Unlocking the Power of Formal Hardware Verification with CoSA and Symbolic QED

F. Lonsing, K. Ganesan, M. Mann,
S. Nuthakki, E. Singh, M. Srouji,
Y. Yang, S. Mitra, and C. Barrett

2019 International Conference on Computer Aided Design (ICCAD)
4-7 November 2019, Westminster, CO

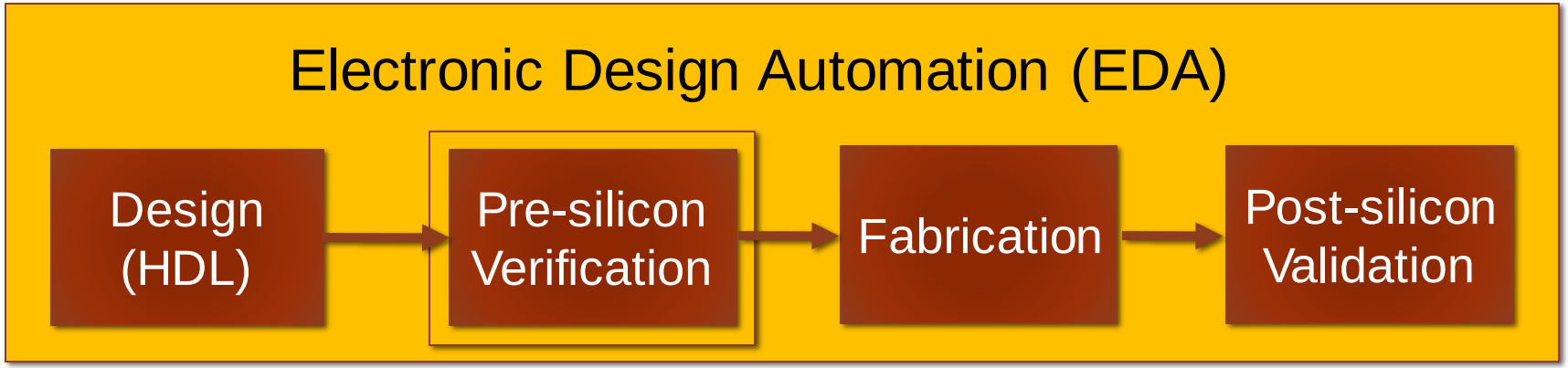
Students, Sponsors, Collaborators

Upscale

Scaling Up Formal Tools for POSH Open Source HW

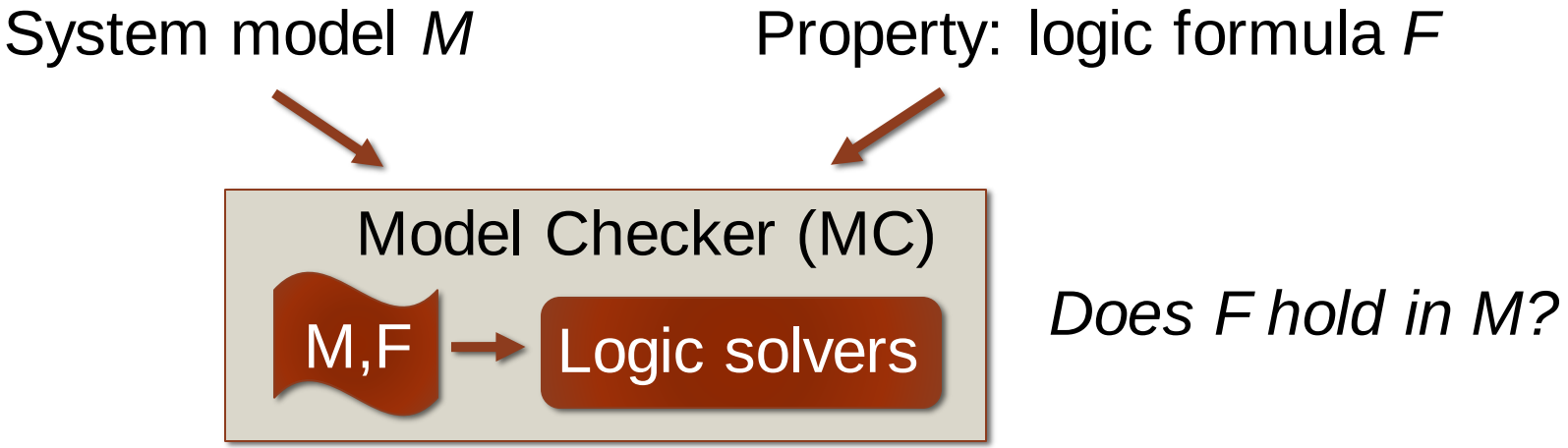
http://upscale.stanford.edu/

Pre-Silicon Verification in EDA



- Our focus: processor designs.
- Goal: verify HW description language (HDL) model.
  - Detect bugs early.
- Simulation, testing: labor-intensive, non-exhaustive.

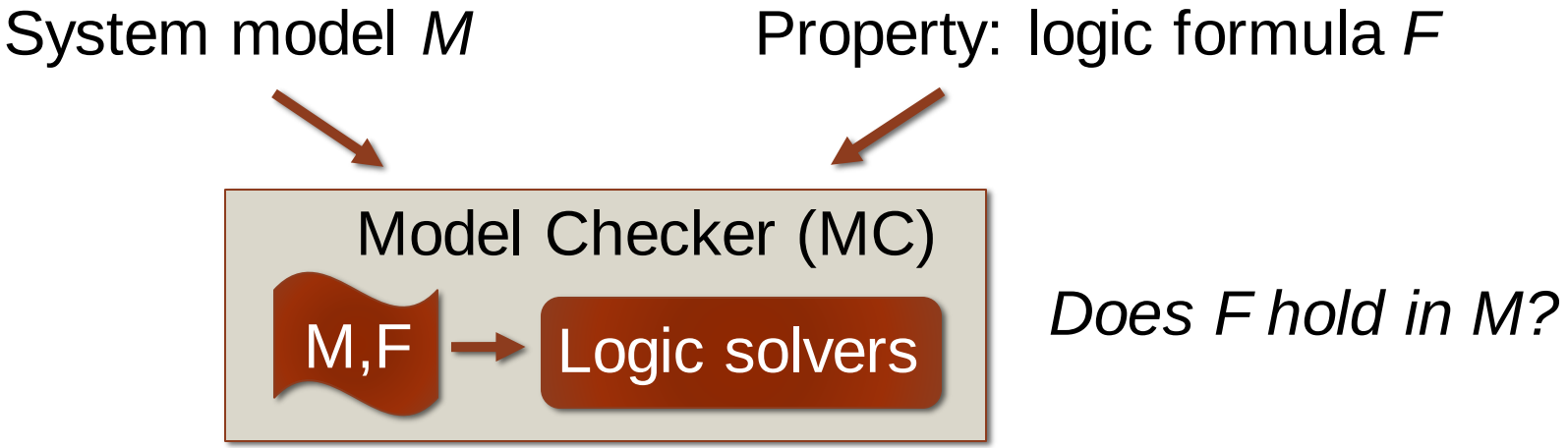
Model Checking and Formal Verification



Benefits:

- Formal guarantees, exhaustiveness.
- Progress in automated reasoning in various logics.

Model Checking and Formal Verification



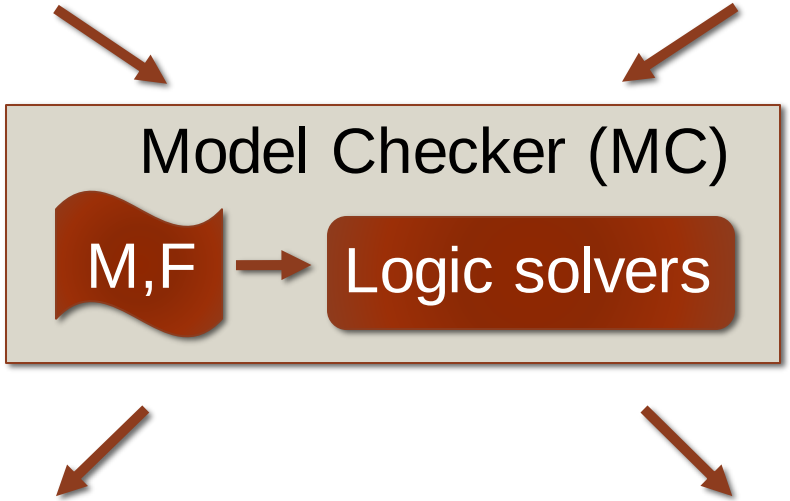
Challenges:

- Writing properties: expert knowledge, design-specific.
- Limited scaling to design sizes.

