Automated Verification of Memory Consistency Model Implementations

Yatin A. Manerkar

Princeton University

manerkar@princeton.edu

DeepSpec Summer School 2017 Student Talk

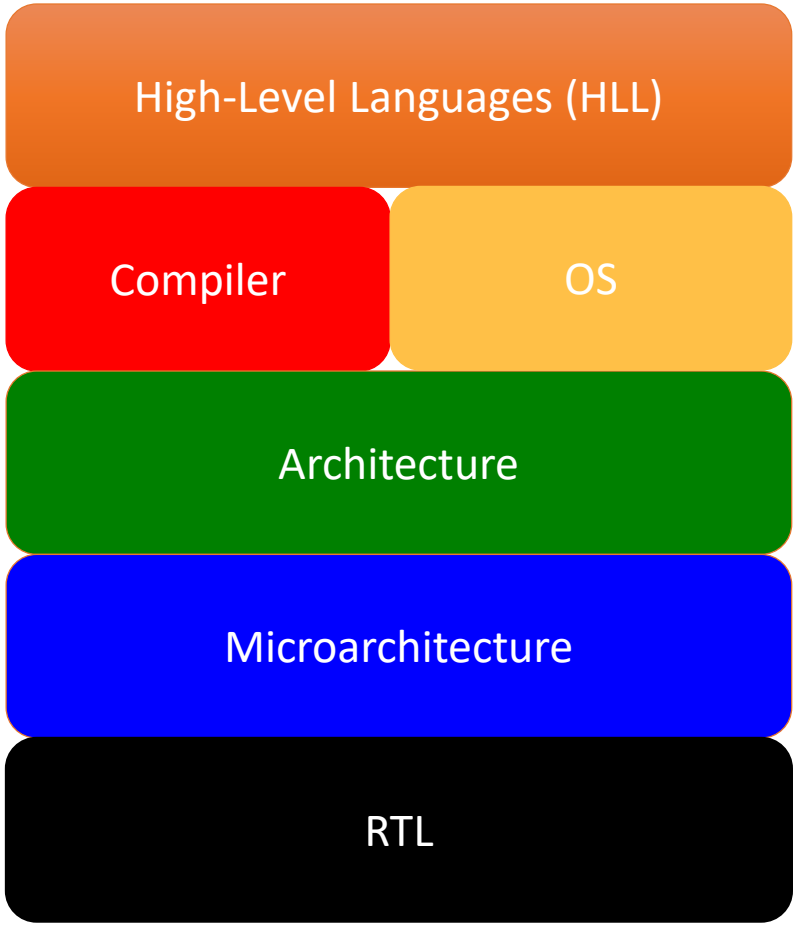
http://check.cs.princeton.edu

Why do Memory Consistency Models Matter?

Core 0	Core 1
Data = 100;	While (Flag != 1) {}
Flag = 1;	int r1 = Data;
(All locations initially have a value of 0)	

