

# Computer Organization and Design

## Chapter 1

### Computer Abstractions and Technology

# The Computer Revolution

- Progress in computer technology
  - Underpinned by **Moore's Law**
- Makes novel applications feasible
  - Computers in automobiles
  - Cell phones
  - Human genome project
  - World Wide Web
  - Search Engines
- Computers are **pervasive**

# Classes of Computers

## ❑ Desktop and laptop computers

Designed to deliver good performance to a single user at low cost usually executing 3<sup>rd</sup> 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

## ❑ 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)

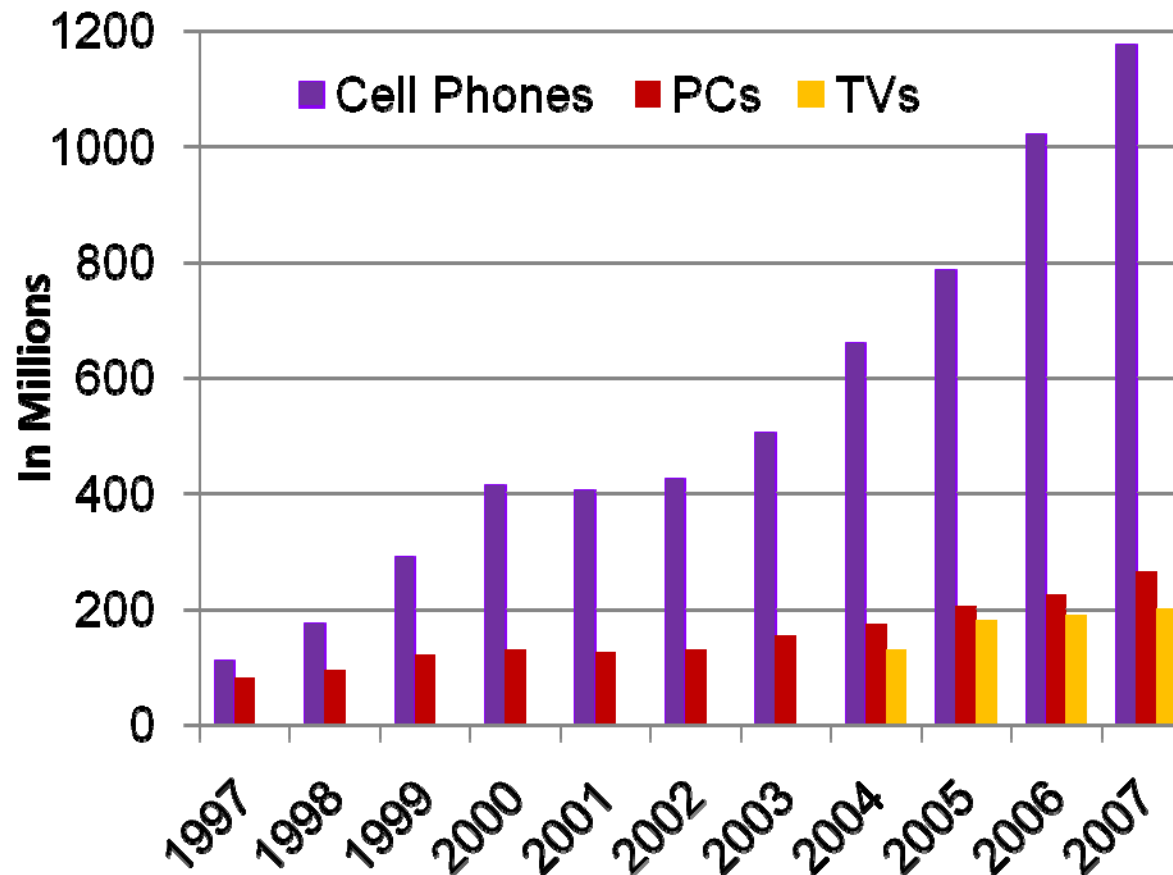
A computer inside another device used for running one predetermined application

# Review: Some Basic Definitions

- ❑ **Kilobyte** –  $2^{10}$  or 1,024 bytes
- ❑ **Megabyte** –  $2^{20}$  or 1,048,576 bytes
  - 1 sometimes “rounded” to  $10^6$  or 1,000,000 bytes
- ❑ **Gigabyte** –  $2^{30}$  or 1,073,741,824 bytes
  - 1 sometimes rounded to  $10^9$  or 1,000,000,000 bytes
- ❑ **Terabyte** –  $2^{40}$  or 1,099,511,627,776 bytes
  - 1 sometimes rounded to  $10^{12}$  or 1,000,000,000,000 bytes
- ❑ **Petabyte** –  $2^{50}$  or 1024 terabytes
  - 1 sometimes rounded to  $10^{15}$  or 1,000,000,000,000,000 bytes
- ❑ **Exabyte** –  $2^{60}$  or 1024 petabytes
  - 1 Sometimes rounded to  $10^{18}$  or 1,000,000,000,000,000,000 bytes

# The Processor Market

embedded growth >> desktop growth



- Where else are embedded processors found?

# Embedded Processor Characteristics

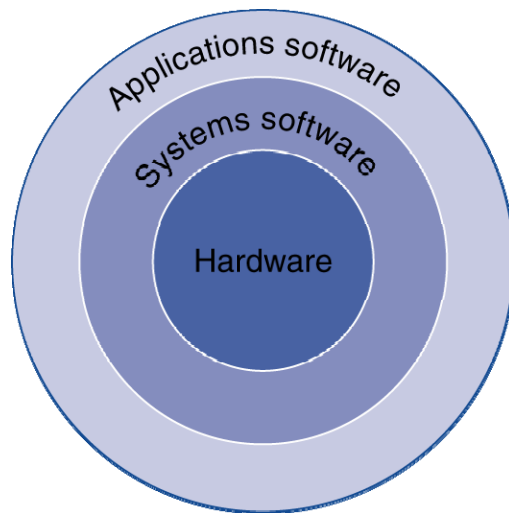
*The largest class of computers spanning the widest range of applications and performance*

- ❑ Often have minimum performance requirements. Example?
- ❑ Often have stringent limitations on cost. Example?
- ❑ Often have stringent limitations on power consumption. Example?
- ❑ Often have low tolerance for failure. Example?

# Understanding Performance

- **Algorithm**
  - Determines number of operations executed
- **Programming language, compiler, ISA**
  - 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

# Below Your Program



- Application software
  - Written in high-level language
- System software
  - Compiler: translates HLL code to machine code
  - Operating System: service code
    - Handling input/output
    - Managing memory and storage
    - Scheduling tasks & sharing resources
- Hardware
  - Processor, memory, I/O controllers



# Levels of Program Code

- High-level language program (in C)