Symbolic Quick Error Detection (SQED)

Design (HDL) *M* + ISA

Design-independent property *F*

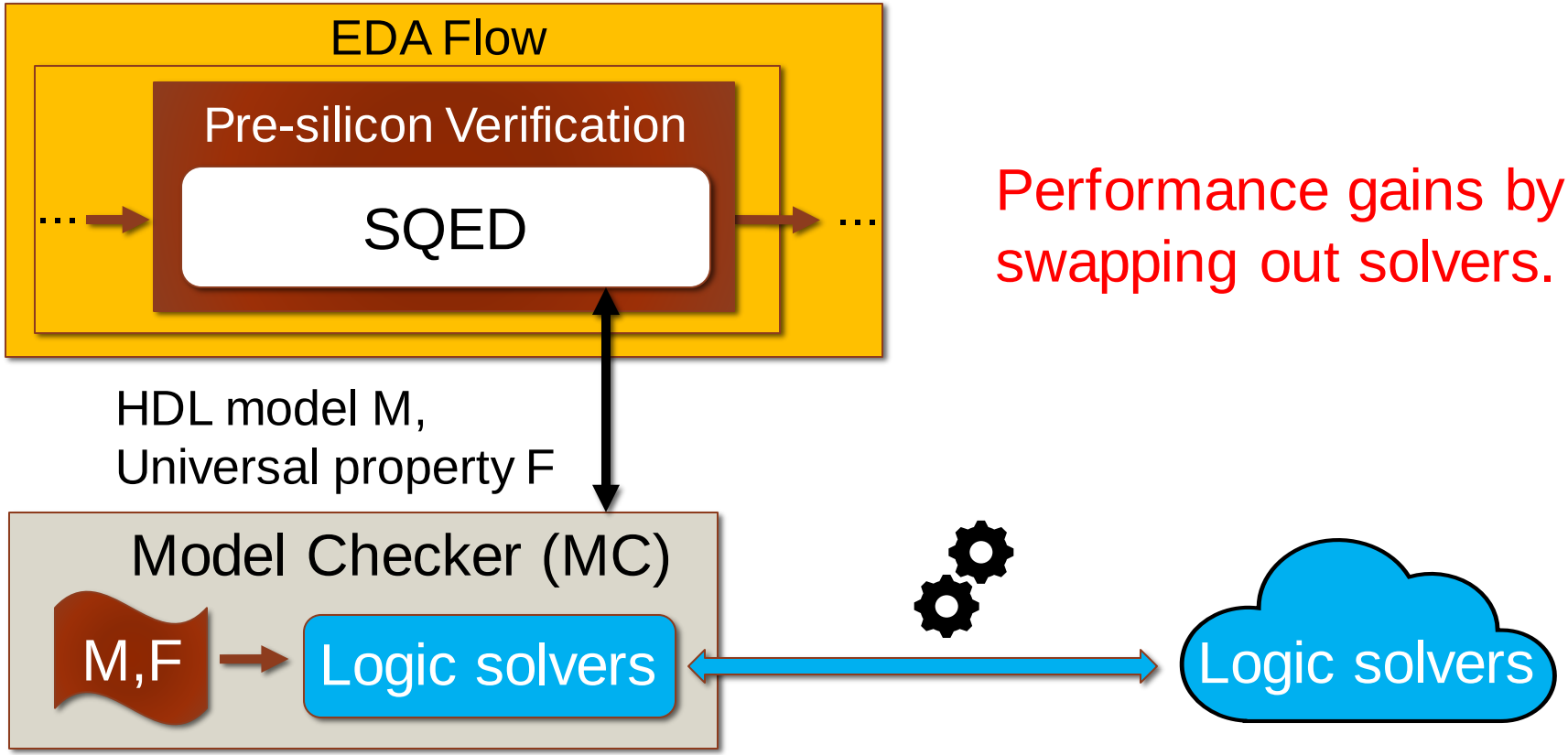


Does F hold in M?

Shortest bug trace

Correctness guarantee

SQED for Pre-Silicon Verification



Performance gains by swapping out solvers.

SQED: Industrial Strength

INFINEON case study: 16 automotive IP versions verified over 5 years

Thoroughness

All known bugs + more

Industry Flow SQED

60X Productivity

Industry Flow SQED

Industry flow:

(Constrained) random simulation, directed tests, formal.

SQED: Industrial Strength

INFINEON case study: 16 automotive IP versions verified over 5 years

Thoroughness

All known bugs + more

Industry Flow SQED

60X Productivity

Industry Flow SQED

Applications beyond processors:

Uncore components, accelerators, security,…

Unlocking the Power of Formal HW Verification

“Symbolic”:

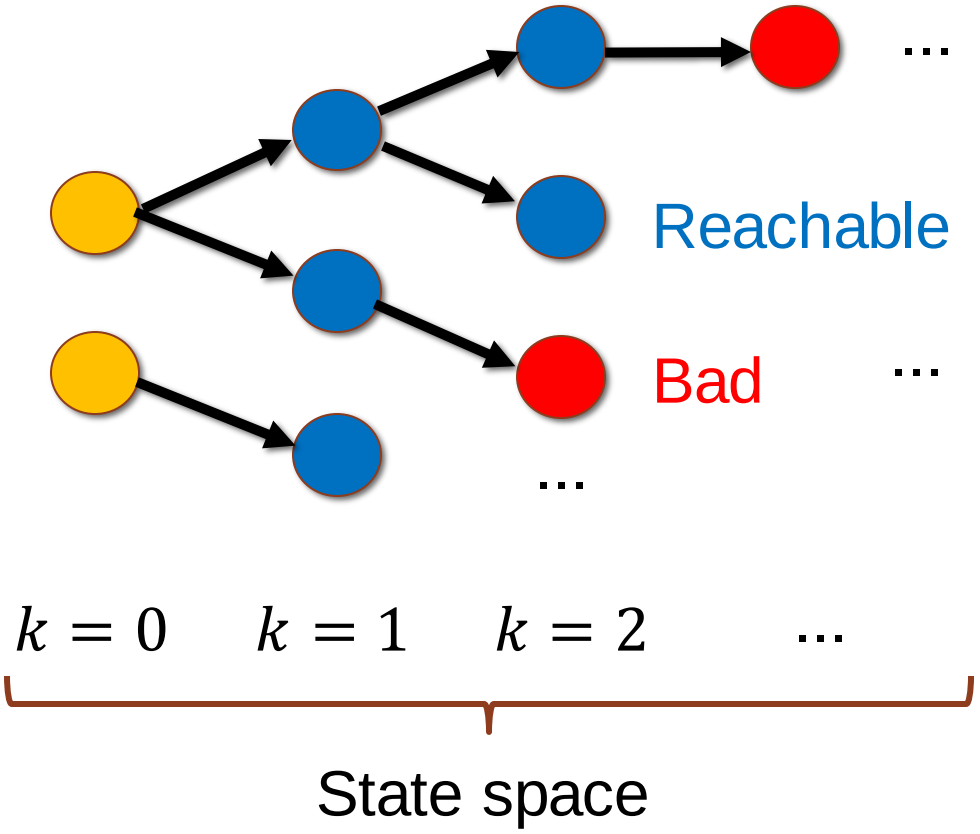
- Logic solving.
- Our CoSA model checker.
- Open-source tool chain.
- Performance ≈ industry.

“Quick Error Detection”:

- Exhaustiveness.
- Automation.
- New bugs (RISC-V): ≤ 100sec.
- Speed-up: logic solving.

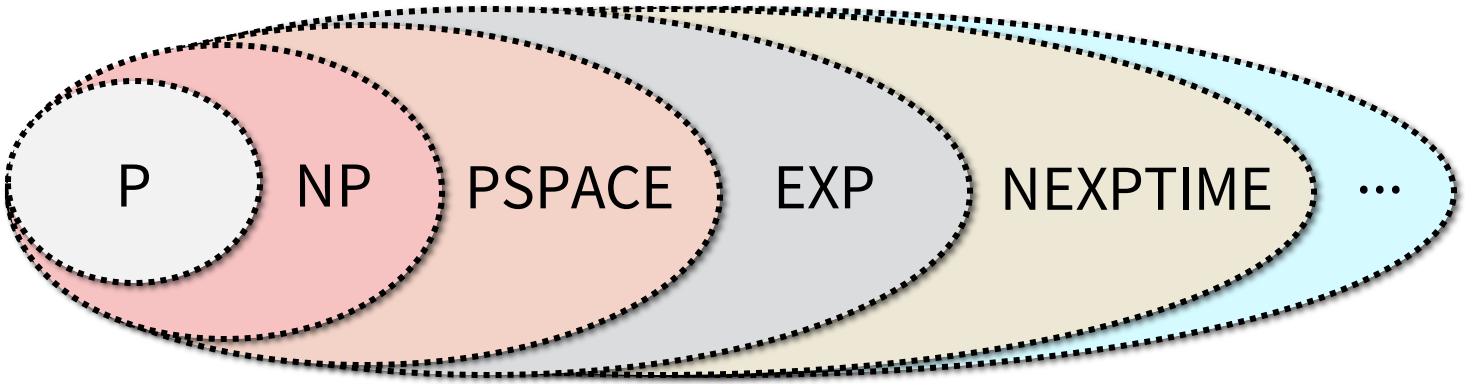
Bounded Model Checking (BMC)

- Symbolic breadth-first search for bad state (property violation).
- Model unrolled step by step: $k = 0$, $k = 1$, …
- Focus on bug hunting.

