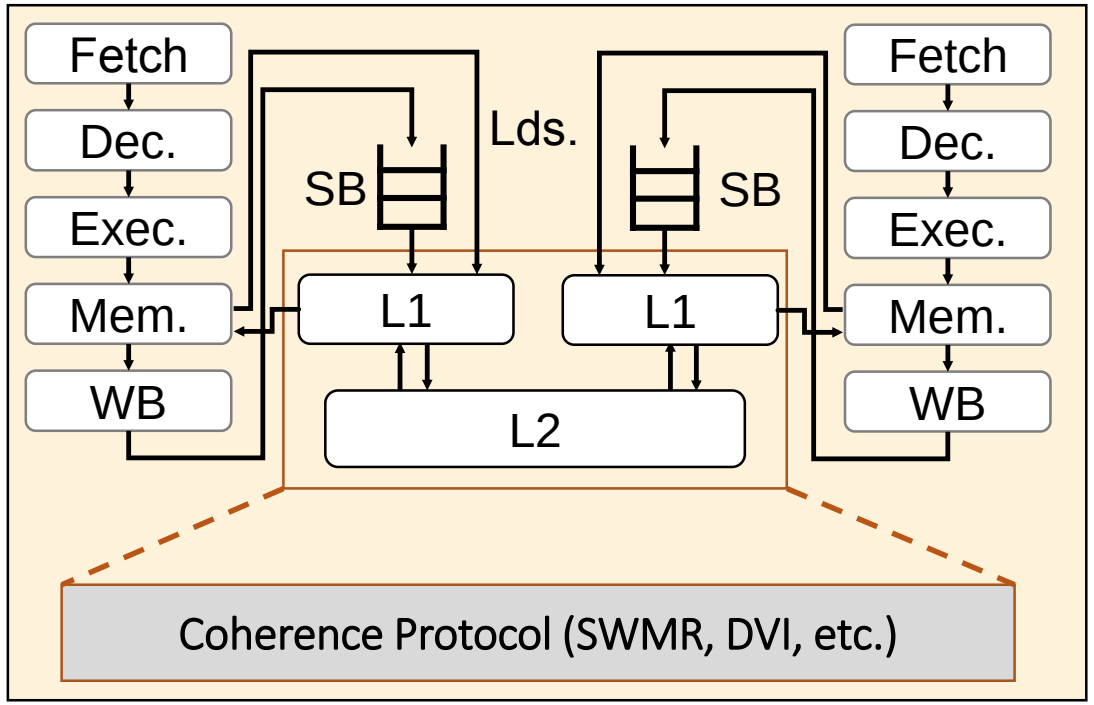
Background: Basics of our Approach

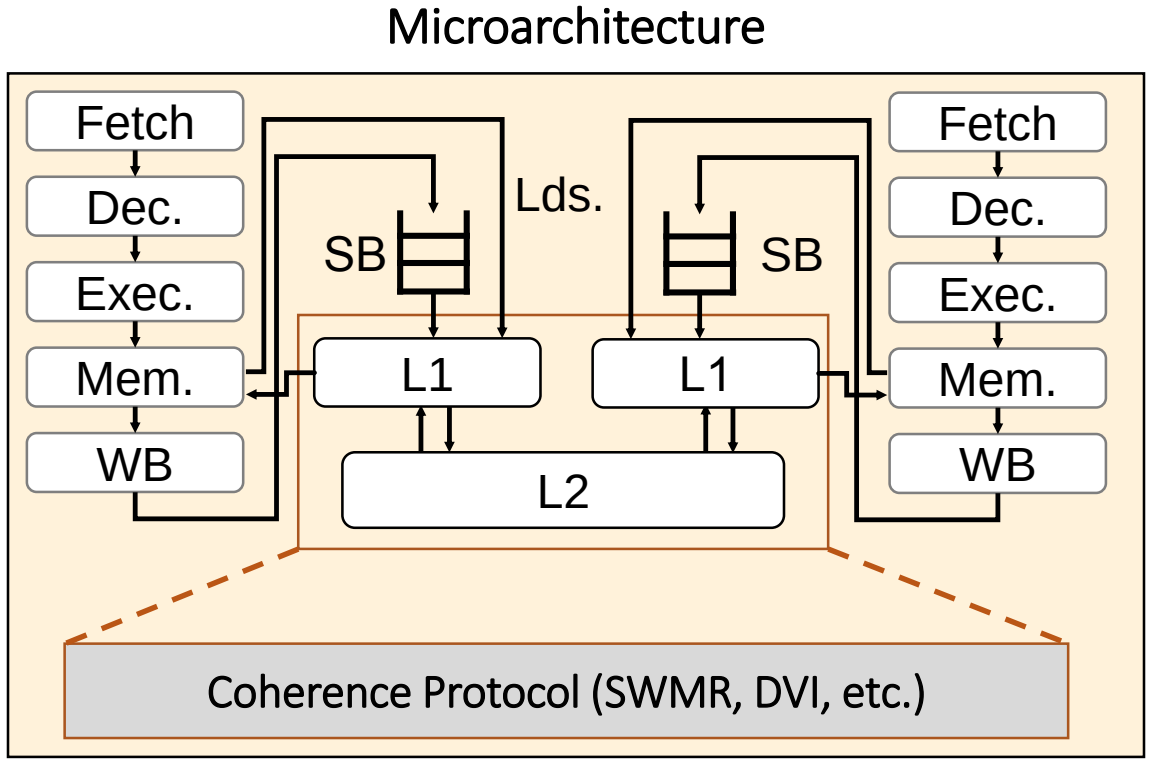
Microarchitectural Consistency Verification

