Computer Organization and Design

Performance

Henry Fuchs

Slides adapted from Don Smith, & Montek Singh, who adapted them

from Leonard McMillan and from Gary Bishop

Back to McMillan & Chris Terman, MIT 6.004 1999

Monday, Jan. 13, 2015

Lecture 2

Topics

- Defining "Performance"
- Performance Measures
- How to improve performance
- Performance pitfalls
- Examples

Reading: Patterson & Hennessy Ch 1.4 – 1.6

Why Study Performance?

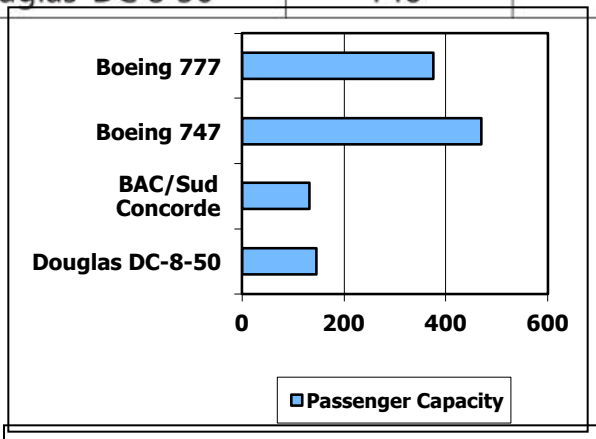
- Helps us make intelligent design choices
  - See through the marketing hype
- Key to understanding underlying computer organization
  - Why is some hardware faster than others for different programs?
  - What factors of system performance are hardware related?
    - e.g., Do we need a new machine …
    - … or a new operating system?
  - How does a machine's instruction set affect its performance?

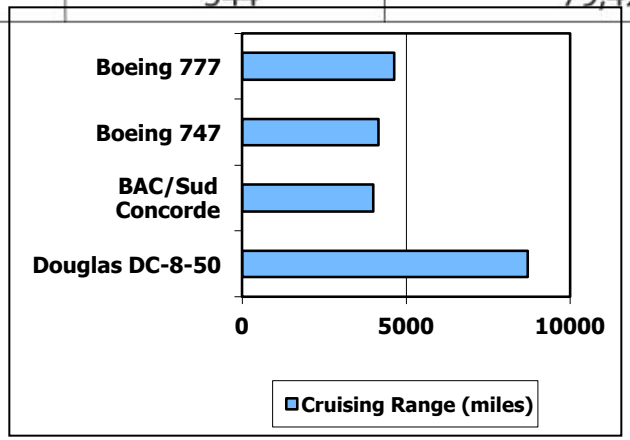
What is Performance?

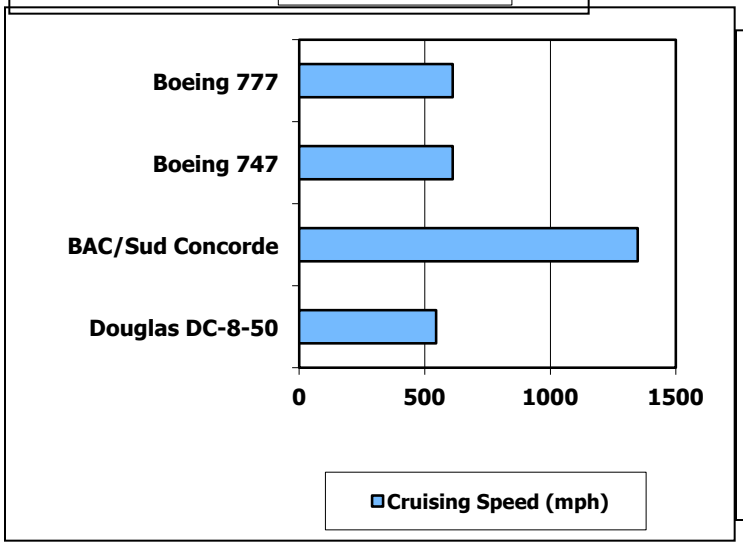
- Loosely:
  - How fast can a computer complete a task
- Examples of "tasks":
  - Short tasks:
    - Crunch a bunch of numbers (say calculate mean)
    - Display a PDF document
    - Respond to a game console button press
  - Longer ones:
    - Search for a document on hard drive
    - Rip a CD track into an mp3 file
    - Apply a Photoshop filter
    - Transcode/recode a video

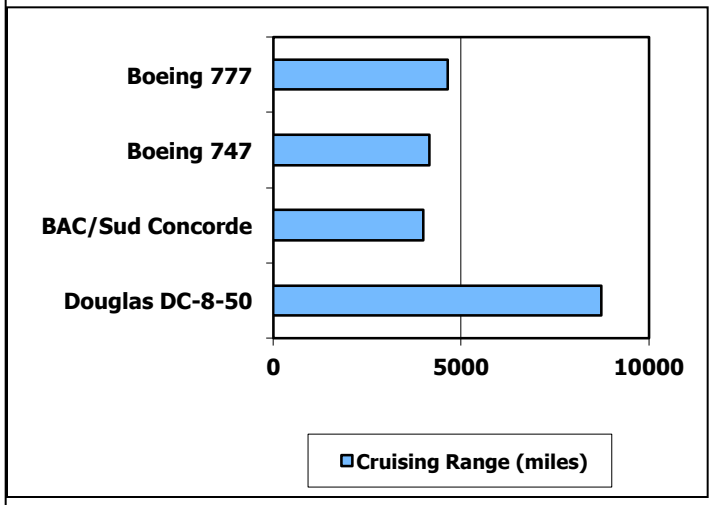
Which airplane is "best"?

Airplane	Passenger capacity	Cruising range (miles)	Cruising speed (mph)	Passenger throughput (passengers x mph)
Boeing 777	375	4630	610	228,750
Boeing 747	470	4150	610	286,700
BAC/Sud Co ncorde	132	400 0	1350	178,200
Douglas DC-8-50	146	8720	544	79,424



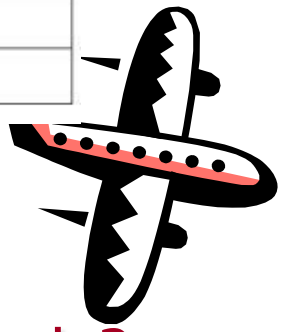






Which airplane is "best"?

Airplane	Passenger capacity	Cruising range (miles)	Cruising speed (mph)	Passenger throughput (passengers x mph)
Boeing 777	375	4630	610	228,750
Boeing 747	470	4150	610	286,700
BAC/Sud Co ncorde	132	400 0	1350	178,200
Douglas DC-8-50	146	8720	544	79,424



- How much faster is the Concorde than the 747?
  - 2.2 X *("X" means "factor of")*
- How much larger is the 747's capacity than the Concorde?
  - 3.6 X
- It is roughly 4000 miles from Raleigh to Paris. What is the throughput of the 747 in passengers/hr? The Concorde?

$$470 \times \frac{610}{4000} = 71.7 \text{ passengers/hr} \qquad 132 \times \frac{1350}{4000} = 44.6 \text{ passengers/hr}$$

- What is the latency (trip time) of the 747? The Concorde?
  - 6.56 hours, 2.96 hours

Performance Metrics

- Latency: Time from input to corresponding output
  - How long does it take for my program to run?
  - How long must I wait after typing return for the result?
  - Other examples?

