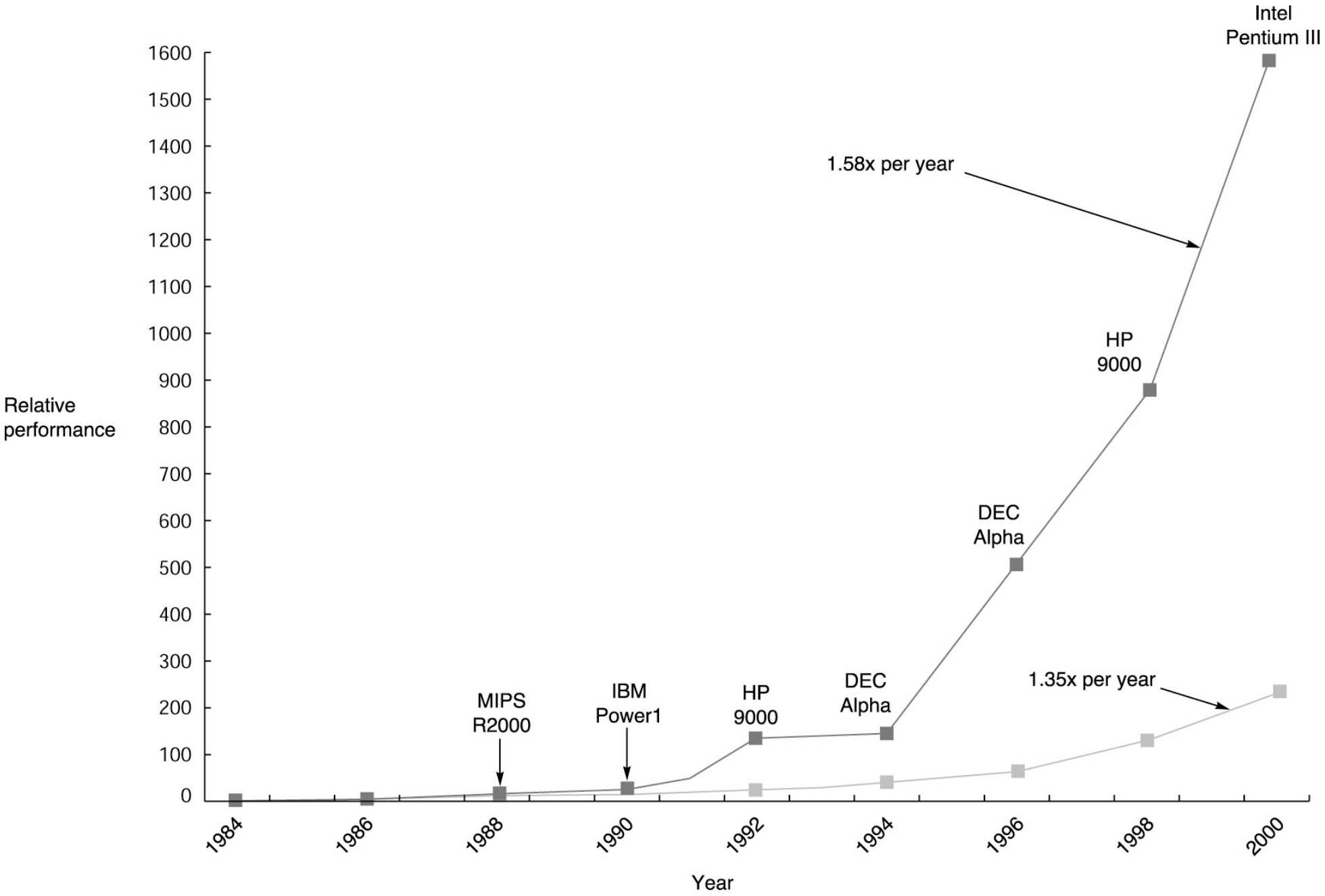
Chap. 1: **Fundamentals of Computer Design**

Topics	Pages	Outcome
Growth in performance	2—17	Trends in computing industry
What are computers made of	14—24	System components
Cost model of IC	17—21	Importance of minimizing die size
Performance	24—39	What performance means in computing
Benchmarking	26—31	The problem of choosing benchmarks
Reporting performance	32—39	Averaging methods
Quantitative principles	39—48	How to use measurements
Amdhal's Law	40—42	How to achieve speed up
CPU performance	43—44	CPU performance eqn (notion of CPI)
Locality principle	47	Justification of many design ideas
Parallelism	48	Another broad class of design ideas



(pp. 2—3)

- Overall progress is a product of three factors of improvement:

- Technology, Architecture, Compiler.
  - 1.17 x 1.17 x 1.17 = **1.6 / year**.

- No other technological systems (say, airplanes or water-treatment plants) improve that fast.

(pp. 4—8)

Three broad markets

- Desktop computing (business, engineering, amusement) ($1,000 – $10,000)
- Servers (web, high availability) ($10,000 – $10,000,000)
- Embedded computers (appliances, vehicles, hand-held) ($10 – $100,000)

Feature	Desktop	Server	Embedded
Price of system	$1000–$10,000	$10,000–$10,000,000	$10–$100,000 (including network routers at the high end)
Price of microprocessor module	$100–$1000	$200–$2000 (per processor)	$0.20–$200 (per processor)
Microprocessors sold per year (estimates for 2000)	150,000,000	4,000,000	300,000,000 (32-bit and 64-bit processors only)
Critical system design issues	Price-performance, graphics performance	Throughput, availability, scalability	Price, power consumption, application-specific performance

Important terms related to *Computer Architecture*

- ***Implementation***: how an abstract description is turned into hardware.

- The ***instruction set architecture*** (ISA) is such abstraction.
  - HW / SW interface.
  - ADD C,A,B.

- Examples of ISAs
  - IA-32
  - IA-64
  - MIPS64
  - ARM

- ***Organization***: high level aspects of the design
  - memory, bus structure, CPU, I/O
  - design of these sometimes called “micro-architecture”.

- It is possible to *implement* the same *instruction set architecture* using different *organizations*, resulting in different systems.
  - E.x. Different Bus, memory organization, pipeline structure and so-on.

- ***Hardware*** refers to the specifics of an implementation.
  - For example the Pentium II and the Celeron have different hardware.

- It is even possible to *emulate* a function normally carried out in hardware (say floating point calculations) using software (lists of instructions).

