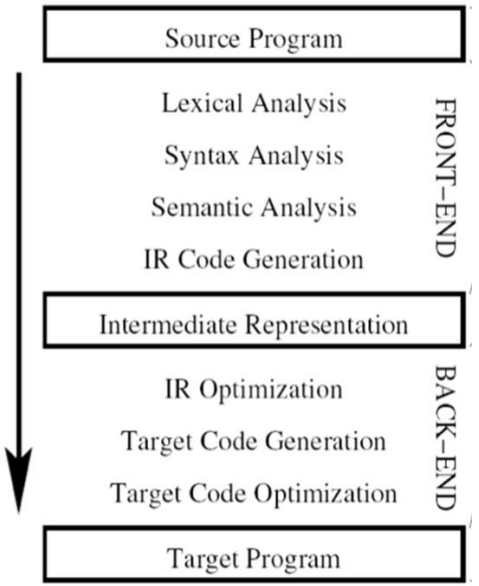
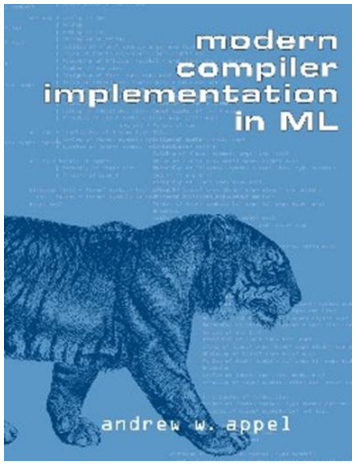
Topic 1: Introduction



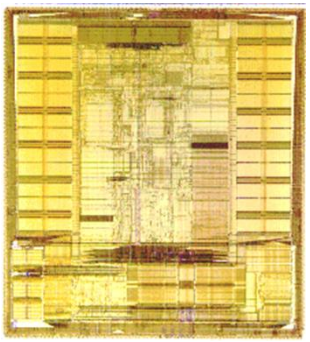
COS 320

Compiling Techniques



Princeton University
Spring 2016

Lennart Beringer



(Half) The Cast

Me: Lennart Beringer, Room 217 CS Building
eberinge@cs.princeton.edu, 258-0451
Office Hours: after class and by appointment

TA: Mikkel Kringelbach, Room 004 CS Building
mikkelk@cs.princeton.edu
Office Hours: TBC

(thanks to David August, David Walker, and Andrew Appel)

The more important half of the cast

Me: Lennart Beringer, Room 217 CS Building
eberinge@cs.princeton.edu, 258-0451
Office Hours: after class and by appointment

TA: Mikkel Kringelbach, Room 004 CS Building
mikkelk@cs.princeton.edu
Office Hours: TBC

YOU

What is a Compiler?

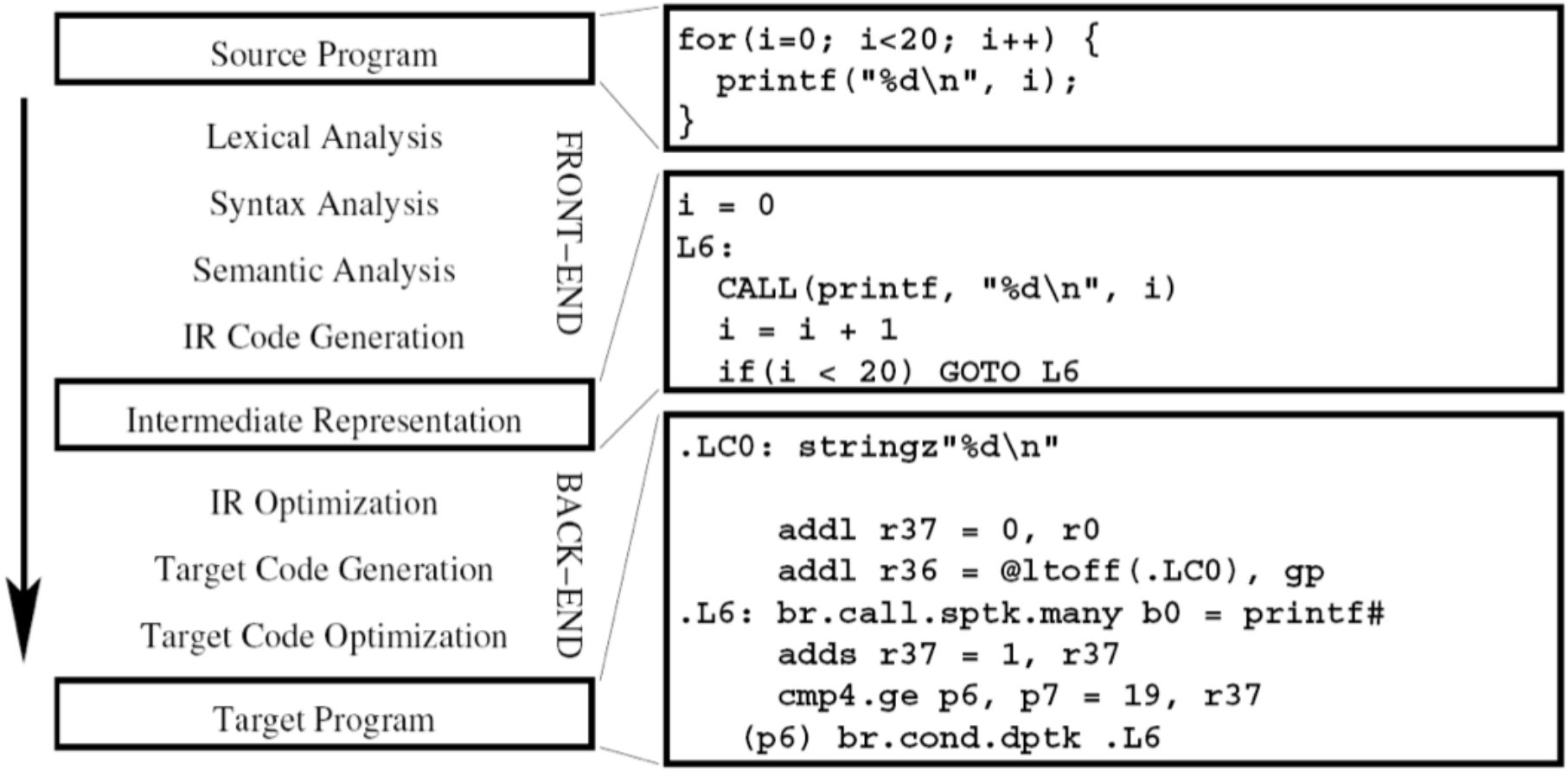
- A *compiler* is a program that takes a program written in a *source language* and translates it into a functionally equivalent program in a *target language*.