- Throughput: Results produced per unit time
  - How many results can be processed per second?
  - What is the average execution rate of my program?
  - How much work is getting done each second?
  - Other examples?

Design Tradeoffs

- Performance is rarely the sole factor
  - Cost is important too
  - Energy/power consumption is often critical

- Frequently used compound metrics
  - Performance/Cost (throughput/$)
  - Performance/Power (throughput/watt)
  - Work/Energy (total work done per joule)
    - for battery-powered devices

Execution Time

- Elapsed Time/Wall Clock Time
  - counts everything (disk and memory accesses, I/O, etc.)
  - includes the impact of other programs
  - a useful number, but often not good for comparison purposes
- CPU time
  - does not include I/O or time spent running other programs
  - can be broken up into system time, and user time
- Our focus: user CPU time
  - time spent executing actual instructions of "our" program

Performance

- For some program running on machine X,
  - Performance_X = Program Executions / Time_X (executions/sec)
    = 1 / Execution Time_X
- Relative Performance
  - "X is *n* times faster than Y"
  - Performance_X / Performance_Y = *n*
- Example:
  - Machine A runs a program in 20 seconds
  - Machine B runs the same program in 25 seconds
    - By how much is A faster than B?
    - By how much is B slower than A?

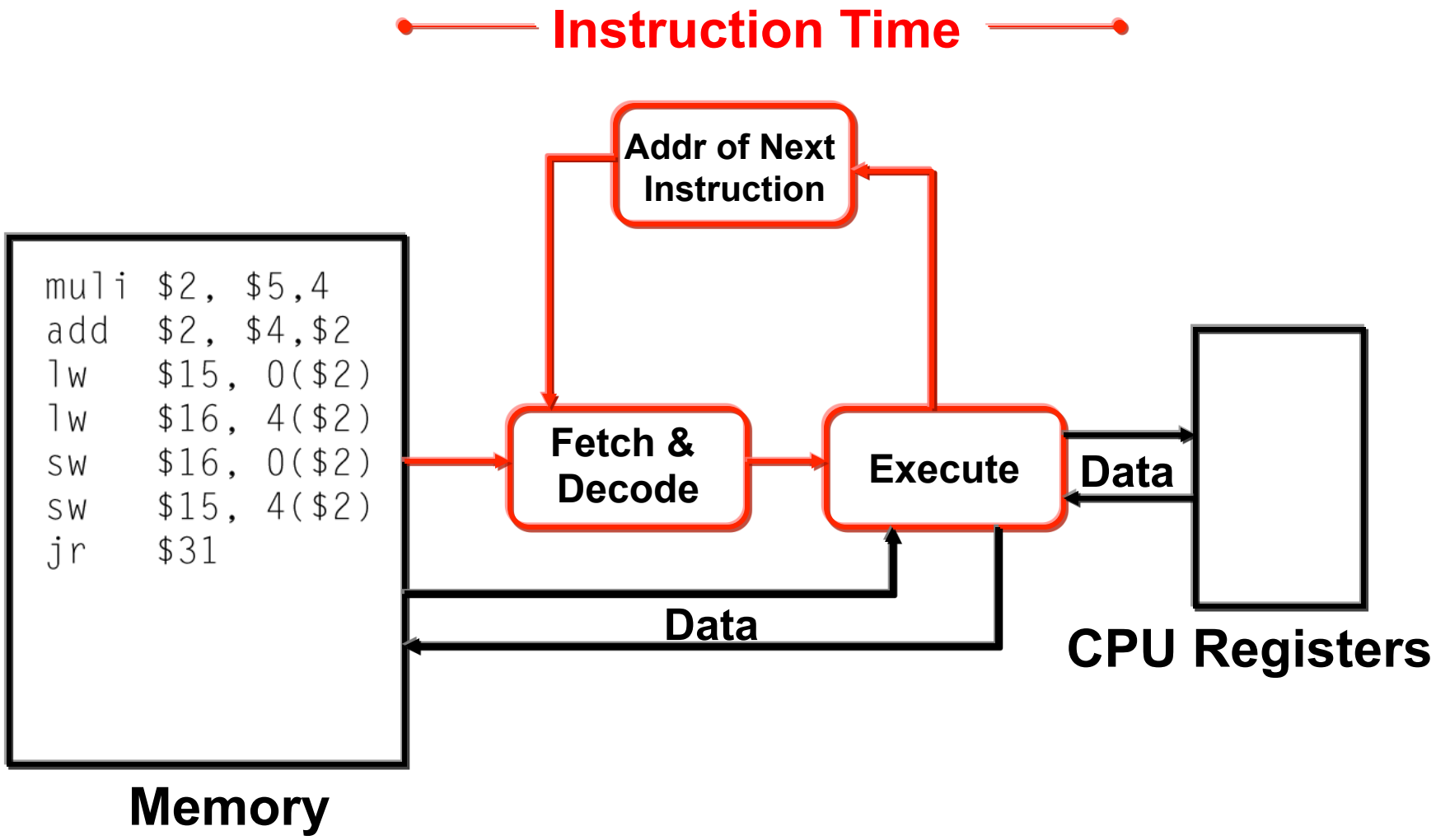
$\text{Performance}_A = 1/20$ $\text{Performance}_B = 1/25$

Machine A is (1/20)/(1/25) = 1.25 times faster than Machine B

Performance: Pitfalls of using %

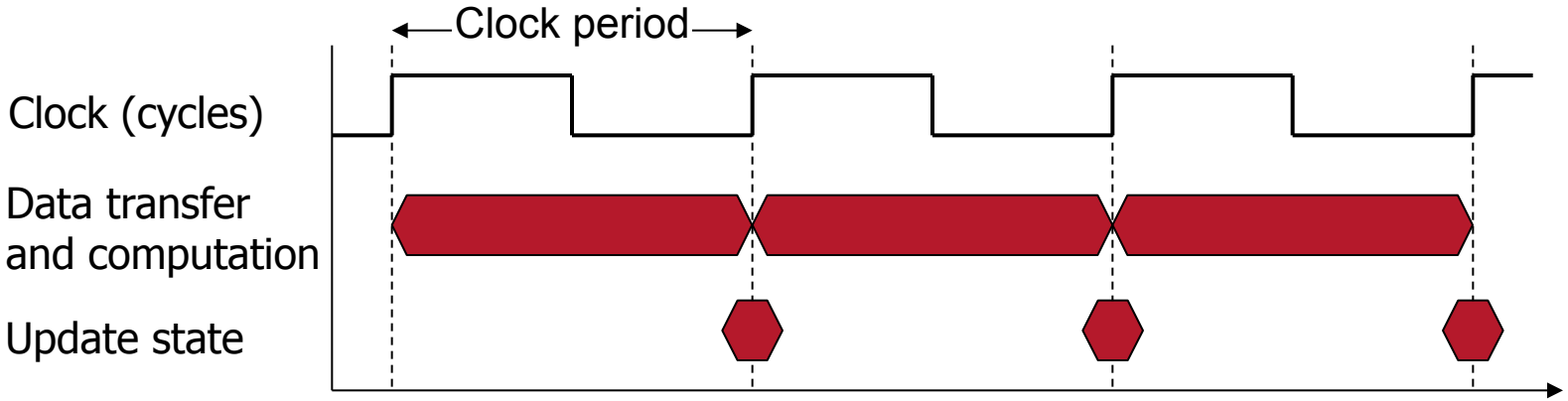
- Same Example:
  - Machine A runs a program in 20 sec; B takes 25 sec
  - By how much is A faster than B?
    - Is it (25-20)/25 = 20% faster?
    - Is it (25-20)/20 = 25% faster?
    - Correct answer is: A is $(Perf_A - Perf_B)/Per_B$ faster
      (0.05 - 0.04) / (0.04) = 25% faster
  - By how much is B slower than A?
    - Correct answer is: B is $(Perf_B - Perf_A)/Per_A$ faster/slower
      (0.04 - 0.05) / (0.05) = -20% faster = 20% slower
  - Confusing: A is 25% faster than B; B is 20% slower
  - Better: A is 1.25 times faster than B; B is 1/1.25 as fast
- Also: %ages are only good up to 100%
  - Don't say A is 13000% faster than B; say A is 130 times faster than B

