中国科学院计算技术研究所
INSTITUTE OF COMPUTING TECHNOLOGY, CHINESE ACADEMY OF SCIENCES

构建“香山”开源芯片研发的基础设施

香山团队 徐易难

中国科学院计算技术研究所

2025 年 7 月

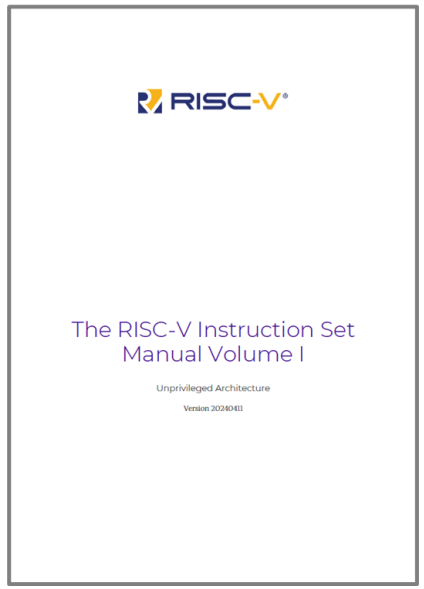
处理器的开发流程

工程开发

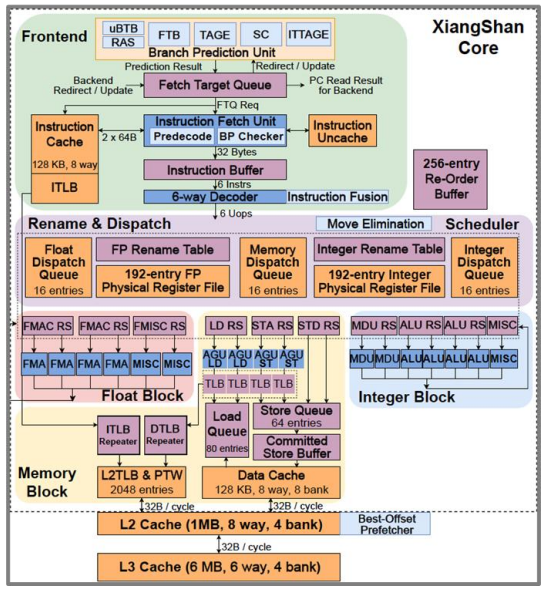
微架构实现

EDA工具

指令集手册



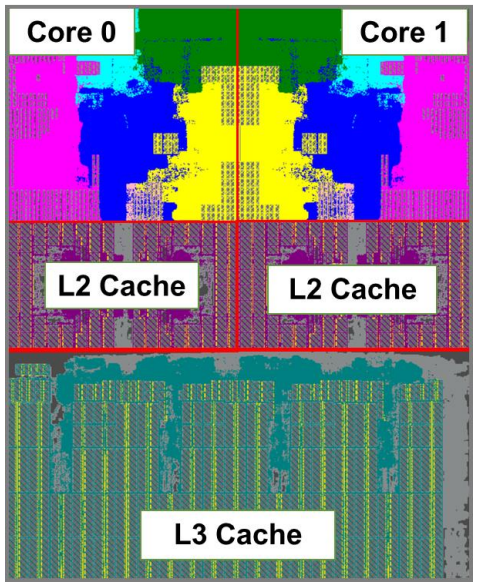
设计文档



RTL代码

```
class XSTop()(implicit p: Parameters) extends BaseXSSoc() with HasSoCParameter
{
  ResourceBinding {
    val width = ResourceInt(2)
    val model = "freechips,rocketchip-unknown"
    Resource(ResourceAnchors.root, "model").bind(ResourceString(model))
    Resource(ResourceAnchors.root, "compat").bind(ResourceString(model + "-dev"))
    Resource(ResourceAnchors.soc, "compat").bind(ResourceString(model + "-soc"))
    Resource(ResourceAnchors.root, "width").bind(width)
    Resource(ResourceAnchors.soc, "width").bind(width)
    Resource(ResourceAnchors.cpus, "width").bind(ResourceInt(1))
    def bindManagers(xbar: TLNexusNode) = {
      ManagerUnification(xbar.edges.in.head.manager.managers).foreach{ manager =>
        manager.resources.foreach(r => r.bind(manager.toResource))
      }
    }
    bindManagers(misc.l3_xbar.asInstanceOf[TLNexusNode])
    bindManagers(misc.peripheralXbar.asInstanceOf[TLNexusNode])
  }

  println(s"FPGASoC cores: $NumCores banks: $L3NBanks block size: $L3BlockSize bus size: $L3OuterBusWidth")

  val core_with_l2 = tiles.map(coreParams =>
    LazyModule(new XSTile()(p.alterPartial({
      case XSCoreParamsKey => coreParams
    })))
  )

  val l3cacheOpt = soc.L3CacheParamsOpt.map(l3param =>
    LazyModule(new HuanCun()(new Config((_, _, _) => {
      case HCCacheParamsKey => l3param
    })))
  )

  for (i <- 0 until NumCores) {
    core_with_l2(i).clint_int_sink := misc.clint.intnode
    core_with_l2(i).plic_int_sink :*= misc.plic.intnode
    core_with_l2(i).debug_int_sink := misc.debugModule.debug.dmOuter.dmOuter.intnode
    misc.plic.intnode := IntBuffer() := core_with_l2(i).beu_int_source
    misc.peripheral_ports(i) := core_with_l2(i).uncache
    misc.core_to_l3_ports(i) :=* core_with_l2(i).memory_port
  }

  l3cacheOpt.map(_.ctlnode.map(_ := misc.peripheralXbar))
  l3cacheOpt.map(_.intnode.map(int => {
    misc.plic.intnode := IntBuffer() := int
  }))
```

芯片版图

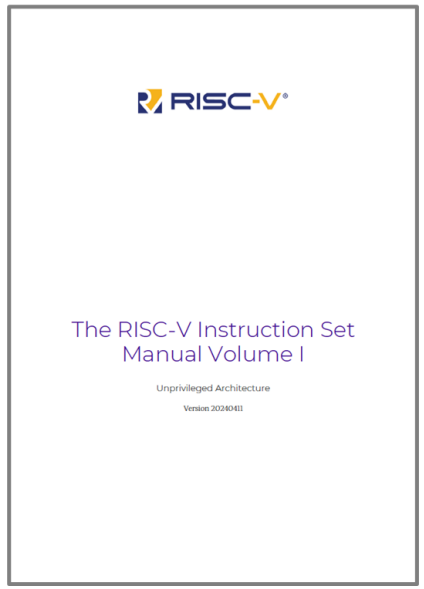


来源：包云岗, 孙凝晖. 开源芯片生态技术体系构建面临的机遇与挑战. 中国科学院院刊, 2022, 37(1): 24-29.

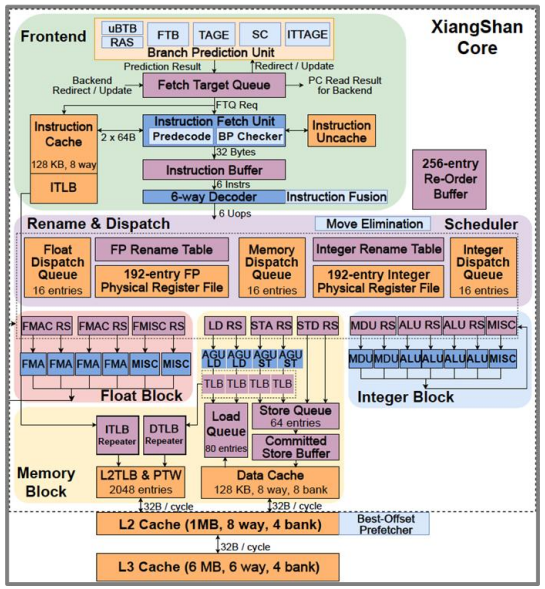
开源芯片生态

L3 工程开发 → L3 微架构实现 → L3 EDA工具

L1: 指令集手册



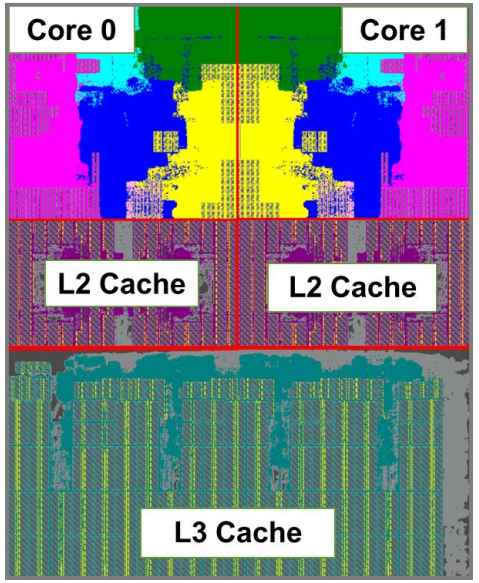
L2: 设计文档



L2: RTL代码

```
class XSTop()(implicit p: Parameters) extends BaseXSSoc() with HasSoCParameter
{
  ResourceBinding {
    val width = ResourceInt(2)
    val model = "freechips,rocketchip-unknown"
    Resource(ResourceAnchors.root, "model").bind(ResourceString(model))
    Resource(ResourceAnchors.root, "compat").bind(ResourceString(model + "-dev"))
    Resource(ResourceAnchors.soc, "compat").bind(ResourceString(model + "-soc"))
    Resource(ResourceAnchors.root, "width").bind(width)
    Resource(ResourceAnchors.soc, "width").bind(width)
    Resource(ResourceAnchors.cpus, "width").bind(ResourceInt(1))
    def bindManagers(xbar: TLNexusNode) = {
      ManagerUnification(xbar.edges.in.head.manager.managers).foreach{ manager =>
        manager.resources.foreach(r => r.bind(manager.toResource))
      }
    }
    bindManagers(misc.l3_xbar.asInstanceOf[TLNexusNode])
    bindManagers(misc.peripheralXbar.asInstanceOf[TLNexusNode])
  }

  println(s"FPGASoC cores: $NumCores banks: $L3NBanks block size: $L3BlockSize bus size: $L3OuterBusWidth")

  val core_with_l2 = tiles.map(coreParams =>
    LazyModule(new XSTile()(p.alterPartial({
      case XSCoreParamsKey => coreParams
    })))
  )

  val l3cacheOpt = soc.L3CacheParamsOpt.map(l3param =>
    LazyModule(new HuanCun()(new Config((_, _, _) => {
      case HCCacheParamsKey => l3param
    })))
  )

  for (i <- 0 until NumCores) {
    core_with_l2(i).clint_int_sink := misc.clint.intnode
    core_with_l2(i).plic_int_sink :*= misc.plic.intnode
    core_with_l2(i).debug_int_sink := misc.debugModule.debug.dmOuter.dmOuter.intnode
    misc.plic.intnode := IntBuffer() := core_with_l2(i).beu_int_source
    misc.peripheral_ports(i) := core_with_l2(i).uncache
    misc.core_to_l3_ports(i) :=* core_with_l2(i).memory_port
  }

  l3cacheOpt.map(_.ctlnode.map(_ := misc.peripheralXbar))
  l3cacheOpt.map(_.intnode.map(int => {
    misc.plic.intnode := IntBuffer() := int
  }))
```

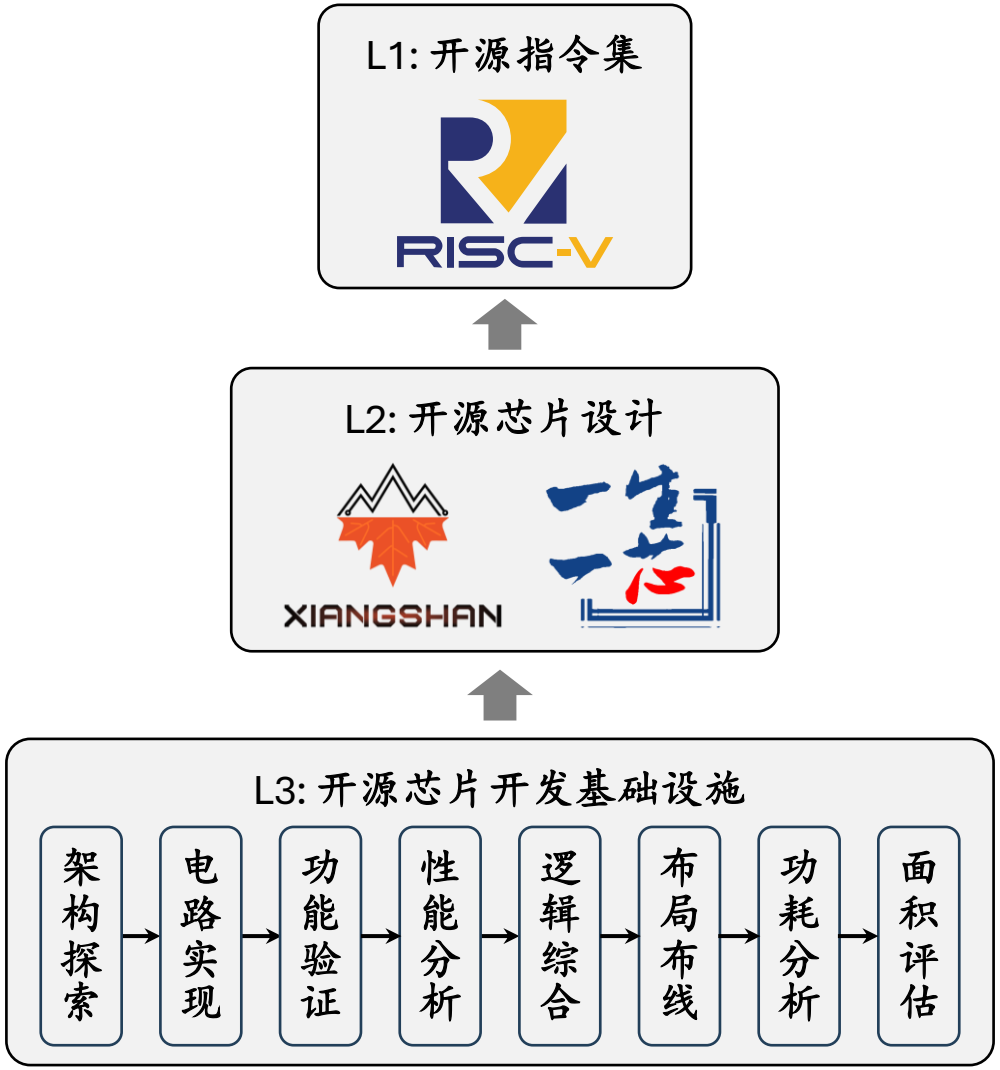
L2: 芯片版图



开源芯片技术体系 = (L1)开源指令集 + (L2)开源设计实现 + (L3)开源工具

来源：包云岗, 孙凝晖. 开源芯片生态技术体系构建面临的机遇与挑战. 中国科学院院刊, 2022, 37(1): 24-29.

开源芯片技术体系


L1: 开源指令集
RISC-V
L2: 开源芯片设计
XIANGSHAN
L3: 开源芯片开发基础设施
架构探索
电路实现
功能验证
性能分析
逻辑综合
布局布线
功耗分析
面积评估

“香山”开源芯片项目

开源软件生态

开源芯片生态

L1: 开放指令集

L2: 开源硬件设计

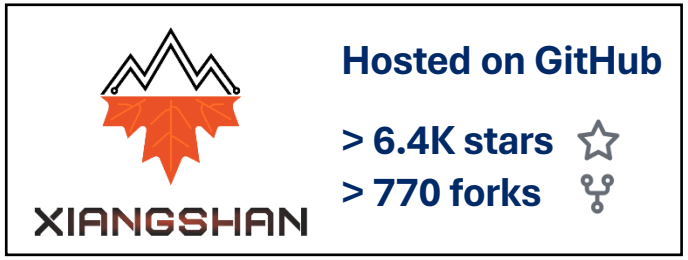
L3: 开源芯片研发基础设施

RISC-V®

XIANGSHAN

“香山”：开源高性能RISC-V处理器

- Highest performing open-source processor series by far
  - Open RTL design with comprehensive documentation
  - Open development tools and platform



Kunminghu (3rd gen.)

versus ARM Neoverse N2

- Designed for high performance
- Targeting server/data-center segment

- RVA-23 profile
- Leading RISC-V feature sets (H/V ext.)

Nanhu (2nd gen.)

versus ARM Cortex A76

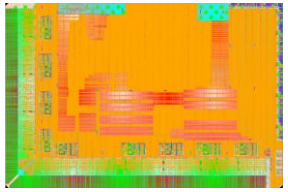
- Designed for power/area efficiency
- Targeting industry-control segment

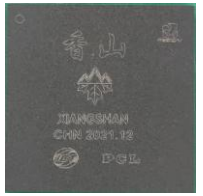
- RVA-20 profile
- Taped out and tested

- Managed by the Beijing Institute of Open Source Chip (BOSC), an NPO
  - Founding members from industry, academia, investors, law officers
  - Supported by tens of members

