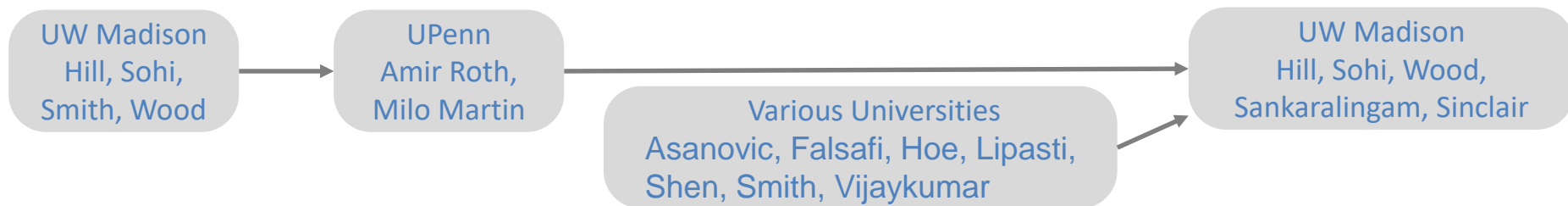


UCLA CS 251a: Advanced Computer Architecture

Lecture 1. Introduction

Tony Nowatzki

Slide History/Attribution Diagram:



Welcome to Week 0

- Enrollment:
 - If you're not enrolled yet, it should be fine. I have PTEs left and I expect drops.
- About Me:
 - Prof. @UCLA since January 2017
 - Dieing to teach this course for the last 2 years.
 - Research focuses on accelerators/specialization/heterogeneous computers (intellectual curiosity focused on generality)
- Teaching Style: non-authoritarian?
 - Much of this course contains architecture "wisdom" – everything is open for debate.

Warmup 1

- Which of these is faster?

Version 1

```
void copyij(int src[2048][2048],
            int dst[2048][2048])
{
    int i,j;
    for (i = 0; i < 2048; i++)
        for (j = 0; j < 2048; j++)
            dst[i][j] = src[i][j];
}
```

Version 2

```
void copyji(int src[2048][2048],
            int dst[2048][2048])
{
    int i,j;
    for (j = 0; j < 2048; j++)
        for (i = 0; i < 2048; i++)
            dst[i][j] = src[i][j];
}
```

Warmup 2

- Which of these is faster?

Version 1

```
for (unsigned c = 0; c < n; ++c)
    data[c] = std::rand() % 256;

//std::sort(data, data + n);

// BEGIN TIMER
for (int i = 0; i < 100000; ++i) {
    for (int c = 0; c < n; ++c) {
        if (data[c] >= 128)
            sum += data[c]*2;
    }
}
// END TIMER
```

Version 2

```
for (unsigned c = 0; c < n; ++c)
    data[c] = std::rand() % 256;

std::sort(data, data + n);

// BEGIN TIMER
for (int i = 0; i < 100000; ++i) {
    for (int c = 0; c < n; ++c) {
        if (data[c] >= 128)
            sum += data[c]*2;
    }
}
// END TIMER
```

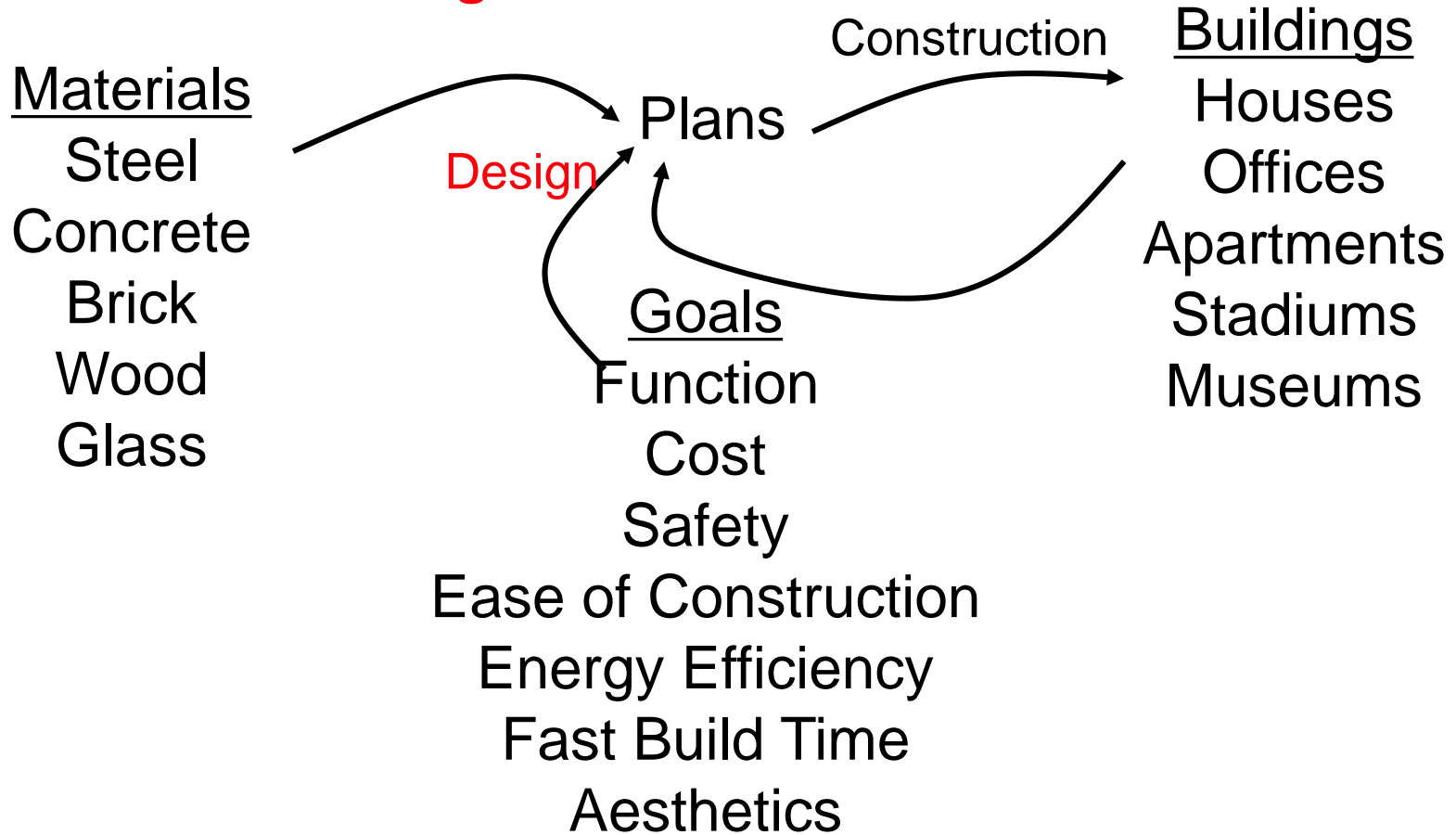
What is computer architecture?

