# CS/ECE 552: Course Introduction

Prof. Matthew D. Sinclair

Lecture notes based in part on slides created by Mikko Lipasti, Mark Hill, David Wood, Guri Sohi, Josh San Miguel, John Shen, and Jim Smith

# Who am I?

**Prof. Matt Sinclair**
sinclair@cs.wisc.edu
Computer Sciences 6369

**Education:**
BS in CMPE & CS, University of Wisconsin-Madison, 2009
MS in ECE, University of Wisconsin-Madison, 2011
PhD, University of Illinois at Urbana-Champaign, 2017

**Research Interests:**
- Caches, coherence protocols and memory systems
- Heterogeneous systems
- Parallel programming and algorithms
- Mobile computing
- Processor microarchitecture

# What is Education?

# What is Education?

Provides a set of fundamental axioms for producing and interpreting knowledge, which serves as an accessible bridge between the student learner and the infinite capabilities and complexities of the natural world.

**everyday people**

**services**

**education**

**sciences**

**physical world**

# What is Computer Architecture?

Provides a set of fundamental axioms for producing and interpreting knowledge, which serves as an accessible bridge between the student learner and the infinite capabilities and complexities of the natural world.

- **everyday people**
- **services**
- **education**
- **sciences**
- **physical world**

# What is Computer Architecture?

Provides a set of fundamental axioms for producing and interpreting **data**, which serves as an accessible bridge between the **digital programmers** and the infinite capabilities and complexities of the **analog hardware**.

**everyday people**

**software services (OS, apps)**

**computer architecture**

**hardware circuits**

**physical world**

# Computer Architecture

**Instruction Set Architecture (IBM 360)**

- *… 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, Blaauw, & Brooks, 1964*

**Machine Organization (microarchitecture)**

- ALUs, Buses, Caches, Memories, etc.

**Machine Implementation (realization)**

- Gates, cells, transistors, wires

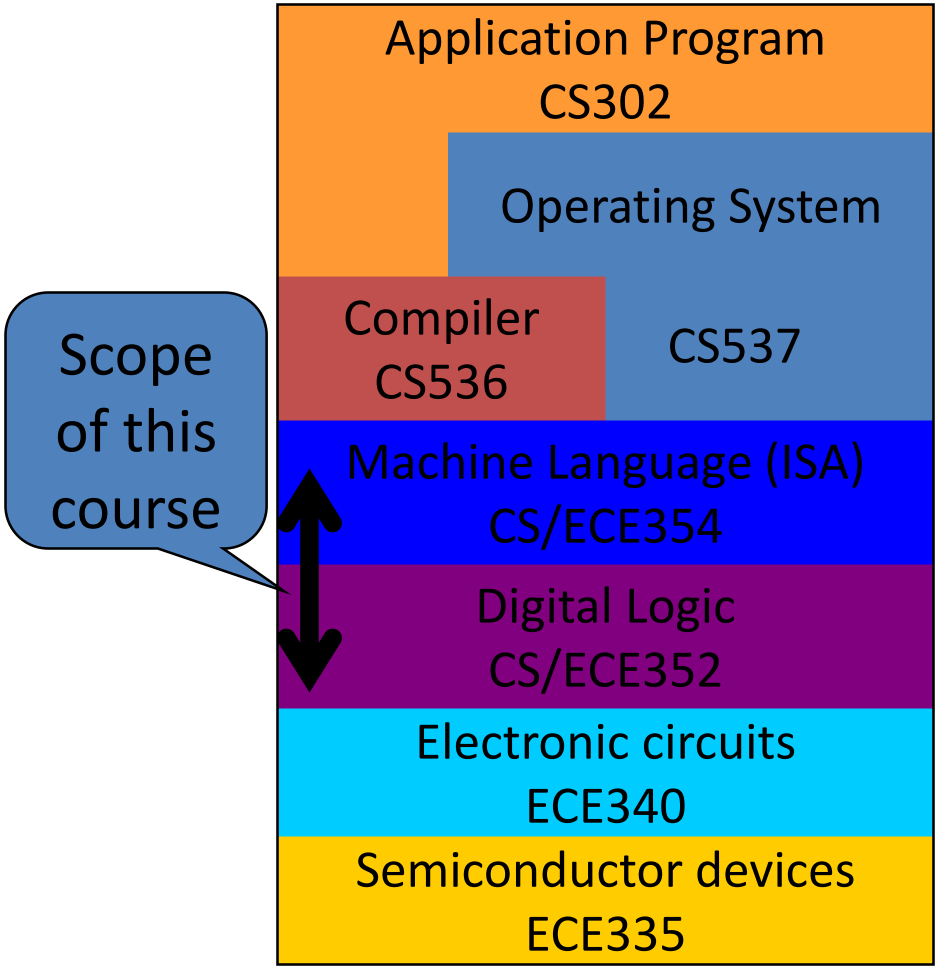
# Why Take 552?

