



LUBISEDA

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

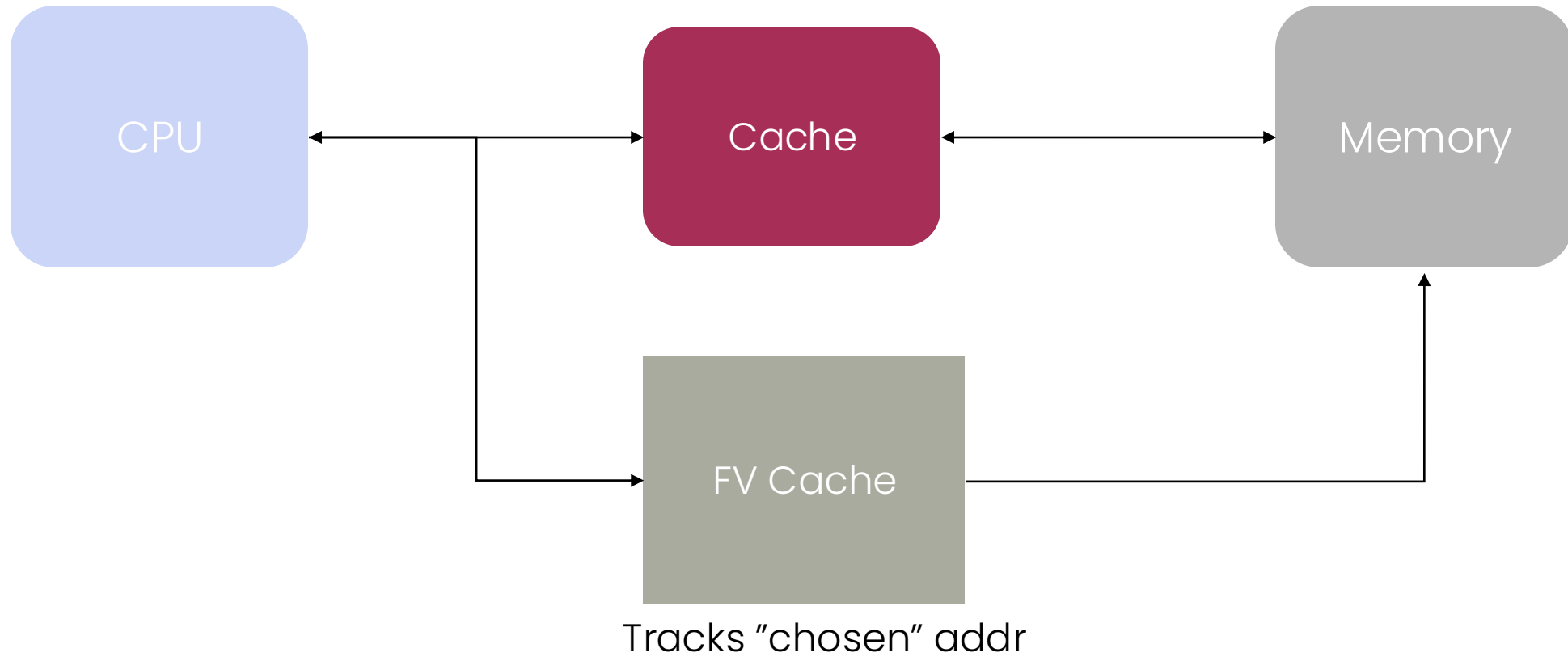
Out-of-order
responses

What do we do in formal to get it done?

