Computer Architecture Overview

'22H2

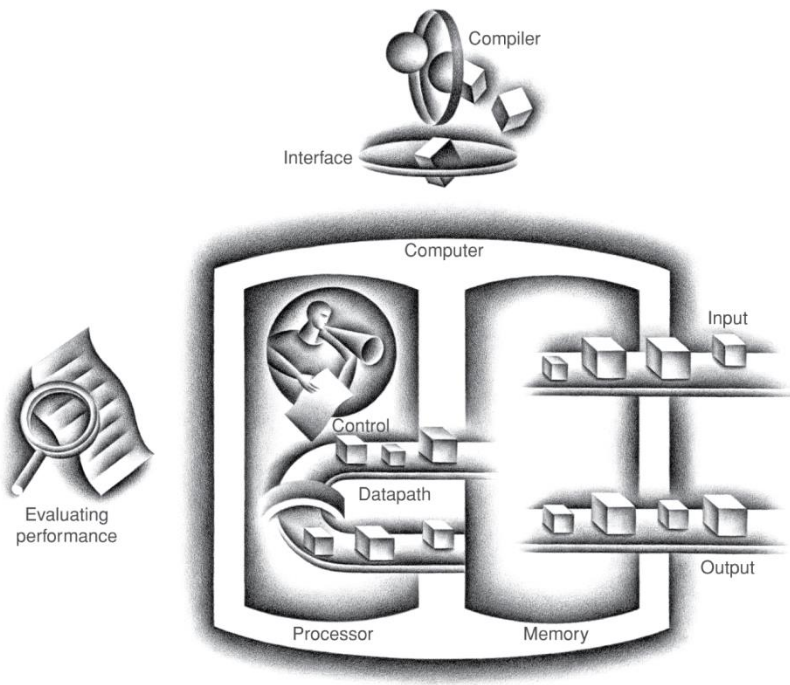
송 인 식

Outline

- Evolution of Computers
- Abstractions
- Design Principles
- Performance

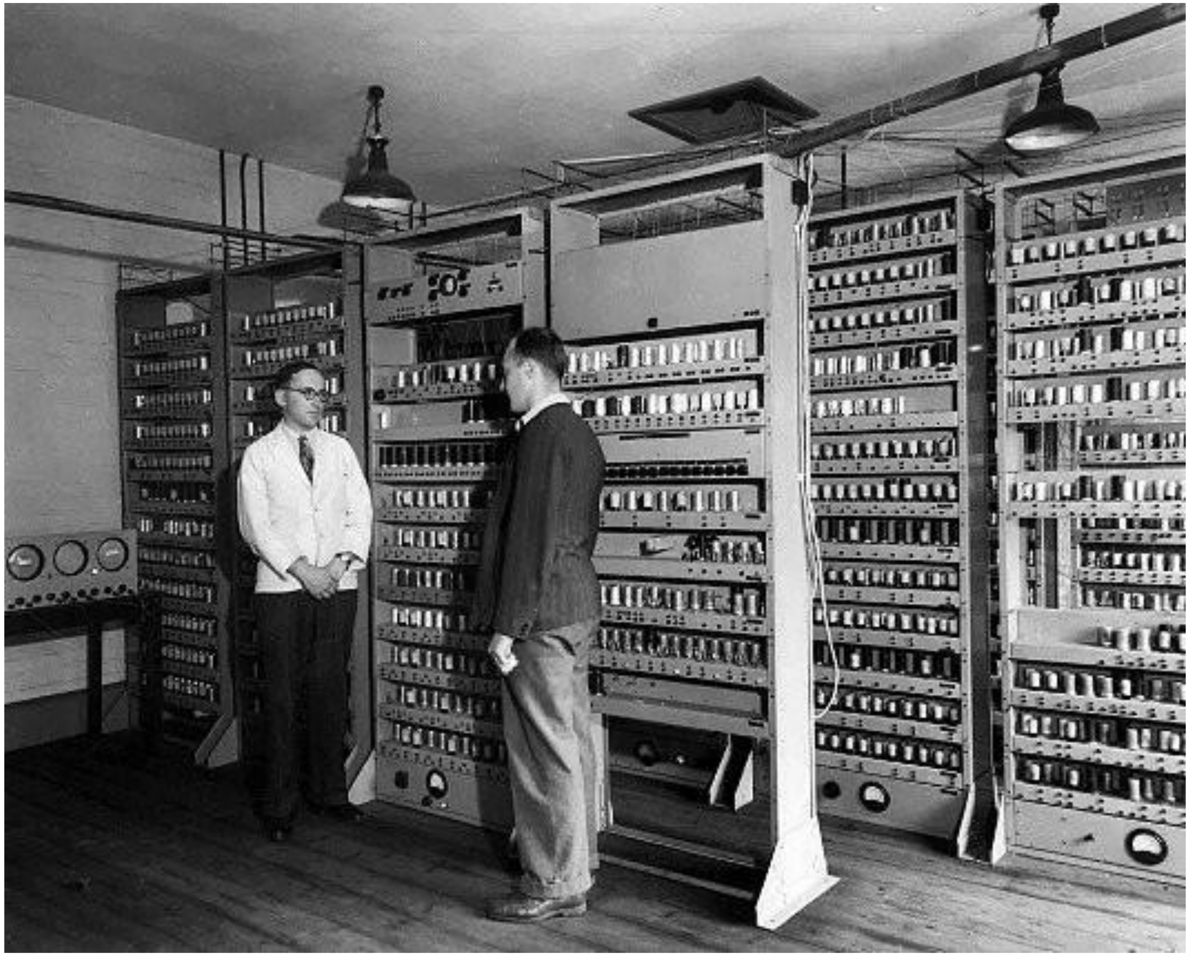
Components of a Computer

The BIG Picture



- Same components for all kinds of computer
  - Desktop, Server, Embedded
- Input/Output includes
  - User-interface devices
    - Display, keyboard, mouse
  - Storage devices
    - Hard disk, CD/DVD, flash
  - Network adapters
    - For communicating with other computers

