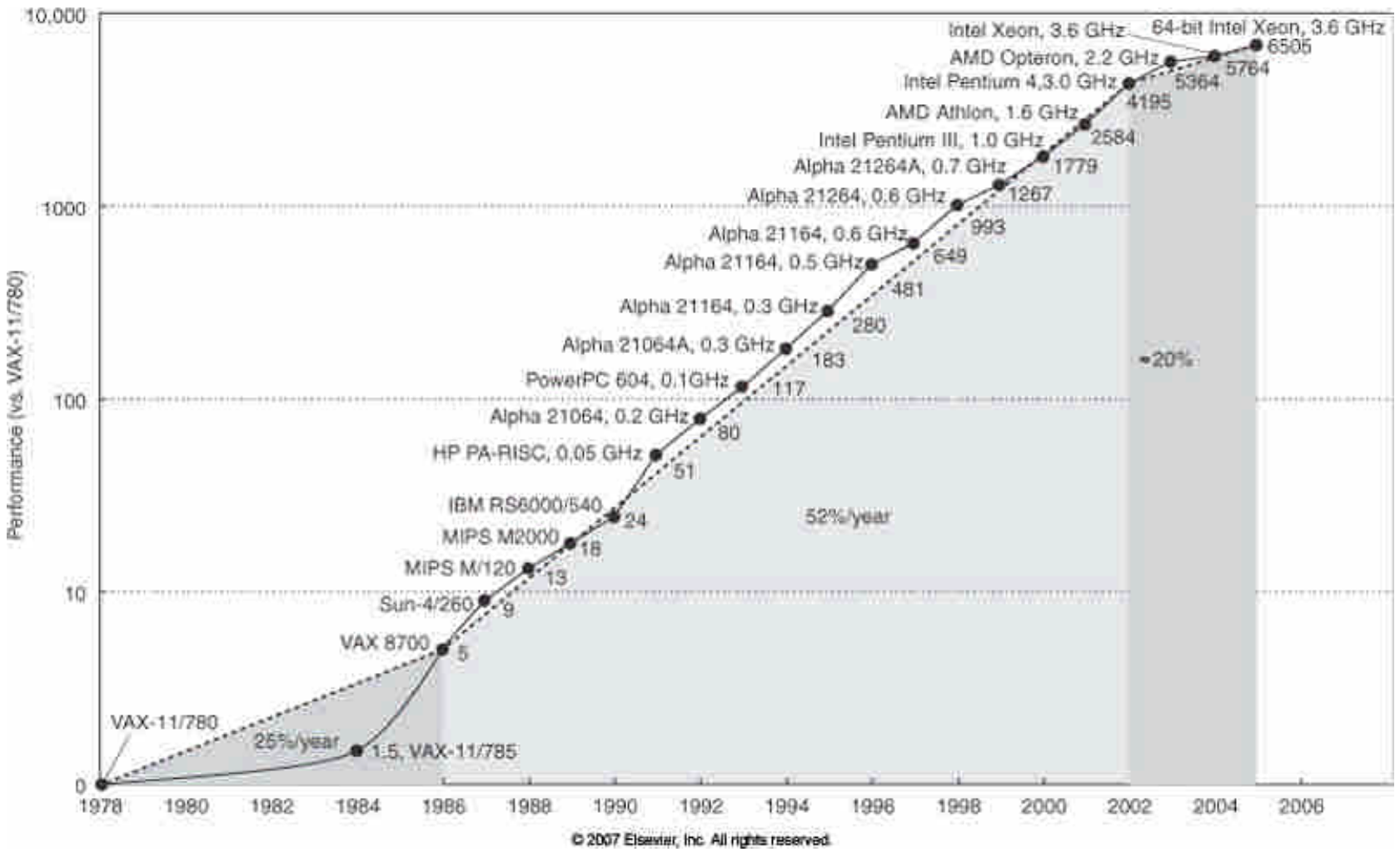
Computer Architecture

Chapter 1
Fundamentals of Computer Design

Outline

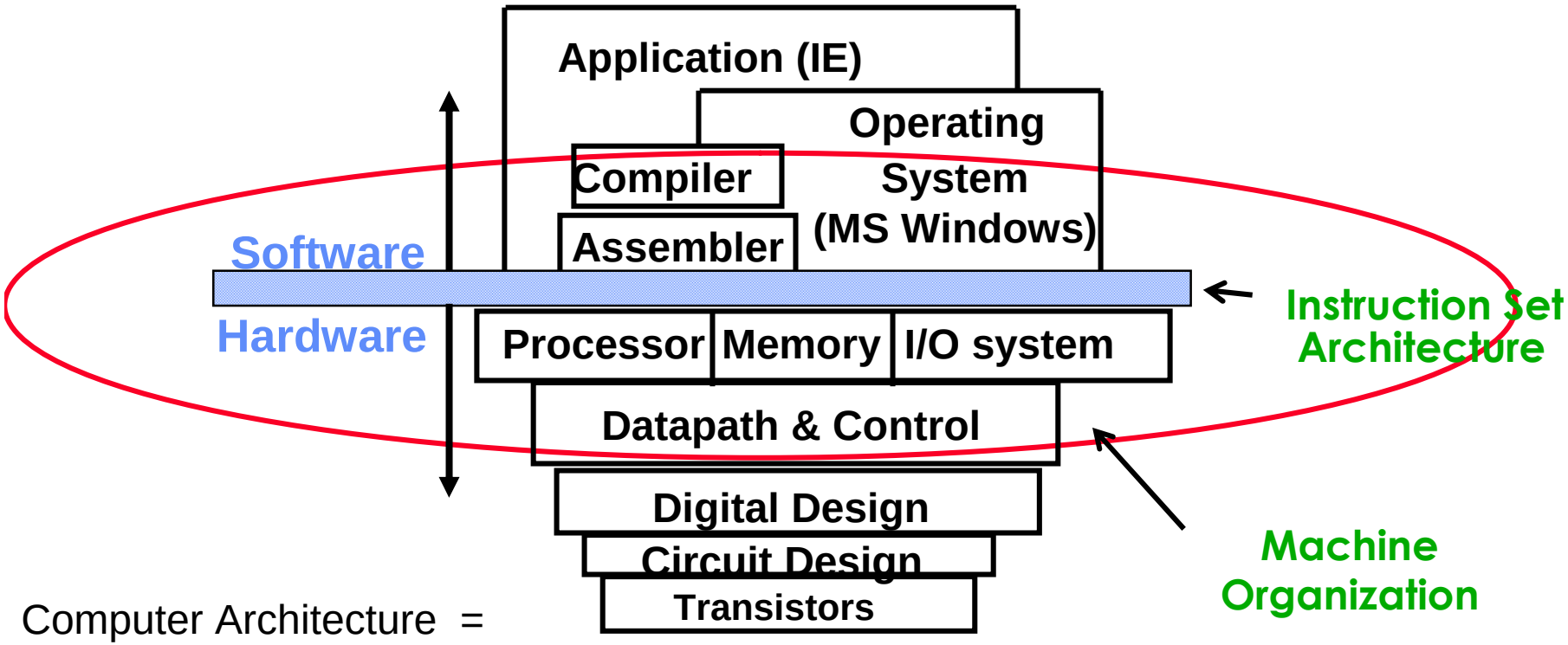
- Computer Science at a Crossroads
- Computer Architecture v. Instruction Set Arch. (1.3)
- Trends in Technology (1.4)
- Trends in Power (1.5)
- Dependability (1.7)
- Measuring, Reporting, and Summarizing Performance (1.8)
- Quantitative Principles of Design (1.9)

Crossroads: Uniprocessor Performance



- VAX : 25%/year 1978 to 1986
- RISC + x86: 52%/year 1986 to 2002
- RISC + x86: ??%/year 2002 to present

What is Computer Architecture?



Computer Architecture =
Instruction Set Architecture
+ Machine Organization

- **Coordination of many *levels of abstraction***
- **Under a rapidly changing set of forces**
- **Design, Measurement, *and* Evaluation**

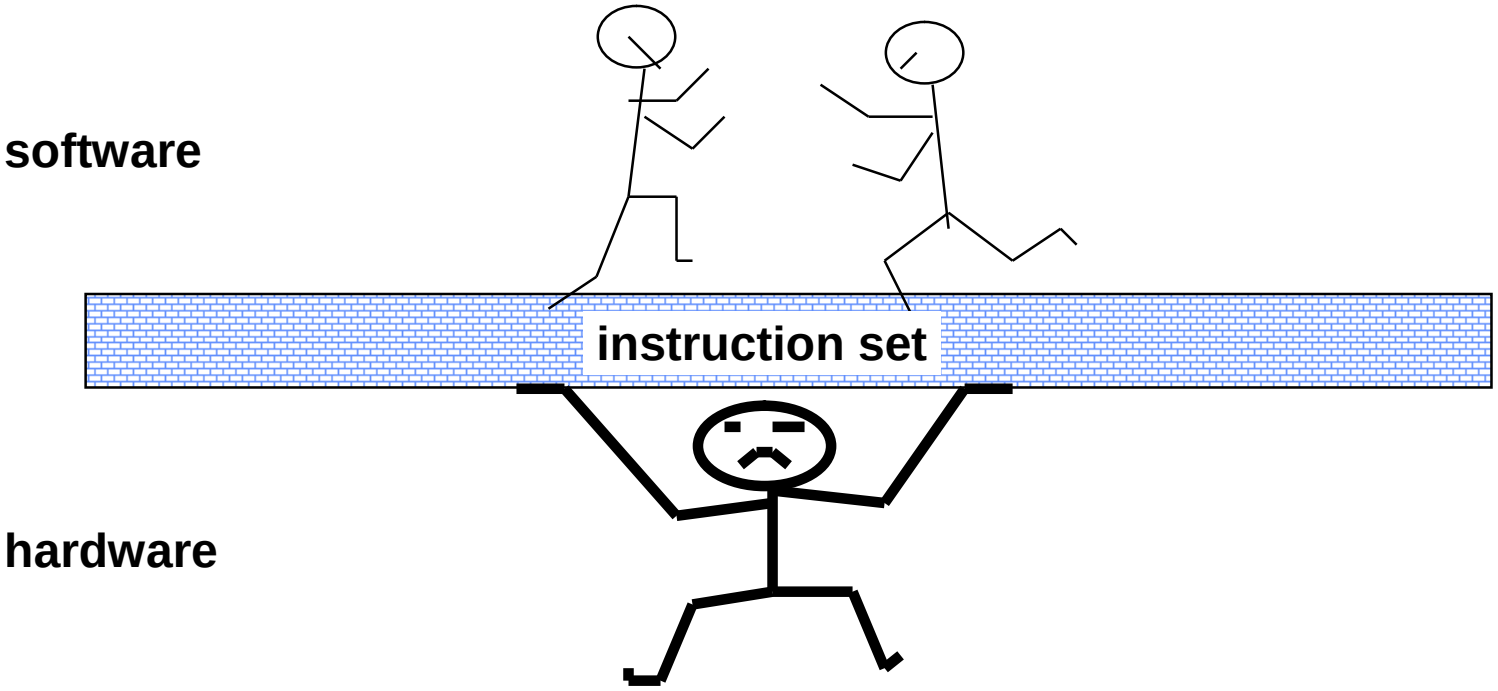
Computer Architecture

- **Instruction set design**
- **Functional design**
- **Logical design**
- **Hardware implementation**

Instruction Set Architecture

- **The actual programmer-visible instruction set**
- **ISA serves as the boundary between the software and hardware.**
- **All recent ISAs are load-store, which can access memory only with load and store instructions.**

Instruction Set Architecture: Critical Interface



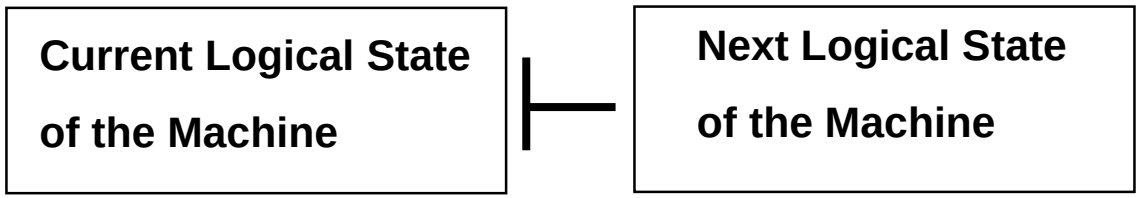
- **Properties of a good abstraction**
  - Lasts through many generations (portability)
  - Used in many different ways (generality)
  - Provides convenient functionality to higher levels
  - Permits an efficient implementation at lower levels

ISA

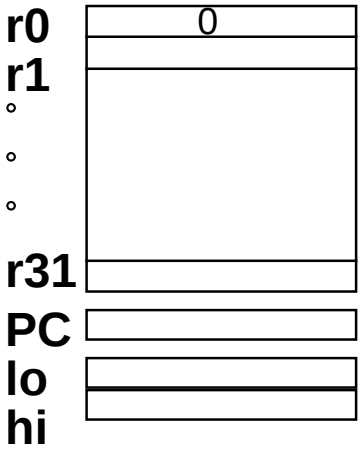
- Class of ISA:
  - register-memory for 80x86
  - Load-store
- Memory addressing
  - Address A is aligned if A mod s =0
- Addressing modes
  - Register, Immediate, and Displacement
- Types and size of operands
  - 8-bit, 16-bit, 32-bit, 64-bit
- Operations
  - Data transfer, arithmetic logic, control, floating point
- Control flow instructions
  - Conditional branch, unconditional jump, procedure call, return
- Encoding an ISA
  - Fixed length, variable length

Computer as a State Machine

- **State: defined by storage**
  - Registers, Memory, Disk, …
- **Next state is influenced by the operation**
  - Instructions, I/O events, interrupts, …
- **When is the next state decided?**
  - Result Store: Register write, Memory write
  - Output: Device (disk, network) write



Example: MIPS



Programmable storage

- 2^32 x bytes
- 31 x 32-bit GPRs (R0=0)
- 32 x 32-bit FP regs (paired DP)
- HI, LO, PC

Data types ?

