Formal verification of a realistic compiler

Xavier Leroy

CACM 2009

CS 7194: Great Works in Programming Languages

Presenter : **Irene Yoon** | Mentor : **Ryan Doenges**

Building robust compilers is Hard.

Bugs

Bugs

[Yang et al 2011]

- Random testing finds bugs in 11 C compilers

Bugs

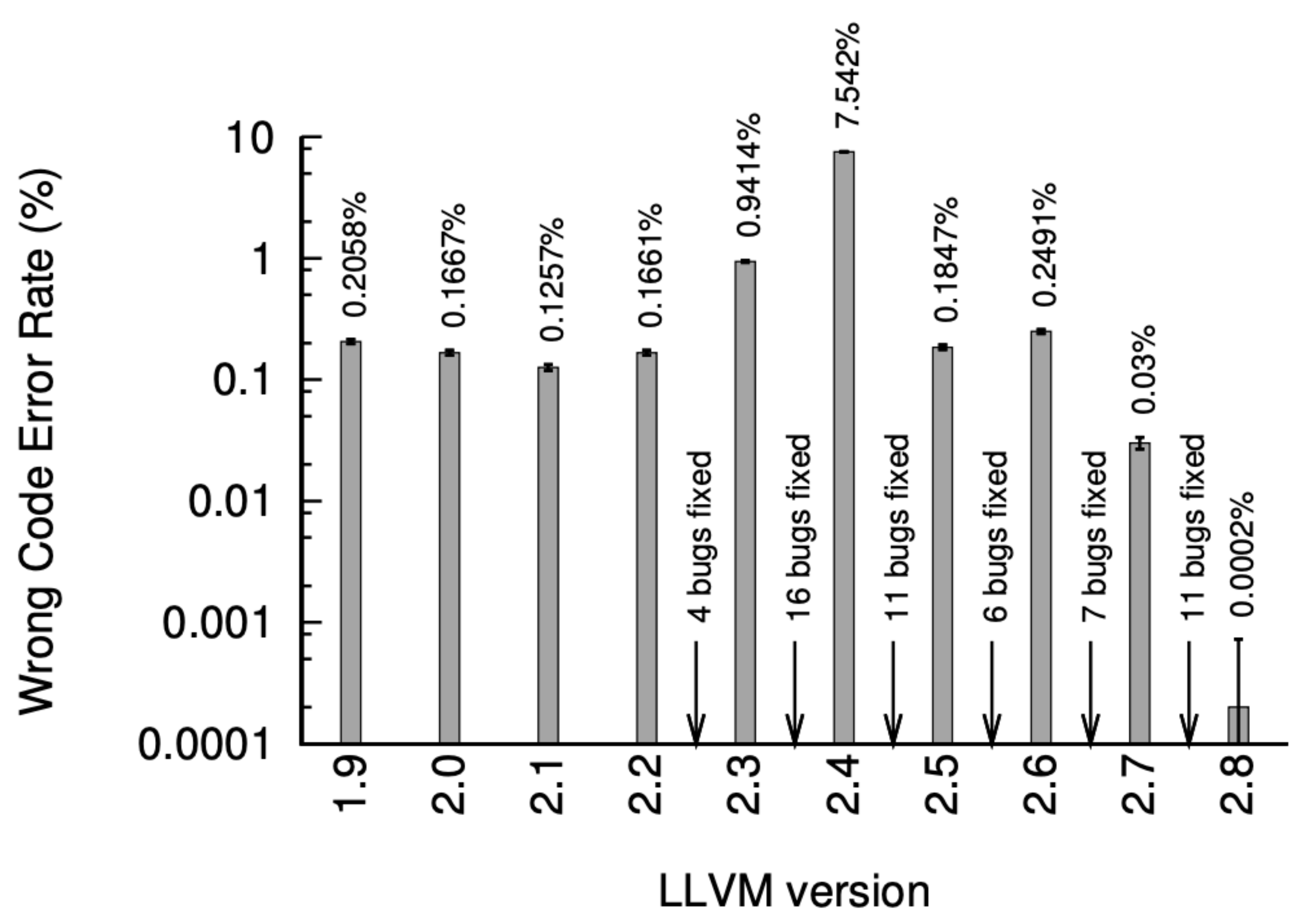
[Yang et al 2011]

- Random testing finds bugs in 11 C compilers
- **Hundreds** of previously unknown bugs

Bugs

[Yang et al 2011]

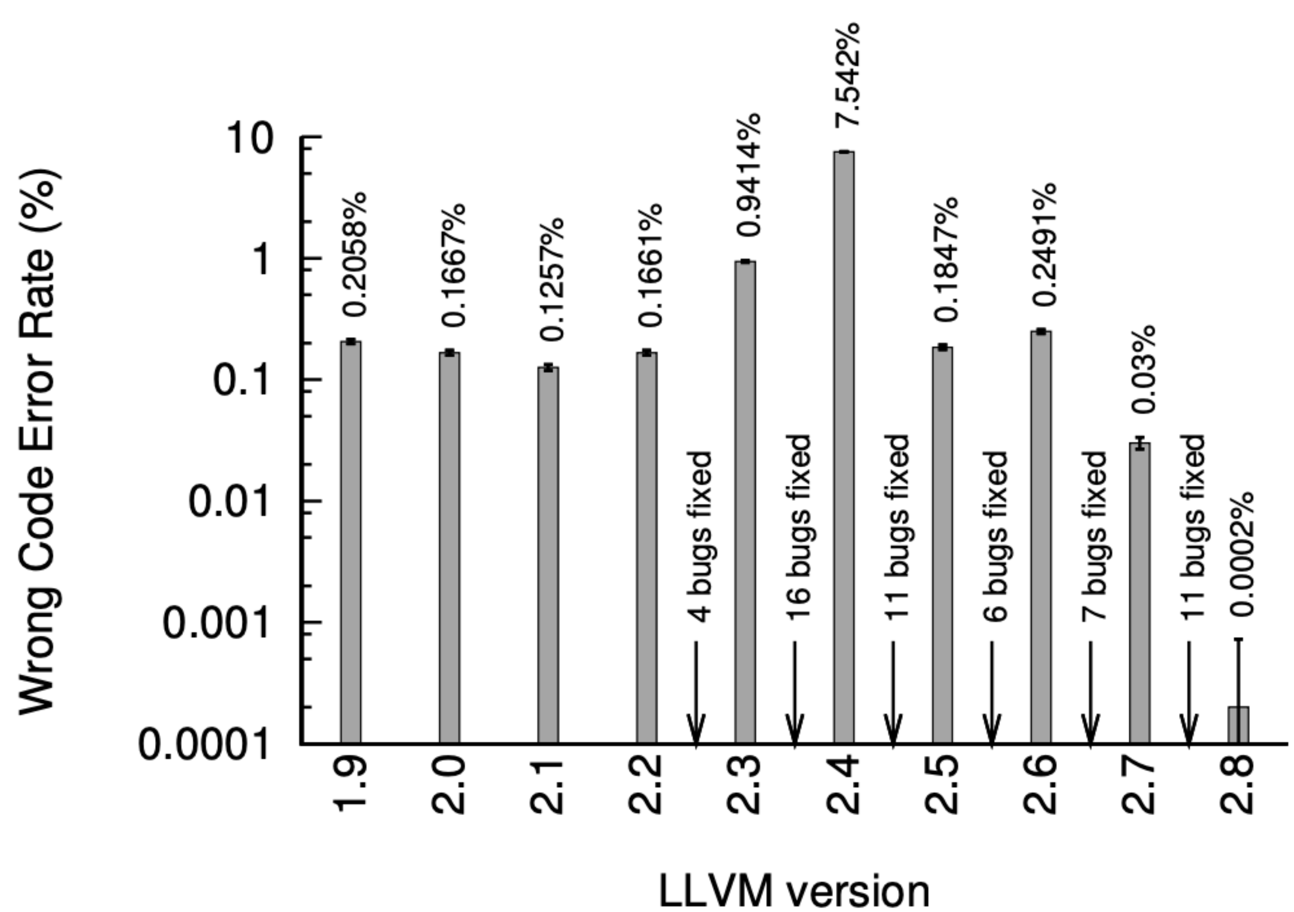
- Random testing finds bugs in 11 C compilers
- **Hundreds** of previously unknown bugs



Bugs

[Yang et al 2011]

- Random testing finds bugs in 11 C compilers
- **Hundreds** of previously unknown bugs
- LLVM has a large test suite



✅ Building compilers is hard

✅ Building compilers is hard

✅ Testing sucks

✅ Building compilers is hard

✅ Testing sucks

✅ Formalisms are good

✅ Building compilers is hard

✅ Testing sucks

✅ Formalisms are good

Formal verification of a compiler

First Published Proof of Compiler Correctness

[1967]