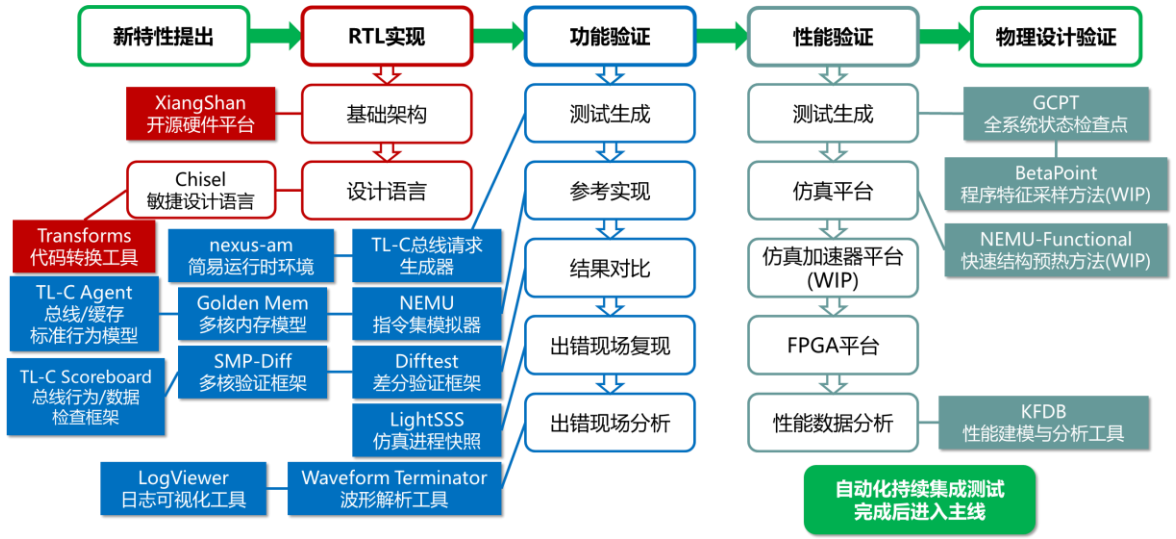
“香山”开源项目的冰山模型

- **香山核心价值**是构建一套**芯片敏捷设计基础设施**，缩短迭代优化周期
- **开源开放能力体系**，联合企业加速处理器研发节奏，支持按需快速定制芯片



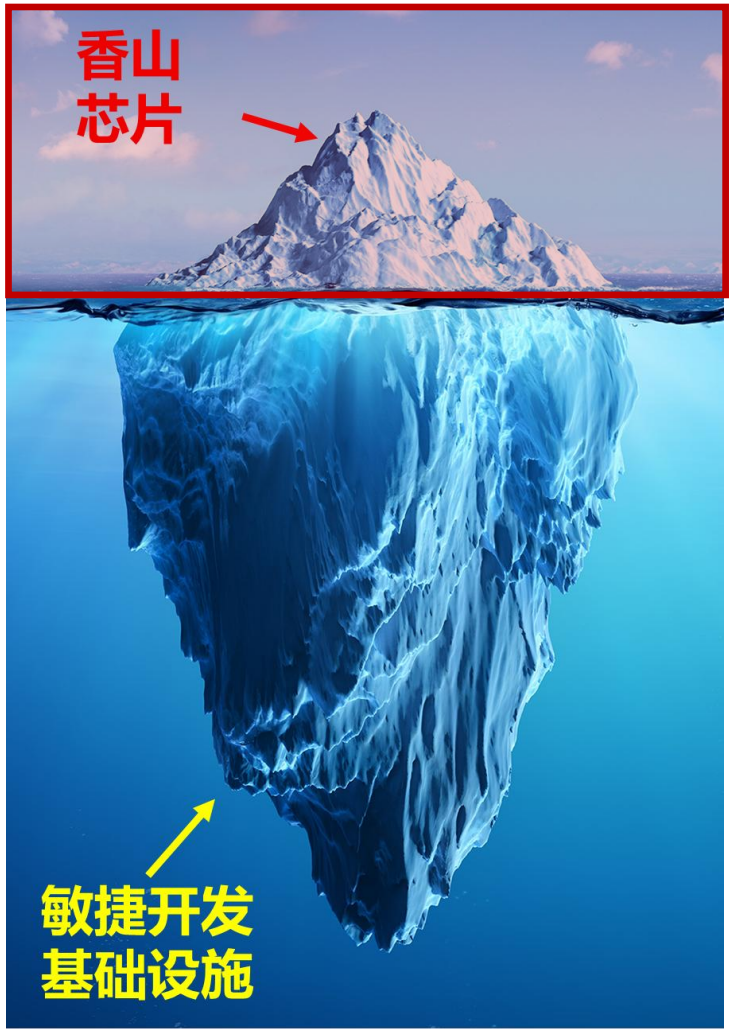


成果开源

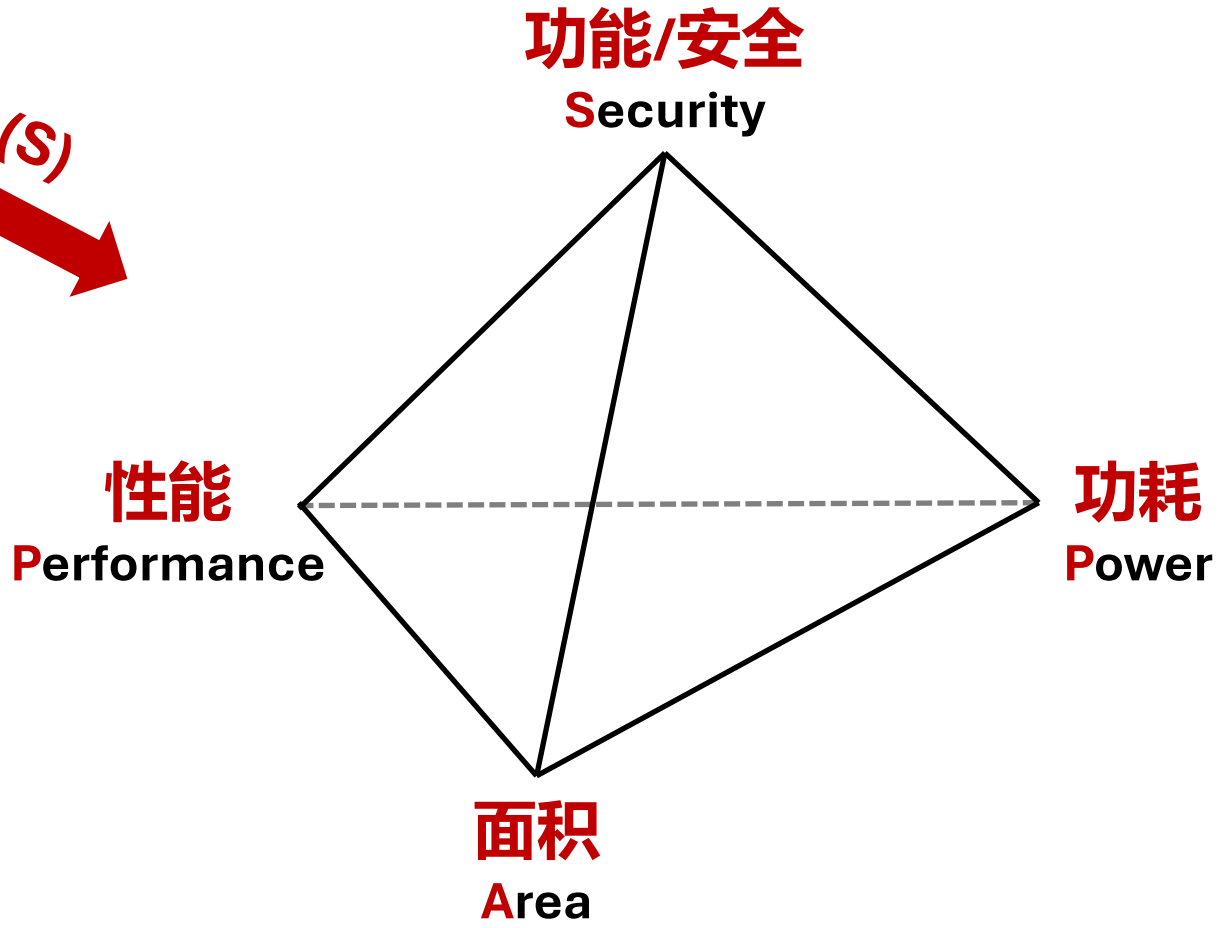


能力开放

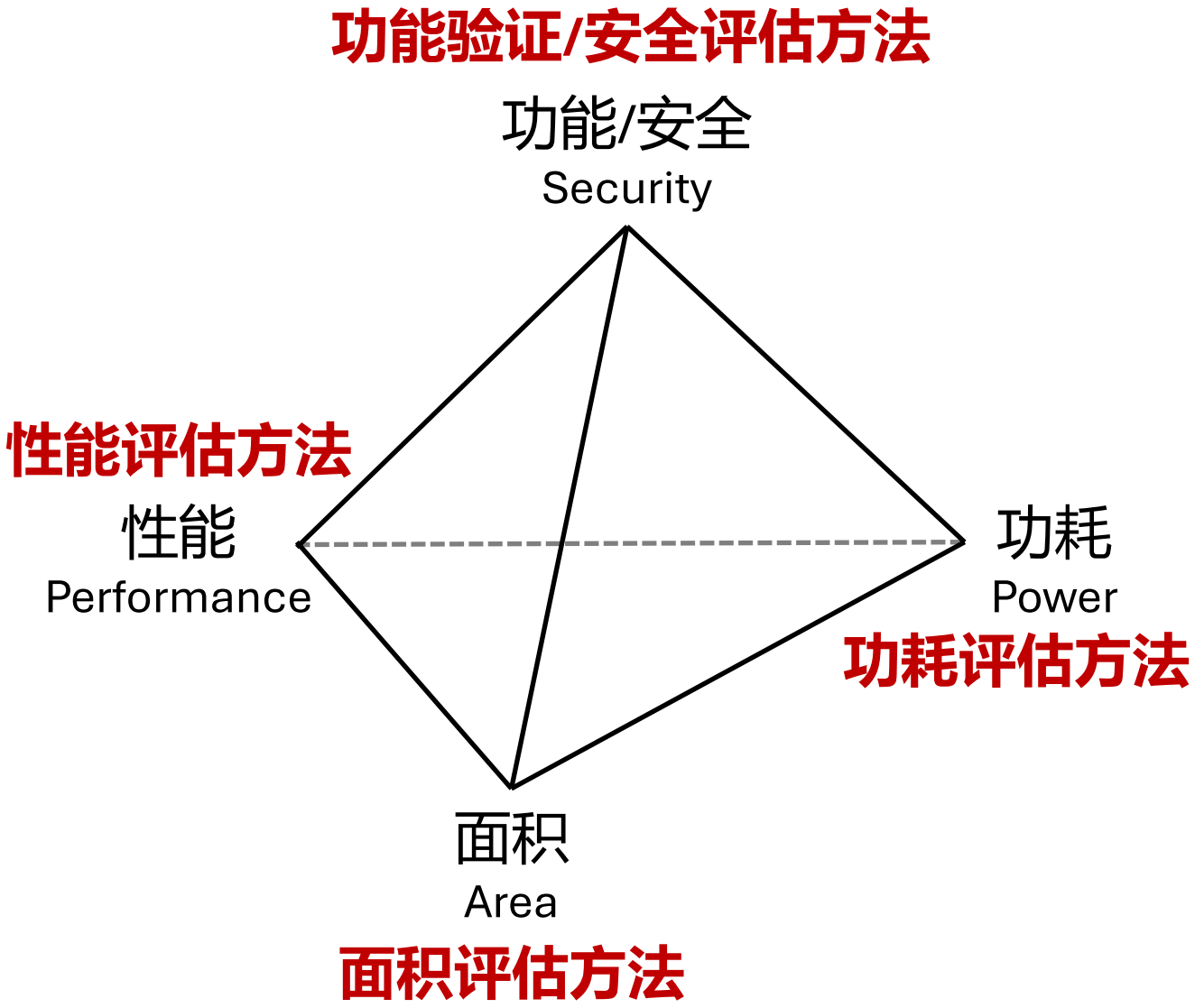
高性能 RISC-V 处理器的设计目标



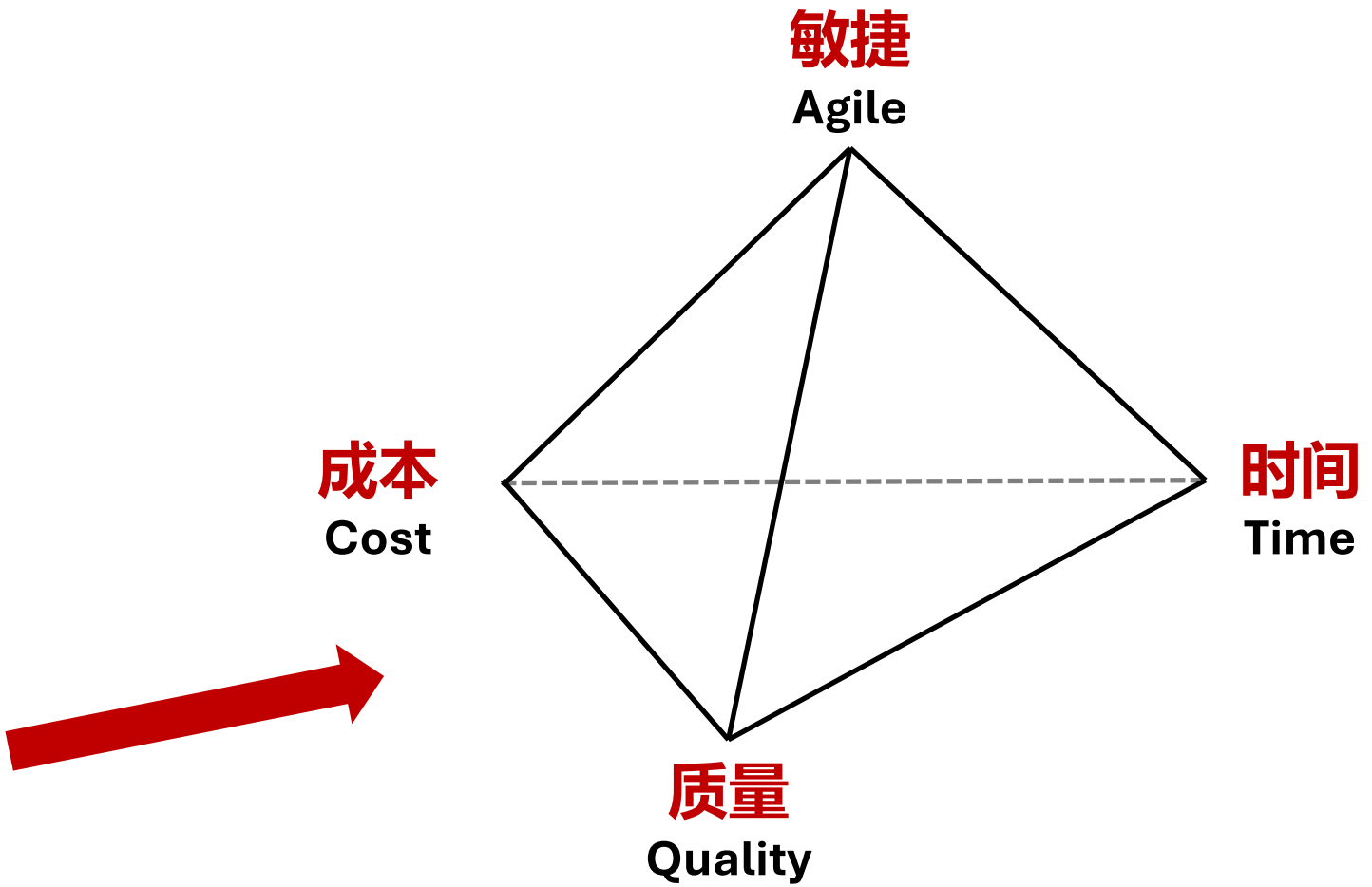
PPA(S)



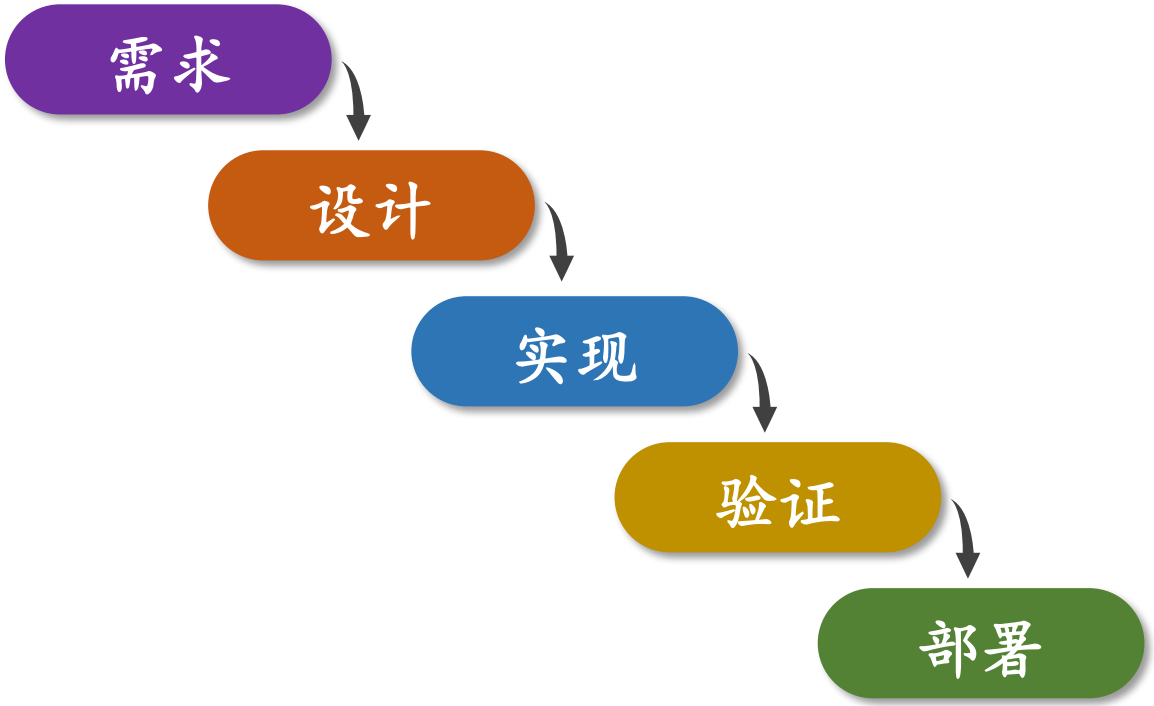
开发基础设施 (Infra)



开发基础设施 (Infra) 的设计目标



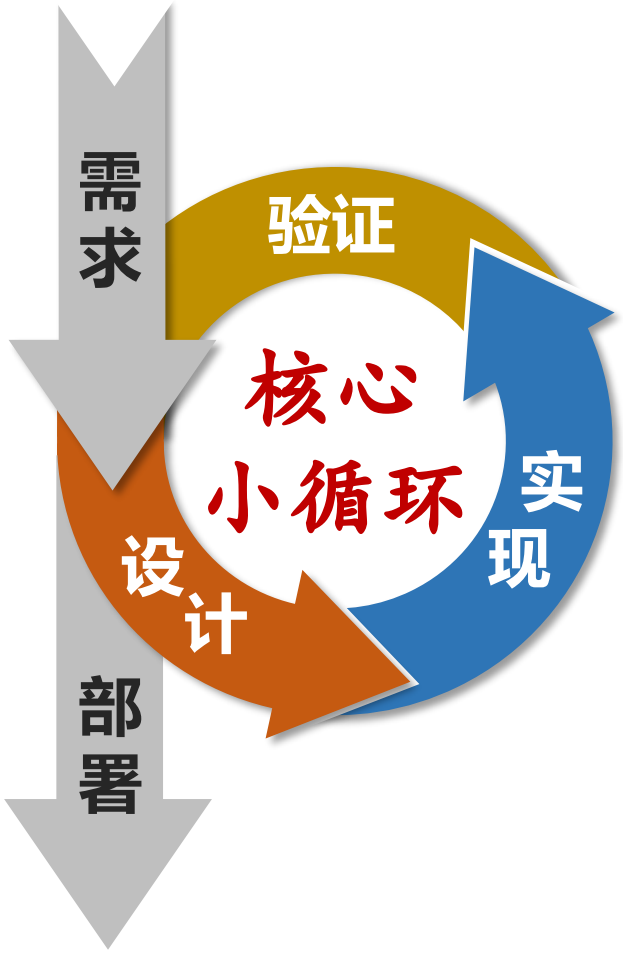
什么是敏捷：从敏捷开发说起



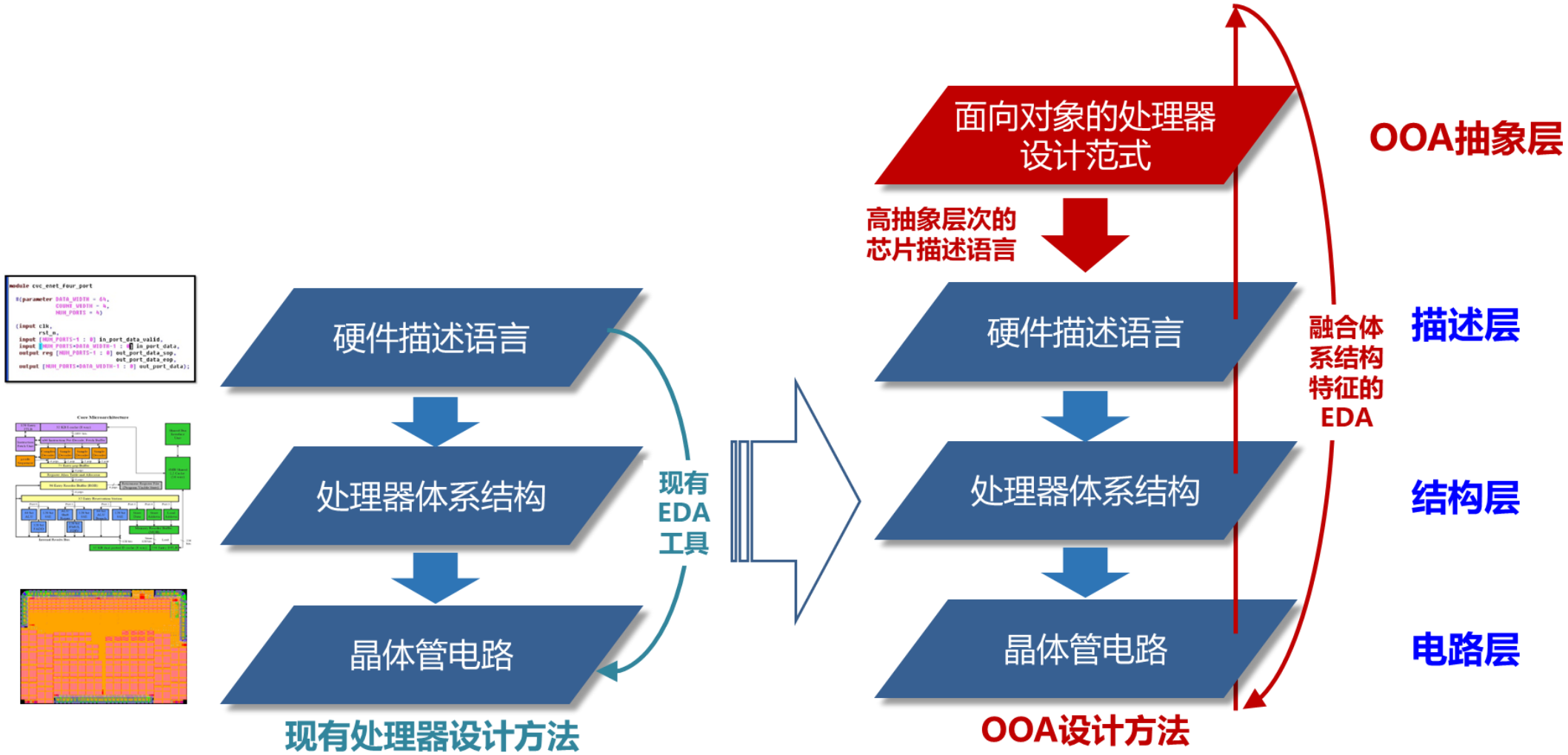
瀑布流开发模式

敏捷开发模式

芯片敏捷开发的特征①：高效的迭代小循环



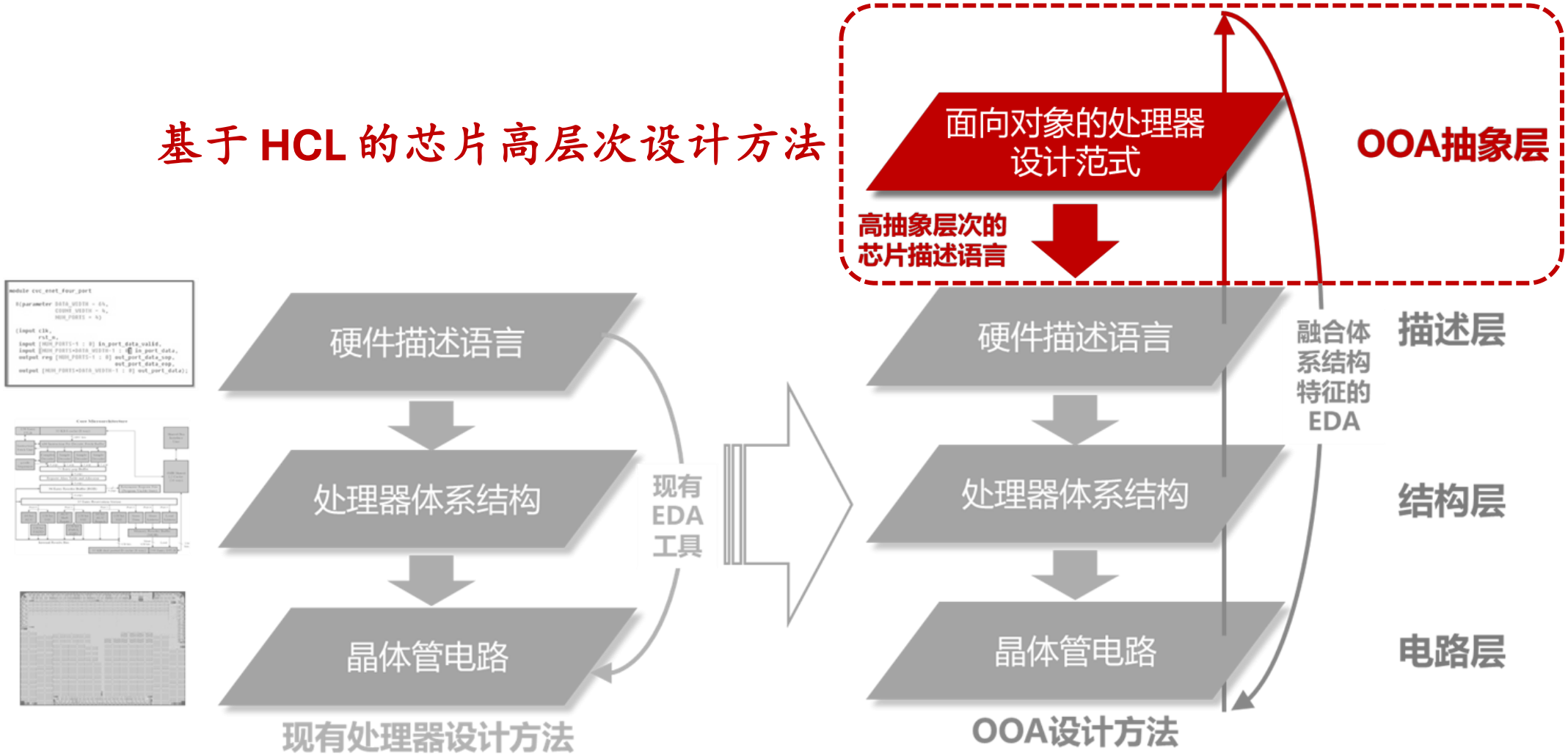
芯片敏捷开发的特征②：硬件构建语言 (HCL)



来源：孙凝晖. 处理器硬件能用面向对象方法敏捷设计吗？CCF DAC 2021.

芯片敏捷开发的特征②：硬件构建语言 (HCL)

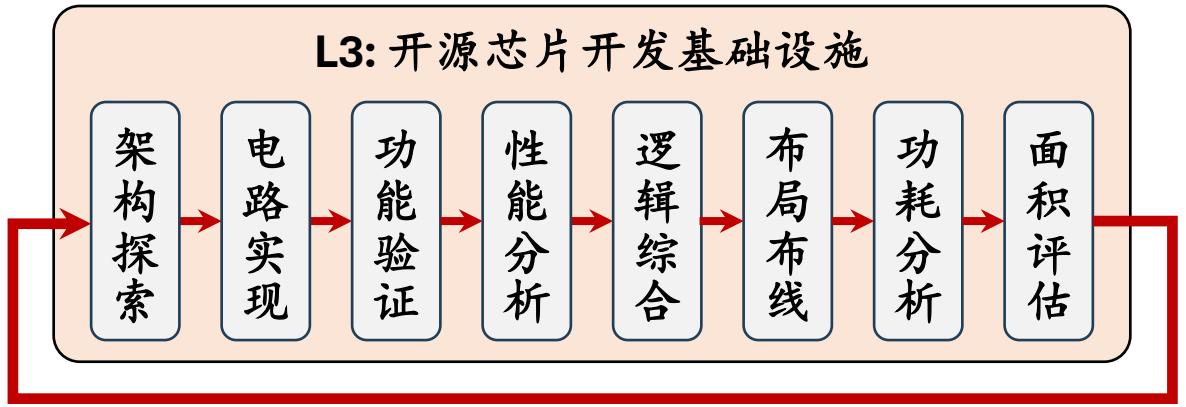
基于 HCL 的芯片高层次设计方法



来源：孙凝晖. 处理器硬件能用面向对象方法敏捷设计吗？CCF DAC 2021.

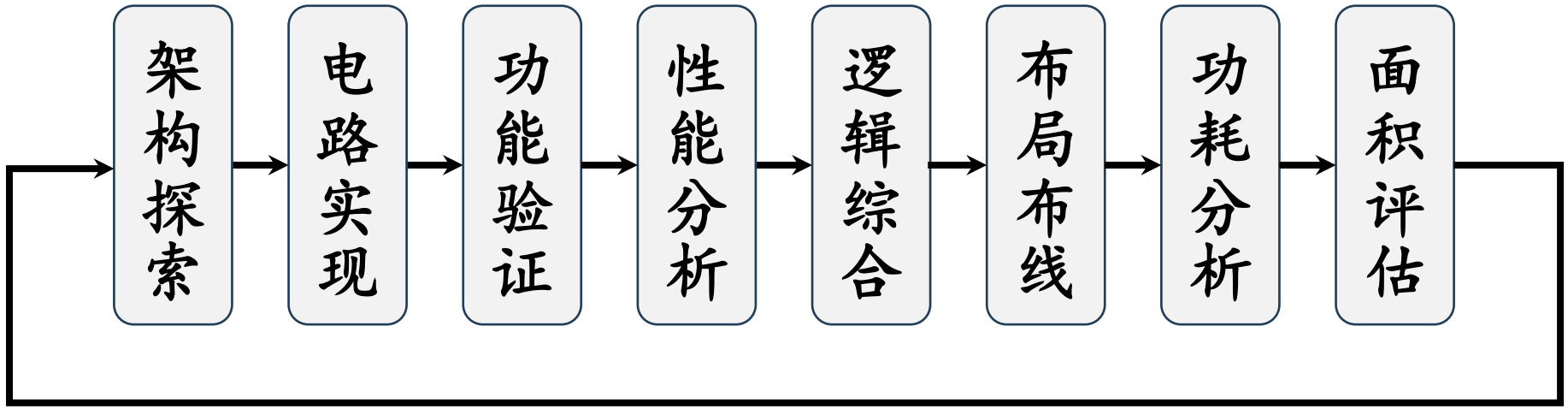
敏捷开发基础设施 (Agile Infra)

① 支持 HCL 和高层次敏捷设计方法



② 支持低成本、短时间、高质量的敏捷迭代流程

敏捷开发基础设施 (Agile Infra)



香山团队：产业需求、学术研究、项目实践...

