



**HAL**  
open science

# Formally Verified Advanced Optimizations for RISC-V

David Monniaux, Sylvain Boulmé, Léo Gourdin

► **To cite this version:**

David Monniaux, Sylvain Boulmé, Léo Gourdin. Formally Verified Advanced Optimizations for RISC-V. RISC-V Summit Europe 2023, Jun 2023, Barcelona, Spain. hal-04748961

**HAL Id: hal-04748961**

**<https://hal.science/hal-04748961v1>**

Submitted on 22 Oct 2024

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



# FORMALLY VERIFIED ADVANCED OPTIMIZATIONS FOR RISC-V

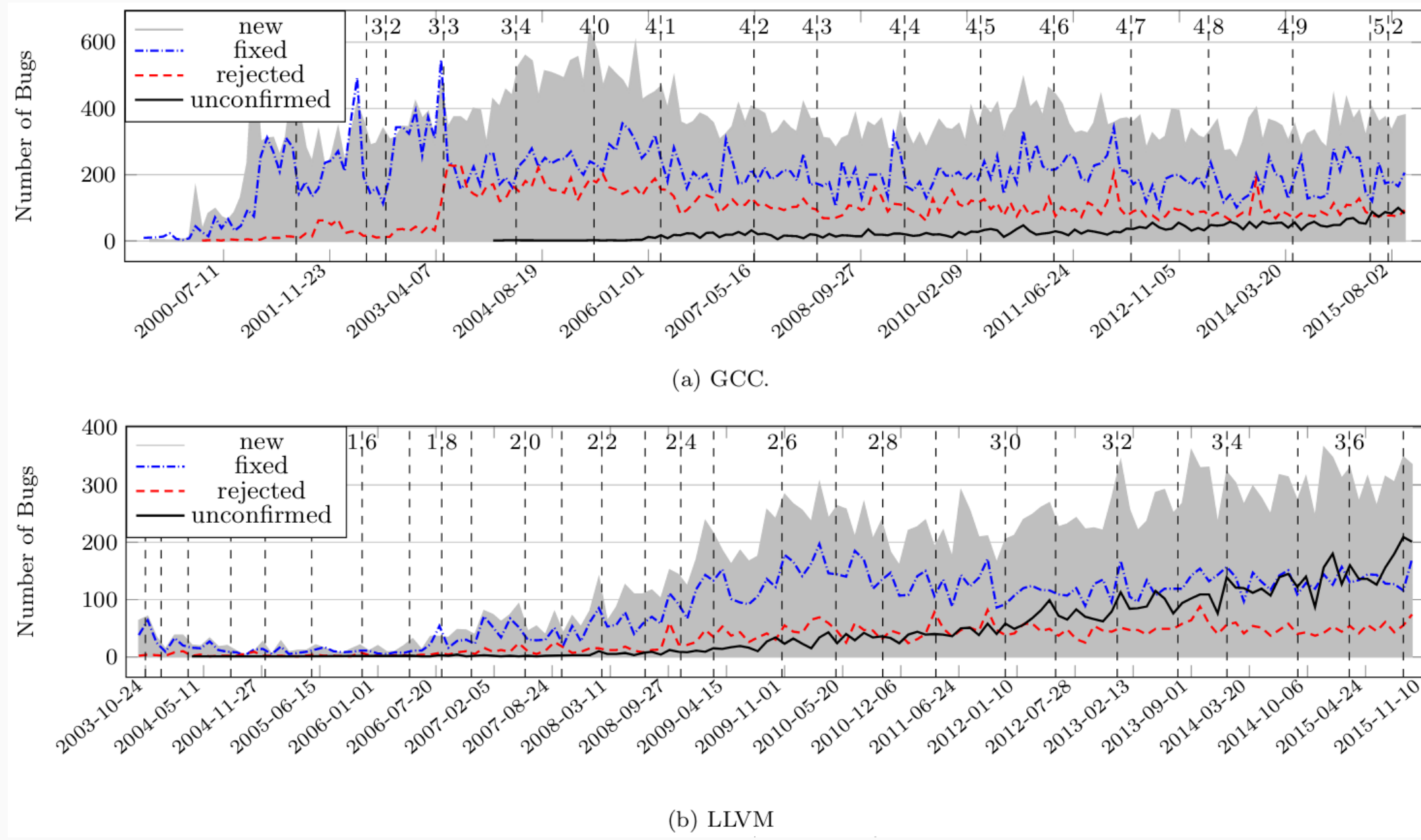


David Monniaux, Sylvain Boulmé, Léo Gourdin\*  
 Université Grenoble Alpes, CNRS, Grenoble INP, Verimag



<https://gricad-gitlab.univ-grenoble-alpes.fr/certicompil/Chamois-CompCert>

## Avoiding Bugs in GCC & LLVM (cf. [1])?



**CompCert<sup>1</sup> solution** (ACM Software System Award 2021):  
 the *1<sup>st</sup> formally verified* (= machine-checked mathematical proof of correctness) compiler optimizing **safety-critical** software [2, 3].

<sup>1</sup><https://www.absint.com/compcert/>

## Our goal:

### verified RISC-V optimizations

Reduced ISA & In-order cores  
 ⇒ clever optimizations needed!  
 E.g. way simpler addressing modes:

```
ldr x0, [x0, w1, sxtw#3]
```