CORRECTNESS OF A COMPILER FOR ARITHMETIC EXPRESSIONS*

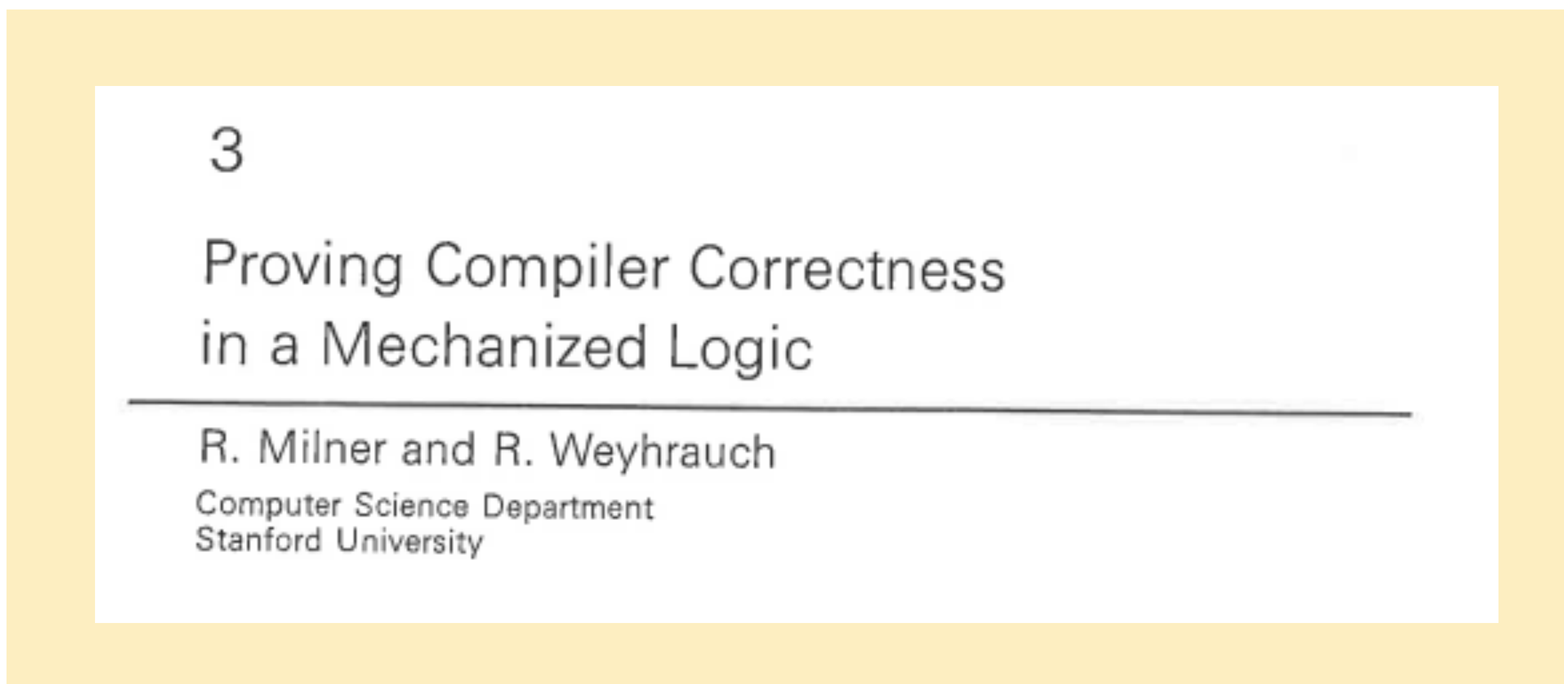
JOHN McCARTHY and JAMES PAINTER

1967

- arithmetic expressions → stack machine code
- prototype for proving usable compilers

First Mechanized Proof of Compiler Correctness

[1972]


3

Proving Compiler Correctness
in a Mechanized Logic

R. Milner and R. Weyhrauch
Computer Science Department
Stanford University

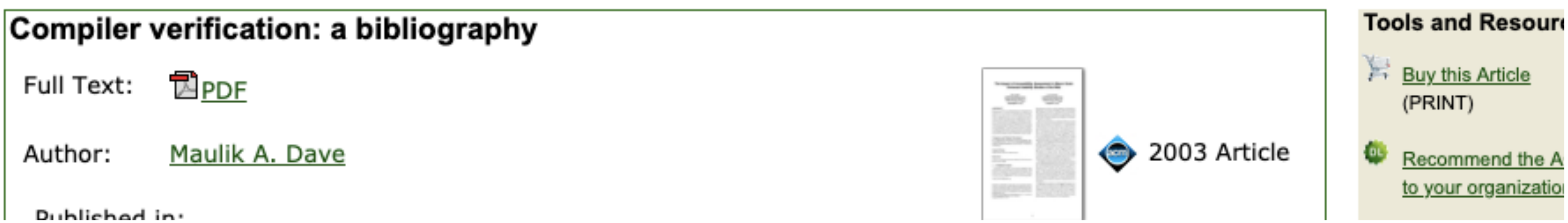
- ALGOL-like language → elementary assembly language
- Stanford LCF

Compiler Verification

- 100+ papers on compiler verification since 1967

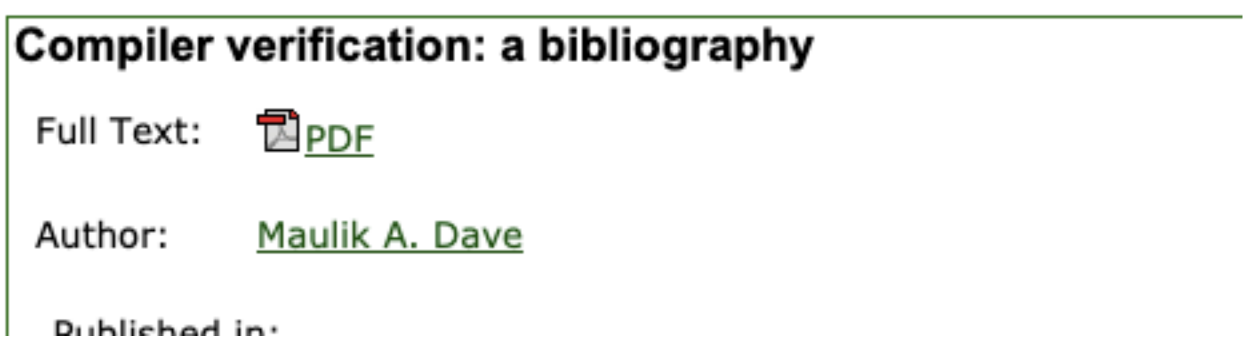
Compiler Verification

- 100+ papers on compiler verification since 1967



Compiler Verification

- 100+ papers on compiler verification since 1967



Compiler Verification

- 1

The Verifying Compiler: A Grand Challenge for Computing Research

[2003]

TONY HOARE

Microsoft Research Ltd., Cambridge, UK

CompCert

CompCert

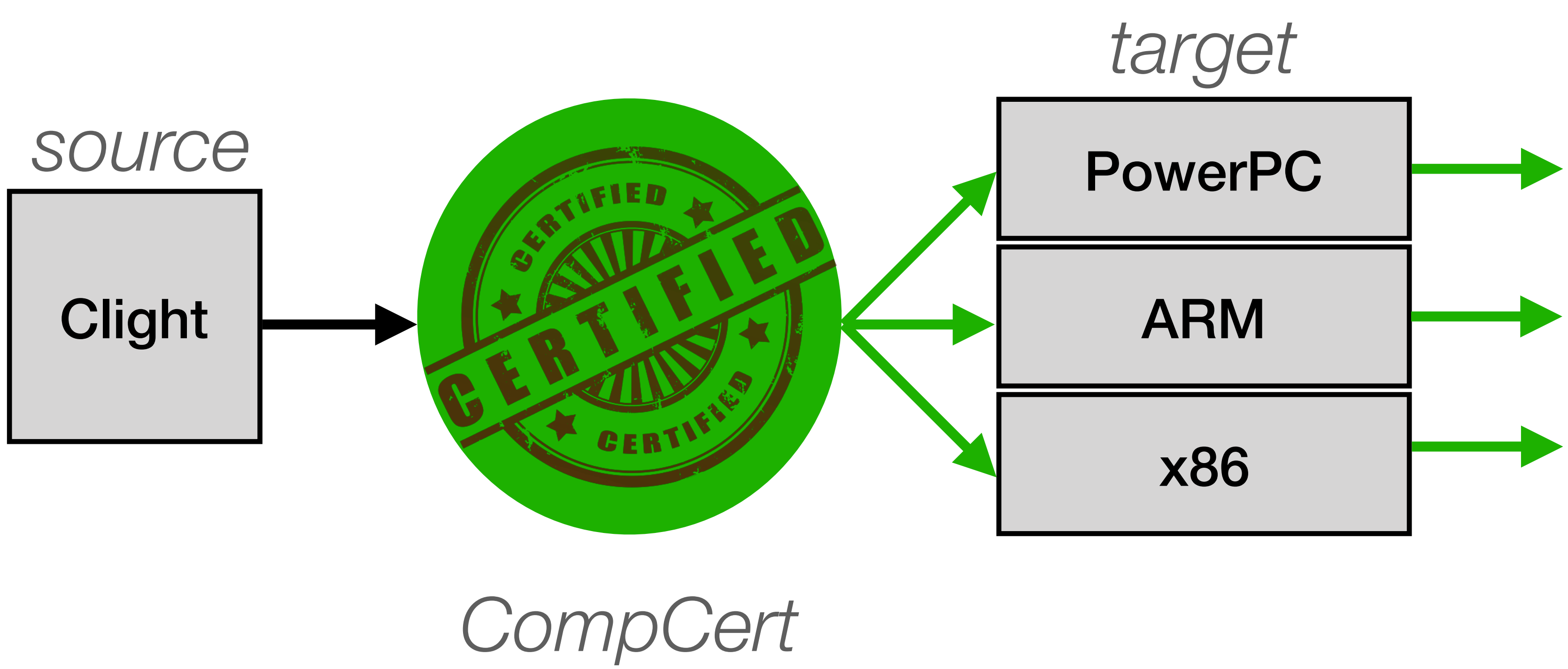
[2009]

“Develop and prove correct a ***realistic*** compiler, usable for critical embedded software.”

CompCert

[2009]

“Develop and prove correct a ***realistic*** compiler, usable for critical embedded software.”

