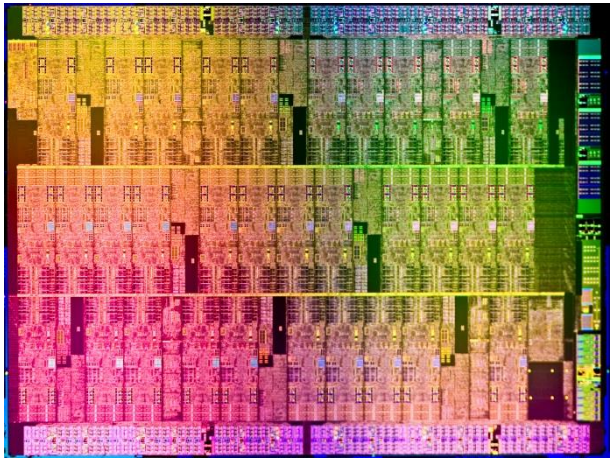
# Computer Organization and Design
## ECE 452 (Spring 2015)

# Introduction




**Sudeep Pasricha**

(http://www.engr.colostate.edu/~sudeep/)

# Course Info: Who Am I?

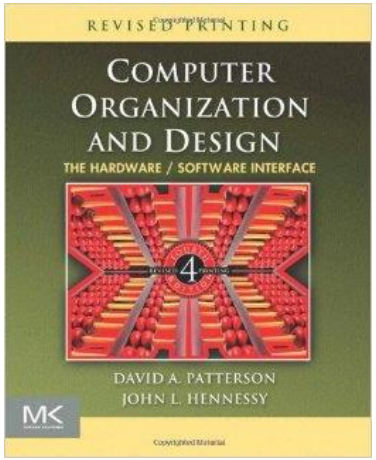
- Instructor: Prof. Sudeep Pasricha
  - Director, MECS Lab @ CSU
  - Several years of work experience in embedded and computer system design companies
    - Conexant, STMicroelectronics
  - My research interests:
    - Embedded systems and computer architecture
    - Mobile computing
    - High performance computing (supercomputing)
    - Design for energy-efficiency and reliability
    - Interconnection network and memory subsystem design
    - VLSI CAD (chip design, scheduling and optimization theory)
    - Emerging technologies (photonic, nanotube, 3D ICs)
  - Research funding
    - Government agencies: NSF, DOE, AFOSR (your tax $$$ at work)
  - Industry funding and equipment grants
    - SRC (Intel, IBM, etc), Xilinx/Altera, Microsoft, ARM, AMD, RIM/Blackberry, …

# Advising

- Currently I am an advisor for:
  - 9 PhD students
  - 10 MS students
  - 8 undergraduate students
    - 2 Honors thesis
    - 2 senior design projects
- Looking for motivated undergraduate students for several ongoing projects:
  - Brain controlled smart home
  - Games/solutions for limb rehabilitation
  - Ecocar3 automated driver assistance system design
  - Multi-core processor chip design and exploration
  - Smartphone software/middleware design and optimization
  - HPC/supercomputer analysis and resource management

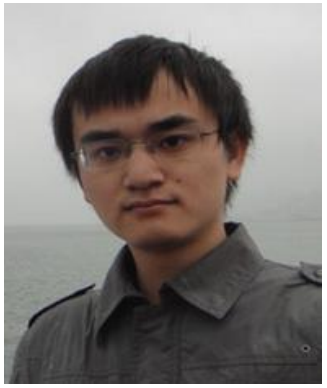
# Course Logistics

- Instructor
  - Sudeep Pasricha (sudeep@colostate.edu)
  - Office Hours: Fri: 3:00-5:00 pm (ENGR C103A)
    - or by appointment (send email)
- Course Page
  - http://www.engr.colostate.edu/~sudeep/teaching/ece_452.htm
  - Visit Canvas and course page regularly
    - for homework updates and announcements
- Class Meets
  - Tu/Thu 11:00-12:15 pm in Beh. Sci. Bldg 107
- Textbook (Required)
  - David A. Patterson and John L. Hennessy, *Computer Organization and Design: The Hardware/Software Interface*, **4th Revised Edition**, Morgan Kaufmann, 2011
  - New versions also ok



# Course Grader

- Yi Xiang
  - yix@colostate.edu
  - Email him if you have questions on assignments or course content
    - Note: he will not help you solve your assignment questions



# What You Should Know

- Prerequisite: ECE 251 (or equivalent)
  - Intro to Microprocessors
- Basic digital logic design (Appendix C)
  - FSM, synchronous design
- Basic structure of a microprocessor
  - including memory subsystem, I/O
- Some experience with assembly language and C programming, debugging

# Evaluation and Grading

- Homework Assignments (6): 30%
  - All HW submissions via Canvas
    - Grades/scores will also be posted on Canvas
- Class Participation: 10%
- Examinations (closed book/notes): 60%
  - Midterm: 20%
  - Quizzes (1): 10%
  - Comprehensive Final: 30%
- Grading scale

| | | | |
|---|---|---|---|
| *>95% A+* | *80-84% B+* | *65-69% C+* | *<40% F* |
| *90-94% A* | *75-79% B* | *55-64% C* | |
| *85-89% A-* | *70-74% B-* | *40-55% D* | |

# Evaluation and Grading

- **Homework Policy**
  - No late submissions accepted unless you have a valid excuse
- **Attendance**
  - Your responsibility to keep track of what you missed if absent
  - Keep track of assignments and due dates
- **Academic Honesty**
  - All submitted work should be your own. Plagiarism/cheating will result in all students involved being severely penalized
- **Appointment**
  - I encourage you to make at least one appointment with me during the semester for advice or to discuss research opportunities, research ideas, course suggestions, concerns etc.

# Cheating and Plagiarism

- **What is cheating & plagiarism?**
  - Acting dishonestly, practicing fraud
  - copying of language, structure, images, **ideas**, or thoughts of another, and representing them as one's own without proper acknowledgement (from web sites, books, papers, other students, solutions from previous offerings of this course, etc);
- **My policy: zero tolerance**
  - Minor first infraction in HWs:
    - 0 score + one letter level (e.g. A to B) reduction in course grade
  - Project or Major or repeated infractions in HWs, presentations:
    - "F" grade for the course + report to Dean's Office
- Moreover, remember that you may have to face me in other exams (e.g. Honors thesis, M.S. project, Ph.D. prelims/qualifiers) and professionally!
  - Bottom-line: don't risk engaging in such behavior.
  - For more information:
    - CSU's Academic Integrity Policy: http://tilt.colostate.edu/integrity/
    - Student Conduct Code: http://www.conflictresolution.colostate.edu/conduct-code

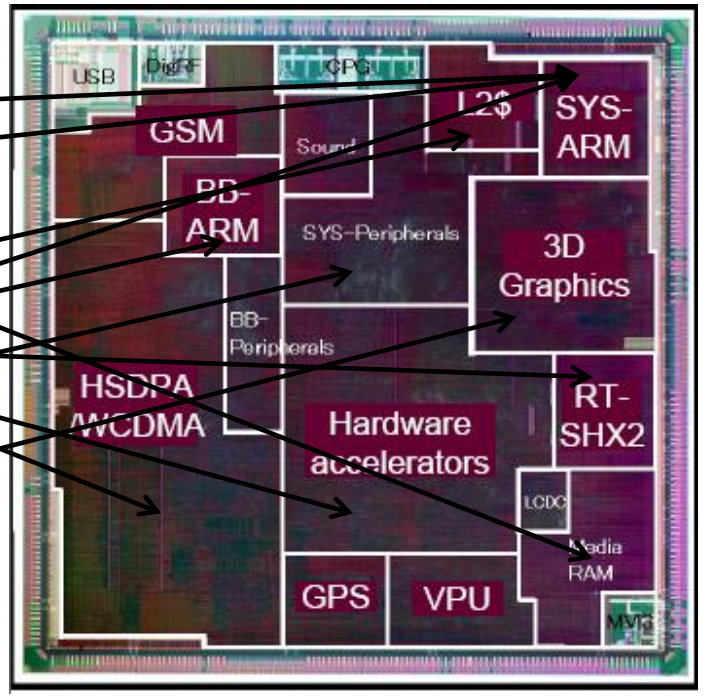
# Why study Computer Architecture?

- **Understand where computers are going**
  - Future capabilities drive the (computing) world
  - Real world-impact: no computer architecture → no computers!
- **Understand high-level design concepts**
  - The best system designers understand all the levels
  - Hardware, compiler, operating system, applications
- **Understand computer performance**
  - Writing well-tuned (fast) software requires knowledge of hardware
- **Write better software**
  - The best software designers also understand hardware
  - Understand the underlying hardware and its limitations
- **Design hardware**
  - Intel, AMD, IBM, ARM, Microsoft, Google, Qualcomm, Broadcom, Freescale, TI, Apple, Oracle, NVIDIA, Samsung, Avago, LSI, GM…

# Scope of Course

**Major Topics**

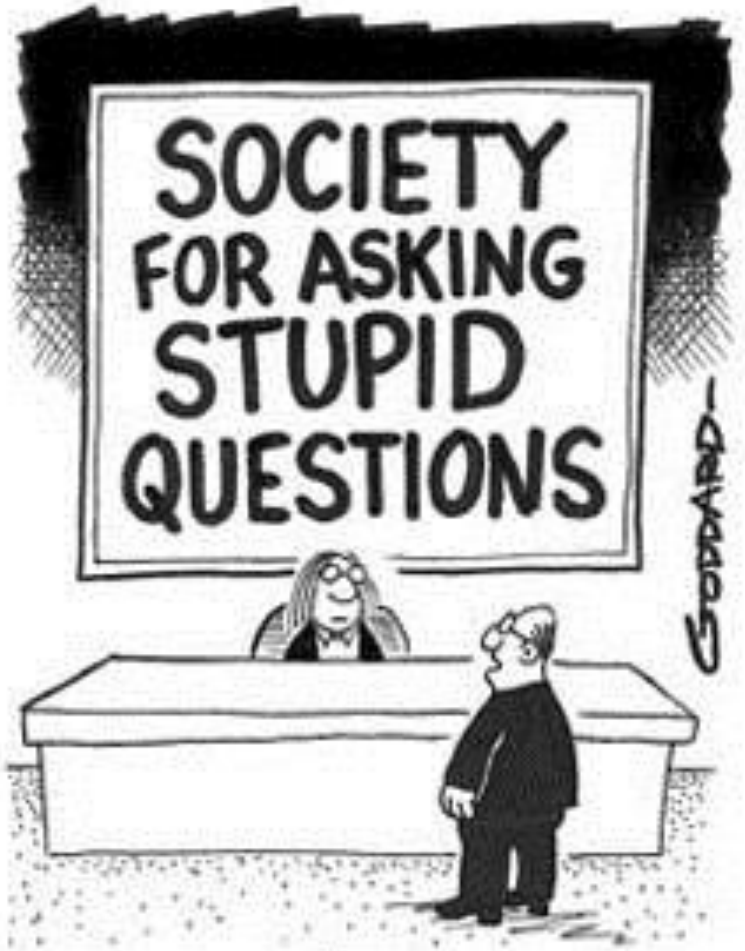
- Instruction Set Architectures (MIPS)
- Processor Design
- Memory Hierarchy, Storage and I/O
- Multicores and multiprocessors
- Interconnection Networks
- Graphics and Computing GPUs




- **Primary goals**
  - Understand key hardware concepts
    - Pipelining, parallelism, caching, locality, abstraction, etc.
    - Use ‘mobile computing’ as an exemplar for many concepts
  - Understand interface between software and hardware
  - A bit of scientific/experimental exposure and/or analysis
- **My role:**
  - **Trick you into learning something**

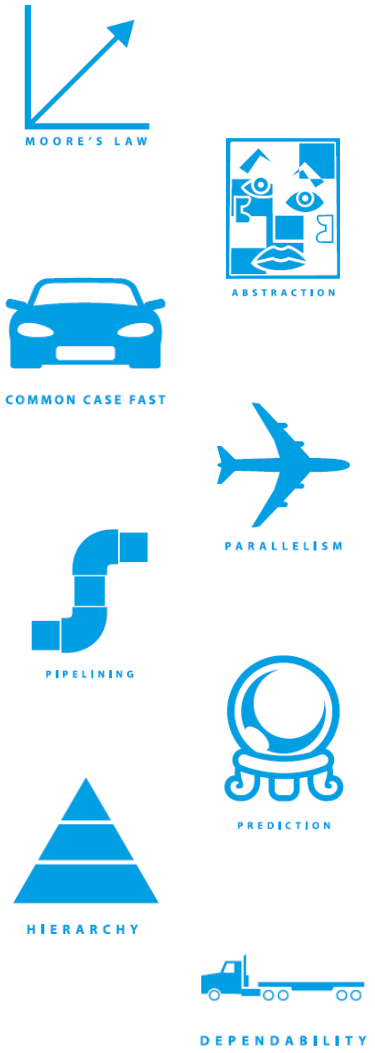
# A Final Expectation …

- Asking questions in class is the best way to clear doubts immediately
  - Don’t be shy!



# Eight Great Ideas in Computer Architecture

- Design for ***Moore's Law***
- Use ***abstraction*** to simplify design
- Make the ***common case fast***
- Performance *via* ***parallelism***
- Performance *via* ***pipelining***
- Performance *via* ***prediction***
- ***Hierarchy*** of memories
- ***Dependability*** *via* redundancy

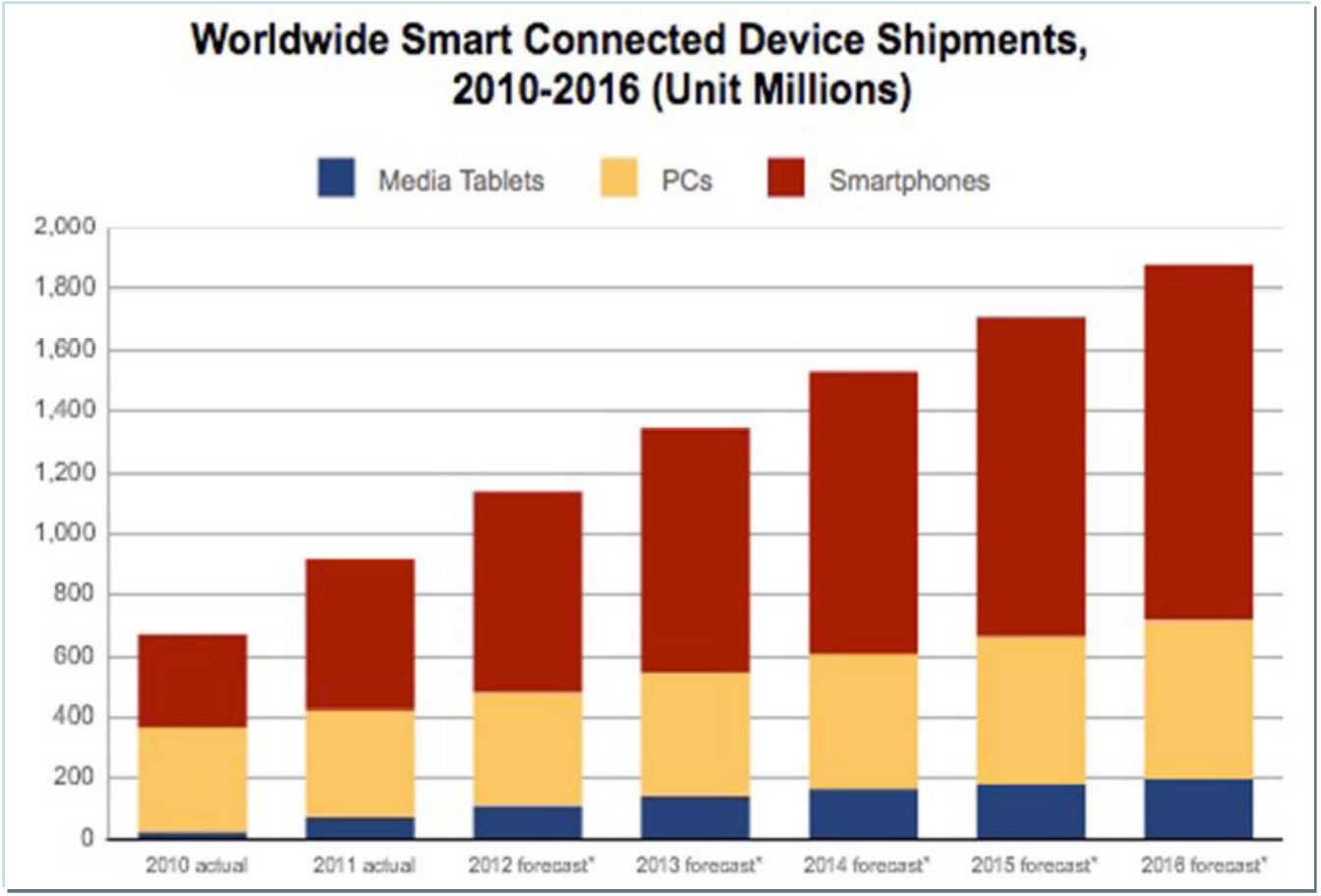


# Computers today are “embedded” everywhere!

- **According to forecasts, future of IT characterized by terms such as**
  - Disappearing computer,
  - Ubiquitous computing,
  - Pervasive computing,
  - Ambient intelligence,
  - Post-PC era,
  - Cyber-physical systems.