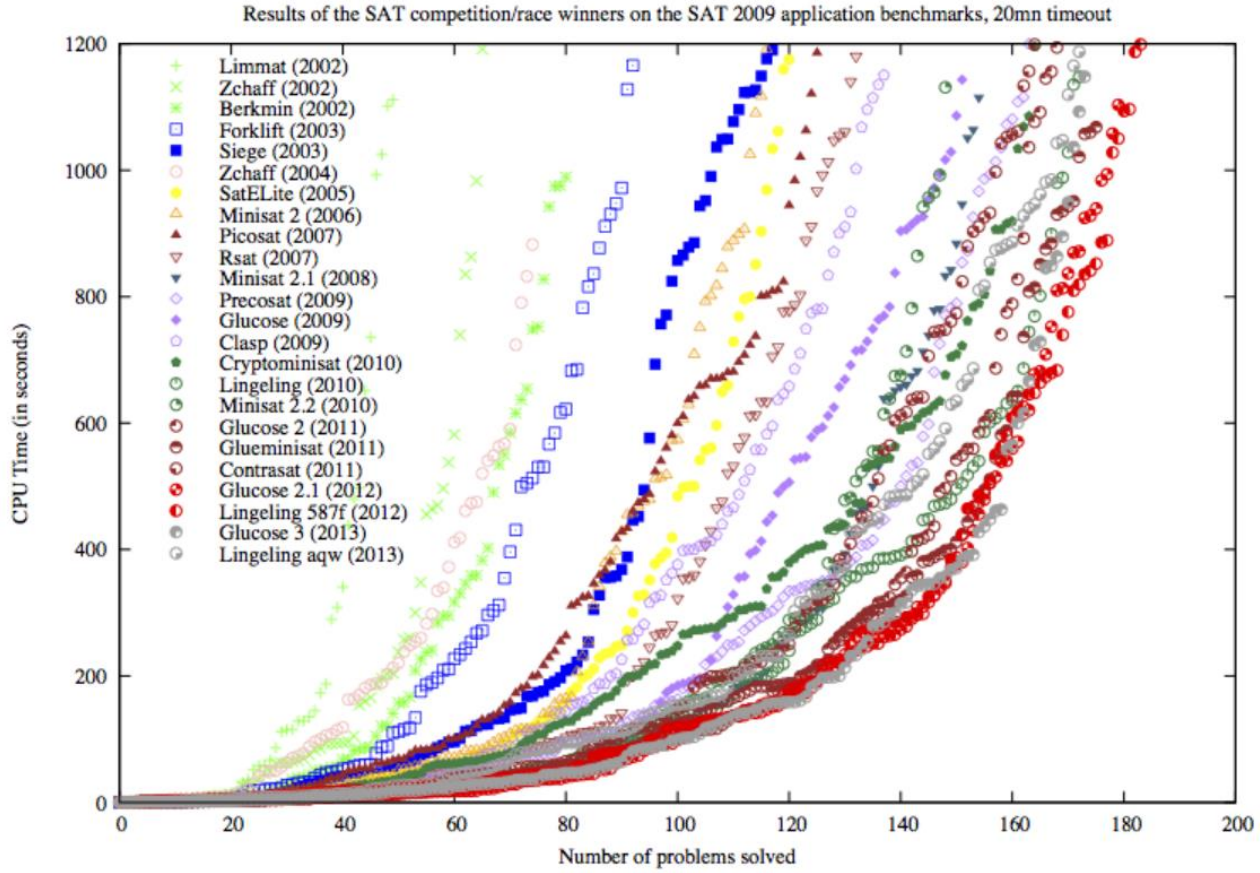


Satisfiability Solving

- Propositional (**SAT**), satisfiability modulo theories (**SMT**).
- Word-level properties for HW: bitvectors, arrays.
- Since late 1990s: "SAT revolution" [cf. Vardi, CACM'14].
- Solvers scale well on structured problems.

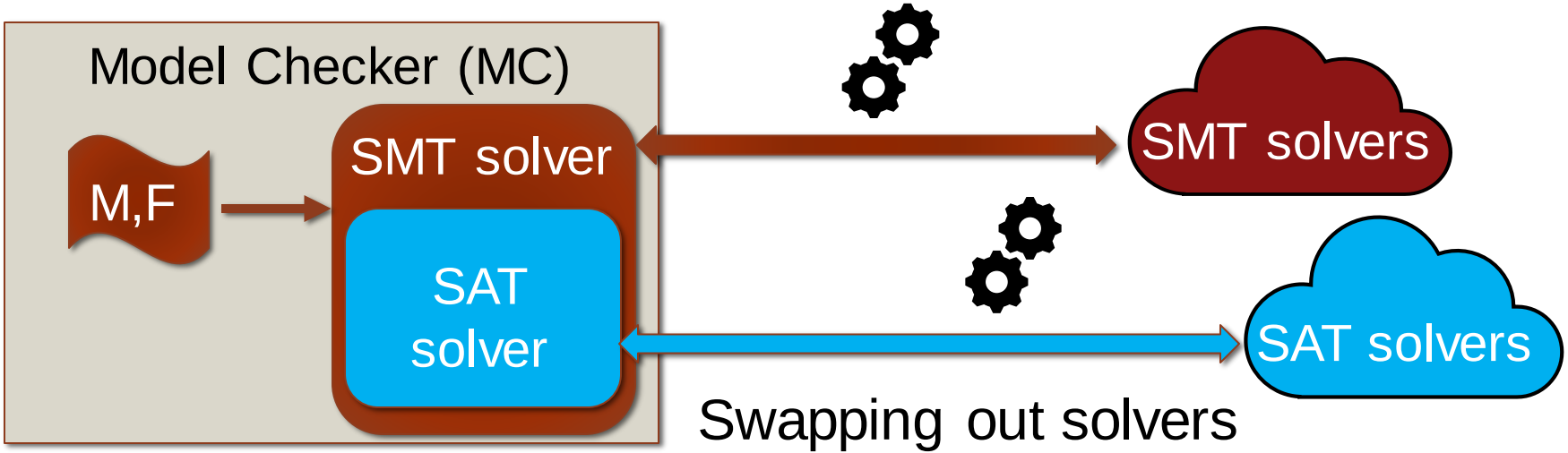


SAT Revolution



Plot due to Daniel Le Berre.

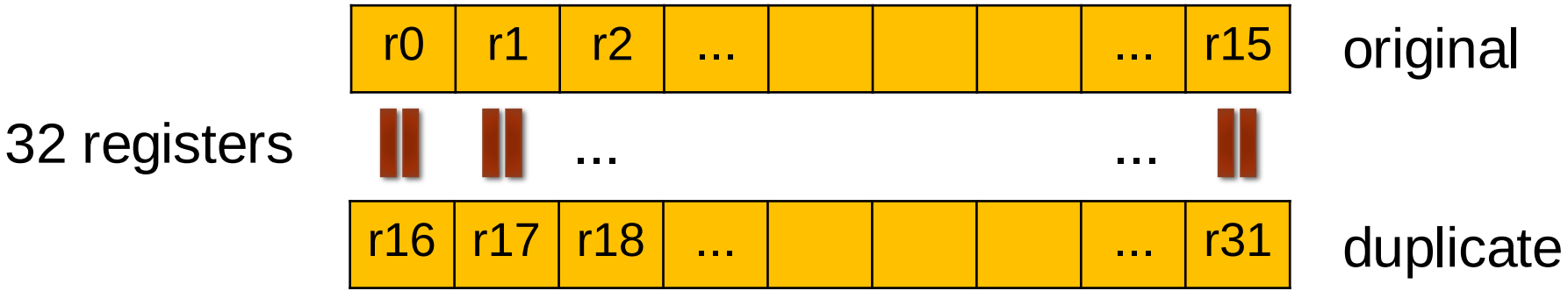
Interplay: BMC and SAT/SMT Solving



Solving BMC reachability as SAT/SMT problems:

$$I(s_0) \wedge T(s_0, s_1) \wedge \ldots \wedge T(s_{k-1}, s_k) \wedge B(s_k)$$

SQED Basic Idea: Self-Consistency

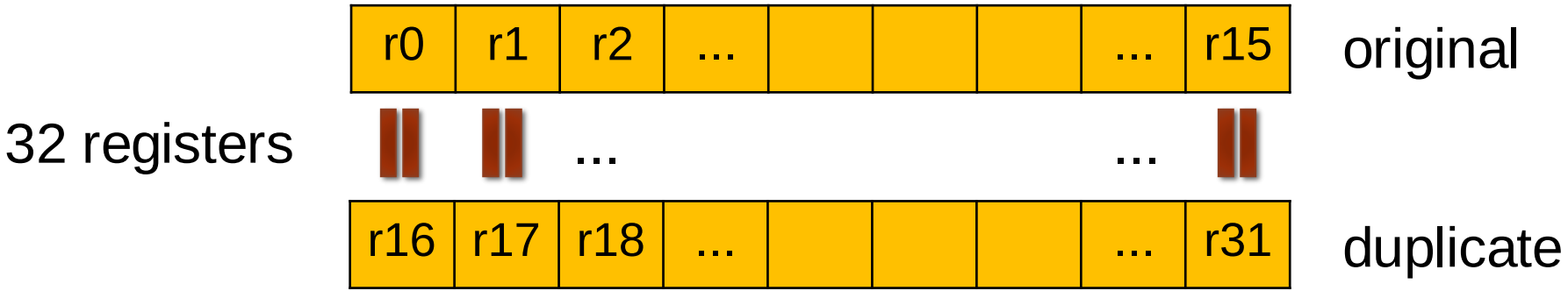


- Divide register/memory space in two halves, e.g.:

$$r(i) \rightarrow r(i+16) \qquad i := 0, \ldots, 15$$

- QED-consistent state: $\bigwedge REGS[i] = REGS[i+16]$

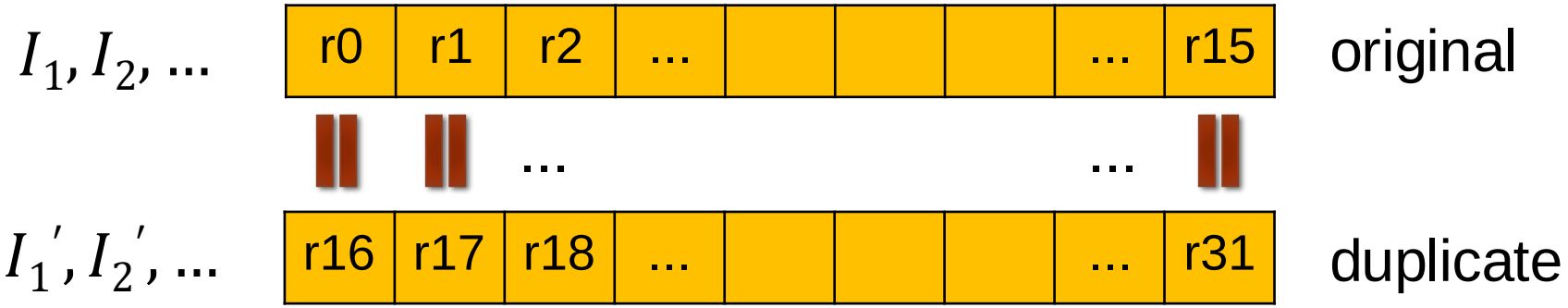
Instruction Duplication



Duplicate Instruction:
Same semantics, using only duplicate registers.