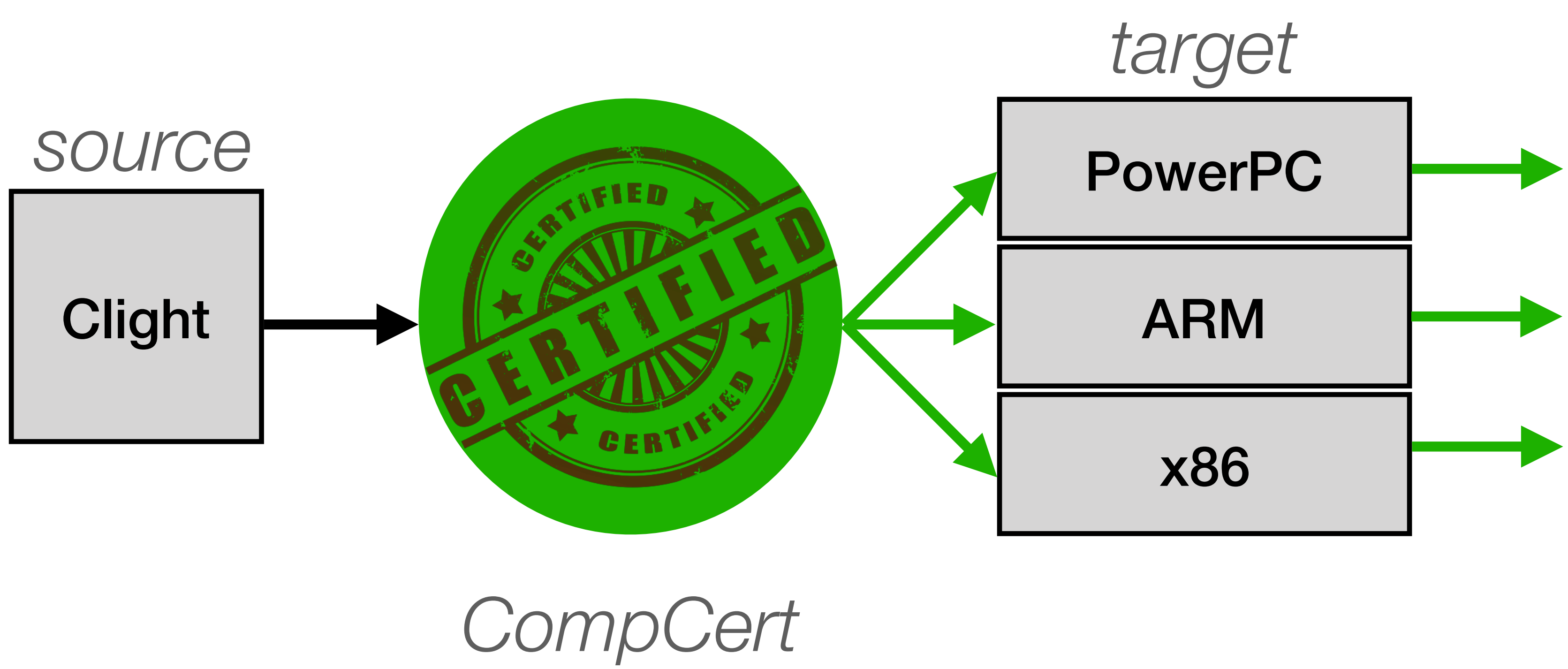


CompCert

[2009]

"Develop and prove correct a ***realistic*** compiler, usable for critical embedded software."

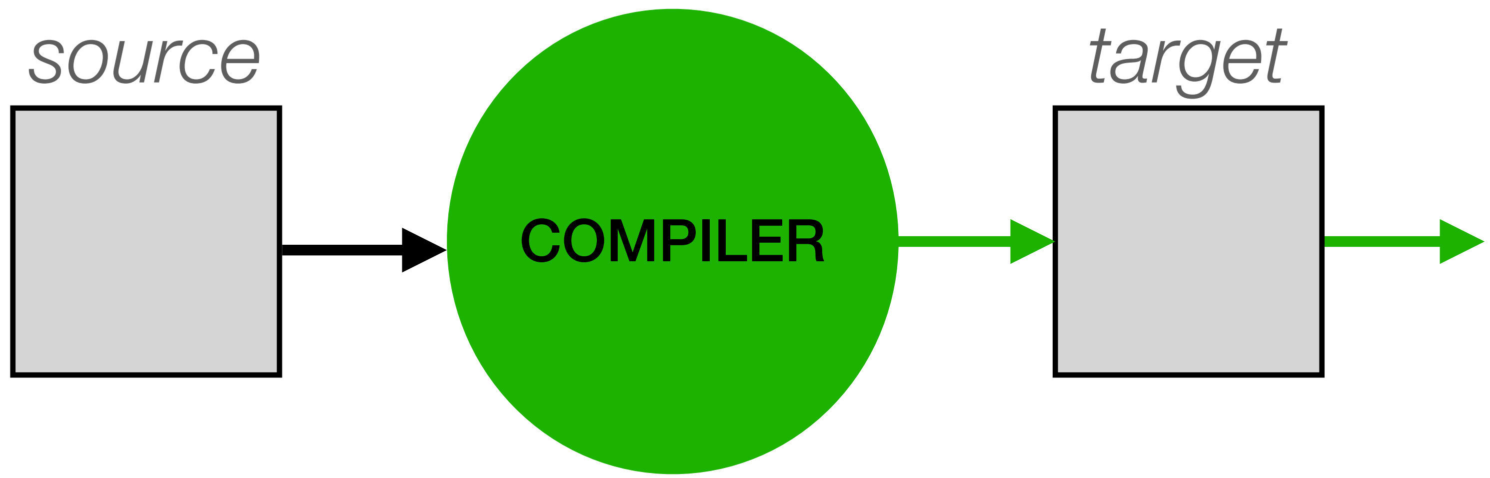
- 42k Coq, 3 person years



Verified, Validated, Certifying

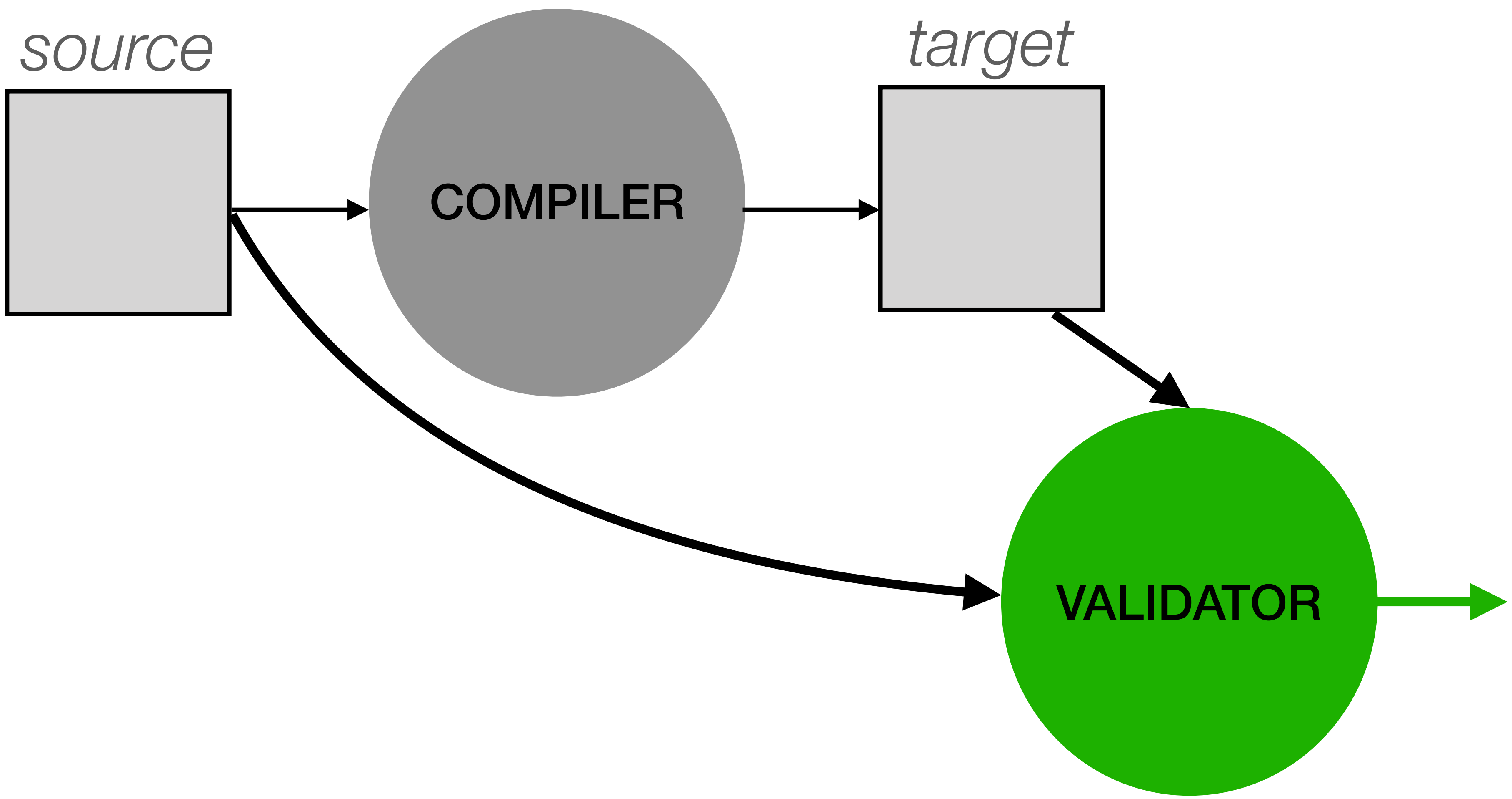
Verified, Validated, Certifying

1. Verified transformation [***Compiler Correctness***]



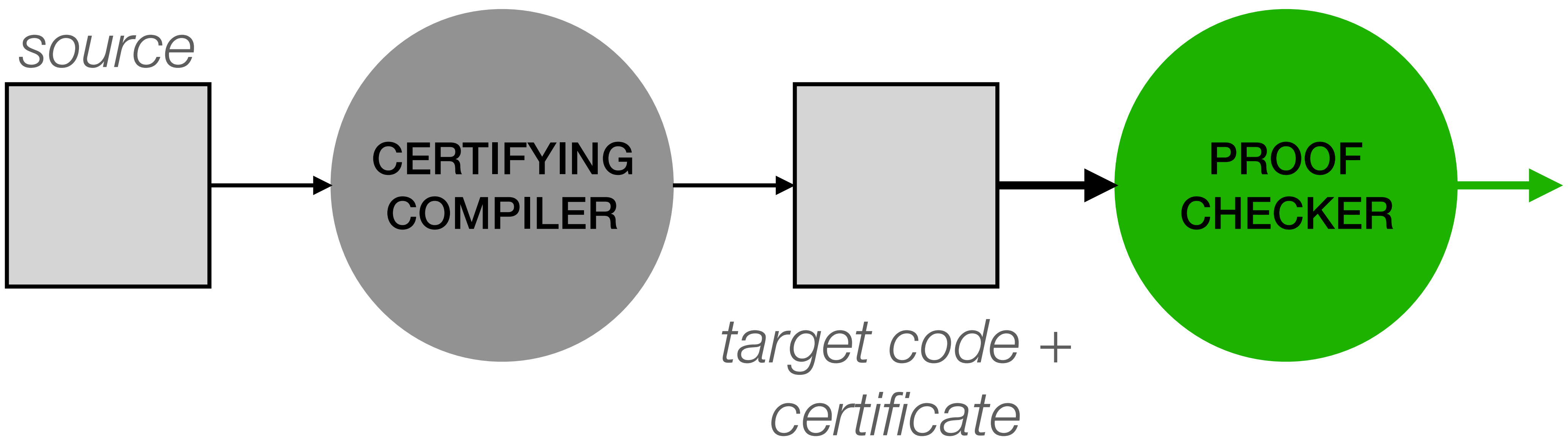
Verified, Validated, Certifying

2 . Translation validation [***Translation Verification***]

