Verifying the Security of Microarchitectures based on Hardware-Software Contracts

Jan Reineke @ Saarland University

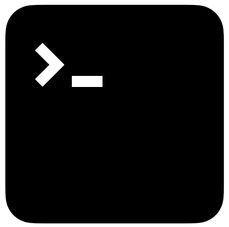
Joint work with
Zilong Wang, Marco Guarnieri @ IMDEA Software, Madrid
Klaus v. Gleissenthall @ VU Amsterdam
Gideon Mohr, Valentin Touzeau @ Saarland University

This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 101020415)"

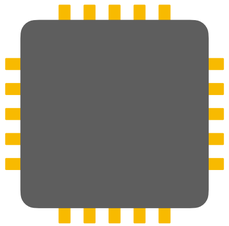
An Introduction to HW/SW Contracts

ISA: Benefits

High-level language

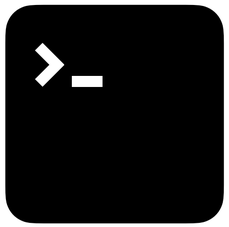


Microarchitecture



ISA: Benefits

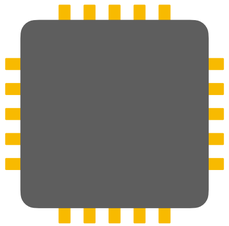
High-level language



Can program **independently** of microarchitecture

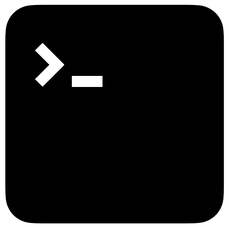
Instruction set architecture (ISA)

Microarchitecture



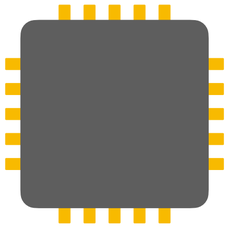
ISA: Benefits

High-level language



Instruction set architecture (ISA)

Microarchitecture

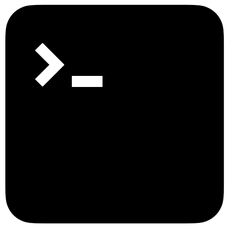


Can program **independently** of microarchitecture

Can implement **arbitrary optimizations** as long as ISA semantics are obeyed

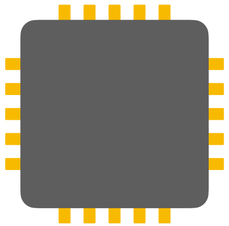
Inadequacy of the ISA: Side channels

High-level language



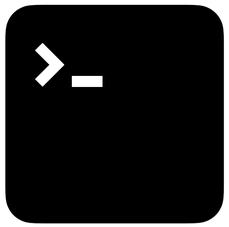
Instruction set architecture (ISA)

Microarchitecture



Inadequacy of the ISA: Side channels

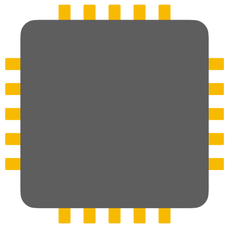
High-level language



Instruction set architecture (ISA)

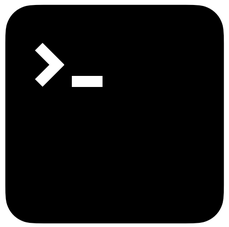
No guarantees about side channels

Microarchitecture



Inadequacy of the ISA: Side channels

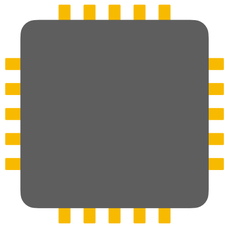
High-level language



Instruction set architecture (ISA)

No guarantees about side channels

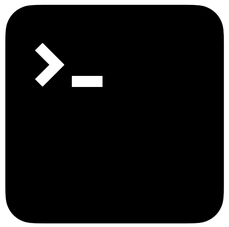
Microarchitecture



Can implement arbitrary **insecure** optimizations as long as ISA is implemented correctly

Inadequacy of the ISA: Side channels

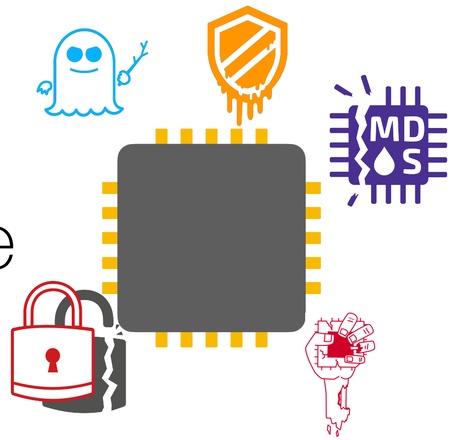
High-level language



Instruction set architecture (ISA)

No guarantees about side channels

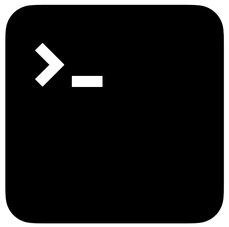
Microarchitecture



Can implement arbitrary **insecure** optimizations as long as ISA is implemented correctly

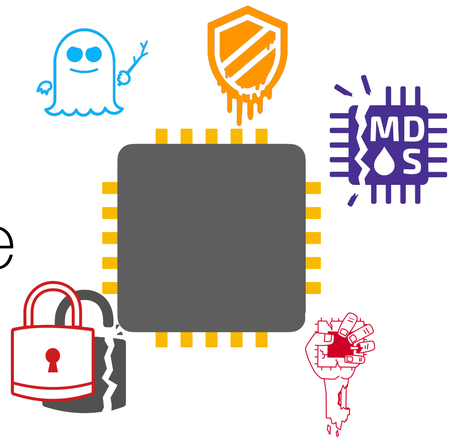
Inadequacy of the ISA: Side channels

High-level language



Instruction set architecture (ISA)

Microarchitecture



Impossible to program securely
cryptographic algorithms?
sandboxing untrusted code?

No guarantees about side channels

Can implement arbitrary **insecure** optimizations as long as ISA is implemented correctly

A way forward: HW/SW security contracts

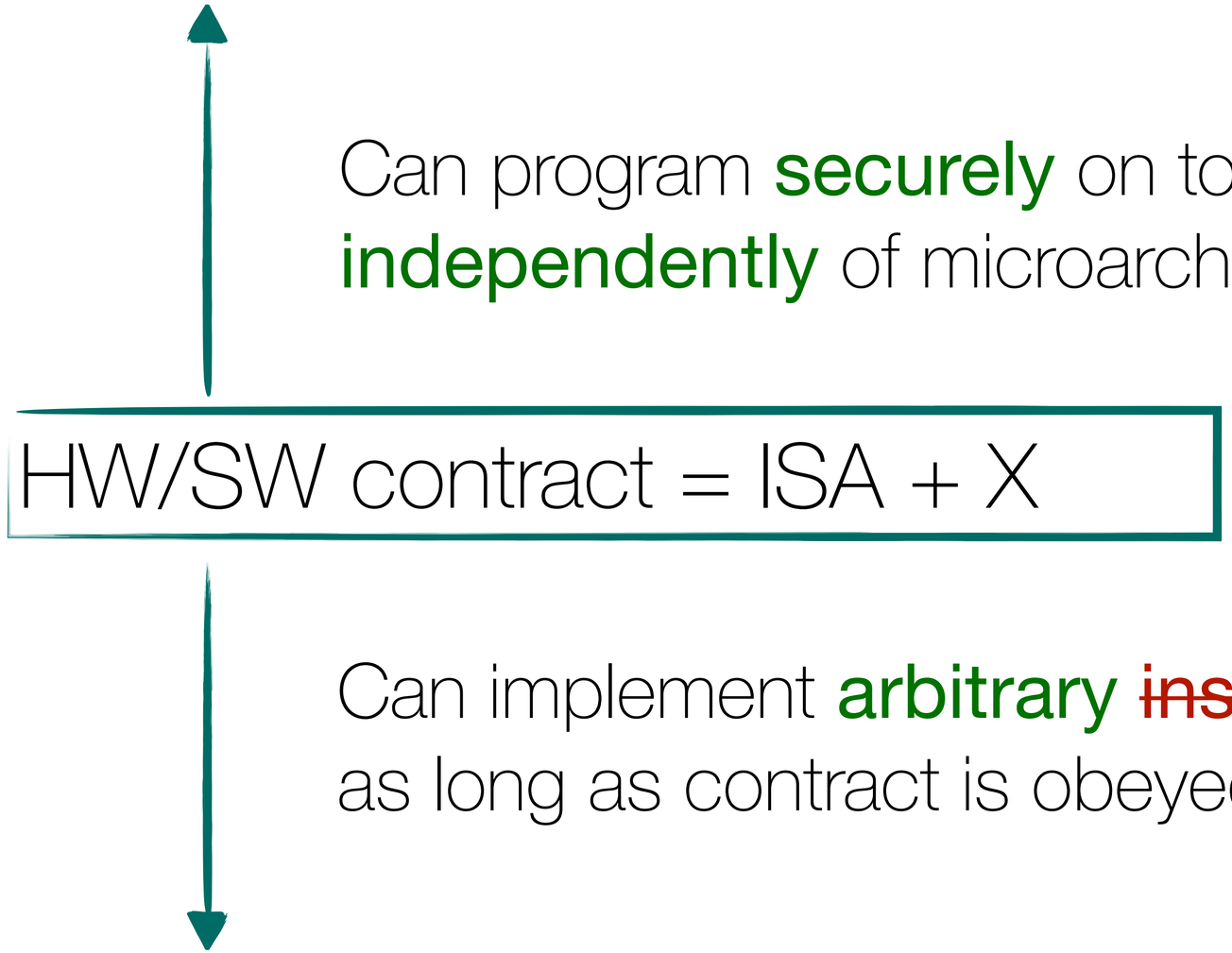
HW/SW contract = ISA + X

Succinctly captures
possible information leakage
in a mechanism-independent way

A way forward: HW/SW security contracts

Succinctly captures
possible information leakage
in a mechanism-independent way

A way forward: HW/SW security contracts



Succinctly captures
possible information leakage
in a mechanism-independent way

HW/SW contracts

Contracts specify which **program executions**
a side-channel adversary **cannot distinguish**

Contract

ISA

+

Observations

HW/SW contracts

Contracts specify which **program executions**
a side-channel adversary **cannot distinguish**

Contract

ISA

+

Observations

Captures how
program is executed

HW/SW contracts

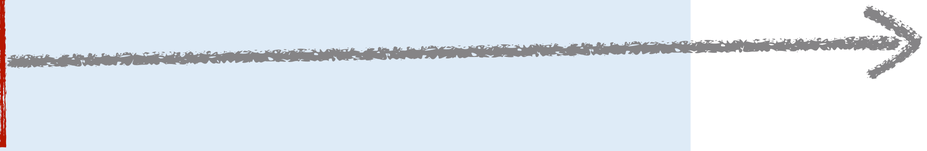
Contracts specify which **program executions** a side-channel adversary **cannot distinguish**

