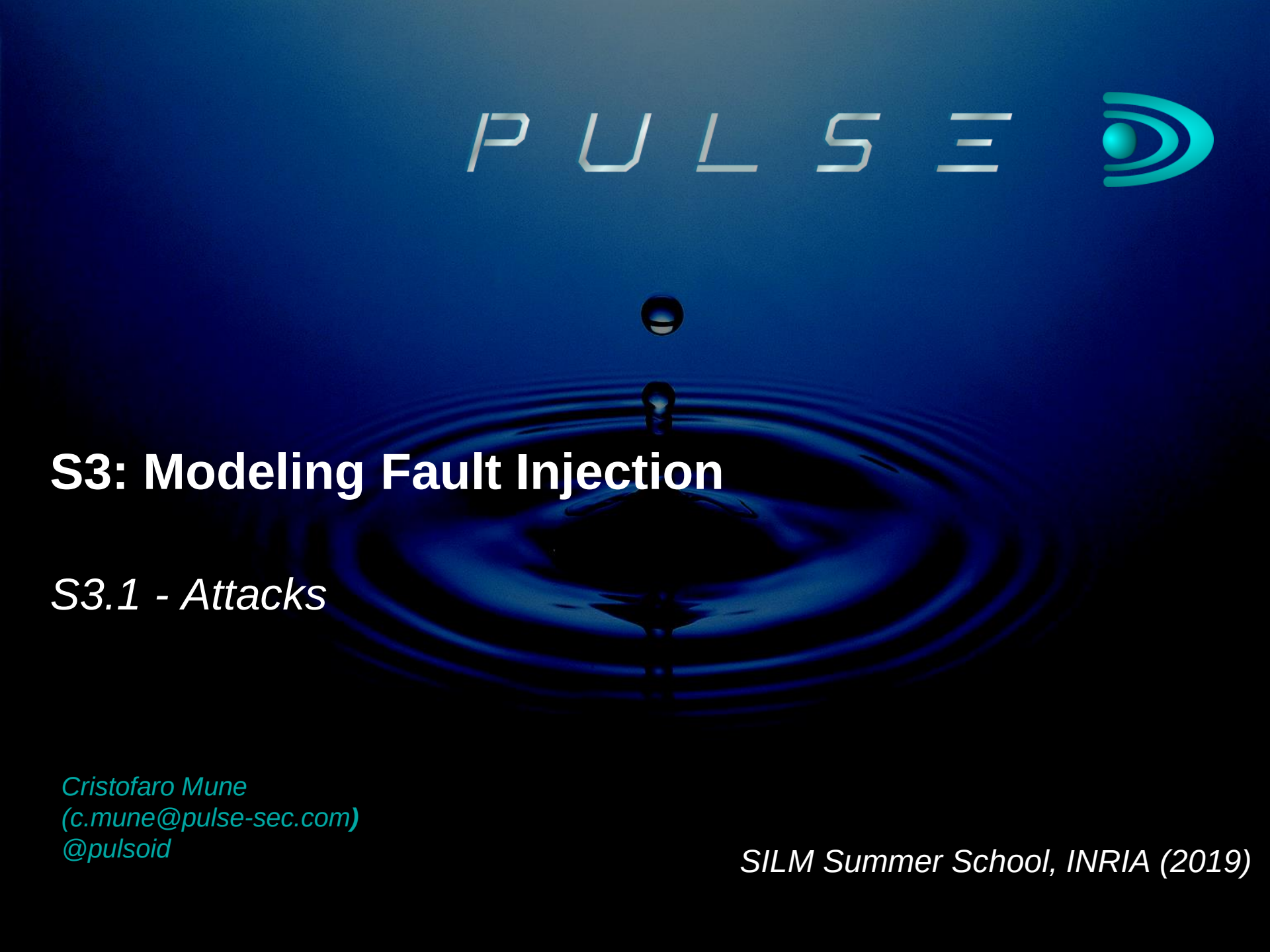


S3: Modeling Fault Injection

S3.1 - Attacks

Cristofaro Mune
(c.mune@pulse-sec.com)
@pulsoid

SILM Summer School, INRIA (2019)

Naming FI attacks

Laser-Induced Fault Injection on Smartphone Bypassing the Secure Boot

BADFET: Defeating Modern Secure Boot Using Second-Order Pulsed Electromagnetic Fault Injection

Exploiting the DRAM rowhammer bug to gain kernel privileges

Escalating Privileges in Linux using Voltage Fault Injection

What do they have in common?

