CMSC 611: Advanced Computer Architecture

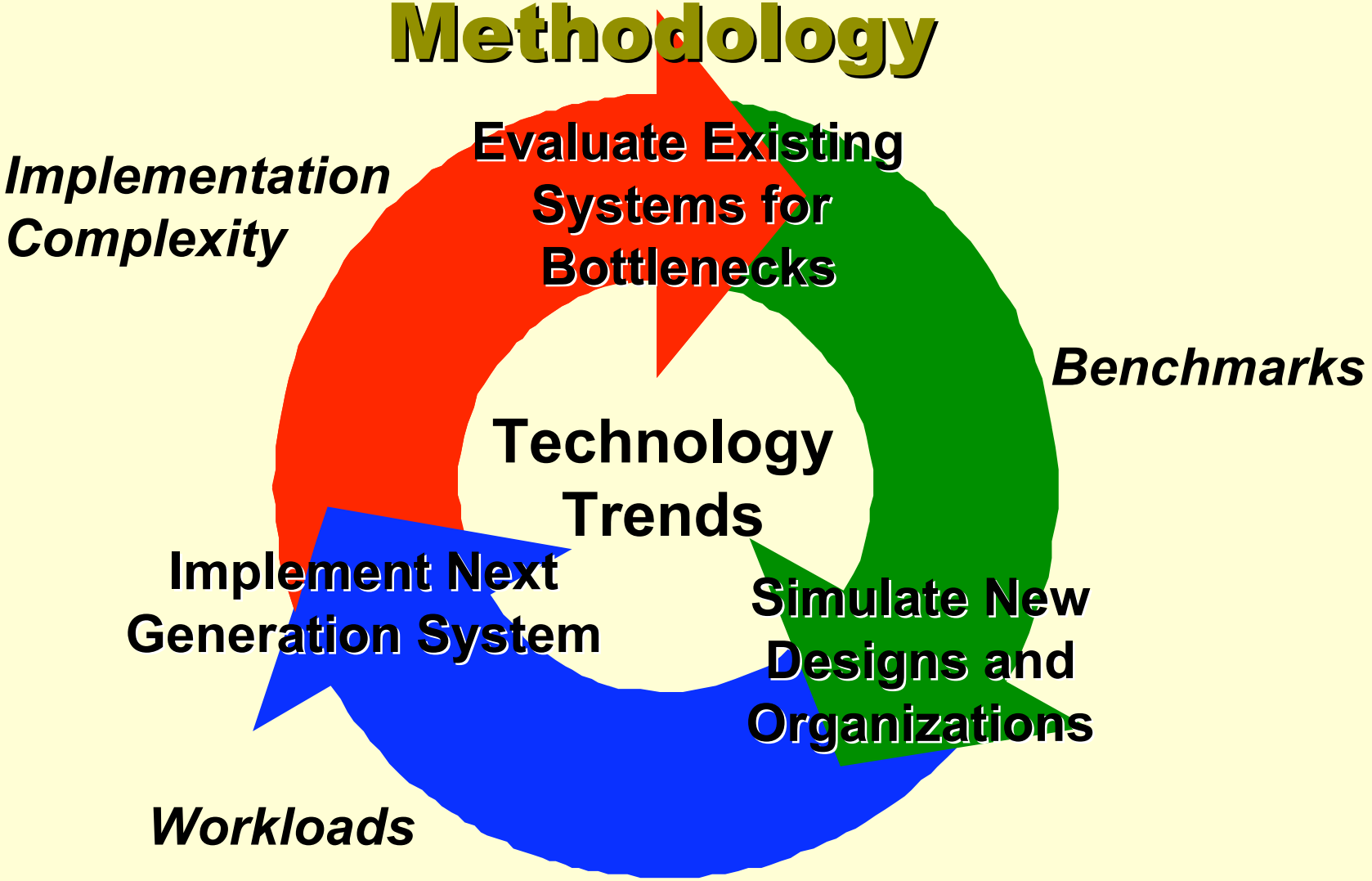
Cost, Performance & Benchmarking

Some material adapted from Mohamed Younis, UMBC CMSC 611 Spr 2003 course slides
Some material adapted from David Culler, UC Berkeley CS252, Spr 2002 course slides, © 2002 UC Berkeley
Some material adapted from Hennessy & Patterson / © 2003 Elsevier Science

Overview

- Previous Lecture
  - What computer architecture
  - Why it is important to study
  - Organization and anatomy of computers
  - Impact of microelectronics technology on computers
  - Evolution and generations of the computer industry
- This Lecture
  - Cost considerations in computer design
  - Why measuring performance is important
  - Different performance metrics
  - Performance comparison

Computer Engineering Methodology



Cost and performance are the main evaluation metrics for a design quality

Slide: Dave Patterson

Circuits

- Need connectors & switches
- Generation defined by switch technology

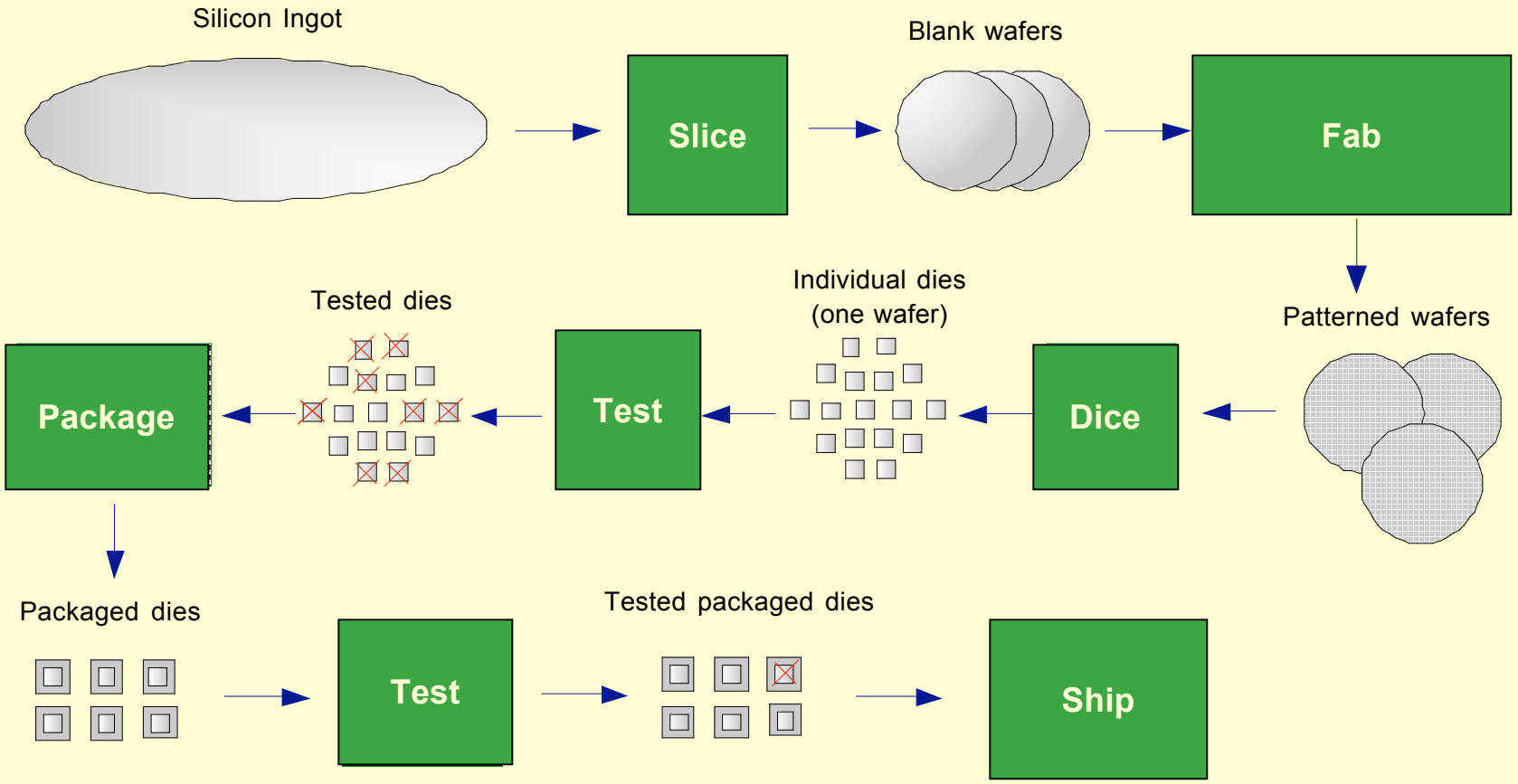
Year	Technology used in computers	Relative performance/unit cost
1951	Vacuum tube	1
1965	Transistor	35
1975	Integrated circuits	900
1995	Very large-scale integrated circuit	2,400,000

Advances of the IC technology affect H/W and S/W design philosophy

Integrated Circuits

- Start with silicon (found in sand)
- Silicon does not conduct electricity well
  - thus semiconductor
- Chemical process can transform tiny areas to
  - Excellent conductors of electricity (like copper)
  - Excellent insulator from electricity (like glass)
  - Areas that can conduct or insulate under a special condition (a switch)
- A transistor is simply an on/off switch controlled by electricity
- Integrated circuits combines dozens to millions of transistors in a chip

