ECE 658
Advanced Computer Architecture
Fall 2018

Administrivia
The need for Parallel Computing
Introduction to Parallel Computer Architecture

Nikos Bellas

Computer and Communications Engineering Department
University of Thessaly

Διαδικαστικά

- Όνομα διδάσκοντος: Νίκος Μπέλλας
- Email: nbellas@inf.uth.gr
- Personal Webpage: www.inf.uth.gr/~nbellas

- Webpage: www.inf-server.inf.uth.gr/courses/CE658
- Γραφείο: Γκλαβάνη 37, Γραφείο Β3.7

- Ώρες μαθήματος: Τρίτες 10:00-13:00
- Ώρες γραφείου: Μετά το μάθημα και με ραντεβού

Προαπαιτούμενα για το μάθημα

- *ECE432 Computer Architecture* ή να έχετε (αυτο)διδαχθεί τα κεφάλαια 1-4 του βιβλίου "Computer Architecture: A Quantitative Approach", by J. Hennessy, D. Patterson, >= 3η έκδοση
- Καλή γνώση της γλώσσας C
- Ψηφιακή σχεδίαση και Λογικό σχεδιασμό
- Βασικές γνώσεις σε Unix/Linux
- Πολύ καλή γνώση της Αγγλικής, κυρίως της σχετικής ορολογίας
- Όρεξη γιά δουλειά και επιθυμία για γνώσεις πάνω σε "Computer Architecture"
- Τα slides και η ορολογία θα είναι στα Αγγλικά

- Θα συμβούλευα να μην πάρετε το μάθημα εάν δεν έχετε εμπειρία απο προηγούμενα μαθήματα όπως το *CE432*
  - Εκτός εαν μπορείτε να καλύψετε το χαμένο έδαφος με προσωπική μελέτη.

Curriculum I

- **Introductory Material**
  - Introduction to Parallel Computer Architecture
  - The need for parallel architectures
  - Patterns of Parallelism
- **Instruction and Data Level parallelism**
  - Superscalar and Out Of Order execution – Precise exceptions - Reorder Buffers
  - **Case study**: Multicore architecture from Intel (Nehalem, Sandy Bridge)
  - VLIW technology, Loop scheduling, Software pipeline, Modulo Scheduling
  - Superblocks, Hyperblocks, Predication, Speculation
  - **Case study** : Itanium ISA

Curriculum II

- **Thread Level Parallelism**
  - Simultaneous Multithreading (SMT)
  - **Case study**: Power6 and Power7 from IBM
  - Shared and Distributed Shared Memory Systems
  - **Case study**: GPUs and the CUDA API
  - Memory Coherence and Memory Consistency models
  - Synchronization primitives
  - Transactional Memory Hardware & Software
  - **Case study** : Sun Niagara processor
  - Streaming (Compilation issues, processors)
  - **Case Study** : Merrimac from Stanford and RSVP processor from Motorola

Curriculum III

- Customizable processors
- Approximate computing
- DRAM technology and memory controllers

Συγγράμματα

- *"Computer Architecture: A Quantitative Approach", by J. Hennessy, D. Patterson, Morgan Kaufmann Publishers, 3-4-5η έκδοση*
- *"Parallel Computer Architecture: A Hardware/Software Approach", by D. Culler, JP Singh Morgan Kaufmann Publishers, 1999*

- Επιλεγμένες δημοσιεύσεις από συνέδρια "Αρχιτεκτονικής Υπολογιστών" όπως (ISCA, Micro, HPCA, κοκ).
- Συμβουλή: Το internet έχει ένα τεράστιο αριθμό από πηγές για "Computer Architecture". Χρησιμοποιείστε τις.

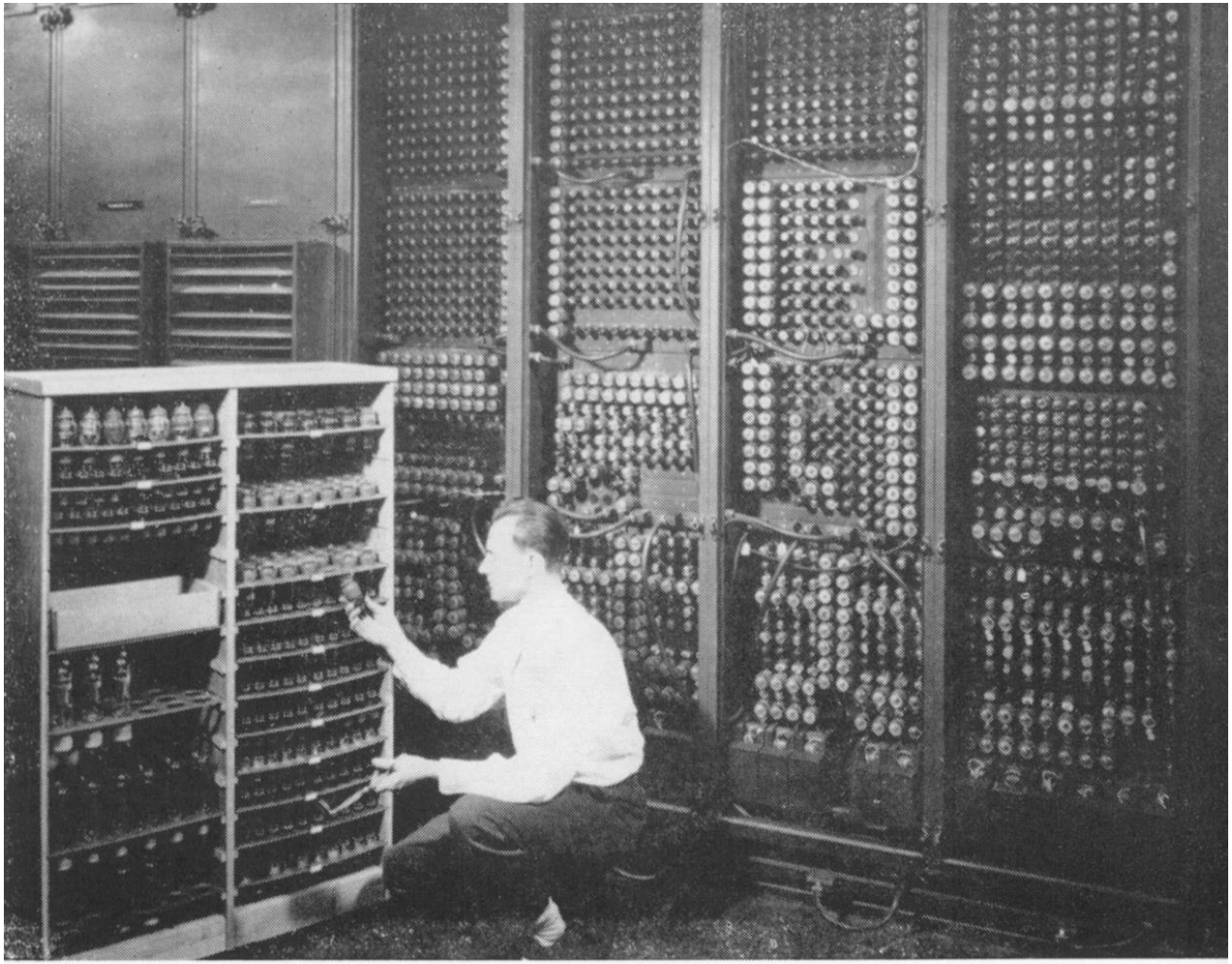
Grading

- **Final Exam: 30%**
  - You should take at least 5 to pass the class
- **Term project : 40%**
  - Some ideas:
    - Study a set of research papers on a topic of your interest, write a comprehensive summary and present to the class
    - Multiprocessor system analysis and benchmarking using the gem5 simulator
    - Other
- **Homeworks : 30%**

- **Notes:**
  - this is a demanding class. You have to devote a lot of time to keep up with the lectures, and do the homework, the project and paper study.

Let's get started

The Past…

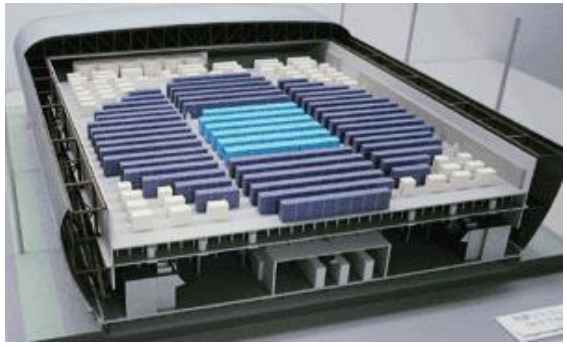


Replacing a bad tube meant checking among ENIAC's 19,000 possibilities.

