THE CHIPS TO SYSTEMS CONFERENCE
SHAPING THE NEXT GENERATION OF ELECTRONICS
JUNE 23-27, 2024
MOSCONE WEST CENTER
SAN FRANCISCO, CA, USA

ChinaSys

PathFuzz: Broadening Fuzzing Horizons with Footprint Memory for CPUs

Yinan Xu, Sa Wang, Dan Tang, Ninghui Sun, Yungang Bao

[1]Institute of Computing Technology, Chinese Academy of Sciences (ICT, CAS)

[2]University of Chinese Academy of Sciences (UCAS)

[3]Beijing Institute of Open Source Chip (BOSC)

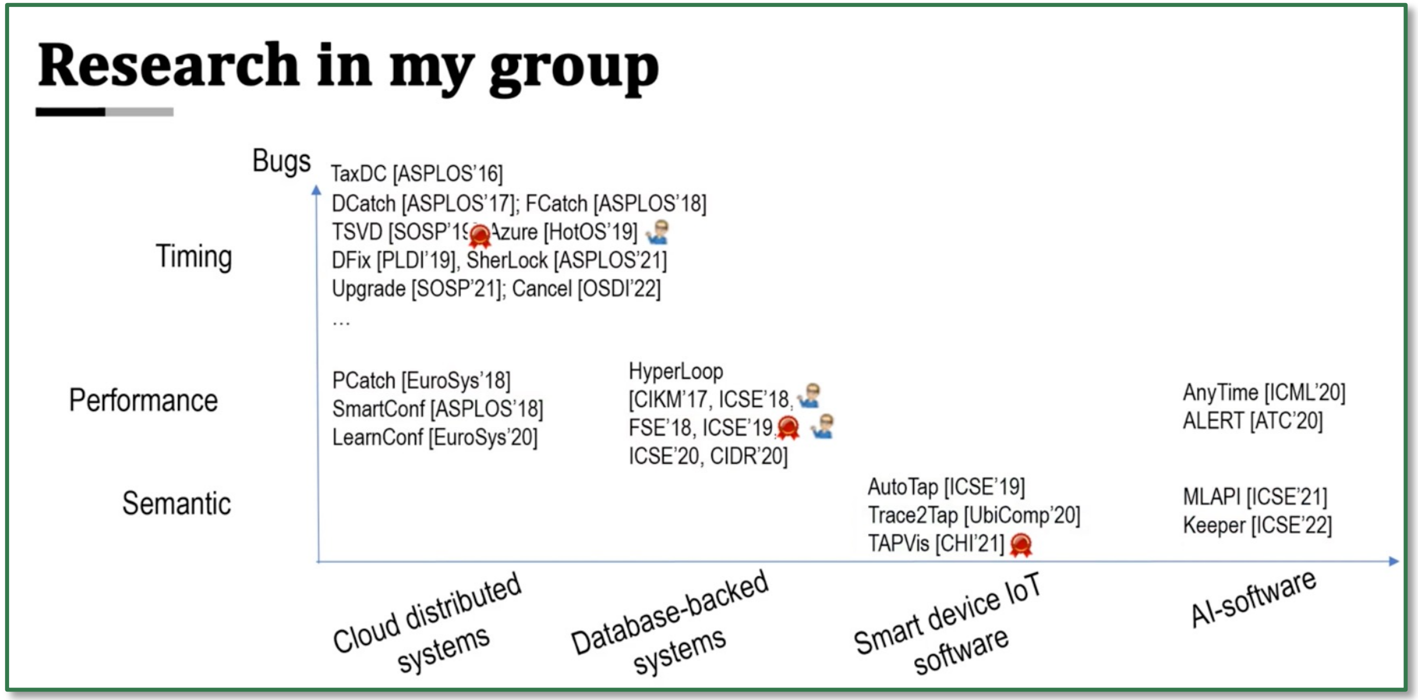
中国科学院计算技术研究所
INSTITUTE OF COMPUTING TECHNOLOGY ,CHINESE ACADEMY OF SCIENCES

BOSC

Open-Source Software and Software Testing

- Software testing research *greatly benefits from* open-source software
    - And the vice versa!

- Linux
- FreeBSD
- MySQL
- PostgreSQL
- Apache
- Mozilla
- OpenOffice
- ……

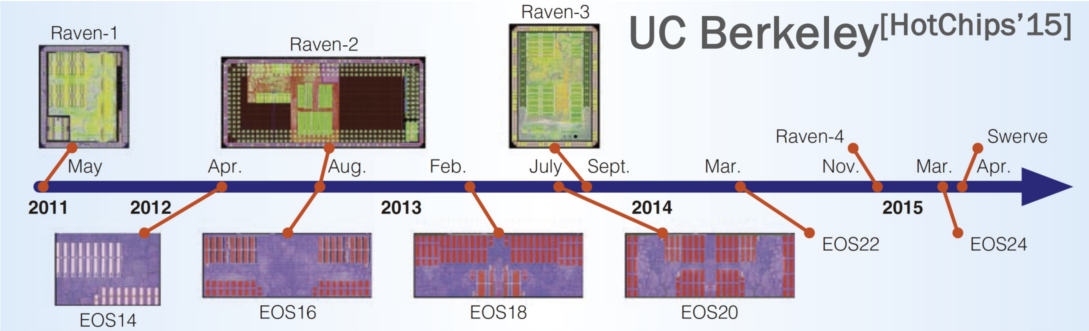


Research in my group

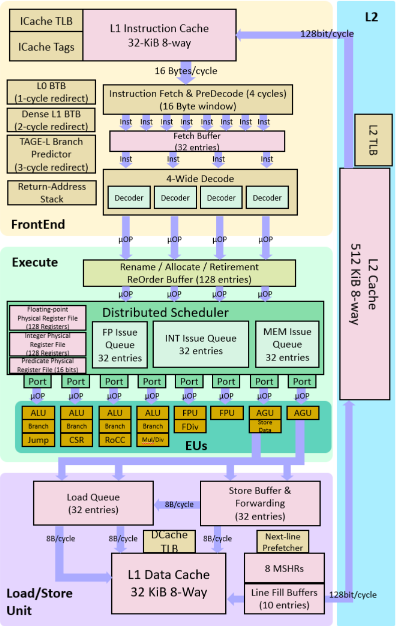
Bugs	Cloud distributed systems	Database-backed systems	Smart device IoT software	AI-software
Timing	TaxDC [ASPLOS'16] DCatch [ASPLOS'17]; FCatch [ASPLOS'18] TSVD [SOSP'19], Azure [HotOS'19] DFix [PLDI'19], SherLock [ASPLOS'21] Upgrade [SOSP'21]; Cancel [OSDI'22] …			
Performance	PCatch [EuroSys'18] SmartConf [ASPLOS'18] LearnConf [EuroSys'20]	HyperLoop [CIKM'17, ICSE'18, FSE'18, ICSE'19, ICSE'20, CIDR'20]		AnyTime [ICML'20] ALERT [ATC'20]
Semantic			AutoTap [ICSE'19] Trace2Tap [UbiComp'20] TAPVis [CHI'21]	MLAPI [ICSE'21] Keeper [ICSE'22]

Source: Shan Lu, 15 Years of Learning from Mistakes in Building System Software, 22nd ChinaSys Workshop.

