# End to End Formal Verification of the CHERIoT-ibex Processor

PROFESSOR TOM MELHAM

# Why FV of RISC-V CHERI Processors?

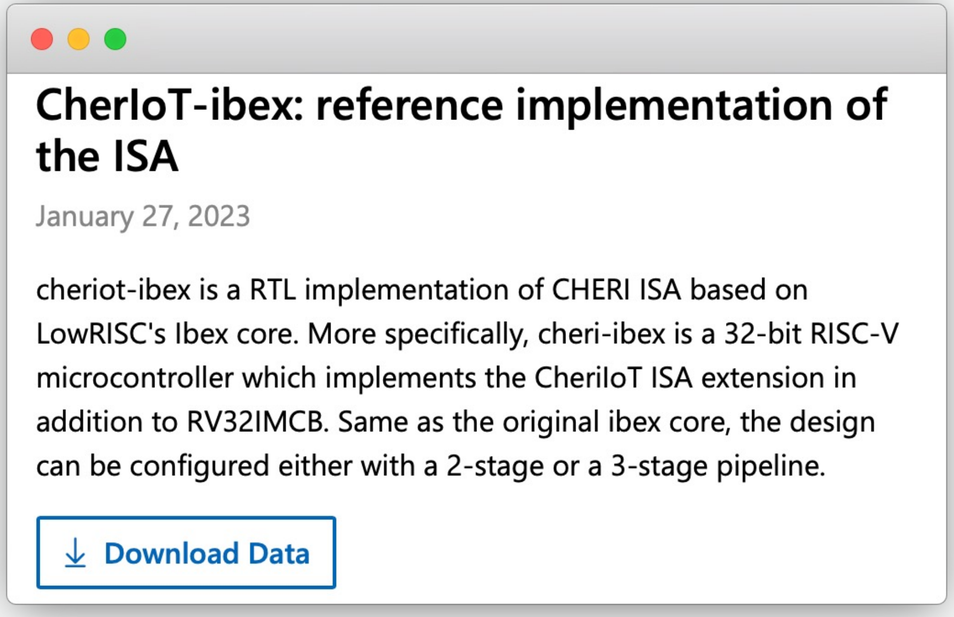
- Correctness of the processor implementation is the indispensable foundation of CHERI.
- Novelty of the ISA extensions and microarchitecture mean bugs, innovation.
- Full specification in Sail is available - we just have to get it into the FV tool.
- RISC-V is relatively tractable for formal. Full proofs, not bounded.

# FV of CHERIoT-ibex

- Research programme at Oxford to verify increasingly complex CHERI processors:
  - CHERI-RISC-V Flute (FMCAD 2021).
  - CHERIoT-ibex (ongoing).

- RTL implementation of a CHERI ISA based on LowRISC's open-source ibex core:
  - 32-bit RISC-V microcontroller: RV32IMCB.
  - 3 stage pipeline.
  - Implements the CHERIoT ISA extension.

- Comes with CHERIoT Sail
  - A complete formal specification of the ISA in Cambridge's Sail specification language.




https://www.microsoft.com/en-us/research/publication/cheriot-rethinking-security-for-low-cost-embedded-systems/

# FV of CHERIoT-ibex

- Research programme at Oxford to verify increasingly complex CHERI processors:
  - CHERI-RISC-V Flute (FMCAD 2021).
  - CHERIoT-ibex (ongoing).

- RTL implementation of a CHERI ISA based on LowRISC's open-source ibex core:
  - 32-bit RISC-V microcontroller: RV32IMCB.
  - 3 stage pipeline.
  - Implements the CHERIoT ISA extension.

- Comes with CHERIoT Sail
  - A complete formal specification of the ISA in Cambridge's Sail specification language.

