{\prod_{i=1}^{n} \text{Execution time ratio}_i}$$

CINT2006 for Opteron X4 2356

Name	Description	IC×10^9	CPI	Tc (ns)	Exec time	Ref time	SPECratio
perl	Interpreted string processing	2,118	0.75	0.40	637	9,777	15.3
bzip2	Block-sorting compression	2,389	0.85	0.40	817	9,650	11.8
gcc	GNU C Compiler	1,050	1.72	0.40	724	8,050	11.1
mcf	Combinatorial optimization	336	10.00	0.40	1,345	9,120	6.8
go	Go game (AI)	1,658	1.09	0.40	721	10,490	14.6
hmmer	Search gene sequence	2,783	0.80	0.40	890	9,330	10.5
sjeng	Chess game (AI)	2,176	0.96	0.40	837	12,100	14.5
libquantum	Quantum computer simulation	1,623	1.61	0.40	1,047	20,720	19.8
h264avc	Video compression	3,102	0.80	0.40	993	22,130	22.3
omnetpp	Discrete event simulation	587	2.94	0.40	690	6,250	9.1
astar	Games/path finding	1,082	1.79	0.40	773	7,020	9.1
xalancbmk	XML parsing	1,058	2.70	0.40	1,143	6,900	6.0
Geometric mean							11.7

High cache miss rates

SPEC CPU2000 Benchmark Sample Result

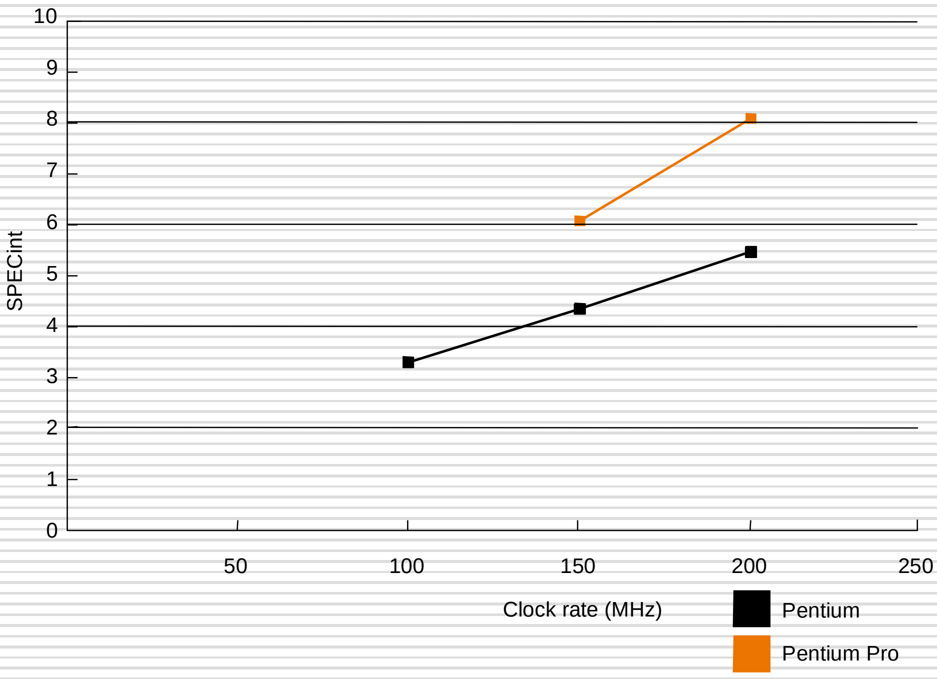


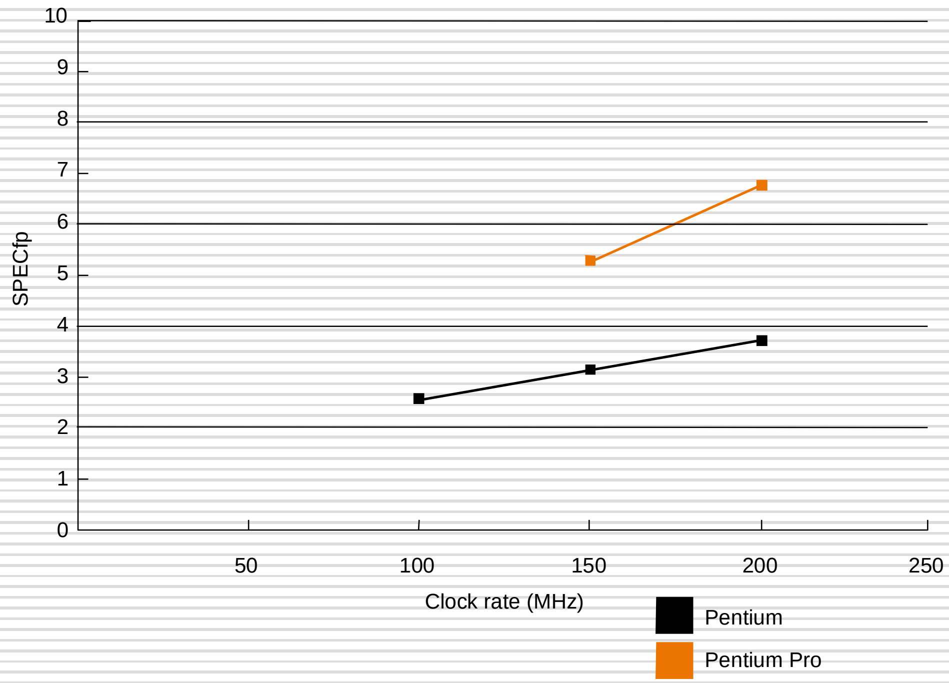
Source: Sun Microsystems
W1100z uses AMD Opteron 100 series CPU