Processor (based on Von Neumann)



CPU Clocking

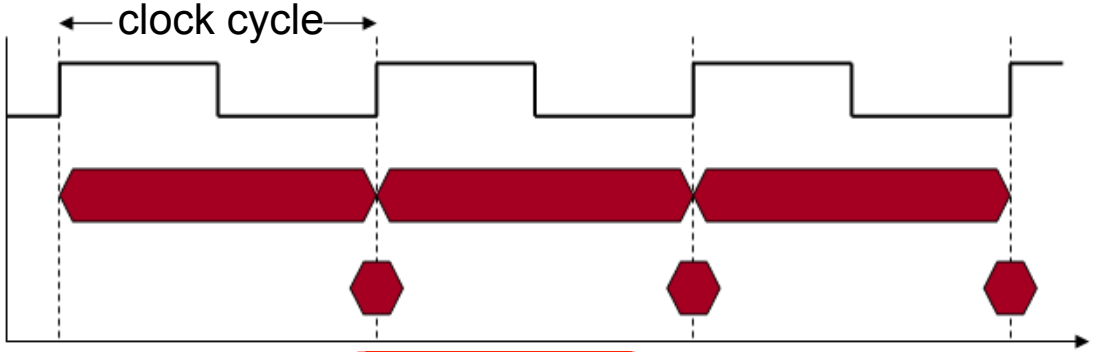
- Operation of digital hardware governed by a constant-rate clock

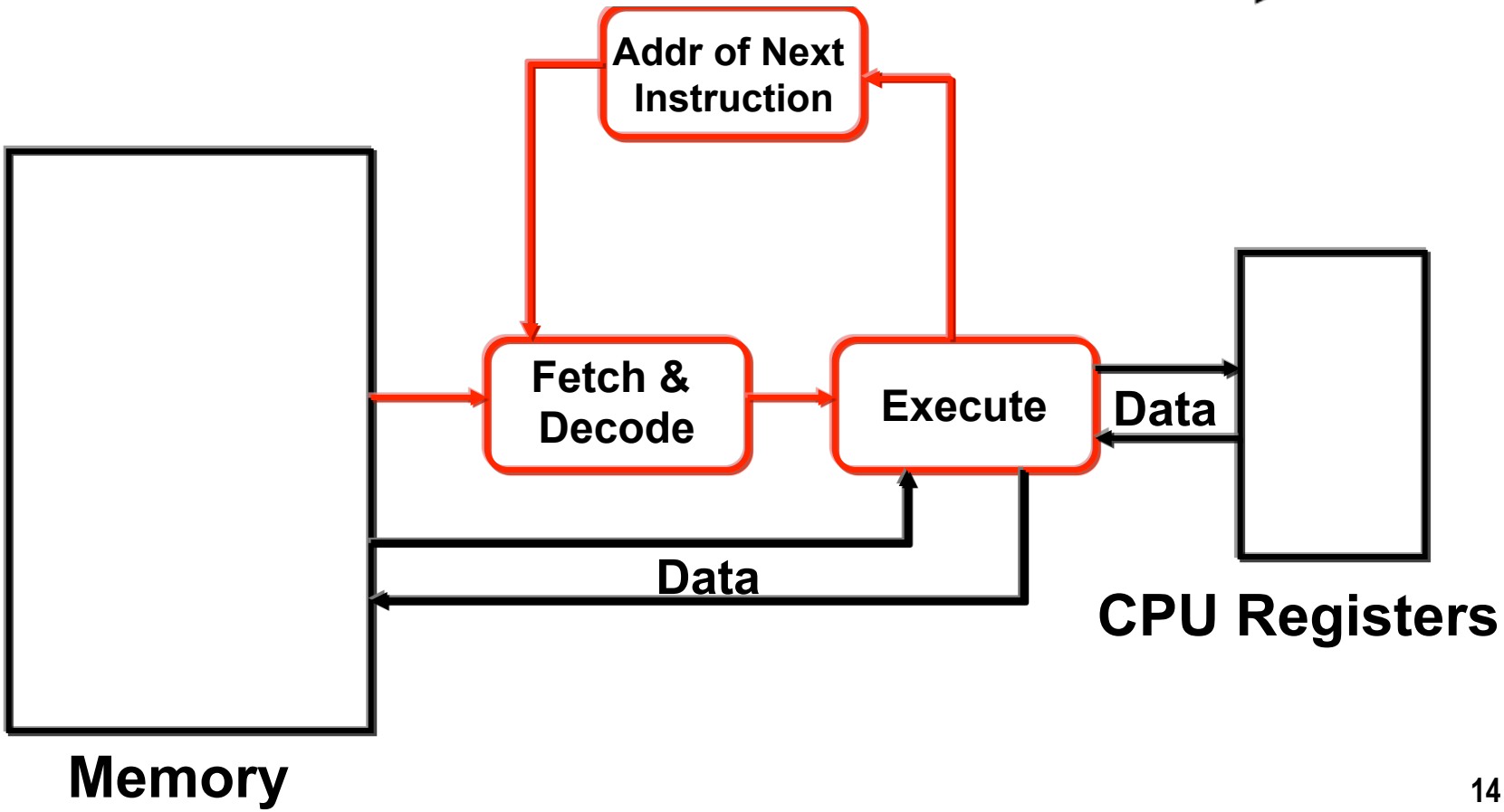


- Clock period: duration of a clock cycle
  - e.g., 250ps = 0.25ns = 250×10^{-12}s
- Clock frequency (rate): cycles per second
  - e.g., 1/(0.25ns) = 4GHz = 4,000MHz = 4.0×10^{9}Hz

Clock Cycles and Instruction Time

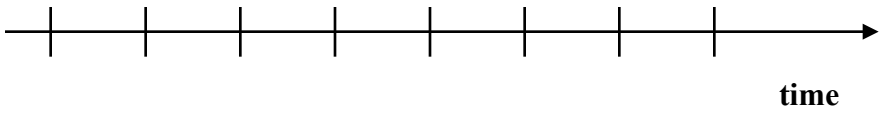
Cycles per Instruction (CPI)





