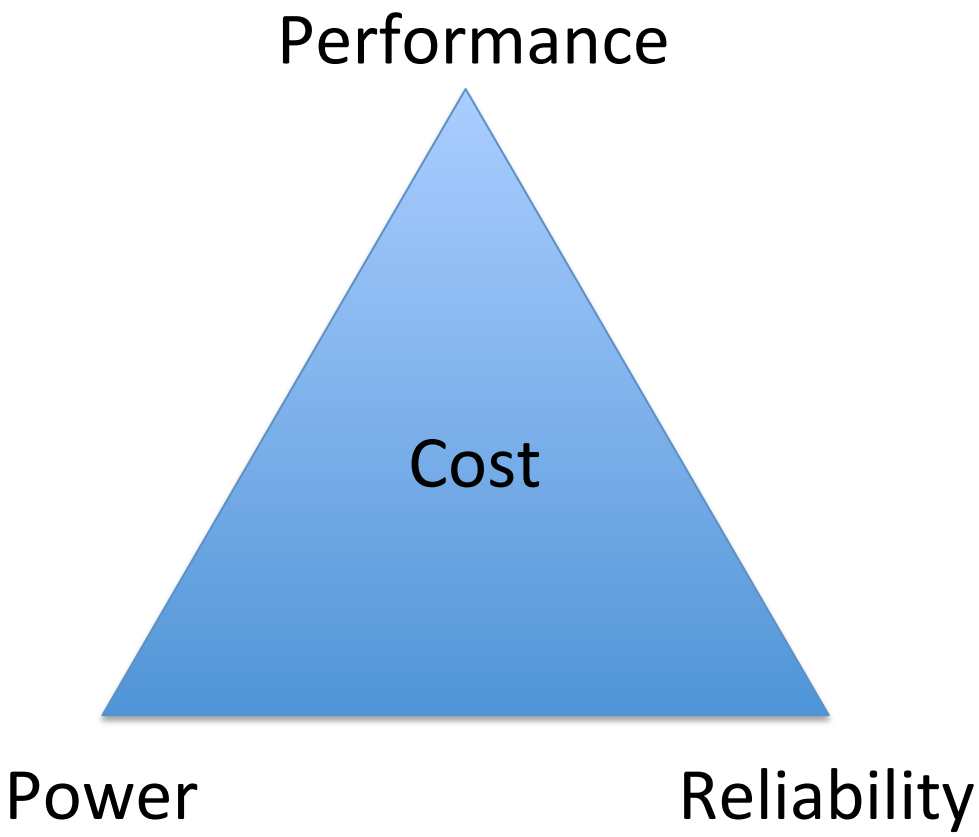
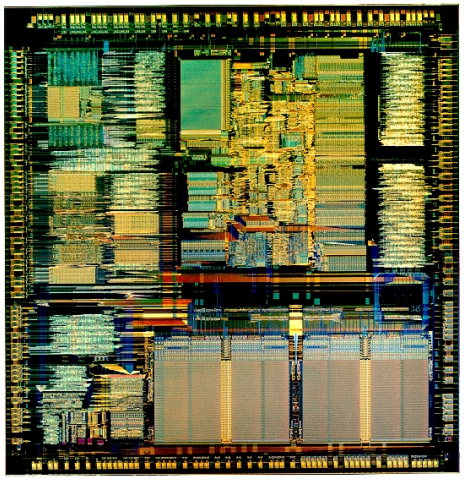
SPECS: A Lightweight Runtime Mechanism for Protecting Software from Security-Critical Processor Bugs

Matthew Hicks, Michigan
Cynthia Sturton, UNC
Samuel T. King, Twitter
Jonathan M. Smith, UPenn

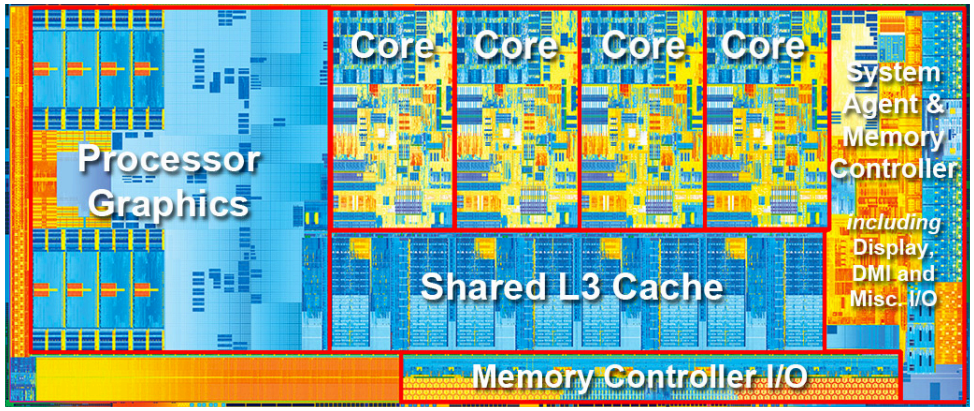
Architectural success leads to increased complexity