INSTR $r(i)\ r(j)\ r(k)$ → INSTR $r(i+16)\ r(j+16)\ r(k+16)$

QED Consistency: Universal BMC Property



Interleaving: $I_1, I_1', I_2', I_2, \ldots$

- Execute interleaving of original and duplicate instructions.
- Property: QED-consistency preserved by interleaving.

Real Life Bug Example: RIDECORE (RISC-V)

Original:

XOR r5, r0, 3547

MULH r1, r5, r5

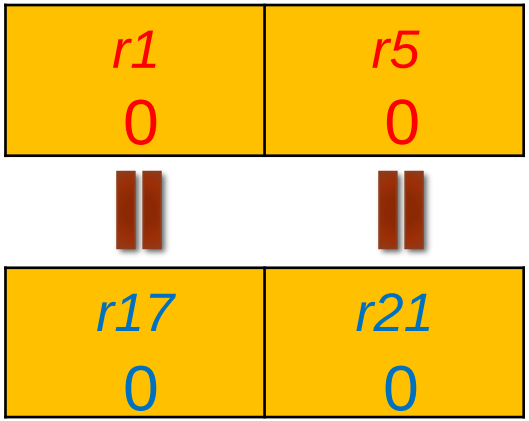
Duplicate:

XOR r21, r16, 3547

MULH r17, r21, r21

Bug in reservation station:

Back-to-back MULHs corrupt result.



Real Life Bug Example: RIDECORE (RISC-V)

Original:

XOR r5, r0, 3547

MULH r1, r5, r5

Duplicate:

XOR r21, r16, 3547

MULH r17, r21, r21

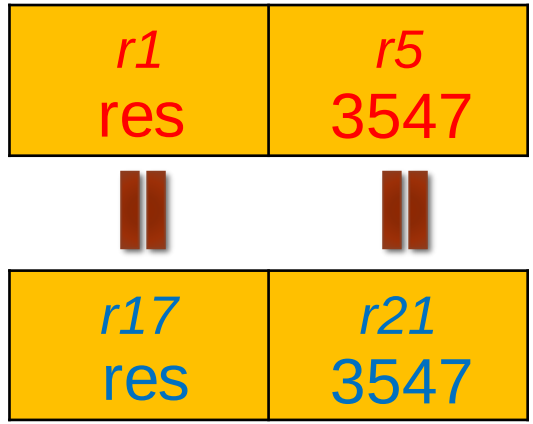
Bad interleaving: bug undetected

XOR r5, r0, 3547

MULH r1, r5, r5

XOR r21, r16, 3547

MULH r17, r21, r21

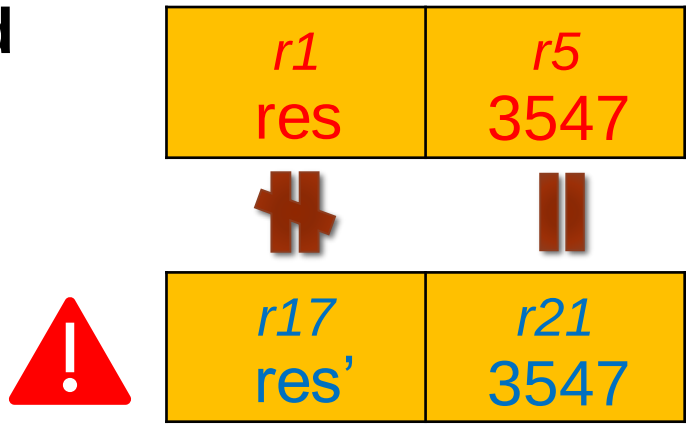


Real Life Bug Example: RIDECORE (RISC-V)

Original:
XOR r5, r0, 3547
MULH r1, r5, r5

Duplicate:
XOR r21, r16, 3547
MULH r17, r21, r21

Good interleaving: bug detected
XOR r5, r0, 3547
XOR r21, r16, 3547
MULH r1, r5, r5
MULH r17, r21, r21



SQED: Big Picture

Design (HDL) + ISA

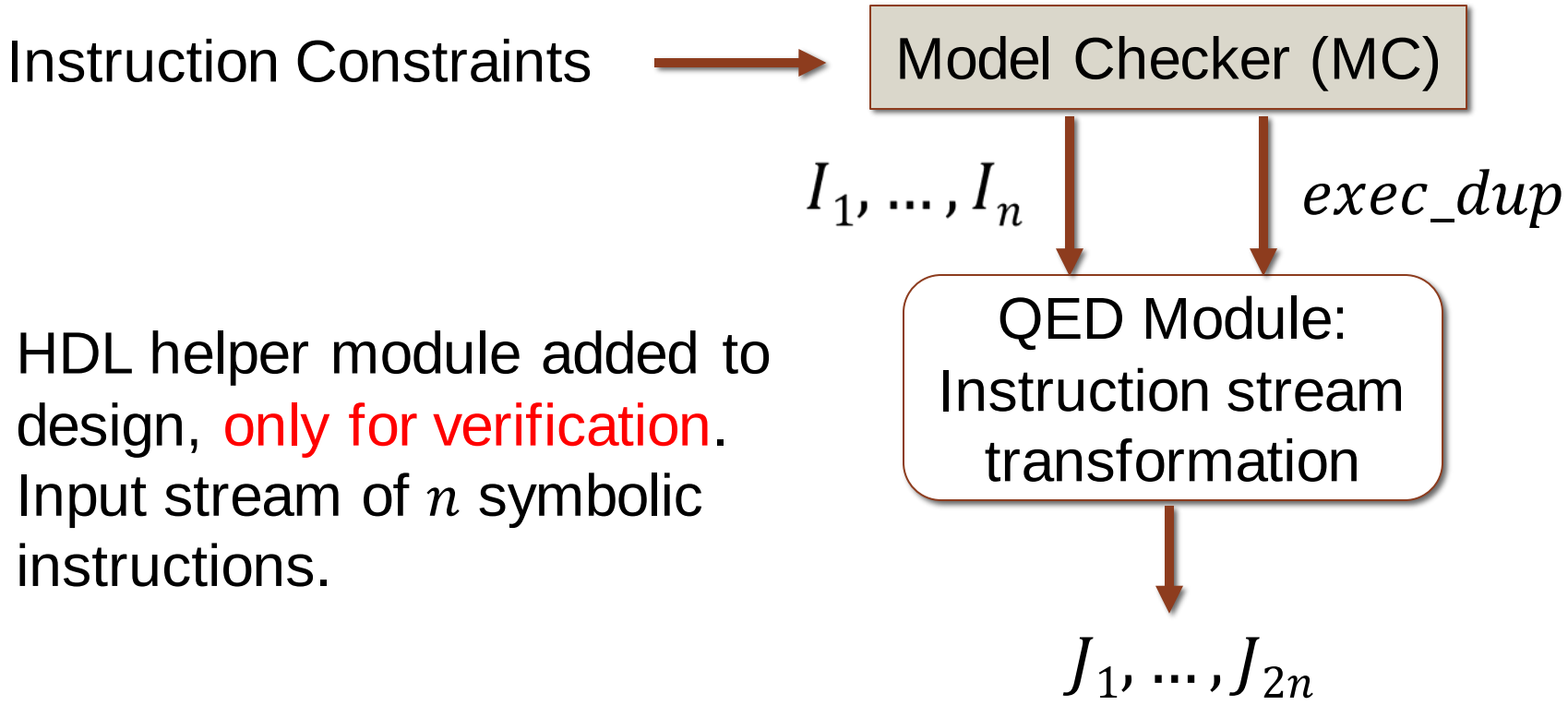
QED consistency:

$$\bigwedge REGS[i] = REGS[i+16]$$

Model Checker (MC): Explore **all** possible instruction sequences of increasing length k.