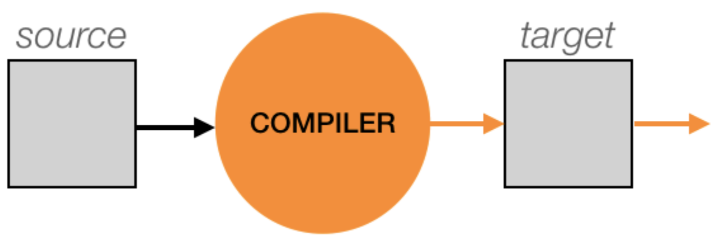


Verified, Validated, Certifying

3. Certifying compiler [***Proof-carrying Code***]

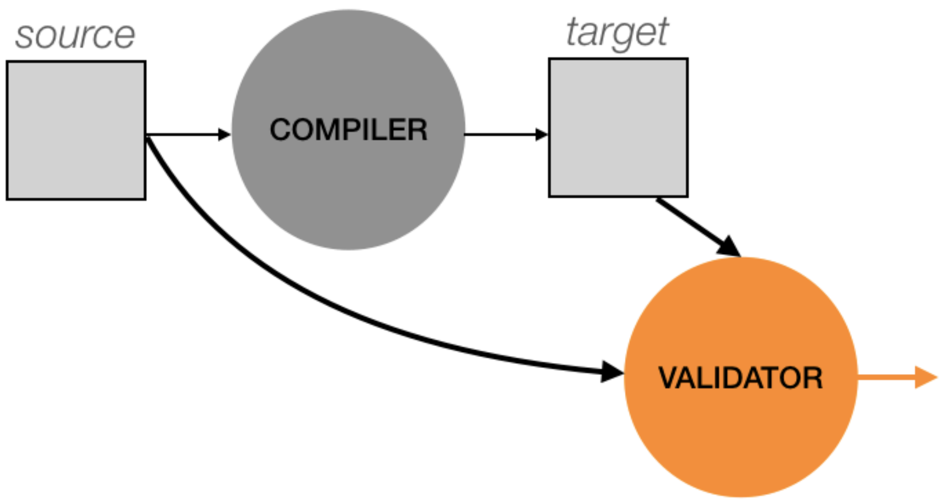


1. Verified transformation [***Compiler Correctness***]

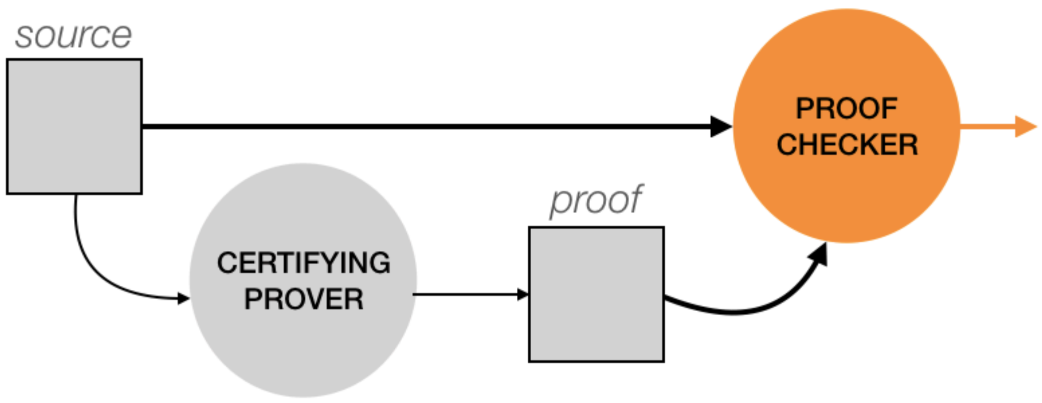


*total and correct

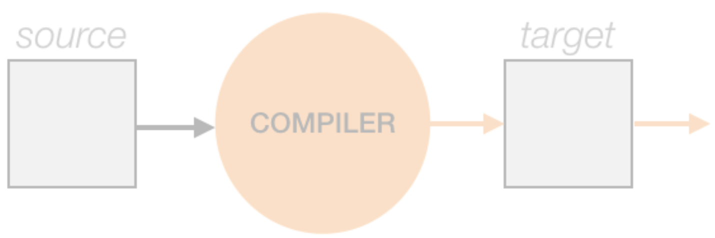
2 . Translation validation [***Translation Verification***]



3. Certifying compiler [***Proof-carrying Code***]

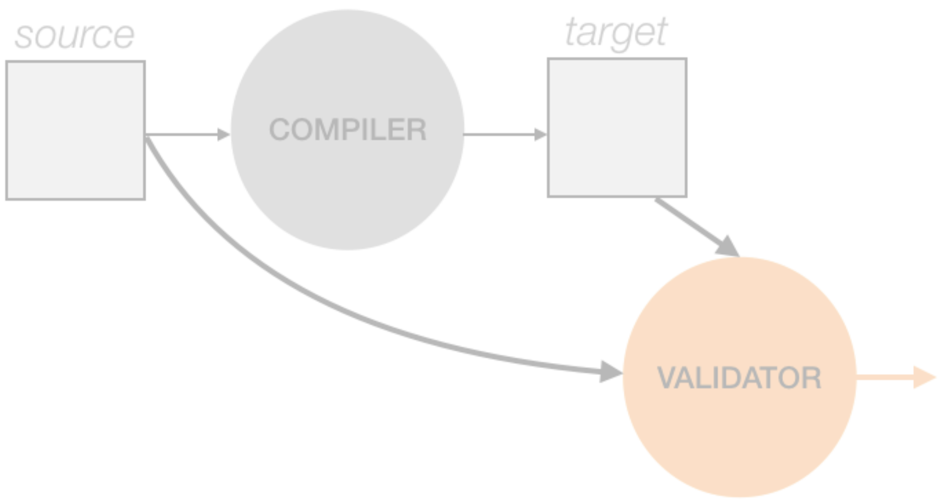


1. Verified transformation [***Compiler Correctness***]

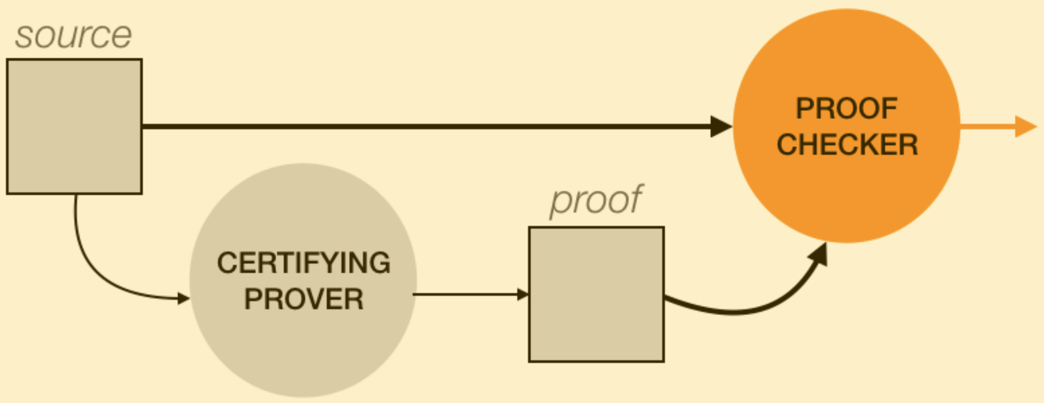


*total and correct

2 . Translation validation [***Translation Verification***]



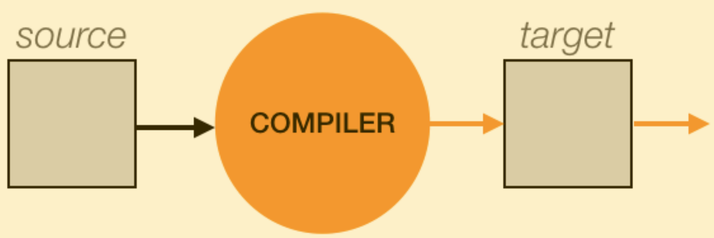
3. Certifying compiler [***Proof-carrying Code***]



[Leroy `06]