Format ?

Addressing Modes?

Arithmetic logical

Add, AddU, Sub, SubU, And, Or, Xor, Nor, SLT, SLTU,
AddI, AddIU, SLTI, SLTIU, AndI, OrI, XorI, *LUI*
SLL, SRL, SRA, SLLV, SRLV, SRAV

Memory Access

LB, LBU, LH, LHU, LW, LWL,LWR
SB, SH, SW, SWL, SWR

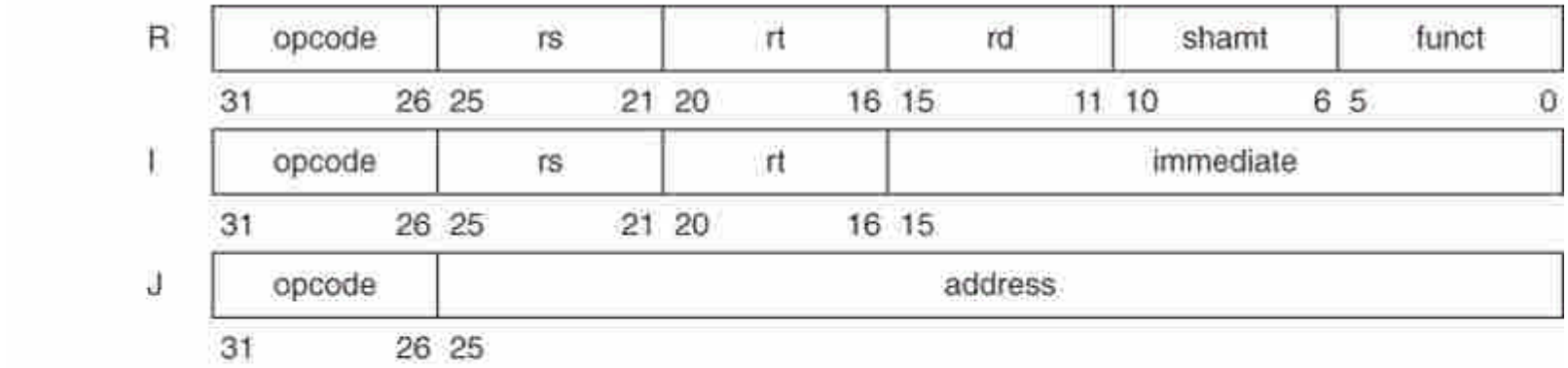
Control

32-bit instructions on word boundary

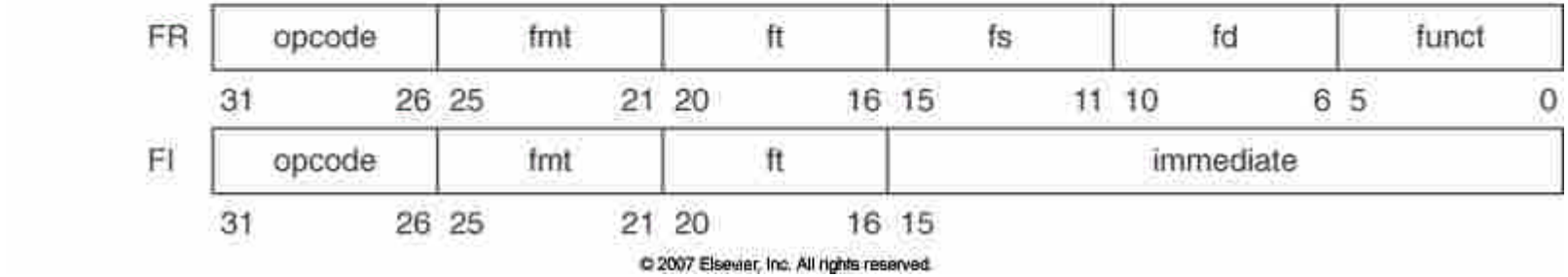
J, JAL, JR, JALR
BEq, BNE, BLEZ,BGTZ,BLTZ,BGEZ,BLTZAL,BGEZAL

MIPS Instruction Set Architecture Formats

Basic instruction formats



Floating-point instruction formats

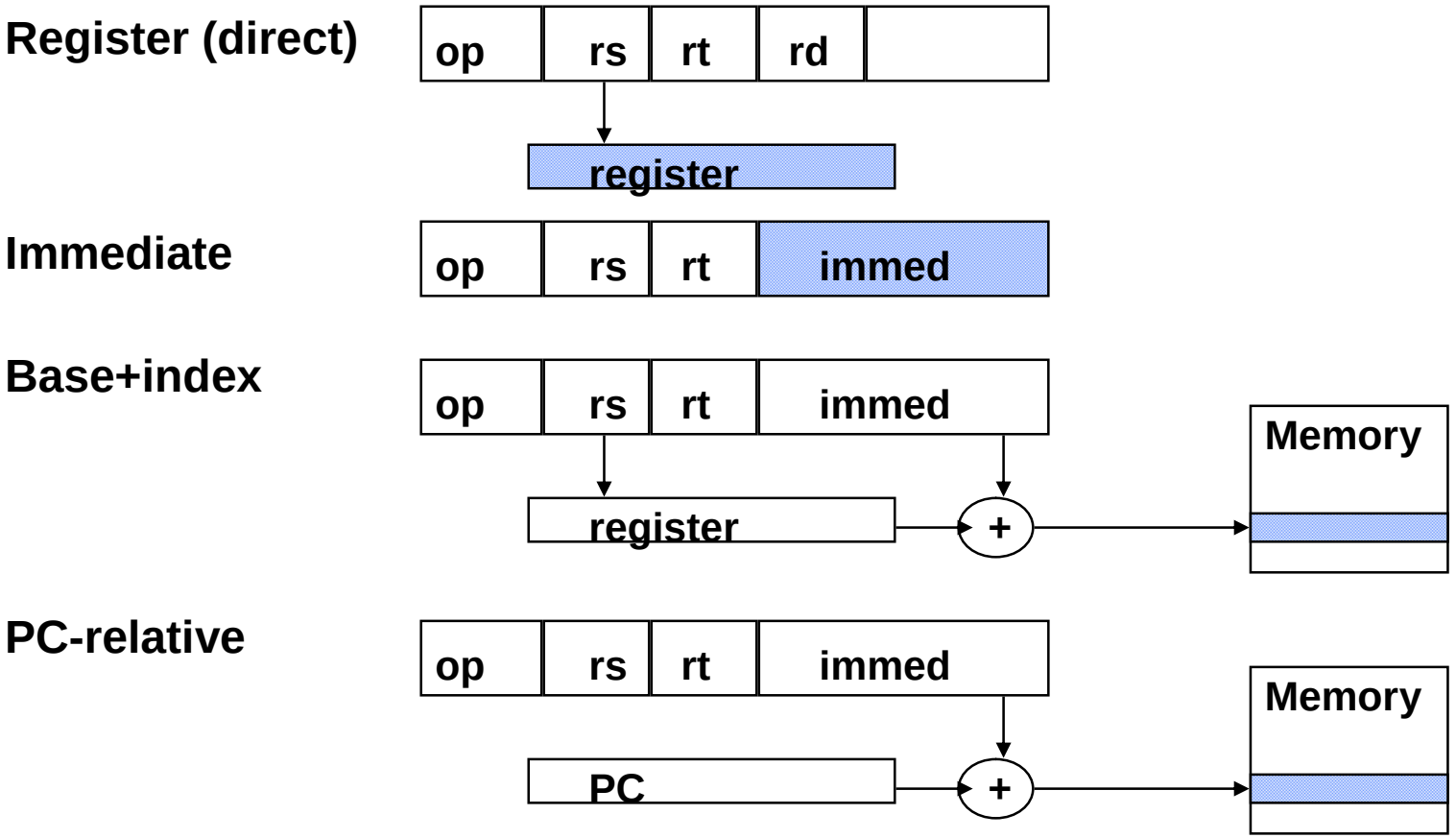


Basic ISA Classes

Accumulator:		
1 address	add A	acc ← acc + mem[A]
1+x address	addx A	acc ← acc + mem[A + x]
Stack:		
0 address	add	tos ← tos + next
General Purpose Register:		
2 address	add A B	EA(A) ← EA(A) + EA(B)
3 address	add A B C	EA(A) ← EA(B) + EA(C)
Load/Store:		
3 address	add Ra Rb Rc	Ra ← Rb + Rc
	load Ra Rb	Ra ← mem[Rb]
	store Ra Rb	mem[Rb] ← Ra

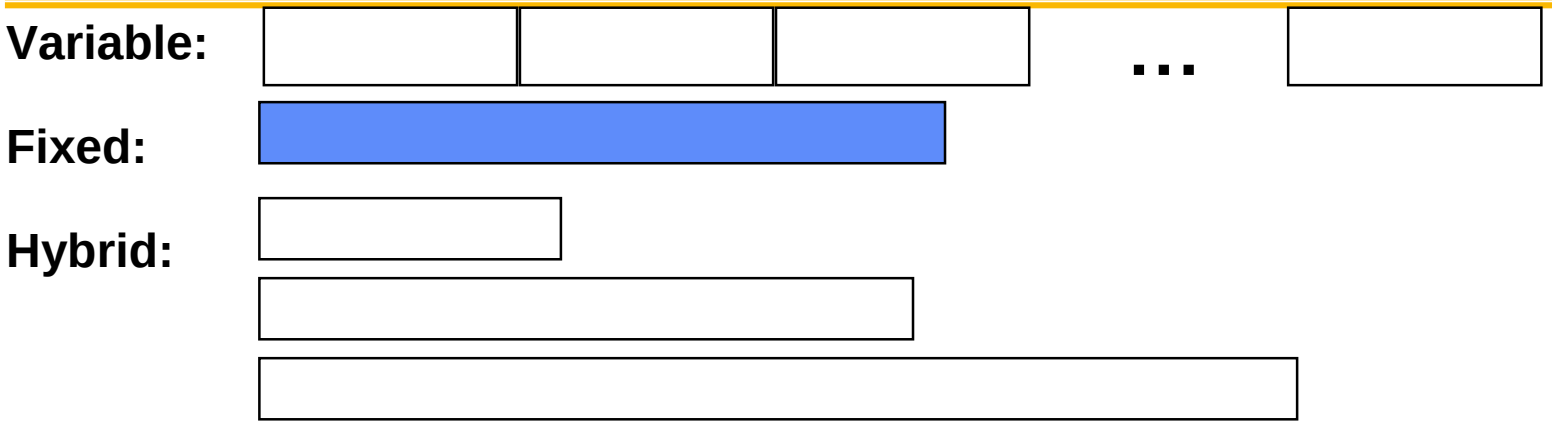
MIPS Addressing Modes & Formats

- Simple addressing modes
- All instructions 32 bits wide



- Register Indirect?

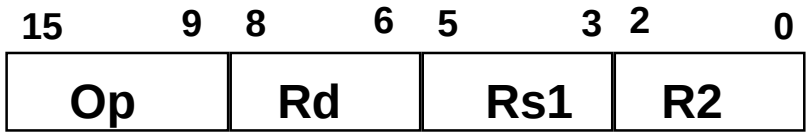
Instruction Formats & RISC



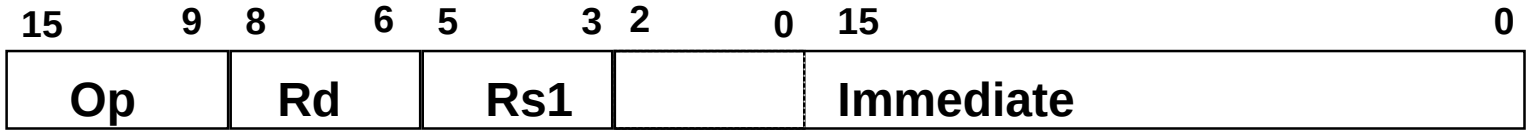
- **Addressing modes**
  - –each operand requires addess specifier => variable format
- **Code size => variable length instructions**
- **Performance => fixed length instructions**
  - –simple decoding, predictable operations
- **RISC: With load/store instruction arch, only one memory address and few addressing modes => simple format, address mode given by opcode** *(Why would RISC perform better than CISC?)*

Cray-1: the Original RISC

Register-Register

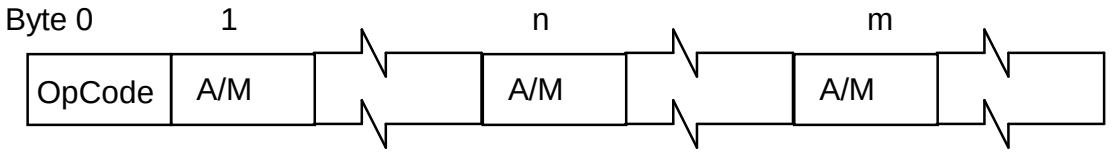


Load, Store and Branch



VAX-11: the Canonical CISC

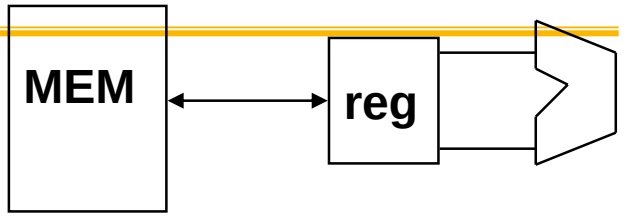
Variable format, 2 and 3 address instruction



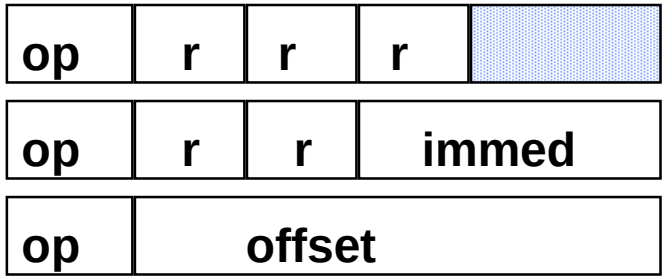
- **Rich set of orthogonal address modes**
  - immediate, offset, indexed, autoinc/dec, indirect, indirect+offset
  - applied to any operand
- **Simple and complex instructions**
  - synchronization instructions
  - data structure operations (queues)
  - polynomial evaluation

1. In programming, canonical means "according to the rules."
2. A canonical book is considered inspired and authoritative and is a part of the rule or standard of faith.

