BEC: Bit-Level Static Analysis for Reliability against Soft Errors

Yousun Ko and Bernd Burgstaller

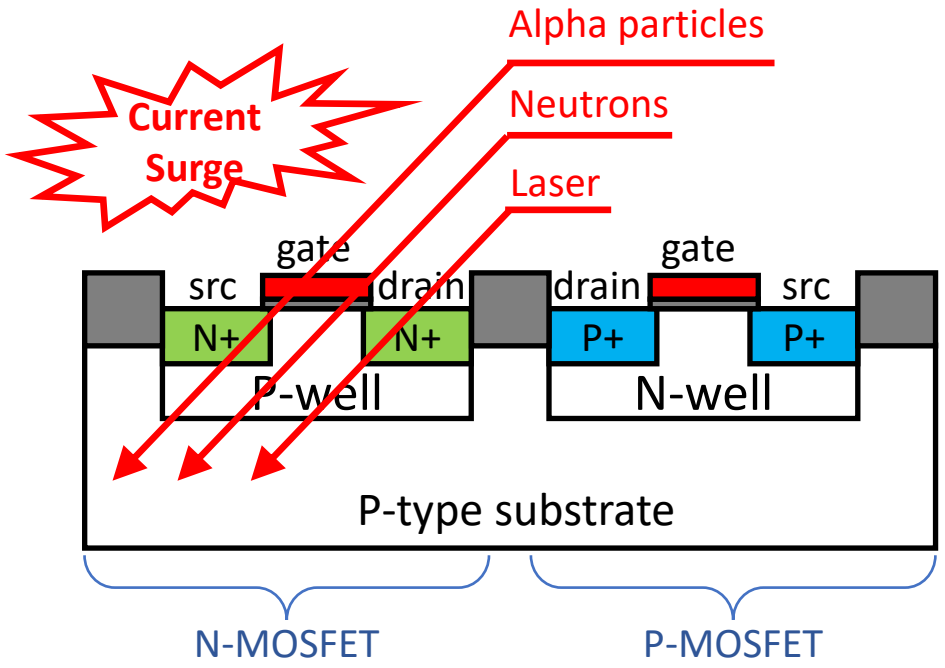
Yonsei University

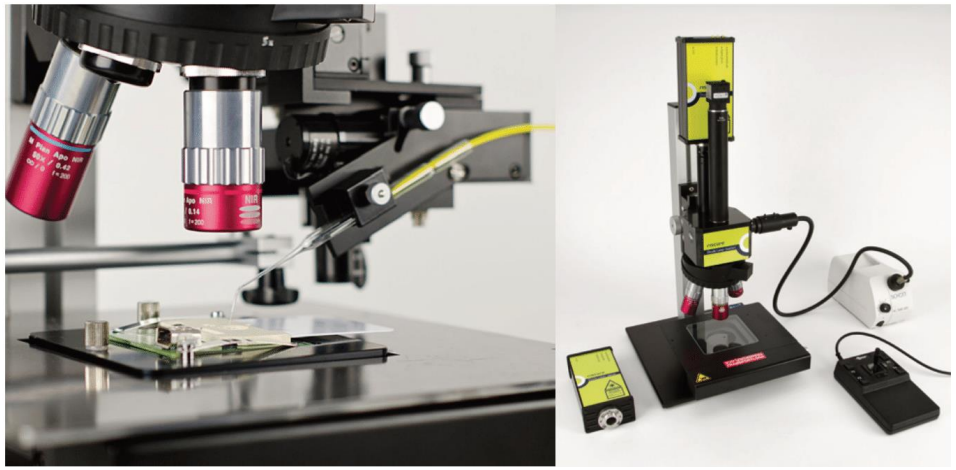
CGO 2024

2024. 03. 05

What are Soft Errors?

- Transient bit flips by accident or on purpose. Stay until overwritten.
- On sequential logic (registers, internal flip-flops), combinational logic, DRAM, caches, signal buses etc.
- Real-world observations
  - 2128 single bit upsets in SRAM of a satellite during a 286 day mission in low-earth orbit [NBM+2021]
  - Linux kernel development delayed due to random memory corruption [Torvalds2022]





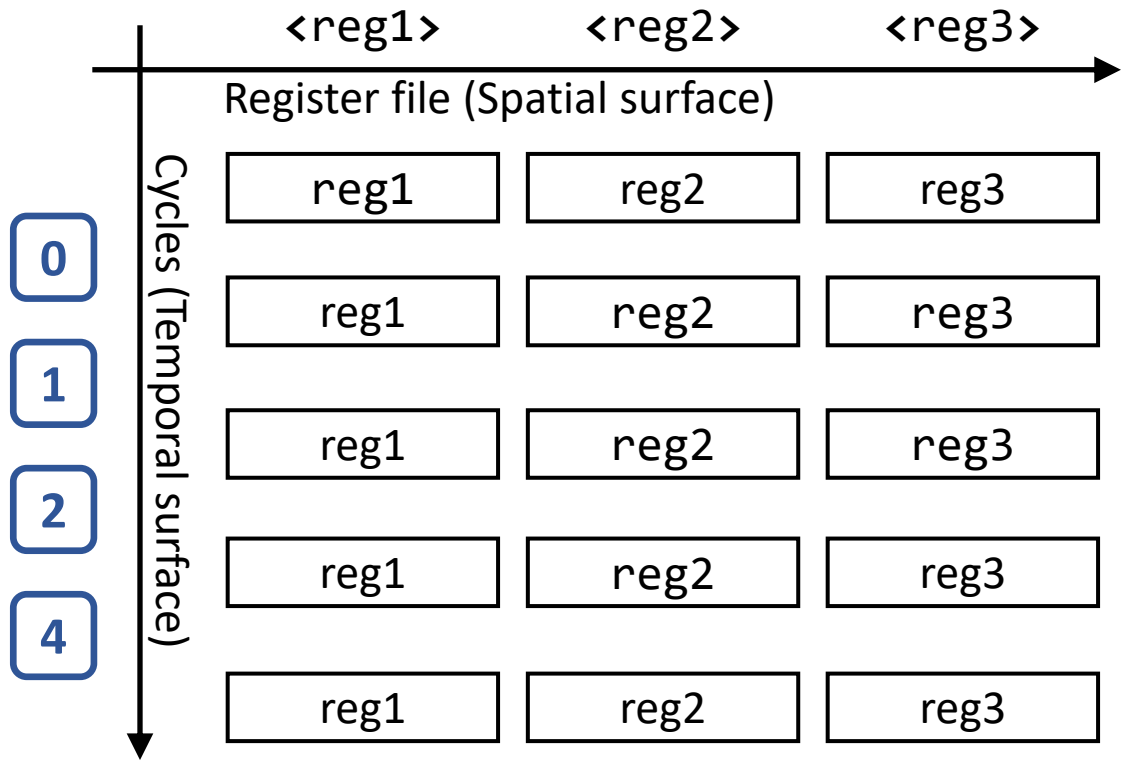
Diode Laser Station with Multi-Area upgrade option

