Lecture 1

CIS 500: SOFTWARE FOUNDATIONS

Stephanie Weirich Fall, 2014

Administrivia

- Instructor: Stephanie Weirich
  Office hours: Tomorrow 10-12, TBA in general
  Levine 510

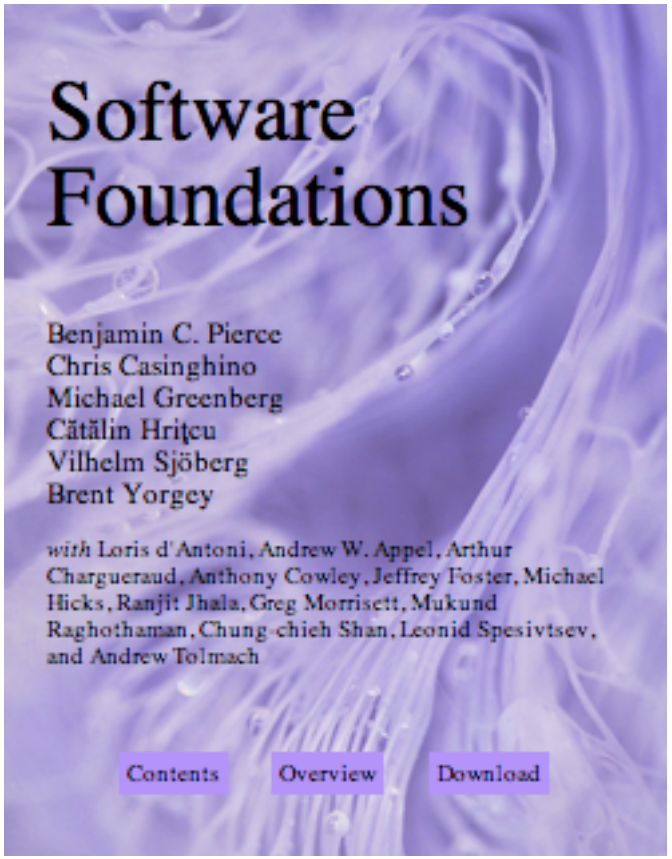
- TAs:
  - Arthur Azevedo de Amorim
    Office hours: TBA
  - Justin Chiu
    Office hours: TBA

Dr. Weirich will be in Sweden for ICFP Aug 30-Sep 5. Arthur will lecture next week.

- Location: Tues LRSM AUD, Thurs Levine 101
- E-mail: cis500@seas.upenn.edu
- Web site: http://www.seas.upenn.edu/~cis500
- Canvas: https://upenn.instructure.com
- Piazza: http://piazza.com/upenn/fall2014/cis500

Resources

- Course textbook: *Software Foundations*
  - Electronic edition tailor-made for this class
  - Use the version available from the cis500 course web pages.



- Additional books:
  - *Types and Programming Languages* (Pierce, 2002 MIT Press)
  - *Interactive Theorem Proving and Program Development* (Bertot and Castéran, 2004 Springer)
  - *Certified Programming with Dependent Types* (Chlipala, electronic edition)

Course Policies

- Prerequisites:
  - Significant programming experience
  - Mathematical sophistication
  - Undergraduate functional programming or compiler class

Grading:

• 24% Homework	~12 weekly assignments
• 18% Midterm I	(tentatively) Sep 30th
• 18% Midterm 2	(tentatively) Nov. 6th
• 36% Final	Thursday, Dec 18th 9-11AM
• 4% Class participation	

⇒ Lecture attendance is crucial!

"Regular" and "Advanced" tracks (graded separately).

"Regular" vs. "Advanced" Tracks

- "Advanced" track:
  - More and harder exercises
  - More challenging exams.
  - It is a superset of the "regular" material.

- All students start in the advanced track by default.
- Students who wish to take CIS 500 for WPE I credit (Ph.D.) *must* take the advanced track.
- Students may switch from advanced to regular track at any time.
  - Notify the course staff in writing.
  - The change is *permanent* after the first midterm.
- Students wishing to switch (back) to the advanced track:
  - Must do so *before* the first midterm exam.
  - Must make up all the advanced exercises (or accept the grade penalty).
- Only students taking the advanced track are eligible for an A+.