Load/Store Architectures



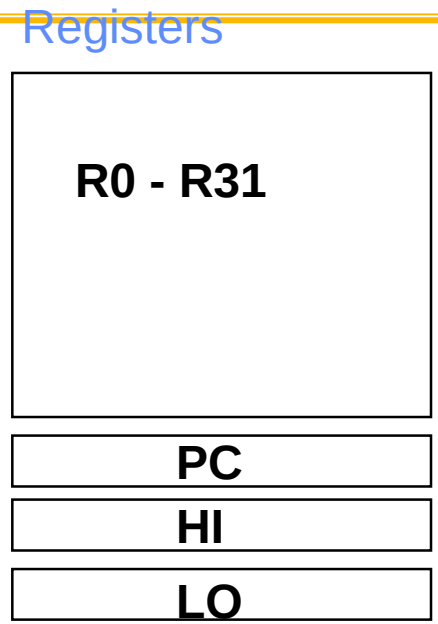
- 3-address GPR
- Register-to-register arithmetic
- Load and store with simple addressing modes (reg + immediate)
- Simple conditionals
  - compare ops + branch z
  - compare&branch
  - condition code + branch on condition
- Simple fixed-format encoding



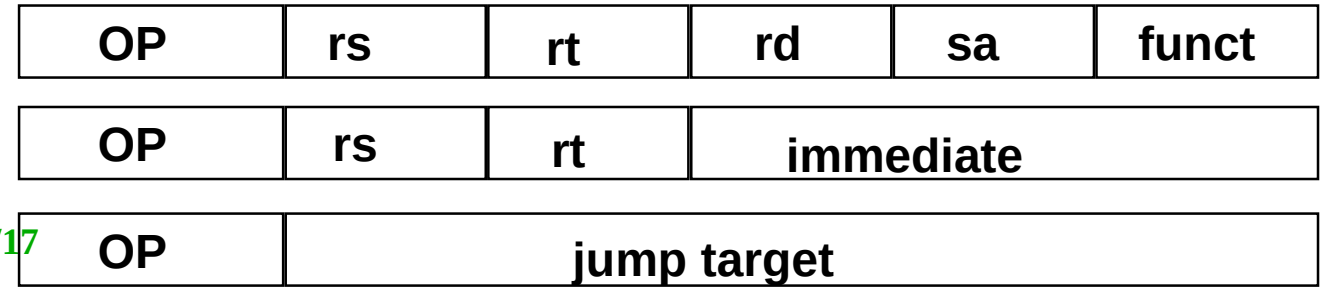
- Substantial increase in instructions
- Decrease in data BW (due to many registers)
- Even more significant decrease in CPI (pipelining)
- Cycle time, Real estate, Design time, Design complexity

MIPS R3000 ISA (Summary)

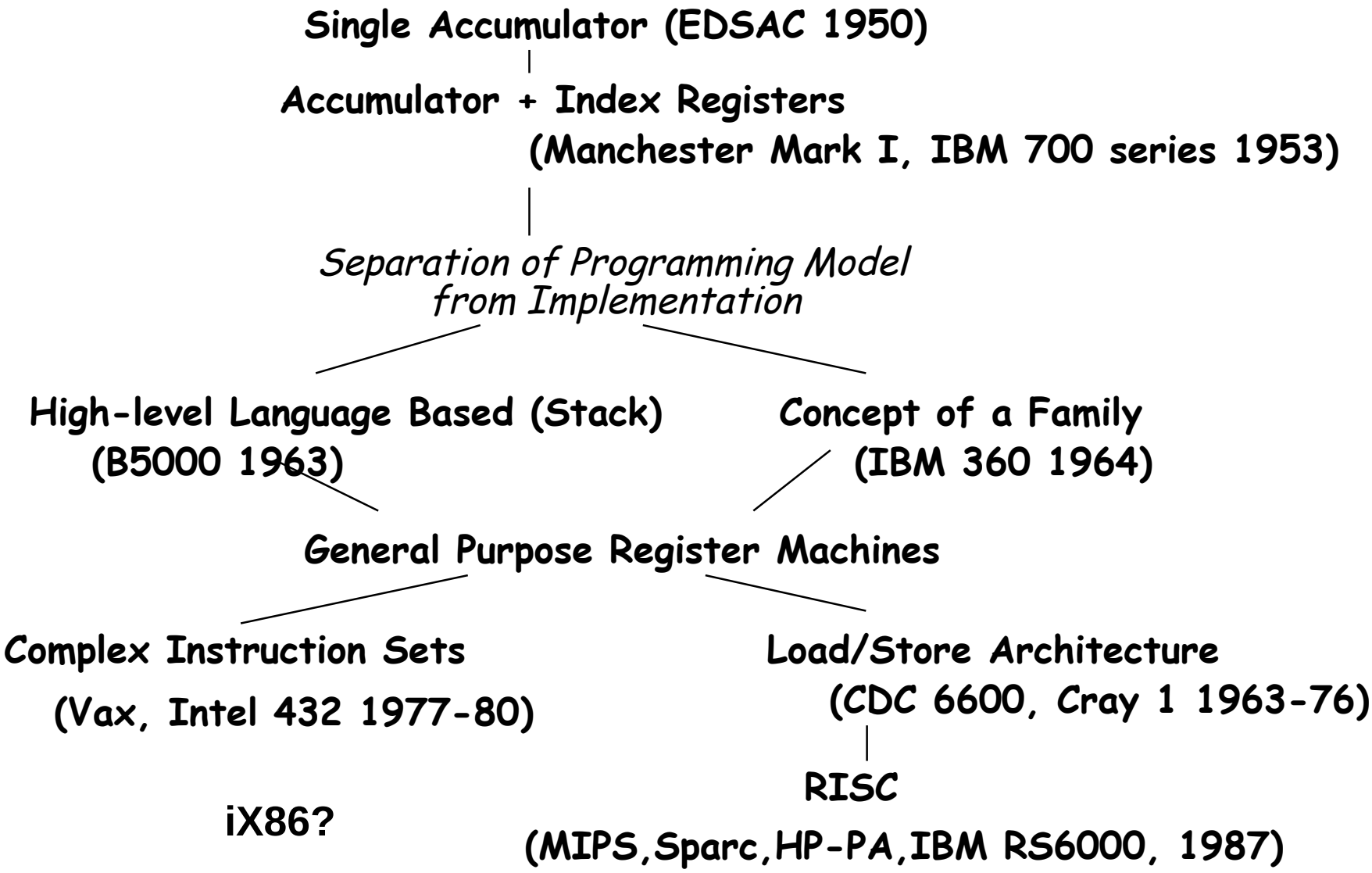
- **Instruction Categories**
  - Load/Store
  - Computational
  - Jump and Branch
  - Floating Point
    - » coprocessor
  - Memory Management
  - Special



3 Instruction Formats: all 32 bits wide



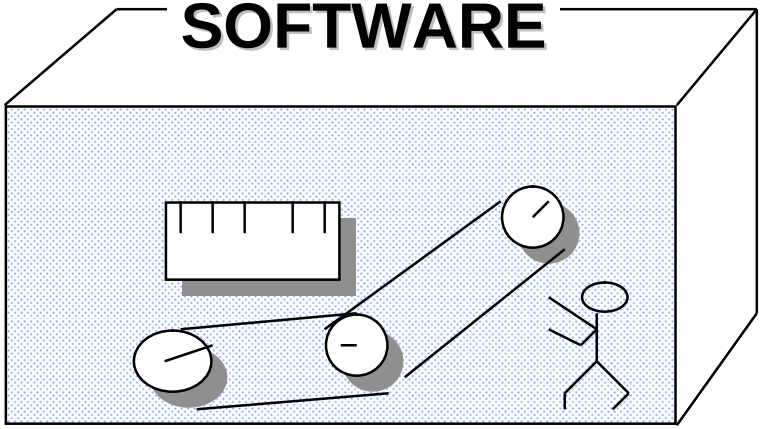
Evolution of Instruction Sets



Instruction Set Architecture

"... the attributes of a [computing] system as seen by the programmer, *i.e.* the conceptual structure and functional behavior, as distinct from the organization of the data flows and controls the logic design, and the physical implementation."

– Amdahl, Blaauw, and Brooks, 1964



- **Organization of Programmable Storage**
- **Data Types & Data Structures: Encodings & Representations**
- **Instruction Formats**
- **Instruction (or Operation Code) Set**
- **Modes of Addressing and Accessing Data Items and Instructions**
- **Exceptional Conditions**

Computer (Machine) Organization

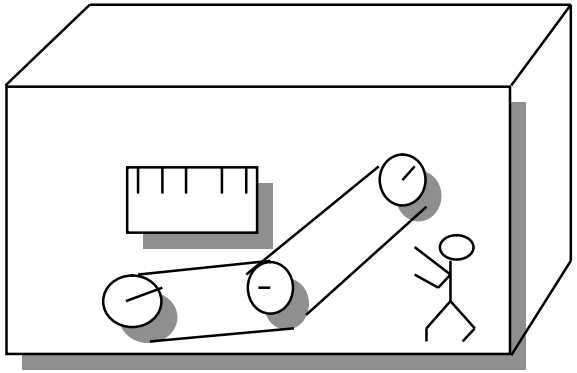
- **Capabilities & Performance Characteristics of Principal Functional Units (FUs)**
  - (Registers, ALU, Shifters, Logic Units, ...)
- **Ways in which these components are interconnected** (Bus, Network, …)
- **Information flows between components** (Data, Messages, Packets, Data path)
- **Logic and means by which such information flow is controlled** (Controller, Protocol handler, Control path, Microcode)

- **Choreography of FUs to realize the ISA** (Execution, Architectural description)
- **Register Transfer Level (RTL) Description** (Implementation description)

Logic Designer's View

ISA Level

FUs & Interconnect



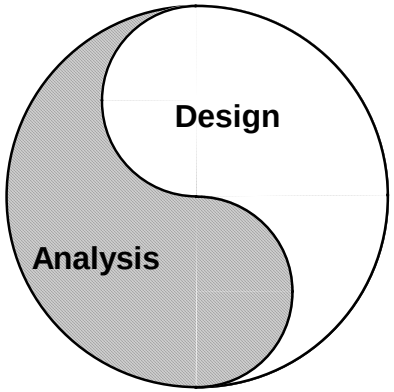
ISA vs. Computer Architecture

- **Old definition of computer architecture = instruction set design**
  - **Other aspects of computer design called implementation**
  - **Insinuates implementation is uninteresting or less challenging**
- **Our view is computer architecture >> ISA**
- **Architect's job much more than instruction set design; technical hurdles today *more* challenging than those in instruction set design**

Comp. Arch. is an Integrated Approach

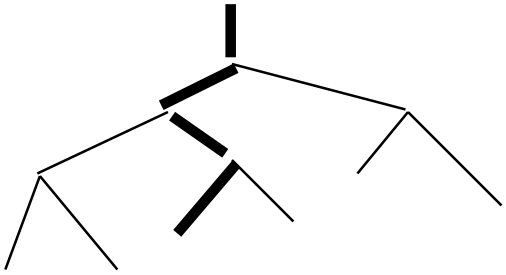
- **What really matters is the functioning of the complete system**
  - **hardware, runtime system, compiler, operating system, and application**
  - **In networking, this is called the "End to End argument"**
- **Computer architecture is not just about transistors, individual instructions, or particular implementations**
  - **E.g., Original RISC projects replaced complex instructions with a compiler + simple instructions**

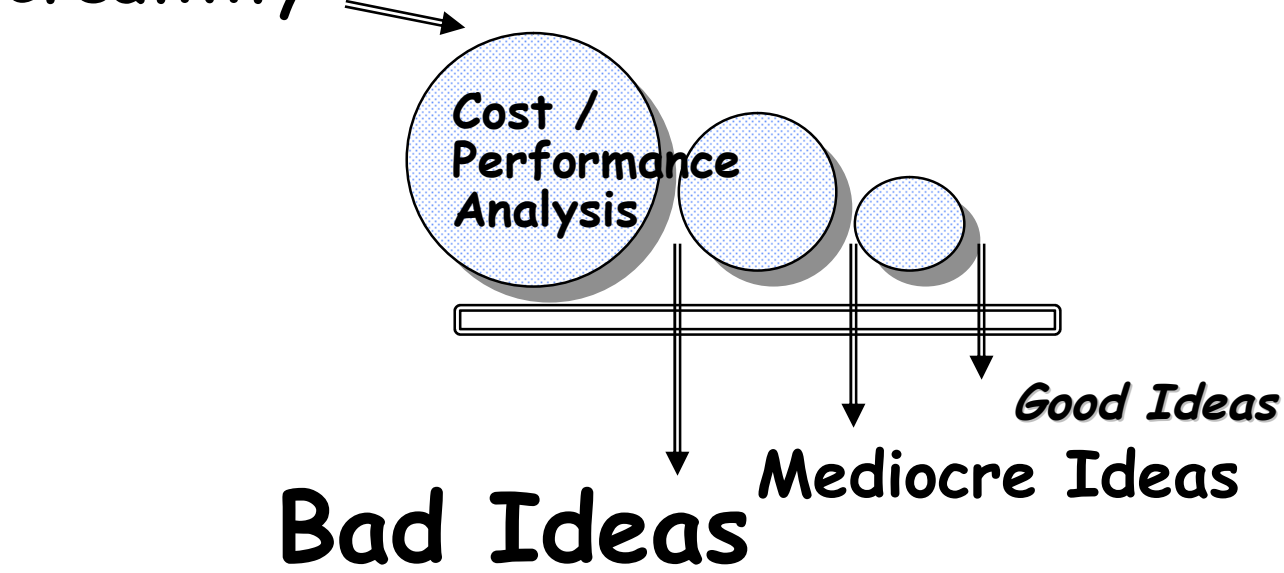
Computer Architecture is Design and Analysis



