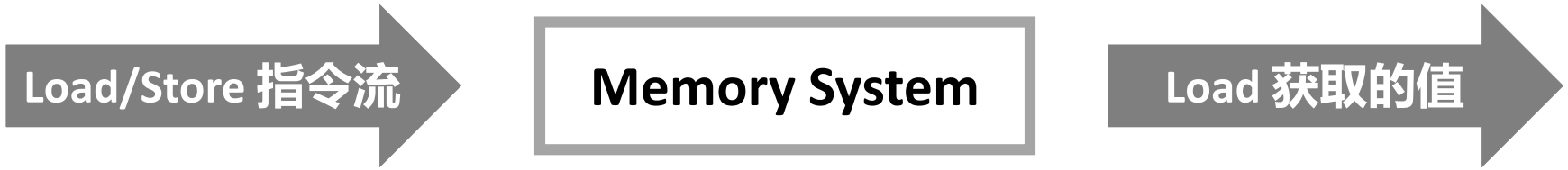
Pipe Mimic: RVWMO
内存一致性模型验证工具

周意可
华中科技大学
2021年6月25日

目录

什么是一致性模型?

- 对共享内存系统行为的定义



为何需要验证一致性模型?

- 描述了软件和硬件之间的接口

验证思路

- 对硬件模型进行形式化证明
- 执行大量测试程序

什么是一致性模型?

- 对共享内存系统行为的定义

为何需要验证一致性模型?

- 描述了软件和硬件之间的接口

验证思路

- 对硬件模型进行**形式化证明**
- 执行大量**测试**程序

Litmus Test

- Litmus Test 是一组并行程序，通过执行少量指令，比较指令的执行结果，对真实处理器或其模型的一致性特征进行测试

SB.litmus ①	
Core 1	Core 2
`A = 1` `print(B)`	`B = 1` `print(A)`

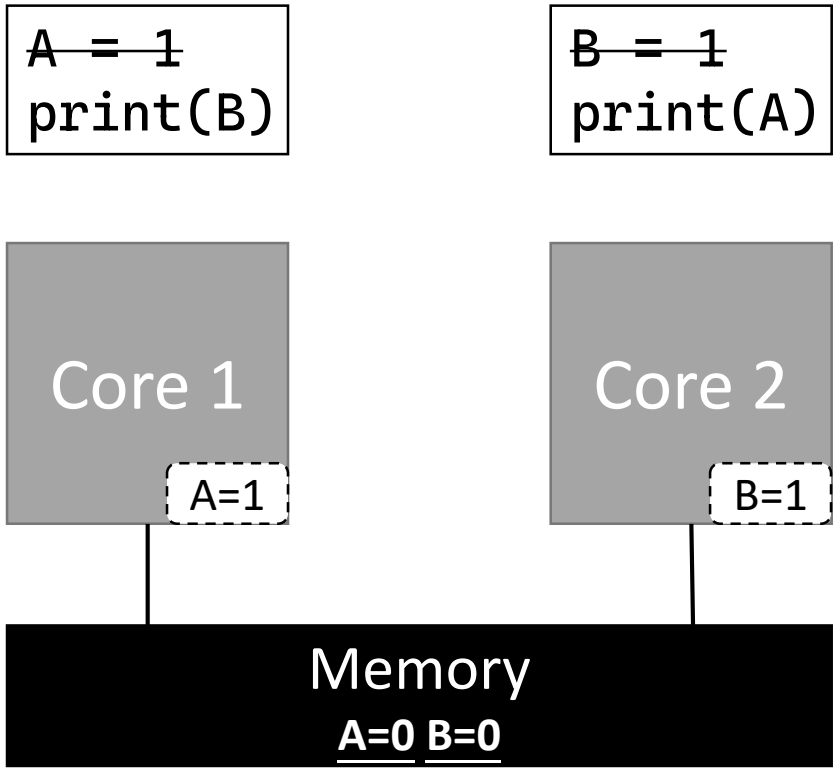
Under **TSO**:
A = 0, B = 0
is **Permitted**

- 若 Core a 写入的数据不能立刻被 Core b (b ≠ a) 观察到（例如处理器中存在 Store Buffer 结构），则程序可能输出 A=0 和 B=0

① 此处使用伪代码表示指令的含义

SB.litmus 执行结果 (1)

