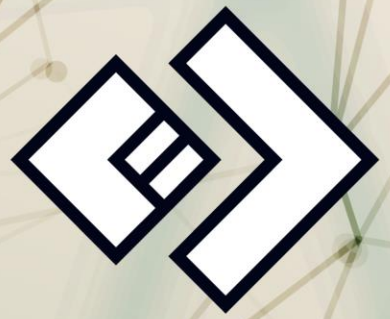


LUBIS EDA

Bringing CI Into Formal Verification

Dr. Tobias Ludwig, CEO

Semiconductor Family Feud

We have asked 100 hardware engineers: “What's notoriously hard to verify?”

Rank	IP Block
1	CPU / Processor Cores
2	DDR / DRAM Controllers
3	PCIe / High-Speed I/O
4	GPU / Graphics Units
5	Cache Coherency Units
6	AI/ML Accelerators
7	Network Interface (Ethernet, etc.)
8	Security / Crypto Engines
9	SoC Interconnect / NoC
10	Power Management Units (PMU)

What makes verifying caches so difficult?

Hazards Resolution

Eviction

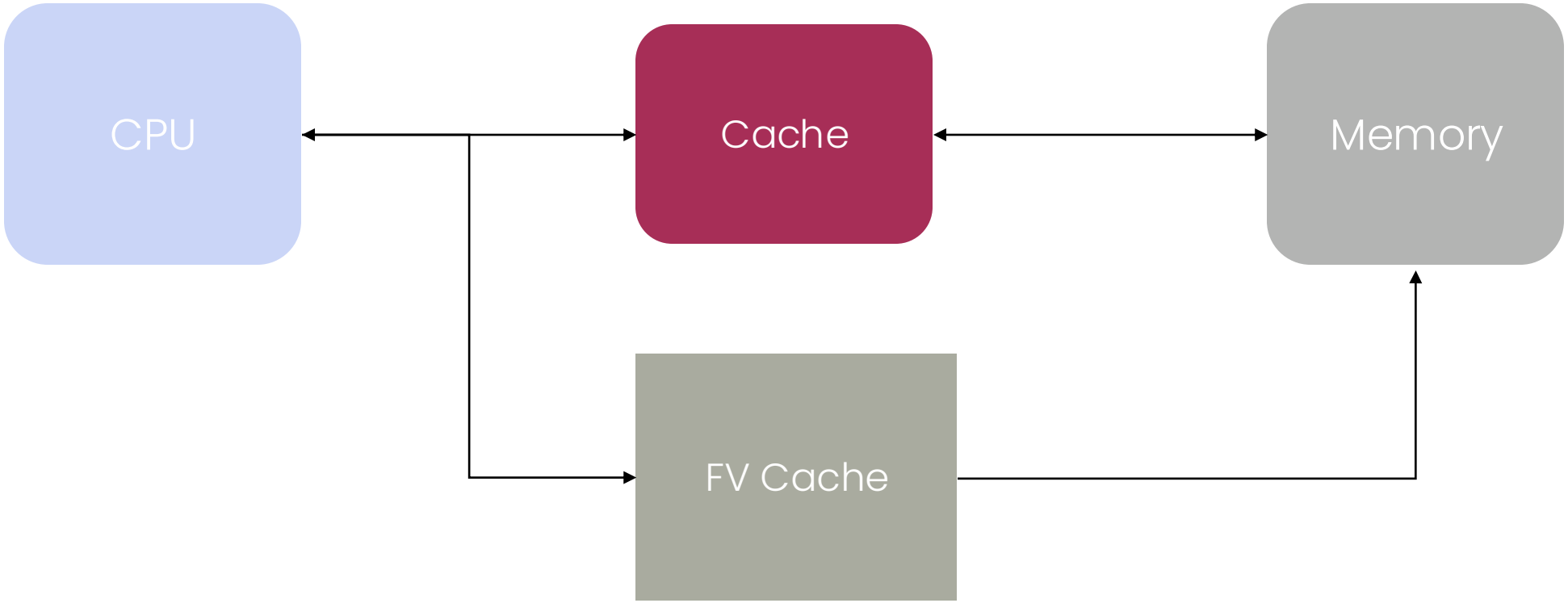
Out-of-order responses

What do we do in formal to get it done?