(pp. 10—11)

Task of the Computer Designer

Functional requirements	***Required Features***
Application	***Target***
General-purpose desktop	Balanced performance for many tasks
Scientific/Engineering	High performance FP/Graphics
Commercial Servers	Reliability, availability, scalability
Embedded	Focus on few features (power, throughput, …)
Software compatibility	***re-use existing software***
At programming language	Flexible for designer, new compiler back-end
At object code	Same ISA
Operating System	***Necessary OS support***
Size of address space	Key
Memory management	Virtual Memory
Protection	Different OS, paging, segmentation
Standards	***Certain standards may be required by marketplace***
Floating point	Format, arithmetic, IEEE 734, Graphics
I/O	ATA, SCSI, PCI
Operating System	Unix, PalmOS, Windows, CE, IOS
Network	Ethernet, Infiniband
Programming Language	C, C++, Java, FORTRAN: ISA

Trends

- valid over long time periods, e.g. ISA can last decades

  - **Clock Rate: ~ 30% per year**
  - **Transistor Density: ~ 35%**
  - **Chip Area: ~ 15%**
  - **Transistors per chip: ~ 55%**
  - **Total Performance Capability: ~ 100%**

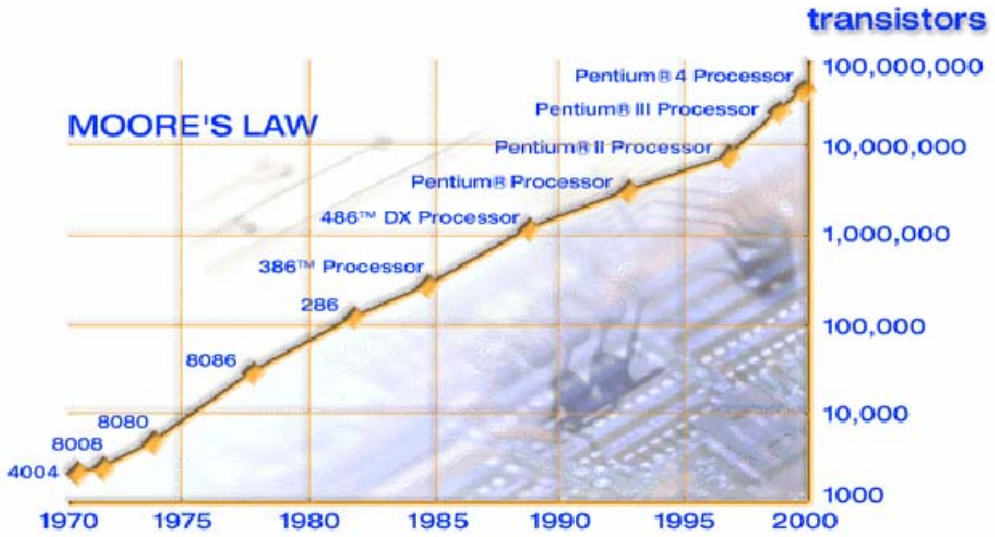
Moore's Law

2x number of transistors every 12-18 months

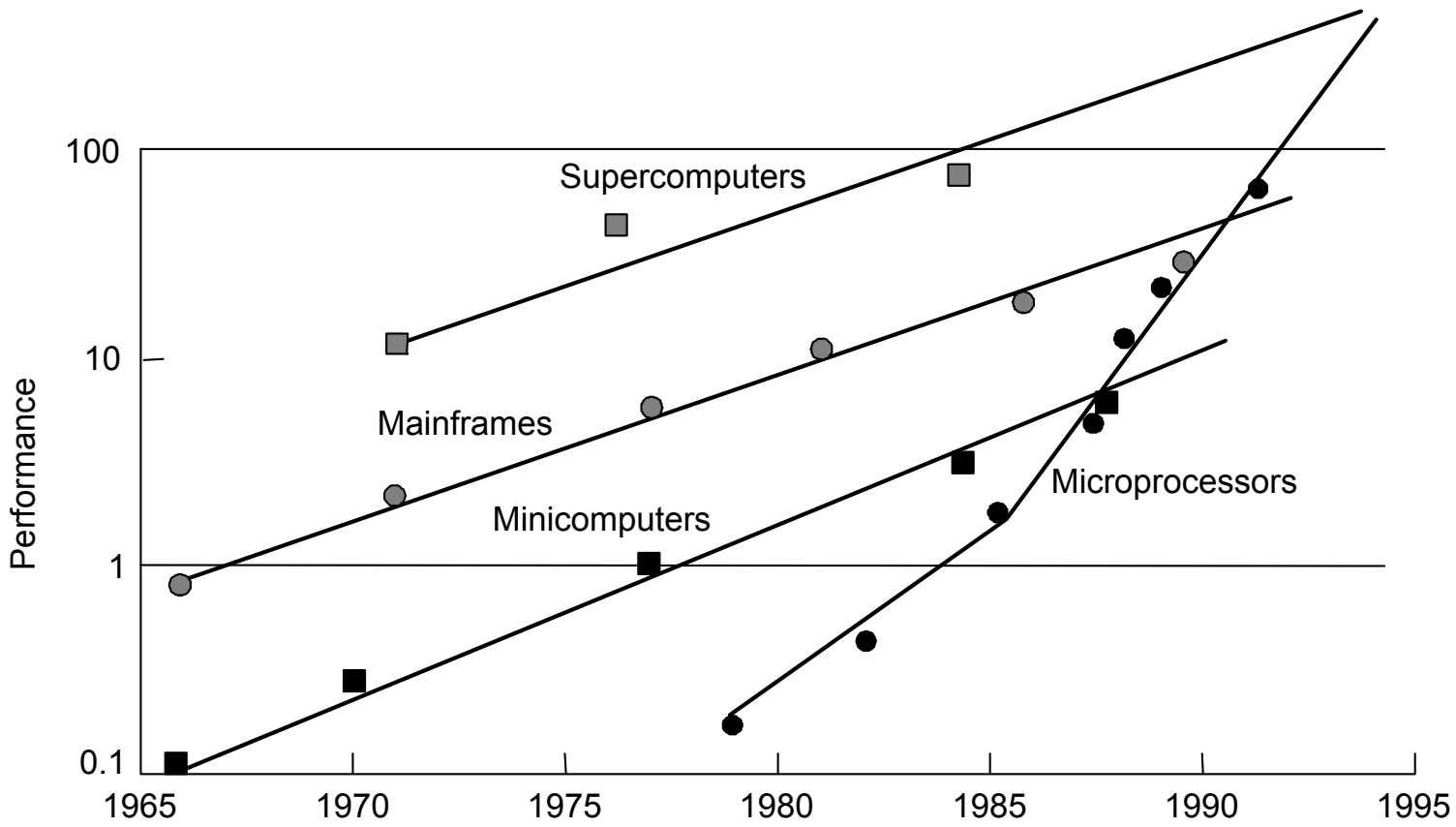
"The complexity for minimum component costs has increased at a rate of roughly a factor of two per year. Certainly over the short term this rate can be expected to continue, if not increase. Over the longer term, the rate of increase is a bit more uncertain, although there is no reason to believe it will not remain for nearly constant for at least 10 years. That means by 1975, the number of components per integrated circuit for minimum cost will be 65,000.
"I believe that such a large circuit can be built on a single wafer."

