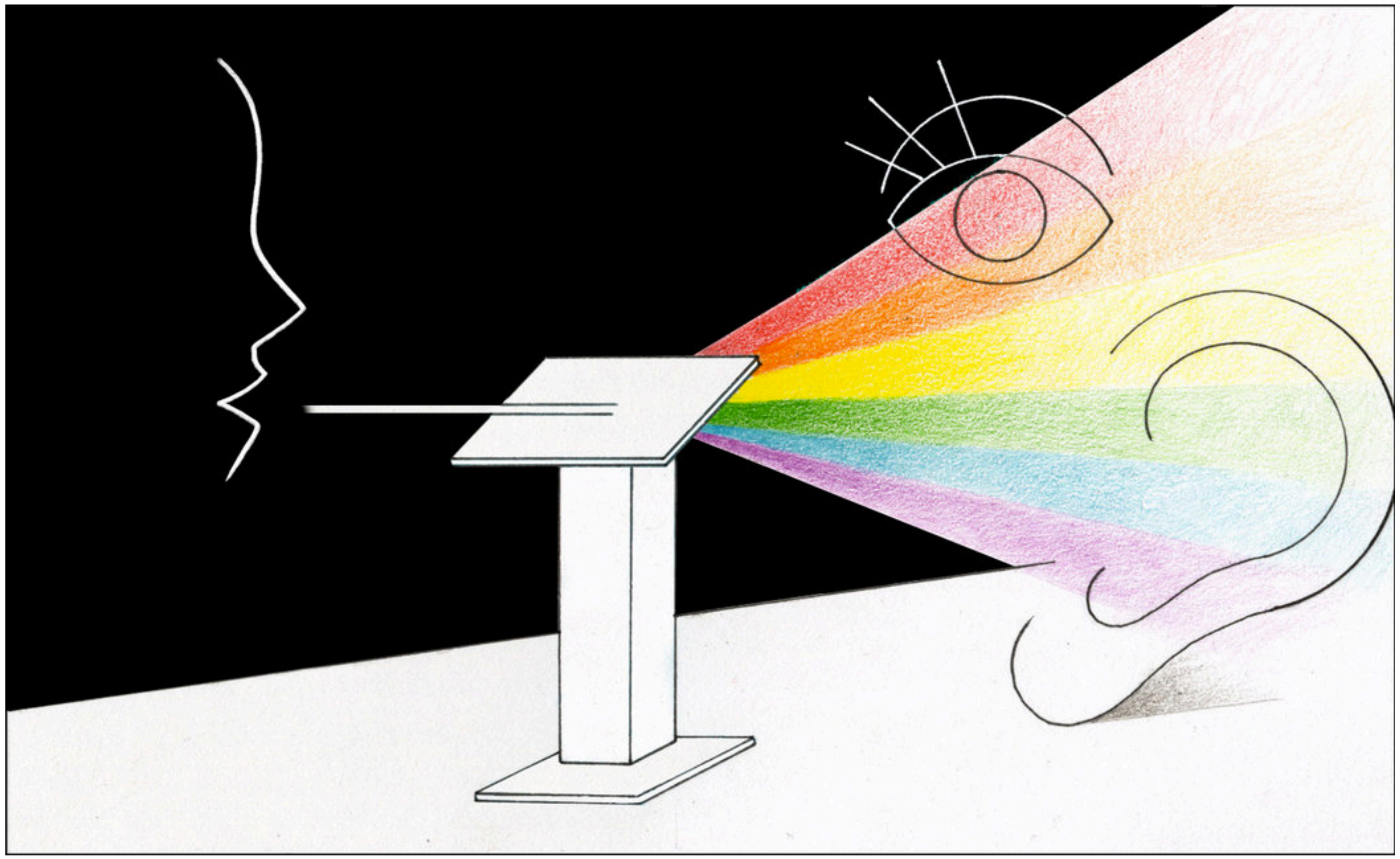
How to Give Talks That People Can Follow



