CSE 530
Fundamentals of Computer Architecture

Spring 2021

Introduction

John (Jack) Sampson

Course material on CANVAS: canvas.psu.edu

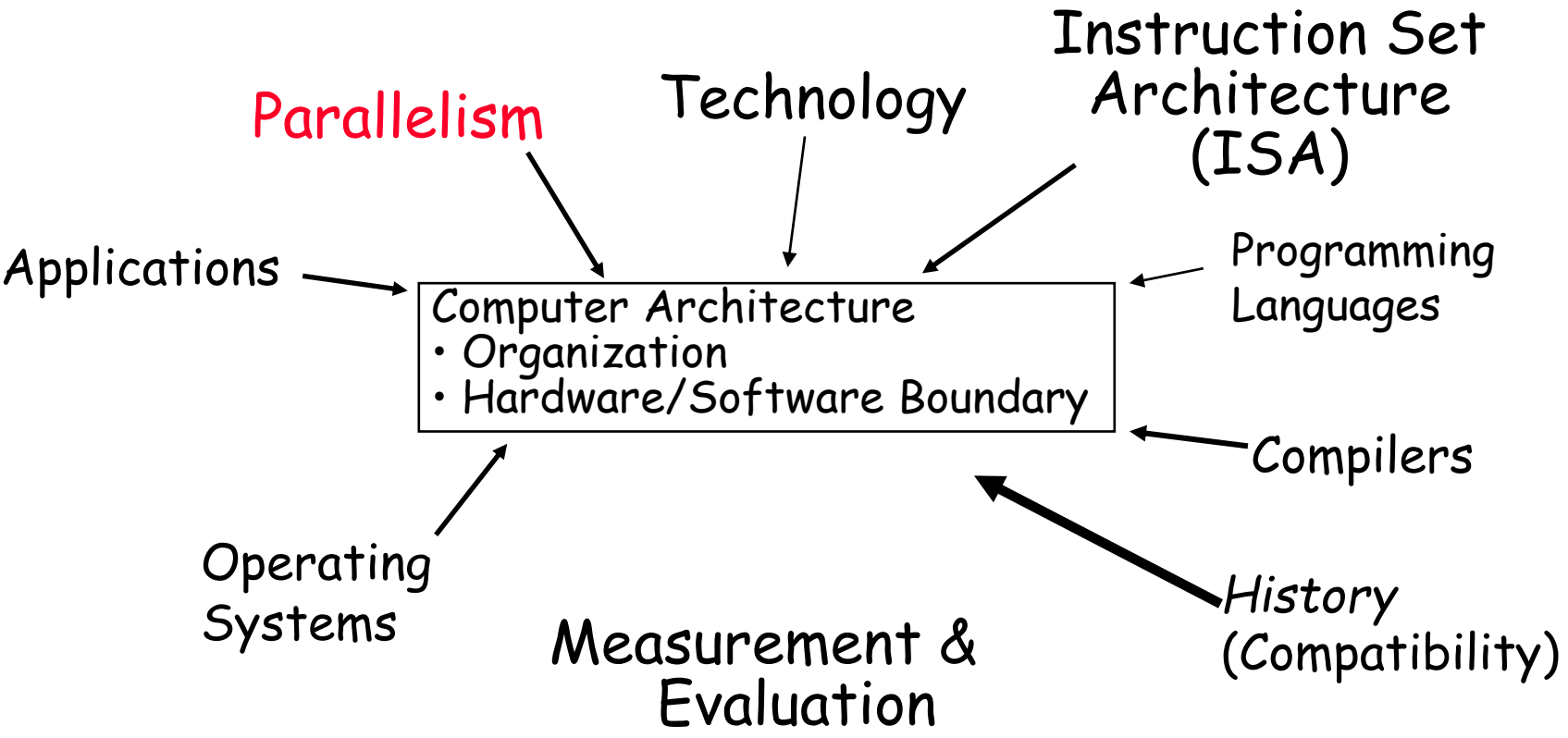
[Adapted in part from Mary Jane Irwin, V. Narayanan, A. Kolli @PSU, and includes materials originally developed by Profs. Austin, Brehob, Falsafi, Hill, Hoe, Lipasti, Martin, Roth, Shen, Smith, Sohi, Tyson, Vijaykumar, and Wenisch of Carnegie Mellon University, Purdue University, University of Michigan, University of Pennsylvania, and University of Wisconsin. Material flow organized in part around *Computer Architecture: A Quantitative Approach* by Hennessey and Patterson]

Course administrivia

- Class Meetings: TuTh, **10:35-11:50**am on ZOOM
  - There will be in-Zoom participation tracking implemented later
- Instructor: John (Jack) Sampson (sampson@cse.psu.edu)
  - Office & office hours: See Zoom link in Syllabus
- TAs: R. Pasculano
- Web: CANVAS: canvas.psu.edu
  All materials and communications via Canvas
- Lab: Accounts on CSE machines in W204 via VPN
- Texts: *Computer Architecture: A Quantitative Approach, 6th Edition,* Hennessy and Patterson

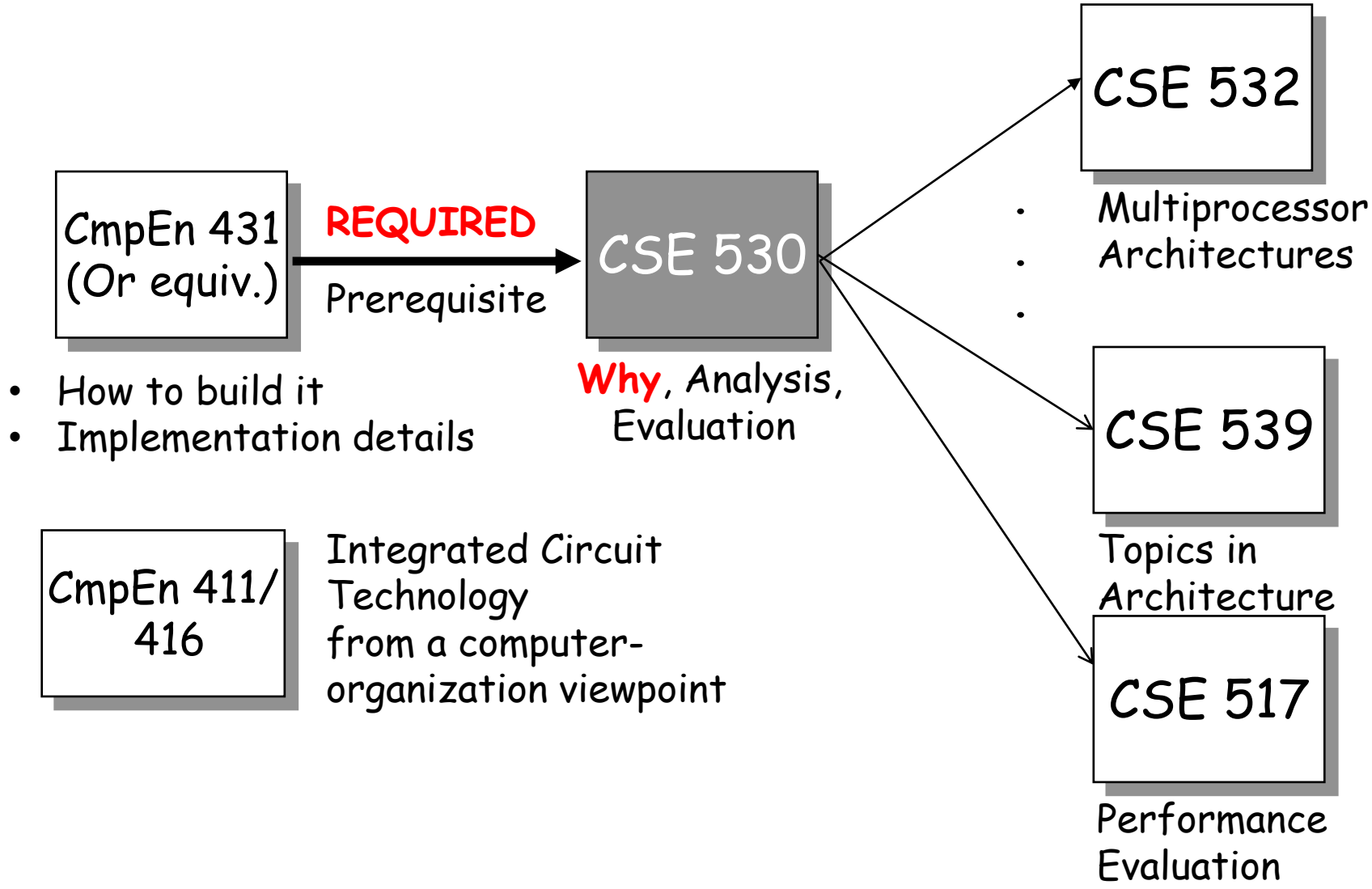
CSE 530 course focus

- Understanding the technology factors, design techniques, architectural innovations, and evaluation methods that will determine the form of computers in the 21st century



Background requirement & related courses

- Basic knowledge of computer organization is assumed



Beyond lectures: Reading technical papers

- As graduate students, you are now researchers. Most of the information you need will be in technical papers
  - The ability to rapidly scan and understand technical papers is key
  - You will learn a lot about doing good systems research by reading others' papers
- This class will feature two sets of paper-reading assignments. Papers for both will be provided on CANVAS
  - Type 1: "Foundational" papers
    - Important supplement to book
    - Practice delivering "elevator" summaries of research and interactive responses to pre-reading assigned questions in class
  - Type 2: "Modern(ish)" readings
    - These papers provide examples of where research has continued since the foundational works introduced the concepts
  - Be prepared to discuss all assigned readings in class. You will prepare brief written responses for some of these papers

What is computer architecture?