Diode Laser Station with Microscope

Courtesy of *RISCURE* *https://www.riscure.com*

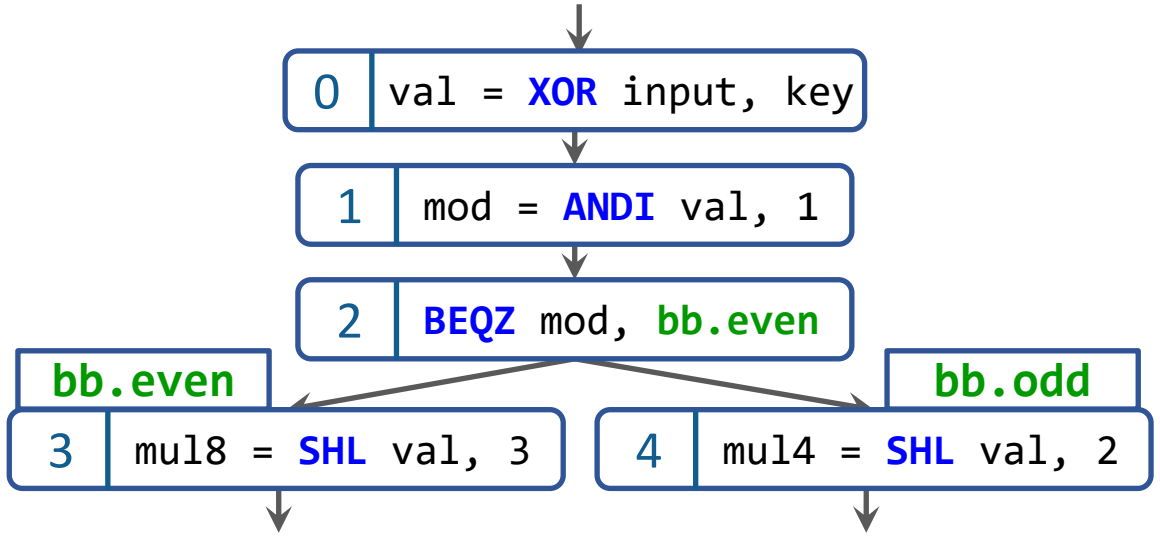
Testing Reliability against Soft Errors

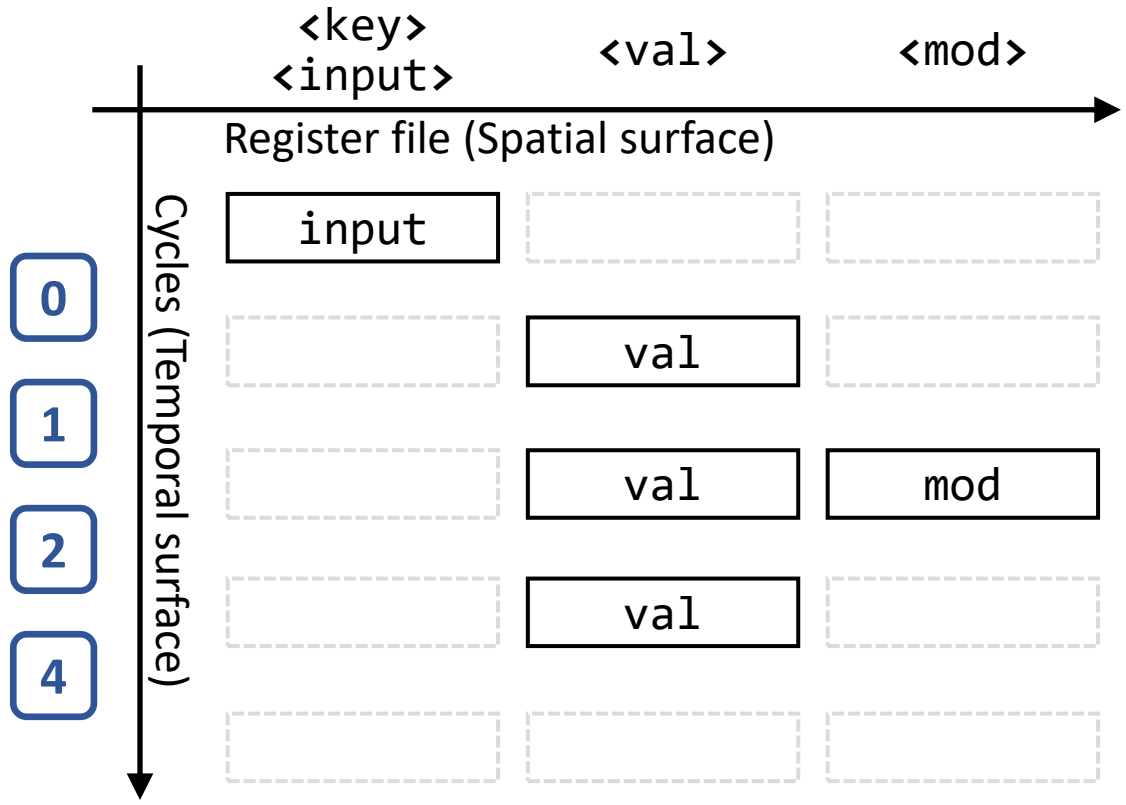
- Inject a soft error on hardware while a program is running, and observe the outcome



Testing Reliability against Soft Errors

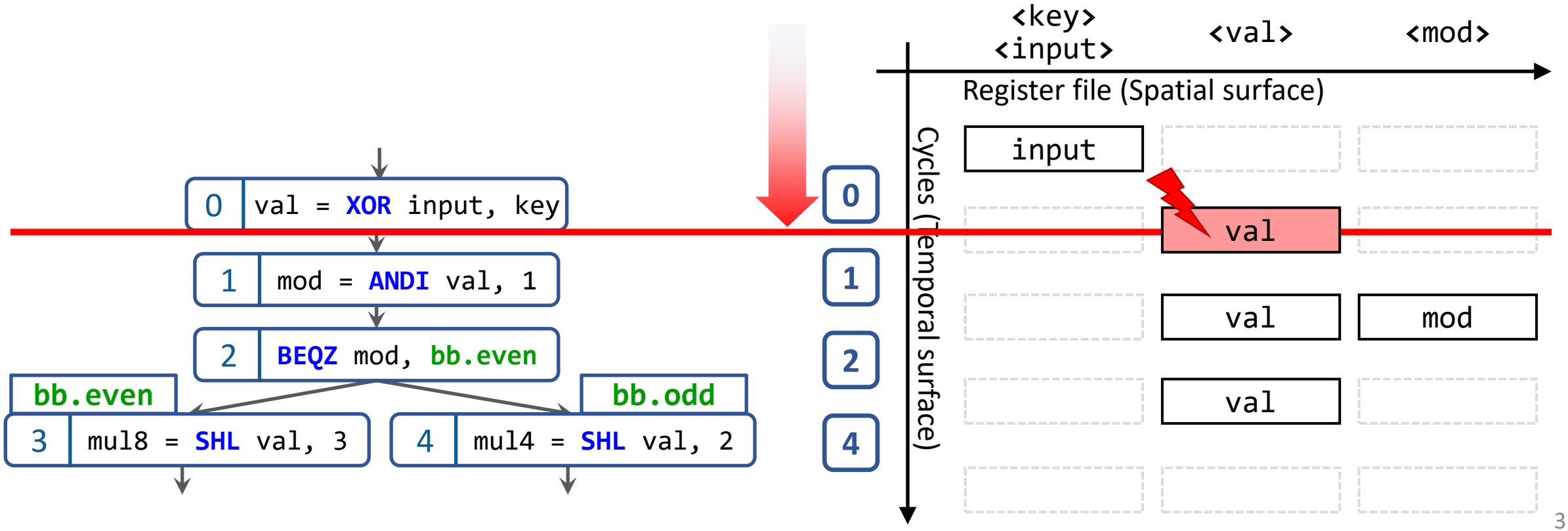
- Inject a soft error on hardware while a program is running, and observe the outcome
- Temporal-spatial location to inject a soft error can be narrowed down to live registers





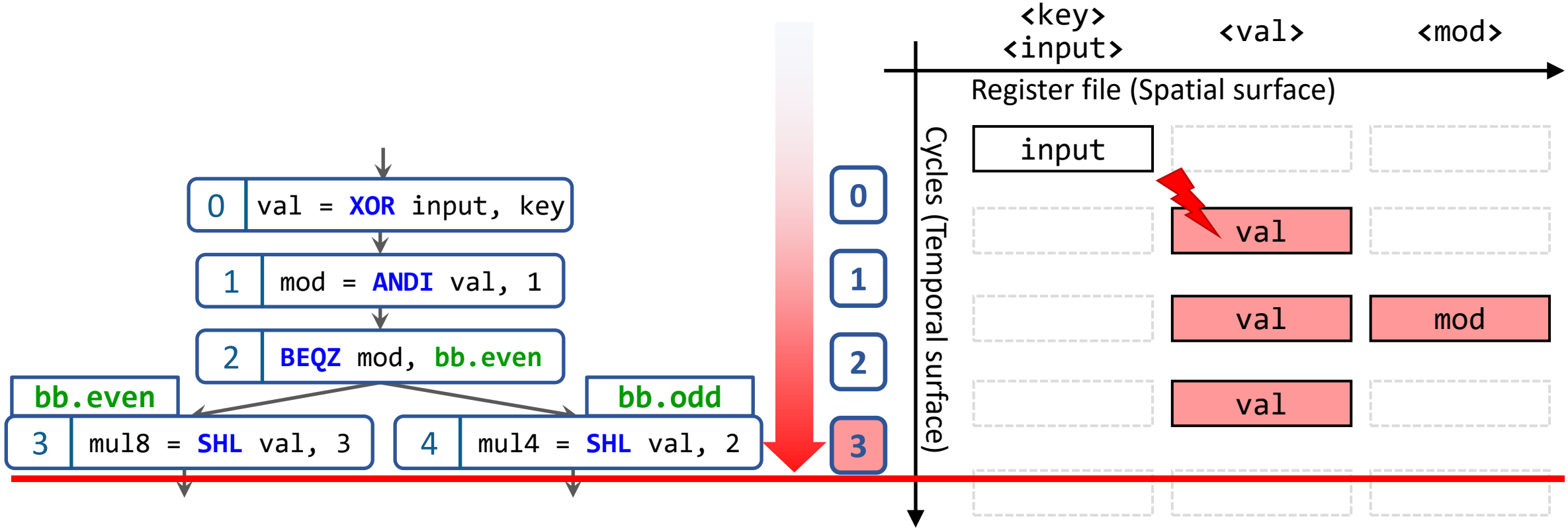
Testing Reliability against Soft Errors

- Inject a soft error on hardware while a program is running, and observe the outcome
- Temporal-spatial location to inject a soft error can be narrowed down to live registers