- Microarch. enforces ISA-level MCM through many small orderings
  - In-order fetch/commit
  - FIFO store buffers
  - Coherence protocol
  - …
- Difficult to ensure that these orderings ***always*** enforce the required orderings



- Designs may also be complicated by optimizations (**speculative load reordering, early fence retirement, OoO execution**), or novel organization (**heterogeneity**)

Does hardware correctly implement ISA MCM?

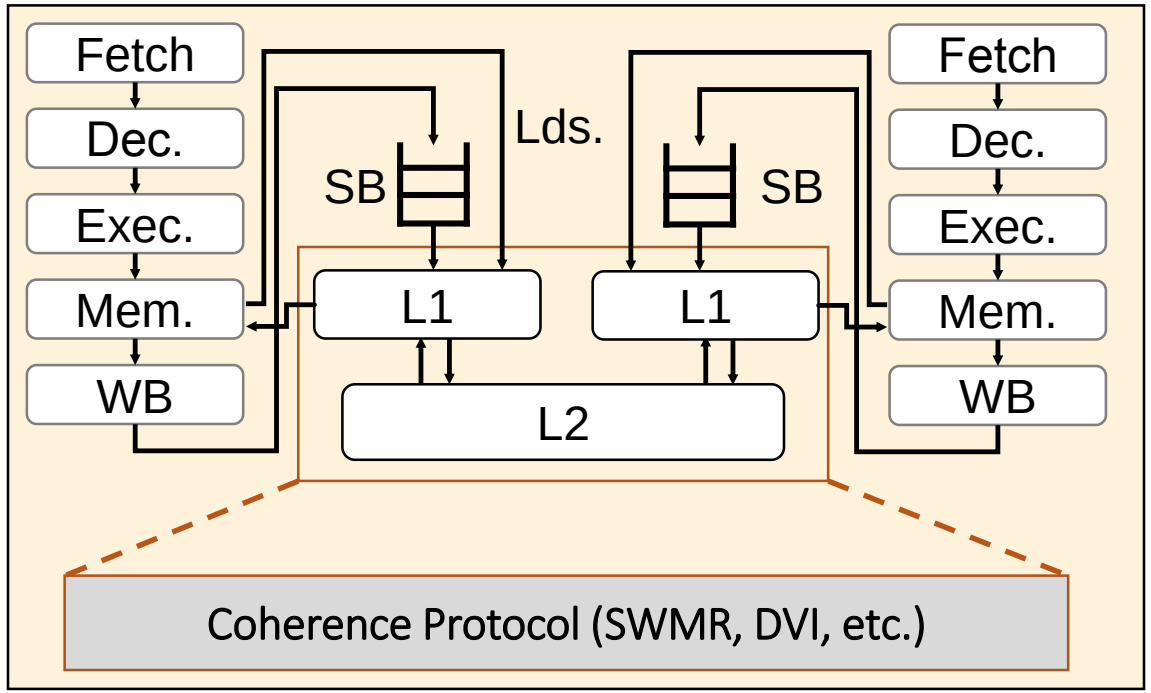


?

SC/TSO/RISC-V MCM?

