Computer Architecture
- Introduction

Chin-Fu Kuo

About This Course

- Textbook
  - J. L. Hennessy and D. A. Patterson, Computer Architecture: A Quantitative Approach, 3rd Edition, Morgan Kaufmann Publishing Co., 2002.
- Course Grading
  - 30% Project and Quiz
  - 35% Mid-term Examination
  - 35% Final-term Examination
  - 5~10% Class Participation & Discussion

This Is an Advanced Course

- Have you taken “Computer Organization” before?

- If you never took “Computer Organization” before
  - You MUST take it if you are an undergraduate student;
  - You may still take this course if you insist, but be prepared to work hard and read some chapters in “Computer Organization and Design (COD)3/e”

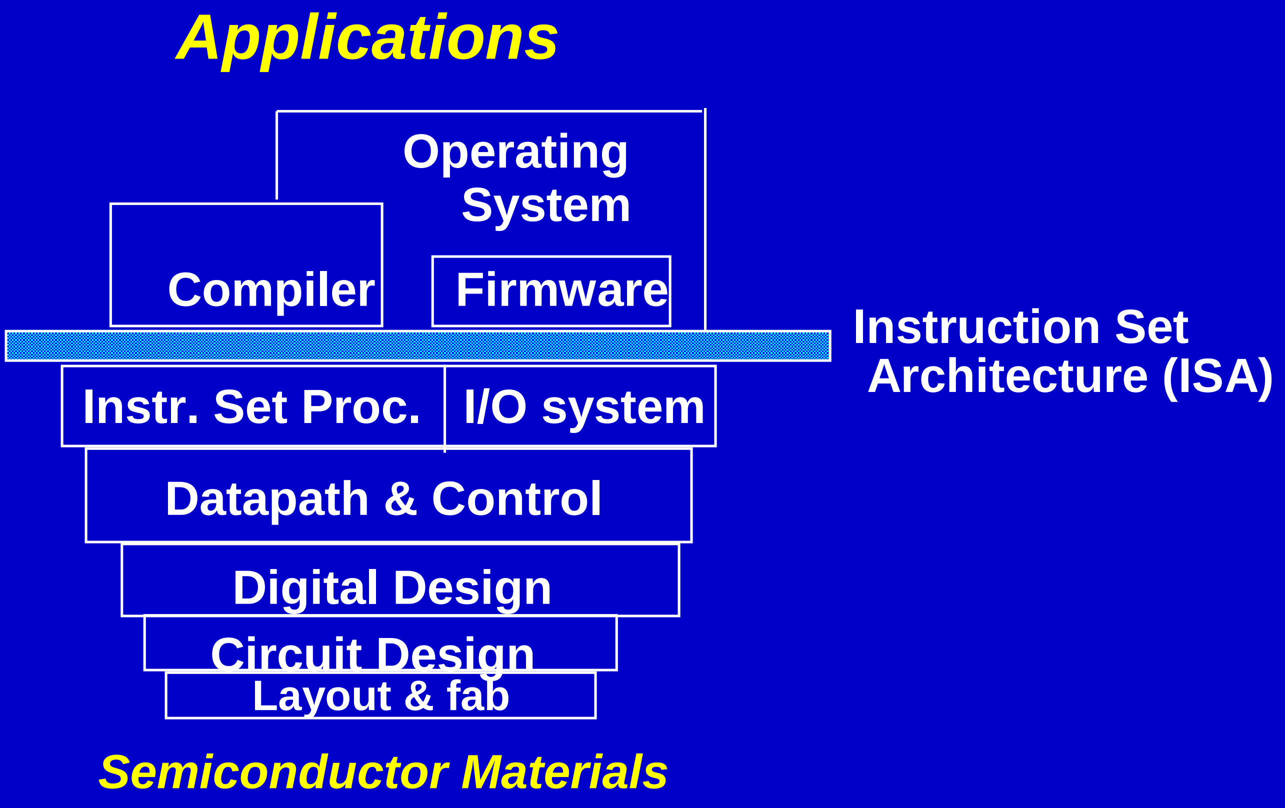
Reference Resources

- Patterson, UC-Berkeley Spring 2001
  http://www.cs.berkeley.edu/~pattrsn/252S01/
- David E. Culler, UC-Berkeley, Spring 2002
  http://www.cs.berkeley.edu/~culler/cs252-s02/
- David E. Culler, UC-Berkeley, Spring 2005
  http://www.cs.berkeley.edu/~culler/courses/cs252-s05/
- Many slides in this course were adapted from UC Berkeley’s CS252 Course. Copyright 2005, UC Berkeley.

Outline

- What is Computer Architecture?
  - Fundamental Abstractions & Concepts
- Instruction Set Architecture & Organization
- Why Take This Course?
- Technology
- Performance
- Computer Architecture Renaissance

What is “Computer Architecture”?

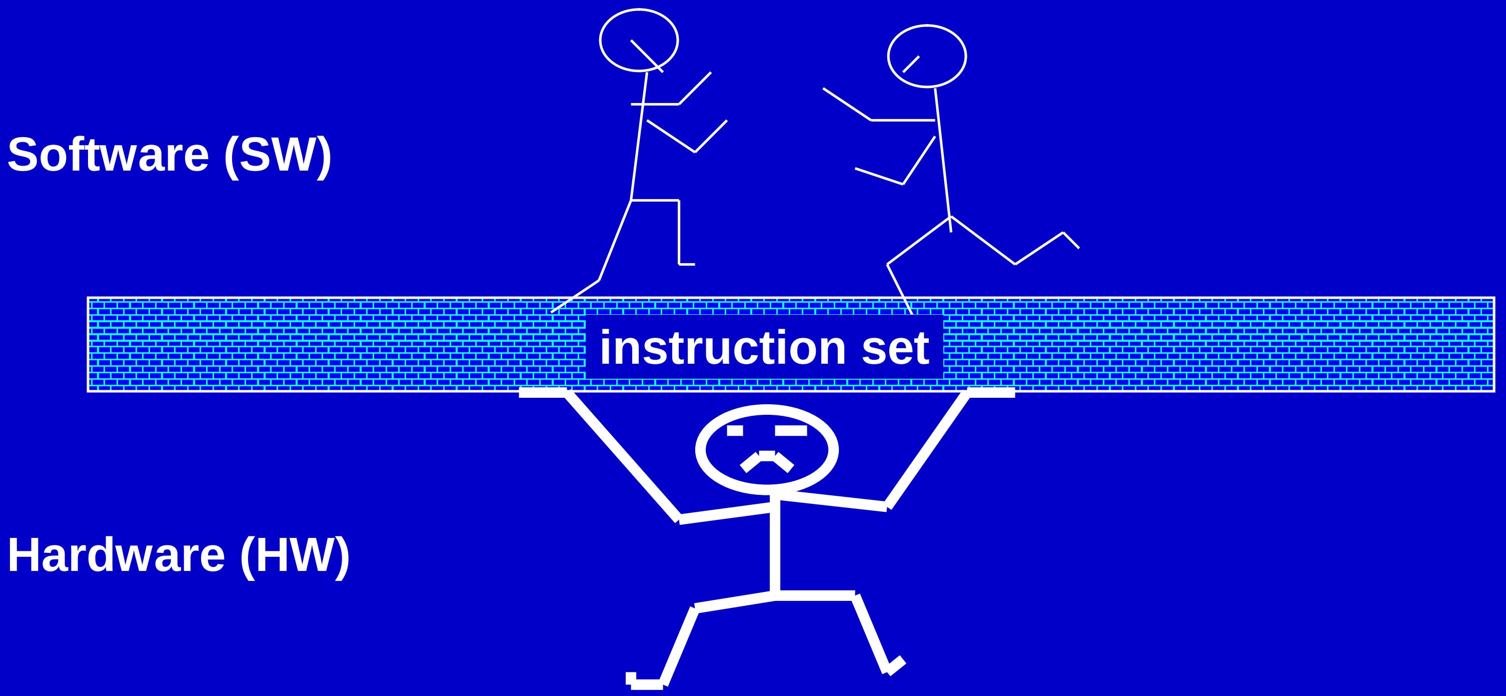


- Coordination of many *levels of abstraction*
- Under a rapidly changing set of forces
- Design, Measurement, *and* Evaluation

Outline

- What is Computer Architecture?
  - Fundamental Abstractions & Concepts
- Instruction Set Architecture & Organization
- Why Take This Course?
- Technology
- Performance
- Computer Architecture Renaissance

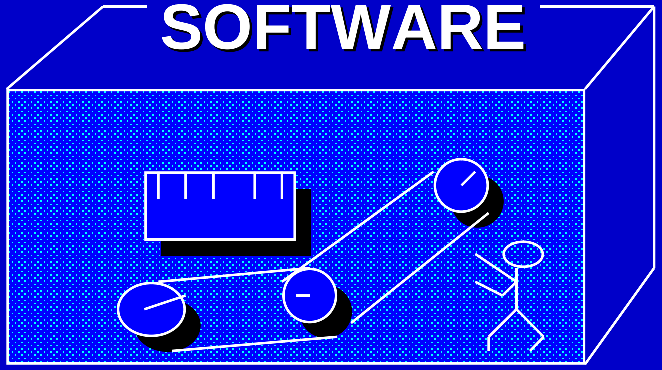
The Instruction Set: a Critical Interface



- 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

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, Blaaw, 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**

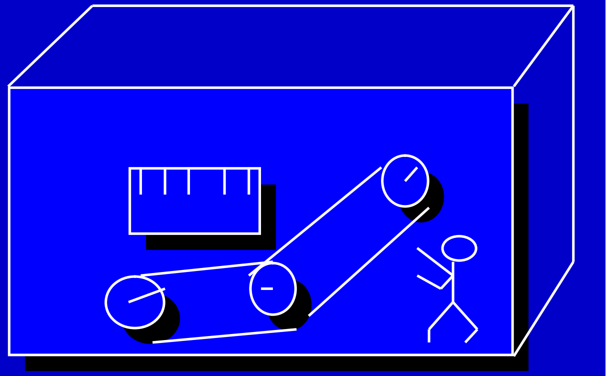
Computer (Machine) Organization

- 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)**

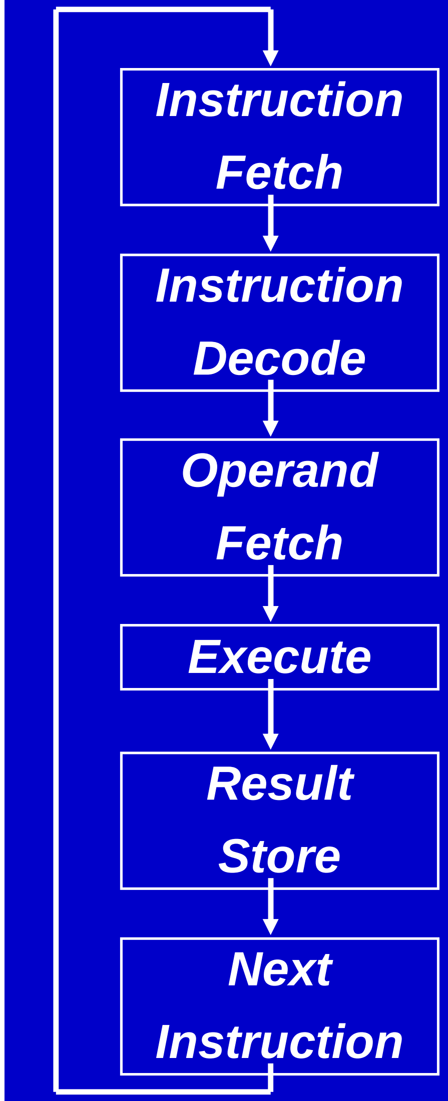
Logic Designer's View

ISA Level

FUs & Interconnect



Fundamental Execution Cycle



Instruction Fetch

Obtain instruction from program storage

Instruction Decode

Determine required actions and instruction size

Operand Fetch

Locate and obtain operand data

Execute

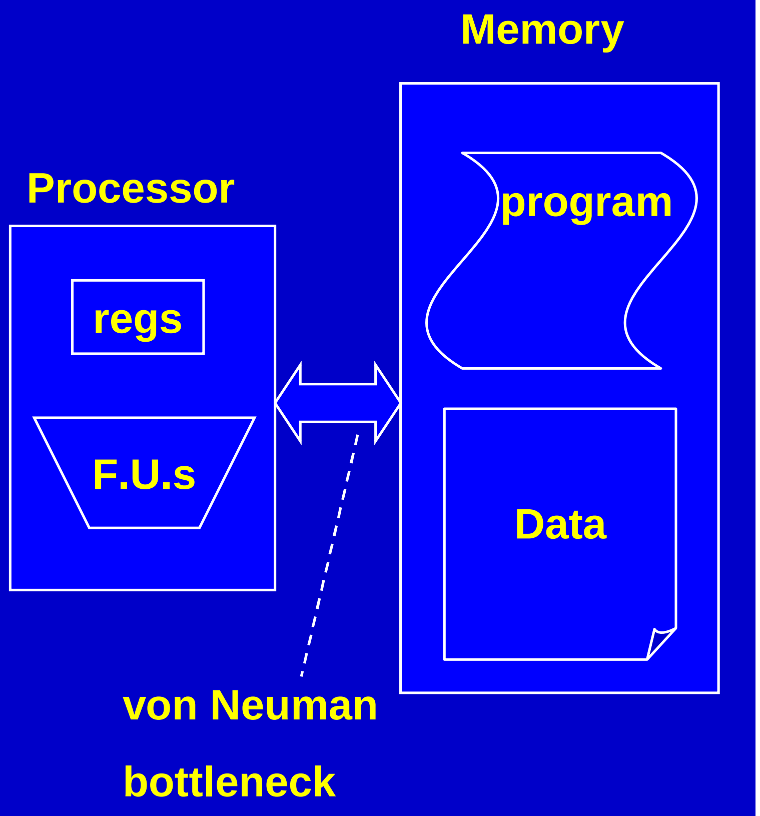
Compute result value or status

Result Store

Deposit results in storage for later use

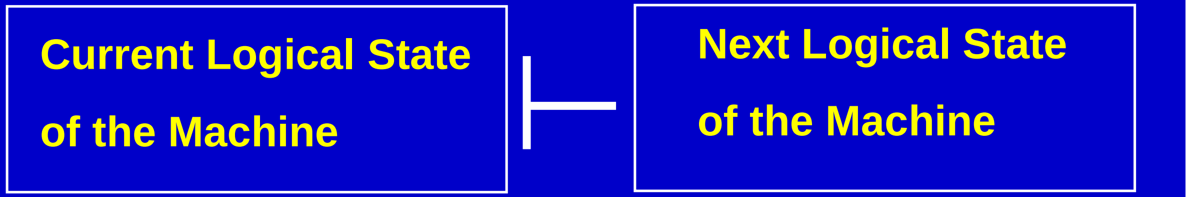
Next Instruction

Determine successor instruction



Elements of an ISA

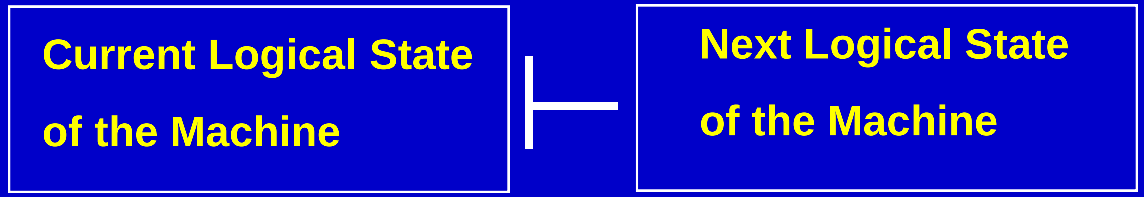
- Set of machine-recognized data types
  - **bytes, words, integers, floating point, strings, . . .**
- Operations performed on those data types
  - **Add, sub, mul, div, xor, move, ….**
- Programmable storage
  - **regs, PC, memory**
- Methods of identifying and obtaining data referenced by instructions (addressing modes)
  - **Literal, reg., absolute, relative, reg + offset, …**
- Format (encoding) of the instructions
  - **Op code, operand fields, …**



Current Logical State of the Machine ⊢ **Next Logical State of the Machine**

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**

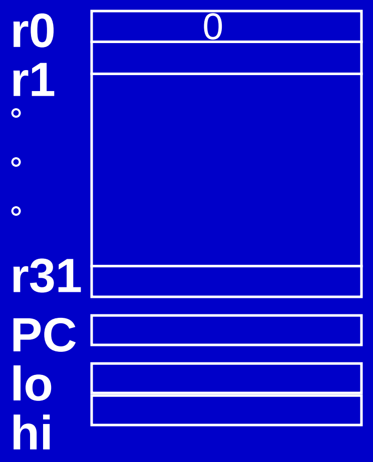


Current Logical State of the Machine ⊢ **Next Logical State of the Machine**

Time for a Long Break and Please …

- Partner w/ a classmate who you didn't know
- Get the following information from your partner:
  - **Personal Information & Interests:**
    Name, Department, Hometown, Favorite sports, …
  - **Research Directions:**
    Research Lab, Advisor, Projects, …
  - **Career Plan:**
    Engineer, Manager, Teacher, …
  - **Why take this course**
- Introduce your partner to the class after the break.

Example: MIPS R3000



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

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:

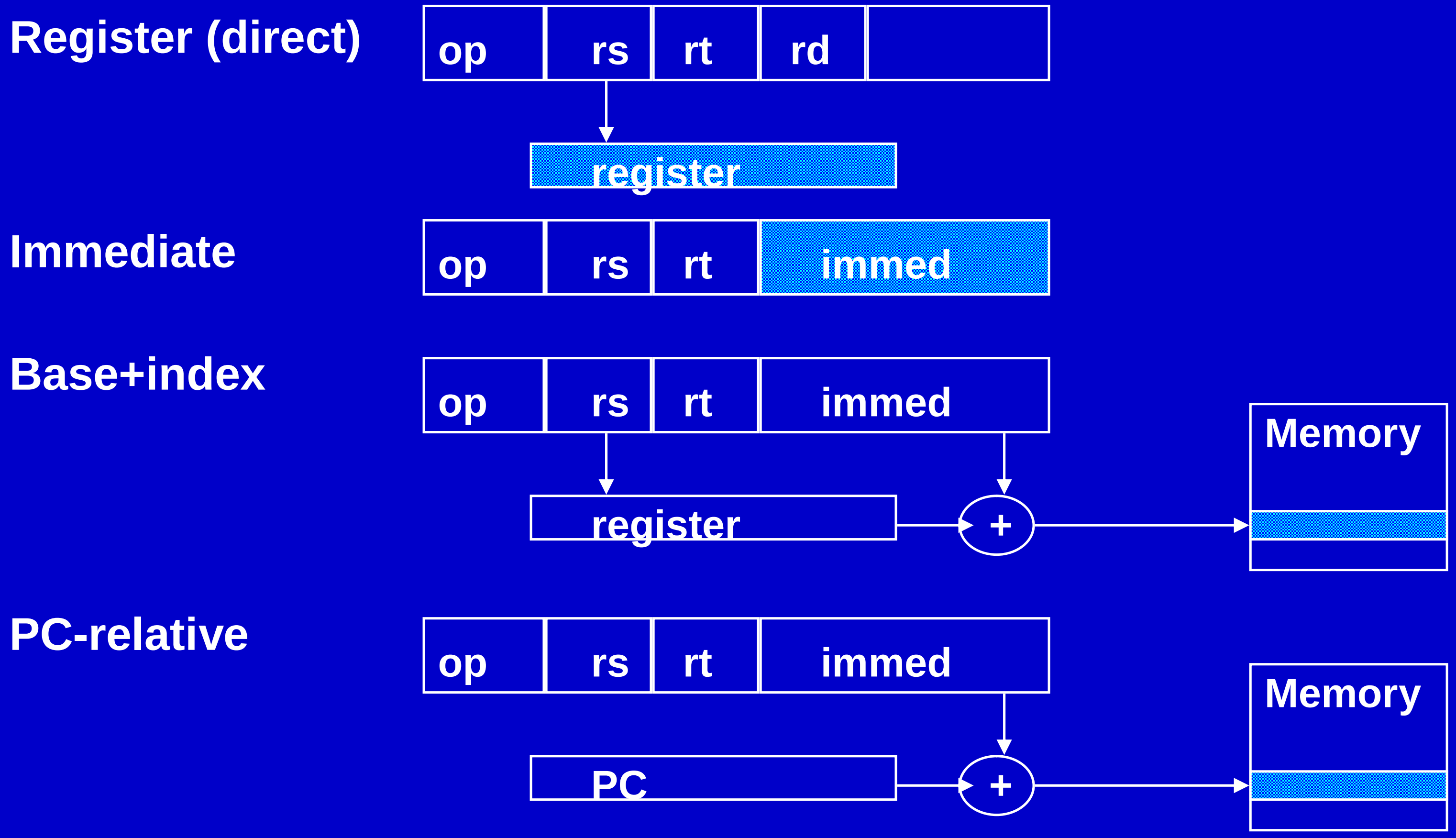
2 address	add A B	EA(A) ← EA(A) + EA(B)
3 address	add A B C	EA(A) ← EA(B) + EA(C)

Load/Store:

3 address	add Ra Rb Rc	Ra ← Rb + Rc
	load Ra Rb	Ra ← mem[Rb]
	store Ra Rb	mem[Rb] ← Ra

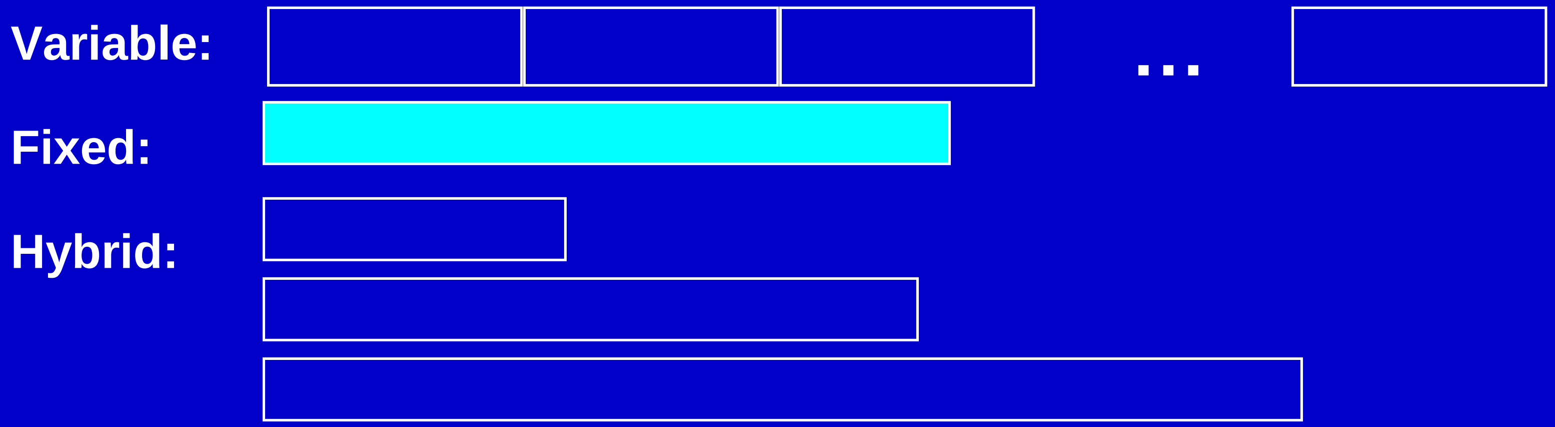
MIPS Addressing Modes & Formats

- Simple addressing modes
- All instructions 32 bits wide



- Register Indirect?

