Automated Formal Verification of Event Orderings in Parallel Hardware and Software

Yatin A. Manerkar (Princeton University)

Memory Consistency Models (MCMs)

- MCMs specify ordering rules which constrain values that can be read by load instructions in parallel programs
  - Sequential Consistency [Lamport IEEE Tran. 1979]
- ISA-level MCMs defined in terms of acyclicity, irreflexivity, etc. of relational patterns
  - [Shasha & Snir TOPLAS 1988] [Alglave et al. TOPLAS 2014]

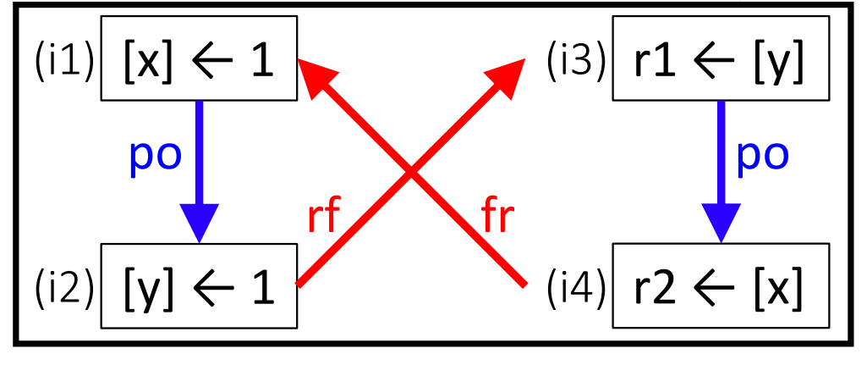
Message passing (mp) litmus test

Core 0	Core 1
(i1) [x] ← 1 (i2) [y] ← 1	(i3) r1 ← [y] (i4) r2 ← [x]
Under SC: Forbid r1=1, r2=0	