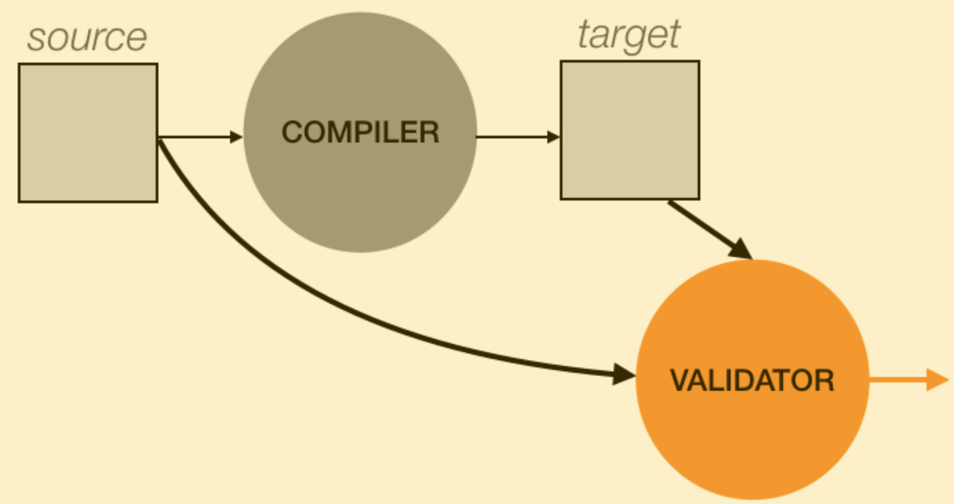
=> External solver with verified validation

1. Verified transformation [***Compiler Correctness***]

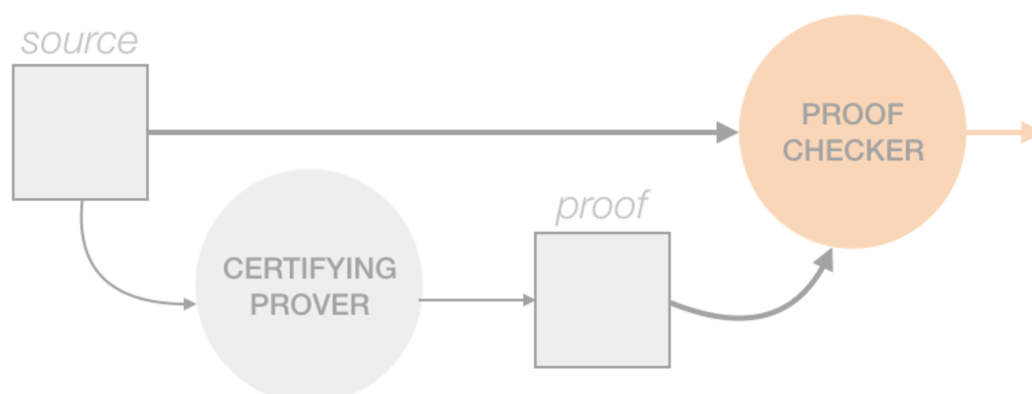


*total and correct

2 . Translation validation [***Translation Verification***]

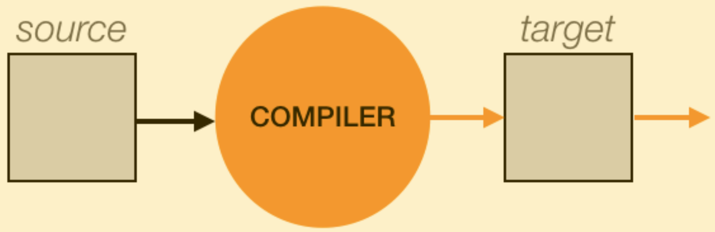


3. Certifying compiler [***Proof-carrying Code***]



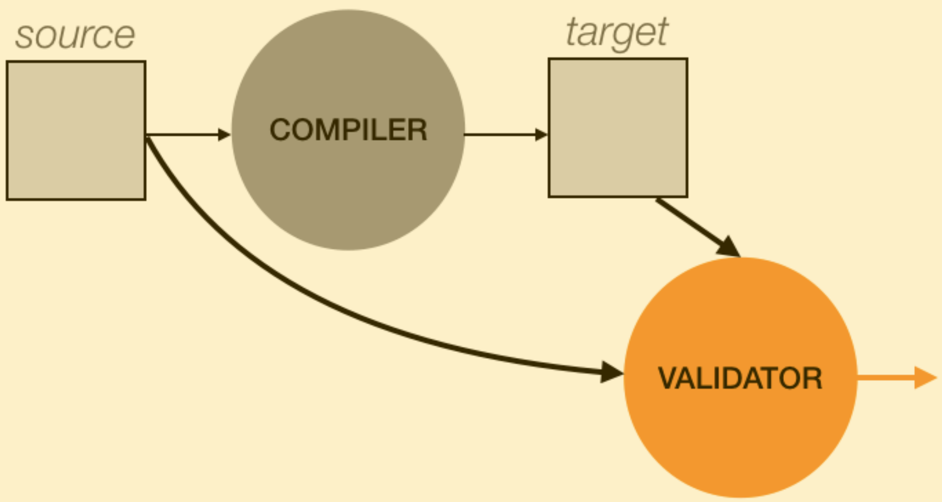
=> External solver with verified validation

1. Verified transformation [***Compiler Correctness***]



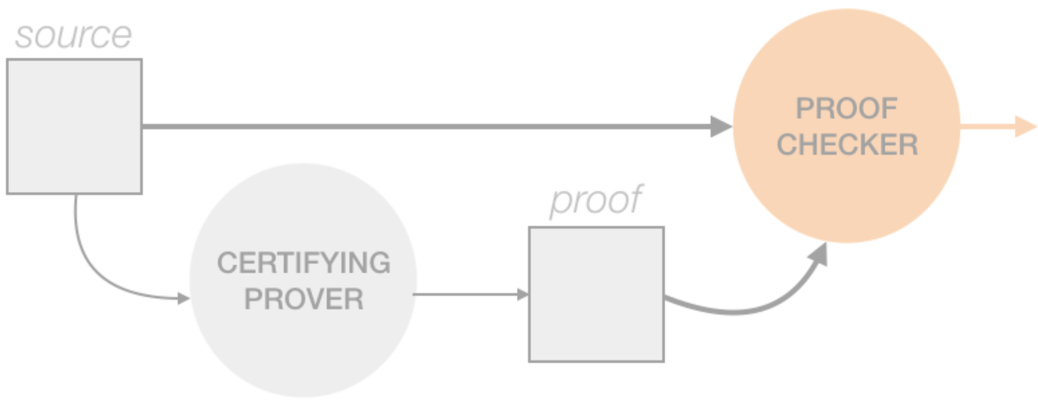
*total and correct

2 . Translation validation [***Translation Verification***]



[Tristan and Leroy `10]

3. Certifying compiler [***Proof-carrying Code***]



CompCert

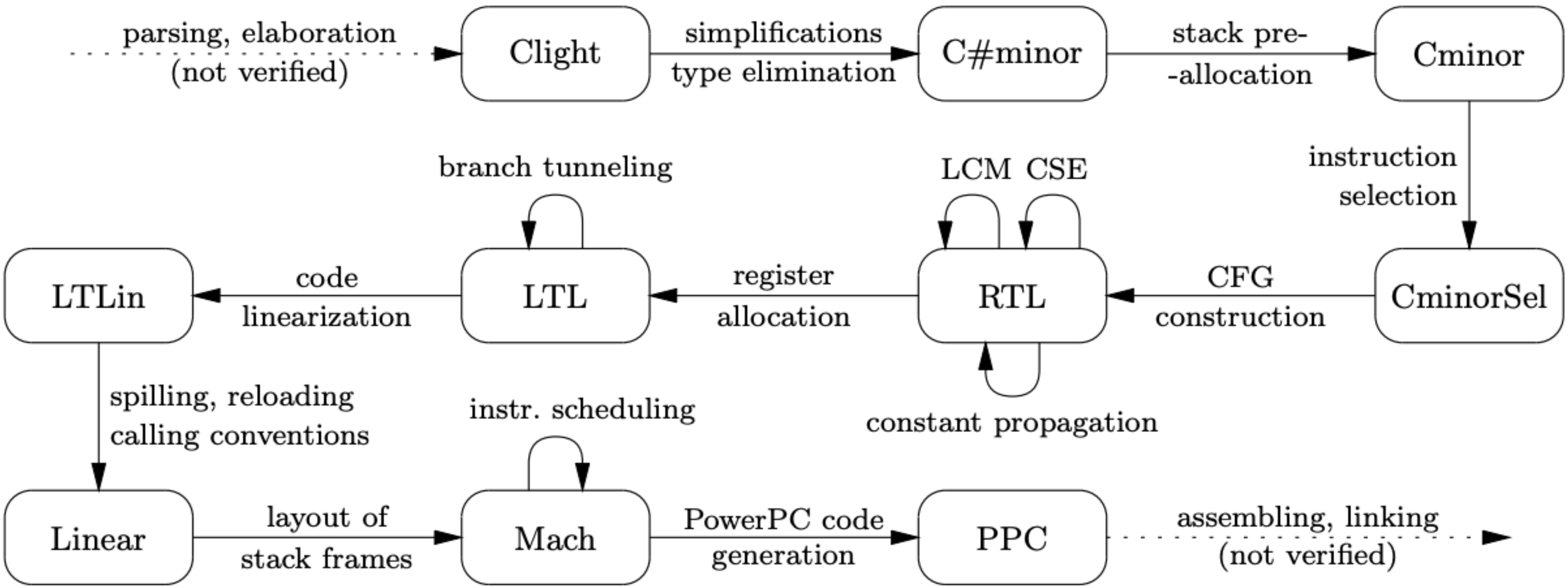


Figure 1: Compilation passes and intermediate languages.

