

Chapter 1-2



Question1

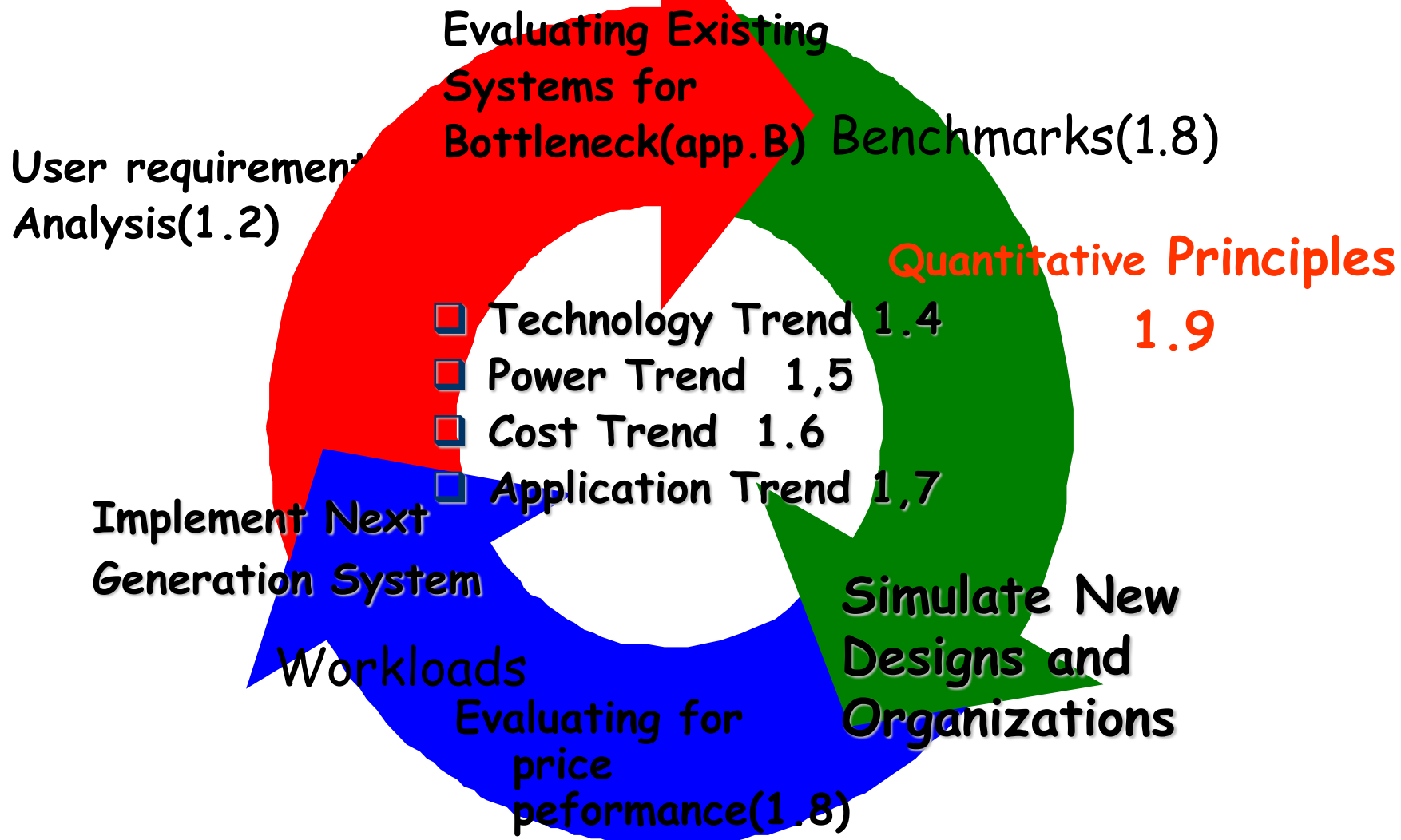
() 1. From the view of computer architecture, the machine attributes that assembler programmer can see are _____.

- A. the functions to be accomplished by computer software.
- B. the organization of all computer hardware.
- C. the hardware abstraction to be used when programming.
- D. the hardware implementation of the various computer components

Question2

- ()2 . Which of the following descriptions about “Computer architecture” is true?
- A. Computer architecture refers only to instruction set architecture design.
 - B. Computer architecture means the implementation of a machine, which has two components: organization and hardware.
 - C. Computer architecture design doesn't need to provide support to compilers.
 - D. Computer architecture is intended to cover all three aspects of computer design ---- instruction set architecture, organization and hardware.

Computer Design Engineering life cycle



Topics in Chapter

1.1 Introduction

1.2 Classes of computers

1.3 Defining computer architecture and What's the task of computer design?

1.4 Trends in Technology

1.5 Trends in power in Integrated circuits

1.6 Trends in Cost

1.7 Dependability

1.8 Measuring, Reporting and summarizing Perf.

1.9 Quantitative Principles of computer Design

1.10 Putting it altogether

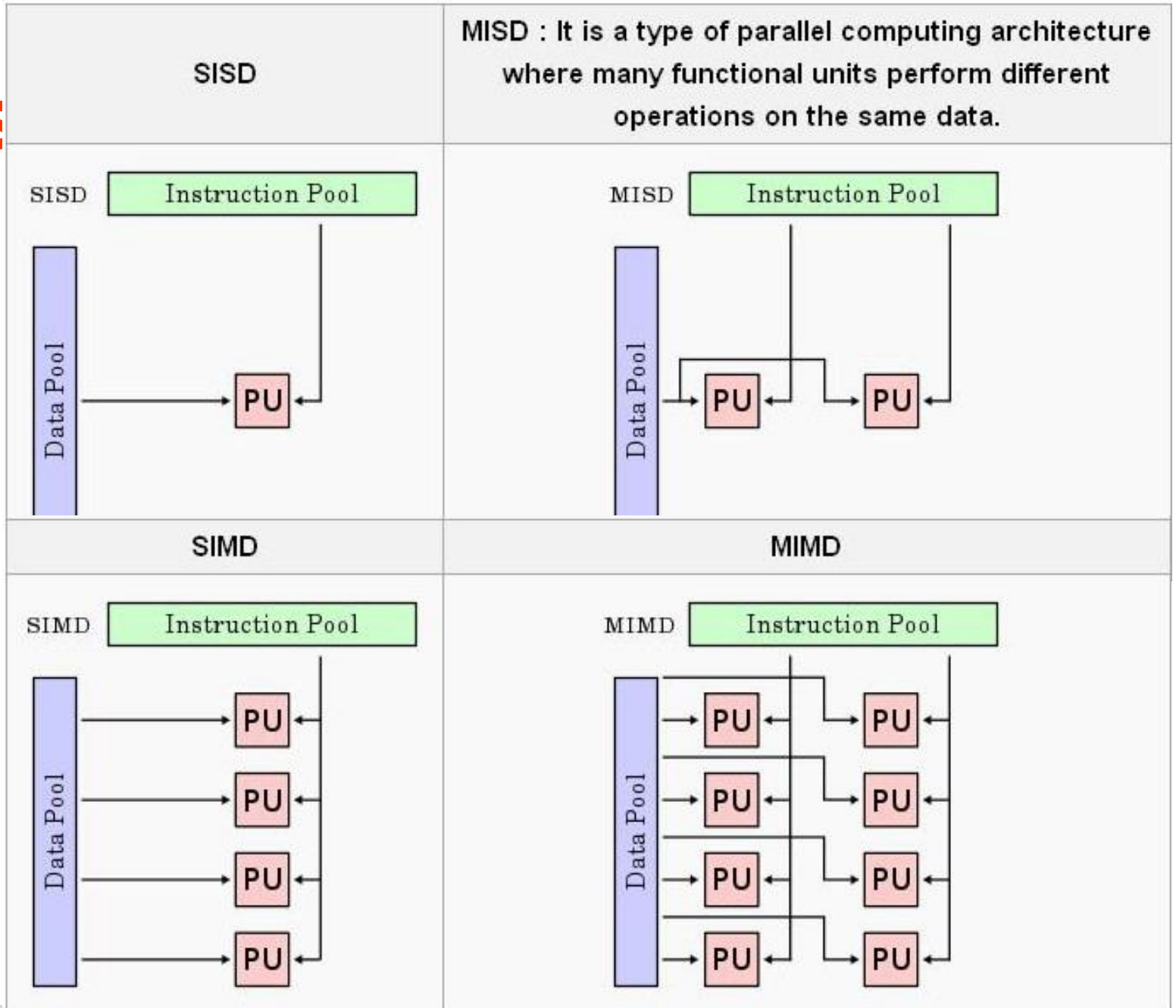
Classes of computers

• **Flynn's Taxonomy:** A classification of computer architectures based on the number of streams of instructions and data

- **SISD (Single Instruction Single Data)**
 - Uniprocessors
- **MISD (Multiple Instruction Single Data)**
 - ???
- **SIMD (Single Instruction Multiple Data)**
 - Examples: Illiac-IV, CM-2
 - » Simple programming model
 - » Low overhead
 - » Flexibility
 - » All custom
- **MIMD (Multiple Instruction Multiple Data)**
 - Examples: SPARCCenter, T3D
 - » Flexible
 - » *Use off-the-shelf micros*

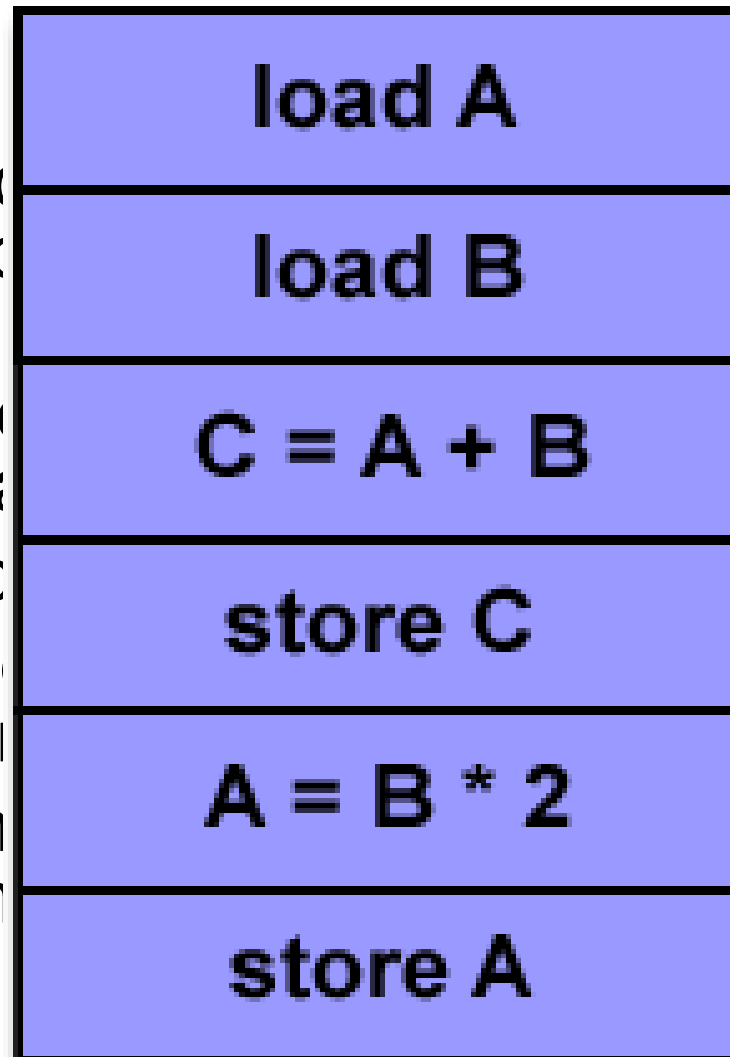


clas



SISD

- A serial (non
- Single instruction being acted on each cycle
- Single data: one input during a cycle
- Deterministic
- This is the oldest prevalent form
- Examples: microprocessors and mainframes