ISA-Level MCM Specification of SC

```
acyclic (po U co U rf U fr)
```

An ISA-level execution of mp

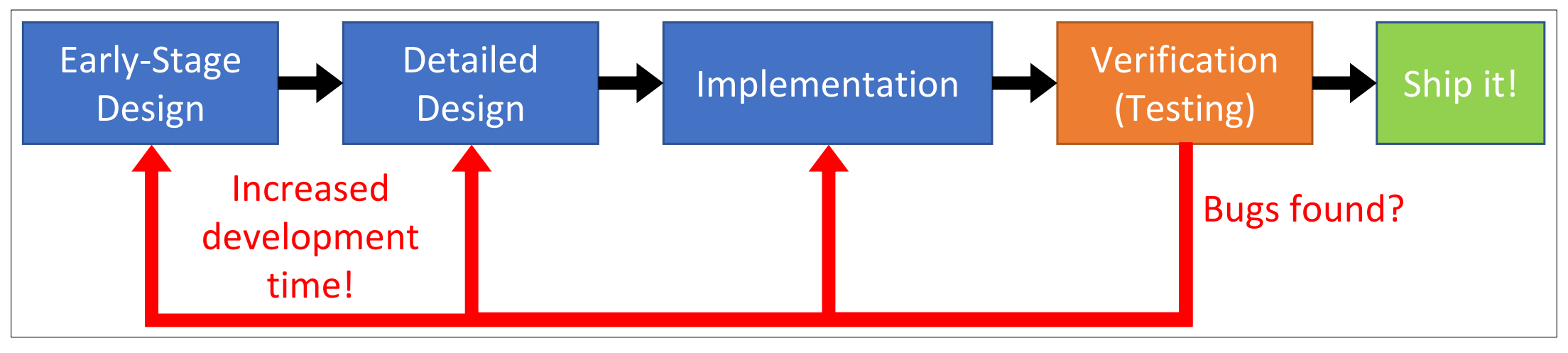


Nodes: Instructions
Edges: ISA-level relations

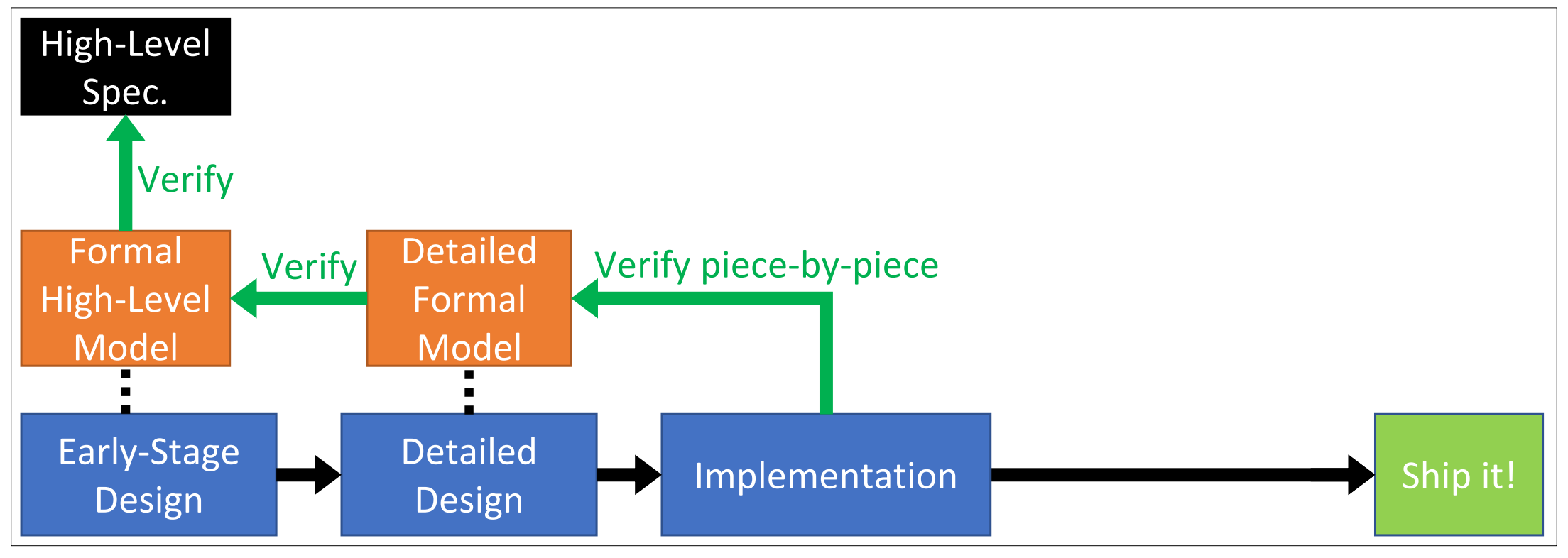
Bringing Verification Earlier in the Design Timeline

- Verification costs now dominate total hardware design cost [Foster DAC 2014]
- Formal verification early in the design timeline can catch bugs quicker, and allow models to evolve with the system!

Standard Timeline



Proposed Timeline



Prior Work: Microarchitectural MCM Modelling

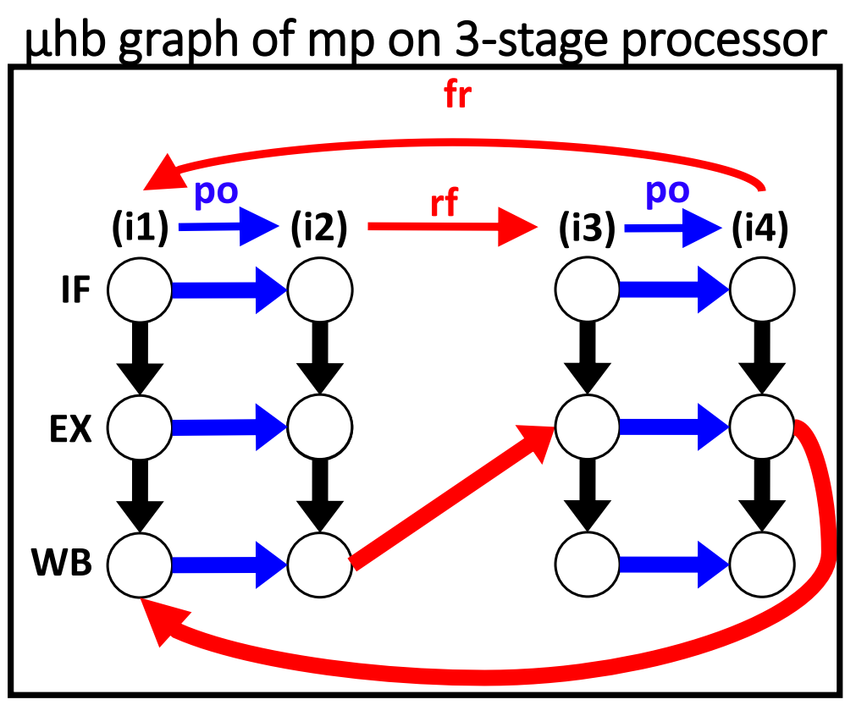
- Microarchitectural ordering specifications defined as set of µspec axioms [Lustig. et al. ASPLOS 2016]
- Microarchitectural executions are µhb (microarchitectural happens-before) graphs