--- Gordon Moore, 1965

	Year of Introduction	Transistors
4004	1971	2,250
8008	1972	2,500
8080	1974	5,000
8086	1978	29,000
286	1982	120,000
386™	1985	275,000
486™ DX	1989	1,180,000
Pentium®	1993	3,100,000
Pentium II	1997	7,500,000
Pentium III	1999	24,000,000
Pentium 4	2000	42,000,000



Source: Intel



Source: D. Patterson

IC Technology

- Feature size (min size of transistor or wire)
  - 1971: 10 μm
  - 2001: 0.18 μm
  - 2002: 0.13 μm
  - 2003: 0.10 μm
- Quadratic increase in density, linear increase in performance

⇒ *Architectural improvement!*

- 8, 16, 32, 64 bit architectures (buses, ALU's)
- Pipelines and caches

- Transistor performance benefits from smaller resistance and capacitance.
- Interconnect propagation delay major problem.
  - e.g. Pentium 4 accounts for propagation of signals across chip.

Switching Power

$$P = f\ C\ V^2$$

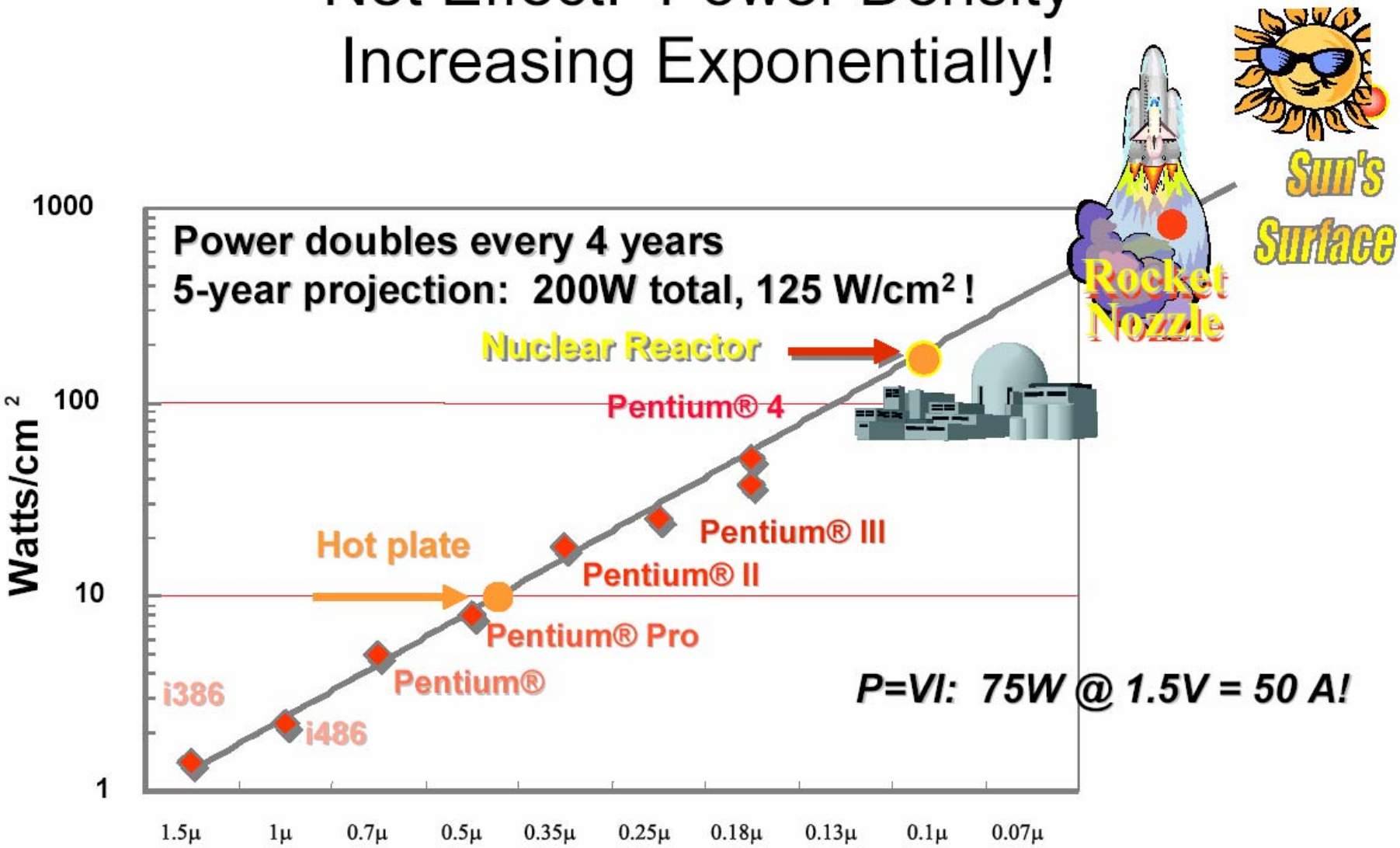
f = clock frequency

C = capacitance

V = voltage

- 2 GHz Pentium consumes 100 W $\Rightarrow$ Heat removal.
- Alternatively, portable computing requires low power.

Net Effect: Power Density Increasing Exponentially!



* "New Microarchitecture Challenges in the Coming Generations of CMOS Process Technologies" – Fred Pollack, Intel Corp. Micro32 conference key note - 1999. Courtesy Avi Mendelson, Intel.

Other "Famous" Predictions

"There is no reason for any individual to have a computer in his home."

Kenneth H. Olson, President of DEC,
Convention of the World Future Society, 1977

"640 kilobytes (of computer memory) ought to be enough for anybody."

