Improving Fault Attacks On Embedded Software Using RISC Pipeline Characterization

Bilgiday Yuce, Nahid Farhady Ghalaty, Patrick Schaumont

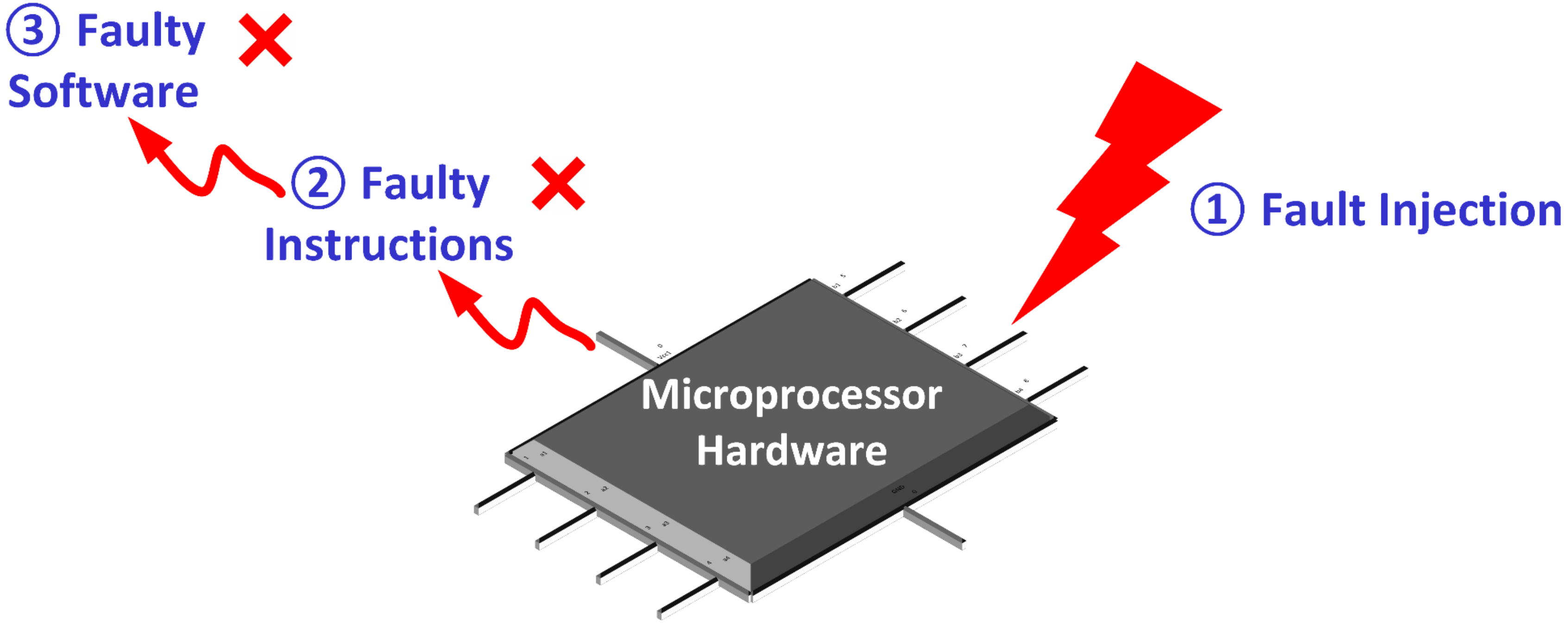
Virginia Tech

FDTC 2015

This research was supported through NSF Grant 1441710, Grant 1115839, and through SRC.

Fault Injection into Embedded Software

- Hardware determines the fault behavior of software.



③ Faulty Software ✗

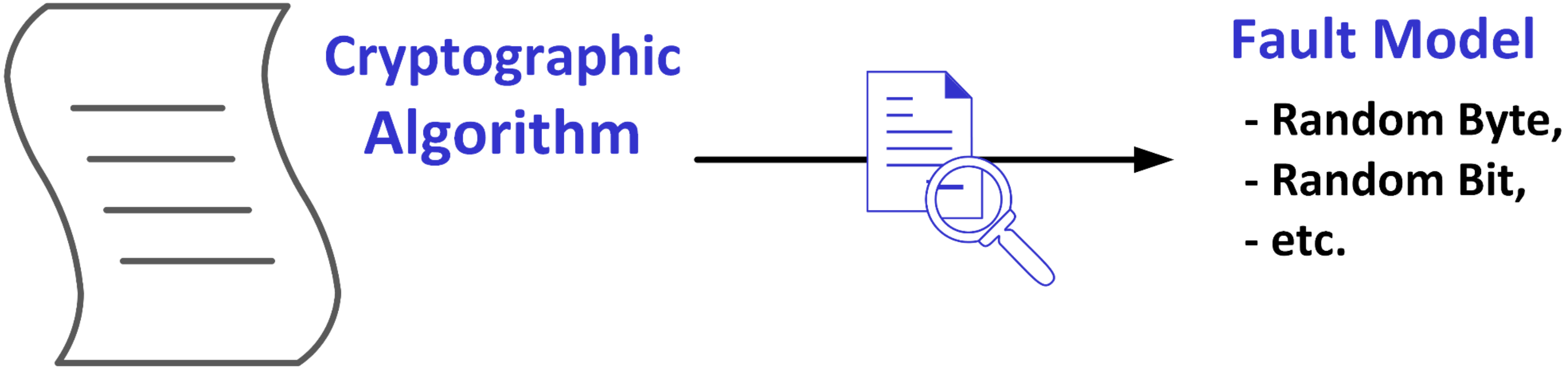
② Faulty Instructions ✗

① Fault Injection

Microprocessor Hardware

Traditional Methods (1)

- Start with a high-level assumption on fault behavior



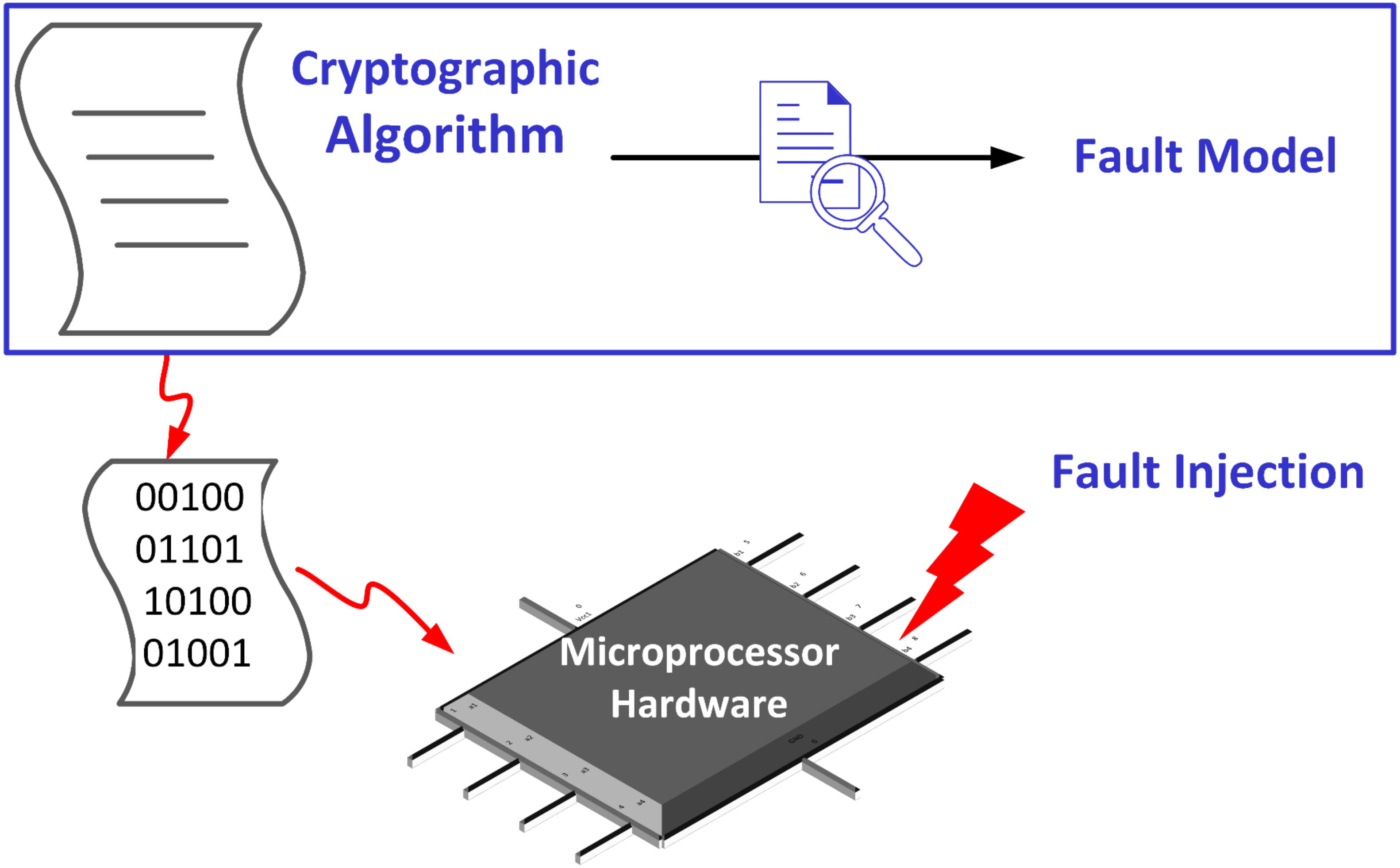
Cryptographic Algorithm

Fault Model

- Random Byte,
- Random Bit,
- etc.