- Source Languages: C, C++, Java, FORTRAN,Haskell...

- Target Languages: x86 Assembly, Arm Assembly, C, JVM bytecode

- Compiler can also:
  - Report errors in source
  - Warn of potential problems in source
  - Optimize program, including parallelization eg for multi-core

What is a Compiler?



Interpreters versus compilers

I

C

Interpreters versus compilers

I

C

- **dynamic** processing of commands
- often: interaction loop with user
- access to stream of input (data)
- future commands/input unknown
- often: transformation of state / top-level environment
- easy to implement/extend
- helps rapid exploration of language features
- single execution
- little optimization potential

Interpreters versus compilers

I

- **dynamic** processing of commands
- often: interaction loop with user
- access to stream of input (data)
- future commands/input unknown
- often: transformation of state / top-level environment
- easy to implement/extend
- helps rapid exploration of language features
- single execution
- little optimization potential

C

- **static** (i.e. compile-time) processing program module
- programmatic interaction with user (read/write/files,…)
- little prior knowledge of input
- entire program text is known
- goal: efficient execution
- difficult to implement/extend
- implemented for mature languages
- huge optimization potential
- effort pays off over many runs

Interpreters versus compilers

I

- **dynamic** processing of commands
- often: interaction loop with user
- access to stream of input (data)
- future commands/input unknown
- often: transformation of state / top-level environment
- easy to implement/extend
- helps rapid exploration of language features
- single execution
- little optimization potential

C

- **static** (i.e. compile-time) processing program module
- programmatic interaction with user (read/write/files,…)
- little prior knowledge of input
- entire program text is known
- goal: efficient execution
- difficult to implement/extend
- implemented for mature languages
- huge optimization potential
- effort pays off over many runs

Many languages implemented in interpreters **and** compilers (Java, ML, ..).
Primary view affects language design. Sharing of techniques/components.

Why Learn About Compilers?

Compiler technology everywhere.

- C++ → Assembly
- Assembly → Machine Code
- Microcode → microcode binary
- Interpreters: Perl, Python, Java, …
- JITs: Android Dalvik VM, Java VM, …
- Publishing: Latex → PDF → Print on Paper
- Hardware Design: HW Description → Circuit/FPGA
- Automation: Water Fountain DL → Water Display

Bellagio, Las Vegas

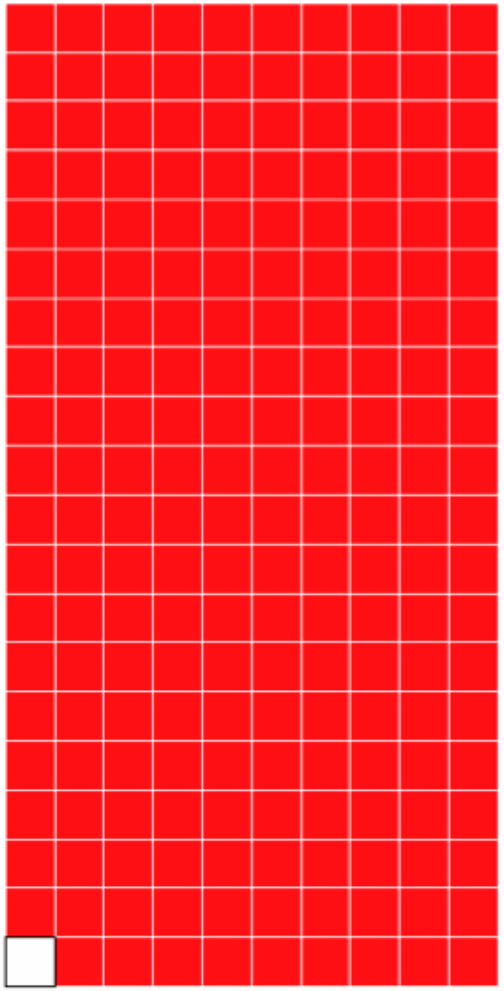
Why Learn About Compilers?

Almost all code goes through a compiler.