Bill Gates
Founder and head of Microsoft, 1981

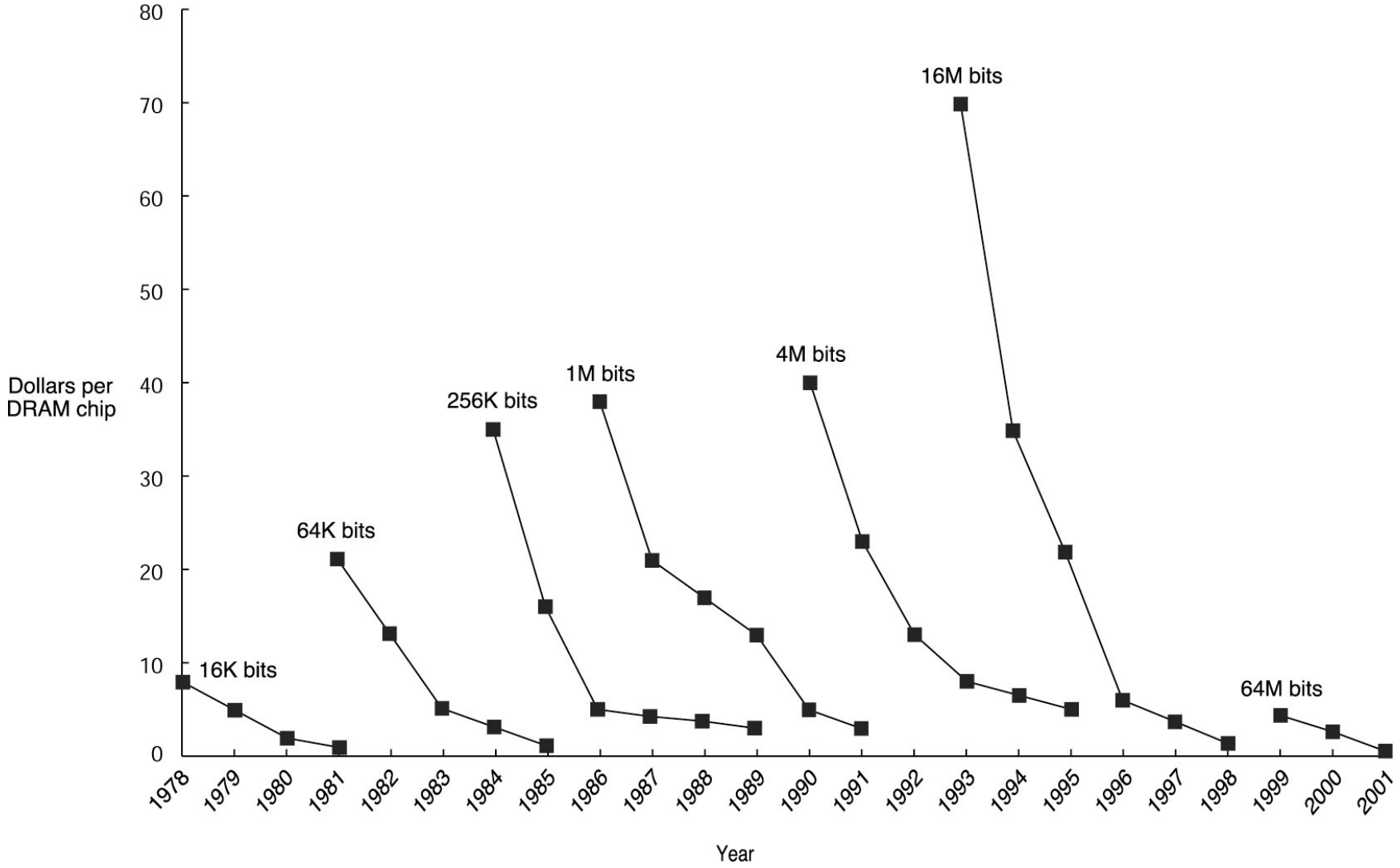
Cost, Price, and Trends

- What is the nature of the cost-performance tradeoff?
- Driven by cost of components
  - one important aspect is their change over time.

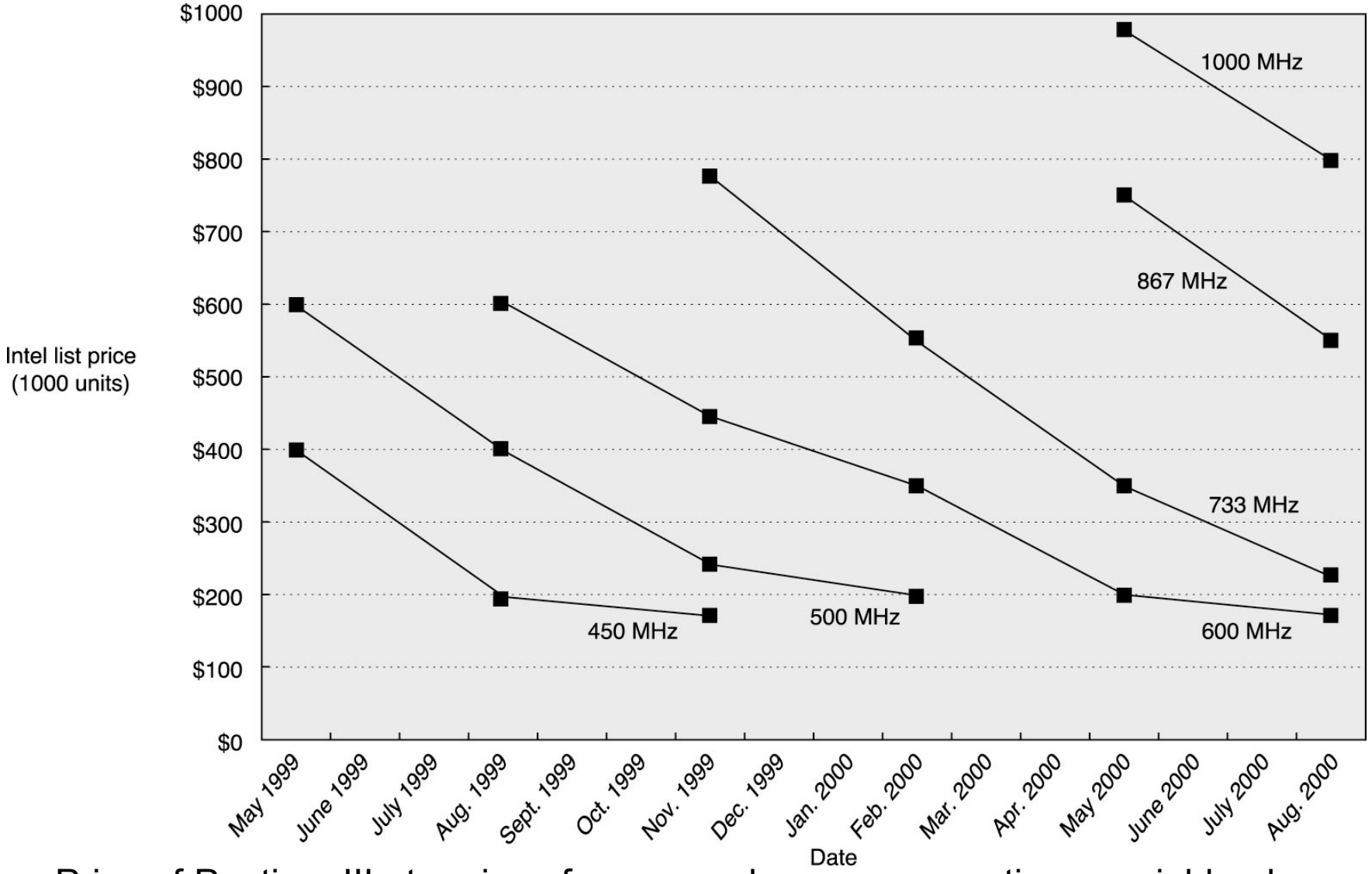
Time and volume

- Manufacturing learning curve.
  - ***Yield*** improves with time.
  - Doubling the yield halves the cost.
- Example: DRAM chips have strange business behaviors because of rapid changes, price can be lower than cost for a short moment.