Architecture is an iterative process:

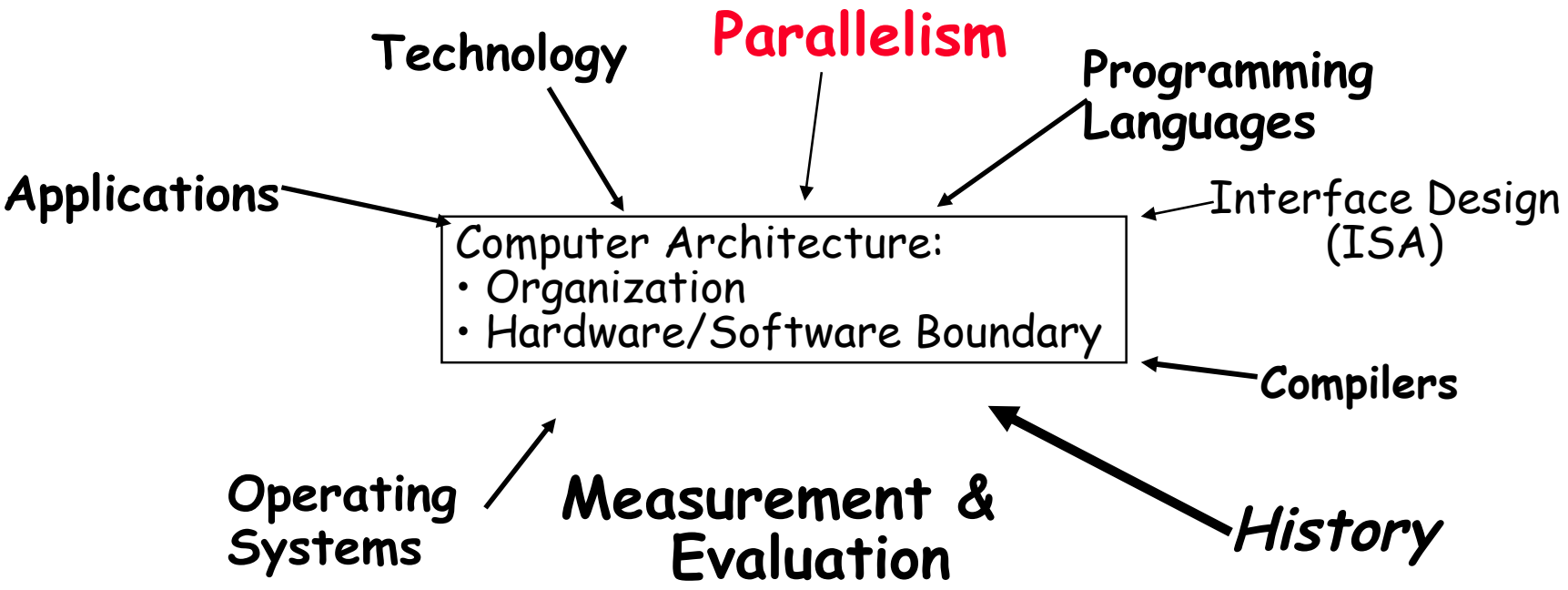
- Searching the space of possible designs
- At all levels of computer systems



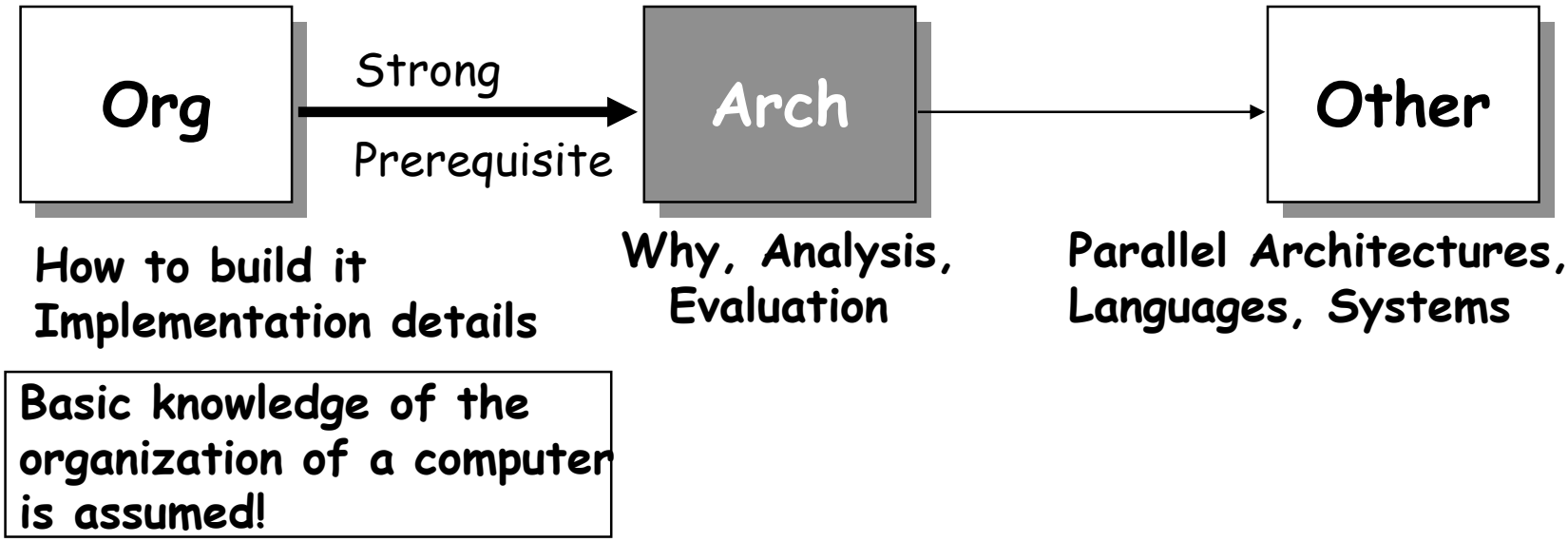


Course Focus

Understanding the design techniques, machine structures, technology factors, evaluation methods that will determine the form of computers in 21st Century



Related Courses



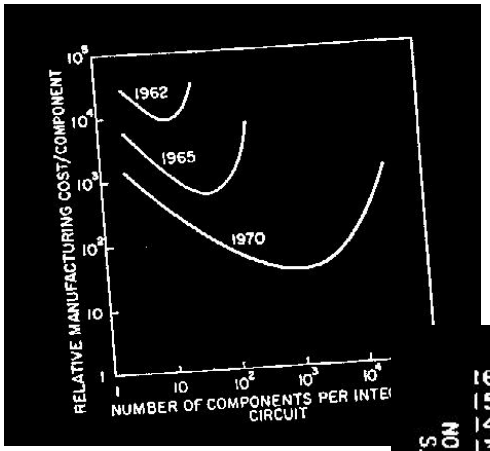
Course Requirement

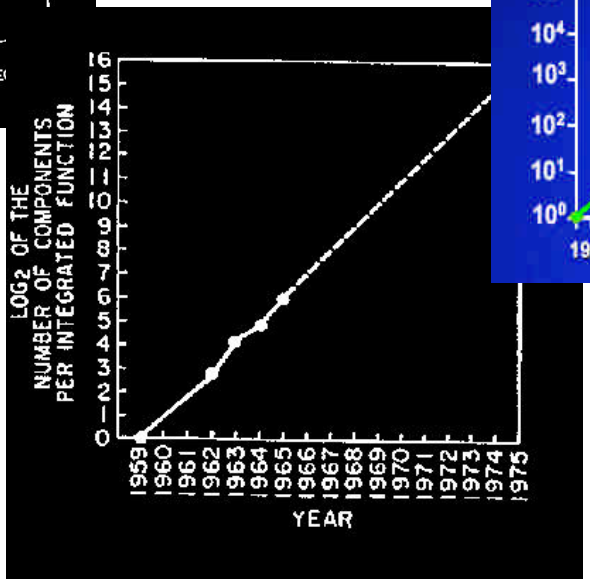
- **Students without computer organization equivalent may have to work hard;**
  - **Review: Appendix A, B, C; *Computer Organization and Design (COD) 3/e***

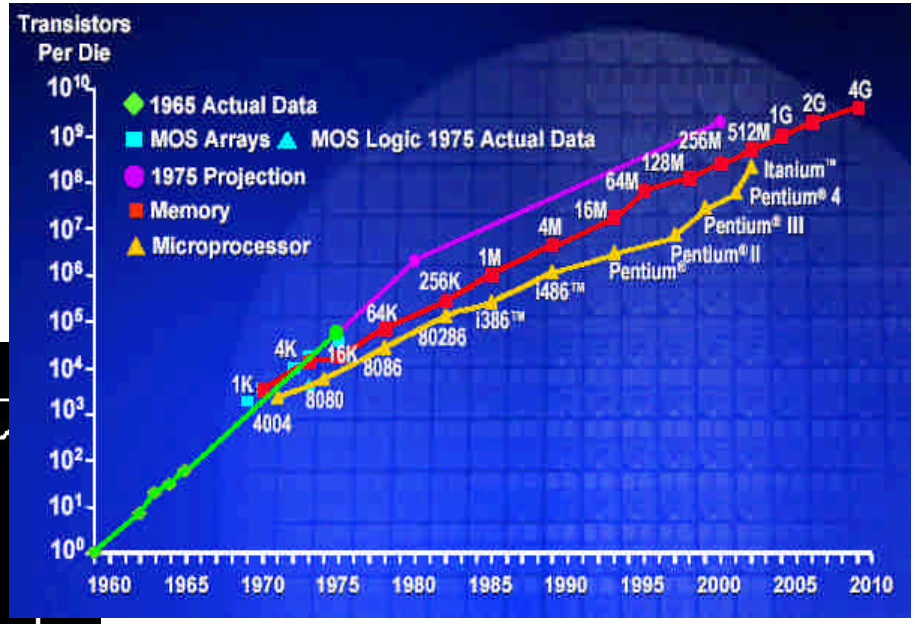
Outline

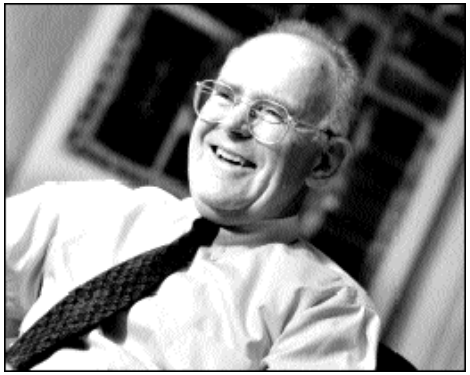
- Computer Science at a Crossroads
- Computer Architecture v. Instruction Set Arch. (1.3)
- **Trends in Technology (1.4)**
- **Trends in Power (1.5)**
- **Dependability (1.7)**
- **Measuring, Reporting, and Summarizing Performance (1.8)**
- **Quantitative Principles of Design (1.9)**

Moore's Law: 2X transistors / "year"









- "Cramming More Components onto Integrated Circuits"
  - Gordon Moore, Electronics, 1965
- # on transistors / cost-effective integrated circuit double every N months ($12 \leq N \leq 24$)

Tracking Technology Performance Trends

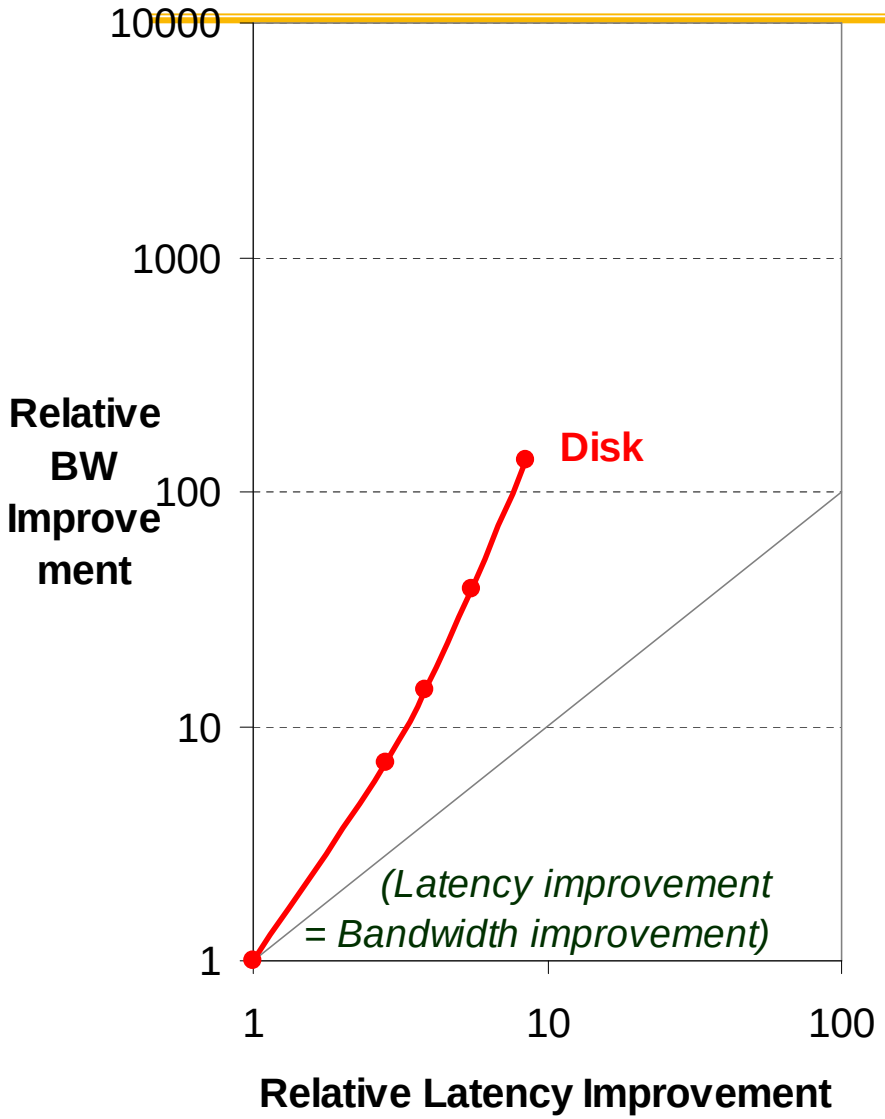
- **Drill down into 4 technologies:**
  - **Disks,**
  - **Memory,**
  - **Network,**
  - **Processors**
- **Compare ~1980 Archaic (Nostalgic) vs. ~2000 Modern (Newfangled)**
  - **Performance Milestones in each technology**
- **Compare for Bandwidth vs. Latency improvements in performance over time**
- **Bandwidth: number of events per unit time**
  - **E.g., M bits / second over network, M bytes / second from disk**
- **Latency: elapsed time for a single event**
  - **E.g., one-way network delay in microseconds, average disk access time in milliseconds**