Linux

- C = 2,558,100 lines
- x86 assembly = 12,164 lines

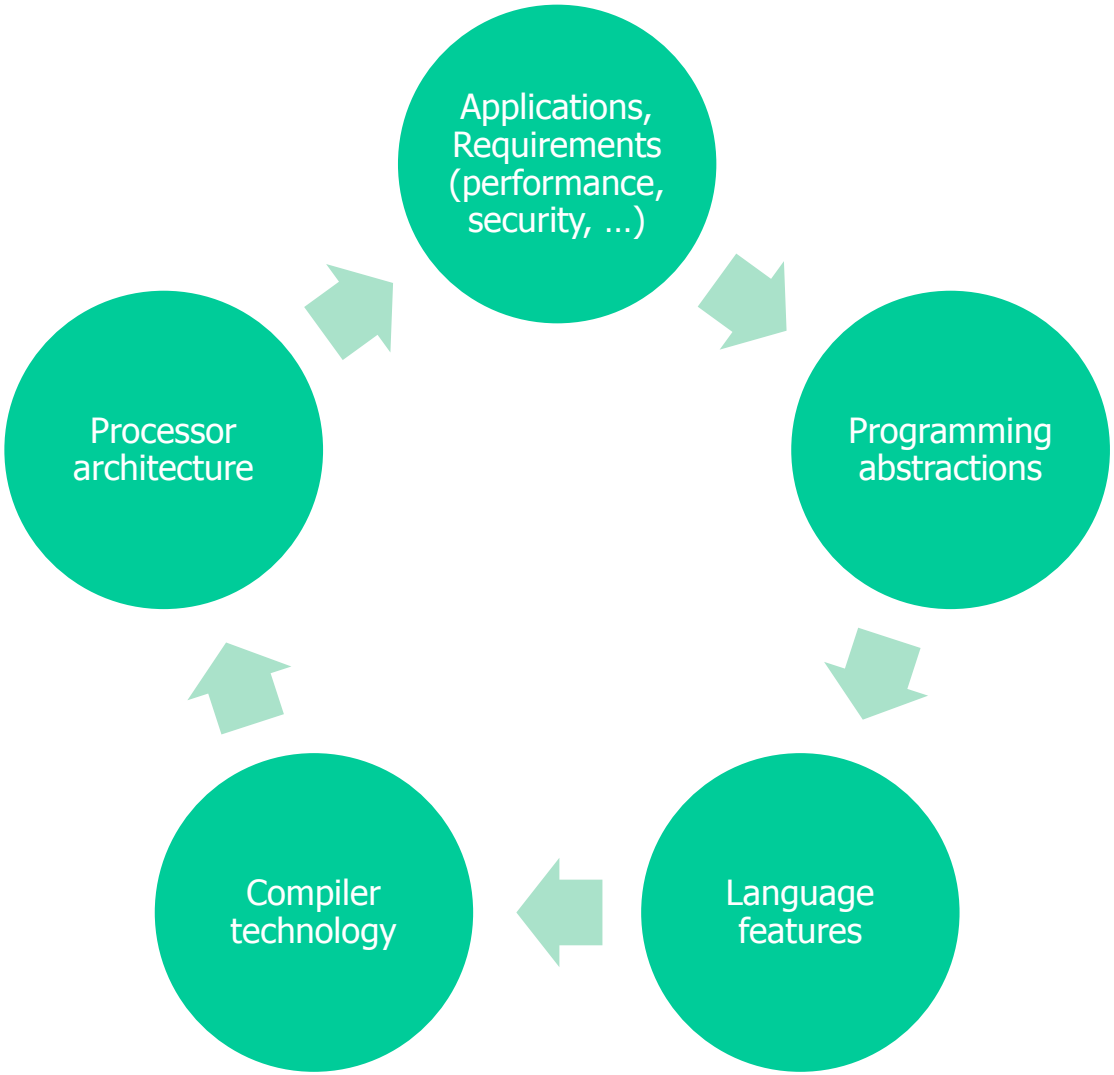
99.5% of Linux source goes through a compiler!



Compilers teach us about:

- Programming Languages
- Computer Architectures

Why learn about compilers?



Why Learn About Compilers?

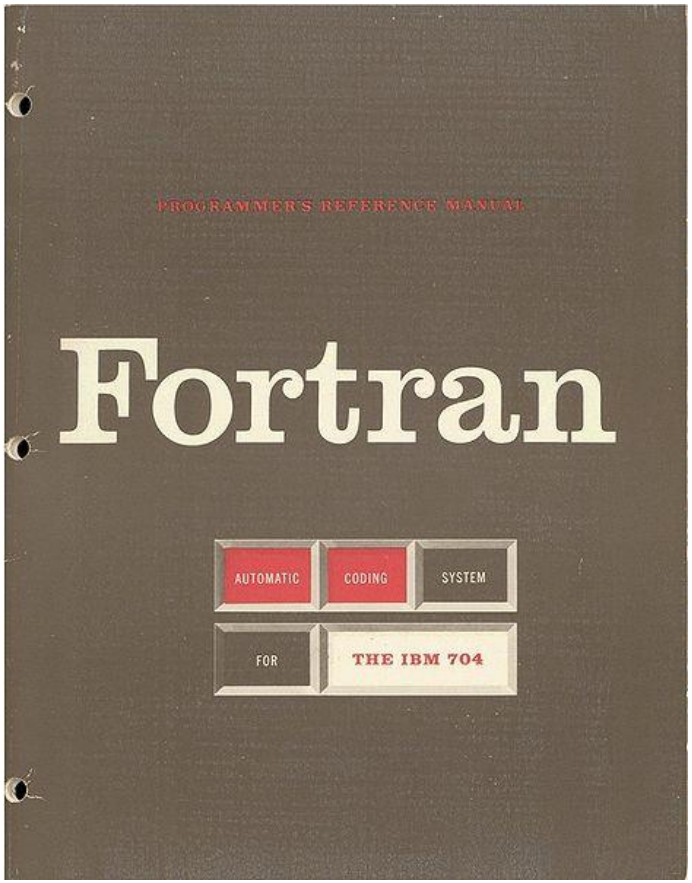
```
sum = 0;
for(i = 0; i < 1000000; i++)
  {
    sum = sum + big_array[i];
  }
```

```
sum = 0;
for(i = 0; i < 250000; i+=4)
  {
    sum = sum + big_array[i];
    sum = sum + big_array[i+1];
    sum = sum + big_array[i+2];
    sum = sum + big_array[i+3];
  }
```

- preparatory step for later program optimizations and parallelization
- clarifies model of computation: are the above code snippets equivalent?

