EE204
Computer Architecture
Lecture 1: Introduction and Basics

Instructor: Dr. Hassan Jamil Syed

Fall 2019

Computer Architecture

Computer Architecture is … the ***science*** and ***art*** of selecting (or designing) and interconnecting hardware and software components to ***create computers*** …

- Computer Architecture is an umbrella term

– *Architecture*: software-visible interface

– *Micro-architecture*: internal organization of components

- This course is mostly about ***micro-architecture***

– What's inside the processor (CPU)

– What implications this has on software

Introduction

- Background: EE 213 or equivalent, based on Hennessy and Patterson’s Computer Organization and Design
- Text for EE 204: Hennessy and Patterson’s Computer Architecture, A Quantitative Approach, 5th Edition
- Topics
  - Measuring performance/cost/power
  - Instruction level parallelism, dynamic and static
  - Memory hierarchy
  - Multiprocessors
  - Storage systems and networks

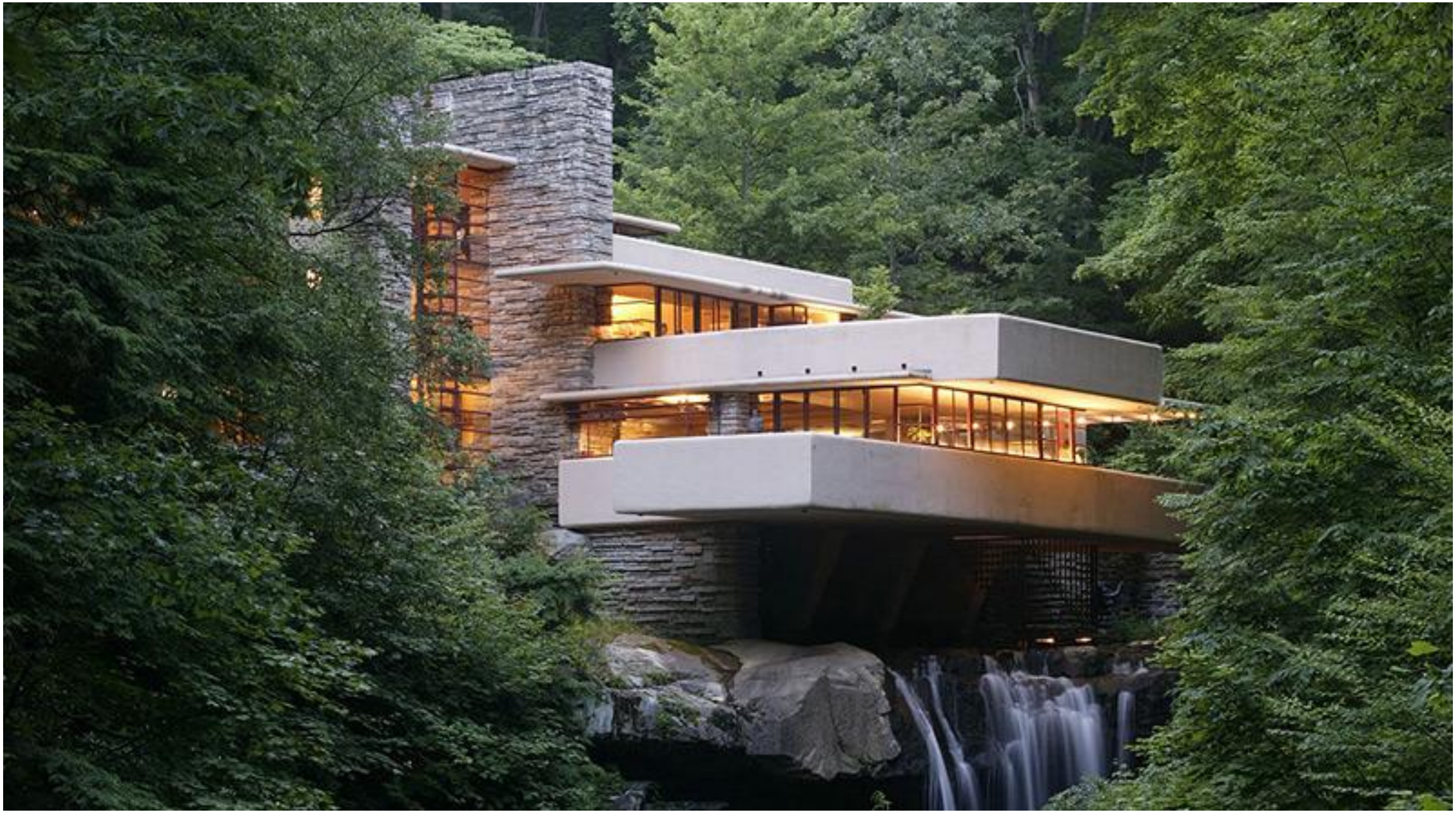
Grading Criteria and Class Discipline Points

Assessment Weightages:

- 3 Quizzes (2 + 2 + 4 Marks)
  - No Retake of class quizzes. Retake only allowed as per university policy
- 3 Assignments (2 + 2 + 3 Marks)
  - No Assignment Submission after Due Date.
  - Submissions are also not accepted through email.
- 1 Project (5 Marks)
- Sessional I , Sessional II (15 +15 Marks)
- Final (50 Marks)

- Plagiarism will be marked as Zero.
- Passing Marks = 50
- Class Attendance should be >= 80.
- No Student will be allowed after 10 Mins starting of the class.

Question: What Is This?



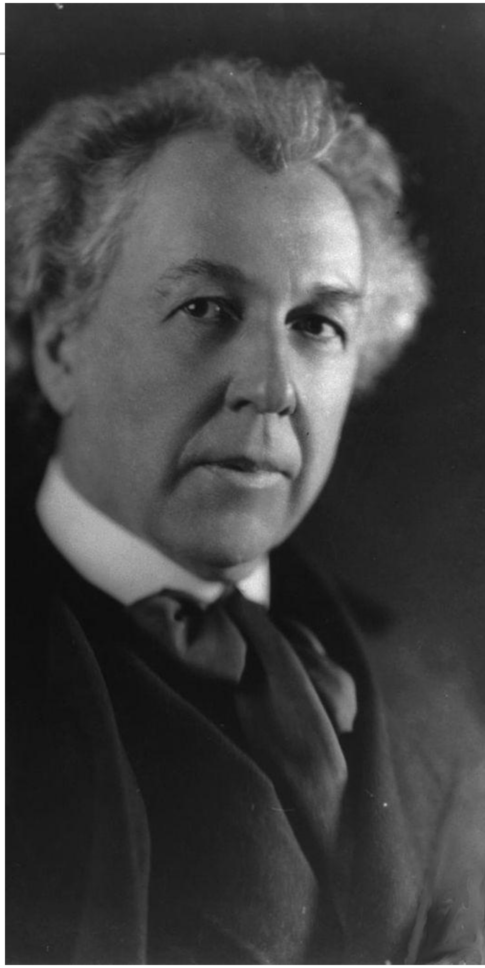
Answer: Masterpiece of A Famous Architect

Fallingwater

From Wikipedia, the free encyclopedia

Fallingwater or **Kaufmann Residence** is a house designed by architect Frank Lloyd Wright in 1935 in rural southwestern Pennsylvania, 43 miles (69 km) southeast of Pittsburgh.[4] The home was built partly over a waterfall on Bear Run in the Mill Run section of Stewart Township, Fayette County, Pennsylvania, in the Laurel Highlands of the Allegheny Mountains.

Time cited it after its completion as Wright's "most beautiful job";[5] it is listed among *Smithsonian*'s Life List of 28 places "to visit before you die."[6] It was designated a National Historic Landmark in 1966.[3] In 1991, members of the American Institute of Architects named the house the "best all-time work of American architecture" and in 2007, it was ranked twenty-ninth on the list of America's Favorite Architecture according to the AIA.

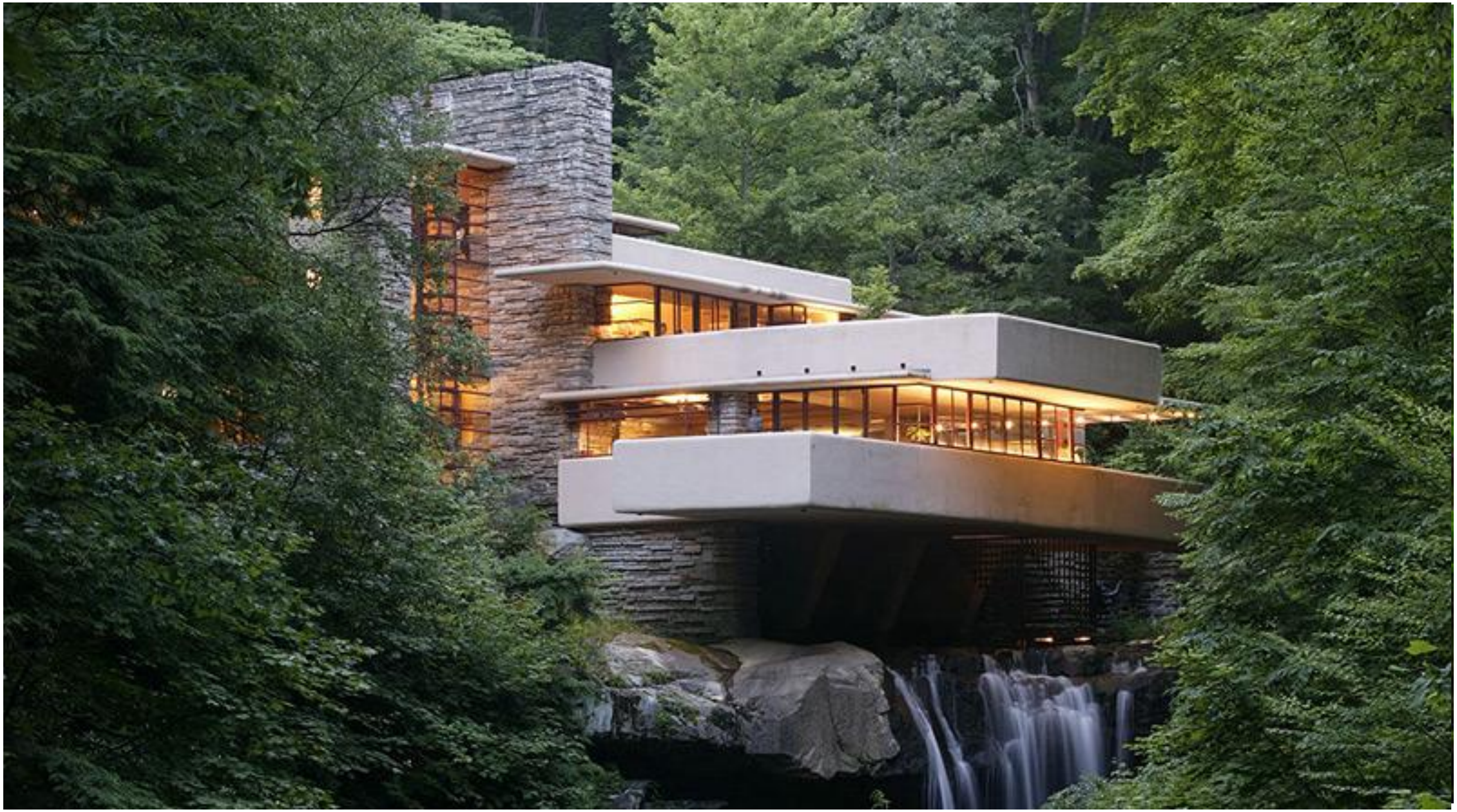


Your First Assignment

- Appreciate the importance of out-of-the-box and creative thinking
- Think about tradeoffs in the design of the building
  - Strengths, weaknesses
- Derive principles on your own for good design and innovation