The setup

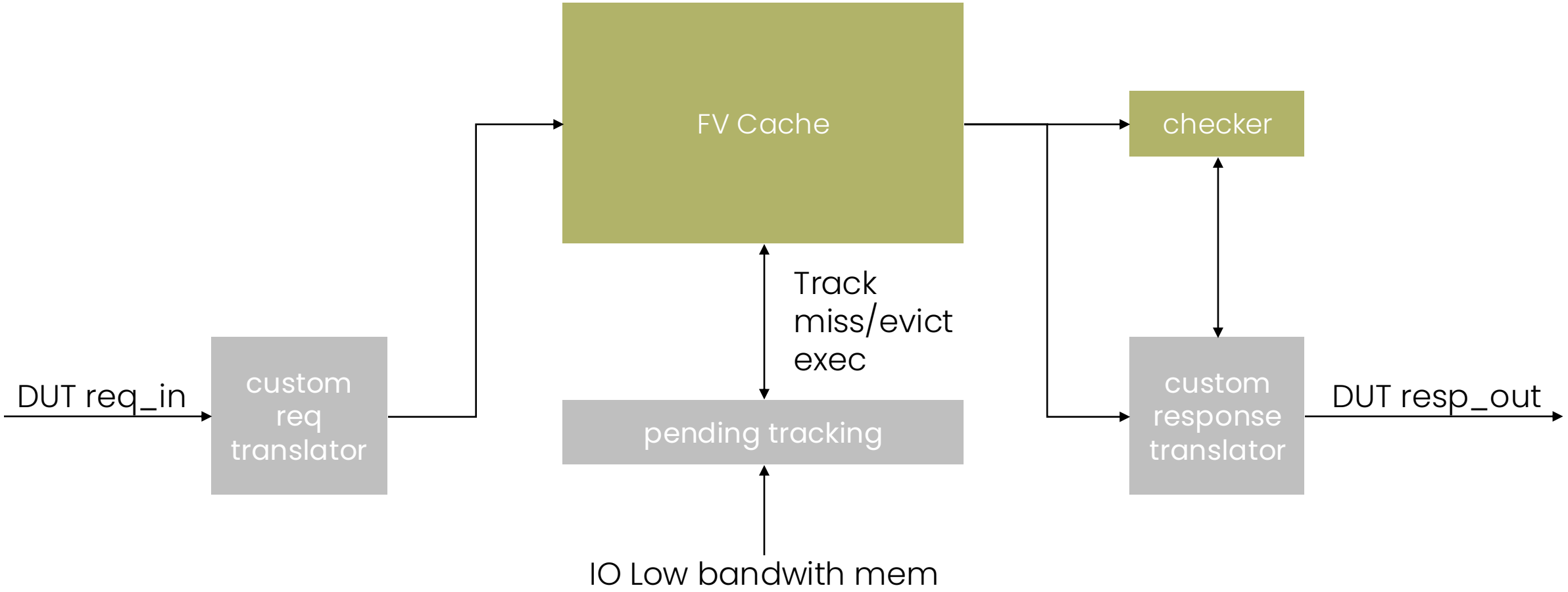
Most important property of a cache?