报告大纲：香山团队的四方面代表性工作

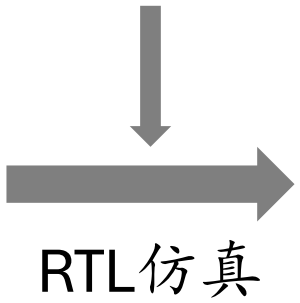
- 问题1：如何确认处理器的行为正确性
- 问题2：如何加速处理器的仿真验证
- 问题3：如何生成处理器的仿真验证用例
- 问题4：早期硬件设计 + 弱软件生态下的快速性能评估

问题1：如何确认处理器的行为正确性

RTL代码

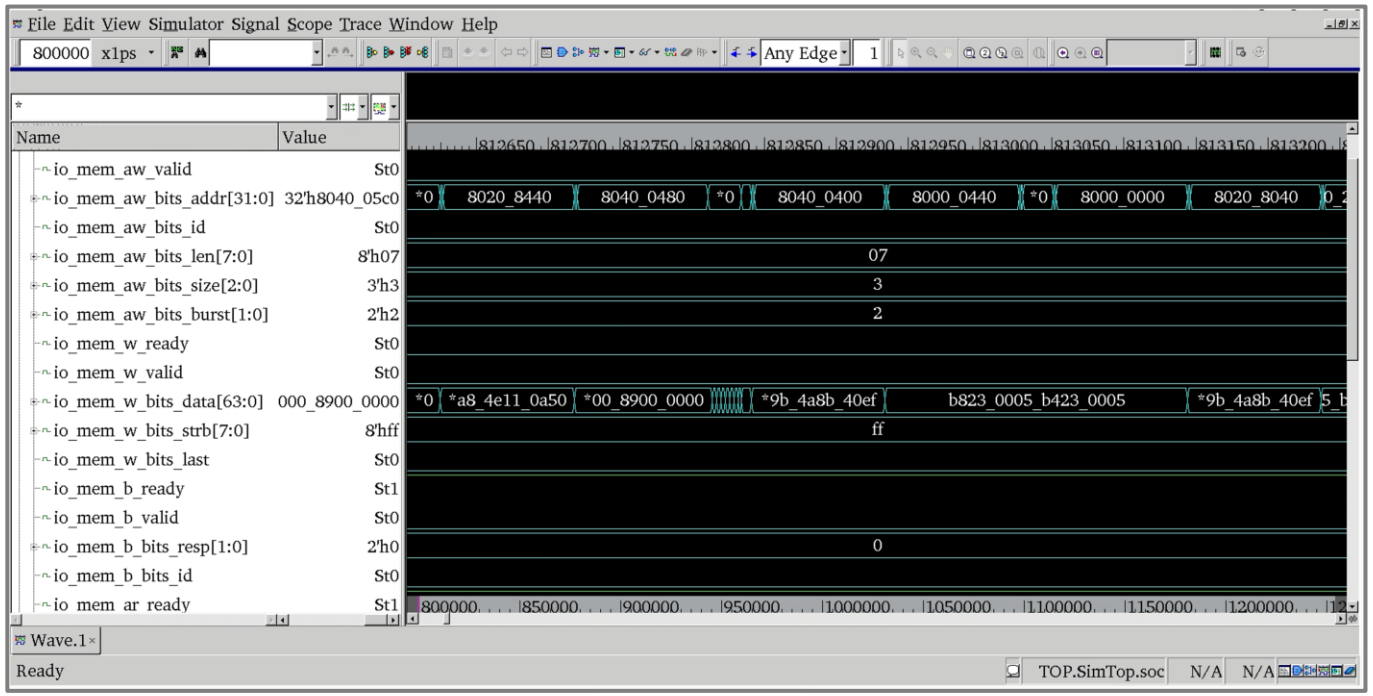
```
class XSTop()(implicit p: Parameters) extends BaseXSSoc() with HasSoCParameter
{
  ResourceBinding {
    val width = ResourceInt(2)
    val model = "freechips,rocketchip-unknown"
    Resource(ResourceAnchors.root, "model").bind(ResourceString(model))
    Resource(ResourceAnchors.root, "compat").bind(ResourceString(model + "-dev"))
    Resource(ResourceAnchors.soc, "compat").bind(ResourceString(model + "-soc"))
    Resource(ResourceAnchors.root, "width").bind(width)
    Resource(ResourceAnchors.soc, "width").bind(width)
    Resource(ResourceAnchors.cpus, "width").bind(ResourceInt(1))
    def bindManagers(xbar: TLNexusNode) = {
      ManagerUnification(xbar.edges.in.head.manager.managers).foreach{ manager =>
        manager.resources.foreach(r => r.bind(manager.toResource))
      }
    }
    bindManagers(misc.l3_xbar.asInstanceOf[TLNexusNode])
    bindManagers(misc.peripheralXbar.asInstanceOf[TLNexusNode])
  }

  println(s"FPGASoC cores: $NumCores banks: $L3NBanks block size: $L3BlockSize bus size: $L3OuterBusWidth")

  val core_with_l2 = tiles.map(coreParams =>
    LazyModule(new XSTile()(p.alterPartial({
      case XSCoreParamsKey => coreParams
    })))
  )

  val l3cacheOpt = soc.L3CacheParamsOpt.map(l3param =>
    LazyModule(new HuanCun()(new Config((_, _, _) => {
      case HCCacheParamsKey => l3param
    })))
  )

  for (i <- 0 until NumCores) {
    core_with_l2(i).clint_int_sink := misc.clint.intnode
    core_with_l2(i).plic_int_sink :*= misc.plic.intnode
    core_with_l2(i).debug_int_sink := misc.debugModule.debug.dmOuter.dmOuter.intnode
    misc.plic.intnode := IntBuffer() := core_with_l2(i).beu_int_source
    misc.peripheral_ports(i) := core_with_l2(i).uncache
    misc.core_to_l3_ports(i) :=* core_with_l2(i).memory_port
  }

  l3cacheOpt.map(_.ctlnode.map(_ := misc.peripheralXbar))
  l3cacheOpt.map(_.intnode.map(int => {
    misc.plic.intnode := IntBuffer() := int
  }))
```

验证用例



RTL仿真

仿真结果 → 对？错？



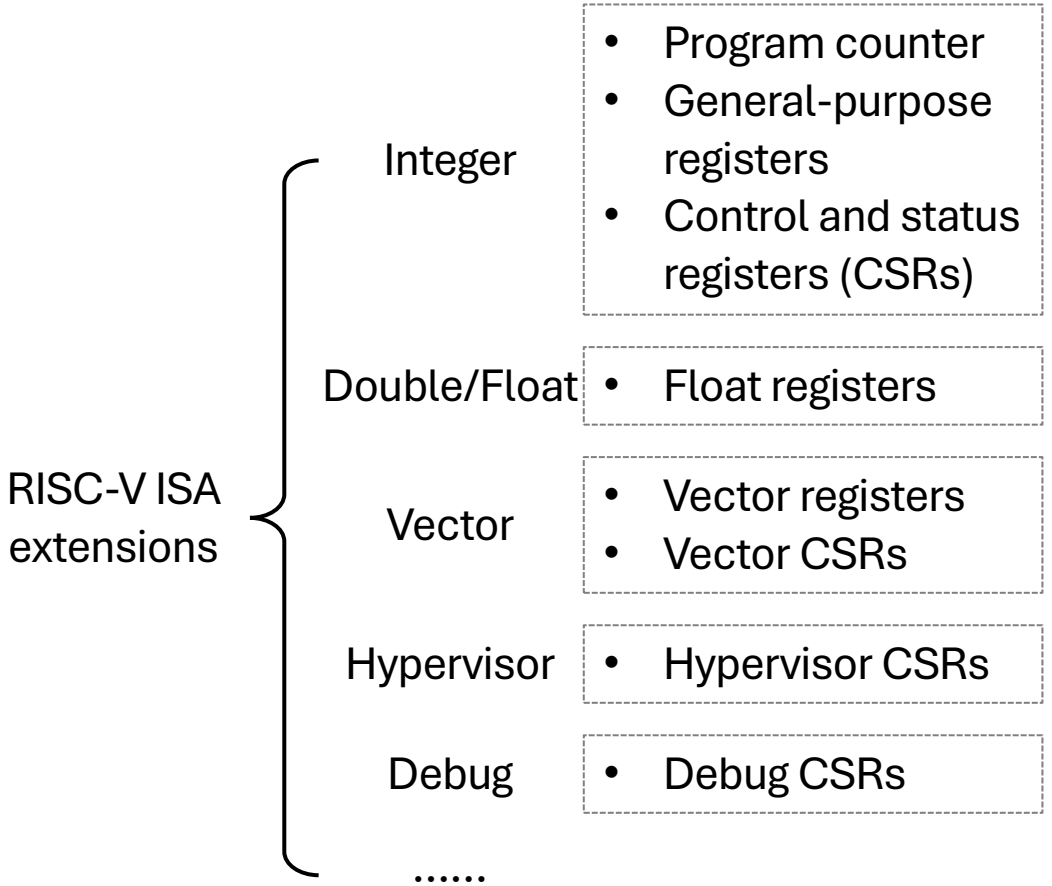
问：待测处理器 (Design Under Test, DUT) 的仿真结果是否正确

背景：指令集复杂度提升

- **RISC-V 指令集复杂度正在迅速膨胀**

- **以 RVA23 Profile 为例**
  - 33 个必选扩展
  - 830 页指令集手册
  - 相比 2019 年手册篇幅翻倍

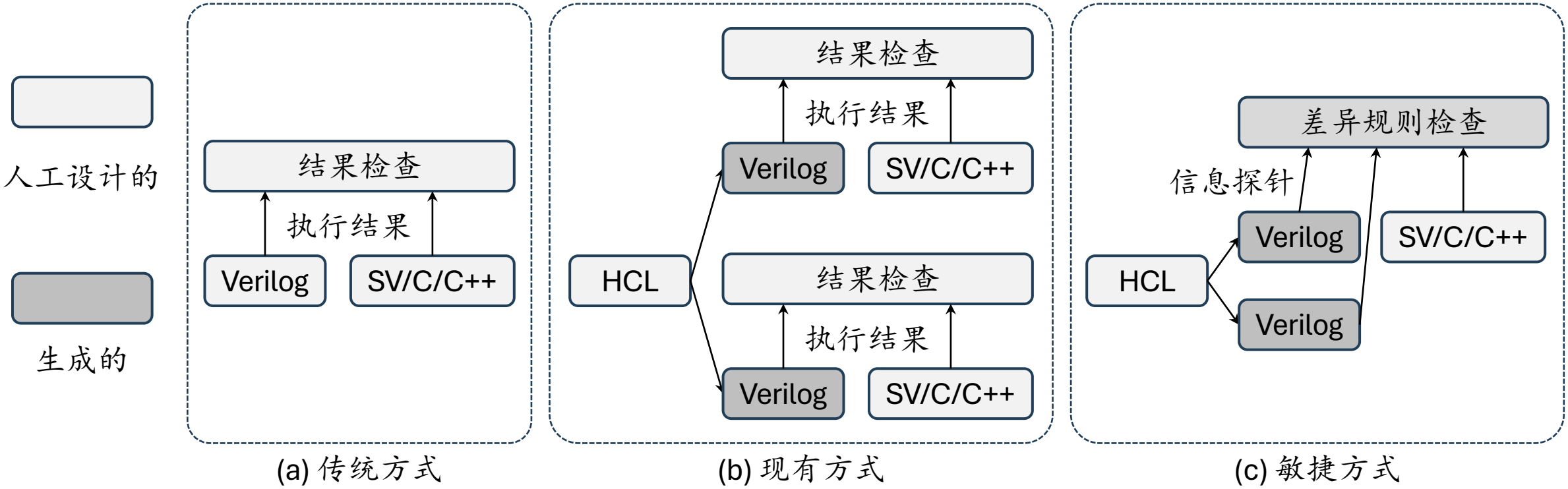
- 功能点、状态量、可选行为、……



问题：如何验证?

DiffTest: A Co-Simulation Framework for RISC-V CPUs

- 设计思路： RISC-V CPU 在**指令粒度**都应符合 RISC-V ISA 要求
  - 使用标准化、参数化的**信息探针**提取验证信息
  - 使用**指令集模拟器（如 Spike）**结合**差异规则**检查验证信息



RISC-V 处理器验证的标准接口

- **挑战**：面向多样化处理器设计灵活可配置；支持非确定性行为
- **关键思路**：以**信息探针**为基础，面向**不同场景**支持自由组合

指令集组合

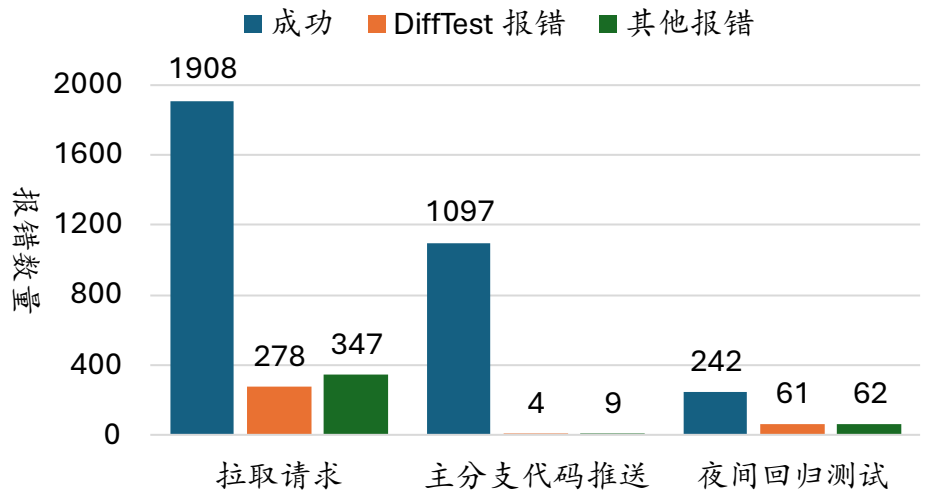
- 基础(Base)
  - ArchEvent
  - TrapEvent
  - InstrCommit
  - ArchIntRegState
  - CSRState
- D/F扩展
  - ArchFpRegState
- V扩展
  - ArchVecRegState
  - VecCSRState
- H扩展
  - HCSRState
- Debug扩展
  - DebugMode

增强验证组合

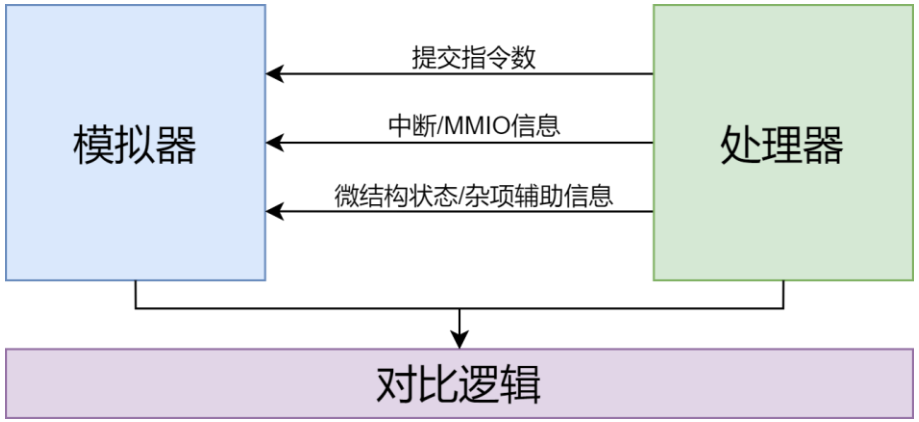
- 多发射
  - IntWriteback
  - FpWriteback
  - VecWriteback
- RF延迟更新
  - ArchIntDelayedUpdate
  - ArchFpDelayedUpdate
- Store检查
  - StoreEvent
- 一致性检查
  - SbufferEvent
  - LoadEvent
  - RefillEvent
- TLB检查
  - L1TLBEvent
  - L2TLBEvent

DiffTest：指令级在线差分验证框架

- **基本流程**
  - 处理器仿真产生指令提交/其他状态更新
  - 模拟器执行相同的指令
  - 比较两者状态，报错或继续
- **支持通用 RISC-V 处理器的仿真验证**
  - 提供香山、NutShell、rocket-chip 等使用案例
  - 提供 Chisel 和 Verilog API
  - 支持 Verilator/VCS/帕拉丁/Xilinx FPGA/GSIM 等 RTL 仿真器
  - 支持 NEMU/Spike 指令集模拟器作为 golden model



2024年3月至2025年2月
通过 CI 发现约 339 个
香山处理器潜在设计缺陷



基本验证框架

```
while (1) {
    icnt = cpu_step();
    nemu_step(icnt);
    r1s = cpu _getregs();
    r2s = nemu_getregs();
    if (r1s != r2s) { abort(); }
}
```

在线对比机制

问题2：如何加速处理器的仿真验证

RTL代码

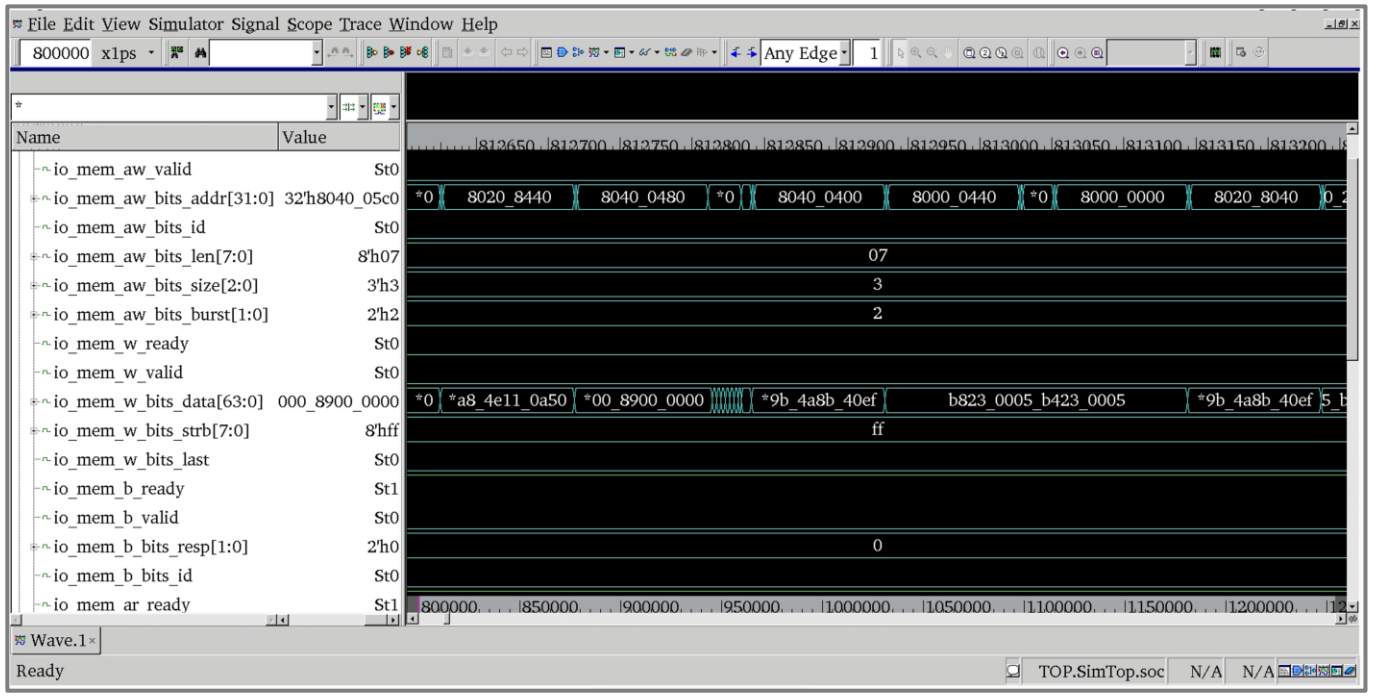
```
class XSTop()(implicit p: Parameters) extends BaseXSSoc() with HasSoCParameter
{
  ResourceBinding {
    val width = ResourceInt(2)
    val model = "freechips,rocketchip-unknown"
    Resource(ResourceAnchors.root, "model").bind(ResourceString(model))
    Resource(ResourceAnchors.root, "compat").bind(ResourceString(model + "-dev"))
    Resource(ResourceAnchors.soc, "compat").bind(ResourceString(model + "-soc"))
    Resource(ResourceAnchors.root, "width").bind(width)
    Resource(ResourceAnchors.soc, "width").bind(width)
    Resource(ResourceAnchors.cpus, "width").bind(ResourceInt(1))
    def bindManagers(xbar: TLNexusNode) = {
      ManagerUnification(xbar.edges.in.head.manager.managers).foreach{ manager =>
        manager.resources.foreach(r => r.bind(manager.toResource))
      }
    }
    bindManagers(misc.l3_xbar.asInstanceOf[TLNexusNode])
    bindManagers(misc.peripheralXbar.asInstanceOf[TLNexusNode])
  }

  println(s"FPGASoC cores: $NumCores banks: $L3NBanks block size: $L3BlockSize bus size: $L3OuterBusWidth")

  val core_with_l2 = tiles.map(coreParams =>
    LazyModule(new XSTile()(p.alterPartial({
      case XSCoreParamsKey => coreParams
    })))
  )

  val l3cacheOpt = soc.L3CacheParamsOpt.map(l3param =>
    LazyModule(new HuanCun()(new Config((_, _, _) => {
      case HCCacheParamsKey => l3param
    })))
  )

  for (i <- 0 until NumCores) {
    core_with_l2(i).clint_int_sink := misc.clint.intnode
    core_with_l2(i).plic_int_sink :*= misc.plic.intnode
    core_with_l2(i).debug_int_sink := misc.debugModule.debug.dmOuter.dmOuter.intnode
    misc.plic.intnode := IntBuffer() := core_with_l2(i).beu_int_source
    misc.peripheral_ports(i) := core_with_l2(i).uncache
    misc.core_to_l3_ports(i) :=* core_with_l2(i).memory_port
  }

  l3cacheOpt.map(_.ctlnode.map(_ := misc.peripheralXbar))
  l3cacheOpt.map(_.intnode.map(int => {
    misc.plic.intnode := IntBuffer() := int
  }))
```

