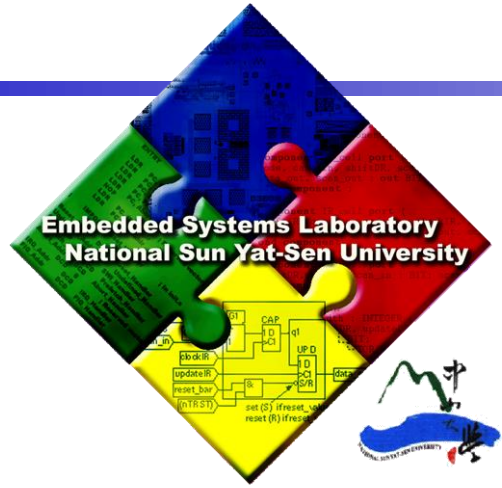


Embedded Systems Laboratory

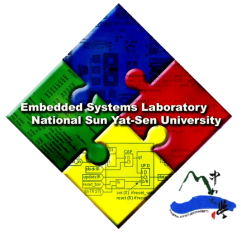
Department of Computer Science and Engineering
National Sun Yat-Sen University

Critical Signature Assertion and On-the-Fly Recovery for Control Flow Errors in Processors

Authors: Ing-Jer Huang, Yi-Ju Ke, Shih-Lung Pao

Presenter：Shih-lung Pao
Date：Monday, December 19, 2022

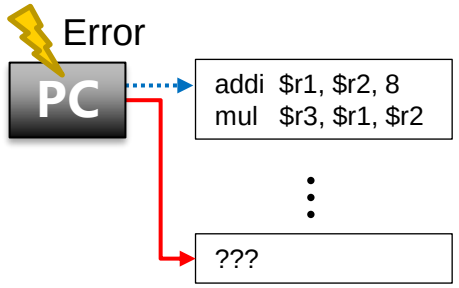
Outline



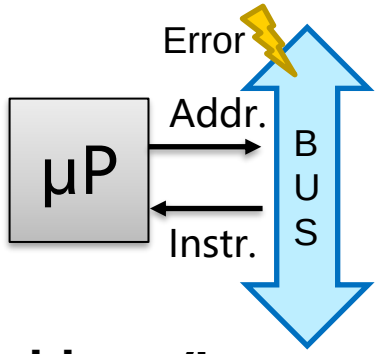
- **Why Control Flow Error Recovery?**
- **Proposed Method**
  - Critical Signature Assertion
  - On-the-Fly Recovery
- **Experiment**
- **Results**
- **Conclusion**
- **Future Work**

Why Control Flow Error (CFE) Recovery?

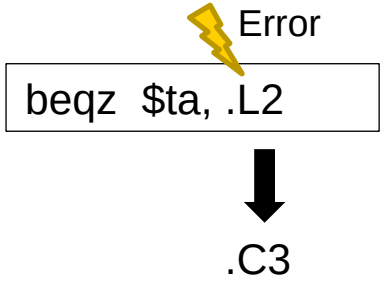
- **CFE: Unexpected sequence of instruction**
  - Usually caused by **Transient Fault**
- **Examples of CFE:**



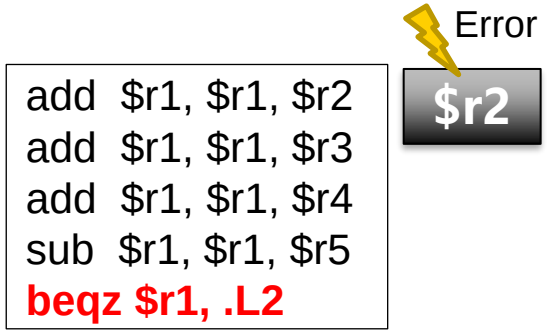
PC Error



Address/Instruction Bus Error



Instruction Memory Error

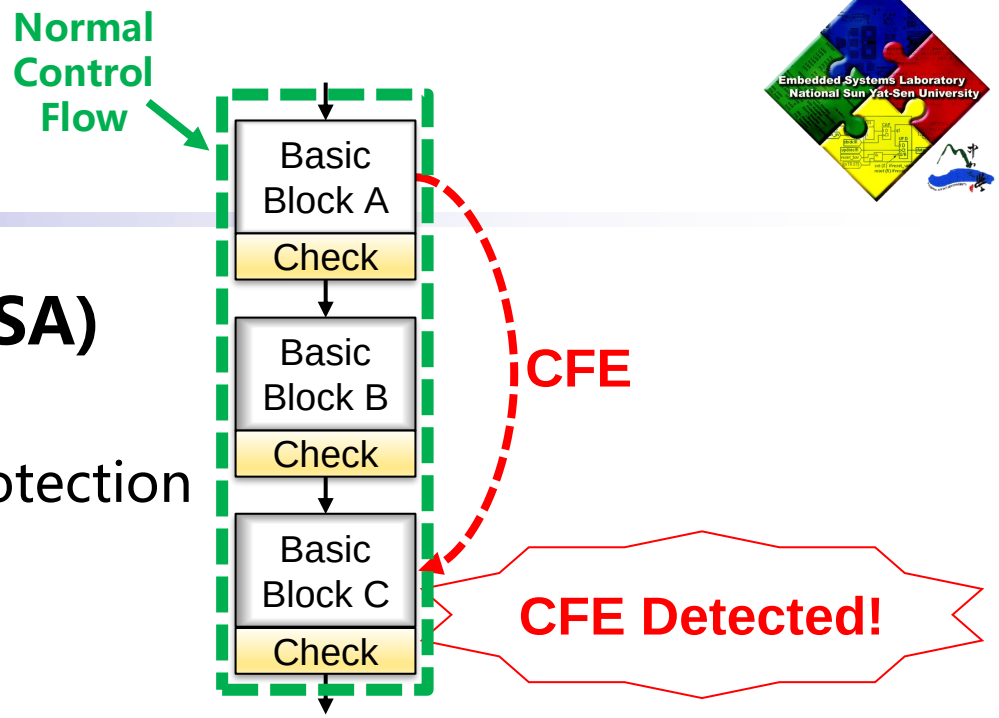


Incorrect Branch Led by Data Error

- **CFE: 33~77% of all hardware faults***
- **Reliability of safety critical systems**
  - Expensive systems (e.g., spaceship, airplane): **Hardware Redundancy**
  - Cost-sensitive systems (e.g., IoT, wearable device): **Detection & Recovery**