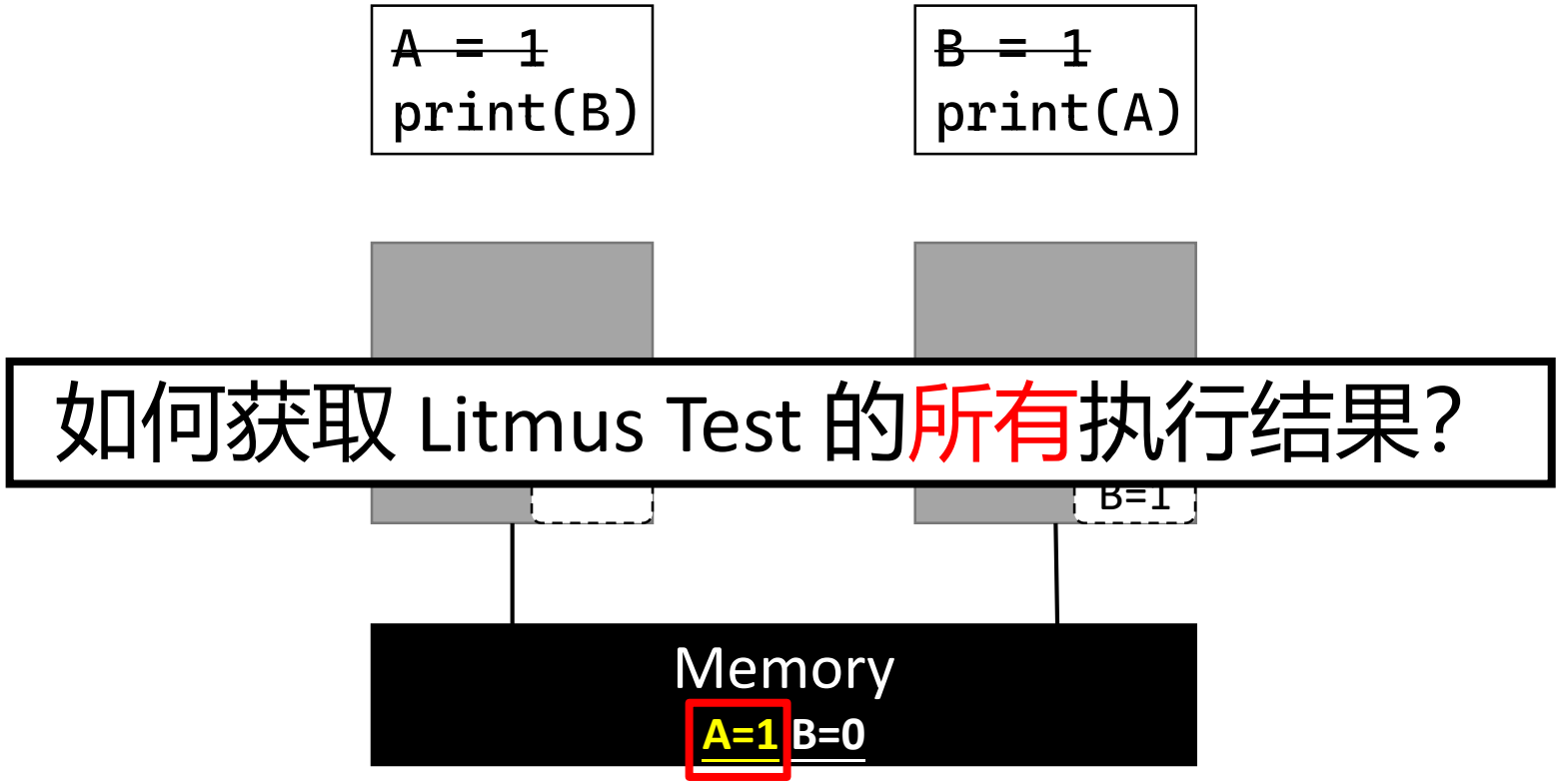
- 由于 Store Buffer 的存在，执行指令的核 (Core 1) 将先于其它核 (Core 2) 看到写入的结果 (A = 1)



① Memory Consistency Models: A Tutorial (https://www.cs.utexas.edu/~bornholt/post/memory-models.html)

SB.litmus 执行结果 (2)

- 另一种可能的结果是，Core 1 中的 Store Buffer 已经将结果写回内存
- 此时，print(B) 输出 0，print(A) 输出 1

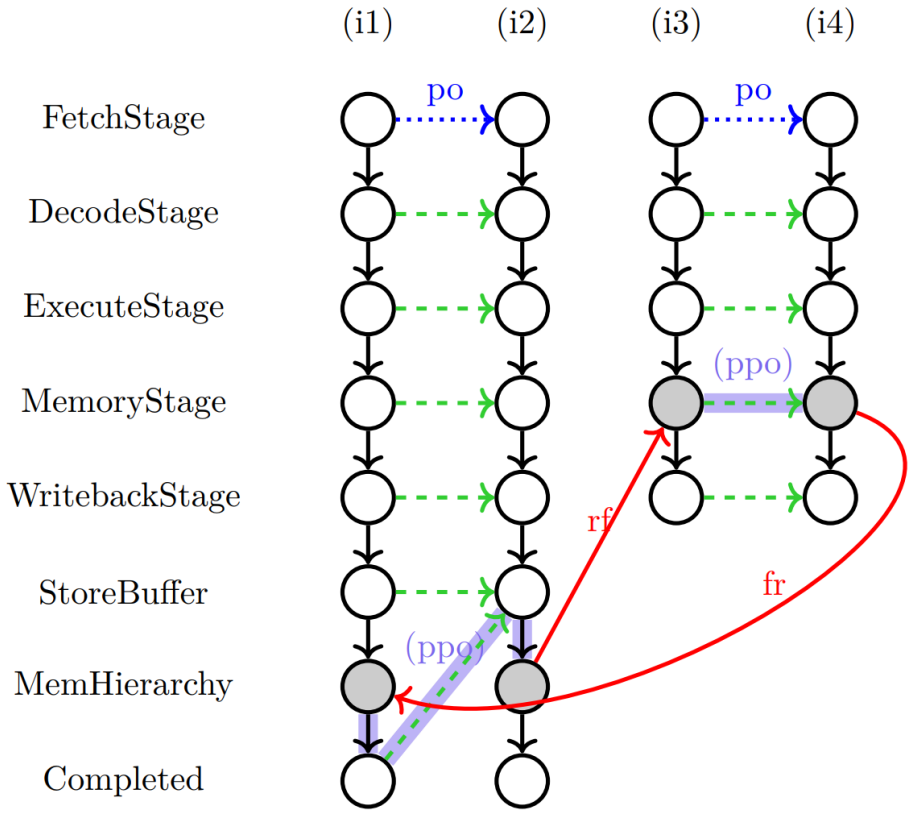


① Memory Consistency Models: A Tutorial (https://www.cs.utexas.edu/~bornholt/post/memory-models.html)

