CMPEN 431
Computer Architecture
Fall 2018

Chapter 1: Abstractions and Technology

Jack Sampson (www.cse.psu.edu/~sampson)

[Slides adapted from work by Mary Jane Irwin, in turn adapted from *Computer Organization and Design, 5th Edition*, Patterson & Hennessy, © 2014, Morgan Kaufmann]

Course Organization

- **Instructor:** Jack Sampson sampson@cse.psu.edu
  (please use Canvas e-mail for class-related mail)
  W324 Westgate Bldg
  Office Hrs: Tu/Th 2:30-4:00pm
  Instructor-in-training: Xulong Tang OH: TBD
- **TAs:** Prashanth Thinakaran OH: TBD
- **Labs:** Accounts on machines in W204
  (Dells running RedHat Linux on Intel cores)
- **CMS**: CANVAS
- **Text:** Suggested:
  Computer Organization and Design, 5th Edition,
  Patterson & Hennessy, © 2014, Morgan Kaufmann

Grading Information

- Exams **<u>in-class</u>**
  - **3** exams, all in-class
  - No Final exam; Final project

- Let me know about exam conflicts ASAP
  - Dates posted on syllabus

- Grades will be posted on Canvas

% of grade	**# of assessments**	**Category**
5	7	Practice Exercises
5	10	Online Quizzes
6	3	Written Reports
20	3	Projects
64	3	Exams

Course Structure & Schedule

- Design focused class
  - ISA Emulation and Cache simulation (coding projects)
  - Microarchitecture Design Space Exploration using SimpleScalar
- Topics:
  - 1 week overview; what do computer architects care about?
  - 1 week review of the MIPS ISA and basic architecture (331)
  - 1 week cache optimizations
  - 2 weeks dependencies & scalar pipelined datapath design
  - 1.5 weeks superscalar datapath design issues
  - 1.5 weeks dynamic scheduling and SMT
  - 1 weeks multiprocessor/multicore design issues
  - 2 weeks power efficiency / mobile design / future arch
  - 3 weeks exams / reviews / make-up

Course Content

- Memory hierarchy and design, CPU design, pipelining, multiprocessor architecture.
  - “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.”
    - 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 – 270, 331 (& 311 helps)

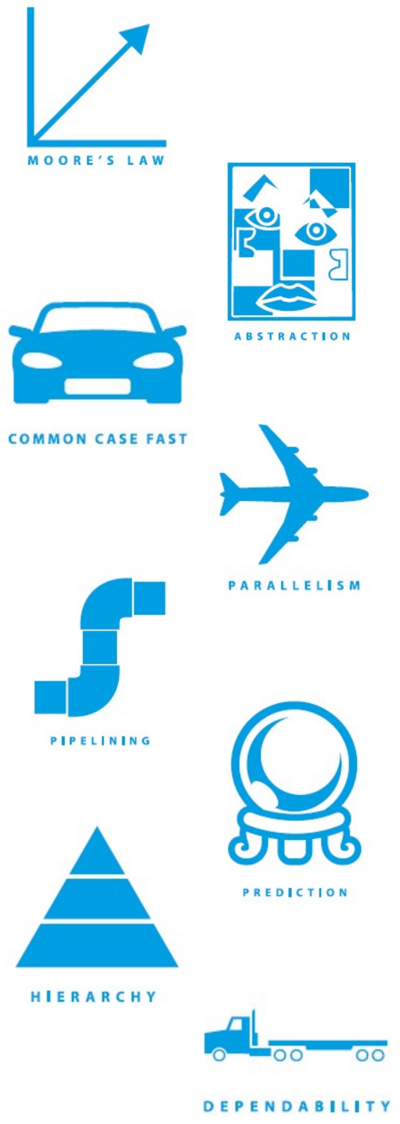
- Basic logic design & machine organization
  - logical minimization, FSMs, component design
  - processor, memory, I/O
- Create, assemble, run, debug programs in an assembly language
  - MIPS preferred
- Create, simulate, and debug hardware structures in a hardware description language
  - VHDL or verilog
- Create, compile, and run C (C++, Java) programs
- Create, organize, and edit files and run programs on Unix/Linux
- See https://lynda.psu.edu/ if you need a refresher/extension of specific skill practice

Academic Integrity & Other Policies

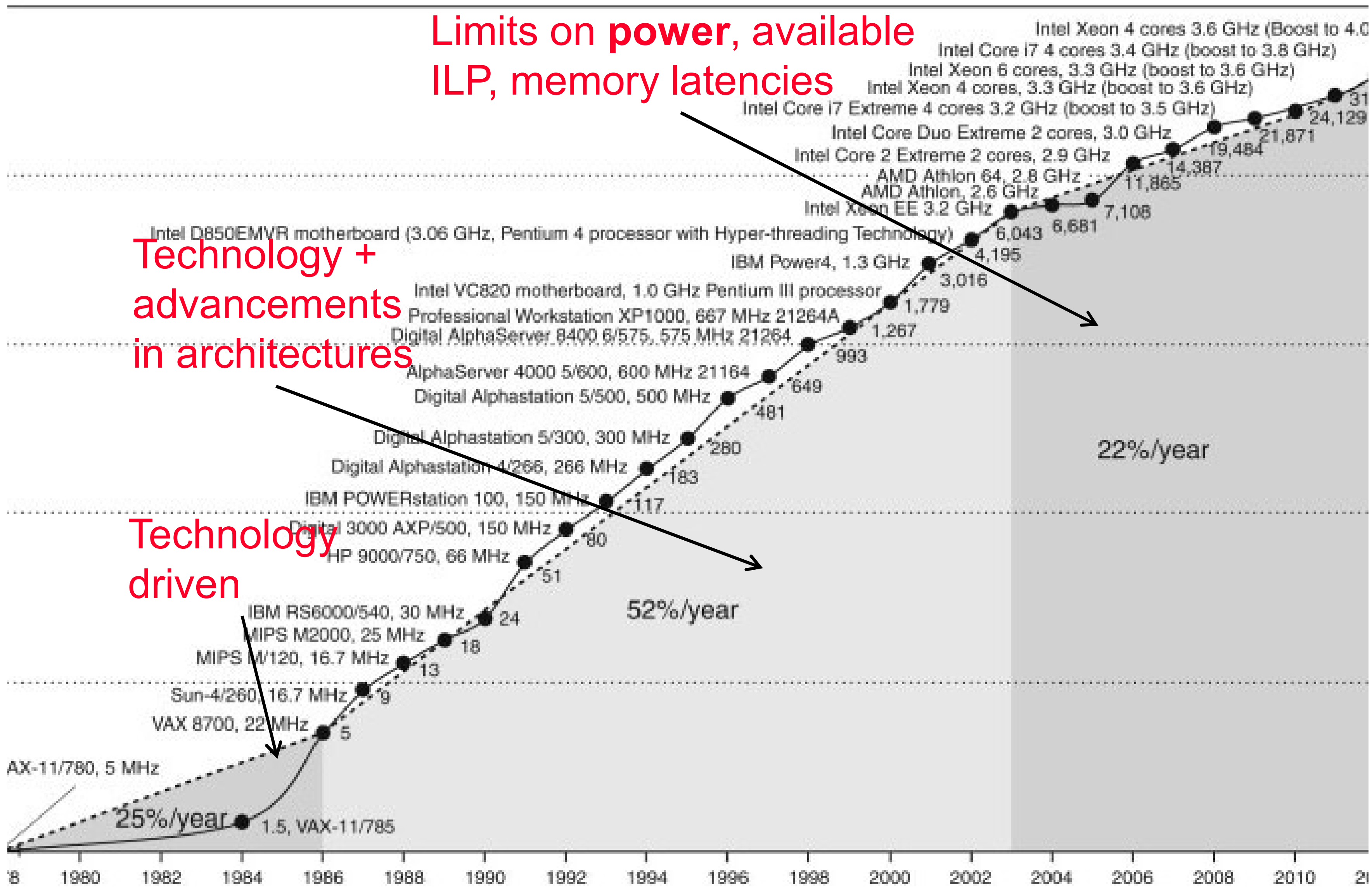
- Please read the relevant policy section in the syllabus
- No, really, actually read the syllabus
- … the whole thing