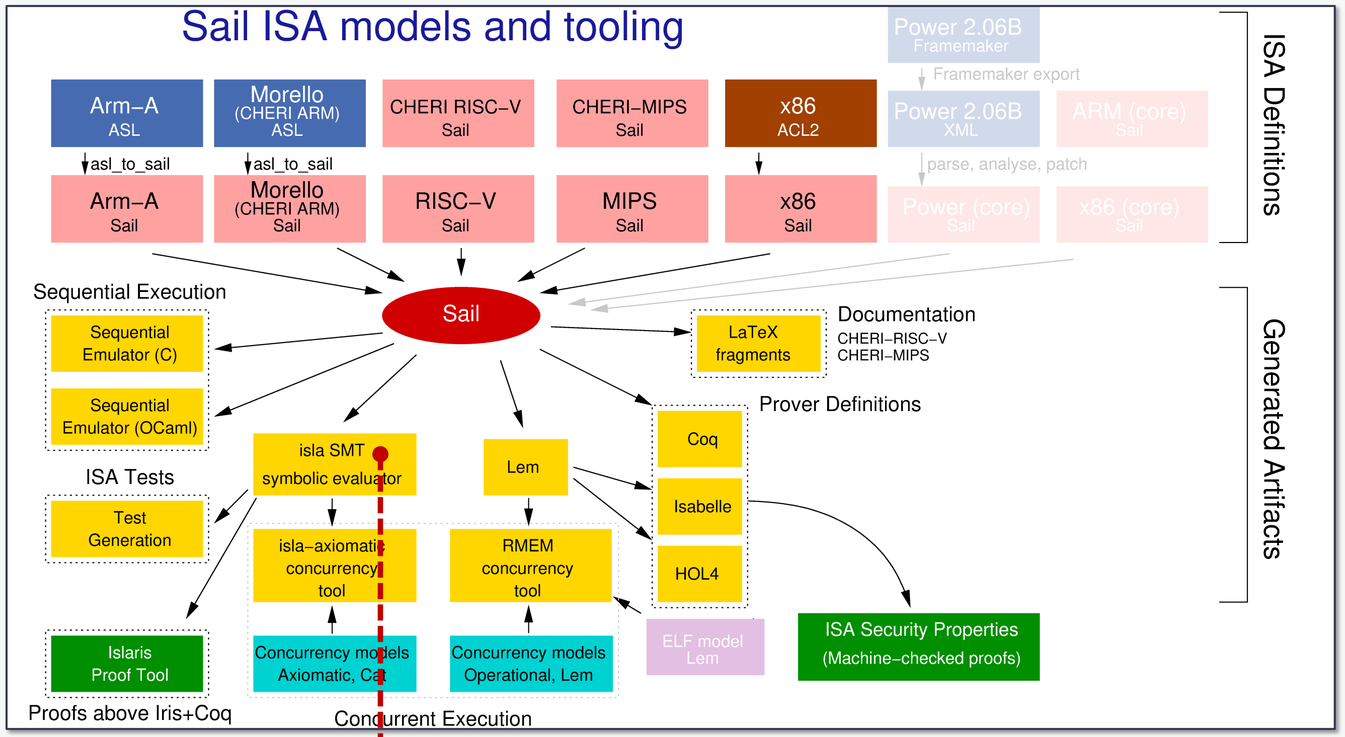
- Our principles:
  - Full coverage of entire ISA.
  - Cover both CHERI extensions and RISC-V part.
  - No FV-driven modifications to Sail spec.
  - Complete proofs, not bounded.
  - As "end to end" as possible.
  - Regression as fast as possible.

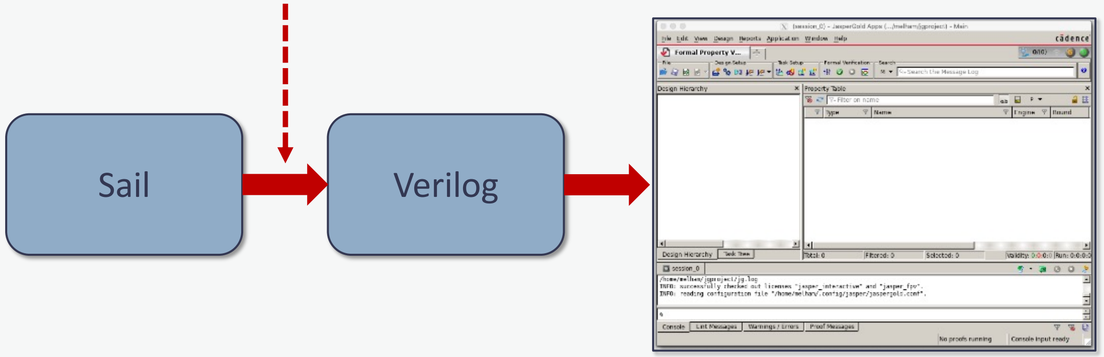
- Outcomes:
  - Found numerous bugs, some serious.
  - Higher confidence in correctness/security guarantees.
  - New methodology and tools.
  - Fully open-source example with permissive license.

  - Eventually… a configurable proof kit for RISC-V.
  - Maybe… full integration of Sail (or similar) into EDA.

