Lecture 1

SOFTWARE FOUNDATIONS IN COQ

Steve Zdancewic OPLSS June, 2014

SOFTWARE FOUNDATIONS

Images in the following slides taken from Wikipedia.

The Story Begins…

- 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.

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 1902, Frege received a letter…

Bertrand Russell

- *Russell's paradox:*

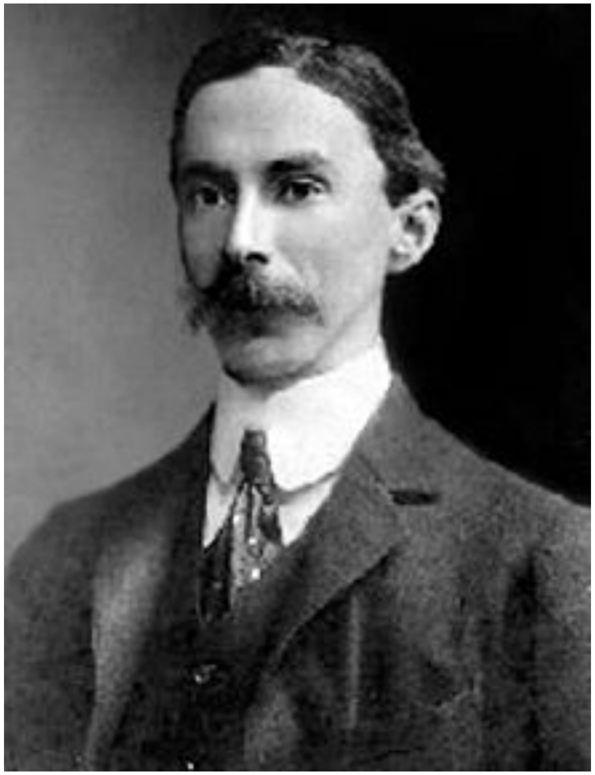
  1. Set comprehension notation:
     $\{ x \mid P(x) \}$ "The set of x such that P(x)"

  2. Let X be the set $\{ Y \mid Y \notin Y \}$.

  3. Ask the logical question:
     Does $X \in X$ hold?

  4. Paradox! If $X \in X$ then $X \notin X$.
     If $X \notin X$ then $X \in X$.

- Russell's paradox destroyed Frege's logical foundations…



Bertrand Russell
1872 - 1970

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

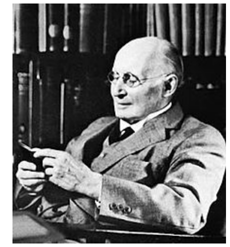
Aftermath of Frege and Russell

- 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*.

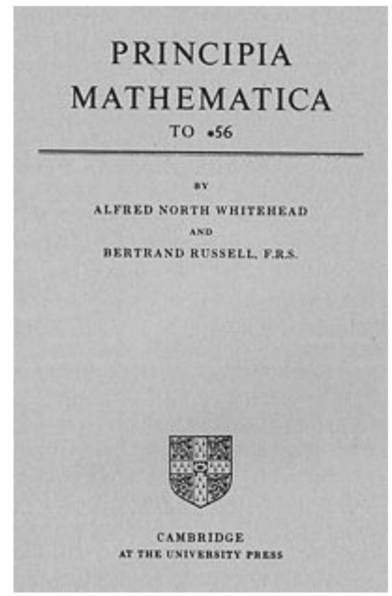
- 1910, 1912, 1913, (revised 1927):
  Principia Mathematica (Whitehead & Russell)
  - Goal: axioms and rules from which *all* mathematical truths could be derived.
  - It was a bit unwieldy…

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



Whitehead

Russell



PRINCIPIA
MATHEMATICA
TO *56

BY
ALFRED NORTH WHITEHEAD
AND
BERTRAND RUSSELL, F.R.S.

CAMBRIDGE
AT THE UNIVERSITY PRESS

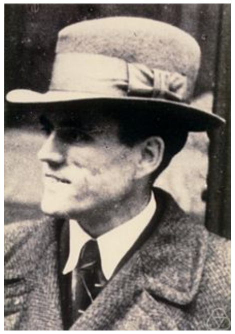
Logic in the 1930s and 1940s

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

- 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.