- To become a computer designer
  - Alumni of this class helped design your computer
- To learn what is *under the hood* of a computer
  - Innate curiosity
  - To better understand when things break
  - To write better code/applications
  - To write better system software (O/S, compiler, etc.)
- Because it is intellectually fascinating!
  - CPUs are arguably the most complex highly-integrated man-made devices

# 552 In Context

- Prerequisites
  - 252/352 – gates, logic, memory, organization
  - 252/354 – high-level language down to machine language interface or instruction set architecture (ISA)
- This course – 552 – puts it all together
  - Implement the logic that provides ISA interface
  - Must do datapath and control, but no magic
  - Manage tremendous complexity with abstraction
- Follow-on courses explore trade-offs
  - CS/ECE 752, ECE 555/ECE 755, CS/ECE 757, CS 758

# 552 In Context



# Computer Architecture

- Exercise in engineering tradeoff analysis
  - Find the fastest/cheapest/power-efficient/etc. solution
  - Optimization problem with 100s of variables
- All the variables are changing
  - At non-uniform rates
  - With inflection points
  - Only one guarantee: Today's right answer will be wrong tomorrow
- Two high-level effects:
  - Technology push
  - Application pull

# Technology Push

- What do these two intervals have in common?
  - 1776-1999 (224 years)
  - 2000-2001 (2 years)

# Technology Push

- What do these two intervals have in common?
  - 1776-1999 (224 years)
  - 2000-2001 (2 years)
- Answer: Equal progress in processor speed!
- The power of exponential growth!
- Driven by Moore's Law
  - Device per chips doubles every 18-24 months
- Computer architects work to turn the additional resources into speed/power savings/functionality!

# Semiconductor History

| Date | Event | Comments |
|---|---|---|
| 1947 | $1^{st}$ transistor | Bell Labs |
| 1958 | $1^{st}$ IC  | Jack Kilby (MSEE ’50) @TI<br>Winner of 2000 Nobel prize |
| 1971 | $1^{st}$ microprocessor | Intel (calculator market) |
| 1974 | Intel 4004 | 2300 transistors |
| 1978 | Intel 8086 | 29K transistors |
| 1989 | Intel 80486 | 1M transistors |
| 1995 | Intel Pentium Pro | 5.5M transistors |
| 2006 | Intel Montecito | 1.7B transistors |
| 2015 | Oracle SPARC M7 | 10B+ transistors |

# Performance Growth

Unmatched by any other industry !
[John Crawford, Intel]

- Doubling every 18 months (1982-1996): 800x
  - Cars travel at 44,000 mph and get 16,000 mpg
  - Air travel: LA to NY in 22 seconds (MACH 800)
  - Wheat yield: 80,000 bushels per acre

# Performance Growth

Unmatched by any other industry !
[John Crawford, Intel]

- Doubling every 18 months (1982-1996): 800x
  - Cars travel at 44,000 mph and get 16,000 mpg
  - Air travel: LA to NY in 22 seconds (MACH 800)
  - Wheat yield: 80,000 bushels per acre

- Doubling every 24 months (1971-1996): 9,000x
  - Cars travel at 600,000 mph, get 150,000 mpg
  - Air travel: LA to NY in 2 seconds (MACH 9,000)
  - Wheat yield: 900,000 bushels per acre

# Technology Push

- Technology advances at varying rates
  - E.g. DRAM capacity increases at 60%/year
  - But DRAM speed only improves 10%/year
  - Creates gap with processor frequency!
- Inflection points
  - Crossover causes rapid change
  - E.g. enough devices for multicore processor (2001)
- Current issues causing an "inflection point"
  - Power consumption
  - Reliability
  - Variability

# Application Pull

- Corollary to Moore's Law:
  Cost halves every two years

  *In a decade you can buy a computer for less than its sales tax today. –Jim Gray*

- Computers cost-effective for
  - National security – weapons design
  - Enterprise computing – banking
  - Departmental computing – computer-aided design
  - Personal computer – spreadsheets, email, web
  - Mobile computing – smartphones

# Application Pull

- What about the future?
- Must dream up applications that are not cost-effective today
  - Augmented/Virtual reality
  - Machine learning
  - Telepresence
  - Mobile applications
  - Sensing, analyzing, actuating in real-world environments
- This is your job!