Derek Dreyer

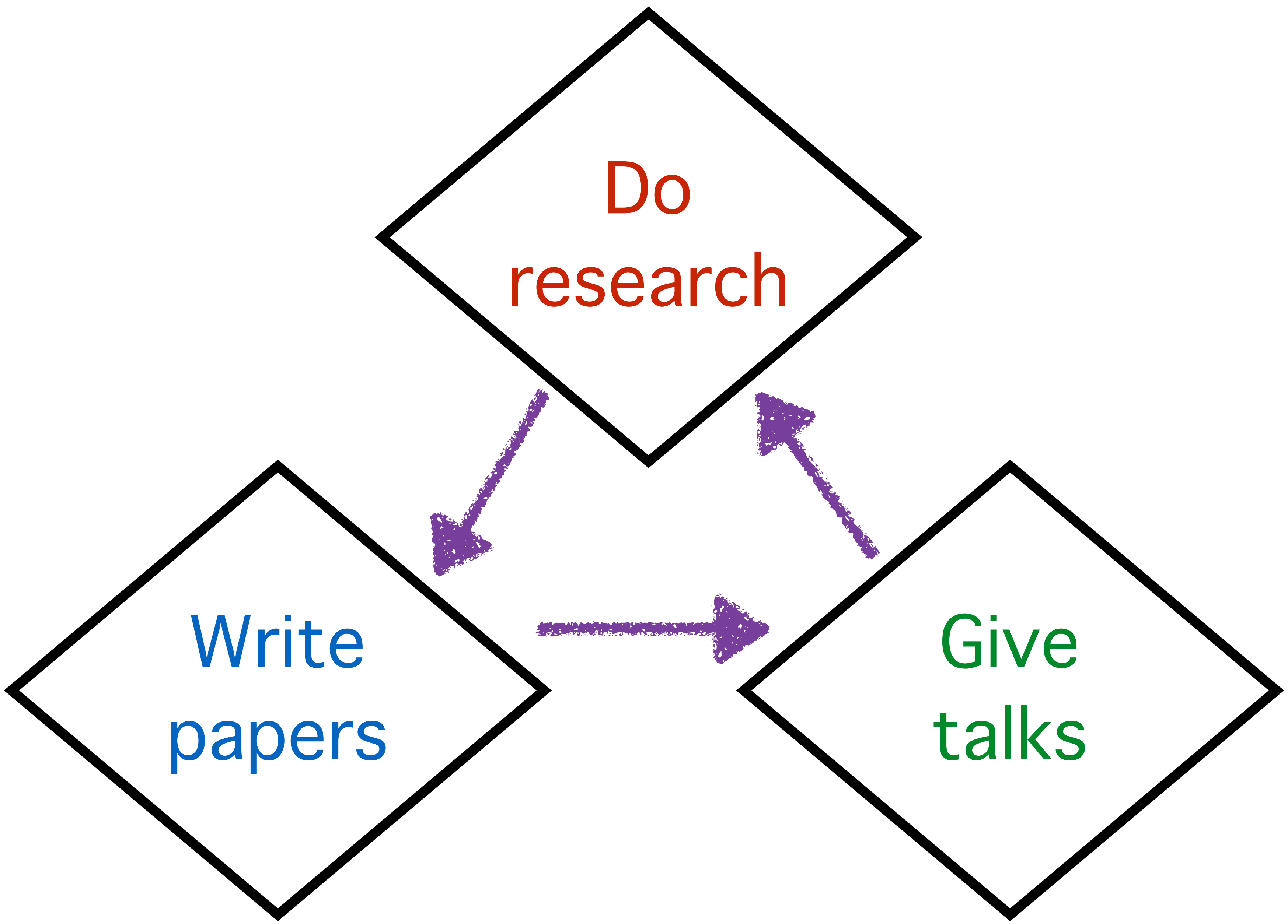
MPI for Software Systems

PLMW@POPL 2017

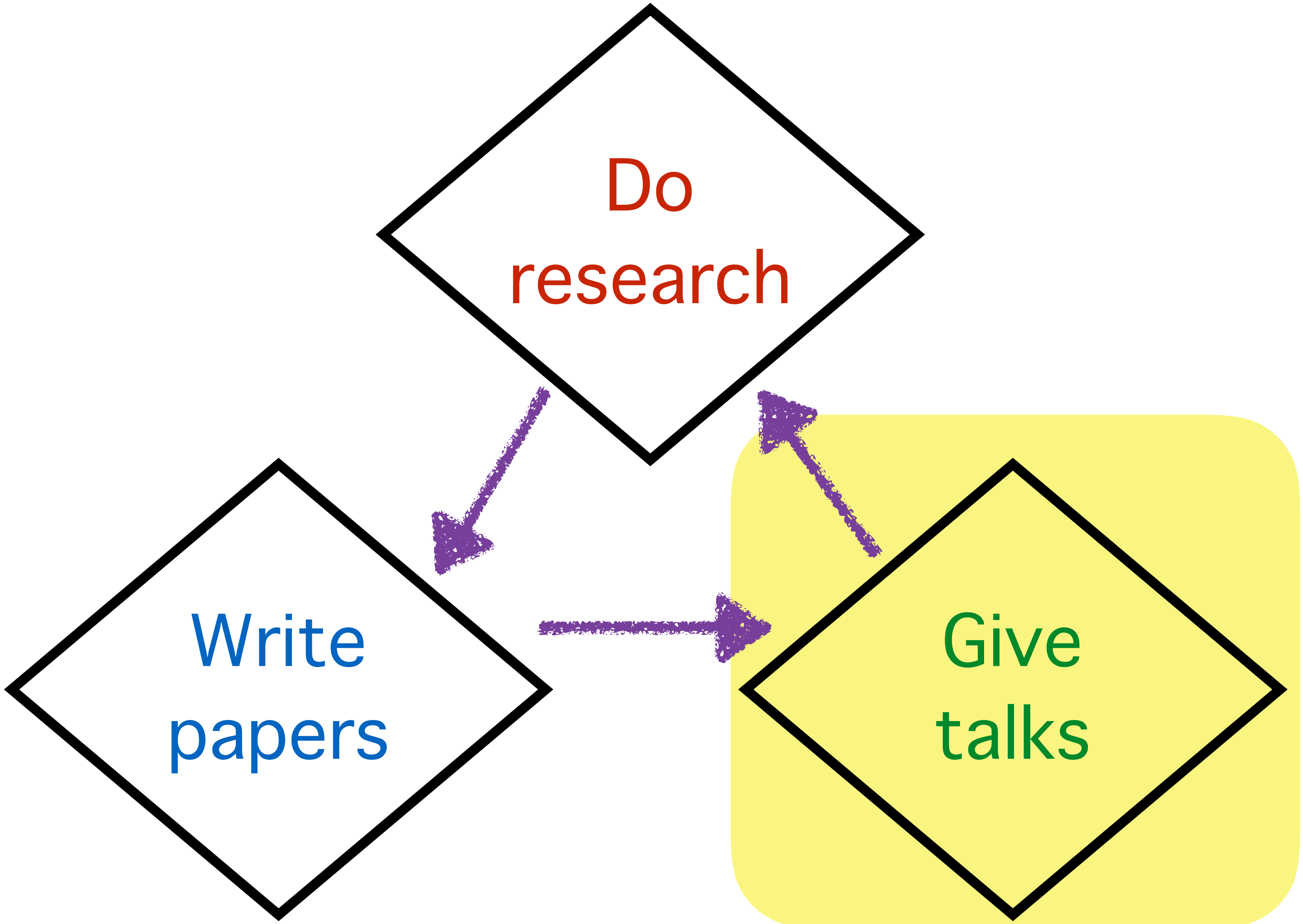
Paris, France

My job as a researcher

My job as a researcher



My job as a researcher



Entertain your audience!

- **Simon Peyton Jones.** *How to give a great research talk*. (MSR Summer School, 2016)
  - "Your mission is to **wake them up**!"
  - "Your most potent weapon, by far, is **your enthusiasm**!"

- **John Hughes.** *Unaccustomed as I am to public speaking*. (PLMW, 2016)
  - "**Put on a show**!"

Entertain your audience!

- **Simon Peyton Jones.** *How to give a great research talk*. (MSR Summer School, 2016)

Good advice, but I don't know how to teach people to be entertaining…

- **John Hughes.** *Unaccustomed as I am to public speaking*. (PLMW, 2016)
  - "**Put on a show**!"

Instead, we'll focus on…

STRUCTURE

STRUCTURE

Jean-Luc Godard

STRUCTURE

Quentin Tarantino

STRUCTURE

A movie should have
a beginning, a middle, and an end…

— Jean-Luc Godard

STRUCTURE

A movie should have
a beginning, a middle, and an end…
…but not necessarily in that order.

— Jean-Luc Godard

STRUCTURE

A ~~movie~~ talk should have
a beginning, a middle, and an end…
…~~but not~~ necessarily in that order.

— ~~Jean-Luc Godard~~ Derek Dreyer

Structure of 20-min. talk

- **Motivation** (~6 minutes)
  - What problem are you solving and why?
- **Contributions & key idea** (~3 minutes)
  - What did you actually do, and what is the key idea behind your solution?
- **Explanation of key idea** (~9 minutes)
- **Conclusion** (~2 minutes)

Motivation

- **Goal**:
  - Explain your problem and why the audience should care about it.
- **Pitfalls**: ???

Motivation

- **Goal**:
  - Explain your problem and why the audience should care about it.

- **Pitfalls**:
  - Fail to clearly state your problem.
  - Take too long to explain your problem.

Stage the motivation

- **First, get to a problem.**
  - Explain a **general** version of your problem (but not too general) **in the first 2 minutes**.

- **Then, get to the problem.**
  - Motivate and **explicitly state** your **specific** problem in the next 4 minutes.
  - Limit discussion of prior work only to what is needed to explain your problem.

Pilsner:

A Compositionally Verified Compiler for a Higher-Order Imperative Language

Georg Neis, Chung-Kil Hur,
Jan-Oliver Kaiser, Craig McLaughlin,
Derek Dreyer, Viktor Vafeiadis

**MPI-SWS (Germany),
Seoul National University,
University of Glasgow**

ICFP 2015
Vancouver

Pilsner:

A Compositionally Verified Comp
Higher-Order Imperative Langua

Georg Neis, Chung-Kil Hur,
Jan-Oliver Kaiser, Craig McLaughlin,
Derek Dreyer, Viktor Vafeiadis

MPI-SWS (Germany),
Seoul National University,
University of Glasgow

ICFP 2015
Vancouver

Compiler Verification

- **Goal**: Formally guarantee that output of compiler "preserves semantics" of input
  - Successes: CompCert, CakeML

- Semantics preservation (traditionally):
  - If $P_T = \text{Compile}(P_S)$,
    then $\text{Behaviors}(P_T) \subseteq \text{Behaviors}(P_S)$.

Compiler Verification

- **Goal**: Formally guarantee th… f compiler "prese…

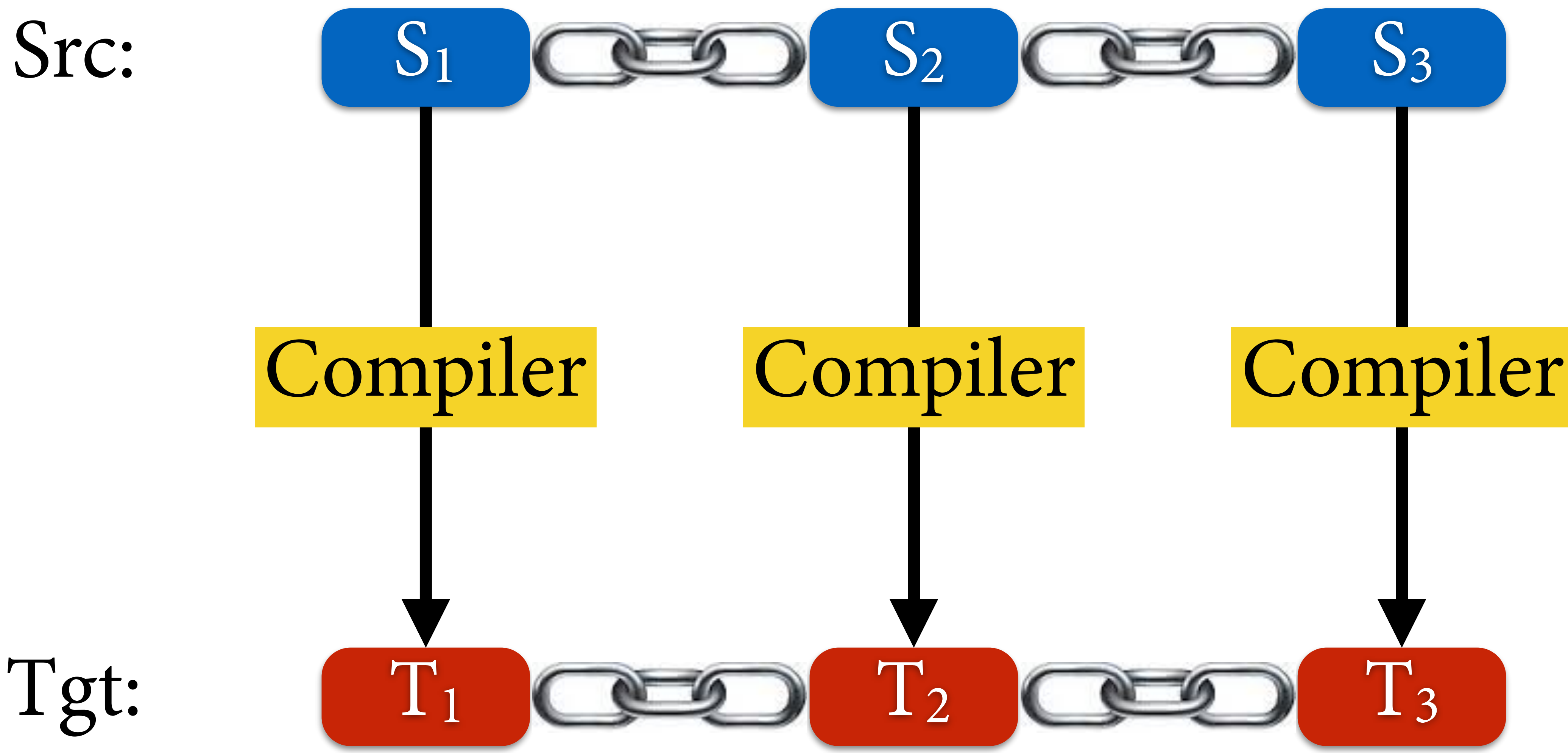
Says nothing about separate compilation!

- S… ervation (traditionally):
  - If $P_T = \text{Compile}(P_S)$,
    then $\text{Behaviors}(P_T) \subseteq \text{Behaviors}(P_S)$.