Naming FI attacks

Laser-Induced Fault Injection on Smartphone Bypassing the Secure Boot

BADFET: Defeating Modern Secure Boot Using Second-Order Pulsed Electromagnetic Fault Injection

Exploiting the DRAM rowhammer bug to gain kernel privileges

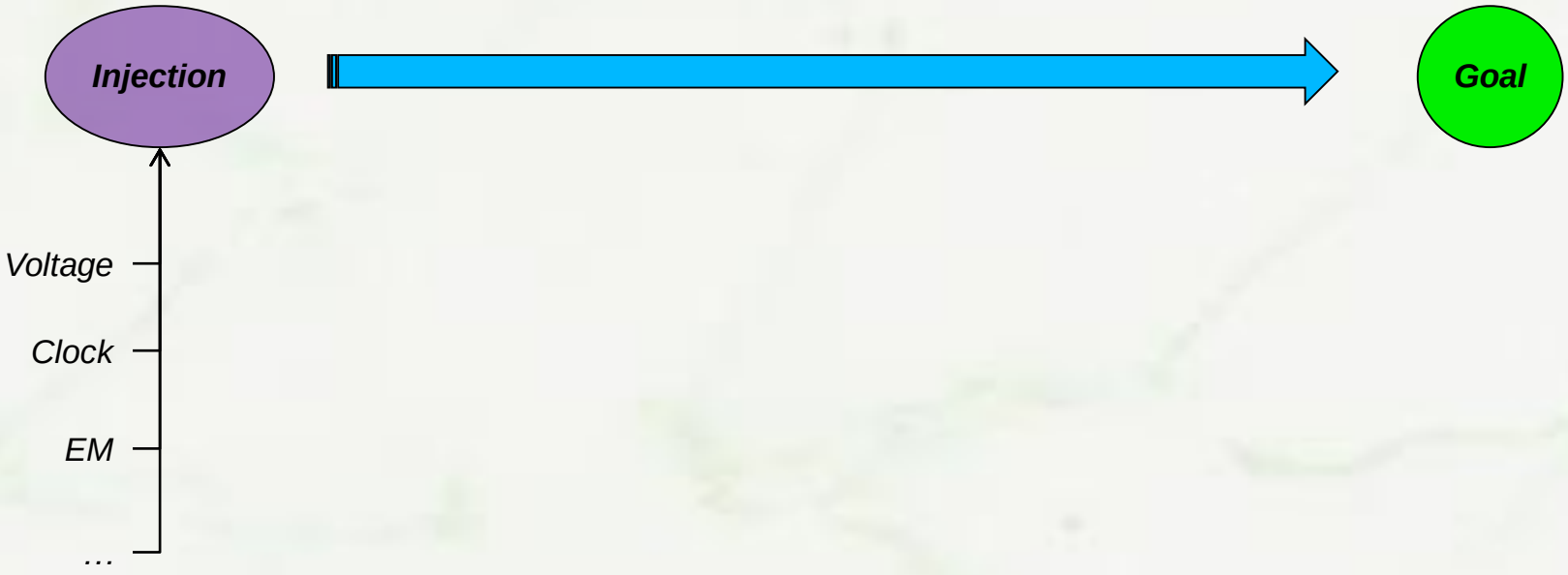
Escalating Privileges in Linux using Voltage Fault Injection

Injection Technique

Goal

Observation

- Naming only connects:
  - *Injection technique* and
  - *Achieved Goal*



FI technique

Reflection

- *Naming may suggest* that entire attack chain:
  - fully *depends on the injection technique*
  - *May not be valid with other injection techniques*

- Questions:
  - Is this true?
  - Are we effectively naming attacks?

(Our) Definitions

Fault Injection

“Introducing faults in a target to alter its intended behavior.”