Example Architectures



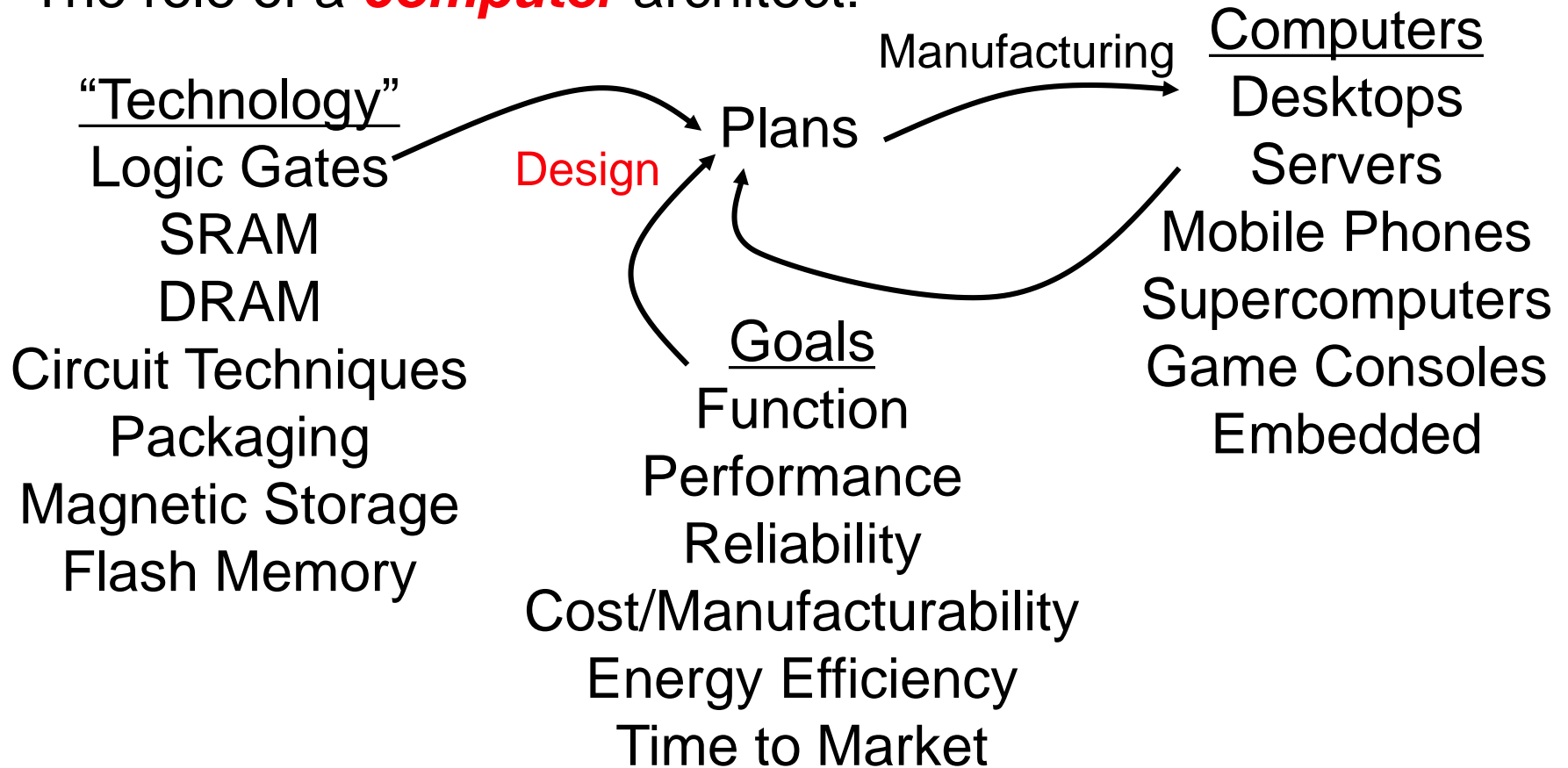
What is ~~Computer~~ Architecture?

The role of a **building** architect:



What is Computer Architecture?

The role of a **computer** architect:



Analogy Breakdown

- **Age of discipline**
 - 60 years (vs. five thousand years)
- **Fungibility**
 - No intrinsic value to a particular instance (or aesthetic value)
 - Don't care where my program lives
- **Durability**
 - Every two years you throw away your personal out-of-order multicore chip and buy a new one.
 - Compute devices will anyways become obsolete due to technology
- **Manufacturing Tradeoffs**
 - Nth+1 chip costs $\sim \$0$
- **Boost-strapping**
 - Better computers help design next generation

Design Goals / Constraints

- **Functional**

- Needs to be correct
 - And unlike software, difficult to update once deployed
- Security: Should provide guarantees to software

- **Reliable**

- Does it ***continue*** to perform correctly?
- Hard fault vs transient fault
- Space satellites vs desktop vs server reliability

- **High performance**

- Not just “Gigahertz” – truck vs sports car analogy

- **Generality**

- “Fast” is only meaningful in the context of a set of important tasks
- Impossible goal: fastest possible design for all programs

Design Goals / Constraints

- **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

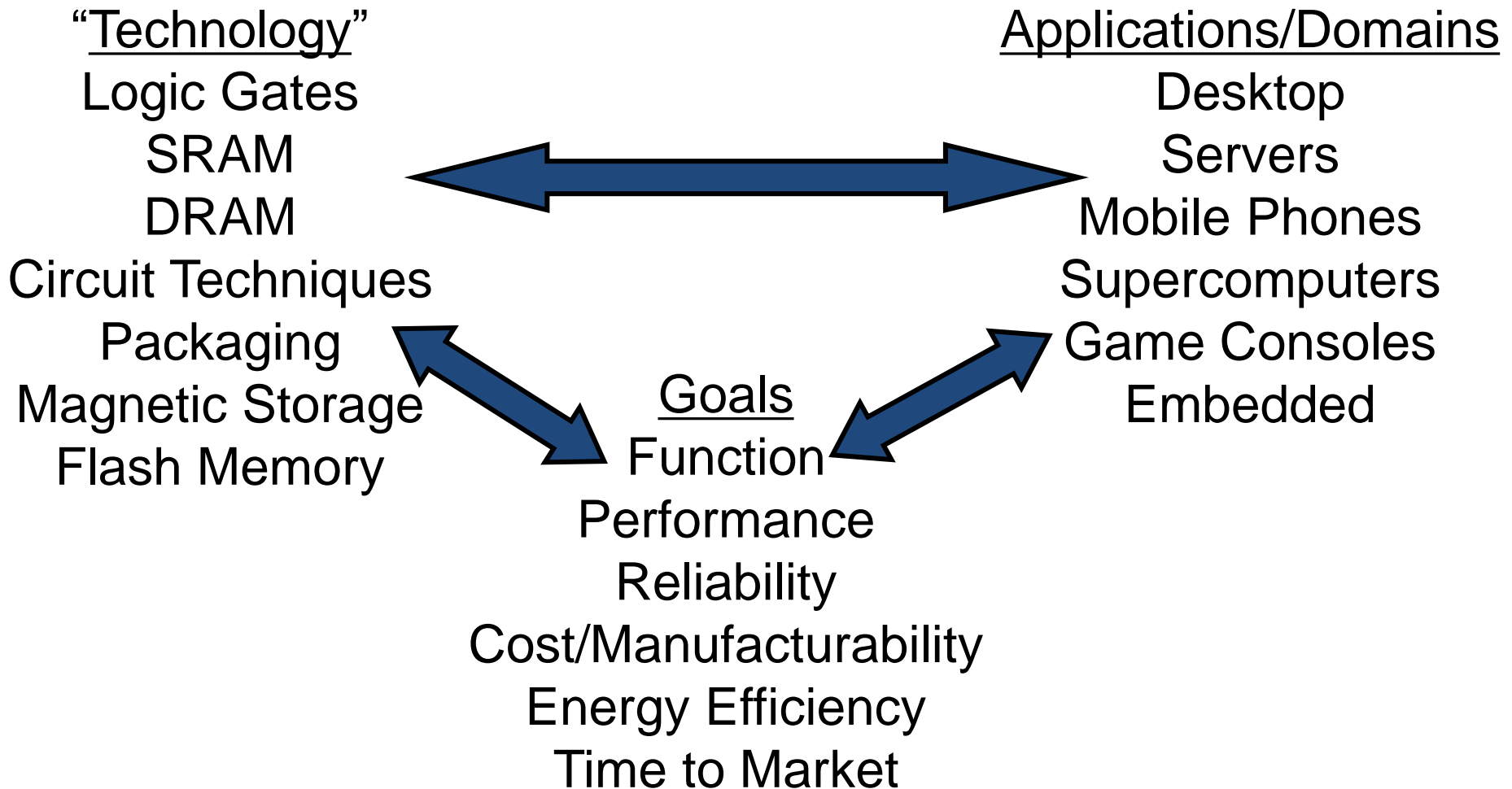
- **Challenge: balancing the relative importance of these goals**