Microelectronics Process



Integrated Circuits Costs

$$\text{IC Cost} = \frac{\text{Die Cost} + \text{Testing Cost} + \text{Packing Cost}}{\text{Final Test Yield}}$$

$$\text{Die Cost} = \frac{\text{Wafer Cost}}{\text{Dies per Wafer} \times \text{Die Yield}}$$

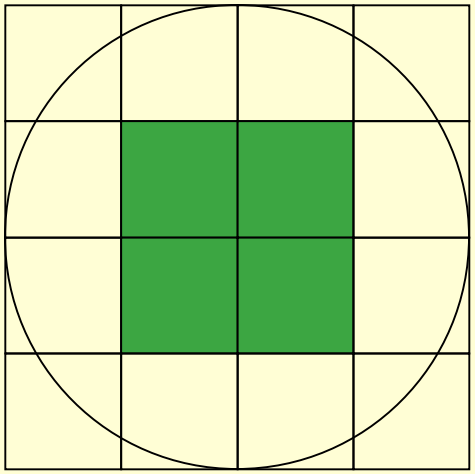
Die cost roughly goes with die area^4

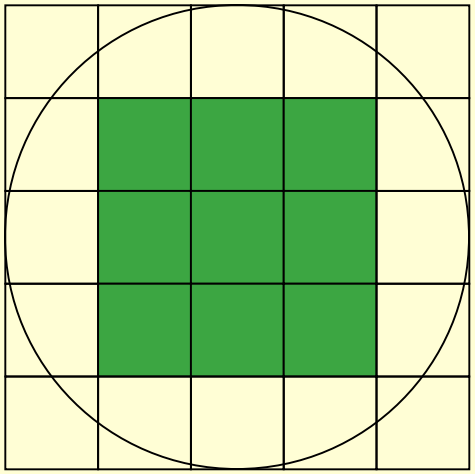
Slide: Dave Patterson

Die Cost

$$\text{Die Cost} = \frac{\text{Wafer Cost}}{\textbf{Dies per Wafer} \times \text{Die Yield}}$$

$$\text{Dies per Wafer} = \frac{\pi \times (\text{Wafer diameter}/2)^2}{\text{Die Area}} - \frac{\pi \times \text{Wafer diameter}}{\sqrt{2 \times \text{Die Area}}}$$





Slide: Dave Patterson

Die Cost

$$\text{Die Cost} = \frac{\text{Wafer Cost}}{\text{Dies per Wafer} \times \textbf{Die Yield}}$$

$$\text{Die Yield} = \text{Wafer yield} \times \left(1 + \frac{\text{Defects per unit area} \times \text{Die Area}}{1}\right)^{-1}$$

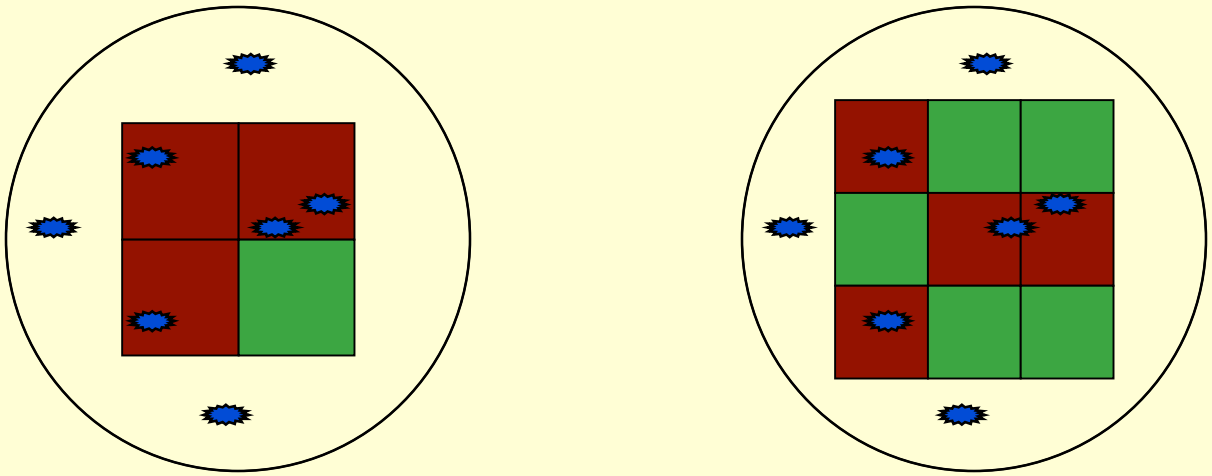


Figure: Dave Patterson

Real World Examples

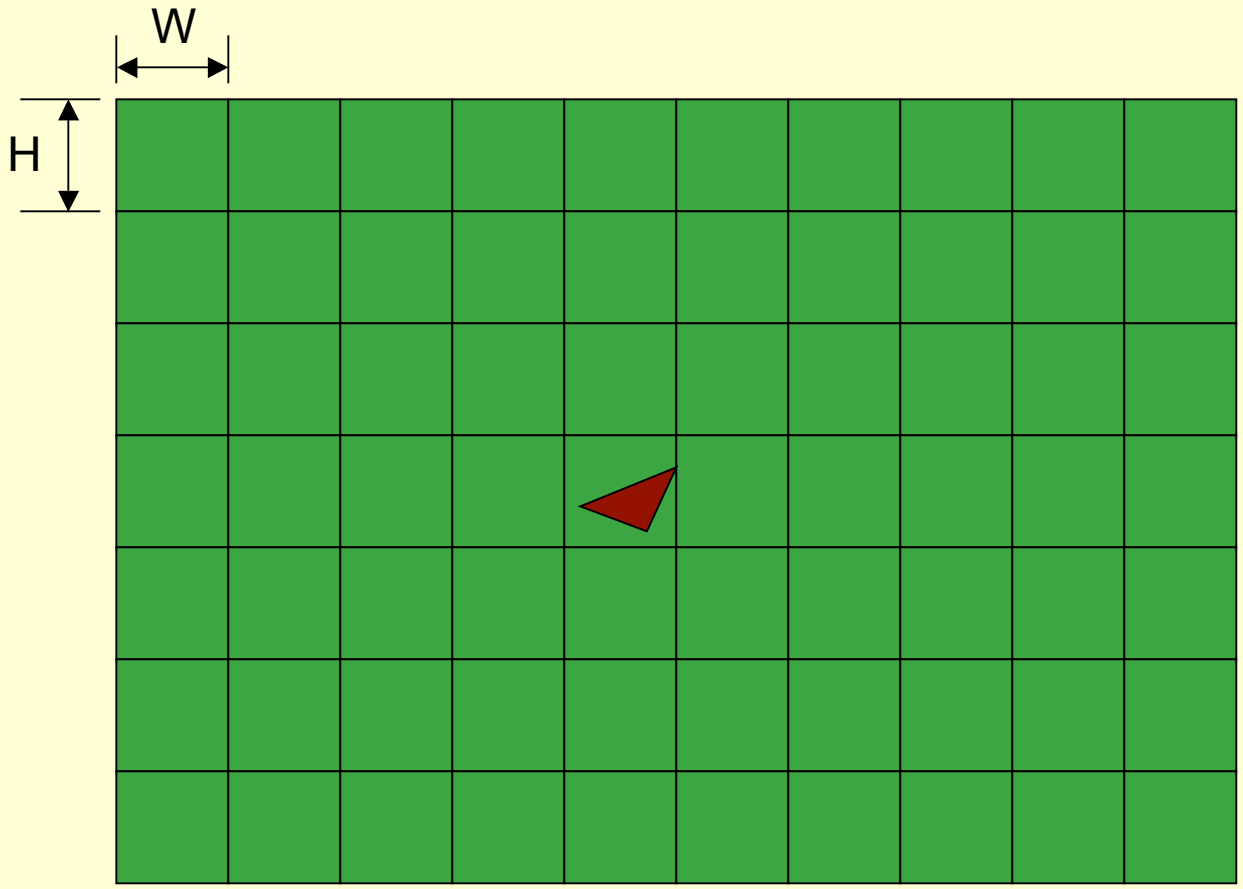
Chip	Layers	Wafer cost	Defect /cm^2	Area (mm^2)	Dies/ Wafer	Yield	Die Cost
386DX	2	$900	1.0	43	360	71%	$4
486DX2	3	$1200	1.0	81	181	54%	$12
PowerPC 601	4	$1700	1.3	121	115	28%	$53
HP PA 7100	3	$1300	1.0	196	66	27%	$73
DEC Alpha	3	$1500	1.2	234	53	19%	$149
SuperSPARC	3	$1700	1.6	234	48	13%	$272
Pentium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