team is
the clock

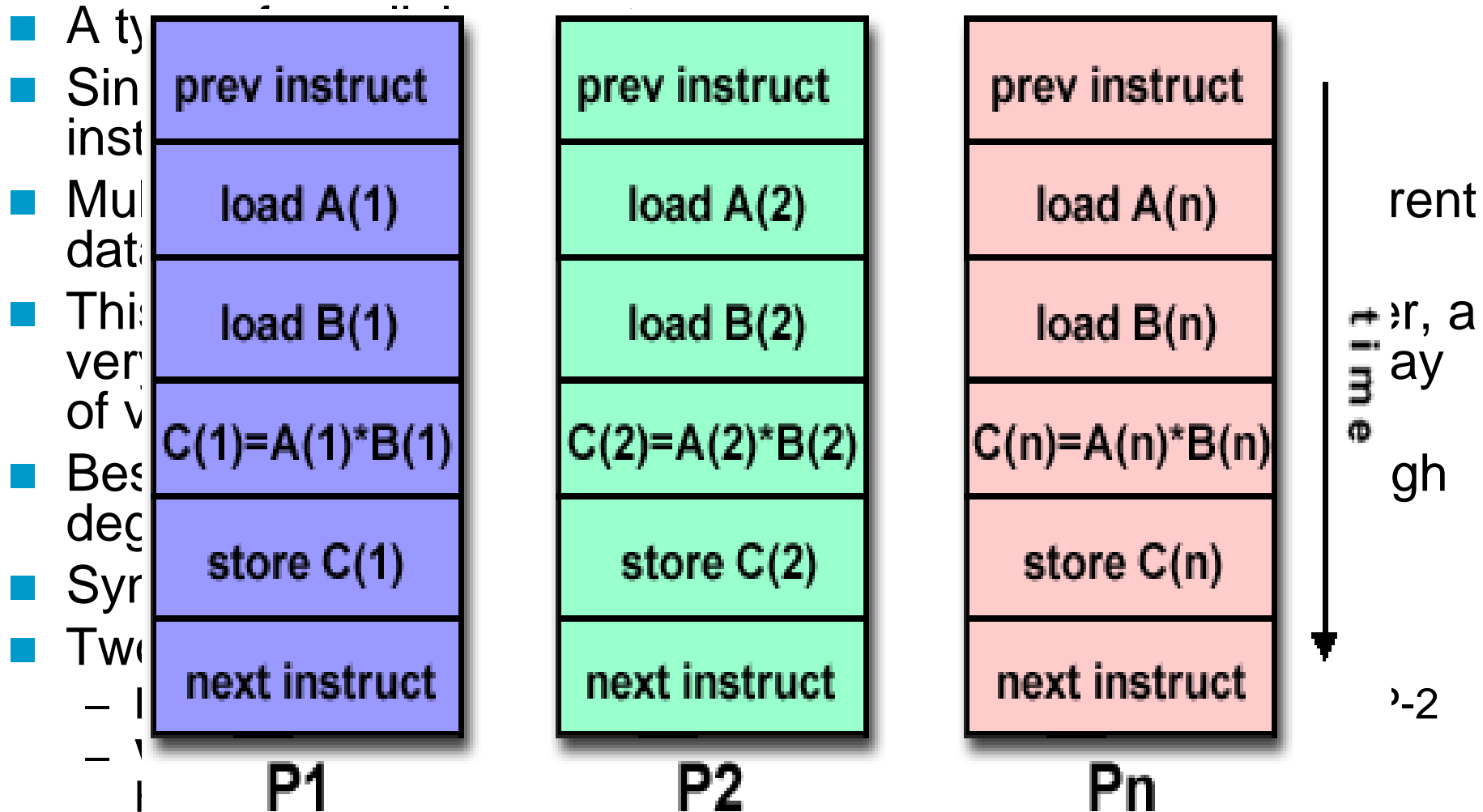
used as

time

lost

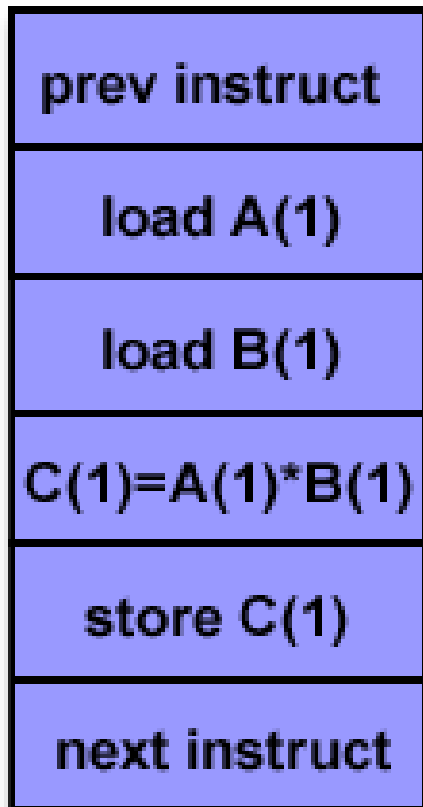
ations

SIMD

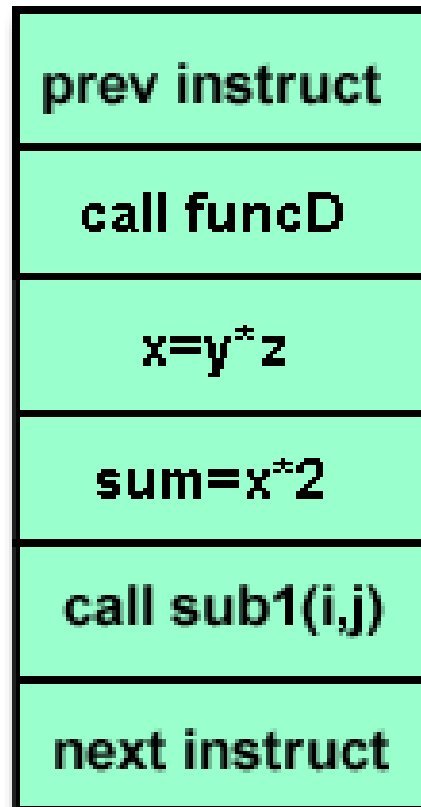


MIMD

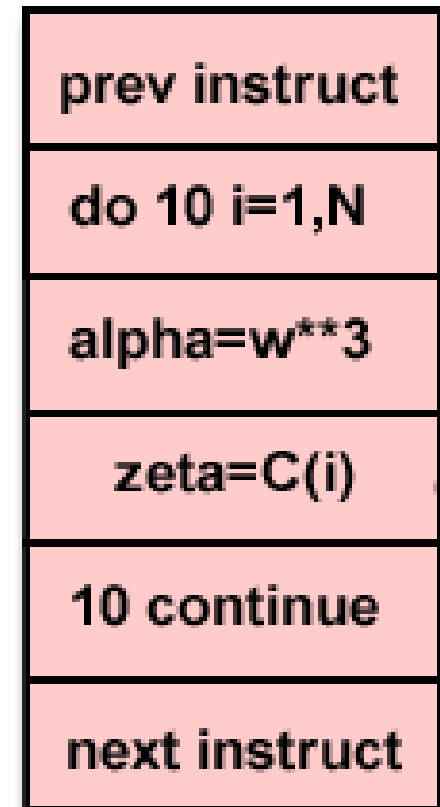
- Current processor
- Multiple different processors
- Multiple different instructions
- Execution determined by local processor
- Execution parallel and concurrent



P1



P2



Pn

time



Two programming mode for MIMD

- SPMD--

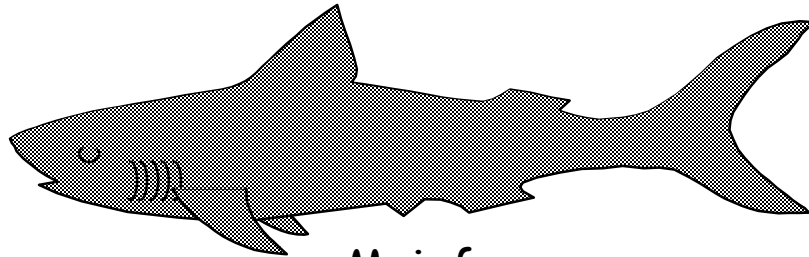
Multiple autonomous processors simultaneously executing the same program (but at independent points, rather than in the lockstep that SIMD imposes) on different data.

SPMD is the most common style of parallel programming.

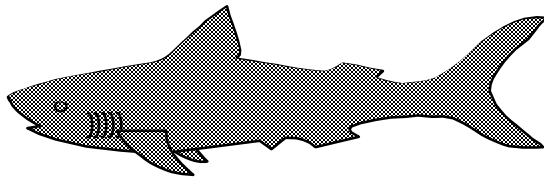
- MPMD--

Multiple autonomous processors simultaneously operating at least 2 independent programs. Typically such systems pick one node to be the "host" (host/node) or "manager" ("Manager/Worker"), which runs one program that farms out data to all the other nodes which all run a second program. Those other nodes then return their results directly to the manager.

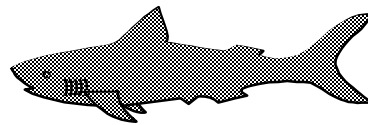
Classification-market



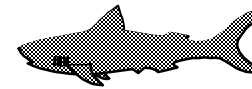
Mainframe
1960s



Supercomputer
1970s



Mini-
Supercomputer
1970s



Mini-
Computer
1970s



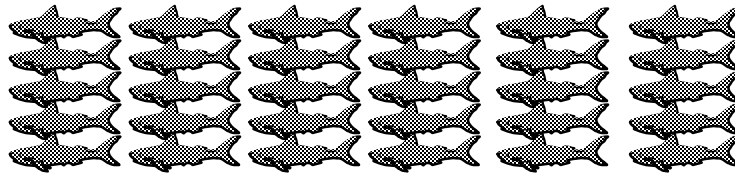
Work-
Station
1980s-2000s



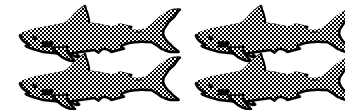
Embedded
system
2000s



PC



Massively Parallel
Processors
1980s



Server
2000s

3 computing markets

Feature	Desktop	Server	Embedded
Price of system	\$500-\$5000	\$5000 -\$5,000,000	\$10 -\$100,000
Price of microprocess or module	\$50-\$500 per proc.	\$200 -\$10,000 per proc.	\$0.01 -\$100 per proc.
Critical system design issues	Price-perf. Graphics perf.	Throughput, av ailability, scalability	Price, Power consumption, application- specific perf.

Desktop Computing

- The first, and still the largest market in dollar terms, is desktop computing.
- Requirement:
 - Optimized price-performance
- New challenges:
 - Web-centric, interactive application
 - How to evaluate performance ?

Servers

- The role of servers to provide larger scale and more reliable file and computing services grew.
 - For servers, different characteristics are important. First, **dependability** is critical.
 - A second key feature of server systems is an emphasis on **scalability**.
 - Lastly, servers are designed for **efficient throughput**.

Embedded systems

Retail POS

Thin Clients

Set-Top-Box



Gateway/Media Store



Network Devices



Kiosk/ATM

Intel® Architecture
Intel® XScale™

Game Platforms



Office Automation



Industrial Automation



Embedded Computers

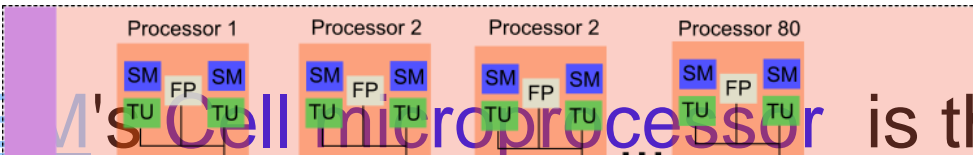
- The fastest growing portion of the computer market.
- 8-bit 16-bit 32-bit 64-bit
- Real time performance (soft & hard)
- Strict resource constraints
 - limited memory size, lower power consumption,...
- The use of processor cores together with application-specific circuitry.
 - DSP, mobile computing, mobile phone, Digital TV

Other classification

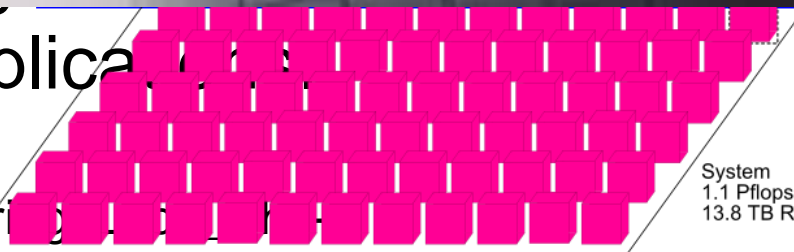
- Quantum computer vs Chemical computer,
- Scalar processor vs Vector processor
- Non-Uniform Memory Access (NUMA)/ UMA
- **Register machine vs Stack machine vs Accumulator machine**
- Harvard architecture vs Von Neumann architecture
/Non Von Neumann architecture
- **RISC vs. CISC**
- Cellular architecture

Cyclops64 (known as Blue Gene/C)