The 1st Generation Computer



Source: http://www.computerhistory.org

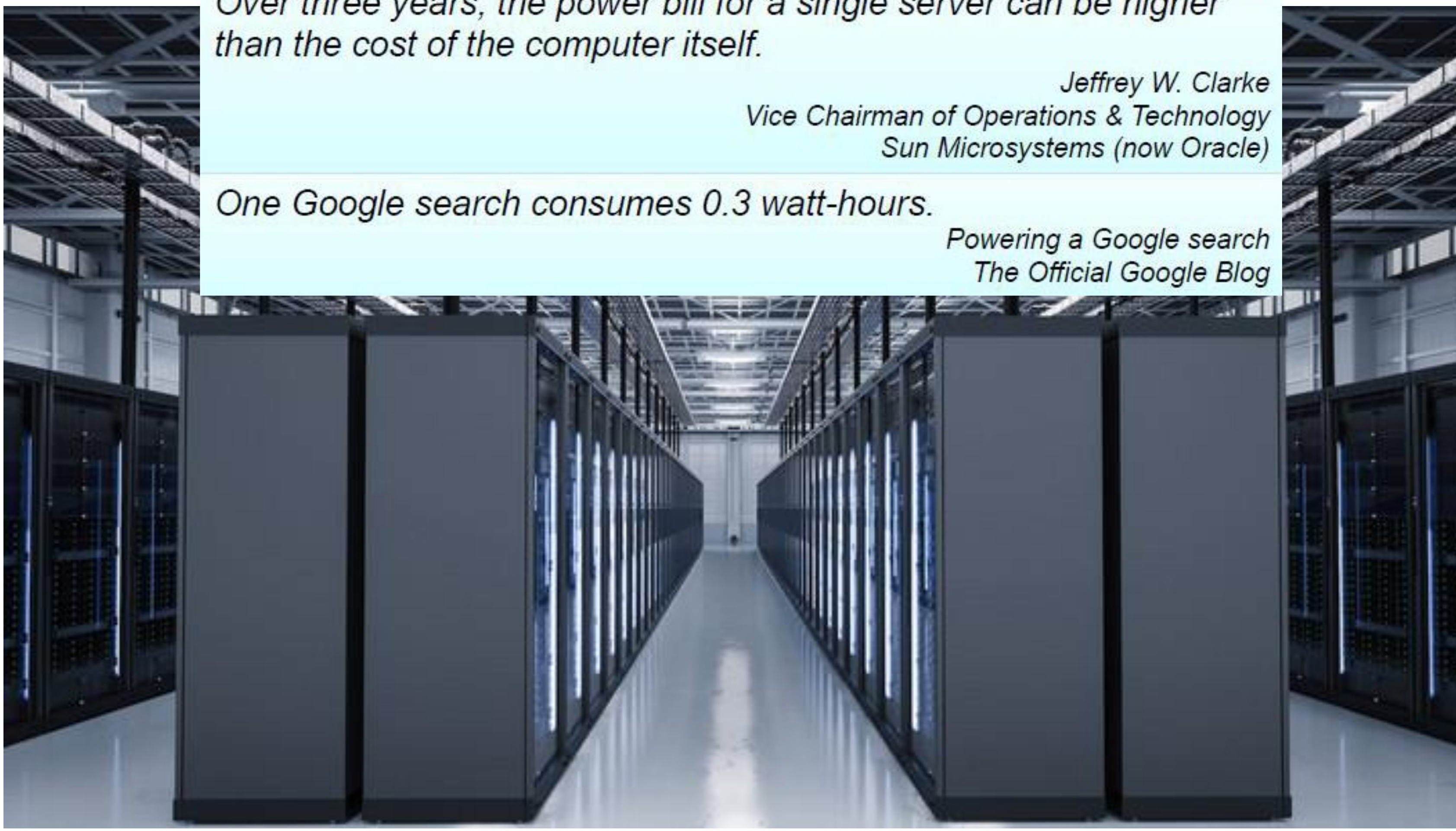
Data Center

Over three years, the power bill for a single server can be higher than the cost of the computer itself.

Jeffrey W. Clarke
Vice Chairman of Operations & Technology
Sun Microsystems (now Oracle)

One Google search consumes 0.3 watt-hours.

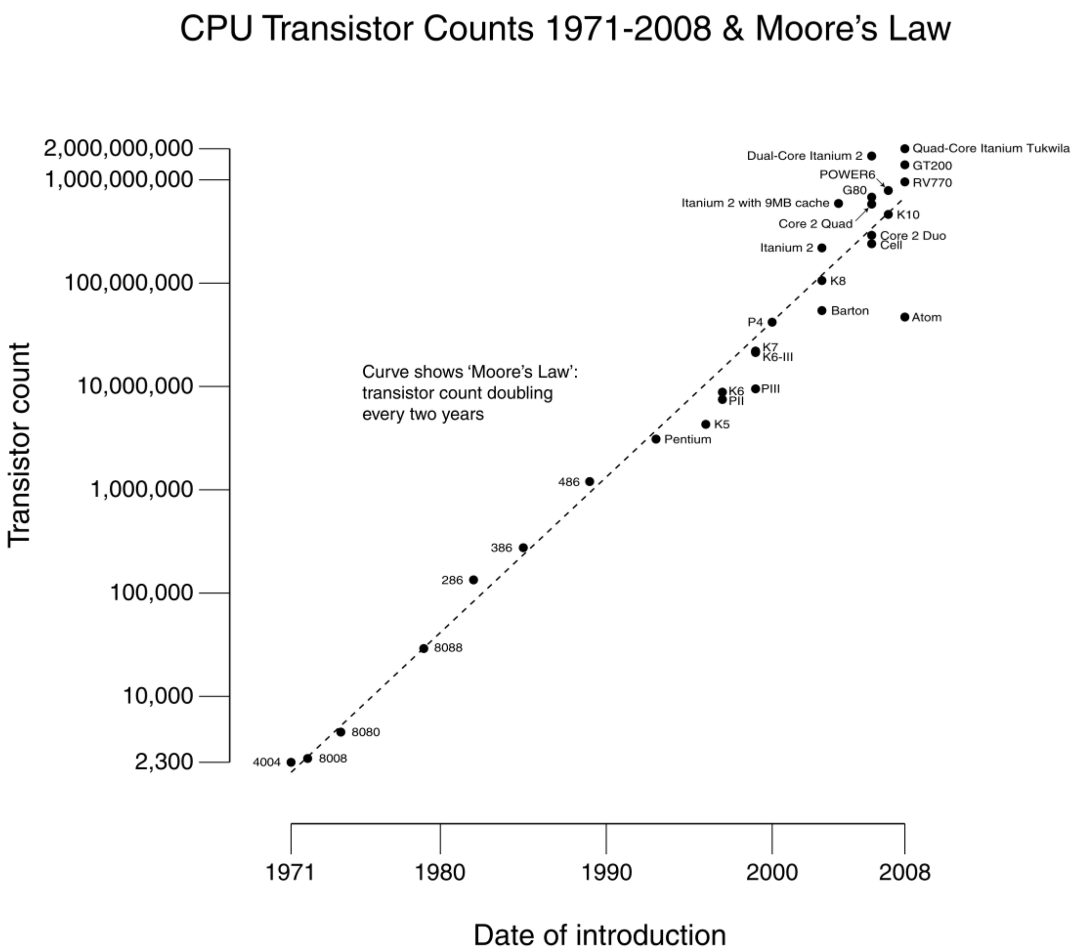
Powering a Google search
The Official Google Blog



The Computer Revolution

- Progress in computer technology
  - Underpinned by Moore's Law
- Makes novel applications feasible
  - World Wide Web (WWW)
  - Smartphones
  - Search engines
  - Human genome project
  - Self-driving cars
  - Artificial intelligence
  - VR/AR
- Computers are pervasive

Moore's Law

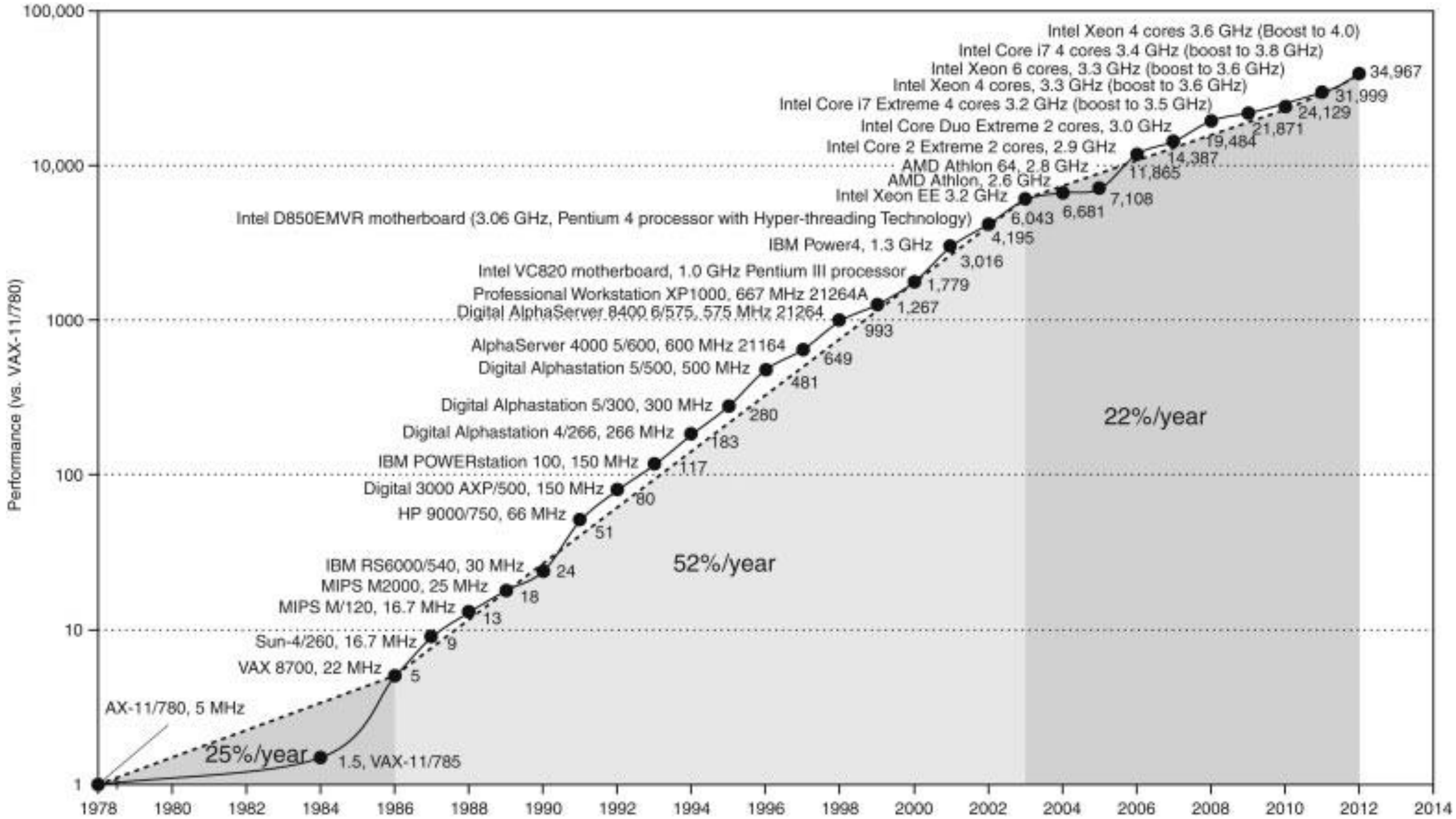


"The number of transistors incorporated in a chip will approximately double every 24 months."