CompCert

formal specification

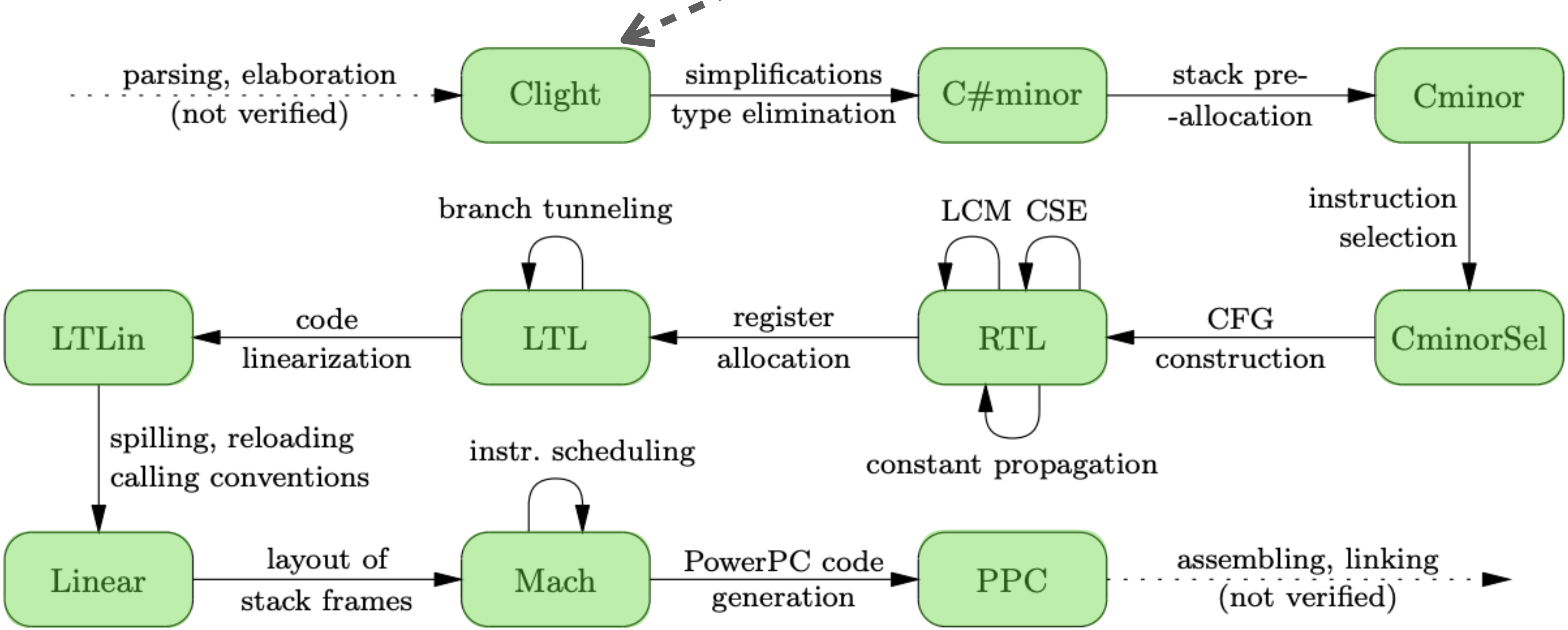


Figure 1: Compilation passes and intermediate languages.

CompCert

Verified Software Toolchain

formal specification

→ semantic analysis tools [Appel '11]

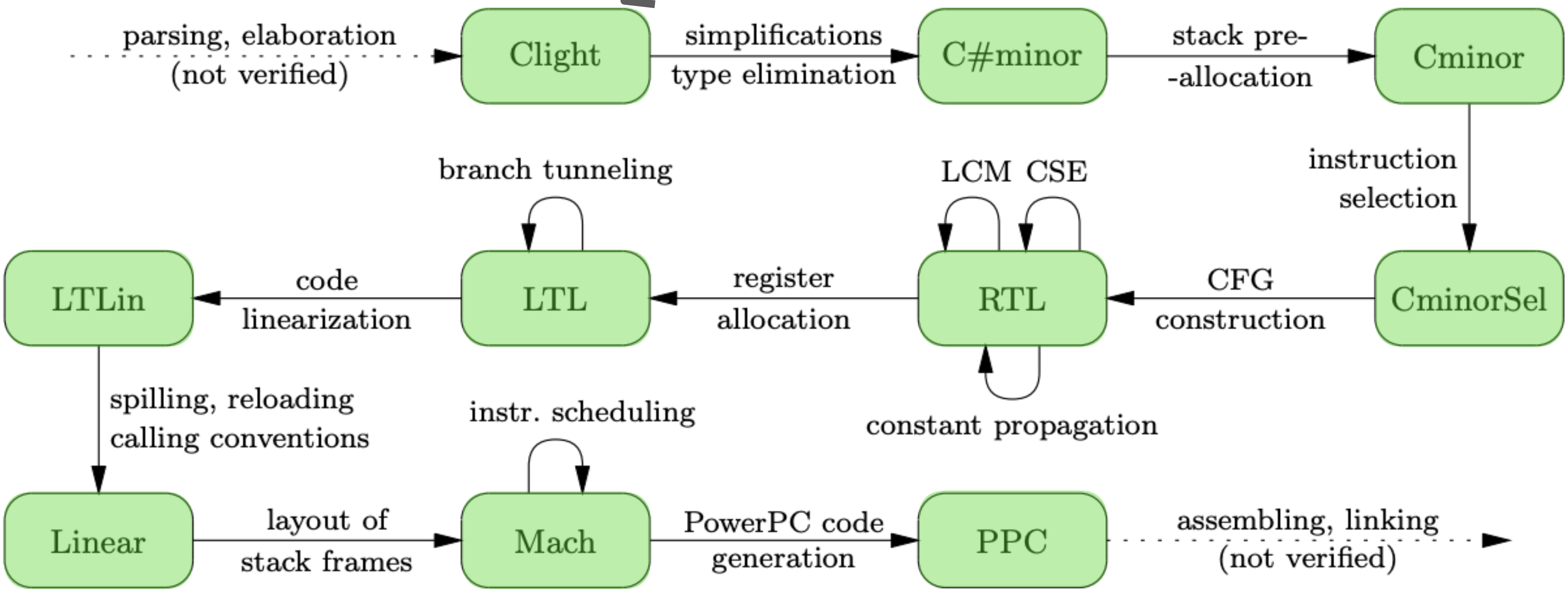


Figure 1: Compilation passes and intermediate languages.

CompCert

semantic preservation*

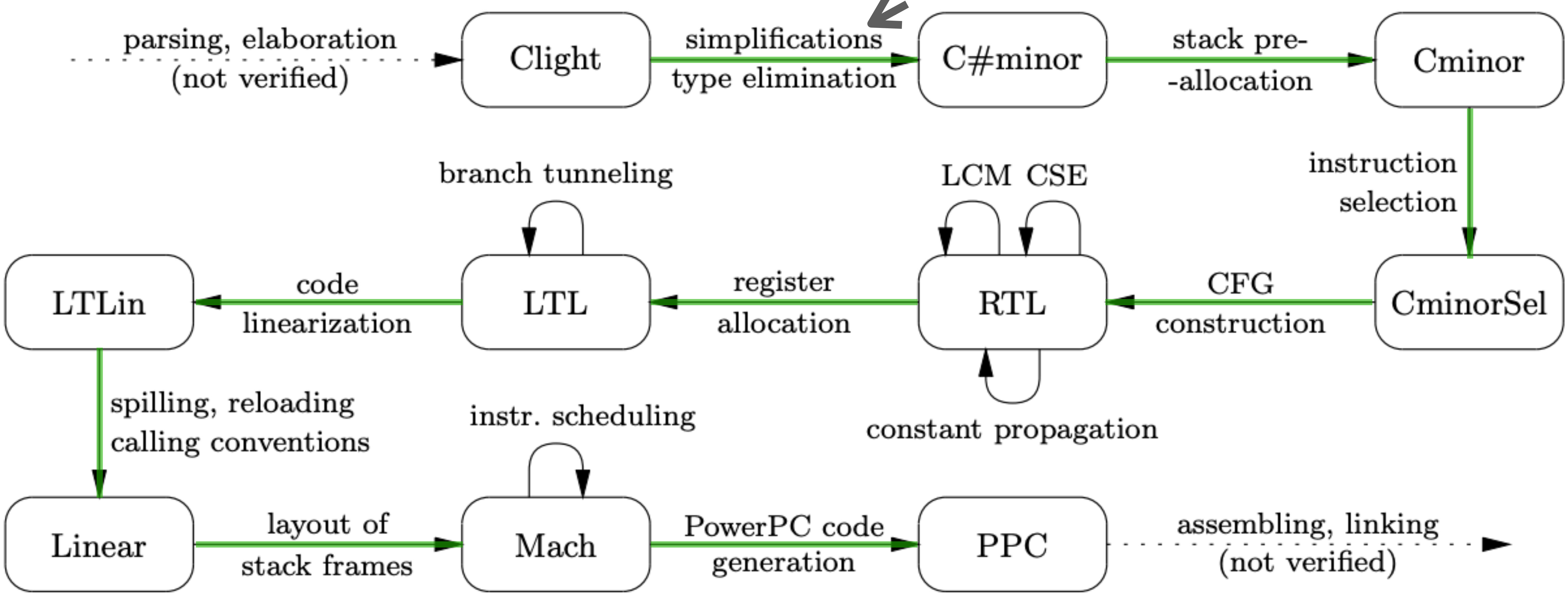


Figure 1: Compilation passes and intermediate languages.