- IBM's Cell microprocessor is the first one to



many applications



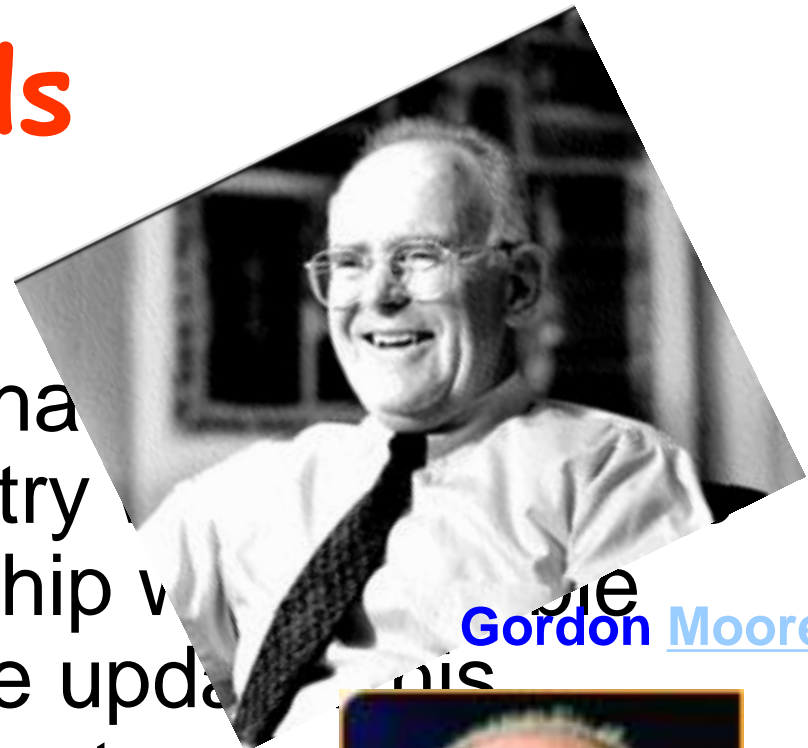
System
1.1 Pflops
13.8 TB RAM

Topics in Chapter

- 1.1 Why take this course ?
- 1.2 Classes of computers in current computer market
- 1.3 Defining computer architecture and What's the task of computer design?
- 1.4 Trends in Technology**
- 1.5 Trends in power in Integrated circuits
- 1.6 Trends in Cost
- 1.7 Dependability
- 1.8 Measuring, Reporting and summarizing Perf.
- 1.9 Quantitative Principles of computer Design
- 1.10 Putting it altogether

Technology Trends

- Moore Law
- In 1965 he predicted that the number of components the industry could place on a computer chip would double every year. In 1975, he updated his prediction to once every two years. This prediction became the guiding principle for the semiconductor industry to deliver more-powerful chips while decreasing the cost of electronics.



[Gordon Moore](#)



Technology Trends

■ Integrated circuit logic technology

- Transistor Density: incr. 35% per year, (4x every 4 years)
- Die size: 10%-20% per year
- Transistor count per chip: 40-55% per year

■ Semiconductor DRAM

- Capacity: 40% per year (2x every 2 years)
- Memory speed: about 10% per year

■ Magnetic Disk tech.

- Density: 30% p.y. Before 1990; 60% p.y. 1990-1996
- 100 p.y. 1996-2004 ; 30% p.y. after 2004
- capacity: about 60% per year

■ Network

- bandwidth: 10Mb $\xrightarrow{10 \text{ years}}$ 100Mb $\xrightarrow{5 \text{ years}}$ 1Gb

Designers often design for the next technology.

Gordon Moore on Moore's law

- **Moore's brief Bio**

<http://www.intel.com/pressroom/kits/bios/moore.htm>

- **Gordon Moore on Moore's law**

<http://www.sichinamag.com/Article/html/2007-09/2007919032802.htm>

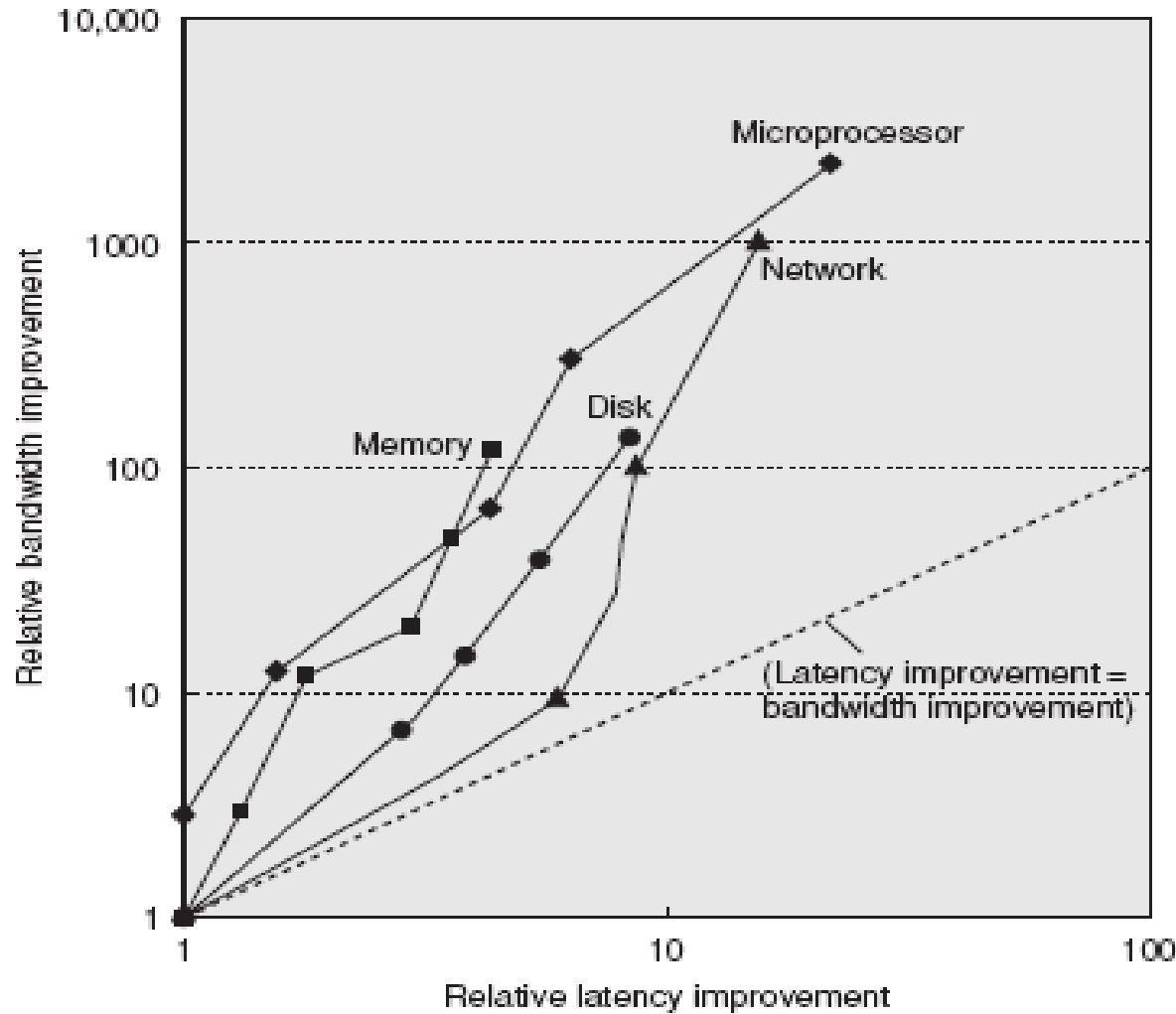
- **Video** on conversation with Moore

<http://you.video.sina.com.cn/b/7076856-1282136212.html>

Import Notes

- A rule of thumb
 - Cost decrease rate \sim density increase rate
- Technology thresholds
 - Technology improves continuously, an impact of this improvements can be in discrete leaps.

Perf. Trends: Bandwidth over latency



rk done

the

– Bandwidth grow rate \sim improvement in latency²

Performance milestones in microprocessor

Microprocessor	16-bit address/bus, microcoded	32-bit address.bus, microcoded	5-stage pipeline, on-chip I & D caches, FPU	2-way superscalar, 64-bit bus	Out-of-order 3-way superscalar	Out-of-order superpipelined, on-chip 1.2 cache
Product	Intel 80286	Intel 80386	Intel 80486	Intel Pentium	Intel Pentium Pro	Intel Pentium 4
Year	1982	1985	1989	1993	1997	2001
Die size (mm ²)	47	43	81	90	308	217
Transistors	134,000	275,000	1,200,000	3,100,000	5,500,000	42,000,000
Pins	68	132	168	273	387	423
Latency (clocks)	6	5	5	5	10	22
Bus width (bits)	16	32	32	64	64	64
Clock rate (MHz)	12.5	16	25	66	200	1500
Bandwidth (MIPS)	2	6	25	132	600	4500
Latency (ns)	320	313	200	76	50	15

Challenges for IC Technology

- IC characteristic: **feature size**(特征尺寸)
 - 10 microns in 1971 → 0.18microns in 2001
 - → 0.09 microns in 2006 → 65nm is underway
 - **Rule of thumb**: transistor perf. Improves linearly with decreasing feature size.
- IC density improvement is both opportunity and Challenge:
 - **signal delay** for a wire increase in proportion to the product of its resistance and capacitance.
- **Wire delay---major design limitation**

Bad prediction with big guy

- “There is no reason anyone would want a computer in their home.”

--Ken Olson, president,
Chairman and founder of Digital Equipment Corp.,
1977

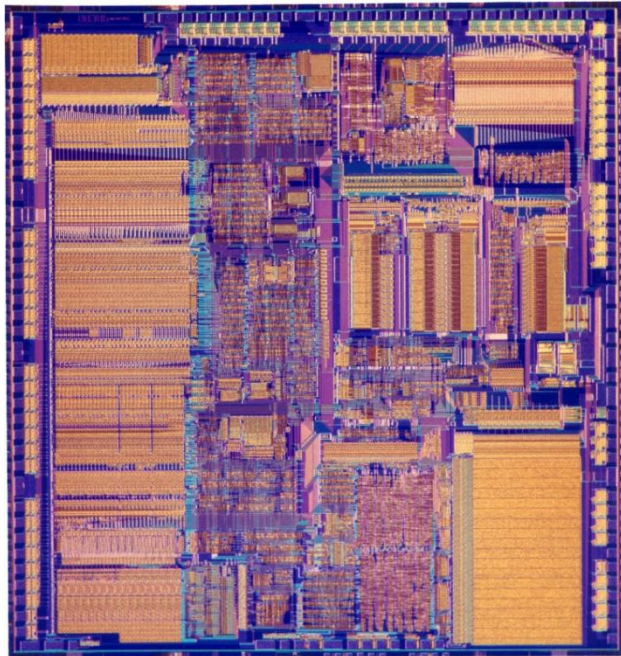
Topics in Chapter

- 1.1 Why take this course ?
- 1.2 Classes of computers in current computer market
- 1.3 Defining computer architecture and What's the task of computer design?
- 1.4 Trends in Technology
- 1.5 Trends in power in Integrated circuits**
- 1.6 Trends in Cost
- 1.7 Dependability
- 1.8 Measuring, Reporting and summarizing Perf.
- 1.9 Quantitative Principles of computer Design
- 1.10 Putting it altogether

Trends in Power

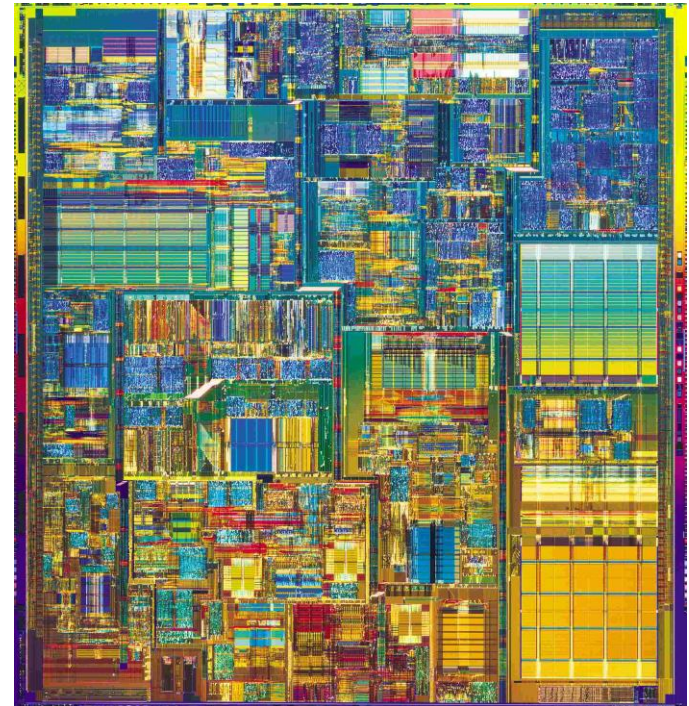
- **Power** also provide challenges as device scaled
 - first microprocessor: 1/10watt -->
2GHz P4: 135watt
- **Challenges:**
 - distributing the power
 - removing the heat
 - preventing hot spot