Contract

ISA

+

Observations

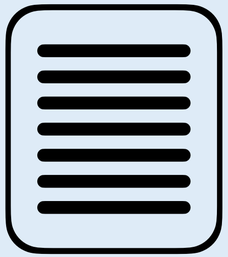


Captures how program is executed

What leaks about an execution

Contracts

Contracts

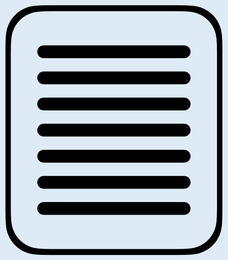


Contract

ISA +

Observations

Contracts



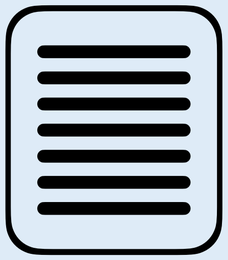
Contract

ISA +

Observations

Contract traces: (p, σ)

Contracts

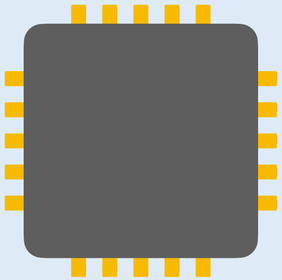


Contract

ISA +

Observations

Contract traces: (p, σ)

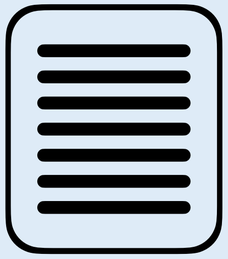


Hardware

µArch design +

Attacker observations

Contracts

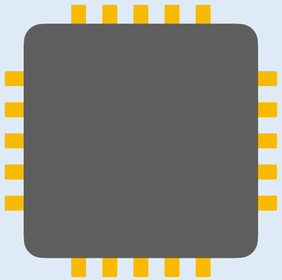


Contract

ISA +

Observations

Contract traces: (p, σ)



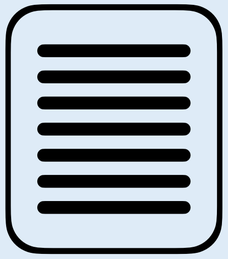
Hardware

µArch design +

Attacker observations

Hardware traces: (p, σ)

Contracts

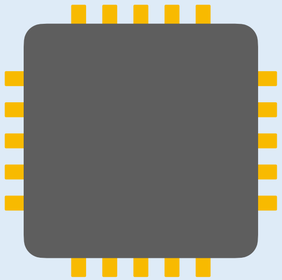


Contract

ISA +

Observations

Contract traces: (p, σ)



Hardware

µArch design +

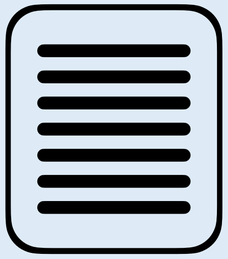
Attacker observations

Hardware traces: (p, σ)

Contract satisfaction

Hardware satisfies contract if for all programs p and arch. states σ, σ': if $(p, \sigma) = (p, \sigma')$ then $(p, \sigma) = (p, \sigma')$

Contracts

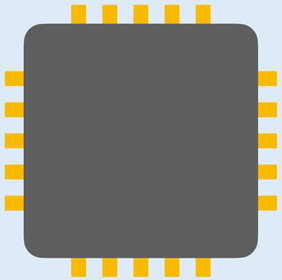


Contract

ISA +

Observations

Contract traces: (p, σ)



Hardware

µArch design +

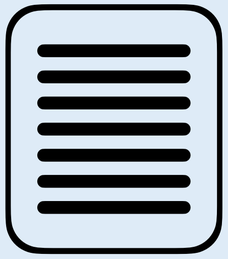
Attacker observations

Hardware traces: (p, σ)

Contract satisfaction

Hardware satisfies contract if for all programs p and arch. states σ, σ': if $(p, \sigma) = (p, \sigma')$ then $(p, \sigma) = (p, \sigma')$

Contracts

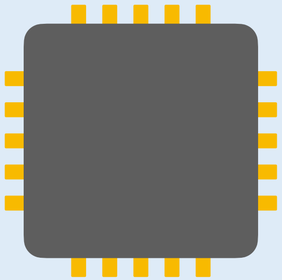


Contract

ISA +

Observations

Contract traces: (p, σ)



Hardware

µArch design +

Attacker observations

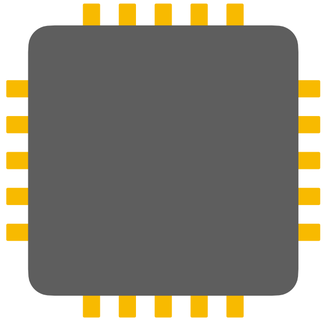
Hardware traces: (p, σ)

Contract satisfaction

Hardware satisfies contract if for all programs p and arch. states σ, σ': if $(p, \sigma) = (p, \sigma')$ then $(p, \sigma) = (p, \sigma')$

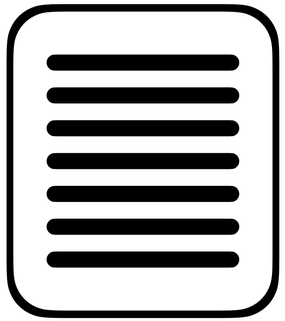
Verifying Contract Satisfaction

Contracts for Real ISAs + Real CPUs



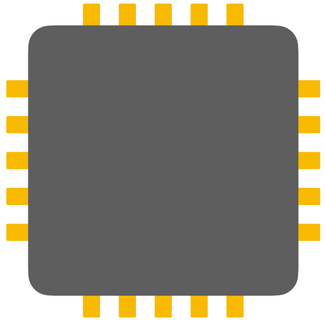
Microarchitecture

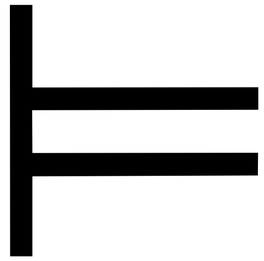
$\models$

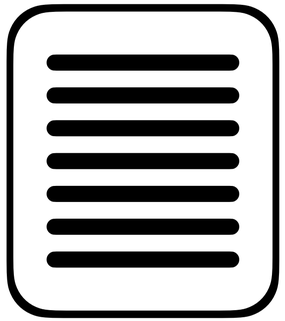


Contract

Contracts for Real ISAs + Real CPUs





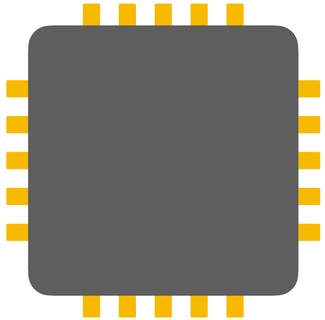


Microarchitecture

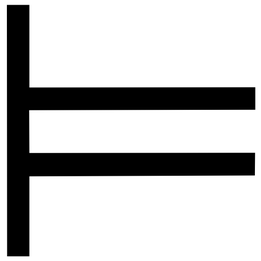
Contract

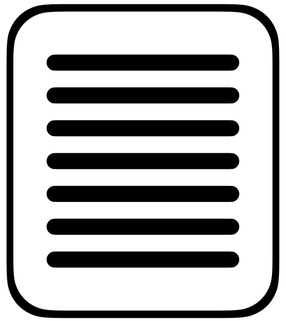
Register transfer level designs

Contracts for Real ISAs + Real CPUs



Microarchitecture



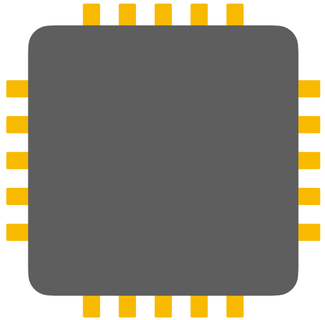


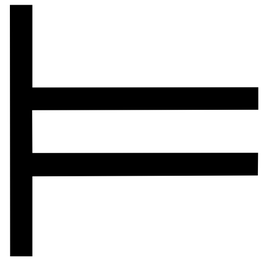
Contract

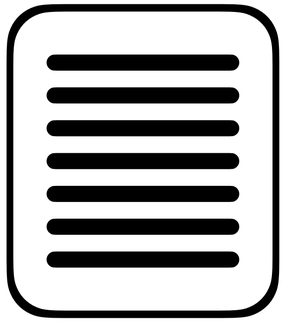
Register transfer level designs

Open-source RISC-V cores

Contracts for Real ISAs + Real CPUs







Microarchitecture

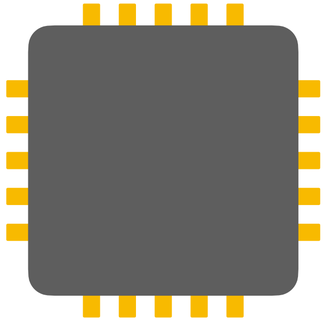
Register transfer level designs

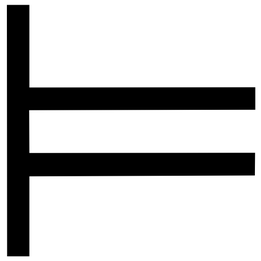
Open-source RISC-V cores

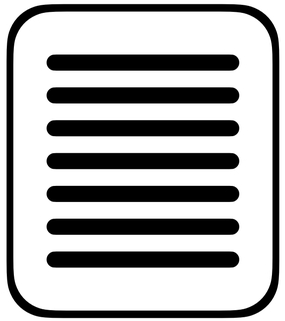
Contract

ISA spec.
+ Observations

Contracts for Real ISAs + Real CPUs







Microarchitecture

Register transfer level designs

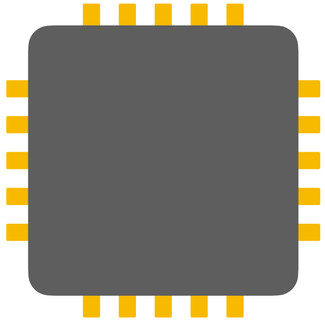
Open-source RISC-V cores

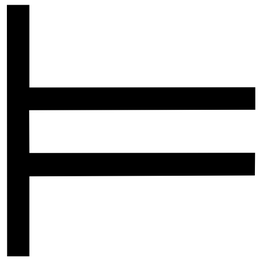
Contract

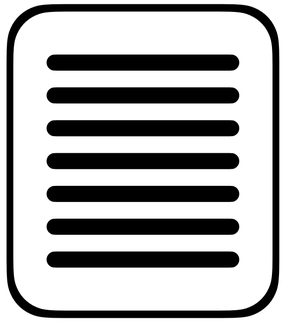
ISA spec. + Observations

1. Sail (https://github.com/rems-project/sail)

Contracts for Real ISAs + Real CPUs







Microarchitecture

Register transfer level designs

Open-source RISC-V cores

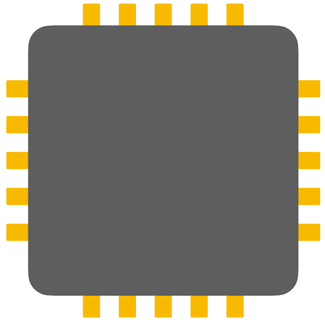
Contract

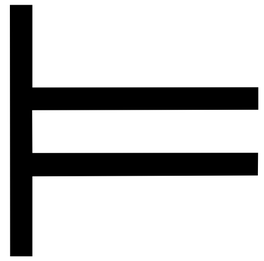
ISA spec. + Observations

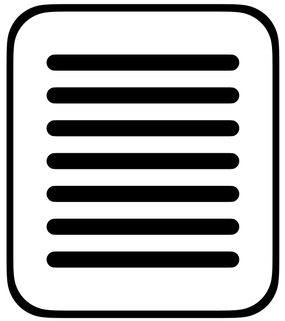
1. Sail (https://github.com/rems-project/sail)