Instruction Formats & RISC



Variable: …

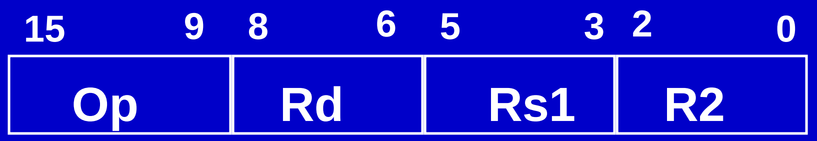
Fixed:

Hybrid:

- **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

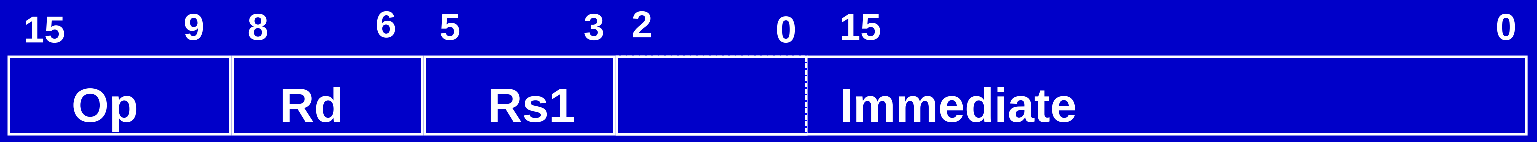
Register-Register



15 9 8 6 5 3 2 0

Op Rd Rs1 R2

Load, Store and Branch

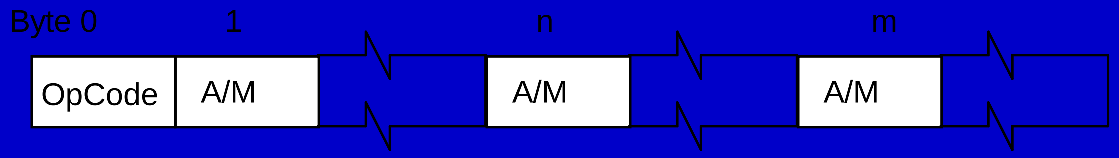


15 9 8 6 5 3 2 0 15 0

Op Rd Rs1 Immediate

VAX-11: the Canonical CISC

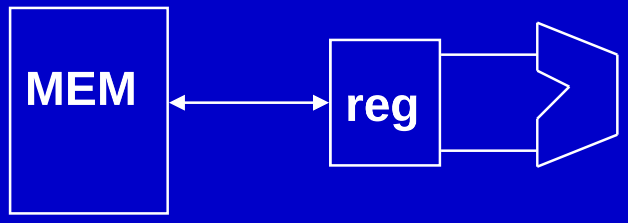
Variable format, 2 and 3 address instruction



- 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

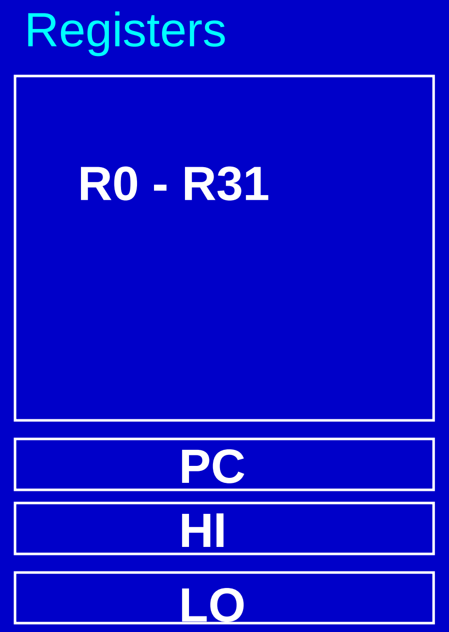
  condition code + branch on condition
- Simple fixed-format encoding

op	**r**	**r**	**r**	▨
op	**r**	**r**	**immed**	
op	**offset**			

- **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


Registers

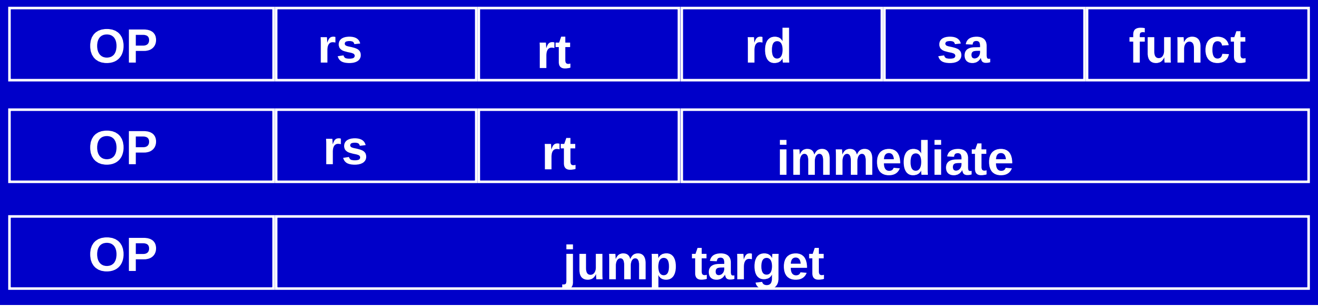
R0 - R31

PC

HI

LO

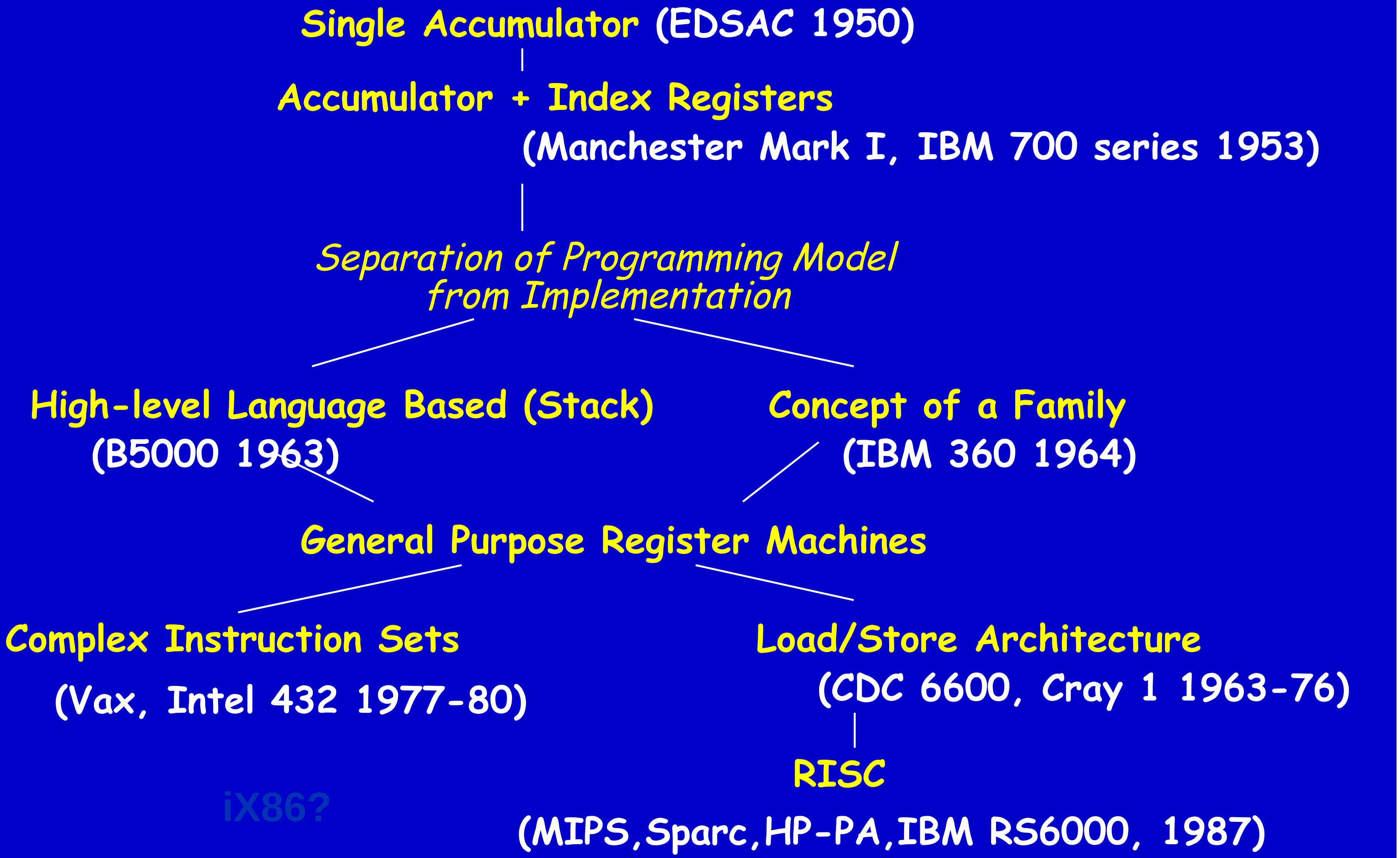
3 Instruction Formats: all 32 bits wide


OP | rs | rt | rd | sa | funct

OP | rs | rt | immediate

OP | jump target

Evolution of Instruction Sets



Single Accumulator (EDSAC 1950)

Accumulator + Index Registers

(Manchester Mark I, IBM 700 series 1953)

Separation of Programming Model
from Implementation

High-level Language Based (Stack)

(B5000 1963)

Concept of a Family

(IBM 360 1964)

General Purpose Register Machines

Complex Instruction Sets

(Vax, Intel 432 1977-80)

iX86?

Load/Store Architecture

(CDC 6600, Cray 1 1963-76)

RISC

(MIPS,Sparc,HP-PA,IBM RS6000, 1987)

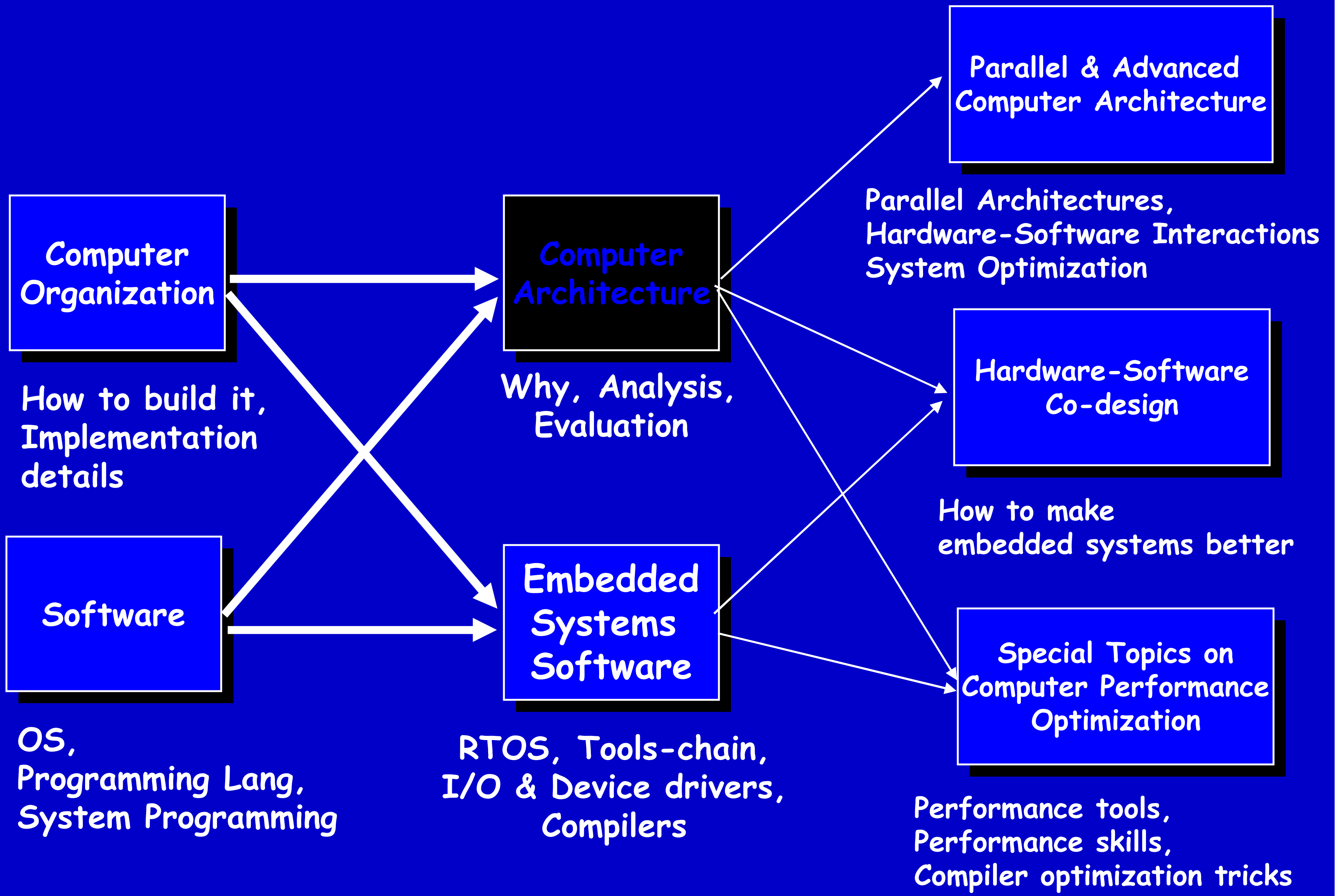
Outline

- What is Computer Architecture?
  - Fundamental Abstractions & Concepts
- Instruction Set Architecture & Organization
- Why Take This Course?
- Technology
- Performance
- Computer Architecture Renaissance

Why Take This Course?

- To design the next great instruction set?...well...
  - instruction set architecture has largely converged
  - especially in the desktop / server / laptop space
  - dictated by powerful market forces
- Tremendous organizational innovation relative to established ISA abstractions
- Many New instruction sets or equivalent
  - embedded space, controllers, specialized devices, ...
- Design, analysis, implementation concepts vital to all aspects of EE & CS
  - systems, PL, theory, circuit design, VLSI, comm.
- Equip you with an intellectual toolbox for dealing with a host of systems design challenges

Related Courses



Computer Organization

How to build it, Implementation details

Software

OS, Programming Lang, System Programming

Computer Architecture

Why, Analysis, Evaluation

Embedded Systems Software

RTOS, Tools-chain, I/O & Device drivers, Compilers

Parallel & Advanced Computer Architecture

Parallel Architectures, Hardware-Software Interactions System Optimization

Hardware-Software Co-design

How to make embedded systems better

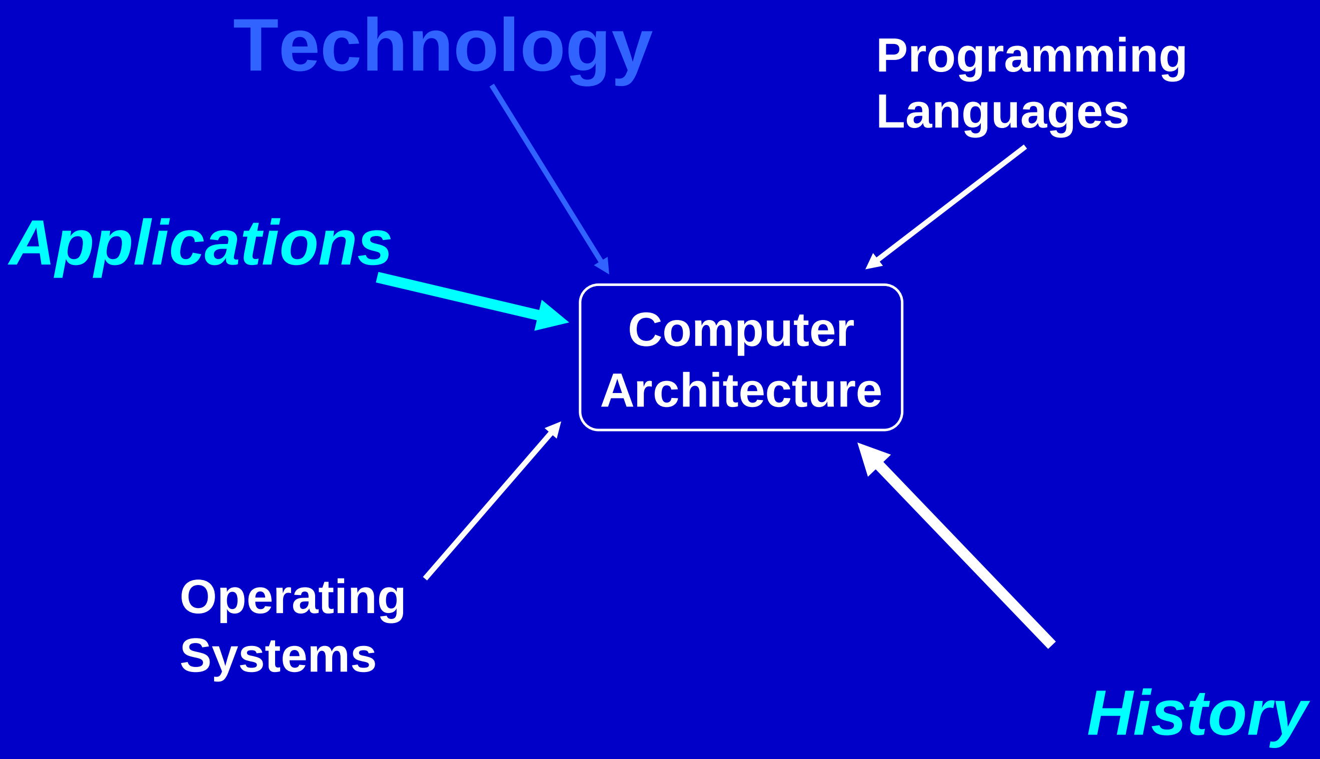
Special Topics on Computer Performance Optimization