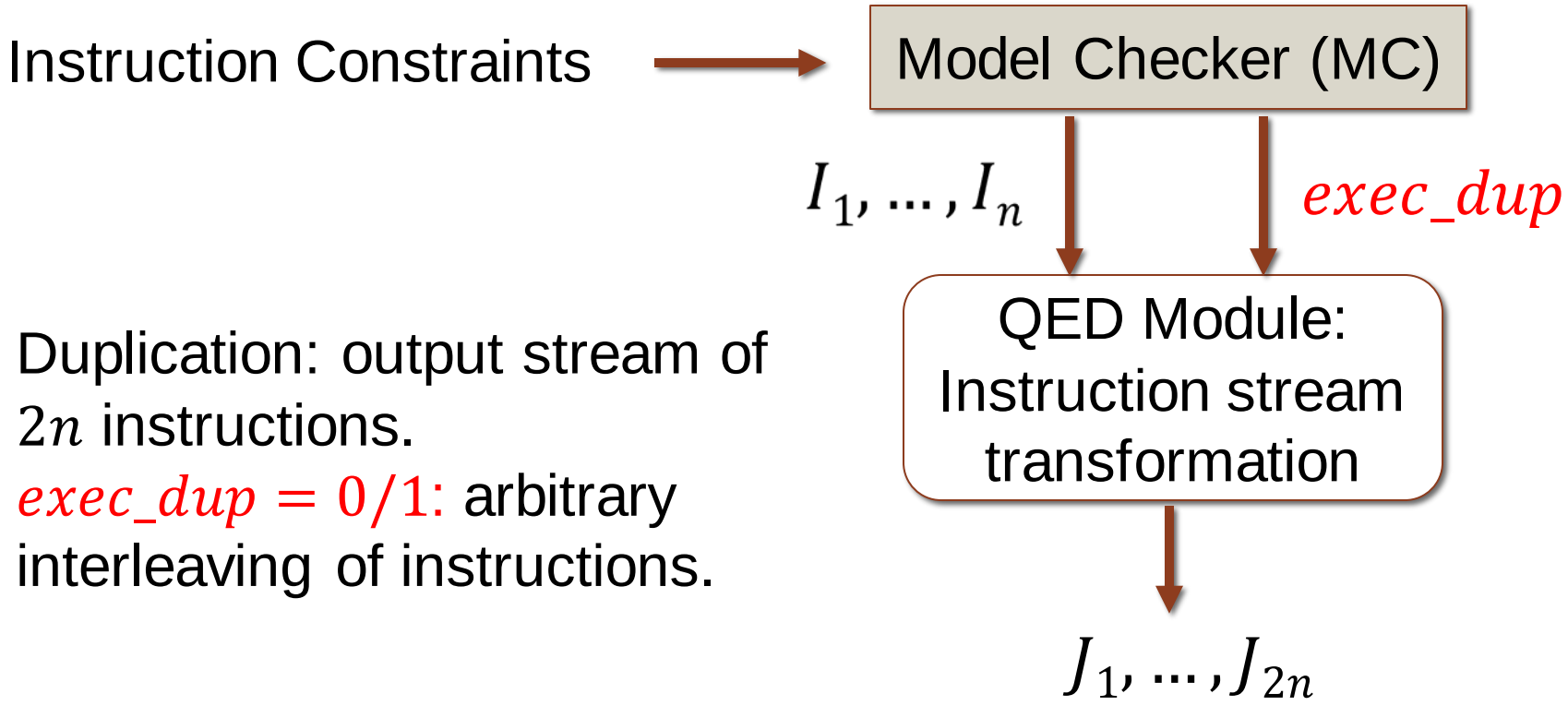
$k=0$	$k=1$	$k=2$	…
	$INST_1$	$INST_1$	
		$INST_2$	

SQED in Practice: QED Module



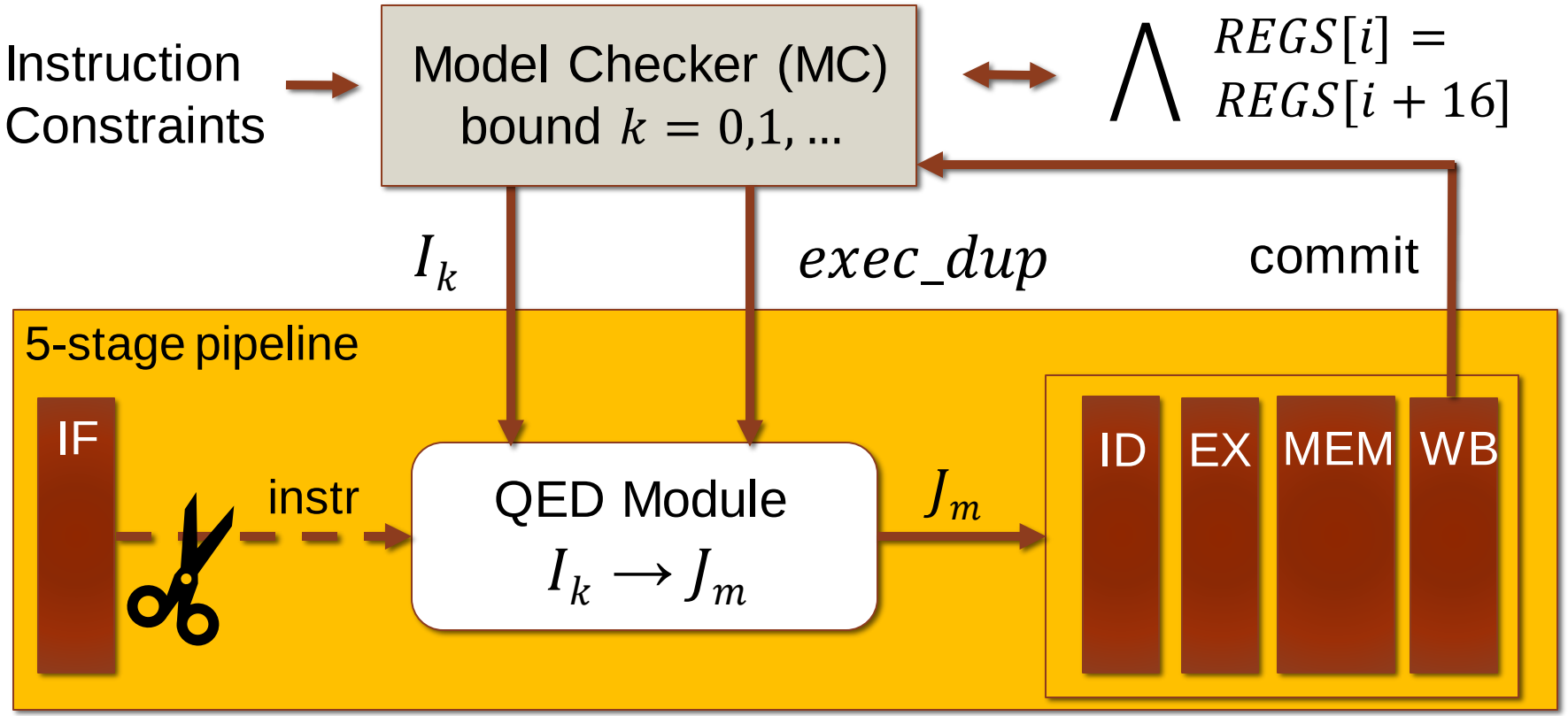
- HDL helper module added to design, only for verification.
- Input stream of n symbolic instructions.

SQED in Practice: QED Module

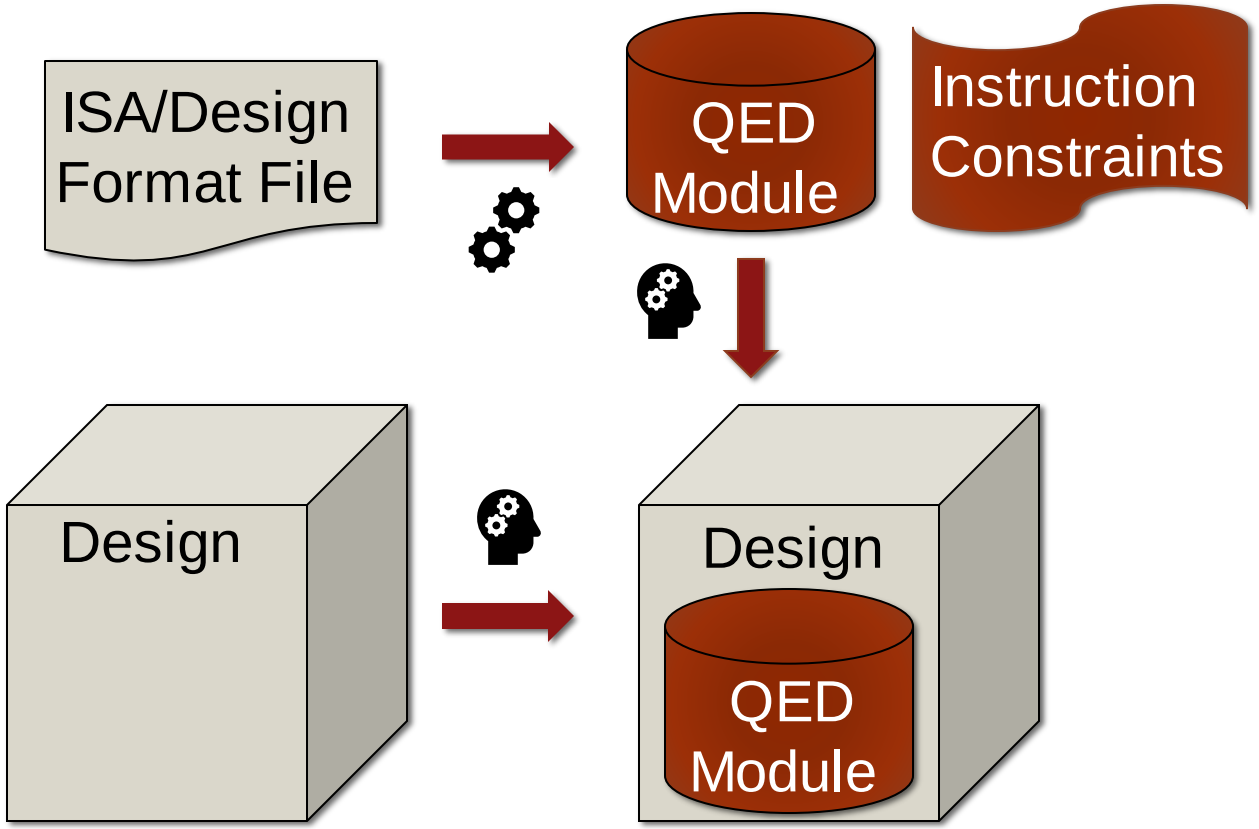


- Duplication: output stream of $2n$ instructions.
- $exec\_dup = 0/1$: arbitrary interleaving of instructions.