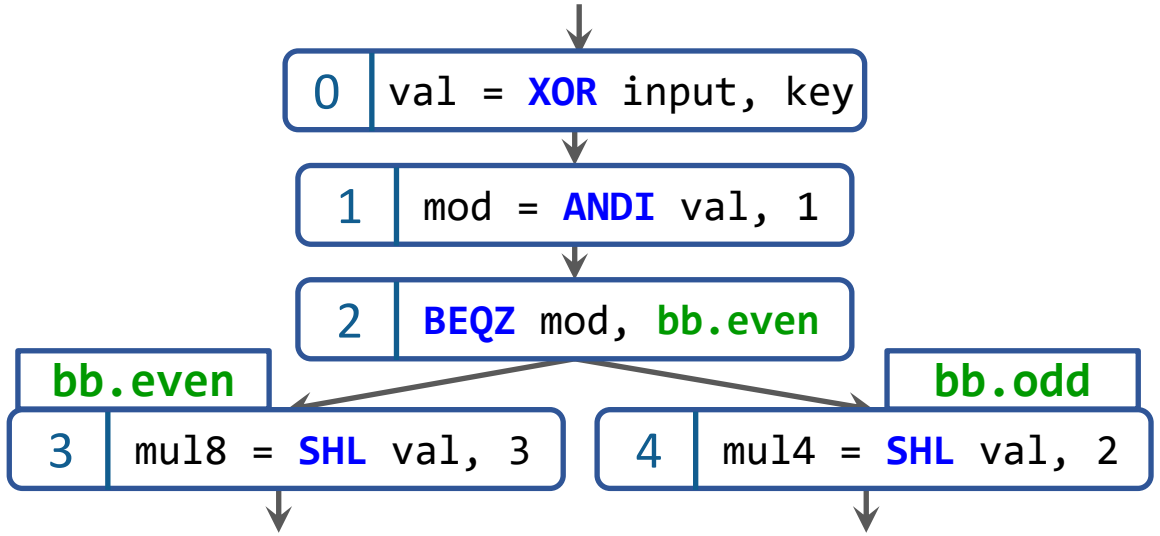
Testing Reliability against Soft Errors

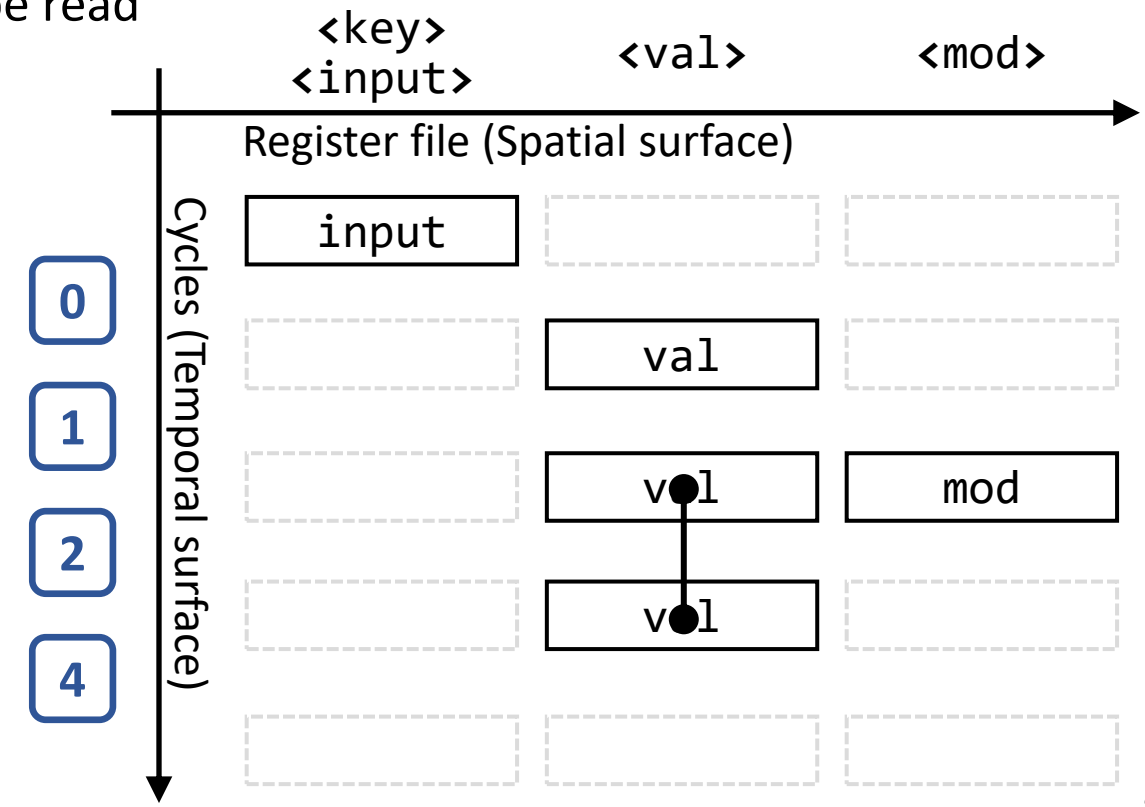
- Inject a soft error on hardware while a program is running, and observe the outcome
- Temporal-spatial location to inject a soft error can be narrowed down to live registers
- **`val`** is corrupted, error propagated to **`mod`**, and the control flow is diverted



Previous Approaches: Value-level and Dynamic

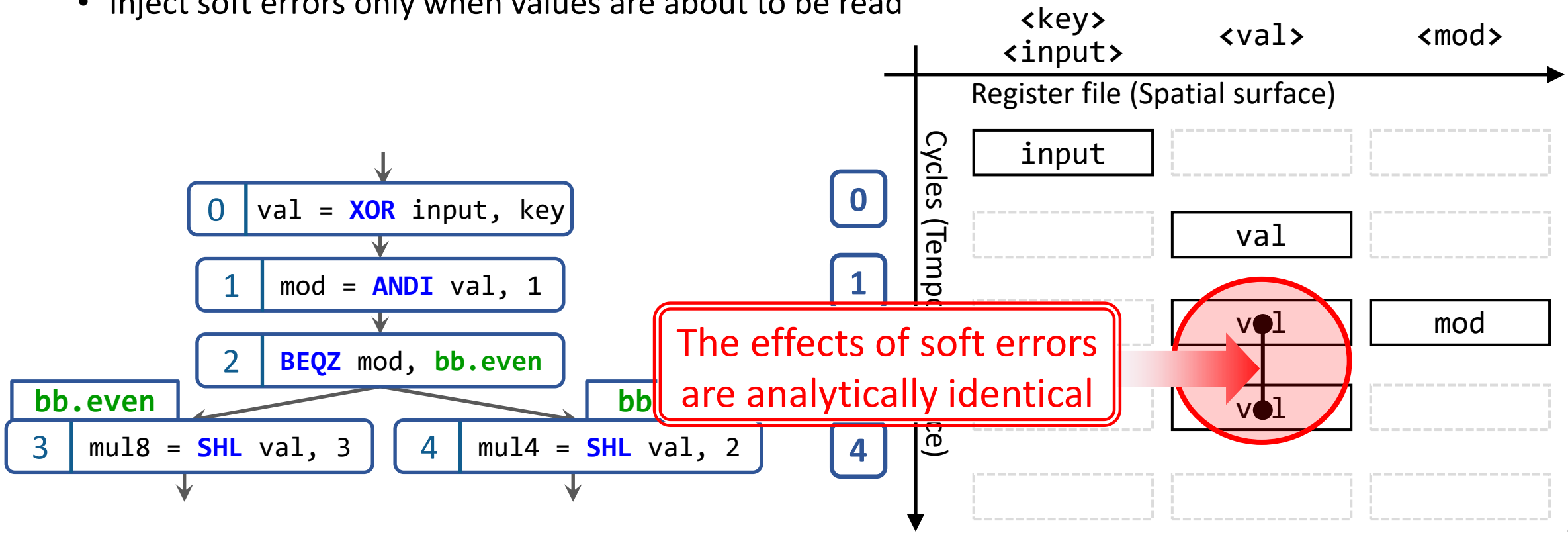
- Dynamic analysis
  - Performed on each program trace
  - Under-approximates. Analysis required for every run of a program
- Value-level analysis (Inject-on-Read)
  - Inject soft errors only when values are about to be read





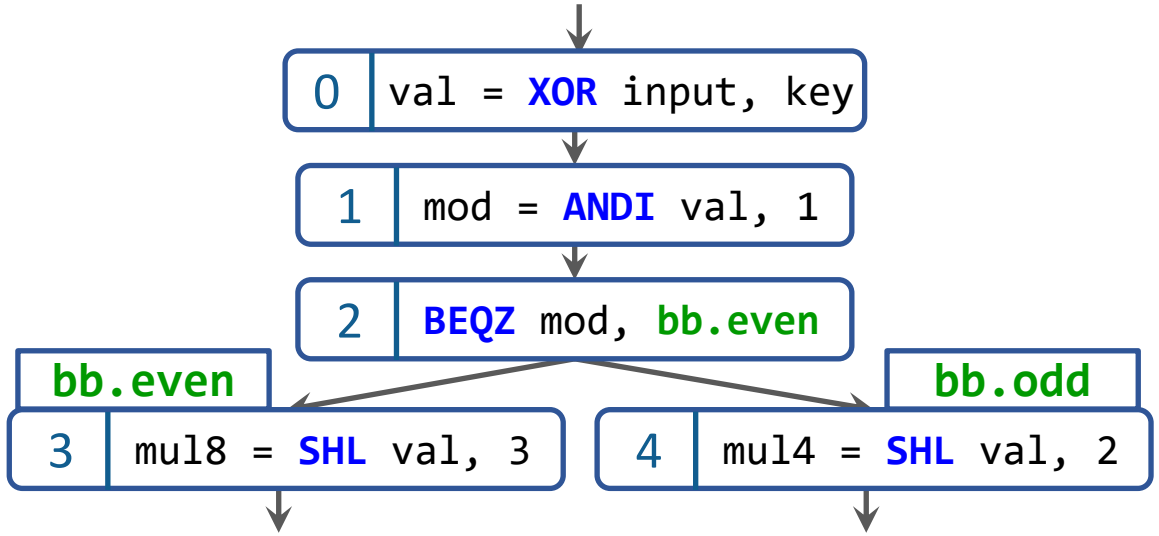
Previous Approaches: Value-level and Dynamic