Participation Policy

- Class attendance is mandatory.
- We will be using "clickers" for
  - in-class mini quizzes
  - in-class polls about course material
- TurningPoint clickers use will be your attendance record.
- For next time: *buy a clicker.*
- *Any TurningPoint RF clicker will work, see note on course website.*

Homework Policies

- Homework is to be done *individually*.
- Homework must be *submitted via Canvas*
- Homework that is late is subject to:
  - 25% penalty for 1 day late
  - 50% penalty for 2 days late
  - 75% penalty for 3 days late

- Homework is due at *8:00pm* on the due date (generally Thurs.).

- Advanced track students must complete (or try to complete) all non-optional exercises.
  - Missing "advanced" exercises will count against your score.
- Regular track students must complete (or try to complete) all non-optional exercises except those marked "advanced".
  - Missing "advanced" exercises will not count against your score.
  - (But may help in your understanding of the material)

SOFTWARE FOUNDATIONS

LOGICAL FOUNDATIONS

A: How do we know something is true?

B: We test it out

A: But that isn't truth; testing can only give us evidence. How do we know something is **true**?

B: We prove it

A: How do we know that we have a proof?

B: We need to define what it means to be a proof.
A proof is a logical sequence of arguments, starting from some initial assumptions

A: How do we know that we have a valid sequence of arguments? Can any list be a proof?

All humans are mortal

All Greeks are human

I am a Greek

B: No, no, no! We need to think about how we *think*….

First we need a language…

- Gottlob Frege: a German mathematician who started in geometry but became interested in logic and foundations of arithmetic.

- 1879 Published "*Begriffsschrift, eine der arithmetischen nachgebildete Formelsprache des reinen Denkens*" (Concept-Script: A Formal Language for Pure Thought Modeled on that of Arithmetic)
  - First rigorous treatment of functions and quantified variables
  - $\vdash A$, $\neg A$, $\forall x.F(x)$
  - First notation able to express arbitrarily complicated logical statements

Gottlob Frege
1848-1925


BEGRIFFSSCHRIFT,

EINE DER ARITHMETISCHEN NACHGEBILDETE

FORMELSPRACHE

DES REINEN DENKENS.

VON

Dr. GOTTLOB FREGE

HALLE a/S.

VERLAG VON LOUIS NEBERT.

1879.

Images in this & following slides taken from Wikipedia.

Formalization of Arithmetic

- 1884: *Die Grundlagen der Arithmetik* (The Foundations of Arithmetic)
- 1893: *Grundgesetze der Arithmetik* (Basic Laws of Arithmetic, Vol. 1)
- 1903: *Grundgesetze der Arithmetik* (Basic Laws of Arithmetic, Vol. 2)

- Frege's Goals:
  - isolate logical principles of inference
  - derive laws of arithmetic from first principles
  - set mathematics on a solid foundation of logic

