CMPEN 431
Computer Architecture
Fall 2017
Abstractions Technology, and Performance

Mahmut Taylan Kandemir (www.cse.psu.edu/~kandemir)

[Adapted from *Computer Organization and Design, 5th Edition*, Patterson & Hennessy, © 2014, Morgan Kaufmann]

Course Administration

Instructor:	Mahmut Taylan Kandemir (kandemir@cse.psu.edu) W321, Westgate Bldg Office Hours: Tue-Thu 2PM-3PM
TA:	Huaipan Jiang Office Hours: posted on Canvas
URL:	Canvas
Text:	Required: *Computer Org and Design*, 5rd Ed., Patterson & Hennessy, ©2014
Slides:	pdf on Canvas after the lecture
ACK:	Profs. Mary Jane Irwin, John Sampson

Grading Information

- Grade determinates
  - 4 midterm exams (Sep 14th, Oct 10th, Nov 2nd, Nov30th) 30%
  - Final exam (comprehensive) 45%
  - 2 Programming Projects 15%
    - To be submitted on Angel by 23:59 on the due date. No late assignments will be accepted.
  - On-line (Angel) quizzes 10%
    - Due in 7 days (you have to finish each quiz in 50 minutes)
- Let me know about exam conflicts ASAP
- Grades will be posted on Canvas
  - Must submit email request for change of grade after discussions with the TA (Projects/Quizzes) or instructor (Exams)
- Always send emails via Canvas
- Attending the class is VERY IMPORTANT
- Any apparent cases of collaboration on exams, or assignments will be treated as academic dishonesty

Homeworks

- Homeworks will be given (periodically), but they will NOT be graded
- Solutions to some of the homeworks (but NOT all) will be posted on Canvas
- I strongly encourage you to attempt homework questions (in a timely fashion)

Grade Assignment

- Cumulative Score = 0.45 x Final Score + 0.30 x Avg Midterm Score + 0.15 x Avg Project Score + 0.10 x Avg Quiz Score

- If Cumulative Score < 50 ➔ fail, else pass

- I will let you know your standing after each important event

Academic Integrity & Other Policies

- Please read the syllabus in Canvas
- No, really, actually read the syllabus
- … the whole thing
- I cannot overemphasize the importance of it

Academic Integrity

- Academic integrity is a core value at Penn State. Policies guiding what behaviors are considered to be acting in accordance with community standards for academic integrity exist at multiple levels within the institution, and all must be heeded.

- The University defines academic integrity as the pursuit of scholarly activity in an open, honest and responsible manner. All students should act with personal integrity, respect other students' dignity, rights and property, and help create and maintain an environment in which all can succeed through the fruits of their efforts (refer to Senate Policy 49-20). Dishonesty of any kind will not be tolerated in this course. Dishonesty includes, but is not limited to, cheating, plagiarizing, fabricating information or citations, facilitating acts of academic dishonesty by others, having unauthorized possession of examinations, submitting work of another person or work previously used without informing the instructor, or tampering with the academic work of other students. Students who are found to be dishonest will receive academic sanctions and will be reported to the University's Office of Student Conduct for possible further disciplinary sanctions (refer to Senate Policy G-9).

- The CSE department has a departmental academic integrity statement that can be found here: http://www.eecs.psu.edu/students/resources/EECS-CSE-Academic-Integrity.aspx

Course Structure and Schedule

- Lecture: 9:05AM to 10:20AM Tuesdays and Thursdays
- Lectures:
  - 2 weeks: review of the MIPS ISA and basic architecture
  - 2 weeks: scalar pipelined datapath design issues
  - 2 weeks: memory hierarchies and memory design issues
  - 2.5 weeks: superscalar datapath design issues
  - 1 week: storage and I/O design issues
  - 2.5 weeks: multiprocessor/multicore design issues
- Design experience
  - Simulation of architecture alternatives using SimpleScalar

Course Content

- CPU design, pipelining, cache/memory hierarchy design, multiprocessor/multicore architectures, storage.
  - "This course will introduce students to the architecture-level design issues of a computer system. They will apply their knowledge of digital logic design to explore the high-level interaction of the individual computer system hardware components. Concepts of sequential and parallel architecture including the interaction of different memory components, their layout and placement, communication among multiple processors, effects of pipelining, and performance issues, will be covered. Students will apply these concepts by studying and evaluating the merits and demerits of selected computer system architectures."

学习什么决定计算机系统的功能与性能，理解计算机架构与其软件之间的交互以便于未来的计算机设计者能够设计出高性价比的软件；未来的计算机架构师能够理解软件的设计选择的影响。

- To learn what determines the capabilities and performance of computer systems and to understand the interactions between the computer's architecture and its software so that future software designers (compiler writers, operating system designers, database programmers, application programmers, …) can achieve the best cost-performance trade-offs and so that future computer architects understand the effects of their design choices on software.

What You Should Know – 271, 331, 311

- Basic logic design and machine organization
  - logical minimization, FSMs, component design
  - processor, memory, I/O
- Create, assemble, run, debug programs in an assembly language
  - MIPS preferred
- Create, simulate, and debug hardware structures in a hardware description language
  - VHDL or Verilog
- Create, compile, and run C (C++, Java) programs
- Create, organize, and edit files and run programs on Unix/Linux

Classes of Computers

- Desktop/laptop (PC)/tablet computers
  - Single user
  - General purpose, variety of software/applications
  - Subject to cost/performance/power tradeoff
- Servers/Clouds/Data Centers/Supercomputers
  - Multiple, simultaneous users
  - Network based, terabytes of memory, petabytes of storage
  - High capacity, performance, reliability/availability, security
  - Range from small servers to building sized
- Embedded computers (processors)
  - Hidden as components of systems, used for one predetermined application (or small set of applications)
  - Stringent power/performance/cost constraints
  - Lots of IPs
  - Handhelds, wearables (fitness bands, smart watches, smart glasses)

Embedded Processor Characteristics

The largest class of computers spanning the widest range of applications and performance

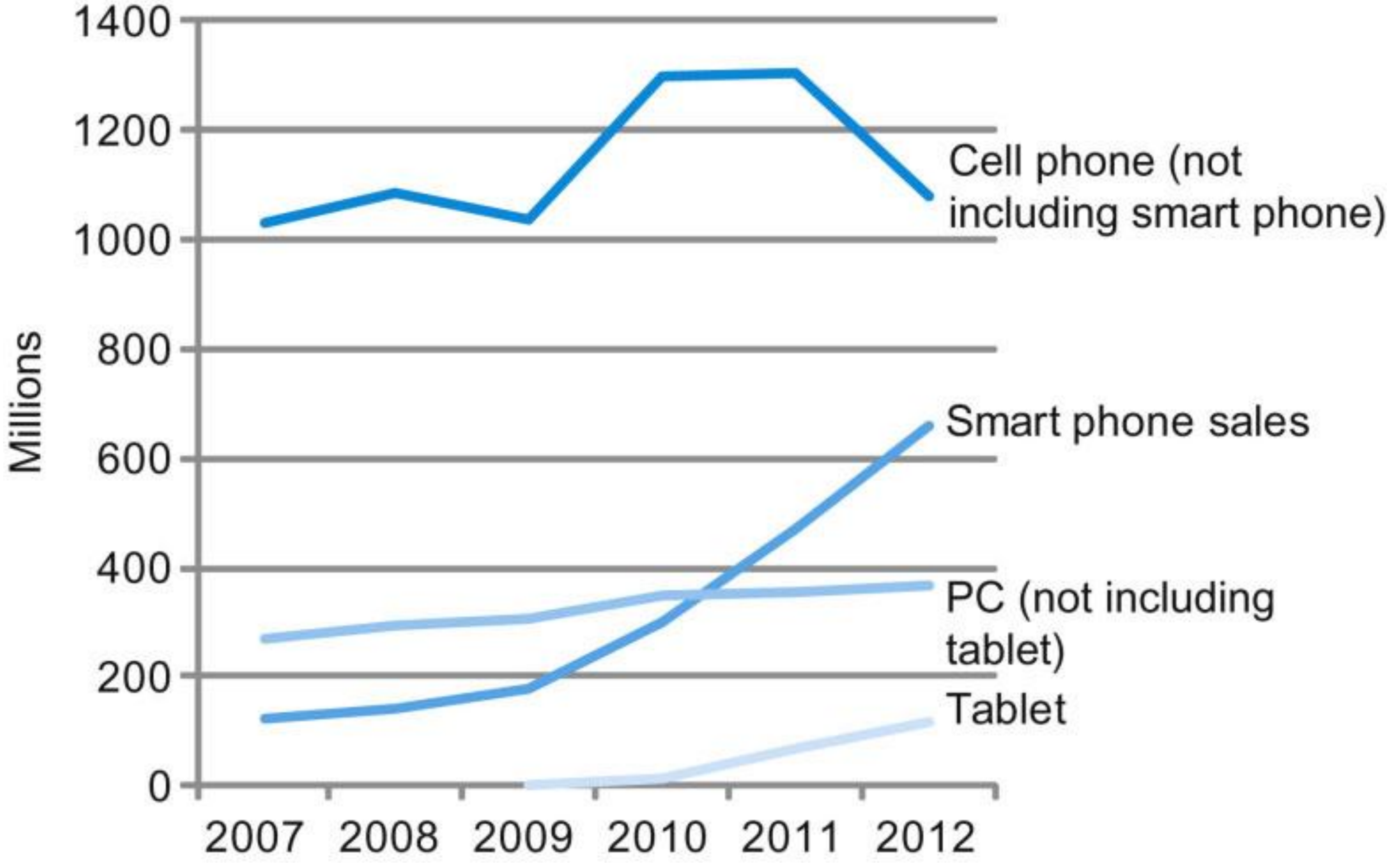
- ❑ Often have minimum performance requirements
- ❑ Often have stringent limitations on cost and form factor
- ❑ Often have stringent limitations on power consumption
- ❑ Often have low tolerance for failure