- “Computer architecture is the science and art of selecting and interconnecting hardware components to create computers that meet functional, performance and cost goals.” – Wisconsin Computer Architecture Group

- “Computer architecture is the conceptual design and fundamental operational structure of a computer system. It is a blueprint and functional description of requirements and design implementations for the various parts of a computer, focusing largely on the way by which the central processing unit (CPU) performs internally and accesses addresses in memory” – Wikipedia

Architecture as abstraction

We eventually want to understand and identify all screnarios where we may have to rethink sw/hw interaction in its entirety

- A computer architecture provides a stable (but not necessarily stationary) target for software

CA provides target for sw

  - E.G. x86-64 continues to grow, but remains backwards compatible
  - Allows SW to run on new HW / compatible family of HW

ISA=contract of hw+sw

- Separates concerns about WHAT happens/can happen from HOW it happens
  - Architecture vs. Microarchitecture
  - Contracts tend to be about functionality rather than implementation
    - Can lead to compliant-but-underspecified behaviors (e.g. Spectre/Meltdown)

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 it sometimes helps quite a bit!)
- Two downsides to layering

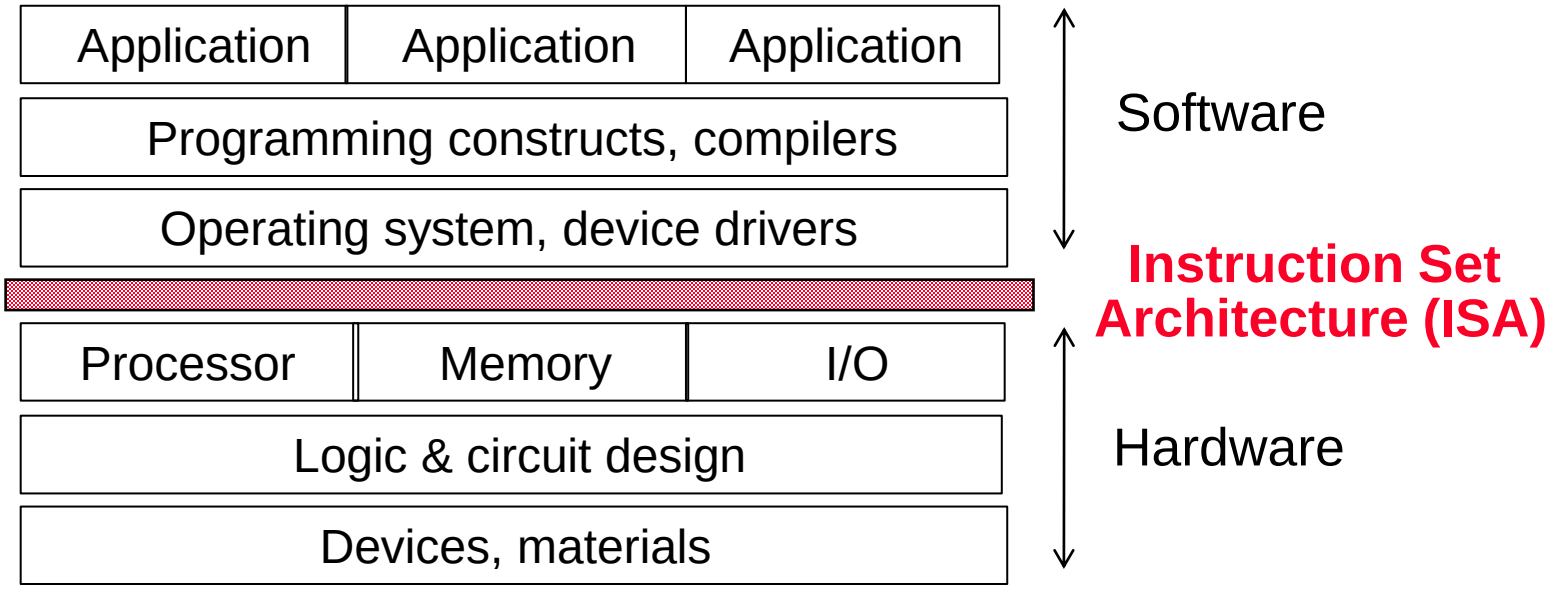
BCD support in h/w

  - 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

When we are not aware of the contract completely, things can go wrong

Abstraction and layering in architecture

sometimes applications cut all the way through to h/w
e.g. accelarator operations, user space drivers



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

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

sth in TLB was botched

(if # of applications is small, may be doable)

if plugged in, think about power.
if unplugged, think about energy rqd to do some operations

Design Goals, Continued

- Low cost

Not the first unit cost
First unit cost always high because u design the entire manufacturing process
Cost per item in long term

  - 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

Deeper dive: Performance

Performance

- Two definitions
  - **Latency (execution time)**: time to finish a fixed task
  - **Throughput (bandwidth)**: number of tasks in fixed time
  - Very different: throughput can exploit parallelism, latency can't
    - Baking bread analogy
  - Often contradictory
  - Choose definition to matches measurement goals

- Example: move people from A to B, 10 miles
  - Car: capacity = 5, speed = 60 miles/hour
  - Bus: capacity = 60, speed = 20 miles/hour
  - Latency: **car = 10 min**, bus = 30 min
  - Throughput: car = 15 PPH (count return trip), **bus = 60 PPH**

Five key principles to performance

Parallelism

cannot help latency

- Go faster by doing many things at once.

Speculation

there are patterns in programs that are executed, the sw that is written, so we can make guesses and pre-do stuff

- Guess -- if you can be right most of the time.

Locality

pattern also comes in here

- Related data are “near” one another in time and space.

Memoization

again, patterns

- Programs do the same thing over and over. Remember it.

Amdahl's Law

- Make the common case fast…
  … but speedup ultimately limited by the uncommon case

technology gives us resource, CA uses those resource and prepares target for sw

Parallelism: Work and Critical Path

not important slide

Parallelism - the amount of independent sub-tasks available

Work (T_1) - time to complete a computation on a sequential system

Critical Path (T_∞) - time to complete the same computation on an infinitely-parallel system

Average Parallelism

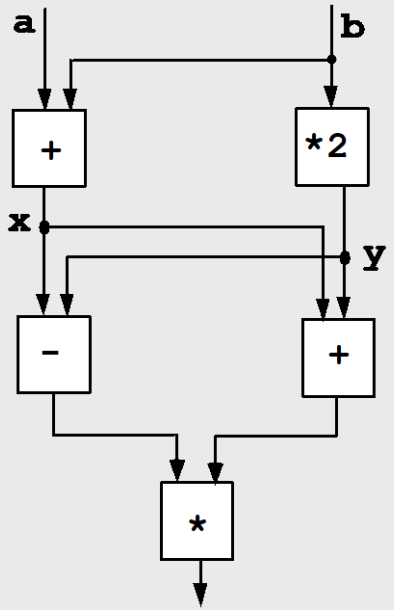
$P_{avg} = T_1 / T_\infty$

For a p wide system

$T_p \geq \max\{ T_1/p, T_\infty \}$

$P_{avg} >> p \Rightarrow T_p \approx T_1/p$

```
x = a + b;
y = b * 2
z =(x-y) * (x+y)
```



Amortization & Speculation

overhead cost : one-time cost to set something up

per-unit cost : cost for per unit of operation

total cost = overhead + per-unit cost x N

high overhead OK if distributed over many units

$\Rightarrow$ lower the *average cost*

average cost = total cost / N

= (overhead / N) + per-unit cost

Speculation – make educated guesses to avoid expensive operations if you can be right most of the time

*Make the **common case** fast*

Make the uncommon case correct

Locality Principle

One's recent past is a good indication of near future.

- Temporal Locality: If you looked something up, it is very likely that you will look it up again soon
- Spatial Locality: If you looked something up, it is very likely you will look up something nearby next

Locality == Patterns == Predictability

Converse:

Anti-locality : If you haven't done something for a very long time, it is very likely you won't do it in the near future either

Memoization

Dual of temporal locality but for computation

If something is expensive to compute, you might want to remember the answer for a while, just in case you will need the same answer again

Why does memoization work??

… and when does it lead us astray?

Examples

- Branch predictor
- Trace cache

Amdahl's Law

usually there are overheads when S comes in
S alwayys > 1

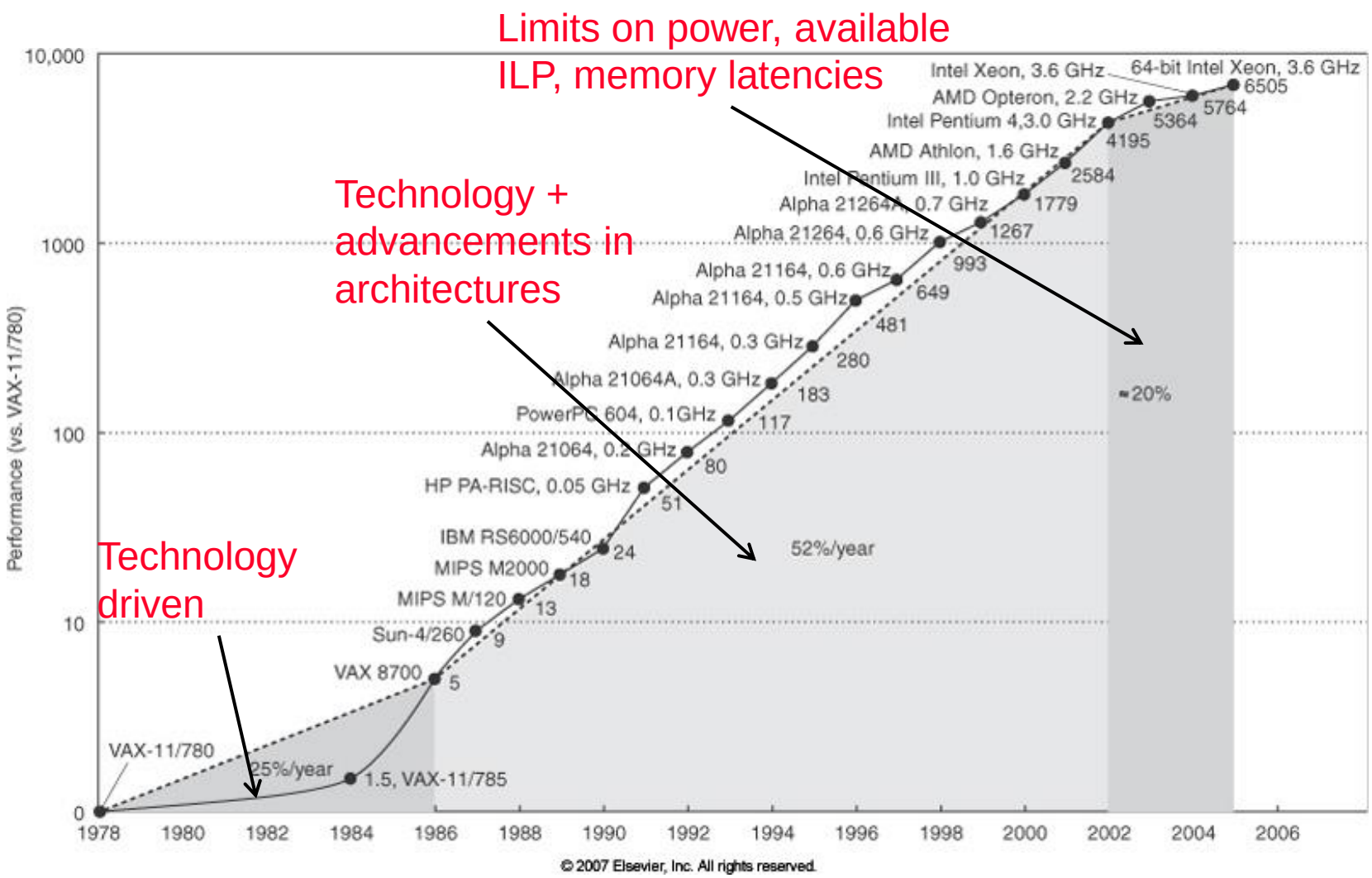
Speedup= $time_{without\ enhancement}$ / $time_{with\ enhancement}$

