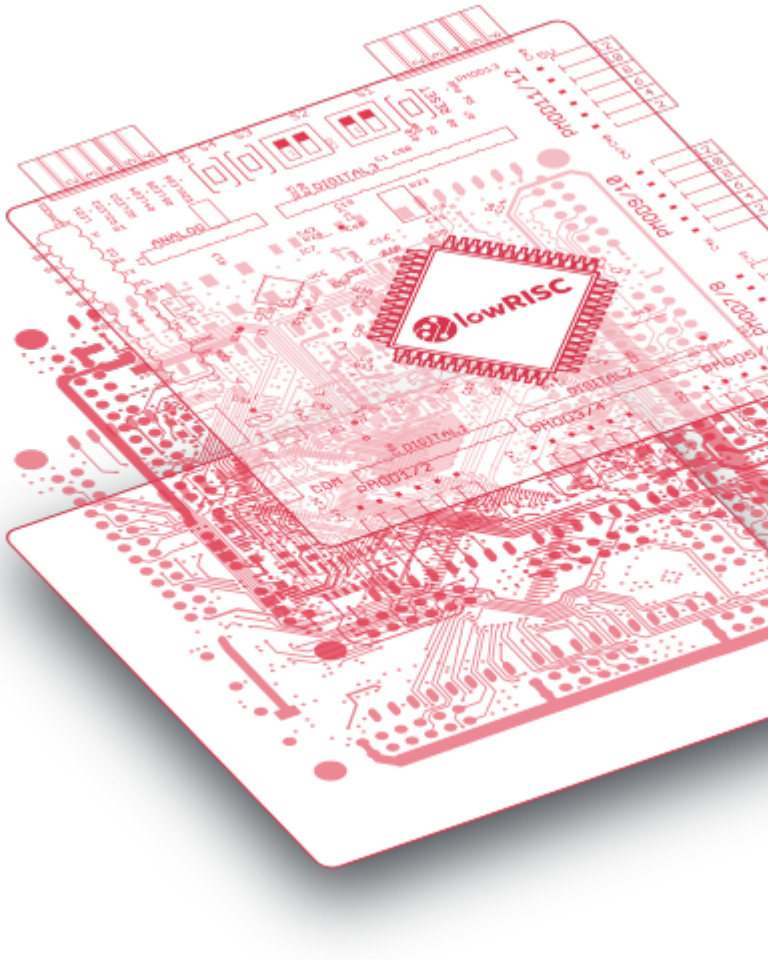
lowRISC

COSMIC verification

Comprehensive, end-to-end formal proofs of CVA6-CHERI

Louis-Emile Ploix, Marno van der Maas

CHERI Blossoms, 27/03/2026



Talk organisation

- Part 1:
  Verification strategy for Mocha
- Part 2:
  Comprehensive formal verification of CVA6-CHERI

Verification strategy in Mocha

Design Verification Dashboard

This summarizes the results from our nightly OpenTitan regression which runs a wide variety of tests for each block as well as a chip-level tests.

Chip-Level

Block	Tests	Passing	Code Coverage	Toggle Coverage	Assert Coverage	Functional Coverage
chip_earlgrey	2956	99.1 %	94.4 %	95.4 %	97.7 %	99.6 %

Block-Level

Block	Tests	Passing	Code Coverage	Toggle Coverage	Assert Coverage	Functional Coverage
adc_ctrl	920	99.8 %	98.6 %	100 %	98.3 %	91.6 %
aes/masked	1602	97.6 %	98.0 %	97.7 %	99.1 %	96.6 %
aes/unmasked	1602	95.9 %	95.0 %	97.7 %	98.8 %	96.8 %

Development stages

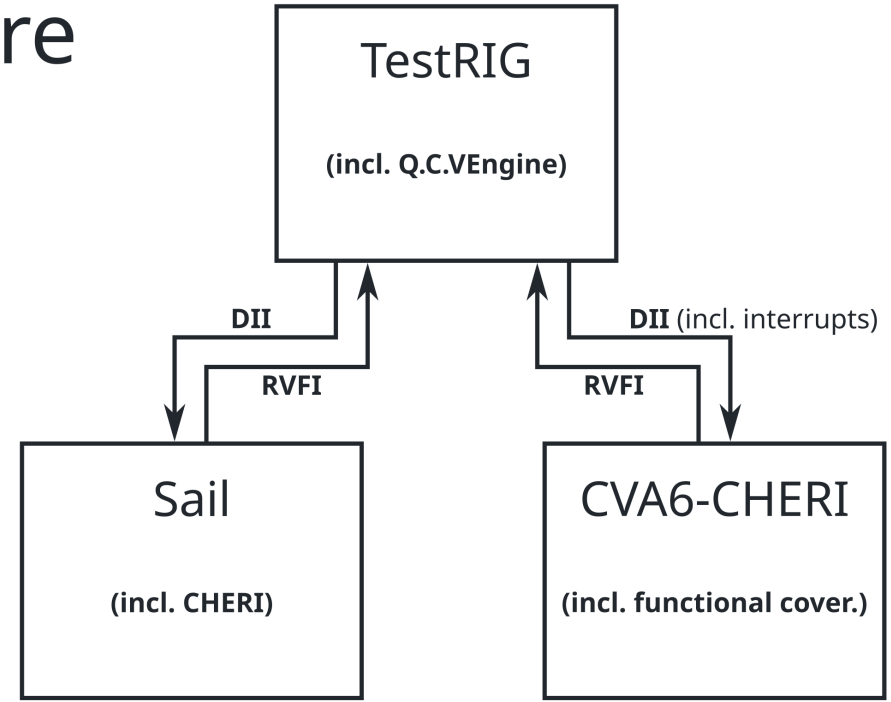
	Design	Verification
Specified	D0	V0
Developed	D1	V1
Functional	D2	V2
Secure	D2S	V2S
Complete	D3	V3

Simulation Results: top_mocha_batch_sim

09/03/2026 07:07:30 | DVSim: v1.11.3
sha: 3cfe15d | json | Branch: main

Block	Tests			Coverage Summary				Code Coverage					
	Pass	Total	%	Overall	Code	Functional	Assertion	Block	Line	Branch	Condition	Toggle	FSM
GPIO	0	970	0.00	-	-	-	-	-	-	-	-	-	-
I2C	0	2042	0.00	-	-	-	-	-	-	-	-	-	-
PRIM_ALERT	0	100	0.00	-	-	-	-	-	-	-	-	-	-
PRIM_ESC	0	20	0.00	-	-	-	-	-	-	-	-	-	-
PRIM_LFSR	0	200	0.00	-	-	-	-	-	-	-	-	-	-
ROM_CTRL/32KB	0	460	0.00	-	-	-	-	-	-	-	-	-	-
RV_DM/USE_JTAG_INTERFACE	0	483	0.00	-	-	-	-	-	-	-	-	-	-
RV_TIMER	306	350	87.43	97.46	95.85	100.00	96.52	99.10	99.02	99.26	-	89.26	-
SPI_DEVICE/2P	2396	2446	97.96	83.51	92.74	63.39	94.41	98.50	98.95	97.23	-	85.20	89.58
TOP_MOCHA_SIM	10	20	50.00	57.86	47.14	-	68.58	77.15	82.75	56.54	-	37.09	12.20
UART	1797	1865	96.35	94.82	96.65	90.69	97.12	99.08	99.52	98.34	-	88.74	100.00

Test architecture



Formal verification of CHERI-CVA6

What is large scale FV?

