18-447 Computer Architecture Lecture 2: Fundamental Concepts and ISA

Prof. Onur Mutlu
Carnegie Mellon University
Spring 2015, 1/14/2014

Agenda for Today

- Finish up logistics from last lecture
- Why study computer architecture?
- Some fundamental concepts in computer architecture
- ISA

Last Lecture Recap

- What it means/takes to be a good (computer) architect
  - Roles of a computer architect (look everywhere!)
- Goals of 447 and what you will learn in this course
- Levels of transformation
- Abstraction layers, their benefits, and the benefits of comfortably crossing them
- Three example problems and solution ideas
  - Memory Performance Attacks
  - DRAM Refresh
  - Row Hammer: DRAM Disturbance Errors
- Hamming Distance and Bloom Filters
- Course Logistics
- Assignments: HW0 (Jan 16), Lab1 (Jan 23), HW1 (Jan 28)

Review: Key Takeaway (from 3 Problems)

- Breaking the abstraction layers (between components and transformation hierarchy levels) and knowing what is underneath enables you to solve problems and design better future systems

- Cooperation between multiple components and layers can enable more effective solutions and systems

A Note on Hardware vs. Software

- This course is classified under "Computer Hardware"

- However, you will be much more capable if you master both hardware and software (and the interface between them)
  - Can develop better software if you understand the underlying hardware
  - Can design better hardware if you understand what software it will execute
  - Can design a better computing system if you understand both

- This course covers the HW/SW interface and microarchitecture
  - We will focus on tradeoffs and how they affect software

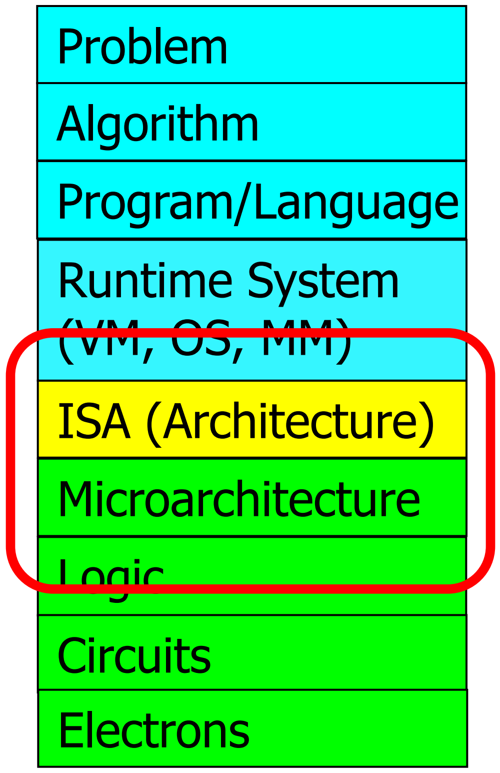
What Will You Learn

- **Computer Architecture:** 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.

- **Traditional definition:** "The term *architecture* is used here to describe the attributes of a programmer, i.e., the conceptual s behavior as distinct from the orga and controls, the logic design, and implementation." *Gene Amdahl*, IE 1964

Dr. Amdahl holding a 100gate LSI air-cooled chip. On his desk is a circuit board with the chips on it. This circuit board was for an Amdahl 470 V/6 (photograph dated March 1973).

Computer Architecture in Levels of Transformation



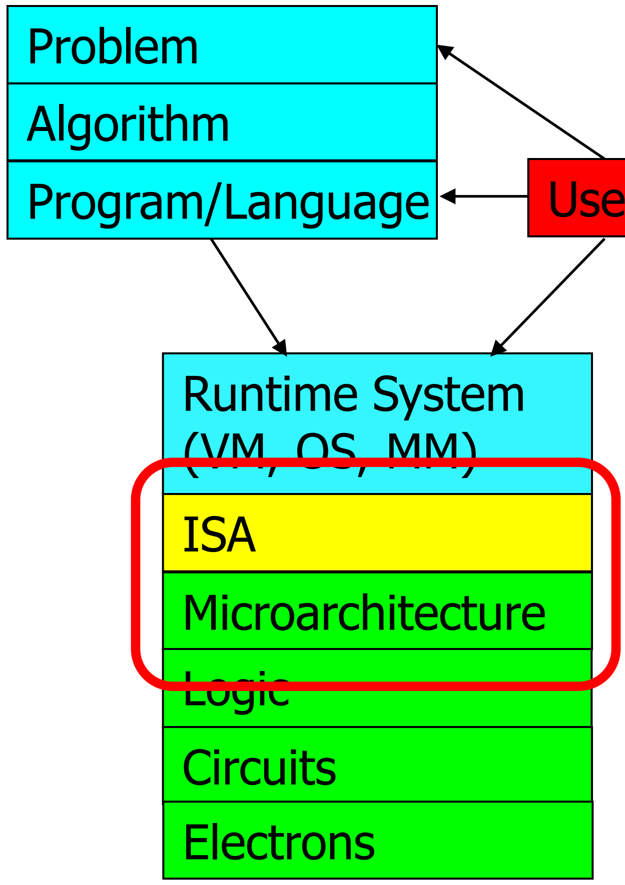
- Read: Patt, "Requirements, Bottlenecks, and Good Fortune: Agents for Microprocessor Evolution," Proceedings of the IEEE 2001.

Aside: What Is An Algorithm?

- Step-by-step procedure where each step has three properties:
  - Definite (precisely defined)
  - Effectively computable (by a computer)
  - Terminates

Levels of Transformation, Revisited

- A user-centric view: computer designed for users



- The entire stack should be optimized for user

What Will You Learn?

- Fundamental principles and tradeoffs in designing the hardware/software interface and major components of a modern programmable microprocessor
  - Focus on state-of-the-art (and some recent research and trends)
  - Trade-offs and how to make them

- How to design, implement, and evaluate a functional modern processor
  - Semester-long lab assignments
  - A combination of RTL implementation and higher-level simulation
  - Focus is functionality first (then, on "how to do even better")

- How to dig out information, think critically and broadly
- How to work even harder and more efficiently!

Course Goals

- Goal 1: To familiarize those interested in computer system design with both fundamental operation principles and design tradeoffs of processor, memory, and platform architectures in today's systems.
    - Strong emphasis on fundamentals, design tradeoffs, key current/future issues
    - Strong emphasis on looking backward, forward, up and down

- Goal 2: To provide the necessary background and experience to design, implement, and evaluate a modern processor by performing hands-on RTL and C-level implementation.
    - Strong emphasis on functionality, hands-on design & implementation, and efficiency.
    - Strong emphasis on making things work, realizing ideas

Reminder: What Do I Expect From You?

- Required background: 240 (digital logic, RTL implementation, Verilog), 213 (systems, virtual memory, assembly)

- Learn the material thoroughly
  - attend lectures, do the readings, do the homeworks
- Do the work & work hard
- Ask questions, take notes, participate
- Perform the assigned readings
- **Come to class on time**
- Start early – do not procrastinate
- If you want feedback, come to office hours

- Remember "Chance favors the prepared mind." (Pasteur)

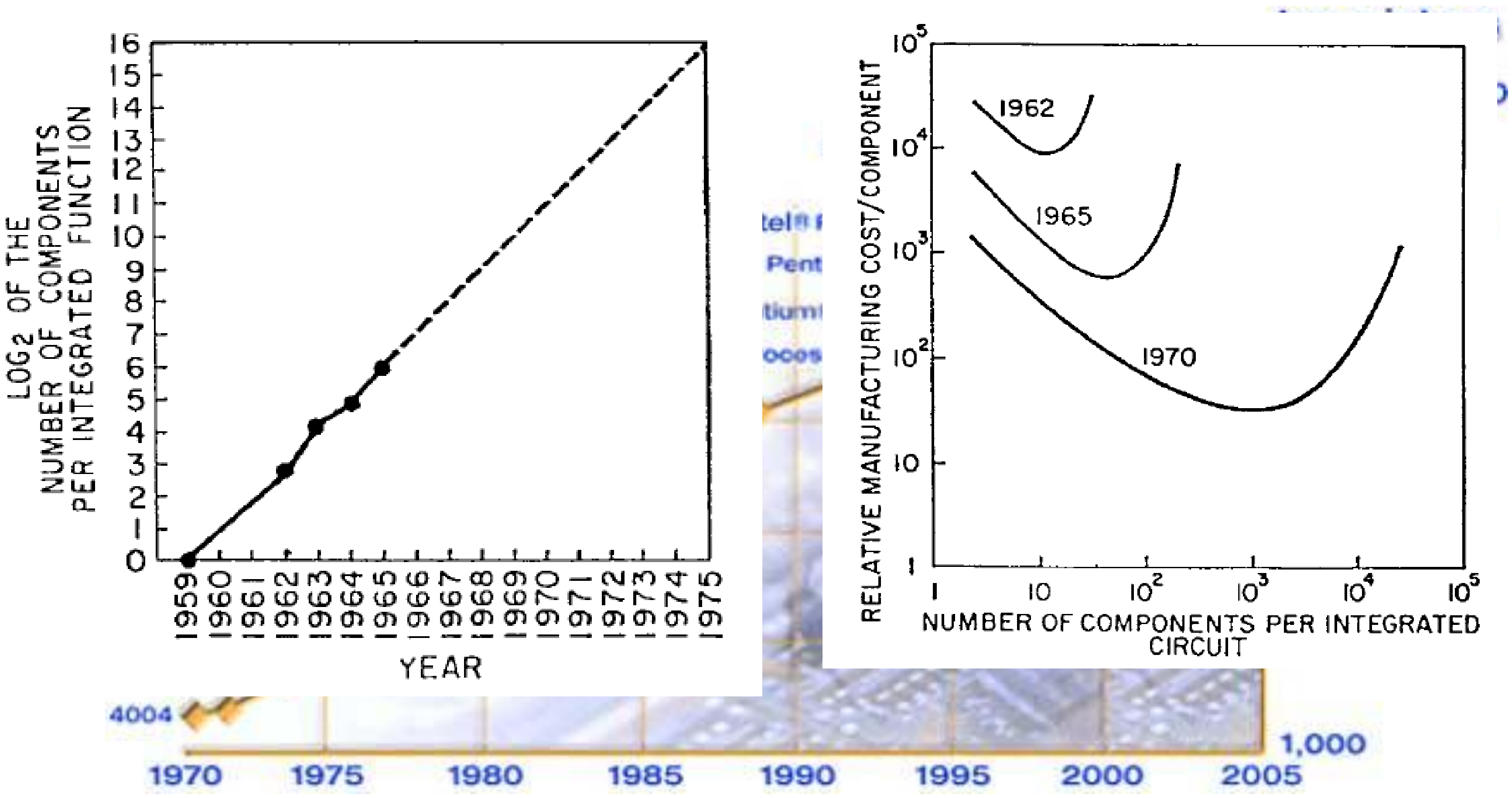
Why Study Computer Architecture?