- 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

Application Specific Designs


- This class is (mostly) about **general-purpose CPUs**
 - Processor that can do anything, run a full OS, etc.
 - E.g., Intel Core i7, 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

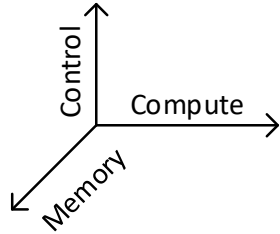


- Absolute improvement, **different rates of change**
- New application domains enabled by technology advances

Layers of Abstraction

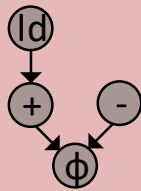
- Architects need to understand computers at many levels
 - Applications
 - Operating Systems
 - Compilers
 - Instruction Set Architecture
 - Microarchitecture
 - Circuits
 - Technology
- 
- Good architects are “Jacks of most trades”

Layers of Abstraction



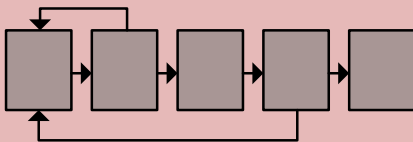
Applications

sub
ld
add
br

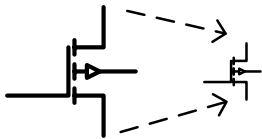


**ISA: Hardware/
Software Interface**

**Architects'
Domain**



Microarchitecture



Technology

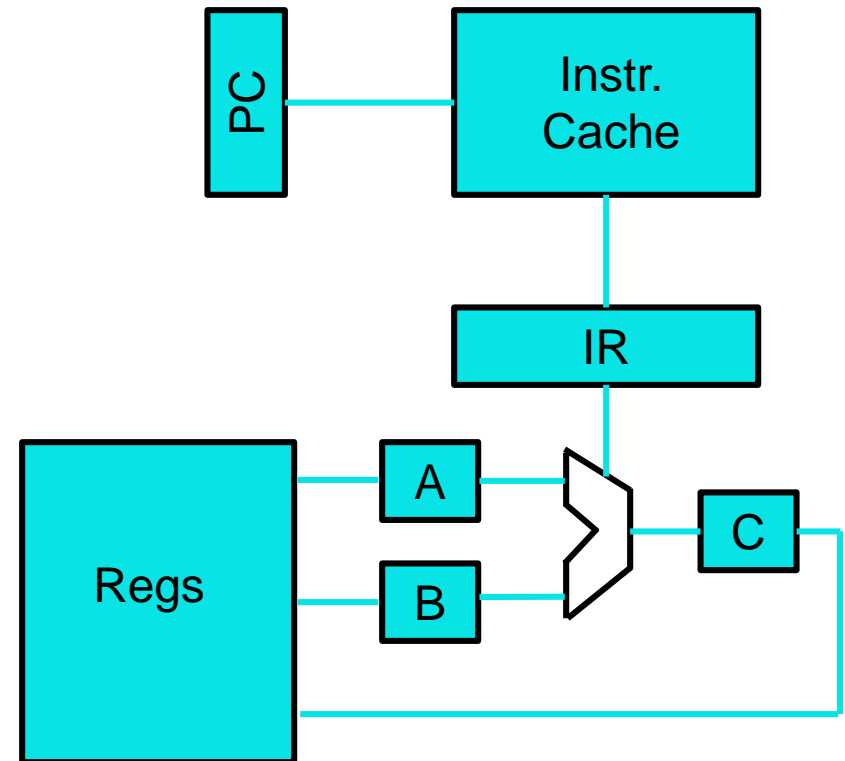
Instruction Set Architecture

- Hardware/Software interface
 - Software impact
 - support OS functions
 - restartable instructions
 - memory relocation and protection
 - a good compiler target
 - simple
 - orthogonal
 - Dense
 - Improve memory performance
 - Hardware impact
 - admits efficient implementation
 - across generations
 - admits parallel implementation
 - no 'serial' bottlenecks
- Abstraction without interpretation



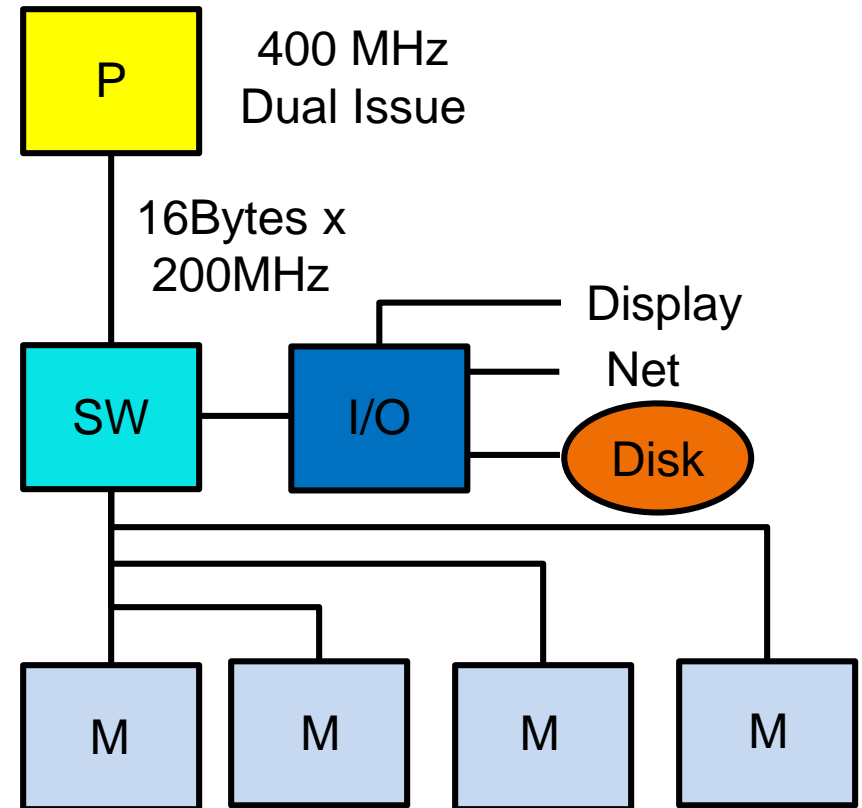
Microarchitecture

- Register-transfer-level (RTL) design
- Implement instruction set
- Exploit capabilities of technology
 - Locality and concurrency
- Iterative process
 - Generate proposed architecture
 - Estimate cost
 - Measure performance
- Still emphasis is on overcoming sequential nature of programs
 - Deep pipelining
 - Multiple issue
 - Dynamic scheduling
 - Branch prediction/speculation



System-Level Design

- Design at the level of processors, memories, and interconnect
- More important to application performance, cost, and power than CPU design
- Feeds and speeds
 - Constrained by IC pin count, module pin count, and signaling rates
- System balance
 - For a particular application
- Driven by
 - Performance/cost gains
 - Available components (cost/perf)
 - Technology constraints



Large-system example



- Google Project 02: Oregon/Washington border
- Cheap electricity: Columbia river
- Cheap cooling: Columbia river
- Cheap B/W: surplus optic network from tech-boom era

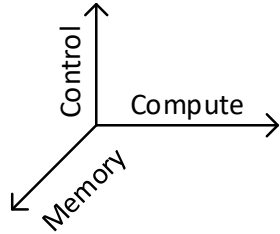
Rapid Change

Exciting: perhaps the fastest moving field ... ever

Processors vs. cars

- 1985: processors = 1 MIPS, cars = 60 MPH
- 2000: processors = 500 MIPS, cars = 30,000 MPH?