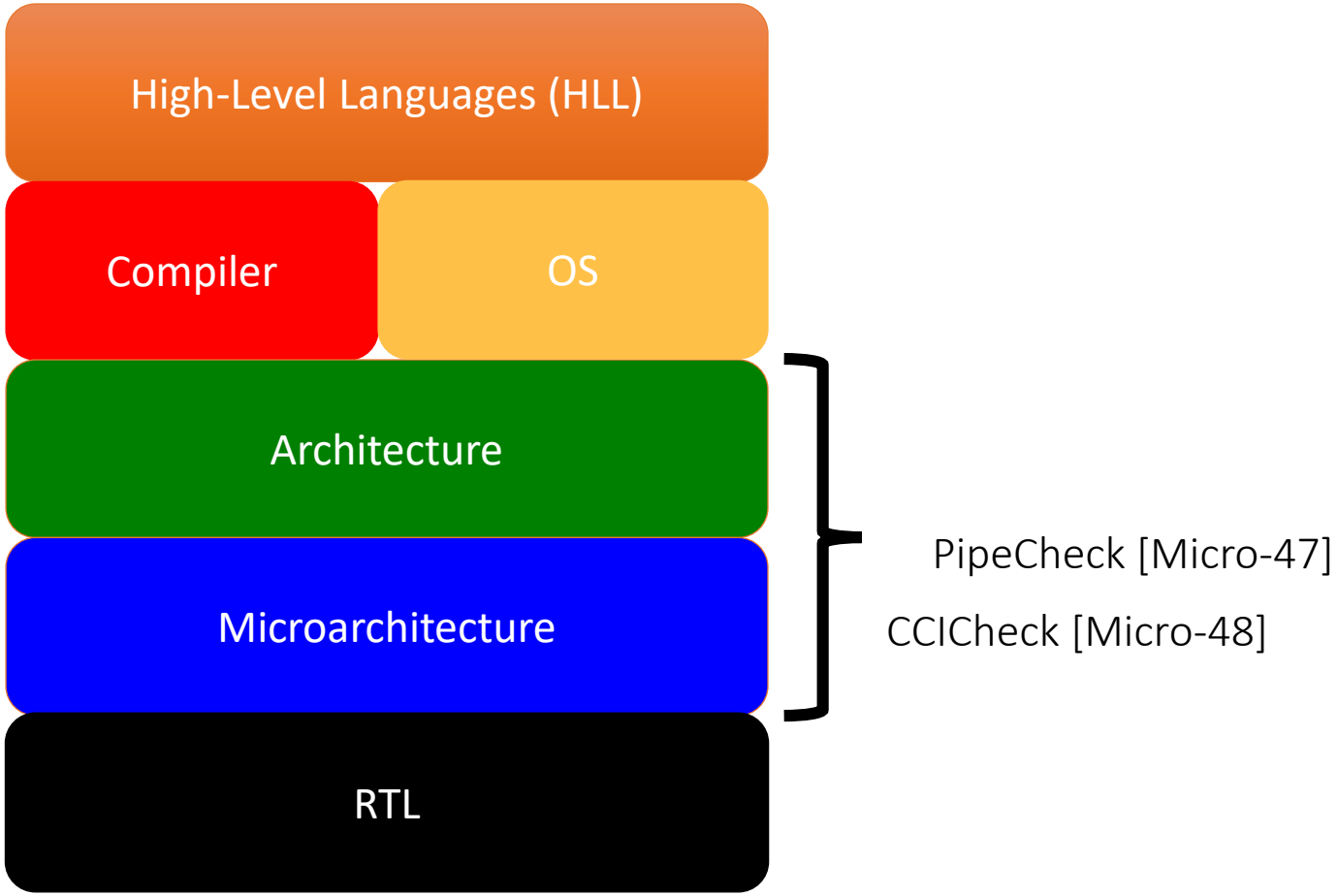
- Can r1 get a value other than 100?
  - Intuitively, no, but…
- What if hardware reorders statements? Or compiler reorders stores?
  - Is this allowed or forbidden?
- Dictated by the Memory Consistency Model (MCM) of the system!
  - SC (sequential consistency), TSO, etc…