The plot thickens…

Just as Volume 2 was going to print in 1903, Frege received a letter…

Bertrand Russell

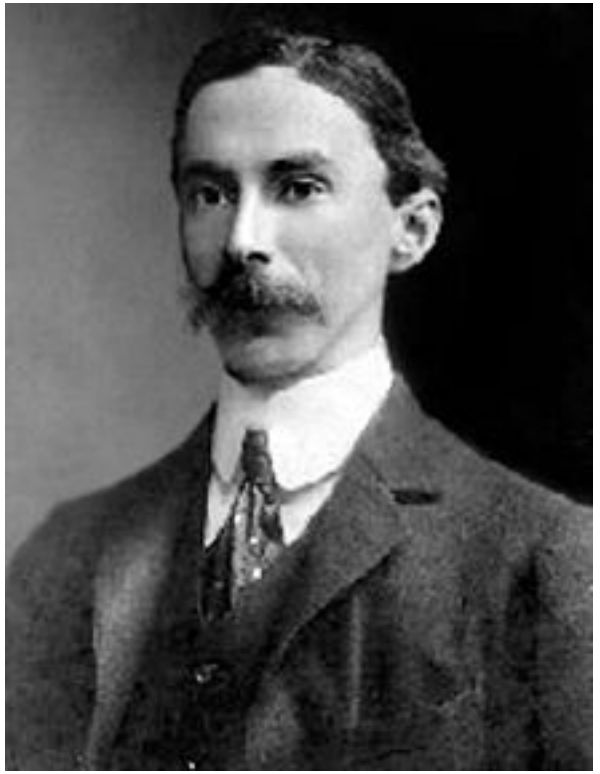
- *Russell's paradox:*

1. Set comprehension notation:
   { x | P(x) } "The set of x such that P(x)"

2. Let X be the set { Y | Y ∉ X }.

3. Ask the logical question:
   Does X ∈ X hold?

4. Paradox! If X ∈ X then X ∉ X.
   If X ∉ X then X ∈ X.



Bertrand Russell
1872 - 1970

- Frege's language could derive Russell's paradox.
- Frege's logical system could derive anything. Oops.

Addendum to Frege's 1903 Book

"Hardly anything more unfortunate can befall a scientific writer than to have one of the foundations of his edifice shaken after the work is finished. This was the position I was placed in by a letter of Mr. Bertrand Russell, just when the printing of this volume was nearing its completion." – *Frege, 1903*

Aftermath of Frege and Russell

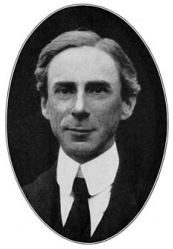
- Frege came up with a fix, but it made his logic trivial…

- 1908: Russell fixed the inconsistency of Frege's logic by developing a *theory of types*.

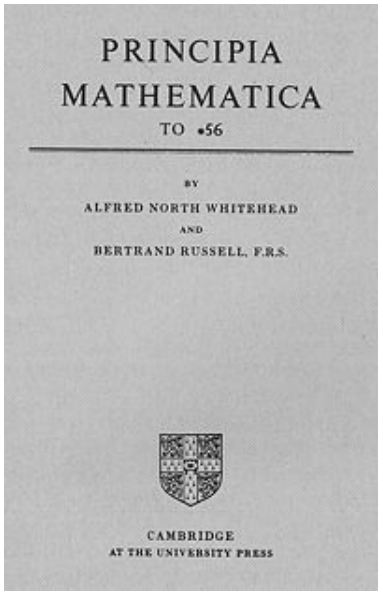
- 1910, 1912, 1913, (revised 1927): *Principia Mathematica* (Whitehead & Russell)
  - Goal: axioms and rules from which *all* mathematical truths could be derived.
  - It was a bit unwieldy…

> "From this proposition it will follow, when arithmetical addition has been defined, that 1+1=2."
> —Volume I, 1st edition, *page 379*

Whitehead



Russell



Logic in the 1930s and 1940s

- 1931: Kurt Gödel's first and second incompleteness theorems.
  - Demonstrated that any consistent formal theory capable of expressing arithmetic cannot be complete.

Kurt Gödel
1906 - 1978

- 1936: Genzen proves consistency of arithmetic.
- 1936: Church introduces the λ-calculus.
- 1936: Turing introduces Turing machines
  - Is there a decision procedure for arithmetic?
  - Answer: no it's undecidable
  - The famous "halting problem"
    - only in 1938 did Turing get his Ph.D.

Gerhard Gentzen
1909 - 1945

- 1940: Church introduces the *simple theory of types*

Alonzo Church
1903 - 1995

Alan Turing
1912 - 1954

Fast Forward…

- 1958 (Haskell Curry) and 1969 (William Howard) observe a remarkable correspondence:

types	~	propositions
programs	~	proofs
computation	~	simplification

N.G. de Bruijn
1918 - 2012

- 1967 – 1980's: N.G. de Bruijn runs Automath project
  - uses the Curry-Howard correspondence for computer-verified mathematics

- 1971: Jean-Yves Girard introduces System F
- 1972: Girard introduces Fω
- 1972: Per Marin-Löf introduces intuitionistic type theory
- 1974: John Reynolds independently discovers System F

Basis for modern type systems: OCaml, Haskell, Scala, Java, C#, …

… to the Present

- 1984: Coquand and Huet first begin implementing a new theorem prover "Coq"
- 1985: Coquand introduces the calculus of constructions
  - combines features from intuitionistic type theory and Fω
- 1989: Coquand and Paulin extend CoC to the calculus of inductive constructions
  - adds "inductive types" as a primitive
- 1992: Coq ported to Xavier Leroy's Caml

Thiery Coquand
1961 –

Gérard Huet
1947 –

- 1990's: up to Coq version 6.2
- 2000-2010: Coq version 8.3
- 2012: Coq version 8.4 ← CIS 500

Too many contributors to mention here…

- 2013: Coq receives ACM Software System Award

http://coq.inria.fr/refman/Reference-Manual002.html

So much for foundations… what about software?