Eight Great Ideas in Computer Architecture

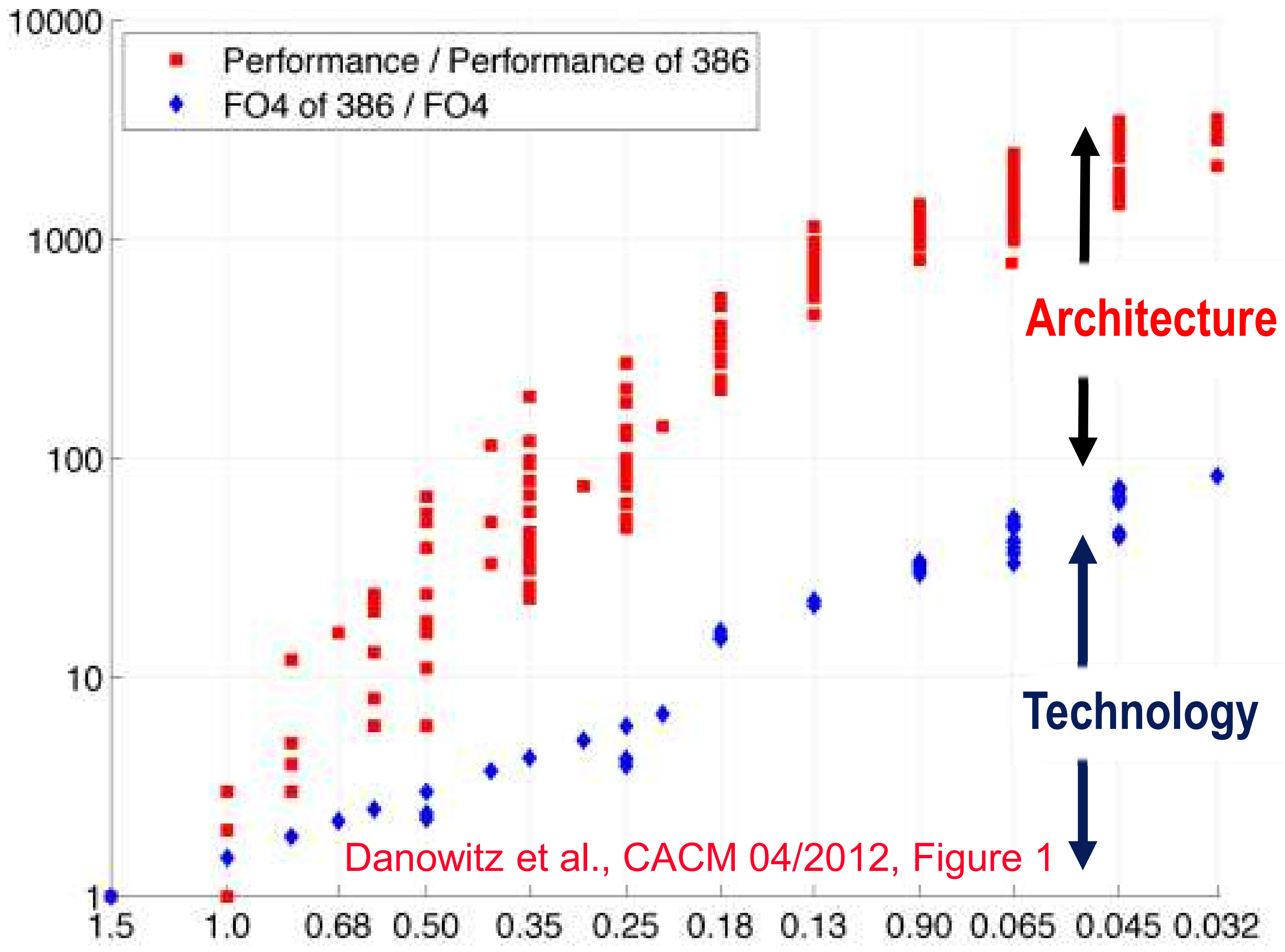
- Design for ***Moore's Law***
- Use ***abstraction*** to simplify design
- Make the ***common case fast***
- Performance *via* ***parallelism***
- Performance *via* ***pipelining***
- Performance *via* ***prediction***
- ***Hierarchy*** of memories
- ***Dependability*** *via* redundancy



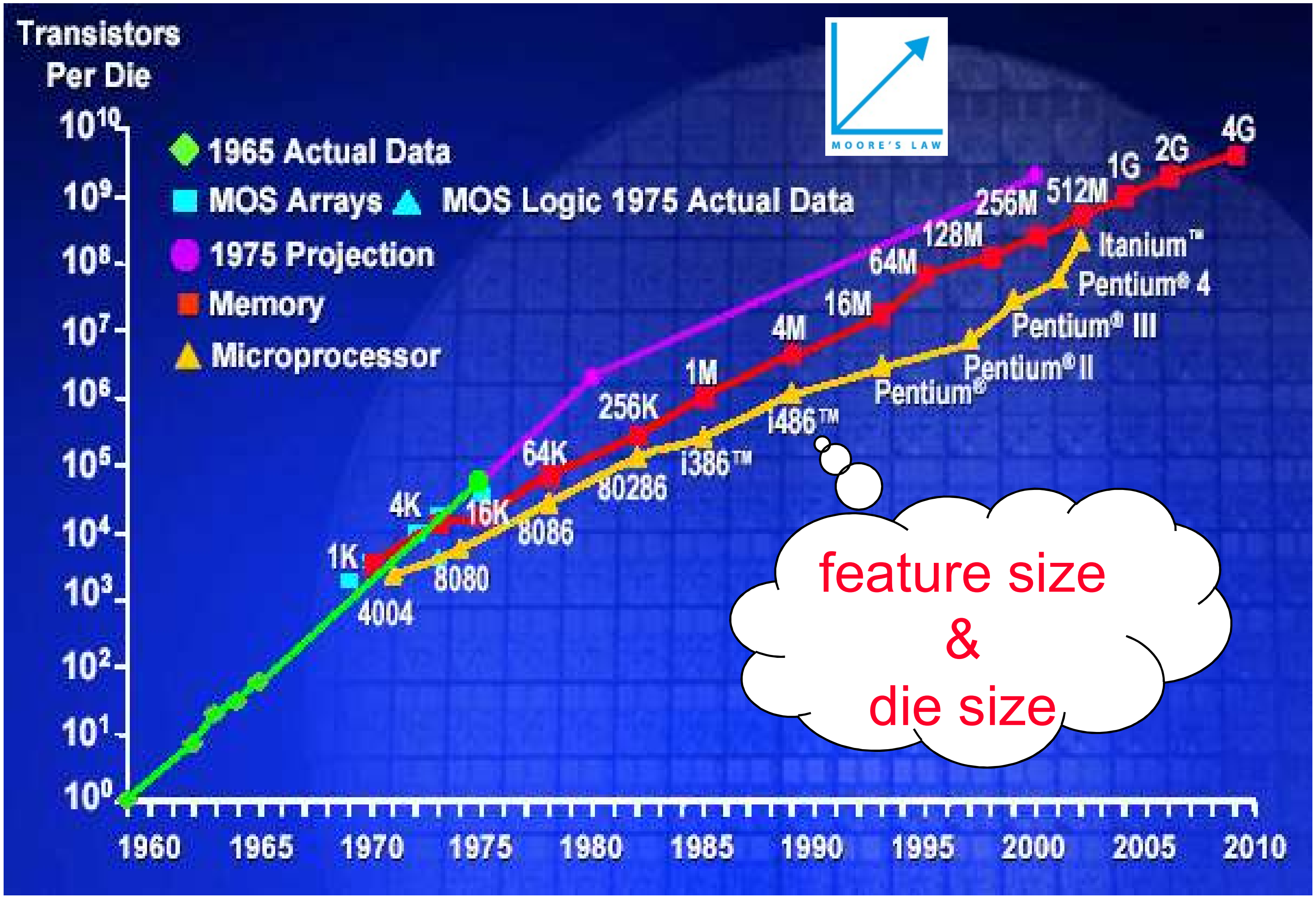
Growth in Processor Performance (SPECint)



How much benefit from whom?



Moore’s Law: 2X transistors / “2 years”



Technology Scaling Road Map (ITRS)

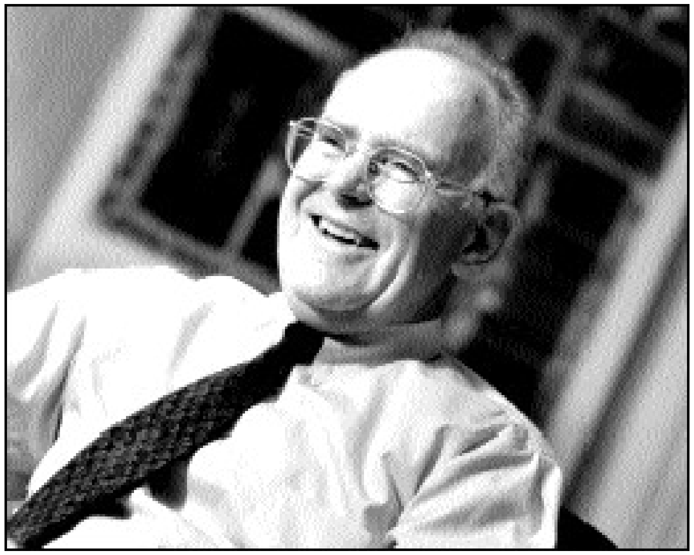
Year	2008	2010	2012	2014	2018
Feature size (nm)	45	32	22	14	10?
Intg. Capacity (BT)	0.5	1	2	4	8

