Setting the Stage: on the Mechanized Verification of a Compiler

Compilation and Program Analysis (#11)

2024/2025

Yannick Zakowski

Inria

Introduction

Or How I Learned to Stop Worrying…

In 1956, Nikita Khrushchev is addressing western diplomats in Moscow:

Or How I Learned to Stop Worrying…

In 1956, Nikita Khrushchev is addressing western diplomats in Moscow:

"We will bury you"

Or How I Learned to Stop Worrying…

In 1956, Nikita Khrushchev is addressing western diplomats in Moscow:

“We will bury you”

It appears that the original Russian sentence something closer to

“We shall outlive you”

Or How I Learned to Stop Worrying…

In 1956, Nikita Khrushchev is addressing western diplomats in Moscow:

“We will bury you”

It appears that the original Russian sentence something closer to

“We shall outlive you”

Khrushchev had a buggy translator!

Traduttore, traditore[1]

A nightmare scenario…

[1] The use of this quote in this context is stolen from Xavier Leroy's excellent course at Collège de France. It's available online!

Traduttore, traditore[1]

A nightmare scenario…

Our algorithm satisfies the specification

[1] The use of this quote in this context is stolen from Xavier Leroy's excellent course at Collège de France. It's available online!

Traduttore, traditore[1]

A nightmare scenario…

Our algorithm satisfies the specification

Better! Our implementation satisfies the specification

[1] The use of this quote in this context is stolen from Xavier Leroy's excellent course at Collège de France. It's available online!

Traduttore, traditore[1]

A nightmare scenario…

Our algorithm satisfies the specification

Better! Our implementation satisfies the specification

But the compiler has changed the meaning of my program

[1] The use of this quote in this context is stolen from Xavier Leroy's excellent course at Collège de France. It's available online!

Traduttore, traditore[1]

A nightmare scenario…

Our algorithm satisfies the specification

Better! Our implementation satisfies the specification

But the compiler has changed the meaning of my program

The executable code does not satisfy the specification

[1] The use of this quote in this context is stolen from Xavier Leroy's excellent course at Collège de France. It's available online!

Traduttore, traditore[1]

A nightmare scenario…

Our algorithm satisfies the specification

Better! Our implementation satisfies the specification

But the compiler has changed the meaning of my program

The executable code does not satisfy the specification

Natural languages are hard. But when it comes to programming languages, can we guarantee that our translators won't betray us?

[1] The use of this quote in this context is stolen from Xavier Leroy's excellent course at Collège de France. It's available online!

Add Some Tests?

A compiler is a program, and we want it to behave:
can't we just test them?

Note: **gcc** is composed of roughly 15 millions line of codes…

Add Some Tests?

A compiler is a program, and we want it to behave:
can't we just test them?

input

Note: **gcc** is composed of roughly 15 millions line of codes…

Add Some Tests?

A compiler is a program, and we want it to behave:
can't we just test them?

How to generate inputs?

input

llvm
gcc
ghc

Valid C program
fit to stress test your compiler

Note: **gcc** is composed of roughly 15 millions line of codes…

Add Some Tests?

A compiler is a program, and we want it to behave:
can't we just test them?

How to generate inputs?

llvm
gcc
ghc

output

Valid C program
fit to stress test your compiler

Note: **gcc** is composed of roughly 15 millions line of codes…

Add Some Tests?

A compiler is a program, and we want it to behave:
can't we just test them?

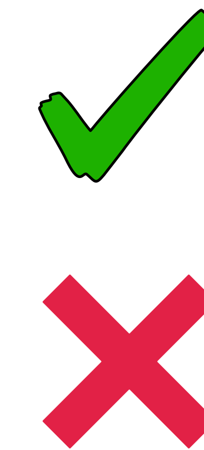
How to generate inputs?

Valid C program
fit to stress test your compiler

llvm
gcc
ghc

output

Some x86 assembly



Comment valider?

Note: **gcc** is composed of roughly 15 millions line of codes…

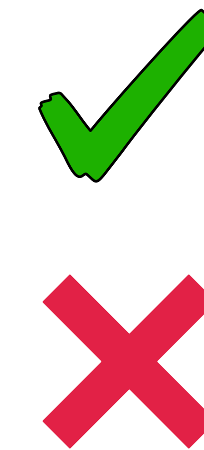
Add Some Tests?

A compiler is a program, and we want it to behave:
can't we just test them?

How to generate inputs?

llvm
gcc
ghc

output



Comment valider?

Valid C program
fit to stress test your compiler

Some x86 assembly

Simplest solution: *differential testing*. Have your compilers argue!

Note: **gcc** is composed of roughly 15 millions line of codes…

It's hard work, but it can be done!

Finding and Understanding Bugs in C Compilers

Xuejun Yang   Yang Chen   Eric Eide   John Regehr

1. Write a C program random generator (*CSmith*)
2. Have several (>= 3) C compilers run the programs and vote on the result

But C is no ML: a syntactically correct program is likely no C!

Undefined behaviours: null pointer dereference, array access out-of-bound, etc…

Your random generator must be paired with complex static analyses

It's hard work, but it's worth it!

Finding and Understanding Bugs in C Compilers

Xuejun Yang   Yang Chen   Eric Eide   John Regehr

"Every compiler we tested was found **to crash** and also **to silently generate wrong code** when presented with valid input."

It’s hard work, but it’s worth it!

Finding and Understanding Bugs in C Compilers

Xuejun Yang   Yang Chen   Eric Eide   John Regehr

“Every compiler we tested was found **to crash**
and also **to silently generate wrong code** when presented with valid input.”

“The striking thing about our CompCert results is that
the middle-end bugs we found in all other compilers **are absent**.”

Here enters the hero of our story: the verified compiler

Pre-history

CORRECTNESS OF A COMPILER FOR ARITHMETIC EXPRESSIONS*

JOHN McCARTHY and JAMES PAINTER

1967

1 Introduction

This paper contains a proof of the correctness of a simple compiling algorithm for compiling arithmetic expressions into machine language.

The definition of correctness, the formalism used to express the description of source language, object language and compiler, and the methods of proof are all intended to serve as prototypes for the more complicated task of proving the correctness of usable compilers. The ultimate goal, as outlined in references [1], [2], [3] and [4] is to make it possible to use a computer to check proofs that compilers are correct.

Pre-history

CORRECTNESS OF A COMPILER FOR ARITHMETIC EXPRESSIONS*

JOHN McCARTHY and JAMES PAINTER

1967

1 Introduction

This paper contains a proof of the correctness of a simple compiling algorithm for compiling arithmetic expressions into machine language.

The definition of correctness, the formalism used to express the description of source language, object language and compiler, and the methods of proof are all intended to serve as prototypes for the more complicated task of proving the correctness of usable compilers. The ultimate goal, as outlined in references [1], [2], [3] and [4] is to make it possible to use a computer to check proofs that compilers are correct.

Pre-history

CORRECTNESS OF A COMPILER FOR ARITHMETIC EXPRESSIONS*

JOHN McCARTHY and JAMES PAINTER

1967

1 Introduction

This paper contains a proof of the correctness of a simple compiling algorithm for compiling arithmetic expressions into machine language.

The definition of correctness, the formalism used to express the description of source language, object language and compiler, and the methods of proof are all intended to serve as prototypes for the more complicated task of proving the correctness of usable compilers. The ultimate goal, as outlined in references [1], [2], [3] and [4] is to make it possible to use a computer to check proofs that compilers are correct.

Pre-history

1972

Proving Compiler Correctness in a Mechanized Logic

R. Milner and R. Weyhrauch

Computer Science Department
Stanford University

Abstract

We discuss the task of machine-checking the proof of a simple compiling algorithm. The proof-checking program is LCF, an implementation of a logic for computable functions due to Dana Scott, in which the abstract syntax and extensional semantics of programming languages can be naturally expressed. The source language in our example is a simple ALGOL-like language with assignments, conditionals, whiles and compound statements. The target language is an assembly language for a machine with a pushdown store. Algebraic methods are used to give structure to the proof, which is presented only in outline. However, we present in full the expression-compiling part of the algorithm. More than half of the complete proof has been machine checked, and we anticipate no difficulty with the remainder. We discuss our experience in conducting the proof, which indicates that a large part of it may be automated to reduce the human contribution.