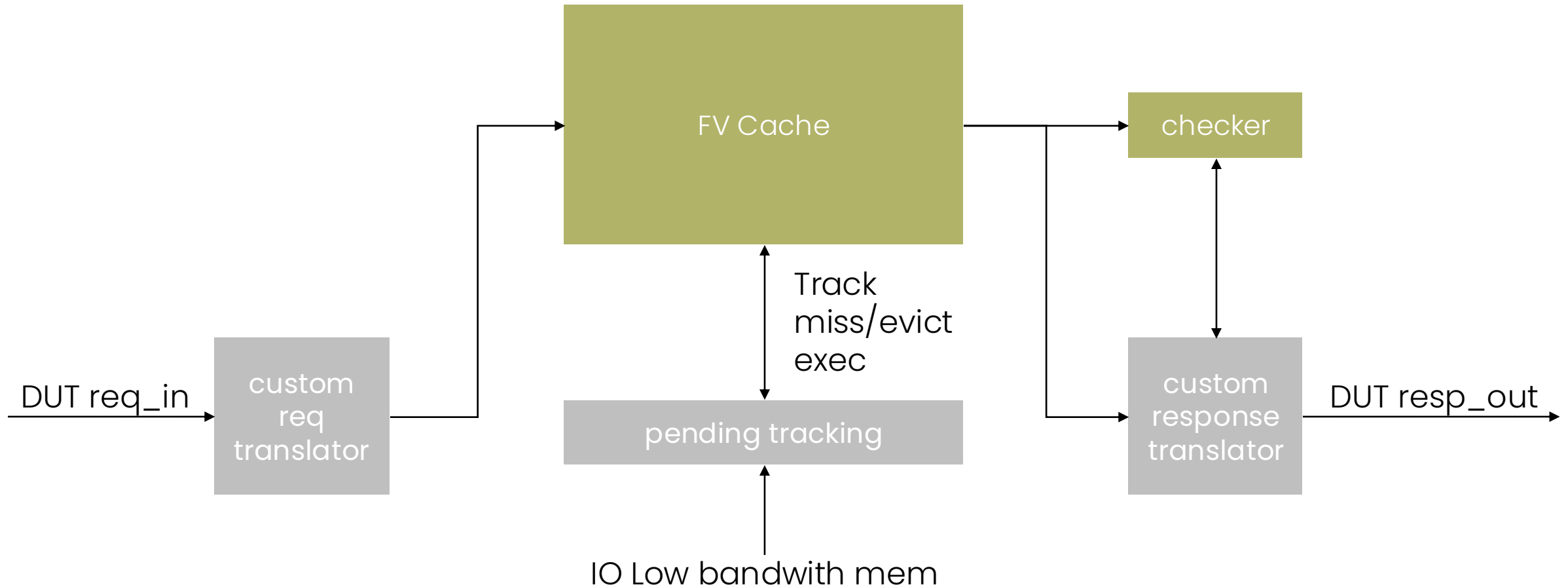
The setup

Most important property of a cache?