386 Processor

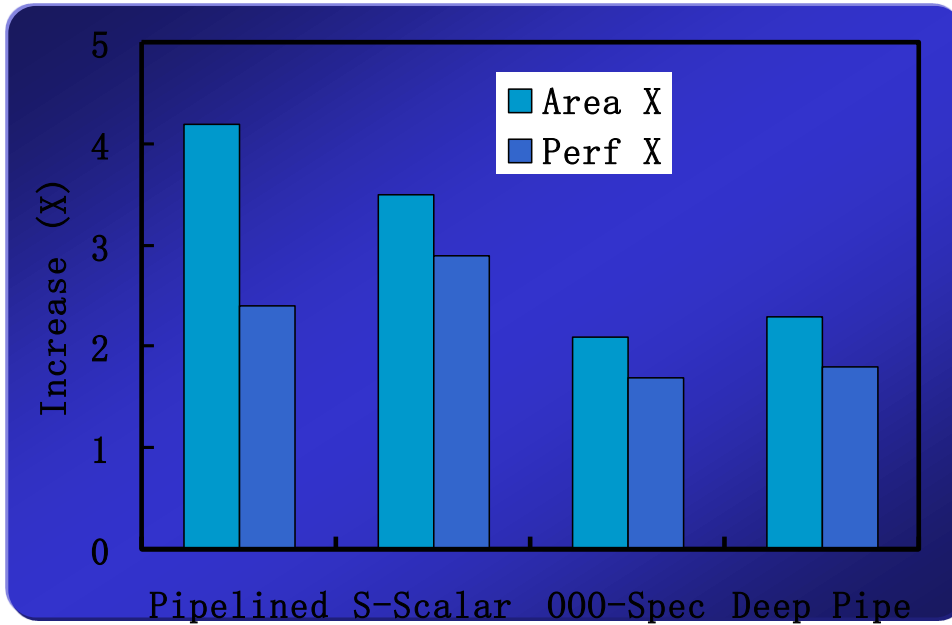


May 1986
17 Years
@16 MHz core
200x
275,000 1.5 μ transistors
200x/11x
~1.2 SPECint2000
1000x

Pentium® 4 Processor

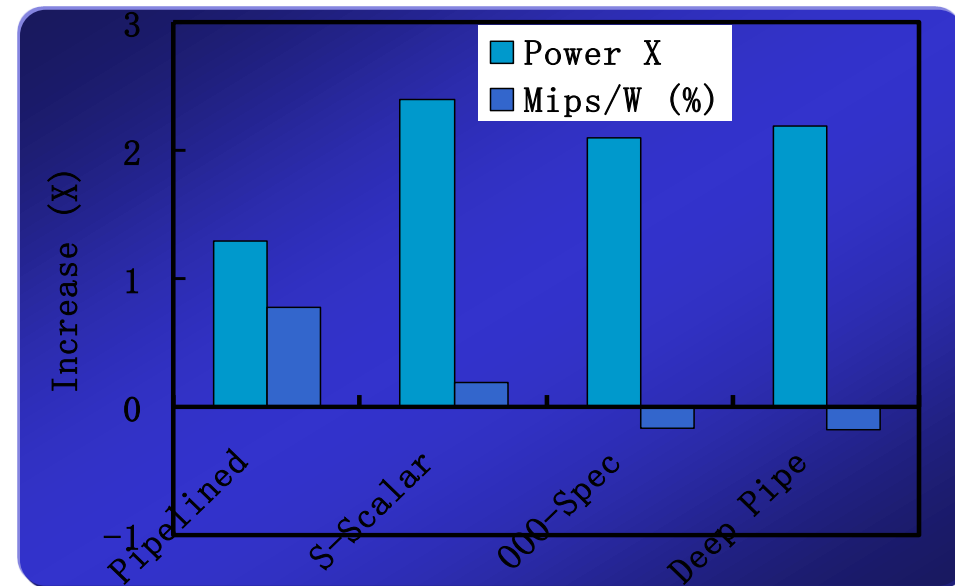


August 27, 2003
@3.2 GHz core
55 Million 0.13 μ transistors
1249 SPECint2000



Performance scales
with $\text{area}^{*.5}$

Power efficiency
has dropped



Two concepts

- **Dynamic power:** power consumption in switching transistors.
 - $\text{Power}_{\text{dynamic}} = \frac{1}{2} * \text{Capacitive load} * \text{Voltage}^2 * \text{Frequency switched}$
 - $\text{Energy}_{\text{dynamic}} = \text{Capacitive load} * \text{Voltage}^2$
- **Static power:** power consumption when a transistor is off due to power leakage
 - $\text{Power}_{\text{static}} = \text{current static} * \text{Voltage}$

Rule of Thumb

- **10% reduction of voltage yields**
 - 10% reduction in frequency
 - 30% reduction in power
 - Less than 10% reduction in performance

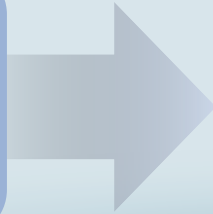
Rule of Thumb

Voltage	Frequency	Power	Performance
1%	1%	3%	0.66%

Dual core with voltage scaling

RULE OF THUMB

**A 15%
Reduction
In Voltage
Yields**

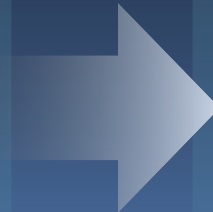


Frequency Reduction	Power Reduction	Performance Reduction
15%	45%	10%

SINGLE CORE



Area = 1
Voltage = 1
Freq = 1
Power = 1
Perf = 1

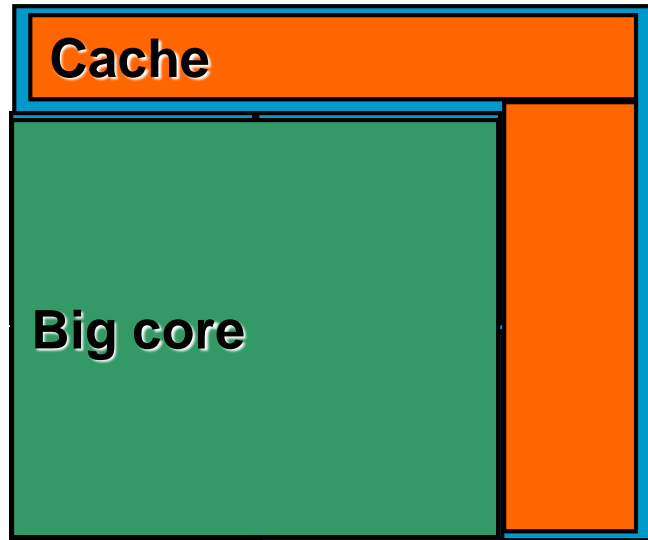


DUAL CORE

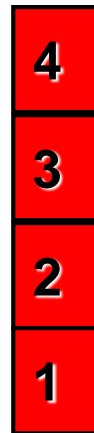


Area = 2
Voltage = 0.85
Freq = 0.85
Power = 1
Perf = ~1.8

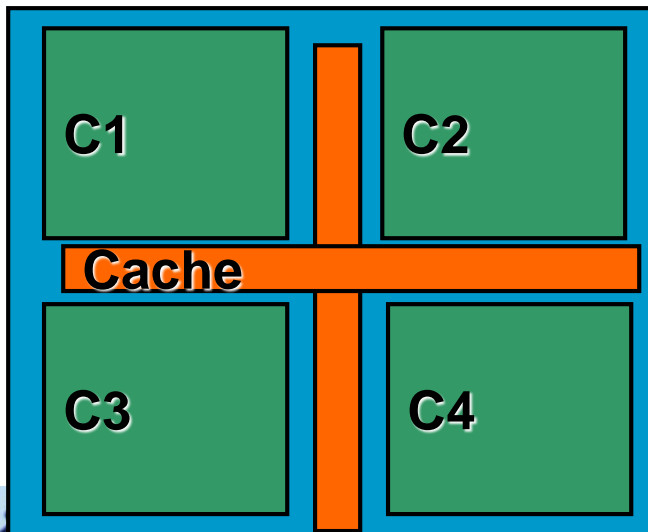
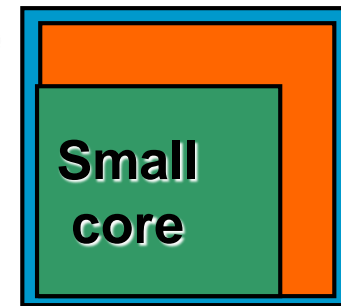
Multiple cores deliver more performance per watt



Power



Performance



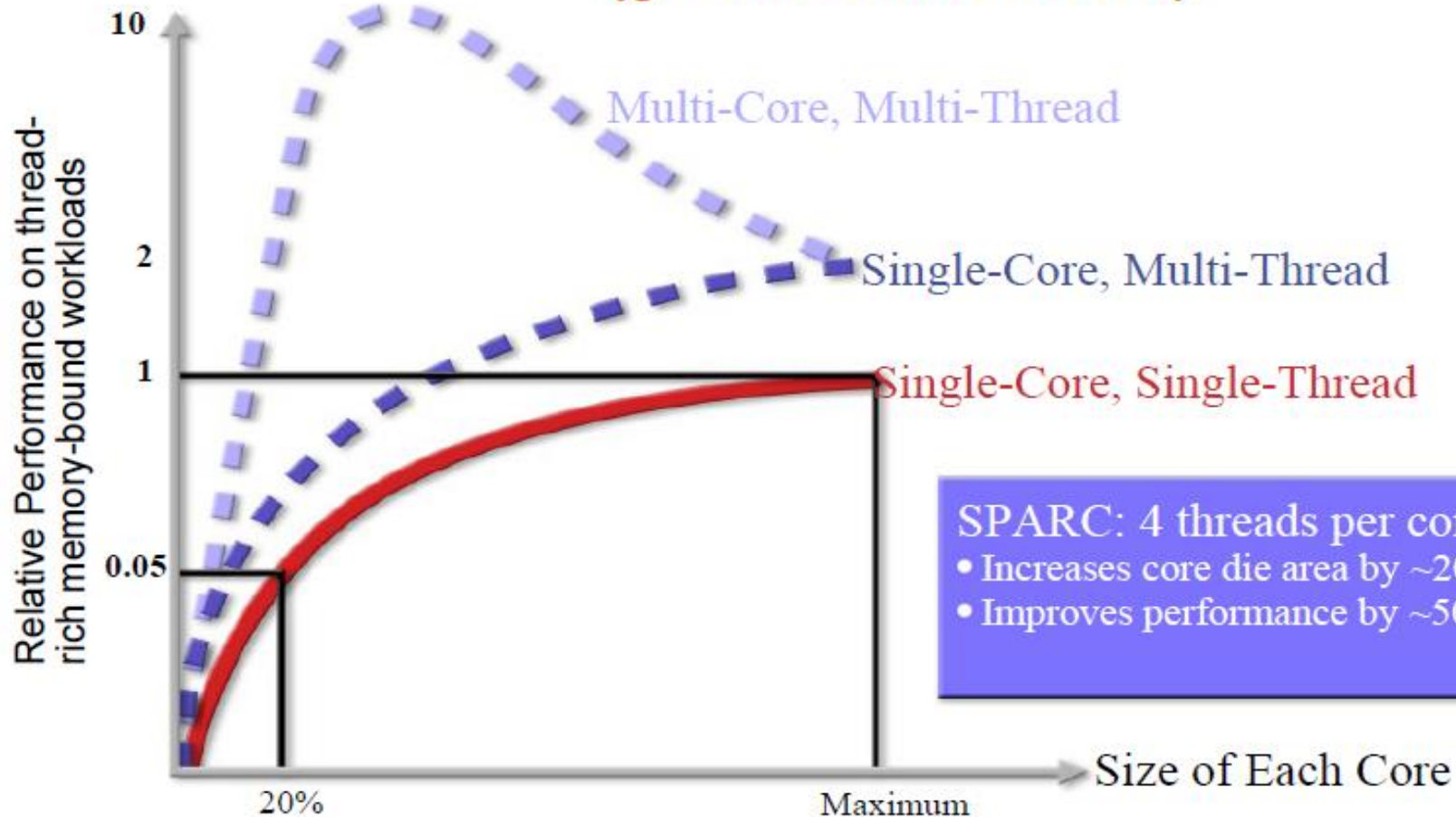
Many core is more power efficient

Power ~ area

Single thread performance ~ area^{0.5}

Why Multicore?