# The Post PC Era

embedded growth >> desktop/laptop growth



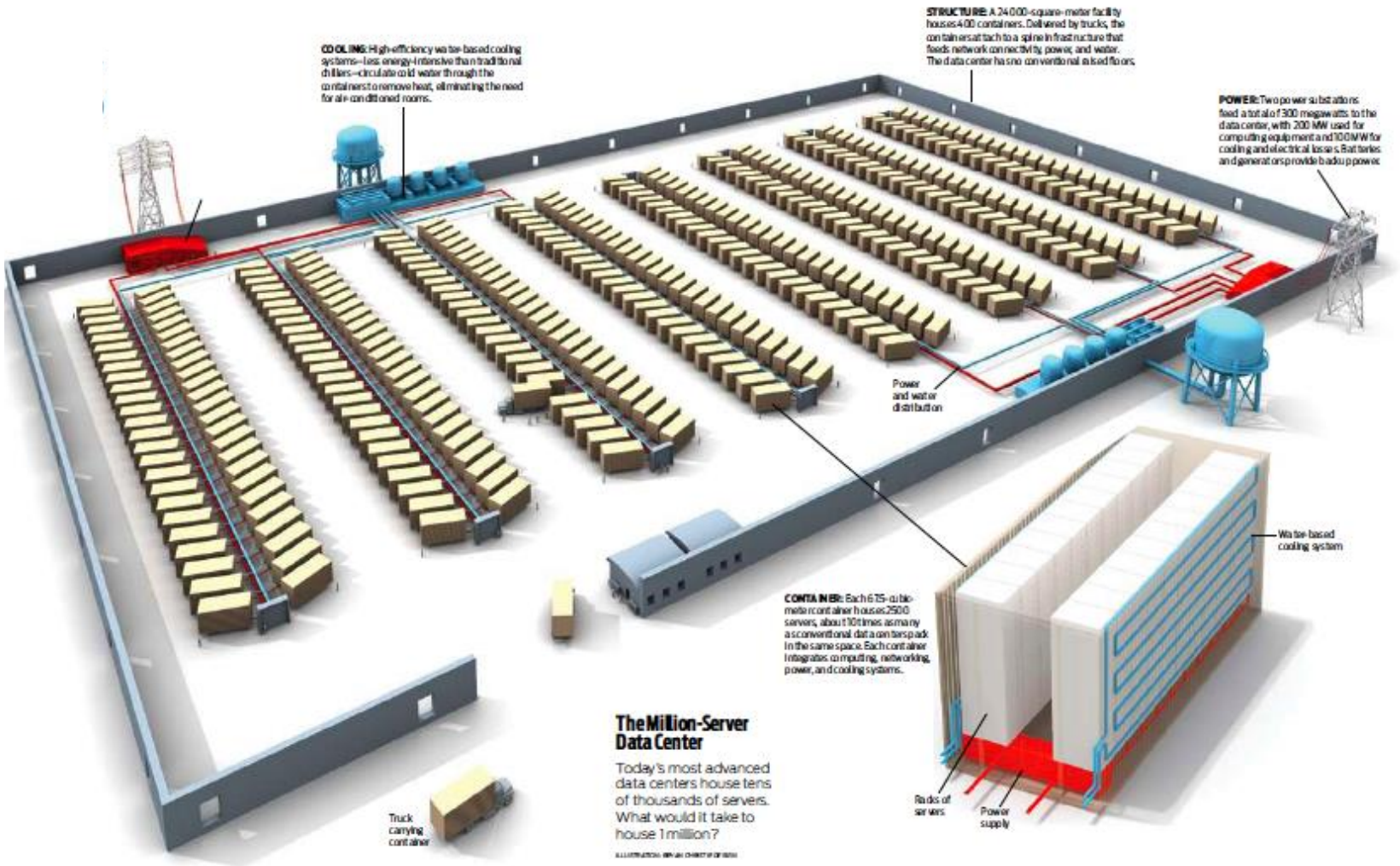
# Mobile and Cloud Computing

- Personal Mobile Device (PMD)
  - A sub-class of “embedded systems”
  - Battery operated
  - Connects to the Internet
  - Hundreds of dollars
  - Smart phones, tablets, electronic glasses
- Cloud computing
  - Datacenter computing
  - Software as a Service (SaaS)
  - Portion of software run on a PMD and a portion run in the Cloud
  - Amazon, Google, Facebook, Microsoft, …

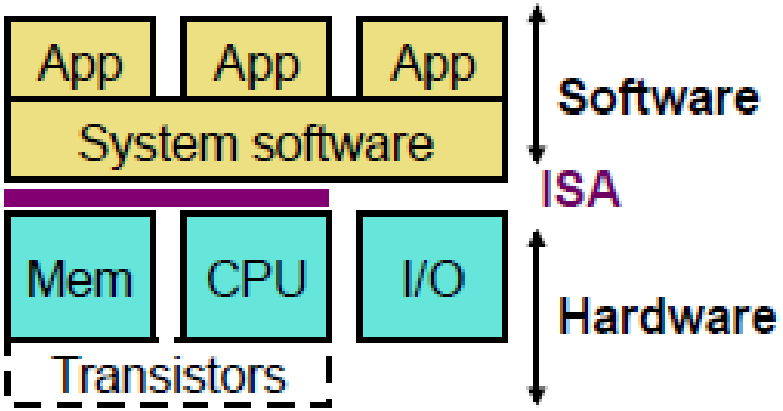
# Mobile Computing is Evolving

# Cloud Computing is Evolving

- Warehouse Scale Computers (WSC)



# Abstraction, Layering, and Computers




- Computers are complex, built in layers
  - Several **software** layers: assembler, compiler, OS, applications
  - **Instruction set architecture (ISA)**
  - Several **hardware** layers: transistors, gates, CPU/Memory/IO
- 99% of users don't know hardware layers implementation
- 90% of users don't know implementation of any layer
  - That's okay, world still works just fine
  - As a designer/engineer/scientist it is important to understand what's "under the hood"

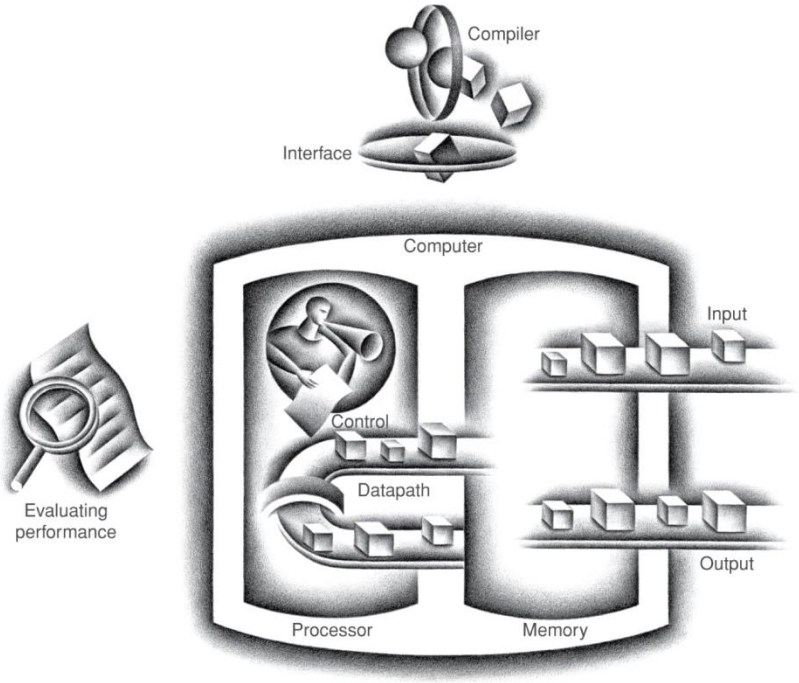
# Abstraction and Layering

Build computer bottom up by raising level of abstraction

- Hardware
  - Solid-state semi-conductor materials → transistors
  - Transistors → gates
  - Gates → digital logic elements: latches, muxes, adders
  - Logic elements → datapath + control = processor
    - Key insight: stored program (instructions just another form of data)
    - Another one: few insns can be combined to do anything (software)
- Software
  - Machine language → assembly language
  - Assembly language → high-level language
  - Code → graphical user interface

# "Under the Hood" of a Computer

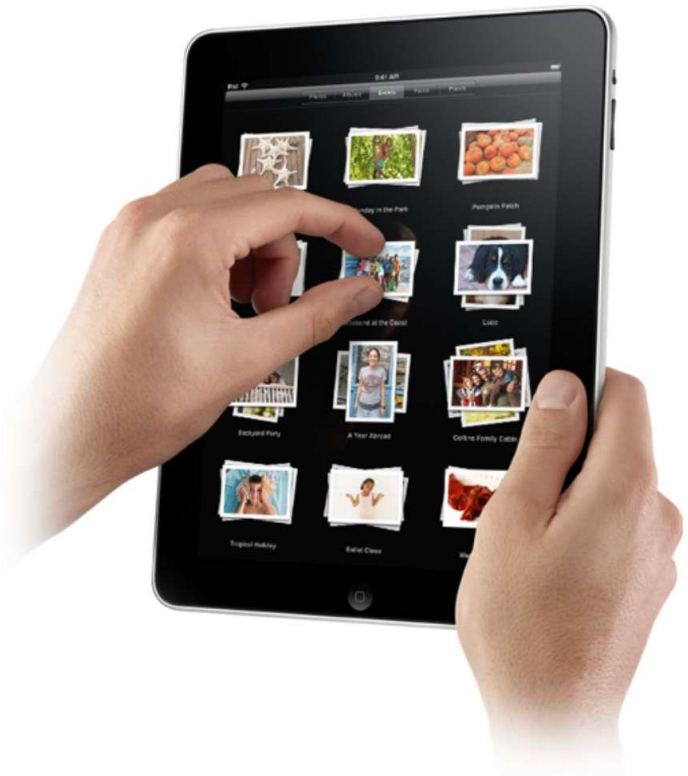
**The BIG Picture**




- Same components for all kinds of computer
  - Desktop, server, embedded
- Input/output includes
  - User-interface devices
    - Display, keyboard, mouse
  - Storage devices
    - Hard disk, CD/DVD, flash
  - Network adapters
    - For communicating with other computers

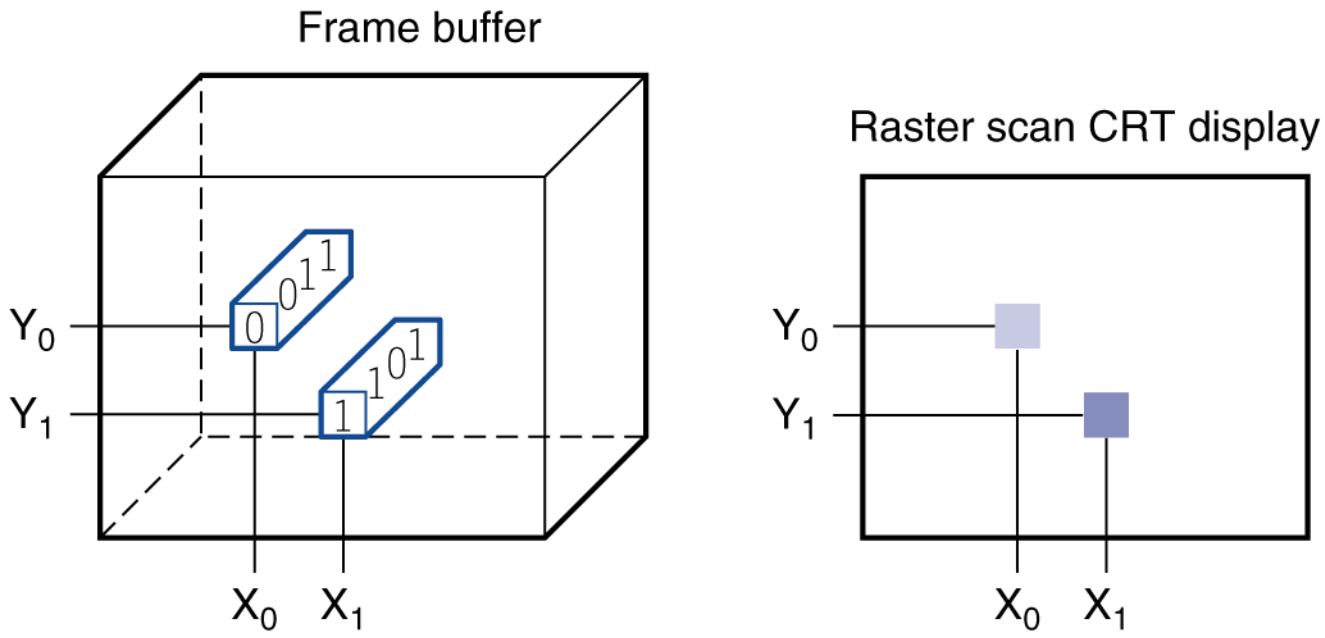
# Touchscreen

- PostPC device
- Supersedes keyboard and mouse
- Resistive and Capacitive types
  - Most tablets, smart phones use capacitive
  - Capacitive allows multiple touches simultaneously

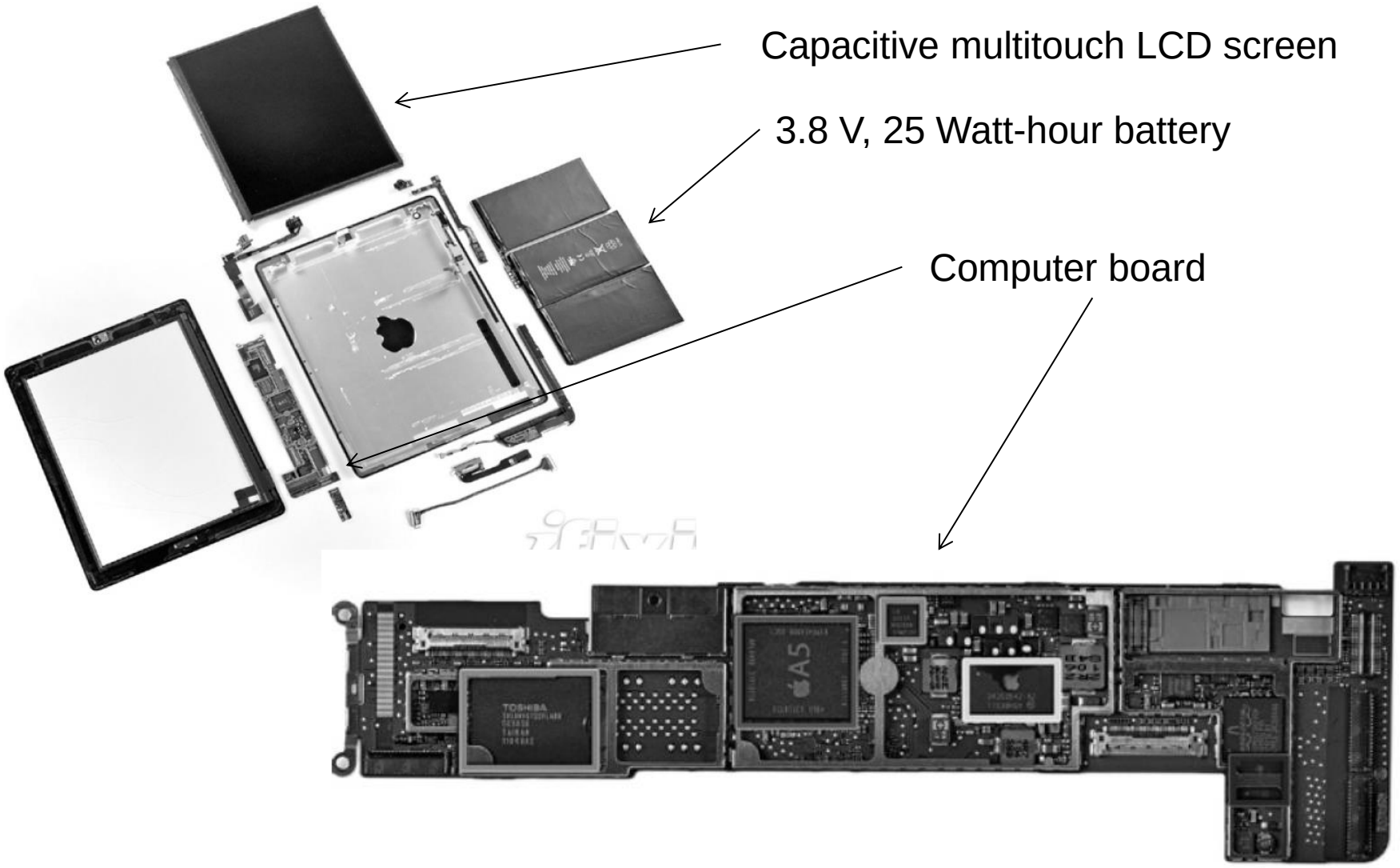


# Through the Looking Glass

- LCD/OLED screen: picture elements (pixels)
  - Mirrors content of frame buffer memory