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

Slide: Dave Patterson

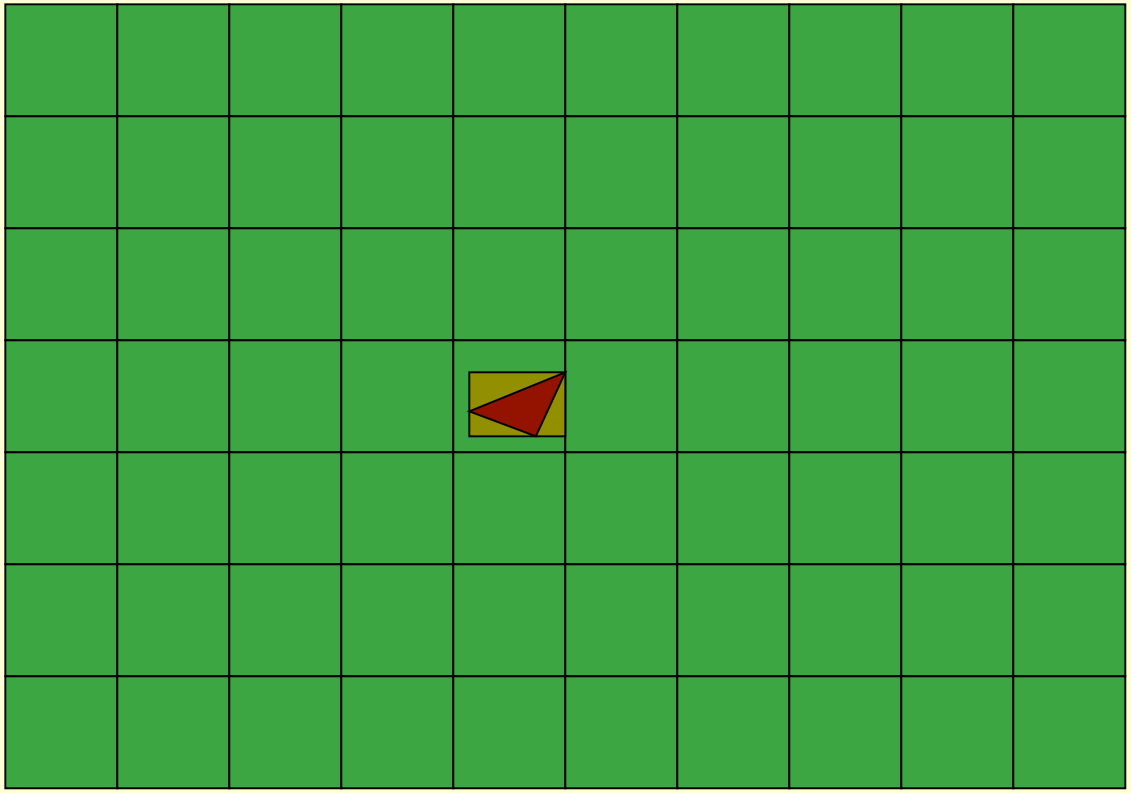
Aside: Geometric Reasoning

- Accelerate triangle rendering by dividing screen into W × H pixel regions (processors)



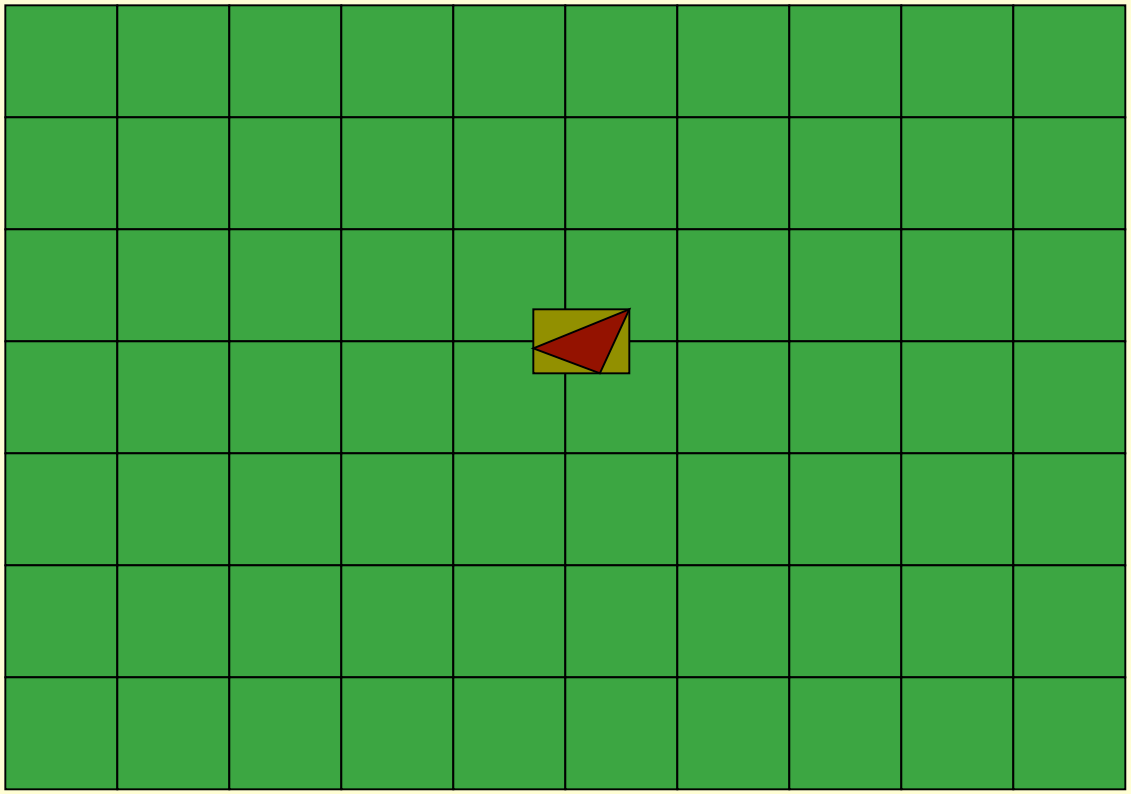
Aside: Geometric Reasoning

- Only render triangle in region hit by triangle bounding box (size w × h)



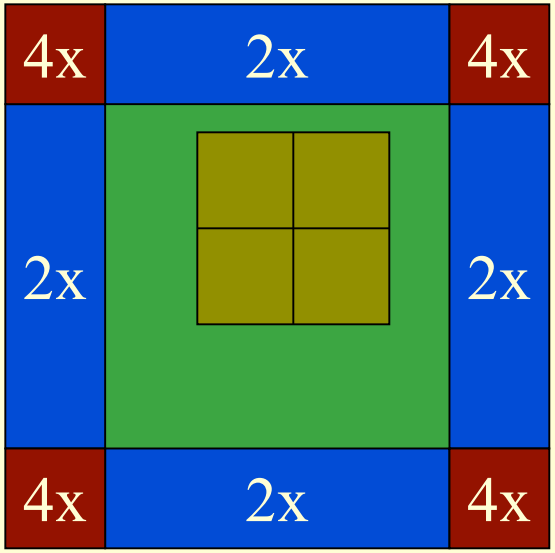
Aside: Geometric Reasoning

- Triangle replicated for each region overlapping triangle bounding box
  - Like having N extra triangles



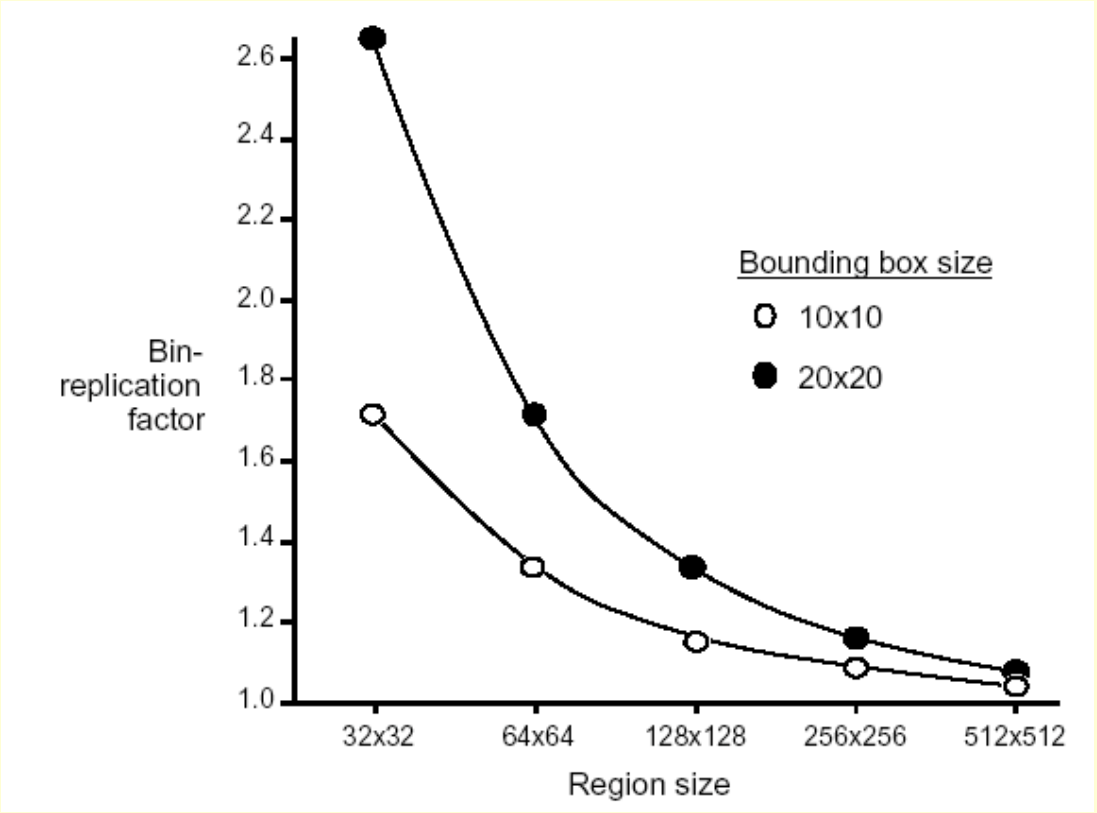
Aside: Geometric Reasoning

- Speedup limited by overlap factor
  - Distribution of box center vs. replication



