Computer Architecture

Melis Sydykbekovich Osmonov,
Candidate of Technical Sciences,
AUCA,
Applied Math and Informatics Program

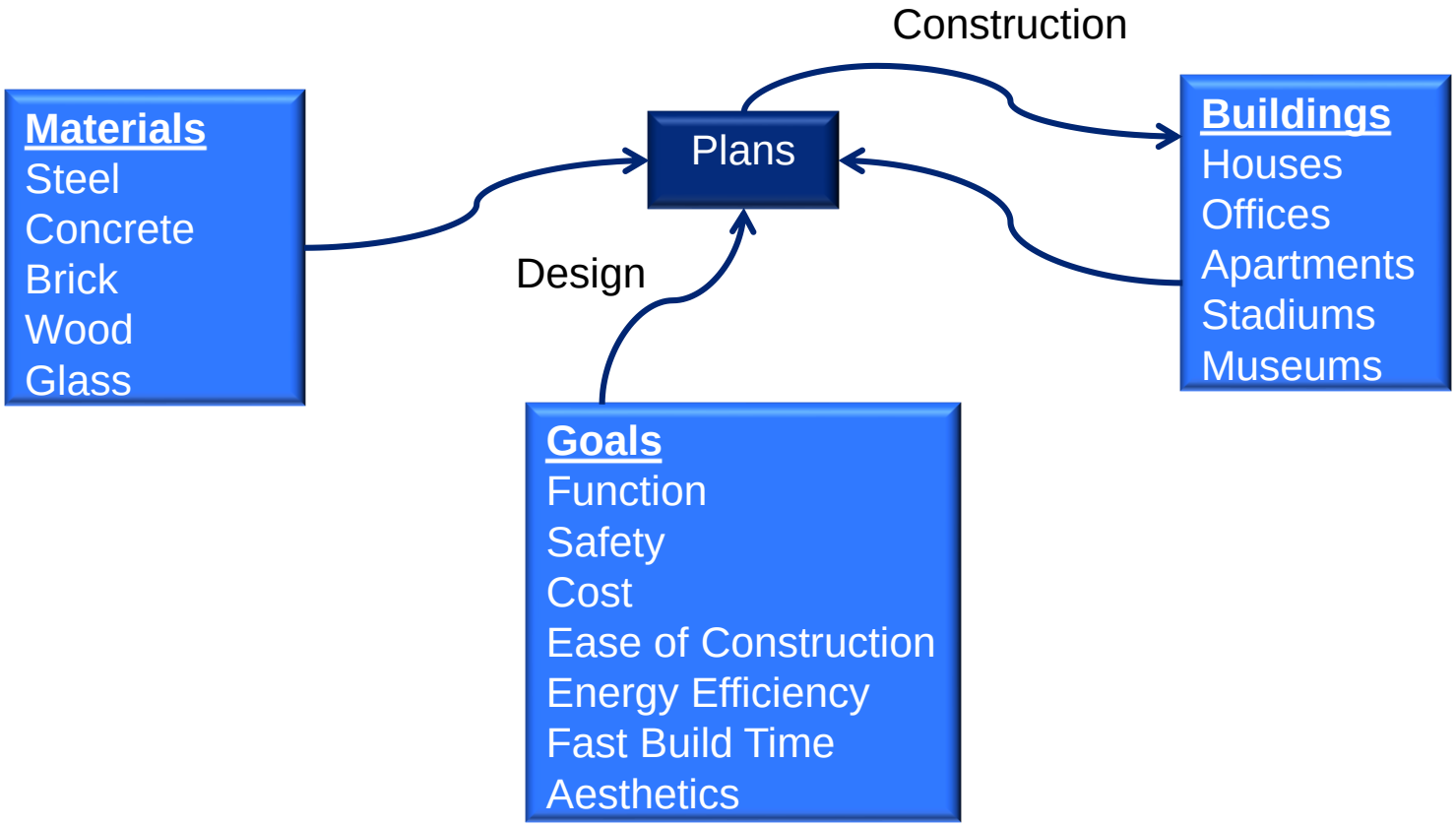
Lecture 1: Introduction

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." - WWW Computer Architecture Page
- An analogy to architecture of buildings…