Layers of Abstraction

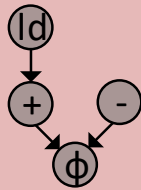


Applications

**Major Driver
Going forward?**

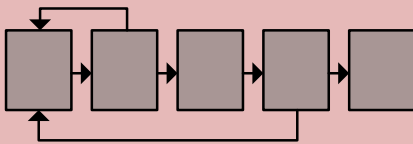


sub
ld
add
br

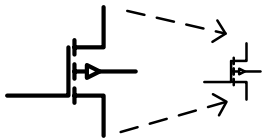


**ISA: Hardware/
Software Interface**

Architects'
Domain



Microarchitecture



Technology

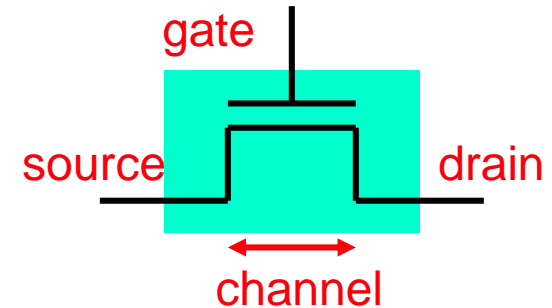
**Major Driver
for 50+ years**



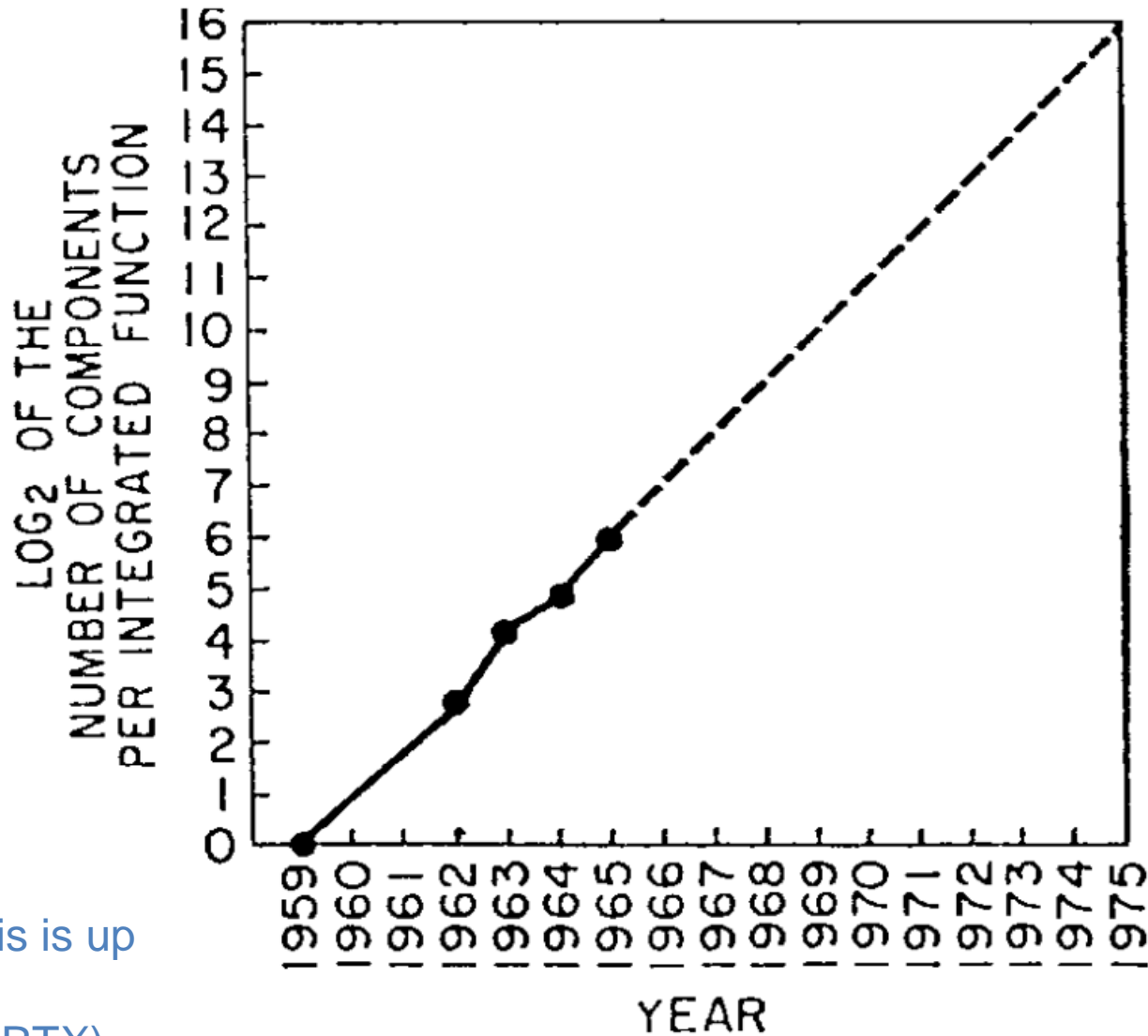
Technology as a Driver

“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**: optimized for density, cost (used for storage)
 - Increasing opportunities for integrating multiple technologies
- Non-transistor storage and inter-connection technologies
 - Disk, optical storage, ethernet, fiber optics, wireless



Moore's Law -- 1965



Today this is up
to 2^{34}
(NVIDIA RTX)

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

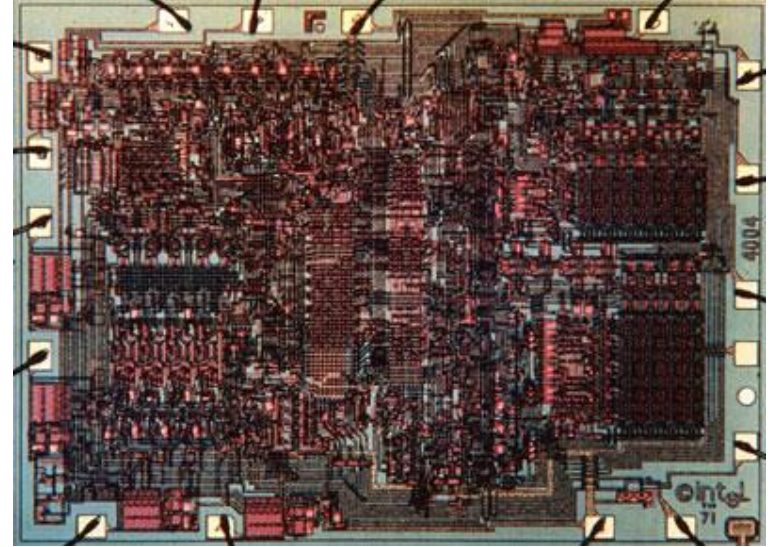
- Computers get 10x faster, smaller, cheaper every 5-6 years!
 - A 10x quantitative change is qualitative change
 - Plane is 10x faster than car, and fundamentally different travel mode
- New applications become self-sustaining market segments
 - Recent examples: mobile phones, digital cameras, mp3 players, 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 ($\sim 25K$) 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.