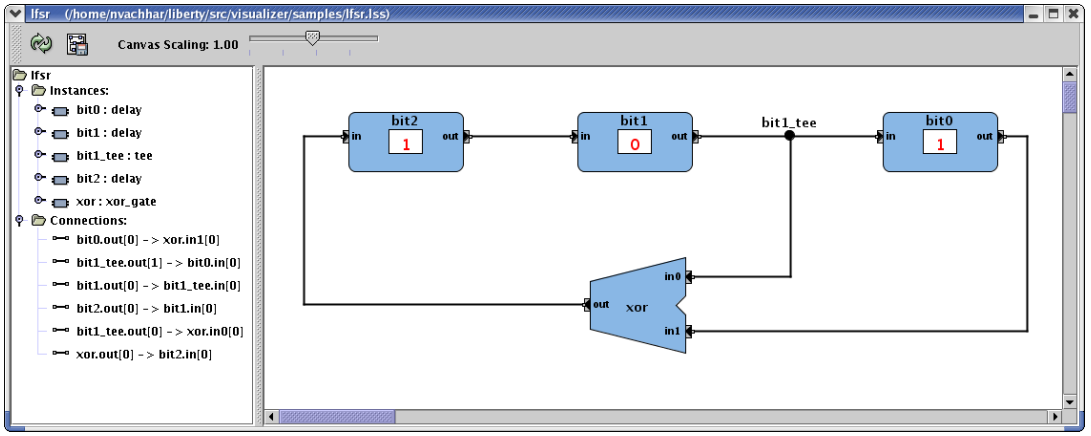
Why Learn About Compilers?

- IBM developed the first FORTRAN compiler in 1957
- Took 18 person-years of effort
- You will be able to do it in less than a week!



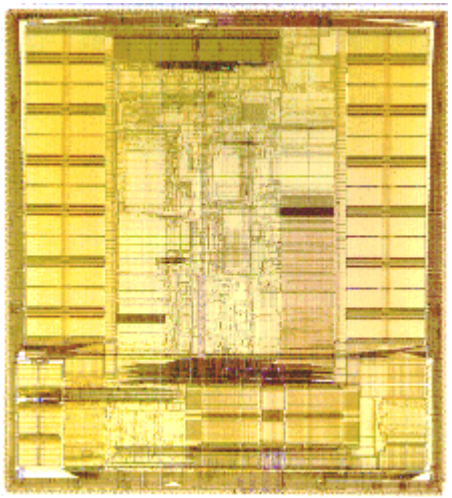
Why Learn About Compilers? Hardware Design

Compilation I

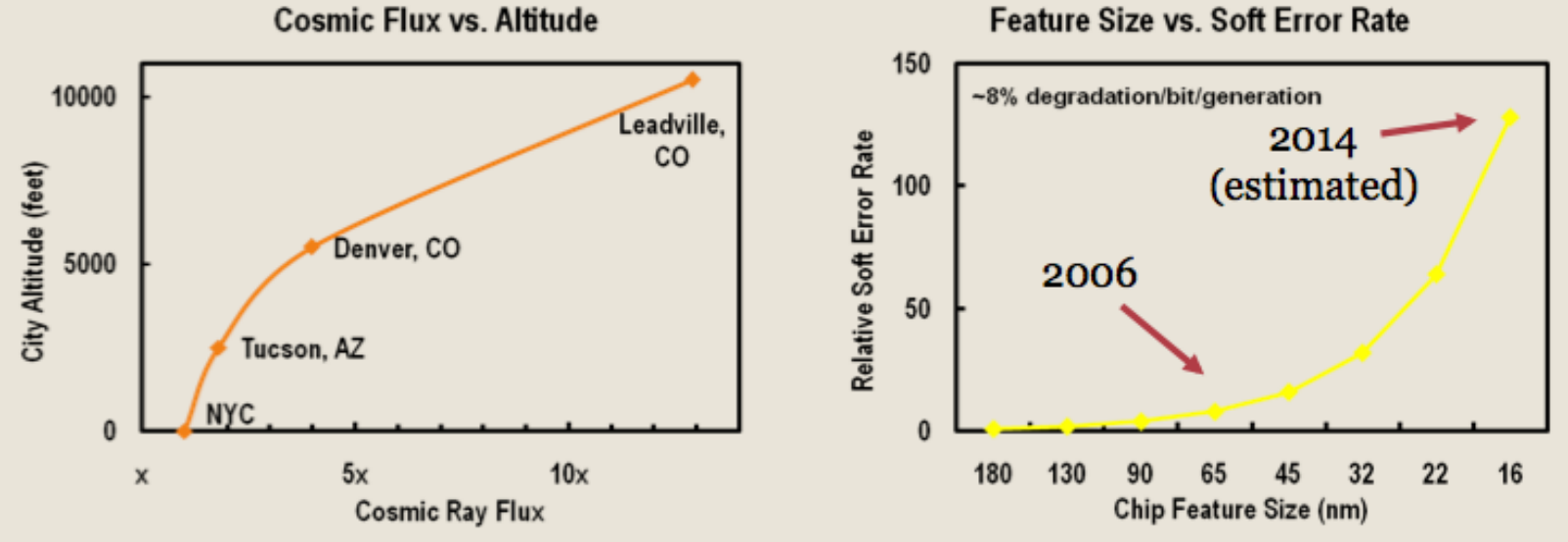



```
module toplevel(clock,reset);
  input clock;
  input reset;

  reg flop1;
  reg flop2;

  always @ (posedge reset or posedge clock)
    if (reset)
      begin
        flop1 <= 0;
        flop2 <= 1;
      end
    else
      begin
        flop1 <= flop2;
        flop2 <= flop1;
      end
endmodule
```