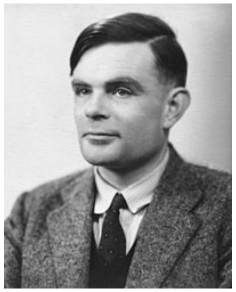
- 1940: Church introduces the *simple theory of types*

Kurt Gödel
1906 - 1978



Gerhard Gentzen
1909 - 1945

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

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

- 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

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

Too many contributors to mention here…

Thiery Coquand
1961 –

Gérard Huet
1947 –

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 Google or Nissan
  - Gene therapy DNA tailoring algorithms
  - Super-efficient green-energy power grid control software
- 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 Reliability

- Social
  - Code reviews
  - Extreme/Pair programming
- Methodological
  - Design patterns
  - Test-driven development
  - Version control
  - Bug tracking
- Technological
  - “lint” tools
  - Fuzzers
- 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?

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

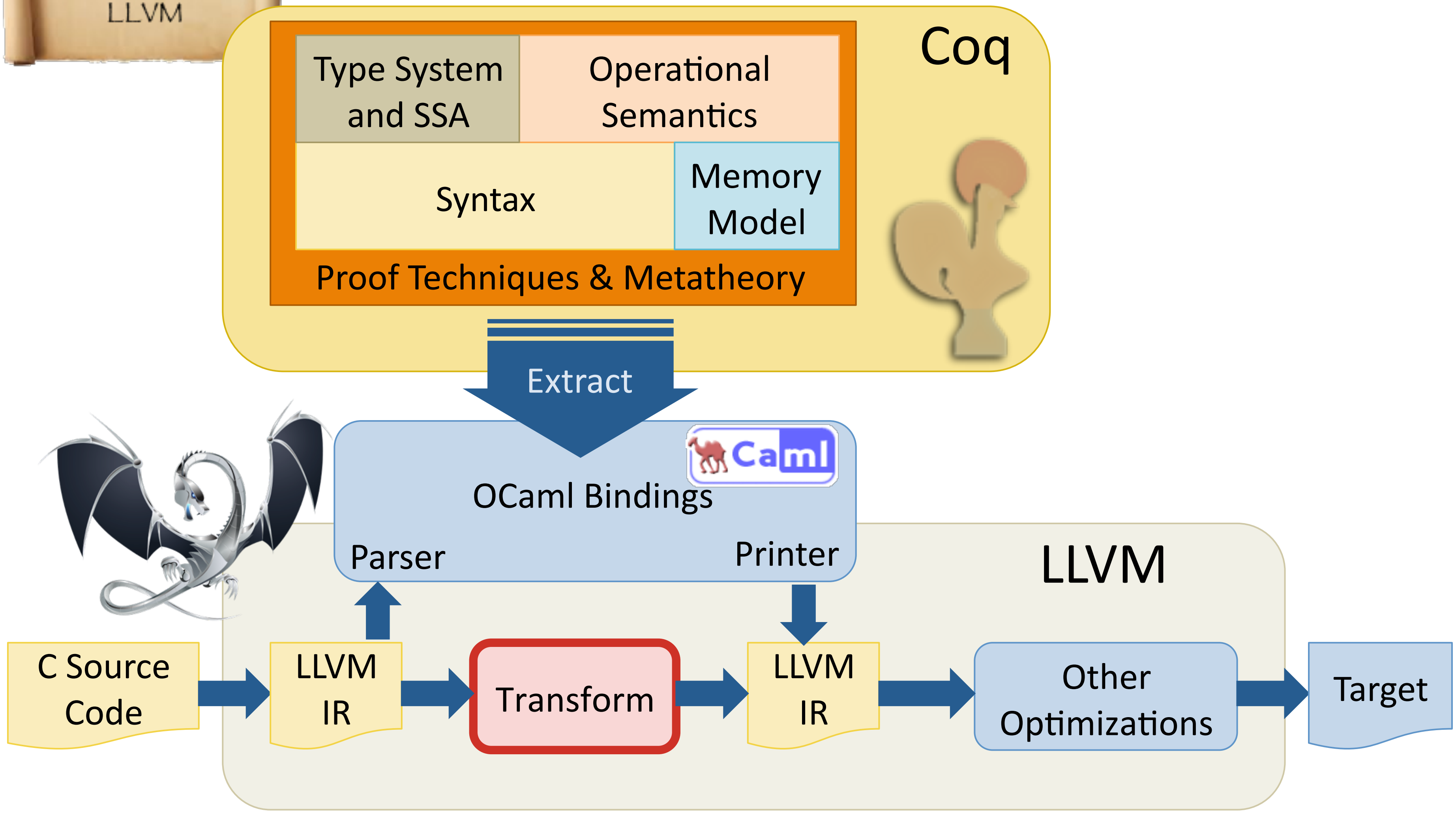
5. the use of type systems for establishing well-behavedness guarantees for all programs in a given language