Eyles formula:

$$\left(\frac{w+W}{W}\right)\left(\frac{h+H}{H}\right)$$



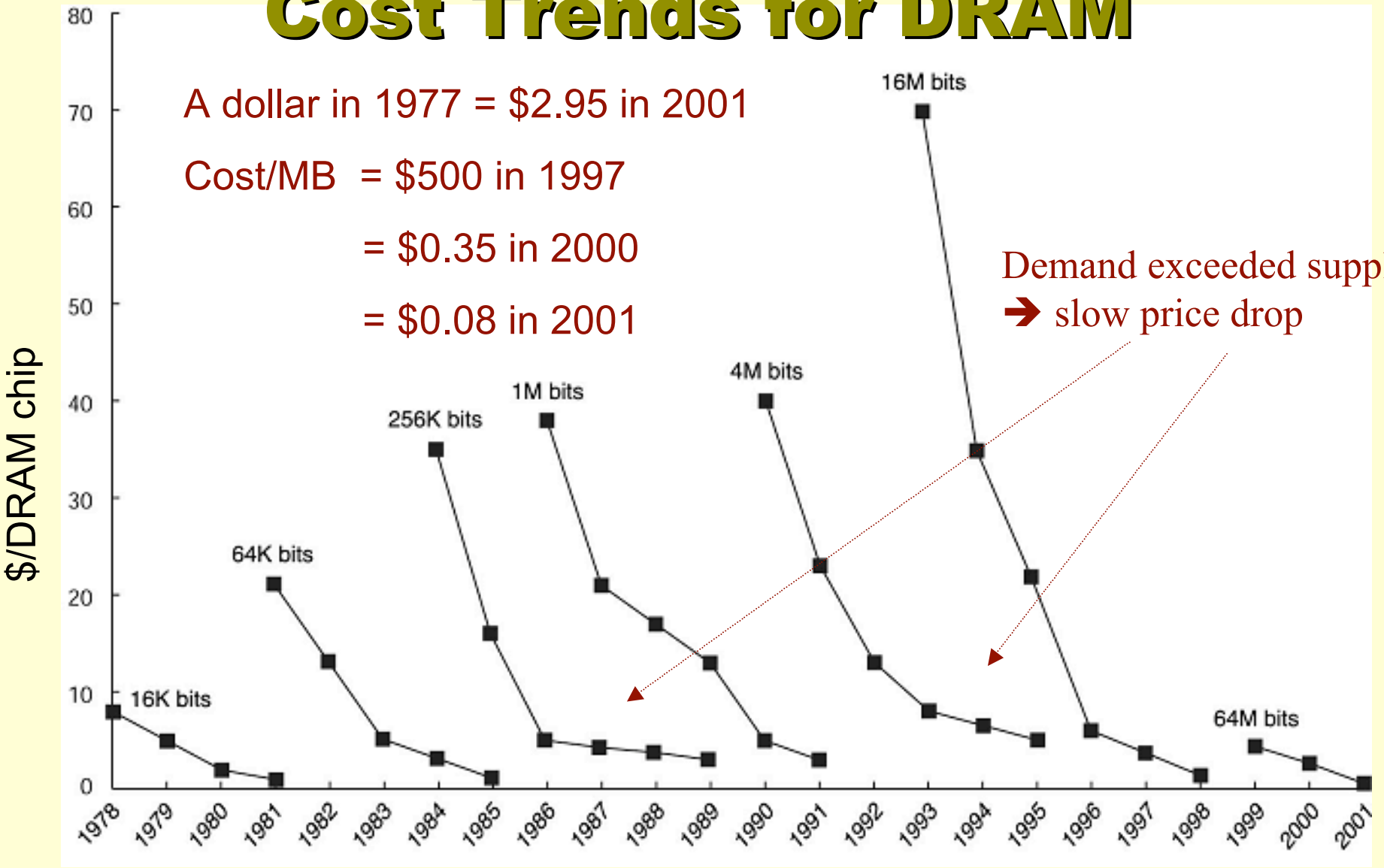
Costs and Trends in Cost

- Understanding trends in component costs (how they will change over time) is an important issue for designers
- Component prices drop over time without major improvements in manufacturing technology

What Affects Cost

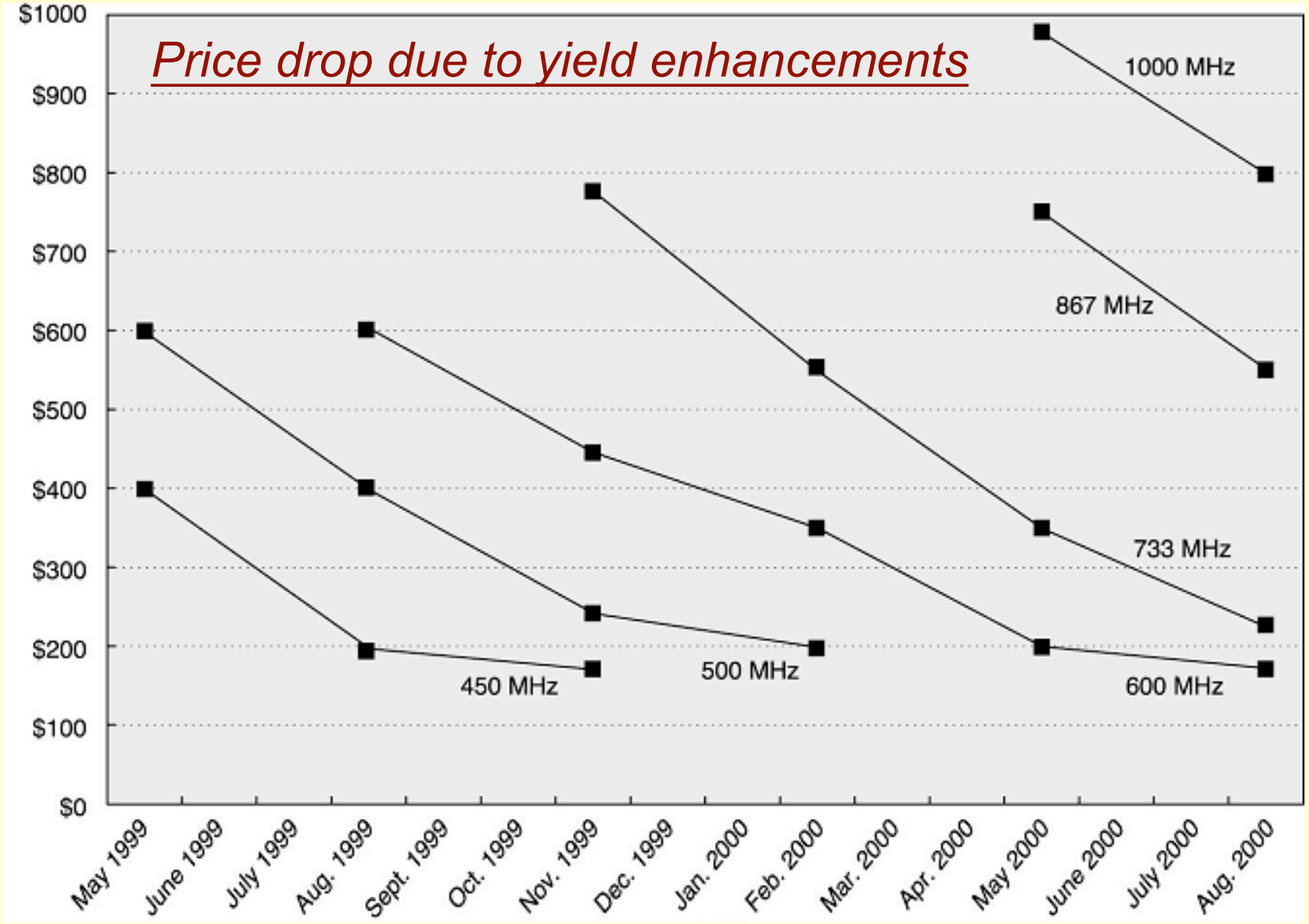
1. Learning curve:
   - The more experience in manufacturing a component, the better the yield
   - In general, a chip, board or system with twice the yield will have half the cost.
   - The learning curve is different for different components, complicating new system design decisions
2. Volume
   - Larger volume increases rate of learning curve and manufacturing efficiency
   - Doubling the volume typically reduces cost by 10%
3. Commodities
   - Essentially identical products sold by multiple vendors in large volumes
   - Foil the competition and drive the efficiency higher and thus the cost down