SPEC ‘95

Does doubling the clock rate double the performance?

Can a machine with a slower clock rate have better performance?





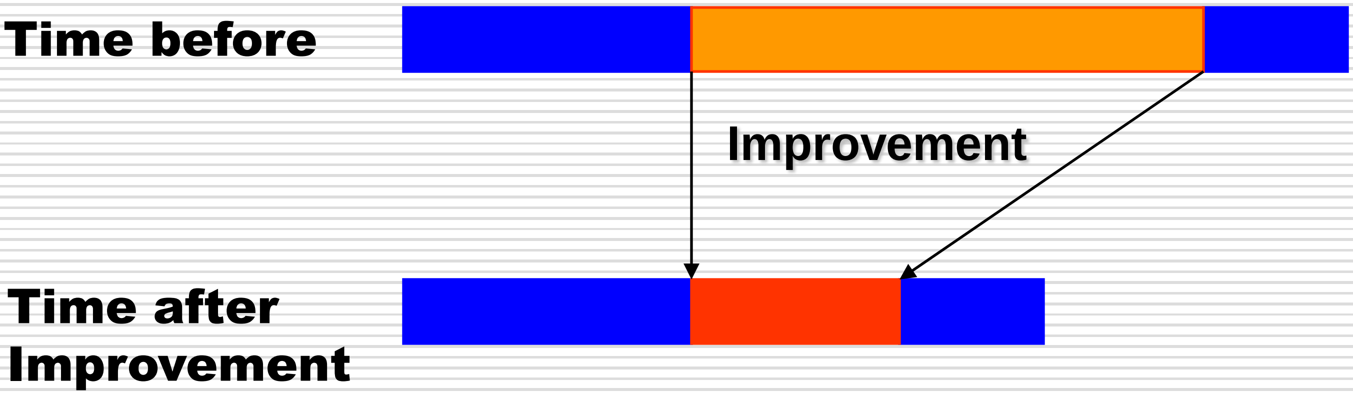
Specialized SPEC Benchmarks

- I/O
- Network
- Graphics
- Java
- Web server
- Transaction processing (databases)

Amdahl's Law

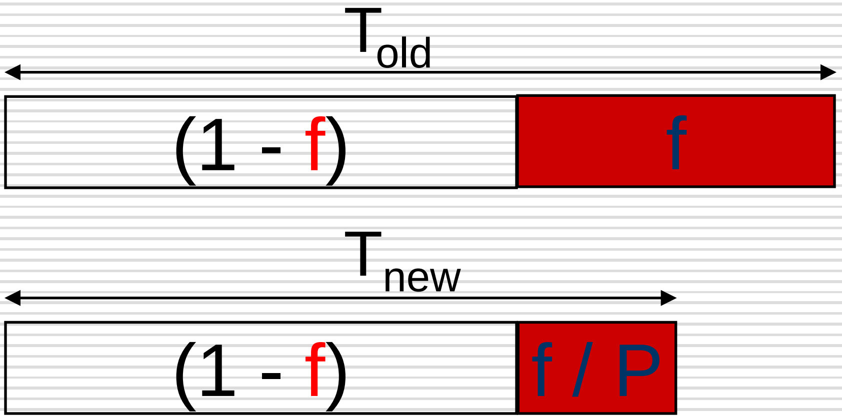
Execution Time After Improvement =

Execution Time Unaffected +(Execution Time Affected / Amount of Improvement)



Amdahl's Law

- Speed-up =
  $Perf_{new}$ / $Perf_{old}$ = $Exec\_time_{old}$ / $Exec\_time_{new}$ = $$\frac{1}{(1-f)+\frac{f}{P}}$$
- Performance improvement from using faster mode is limited by the fraction the faster mode can be applied.



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?"

  How about making it 5 times faster?

- *Principle: Make the common case fast*

Remember

- Performance is specific to a particular program/s
  - Total execution time is a consistent summary of performance

- For a given architecture performance increases come from:
  - increases in clock rate (without adverse CPI affects)
  - improvements in processor organization that lower CPI
  - compiler enhancements that lower CPI and/or instruction count

- Pitfall: expecting improvement in one aspect of a machine's performance to affect the total performance

- You should not always believe everything you read! Read carefully!