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

one-to-many

C compiler

- 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
```

one-to-one

assembler

- Machine (object, binary) code (for MIPS)

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

# Advantages of Higher-Level Languages ?

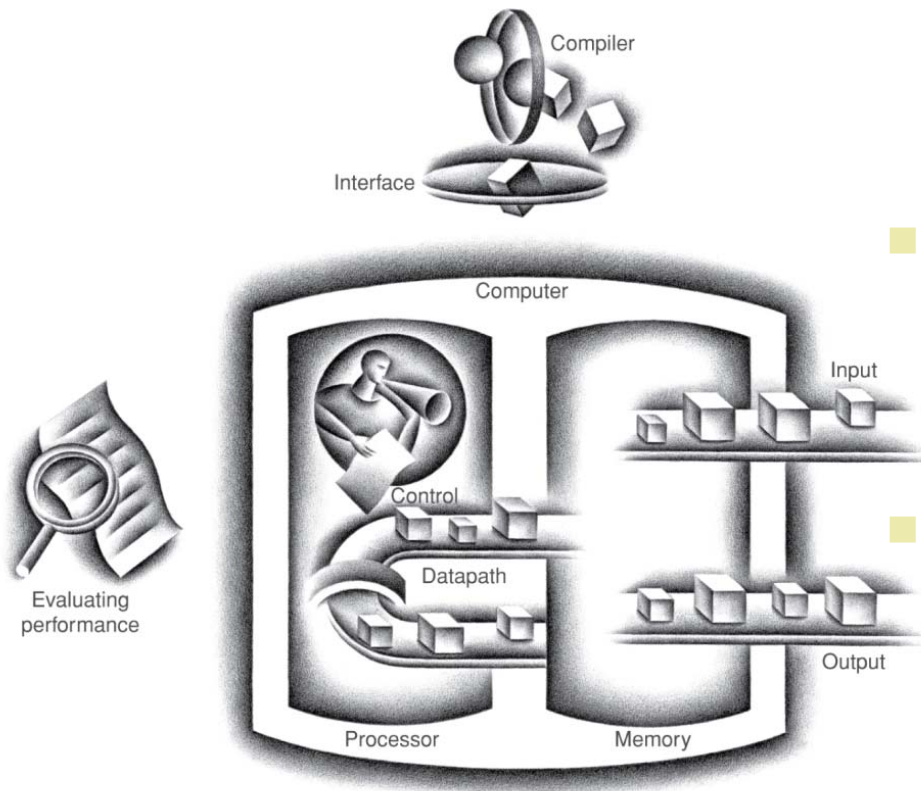
## □ Higher-level languages

- 1 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, ...)
- 1 Improve programmer **productivity** – more understandable code that is easier to debug and validate
- 1 Improve program **maintainability**
- 1 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)
- 1 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

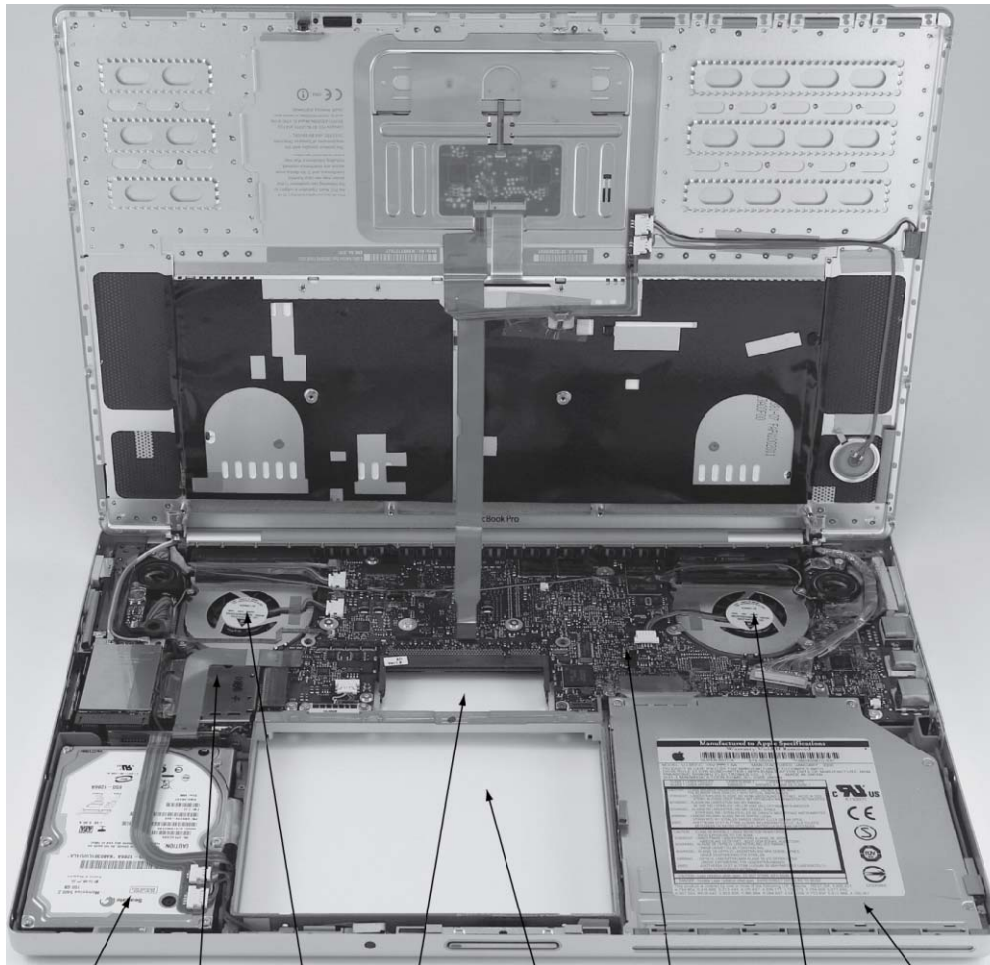
# Components of a Computer

## The BIG Picture

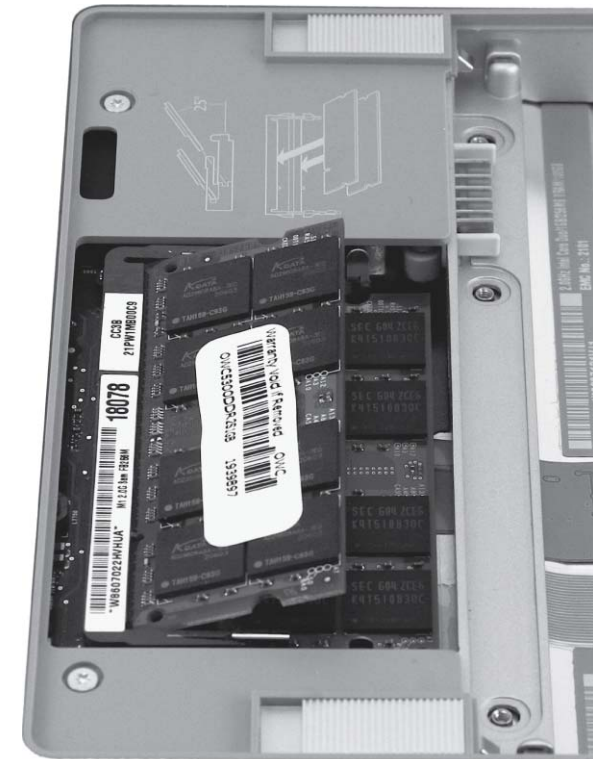


- Five main components
  - Input, output, memory, datapath, control
  - **Datapath + Control = Processor**
- Same components for all kinds of computers
  - Desktop, server, embedded
- Input/output includes
  - User-interface devices, storage devices, network adapters

# Opening the Box

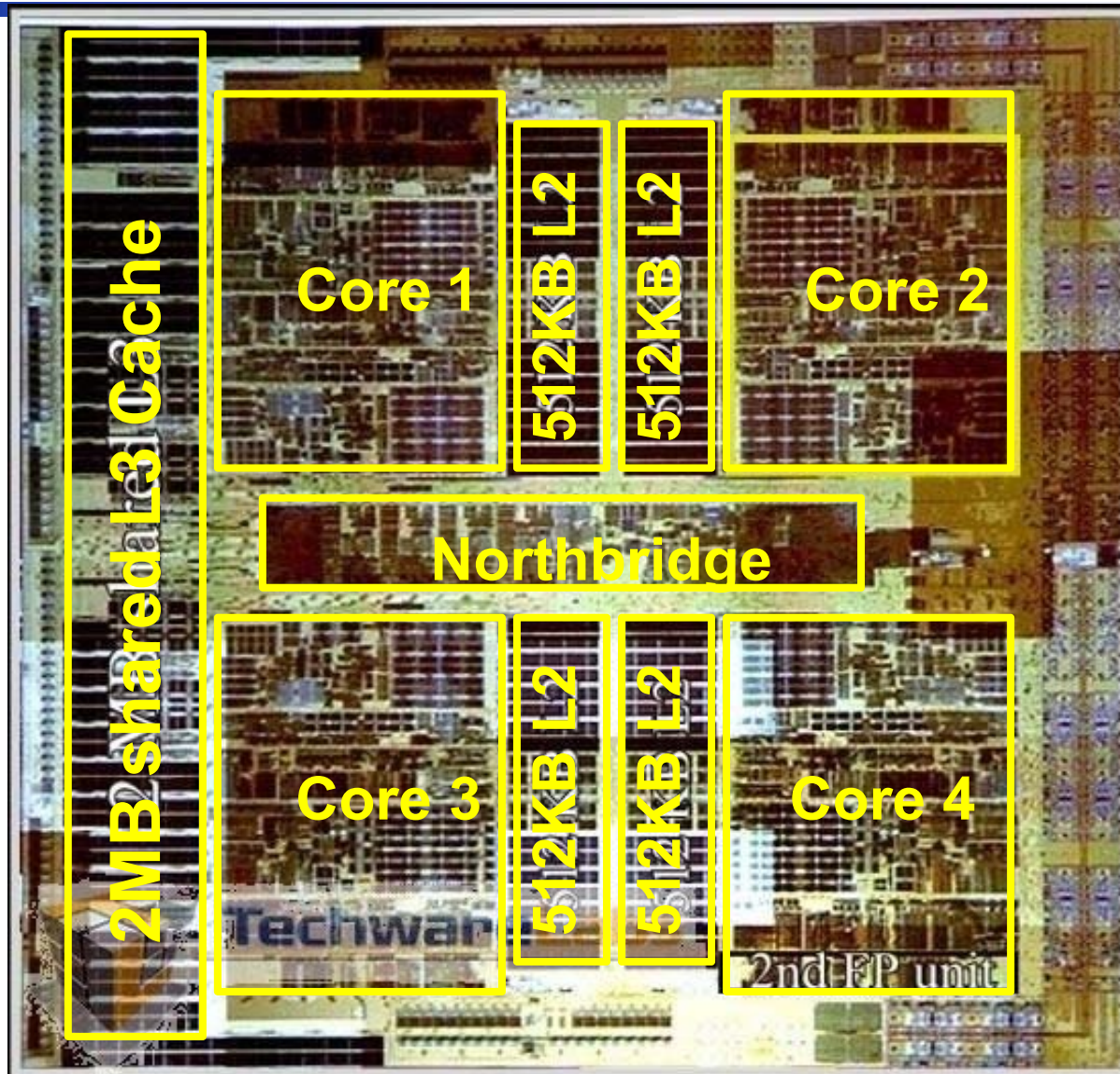


Hard drive Processor Fan with cover Spot for memory DIMMs Spot for battery Motherboard Fan with cover DVD drive





# Inside the Processor



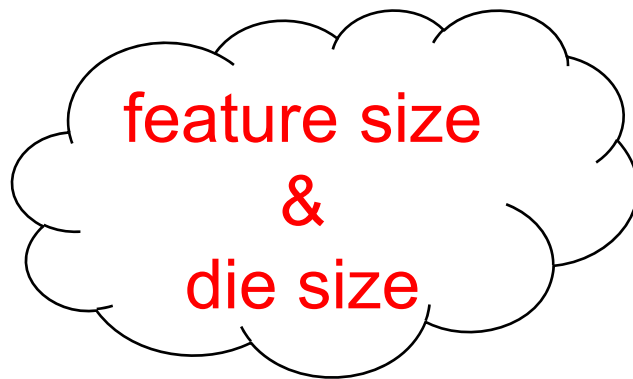
- ❑ AMD's Barcelona Multicore Chip
- ❑ Four out-of-order cores on one chip
- ❑ 1.9 GHz clock rate
- ❑ 65nm technology
- ❑ Three levels of caches (L1, L2, L3) on chip
- ❑ Integrated Northbridge

# Instruction Set Architecture (ISA)

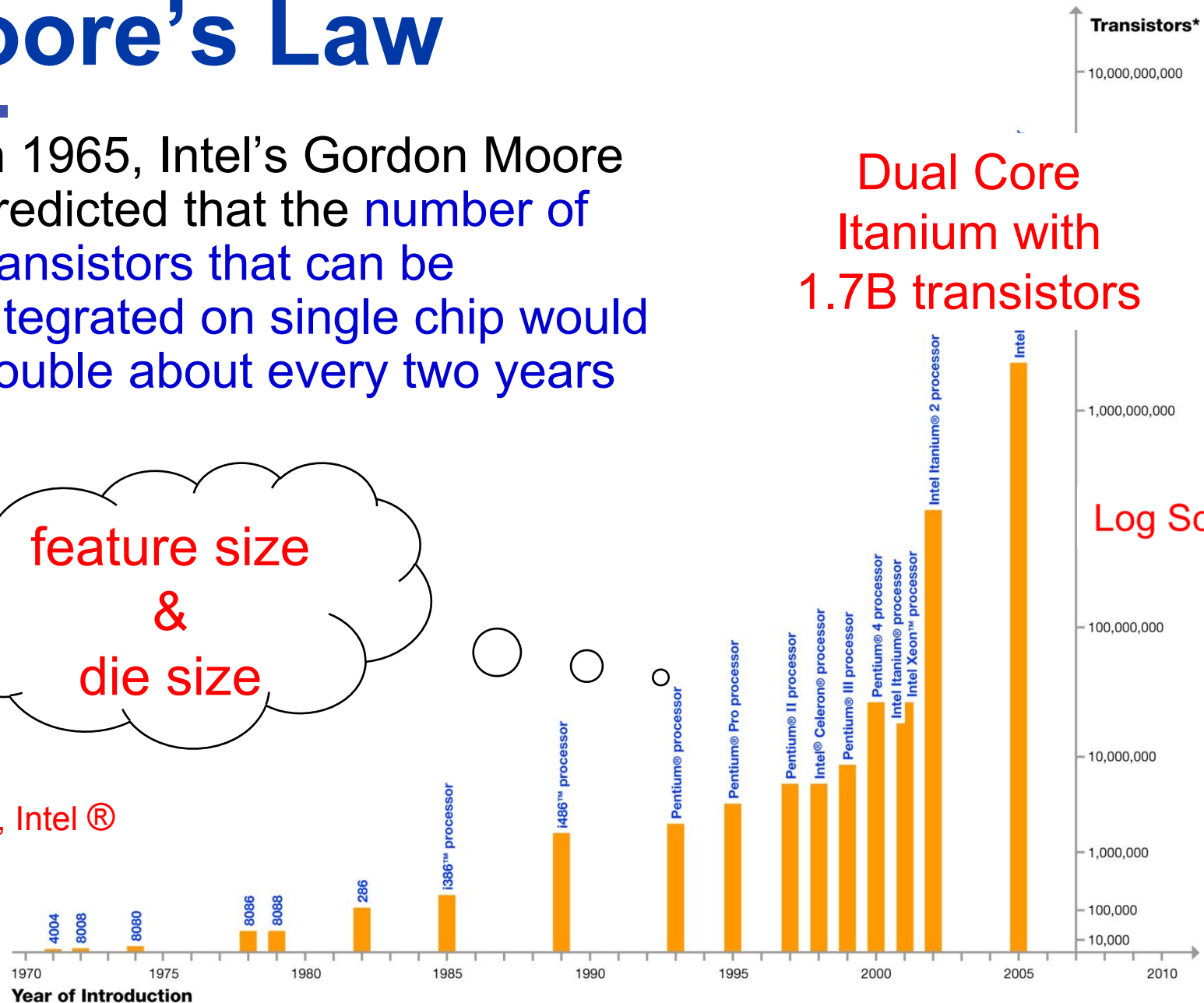
- ❑ ISA, or simply architecture – the abstract **interface** between the hardware and the lowest level software that encompasses all the information necessary to write a machine language program, including instructions, registers, memory access, I/O, ...
  - 1 Enables **implementations** of varying cost 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)