2. Single-cycle RISC-V reference implementation + Combinatorial observer

Contracts for Real ISAs + Real CPUs







Microarchitecture

Contract

Register transfer level designs

Automatic proof

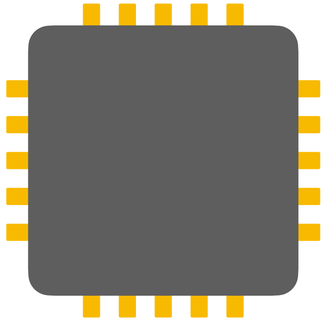
ISA spec.
+ Observations

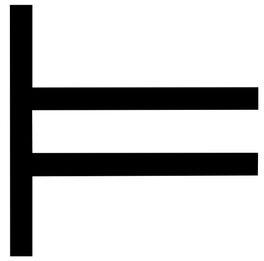
Open-source
RISC-V cores

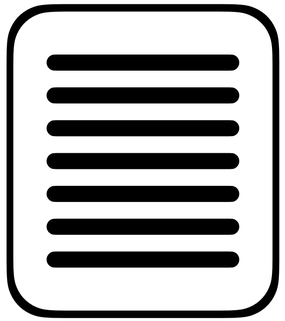
1. Sail (https://github.com/rems-project/sail)

2. Single-cycle RISC-V reference implementation + Combinatorial observer

Contracts for Real ISAs + Real CPUs







Microarchitecture

Contract

Register transfer level designs

Automatic proof

ISA spec.
+ Observations

Open-source RISC-V cores

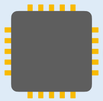
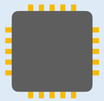
SMT solvers + Invariant inference

1. Sail (https://github.com/rems-project/sail)

2. Single-cycle RISC-V reference implementation + Combinatorial observer

Verification: 4-way product circuit + Induction

Contract satisfaction

Hardware  satisfies contract  if for all programs p and arch. states σ, σ': if  $(p, \sigma) =$  (p, σ') then  $(p, \sigma) =$  (p, σ')

Verification: 4-way product circuit + Induction

Contract satisfaction

Hardware 🔲 satisfies contract 📄 if for all programs p and arch. states σ, σ': if 📄$(p, \sigma) =$ 📄(p, σ') then 🔲$(p, \sigma) =$ 🔲(p, σ')

ISA_1

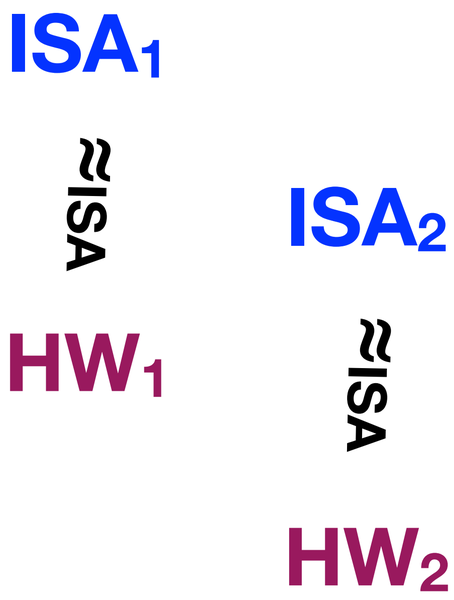
$\approx_{ISA}$

HW_1

Verification: 4-way product circuit + Induction

Contract satisfaction

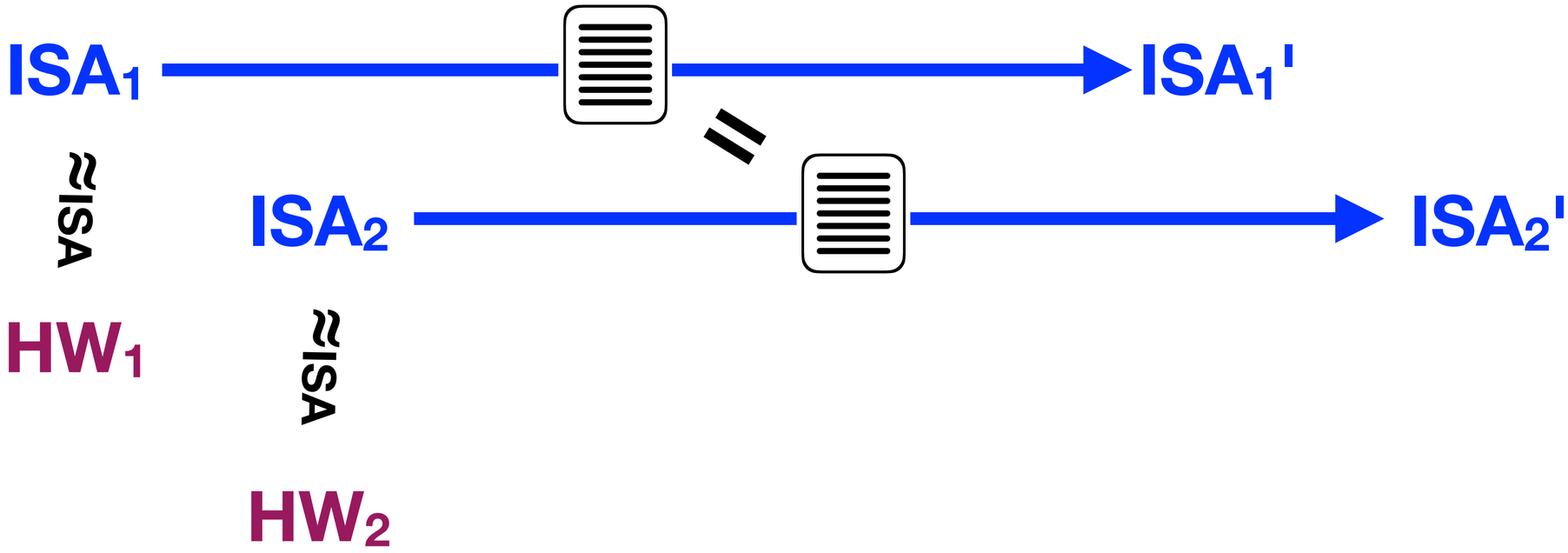
Hardware 🔲 satisfies contract 📄 if for all programs p and arch. states σ, σ': if 📄(p, σ)=📄(p, σ') then 🔲(p, σ) =🔲(p, σ')



Verification: 4-way product circuit + Induction

Contract satisfaction

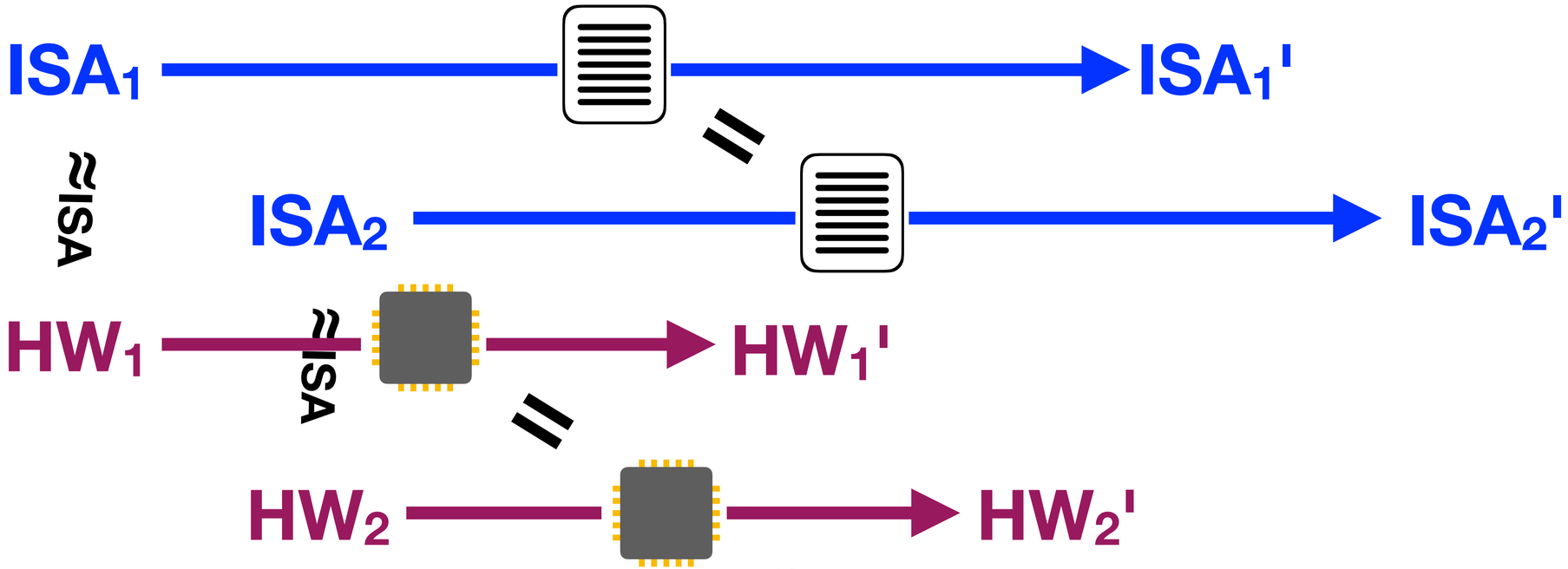
Hardware satisfies contract if for all programs p and arch. states σ, σ': if $(p, \sigma) = (p, \sigma')$ then $(p, \sigma) = (p, \sigma')$