CompCert

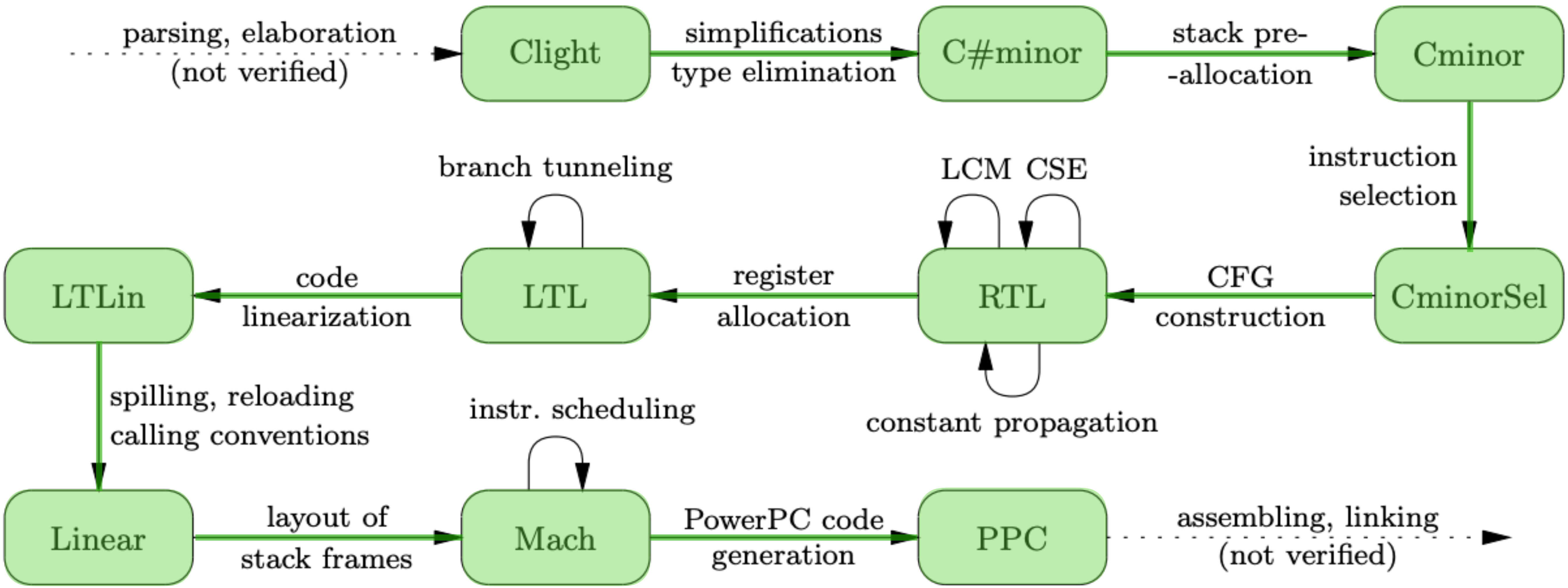


Figure 1: Compilation passes and intermediate languages.

Semantic Preservation

- $Spec(B)$: functional specification of observable behavior

Semantic Preservation

- $Spec(B)$: functional specification of observable behavior
- B : observable behavior (trace properties of I/O)
    - “going wrong” (run-time error), termination, divergence

Semantic Preservation

- $Spec(B)$: functional specification of observable behavior
- B : observable behavior (trace properties of I/O)
  - “going wrong” (run-time error), termination, divergence
- $C \models Spec$ if

  A. C cannot go wrong

  B. All behaviors B satisfy $Spec$

Correctness Property

$$S \models Spec \implies C \models Spec$$

Compiled code C preserves the fact that the source code S satisfies the specification.

Proving Semantic Preservation

$$\begin{tikzcd}
S_1 \arrow[r, dash, "\sim"] \arrow[d, "t"'] & S_1' \arrow[d, dotted, "t"] \\
S_2 \arrow[r, dash, dotted, "\sim"] & S_2'
\end{tikzcd}$$

Proving Semantic Preservation

$$
\begin{tikzcd}
S_1 \arrow[r, dash, "\sim"] \arrow[d, "t"'] & S_1' \arrow[d, dotted, "t^*"] \\
S_2 \arrow[r, dash, dotted, "\sim"] & S_2'
\end{tikzcd}
$$

Safety Precondition

- Compilation result will match the semantics of the input if **if program is "safe"** (no runtime errors)

Safety Precondition

- Compilation result will match the semantics of the input if **if program is "safe"** (no runtime errors)
- Need to prove that **input program is safe**

Correctness Weakness

TURING AWARD LECTURE

Reflections on Trusting Trust

To what extent should one trust a statement that a program is free of Trojan horses? Perhaps it is more important to trust the people who wrote the software.

KEN THOMPSON

Correctness Weakness

TURING AWARD LECTURE

Reflections on Trusting Trust

To what extent should one trust a statement that a program is free of Trojan horses? Perhaps it is more important to trust the people who wrote the software.

You can't trust code that you did not totally create yourself.

KEN THOMPSON

CompCert

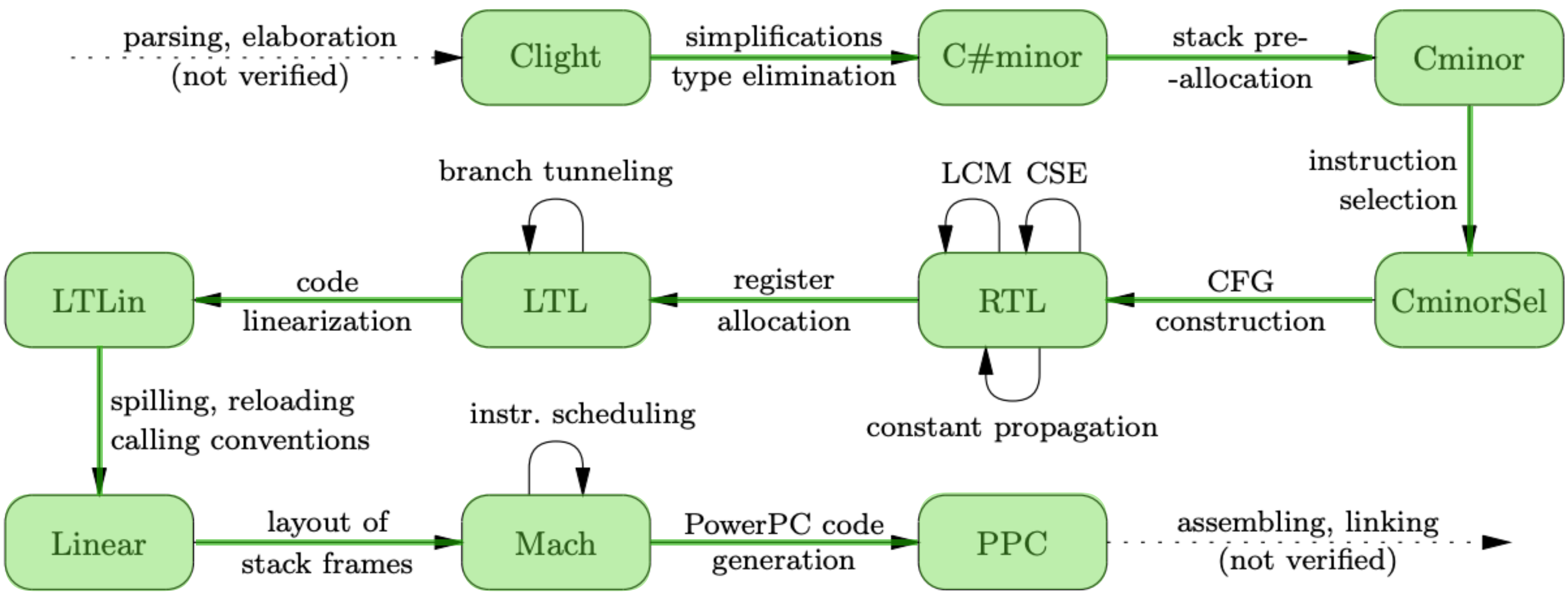


Figure 1: Compilation passes and intermediate languages.

CompCert

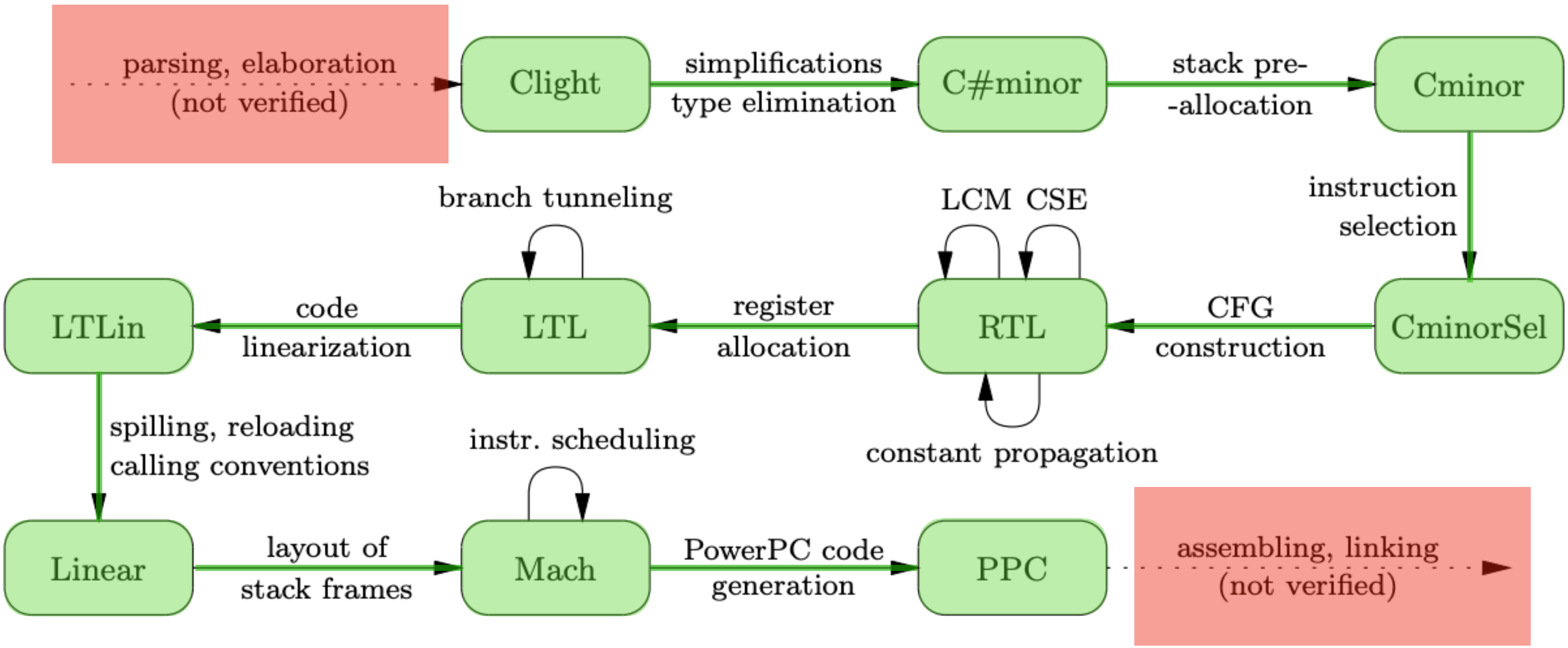


Figure 1: Compilation passes and intermediate languages.