Other definitions

Vulnerability (FI)

*"Susceptibility of a given **hardware subsystem** to a specific **fault injection technique,** which has an impact on security."*

Glitch

"Controlled environmental change"

Fault (informal)

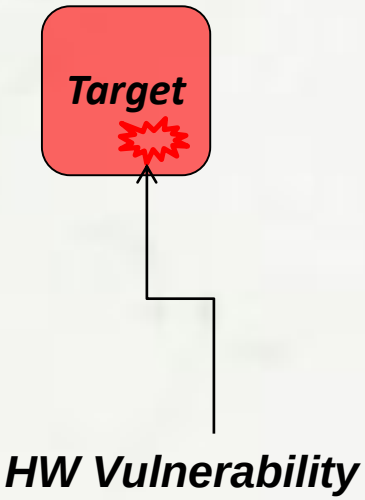
*An unintended alteration of a target, as a consequence of a glitch triggering a **vulnerability**."*

FI Attack Anatomy

Goal

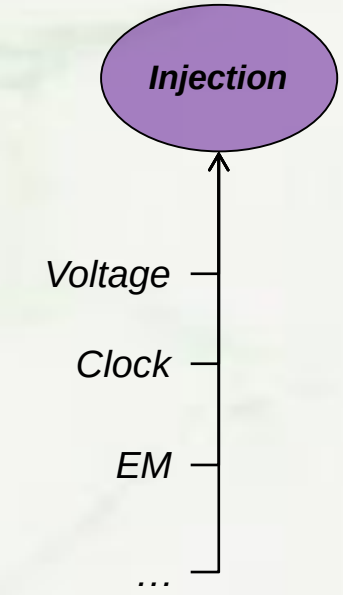
- *Goal:* What attacker wants to achieve
  - Code execution
  - …

Root cause: HW Vulnerability



- When triggered it *leads to* ***faults***
- Located in HW
- Can only be entirely resolved in HW
- Reviews insufficient for identification (Physical parameters)