Compositional Compiler Verification

- Goal: Define semantics preservation for *separately compiled* modules
- Three key criteria (in our view):
  - Modularity
  - Flexibility
  - Transitivity

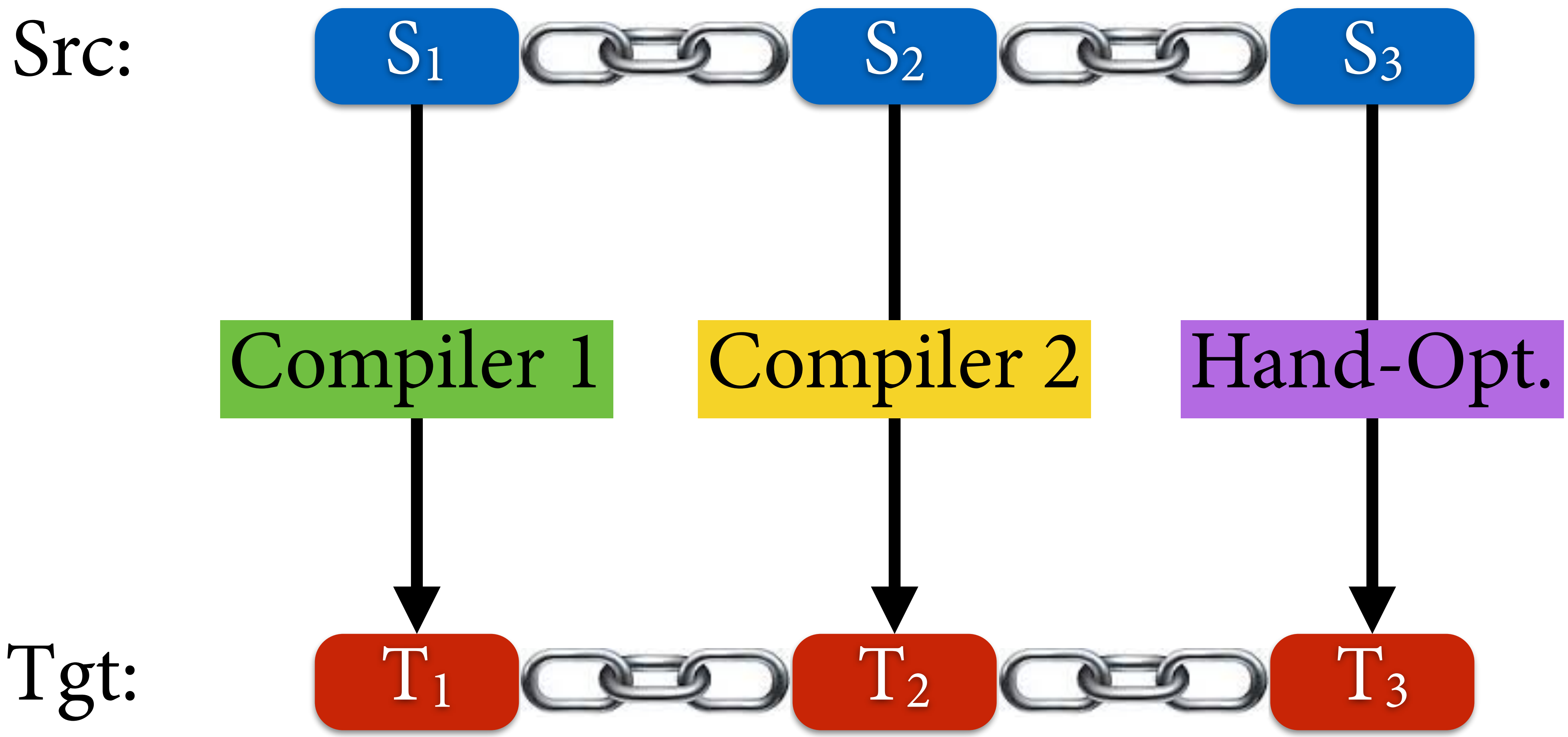
Modularity



Semantics preservation preserved by linking:

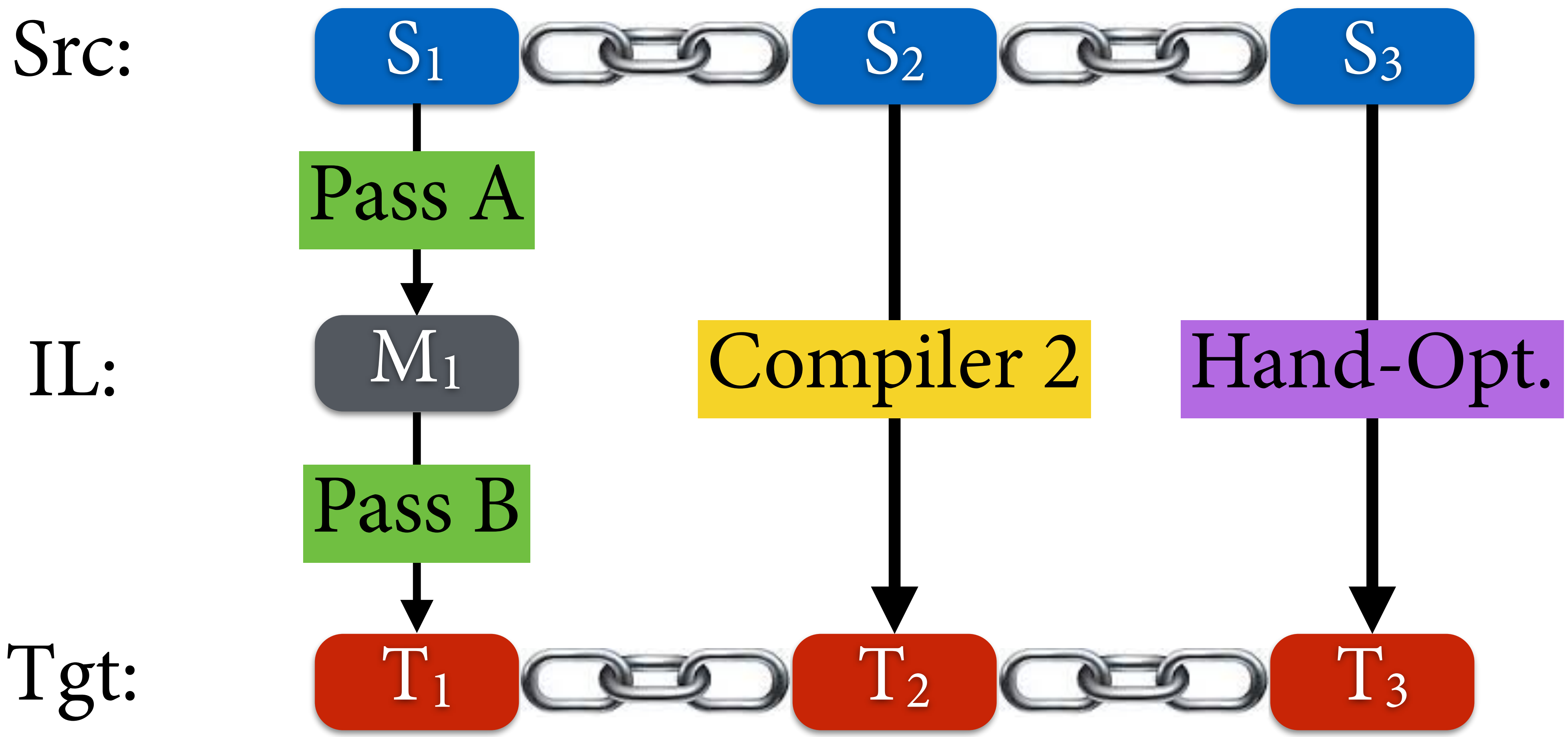
$\text{Behaviors}(T_1 \bullet T_2 \bullet T_3) \subseteq \text{Behaviors}(S_1 \bullet S_2 \bullet S_3)$

Flexibility



Can link results of *different* verified compilers, together with hand-optimized modules

Transitivity



Can verify individual passes of a compiler, then link the results transitively

Prior Work

A number of **modular** techniques proposed, but all are either:

1. **Not flexible enough**
2. **Not transitive**

Takeaway:

First, get to a problem.
Then, get to the problem.

Contributions & key idea

- **Goal**:
  - State what you did to solve the problem.
  - Briefly describe key idea of your solution.

- **Pitfall**: ???

Contributions & key idea

- **Goal**:
  - State what you did to solve the problem.
  - Briefly describe key idea of your solution.
- **Pitfall**:
  - Fail to have this section in your talk.

Don't blow this golden opportunity!

- **Proudly state your contributions.**
  - After the motivation, the audience eagerly wants to hear what you did. Tell them!

- **Have a key idea.**
  - It will give audience a take-home message, and give focus to the rest of your talk.

Prior Work

A number of **modular** techniques proposed, but all are either:

1. **<u>Not</u> flexible enough**
2. **<u>Not</u> transitive**

Our Contributions

Parametric Inter-Language Simulations (PILS):

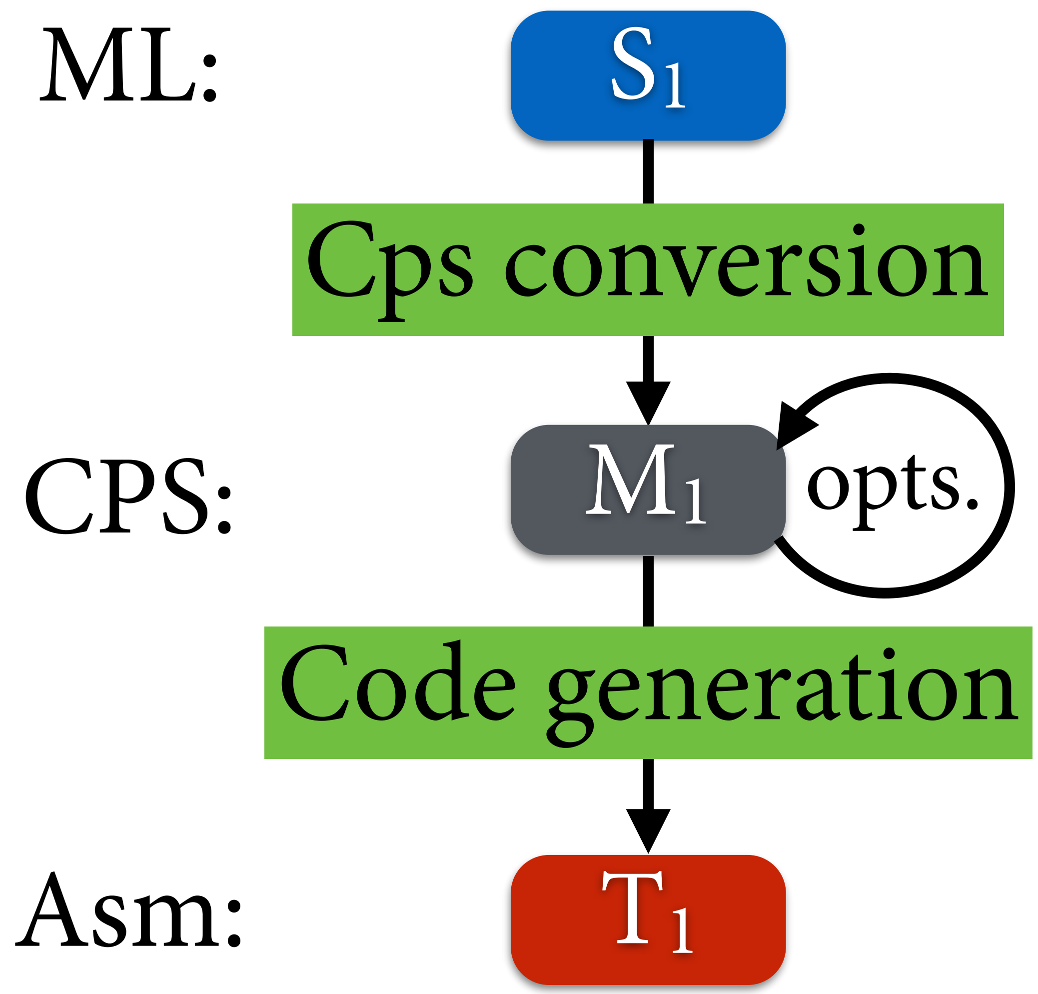
- New way to define semantics preservation
- Modular, flexible, ***and*** transitive