What is Computer Architecture?

- 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.

- We will soon distinguish between the terms *architecture*, and *microarchitecture*.

An Enabler: Moore's Law

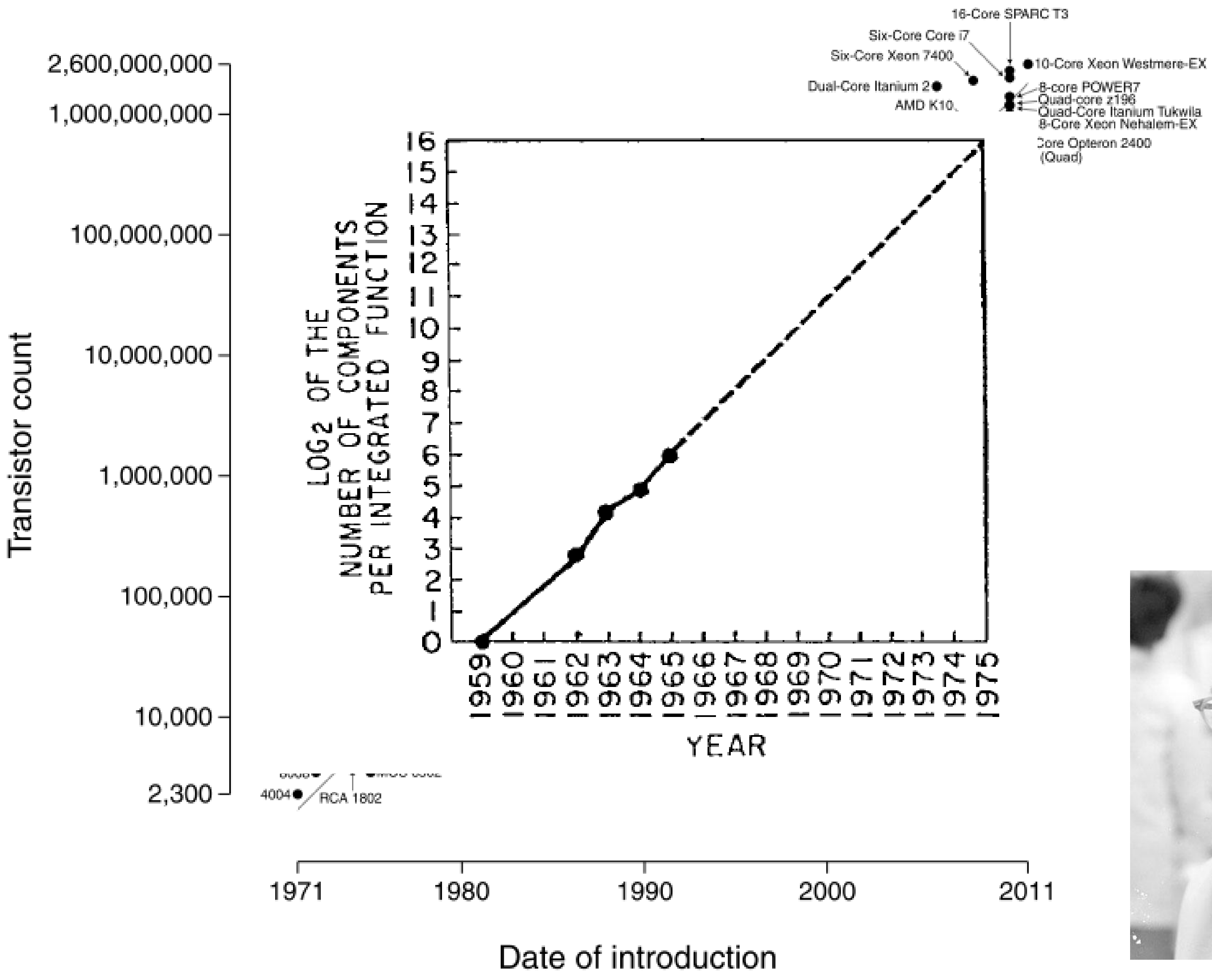


Moore, "Cramming more components onto integrated circuits," Electronics Magazine, 1965.

Component counts double every other year

Image source: Intel

Microprocessor Transistor Counts 1971-2011 & Moore's Law



Number of transistors on an integrated circuit doubles ~ every two years

Image source: Wikipedia

Recommended Reading

- Moore, "Cramming more components onto integrated circuits," Electronics Magazine, 1965.

- Only 3 pages

- A quote:

  "*With unit cost falling as the number of components per circuit rises, by 1975 economics may dictate squeezing as many as 65 000 components on a single silicon chip.*"

- Another quote:

  "*Will it be possible to remove the heat generated by tens of thousands of components in a single silicon chip?*"

What Do We Use These Transistors for?

- Your readings for this week should give you an idea…

- Patt, "Requirements, Bottlenecks, and Good Fortune: Agents for Microprocessor Evolution," Proceedings of the IEEE 2001.

- One of:
  - Moscibroda and Mutlu, "Memory Performance Attacks: Denial of Memory Service in Multi-Core Systems," USENIX Security 2007.
  - Liu+, "RAIDR: Retention-Aware Intelligent DRAM Refresh," ISCA 2012.
  - Kim+, "Flipping Bits in Memory Without Accessing Them: An Experimental Study of DRAM Disturbance Errors," ISCA 2014.

Why Study Computer Architecture?

- Enable better systems: make computers faster, cheaper, smaller, more reliable, …
  - By exploiting advances and changes in underlying technology/circuits

- Enable new applications
  - Life-like 3D visualization 20 years ago?
  - Virtual reality?
  - Personalized genomics? Personalized medicine?

- Enable better solutions to problems
  - Software innovation is built into trends and changes in computer architecture
    - > 50% performance improvement per year has enabled this innovation

- Understand why computers work the way they do

Computer Architecture Today (I)