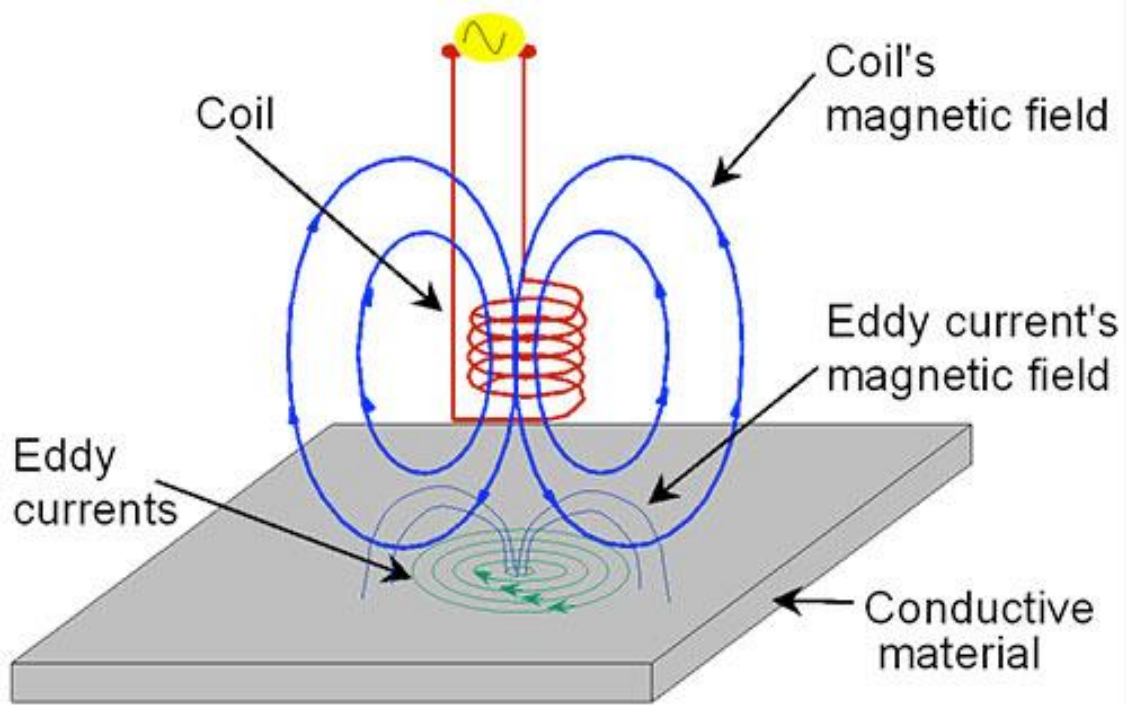
Injection



- *Injection:* technique used for *“triggering”* the HW vulnerability
  - Physical level

Example: EM-FI

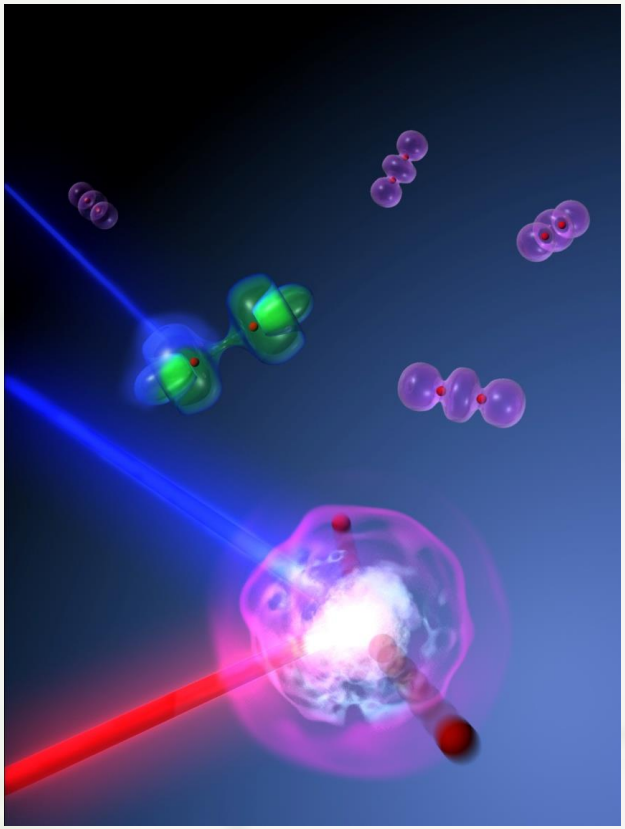
- Physics:
  - Induced current in circuits net
- Logic:
  - Logic state may change (closed → open)



System: Wrong values represented/stored/propagated

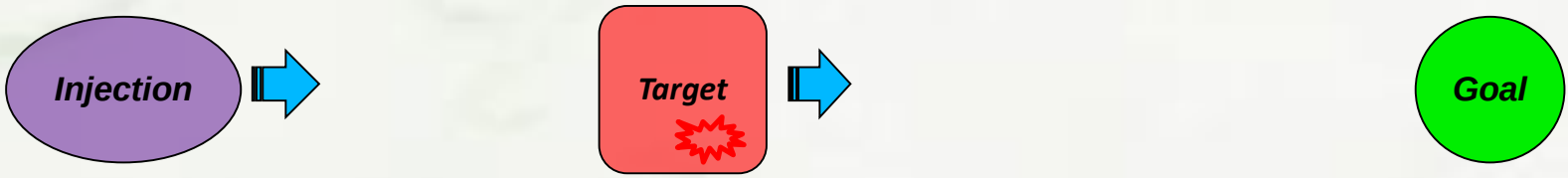
Example: Optical

- Physics:
  - Affects available carriers
- Logic:
  - Logic state may change (closed → open)