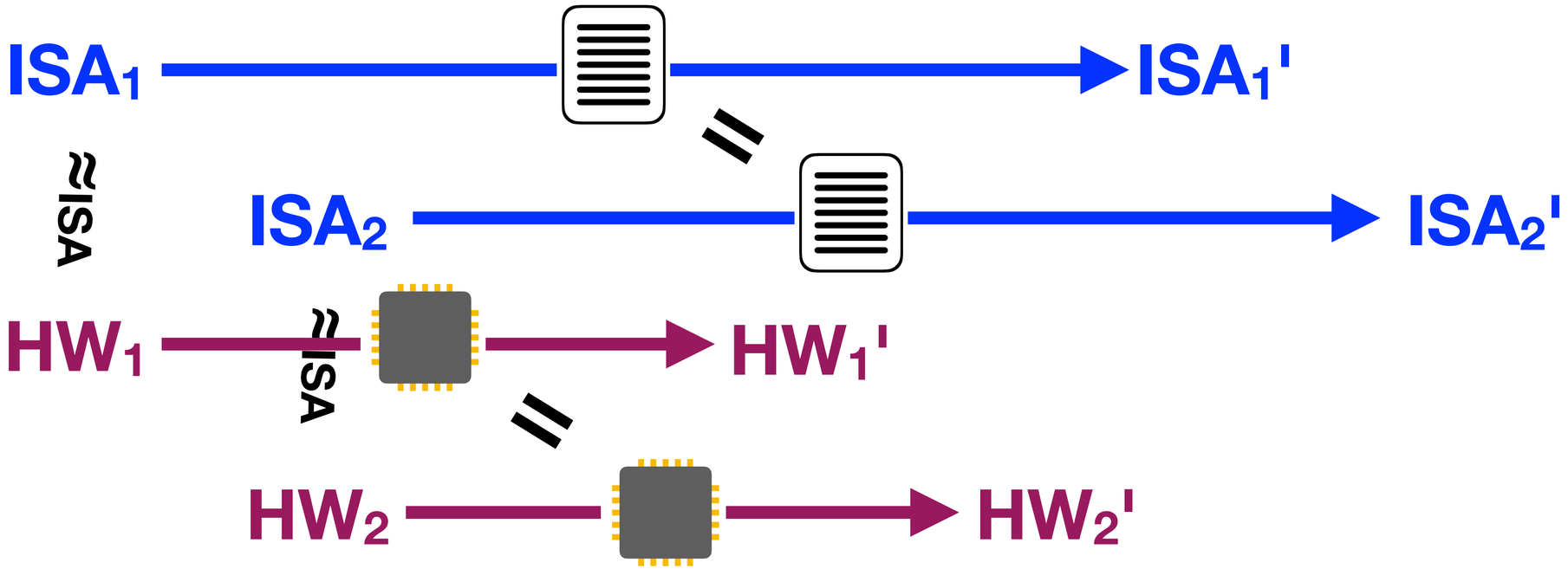
Verification: 4-way product circuit + Induction

Contract satisfaction

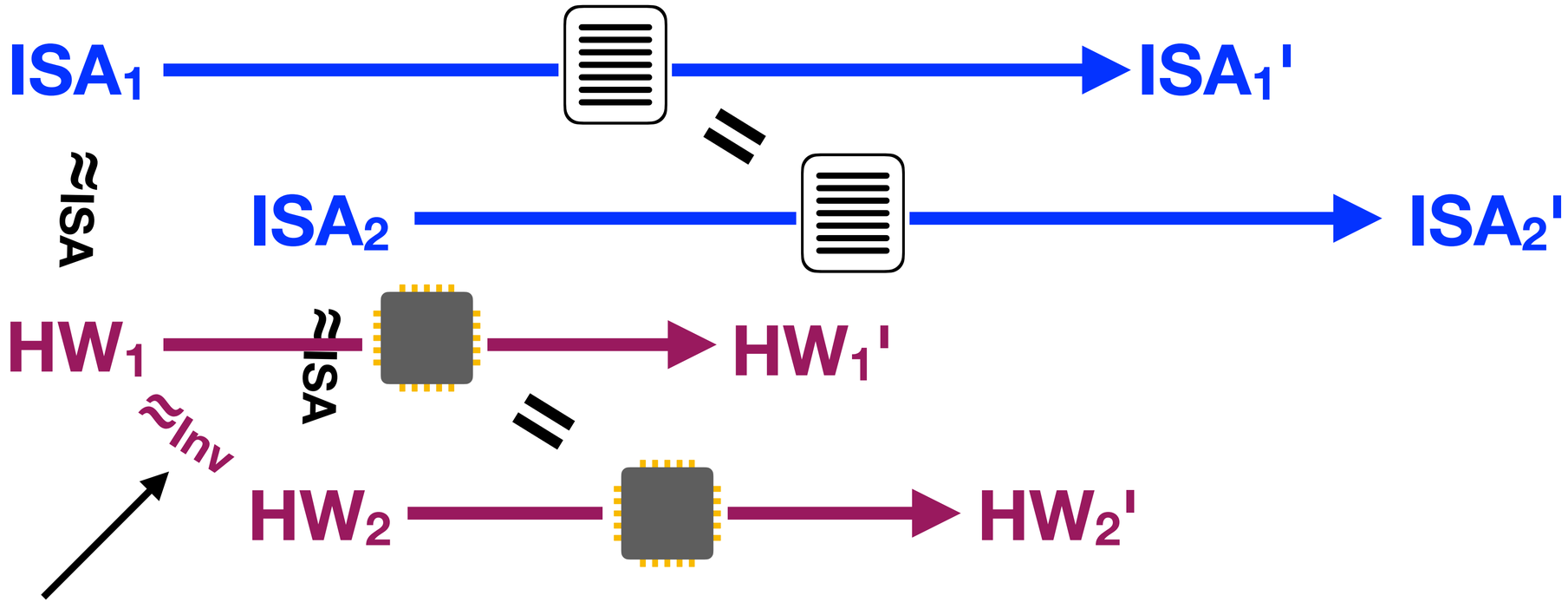
Hardware satisfies contract if for all programs p and arch. states σ, σ': if $(p, \sigma) = (p, \sigma')$ then $(p, \sigma) = (p, \sigma')$



