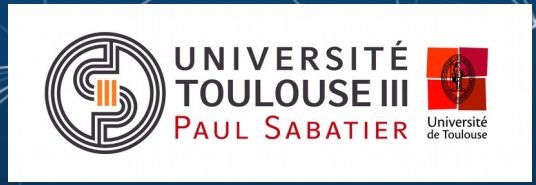
RISC-V support in OTAWA: Validation of the ISA description

Emmanuel Caussé, Hugues Cassé, Pascal Sainrat,
TRACES – IRIT – University of Toulouse

Paris 1-2 / 10 / 2019



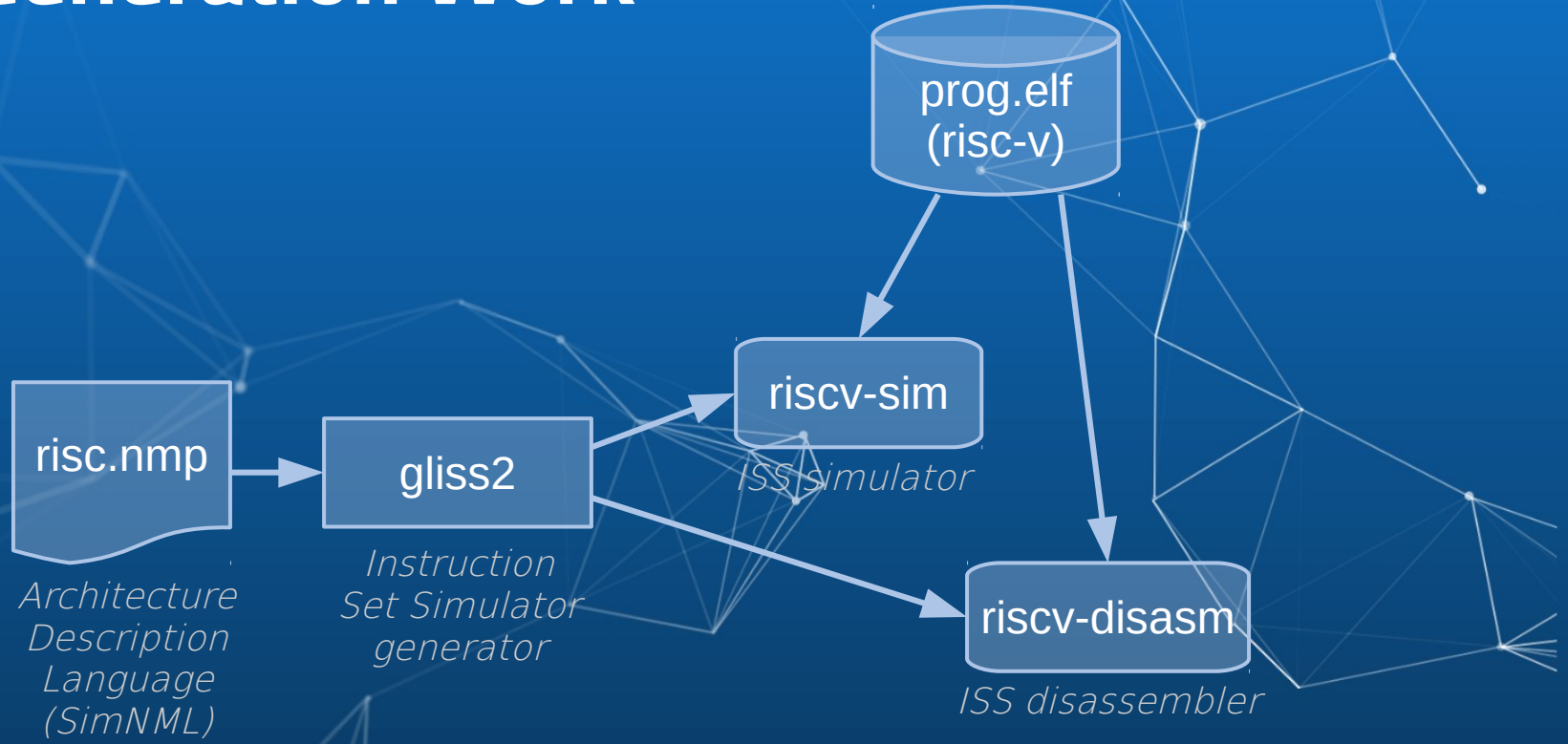
Introduction

- static analysis tools on machine code (WCET)
  - decoding and processing machine code
  - sound model of the Instruction Set Architecture (ISA)
- applied to RISC-V
  - Architecture Description Language (ADL) model
  - “verified” by co-simulation