The PostPC Era

- Personal mobile devices (PMDs)
  - 电池供电
  - Battery operated, touch screen (no mouse, no keyboard)
  - Connects to the Internet, download “apps”
  - A few hundreds of dollars (or less) in cost
  - Smart phones, tablets, electronic glasses, cameras, …
- Cloud computing
  - Warehouse Scale Computers (WSC)
  - Software as a Service (SaaS) deployed via the Cloud
  - Portion of software run on a PMD and a portion runs in the Cloud
  - Amazon, Google, …

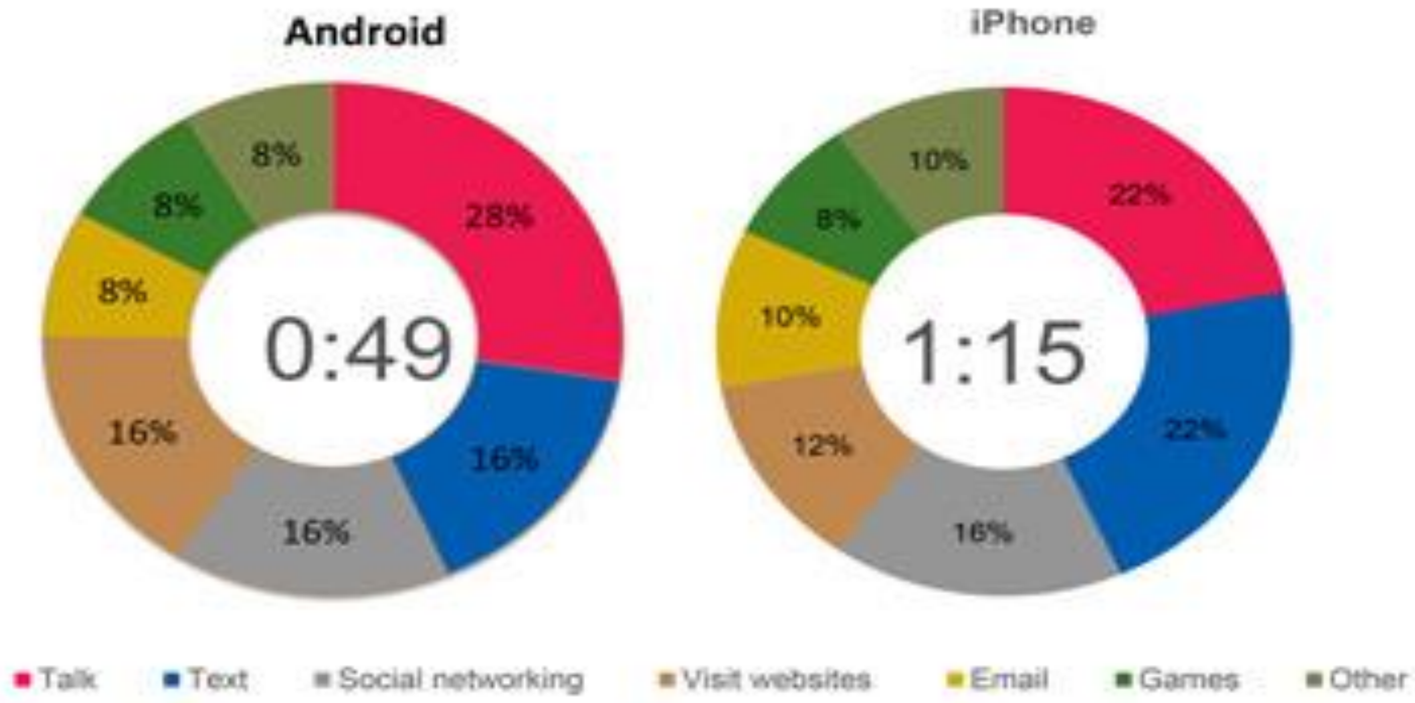
Growth in PMDs (Personal Mobile Devices)

PMDs growth >> PC growth



Smartphone Usage

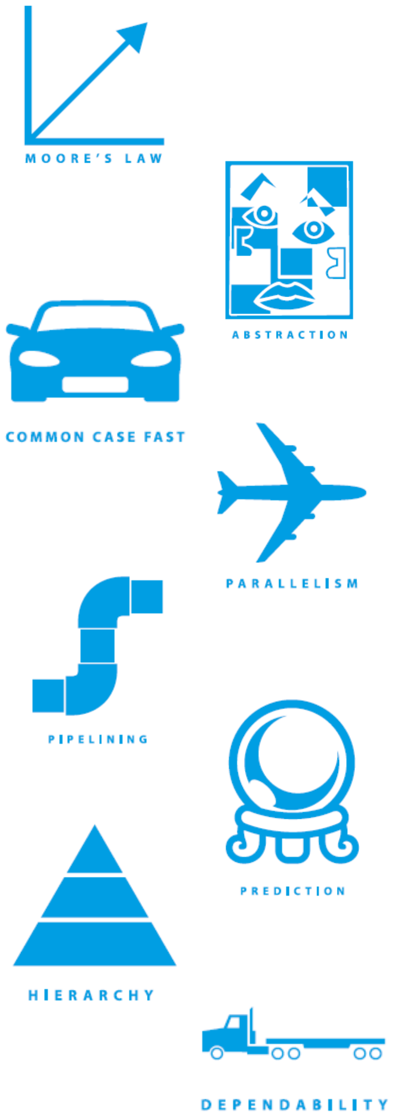
Total smartphone time spent daily and activity share, by device