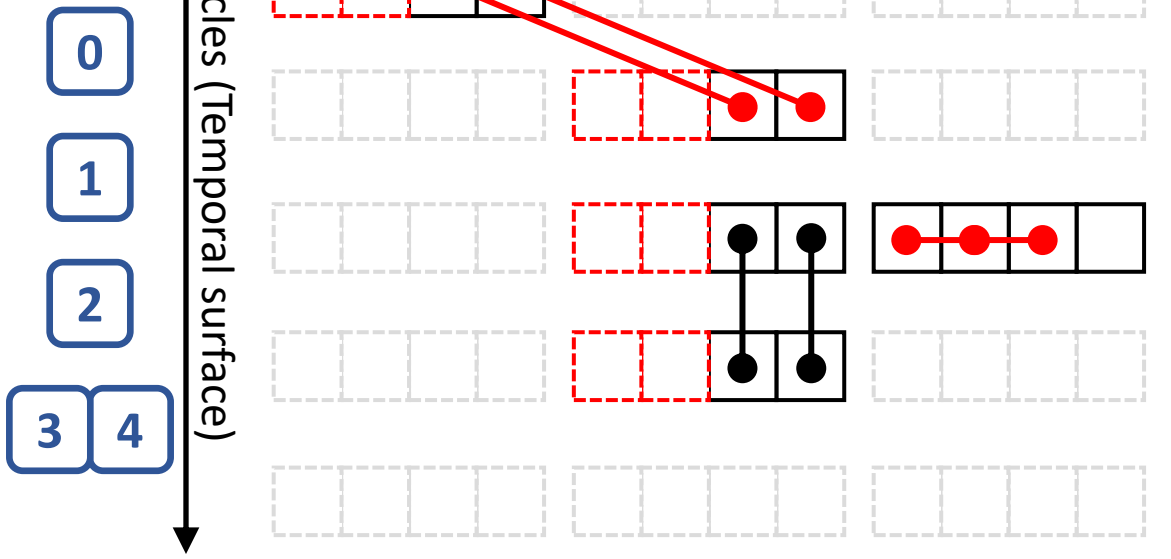
- Dynamic analysis
  - Performed on each program trace
  - Under-approximates. Analysis required for every run of a program
- Value-level analysis (Inject-on-Read)
  - Inject soft errors only when values are about to be read



The Proposed Approach: Bit-level Static Analysis

- Static analysis: **BEC** (Bit-level Error Coalescing) Analysis
  - The first bit-level static program analysis that tracks and classifies the effect of soft errors based on program semantics
  - Performed only once per program, at compile-time
  - Overapproximates. Analysis holds for any program run
  - Can be used by other compiler analyses and optimizations
- Bit-level analysis
  - A natural match for soft errors
  - Unveil hidden optimization opportunities in live values

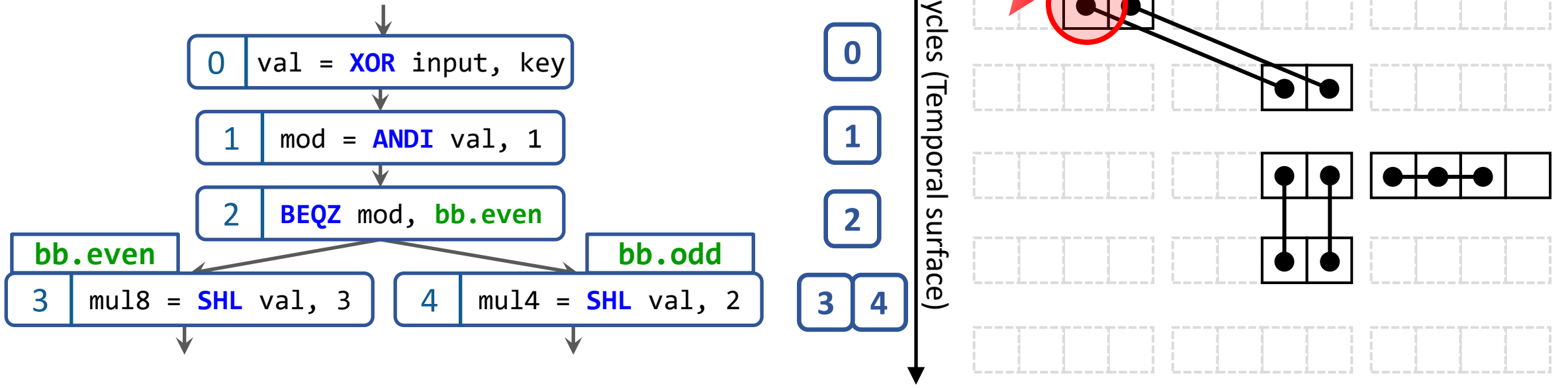




Use Case 1: Fault Injection Campaign Pruning

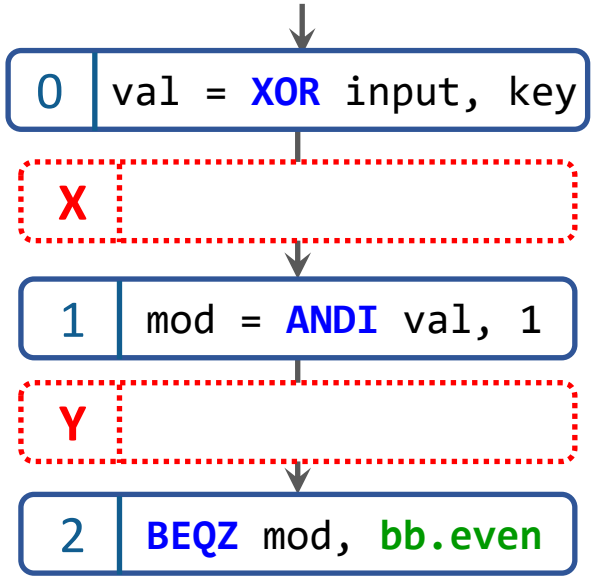
- Inject soft errors only on bits whose effects are analytically distinguishable
- **16** fault injection runs → **6** fault injection runs
- No loss in coverage

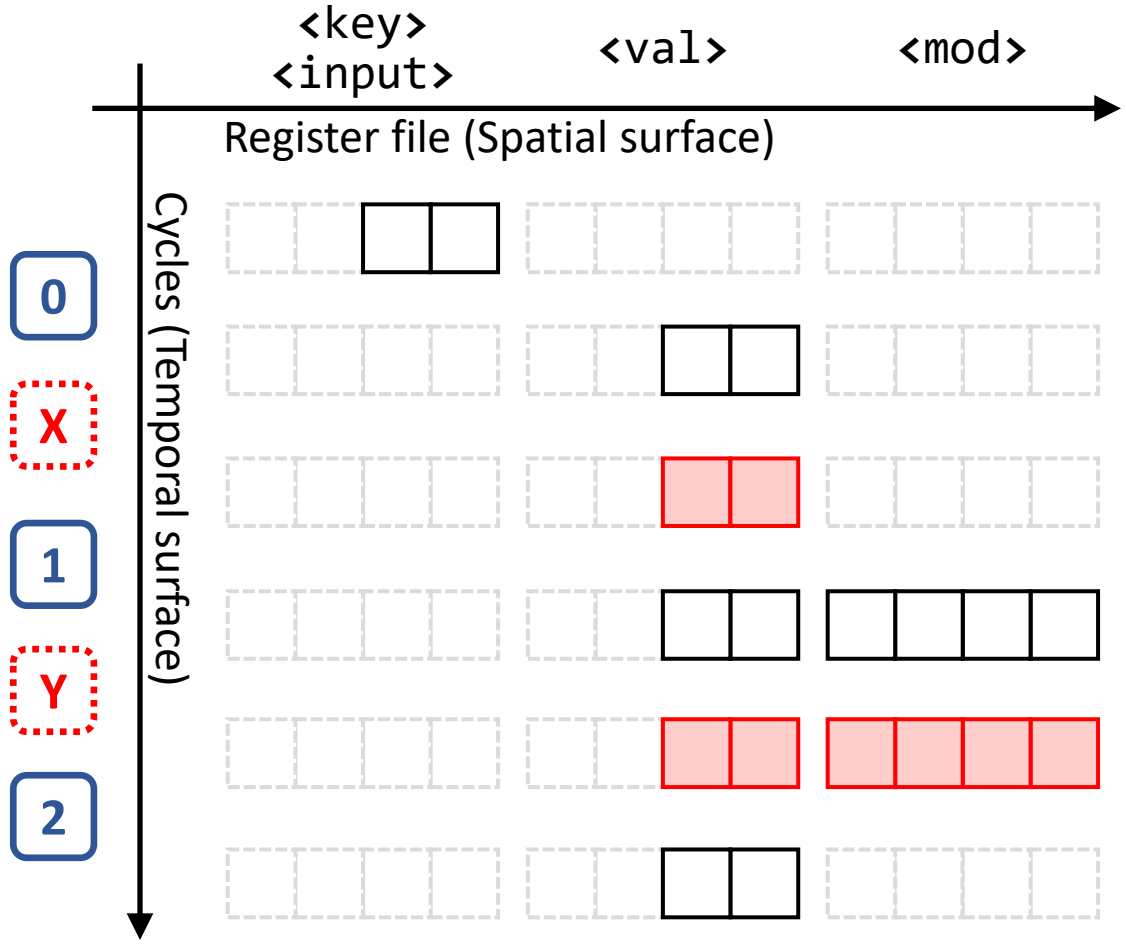
A fault site:
One fault-injection run is required per fault site to assess its vulnerability.