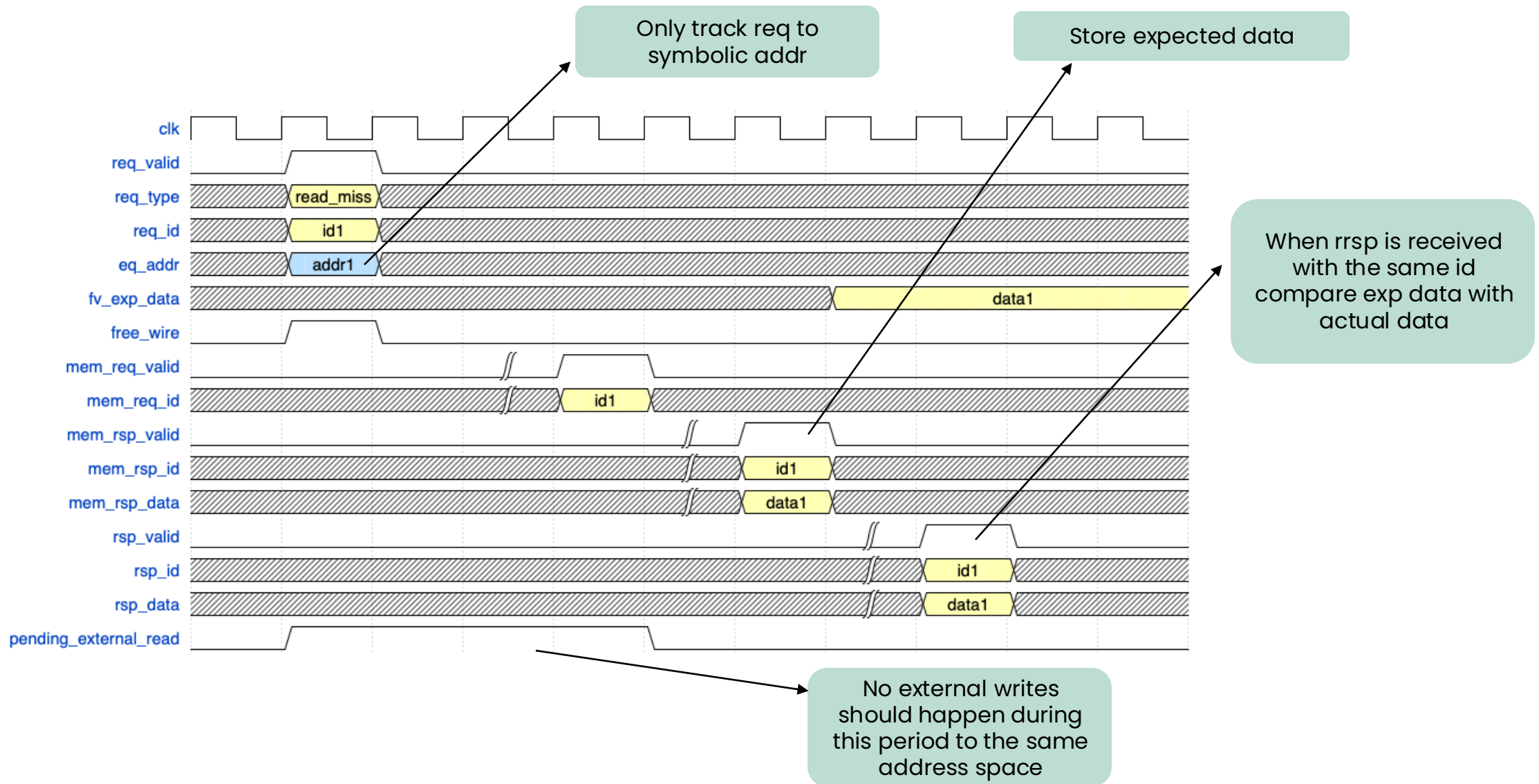
read $\rightarrow$ right data



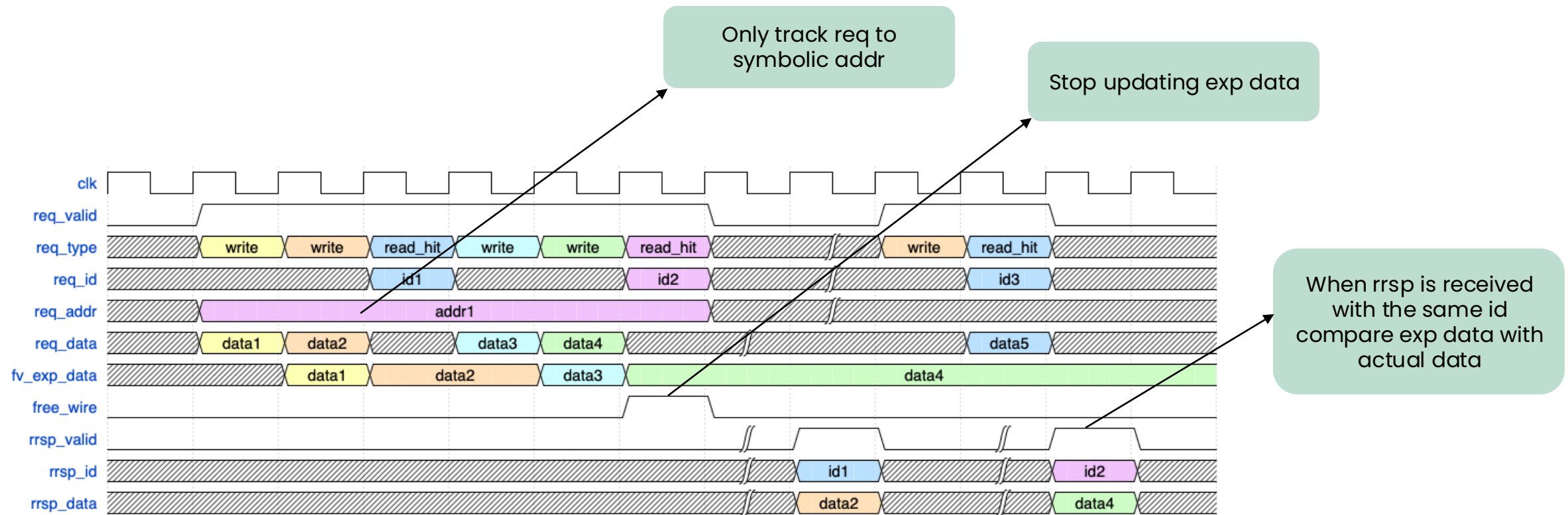
Zooming into the FV cache