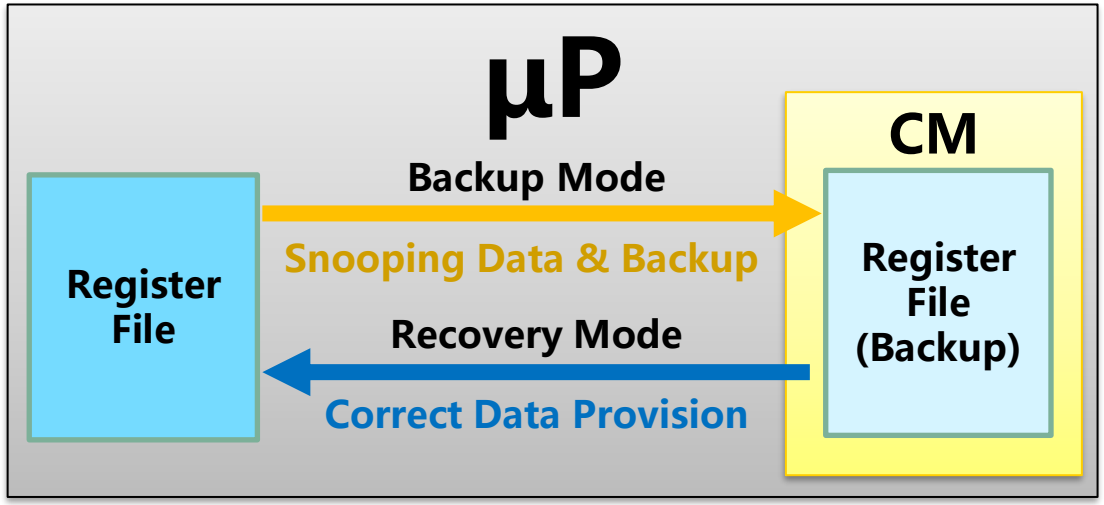
*Reference: [4, 5, 11, 12]

Proposed Method



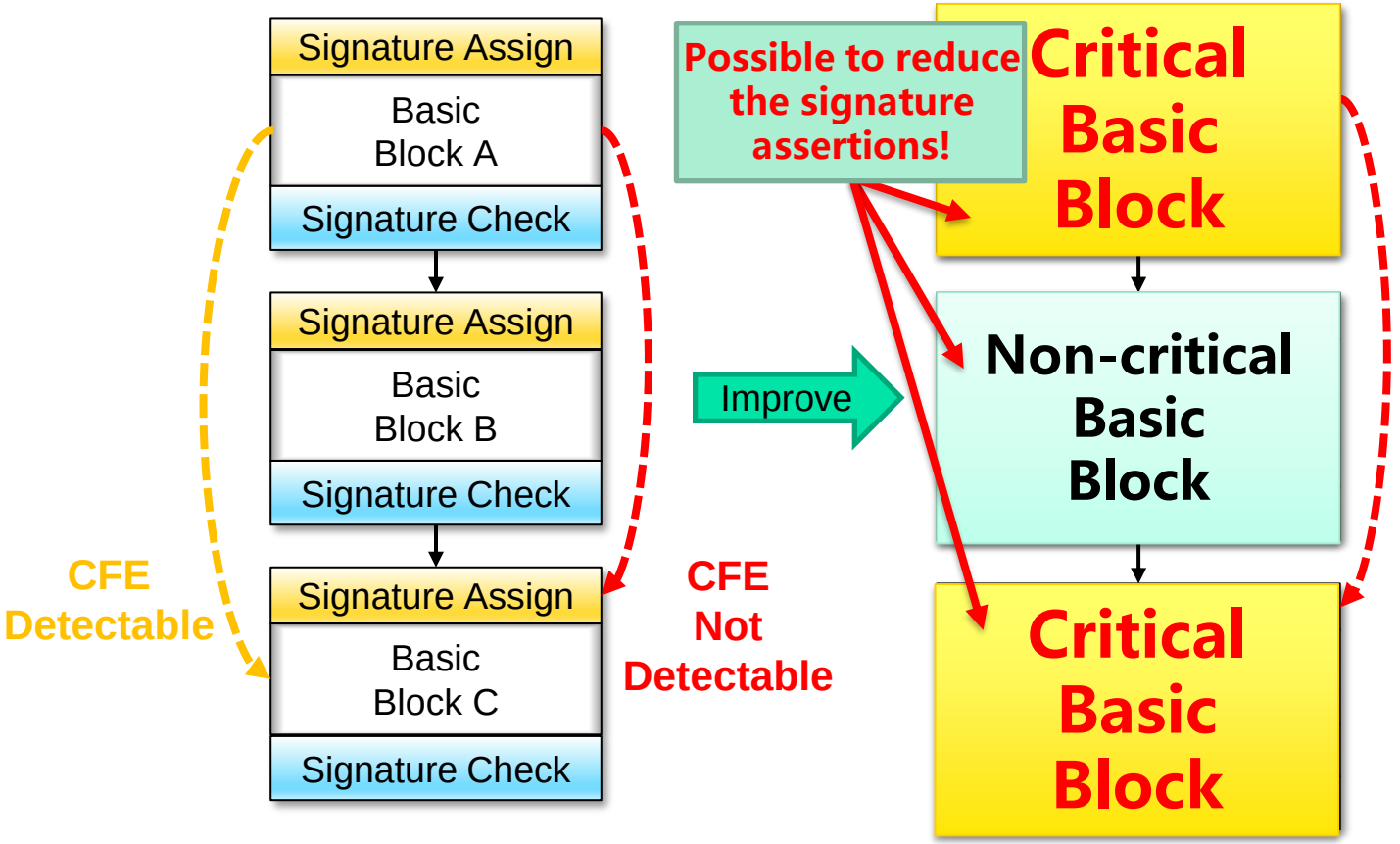
- **Detection: Critical Signature Assertion (CSA)**
  - Signature-based CFE detection
  - Minimized extra instructions with maximized protection

- **Recovery: On-the-Fly Recovery (OTFR)**
  - Hardware support recovery
  - Logging-based checkpointing
  - **Recovery by zero latency**



Critical Signature Assertion:
Concept of Signature Assertion

- To detect CFEs, we can assign signature assertions into Basic Blocks (BB)



Example: Signature Assertion

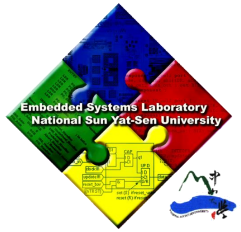
```
.L1:
    addi    $r1, $r2, 8
    mul     $r3, $r1, $r2
    sub     $r4, $r1, $r2
    lwi     $r0, [$sp+(4)]
    beqz    $ta, .L2
```

```
S = func(S, d1)
    addi    $r1, $r2, 8
    mul     $r3, $r1, $r2
    sub     $r4, $r1, $r2
    lwi     $r0, p$sp+(4)
S = func(S, d2)
If S != S_expect, goto ERR
Checkpoint
    beqz    $ta, .L2
```