Use Case 2: Vulnerability-aware Instr. Scheduling

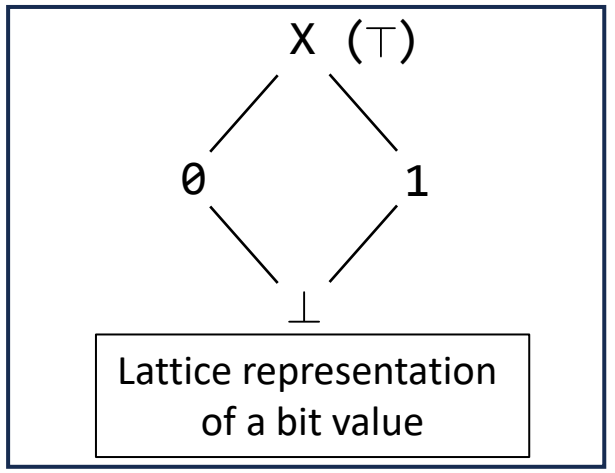
- The use case within the compiler
- Schedule instructions to minimize the number of live fault sites
- Scheduling a new instruction
  - at **X** adds **2** live fault sites
  - at **Y** adds **6** live fault sites

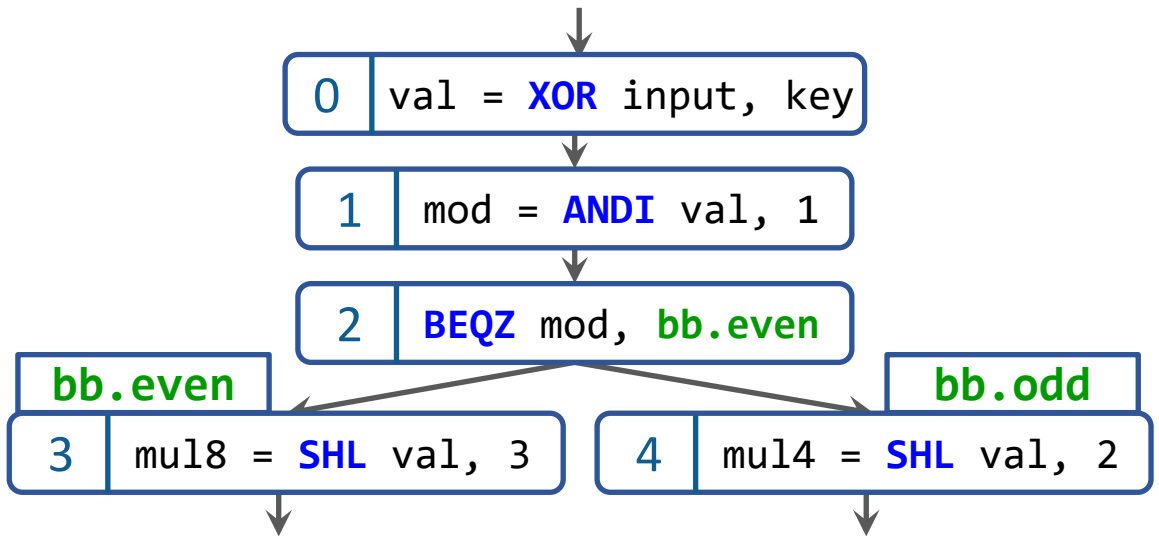


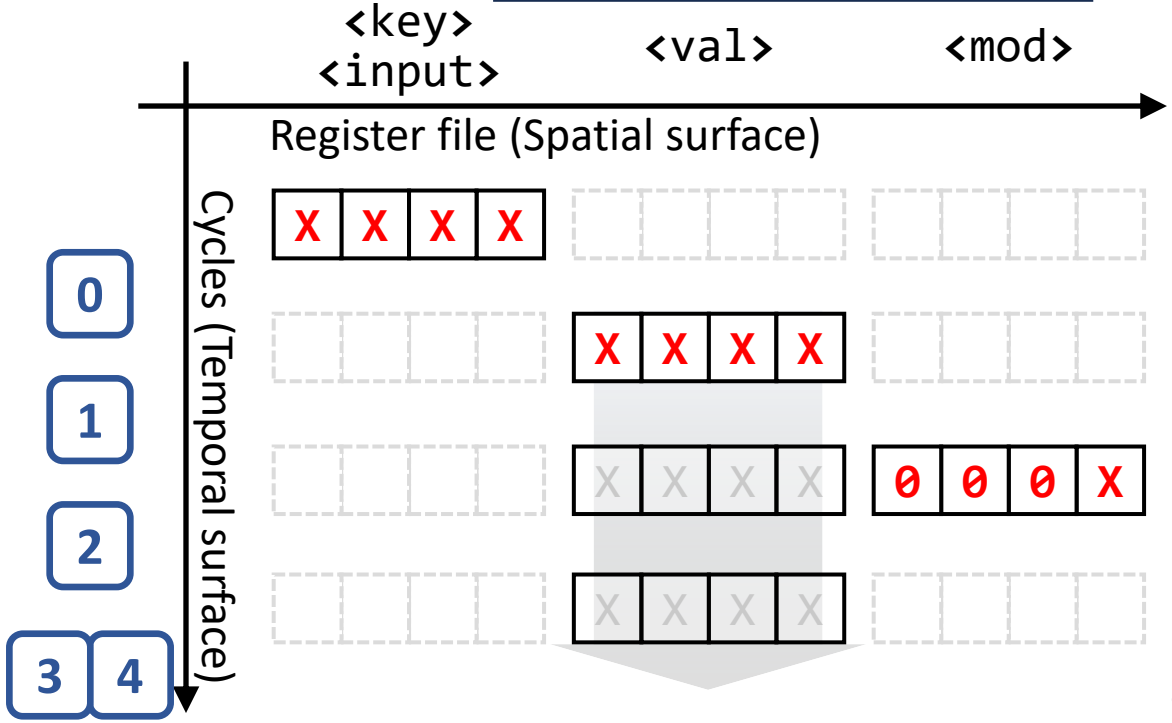


BEC: Bit-value Analysis

- Bit values help to understand the semantics of instructions
- Forward data-flow analysis
- Abstract interpretation
- Performed once per value definition





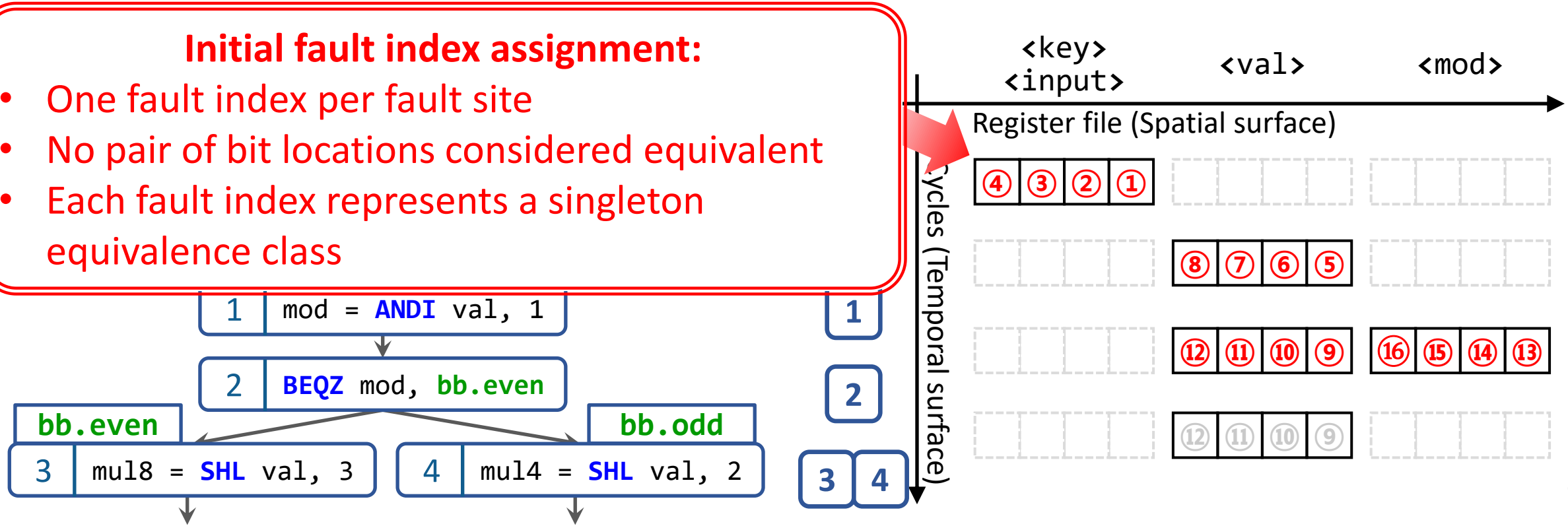


BEC: Fault Index

- A Fault Index labels the effect of soft errors at each fault site
  - **Fault index ⓪** is reserved for dead (ineffective) soft errors
- Equivalence relation
- Minimize the number of classes in the equivalence relation by coalescing