验证用例

协同仿真

快？慢？

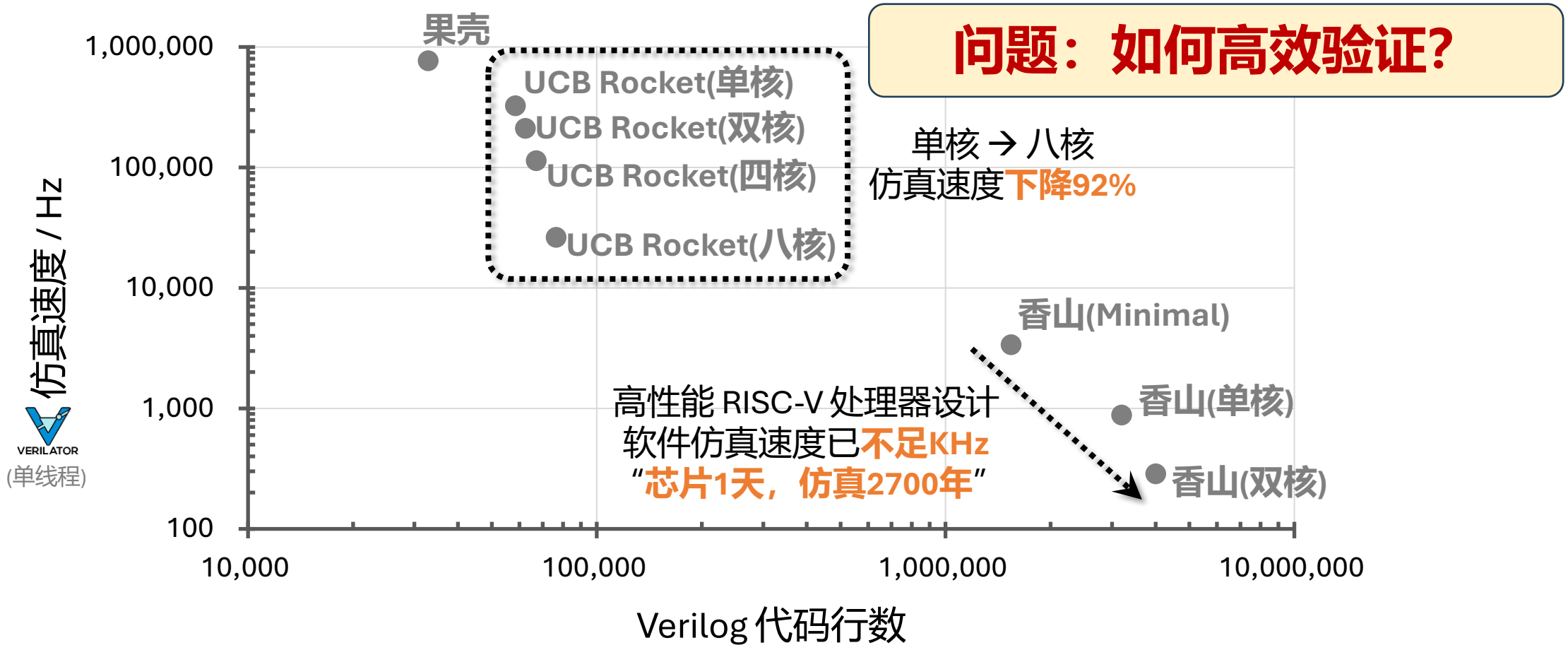
仿真结果



问：能否加速处理器的协同仿真验证？

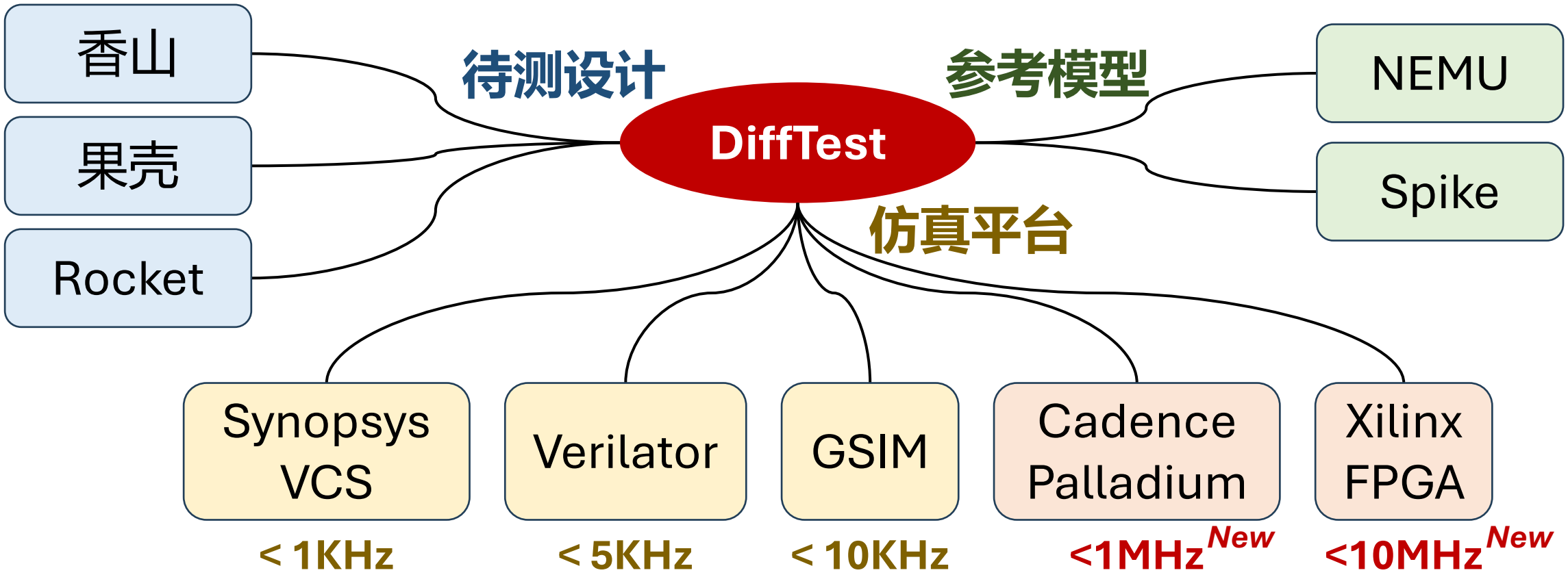
背景：电路仿真速度慢

- 最常使用**软件仿真**，但当**处理器规模扩大**，其**仿真速度大幅下降**



基于Emulator/FPGA的处理器验证加速

- **香山团队长期维护开源验证框架DiffTest，现已支持硬件仿真加速**
  - https://docs.xiangshan.cc/zh-cn/latest/tools/difftest/



基于Emulator/FPGA的处理器验证加速

- **动机**：**硬件仿真平台**数量级地加速电路仿真，在此基础上优化DUT-REF整体速度
- **特点**：将 REF 部署于 Host 环境，以**软硬件(RTL-Host)通信**为核心
  - **硬件**(RTL侧)：利用仿真加速器、FPGA等加速 **DUT 电路仿真**
  - **软件**(Host侧)：仍使用x86、ARM宿主机运行 **REF 软件逻辑**
  - **通信**：利用PCIe、以太网、InfiniBand(IB)等**连接手段**在软硬件间传递数据

- Emulator、FPGA等
- 速度 1~100 MHz

- x86/ARM CPU
- 速度 >> 100 MHz

RISC-V 处理器验证的标准接口

- 包含 **30+** 不同种类、大小、结构的**验证接口**
- **每个时钟周期**平均需**进行13.5次通信**，**传输1251字节**

通信开销如何?

指令集组合

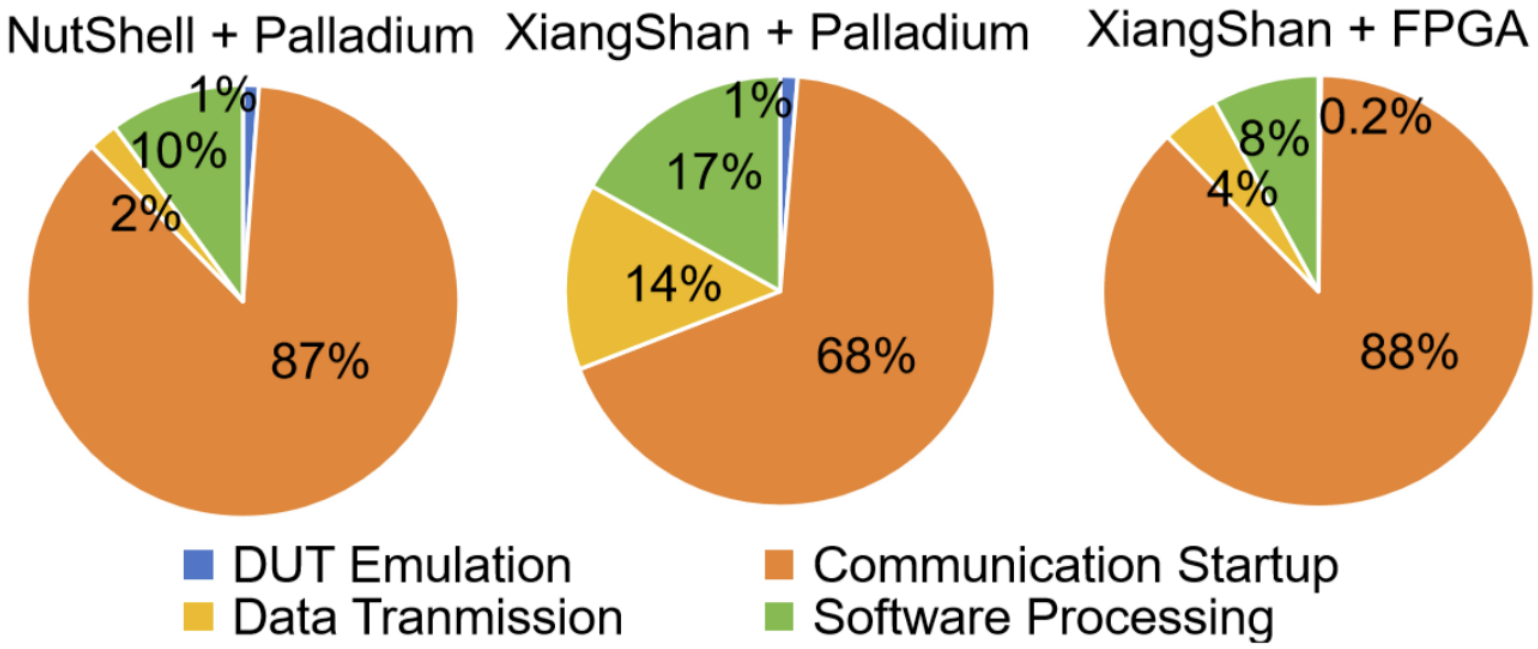
基础(Base)	• ArchEvent • TrapEvent • InstrCommit • ArchIntRegState • CSRState
D/F扩展	• ArchFpRegState
V扩展	• ArchVecRegState • VecCSRState
H扩展	• HCSRState
Debug扩展	• DebugMode

增强验证组合

多发射	• IntWriteback • FpWriteback • VecWriteback
RF延迟更新	• ArchIntDelayedUpdate • ArchFpDelayedUpdate
Store检查	• StoreEvent
一致性检查	• SbufferEvent • LoadEvent • RefillEvent
TLB检查	• L1TLBEvent • L2TLBEvent

RTL-Host架构的通信开销显著

- **动机**：**硬件仿真平台**数量级地加速电路仿真，在此基础上优化DUT-REF整体速度
- **特点**：将 REF 部署于 Host 环境，以**软硬件(RTL-Host)通信**为核心



图：仿真时间开销占比

问题：通信成为速度瓶颈

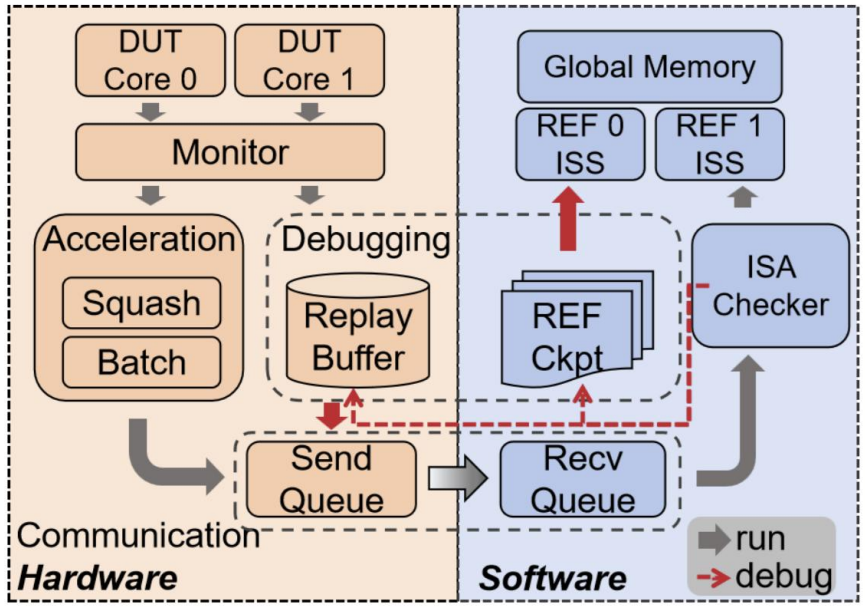
- **建立通信：68%~88%**
- **数据传输：2%~14%**
- **数据处理：8%~17%**
- RTL仿真：<1%

语义感知的协同仿真验证加速 DiffTest-H

- **特点：针对不同的验证数据包，相应进行通信开销优化**
  - Batch：降低通信频次
  - Squash：减少通信数据量
  - Replay：保持调试粒度

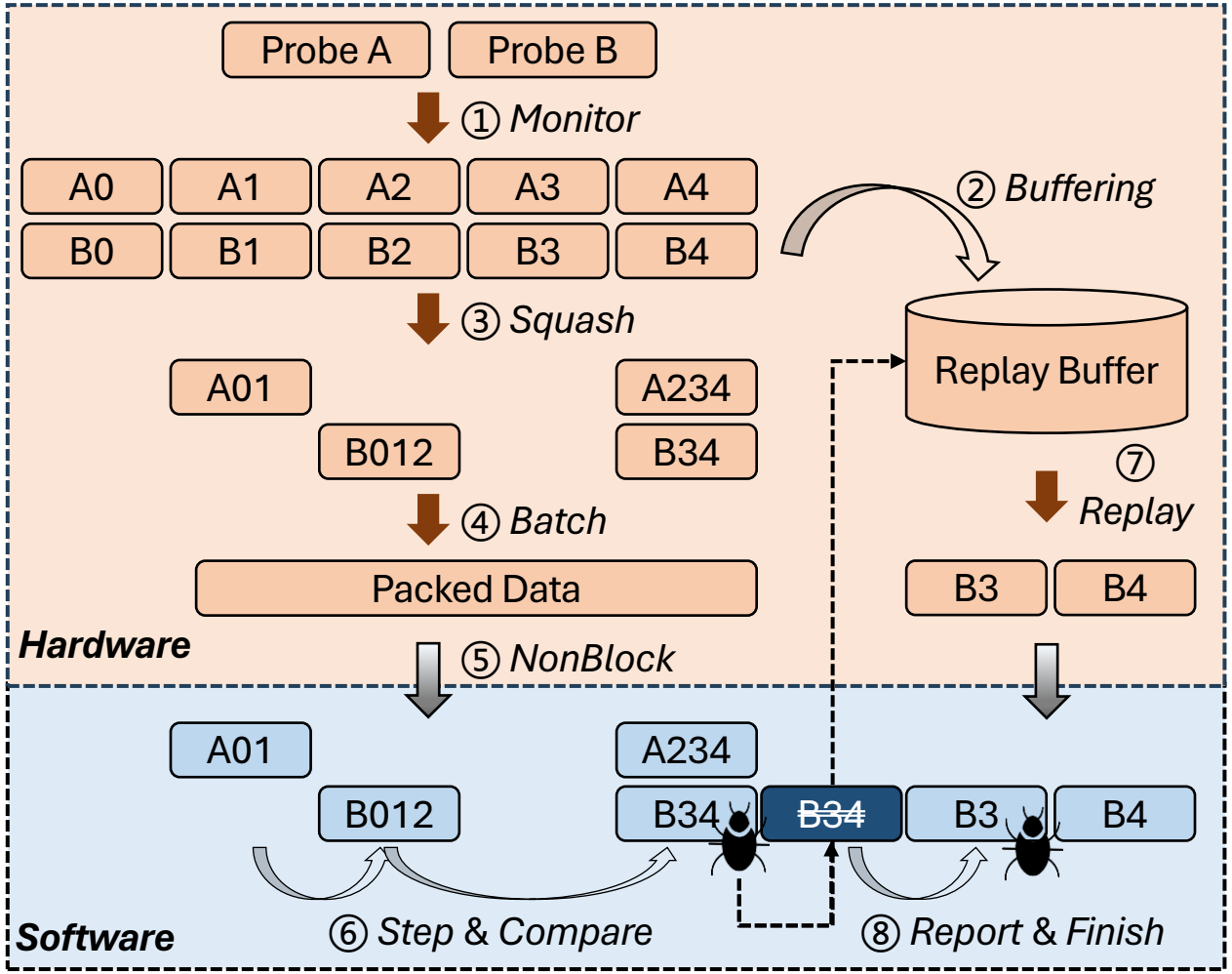
Category	Types	Representative Examples
Control Flow	5	Exceptions and interrupts, Instruction commits, Traps, ...
Register Updates	9	CSRs, General-purpose registers, Floating-point registers, ...
Memory Access	3	Load/store operations, Atomic memory operations, ...
Memory Hierarchy	6	Cache refill operations, L1/L2 TLB operations, ...
RISC-V Extensions	9	Vector/Hypervisor CSRs, Vector registers, ...

表：不同的验证数据包类型



图：DiffTest-H工作流程

DiffTest-H的工作流程



DiffTest-H加速仿真验证达57~109倍

代码已开源
论文发表于MICRO'25
敬请关注香山公众号

配置	NutShell (帕拉丁)	香山 (帕拉丁)	香山 (FPGA)
无优化	14KHz	4KHz	0.1MHz
+Batch	102 KHz (7×)	24 KHz (6×)	1.3 MHz (13×)
+NonBlock	398 KHz (28×)	71 KHz (12×)	2.2 MHz (22×)
+Squash	1080 KHz (77×)	478 KHz (109×)	5.7 MHz (57×)

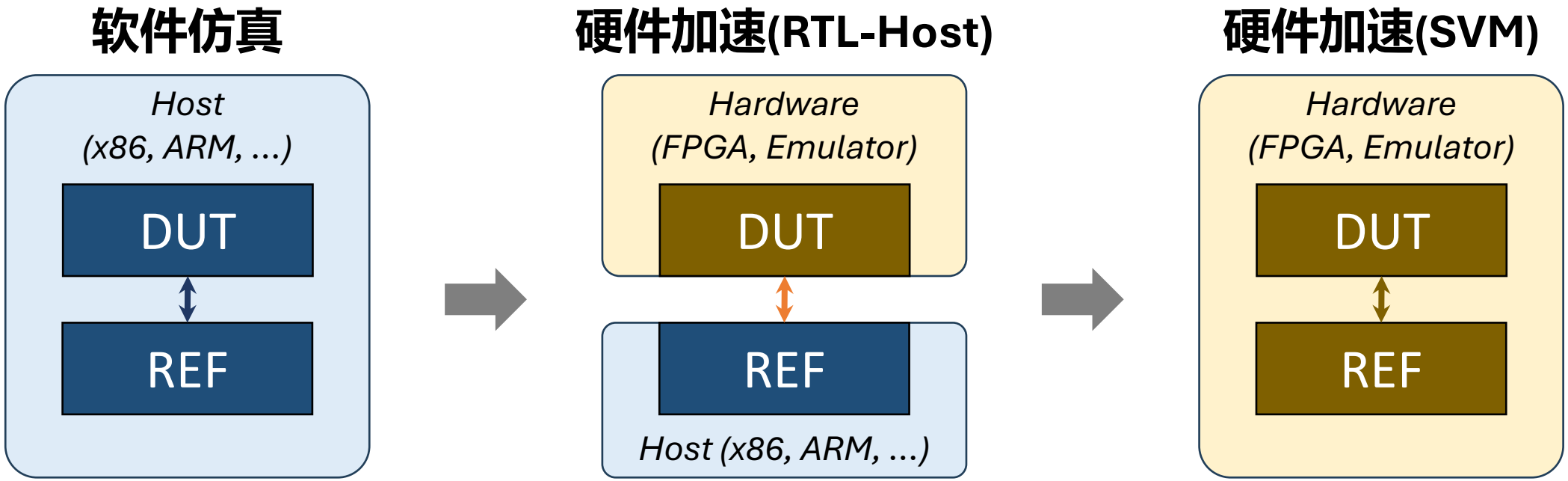
Work	Platform	Verification States/Bytes †	Communication Overhead	Area Overhead	DUT-only Speed	Co-sim Speed
IBI-check [8]	IBM AWAN [13]	2 / 7	20 %	20 %	100 KHz	80 KHz
SBS-check [19]	Gem5 [5] (for estimation)	2 / 7	2 %‡	22%‡	100 KHz‡	98 KHz‡
DiffTest-H	Cadence Palladium [7]	32 / 1200	0.6 %	26%	725 KHz	**722 KHz**
Fromajo [54, 55]	FireSim [22]	7 / 24	99 %	Unknown	100 MHz	1 MHz
DiffTest-H	Xilinx VU19P	32 / 1200	95.6 %	24 %	50 MHz	**5.7 MHz**

表：DiffTest-H工作与现有工作的对比

有没有可能更进一步?

- FPGA理论速度上限可到100MHz
- 当前实际仿真加速仅有10MHz
- 还有可能进一步提升吗?

可综合验证方法 (SVM)



Native Slow

Fast yet Link-Constrained

Native Fast

本工作：可综合验证方法 (SVM)

- **另一种不同思路**：**可综合硬件代码**实现所有的协同仿真验证逻辑
  - 完全实现为硬件逻辑后，原有验证数据通信转为片上连线，无开销

- Emulator、FPGA等
- 速度 1~100 MHz

- Emulator、FPGA等

- **实现 SVM 面临的挑战**
  - **编码层**：指令语义复杂，如何确保**硬件 REF 的正确性**?
  - **架构层**：硬件实现约束多，如何确保**硬件 REF 的执行效率**?
  - **运行时**：硬件调试困难，如何提升 **SVM 框架的可调试性**?

挑战1：REF 的电路代码实现

- 现有 REF 通常为软件指令集模拟器，**设计简洁、规整**，保障可靠性
  - 使用 C/C++ 提供的高层次数据结构进行描述
  - 使用更方便的编程方式，如封装、抽象等，降低出错风险

类别	案例	软件描述方式	
指令	add	指令模板 insns	`WRITE_RD(sext_xlen(RS1 + RS2))`
CSR	访问权限	CSR 基类 csr_t	`csr_t::verify_permissions`
	读		`cst_r::read`
	写		`csr_t::write`
常量	ADD指令掩码	常量列表 riscv-opcodes	`#define MATCH_ADD 0x33`
运算	提取/写入字段	标准接口 decode.h	`get_field/set_field`

挑战1：REF 的电路代码实现

- 现有 REF 通常为软件指令集模拟器，**设计简洁、规整**，保障可靠性
  - 使用 C/C++ 提供的高层次数据结构进行描述
  - 使用更方便的编程方式，如封装、抽象等，降低出错风险

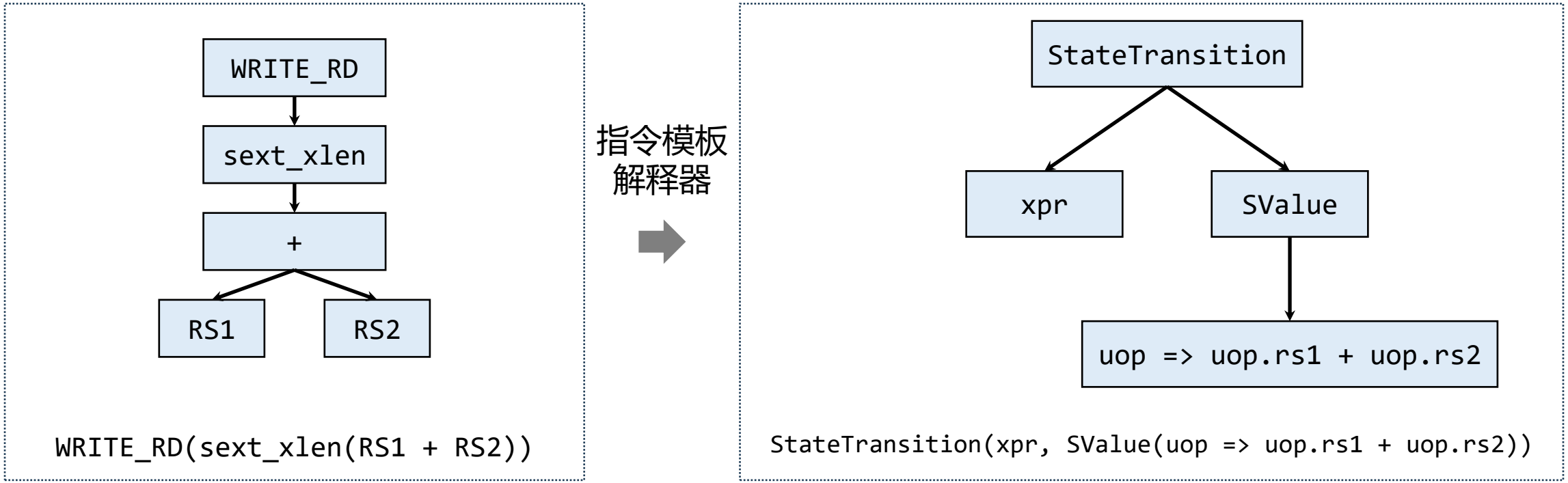
- **问题**：缺少支持**可综合 REF 的高效硬件描述方式**

类别	案例	软件描述方式		硬件描述方式
指令	add	指令模板 insns	`WRITE_RD(sext_xlen(RS1 + RS2))`	???
CSR	访问权限	CSR 基类 csr_t	`csr_t::verify_permissions`	???
	读		`cst_r::read`	
	写		`csr_t::write`	
常量	ADD指令掩码	常量列表 riscv-opcodes	`#define MATCH_ADD 0x33`	???
运算	提取/写入字段	标准接口 decode.h	`get_field/set_field`	???