Aarch64 (ARMv8-A)

```
slli x6, x11, 3
add x6, x10, x6
ld x6, 0(x6)
```

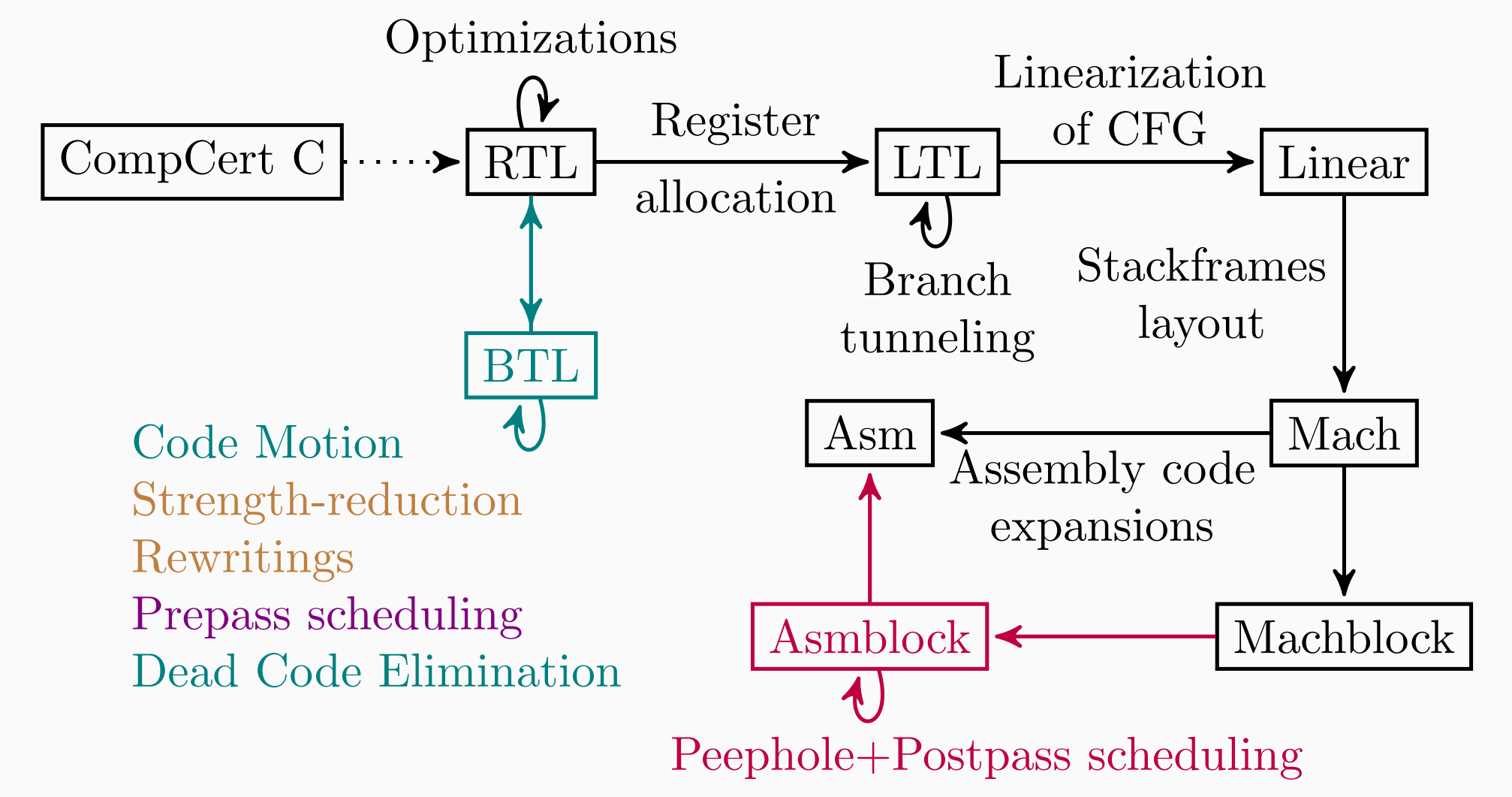
RISC-V

## Our solution:

### a versatile validation framework

- Supports many optimizations;
- Independent of the architecture;
- **1<sup>st</sup> verified strength-reduction** of induction variables.