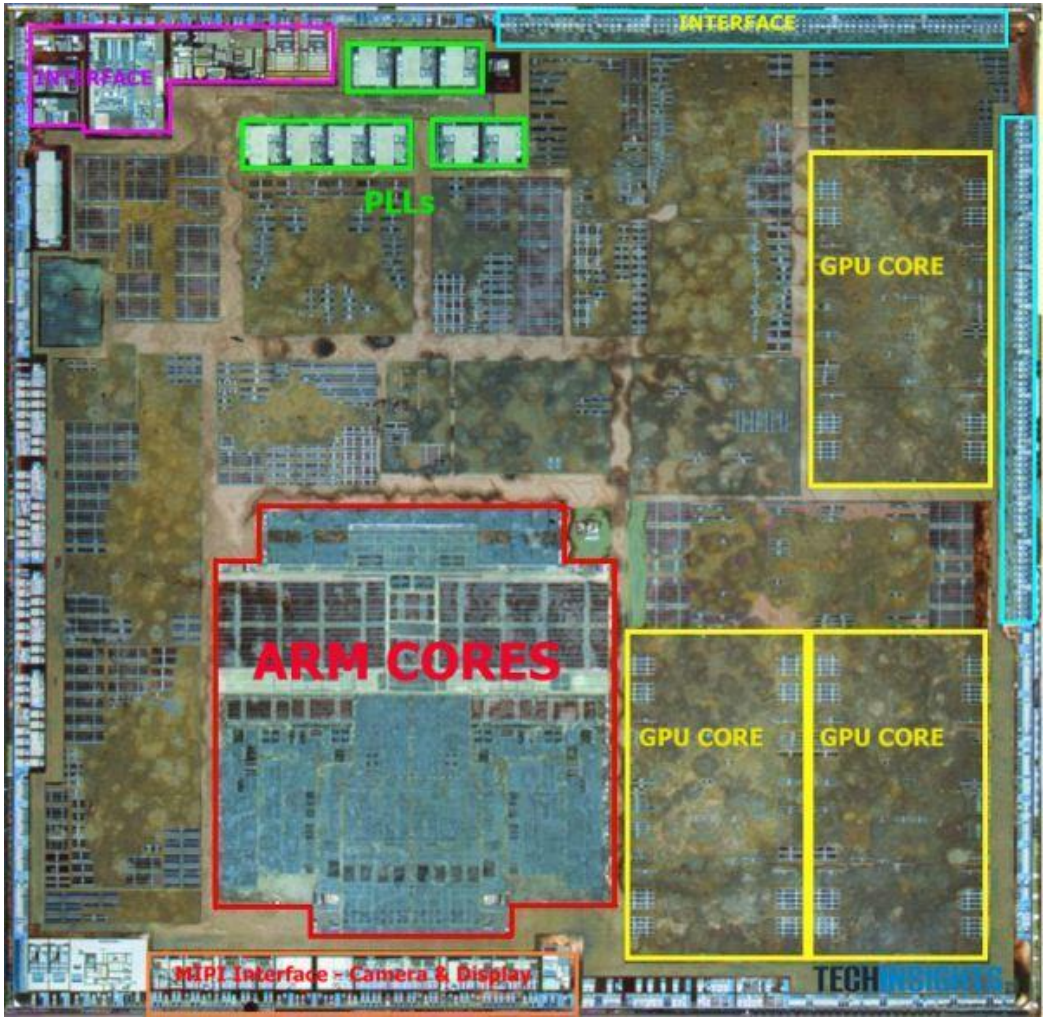
# Opening the Box: iPad



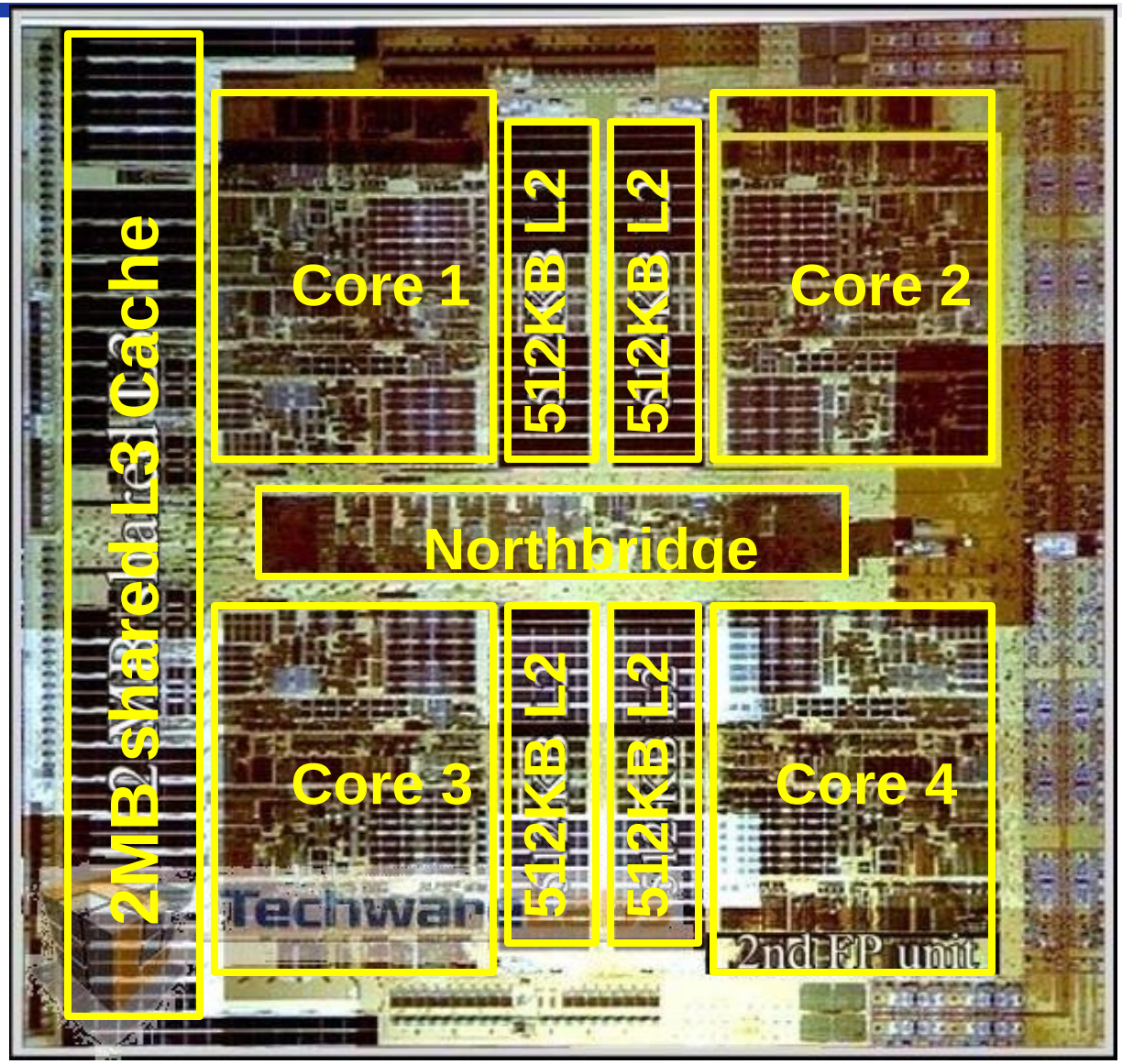
# Inside the Processor (iPad/iPhone)

## Apple A6

- 1.3 GHz custom Apple-designed ARMv7 based dual-core CPU
- integrated triple-core PowerVR SGX 543MP3 GPU
- A6 is in iPhone 5
- A6x with higher frequency and 4 graphic cores is found in the 4th generation iPad
- A7 in iPad Air, iPhone 5s
  - 64-bit CPU, M7 motion-coprocessor
- A8 in iPhone6



# Inside the Processor (PCs)




- AMD's Barcelona Multicore Chip
- Four out-of-order cores on one chip
- 1.9 GHz clock rate
- 65nm technology
- Three levels of caches (L1, L2, L3) on chip
- Integrated Northbridge

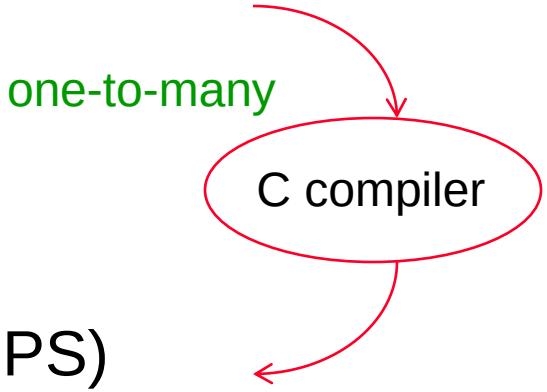
# Inside the Processor

- Datapath
  - performs operations on data
- Control
  - sequences datapath, memory, ...
- Cache memory
  - Small fast SRAM memory for immediate access to data

# Levels of (Software) Program Code

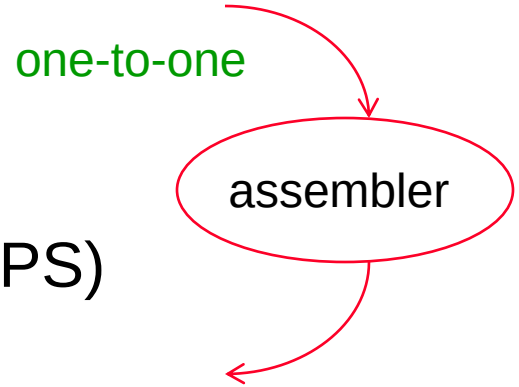
- High-level language program (in C)

```
void swap (int v[], int k)
{       int temp;
        temp = v[k];
        v[k] = v[k+1];
        v[k+1] = temp;
}
```




- Assembly language program (for MIPS)

```
swap:   sll     $2, $5, 2
        add     $2, $4, $2
        lw      $15, 0($2)
        lw      $16, 4($2)
        sw      $16, 0($2)
        sw      $15, 4($2)
        jr      $31
```

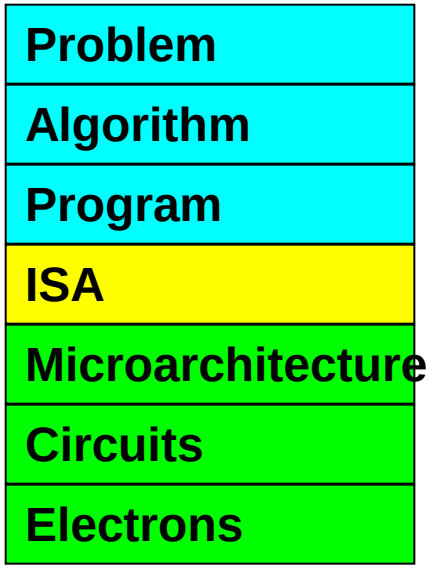



- Machine (object, binary) code (for MIPS)

```
000000 00000 00101 0001000010000000
000000 00100 00010 0001000000100000
. . .
```

# Instruction Set Architecture

- ISA
  - Agreed upon interface between software and hardware
    - SW/compiler assumes, HW promises
  - What the software writer needs to know to write system/user programs
- Microarchitecture
  - Specific implementation of an ISA
  - Not visible to the software
- Microprocessor
  - ISA, uarch, circuits
  - "Architecture" = ISA + microarchitecture