Gordon Moore, Intel Co-founder

(1965)

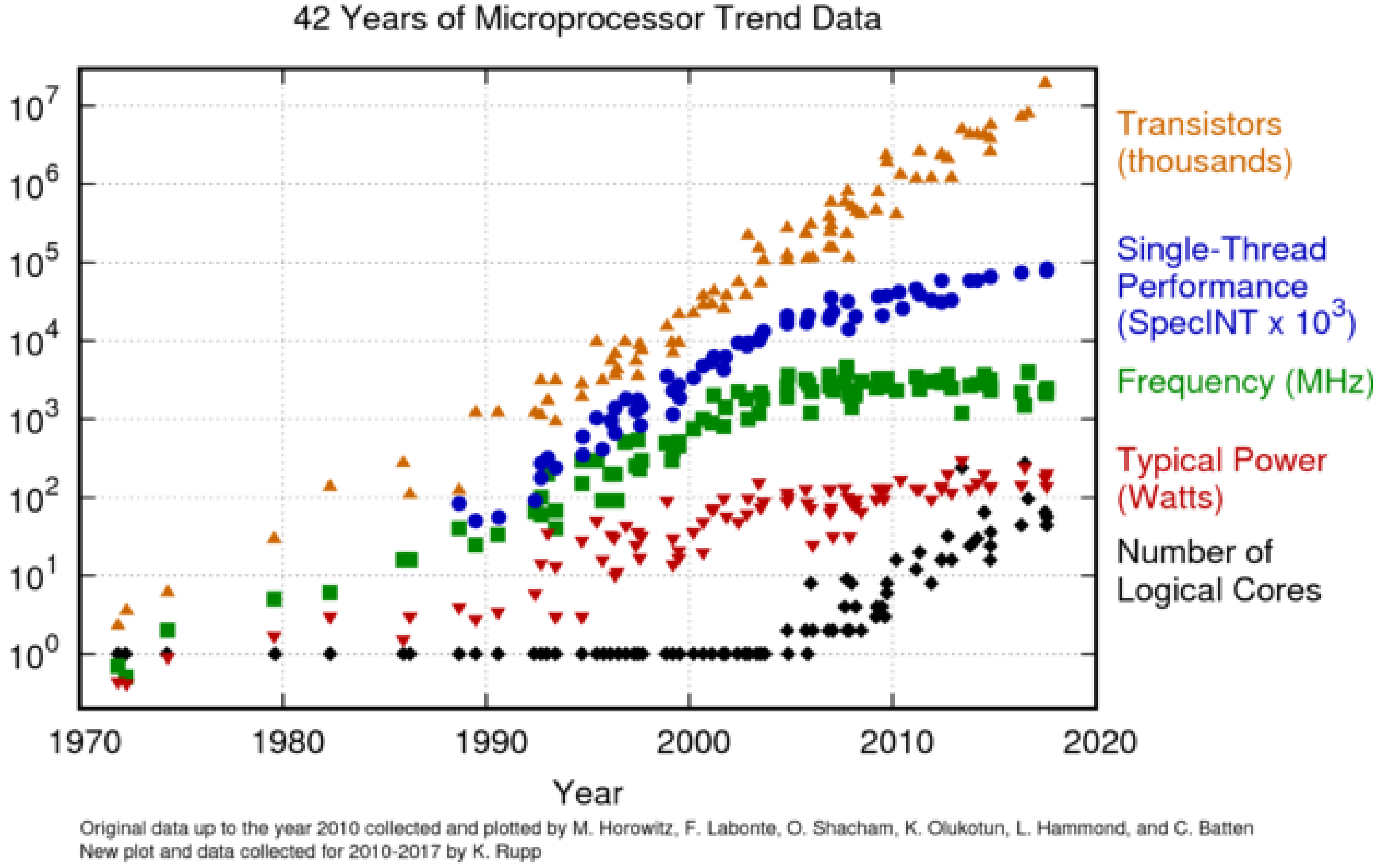
Microprocessor Performance



50% improvement every year!!
What contributes to this improvement?

Source: H&P

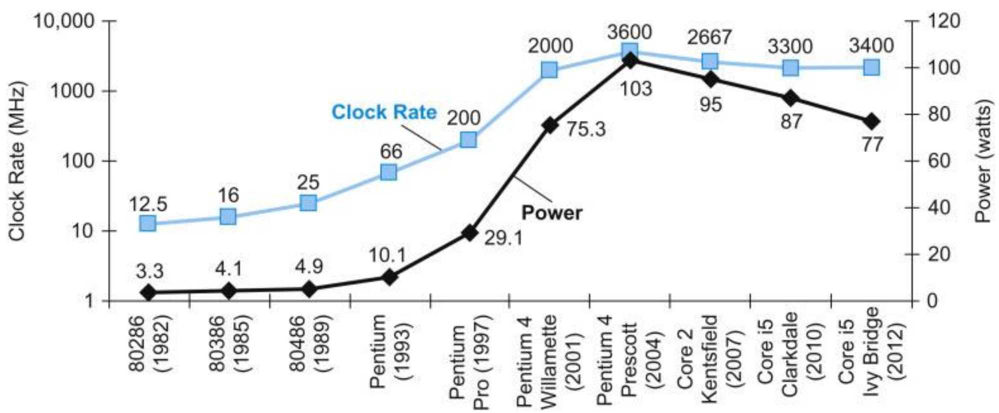
Microprocessor Performance



Source: karlrupp.net

Power Consumption Trends

- Dyn. power ∝ activity x capacitance x voltage^2 x frequency
- Voltage and frequency are somewhat constant now, while capacitance per transistor is decreasing and number of transistors (activity) is increasing
- Leakage power is also rising (function of #trans and voltage)