- Today is a very exciting time to study computer architecture

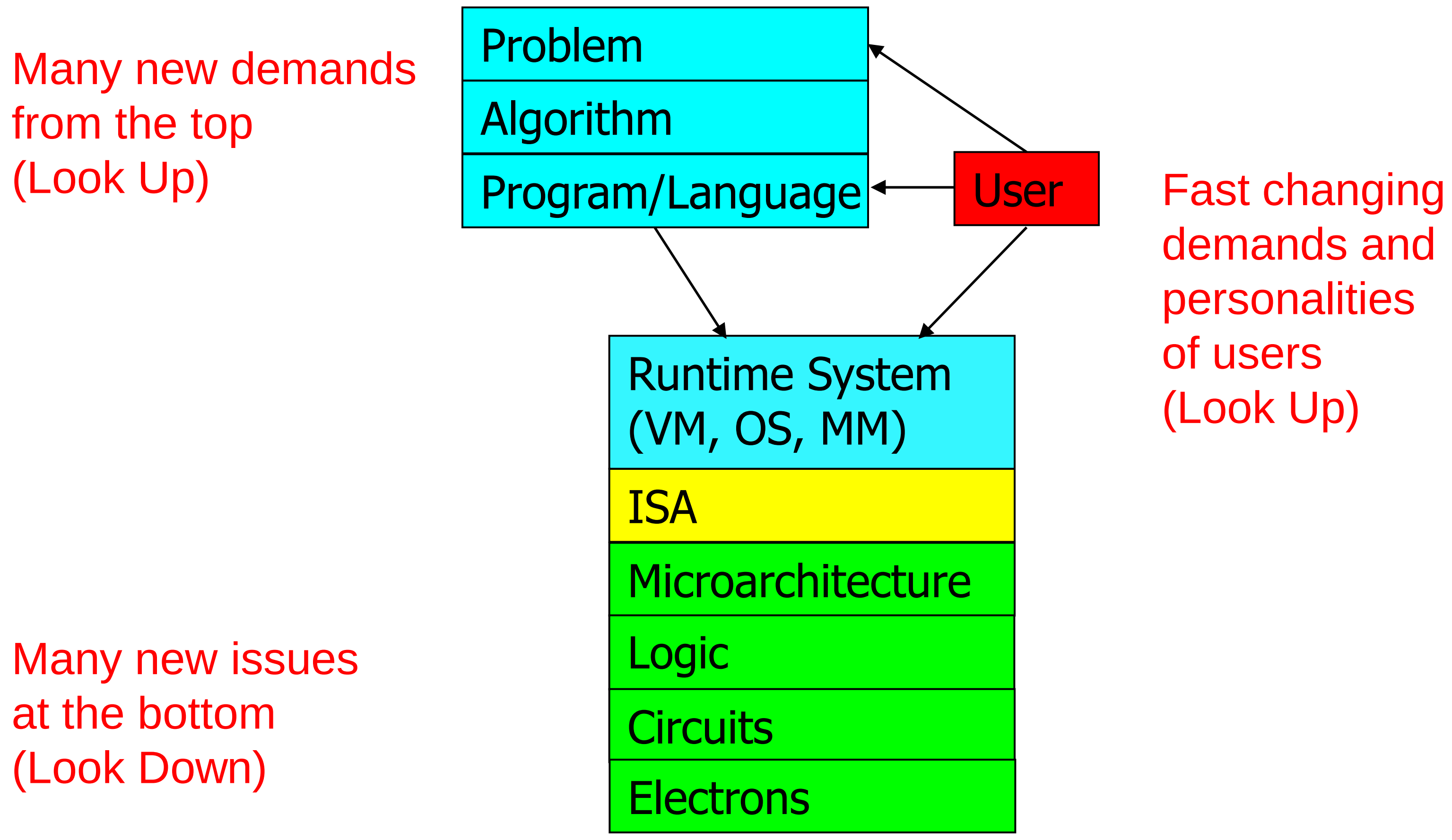
- Industry is in a large paradigm shift (to multi-core and beyond) – many different potential system designs possible

- Many difficult problems *motivating* and *caused by* the shift
    - Power/energy constraints → multi-core?
    - Complexity of design → multi-core?
    - Difficulties in technology scaling → new technologies?
    - Memory wall/gap
    - Reliability wall/issues
    - Programmability wall/problem
    - Huge hunger for data and new data-intensive applications

- No clear, definitive answers to these problems

Computer Architecture Today (II)

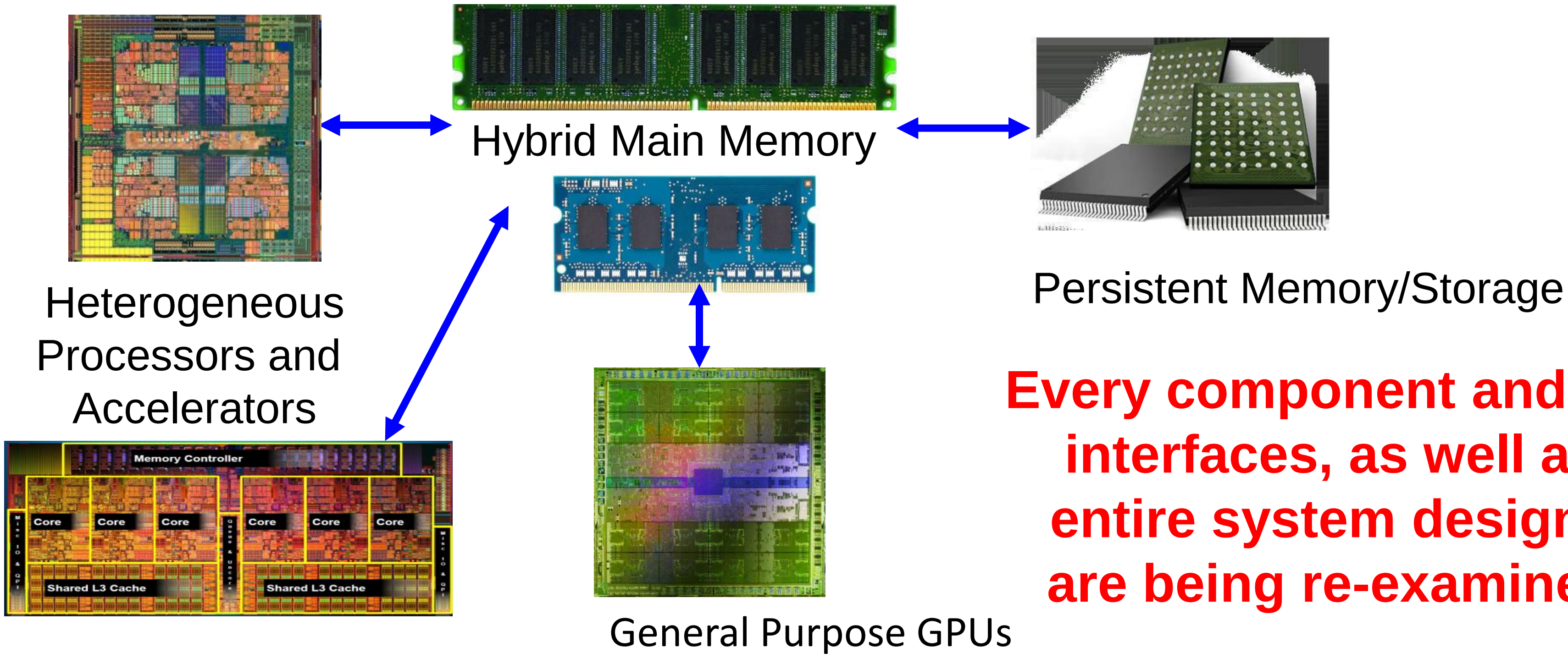
- These problems affect all parts of the computing stack – if we do not change the way we design systems


Many new demands from the top (Look Up)
Problem
Algorithm
Program/Language
User
Fast changing demands and personalities of users (Look Up)
Runtime System (VM, OS, MM)
ISA
Microarchitecture
Logic
Circuits
Electrons
Many new issues at the bottom (Look Down)

- No clear, definitive answers to these problems

Computer Architecture Today (III)

- Computing landscape is very different from 10-20 years ago
- Both UP (software and humanity trends) and DOWN (technologies and their issues), FORWARD and BACKWARD, and the resulting requirements and constraints



Every component and its interfaces, as well as entire system designs are being re-examined

Computer Architecture Today (IV)

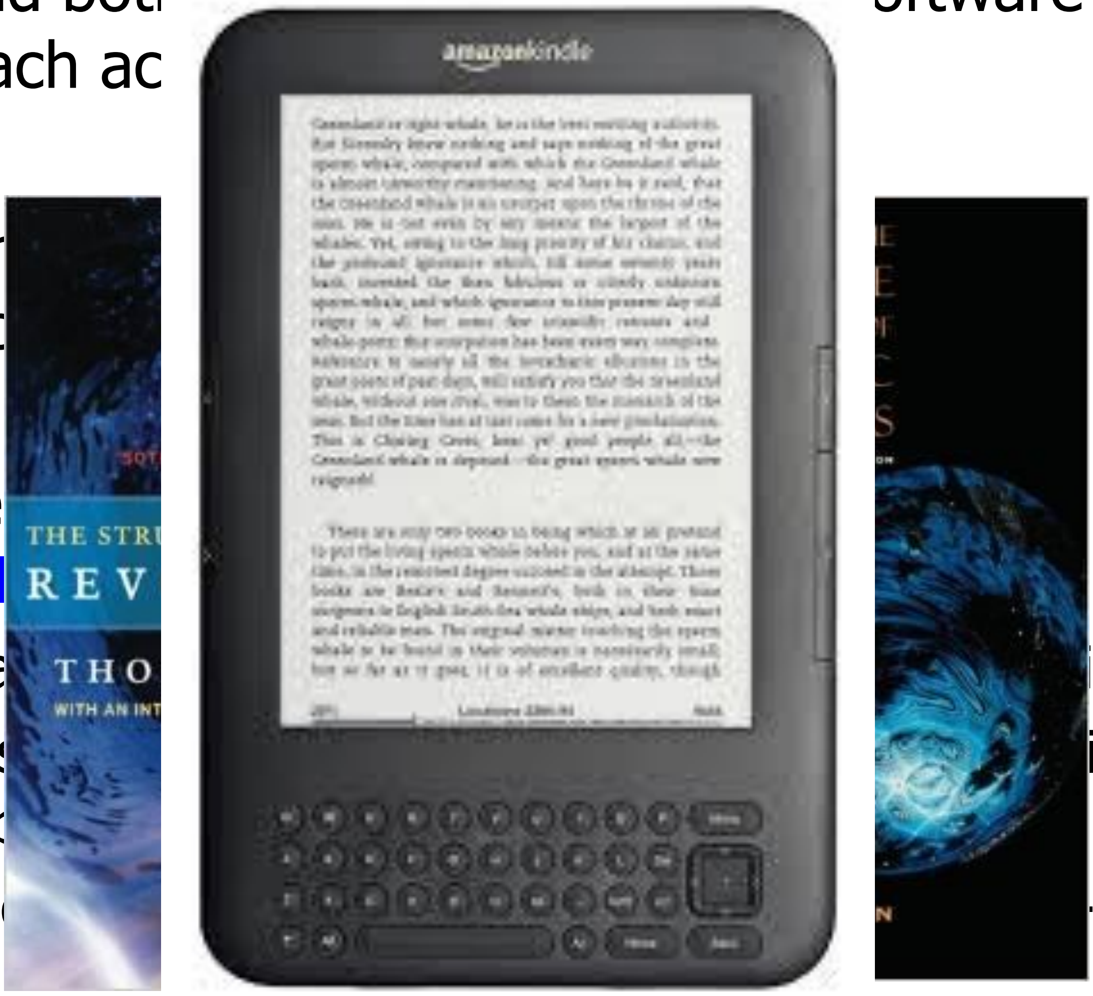
- You can revolutionize the way computers are built, if you understand both the hardware and the software (and change each accordingly)