Presentation Outline

- Introduction
- ISS generation
- ISA validation
- Towards static analysis
- Conclusion

ISS Generation Work



Instruction Set Architecture Description

SimNML

- types
- registers
- memories
- operations
- modes

[Freericks – 1991]

risc.nml

```
type byte      = int ( 8)
type word      = int (32)
type address   = card(32)

mem M          [32,  byte]
mem M32        [32,  word] alias = M

reg PC         [ 1, address] is_pc = 1
reg NPC        [ 1, address]
reg R          [32, word]
reg F          [32, float(23,9)]
```

Mode description

- addressing modes
- special format (register)
- special calculation (hardwired register)
- ...

risc.nml

```
mode reg_t (r: index) = r
        syntax =
                switch ( r ) {
                case  0:  "zero"
                case  1:  "ra"
                case  2:  "sp"
                case  3:  "gp"
                case  4:  "tp
                …
        }
        image  = format ( "%5b", r )
```

Operation description

- image – binary
- syntax – assembly
- action
  - imperative language
  - bit oriented
  - formally defined
  - synthesizable
    [Basu, Moona - 2003]
  - close to handbook pseudo-code
    (less error-prone?)

risc.nml

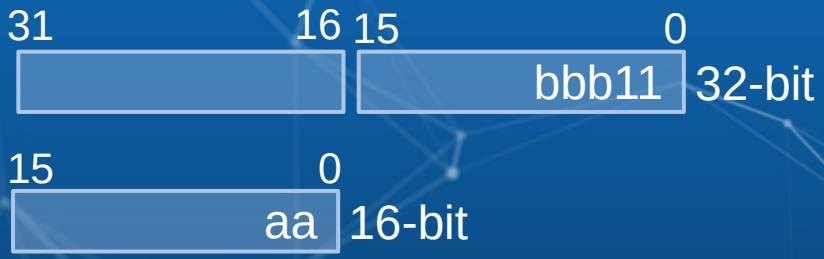
```
op addi(imm: int(12), s: reg_t, d: reg_t)

  syntax = format("addi %s, %s, %d", d, s, imm)

  image = format("%12b %s 000 %s 0010011", i, s, d)

  action = {
    if d != 0 then
      R[d] = R[s] + imm;
    endif;
  }
```

Our implementation

- no pseudo-code in [The RISC-V Instruction Set Manual, Volume I: User-Level ISA, Version 2.0 – 2014]
- several contributors
  - M. Frieb – Augsburg University (initial implementation)
  - E. Caussé – University of Toulouse
  - P. Sainrat – University of Toulouse
- overall results
  - 174 instructions 32-bit, 13 instructions 64-bit
  - extensions – 32-bit (I, M, A, F, D, C), 64-bit (I)
  - missing – 2 instructions 32-bit, 25 instructions 64-bit

RISC-V Freaks

Compact extension:



16-bit if aa = 11 ∧ bbb ≠ 111

risc.nml

```
let gliss_isize = “32,16”

op c_add(dest: enum(1..31), src2: enum(1..31))
        image = format("100 1 %5b %5b 10",dest,src2)
        ...
op add(src2: reg_t, src1: reg_t, dest: reg_t)
        image = format("0000000 %s %s 000 %s 0110011",
                src2, src1, dest)
        ...
```

Already supported by GLISS2 for ARM Thumb-2, PowerPC VLE , TriCore, Star12X, x86.