Does hardware correctly implement ISA MCM?

Microarchitecture



?

SC/TSO/RISC-V MCM?
(for the litmus test)

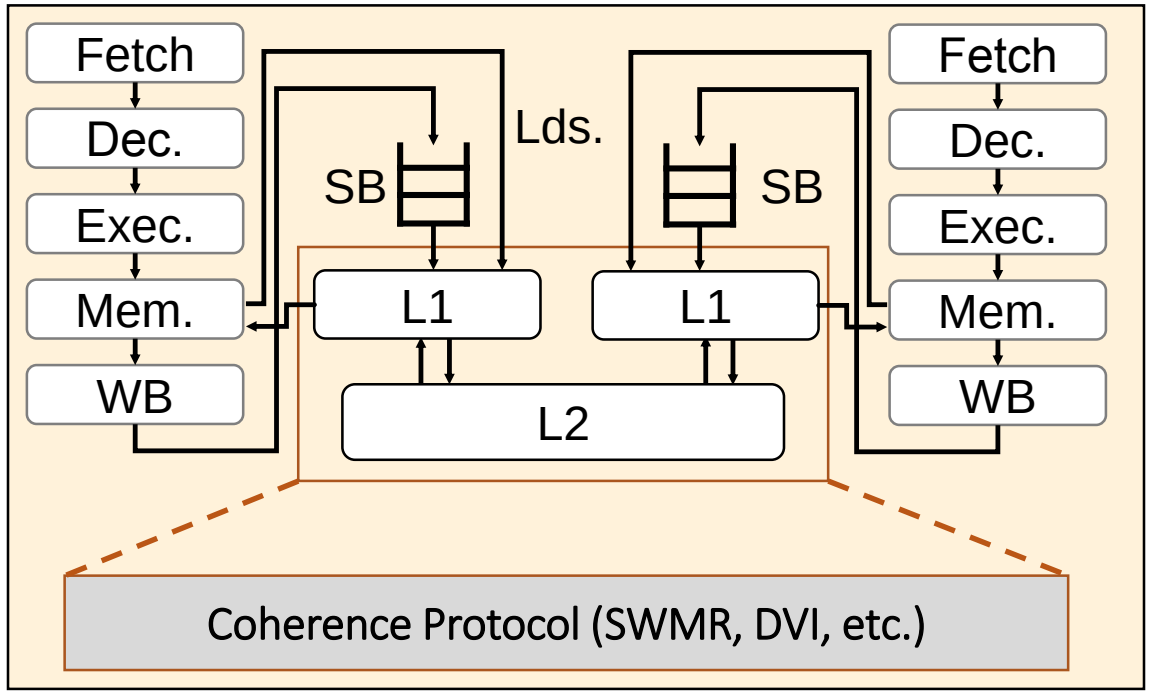
+

Litmus Test

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

Does hardware correctly implement ISA MCM?

Microarchitecture



?

SC/TSO/RISC-V MCM?
(for the litmus test)

+

Litmus Test

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

Instruction level analysis of litmus test

Microarch. analysis

	Observable	Unobservable
Permitted	OK	OK
Forbidden	BUG	OK

Does hardware correctly implement ISA MCM?