Can it Scale?

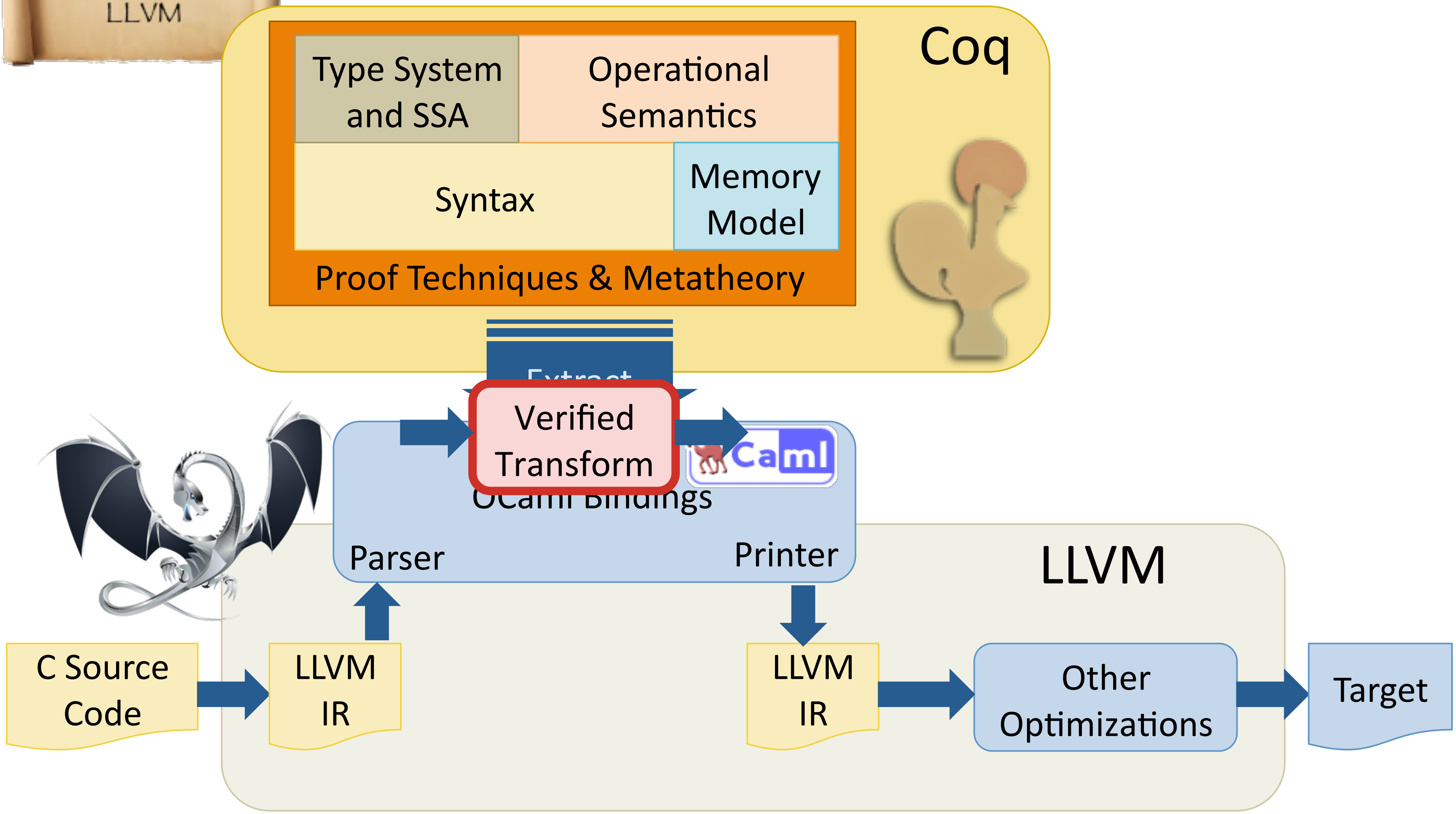
- Use of theorem proving to verify “real” software is still considered to be the bleeding edge of PL research.

