

Fast, Accurate, and Novel Performance Evaluations with PinCPU

gem5 Workshop @ ISCA • June 22, 2025

Nicholas Mosier,¹ Hamed Nemati,² John C. Mitchell,¹ Caroline Trippel¹

¹ Stanford University ² KTH Royal Institute of Technology

Detailed end-to-end simulation of modern benchmarks in gem5 is intractible.



SPEC CPU2017 contains **>160 trillion instructions** total

+



gem5's O3CPU simulates **≈140 KIPS** on host

=



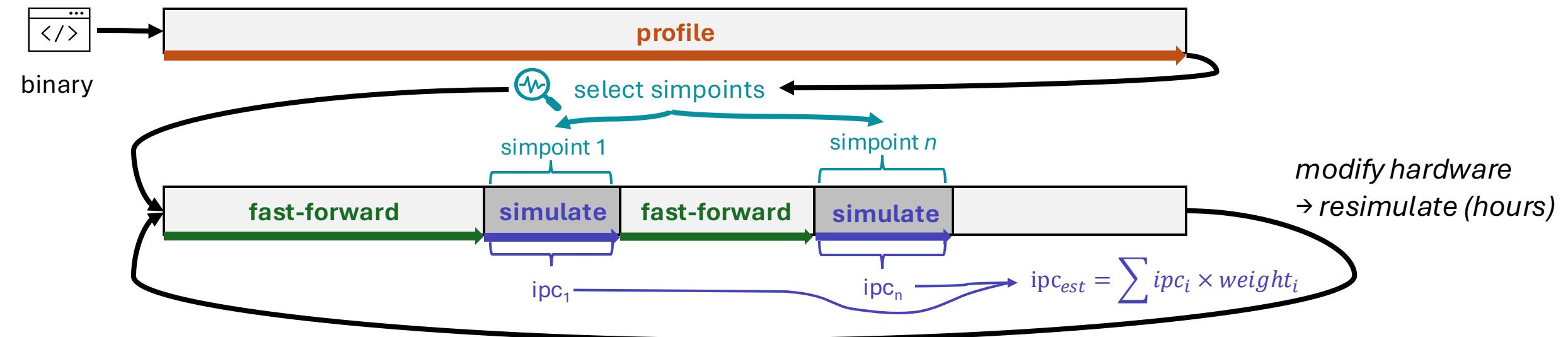
> 36 host CPU-years to simulate all of SPEC CPU2017 with O3CPU

Solution: SimPoint

[Perelman+ SIGMETRICS'03]

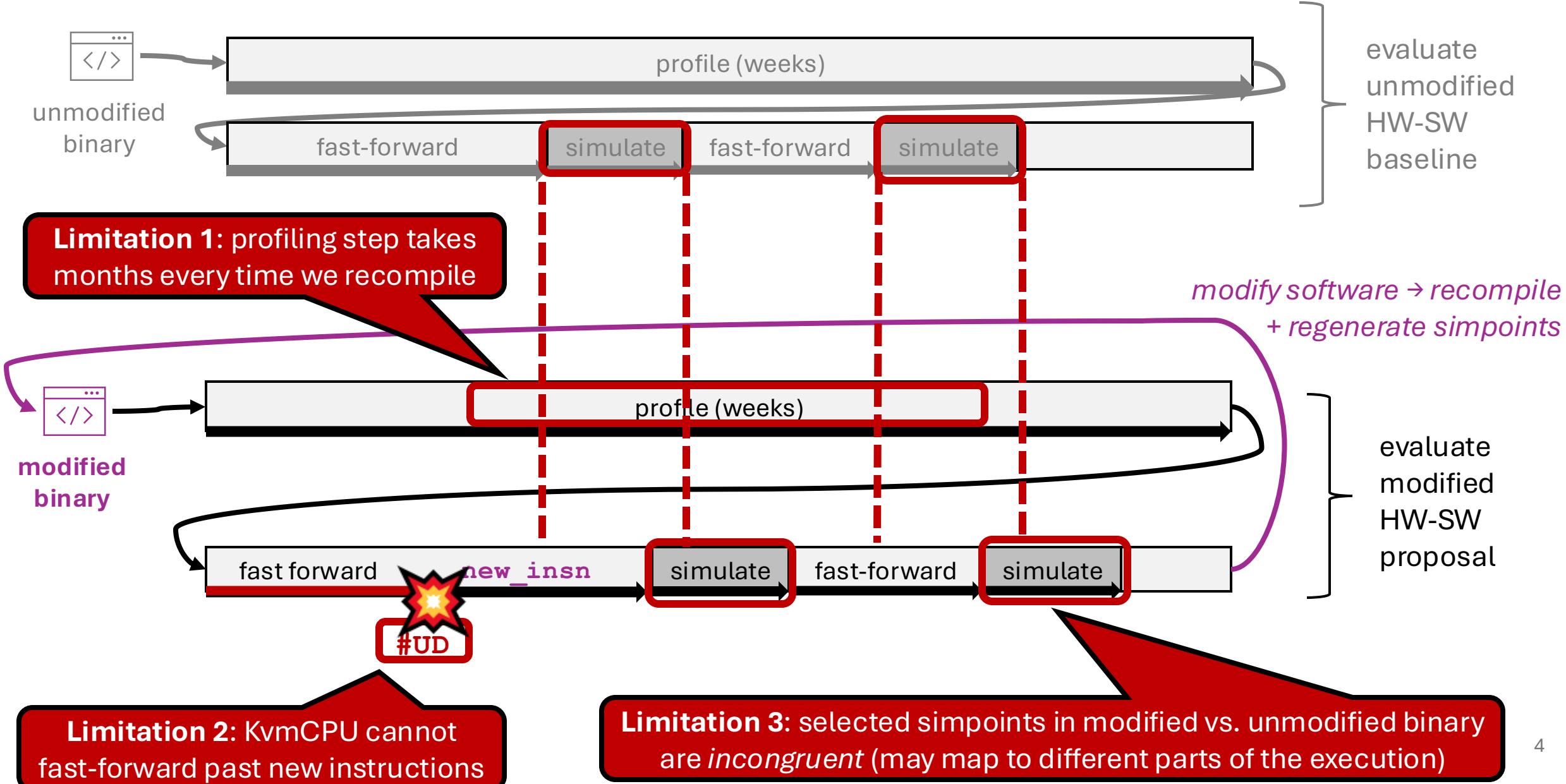
SimPoint is a phased-based sampling technique:

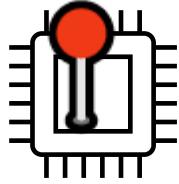
1. **Profile** the workload using **AtomicSimpleCPU** (weeks)
2. **Select** $\leq k$ representative regions (simpoints) (seconds)
3. **Fast-forward** to each simpoint using **KvmCPU** (minutes)
4. **Simulate** each simpoint using **O3CPU** (hours)



Can we use SimPoint to evaluate hardware-software co-designs in gem5?

SimPoint for HW-SW co-design





We present PinCPU: gem5's first fast-forwarding CPU model to support dynamic binary instrumentation.

Limitation 1: profiling step takes months every time we recompile

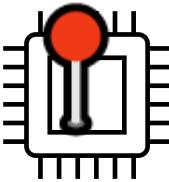


Key observation:

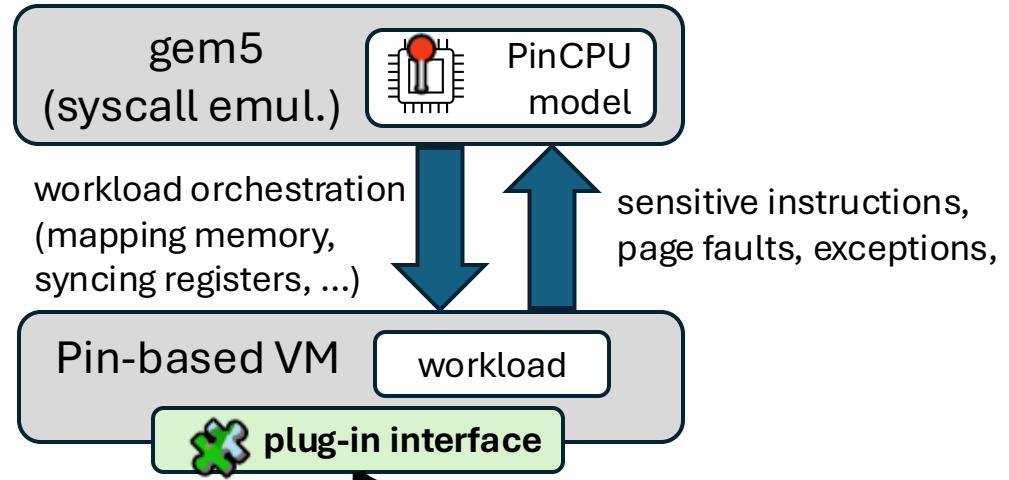
We can resolve all these limitations if gem5 could **dynamically instrument the execution...**

Limitation 2: cannot fast-forward past new instructions

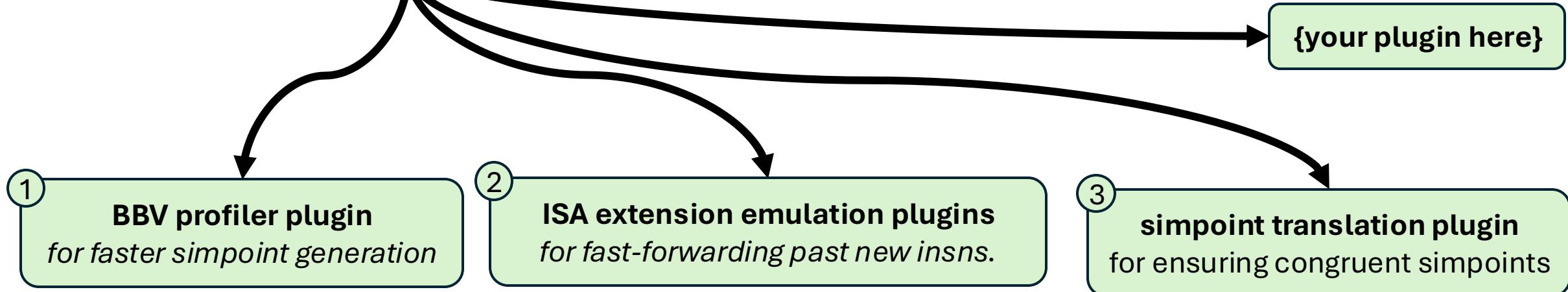
Limitation 3: selected simpoints in modified vs. unmodified binary may map to different parts of the execution