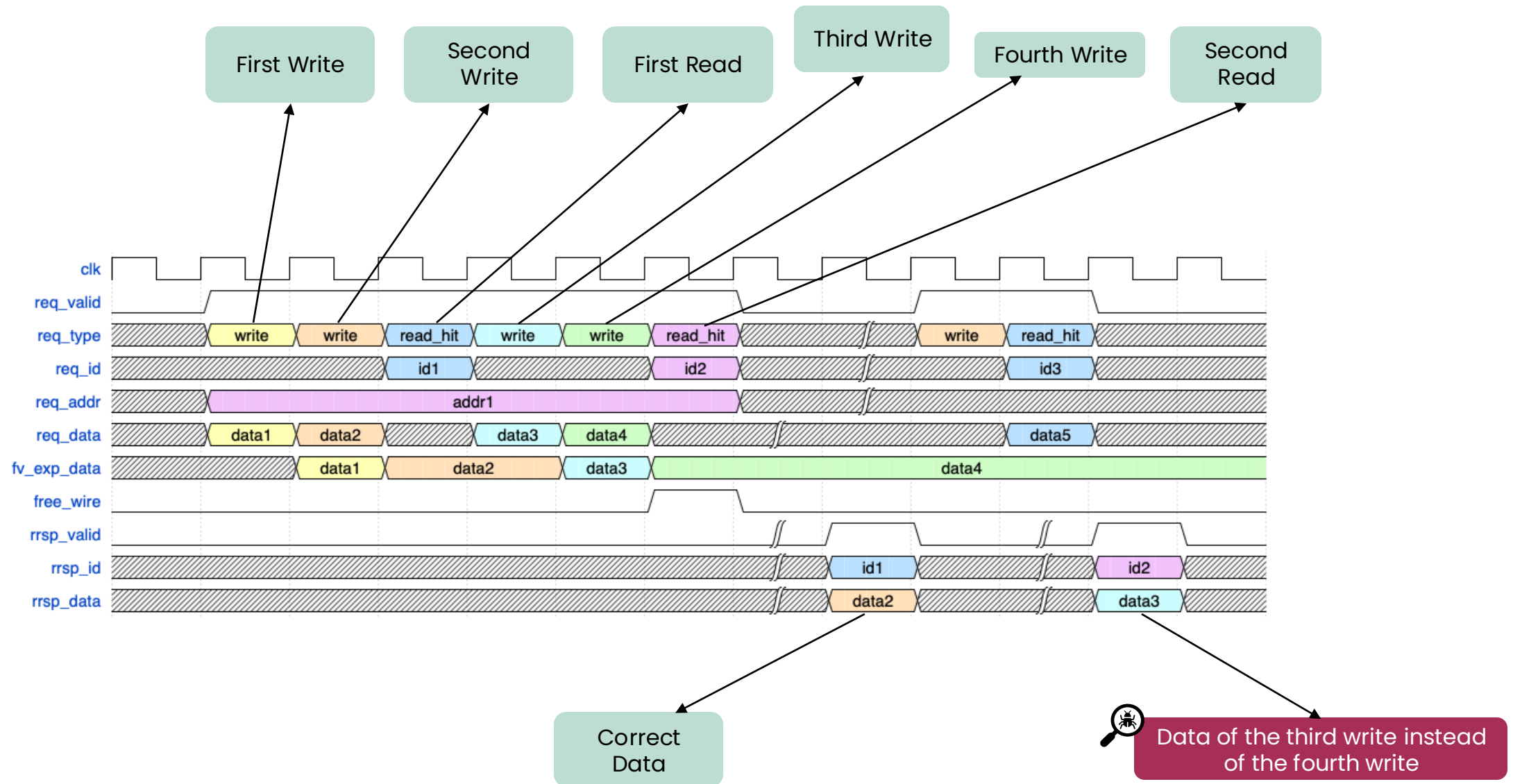
Cache Miss Case



Cache Hit Case



Bug Example



What makes verifying caches so difficult?