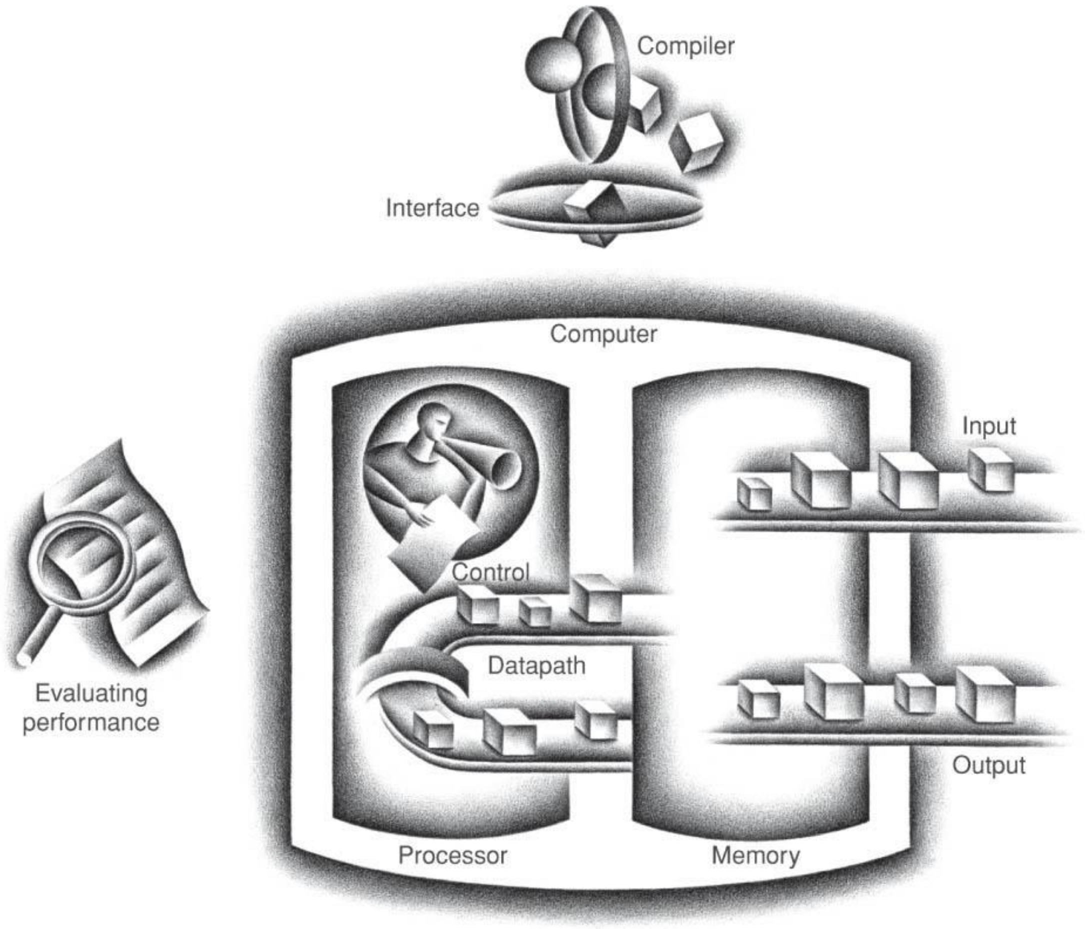
Source: Experian Marketing Services

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



The Five Classic Components of a Computer



input/output includes

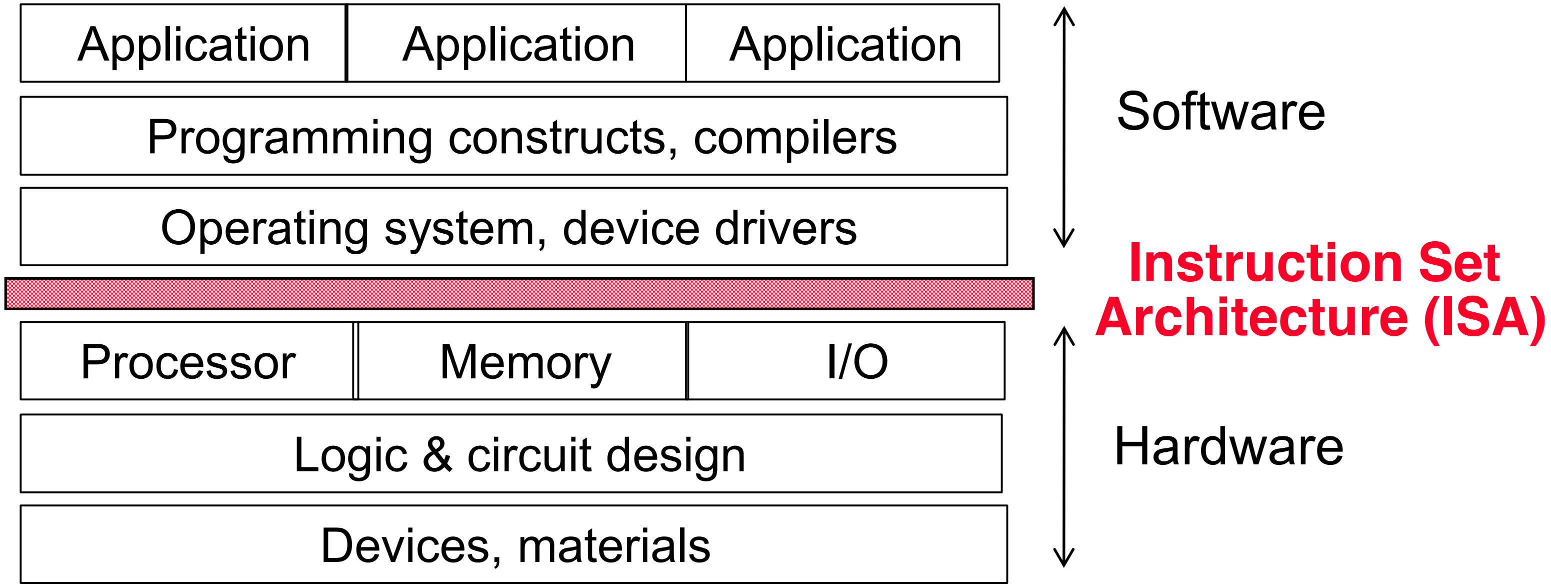
- User-interface devices (Display, keyboard, mouse)
- Storage devices (Hard disk, CD/DVD, flash)
- Network adapters (For communicating with other computers)

datapath + control = processor (CPU)

Abstraction and layering

- Abstraction is the only way to deal with complex systems
  - Divide the processor into components, each with an
    - Interface: inputs, outputs, behaviors
    - Implementation: “black box” with timing information
- Layering the abstractions makes life even simpler
  - Divide the components into layers
    - Implement layer X using the interfaces of layer X-1
    - Don’t need to know the interfaces of layer X-2 (but sometimes it helps)
- Two downsides to layering
  - Inertia: layer interfaces become entrenched over time (“standards”) which are very difficult to change even if the benefit is clear
  - Opaque: can be hard to reason about performance

Abstraction and layering in architecture



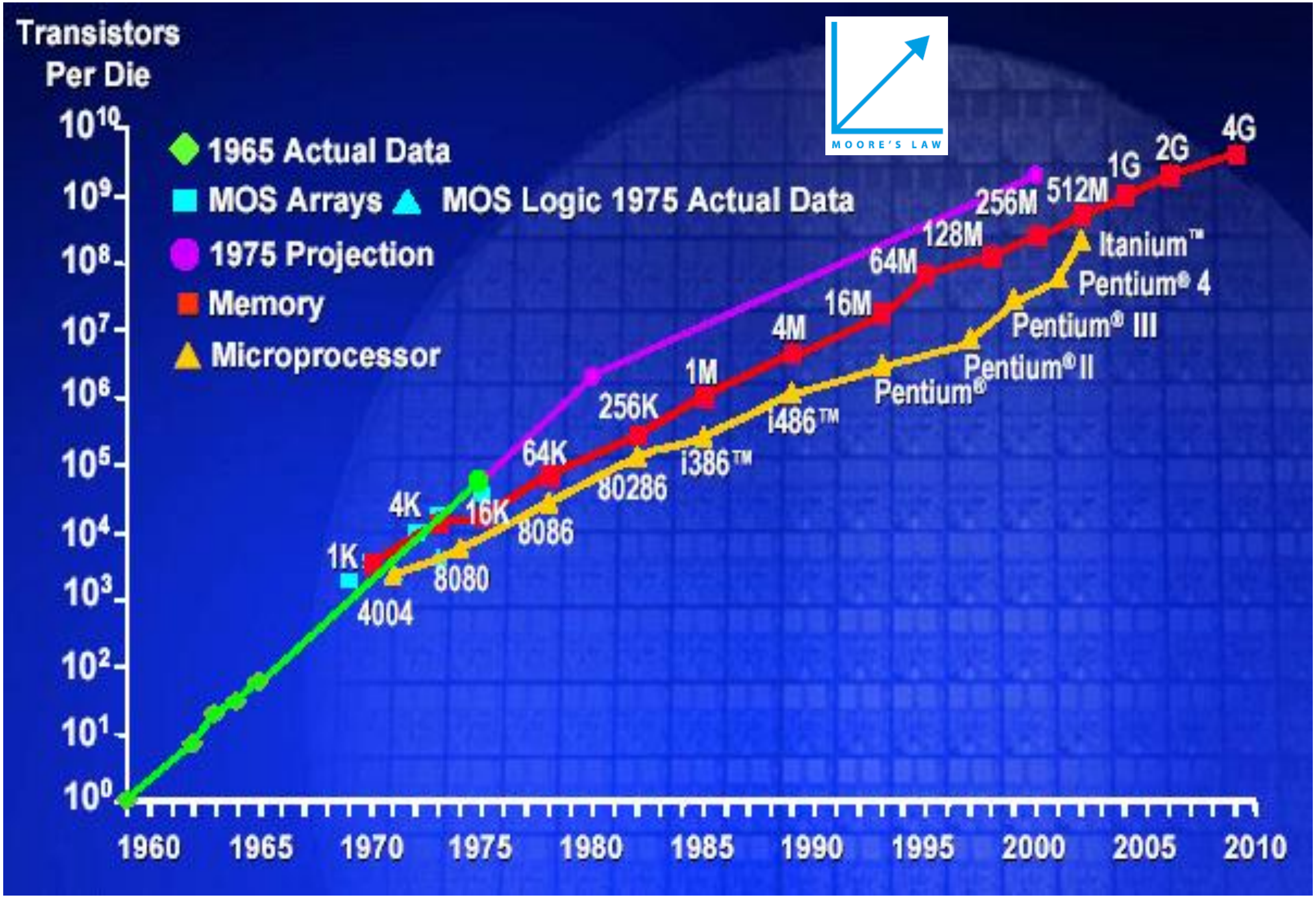
- ❑ The ISA serves at the boundary between the software and hardware
  - Facilitates the parallel development of the software layers and the hardware layers
  - Lasts through many generations (portable)

Instruction Set Architecture (ISA)