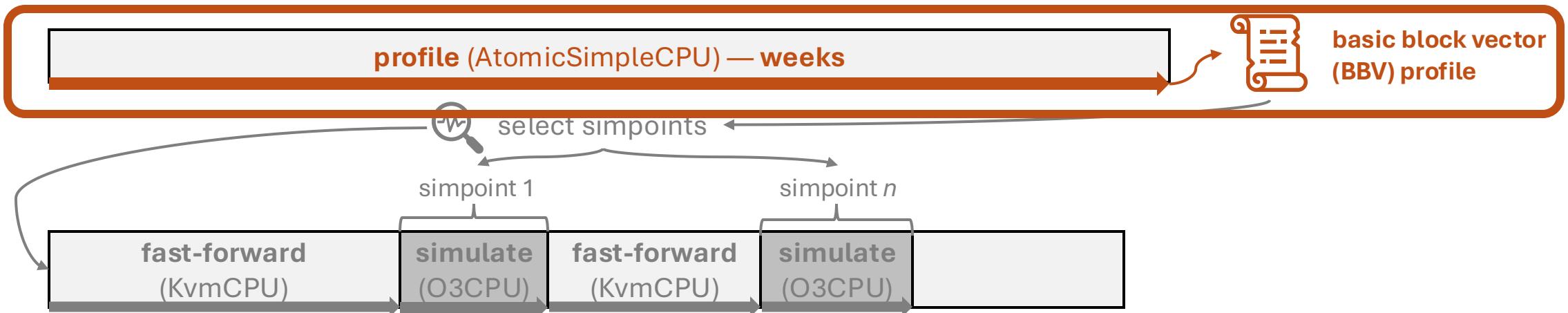
PinCPU uses Intel Pin [Luk+ PLDI'05] to allow gem5 users to both **fast-forward** and **dynamically instrument** userspace workloads.



CPU model	Description	Host insn rate (approx.)
O3CPU	Out-of-order simulated CPU	140 KIPS
AtomicSimpleCPU	Fastest simulated CPU	1.6 MIPS
KvmCPU	KVM-based native CPU	8.6 GIPS
PinCPU	Pin-based native CPU supporting DBI	3.4 GIPS

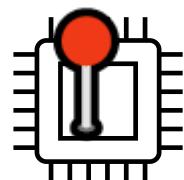


Recall Limitation 1: slow simpoint generation



A **basic block vector (BBV)** counts the dynamic executions of each basic block in a given n -instruction interval. Collection method:

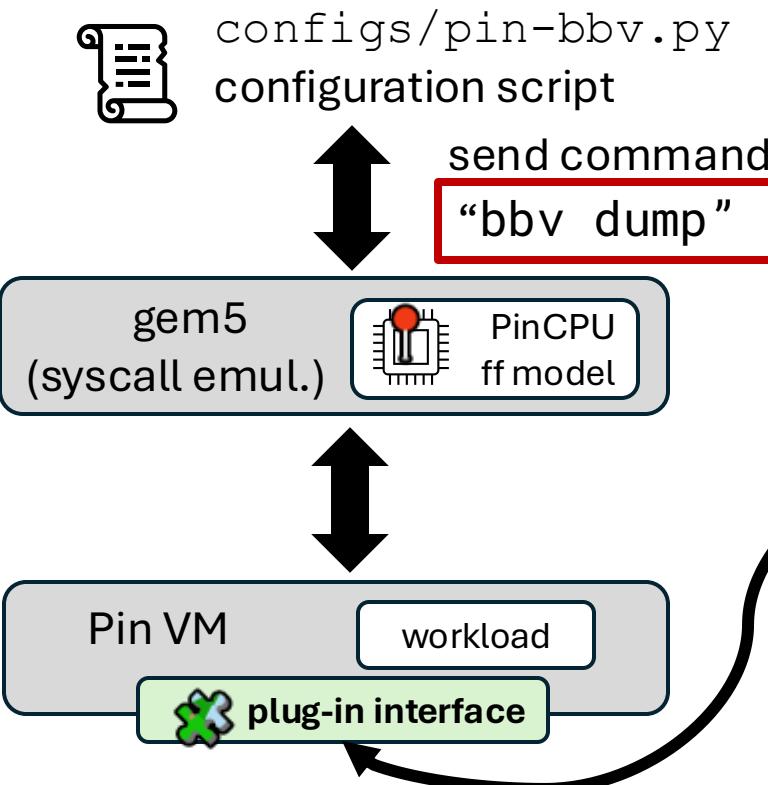
- Official: AtomicSimpleCPU SimPoint probe ! **slow**
- Unofficial: valgrind's BBV profiler ! **unreliable** (due to different HW capabilities, address space layout, syscall behavior, ...) → divergent execution paths → BBV misalignment