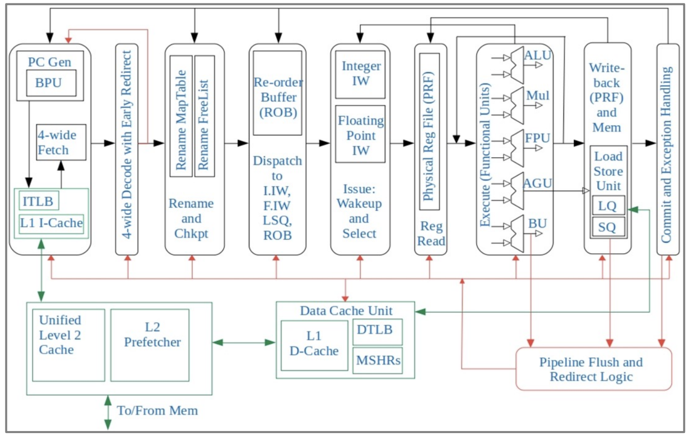
The Era of Open-Source Chip



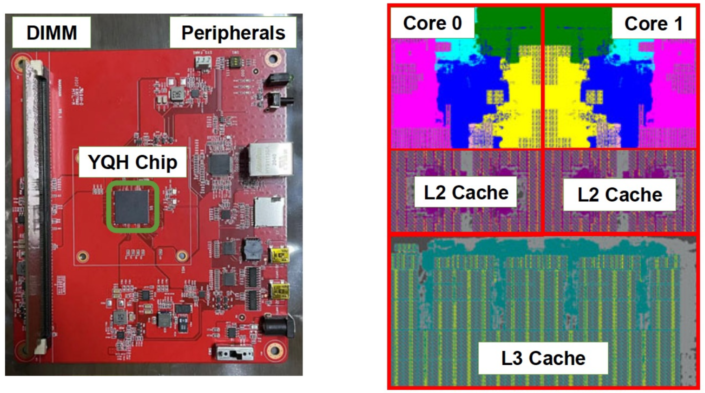
UC Berkeley[HotChips'15]



SonicBOOM[CARRV'20]



Shakti



XiangShan[MICRO'22]

Beihai[Intelligent Computing]

Hardware Design Verification (DV) is Challenging

≈ (Software) Testing + Verification

50%

Increase in ***design*** engineers since 2007

146%

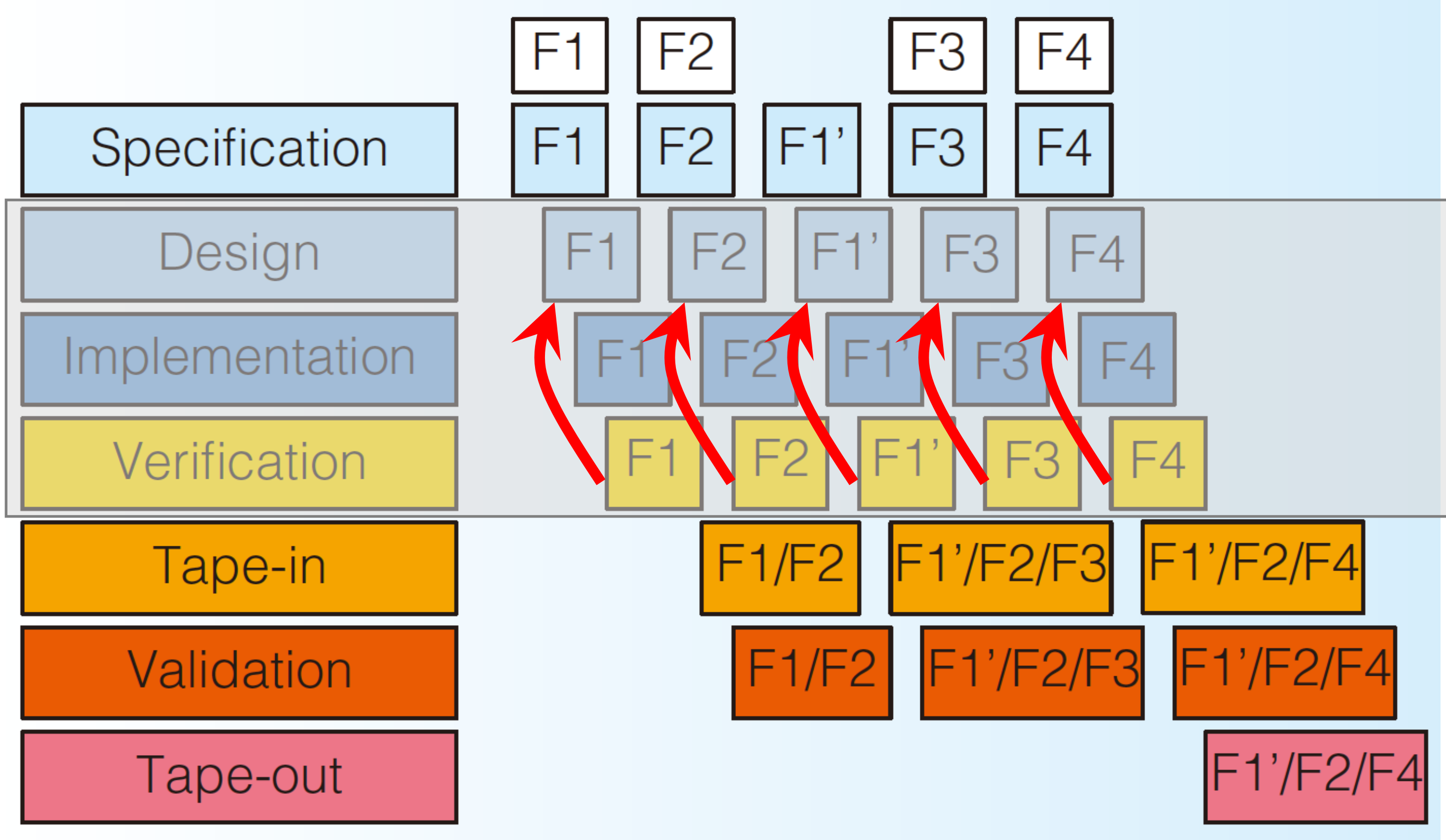
Increase in ***verification*** engineers since 2007

>50%

Median project time spent in ***verification***

Source: https://blogs.sw.siemens.com/verificationhorizons/2022/12/12/part-8-the-2022-wilson-research-group-functional-verification-study/