  - 1 ABI – The user portion of the instruction set plus the operating system interfaces used by application programmers.
    - 1 Defines a standard for binary portability across computers.

# 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



Courtesy, Intel ®

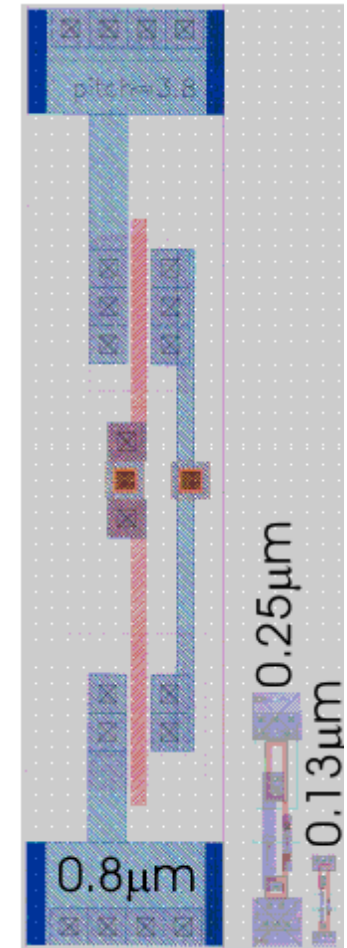


Dual Core  
Itanium with  
1.7B transistors

\*Note: Vertical scale of chart not proportional to actual Transistor count.

# Technology Scaling

- Key enabler for smaller, faster and more power efficient computing systems
- Technology scaling has a threefold objective:
  - Increase the transistor density
  - Reduce the gate delay
  - Reduce the power consumption
- Device dimensions (lateral and vertical) and voltages are reduced by  $1/\alpha$  ( $\sim 0.7$ )
- Technology generation spans 2-3 years





# Technology Scaling Roadmap (ITRS)

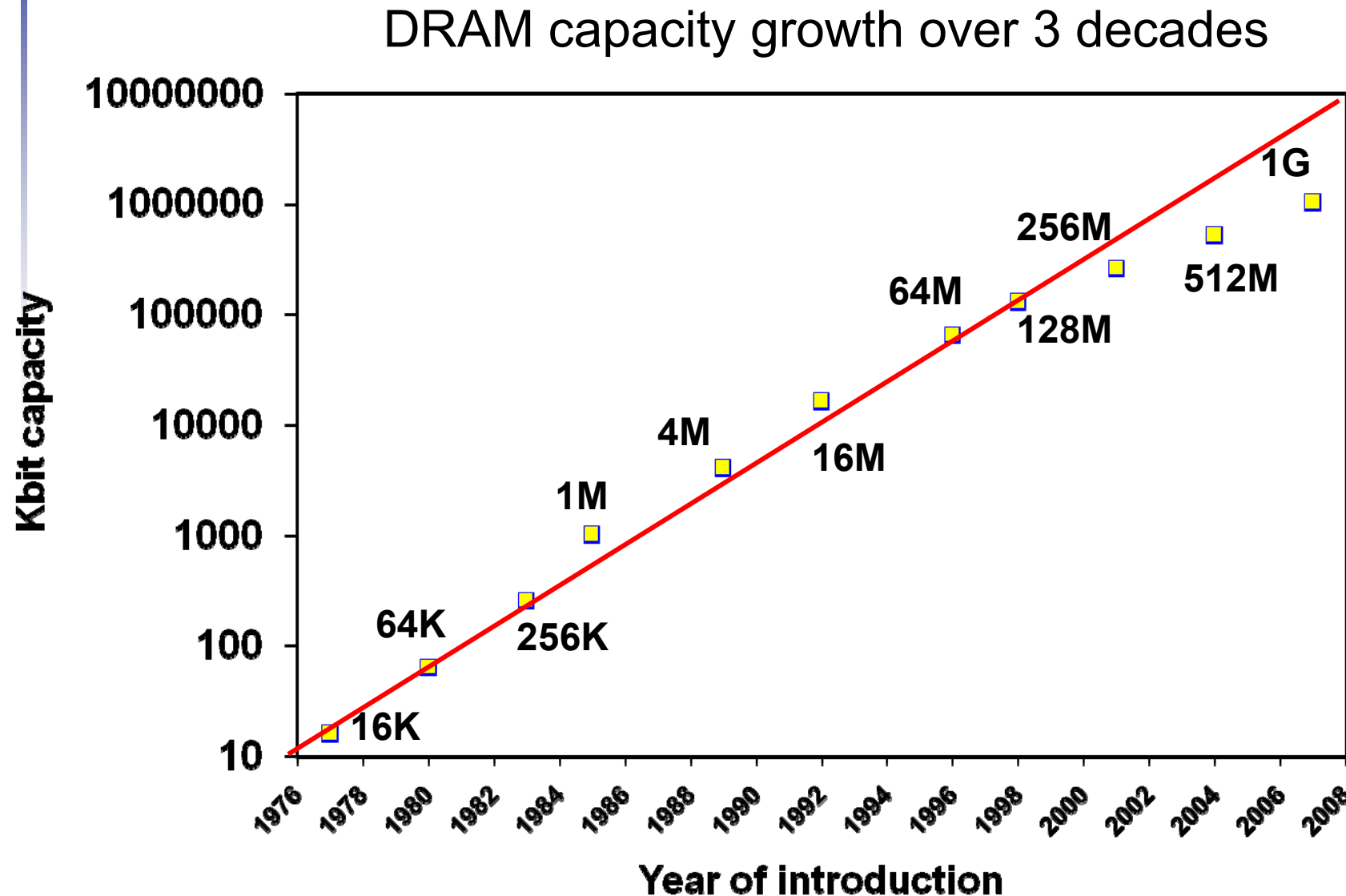
- Electronics technology continues to evolve
  - Increased capacity and performance; reduced cost

Year	2006	2008	2010	2012	2014
Feature size (nm)	90	65	45	32	22
Intg. Capacity (BT)	2	4	6	16	32

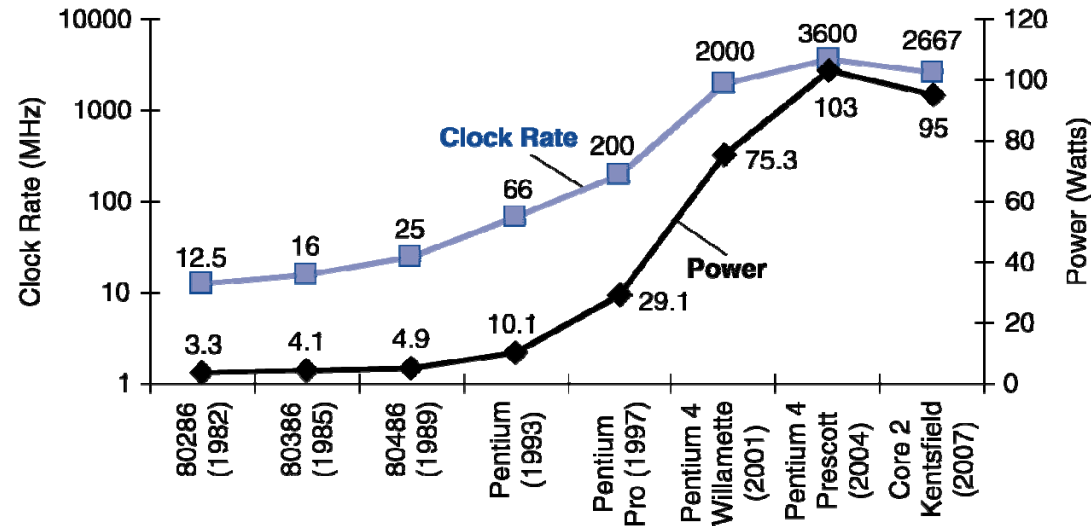
## □ Fun facts about 45nm transistors

- 1 30 million can fit on the head of a pin
- 1 You could fit more than 2,000 across the width of a human hair
- 1 If car prices had fallen at the same rate as the price of a single transistor has since 1968, a new car today would cost about ?
- 1 1 cent

# Another Example of Moore's Law Impact



# Power Trends



- In CMOS IC technology

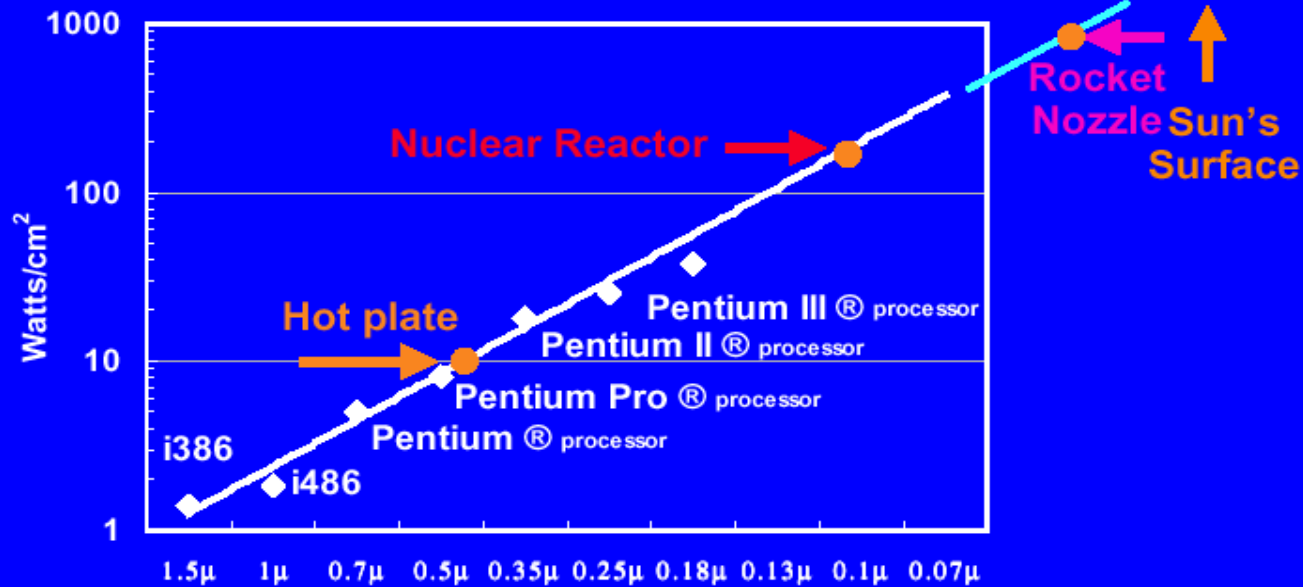
$$\text{Power} = \text{Capacitive load} \times \text{Voltage}^2 \times \text{Frequency}$$

× 30

5V → 1V

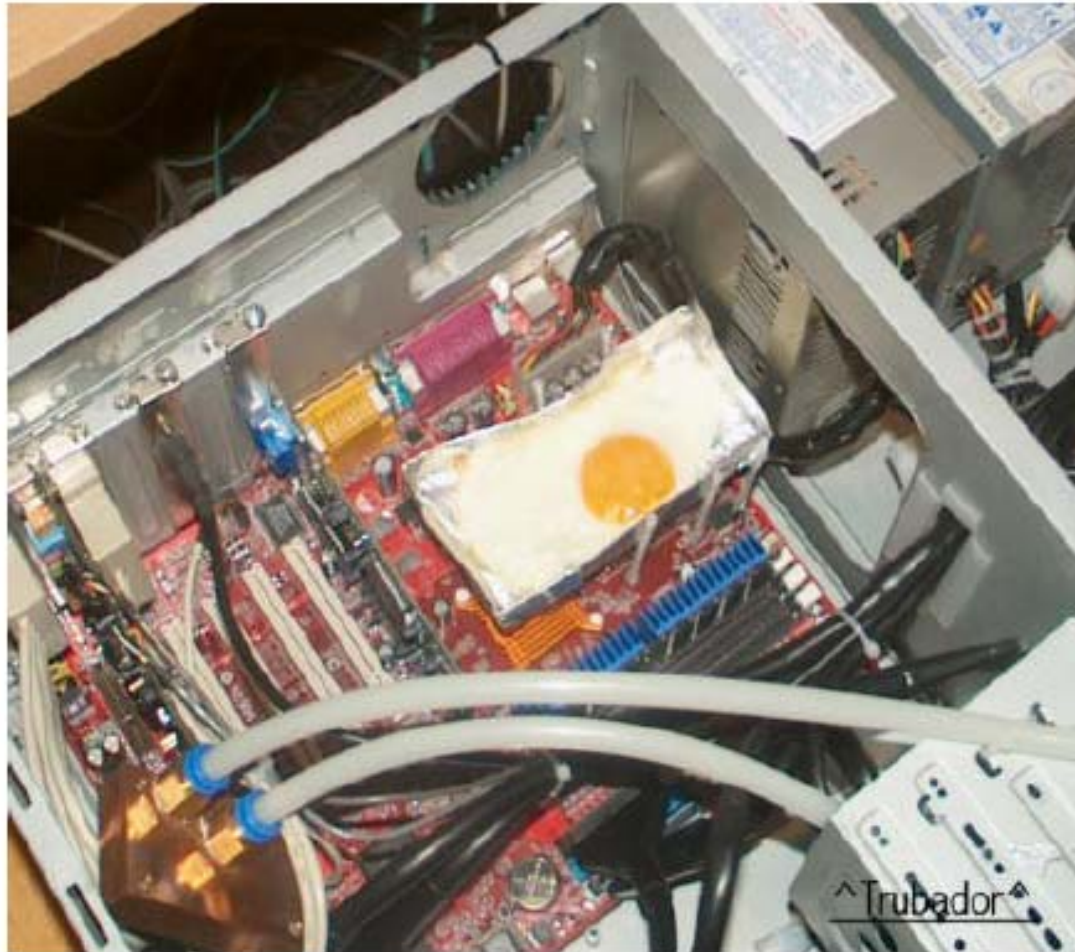
× 1000

# Power Density Getting Worse

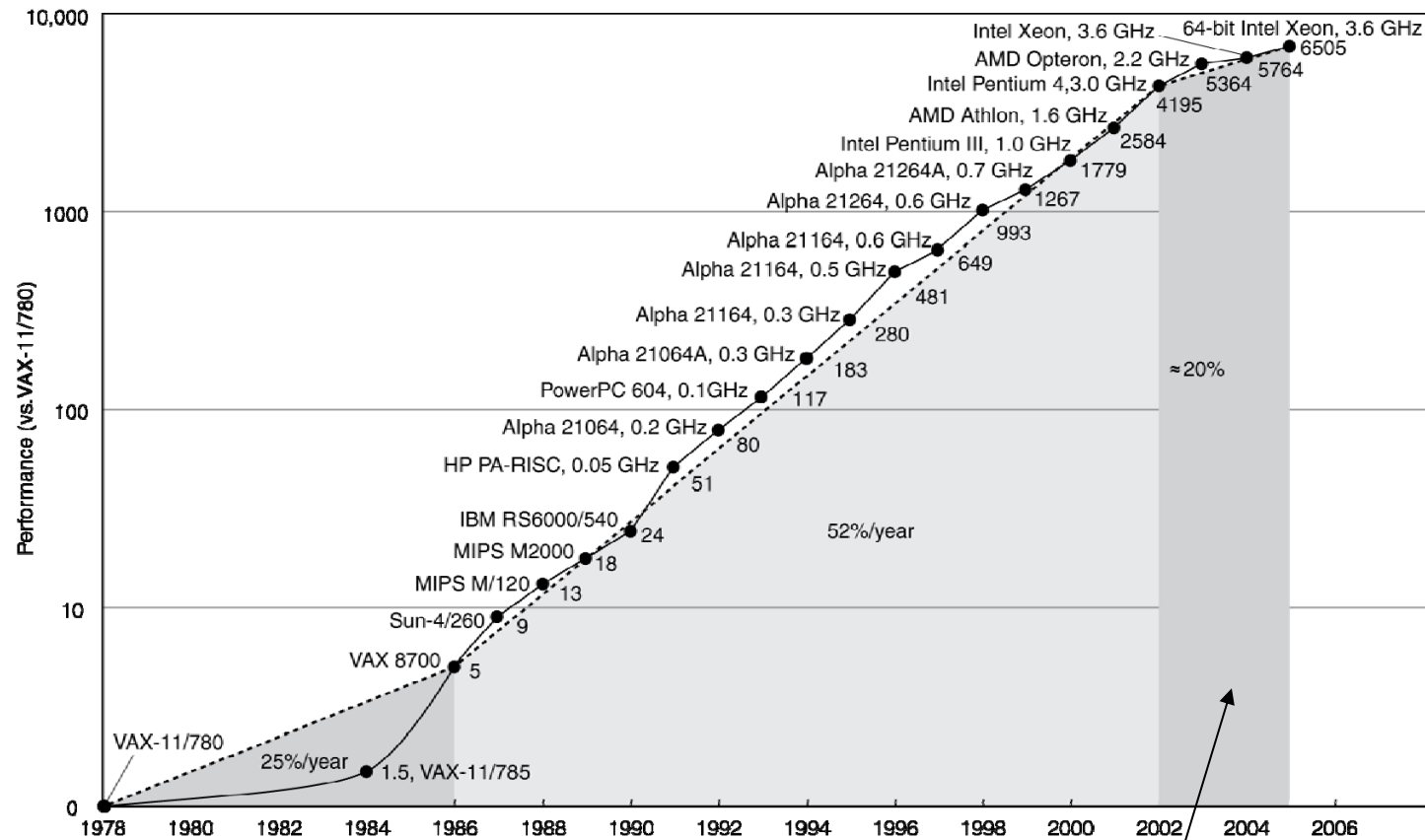


Surpassed hot-plate power density in 0.5μ  
Not too long to reach nuclear reactor

## Surpassed hot (kitchen) plate ... why not use it?



# Uniprocessor Performance



Constrained by power, instruction-level parallelism,  