- 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

Scaling 101: Moore's Law

$$\begin{tikzcd} 90 \arrow[r] & 65 \arrow[r] & 45 \arrow[r] & 32 \arrow[r] & 22 \arrow[r] & 16 \arrow[r] & 11 \arrow[r] & 8 \arrow[r] & \text{nm} \end{tikzcd}$$

$$\underbrace{22 \rightarrow 16}_{S = \frac{22}{16} = \sim 1.4x}$$



Scaling 101: Moore's Law

- Transistor count scales as S^2

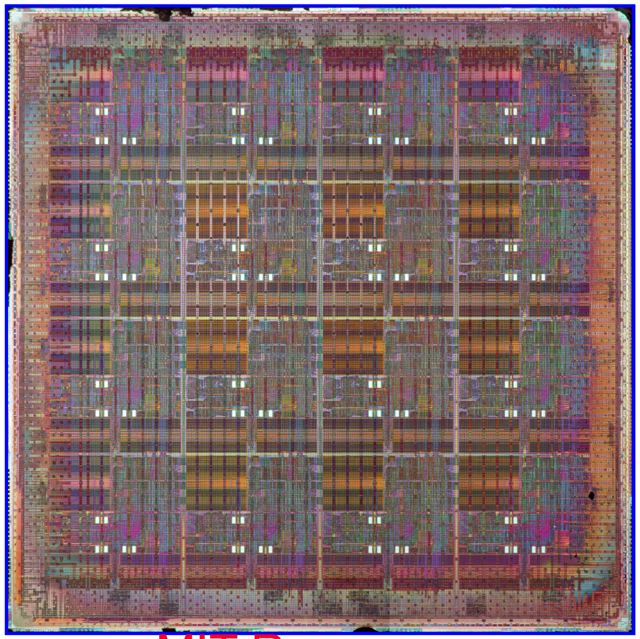
180 nm

16 cores

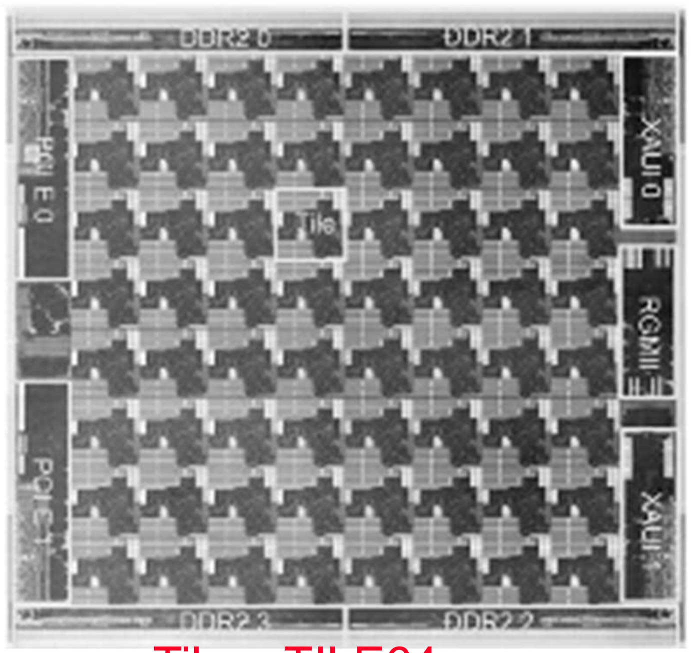
S = 2x
Transistors = 4x

90 nm

64 cores


MIT Raw

$\longrightarrow$



Tilera TILE64

Advanced Scaling: ***Dennard -***

"Computing Capabilities Scale by S^3 = 2.8x"

If S=1.4x …

S^3

S^2

S

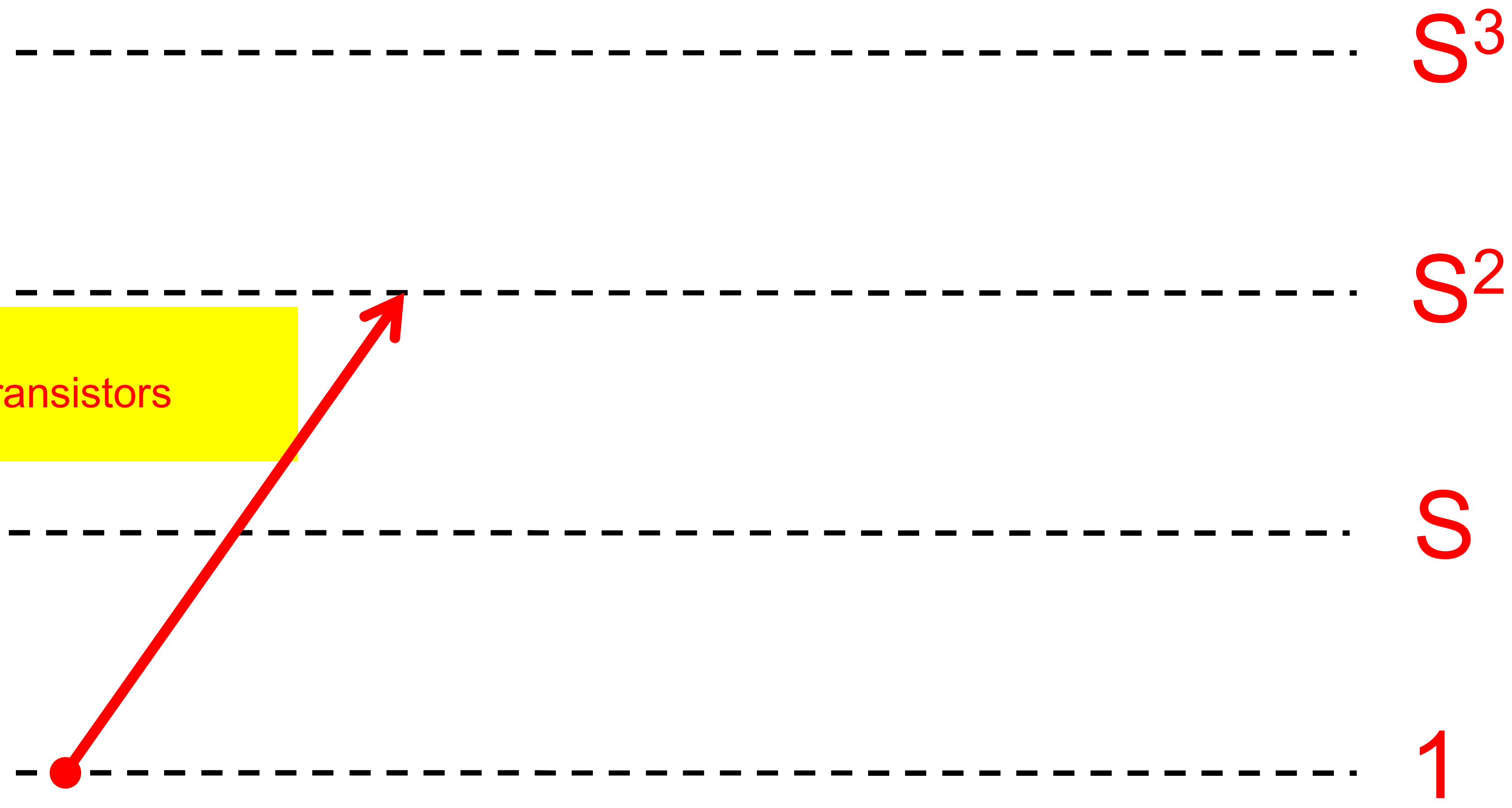
1

Design of Ion-Implanted MOSFETs with Very Small Dimensions
Dennard et al, 1974

Advanced Scaling: ***Dennard -***

"Computing Capabilities Scale by S^3 = 2.8x"