The Lockstep Between Design and Verification

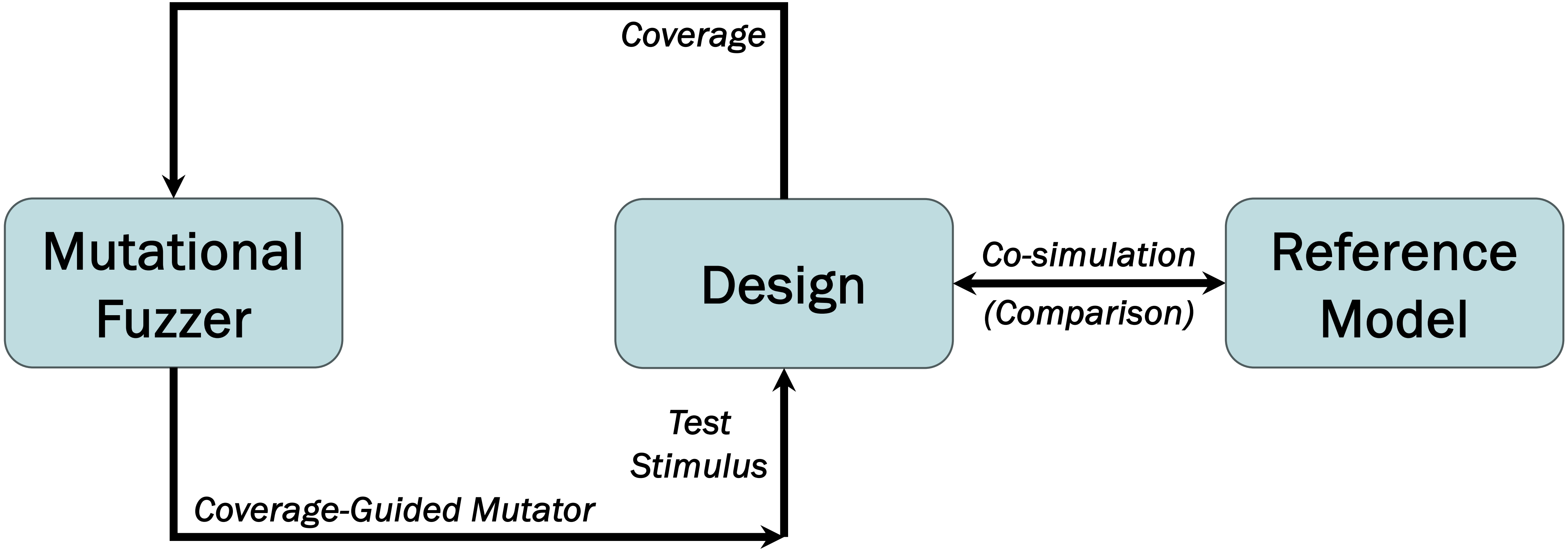

F1 F2 F3 F4
Specification F1 F2 F1’ F3 F4
Design F1 F2 F1’ F3 F4
Implementation F1 F2 F1’ F3 F4
Verification F1 F2 F1’ F3 F4
Tape-in F1/F2 F1’/F2/F3 F1’/F2/F4
Validation F1/F2 F1’/F2/F3 F1’/F2/F4
Tape-out F1’/F2/F4

- The design is changing.
- Verification verifies the design.
- Verification must keep up!

Figure. The agile model of hardware design

Source: Yunsup Lee *et al.*, "An Agile Approach to Building RISC-V Microprocessors," in *IEEE Micro*, vol. 36, no. 2, pp. 8-20, Mar.-Apr. 2016.

Fuzzing: Automated Design-Directed Verification

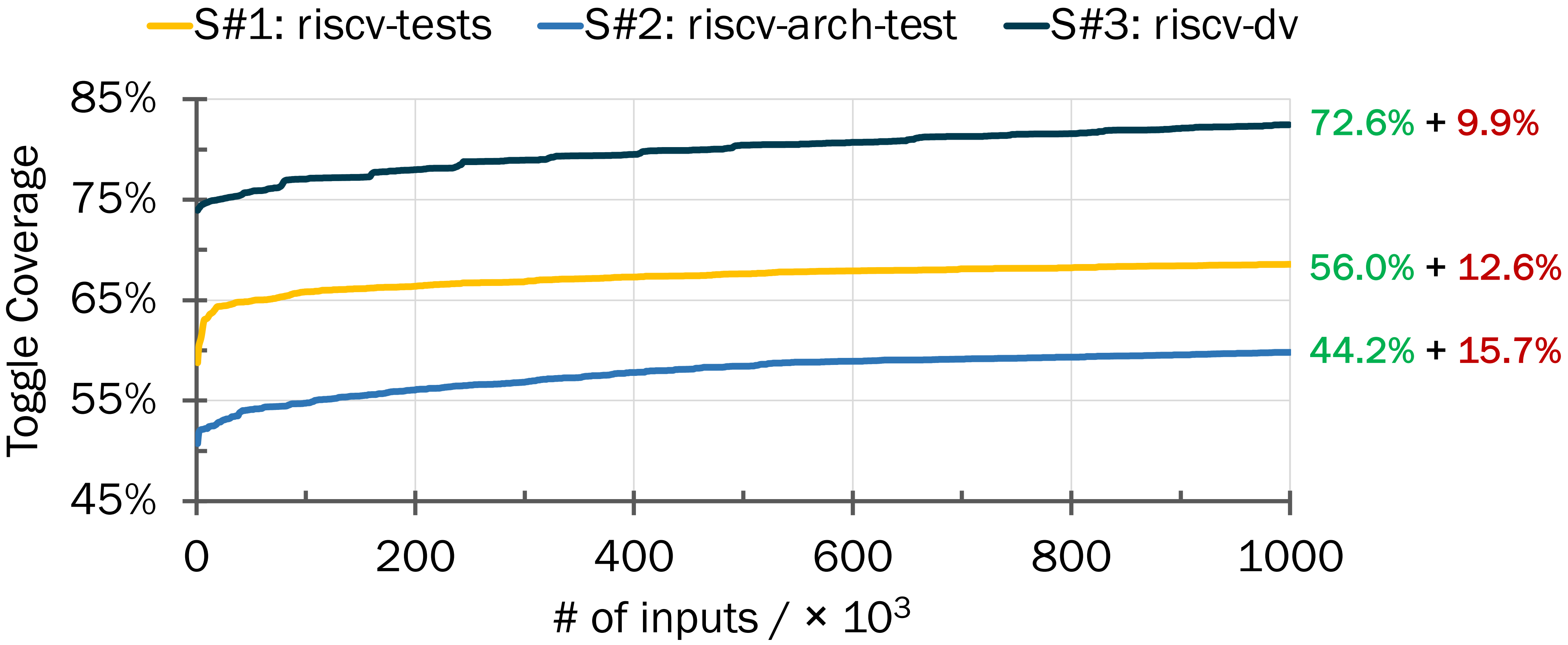


Fuzzing: Hardware vs. Software

- Software fuzzing has been widely accepted and adopted
  - Highly automated and efficient; significant return on investment (ROI)
  - [AFLplusplus/AFLplusplus] 4 steps: instrumenting, preparing, fuzzing, managing
  - [google/oss-fuzz] As of August 2023, OSS-Fuzz has helped identify and fix over 10,000 vulnerabilities and 36,000 bugs across 1,000 projects.

- Hardware fuzzing is *more challenging* due to various issues
  - Sophisticated binary-level input/output semantic
  - High design complexity; similar to highly concurrent programs
  - Low simulation speed/throughput
  - Lack of open-source practice (designs, corpus, crash detection, ...)

Fuzzing CPUs: Coverage Increase



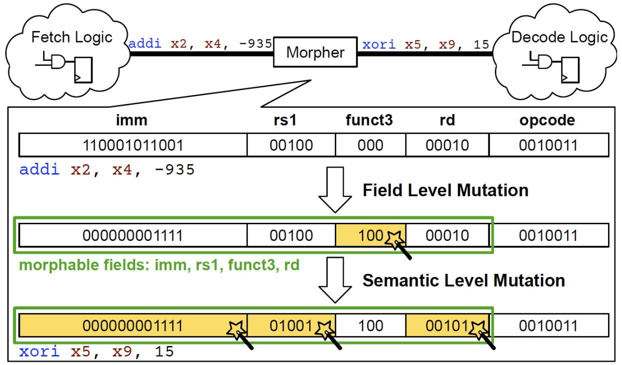
Fact: the fuzzer achieves limited coverage increase from the start points

Note: data collected by the LibAFL fuzzer with havoc mutator and toggle coverage feedback from rocket-chip.

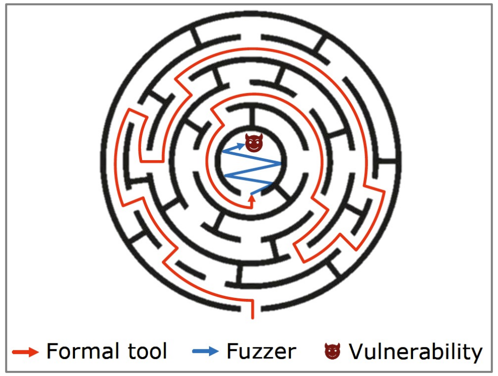
Insight[1]: Exploitation and Exploration

- Literally, mutational fuzzers are *very good at exploitation*
  - Mutators generally create a large number of input cases
- However, *the exploration is inefficient* when applying fuzzing to CPUs
  - Reason: instruction set architectures (ISAs) are complicated and sophisticated
  - Recent works improve it with domain-specific knowledge or formal methods

[1] MorFuzz proposes RISC-V specific instruction mutations



imm	rs1	funct3	rd	opcode
110001011001	00100	000	00010	0010011
addi x2, x4, -935				
000000001111	00100	100	00010	0010011
000000001111	01001	100	00101	0010011
xori x5, x9, 15				