# Sail to Verilog Compiler

https://www.cl.cam.ac.uk/~pes20/sail/



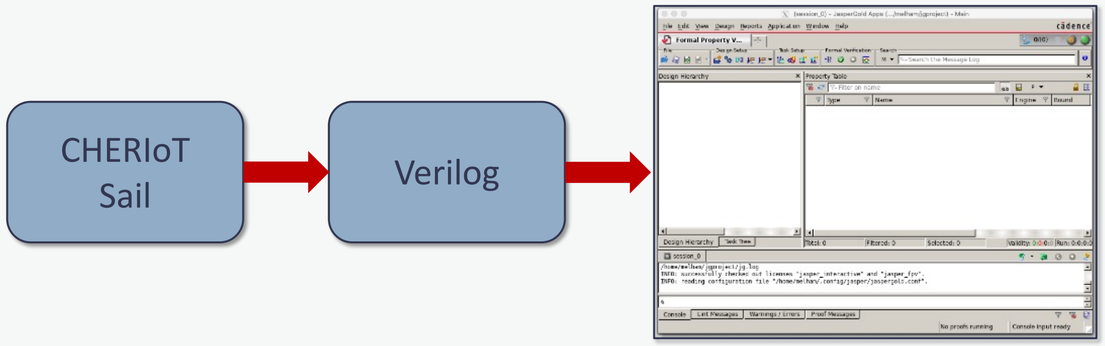





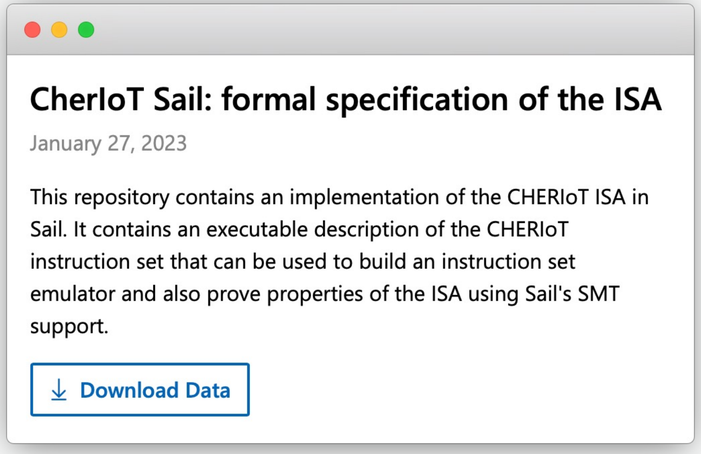

Jasper from Cadence.

- CHERIoT-ibex FV is driving development of a Sail to Verilog compiler.
- Cambridge lead – Alasdair Armstrong, with contributions from Oxford team.
- This allows Sail specifications – for the very first time – to be used in commercial, best-in-class formal verification software tools.
- Overcomes the severe barrier to productivity in previous CHERI RISC-V verification efforts.