Compilation II



Why Learn About Compilers?
Computer Architecture



Princeton Research on Fault Tolerance wins CGO Test of Time Award

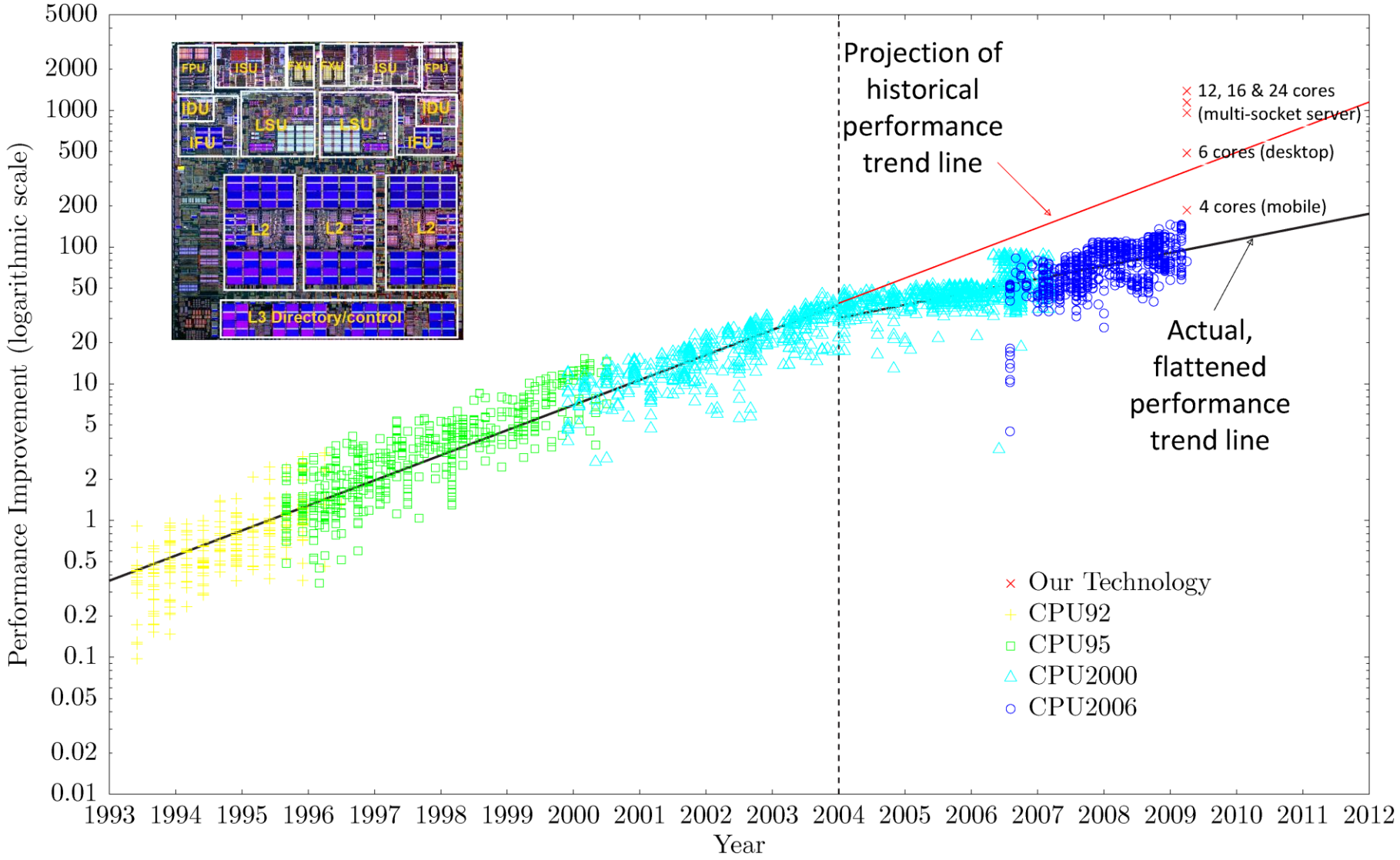
February 2, 2015

Every year, the International Symposium on Code Generation and Optimization (CGO) recognizes the paper appearing 10 years earlier that is judged to have had the most impact on the field over the intervening decade. This year at CGO 2015, the paper entitled "SWIFT: Software Implemented Fault Tolerance" by George A. Reis, Jonathan Chang, Neil Vachharajani, Ram Rangan, and David I. August won the award. The paper originally appeared at CGO 2005 and also won the best paper award that year at the conference. Congratulations to Princeton's Liberty Research Group for winning this prestigious award!