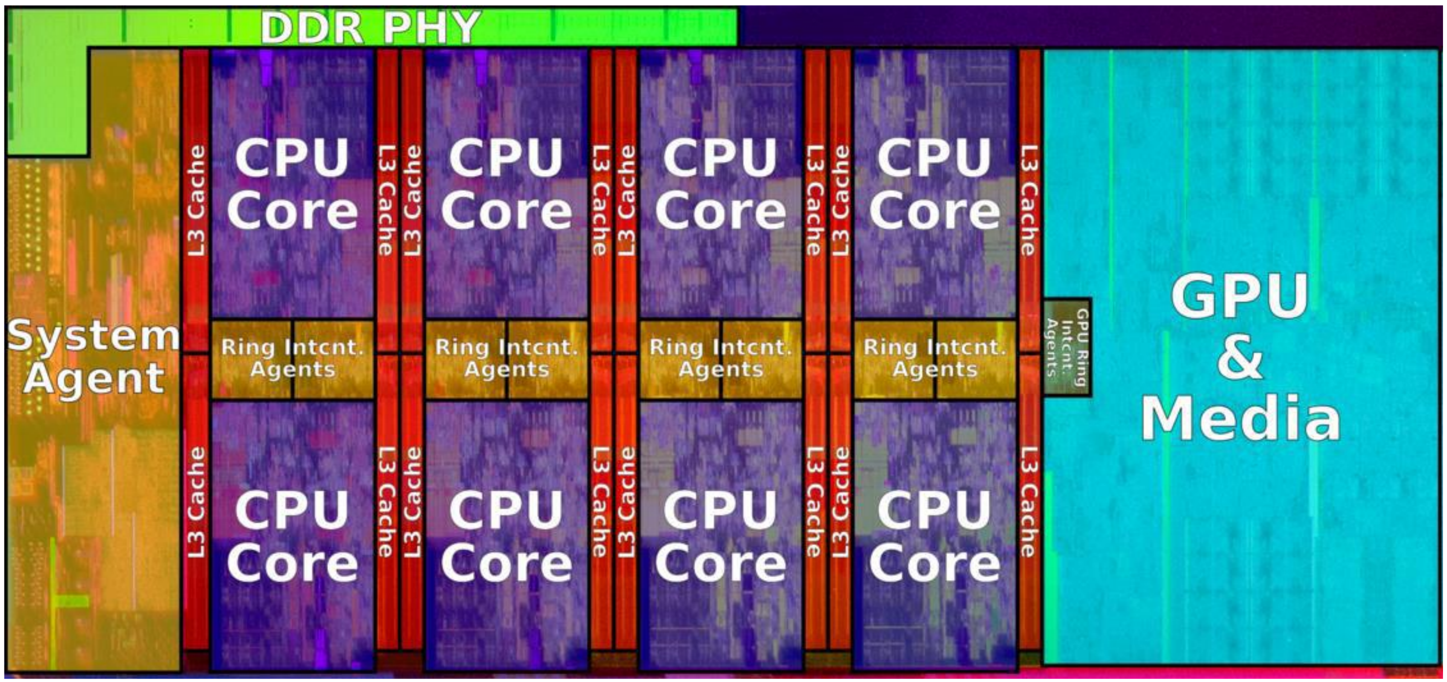


Source: H&P

Important Trends

- Running out of ideas to improve single thread performance
- Power wall makes it harder to add complex features
- Power wall makes it harder to increase frequency
- Additional performance provided by: more cores, occasional spikes in frequency, accelerators

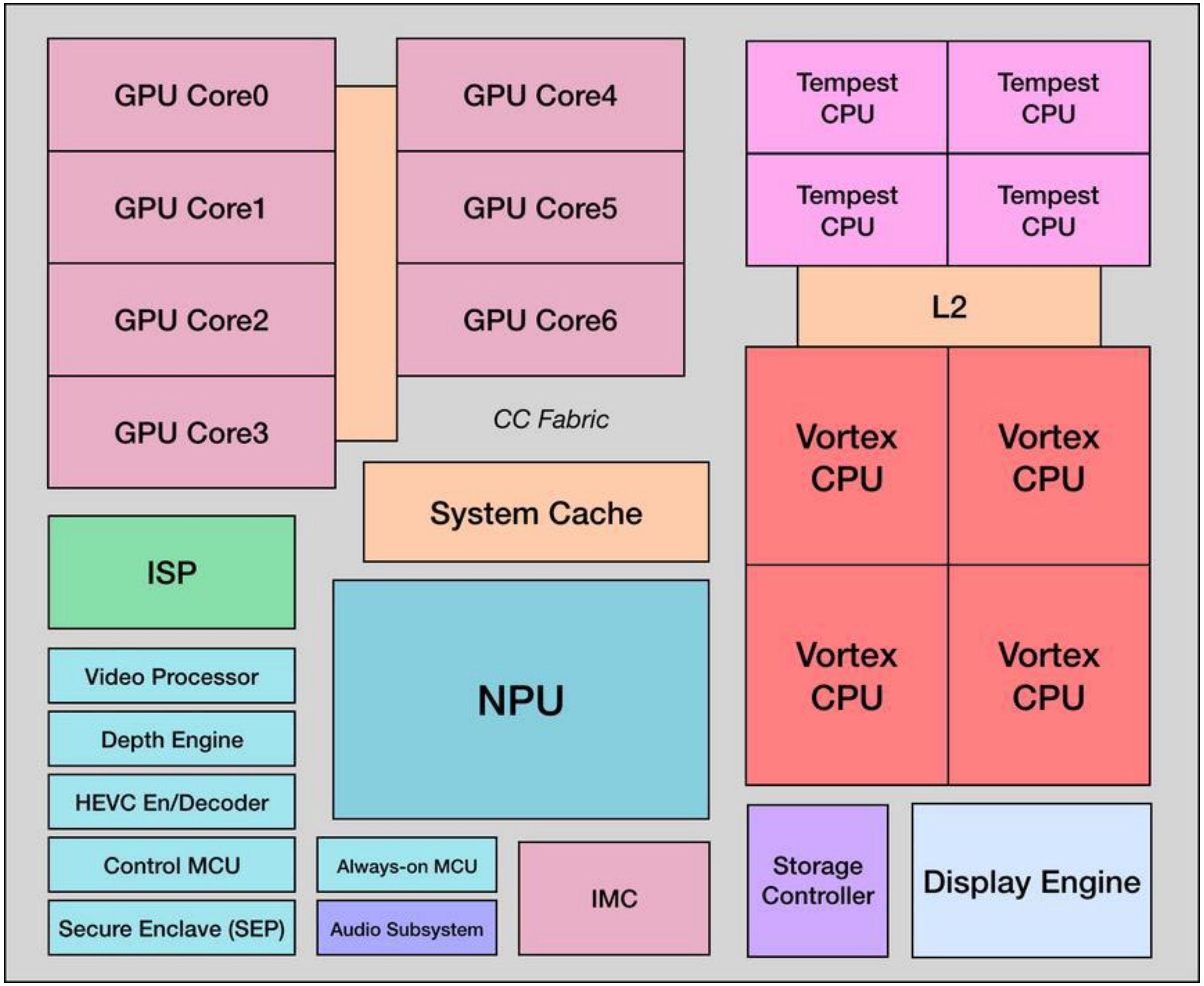
Intel Core i9-9900K (Coffee Lake, 2018)



Process:
14nm
Transistors:
~ 3B
Die size:
~ 177 mm²

Source: https://en.wikichip.org/wiki/intel/core_i9/i9-9900k

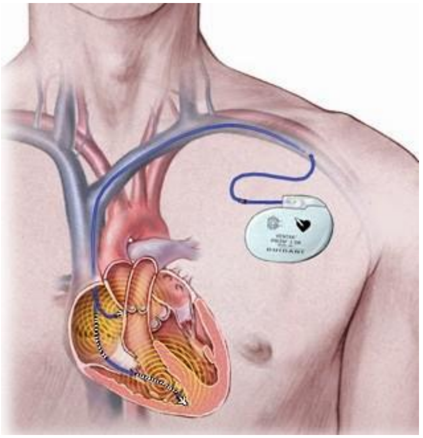
Apple A12X Bionic (2018)



Process:
7nm
Transistors:
~ 10B
Die size:
~ 122 mm²

Source: https://en.wikichip.org/wiki/apple/ax/a12x

Computers Today



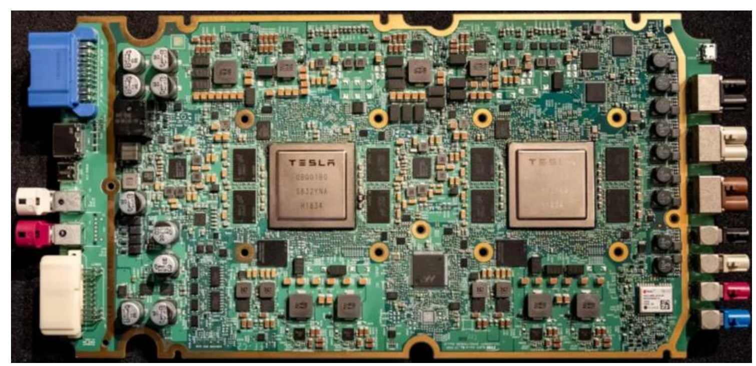
Source: http://www.computerhistory.org

Classes of Computers

- Personal computers
  - General purpose, variety of software
  - Subject to cost/performance tradeoff
- Server computers
  - Network based
  - High capacity, performance, reliability
  - Range from small servers to large data centers
- Supercomputers
  - High-end scientific and engineering calculations
  - Highest capability but represent a small fraction of the overall computer market
- Embedded computers
  - Hidden as components of systems
  - Stringent power/performance/cost constraints