Program Clock Cycles

- Instead of reporting execution time in seconds, we often use clock cycle counts
  - Why? A newer generation of the same processor…
    - Often has the same cycle counts for the same program
    - But often has different clock speed (ex, 1 GHz changes to 1.5 GHz)
- Clock "ticks" indicate when machine state changes
  - an abstraction: allows time to be discrete instead of continuous



- Simple relation:

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

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

Program Clock Cycles

- Relation:

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

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

- or:

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

- cycle time = time between ticks = seconds per cycle
- clock rate (frequency) = cycles per second (1 Hz = 1 cycle/s)

- A 200 MHz clock has a cycle time of:

$$\frac{1}{200 \times 10^6} = 5 * 10^{-9} = 5 \text{ ns}$$

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
    - Instruction set architecture (ISA)
    - compiler
- Average cycles per instruction ("CPI")
  - Determined by CPU hardware
  - If different instructions have different CPI
    - Average CPI affected by instruction mix

Computer Performance Measure

Millions of Instructions per Second

Frequency in MHz

$$MIPS = \frac{\text{clocks/sec}}{\text{AVE(clocks/instruction)}}$$

Unfortunate coincidence:
This "MIPS" has nothing to do with the name of the MIPS processor we will be studying!

CPI (Average Clocks Per Instruction)

Historically:

PDP-11, VAX, Intel 8086:	CPI > 1
Load/Store RISC machines	
MIPS, SPARC, PowerPC, miniMIPS:	CPI = 1
Modern CPUs, Pentium, Athlon:	CPI < 1

How to Improve Performance?

- Many ways to write the same equations:

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

$$\text{MIPS} = \frac{\text{Freq}}{\text{CPI}}$$

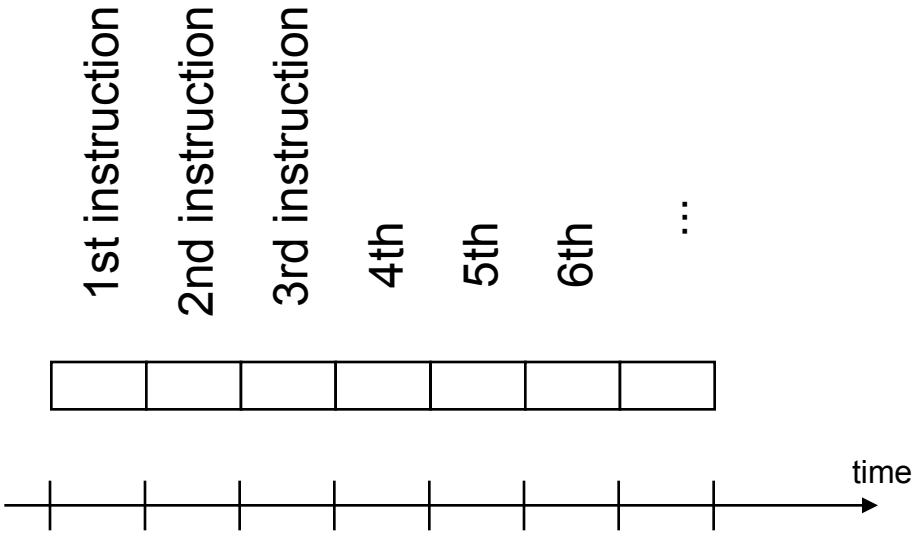
- So, to improve performance (everything else being equal) you can either
  - Decrease the # of required cycles for a program;
  - Decrease the clock cycle time or, said another way,
  - Increase the clock rate;
  - Decrease the CPI (average clocks per instruction).

- MIPS Pitfall
  - Cannot compare MIPS of two different processors if they run different sets of instructions!

➔ **M**eaningless **I**ndicator of **P**rocessor **S**peed!

How Many Cycles in a Program?

- For some processors (e.g., MIPS processor)
  - # of cycles = # of instructions



This assumption can be incorrect,

Different instructions take different amounts of time on different machines.
Memory accesses might require more cycles than other instructions.
Floating-Point instructions might require multiple clock cycles to execute.
Branches might stall execution rate

Example

- 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 a new machine B, to 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?

$$\frac{\text{cycles}}{\text{program}} = \left(\frac{\text{seconds}}{\text{program}}\right)_A \times \frac{\text{cycles}}{\text{second}} = 10 \times 2 \times 10^9 = 2 \times 10^{10}$$

$$\frac{\text{cycles}}{\text{second}} = \frac{\text{cycles/program}}{(\text{seconds/program})_B} = \frac{1.2 \times 2 \times 10^{10}}{6} = 4 \times 10^9 = 4GHz$$

Now that we understand cycles

- 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)
  - CPI (average clocks per instruction)
    - a floating point intensive application might have a higher CPI
  - MIPS (millions of instructions per second)
    - this would be higher for a program using simple instructions

Performance Traps

- Performance is determined by the execution time of a program that you care about.

- Do any of the 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 that only one of the variables is indicative of performance when it really is not!

CPI Example

- Suppose we have two implementations of the same instruction set architecture (ISA)
  - If two machines have the same ISA which quantity (e.g., clock rate, CPI) is the same?
  - For some program:
    - Computer A has a clock cycle time of 250 ps and a CPI of 2.0
    - Computer B has a clock cycle time of 500 ps and a CPI of 1.2
  - What machine is faster for this program, and by how much?

$$Time_A = InstructionCount * CPI_A * CycleTime_A$$
$$= IC * 2.0 * 250\,ps = IC * 500\,ps$$

$$Time_B = InstructionCount * CPI_B * CycleTime_B$$
$$= IC * 1.2 * 500\,ps = IC * 600\,ps$$

$$\text{Relative Performance} = \frac{Time_B}{Time_A} = \frac{600}{500} = 1.2$$

A is faster by 1.2 X

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:

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

Example: Compiler's Impact

- Two different compilers are being tested for a 500 MHz machine with three different classes of instructions: Class A, Class B, and Class C, which require one, two, and three cycles (respectively). Both compilers are used to produce code for a large piece of software. The first compiler's code uses 5 million Class A instructions, 1 million Class B instructions, and 2 million Class C instructions. The second compiler's code uses 7 million Class A instructions, 1 million Class B instructions, and 1 million Class C instructions.

Which program uses fewer instructions?