Consistency Verification Layers



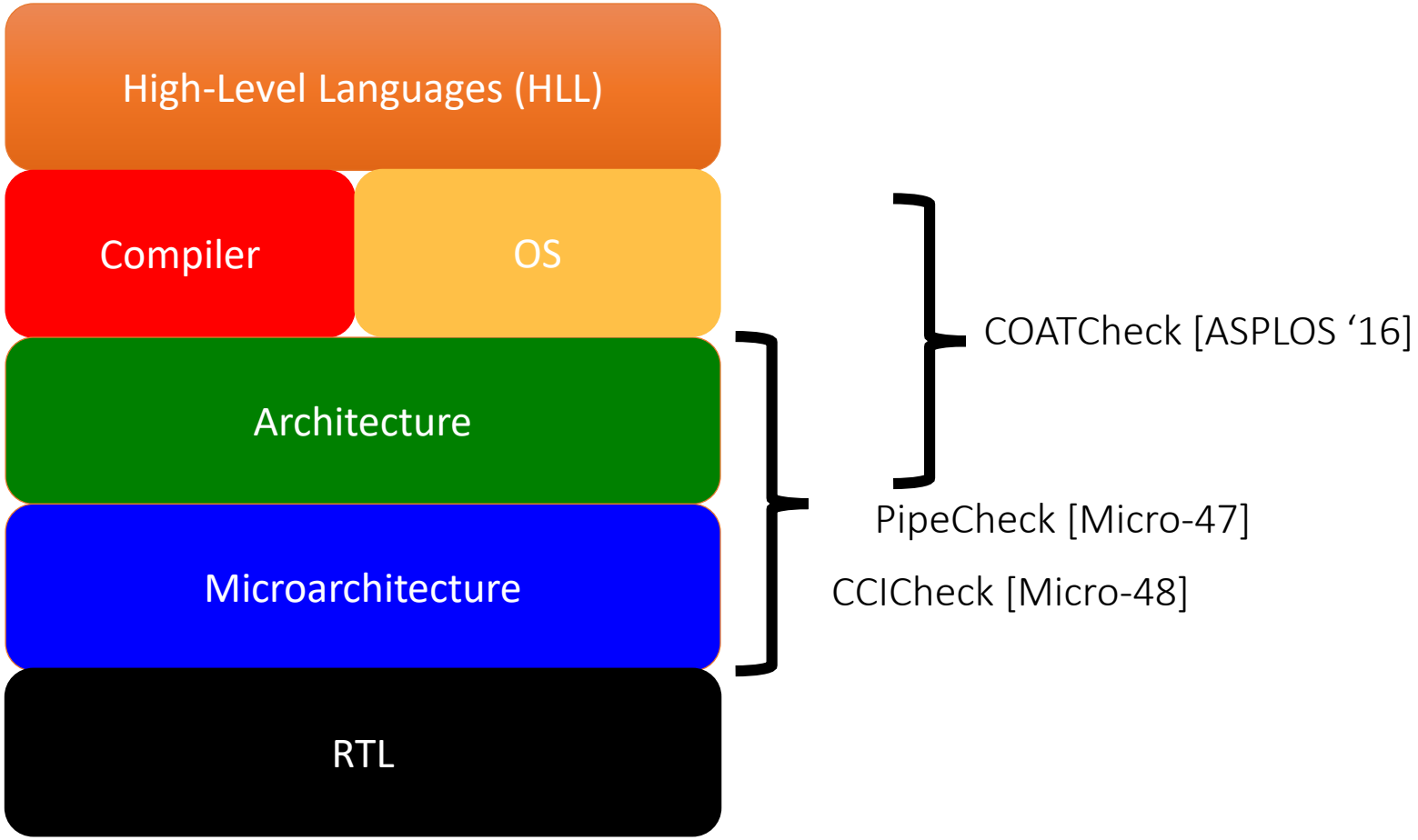
- MCMs are defined at multiple levels of the stack
  - HLL Memory Model (C11, Java, etc)
  - ISA-level Memory Model (x86-TSO, ARMv7,…)
- Bugs can occur at any layer of this stack!
- Need to verify that each layer correctly fulfills its guarantees to layers above it
- All levels need to work **together** to correctly implement the MCM of the system