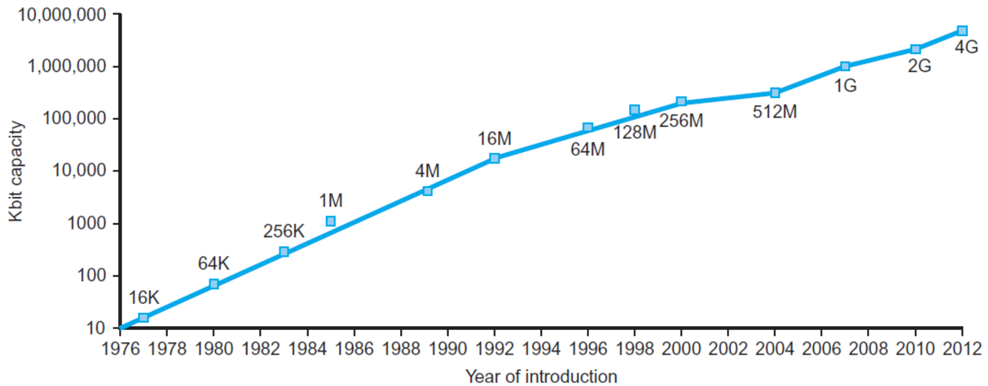
Different Platforms, Different Goals

Different Platforms, Different Goals



Technology Trends

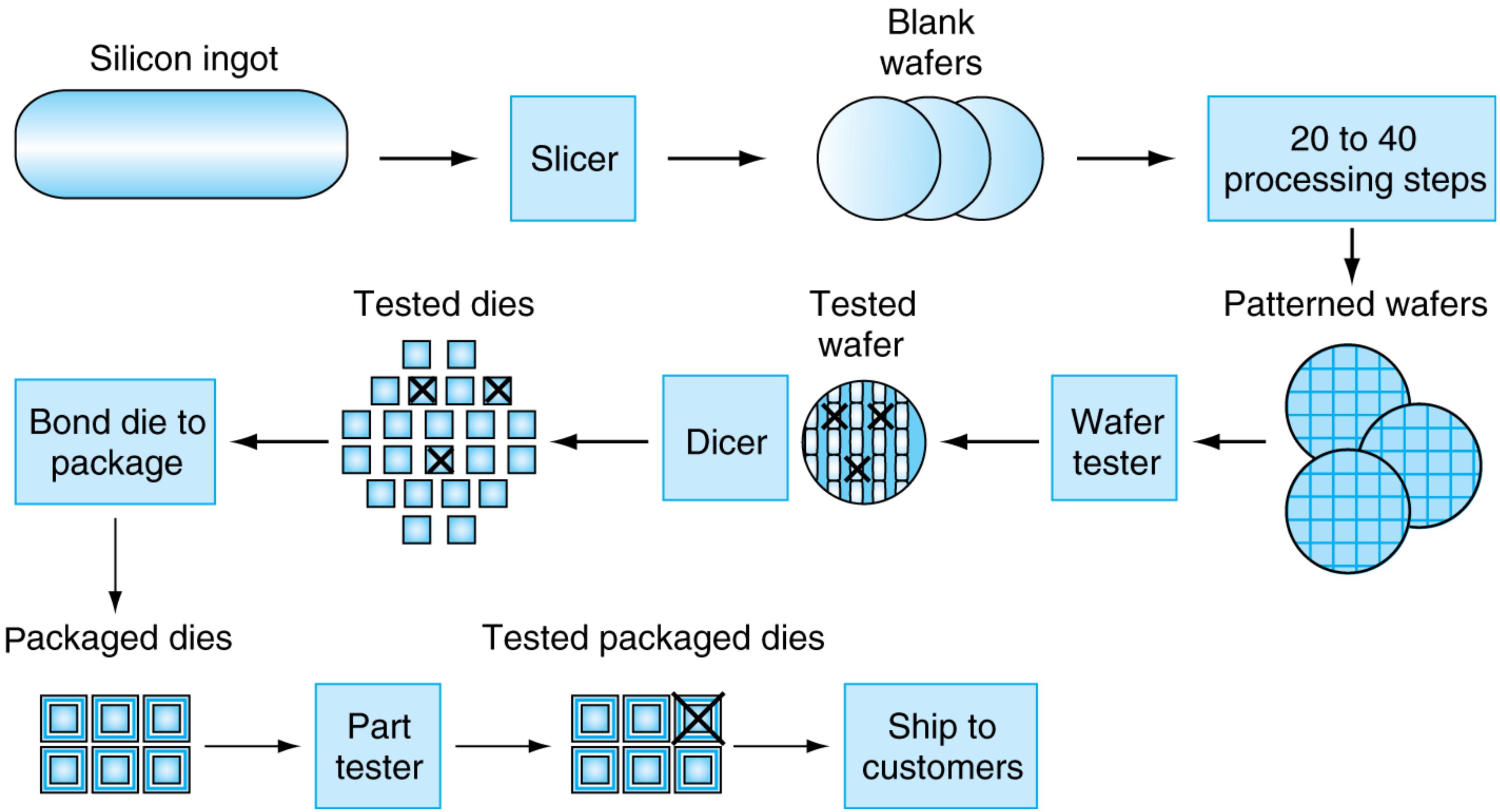
- Electronics technology continues to evolve
  - Increased capacity and performance
  - Reduced cost



DRAM capacity per chip over time

Year	Technology	Relative performance/cost
1951	Vacuum tube	1
1965	Transistor	35
1975	Integrated circuit (IC)	900
1995	Very large scale IC (VLSI)	2,400,000
2013	Ultra large scale IC	250,000,000,000

Manufacturing ICs



- Yield: proportion of working dies per wafer

Outline

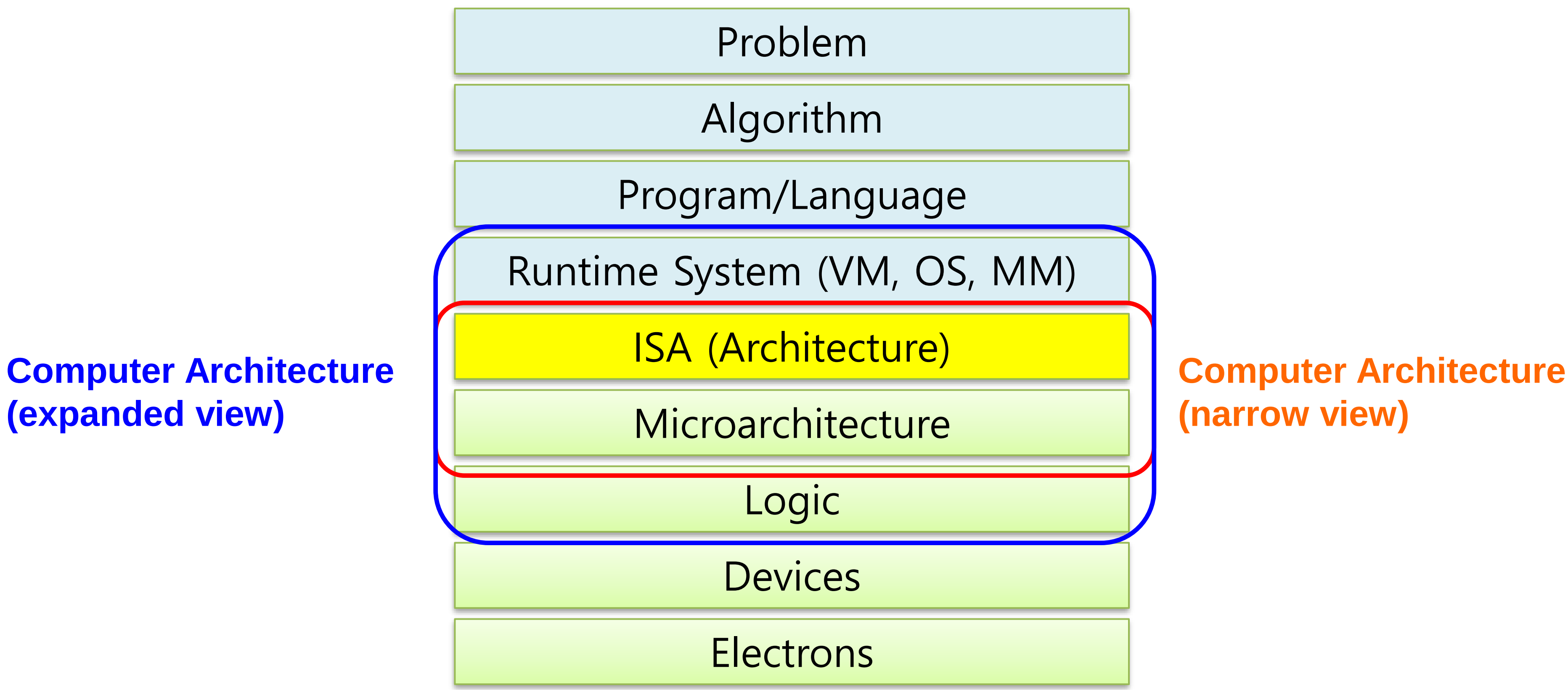
- Evolution of Computers
- Abstractions
- Design Principles
- Performance