- **ISA**, or simply architecture – the abstract interface between the hardware and the lowest level software that encompasses all the information necessary to write a machine language program, including instructions, registers, memory access, I/O, …
  - Enables implementations of varying cost and performance to run identical software

- **ABI** (application binary interface) – the user portion of the instruction set (the ISA) plus the operating system interfaces used by application programmers
  - Defines a standard for binary portability across computers

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

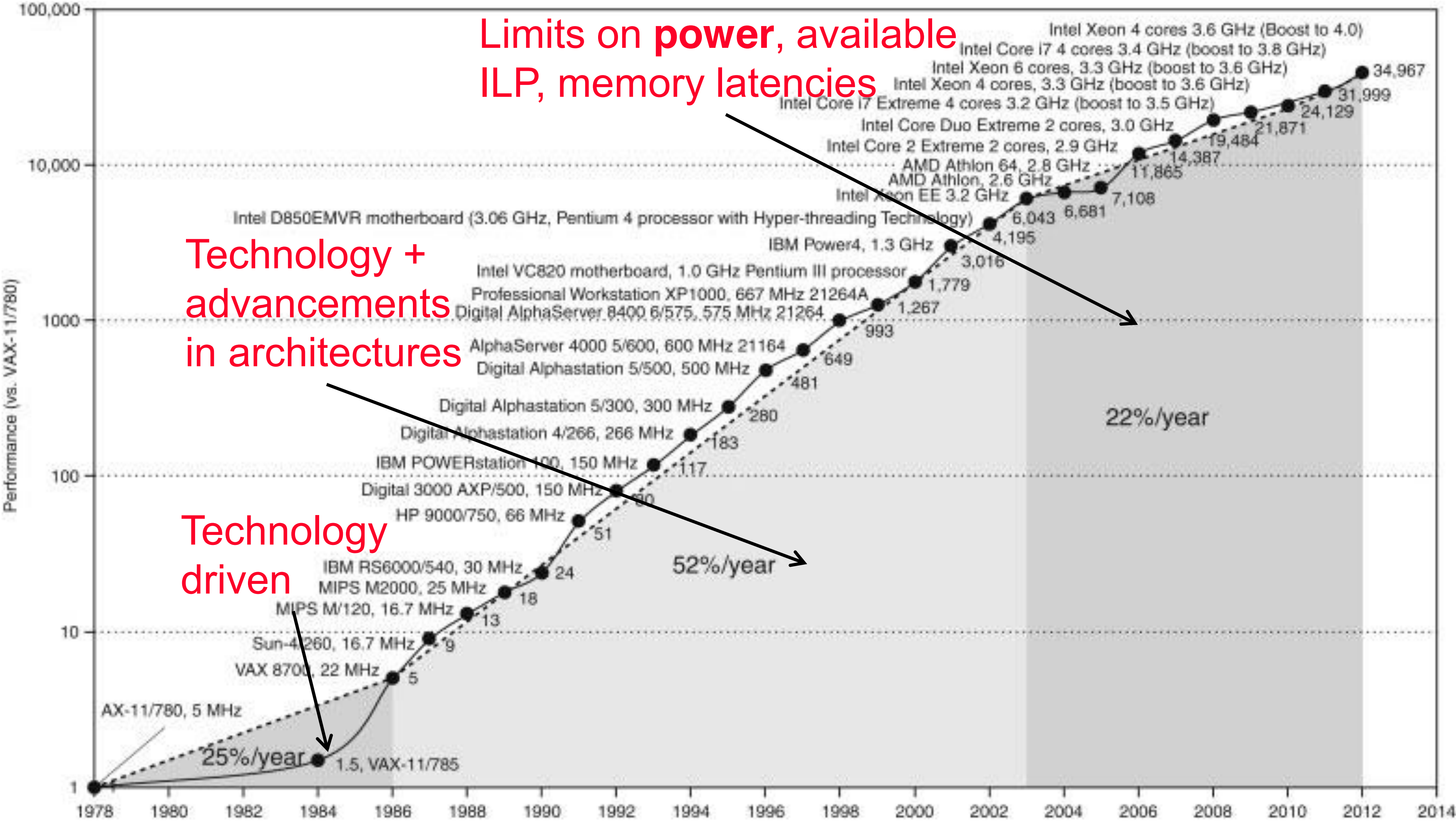


Technology Scaling Road Map (ITRS)

Year	2008	2010	2012	2014	2016
Feature size (nm)	45	32	22	18	15.3

- Fun facts about 45nm transistors
  - 30 million can fit on the head of a pin
  - You could fit more than 2,000 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 1 cent

Growth in Processor Performance (SPECint)