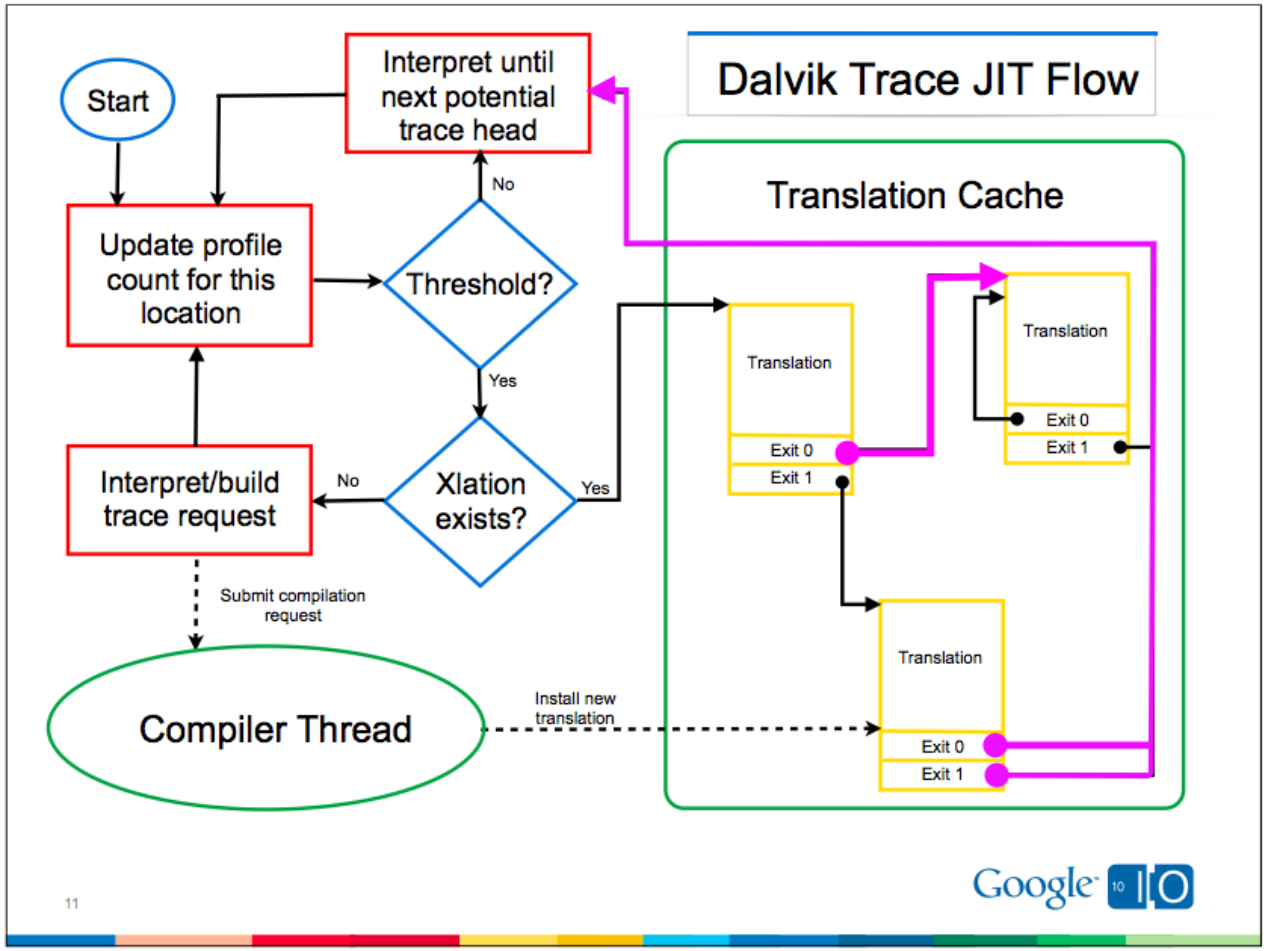
Selectively "harden" code regions where correctness is paramount

Why Learn About Compilers?

Your chosen field of computer architecture effectively dead?



Why Learn About Compilers?



Dynamic (re-)compilation of hot spots

Interaction between compiler and runtime monitoring

Grading

HW assignments	40%
Exams	20% + 35%
Participation/Quizzes	5%

Project

Build an optimizing compiler

- Front end
  - Lexer
  - Parser
  - Abstract Syntax Generator
  - Type Checker
  - Code Generator
- Back End Optimization
  - MakeGraph
  - Liveness
  - RegAlloc
  - Frame
- HW1 through 6 individual projects; HW7-HW9 probably in small groups

Exams

- Exams cover concepts presented in the lecture material, homework assignments, and required readings
- One double sided 8.5x11 page of notes allowed

Midterm Exam

- Thursday before break
- In class

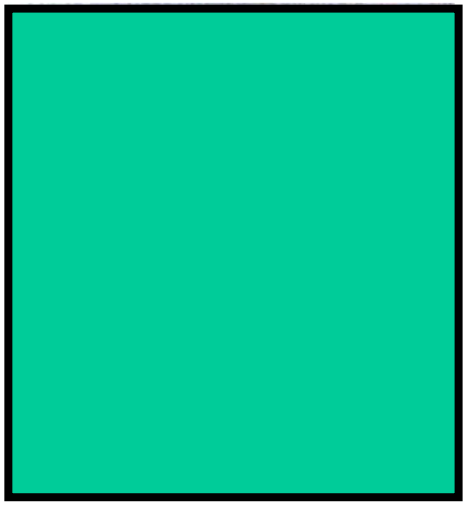
Final Exam

- The final exam will be cumulative
- Time/Place determined by the Registrar

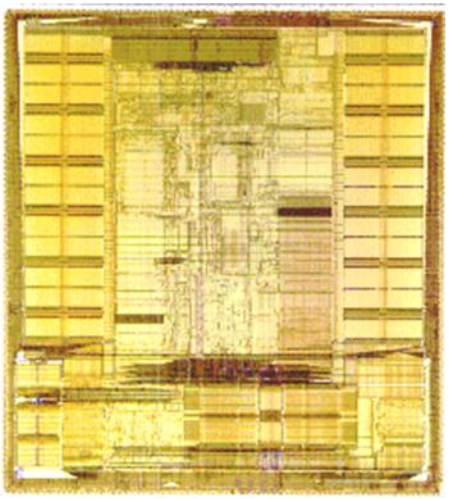
Quizzes

- Tuesday classes may have quizzes
- Not intended as a scare tactic – liberally graded
- Helps us assess progress of class
- Helps you to identify items you may want to revise
- Just one question (usually)