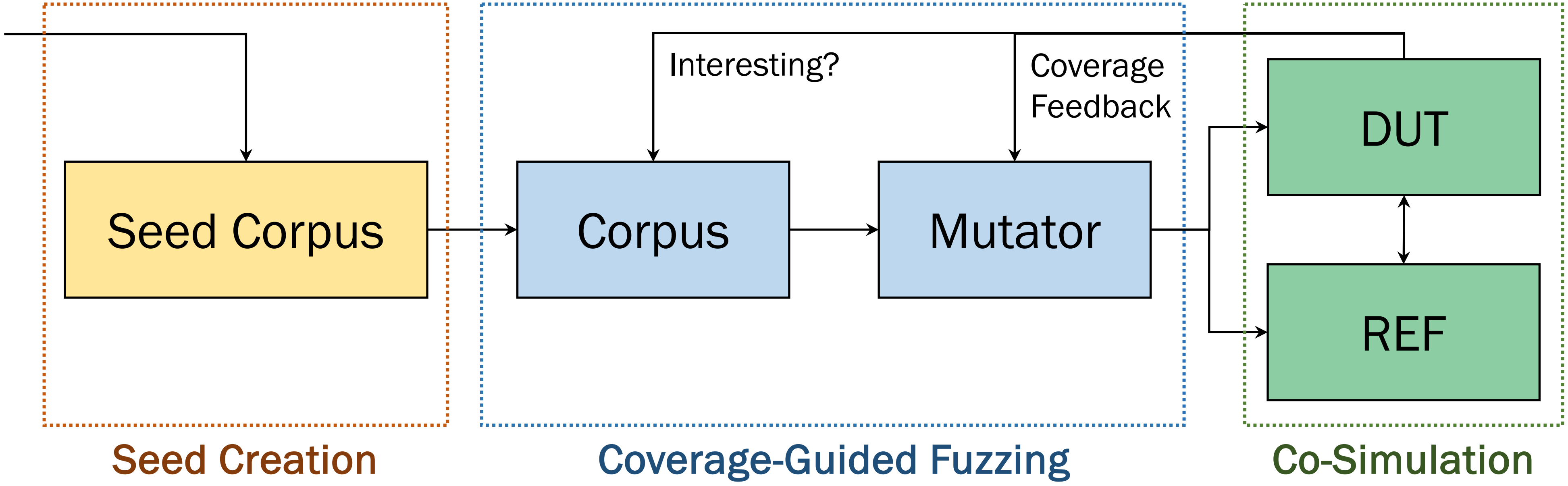


[2] HyPFuzz proposes formal-assisted state exploration

[1] Jinyan Xu, Yiyuan Liu, Sirui He, Haoran Lin, Yajin Zhou, and Cong Wang. 2023. MorFuzz: fuzzing processor via runtime instruction morphing enhanced synchronizable co-simulation. In *Proceedings of the 32nd USENIX Conference on Security Symposium (SEC '23)*. USENIX Association, USA, Article 74, 1307–1324.

[2] Chen Chen, Rahul Kande, Nathan Nguyen, Flemming Andersen, Aakash Tyagi, Ahmad-Reza Sadeghi, and Jeyavijayan Rajendran. 2023. HyPFuzz: formal-assisted processor fuzzing. In *Proceedings of the 32nd USENIX Conference on Security Symposium (SEC '23)*. USENIX Association, USA, Article 77, 1361–1378.

Fuzzing CPUs: SOTAs have done good jobs



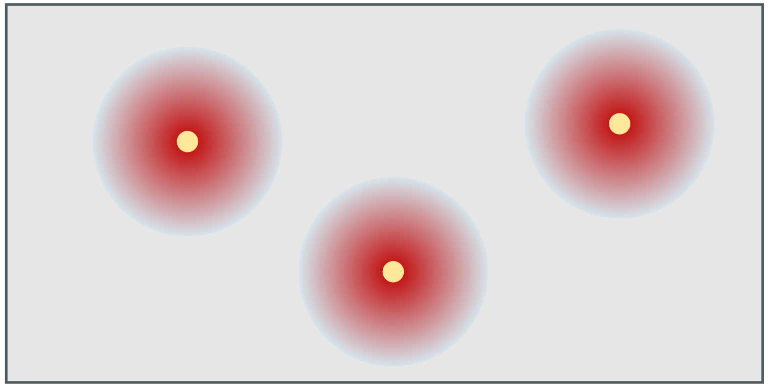
Coverage-Guided Fuzzing

- [SP’21] DifuzzRTL
- [USENIX Security’22] TheHuzz
- [GLSVLSI’22] CFG for Processor
- [USENIX Security’23] HyPFuzz
- [USENIX Security’23] MorFuzz
- [DATE’23] SoCFuzzer

Co-Simulation

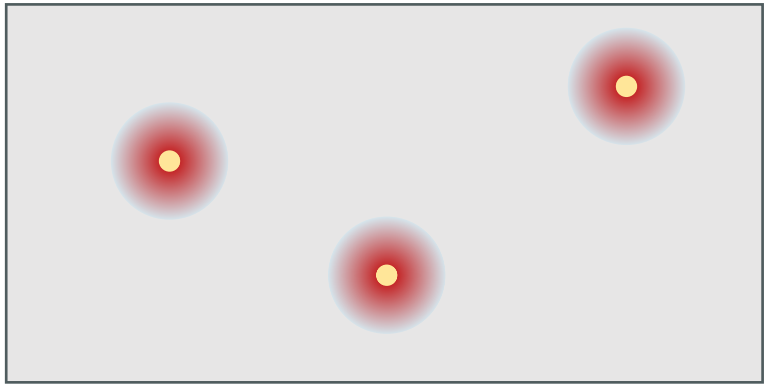
- [MICRO’21] Dromajo
- [MICRO’22] DiffTest

Insight^2: Ways to Exploring the State Space

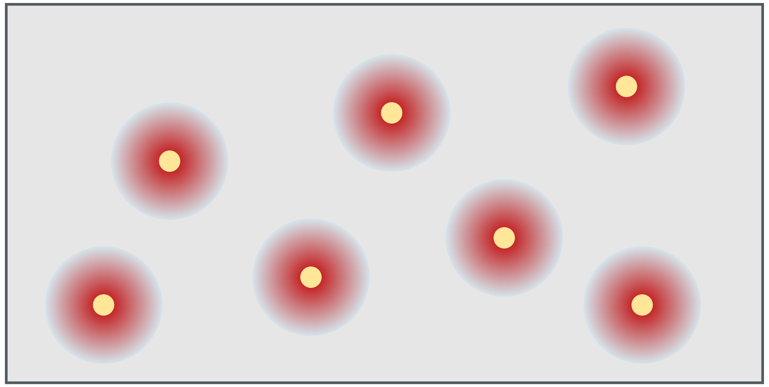


Wider Search Scope

Better mutations



Fuzzing Basics



More Start Points

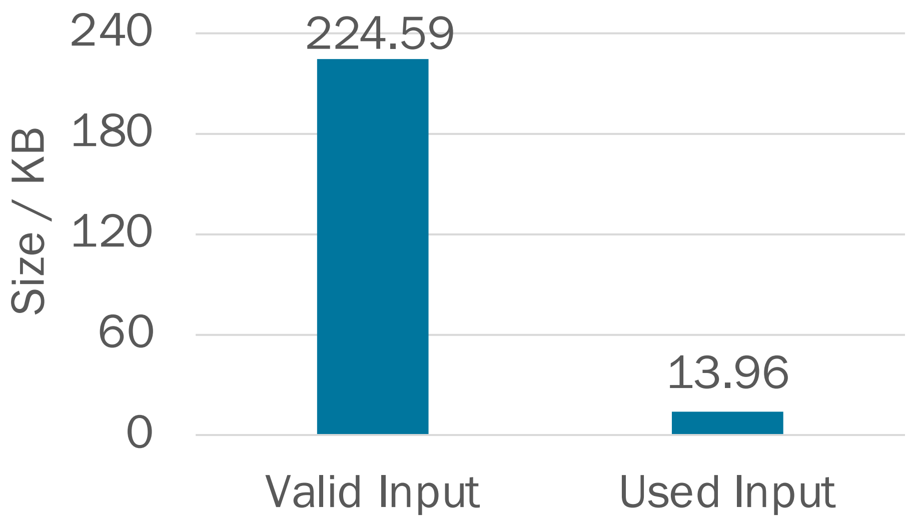
Richer seeds

- ● Start point
- ● Search range

Observations: Fuzzing Horizons are Constrained

(1) *Mutations* are not that effective

Given the seeds S#3 (riscv-dv):