Verification: 4-way product circuit + Induction

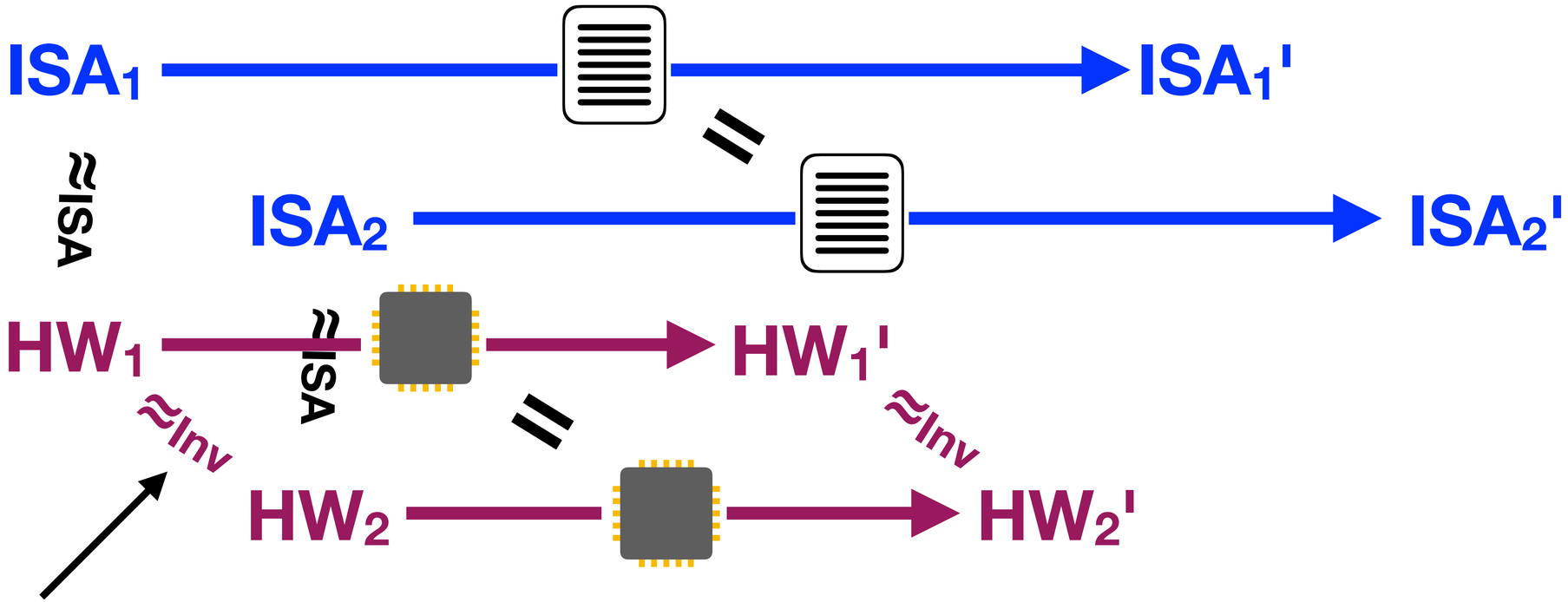


Verification: 4-way product circuit + Induction



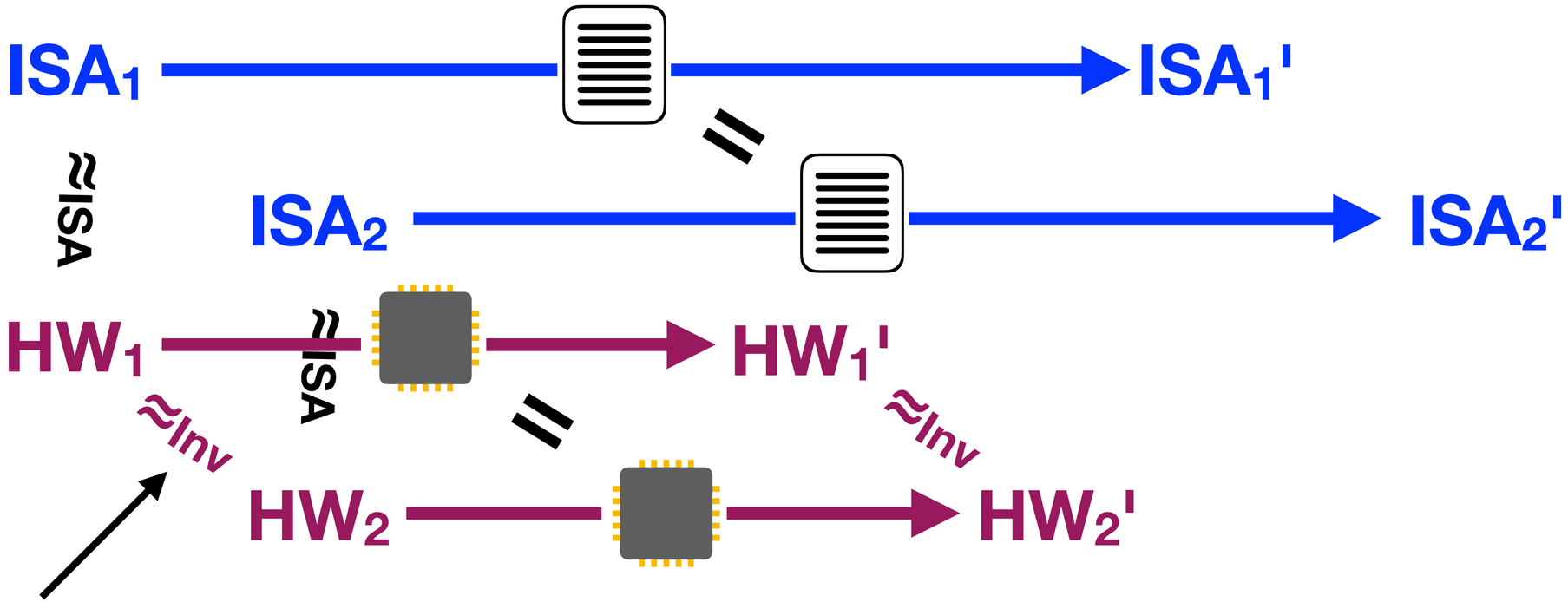
Relational
Inductive
Invariant

Verification: 4-way product circuit + Induction



Relational Inductive Invariant

Verification: 4-way product circuit + Induction

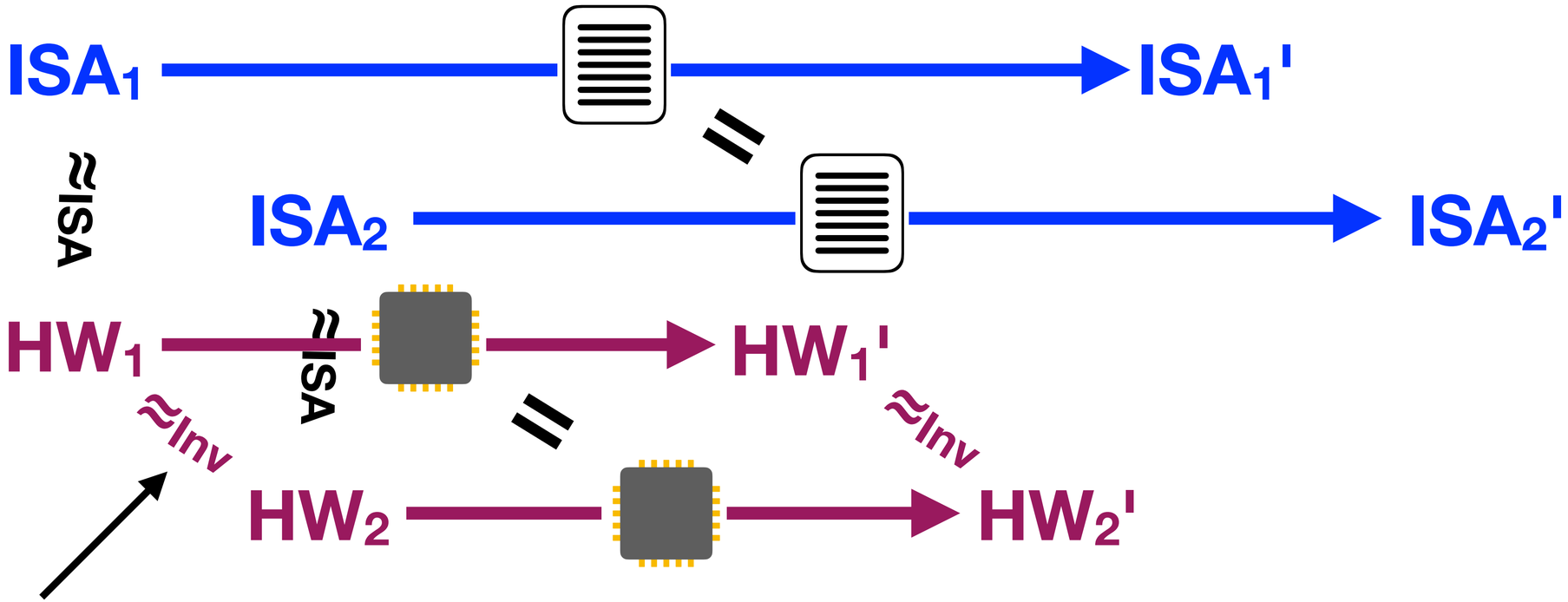


Relational Inductive Invariant

... synthesized using Houdini algorithm

(Flanagan+Leino: Houdini, an Annotation Assistant for ESC/Java, Formal Methods Europe)

Verification: 4-way product circuit + Induction



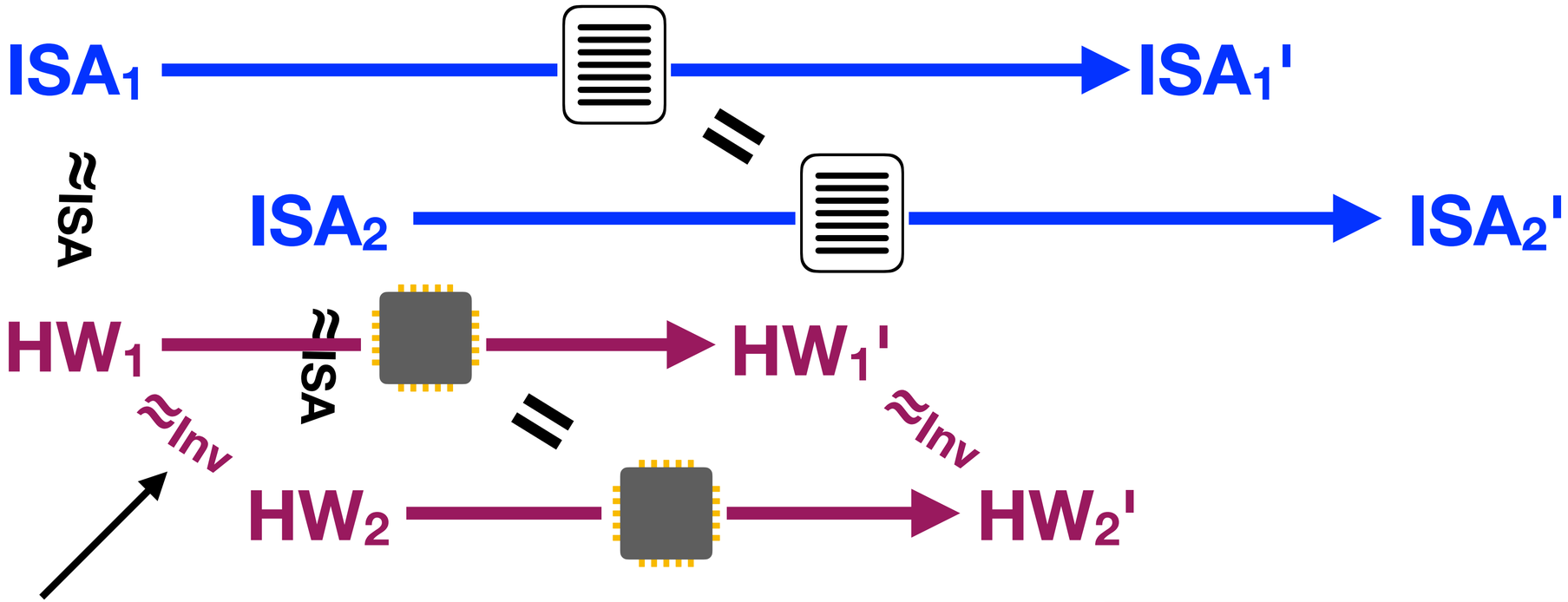
Relational Inductive Invariant

Works on small examples.
RISC-V Sodor Work-in-progress

... synthesized using Houdini algorithm

(Flanagan+Leino: Houdini, an Annotation Assistant for ESC/Java, Formal Methods Europe)

Verification: 4-way product circuit + Induction



Relational Inductive Invariant

Works on small examples.
RISC-V Sodor Work-in-progress

Scalability limited due to 4-way product circuit

... synthesized using Houdini algorithm

(Flanagan+Leino: Houdini, an Annotation Assistant for ESC/Java, Formal Methods Europe)

Disentangling Leakage and ISA satisfaction

Contract satisfaction

Hardware satisfies contract if for all programs p and states σ, σ':

if $Obs(ISA)(p, \sigma) = Obs(ISA)(p, \sigma')$

then $Atk(\mu Arch)(p, \sigma) = Atk(\mu Arch)(p, \sigma')$

Disentangling Leakage and ISA satisfaction

Contract satisfaction

Hardware satisfies contract if for all programs p and states σ, σ':

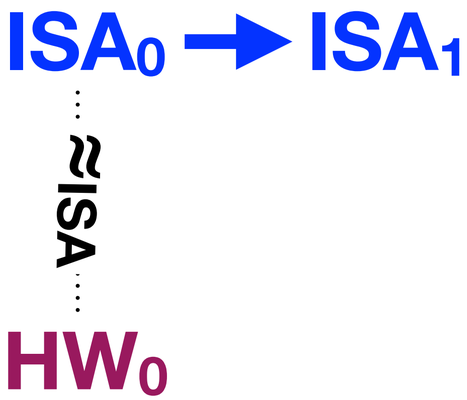
if $Obs(ISA)(p, \sigma) = Obs(ISA)(p, \sigma')$

then $Atk(\mu Arch)(p, \sigma) = Atk(\mu Arch)(p, \sigma')$

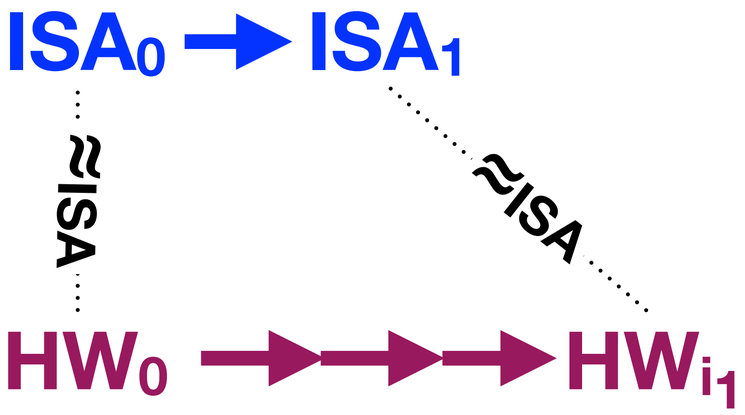
ISA satisfaction + **Leakage satisfaction**