# Future of Computer Architecture?

Mobile Computers

Graphics Processors Intermittent Computing

Internet-Of-Things Approximate Computing

Stochastic Computing Ultra-Low-Power Processors

In-Memory Computing Many-Core Processors Cloud Computing

Reconfigurable Architectures Quantum Computers

Energy-Harvesting Processors Near-Threshold Computing

Warehouse-Scale Computers Biodegradable Processors

Neuromorphic Processors

# CS/ECE 552: Introduction (Part 2)

Prof. Matthew D. Sinclair

Lecture notes based in part on slides created by Mikko Lipasti, Mark Hill, David Wood, Guri Sohi, Josh San Miguel, John Shen, and Jim Smith

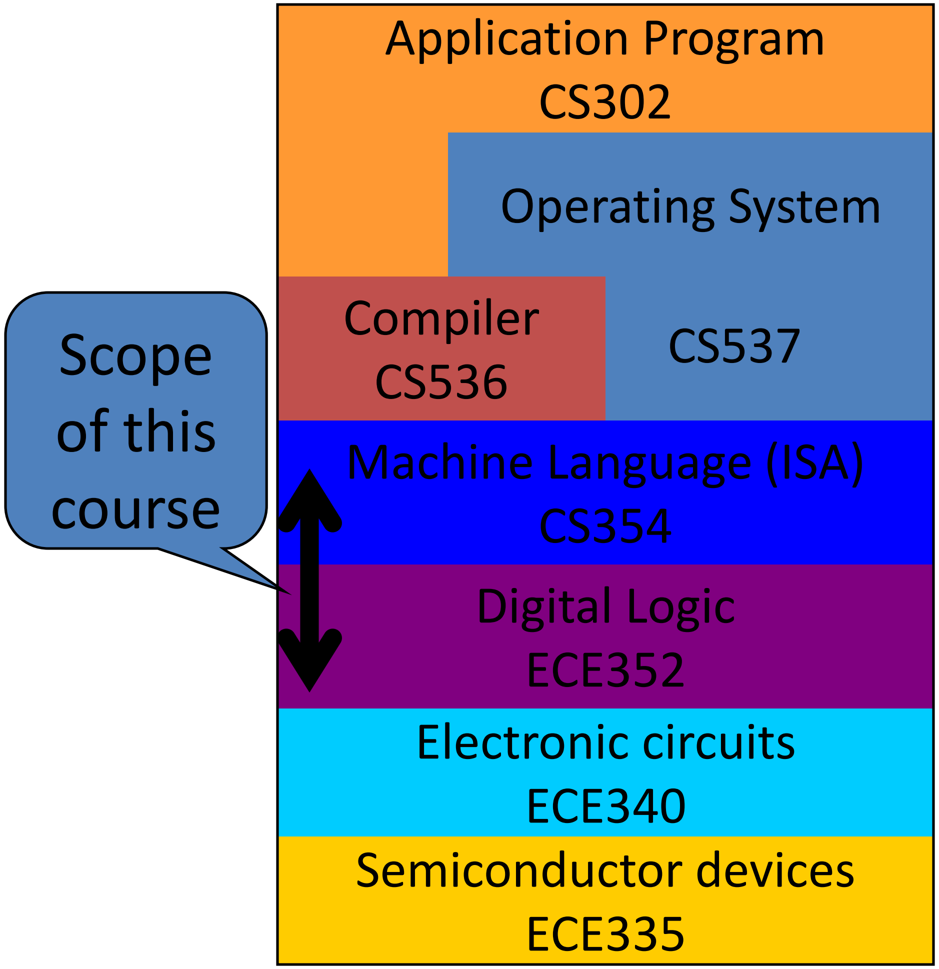
# Last Class

- Computer architecture and 552
- Technology push
- Application pull

# This Class

- Abstraction
- Amdahl’s Law
- Performance metrics

# 552 In Context

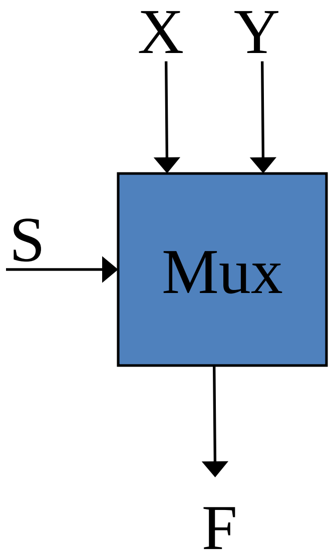


# Abstraction

- Difference between interface and implementation
  - Interface: WHAT something does
  - Implementation: HOW it does so