What is large scale FV?

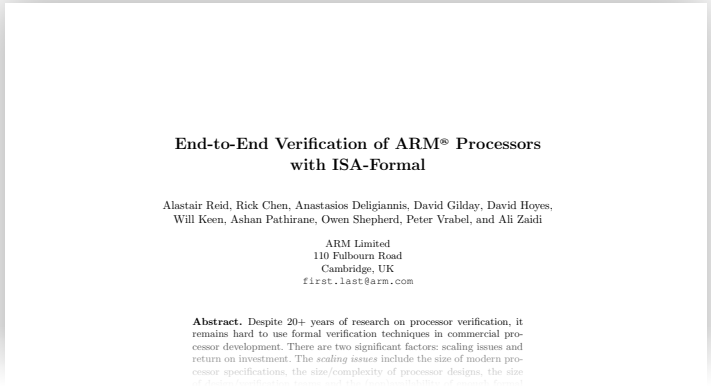
- Architectural, not block level

What is large scale FV?

- Architectural, not block level
- Extend ISA-Formal, from Arm (but fully convergent!)

What is large scale FV?

- Architectural, not block level
- Extend ISA-Formal, from Arm (but fully convergent!)



End-to-End Verification of ARM® Processors with ISA-Formal

Alastair Reid, Rick Chen, Anastasios Deligiannis, David Gilday, David Hoyes, Will Keen, Ashan Pathirane, Owen Shepherd, Peter Vrabel, and Ali Zaidi

ARM Limited
110 Fulbourn Road
Cambridge, UK
first.last@arm.com

Abstract. Despite 20+ years of research on processor verification, it remains hard to use formal verification techniques in commercial processor development. There are two significant factors: scaling issues and return on investment. The *scaling issues* include the size of modern processor specifications, the size/complexity of processor designs, the size

Why?

Why?

- Ibex/CHERIoT-Ibex has taught us that we can get value kinda fast!

Why?

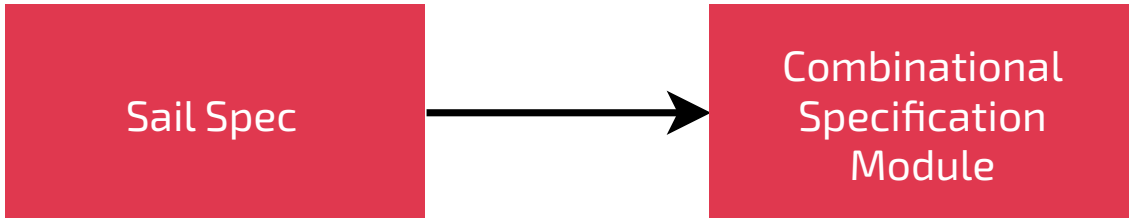
- Ibex/CHERIoT-Ibex has taught us that we can get value kinda fast!
- CHERI is hard to implement and small bugs affect security

Why?

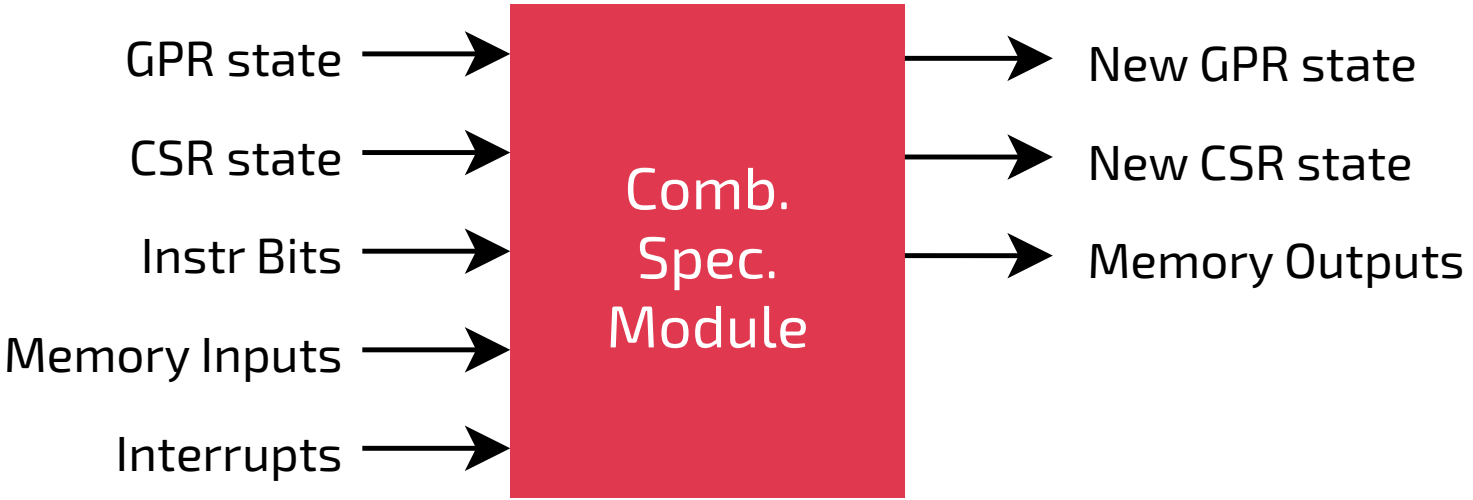
- Ibex/CHERIoT-Ibex has taught us that we can get value kinda fast!
- CHERI is hard to implement and small bugs affect security
- CHERI non-trivially affects the whole architecture