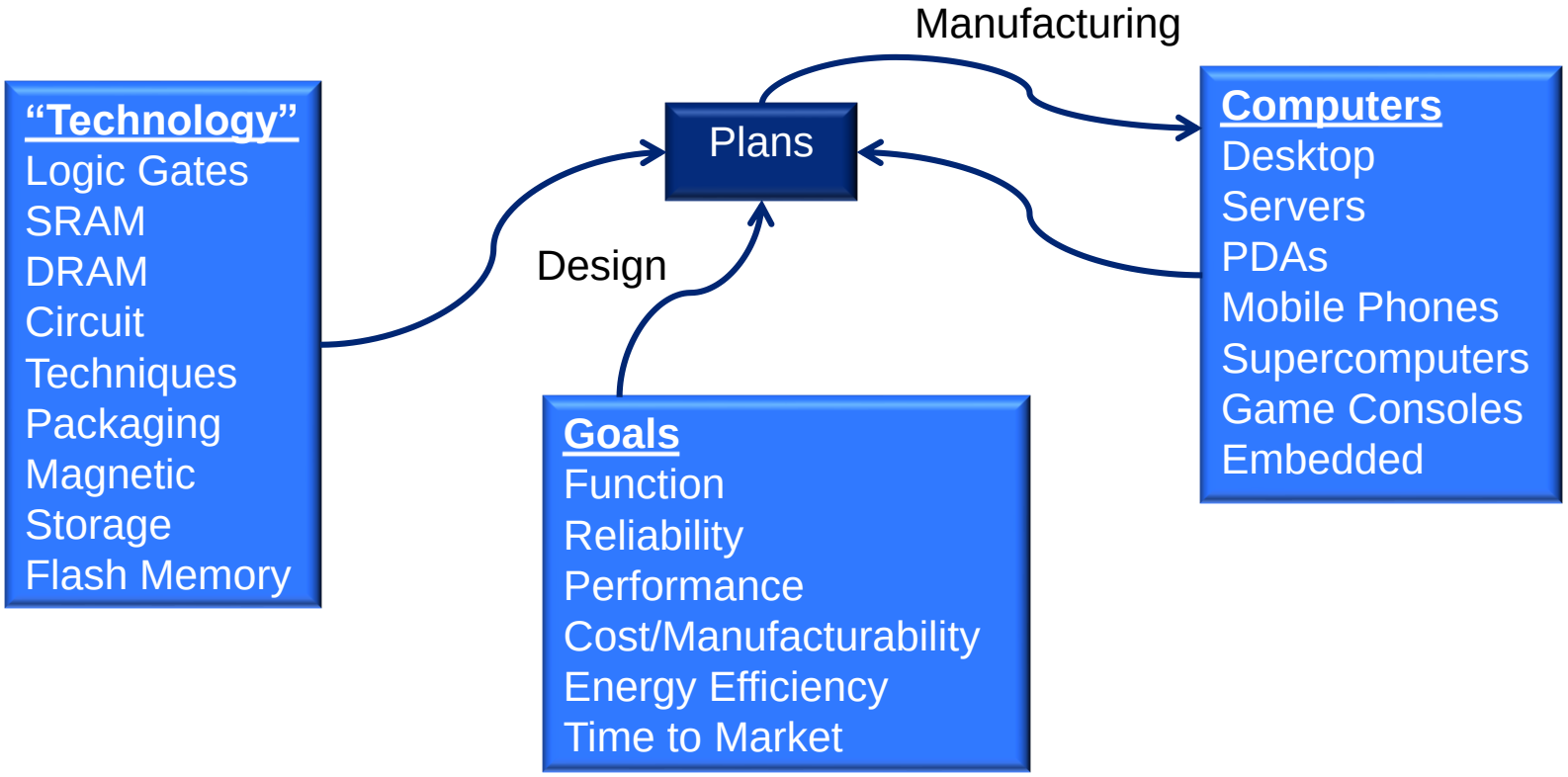
What is Architecture?

- The role of a ***building*** architect:



What is Computer Architecture?

- The role of a ***computer*** architect:



Three important differences: age (~60 years vs. thousands), rate of change, automated mass production (magnifies design)

Computer Architecture Is Different…

- Age of discipline
  - 60 years (vs. five thousand years)
- Rate of change
  - All three factors (technology, applications, goals) are changing
  - Quickly
- Automated mass production
  - Design advances magnified over millions of chips
- Boot-strapping effect
  - Better computers help design next generation

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

- In Computer Architecture

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)

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.

Conclusions

- 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

Design Goals

- **Functional**
  - Needs to be correct
  - And unlike software, difficult to update once deployed
  - What functions should it support
- **Reliable**
  - Does it **continue** to perform correctly?
  - Hard fault vs. transient fault
  - Space probe vs. desktop vs. server reliability
- **High performance**
  - “Fast” is only meaningful in the context of a set of important tasks
  - Not just “Gigahertz” – truck vs. sports car analogy
  - Impossible goal: fastest possible design for all programs