Performance tools, Performance skills, Compiler optimization tricks

Computer Industry

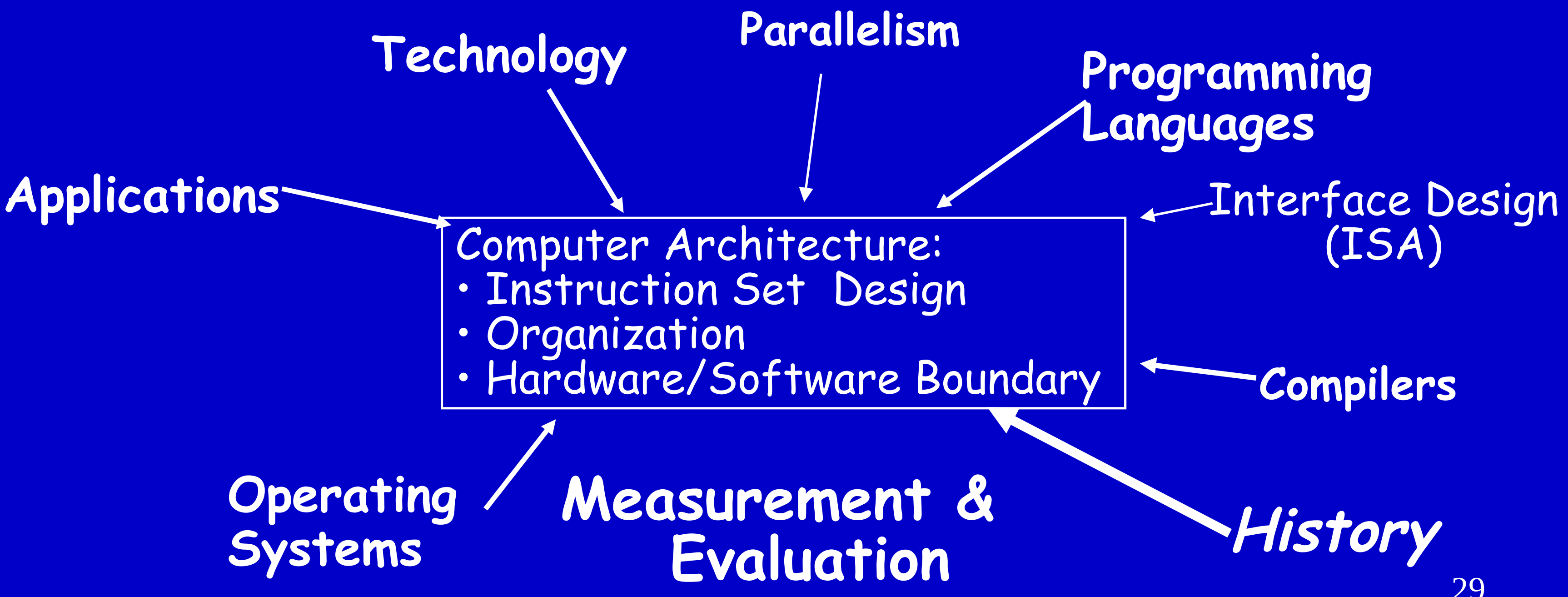
- Desktop Computing
  - Price-performance, Graphics performance
  - Intel, AMD, Apple, Microsoft, Linux
  - System integrators & Retailers
- Servers
  - Availability, Scalability, Throughput
  - IBM, HP-Compaq, Sun, Intel, Microsoft, Linux
- Embedded Systems
  - Application-specific performance
  - Power, Integration

Forces on Computer Architecture



Course Focus

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



Technology

Parallelism

Programming Languages

Applications

Interface Design (ISA)

Computer Architecture:
- Instruction Set Design
- Organization
- Hardware/Software Boundary

Compilers

Operating Systems

Measurement & Evaluation

History

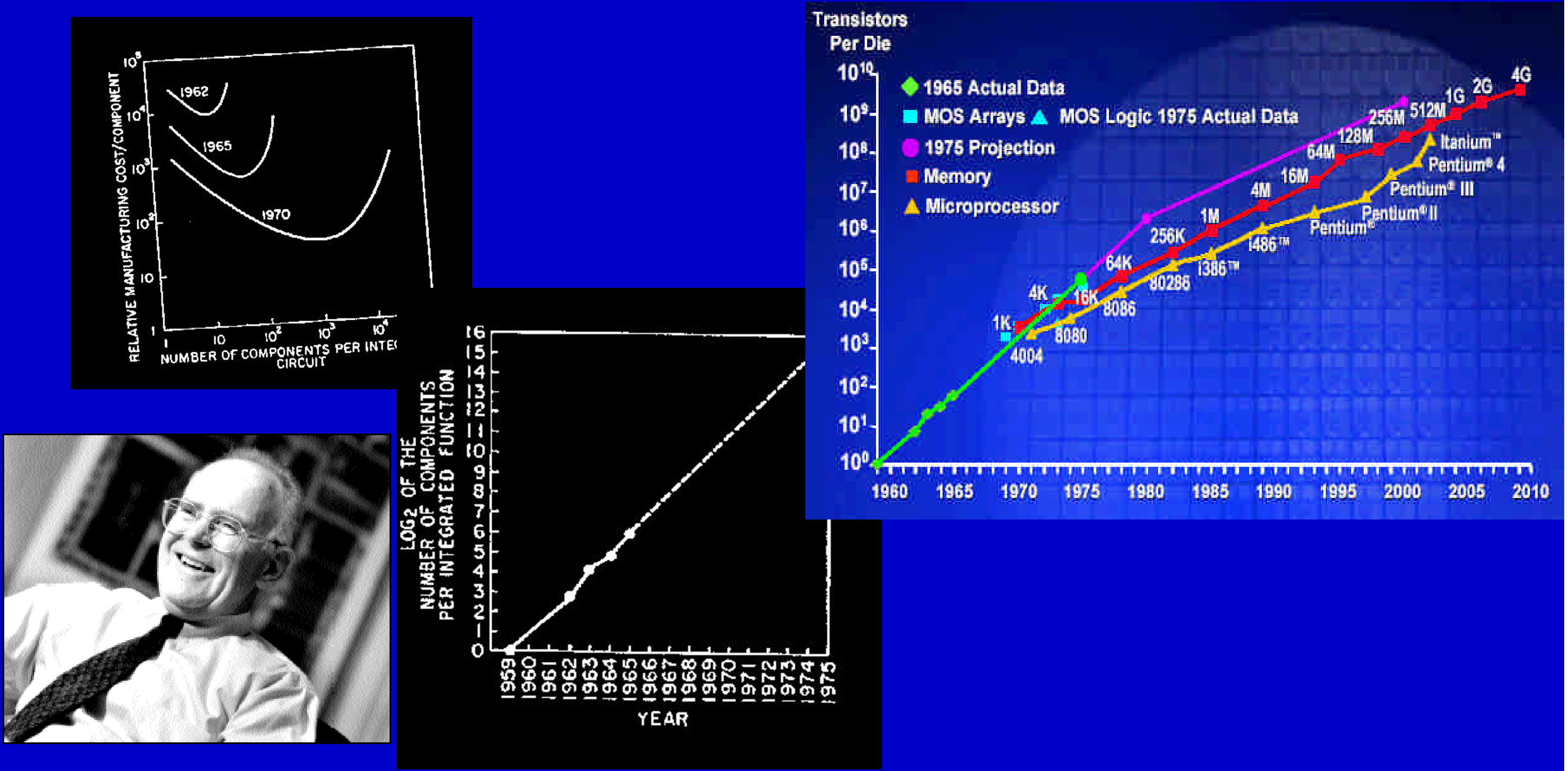
Outline

- What is Computer Architecture?
  - Fundamental Abstractions & Concepts
- Instruction Set Architecture & Organization
- Why Take This Course?
- Technology Trend
- Performance
- Computer Architecture Renaissance

Dramatic Technology Advance

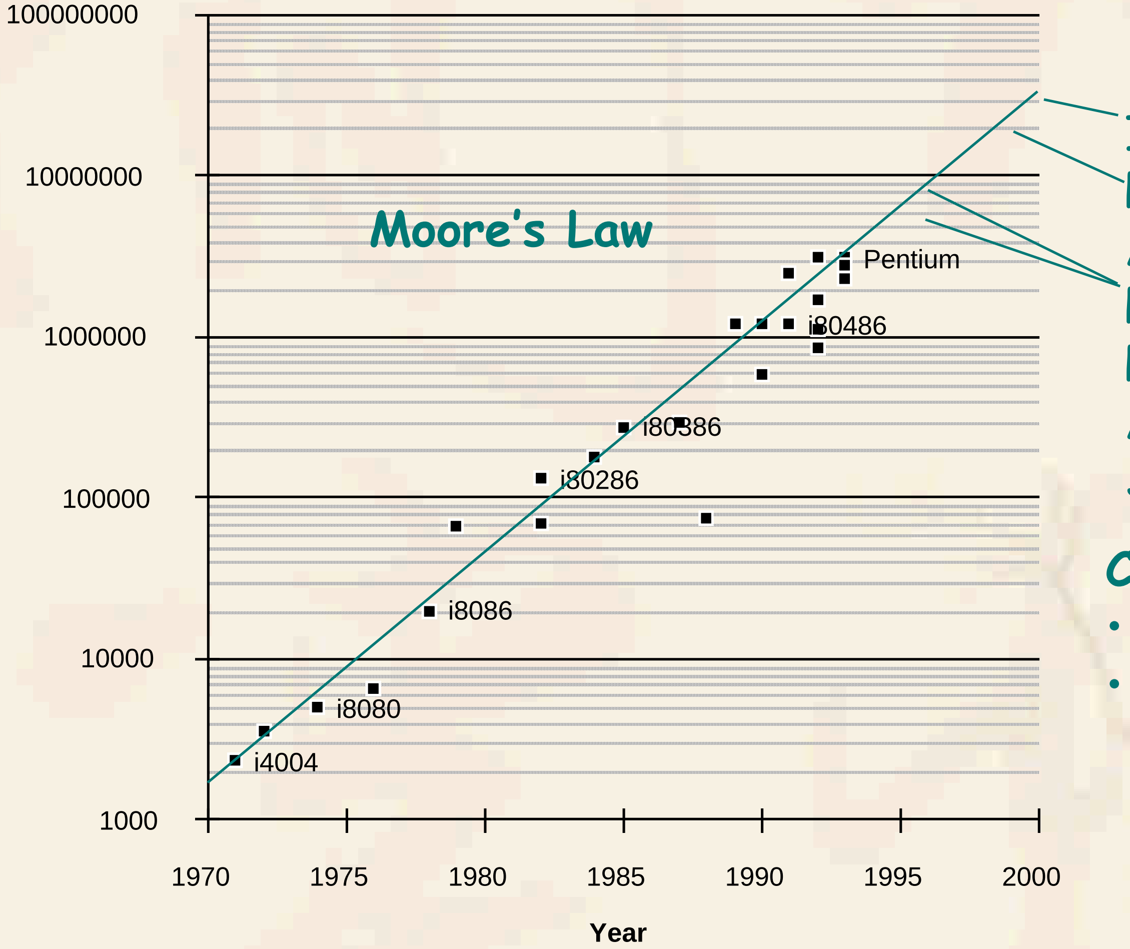
- Prehistory: Generations
  - 1st Tubes
  - 2nd Transistors
  - 3rd Integrated Circuits
  - 4th VLSI….
- Discrete advances in each generation
  - Faster, smaller, more reliable, easier to utilize
- Modern computing: Moore’s Law
  - Continuous advance, fairly homogeneous technology

Moore’s Law



- ***“Cramming More Components onto Integrated Circuits”***
  - ***Gordon Moore, Electronics, 1965***
- ***# on transistors on cost-effective integrated circuit double every 18 months (IC上可容納的電晶體數目，約每隔18個月便會增加一倍，性能也將提升一倍。）***

Technology Trends: Microprocessor Capacity

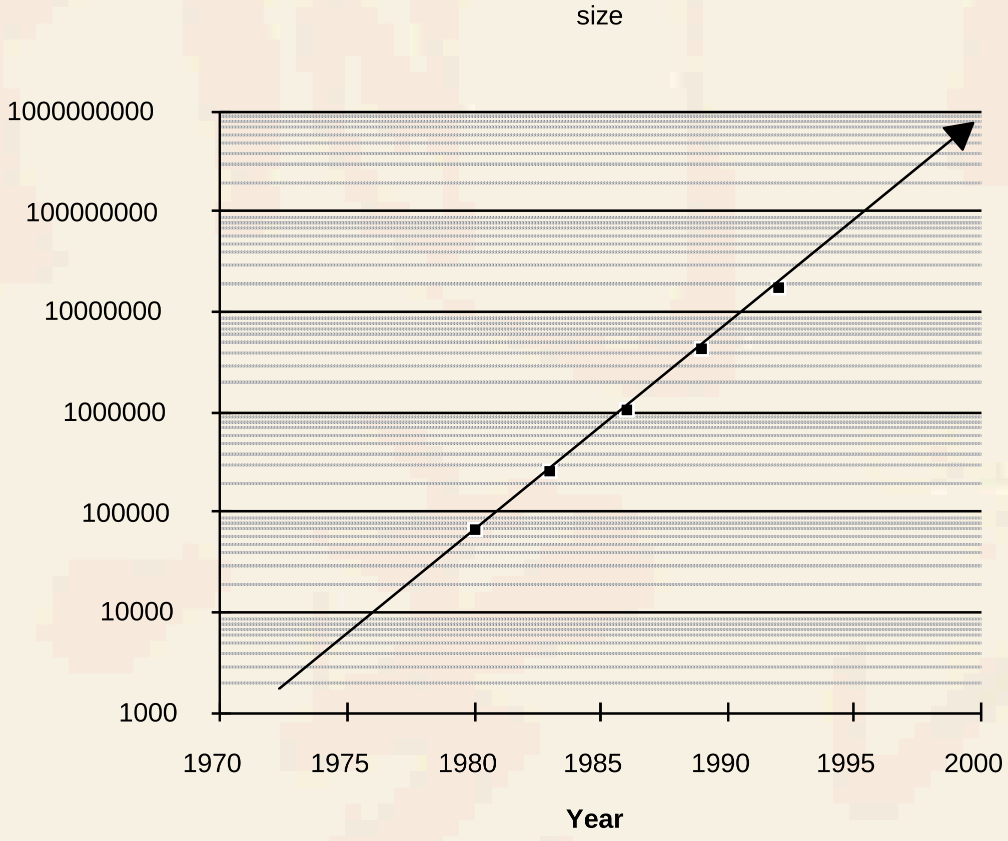


Itanium II: 241 million
Pentium 4: 55 million
Alpha 21264: 15 million
Pentium Pro: 5.5 million
PowerPC 620: 6.9 million
Alpha 21164: 9.3 million
Sparc Ultra: 5.2 million

CMOS improvements:

- **Die size: 2X every 3 yrs**
- **Line width: halve / 7 yrs**

Memory Capacity (Single Chip DRAM)



year	size(Mb)	cyc time
1980	0.0625	250 ns
1983	0.25	220 ns
1986	1	190 ns
1989	4	165 ns
1992	16	145 ns
1996	64	120 ns
2000	256	100 ns
2003	1024	60 ns

Optimizing the Design

- Functional requirements set by:
  - market sector
  - particular company's product plan
  - what the competition is expected to do
- Usual pressure to do everything
  - minimize time to market
  - maximize performance
  - minimize cost & power
- And you only get 1 shot
  - no time to try multiple prototypes and evolve to a polished product
  - requires heaps of simulations to quantify everything
    - quantify model is focus of this course
  - requires deep infrastructure and support

Technology Trends

- Integrated Circuits
  - density increases at 35%/yr.
  - die size increases 10%-20%/yr
  - combination is a chip complexity growth rate of 55%/yr
  - transistor speed increase is similar but signal propagation doesn’t track this curve - so clock rates don’t go up as fast
- DRAM
  - density quadruples every 3-4 years (40 - 60%/yr) [4x steps]
  - cycle time decreases slowly - 33% in 10 years
  - interface changes have improved bandwidth however
- Network
  - rapid escalation - US bandwidth doubles every year at the machine the expectation bumps periodically - gigabit ether is here now

3 Categories Emerge

- Desktop
  - optimized for price-performance (frequency is a red herring)
- Server
  - optimized for: availability, scalability, and throughput
  - plus a new one: power ==> cost and physical plant site
- Embedded
  - fastest growing and the most diverse space
    - washing machine controller to a network core router
  - optimizations: cost, real-time, specialized performance, power
    - minimize memory and logic for the task at hand

Outline

- What is Computer Architecture?
  - Fundamental Abstractions & Concepts
- Instruction Set Architecture & Organization
- Why Take This Course?
- Technology
- Cost and Price
- Performance
- Computer Architecture Renaissance

Cost

- Clearly a market place issue
  - time, volume, commoditization play a big role
  - WCT (whole cost transfer) also a function of volume
    - CS is all about the cheap copy
- However it's not that simple – what kind of cost
  - cost to buy – this is really price
  - cost to maintain
  - cost to upgrade – never known at purchase time
  - cost to learn to use – Apple won this one for awhile
  - cost of ISV (indep. SW vendor) software
  - cost to change platforms – the vendor lock
  - cost of a failure – pandora's box opens …
- Let's focus on hardware costs
  - it's simpler

Cost Impact

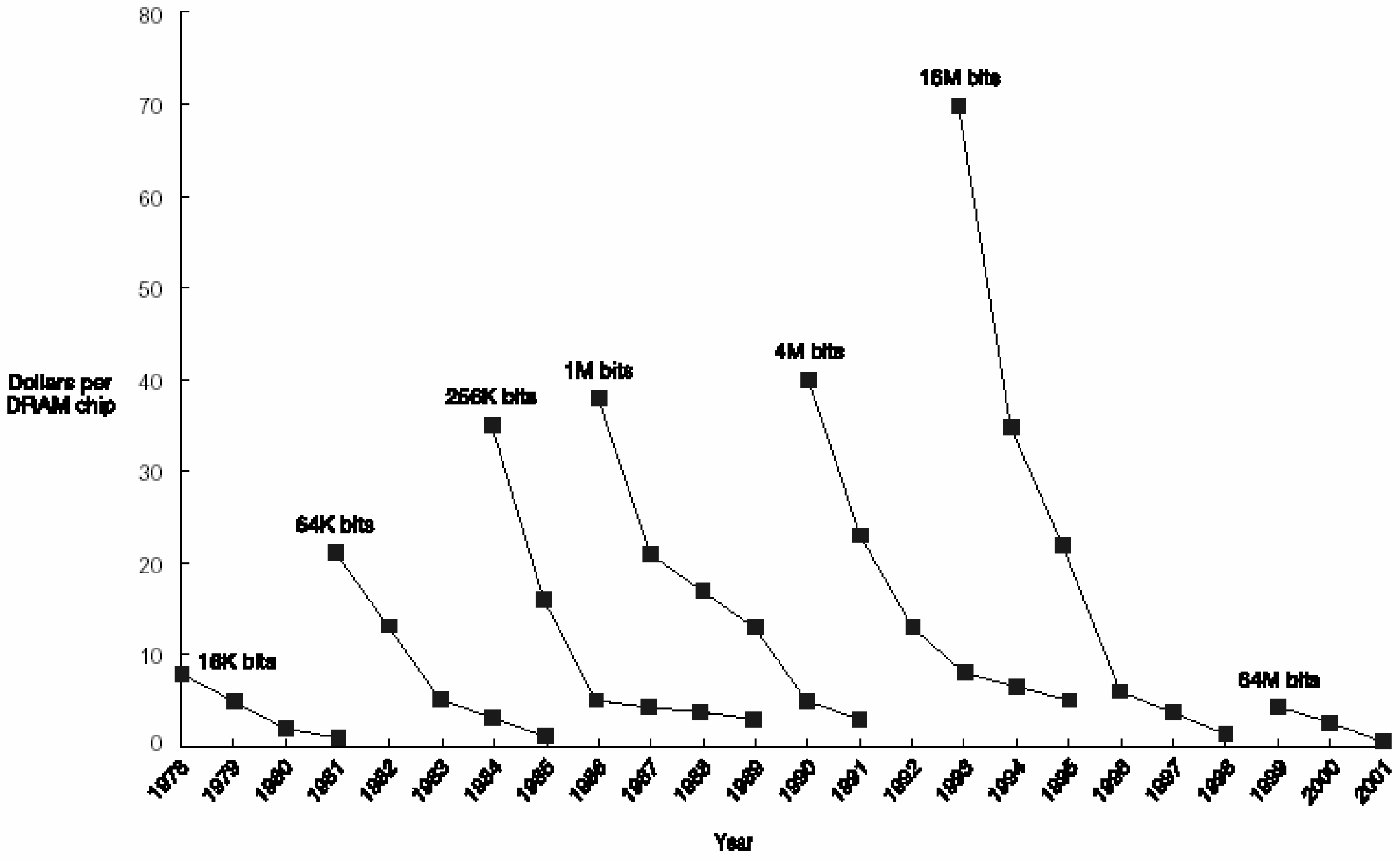
- Fast paced industry
  - early use of technology is promoted
- Learning curve & process stabilization
  - reduces costs over time
- Yield - metric of technology maturity
  - yield is % of manufactured chips that actually work
  - ==> things get cheaper with time till they hit 10-20% of initial
- Increasing cost of fab capital
  - price per unit has increased
  - BUT - cost/function/second going down very rapidly
    - what’s missing from this metric??