- Due date: **After passing this course**
  - Apply what you have learned in this course
  - Think out-of-the-box

Find Differences Of This and That



A Key Question

- How Was Wright Able To Design Fallingwater?
- Can have many guesses
  - (Ultra) hard work, perseverance, dedication (over decades)
  - Experience of decades
  - Creativity
  - Out-of-the-box thinking
  - Principled design
  - A good understanding of past designs
  - Good judgment and intuition
  - Strong combination of skills (math, architecture, art, …)
  - …
- (You will be exposed to and hopefully develop/enhance many of these skills in this course)

A Quote from The Architect Himself

- “architecture […] based upon principle, and not upon precedent”

Major High-Level Goals of This Course

- Understand the principles
- Understand the precedents

- Based on such understanding:
  - Enable you to evaluate tradeoffs of different designs and ideas
  - Enable you to develop principled designs
  - Enable you to develop novel, out-of-the-box designs

- The focus is on:
  - Principles, precedents, and how to use them for new designs

Computer Architecture

Computer Architecture is … the ***science*** and ***art*** of selecting (or designing) and interconnecting hardware and software components to ***create computers*** …

- Computer Architecture is an umbrella term
  - *Architecture*: software-visible interface
  - *Micro-architecture*: internal organization of components
- This course is mostly about ***micro-architecture***
  - What's inside the processor (CPU)
  - What implications this has on software

Computer Architecture:

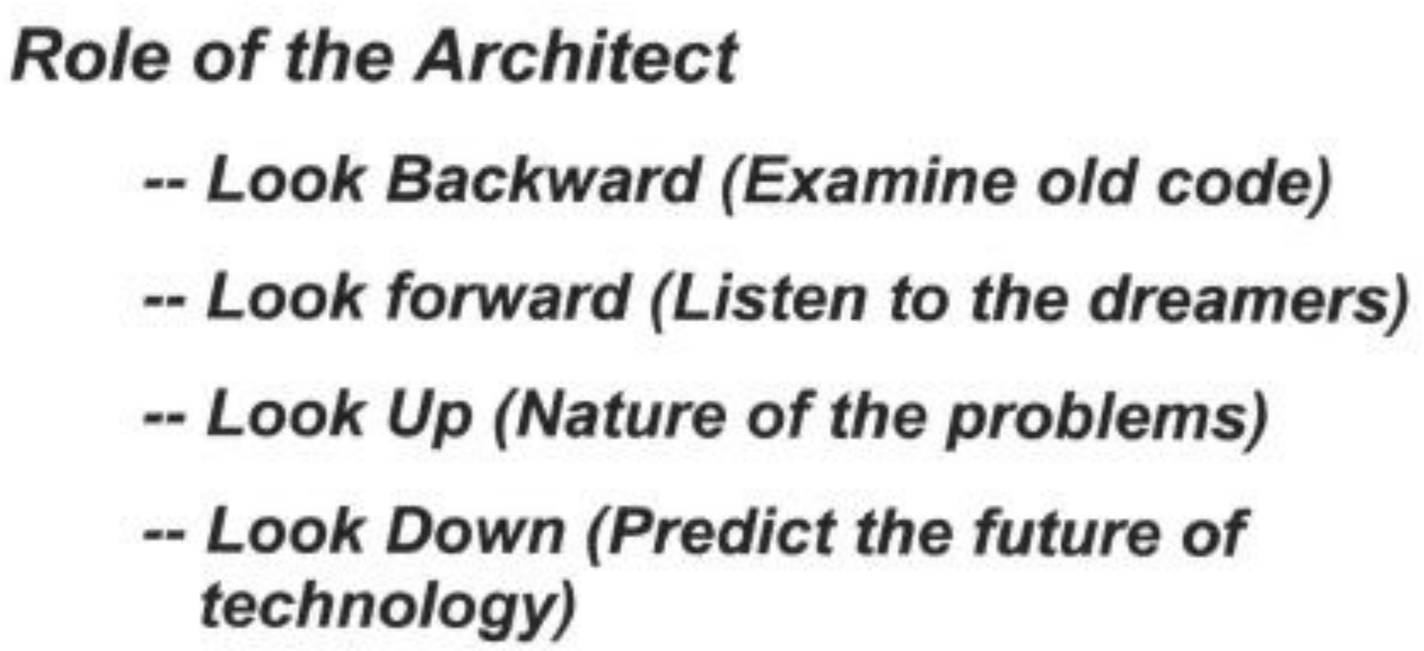
1. The science and art of designing, selecting, and interconnecting hardware components and designing the hardware/software interface to create a computing system that meets functional, performance, energy consumption, cost, and other specific goals.

2. The attributes of a [computing] system as seen by the programmer, i.e., the conceptual structure and functional behavior, as distinct from the organization of the data flows and controls, the logic design, and the physical implementation.

Amdahl, Blaaw, and Brooks, 1964

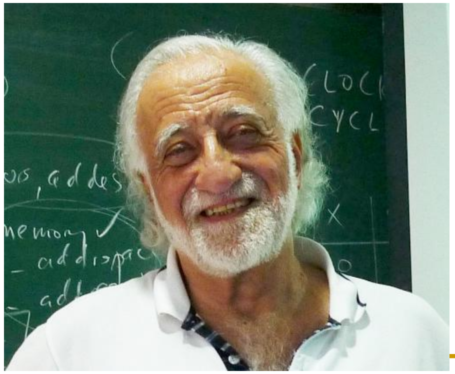
Computer architecture covers the specification of an instruction set and the hardware units that implement the instruction set.

Role of the (Computer) Architect



Role of the Architect

- -- Look Backward (Examine old code)
- -- Look forward (Listen to the dreamers)
- -- Look Up (Nature of the problems)
- -- Look Down (Predict the future of technology)



from Yale Patt’s lecture notes

Role of The (Computer) Architect

- Look backward (to the past)
  - Understand tradeoffs and designs, upsides/downsides, past workloads. Analyze and evaluate the past.
- Look forward (to the future)
  - Be the dreamer and create new designs. Listen to dreamers.
  - Push the state of the art. Evaluate new design choices.
- Look up (towards problems in the computing stack)
  - Understand important problems and their nature.
  - Develop architectures and ideas to solve important problems.
- Look down (towards device/circuit technology)
  - Understand the capabilities of the underlying technology.
  - Predict and adapt to the future of technology (you are designing for N years ahead). Enable the future technology.

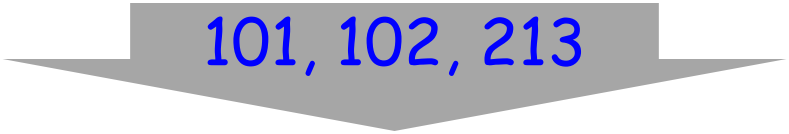
Takeaways

- Being an architect is not easy
- You need to consider **many** things in designing a new system + have good intuition/insight into ideas/tradeoffs

- But, it is fun and can be very technically rewarding
- And, enables a great future
  - E.g., many scientific and everyday-life innovations would not have been possible without architectural innovation that enabled very high performance systems
  - E.g., your mobile phones

- This course will teach you how to become a good computer architect

So, I Hope You Are Here for This



101, 102, 213

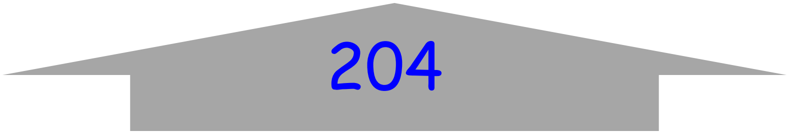
"C/Java" as a model of computation

Programmer's view of how
a computer system works

- How does an assembly program end up executing as digital logic?
- What happens in-between?
- How is a computer designed using logic gates and wires to satisfy specific goals?

Architect/microarchitect's view:
How to design a computer that
meets system design goals.
Choices critically affect both
the SW programmer and
the HW designer

HW designer's view of how
a computer system works

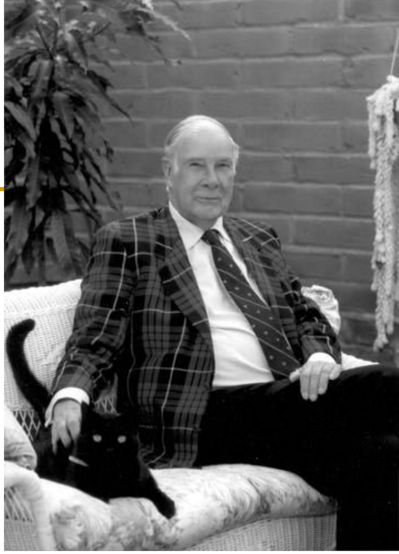


204

Digital logic as a
model of computation

Levels of Transformation

“The purpose of computing is insight” (*Richard Hamming*)
We gain and generate insight by solving problems
How do we ensure problems are solved by electrons?

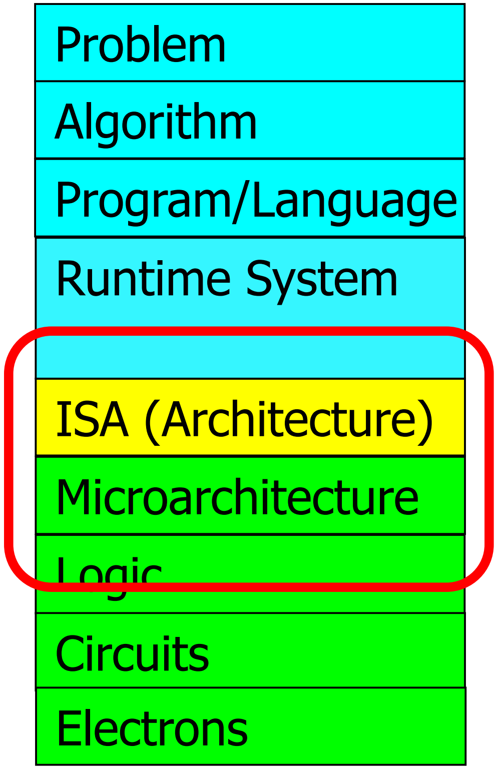


Program gets translated to instructions that will be executed by the microarchitecture.

Microarchitecture is implemented in logic gates.

Logic gates in terms are implemented in circuits and

Circuit operates based on the Physics principles using moving electrons.



Problem

Algorithm

Program/Language

Runtime System

ISA (Architecture)

Microarchitecture

Logic

Circuits

Electrons

A **runtime system** is a collection of software and hardware resources that enable a software program to be executed on a computer *system*

ISA is the interface between hardware and software… It is a contract that the hardware promises to satisfy.

Computer architecture covers the specification of an instruction set and the hardware units that implement the instruction set.

The Power of Abstraction

- Levels of transformation create abstractions
  - Abstraction: A higher level only needs to know about the interface to the lower level, not how the lower level is implemented
  - E.g., high-level language programmer does not really need to know what the ISA is and how a computer executes instructions

- Abstraction improves productivity
  - No need to worry about decisions made in underlying levels
  - E.g., programming in Java vs. C vs. assembly vs. binary vs. by specifying control signals of each transistor in every cycle

- Then, why would you want to know what goes on underneath or above?