Design Goals

- **Low cost**
  - Per unit manufacturing cost (wafer cost)
  - Cost of making first chip after design (mask cost)
  - Design cost (huge design teams, why? Two reasons…)
- **Low power/energy**
  - Energy in (battery life, cost of electricity)
  - Energy out (cooling and related costs)
  - Cyclic problem, very much a problem today
- **Challenge: balancing the relative importance of these goals**
  - And the balance is constantly changing
    - No goal is absolutely important at expense of all others
  - Our focus: performance, only touch on cost, power, reliability

Shaping Force: Applications/Domains

- Another shaping force: **applications** (usage and context)
  - Applications and application domains have different requirements
    - Domain: group with similar character
  - Lead to different designs
- ***Scientific***: weather prediction, genome sequencing
  - First computing application domain: naval ballistics firing tables
  - Need: large memory, heavy-duty floating point
  - Examples: CRAY T3E, IBM BlueGene
- ***Commercial:*** database/web serving, e-commerce, Google
  - Need: data movement, high memory + I/O bandwidth
  - Examples: Sun Enterprise Server, AMD Opteron, Intel Xeon

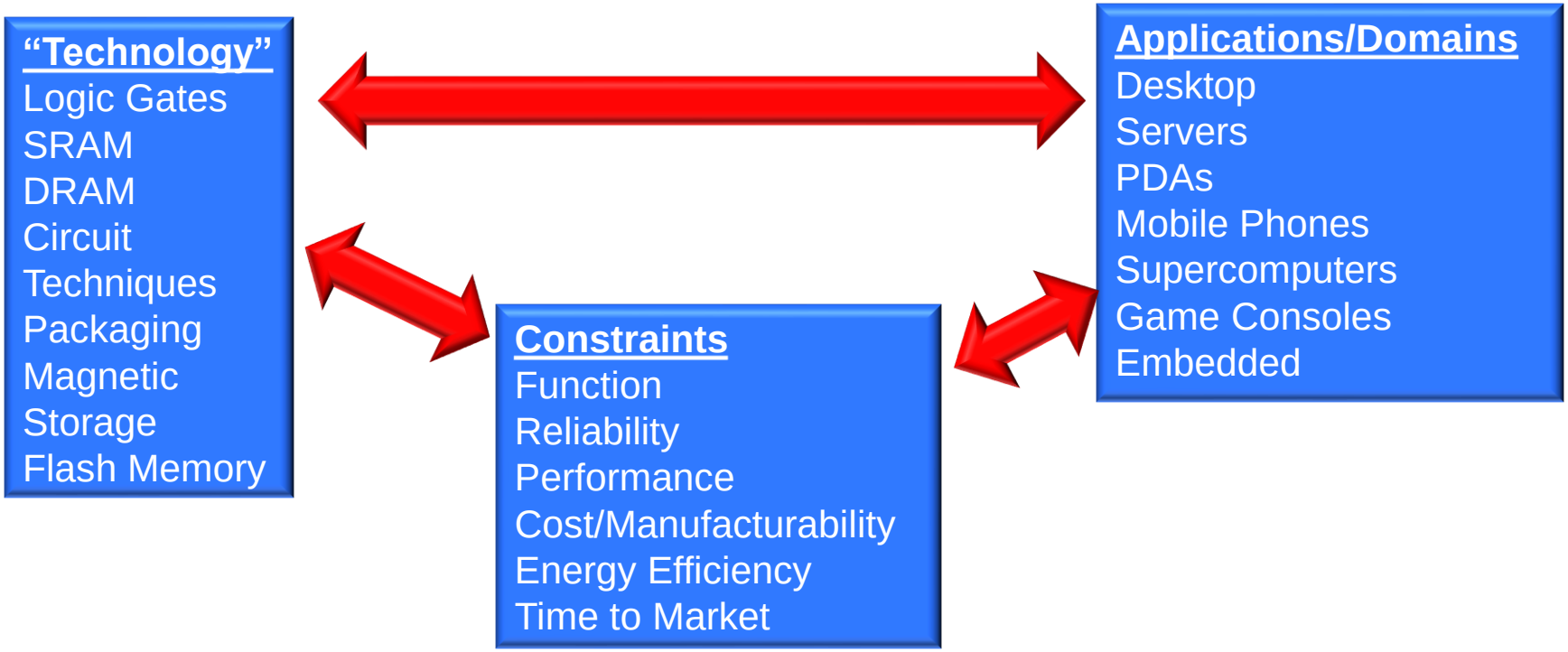
More Recent Applications/Domains

- **Desktop**: home office, multimedia, games
  - Need: integer, memory bandwidth, integrated graphics/network?
  - Examples: Intel Core 2, Core i7, AMD Athlon
- **Mobile**: laptops, mobile phones
  - Need: **low power**, integer performance, integrated wireless
  - Laptops: Intel Core 2 Mobile, Atom, AMD Turion
  - Smaller devices: ARM chips by Samsung and others, Intel Atom
- **Embedded**: microcontrollers in automobiles, door knobs
  - Need: low power, **low cost**
  - Examples: ARM chips, dedicated digital signal processors (DSPs)
  - Over 1 billion ARM cores sold in 2006 (at least one per phone)
- **Deeply Embedded**: disposable “smart dust” sensors
  - Need: extremely low power, extremely low cost