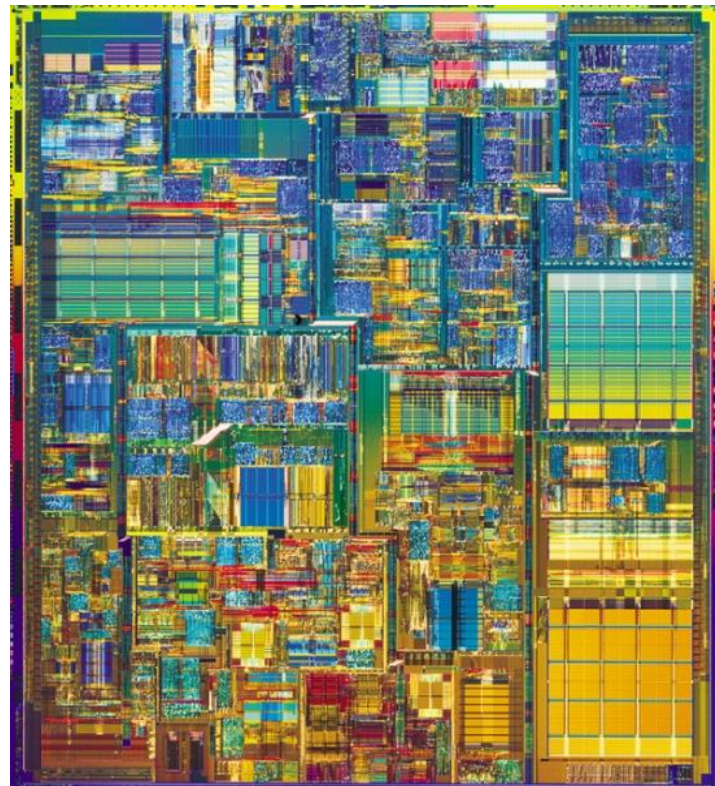
First Microprocessor

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



Not-so-recent Microprocessors

- Intel Pentium4 (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
 - 3 instructions per cycle (superscalar)
 - Two levels of on-chip cache
 - data-parallel vector (SIMD) instructions, hyperthreading
- Pinnacle of single-core microprocessors



By the end of the course, this will make sense!

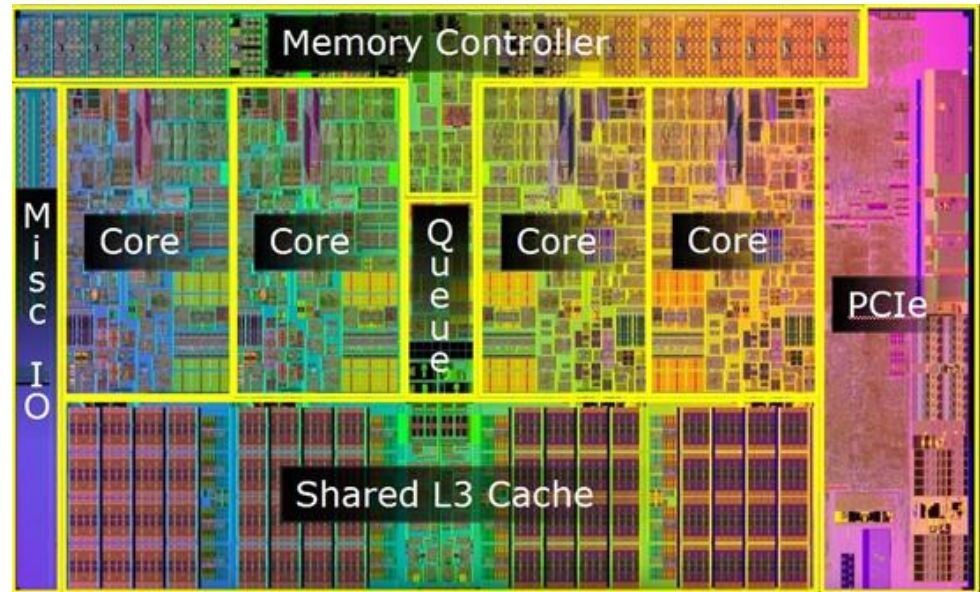
- Pentium 4 specifications:
 - Technology:
 - 55M transistors, 0.90 μm CMOS, 101 mm^2 , 3.4 GHz, 1.2 V
 - Performance
 - 1705 SPECint, 2200 SPECfp
 - ISA
 - X86+MMX/SSE/SSE2/SSE3 (X86 translated to RISC uops inside)
 - Memory hierarchy
 - 64KB 2-way insn trace cache, 16KB D\$, 512KB–2MB L2
 - MESI-protocol coherence controller, processor consistency
 - Pipeline
 - 22-stages, dynamic scheduling/load speculation, MIPS renaming
 - 1K-entry BTB, 8Kb hybrid direction predictor, 16-entry RAS
 - 2-way hyper-threading

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 instructions per cycle (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 vector (SIMD) instructions, hyperthreading
 - **Four-core multicore** (4x)



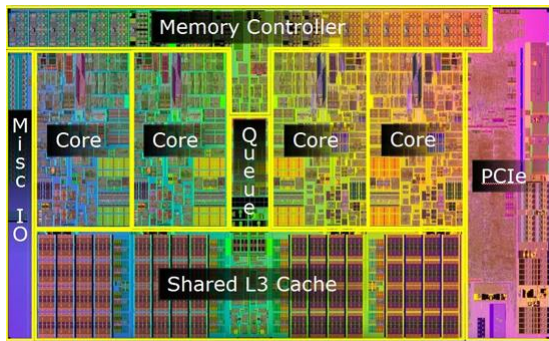
Revolution III: Explicit Parallelism

- 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 four 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

A Plethora of Processors

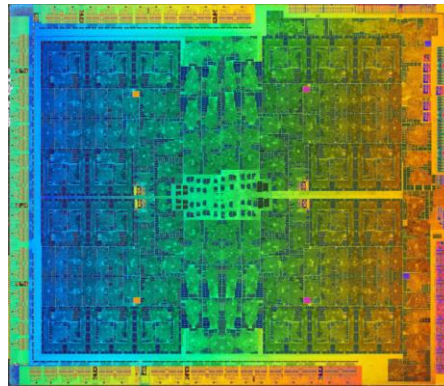
CPU

("Ordinary" Apps)

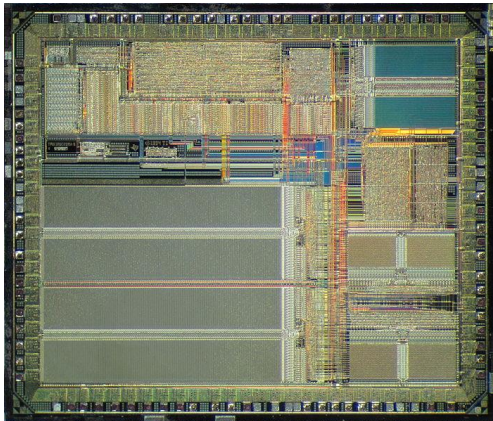


GPU

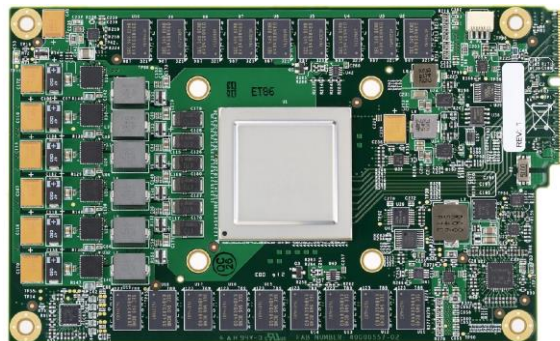
(Graphics, Data-proc.)