- CompCert – fully verified C compiler
  Leroy, INRIA
- Ynot – verified DBMS, web services
  Morrisett, Harvard
- Verified Software Toolchain
  Appel, Princeton
- Bedrock
  Chlipala, MIT
- CertiKOS – certified OS kernel
  Shao & Ford, Yale
- Vellvm – formalized LLVM IR
  Zdancewic, Penn

Vellvm Framework

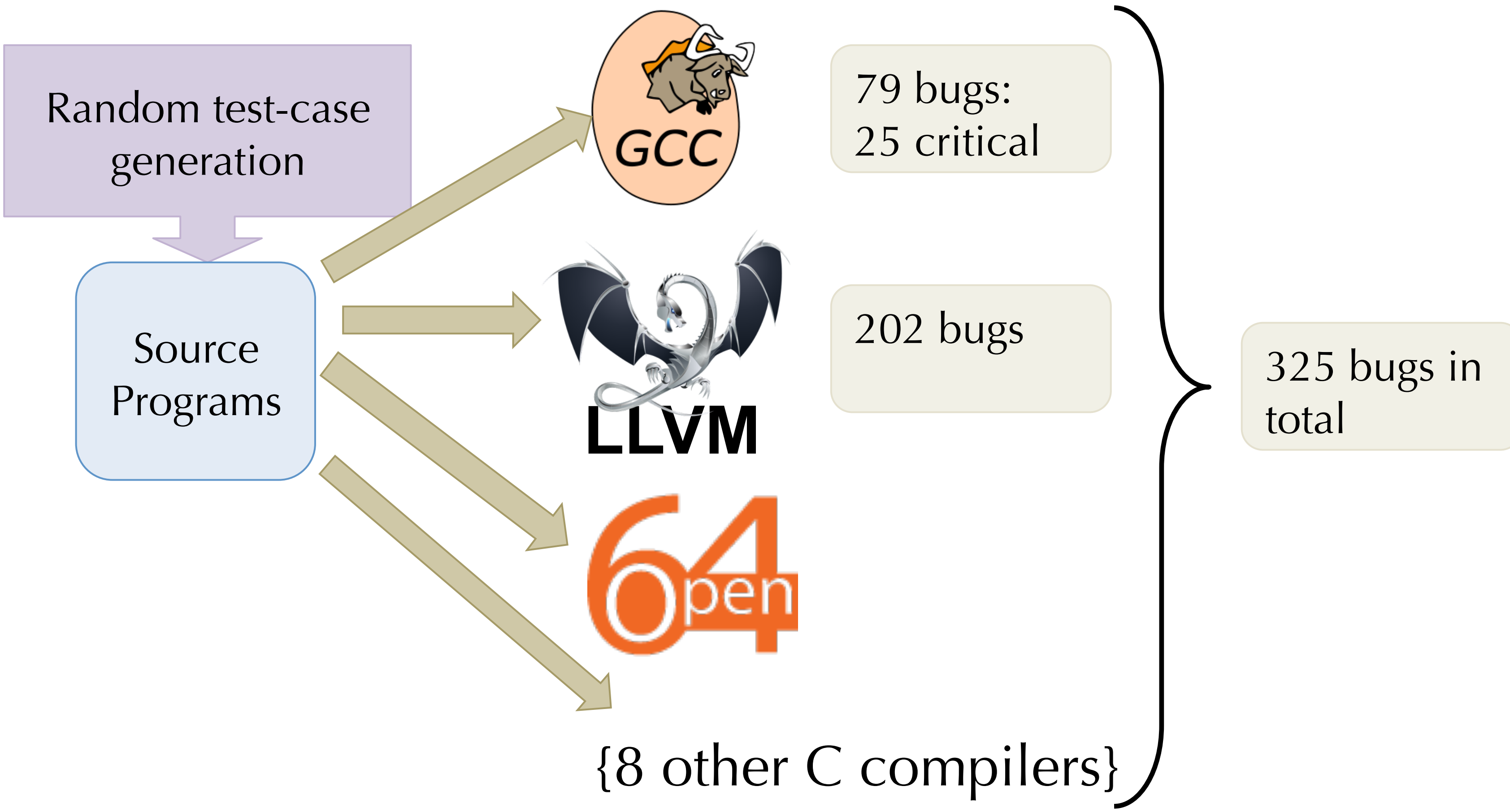


Vellvm Framework



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)