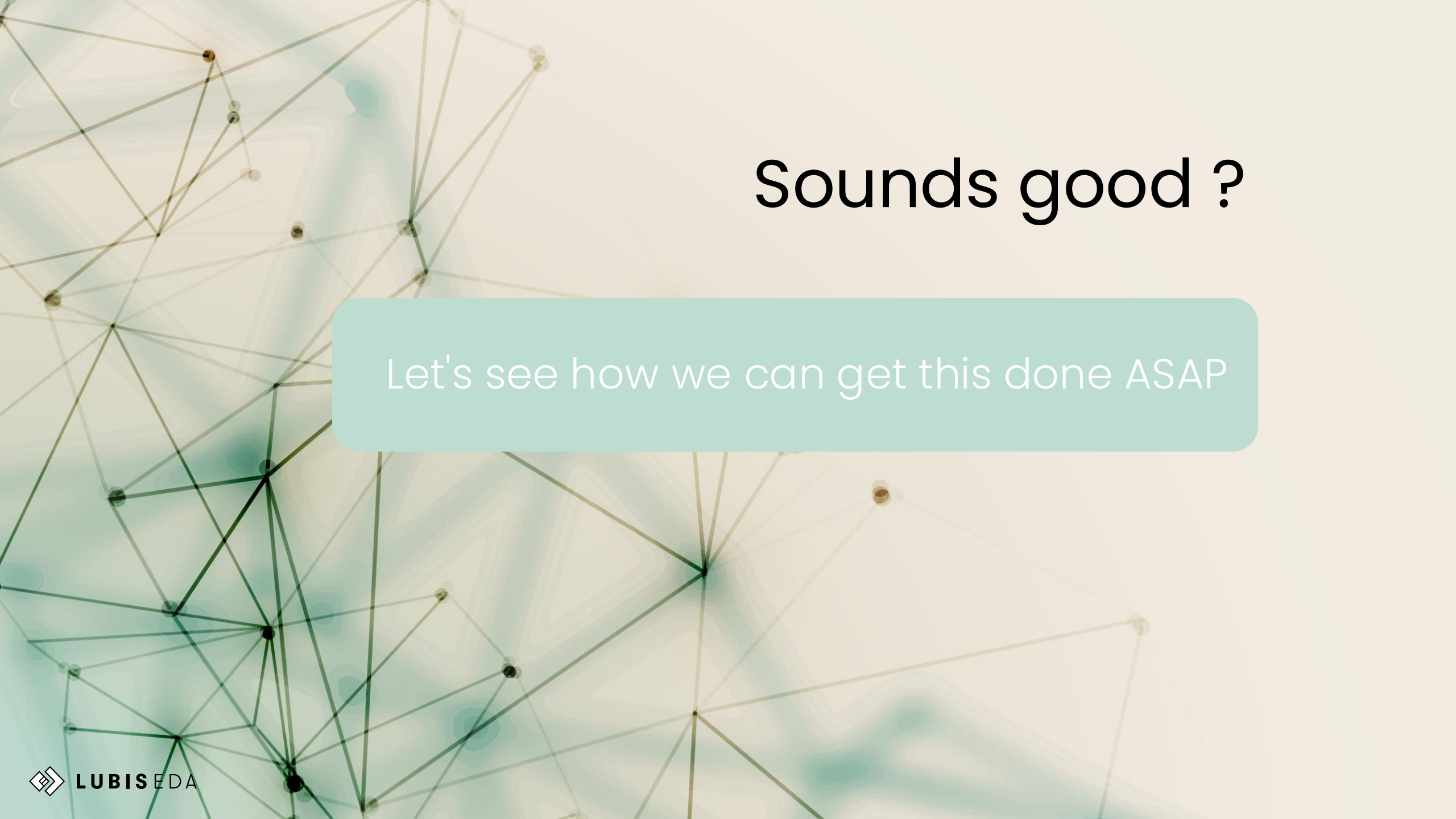
Hazards Resolution



Eviction



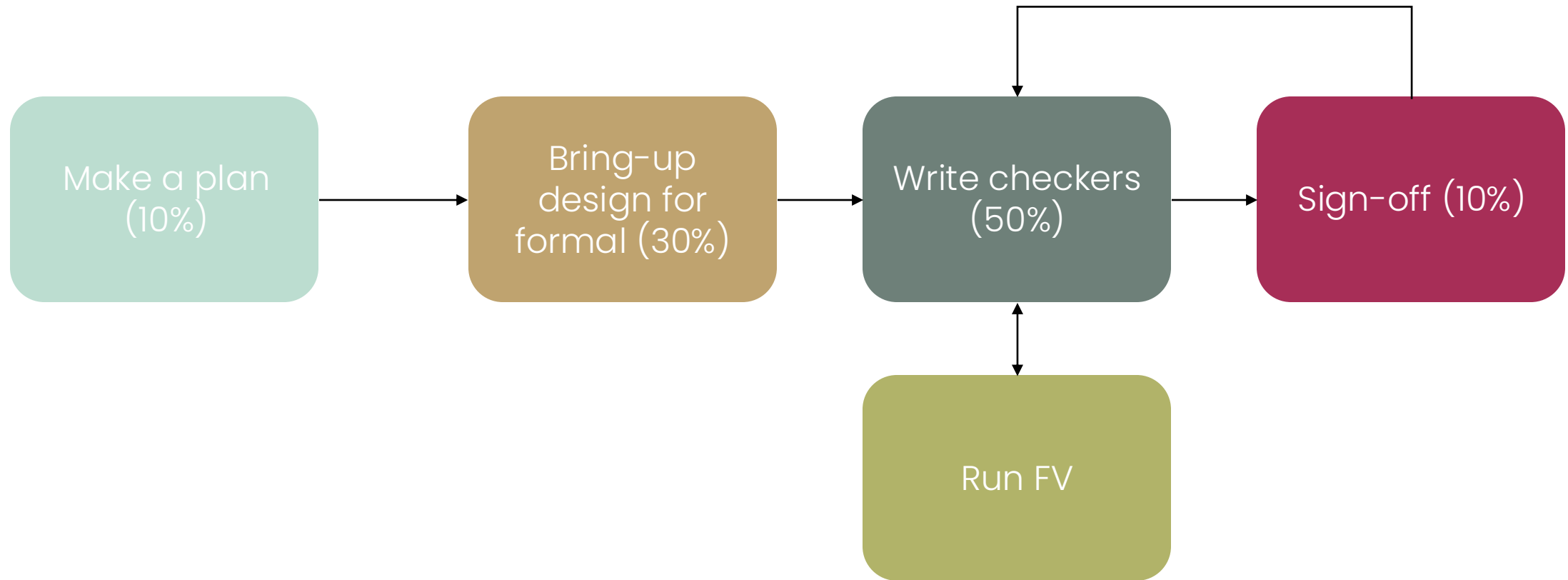