Pre-history

1972

Proving Compiler Correctness in a Mechanized Logic

R. Milner and R. Weyhrauch

Computer Science Department
Stanford University

Abstract

We discuss the task of machine-checking the proof of a simple compiling algorithm. The proof-checking program is LCF, an implementation of a logic for computable functions due to Dana Scott, in which the abstract syntax and extensional semantics of programming languages can be naturally expressed. The source language in our example is a simple ALGOL-like language with assignments, conditionals, whiles and compound statements. The target language is an assembly language for a machine with a pushdown store. Algebraic methods are used to give structure to the proof, which is presented only in outline. However, we present in full the expression-compiling part of the algorithm. More than half of the complete proof has been machine checked, and we anticipate no difficulty with the remainder. We discuss our experience in conducting the proof, which indicates that a large part of it may be automated to reduce the human contribution.

Pre-history

1972

Proving Compiler Correctness in a Mechanized Logic

R. Milner and R. Weyhrauch

Computer Science Department
Stanford University

Abstract

We discuss the task of machine-checking the proof of a simple compiling algorithm. The proof-checking program is LCF, an implementation of a logic for computable functions due to Dana Scott, in which the abstract syntax and extensional semantics of programming languages can be naturally expressed. The source language in our example is a simple ALGOL-like language with assignments, conditionals, whiles and compound statements. The target language is an assembly language for a machine with a pushdown store. Algebraic methods are used to give structure to the proof, which is presented only in outline. However, we present in full the expression-compiling part of the algorithm. More than half of the complete proof has been machine checked, and we anticipate no difficulty with the remainder. We discuss our experience in conducting the proof, which indicates that a large part of it may be automated to reduce the human contribution.

Pre-history

Compiler Correctness
chanized Logic

1972

ınd R. Weyhrauch
nce Department
ersity

Turned 90 Yesterday!

1976's Turing Prize (with Rabin)

ask of machine-checking the proof of a simple compiling
oof-checking program is LCF, an implementation of a logic
unctions due to Dana Scott, in which the abstract syntax
semantics of programming languages can be naturally
ource language in our example is a simple ALGOL-like
ignments, conditionals, whiles and compound statements.
ge is an assembly language for a machine with a pushdown
nethods are used to give structure to the proof, which is
utline. However, we present in full the expression-compiling
m. More than half of the complete proof has been machine
nticipate no difficulty with the remainder. We discuss our
ducting the proof, which indicates that a large part of it
l to reduce the human contribution.

CompCert (2009-) : a Verified C99 Optimising Compiler

If CompCert successfully compiles a C source program p
down to a PowerPC assembly program asm,
then « asm and p behave the same »

CompCert in production : safer code?

CompCert is commercialized by AbsInt and known to be used internally by:

- Airbus (avionic)
- MTU Friedrichshafen (civil nuclear energy)
- TUM (avionic)

CompCert in production : safer code?

CompCert is commercialized by AbsInt and known to be used internally by:

- Airbus (avionic)
- MTU Friedrichshafen (civil nuclear energy)
- TUM (avionic)

Why does CompCert interest so much these industries?

CompCert in production : safer code?

CompCert is commercialized by AbsInt and known to be used internally by:

- Airbus (avionic)
- MTU Friedrichshafen (civil nuclear energy)
- TUM (avionic)

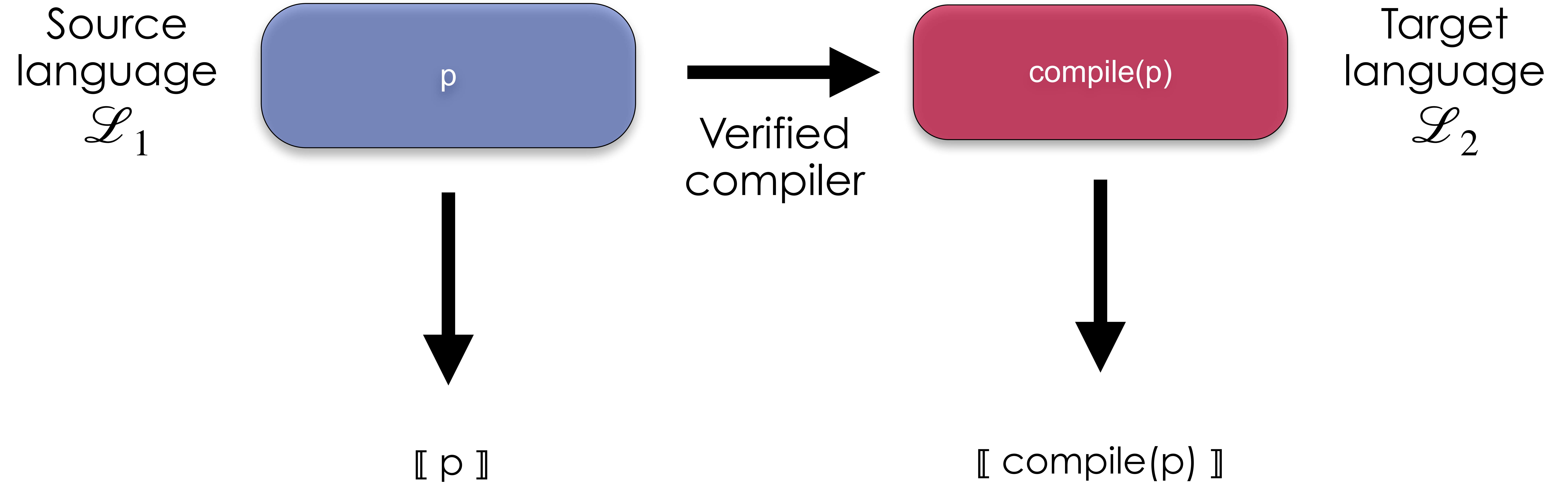
Why does CompCert interest so much these industries?

Paradoxically, not so much to increase trust than to improve performances!

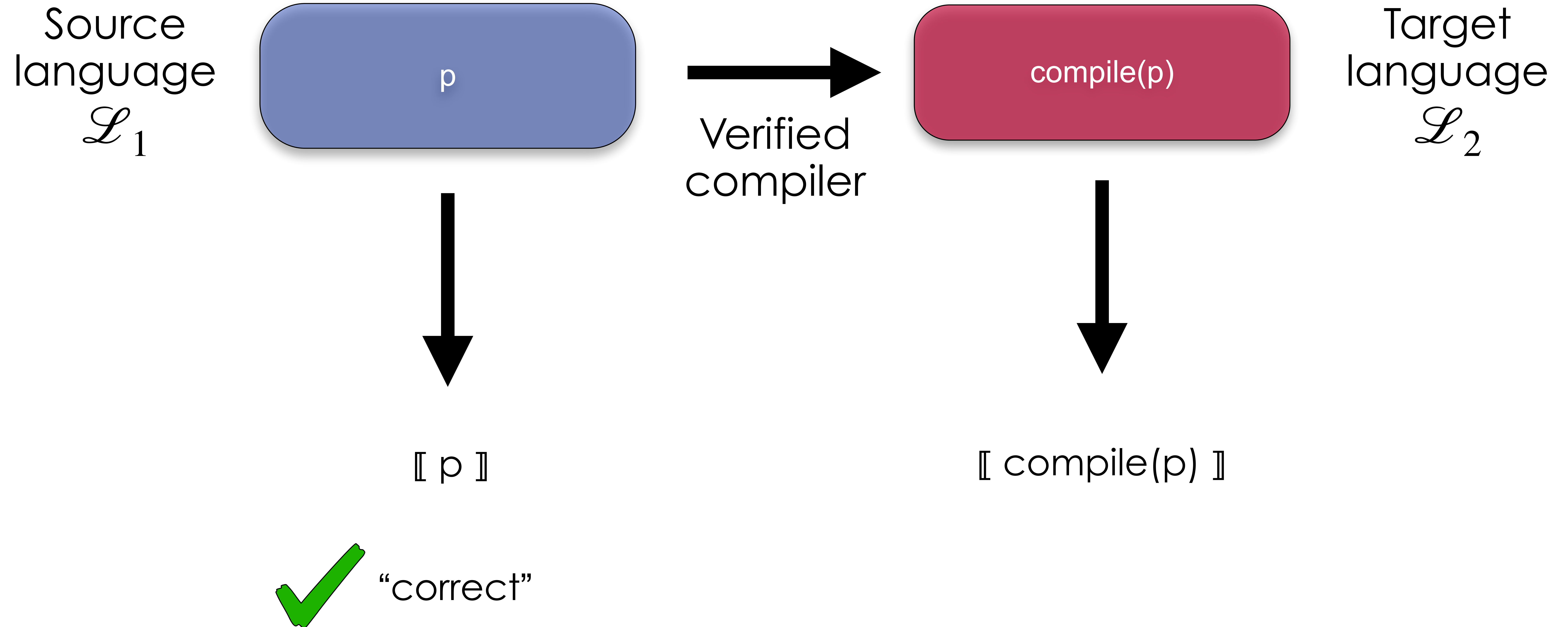
"With CompCert it is possible to decrease the execution time of our flight control algorithms by a significant amount" (TUM)

The standards for certification are extremely stricts for such fields: optimisations were usually completely ruled out!