SOFTWARE FOUNDATIONS

Building Reliable Software

- Suppose you work at (or run) a software company.

- Suppose, like Frege, you've sunk 30+ person-years into developing the "next big thing":
  - Boeing Dreamliner2 flight controller
  - Autonomous vehicle control software for Nissan
  - Gene therapy DNA tailoring algorithms
  - Super-efficient green-energy power grid controller

- Suppose, like Frege, your company has invested a lot of material resources that are also at stake.

- How do you avoid getting a letter like the one from Russell?

Or, worse yet, *not* getting the letter to disastrous consequences?

Approaches to Software Reliability

- Social
  - Code reviews
  - Extreme/Pair programming
- Methodological
  - Design patterns
  - Test-driven development
  - Version control
  - Bug tracking
- Technological
  - "lint" tools, static analysis
  - Fuzzers, random testing
- Mathematical
  - Sound type systems
  - "Formal" verification

Less "formal": Techniques may miss problems in programs

This isn't a tradeoff… all of these methods should be used.

Even the most "formal" can still have holes:

- Did you prove the right thing?
- Do your assumptions match reality?
- Knuth. "Beware of bugs in the above code; I have only proved it correct, not tried it."

More "formal": eliminate *with certainty* as many problems as possible.

Five Interwoven Threads

1. basic tools from logic for making and justifying precise claims about programs

2. the use of proof assistants to construct rigorous, machine checkable, logical arguments

3. the idea of functional programming, both as a method of programming and as a bridge between programming and logic

4. techniques for formal verification of properties of specific programs

5. the use of type systems for establishing well-behavedness guarantees for all programs in a given language

Can it Scale?

- Use of theorem proving to verify "real" software is still considered to be the bleeding edge of PL research.