- Microprocessors are less predictable.
- E.g. Pentium III
- Roughly cost x 0.9 for volume doubling.
- Expansion of low-end market has produced "commoditization" with fierce competition and razor-thin margins.



Prices of six generations of DRAMs over time in 1977 dollars. This shows the importance of the learning curve.



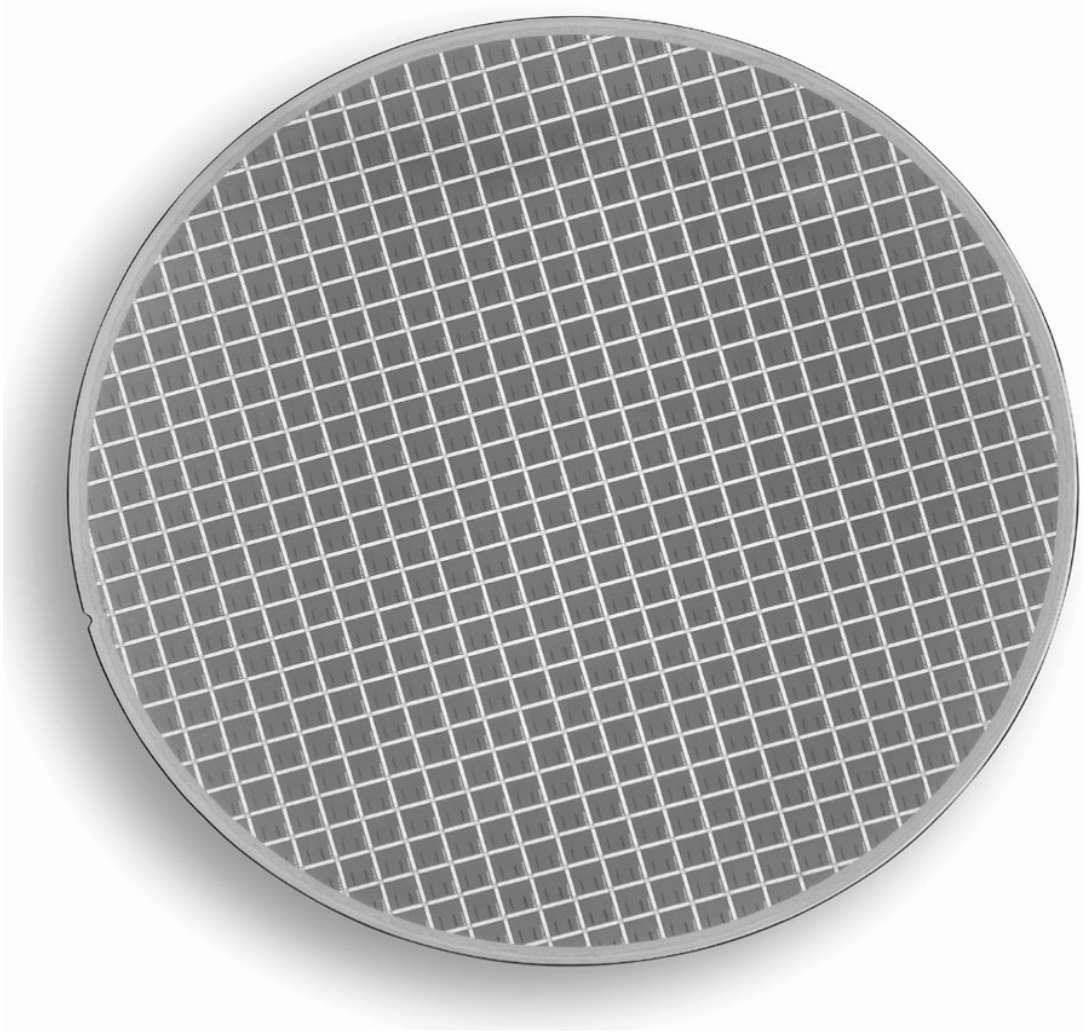
Price of Pentium III at a given frequency decreases over time as yield enhancements decrease the cost of a good die and competition forces price reductions.

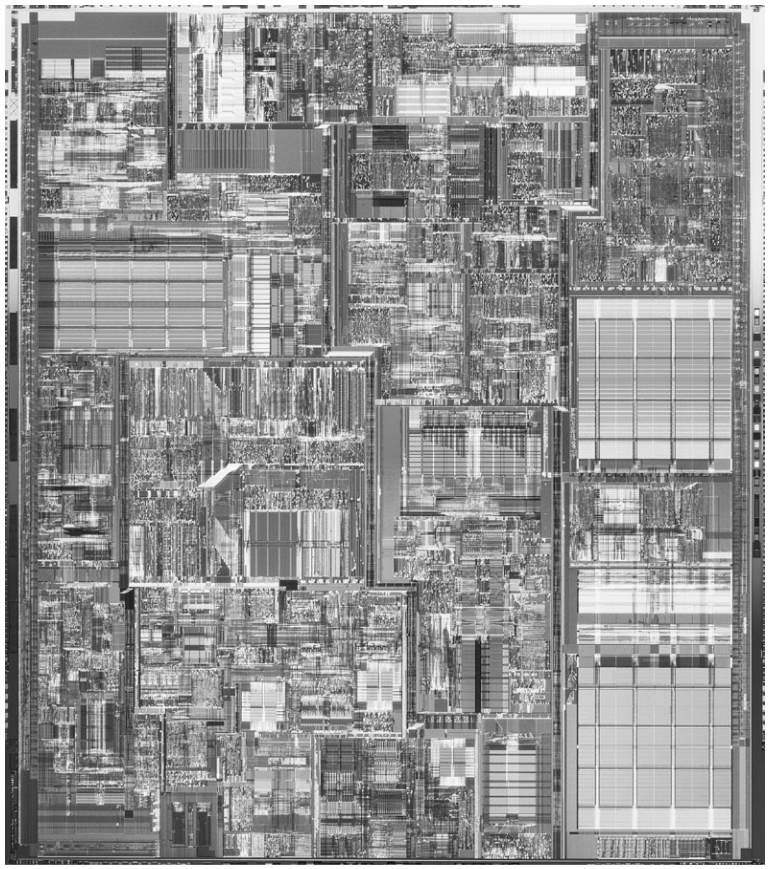
Cost of an IC

Manufacturing Steps.