# CHERIoT-ibex Sail Specification

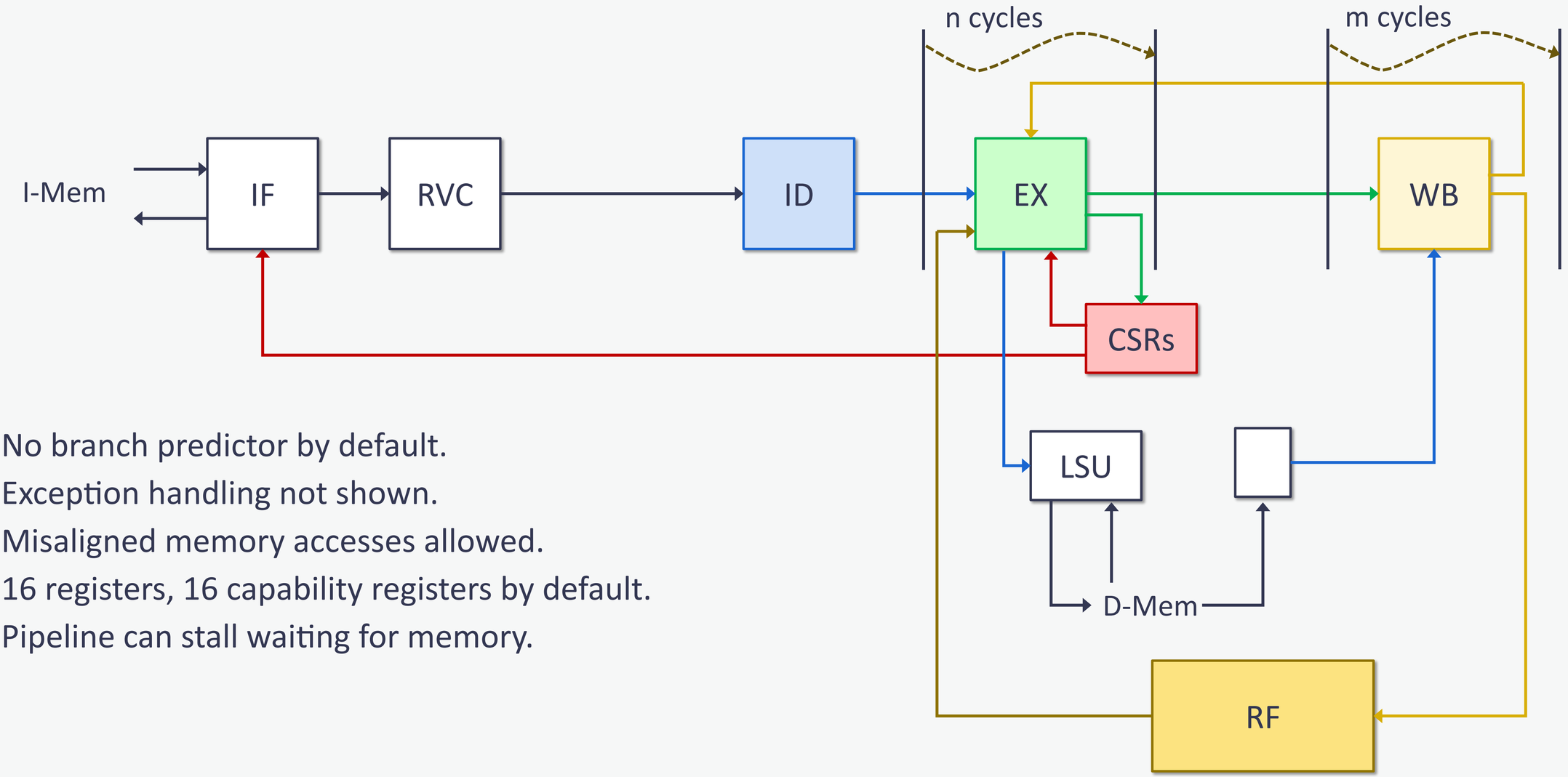



- Purely combinational: state in → state out
- Elaborated size is quite large
  - ~20mins to load the entire thing into Jasper.
  - Proofs noticeably slower if everything is included.
- This means we can't realistically instantiate the spec several times.
  - Could be improved with better abstraction.
  - Or symbolic trajectory evaluation!
- For development, we have a script to carve out only the instructions we want to work on.




https://www.microsoft.com/en-us/research/publication/cheriot-rethinking-security-for-low-cost-embedded-systems/