Effective only if targeting 6.2% of input bytes

Further decreased to 2.5% after 1M mutations

(2) *Sources of seeds* are limited

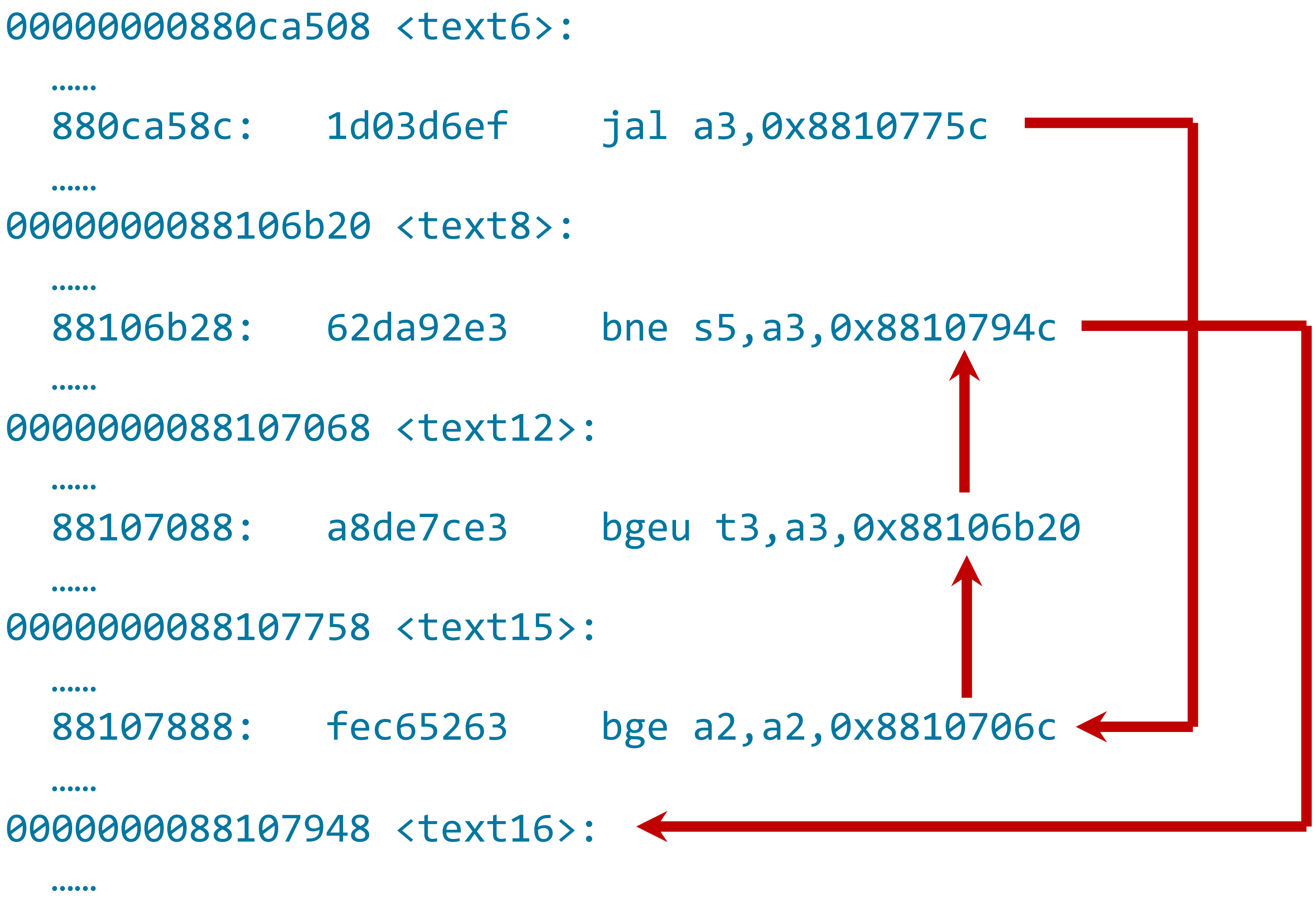
Given the seeds S#4 (force-riscv):