End-to-End

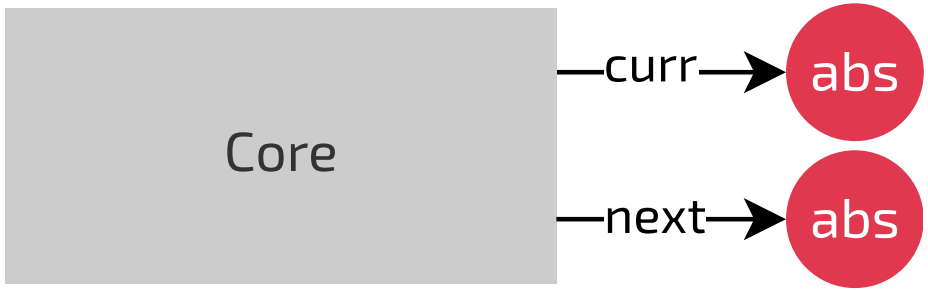
End-to-End



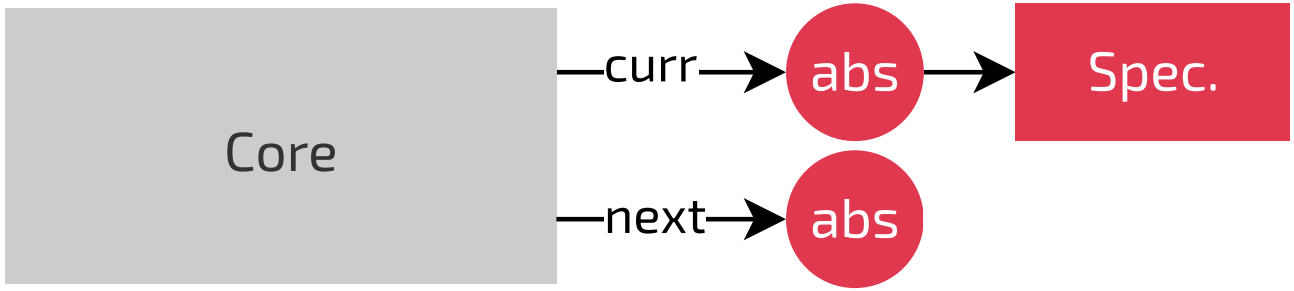
End-to-End



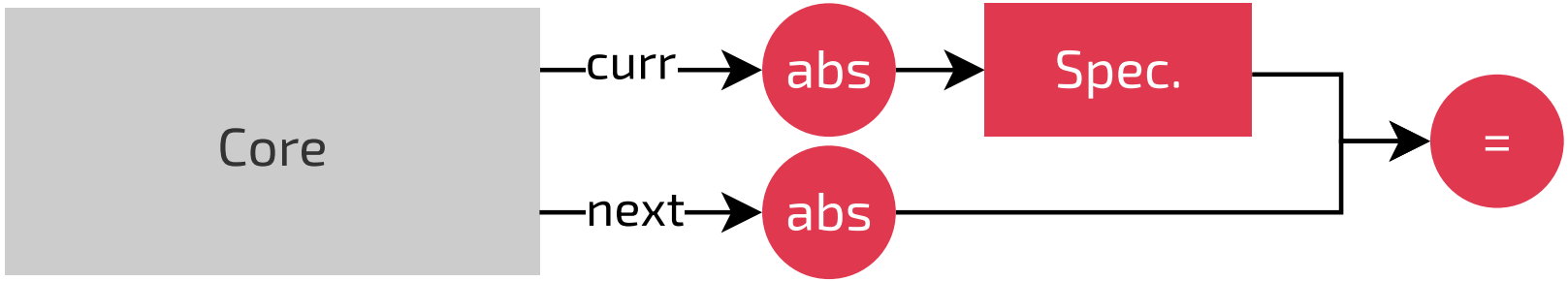
End-to-End



End-to-End



End-to-End



End-to-End

Provided the GPRs, CSRs and instruction bits are all correct,

End-to-End

Provided the GPRs, CSRs and instruction bits are all correct,

End-to-End

Provided the GPRs, CSRs and instruction bits are all correct,

then at commit the new GPRs and CSRs will be correct.

End-to-End

End-to-End

Does not tell us:

End-to-End

Does not tell us:

- If the CSRs and GPRs are always correct (especially the PC)

End-to-End

Does not tell us:

- If the CSRs and GPRs are always correct (especially the PC)
- If the instruction bits are correct

End-to-End

Does not tell us:

- If the CSRs and GPRs are always correct (especially the PC)
- If the instruction bits are correct
- Anything at all if our interpretation of internal state is wrong

End-to-End: CVA6-CHERI

End-to-End: CVA6-CHERI

- So far:

End-to-End: CVA6-CHERI

- So far:
  - GPRs for:

End-to-End: CVA6-CHERI

- So far:
  - GPRs for:
    - CHERI register manipulation instructions

End-to-End: CVA6-CHERI

- So far:
  - GPRs for:
    - CHERI register manipulation instructions
    - R-types and I-types

End-to-End: CVA6-CHERI

- So far:
  - GPRs for:
    - CHERI register manipulation instructions
    - R-types and I-types
    - (C?)JAL(R?) (excl. exceptions)

End-to-End: CVA6-CHERI

- So far:
  - GPRs for:
    - CHERI register manipulation instructions
    - R-types and I-types
    - (C?)JAL(R?) (excl. exceptions)
  - Large suite of lemmas to get all of that working!

End-to-End: CVA6-CHERI

- So far:
  - GPRs for:
    - CHERI register manipulation instructions
    - R-types and I-types
    - (C?)JAL(R?) (excl. exceptions)
  - Large suite of lemmas to get all of that working!
- Still to do:

End-to-End: CVA6-CHERI

- So far:
  - GPRs for:
    - CHERI register manipulation instructions
    - R-types and I-types
    - (C?)JAL(R?) (excl. exceptions)
  - Large suite of lemmas to get all of that working!
- Still to do:
  - CSRs, PC, Exceptions, Memory

Bugs (so far)

Bugs (so far)

- ACPERM not properly legalising permissions — *#28*

Bugs (so far)

- ACPERM not properly legalising permissions — *#28*
- GCMODE/CGetFlags failing to check that perms are valid — *#26*

Bugs (so far)

- ACPERM not properly legalising permissions — *#28*
- GCMODE/CGetFlags failing to check that perms are valid — *#26*
- CLU and control flow instructions may execute concurrently — *#23*

Bugs (so far)

- ACPERM not properly legalising permissions — *#28*
- GCMODE/CGetFlags failing to check that perms are valid — *#26*
- CLU and control flow instructions may execute concurrently — *#23*
- SCBNDSR can produce bad bounds — *#19*

Bugs (so far)

- ACPERM not properly legalising permissions — *#28*
- GCMODE/CGetFlags failing to check that perms are valid — *#26*
- CLU and control flow instructions may execute concurrently — *#23*
- SCBNDSR can produce bad bounds — *#19*
- SCBNDSR addr_bits not maintained — *#15*

How do we get proofs? k-induction.

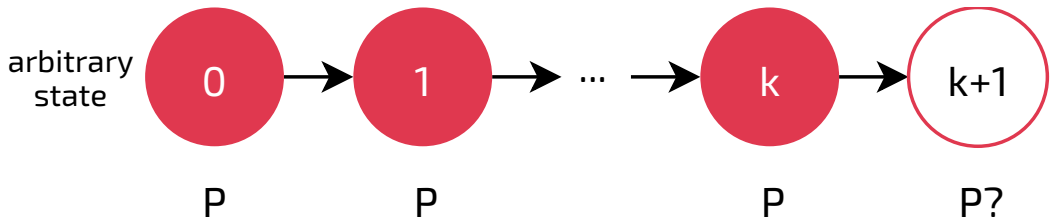
How do we get proofs? k-induction.

Intuitively:

How do we get proofs? k-induction.

Intuitively:

Proofs come from looking at a sequence of k states starting from some potentially unreachable state, and finding no counterexample in state (k + 1).