Solution: develop a PinCPU plugin

BBV Plugin for fast + accurate simpoint generation:

Plugin code (simplified)



BBV plugin
src/cpu/pin/tools/bbv.cc (124 LoC)

```
std::map<ADDRINT, uint64_t> bbv;  
  
void analyze(uint64_t &count) { ++count; }  
  
void instrument(BBL bbl) {  
    BBL_InsertCall(bbl, analyze, &bbv[BBL_Address(bbl)]);  
}  
  
struct BBVPlugin final : PinCPUPlugin {  
    void reg() override {  
        BBL_AddInstrumentFunction(instrument);  
    }  
    " {1:42, 2:2, 10:  
     160, 14:220, ... }" override {  
    }  
};  
} plugin;
```

call Analyze() whenever a basic block is executed

register instrumentation callbacks using Intel Pin API

BBV Plugin for fast + accurate simpoint generation: *Usage (simplified)*

```
./build/X86/gem5.opt configs/pin-bbv.py --bbv=bbv.txt --interval=50_000_000 -- ./exe
```



BBV file
bbv.txt

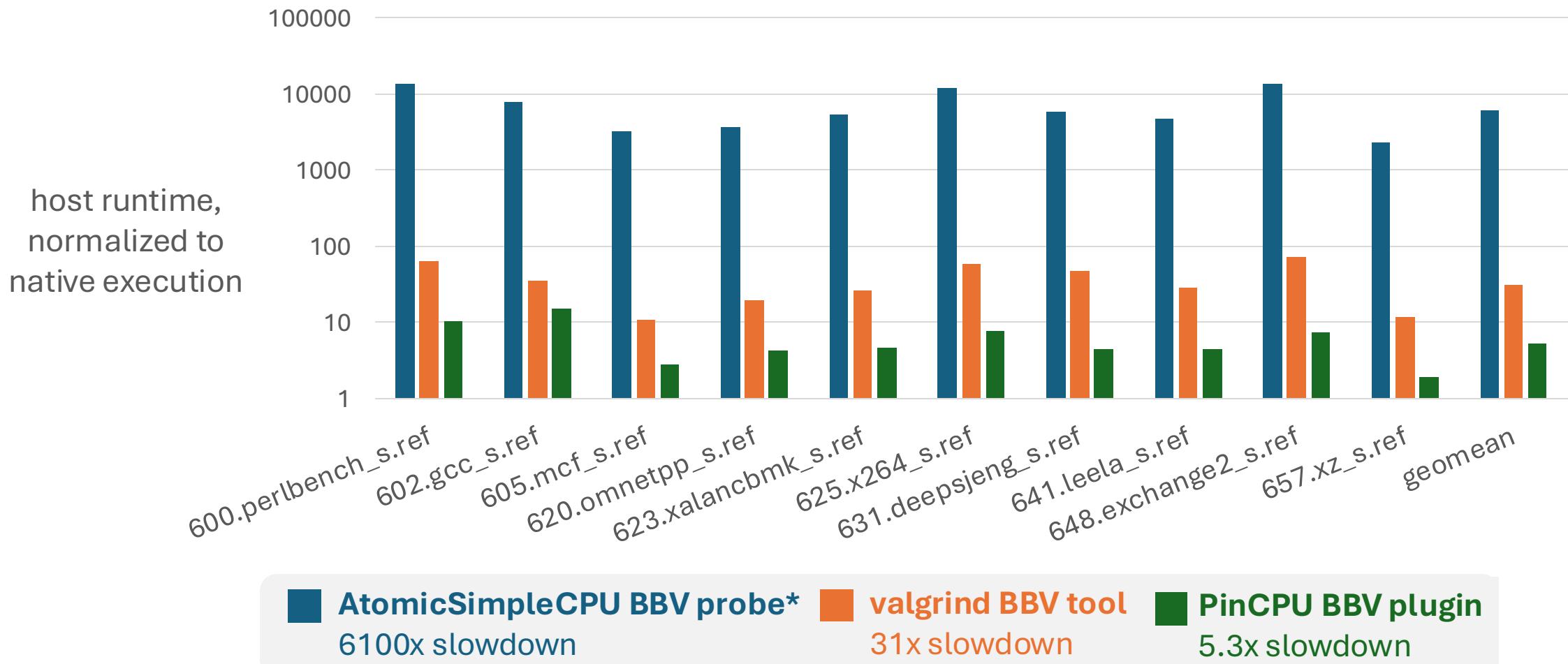


```
./simpoint –loadFVFile bvv.txt –saveSimpoints simpoints.txt
```



simpoint list
simpoints.txt

BBV Plugin slowdown vs. AtomicSimpleCPU BBV probe and valgrind BBV profile (SPEC CPU2017 intspeed ref)