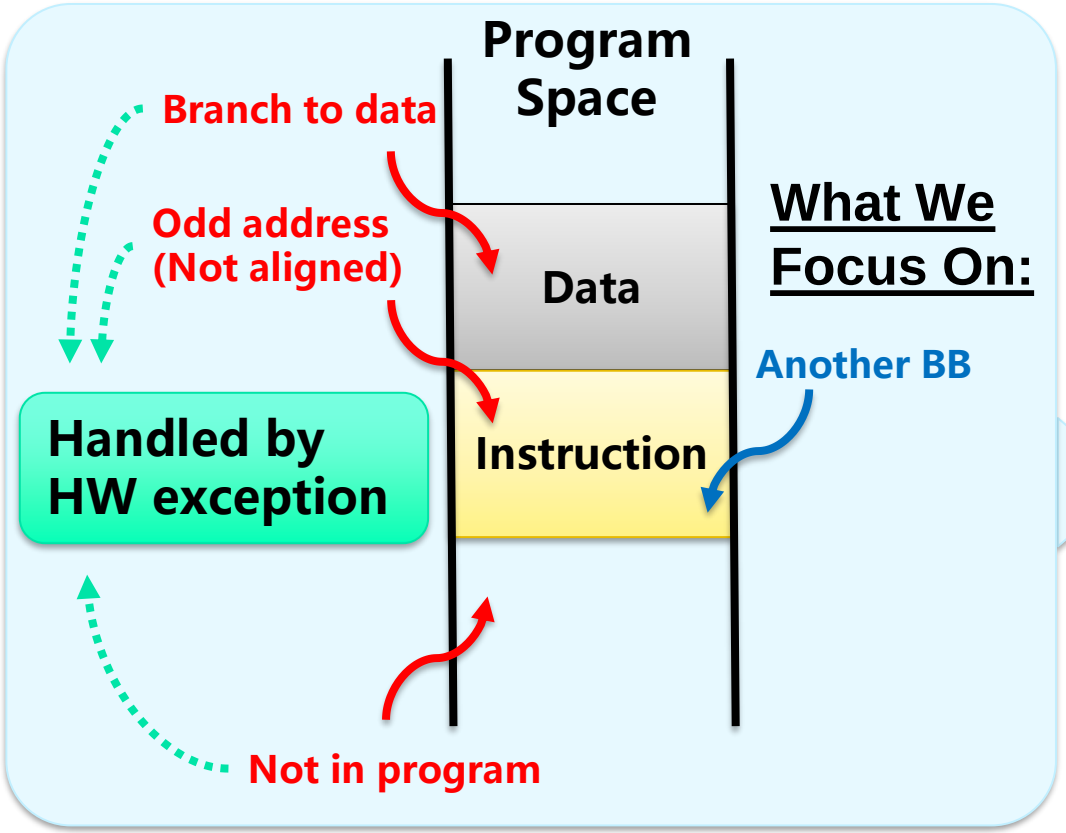
Extra Instruction for **Recovery**: Add checkpoint if nothing wrong

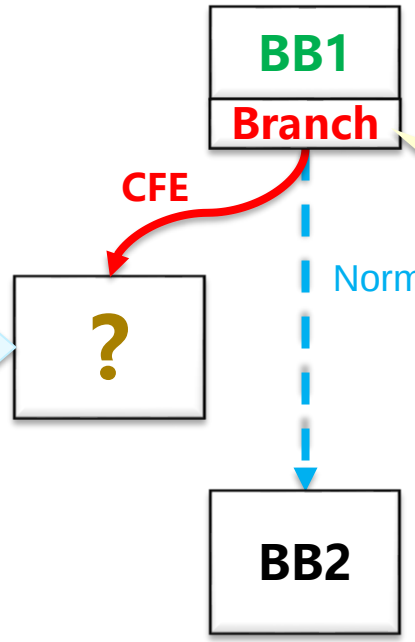
High Overhead!
4 extra instr. on every BB

Critical Signature Assertion:
Criticality Analysis



- **What positions that CFEs are more likely to occur?**

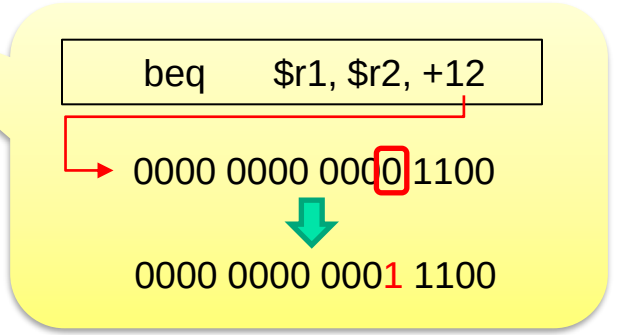




Bit-flip Analysis on Branch:

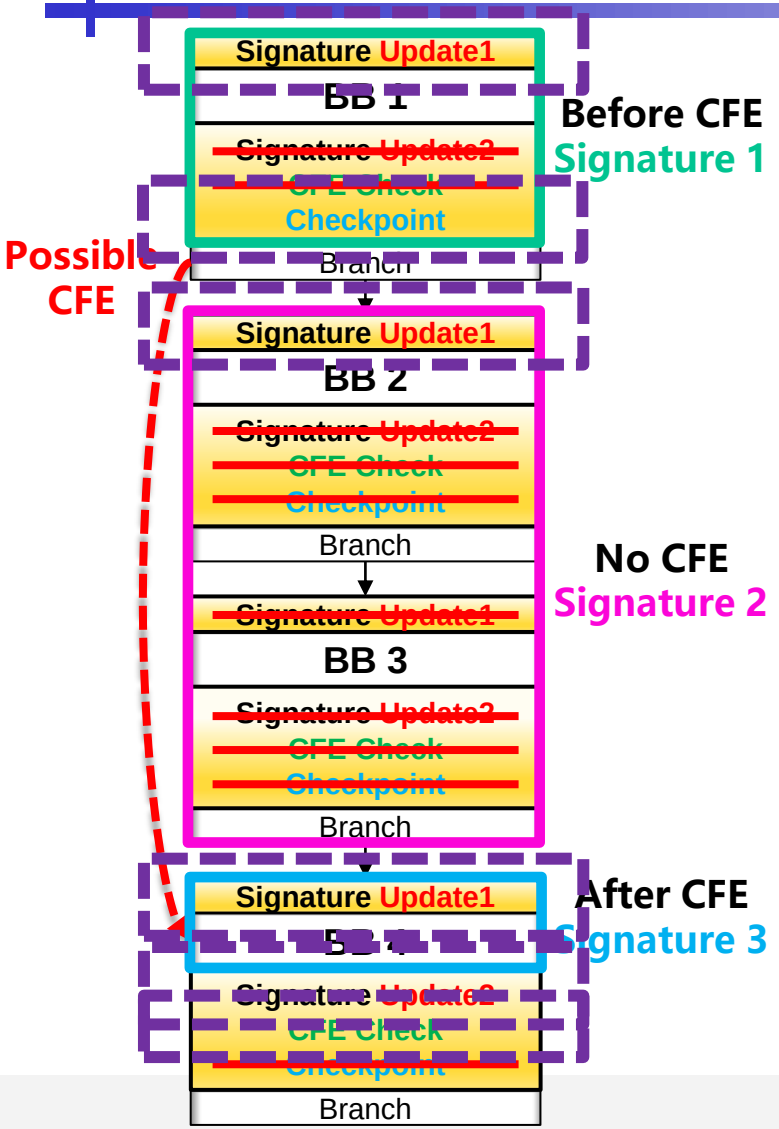
Focus on **branch target address**

1. Flip one bit in **address section**, check all possible **target positions**
2. Record: branches to another BB
3. Repeat step 1,2, and get a list of **Possible CFEs**