Goal: “100% Resource Utilization”
(given a fixed die size)



Topics in Chapter

- 1.1 Why take this course ?
- 1.2 Classes of computers in current computer market
- 1.3 Defining computer architecture and What's the task of computer design?
- 1.4 Trends in Technology
- 1.5 Trends in power in Integrated circuits
- 1.6 Trends in Cost**
- 1.7 Dependability
- 1.8 Measuring, Reporting and summerizing Perf.
- 1.9 Quantitative Principles of computer Design
- 1.10 Putting it altogether

Major Theme: Lower Cost

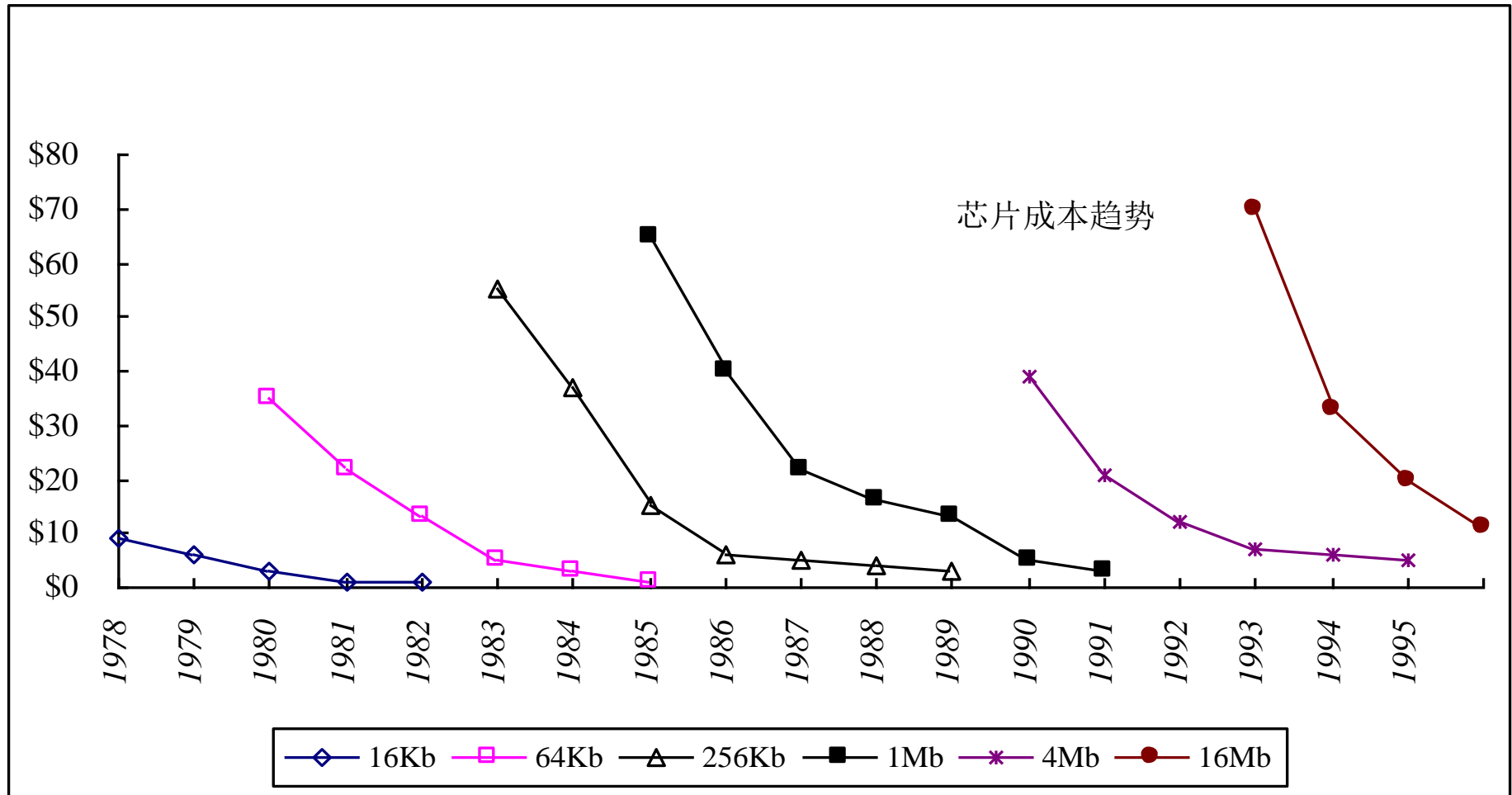
■ Cost Trend

- Understanding cost trends of component is important for designers, since we design for tomorrow !

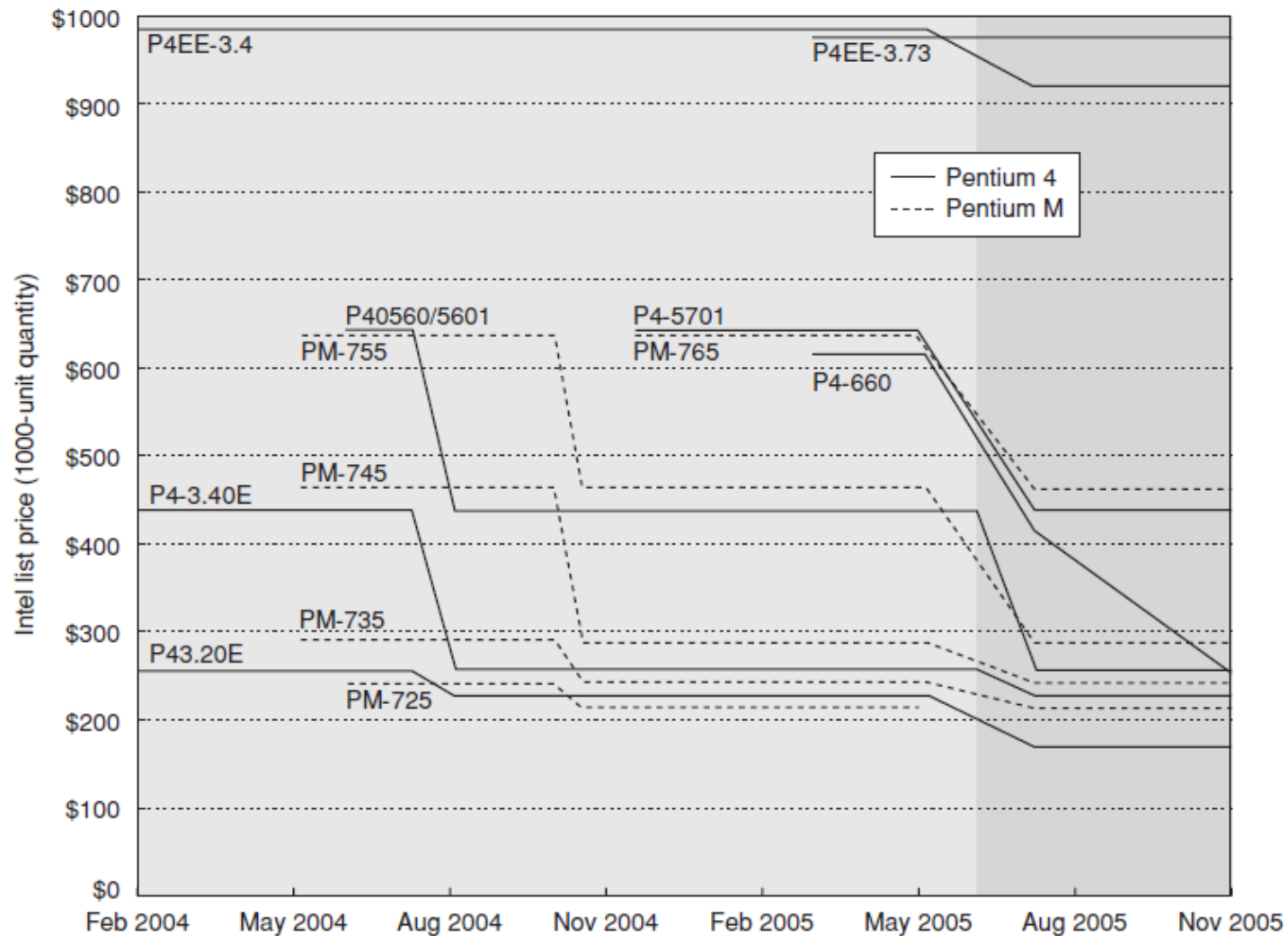
■ The impact factors for cost:

- **Time**-----Component prices drop over time without major improvements in manufacturing technology
- **Volume** -----Volume decreases cost due to increases in manufacturing efficiency.
- **Commodification**-----The competition among the suppliers of the components will decrease overall product cost.

Understanding Cost Trend by Learning Curve



Price of Pentium4 & PentiumM



Rules of Thumb

- Time: learning curve ----yield
 - Twice the yield will have half the cost.(for chip, board, or a system)
- Volume:
 - Cost decrease about 10% for each doubling of volume.
- Commodities:
 - Vendor competition
 - Supplier competition
 - Volume increase, however limited profits.

Cost of an Integrated Circuit

$$\text{Cost of integrated circuit} = \frac{\text{Cost of die} + \text{Cost of testing die} + \text{Cost of packaging and final test}}{\text{Final test yield}}$$

$$\text{Cost of die} = \frac{\text{Cost of wafer}}{\text{Dies per wafer} \times \text{Die yield}}$$

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

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

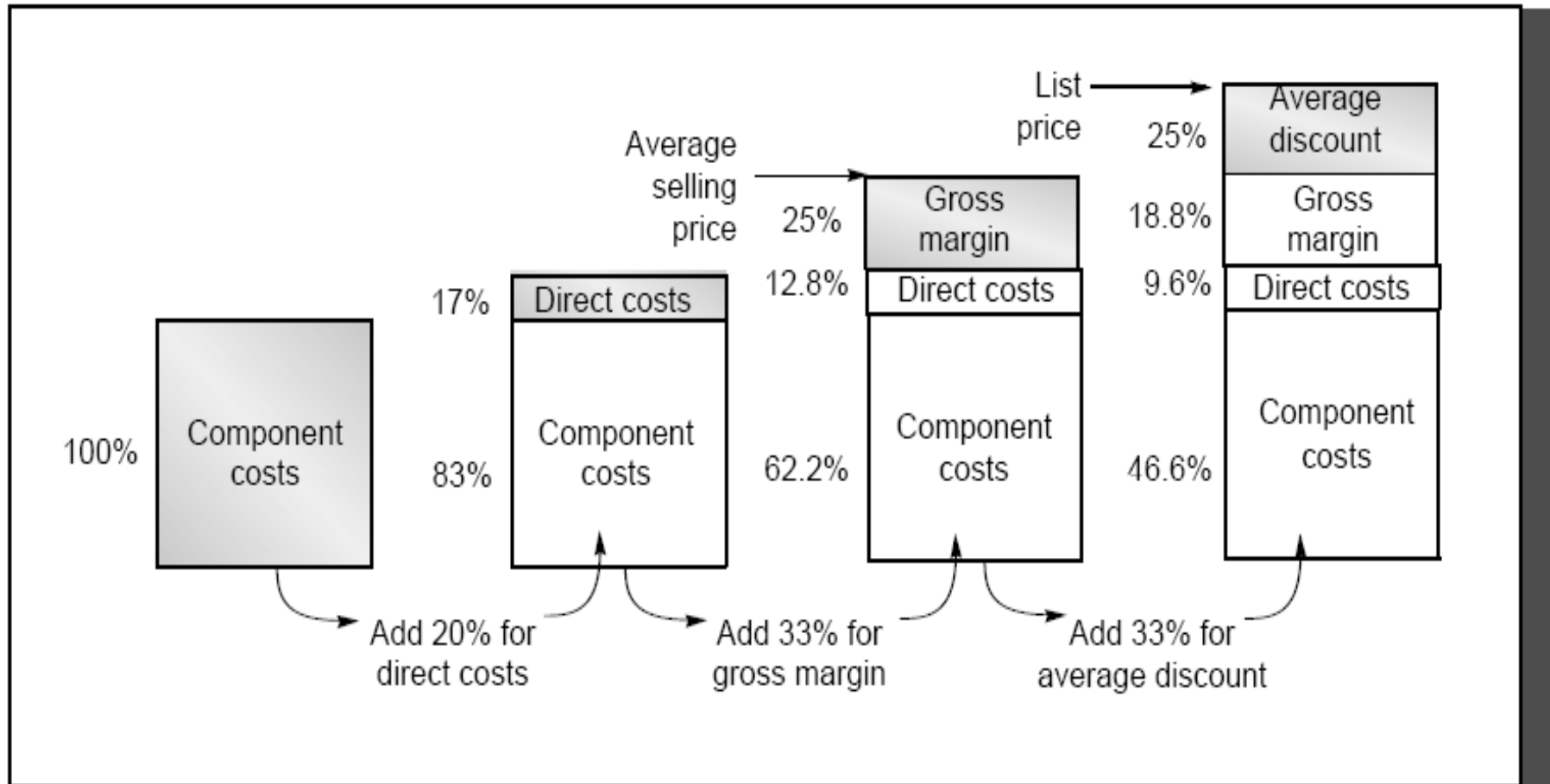
Distribution of Cost in a System

System	Subsystem	Fraction of total
Cabinet	Sheet metal, plastic	2%
	Power supply, fans	2%
	Cables, nuts, bolts	1%
	Shipping box, manuals	1%
	Subtotal	6%
Processor board	Processor	22%
	DRAM (128 MB)	5%
	Video card	5%
	Motherboard with basic I/O support, networking	5%
	Subtotal	37%
I/O devices	Keyboard and mouse	3%
	Monitor	19%
	Hard disk (20 GB)	9%
	DVD drive	6%
	Subtotal	37%
Software	OS + Basic Office Suite	20%

Cost vs. Price

- Component costs
 - Raw material cost.
- Direct cost:
 - Costs incurred to make a single item. Adds 20% to 40% to component cost.
- Gross margin (Indirect cost):
 - Overhead not associated with a single item, i.e. R&D, marketing, manufacturing equipment, taxes, etc.
 - Only 4%-12% of income are spent on R&D
- Average Selling Price (ASP):
 - Component cost + direct cost + indirect cost.
- List price :
 - Not ASP. Stores add to the ASP to get their cut. Want 50% to 75% of list price.