Pipe Check①: 核心算法

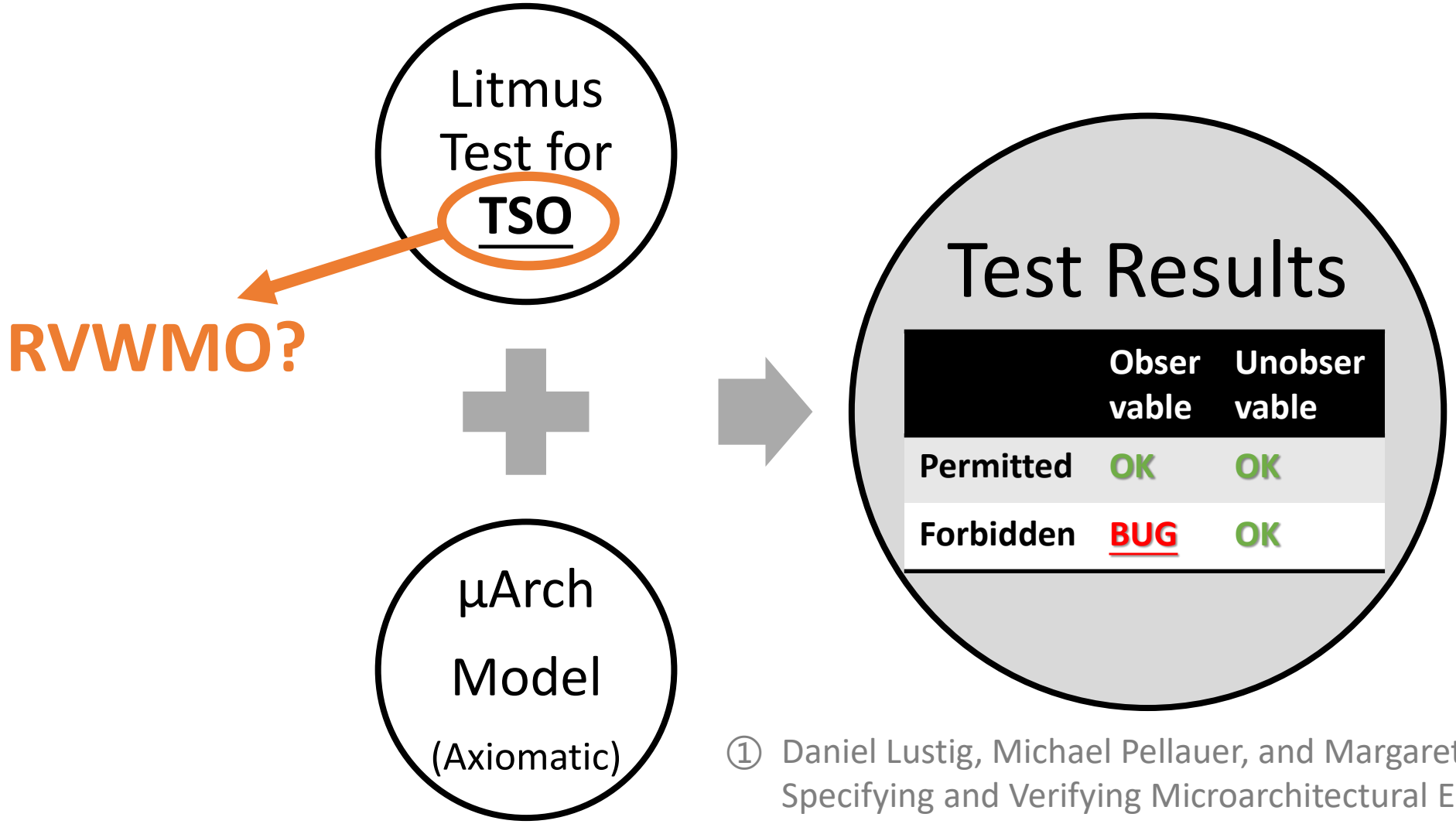
- 将指令对应读写操作的执行抽象为有向图
  - **点**——指令经过流水线的某一级
  - **边**——点对应事件的时间先后关系

- 枚举符合 Litmus Test 预期结果的情况下，所有读写操作可能的执行顺序
  - **有环图**——不可能出现 (Unobservable)
  - **无环图**——可能出现 (Observable)



① Daniel Lustig, Michael Pellauer, and Margaret Martonosi. "PipeCheck: Specifying and Verifying Microarchitectural Enforcement of Memory Consistency Models", *47th International Symposium on Microarchitecture (MICRO)*, Cambridge UK, December 2014.