µspec Microarchitectural Ordering Specification

```
Axiom "Fetch_is_FIFO":
... EdgeExists ((i1, IF), (i2, IF))
      => AddEdge ((i1, EX), (i2, EX)).

...
```

µhb graph of mp on 3-stage processor

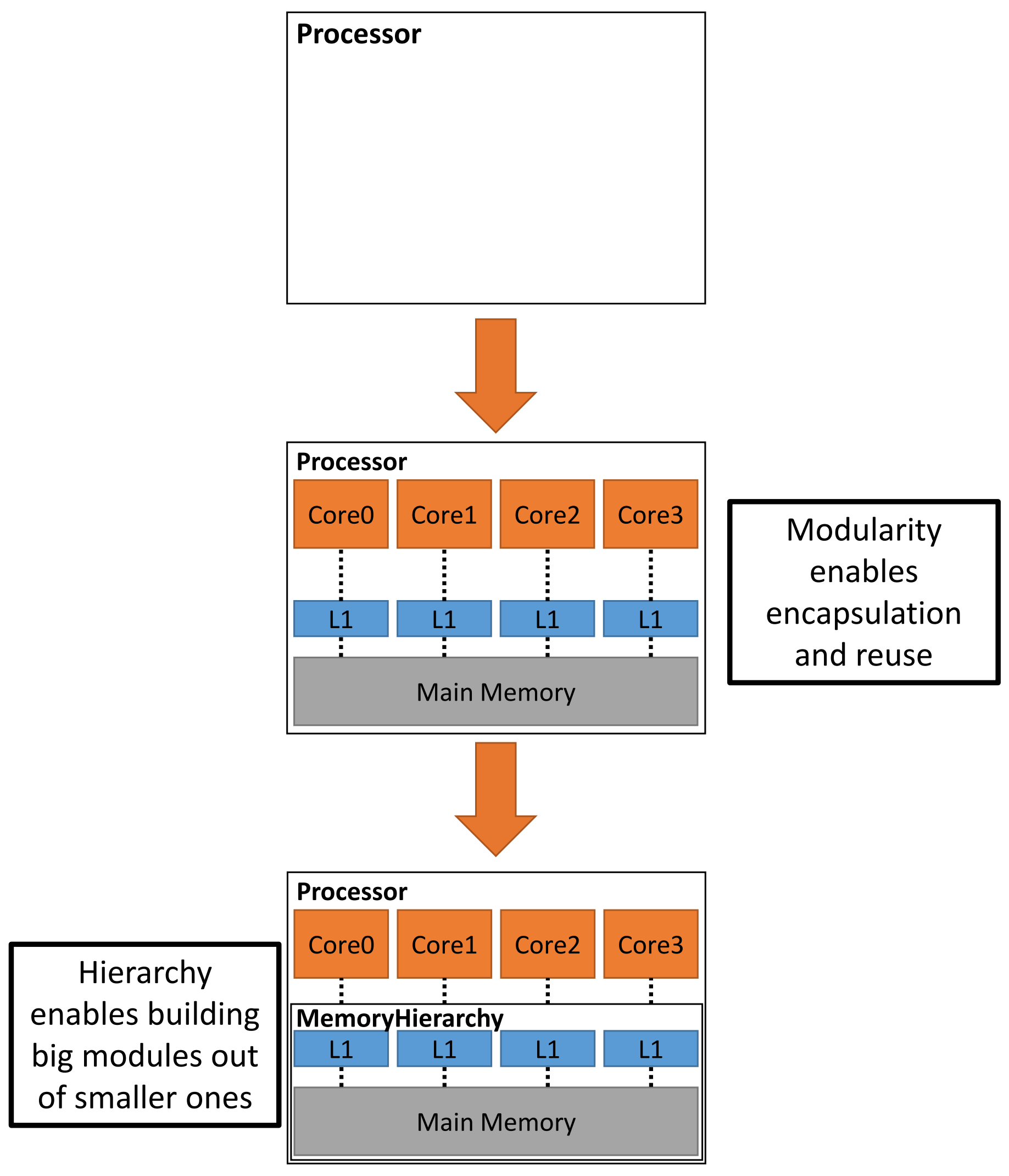


Nodes: Instr. sub-events
Edges: Happens-before order

Cyclic graph => Unobservable
Acyclic graph => Observable