ISA satisfaction

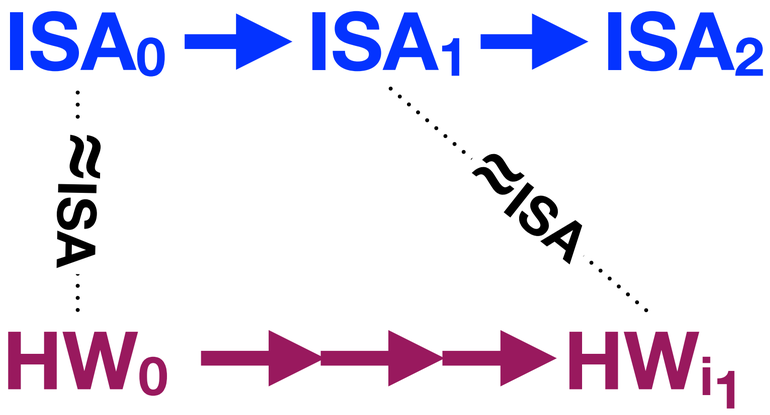
ISA satisfaction



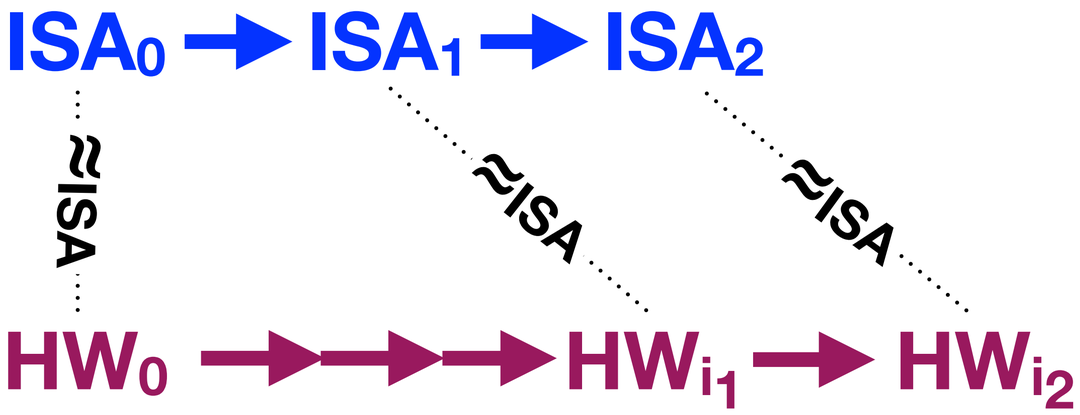
ISA satisfaction



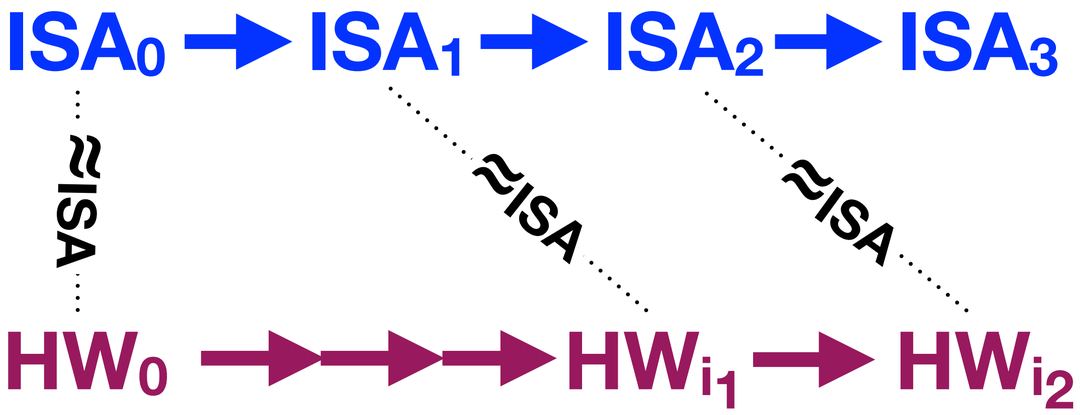
ISA satisfaction



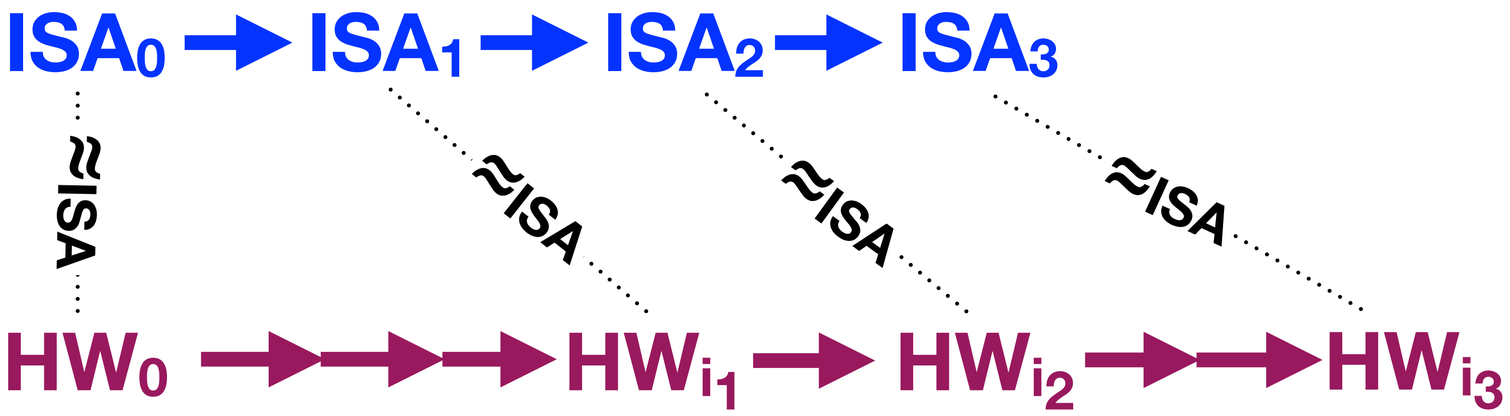
ISA satisfaction



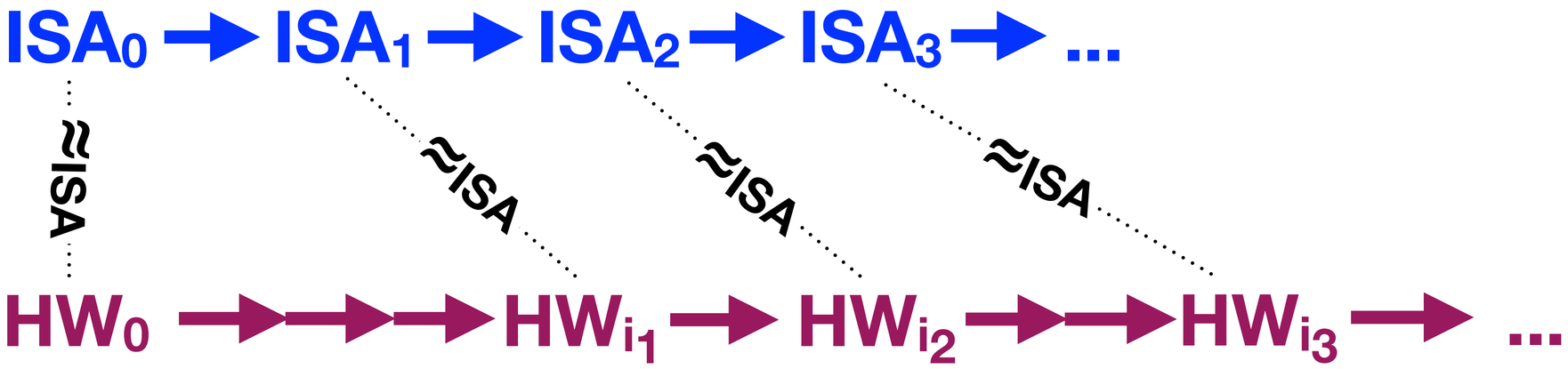
ISA satisfaction



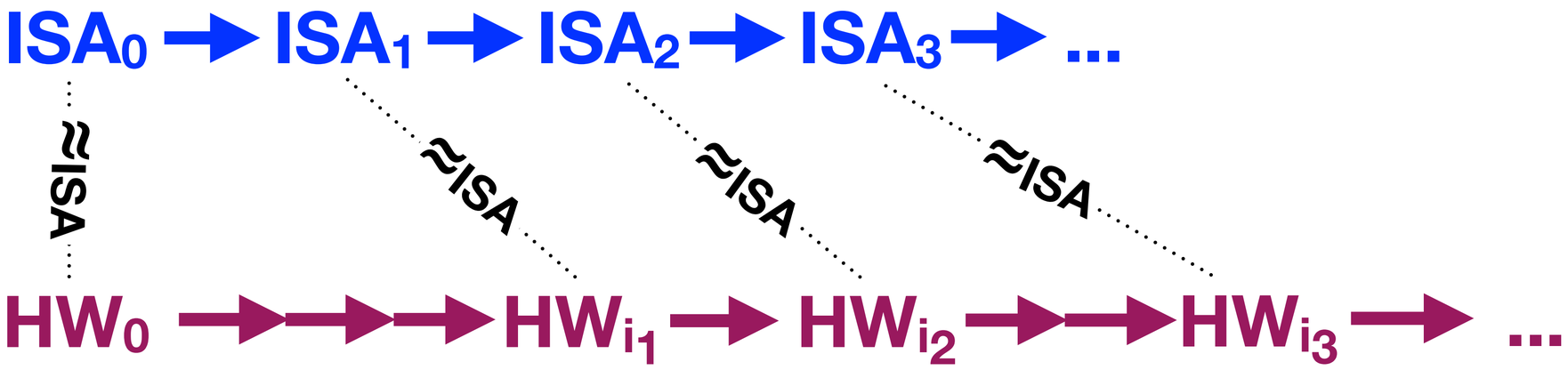
ISA satisfaction



ISA satisfaction



ISA satisfaction



The ISA trace is embedded in the HW trace.

Leakage satisfaction

Leakage satisfaction

For all programs p and states σ, σ':

$$\text{if } Obs(\mu Arch)(p, \sigma) = Obs(\mu Arch)(p, \sigma')$$

$$\text{then } Atk(\mu Arch)(p, \sigma) = Atk(\mu Arch)(p, \sigma')$$

+ Observations determine the timing of instruction retirement.

Leakage satisfaction

Leakage satisfaction

For all programs p and states σ, σ':

if $Obs(\mu Arch)(p, \sigma) = Obs(\mu Arch)(p, \sigma')$

then $Atk(\mu Arch)(p, \sigma) = Atk(\mu Arch)(p, \sigma')$

+ Observations determine the timing of instruction retirement.

Can be checked with 2-way product circuit.

Leakage satisfaction

Leakage satisfaction

For all programs p and states σ, σ':

$$\text{if } Obs(\mu Arch)(p, \sigma) = Obs(\mu Arch)(p, \sigma')$$

$$\text{then } Atk(\mu Arch)(p, \sigma) = Atk(\mu Arch)(p, \sigma')$$

+ Observations determine the timing of instruction retirement.

Can be checked with 2-way product circuit.

Verified DarkRISCV leakage

Leakage satisfaction

Leakage satisfaction

For all programs p and states σ, σ':