Disks: Archaic(Nostalgic) v. Modern(Newfangled)

- CDC Wren I, 1983
- 3600 RPM
- 0.03 GBytes capacity
- Tracks/Inch: 800
- Bits/Inch: 9550
- Three 5.25” platters
- Bandwidth: 0.6 MBytes/sec
- Latency: 48.3 ms
- Cache: none

- Seagate 373453, 2003
- 15000 RPM (4X)
- 73.4 GBytes (2500X)
- Tracks/Inch: 64000 (80X)
- Bits/Inch: 533,000 (60X)
- Four 2.5” platters (in 3.5” form factor)
- Bandwidth: 86 MBytes/sec (140X)
- Latency: 5.7 ms (8X)
- Cache: 8 MBytes

Latency Lags Bandwidth (for last ~20 years)



- **Performance Milestones**

- **Disk: 3600, 5400, 7200, 10000, 15000 RPM (8x, 143x)**

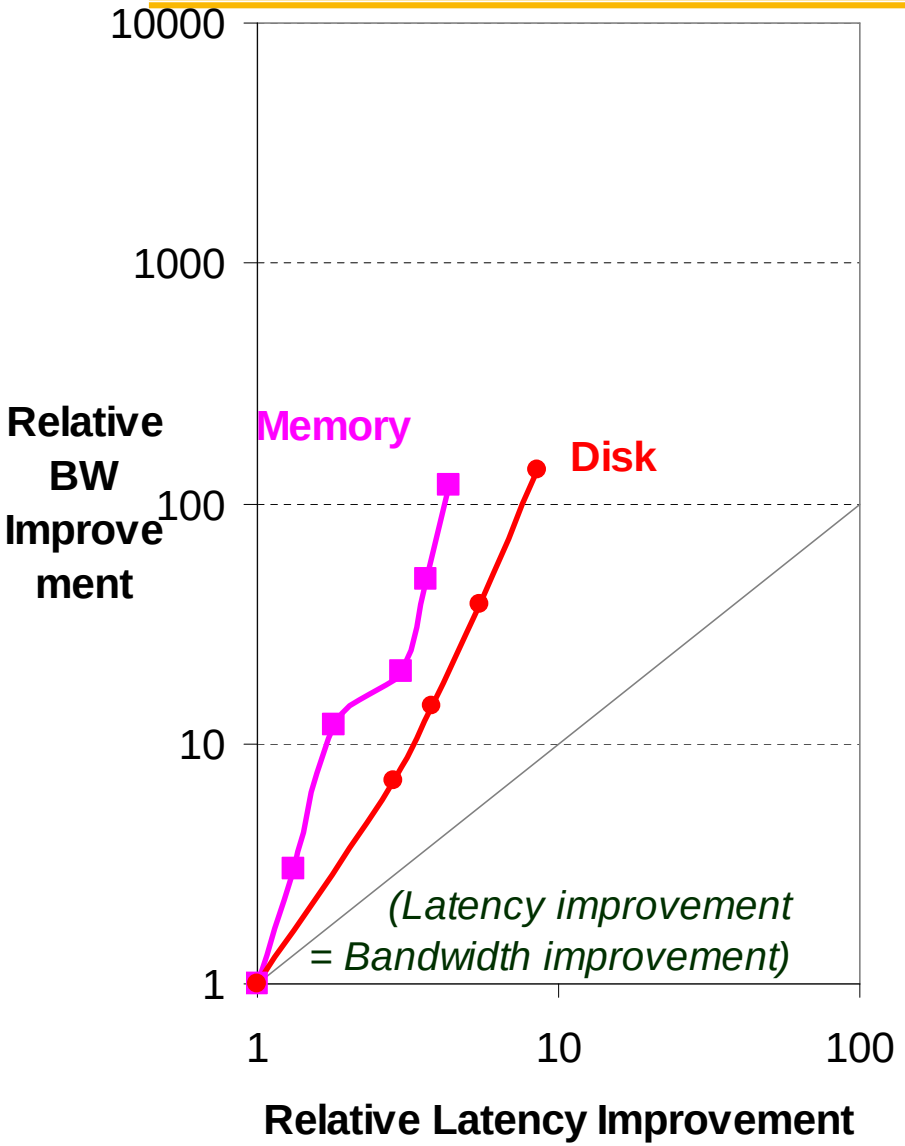
  (latency = simple operation w/o contention
  BW = best-case)

Memory: Archaic (Nostalgic) v. Modern (Newfangled)

- 1980 DRAM (asynchronous)
- 0.06 Mbits/chip
- 64,000 xtors, 35 mm^2
- 16-bit data bus per module, 16 pins/chip
- 13 Mbytes/sec
- Latency: 225 ns
- (no block transfer)

- 2000 Double Data Rate Synchr. (clocked) DRAM
- 256.00 Mbits/chip (4000X)
- 256,000,000 xtors, 204 mm^2
- 64-bit data bus per DIMM, 66 pins/chip (4X)
- 1600 Mbytes/sec (120X)
- Latency: 52 ns (4X)
- Block transfers (page mode)

Latency Lags Bandwidth (last ~20 years)



- **Performance Milestones**

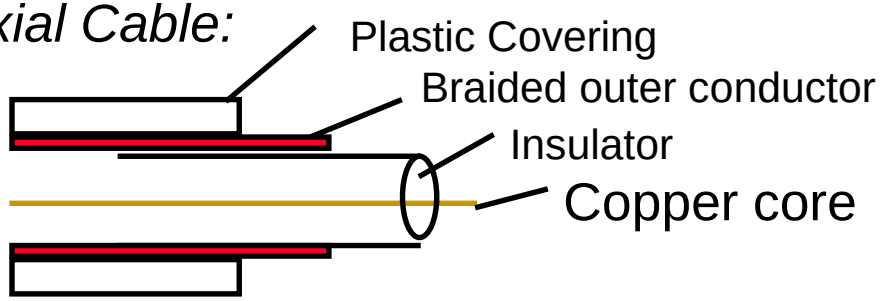
- **Memory Module: 16bit plain DRAM, Page Mode DRAM, 32b, 64b, SDRAM, DDR SDRAM (4x,120x)**
- **Disk: 3600, 5400, 7200, 10000, 15000 RPM (8x, 143x)**

(latency = simple operation w/o contention
BW = best-case)

LANs: Archaic (Nostalgic)v. Modern (Newfangled)

- **Ethernet 802.3**
- **Year of Standard: 1978**
- **10 Mbits/s link speed**
- **Latency: 3000 μsec**
- **Shared media**
- **Coaxial cable**

Coaxial Cable:



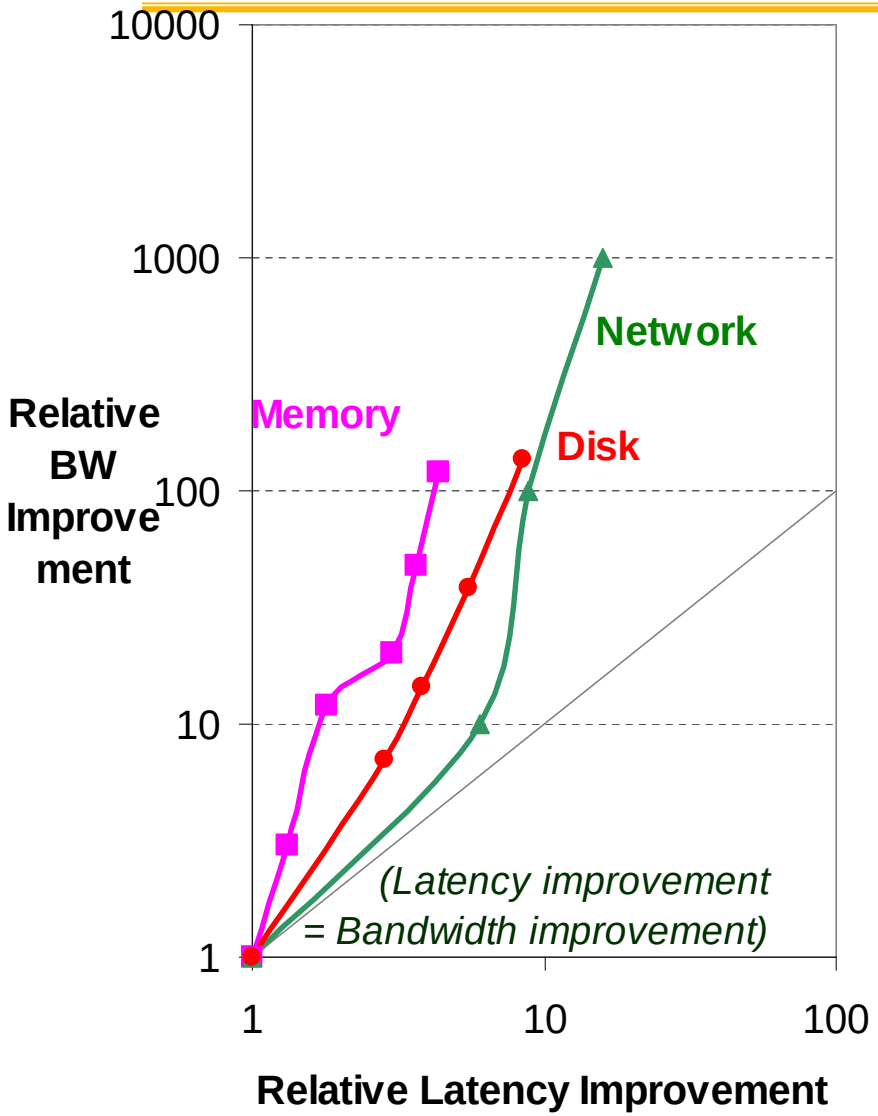
- **Ethernet 802.3ae**
- **Year of Standard: 2003**
- **10,000 Mbits/s (1000X) link speed**
- **Latency: 190 μsec (15X)**
- **Switched media**
- **Category 5 copper wire**

"Cat 5" is 4 twisted pairs in bundle

Twisted Pair:

Copper, 1mm thick,
twisted to avoid antenna effect

Latency Lags Bandwidth (last ~20 years)



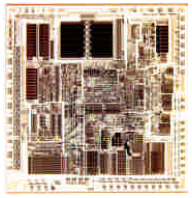
- **Performance Milestones**

- **Ethernet: 10Mb, 100Mb, 1000Mb, 10000 Mb/s** (16x,1000x)
- **Memory Module: 16bit plain DRAM, Page Mode DRAM, 32b, 64b, SDRAM, DDR SDRAM** (4x,120x)
- **Disk: 3600, 5400, 7200, 10000, 15000 RPM** (8x, 143x)

(latency = simple operation w/o contention
BW = best-case)

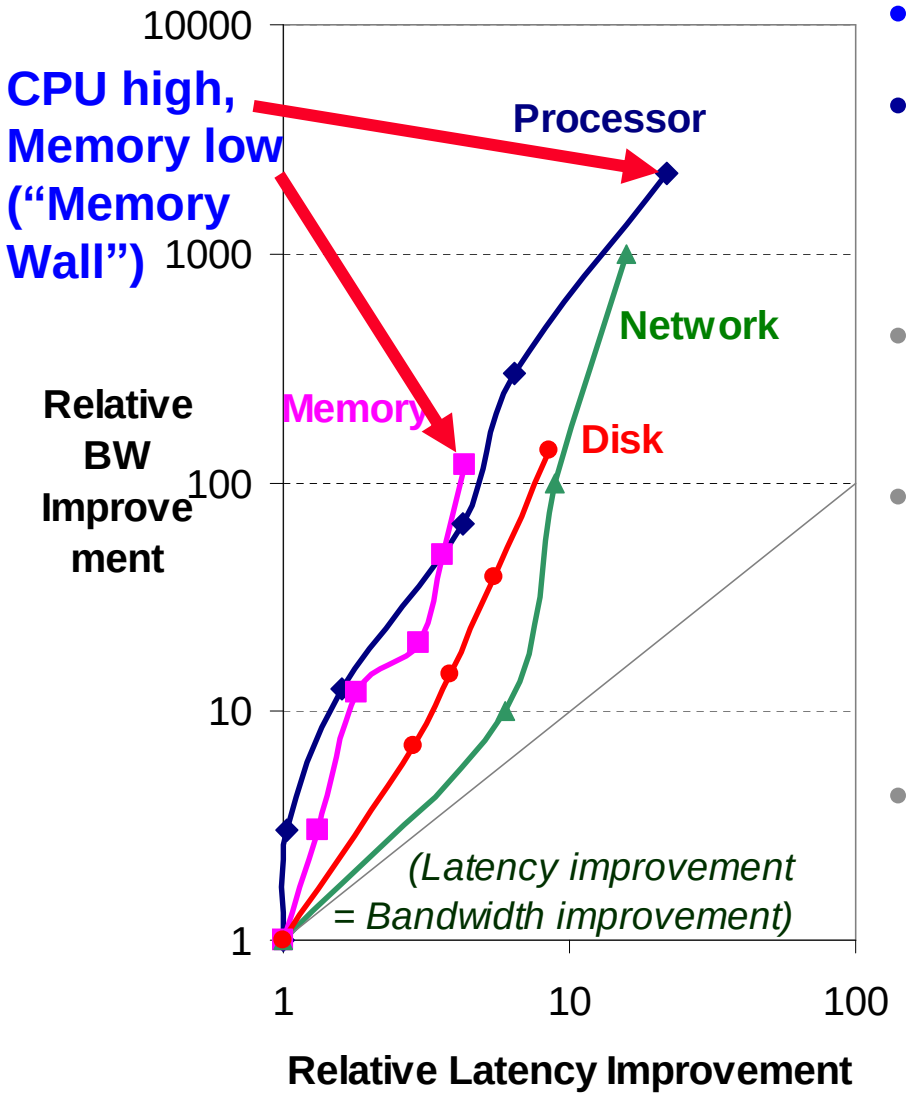
CPUs: Archaic (Nostalgic) v. Modern (Newfangled)