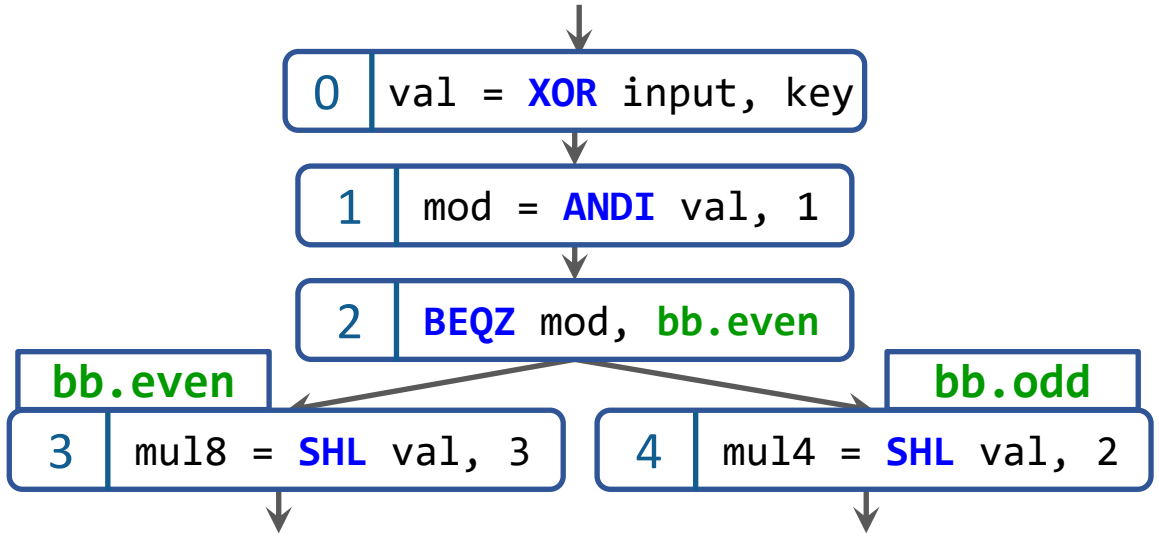
Initial fault index assignment:

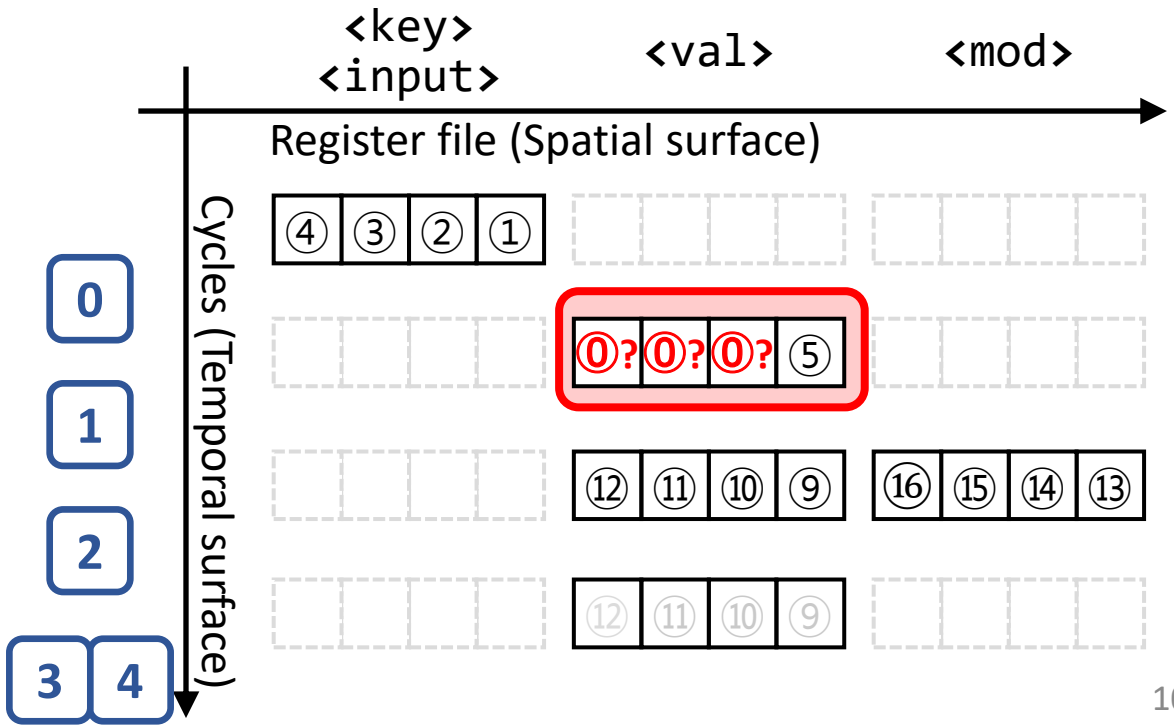
- One fault index per fault site
- No pair of bit locations considered equivalent
- Each fault index represents a singleton equivalence class



BEC: Fault Index Coalescing

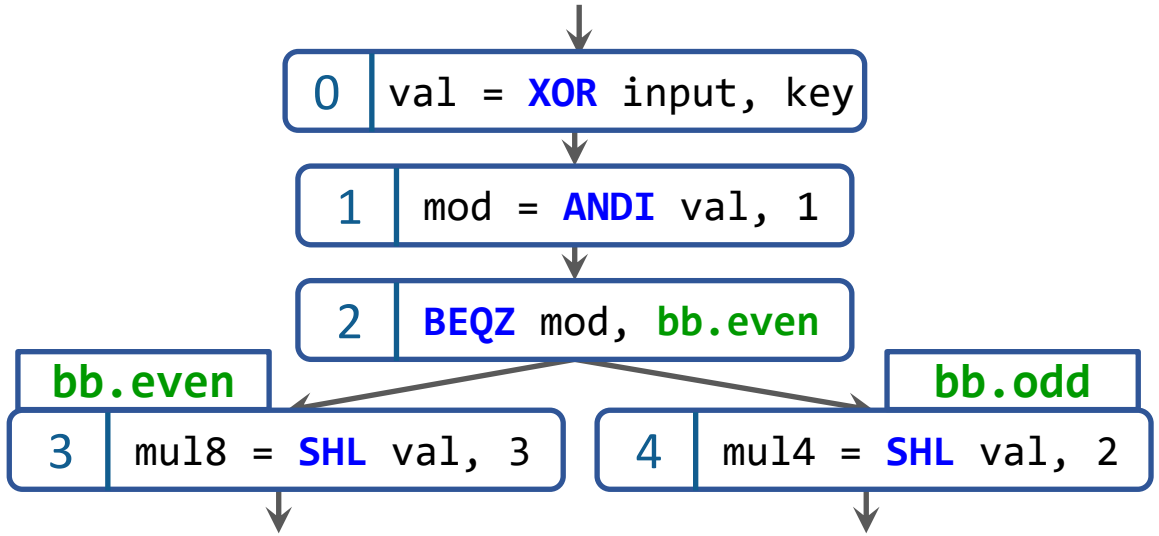
- Coalesces two equivalence classes if analysis finds fault indices to be equivalent
- Backward data-flow analysis
- Exploits the semantics of instructions
  - **ANDI** instruction may mask three most-significant bits