The Check Suite



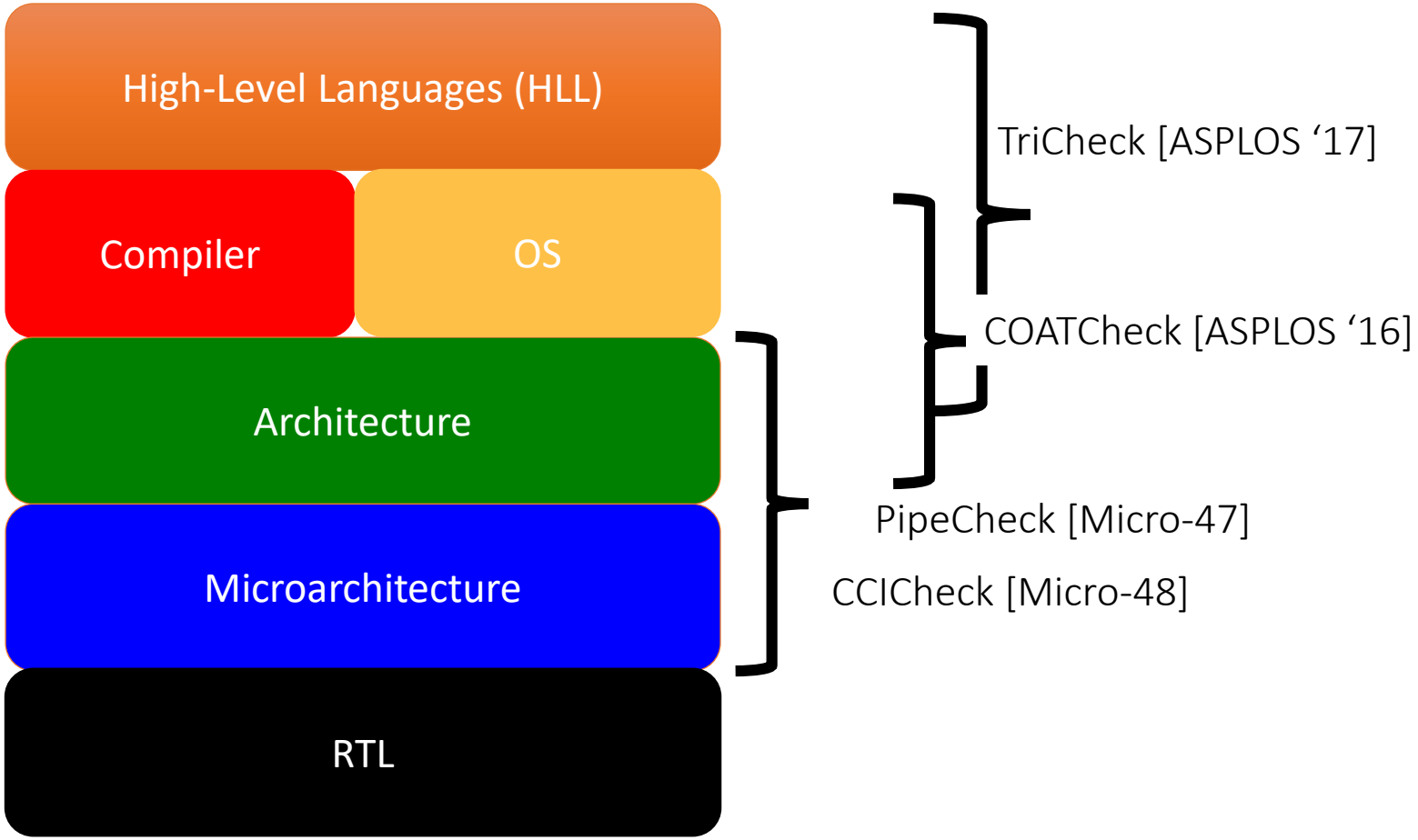
- Our group has developed tools that can **automatically** verify whether implementations correctly respect their MCM for a suite of test programs