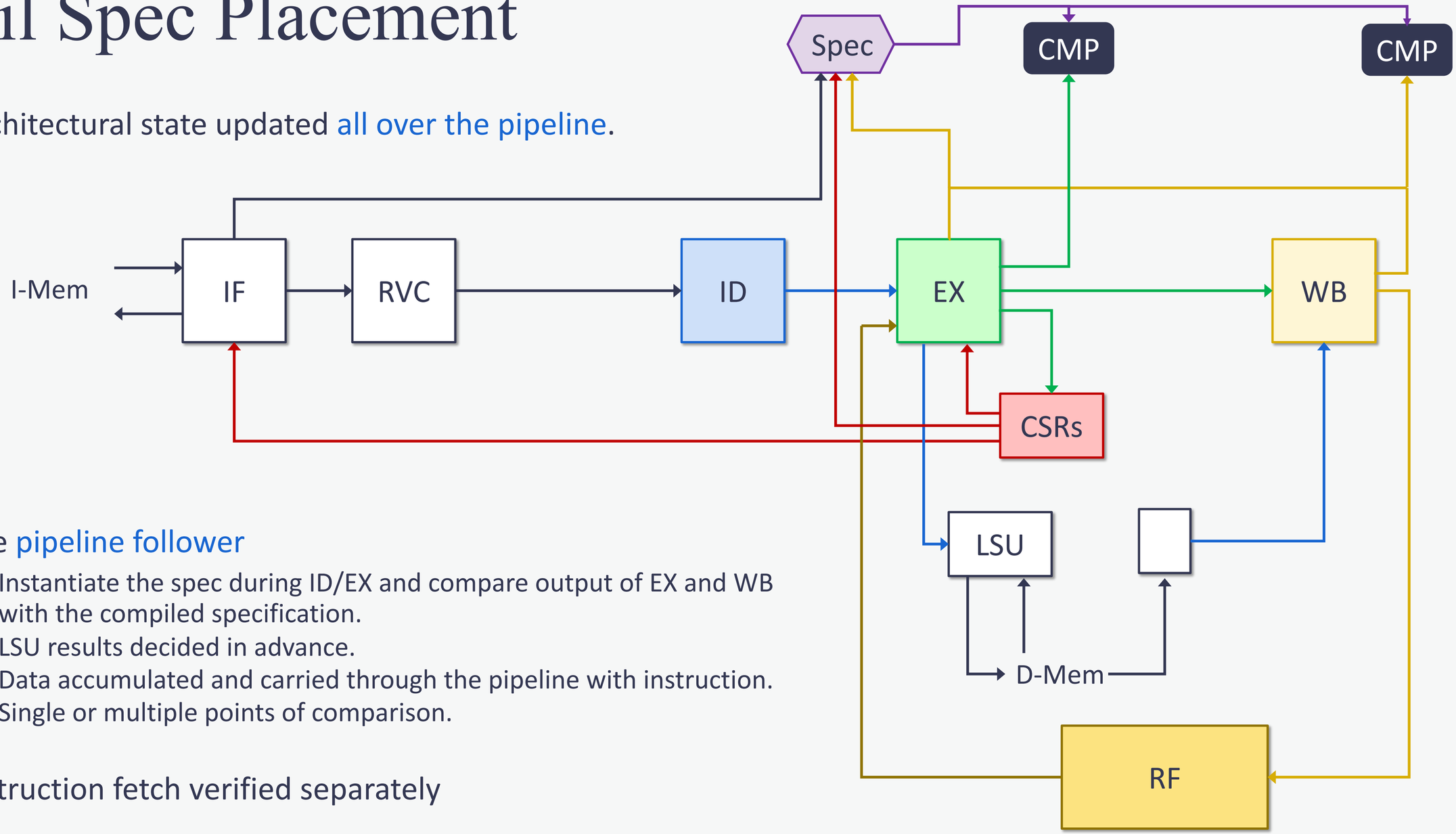
# CHERIoT Pipeline




- No branch predictor by default.
- Exception handling not shown.
- Misaligned memory accesses allowed.
- 16 registers, 16 capability registers by default.
- Pipeline can stall waiting for memory.

# Sail Spec Placement

- Architectural state updated all over the pipeline.




- The pipeline follower
  - Instantiate the spec during ID/EX and compare output of EX and WB with the compiled specification.
  - LSU results decided in advance.
  - Data accumulated and carried through the pipeline with instruction.
  - Single or multiple points of comparison.

- Instruction fetch verified separately

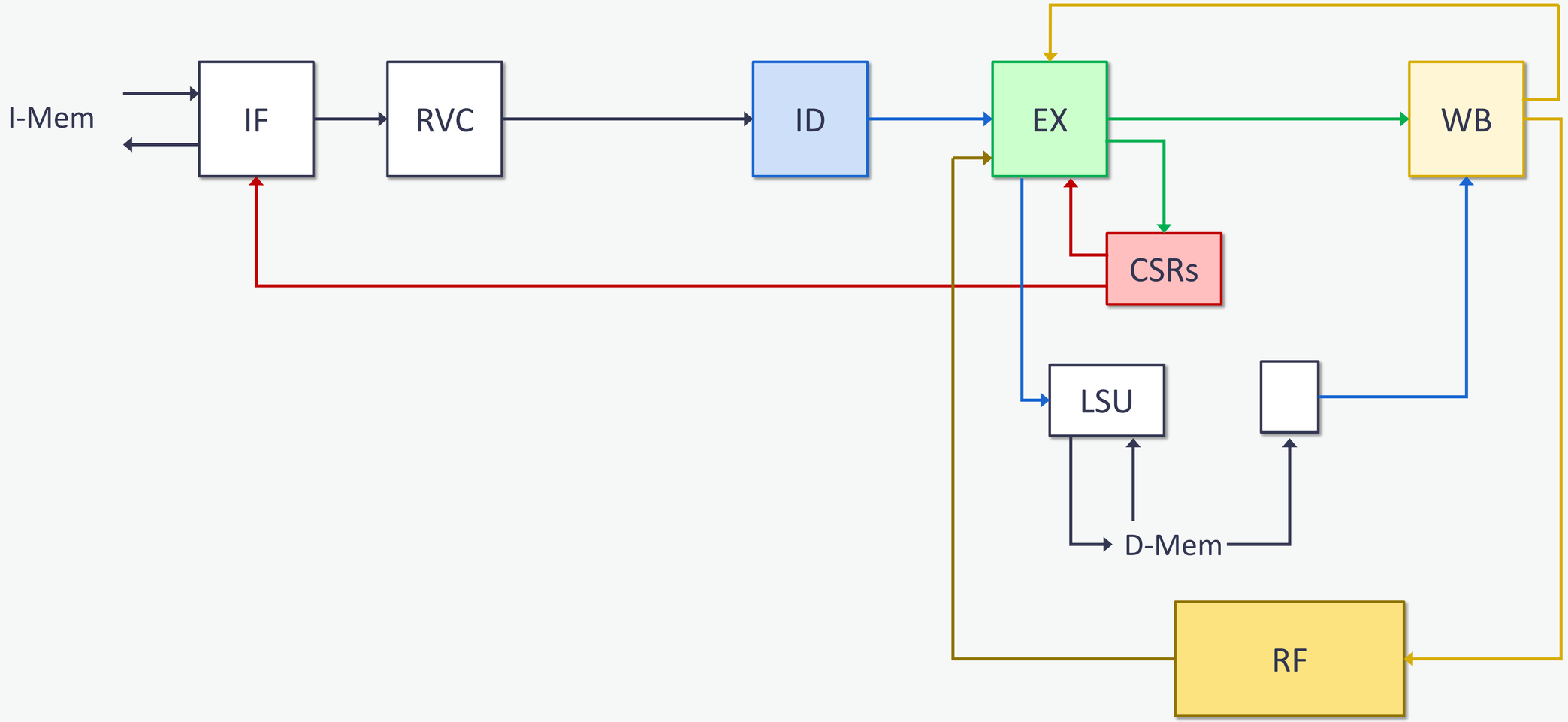
# CHERI Microarchitacture – Data Caching

- Problem: top_cor, base_cor not cached in the Sail, but cached in CherIoT-ibex.