- Silicon Crystal Growth extracted from molten silicon bath
- Processed (cleaned to very high level of purity) into cylinder
- Cylinder sliced to make wafers
- Wafers cleaned, polished and chemically processed
- Long sequence of steps involving deposit and removal of substances to etch the circuit according to patterns specified by optical masks.
- Dies cut, tested and packaged.

MIPS64 R20K WAFER (564 processors)





Intel Pentium 4

Cost model

$$ICCost = \frac{DieCost + DieTestCost + PackagingAndTestCost}{FinalYield}$$

$$DieCost = \frac{WaferCost}{DiesPerWafer \times DieYield}$$

$$DiesPerWafer = \frac{\pi \times WaferRadius^2}{DieArea} - \frac{\pi \times WaferDiameter}{\sqrt{2 \times DieArea}}$$

$$DieYield = WaferYield \times \left(1 + \frac{DefectDensity \times DieArea}{\alpha}\right)^{-\alpha}$$

DieYield is from an empirical formula where α reflect the number of process steps (complexity). α can be of the order of 3 or 4. *DefectDensity* is of the order of 0.4--0.8/cm^2.

$$ICCost = \frac{DieCost + DieTestCost + PackagingAndTestCost}{FinalYield}$$

Realistic example: $8000 / wafer, 350 raw dies / wafer

60% good dies, $80 to test wafer,

$4/unit to package and final test, 97% final test yield

$$DieCost = \frac{WaferCost}{DiesPerWafer \times DieYield}$$

DieCost =

DieTestCost (good dies) =

ICCost =

Real World Examples

Chip	Metal layers	Line width	Wafer cost	Defect /cm^2	Area mm^2	Dies/ wafer	Yield	Die Cost
386DX	2	0.90	$900	1.0	43	360	71%	$4
486DX2	3	0.80	$1200	1.0	81	181	54%	$12
PowerPC 601	4	0.80	$1700	1.3	121	115	28%	$53
HP PA 7100	3	0.80	$1300	1.0	196	66	27%	$73
DEC Alpha	3	0.70	$1500	1.2	234	53	19%	$149
SuperSPARC	3	0.70	$1700	1.6	256	48	13%	$272
Pentium	3	0.80	$1500	1.5	296	40	9%	$417

– From "Estimating IC Manufacturing Costs," by Linley Gwennap, *Microprocessor Report*, August 2, 1993, p. 15

Architect's quandry: price/performance tradeoff:

Given a 20 cm diameter wafer at $2000, α=4.

A. Using mid-life process (medium defect density = 0.6)

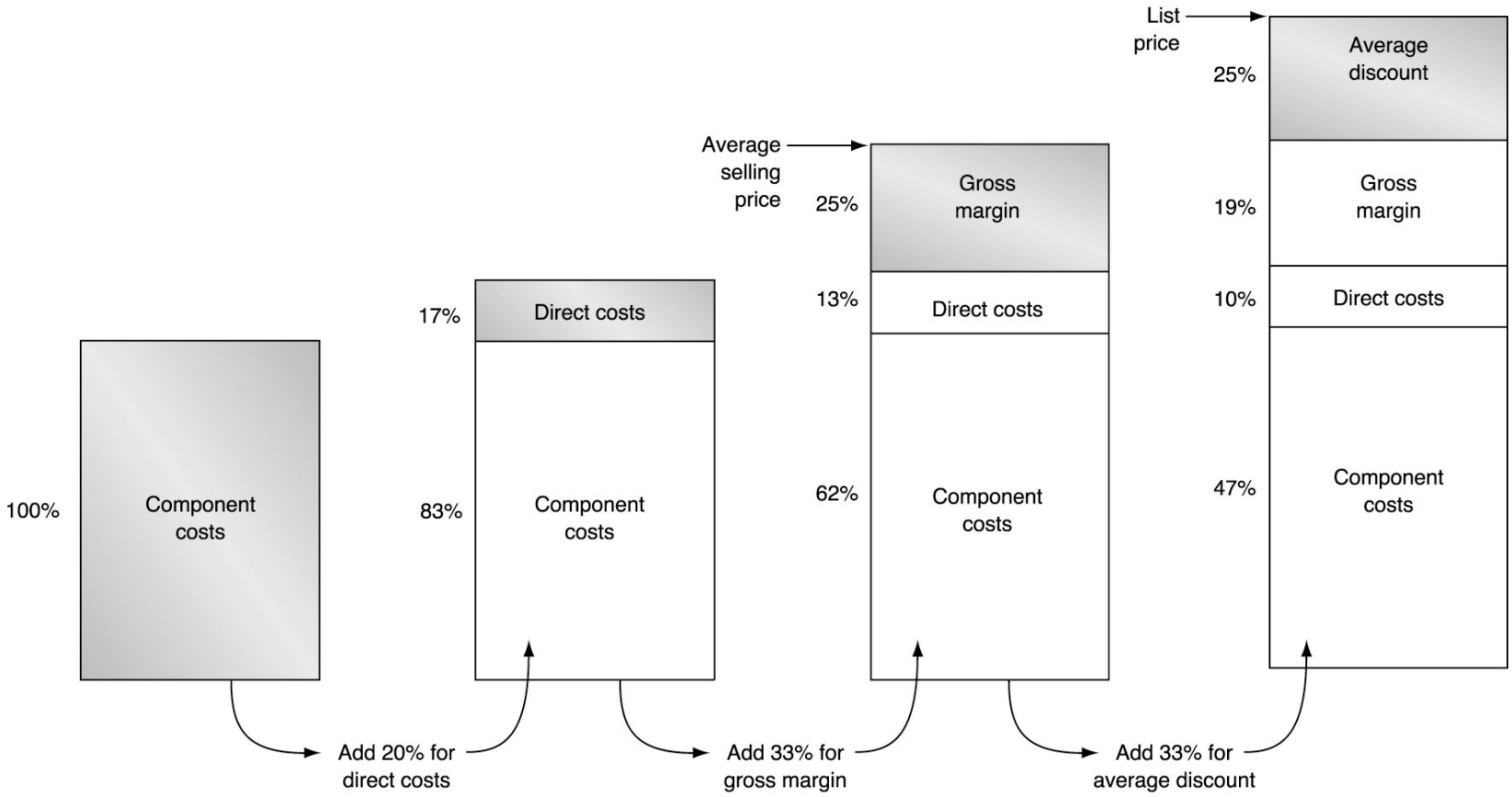
Relative Performance	Die Area	Dies/Wafer	Die Yield	Good Die	$ / Good Die
1	0.64	478	0.69	331	$ 6.03
2	1.23	243	0.51	124	$16.19
4	2.40	120	0.29	35	$56.78

B. Using end of life process (low defect density = 0.4)

Relative Performance	Die Area	Dies/Wafer	Die Yield	Good Die	$ / Good Die
1	0.64	478	0.78	373	$ 5.36
2	1.23	243	0.63	153	$13.08
4	2.40	120	0.42	51	$39.24

Architectural problem: more performance from smaller dies.