## Chamois-CompCert Extensions

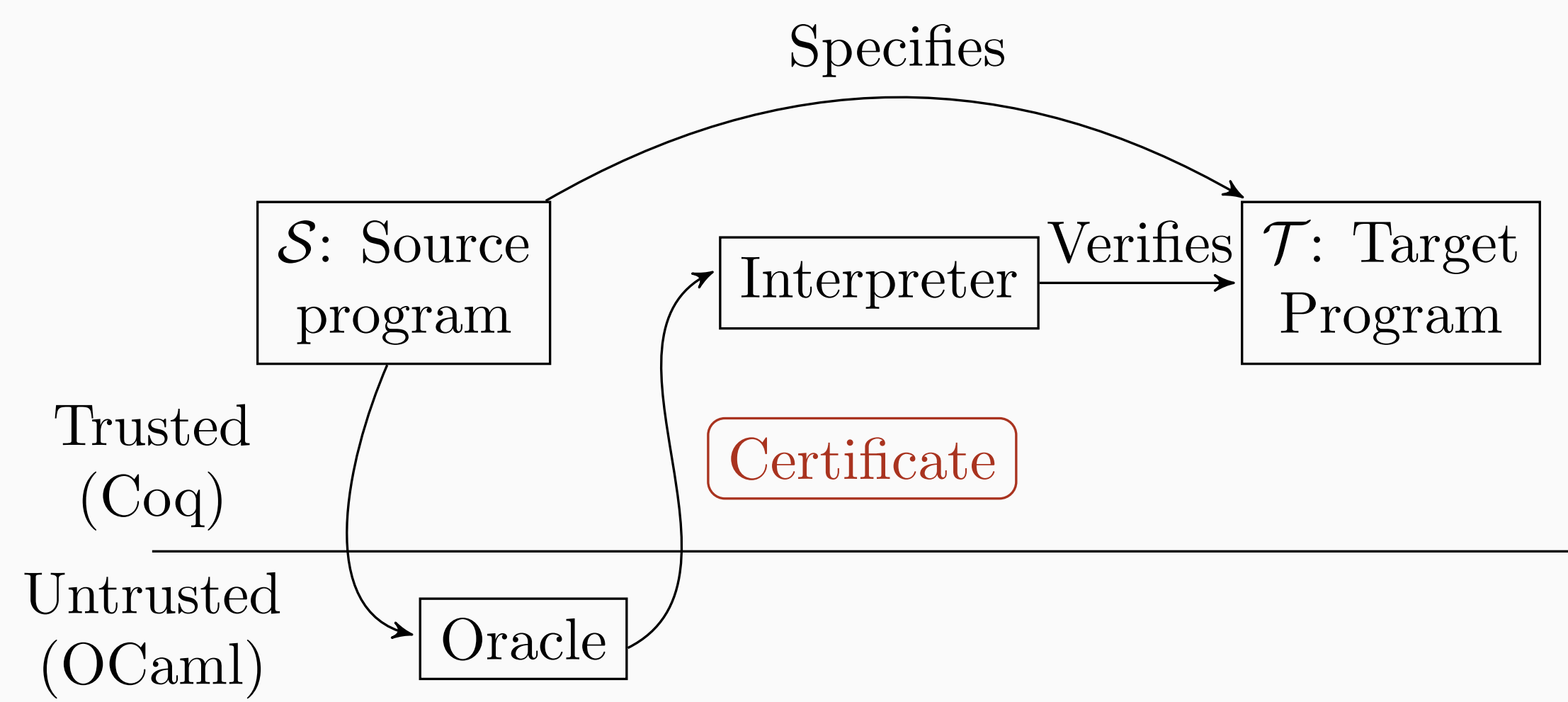


Legend:

- **Brown:** RISC-V only
- **Violet:** AArch64 + ARMv7 + RISC-V + K VX
- **Red:** AArch64 + K VX
- **Teal:** All (AArch64 + ARMv7 + RISC-V + K VX + PPC + x86)

## Our General Purpose Translation Validator

The oracle takes source program  $\mathcal{S}$  and yields its optimized version  $\mathcal{T}$  along with a certificate. A verified symbolic execution interpreter then ensures semantic preservation, and aborts compilation in case of failure.



## Validating the Lazy Code Transformations Oracle

### Combining and improving Lazy Code Motion [4] & Lazy Strength Reduction [5].

- Search for *reducible* multiplicative operators;
- Based on data-flow analyses performed by an OCaml oracle;
- Supports decomposed patterns like a *left shift* + an *addition*;

### Optimizing in two steps:

1. *Lifting* the multiplication out of the loop;
2. Inserting *compensation code* in the loop body.

```
long main(long x, long n) {
    long i = 0;
    while (i < n) {
        x += i * 5;
        i += 3;
    }
    return x;
}
```

C source code

```
main:
    [...] //prelude
    addi x12, x0, 0 //i=0
.L100:
    slli x7, x12, 2 //t=i*4
    add x6, x12, x7 //t=i*4+i*5
    bge x12, x11, .L101 //i>=n?
    add x10, x10, x6 //x+=t
    addi x12, x12, 3 //i+=3
    j .L100
```