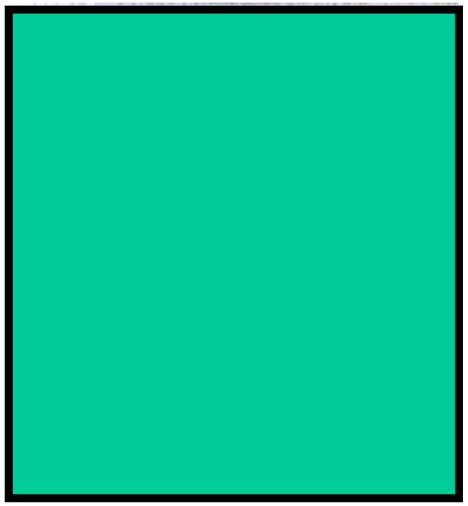
1



2



3



Participation

Negatives

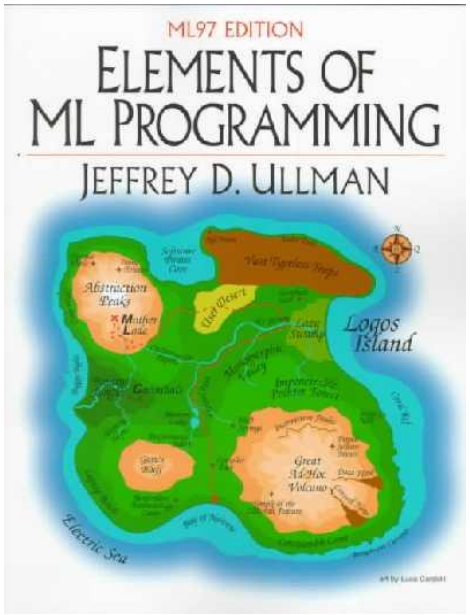
- Class disruptions (snoring, email, reading a book, etc.)
- Mistreatment of TA

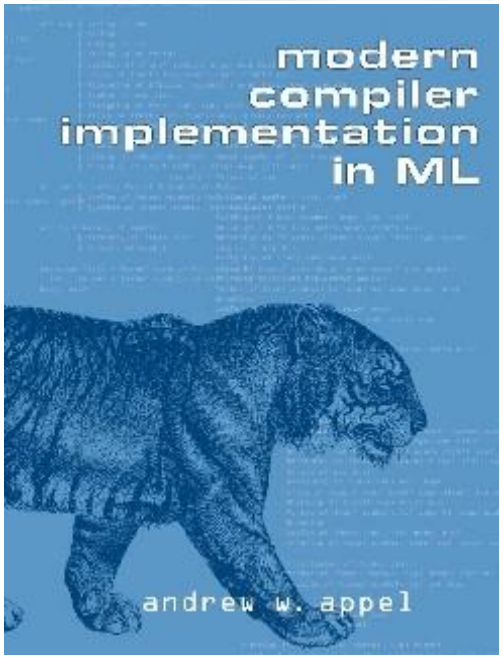
Positives

- Contribute questions and comments to class
- Participate in discussions in class and in Piazza
- Constructive feedback

Reading

- ML (optional)
  - Jeffrey D. Ullman, Elements of ML Programming, 2nd Edition, Prentice Hall.
  - Lawrence Paulson, ML for the working programmer
  - Bob Harper's online course book
- Required: Andrew W. Appel, Modern Compiler Implementation in ML. Cambridge University Press.
- CHECK ERRATA ON BOOK WEB SITES!
- Course Web Page – Off of CS page
  - Project Assignments (released successively)
  - Course Announcements
  - Blackboard, Piazza





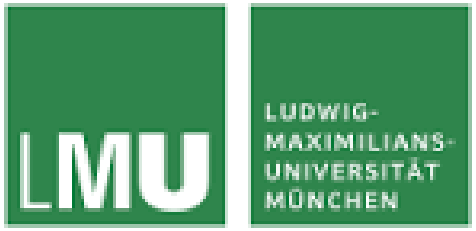
Who Am I?

Research Scholar at Princeton since 2009:

- interactive theorem proving (Coq)
- program verification, compiler correctness
- applications to crypto & security

Postdoc & Researcher (Edinburgh & LMU Munich)

- proof-carrying code
- resource-bounded computation



Mobile Resource Guarantees

Mobius

Ph.D. Edinburgh:

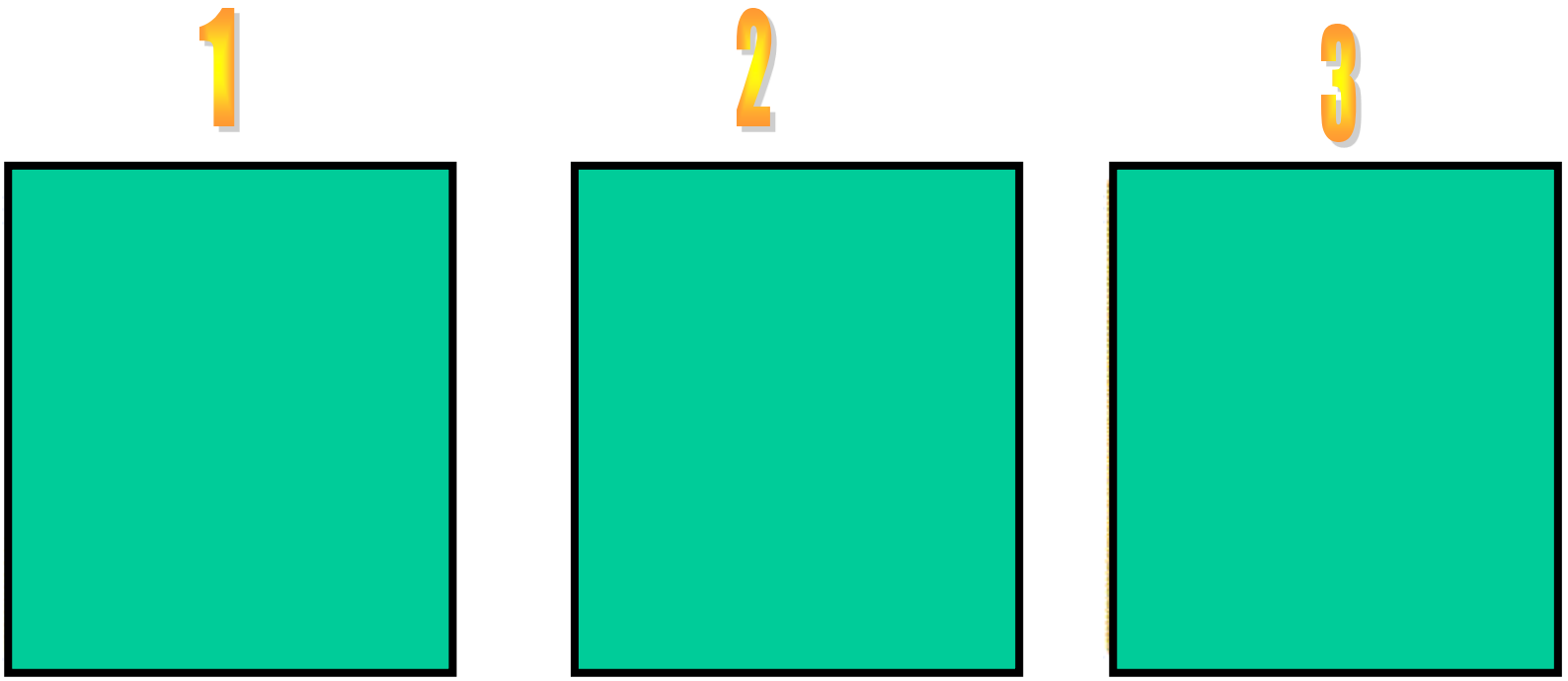
- Thesis: Reasoning about asynchronous processors

Our Pledge to You

- Quick response to questions and issues
- Reasonable late policy
  - Up to 2 days late for any single assignment without penalty
  - Up to 6 days late total across assignments I through VI
  - Lateness policy for HW7-9 announced later in semester
  - Contact me prior to deadline for special circumstances
- Fast turn-around on grading

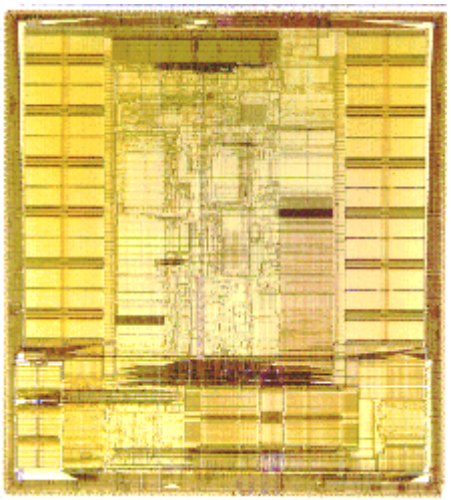
END OF ADMINISTRATIVE STUFF

It’s Tuesday

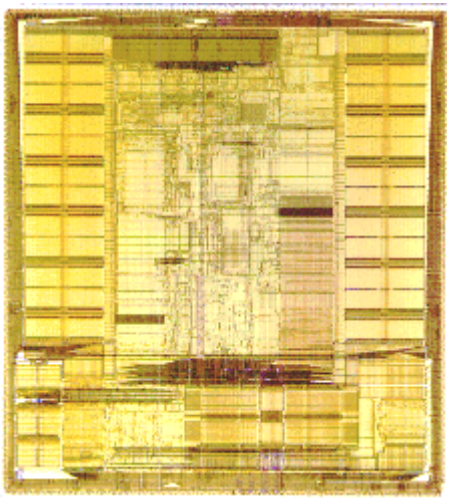


It’s Tuesday

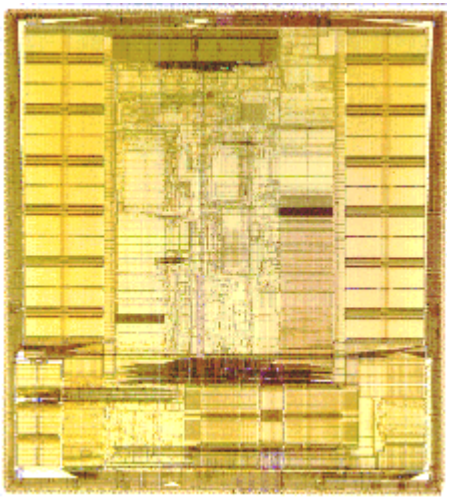
1



2



3



Quiz 0: Background (use index cards)

Front:

1. Full name and Email Address above the red line
2. Major/UG or G/Year (immediately below the red line)
3. Area (G: Research Area/UG: Interests)
4. Briefly describe ML/Ocaml/Haskell experience.
5. Briefly describe any C/C++ experience.
6. Briefly describe any compiler experience.
7. In which assembly languages are you fluent?

Back:

1. Why do processors have registers?
2. What is an instruction cache?
3. Can one always convert an NFA to a DFA? (yes, no, or wha?)