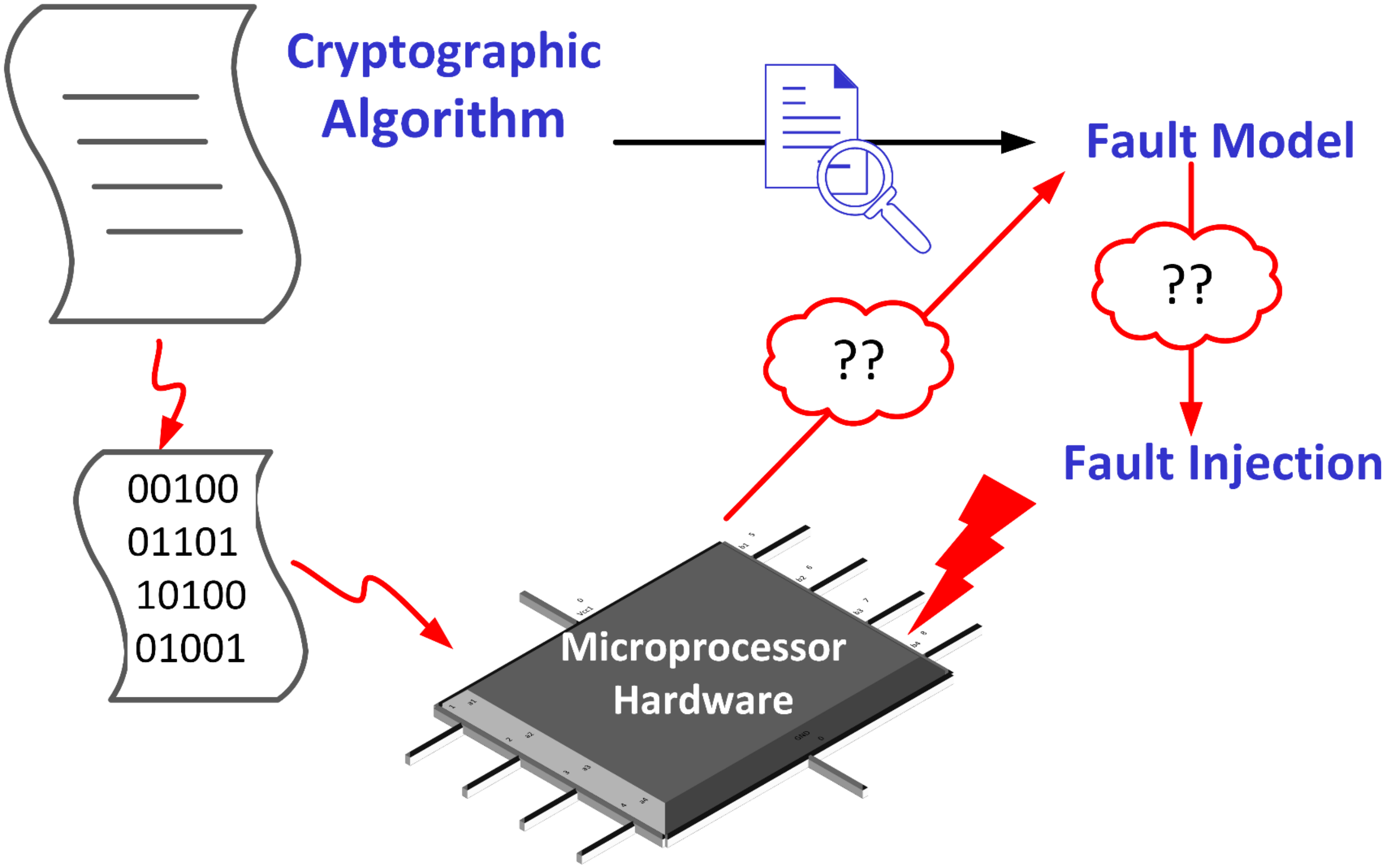
Traditional Methods (2)

- There is a gap between assumptions and reality.



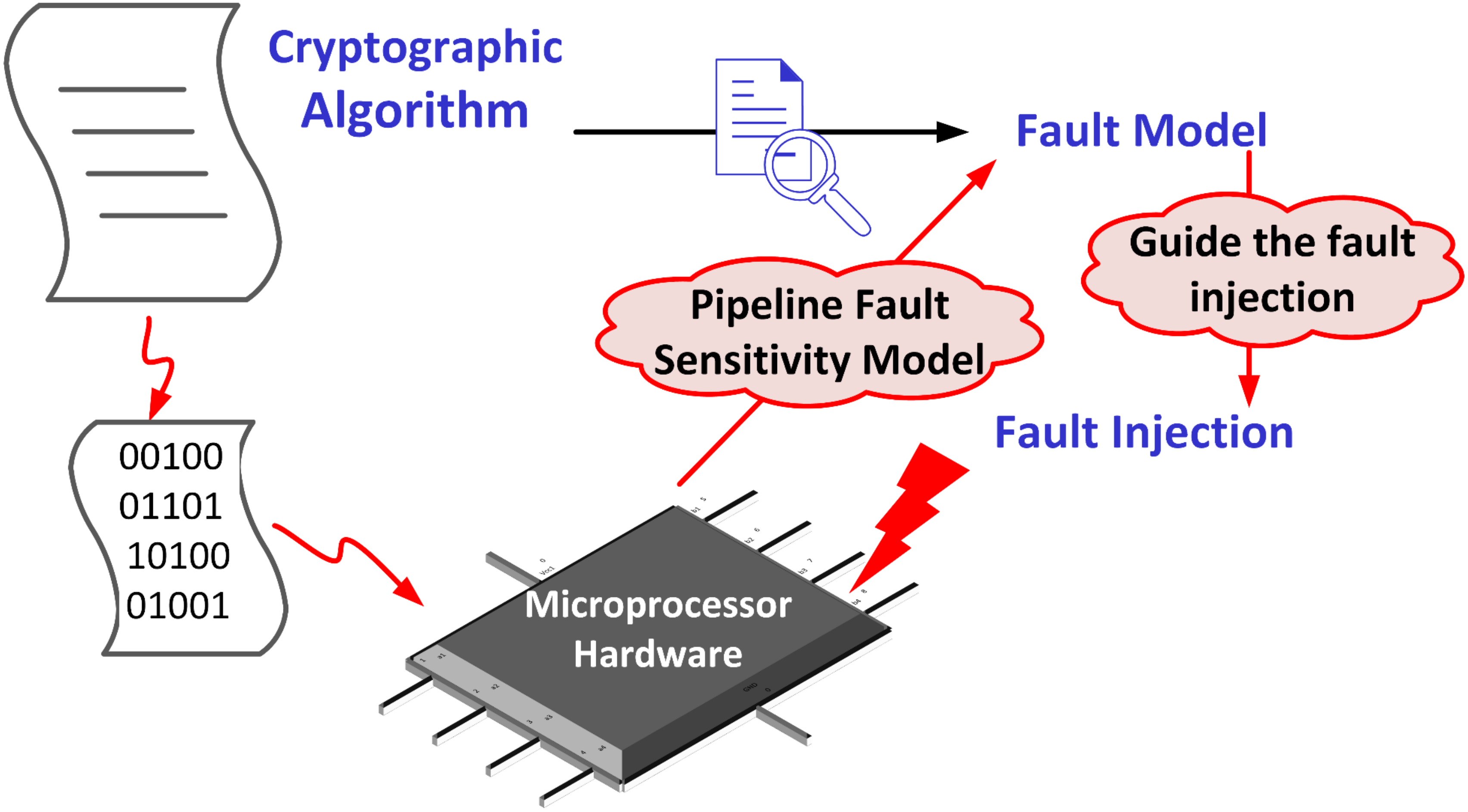
Problems of the Traditional Methods

- Microprocessor hardware is not fully utilized.