Cost of an IC

- More integration ➔ IC cost is bigger piece of total

$$\text{IC-cost} = \frac{\text{Die-cost} + \text{Die-test-cost} + \text{Die-package-cost}}{\text{Final-Test-Yield}}$$

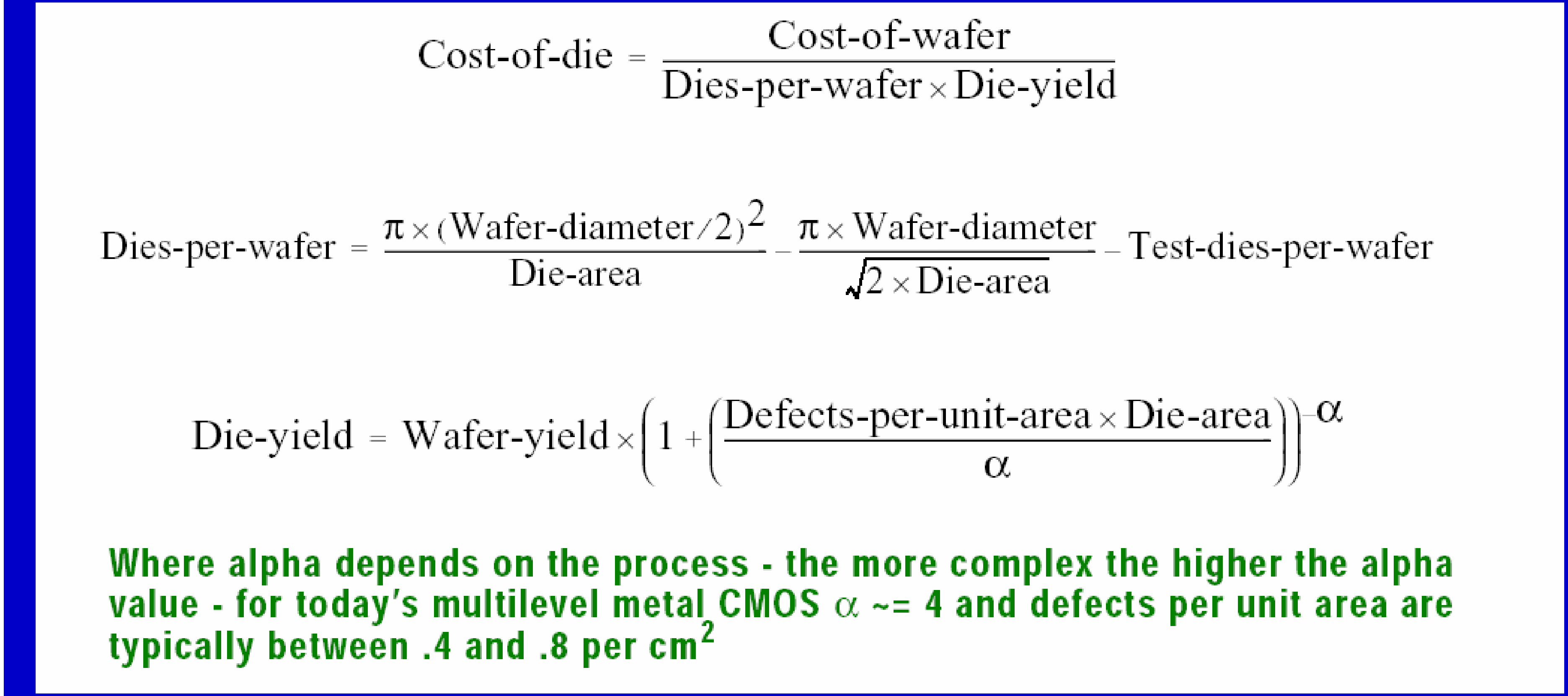
DRAM costs



Dollars per DRAM chip
80
70
60
50
40
30
20
10
0
16K bits
64K bits
256K bits
1M bits
4M bits
16M bits
64M bits
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
Year

Cost of Die

- Compute
  - # dies/wafer & yield as a function of die area



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

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

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

Where alpha depends on the process - the more complex the higher the alpha value - for today's multilevel metal CMOS α ~= 4 and defects per unit area are typically between .4 and .8 per cm^2

Modern Processor Die Sizes

- Pentium Clones
  - AMD
    - .35u K6 = 162 mm^2
    - .25u K6-2 = 68 mm^2
    - .25u K6-3 (256K L1) = 135 mm^2
    - .18u Athlon, 37M T's, 6-layer copper, = 120mm^2
  - Cyrix (they died)
    - 6u 6x86 = 394/225 mm^2
    - .35u 6x86 = 169 mm^2
    - .25u 6x86 (64K L1) = 65 mm^2
  - IDT (they died)
    - .35u Centaur C6 = 88 mm^2
    - .25u C6 = 60 mm^2

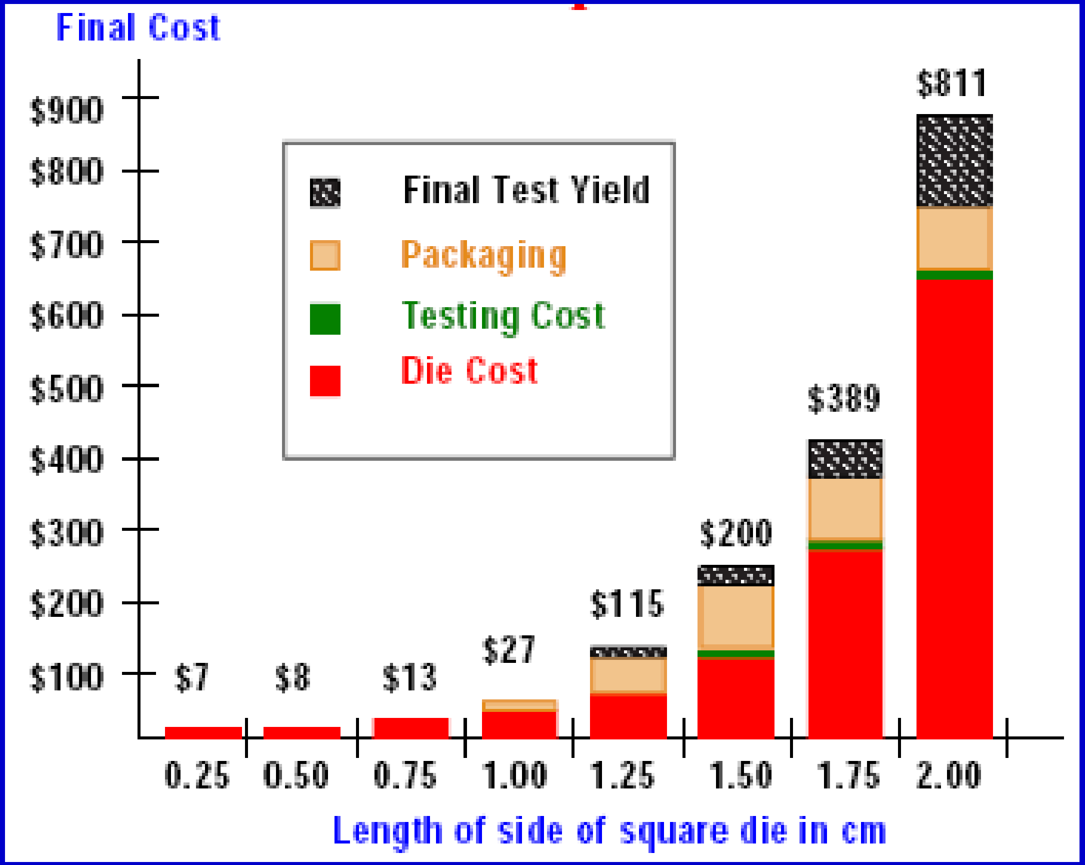
More Die Sizes

- Intel
  - Pentium
    - .8u Pentium 60 = 288 mm^2
    - .6u Pentium 90 = 156 mm^2
    - .35u = 91 mm^2
    - .35u MMX = 140/128 mm^2
  - Pentium Pro
    - .6u = 306 mm^2
    - .35u = 195 mm^2
    - .6u w/ 256K L2 = 202 mm^2
    - .35u w/512K L2 = 242 mm^2
- Pentium II
    - .35u = 205 mm^2
    - .25u = 105 mm^2

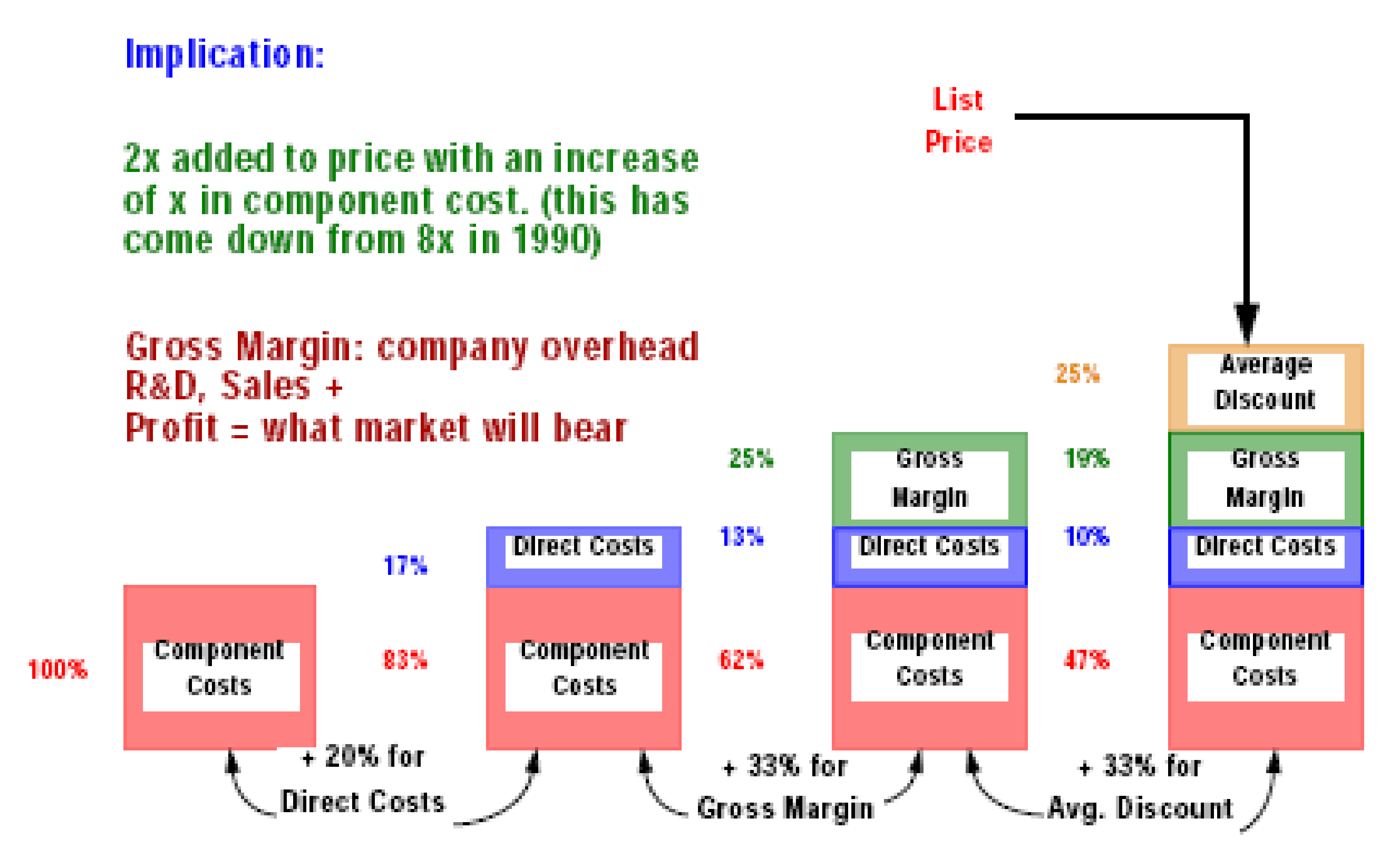
RISC Die Sizes

- HP
  - .5u PA-8200 = ~400 mm^2
  - .25u PA-8600, 116M T’s, 5-metal = 468mm^2
  - .18u SOI, 186M T’s, 7-copper = 305mm^2
- DEC
  - .5u 21164 = 298 mm^2
  - .35u 21264 = 310 mm^2
- Motorola
  - .5u PPC 604 = 196 mm^2
  - .35u PPC 604e = 148/96mm^2
  - .25u PPC 604e = 47.3 mm^2
  - .81u, G4 7410, 10.5M T’s, 6-metal = 62mm^2
- • Transmeta
  - Crusoe TM5600
  - .18u, 5-copper, 36.8M T’s = 88 mm^2

Final Chip Cost vs. Size



Turning Cost into Price



Implication:

2x added to price with an increase of x in component cost. (this has come down from 8x in 1990)

Gross Margin: company overhead R&D, Sales + Profit = what market will bear

Direct Costs: labor costs, purchasing components

Outline

- What is Computer Architecture?
  - Fundamental Abstractions & Concepts
- Instruction Set Architecture & Organization
- Why Take This Course?
- Technology
- Performance
- Computer Architecture Renaissance

Measuring Performance

- Several kinds of time”
  - stopwatch - it’s what you see but is dependent on
    - load
    - I/O delays
    - OS overhead
  - CPU time - time spent computing your program
    - factors out time spent waiting for I/O delays
    - but includes the OS + your program
  - Hence system CPU time, and user CPU time

OS Time

- Unix time command reports

  27.2u 11.1s 56.6 68%

  – 27.2 seconds of user CPU time

  – 11.1 seconts of system CPU time

  – 56.6 seconds total elapsed time

  – % of elapsed time that is user + system CPU time

    - tells you how much time you spent waiting as a %

Benchmarks

- Toy benchmarks
  - quicksort, 8-queens
    - best saved for intro. to programming homeworks
- Synthetic benchmarks
  - most commonly used since they try to mimic real programs
  - problem is that they don’t - each suite has it’s own bias
  - no user really runs them
  - they aren’t even pieces of real programs
  - they may reside in cache & don’t test memory performance
- At the very least you must understand what the benchmark code is in order to understand what it might be measuring

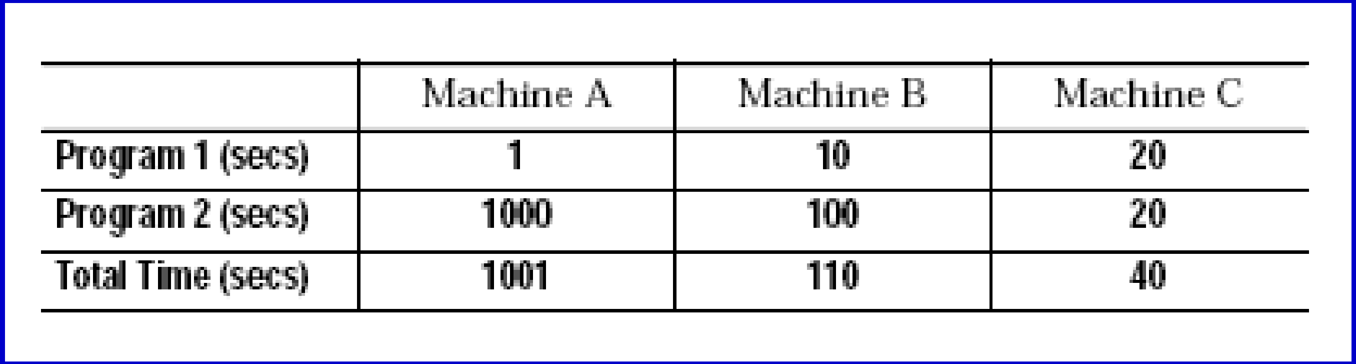
Benchmarks

- Lots of suites - examples
  - Dhrystone - tells you how well integers work
  - Loops and Linpack - mostly floating point matrix frobbing
  - PC specific
    - Business Winstone - composite of browser and office apps
    - CC Winstone - content creation version - Photoshop, audio editing etc.
    - Winbench - collection of subsystem tests that target CPU, disk, and video subsystems
- SPEC2000 (text bias lies here - see table 1.12 for details)
  - 4th generation - primarily designed to test CPU performance
  - CINT2000 - 11 integer benchmarks
  - CFP2000 - 14 floating point benchmarks
  - SPECviewperf - graphics performance of systems using OpenGL
  - SPECapc - several large graphics apps
  - SPECSFS - file system test
  - SPECWeb - web server test

More Benchmarks

- TPC
  - transaction processing council
  - many variants depending on transaction complexity
    - TPC-A: simple bank teller transaction style
    - TPC -C: complex database query
    - TPC-H: decision support
    - TPC-R: decision support but with stylized queries (faster than -H)
    - TPC-W: web server
- For embedded systems EEMBC “embassy”
  - 35 kernels in 5 classes
  - 16 automotive/industrial - arithmetic, pointer chasing, table lookup, bit manip, ...
  - 5 consumer - JPEG codec, RGB conversion, filtering
  - 3 networking - shortest path, IP routing, packet classification
  - 4 office automation - graphics and text processing
  - 6 telecommunications - DSP style autocorrelation, FFT, decode, FIR filter, ...

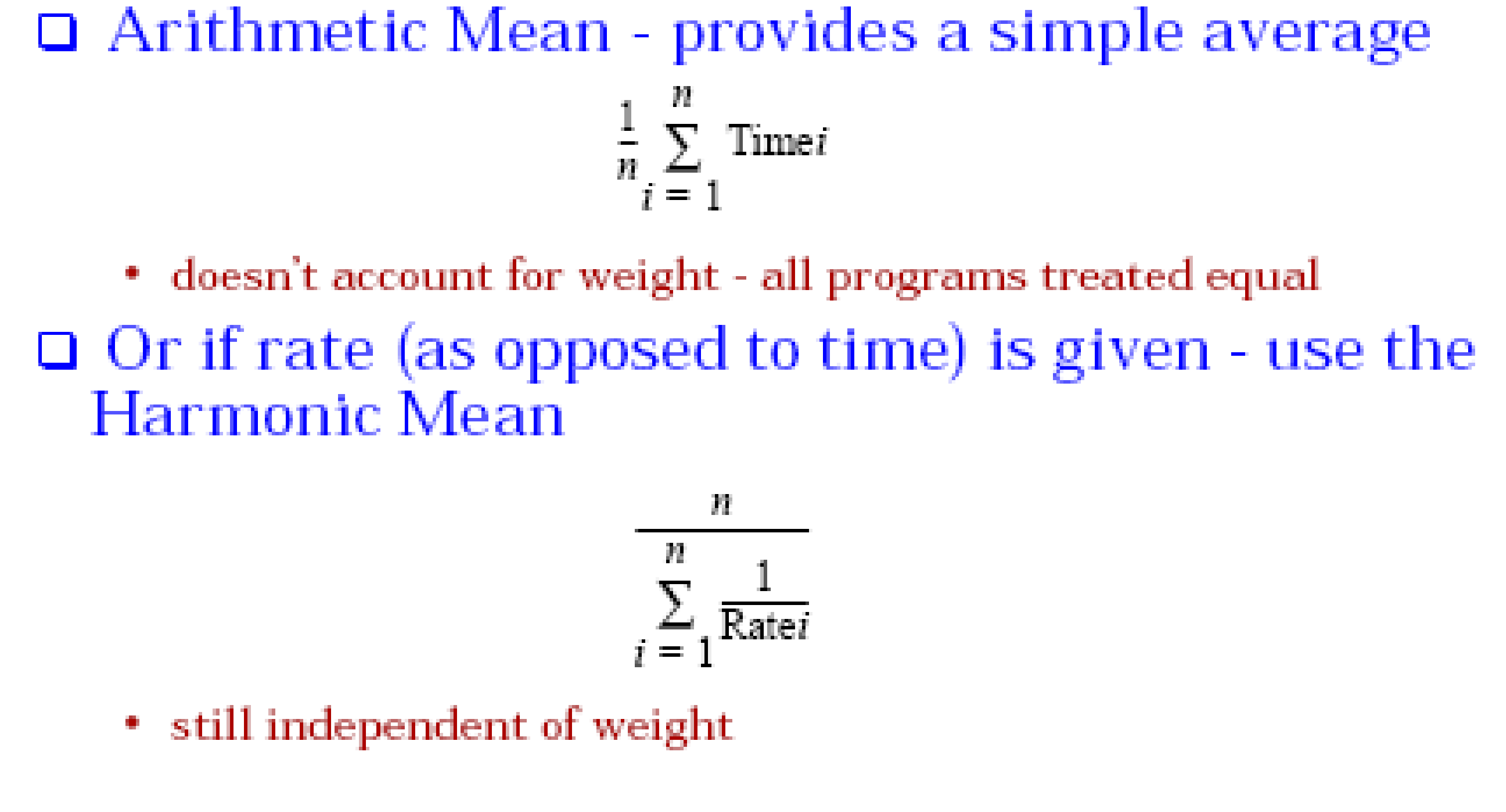
Other Problems



	Machine A	Machine B	Machine C
Program 1 (secs)	1	10	20
Program 2 (secs)	1000	100	20
Total Time (secs)	1001	110	40

- Which is better?
- By how much?
- Are the programs equally important?