- Instructions$_1$ = (5+1+2) x 10^6 = 8 x 10^6
- Instructions$_2$ = (7+1+1) x 10^6 = 9 x 10^6

CPI_1 = ? 13/8 = 1.625

CPI_2 = ? 12/9 = 1.33

- Which sequence uses fewer clock cycles?
  - Cycles$_1$ = (5(1)+1(2)+2(3)) x 10^6 = 13 x 10^6
  - Cycles$_2$ = (7(1)+1(2)+1(3)) x 10^6 = 12 x 10^6

CPI Example

- Alternative compiled code versions using instructions in classes A, B, C

Class	A	B	C
CPI for class	1	2	3
IC for version 1	2	1	2
IC for version 2	4	1	1

- Version 1: IC = 5
  - Clock Cycles
    = 2×1 + 1×2 + 2×3
    = 10
  - Avg. CPI = 10/5 = 2.0
- Version 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, Cycle Time

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.
- Small benchmarks
  - nice for architects and designers
  - easy to standardize
  - can be abused
- SPEC (System Performance Evaluation Cooperative)
  - companies have agreed on a set of real program and inputs
  - can still be abused
  - valuable indicator of performance (and compiler technology)

SPEC '95

Benchmark	Description
go	Artificial intelligence; plays the game of Go
m88ksim	Motorola 88k chip simulator; runs test program
gcc	The Gnu C compiler generating SPARC code
compress	Compresses and decompresses file in memory
li	Lisp interpreter
ijpeg	Image compression and decompression
perl	Manipulates strings and prime numbers in the special-purpose programming language Perl
vortex	A database program
tomcatv	A mesh generation program
swim	Shallow water model with 513 x 513 grid
su2cor	quantum physics; Monte Carlo simulation
hydro2d	Astrophysics; Hydrodynamic Naiver Stokes equations
mgrid	Multigrid solver in 3-D potential field
applu	Parabolic/elliptic partial differential equations
trub3d	Simulates isotropic, homogeneous turbulence in a cube
apsi	Solves problems regarding temperature, wind velocity, and distribution of pollutant
fpppp	Quantum chemistry
wave5	Plasma physics; electromagnetic particle simulation

- Periodically updated with newer benchmarks
  - SPEC 2000, SPEC 2006

SPEC 2006

- Interesting to see how the mix of applications has changed over the years…
  - See http://www.spec.org/cpu2006/{CINT2006, CFP2006}

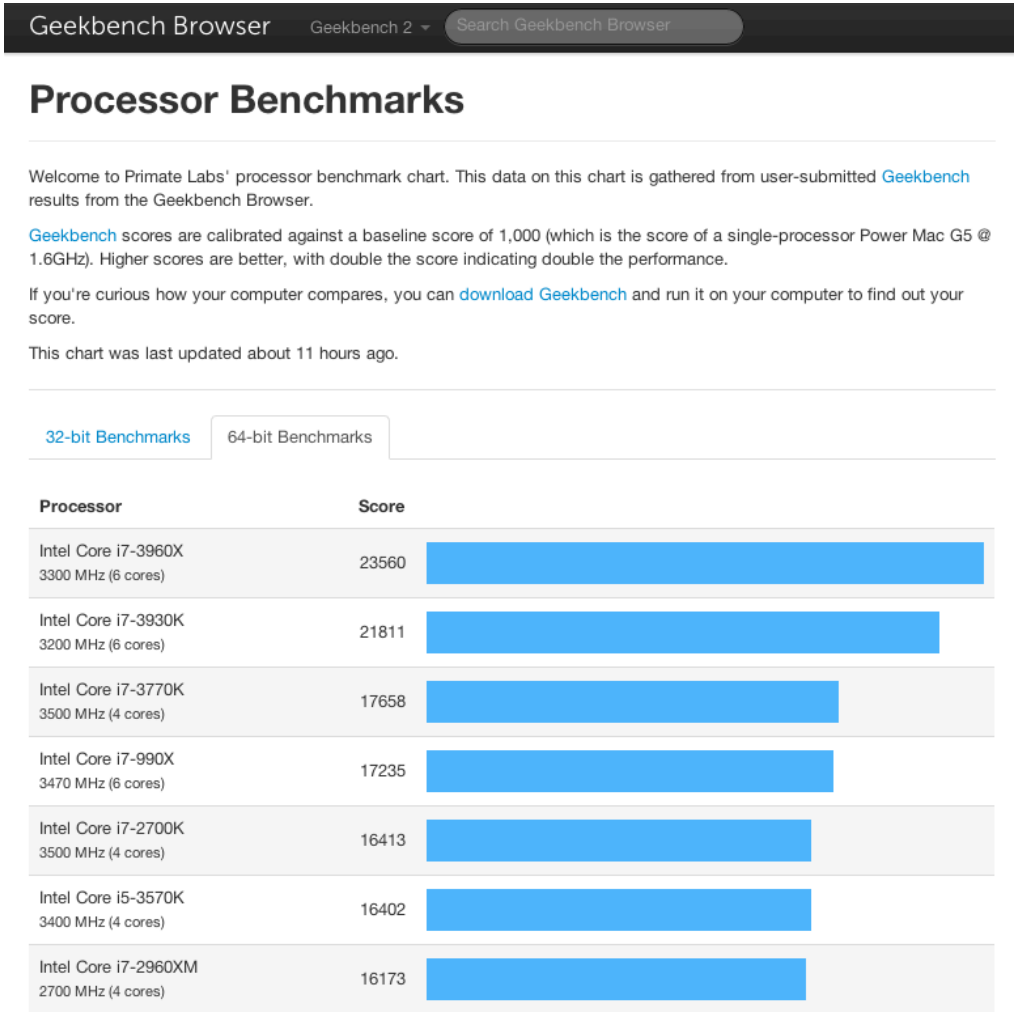
CINT2006 (Integer Component of SPEC CPU2006):

Benchmark	Language	Application Area	Brief Description
400.perlbench	C	Programming Language	Derived from Perl V5.8.7. The workload includes SpamAssassin, MHonArc (an email indexer), and specdiff (SPEC's tool that checks benchmark outputs).
401.bzip2	C	Compression	Julian Seward's bzip2 version 1.0.3, modified to do most work in memory, rather than doing I/O.
403.gcc	C	C Compiler	Based on gcc Version 3.2, generates code for Opteron.
429.mcf	C	Combinatorial Optimization	Vehicle scheduling. Uses a network simplex algorithm (which is also used in commercial products) to schedule public transport.
445.gobmk	C	Artificial Intelligence: Go	Plays the game of Go, a simply described but deeply complex game.
456.hmmer	C	Search Gene Sequence	Protein sequence analysis using profile hidden Markov models (profile HMMs)
458.sjeng	C	Artificial Intelligence: chess	A highly-ranked chess program that also plays several chess variants.
462.libquantum	C	Physics / Quantum Computing	Simulates a quantum computer, running Shor's polynomial-time factorization algorithm.

Other Popular Benchmarks

- Several others popular
  - industry uses SPEC
  - but ordinary consumers use others
    - more representative of the work they do!

      e.g., gaming, Photoshop/ Aperture, copying huge files,
      multimedia coding/ decoding, etc.
    - Geekbench is quite popular!



Amdahl's "Law"

Possibly the most important observation regarding computer performance
(AFIPS Spring Joint Computer Conference 1967)

$$t_{improved} = \frac{t_{affected}}{r_{speedup}} + t_{unaffected}$$

Gene Amdahl (1922-)
Comp. architect &
entrepreneur

- Principle: Make the common case fast!
- Eventually, performance gains will be limited by what cannot be improved
  - e.g., you can raise the speed limit, but there are still traffic lights
- Examples (next slide)

Amdahl's Law: Example

$$t_{improved} = \frac{t_{affected}}{r_{speedup}} + t_{unaffected}$$

- Example: "Suppose a program runs in 100 seconds on a machine, where multiplies are executed 80% of the time. How much do we need to improve the speed of multiplication if we want the program to run 4 times faster?"

  25 = 80/r + 20 r = 16x

  How about making it 5 times faster?