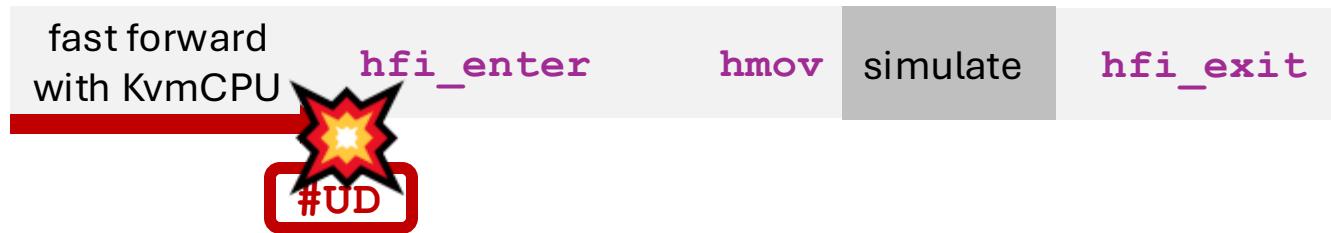


* estimated (extrapolated from prefix of first benchmark input)

Recall Limitation 2: can't fast-forward past new instructions

Hardware Fault Isolation (HFI) [Narayan+ ASPLOS'23] introduces new sandbox instructions:

- **hfi_enter**: enter sandbox
- **hfi_exit**: exit sandbox
- **hmov**: sandboxed load/store



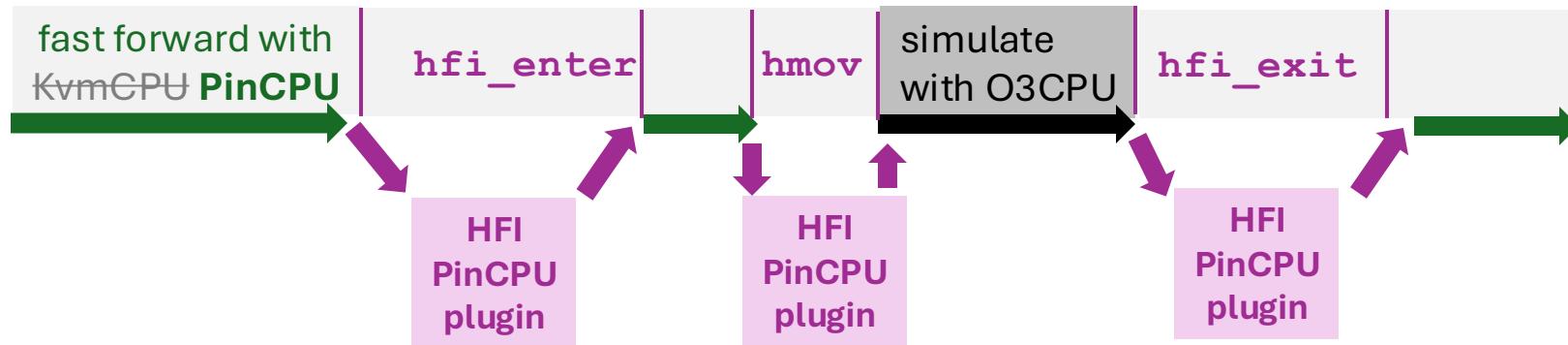
- Workaround: write standalone emulator to roughly approximate performance on host [Narayan+ ASPLOS'23]
- Workaround: trap+emulate with KvmCPU