memory latency

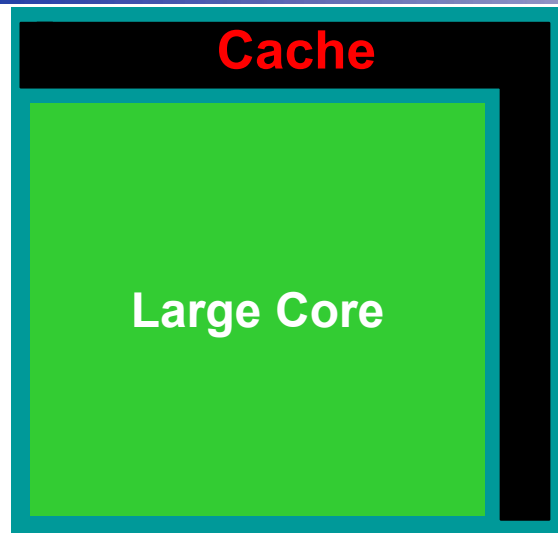
# The Sea Change

- ❑ The power challenge has forced a change in the design of microprocessors
  - 1 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 – cores – per chip

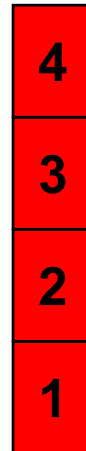
Product	AMD Barcelona	Intel Nehalem	IBM Power 6	Sun Niagara 2
Cores per chip	4	4	2	8
Clock rate	2.5 GHz	~2.5 GHz?	4.7 GHz	1.4 GHz
Power	120 W	~100 W?	~100 W?	94 W

- ❑ Plan of record is to double the number of cores per chip per generation (about every two years)

# Why Multiple Cores on a Chip?



Power

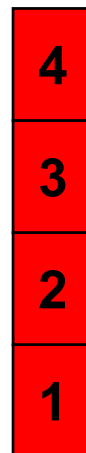
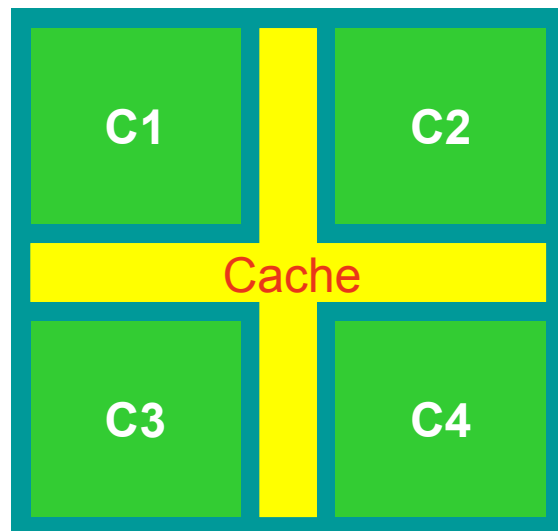
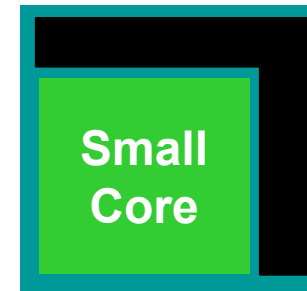


Performance



Power = 1/4

Performance = 1/2



**Multi-Core:**  
Power efficient  
Better power and  
thermal management



# Performance Metrics

- ❑ Purchasing perspective

- 1 given a collection of machines, which has the
  - best performance ?
  - least cost ?
  - best cost/performance?

- ❑ Design perspective

- 1 faced with design options, which has the
  - best performance improvement ?
  - least cost ?
  - best cost/performance?

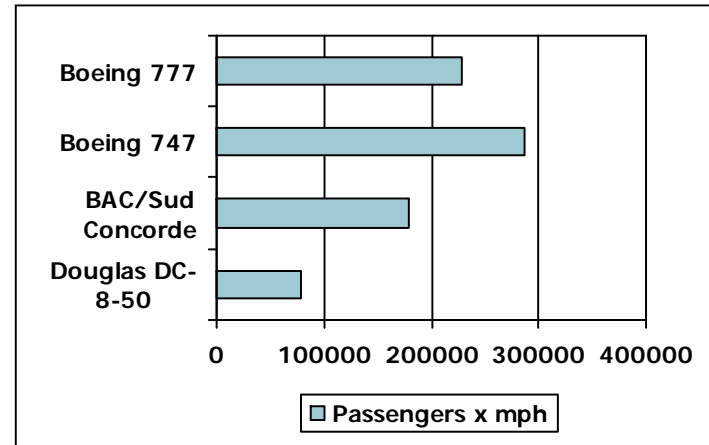
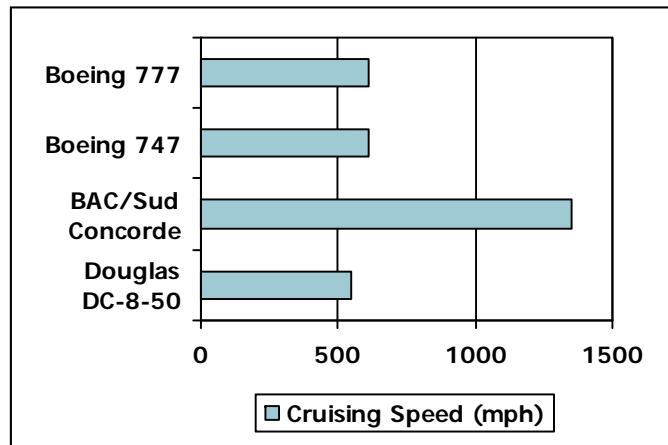
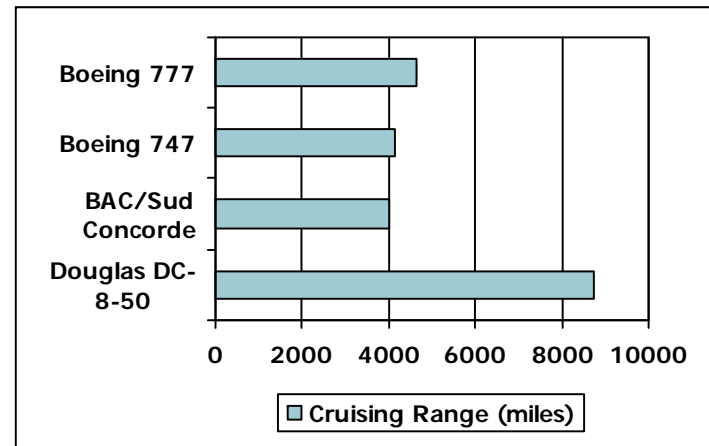
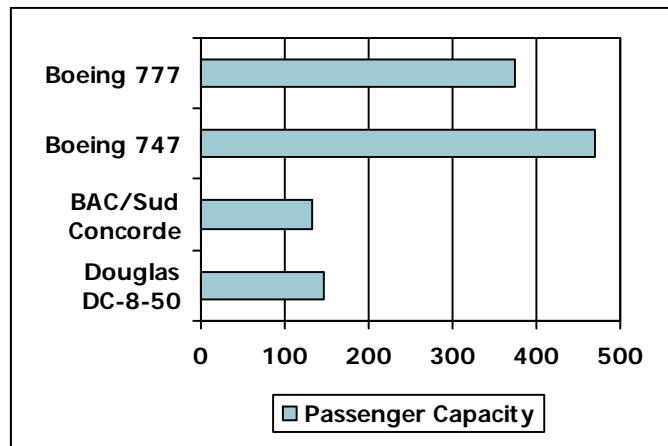
- ❑ Both require

- 1 basis for comparison
- 1 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

# Defining Performance

- Which airplane has the best performance?



# Response Time and Throughput

- Response time
  - How long it takes to do a task
    - Important to **individual** users
- Throughput
  - Total work done per unit time
    - e.g., tasks/transactions/... per hour
    - Important to **data center** managers
- How are response time & 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 =  $1/\text{Execution Time}$
- “X is  $n$  time faster than Y”

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

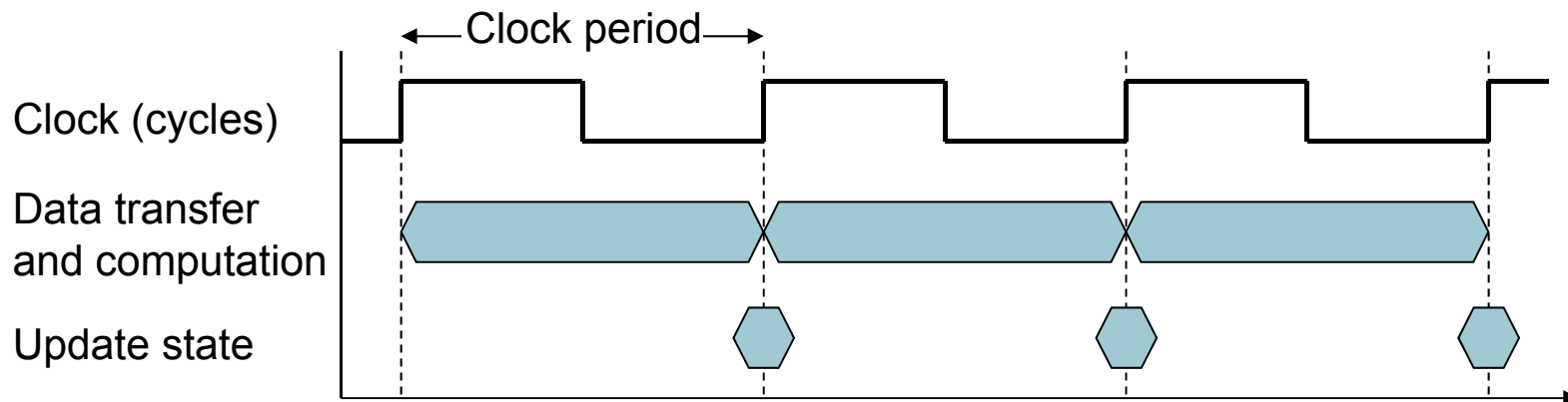