If S=1.4x …

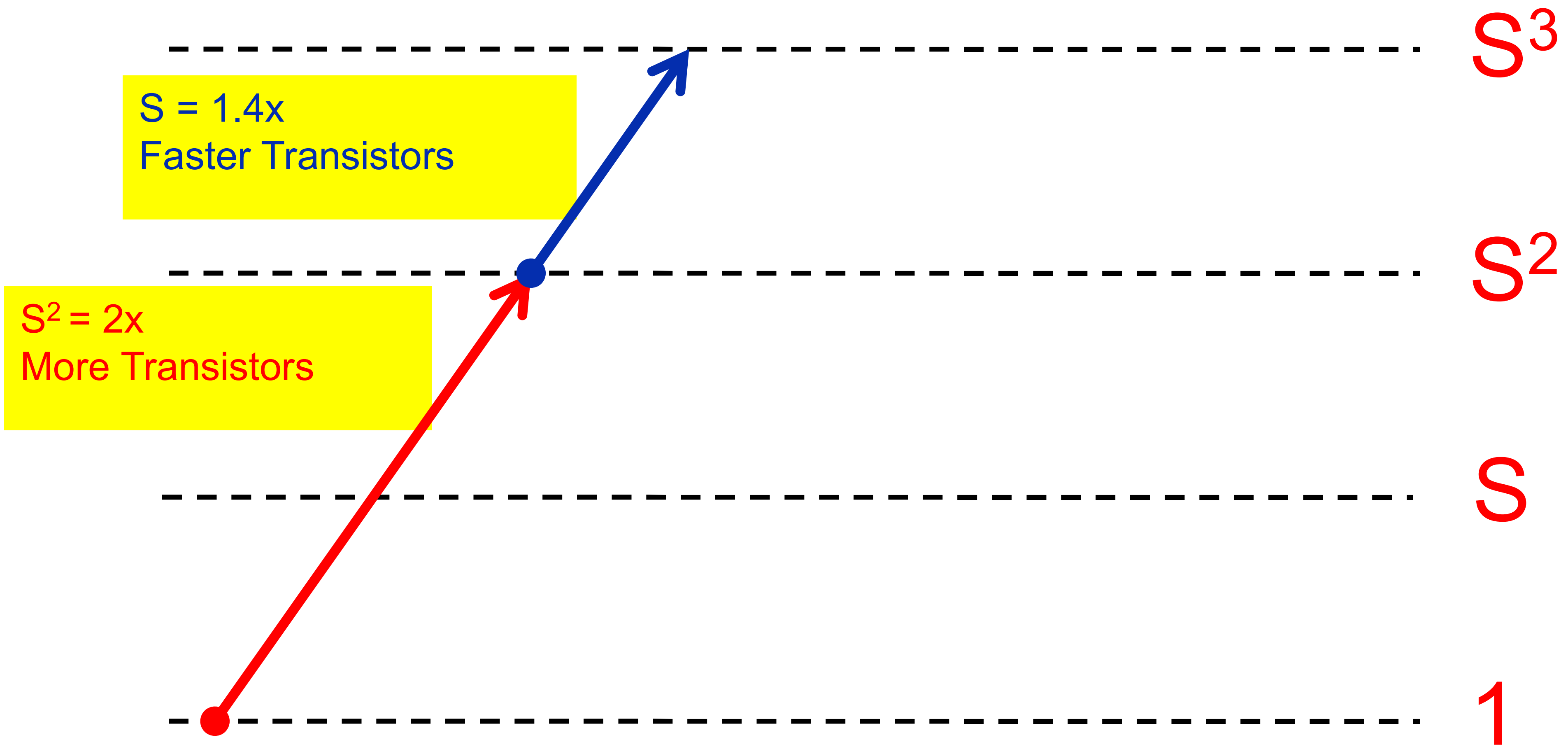


S^2 = 2x
More Transistors

Advanced Scaling: ***Dennard -***

"Computing Capabilities Scale by S^3 = 2.8x"

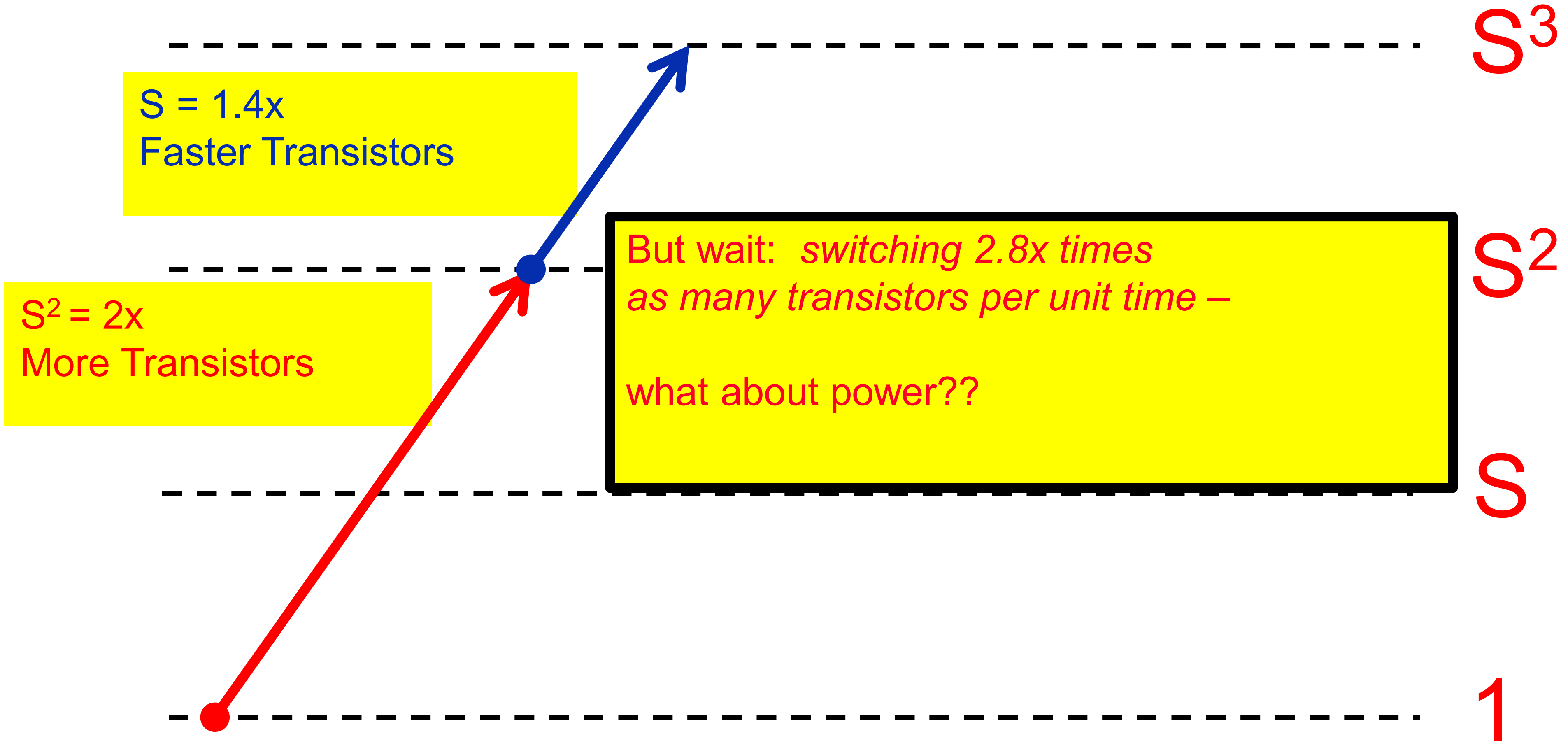
If S=1.4x …



Advanced Scaling: ***Dennard -***

"Computing Capabilities Scale by S^3 = 2.8x"