⚠️ **laborious, inaccurate**

⚠️ **slow**

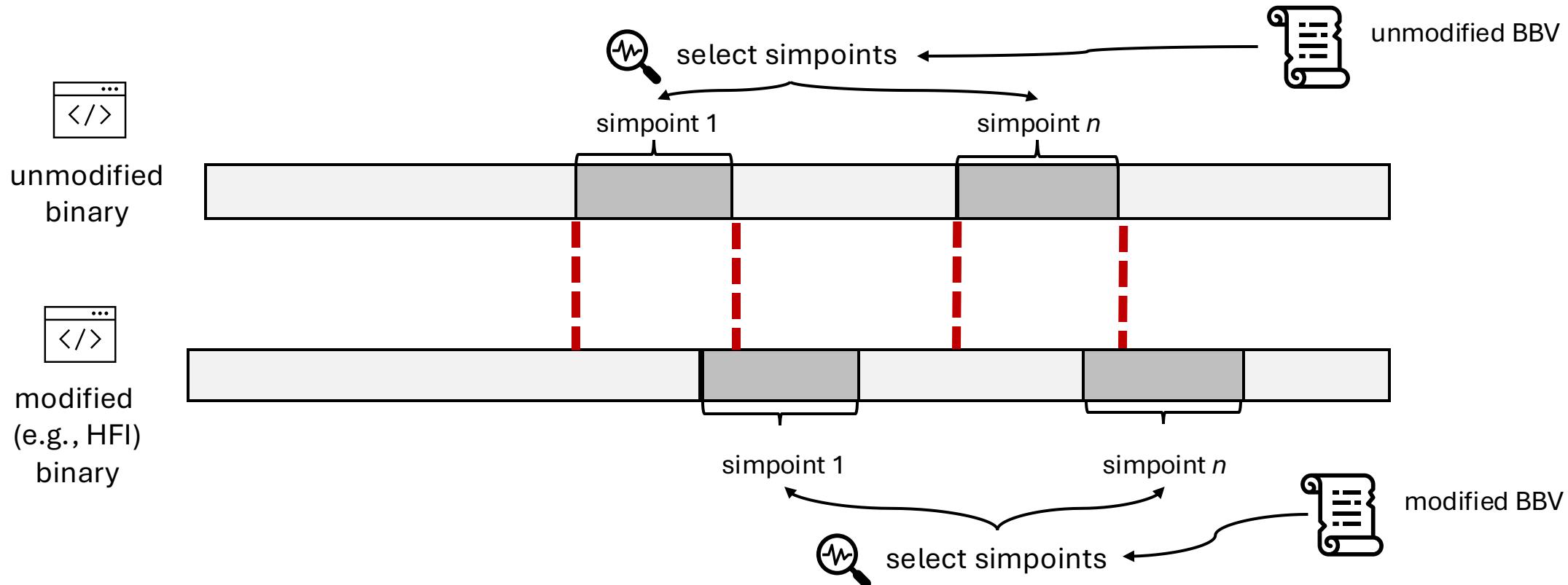
HFI Plugin for ISA extension emulation

Our HFI plugin for PinCPU hooks onto `hfi_enter`, `hmov`, `hfi_exit` and emulates them, rather than executing them directly.



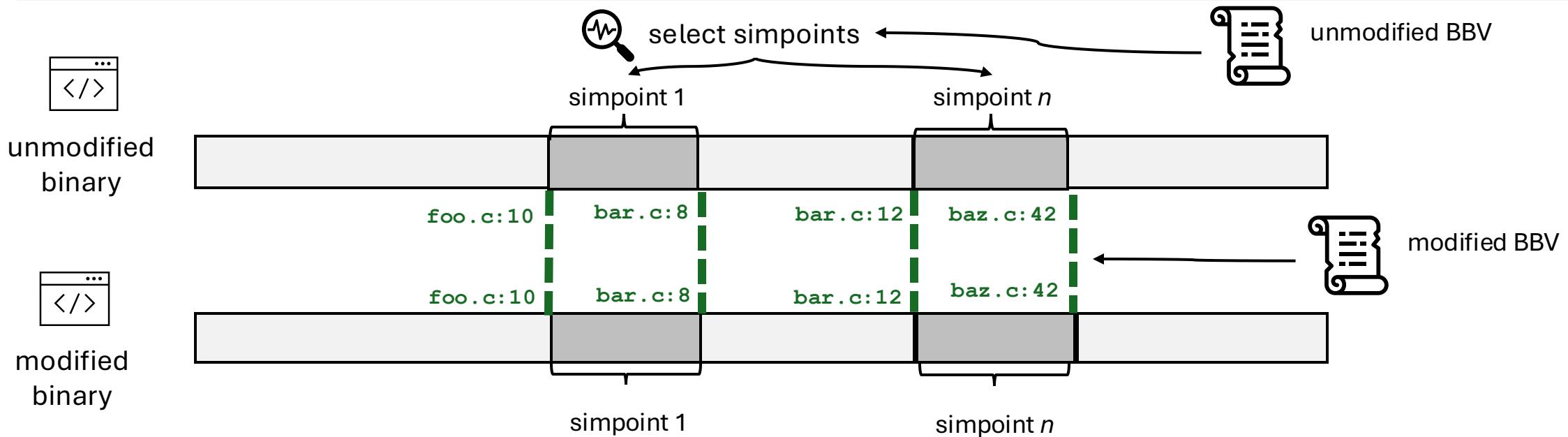
Recall Limitation 3: incongruent simpoints

- **Problem:** evaluating unmodified baseline and HW-SW proposal on different regions of the workload.
- **Solution: SimPoint Translation plugin** for PinCPU