ENIAC, "US Army photo, around 1946"

The present…

Today a $500 laptop has more performance, more main memory and more disk storage than a computer bought in 1985 for $1 M



What is Computer Architecture

- Computer architecture is a description of the structure and the functionality of a computer system.

- Computer architecture comprises at least three main subcategories:

- Instruction set architecture (ISA), also known as assembly language is the lowest point of control of the programmer on the processor.
- Microarchitecture or Computer Organization is at a lower level description of the system. What are the modules of the system, how they interconnect and how they interact. Microarchitecture is beyond the control of the programmer. For example, the number of functional units in a CPU is a microarchitectural detail.
- System Design which includes all of the other hardware components within a computing system such as system interconnects, memory hierarchies, peripherals, etc. System design is sometimes visible to the programmer.

ISA vs. Computer Architecture

- Old definition of computer architecture = instruction set design
  - Other aspects of computer design called implementation
  - Insinuates implementation is uninteresting or less challenging
- Today computer architects do much more. Technical hurdles *more* challenging than in the earlier days
- Two very important trends:
  - Implementation of microarchitecture has become critical, and more so as technology scales down
  - What really matters now is the whole system, NOT only the CPU (end-to-end system design). Computer architecture is an *integrated approach*
- All these are in the plate of the computer architect.

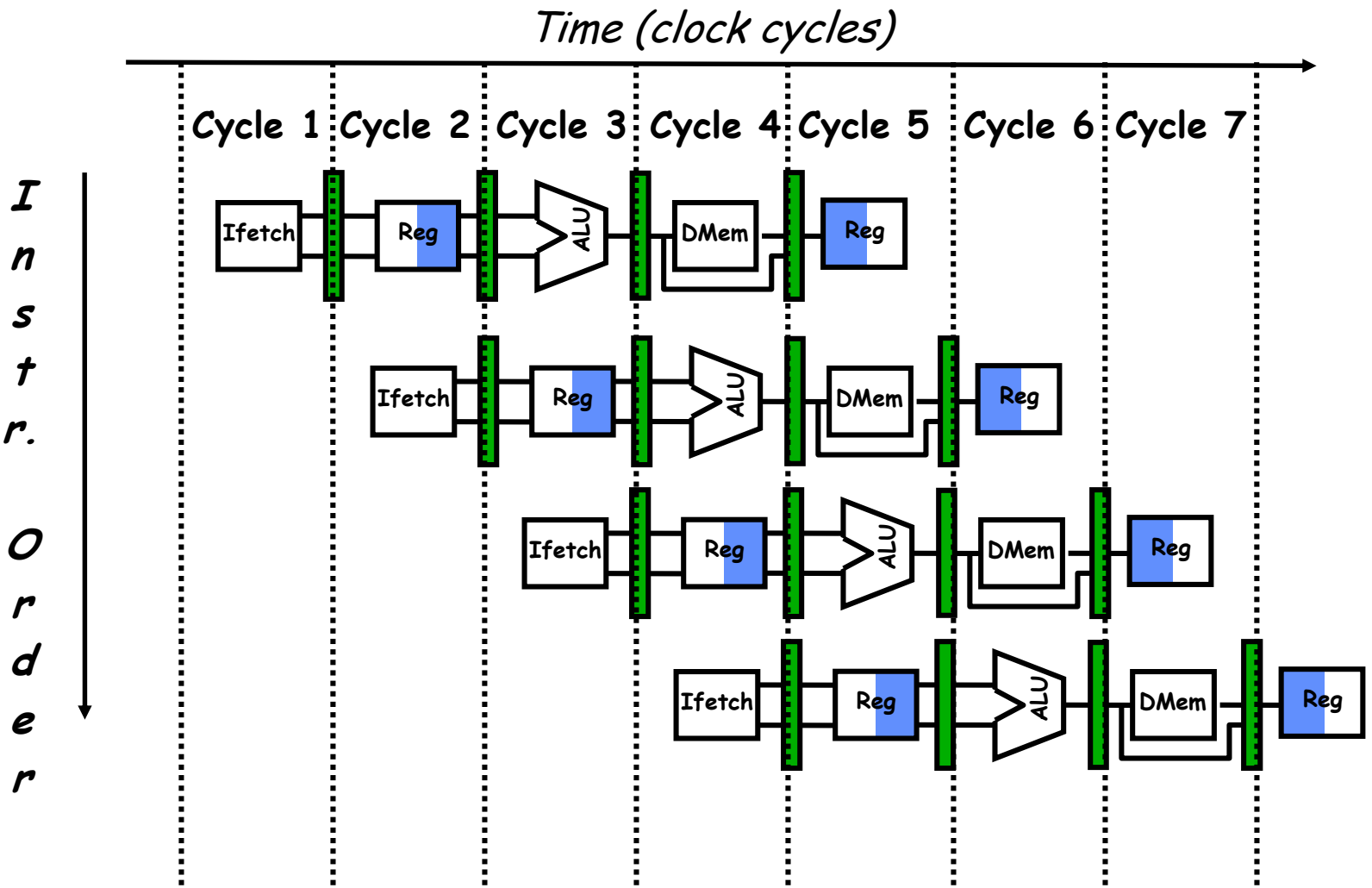
So, what does a good computer architect do

- Exploit quantitative principles of design
  - Take Advantage of Parallelism
  - Principle of Locality
  - Focus on the Common Case
  - Amdahl's Law
  - The Processor Performance Equation
- Performs careful, quantitative comparisons
  - Define, quantify, and summarize relative performance, cost, dependability, power dissipation of multiple solution
- Anticipate and exploit advances in technology
- Define and thoroughly verify well-defined interfaces

1) Taking Advantage of Parallelism

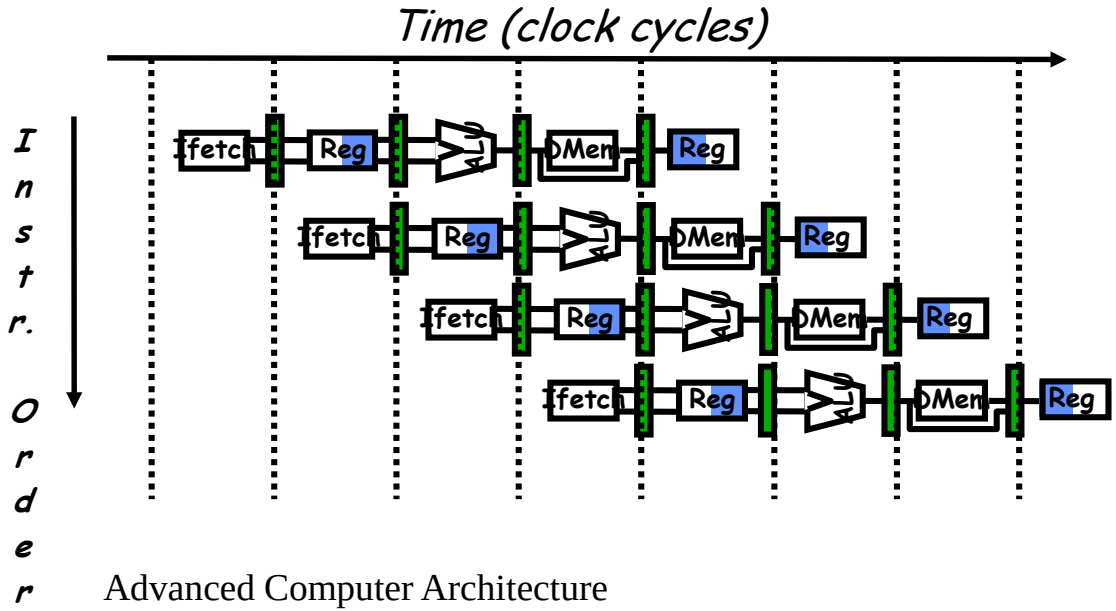
- Parallelism in Space
  - Multiple CPUs running different threads of the program
  - Carry lookahead adders uses parallelism to speed up computing sums from linear to logarithmic in number of bits per operand
  - Multiple memory banks searched in parallel in set-associative caches
- Parallelism in Time
  - overlap instruction execution (pipelining) to reduce the total time to complete an instruction sequence
  - Not every instruction depends on immediate predecessor $\Rightarrow$ executing instructions completely/partially in parallel possible
  - Classic 5-stage pipeline:
    1) Instruction Fetch (Ifetch),
    2) Register Read (Reg),
    3) Execute (ALU),
    4) Data Memory Access (Dmem),
    5) Register Write (Reg)