Pipe Check①: 输入输出



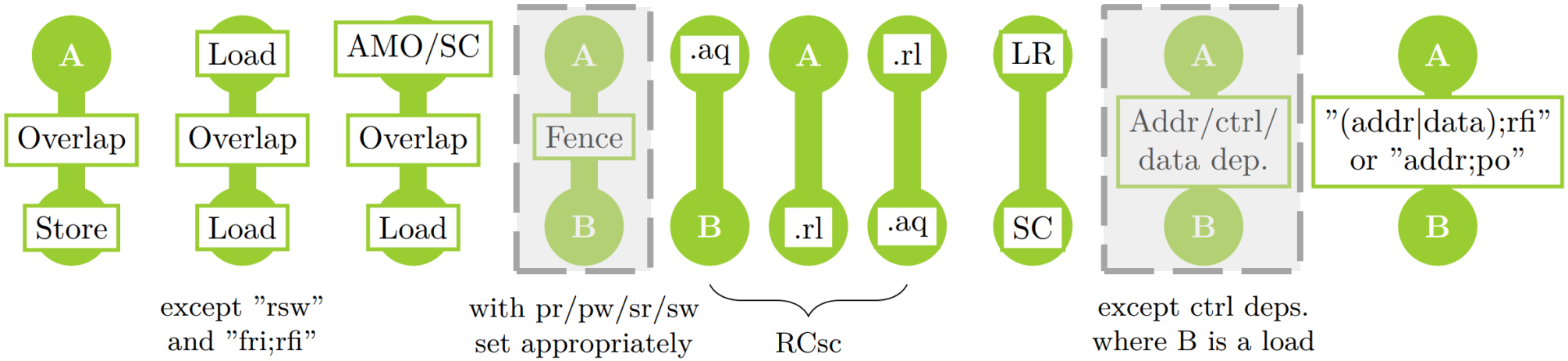
① Daniel Lustig, Michael Pellauer, and Margaret Martonosi. "PipeCheck: Specifying and Verifying Microarchitectural Enforcement of Memory Consistency Models", *47th International Symposium on Microarchitecture (MICRO)*, Cambridge UK, December 2014.

TSO → RVWMO

- **TSO** (Total Store Ordering) **x86**

Program Order	Preserved?
W→R	**No!**
Others	*Yes*

- **RVWMO** (RISC-V Weak Memory Ordering)① **RISC-V**



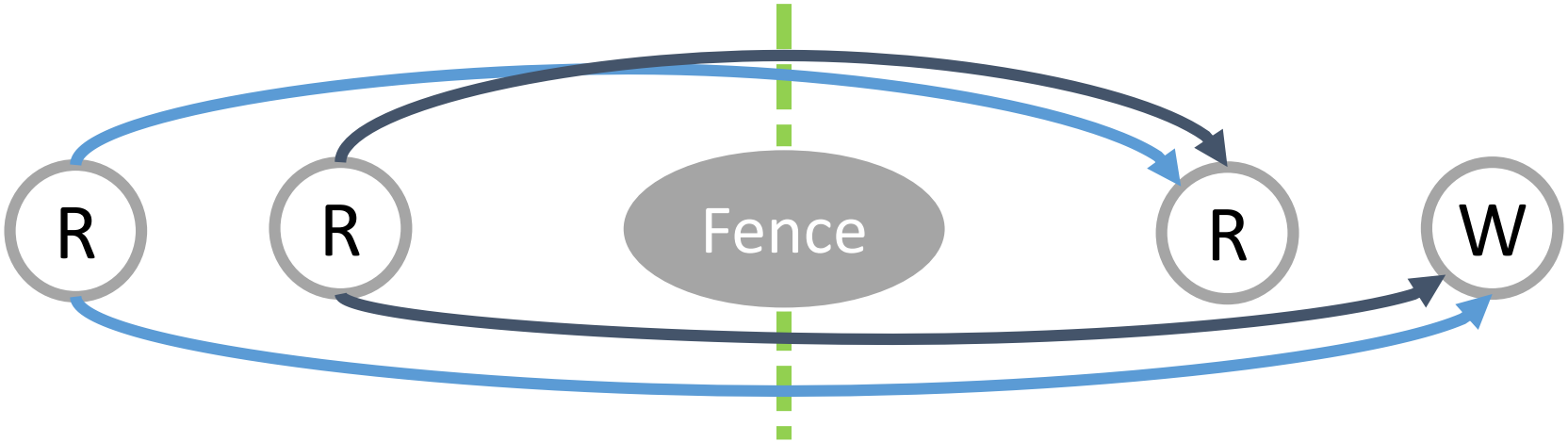
① Daniel Lustig. "RISC-V Memory Consistency Model Tutorial", May 2018.

目录

- 背景
- **设计与实现**
- 测试与分析
- 总结

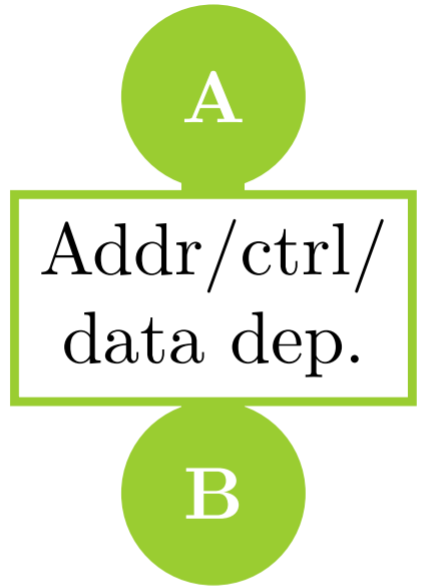
Fence 指令

- Fence 指令约束了可以乱序执行的指令的范围
  - …
  - MEMORY FENCE
  - 可交换顺序的读写指令
  - MEMORY FENCE
  - …
- 在有向图中添加对应的边来表示 Fence 前后的指令顺序得到了遵守