- **Example:** time taken to run a program
  - 10s on A, 15s on B
  - $\text{Execution Time}_B / \text{Execution Time}_A$   
 $= 15\text{s} / 10\text{s} = 1.5$
  - So A is 1.5 times faster than B

# Measuring Execution Time

- Execution time: seconds/program
- Elapsed time (wall clock 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
    - Discounts I/O time, other jobs' shares
  - Comprises user CPU time and system CPU time
  - Different programs are affected differently by CPU and system performance

# CPU Clocking: Review

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



- **Clock period:** duration of a clock cycle
  - e.g.,  $250\text{ps} = 0.25\text{ns} = 250 \times 10^{-12}\text{s}$
- **Clock frequency (rate):** cycles per second
  - e.g.,  $4.0\text{GHz} = 4000\text{MHz} = 4.0 \times 10^9\text{Hz}$

# CPU Clocking: Review

- Clock rate (clock cycles per second in MHz or GHz) is inverse of clock cycle time (clock period)

$$CC = 1 / CR$$

10 nsec clock cycle => 100 MHz clock rate

5 nsec clock cycle => 200 MHz clock rate

2 nsec clock cycle => 500 MHz clock rate

1 nsec ( $10^{-9}$ ) clock cycle => 1 GHz ( $10^9$ ) clock rate

500 psec clock cycle => 2 GHz clock rate

250 psec clock cycle => 4 GHz clock rate

200 psec clock cycle => 5 GHz clock rate

# CPU Time

$$\begin{aligned}\text{CPU Time} &= \text{CPU Clock Cycles} \times \text{Clock Cycle Time} \\ &= \frac{\text{CPU Clock Cycles}}{\text{Clock Rate}}\end{aligned}$$

- Performance improved by
  - Reducing number of clock cycles
  - Increasing clock rate
- Hardware designer must often trade off clock rate against cycle count
  - Many techniques that decrease the number of clock cycles also increase the clock cycle time



# CPU Time Example

- A program runs on computer A with a 2 GHz clock in 10 seconds. What clock rate must computer B run at to run this program in 6 seconds? Unfortunately, to accomplish this, computer B will require 1.2 times as many clock cycles as computer A to run the program.