Crossing the Abstraction Layers

- As long as everything goes well, not knowing what happens in the underlying level (or above) is not a problem.

- What if
  - The program you wrote is running slow?
  - The program you wrote does not run correctly?
  - The program you wrote consumes too much energy?

- What if
  - The hardware you designed is too hard to program?
  - The hardware you designed is too slow because it does not provide the right primitives to the software?

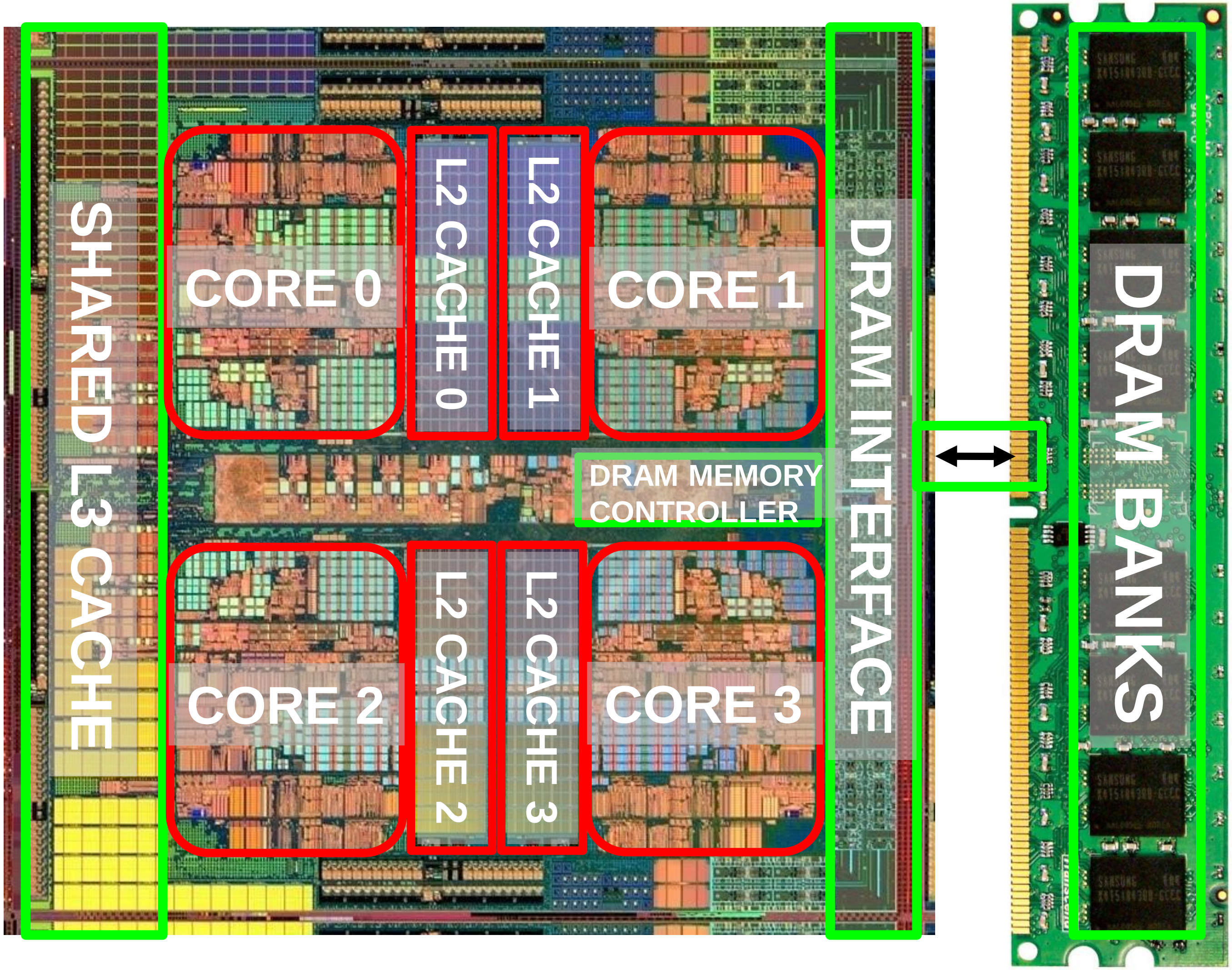
- What if
  - You want to design a much more efficient and higher performance system?

Crossing the Abstraction Layers

- Two key goals of this course are
  - to understand how a processor works underneath the software layer and how decisions made in hardware affect the software/programmer
  - to enable you to be comfortable in making design and optimization decisions that cross the boundaries of different layers and system components

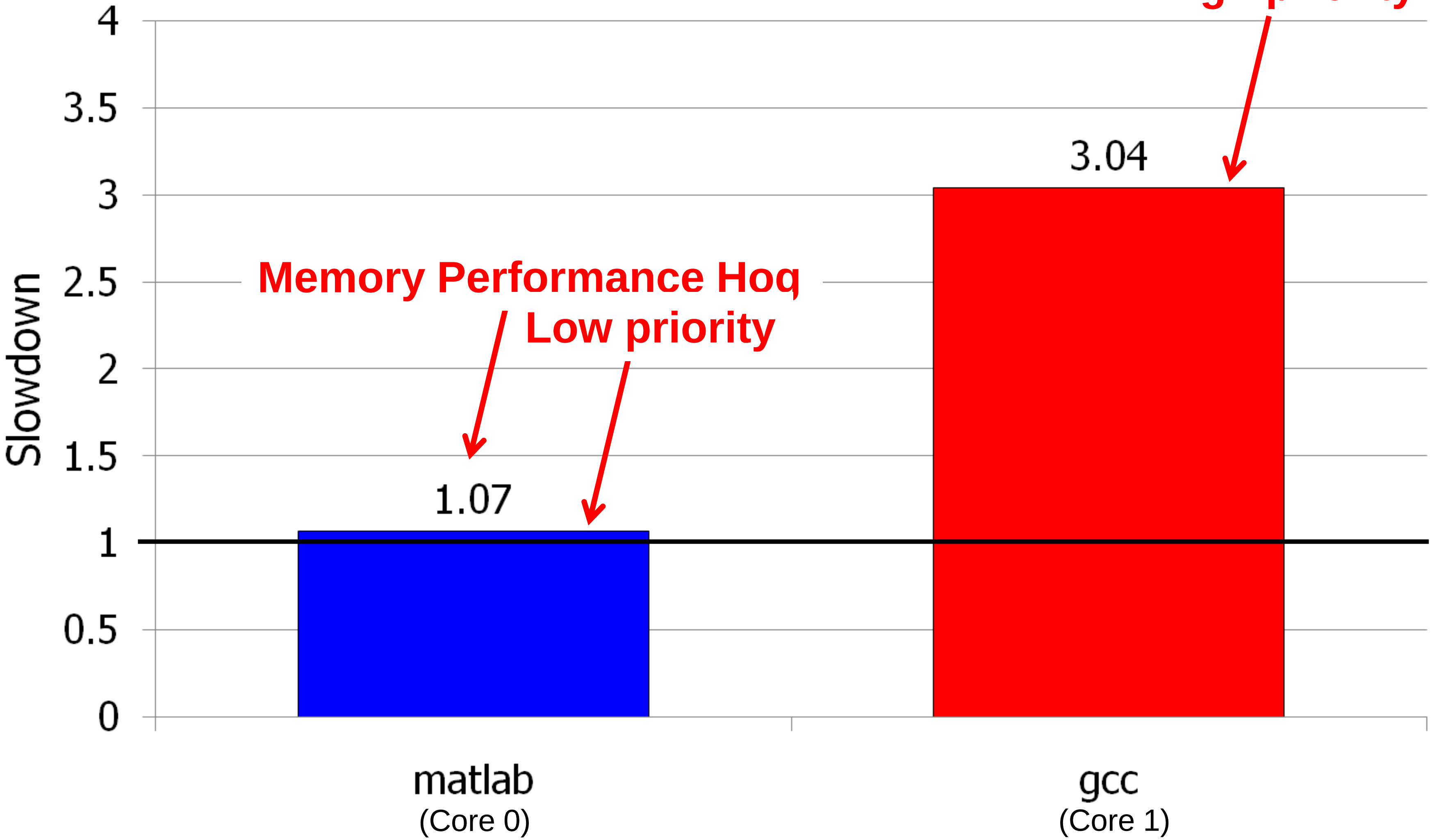
An Example: Multi-Core Systems

Multi-Core Chip



*Die photo credit: AMD Barcelona

Unexpected Slowdowns in Multi-Core



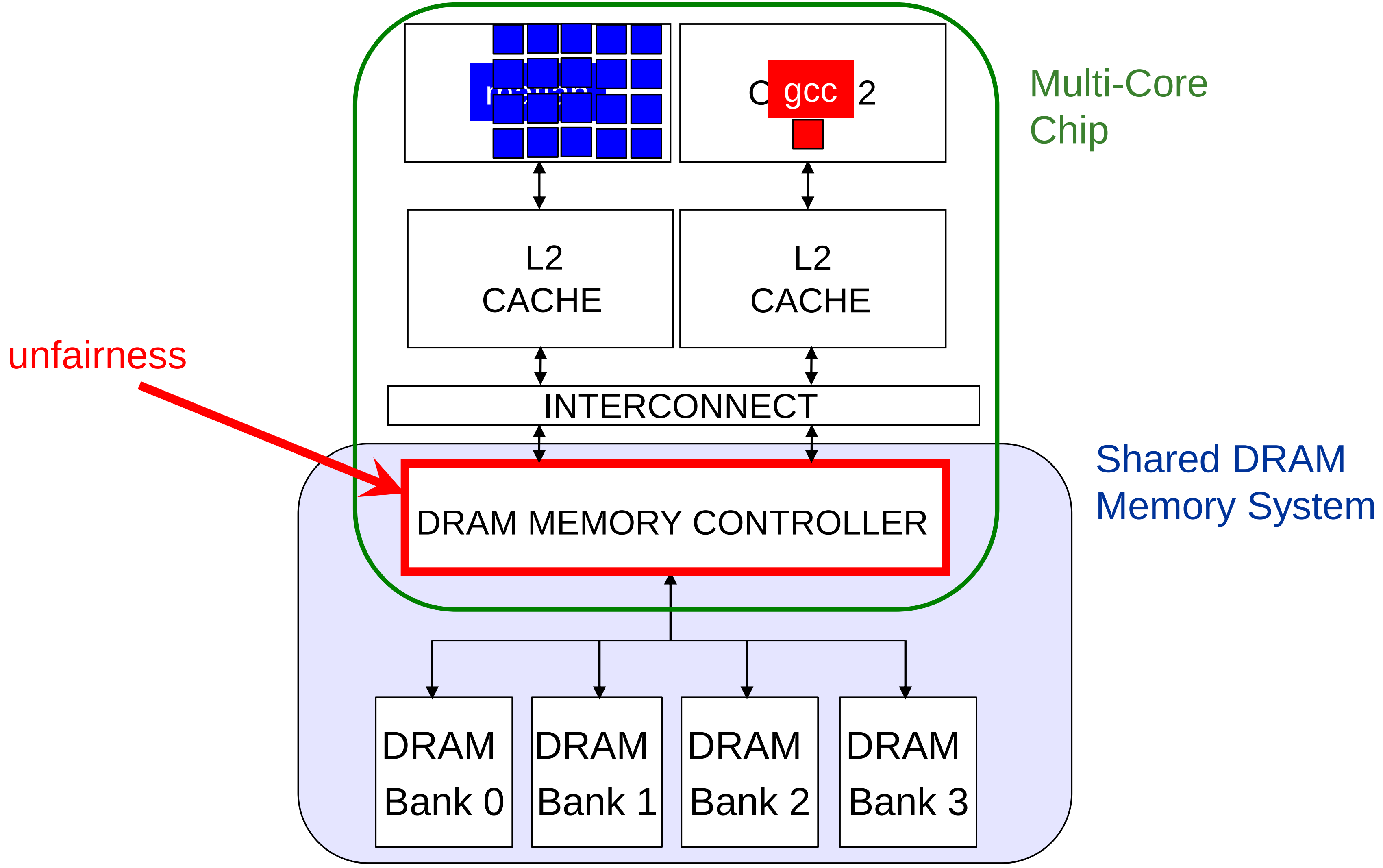
Moscibroda and Mutlu, "Memory performance attacks: Denial of memory service in multi-core systems," USENIX Security 2007.