  20 = 80/r + 20 r = ?

Example

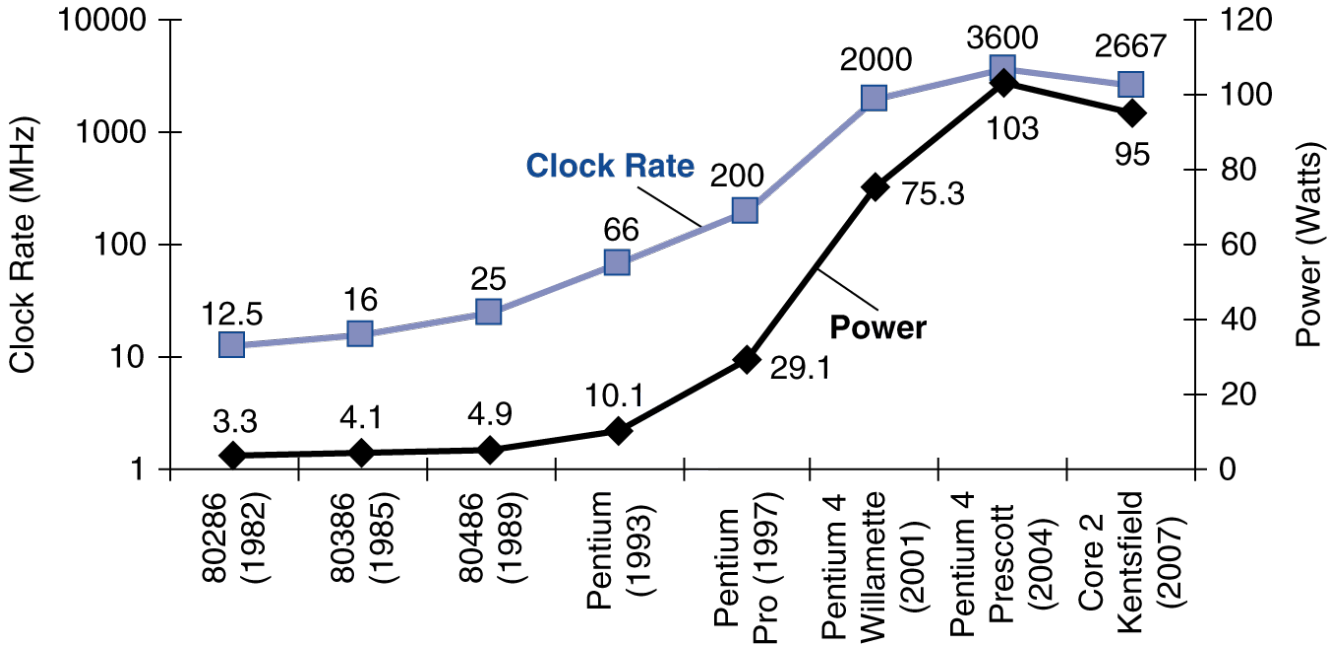
- Suppose we enhance a machine making all floating-point instructions run FIVE times faster. If the execution time of some benchmark before the floating-point enhancement is 10 seconds, what will the speedup be if only half of the 10 seconds is spent executing floating-point instructions?

  5/5 + 5 = 6 Relative Perf = 10/6 = 1.67 x

- We are looking for a benchmark to show off the new floating-point unit described above, and want the overall benchmark to show at least a speedup of 3. What percentage of the execution time would floating-point instructions have to account for in this program in order to yield our desired speedup on this benchmark?

  100/3 = f/5 + (100 – f) = 100 – 4f/5 f = 83.33

Power Trends



- In CMOS IC technology

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

×30 | 5V → 1V | ×1000

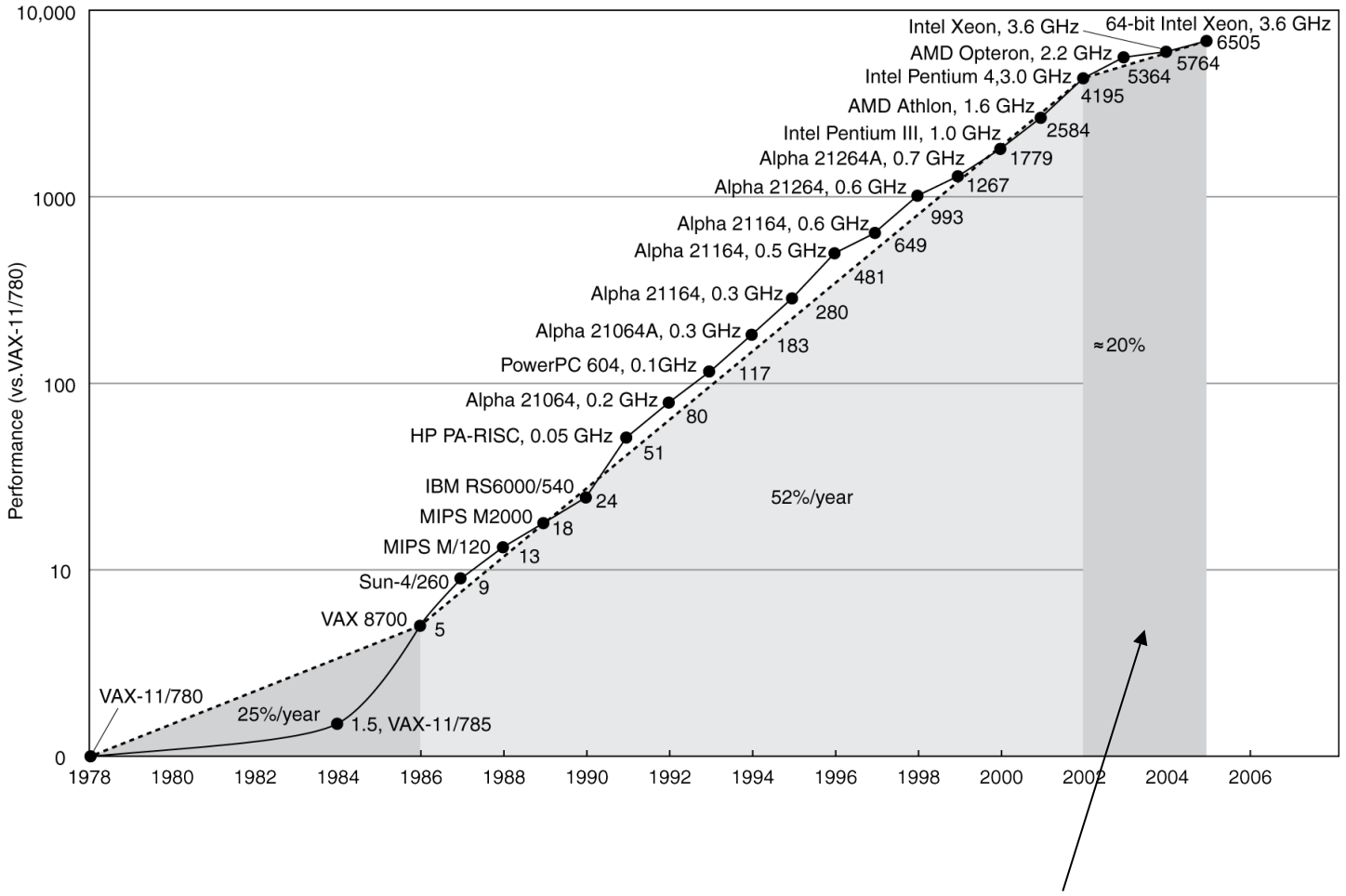
Reducing Power

- Suppose a new CPU has
  - 85% of capacitive load of old CPU
  - 15% voltage and 15% frequency reduction