# Abstraction, E.g.

- 2:1 Mux (352)
- Interface

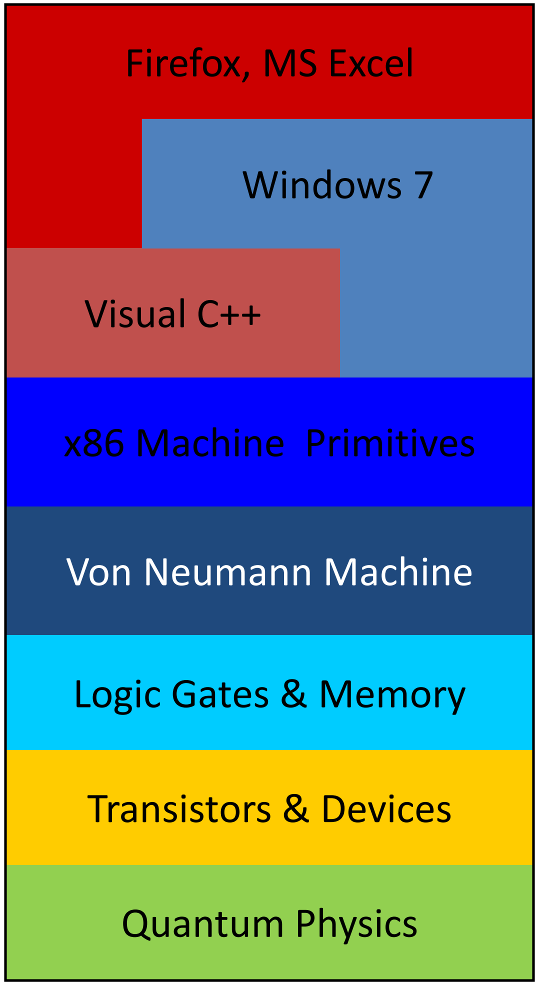



| S | F |
|---|---|
| 0 | X |
| 1 | Y |

- Implementations
  - Gates (fast or slow), pass transistors

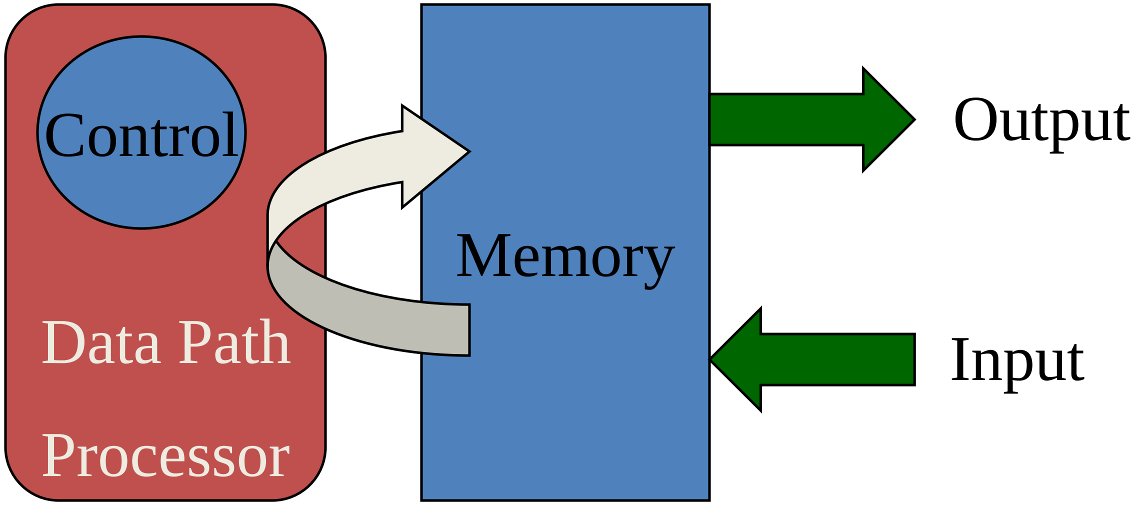
# What’s the Big Deal?