DSP (signal proc.)



Google TPU:
(Deep Learning)



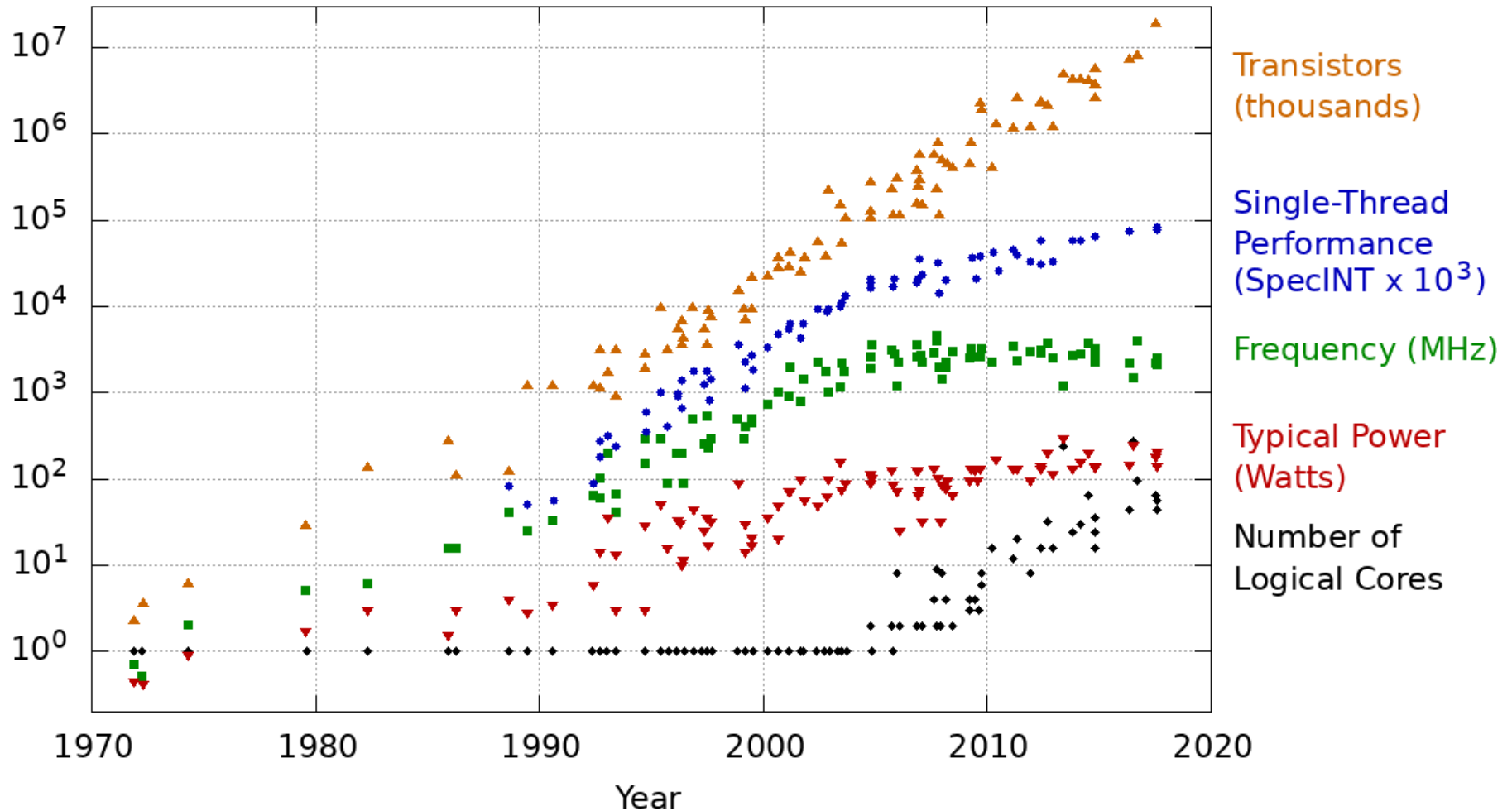
Machine Learning
And Computer
Vision:

Alibaba AI Chip,
Habana, NextVPU,
Syntiant, AlphaC,
Nvidia Turing, DeePhi,
Baidu Kunlun,
Tachyum, Hailo, Kortiq,
Innogrit, ThinkSilicon,
Cambricon, Microsoft
Brainwave, Graphcore,
Qualcomm AI Engine,
Videantis, Arm "Project
Trillium"

Revolution IV: Specialization

- Combine implicit/explicit parallelism with a focus on a particular domain
 - Scope can be very different: GPGPUs are quite broad, while TPUs are not...
- Tradeoff the overheads of supporting “general purpose” workloads for efficiency on a smaller set of workloads.
- But why is this happening now?
 - (dark silicon)

42 Years of Microprocessor Trend Data



Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten
New plot and data collected for 2010-2017 by K. Rupp

Applications as a Driver

Applications Views

- Many ways to view distinction between application settings:
 - Domain-centric view:
 - Set applications from a similar background
 - Deployment-centric view
 - Where the machine is deployed affects the set of applications
 - Property-centric view:
 - Set of applications with different properties