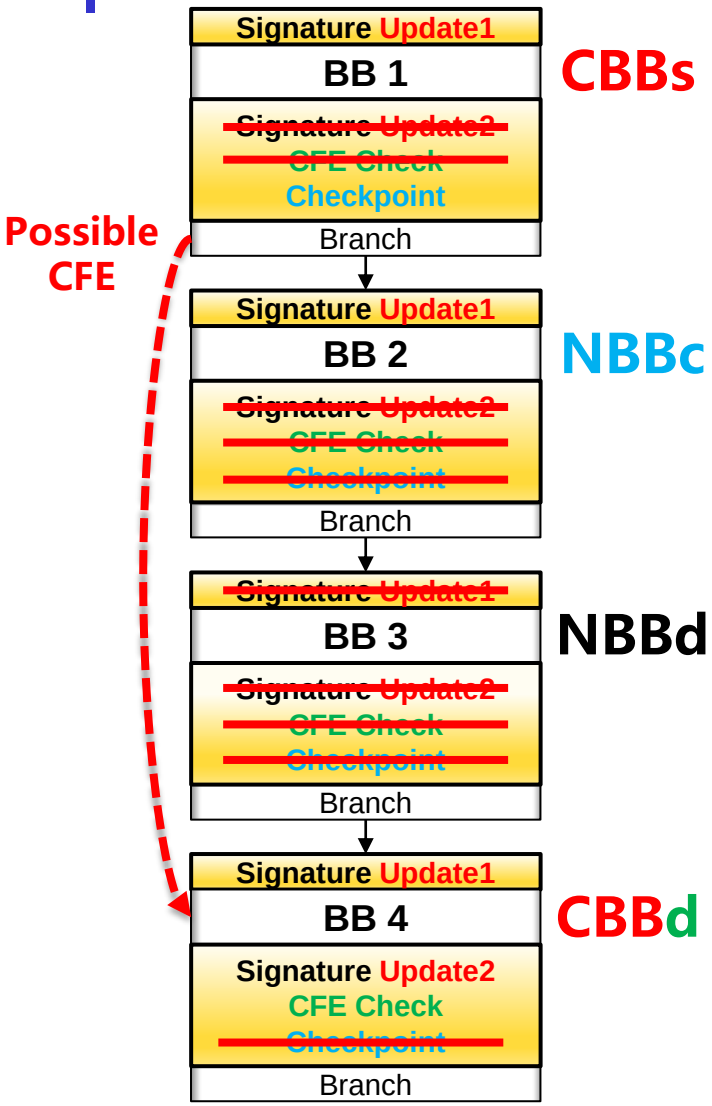
Criticality:
The chance of a CFE resulting in a jump to a wrong BB

Critical Signature Assertion:
Signature Assertion Optimization Based on Criticality



- **Possible CFE edge can be added into Control Flow Graph (CFG)**
  - According to the criticality
- **To detect CFE, the code sequence can be divided into 3 sections:**
  - **Before CFE / No CFE / After CFE**
  - **Assign unique signature for each section**
- **To reduce the overhead, only keep:**
  - Signature update at the **beginning of each section**
  - Checkpoint in **Before CFE** section for **recovery**
  - The CFE check instruction in **After CFE** section
  - "**Signature Update2**" in BB4, as the sign of ending the sequence normally
- **Remove all other instructions since no influence on the Possible CFE detection**

Critical Signature Assertion:
Rules of Signature Assertion Optimization



- **Conclude optimization rules referring to the type of BBs**

Table. Assertion reduction rules referring to types of BB

Types	Degree of criticality	Definition	Signature Assertion Instructions			
			UD1	UD2	CHK	Backup
1	CBBs	A **critical basic block (CBB)** which is the source of CFE	V	-	-	V
2	CBBd	CBB, destination of CFE	V	V	V	-
3	NBBc	non-CBB which is a child node of CBBs	V	-	-	-
4	NBBd	not case1~case3	-	-	-	-

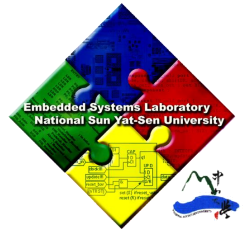
- **Huge reduction on code size**

  Original: 16 Extra Instr. ⟶ **New: 6 Extra Instr.**

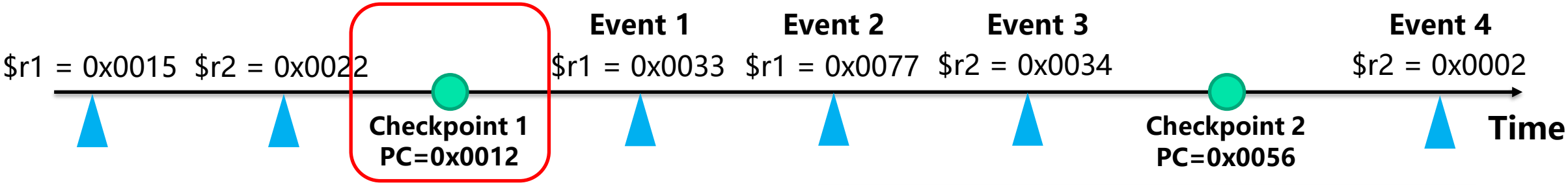
  Reduce > 70%

- **By the scheme, only one checkpoint is needed**
  - **Checkpoint only before a possible CFE**
  - **HW support checkpoint/recovery is possible!**

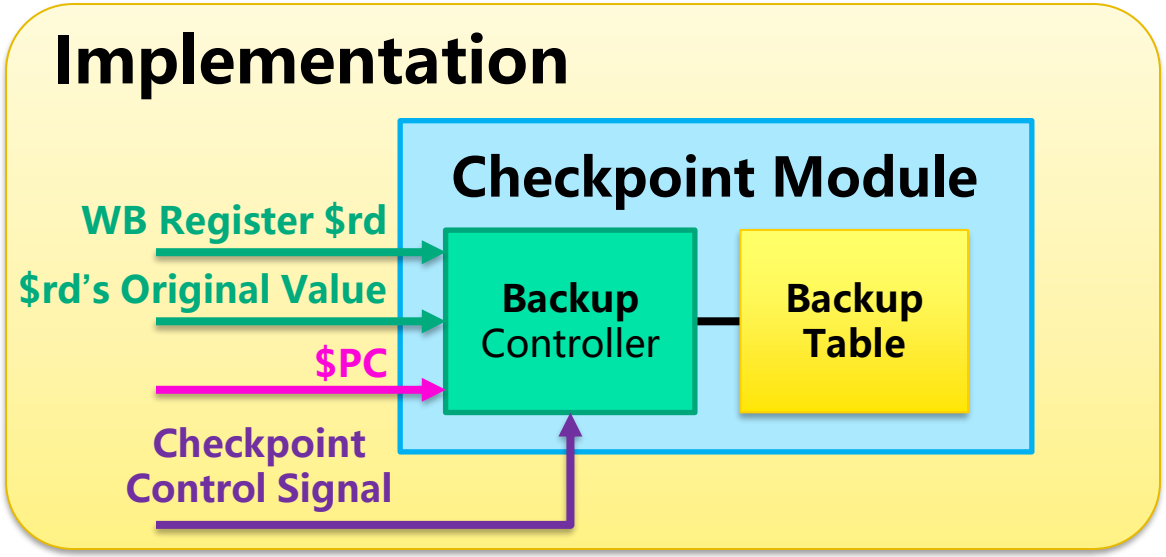
On-the-Fly Recovery
Before Recovery, How to Checkpoint?