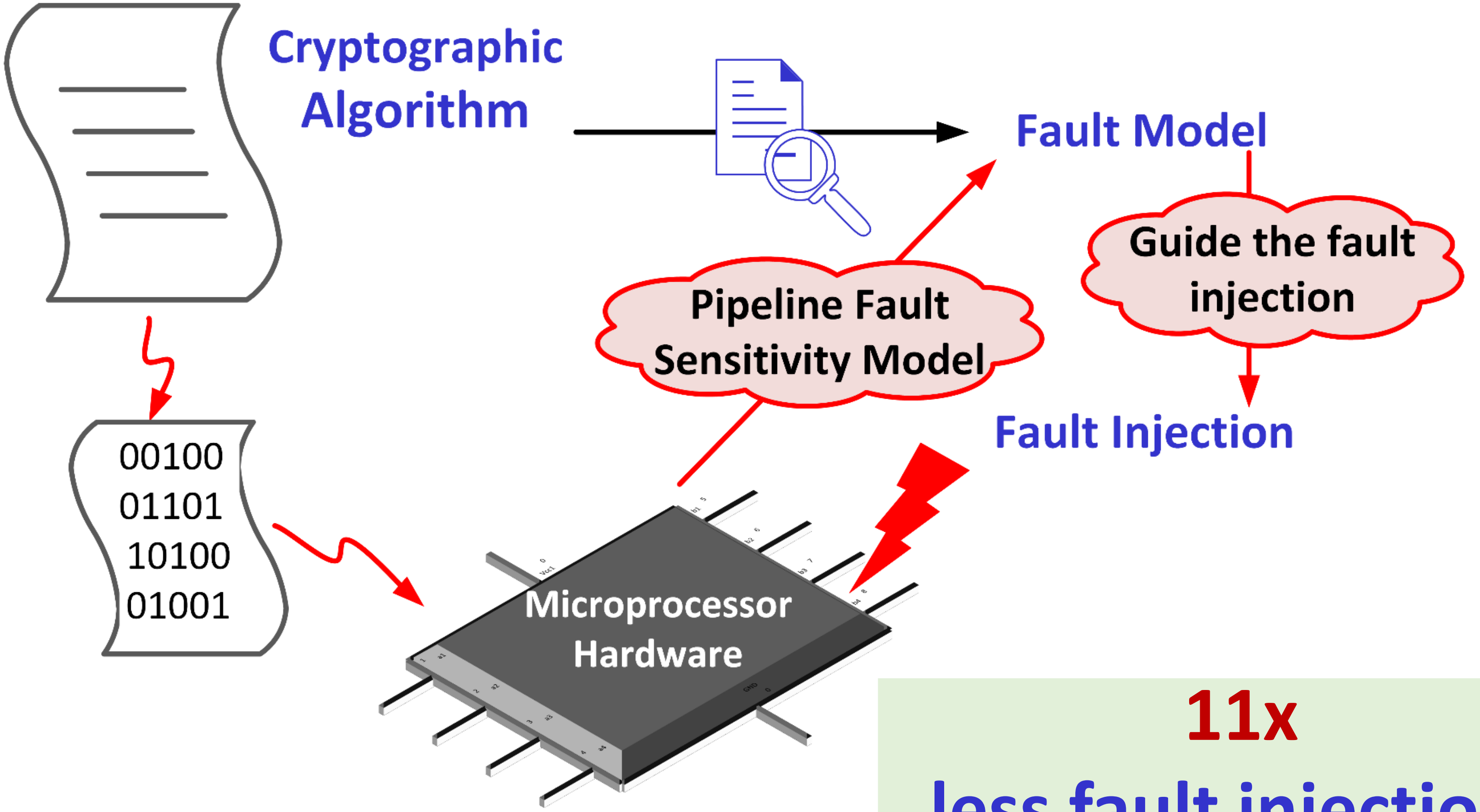
Proposed Method (1)

- Microprocessor Aware Fault Attack



Proposed Method (2)

- More practical fault models and efficient injection

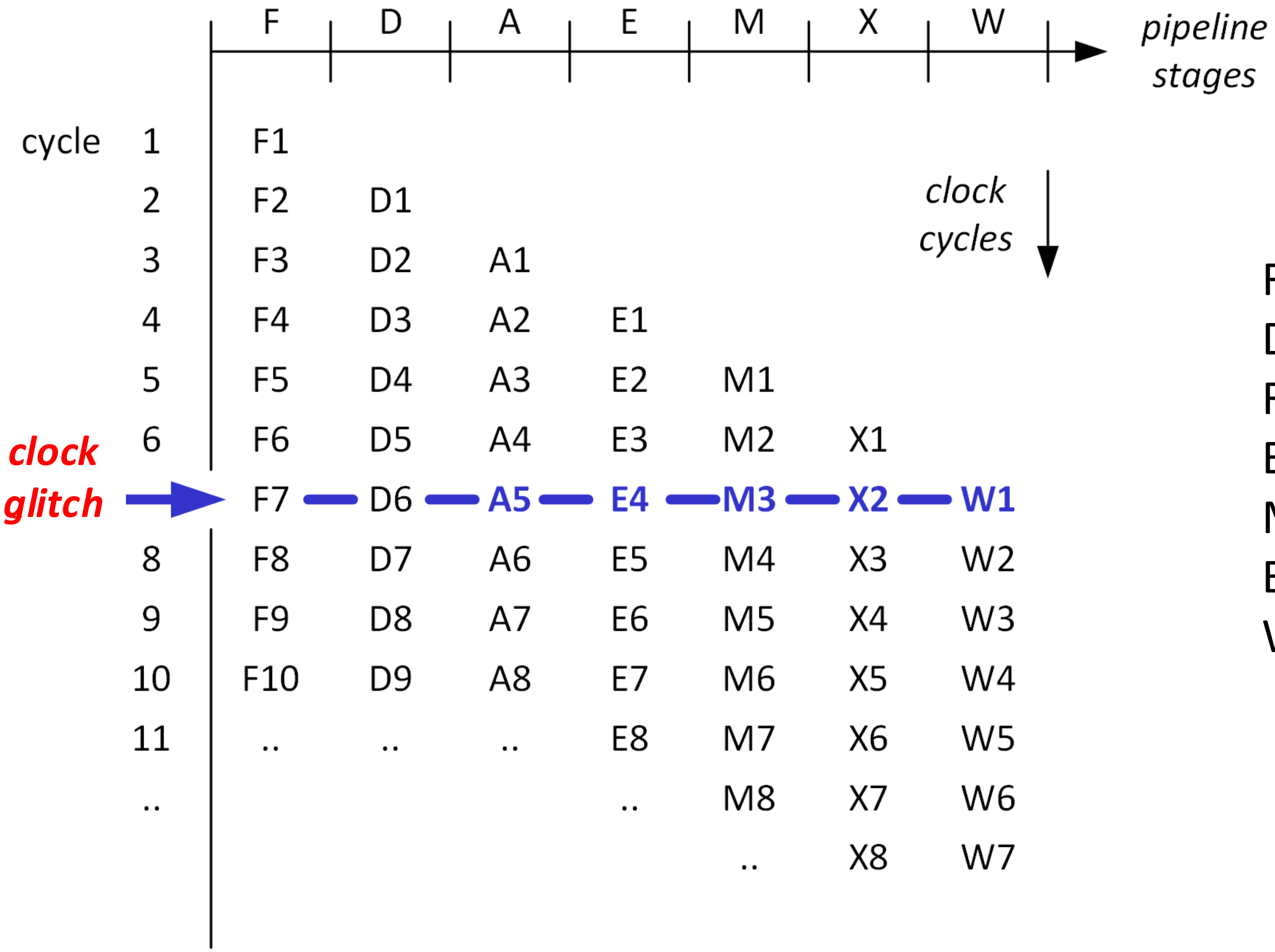


11x

less fault injections

Fault Behavior in a RISC Pipeline (1)

- 7-Stage RISC Pipeline:

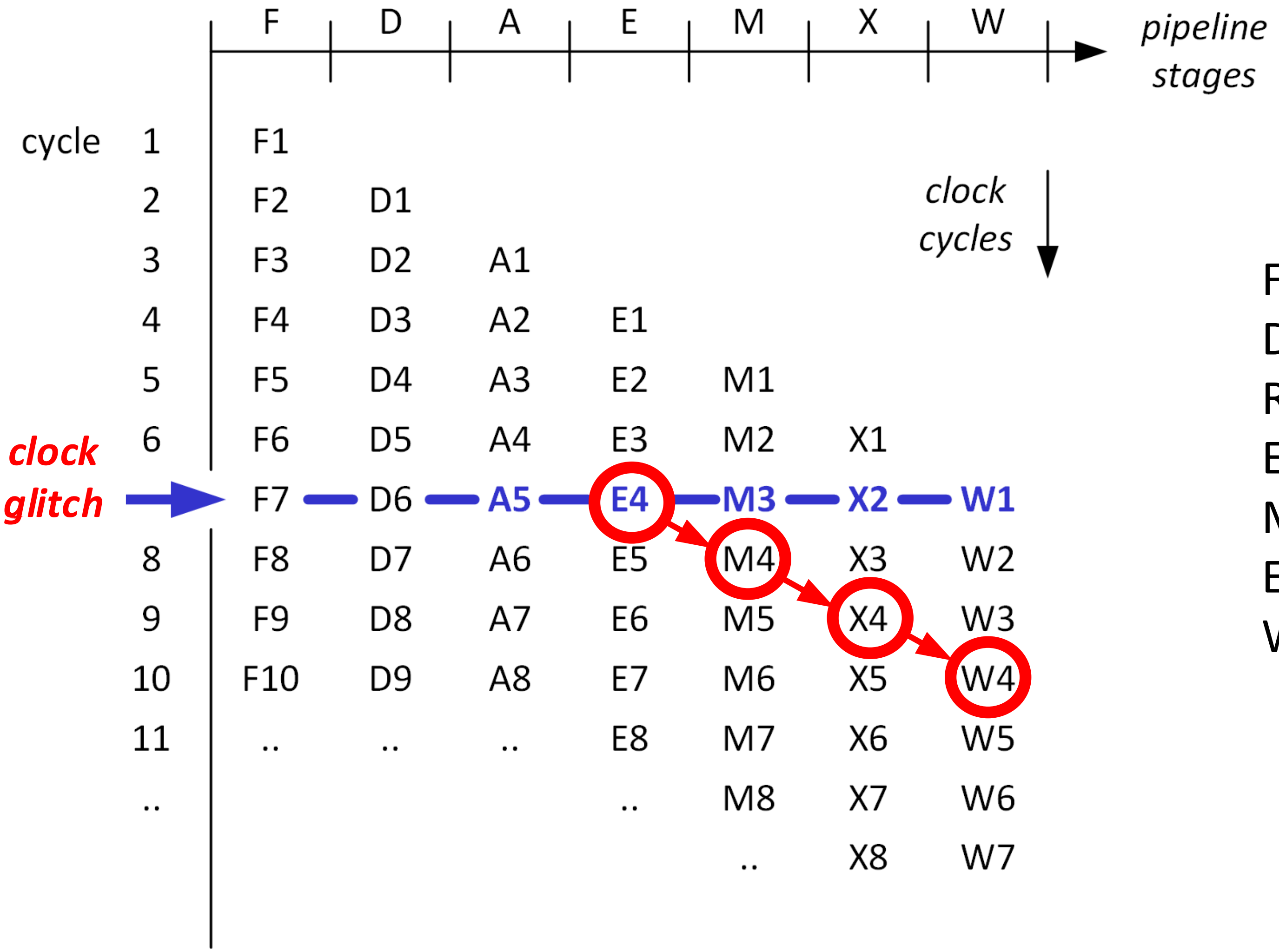


cycle	F	D	A	E	M	X	W
1	F1						
2	F2	D1					
3	F3	D2	A1				
4	F4	D3	A2	E1			
5	F5	D4	A3	E2	M1		
6	F6	D5	A4	E3	M2	X1	
	F7	D6	**A5**	**E4**	**M3**	**X2**	**W1**
8	F8	D7	A6	E5	M4	X3	W2
9	F9	D8	A7	E6	M5	X4	W3
10	F10	D9	A8	E7	M6	X5	W4
11	..	..	..	E8	M7	X6	W5
..				..	M8	X7	W6
					..	X8	W7

Fetch (F)
Decode (D)
Register Access (A)
Execute (E)
Memory (F)
Exception (X)
Write-Back (W)

Fault Behavior in a RISC Pipeline (2)

- If E4 has the highest critical path (i.e, fault sensitivity):

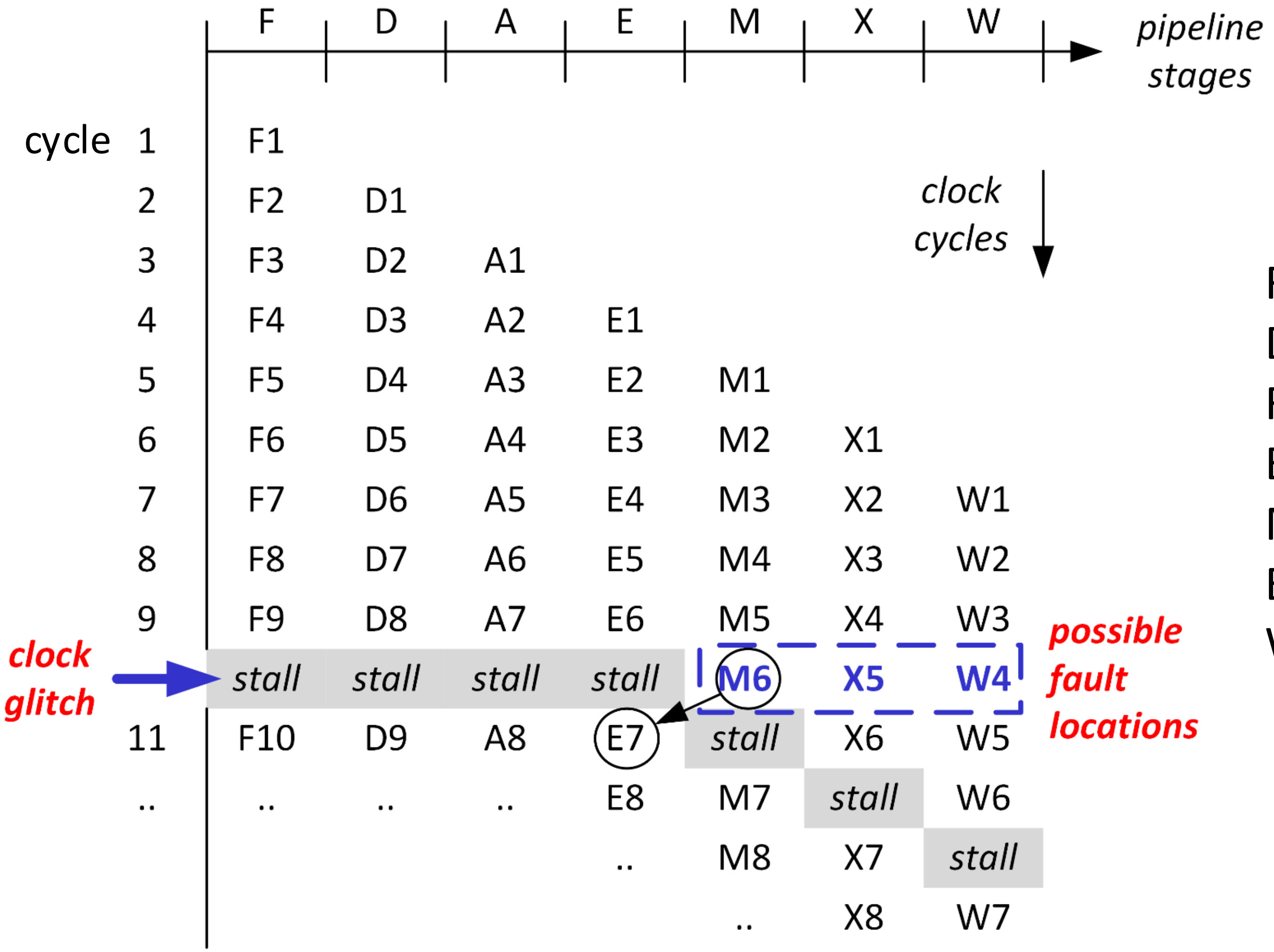


cycle	F	D	A	E	M	X	W
1	F1						
2	F2	D1					
3	F3	D2	A1				
4	F4	D3	A2	E1			
5	F5	D4	A3	E2	M1		
6	F6	D5	A4	E3	M2	X1	
	F7	D6	A5	E4	M3	X2	W1
8	F8	D7	A6	E5	M4	X3	W2
9	F9	D8	A7	E6	M5	X4	W3
10	F10	D9	A8	E7	M6	X5	W4
11	..	..	..	E8	M7	X6	W5
..				..	M8	X7	W6
					..	X8	W7

Fetch (F)
Decode (D)
Register Access (A)
Execute (E)
Memory (F)
Exception (X)
Write-Back (W)

Fault Behavior in a RISC Pipeline (3)

- Pipeline stalls blind the stalled stages from glitches.



pipeline stages

clock cycles

clock glitch

possible fault locations

	F	D	A	E	M	X	W
cycle 1	F1						
2	F2	D1					
3	F3	D2	A1				
4	F4	D3	A2	E1			
5	F5	D4	A3	E2	M1		
6	F6	D5	A4	E3	M2	X1	
7	F7	D6	A5	E4	M3	X2	W1
8	F8	D7	A6	E5	M4	X3	W2
9	F9	D8	A7	E6	M5	X4	W3
	stall	*stall*	*stall*	*stall*	**M6**	**X5**	**W4**
11	F10	D9	A8	E7	*stall*	X6	W5
..	..	..	..	E8	M7	*stall*	W6
				..	M8	X7	*stall*
					..	X8	W7

Fetch (F)
Decode (D)
Register Access (A)
Execute (E)
Memory (F)
Exception (X)
Write-Back (W)