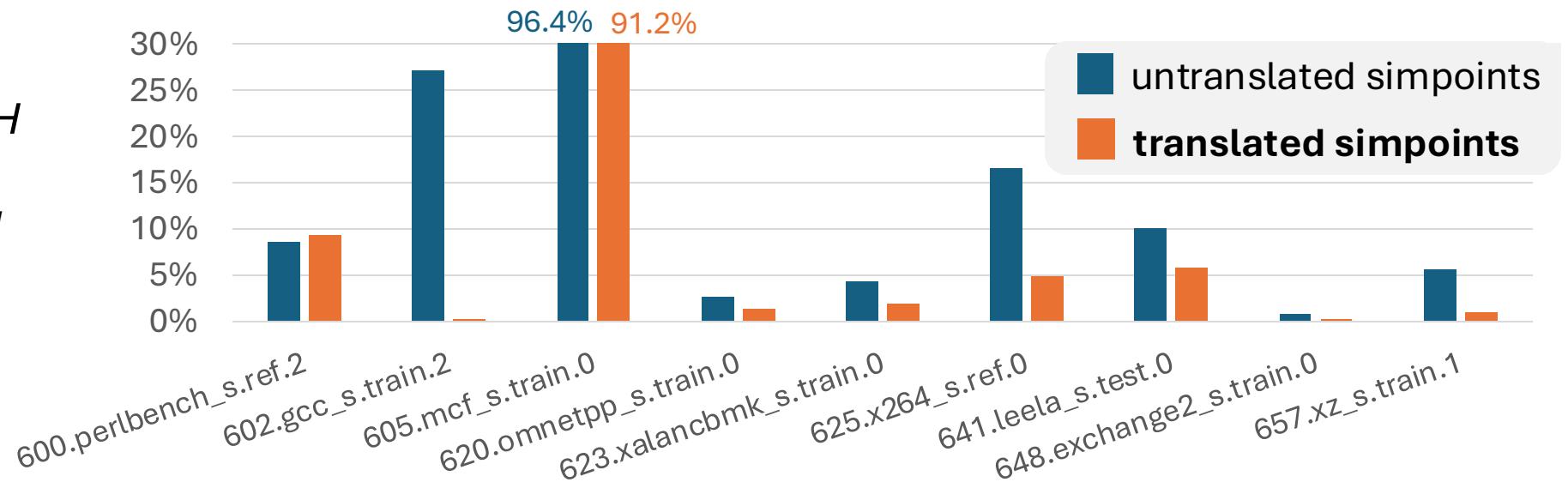
SimPoint Translation Plugin translates unmodified binary simpoints to modified binary

- Use stable source locations as an indicator of forward progress
 - **stable** = executes same number of times in both binaries
 - stable source locations are collected using BBV plugin + offline analysis



SimPoint Translation Plugin: error of estimated SLH overhead on gem5 O3CPU for SPEC CPU2017 intspeed

*error in estimated SLH
runtime overhead
(ground truth: end-to-end
execution in O3CPU)*



Speculative load hardening (SLH)
[Carruth LLVM'18] is a software-only
Spectre defense that involves
heavy program transformation.

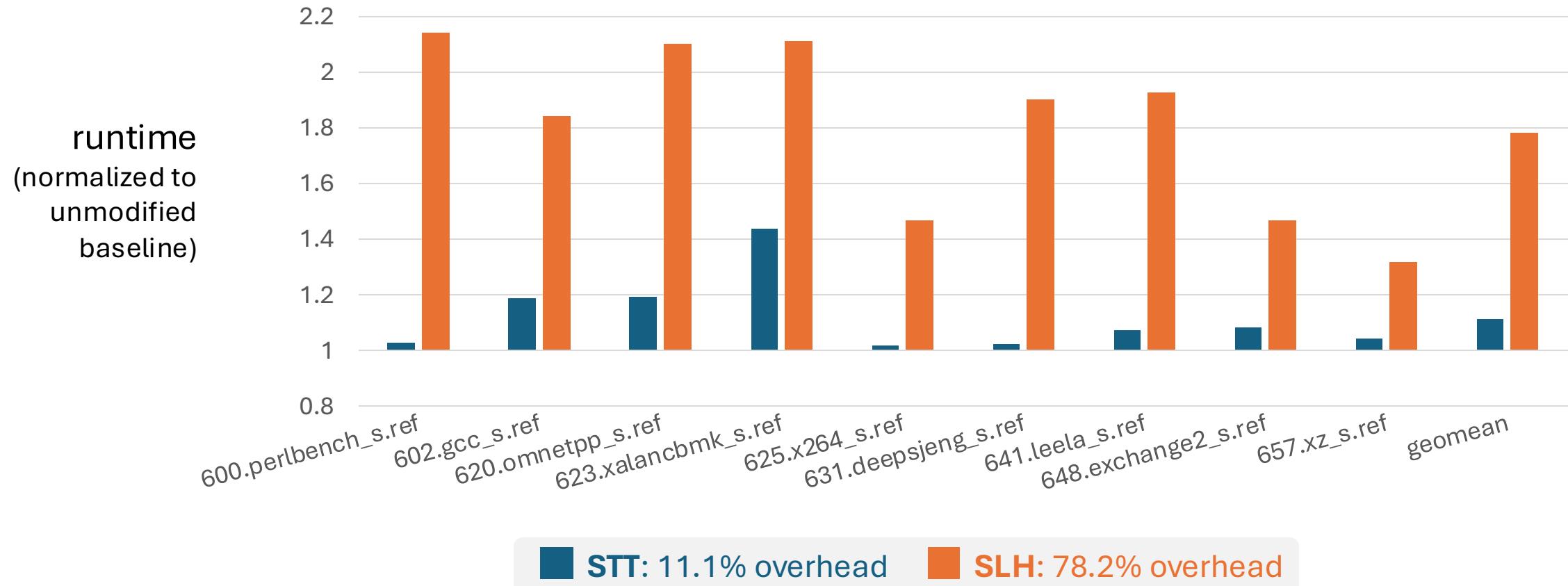
**Simpoint translation reduces
error by >50% on average!**

Putting it all together: PinCPU + our 3 plugins enable fast+accurate hardware-software co-evaluation in gem5