- **Logging-based Checkpointing**
  - Record the original value right before the register is written
  - Example: single checkpointing to a register file with 2 registers: $r1, $r2

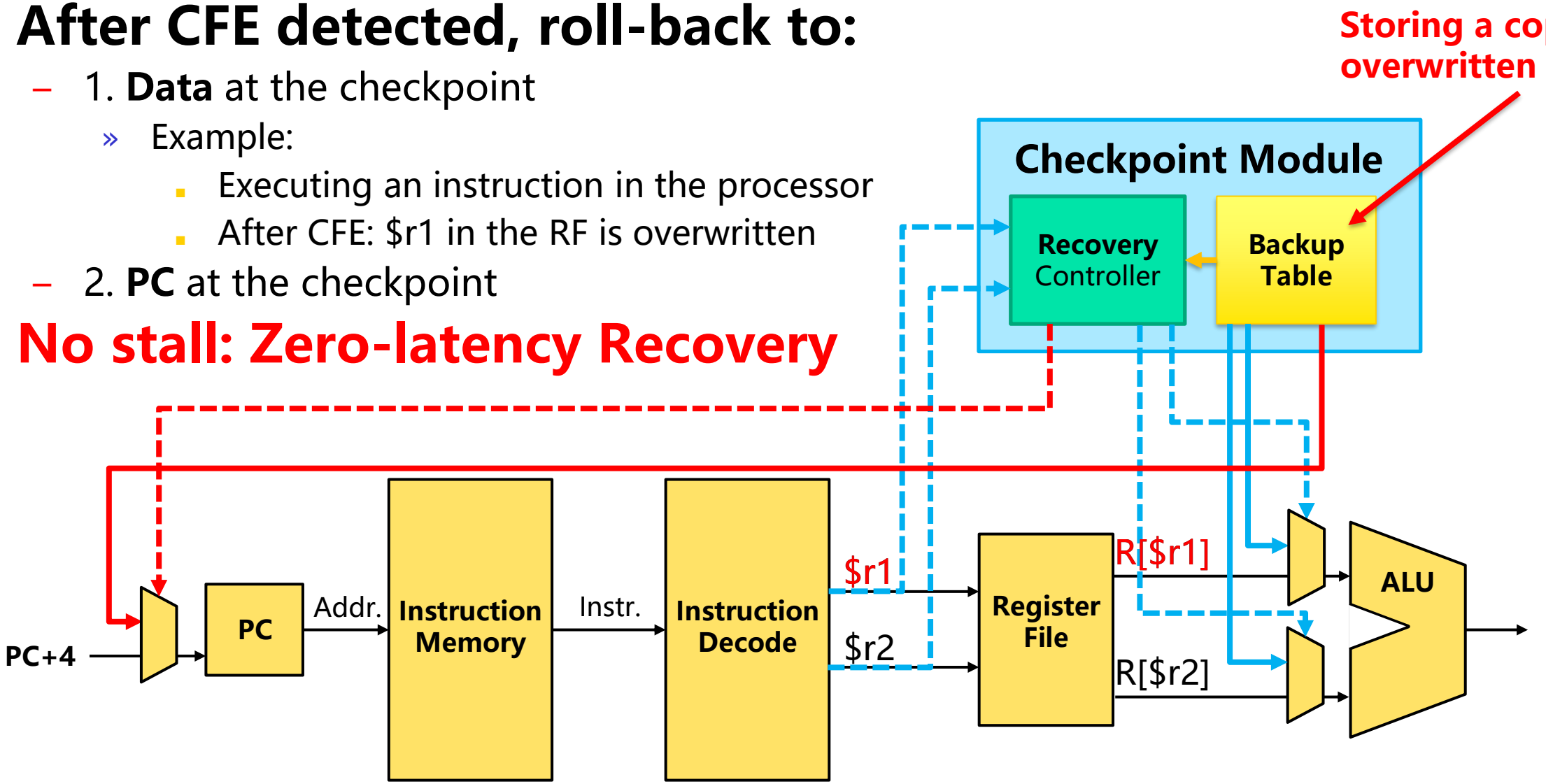


Backup Table	Register	Backup	Value
	$r1	0	0x0015
	$r2	1	0x0034
Time: Event 4	$PC	1	0x0056

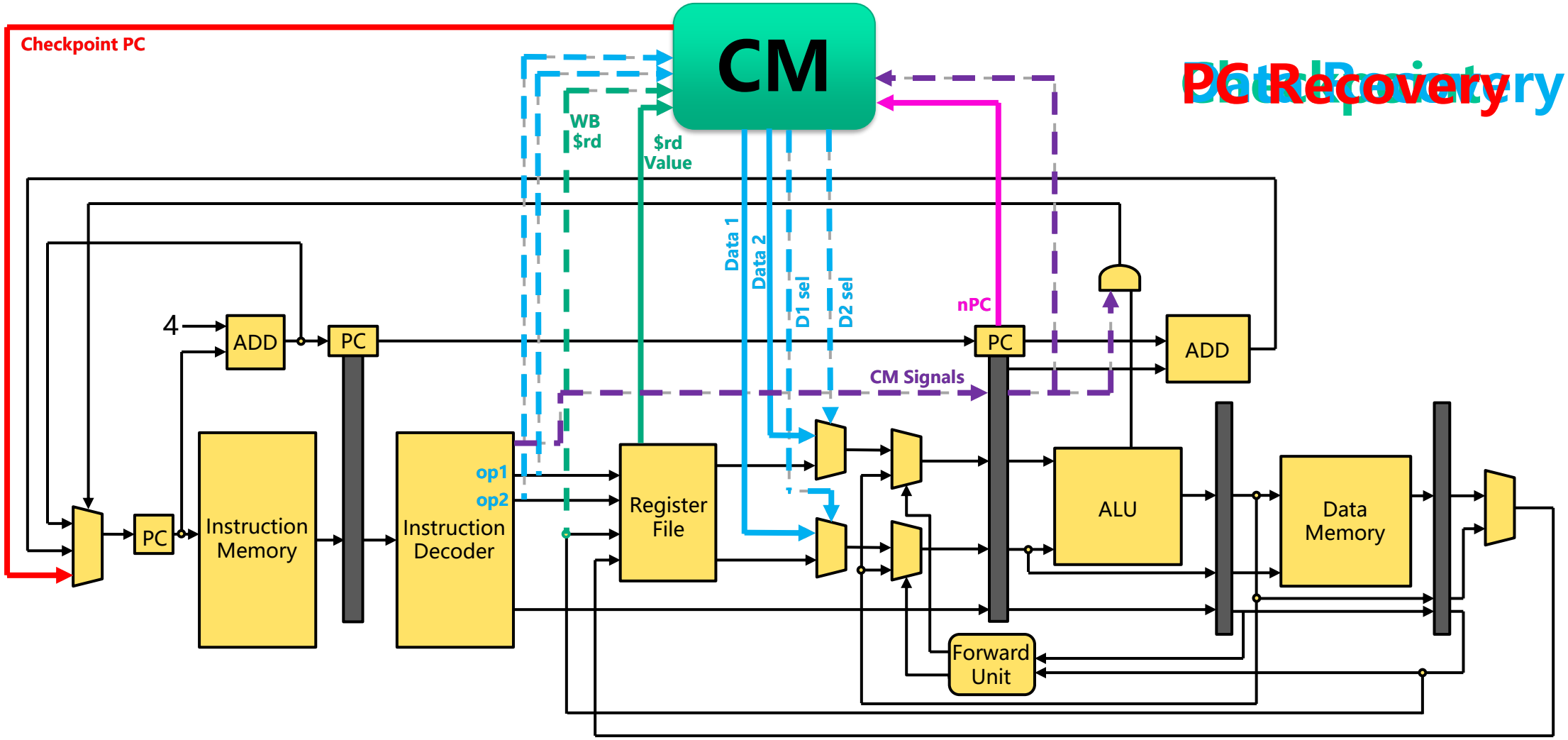


On-the-Fly Recovery
Recovery: After CFE Detected

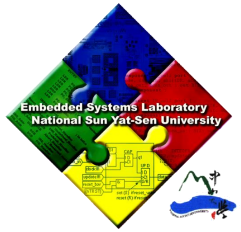
- **After CFE detected, roll-back to:**
  - 1. **Data** at the checkpoint
    - » Example:
      - Executing an instruction in the processor
      - After CFE: $r1 in the RF is overwritten
  - 2. **PC** at the checkpoint
- **No stall: Zero-latency Recovery**