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), "").
```

?

SC/TSO/RISC-V MCM?
(for the litmus test)

+

Litmus Test

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

Instruction level analysis of litmus test

Microarch. analysis

	Observable	Unobservable
Permitted	OK	OK
Forbidden	BUG	OK

Verifying a Single Litmus Test with the Check Suite

Microarchitecture Specification in *μSpec* DSL

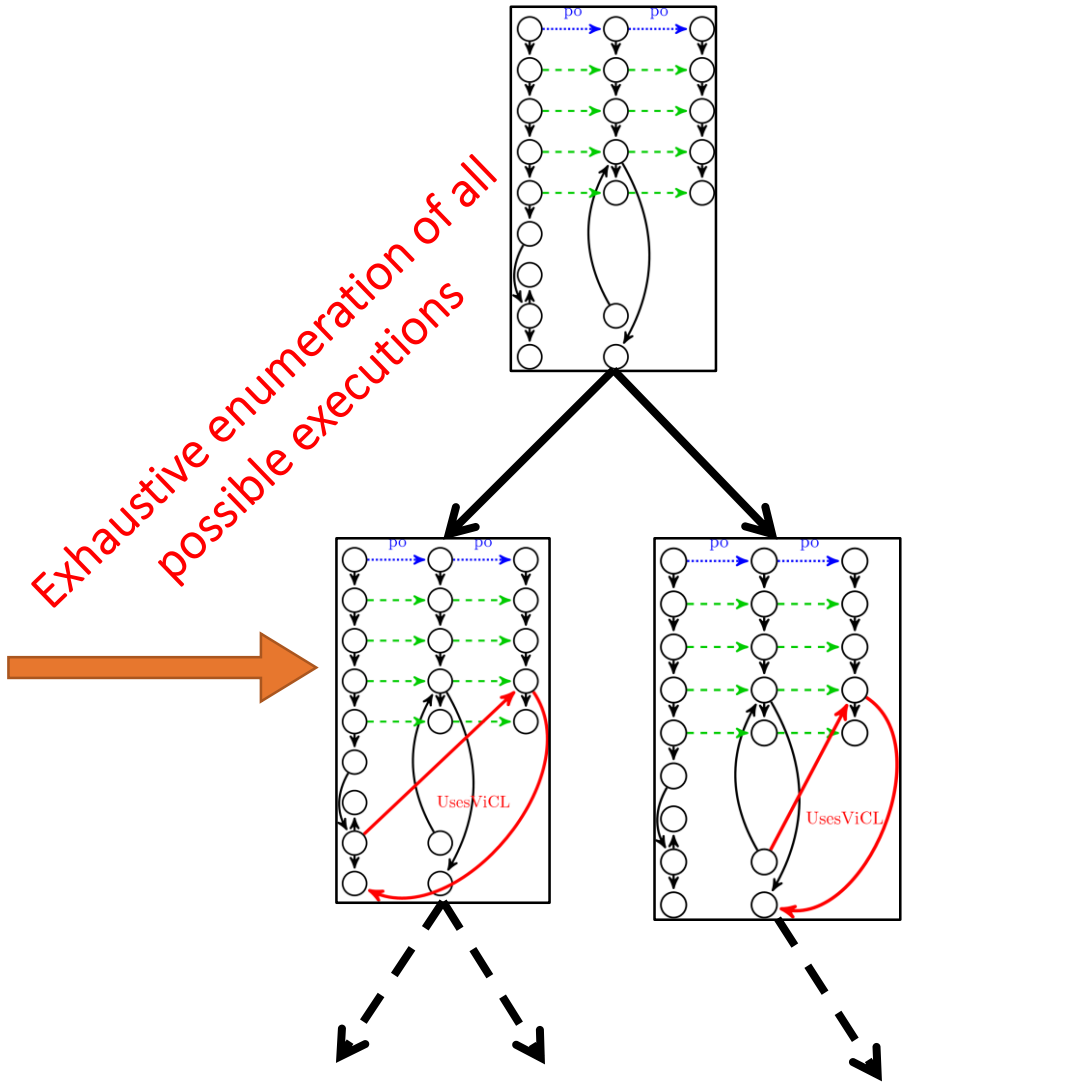
```
Axiom "PO_Fetch":
forall microops "i1",
forall microops "i2",
SameCore i1 i2 /\ ProgramOrder i1 i2 =>
  AddEdge ((i1, Fetch), (i2, Fetch), "PO").