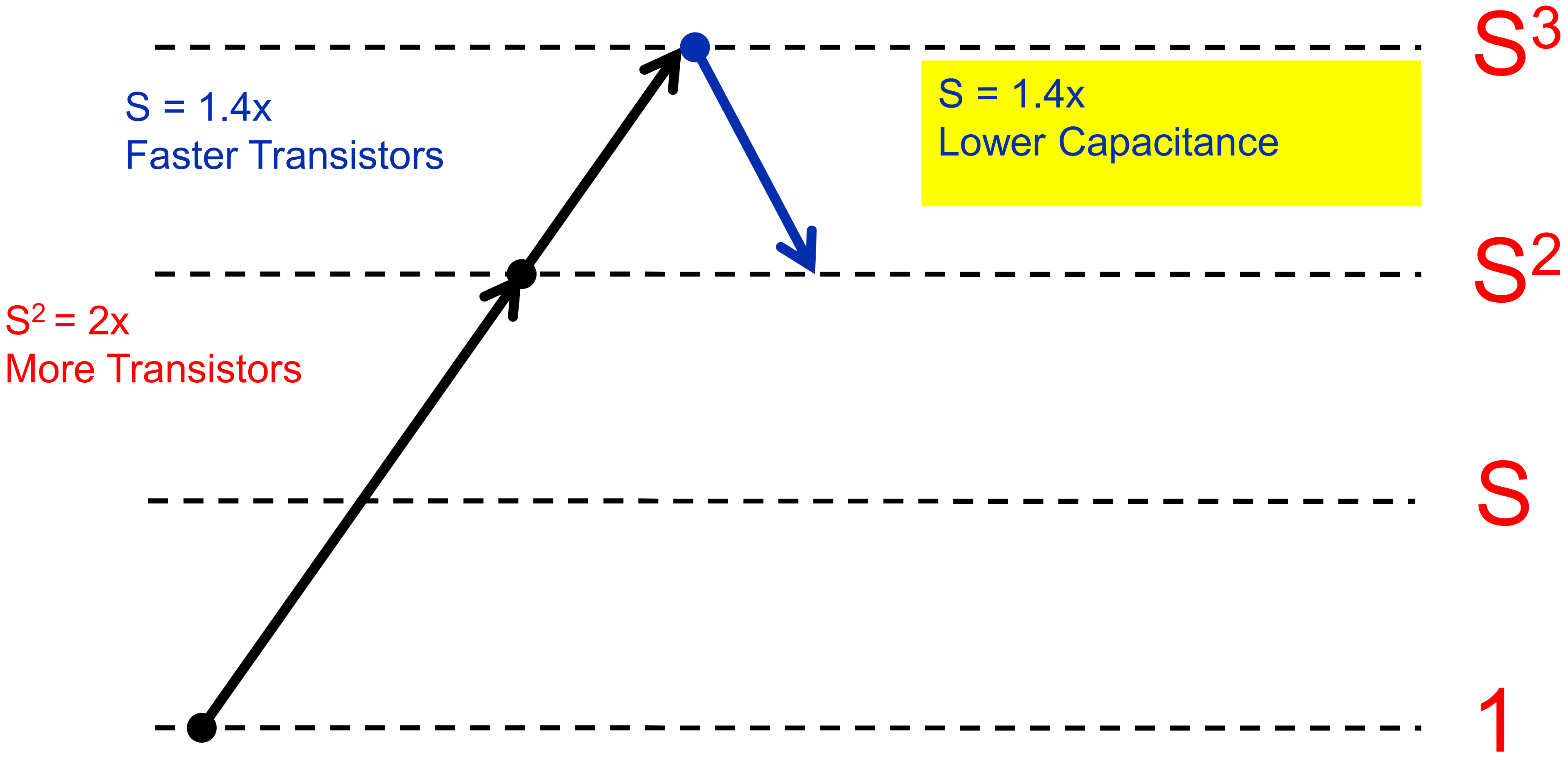
If S=1.4x …



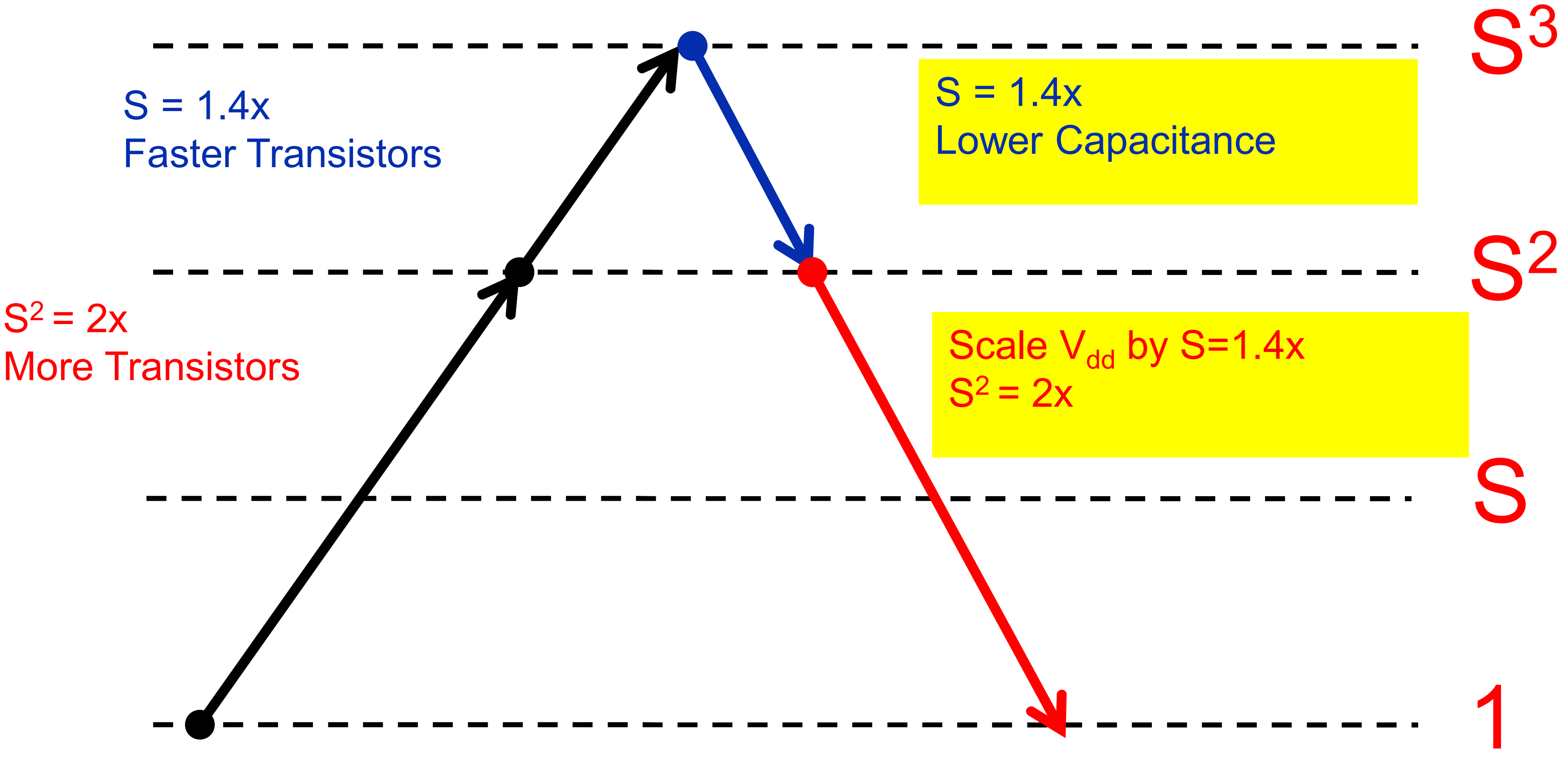
But wait: *switching 2.8x times as many transistors per unit time –*

what about power??

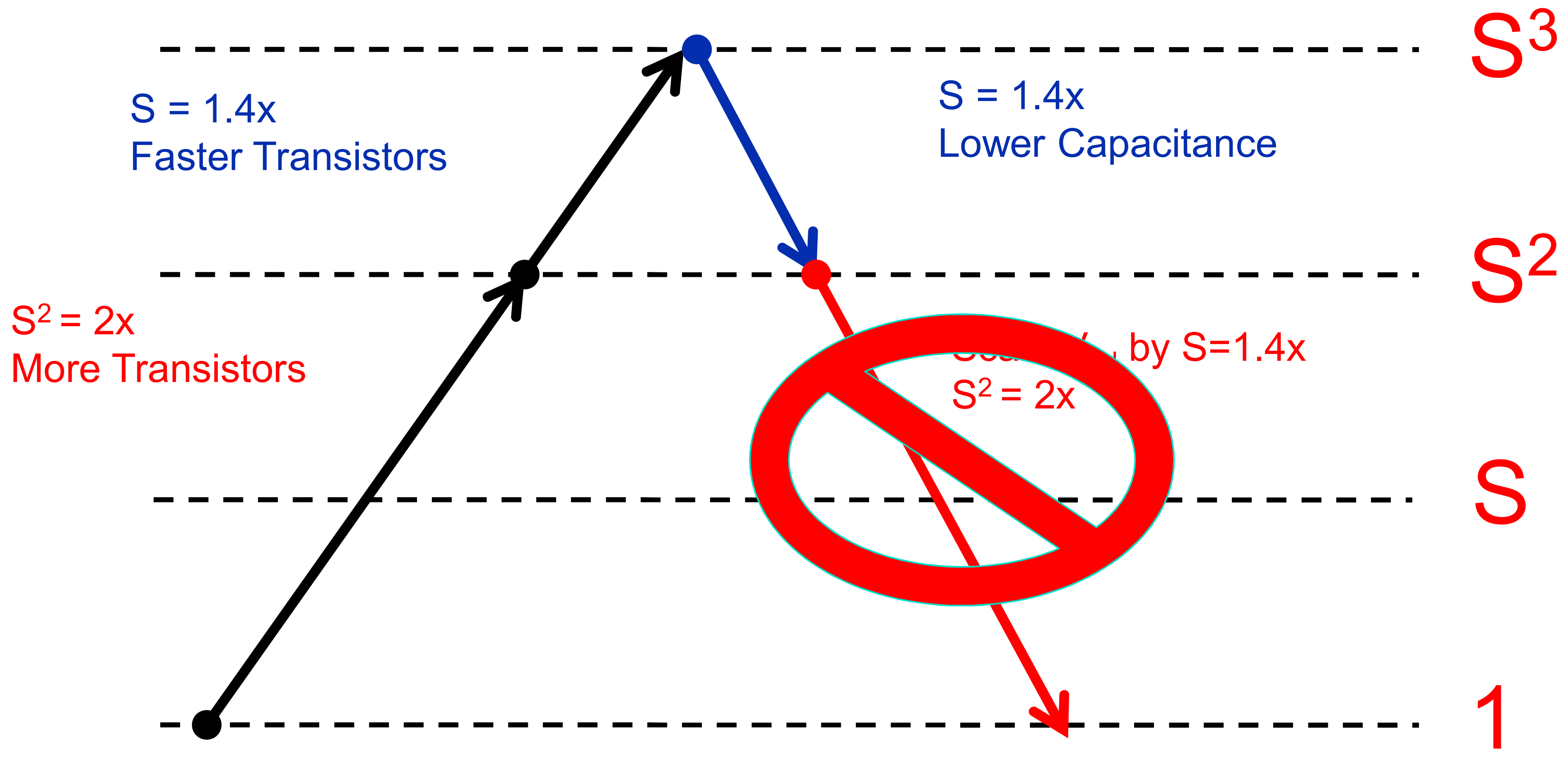
Advanced Scaling: *Dennard* – *"We can keep power consumption constant"*