Pilsner:

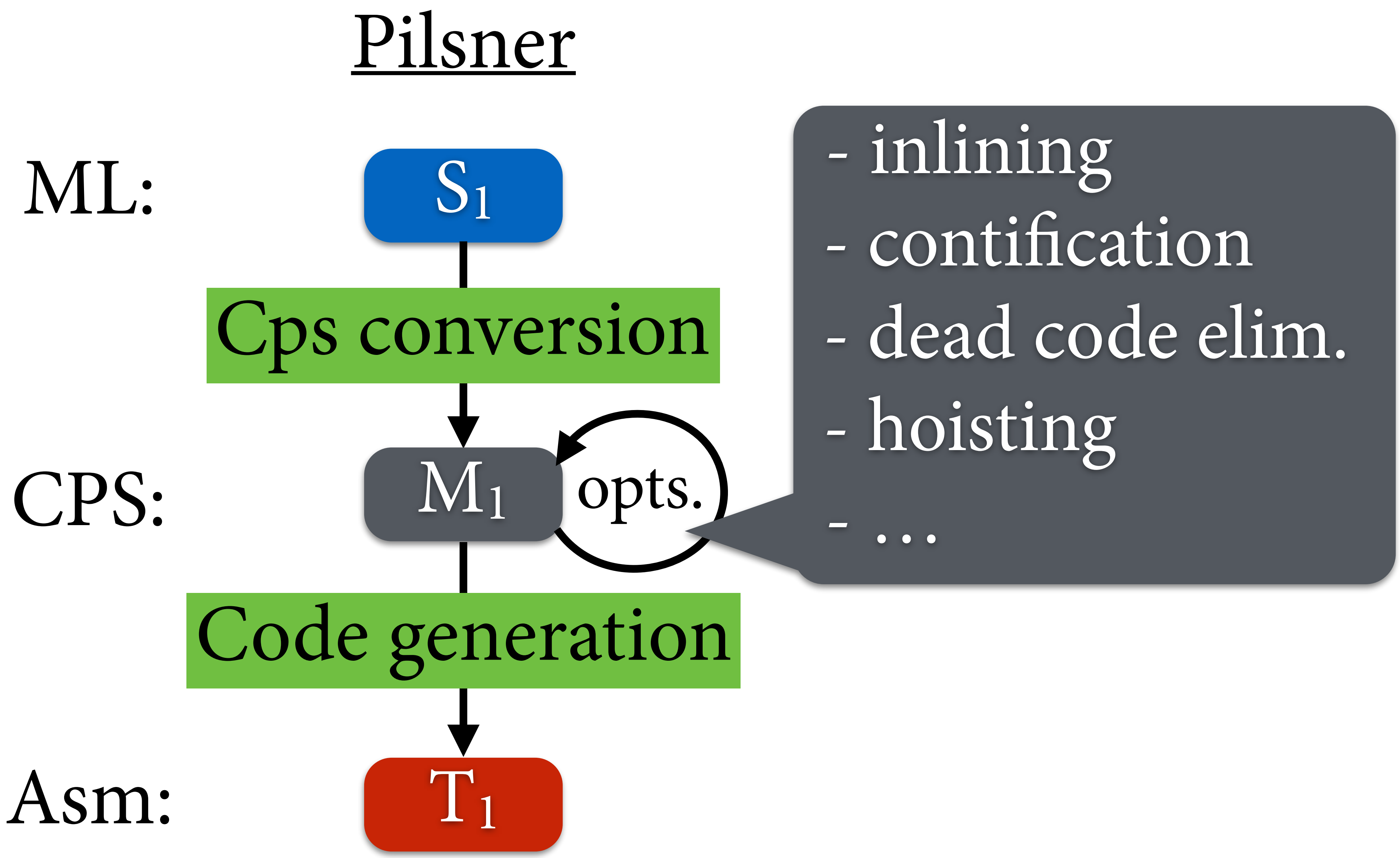
- The ***first*** compositionally verified multi-pass compiler for an ML-like language
- Verified using PILS in Coq!