Cost Trends for DRAM

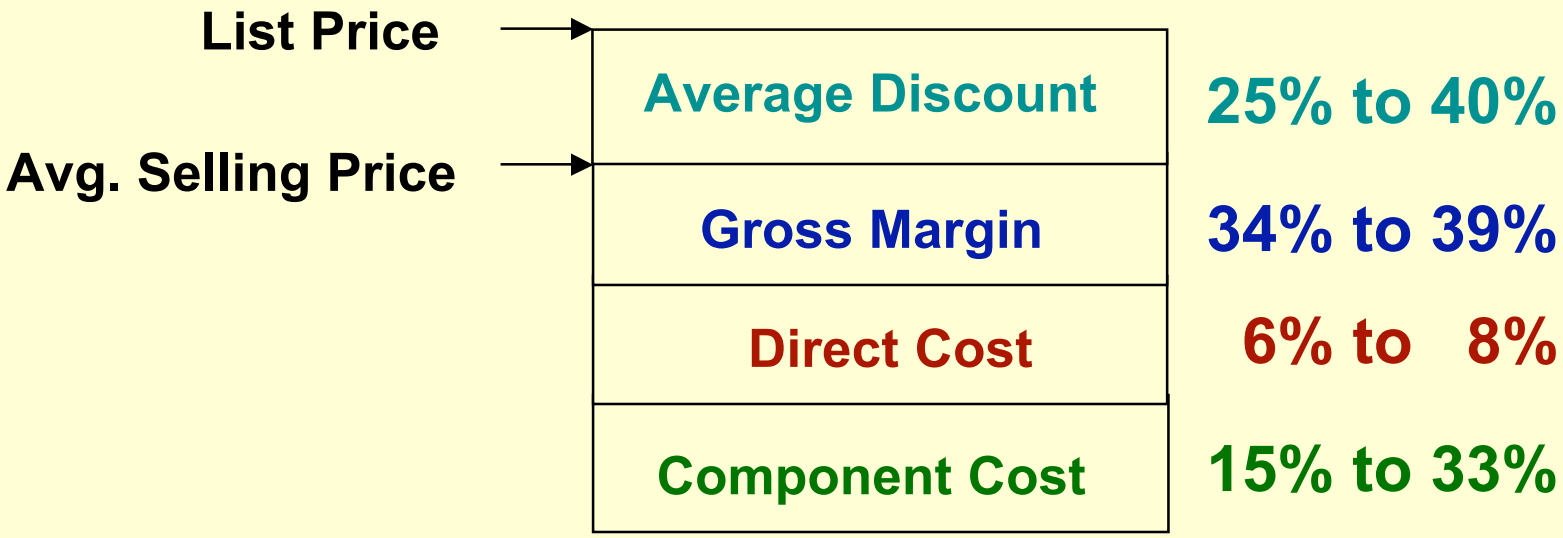


Each generation drops in price by a factor of 10 to 30 over its lifetime

Cost Trends for Processors



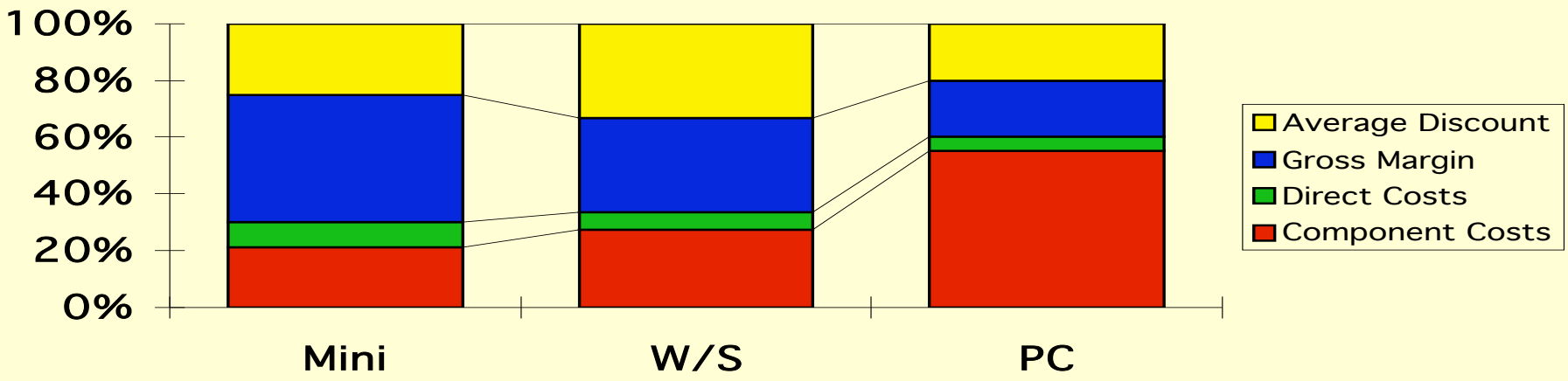
Cost vs. Price



- **Component Cost**
  - Raw material cost for the system's building blocks
- **Direct Cost** (add 25% to 40%)
  - recurring costs: labor, purchasing, scrap, warranty
- **Gross Margin** (add 82% to 186%)
  - nonrecurring costs: R&D, marketing, sales, equipment maintenance, rental, financing cost, pretax profits, taxes
- **Average Discount** (add 33% to 66%)
  - volume discounts and/or retailer markup

Slidei: Dave Patterson

Example: Price vs. Cost



Chip Prices (August 1993) for a volume of 10,000 units

Chip	Area (mm^2)	Total Cost	Price	Comment
386DX	43	$9	$31	
486DX2	81	$35	$245	No Competition
PowerPC 601	121	$77	$280	
DEC Alpha	234	$202	$1231	Recoup R&D?
Pentium	296	$473	$965	

Slide: Dave Patterson

The Role of Performance

- Hardware performance is key to the effectiveness of the entire system
- Performance has to be measured and compared
  - Evaluate various design and technological approaches
- Different types of applications:
  - Different performance metrics may be appropriate
  - Different aspects of a computer system may be most significant
- Factors that affect performance
  - Instruction use and implementation, memory hierarchy, I/O handling

Defining Performance

- Performance means different things to different people
- Analogy from the airline industry:
  - Cruising speed (How fast)
  - Flight range (How far)
  - Passengers (How many)

Airplane	Passenger capacity	Cruising range (miles)	Cruising speed (m.p.h)	Passenger throughput (Passenger □ m.p.h)
Boeing 777	375	4630	610	228,750
Boeing 747	470	4150	610	286,700
BAC/Sud Concorde	132	4000	1350	178,200
Douglas DC-8-50	146	8720	544	79,424

Criteria of performance evaluation differs among users and designers

Performance Metrics

- Response (execution) time:
  - Time between the start and completion of a task
  - Measures user perception of the system speed
  - Common in reactive and time critical systems, single-user computer, etc.
- Throughput:
  - Total number of tasks done in a given time
  - Most relevant to batch processing (billing, credit card processing, etc.)
  - Mainly used for input/output systems (disk access, printer, etc.)

Decreasing response time always improves throughput

Performance Metric Examples

- Replacing the processor of a computer with a faster version
  - Both response time AND throughput
- Adding additional processors to a system that uses multiple processors for separate tasks (e.g. handling of airline reservations system)
  - Throughput but NOT response time

Response-time Metric

- Maximizing performance means minimizing response (execution) time

$$\text{Performance} = \frac{1}{\text{Execution time}}$$

Response-time Metric

- Performance of Processor P_2 is better than P_1 if
  - for a given work load L
  - P_2 takes less time to execute L than P_1

  Performance(P_2) > Performance(P_1) w.r.t. L

  Execution time(P_2) < Execution time(P_1)
- Relative performance: ratio for same workload

$$\text{Speedup} = \frac{\text{Performance } (P_2)}{\text{Performance } (P_1)} = \frac{\text{Execution time } (P_1)}{\text{Execution time } (P_2)}$$

Designer's Performance Metrics

- Users and designers use different metrics
- Designers look at the bottom line of program execution

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

$$= \frac{\text{CPU clock cycles for a program}}{\text{Clock rate}}$$

- To enhance the hardware performance, designers focus on reducing the clock cycle time and the number of cycles per program
- Many techniques to decrease the number of clock cycles also increase the clock cycle time or the average number of cycles per instruction (CPI)

Example

A program runs in 10 seconds on computer "A" with 400 MHz clock.

Want a computer "B" that could run the program in 6 seconds.

Substantial increase in the clock speed possible, but would cause computer "B" to require 1.2 times as many clock cycles as computer "A".

What should be the clock rate of computer "B"?

$$\text{CPU time(A)} = \frac{\text{clock cycles(A)}}{\text{clock rate(A)}} = \frac{\text{clock cycles(A)}}{400 \times 10^6 \text{ cyc/sec}} = 10 \text{ sec}$$

$$\text{clock cycles(A)} = 10 \text{ sec} \times 400 \times 10^6 \text{ cyc/sec} = 4 \times 10^9 \text{ cycles}$$

To get the clock rate of the faster computer, we use the same formula

$$6 \text{ seconds} = \frac{\text{clock cycles(B)}}{\text{clock rate(B)}} = \frac{1.2 \times \text{clock cycles(A)}}{\text{clock rate(B)}} = \frac{4.8 \times 10^9 \text{ cycles}}{\text{clock rate(B)}}$$

$$\text{clock rate(B)} = \frac{4.8 \times 10^6 \text{ cycles}}{6 \text{ second}} = 800 \times 10^6 \text{ cycles/second}$$

Calculation of CPU Time

$$\text{CPU time} = \text{Instruction count} \times \text{CPI} \times \text{Clock cycle time}$$

$$\text{CPU time} = \frac{\text{Instruction count} \times \text{CPI}}{\text{Clock rate}}$$

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

Component of performance	Units of measure
CPU execution time for a program	Seconds for the program
Instruction count	Instructions executed for the program
Clock cycles per instructions (CPI)	Average number of clock cycles/instruction
Clock cycle time	Seconds per clock cycle
Clock rate	Clock cycles per second

CPU Time (Cont.)