Pipelined Instruction Execution



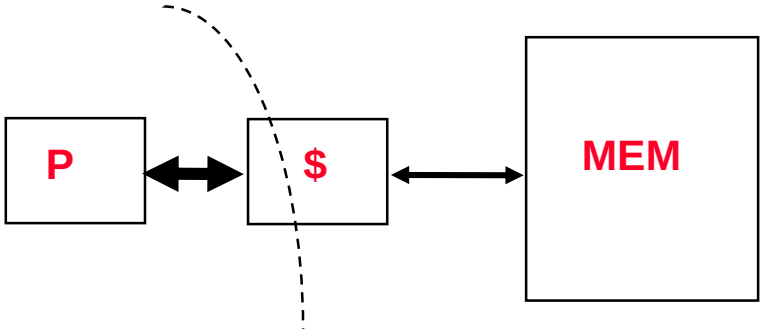
Limits to pipelining

- Hazards prevent next instruction from executing during its designated clock cycle
  - Structural hazards: attempt to use the same hardware to do two different things at once
  - Data hazards: Instruction depends on result of prior instruction still in the pipeline
  - Control hazards: Caused by delay between the fetching of instructions and decisions about changes in control flow (branches and jumps).

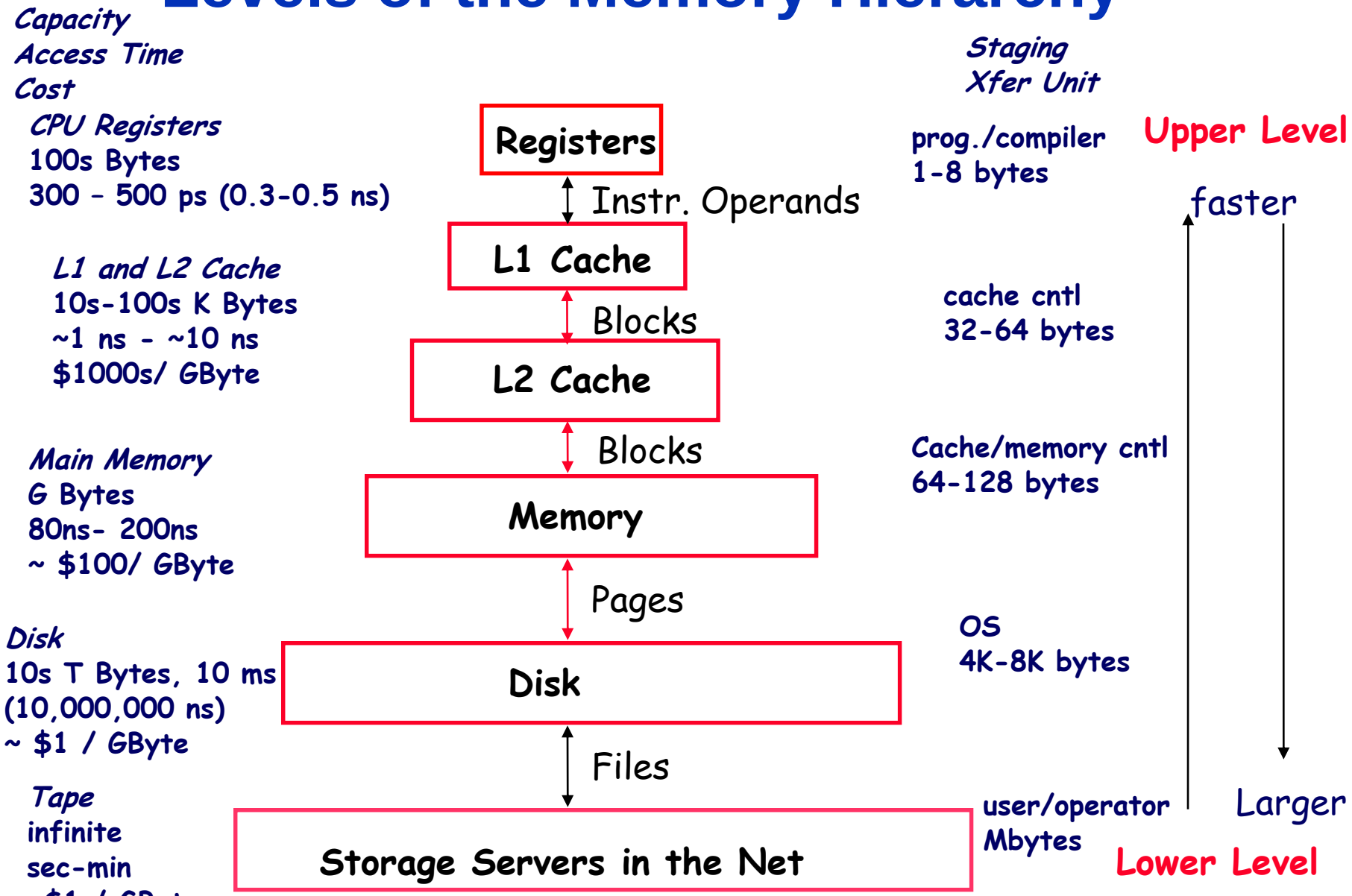


2) The Principle of Locality

- The Principle of Locality:
  - Program access a relatively small portion of the address space at any instant of time.
- Two Different Types of Locality:
  - Temporal Locality (Locality in Time): If an item is referenced, it will tend to be referenced again soon (e.g., loops, reuse)
  - Spatial Locality (Locality in Space): If an item is referenced, items whose addresses are close by tend to be referenced soon (e.g., straight-line code, array access)
- Last 30 years, computer architecture relied on locality for memory perfomance



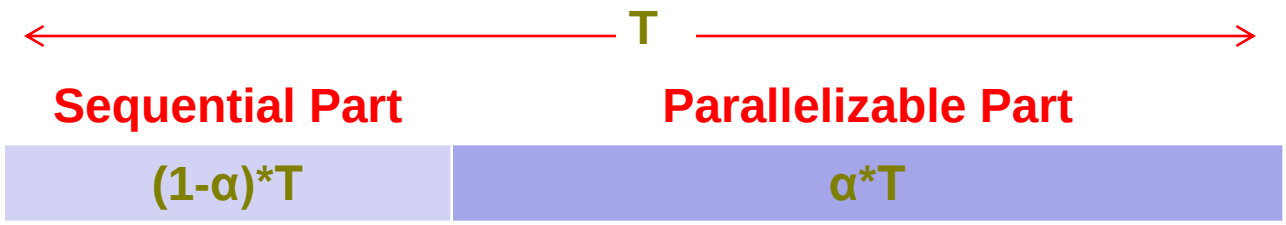
Levels of the Memory Hierarchy



3) Focus on the Common Case

- Common sense guides computer design
  - Since we're in engineering, common sense is valuable
- In making a design trade-off, favor the frequent case over the infrequent case
  - E.g., Instruction fetch and decode unit used more frequently than multiplier, so optimize it first
- Frequent case is often simpler and can be done faster than the infrequent case
  - E.g., overflow is rare when adding 2 numbers, so improve performance by optimizing more common case of no overflow
  - May slow down overflow, but overall performance improved by optimizing for the normal case
- What is frequent case and how much performance improved by making case faster => Amdahl's Law

4) Amdahl's Law



$$ExTime_{new} = ExTime_{old} \times \left[(1-\alpha) + \frac{\alpha}{Speedup_{enhanced}} \right]$$

$$Speedup_{overall} = \frac{ExTime_{old}}{ExTime_{new}} = \frac{1}{(1-\alpha) + \frac{\alpha}{Speedup_{enhanced}}}$$