Implementation activity

Adding a new instruction

-  Understanding The handbook
-  Writing the binary encoding
- Writing the disassembly
-  Writing the action

Fixing an existing instruction

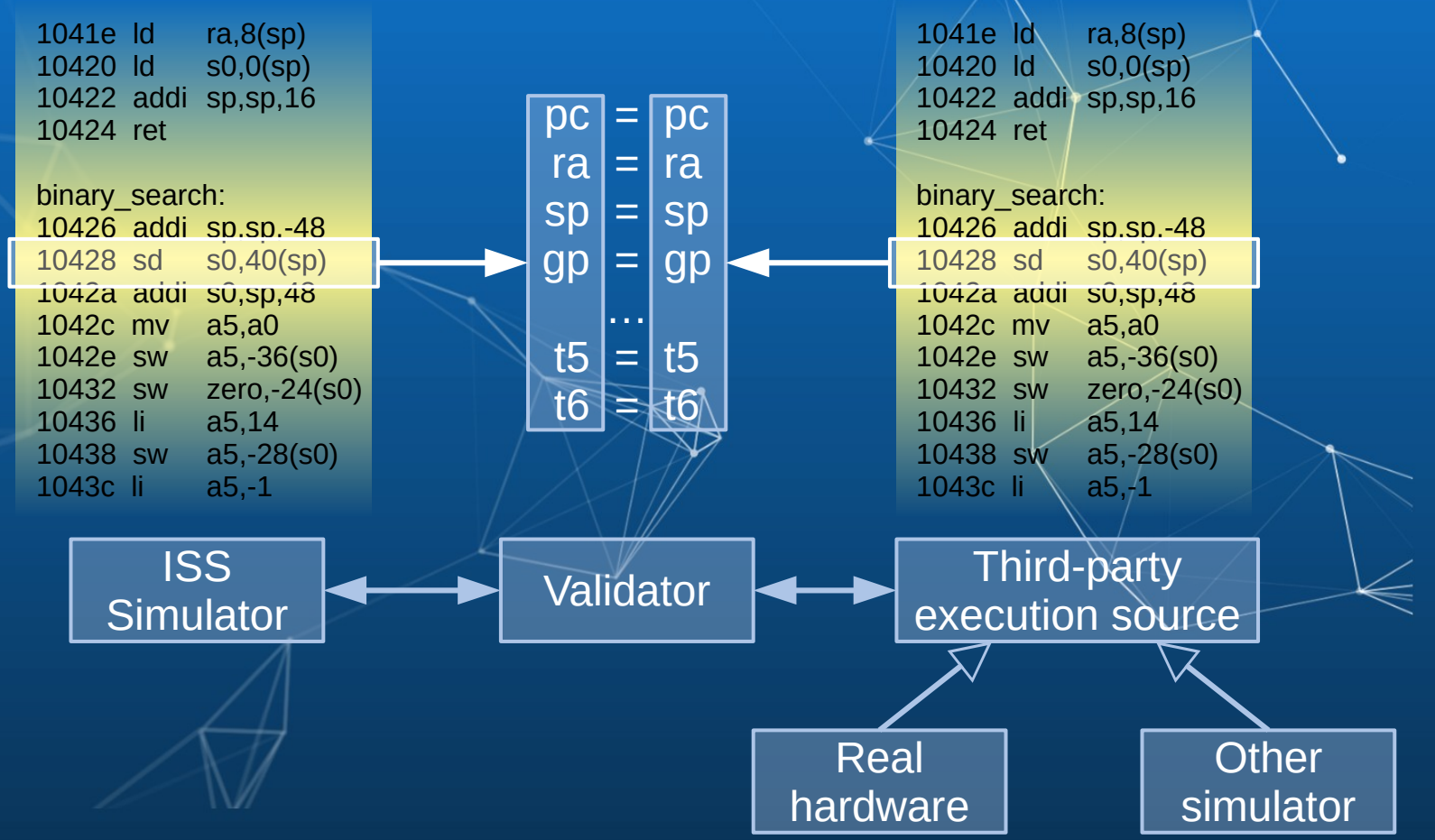
-  re-read the action → fix obvious mistakes
-  examine the simulation → detect anomalies

What's about the validity of the result?