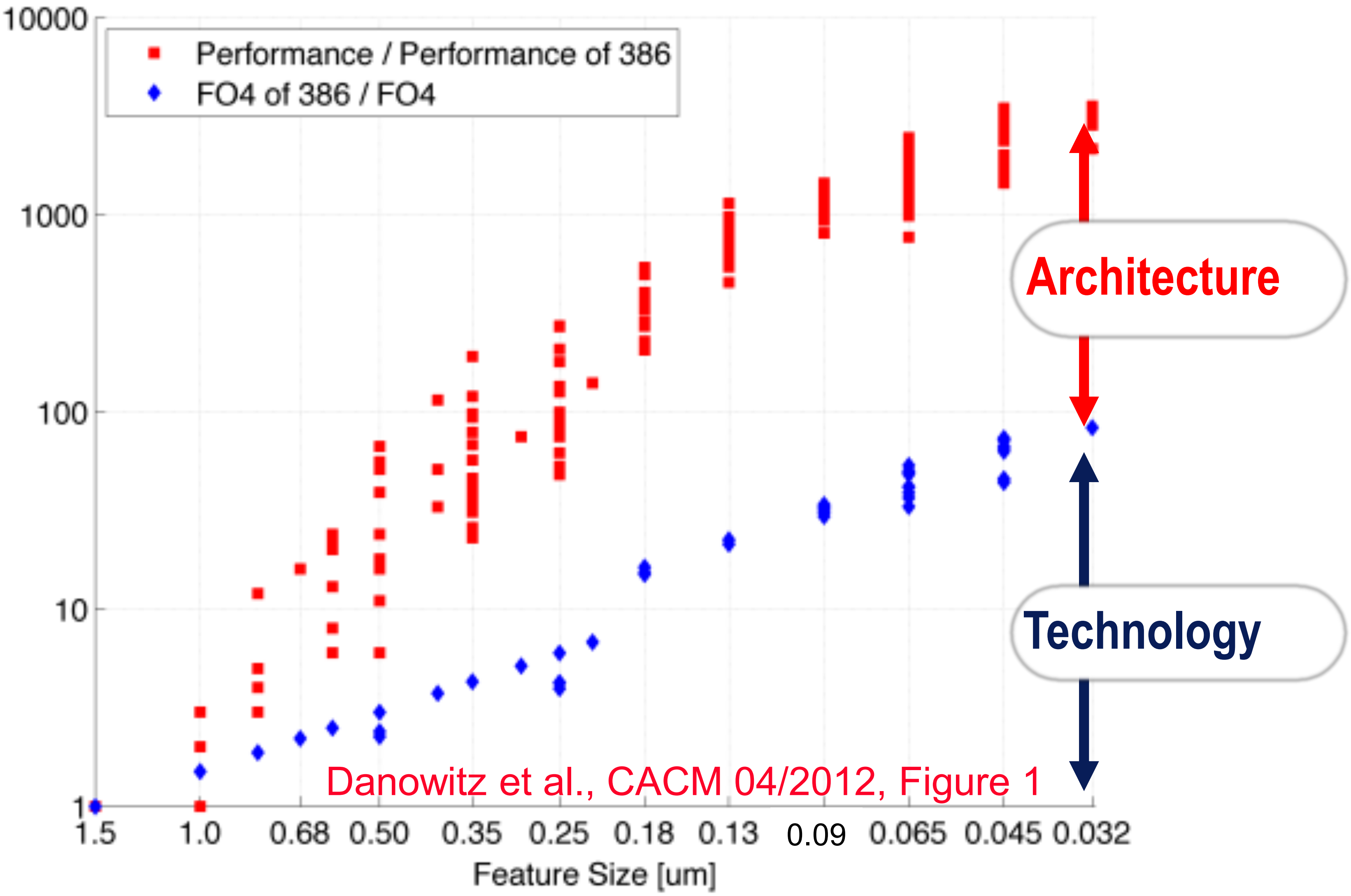


Limits on **power**, available ILP, memory latencies

Technology + advancements in architectures

Technology driven

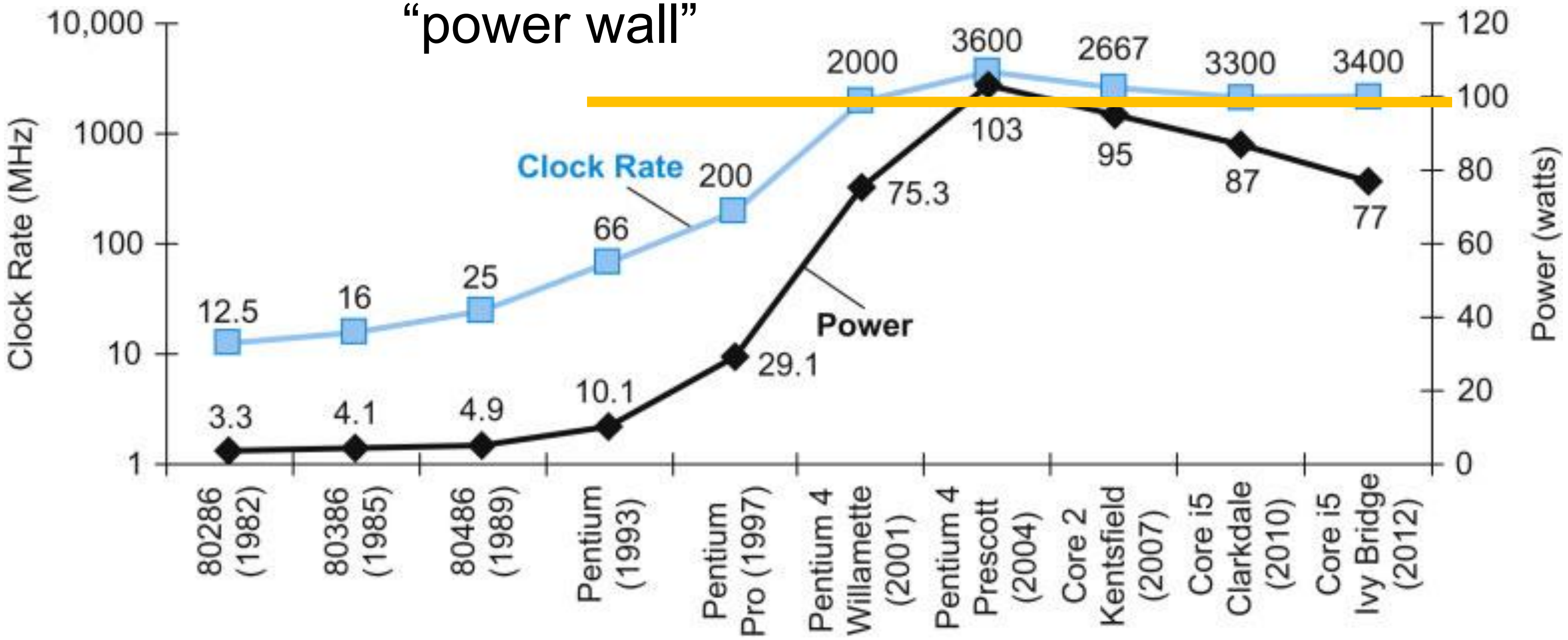
Technology + Architecture = Performance



Danowitz et al., CACM 04/2012, Figure 1

But What Happened to Clock Rates and Why?

Clock rates hit a "power wall"



- ❑ In CMOS IC technology

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

??? 5V → 1V x272

能耗=电容性负载*电压2*变换频率

Reducing Power

- Suppose a new CPU has
  - 85% of the capacitive load of the previous generation
  - 15% voltage reduction, 15% slower clock

$$\frac{P_{new}}{P_{old}} = \frac{(C_{old} \times 0.85) \times (V_{old} \times 0.85)^2 \times (F_{old} \times 0.85)}{C_{old} \times V_{old}{}^2 \times F_{old}} = 0.52$$

- We have hit the power wall
  - We can’t reduce the supply voltage much further (Dennard scaling is over), or the capacitive load
  - We can’t remove more heat without new cooling technologies (e.g., liquid cooled)

- How can we increase the performance while lowering (or keeping the same) clock rate?

The Move to Multicore Processors

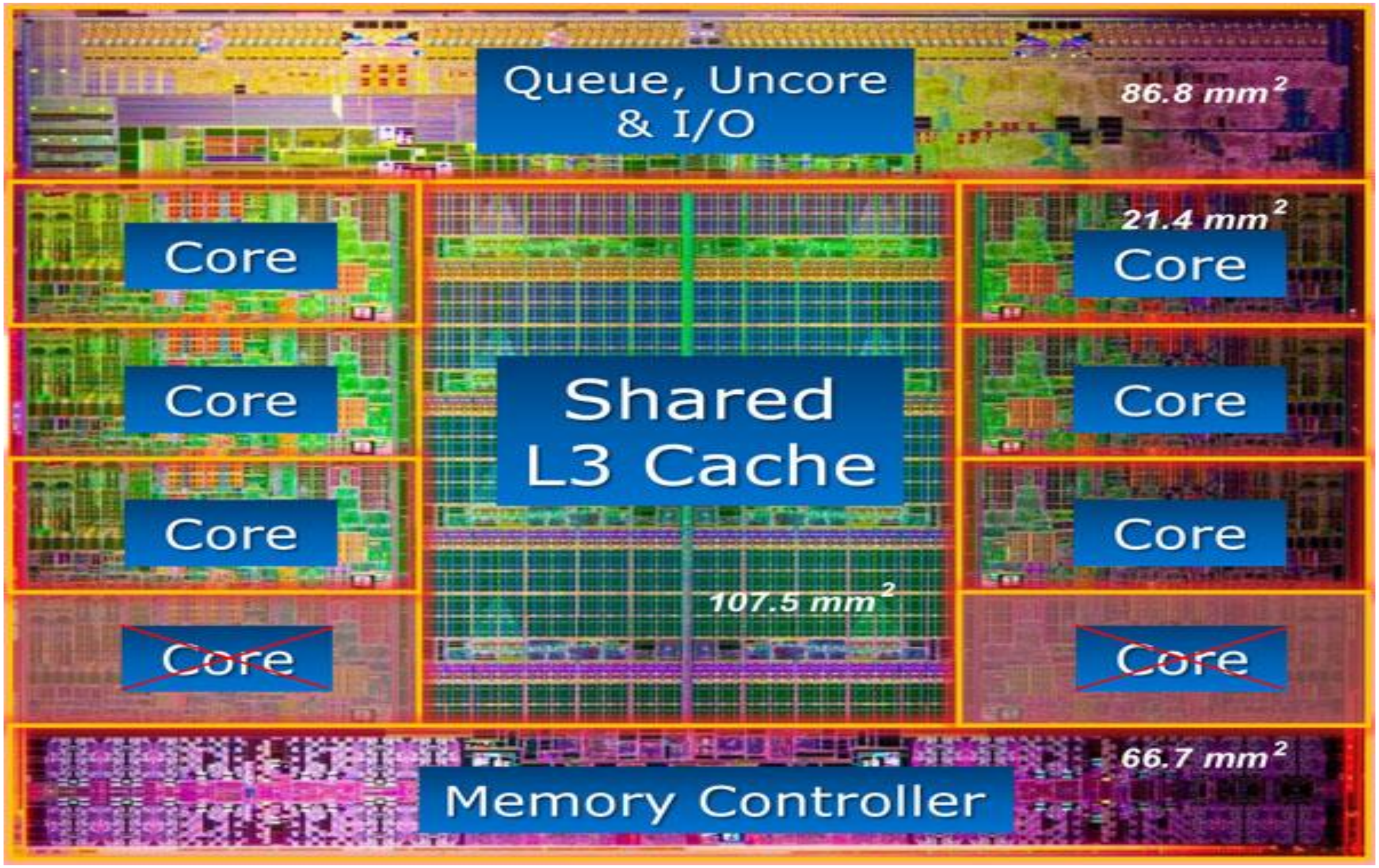
- The power challenge has forced a change in the design of microprocessors

- As of 2006 all server companies were shipping microprocessors with multiple cores per chip (processor)