- The processor can be 20% of the cost of a desktop system,
- perhaps 40% of the total cost is in the motherboard.
  - It has a huge impact on the price and performance of the system.
- The next item competing for importance is the monitor! (20% of total cost, 40% for all I/O: hard disk, DVD).
- The rest is the cabinet and power supply.



Measuring and Reporting Performance

Problem: if you speak of a car, performance indicators like speed, turning radius, or mileage are measured in physical units. For channels and memory, there are bits and bits/s to measure capacity and bandwidth, but for computers, there is no unit for computation! It is rather hard to define what computation is. As a result, computer performance is always measured relatively to another computer.

There are two aspects:

1. *Response time* (latency) (needed to get a result from the givens).
2. *Throughput* (bandwidth) (how much computation per unit of time).

Relative performance:

$$n = \frac{\text{ExecTime}_\text{Y}}{\text{ExecTime}_\text{X}} = \frac{1/\text{Performance}_\text{Y}}{1/\text{Performance}_\text{X}} = \frac{\text{Performance}_\text{X}}{\text{Performance}_\text{Y}}$$

Different "Times"

Wall-clock time is the *elapsed time* to complete a task. This includes I/O, memory, OS overhead, …, everything. With multiprogramming and multitasking (as in UNIX), for a given task, it changes with the load.

CPU time is the time spent by the CPU on behalf of one task. It is subdivided into *user CPU time* (time spent by the CPU running user code) and *system CPU time* (time spent running OS code on the behalf of the user).

The UNIX time command reports all three. For example

```
prompt$ time sleep 5
0.00u 0.02s 0:05.02 0.3%
```

tells us that the sleep 5 command spent (almost) no CPU time to run, 20 milliseconds to execute OS code and that the elapsed was 5.02 s (per the definition of sleep). It also says that (0.0+0.02)/5.02=0.3% of the elapsed time was to do some work.

What is a task?

We want to be able to predict the performance of a computer: *how do we evaluate performance?*

1. *Real applications*: C compiler (if you are code developer), TeX if you are a typesetter, Photoshop if you are a graphic designer, Spice if you are an electronic engineer, MatLab, and so-on.

2. *Scripted applications*: real applications stripped from I/O.

3. *Kernels*: A general principle about computing (Knuth) is that programs tend to spend most of their time in a very small portion of the code. (For example, the integrator routine in MathLab, searching and sorting while compiling, manipulating matrices in scientific code).

4. *Toy benchmark*s: Small and interesting programs, Sieve of Eratosthenes (prime numbers), Towers of Hanoi, Puzzles, Quicksort.

5. *Synthetic Benchmarks*: Attempt of reproduce the load of a set of programs.

(pp. 29)

SPEC (Standard Performance Evaluation Committee, www.spec.org) is a consortium dedicated to the design of documented benchmarks suites. There are also PC benchmarks (winbench) or the EEMBC benchmarks (www.eembc.org) which permit code tweaking (why not?).

(pp. 30)

Benchmark	Type	Source	Description
gzip	Integer	C	Compression using the Lempel-Ziv algorithm
vpr	Integer	C	FPGA circuit placement and routing
gcc	Integer	C	Consists of the GNU C compiler generating optimized machine code
mcf	Integer	C	Combinatorial optimization of public transit scheduling
crafty	Integer	C	Chess-playing program
parser	Integer	C	Syntactic English language parser
eon	Integer	C++	Graphics visualization using probabilistic ray tracing
perlmbk	Integer	C	Perl (an interpreted string-processing language) with four input scripts
gap	Integer	C	A group theory application package
vortex	Integer	C	An object-oriented database system
bzip2	Integer	C	A block-sorting compression algorithm
twolf	Integer	C	Timberwolf: a simulated annealing algorithm for VLSI place and route
wupwise	FP	F77	Lattice gauge theory model of quantum chromodynamics
swim	FP	F77	Solves shallow water equations using finite difference equations
mgrid	FP	F77	Multigrid solver over three-dimensional field
apply	FP	F77	Parabolic and elliptic partial differential equation solver
mesa	FP	C	Three-dimensional graphics library
galgel	FP	F90	Computational fluid dynamics
art	FP	C	Image recognition of a thermal image using neural networks
equake	FP	C	Simulation of seismic wave propagation
facerec	FP	C	Face recognition using wavelets and graph matching
ammp	FP	C	Molecular dynamics simulation of a protein in water
lucas	FP	F90	Performs primality testing for Mersenne primes
fma3d	FP	F90	Finite element modeling of crash simulation
sixtrack	FP	F77	High-energy physics accelerator design simulation
apsi	FP	F77	A meteorological simulation of pollution distribution

Figure 1.12 The programs in the SPEC CPU2000 benchmark suites. The 11 integer programs (all in C, except one in C++) are used for the CINT2000 measurement, while the 14 floating-point programs (6 in FORTRAN-77, 5 in C, and 3 in FORTRAN-90) are used for the CFP2000 measurement. See *www.spec.org* for more on these benchmarks.

Summarizing Performance

Take *n* programs and average. Arithmetic mean:

$$\text{ArithmeticMean} = \frac{1}{n}\sum_{i=1}^{n}\text{Time}_i$$

or weight them:

$$\sum_{i=1}^{n}\text{Weight}_i \times \text{Time}_i$$

A particular choice of weights equalizes running times:

$$\text{Weight}_i = \frac{1}{\text{Time}_i \times \sum_{i=1}^{n}\left(\frac{1}{\text{Time}_i}\right)}$$

Normalized times to a reference machine can be averaged geometrically:

$$\text{GeometricMean} = \sqrt[n]{\prod_{i=1}^{n}\frac{\text{ExecTime}_i}{\text{ExecTime}_{ref}}}$$, but since

$$\frac{\text{GeometricMean}(X_i)}{\text{GeometricMean}(Y_j)} = \text{GeometricMean}\left(\frac{X_i}{Y_j}\right)$$

the relative results do not depend on the machine taken as reference.

Quantitative Principles of Computer Design

Make the common case fast. This is to help deciding where to allocate a given design (or cost) effort to get maximum results. For example, certain computers load and store 64 bit long numbers is faster that bytes because they are optimized for FP operations rather than business applications. There are hundreds of examples.