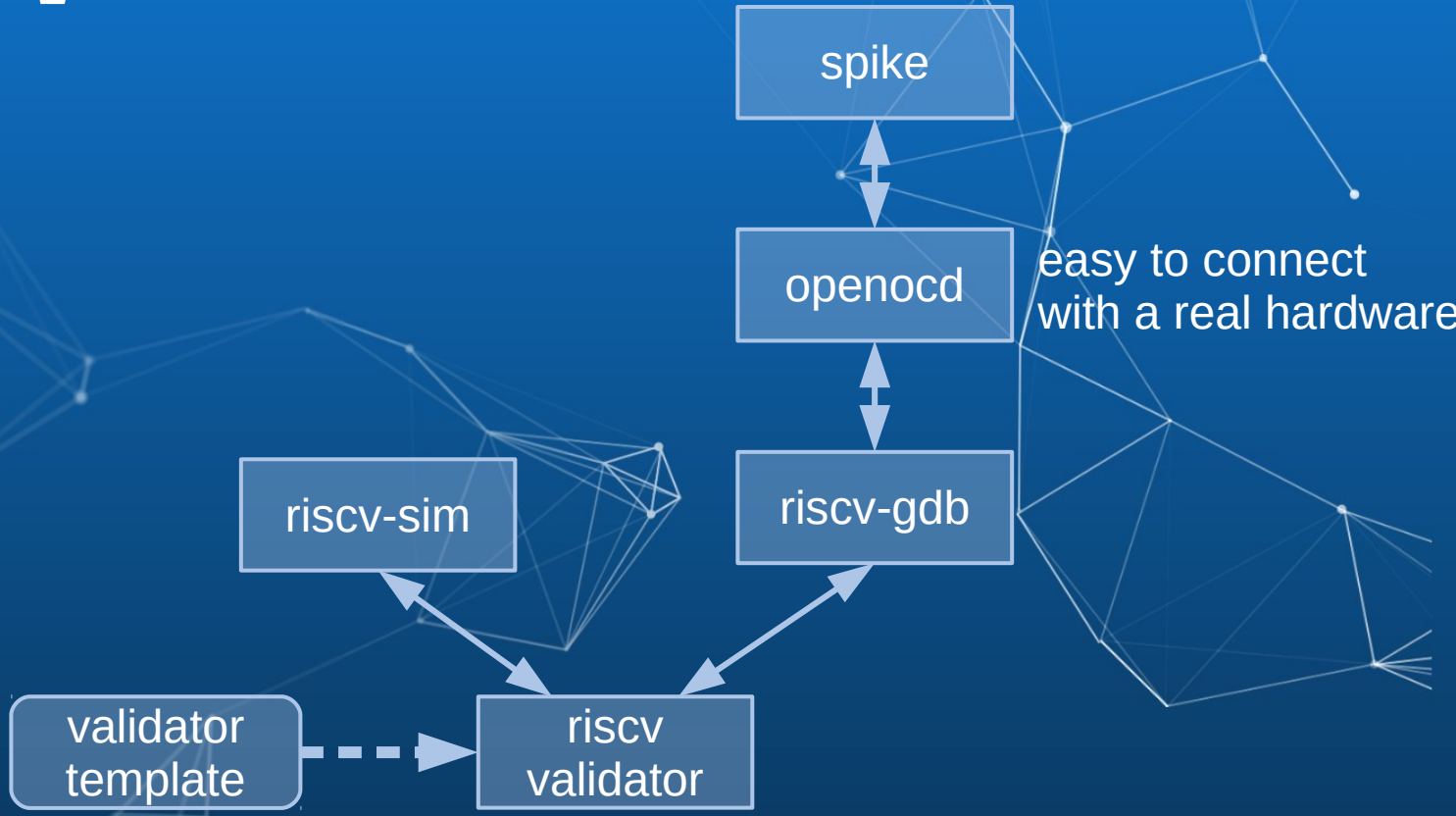
Presentation Outline

- Introduction
- ISS generation
- **ISA validation**
- Towards static analysis
- Conclusion