- You can invent new paradigms for computation, communication, and storage

- Recommended book: Thomas Kuhn, "The Structure of Scientific Revolutions" (1962)
  - Pre-paradigm science: no clear consensus in the field
  - Normal science: dominant theory used to explain/improve things (business as usual); exceptions considered anomalies
  - Revolutionary science: underlying assumptions re-examined

Computer Architecture Today (IV)

- You can revolutionize the way computers are built, if you understand bot oftware (and change each ac
- You can ir communic
- Recomme ure of Scientific I
  - Pre-para ield
  - Normal s improve things (t anomalies
  - Revoluti examined



… but, first …

- Let’s understand the fundamentals…

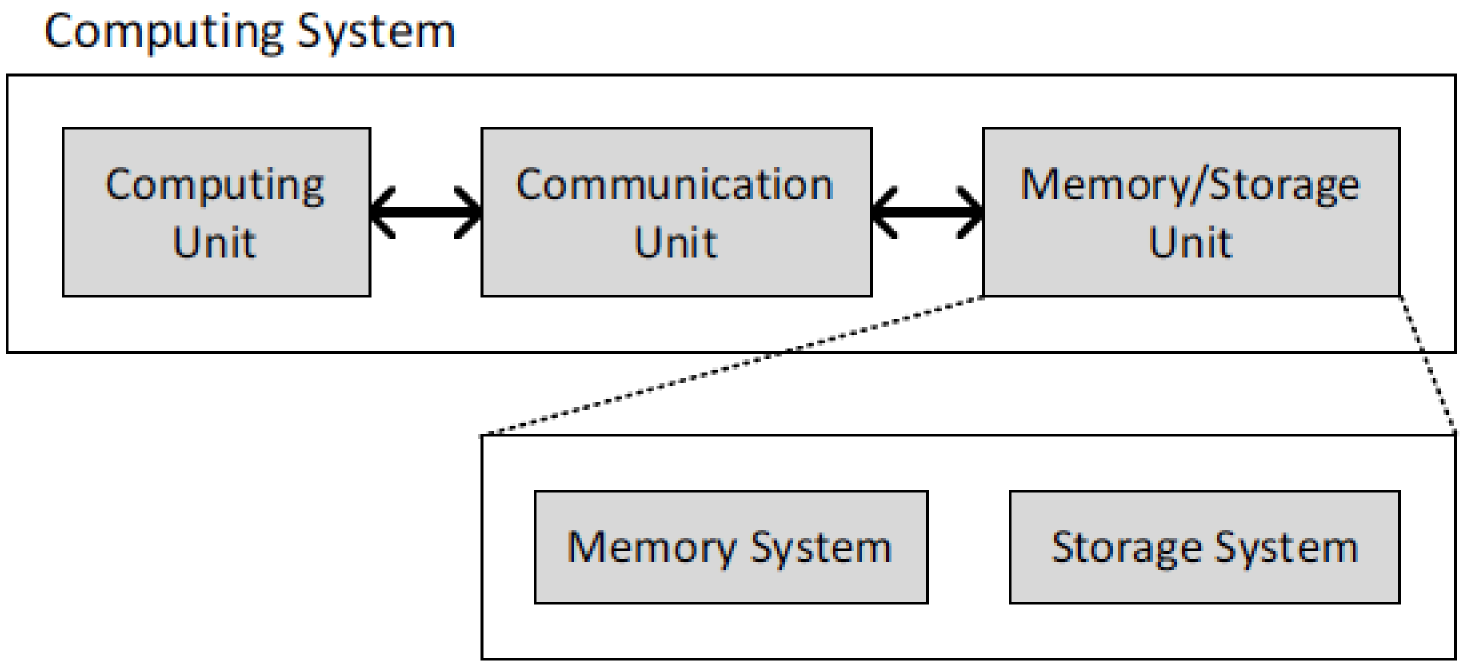
- You can change the world only if you understand it well enough…
  - Especially the past and present dominant paradigms
  - And, their advantages and shortcomings – tradeoffs
  - And, what remains fundamental across generations
  - And, what techniques you can use and develop to solve problems

Fundamental Concepts

What is A Computer?

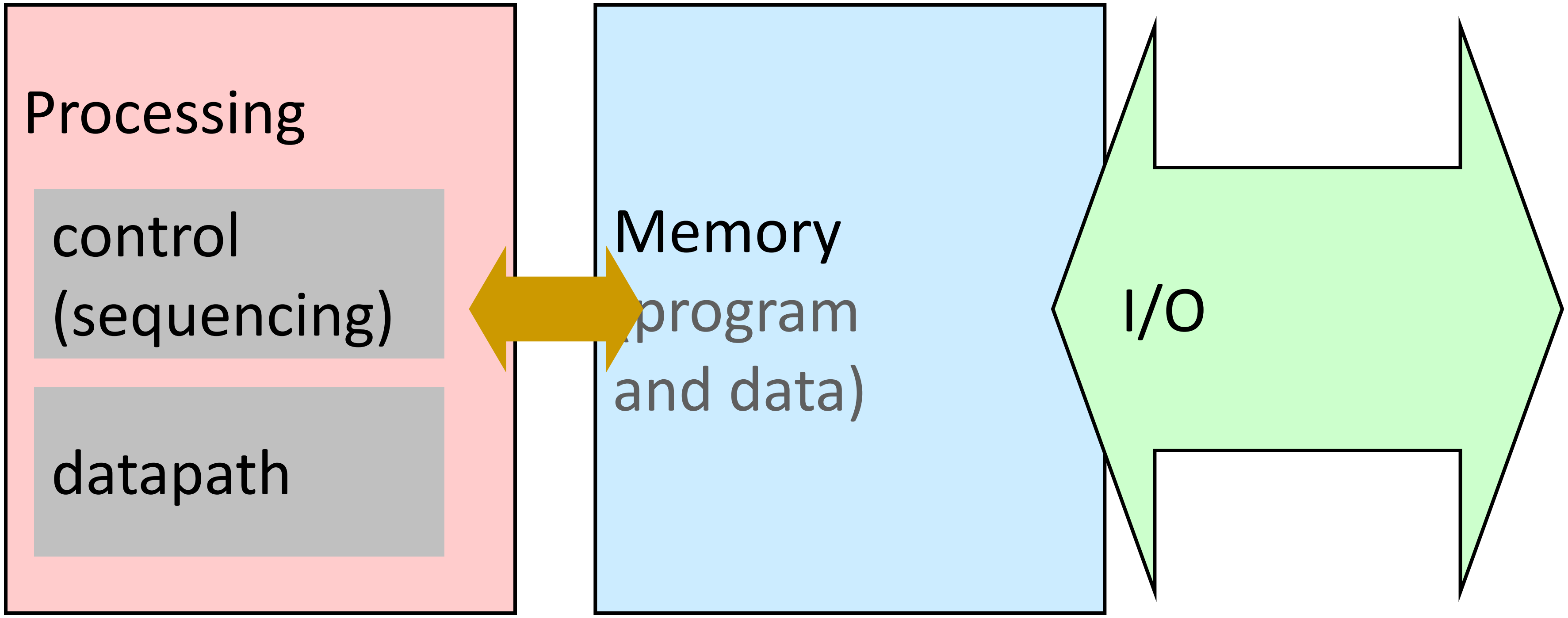
- Three key components

- Computation
- Communication
- Storage (memory)



What is A Computer?