Application Specific Designs

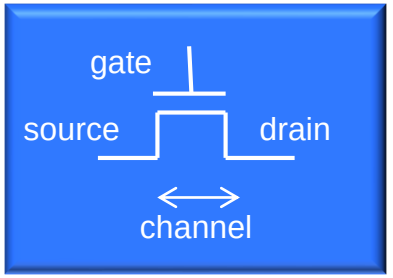
- This class is about **general-purpose CPUs**
  - Processor that can do anything, run a full OS, etc.
  - E.g., Intel Core 2, AMD Athlon, IBM Power, ARM, Intel Itanium
- In contrast to **application-specific chips**
  - Or ASICs (Application specific integrated circuits)
    - Also application-domain specific processors
  - Implement critical domain-specific functionality in hardware
    - Examples: video encoding, 3D graphics
  - General rules
    - - Hardware is less flexible than software
    - + Hardware more effective (speed, power, cost) than software
    - + Domain specific more “parallel” than general purpose
      - But general mainstream processors becoming more parallel
- Trend: from specific to general (for a specific domain)

Constant Change: Technology



- Absolute improvement, **different rates of change**
- New application domains enabled by technology advances

“Technology”



- Basic element
  - Solid-state **transistor** (i.e., electrical switch)
  - Building block of **integrated circuits (ICs)**
- What’s so great about ICs? Everything
  - + High performance, high reliability, low cost, low power
  - + Lever of mass production
- Several kinds of IC families
  - **SRAM/logic**: optimized for speed, used for processors
  - **DRAM**: optimized for density, cost, power, used for memory
  - **Flash**: non-volatile memory
  - Increasing opportunities for integrating multiple technologies
- Non-transistor storage and inter-connection technologies
  - Disk, optical storage, ethernet, fiber optics, wireless

Technology Trends

- **Moore's Law**
  - Continued (up until now, at least) transistor miniaturization
- Some technology-based ramifications
  - Absolute improvements in density, speed, power, costs
  - SRAM/logic: density: ~30% (annual), speed: ~20%
  - DRAM: density: ~60%, speed: ~4%
  - Disk: density: ~60%, speed: ~10% (non-transistor)
  - Big improvements in flash memory and network bandwidth, too
- **Changing quickly and with respect to each other!!**
  - Example: density increases faster than speed
  - Trade-offs are constantly changing
  - Re-evaluate/re-design for each technology generation

Technology Change Drives Everything

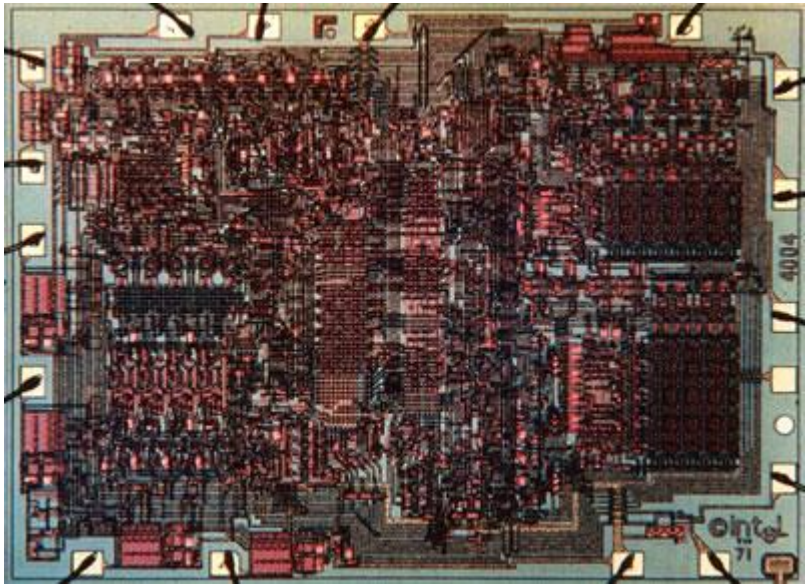
- Computers get 10X faster, smaller, cheaper every 5-6 years!
  - A 10X quantitative change is qualitative change
  - Plane is 10X faster than car, and fundamentally different travel mode
- New applications become self-sustaining market segments
  - Recent examples: mobile phones, digital cameras, mp3 players, etc.
- Low-level improvements appear as discrete high-level jumps
  - Capabilities cross thresholds, enabling new applications and uses

Revolution I: The Microprocessor

- **Microprocessor revolution**
  - One significant technology threshold was crossed in 1970s
  - Enough transistors (~25K) to put a 16-bit processor on one chip
  - Huge performance advantages: fewer slow chip-crossings
  - Even bigger cost advantages: one “stamped-out” component
- Microprocessors have allowed new market segments
  - Desktops, CD/DVD players, laptops, game consoles, set-top boxes, mobile phones, digital camera, mp3 players, GPS, automotive
- And replaced incumbents in existing segments
  - Microprocessor-based system replaced supercomputers, “mainframes”, “minicomputers”, etc.