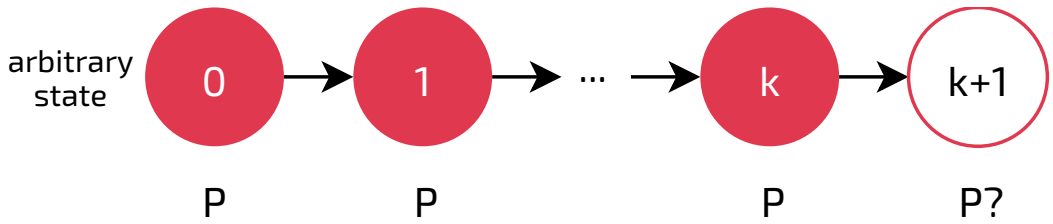


How do we get proofs? k-induction.

Intuitively:

Proofs come from looking at a sequence of k states starting from some potentially unreachable state, and finding no counterexample in state (k + 1).

(+ base case)



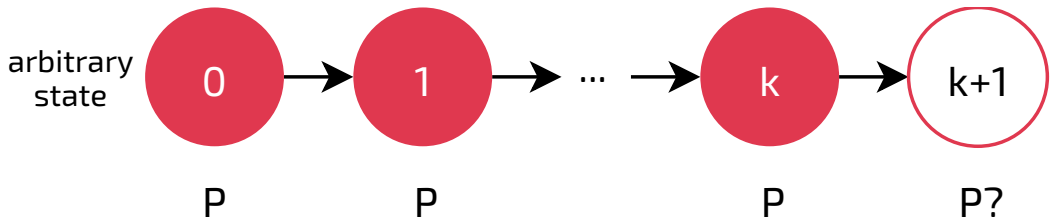
How do we get proofs? k-induction.

Intuitively:

Proofs come from looking at a sequence of k states starting from some potentially unreachable state, and finding no counterexample in state (k + 1).

(+ base case)

k-induction is backwards looking.



How do we get proofs? k-induction.

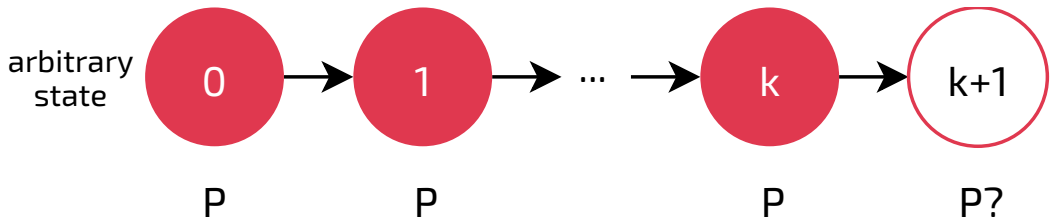
Intuitively:

Proofs come from looking at a sequence of k states starting from some potentially unreachable state, and finding no counterexample in state (k + 1).

(+ base case)

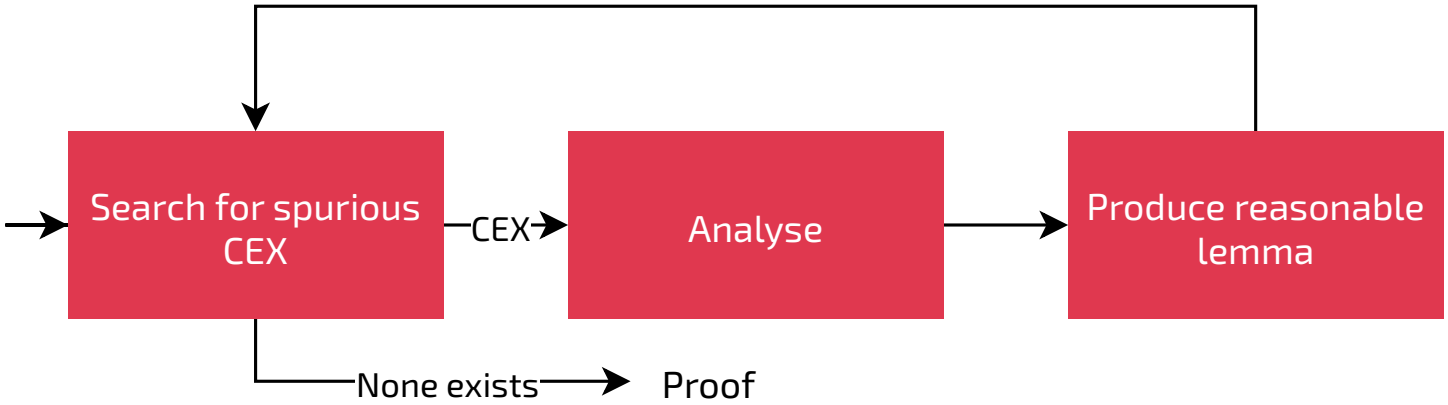
k-induction is backwards looking.

We can prune away some unreachable states with lemmas.



How do we prove it?

How do we prove it?



Time to get into how CVA6 works...