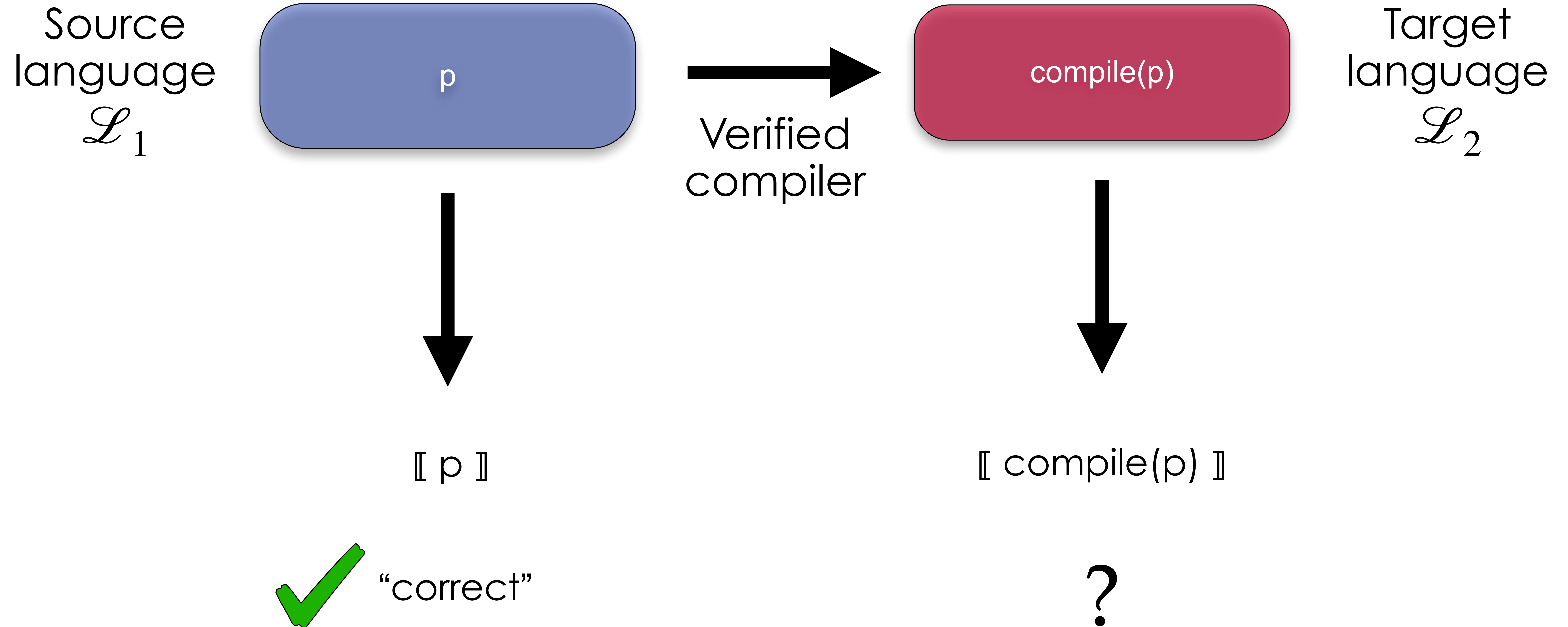
Verified Compilation



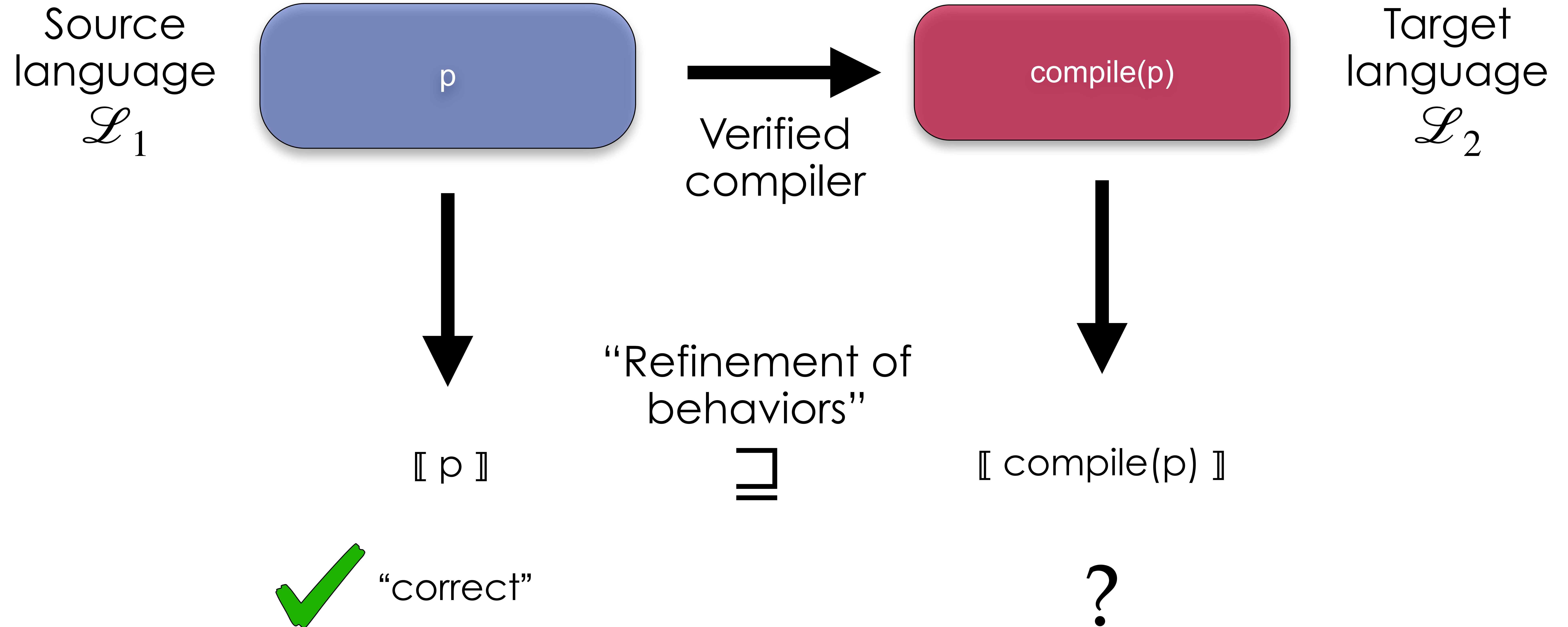
Verified Compilation



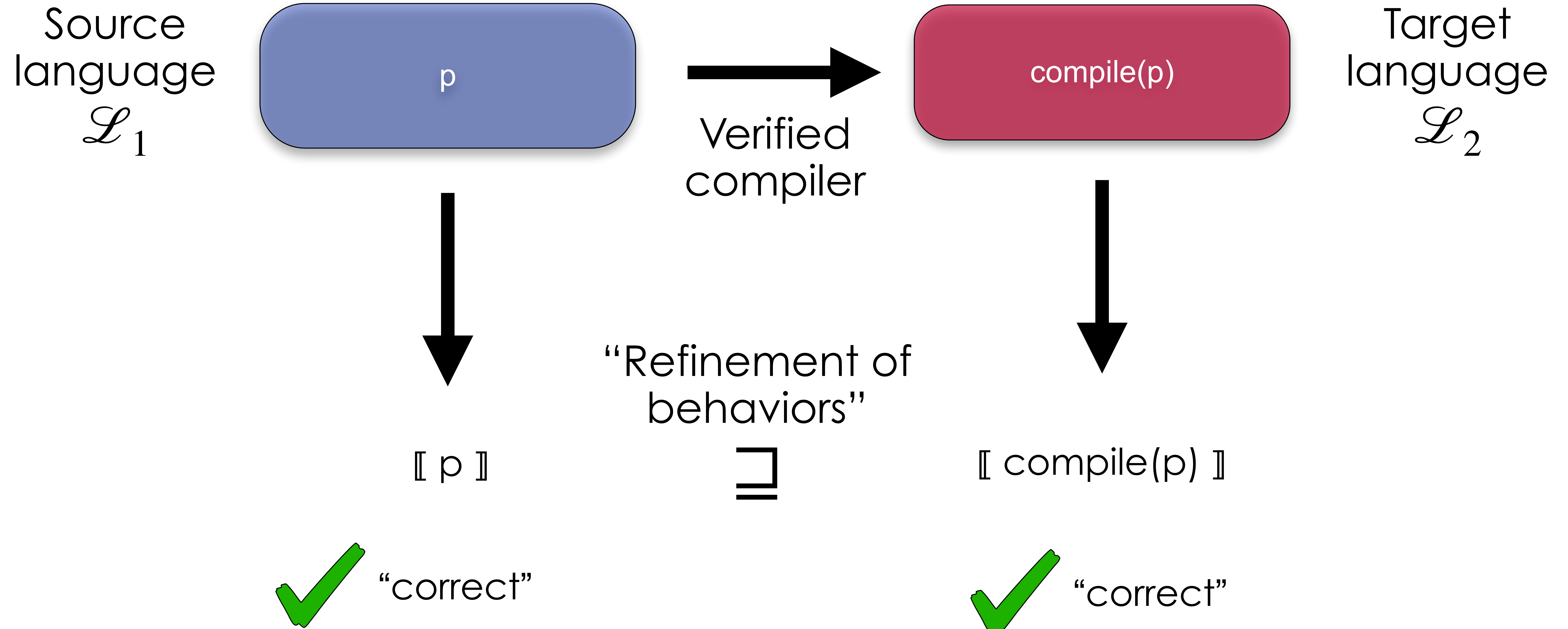
Verified Compilation



Verified Compilation



Verified Compilation



Verified Compilation