read |-> right data

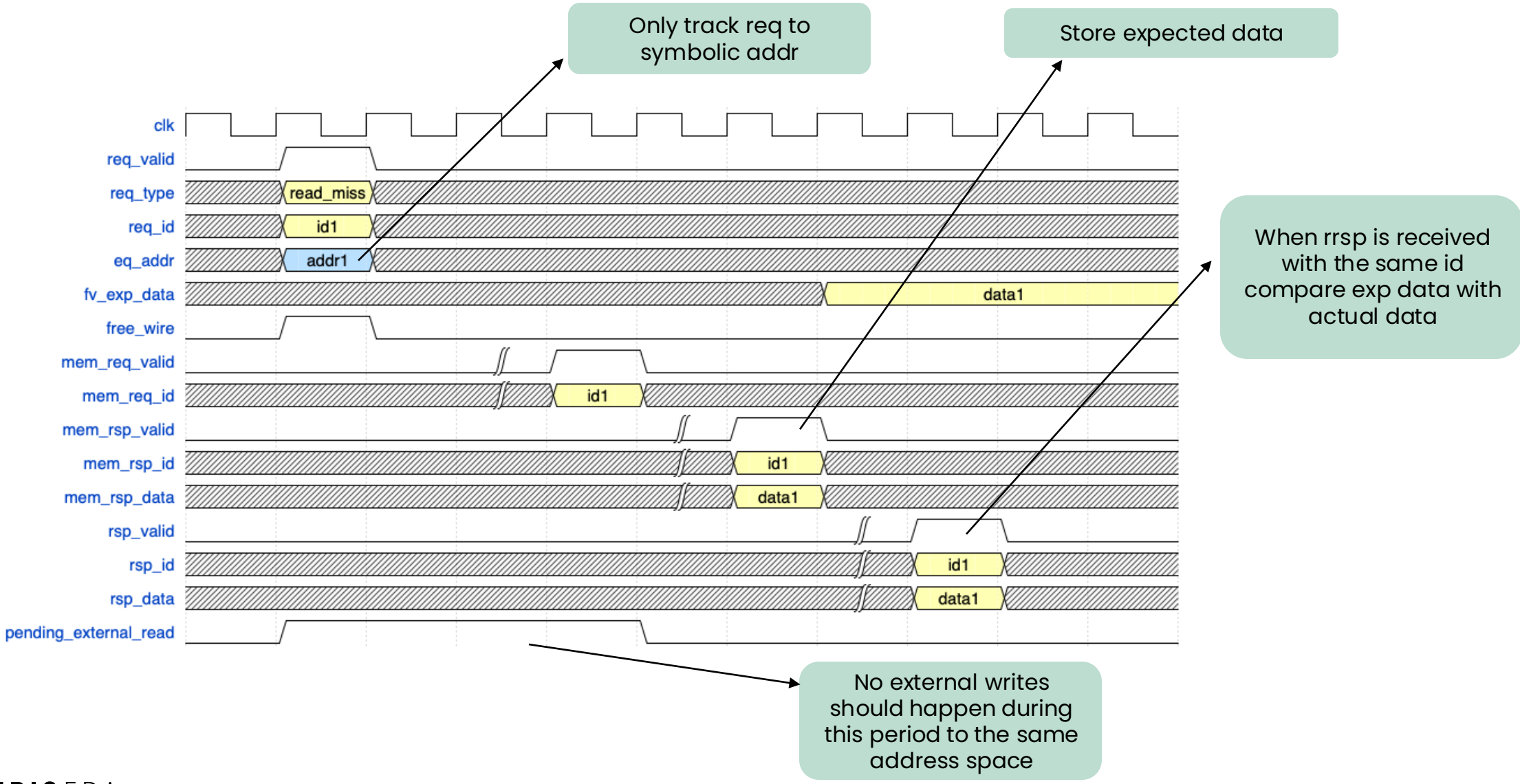


Tracks "chosen" addr

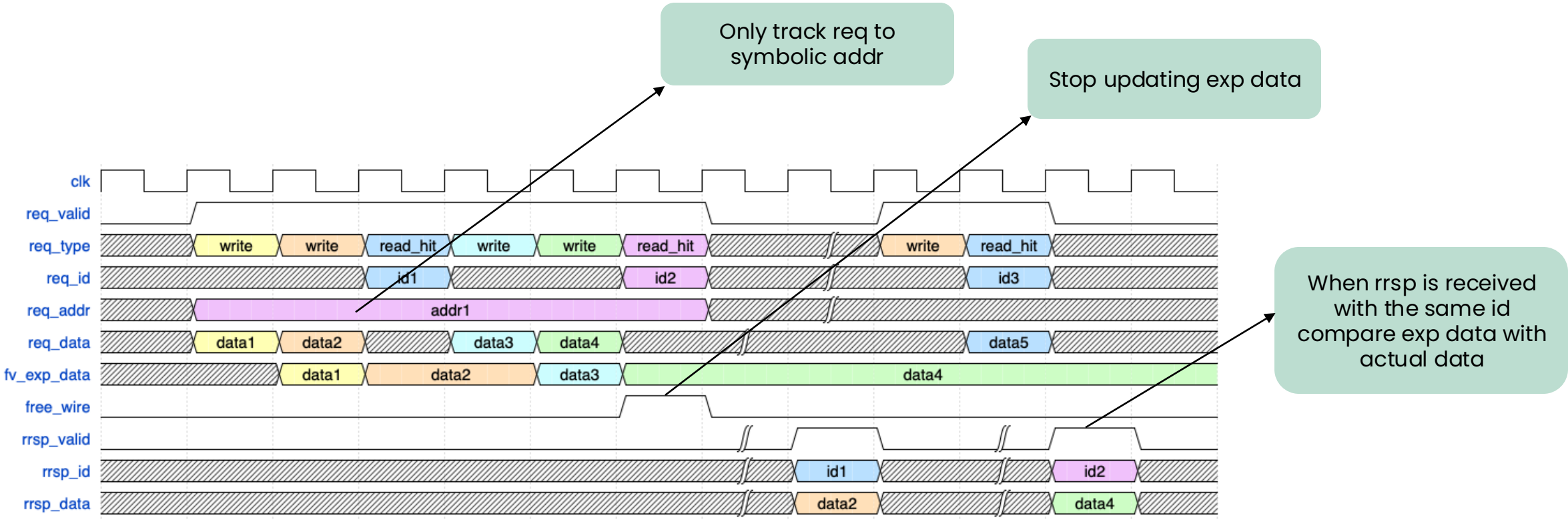
Zooming into the FV cache



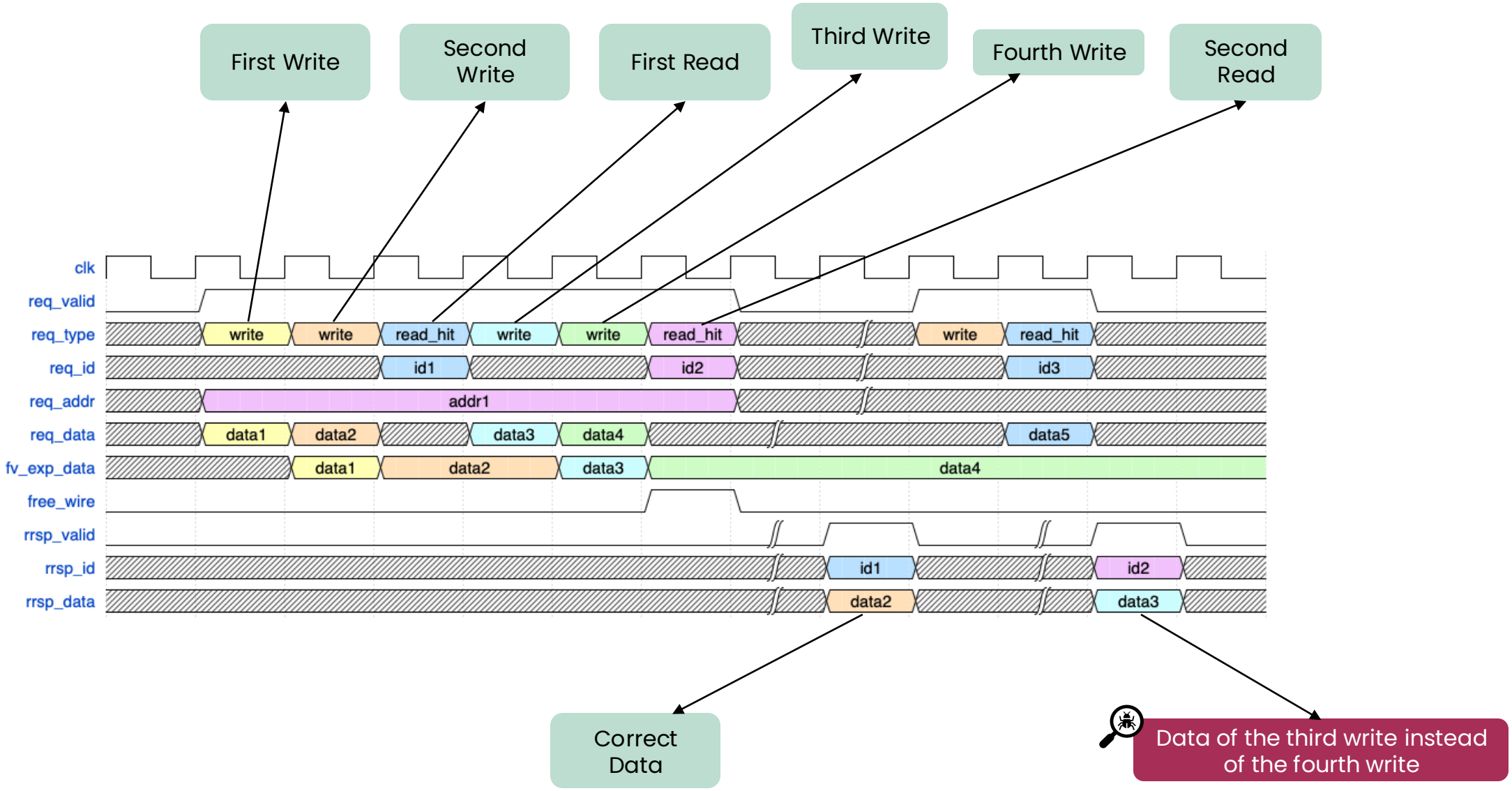
Cache Miss Case



Cache Hit Case



Bug Example


First Write
Second Write
First Read
Third Write
Fourth Write
Second Read
clk
req_valid
req_type
req_id
req_addr
req_data
fv_exp_data
free_wire
rrsp_valid
rrsp_id
rrsp_data
write
write
read_hit
write
write
read_hit
write
read_hit
id1
id2
id3
addr1
data1
data2
data3
data4
data5
data1
data2
data3
data4
id1
id2
data2
data3
Correct Data
Data of the third write instead of the fourth write

What makes verifying caches so difficult?