Abstractions

The BIG Picture

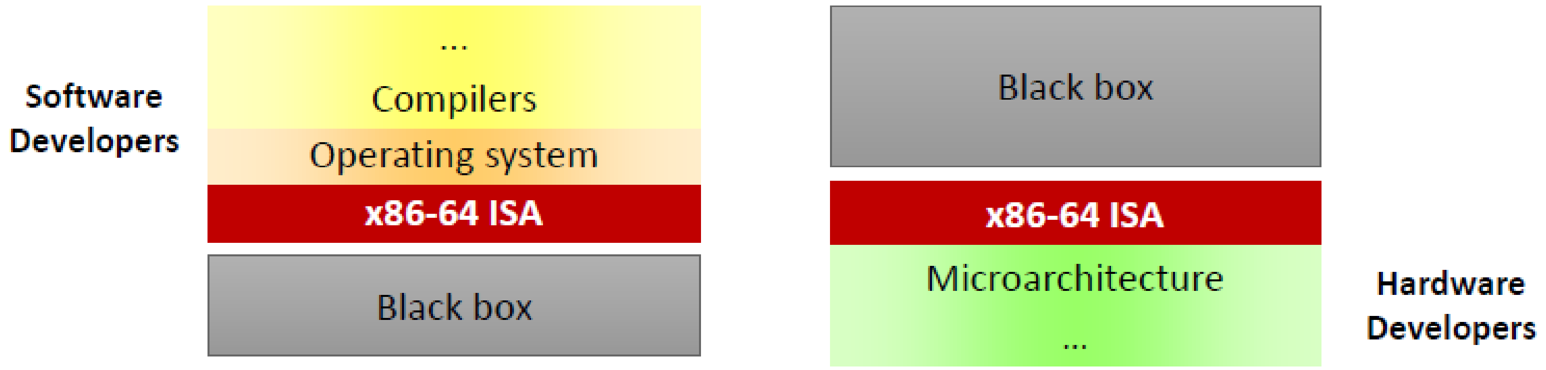
- Abstraction helps us deal with complexity
  - Hide lower-level detail
- Instruction set architecture (ISA)
  - The hardware/software interface
- Application binary interface
  - The ISA plus system software interface
- Implementation
  - The details underlying and interface

Levels of Abstractions



Instruction Set Architecture

- The hardware/software interface
  - Hardware abstraction visible to software (OS, compilers, …)
  - Instructions and their encodings, registers, data types, addressing modes, etc.
  - Written documents about how the CPU behaves
  - e.g. All 64-bit Intel CPUs follow the same x86-64 (or Intel 64) ISA



Abstraction is Good, But …

- Abstraction helps us deal with complexity
  - Hide lower-level details
  - E.g. Abstract data types, Asymptotic analysis
- These abstractions have limits
  - Especially in the presence of bugs
  - Need to understand details of underlying implementations
- This is why you should take this course seriously even if you don’t want to be a computer architect!

Outline

- Evolution of Computers
- Abstractions
- Design Principles
- Performance

Eight Great Design Ideas

- Design for ***Moore's Law***
  - Computer designs can take years, resources available per chip can easily double or quadruple between start and finish of project
  - Anticipate where technology will be when design finishes

- Use ***Abstraction*** to simplify design
  - Hide lower-level details to offer a simpler model at higher levels
- Make the ***Common Case Fast***
  - To enhance performance better than optimizing the rare case

Eight Great Design Ideas (2)

- Performance *via* ***Parallelism***
  - A form of computation in which many calculations are carried out simultaneously

- Performance *via* ***Pipelining***
  - A particular pattern of parallelism
  - A set of data processing elements connected in series, so that the output of one element is the input of the next one
- Performance *via* ***Prediction***
  - It can be faster on average to guess and start working rather than wait

Eight Great Design Ideas (3)

- ***Hierarchy*** of memories
  - The closer to the top, the faster and more expensive per bit of memory
  - The wider the base of the layer, the bigger the memory
- ***Dependability*** *via* redundancy
  - Design systems dependable by including redundant components
    - to take over when failure occurs, and
    - to help detect failures

Outline

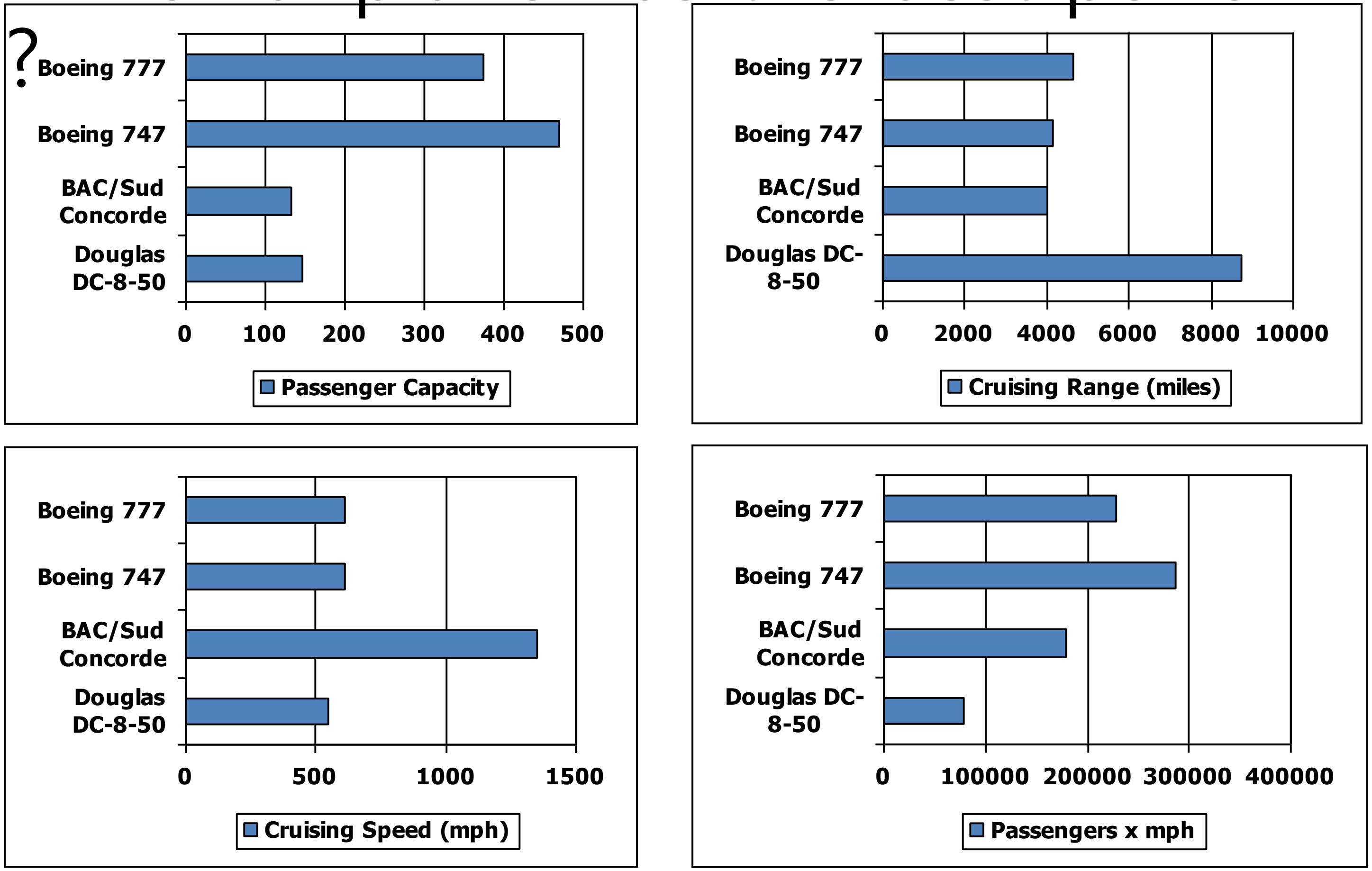
- Evolution of Computers
- Abstractions
- Design Principles
- Performance

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)
  - determine how fast I/O operations are executed

Defining Performance

- Which airplane has the best performance?