RealityCheck: The Need for Modularity

- Flat verification is at odds with realities of design process
- Teams develop components independently and later connect them together

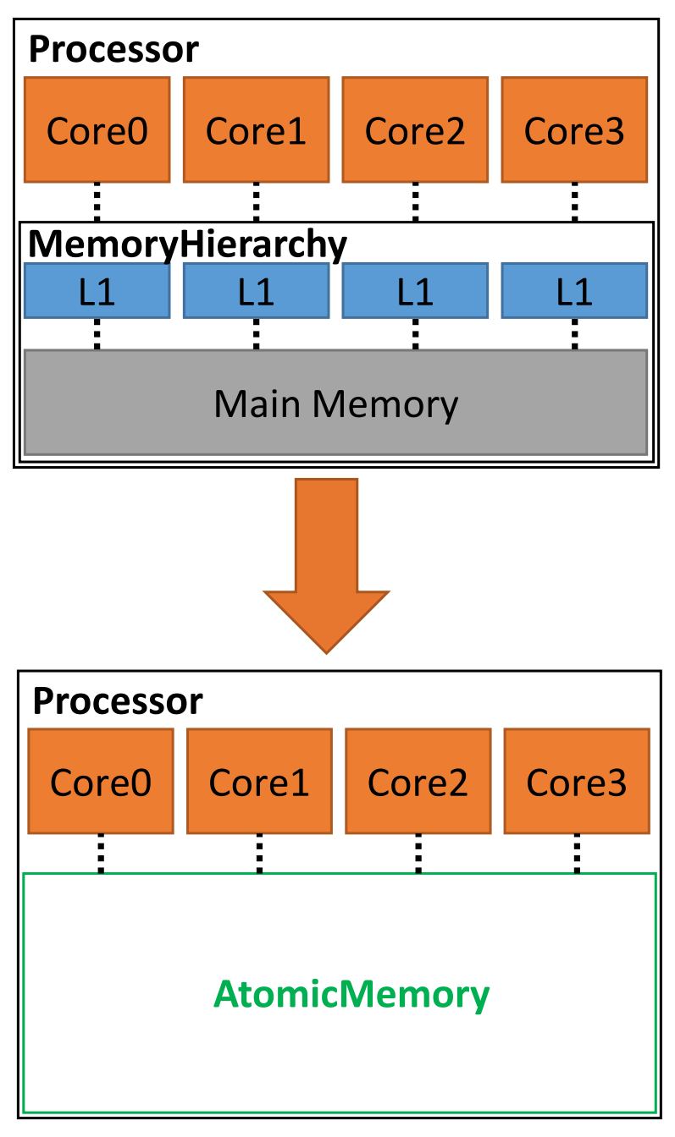


Modularity enables encapsulation and reuse

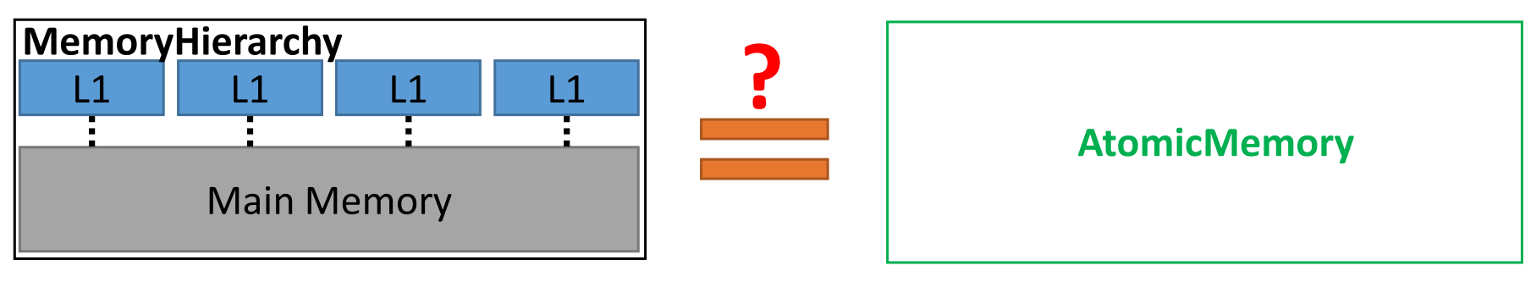
Hierarchy enables building big modules out of smaller ones