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

OOM crashes for in-memory fuzzers

Significant slow down fuzzers with corpus on disk

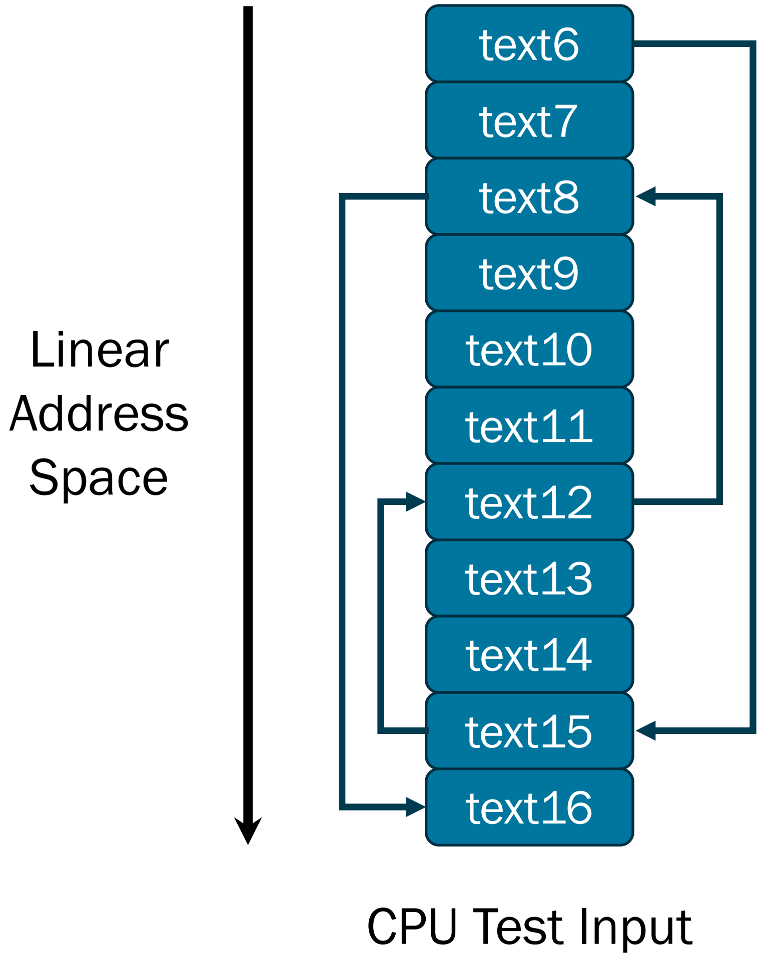
Why: How CPU Fetches and Executes



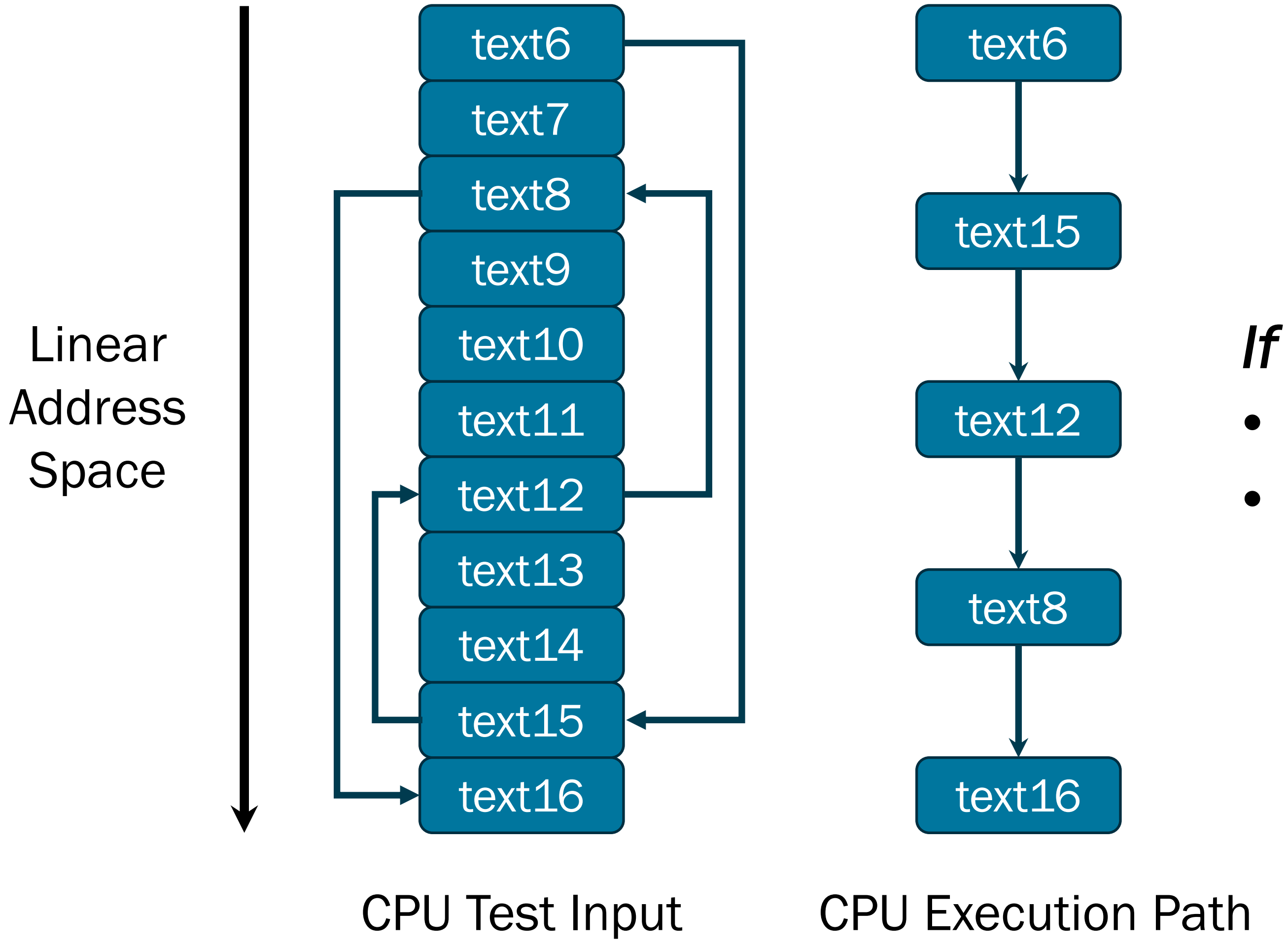
```
00000000880ca508 <text6>:
  ……
  880ca58c:   1d03d6ef    jal a3,0x8810775c
  ……
0000000088106b20 <text8>:
  ……
  88106b28:   62da92e3    bne s5,a3,0x8810794c
  ……
0000000088107068 <text12>:
  ……
  88107088:   a8de7ce3    bgeu t3,a3,0x88106b20
  ……
0000000088107758 <text15>:
  ……
  88107888:   fec65263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

This is a case from the seeds S#4 (force-riscv)

Input Format: The Linear Memory



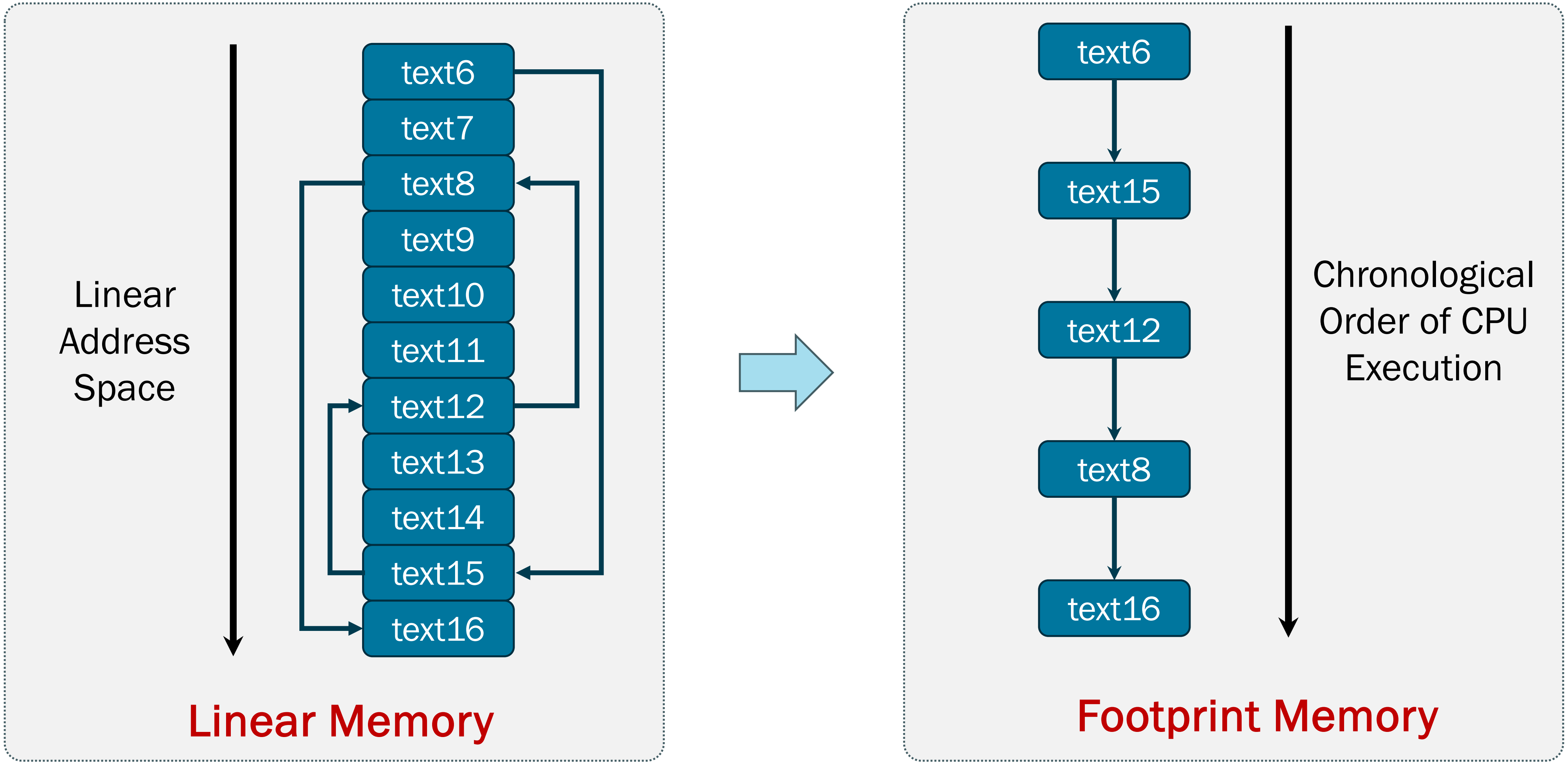
Insight^3: Linear Memory Hides Execution Paths



If removing untouched memory contents …

- *Mutations become more effective*
- *Seeds' size is significantly reduced*

Footprint Memory: Capturing Execution Paths



Linear Address Space

text6
text7
text8
text9
text10
text11
text12
text13
text14
text15
text16

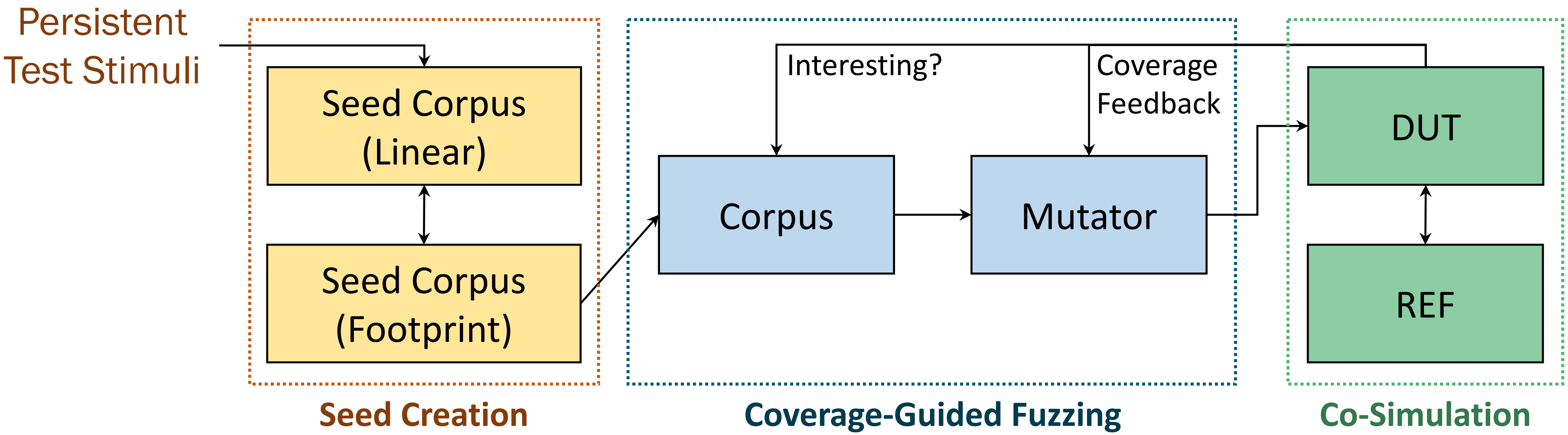
Linear Memory

text6
text15
text12
text8
text16

Chronological Order of CPU Execution

Footprint Memory

PathFuzz: Overview of the Workflow



Refer to our Paper Section 3.2 for more details in enhancing/adapting the three stages.

PathFuzz: Broadening Sources of Seed Corpus

- Modern CPU DV reaches a good coverage; let fuzzers take a step further

- The test cases we are currently using for the system-level DV of CPUs

1) hand-written directed tests

- riscv-software-src/riscv-tests
- riscv-non-isa/riscv-arch-test
- riscv-ovpsim/imperas-riscv-tests
- litmus-tests/litmus-tests-riscv
- josecm/riscv-hyp-tests

2) instruction-stream generators

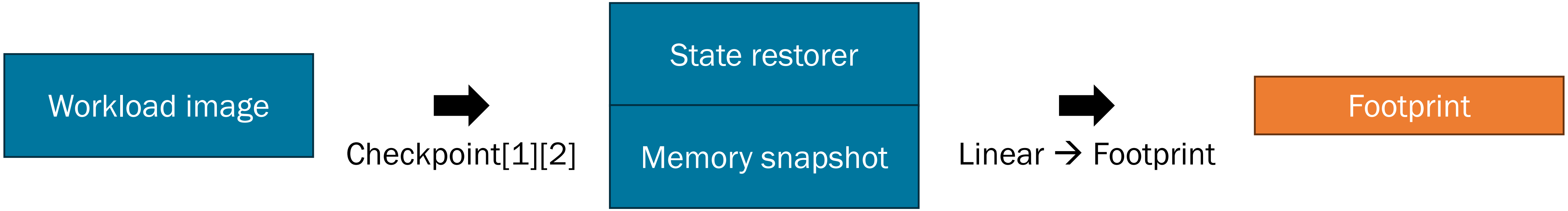
- chipsalliance/riscv-dv