Suppose an enhancement speeds up a fraction f of a task by a factor of S

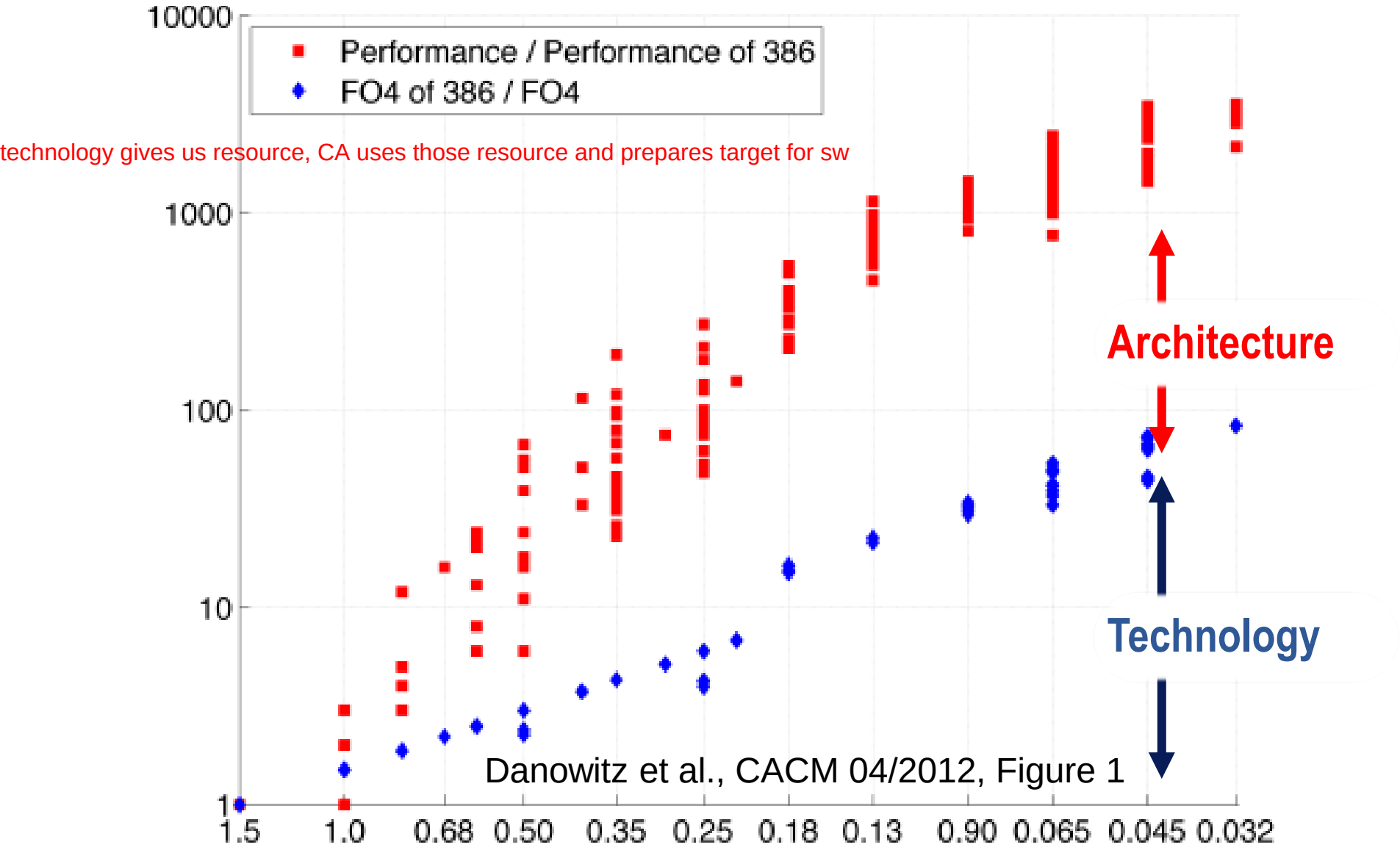
$$time_{new} = time_{orig} \cdot ((1-f) + f/S)$$

$$S_{overall} = 1 / ((1-f) + f/S)$$

Growth in processor performance (SPECint)



Enablers: Technology + Architecture



technology gives us resource, CA uses those resource and prepares target for sw

Measuring performance

Performance Improvement

Recall the example from Slide 13: move people from A to B, 10 miles

- Car: capacity = 5, speed = 60 miles/hour
- Bus: capacity = 60, speed = 20 miles/hour
- Latency: **car = 10 min**, bus = 30 min
- Throughput: car = 15 PPH (count return trip), **bus = 60 PPH**

- Car/bus example
  - Latency? Car is 3 times (+200%=*300%) faster than bus
  - Throughput? Bus is 4 times (+300%=*400%) faster than car

Averaging Performance Numbers I

*** IMPORTANT!

You can add latencies, but not throughput

- Latency(P1+P2, A) = Latency(P1,A) + Latency(P2,A)
- Throughput(P1+P2,A) != Throughput(P1,A) + Throughput(P2,A)

E.g.,

- 1 mile @ 30 miles/hour + 1 mile @ 90 miles/hour
  - Average is **not** 60 miles/hour
- 0.033 hours at 30 miles/hour + 0.01 hours at 90 miles/hour
  - Average is only 47 miles/hour! (2 miles / (0.033 + 0.01 hours))

Averaging Performance Numbers II

- Latency(P1+P2, A) = Latency(P1,A) + Latency(P2,A)
- Throughput(P1+P2,A) =

  1 / [(1/ Throughput(P1,A)) + (1/ Throughput(P2,A))]

- Three averaging techniques:
  - **Arithmetic** : $(1/N) * \sum_{P=1..N}$ Latency(P)
    - **For times**: units proportional to time (e.g., latency)
  - **Harmonic** : $N / \sum_{P=1..N}$ 1/Throughput(P)
    - **For rates**: units inversely proportional to time (e.g., throughput)
  - **Geometric** : $\sqrt[N]{\prod_{P=1..N}}$ Speedup(P)

For ratios,geomean is used

    - **For ratios**: unitless quantities (e.g., speedups)

The "Iron Law" of Processor Performance

$$\text{Processor Performance} = \frac{\text{Time}}{\text{Program}}$$

$$= \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Cycles}}{\text{Instruction}} \times \frac{\text{Time}}{\text{Cycle}}$$

(code size) (CPI) (cycle time)

Architecture --> Implementation --> Realization

Compiler Designer Processor Designer Chip Designer

Performance – Key Points

Amdahl's law

$$S_{overall} = 1 / ((1-f) + f/S)$$

Iron law

$$\frac{Time}{Program} = \frac{Instructions}{Program} \times \frac{Cycles}{Instruction} \times \frac{Time}{Cycle}$$

Averaging Techniques

Arithmetic Time

$$\frac{1}{n}\sum_{i=1}^{n} Time_i$$

Harmonic Rates

$$\frac{n}{\sum_{i=1}^{n} \frac{1}{Rate_i}}$$

Geometric Ratios

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

Trends in technology

- Integrated circuit (IC) technology – transistor density quadruples every ~4 years; die size increases 10% to 20% per year → chip transistor count doubled every ~2 years
  - More transistors *can* be used to increase the performance
  - **READ for Thursday: F1 – Moore's Law**

the budget is increasing with these trends

with more budget, we things differently

- Processor raw speeds (clock rates) also increased at an exponential rate (doubled every ~2 years) until around 2005 (the "why" is to come soon)

more buget, and better things

- DRAM capacity is doubling (roughly) every 2 years

- Prior to 1990 and since 2004, disk densities are doubling every 3 yrs (disks are 50-100 times cheaper/ bit than DRAM)

Trends in performance, speed, clock rates

- While transistor count was improving quadratically every 4 years, transistor speed was only improving linearly
  - And since the signal delay down a wire increases in proportion to the product of its resistance (R) and capacitance (C), as feature size shrinks, wires get shorter, but R and C (per unit length) get worse → wire speed doesn't scale (wires are getting slower compared to transistors)