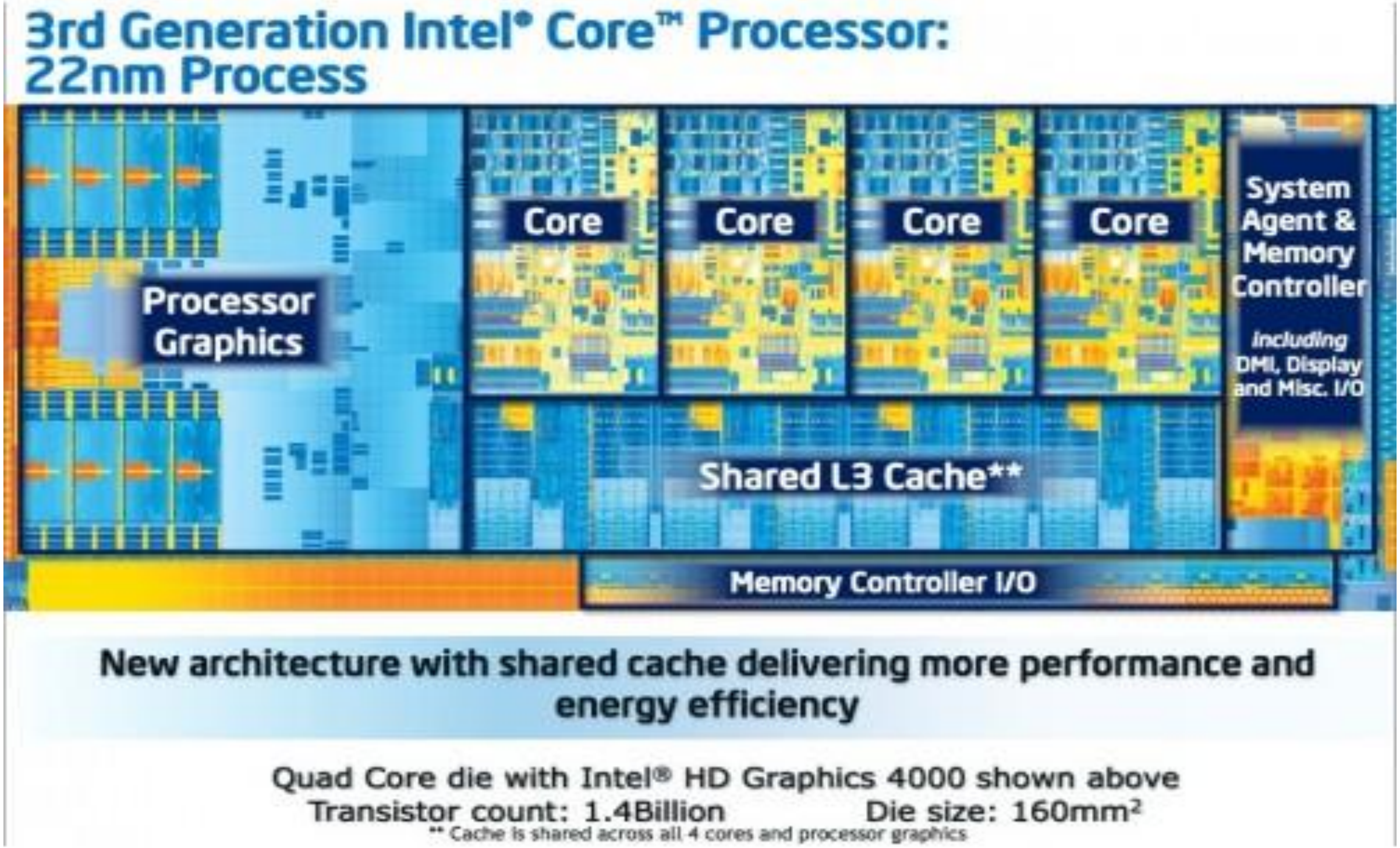
Product	AMD Opteron X	Intel i7 Haswell	IBM Power 7+
Release date	2013	2013	2012
Technology	28nm bulk	22nm FFET	32nm SOI
Cores/Clock	4/2.0 GHz	4/3.5GHz	8/4.4 GHz
Power (TDP)	22W	84W	~120 W

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

Intel SandyBridge



Intel IvyBridge



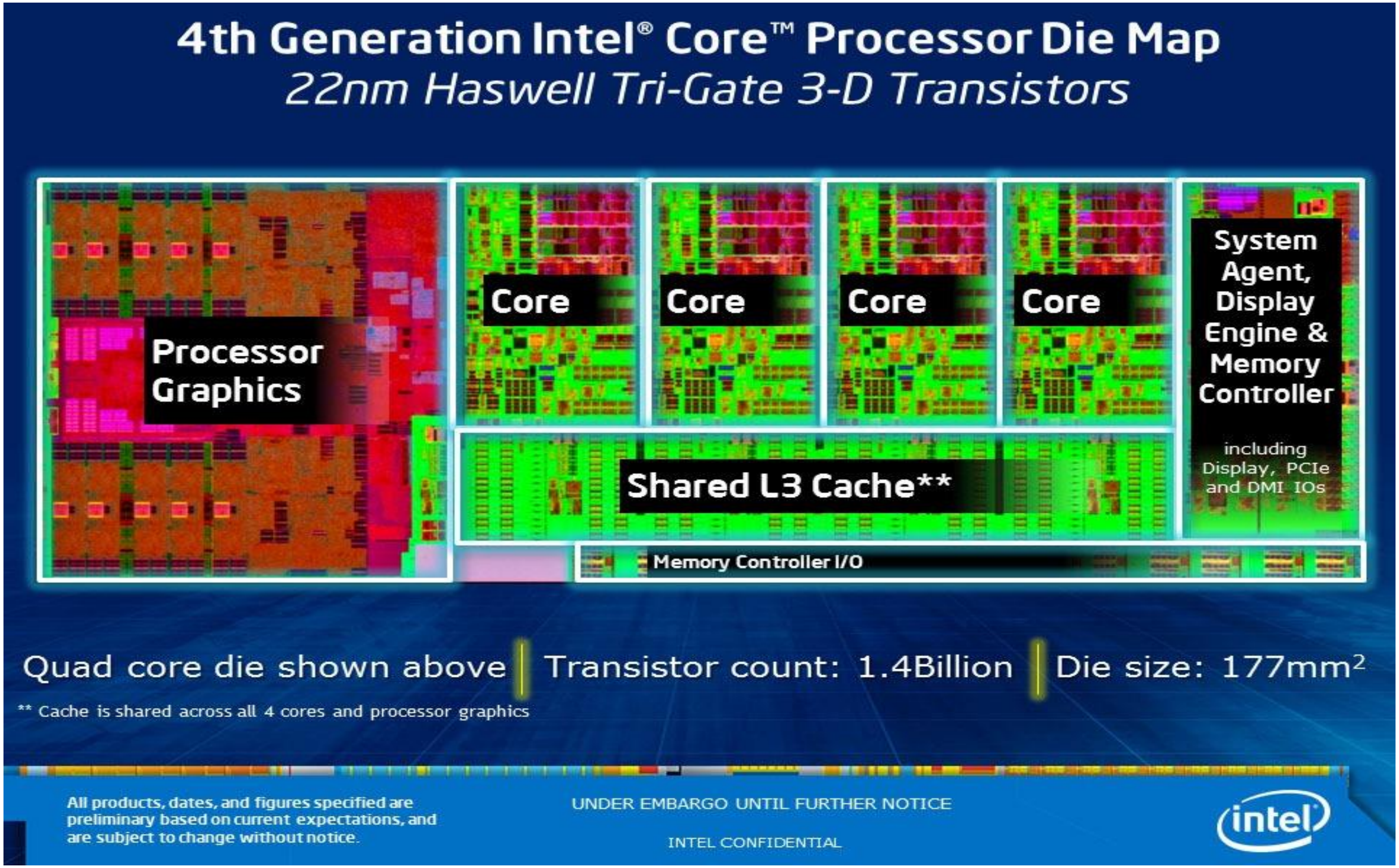
3rd Generation Intel® Core™ Processor:
22nm Process

New architecture with shared cache delivering more performance and energy efficiency

Quad Core die with Intel® HD Graphics 4000 shown above
Transistor count: 1.4Billion Die size: 160mm^2

** Cache is shared across all 4 cores and processor graphics

Intel Haswell



4th Generation Intel® Core™ Processor Die Map

22nm Haswell Tri-Gate 3-D Transistors

Processor Graphics
Core
Core
Core
Core
System Agent, Display Engine & Memory Controller
including Display, PCIe and DMI IOs
Shared L3 Cache**
Memory Controller I/O