指令间的依赖关系

- **地址依赖** (Address dependency)
  - 指令 A 的执行结果决定指令 B 的**访问地址**
- **控制依赖** (Control dependency)
  - 指令 A 的执行结果决定指令 B **是否执行**
- **数据依赖** (Data dependency)
  - 指令 A 的执行结果决定指令 B **写入内存的值**



except ctrl deps.
where B is a load

检测

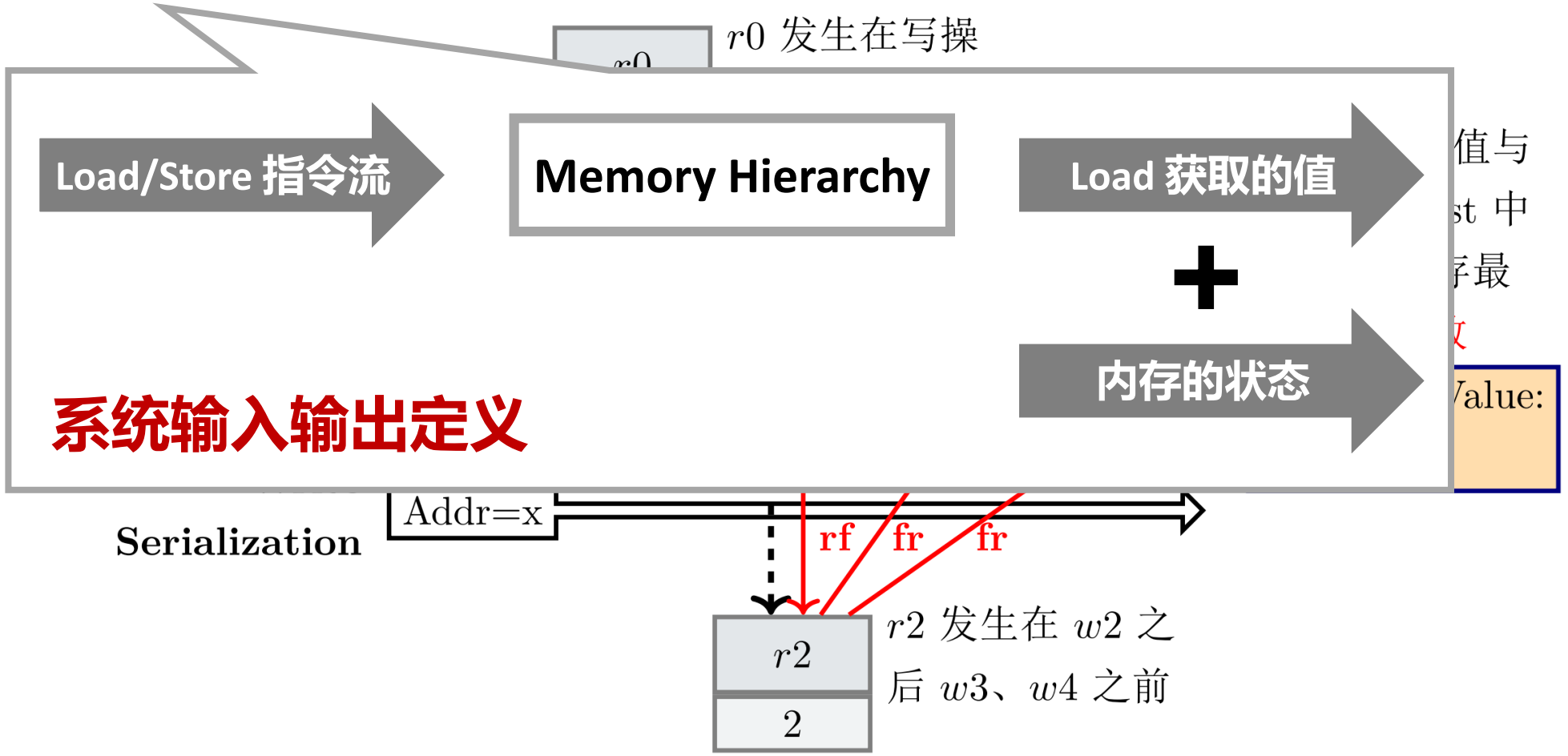
- 判断指令 B 的源寄存器是否与区间 [A, B) 中指令的目的寄存器相同

处理

- 指令 A 必须在指令 B 之前执行，在有向图中添加对应的边

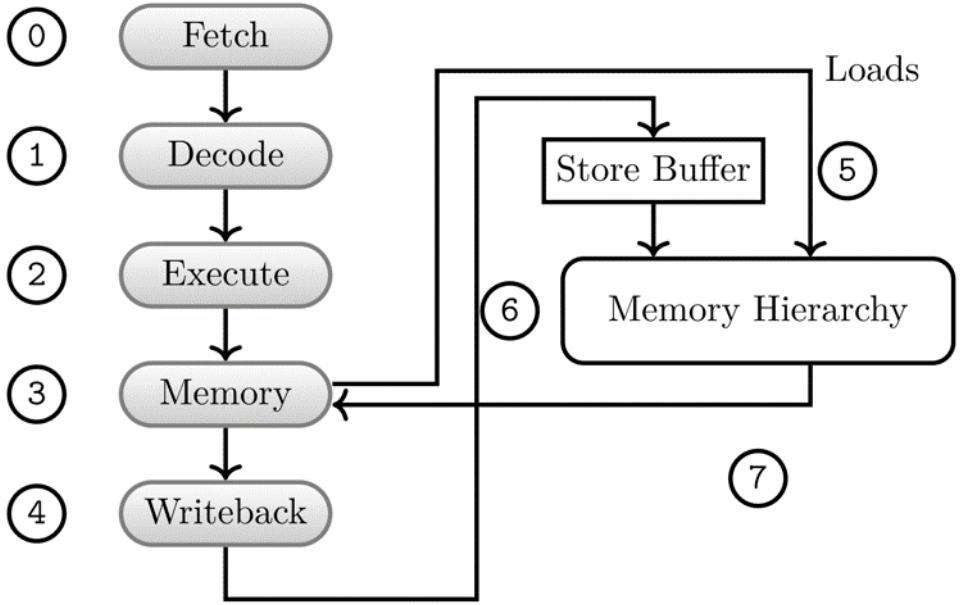
内存最终状态对写入顺序的约束

- 最后执行的写指令写入的值必须与 Litmus Test 给出的测试执行结束后**内存中的预期值**相同

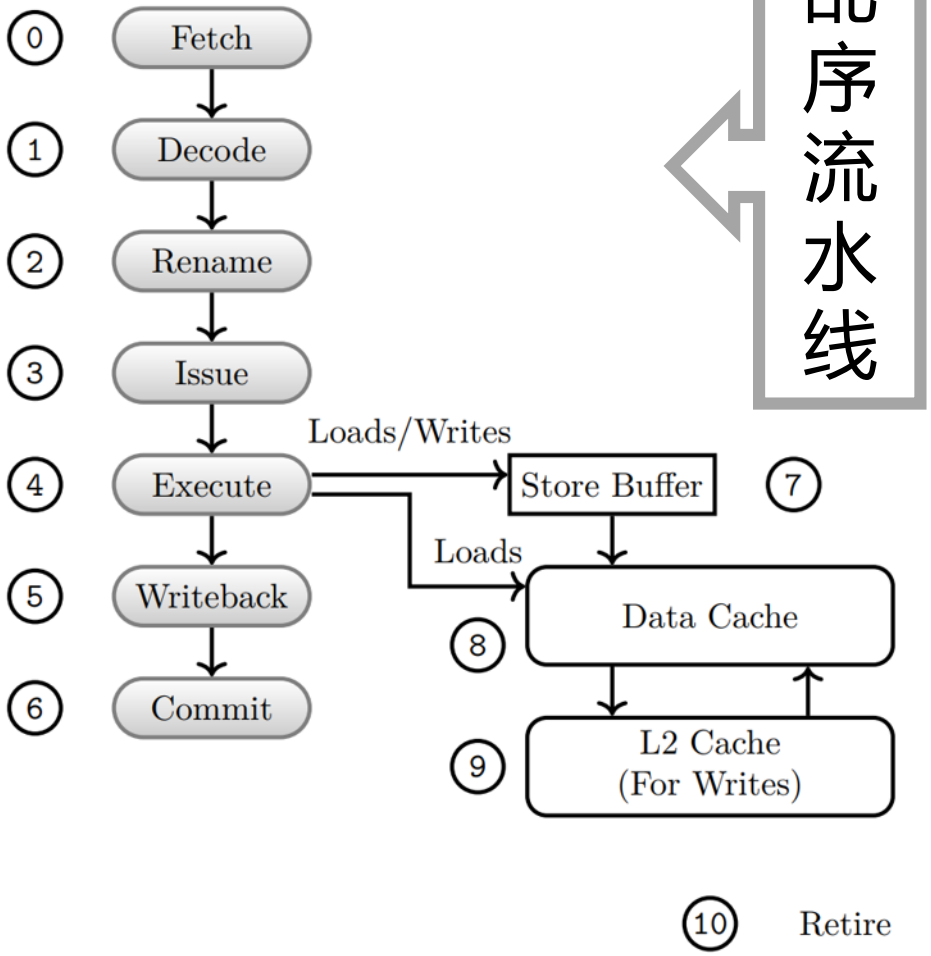


目录

测试输入 (1) ——流水线结构



顺序五级流水线



乱序流水线

测试输入 (2) —— Litmus Tests

- 从 GitHub 上获取官方 RVWMO Litmus Test 测试集
  - litmus-tests-riscv: litmus tests for the RISC-V concurrency architecture used by members of the RISC-V Memory Model Task Group
  - 对其中 **BASIC 2 THREADS** 子文件夹下的程序进行测试

- 测试集的特点
  - 1+N：1个基础 Litmus Test + 插入若干指令形成的衍生 Litmus Test
  - 衍生的 Litmus Test 对应 RVWMO 的特性

Litmus Test 输入解析①

- diy7 生成的 Litmus Test 文件

```
RISCV 2+2W+fence.rw.rw+po
"Fence.rw.rwdWW Wse PodWW Wse"
Cycle=Wse PodWW Wse Fence.rw.rwdWW
Relax=
Safe=Wse PodWW Fence.rw.rwdWW
Generator=diy7 (version 7.51+4(dev))
Prefetch=0:x=F,0:y=W,1:y=F,1:x=W
Com=Ws Ws
Orig=Fence.rw.rwdWW Wse PodWW Wse
{
0:x5=2; 0:x6=x; 0:x7=1; 0:x8=y;
1:x5=2; 1:x6=y; 1:x7=1; 1:x8=x;
}
P0           | P1           ;
sw x5,0(x6) | sw x5,0(x6) ;
fence rw,rw | sw x7,0(x8) ;
sw x7,0(x8) |              ;
exists
(x=2 /\ y=2)
```