- 1982 Intel 80286
- 12.5 MHz
- 2 MIPS (peak)
- Latency 320 ns
- 134,000 xtors, 47 mm²
- 16-bit data bus, 68 pins
- Microcode interpreter, separate FPU chip
- (no caches)



- 2001 Intel Pentium 4
- 1500 MHz (120X)
- 4500 MIPS (peak) (2250X)
- Latency 15 ns (20X)
- 42,000,000 xtors, 217 mm²
- 64-bit data bus, 423 pins
- 3-way superscalar, Dynamic translate to RISC, Superpipelined (22 stage), Out-of-Order execution
- On-chip 8KB Data caches, 96KB Instr. Trace cache, 256KB L2 cache

Latency Lags Bandwidth (last ~20 years)



- **Performance Milestones**
- **Processor: '286, '386, '486, Pentium, Pentium Pro, Pentium 4** (21x,2250x)
- **Ethernet: 10Mb, 100Mb, 1000Mb, 10000 Mb/s** (16x,1000x)
- **Memory Module: 16bit plain DRAM, Page Mode DRAM, 32b, 64b, SDRAM, DDR SDRAM** (4x,120x)
- **Disk : 3600, 5400, 7200, 10000, 15000 RPM** (8x, 143x)

Rule of Thumb for Latency Lagging BW

- **In the time that bandwidth doubles, latency improves by no more than a factor of 1.2 to 1.4**
  (and capacity improves faster than bandwidth)
- **Stated alternatively:**
  Bandwidth improves by more than the square of the improvement in Latency

6 Reasons Latency Lags Bandwidth

1. Moore's Law helps BW more than latency

- **Faster transistors, more transistors, more pins help Bandwidth**
  - » **MPU Transistors: 0.130 vs. 42 M xtors (300X)**
  - » **DRAM Transistors: 0.064 vs. 256 M xtors (4000X)**
  - » **MPU Pins: 68 vs. 423 pins (6X)**
  - » **DRAM Pins: 16 vs. 66 pins (4X)**
- **Smaller, faster transistors but communicate over (relatively) longer lines: limits latency**
  - » **Feature size: 1.5 to 3 vs. 0.18 micron (8X,17X)**
  - » **MPU Die Size: 35 vs. 204 mm^2 (ratio sqrt $\Rightarrow$ 2X)**
  - » **DRAM Die Size: 47 vs. 217 mm^2 (ratio sqrt $\Rightarrow$ 2X)**

6 Reasons Latency Lags Bandwidth (cont'd)

2. Distance limits latency

- **Size of DRAM block ⇒ long bit and word lines**
  ⇒ most of DRAM access time
- **Speed of light and computers on network**
- **1. & 2. explains linear latency vs. square BW?**

3. Bandwidth easier to sell ("bigger=better")

- **E.g., 10 Gbits/s Ethernet ("10 Gig") vs.**
  10 μsec latency Ethernet
- **4400 MB/s DIMM ("PC4400") vs. 50 ns latency**
- **Even if just marketing, customers now trained**
- **Since bandwidth sells, more resources thrown at bandwidth, which further tips the balance**

6 Reasons Latency Lags Bandwidth (cont'd)

4. Latency helps BW, but not vice versa

- **Spinning disk faster improves both bandwidth and rotational latency**
  - » **3600 RPM ⇒ 15000 RPM = 4.2X**
  - » **Average rotational latency: 8.3 ms ⇒ 2.0 ms**
  - » **Things being equal, also helps BW by 4.2X**
- **Lower DRAM latency ⇒ More access/second (higher bandwidth)**
- **Higher linear density helps disk BW (and capacity), but not disk Latency**
  - » **9,550 BPI ⇒ 533,000 BPI ⇒ 60X in BW**

6 Reasons Latency Lags Bandwidth (cont'd)

5. Bandwidth hurts latency

- **Queues help Bandwidth, hurt Latency (Queuing Theory)**
- **Adding chips to widen a memory module increases Bandwidth but higher fan-out on address lines may increase Latency**

6. Operating System overhead hurts Latency more than Bandwidth

- **Long messages amortize overhead; overhead bigger part of short messages**

Summary of Technology Trends

- **For disk, LAN, memory, and microprocessor, bandwidth improves by square of latency improvement**
  - **In the time that bandwidth doubles, latency improves by no more than 1.2X to 1.4X**
- **Lag probably even larger in real systems, as bandwidth gains multiplied by replicated components**
  - **Multiple processors in a cluster or even in a chip**
  - **Multiple disks in a disk array**
  - **Multiple memory modules in a large memory**
  - **Simultaneous communication in switched LAN**
- **HW and SW developers should innovate assuming Latency Lags Bandwidth**
  - **If everything improves at the same rate, then nothing really changes**
  - **When rates vary, require real innovation**

Outline

- Computer Science at a Crossroads
- Computer Architecture v. Instruction Set Arch. (1.3)
- Trends in Technology (1.4)
- **Trends in Power (1.5)**
- **Dependability (1.7)**
- **Measuring, Reporting, and Summarizing Performance (1.8)**
- **Quantitative Principles of Design (1.9)**

Define and quantity power (1 / 2)

- **For CMOS chips, traditional dominant energy consumption has been in switching transistors, called *dynamic power***

$$Power_{dynamic} = 1/2 \times CapacitiveLoad \times Voltage^2 \times FrequencySwitched$$

- **For mobile devices, energy better metric**

$$Energy_{dynamic} = CapacitiveLoad \times Voltage^2$$

- **For a fixed task, slowing clock rate (frequency switched) reduces power, but not energy**
- **Capacitive load a function of number of transistors connected to output and technology, which determines capacitance of wires and transistors**
- **Dropping voltage helps both, so went from 5V to 1V**
- **To save energy & dynamic power, most CPUs now turn off clock of inactive modules (e.g. Fl. Pt. Unit)**

Example of quantifying power

- **Suppose 15% reduction in voltage results in a 15% reduction in frequency. What is impact on dynamic power?**

$$
\begin{aligned}
Power_{dynamic} &= 1/2 \times CapacitiveLoad \times Voltage^2 \times FrequencySwitched \\
&= 1/2 \times .85 \times CapacitiveLoad \times (.85 \times Voltage)^2 \times FrequencySwitched \\
&= (.85)^3 \times OldPower_{dynamic} \\
&\approx 0.6 \times OldPower_{dynamic}
\end{aligned}
$$

Define and quantity power (2 / 2)

- **Because leakage current flows even when a transistor is off, now *static power* important too**

$$Power_{static} = Current_{static} \times Voltage$$

- **Leakage current increases in processors with smaller transistor sizes**
- **Increasing the number of transistors increases power even if they are turned off**
- **In 2006, goal for leakage is 25% of total power consumption; high performance designs at 40%**
- **Very low power systems even gate voltage to inactive modules to control loss due to leakage**

Outline

- Computer Science at a Crossroads
- Computer Architecture v. Instruction Set Arch. (1.3)
- Trends in Technology (1.4)
- Trends in Power (1.5)
- **Dependability (1.7)**
- **Measuring, Reporting, and Summarizing Performance (1.8)**
- **Quantitative Principles of Design (1.9)**

Define and quantity dependability (1/3)

- How decide when a system is operating properly?
- Infrastructure providers now offer Service Level Agreements (SLA) to guarantee that their networking or power service would be dependable
- Systems alternate between 2 states of service with respect to an SLA:

1. Service accomplishment, where the service is delivered as specified in SLA
2. Service interruption, where the delivered service is different from the SLA

- Failure = transition from state 1 to state 2
- Restoration = transition from state 2 to state 1

Define and quantity dependability (2/3)

- *Module reliability* = measure of continuous service accomplishment (or time to failure).
  2 metrics

1. *Mean Time To Failure* (*MTTF*) measures Reliability
2. *Failures In Time* (*FIT*) = 1/MTTF, the rate of failures
   - Traditionally reported as failures per billion hours of operation

- *Mean Time To Repair* (*MTTR*) measures Service Interruption
  - *Mean Time Between Failures* (*MTBF*) = MTTF+MTTR
- *Module availability* measures service as alternate between the 2 states of accomplishment and interruption (number between 0 and 1, e.g. 0.9)
- *Module availability = MTTF / (MTTF + MTTR)*

Example calculating reliability

- **If modules have *exponentially distributed lifetimes* (age of module does not affect probability of failure), overall failure rate is the sum of failure rates of the modules**
- **Calculate FIT and MTTF for 10 disks (1M hour MTTF per disk), 1 disk controller (0.5M hour MTTF), and 1 power supply (0.2M hour MTTF):**

$$FailureRate =$$

$$MTTF =$$

Example calculating reliability

- **If modules have *exponentially distributed lifetimes* (age of module does not affect probability of failure), overall failure rate is the sum of failure rates of the modules**
- **Calculate FIT and MTTF for 10 disks (1M hour MTTF per disk), 1 disk controller (0.5M hour MTTF), and 1 power supply (0.2M hour MTTF):**

$$\begin{aligned} FailureRate &= 10 \times (1/1{,}000{,}000) + 1/500{,}000 + 1/200{,}000 \\ &= 10 + 2 + 5/1{,}000{,}000 \\ &= 17/1{,}000{,}000 \\ &= 17{,}000 FIT \\ MTTF &= 1{,}000{,}000{,}000/17{,}000 \\ &\approx 59{,}000 hours \end{aligned}$$

Outline

- Computer Science at a Crossroads
- Computer Architecture v. Instruction Set Arch. (1.3)
- Trends in Technology (1.4)
- Trends in Power (1.5)
- Dependability (1.7)
- **Measuring, Reporting, and Summarizing Performance (1.8)**
- **Quantitative Principles of Design (1.9)**

Definition: Performance

- **Performance is in units of things per sec**
  - **bigger is better**
- **If we are primarily concerned with response time**

$$\text{performance}(x) = \frac{1}{\text{execution\_time}(x)}$$

" X is n times faster than Y" means

$$n = \frac{\text{Performance}(X)}{\text{Performance}(Y)} = \frac{\text{Execution\_time}(Y)}{\text{Execution\_time}(X)}$$

Performance: What to measure

- Usually rely on benchmarks vs. real workloads
- To increase predictability, collections of benchmark applications, called *benchmark suites*, are popular
- SPECCPU: popular desktop benchmark suite
  - CPU only, split between integer and floating point programs
  - SPECint2000 has 12 integer, SPECfp2000 has 14 integer pgms
  - SPECCPU2006 to be announced Spring 2006
  - SPECSFS (NFS file server) and SPECWeb (WebServer) added as server benchmarks
- Transaction Processing Council measures server performance and cost-performance for databases
  - TPC-C Complex query for Online Transaction Processing
  - TPC-H models ad hoc decision support
  - TPC-W a transactional web benchmark
  - TPC-App application server and web services benchmark

How Summarize Suite Performance (1/5)

- Arithmetic average of execution time of all pgms?
  - But they vary by 4X in speed, so some would be more important than others in arithmetic average
- Could add a weights per program, but how pick weight?
  - Different companies want different weights for their products
- SPECRatio: Normalize execution times to reference computer, yielding a ratio proportional to performance =

$$\frac{\text{time on reference computer}}{\text{time on computer being rated}}$$

How Summarize Suite Performance (2/5)

- **If program SPECRatio on Computer A is 1.25 times bigger than Computer B, then**

$$1.25 = \frac{SPECRatio_A}{SPECRatio_B} = \frac{\dfrac{ExecutionTime_{reference}}{ExecutionTime_A}}{\dfrac{ExecutionTime_{reference}}{ExecutionTime_B}}$$

$$= \frac{ExecutionTime_B}{ExecutionTime_A} = \frac{Performance_A}{Performance_B}$$

  - **Note that when comparing 2 computers as a ratio, execution times on the reference computer drop out, so choice of reference computer is irrelevant**

How Summarize Suite Performance (3/5)

- **Since ratios, proper mean is geometric mean (SPECRatio unitless, so arithmetic mean meaningless)**

$$GeometricMean = \sqrt[n]{\prod_{i=1}^{n} SPECRatio_i}$$

1. **Geometric mean of the ratios is the same as the ratio of the geometric means**
2. **Ratio of geometric means = Geometric mean of performance ratios $\Rightarrow$ choice of reference computer is irrelevant!**