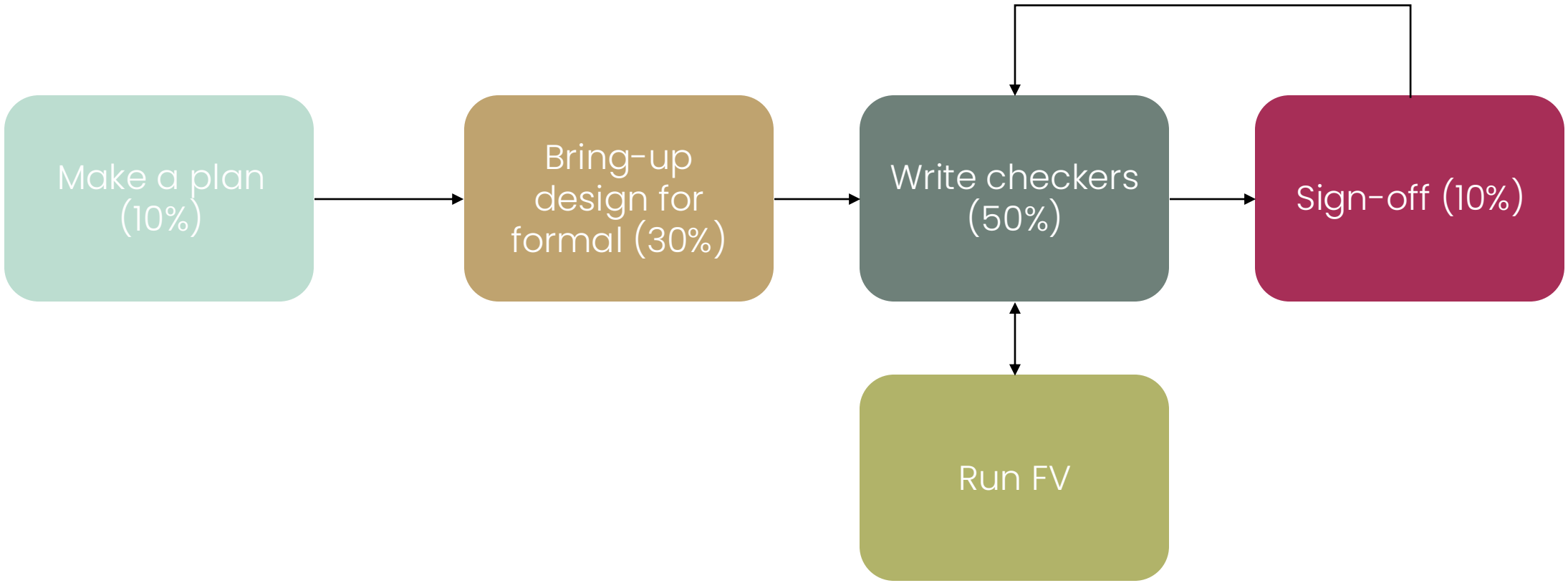
- ☑ Hazards Resolution
- ☑ Eviction
- ☑ Out-of-order responses

Sounds good ?