- **Hardware-software co-evaluation:** comparison of two proposals, at least one of which modifies {software, hardware}
- Co-evaluation #1: Spectre defenses
 - Hardware: **STT** [Yu+ MICRO'19]
 - Software: **SLH** [Carruth LLVM'18]
- Co-evaluation #2: Sandboxing proposals
 - Hardware-software: **HFI** [Narayan+ ASPLOS'23]
 - Software: **Segue** [Narayan+ ASPLOS'25]
- **These proposals have never been compared before**

Co-evaluation #1: Spectre defenses

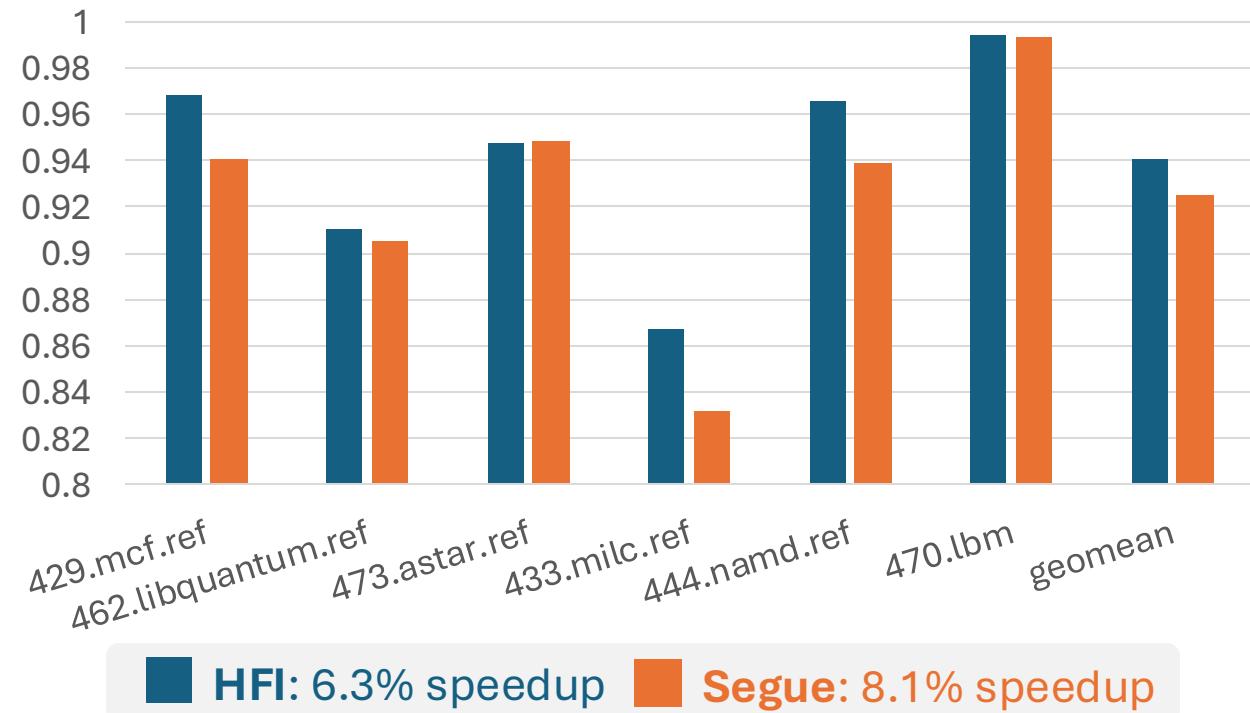
Runtime on SPEC CPU2017 intspeed, estimated using our PinCPU BBV profiler + PinCPU-enabled SimPoint translation methodology



STT outperforms SLH while offering stronger security guarantees → hardware-supported Spectre defenses well worth the performance-complexity trade-off.

Co-evaluation #2: Sandboxing proposals

Runtime on SPEC CPU2006, estimated using our PinCPU BBV profiler
+ HFI plugin + PinCPU-enabled SimPoint translation methodology



Segue (software-only) **outperforms HFI** (hardware-software)
→ HFI not worth the performance/complexity trade-off?

* only WebAssembly-compatible benchmarks evaluated

Other Features & Limitations

- Features:
 - PinCPU supports **checkpointing**
 - **Semantic breakpoints**: “stop after instruction 0x1234 executes 100 times”
- Limitations (as of this presentation):
 - Only supports syscall emulation (SE) mode, but not gem5 full system (FS) mode
 - Doesn’t support multithreaded workloads
 - Only supports x86-64 ISA
- Future work:
 - QemuCPU for full system instrumentation
 - Add multi-threading support
 - Add other DBI backends (e.g., DynamoRIO) to support more ISAs

Conclusion

- PinCPU is the **first** gem5 CPU model to support both **fast-forwarding** and **dynamic binary instrumentation**.
- PinCPU plugins allow gem5 users to quickly profile workloads, emulate new instructions, and develop new evaluation methodologies.
- Our three example PinCPU plugins make it possible to perform **fast and accurate hardware-software co-evaluations** in gem5.

Email: nmosier@stanford.edu

GitHub: github.com/StanfordPLArchSec/pincpu-gem5