Best you could ever hope to do (perfect speedup):

$$Speedup_{maximum} = \frac{1}{(1-\alpha)}$$

Amdahl's Law example

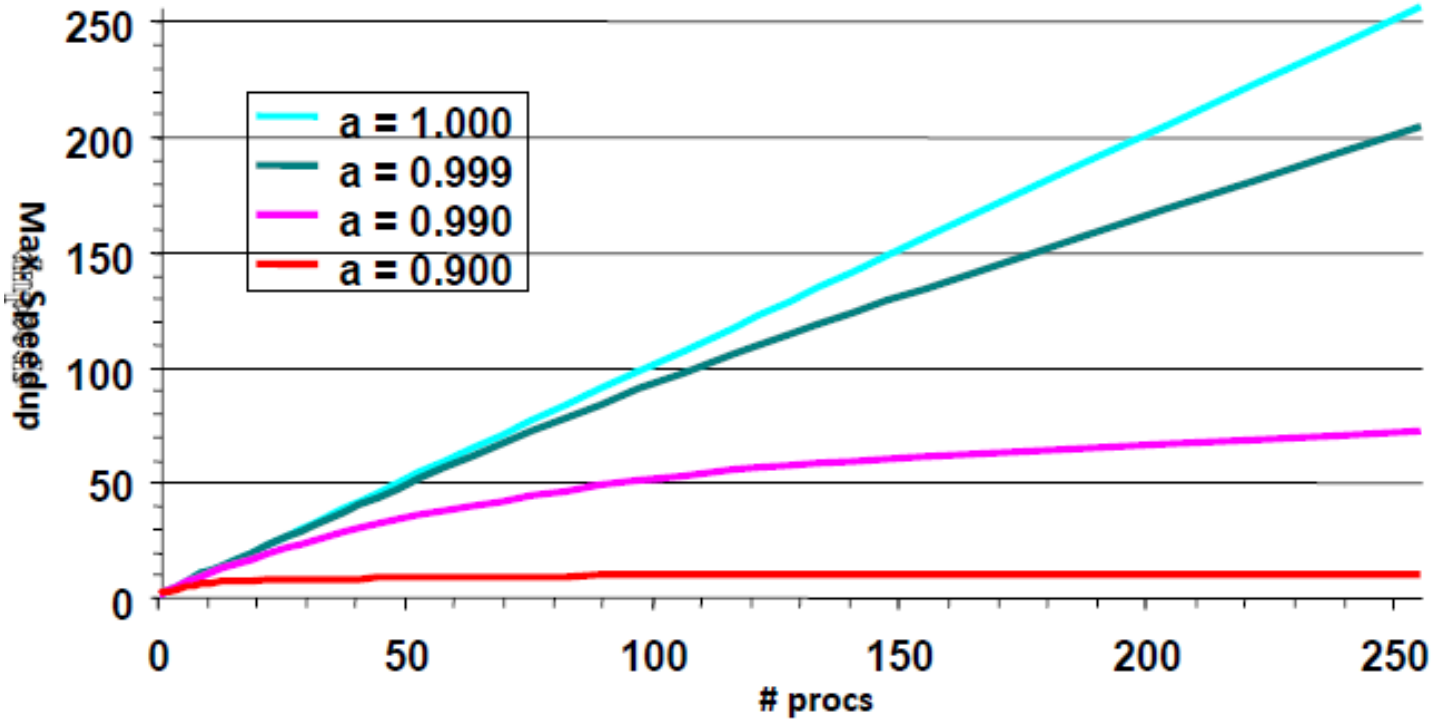
- New CPU 10X faster
- I/O bound server, so 60% time waiting for I/O

$$\text{Speedup}_{\text{overall}} = \frac{1}{(1-\alpha) + \frac{\alpha}{\text{Speedup}_{\text{enhanced}}}} =$$

$$\frac{1}{(1-0.4) + \frac{0.4}{10}} = 1.56$$

- Apparently, it's human nature to be attracted by 10X faster, vs. keeping in perspective it's just 1.6X faster

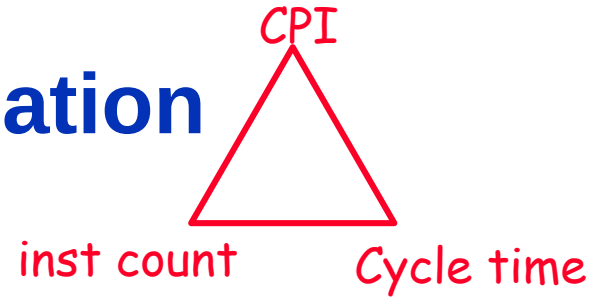
Amdahl's Law for different parallelism granularities



Amdahl's law main idea: the sequential part of an application limits performance scaling.

5) Processor performance equation

"Iron Law of Performance"



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

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

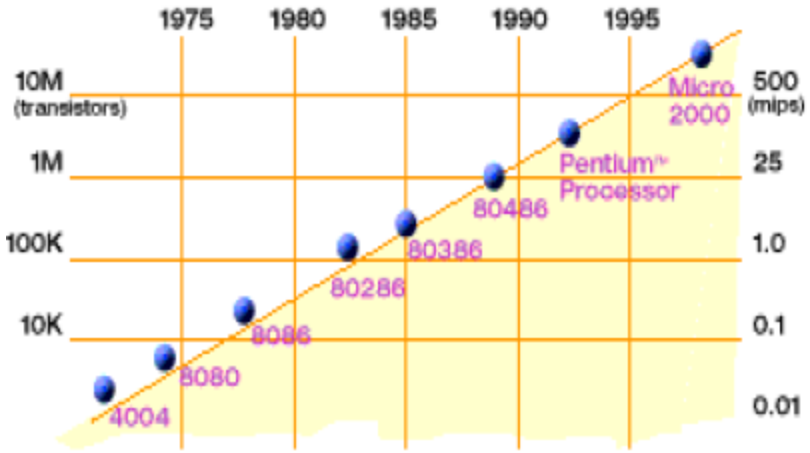
What drives new architectures ? Why are there so many of them?

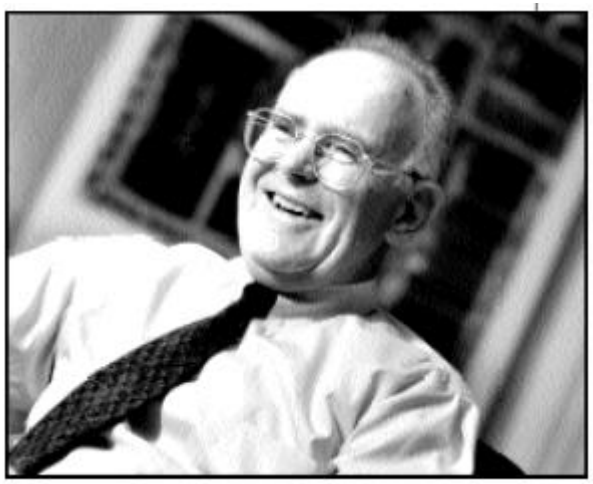
- Technology
  - Determines what is plausible and what is not
  - What is cheap and what is expensive in terms of performance, cost and power
  - For example, memory hierarchy (e.g. cache memories) became necessary due to DRAM technology being much slower than CPU technology
- Applications
  - High volume, mainstream applications drive architectural decisions
  - SIMD parallelism driven by a market need for multimedia products

Technology drives new architectures

- General-purpose unicores have stopped historic performance scaling
  - Power consumption
  - Wire delays
  - DRAM access latency
  - Diminishing returns of more instruction-level parallelism