Out-of-order
responses



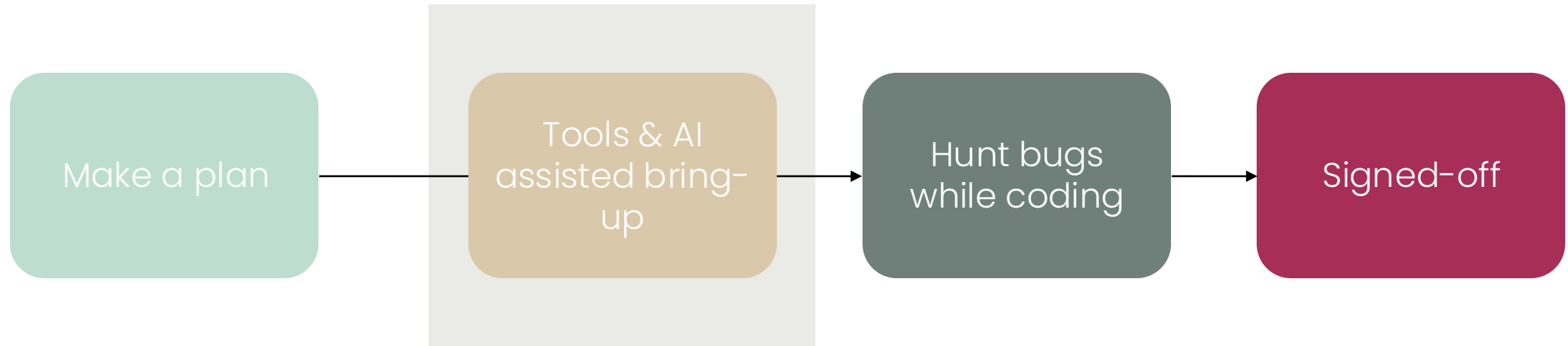
Sounds good ?