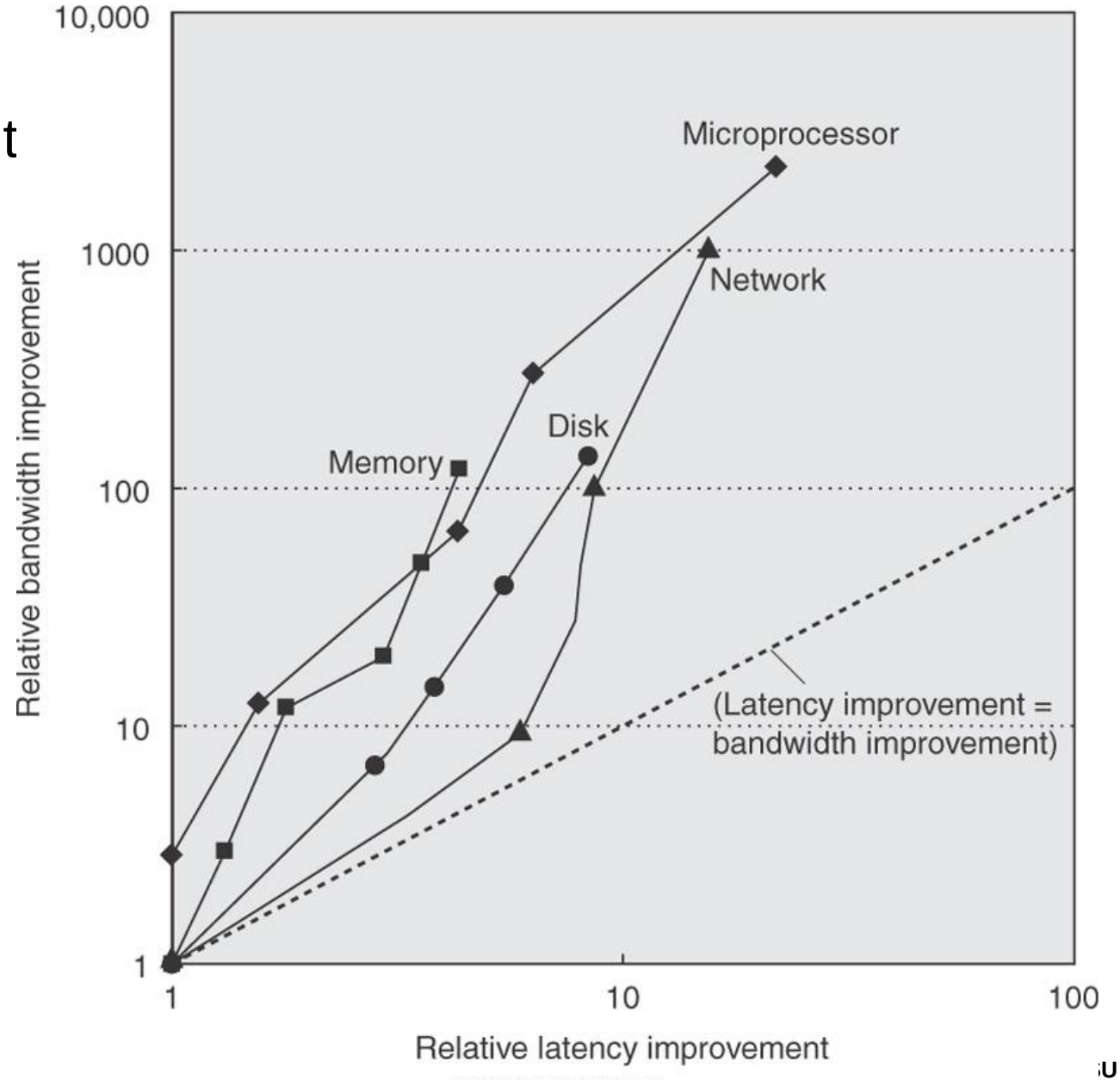
shrink device, resistance of wires will increase
so, speed does not improve as much
so, only linear improvement

since closer to each other, improves somewhat in terms of interconnect

- Also it depends on what is being measured
  - Latency (response time) – the time between the start and the completion of an event (e.g., milliseconds for a disk access)
  - Bandwidth (throughput) – the total amount of work done in a given time (e.g., MBs per second for a disk transfer)

Latency lags bandwidth

- Latency improved about 10X while bandwidth improved about 100X to 1000X
- Implications?
- Read this paper (**F2**) for in-class discussion on Thursday



1st reading assignment – Foundational Paper #1

- Read “Cramming More Components onto Integrated Circuits” by Gordon Moore – for in class discussion on Thursday
- Be prepared to discuss the paper **in class**. It may be useful to discuss the paper with classmates prior to class.
  - I will call on random people for responses, so be prepared
- Elevator summary of the paper in 30 seconds (or less) – three (20 words or less each) sentences
  1. What problem the paper is trying to solve, or insight it is trying to provide
  2. Why the problem or insight is important
  3. Why the proposed solution/insight is novel and/or significant
- Favorite one liner quote

1st reading assignment – Foundational Paper #1

less components -> more yield cost, more components -> lot of packaging cost

- Be prepared to answer <u>at least</u> the following questions:
  - Q1: The figure on page 2 graphs relative manufacturing cost per component against the number of components per IC. Why do chips become less cost effective per component for both very large and very small numbers of components per chip?
  - Q2: One of the potential problems which Moore raises is heat. Do you agree with his conclusions? Either justify or refute Moore's conclusions. heat density increases very much with shrinked area
  - Q3: A popular misconception of Moore's law is that it states that the speed of computers increases exponentially. Explain what Moore's law actually says based on this paper. better resources, low cost
  - Q4: What does Moore mean by "linear" circuits (in contrast to digital circuits)?

Next reading assignment – Foundational #2

- Read “Latency Lags Bandwidth” by David Patterson – for in class discussion on Thursday
- A < 30 second elevator summary of the paper
- Favorite one liner quote
- And be ready to answer at least the following questions
  - Q1: In which four of the technologies (microprocessors, memory, networks, disks) is bandwidth more important to you and why? In which is latency and why? both latency+bw for uP, latency for network, b/w for DRAM and Hard
  - Q2: Give an example of when faster latency increases bandwidth and one where it does not. Give an example of when improved bandwidth hurts latency and when it does not. whenever we take away all sorts of buffers
  - Q3: What is a key downside of both replication and prediction? coherence compensation for misprediction

Trends in power and energy

- Power and energy are increasingly important
  - Battery life for mobile devices
  - Maximum temperatures for devices without active cooling
  - Electric bills for compute/data centers
    - Pay for power “twice” – once in and once out (to cool)
  - Environmental concerns

Ferguson township

	Light Bulb 100W	**BGA 25W**
Surface Area	106 cm^2	1.96 cm^2
Heat Flux	0.9 W/cm^2	12.75 W/cm^2

	90nm	**65nm**	**40nm**	**28nm**
Density	1	2	5	10
Power/gate	1	0.6	0.36	0.22

heat density off the chips was higher

The power equations of importance

- Dynamic power consumption (switching transistors) is the primary metric (measured in Watts)

xsitor switch factor lkg power per time

$$Power_{dyn} = \frac{1}{2} \times CapLoad_{dyn} \times V_{dd}^2 \times ClockFreq$$

  - Can reduce the number of transistors switching, the CapLoad, lower V_{dd} (now at < 1V) and/or can slow the clock

most xsitors don't switch too often,so the heat generation is not uniform across chip. hotspots

- For battery powered devices, energy is the important metric (measured in joules)

$$Energy_{dyn} = CapLoad \times V_{dd}^2$$

voltage reduce means biggest win, but exponentially increases latency so lkg voltage may dominate

  - For a fixed task, a slower clock reduces the power, but not the energy consumed (just takes longer to finish the task)
- Static power consumption, due to leakage current even when the transistor is turned off, is now also of concern

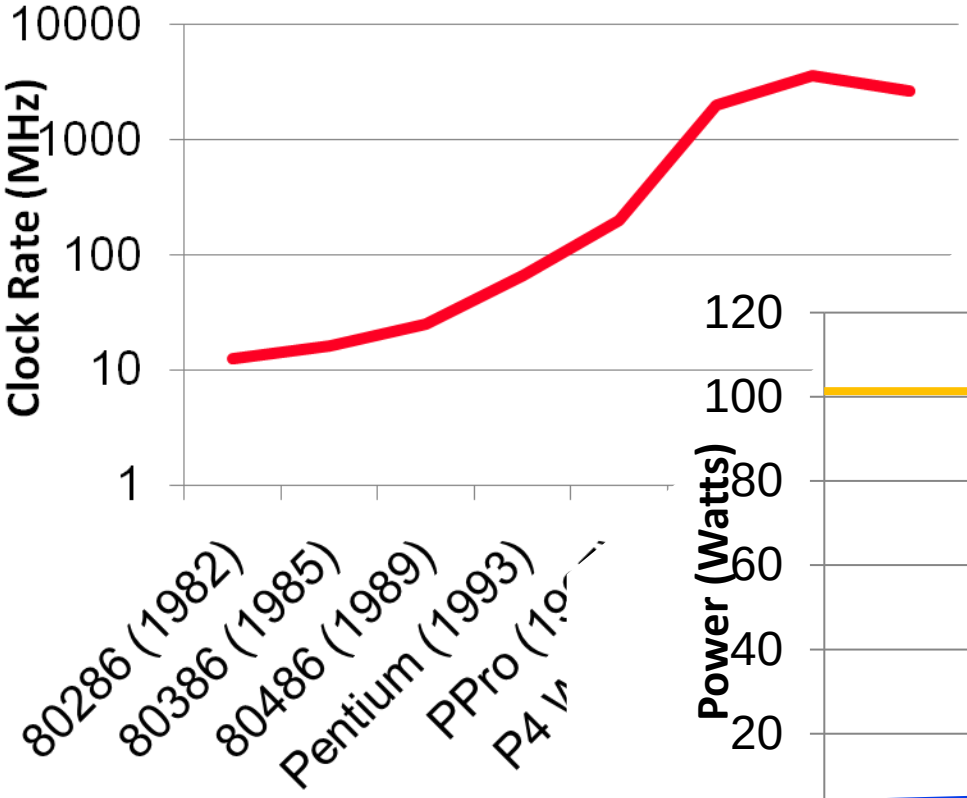
$$Power_{static} = LeakCurrent \times V_{dd}$$

  - Gate the supply voltage (V_{dd}) to inactive units
  - Static power can increase with reduced clock frequency. Why?