A Question or Two

- Can you figure out why there is a disparity in slowdowns if you do not know how the system executes the programs?

- Can you fix the problem without knowing what is happening “underneath”?

Why the Disparity in Slowdowns?



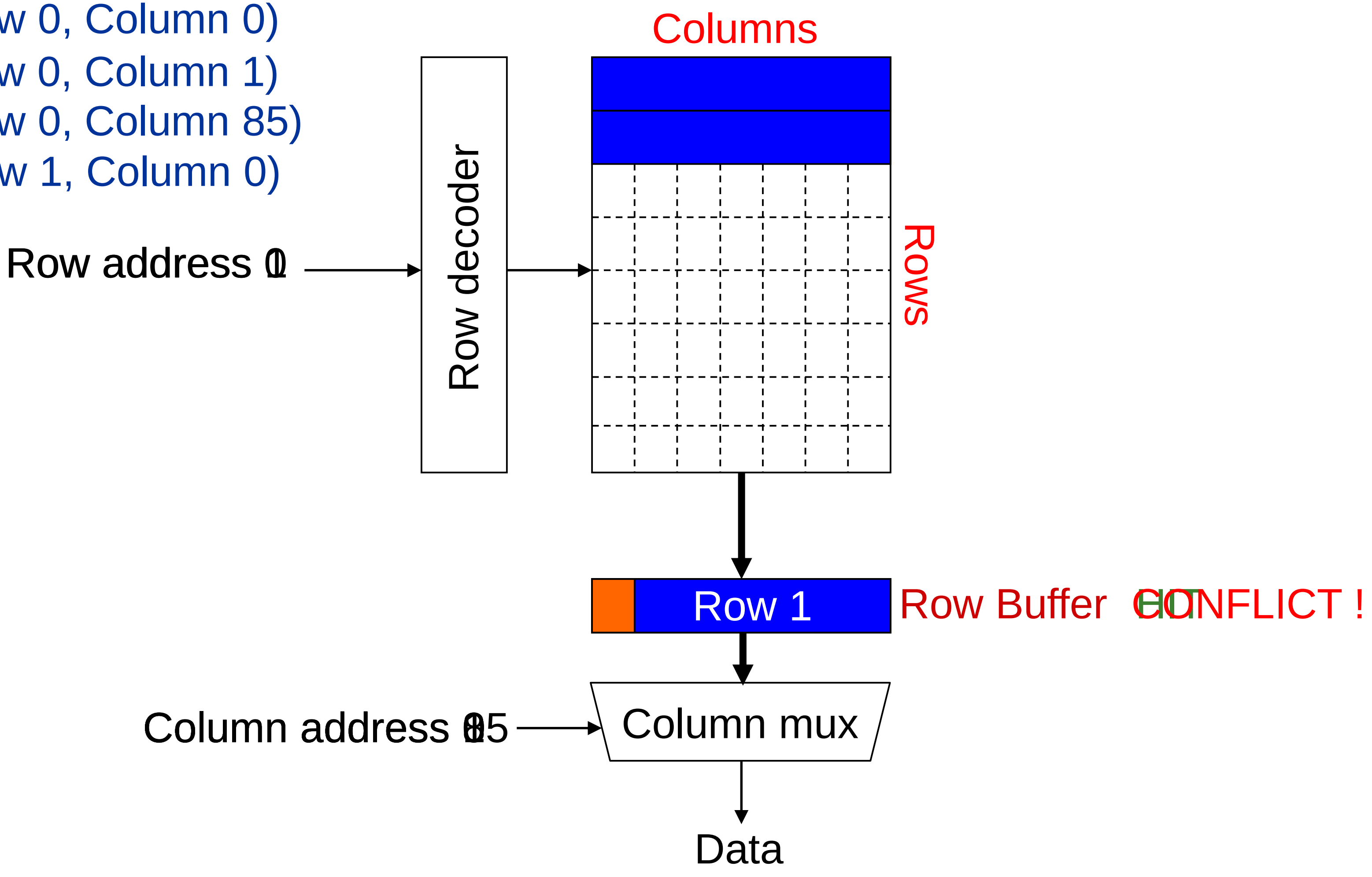
Multi-Core Chip

Shared DRAM Memory System

unfairness

DRAM Bank Operation

Access Address:
(Row 0, Column 0)
(Row 0, Column 1)
(Row 0, Column 85)
(Row 1, Column 0)



DRAM Controllers

- A row-conflict memory access takes significantly longer than a row-hit access

- Current controllers take advantage of the row buffer

- Commonly used scheduling policy (FR-FCFS) [Rixner 2000]*

  (1) Row-hit first: Service row-hit memory accesses first

  (2) Oldest-first: Then service older accesses first

- This scheduling policy aims to maximize DRAM throughput

*Rixner et al., "Memory Access Scheduling," ISCA 2000.
*Zuravleff and Robinson, "Controller for a synchronous DRAM …," US Patent 5,630,096, May 1997.

The Problem

- Multiple applications share the DRAM controller
- DRAM controllers designed to maximize DRAM data throughput

- DRAM scheduling policies are unfair to some applications
  - Row-hit first: unfairly prioritizes apps with high row buffer locality
    - Threads that keep on accessing the same row
  - Oldest-first: unfairly prioritizes memory-intensive applications

- DRAM controller vulnerable to denial of service attacks
  - Can write programs to exploit unfairness

A Memory Performance Hog

```
// initialize large arrays A, B

for (j=0; j<N; j++) {
    index = j*linesize;   streaming
    A[index] = B[index];
    …
}
```

```
// initialize large arrays A, B

for (j=0; j<N; j++) {
    index = rand();   random
    A[index] = B[index];
    …
}
```

STREAM

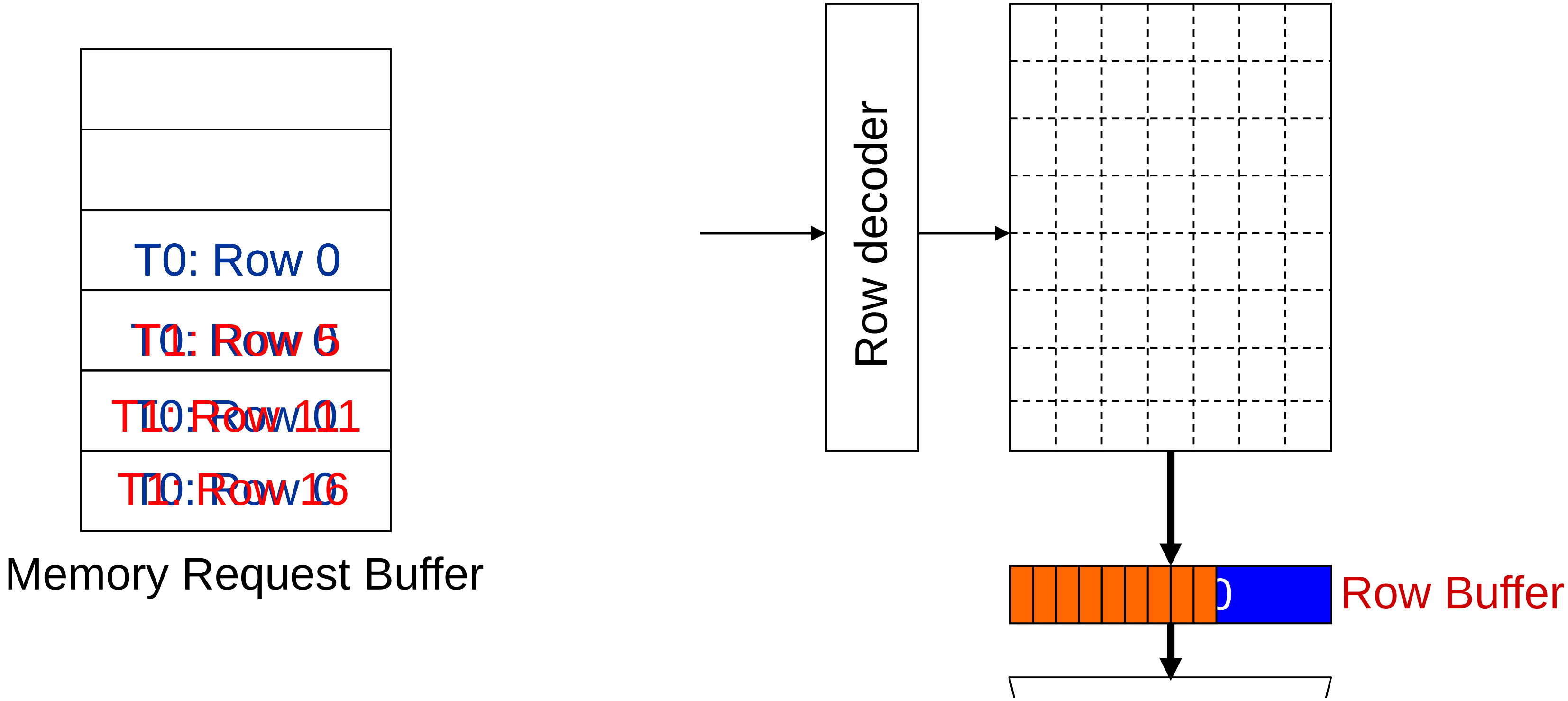
- Sequential memory access
- Very high row buffer locality (96% hit rate)
- Memory intensive

RANDOM

- Random memory access
- Very low row buffer locality (3% hit rate)
- Similarly memory intensive

Moscibroda and Mutlu, "Memory Performance Attacks," USENIX Security 2007.

What Does the Memory Hog Do?