Let's see how we can get this done ASAP

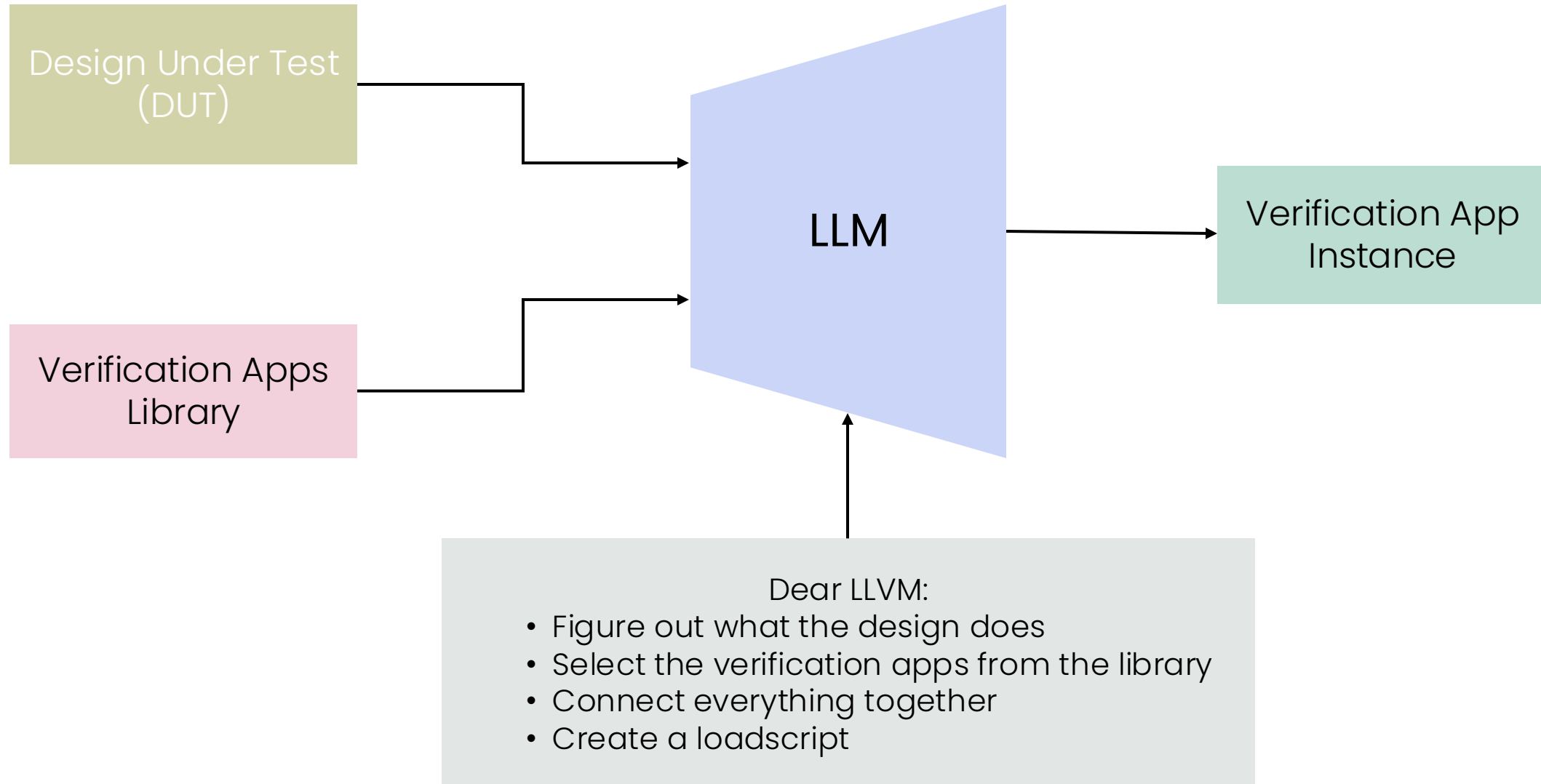
The Original Flow



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));  
    end  
endmodule
```

What does it do?

Large Language
Model
(LLM)

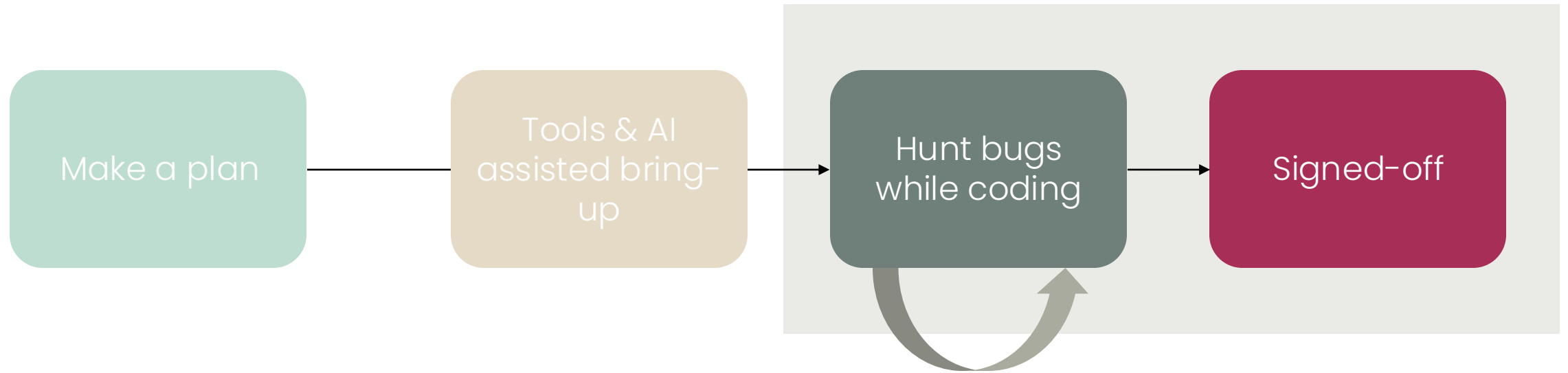
Verification Apps
Library

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

LUBIS on CloudMy ProjectsStatisticsAdmin

User View

Property	Proof	Witness	Proof Runtime	Witness Runtime	Proof Memory [MB]	Witness Memory [MB]	Run Management	Actions
SRAI.mem_addr	✓	✓	8m 52s	1m 30s	4661	5104	Start Run	Show witness
SRAI.mem_data	✓	✓	8m 55s	1m 30s	4661	5104	Start Run	Show witness
SRAI.pc	✓	✓	14m 22s	1m 31s	4661	5104	Start Run	Show witness
SRAI.regfile	✗	✓	3m 37s	1m 31s	4661	5104	Start Run	Show counterexample

Counterexample for property: SRAI.regfile

Signal	Time (ns)	Expected Value	Actual Value
ibex_top.gen_regfile_ff.register_file_i.rf_reg[12]	57	0xffffffff	0x3

Bug description

Instruction sequence leading to bug	Clock cycle
SRAI r12, r4, -1	14

Clock 14:

- Instruction: SRAI r12, r4, -1
- Expected Value: 0xffffffff
- Actual Value: 0x3