关键技术1：语义代码迁移技术

- **思路**：**自动化转换**软件 REF 中的关键指令集语义信息

- **例**：构造**指令操作树**完成**指令功能**的迁移



(a) Spike

(b) 本工作

关键技术1：语义代码迁移技术

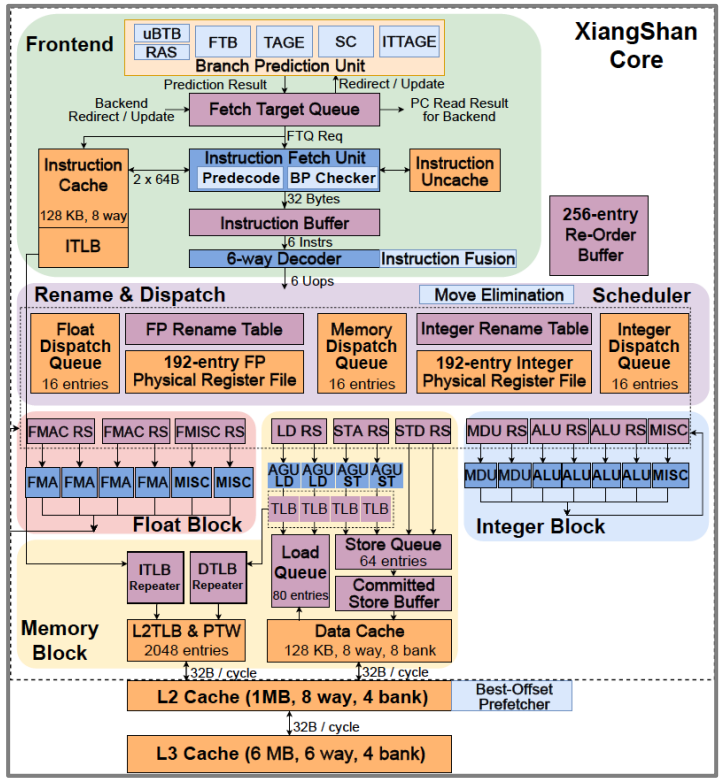
- 支持**指令功能、控制和状态寄存器(CSR)、常量**等语义信息自动迁移
  - 部分提供了和 Spike 类似的 RISC-V REF 编程接口

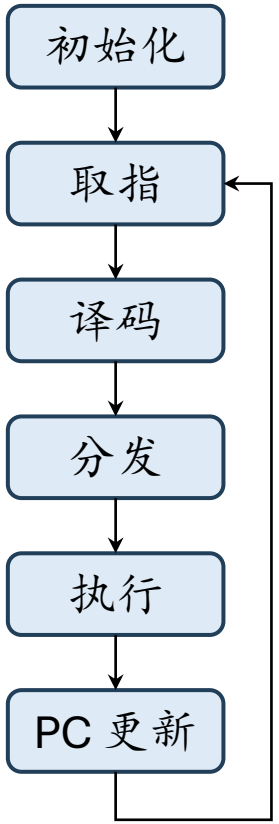
类别	案例	软件描述方式		硬件描述方式
指令	add	指令模板 insns	`WRITE_RD(sext_xlen(RS1 + RS2))`	**指令模板解释器**
CSR	访问权限	CSR基类 csr_t	`csr_t::verify_permissions`	**CSR基类 CSRDef**
	读		`cst_r::read`	
	写		`csr_t::write`	
常量	ADD指令掩码	常量列表 riscv-opcodes	`#define MATCH_ADD 0x33`	**字符串插值**
运算	提取/写入字段	标准接口 decode.h	`get_field/set_field`	**标准接口**

挑战2：硬件 REF 的执行效率

- 协同仿真验证的**指令吞吐**取决于 DUT 和 REF 两者中较小的那个
  - DUT 设计复杂，微结构细节丰富，指令执行效率高
  - **REF 须设计简洁**以保障功能可靠，如何提升其**执行效率**？

DUT 微结构设计复杂





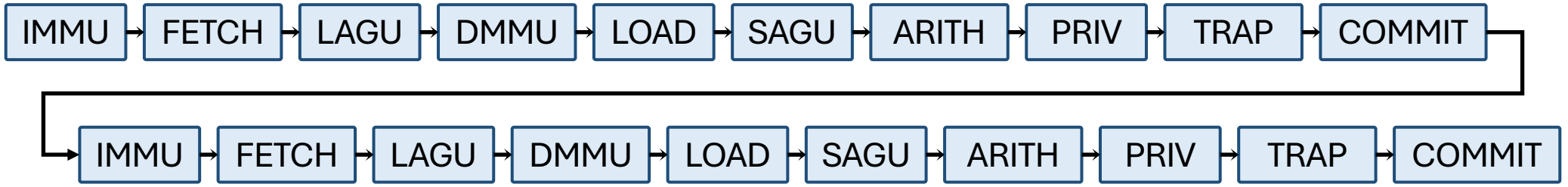
REF 微结构设计简单

关键技术2：硬件参考模型(SRef)设计

- **工作流程**：当 DUT 提交 *N* 条指令，SRef 执行 *N* 条指令，对比结果

关键技术2：硬件参考模型(SRef)设计

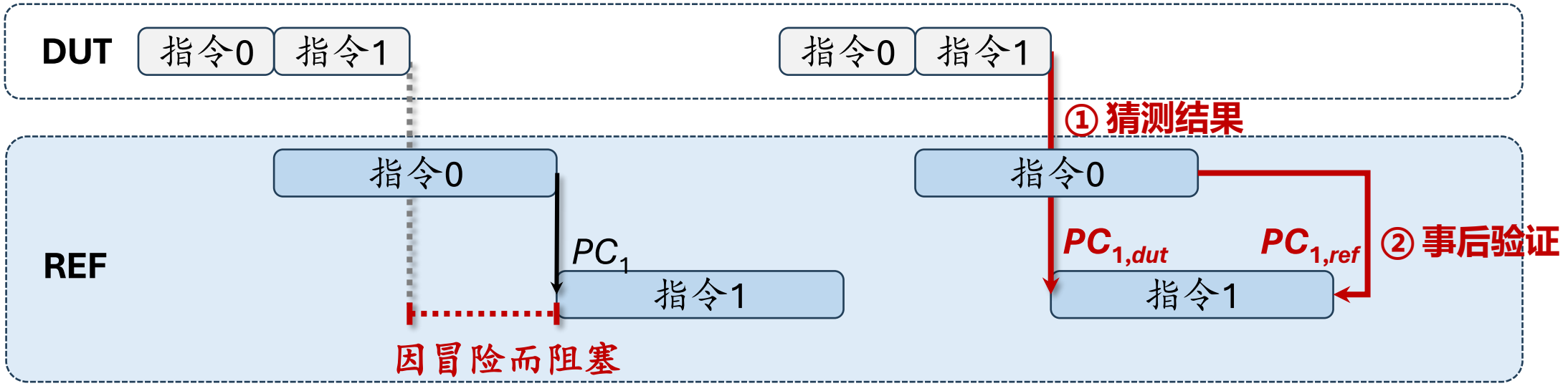
- **工作流程**：当 DUT 提交 *N* 条指令，SRef 执行 *N* 条指令，对比结果
- **简单架构设计：全流水、无阻塞**
  - **预期**：在 *N* * pipeline_length 时钟周期内， SRef 一定能完成指令执行
  - 如果预期成立，那么 SRef 能和任意性能(IPC)的处理器进行协同仿真



例：DUT 提交 2 条指令后，SRef 依次通过各流水级完成 2 条指令执行

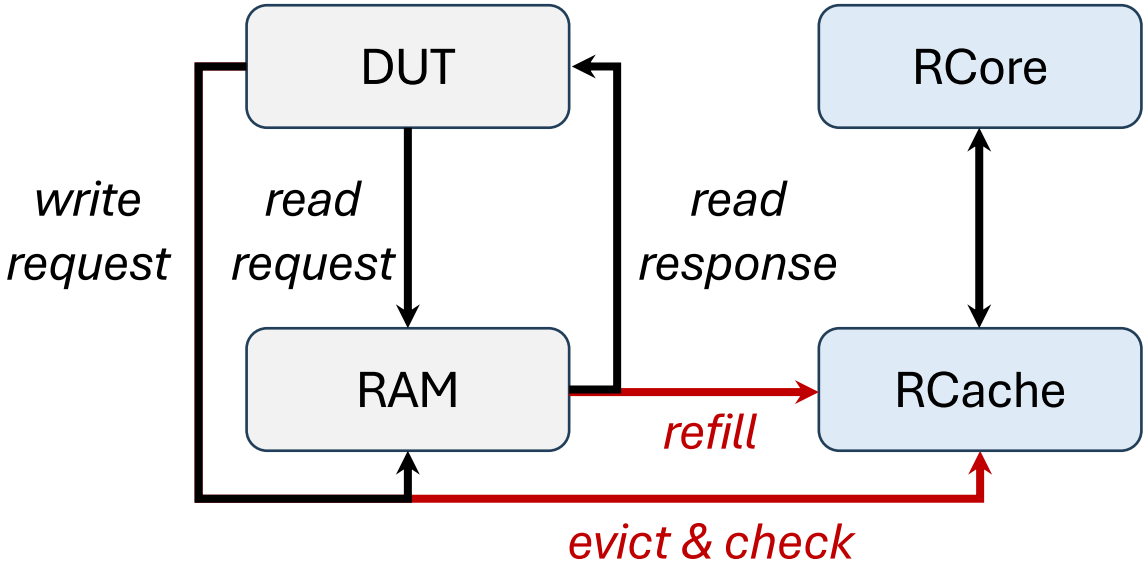
关键技术2：参考模型的主流水线(RCore)设计

- **问题1**：指令间存在**控制和数据冒险(Hazard)**，造成流水线阻塞
- **解决思路**：利用 DUT 信息的**推测执行**机制
  - 利用 DUT 的执行结果，通过**猜测结果 + 事后验证**，消除控制和数据依赖



关键技术2：参考模型的缓存系统(RCache)设计

- **问题2**：内存访问存在**结构冒险**，不足以支撑 REF **访存带宽**需求
- **解决思路**：利用 DUT 访存结果，尽可能避免重复内存访问
  - 等同于：RCache 仅保存和 DUT RAM 不相同的数据块（REF/DUT RAM 差集）
  - DUT 读：内存返回的数据一定是正确的，根据地址回填 RCache
  - DUT 写：确认 RCache 中存在相同数据块并标记为驱逐，数据不同则报错



挑战3：可综合验证框架的可调试性

- 软件验证框架支持**断言、错误日志、计数器**等基本调试方式
  - 但硬件环境缺乏对此的原生支持

调试需求	典型软件机制	现有硬件支持
断言	assert	**有硬件原语，不可综合**
错误日志	display, printf	
计数器	counter	**无硬件原语**
异常处理	signal, sig_handler	
	coredump	

关键技术3：可综合验证调试技术

- 软件验证框架支持**断言、错误日志、计数器**等基本调试方式
  - 但硬件环境缺乏对此的原生支持
- **解决思路**：**REF 可被转换为独立执行的通用 CPU 用于调试**
  - 基于硬件化断言、结果检查等条件进行报错，并切换 REF 状态
  - 验证模式：用于 DUT 执行结果的正确性检查
  - 调试模式：用于出错后的错误现场调试

调试需求	典型软件机制	本工作
断言	assert	**硬件化断言提取器**
错误日志	display, printf	**调试模式 + 串口**
计数器	counter	**硬件化计数器**
异常处理	signal, sig_handler	**状态控制和检查器**
	coredump	**状态记录器**

SVM 工具链

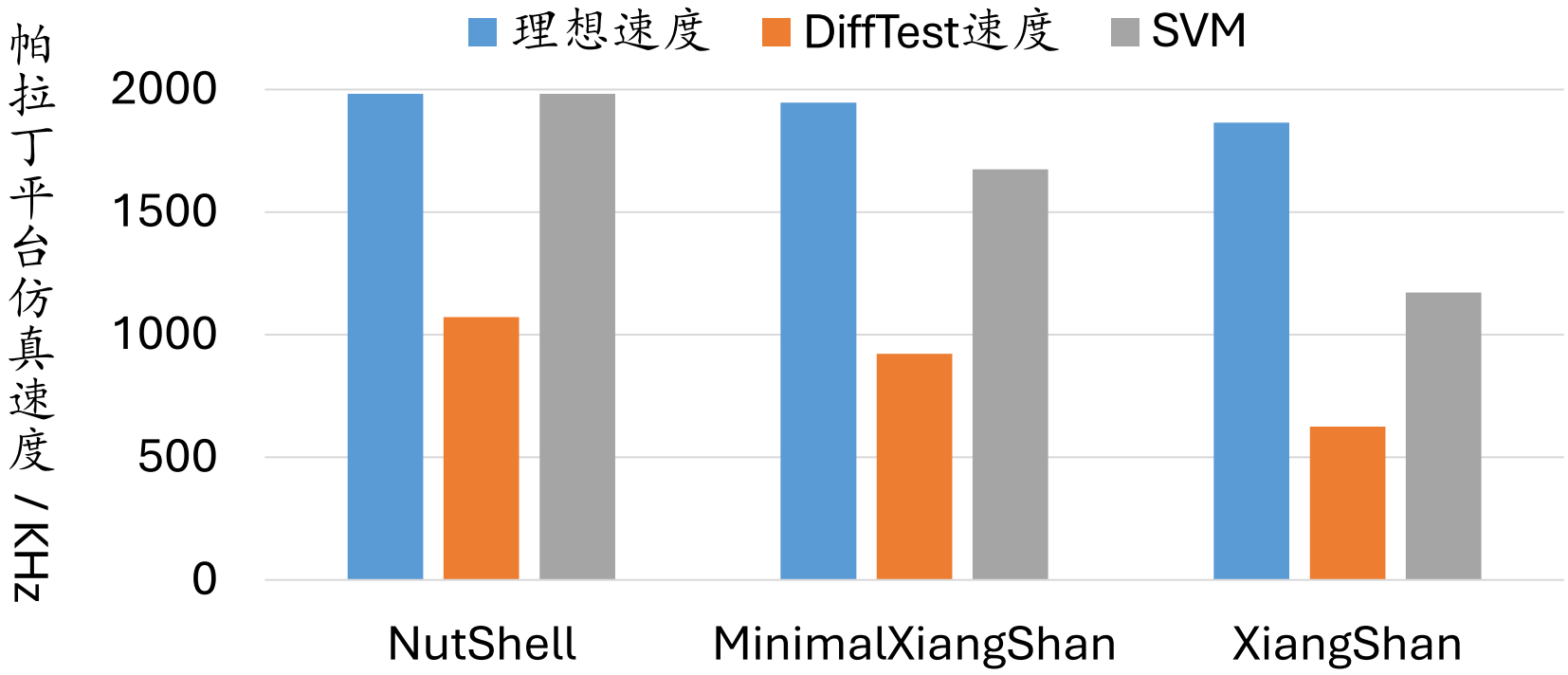
- 本工作实现了一套面向RISC-V处理器的**可综合验证工具链**
  - 即将整理开源

- **兼容 DiffTest[1] 的用户侧接口**，接管其**仿真逻辑**
  - 支持不同配置的处理器，如香山、果壳等
  - 支持 RISC-V 指令集的 I、M、A、C、B 扩展指令
  - 支持 RISC-V 指令集的 M、S、U 特权模式和对应特权操作
  - 支持自动解析 Spike 指令集模拟器源码中的语义信息

[1] https://github.com/OpenXiangShan/difftest

实验评估：接近硬件平台理想仿真速度

- 将 SVM 用于不同配置的**香山、果壳处理器**协同仿真验证
  - **FPGA**[1]：果壳，使用 512KB RCache，速度达 **60MHz**，比 DiffTest **快6倍**
  - **帕拉丁**：果壳最高达 **1.9MHz**，与理想速度一致，比 DiffTest **快 90%**



[1] 由于容量限制，当前Xilinx zu19eg FPGA仅实现果壳处理器的SVM验证

问题3：如何生成处理器的仿真验证用例

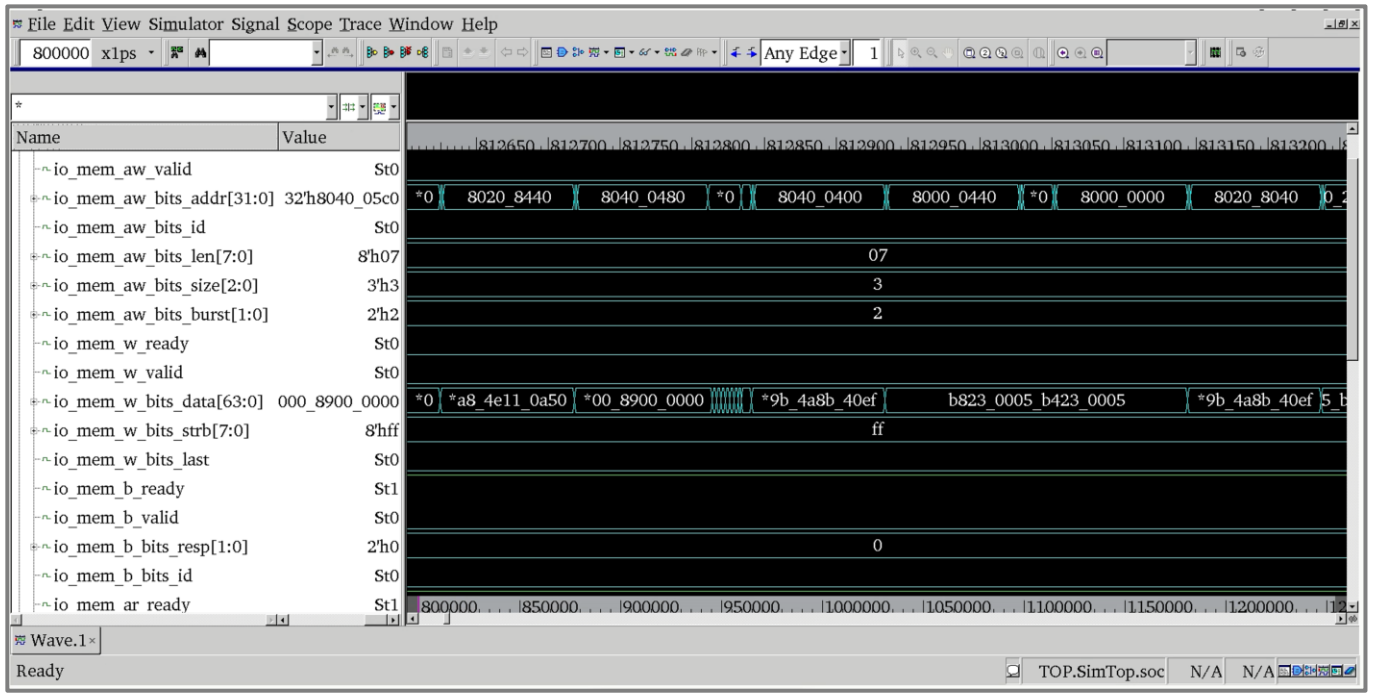
RTL代码

```
class XSTop()(implicit p: Parameters) extends BaseXSSoc() with HasSoCParameter
{
  ResourceBinding {
    val width = ResourceInt(2)
    val model = "freechips,rocketchip-unknown"
    Resource(ResourceAnchors.root, "model").bind(ResourceString(model))
    Resource(ResourceAnchors.root, "compat").bind(ResourceString(model + "-dev"))
    Resource(ResourceAnchors.soc, "compat").bind(ResourceString(model + "-soc"))
    Resource(ResourceAnchors.root, "width").bind(width)
    Resource(ResourceAnchors.soc, "width").bind(width)
    Resource(ResourceAnchors.cpus, "width").bind(ResourceInt(1))
    def bindManagers(xbar: TLNexusNode) = {
      ManagerUnification(xbar.edges.in.head.manager.managers).foreach{ manager =>
        manager.resources.foreach(r => r.bind(manager.toResource))
      }
    }
    bindManagers(misc.l3_xbar.asInstanceOf[TLNexusNode])
    bindManagers(misc.peripheralXbar.asInstanceOf[TLNexusNode])
  }

  println(s"FPGASoC cores: $NumCores banks: $L3NBanks block size: $L3BlockSize bus size: $L3OuterBusWidth")

  val core_with_l2 = tiles.map(coreParams =>
    LazyModule(new XSTile()(p.alterPartial({
      case XSCoreParamsKey => coreParams
    })))
  )

  val l3cacheOpt = soc.L3CacheParamsOpt.map(l3param =>
    LazyModule(new HuanCun()(new Config((_, _, _) => {
      case HCCacheParamsKey => l3param
    })))
  )

  for (i <- 0 until NumCores) {
    core_with_l2(i).clint_int_sink := misc.clint.intnode
    core_with_l2(i).plic_int_sink :*= misc.plic.intnode
    core_with_l2(i).debug_int_sink := misc.debugModule.debug.dmOuter.dmOuter.intnode
    misc.plic.intnode := IntBuffer() := core_with_l2(i).beu_int_source
    misc.peripheral_ports(i) := core_with_l2(i).uncache
    misc.core_to_l3_ports(i) :=* core_with_l2(i).memory_port
  }

  l3cacheOpt.map(_.ctlnode.map(_ := misc.peripheralXbar))
  l3cacheOpt.map(_.intnode.map(int => {
    misc.plic.intnode := IntBuffer() := int
  }))
```

验证用例好？坏？

协同仿真快？慢？

仿真结果



问：能否尽可能高效、完整地覆盖待验证的处理器功能?