Row size: 8KB, cache block size: 64B

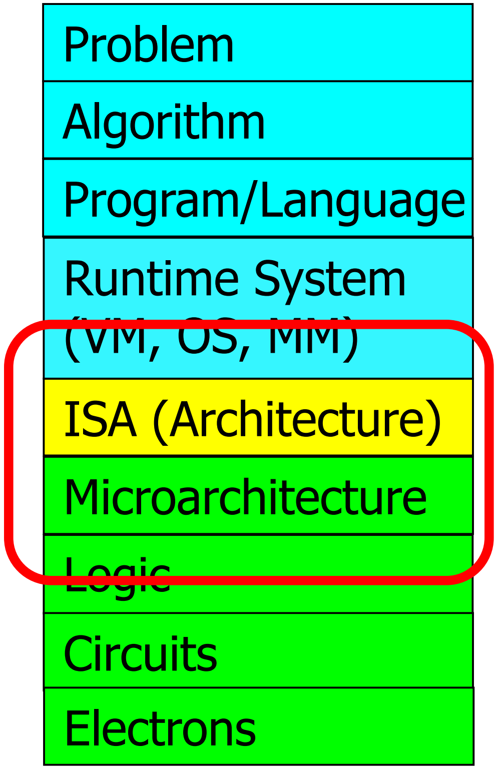
128 (8KB/64B) requests of T0 serviced before T1

Moscibroda and Mutlu, "Memory Performance Attacks," USENIX Security 2007.

Now That We Know What Happens Underneath

- How would you solve the problem?

- What is the right place to solve the problem?
  - Programmer?
  - System software?
  - Compiler?
  - Hardware (Memory controller)?
  - Hardware (DRAM)?
  - Circuits?



Problem

Algorithm

Program/Language

Runtime System (VM, OS, MM)

ISA (Architecture)

Microarchitecture

Logic

Circuits

Electrons

- Two other goals of this course:
  - Enable you to think critically
  - Enable you to think broadly

Introduction

- RISC (Reduced Instruction Set Computer) processors have focused on two critical performance improvement techniques
  1. Exploit Instruction-level parallelism
     Pipelining and superscalar execution
  2. Use of caches
- **Effects of large technological growth**
- Significant enhancement in the capability available to computer users
- New classes of computers have emerged
- Microprocessor-based computers are dominating the entire range of computer design
- The developments have a major impact on software development

Introduction

- Performance is traded with productivity
- Strategy for software development has also changed

  Software as a service (Saas) is used instead of the conventional shrink-wrapped software
- Nature of applications have also undergone a change
- Shift from ILP (Instruction-Level Parallelism) to DLP (Data-Level Parallelism), TLP (Thread-Level Parallelism) and RLP (Request-Level Parallelism)
- ILP has been implicit but TLP, DLP and RLP are explicit parallelism

  The major burden for exploitation of this shift is on the programmers

Classes of Parallelism and Parallel Architectures

- Classes of Parallelism and Parallel Architectures

  Parallelism is present at multiple levels

- Parallelism in application

  DLP: Many data items can be operated on at the same time

  TLP: Tasks of work are created that can operate in parallel

- Computer hardware can exploit these parallelism in four ways

Classes of Parallelism

1. **Instruction-Level Parallelism**

   Exploits data-level parallelism

   Pipelining and speculative execution

2. **Vector Architectures and Graphic Processor Units**

   Exploits DLP by applying a single instruction on a collection of data in parallel

3. **Thread-Level Parallelism**

   Exploits DLP or TLP in a tightly coupled hardware model that allows interaction among parallel threads

4. **Request-Level Parallelism**

   Exploits parallelism among largely decoupled tasks specified by the programmer or the OS

   - Set of requests which we are going to run in parallel.

Why we need a Performance Metrics?

- To understand:
  - why one instruction set can be implemented to perform better than another,
  - or how some hardware feature affects performance.

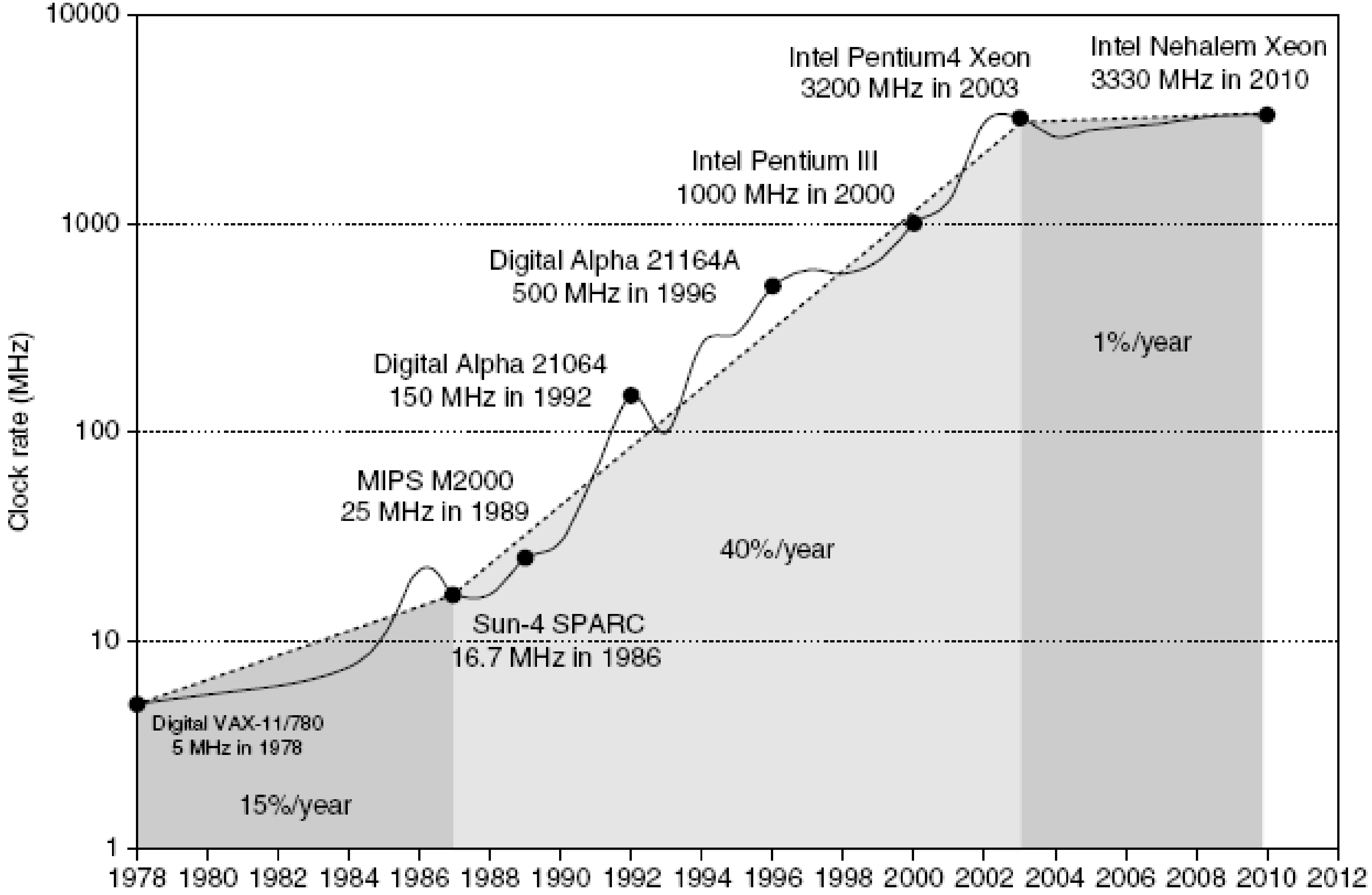
Ref. Ch # 4 - Computer Organization Design by Patterson and Hennessey

Why we need a Performance Metrics?

- To understand why a piece of software performs as it does, why one instruction set can be implemented to perform better than another, or how some hardware feature affects performance, we need to understand what determines the performance of a computer.

Ref. Ch # 4 - Computer Organization Design by Patterson and Hennessey

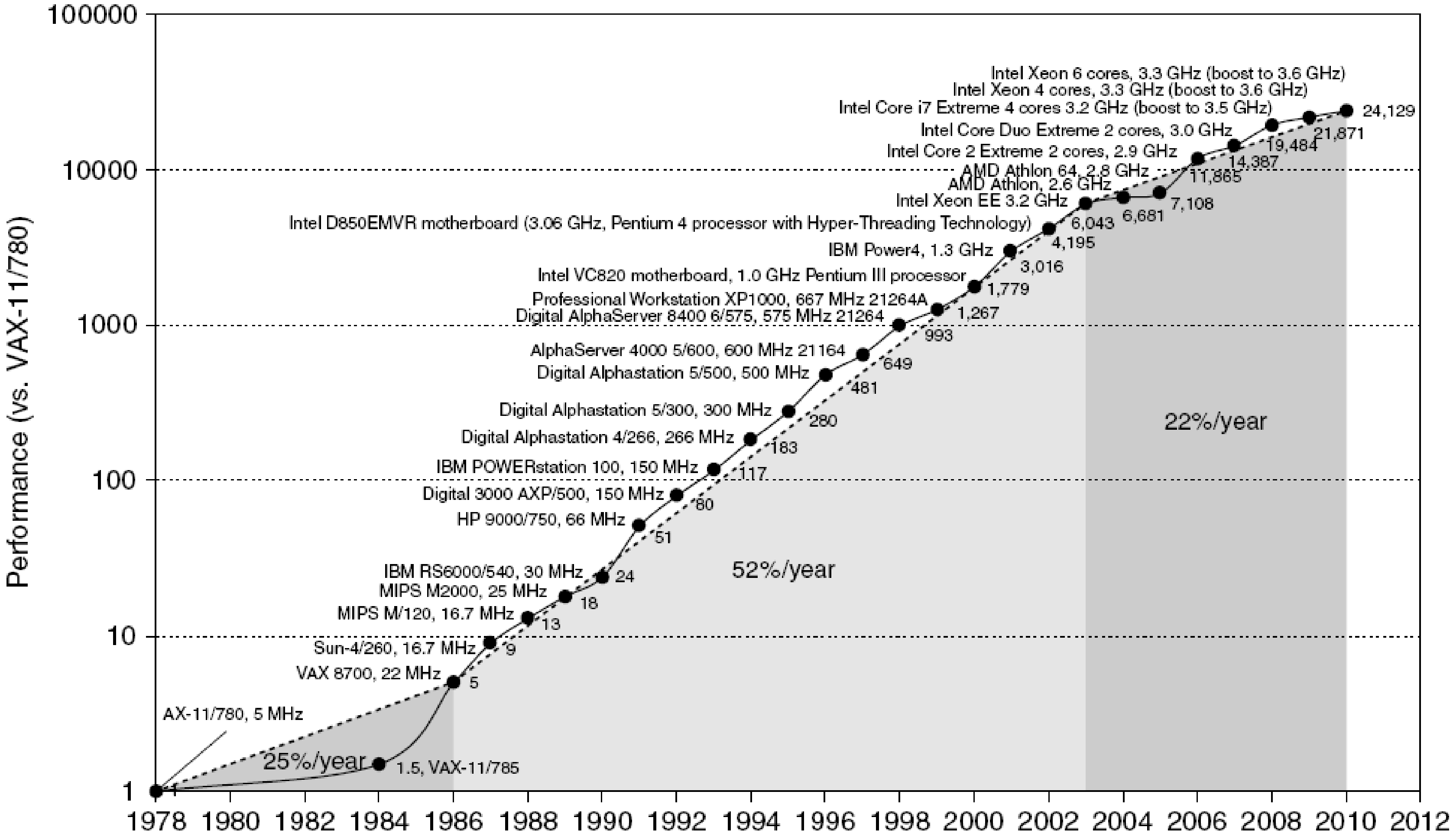
Clock Speed Increases



Source: H&P textbook

Source: H&P textbook

Historical Microprocessor Performance



The 52% growth per year is because of faster clock speeds and architectural innovations