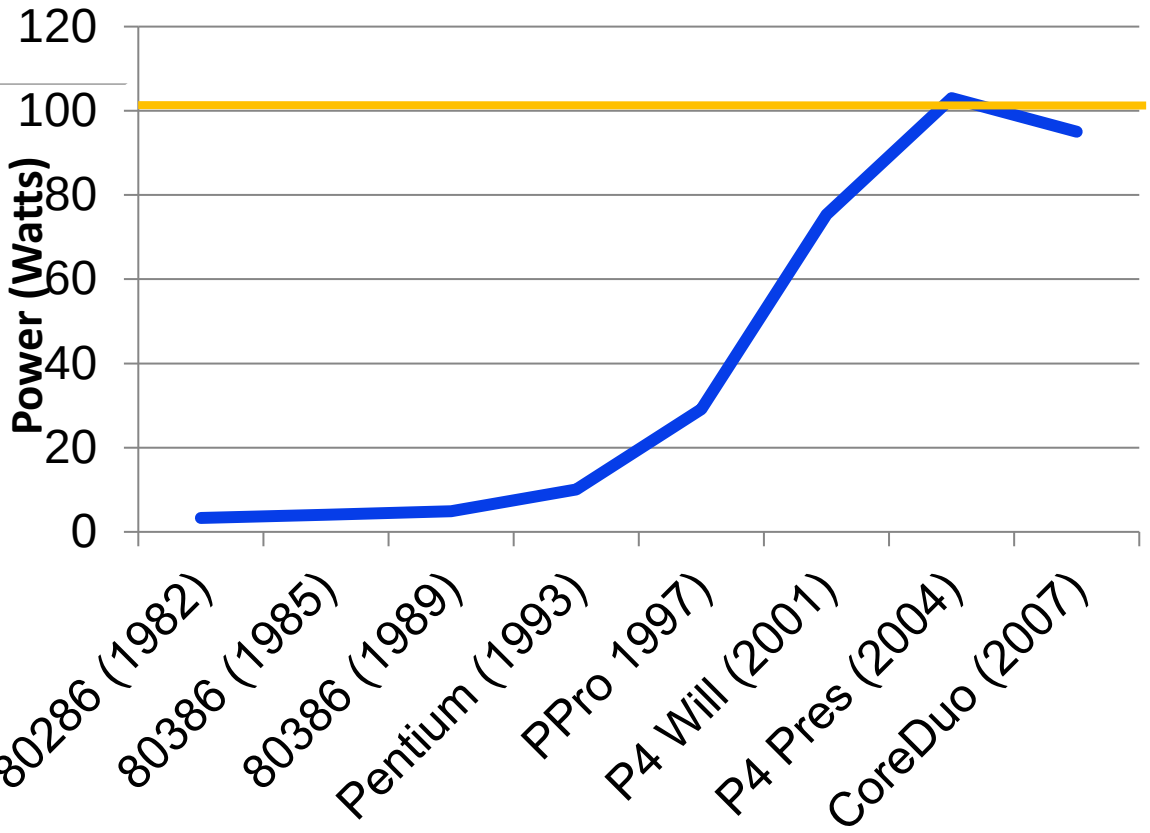
not discussed

Why clock rates stagnated

we were not interested in expending too much in cooling down anymore



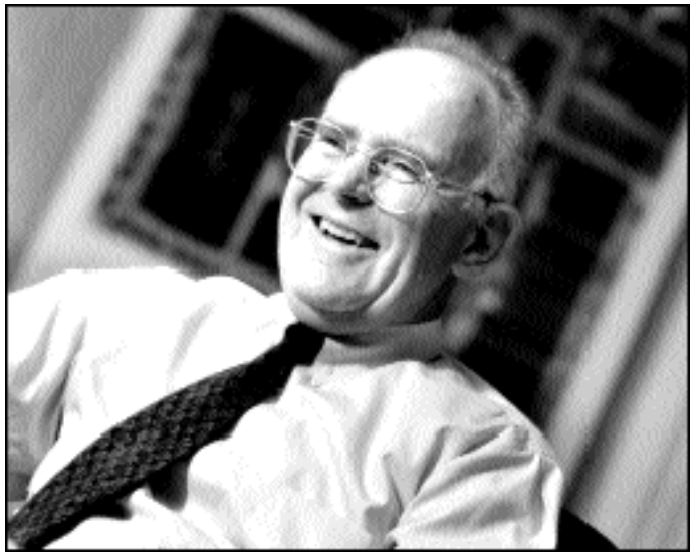
- Clock rates hit the "power wall"



Scaling 101: Moore's Law

scaling factor, by which smallest feature size reducing

90 65 45 32 22 16 11 8 nm



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

Scaling 101: Moore's Law

- Transistor count scales as S^2

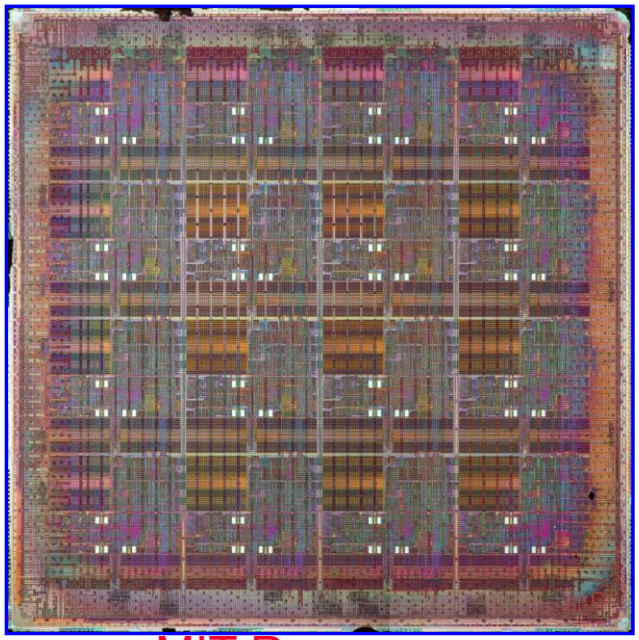
180 nm

16 cores

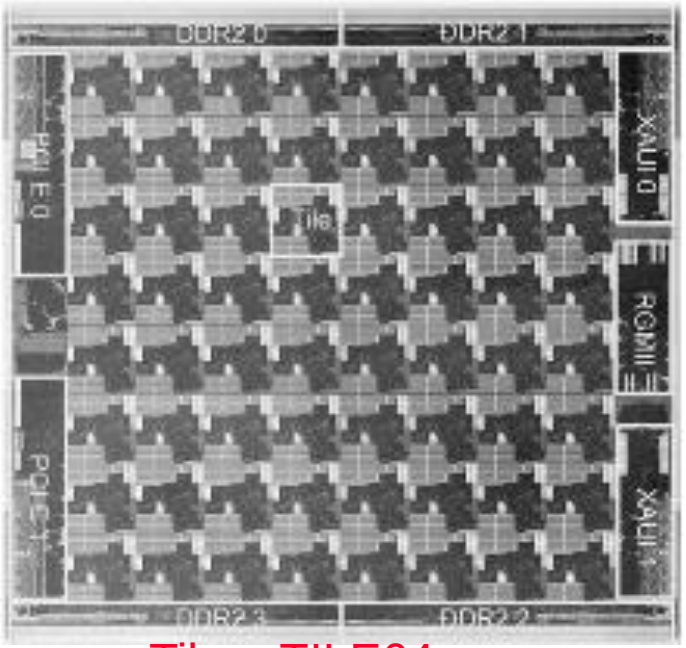
S = 2x
Transistors = 4x

90 nm

64 cores



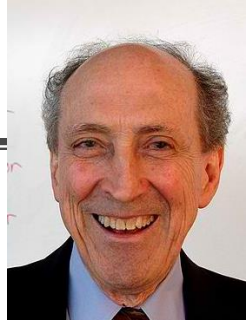
MIT Raw



Tilera TILE64

Advanced Scaling: ***Dennard -***

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



If S=1.4x …

- S^3

- S^2

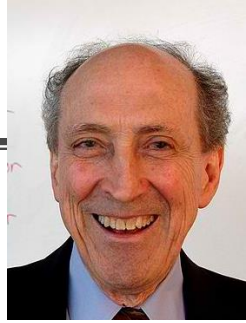
- S

- 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^3

S^2

$S^2 = 2x$
More Transistors

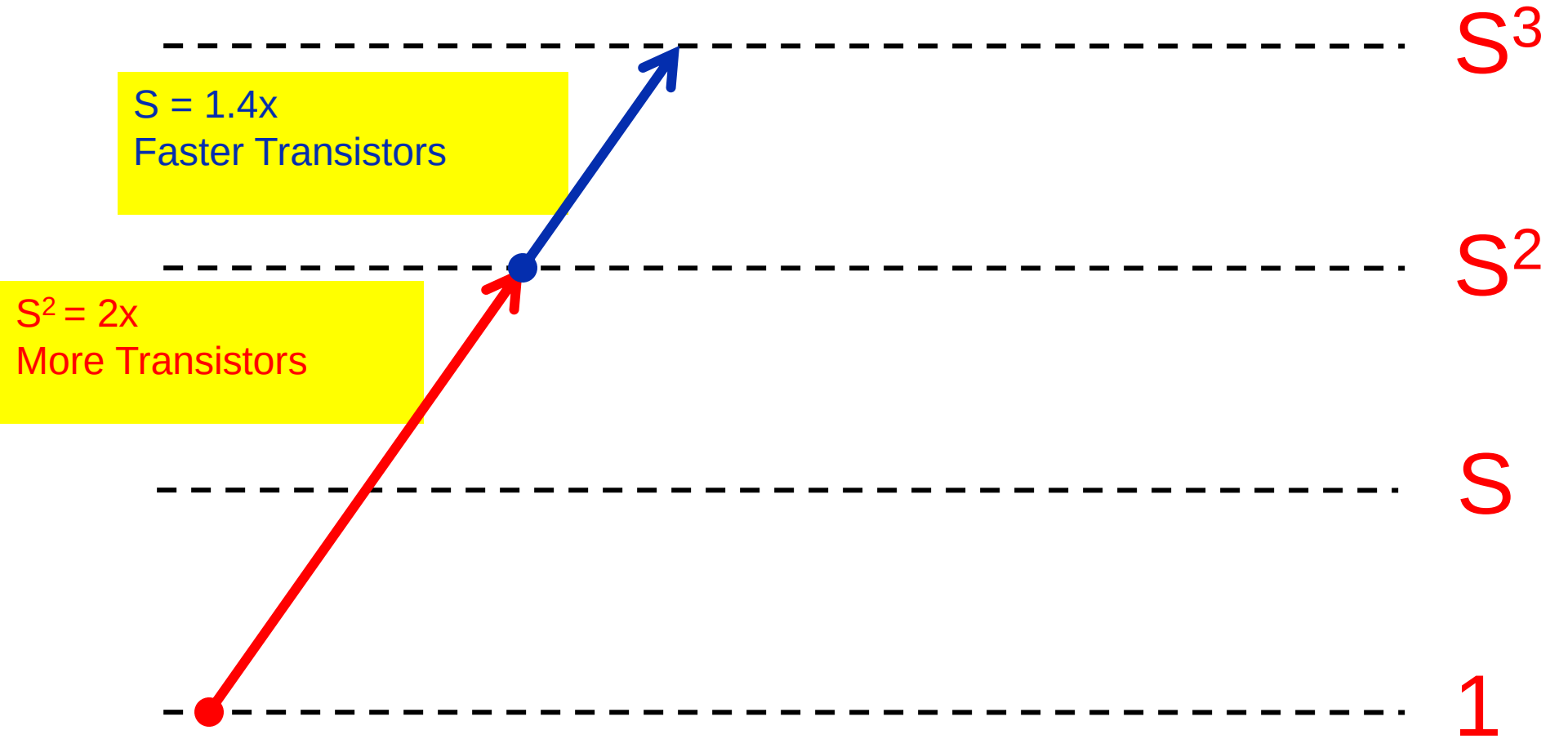
S

1

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 …

S^3

S = 1.4x
Faster Transistors
linearly faster

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

what about power??