Source language $\mathscr{L}_1$

p

Verified compiler

compile(p)

Target language $\mathscr{L}_2$

⟦ p ⟧

"Refinement of behaviors"

$\sqsupseteq$

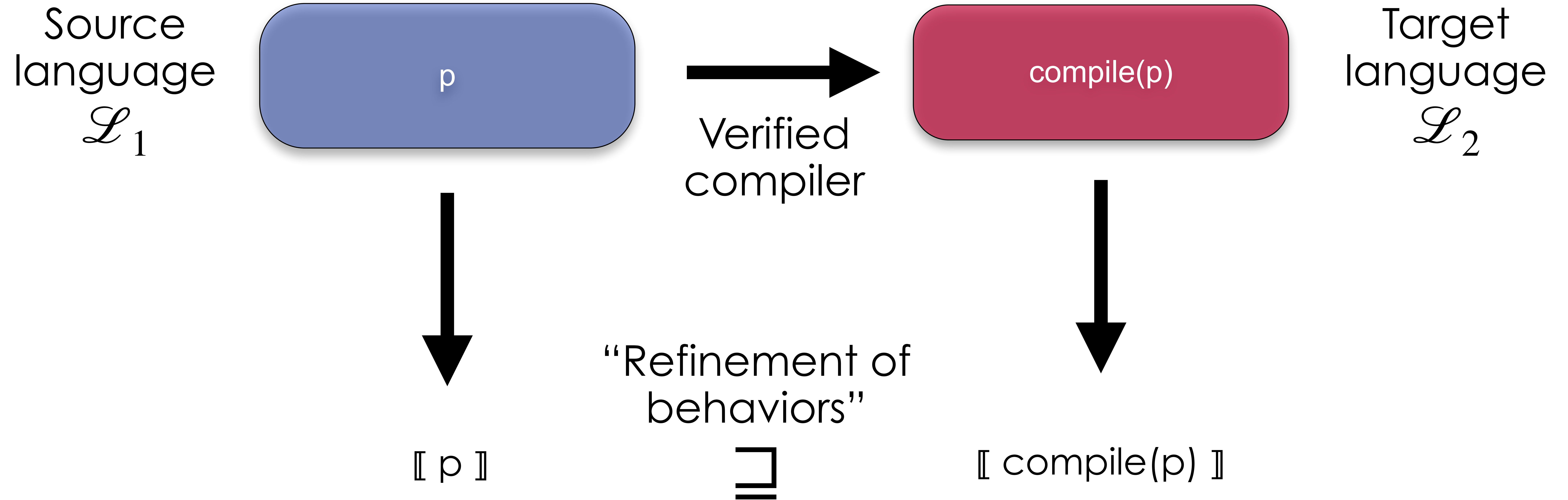
⟦ compile(p) ⟧

"correct"

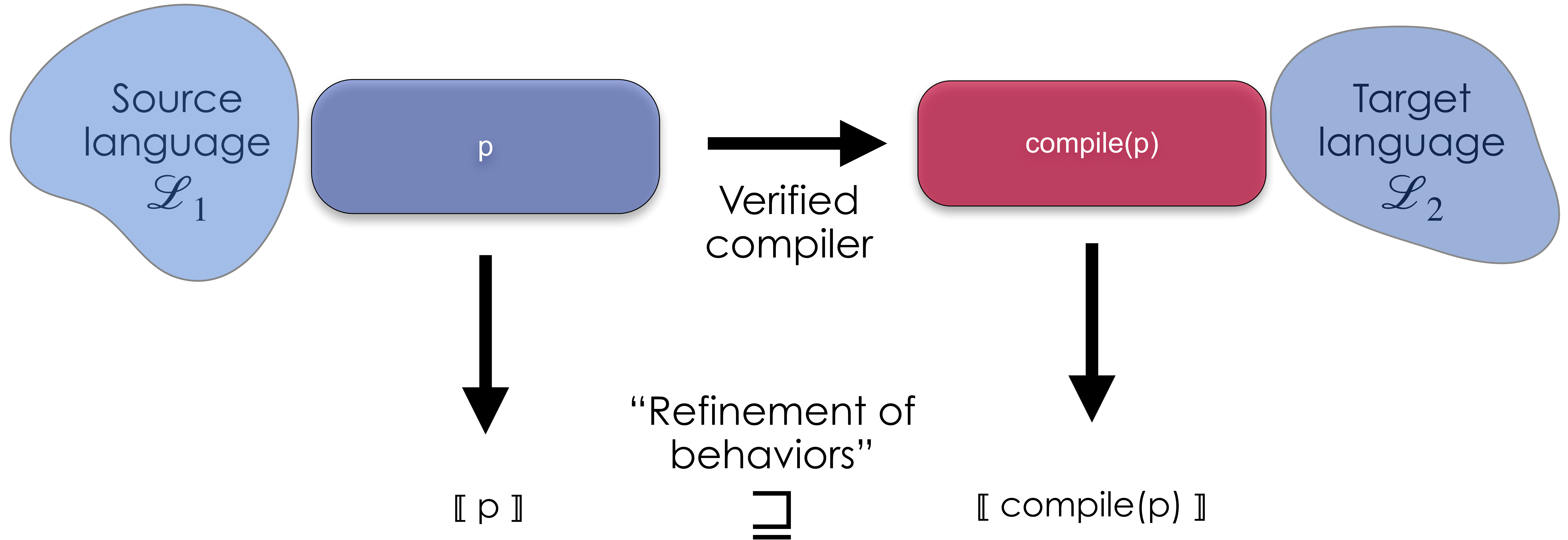
"correct"