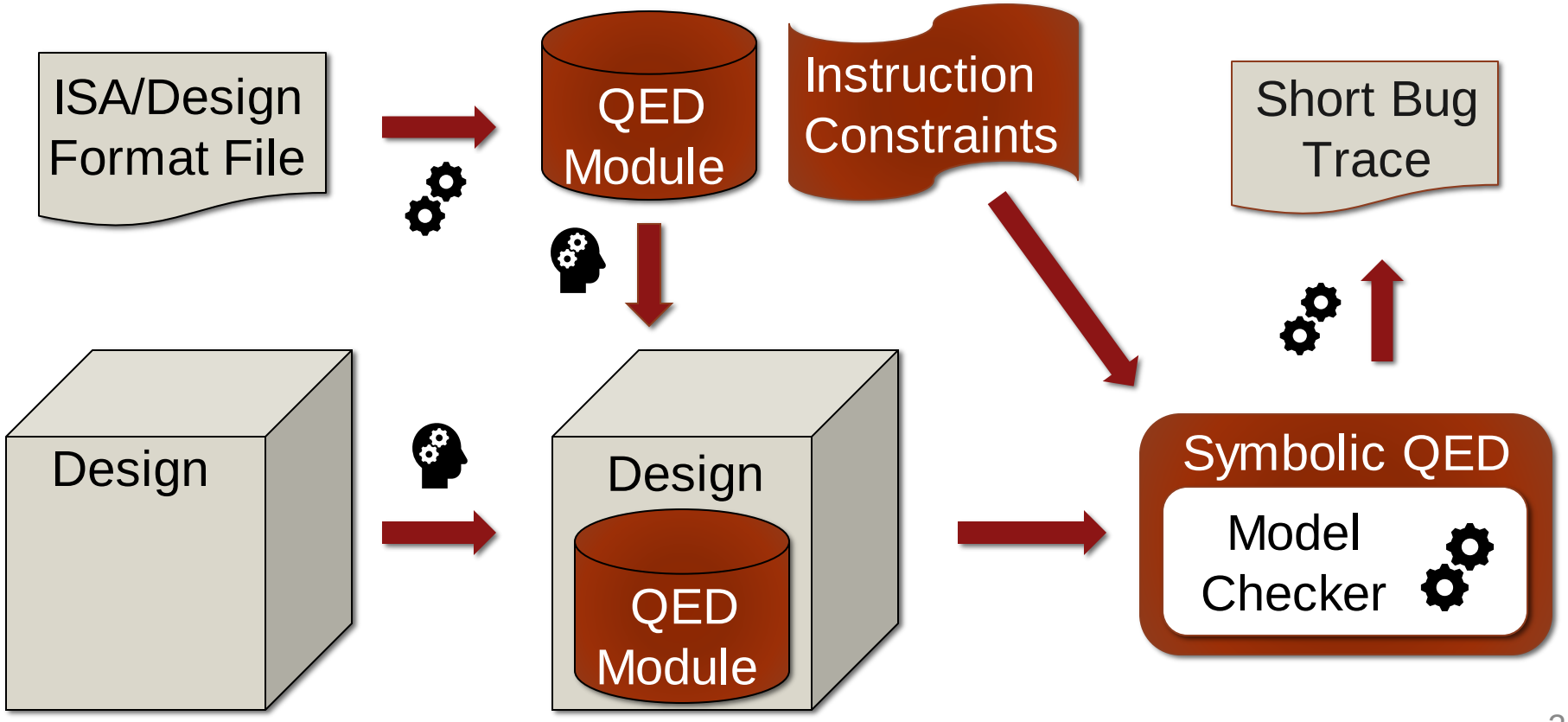
QED Module Integration



SQED: Generator-Based Approach

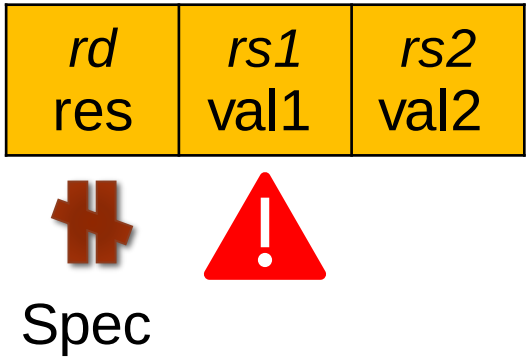


SQED: Generator-Based Approach



Single-Instruction (SI) Checking

INSTR rd, rs1, rs2



- SI bug: instruction fails in every system state/context.
  - Not detectable by SQED.
- Model checker searches over symbolic inputs.
- Our open-source approach: CoSA.
- Standard industry approaches.

Experimental Results

- SQED demo platform: model checker CoSA.
  - https://github.com/cristian-mattarei/CoSA
- New bugs found:
  - RISC-V designs: RIDECORE, Vscale (corner cases).
  - Crucial: QED module interleaves instructions.
- RIDECORE case study:
  - Impact of more efficient SAT/SMT solving.
  - Up to 7X speedup in model checking.

RIDECORE: New Multiplier Bug

Bug Activation	Bug Effect	Time (s)
Executing back-to-back MULHs.	Result corrupted.	93

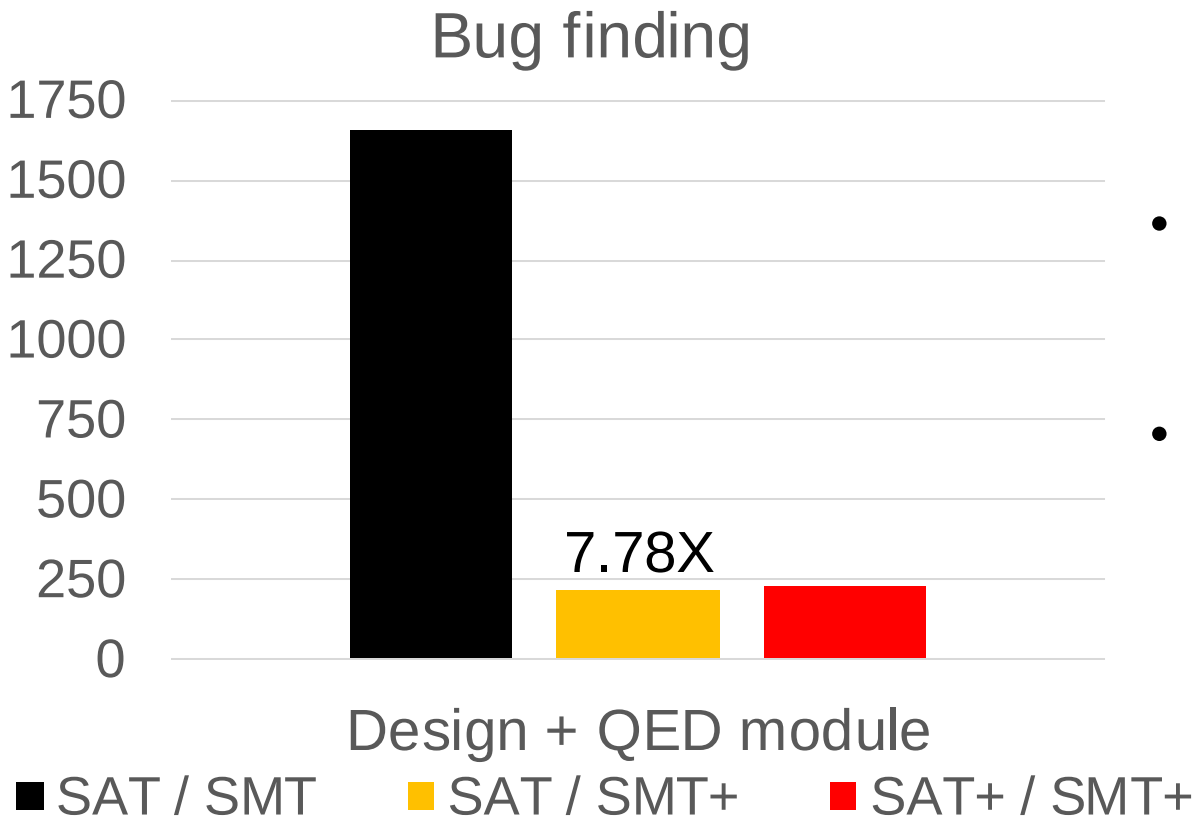
- Using “Questa” formal tool (Mentor Graphics).
- Bugs in reservation station (RS-m) of multiplier.
- Using CoSA (variants): 86s - 226s.

Vscale: New Bugs in Interrupt Logic

Bug Activation	Bug Effect	Time (s)
“1” written to specific bit positions in CSR MIP.	Repeated interrupts, MTIME CMP register corrupted.	2
Lower two bits of CSR MSTATUS set to 01/10.	Unspecified privilege level entered, MEPC corrupted.	33