S^2

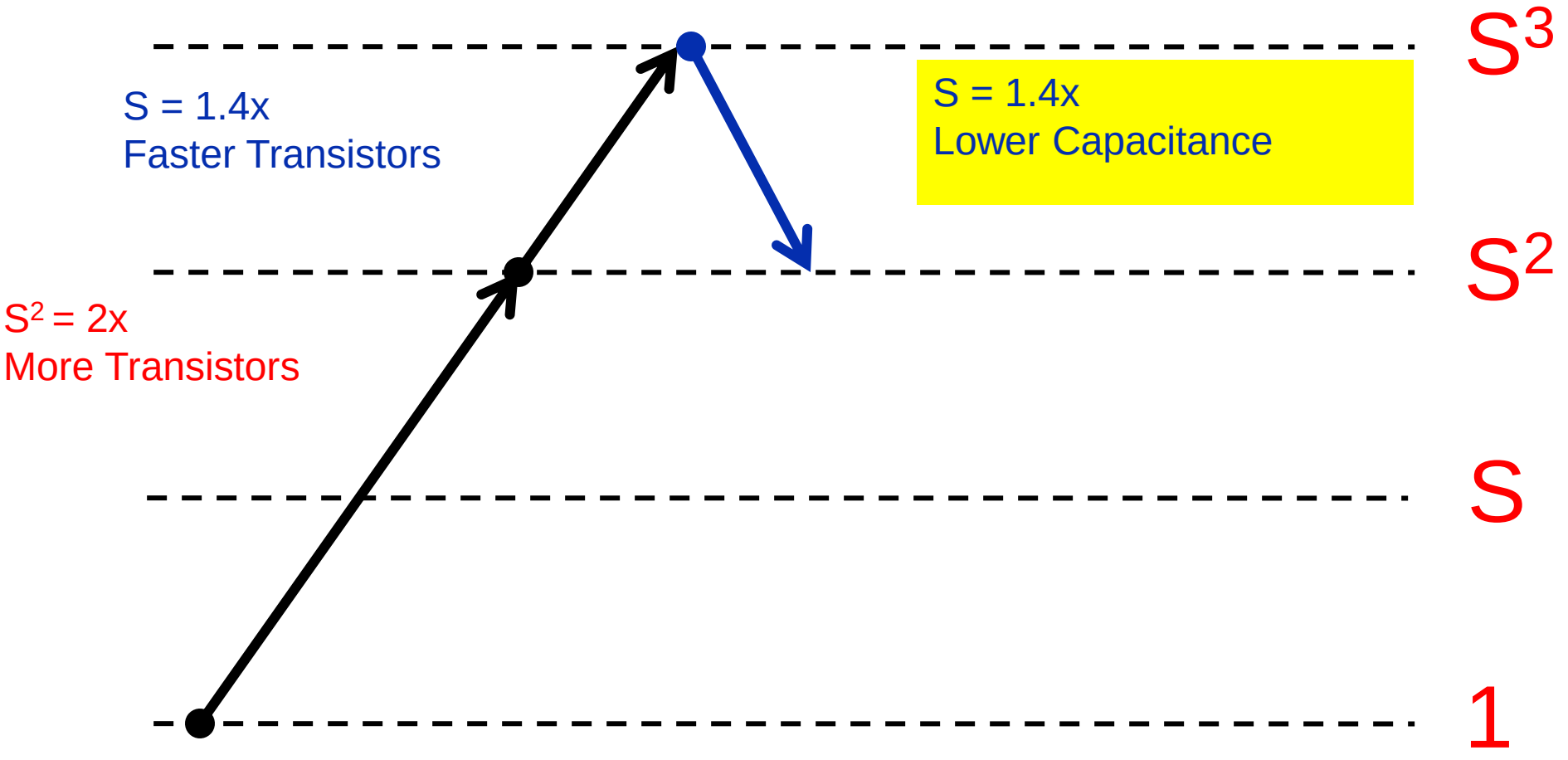
$S^2 = 2x$
More Transistors

S

1

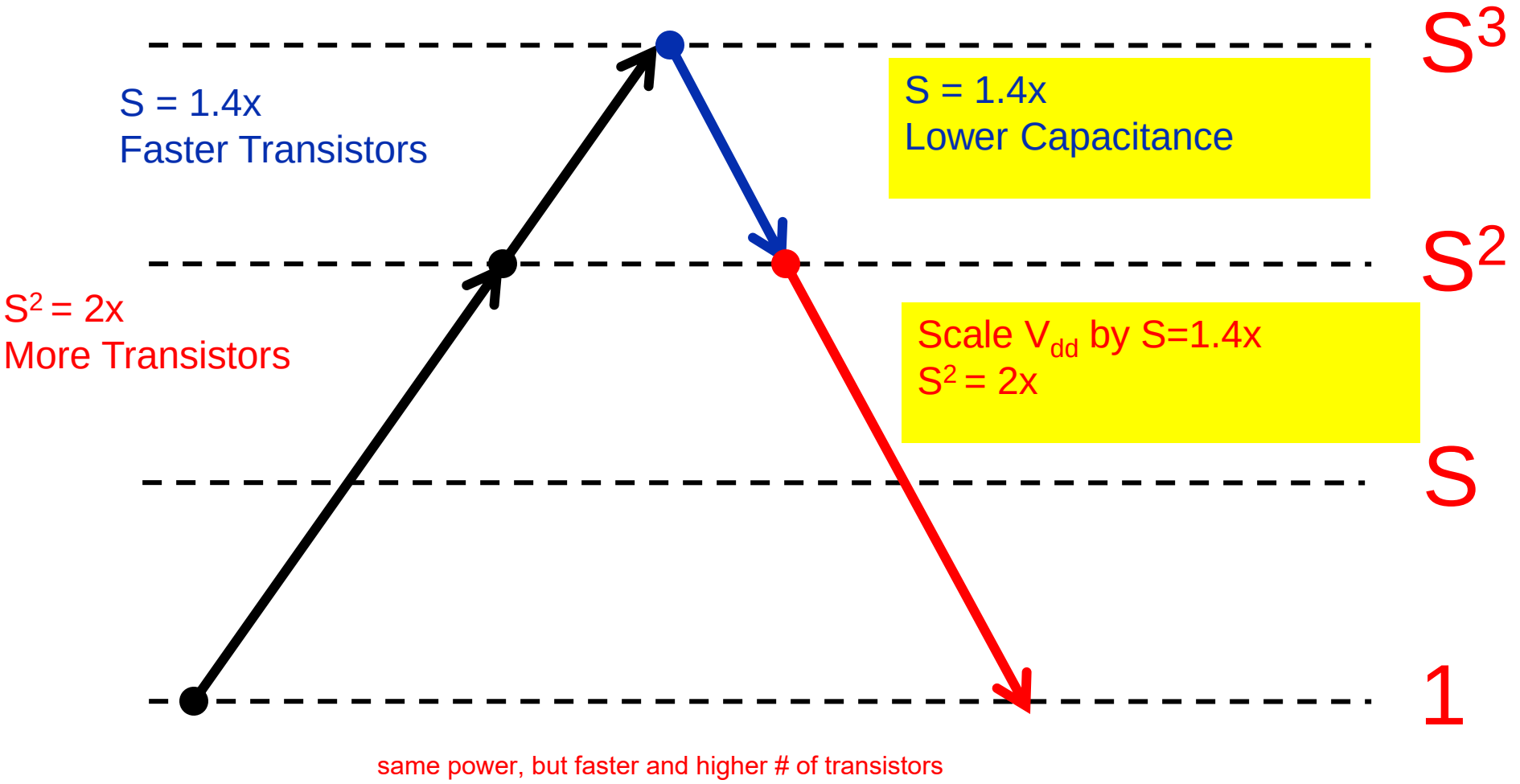
Advanced Scaling: ***Dennard –***

"We can keep power consumption constant"