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

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

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

# 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?

$$\begin{aligned}\text{CPU Time}_A &= \text{Instruction Count} \times \text{CPI}_A \times \text{Cycle Time}_A \\ &= 1 \times 2.0 \times 250\text{ps} = 1 \times 500\text{ps} \leftarrow \text{A is faster...}\end{aligned}$$

$$\begin{aligned}\text{CPU Time}_B &= \text{Instruction Count} \times \text{CPI}_B \times \text{Cycle Time}_B \\ &= 1 \times 1.2 \times 500\text{ps} = 1 \times 600\text{ps}\end{aligned}$$

$$\frac{\text{CPU Time}_B}{\text{CPU Time}_A} = \frac{1 \times 600\text{ps}}{1 \times 500\text{ps}} = 1.2 \leftarrow \text{...by this much}$$

# 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

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

Relative frequency

# CPI Example

- Alternative compiled code sequences using instructions in classes A, B, C. **What is avg. CPI?**

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  
 $= 2 \times 1 + 1 \times 2 + 2 \times 3$   
 $= 10$
- Avg. CPI =  $10/5 = 2.0$

- Sequence 2: IC = 6**

- Clock Cycles  
 $= 4 \times 1 + 1 \times 2 + 1 \times 3$   
 $= 9$
- Avg. CPI =  $9/6 = 1.5$

# Performance Summary

## The BIG Picture

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

	Instruction_count	CPI	clock_cycle
Algorithm	X	X	
Programming language	X	X	
Compiler	X	X	
ISA	X	X	X
Core organization		X	X
Technology			X

# A Simple Example

Op	Freq	CPI <sub>i</sub>	Freq x CPI <sub>i</sub>			
ALU	50%	1	.5	.5	.5	.25
Load	20%	5	1.0	.4	1.0	1.0
Store	10%	3	.3	.3	.3	.3
Branch	20%	2	.4	.4	.2	.4
			Σ = 2.2	1.6	2.0	1.95

- How much faster would the machine be if a better data cache reduced the average load time to 2 cycles?

CPU time new = 1.6 x IC x CC so 2.2/1.6 means 37.5% faster

- How does this compare with using branch prediction to shave a cycle off the branch time?