The 22% growth includes the parallelization from multiple cores

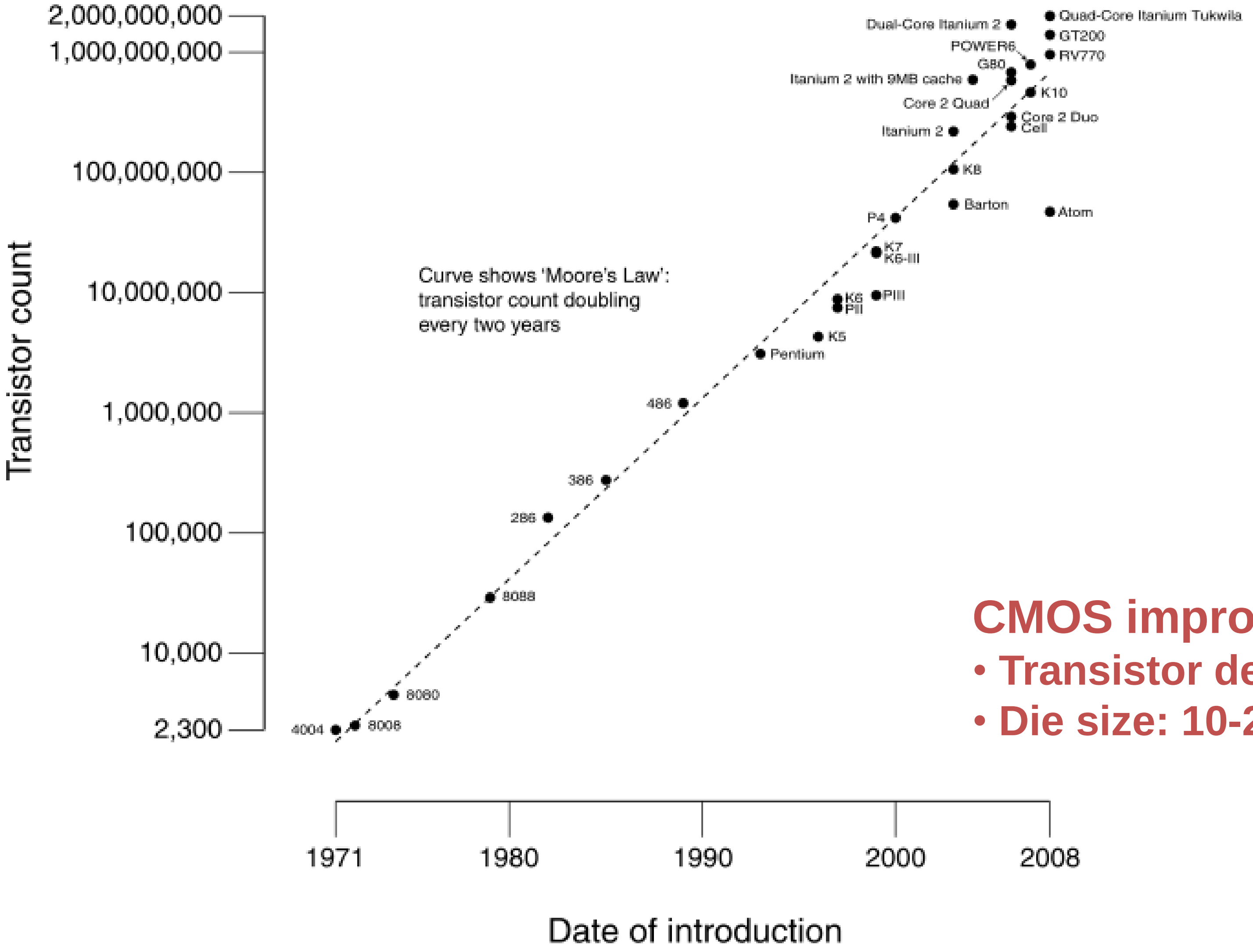
Points to Note

- The 52% growth per year is because of faster clock speeds and architectural innovations

- Clock speed increases have dropped to 1% per year in recent years

- The 22% growth includes the parallelization from multiple cores

CPU Transistor Counts 1971-2008 & Moore's Law



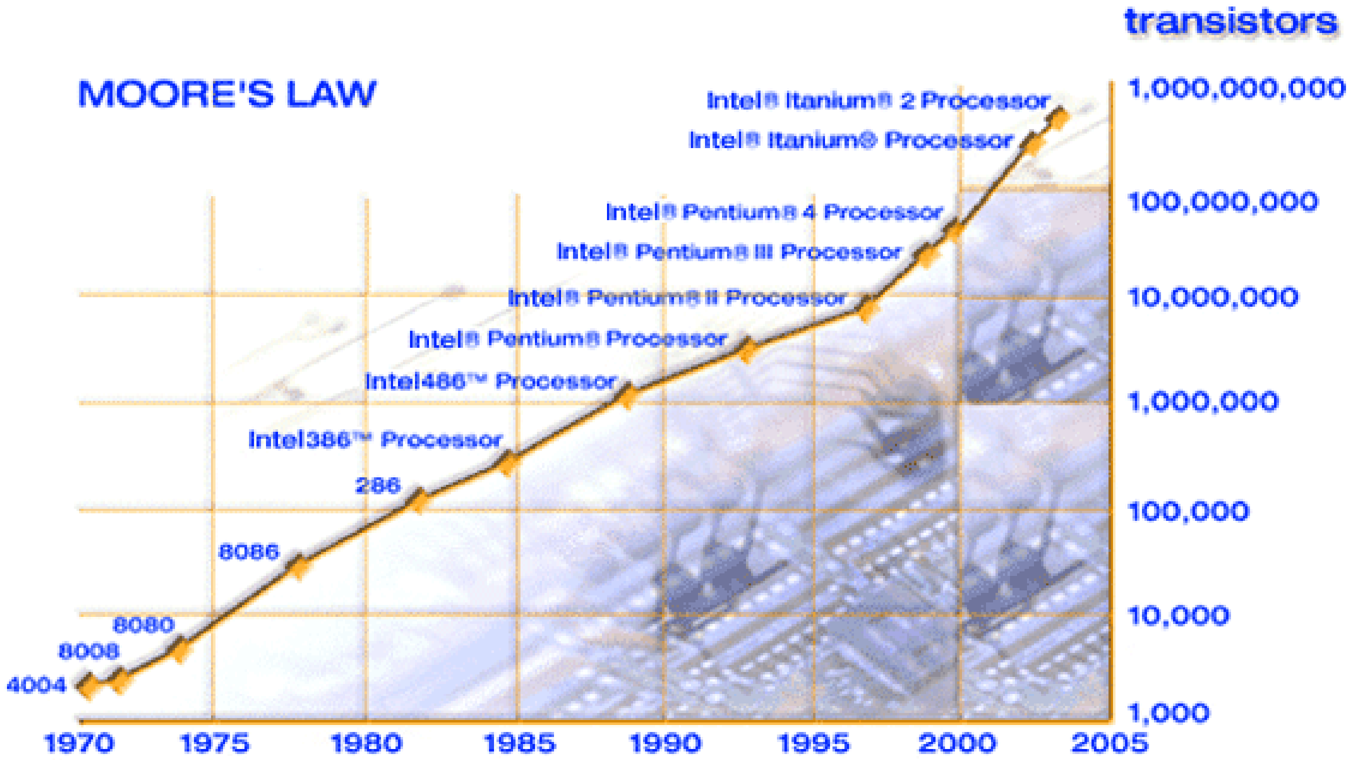
CMOS improvements:

- **Transistor density: 4x / 3 yrs**
- **Die size: 10-20% / yr**

Moore's Law: Transistors on a chip double every 24 months

Technology Trends: Moore's Law

- Gordon Moore (Founder of Intel) observed in 1965 that the number of transistors on a chip doubles about every 24 months.
- In fact, the number of transistors on a chip doubles about every 18 months.



From intel

Processor Technology Trends

- Transistor density increases by 35% per year and die size increases by 10-20% per year… more functionality

- Transistor speed improves linearly with size (complex equation involving voltages, resistances, capacitances)… can lead to clock speed improvements!

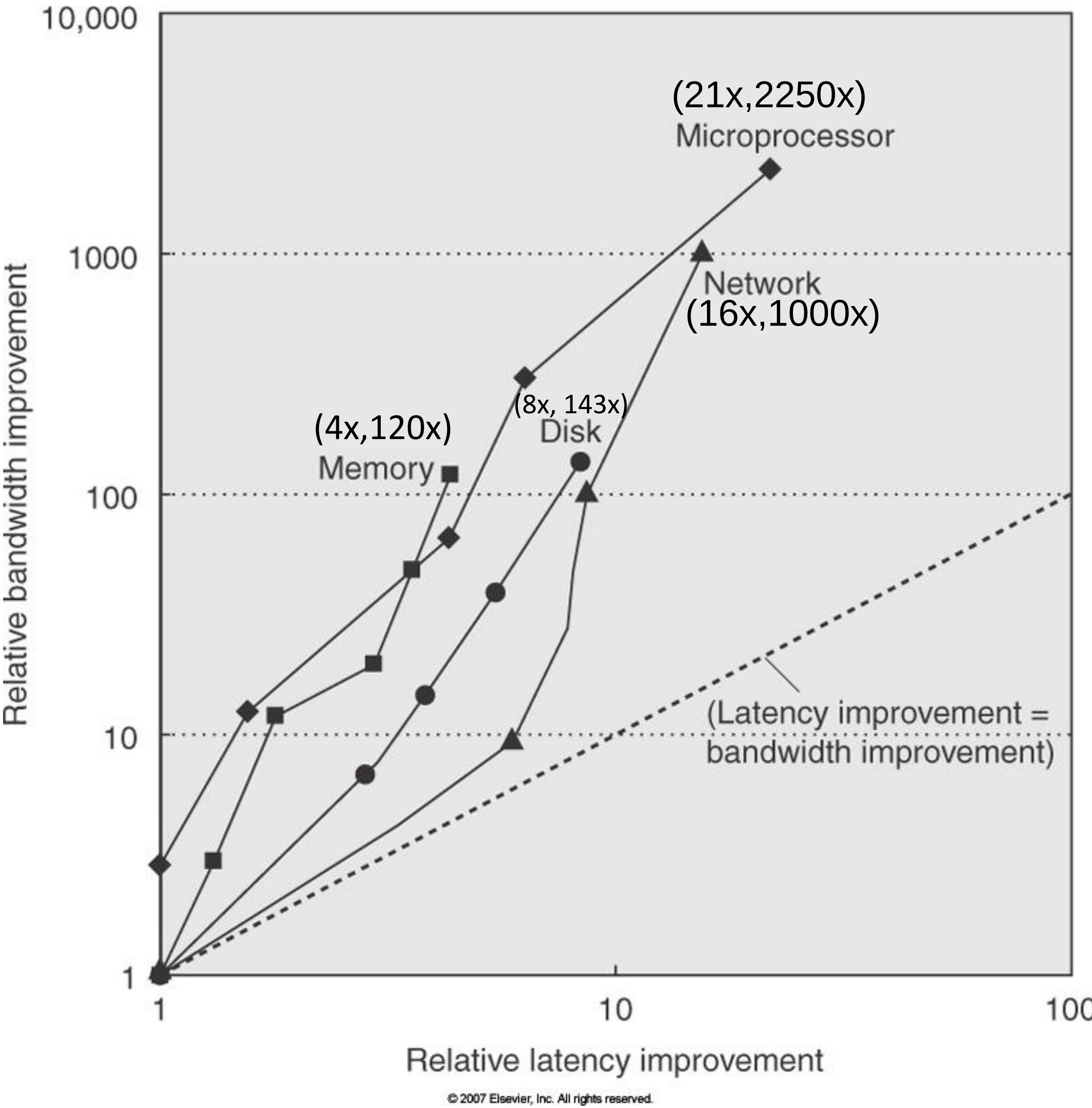
- Wire delays do not scale down at the same rate as logic delays

- The power wall: it is not possible to consistently run at higher frequencies without hitting power/thermal limits

Bandwidth vs Latency

Bandwidth or *throughput* is the total amount of work done in a given time.