$$\text{if } Obs(\mu Arch)(p, \sigma) = Obs(\mu Arch)(p, \sigma')$$

$$\text{then } Atk(\mu Arch)(p, \sigma) = Atk(\mu Arch)(p, \sigma')$$

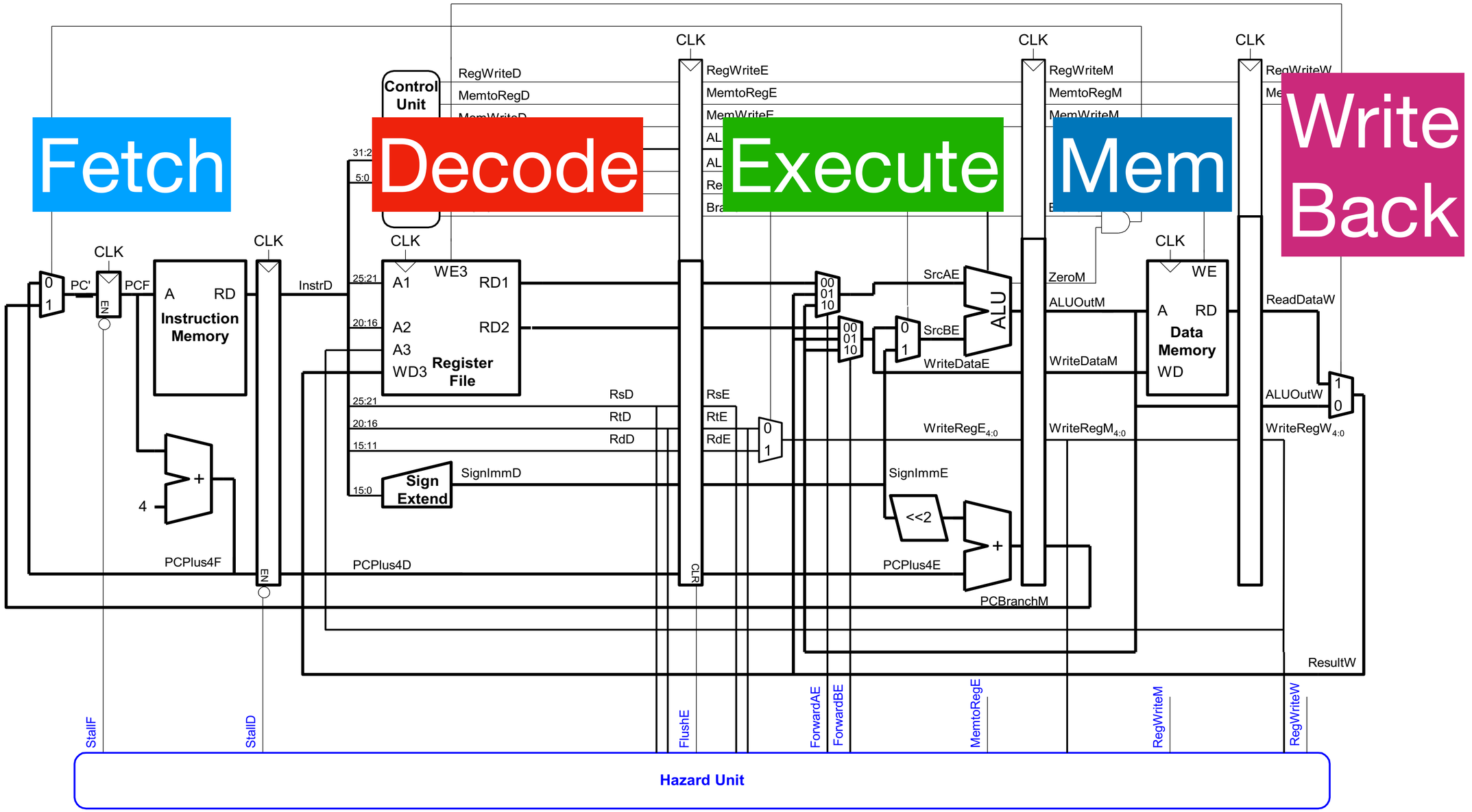
+ Observations determine the timing of instruction retirement.

Can be checked with 2-way product circuit.

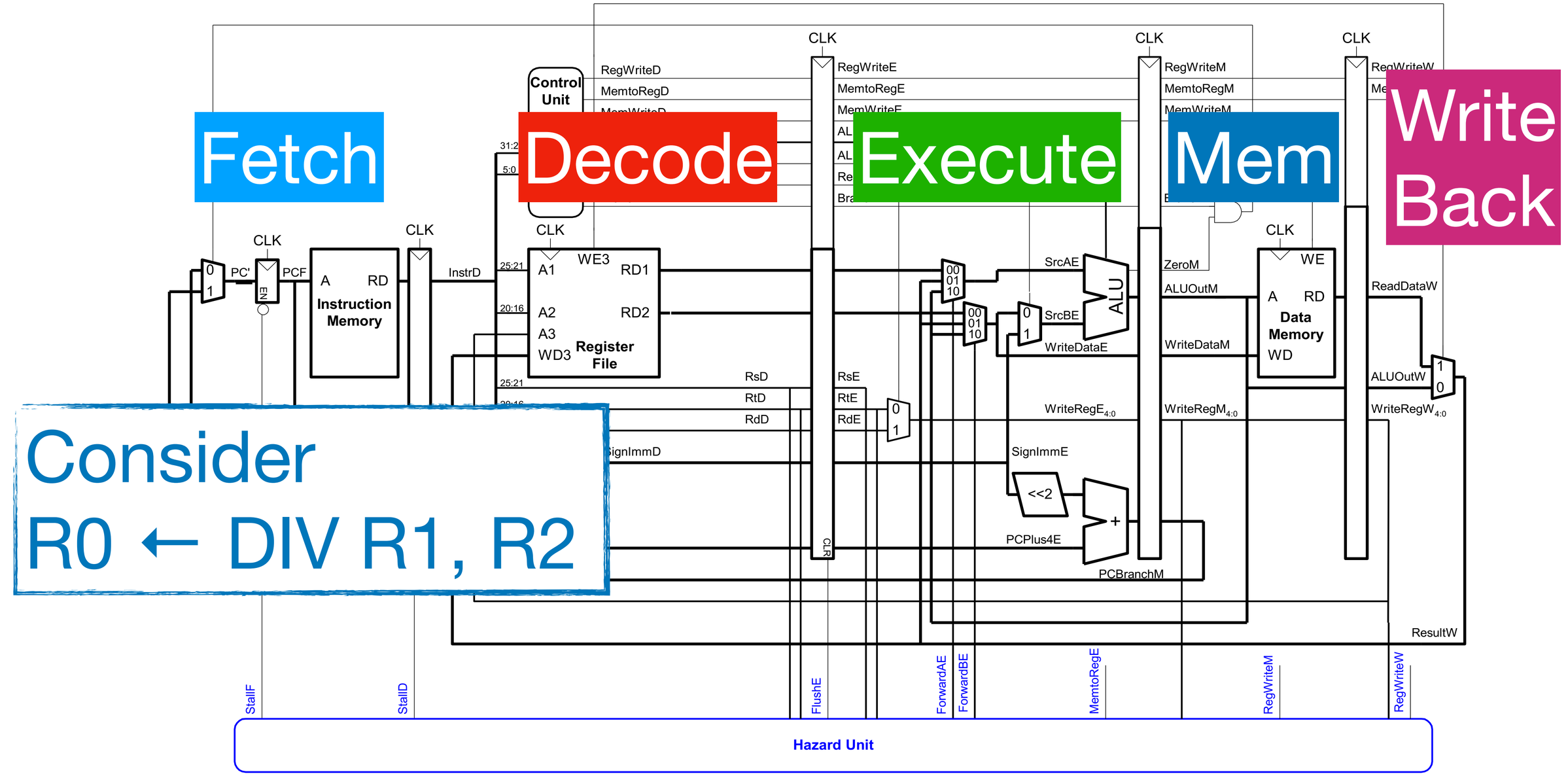
Verified DarkRISCV leakage

Proof relies on "pipeline invariants"