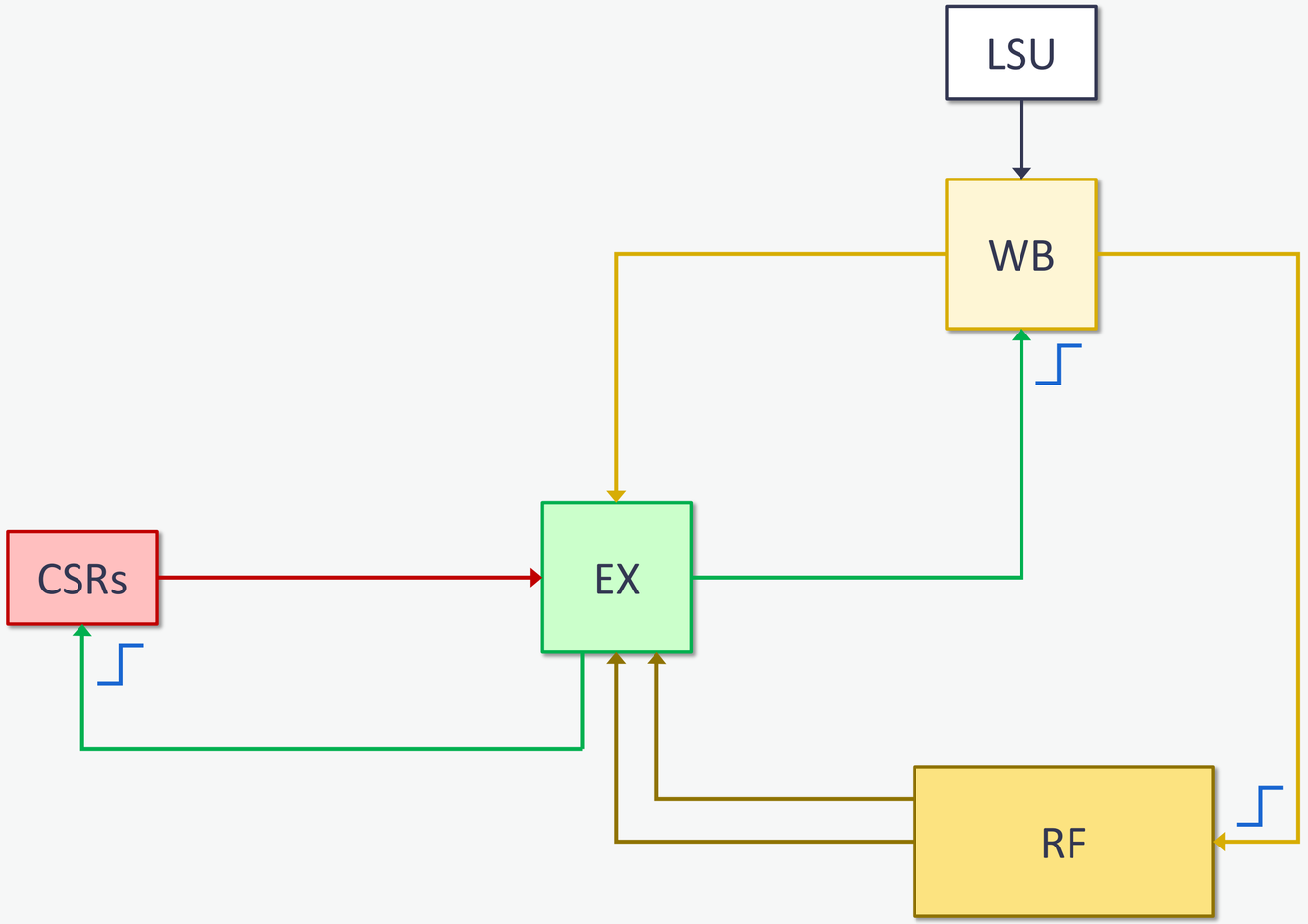
```
// Compressed (regFile) capability type
typedef struct packed {
  logic                  valid;
  logic [1:0]            top_cor;
  logic [1:0]            base_cor;
  logic [EXP_W-1    :0] exp;
  logic [TOP_W-1    :0] top;
  logic [BOT_W-1    :0] base;
  logic [OTYPE_W-1  :0] otype;
  logic [CPERMS_W-1:0] cperms;
} reg_cap_t;
```

```
// Decompressed (execute) capability type
typedef struct packed {
  logic                  valid;
  logic [EXP_W-1    :0] exp;
  logic [ADDR_W     :0] top33;
  logic [ADDR_W-1   :0] base32;
  logic [OTYPE_W-1  :0] otype;
  logic [PERMS_W-1: 0] perms;
  logic [1:0]            top_cor;
  logic [1:0]            base_cor;
  logic [TOP_W-1    :0] top;
  logic [BOT_W-1    :0] base;
  logic [CPERMS_W-1:0] cperms;
  logic [31:0]           maska;
  logic [31:0]           rlen;
} full_cap_t;
```