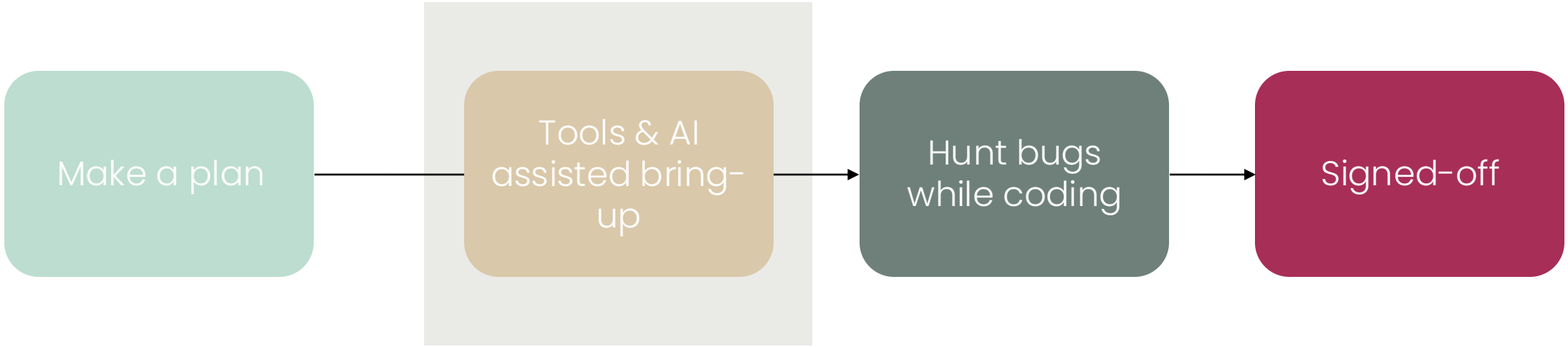
Let's see how we can get this done ASAP

LUBISEDA

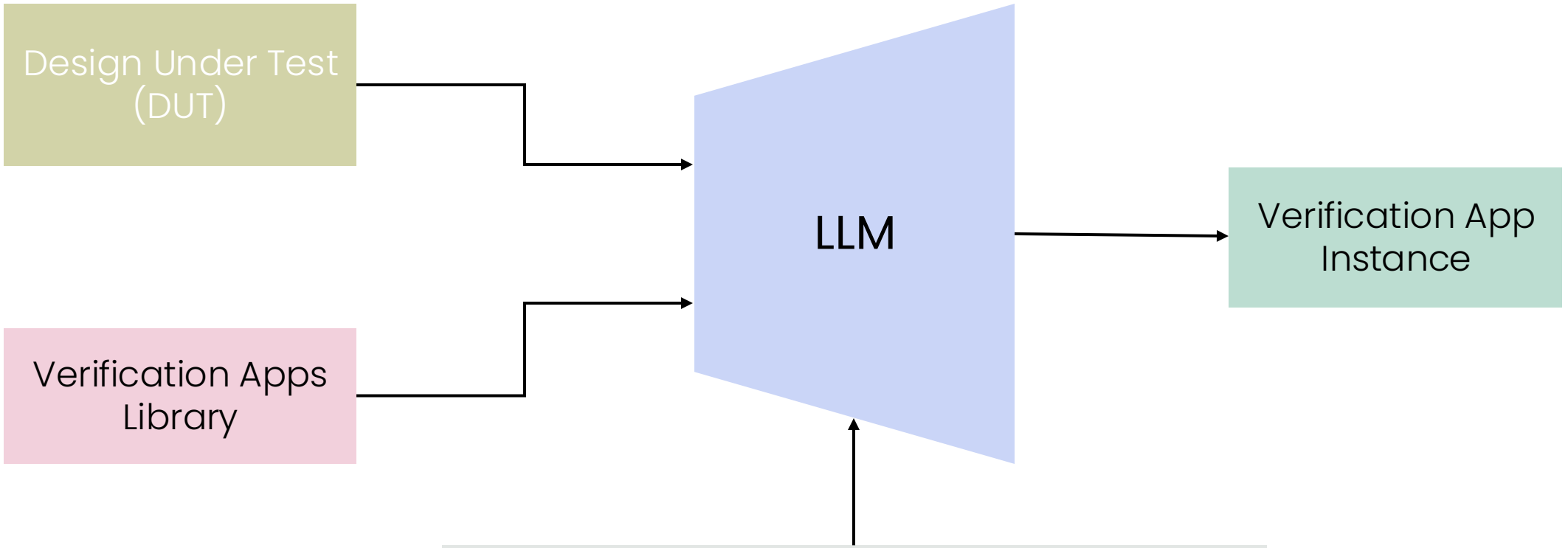
The Original Flow



Make it less tedious step #1



AI assisted setup



AI assisted setup

Ambiguous DUT

```
module design_x (
    input logic clk,
    input logic [1:0] a,
    output logic b
);

    logic [1:0] x = 2'b00;

    always_ff @(posedge clk) begin
      if (a != 2'b00) begin
        x <= x + 1'b1;
        if (x == 2'b10)
          x <= 2'b00;
        end
    end

    assign b = (a & (x == 2'b00)) | ((a >> 1) & (x == 2'b01));

endmodule
```

What does it do?