The components of price for a \$1000 PC



Cost vs. Price

- This gives you insight on how a design decision will affect selling price,
 - i.e. changing cost by \$1,000 increases selling price by \$3,000 to \$4,000.
- Also, consider volume and price relationship:
 - In general, the fewer computers that are sold, the higher the price.
 - Also, a decrease in volume causes cost to increase, further increasing price.
- Therefore, **small changes in cost can have an unexpected large increase in price.**

Topics in Chapter

- 1.1 Why take this course ?
- 1.2 Classes of computers in current computer market
- 1.3 Defining computer architecture and What's the task of computer design?
- 1.4 Trends in Technology
- 1.5 Trends in power in Integrated circuits
- 1.6 Trends in Cost
- 1.7 Dependability**
- 1.8 Measuring, Reporting and summarizing Perf.
- 1.9 Quantitative Principles of computer Design
- 1.10 Putting it altogether

Why Business Continuity ?

Lost Productivity

- Number of employees impacted (x hours out * hourly rate)

Know the downtime costs
(per hour, day, two days...)

Lost Revenue

- Direct loss
- Compensatory payments
- Lost future revenue
- Billing losses
- Investment losses

Damaged Reputation

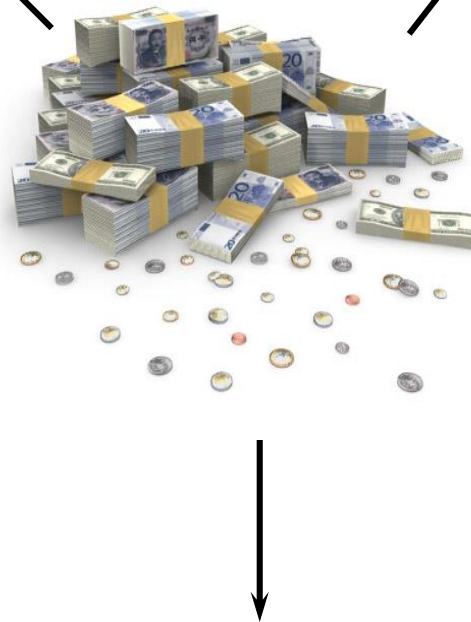
- Customers
- Suppliers
- Financial markets
- Banks
- Business partners

Financial Performance

- Revenue recognition
- Cash flow
- Lost discounts (A/P)
- Payment guarantees
- Credit rating
- Stock price

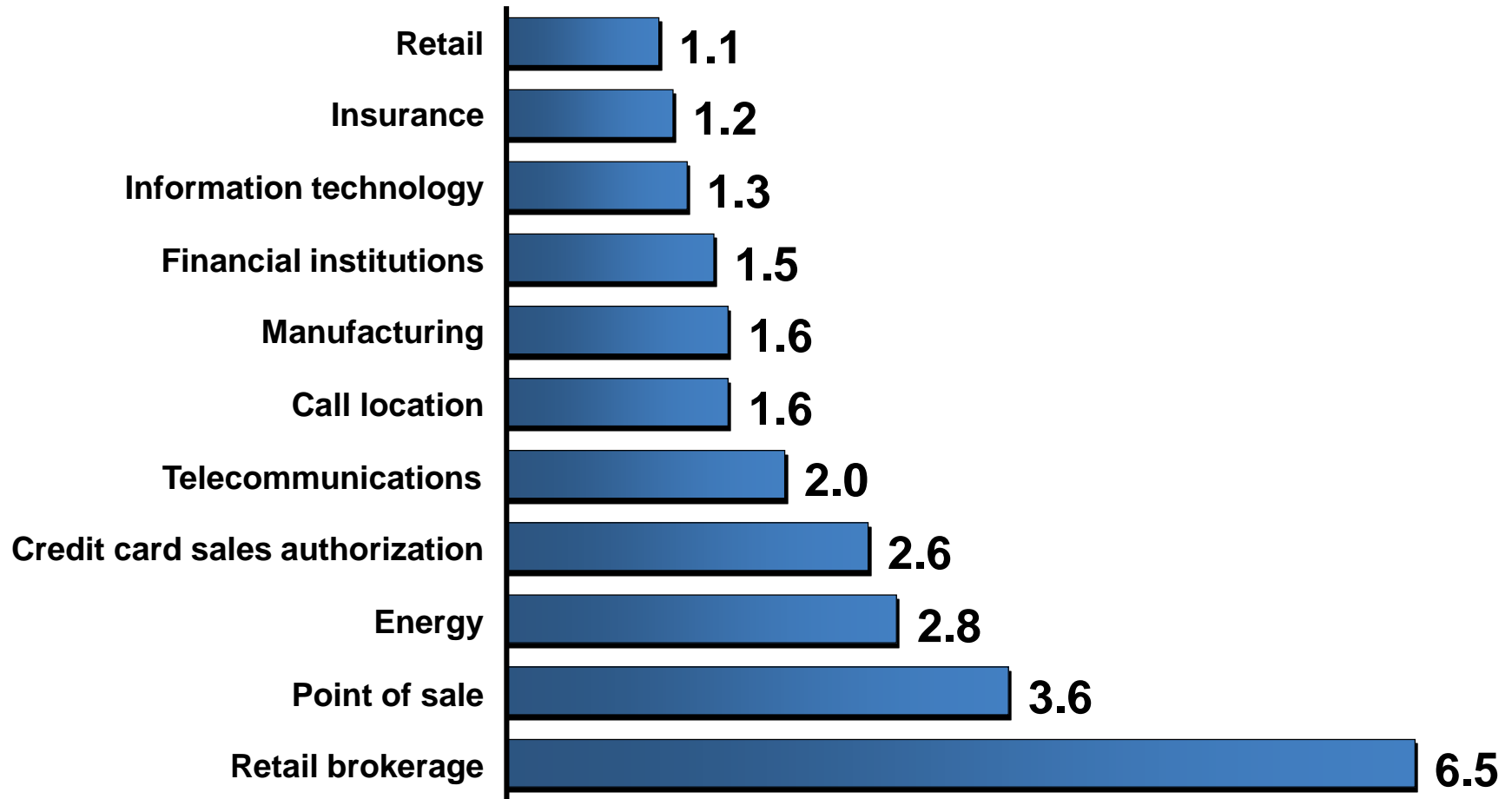
Other Expenses

Temporary employees, equipment rental, overtime costs, extra shipping costs, travel expenses...



Why high Dependability ?

Millions of US Dollars per Hour in Lost Revenue



Source Meta Group, 2005

Information Availability(5↑9)

% Uptime	% Downtime	Downtime per Year	Downtime per Week
98%	2%	7.3 days	3hrs 22 min
99%	1%	3.65 days	1 hr 41 min
99.8%	0.2%	17 hrs 31 min	20 min 10 sec
99.9%	0.1%	8 hrs 45 min	10 min 5 sec
99.99%	0.01%	52.5 min	1 min
99.999%	0.001%	5.25 min	6 sec
99.9999%	0.0001%	31.5 sec	0.6 sec

Evolvment of Dependability-1

- Reliability (可靠性) 1960s
 - ----field of fault tolerance and system reliability
- Definition
 - In general, **reliability** (systemic def.) is the ability of a person or system to perform and maintain its functions in routine circumstances, as well as hostile or unexpected circumstances.
 - The [IEEE](#) defines it as ". . . the ability of a system or component to perform its required functions under stated conditions for a specified period of time."

The Institute of Electrical and Electronics Engineers

Evolverment of Dependability-2

■ Dependability (可信性)

– 1985, Jean-Claude Laprie

- J.C. Laprie. "Dependable Computing and Fault Tolerance: Concepts and terminology," in Proc. 15th IEEE Int. Symp. on Fault-Tolerant Computing, 1985
- International Conference on dependable Systems and networks
- International Symposium on Reliable Distributed Systems
- International Symposium on Software Reliability Engineering.

Definition 1 of Denpendability

- Computer system dependability is The quality of the delivered service such that reliance can justifiably be placed on this service.
 - J.C. Laprie. "Dependable Computing and Fault Tolerance: Concepts and terminology," in Proc. 15th IEEE Int. Symp. on Fault-Tolerant Computing, 1985
 - reliability, maintainability, availability, safety, are quantitative measures corresponding to distinct perceptions of the same attribute of a system: it's dependability
 - general concept

Definition 2 of Dependability

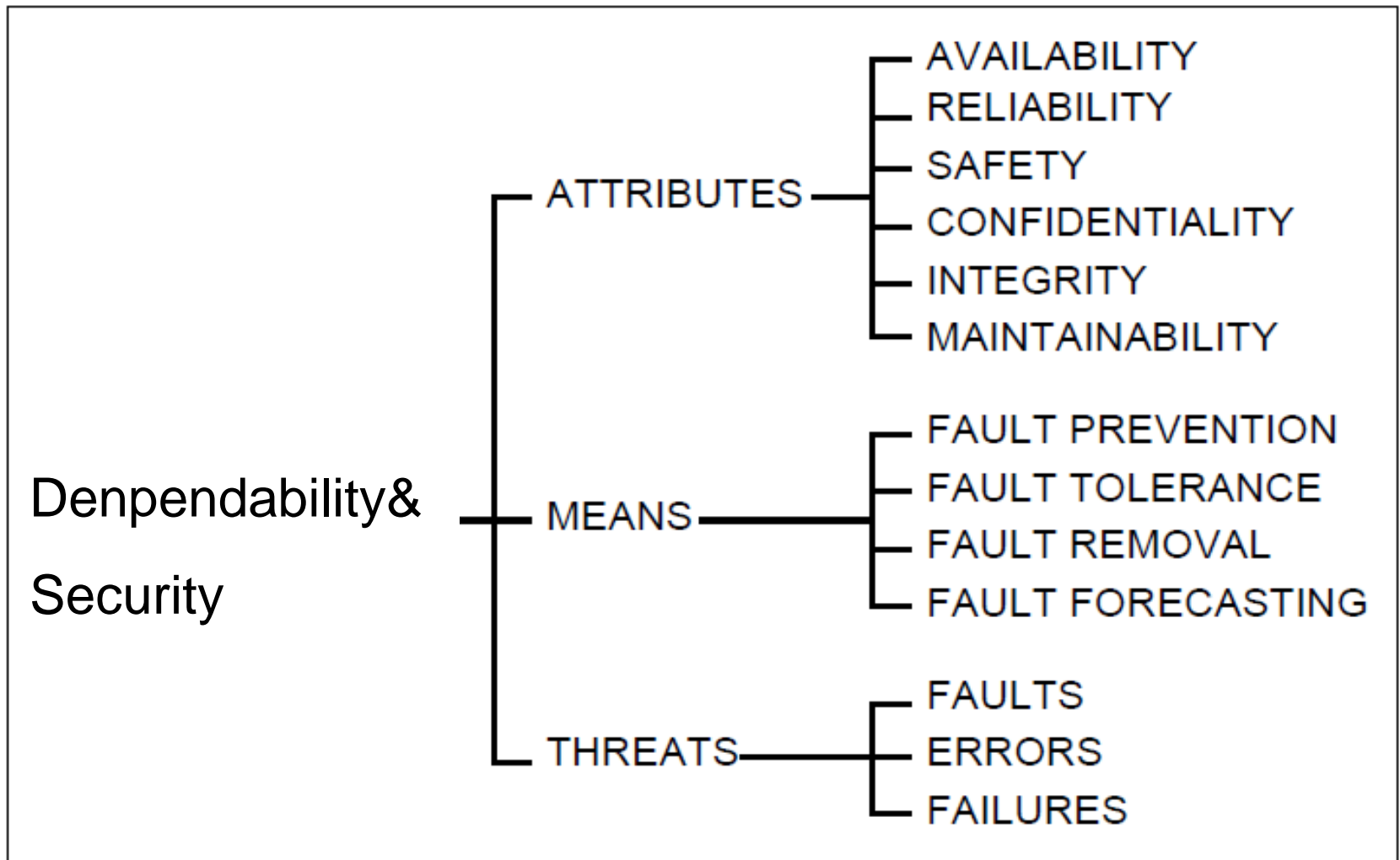


Figure 1 - The dependability tree

Related concepts

- Availability(可用性) - readiness for correct service
- Reliability(可靠性) - continuity of correct service
- Safety(平安性?) - absence of catastrophic consequences on the user(s) and the environment
- Integrity(完整性) - absence of improper system alteration
- Maintainability(可维护性) - ability to undergo modifications and repairs
- Confidentiality(机密性), *the absence of unauthorized disclosure of information*
- Security(安全保障) is a composite of Confidentiality, Integrity, and Availability.