- We will cover all three components



The Von Neumann Model/Architecture

- Also called *stored program computer* (instructions in memory). Two key properties:

- Stored program
  - Instructions stored in a linear memory array
  - Memory is unified between instructions and data
    - The interpretation of a stored value depends on the control signals

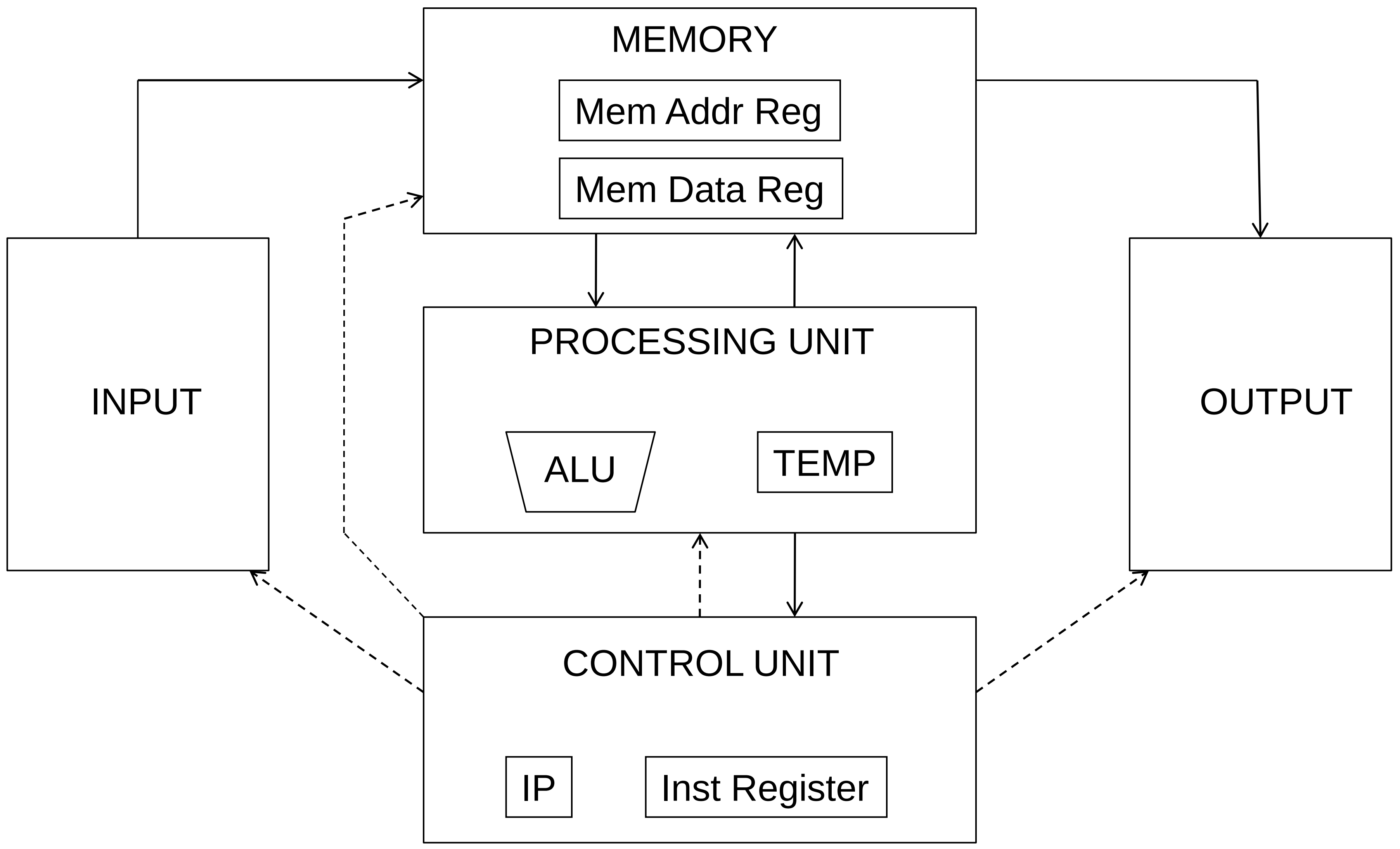
When is a value interpreted as an instruction?

- Sequential instruction processing
  - One instruction processed (fetched, executed, and completed) at a time
  - Program counter (instruction pointer) identifies the current instr.
  - Program counter is advanced sequentially except for control transfer instructions

The Von Neumann Model/Architecture

- Recommended reading
  - Burks, Goldstein, von Neumann, "Preliminary discussion of the logical design of an electronic computing instrument," 1946.
  - Patt and Patel book, Chapter 4, "The von Neumann Model"
- Stored program
- Sequential instruction processing

The Von Neumann Model (of a Computer)



The Von Neumann Model (of a Computer)

- Q: Is this the only way that a computer can operate?

- A: No.
- Qualified Answer: But, it has been the dominant way
  - i.e., the dominant paradigm for computing
  - for N decades

The Dataflow Model (of a Computer)

- Von Neumann model: An instruction is fetched and executed in control flow order
  - As specified by the instruction pointer
  - Sequential unless explicit control flow instruction

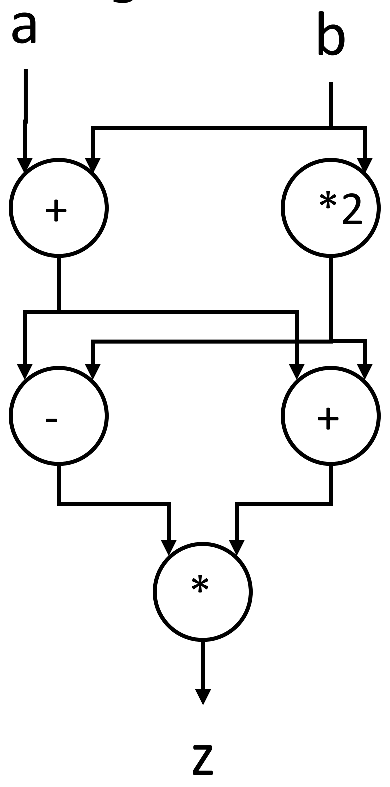
- Dataflow model: An instruction is fetched and executed in data flow order
  - i.e., when its operands are ready
  - i.e., there is no instruction pointer
  - Instruction ordering specified by data flow dependence
    - Each instruction specifies "who" should receive the result
    - An instruction can "fire" whenever all operands are received
  - Potentially many instructions can execute at the same time
    - Inherently more parallel

Von Neumann vs Dataflow

- Consider a Von Neumann program
  - What is the significance of the program order?
  - What is the significance of the storage locations?

```
v <= a + b;
w <= b * 2;
x <= v - w
y <= v + w
z <= x * y
```

Sequential

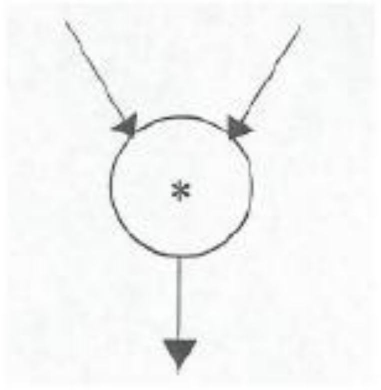


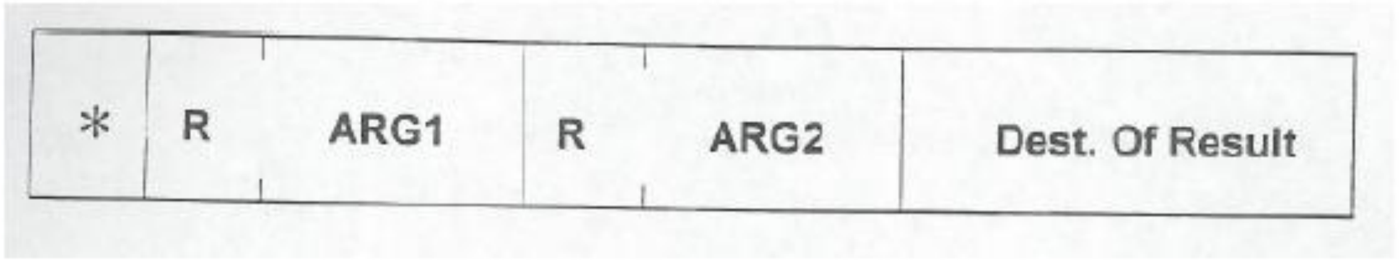
Dataflow

- Which model is more natural to you as a programmer?