On-the-Fly Recovery
OTFR with Typical 5-Stage Pipeline Processor



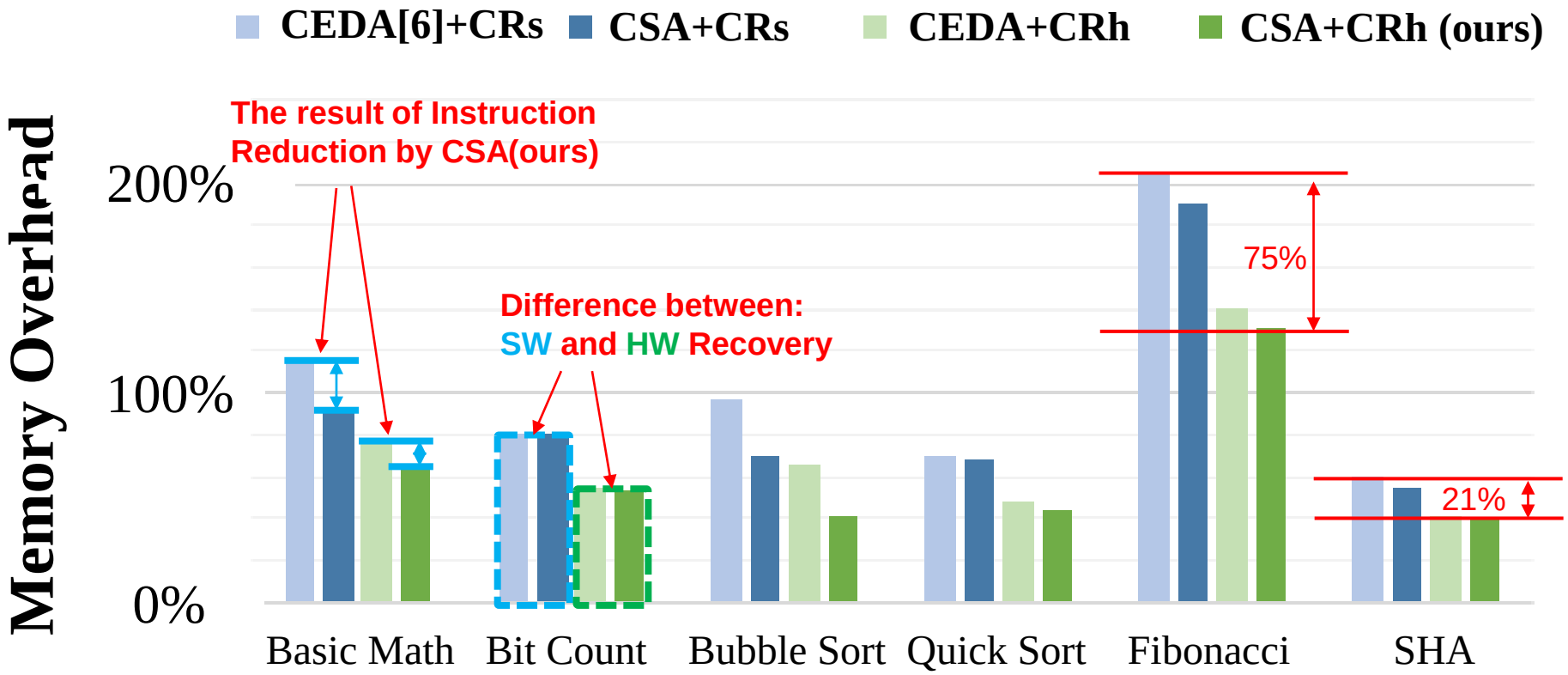
Experiment



- **Hardware platform**
  - AndesCore™ N801s (Commercial Processor from Andes)
    - 32-bit instruction set architecture
    - 3 stage pipeline microprocessor
- **Software benchmark**
  - MiBench
    - Standard benchmark for embedded systems
    - 6 programs from MiBench: *Basic Math, Bit Count, Bubble Sort, Quick Sort, Fibonacci, SHA*
- **Experiment setup**
  - **Transient faults** are **randomly injected** into each program, simulated **1000 times**
- **Comparison**
  - Detection method: **CEDA[5]** vs. Critical Signature Assertion, **CSA** (ours)
  - Recovery method: **CRs** vs. **CRh**
    - **CRs**: Checkpoint recovery method, software implementation
      - **Checkpoint** and **Recovery instructions** are implemented as subroutines
    - **CRh**: Checkpoint recovery method, proposed hardware architecture (OTFR)

Experiment Results
Memory Overhead

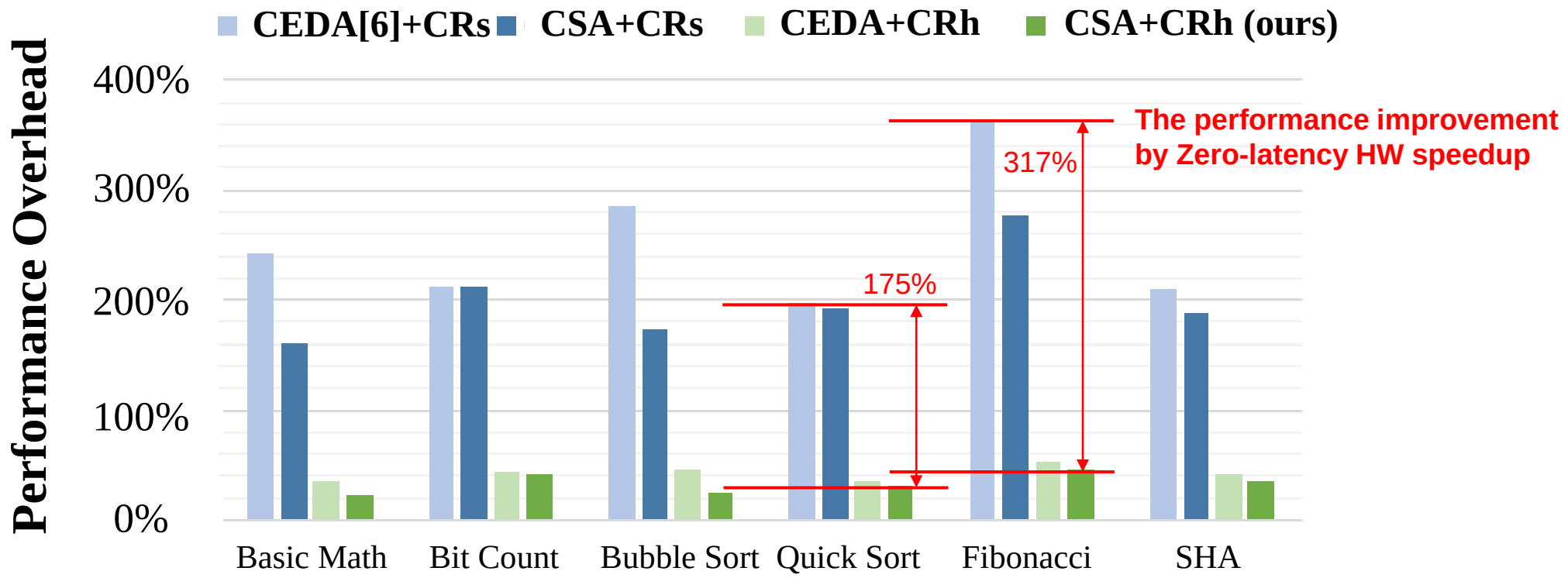
- **The CSA+CRh (ours) has 21%~75% less memory overhead compared with CEDA[6]+CRs (previous work)**