Amdahl's law (1967) captures this. It was used to make the case for single CPU processors. Suppose we have an enhancement for a given design.

$$\text{SpeedUp} = \frac{\text{ExecTimeBase}}{\text{ExecTimeEnhanced}}$$

Define *FractionEnhanced*, the fraction of computation time concerned by an enhancement. This fraction is sped up by *SpeedUpEnhanced*.

$$\text{ExecTimeEnhanced} = \text{ExecTimeBase} \times \left(\left(1 - \text{FractionEnhanced}\right) + \frac{\text{FractionEnhanced}}{\text{SpeedUpEnhanced}} \right)$$

$$\text{SpeedUp} = \frac{1}{(1 - \text{FractionEnhanced}) + \frac{\text{FractionEnhanced}}{\text{SpeedUpEnhanced}}}$$

Amdahl's Law Applied

Flying New York to Paris (or Brussels to Montréal).

On an airliner, you have 2 hours check-in then 8 hours flight.

What is speedup if you take the Concorde (about 2 times faster flying)?

Computer example:

Consider the enhancement to a processor for Web serving. New CPU 10 x faster than original for Web serving. Original CPU busy with computation 40% of time and is waiting for I/O 60% of time. What is overall speedup gained by enhancement?

Only spends 40% doing work so limited by that amount.

CPU Performance

Total CPU time for a task (CPU is a clock driven sequential circuit):

$$\mathrm{CPU\_Time} = \mathrm{ClockCycles} \times \mathrm{ClockCycleTime} = \mathrm{ClockCycles} \times \frac{1}{\mathrm{ClockRate}}$$

Programs (tasks) are made of lists of instructions:

$$\mathrm{ClockCyclesPerInstruction} = \mathrm{CPI} = \frac{\mathrm{ClockCycles}}{\mathrm{InstructionCount}}$$

$$\mathrm{CPU\_Time} = \mathrm{InstructionCount} \times \mathrm{CPI} \times \mathrm{ClockCycleTime}$$

$$\mathrm{CPU\_Time} = \frac{\mathrm{Instructions}}{\mathrm{Program}} \times \frac{\mathrm{ClockCycles}}{\mathrm{Instructions}} \times \frac{\mathrm{Seconds}}{\mathrm{ClockCycle}}$$

Significance:

1. First factor is a function of the ISA and of the compiler technology.
2. Second factor is a function of the organization and of the compiler.
3. Third factor is a function of the organization and of the technology.

Faced with a giant tradeoff! The art of computer design is contained in this formula.

(pp. 44—45)

How to improve a factor without affecting the others?

It is useful to breakdown this into more components, i.e. by classes of instructions.

$$\text{ClockCycles} = \sum_{i=1}^{n} \text{IC}_i \times \text{CPI}_i$$

$$\text{CPI} = \sum_{i=1}^{n} \frac{\text{IC}_i}{\text{InstructionCount}} \times \text{CPI}_i$$

(Error in Book!).

IC_i = number of instructions of class i executed in a program

CPI_i = average number of clock cycles/ instruction of class i

which breaks down the problem into the design of classes of instructions.

(e.g. Total CC = Number of FP operations occuring in program x number of clocks for a FP operation + number integer operations x clocks for integer operations …)

This also define the notion of instruction mix, which is the relative frequency of occurrence of classes of instructions (e.g. branches, FP, …), say in a given benchmark.

Ex. gcc

Op	Freq	CPIi		Term
ALU	50%	1		0.5
Load	20%	2		0.4
Store	10%	2		0.2
Branch	20%	2		0.4
			CPI=	1.5

Other Example:

How to measure these numbers?

Clock speed is known, CPU time is also easy to measure.

How to measure fraction of instruction count and CPI?

Software or hardware simulations:

- Designers add counters in the hardware to count important events (such as number of instructions, or number of clock cycles), periodically saved. This has limited practicality and accuracy.

- Instrumented execution by inserting extra instructions in the code (e.g. tracing memory addresses). Gives exact profile.

- Interrupt the processor at random intervals to get statistics.

(pp. 47—48)

Another fundamental principle was observed by Knuth (1960). Most programs are, by definition, highly structured. They rarely use data and instructions in a completely random fashion. The ***principle of locality*** observes that data that are used at close time intervals (*temporal locality*) are also stored close to each other in memory (*spatial locality*) (because of arrays, loop, structures, due generally to how humans turn their thoughts into programs). This drives most of the ideas in computer architecture — we can exploit this knowledge.

Parallelism is related to performing many operations simultaneously. It is applicable to single processors or to memory management as well. In fact, modern CPUs can executes 10's of instructions simultaneously and perform many memory transactions simultaneously. It is applicable to basic circuits (such as carry-look-ahead adders) to entire systems (many CPU or hard drives operating simultaneously).

In an ideal CPU, every transistor would switch in concert with 100 million others, doing something useful. Not only we are far from this ideal, but also far from the system of the some 100 billions neurons firing in our heads!

Pitfall: *The relative performance of two processors with the same ISA can be judged by clock rate* (or by a single benchmark suite).
No! the architecture can differ.

Fallacy: *Benchmarks remain valid through time*.
No! they must be redesigned periodically to account for evolving application and computing technology.

Pitfall: *Hand coded assembly code and compiler generated code can be compared*.
No! Each has its use. Hand code can make use of very special instructions to boost performance in some applications. On the other hand, in many computers, compilers do better jobs at taking advantage of pipelines and other features.

Fallacy: *Peak performance tracks observed performance*.
No! “Peak performance is the performance guaranteed not to be exceeded”. There can be huge gaps.

Fallacy: *The best computer design is the one that optimizes the primary objective without considering implementation*.
No! complex implementations take a long time to develop. Each week of delay is equivalent to 1% loss of performance.

Pitfall: *Neglecting the cost of software*.
Don't!
Pitfall: *Falling prey to Amdahl's Law*.
Measure usage before optimizing.

Fallacy: *Synthetic benchmarks predict real performance*.
No! For example compilers discard useless code and are capable of all kind of code transformations.

Fallacy: *MIPS is an accurate measure of performanc*e.
No! MIPS (= ClockRate / (CPI x 10^6) are ISA *and* program dependent.
e.g. clock: 1 cycle / second.
Example:

FP software: 4 instructions (1 clock each) to perform FP operation (1FP/4s)

= 4 instr/4 sec = 1 instr/sec -> high MIPS

FP Hardware: 1 instruction (2 clocks long) to perform FP operations

= 2 instr/4 sec = 0.5 instructions / sec -> Lower MIPS

even though does more useful work in 4 seconds (completes 2 FP operations/4s)