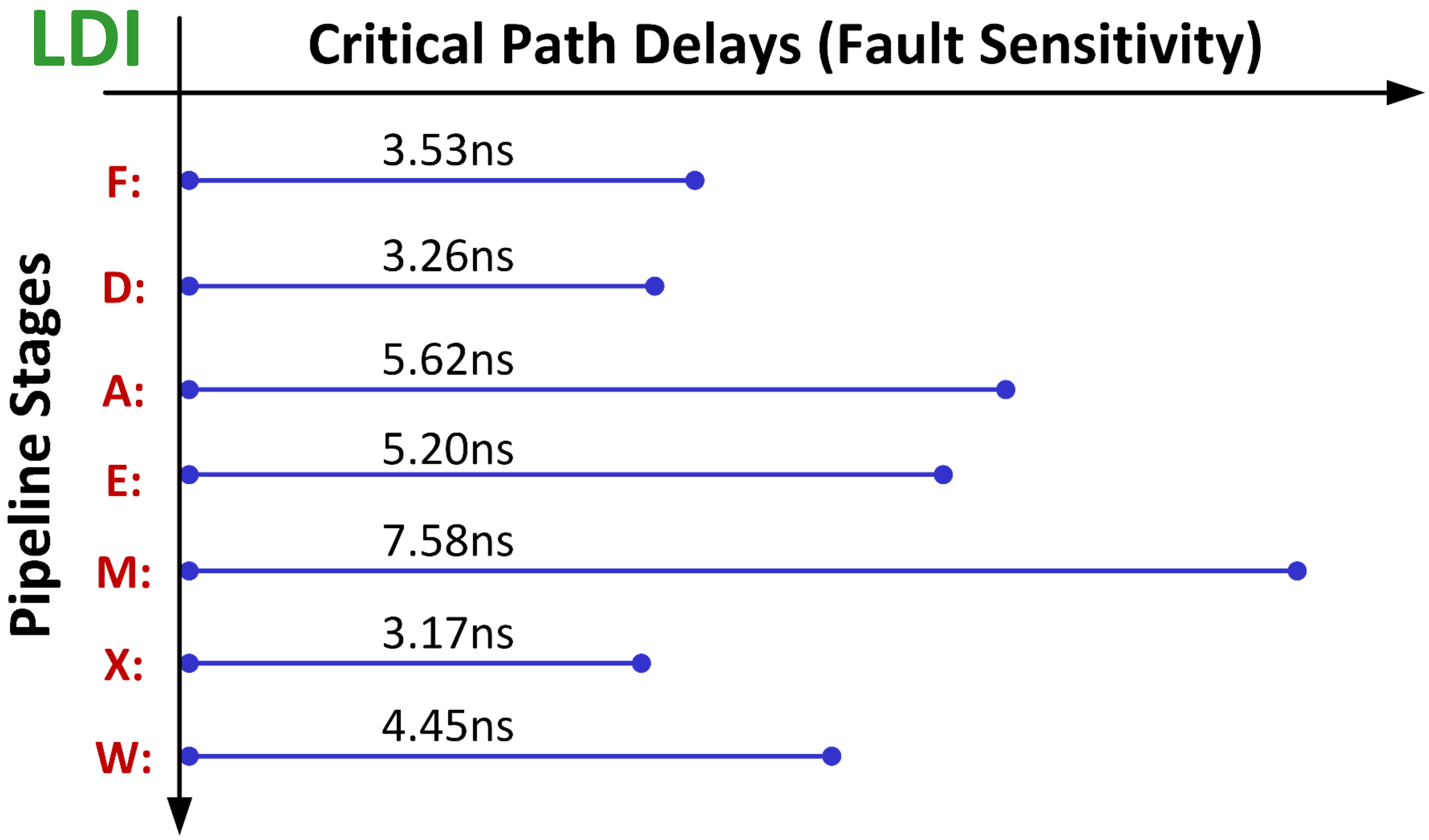
Case Study

- Case Study:
  - Fault Analysis: Differential Fault Intensity Analysis (DFIA)
  - Software: AES
  - Hardware: LEON3 Processor

- DFIA [Ghalaty et. al, FDTC'14]:
  - Relies on a biased fault behavior
  - Gradual fault behavior **in proportion to** the fault intensity

Creating Fault Sensitivity Model

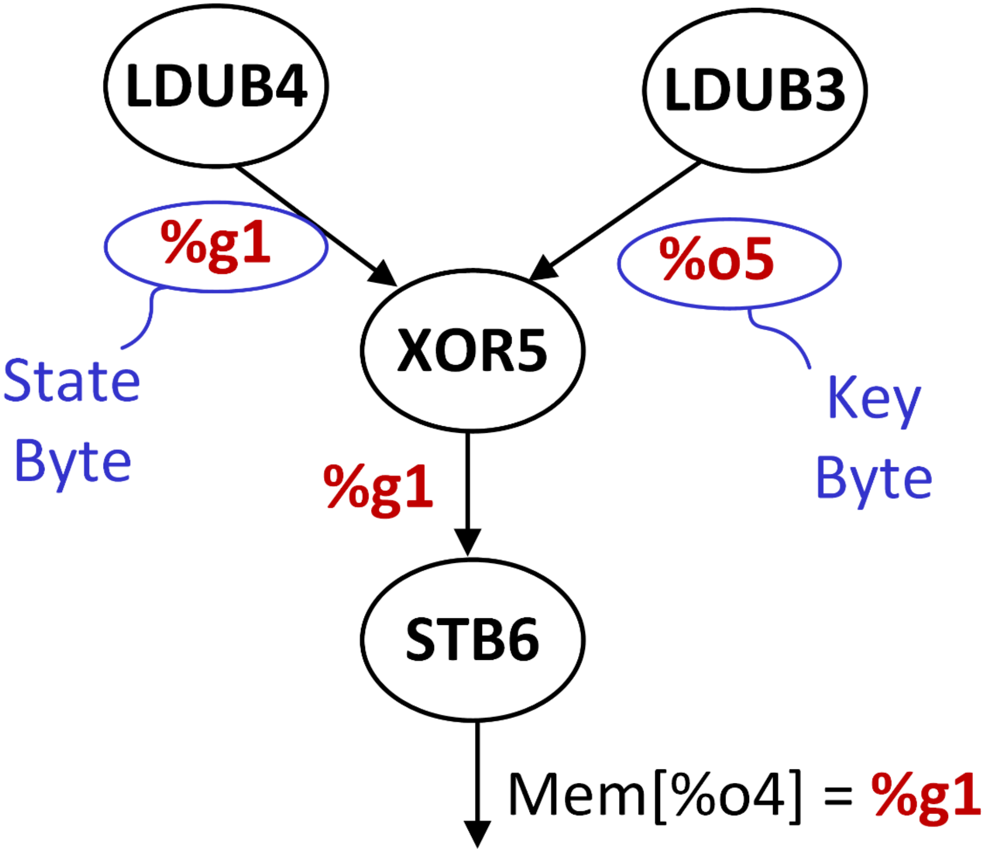
- Fault sensitivity of each (instruction, pipeline stage)



Determining Target Instructions

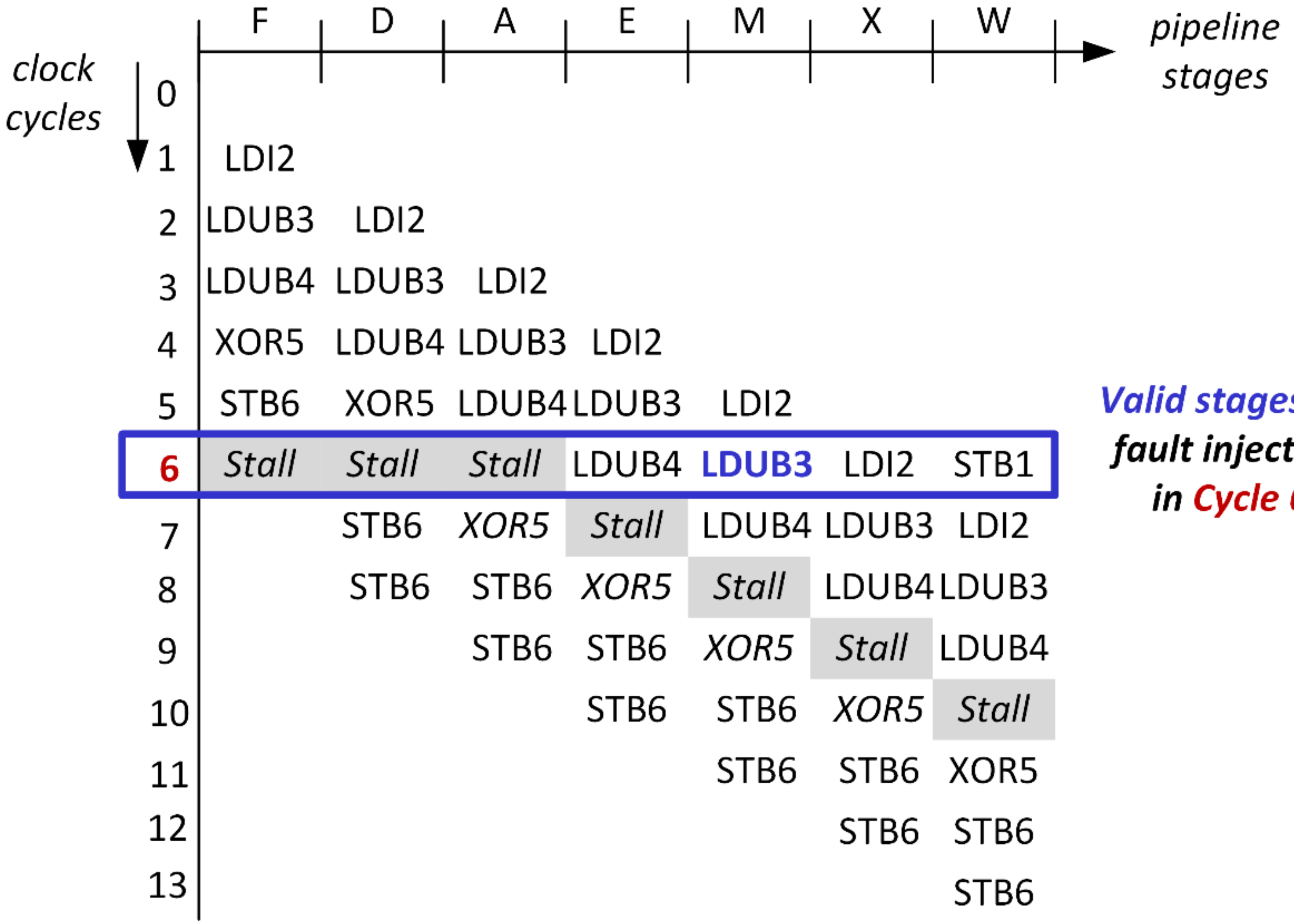
- Objective:
  - Biased faults in the *AddRoundKey* at AES Round 9