处理器验证用例的典型类型

✓ **约束随机**

指令序列生成器

- riscv-torture
- riscv-dv
- force-riscv

✓ **专家经验**

定向兼容性测试

- riscv-tests
- riscv-arch-test
- imperas-riscv-tests

✓ **目标场景**

现实程序负载

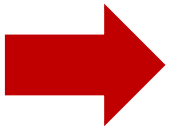
- CoreMark
- UnixBench
- SPEC CPU® 2006

测试生成技术存在依赖人工经验的问题

✓ **约束随机**

指令序列生成器

- riscv-torture
- riscv-dv
- force-riscv

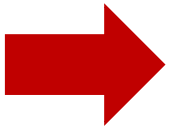


依赖人工调整约束条件

✓ **专家经验**

定向兼容性测试

- riscv-tests
- riscv-arch-test
- imperas-riscv-tests

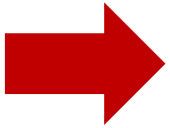


依赖人工设计用例内容

✓ **目标场景**

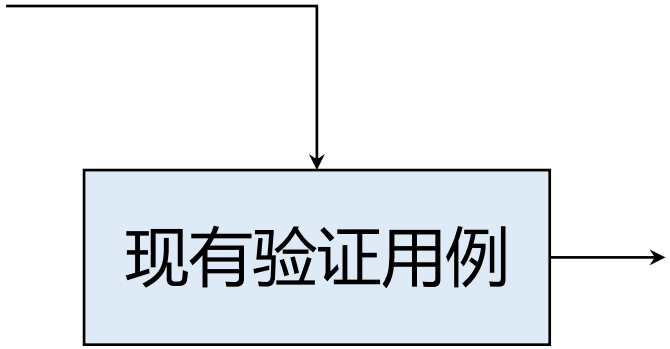
现实程序负载

- CoreMark
- UnixBench
- SPEC CPU® 2006



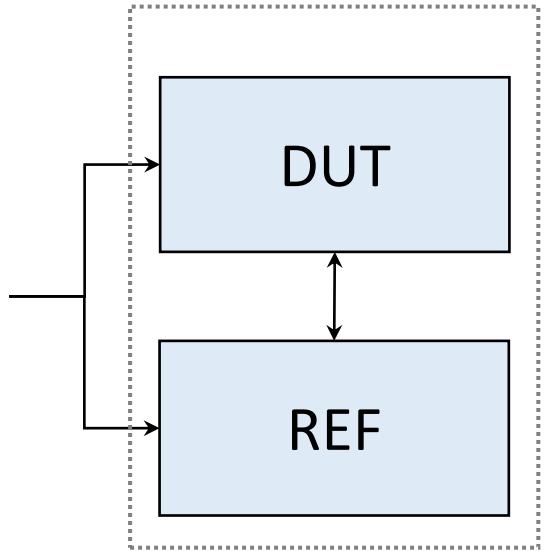
依赖人工选择负载来源

问：现有大量用例，如何进一步自动提升验证质量



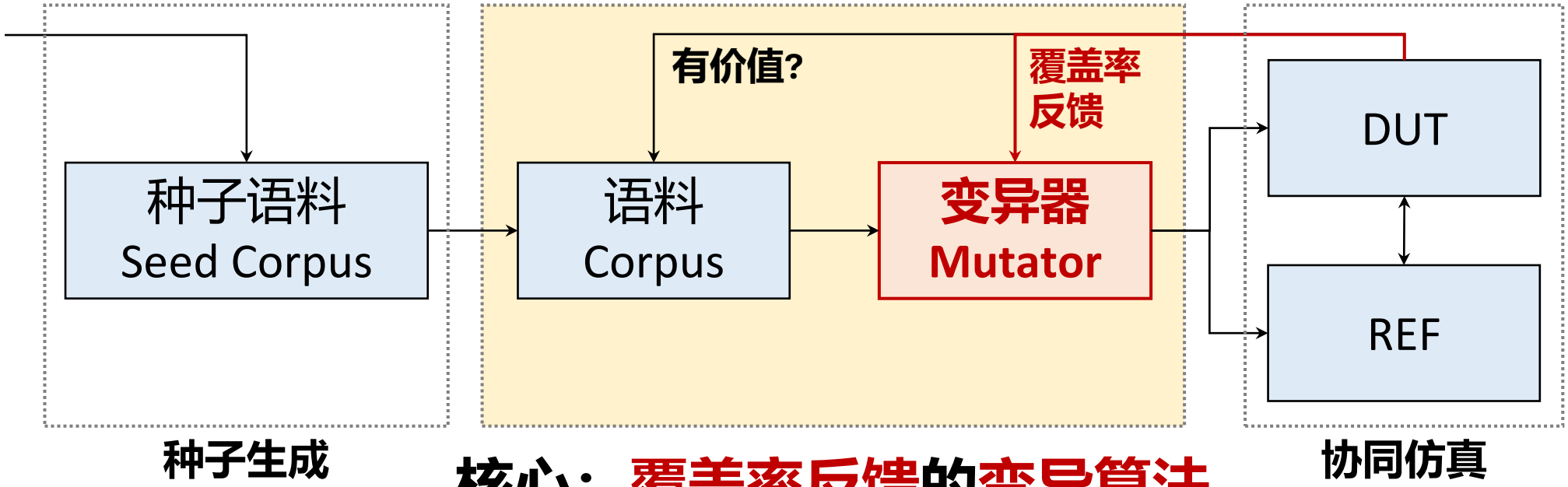
长期积累的人类经验知识

如何利用并改进?



协同仿真

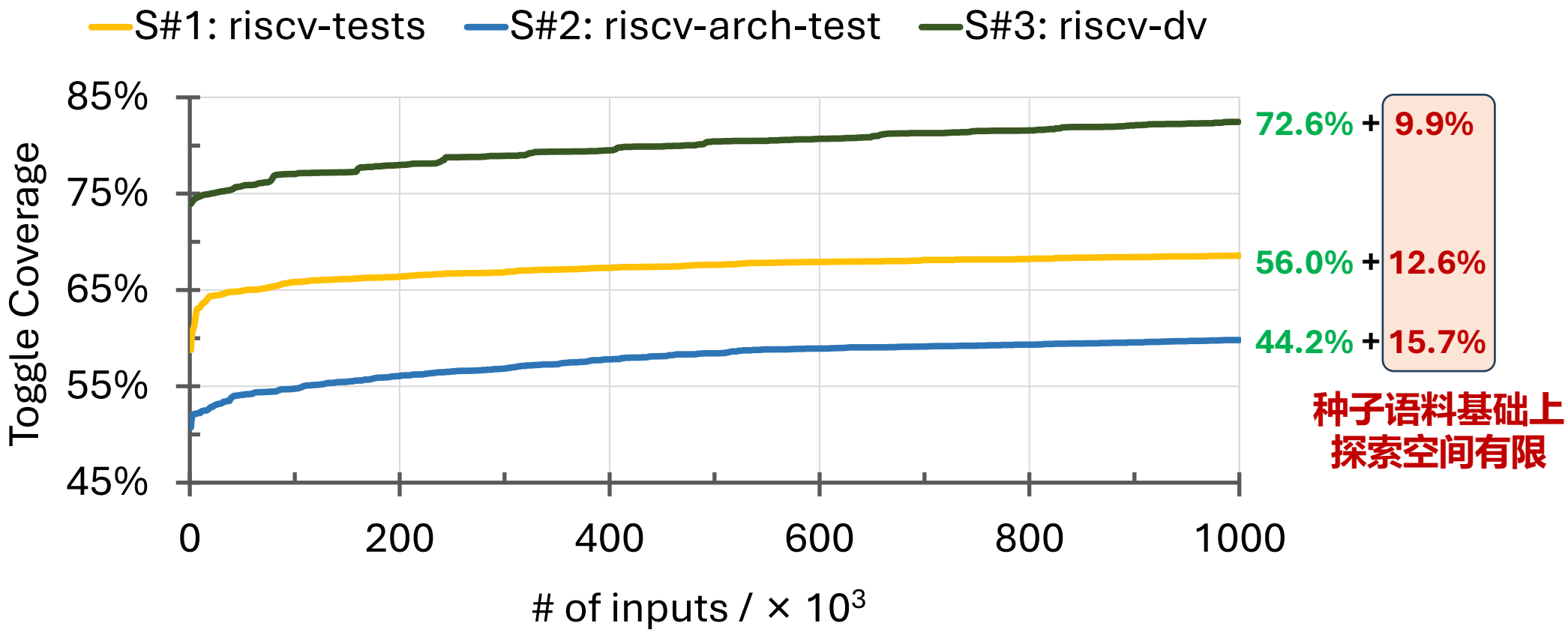
模糊测试 Fuzzing： 设计感知的自动化测试生成



核心： 覆盖率反馈的变异算法

- 以语料库为空间搜索起点
- 利用变异探索待验证空间
- 基于覆盖率指标引导探索

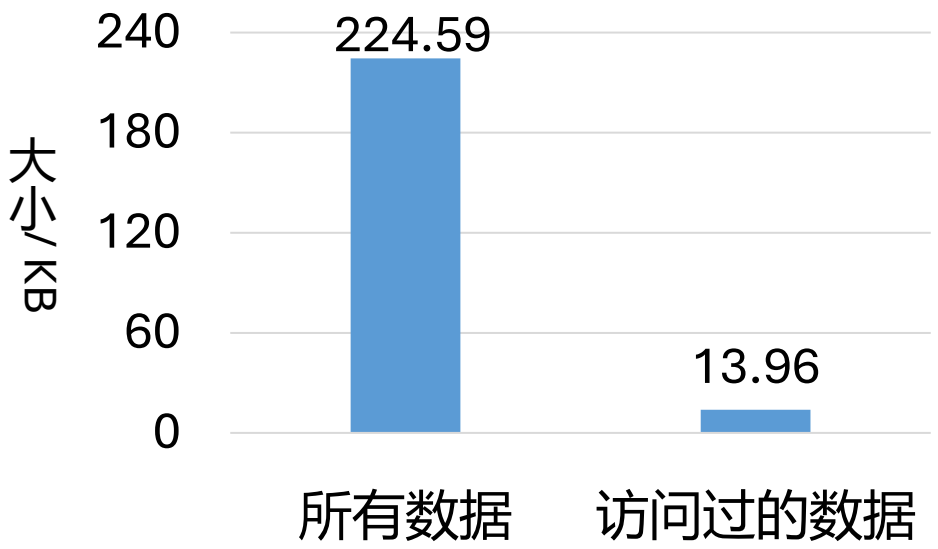
当前进展：验证覆盖率提升有限



不足1：变异效率有限

➢ 局限性1：变异的低效利用

案例：S#3 (riscv-dv)



高达**93.8%**的数据未曾被访问
绝大部分的变异很可能是无效的

➢ 局限性2：变异的过度利用

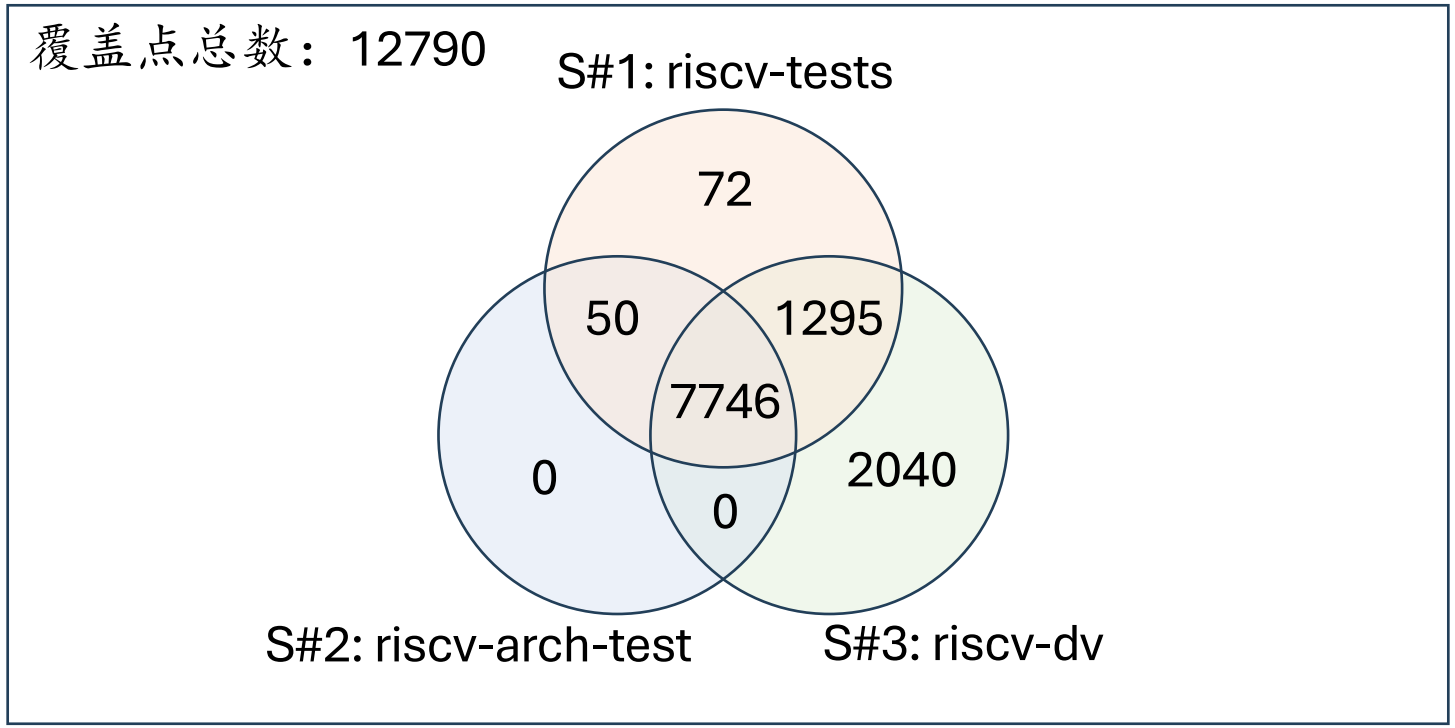
案例：S#4 (force-riscv)

```
0000000080000000 <text0>:  # base
  ……
0000000080000100 <text1>:  # base+0.25KB
  ……
0000000080011000 <text2>:  # base+68KB
  ……
000000008477fff8 <text3>:  # base+71.5MB
  ……
00000000a6411d80 <text77>: # base+612MB
  ……
```

动辄数百MB甚至更大的输入大小
但变异访问越界数据时仍不存在

不足2：种子语料库来源有限

- **观察**：使用不同的种子语料，模糊测试能够探索不同的设计空间
- **但是，动辄数百 MB 的语料如何被用于高效模糊测试?**
  - 例：现实负载，如SPEC CPU2006，可能需要超过 1GB 内存空间



图：使用不同的种子语料进行模糊测试实现的验证覆盖

模糊测试如何解决复杂处理器的验证质量难题

- **动机**：更广泛地**利用现有验证用例**，**站在人类经验知识的肩膀上**
  - 去充分借鉴积累下来的先验经验，而非去取代已有方法

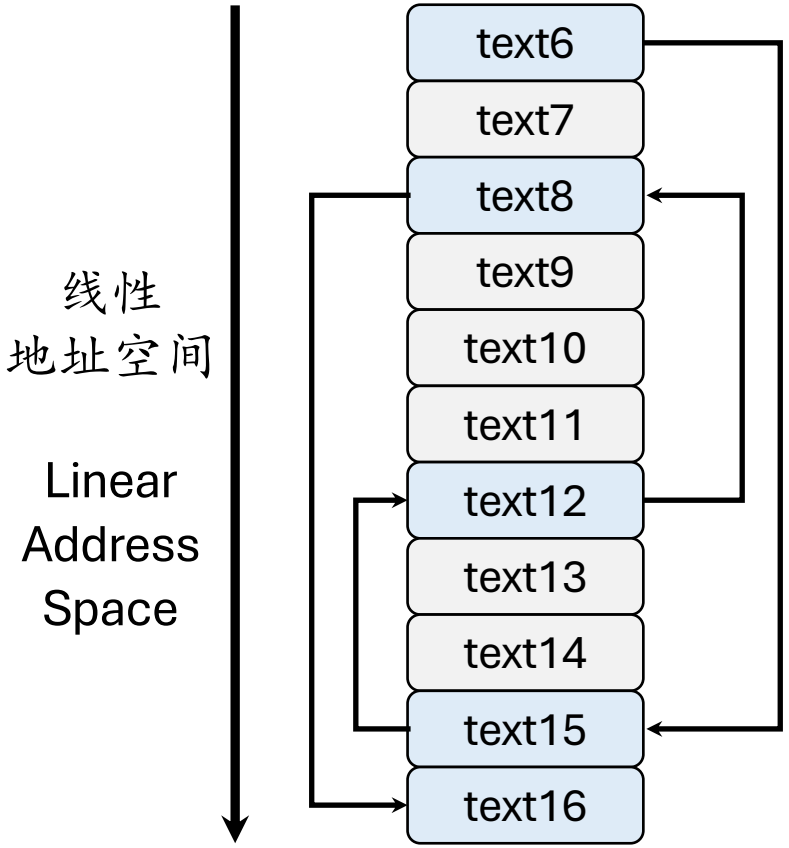
- 以**已有用例**为基础，如何**扩充模糊测试的搜索视野?**

- 优化**模糊测试的基础流程**，形成**更现代的工具和基础设施!**

更高效的**输入格式**、更丰富的**种子语料**和**覆盖率指标**

洞察：现有测试用例组织方式与CPU执行特征不匹配

- **线性内存**按**地址序**存储数据，但**CPU随机访问**不同地址
  - 带来了严重的无用数据、越界访问等问题，限制了种子语料的地址空间

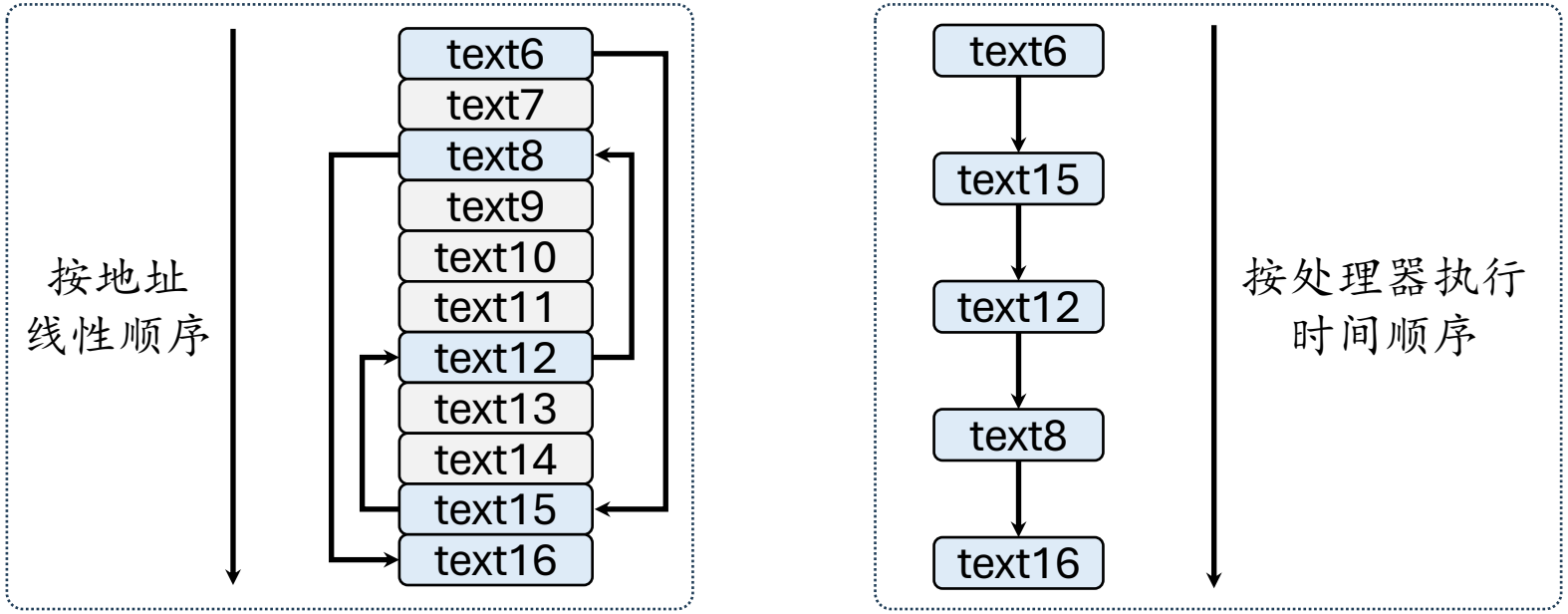



```
00000000880ca508 <text6>:
  ……
  880ca58c:   1d03d6ef    jal a3,0x8810775c
  ……
0000000088106b20 <text8>:
  ……
  88106b28:   62da92e3    bne s5,a3,0x8810794c
  ……
0000000088107068 <text12>:
  ……
  88107088:   a8de7ce3    bgeu t3,a3,0x88106b20
  ……
0000000088107758 <text15>:
  ……
  88107888:   fec65263    bge a2,a2,0x8810706c
  ……
0000000088107948 <text16>:
  ……
```

核心：线性内存格式的不足

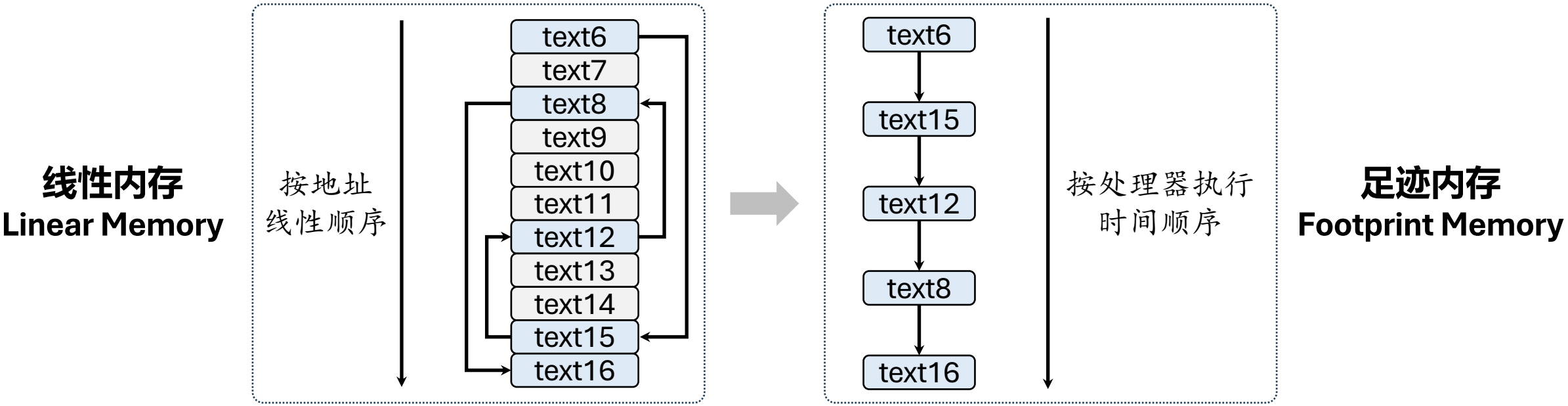
- **动机**：更广泛地利用现有验证用例，站在人类经验知识的肩膀上
- **观察**：现有**线性内存格式**与**处理器执行路径**特征不匹配
  - 输入按地址序组织，但CPU基于指令执行情况随机访问

线性内存
Linear Memory

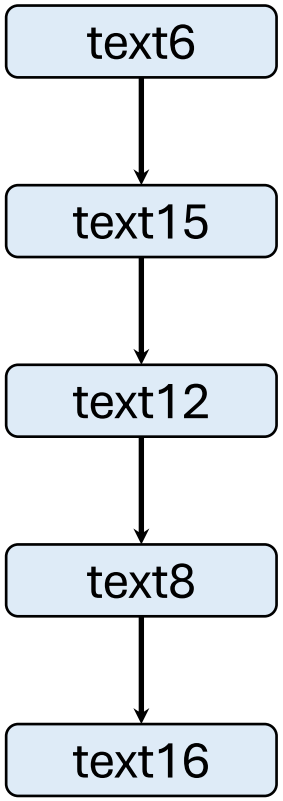


关键技术1：足迹内存

- **动机**：更广泛地利用现有验证用例，站在人类经验知识的肩膀上
- **观察**：现有**线性内存格式**与**处理器执行路径**特征不匹配
  - 输入按地址序组织，但CPU基于指令执行情况随机访问
- **创新方法**：捕获**处理器执行路径**形成**足迹内存格式**

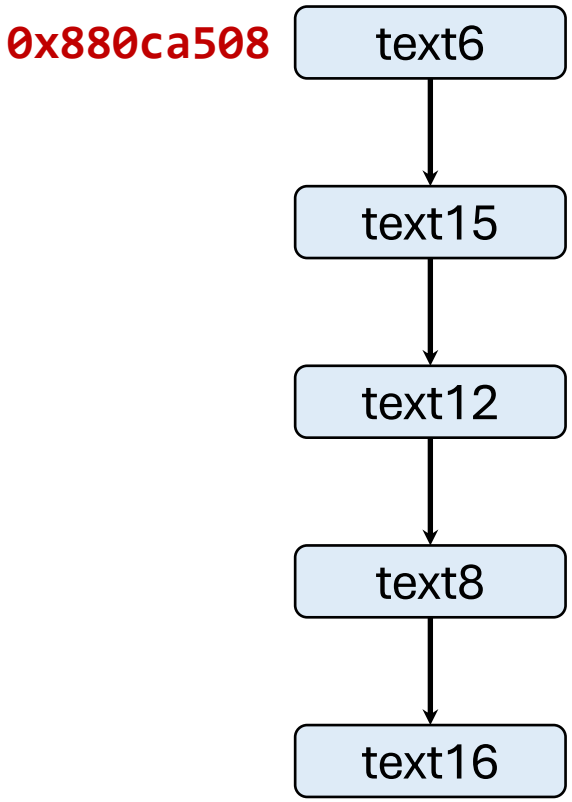


足迹内存：按处理器访问顺序组织数据



足迹内存
Footprint Memory

足迹内存：按处理器访问顺序组织数据



足迹内存
Footprint Memory

```
00000000880ca508 <text6>:
  ……
  880ca58c:   1d03d6ef    jal a3,0x8810775c
  ……
```