Architectural success leads to increased complexity

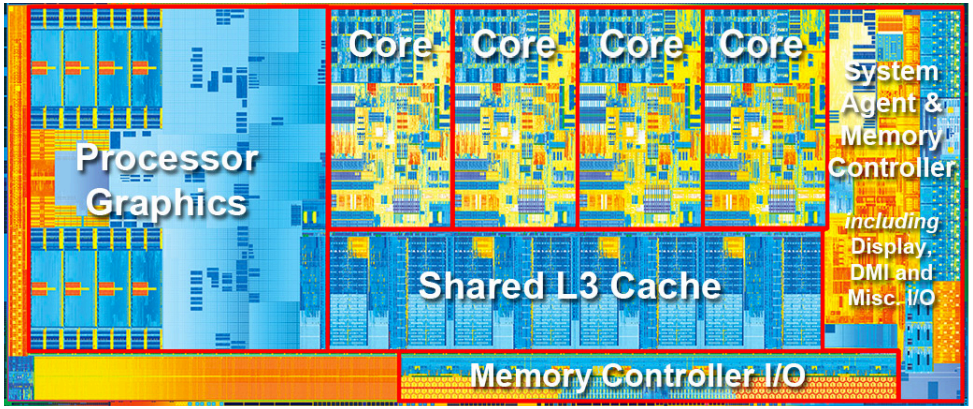


1985
.3M Transistors



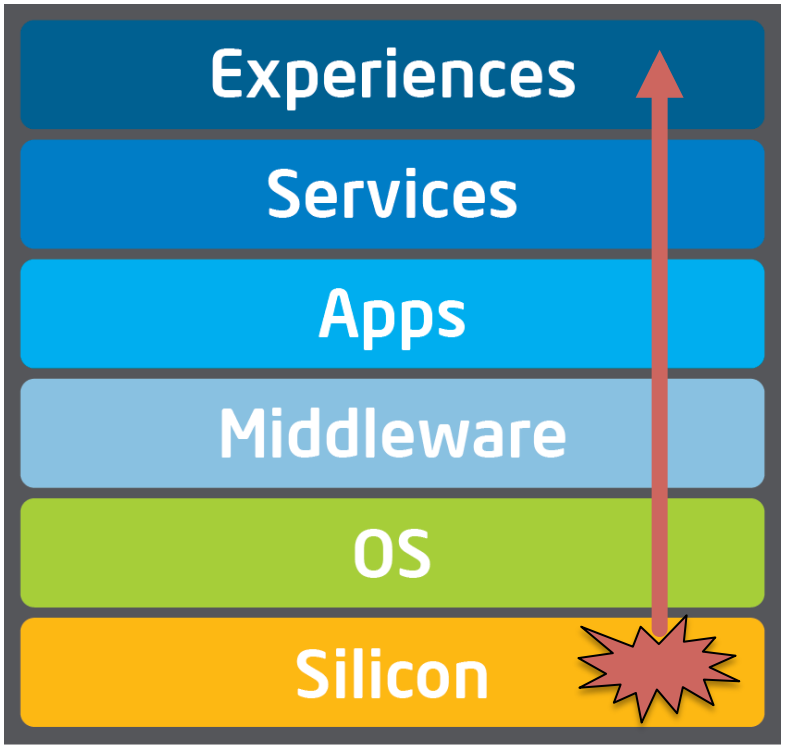
2012
1,400M Transistors

Increased complexity leads to more bugs



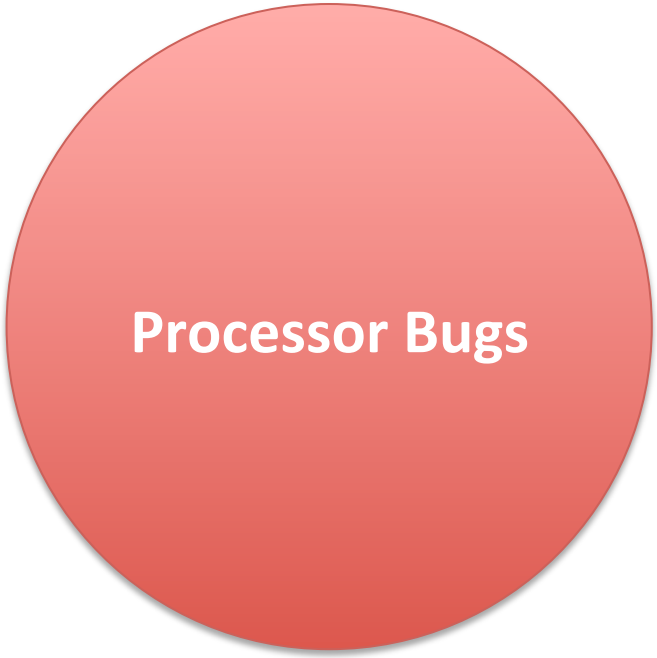
4 years
136 errata
3 bugs/month

Processor bugs are permanent and powerful

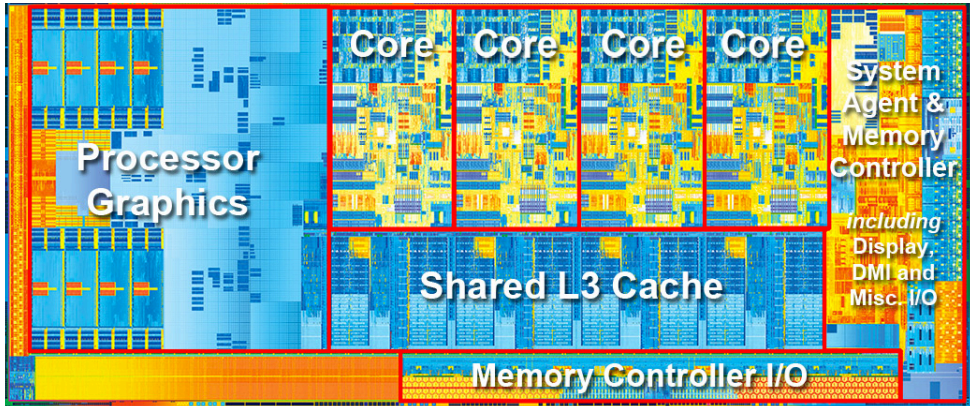


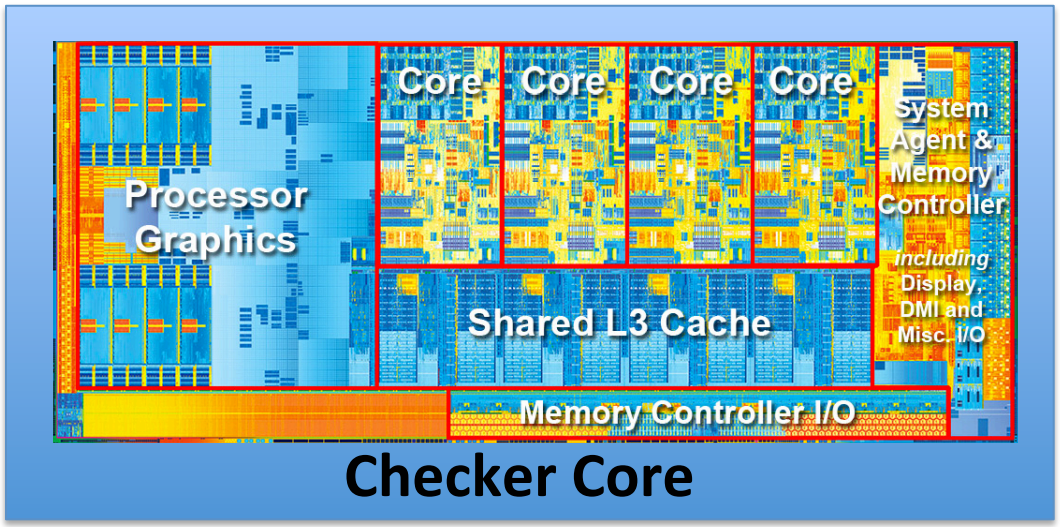
Adapted by Intel IT Center from Genevieve Bell's IDF 2013 Keynote Address

Hardware-only approaches are expensive, while software-only approaches are limited



Hardware-only approaches are expensive, while software-only approaches are limited

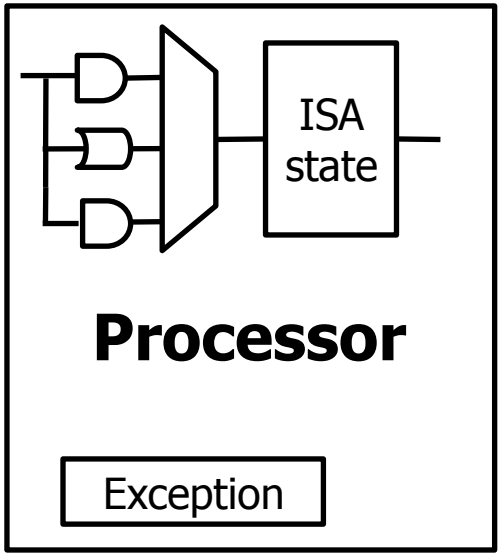




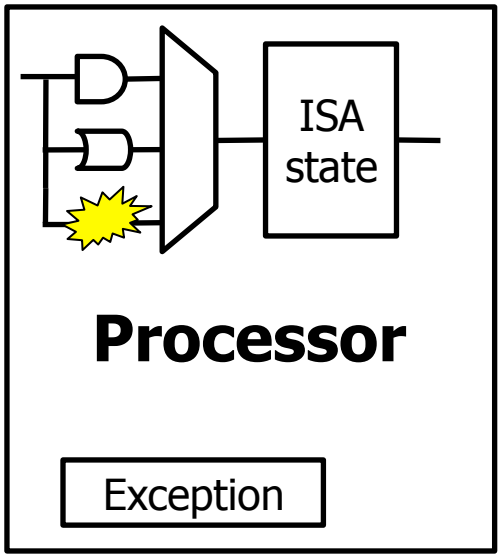
Hardware-only approaches are expensive, while software-only approaches are limited

SPECS is targeted hardware that provides ad hoc introspection points to recovery/ repair software

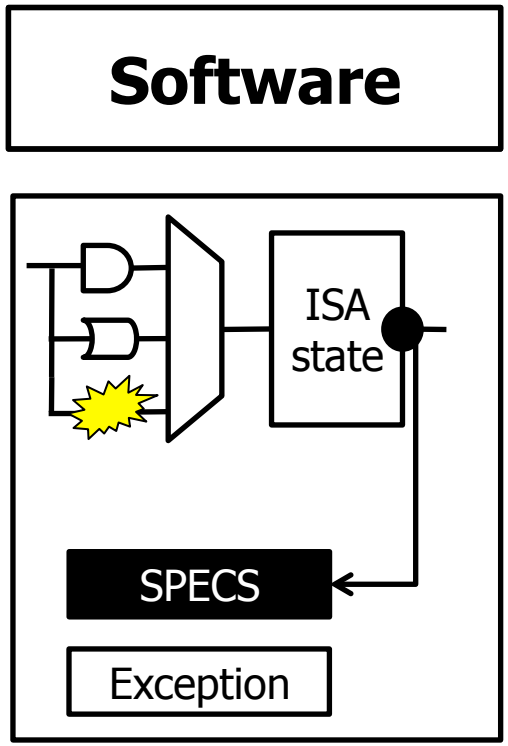
SPECS - Security-critical Processor Errata Catching System