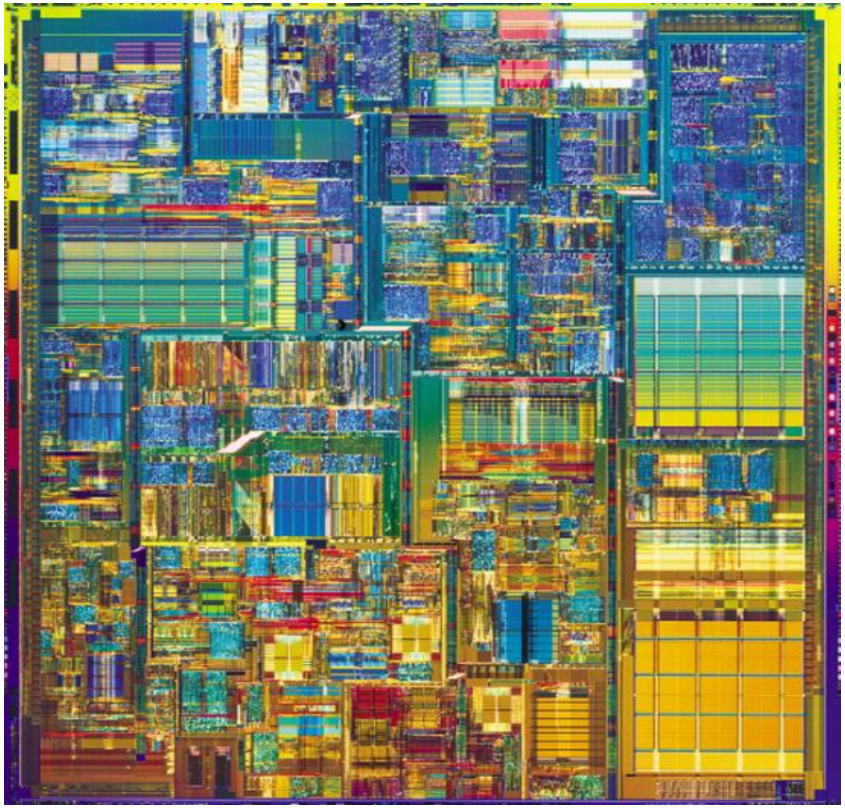
First Microprocessor

- Intel 4004 (1971)
  - Application: calculators
  - Technology: 10000 nm
  - 2300 transistors
  - 13 mm2
  - 108 KHz
  - 12 Volts
  - 4-bit data
  - Single-cycle datapath



Pinnacle of Single-Core Microprocessors

- Intel Pentium 4 (2003)
  - Application: desktop/server
  - Technology: 90nm (1/100x)
  - 55M transistors (20,000x)
  - 101 mm2 (10x)
  - 3.4 GHz (10,000x)
  - 1.2 Volts (1/10x)
  - 32/64-bit data (16x)
  - 22-stage pipelined datapath
    - Later: 31-stage pipeline
  - 3 or 4 instructions per cycle (superscalar)
  - Two levels of on-chip cache
  - data-parallel (SIMD) instructions, hyperthreading



Tracing the Microprocessor Revolution

- How were growing transistor counts used?