Dependability(可信性)

- Service level Agreements(SLA) / Service level objectives (SLO)
 - Service accomplishment
 - Service interruption
- Failures:
 - $S_{\text{accomplishment}} \rightarrow S_{\text{interruption}}$
- Restorations:
 - $S_{\text{interruption}} \rightarrow S_{\text{accomplishment}}$

Measurements of Dependability

- Module **reliability**: continuous service accomplishment (of the time to failure)
 - MTTF: Mean Time To Failure
 - MTTR: Mean Time To Repair (service interruption)
 - FIT: Failure In Time = $1/\text{MTTF}$
 - MTBF: Mean Time Between Failure = $\text{MTTF} + \text{MTTR}$
- Module **availability**
 - $$\frac{\text{MTTF}}{\text{MTTF} + \text{MTTR}} = \frac{\text{MTTF}}{\text{MTBF}}$$

Example: disk system

■ Assumption:

- 10 disks, 1000000 hour MTTF
- 1 SCSI controller, 500000 hour MTTF
- 1 power supply, 1 fan, both 200000 hour MTTF
- 1 SCSI cable, 1000000 hour MTTF

■ Question: MTTR of the whole system

■ Answer:

- Failure rate_{system} = $10 \cdot 1/1000,000 + 1/500,000 + 1/200,000 + 1/100,000$
= $23000/1000,000,000$
- MTTF_{system} = $1/\text{Failure}_{\text{system}}$ = 43500 hours ~ 5 years

Way to cope with failure

■ Redundancy:

- **Time redundancy**: repeat the operation again to see if it is still in erroneous.
- **Resource redundancy**: have other components to take over from the one that failed.

Dependability

- Compare with
 - Trusted computing
 - Trustworthy computing
 - Credibility

Topics in Chapter

- 1.1 Why take this course ?
- 1.2 Classes of computers in current computer market
- 1.3 Defining computer architecture and What's the task of computer design?
- 1.4 Trends in Technology
- 1.5 Trends in power in Integrated circuits
- 1.6 Trends in Cost
- 1.7 Dependability
- 1.8 Measuring, Reporting and summerizing Perf.
- 1.9 Quantitative Principles of computer Design
- 1.10 Putting it altogether

Performance



Measuring and Reporting Performance

■ Comparing Machines

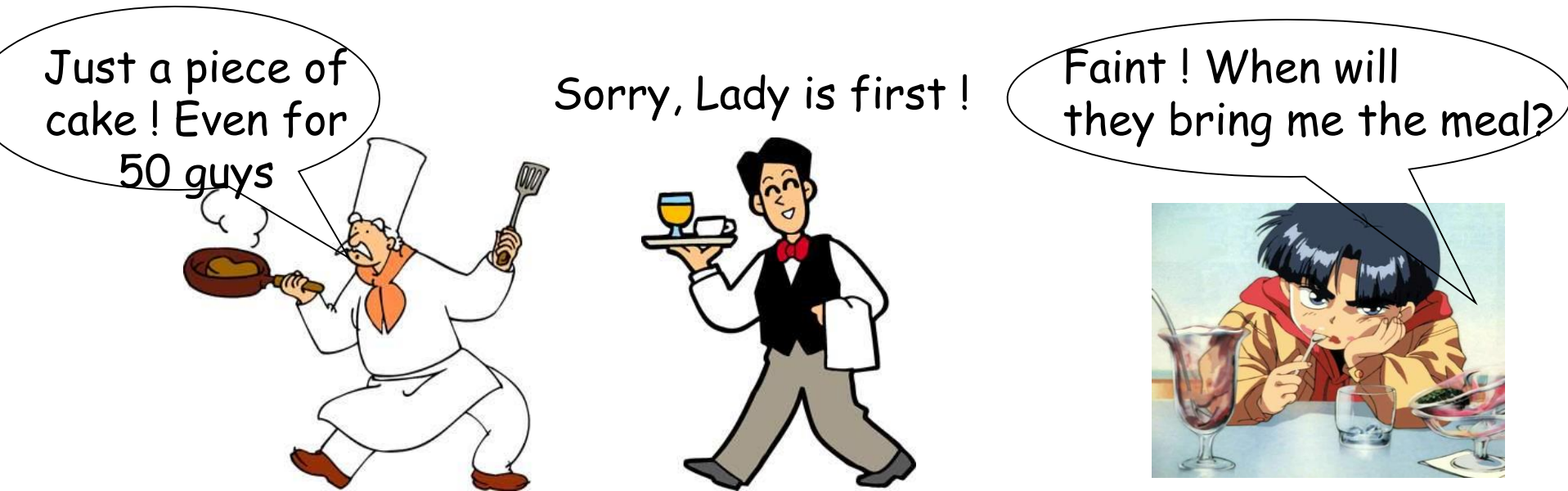
- Execution time (latency)
- Throughput
- MIPS - millions of instructions per second

■ Comparing Machines Using Sets of Programs

- Choosing which program to evaluate performance
 - Benchmark Suites
- Different Means: Arithmetic, Harmonic, and Geometric Means

Defining performance

- Performance means different things to different people, therefore its assessment is subtle



- different perception
 - Criteria of performance evaluation differs among users and designers

Pert. Metrics --response time

■ Wall-clock time

- Start the program and watch the clock -
- when the program ends, that's the total **wall-clock time**
- Also called **response time** or **elapsed time** or
- Measures **user perception** of the system speed

■ Problems with wall-clock time

- What if more than one program is running on the same machine ?
- What if the program asks for user input ?



Performance Metrics --CPU time

- Measures the time the CPU is computing, (not waiting for I/O)
 - Measures **designer perception** of the CPU speed
- CPU time is further divided into:
 - **User time** - time spent in user mode
 - **System time** - time spent in the operating system (OS)
- Unix time command reports CPU time as:
 - **90.7u 12.9s 2:39 65%**
 - 90.7 user CPU seconds (in the user's program)
 - 12.9 system CPU seconds (in the system calls e.g. printf)
 - 2 minutes, 39 seconds wall-clock time
 - 65% of the wall clock time was spent running on the CPU

Performance Metrics --throughput

- Amount of work done in a given time
 - Measure administrator perception of the system perf.
- We often use throughput to measure
 - Number of lines of code per day
 - Number bits per second transmitted over a wire
 - Number of web pages served
- In contrast to latency
 - amount of time to produce 1 line of code
 - amount of time to send 1 bit over a wire
 - Amount of time spent waiting to receive web page
- Often, processor performance is only quoted in terms of relative latency
 - Program A ran 10 times faster than program B
- But, for many apps, throughput **much** more important than latency
 - Financial markets, government statistics (census)

Response time vs. Throughput

- If you improve response time, you usually improve throughput
 - Replacing the processor of a computer with a faster version
- you can also improve throughput without improving response time
 - Adding additional processors to a system that uses multiple processors
 - for separate tasks (e.g. handling of airline reservations system)

Another industry Metric: MIPS

- MIPS - Millions of Instructions per Second

$$\text{MIPS} = \frac{\frac{\text{\# of instructions}}{\text{benchmark}} \times \frac{\text{benchmark}}{\text{total run time}}}{1,000,000}$$

- When comparing two machines (A, B) with the same **instruction set**, MIPS is a fair comparison(sometimes...)
- But, MIPS can be a "**meaningless indicator of performance...**"

Ex: MIPS might be meaningless

- Machine A has a special instruction for performing square root calculations. It takes 100 cycles to execute.
- Machine B doesn't have the special instruction -
- must perform square root calculations in software using simple instructions (.e.g, Add, Mult, Shift) that each take 1 cycle to execute
- Machine A: $1/100 \text{ MIPS} = 0.01 \text{ MIPS}$
- Machine B: 1 MIPS

Another view: Power consumption and Efficiency

■ Critical factors for embedded systems:

- cost
- physical size
- memory
- power consumption

■ Fig. 1.27

- AMD ElanSC520
- AMD K6-2E
- IBM PowerPC 750CX
- NEC VR 5432
- NEC VR 4122

The NEC VR 4122 is the big winner for its **best performance/watt**, though it is the second lowest performing processor.

Summary of performance metrics

- Response (Execution) time
 - user perception
 - system performance
 - the only unimpeachable measure of performance
- CPU time
 - designer perception
 - CPU performance
- Throughput
 - administrator perception
- MIPS
 - merchant perception

Choose Programs to Evaluate Performance

- Ideal performance evaluation:
 - A random sample of users running their programs and OS commands.
- Many different types of benchmarks
 - Real applications --- Scientific and engineering
 - Modified (or scripted) applications --- focus on specific features
 - Kernels --- critical program fragments
 - Toy --- small programs, often measure very little
 - Synthetic -- created to represent some aspects of a program (e.g., mix of instruction types)
 - Database -- a world unto itself
 - What really matters is how YOUR application performs

Something about Synthetic

■ Synthetic benchmarks :

- Programs that try to "exercise" the system in the same way to match the average frequency of operations and operands of a large set of programs.
- Whetstone and Dhrystone.
- Similar to kernels but are **NOT** real programs !
- Compiler and hardware optimizations can artificially inflate performance of these benchmarks but not of real programs.
- These benchmarks don't reward optimizations!
- $\text{SQRT}(\text{EXP}(x)) = \sqrt{e^x} = e^{x/2} = \text{EXP}(X/2)$

—

Notes on performance benchmark

- Benchmarks can focus on specific aspects of a system
 - floating point & integer ALU, memory system, I/O, OS
- Universal benchmarks **can be misleading** since hardware and compiler vendors might optimize their design for **ONLY** these programs
- **The best types of benchmarks are real applications since they reflect the 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.

SPEC

■ SPEC - The System Performance Evaluation Cooperative

- founded in 1988 by a small number of workstation vendors who realized that the marketplace was in desperate need of realistic, standardize performance tests.
- Grown to become successful performance standardization bodies with more than 40 member companies.
- <http://www.spec.org>

■ SPEC's Philosophy

- The goal of SPEC is to ensure that the marketplace has a fair and useful set of metrics to differentiate candidate systems.
- The basic SPEC methodology is to provide the benchmarker with a standardized suite of source code based upon existing applications

SPEC benchmarks

Desktop Benchmarks

■ CPU-intensive benchmarks

- SPEC89
- SPEC92
- SPEC95
- SPEC2000
- SPEC CPU2006 (12 CINT2006, 17 CFP2006)

■ graphics-intensive benchmarks

- SPEC2000
 - SPECviewperf
 - is used for benchmarking systems supporting the OpenGL graphics library
 - SPECapc
 - consists of applications that make extensive use of graphics.

SPEC INT 95 Benchmark descriptions

<u>Benchmark</u>	<u>Ref Time (Sec)</u>	<u>Application Area</u>	<u>Specific Task</u>
099.go	4600	Game playing; artificial intelligence	Plays the game Go against itself.
124.m88ksim	1900	Simulation	Simulates the Motorola 88100 processor running Dhrystone and a memory test program.
126.gcc	1700	Programming & compilation	Compiles pre-processed source into optimized SPARC assembly code.
129.compress	1800	Compression	Compresses large text files (about 16MB) using adaptive Lempel-Ziv coding.
130.li	1900	Language interpreter	Lisp interpreter.
132.jpeg	2400	Imaging	Performs jpeg image compression with various parameters.
134.perl	1900	Shell interpreter	Performs text and numeric manipulations (anagrams/prime number factoring).
147.vortex	2700	Database	Builds and manipulates three interrelated databases.

SPEC FP 95 Benchmark Descriptions

<u>Ben</u> <u>chmark</u>	<u>Ref Time (Sec)</u>	<u>Application Area</u>	<u>Specific Task</u>
101.tomcatv	3700	Fluid Dynamics / Geometric Translation	Generation of a two-dimensional boundary-fitted coordinate system around general geometric domains.
102.swim	8600	Weather Prediction	Solves shallow water equations using finite difference approximations. (The only single precision benchmark in CFP95.)
103.su2cor	1400	Quantum Physics	Masses of elementary particles are computed in the Quark-Gluon theory.
104.hydro2d	2400	Astrophysics	Hydrodynamical Navier Stokes equations are used to compute galactic jets.
107.mgrid	2500	Electromagnetism	Calculation of a 3D potential field.
110.applu	2200	Fluid Dynamics/Math	Solves matrix system with pivoting.
125.turb3d	4100	Simulation	Simulates turbulence in a cubic area.
141.apsi	2100	Weather Predication	Calculates statistics on temperature and pollutants in a grid.
145.fpppp	9600	Chemistry	Performs multi-electron derivatives.
146.wave	3000	Electromagnetics	Solve's Maxwell's eqn on cartesian mesh.