- Tower of abstraction
- Complex interfaces implemented by layers below
- Abstraction hides detail
- Hundreds of engineers build one product
- Complexity unmanageable otherwise



# Basic Division of Hardware

- In space:

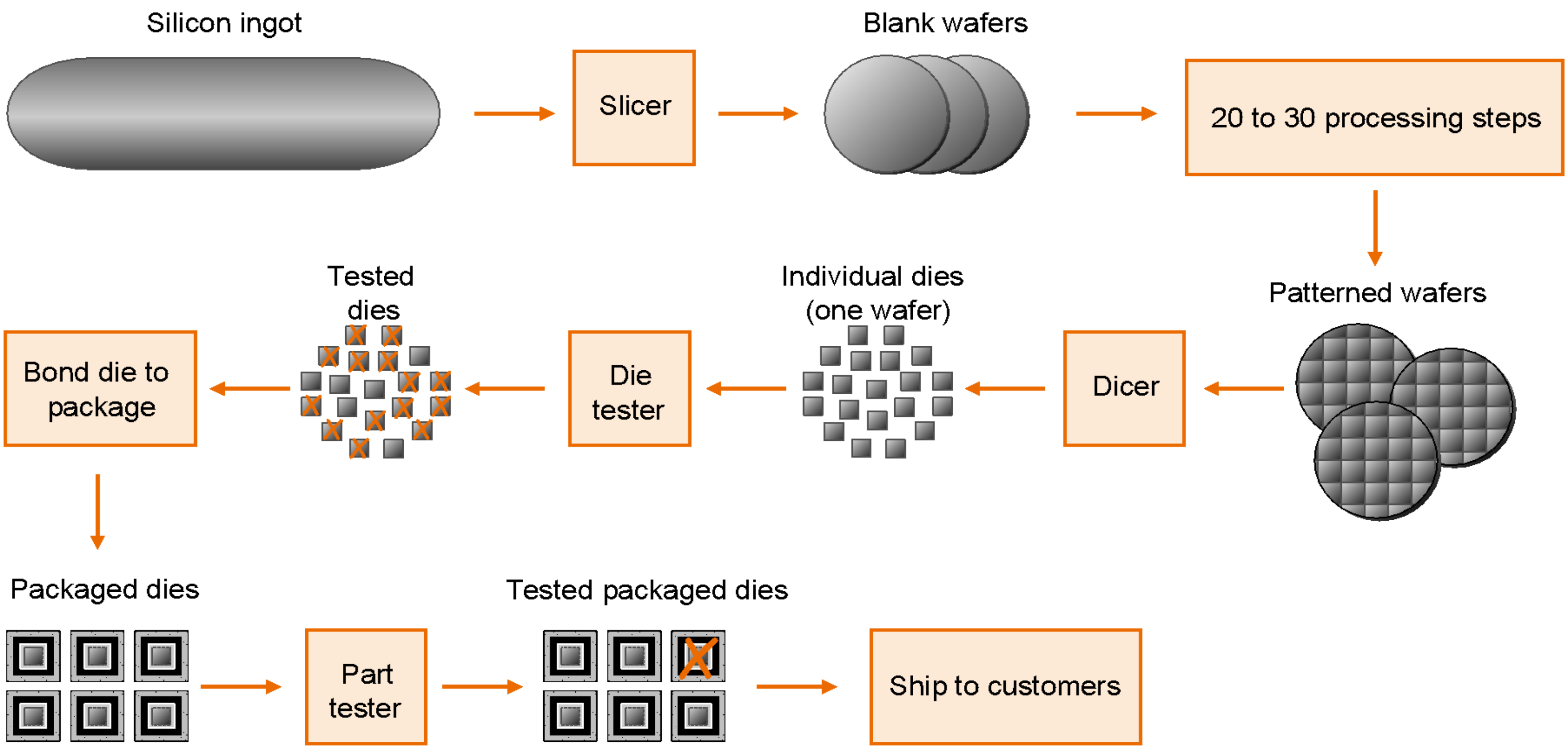


# Basic Division of Hardware

- In time:
  - Fetch instruction from memory 001011001001
  - Decode the instruction “add r1, r2, r3”
  - Read input operands read [r2], [r3]
  - Perform operation [r2] + [r3]
  - Write results write to [r1]
  - Determine the next instruction pc = pc + 4

# Building Computer Chips

- Complex multi-step process



# Summary

The ART and Science of Instruction-Set Processor Design

[Gerrit Blaauw & Fred Brooks, 1981]

- CPU designers must know BOTH software and hardware
- Both contribute to layers of abstraction

# Iron Law

$$processor\ performance =$$

$$\frac{\boldsymbol{instructions}}{\boldsymbol{program}} \times \frac{\boldsymbol{cycles}}{\boldsymbol{instruction}} \times \frac{\boldsymbol{seconds}}{\boldsymbol{cycle}}$$

$$= dynamic\ inst\ count\ \times\ CPI\ \times\ clock\ period$$

# Next Class

- Instruction set architectures (ISAs)
- MIPS