Co-simulation



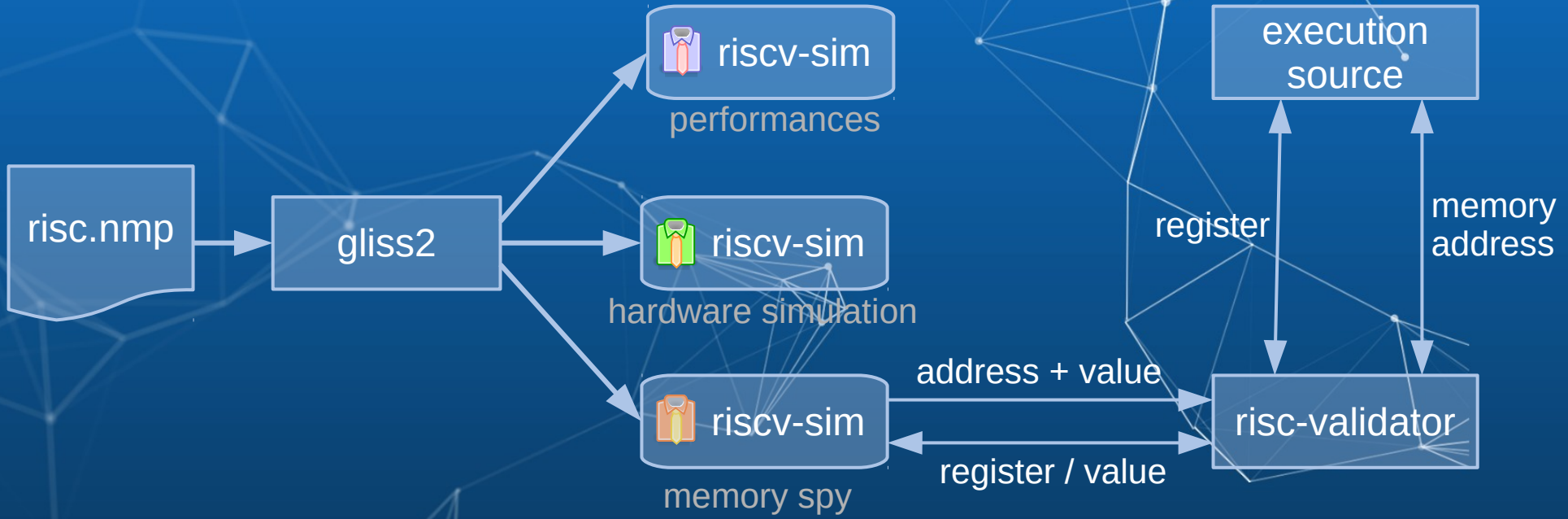
For RISC-V


spike
openocd
easy to connect
with a real hardware
riscv-sim
riscv-gdb
validator
template
riscv
validator
tuned by hand → gliss2

Experimentation

- benchmarks
  - riscv-tests (github) – (c) University of California
  - 1 test / instruction (217 tests – ~9500 lines of code)
- results
  - slow – 5-6x (doesn't matter)
  - > 100 – fixes
  - some instructions can't be tested!
    (internal / system – 13 instructions)