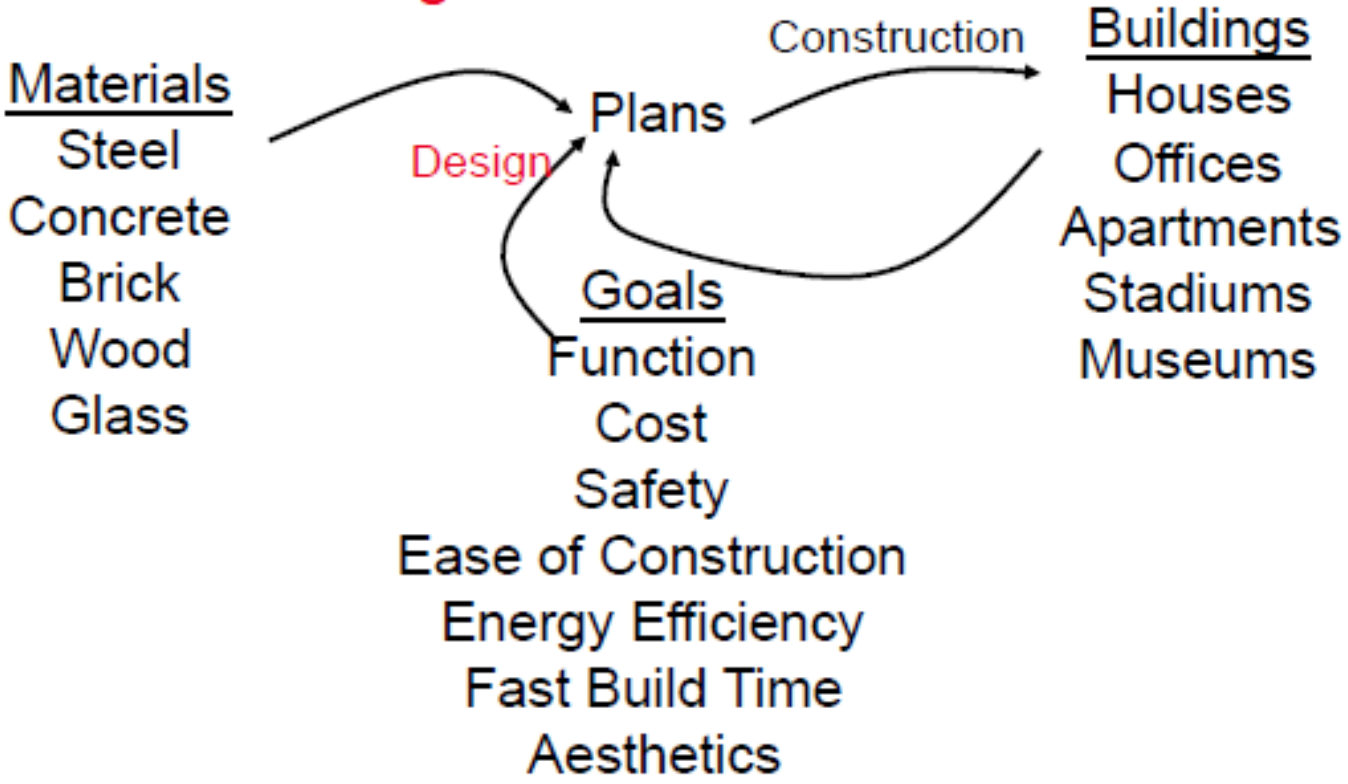


# What is Computer Architecture?

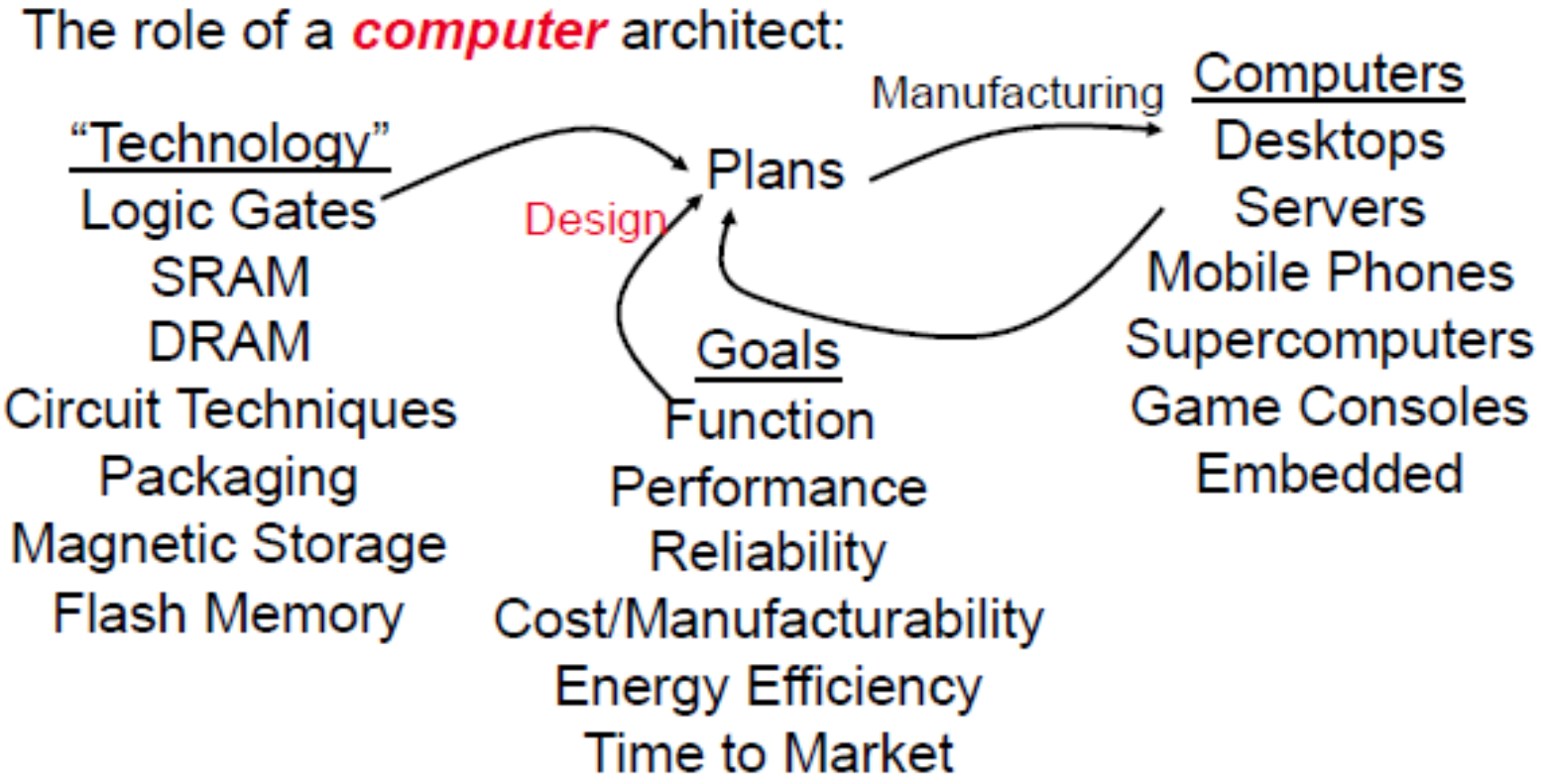
- *"Computer Architecture* is the science and art of selecting and interconnecting hardware components to create computers that meet functional, performance, power, and cost goals."

# What is ~~Computer~~ Architecture?

The role of a ***building*** architect:



# What is Computer Architecture?



# Computer Architecture Is Different…

- Age of discipline
  - 60 years (vs. five thousand years)
- Rate of change
  - All three factors (technology, applications, goals) are changing
  - Quickly
- Automated mass production
  - Design advances magnified over millions of chips
- Boot-strapping effect
  - Better computers help design next generation

# Design Goals and Constraints

- **Functional**
  - What functions should the design support?
  - Needs to be correct
  - Unlike software, hardware is difficult to update once deployed
- **Reliable**
  - Does it **continue** to perform correctly?
  - Hard fault vs transient fault
  - Google story - memory errors and sun spots
  - Space satellites vs desktop vs server reliability
- **High performance**
  - “Fast” is only meaningful in the context of a set of important tasks
  - Not just “Gigahertz”
  - Impossible goal: fastest possible design for all programs

# Design Goals and Constraints

- **Low cost**
  - Per unit manufacturing cost (wafer cost)
  - Cost of making first chip after design (mask cost)
  - Design cost (huge design teams)
- **Low power/energy**
  - Energy in (battery life, cost of electricity)
  - Energy out (cooling and related costs)
  - Cyclic problem, very much a problem today

- **Challenge: balancing relative importance of these goals**
  - And the balance is constantly changing
  - No goal is absolutely important at expense of all others
  - Our focus: performance; and touch on power, cost, reliability

# Understanding Performance

- Suppose you want to design a game console to play games with excellent performance. What factors impact the performance of this system?

- Algorithm
  - Determines number of operations executed
- Programming language, compiler, ISA
  - Determine number of machine instructions executed per operation
- Processor and memory system
  - Determine how fast instructions are executed
- I/O system (including OS)
  - Determines how fast I/O operations are executed

# Design Goals: Applications/Domains

- Another shaping force: **applications** (usage and context)
  - Applications and application domains have different requirements
    - Domain: group with similar character
  - Lead to different designs
- **Scientific**: weather prediction, genome sequencing
  - First computing application domain: naval ballistics firing tables
  - Need: large memory, heavy-duty floating point
  - Examples: CRAY T3E, IBM BlueGene
- **Commercial**: database/web serving, e-commerce, Google
  - Need: data movement, high memory + I/O bandwidth
  - Examples: Sun Enterprise Server, AMD Opteron, Intel Xeon

# More Recent Applications/Domains

- **Desktop**: home office, multimedia, games
  - Need: integer, memory bandwidth, integrated graphics/network?
  - Examples: Intel Core 2, Core i7, AMD Athlon
- **Mobile**: laptops, tablets, mobile phones
  - Need: **low power**, integer performance, integrated wireless
  - Laptops/tablets: Intel Core 2 Mobile, Atom, AMD Turion
  - Smaller devices: ARM chips by Samsung, Qualcomm; Intel Atom
- **Embedded**: microcontrollers in automobiles, door knobs
  - Need: low power, **low cost**
  - Examples: ARM chips, dedicated digital signal processors (DSPs)
  - Over 6 billion ARM cores sold in 2010 (multiple per phone)
- **Deeply Embedded**: disposable “smart dust” sensors
  - Need: extremely low power, extremely low cost
  - Very small microcontrollers: 8086 derivatives, ARM M series

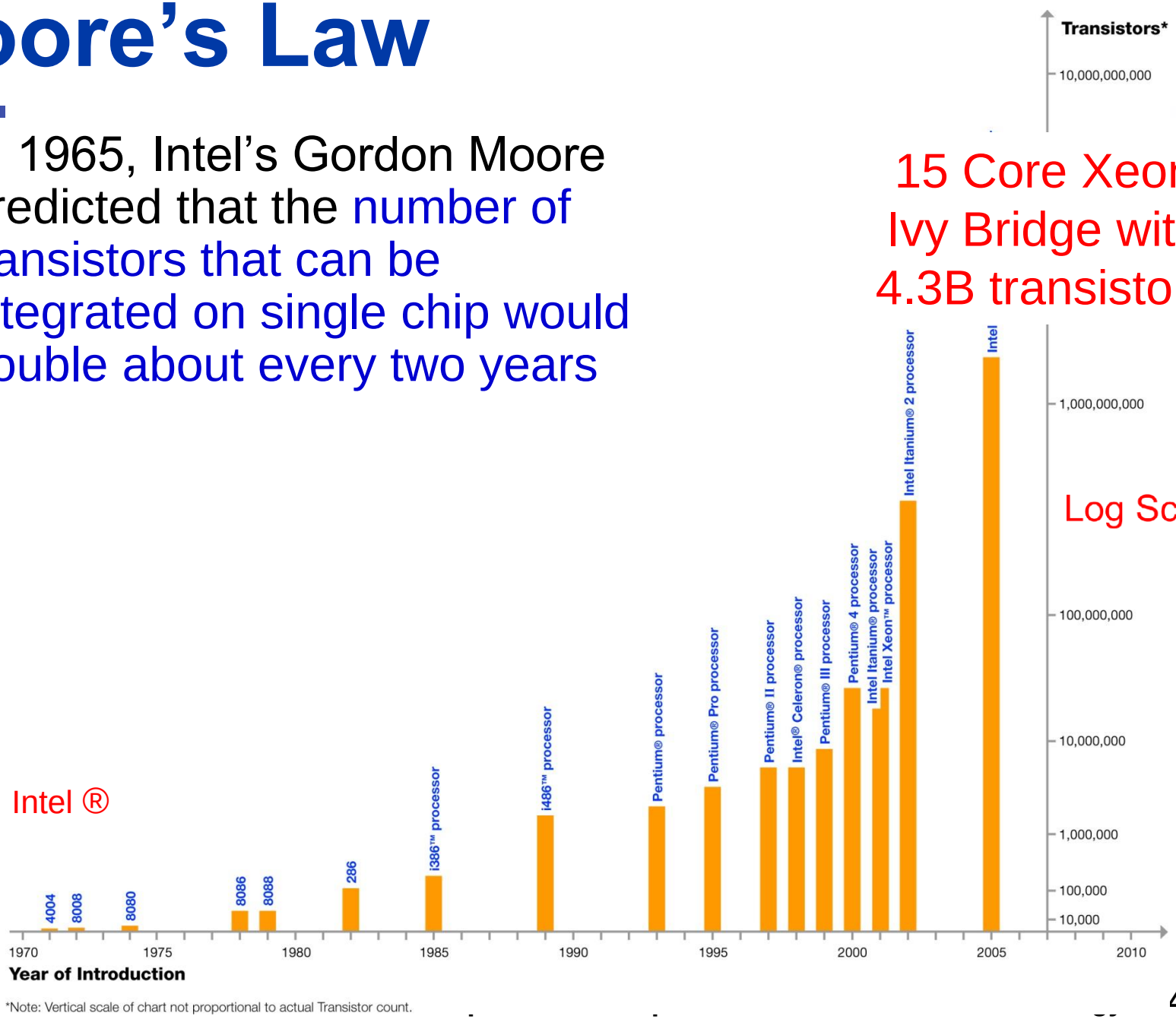
# Application Specific Designs

- This class is about **general-purpose CPUs**
  - Processor that can do anything, run a full OS, etc.
  - E.g., Intel Core i7, AMD Athlon, IBM Power, ARM, Intel Itanium
- In contrast to **application-specific chips**
  - Or **ASICs** (Application specific integrated circuits)
    - Also application-domain specific processors
  - Implement critical domain-specific functionality in hardware
    - Examples: video encoding, 3D graphics
  - General rules
    - - Hardware is less flexible than software
    - + Hardware more effective (speed, power, cost) than software
    - + Domain specific more “parallel” than general purpose
      - But general mainstream processors becoming more parallel
  - CS/ECE561: HW/SW Design of Embedded Systems looks into ASIC design (and general purpose design) in a lot more detail

# Moore's Law

- In 1965, Intel's Gordon Moore predicted that the number of transistors that can be integrated on single chip would double about every two years

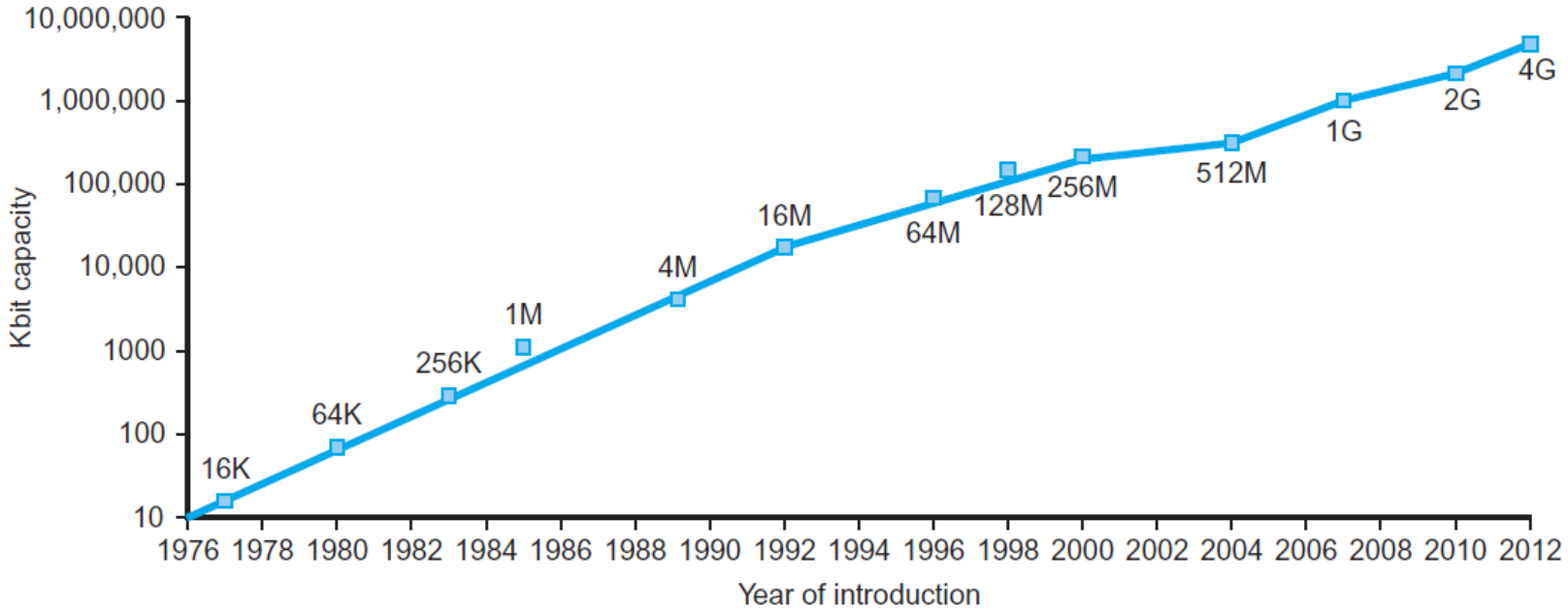
15 Core Xeon Ivy Bridge with 4.3B transistors




Courtesy, Intel ®

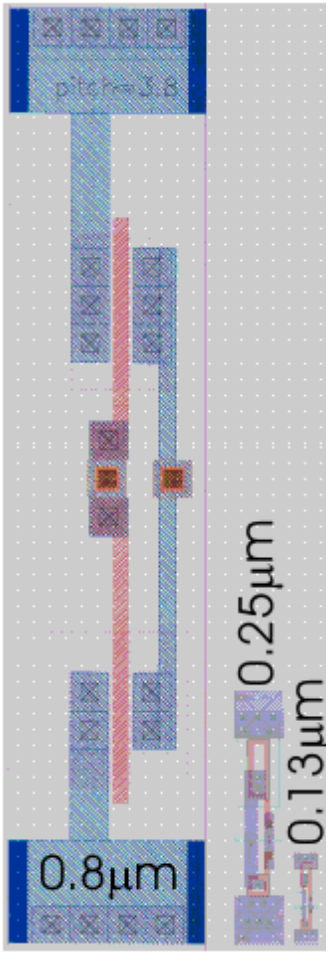
# Another Example of Moore's Law Impact

DRAM capacity growth over 3 decades



# Technology Scaling

- Key enabler for smaller, faster and more power efficient computing systems
- Technology scaling has a threefold objective:
  - Increase the transistor density
  - Reduce the gate delay
  - Reduce the power consumption
- Device dimensions (lateral and vertical) and voltages are reduced by $1/\alpha$ (~0.7)
- Technology generation spans 2-3 years