What's in the Software Foundations Text?

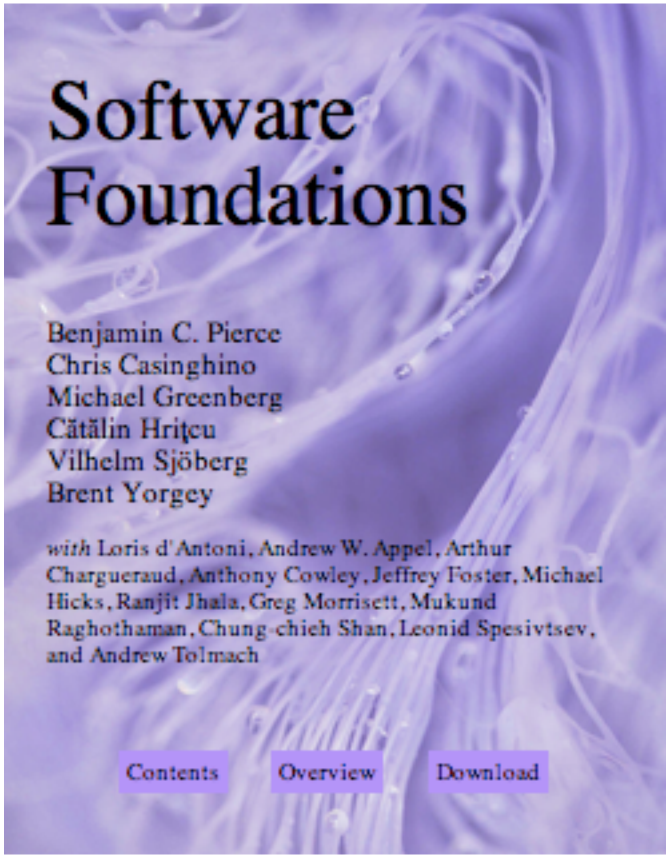
- Foundations
  - Functional programming
  - Constructive logic
  - Logical foundations
  - Proof techniques for inductive definitions
- Semantics
  - Operational semantics
  - Modeling imperative "While" programs
  - Hoare logic for reasoning about program correctness
- Type Systems
  - Simply typed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