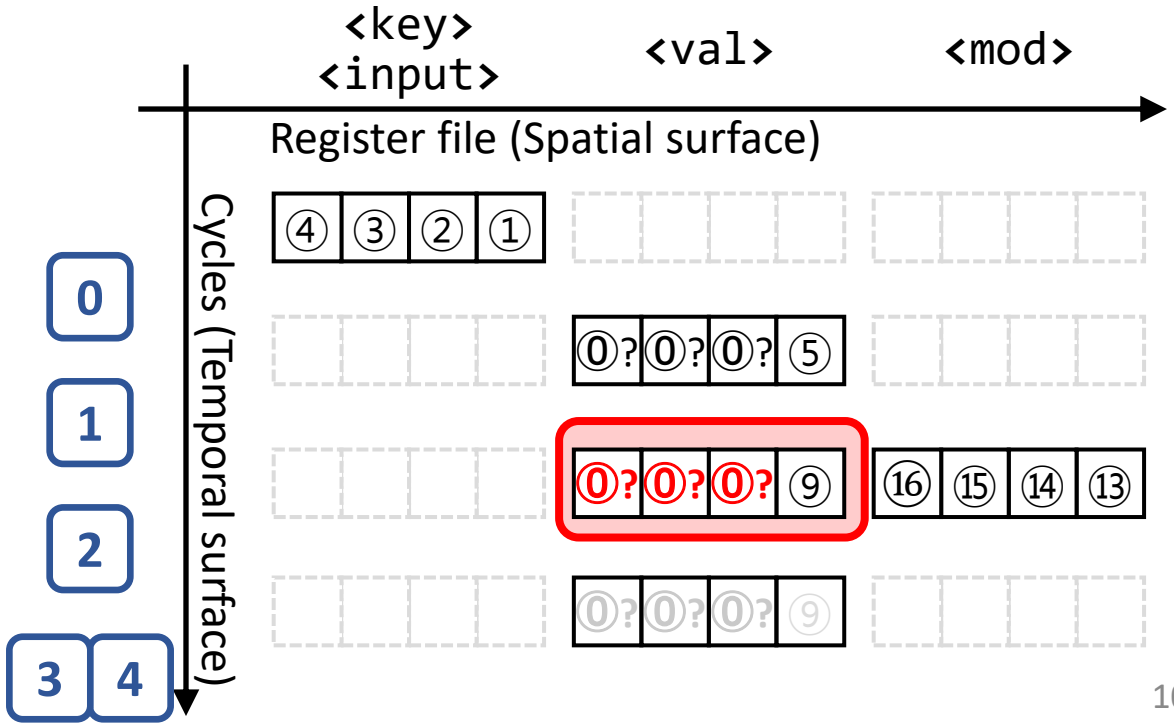




BEC: Fault Index Coalescing

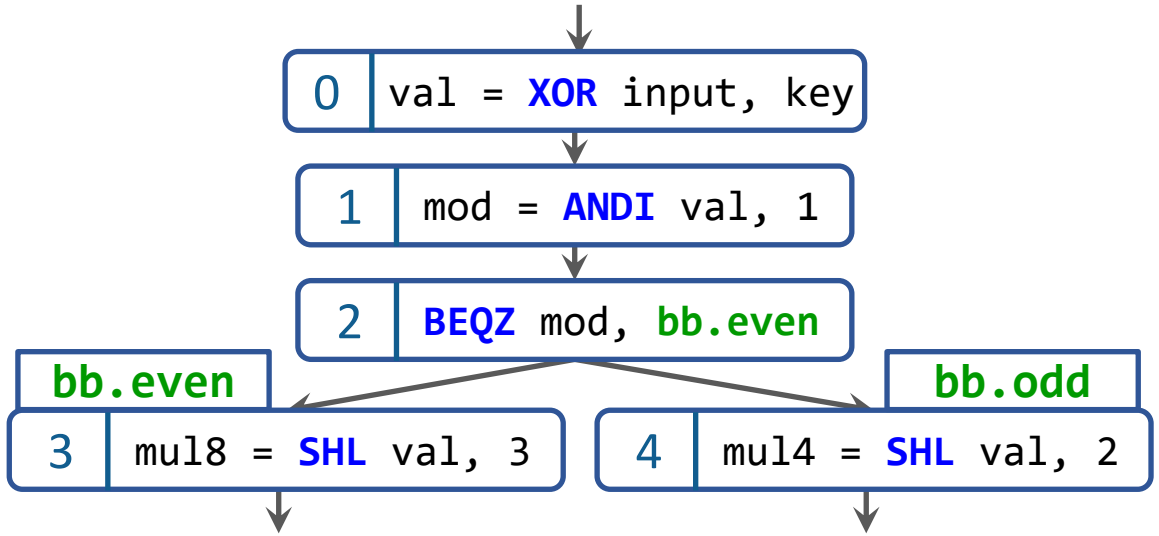
- Coalesces two equivalence classes if analysis finds fault indices to be equivalent
- Backward data-flow analysis
- Exploits the semantics of instructions
  - **ANDI** instruction may mask three most-significant bits
  - **SHL** instructions may mask two or three most-significant bits

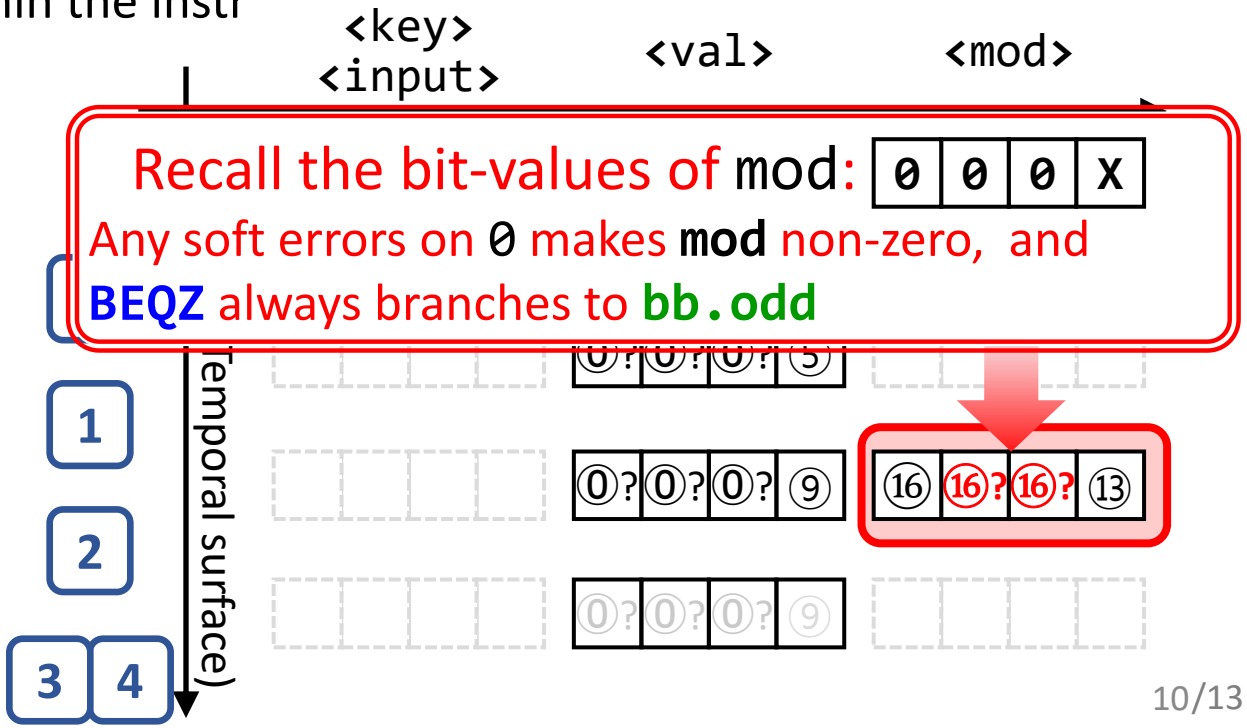




BEC: Fault Index Coalescing

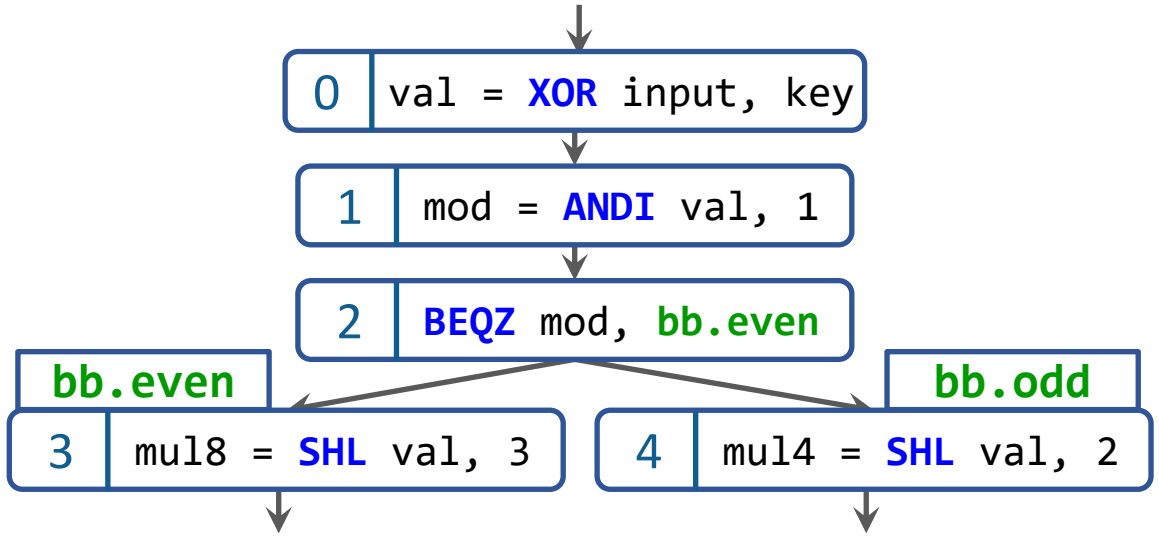
- Coalesces two equivalence classes if analysis finds fault indices to be equivalent
- Backward data-flow analysis
- Exploits the semantics of instructions
  - **ANDI** instruction may mask three most-significant bits
  - **SHL** instructions may mask two or three most-significant bits
  - **BEQZ** instruction may coalesce fault indices within the instr