# Solution – Prove a Global Data Type Invariant



# Solution – Analyse the Flow of Capabilities

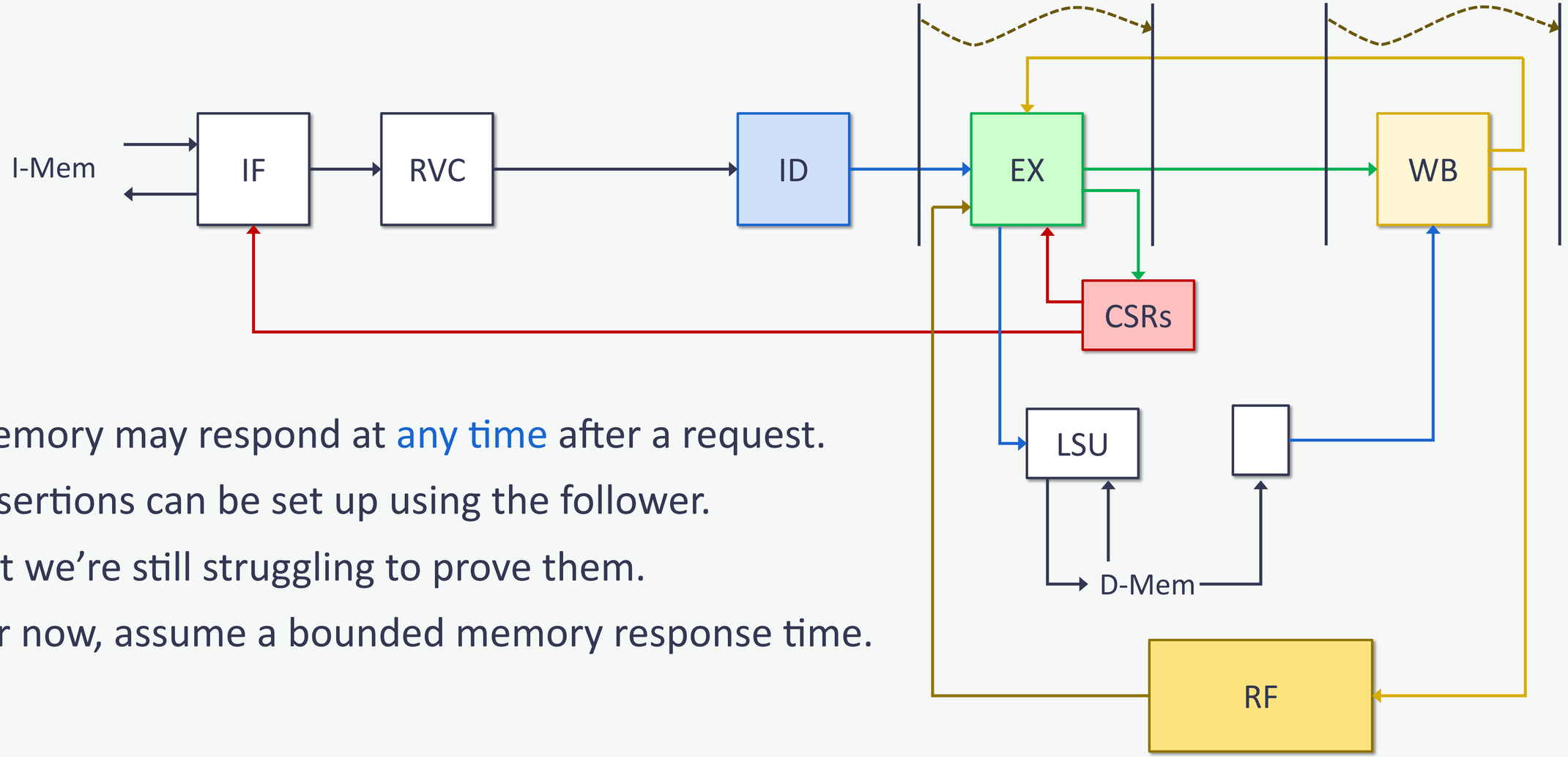


# Solution – An Inductive Argument, in a Model Checker


```
logic rf_s; // Internal state satisfies DTI
assign rf_s = regCapSatsDTI(rf_cap_q[1], rf_reg_q[1]) &
              /* ... */ &
              regCapSatsDTI(rf_cap_q[31], rf_reg_q[31]);

logic rf_in; // Input satisfies DTI
assign rf_in = regCapSatsDTI(wcap_a_i, wdata_a_i);

logic rf_a, rf_b; // Output satisfies DTI
assign rf_a = regCapSatsDTI(rcap_a_o, rdata_a_o);
assign rf_b = regCapSatsDTI(rcap_b_o, rdata_b_o);

DTIInt_Rf: assert property (rf_s & rf_in |=> rf_s);
DTIExt_Rf: assert property (rf_s |-> rf_a & rf_b);
DTI_Rf: assert property (rf_s);
```

# Memory

We'd like to verify under an unbounded liveness assumption about memory.




- Memory may respond at any time after a request.
- Assertions can be set up using the follower.
- But we're still struggling to prove them.
- For now, assume a bounded memory response time.

# A Rich Harvest of Bugs

- Approx. 25 bugs have been found and reported by the Oxford team.
- At least 4 bugs break montonicity.
- At least one bug has a software exploit allowing us to move the bounds of a capability, breaking CHERI security.
- Formal has also prompted a discussion of exactly what, in CHERIoT-ibex, is a valid capability.

Illegal CLC memory load (breaks monotonicity)
CLC tag bit leak
CSeal otypes
CJALR alignment check
CSEQX memory vs decoded
MTVEC, MEPC legalisation
CSC alignment checks
CSC decoding
Store local violation
Memory capability layout
PCC.address ≠ PC
CJAL/CJALR differences
Memory bounds check overflow (breaks monotonicity)
CLC tag/perms clearing (breaks monotonicity)
MSHWM/MSHWMB boundary updates 16. tvec_addr alignment (spec bug?)
Sealed PCC
IF overflow
CSetBounds lower bound check (breaks monotonicity)
Countless exception priority issues

# Progress to Date

- All CHERI, memory and some RISC-V ({IRUB}-TYPE) instructions verified with fully conclusive unbounded proofs.