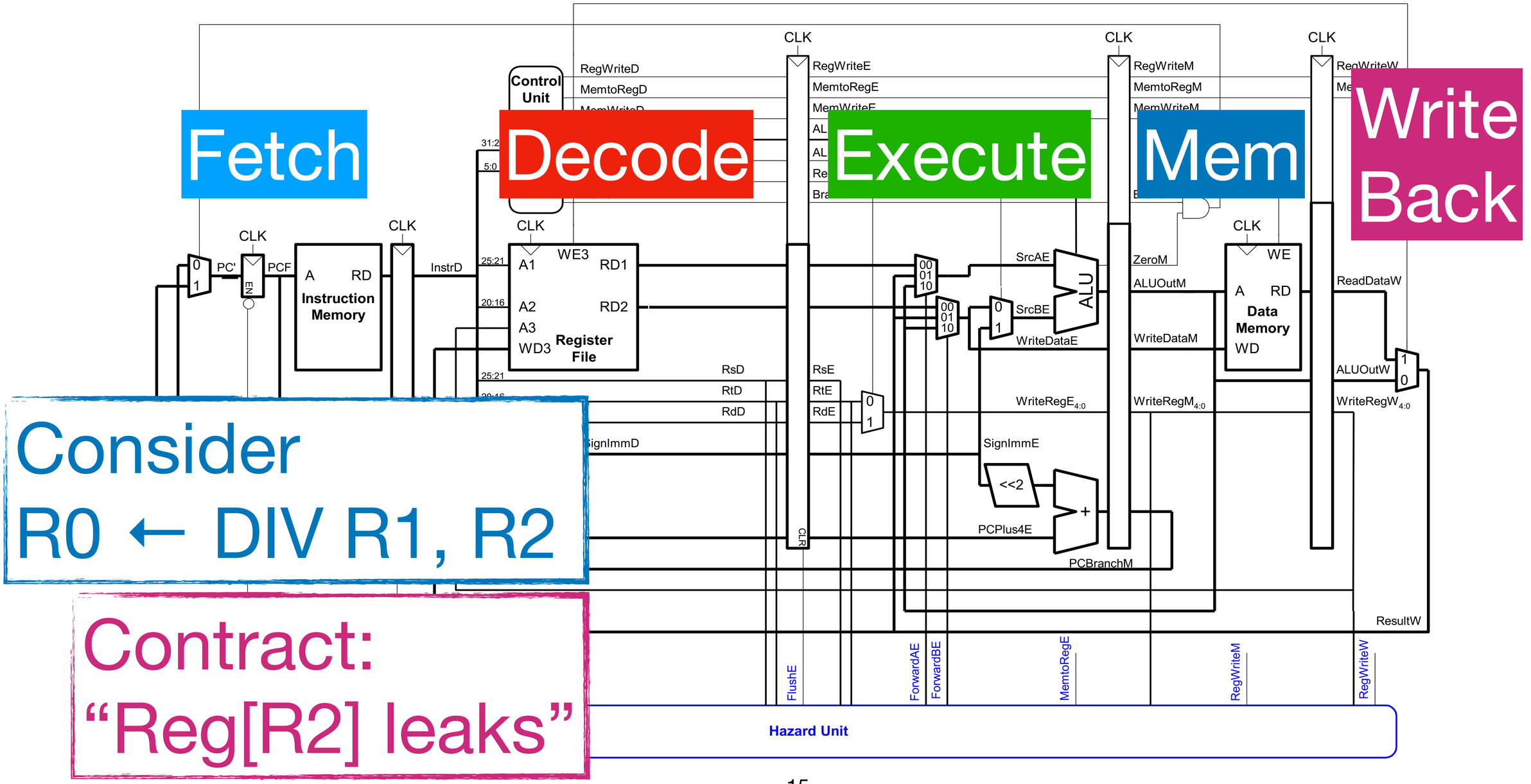
Pipeline invariants: Example



Pipeline invariants: Example



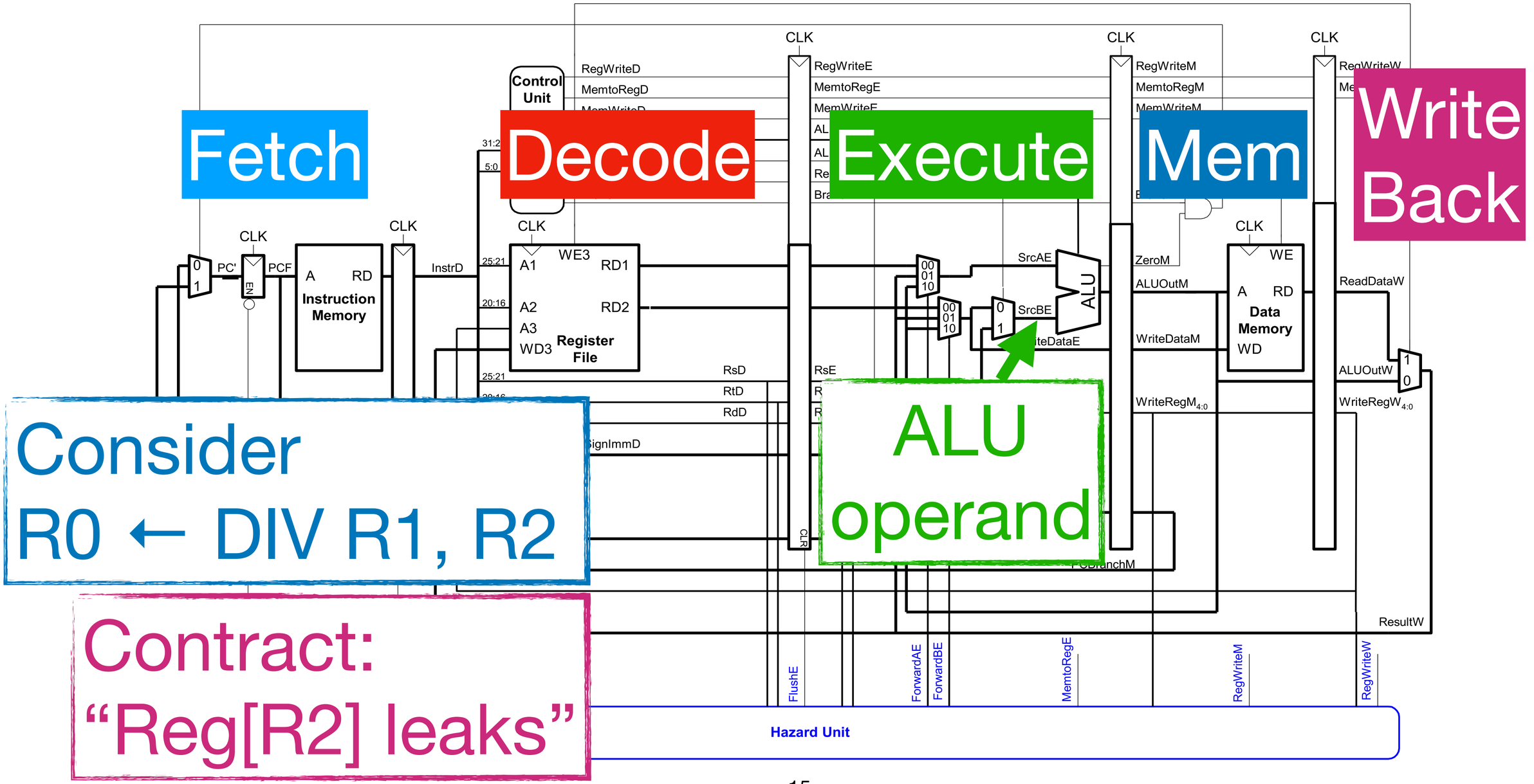
Pipeline invariants: Example



Consider
R0 ← DIV R1, R2

Contract:
“Reg[R2] leaks”

Pipeline invariants: Example

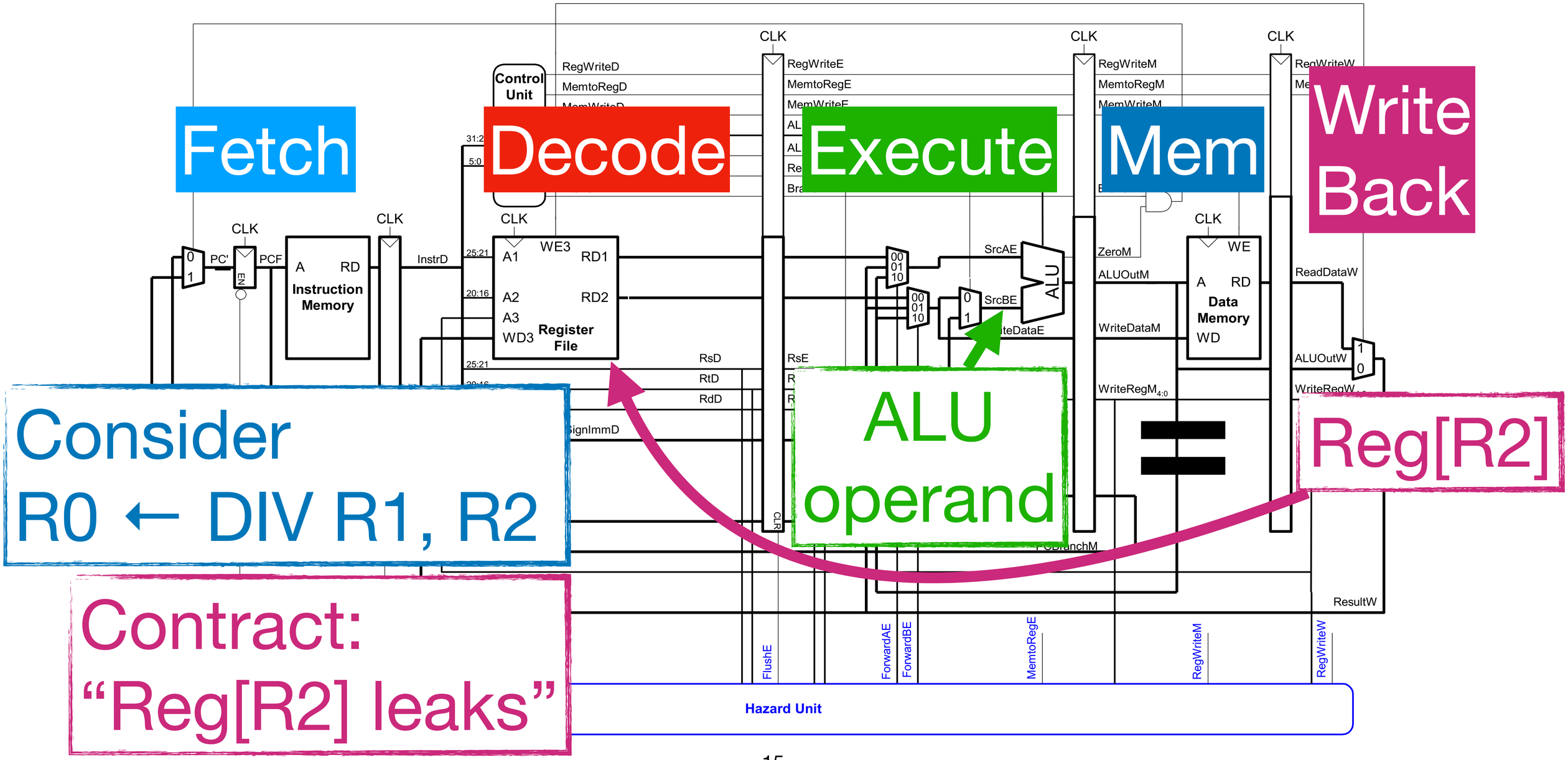


Consider
R0 ← DIV R1, R2

ALU operand

Contract:
“Reg[R2] leaks”

Pipeline invariants: Example



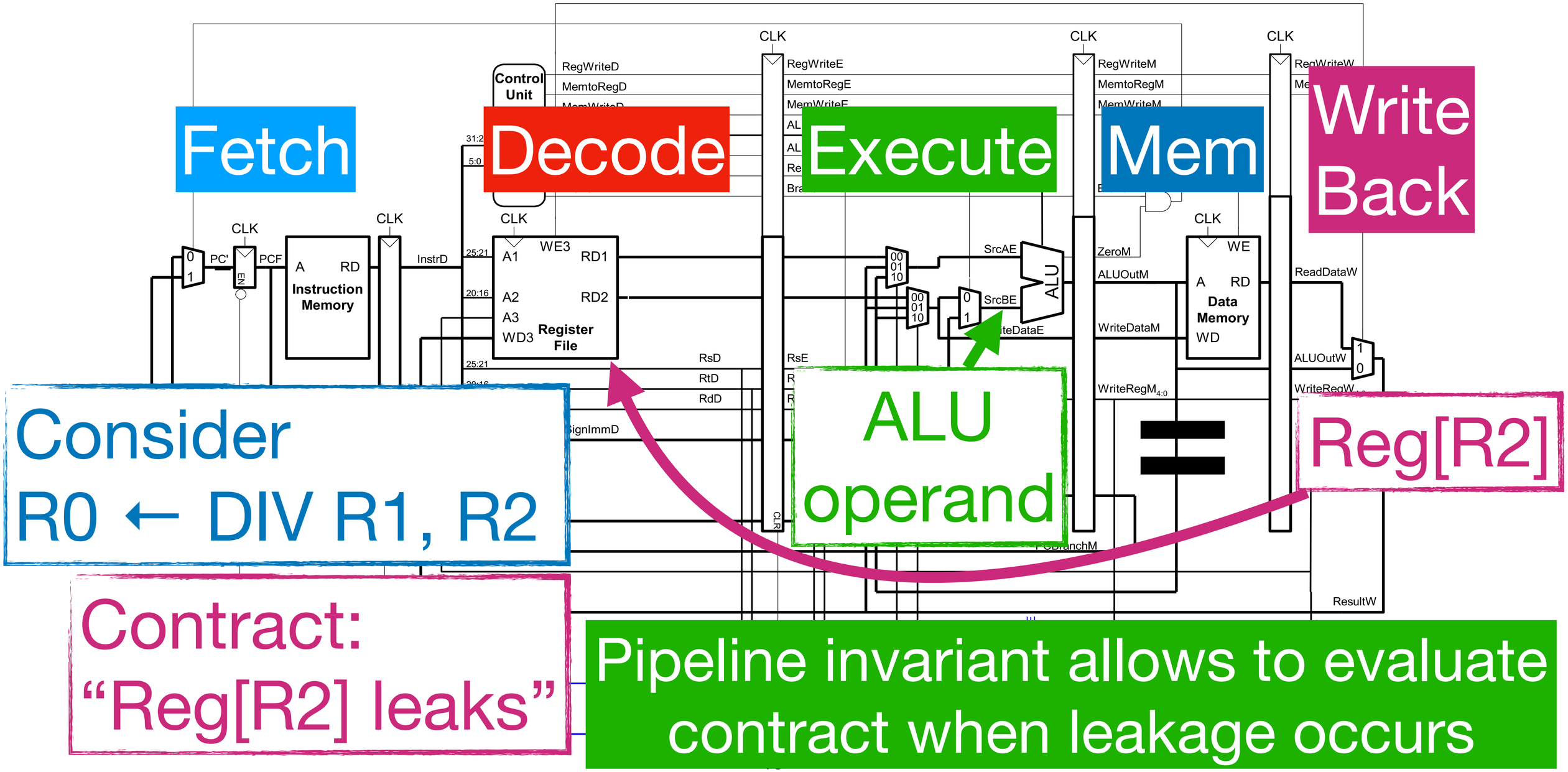
Consider
R0 ← DIV R1, R2

Contract:
“Reg[R2] leaks”

ALU operand

Reg[R2]

Pipeline invariants: Example



Conclusions

Conclusions

HW/SW contracts capture leakage at ISA level

… thus enable secure programming

Conclusions

HW/SW contracts capture leakage at ISA level
… thus enable secure programming

Verifying microarchitectures is challenging
… leakage verification and functional verification
can be disentangled

Conclusions

HW/SW contracts capture leakage at ISA level
… thus enable secure programming

Verifying microarchitectures is challenging
… leakage verification and functional verification can be disentangled

Come to our poster to discuss!