# Technology Scaling Roadmap (ITRS)

- Electronics technology continues to evolve
  - Increased capacity and performance; reduced cost

| Year | 2006 | 2008 | 2010 | 2012 | 2014 |
|---|---|---|---|---|---|
| Feature size (nm) | 90 | 65 | 45 | 32 | 22 |
| Intg. Capacity (BT) | 2 | 4 | 6 | 16 | 32 |

- Fun facts about 22nm transistors
  - 150 million can fit on the head of a pin
  - You could fit more than 10,000 across the width of a human hair
  - If car prices had fallen at the same rate as the price of a single transistor has since 1968, a new car today would cost about ?
  - 0.2 cent

# Trends due to Technology Scaling

- Computers get 10x faster, smaller, cheaper every 5-6 years!
  - A 10x quantitative change is qualitative change
  - Plane is 10x faster than car, and fundamentally different travel mode
- New applications become self-sustaining market segments
  - Recent examples: mobile phones, digital cameras, mp3 players, etc.
- Some technology-based ramifications
  - Absolute improvements in density, speed, power, costs
  - SRAM/logic: density: ~30% (annual), speed: ~20%
  - DRAM: density: ~60%, speed: ~4%
  - Disk: density: ~60%, speed: ~10% (non-transistor)
  - Big improvements in flash memory and network bandwidth, too
- **Changing quickly and with respect to each other!!**
  - Example: density increases faster than speed and power caps
  - Trade-offs are constantly changing
  - **Re-evaluate/re-design for each technology generation**

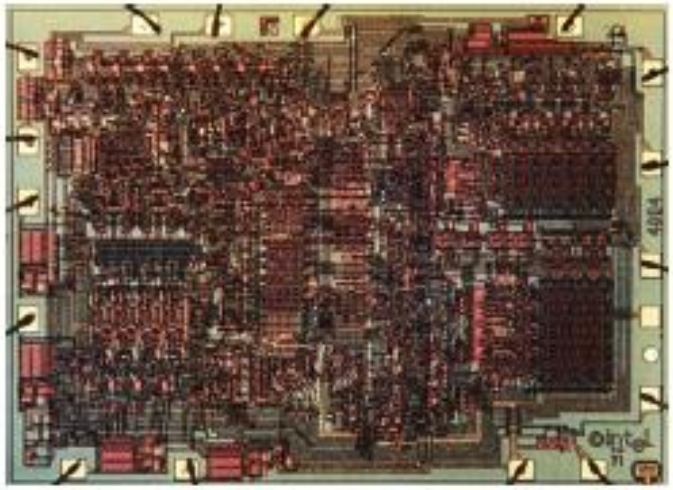
# History of Computing: a (very) brief perspective

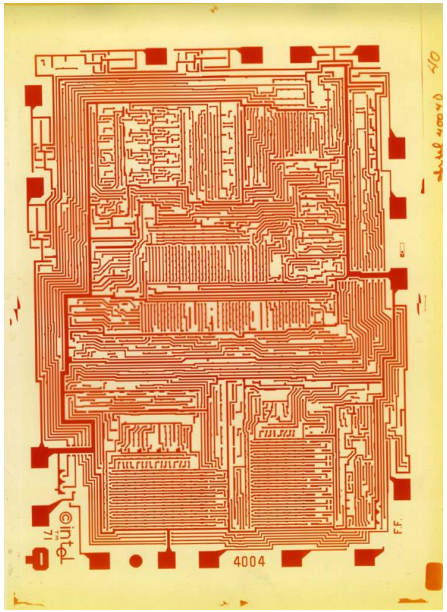
# Revolution I: The Microprocessor

- **Microprocessor revolution**
  - One significant technology threshold was crossed in 1970s
  - Enough transistors (~25K) to put a 16-bit processor on one chip
  - Huge performance advantages: fewer slow chip-crossings
  - Even bigger cost advantages: one “stamped-out” component
- Microprocessors have allowed new market segments
  - Desktops, CD/DVD players, laptops, game consoles, set-top boxes, mobile phones, digital camera, mp3 players, GPS, automotive
- And replaced incumbents in existing segments
  - Microprocessor-based system replaced supercomputers, “mainframes”, “minicomputers”, etc.

# First Microprocessor

- Intel 4004 (1971)
  - Application: calculators
  - Technology: 10000 nm
  - 2300 transistors
  - 13 $mm^2$
  - 108 KHz
  - 12 Volts
  - 4-bit data
  - Single-cycle datapath






Metal layout

# Revolution II: Implicit Parallelism

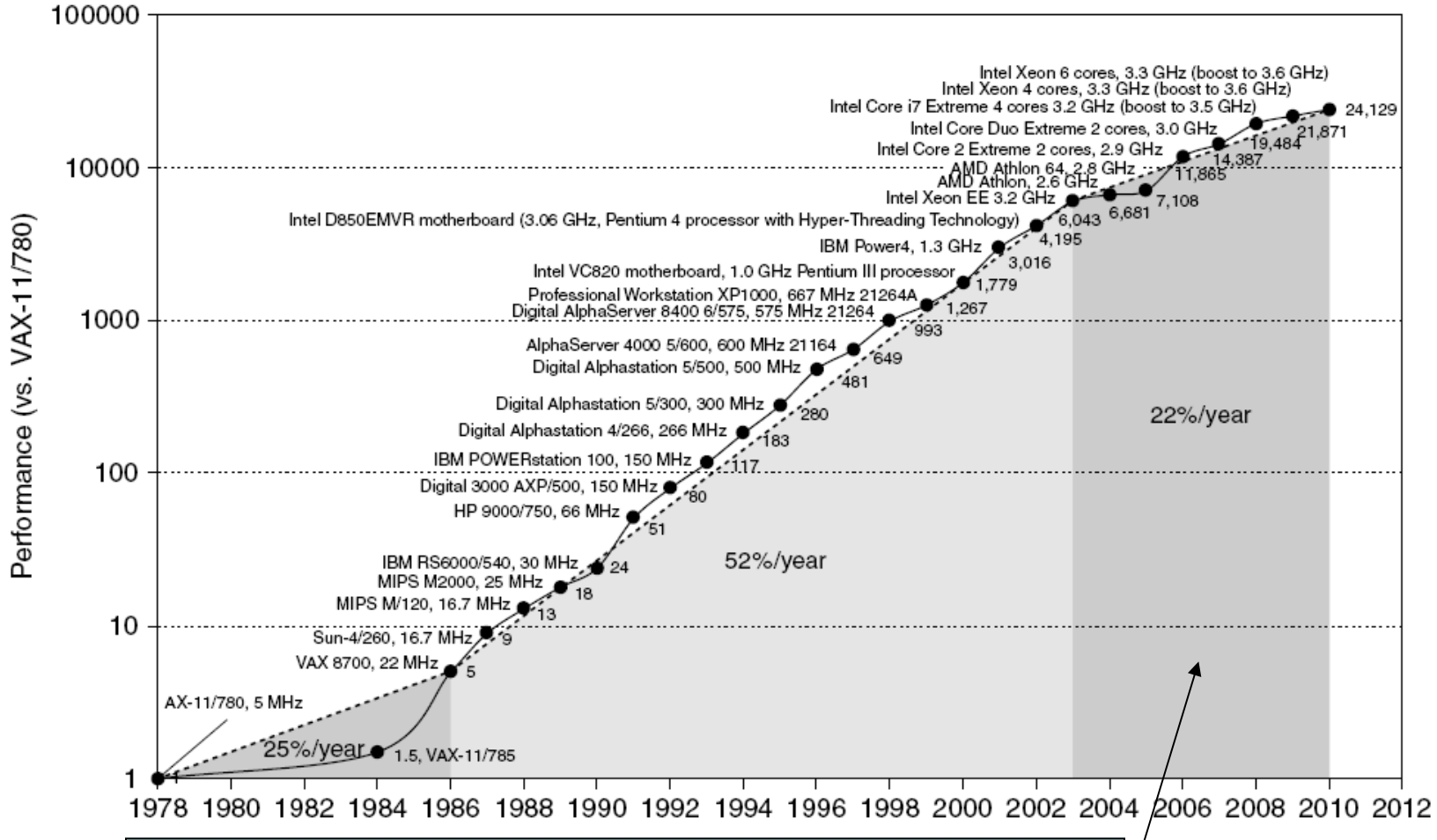
- Then to **extract implicit instruction-level parallelism**
  - Hardware provides parallel resources, figures out how to use them
  - Software is oblivious
- Initially using pipelining …
  - Which also enabled increased clock frequency
- … caches …
  - Which became necessary as processor clock frequency increased
- … and integrated floating-point
- Then deeper pipelines and branch speculation
- Then multiple instructions per cycle (superscalar)
- Then dynamic scheduling (out-of-order execution)

- **We will talk about these things**

# Pinnacle of ILP-driven Microprocessors

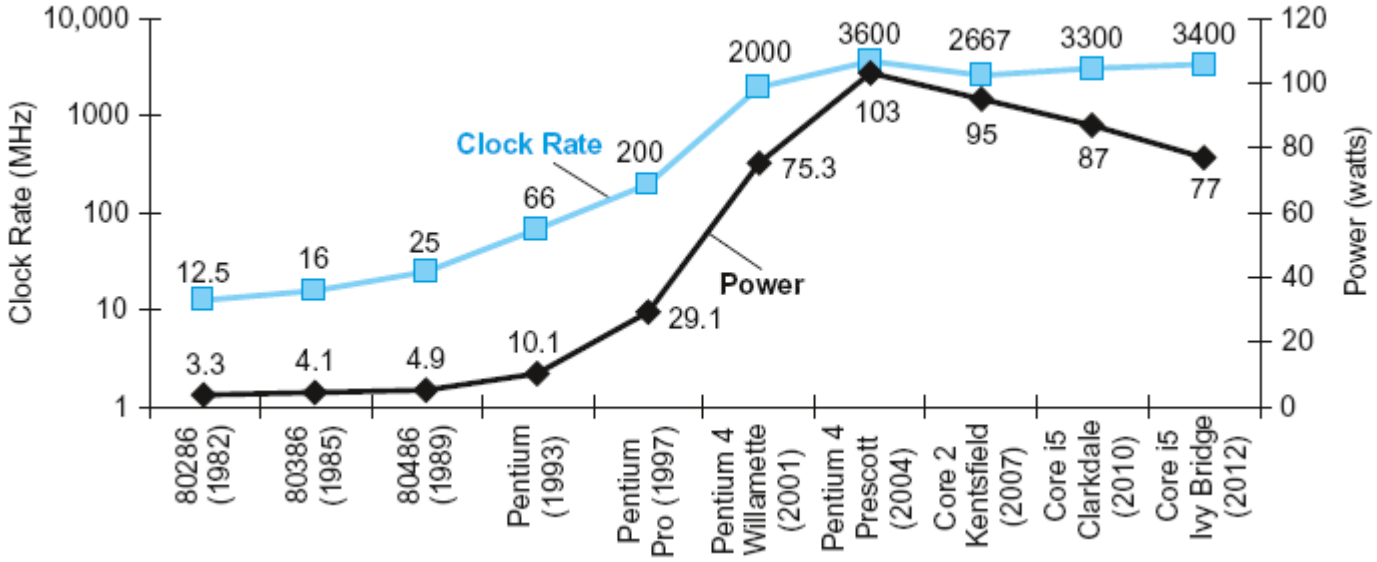
- Intel Pentium4 (2003)
  - Application: desktop/server
  - Technology: 90nm (1/100x)
  - 55M transistors (20,000x)
  - 101 $mm^2$ (10x)
  - 3.4 GHz (10,000x)
  - 1.2 Volts (1/10x)
  - 32/64-bit data (16x)
  - 22-stage pipelined datapath
  - 3 instructions per cycle (superscalar)
  - Two levels of on-chip cache
  - data-parallel vector (SIMD) instructions, hyperthreading