Correctness Weakness

- Only runs after the preprocessing step

Correctness Weakness

- Only runs after the preprocessing step
  - *Astrèe* [Cousot et al ’05], *Verasco* [Jourdan et al ’15]

Correctness Weakness

- Only runs after the preprocessing step
  - *Astrèe* [Cousot et al ’05], *Verasco* [Jourdan et al ’15])
- Reliant on less verifiable assumptions

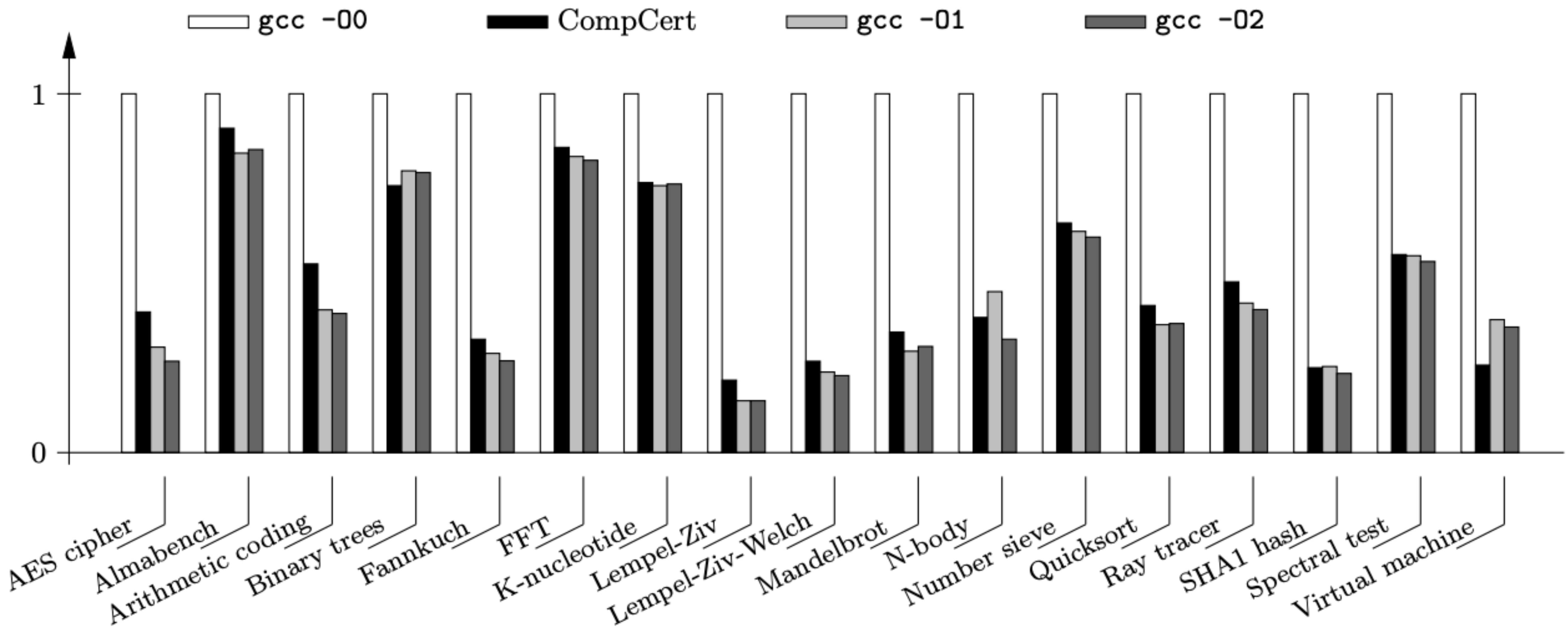
Correctness Weakness

- Only runs after the preprocessing step
  - *Astrèe* [Cousot et al ’05], *Verasco* [Jourdan et al ’15])
- Reliant on less verifiable assumptions
  - Coq’s correctness (*CertiCoq* [Anand et al ’17])

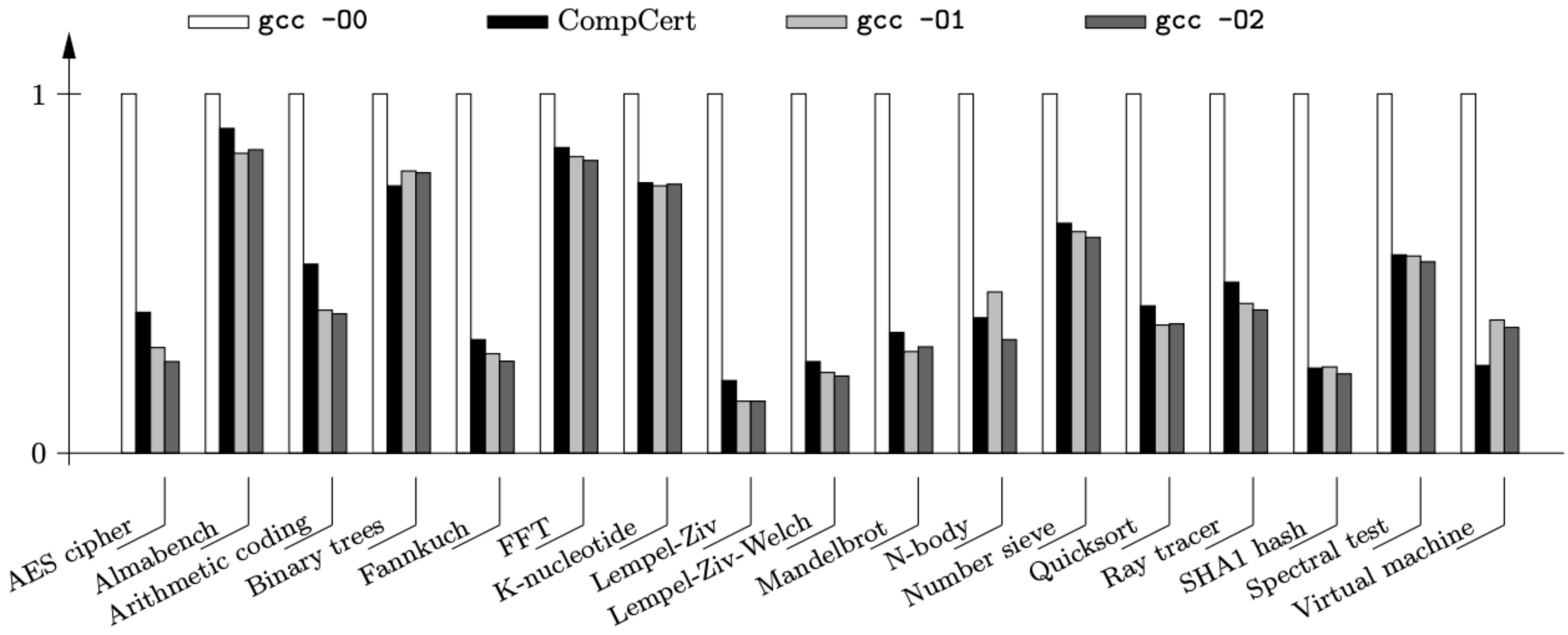
Correctness Weakness

- Only runs after the preprocessing step
  - *Astrèe* [Cousot et al ’05], *Verasco* [Jourdan et al ’15])
- Reliant on less verifiable assumptions
  - Coq’s correctness (*CertiCoq* [Anand et al ’17])
  - Formal specification of C & PowerPC assembly

Performance



Performance



competitive with `gcc -O1`

Bugs, revisited.

[Yang et al 2011]

Bugs, revisited.

[Yang et al 2011]

- CompCert: errors only found in unverified parts (parser and model of machine)

Bugs, revisited.

[Yang et al 2011]

- CompCert: errors only found in unverified parts (parser and model of machine)

- Other compilers: errors everywhere

Bugs, revisited.

[Yang et al 2011]

- CompCert: errors only found in unverified parts (parser and model of machine)

- Other compilers: errors everywhere

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

Critical Use Cases

- AirBus

Critical Use Cases

- AirBus
- MTU Friedrichshafen (nuclear energy)

Critical Use Cases

- AirBus
- MTU Friedrichshafen (nuclear energy)
- High-Assurance Cyber Military Systems (HACMS) [Fisher et al, ’17]

Critical Use Cases

- AirBus
- MTU Friedrichshafen (nuclear energy)
- High-Assurance Cyber Military Systems (HACMS) [Fisher et al, '17]
- PhD Theses

Critical Use Cases

"a ***realistic*** compiler"

- AirBus
- MTU Friedrichshafen (nuclear energy)
- High-Assurance Cyber Military Systems (HACMS) [Fisher et al, '17]
- PhD Theses

Concluding Remarks

- Still some correctness and safety weaknesses
- Useful for safety critical code (that doesn't have to run fast)

Concluding Remarks

- Still some correctness and safety weaknesses
- Useful for safety critical code (that doesn't have to run fast)
- Future work -

CAKEML
A Verified Implementation of ML

deep spec

Verified
Software
Toolchain

Certi Coq

Vellvm
verified
LLVM

...

Concluding Remarks

- Still some correctness and safety weaknesses
- Useful for safety critical code (that doesn't have to run fast)
- Future work -

CAKEML
A Verified Implementation of ML

deep spec

Verified Software Toolchain

Certi Coq

Vellvm verified LLVM

...

Concluding Remarks

- Still some correctness and safety weaknesses
- Useful for safety critical code (that doesn't have to run fast)

Type Preserving Compilation

- Future work -

CAKEML
A Verified Implementation of ML

deep spec

Verified Software Toolchain

Certi Coq

Vellvm
verified LLVM

...

Concluding Remarks

- Still some correctness and safety weaknesses
- Useful for safety critical code (that doesn't have to run fast)

Type Preserving Compilation

- Future work -

CAKEML
A Verified Implementation of ML

deep spec

Verified Software Toolchain

Certi Coq

Vellvm
verified LLVM

...

Principle 1: Erase the types! Compiler correctness is a stronger property than type preservation, anyway.

Thanks!