Response Time and Throughput

- Response time
  - How long it takes to do a task
- Throughput
  - Total work done per unit time

    e.g., tasks/transactions/… per hour
- How are response time and throughput affected by
  - Replacing the processor with a faster version?
  - Adding more processors?
- We'll focus on response time for now…

Relative Performance

- Define Performance = 1/Execution Time
- "X is n time faster than Y"

$$
\begin{aligned}
&\text{Performance}_X / \text{Performance}_Y \\
&= \text{Execution time}_Y / \text{Execution time}_X = n
\end{aligned}
$$

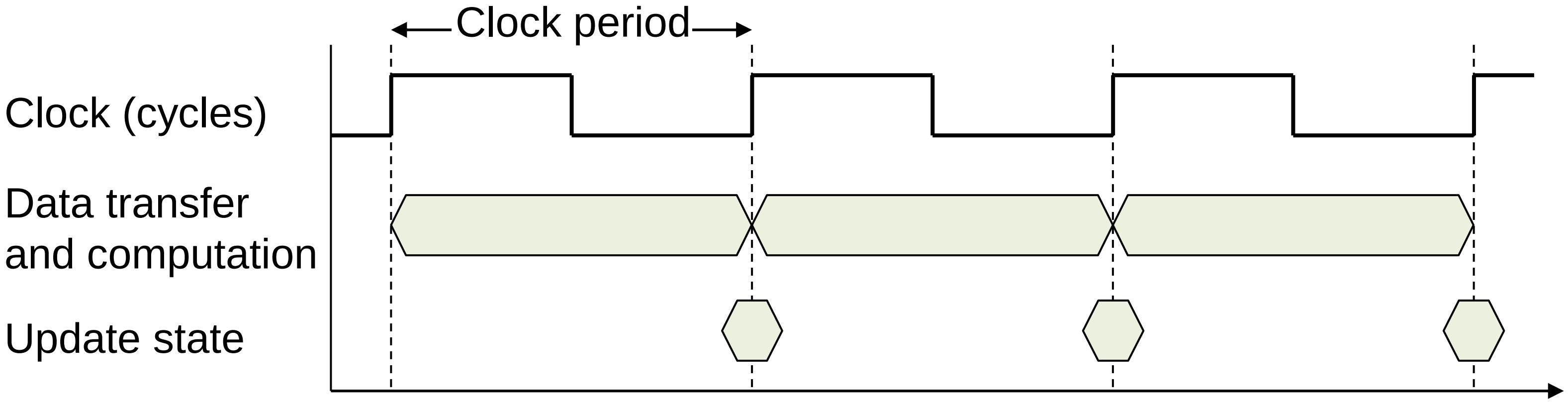
- Example: time taken to run a program
  - 10s on A, 15s on B
  - Execution Time_B / Execution Time_A
    = 15s / 10s = 1.5
  - So A is 1.5 times faster than B

Measuring Execution Time

- Elapsed time
  - Total response time, including all aspects
    - Processing, I/O, OS overhead, idle time
  - Determines system performance
- CPU time
  - Time spent processing a given job
    - Minus I/O time, other jobs' shares
  - Includes user CPU time and system CPU time
  - Different programs are affected differently by CPU and system performance
    - Running on servers–I/O performance–hardware and software
    - Total elapsed time is of interest
    - Define performance metric and then proceed

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?

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

$$\begin{aligned}\text{Clock Cycles}_A &= \text{CPU Time}_A \times \text{Clock Rate}_A \\ &= 10\text{s} \times 2\text{GHz} = 20 \times 10^9\end{aligned}$$

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

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 gets affected by instruction mix (dynamic frequency of instructions)

CPI Example

- Computer A: Cycle Time = 250ps, CPI = 2.0
- Computer B: Cycle Time = 500ps, CPI = 1.2
- Same ISA
- Which is faster? 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