Advanced Scaling: ***Dennard*** – ***"We can keep power consumption constant"***



Threshold scaling problems due to leakage



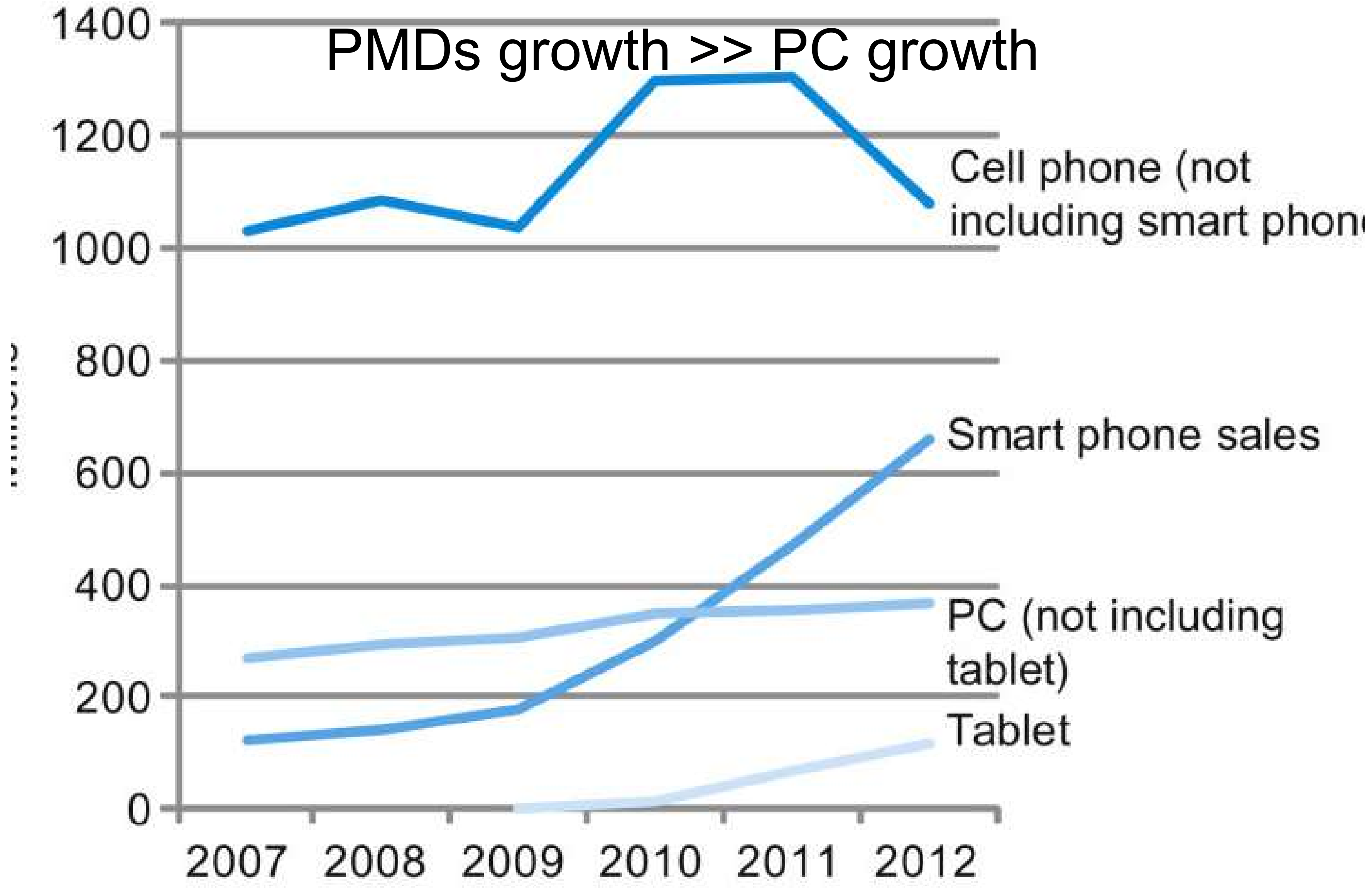
Classes of Computers

- 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
- Desktop / laptop / tablet computers
  - Single user
  - General purpose, variety of software/applications
  - Subject to cost/performance/power/reliability tradeoff
- Embedded computers (processors)
  - Hidden as components of systems, used for one predetermined application (or small set of applications)
  - Stringent power/performance/cost constraints

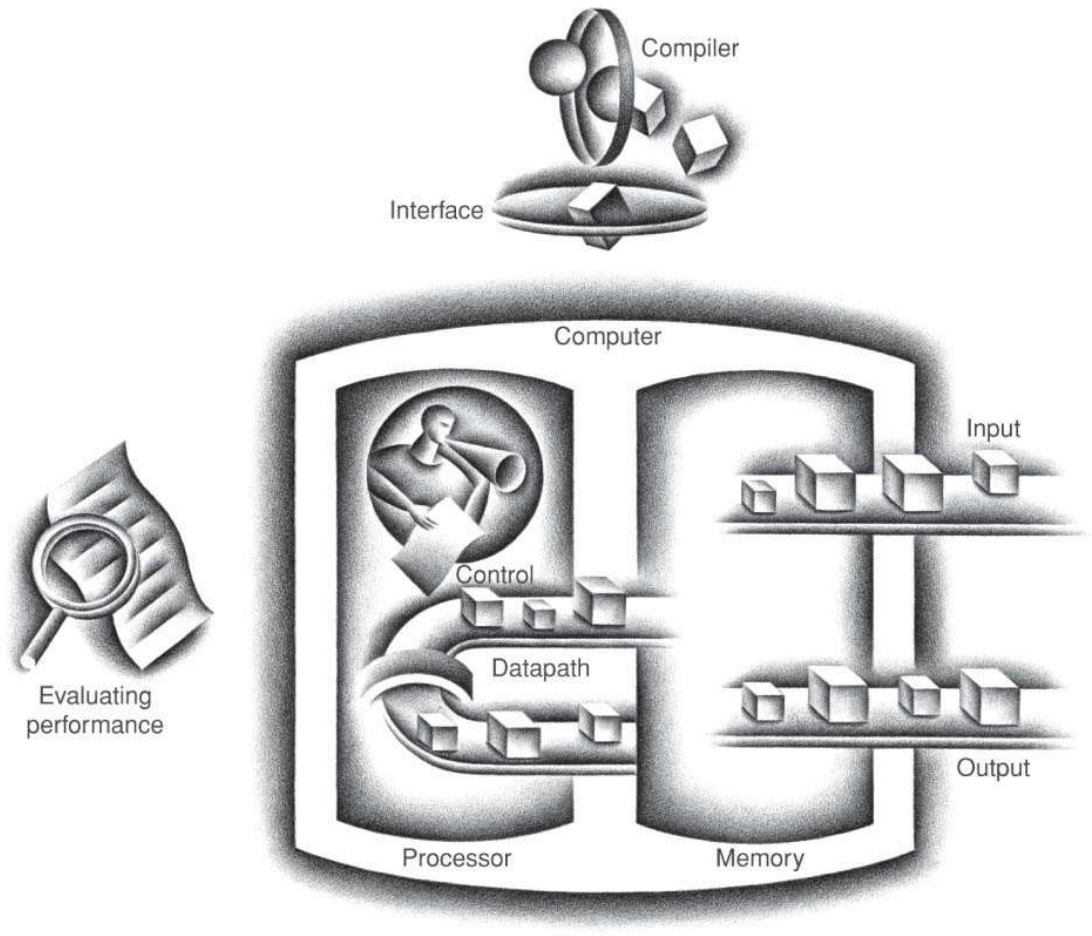
The Post-PC Era

- Personal mobile devices (PMDs)
  - 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)



The Five Classic Components of a Computer



input/output includes

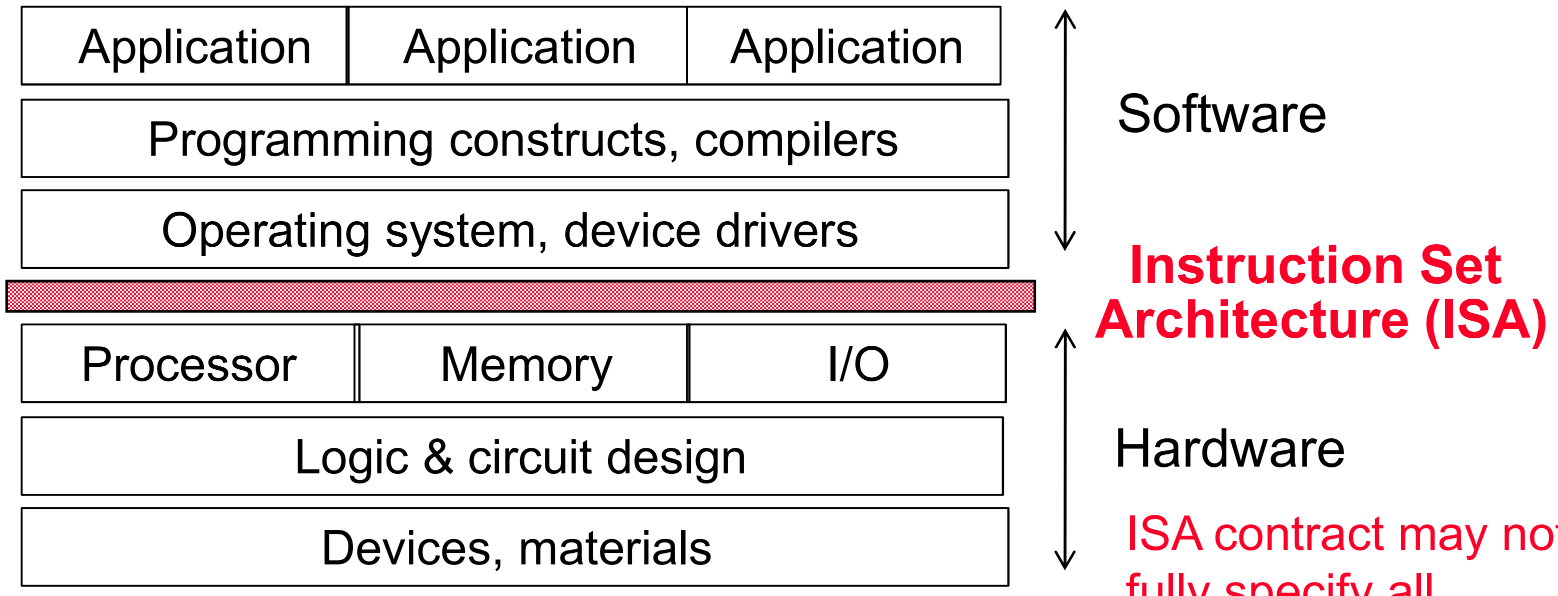
- User-interface devices (Display, keyboard, mouse)
- Storage devices (Hard disk, CD/DVD, flash)
- Network adapters (For communicating with other computers)

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 (e.g., Analog vs. Digital TV)
  - Opaque: can be hard to reason about performance

Abstraction and layering in architecture



ISA contract may not fully specify all desirable properties! (See: **Meltdown** & **Spectre**)

- 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

What does success look like (for an architect)?