Quad core die shown above | Transistor count: 1.4Billion | Die size: 177mm^2

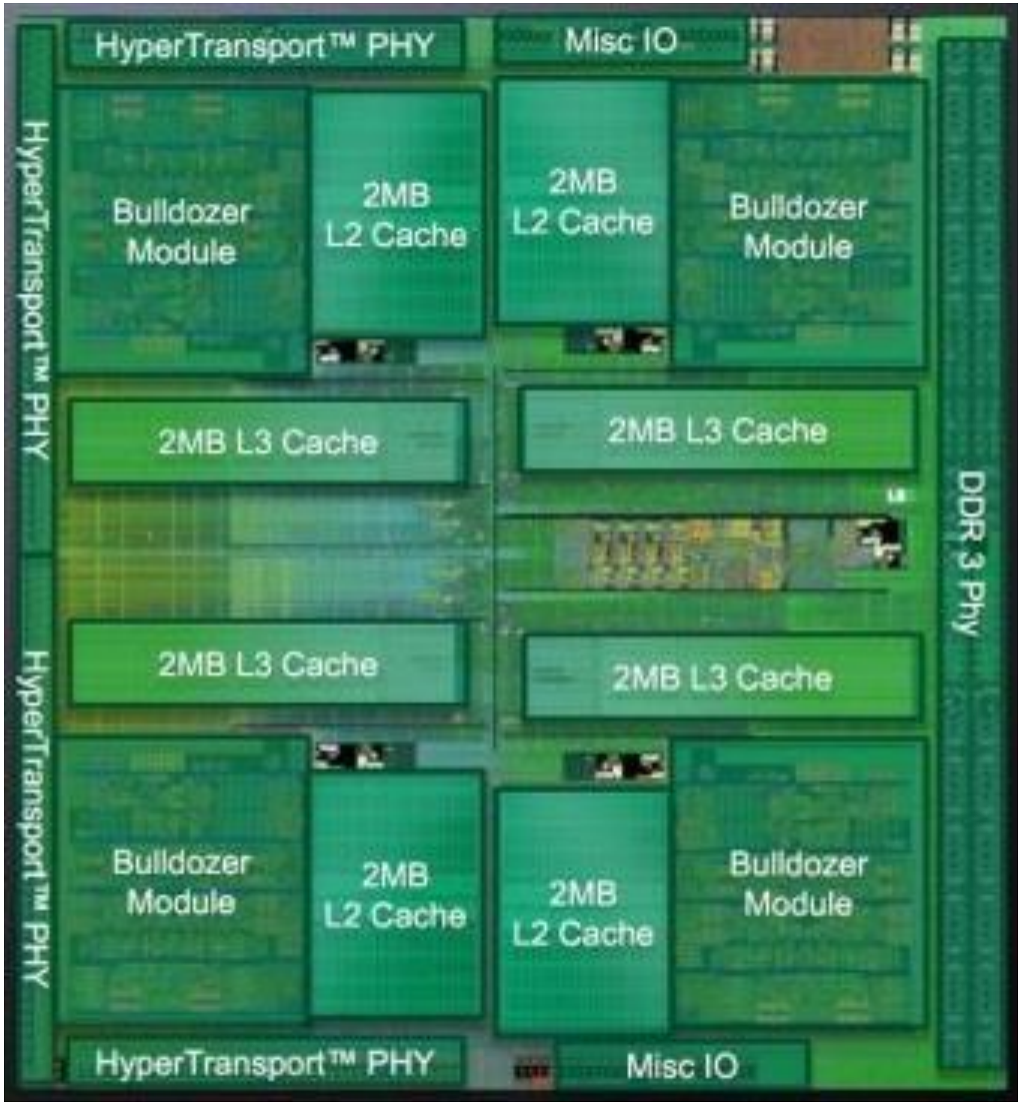
** Cache is shared across all 4 cores and processor graphics

All products, dates, and figures specified are preliminary based on current expectations, and are subject to change without notice.

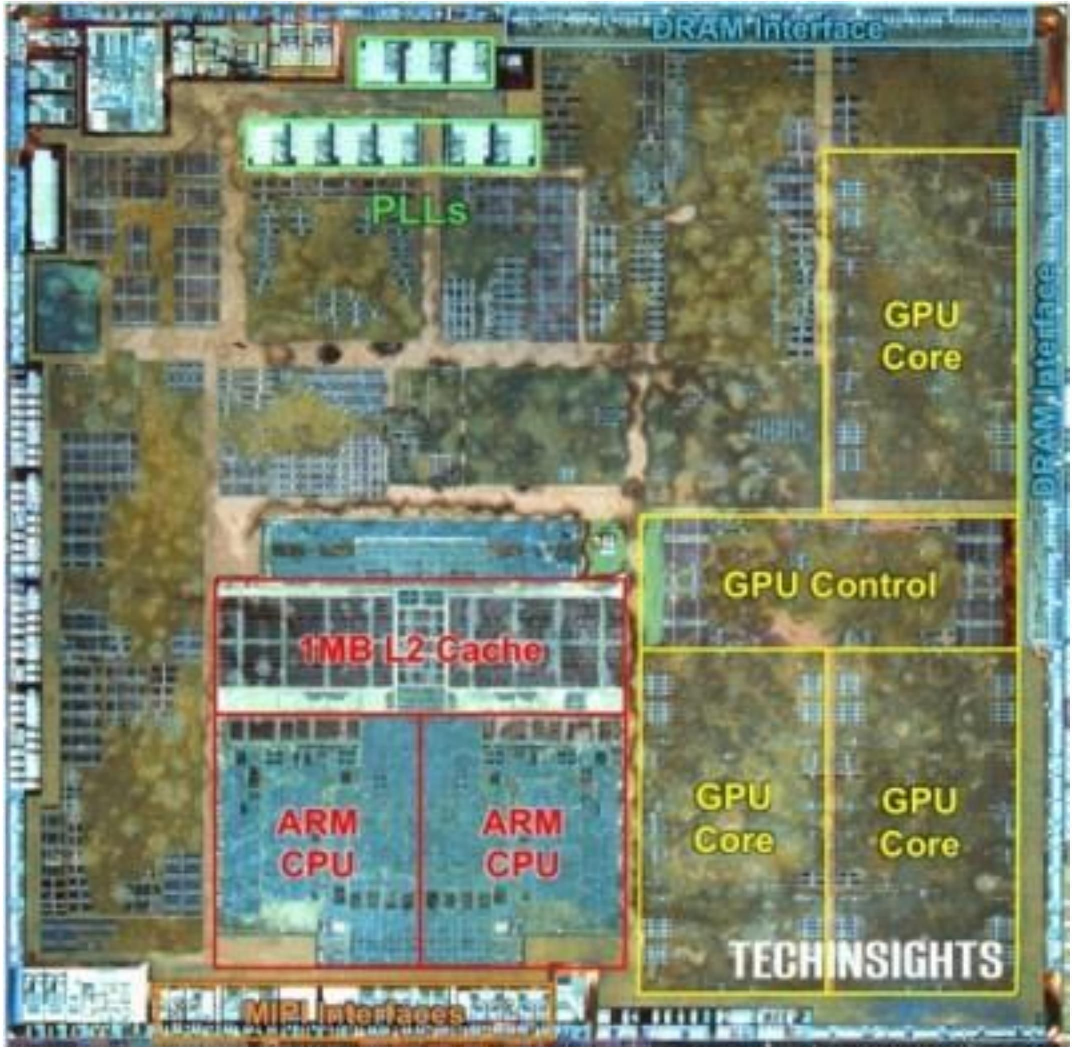
UNDER EMBARGO UNTIL FURTHER NOTICE

INTEL CONFIDENTIAL

AMD Opteron (Bulldozer)



Apple A6 (iPhone 5)



Multicore Performance Issues

- Private L1 caches, private or shared L2, … LL caches?
  - Best performance depends upon the applications and how much information they "share" in the cache, or how much they conflict in the cache
- Contention for memory controller(s) and port(s) to DRAM
- Requires explicitly parallel programming (multiple (parallel) threads for one application) – CmpSc 497D, CmpSc 450 (Spr15), CSE531, CSE521
  - Compare with instruction level parallelism (ILP) where the hardware executes multiple instructions at once (so hidden from the programmer)
  - Parallel programming for performance is hard to do
    - Load balancing across cores
    - Cache sharing/contention, contention for DRAM controller(s)
    - Have to optimize for thread communication and synchronization

Defining Performance

响应时间：从提交任务到完成任务所需要的时间。

- ❑ Response time (execution time) – how long does it take to do a task
  - ● Important to individual users

吞吐率：一定时间内完成的工作量。

- ❑ Throughput (bandwidth) – number of tasks completed per unit time
  - ● Important to data center managers

- ❑ How are response time and throughput affected by
  1. Replacing the core with a faster version?
  2. Adding more cores?

- ❑ Our focus, for now, will be response time

Relative Performance

- To maximize performance, need to minimize execution time

$$\text{performance}_X = 1 / \text{execution\_time}_X$$

If computer X is n times faster than computer Y, then

$$\frac{\text{performance}_X}{\text{performance}_Y} = \frac{\text{execution\_time}_Y}{\text{execution\_time}_X} = n$$

一般地，降低响应时间可以提高吞吐率。

- Decreasing response time almost always improves throughput

Relative Performance Example

- If computer A runs a program in 10 seconds and computer B runs the same program in 15 seconds, how much faster is A than B?