Some Aggregate Job Mix Options



❑ Arithmetic Mean - provides a simple average

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

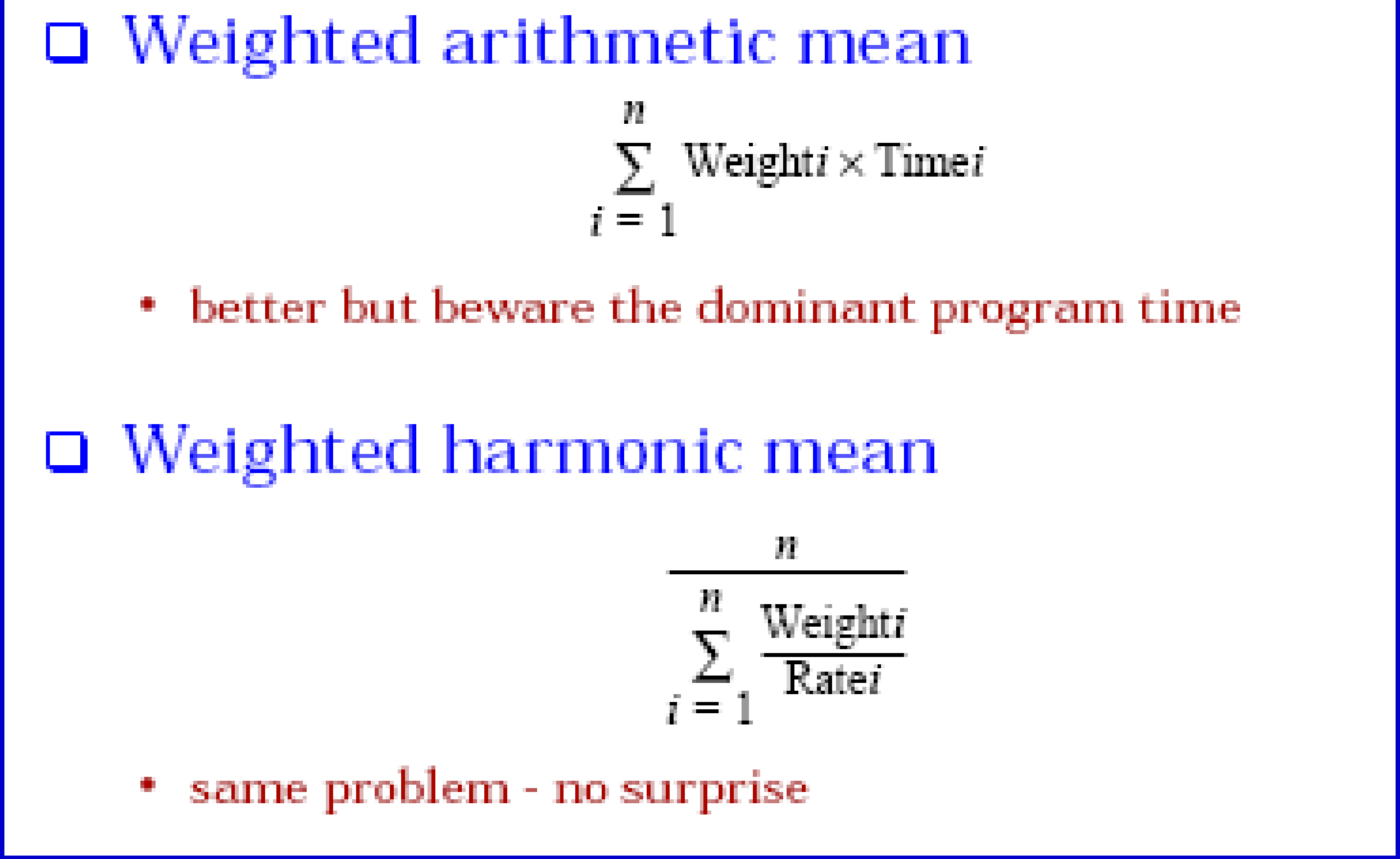
- doesn't account for weight - all programs treated equal

❑ Or if rate (as opposed to time) is given - use the Harmonic Mean

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

- still independent of weight

Weighted Variants



□ Weighted arithmetic mean

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

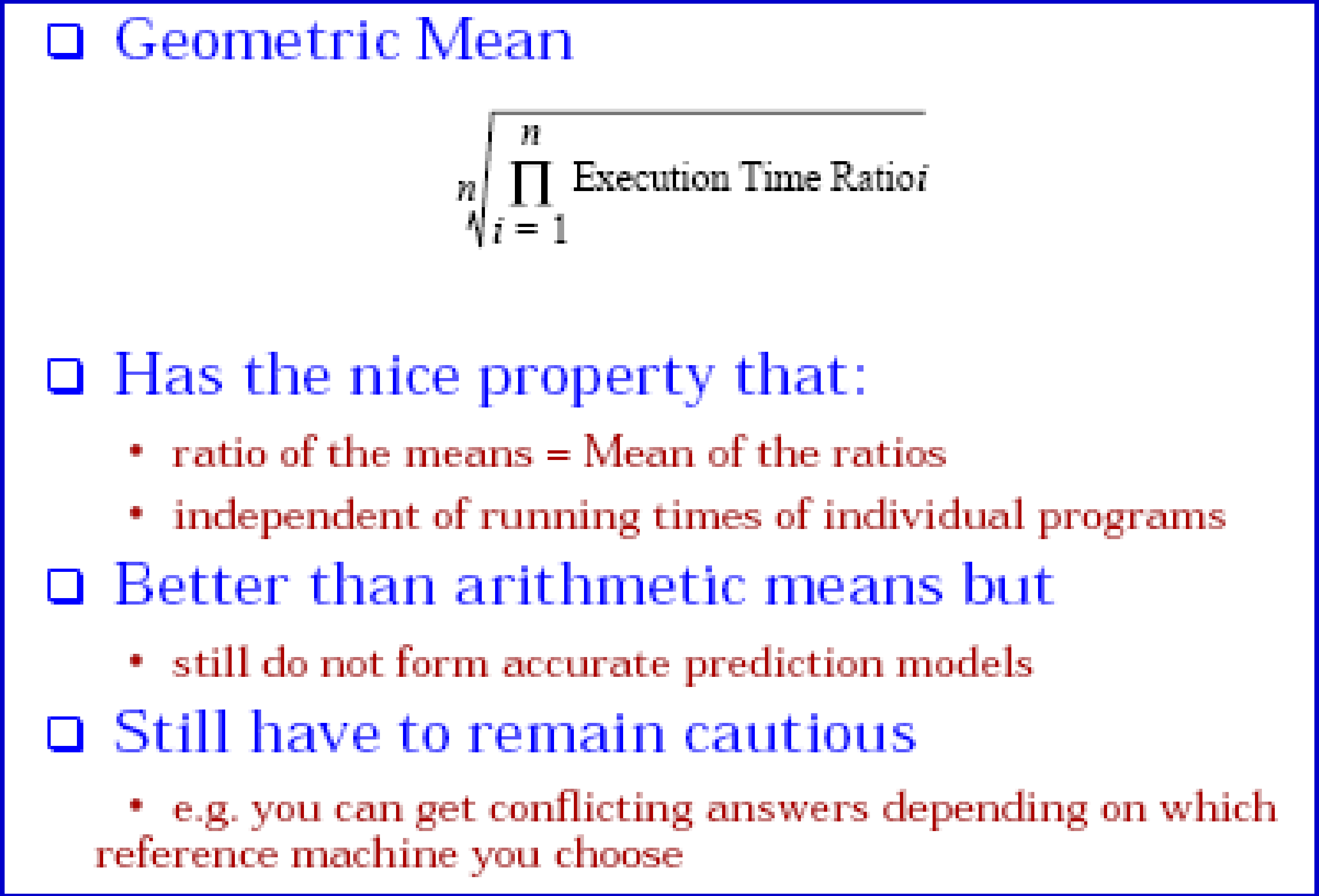
- better but beware the dominant program time

□ Weighted harmonic mean

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

- same problem - no surprise

Normalized Time Metrics



❑ Geometric Mean

$$\sqrt[n]{\prod_{i=1}^{n} \text{Execution Time Ratio}\,i}$$

❑ Has the nice property that:

- ratio of the means = Mean of the ratios
- independent of running times of individual programs

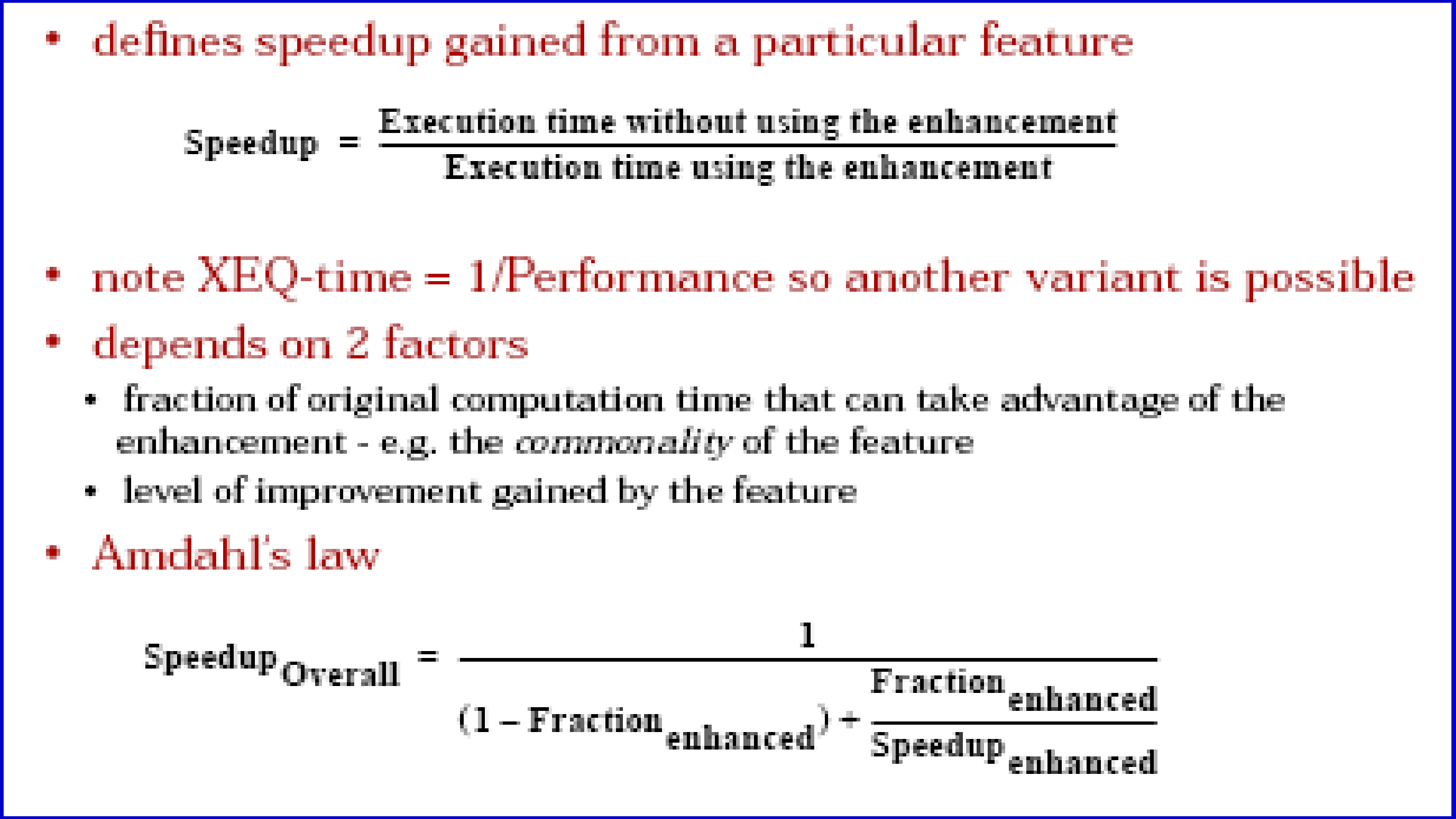
❑ Better than arithmetic means but

- still do not form accurate prediction models

❑ Still have to remain cautious

- e.g. you can get conflicting answers depending on which reference machine you choose

Amdahl’s Law



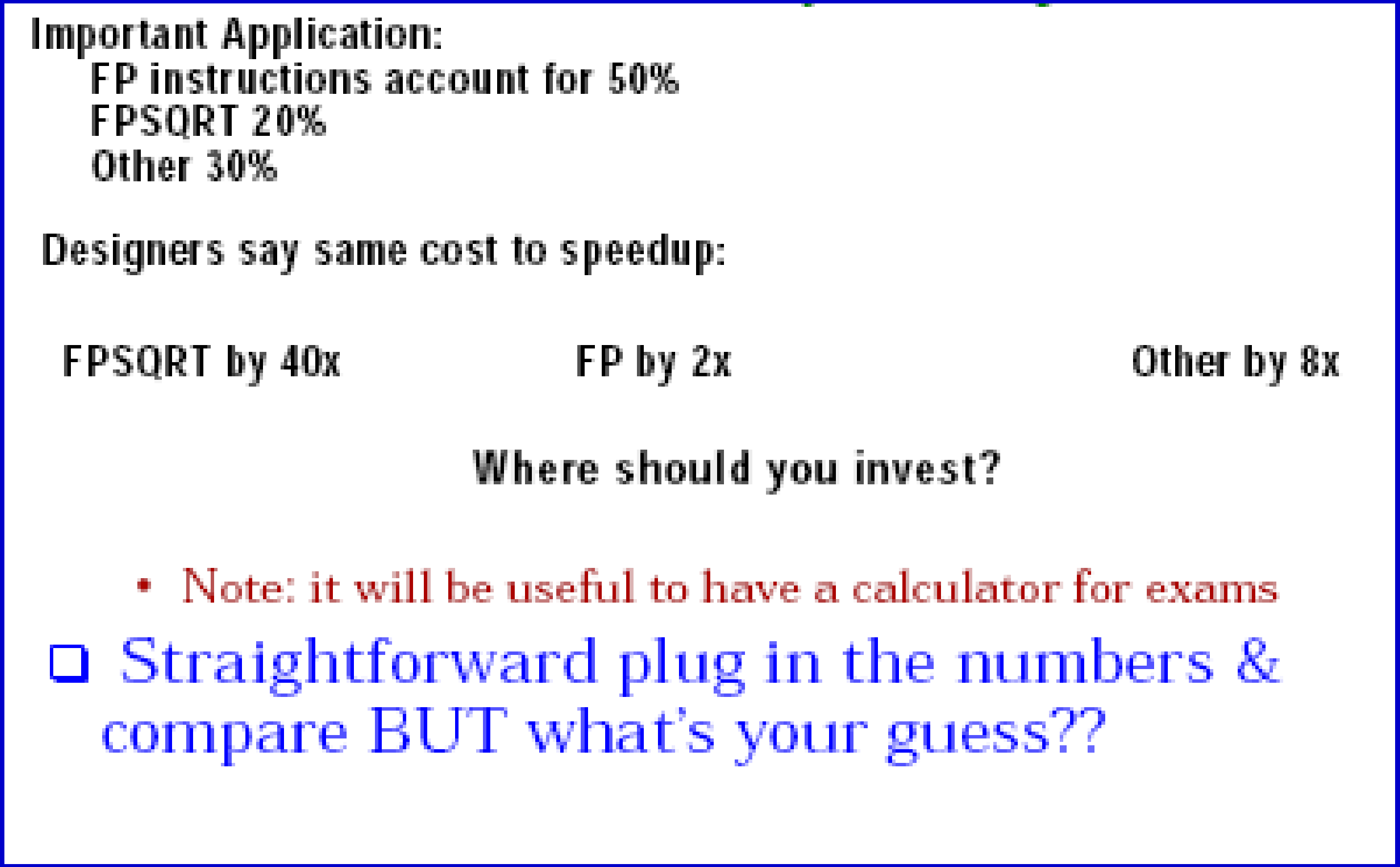
- defines speedup gained from a particular feature

$$\text{Speedup} = \frac{\text{Execution time without using the enhancement}}{\text{Execution time using the enhancement}}$$

- note XEQ-time = 1/Performance so another variant is possible
- depends on 2 factors
  - fraction of original computation time that can take advantage of the enhancement - e.g. the *commonality* of the feature
  - level of improvement gained by the feature
- Amdahl's law

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

Simple Example



Important Application:

FP instructions account for 50%
FPSQRT 20%
Other 30%

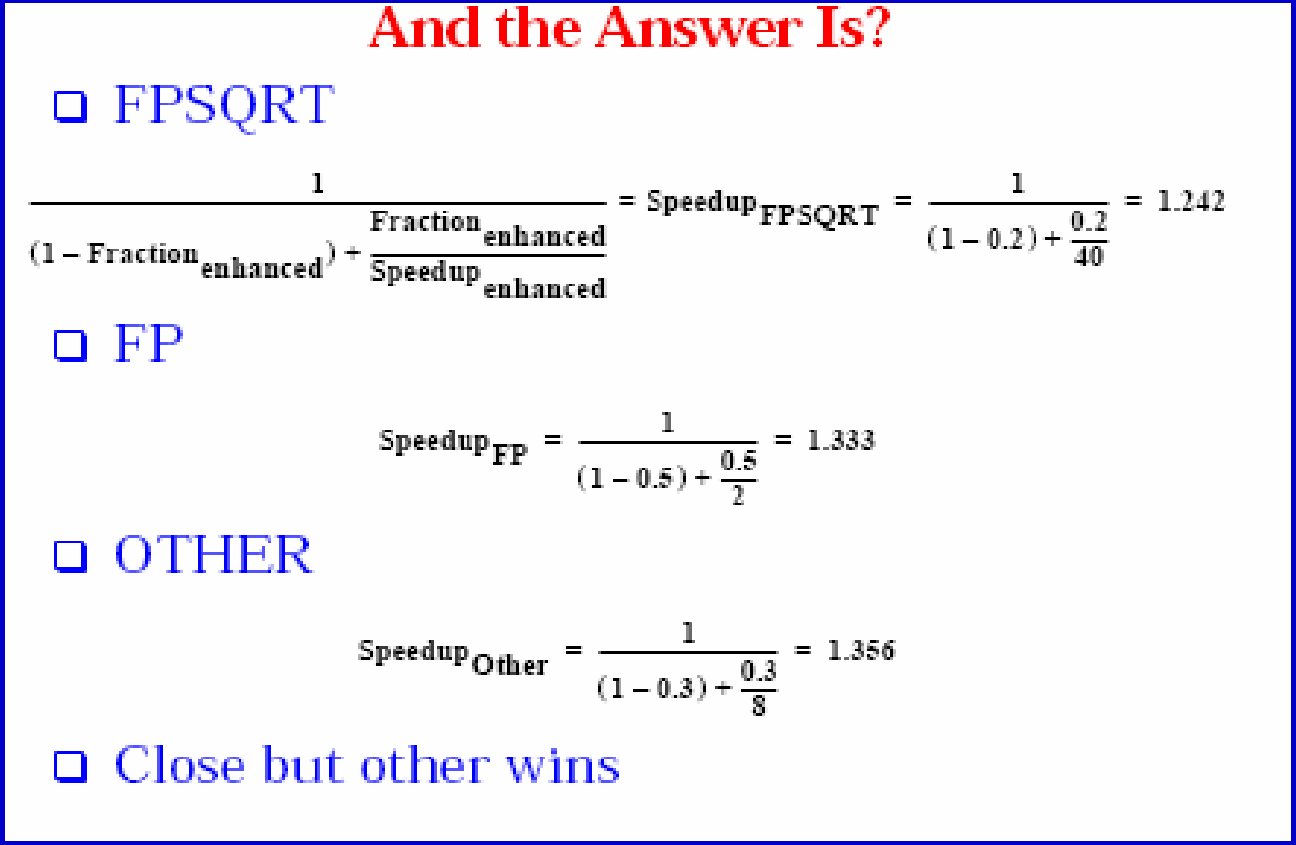
Designers say same cost to speedup:

FPSQRT by 40x FP by 2x Other by 8x

Where should you invest?

- Note: it will be useful to have a calculator for exams

❑ Straightforward plug in the numbers & compare BUT what's your guess??



And the Answer Is?

- ❑ FPSQRT

$$\frac{1}{(1 - \text{Fraction}_{\text{enhanced}}) + \frac{\text{Fraction}_{\text{enhanced}}}{\text{Speedup}_{\text{enhanced}}}} = \text{Speedup}_{\text{FPSQRT}} = \frac{1}{(1 - 0.2) + \frac{0.2}{40}} = 1.242$$

- ❑ FP

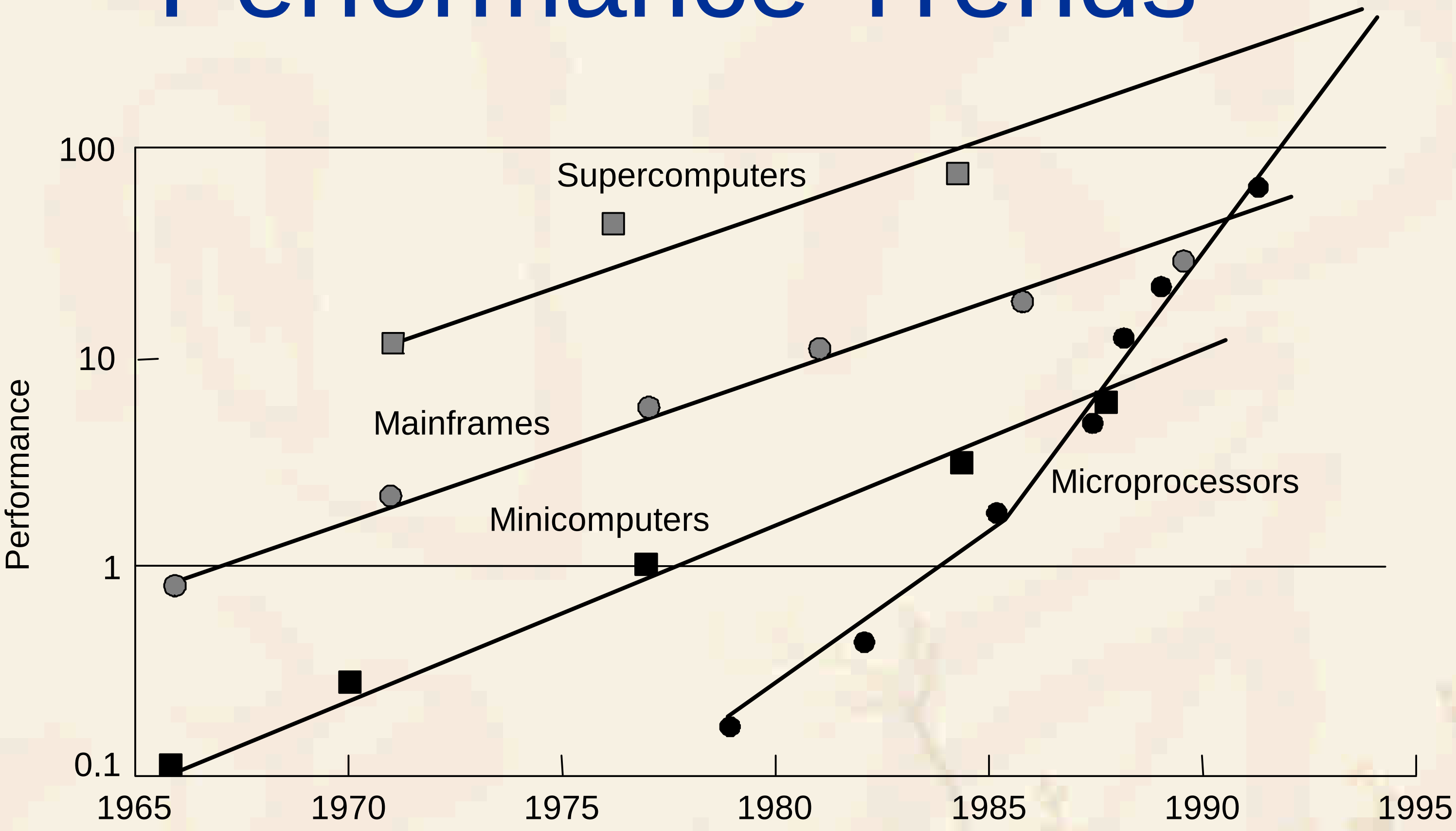
$$\text{Speedup}_{\text{FP}} = \frac{1}{(1 - 0.5) + \frac{0.5}{2}} = 1.333$$