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), "").
```

+

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	

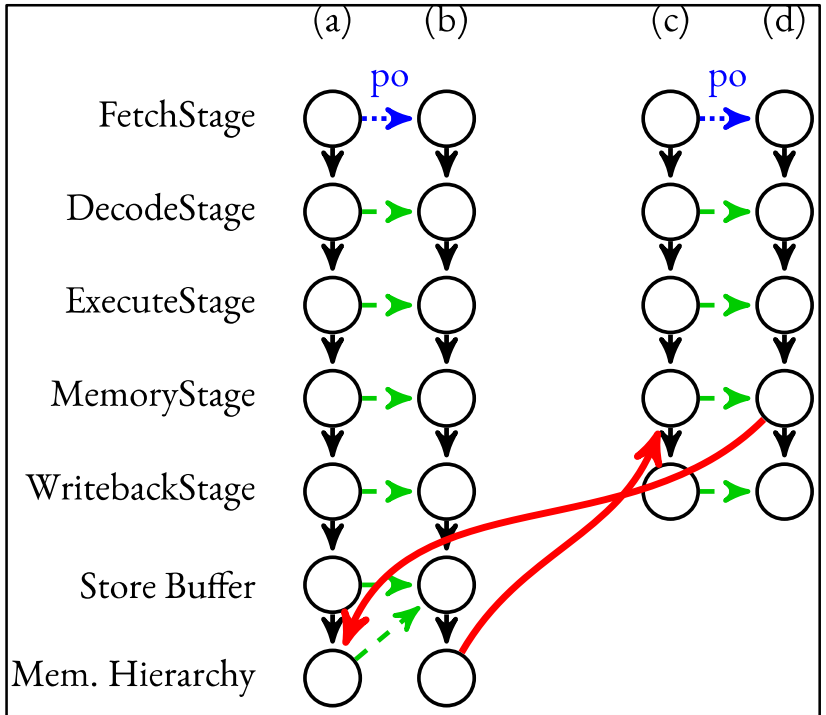


Microarchitectural happens-before (μhb) graphs

Microarchitectural Consistency Verification with Check

- **Early stage**, **design-time** verification
- Key Idea: Model executions as **μhb graphs**
  - **Nodes:** Microarchitectural events or pipeline stages
  - **Edges:** Happens-before relationships between nodes
- **Automatic exhaustive enumeration** of all possible litmus test executions
  - Cyclic Graph → Unobservable execution
  - Acyclic Graph → Observable execution

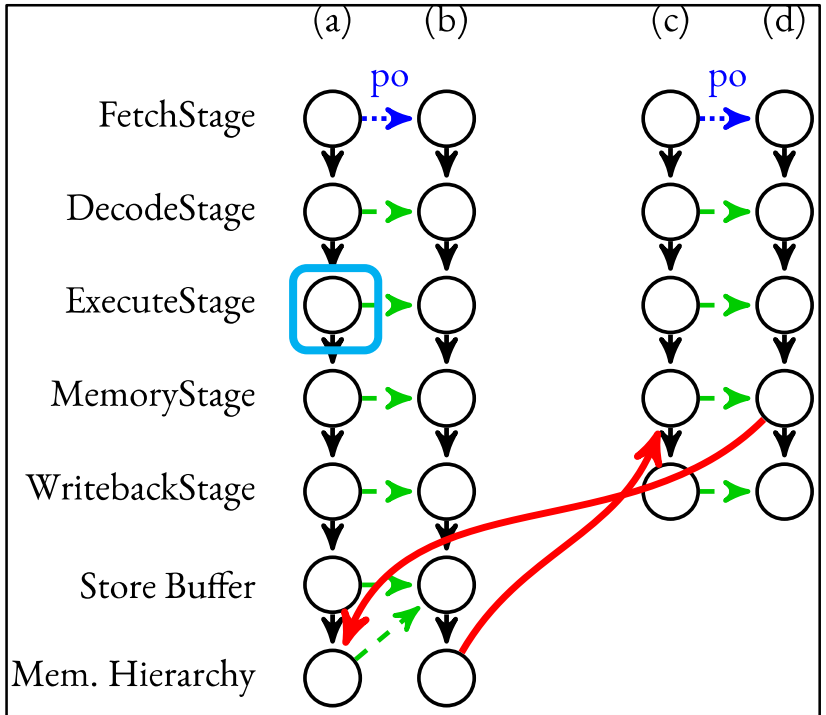
	≥1 acyclic (Observable)	0 acyclic (Unobservable)
Permitted	OK	OK (Stricter than necessary)
Forbidden	BUG	OK



Microarchitectural Consistency Verification with Check

- **Early stage**, **design-time** verification
- Key Idea: Model executions as **μhb graphs**
  - **Nodes:** Microarchitectural events or pipeline stages
  - **Edges:** Happens-before relationships between nodes
- **Automatic exhaustive enumeration** of all possible litmus test executions
  - Cyclic Graph → Unobservable execution
  - Acyclic Graph → Observable execution

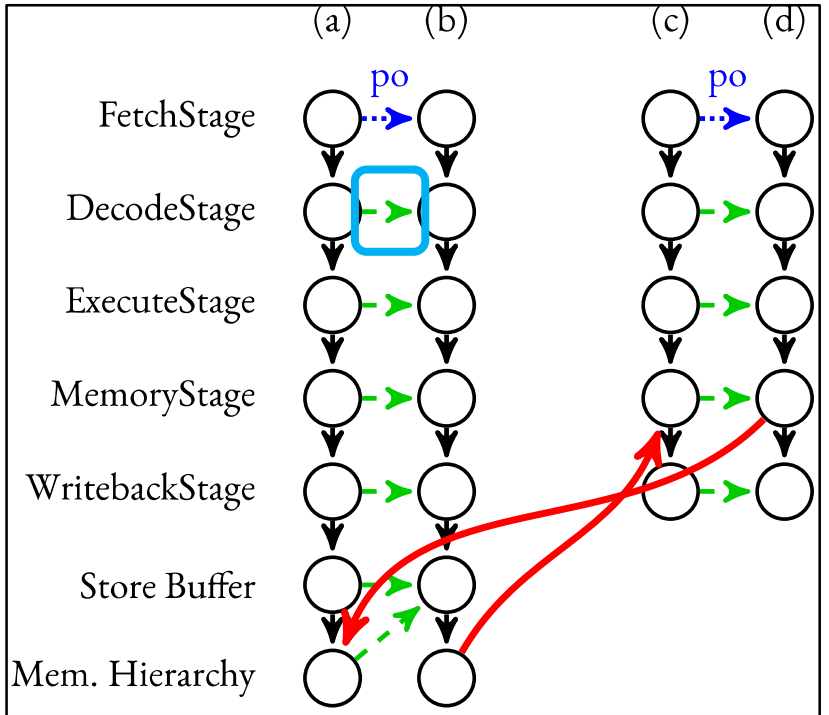
	≥1 acyclic (Observable)	0 acyclic (Unobservable)
Permitted	OK	OK (Stricter than necessary)
Forbidden	BUG	OK



Microarchitectural Consistency Verification with Check

- **Early stage**, **design-time** verification
- Key Idea: Model executions as **μhb graphs**
  - **Nodes:** Microarchitectural events or pipeline stages
  - **Edges:** Happens-before relationships between nodes
- **Automatic exhaustive enumeration** of all possible litmus test executions
  - Cyclic Graph → Unobservable execution
  - Acyclic Graph → Observable execution

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



Litmus test-based verification