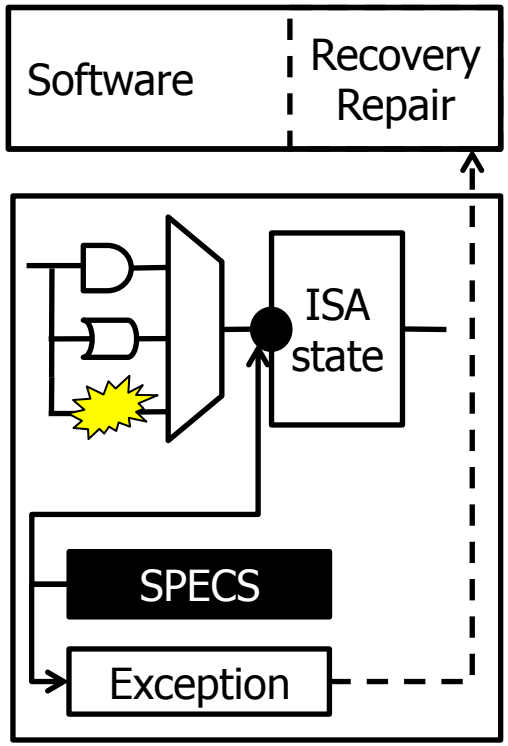
SPECS - Security-critical Processor Errata Catching System



SPECS - Security-critical Processor Errata Catching System



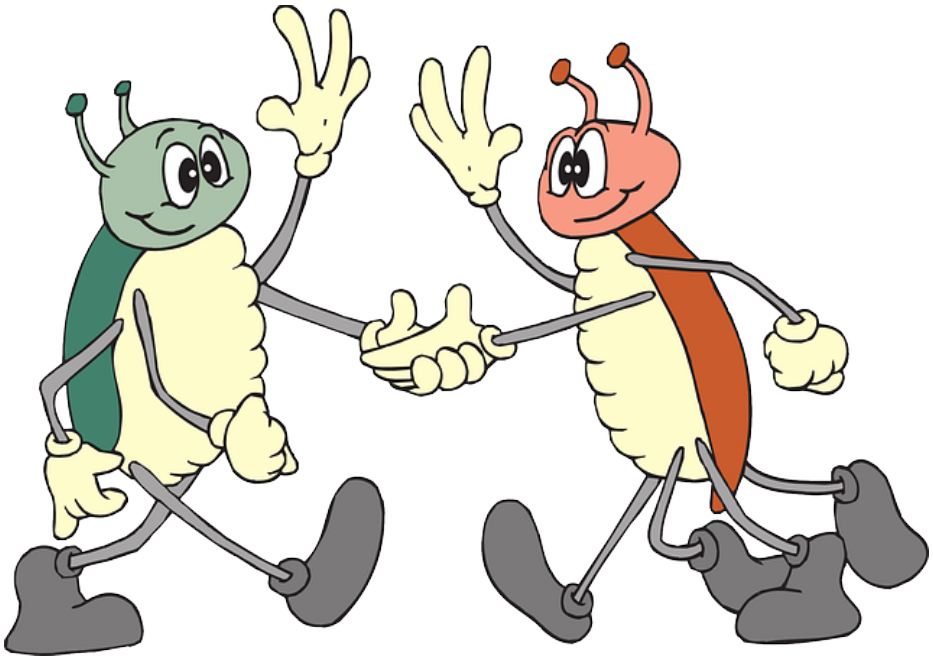
SPECS - Security-critical Processor Errata Catching System



Analyze commercial processor errata to quantify the magnitude of the problem

Athlon
Athlon X2
Athlon Neo
Athlon Neo X2
Athlon 64
Athlon 64 FX
Athlon 64 X2
Opteron
Opteron Dual-Core
Sempron
Sempron Dual-Core
Turion 64
Turion 64 X2
Turion Neo X2
Phenom II X3
Phenom II X4
Phenom II X6
Phenom II XLT
Phenom II Dual-Core
Phenom II Triple-Core
Phenom II Quad-Core
Sempron X2
Sempron Mobile
Turion II Dual-Core Mobile
Turion II Ultra Dual-Core Mobile
Turion II Neo Dual-Core Mobile
V-Series
V-Series Dual-Core
Athlon X2 Dual-Core
Sempron X2 Dual-Core
Turion X2 Dual-Core Mobile
Turion X2 Ultra Dual-Core Mobile
A-Series Mobile APU
E2-Series Mobile APU
A-Series APU
E2-Series APU
Athlon II X2 Dual-Core
Athlon II X4 Quad-Core
C-Series
C-Series Dual-Core
E-Series Dual-Core
E-Series
E-Series Dual-Core
G-Series
G-Series Dual-Core
Z-Series Dual-Core
FX-Series
Opteron 3200 Series
Opteron 3300 Series
Opteron 4200 Series
Opteron 4300 Series
Opteron 6200 Series
Opteron 6300 Series
R-Series APU
R-Series
FirePro APU
Athlon Dual-Core
Athlon Quad-Core
E-Series Mobile APU
G-Series Mobile APU
G-Series SoC
Opteron X1100 Series
Opteron X2100 Series APU

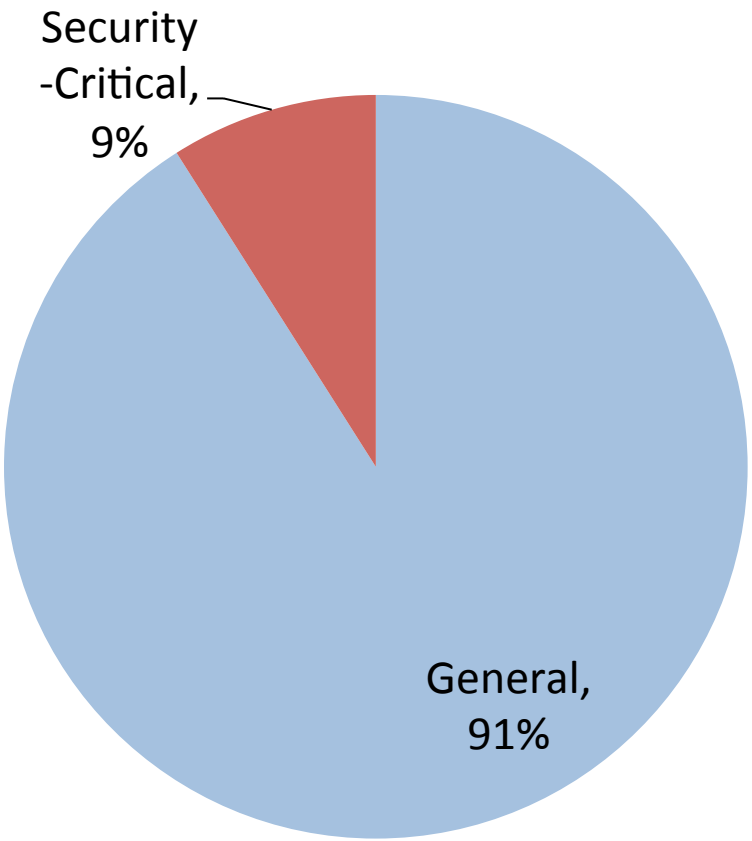
Security-critical errata impact privileged state or events