……

……

……

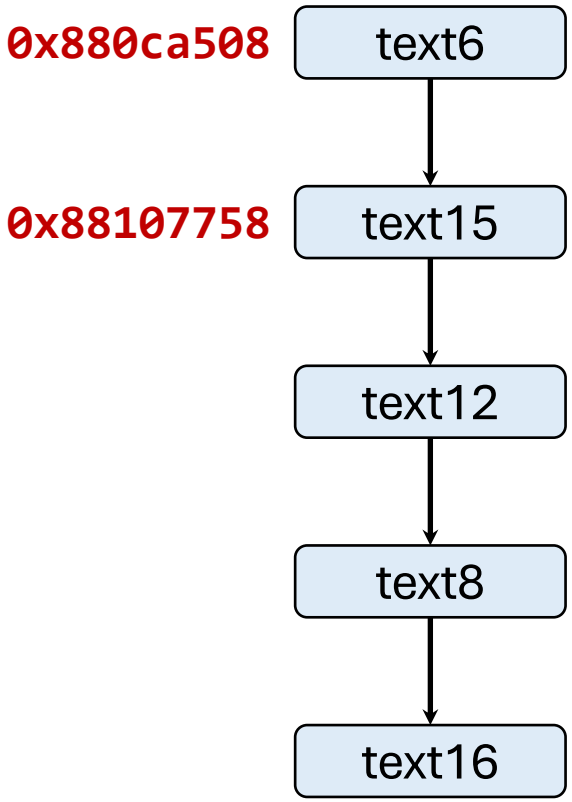
……

……

……

……

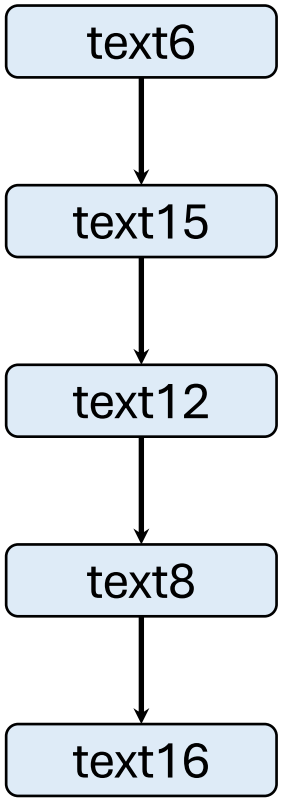
足迹内存：按处理器访问顺序组织数据



足迹内存
Footprint Memory

```
00000000880ca508 <text6>:
  ……
  880ca58c:   1d03d6ef    jal a3,0x8810775c
  ……

  ……

  ……

  ……

  ……
0000000088107758 <text15>:
  ……
  88107888:   fec65263    bge a2,a2,0x8810706c
  ……

  ……
```

足迹内存：按处理器访问顺序组织数据

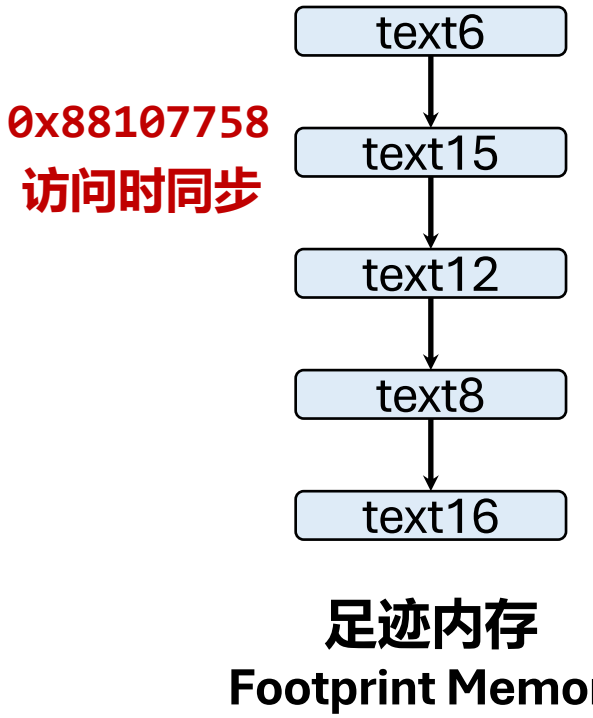


足迹内存
Footprint Memory

```
00000000880ca508 <text6>:
  ……
  880ca58c:   1d03d6ef    jal a3,0x8810775c
  ……
0000000088106b20 <text8>:
  ……
  88106b28:   62da92e3    bne s5,a3,0x8810794c
  ……
0000000088107068 <text12>:
  ……
  88107088:   a8de7ce3    bgeu t3,a3,0x88106b20
  ……
0000000088107758 <text15>:
  ……
  88107888:   fec65263    bge a2,a2,0x8810706c
  ……
0000000088107948 <text16>:
  ……
```

如何进行基于足迹内存的协同仿真验证

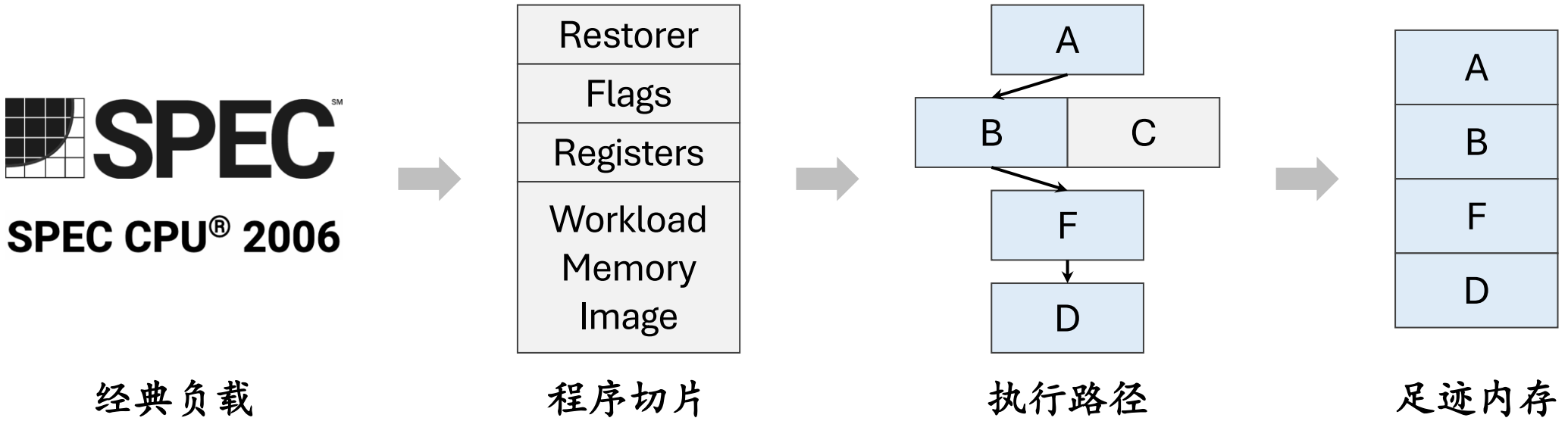
- **背景**：协同仿真需要 DUT 与 REF 具有**相同的内存初始值**
- **问题**：足迹内存的**语义** (地址-数据映射关系) **与处理器微结构相关**
- **解决方法**：**按需内存同步**，而非初始化时一次性同步



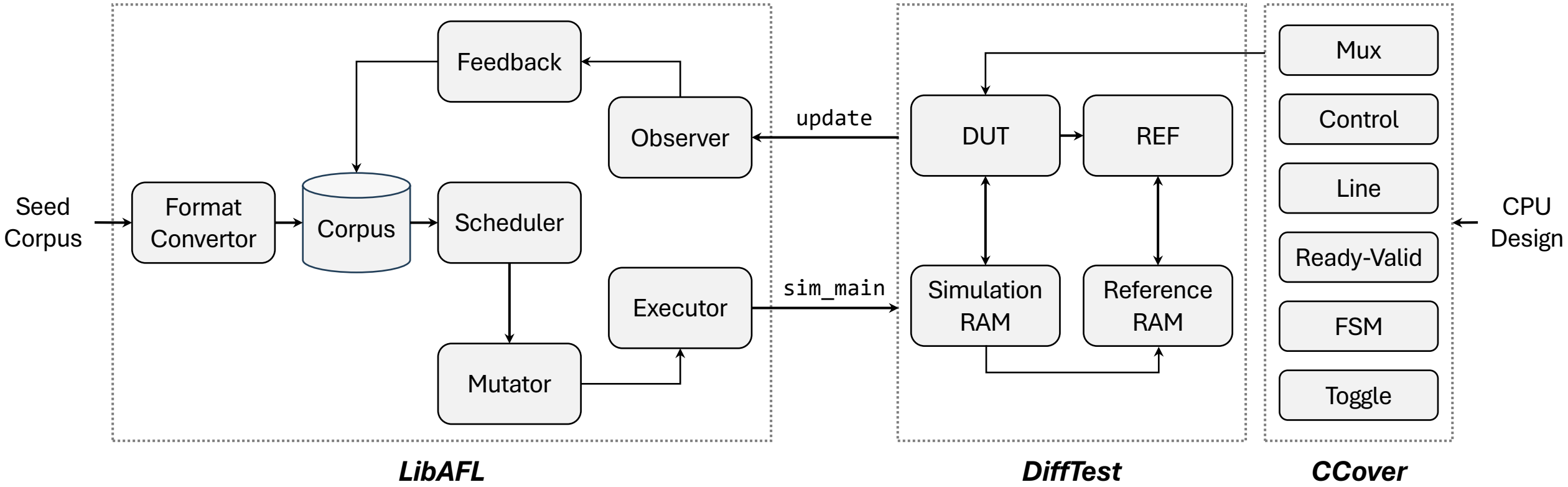

```
0000000880ca508 <text6>:
  ……
  880ca58c:   1d03d6ef    jal a3,0x8810775c
  ……
  ……
  ……
0000000088107758 <text15>:
  ……
  88107888:   fec65263    bge a2,a2,0x8810706c
  ……
```

关键技术2：扩充种子语料

- 引入**大规模现实世界负载**，捕捉它们的执行路径，形成高质量语料
  - 利用**程序切片**技术进行**寄存器状态**的保存和恢复



面向执行路径的处理器模糊测试工作流程



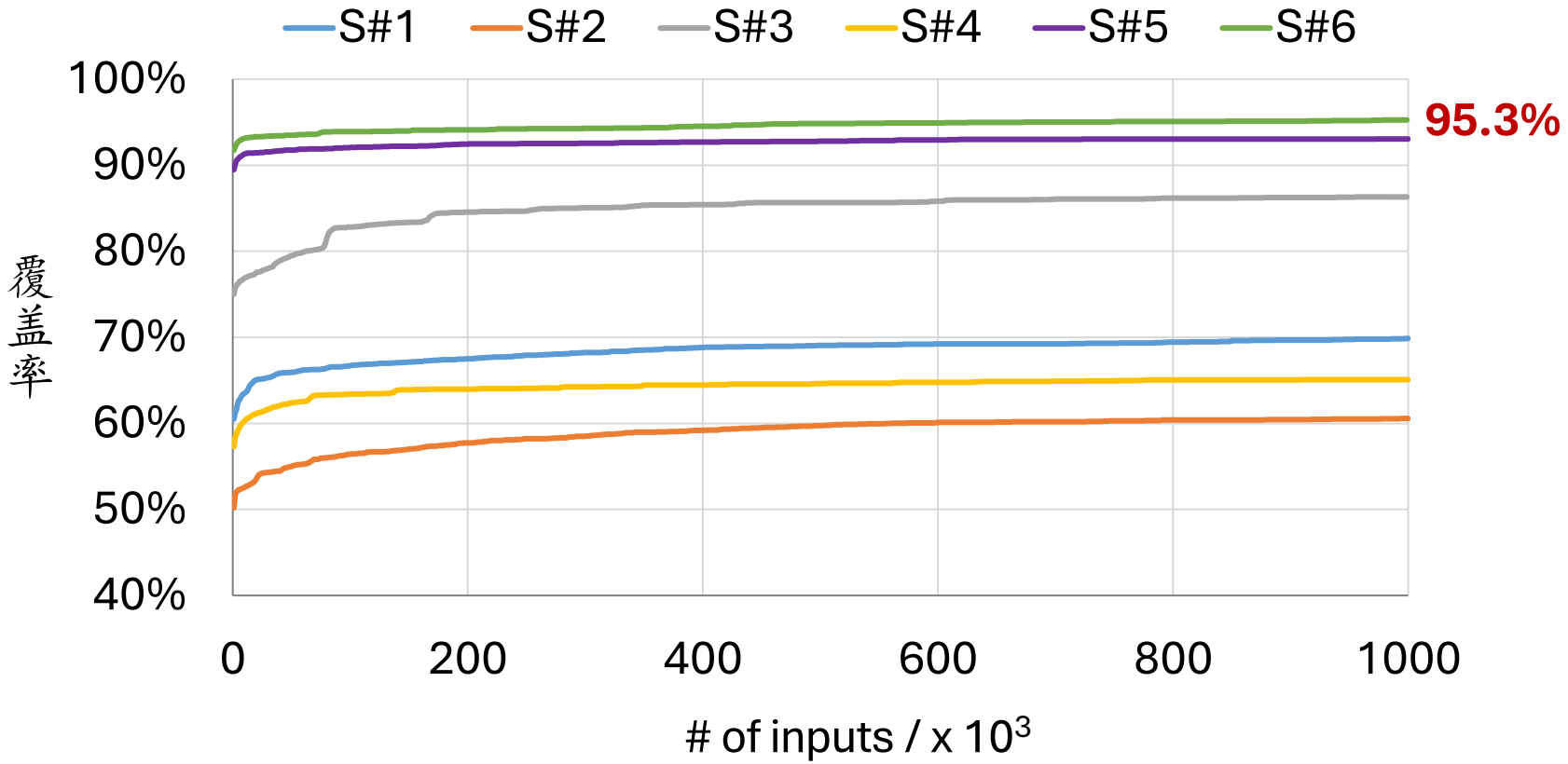
PathFuzz 工具链

- 本工作**实现并开源**了一套面向RISC-V处理器的模糊测试**工具链**
  - https://github.com/OpenXiangShan/xfuzz

- **兼容 DiffTest 的用户侧接口**，接管其**输入逻辑**
  - 自动化地完成覆盖率指标的插桩、收集、反馈
  - 模糊测试算法层使用软件开源社区的通用模糊测试框架 LibAFL
  - 正确性判断使用课题研究内容1提出的协同仿真框架 DiffTest
  - 通过开源项目的组合为社区提供基础研究平台

实验评估：达到更高验证覆盖率

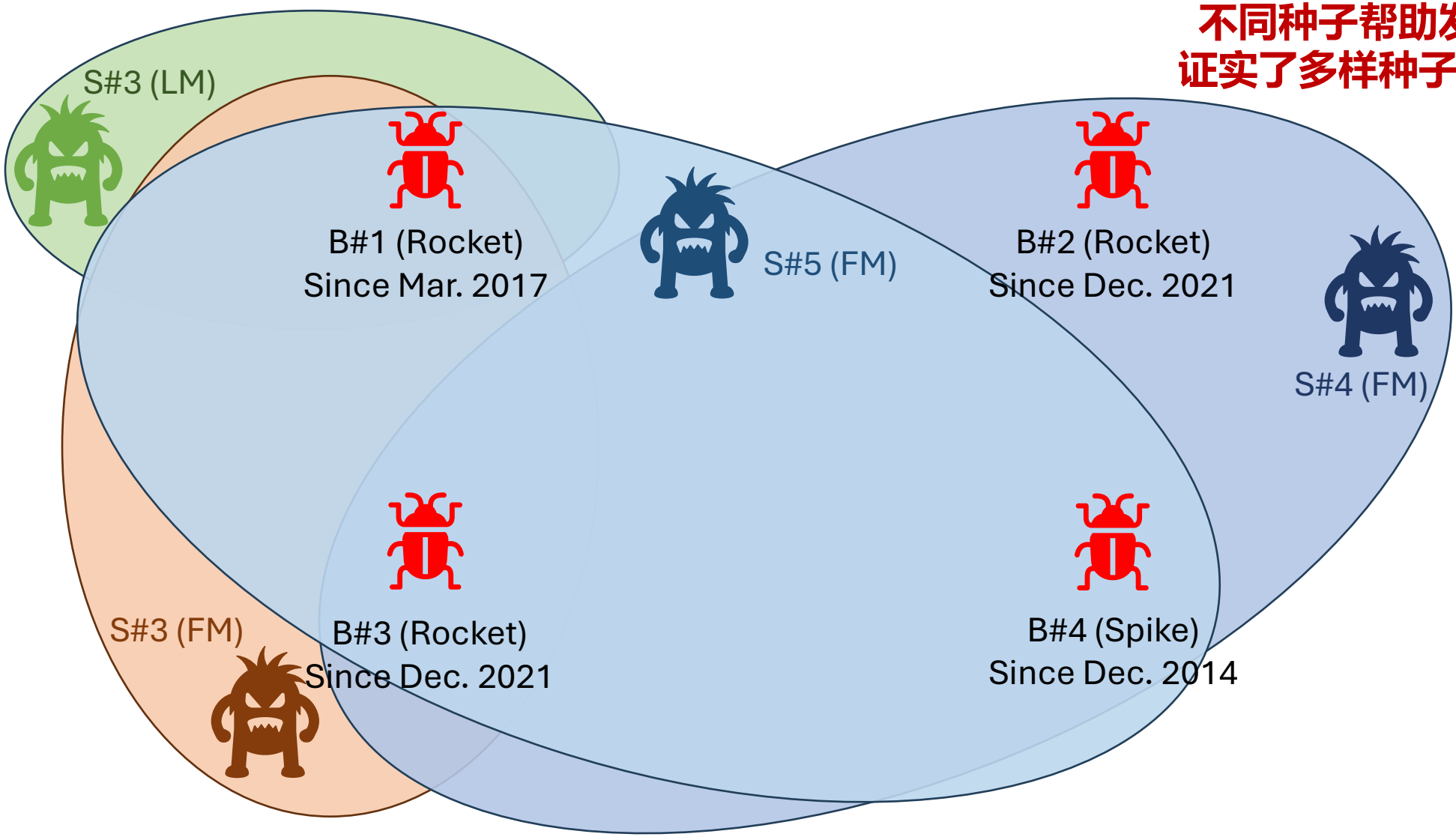
- 本工作在 **15 小时**内实现最高 **95.3%** 的验证覆盖率 (Rocket)
  - SOTA 工作[1]结合形式化验证工具在72小时内达到94.9%覆盖率 (CVA6)
  - 本工作：更复杂待测设计、更难达到覆盖率、更自动方法、更短时间



[1] Chen Chen, Rahul Kande, Nathan Nguyen, Flemming Andersen, and Aakash Tyagi. HyPFuzz: Formal-Assisted Processor Fuzzing. USENIX Security '23.

实验评估：发现设计缺陷

不同种子帮助发现不同漏洞
证实了多样种子语料的重要性



应用至“香山”处理器核：发现10+个潜在设计漏洞

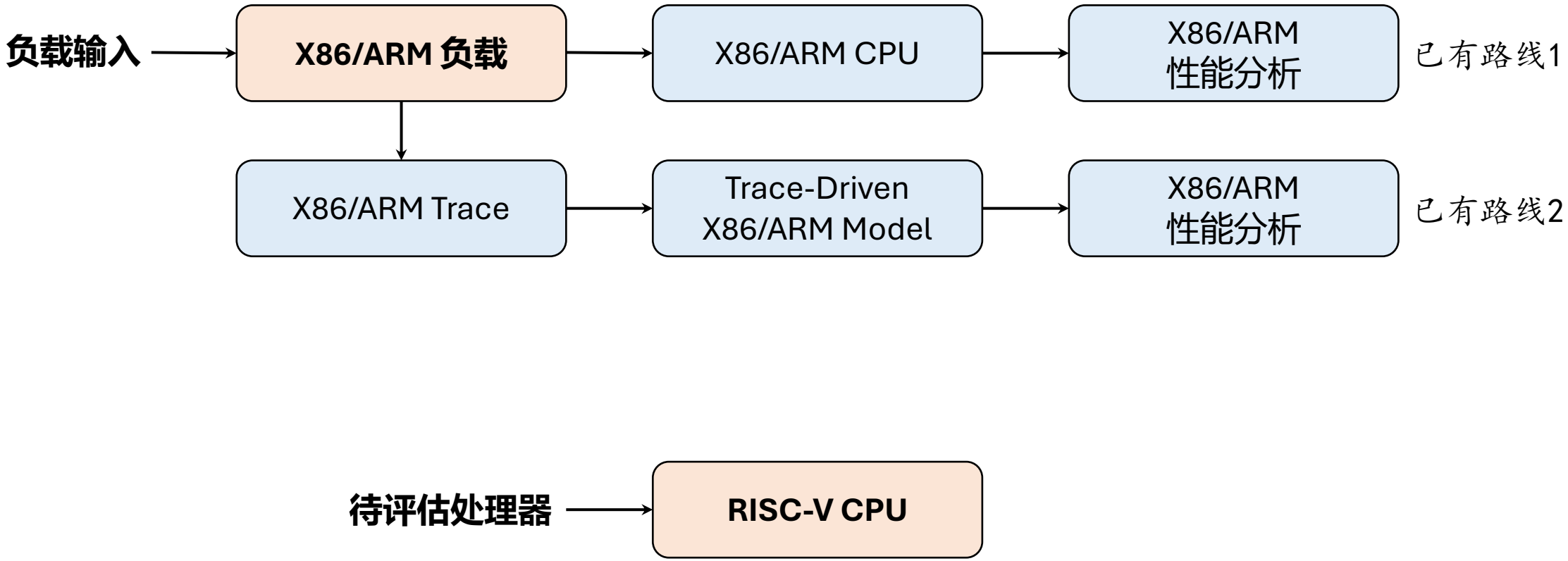
版本	报错比例（种子语料）	发现、确认并修复的设计缺陷
20230905	725 / 50000	1. (香山) 越界访存指令未报异常 2. (香山) 保留的舍入模式未报异常 3. (香山) mstatus CSR 的 fs 域被错误赋值 4. (香山) 越界取指未报异常 5. (香山) fmadd.d 指令计算错误 6. (香山) 对 MMIO 地址的访问未被正确识别 7. (香山) ebreak 指令实现错误 8. (香山) mstatus CSR 的保留位被错误写入 9. (香山) mie CSR 的保留位被错误写入 10. (香山) 非法指令异常时 mtval CSR 赋值错误 11. (NEMU) 浮点或向量指令未检查 mstatus CSR 的 fs 或 vs 域 12. (NEMU) 未对 c.lui 保留指令报异常
20230907	9531 / 300000	
20230915	87 / 1838 (S#1, LM)	
	179 / 25087 (S#1, FM)	
	16 / 3772 (S#2, LM)	
	25 / 4132 (S#2, FM)	
	174 / 2181 (S#3, LM)	
	476 / 2532 (S#3, FM)	
	133 / 2196 (S#4, FM)	
	121 / 3751 (S#6, FM)	
20241028	400 / 10000 (S#1, LM)	
	993 / 10000 (S#2, LM)	
	523 / 10000 (S#3, LM)	

问题4：早期硬件设计 + 弱软件生态下的快速性能评估

- 设计了一个RISC-V CPU
- 但是用户负载还跑不起来
- CPU自己的功能也还很弱

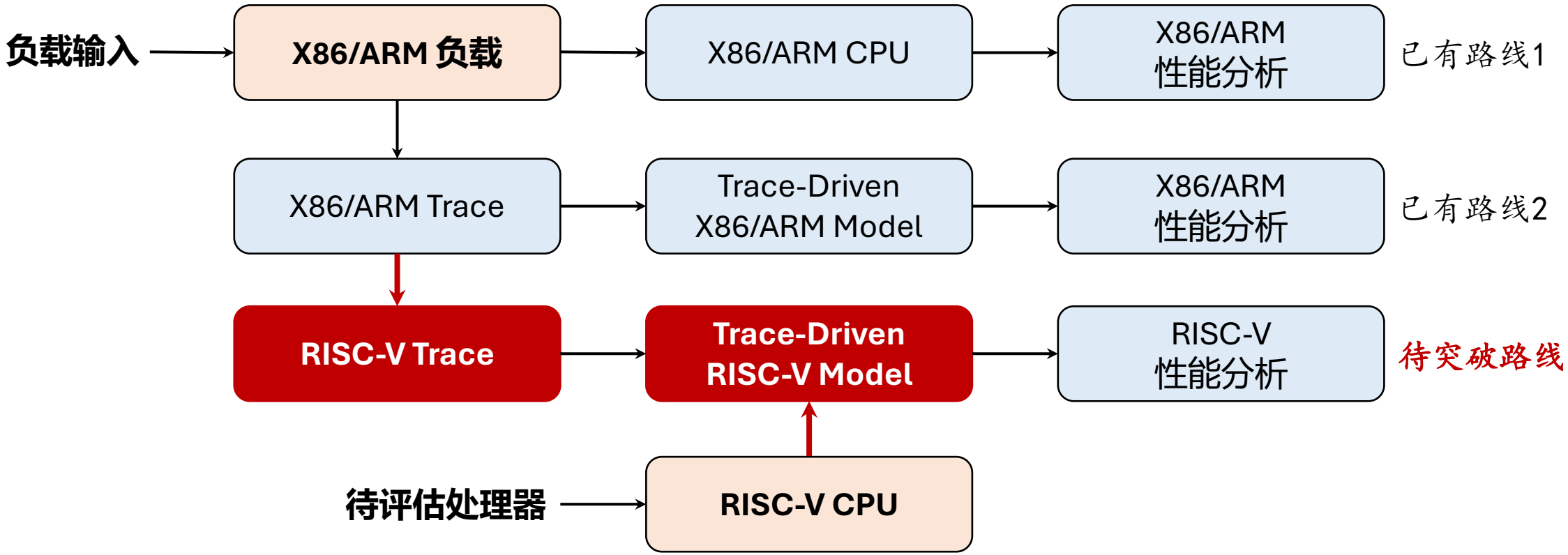
- 怎么快速评估CPU性能?