Our Contributions

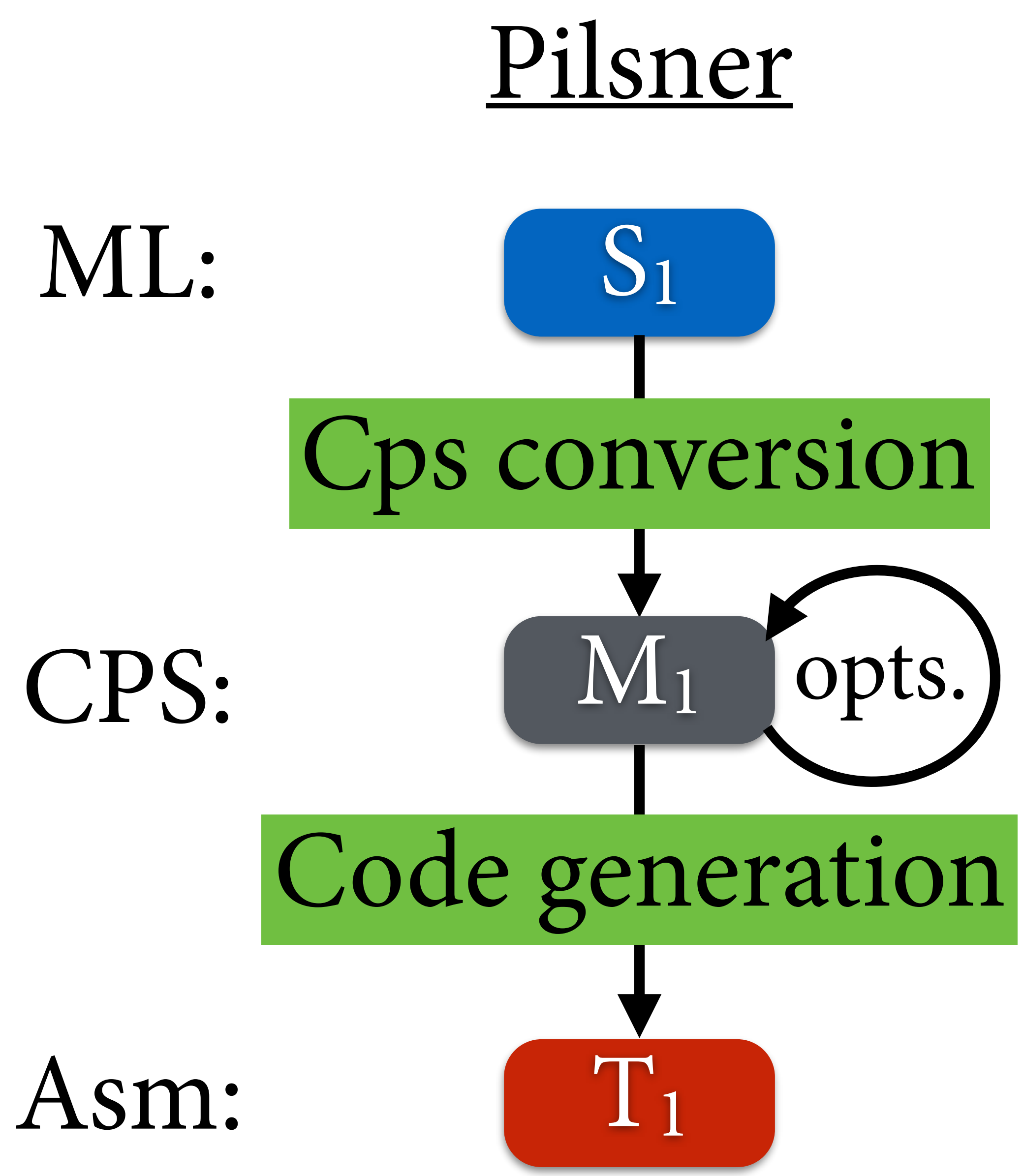
Pilsner



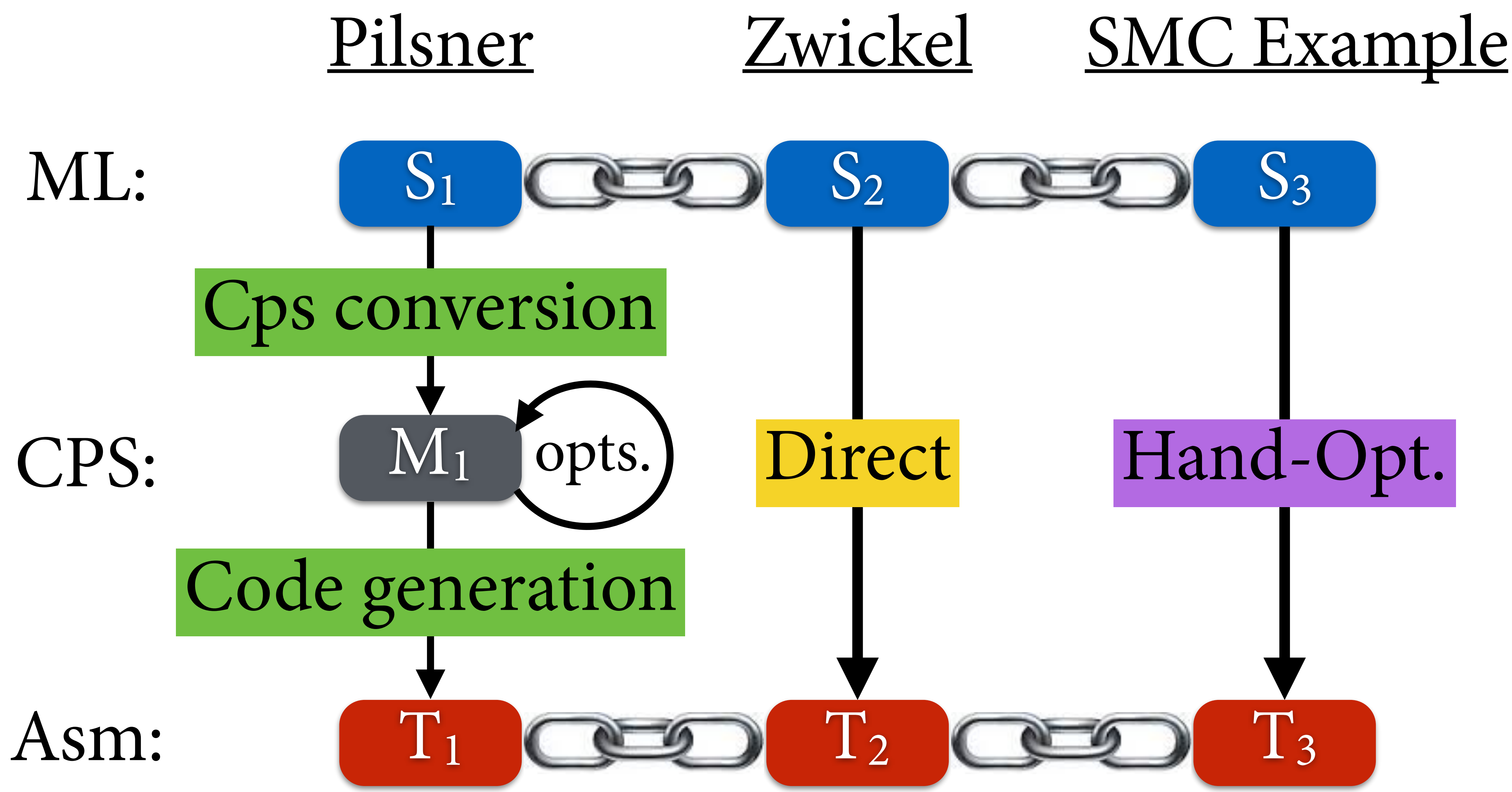
Our Contributions



Our Contributions



Our Contributions



Our Contributions

Parametric Inter-Language Simulations (PILS):

- New way to define semantics preservation
- Modular, flexible, ***and*** transitive

Pilsner:

- The ***first*** compositionally verified multi-pass compiler for an ML-like language
- Verified using PILS in Coq!

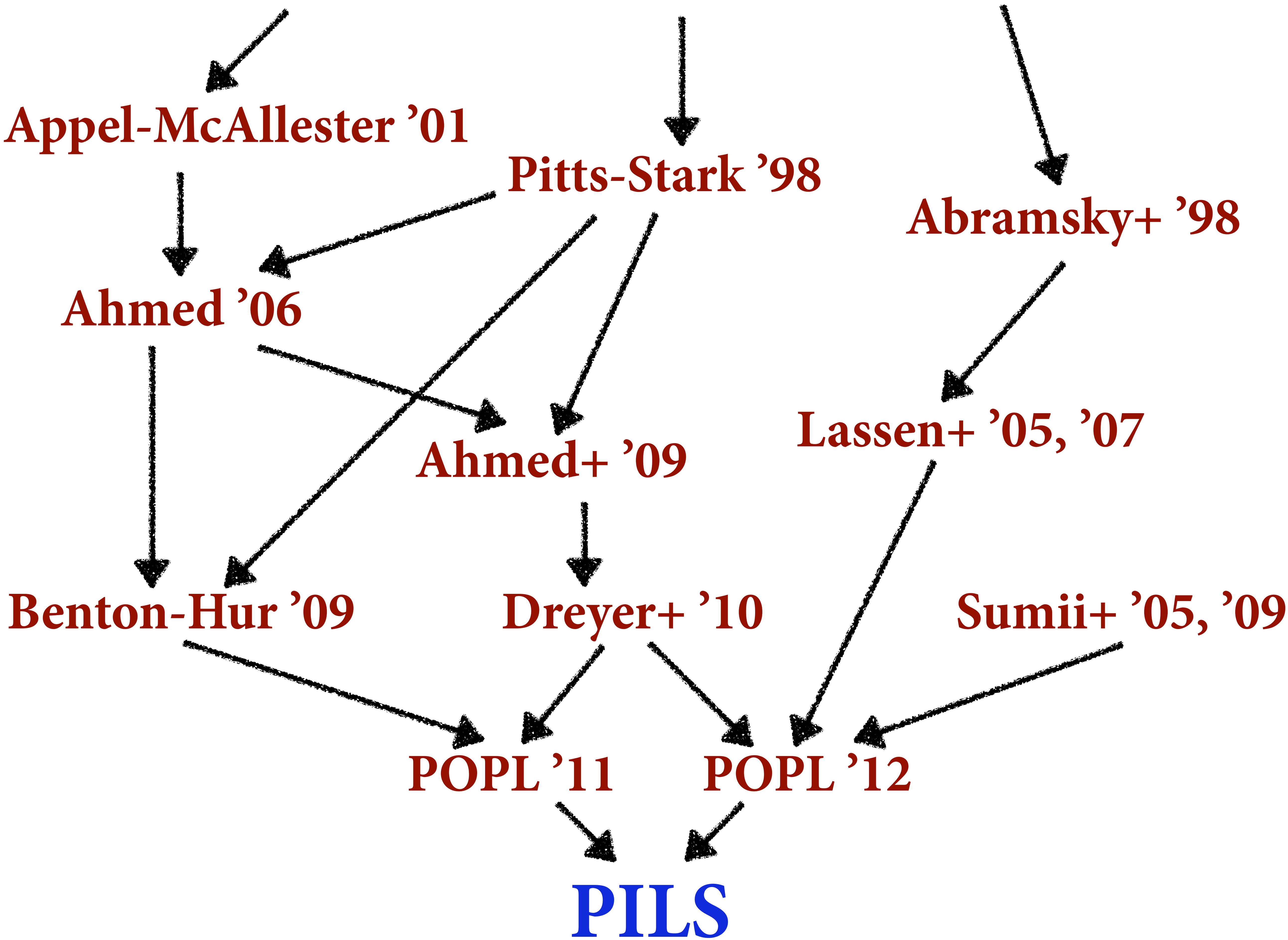
Our Contributions

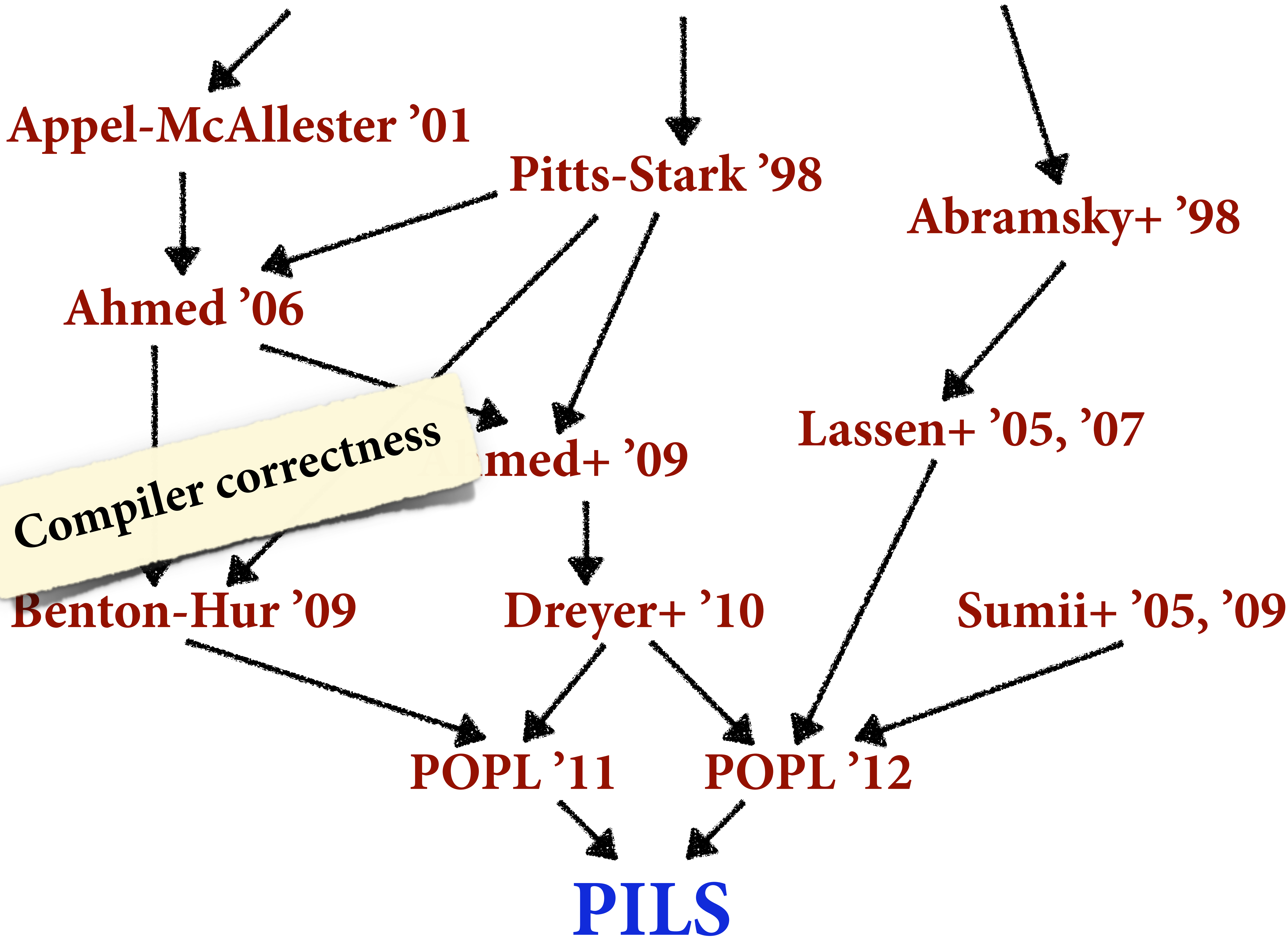
Parametric Inter-Language Simulations (PILS):