We know that A is n times faster than B if

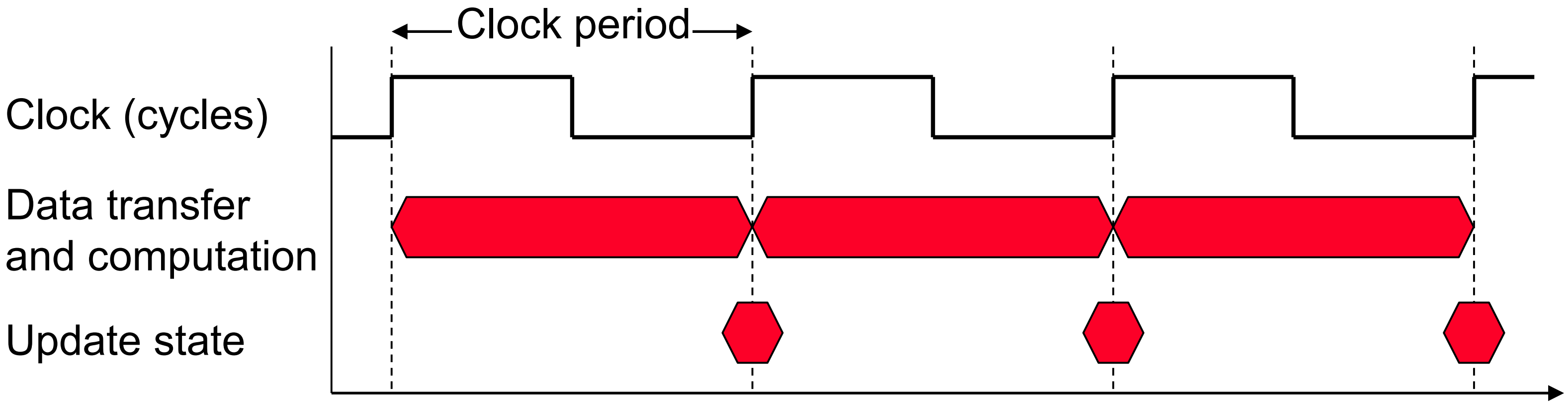
$$\frac{\text{performance}_A}{\text{performance}_B} = \frac{\text{execution\_time}_B}{\text{execution\_time}_A} = n$$

The performance ratio is $$\frac{15}{10} = 1.5$$

So A is 1.5 times faster than B (or A is 50% faster than B!)

CPU Clocking

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



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

Performance Factors

- CPU execution time (CPU time) – time the CPU spends working on a task (not including time waiting for I/O or running other programs)

$$\begin{array}{c}\text{CPU execution time}\\\text{for a program}\end{array} = \begin{array}{c}\text{\# CPU clock cycles}\\\text{for a program}\end{array} \text{x} \;\; \text{clock cycle time}$$

or

$$\begin{array}{c}\text{CPU execution time}\\\text{for a program}\end{array} = \frac{\text{\# CPU clock cycles for a program}}{\text{clock rate}}$$

- Can improve performance by reducing either the length of the clock cycle (increasing clock rate) or reducing the number of clock cycles required for a program
- The architect must often trade off clock rate against the number of clock cycles for a program

Instruction Performance

- Not all instructions take the same amount of time to execute
  - One way to think about execution time is that it equals the number of instructions executed multiplied by the average time per instruction

执行一个程序所需要的时钟周期数　　该程序的指令数　　每条指令的平均时钟周期数

CPU clock cycles for a program = # Instructions for a program x Average clock cycles per instruction

每条指令的平均时钟周期数->CPI

- Clock cycles per instruction (CPI) – the average number of clock cycles each instruction takes to execute
  - A way to compare two different implementations of the same ISA

	Computer_A	Computer_B
Avg CPI	2	1.2

THE Performance Equation

- Our basic performance equation is then

$$\text{CPU time} = \text{Instruction\_count} \times \text{CPI} \times \text{clock\_cycle}$$

or

$$\text{CPU time} = \frac{\text{Instruction\_count} \times \text{CPI}}{\text{clock\_rate}}$$

- This equation separates the three key factors that affect performance
  - Can measure the CPU execution time by running the program
  - The clock rate is usually given
  - Can measure overall instruction count by using profilers/ simulators without knowing all of the implementation details
  - CPI varies by instruction type and ISA implementation for which we must know the implementation details

Average (Effective) CPI

- Computing the overall average CPI is done by looking at the different types of instructions and their individual cycle counts and averaging

$$\text{Overall effective CPI} = \sum_{i=1}^{n} (\text{CPI}_i \times \text{IC}_i)$$

  - Where IC_i is the count (percentage) of the number of instructions of class i executed
  - CPI_i is the number of clock cycles per instruction for that instruction class
  - n is the total number of instruction classes

- The overall effective CPI varies by instruction mix – a measure of the dynamic frequency of instructions for one or many programs

A Simple Performance Tradeoff 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
Average (Effective) CPI			$\sum =$ 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

工作负载 基准测试程序

Workloads and Benchmarks

- Benchmarks – a set of programs that form a “workload” specifically chosen to measure performance

- SPEC (System Performance Evaluation Cooperative) creates standard sets of benchmarks starting with SPEC89. SPEC CPU2006 consists of 12 integer benchmarks (CINT2006) and 17 floating-point benchmarks (CFP2006).

www.spec.org

- There are also benchmark collections for power workloads (SPECpower_ssj2008), for mail workloads (SPECmail2008), for multimedia workloads (mediabench), …

SPEC CINT2006 on Intel i7 (CR = 2.66GHz)

IC：执行第I类的指令条数

Name	ICx10^9	CPI	ExTime (sec)	RefTime (sec)	SPEC ratio
perl	2,252	0.60	508	9,770	19.2
bzip2	2,390	0.70	629	9,650	15.4
gcc	794	1.20	358	8,050	22.5
mcf	**221**	**2.66**	221	9,120	41.2
go	1,274	1.10	527	10,490	19.9
hmmer	**2,616**	0.60	590	9,330	15.8
sjeng	1,948	0.80	586	12,100	20.7
libquantum	659	**0.44**	109	20,720	190.0
h264avc	**3,793**	**0.50**	713	22,130	31.0
omnetpp	**367**	**2.10**	290	6,250	21.5
astar	1,250	1.00	470	7,020	14.9
xalancbmk	1,045	0.70	275	6,900	25.1
Geometric Mean					**25.7**

Comparing and Summarizing Performance

- How do we summarize the performance for benchmark set with a single number?
  - First the execution times are normalized giving the "SPEC ratio" (bigger is faster, i.e., SPEC ratio is the inverse of execution time)
  - The SPEC ratios are then "averaged" using the geometric mean (GM)

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

- Guiding principle in reporting performance measurements is reproducibility – list everything another experimenter would need to duplicate the experiment (version of the operating system, compiler settings/flags, input set used, specific computer configuration (clock rate, cache sizes and speed, memory size and speed, etc.))

Speedup Measurements

- The speedup of the SS core is
  - Assumes the cores have the same IC & CC

$$\text{speedup} = s_n = \frac{\#\ \text{scalar cycles}}{\#\ \text{superscalar cycles}}$$

- To compute average speedup performance can use
  - Geometric mean

$$GM = \sqrt[n]{\prod_{i=1}^{n} s_i}$$

  - Harmonic mean

$$HM = n / \left(\sum_{i=1}^{n} 1/s_i \right)$$

    - assigns a larger weighting to the programs with the smallest speedup
  - EX: two programs with same scalar cycles, with a SS speedup of 2 for program1 and 25 for program2
    - GM = $\sqrt{(2 * 25)} = 7.1$
    - HM = $2 / (.5 + .04) = 2 /.54 = 3.7$

Amdahl’s Law

- Used to determine the maximum expected improvement to overall system performance when only part of the system is improved

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

- How much faster must the multiplier be to get a 2x performance improvement overall if multiples account for 20 seconds of the 100 second run time?

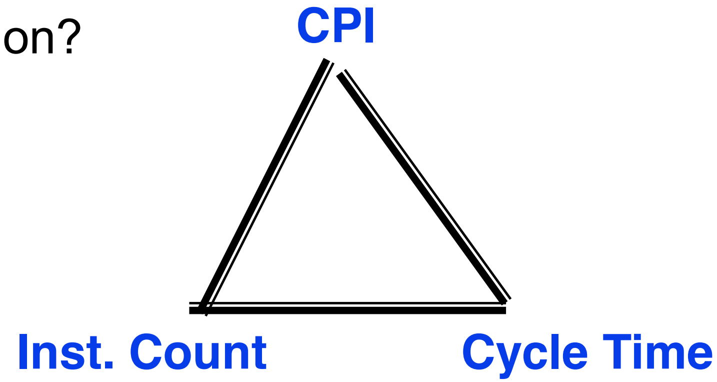
- Corollary: Make the common case fast

Summary: Evaluating ISAs

- Design-time metrics
  - Can it be implemented, in how long, at what cost (size, power)?
  - Can it be programmed? Ease of compilation?
- Static Metrics
  - How many bytes does the program occupy in memory?
- Dynamic Metrics
  - How many instructions are executed? How many bytes does the corefetch to execute the program?
  - How many clocks are required per instruction?
  - How "lean" (fast) a clock is practical?

Best Metric: Time to execute the program!

depends on the instructions set, the processor organization, and compilation techniques.


CPI
Inst. Count
Cycle Time