The Check Suite



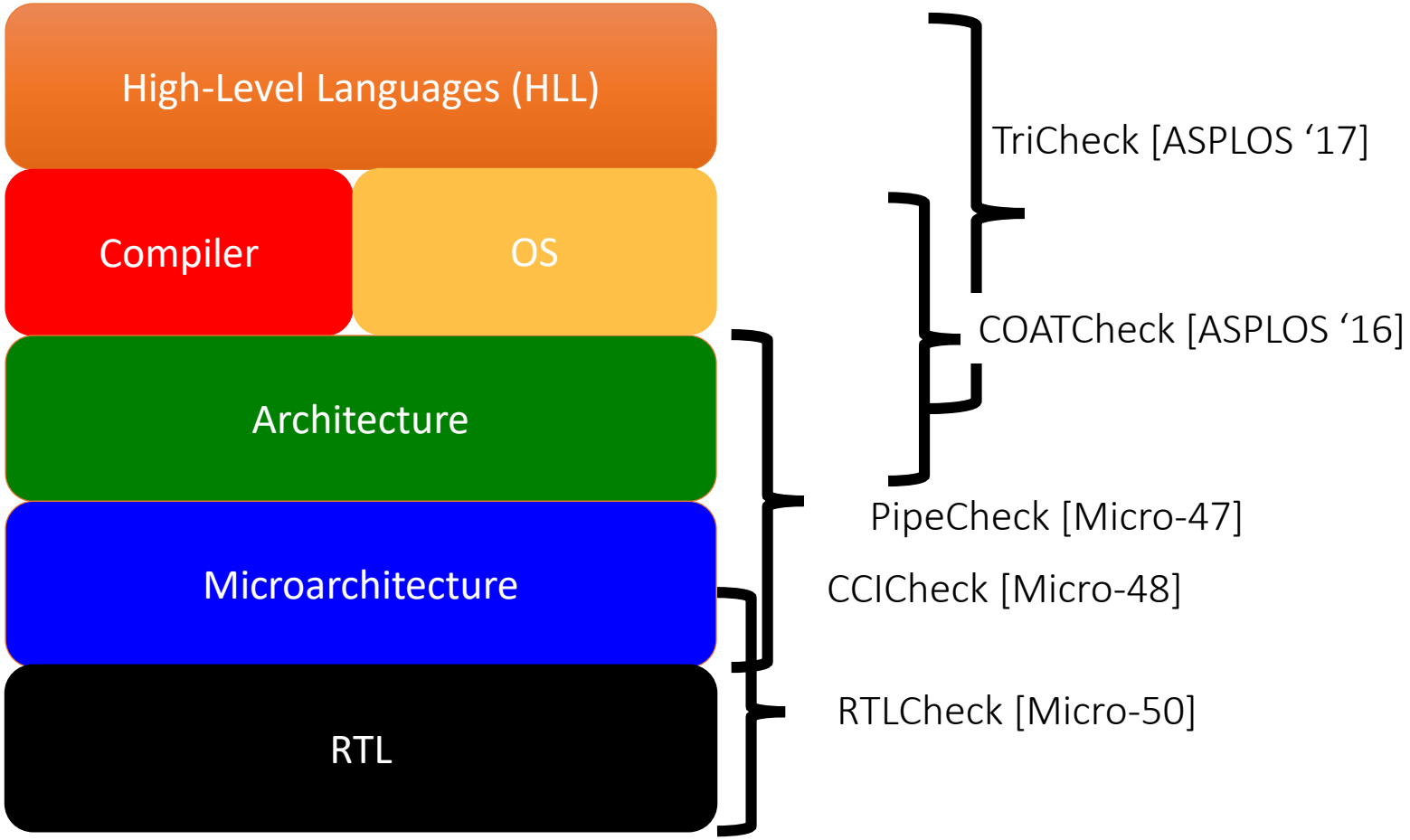
- Our group has developed tools that can **automatically** verify whether implementations correctly respect their MCM for a suite of test programs

The Check Suite



- Our group has developed tools that can **automatically** verify whether implementations correctly respect their MCM for a suite of test programs

The Check Suite



- Our group has developed tools that can **automatically** verify whether implementations correctly respect their MCM for a suite of test programs