- ❑ OTHER

$$\text{Speedup}_{\text{Other}} = \frac{1}{(1 - 0.3) + \frac{0.3}{8}} = 1.356$$

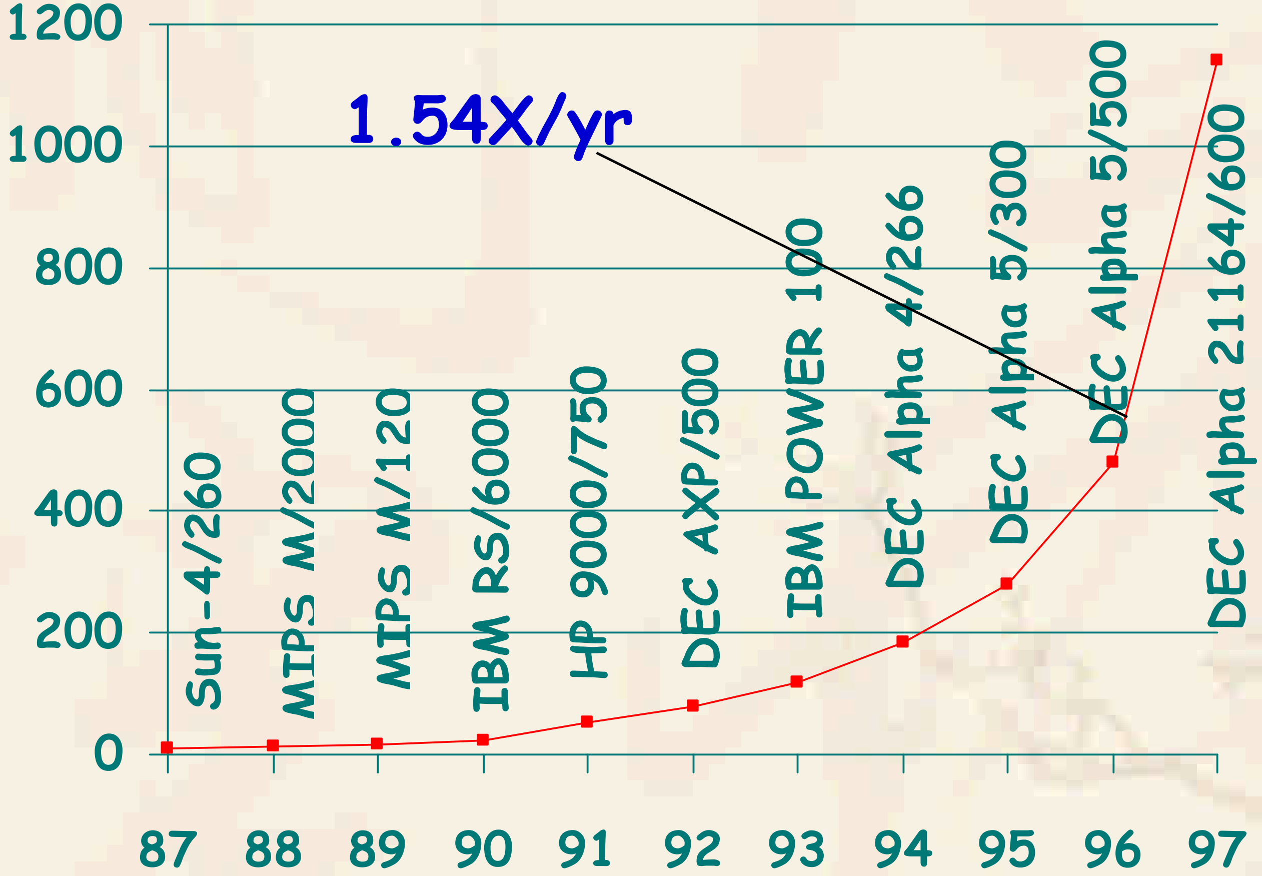
- ❑ Close but other wins

Performance Trends



What do we have Today?

Processor Performance (1.35X before, 1.55X now)



Will Moore’s Law Continue?

- Search “Moore’s Law” on the Internet, and you will see a lot of predictions and arguments.
- Don’t bet your house on it (or any technology stock)…

Definition: Performance

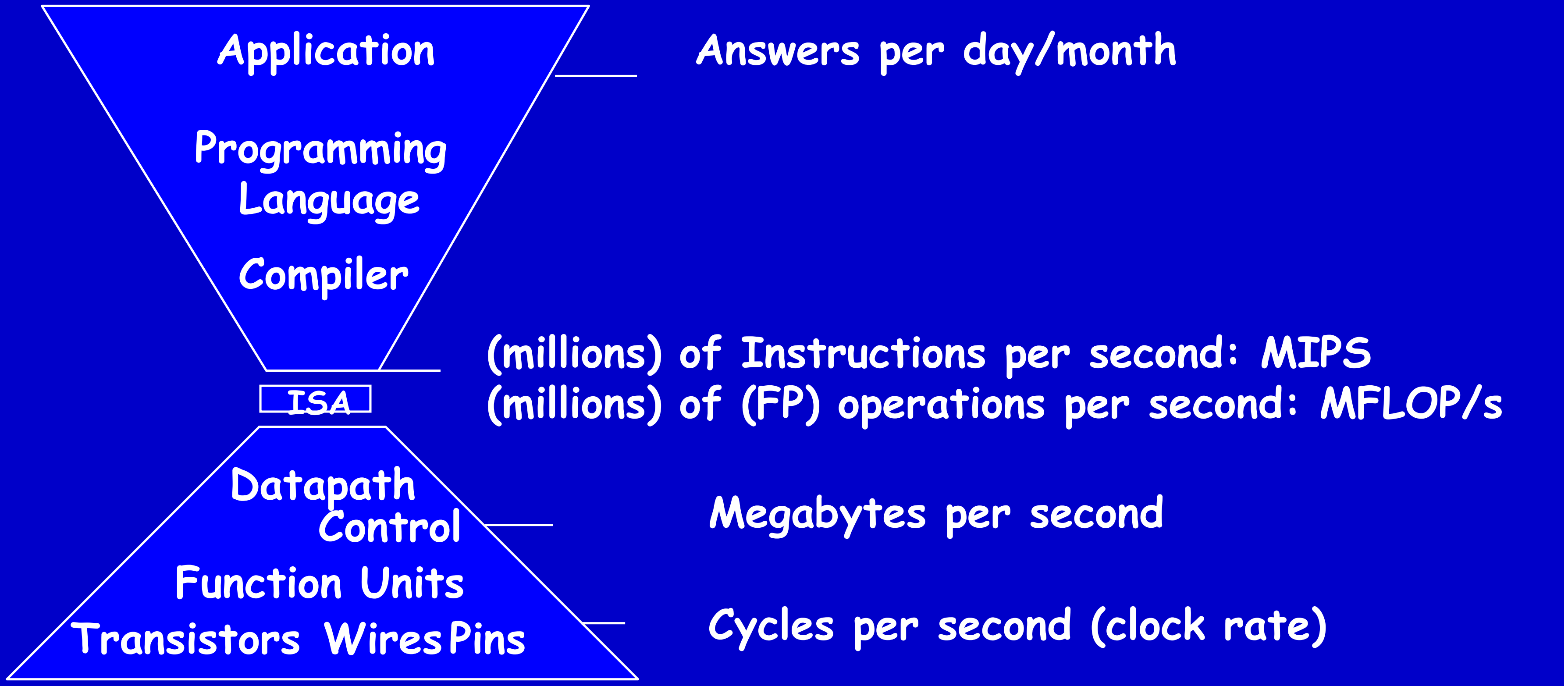
- Performance is in units of things per sec
  - bigger is better
- If we are primarily concerned with response time

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

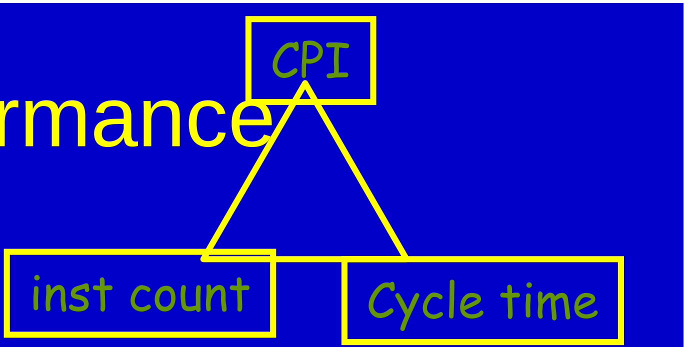
" X is n times faster than Y" means

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

Metrics of Performance



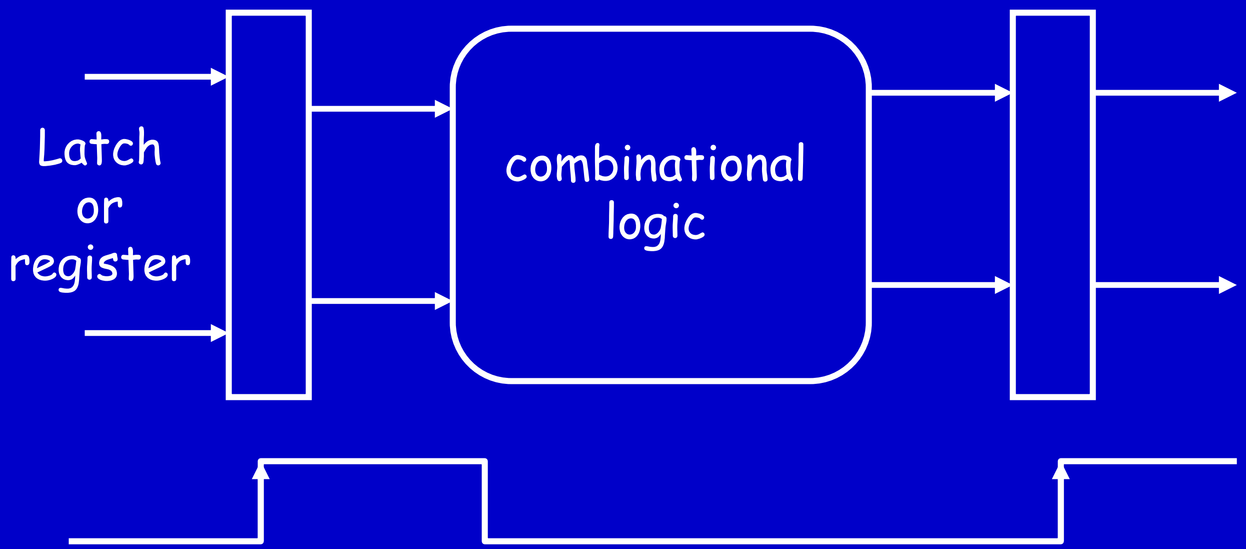
Components of Performance



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

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

What’s a Clock Cycle?



- State changes as Clock “ticks”
- Old days: 10 levels of gates
- Today: determined by numerous time-of-flight issues + gate delays
  - clock propagation, wire lengths, drivers

Integrated Approach

What really matters is the functioning of the complete system, I.e. hardware, runtime system, compiler, and operating system

In networking, this is called the "End to End argument"

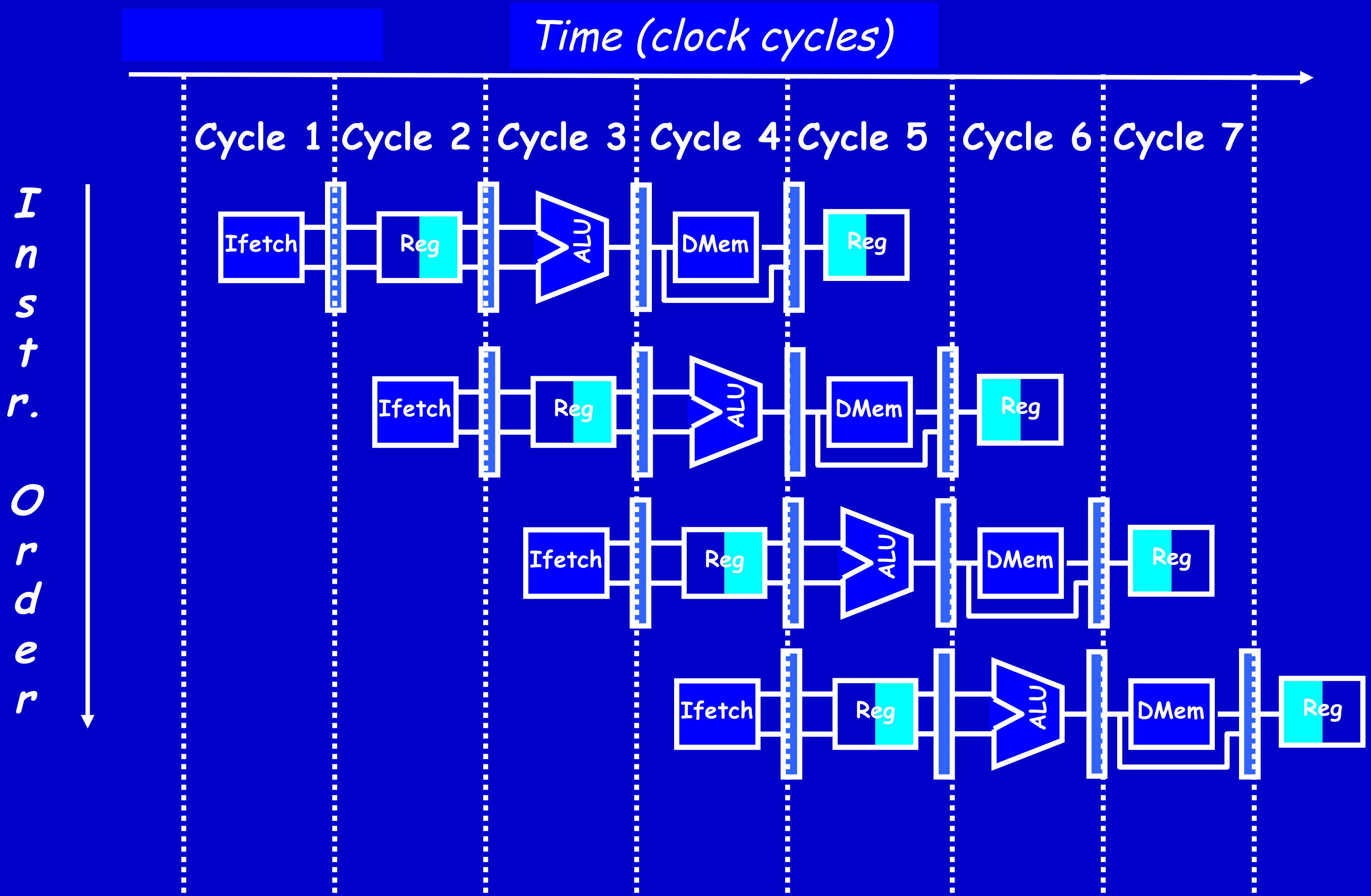
- Computer architecture is not just about transistors, individual instructions, or particular implementations
- Original RISC projects replaced complex instructions with a compiler + simple instructions

How do you turn more stuff into more performance?

- Do more things at once
- Do the things that you do faster

- Beneath the ISA illusion….

Pipelined Instruction Execution

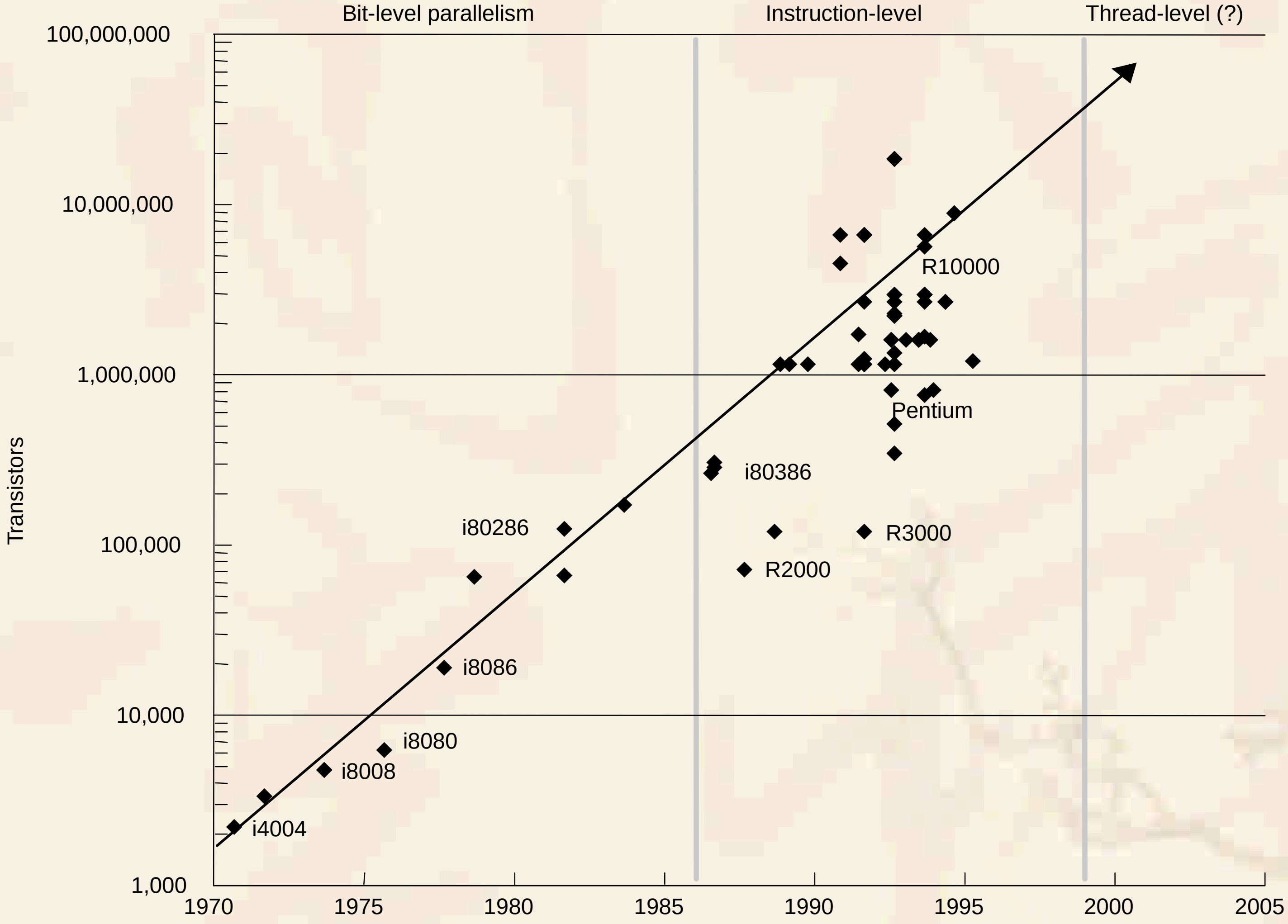


]

Limits to pipelining

- Maintain the von Neumann “illusion” of one instruction at a time execution
- 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).

A take on Moore’s Law



Progression of ILP

- 1st generation RISC - pipelined
  - Full 32-bit processor fit on a chip => issue almost 1 IPC
    - Need to access memory 1+x times per cycle
  - Floating-Point unit on another chip
  - Cache controller a third, off-chip cache
  - 1 board per processor → multiprocessor systems
- 2nd generation: superscalar
  - Processor and floating point unit on chip (and some cache)
  - Issuing only one instruction per cycle uses at most half
  - Fetch multiple instructions, issue couple
    - Grows from 2 to 4 to 8 …
  - How to manage dependencies among all these instructions?
  - Where does the parallelism come from?
- VLIW
  - Expose some of the ILP to compiler, allow it to schedule instructions to reduce dependences

Modern ILP

- Dynamically scheduled, out-of-order execution
- Current microprocessor fetch 10s of instructions per cycle
- Pipelines are 10s of cycles deep

=> many 10s of instructions in execution at once

- Grab a bunch of instructionsdetermine all their dependences, eliminate dep's wherever possible, throw them all into the execution unit, let each one move forward as its dependences are resolved
- Appears as if executed sequentially
- On a trap or interrupt, capture the state of the machine between instructions perfectly
- Huge complexity

Have we reached the end of ILP?