CRs: Checkpoint recovery, software implementation
CRh: Checkpoint recovery, proposed hardware architecture (OTFR)

Experiment Results
Performance Overhead

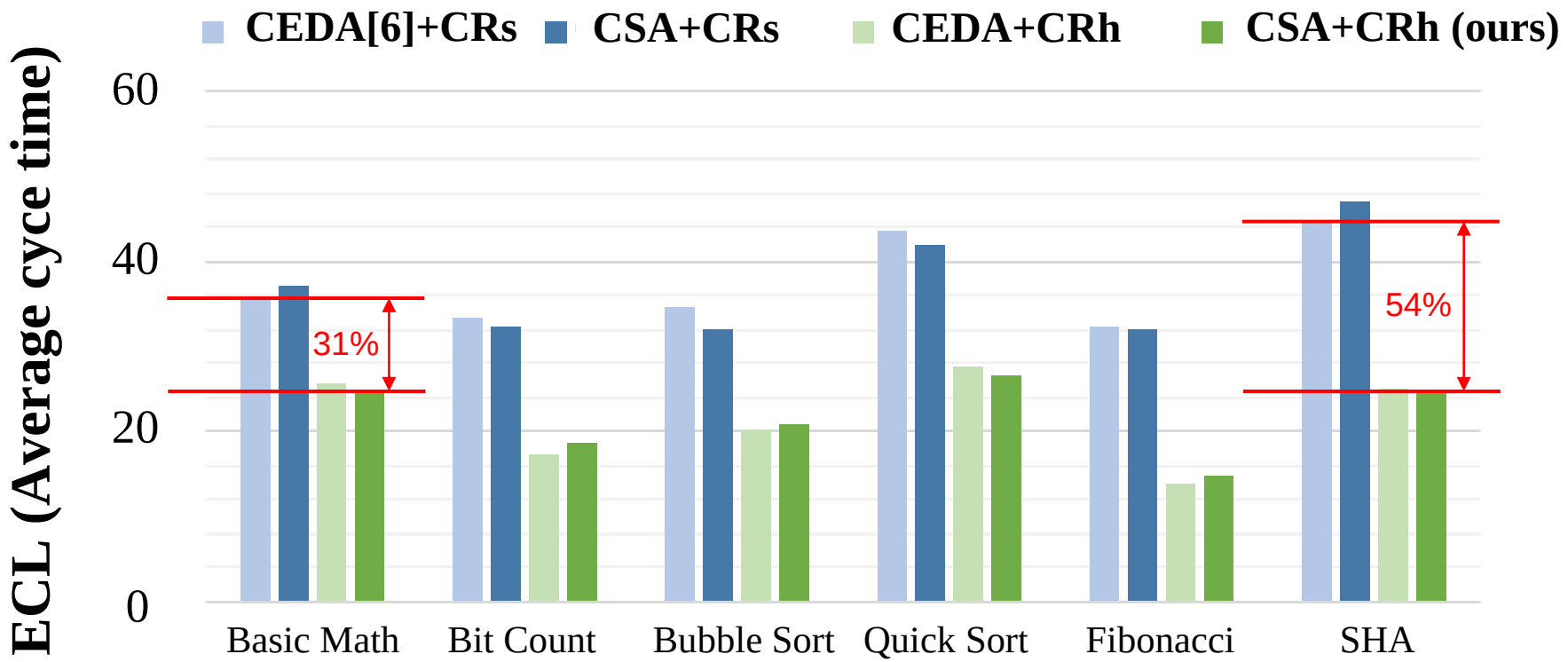
- **The CSA+CRh (ours) has 175%~317% less performance overhead compared with CEDA[6]+CRs (previous work)**



CRs: Checkpoint recovery, software implementation
CRh: Checkpoint recovery, proposed hardware architecture (OTFR)

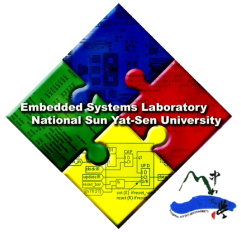
Experiment Results
Error Correction Latency

- **The CSA+CRh (ours) has 31%~54% less error correction latency (ECL) compared with CEDA[6]+CRs (previous work)**



CRs: Checkpoint recovery, software implementation
CRh: Checkpoint recovery, proposed hardware architecture (OTFR)

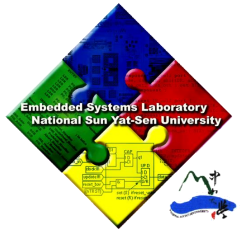
Experiment Results
Fault Coverage



- **The fault coverage decrease of the CSA** (ours) **is only 0.3% compared with CEDA[5]** (previous work)

Benchmark	Faults: CEDA [5] / CSA (ours)				Fault coverage (%)
	Masked	Hardware exception	Detected	Undetected	
Basic math	719 / 750	166 / 111	103 / 123	12 / 16	98.8% / 98.4%
Bit count	763 / 758	61 / 73	147 / 140	29 / 29	97.1% / 97.1%
Bubble Sort	758 / 796	133 / 127	101 / 68	8 / 9	99.2% / 99.1%
Quick sort	770 / 765	121 / 110	86 / 100	23 / 25	97.7% / 97.5%
Fibonacci	738 / 734	113 / 115	95 / 91	54 / 60	94.6% / 94.0%
Sha	782 / 776	74 / 90	118 / 108	26 / 26	97.4% / 97.4%
Geometric average				**Only 0.3% difference**	97.5% / 97.2%

Experiment Results
Area and Power Overhead

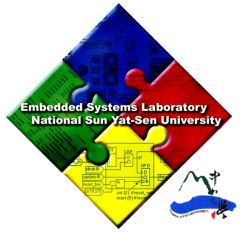


- **The proposed hardware architecture (OTFR) requires additional 3470 gates (+19%) and 967uW (+17%) power consumption**

HW Components	Area (um^2) / gate count	Power (uW)
Andes N801s Processor	142097.69 / 17838.15	5621.5
Andes N801s Processor + OTFR	169739.09 / 21308.10	6588.6
Overhead (%)	19.45% / 19.45% (+3470 gates)	17.2% (+967uW)

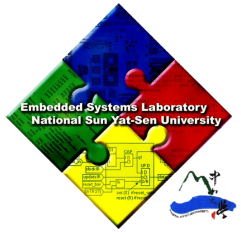
Synthesized with SYNOPSYS 90nm Process

Conclusion



- **This study presents a highly effective hybrid (SW + HW) CFE detection and recovery mechanism.**
- **The mechanism consists of two innovations:**
  - **Critical Signature Assertion**
  - **On-the-Fly Recovery**
- **Commercial platform verified**
  - Andes 32-bit microprocessor N801s.
- **Compared with the previous work, our approach:**
  - Memory overhead: **- 75%**
  - Performance overhead: **- 317%**
  - Error correction latency : **- 54%**
- **HW overhead:**
  - Gate count: +3470 gates (+19%)
  - Power: +967uW (+17%)
- **Fault coverage:** only **-0.3%**