Latency or response time is the time between the start and the completion of an event. (for eg. Millisecond for disk access)

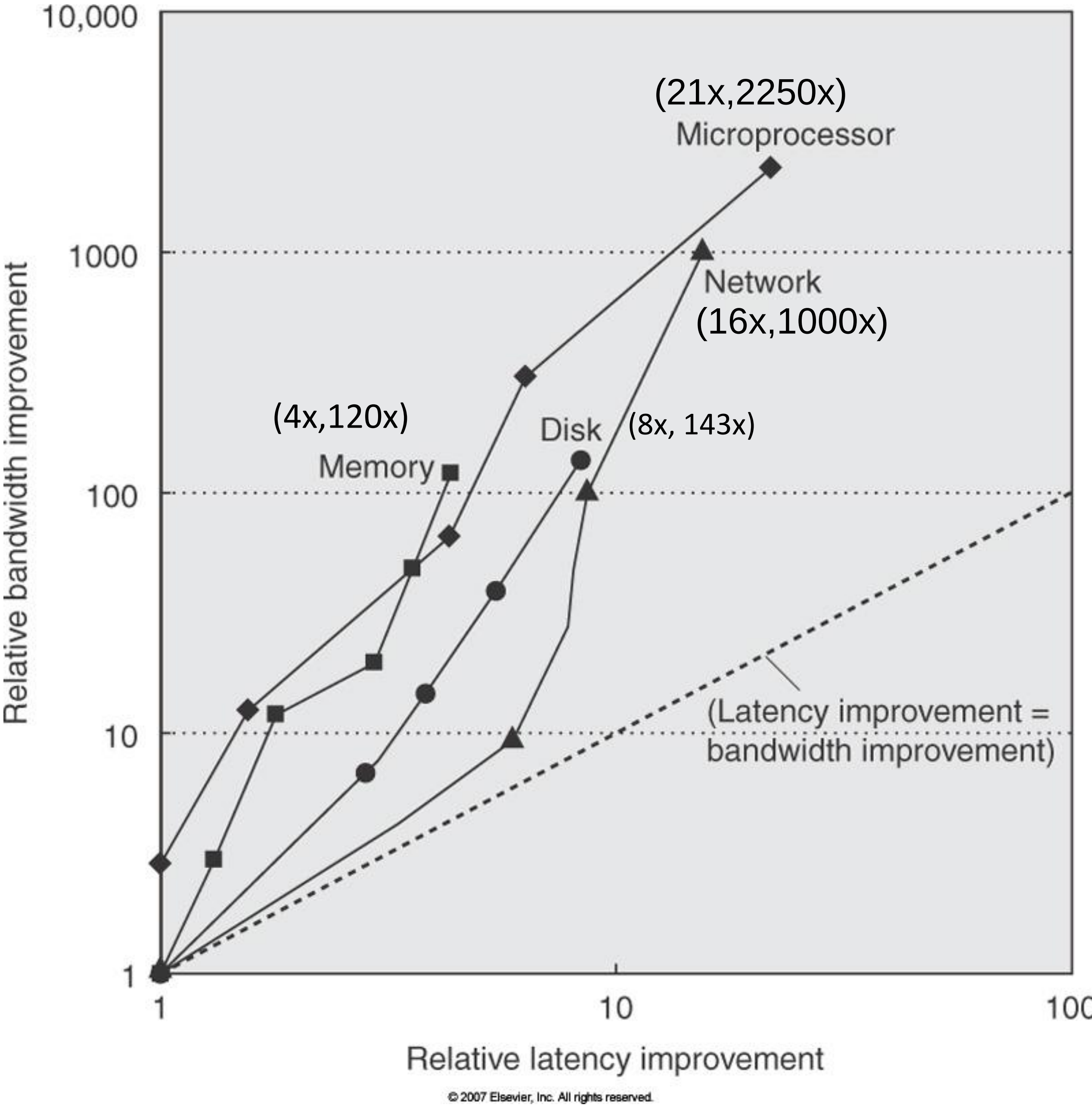


© 2007 Elsevier, Inc. All rights reserved.

Bandwidth vs Latency

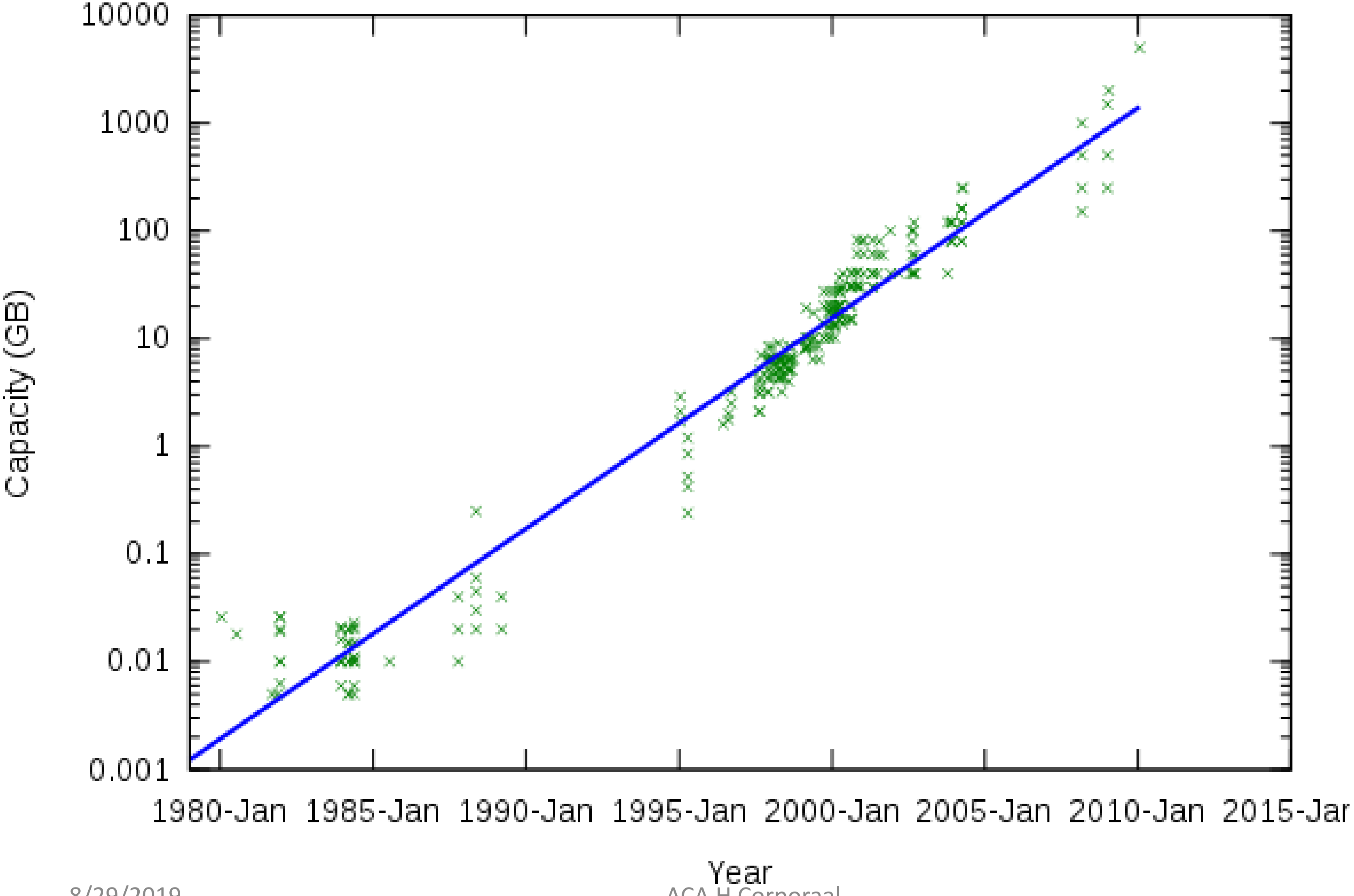
Performance Milestones

- Processor: ‘286, ‘386, ‘486, Pentium, Pentium Pro, Pentium 4 (21x,2250x)
- Ethernet: 10Mb, 100Mb, 1000Mb, 10000 Mb/s (16x,1000x)
- Memory Module: 16bit plain DRAM, Page Mode DRAM, 32b, 64b, SDRAM, DDR SDRAM (4x,120x)
- Disk : 3600, 5400, 7200, 10000, 15000 RPM (8x, 143x)



© 2007 Elsevier, Inc. All rights reserved.

PC hard drive capacity



VLSI Developments

1946: ENIAC electronic numerical integrator and computer

- Floor area
  - 140 m^2
- Performance
  - multiplication of two 10-digit numbers in 2 ms
- Power
  - 160 KWatt

2017: High Performance microprocessor

- Chip area
  - 100-400 mm^2 (for **multi-core**)
- Board area
  - 200 cm^2; improvement of 10^4
- Performance:
  - 64 bit multiply in O(1 ns); improvement of 10^6
- Power
  - 20 Watt; improvement 8000
- On top
  - architectural improvements, like ILP exploitation
  - extreme cost reduction

Technology Improvement

Where Has This Performance Improvement Come From?

- Technology
  - More transistors per chip
  - Faster logic
- Machine Organization/Implementation
  - Deeper pipelines
  - More instructions executed in parallel
- Instruction Set Architecture
  - Reduced Instruction Set Computers (RISC)
  - Multimedia extensions
  - Explicit parallelism
- Compiler technology
  - Finding more parallelism in code
  - Greater levels of optimization

What Helps Performance?

- Note: no increase in clock speed

- In a clock cycle, can do more work -- since transistors are faster, transistors are more energy-efficient, and there's more of them

- Better architectures: finding more parallelism in one thread, better branch prediction, better cache policies, better memory organizations, more thread-level parallelism, etc.