- Multiple processor easily fit on a chip
- Every major microprocessor vendor has gone to multithreading
  - Thread: loci of control, execution context
  - Fetch instructions from multiple threads at once, throw them all into the execution unit
  - Intel: hyperthreading, Sun:
  - Concept has existed in high performance computing for 20 years (or is it 40? CDC6600)
- Vector processing
  - Each instruction processes many distinct data
  - Ex: MMX
- Raise the level of architecture – many processors per chip

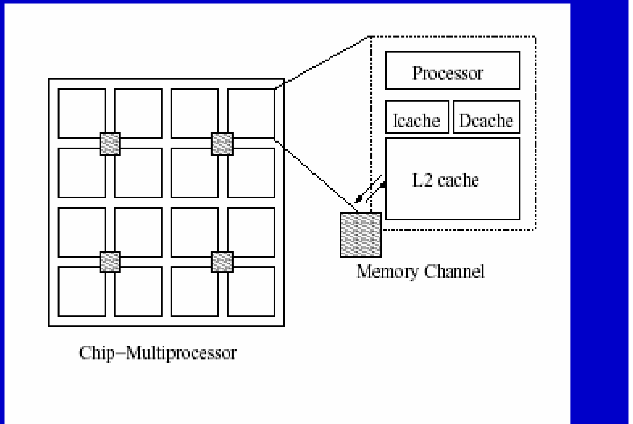
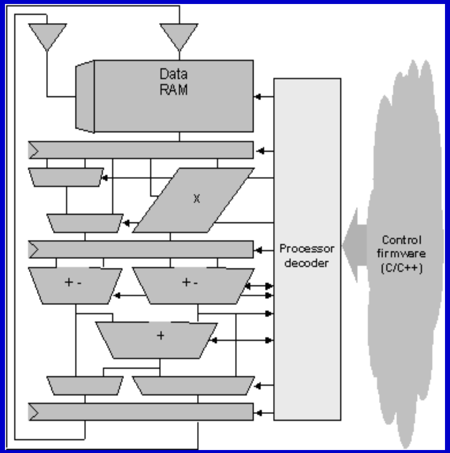


Figure 1. Chip-multiprocessor model.



Tensilica Configurable Proc

When all else fails - guess

- Programs make decisions as they go
  - Conditionals, loops, calls
  - Translate into branches and jumps (1 of 5 instructions)
- How do you determine what instructions for fetch when the ones before it haven’t executed?
  - Branch prediction
  - Lot’s of clever machine structures to predict future based on history
  - Machinery to back out of mis-predictions
- Execute all the possible branches
  - Likely to hit additional branches, perform stores

⇒speculative threads

⇒What can hardware do to make programming (with performance) easier?

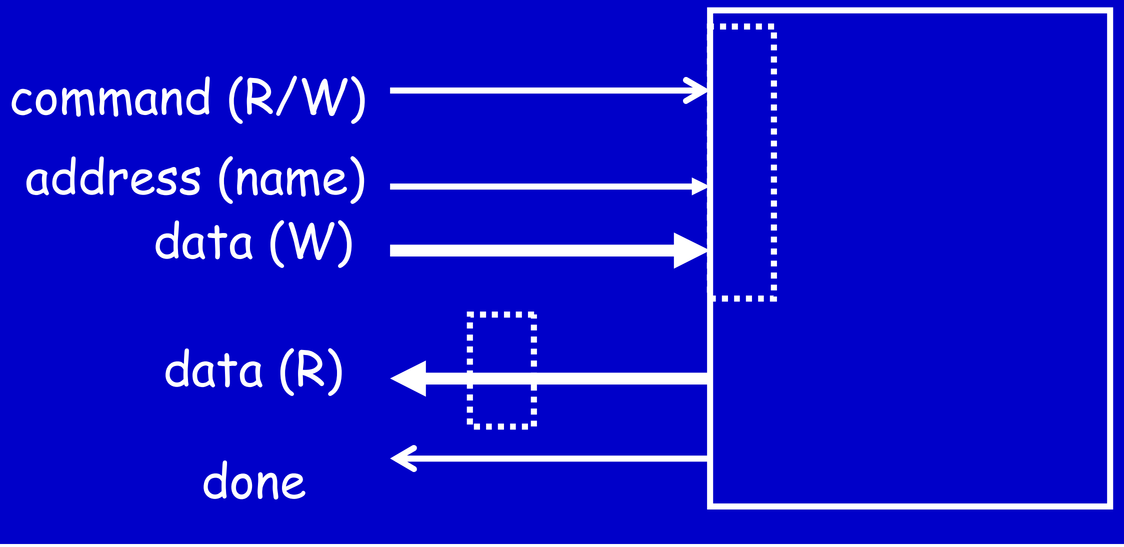
Numbers and Pictures

- Numbers talk!
  - What is a quantitative approach?
  - How to collect VALID data?
  - How to analyze data and extract useful information?
  - How to derive convincing arguments based on numbers?

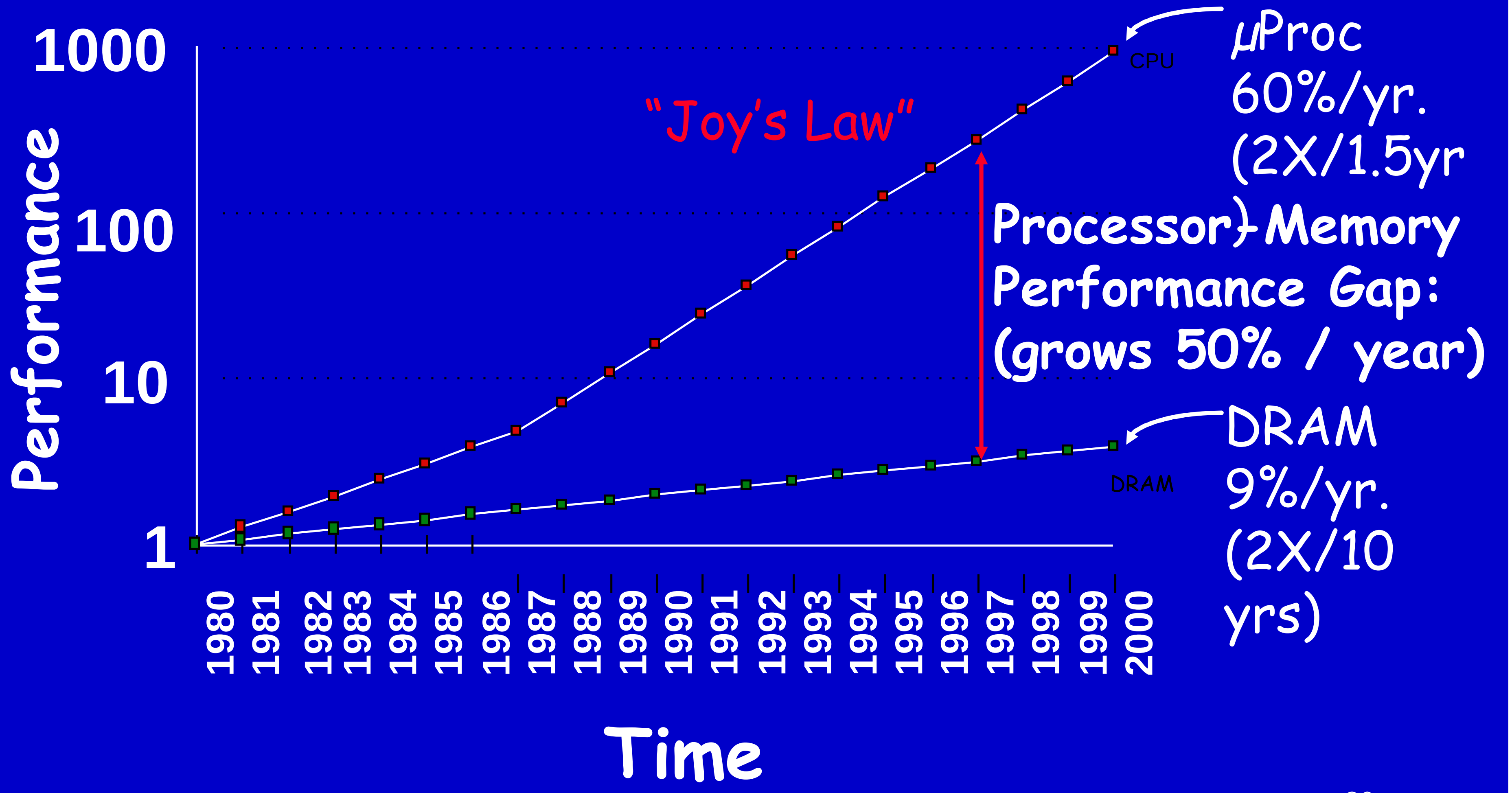
- Pictures
  - A good picture = a thousand words
  - Good for showing trends and comparisons
  - High-level managers have no time to read numbers
  - Business people want pictures and charts

The Memory Abstraction

- Association of <name, value> pairs
  - typically named as byte addresses
  - often values aligned on multiples of size
- Sequence of Reads and Writes
- Write binds a value to an address
- Read of addr returns most recently written value bound to that address



Processor-DRAM Memory Gap (latency)



Levels of the Memory Hierarchy

Capacity
Access Time
Cost

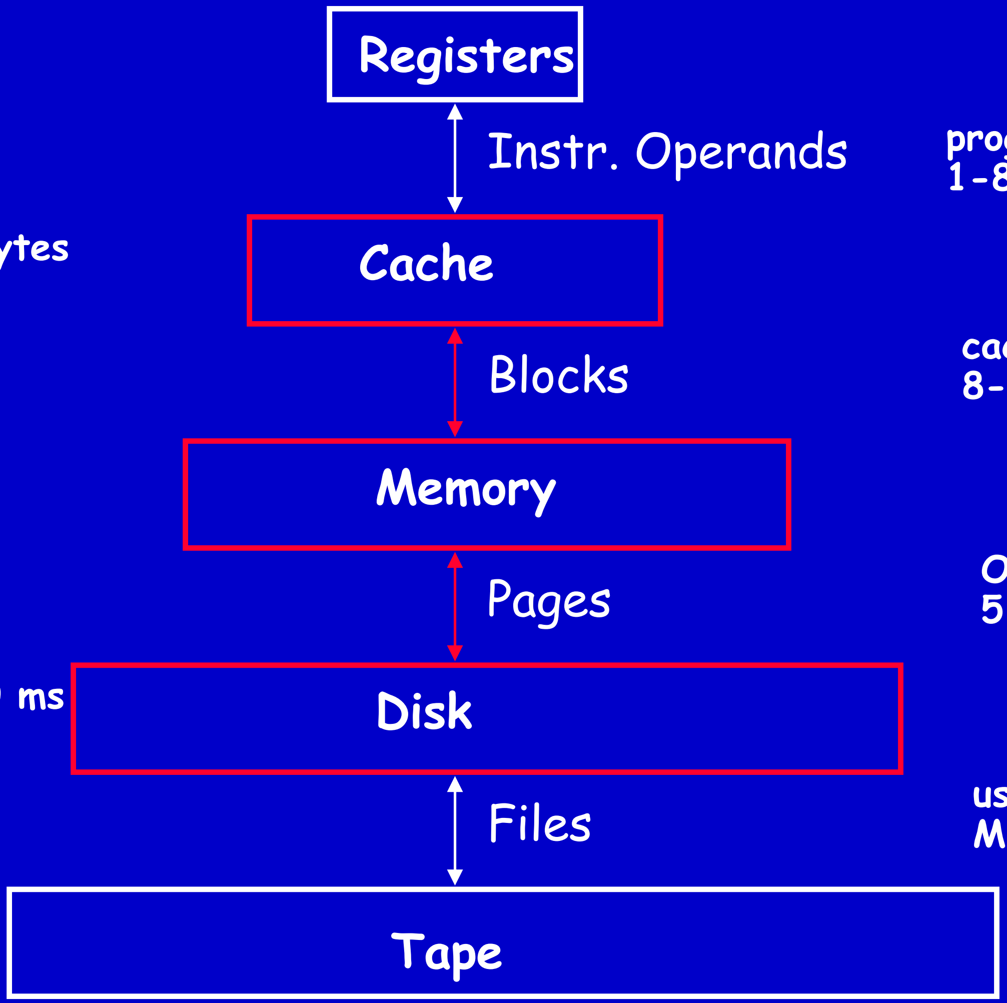
CPU Registers
100s Bytes
<< 1s ns

Cache
10s-100s K Bytes
~1 ns
$1s/ MByte

Main Memory
M Bytes
100ns- 300ns
$< 1/ MByte

Disk
10s G Bytes, 10 ms
(10,000,000 ns)
$0.001/ MByte

Tape
infinite
sec-min
$0.0014/ MByte



Upper Level

Staging
Xfer Unit

prog./compiler
1-8 bytes

cache cntl
8-128 bytes

OS
512-4K bytes

user/operator
Mbytes

faster

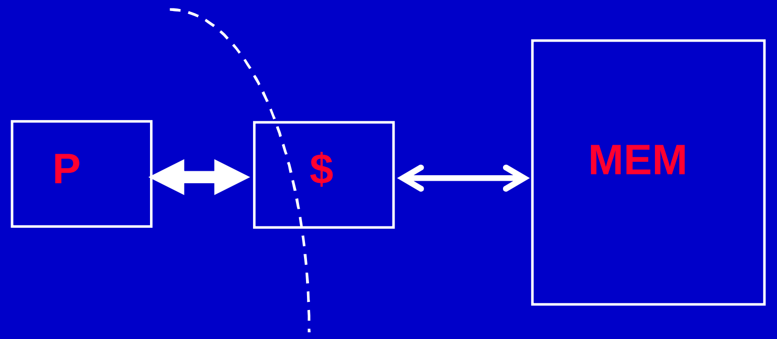
Larger

Lower Level

circa 1995 numbers

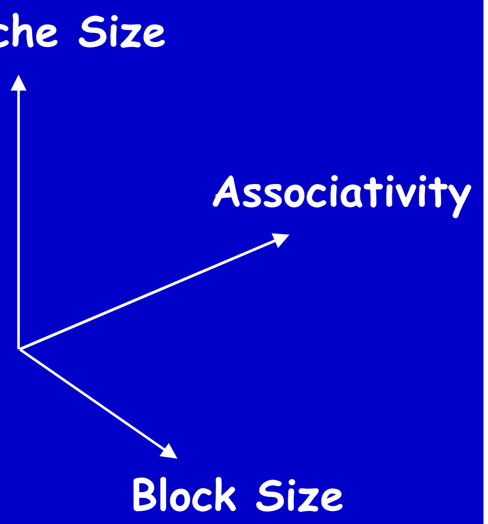
The Principle of Locality

- 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., straightline code, array access)
- Last 30 years, HW relied on locality for speed

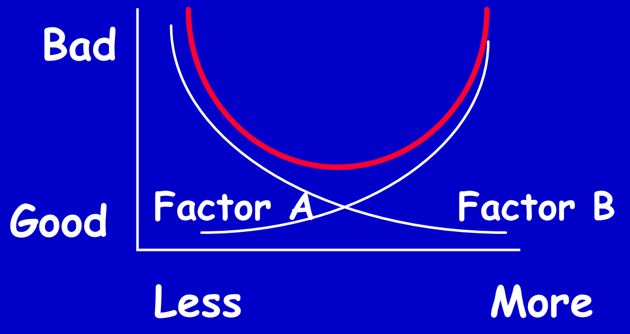


The Cache Design Space

- Several interacting dimensions
  - cache size
  - block size
  - associativity
  - replacement policy
  - write-through vs write-back



- The optimal choice is a compromise
  - depends on access characteristics
    - workload
    - use (I-cache, D-cache, TLB)
  - depends on technology / cost
- Simplicity often wins

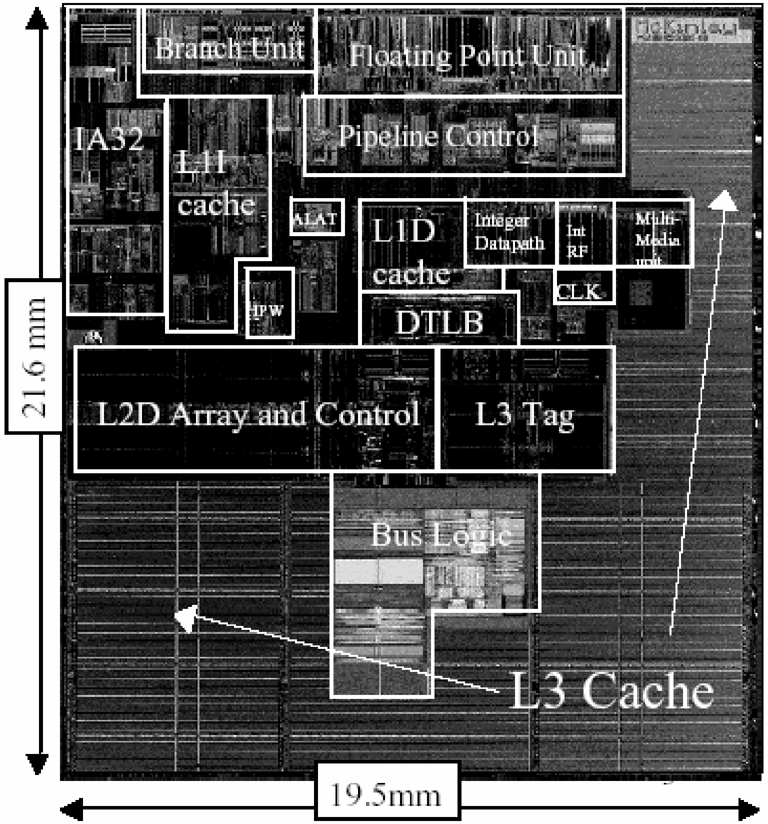


Is it all about memory system design?

- Modern microprocessors are almost all cache

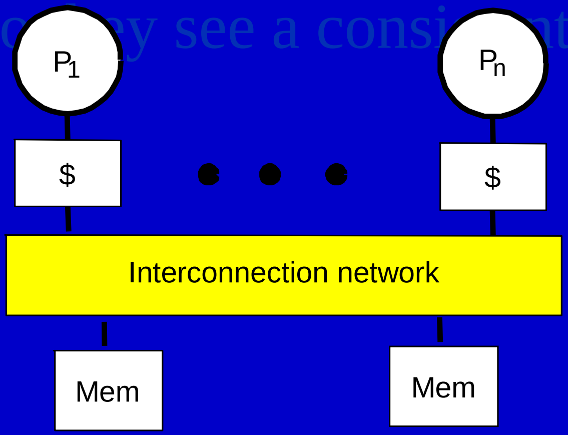
McKinley Floorplan

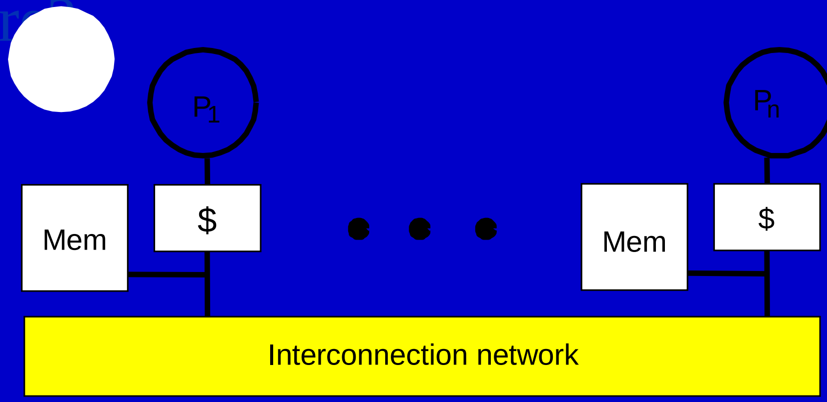
- 0.18 µm, Al process
- 200MHz system clock
- 1GHz core clock
- Core clocking:
  - 260 mm^2
  - 1 primary driver
  - 5 repeaters
  - 33 delay SLCBs
  - 18k gated buffers
  - 157k clocked latches



Memory Abstraction and Parallelism

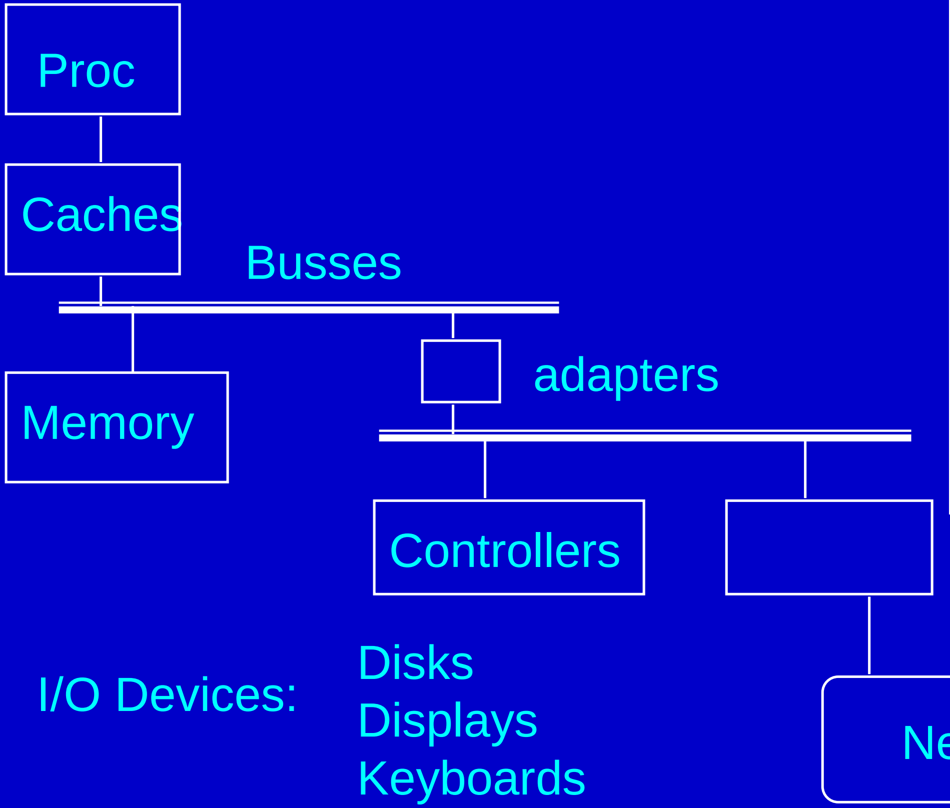
- Maintaining the illusion of sequential access to memory
- What happens when multiple processors access the same memory at once?
  - Do they see a consistent picture?