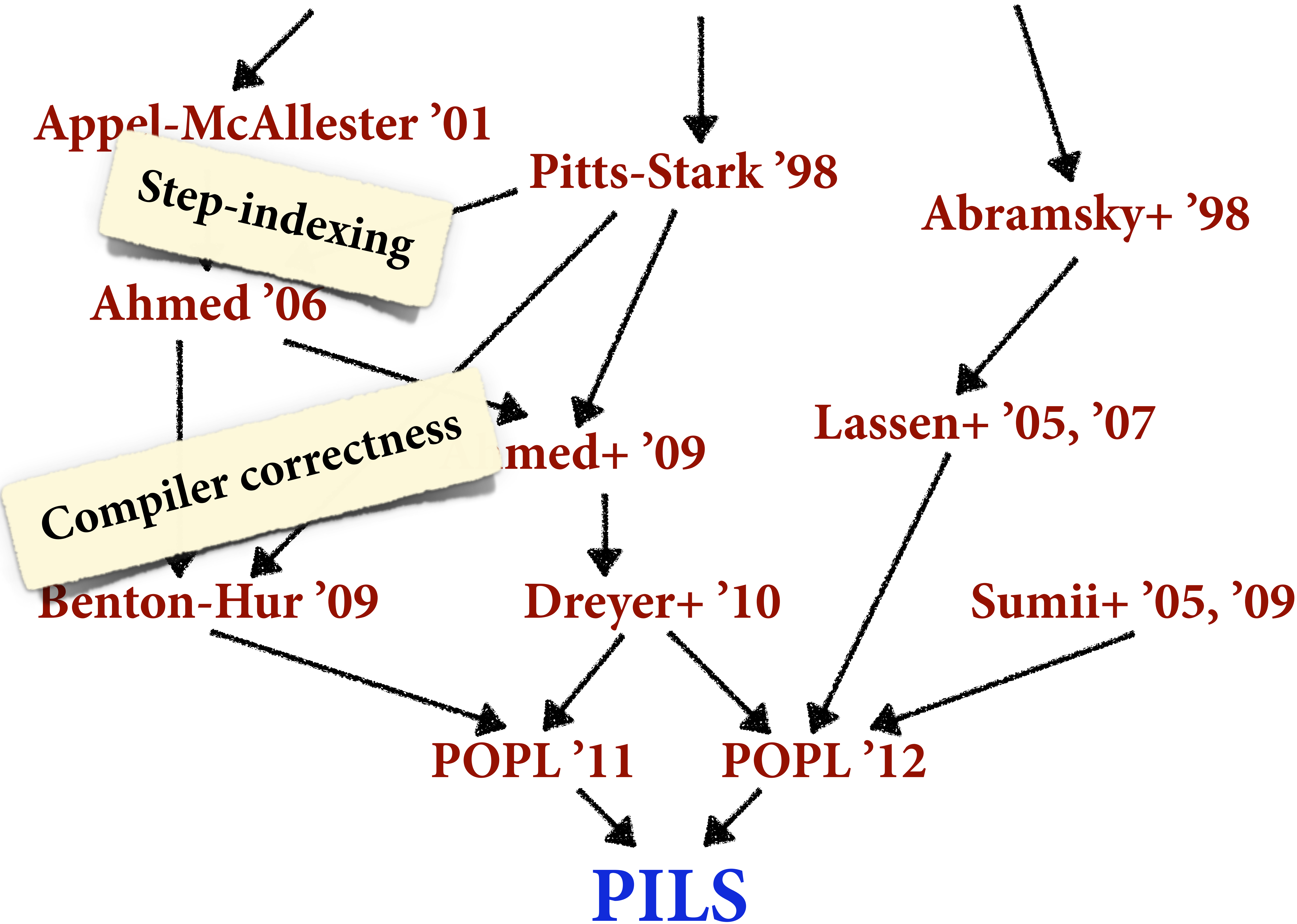
- New way to define semantics preservation
- Modular, flexible, ***and*** transitive

Pilsner:

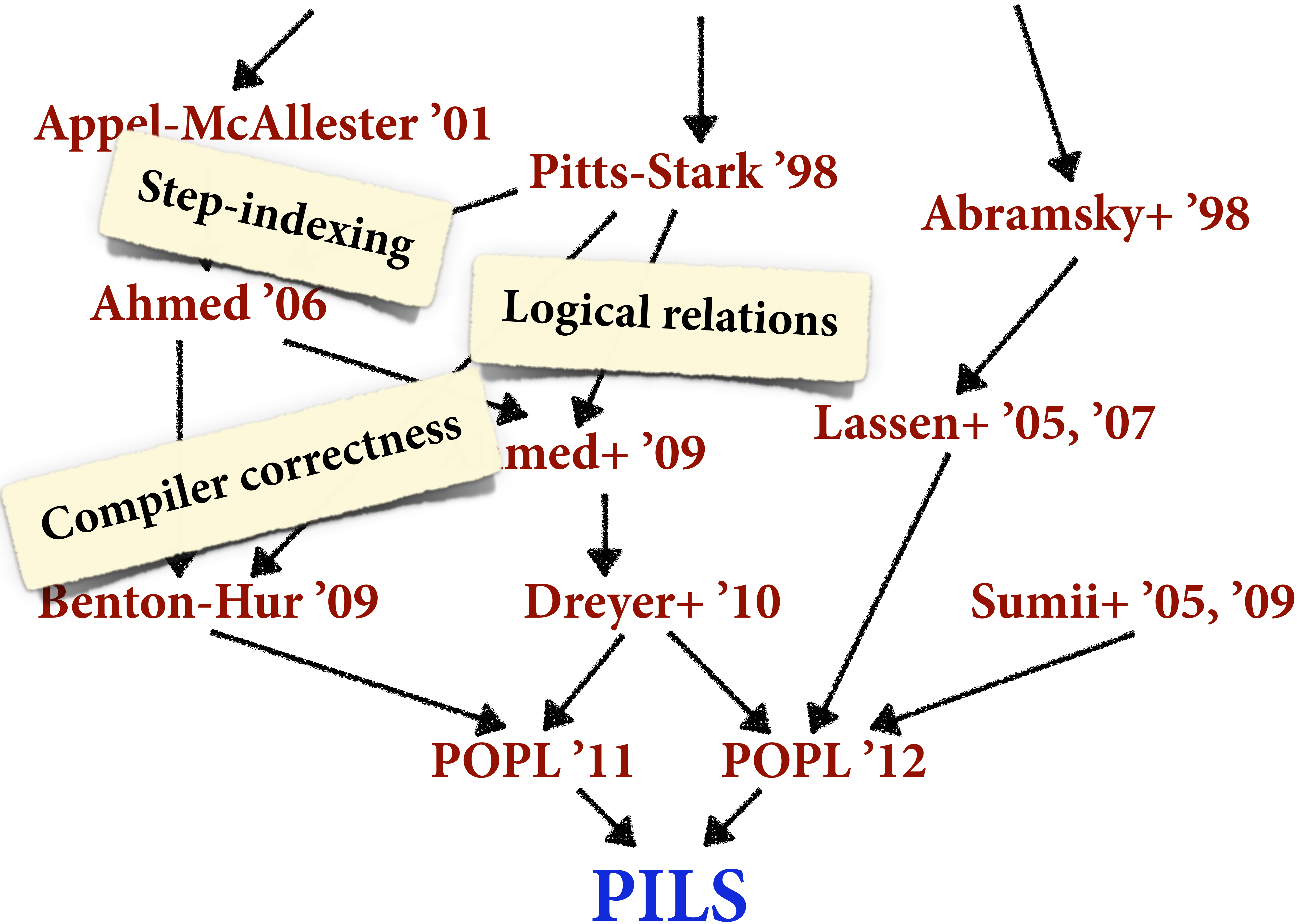
- The ***first*** compositionally verified multi-pass compiler for an ML-like language
- Verified using PILS in Coq!