RealityCheck: Scalability Using Abstraction

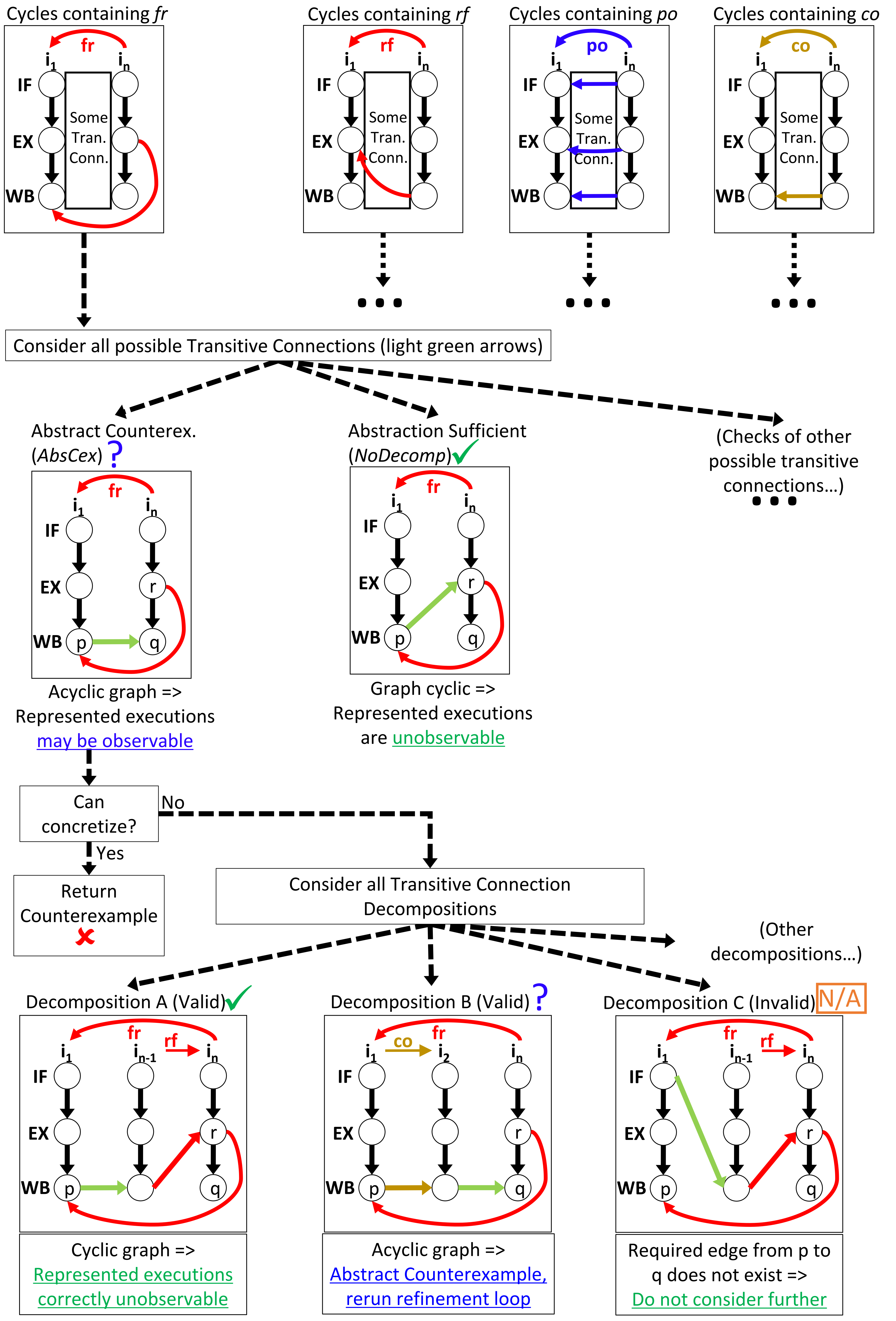
- Break down verification into smaller verification problems
- 1) Represent component using abstract interface
  - Hide internal details => simpler verification