- CPU execution time can be measured by running the program
- Clock rate usually published by manufacturer
- Measuring CPI and instruction count non-trivial
- Instruction counts can be measured by
  - software profiling
  - an architecture simulator
  - hardware counters on some architecture
- The CPI depends on many factors including
  - processor structure
  - memory system
  - mix of instruction types
  - implementation of these instructions

CPU Time (Cont.)

- Designers sometimes use the following formula:

$$\text{CPU clock cycles} = \sum_{i=1}^{n} CPI_i \times C_i$$

  - C_i executed of instructions of class i
  - CPI_i average cyc. per instruction in class i
  - n number of instruction classes

Example

We have two implementation of the same instruction set architecture.

Machine "A" has a clock cycle time of 1 ns and CPI of 2.0 for some program.

Machine "B" has a clock cycle time of 2 ns and CPI of 1.2 for the same.

Which machine is faster for this program and by how much?

Both execute the same instructions. Assume number of instructions is "N",

CPU clock cycles (A) = N × 2.0

CPU clock cycles (B) = N × 1.2

CPU time (A) = CPU clock cycles (A) × Clock cycle time (A)
= N × 2.0 × 1 ns = 2 × N ns

CPU time (B) = CPU clock cycles (B) × Clock cycle time (B)
= N × 1.2 × 2 ns = 2.4 × N ns

Therefore machine A will be faster by the following ratio:

$$\frac{\text{CPU Performance (A)}}{\text{CPU Performance (B)}} = \frac{\text{CPU time (B)}}{\text{CPU time (A)}} = \frac{2.4 \times \text{N ns}}{2 \times \text{N ns}} = 1.2$$

Comparing Code Segments

A compiler designer is trying to decide between two code sequences for a particular machine. The hardware designers have supplied the following facts:

Instruction class	CPI for this instruction class
A	1
B	2
C	3

For a particular high-level language statement, the compiler writer is considering two code sequences that require the following instruction counts:

Code sequence	Instruction count for instruction class		
	A	B	C
1	2	1	2
2	4	1	1

Which code sequence executes the fewest instructions? Which will be faster? What is the CPI for each sequence?

Instructions: $\sum C_i$

Sequence 1: 2 + 1 + 2 = 5 instructions ✓

Sequence 2: 4 + 1 + 1 = 6 instructions

Comparing Code Segments

A compiler designer is trying to decide between two code sequences for a particular machine. The hardware designers have supplied the following facts:

Instruction class	CPI for this instruction class
A	1
B	2
C	3

For a particular high-level language statement, the compiler writer is considering two code sequences that require the following instruction counts:

Code sequence	Instruction count for instruction class		
	A	B	C
1	2	1	2
2	4	1	1

Which code sequence executes the fewest instructions? Which will be faster? What is the CPI for each sequence?

Execution time: $\sum CPI_i \times C_i$

Sequence 1: (2×1) + (1×2) + (2×3) = 10 cycles

Sequence 2: (4×1) + (1×2) + (1×3) = 9 cycles ✓

Comparing Code Segments

A compiler designer is trying to decide between two code sequences for a particular machine. The hardware designers have supplied the following facts:

Instruction class	CPI for this instruction class
A	1
B	2
C	3

For a particular high-level language statement, the compiler writer is considering two code sequences that require the following instruction counts:

Code sequence	Instruction count for instruction class		
	A	B	C
1	2	1	2
2	4	1	1

Which code sequence executes the most instructions? Which will be faster? What is the CPI for each sequence?

CPI: CPU clock cycles/Instruction count

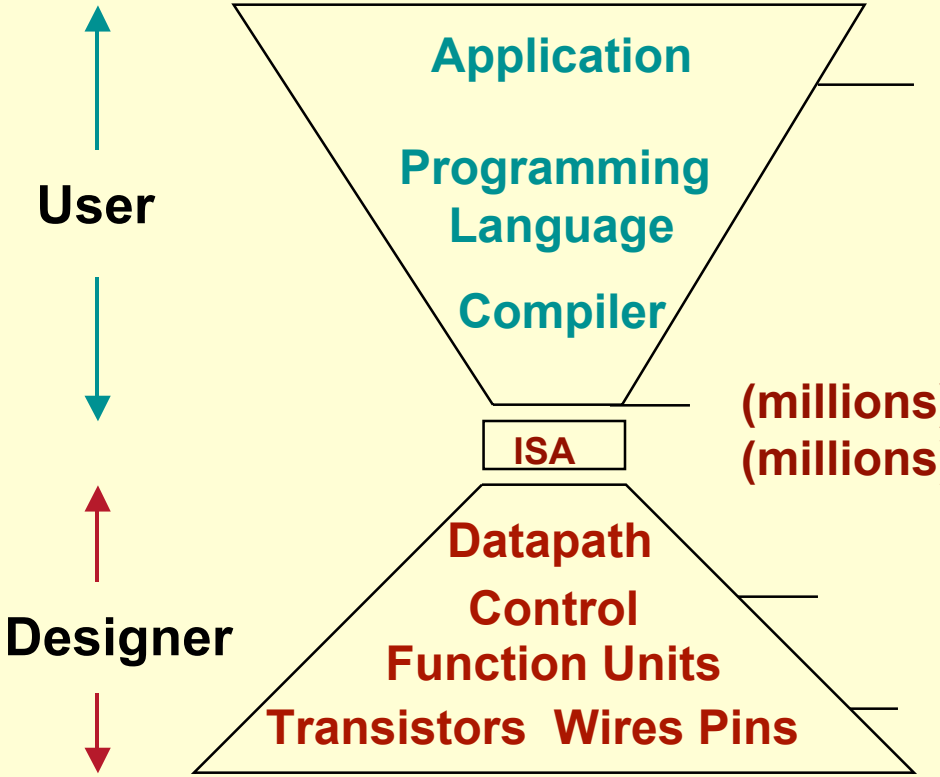
Sequence 1: 10/5 = 2 cycles per instruction

Sequence 2: 9/6 = 1.5 cycles per instruction

The Role of Performance

- Hardware performance is key to the effectiveness of the entire system
- Performance has to be measured and compared
  - Evaluate various design and technological approaches
- Different types of applications:
  - Different performance metrics may be appropriate
  - Different aspects of a computer system may be most significant
- Factors that affect performance
  - Instruction use and implementation, memory hierarchy, I/O handling

Metrics of Performance



Operations per second

(millions) of Instructions per second: MIPS
(millions) of (FP) operations per second: MFLOP/s

Megabytes per second

Cycles per second (clock rate)

➡ Maximizing performance means minimizing response (execution) time

$$\text{Performance} = \frac{1}{\text{Execution time}}$$

Figure: Dave Patterson

Calculation of CPU Time

$$\text{CPU time} = \frac{\text{Instruction count} \times \text{CPI}}{\text{Clock rate}}$$

	Instr. Count	CPI	Clock Rate
Program	**X**		
Compiler	**X**	**X**	
Instruction Set	**X**	**X**	
Organization		**X**	**X**
Technology			**X**

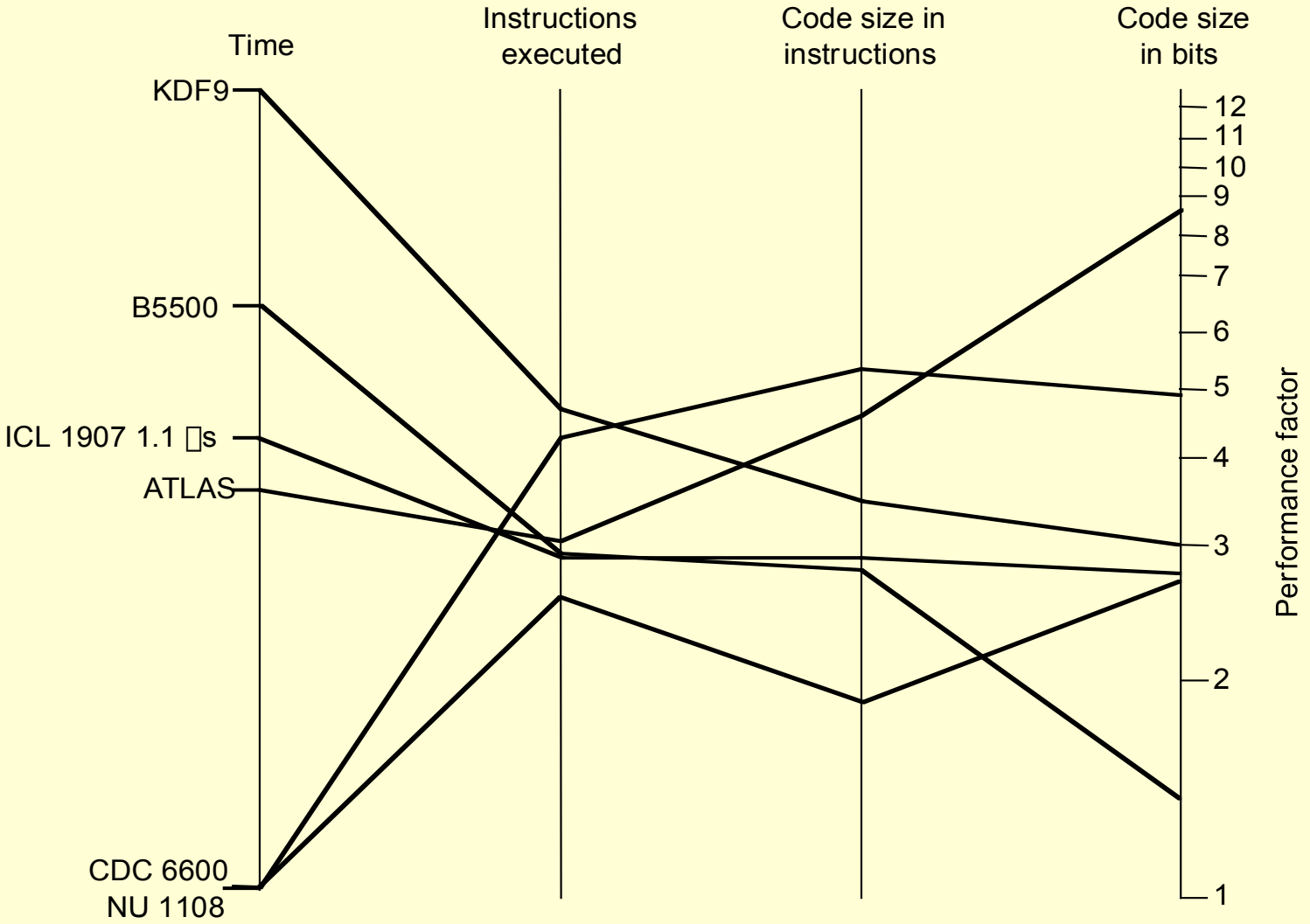
$$\text{CPU clock cycles} = \sum_{i=1}^{n} CPI_i \times C_i$$