- CompCert – fully verified C compiler
  Leroy, INRIA

- Vellvm – formalized LLVM IR
  Zdancewic, Penn

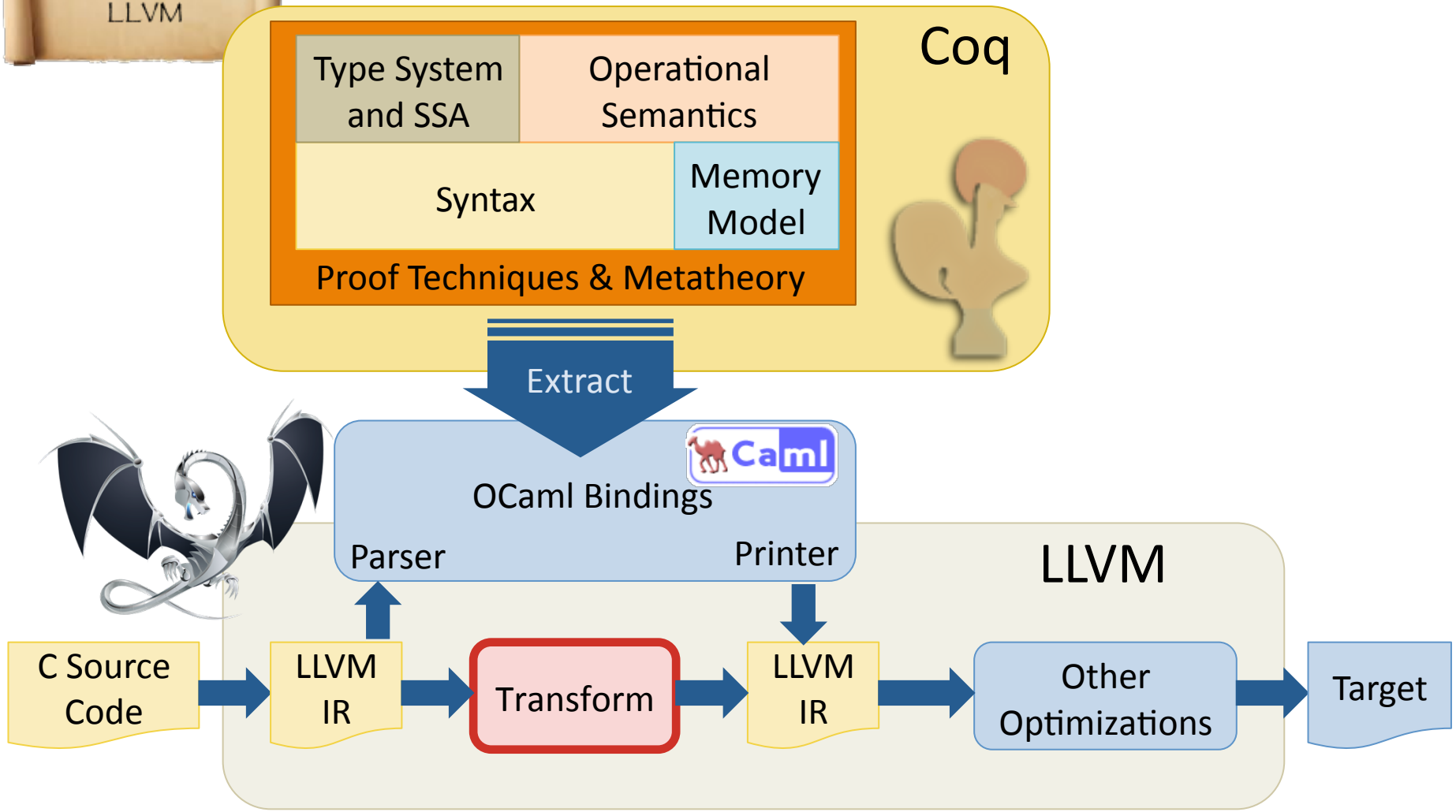
- Ynot – verified DBMS, web services
  Morrisett, Harvard