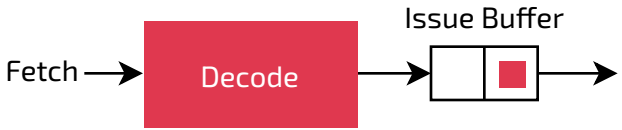
The lifetime of an instruction

The lifetime of an instruction

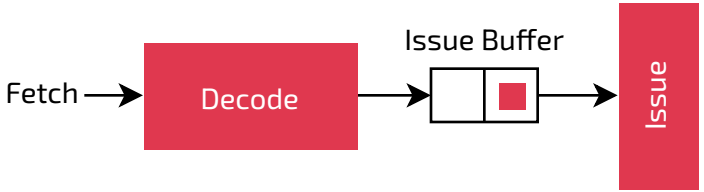
Fetch →

The lifetime of an instruction

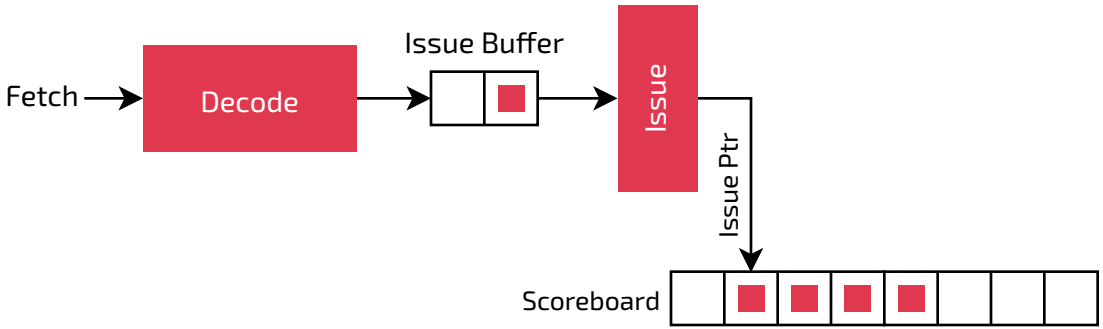
The lifetime of an instruction



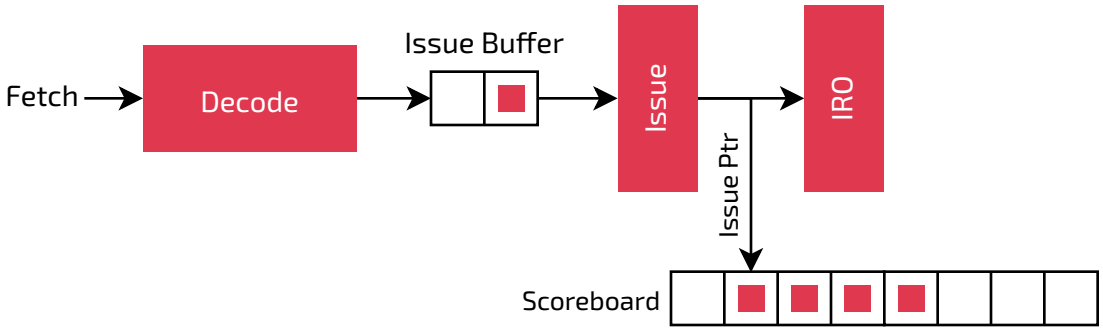
The lifetime of an instruction



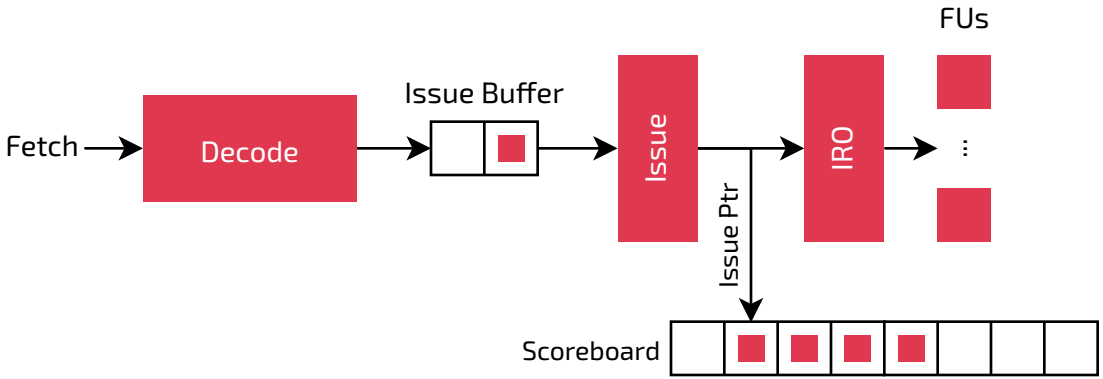
The lifetime of an instruction



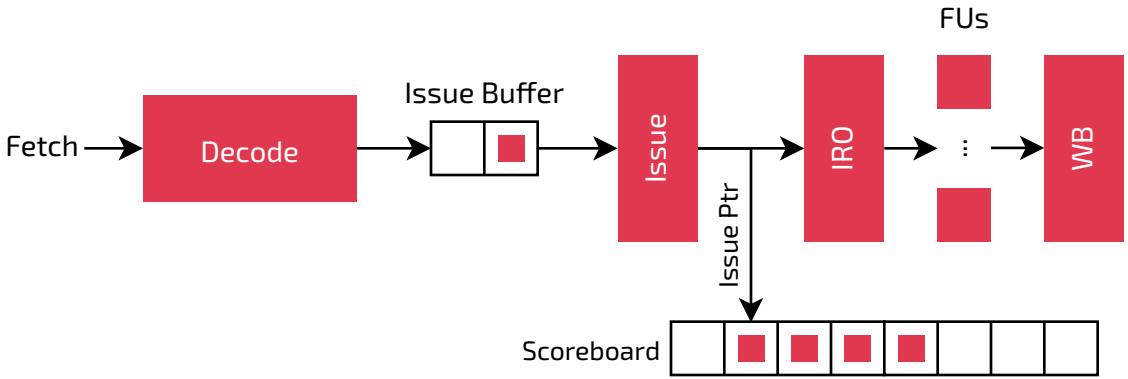
The lifetime of an instruction