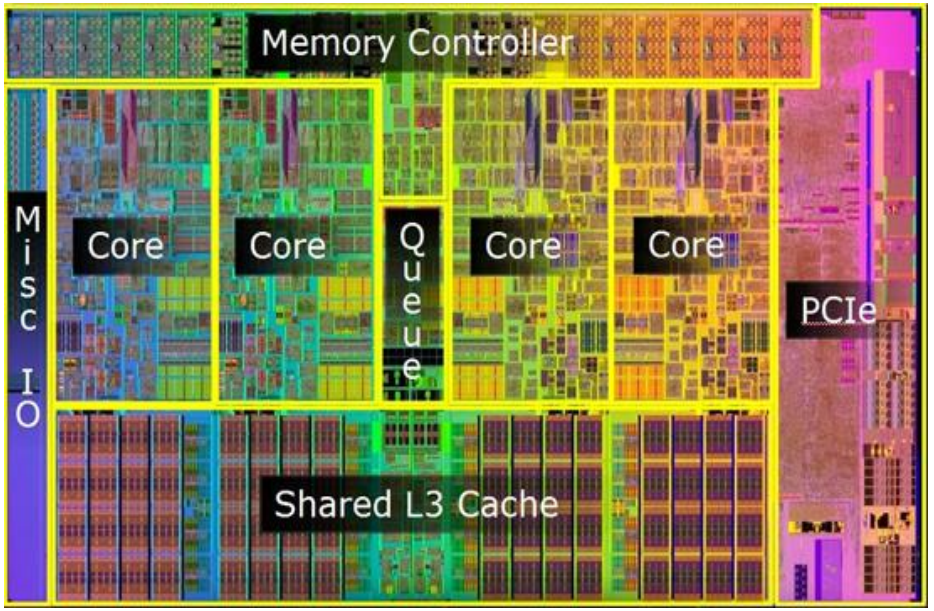
- Initially to widen the datapath
  - 4004: 4 bits ! Pentium 4: 64 bits
- … and also to add more powerful instructions
  - To amortize overhead of fetch and decode
  - To simplify programming (which was done by hand then)

Revolution II: Implicit Parallelism

- Then to **extract implicit instruction-level parallelism**
  - Hardware provides parallel resources, figures out how to use them
  - Software is oblivious
- Initially using pipelining …
  - Which also enabled increased clock frequency
- … caches …
  - Which became necessary as processor clock frequency increased
- … and integrated floating-point
- Then deeper pipelines and branch speculation
- Then multiple issue (superscalar)
- Then dynamic scheduling (out-of-order execution)
- We will talk about these things

Modern Multicore Processor

- Intel Core i7 (2009)
  - Application: desktop/server
  - Technology: 45nm (1/2x)
  - 774M transistors (12x)
  - 296 mm2 (3x)
  - 3.2 GHz to 3.6 Ghz (~1x)
  - 0.7 to 1.4 Volts (~1x)
  - 128-bit data (2x)
  - 14-stage pipelined datapath (0.5x)
  - 4 instructions per cycle (~1x)
  - Three levels of on-chip cache
  - data-parallel (SIMD) instructions, hyperthreading
  - **Four-core multicore** (4x)



Revolution III: Explicit Parallelism

- Then to support **explicit data & thread level parallelism**
  - Hardware provides parallel resources, software specifies usage
  - Why? Diminishing returns on instruction-level-parallelism
- First using (subword) vector instructions…, Intel's SSE
  - One instruction does 4 parallel multiplies
- … and general support for multi-threaded programs
  - Coherent caches, hardware synchronization primitives
- Then using support for multiple concurrent threads on chip
  - First with single-core multi-threading, now with multi-core
- Graphics processing units (GPUs) are highly parallel
  - Converging with general-purpose processors (CPUs)?