- Verified Software Toolchain
  Appel, Princeton

- Bedrock – web programming, packet filters
  Chlipala, MIT

- CertiKOS – certified OS kernel
  Shao & Ford, Yale

Vellvm Framework



Vellvm Framework

Coq

Type System and SSA

Operational Semantics

Syntax

Memory Model

Proof Techniques & Metatheory

Extract

Verified Transform

OCaml Bindings

Parser

Printer

LLVM

C Source Code

LLVM IR

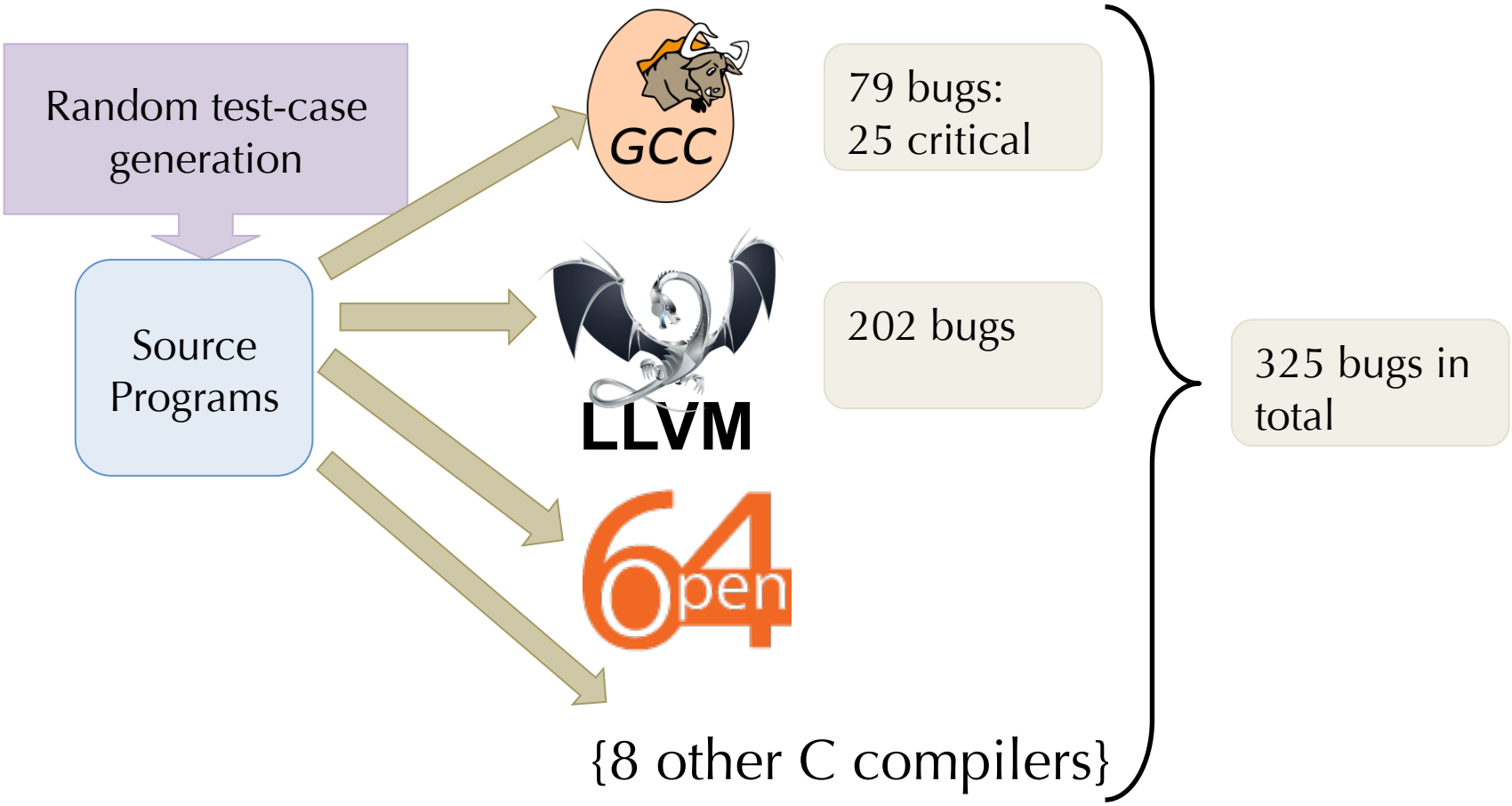
LLVM IR

Other Optimizations

Target

Does it work?

Finding and Understanding Bugs in C Compilers [Yang et al. PLDI 2011]



Verified Compiler: CompCert [Leroy et al.]
<10 bugs found in *unverified* front-end component

Regehr's Group Concludes

The striking thing about our CompCert results is that the *middle-end bugs* we found in all other compilers are *absent*. As of early 2011, the under-development version of *CompCert is the only compiler we have tested for which Csmith cannot find wrong-code errors*. This is not for lack of trying: we have devoted about six CPU-years to the task. *The apparent unbreakability of CompCert supports a strong argument that developing compiler optimizations within a proof framework, where safety checks are explicit and machine-checked, has tangible benefits for compiler users.*

(emphasis mine)

Why CIS 500?

- Foundations
  - Functional programming
  - Constructive logic
  - Logical foundations
  - Proof techniques for inductive definitions
- Semantics
  - Operational semantics
  - Modeling imperative "While" programs
  - Hoare logic for reasoning about program correctness
- Type Systems
  - Simply typed λ-calculus
  - Type safety
  - Subtyping
  - Dependently-typed programming
- Coq interactive theorem prover
  - turns doing proofs & logic into programming → fun!