“For the P6, success criteria included performance above a certain level and failure criteria included power dissipation above some threshold.”

Bob Colwell, Pentium Chronicles

Design Goals

- Function
  - Needs to be correct (witness the Pentium FDIV divider bug) http://en.wikipedia.org/wiki/Pentium_FDIV_bug
    - Unlike software, its difficult to update once deployed
  - Varies as to what functions should be supported as “base” hardware primitives
- High performance
  - Can’t measure performance by just looking at the clock rate
  - What matters is the performance for the intended applications (real-time, server, etc.)
    - An impossible goal is to have the fastest possible design for all applications
- Power (energy) consumption
  - Energy in – battery life, cost of electricity
  - Energy out – cooling costs, machine room costs

Design Goals, Continued

- Low cost
  - Per unit manufacturing costs (wafer cost)
  - Mask costs and design costs
- Reliability
  - Once verified as functioning correctly, does it continue to? For how long between failures?
  - Hard (permanent) faults versus transient faults
  - Reliability demands may be application specific
- Form factor
  - Embedded systems – e.g., RF ID tags

Challenge is to balance the relative importance of the design goals

The Move to Multicore Processors

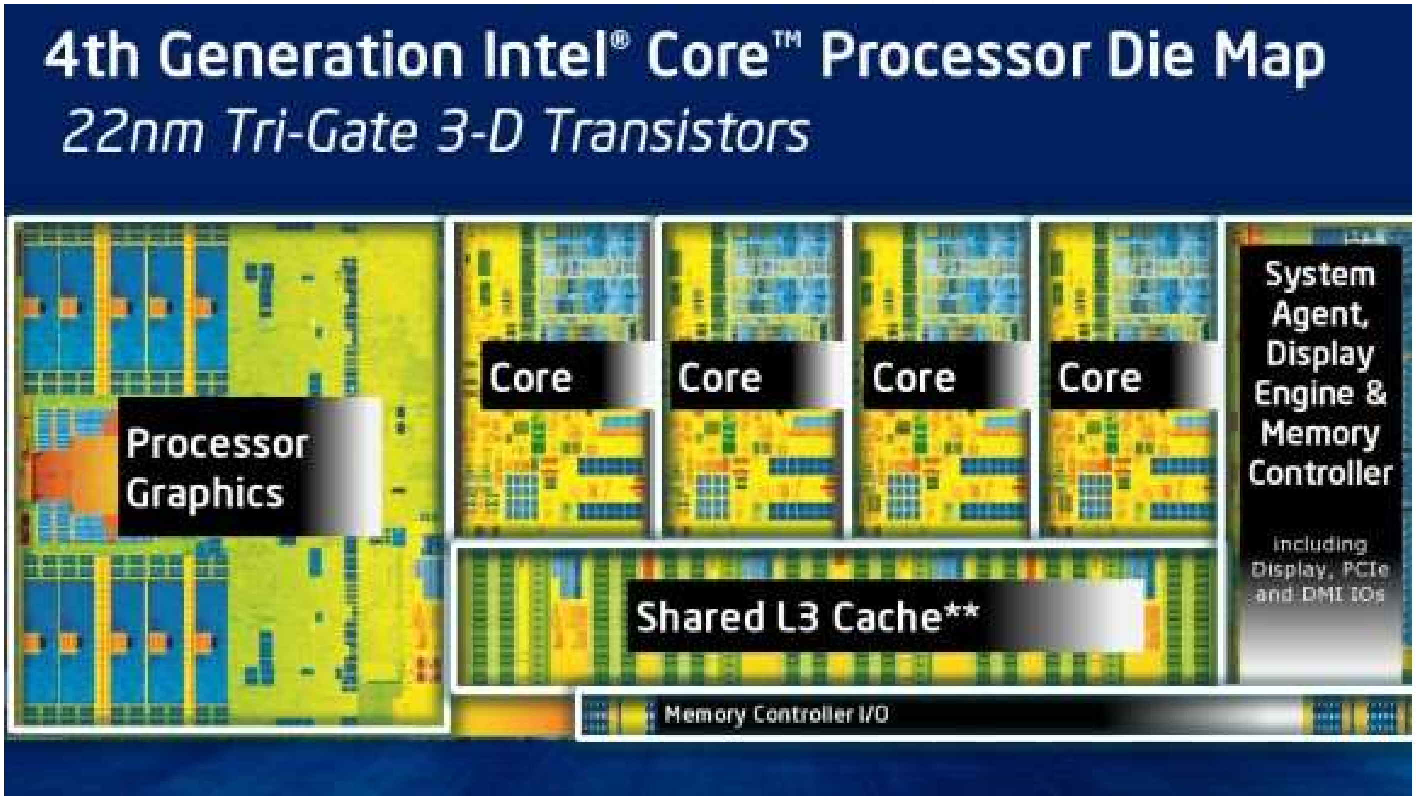
- The power challenge has forced a change in the design of microprocessors
  - Since 2002 the rate of improvement in the response time of programs on desktop computers slowed from a factor of 1.5 per year to less than a factor of 1.2 per year
- As of 2006 all server companies were shipping microprocessors 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 FinFET	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)

E.g., Intel's Core i7 (Haswell)

- 22nm/FinFET technology, ~1.4 billion transistors, 4 cores/8 threads, 8MBs of shared L3 cache


4th Generation Intel® Core™ Processor Die Map
22nm 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

http://en.wikipedia.org/wiki/Haswell_%28microarchitecture%29

Courtesy, Intel ®

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)
  - 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

Performance Metrics

- Purchasing perspective
  - given a collection of machines, which has the
    - best performance (speed and/or power consumption)?
    - least cost?
    - best cost/performance?

- Design perspective
  - faced with design options, which has the
    - best performance improvement (speed and/or power consumption)?
    - least cost?
    - best cost/performance?

- Both require
  - basis for comparison
  - metric for evaluation

- Our goal is to understand what factors in the architecture contribute to overall system performance and the relative importance (and cost) of these factors

Defining Performance

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

“Never let an engineer get away with simply presenting the data. Always insist that he or she lead off with the conclusions to which the data led.”

Bob Colwell, Pentium Chronicles

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

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

- 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!)

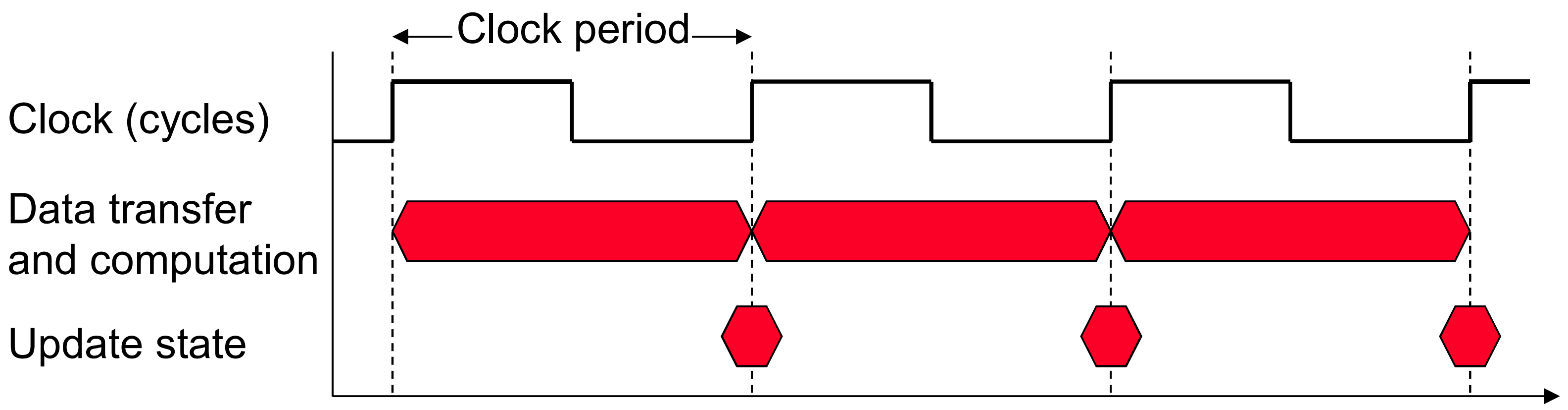
Measuring Execution Time

- Elapsed 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

- 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

Execution Time 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)

$$\text{CPU execution time for a program} = \text{\# CPU clock cycles for a program} \times \text{clock cycle time}$$

or

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

- 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

Improving Performance 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.

$$\text{CPU time}_A = \frac{\text{CPU clock cycles}_A}{\text{clock rate}_A}$$

$$\text{CPU clock cycles}_A = 10 \text{ sec} \times 2\text{x}10^9 \text{ cycles/sec}$$
$$= 20 \times 10^9 \text{ clock cycles}$$

$$\text{CPU time}_B = \frac{1.2 \times 20 \times 10^9 \text{ clock cycles}}{\text{clock rate}_B}$$

$$\text{clock rate}_B = \frac{1.2 \times 20 \times 10^9 \text{ cycles}}{6 \text{ seconds}} = 4 \text{ GHz}$$

Clock Cycles per Instruction

- 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

$$\begin{matrix}\text{\# CPU clock cycles} \\ \text{for a program}\end{matrix} = \begin{matrix}\text{\# Instructions} \\ \text{for a program}\end{matrix} \times \begin{matrix}\text{Average clock cycles} \\ \text{per instruction}\end{matrix}$$

- 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 Execution Time 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

Using the Execution Time Equation

- Computers A and B implement the same ISA. Computer A has a clock cycle time of 250 ps and an average CPI of 2.0 for some program and computer B has a clock cycle time of 500 ps and an average CPI of 1.2 for the same program. Which computer is faster and by how much?