The compiler is written and formally proved correct in a Proof Assistant

Verified Compilation

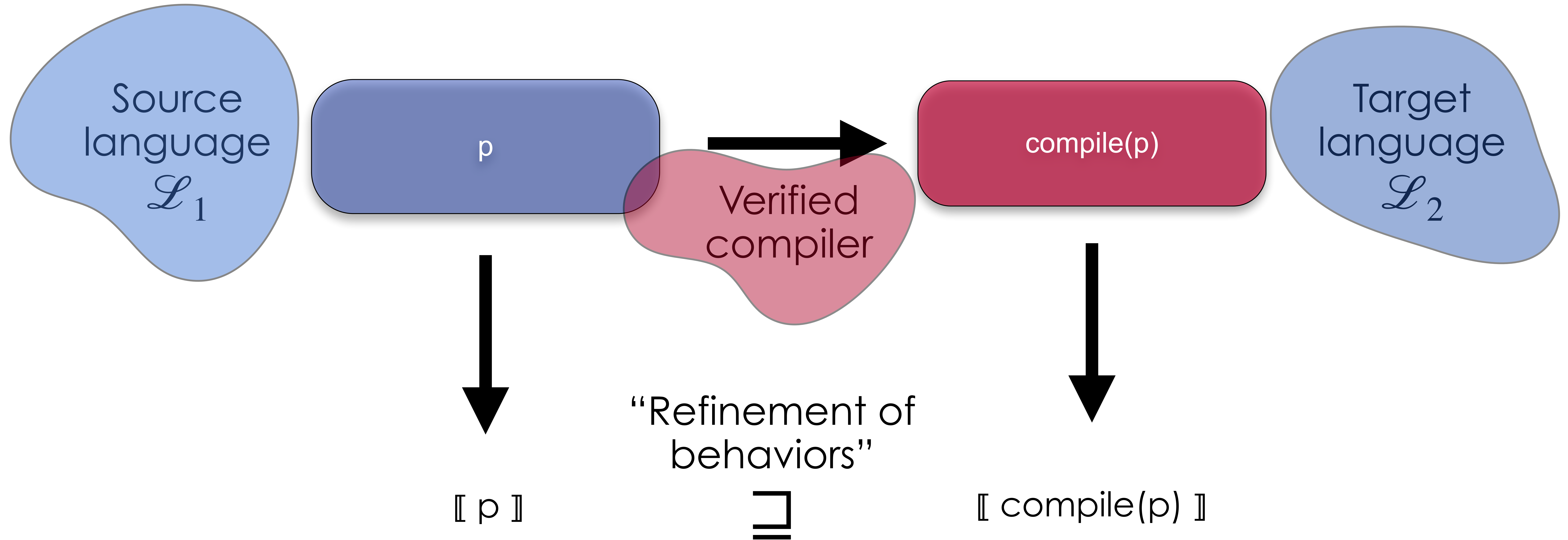


Verified Compilation



New languages, new constructions

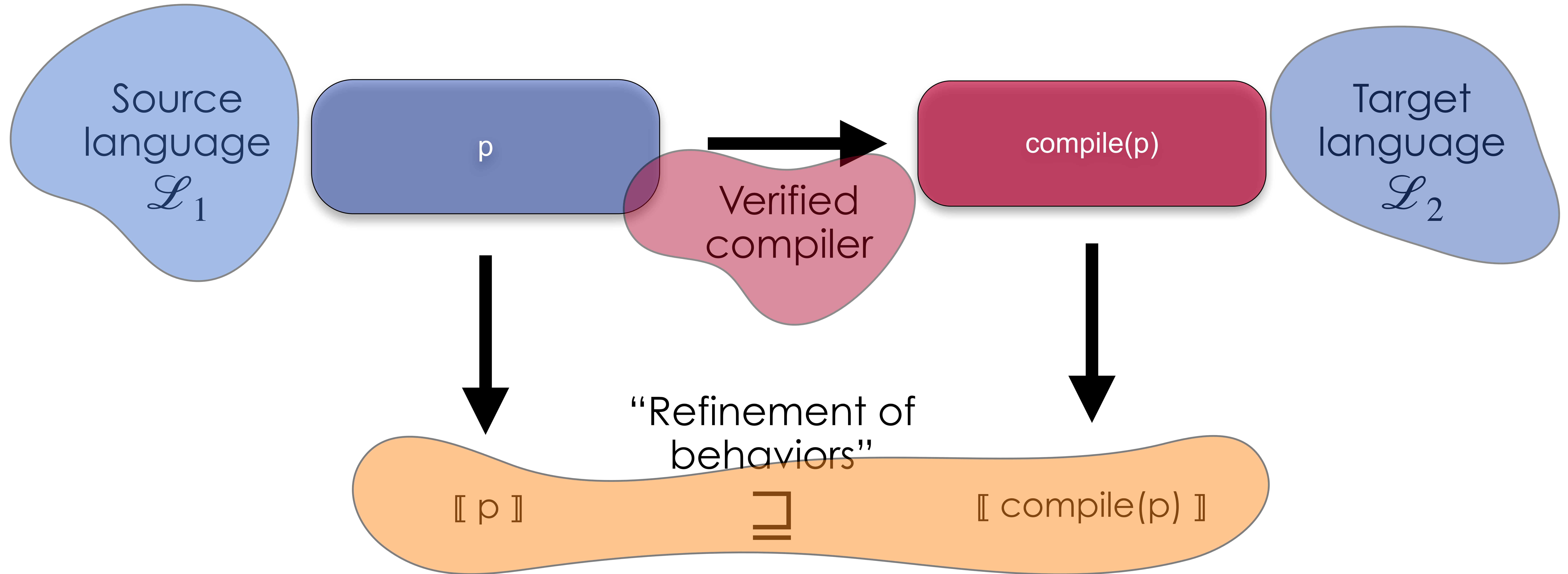
Verified Compilation



New languages, new constructions

Supporting better/more optimizations

Verified Compilation

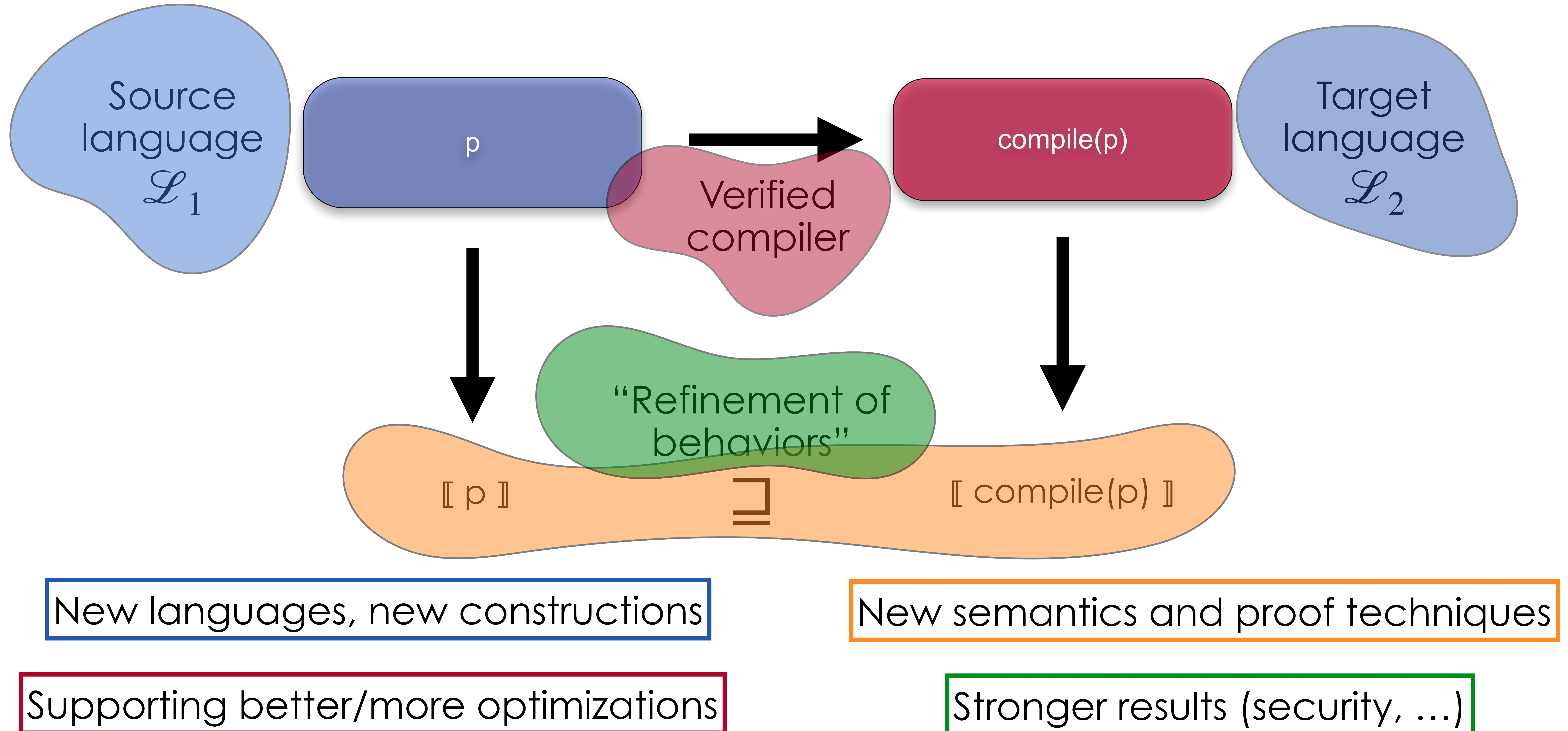


New languages, new constructions

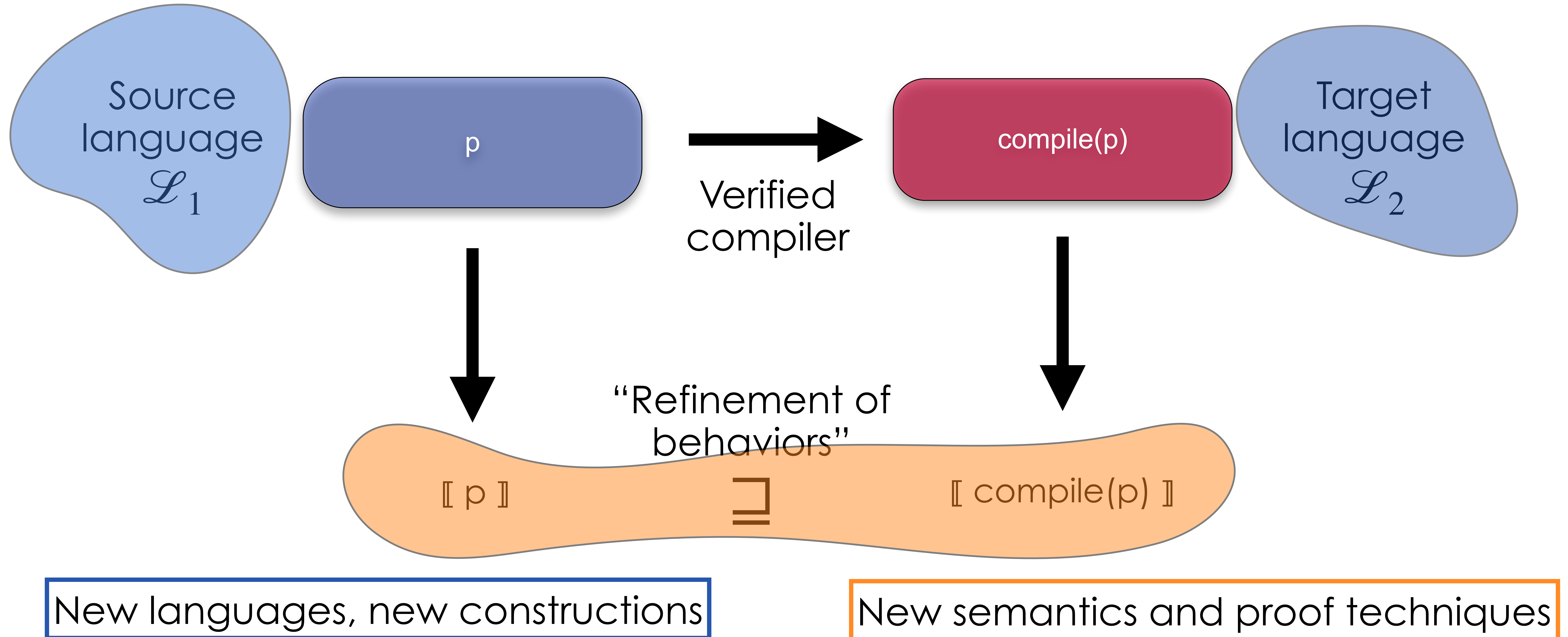