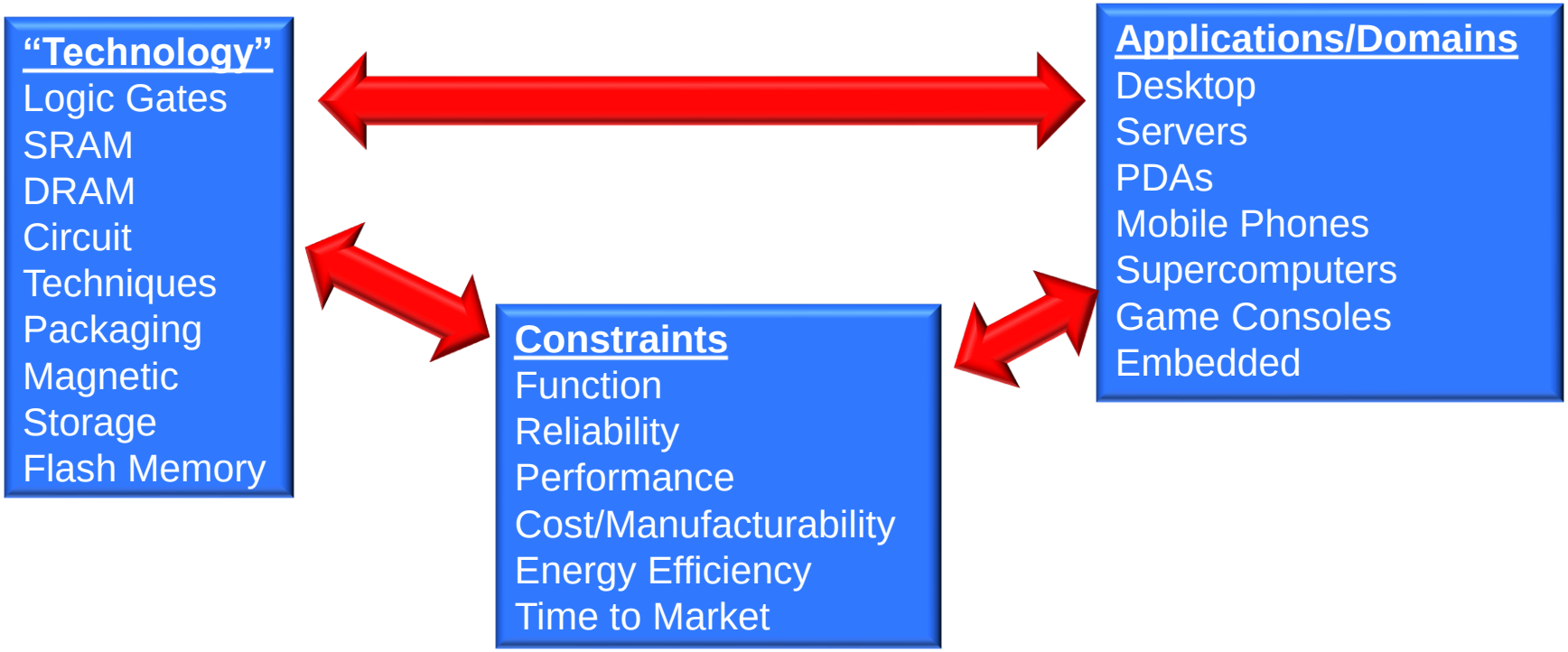
To ponder…

- Is this decade's "**multicore revolution**" comparable to the original "**microprocessor revolution**"?

Technology Disruptions

- Classic examples:
  - The transistor
  - Microprocessor
- More recent examples:
  - Multicore processors
  - Flash-based solid-state storage
- Near-term potentially disruptive technologies:
  - Phase-change memory (non-volatile memory)
  - Chip stacking (also called 3D die stacking)
- Disruptive “end-of-scaling”
  - “If something can’t go on forever, it must stop eventually”
  - Can we continue to shrink transistors for ever?
  - Even if more transistors, not getting as energy efficient as fast

Recap: Constant Change



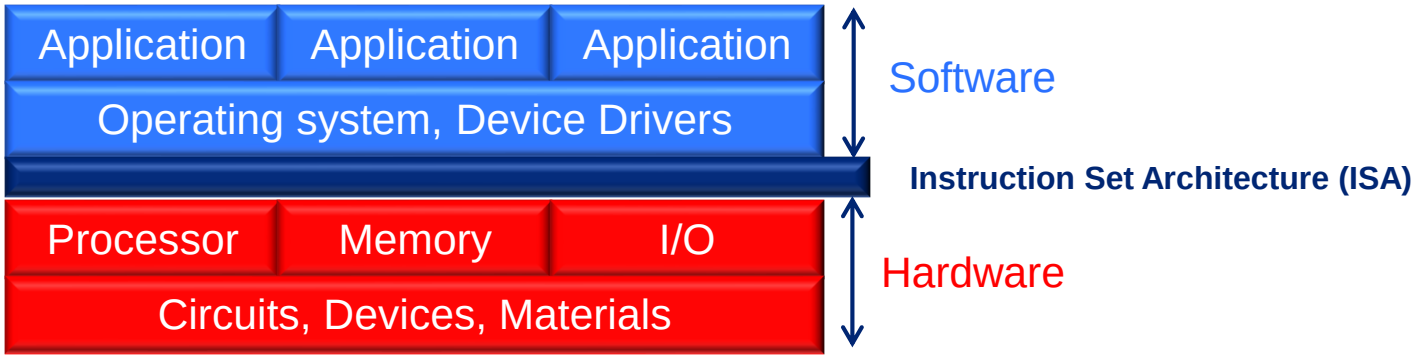
Managing This Mess

- Architect must consider all factors
  - Goals/constraints, applications, implementation technology
- Questions
  - How to deal with all of these inputs?
  - How to manage changes?
- Answers
  - Accrued institutional knowledge
  - Experience
  - Discipline: clearly defined end state, keep your eyes on the ball
  - **Abstraction and layering**

Pervasive Idea: Abstraction and Layering

- **Abstraction**: only way of dealing with complex systems
  - Divide world into objects, each with an…
    - **Interface**: knobs, behaviors, knobs → behaviors
    - **Implementation**: "black box" (ignorance+apathy)
  - Only specialists deal with implementation, rest of us with interface
  - Example: car, only mechanics know how implementation works
- **Layering**: abstraction discipline makes life even simpler
  - Divide objects in system into layers, layer n objects…
    - Implemented using interfaces of layer n –1
    - Don't need to know interfaces of layer n –2 (sometimes helps)
- **Inertia**: a dark side of layering
  - Layer interfaces become entrenched over time ("standards")
  - – Very difficult to change even if benefit is clear
- **Opacity**: hard to reason about performance across layers

Abstraction, Layering, and Computers



- **Computer architecture**
  - Definition of **ISA** to facilitate implementation of software layers
- This course mostly about **computer organization**
  - Design **Processor**, **Memory**, I/O to implement **ISA**

Why Study Computer Architecture?