- Explanation: The SRAI (Shift Right Arithmetic Immediate) instruction shifts the value in r4 right by the immediate value. The immediate value is -1, which is not valid for the SRAI instruction as it expects a non-negative immediate value. However, if we consider the immediate value as 31 (since -1 in 5-bit immediate is 31), the operation would be 0x80000003 >> 31, resulting in 0xffffffff due to the arithmetic shift of a negative number. The actual value remains 0x3, indicating the shift operation did not execute correctly.

FileViewSettingsHelp

Scopes

ibex_top

Variables

2ns4ns6ns8ns10ns12ns14ns16ns18ns20ns22ns24ns26ns28ns30ns32ns34ns36ns38ns40ns42ns44ns46ns48ns50ns52ns54ns56ns58ns60ns62ns64ns

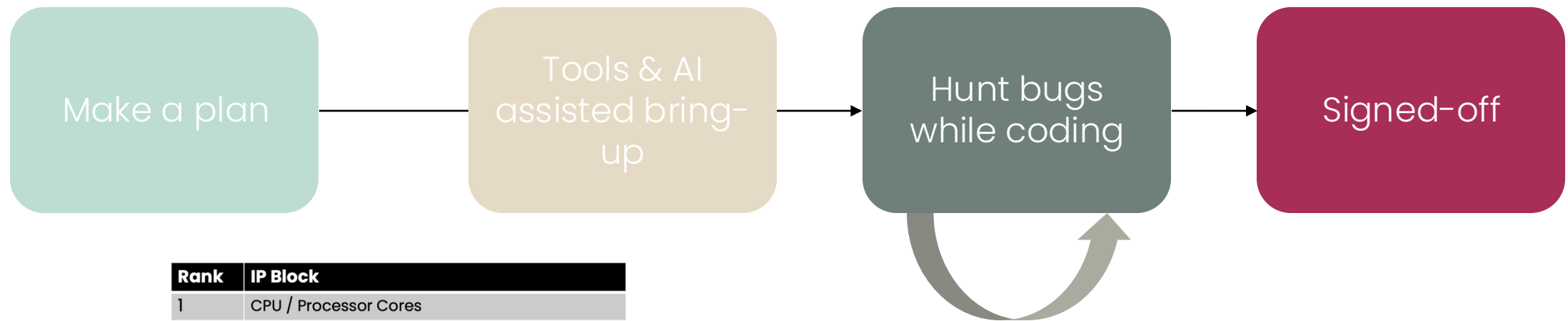
Undo: Add scope register_file_i

Web-based control

AI generated debug information

Web-based debugging

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

LUBIS Formal EDA Ecosystem

From an idea to tape-out



SPARK

Kick start your
formal verification
efforts



VERIFICATION IPS



PROPERTY GENERATOR

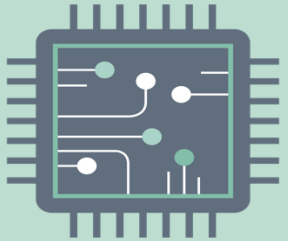


RECHECK

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