New semantics and proof techniques

Supporting better/more optimizations

Verified Compilation



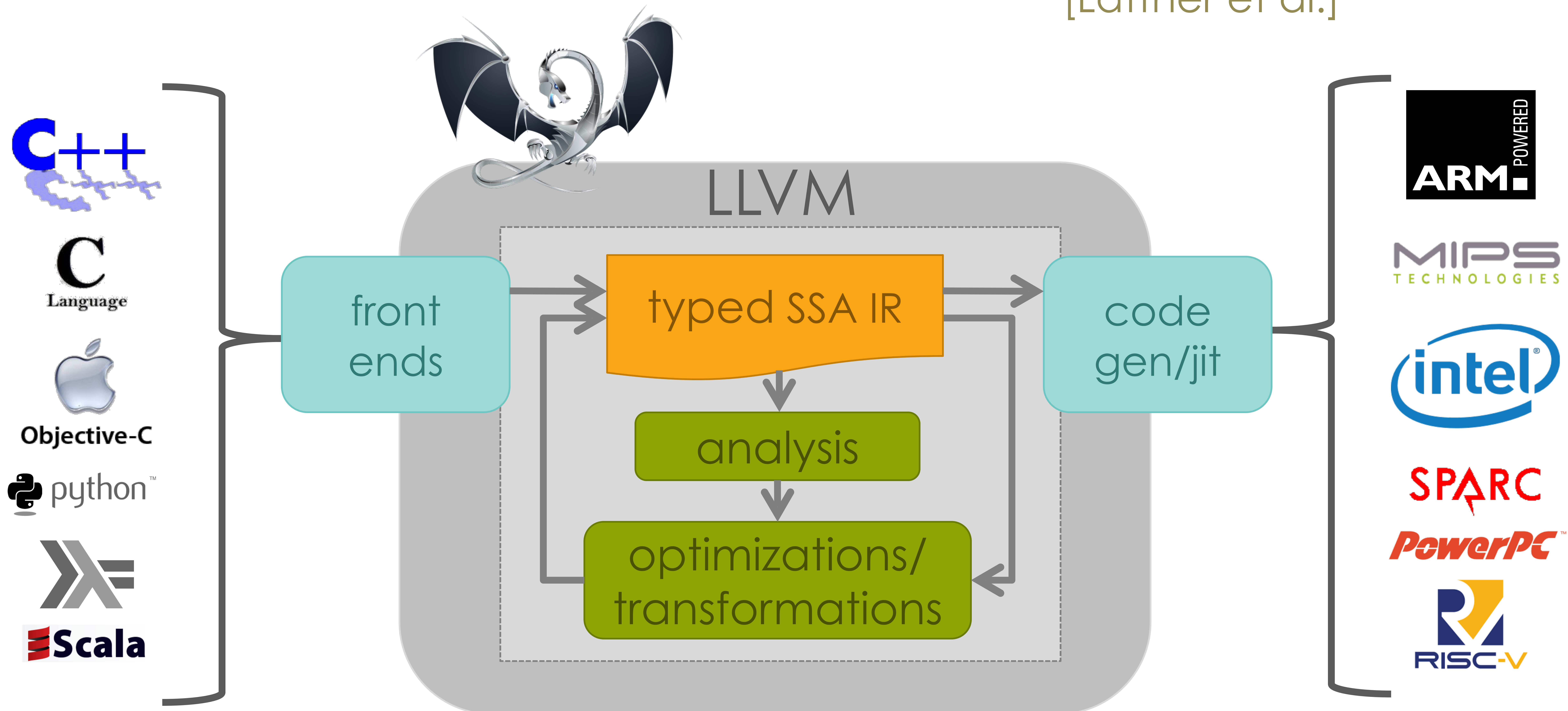
Verified Compilation



A modular, compositional, and executable semantics for LLVM IR

LLVM Compiler Infrastructure

[Lattner et al.]



The LLVM IR

LLVM Reference Manual
table of contents

The LLVM IR

LLVM Reference Manual table of contents

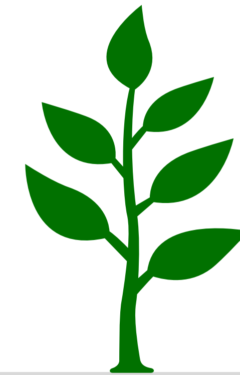
LLVM Home | Documentation»