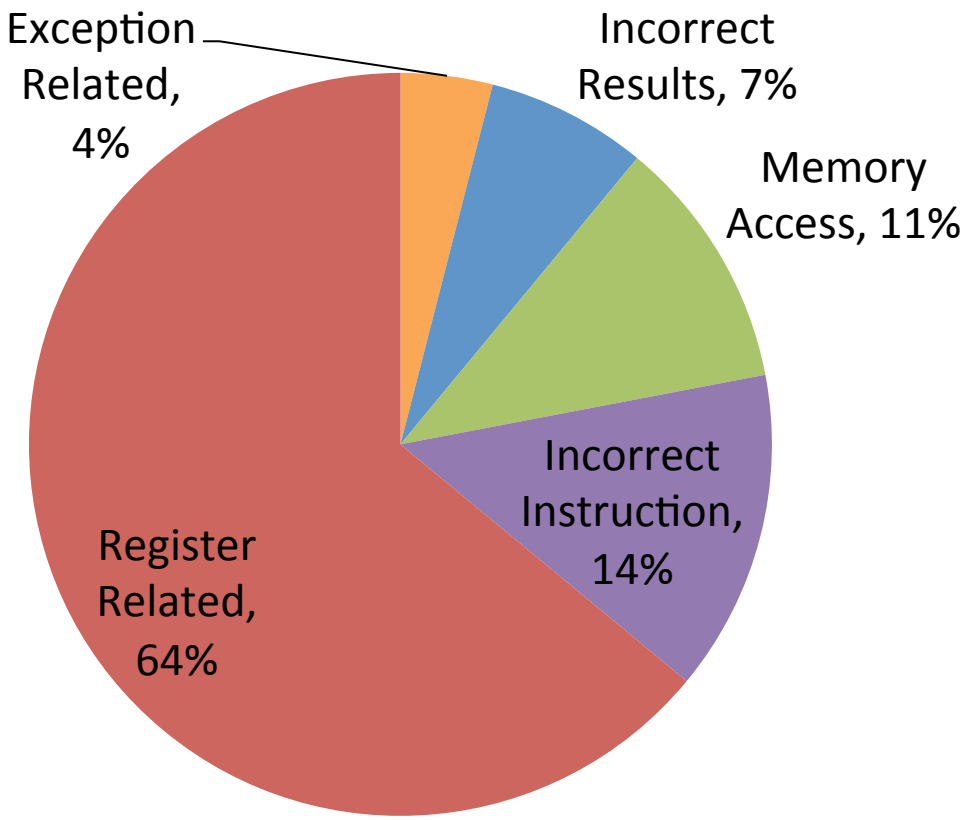
General Bug

Security-Critical Bug

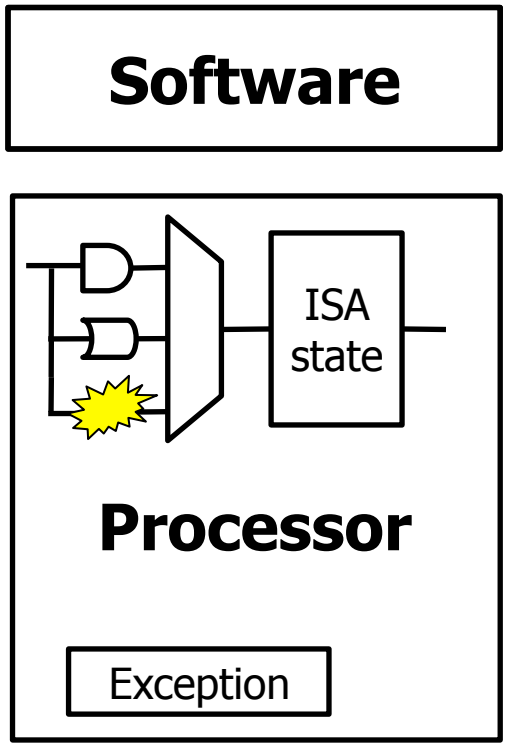
Security-critical errata exist in all inspected processors



Security-critical errata come from five effects-based classes

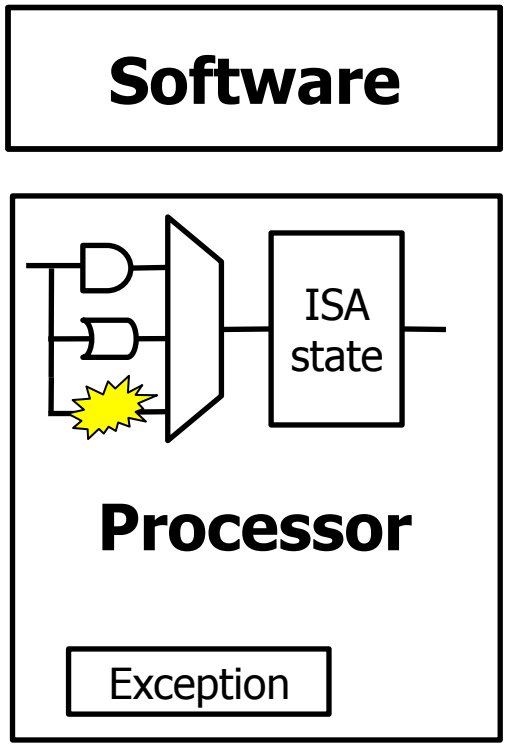


SPECS protects security-critical processor state



- 9% errata are security-critical
- All processors afflicted
- Come from 1 of 5 effects-based classes

SPECS protects security-critical processor state

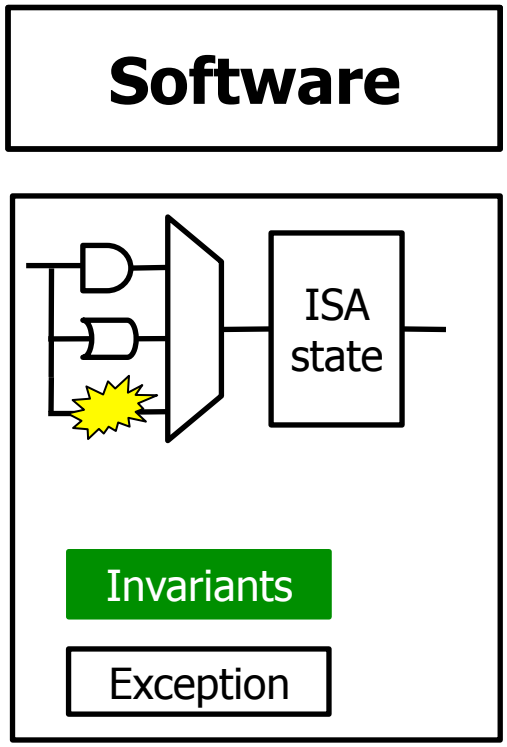


- 9% errata are security-critical
- All processors afflicted
- Come from 1 of 5 effects-based classes

Observation

1. Security-critical processor state is updated in a few, simple ways

SPECS protects security-critical processor state



- 9% errata are security-critical
- All processors afflicted
- Come from 1 of 5 effects-based classes