如何评估 x86/ARM 负载在 RISC-V 的性能?



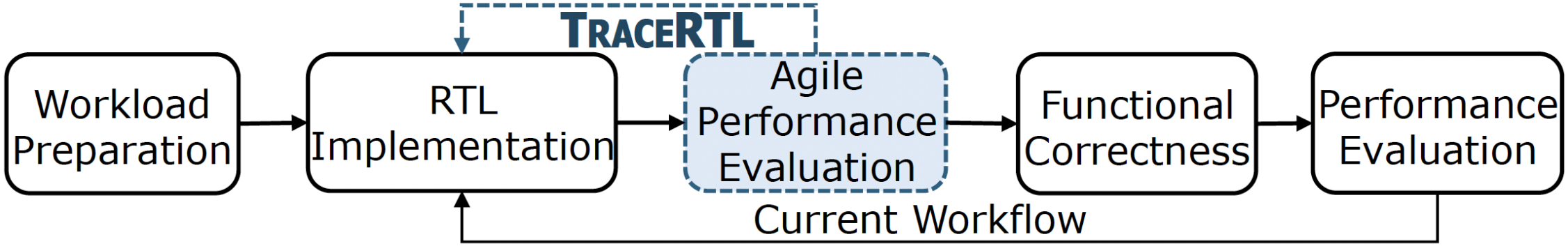
跨 ISA Trace 转换 + Trace 输入处理器

- 实现在不够成熟生态处理器上评估成熟生态软件负载的性能



TraceRTL: 构建 Trace-Driven 处理器性能模型

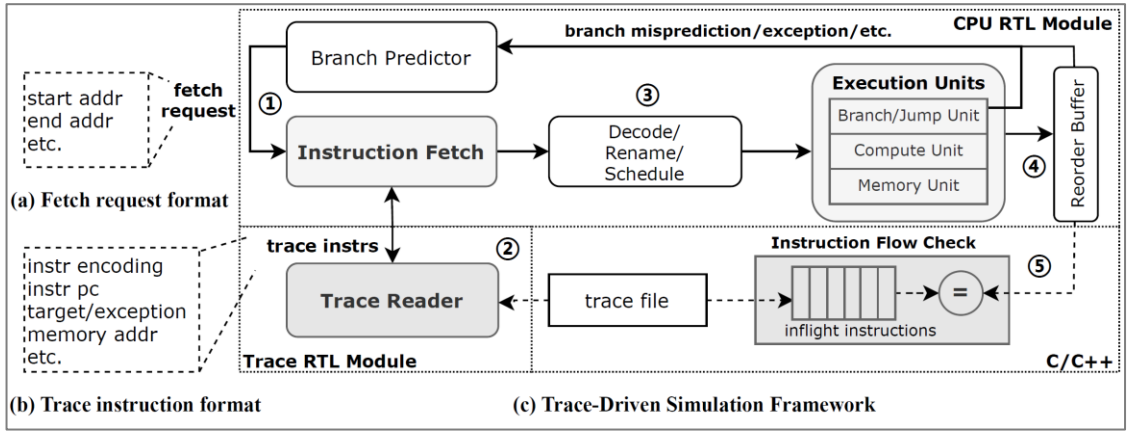
- 洞察：只需要**必要的性能组件**，无需保证完整CPU功能正确性



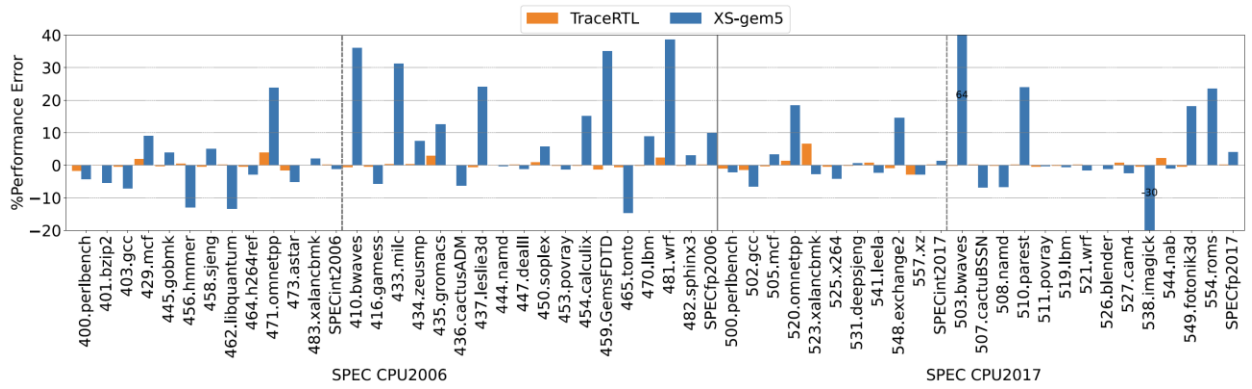
TraceRTL: 构建 Trace-Driven 处理器性能模型

- 洞察：只需要**必要的性能组件**，无需保证完整CPU功能正确性

- **自动化、低侵入**地改造并维护一个 Trace 驱动的香山
  - 消除功能依赖：Chisel、电路、架构、功能抽象、性能敏感、……
  - 弥补性能误差：Trace 中丢失的信息的影响，包括地址、数据、……



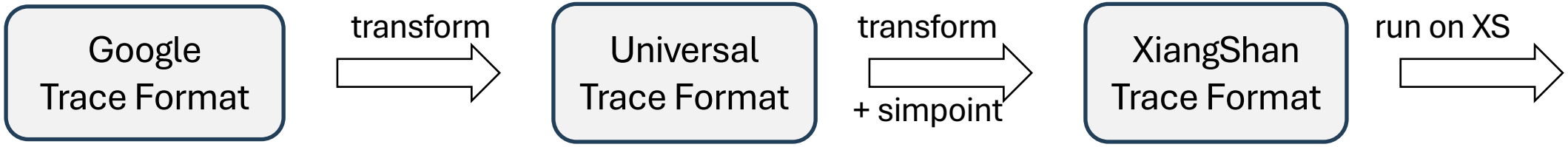
图：基于香山改造形成Trace-driven CPU



图：SPECint 2017平均性能误差0.13%，
相比XS-gem5减少10.3倍

Trace 转换和采样机制

- 原始Trace为x86且过长，对性能分析不友好，需转换为短小的RISC-V Trace



① 解析 trace 信息

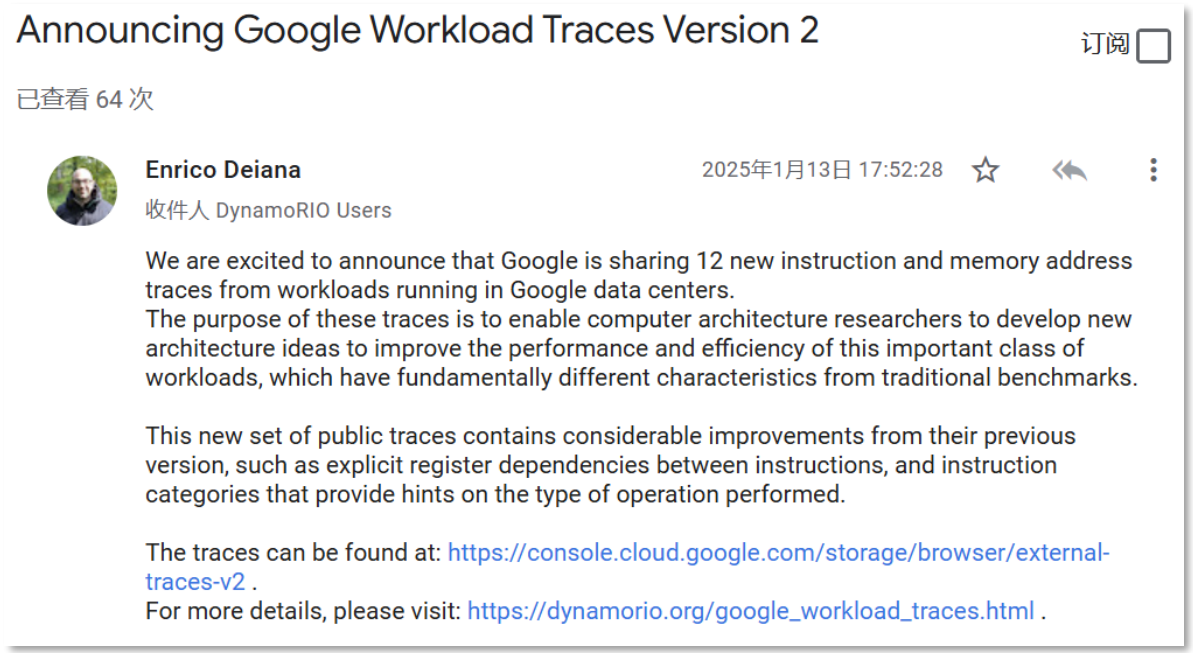
1. 取指：**PC** / size
2. 指令 synthetic encoding：
   - category
   - src-reg / dst-reg
3. 跳转：branch-type/**target/taken**
4. 访存：memory-type/size/**addr**
5. 其他：?prefetch/?flush

② 转换 trace 格式/内容

1. 取指：**PC**
2. **指令 riscv-encoding**
3. 跳转：**branch-target/taken**
4. 访存：**memory-addr**

谷歌公司发布Google Workload Traces (Version 2)

- 2025年1月13日，谷歌公司宣布发布 V2 版本 Google Workload Traces
  - 支持体系结构研究人员针对谷歌数据中心负载开展性能和效率优化研究工作
  - 发布于DynamoRIO (一个程序动态插桩工具) 用户讨论组；V1版本发布于2022年3月



Announcing Google Workload Traces Version 2

订阅

已查看 64 次

Enrico Deiana 2025年1月13日 17:52:28

收件人 DynamoRIO Users

We are excited to announce that Google is sharing 12 new instruction and memory address traces from workloads running in Google data centers.
The purpose of these traces is to enable computer architecture researchers to develop new architecture ideas to improve the performance and efficiency of this important class of workloads, which have fundamentally different characteristics from traditional benchmarks.

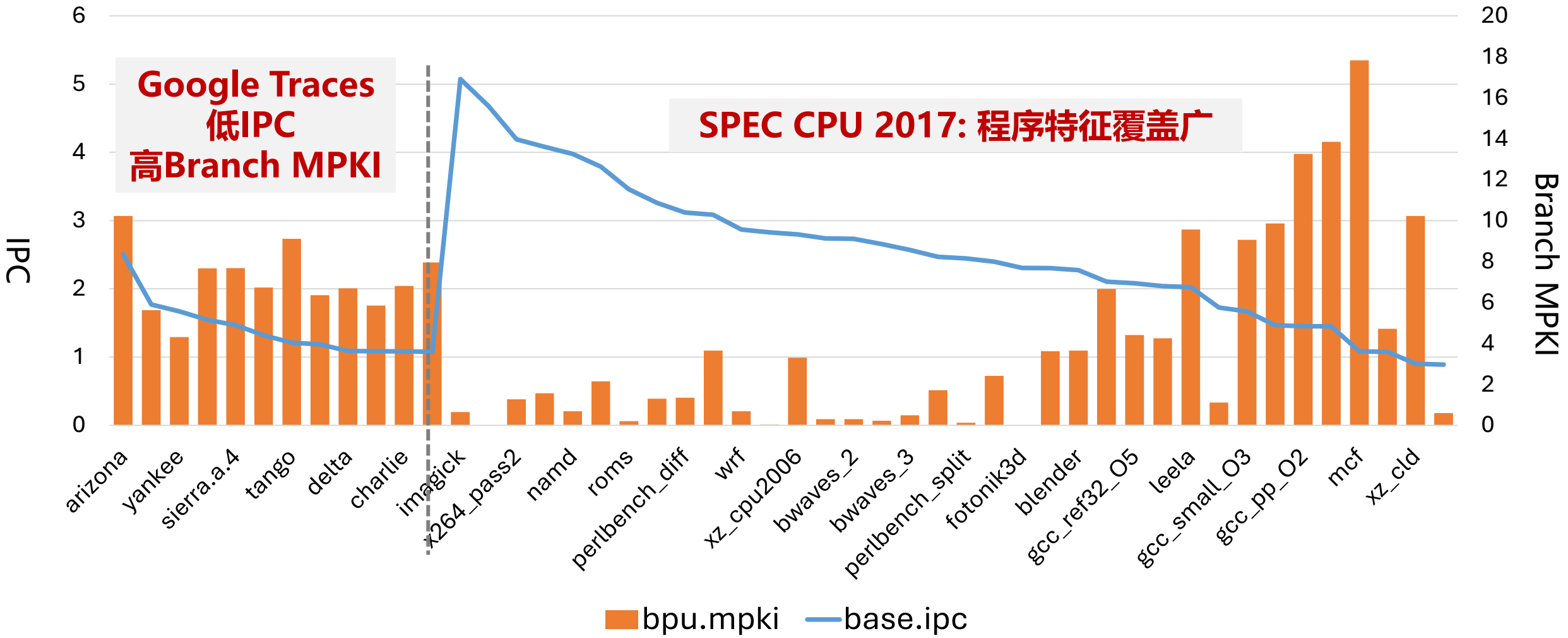
This new set of public traces contains considerable improvements from their previous version, such as explicit register dependencies between instructions, and instruction categories that provide hints on the type of operation performed.

The traces can be found at: https://console.cloud.google.com/storage/browser/external-traces-v2 .
For more details, please visit: https://dynamorio.org/google_workload_traces.html .

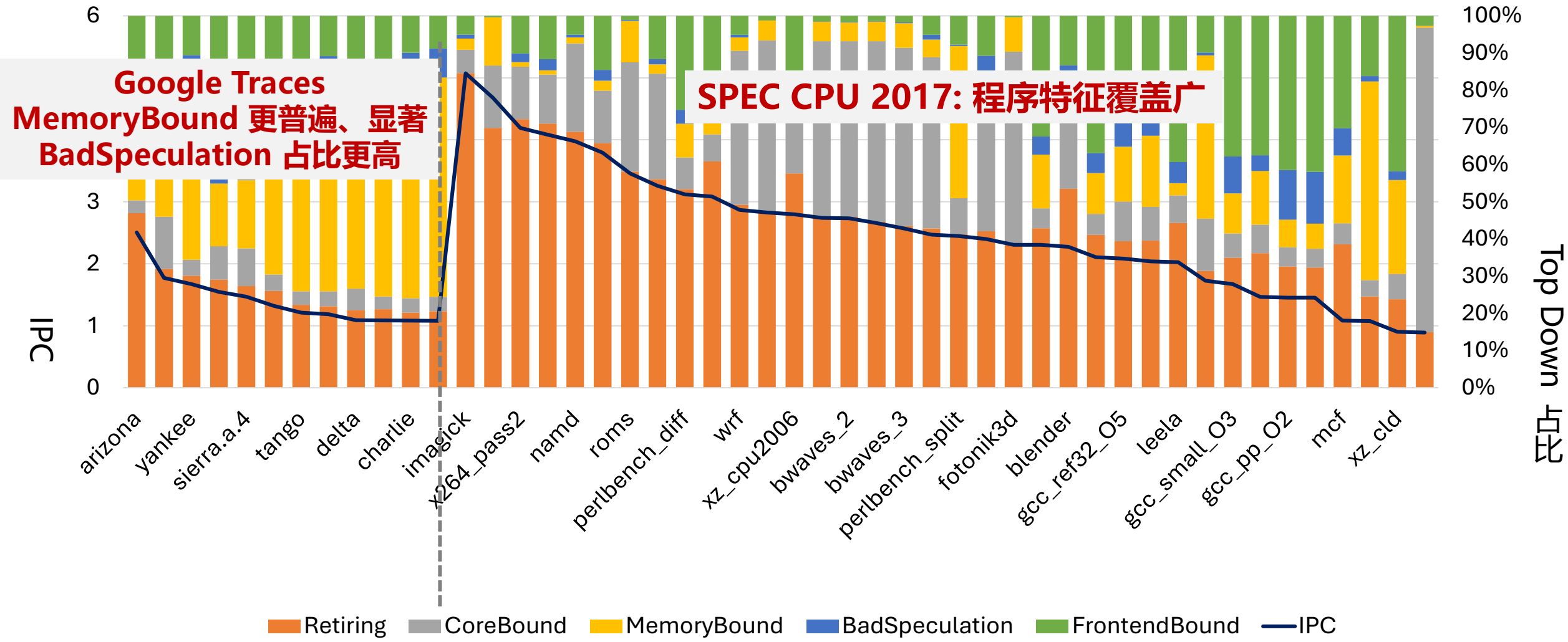
https://groups.google.com/g/DynamoRIO-Users/c/z7vZkLILf-U

问题随之而来：如何利用谷歌数据中心 x86 Trace 优化 RISC-V CPU 性能？

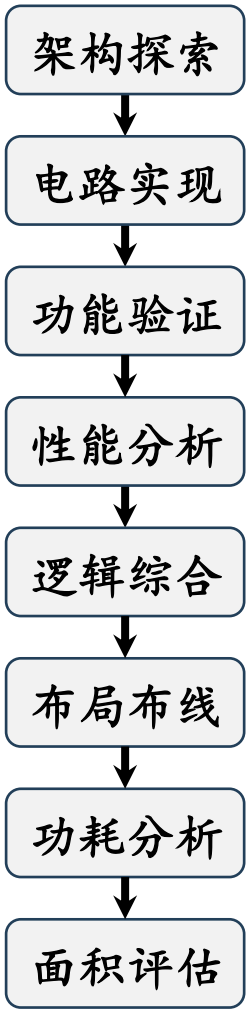
基于香山分析谷歌数据中心负载：IPC/分支误预测



基于香山分析谷歌数据中心负载： Top-Down

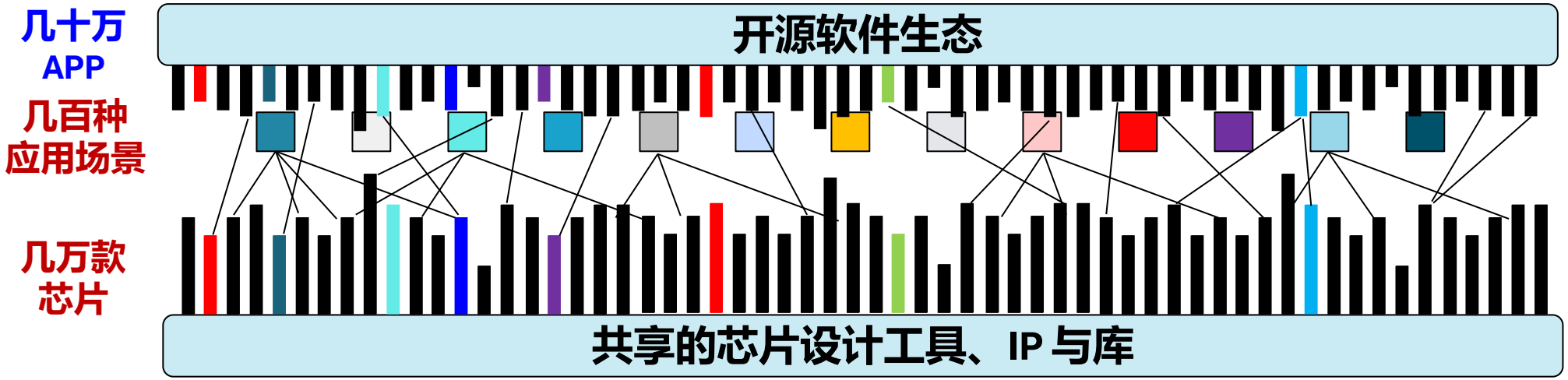


处理器开发流程中更多的研究机会



流程	研究方向	香山团队工作	代表性社区工作
架构探索	设计空间探索(DSE)		BOOM-Explorer
电路实现	编程语言、设计方法/范式	OOA, ChiselDFT	Chisel, PyMTL, Bluespec
功能验证	协同仿真、仿真加速	DiffTest, TL-Test, CHIron	Dromajo, FireSim
	测试生成	xfuzz	RFUZZ, riscv-dv
	调试方法	LightSSS, ChiselDB	REMU, DESSERT
	形式化验证		CHA, ChiselFV, HyPFuzz
性能分析	负载抽象	LibCheckpoint, Deterload	Simpoint, LoopPoint
	性能瓶颈分析	PerfCCT, Cliff	TopDown, TIP, TEA
	性能建模	TraceRTL, AMCE	GEM5, ChampSim
逻辑综合	综合结果预测		SNS, SNSv2
布局布线	布局优化	(iEDA)	(Google)
功耗分析	功耗预测		APOLLO
面积评估	面积预测		
.....	{Arch, EDA} × {AI/ML, ASIC, FPGA, PL, Compiler, SE, Security, Formal, Sys, HPC, ...}		

处理器敏捷开发的挑战和机遇——欢迎交流合作！



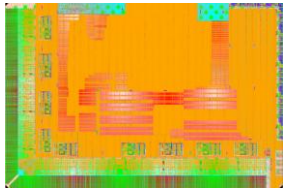
几十个
处理器开发
流程节点

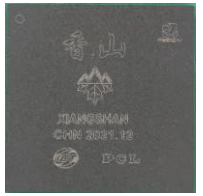
如何实现
处理器开发的
全流程敏捷

谢谢！请批评指正！

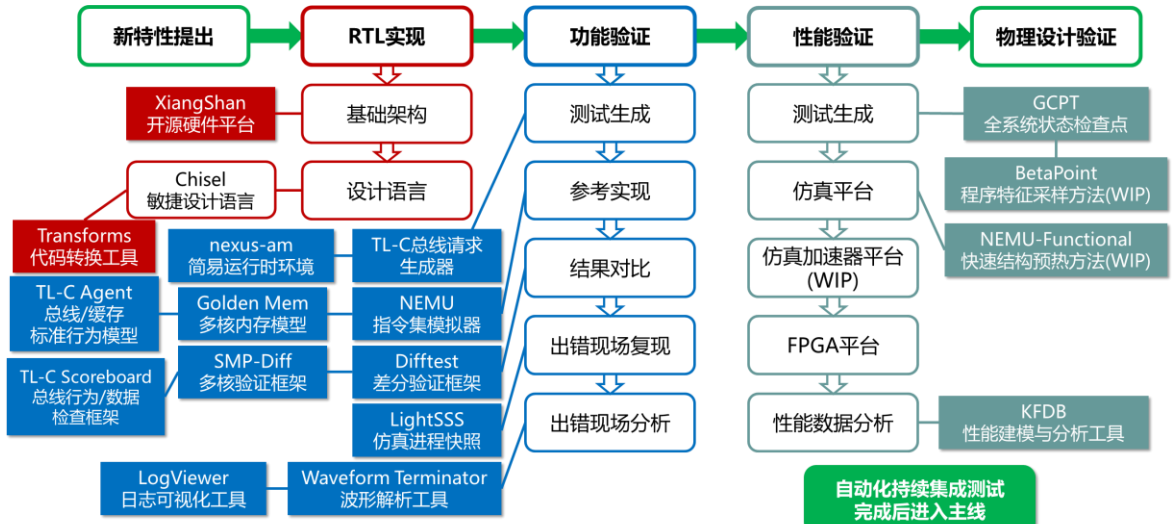
中国科学院计算技术研究所 INSTITUTE OF COMPUTING TECHNOLOGY, CHINESE ACADEMY OF SCIENCES

- **香山核心价值**是构建一套**芯片敏捷设计基础设施**，缩短迭代优化周期
- **开源开放能力体系**，联合企业加速处理器研发节奏，支持按需快速定制芯片





成果开源



能力开放