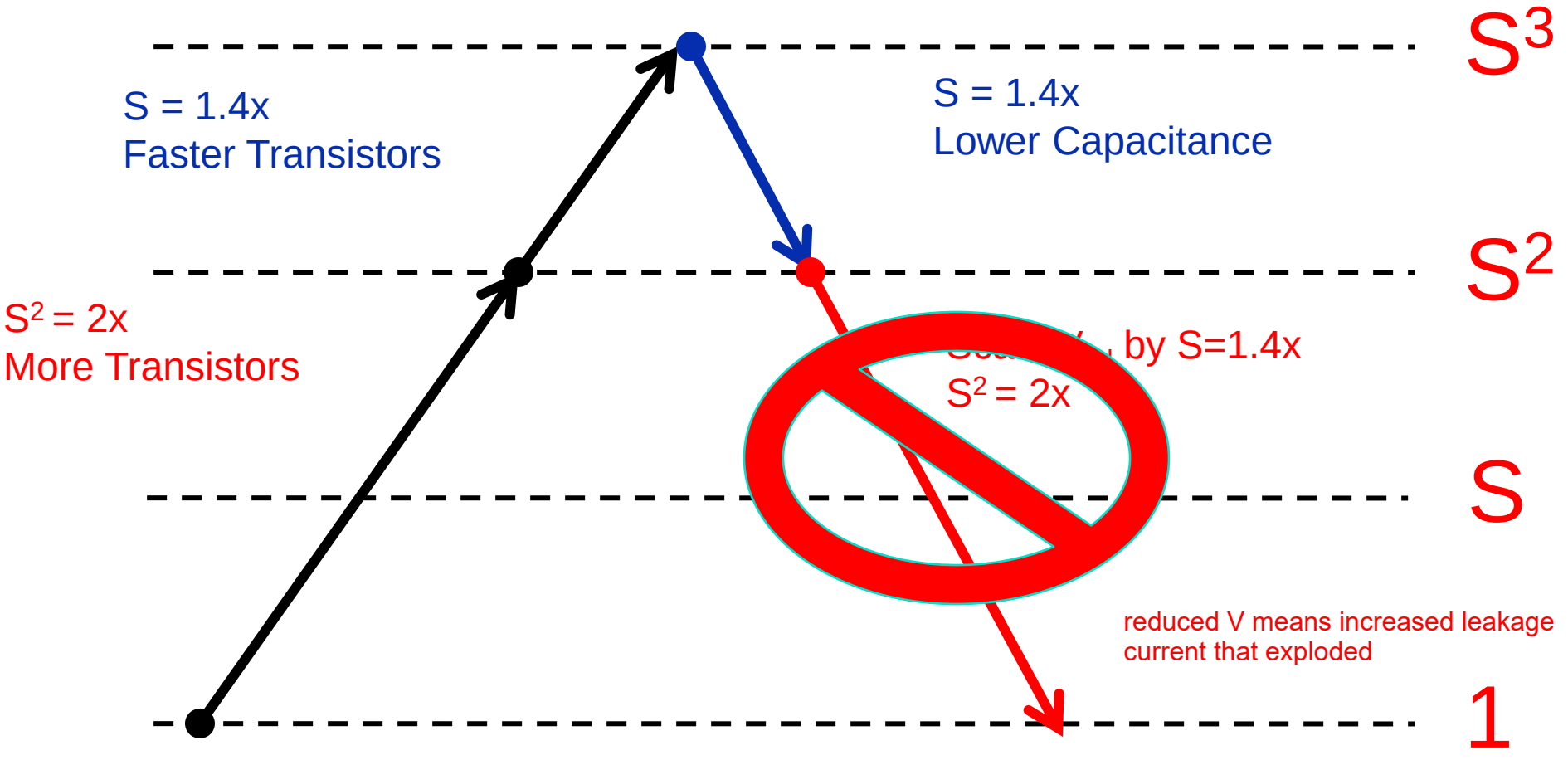
Advanced Scaling: ***Dennard –***

"We can keep power consumption constant"

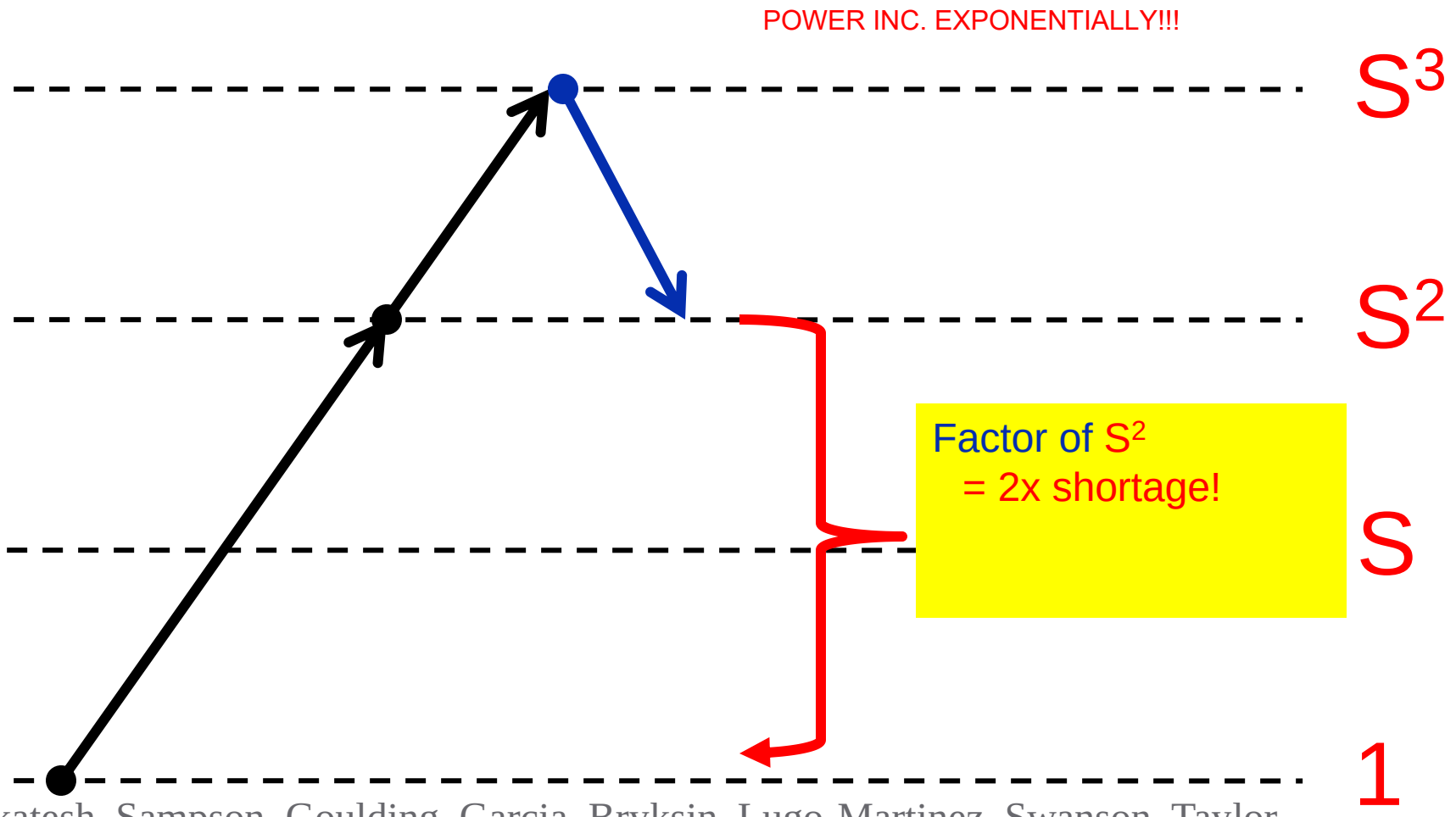


same power, but faster and higher # of transistors

Threshold scaling problems due to leakage



Full chip, full frequency power increases exponentially – 2x per process generation



Venkatesh, Sampson, Goulding, Garcia, Bryksin, Lugo-Martinez, Swanson, Taylor. *Conservation Cores: Reducing the Energy of Mature Computations*. (ASPLOS 2010)

So, how do we stay on the performance curve?

- Moore's law is continuing to give us twice as many transistors each generation (every ~~two~~(three?four?) years)
- Clock rates can increase only slightly with each generation (due to energy density limits)
- Have mined almost all the available ILP in a single-threaded application with pipelining, out-of-order superscalar datapaths, branch (and other) prediction, compiler optimizations, etc.

- But, we want to continue to stay on the performance curve (doubling every ~two(ish) years). How can we?

The multicore (CMP) revolution?

only lasted 3 years

2005: IBM Cell Microprocessor (8 Cores), Xbox 360 (3 Cores)

Multi/Many-Core Chips are here to stay
(which is a good thing)

…but the number of cores/chip grew slowly
(in general purpose consumer devices)
… until very recently (still behind the curve)

2011: Intel's Sandy Bridge (8 Cores, 32nm)

2012: Intel's Ivy Bridge (8 Cores, 22nm)

It was a “sea change”

- “We are dedicating all of our future product development to multicore designs. … This is a sea change in computing” Paul Otellini, President, Intel (2004)
- “We've gone through changes in the past," says Craig Mundie, Microsoft's chief research and strategy officer. But this one, he says, is the most "conceptually different” change “in the history of modern computing.”