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

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

Uniprocessor Performance



Constrained by power, instruction-level parallelism, memory latency

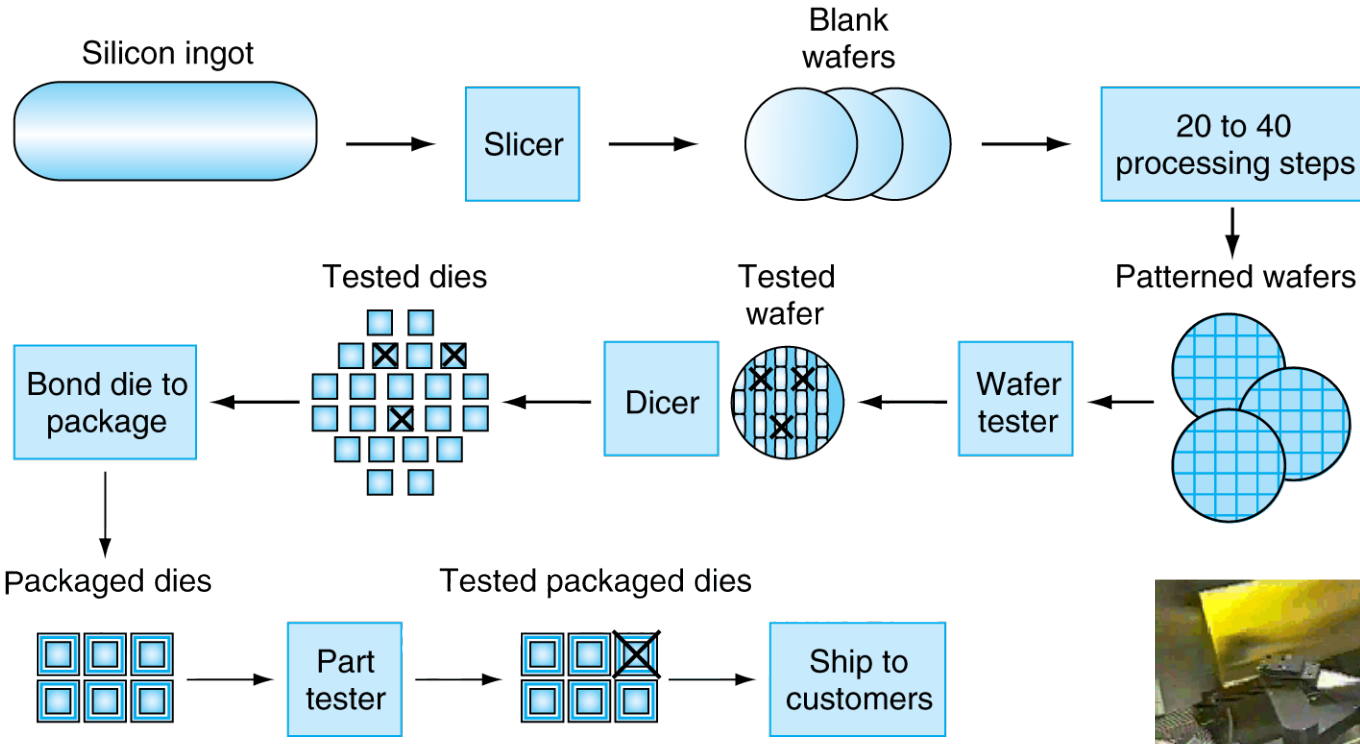
Multiprocessors

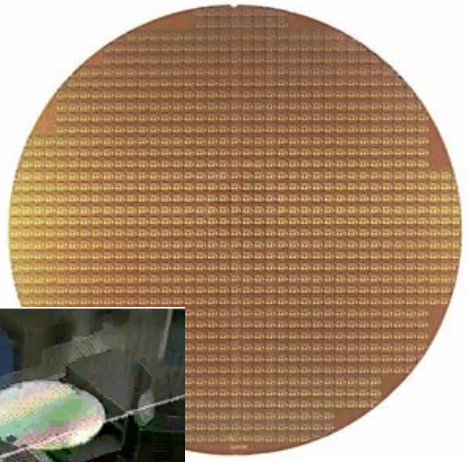
- "Multicore" microprocessors
  - More than one processor core per chip
- Requires explicitly parallel programming
  - Hardware executes multiple instructions at once
    - Ideally, hidden from the programmer
  - Hard to do
    - Programming for performance
    - Load balancing
    - Optimizing communication and synchronization
    - But, newer OSes and libraries have been designed for this

Manufacturing ICs

- IC = Integrated Circuit ("chip")

300mm (~12") diameter wafer





- Yield: proportion of working dies per wafer

Integrated Circuit Cost

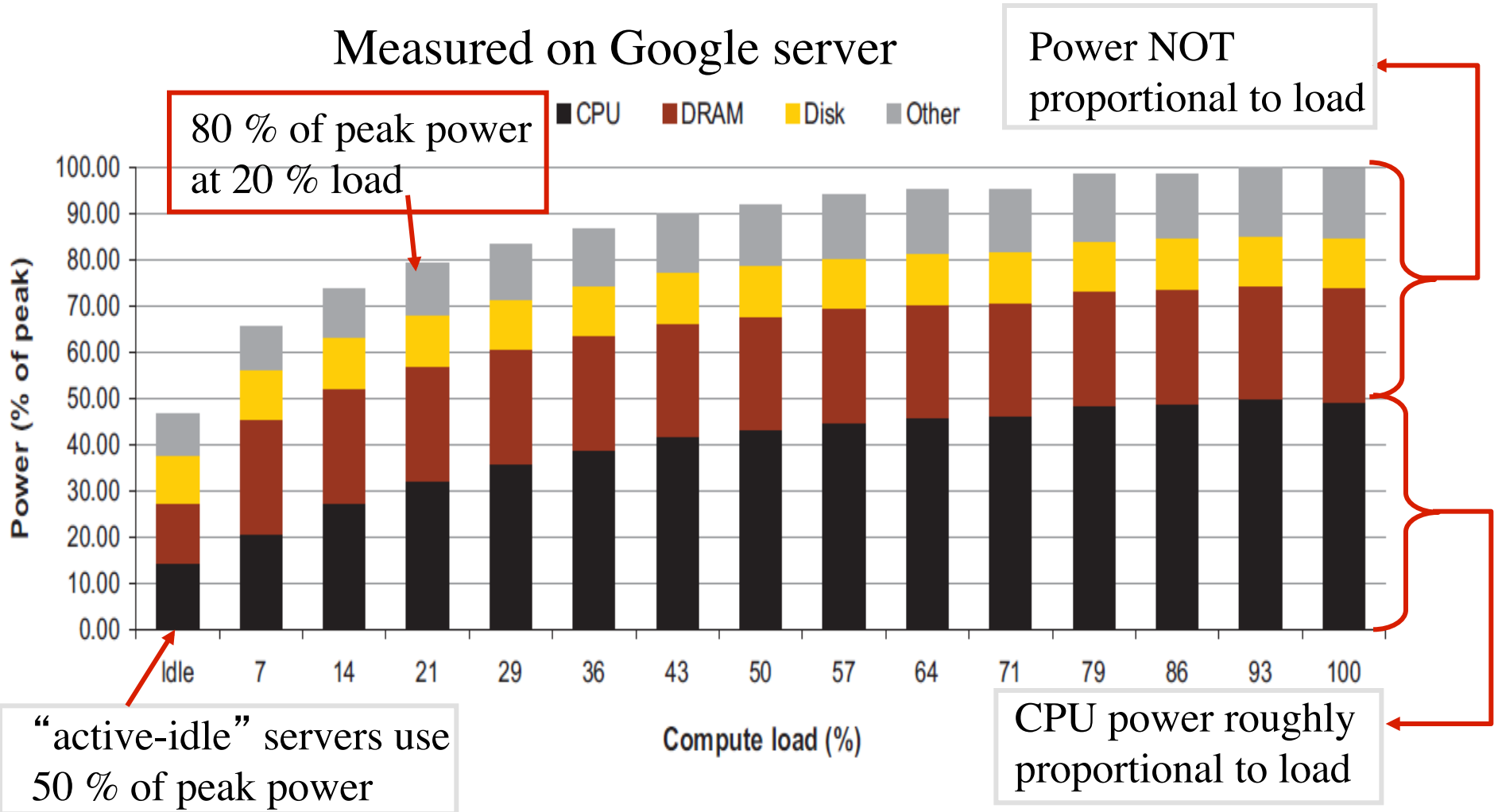
$$\text{Cost per die} = \frac{\text{Cost per wafer}}{\text{Dies per wafer} \times \text{Yield}}$$