***AddRoundKey* *on LEON3*:**



Determining Valid Pipeline Stages (1)

- Analyze each cycle of execution

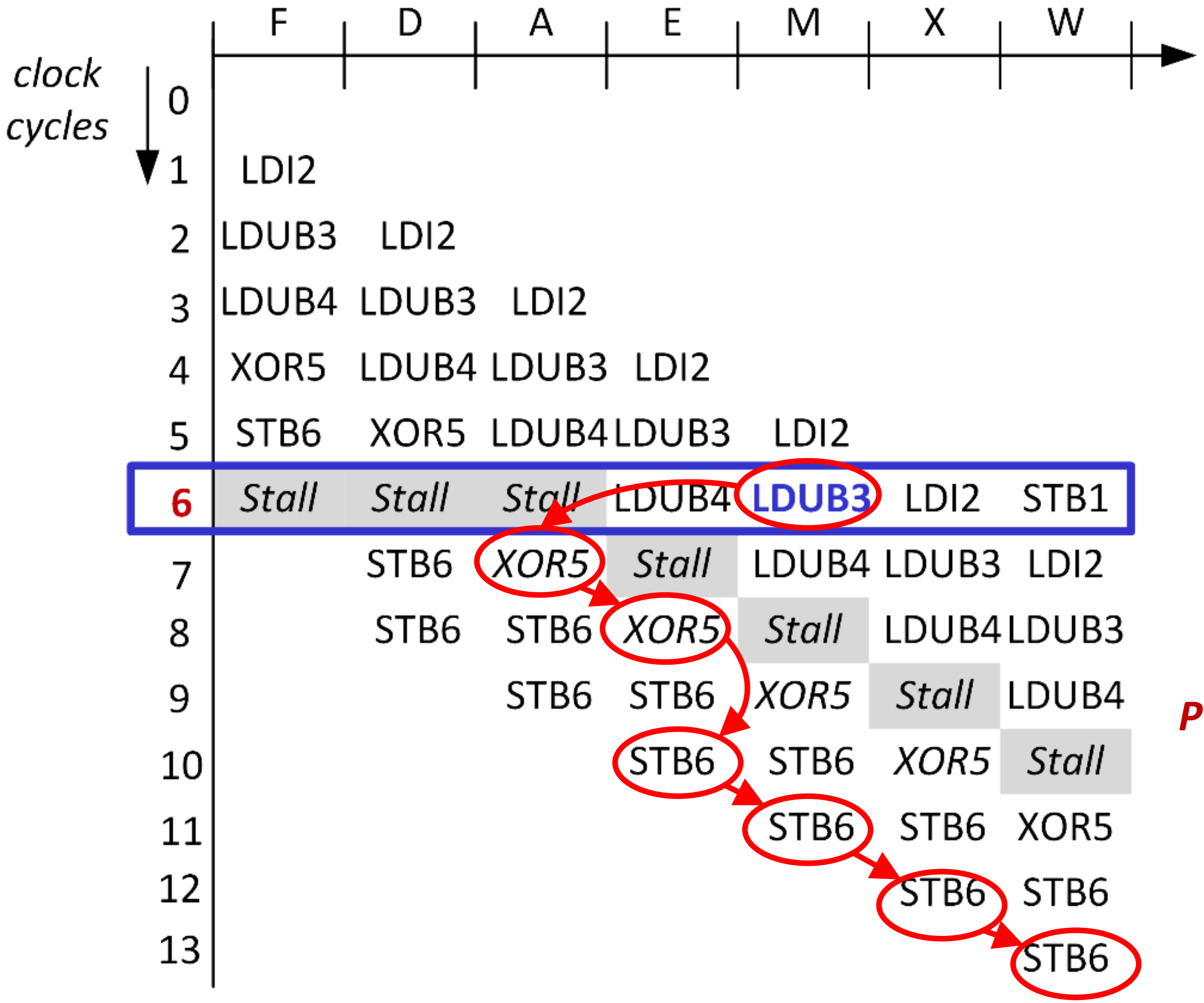


	F	D	A	E	M	X	W
0							
1	LDI2						
2	LDUB3	LDI2					
3	LDUB4	LDUB3	LDI2				
4	XOR5	LDUB4	LDUB3	LDI2			
5	STB6	XOR5	LDUB4	LDUB3	LDI2		
6	*Stall*	*Stall*	*Stall*	LDUB4	**LDUB3**	LDI2	STB1
7		STB6	*XOR5*	*Stall*	LDUB4	LDUB3	LDI2
8		STB6	STB6	*XOR5*	*Stall*	LDUB4	LDUB3
9			STB6	STB6	*XOR5*	*Stall*	LDUB4
10				STB6	STB6	*XOR5*	*Stall*
11					STB6	STB6	XOR5
12						STB6	STB6
13							STB6

Valid stages for fault injection in Cycle 6

Determining Valid Pipeline Stages (2)

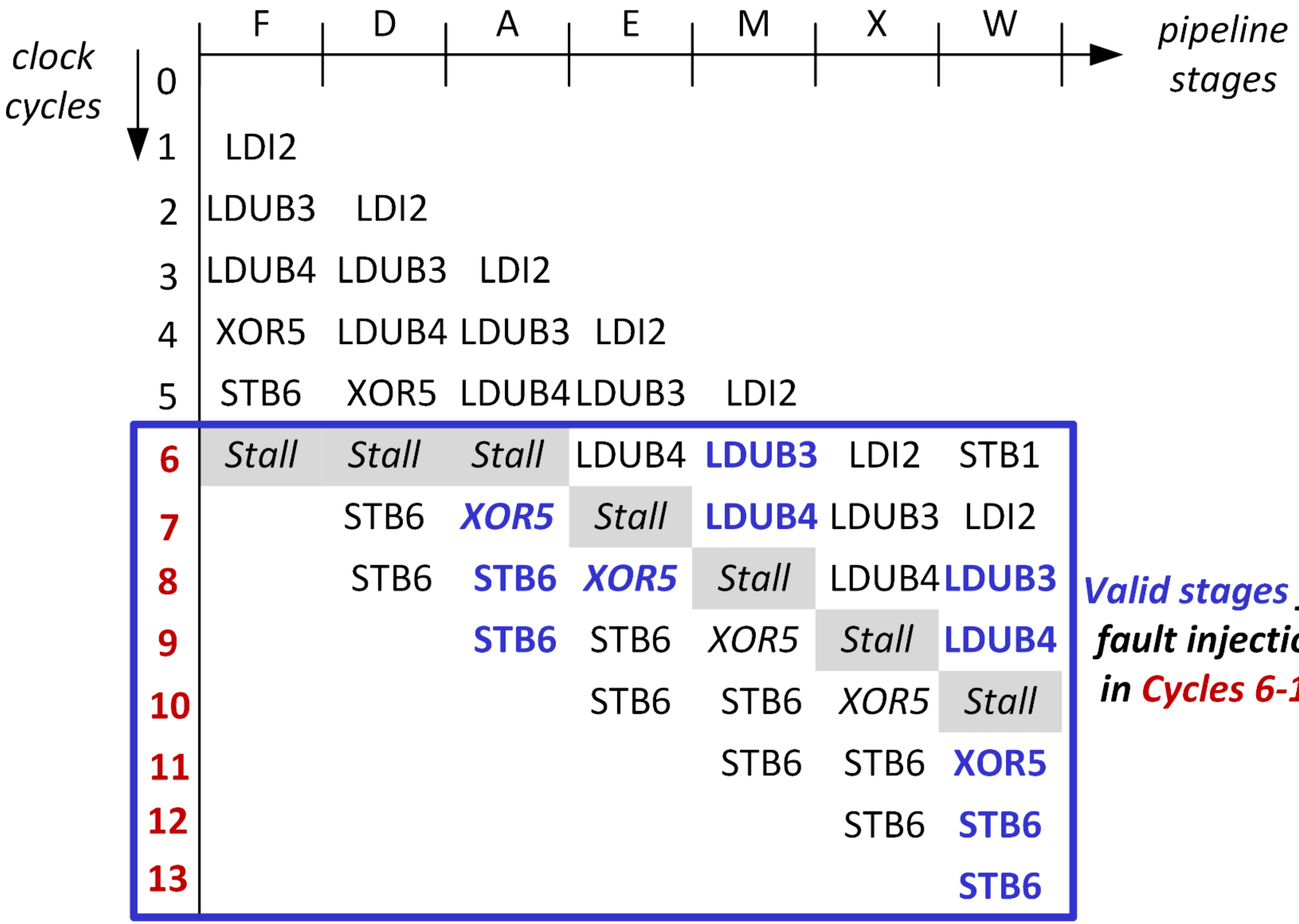
- Propagation of a biased fault injected into (LDUB3, M)