Where: C_i is the count of number of instructions of class *i* executed

CPI_i is the average number of cycles per instruction for that instruction class

n is the number of different instruction classes

Can Hardware-Indep Metrics Predict Performance?



Performance Reports

Hardware	
Model number	Powerstation 550
CPU	41.67-MHz POWER 4164
FPU (floating point)	Integrated
Number of CPU	1
Cache size per CPU	64K data/8k instruction
Memory	64 MB
Disk subsystem	2 400-MB SCSI
Network interface	N/A
Software	
OS type and revision	AIX Ver. 3.1.5
Compiler revision	AIX XL C/6000 Ver. 1.1.5 AIX XL Fortran Ver. 2.2
Other software	None
File system type	AIX
Firmware level	N/A
System	
Tuning parameters	None
Background load	None
System state	Multi-user (single-user login)

Guiding principle is *reproducibility* (report environment & experiments setup)

Comparing & Summarizing Performance

- Wrong summary can be confusing
  - A 10x B or B 10x A?
- Total execution time is a consistent measure
- Relative execution times for the same workload can be informative

	Computer A	Computer B
Program 1 (seconds)	1	10
Program 2 (seconds)	1000	100
Total time (seconds)	1001	110

$$\frac{\text{CPU Performance (B)}}{\text{CPU Performance (A)}} = \frac{\text{Total execution time (A)}}{\text{Total execution time (B)}} = \frac{1001}{110} = 9.1$$

Execution time is the only valid and unimpeachable measure of performance

Performance Summary (Cont.)

$$\text{Arithmetic Mean (AM)} = \frac{1}{n}\sum_{i=1}^{n} \text{Execution\_Time}_i$$

$$\text{Weighted Arithmetic Mean (WAM)} = \sum_{i=1}^{n} w_i \times \text{Execution\_Time}_i \qquad \sum w_i = 1;\ w_i \geq 0$$

	Time on A	Time on B	Norm. to A		Norm. to B	
			A	B	A	B
Program 1	1	10	1	10	0.1	1
Program 2	1000	100	1	0.1	10	1
AM of time or normalized time	500.5	55	1	5.05	5.05	1

- Weighted arithmetic mean summarizes performance while tracking execution time
- Weights can adjust for different running times, balancing the contribution of each benchmark
- Never use AM for normalizing execution time relative to a reference machine

Performance Summary (Cont.)

- Geometric mean is suitable for reporting average normalized execution time

$$\text{Geometric Mean (GM)} = \sqrt[n]{\prod_{i=1}^{n} \text{Execution\_Time\_ratio}_i}$$

$$\frac{\text{Geometric Mean}(X_i)}{\text{Geometric Mean}(Y_i)} = \text{Geometric Mean}\left(\frac{X_i}{Y_i}\right)$$

	Time on A	Time on B	Norm. to A		Norm. to B	
			A	B	A	B
Program 1	1	10	1	10	0.1	1
Program 2	1000	100	1	0.1	10	1
AM of time or normalized time	500.5	55	1	5.05	5.05	1
GM of time or normalized time	31.62	31.62	1	1	1	1

Amdahl's Law

- A common theme in hardware design is to make the common case fast
  - Increasing the clock rate would not affect memory access time
  - Using a floating point processing unit does not speed integer ALU operations

The performance enhancement possible with a given improvement is limited by the amount that the improved feature is used

$$\text{Execution time after improvement} = \frac{\text{Original execution time affected by the improvement}}{\text{Amount of improvement}} + \text{Execution time unaffected}$$

Amdahl's Law

Execution time after improvement =

$$\frac{\text{Original execution time affected by the improvement}}{\text{Amount of improvement}} + \text{Execution time unaffected}$$

Example: Floating point instructions improved to run 2X;

but only 10% of actual instructions are floating point

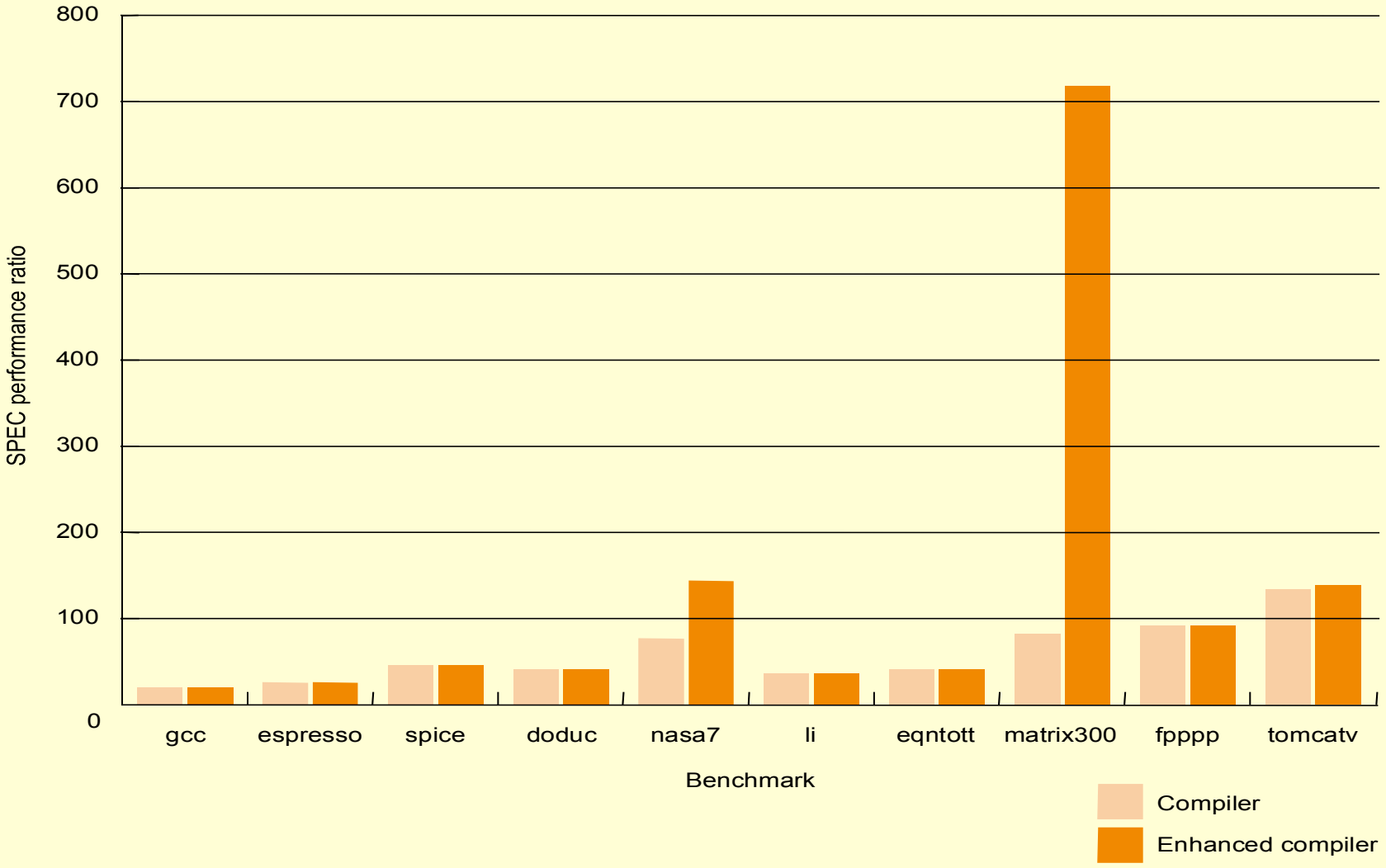
$$\textit{Exec-Time}_{new} = \textit{Exec-Time}_{old} \times (0.9 + .1/2) = 0.95 \times \textit{Exec-Time}_{old}$$

$$\textit{Speedup}_{overall} = \textit{Exec-Time}_{new} / \textit{Exec-Time}_{old} = 1/0.95 = 1.053$$