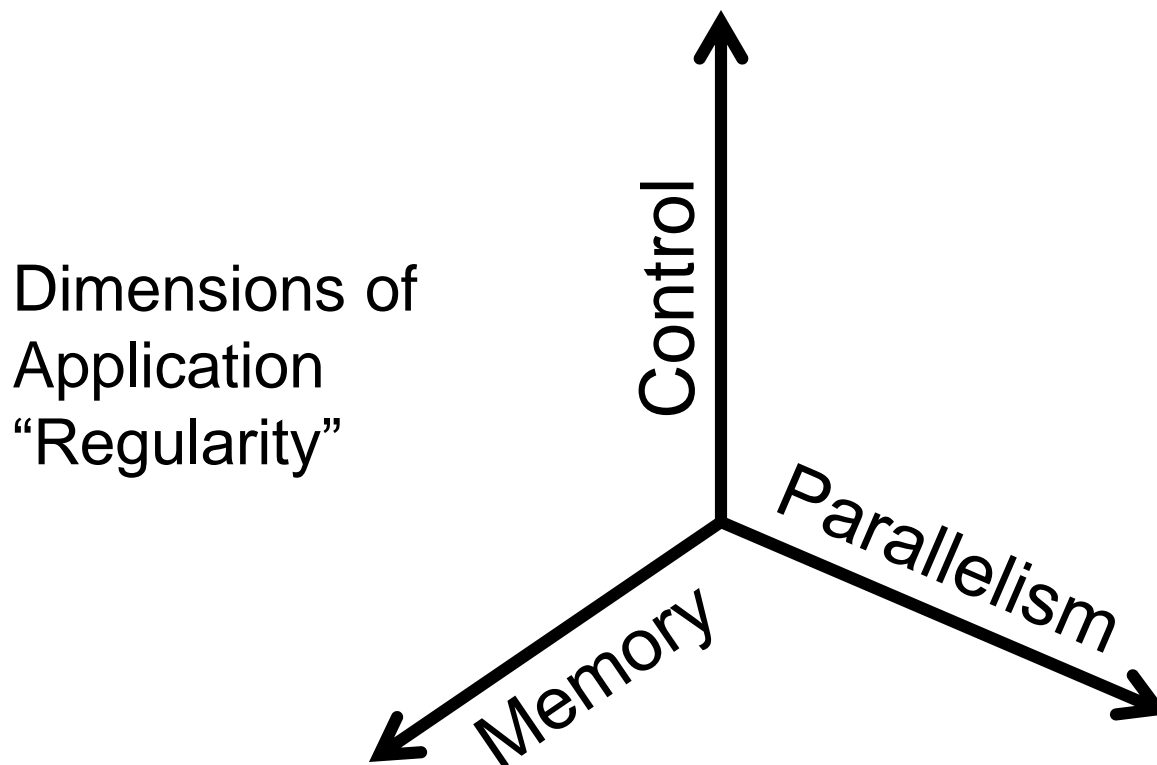
Domain-centric View

- **Scientific:** weather prediction, genome sequencing
 - First computing application domain: naval ballistics firing table
 - Need: large memory, heavy-duty floating point
 - Examples: CRAY T3E, IBM BlueGene
- **Commercial:** database/web serving, e-commerce
 - Need: data movement, high memory + I/O bandwidth
 - Examples: Sun Enterprise Server, AMD Opteron, Intel Xeon, IBM Power 7
- **More recent example:** Deep learning
 - Need: Lots of dense linear algebra :)

Deployment Centric View

- **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
 - Over 1 billion ARM cores sold in 2006 (at least one per phone)
- **Embedded**: microcontrollers in automobiles, door knobs
 - Need: low power, **low cost**
 - Examples: ARM chips, dedicated digital signal processors (DSPs)
- **Deeply Embedded**: disposable “smart dust” sensors
 - Need: extremely low power, extremely low cost

Property-centric View



More regularity \rightarrow Less dependences

Less dependences \rightarrow Easier exploitation
(h/w or s/w)

Control Regularity

Increasing “Irregularity”

- No Control (or non critical)
- Data-Independent
- Data-Dependent, Predictable
- Data-Dependent, Unpredictable

(also, indirect branches)

```
for i
  ... = a[i]
```

```
for i
  if(i%2)
    ... = a[i]
```

```
for i
  if(if(age[i]>0))
    ... = a[i]
```

```
for i
  if(age[i]>22)
    ... = a[i]
```

Memory Regularity

Increasing “Irregularity” 

- Data dependence

```
for i=0 to n  
  ... = a[i]
```

```
while(node)  
  ... = *node  
  node = node->next
```

- Alias freedom

```
for i=0 to n  
  ... = a[i]
```

```
for i=0 to n ... =  
  a[index[i]]++
```

- Locality

```
for i=0 to n  
  for j=0 to n  
    ... = a[j]
```

spatial & temporal

```
for i=0 to n  
  for j=0 to n  
    ... = a[i][j]
```

just spatial

```
for i=0 to n  
  for j=0 to n  
    ... = a[j][i]
```

neither

Ponder this: why does low-locality introduce dependences?

Parallelism Regularity

- Types of Parallelism
 - Instruction-level Parallelism (ILP): Nearby instructions running together
 - Memory-level Parallelism (MLP): Same as ILP, but specifically cache misses.
 - Thread-level Parallelism (TLP): Independent threads (at least to some extent) running simultaneously.
 - Task-level Parallelism: Same as above, but implies dependences.
 - Data-level Parallelism (DLP): Do same thing to many pieces of data.
- Which is the most regular:
 - DLP: more regular
 - ILP: less regular
 - TLP: least regular
- Dimensions of Regularity
 - Granularity: Fine vs Coarse Grain
 - Data-dependence: Static vs Dynamic
- Complexity Ahead:
 - DLP implies ILP... (but not other way around)
 - DLP implies TLP ... (but not other way around)

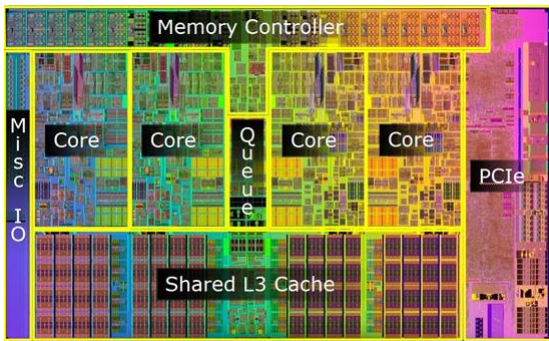
Even lower-level properties...

- Instruction Locality
 - Temporal:
 - Do instructions repeat within some time?
 - Loopy vs function-call code
 - Spatial:
 - Branch Density vs Computation
- Datatype Regularity?
 - Integer vs Floating Point
 - Small vs Large Bitwidth
 - (Ratio of resources + conversion overheads)
- Probably many other important properties, depending on the context and architecture...

A naïve property-based classification

CPU

(“Ordinary” Apps)

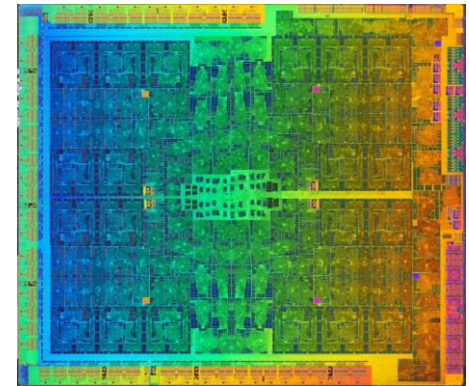


- + Irregular Control/Memory
- + Fine-grain Instruction Level Parallelism

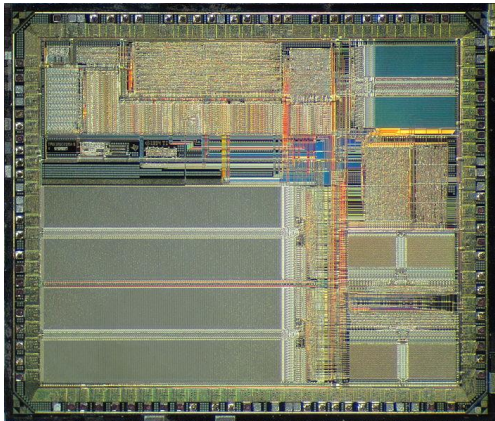
- + Thread-level and Data-level Parallelism
- + Medium Control/Memory

GPU

(Graphics, Data-proc.)



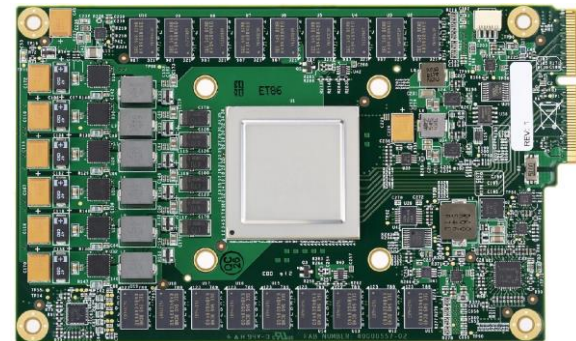
DSP (signal proc.)