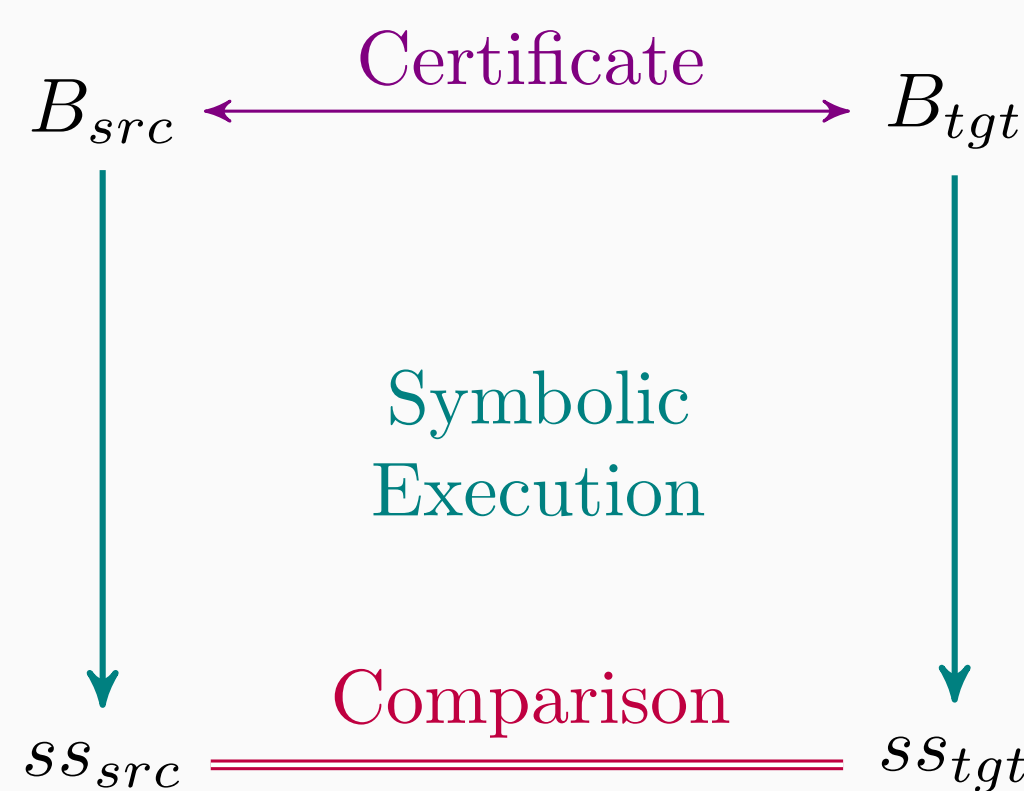
RISC-V ASM Before

```
main:
    [...] //prelude
    addi x12, x0, 0
    slli x7, x12, 2
    add x6, x12, x7
→ .L100:
    bge x12, x11, .L101
    add x10, x10, x6
    addi x6, x6, 15 //t+=15
    addi x12, x12, 3
    j .L100
```