  - Includes GPRs + CSRs in both exception and non-exception cases.
  - Memory proofs are under bounded response time assumptions. The proofs check addresses, write enable, and write data.
  - Includes RISC-V compressed variants, though the solution is hacky.
- Fetch (PCC) checks and exceptions, and instruction fetch correctness.
- IRQ handling.
- A handful of to-dos:
  - SHIFTIOPs (similar to other R-TYPEs)
  - MTYPE (multi+div - classic data path)
  - CSR / ECALL / MRET / SRET / EBREAK (all fiddly but certainly doable)
  - WFI (unclear how difficult that will be)
  - FENCEs (already proving, but essentially as no-ops)
- How do we know we have proved everything?
  - trace equivalence…
- Next: verify the OS/software stack on top

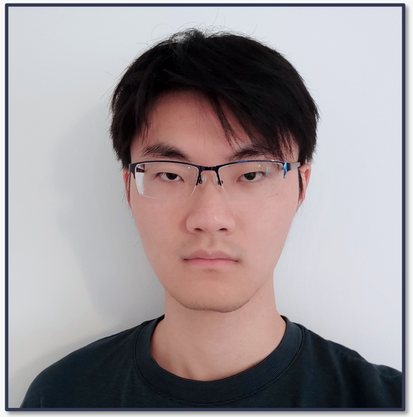
# Progress to Date

- All CHERI, memory and some RISC-V ({IRUB}-TYPE) instructions verified with fully conclusive unbounded proofs.
  - Includes GPRs + CSRs in both exception and non-exception cases.
  - Memory proofs are under bounded response time assumptions. The proofs check addresses, write enable, and write data.
  - Includes RISC-V compressed variants, though the solution is hacky.
- Fetch (PCC) checks and exceptions, and instruction fetch correctness.
- IRQ handling.
- Potential compiler/tool improvements.
  - DAG in-lining (even just on a per instruction basis).
  - Generated stopats (abstractions).
  - Automatic compressed instruction splitting.
    - Including them increases elaborated size a lot (lots of repeated logic).
    - Map decoded ASTs into uncompressed ASTs, then execute (reuses the same logic).
  - Better debuggability (or tools for it).
    - Modules and muxes for better whys.
    - Convert traces into Sail inputs.
  - Dealing with RISC-V configurability.

# The Oxford CHERIoT FV Team

**Lead FV Engineer**
**Louis-Emile Ploix**
**Second year CS, Oxford**

**3rd-Year UG Project Pioneer**
**Ray Lin**
**Fourth year CS, Oxford**



**Haolong Wang**
**MPhil ACS, Cambridge**

**Anastasia Courtney**
**MEng CS, Cambridge**

# Thank You for Listening