Performance Benchmarks

- Many widely-used benchmarks are small programs that have significant locality of instruction and data reference
- Universal benchmarks can be misleading since hardware and compiler vendors might optimize for ONLY these programs
- The best types of benchmarks are real applications — reflect end-user interest
- Architectures might perform well for some applications and poorly for others
- Compilation can boost performance by taking advantage of architecture-specific features
- Application-specific compiler optimization are becoming more popular

Effect of Compilation



App. and arch. specific optimization can dramatically impact performance

The SPEC Benchmarks

- **S**tandard **P**erformance **E**valuation **C**orporation
- Suite of benchmarks by a set of companies
  - improve measurement and reporting of CPU performance
- SPEC CPU2000 is the latest suite (for CPU)
  - 12 integer programs (written in C)
  - 14 floating-point (Fortran 77) programs
- Customized SPEC suites for other areas
  - graphics, mail, web, JVM, …

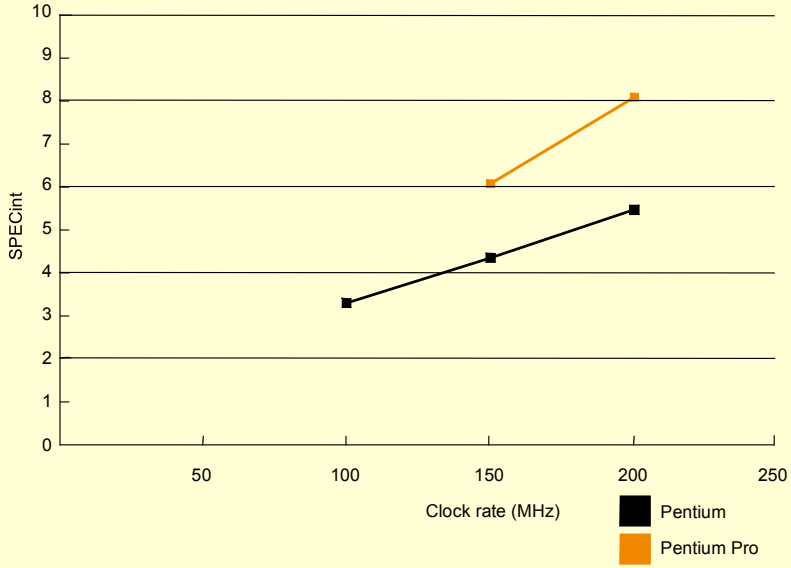
The SPEC Benchmarks

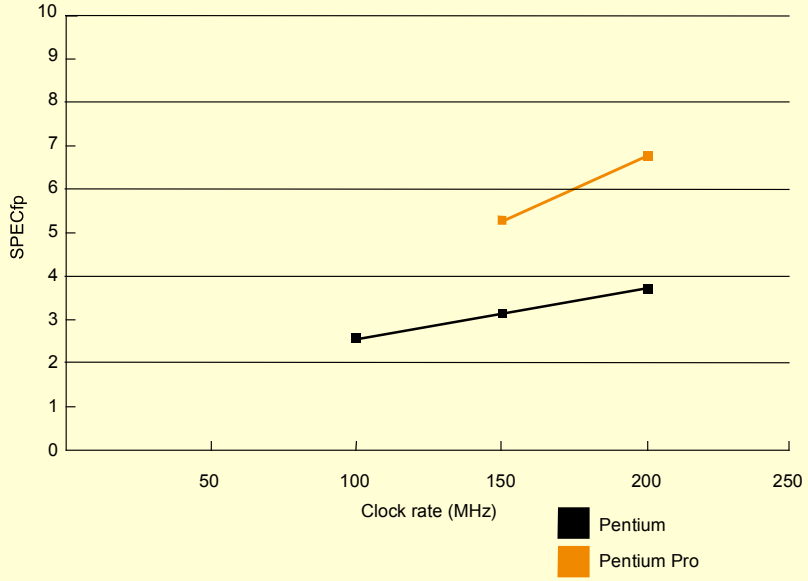
- Requires running applications on real hardware
  - memory system has a significant effect
- Report must include exact configuration

$$\text{SPEC ratio} = \frac{\text{Execution time on SUN SPARCstation 10/40}}{\text{Execution time on the measure machine}}$$

- Bigger numeric values of the SPEC ratio indicate faster machine (performance = 1/execution time)

SPEC95 for Pentium and Pentium Pro





- Comments & Observations:
  - The performance measured may be different on otherwise identical HW with different memory systems or compilers
  - At the same clock rate, the SPECint95 shows Pentium Pro 1.4-1.5 times faster / SPECfp95 shows 1.7-1.8 times faster
    - mostly due to enhanced internal architecture
  - Processor performance increase low relative to clock increase
    - due to memory system
  - Large applications are more sensitive to memory system

MIPS as a Performance Metric

- MIPS = Million Instructions Per Second
  - one of the simplest metrics
  - valid in a limited context

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

- There are three problems with MIPS:
  - MIPS does not account for instruction capabilities
  - MIPS can vary between programs on the same computer
  - MIPS can vary inversely with performance (see next example)

The use of MIPS is simple and intuitive, faster machines have bigger MIPS

Example

Consider the machine with the following three instruction classes and CPI:

Instruction class	CPI for this instruction class
A	1
B	2
C	3

Suppose we measure the code for the same program from two compilers:

Code from	Instruction count in (billions) for each instruction class		
	A	B	C
Compiler 1	5	1	1
Compiler 2	10	1	1

Assume that the machine's clock rate is 500 MHz. Which code sequence will execute faster according to execution time? According to MIPS?

Execution time: $\text{Exection time} = \frac{\text{CPU clock cycles}}{\text{Clock rate}}$; $\text{CPU clock cycles} = \sum_{i=1}^{n} \text{CPI}_i \times C_i$

Sequence 1: CPU clock cycles = $(5 \times 1 + 1 \times 2 + 1 \times 3) \times 10^9 = 10 \times 10^9$ cyc.
Execution time = $(10 \times 10^9) / (500 \times 10^6)$ = 20 seconds

Sequence 2: CPU clock cycles = $(10 \times 1 + 1 \times 2 + 1 \times 3) \times 10^9 = 15 \times 10^9$ cyc.
Execution time = $(15 \times 10^9) / (500 \times 10^6)$ = 30 seconds

Example

Consider the machine with the following three instruction classes and CPI:

Instruction class	CPI for this instruction class
A	1
B	2
C	3

Suppose we measure the code for the same program from two compilers:

Code from	Instruction count in (billions) for each instruction class		
	A	B	C
Compiler 1	5	1	1
Compiler 2	10	1	1

Assume that the machine's clock rate is 500 MHz. Which code sequence will execute faster according to MIPS? According to execution time?

MIPS:

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

Sequence 1: (5+1+1) / (20 × 10^6) = 350

Sequence 2: (10+1+1) / (30 × 10^6) = 400

Native, Peak & Relative MIPS

- Peak MIPS:
  - Choose instruction mix that maximizes the CPI
- Relative MIPS:
  - Compares machines to an agreed-upon reference machine (e.g. Vax 11/780)
  - Comparisons even with different instruction sets
  - However, reference machine may become obsolete and no longer exist!
- Relative MIPS is practical for evolving design of the same computer

$$\text{Relative MIPS} = \frac{\text{Execution time}_{\text{reference}}}{\text{Execution time}_{\text{unrated}}} \times \text{MIPS}_{\text{reference}}$$

Synthetic Benchmarks

- Artificial programs that are constructed to match the characteristics of large set of programs
- Whetstone & Dhrystone popular
  - Whetstone (scientific programs in Algol → Fortran)
    - "Whetstones per second" – the number of executions of one iteration of the whetstone benchmark
  - Dhrystone (systems programs in Ada → C)

Synthetic Benchmarks

- Synthetic benchmarks suffer the following drawbacks:
  1. They may not reflect the user interest since they are not real applications
  2. They do not reflect real program behavior (e.g. memory access pattern)
  3. Compiler and hardware can inflate the performance of these programs far beyond what the same optimization can achieve for real-programs

Final Remarks

- Designing for performance only without considering cost is unrealistic
  - For supercomputing performance is the primary and dominant goal
  - Low-end personal and embedded computers are extremely cost driven
- Performance depends on three major factors
  - number of instructions,
  - cycles consumed by instruction execution
  - clock rate

The art of computer design lies not in plugging numbers in a performance equation, but in accurately determining how design alternatives will affect performance and cost

Conclusion

- Summary
  - Performance reports, summary and comparison

  (reproducibility, arithmetic and weighted arithmetic means)
  - Widely used benchmark programs

  (SPEC, Whetstone and Dhrystone)
  - Example industry metrics

  (e.g. MIPS, MFLOP, etc.)
  - Increasing CPU performance can come from three sources
    1. Increases in clock rate
    2. Improvement in processor utilization that lower the CPI
    3. Compiler enhancement that lower the instruction count or generate instructions with lower CPI
- Next Lecture: Instruction set architecture