  improvement was not like before with many cores

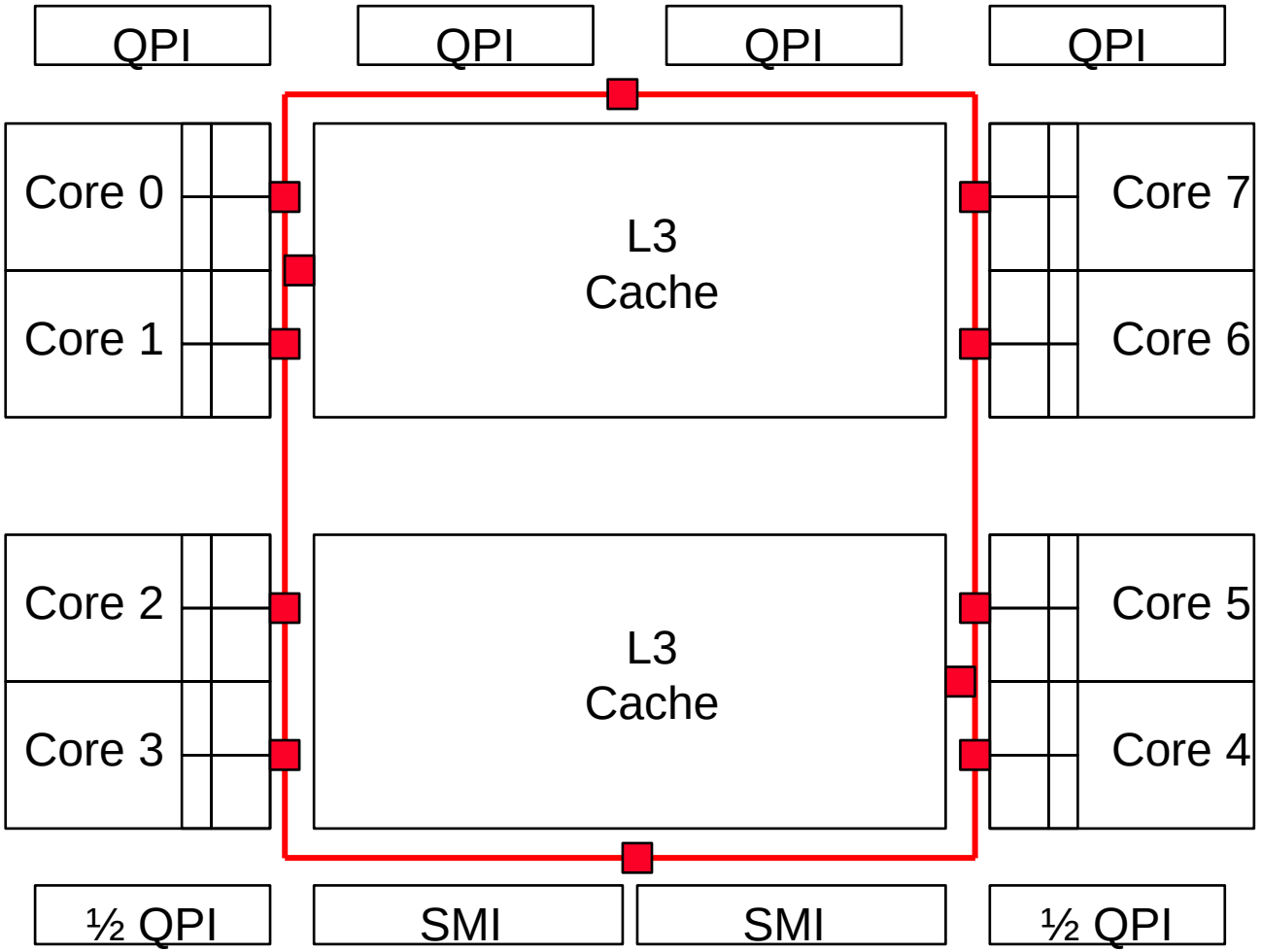
- Architectures are not ready for 1000 core/chip
  - Unlike ILP, cannot be solved by just by architects and compiler writers alone, but also cannot be solved *without* architects
- Algorithms, programming languages, compilers, operating systems, libraries, … not ready to supply TLP or DLP for 1000 cores/chips

Could multicore even get us where we want to go?

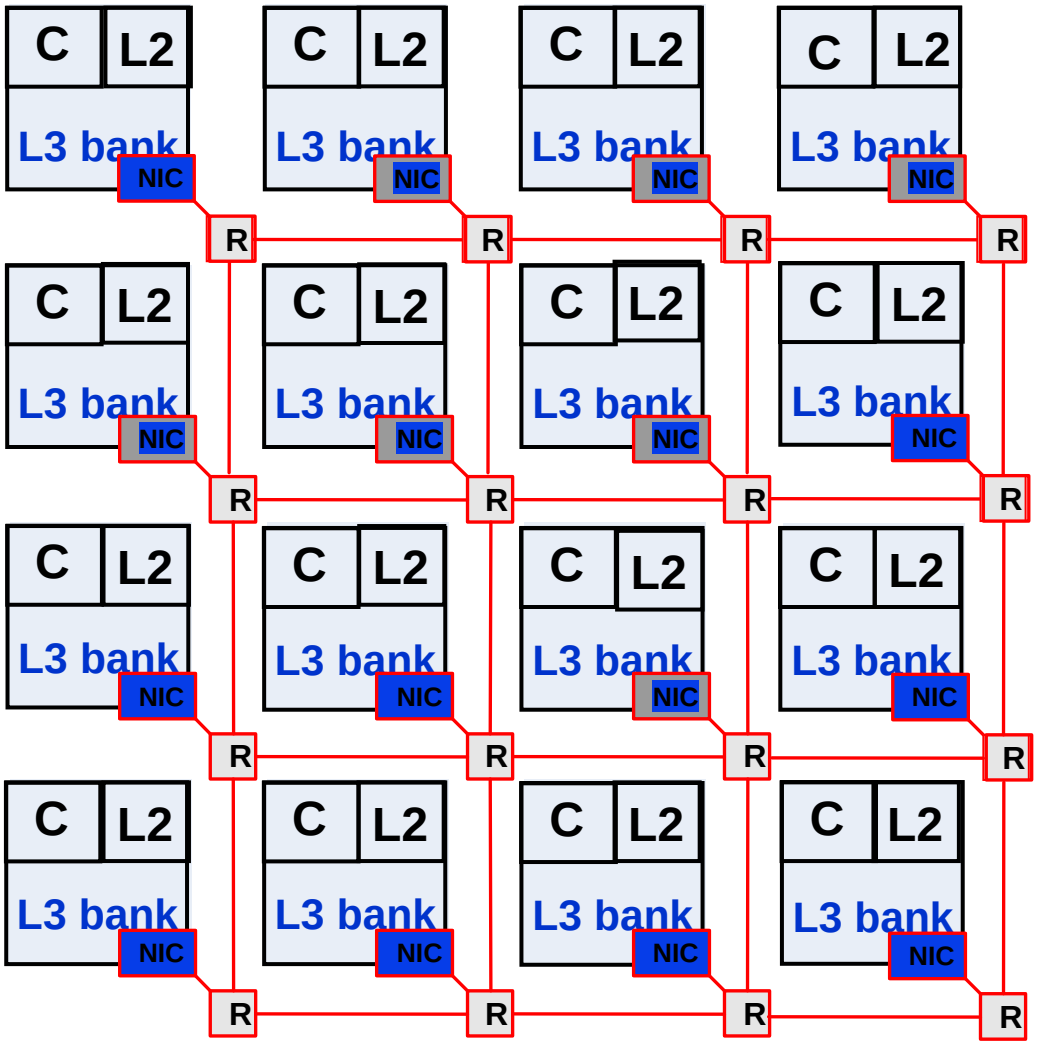
- Rate of increase in # of cores lags previous increases in performance
- Even a parallel revolution may not be able to use all the cores we can build – see: Esmaeilzadeh, et al. “Dark Silicon and the End of Multicore Scaling.” ISCA 2011
- Amdahl’s Law this very law is about multiprocessors
- Can argue that “Multicore Era” only lasted from 2005-2008 (cancelling of Larabee) and the “Heterogeneous computing” era started thereafter

one uniform model for all multicore systems, ruling out GPUs

Abstracted multi-core (a la 2011)



Abstracted multicore of the future



- Private L1 & L2 $
- Shared LLC cache one bank per core
- NoC (Network on Chip) interconnect

Trends in cost

- Cost of an integrated circuit

$$\text{IC Cost} = \frac{\text{Die Cost + Die Testing Cost + Die Packaging and Test Cost}}{\text{Final test yield}}$$

- And the cost of the die is

$$\text{Die Cost} = \frac{\text{Cost of wafer}}{\text{Dies per wafer x Die yield}}$$

- The manufacturing process dictates the wafer cost and the defects per unit area, so the sole control of the architect is the die area
  - Bigger dies mean fewer dies per wafer (so more costly)
  - Bigger dies mean lower die yield (so more costly) and smaller dies mean higher die yield

Other "cost" considerations

Moore's law -> opportunities -> difficult to realize those opportunities

- "Economic realities are staggering" Doug Gross, CEO Global Foundries http://videos.dac.com/47th/grose/grose.html
  - The cost of an advance fabrication line (e.g., 45nm)
    - New *process* R&D costs of $600M to $900M (for 45nm); $1300M (for 22nm)
    - Fab start-up costs –$3.5B to $4B (for 45nm); $4.5Bto $6B (for 22nm)

What is a virtuous cycle could easily be a vicious cycle

  - The cost of the mask set for a 45nm design
    - Many layers in the process (e.g., 11 layers of copper (Cu) interconnect in the 32nm IBM process)
- The cost of design & verification of a complex processor
  - ~$100M (500 person-years @ $200K per person)
  - Not every processor is an entirely "new start" so some of the costs can be amortized over processor generations

Trends in dependability

DRAM - redundancy to ensure reliability in cases of failures
Not so easy for processors

- Used to be able to count on the (near) fact that once the design and implementation was verified for correctness and the chips fabbed and tested as working that reliability of the part was assured

Packaging: used to be of ceramics but we had to switch to plastics

- But as transistors get smaller, they are less reliable
  - Smaller capacitance means fewer electrons to store the "1"
- Transient faults (soft error upsets (SEU))

Occassional glitches

  - A stored bit "flips" randomly (on a stored capacitor) from a hit by a cosmic ray (much more susceptible at higher altitudes and lower V_{dd}'s); state is flipped but device is fine
- Permanent (or hard) faults – device aging

Very difficult because initially it was OK

  - A gate (or memory cell) wears out (from use); it breaks and stays broken
    - Temperature (BTI, oxide breakdown, etc.), electromigration, etc.

Everything has to be thought of as a stochastic process
And think about expected reliabilities

Aside: The dependability "equations"

- Module reliability – measure of continuous service (or time to failure)

1. Mean Time to Failure (MTTF) measures Reliability
2. Failures in Time (FIT) = 1/MTTF measures the rate of failures
   - Traditionally reported as failures per billion hrs of operation

- Mean Time to Repair (MTTR) measures service interruption
  - Mean Time Between Failures (MTBF) = MTTF + MTTR
- Module availability – MTTF/ (MTTF + MTTR)
  - A value between 0 and 1, higher is better!

Delivering dependability

- The job of both the circuit designer and the architect
- In memories it's a bit easier
  - Add an extra bit to each word to detect a bit-flip (parity)
  - More elaborate (and expensive in terms of additional bits and detection hardware) to do single error detect and correct (ECC), double error detect, single error correct (SECDED)
  - Cache scrubbers are common these days (read, check parity, set to Invalid if parity is wrong)
- With logic it's a lot harder
  - Have redundant components (three with "voting")
  - Use multithreading to run two copies of the same thread and check them against one another when completed
  - To first order, device aging is proportional to device stress time (i.e., the average duty cycle) so design for balanced duty cycles (i.e., balanced device on/off-time)

All DRAMs have errors

Some ECC for every DRAM is normal

extremely expensive