More on Data Flow

- In a data flow machine, a program consists of data flow nodes
  - A data flow node fires (fetched and executed) when all it inputs are ready
    - i.e. when all inputs have tokens
- Data flow node and its ISA representation

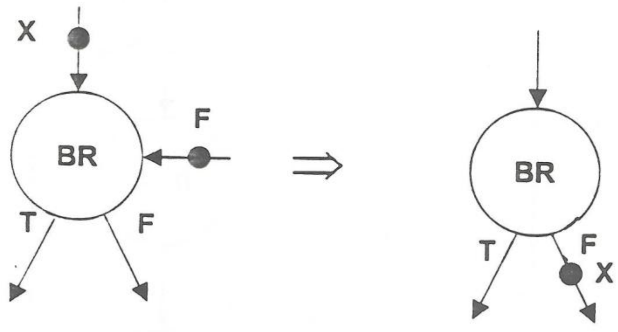




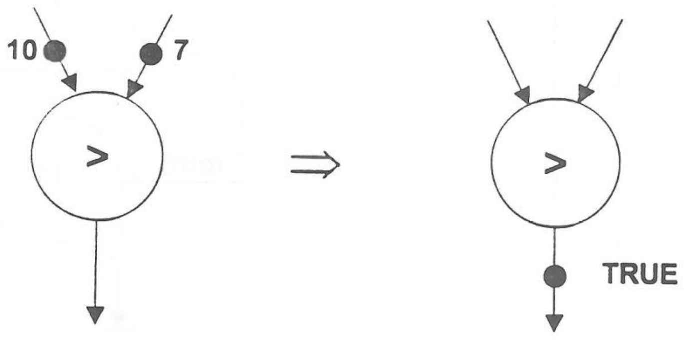
*	R	ARG1	R	ARG2	Dest. Of Result

Data Flow Nodes

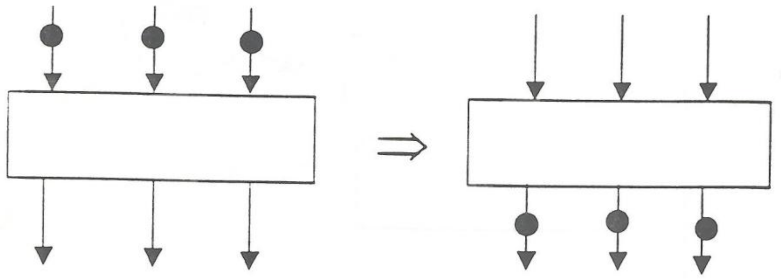
✱*Conditional*



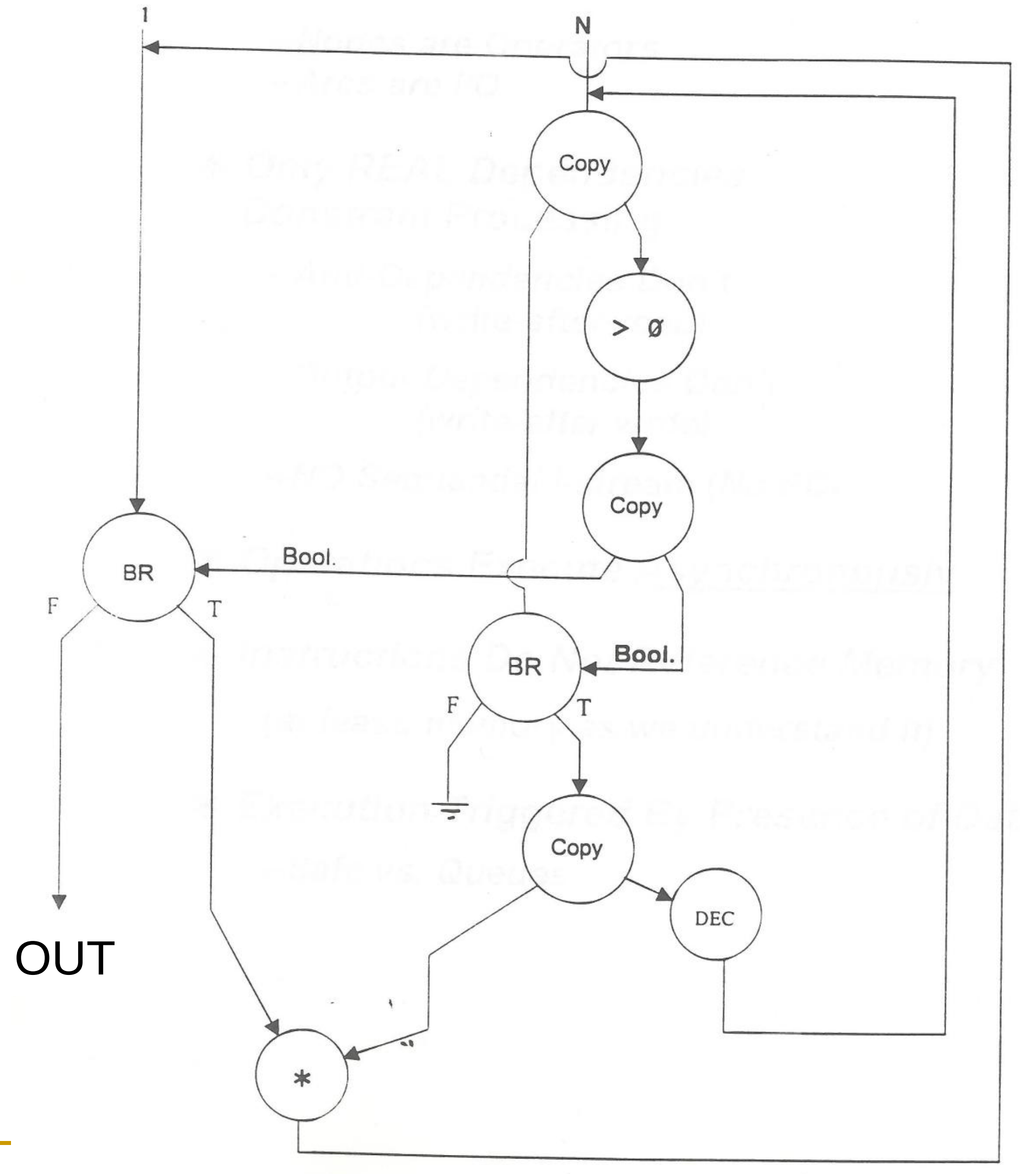
✱*Relational*



✱**Barrier Synch**



An Example Data Flow Program



ISA-level Tradeoff: Instruction Pointer

- Do we need an instruction pointer in the ISA?
  - Yes: Control-driven, sequential execution
    - An instruction is executed when the IP points to it
    - IP automatically changes sequentially (except for control flow instructions)
  - No: Data-driven, parallel execution
    - An instruction is executed when all its operand values are available (data flow)

- Tradeoffs: MANY high-level ones
  - Ease of programming (for average programmers)?
  - Ease of compilation?
  - Performance: Extraction of parallelism?
  - Hardware complexity?

ISA vs. Microarchitecture Level Tradeoff

- A similar tradeoff (control vs. data-driven execution) can be made at the microarchitecture level

- ISA: Specifies how the programmer sees instructions to be executed
  - Programmer sees a sequential, control-flow execution order vs.
  - Programmer sees a data-flow execution order

- Microarchitecture: How the underlying implementation actually executes instructions
  - Microarchitecture can execute instructions in any order as long as it obeys the semantics specified by the ISA when making the instruction results visible to software
    - Programmer should see the order specified by the ISA

Let's Get Back to the Von Neumann Model

- But, if you want to learn more about dataflow…

- Dennis and Misunas, "A preliminary architecture for a basic data-flow processor," ISCA 1974.
- Gurd et al., "The Manchester prototype dataflow computer," CACM 1985.
- A later 447 lecture, 740/742

- If you are really impatient:
  - http://www.youtube.com/watch?v=D2uue7izU2c
  - http://www.ece.cmu.edu/~ece740/f13/lib/exe/fetch.php?media=onur-740-fall13-module5.2.1-dataflow-part1.ppt

The Von-Neumann Model

- All major *instruction set architectures* today use this model
  - x86, ARM, MIPS, SPARC, Alpha, POWER

- Underneath (at the microarchitecture level), the execution model of almost all *implementations (or, microarchitectures)* is very different
  - Pipelined instruction execution: *Intel 80486 uarch*
  - Multiple instructions at a time: *Intel Pentium uarch*
  - Out-of-order execution: *Intel Pentium Pro uarch*
  - Separate instruction and data caches

- But, what happens underneath that is *not* consistent with the von Neumann model is *not* exposed to software
  - Difference between ISA and microarchitecture

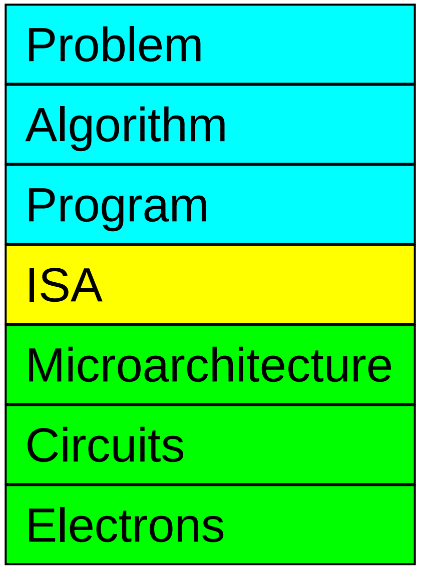
What is Computer Architecture?

- **ISA+implementation definition:** 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.