New SPEC Int 2000 Benchmarks

164.gzip	C	Compression
175.vpr	C	FPGA Circuit Placement and Routing
176.gcc	C	C Programming Language Compiler
181.mcf	C	Combinatorial Optimization
186.crafty	C	Game Playing: Chess
197.parser	C	Word Processing
252.eon	C++	Computer Visualization
253.perlbnk	C	PERL Programming Language
254.gap	C	Group Theory, Interpreter
255.vortex	C	Object-oriented Database
256.bzip2	C	Compression
300.twolf	C	Place and Route Simulator

New SPEC FP 2000 Benchmarks

168.wupwise	Fortran 77	Physics / Quantum Chromodynamics
171.swim	Fortran 77	Shallow Water Modeling
172.mgrid	Fortran 77	Multi-grid Solver: 3D Potential Field
173.applu	Fortran 77	Parabolic / Elliptic Partial Differential Equations
177.mesa	C	3-D Graphics Library
178.galgel	Fortran 90	Computational Fluid Dynamics
179.art	C	Image Recognition / Neural Networks
183.quake	C	Seismic Wave Propagation Simulation
187.facerec	Fortran 90	Image Processing: Face Recognition
188.amp	C	Computational Chemistry
189.lucas	Fortran 90	Number Theory / Primality Testing
191.fma3d	Fortran 90	Finite-element Crash Simulation
200.sixtrack	Fortran 77	High Energy Nuclear Physics Accelerator Design
301.apsi	Fortran 77	Meteorology: Pollutant Distribution

SPEC benchmarks

Server Benchmarks

- SPECrate--processing rate of a multiprocessor
 - SPEC CPU2000—throughput-oriented benchmark
 - SPECrate—processing rate of a multiprocessor
 - SPECIFS--file server benchmark
 - SPECWeb--Web server benchmark
 - Transaction-processing (TP) benchmarks
 - TPC benchmark—Transaction Processing Council
 - TPC-A, 1985
 - TPC-C, 1992,
 - TPC-H → TPC-R → TPC-W

SPEC benchmarks

Embedded Benchmarks

- EDN Embedded Microprocessor Benchmark Consortium (or **EEMBC**, pronounced “embassy”).

Benchmark type	Number of kernels	Example benchmarks
Automotive/industrial	16	6 microbenchmarks (arithmetic operations, pointer chasing, memory performance, matrix arithmetic, table lookup, bit manipulation), 5 automobile control benchmarks, and 5 filter or FFT benchmarks
Consumer	5	5 multimedia benchmarks (JPEG compress/decompress, filtering, and RGB conversions)
Networking	3	Shortest-path calculation, IP routing, and packet flow operations
Office automation	4	Graphics and text benchmarks (Bézier curve calculation, dithering, image rotation, text processing)
Telecommunications	6	Filtering and DSP benchmarks (autocorrelation, FFT, decoder, encoder)



Running Benchmarks

- Key factor: **Reproducibility** by other experimenters.
- **Details, details, and more details !!!** List all assumptions and conditions of your experiments.
 - i.e. program input, version of the program, version of the compiler, optimization level, OS version, main memory size, disk types, etc.
- A system's **software configuration** can significantly affect the performance results for a benchmark.

Comparing Two Machines

Machine (secs)	CPI	Clock Period	Avg Instruction Time
Machine A	1.2	2 ns	
Machine B	2.5	1 ns	

- CPU Time = # of instructions executed * avg instruction time
- Assume 1,000,000, 000 instructions
- Machine A: $1,000,000,000 * 2.4\text{ns} = 2.4 \text{ seconds}$
- Machine B: $1,000,000,000 * 2.5\text{ns} = 2.5 \text{ seconds}$
- Which machine is faster? Machine A
- How much faster? $2.5 / 2.4 = 1.04 \text{ times faster}$

Comparing Performance

- Often, we want to compare the performance of different machines or different programs. Why?
 - To help engineers understand which is "better"
 - To give marketing a "silver bullet" for the press release
 - To help customers understand why they should buy <my machine>
- Performance and Execution time are *reciprocals*
Maximizing performance means minimizing response (execution) time

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

Common used phrases

- "Performance of P_1 is better than P_2 " is, for a given work load L , P_1 takes less time to execute L than P_2 does

performance(P_1) > Performance(P_2)

\Rightarrow Execution Time(P_1, L) < Execution Time(P_2, L)

- "Processor X is n times fast than Y " is

$$n = \frac{\text{Execution time}_Y}{\text{Execution time}_X}$$



Comparing Performance Across Multiple Programs

	<u>Computer A</u>	<u>Computer B</u>	<u>Computer C</u>
Program 1 (secs)	1	10	20
Program 2 (secs)	1000	100	20
Program 3 (secs)	1001	110	40

- A is 10 times faster than B for program 1
- B is 10 times faster than A for program 2
- A is 20 times faster than C for program 1
- C is 50 times faster than A for program 2
- B is 2 times faster than C for program 1
- C is 5 times faster than B for program 2

■ Each statement above is correct...,
...but we want to know **which machine is the best?**

Let's Try a Simpler Example

■ Two machines timed on two benchmarks

- How much faster is Machine A than Machine B?

	<u>Machine A</u>	<u>Machine B</u>
Program 1	2 seconds	4 seconds
Program 2	12 seconds	8 seconds

■ Attempt 1: ratio of run times, normalized to Machine A times

- program1: $4/2$ program2 : $8/12$
- Machine A ran 2 times faster on program 1, $2/3$ times faster on program 2
- On *average*, Machine A is $(2 + 2/3) / 2 = 4/3$ times faster than Machine B

■ It turns this “*averaging*” stuff can fool us

Example: Second answer

- Two machines timed on two benchmarks
 - How much faster is Machine A than Machine B?

	<u>Machine A</u>	<u>Machine B</u>
Program 1	2 seconds	4 seconds
Program 2	12 seconds	8 seconds

- Attempt 2: **ratio of run times, normalized to Machine B times**
 - program 1: $2/4$ program 2 : $12/8$
 - Machine A ran program 1 in $1/2$ the time and program 2 in $3/2$ the time
 - On **average**, $(1/2 + 3/2) / 2 = 1$
 - Put another way, Machine A is **1.0 times faster** than Machine B

Example: Third answer

- Two machines timed on two benchmarks
 - How much faster is Machine A than Machine B?

	<u>Machine A</u>	<u>Machine B</u>
Program 1	2 seconds	4 seconds
Program 2	12 seconds	8 seconds

- Attempt 3: **ratio of run times, aggregate (total sum) times,**
 - Machine A took **14** seconds for both programs
 - Machine B took **12** seconds for both programs
 - Therefore, Machine A takes $14/12$ of the time of Machine B
 - Put another way, Machine A is **$6/7$ faster** than Machine B

Which is Right?

■ Question:

- How can we get *three different answers*?

■ Solution

- Because, while they are all *reasonable* calculations...
- *...each answers a different question*

- We need to be more precise in understanding and posing these performance & metric questions

Arithmetic and Harmonic Mean

■ Total Execution Time: A Consistent Summary Measure

- **Arithmetic mean** is the average of the execution time that tracks total execution time.

$$\frac{1}{n} \sum_{i=1}^n Time_i$$

- If performance is expressed as a **rate**, then the average that tracks total execution time is the **harmonic mean**

$$\frac{n}{\sum_{i=1}^n \frac{1}{Rate_i}}$$

Problems with Arithmetic Mean

- Applications do not have the same **probability** of being run
- Longer programs weigh more heavily in the average
- For example, two machines timed on two benchmarks

	Machine A	Machine B
Program 1	2 seconds (20%)	4 seconds (20%)
Program 2	12 seconds (80%)	8 seconds (80%)

- If we do arithmetic mean, Program 2 “counts more” than Program 1
 - an improvement in Program 2 changes the average more than a proportional improvement in Program 1
- But perhaps Program 1 is 4 times more likely to run than Program 2

Weighted Execution Time

- Often, one runs some programs more often than others. Therefore, we should **weight** the more frequently used programs' execution time

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

- **Weighted** Harmonic Mean

$$\frac{1}{\sum_{i=1}^n \frac{\text{Weight}_i}{\text{Rate}_i}}$$

Using a Weighted Sum (or weighted average)

	<u>Machine A</u>	<u>Machine B</u>
Program 1	2 seconds (20%)	4 seconds (20%)
Program 2	12 seconds (80%)	8 seconds (80%)
Total	10 seconds	7.2 seconds

Allows us to determine relative performance

$$10/7.2 = 1.38$$

--> Machine B is **1.38** times faster than
Machine A

Another Solution

- Normalize run time of each program to a reference

	<u>Machine A (ref)</u>	<u>Machine B</u>
Program 1	2 seconds	4 seconds
Program 2	12 seconds	8 seconds
Total	10 seconds	7.2 seconds

	<u>Machine A (norm to B)</u>	<u>Machine B (norm to A)</u>
Program 1	0.5	2.0
Program 2	1.5	0.666
Average?	1.0	1.333

- So when we normalize A to B, and average, it looks like A & B are the same.
- But when we normalize B to A, it looks like B is 33% better!

Example

	Programs			Weightings		
	A	B	C	W(1)	W(2)	W(3)
Program P1 (secs)	1.00	10.00	20.00	0.50	0.909	0.999
Program P2 (secs)	1000.00	100.00	20.00	0.50	0.091	0.001
Arithmetic mean: W(1)	500.50	55.00	20.00			
Arithmetic mean: W(2)	91.91	18.19	20.00			
Arithmetic mean: W(3)	2.00	10.09	20.00			

$$W(B)_1 = \frac{1}{10 \times (1/10 + 1/100)} = 0.909$$

$$W(B)_2 = \frac{1}{100 \times (1/10 + 1/100)} = 0.091$$

Equal-time weighting

$$W_i = \frac{1}{\text{Time}_i \times \sum_{j=1}^n \left(\frac{1}{\text{Time}_j} \right)}$$

Geometric Mean

- Used for relative rate or performance numbers

$$\sqrt[n]{\prod_{i=1}^n Relative_Rate_i} = \frac{\sqrt[n]{\prod_{i=1}^n Rate_i}}{Rate_{ref}}$$

- Geometric mean's nice property
 - $\frac{\text{Geometric mean}(X_i)}{X_i} = \text{Geometric mean}\left(\frac{Y_i}{Y_i}\right)$

Using Geometric Mean

	Machine A (norm to B)	Machine B (norm to A)
Program 1	0.5	2.0
Program 2	1.5	0.666
Geometric Mean	0.866	1.155

$$1.155 = 1/0.8666!$$

■ Drawback:

- Geometric mean does **NOT** predict run time

■ Normalizes.

- Each application now counts equally.
- **Advantage: Irrelevance of the reference computer in relative performance**

Summary of comparing performance

- Total execution time or arithmetic mean
 - **consistent** result
 - programs in the workload are **NOT** always run an equal number of times
- Weighted arithmetic mean
 - take into account the frequency of use in the workload
 - solution **depends on which machine is the reference.**
- Normalized Geometric Mean---**SPEC Ratio**
 - **consistent** result, no matter which machine is the reference.
 - Geometric mean does **NOT** predict run time
- Ideal solution : **Measure a real workload and weight the programs according to their frequency of execution.**
- **What really matters is how YOUR application performs**

New SPEC Performance Numbers

- Geometric Mean of 12 (SpecInt) and 14 (SpecFP) Benchmarks
 - Performance measured against SPARC 10/40
- 2000 Performance Numbers (Microprocessor Report, Dec. 2000)

	Alpha 21264B 833MHz	Intel PentiumIII 1GHz	MIPS R12000 400MHz	HP PA-8600 552MHz	IBM Power 3-II 450MHz	Sun Ultra III 900MHz
Int	518	454	320	417	286	438
FP	590	329	319	400	356	369

New SPEC Performance Numbers

- Geometric Mean of 12 (SpecInt) and 14 (SpecFP) Benchmarks
 - Performance measured against SPARC 10/40
- 2001 Performance Numbers (Microprocessor Report, Aug. 2001)

	Alpha 21264C 1001MHz	Intel P4 1.8GHz	MIPS R14000 500MHz	HP PA-8600 552MHz	IBM Power 3-II 450MHz	Sun Ultra III 900MHz
Int	561	599	397	417	286	439
FP	585	615	362	400	356	369

Homework for Chapter 1

- Read the section of 1.7(Putting it all together) and 1.9 (Fallacies and Pitfalls)
- Due time: March 4, 2012, 11:59pm.
- Do the homework in English.
- Submit it to course website.
- STID_name_hw1.pdf , NOT via email.
- NOTES: do the homework not only for the score, but for your understanding of the contents.