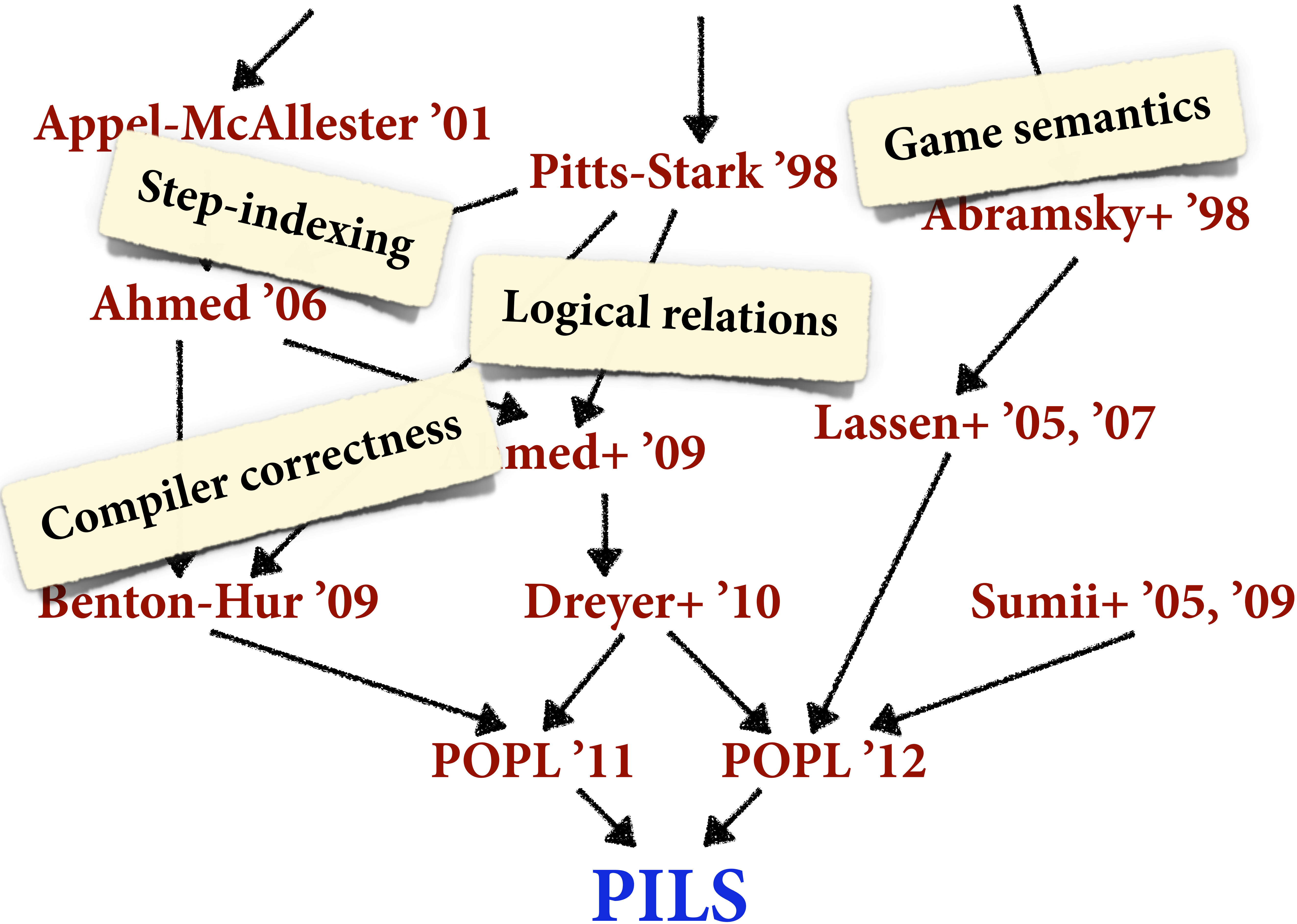

Appel-McAllester ’01
Pitts-Stark ’98
Abramsky+ ’98
Ahmed ’06
Lassen+ ’05, ’07
Ahmed+ ’09
Benton-Hur ’09
Dreyer+ ’10
Sumii+ ’05, ’09
POPL ’11
POPL ’12
PILS

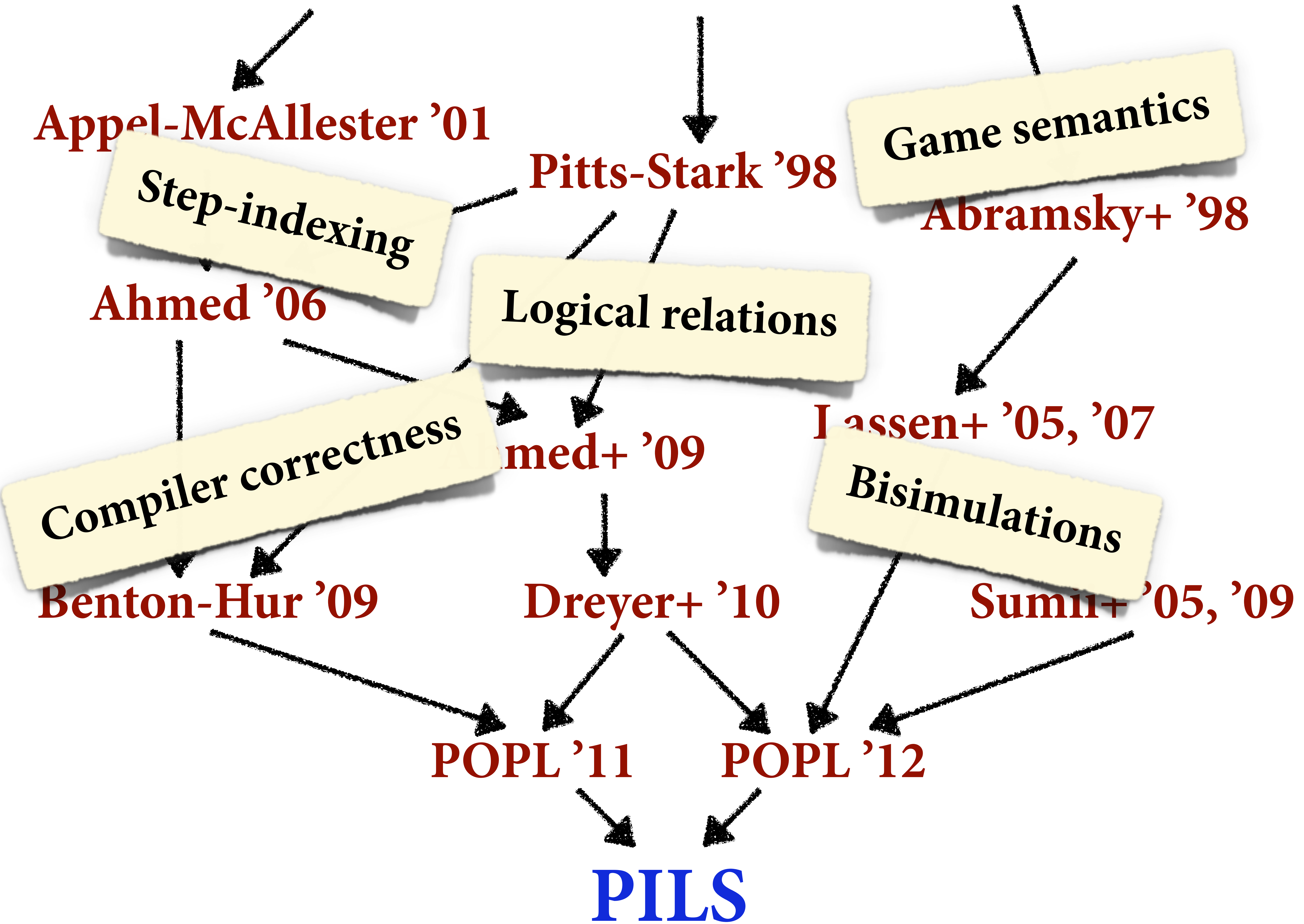

Appel-McAllester '01
Pitts-Stark '98
Abramsky+ '98
Ahmed '06
Lassen+ '05, '07
Compiler correctness
Benton-Hur '09
Dreyer+ '10
Sumii+ '05, '09
POPL '11
POPL '12
PILS



Appel-McAllester ’01
Pitts-Stark ’98
Abramsky+ ’98
Step-indexing
Ahmed ’06
Compiler correctness
med+ ’09
Lassen+ ’05, ’07
Benton-Hur ’09
Dreyer+ ’10
Sumii+ ’05, ’09
POPL ’11
POPL ’12
PILS


Appel-McAllester ’01
Pitts-Stark ’98
Abramsky+ ’98
Step-indexing
Ahmed ’06
Logical relations
Lassen+ ’05, ’07
Compiler correctness
med+ ’09
Benton-Hur ’09
Dreyer+ ’10
Sumii+ ’05, ’09
POPL ’11
POPL ’12
PILS


Appel-McAllester '01
Step-indexing
Pitts-Stark '98
Game semantics
Abramsky+ '98
Ahmed '06
Logical relations
Lassen+ '05, '07
Compiler correctness
med+ '09
Benton-Hur '09
Dreyer+ '10
Sumii+ '05, '09
POPL '11
POPL '12
PILS