RISC-V ASM After

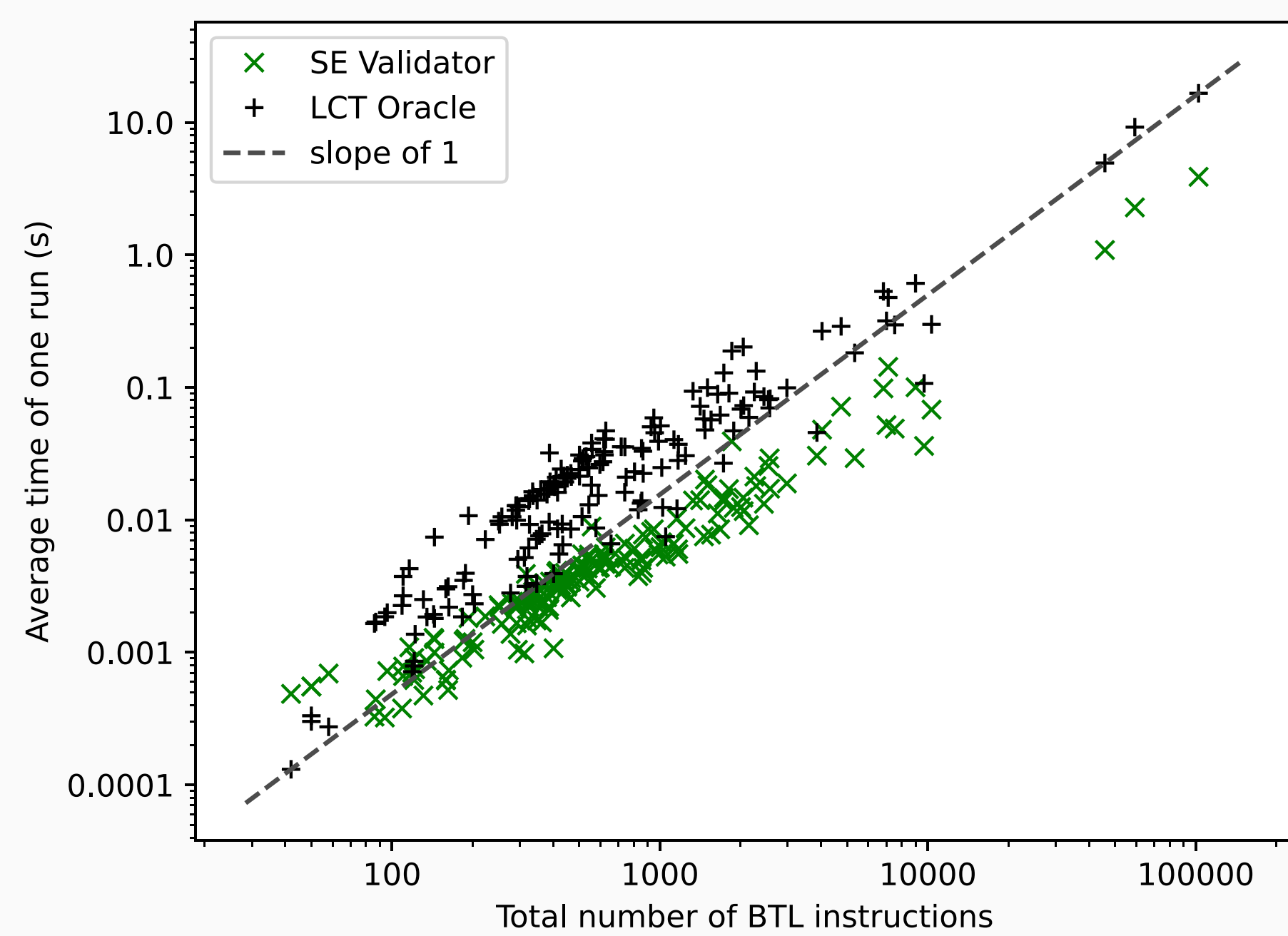
## Defensive Symbolic Simulation

For each pair of loop-free blocks  
 $(B_{src} \in \mathcal{S}, B_{tgt} \in \mathcal{T})$ ;  
 we compare the symbolic *states*  
 $(ss_{src}, ss_{tgt})$  resulting from their  
 symbolic execution.