Transistors and Clock Rating





Microprocessor have become smaller, denser and more powerful

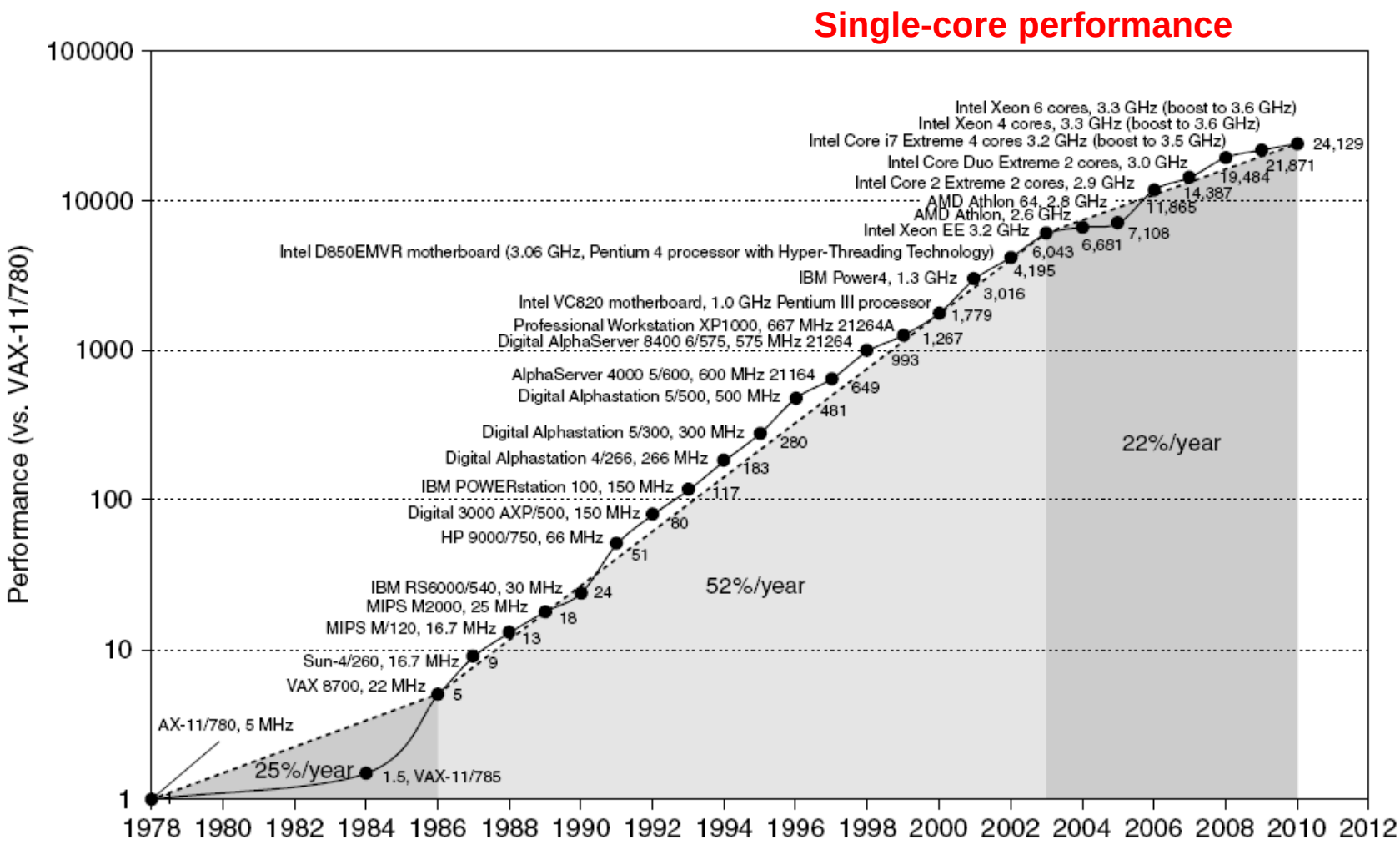
Gordon Moore, Intel's cofounder predicted in 1965 that transistor density will double every 24 months*

*Later revisited to 18 months

Technology drives new architectures

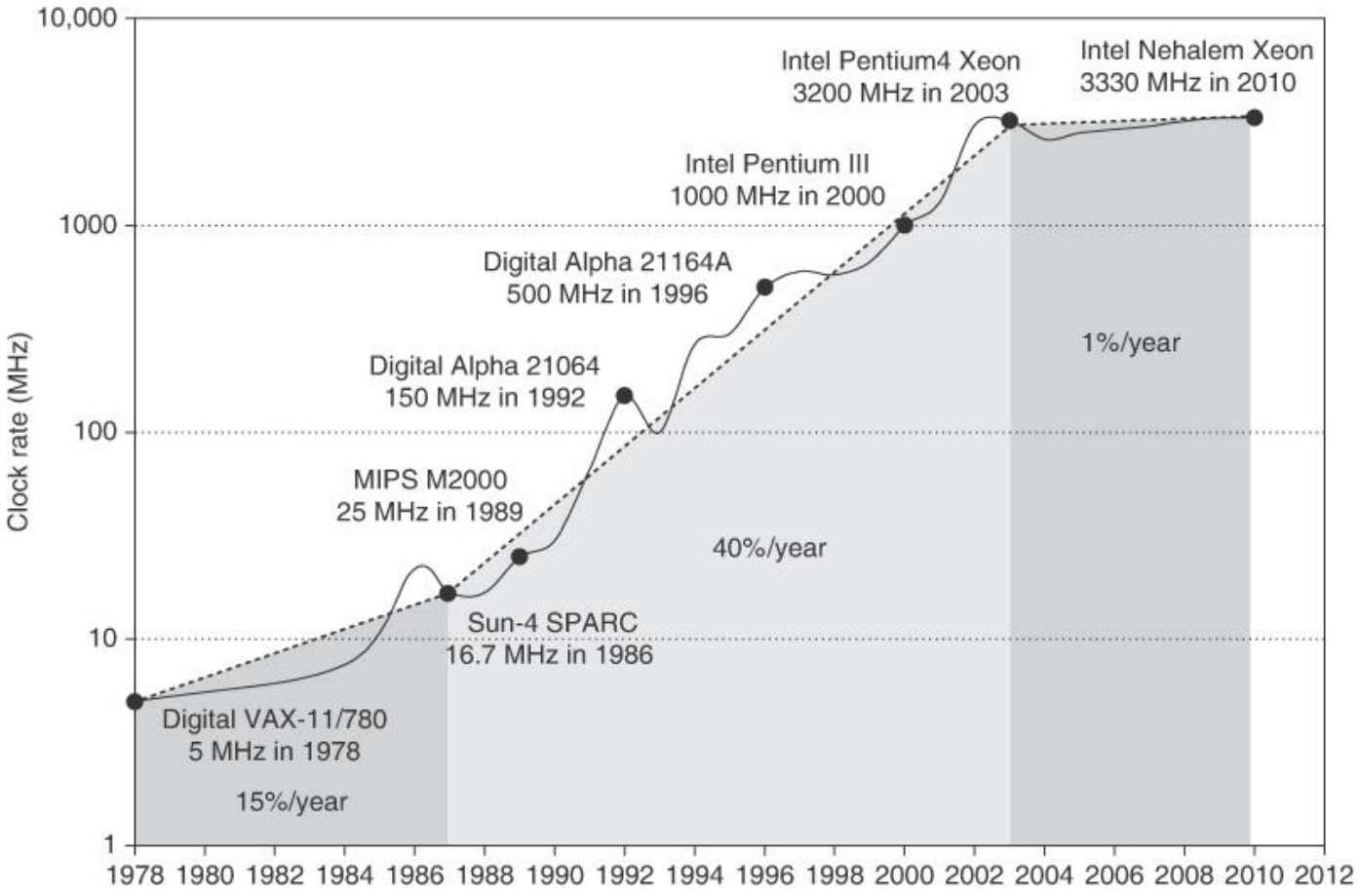
General-purpose unicores have stopped historic performance scaling

From Hennessy and Patterson, *Computer Architecture: A Quantitative Approach*, 5th edition, September 2011



Technology drives new architectures

Growth in clock frequency has stalled since 2003/04



Technology drives new architectures

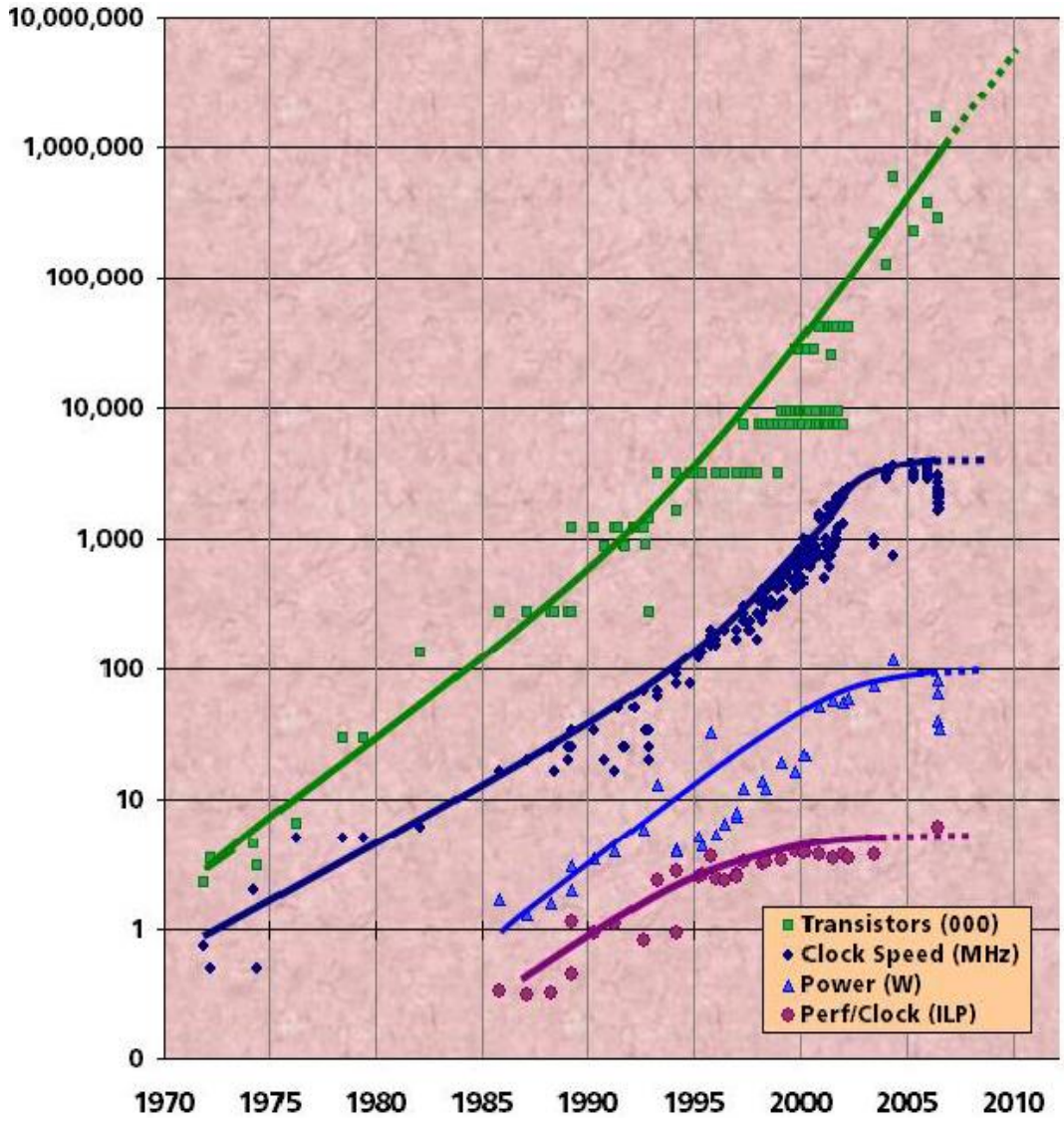
Moore's law continues:

- Chip density doubles every 1.5 years
- Most of it goes to doubling cores

Clock frequency does not increase

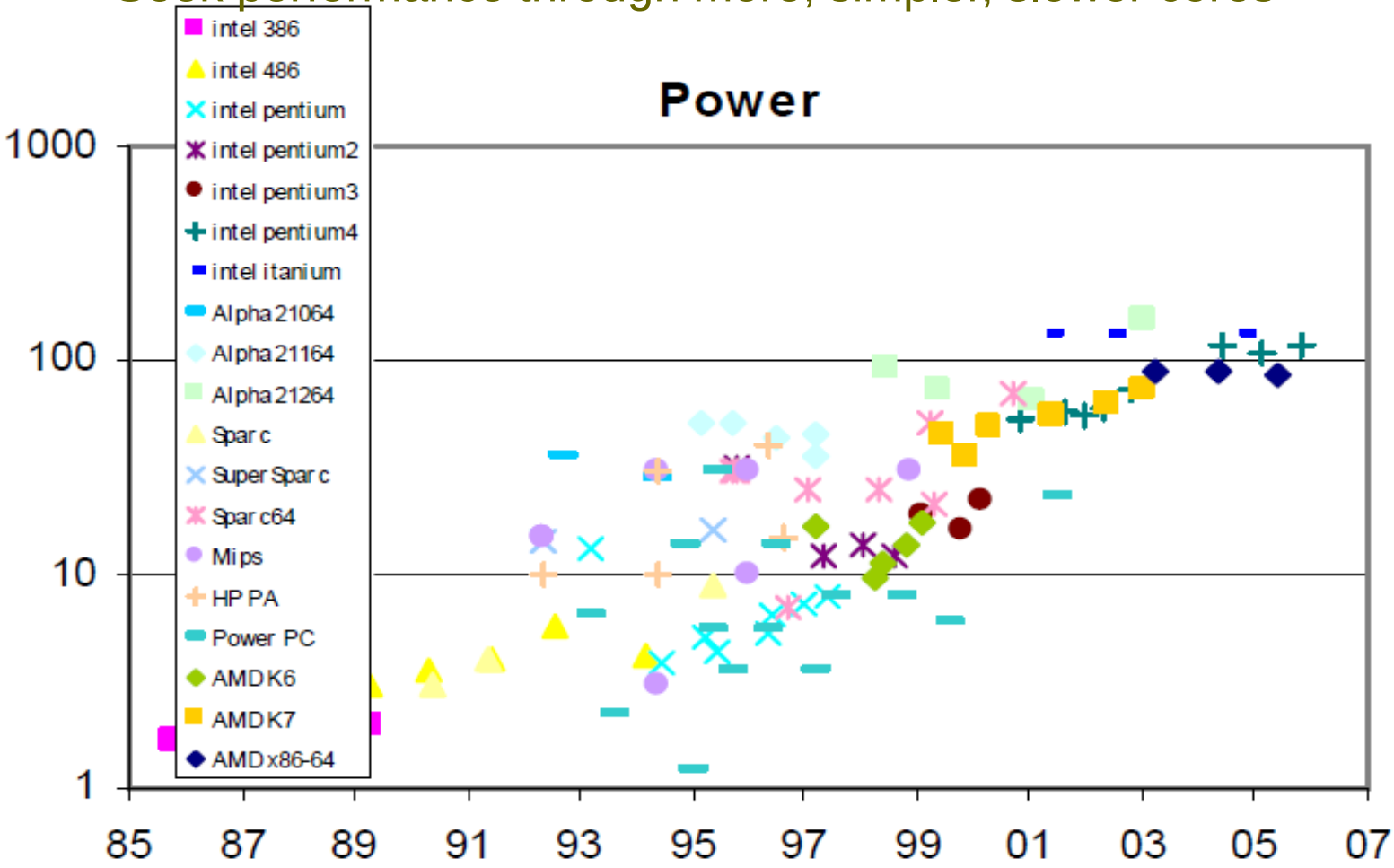
- Almost stagnant at less than 4GHz since 2004

Likewise performance from a single processor, due to lack of ILP

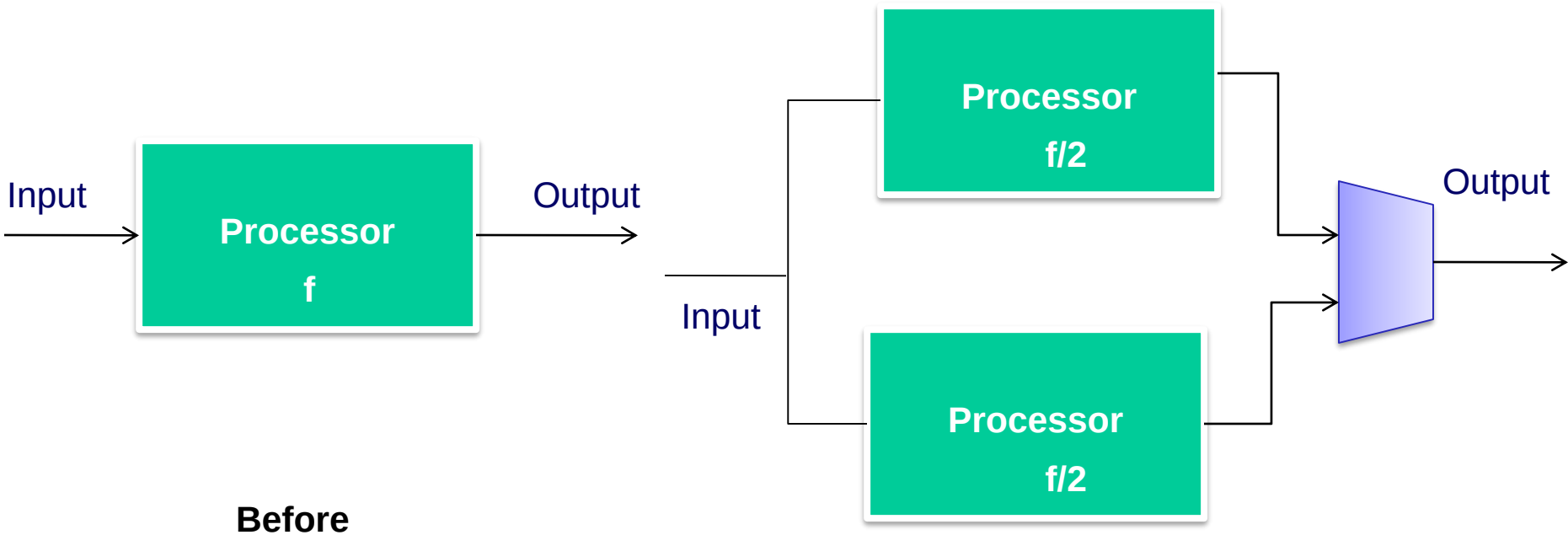


Technology drives new architectures

- » High power dissipation (CV^2f) drives lower clock frequency
- » Simpler, lower frequency unicore architecture
- » Seek performance through more, simpler, slower cores



How multiple cores reduce power



Before

Capacitance = C

Clock frequency = f

Voltage = V

Power = CV^2f

After

Capacitance = 2.2C

Clock frequency = f/2

Voltage = 0.6V

Power = $0.396CV^2f$

Slower processors allow for lower Vdd voltage.