- Processing and processors embedded in the

System Organization:
It's all about communication



Proc

Caches

Busses

Memory

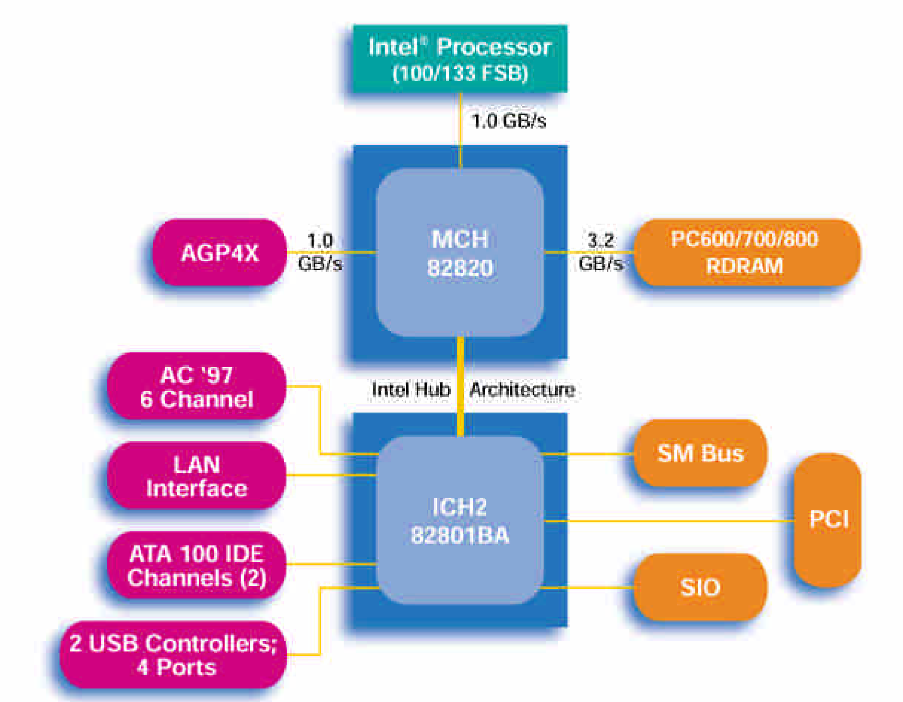
adapters

Controllers

I/O Devices:

Disks
Displays
Keyboards

Networks

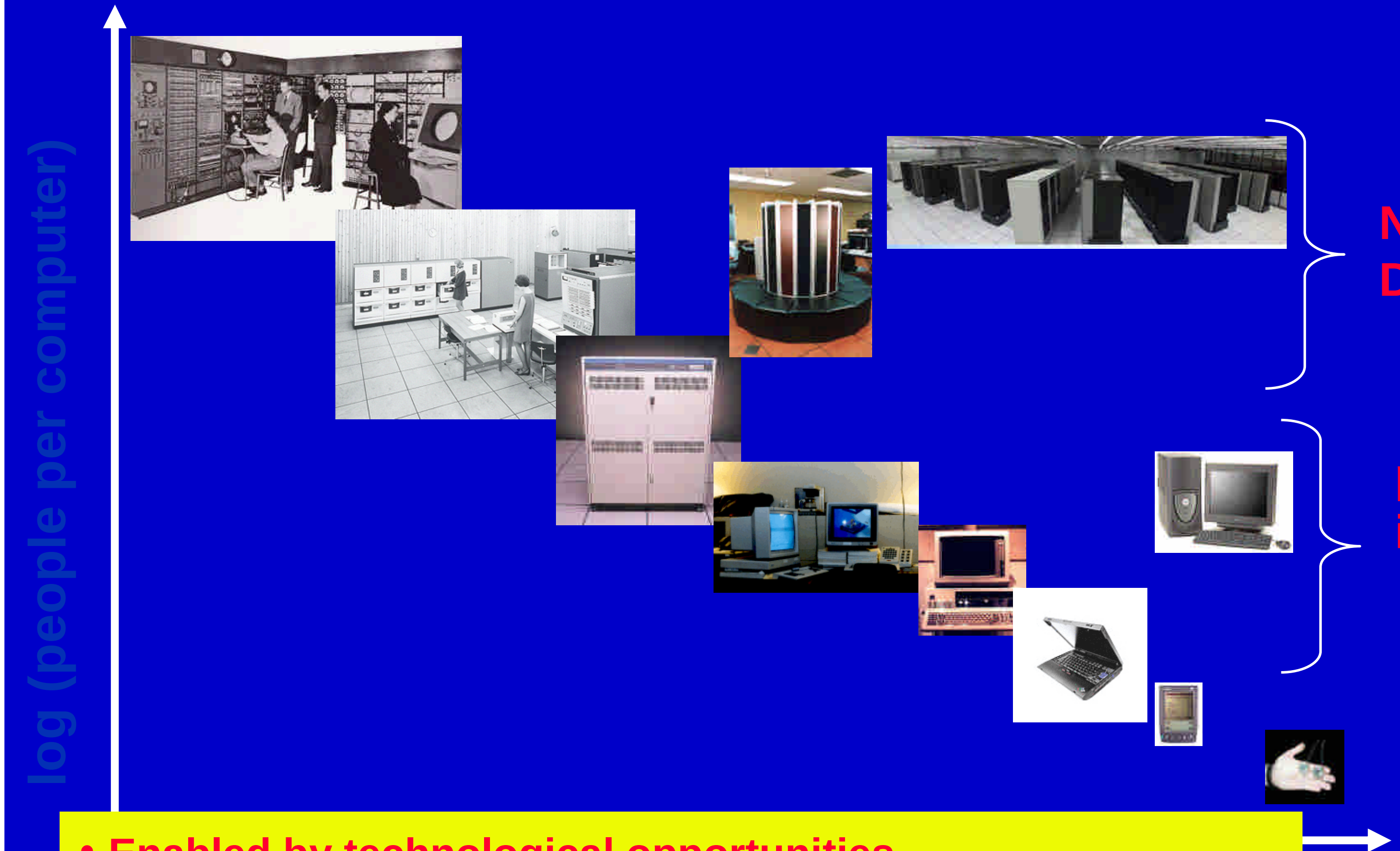


Pentium III Chipset

Breaking the HW/Software Boundary

- Moore's law (more and more trans) is all about volume and regularity
- What if you could pour nano-acres of unspecific digital logic "stuff" onto silicon
  - Do anything with it. Very regular, large volume
- Field Programmable Gate Arrays
  - Chip is covered with logic blocks w/ FFs, RAM blocks, and interconnect
  - All three are "programmable" by setting configuration bits
  - These are huge?
- Can each program have its own instruction set?
- Do we compile the program entirely into hardware?

“Bell’s Law” – new class per decade



Number Crunching
Data Storage

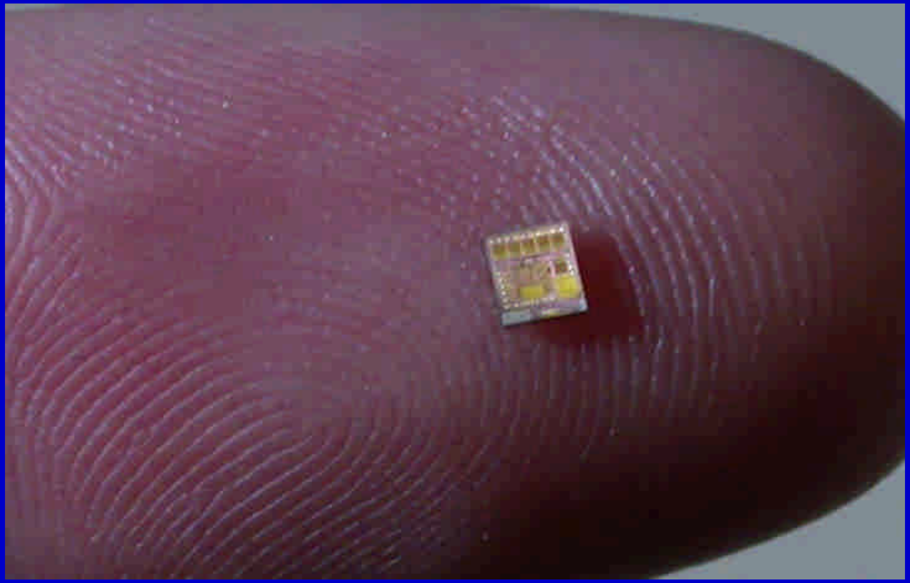
productivity
interactive

streaming
information
to/from physical
world

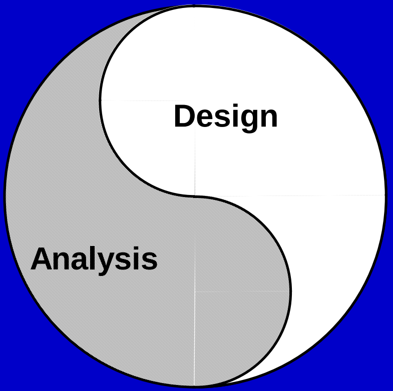
- Enabled by technological opportunities
- Smaller, more numerous and more intimately connected
- Brings in a new kind of application
- Used in many ways not previously imagined

It’s not just about bigger and faster!

- Complete computing systems can be tiny and cheap
- System on a chip
- Resource efficiency
  - Real-estate, power, pins, …

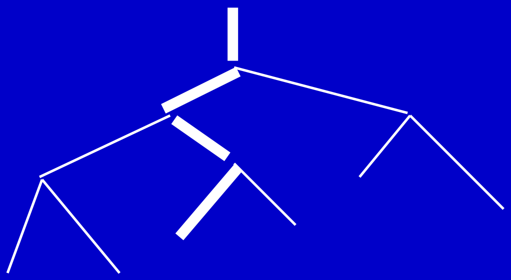


The Process of Design

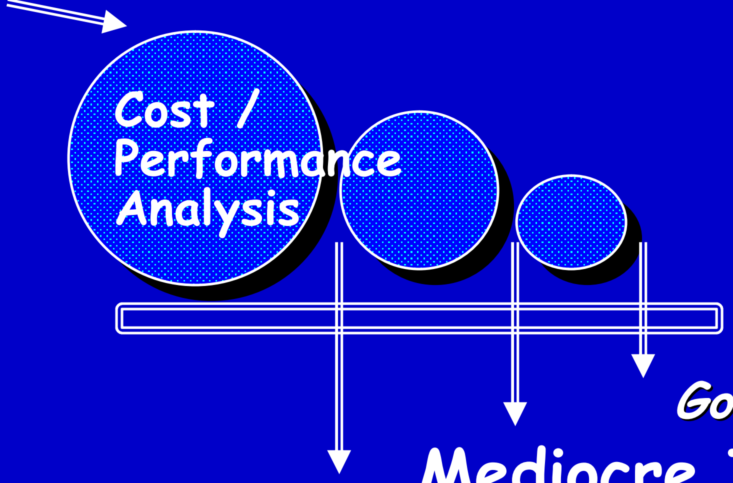


Architecture is an iterative process:

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



Creativity



Cost / Performance Analysis

Good Ideas

Mediocre Ideas

Bad Ideas

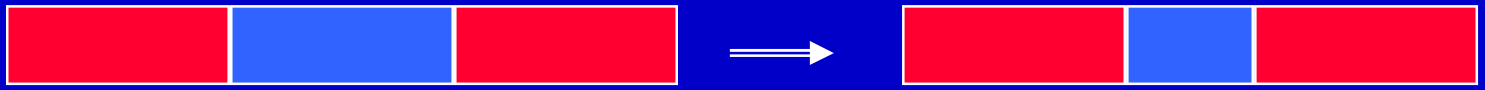
Amdahl's Law

$$\text{ExTime}_{\text{new}} = \text{ExTime}_{\text{old}} \times \left[\left(1 - \text{Fraction}_{\text{enhanced}}\right) + \frac{\text{Fraction}_{\text{enhanced}}}{\text{Speedup}_{\text{enhanced}}} \right]$$

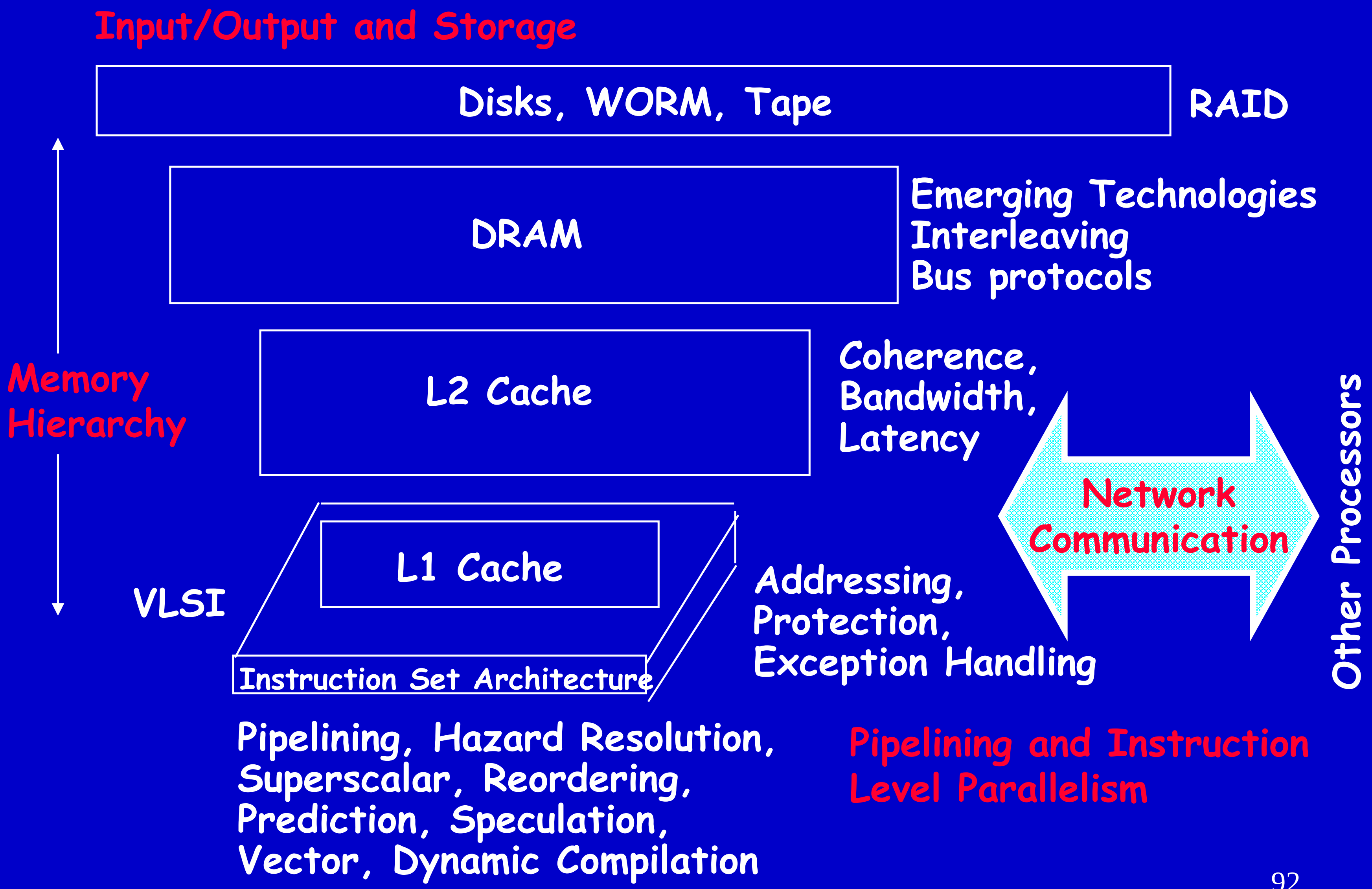
$$\text{Speedup}_{\text{overall}} = \frac{\text{ExTime}_{\text{old}}}{\text{ExTime}_{\text{new}}} = \frac{1}{\left(1 - \text{Fraction}_{\text{enhanced}}\right) + \frac{\text{Fraction}_{\text{enhanced}}}{\text{Speedup}_{\text{enhanced}}}}$$

Best you could ever hope to do:

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



Computer Architecture Topics



Input/Output and Storage

Disks, WORM, Tape

RAID

DRAM

Emerging Technologies
Interleaving
Bus protocols

Memory
Hierarchy

L2 Cache

Coherence,
Bandwidth,
Latency

L1 Cache

VLSI

Instruction Set Architecture

Addressing,
Protection,
Exception Handling

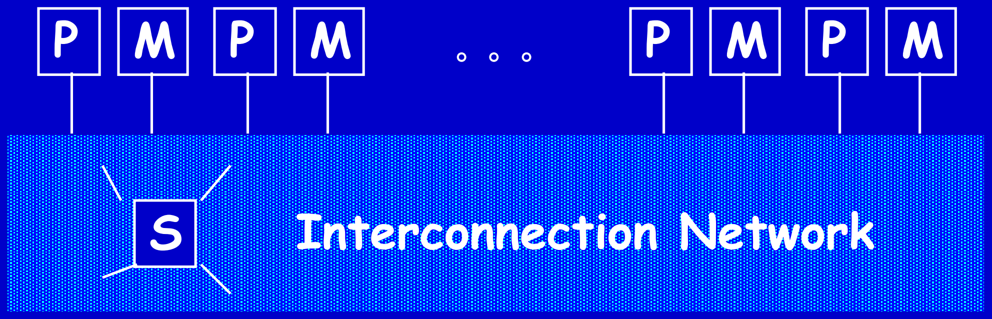
Network
Communication

Other Processors

Pipelining, Hazard Resolution,
Superscalar, Reordering,
Prediction, Speculation,
Vector, Dynamic Compilation

Pipelining and Instruction
Level Parallelism

Computer Architecture Topics



Processor-Memory-Switch

Shared Memory,
Message Passing,
Data Parallelism

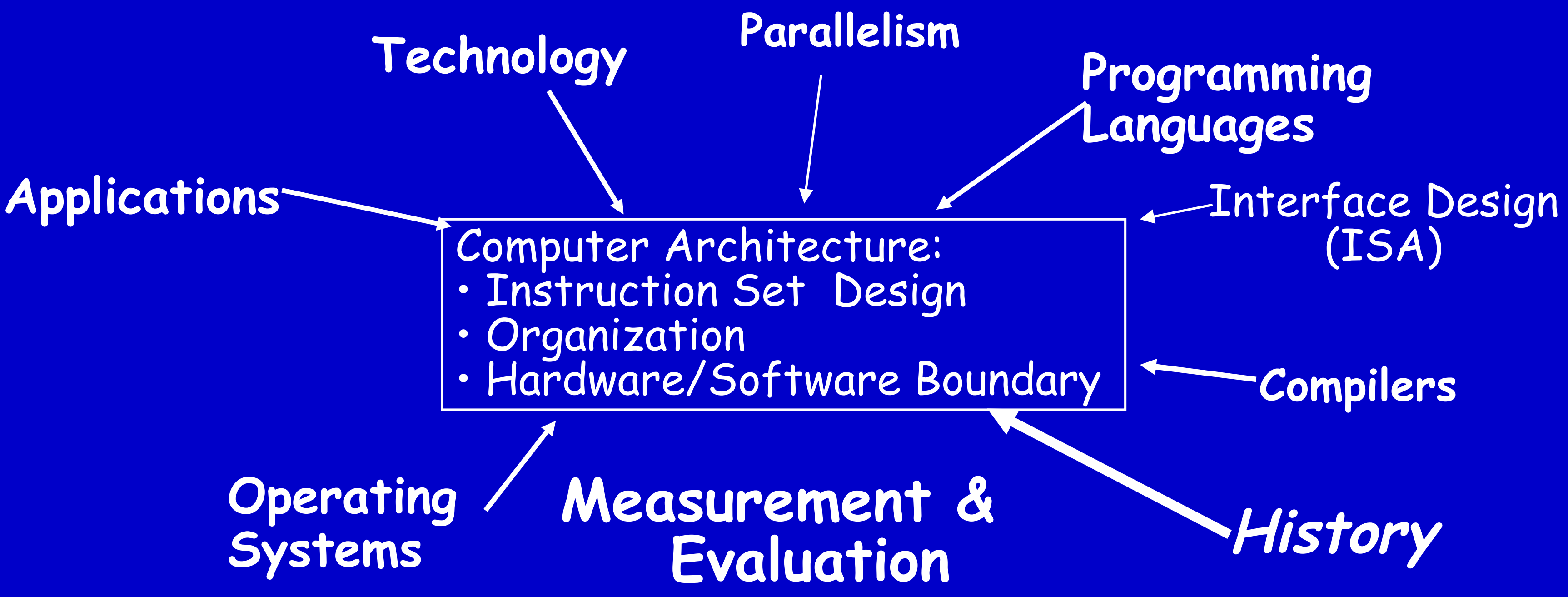
Network Interfaces

Topologies,
Routing,
Bandwidth,
Latency,
Reliability

Multiprocessors
Networks and Interconnections

Course Focus

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



Technology

Parallelism

Programming Languages

Applications

Interface Design (ISA)

Computer Architecture:
- Instruction Set Design
- Organization
- Hardware/Software Boundary

Compilers

Operating Systems

Measurement & Evaluation

History