- 2) Interface verification: check implementation vs interface



PipeProof: Automated All-Program Microarchitectural MCM Correctness Proofs



Consider all possible Transitive Connections (light green arrows)

(Checks of other possible transitive connections…)

Acyclic graph => Represented executions may be observable

Graph cyclic => Represented executions are unobservable

Can concretize?

Return Counterexample ✗

Consider all Transitive Connection Decompositions

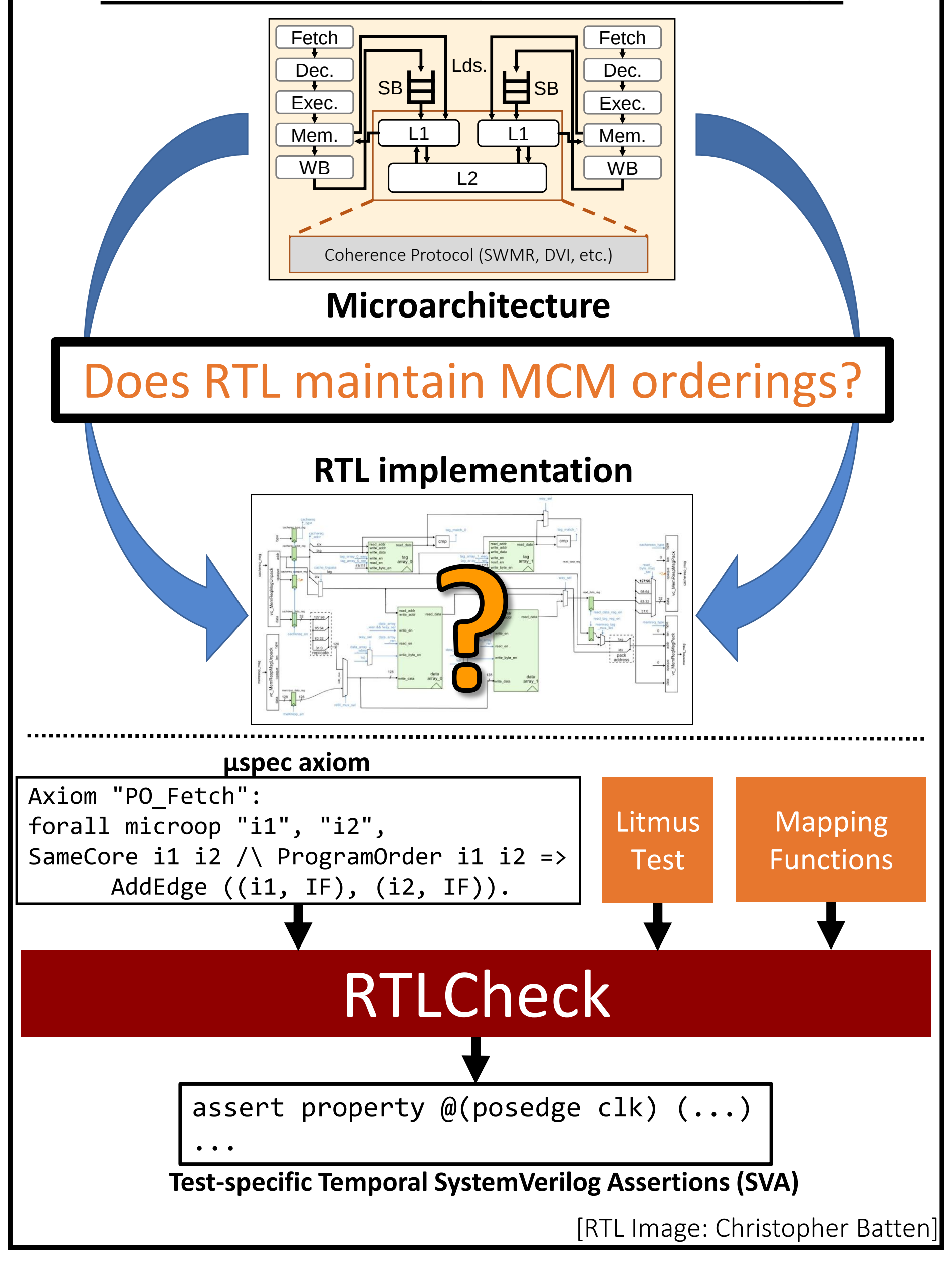
(Other decompositions…)