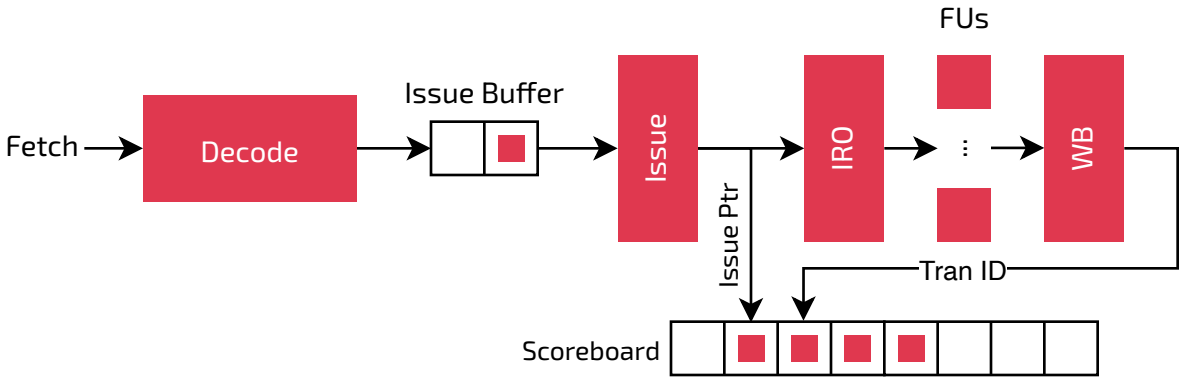
The lifetime of an instruction



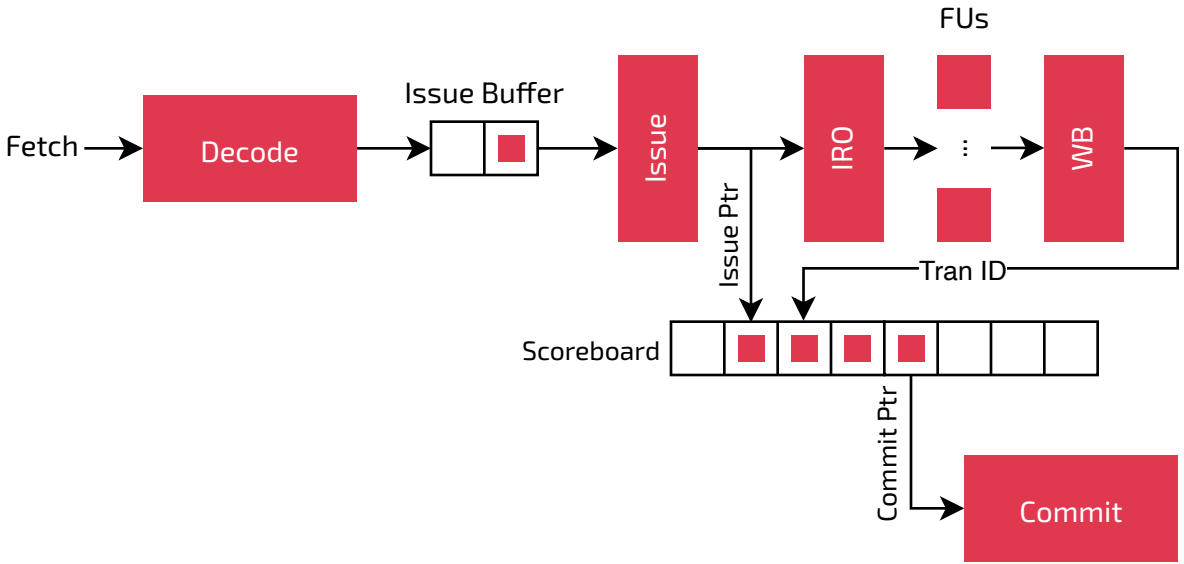
The lifetime of an instruction



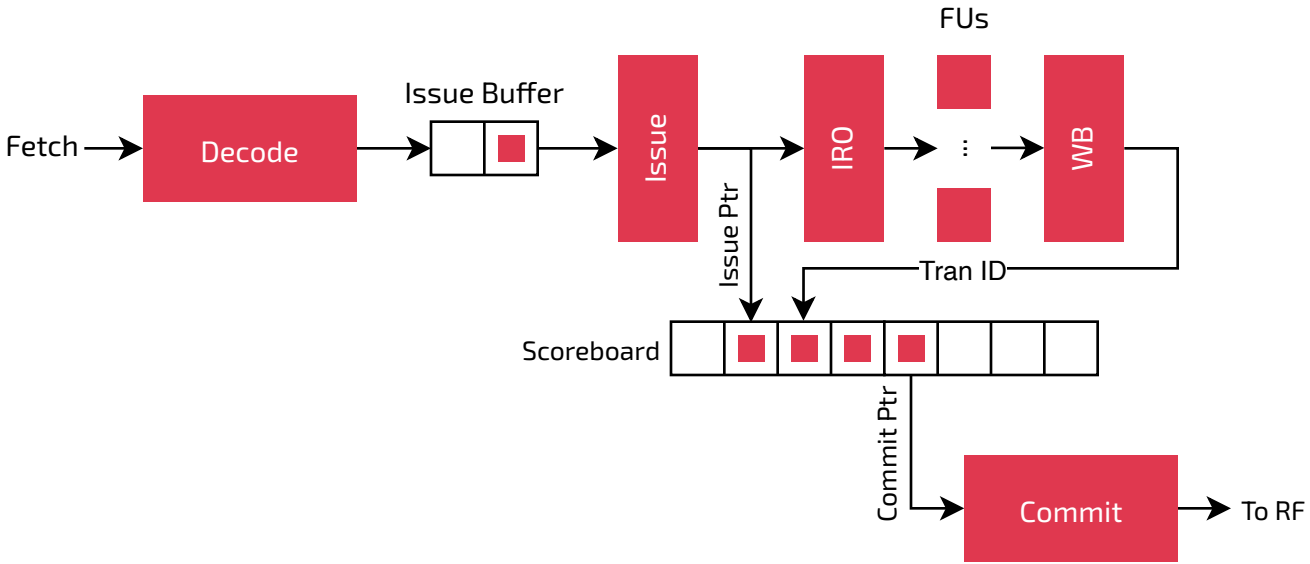
The lifetime of an instruction



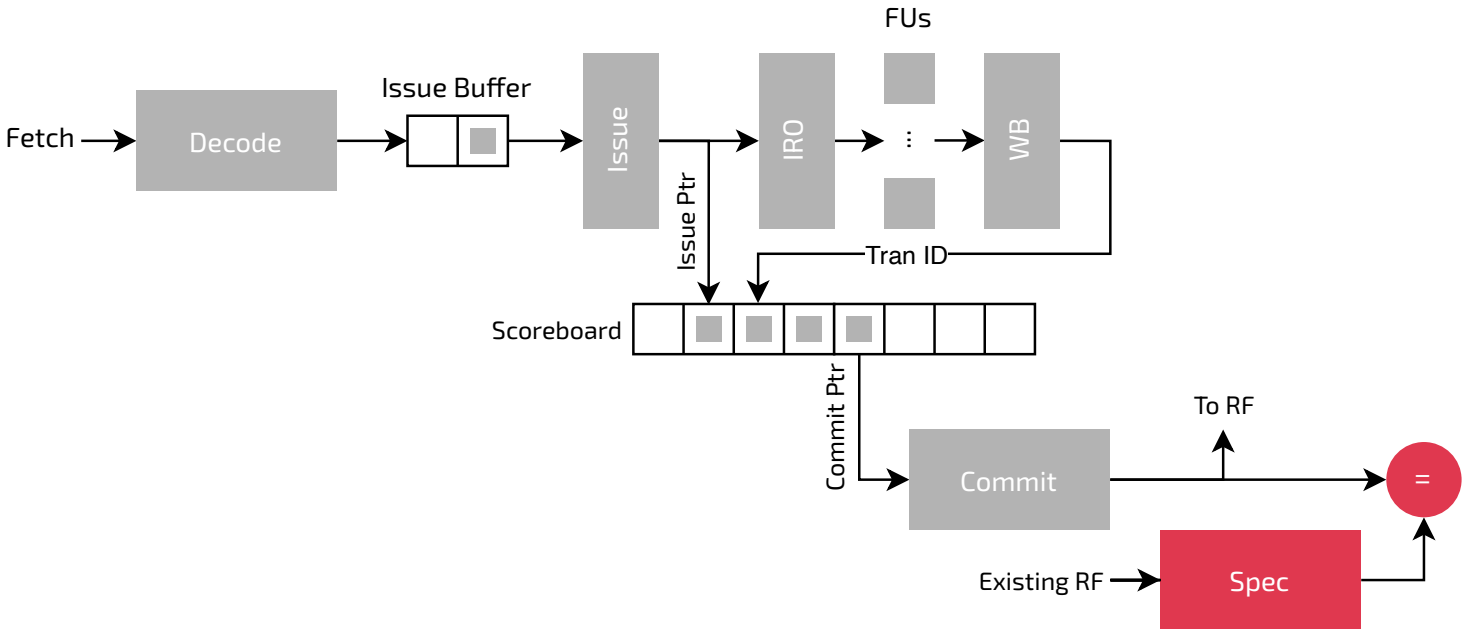
The lifetime of an instruction



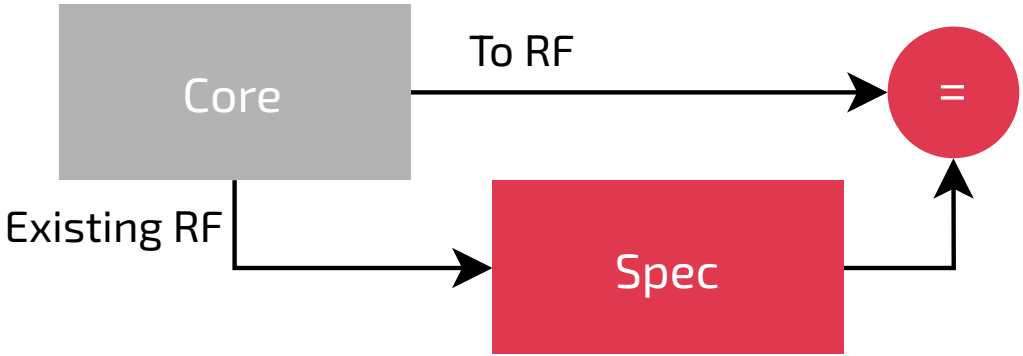
The lifetime of an instruction



Property insertion



Property insertion



Property insertion

Property insertion

Want to prove:

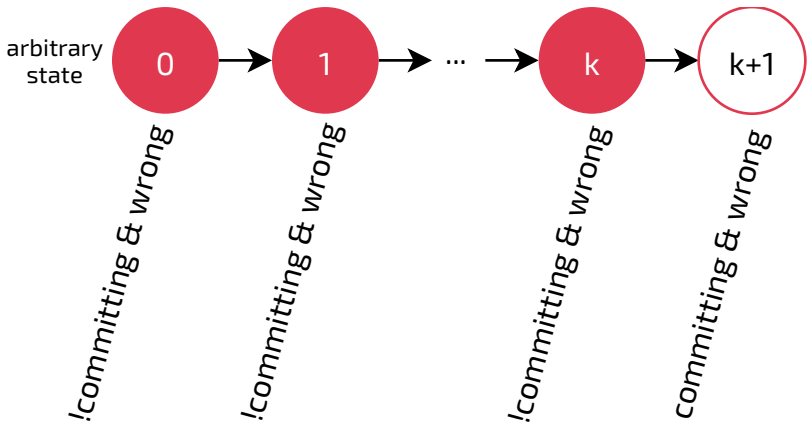
```
assert property (committing -> result == Spec(curr_reg_file))
```

Property insertion

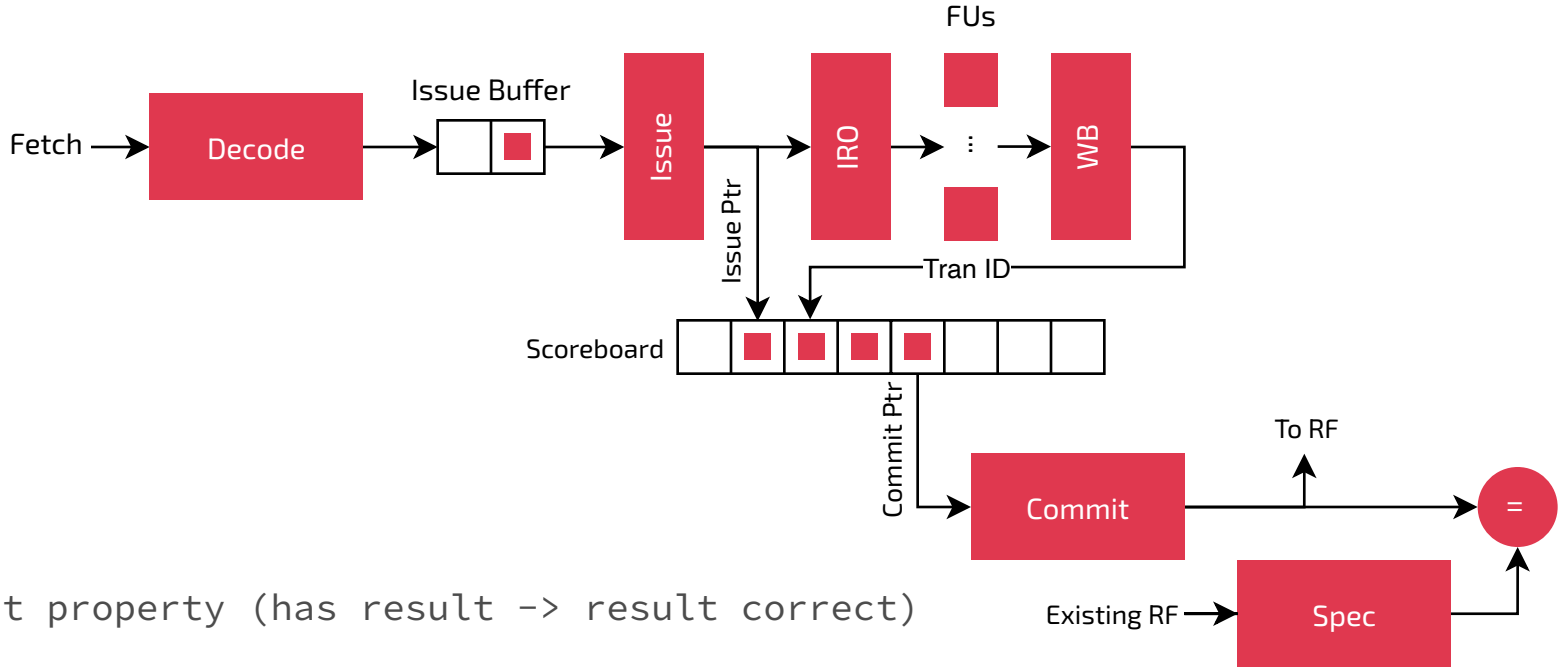
Want to prove:

```
assert property (committing -> result == Spec(curr_reg_file))
```

Problem: Counter-example in k-induction for any k!



So, how do we prove that equality?



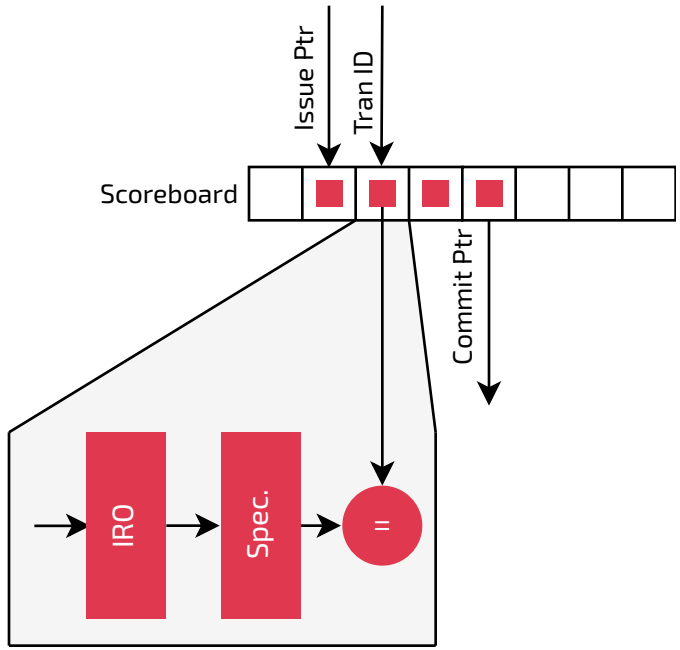

```
assert property (has result -> result correct)
```

How do we define ‘correct’ for intermediate results?

correct means:

```
result == Spec(inputs)
```

Too big, and too slow to elaborate!