	F	D	A	E	M	X	W
0							
1	LDI2						
2	LDUB3	LDI2					
3	LDUB4	LDUB3	LDI2				
4	XOR5	LDUB4	LDUB3	LDI2			
5	STB6	XOR5	LDUB4	LDUB3	LDI2		
6	*Stall*	*Stall*	*Stall*	LDUB4	**LDUB3**	LDI2	STB1
7		STB6	*XOR5*	*Stall*	LDUB4	LDUB3	LDI2
8		STB6	STB6	*XOR5*	*Stall*	LDUB4	LDUB3
9			STB6	STB6	*XOR5*	*Stall*	LDUB4
10				STB6	STB6	*XOR5*	*Stall*
11					STB6	STB6	XOR5
12						STB6	STB6
13							STB6

Fault Propagation Path for LDUB3

Determining Valid Pipeline Stages (3)



clock cycles

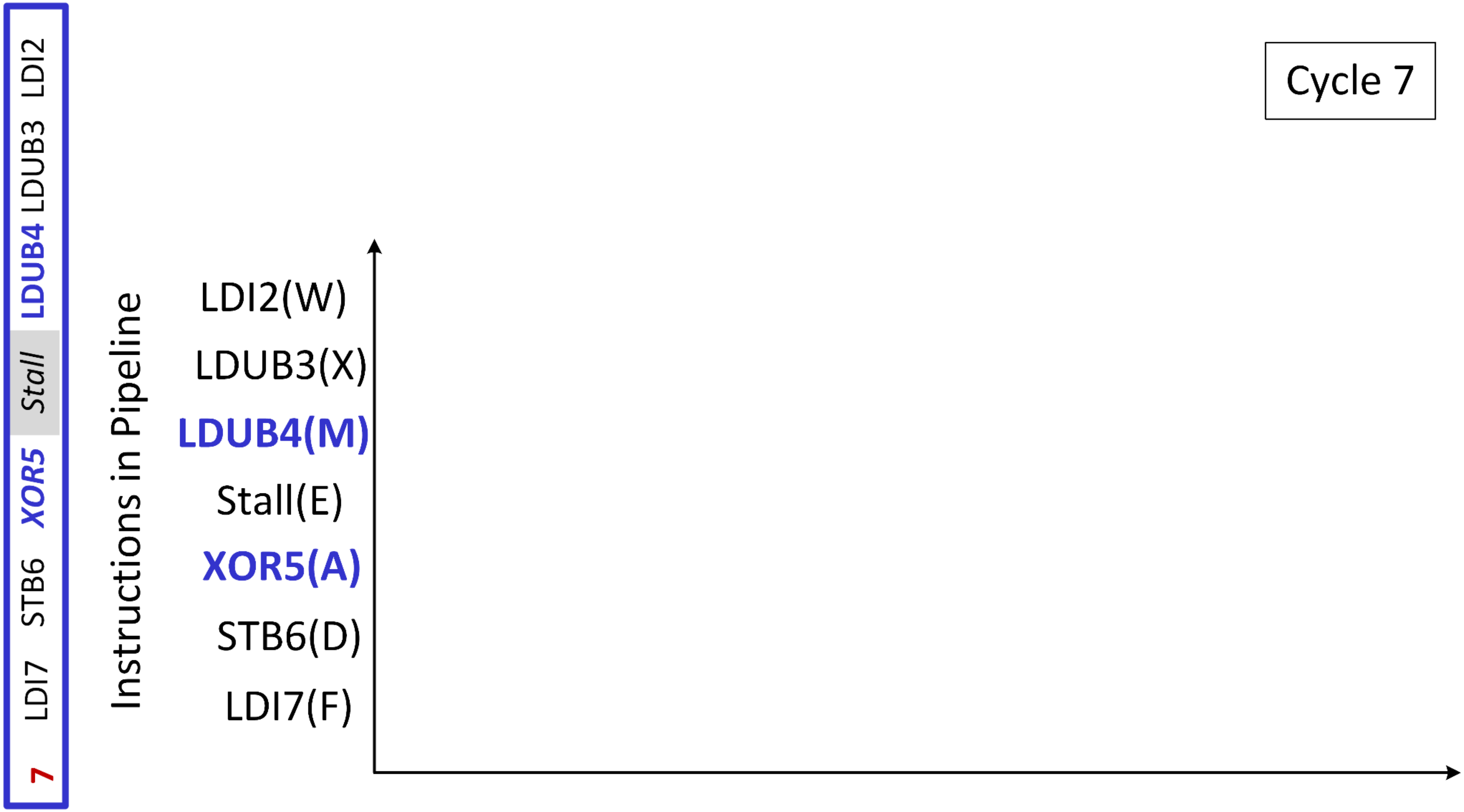
pipeline stages

	F	D	A	E	M	X	W
0							
1	LDI2						
2	LDUB3	LDI2					
3	LDUB4	LDUB3	LDI2				
4	XOR5	LDUB4	LDUB3	LDI2			
5	STB6	XOR5	LDUB4	LDUB3	LDI2		
6	*Stall*	*Stall*	*Stall*	LDUB4	**LDUB3**	LDI2	STB1
7		STB6	***XOR5***	*Stall*	**LDUB4**	LDUB3	LDI2
8		STB6	**STB6**	***XOR5***	*Stall*	LDUB4	**LDUB3**
9			**STB6**	STB6	*XOR5*	*Stall*	**LDUB4**
10				STB6	STB6	*XOR5*	*Stall*
11					STB6	STB6	**XOR5**
12						STB6	**STB6**
13							**STB6**

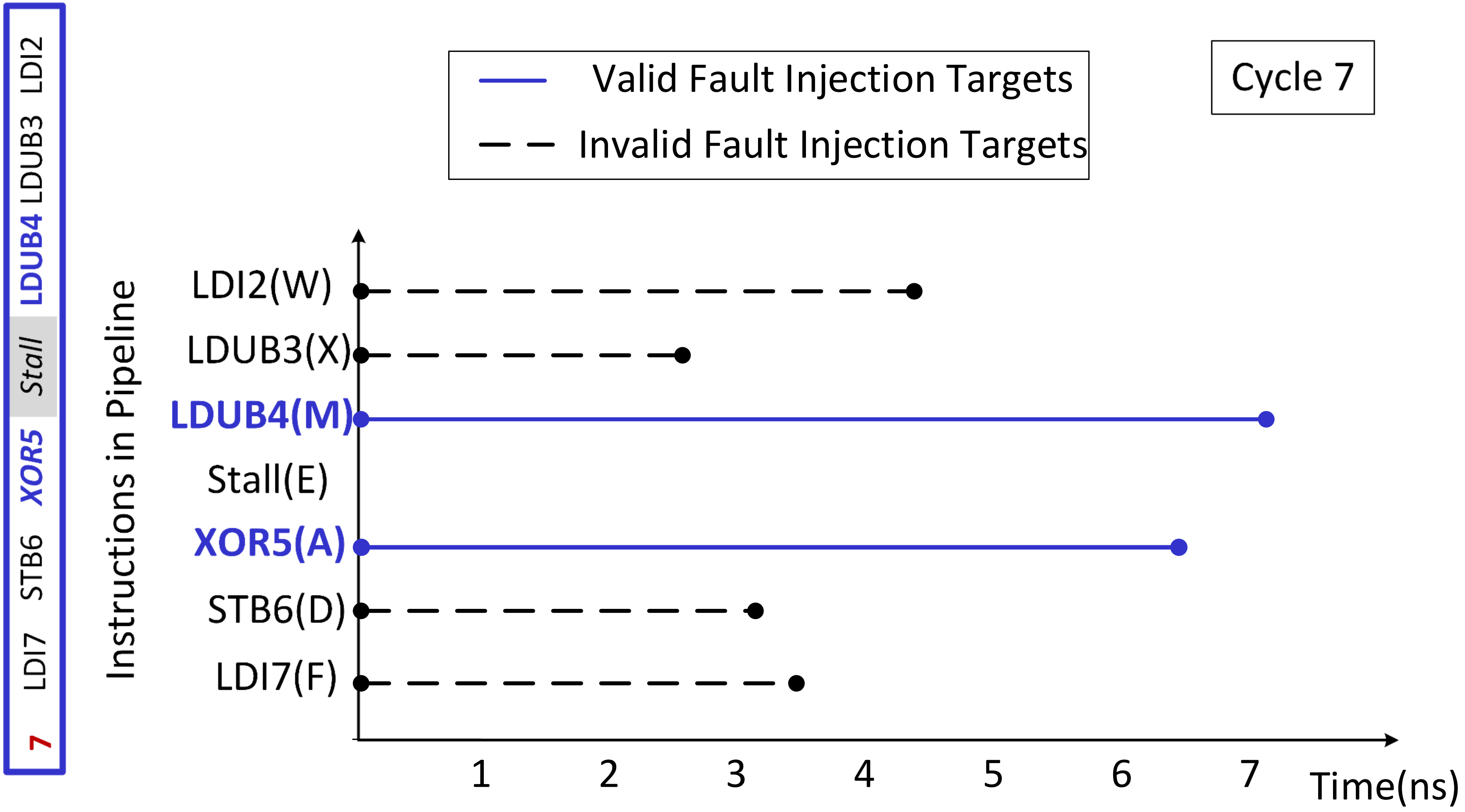
Valid stages for fault injection in Cycles 6-13