Cyclic graph => Represented executions correctly unobservable

Acyclic graph => Abstract Counterexample, rerun refinement loop

Required edge from p to q does not exist => Do not consider further

RTLCheck: Going Down to Verilog


Fetch
Dec.
Exec.
Mem.
WB
Lds.
SB
SB
L1
L1
L2
Coherence Protocol (SWMR, DVI, etc.)
Microarchitecture
Does RTL maintain MCM orderings?
RTL implementation
?
µspec axiom
Litmus Test
Mapping Functions
RTLCheck
Test-specific Temporal SystemVerilog Assertions (SVA)

```
Axiom "PO_Fetch":
forall microop "i1", "i2",
SameCore i1 i2 /\ ProgramOrder i1 i2 =>
      AddEdge ((i1, IF), (i2, IF)).
```

```
assert property @(posedge clk) (...)
...
```

[RTL Image: Christopher Batten]

RTLCheck: The µspec/SVA Mismatch

- Tricky to translate from µspec to SVA correctly
- Situation further complicated by SVA verifiers (e.g. Cadence JasperGold) not fully supporting SVA spec
- Example:

```
Axiom "Read_Values":
Load returns 0 => Load is BeforeAllWrites
```

- In SVA, this is represented as:

```
assume property (a); // Load returns 0
assert property (b); // Load is BeforeAllWrites

//The above is equivalent to...
assert property ((always a) implies (always b));
```

- SVA is based on temporal logic (G = always, F = eventually):

```
G a -> G b = (~(G a)) \/ G b = (F ~a) \/ G b
```

- Assumptions introduce liveness (through use of **F**)
  - Expensive to check! [Cerny et al. 2010]
- SVA verifiers **approximate**: only check assumptions until current state
  - Property is then easier to check…
  - …but assumptions no longer correctly enforced!
- RTLCheck Solution: Generate properties that handle all test outcomes

Note: µspec axiom abstracted for brevity

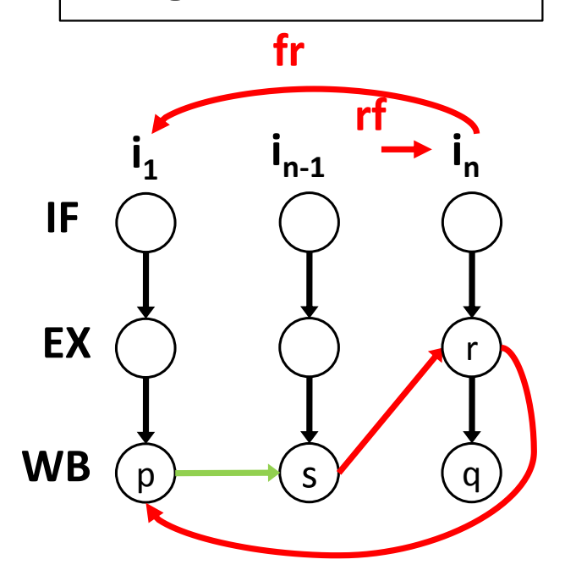
Nuts, Bolts, and Results

- All tools provide automated formal verification
- Core to approach: happens-before graphs checked for cycles using efficient SMT solvers
  - RealityCheck uses Z3 solver from Microsoft Research
- RTLCheck utilises commercial SVA prover (Cadence JasperGold) to check generated SVA assertions
- Some Results:
  - PipeProof can prove correct simple microarchitectures implementing SC and TSO for all programs in < 1 hour
  - JasperGold proved 89% of all RTLCheck-generated properties for 56 litmus tests in 11 hrs/test for open-source RISC-V V-scale processor
  - RTLCheck discovered a bug in the memory implementation of V-scale (reported to developers)
  - Discovered two counterexamples to "trailing-sync" compiler mapping from C11 to Power and ARMv7 [Manerkar et al. CoRR 2016] using TriCheck [Trippel et al. ASPLOS 2017]