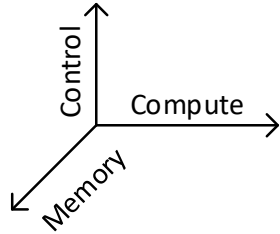
- + Fine-grain ILP
- + Highly-regular Memory

- + Extremely Regular Data Parallelism
- + Extreme Control/Memory Regularity

Google TPU: (Deep Learning)

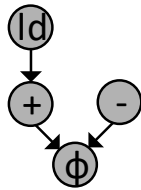


Course Themes

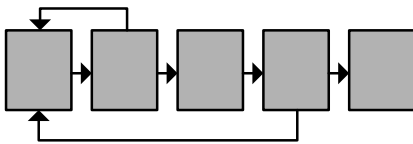


Applications

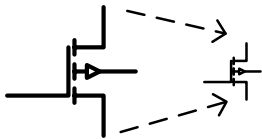
sub
ld
add
br



ISA: Hardware/ Software Interface



Microarchitecture



Technology

CS251a Course Overview

Why Study Computer Architecture?

- **Understand where computers are going**
 - Future capabilities drive the computing world
 - Forced to think 5+ years into the future
- **Exposure to high-level design**
 - Less about “design” than “what to design”
 - Engineering, science, art
 - Architects paint with broad strokes
 - The best architects understand all the levels
 - Devices, circuits, architecture, compilers, applications
- **Understand hardware for software tuning**
- **Real-world impact**
 - No computer architecture → no computers!
- **Get a job** (design or research)

Some Course Goals

- **Exposure to “big ideas” in computer architecture**
 - Pipelining, parallelism, caching, locality, abstraction, etc.
- Exposure to examples of good (and some bad) engineering
- Understanding computer performance and metrics
 - Empirical evaluation
 - Understanding quantitative data and experiments
- “Research” exposure
 - Read research literature (i.e., papers)
 - Course project
 - Cutting edge proposals

Course Prerequisites

- Basic Computer Organization
 - Logic: gates, Boolean functions, latches, memories
 - Datapath: ALU, register file, memory interface, muxes
 - Control: single-cycle control, micro-code
 - Caches & pipelining (will go into these in more detail here)
 - Some familiarity with assembly language
 - Hennessy & Patterson's "Computer Organization and Design"
 - Operating Systems (processes, threads, & virtual memory)
- Significant programming experience
 - Why? assignments require writing code to simulate hardware
 - Not difficult if competent programmer; extremely difficult if not
- **This class will have a gentle ramp, so don't worry.**

Course Components

- **Reviews:**
 - We will read a ~20 of research papers from literature, including classic and modern works. You will write a short review, to be submitted over CCLE, for one or two papers each week.
- **Homeworks:**
 - There will be a 3-4 homeworks during the quarter, in which you will apply your knowledge to real-world architecture evaluation.
 - These homeworks will mostly revolve around gem5.
- **Exams:**
 - There will be two take-home midterm exams during the quarter, covering the first third and second third of the course material.
 - There will be no final exam.
- **Project:**
 - Option 1: Hacking the gem5 simulator based on a suggested idea (evaluate new/existing architecture idea using simulation).
 - Option 2: Open ended project of your choice.

Paper Readings/Reviews

- You are expected to have completed the assigned readings before class and actively participate in discussions.
- Requiring reviews on (most) class days is intended to incentivize this.
- Reviews contain three short paragraphs: (max 600 words)
 - summarize the problem/goal/intended contributions
 - summarize the paper's methods and results
 - give your opinion of the paper
- Scale:
 - 5: Excellent, 4: Satisfactory, 2: Unsatisfactory
 - 4 doesn't mean you did something wrong, just not too insightful
- Rules:
 - Submit on CCLE before class starts that day (TA will grade)
 - Feel free to discuss any readings on piazza or ask questions before class. (TA and I will monitor/participate)

Required Texts

- No required *traditional* textbook for this course.
- Required reading will include:
 - Morgan Claypool Synthesis Lectures
 - Published papers (available on campus network through ACM and IEEE libraries).
- Optional Textbooks:
 - John Shen and Mikko Lipasti, Modern Processor Design: Fundamentals of Superscalar Processors, McGraw-Hill, 2005.
 - John L. Hennessy and David A. Patterson, Computer Architecture: A Quantitative Approach Morgan Kaufmann Publishers, Sixth Edition.

Homeworks

- 3-4 Homeworks during the first 2/3rds of the course.
- Welcome to work in pairs or individually.
- Intention behind homeworks:
 - Teach basics about simulation.
 - Get experience in architecture analysis.
 - Get everyone familiar with a set of tools, so that you can cooperatively work together in the project later on...
 - *Not to cover all principles discussed in class.*
- *HW0: Do Parts 1 and 2 from "learning gem5" online course. "<http://learning.gem5.org/book/index.html>"*

Exams

- Experiment for this course: 24-hour take-home exams
 - Should you time on the exams to think more deeply about the questions, and put together well thought-out responses.
 - Emphasizes reasoning/argumentation over memorization.
- Exam Content:
 - 3-4 short-essay questions (focusing on concepts)
 - Topic fair game: anything discussed in class or in readings
 - Questions may be about a new aspect of a relevant subject
- Exam Rules:
 - You **may** use any paper/textbook resources
 - You may **not** discuss with *anyone* about the questions, including in person or on piazza, etc. (duh)
- Advice:
 - Exam could take as little as 1-3 hours depending on writing speed...
 - Could take entire 24 hours if you do not keep up with papers
- Two exams (no final), testing 1/3 of material each
- Exams are similar to UW arch PhD qualifying exam ([link online](#))

Project

- In lieu of final exam, we will have a ~ 4 week project.
 - Start before the Exam2, but most of work can be done afterwards
 - Work in teams of 2-3. (preferably not 1 or 4...)
- Why Project? (previous 251a did not have one)
 - Give you a chance to put into practice some of the ideas
 - Give you freedom to work on something you like
 - Learn tools/approach that can be useful later
- What is a project? Options:
 1. Hacksim: Hack an architecture simulator to implement a new architecture or microarchitecture idea, and evaluate it. (some sample projects are online, and I will add more during first 2 weeks)
 2. Open-ended: Propose a research idea and evaluate it using any means (okay to combine with ongoing/concurrent work)
- Deliverables: report (+ source code if applicable)
 - Report should be similar to research papers (but shorter)
 - Guidelines online.

Grading

- Grade Breakdown:
 - Participation: 5%
 - Reviews: 15%
 - Homeworks: 15%
 - Exams: 30%
 - 1/3 Term: 15%
 - 2/3 Term: 15%
 - Project: 35%
- Grading philosophy for graduate courses:
 - Grade is only an incentive to meet minimal requirements
 - Your own motivation will determine whether you get anything meaningful out of this course.

Logistics

- CCLE:
 - Turning things in and reporting grades (but that's it!)
- Webpage: <https://github.com/PolyArch/cs251a>
 - Post homeworks, course schedule, project description, etc...
 - I will post presentation pdfs on course schedule, but probably not until just before or just after class
- Piazza: (<http://piazza.com/ucla/fall2018/cs251a>)
 - Discussions & announcements
 - I will sign you up for piazza if you are still enrolled by Friday

Contact Us

- Office Hours:
 - 9:00am-10:00am Wednesday, 2:00pm-3:00pm Friday
 - Room 484, E6
- TA: Vidushi Dadu
 - Office Hours: Fri 10:30am-12:30pm
 - Room 3256S-A, Boelter
 - Simulator support + review/hw grading
 - Ask her if CCLE hand-in is not set :)