COQ

Resources

- Course textbook: *Software Foundations*
  - Electronic edition

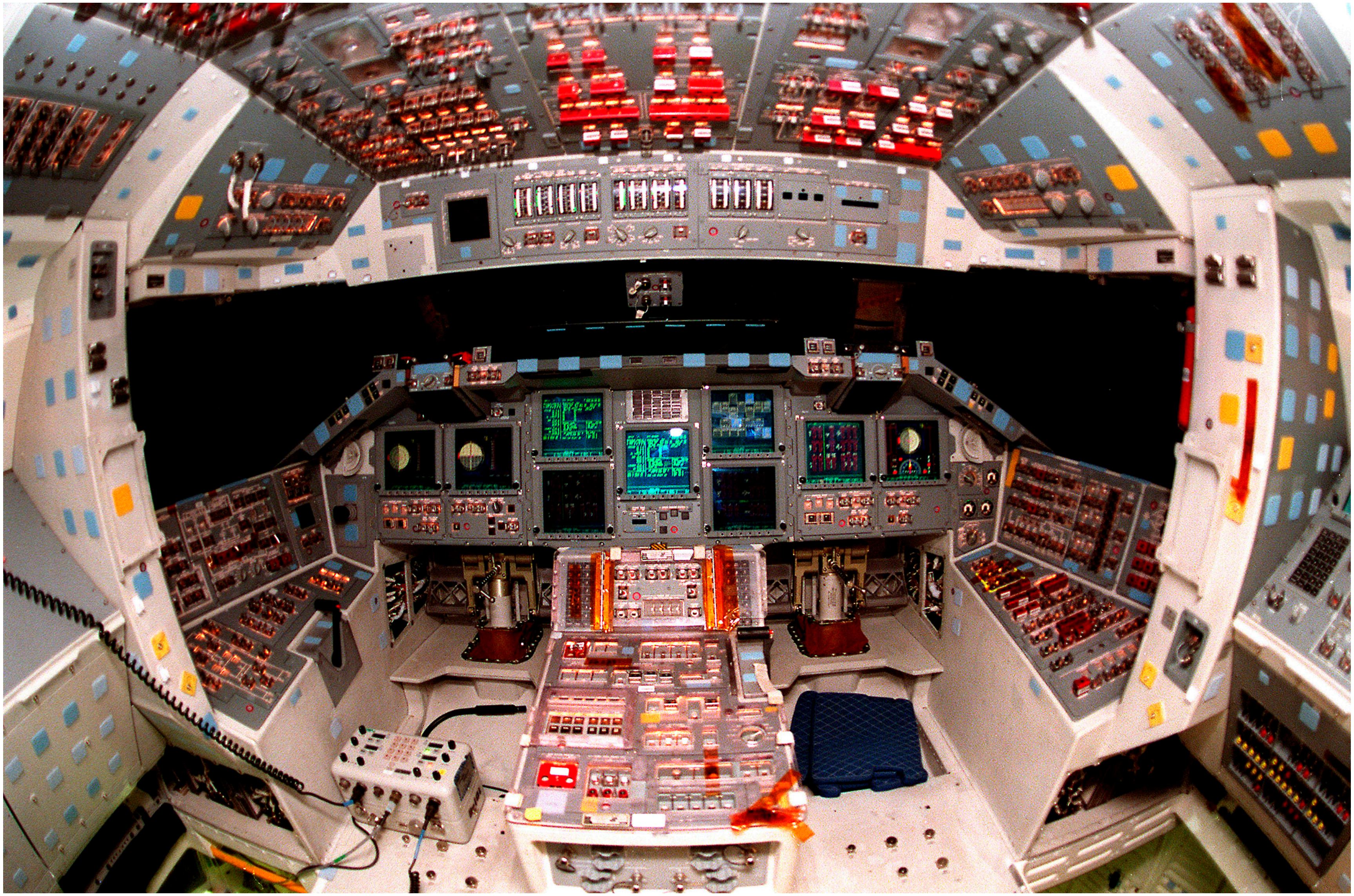
- Additional books:
  - *Types and Programming Languages* (Pierce, 2002 MIT Press)
  - *Interactive Theorem Proving and Program Development* (Bertot and Castéran, 2004 Springer)
  - *Certified Programming with Dependent Types* (Chlipala, electronic edition)



Coq at OPLSS

- We'll use Coq version 8.4
- See the web pages at: coq.inria.fr
- Two different user interfaces
  - CoqIDE – a standalone GUI / editor
  - ProofGeneral – an Emacs-based editing environment
- I will assume that you have Coq up and running…

Coq's Full System



Subset Used in Software Foundations

To start.

By the end.