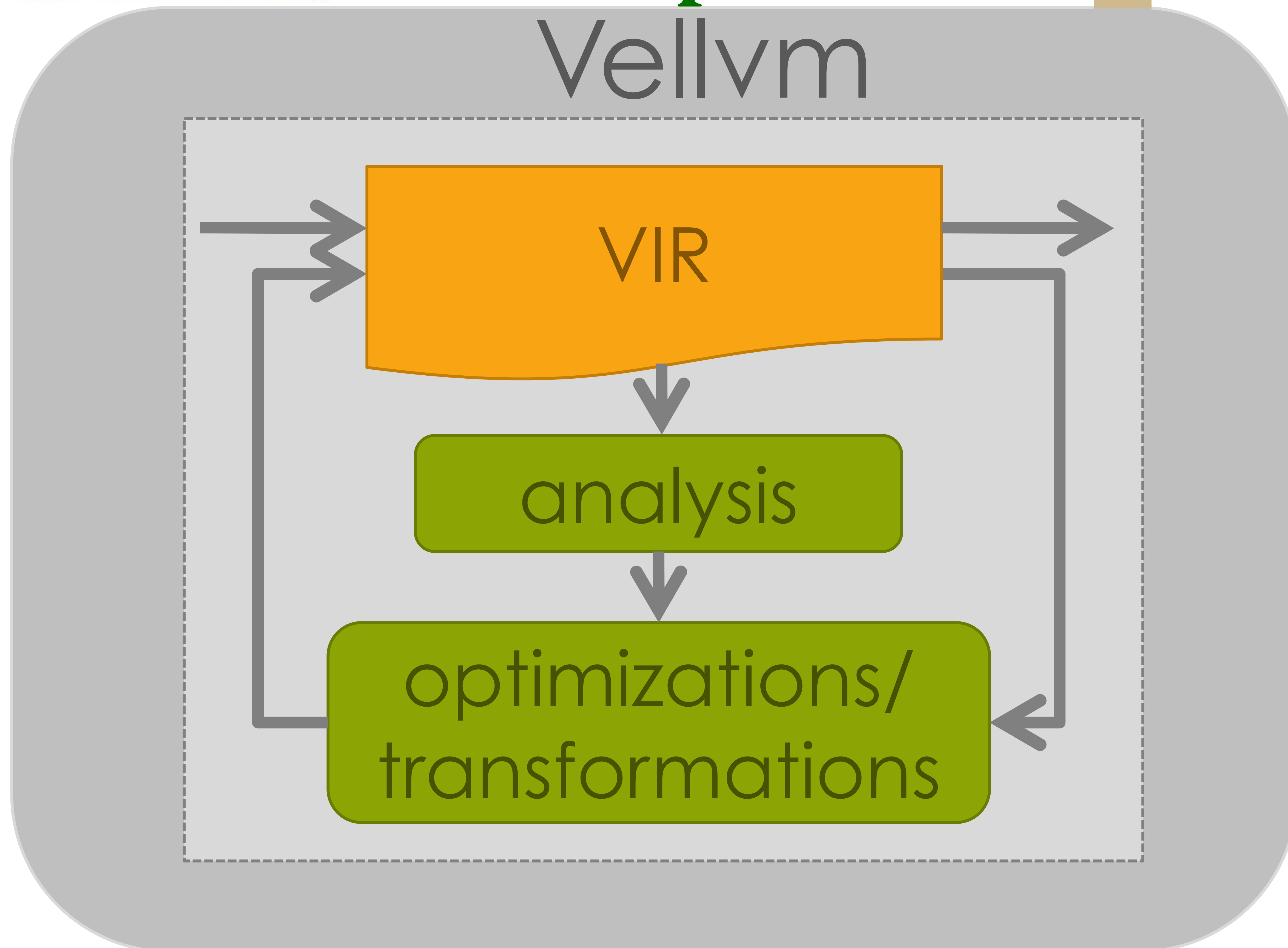
LLVM Language Reference Manual

Abstract

This document is a reference manual for the LLVM assembly language. LLVM is a Static Single Assignment (SSA) based representation that provides type safety, low-level operations, flexibility, and the capability of representing 'all'