- **These two points make geometric mean of ratios attractive to summarize performance**

How Summarize Suite Performance (4/5)

- **Does a single mean well summarize performance of programs in benchmark suite?**
- **Can decide if mean a good predictor by characterizing variability of distribution using standard deviation**
- **Like geometric mean, geometric standard deviation is multiplicative rather than arithmetic**
- **Can simply take the logarithm of SPECRatios, compute the standard mean and standard deviation, and then take the exponent to convert back:**

$$GeometricMean = \exp\left(\frac{1}{n} \times \sum_{i=1}^{n} \ln\left(SPECRatio_i\right)\right)$$

$$GeometricStDev = \exp\left(StDev\left(\ln\left(SPECRatio_i\right)\right)\right)$$

How Summarize Suite Performance (5/5)

- **Standard deviation is more informative if know distribution has a standard form**
  - ***bell-shaped normal distribution*, whose data are symmetric around mean**
  - ***lognormal distribution*, where logarithms of data--not data itself--are normally distributed (symmetric) on a logarithmic scale**
- **For a lognormal distribution, we expect that**

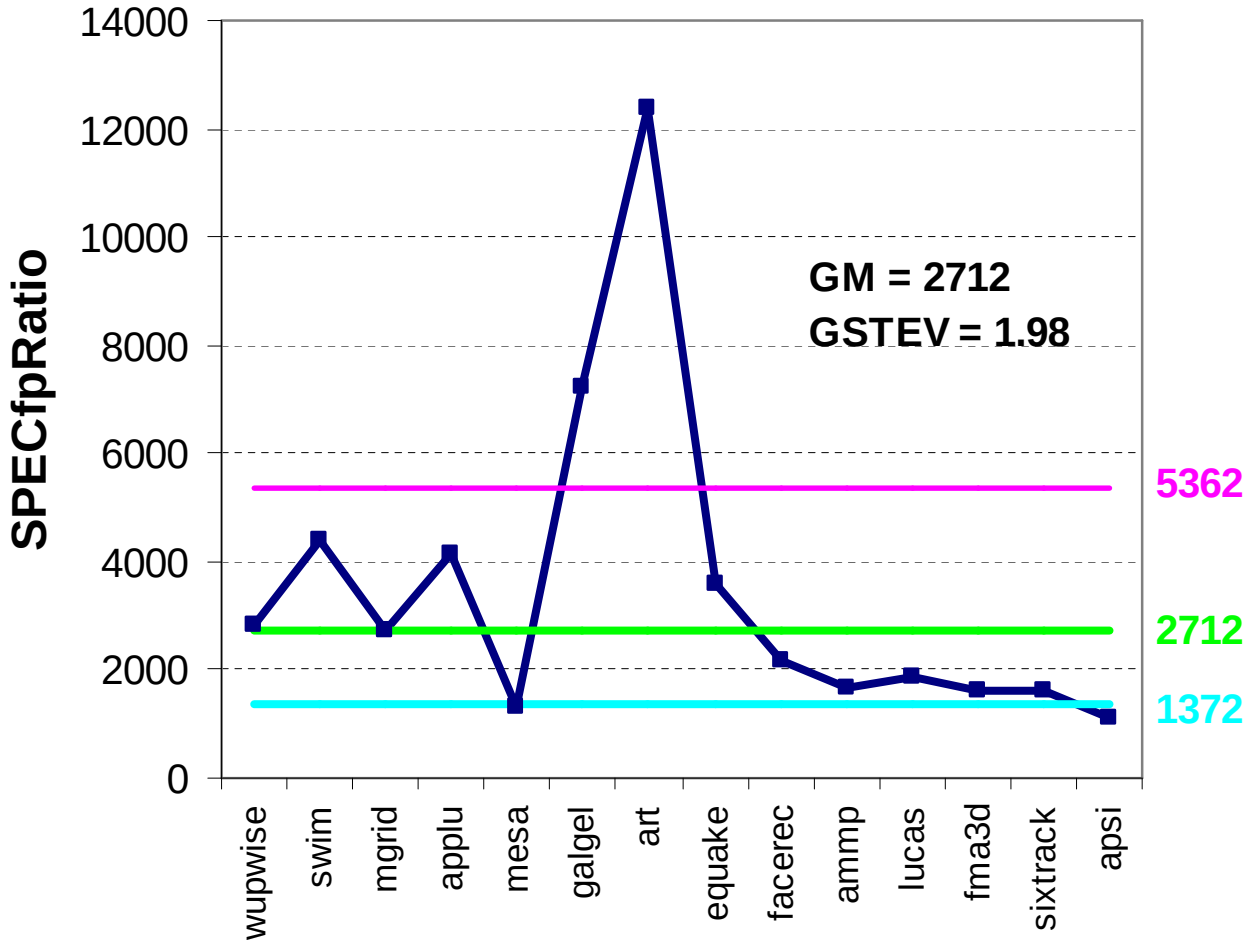
68% of samples fall in range $\left[mean / gstdev, mean \times gstdev\right]$

95% of samples fall in range $\left[mean / gstdev^2, mean \times gstdev^2\right]$

- **Note: Excel provides functions EXP(), LN(), and STDEV() that make calculating geometric mean and multiplicative standard deviation easy**

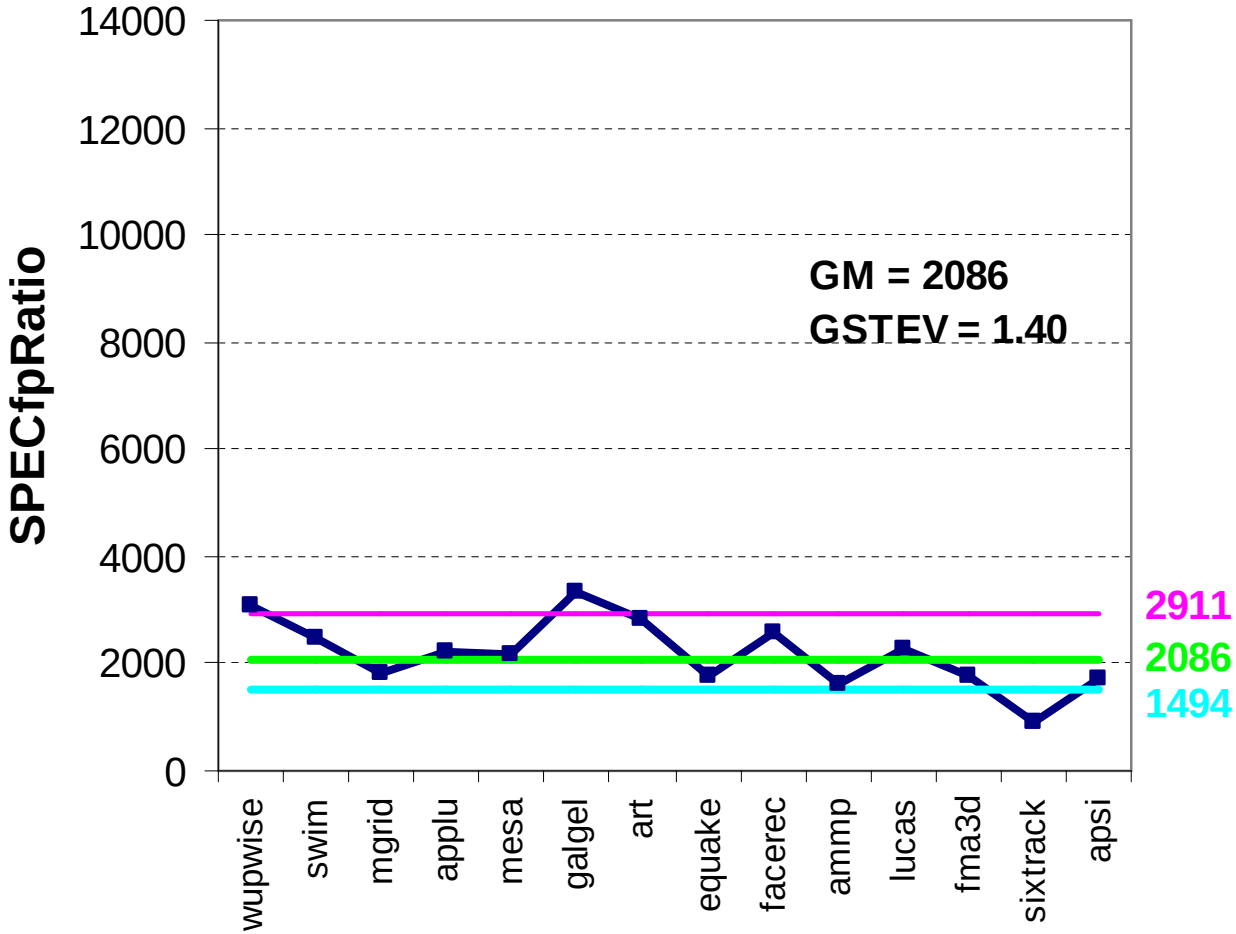
Example Standard Deviation (1/2)

- **GM and multiplicative StDev of SPECfp2000 for Itanium 2**



Example Standard Deviation (2/2)

- **GM and multiplicative StDev of SPECfp2000 for AMD Athlon**



Comments on Itanium 2 and Athlon

- Standard deviation of 1.98 for Itanium 2 is much higher-- vs. 1.40--so results will differ more widely from the mean, and therefore are likely less predictable
- Falling within one standard deviation:
  - 10 of 14 benchmarks (71%) for Itanium 2
  - 11 of 14 benchmarks (78%) for Athlon
- Thus, the results are quite compatible with a lognormal distribution (expect 68%)

Outline

- Computer Science at a Crossroads
- Computer Architecture v. Instruction Set Arch. (1.3)
- Trends in Technology (1.4)
- Trends in Power (1.5)
- Dependability (1.7)
- Measuring, Reporting, and Summarizing Performance (1.8)
- **Quantitative Principles of Design (1.9)**

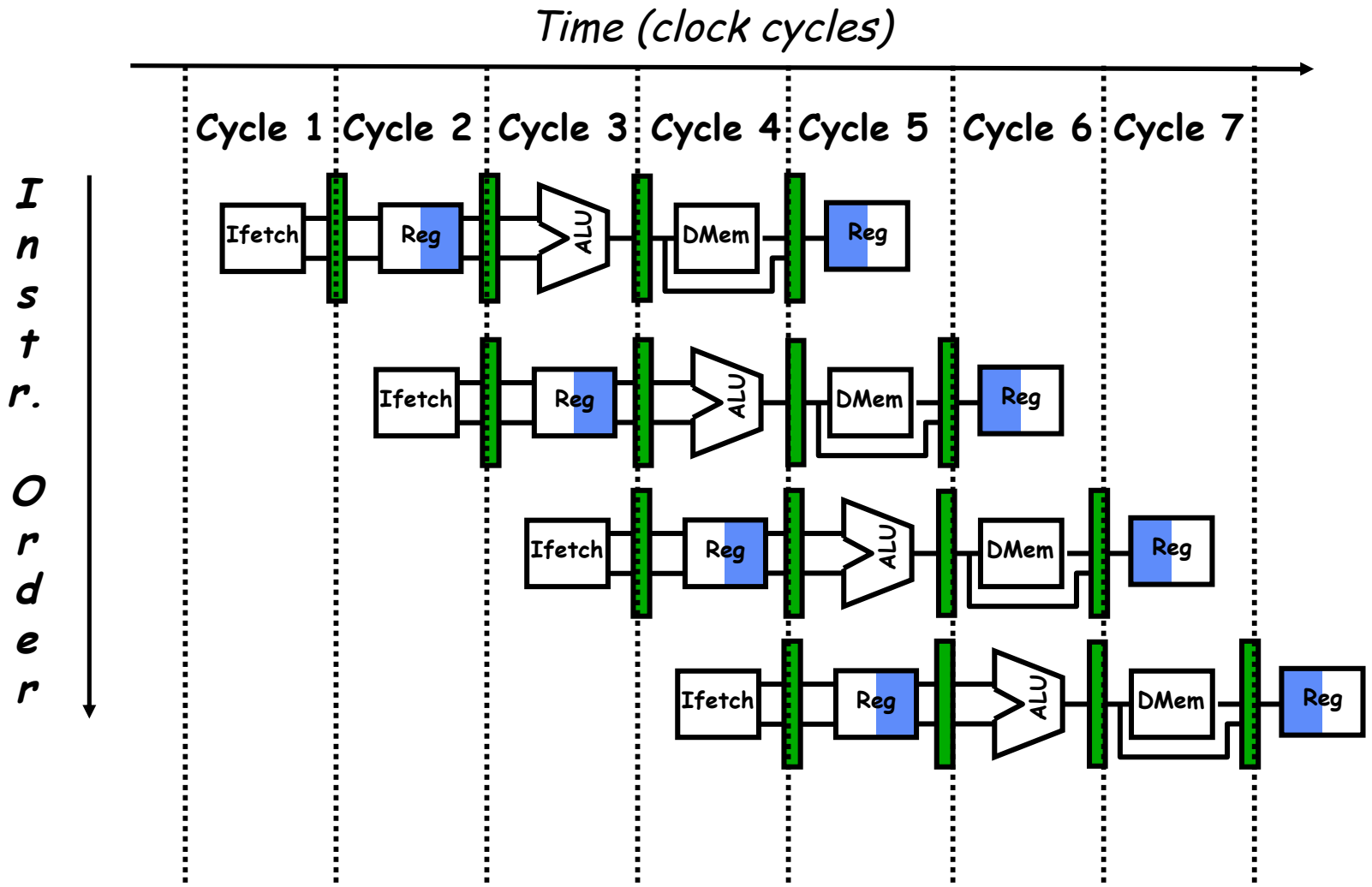
Quantitative Principles of Design

1. Take Advantage of Parallelism
2. Principle of Locality
3. Focus on the Common Case
4. Amdahl's Law
5. The Processor Performance Equation