Using Fault Sensitivity Model (1)

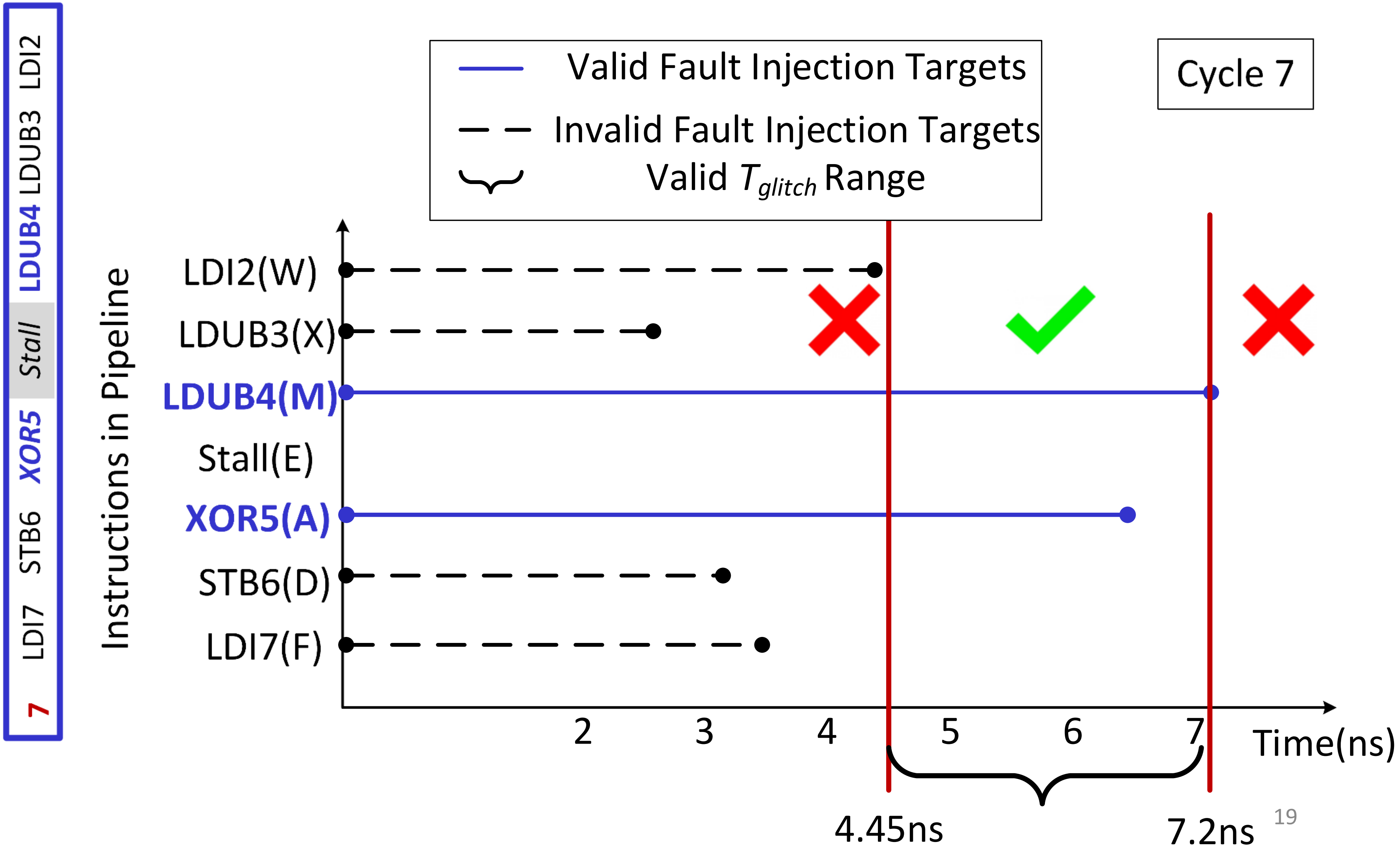
Cycle 7



Using Fault Sensitivity Model (2)



Using Fault Sensitivity Model (3)



Results (1)

- Fault Injection Experiments on a LEON3:
  - Implemented on a SPARTAN-6 FPGA
  - Clock glitch injection

- A DFIA attack on a AES software program:
  - 1 secret key and 1 plaintext

Results (2)

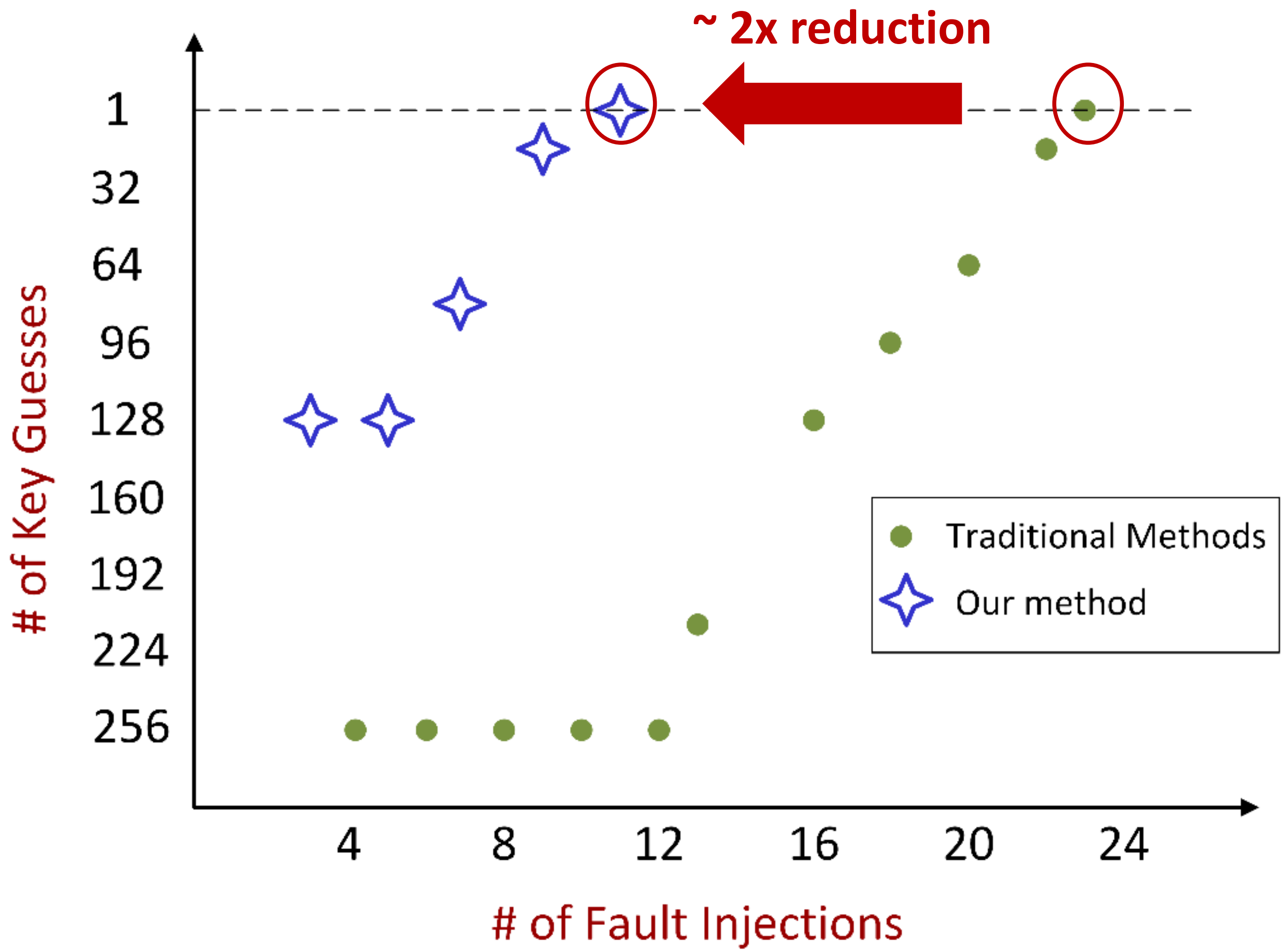
- Our approach requires **~11x** less fault injections.

	Total # of Attacked Cycles	Total # of Fault Injections
Traditional Methods	13	1040
Our Method	6	90

~ 11x
Reduction

Results (3)

- DFIA retrieves the key byte quicker with our method.



Conclusions

- For efficient fault attacks on embedded software, use
  - micro-architectural properties (i.e. fault sensitivity model)
  - architectural properties (i.e. pipeline analysis of the software)
- With a microprocessor aware fault attack method:
  - Possible to tune the injected faults in the software
  - ~11x less fault injections
- Traditional methods need a revision

Thank you!