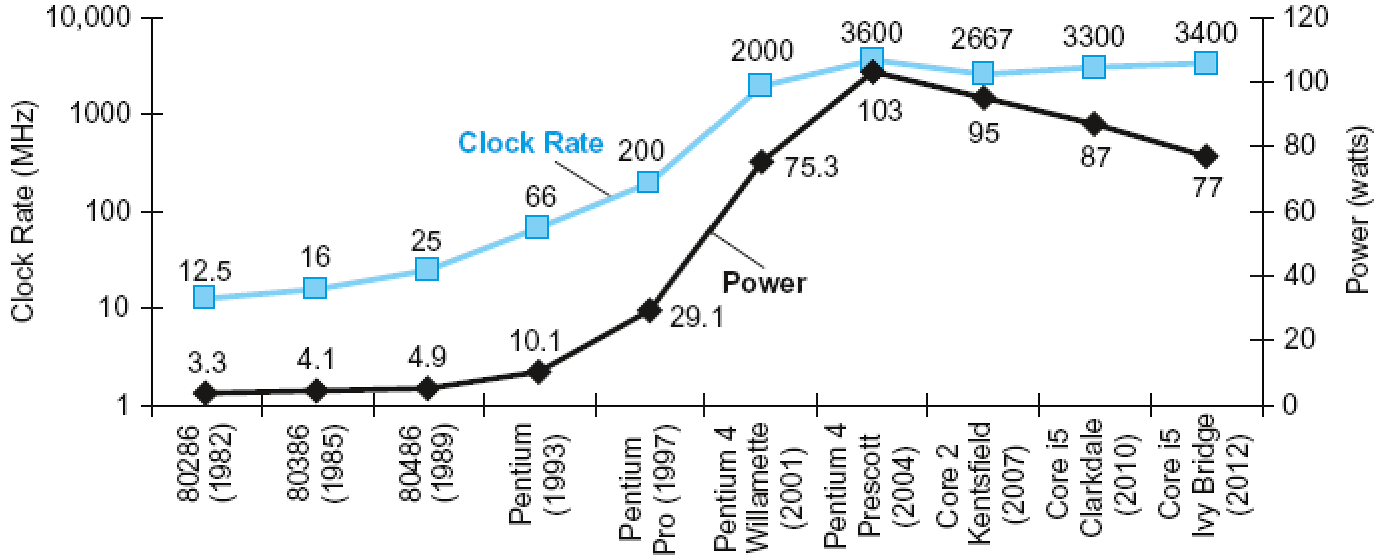
Performance Summary

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

Power Trends



Clock rate and Power for Intel x86 microprocessors over eight generations

- In CMOS IC technology

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

×30

5V → 1V

×1000

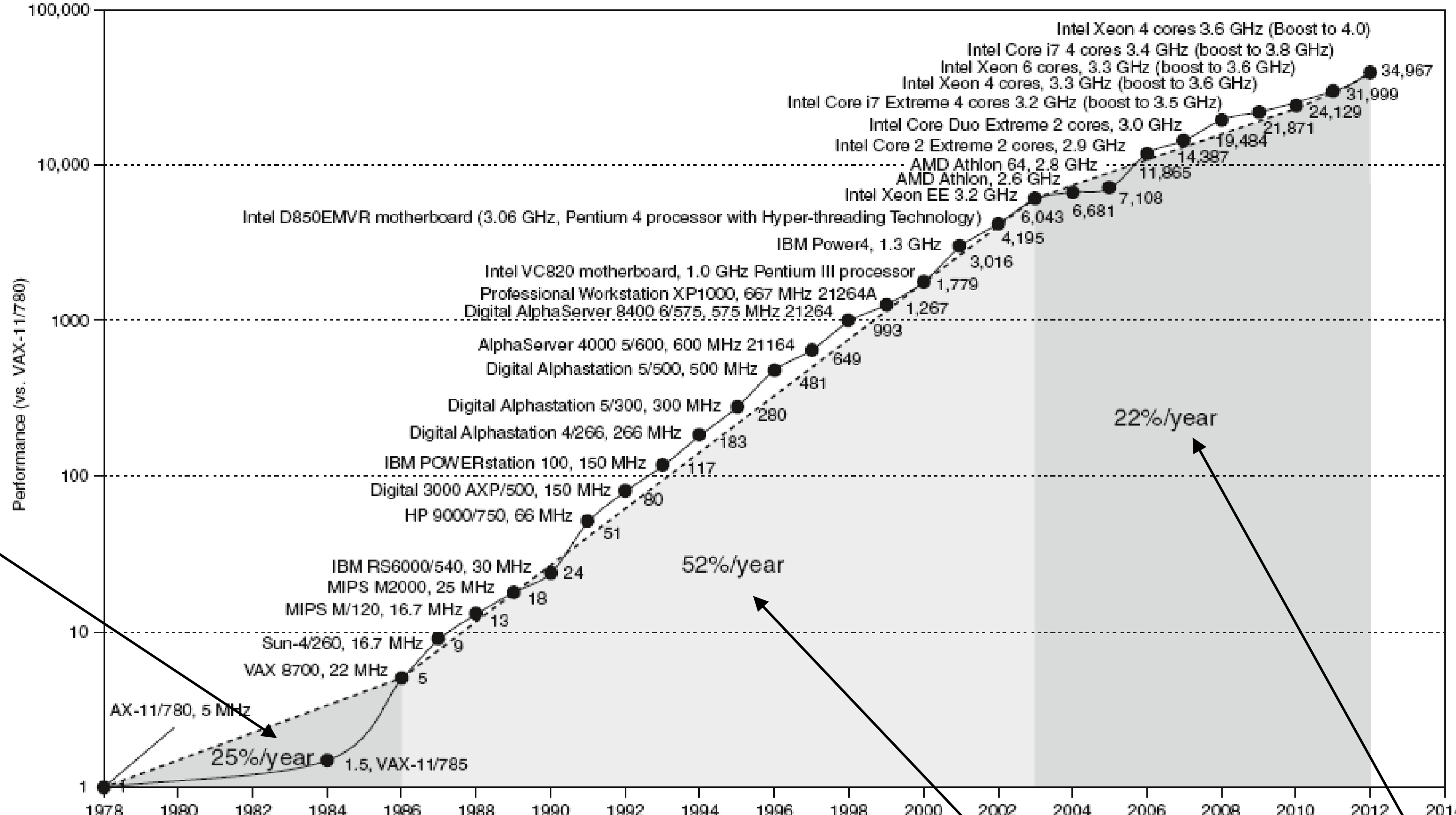
Reducing Power

- 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 can't reduce voltage further
  - We can't remove more heat
- How else can we improve performance?

Uniprocessor Performance



Technology driven

Advanced architectural and organizational ideas

Constrained by power, instruction-level parallelism, memory latency`

Multiprocessors

- Multicore microprocessors
  - More than one processor per chip
- Requires explicitly parallel programming
  - Compare with instruction level parallelism
    - Hardware executes multiple instructions at once
    - Hidden from the programmer
  - Hard to do (Why?)
    - Programming for performance
    - Load balancing
    - Optimizing communication and synchronization

Benchmarks

- How to measure the performance?
  - Performance best determined by running a real application
  - Use programs typical of expected workload
- Small benchmarks
  - Nice for architects and designers
  - Easy to standardize
  - Can be abused

SPEC CPU Benchmark

- Programs used to measure performance
  - Supposedly typical of actual workload
- Standard Performance Evaluation Corp (SPEC)
  - Develops benchmarks for CPU, I/O, Web, …
- 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 Intel Core i7 920

Description	Name	Instruction Count x 10^9	CPI	Clock cycle time (seconds x 10^{-9})	Execution Time (seconds)	Reference Time (seconds)	SPECratio
Interpreted string processing	perl	2252	0.60	0.376	508	9770	19.2
Block-sorting compression	bzip2	2390	0.70	0.376	629	9650	15.4
GNU C compiler	gcc	794	1.20	0.376	358	8050	22.5
Combinatorial optimization	mcf	221	2.66	0.376	221	9120	41.2
Go game (AI)	go	1274	1.10	0.376	527	10490	19.9
Search gene sequence	hmmer	2616	0.60	0.376	590	9330	15.8
Chess game (AI)	sjeng	1948	0.80	0.376	586	12100	20.7
Quantum computer simulation	libquantum	659	0.44	0.376	109	20720	190.0
Video compression	h264avc	3793	0.50	0.376	713	22130	31.0
Discrete event simulation library	omnetpp	367	2.10	0.376	290	6250	21.5
Games/path finding	astar	1250	1.00	0.376	470	7020	14.9
XML parsing	xalancbmk	1045	0.70	0.376	275	6900	25.1
Geometric mean	–	–	–	–	–	–	25.7

SPEC Power Benchmark

- Power consumption of server at different workload levels
  - Performance: ssj_ops/sec
  - Power: Watts (Joules/sec)

$$\text{Overall ssj\_ops per Watt} = \left(\sum_{i=0}^{10} \text{ssj\_ops}_i\right) \bigg/ \left(\sum_{i=0}^{10} \text{power}_i\right)$$

SPECpower_ssj2008 for Xeon X5650

Target Load %	Performance (ssj_ops)	Average Power (Watts)
100%	865,618	258
90%	786,688	242
80%	698,051	224
70%	607,826	204
60%	521,391	185
50%	436,757	170
40%	345,919	157
30%	262,071	146
20%	176,061	135
10%	86,784	121
0%	0	80
Overall Sum	4,787,166	1,922
Σssj_ops/Σpower =		2,490

Pitfall: Amdahl's Law

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

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

- Example: multiply accounts for 80s/100s
  - How much improvement in multiply performance to get 5× overall?

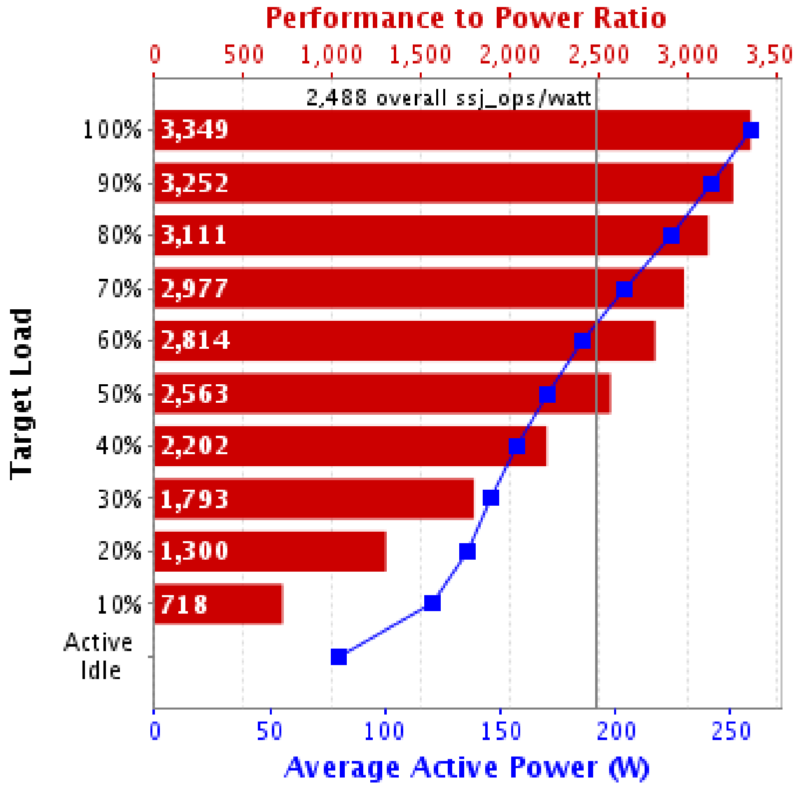
$$20 = \frac{80}{n} + 20$$

Can't be done!

- Corollary: make the common case fast

Pitfall: Amdahl's Law

- Look back at Xeon power benchmark
  - At 100% load: 258W
  - At 50% load: 170W (66%)
  - At 10% load: 121W (47%)
- Google data center
  - Mostly operates at 10% - 50% load
  - At 100% load less than 1% of the time
- Consider designing processors to make power proportional to load



Pitfall: MIPS as a Performance Metric

- MIPS: Millions of Instructions Per Second
  - Doesn't account for
    - Differences in ISAs between computers
    - Differences in complexity between instructions

$$\text{MIPS} = \frac{\text{Instruction count}}{\text{Execution time} \times 10^6}$$

$$= \frac{\text{Instruction count}}{\dfrac{\text{Instruction count} \times \text{CPI}}{\text{Clock rate}} \times 10^6} = \frac{\text{Clock rate}}{\text{CPI} \times 10^6}$$

- CPI varies between programs on a given CPU

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

Questions?