1) Taking Advantage of Parallelism

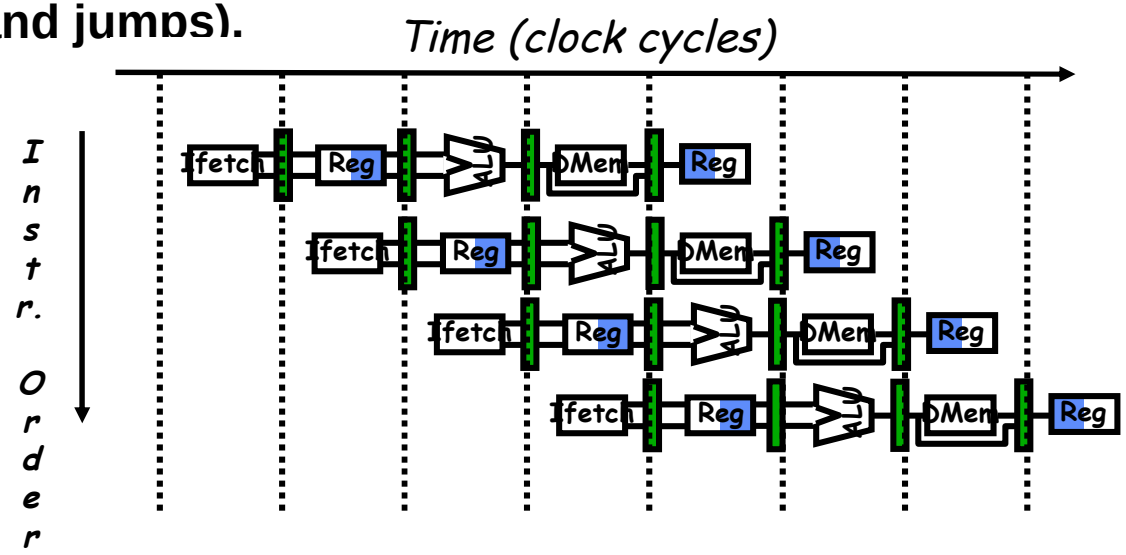
- **Increasing throughput of server computer via multiple processors or multiple disks**
- **Detailed HW design**
  - **Carry lookahead adders uses parallelism to speed up computing sums from linear to logarithmic in number of bits per operand**
  - **Multiple memory banks searched in parallel in set-associative caches**
- **Pipelining: overlap instruction execution to reduce the total time to complete an instruction sequence.**
  - **Not every instruction depends on immediate predecessor ⇒ executing instructions completely/partially in parallel possible**
  - **Classic 5-stage pipeline:**
    1) Instruction Fetch (Ifetch),
    2) Register Read (Reg),
    3) Execute (ALU),
    4) Data Memory Access (Dmem),
    5) Register Write (Reg)

Pipelined Instruction Execution



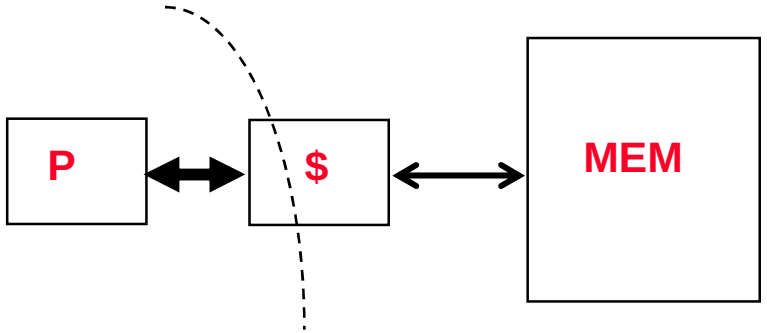
Limits to pipelining

- **Hazards prevent next instruction from executing during its designated clock cycle**
  - **Structural hazards: attempt to use the same hardware to do two different things at once**
  - **Data hazards: Instruction depends on result of prior instruction still in the pipeline**
  - **Control hazards: Caused by delay between the fetching of instructions and decisions about changes in control flow (branches and jumps).**

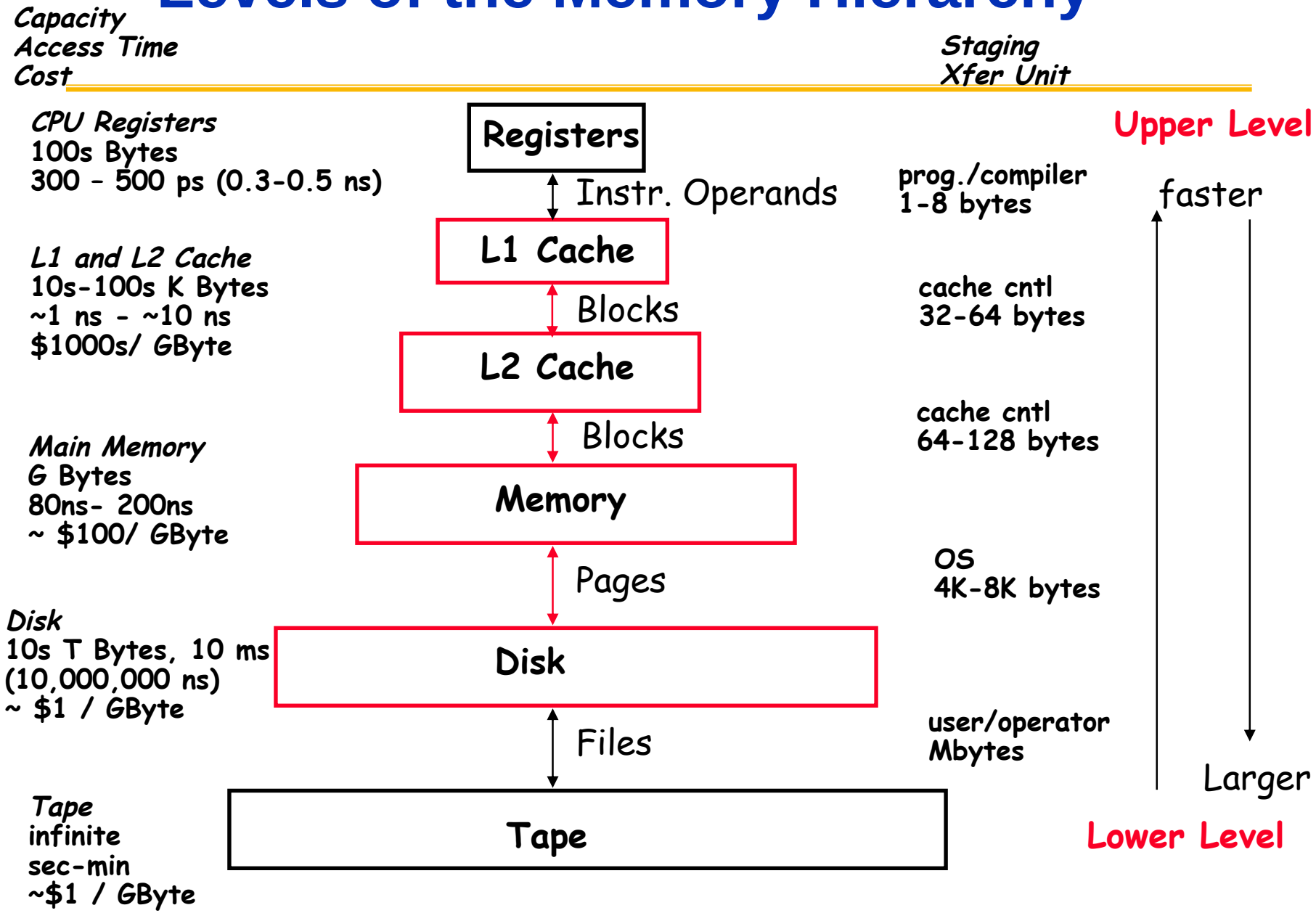


2) The Principle of Locality

- **The Principle of Locality:**
  - **Program access a relatively small portion of the address space at any instant of time.**
- **Two Different Types of Locality:**
  - **Temporal Locality (Locality in Time): If an item is referenced, it will tend to be referenced again soon (e.g., loops, reuse)**
  - **Spatial Locality (Locality in Space): If an item is referenced, items whose addresses are close by tend to be referenced soon (e.g., straight-line code, array access)**
- **Last 30 years, HW relied on locality for memory perf.**



Levels of the Memory Hierarchy



3) Focus on the Common Case

- Common sense guides computer design
  - Since its engineering, common sense is valuable
- In making a design trade-off, favor the frequent case over the infrequent case
  - E.g., Instruction fetch and decode unit used more frequently than multiplier, so optimize it 1st
  - E.g., If database server has 50 disks / processor, storage dependability dominates system dependability, so optimize it 1st
- Frequent case is often simpler and can be done faster than the infrequent case
  - E.g., overflow is rare when adding 2 numbers, so improve performance by optimizing more common case of no overflow
  - May slow down overflow, but overall performance improved by optimizing for the normal case
- What is frequent case and how much performance improved by making case faster => Amdahl's Law

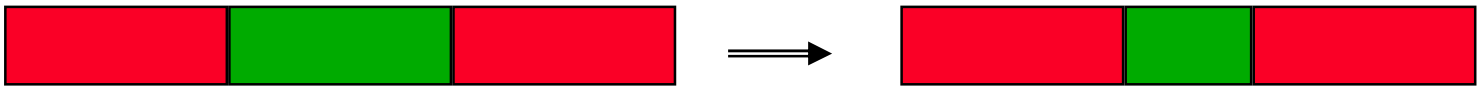
4) Amdahl's Law

$$ExTime_{new} = ExTime_{old} \times \left[\left(1 - Fraction_{enhanced}\right) + \frac{Fraction_{enhanced}}{Speedup_{enhanced}} \right]$$

$$Speedup_{overall} = \frac{ExTime_{old}}{ExTime_{new}} = \frac{1}{\left(1 - Fraction_{enhanced}\right) + \frac{Fraction_{enhanced}}{Speedup_{enhanced}}}$$

Best you could ever hope to do:

$$Speedup_{maximum} = \frac{1}{\left(1 - Fraction_{enhanced}\right)}$$



Amdahl's Law example

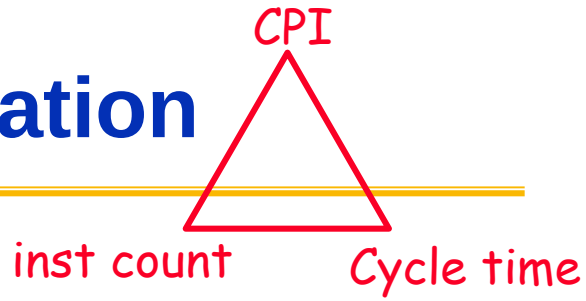
- **New CPU 10X faster**
- **I/O bound server, so 60% time waiting for I/O**

$$\text{Speedup}_{\text{overall}} = \frac{1}{\left(1 - \text{Fraction}_{\text{enhanced}}\right) + \frac{\text{Fraction}_{\text{enhanced}}}{\text{Speedup}_{\text{enhanced}}}}$$

$$= \frac{1}{\left(1 - 0.4\right) + \frac{0.4}{10}} = \frac{1}{0.64} = 1.56$$

- **Apparently, its human nature to be attracted by 10X faster, vs. keeping in perspective its just 1.6X faster**

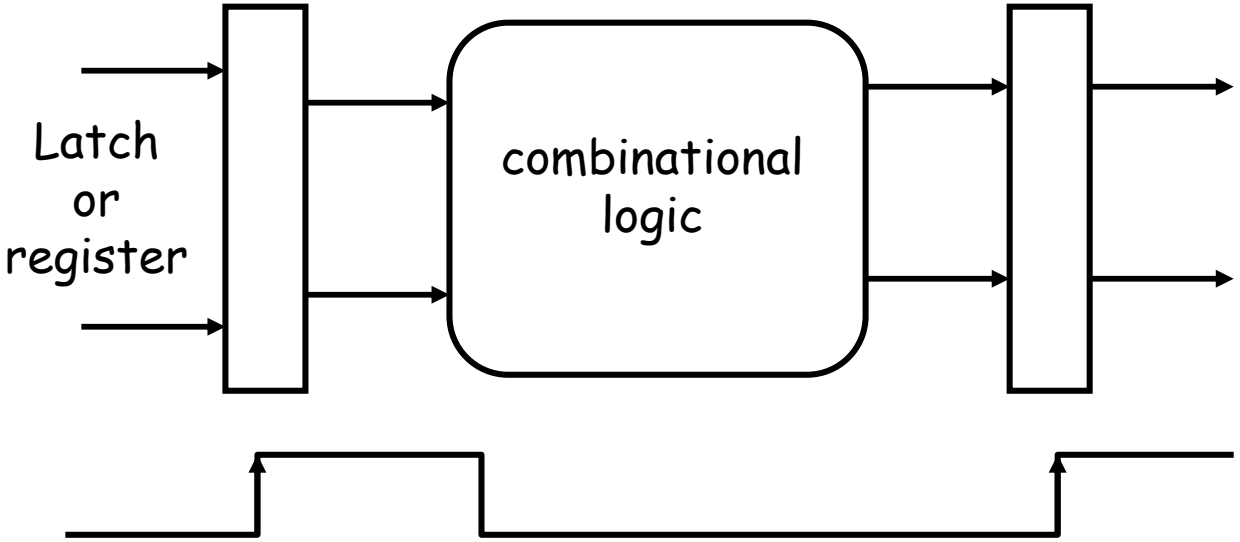
5) Processor performance equation



$$\text{CPU time} = \frac{\text{Seconds}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Cycle}}$$

	Inst Count	CPI	Clock Rate
Program	X		
Compiler	X	(X)	
Inst. Set.	X	X	
Organization		X	X
Technology			X

What's a Clock Cycle?



Summary

- **Computer Architecture >> instruction sets**
- **Computer Architecture skill sets are different**
  - **5 Quantitative principles of design**