Check Inputs: Microarchitecture Spec + Litmus Tests

Microarchitecture Specification in *μSpec* DSL

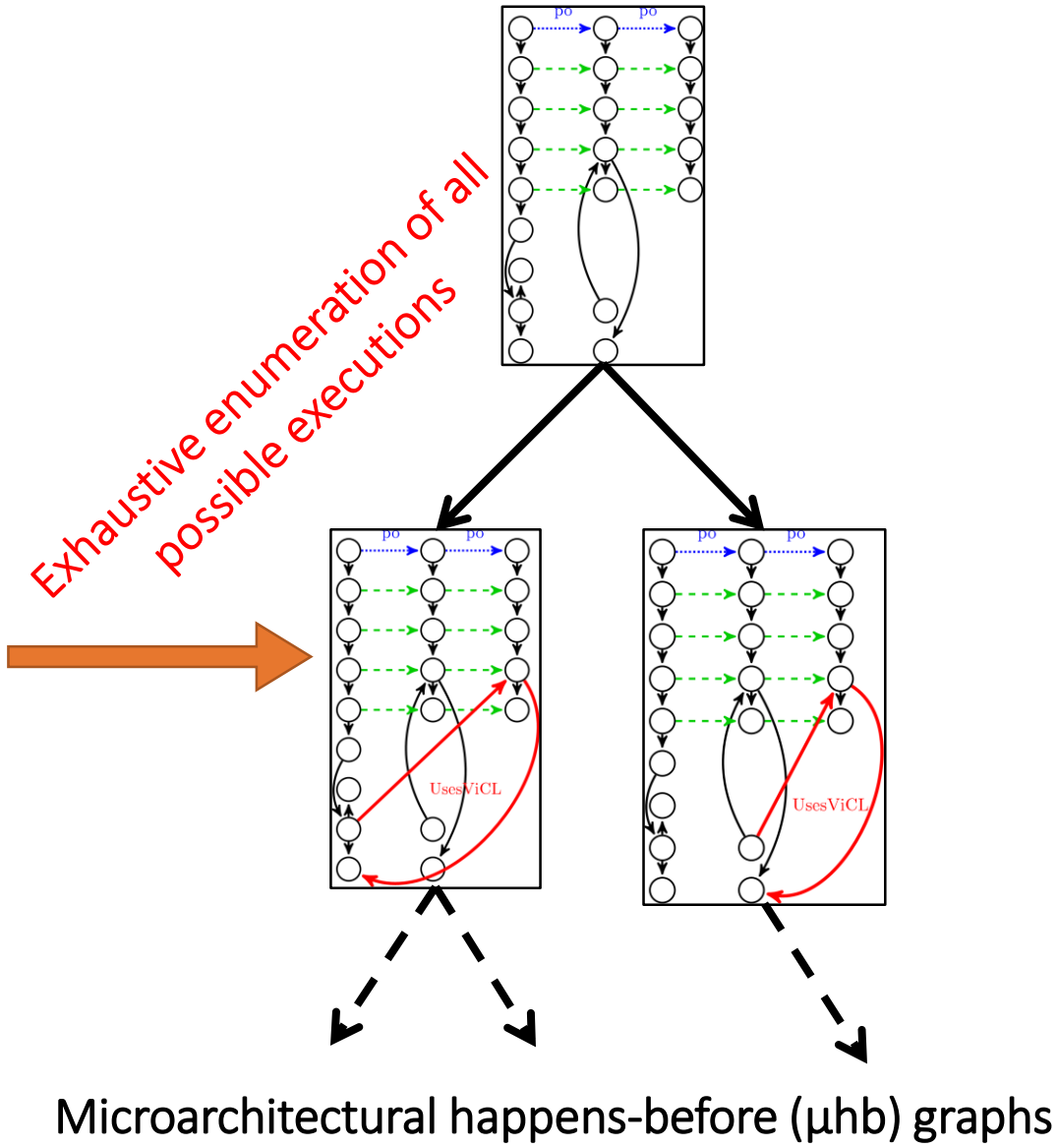
```
Axiom "PO_Fetch":
forall microops "i1",
forall microops "i2",
SameCore i1 i2 /\ ProgramOrder i1 i2 =>
  AddEdge ((i1, Fetch), (i2, Fetch), "PO").

Axiom "Execute_stage_is_in_order":
forall microops "i1",
forall microops "i2",
SameCore i1 i2 /\
 EdgeExists ((i1, Fetch), (i2, Fetch)) =>
   AddEdge ((i1, Execute), (i2, Execute), "PPO").
```