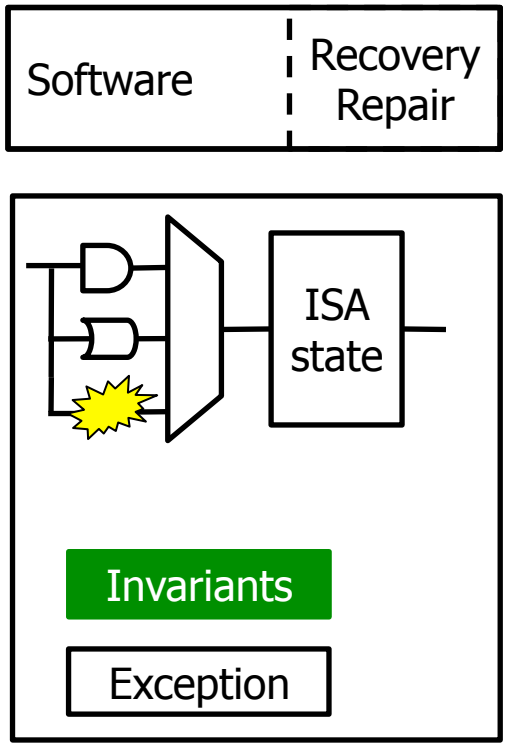
Observation

1. Security-critical processor state is updated in a few, simple ways

Implication

1. Security-critical state checkers are simple

SPECS dynamically verifies ISA state updates



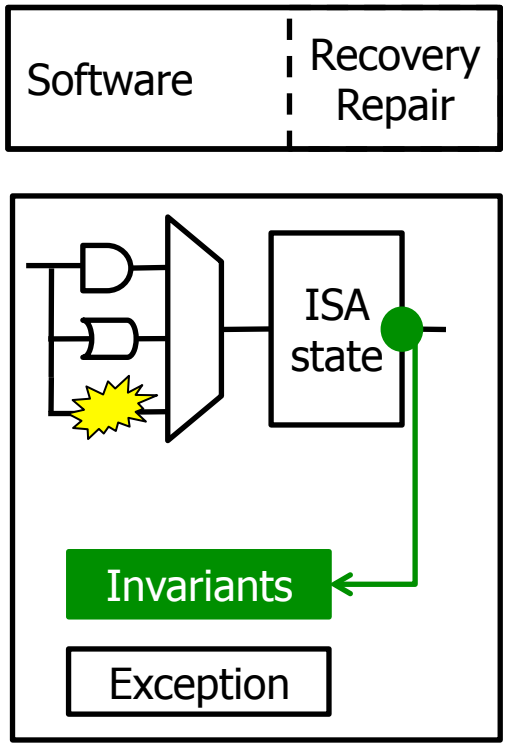
Observations

1. Security-critical processor state is updated in a few, simple ways
2. ISA-level state provides enough information for accurate detection

Implications

1. Security-critical state checkers are simple

SPECS dynamically verifies ISA state updates



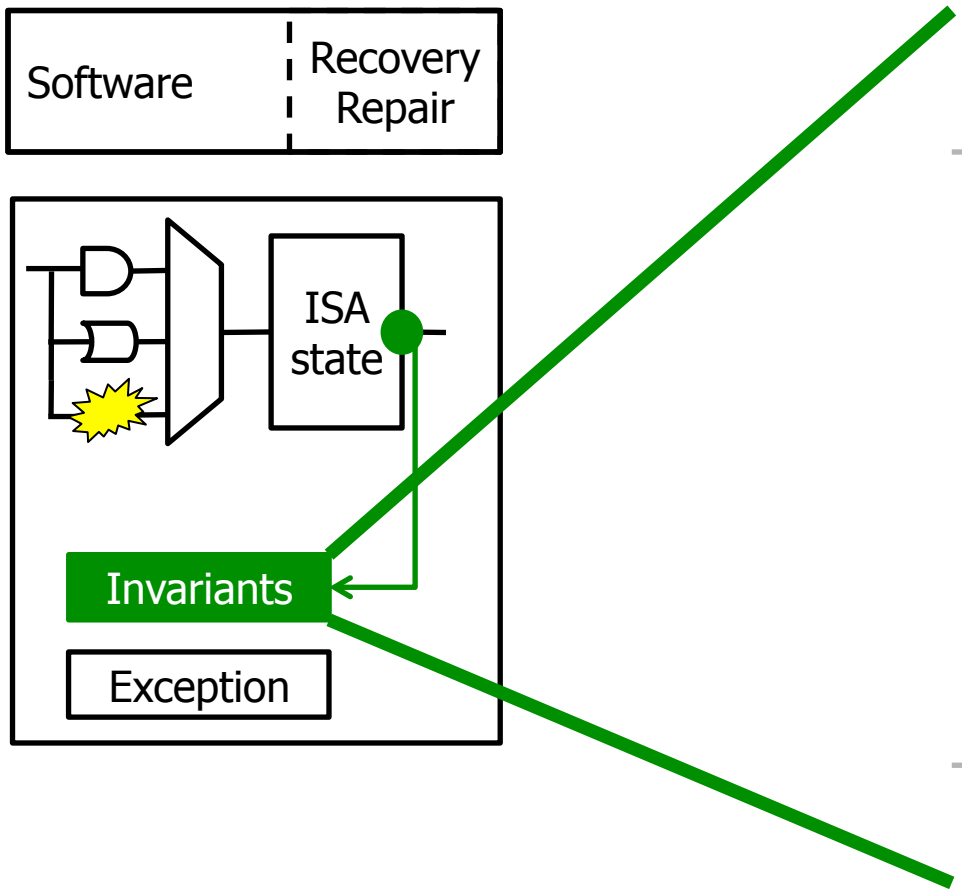
Observations

1. Security-critical processor state is updated in a few, simple ways
2. ISA-level state provides enough information for accurate detection

Implications

1. Security-critical state checkers are simple
2. Security-critical state checkers are understandable and portable

SPECS invariants are a composition of simple assertions on ISA-level state



ISA-level state

---

always

on edge

next

delta

& | ~ == !=

Example SPECS invariant

- No privilege escalation
  - If the processor goes from user to supervisor mode, then it must be due to a reset or exception/interrupt

```
// on reset?
on_edge(PRIV == SUPER, RST == 1) &

// on exception/interrupt?
(on_edge(PRIV == SUPER, PC & 0xFF == 0) |
on_edge(PRIV == SUPER, PC & 0xFFFFF000 == 0))
```

SPECS Invariants

1. Execution privilege matches page privilege
2. SPR = GPR in register move instructions
3. Updates to exception registers make sense
4. Destination matches the target
5. Memory value in = register value out
6. Register value in = memory value out
7. Memory address = effective address
8. Privilege escalates correctly
9. Privilege de-escalates correctly
10. Jumps update the PC correctly
11. Jumps update the LR correctly
12. Instruction is in a valid format
13. Continuous control flow
14. Exception return updates state correctly
15. Register change implies that it is the instruction target
16. SR is not written to a GPR in user mode
17. Interrupt/exception implies handled
18. Instruction not changed in the pipeline

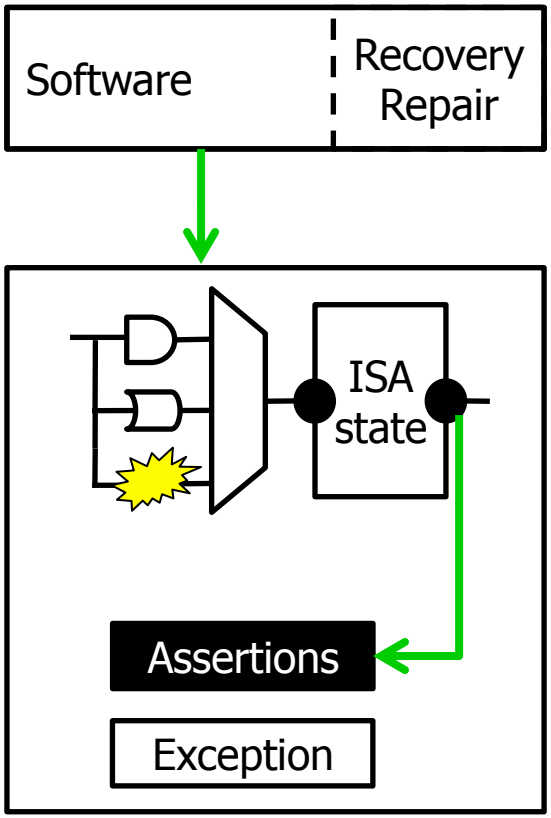
SPECS Invariants

1. Execution privilege matches page privilege
2. SPR = GPR in register move instructions
3. Updates to exception registers make sense
4. Destination matches the target
5. Memory value in = register value out
6. Register value in = memory value out
7. Memory address = effective address
8. Privilege escalates correctly
9. Privilege de-escalates correctly
10. Jumps update the PC correctly
11. Jumps update the LR correctly
12. Instruction is in a valid format
13. Continuous control flow
14. Exception return updates state correctly
15. Register change implies that it is the instruction target
16. SR is not written to a GPR in user mode
17. Interrupt/exception implies handled
18. Instruction not changed in the pipeline