Different Tools for Different Points in the Verification Timeline

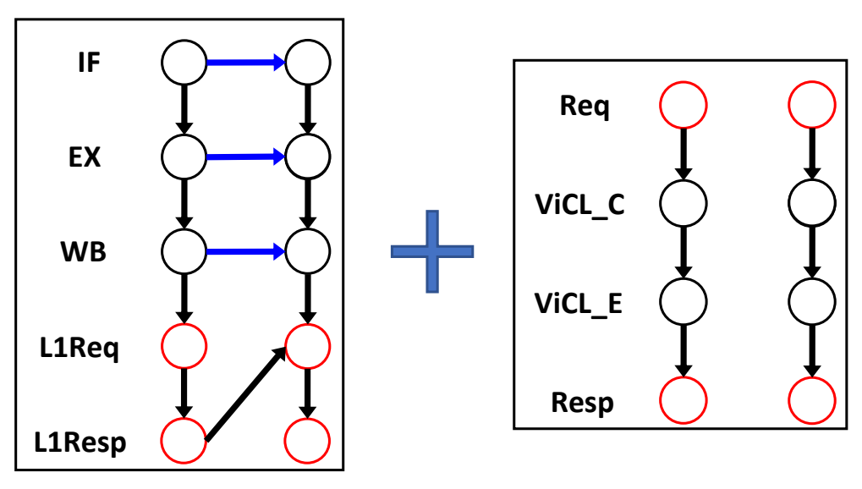
- **PipeProof:** Early-stage flat formal verification of design across all possible programs against ISA-level MCM
- **RealityCheck:** Mid-stage hierarchical formal verification of detailed design against ISA-level MCM (bounded)
- **RTLCheck:** Late-stage formal verification of Verilog implementation against microarchitectural orderings (bounded)

PipeProof



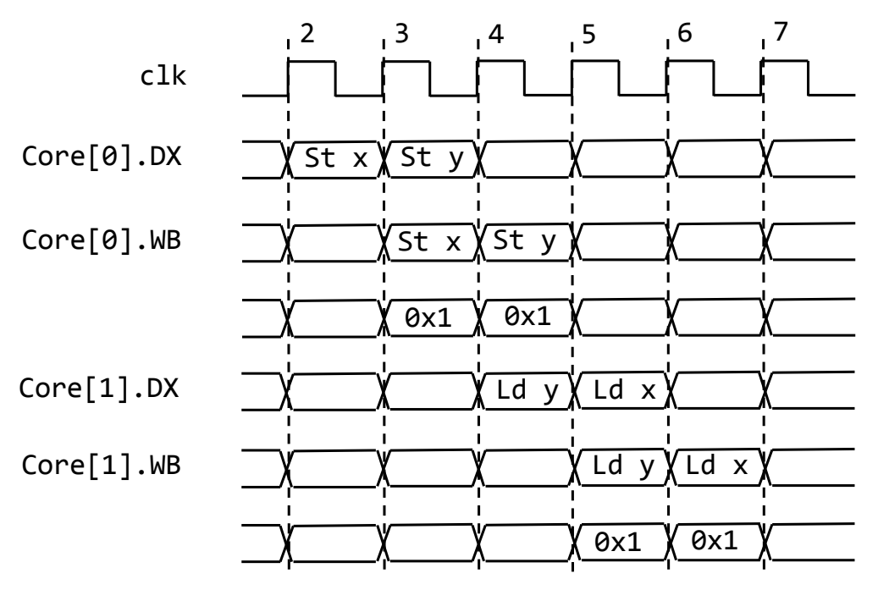
Early-Stage All-Program Verification

RealityCheck



Modular and Scalable Verification of Detailed Design

RTLCheck



Implementation-Time Verification of Verilog RTL

Conclusions

- Verification now dominates total hardware design cost
- Bringing verification earlier in design timeline can find bugs quicker and reduce overall development time
- Memory consistency models (MCMs) specify ordering rules which constrain values read by loads in parallel programs
- My thesis: automated formal methodologies and tools for verification of MCM implementations
- PipeProof: Ensure that design respects MCM for all possible programs before RTL is written!
- RealityCheck: Modular, scalable verification capable of tackling large designs
- RTLCheck: Checking early-stage microarchitectural ordering specifications on real processor implementations

Papers and code available at www.cs.princeton.edu/~manerkar