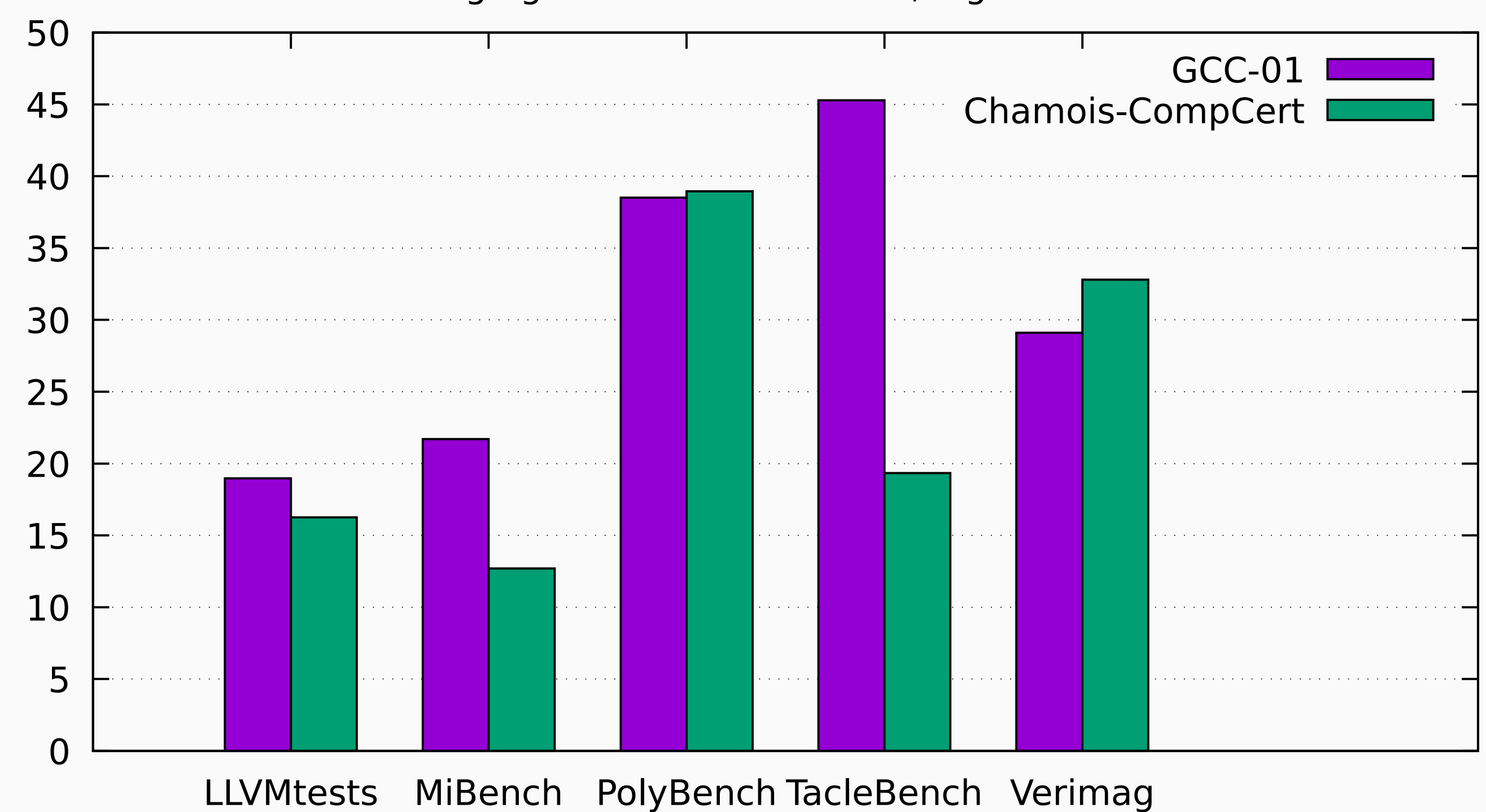
The **certificate** contains *invariants*  
 propagating information between  
 blocks.

## Compile Times That Scale



## Optimized Generated Code That You Can Trust

Comparing w.r.t. Official CompCert over five test suites  
 Percentage gain in execution time, higher is better



## References

- [1] Chengnian Sun et al. "Toward understanding compiler bugs in GCC and LLVM". en. In: *Proceedings of the 25th International Symposium on Software Testing and Analysis*. Saarbrücken Germany: ACM, 2016, pp. 294–305. ISBN: 978-1-4503-4390-9. DOI: 10.1145/2931037.2931074. URL: <https://dl.acm.org/doi/10.1145/2931037.2931074> (visited on 06/17/2022).
- [2] Xavier Leroy. "Formal verification of a realistic compiler". In: *Communications of the ACM* 52.7 (2009). DOI: 10.1145/1538788.1538814.
- [3] Daniel Kästner et al. "CompCert: Practical experience on integrating and qualifying a formally verified optimizing compiler". In: *ERTS 2018: Embedded Real Time Software and Systems*. SEE, 2018.
- [4] Jens Knoop, Oliver Rüthing, and Bernhard Steffen. "Optimal Code Motion: Theory and Practice". In: *ACM Transactions on Programming Languages and Systems* 16 (1995). DOI: 10.1145/183432.183443.
- [5] Jens Knoop, Oliver Rüthing, and Bernhard Steffen. "Lazy Strength Reduction". In: *Journal of Programming Languages* 1 (1993), pp. 71–91.

## Git Repo