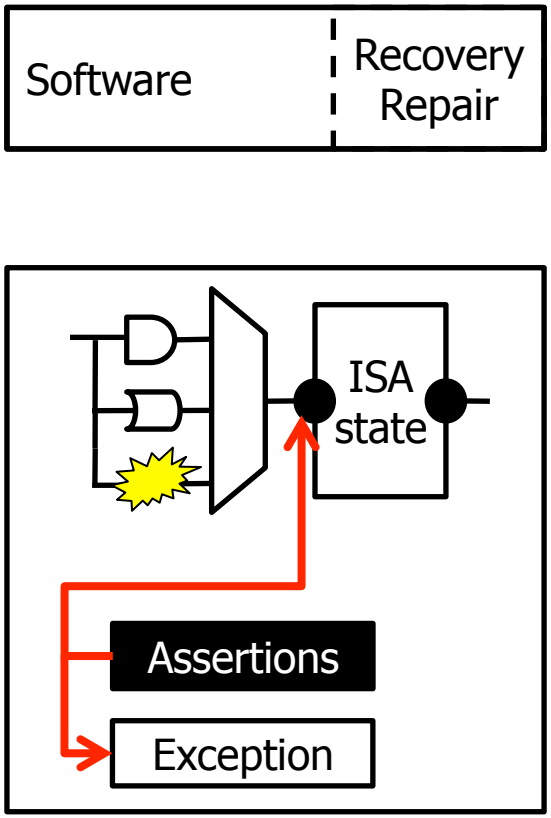
SPECS Invariants

1. Execution privilege matches page privilege
2. SPR = GPR in register move instructions
3. Updates to exception registers make sense
4. Destination matches the target
5. Memory value in = register value out
6. Register value in = memory value out
7. Memory address = effective address
8. Privilege escalates correctly
9. Privilege de-escalates correctly
10. Jumps update the PC correctly
11. Jumps update the LR correctly
12. Instruction is in a valid format
13. Continuous control flow
14. Exception return updates state correctly
15. Register change implies that it is the instruction target
16. SR is not written to a GPR in user mode
17. Interrupt/exception implies handled
18. Instruction not changed in the pipeline

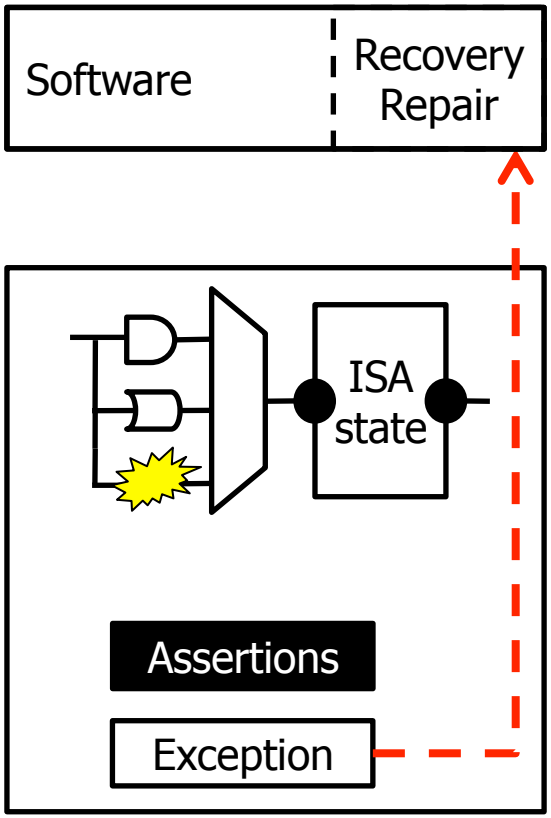
SPECS: dynamic invariant monitoring to create new introspection points



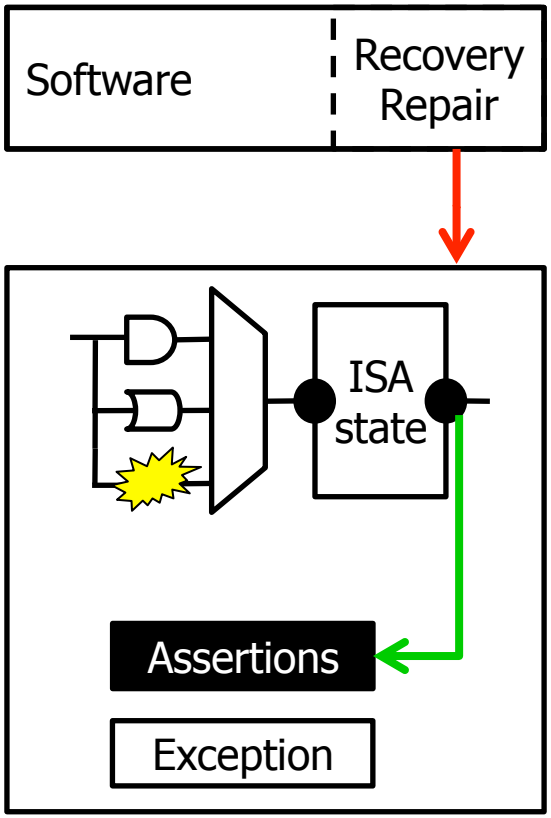
SPECS: dynamic invariant monitoring to create new introspection points



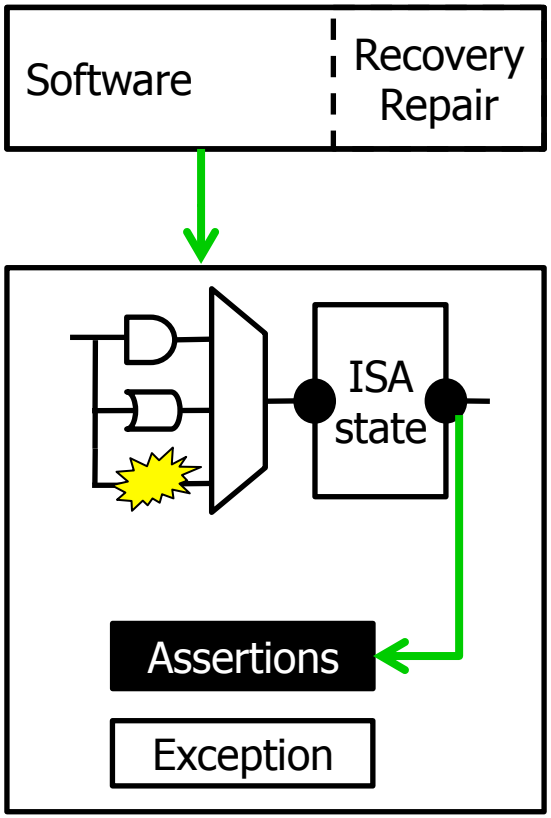
SPECS: dynamic invariant monitoring to create new introspection points