# Uniprocessor Performance




Constrained by power, instruction-level parallelism, memory latency

# Power Trends

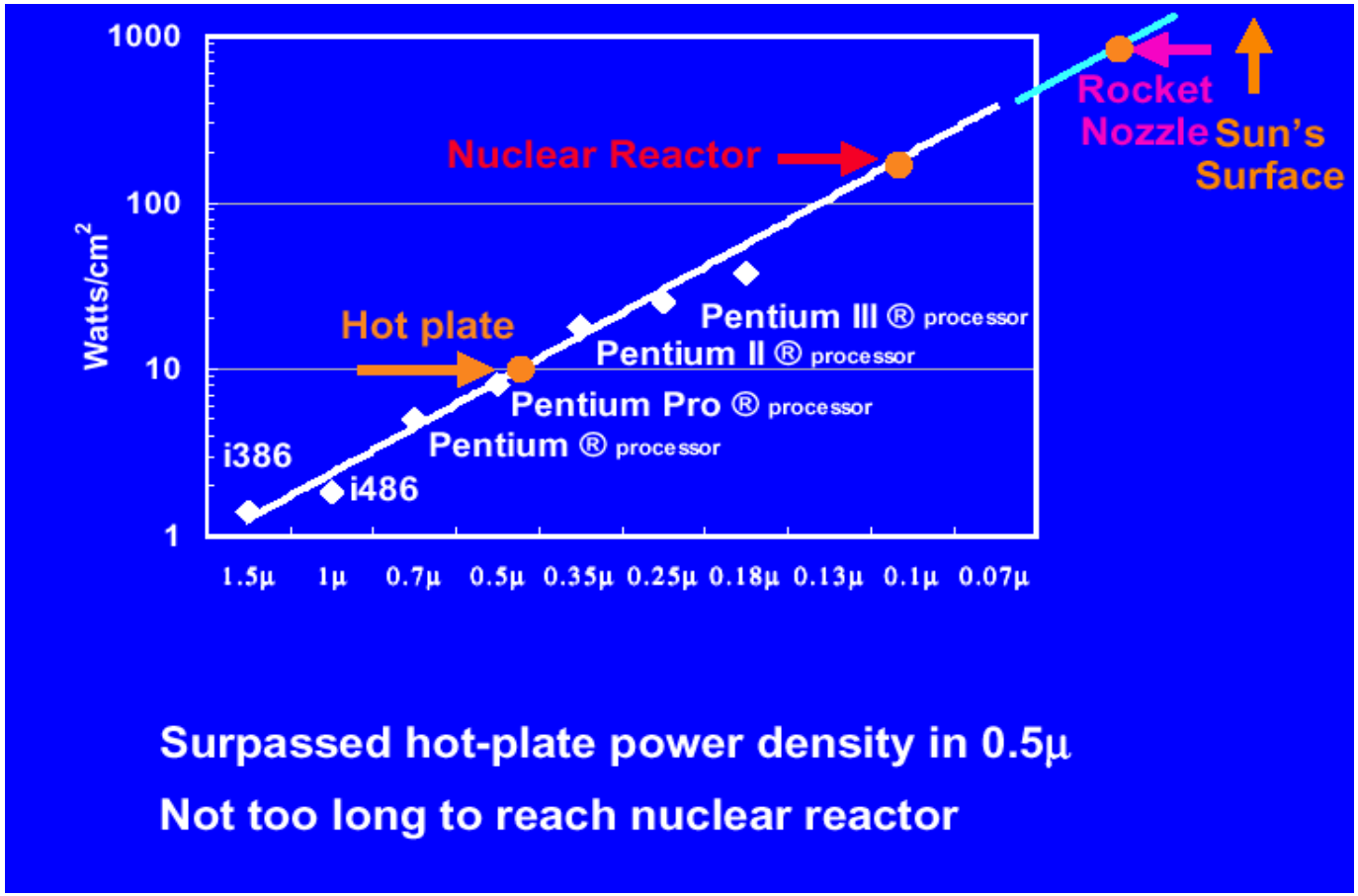



- In CMOS IC technology

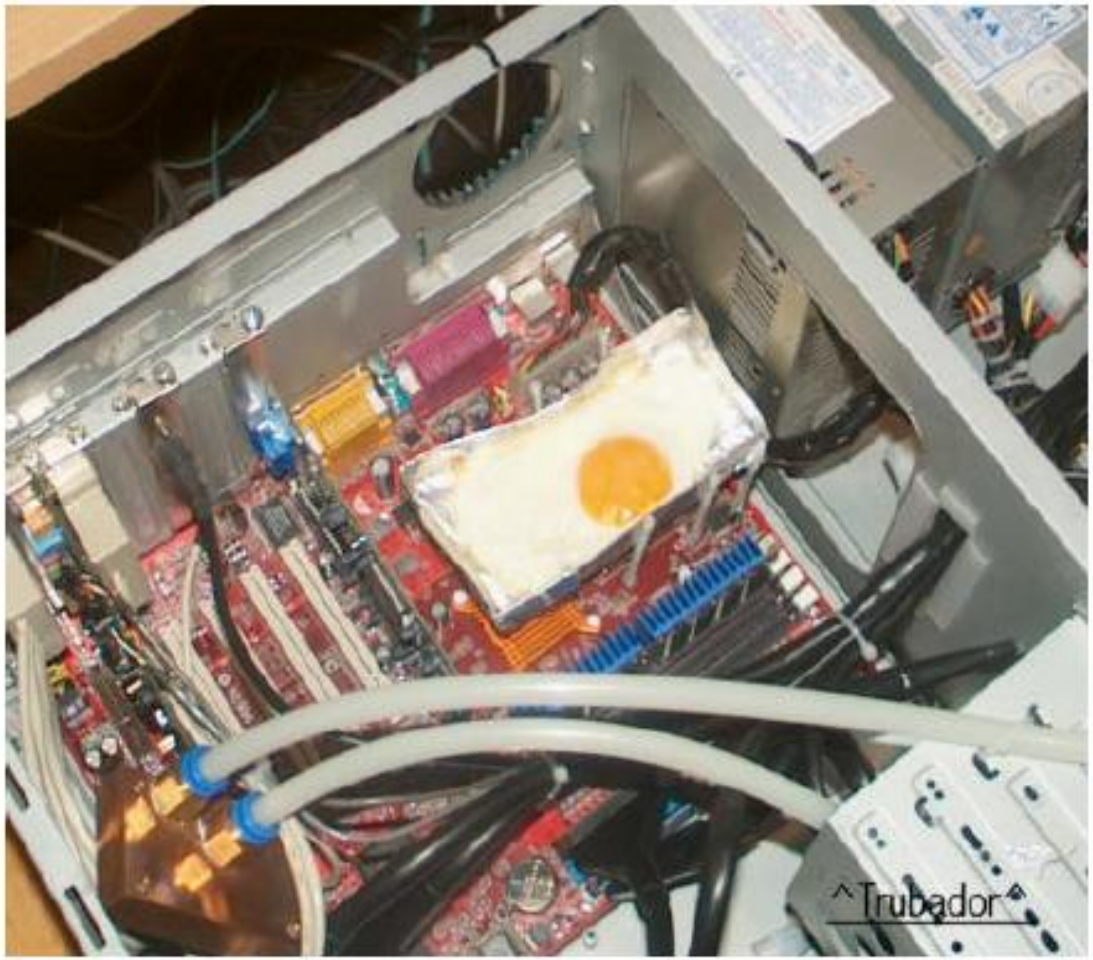
$$\text{Power} = \text{Capacitive load} \times \text{Voltage}^2 \times \text{Frequency}$$

×30 | 5V → 1V | ×1000

# Power Density Getting Worse



# Surpassed hot (kitchen) plate … why not use it?



# The Sea Change

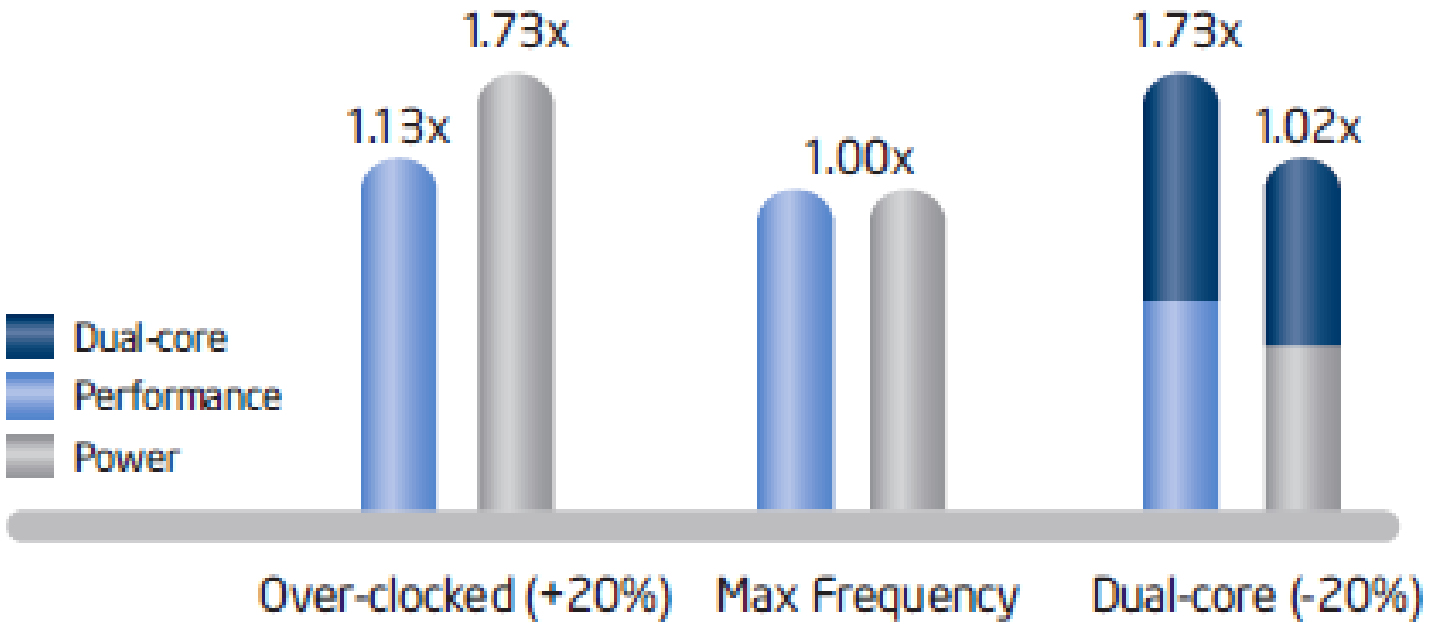
- The power challenge has forced a change in the design of microprocessors
  - Since 2002 the rate of improvement in the response time of programs on desktop computers has slowed from a factor of 1.5 per year to less than a factor of 1.2 per year
- As of 2006 all desktop and server companies are shipping microprocessors with **multiple** processors – cores – per chip

| Product | AMD Barcelona | Intel Nehalem | IBM Power 6 | Sun Niagara 2 |
|---|---|---|---|---|
| Cores per chip | 4 | 4 | 2 | 8 |
| Clock rate | 2.5 GHz | ~2.5 GHz? | 4.7 GHz | 1.4 GHz |
| Power | 120 W | ~100 W? | ~100 W? | 94 W |

- Plan of record is to double the number of cores per chip per generation (about every two years)

# Why Multiple Cores on a Chip?

- Relative to single-core frequency and Vcc



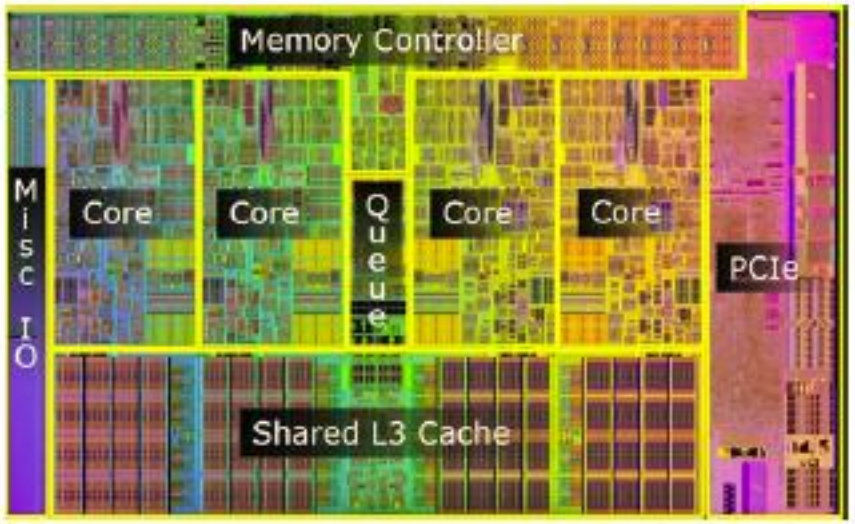
# Revolution III: Explicit Parallelism

- Then to support **explicit data & thread level parallelism**
  - Hardware provides parallel resources, software specifies usage
  - Why? diminishing returns on instruction-level-parallelism
- First using (subword) vector instructions…, Intel's SSE
  - One instruction does four parallel multiplies
- … and general support for multi-threaded programs
  - Coherent caches, hardware synchronization primitives
- Then using support for multiple concurrent threads on chip
  - First with single-core multi-threading, now with multi-core
- Graphics processing units (GPUs) are highly parallel
  - Converging with general-purpose processors (CPUs)?

- **Will cover some of this in ECE452; more in-depth coverage in ECE554 (Spring 2016)**

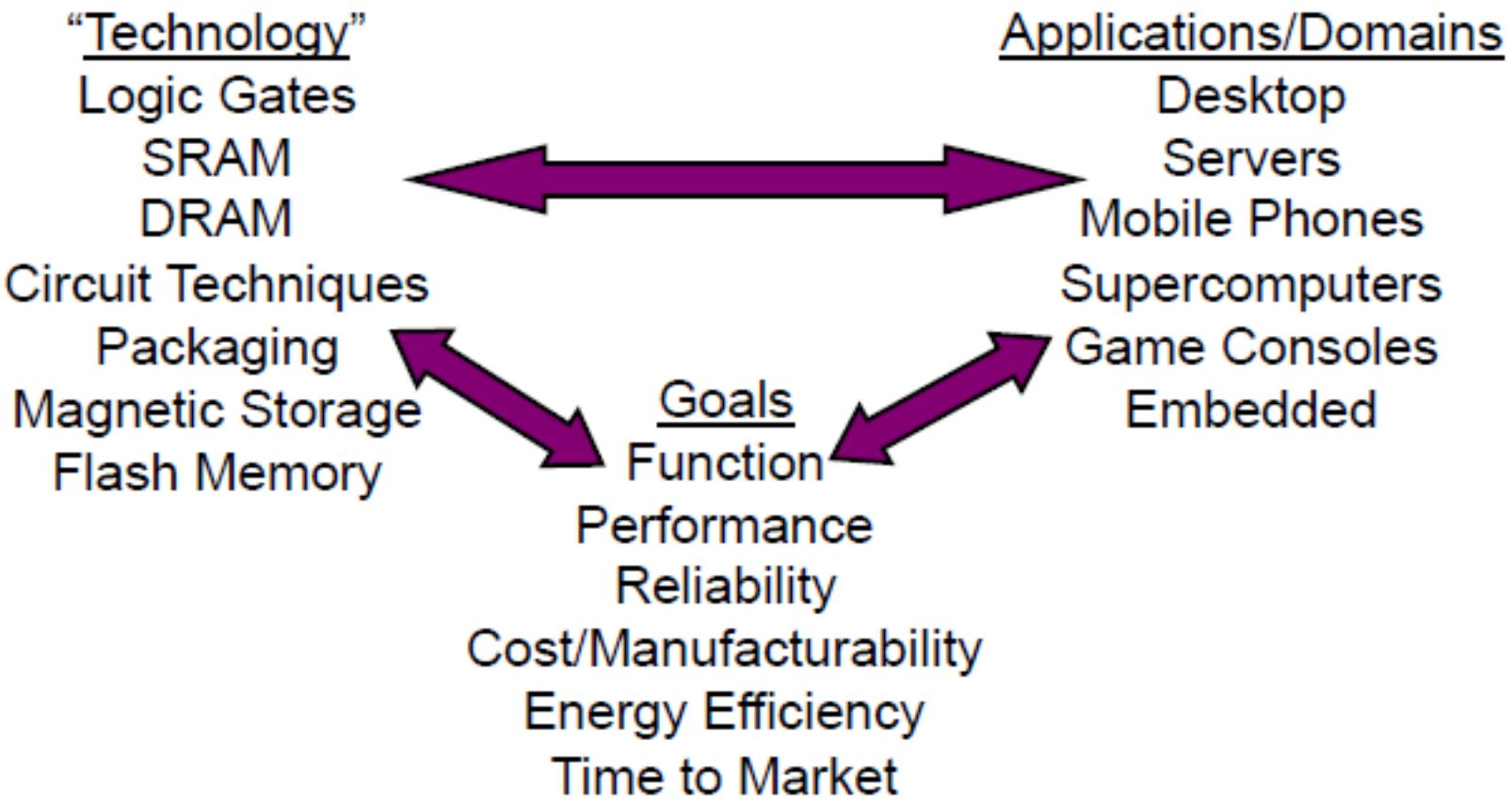
# Modern Multicore Processor

- Intel Core i7 (2009)
  - Application: desktop/server
  - Technology: 45nm (1/2x)
  - 774M transistors (12x)
  - 296 $mm^2$ (3x)
  - 3.2 GHz to 3.6 Ghz (~1x)
  - 0.7 to 1.4 Volts (~1x)




  - 128-bit data (2x)
  - 14-stage pipelined datapath (0.5x)
  - 4 instructions per cycle (~1x)
  - Three levels of on-chip cache
  - data-parallel vector (SIMD) instructions, hyperthreading
  - **Four-core multicore** (4x)

# Recap: Constant Change

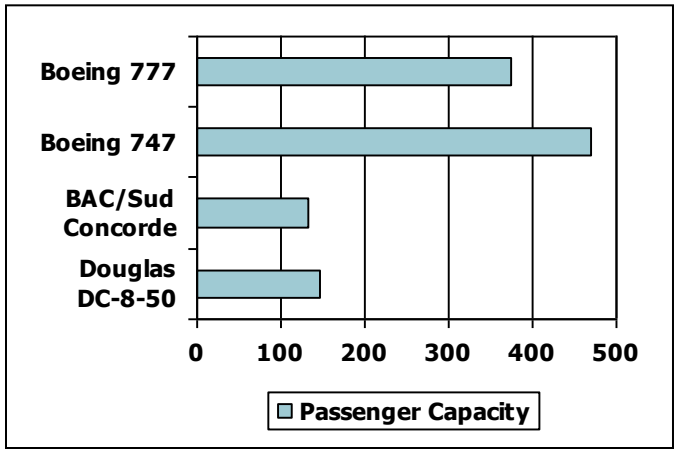


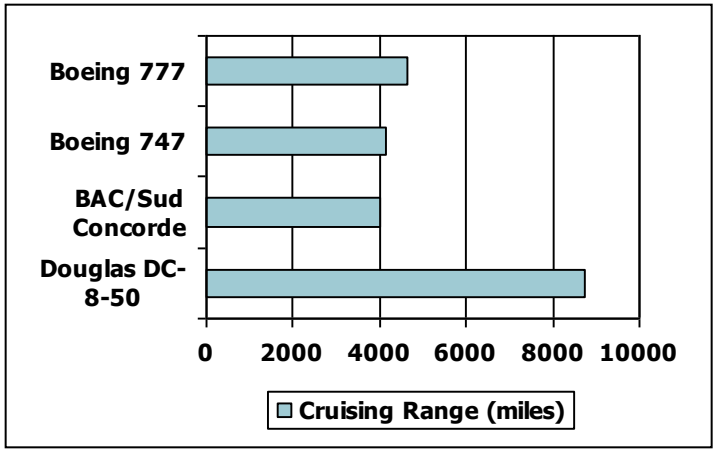
# Performance Metrics

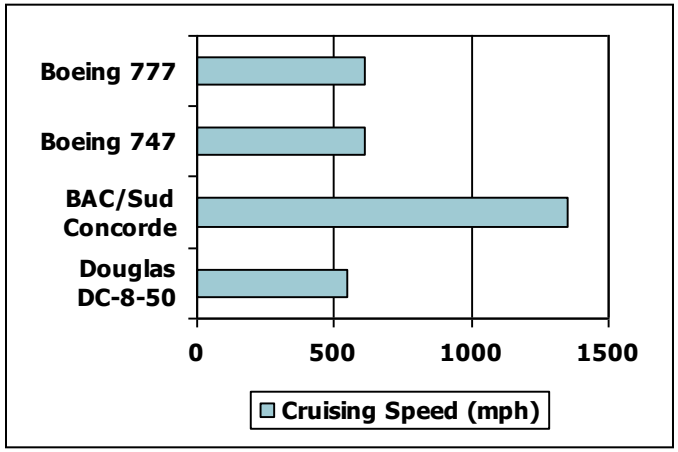
- Purchasing perspective
  - given a collection of machines, which has the
    - best performance ?
    - least cost ?
    - best cost/performance?
- Design perspective
  - faced with design options, which has the
    - best performance improvement ?
    - least cost ?
    - best cost/performance?
- Both require
  - basis for comparison
  - metric for evaluation
- Very important to understand what factors in the architecture contribute to overall system performance (recall last lecture) and relevant metrics to measure performance

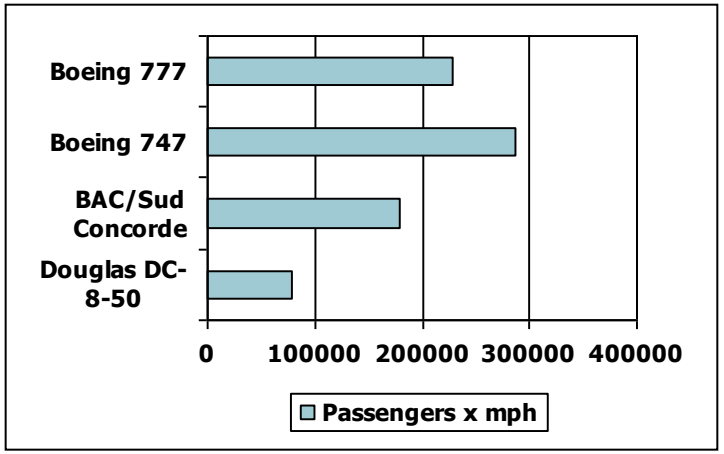
# Defining Performance

- Which airplane has the best performance?