Memory Comparison



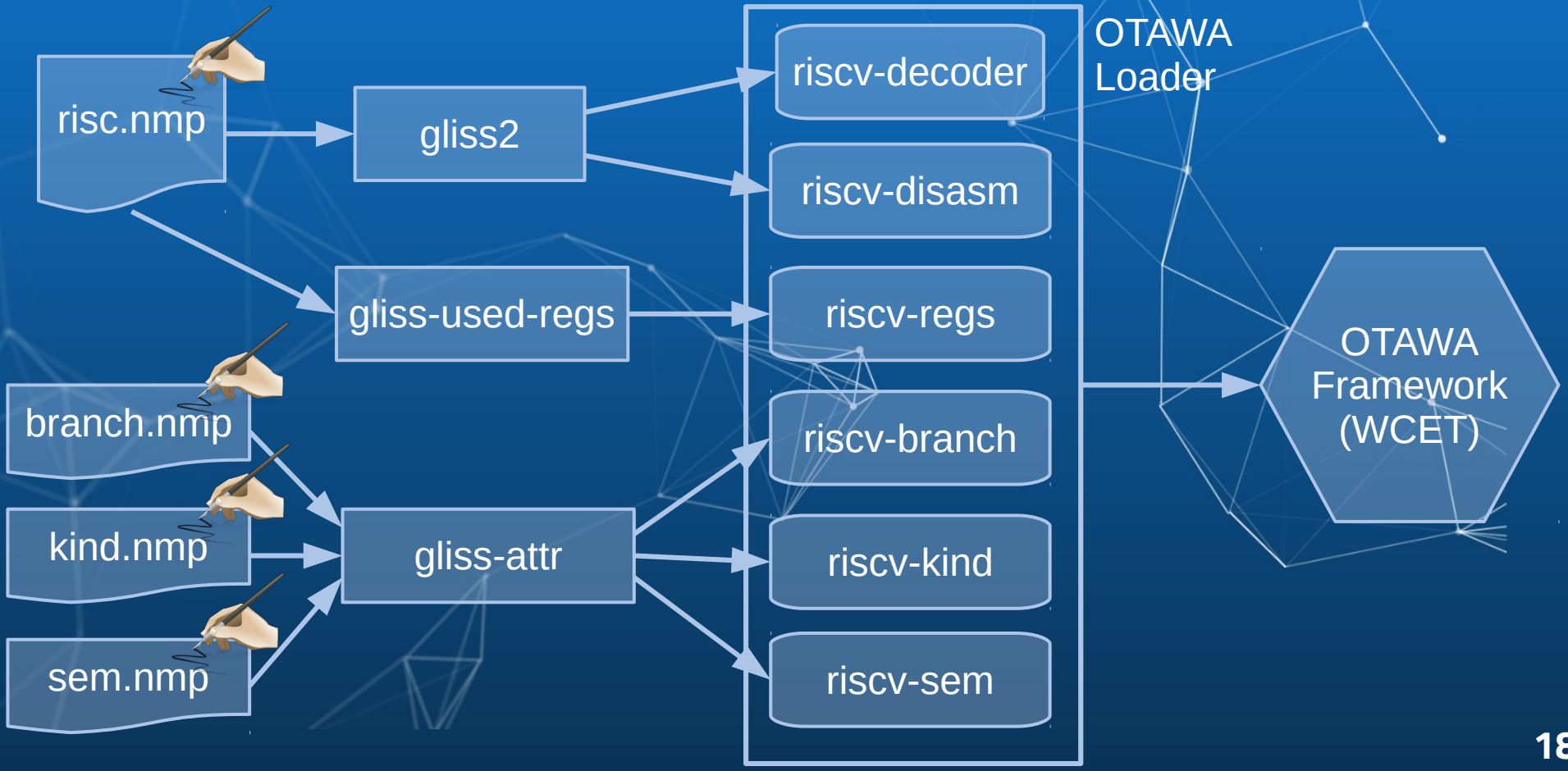
Partial conclusion

- It’s not a proof!
- We test if 2 machines are equivalent…
- Error = machine 1? machine 2? Both?
- But we improve confidence in our RISC-V ADL description (not so bad)

Presentation Outline

- Introduction
- ISS generation
- ISA validation
- **Towards static analysis**
- Conclusion