- openhwgroup/force-riscv
- ksco/riscv-vector-tests
- sifive/riscv-vector-intrinsic-fuzzing
- chad-q/andes-vector-riscv-dv

3) real-world programs

- ucb-bar/riscv-benchmarks
- eembc/coremark
- SPEC CPU® 2017
- SPECjbb® 2015
- gcc,clang,rustc,verilator

PathFuzz: Enhancing DV with Practical Fuzzing

- Incorporating existing, valuable CPU test cases as seeds for fuzzing
  - Extracting the footprints when CPU executes these test cases
  - Using the (short-running) footprints as fuzzing seeds
- Contribution: fuzzing with any start point at any program phase
- How: architectural checkpoints + footprint memory



[1] Nursultan Kabylkas, et al., 2021. Effective Processor Verification with Logic Fuzzer Enhanced Co-simulation. *MICRO'21.*

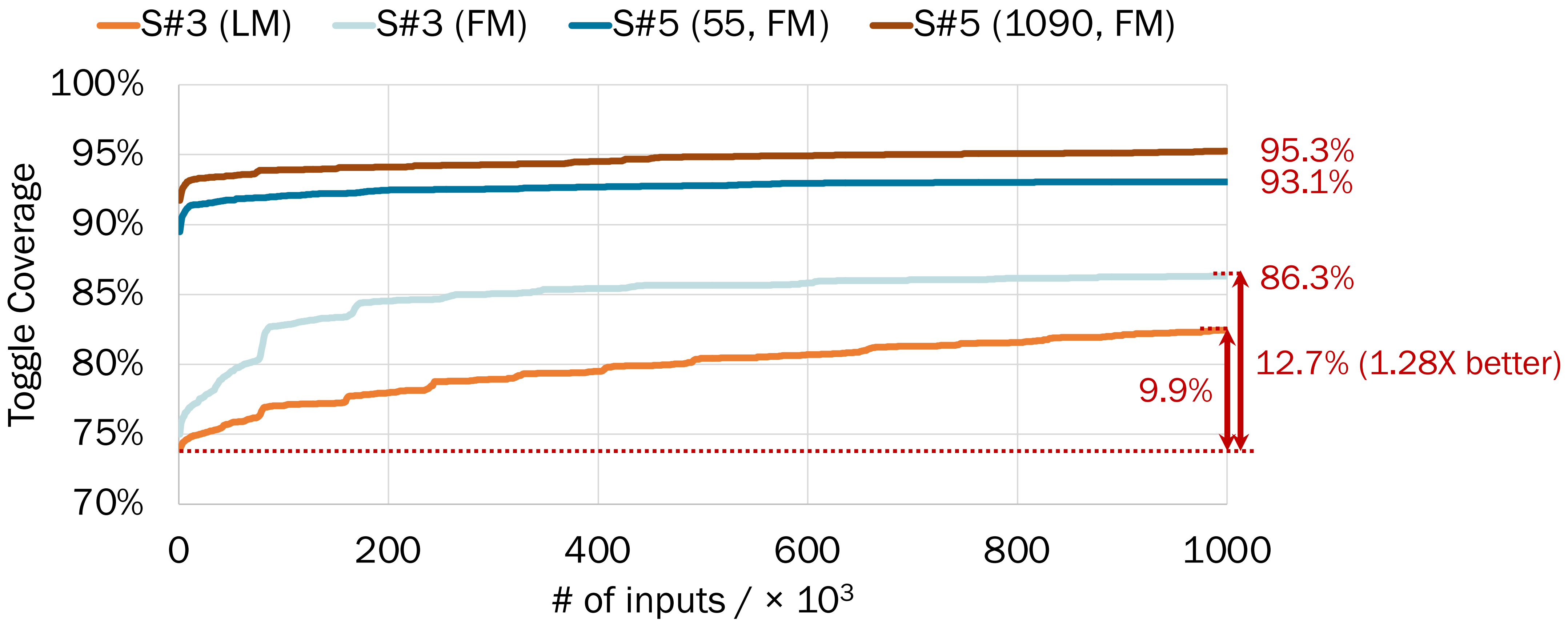
[2] Yinan Xu, *et al.*, 2023. Towards Developing High Performance RISC-V Processors Using Agile Methodology. *MICRO'22*.

Evaluation

- Setup: famous, widely-adopted, open-source projects
  - Fuzzer: LibAFL v0.10.1 (unmodified QueueScheduler, StdMapObserver, StdFuzzer)
  - CPU design under test: rocket-chip
  - CPU reference/golden model: Spike (riscv-isa-sim)
  - Various seeds, seed count (linear/footprint formats)
    - S#1: riscv-tests, 140 (LM, FM)
    - S#2: riscv-arch-test, 257 (LM, FM)
    - S#3: riscv-dv, 1150 (LM, FM)
    - S#4: force-riscv, 969 (FM)
    - S#5: SPEC CPU2006, 1090 (FM)

- To show the coverage increase, coverage reach, discovered bugs

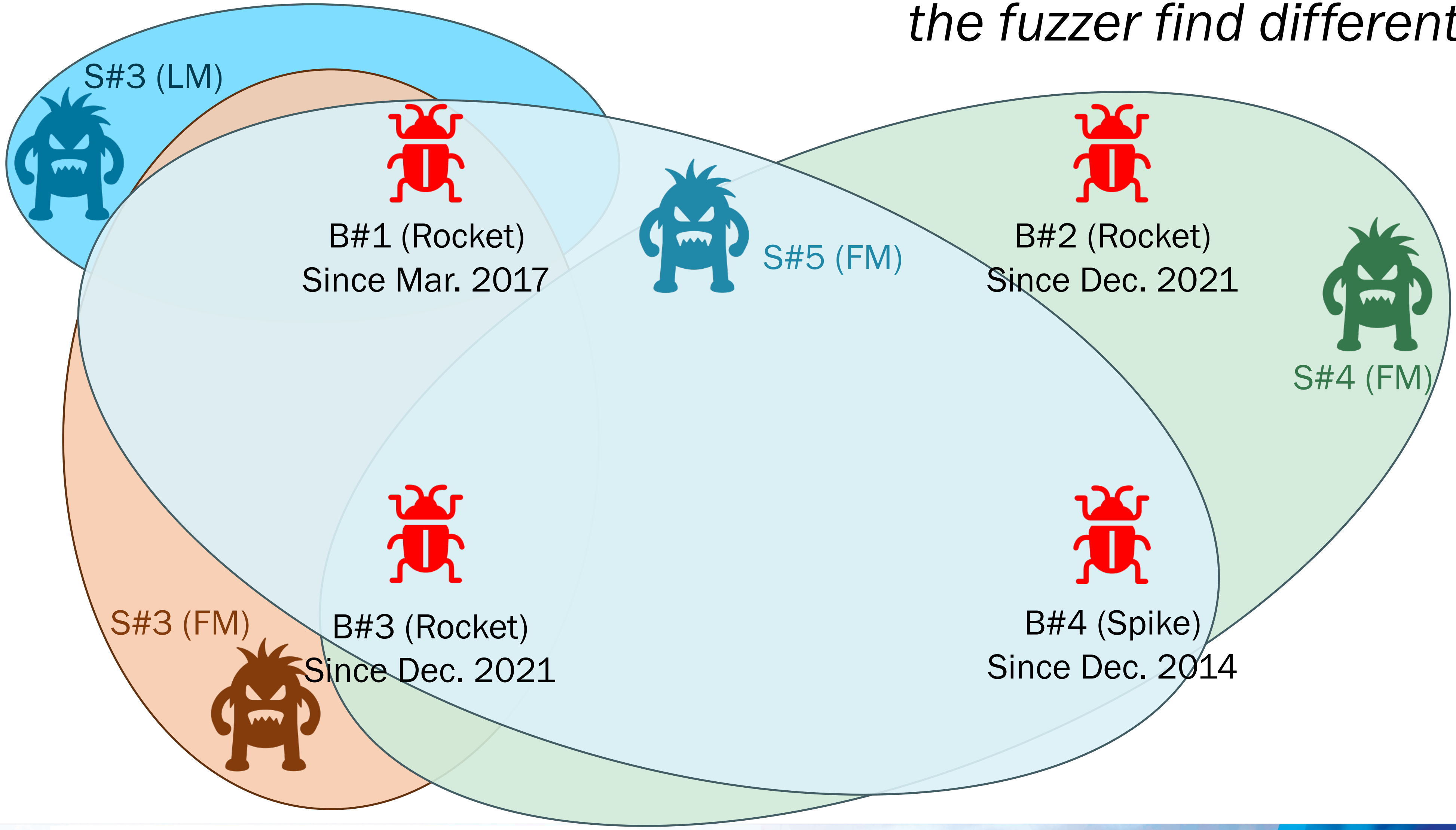
Evaluation: Coverage



* 95% criteria: formal-assisted HyPFuzz takes 72 hours to achieve 94.9% on CVA6; we take ~10 hours to achieve 95.3% on rocket-chip

Evaluation: Bugs

Insight^4: *Different seeds help the fuzzer find different bugs!*



Applying Fuzzing to Open-Source XiangShan

欢迎 ChinaSys 社区研究者多多关注开源硬件验证（测试）领域