Trends in Technology

- ISA must be designed to survive rapid changes in technology
  - Designer must be aware of changes in implementation technology
- Five critical implementation technologies that change very fast
- IC Logic Technology
  - Transistor density increases by 35% per year
  - Transistor count increases by 40% to 55% per year
- Semiconductor DRAM
  - *Capacity increases by about 25% to 40% per year*
  - *Cycle time has improved slowly*

Trends in Technology

- Magnetic Disk technology

  Density increases by about 40% per year

- Network Technology

  Rapidly changing area with a high growth rate

- Performance Trends: Bandwidth over Latency
- *Bandwidth* or *throughput*

  Total amount of work done in a given time

- *Latency* or *response time*

  Time between the start and completion of an event

- Bandwidth grows by at least the square of the improvement in latency

  Designers must plan accordingly

Trends in Technology

- Scaling of Transistor Performance and Wires

  More related to VLSI technology

- *Feature size* has reduced from 10 microns (1971) to .032 microns in 2011 and 22 nanometer chips are also available
- Density of transistors increases quadratically with a linear decrease in feature size
- Performance is a complex issue
- As a first approximation, transistor performance improves linearly with decreasing feature size
- Wire delay scales poorly with decrease in feature size
- Designers have to overcome the problems of implementation

Other Trends

❑ **Trends in Power and Energy in ICs**

- **Distributing the power, removing the heat and preventing hot spots are major challenges**
  - **Power is dissipated as heat and must be removed**
- **A large number of pins of the IC is consumed in power and ground connections**
- **Amount of power consumed, maximum power and its effect on clock frequency are other main issues**

❑ **Trends in Cost**

- **Cost sensitive designs are significant**
- **Major theme in computer industry has been to increase performance and lower the cost**
  - **Maintain good cost-performance ratio**

Where Are We Headed?

- Modern trends:
  - Clock speed improvements are slowing
    - power constraints
  - Difficult to further optimize a single core for performance
  - Multi-cores: each new processor generation will accommodate more cores
  - Need better programming models and efficient execution for multi-threaded applications
  - Need better memory hierarchies
  - Need greater energy efficiency
  - Dark silicon, accelerators
  - Reduced data movement

An Overview of Parallel Processing

- What is parallel processing?
  - Parallel processing is a method to improve computer system performance by executing two or more instructions simultaneously.
- The goals of parallel processing.
  - One goal is to reduce the “wall-clock” time or the amount of real time that you need to wait for a problem to be solved.
  - Another goal is to solve bigger problems that might not fit in the limited memory of a single CPU.

Analogy is having several students grade quizzes simultaneously. Quizzes are distributed to a few students and different problems are graded by each student at the same time. After they are completed, the graded quizzes are then gathered and the scores are recorded.

Parallelism in Uniprocessor Systems

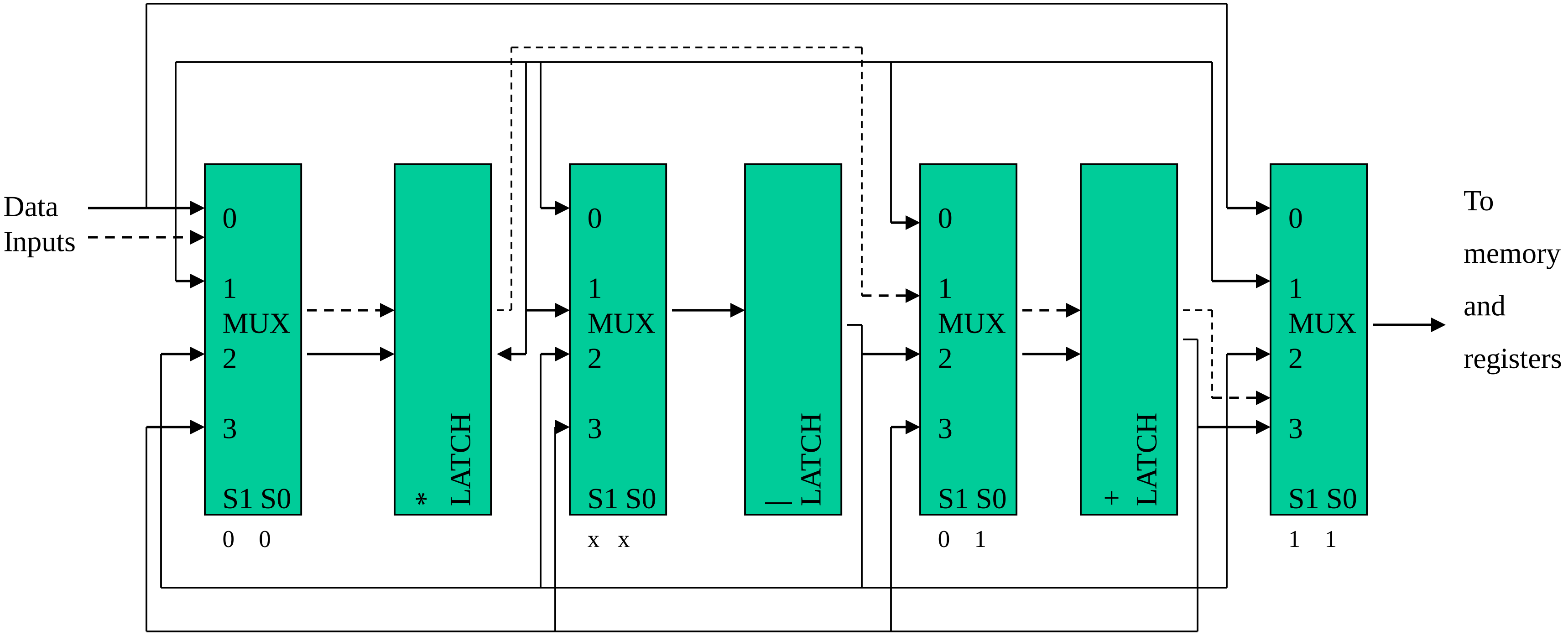
- It is possible to achieve parallelism with a uniprocessor system.
  - Some examples are the instruction pipeline, arithmetic pipeline, I/O processor.
- Note that a system that performs different operations on the same instruction is not considered parallel.
- Only if the system processes two different instructions simultaneously can be considered parallel.

Parallelism in a Uniprocessor System

- A reconfigurable arithmetic pipeline is an example of parallelism in a uniprocessor system.

  Each stage of a reconfigurable arithmetic pipeline has a multiplexer at its input. The multiplexer may pass input data, or the data output from other stages, to the stage inputs. The control unit of the CPU sets the select signals of the multiplexer to control the flow of data, thus configuring the pipeline.

A Reconfigurable Pipeline With Data Flow for the Computation A[i] ← B[i] * C[i] + D[i]



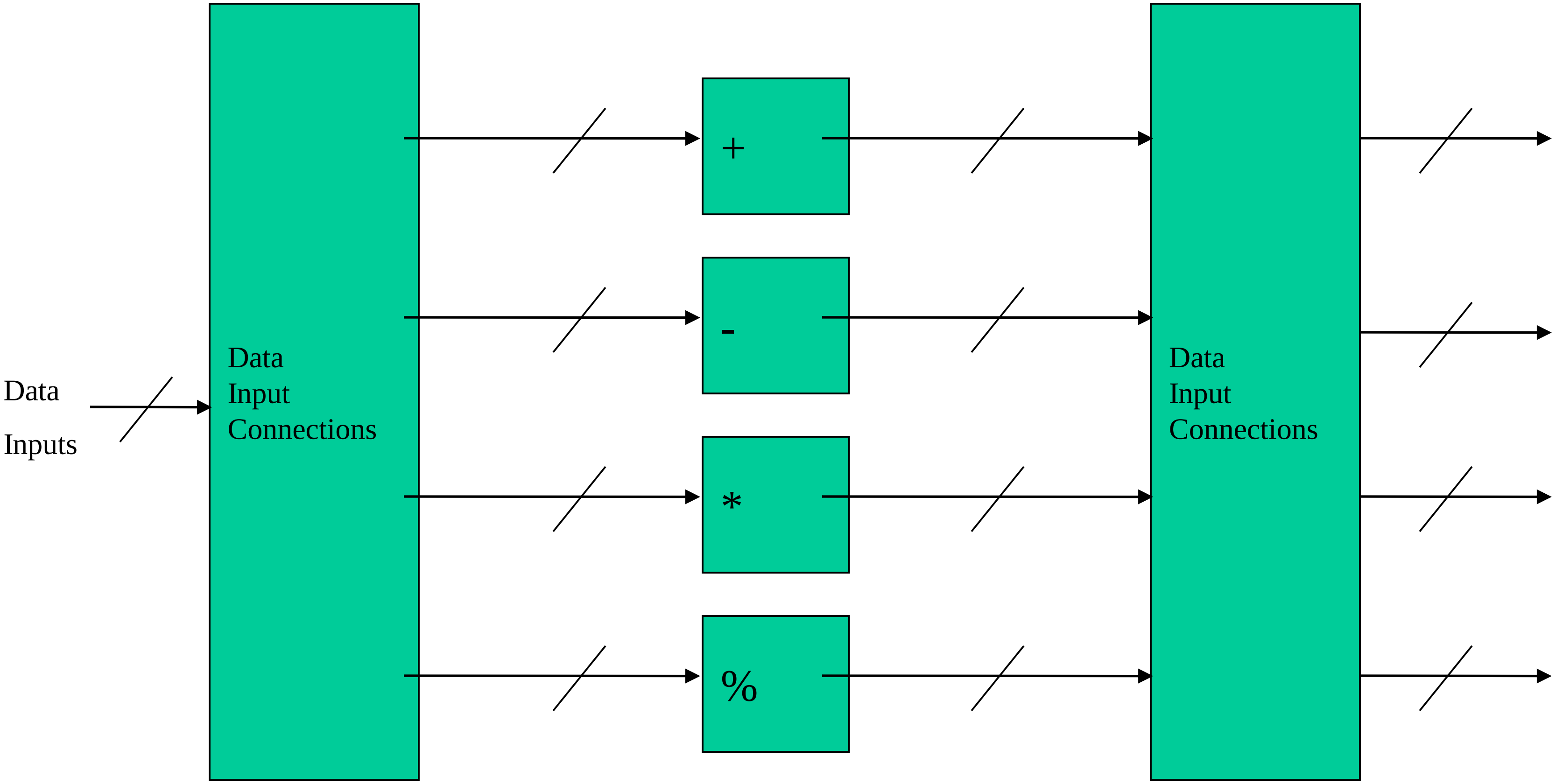
Vector Arithmetic Unit

Although arithmetic pipelines can perform many iterations of the same operation in parallel, they cannot perform different operations simultaneously. To perform different arithmetic operations in parallel, a CPU may include a vectored arithmetic unit.

A vector arithmetic unit contains multiple functional units that perform addition, subtraction, and other functions. The control unit routes input values to the different functional units to allow the CPU to execute multiple instructions simultaneously.

For the operations A←B+C and D←E-F, the CPU would route B and C to an adder and then route E and F to a subtractor for simultaneous execution.

A Vectored Arithmetic Unit



A←B+C

D←E-F

Classes of Parallelism and Parallel Architectures

- Classes of Parallelism and Parallel Architectures

  Parallelism is present at multiple levels

- Parallelism in application

  DLP: Many data items can be operated on at the same time

  TLP: Tasks of work are created that can operate in parallel

- Computer hardware can exploit these parallelism in four ways

Classes of Parallelism

1. Instruction-Level Parallelism

   Exploits data-level parallelism

   Pipelining and speculative execution

2. Vector Architectures and Graphic Processor Units

   Exploits DLP by applying a single instruction on a collection of data in parallel

3. Thread-Level Parallelism

   Exploits DLP or TLP in a tightly coupled hardware model that allows interaction among parallel threads

4. Request-Level Parallelism

   Exploits parallelism among largely decoupled tasks specified by the programmer or the OS