Binary static analyser: OTAWA

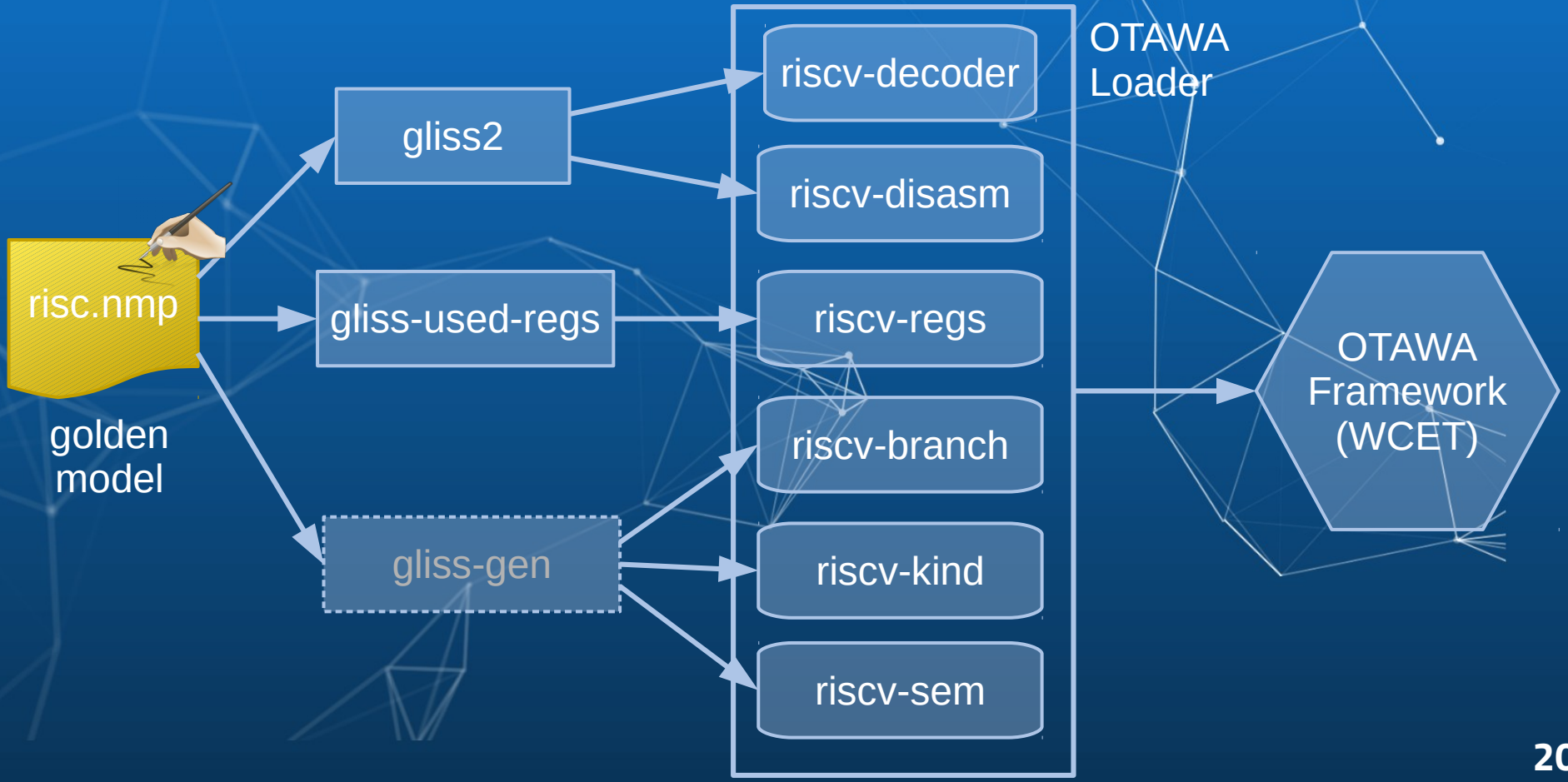


Verification experiment (TriCore)

[W.-T. Sun & al. *Validating Static WCET Analysis: A Method and Its Application.* WCET'19]

- TriCore Instruction Set (~330 instructions)
- co-simulation
  - GLISS ISS / TSIM
  - 67 fixes
- data flow analysis
  - OTAWA: machine instruction → semantic instructions (ISA independent)
  - simulation states ⊆ abstract state?
  - 71 fixes
- partial coverage of instruction set ⇐ compiler

Our objective



Why RISC-V?

- simple and small instruction set
  - gliss-gen demonstrator easier to experiment
- open “standard”
  - “open microarchitecture”?
  - full visibility of internals
  - better timing model for WCET calculation

Presentation Outline

- Introduction
- ISS generation
- ISA validation
- Towards static analysis
- **Conclusion**

Conclusion

- Validator
  - improve confidence in RISC-V ADL model
  - portable to different architectures
- Future...
  - benchmark selection to improve coverage (automatic generation?)
  - implement and experiment gliss-gen
  - WCET for RISC-V with open micro-architecture

Any question?

GLISS2
https://www.irit.fr/hg/TRACES/gliss2/trunk

Instruction Set
https://github.com/hcasse/riscv