Version	#Error / #All (seed corpus)	Potential Bug Count
20230905	*** / 50000	5
20230907	**** / 300000	6+ (***/**** analyzed)
20230915	**/1838 (riscv-tests, LM)	Not analyzed yet
	**/3772 (riscv-arch-test, LM)	
	***/2181 (riscv-dv, LM)	
	***/25087(riscv-tests, FM)	
	**/4132 (riscv-arch-test, FM)	
	***/2532 (riscv-dv, FM)	
	***/2196 (force-riscv, FM)	
	***/3751 (SPECCPU2006, FM)	

* Preliminary testing results on unstable versions of XiangShan; do not necessarily reflect the final design verification quality.

Conclusion

- **Motivation:** broadening the fuzzing horizons on CPUs
  - More effective mutations, richer seed sources for better exploration capabilities
- **PathFuzz:** a coverage-guided CPU fuzzing workflow
  - Input format: both linear and footprint memory
  - Incorporate large-scale programs as fuzzing seeds
- **Evaluation**
  - Achieve better coverage increase/reach
  - Detect 4 long-standing bugs in well-known projects
- **Open-sourced** at GitHub with open-source components*
  - Contribute to the reproducible, reusable research community

* https://github.com/OpenXiangShan/xfuzz. Thank LibAFL, rfuzz, DifuzzRTL, SIC, and DiffTest.

Conclusion; Questions?

- **Motivation:** broadening the fuzzing horizons on CPUs
  - More effective mutations, richer seed sources for better exploration capabilities
- **PathFuzz:** a coverage-guided CPU fuzzing workflow
  - Input format: both linear and footprint memory
  - Incorporate large-scale programs as fuzzing seeds
- **Evaluation**
  - Achieve better coverage increase/reach
  - Detect 4 long-standing bugs in well-known projects
- **Open-sourced** at GitHub with open-source components*
  - Contribute to the reproducible, reusable research community

* https://github.com/OpenXiangShan/xfuzz. Thank LibAFL, rfuzz, DifuzzRTL, SIC, and DiffTest.