CPU time new = 2.0 x IC x CC so 2.2/2.0 means 10% faster

- What if two ALU instructions could be executed at once?

CPU time new = 1.95 x IC x CC so 2.2/1.95 means 12.8% faster

# Workloads and Benchmarks

- ❑ Benchmarks – a set of programs that form a “workload” specifically chosen to measure performance
- ❑ SPEC (System Performance Evaluation Cooperative) 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](http://www.spec.org)

- ❑ There are also benchmark collections for power workloads (SPECpower\_ssj2008), for mail workloads (SPECmail2008), for multimedia workloads (mediabench), ...



# CINT2006 for Opteron X4 2356

Name	Description	IC $\times 10^9$	CPI	Tc (ns)	Exec time	Ref time	SPECratio
perl	Interpreted string processing	2,118	0.75	0.4	637	9,777	15.3
bzip2	Block-sorting compression	2,389	0.85	0.4	817	9,650	11.8
gcc	GNU C Compiler	1,050	1.72	0.4	24	8,050	11.1
mcf	Combinatorial optimization	336	10.00	0.4	1,345	9,120	6.8
go	Go game (AI)	1,658	1.09	0.4	721	10,490	14.6
hmmer	Search gene sequence	2,783	0.80	0.4	890	9,330	10.5
sjeng	Chess game (AI)	2,176	0.96	0.4	37	12,100	14.5
libquantum	Quantum computer simulation	1,623	1.61	0.4	1,047	20,720	19.8
h264avc	Video compression	3,102	0.80	0.4	993	22,130	22.3
omnetpp	Discrete event simulation	587	2.94	0.4	690	6,250	9.1
astar	Games/path finding	1,082	1.79	0.4	773	7,020	9.1
xalancbmk	XML parsing	1,058	2.70	0.4	1,143	6,900	6.0
Geometric mean							11.7

High cache miss rates

# Comparing and Summarizing Performance

- ❑ How do we summarize the performance for benchmark set with a **single** number?
  - 1 First the execution times are normalized giving the “SPEC ratio” (bigger is faster, i.e., SPEC ratio is the inverse of execution time)
  - 1 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 Power Benchmark

- Power consumption of server at different workload levels
  - Performance: ssj\_ops/sec (server side java ops/sec)
  - Power: Watts (Joules/sec)

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