# Response Time and Throughput

- Response time
  - How long it takes to do a task
    - Important to individual users
- Throughput
  - Total work done per unit time
    - e.g., tasks/transactions/… per hour
    - Important to data center managers
- How are response time & throughput affected by
  - Replacing the processor with a faster version?
  - Adding more processors?
- We'll focus on response time for now…

# Relative Performance

- Define Performance = 1/Execution Time
- "X is $n$ time faster than Y"

$$\text{Performance}_X / \text{Performance}_Y$$
$$= \text{Execution time}_Y / \text{Execution time}_X = n$$

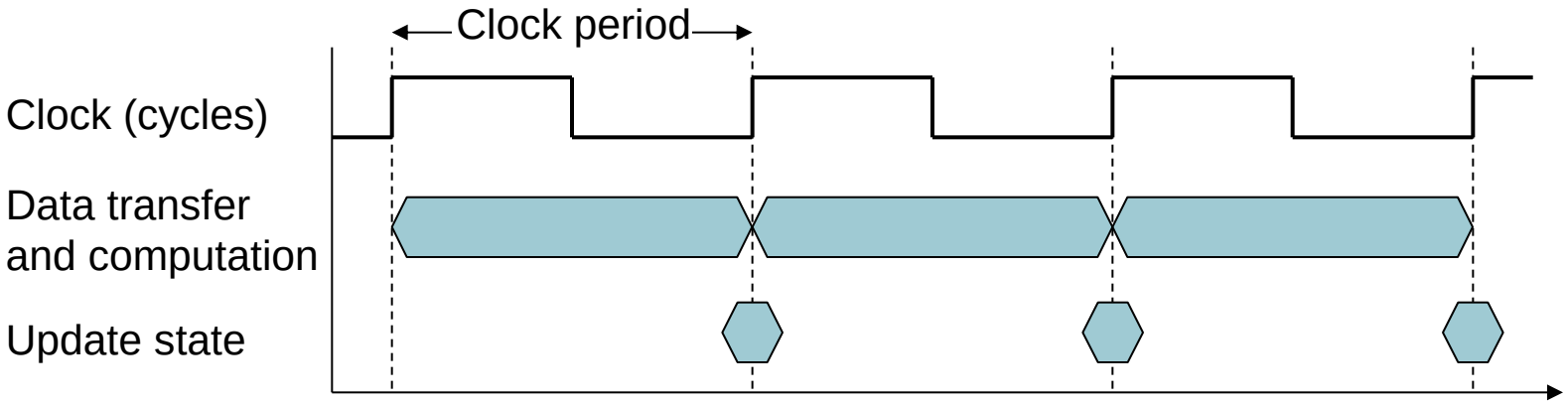
- Example: time taken to run a program
  - 10s on X, 15s on Y
  - Execution Time$_Y$ / Execution Time$_X$
    = 15s / 10s = 1.5
  - So X is 1.5 times faster than Y

# Measuring Execution Time

- Execution time: seconds/program
- Elapsed time (wall clock time)
  - Total response time, including all aspects
    - Processing, I/O, OS overhead, idle time
  - Determines system performance
- CPU time
  - Time spent processing a given job
    - Discounts I/O time, other jobs' shares
  - Comprises user CPU time and system CPU time
  - Different programs are affected differently by CPU and system performance

# CPU Clocking: Review

- Operation of digital hardware governed by a constant-rate clock




- Clock period: duration of a clock cycle
  - e.g., 250ps = 0.25ns = $250 \times 10^{-12}$s
- Clock frequency (rate): cycles per second
  - e.g., 4.0GHz = 4000MHz = $4.0 \times 10^{9}$Hz

# CPU Clocking: Review

- Clock rate (clock cycles per second in MHz or GHz) is inverse of clock cycle time (clock period)

$$CC = 1 / CR$$

10 nsec clock cycle => 100 MHz clock rate

5 nsec clock cycle => 200 MHz clock rate

2 nsec clock cycle => 500 MHz clock rate

1 nsec ($10^{-9}$) clock cycle => 1 GHz ($10^{9}$) clock rate

500 psec clock cycle => 2 GHz clock rate

250 psec clock cycle => 4 GHz clock rate

200 psec clock cycle => 5 GHz clock rate

# CPU Time

$$\text{CPU Time} = \text{CPU Clock Cycles} \times \text{Clock Cycle Time}$$

$$= \frac{\text{CPU Clock Cycles}}{\text{Clock Rate}}$$

- Performance improved by
  - Reducing number of clock cycles
  - Increasing clock rate
- Hardware designer must often trade off clock rate against cycle count
  - Many techniques that decrease the number of clock cycles also decrease the clock rate

# CPU Time Example

- A program runs on computer A with a 2 GHz clock in 10 seconds. What clock rate must computer B run at to run this program in 6 seconds?
  - Unfortunately, to accomplish this, computer B will require 1.2 times as many clock cycles as computer A to run the program.

$$\text{Clock Rate}_B = \frac{\text{Clock Cycles}_B}{\text{CPU Time}_B} = \frac{1.2 \times \text{Clock Cycles}_A}{6\text{s}}$$

$$\text{Clock Cycles}_A = \text{CPU Time}_A \times \text{Clock Rate}_A$$

$$= 10\text{s} \times 2\text{GHz} = 20 \times 10^9$$

$$\text{Clock Rate}_B = \frac{1.2 \times 20 \times 10^9}{6\text{s}} = \frac{24 \times 10^9}{6\text{s}} = 4\text{GHz}$$

# Instruction Count and CPI

$$\text{Clock Cycles} = \text{Instruction Count} \times \text{Cycles per Instruction}$$

$$\text{CPU Time} = \text{Instruction Count} \times \text{CPI} \times \text{Clock Cycle Time}$$

$$= \frac{\text{Instruction Count} \times \text{CPI}}{\text{Clock Rate}}$$

- Instruction Count for a program
  - Determined by program, ISA and compiler
- Average cycles per instruction
  - Determined by CPU hardware
  - If different instructions have different CPI
    - Average CPI affected by instruction mix

# CPI Example

- Computer A: Cycle Time = 250ps, CPI = 2.0
- Computer B: Cycle Time = 500ps, CPI = 1.2
- Same ISA
- Which is faster, and by how much?

$$\text{CPU Time}_A = \text{Instruction Count} \times \text{CPI}_A \times \text{Cycle Time}_A$$
$$= I \times 2.0 \times 250\text{ps} = I \times 500\text{ps}$$

A is faster…

$$\text{CPU Time}_B = \text{Instruction Count} \times \text{CPI}_B \times \text{Cycle Time}_B$$
$$= I \times 1.2 \times 500\text{ps} = I \times 600\text{ps}$$

$$\frac{\text{CPU Time}_B}{\text{CPU Time}_A} = \frac{I \times 600\text{ps}}{I \times 500\text{ps}} = 1.2$$

…by this much

# CPI in More Detail

- If different instruction classes take different numbers of cycles

$$\text{Clock Cycles} = \sum_{i=1}^{n} (\text{CPI}_i \times \text{Instruction Count}_i)$$

  - Weighted average CPI

$$\text{CPI} = \frac{\text{Clock Cycles}}{\text{Instruction Count}} = \sum_{i=1}^{n} \left( \text{CPI}_i \times \underbrace{\frac{\text{Instruction Count}_i}{\text{Instruction Count}}}_{\text{Relative frequency}} \right)$$

# CPI Example

- Alternative compiled code sequences using instructions in classes A, B, C. What is avg. CPI?

| Class | A | B | C |
|---|---|---|---|
| CPI for class | 1 | 2 | 3 |
| IC in sequence 1 | 2 | 1 | 2 |
| IC in sequence 2 | 4 | 1 | 1 |

- Sequence 1: IC = 5
  - Clock Cycles
    = 2×1 + 1×2 + 2×3
    = 10
  - Avg. CPI = 10/5 = 2.0
- Sequence 2: IC = 6
  - Clock Cycles
    = 4×1 + 1×2 + 1×3
    = 9
  - Avg. CPI = 9/6 = 1.5

# Another Weighted CPI Example

| Op | Freq | $CPI_i$ | Freq x $CPI_i$ | | | |
|---|---|---|---|---|---|---|
| ALU | 50% | 1 | .5 | .5 | .5 | .25 |
| Load | 20% | 5 | 1.0 | .4 | 1.0 | 1.0 |
| Store | 10% | 3 | .3 | .3 | .3 | .3 |
| Branch | 20% | 2 | .4 | .4 | .2 | .4 |
| | | | $\Sigma$ = 2.2 | 1.6 | 2.0 | 1.95 |

- How much faster would the machine be if a better data cache reduced the average load time to 2 cycles?
  CPU time new = 1.6 x IC x CC so 2.2/1.6 means 37.5% faster
- How does this compare with using branch prediction to shave a cycle off the branch time?
  CPU time new = 2.0 x IC x CC so 2.2/2.0 means 10% faster
- What if two ALU instructions could be executed at once?
  CPU time new = 1.95 x IC x CC so 2.2/1.95 means 12.8% faster

# Performance Summary

## The BIG Picture

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

| | Instruction_ count | CPI | clock_cycle |
|---|---|---|---|
| Algorithm | X | X | |
| Programming language | X | X | |
| Compiler | X | X | |
| ISA | X | X | X |
| Core organization | | X | X |
| Technology | | | X |

# Processor Performance

- Q: What does performance of a chip mean?
- A: nothing, there must be some associated workload
  - **Workload**: set of tasks someone (you) cares about

- **Benchmarks**: standard workloads
  - Used to compare performance across machines
  - Either are or highly representative of actual programs people run
  - Example: H.264/MPEG decoding, gcc compiler, etc.
- **Micro-benchmarks**: non-standard non-workloads
  - Tiny programs used to isolate certain aspects of performance
  - Not representative of complex behaviors of real applications
  - Examples: binary tree search, towers-of-hanoi, 8-queens, etc.

# Workloads and Benchmarks

- SPEC (Standard Performance Evaluation Corporation)
  - http://www.spec.org/
  - Consortium that collects, standardizes, and distributes benchmarks
  - Post **SPECmark** results for different processors
    - 1 number that represents performance for entire suite
  - Benchmark suites for CPU, Java, I/O, Web, Mail, etc.
  - Updated every few years: so companies don't target benchmarks
- SPEC CPU 2006
  - CINT2006:12 "integer": bzip2, gcc, perl, hmmer (genomics), h264,…
  - CFP2006:17 "floating point": wrf (weather), povray, sphynx3 (speech), ...
  - Written in C/C++ and Fortran

# CINT2006 for Opteron X4 2356

| Name | Description | IC×10$^9$ | CPI | Tc (ns) | Exec time | Ref time | SPECratio |
|---|---|---|---|---|---|---|---|
| perl | Interpreted string processing | 2,118 | 0.75 | 0.4 | 637 | 9,777 | 15.3 |
| bzip2 | Block-sorting compression | 2,389 | 0.85 | 0.4 | 817 | 9,650 | 11.8 |
| gcc | GNU C Compiler | 1,050 | 1.72 | 0.4 | 24 | 8,050 | 11.1 |
| mcf | Combinatorial optimization | 336 | 10.00 | 0.4 | 1,345 | 9,120 | 6.8 |
| go | Go game (AI) | 1,658 | 1.09 | 0.4 | 721 | 10,490 | 14.6 |
| hmmer | Search gene sequence | 2,783 | 0.80 | 0.4 | 890 | 9,330 | 10.5 |
| sjeng | Chess game (AI) | 2,176 | 0.96 | 0.4 | 37 | 12,100 | 14.5 |
| libquantum | Quantum computer simulation | 1,623 | 1.61 | 0.4 | 1,047 | 20,720 | 19.8 |
| h264avc | Video compression | 3,102 | 0.80 | 0.4 | 993 | 22,130 | 22.3 |
| omnetpp | Discrete event simulation | 587 | 2.94 | 0.4 | 690 | 6,250 | 9.1 |
| astar | Games/path finding | 1,082 | 1.79 | 0.4 | 773 | 7,020 | 9.1 |
| xalancbmk | XML parsing | 1,058 | 2.70 | 0.4 | 1,143 | 6,900 | 6.0 |
| Geometric mean | | | | | | | 11.7 |

High cache miss rates

# Comparing and Summarizing Performance

- How do we summarize the performance for benchmark set with a single number?
  - E.g., SPECmark 2006
  - Reference machine: Sun UltraSPARC II (@ 296 MHz)
  - Latency SPECmark: Take latency ratio (reference machine / your machine) giving the “SPEC ratio” (bigger is faster, i.e., SPEC ratio is the inverse of execution time)
  - SPEC ratios are then “averaged” using the geometric mean (GM)

$$GM = \sqrt[n]{\prod_{i=1}^{n} \text{SPEC ratio}_i}$$

  - Throughput SPECmark: Run multiple benchmarks in parallel on multiple-processor system

# Another Example: GeekBench

- Set of cross-platform multicore benchmarks
  - Can run on iPhone, Android, laptop, desktop, etc
- Tests integer, floating point, memory, memory bandwidth performance
- GeekBench stores all results online
  - Easy to check scores for many different systems, processors
- Pitfall: Workloads are simple, may not be a completely accurate representation of performance
  - We know they evaluate compared to a baseline benchmark

# GeekBench Numbers

- Desktop
  - Intel “Ivy bridge” at 3.4 GHz (4 cores) – 11,456
- Laptop:
  - Intel Core i7-3520M at 2.9 GHz (2 cores) – 7,807
- Phones:
  - iPhone 5 - Apple A6 at 1 GHz (2 cores) – 1,589
  - iPhone 4S - Apple A5 at 0.8 GHz (2 cores) – 642
  - Samsung Galaxy S III (North America) – Qualcomm Snapdragon S3 – 1.500 GHz (2 cores) – 1,429

# Assignment 1

- Access assignment from Canvas
- To be submitted via Canvas:
  - 3 Feb (Tuesday)
- Read Chapter 1 of P&H

# SPEC Power Benchmark

- Power consumption of server at different workload levels
  - Performance: ssj_ops/sec (server side java ops/sec)
  - Power: Watts (Joules/sec)

$$\text{Overall ssj\_ops per Watt} = \left(\sum_{i=0}^{10} \text{ssj\_ops}_i\right) \Big/ \left(\sum_{i=0}^{10} \text{power}_i\right)$$