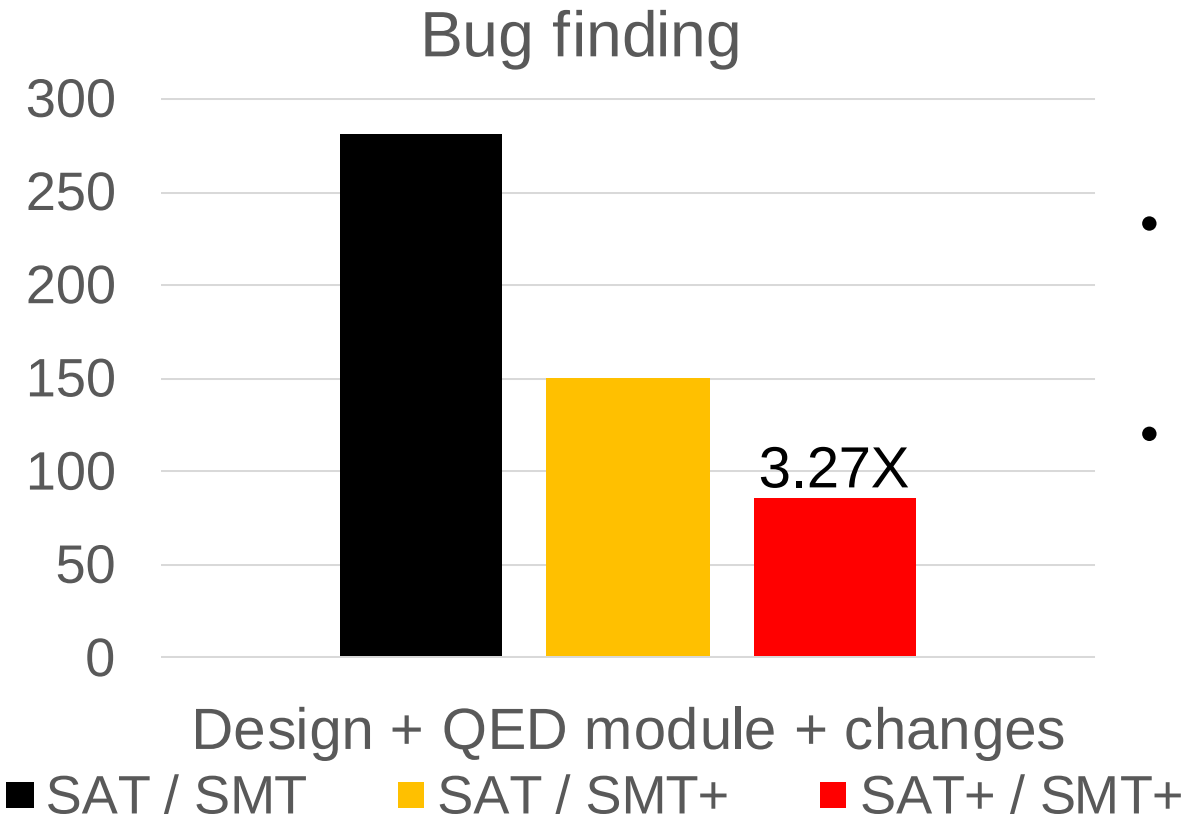
- Using “Questa” formal tool (Mentor Graphics).
- Bugs in implementation of RISC-V privileged ISA.

RIDECORE: Impact of SAT/SMT Solving



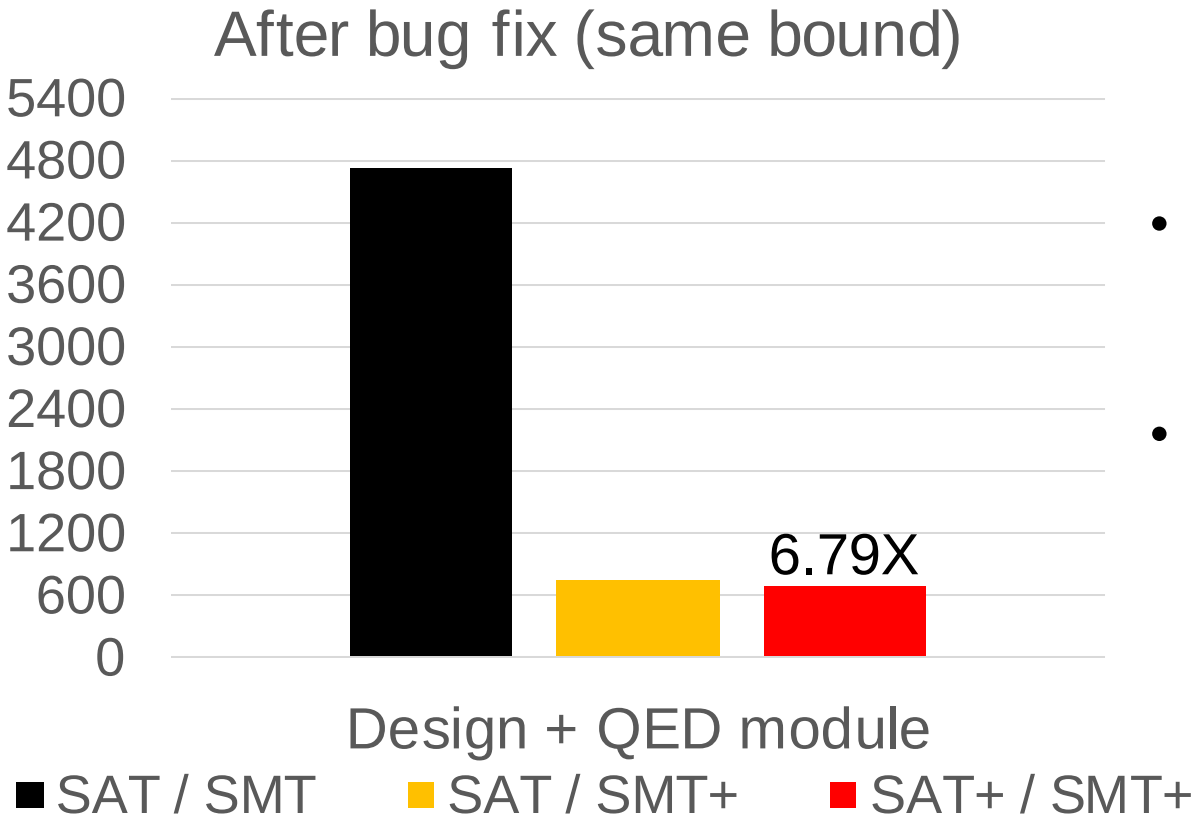
- Boolector:
  - “SMT”: basic
  - “SMT+”: improved
- SAT solvers:
  - “SAT”: Lingeling
  - “SAT+”: CaDiCaL

RIDECORE: Impact of SAT/SMT Solving



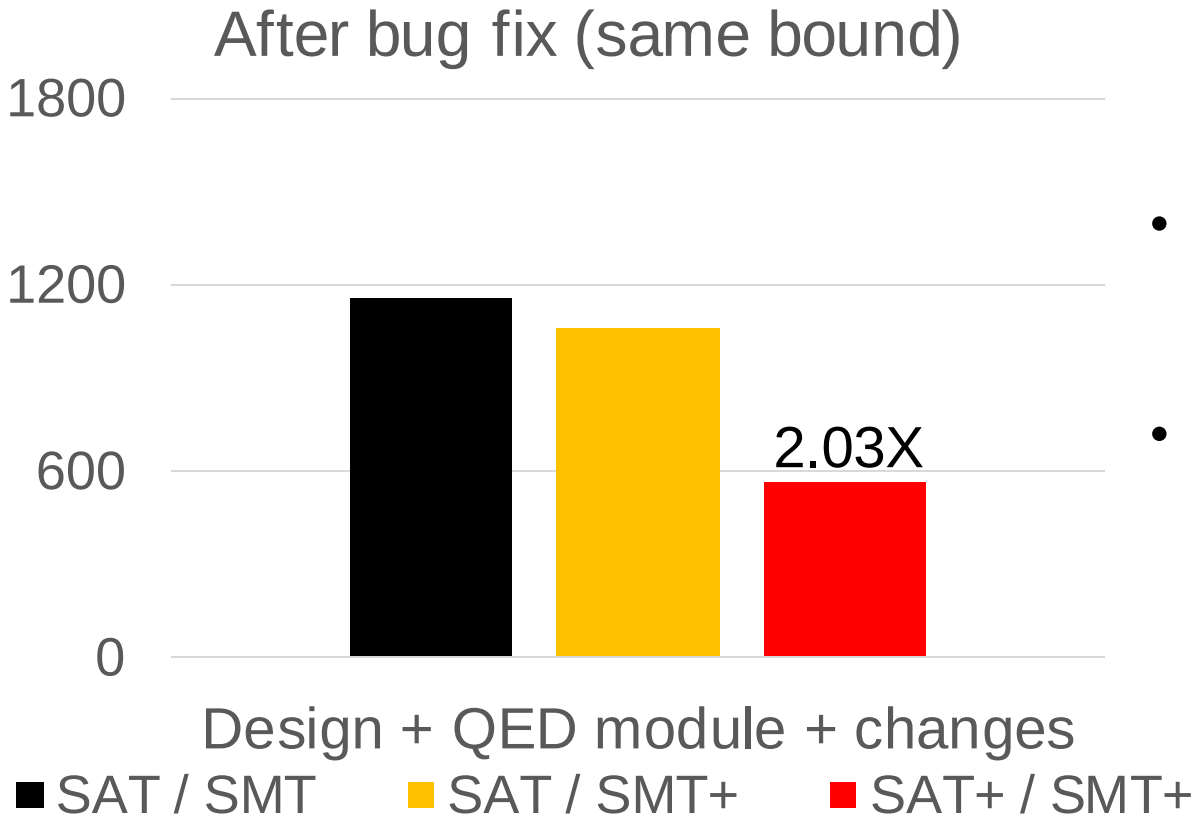
- Boolector:
  - “SMT”: basic
  - “SMT+”: improved
- SAT solvers:
  - “SAT”: Lingeling
  - “SAT+”: CaDiCaL

RIDECORE: Impact of SAT/SMT Solving



- Boolector:
  - “SMT”: basic
  - “SMT+”: improved
- SAT solvers:
  - “SAT”: Lingeling
  - “SAT+”: CaDiCaL

RIDECORE: Impact of SAT/SMT Solving



- Boolector:
  - “SMT”: basic
  - “SMT+”: improved
- SAT solvers:
  - “SAT”: Lingeling
  - “SAT+”: CaDiCaL

Summary: Symbolic Quick Error Detection

- Industrial-strength model checking technique.
- Automatic generation of parameterized QED module.
- Open-source tool chain, on par with industry tools.
- Future work: further increasing automation.