- **Traditional (ISA-only) definition:** "The term *architecture* is used here to describe the attributes of a system as seen by the programmer, i.e., the conceptual structure and functional behavior as distinct from the organization of the dataflow and controls, the logic design, and the physical implementation." *Gene Amdahl*, IBM Journal of R&D, April 1964

ISA vs. Microarchitecture

- ISA
  - Agreed upon interface between software and hardware
    - SW/compiler assumes, HW promises
  - What the software writer needs to know to write and debug system/user programs
- Microarchitecture
  - Specific implementation of an ISA
  - Not visible to the software
- Microprocessor
  - **ISA, uarch**, circuits
  - "Architecture" = ISA + microarchitecture



ISA vs. Microarchitecture

- What is part of ISA vs. Uarch?
    - Gas pedal: interface for "acceleration"
    - Internals of the engine: implement "acceleration"

- Implementation (uarch) can be various as long as it satisfies the specification (ISA)
    - Add instruction vs. Adder implementation
        - Bit serial, ripple carry, carry lookahead adders are all part of microarchitecture
    - x86 ISA has many implementations: 286, 386, 486, Pentium, Pentium Pro, Pentium 4, Core, …

- Microarchitecture usually changes faster than ISA
    - Few ISAs (x86, ARM, SPARC, MIPS, Alpha) but many uarchs
    - *Why?*

ISA

- Instructions
  - Opcodes, Addressing Modes, Data Types
  - Instruction Types and Formats
  - Registers, Condition Codes
- Memory
  - Address space, Addressability, Alignment
  - Virtual memory management
- Call, Interrupt/Exception Handling
- Access Control, Priority/Privilege
- I/O: memory-mapped vs. instr.
- Task/thread Management
- Power and Thermal Management
- Multi-threading support, Multiprocessor support

Intel® 64 and IA-32 Architectures
Software Developer's Manual

Volume 1:
Basic Architecture

Microarchitecture

- Implementation of the ISA under specific design constraints and goals
- Anything done in hardware without exposure to software
  - Pipelining
  - In-order versus out-of-order instruction execution
  - Memory access scheduling policy
  - Speculative execution
  - Superscalar processing (multiple instruction issue?)
  - Clock gating
  - Caching? Levels, size, associativity, replacement policy
  - Prefetching?
  - Voltage/frequency scaling?
  - Error correction?

We did not cover the following slides in lecture. These are for your preparation for the next lecture.

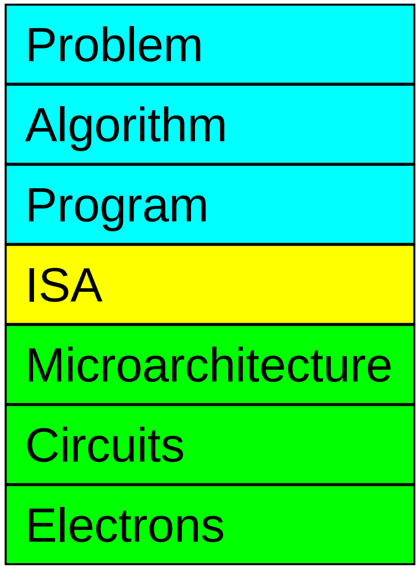
Property of ISA vs. Uarch?

- ADD instruction's opcode
- Number of general purpose registers
- Number of ports to the register file
- Number of cycles to execute the MUL instruction
- Whether or not the machine employs pipelined instruction execution

- Remember
  - Microarchitecture: Implementation of the ISA under specific design constraints and goals

Design Point

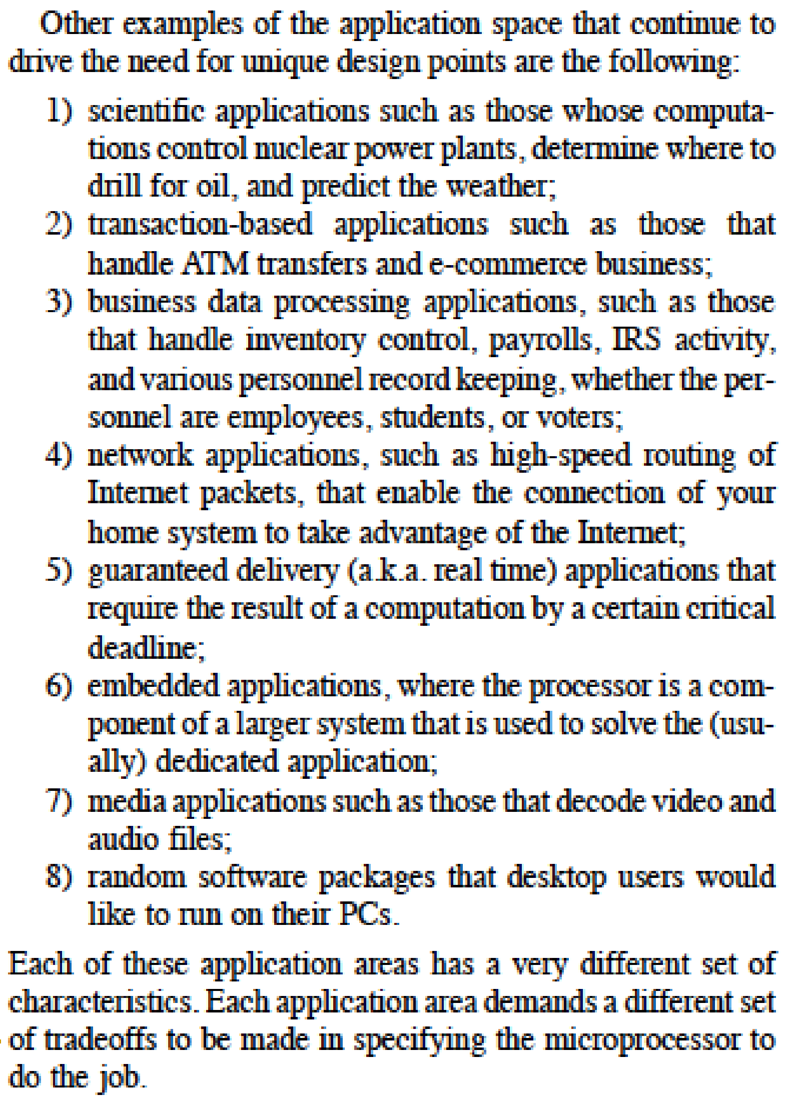
- A set of design considerations and their importance
  - leads to tradeoffs in both ISA and uarch
- Considerations
  - Cost
  - Performance
  - Maximum power consumption
  - Energy consumption (battery life)
  - Availability
  - Reliability and Correctness
  - Time to Market



- Design point determined by the "Problem" space (application space), the intended users/*market*

Application Space

- Dream, and they will appear…



Other examples of the application space that continue to drive the need for unique design points are the following:

1) scientific applications such as those whose computations control nuclear power plants, determine where to drill for oil, and predict the weather;
2) transaction-based applications such as those that handle ATM transfers and e-commerce business;
3) business data processing applications, such as those that handle inventory control, payrolls, IRS activity, and various personnel record keeping, whether the personnel are employees, students, or voters;
4) network applications, such as high-speed routing of Internet packets, that enable the connection of your home system to take advantage of the Internet;
5) guaranteed delivery (a.k.a. real time) applications that require the result of a computation by a certain critical deadline;
6) embedded applications, where the processor is a component of a larger system that is used to solve the (usually) dedicated application;
7) media applications such as those that decode video and audio files;
8) random software packages that desktop users would like to run on their PCs.

Each of these application areas has a very different set of characteristics. Each application area demands a different set of tradeoffs to be made in specifying the microprocessor to do the job.

Tradeoffs: Soul of Computer Architecture

- ISA-level tradeoffs

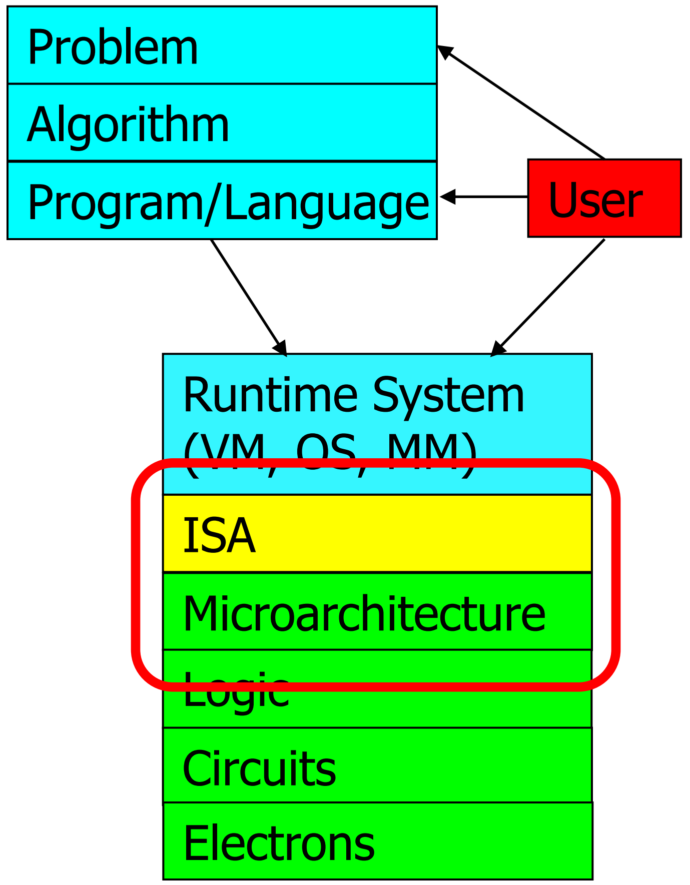
- Microarchitecture-level tradeoffs

- System and Task-level tradeoffs
  - How to divide the labor between hardware and software

- *Computer architecture is the science and art of making the appropriate trade-offs to meet a design point*
  - *Why art?*

Why Is It (Somewhat) Art?