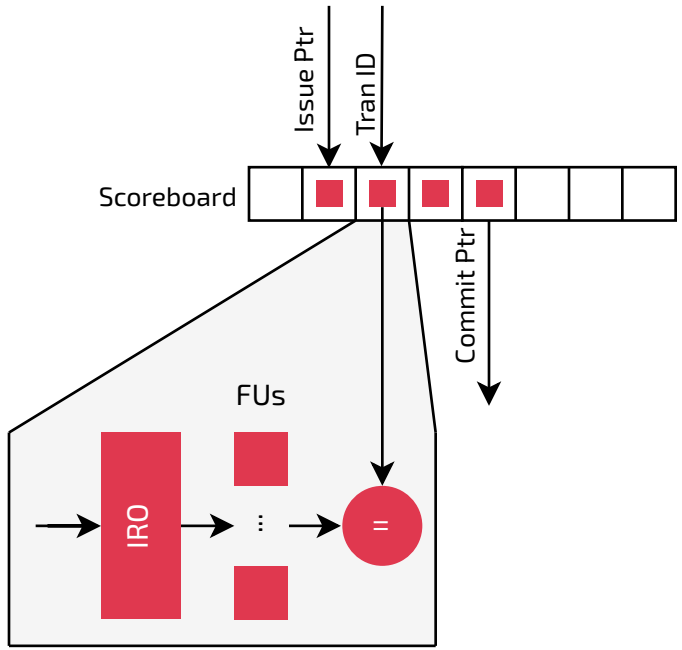


How do we define ‘correct’ for intermediate results?

correct means:

```
result == FUs(inputs)
```

(i.e. clone the execute stage)



Dealing with the inputs

Dealing with the inputs

RS1/RS2:

Derive from instructions ahead (forwarding), or from the RF.

(i.e. clone IRO)

Dealing with the inputs

RS1/RS2:

Derive from instructions ahead (forwarding), or from the RF.

(i.e. clone IRO)

Decoded Instruction:

assert property (decoded metadata == decoder(instr bits))

(i.e. clone decode)

A Methodology is Begging to Emerge

A Methodology is Begging to Emerge

- We saw this in Ibex too!

A Methodology is Begging to Emerge

- We saw this in Ibex too!
- Always trying to flatten time, to create a SAT problem, where the things we're comparing are just different functions of the same inputs

A Methodology is Begging to Emerge

- We saw this in Ibex too!
- Always trying to flatten time, to create a SAT problem, where the things we're comparing are just different functions of the same inputs
- Essentially doing some kind of backward looking symbolic execution

Capability DTIs

Constrains what a valid capability can be.

Capability DTIs

Constrains what a valid capability can be.

e.g.

Capability DTIs

Constrains what a valid capability can be.

e.g.

- Tagged capabilities have reserved bits 0

Capability DTIs

Constrains what a valid capability can be.

e.g.

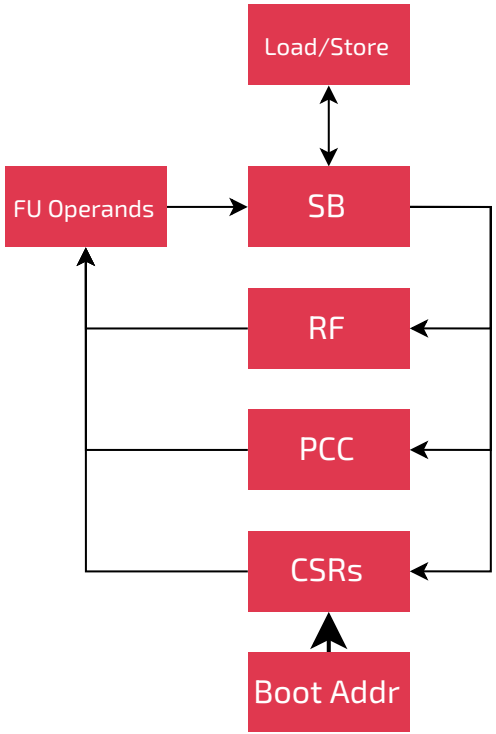
- Tagged capabilities have reserved bits 0
- All capabilities with IMPLIED_EXP have the implied EXP

Capability DTIs

Constrains what a valid capability can be.

e.g.

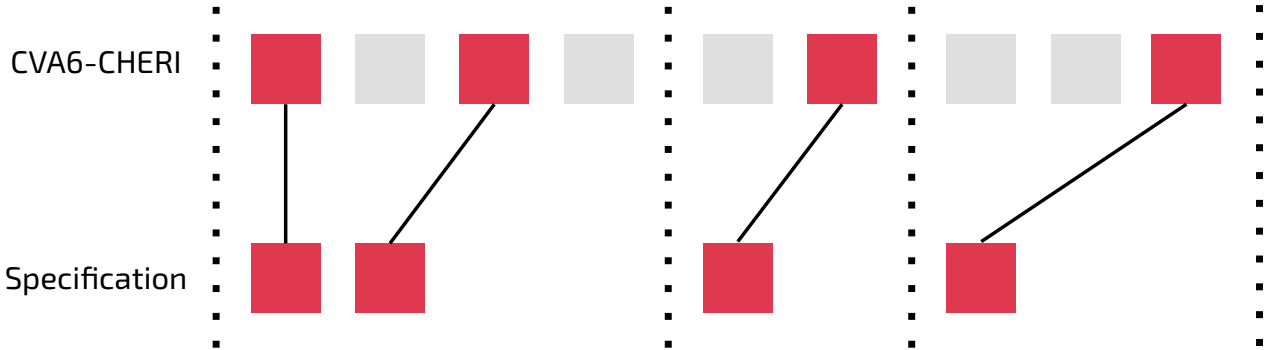
- Tagged capabilities have reserved bits 0
- All capabilities with IMPLIED_EXP have the implied EXP



Observational Equivalence

Observational Equivalence

Aim is eventually to show trace equivalence of memory outputs with spec



Conclusions

Conclusions

- In the process of showing CVA6-CHERI is correct

Conclusions

- In the process of showing CVA6-CHERI is correct
  - Good progress made, more progress to come

Conclusions

- In the process of showing CVA6-CHERI is correct
  - Good progress made, more progress to come
- High level of confidence in the parts of the design we formally verify

Conclusions

- In the process of showing CVA6-CHERI is correct
  - Good progress made, more progress to come
- High level of confidence in the parts of the design we formally verify
  + High confidence from DV in the areas we don't

Thanks

Thanks

- Thanks for listening!

Thanks

- Thanks for listening!
- Thanks to everyone at lowRISC, and to Tom and Brandon from OUI

Thanks

- Thanks for listening!
- Thanks to everyone at lowRISC, and to Tom and Brandon from OUI
- Thanks to Amy and everyone else who supported this work!