Each computer executes the same number of instructions, *I*, so

$$\text{CPU time}_A = I \times 2.0 \times 250 \text{ ps} = 500 \times I \text{ ps}$$

$$\text{CPU time}_B = I \times 1.2 \times 500 \text{ ps} = 600 \times I \text{ ps}$$

Clearly, A is faster … by the ratio of execution times

$$\frac{\text{performance}_A}{\text{performance}_B} = \frac{\text{execution\_time}_B}{\text{execution\_time}_A} = \frac{600 \times I \text{ ps}}{500 \times I \text{ ps}} = 1.2$$

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 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 Execution Time 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

Determinates of CPU Execution Time

CPU time = Instruction_count x CPI x 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

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 Barcelona (CR = 2.5 GHz)

Name	ICx10^9	CPI	ExTime (sec)	RefTime (sec)	SPEC ratio
perl	2,1118	0.75	637	9,770	15.3
bzip2	2,389	0.85	817	9,650	11.8
gcc	1,050	1.72	724	8,050	11.1
mcf	336	10.00	1,345	9,120	6.8
go	1,658	1.09	721	10,490	14.6
hmmer	2,783	0.80	890	9,330	10.5
sjeng	2,176	0.96	837	12,100	14.5
libquantum	1,623	1.61	1,047	20,720	19.8
h264avc	3,102	0.80	993	22,130	22.3
omnetpp	587	2.94	690	6,250	9.1
astar	1,082	1.79	773	7,020	9.1
xalancbmk	1,058	2.70	1,143	6,900	6.0
Geometric Mean					11.7

SPEC CINT2006 on Intel i7 (CR = 2.66GHz)

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, input set used, specific computer configuration (clock rate, cache sizes and speed, memory size and speed, etc.))

SPEC Power Benchmarks

- Power consumption of a server under different workload levels (divided into 10% increments)
  - Performance: ssj_ops/sec
  - Energy: joules
  - Power: Watts (joules/sec)

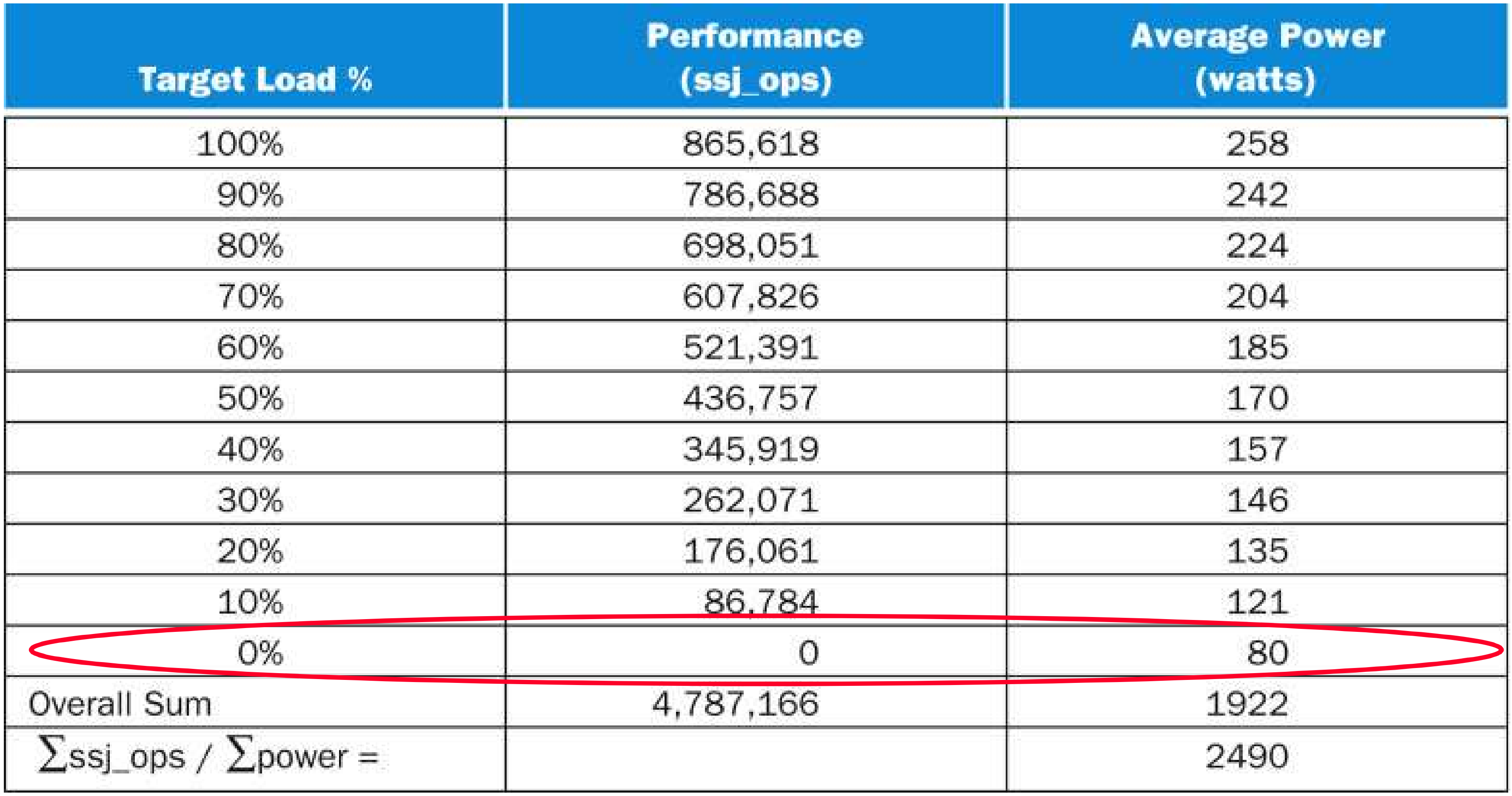
$$\text{Overall ssj\_ops per Watt} = \left(\sum_{i=0}^{10} \text{ssj\_ops}_i\right) \Big/ \left(\sum_{i=0}^{10} \text{power}_i\right)$$

http://www.zdnet.com/blog/ou/spec-launches-standardized-energy-efficiency-benchmark/927

SPECpower_ssj2008 on the Barcelona

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 on 2.66GHz Intel Xeon



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	1922
$\sum$ssj_ops / $\sum$power =		2490

Fallacy: Lowest Power & Energy at (Near) Idle

- Look back at the Intel Xeon benchmark
  - At 100% load: 258W
  - At 50% load: 170W (66%)
  - At 10% load: 121W (47%)
- A Google data center
  - Mostly operates at a 10% to 50% load
  - At 100% load less than 1% of the time
- Data centers should be designed to make the power proportional to the load
  - "Energy Proportional Computing," Barroso and Hölzle, IEEE Computer Magazine, Dec 2007
  - Try to design servers so that they consume 10% of peak power when running at 10% workload levels, etc.

Pitfall: 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 multiplies account for 20 seconds of the 100 second run time?

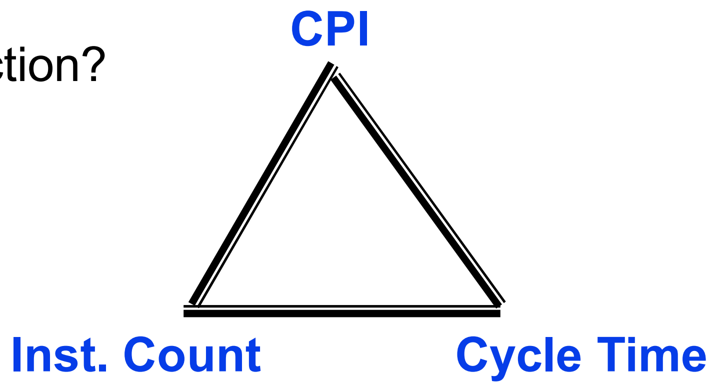
- Corollary: Make the common case fast

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 for performance:
Time to execute the program!

Best Metric for power? Cost? Security?



Tradeoffs depend on the instruction set, the processor organization, and compilation techniques.

Next Week's Material and Reminders

- Next week
  - Topics: Review!
  - A quiz on reviewed materials
  - A set of practice exercises assigned