- Litmus tests: small parallel programs (4-8 instrs)
  - Used to highlight memory model differences/features
  - Typically there is one non-SC outcome of interest (e.g. r1 = 1, r2 = 0 for mp)
- Different litmus tests associated with different ISA models
  - ISA memory model often characterized by their Permitted and Forbidden non-SC litmus test outcomes
  - e.g. TSO litmus test suite, Power litmus test suite, ARM litmus test suite
- Why litmus test-based verification?
  - Focus verification on the scenarios most likely to exhibit bugs, but…
  - …litmus test-based verification is **incomplete** (i.e. won't catch all bugs)
  - PipeProof [Manerkar et al. MICRO 2018] **solves this problem!** (all-program verification)

Some Example Litmus Tests

mp (Message Passing)

Thread 0	Thread 1
i1: Store [x] ← 1 i2: Store [y] ← 1	i3: r1 = Load [y] i4: r2 = Load [x]
SC Forbids: r1=1, r2=0	

co-mp (mp with one addr)

Thread 0	Thread 1
i1: Store [x] ← 1 i2: Store [x] ← 2	i3: r1 = Load [x] i4: r2 = Load [x]
SC Forbids: r1=2, r2=1, Mem[x] = 2	

sb (Store Buffering)

Thread 0	Thread 1
i1: Store [x] ← 1 i2: r1 = Load [y]	i3: Store [y] ← 1 i4: r2 = Load [x]
SC Forbids: r1=0, r2=0	

iriw (Independent Reads, Independent Writes)

Thread 0	Thread 1	Thread 2	Thread 3
i1: Store [x] ← 1	i2: Store [y] ← 1	i3: r1 = Load [x] i4: r2 = Load [y]	i5: r1 = Load [y] i6: r2 = Load [x]
SC Forbids: r1=1, r2=0, r3=1, r4=0			

Other things to note

- Check can handle heterogeneous parallelism (not covered today)
- Check can handle microarch. optimizations like speculative execution
- Different flows into and out of tools over the years
  - Originally: uspec DSL => custom solver (written in Gallina)
    - runtimes of seconds/minutes for a single test
  - More recently:
    - input specifications in Alloy (for CheckMate tool)
    - μspec compiled into Z3 formula (in progress)
- **Solver's search for a satisfying assignment == search for acyclic graph**