New demands
from the top
(Look Up)



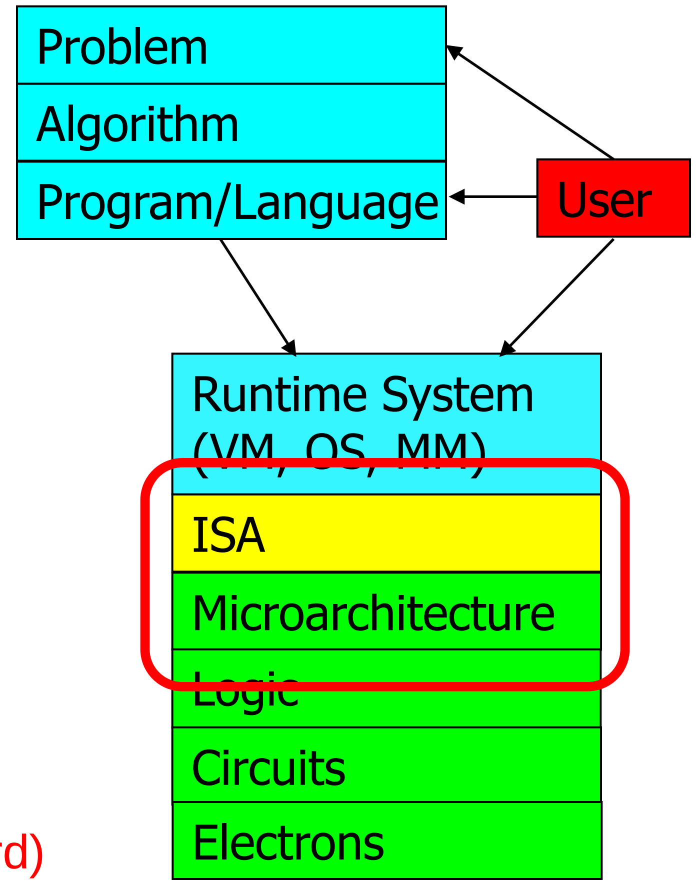
New demands and
personalities of users
(Look Up)

New issues and
capabilities
at the bottom
(Look Down)

- We do not (fully) know the future (applications, users, market)

Why Is It (Somewhat) Art?

Changing demands at the top (Look Up and Forward)



Changing demands and personalities of users (Look Up and Forward)

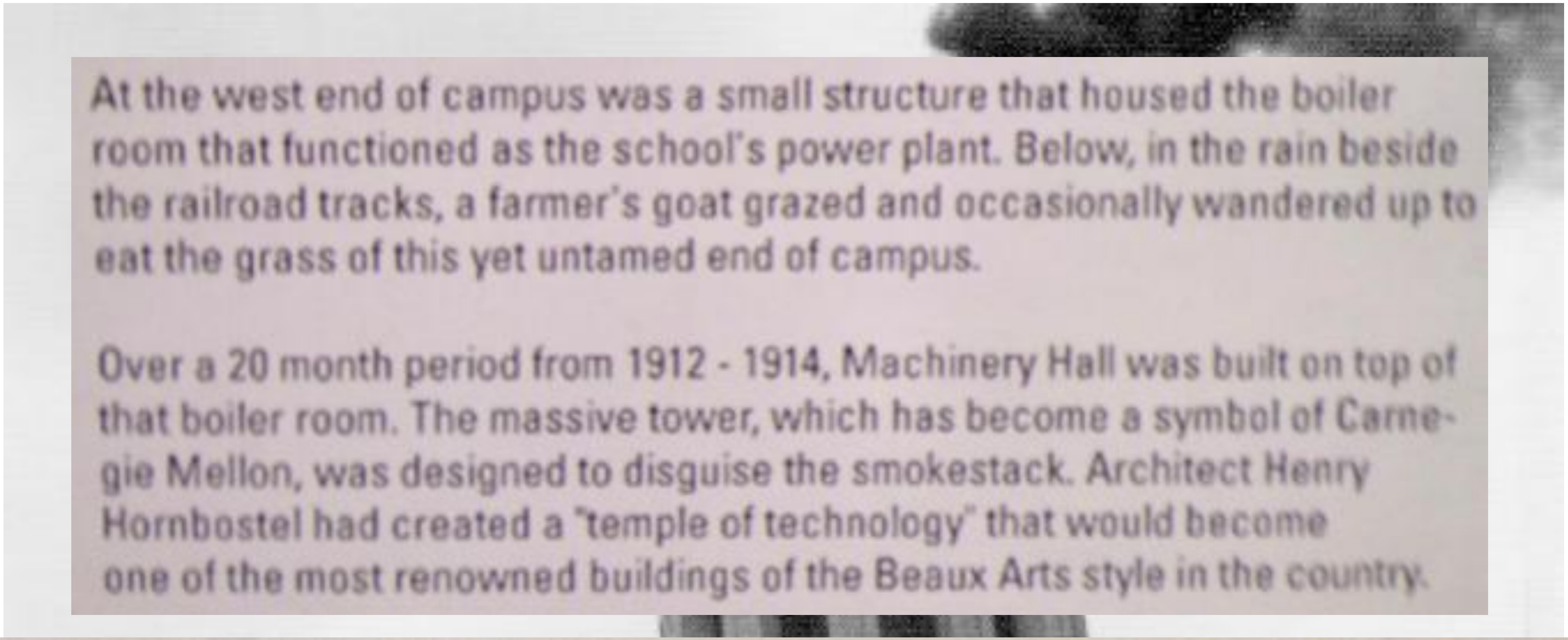
Changing issues and capabilities at the bottom (Look Down and Forward)

- And, the future is not constant (it changes)!

Analog from Macro-Architecture

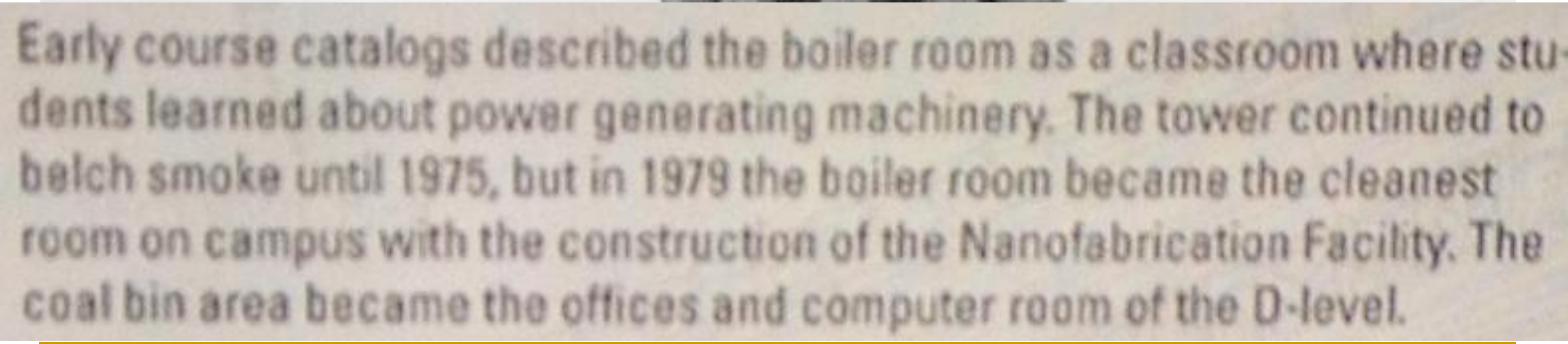
- Future is not constant in macro-architecture, either
- Example: Can a power plant boiler room be later used as a classroom?

Macro-Architecture: Boiler Room



At the west end of campus was a small structure that housed the boiler room that functioned as the school's power plant. Below, in the rain beside the railroad tracks, a farmer's goat grazed and occasionally wandered up to eat the grass of this yet untamed end of campus.

Over a 20 month period from 1912 - 1914, Machinery Hall was built on top of that boiler room. The massive tower, which has become a symbol of Carnegie Mellon, was designed to disguise the smokestack. Architect Henry Hornbostel had created a "temple of technology" that would become one of the most renowned buildings of the Beaux Arts style in the country.



Early course catalogs described the boiler room as a classroom where students learned about power generating machinery. The tower continued to belch smoke until 1975, but in 1979 the boiler room became the cleanest room on campus with the construction of the Nanofabrication Facility. The coal bin area became the offices and computer room of the D-level.

How Can We Adapt to the Future

- This is part of the task of a good computer architect

- Many options (bag of tricks)
  - Keen insight and good design
  - Good use of fundamentals and principles
    - Efficient design
    - Heterogeneity
    - Reconfigurability
    - …
  - Good use of the underlying technology
  - …

Readings for Next Time

- P&H, Chapter 4, Sections 4.1-4.4
- P&P, revised Appendix C – LC3b datapath and microprogrammed operation

- P&P Chapter 5: LC-3 ISA
- P&P, revised Appendix A – LC3b ISA