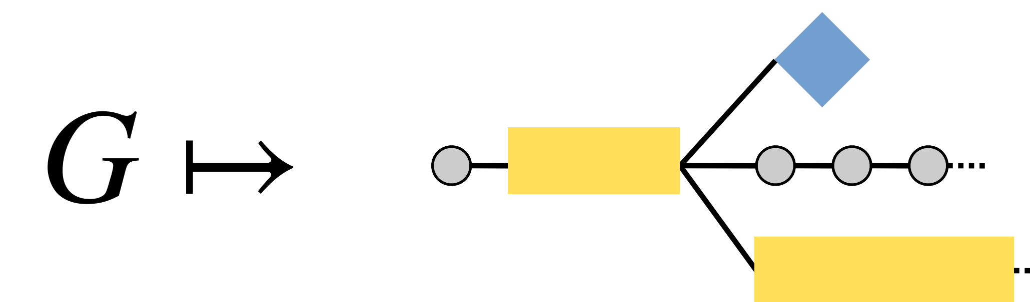
The Vellvm Project



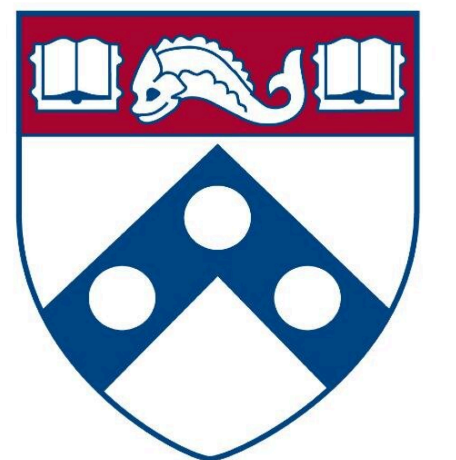


Operational style

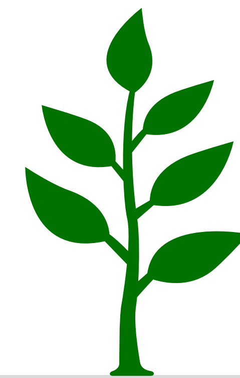
Monadic, denotational

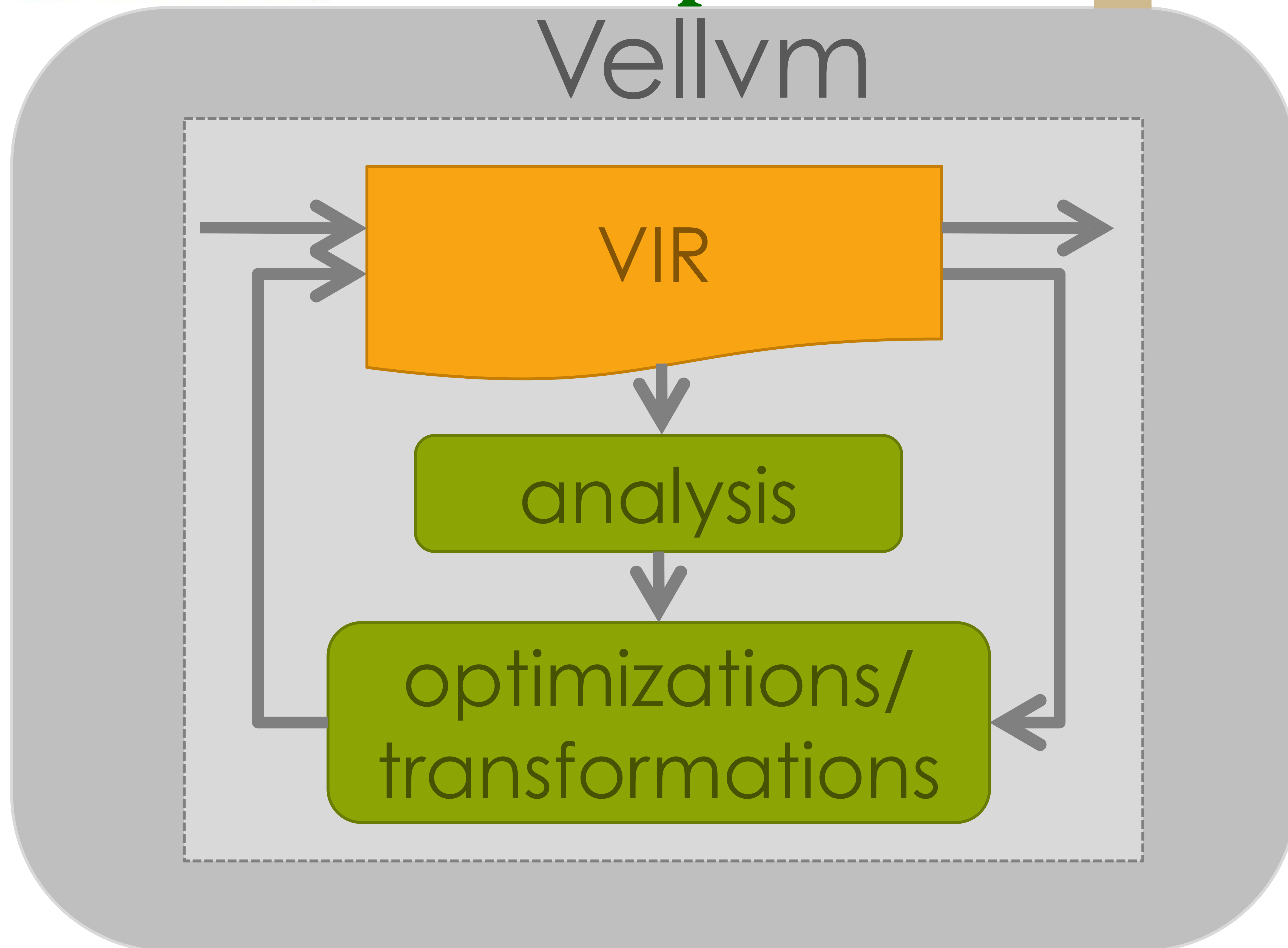


Project in collaboration with the University of Pennsylvania



The Vellvm Project

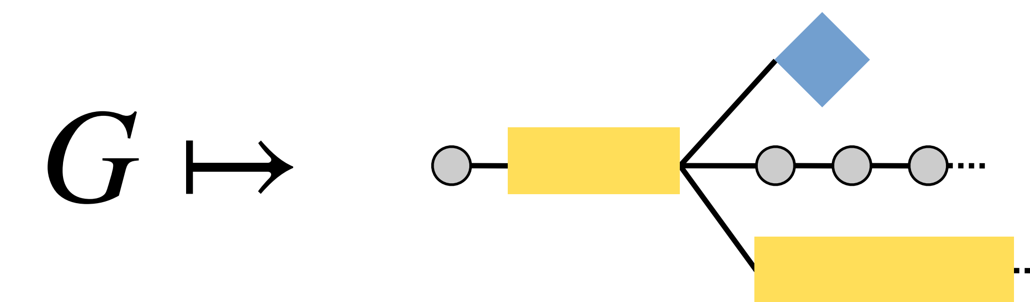




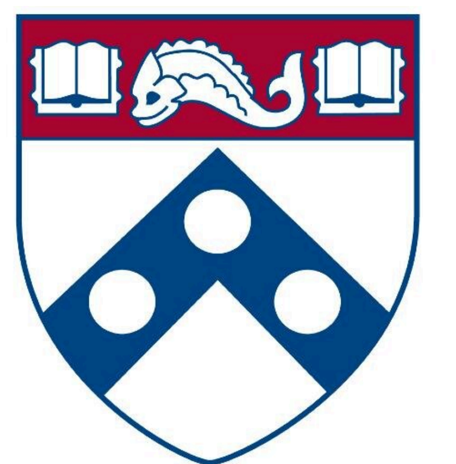
Operational style

$$G \vdash pc,\ mem \rightarrow pc', mem'$$

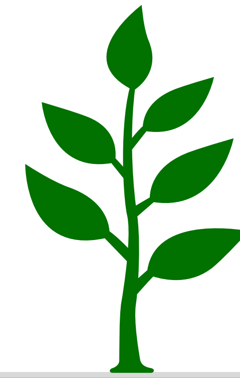
Monadic, denotational

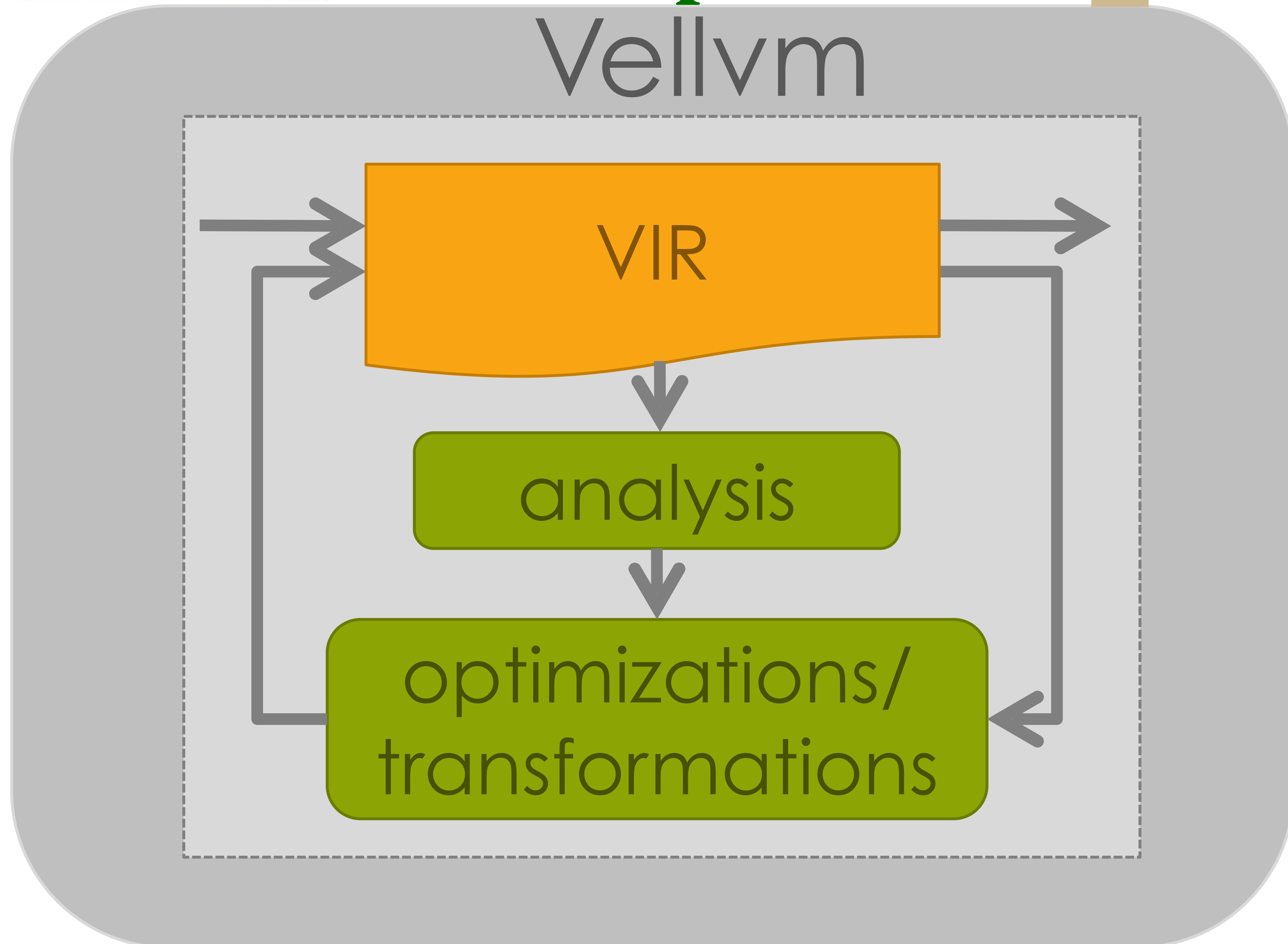
$G \mapsto$ 

Project in collaboration with the University of Pennsylvania



The Vellvm Project

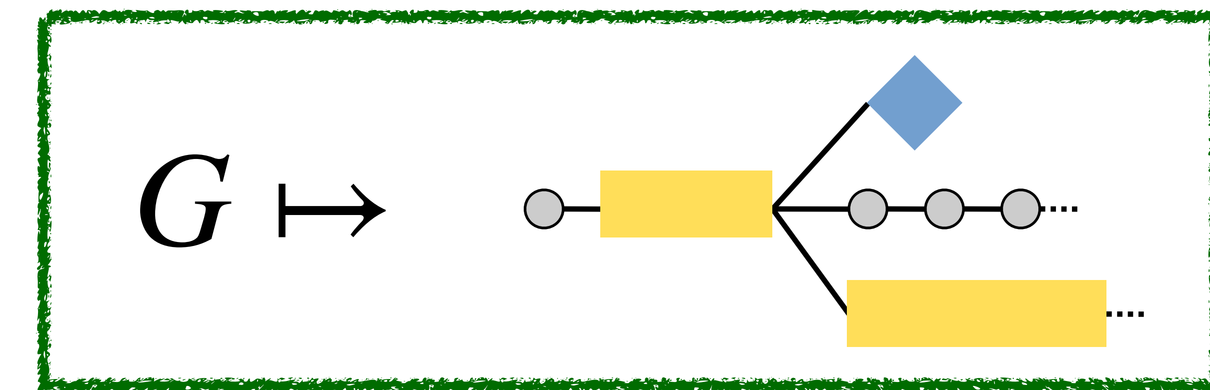




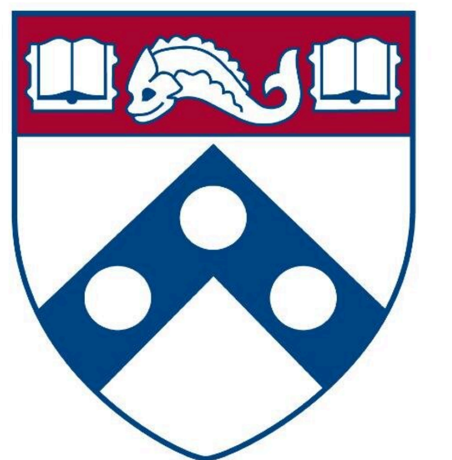
Operational style

$$G \vdash pc,\ mem \rightarrow pc', mem'$$

Monadic, denotational

$G \mapsto$ 

Project in collaboration with the University of Pennsylvania



(Operational) Semantics of an Imperative Language

Moving to the black board