+

Litmus Test

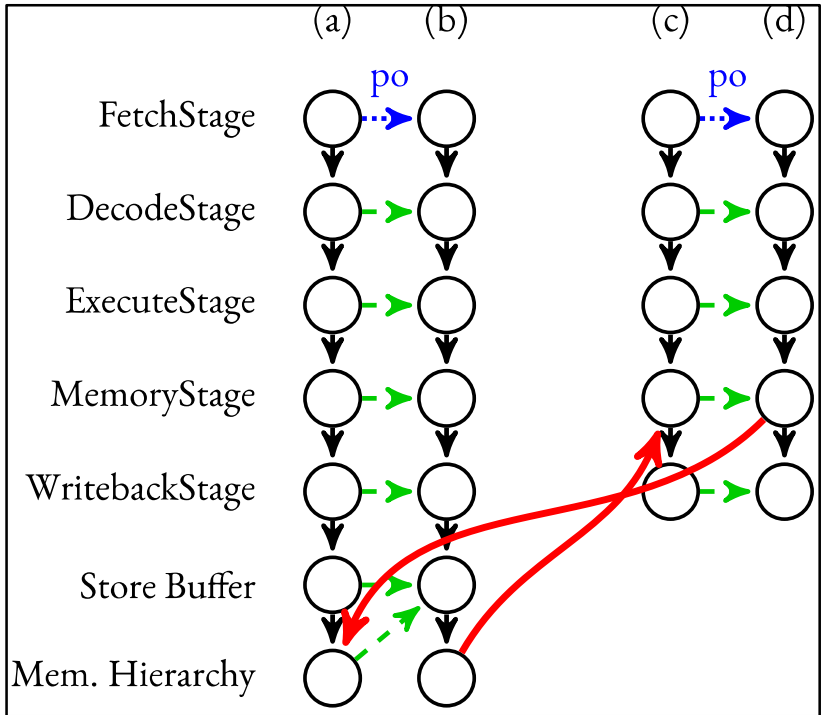
Core 0	Core 1
(i1) St [x] ← 1	(i3) Ld r1 ← [y]
(i2) St [y] ← 1	(i4) Ld r2 ← [x]
Under TSO: Forbid r1=1, r2=0	



Microarchitectural Consistency Verification with Check

- Key Idea: Model executions as **μhb graphs**
  - **Nodes:** Microarchitectural events or pipeline stages
  - **Edges:** *Local* happens-before relationships between nodes
- **Automatic** model checking of all possible executions through **exhaustive enumeration**
  - Multiple graphs generated for a single litmus test
  - Cyclic Graph → Outcome Not Observable
  - Acyclic Graph → Outcome Observable

	≥1 acyclic (Observable)	0 acyclic (Unobservable)
Permitted	OK	OK (Stricter than necessary)
Forbidden	BUG	OK



Modelling Microarchitectures: the *μSpec* DSL

Microarchitecture

```
Axiom "PO_Fetch":
forall microops "i1",
forall microops "i2",
SameCore i1 i2 /\ ProgramOrder i1 i2 =>
  AddEdge ((i1, Fetch), (i2, Fetch), "PO").

Axiom "Execute_stage_is_in_order":
forall microops "i1",
forall microops "i2",
SameCore i1 i2 /\
 EdgeExists ((i1, Fetch), (i2, Fetch)) =>
   AddEdge ((i1, Execute), (i2, Execute), "PPO").
```

- Model microarchitecture in domain-specific language μSpec
- Similar to first-order logic:
  - forall, exists, AND (/\), OR (\/), NOT (~), implication (=>)
  - Can refer to nodes and edges in axioms

- Axioms are each a **partial** ordering on the events in an execution
- Graph-based analysis checks that these axioms correctly work ***together*** to uphold the requirements of the MCM

Implementation

- Backend written in Gallina
- Implements cycle-checking and analysis of μhb graphs using SMT solver-based approaches
  - Solver is also written in Gallina!
- **High performance**
  - Most tests run in seconds (some take a few minutes)
- Open source!
  - Links to github repositories at http://check.cs.princeton.edu

Bugs Found so far..

- So far, tools have helped find bugs in:
  - Widely-used research simulator
  - Lazy coherence protocol
  - In-design commercial processors
  - RISC-V MCM specification
  - Compiler mappings from C11 atomics to Power and ARMv7
  - IBM XL C++ compiler (fixed in v13.1.5)
  - Open-source processor RTL

http://check.cs.princeton.edu/