SPECS: dynamic invariant monitoring to create new introspection points



SPECS: dynamic invariant monitoring to create new introspection points

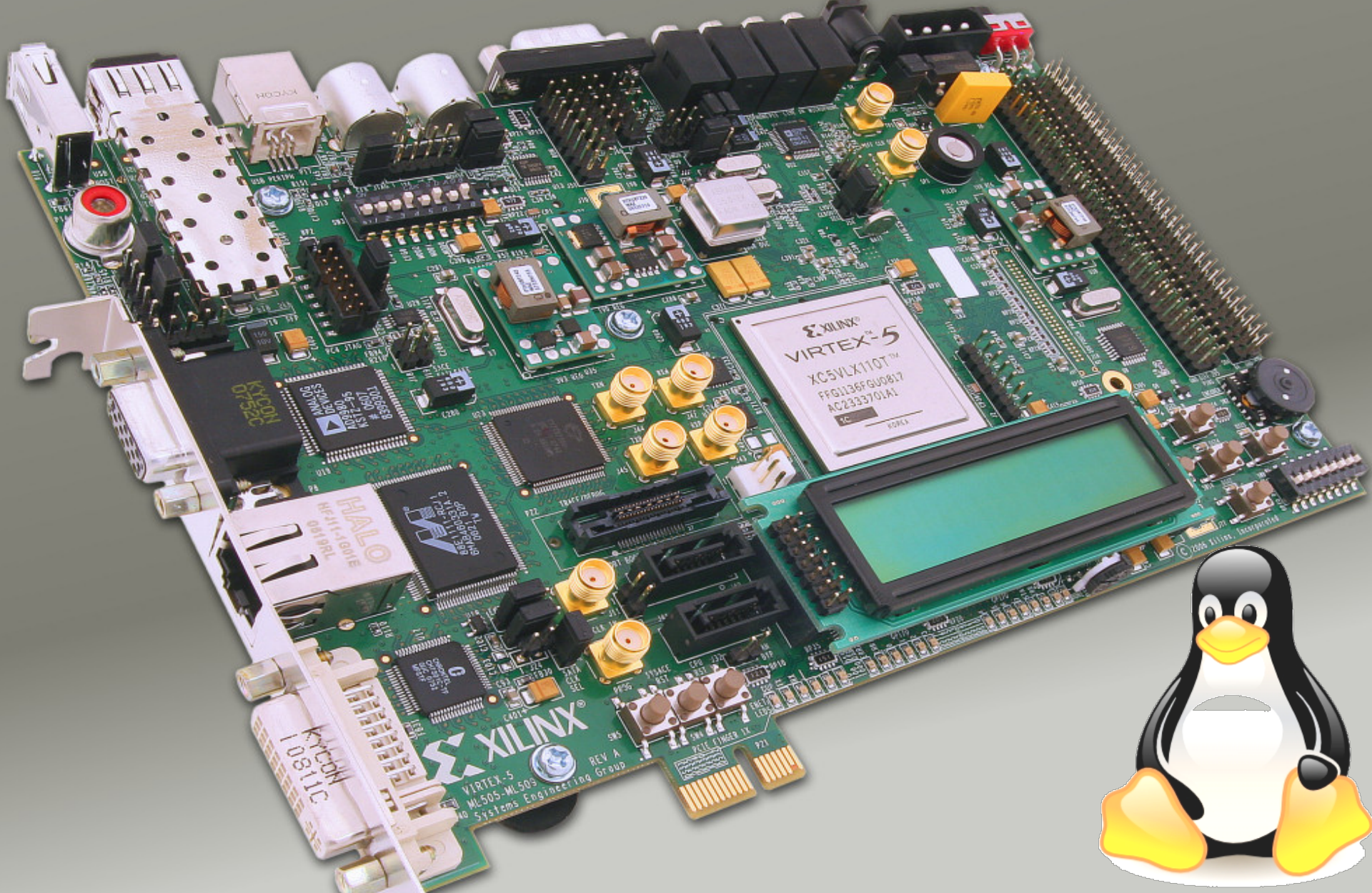


Design Challenges

- How to prevent bugs from altering ISA-level state in the event of a detection
- How to ensure that SPECS has a consistent view of ISA-level state
- What is the tradeoff between using only ISA-level state and detection precision

We evaluate SPECS using errata-based attacks

Bug ID	Synopsis	Class
1	Privilege escalation by direct access	IU
2	Privilege escalation by exception	IU
3	Privilege anti-de-escalation	IU
4	Register target redirection	IU
5	Register source redirection	IU
6	ROP by early kernel exit	IU
7	Disable interrupts by SR contamination	IU
8	EEAR contamination	IU
9	EPCR contamination on exception entry (from PC)	IU
10	EPCR contamination on exception exit (to PC)	IU
11	Code injection into kernel	EI
12	Selective function skip	EI
13	Register source redirection	IR
14	Disable interrupts via micro arch	XR



OpenRISC OR1200

SPECS is effective

Bug ID	Synopsis	Class	Detected
1	Privilege escalation by direct access	IU	✓
2	Privilege escalation by exception	IU	✓
3	Privilege anti-de-escalation	IU	✓
4	Register target redirection	IU	✓
5	Register source redirection	IU	✓
6	ROP by early kernel exit	IU	✓+
7	Disable interrupts by SR contamination	IU	✓
8	EEAR contamination	IU	✓
9	EPCR contamination on exception entry (from PC)	IU	✓
10	EPCR contamination on exception exit (to PC)	IU	✓
11	Code injection into kernel	EI	✓
12	Selective function skip	EI	✓+
13	Register source redirection	IR	✓
14	Disable interrupts via micro arch	XR	✓

SPECS is effective

Bug ID	Synopsis	Class	Detected
1	Privilege escalation by direct access	IU	✓
2	Privilege escalation by exception	IU	✓
3	Privilege anti-de-escalation	IU	✓
4	Register target redirection	IU	✓
5	Register source redirection	IU	✓
6	ROP by early kernel exit	IU	✓+
7	Disable interrupts by SR contamination	IU	✓
8	EEAR contamination	IU	✓
9	EPCR contamination on exception entry (from PC)	IU	✓
10	EPCR contamination on exception exit (to PC)	IU	✓
11	Code injection into kernel	EI	✓
12	Selective function skip	EI	✓+
13	Register source redirection	IR	✓
14	Disable interrupts via micro arch	XR	✓

SPECS aids recovery/repair software

Bug ID	Synopsis	Class	Detected	Pre-recovery Cleanup
1	Privilege escalation by direct access	IU	✓	Correction
2	Privilege escalation by exception	IU	✓	Correction
3	Privilege anti-de-escalation	IU	✓	Correction
4	Register target redirection	IU	✓	Correction
5	Register source redirection	IU	✓	None
6	ROP by early kernel exit	IU	✓+	Backtrack
7	Disable interrupts by SR contamination	IU	✓	None
8	EEAR contamination	IU	✓	Correction
9	EPCR contamination on exception entry (from PC)	IU	✓	Correction
10	EPCR contamination on exception exit (to PC)	IU	✓	None
11	Code injection into kernel	EI	✓	Backtrack
12	Selective function skip	EI	✓+	Backtrack
13	Register source redirection	IR	✓	None
14	Disable interrupts via micro arch	XR	✓	Backtrack

SPECS aids recovery/repair software

Bug ID	Synopsis	Class	Detected	Pre-recovery Cleanup
1	Privilege escalation by direct access	IU	✓	Correction
2	Privilege escalation by exception	IU	✓	Correction
3	Privilege anti-de-escalation	IU	✓	Correction
4	Register target redirection	IU	✓	Correction
5	Register source redirection	IU	✓	None
6	ROP by early kernel exit	IU	✓+	Backtrack
7	Disable interrupts by SR contamination	IU	✓	None
8	EEAR contamination	IU	✓	Correction
9	EPCR contamination on exception entry (from PC)	IU	✓	Correction
10	EPCR contamination on exception exit (to PC)	IU	✓	None
11	Code injection into kernel	EI	✓	Backtrack
12	Selective function skip	EI	✓+	Backtrack
13	Register source redirection	IR	✓	None
14	Disable interrupts via micro arch	XR	✓	Backtrack