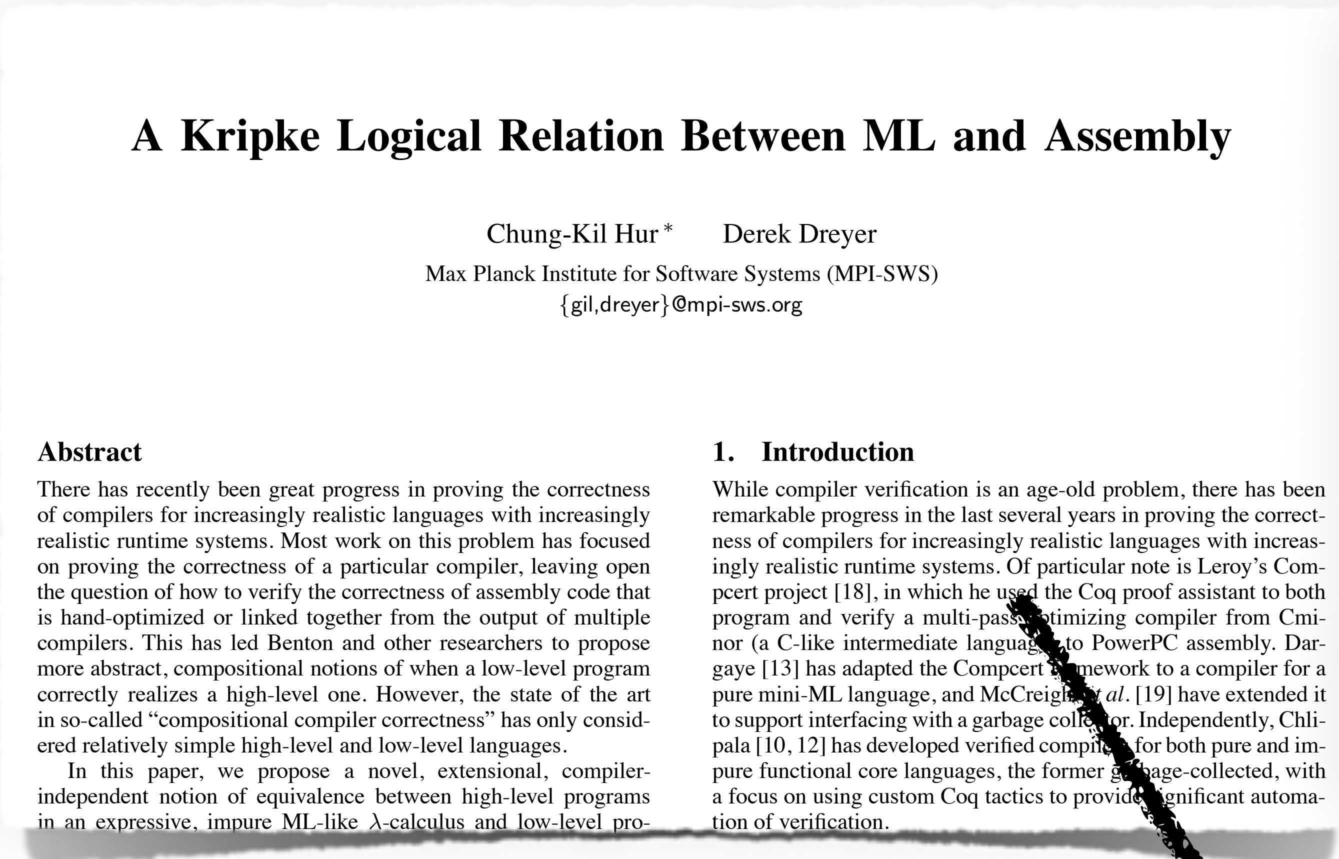
Appel-McAllester ’01
Pitts-Stark ’98
Game semantics
Step-indexing
Abramsky+ ’98
Ahmed ’06
Logical relations
med+ ’09
assen+ ’05, ’07
Compiler correctness
Bisimulations
Benton-Hur ’09
Dreyer+ ’10
’05, ’09
POPL ’11
POPL ’12
PILS

POPL ’11 POPL ’12

PILS

POPL '11

POPL '12



A Kripke Logical Relation Between ML and Assembly

Chung-Kil Hur* Derek Dreyer
Max Planck Institute for Software Systems (MPI-SWS)
{gil,dreyer}@mpi-sws.org

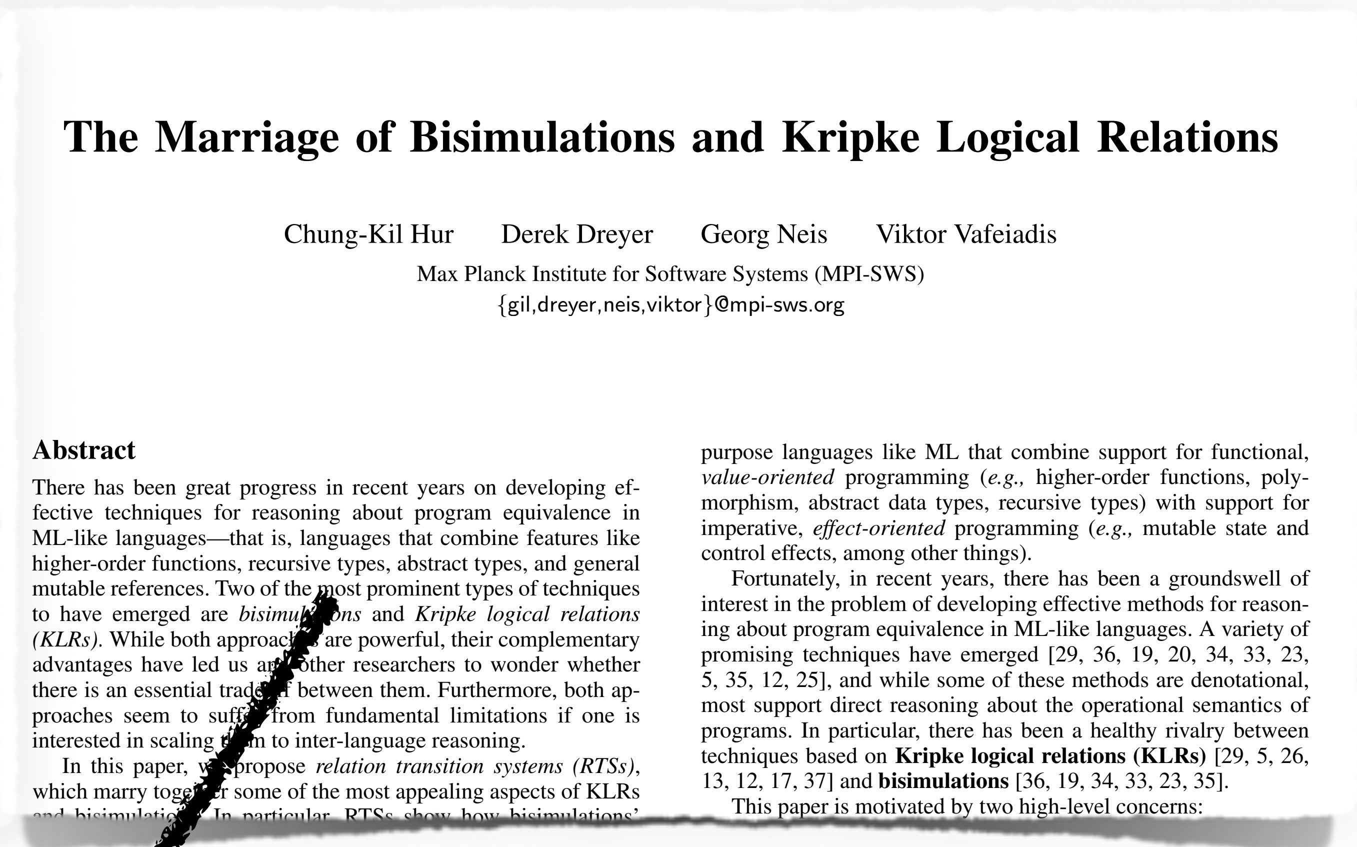
Abstract

There has recently been great progress in proving the correctness of compilers for increasingly realistic languages with increasingly realistic runtime systems. Most work on this problem has focused on proving the correctness of a particular compiler, leaving open the question of how to verify the correctness of assembly code that is hand-optimized or linked together from the output of multiple compilers. This has led Benton and other researchers to propose more abstract, compositional notions of when a low-level program correctly realizes a high-level one. However, the state of the art in so-called "compositional compiler correctness" has only considered relatively simple high-level and low-level languages.

In this paper, we propose a novel, extensional, compiler-independent notion of equivalence between high-level programs in an expressive, impure ML-like λ-calculus and low-level pro-

1. Introduction

While compiler verification is an age-old problem, there has been remarkable progress in the last several years in proving the correctness of compilers for increasingly realistic languages with increasingly realistic runtime systems. Of particular note is Leroy's Compcert project [18], in which he used the Coq proof assistant to both program and verify a multi-pass optimizing compiler from Cminor (a C-like intermediate language) to PowerPC assembly. Dargaye [13] has adapted the Compcert framework to a compiler for a pure mini-ML language, and McCreight *et al.* [19] have extended it to support interfacing with a garbage collector. Independently, Chlipala [10, 12] has developed verified compilers for both pure and impure functional core languages, the former garbage-collected, with a focus on using custom Coq tactics to provide significant automation of verification.



The Marriage of Bisimulations and Kripke Logical Relations

Chung-Kil Hur Derek Dreyer Georg Neis Viktor Vafeiadis
Max Planck Institute for Software Systems (MPI-SWS)
{gil,dreyer,neis,viktor}@mpi-sws.org

Abstract

There has been great progress in recent years on developing effective techniques for reasoning about program equivalence in ML-like languages—that is, languages that combine features like higher-order functions, recursive types, abstract types, and general mutable references. Two of the most prominent types of techniques to have emerged are *bisimulations* and *Kripke logical relations (KLRs)*. While both approaches are powerful, their complementary advantages have led us and other researchers to wonder whether there is an essential tradeoff between them. Furthermore, both approaches seem to suffer from fundamental limitations if one is interested in scaling them to inter-language reasoning.

In this paper, we propose *relation transition systems (RTSs)*, which marry together some of the most appealing aspects of KLRs and bisimulations. In particular, RTSs show how bisimulations'

purpose languages like ML that combine support for functional, *value-oriented* programming (*e.g.,* higher-order functions, polymorphism, abstract data types, recursive types) with support for imperative, *effect-oriented* programming (*e.g.,* mutable state and control effects, among other things).

Fortunately, in recent years, there has been a groundswell of interest in the problem of developing effective methods for reasoning about program equivalence in ML-like languages. A variety of promising techniques have emerged [29, 36, 19, 20, 34, 33, 23, 5, 35, 12, 25], and while some of these methods are denotational, most support direct reasoning about the operational semantics of programs. In particular, there has been a healthy rivalry between techniques based on **Kripke logical relations (KLRs)** [29, 5, 26, 13, 12, 17, 37] and **bisimulations** [36, 19, 34, 33, 23, 35].

This paper is motivated by two high-level concerns:

PILS

Takeaway:

Proudly state your contributions.

Have a key idea.

Explanation of key idea

- **Goal**:
  - Explain your key idea in detail.
- **Pitfall**: ???

Explanation of key idea

- **Goal**:
  - Explain your key idea in detail.
- **Pitfall**:
  - Fail to properly structure a long section.

Talklets

- **Break explanation of key idea into talklets.**
  - More digestible units of story (2-4 min.)
  - But just having talklets is not enough…
- **Use transitions between talklets to remind the audience of the big picture.**
  - Summarize the point of the last talklet and how it connects to the next one.

Summary

- First, get to <u>a</u> problem.
- Then, get to <u>the</u> problem.
- Proudly state your contributions.
- Have a key idea.
- Break explanation of key idea into talklets.
- Use transitions between talklets to remind the audience of the big picture.