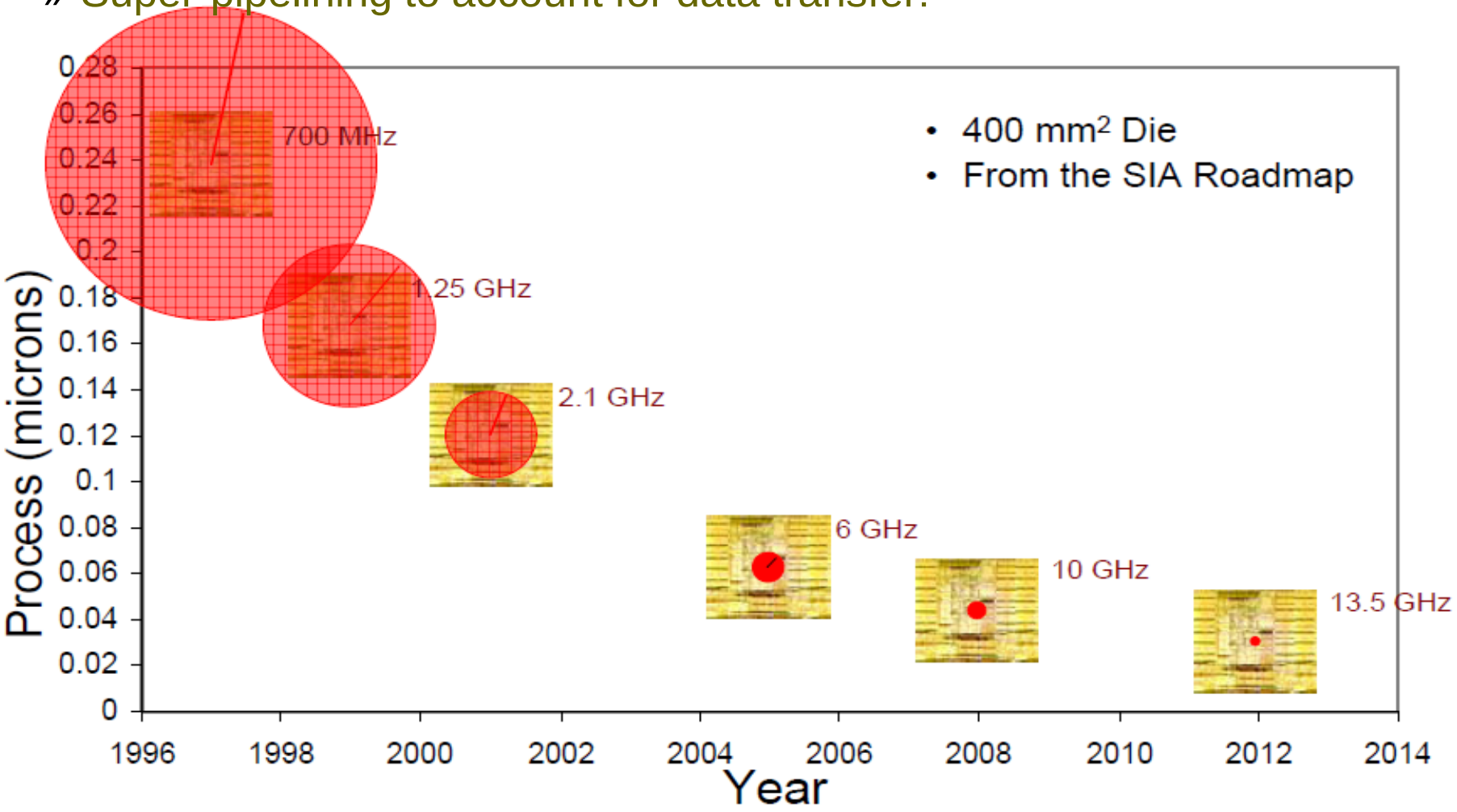
Emphasis on parallelism NOT on clock frequency

Technology drives new architectures

» Wire speed scales slower than transistor speed

» Wire delays drive localized computing == multi cores

» Super-pipelining to account for data transfer!



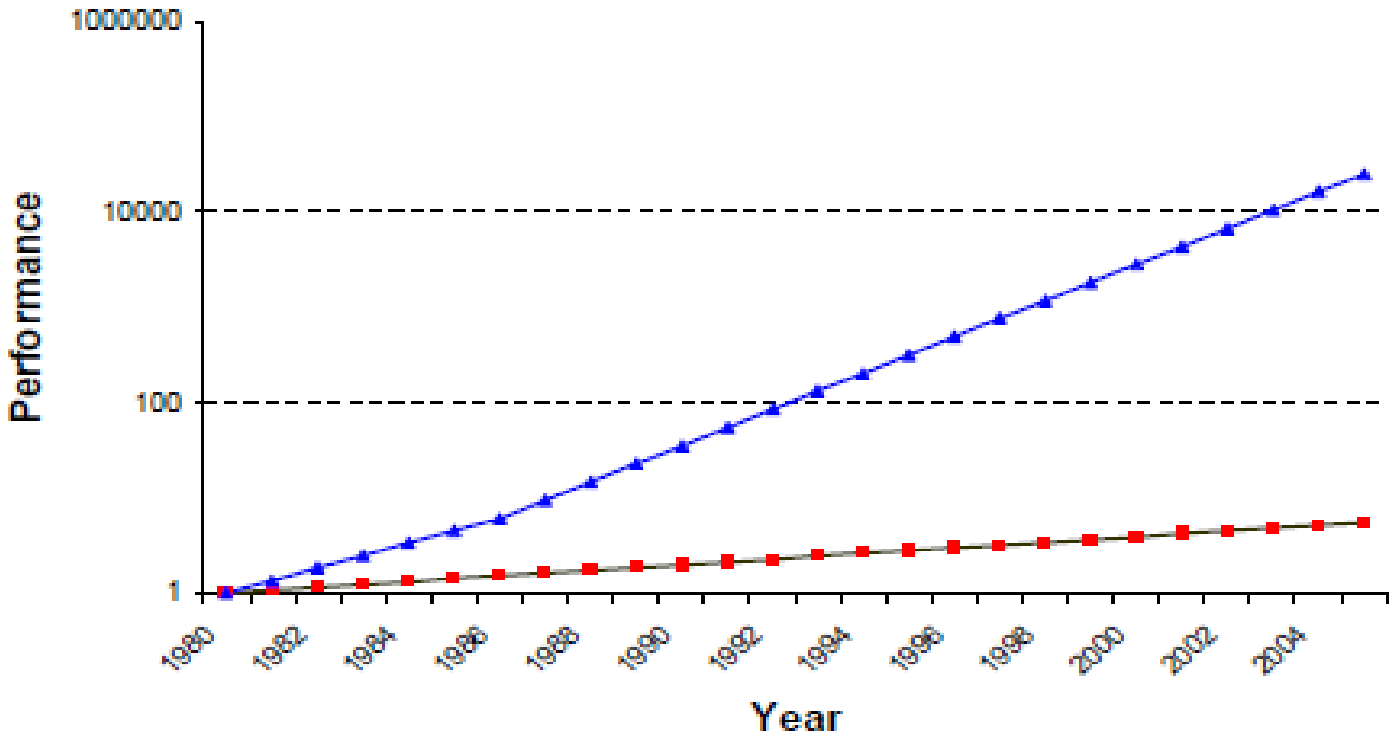
Technology drives new architectures

- **DRAM access latency**

External memory accesses becoming more and more expensive

In the order of hundreds of cycles for HighPerf processors

Need for caches or local memories

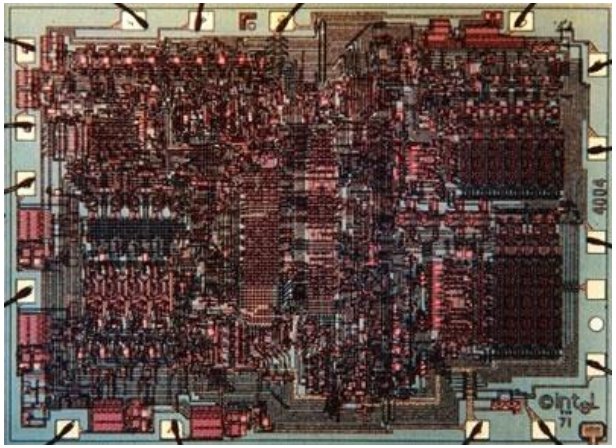


Technology drives new architectures

- Diminishing returns of instruction level parallelism
  - 50% performance improvement every year in the 80's
  - Due to pipeline : 5 CPI → 1 CPI
  - Diminishing returns in the 90's
  - More complexity to detect the last available ILP
  - Superscalar, VLIW, Branch prediction
  - Due to ILP : 1 CPI -> 0.3 CPI
  - The multicore era in the 00's