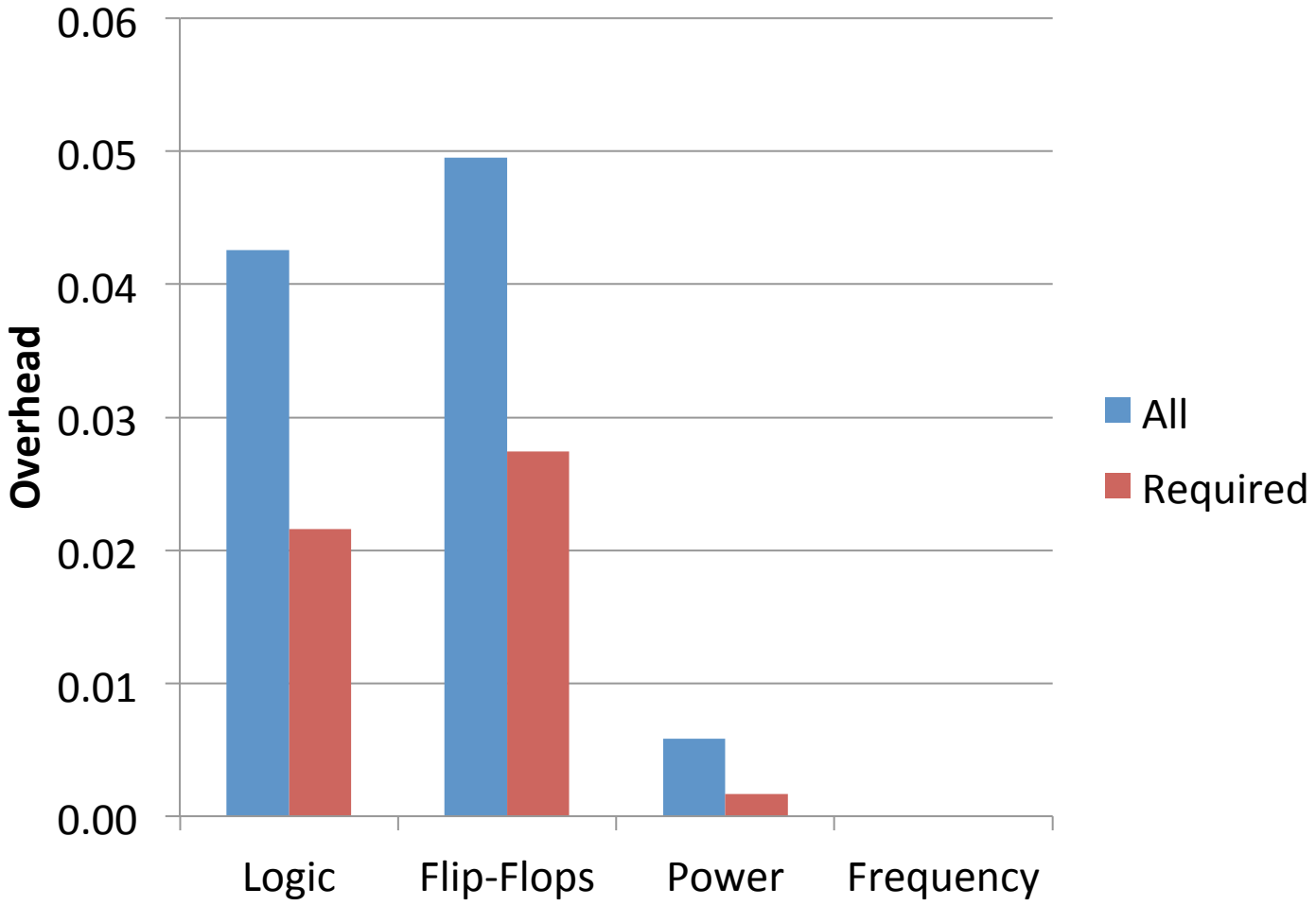
SPECS aids recovery/repair software

Bug ID	Synopsis	Class	Detected	Pre-recovery Cleanup
1	Privilege escalation by direct access	IU	✓	Correction
2	Privilege escalation by exception	IU	✓	Correction
3	Privilege anti-de-escalation	IU	✓	Correction
4	Register target redirection	IU	✓	Correction
5	Register source redirection	IU	✓	None
6	ROP by early kernel exit	IU	✓+	Backtrack
7	Disable interrupts by SR contamination	IU	✓	None
8	EEAR contamination	IU	✓	Correction
9	EPCR contamination on exception entry (from PC)	IU	✓	Correction
10	EPCR contamination on exception exit (to PC)	IU	✓	None
11	Code injection into kernel	EI	✓	Backtrack
12	Selective function skip	EI	✓+	Backtrack
13	Register source redirection	IR	✓	None
14	Disable interrupts via micro arch	XR	✓	Backtrack

SPECS aids recovery/repair software

Bug ID	Synopsis	Class	Detected	Pre-recovery Cleanup
1	Privilege escalation by direct access	IU	✓	Correction
2	Privilege escalation by exception	IU	✓	Correction
3	Privilege anti-de-escalation	IU	✓	Correction
4	Register target redirection	IU	✓	Correction
5	Register source redirection	IU	✓	None
6	ROP by early kernel exit	IU	✓+	Backtrack
7	Disable interrupts by SR contamination	IU	✓	None
8	EEAR contamination	IU	✓	Correction
9	EPCR contamination on exception entry (from PC)	IU	✓	Correction
10	EPCR contamination on exception exit (to PC)	IU	✓	None
11	Code injection into kernel	EI	✓	Backtrack
12	Selective function skip	EI	✓+	Backtrack
13	Register source redirection	IR	✓	None
14	Disable interrupts via micro arch	XR	✓	Backtrack

SPECS is efficient



more in the paper…

asplos242_hicks.pdf (page 1 of 13)

SPECS: A Lightweight Runtime Mechanism for Protecting Software from Security-Critical Processor Bugs

Matthew Hicks
University of Michigan
mdhicks@umich.edu

Cynthia Sturton
University of North Carolina at Chapel Hill
csturton@cs.unc.edu

Samuel T. King
Twitter, Inc.
sking@twitter.com

Jonathan M. Smith
University of Pennsylvania
jms@cis.upenn.edu

Abstract

Processor implementation errata remain a problem, and worse, a subset of these bugs are security-critical. We classified 7 years of errata from recent commercial processors to understand the magnitude and severity of this problem, and found that of 301 errata analyzed, 28 are security-critical.

We propose the SECURITY-CRITICAL PROCESSOR ERRATA CATCHING SYSTEM (SPECS) as a low-overhead solution to this problem. SPECS employs a dynamic verification strategy that is made lightweight by limiting protection to only security-critical processor state. As a proof-of-concept, we implement a hardware prototype of SPECS in an open source processor. Using this prototype, we evaluate SPECS against a set of 14 bugs inspired by the types of security-critical errata we discovered in the classification phase. The evaluation shows that SPECS is 86% effective as a defense when deployed using only ISA-level state; incurs less than 5% area and power overhead; and has no software run-time overhead.

Categories and Subject Descriptors C.0 [*General*]: Hardware/software interfaces; C.0 [*General*]: System architectures; K.6.5 [*Security and Protection*]: Invasive software (e.g., viruses, worms, Trojan horses)

Keywords Processor errata; hardware security exploits; security-critical processor errata

1. Introduction

Modern processors are imperfect. Processors are built from millions of lines of code [42], yielding chips with billions

	Catch All Bugs?	Catch Security-Critical Bugs?	Low Overhead?
SW-only	✗	✗	✓
HW-only	✓	✓	✗
SPECS	✗	✓	✓
SPECS+SW	✓	✓	✓

Table 1: The design space for catching processor bugs: existing software-only approaches are limited, but practical; existing hardware-only approaches are powerful, but impractical; and SPECS, combined with existing software approaches, is both powerful and practical.

hardware typically ships with bugs. For example, the errata document for Intel's Core 2 Duo processor family [20] lists 129 *known* bugs.

Some processor bugs have the potential to weaken the security of software by allowing unprivileged code to modify or control privileged processor state. We deem these bugs *security critical* (we do not claim that this set is unique). While some effects of bugs are detectable by software through periodic checks [8, 27] or special encodings [6, 40], other processor bugs affect software in a manner that software cannot detect practically. Such bugs are security-critical because they affect a critical subset of processor functionality trusted by the software tasked with detecting and recovering from imperfections. One example of a security-critical bug is a design flaw in the MIPS R4000 processor that allows user-mode code to control the location of an exception vector [26]. The effect of this bug is user-mode code that runs at

Targeted hardware detectors plus software is an effective and efficient approach for handling processor bugs

…but we still need reprogrammability to handle gaps in coverage

https://github.com/impedimentToProgress/specs