- 语义信息

Core	Reg	Val
0	x5	2
1	x6	y

寄存器初始值

Core	Type	src1	src2	dst
0	store	x6	0	x5
0	fence	-	-	-

指令

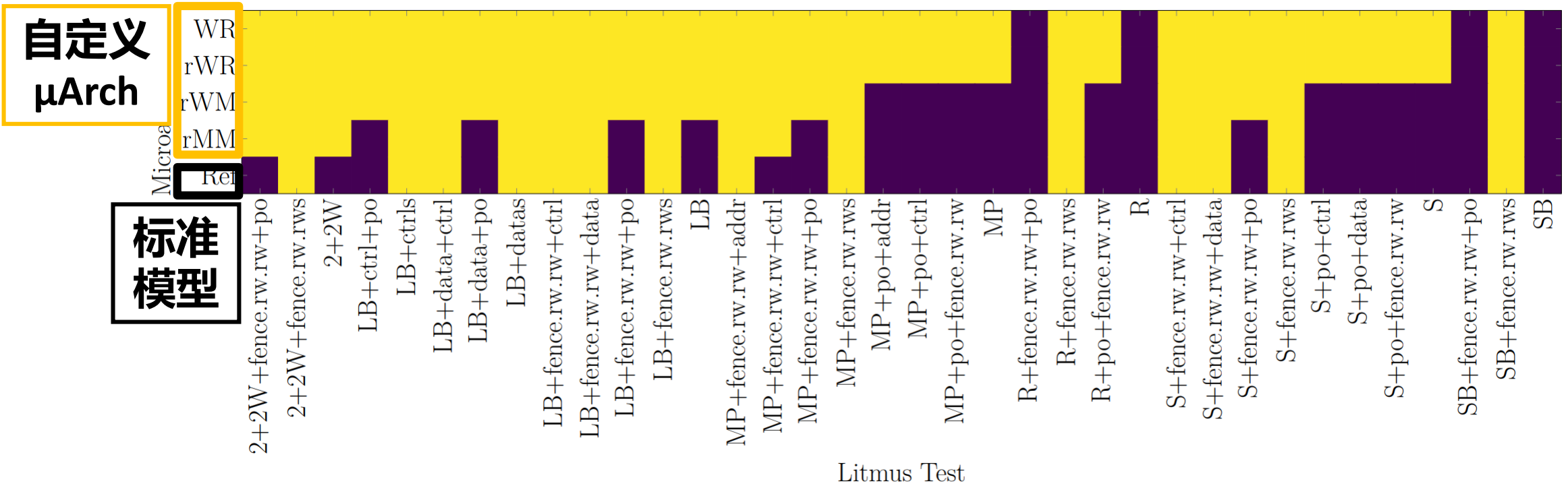
Core	Addr	Reg	Val
-	x	-	2

预期结果

① 使用 ANTLR 实现

Litmus Test 在不同微体系结构上的执行结果

- **紫色**：测试结果与 Litmus Test 给出的预期结果一致
  - 即 RVWMO 模型允许出现的结果在给定的微体系结构上都可能出现
- **黄色**：由于微体系结构的限制，某些允许的结果并不能出现