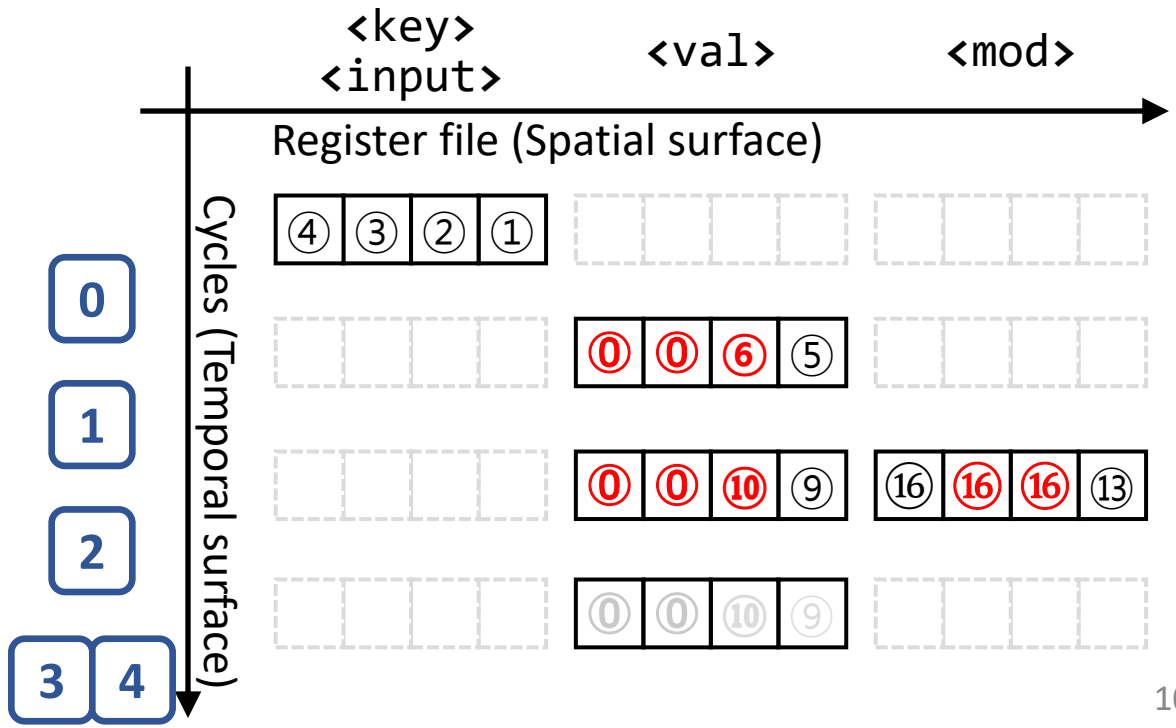




BEC: Fault Index Coalescing

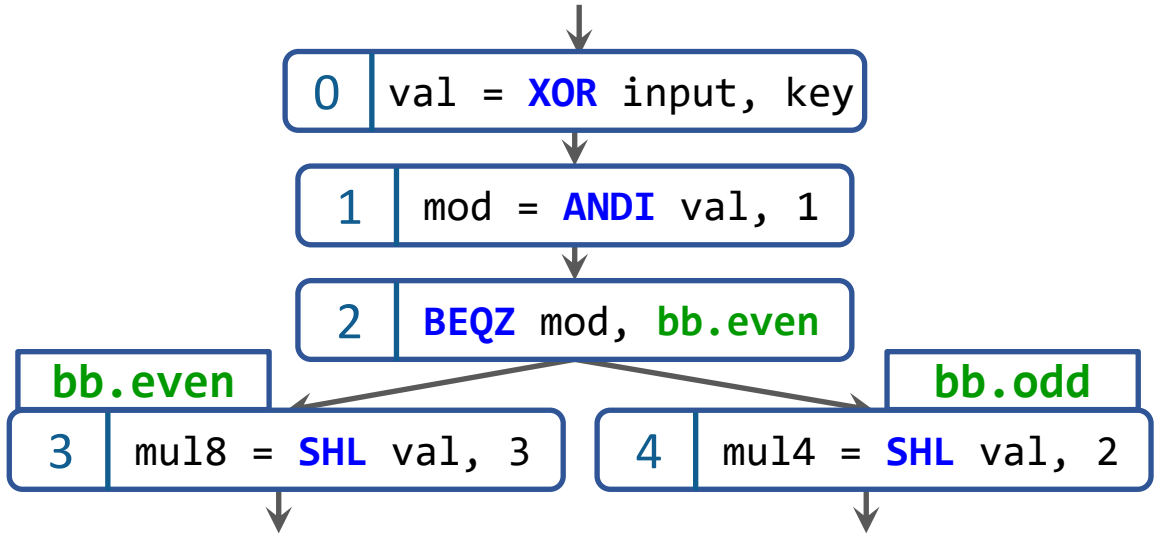
- Coalesces two equivalence classes if analysis finds fault indices to be equivalent
- Backward data-flow analysis
- Probes every paths in the data-flow graph for any conflicts in guesses
- Overapproximates if the corrupted data is read by multiple sites

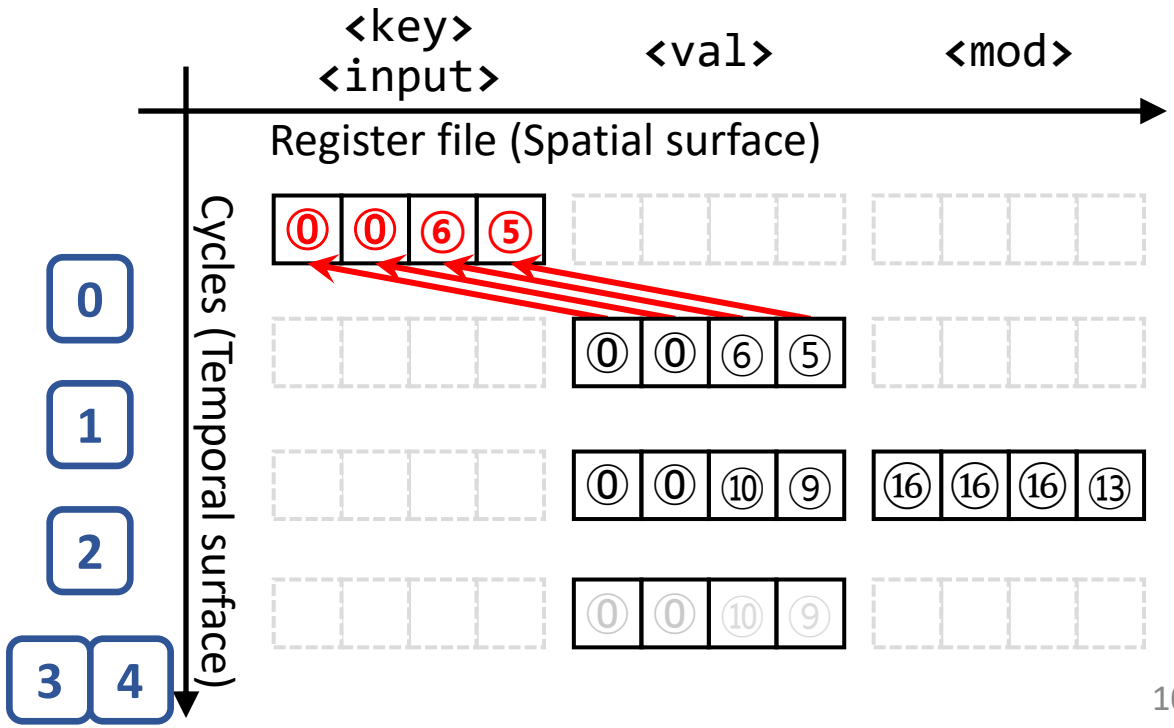




BEC: Fault Index Coalescing

- Coalesces two equivalence classes if analysis finds fault indices to be equivalent
- Backward data-flow analysis
- Coalesces fault indices backward until the analysis reaches the fixed point
  - **XOR** instruction allows fault index coalescing irrespective of bit values





Experimental results: Fault Injection Pruning

	bitcount	dijkstra	CRC32	adpcm enc	adpcm dec	AES	RSA	SHA
Live in values	26 272	230 336	245 760	2 819 904	2 003 744	150 112	1 026 304	421 632
Live in bits	20 571	229 409	211 176	2 424 874	1 653 714	105 025	1 025 436	371 294
Masked bits	2 506	70	7 368	71 000	258 000	680	434	10 660
Inferrable bits	3 195	857	26 216	324 030	92 030	44 407	434	39 678
Total FI runs pruned	**21.70 %**	**0.40 %**	**14.07 %**	**14.01 %**	**17.47 %**	**30.04 %**	**0.08 %**	**11.94 %**

- Implemented in LLVM 16, validated and evaluated on RISC-V
- Pruned up to 30.04% (13.71% on average)
- Effectiveness varies by the benchmark characteristics
  - bitcount: Abundant bit operations ☺
  - Adaptive differential pulse-code modulation: value quantization ☺
  - CRC32: Abstract Binary Interface ☺
  - RSA: arithmetic operations ☹
  - dijkstra: arithmetic and memory operations ☹
  - AES: memory operations but with bit-value agnostic machine instructions ☺ ☺

Experimental results: Vulnerability-aware Instr. Sched.

	bitcount	dijkstra	CRC32	adpcm enc	adpcm dec	AES	RSA	SHA
Total fault space	541 696	27 286 528	2 922 496	58 426 368	44 085 248	3 180 544	18 295 808	7 483 392
Best reliability	85 018	159 966	348 384	28 401 348	19 400 720	1 928 214	8 650 606	2 559 116
Worst reliability	94 366	166 074	394 040	28 530 244	19 538 104	2 077 194	8 764 640	2 688 188
Worst/Best	111.00 %	103.82 %	113.11 %	100.45 %	100.71 %	104.10 %	101.32 %	105.04 %
Improved Reliability	**+11.00 %**	**+3.82 %**	**+13.11 %**	**+0.45 %**	**+0.71 %**	**+4.10 %**	**+1.32 %**	**+5.04 %**

- Implemented in LLVM 16, validated and evaluated on RISC-V
- Reliability improved up to 13.11% (4.94% on average)
  - Comparable to existing stand-alone methods
- The more dead fault sites the better
- Longevity of dead fault sites

More in the Paper and the Code!

- Pre-print: Yousun Ko and Bernd Burgstaller, *BEC: Bit-Level Static Analysis for Reliability against Soft Errors*, https://arxiv.org/abs/2401.05753

- Source code on GitHub: https://github.com/yousunko/BEC

More in the Paper and the Code!

- Pre-print: Yousun Ko and Bernd Burgstaller, *BEC: Bit-Level Static Analysis for Reliability against Soft Errors*, https://arxiv.org/abs/2401.05753

- Source code on GitHub: https://github.com/yousunko/BEC

Thank you! ☺