Verification Apps Library

Large Language Model (LLM)

Smart Prompt

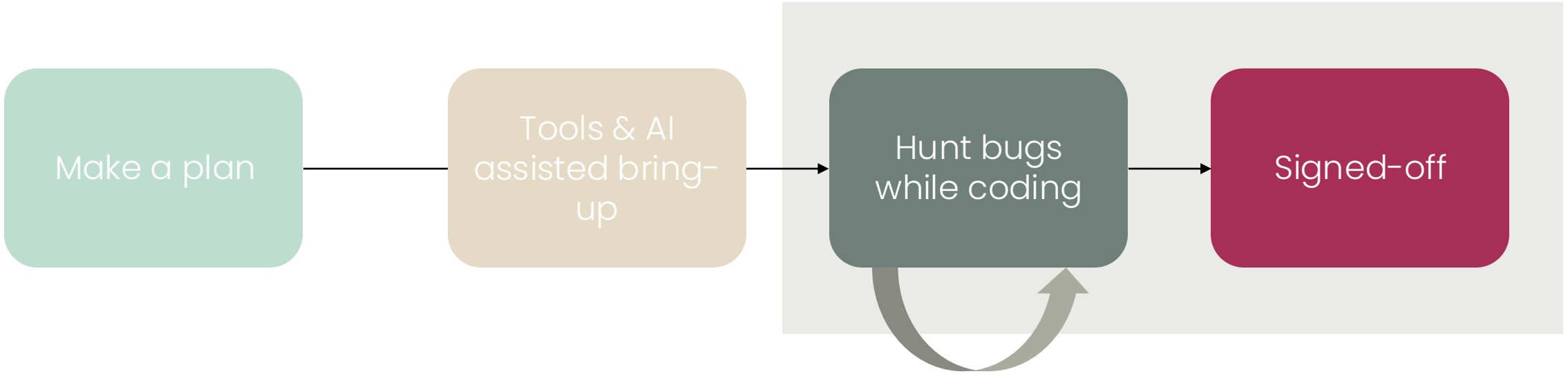
Verification Wrapper

```
module wrapper
#(
    parameter num_reqs = 2
)(
    input clk,
    input a,
    output b
);

    fv_lubis_rr_arbiter_vapp
    #(
        .num_reqs(num_reqs)
    ) fv_lubis_rr_arbiter_vapp_inst
    (
        .clk(clk),
        .reqs(a),
        .grant(b)
    )

    design_x design_x_inst
    (
        .clk(clk),
        .a(a),
        .b(b)
    )
endmodule
```