# SPECpower\_ssjs2008 for X4

Target Load %	Performance (ssj_ops/sec)	Average Power (Watts)
100%	231,867	295
90%	211,282	286
80%	185,803	275
70%	163,427	265
60%	140,160	256
50%	118,324	246
40%	92,035	233
30%	70,500	222
20%	47,126	206
10%	23,066	180
0%	0	141
Overall sum	1,283,590	2,605
$\sum \text{ssj\_ops} / \sum \text{power}$		493

# Pitfall: Amdahl's Law

- **Amdahl's law**: performance enhancement possible with a given improvement is limited by the amount that the improved feature is used
- **Pitfall**: 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?

$$20 = \frac{80}{n} + 20 \quad \blacksquare \text{ Can't be done!}$$

- **Corollary**: make the common case fast

# Fallacy: Low Power at Idle

- Look back at X4 power benchmark
  - At 100% load: 295W
  - At 50% load: 246W (83%)
  - At 10% load: 180W (61%)
- 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

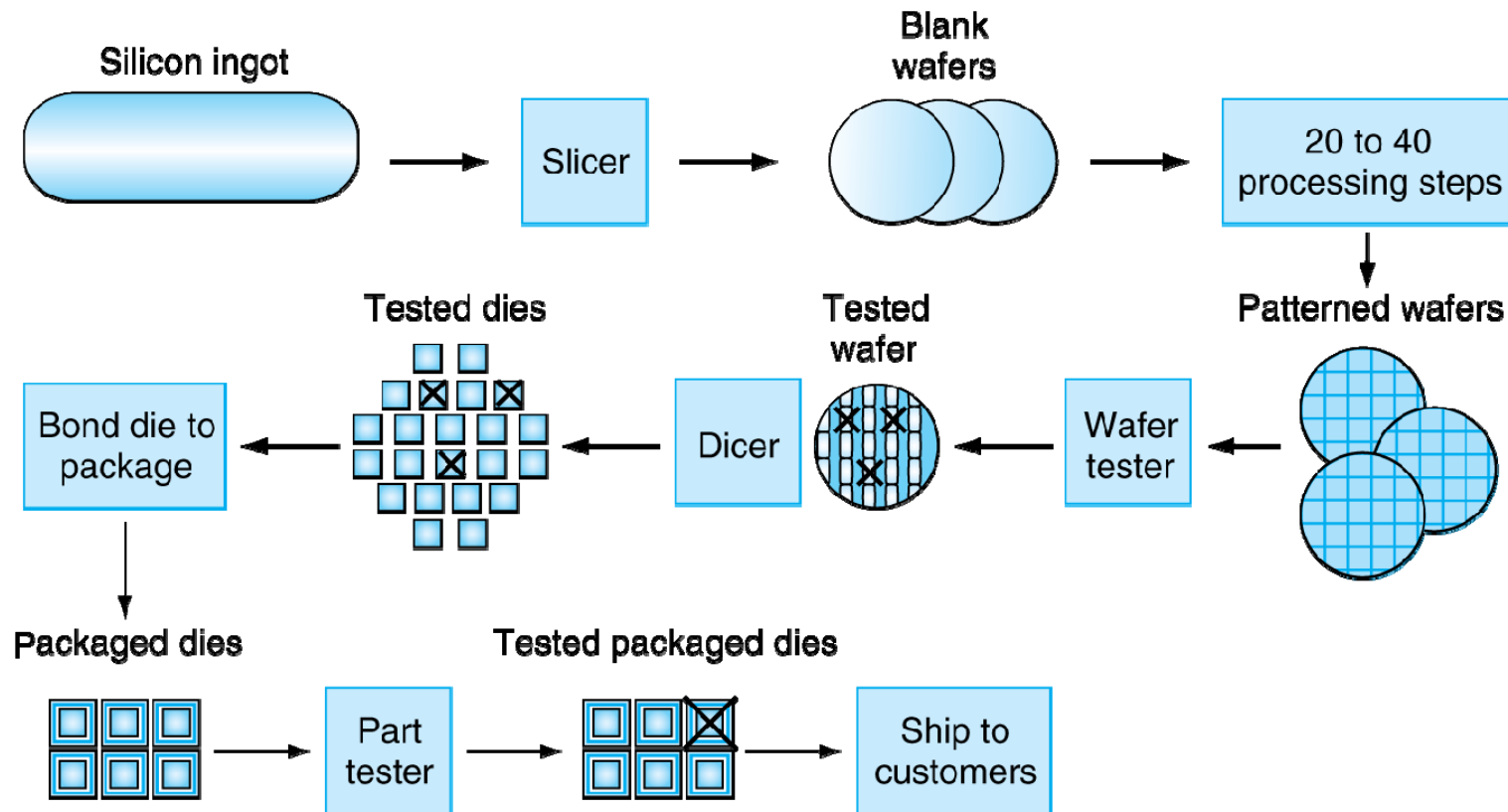
# Pitfall: MIPS as a Performance Metric

- MIPS: Millions of Instructions Per Second
  - If used as a metric to compare computers:
    - Doesn't account for differences in instruction complexity
      - Different ISAs may lead to different instruction counts for same program
    - MIPS varies between programs on the same computer!
      - Computer cannot have a single MIPS rating

$$\begin{aligned}\text{MIPS} &= \frac{\text{Instruction count}}{\text{Execution time} \times 10^6} \\ &= \frac{\text{Instruction count}}{\frac{\text{Instruction count} \times \text{CPI}}{\text{Clock rate}} \times 10^6} = \frac{\text{Clock rate}}{\text{CPI} \times 10^6}\end{aligned}$$

- e.g. CPI varied by 13x for SPEC2006 on AMD Opteron X4, so MIPS does as well

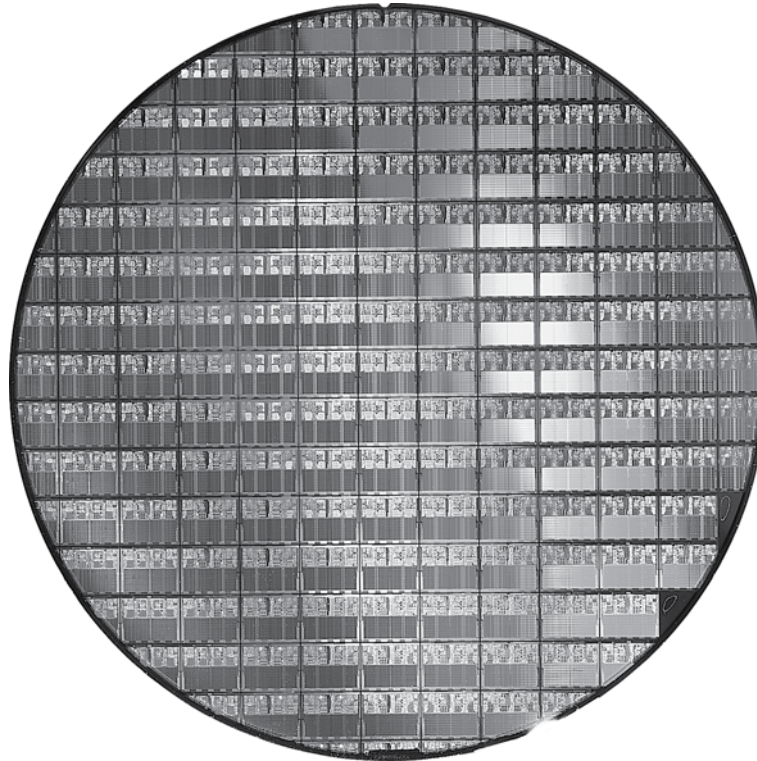
# Manufacturing ICs



- **Yield:** proportion of working dies per wafer



# AMD Opteron X2 Wafer



- X2: 300mm wafer, 117 chips, 90nm technology
- X4: 45nm technology

# 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

# 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