- **Understand where computers are going**
  - Future capabilities drive the (computing) world
  - Real world-impact: no computer architecture → no computers!
- **Understand high-level design concepts**
  - The best architects understand all the levels
    - Devices, circuits, architecture, compiler, applications
- **Understand computer performance**
  - Writing well-tuned (fast) software requires knowledge of hardware
- **Get a (design or research) hardware job**
  - Intel, AMD, IBM, ARM, Motorola, Sun/Oracle, NVIDIA, Samsung
- **Get a (design or research) software job**
  - Best software designers understand hardware
  - Need to understand hardware to write fast software

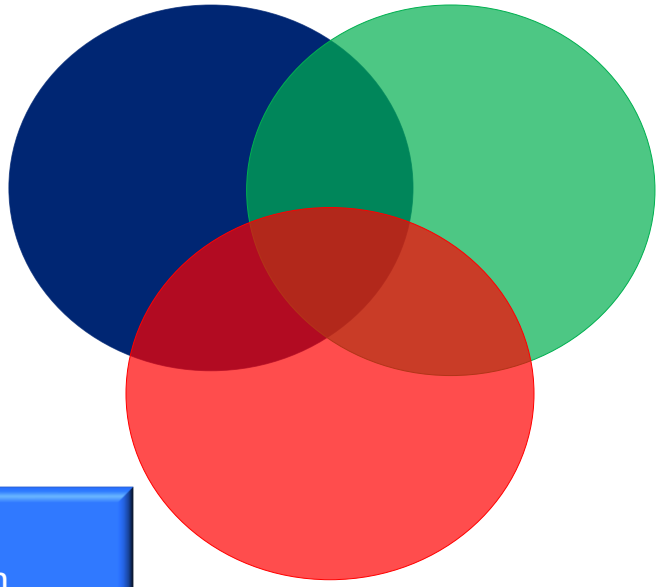
Course Goals

- **See the "big ideas" in computer architecture**
  - Pipelining, parallelism, caching, locality, abstraction, etc.
- Understanding computer performance and metrics
  - Experimental evaluation/analysis ("science" in computer science)
  - Understanding quantitative data and experiments

Computer Science as an Estuary

Where does architecture fit into computer science?
Engineering, some Science

Engineering
Design
Handling complexity
Real-world impact
Examples: Internet, microprocessor



Science
Experiments
Hypothesis
Examples:
Internet behavior,
Protein-folding supercomputer
Human/computer interaction

Mathematics
Limits of computation
Algorithms & analysis
Cryptography
Logic
Proofs of correctness

Other Issues
Public policy, ethics, law, security

Course Topics

- Evaluation metrics and trends
- ISAs (instruction set architectures)
- Datapaths and pipelining
- Memory hierarchies & virtual memory

- Parallelism
  - Instruction: multiple issue, dynamic scheduling, speculation
  - Data: vectors and streams
  - Thread: cache coherence and synchronization, multicore

Prerequisites

- COM 333 Digital Integrated Circuit Design
  - Basic digital logic: gates, boolean functions, latches
  - Binary arithmetic: adders, hardware mul/div, floating-point
  - Basic datapath: ALU, register file, memory interface, muxes
  - Basic control: single-cycle control, microcode
  - Basic caches, pipelining (will review these)

- … also, programming experience

Resources

- Readings
  - J.L. Hennessy, D.A. Patterson. Computer architecture: a quantitative approach. 4th ed. - Morgan Kaufmann Publishers, 2007.
  - J.L. Hennessy, D.A. Patterson. Computer Organization and Design: *The Hardware/Software Interface*, ARM® Edition. - Morgan Kaufmann is an imprint of Elsevier, 2017
  - J.L. Hennessy, D.A. Patterson. Computer Organization and Design: *The Hardware/Software Interface*, Fifth Edition. - Morgan Kaufmann is an imprint of Elsevier, 2014.
  - Jean-Loup Baer. Microprocessor Architecture: From Simple Pipelines to Chip Multiprocessors. - Cambridge University Press, 2010.
  - Research papers
- Free resources
  - ACM digital library: http://www.acm.org/dl/
  - Computer architecture page: http://www.cs.wisc.edu/~arch/www/
- Google Classroom.

The lecture is over

Thanks for attention!!!