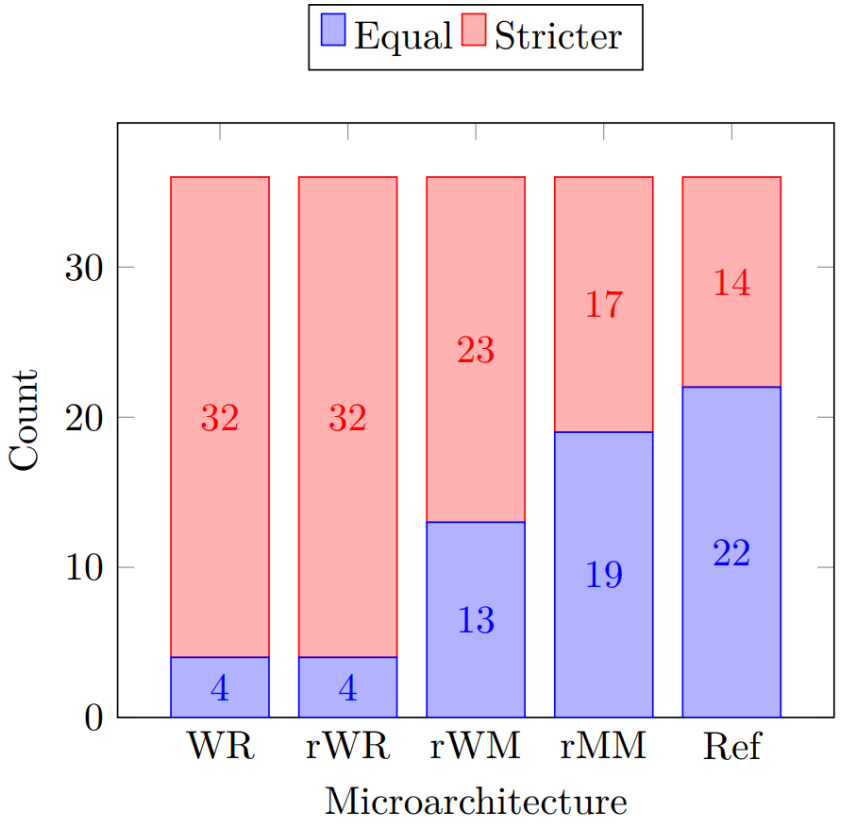


Litmus Test 结果体现体系结构特性

- 统计每种微体系结构上与 Litmus Test 预期结果一致的测试数目

①	Relaxed Program Order			Store Atomicity	
Model	W→R	W→W	R→M	MCA	rMCA
WR	✔			✔	
rWR	✔				✔
rWM	✔	✔			✔
rMM	✔	✔	✔		✔

低
微体系结构比模型更加严格，能乱序执行的指令更少



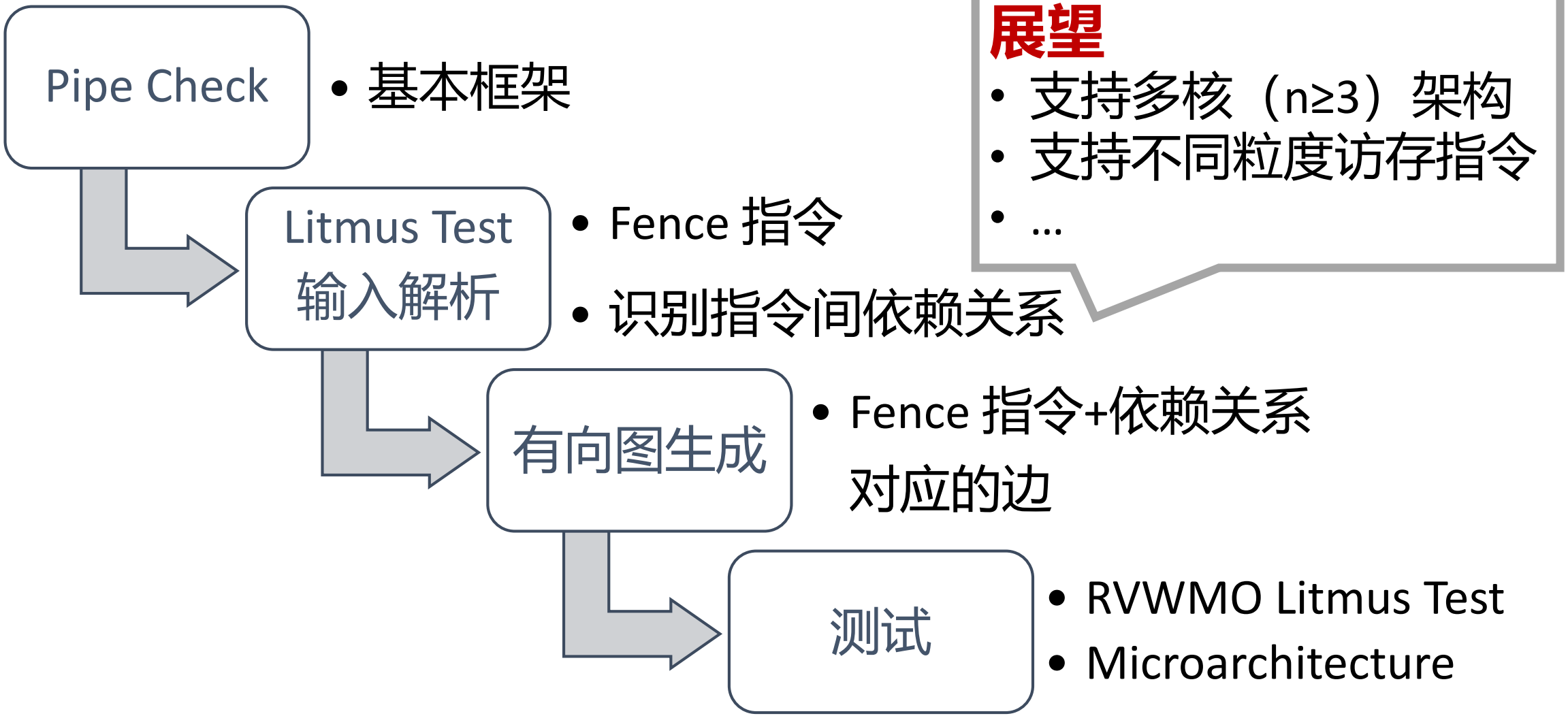
高
微体系结构与模型更加接近，能乱序执行的指令更多

① Caroline Trippel, Yatin Manerkar, Daniel Lustig, Michael Pellauer, and Margaret Martonosi. "TriCheck: Memory Model Verification at the Trisection of Software, Hardware, and ISA", 22nd International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), Xi'an, China., April 2017.

目录

- 背景
- 设计与实现
- 测试与分析
- **总结**

总结


Pipe Check
• 基本框架
Litmus Test
输入解析
• Fence 指令
• 识别指令间依赖关系
有向图生成
• Fence 指令+依赖关系
对应的边
测试
• RVWMO Litmus Test
• Microarchitecture
展望
• 支持多核（n≥3）架构
• 支持不同粒度访存指令
• …

谢谢！