$$\text{Dies per wafer} \approx \text{Wafer area}/\text{Die area}$$

$$\text{Yield} = \frac{1}{(1 + (\text{Defects per area} \times \text{Die area} / 2))^2}$$

- Nonlinear relation to area and defect rate
  - Wafer cost and area are fixed
  - Defect rate determined by manufacturing process
  - Die area determined by architecture and circuit design

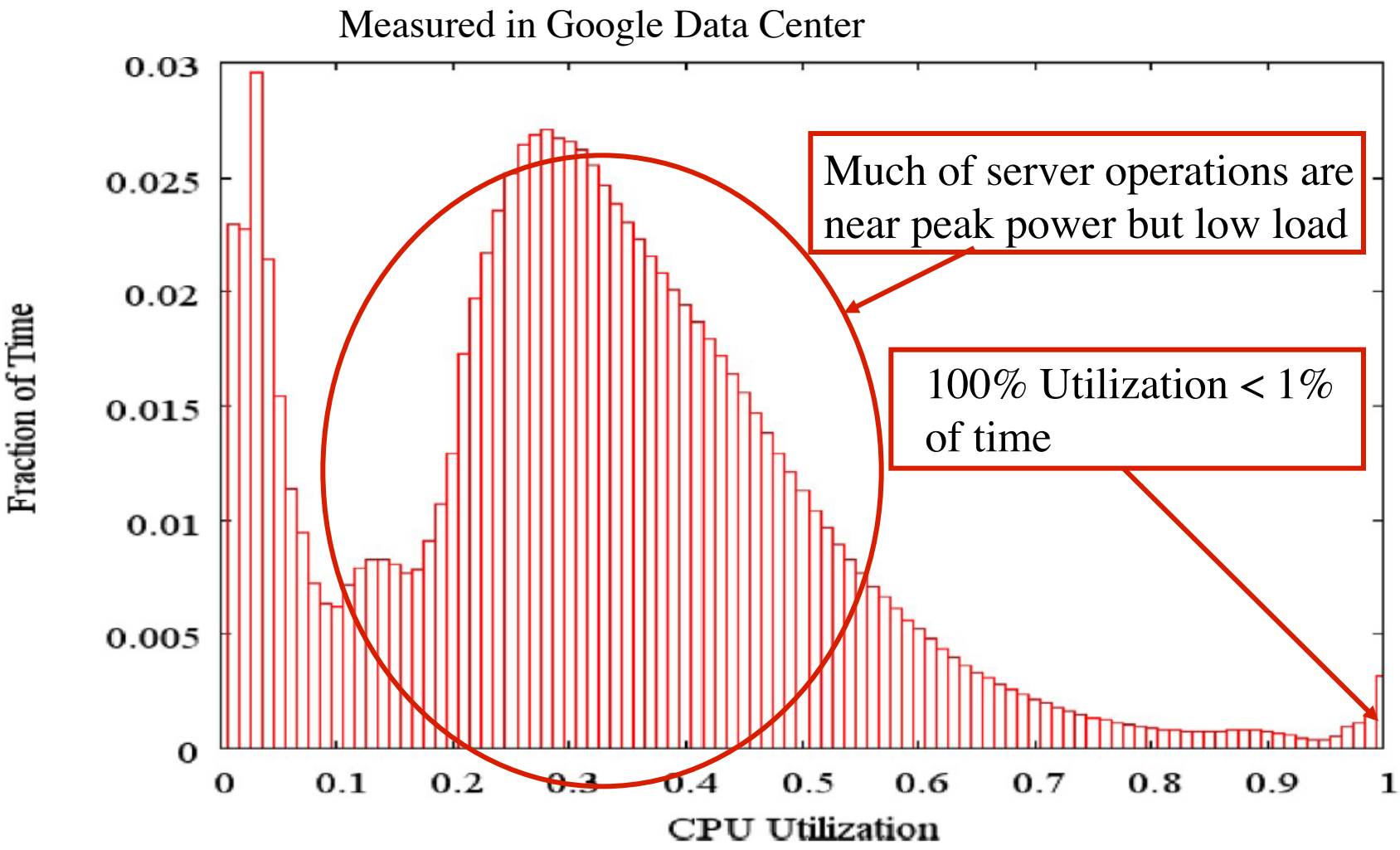
Fallacy: Low Power at Idle (1 of 2)



Barroso and Holzle, *The Datacenter as a Computer*, Synthesis Lectures on Computer Architecture, Morgan & Claypool, 2009, pp. 59

Industry challenge: Design processors to make power proportional to load

Fallacy: Low Power at Idle (2 of 2)



Industry challenge: Design processors to make power proportional to load

Barroso and Holzle, *The Datacenter as a Computer*, Synthesis Lectures on Computer Architecture, Morgan & Claypool, 2009, pp. 55

Remember

- Performance is specific to a particular program
  - Total execution time is a consistent summary of performance
- For a given architecture performance comes 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 improvements in one aspect of a machine's performance to affect the total performance
- You should not always believe everything you read!
  - Read carefully! Especially what the business guys say!

Concluding Remarks

- Cost/performance is improving
  - Due to underlying technology development
- Hierarchical layers of abstraction
  - In both hardware and software
- Instruction set architecture
  - The hardware/software interface
- Execution time: the best performance measure
- Power is a limiting factor
  - Use parallelism to improve performance