Tools and Demos: http://upscale.stanford.edu/

Backup Slides

[BACKUP] QED Module: Instruction Constraints

```
INPUT:[31:0] instruction, clock;
assign opcode = instruction[6:0];
assign funct3 = instruction[14:12];
assign funct7 = instruction[31:25];
assign ADD =
  (funct3 == 3'b000) && (opcode == 7'b0110011) &&
  (funct7 == 7'b0000000);
// add opcode constraints for all instructions
...
always @(posedge clock)
begin
  assume property(ADD ||...||...);
end
```

Opcode constraint example: 32-bit register-type ADD of a RISC-V ISA.

[BACKUP] SQED: Generator-Based Approach

```
SECTIONS = ISA QEDCONSTRAINTS REGISTERS ...
_ISA
num_registers = 32
instruction_length = 32

_QEDCONSTRAINTS
half_registers = 1

_R
ADD
funct3 = 000
funct7 = 0000000
opcode = 0110011
```

```
_BITFIELDS
funct7 = 31 25
funct3 = 14 12
rd = 11 7
rs1 = 19 15
rs2 = 24 20
opcode = 6 0
...
```

RISC-V format file example (excerpt).

[BACKUP] Single-Instruction Checking

```
assumptions: reset = 1;
             clkcnt = 0 | clkcnt >= 2 -> instr = NOP;
             clkcnt = 1 -> instr = ADD & rd != 0;
property:    clkcnt = 7 -> val1 + val2 = regs[rd_copy];
```

RIDECORE: example of ADD-property.

- Checking every instruction individually.
- Correctness property: instruction semantics.
- Model checker searchers over all possible inputs.

[BACKUP] Single-Instruction Checking

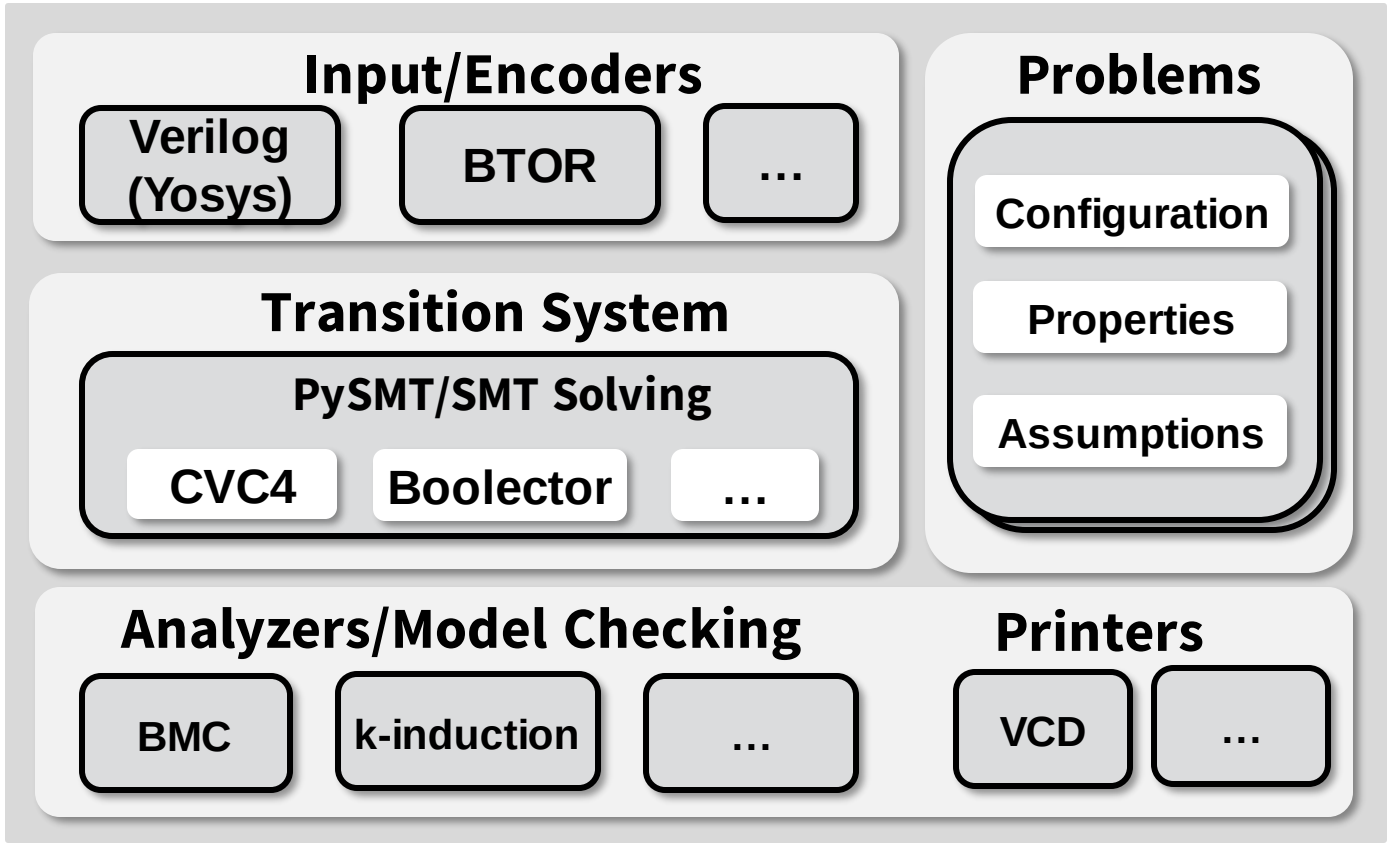
```
assumptions: reset = 1;
             stcnt = 0 | stcnt >= 2 -> instr = NOP;
             stcnt = 1 -> instr = ADD & rd != 0;
property:    stcnt = 7 -> val1 + val2 = regs[rd_copy];
```

RIDECORE: example of ADD-property

```
next(instr)    = stcnt = 0 | stcnt >= 2 ? NOP : instr;
next(stcnt)    = stcnt++;
next(val1)     = stcnt = 1 ? regs[rs1] : val1;
next(val2)     = stcnt = 1 ? regs[rs2] : val2;
next(rd_copy)  = stcnt = 1 ? rd : rd_copy;
```

RIDECORE: helper state transition system

[BACKUP] CoSA Model Checker



[BACKUP] SI Checking: RIDECORE and Vscale

Instructions checked	Time/Check (s)	
	RIDECORE	Vscale
All instructions except MUL	40	3
All instructions with restricted MUL	40	3
MUL with injected bug	40	14

- Checking MUL is known to be hard.
- Future work: apply specialized approaches for MUL.

[BACKUP] RIDECORE: New Multiplier Bugs

Bug Activation	Bug Effect	Time (s)
MULH assigned to last vacant (buggy) RS-m entry.	$1^{st}/2^{nd}$ source operand corrupted.	63/69
Same as above, but with MULHU.	Result of MULHU instruction corrupted.	93

- Using “Questa” formal tool (Mentor Graphics).
- Bugs in reservation station (RS-m) of multiplier.

[BACKUP] RIDECORE: SAT/SMT Solving

	BMC depth	Boolector (impr.) + CaDiCaL	Boolector (impr.) + Lingeling	Boolector (base) + Lingeling
Setup	**k**	**T(b)**	**T(b)**	**T(b)**
A	23	226	213	1658

T(b): time to find bug at BMC depth k

- Setup A: only wiring up QED module.
- Speed-up (vs. base): 7.78.

[BACKUP] RIDECORE: SAT/SMT Solving

	BMC depth	Boolector (impr.) + CaDiCaL	Boolector (impr.) + Lingeling	Boolector (base) + Lingeling
Setup	**k**	**T(b)**	**T(b)**	**T(b)**
A	23	226	213	1658
B	13	98	127	257

T(b): time to find bug at BMC depth k

- Setup B: + only pos-edge clock behavior.
- Speed-up (vs. base): 2.62.

[BACKUP] RIDECORE: SAT/SMT Solving

	BMC depth	Boolector (impr.) + CaDiCaL	Boolector (impr.) + Lingeling	Boolector (base) + Lingeling
Setup	**k**	**T(b)**	**T(b)**	**T(b)**
A	23	226	213	1658
B	13	98	127	257
C	13	86	150	282

T(b): time to find bug at BMC depth k

- Setup C: + only pos-edge clock behavior, reduced data memory.
- Speed-up (vs. base): 3.27.

[BACKUP] RIDECORE: SAT/SMT Solving

	BMC depth	Boolector (impr.) + CaDiCaL	Boolector (impr.) + Lingeling	Boolector (base) + Lingeling
Setup	**k**	**T(b)**	**T(b)**	**T(b)**
A	23	697	746	4739

T(c): time to prove correctness up to BMC depth k after bug fix

- Setup A: only wiring up QED module.
- Speed-up (vs. base): 6.79.

[BACKUP] RIDECORE: SAT/SMT Solving

	BMC depth	Boolector (impr.) + CaDiCaL	Boolector (impr.) + Lingeling	Boolector (base) + Lingeling
Setup	**k**	**T(b)**	**T(b)**	**T(b)**
A	23	697	746	4739
B	13	623	634	1771

T(c): time to prove correctness up to BMC depth k after bug fix

- Setup B: + only pos-edge clock behavior.
- Speed-up (vs. base): 2.84.

[BACKUP] RIDECORE: SAT/SMT Solving

	BMC depth	Boolector (impr.) + CaDiCaL	Boolector (impr.) + Lingeling	Boolector (base) + Lingeling
Setup	**k**	**T(b)**	**T(b)**	**T(b)**
A	23	697	746	4739
B	13	623	634	1771
C	13	568	1062	1156

T(c): time to prove correctness up to BMC depth k after bug fix

- Setup C: + only pos-edge clock behavior, reduced data memory.
- Speed-up (vs. base): 2.03.