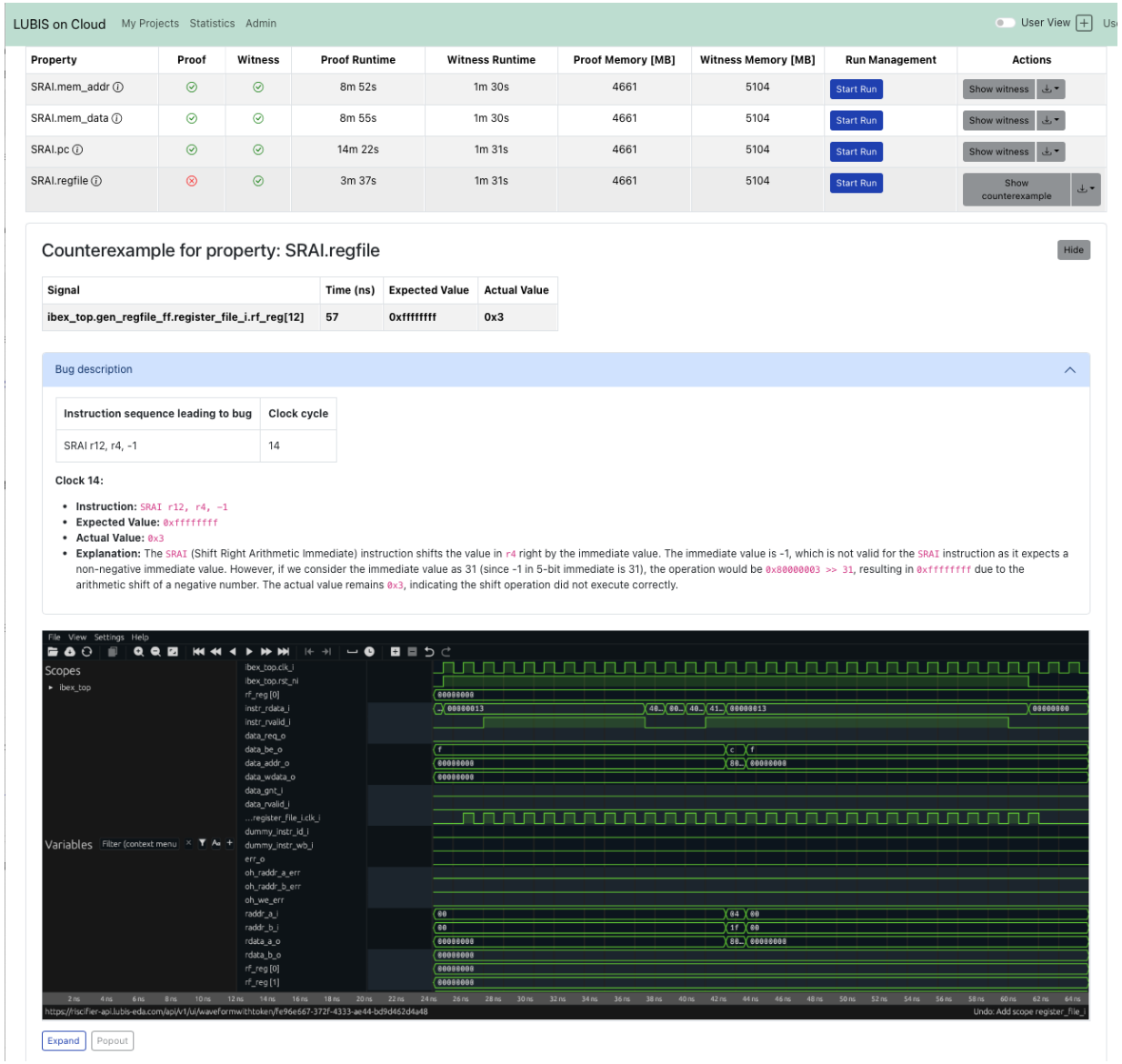
Make it less tedious step #2



In the loop sign-off:

- Incremental formal
- Incremental coverage
- AI assisted-debug

Its on the web



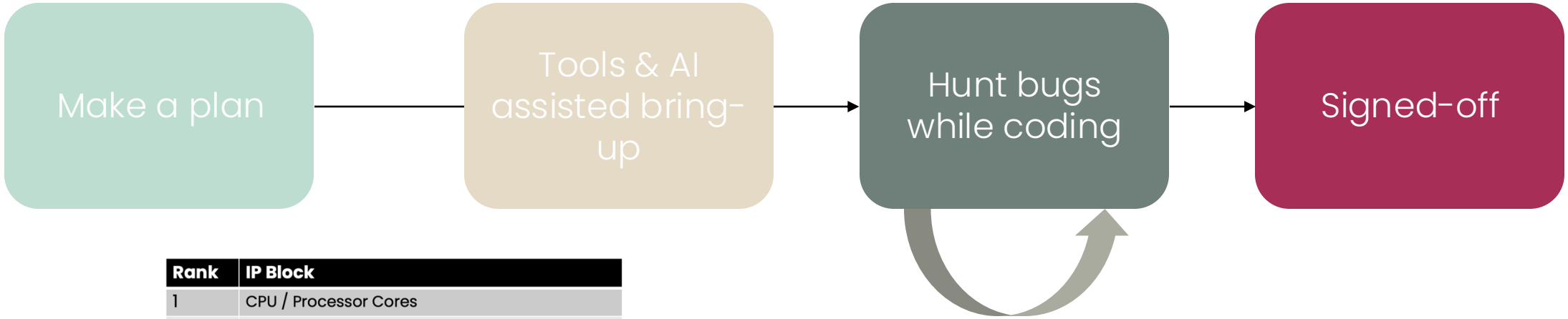
Web-based control

AI generated debug information

Web-based debugging

LUBIS EDA

The Original Flow



Rank	IP Block
1	CPU / Processor Cores
2	DDR / DRAM Controllers
3	PCIe / High-Speed I/O
4	GPU / Graphics Units
5	Cache Coherency Units
6	AI/ML Accelerators
7	Network Interface (Ethernet, etc.)
8	Security / Crypto Engines
9	SoC Interconnect / NoC
10	Power Management Units (PMU)

In the loop checking:

- Incremental formal
- Incremental coverage
- AI assisted-debug

Time for a conclusion

LUBISEDA

LUBIS Formal EDA Ecosystem

From an idea to tape-out

SPARK

Kick start your formal verification efforts

VERIFICATION IPS

PROPERTY GENERATOR

RECHECK

Manage and Iterate through formal regressions

LUBIS EDA

Fast. Reliable. Bug-free.

We help chip teams build confidence in their designs by combining expert formal verification services with automation that scales.

From finding hidden bugs to verifying complex systems.

Team 35+

Kaiserslautern, Germany

We love formal

Tobias Ludwig

CEO

Email: Tobias.Ludwig@lubis-eda.com

Web: www.lubis-eda.com