COQ

Coq in CIS 500

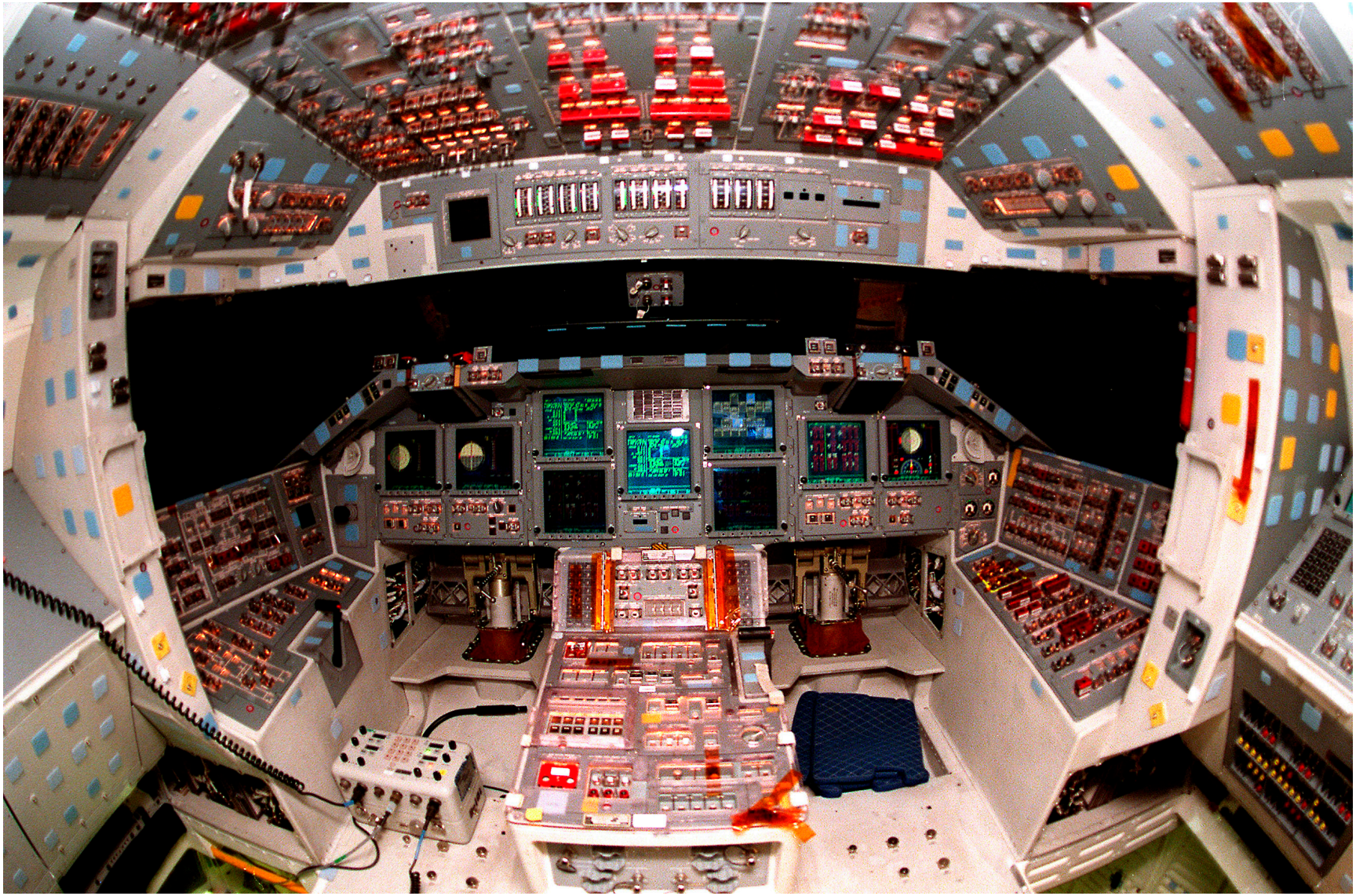
- We'll use Coq version 8.4
  - Available on CETS systems
  - Easy to install on your own machine

- See the web pages at: coq.inria.fr

- Two different user interfaces
  - CoqIDE – a standalone GUI / editor
  - ProofGeneral – an Emacs-based editing environment

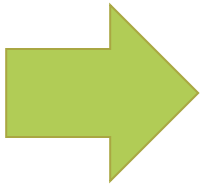
- Course web pages have more information.

Coq's Full System



Subset Used in CIS 500

To start.



By the end of the semester.

Getting acquainted with Coq.

BASICS.V

CIS 500: TODO

- Soon:
  - Register for Piazza
  - Try to log in to Canvas
  - Reading: Preface and Basics

- Before next time:
  - Install Coq v. 8.4
  - Buy a clicker

- HW1: Finish Basics.v
  - Due: Thursday, Sept. 4th at 8:00pm
  - Available on the web pages
  - Complete all non-optional exercises
  - There are no "advanced" for this HW
  - Submit to Canvas