# SPECpower_ssj2008 for X4

| Target Load % | Performance (ssj_ops/sec) | Average Power (Watts) |
|---|---|---|
| 100% | 231,867 | 295 |
| 90% | 211,282 | 286 |
| 80% | 185,803 | 275 |
| 70% | 163,427 | 265 |
| 60% | 140,160 | 256 |
| 50% | 118,324 | 246 |
| 40% | 920,35 | 233 |
| 30% | 70,500 | 222 |
| 20% | 47,126 | 206 |
| 10% | 23,066 | 180 |
| 0% | 0 | 141 |
| Overall sum | 1,283,590 | 2,605 |
| ∑ssj_ops/ ∑power | | 493 |

# SPECpower_ssj2008 for Xeon X5650

| Target Load % | Performance (ssj_ops) | Average Power (Watts) |
|---|---|---|
| 100% | 865,618 | 258 |
| 90% | 786,688 | 242 |
| 80% | 698,051 | 224 |
| 70% | 607,826 | 204 |
| 60% | 521,391 | 185 |
| 50% | 436,757 | 170 |
| 40% | 345,919 | 157 |
| 30% | 262,071 | 146 |
| 20% | 176,061 | 135 |
| 10% | 86,784 | 121 |
| 0% | 0 | 80 |
| Overall Sum | 4,787,166 | 1,922 |
| Σssj_ops/Σpower = | | 2,490 |

# Other Benchmarks

- Parallel benchmarks
  - SPLASH2: Stanford Parallel Applications for Shared Memory
  - NAS: another parallel benchmark suite
  - SPECopenMP: parallelized versions of SPECfp 2000)
  - SPECjbb: Java multithreaded database-like workload

- Transaction Processing Council (TPC)
  - TPC-C: On-line transaction processing (OLTP)
  - TPC-H/R: Decision support systems (DSS)
  - TPC-W: E-commerce database backend workload
  - Have parallelism (intra-query and inter-query)
  - Heavy I/O and memory components

# Pitfall: Amdahl's Law

- Amdahl's law: performance enhancement possible with a given improvement is limited by the amount that the improved feature is used
- Pitfall: Improving an aspect of a computer and expecting a proportional improvement in overall performance

$$T_{improved} = \frac{T_{affected}}{improvement\ factor} + T_{unaffected}$$

- Example: multiply accounts for 80s/100s
  - How much improvement in multiply performance to get 5× overall?

$$20 = \frac{80}{n} + 20$$

  - Can't be done!
- Corollary: make the common case fast

# Fallacy: Low Power at Idle

- Look back at X4 power benchmark
  - At 100% load: 295W
  - At 50% load: 246W (83%)
  - At 10% load: 180W (61%)
- Google data center
  - Mostly operates at 10% – 50% load
  - At 100% load less than 1% of the time
- Need to design future processors to make power proportional to load
  - Not easy!

# Pitfall: MIPS as a Performance Metric

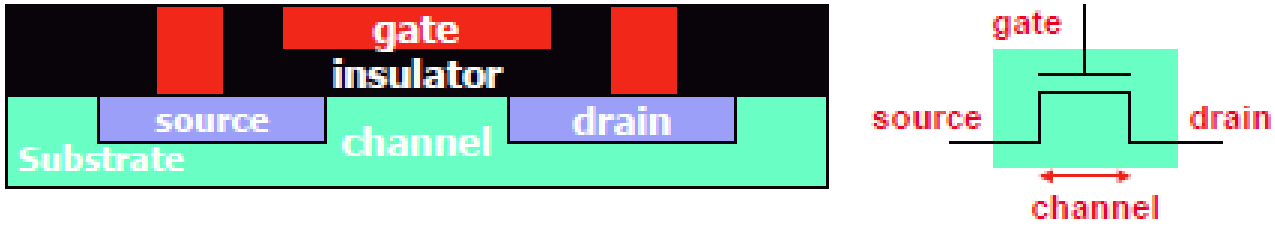
- MIPS: Millions of Instructions Per Second
  - If used as a metric to compare computers:
    - Doesn't account for differences in instruction complexity
      - Different ISAs may lead to different instruction counts for same program
    - MIPS varies between programs on the same computer!
      - Computer cannot have a single MIPS rating

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

$$= \frac{\text{Instruction count}}{\frac{\text{Instruction count} \times \text{CPI}}{\text{Clock rate}} \times 10^6} = \frac{\text{Clock rate}}{\text{CPI} \times 10^6}$$

  - e.g. CPI varied by 13x for SPEC2006 on AMD Opteron X4, so MIPS does as well

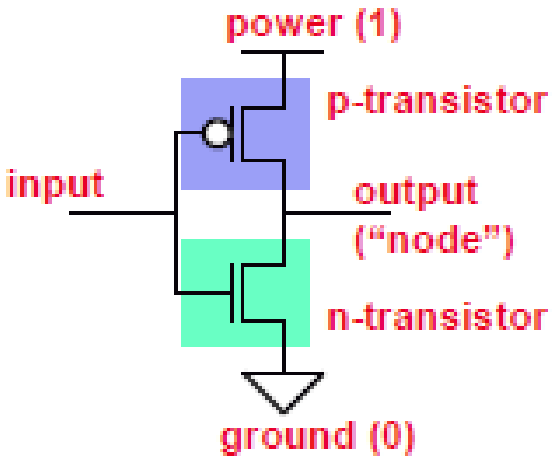
# Semiconductor Technology




- Basic technology element: **MOSFET**
  - Solid-state component acts like electrical switch
  - **MOS**: metal-oxide-semiconductor
    - Conductor, insulator, semi-conductor
- **FET**: field-effect transistor
  - Channel conducts source→drain only when voltage applied to gate
- **Channel length**: characteristic parameter (short → fast)
  - Aka “feature size” or “technology”
  - Currently: 0.022 micron (μm), 22 nanometers (nm)
  - Continued miniaturization (scaling) known as “**Moore’s Law**”
    - Won’t last forever, physical limits approaching

# Complementary MOS (CMOS)

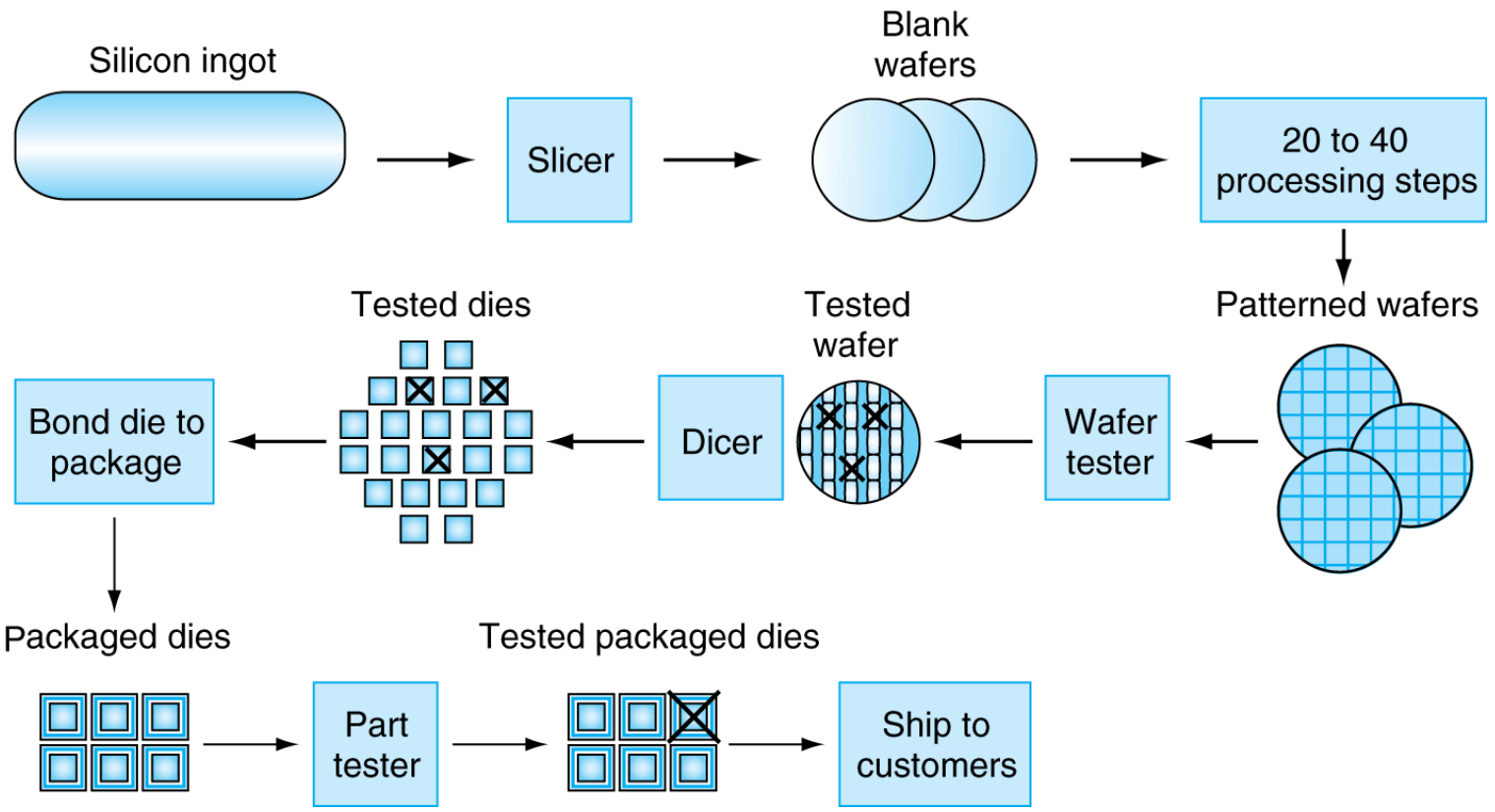
- Voltages as values
  - Power (VDD) = “1”, Ground = “0”
- Two kinds of MOSFETs
  - **N-transistors**
    - Conduct when gate voltage is 1
    - Good at passing 0s
  - **P-transistors**
    - Conduct when gate voltage is 0
    - Good at passing 1s




CMOS inverter gate

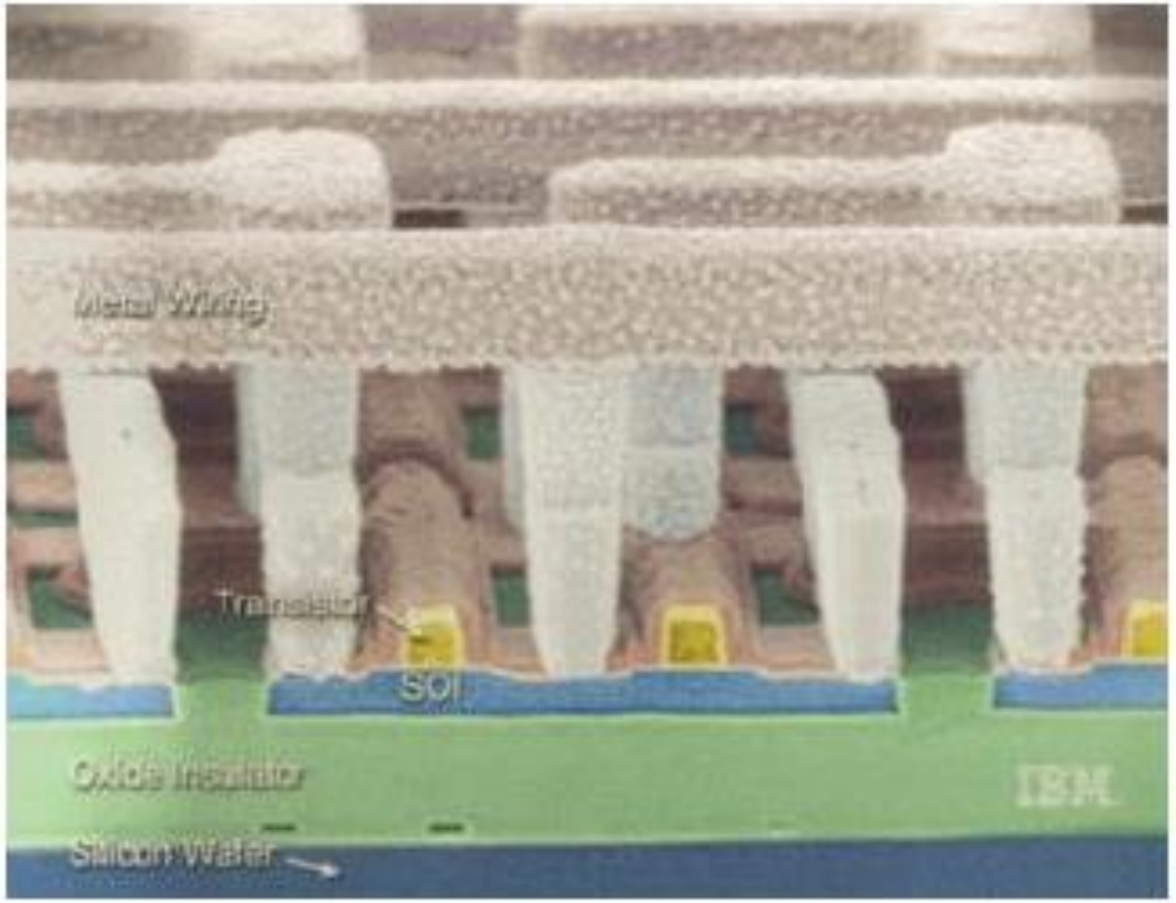
- **CMOS**
  - Complementary n-/p- networks form boolean logic (i.e., gates)
  - And some non-gate elements too (important example: RAMs)

# Manufacturing ICs

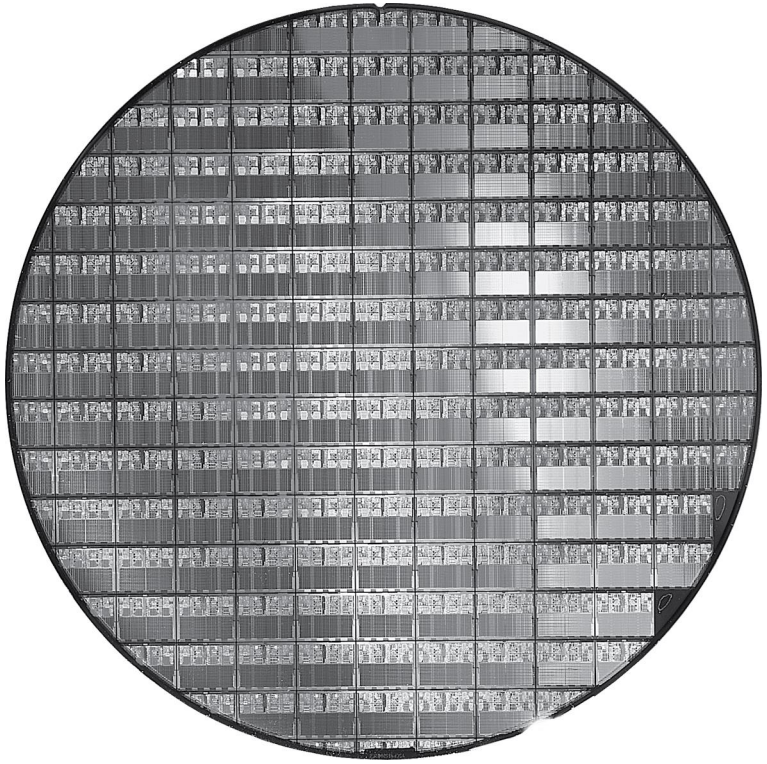



- Yield: proportion of working dies per wafer

# Transistors and Wires



# AMD Opteron X2 Wafer



- X2: 300mm wafer, 117 chips, 90nm technology
- X4: 45nm technology

# Integrated Circuit Cost

$$\text{Cost per die} = \frac{\text{Cost per wafer}}{\text{Dies per wafer} \times \text{Yield}}$$

$$\text{Dies per wafer} \approx \text{Wafer area} / \text{Die area}$$

$$\text{Yield} = \frac{1}{(1 + (\text{Defects per area} \times \text{Die area}/2))^2}$$

- Nonlinear relation to area and defect rate
  - Wafer cost and area are fixed
  - Defect rate determined by manufacturing process
  - Die area determined by architecture and circuit design

# Concluding Remarks

- Performance is improving
  - Due to underlying technology development
- Hierarchical layers of abstraction
  - In both hardware and software
- Instruction set architecture
  - The hardware/software interface
- Execution time: the best performance measure!
- Power is a limiting factor
  - Use parallelism to improve performance