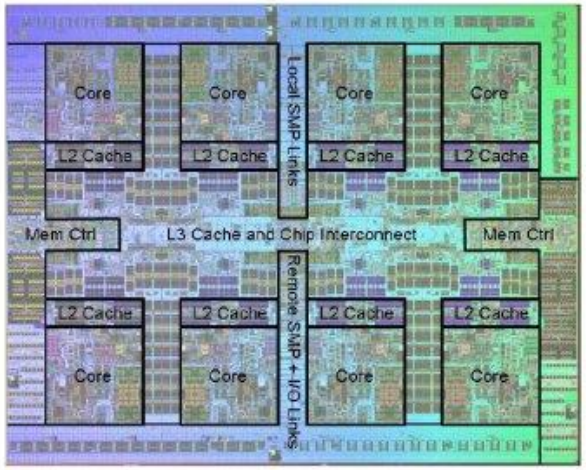
Tremendous change in Design Technology

- Intel 4004 (1971): 4-bit processor, 2250 transistors, 750 KHz, 10 micron PMOS, 11 mm^2 chip



- RISC II (1983): 32-bit, 5 stage pipeline, 40,760 transistors, 3 MHz, 3 micron NMOS, 60 mm^2 chip

- IBM Power 7 (2010): 8-core, 64-bit, 32 threads, 1.2 billion transistors, 0.045 micron CMOS technology



- State of art is 28nm (0.028 micron) in 2012

- **Processor is the new transistor?**

The End of the Uniprocessor Era

Maybe the biggest change of the computing systems ever

Déjà vu?

- Multiprocessors imminent in 1970s, ‘80s, ‘90s, …
- “… today’s processors … are nearing an impasse as technologies approach the speed of light..”

  David Mitchell, *The Transputer: The Time Is Now* (1989)
- Transputer was premature
  ⇒ Custom multiprocessors tried to beat uniprocessors
  ⇒ Procrastination rewarded: 2X seq. perf. / 1.5 years
- “We are dedicating all of our future product development to multicore designs. … This is a sea change in computing”

  Paul Otellini, President, Intel (2004)
- Difference is all microprocessor companies have switched to multiprocessors (AMD, Intel, IBM, Sun; all new Apples 2+ CPUs)
  ⇒ Procrastination penalized: 2X sequential perf. / 5 yrs
  ⇒ Biggest programming challenge: from 1 to 2 CPUs

Problems with Multicores

- Algorithms, Programming Languages, Compilers, Operating Systems, Architectures, Libraries, … not ready to supply Thread-Level Parallelism or Data-Level Parallelism for 1000 CPUs / chip

- Architectures not ready for 1000 CPUs / chip
  - Unlike Instruction-Level Parallelism, cannot be solved by computer architects and compiler writers alone, but also cannot be solved without participation of architects

- Inertia: engineers (esp. software engineers) unwilling to retrain to write parallel software

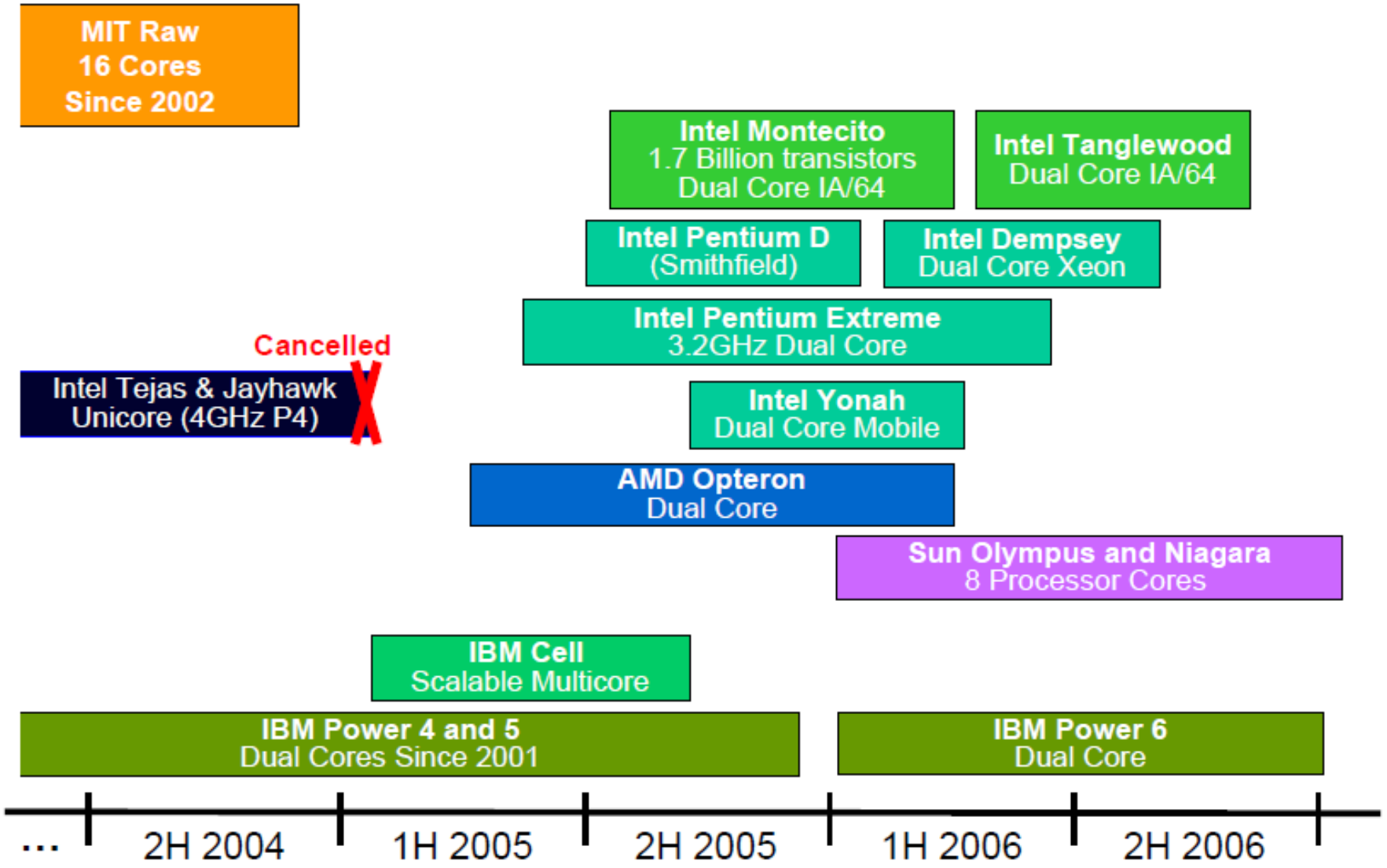
Conventional Wisdom in Computer Architecture

- Old Conventional Wisdom: Power is free, Transistors expensive
- New Conventional Wisdom: "Power wall" Power expensive, Transistors free (Can put more on chip than can afford to turn on)
- Old CW: Sufficient increasing Instruction-Level Parallelism via compilers, innovation (Out-of-order, speculation, VLIW, …)
- New CW: "ILP wall" law of diminishing returns on more HW for ILP
- Old CW: Multiplies are slow, Memory access is fast
- New CW: "Memory wall" Memory slow, multiplies fast (200 clock cycles to DRAM memory, 4 clocks for multiply)
- Old CW: Uniprocessor performance 2X / 1.5 yrs (Moore's law)
- New CW: Power Wall + ILP Wall + Memory Wall = Brick Wall
  - Uniprocessor performance now 2X / 5(?) yrs

⇒ Sea change in chip design: multiple "cores" (2X processors per chip / ~ 2 years)

» More, simpler processors are more power efficient

Uniprocessor systems are extinct!



Multicores Are Here!