Future Work



- **1. CFE protection on fault-tolerant instructions**
  - The CFE that jumps to fault-tolerant instructions
    - E.g., checkpoint instruction

- **2. Recovery on the data error in memory caused by CFE**
  - Before CFE detected:
    - Situation 1: Data within the memory might be modified
    - Situation 2: Data might be written back to the memory

Reference

[1] AndesCore™ N801-S, available at: http://www.andestech.com/product-details01.php?cls=3&id=4

[2] Zeyad Alkhalifa, VS Sukumaran Nair, Narayanan Krishnamurthy, and Jacob A. Abraham. 1999. Design and evaluation of system-level checks for on-line control flow error detection. IEEE Transactions on Parallel and Distributed Systems 10, 6 (1999), 627–641.

[3] Nahmsuk Oh, Philip P Shirvani, and Edward J McCluskey. 2002. Control-flow checking by software signatures. IEEE transactions on Reliability 51, 1 (2002), 111–122.

[4] Aiguo Li and Bingrong Hong. 2010. On-line control flow error detection using relationship signatures among basic blocks. Computers & electrical engineering 36, 1 (2010), 132–141.

[5] Ramtilak Vemu and Jacob Abraham. 2011. CEDA: Control-flow error detection using assertions. IEEE Trans. Comput. 60, 9 (2011), 1233–1245.

[6] José Rodrigo Azambuja, Ângelo Lapolli, Lucas Rosa, and Fernanda Lima Kastens- midt. 2011. Detecting SEEs in microprocessors through a non-intrusive hybrid technique. IEEE Transactions on Nuclear Science 58, 3 (2011), 993–1000.

[7] Luis Parra, Almudena Lindoso, Marta Portela, Luis Entrena, Felipe Restrepo-Calle, Sergio Cuenca-Asensi, and Antonio Martínez-Álvarez. 2014. Efficient mitigation of data and control flow errors in microprocessors. IEEE Transactions on Nuclear Science 61, 4 (2014), 1590–1596.

[8] Roshan G Ragel and Sri Parameswaran. 2006. IMPRES: integrated monitoring for processor reliability and security. In Proceedings of the 43rd annual Design Automation Conference. ACM, 502–505.

[9] Ghazaleh Nazarian, Diego G Rodrigues, Alvaro Moreira, Luigi Carro, and Georgi N Gaydadjiev. 2015. Bit-flip aware control-flow error detection. In Parallel, Distributed and Network-Based Processing (PDP), 2015 23rd Euromicro International Conference on. IEEE, 215–221.

[10] Ghazaleh Nazarian, Razvan Nane, and Georgi N Gaydadjiev. 2015. Low-cost Software Control-Flow Error Recovery. In Digital System Design (DSD), 2015 Euromicro Conference on. IEEE, 510–517.

[11] Lanfang Tan, Ying Tan, and Jianjun Xu. 2013. CFEDR: Control-flow error de- tection and recovery using encoded signatures monitoring. In Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFT), 2013 IEEE International Symposium on. IEEE, 25–32.

[12] Ramtilak Vemu, Sankar Gurumurthy, and Jacob A Abraham. 2007. ACCE: Auto- matic correction of control-flow errors. In Test Conference, 2007. ITC 2007. IEEE International. IEEE, 1–10.

[13] Eric Delano. 2005. Checkpointing of register file. (Sept. 6 2005). US Patent 6,941,489.

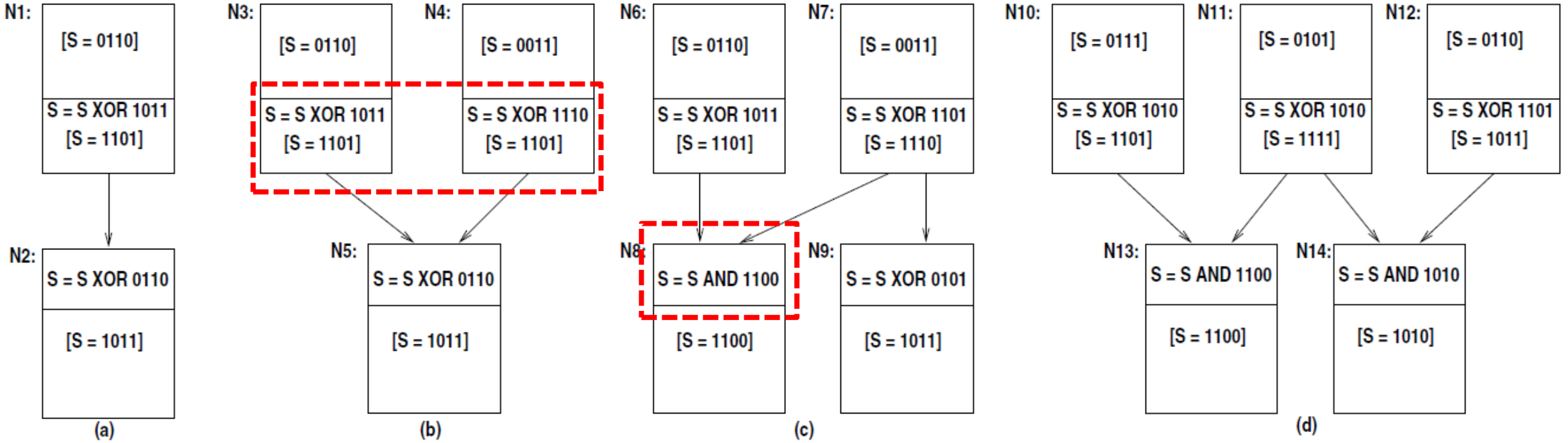
[14] Tuo Li, Roshan Ragel, and Sri Parameswaran. 2012. Reli: Hardware/software checkpoint and recovery scheme for embedded processors. In Proceedings of the Conference on Design, Automation and Test in Europe, 875–880.

[15] Tuo Li, Jude Angelo Ambrose, and Sri Parameswaran. 2016. ReCoRD: reducing register traffic for checkpointing in embedded processors. In Proceedings of the 2016 Conference on Design, Automation & Test in Europe, 582– 587.

[16] H. Mushtaq, Z. Al-Ars and K. Bertels, “Survey of Fault Tolerance Techniques for Shared Memory Multicore/Multiprocessor Systems,” in *IEEE 6th International Design and Test Workshop (IDT)*, 2011

Appendix
Signature Assignment and Update Op.

- **Instrumenting signature into Basic Blocks (BB), CFE can be detected**
  - A basic block can be classified into A type (and-op) or X type (xor-op)



Same successor, exit signatures should be the same

A type:
>=2 predecessors
Predecessor's successor >=2

By carefully arrange signature, signatures of N13 and N14 can be different.

Reference: [5]

Appendix
Error Correction Latency Factors

- **ECL: the latency from recover a CFE after it occurred**
  - Factor 1: From CFE occurred to CFE detected
  - Factor 2: Length of recovery procedure
  - Factor 3: From checkpoint to where CFE occurred