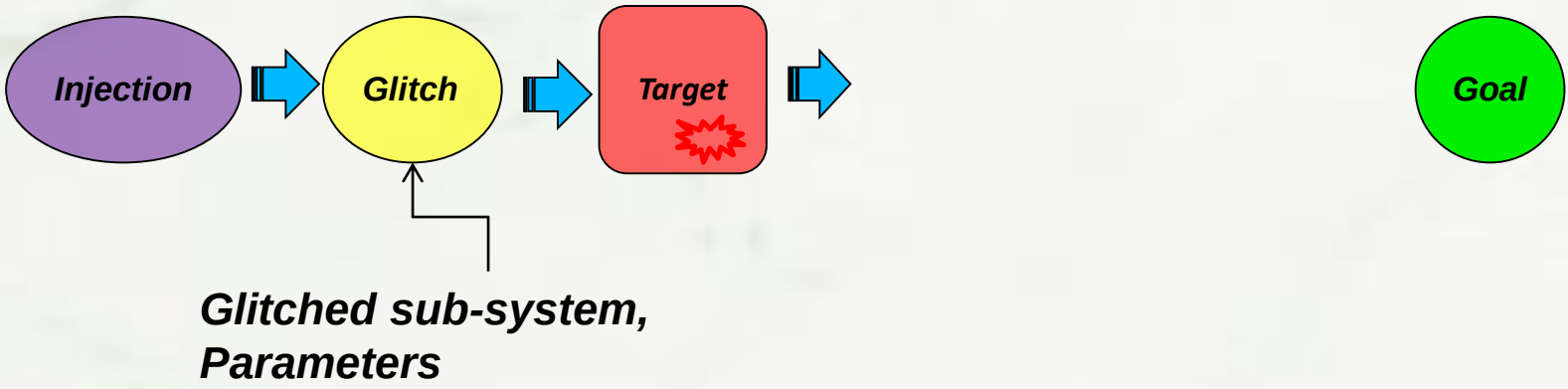
System: Wrong values represented/stored/propagated

Is this enough?



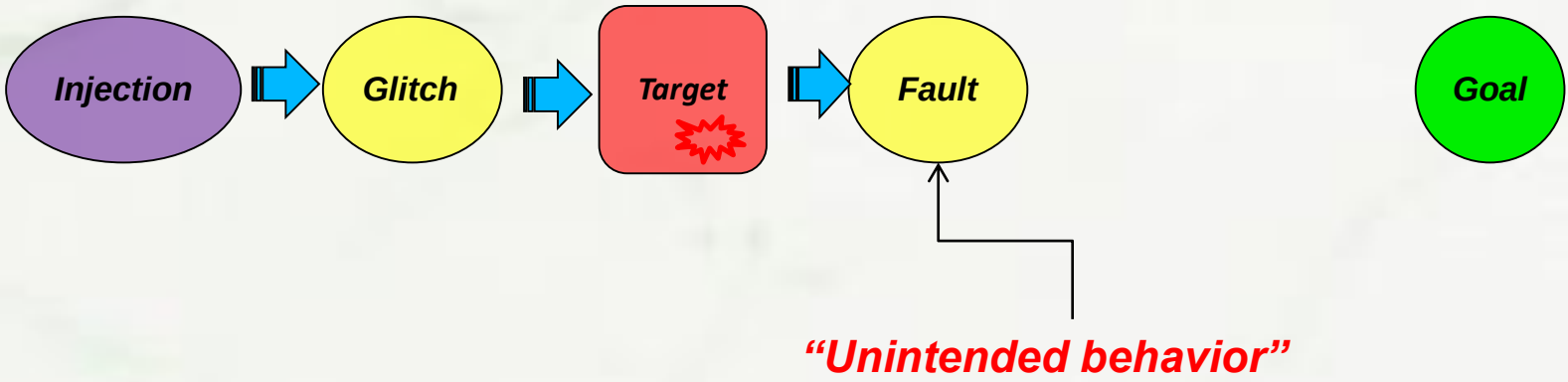
- Where are the *glitch parameters*?
  - Localization (EM-FI, Optical), Duration, Voltage Thresholds,…
- How the *physical effects are turned into a successful attack*?
- Don’t we need SW to *exploit?*
  - Shellcodes, Memory “massaging”, Payloads, Protocols…

Glitch: triggering vulnerabilities



- The *environmental change* for triggering vulnerabilities
- HOW the injection is performed
  - *"Location":* Sub-system, VCC rail, Area, Time window,…
  - *Parameters:* Duration, Intensity, Repetitions,…

Fault

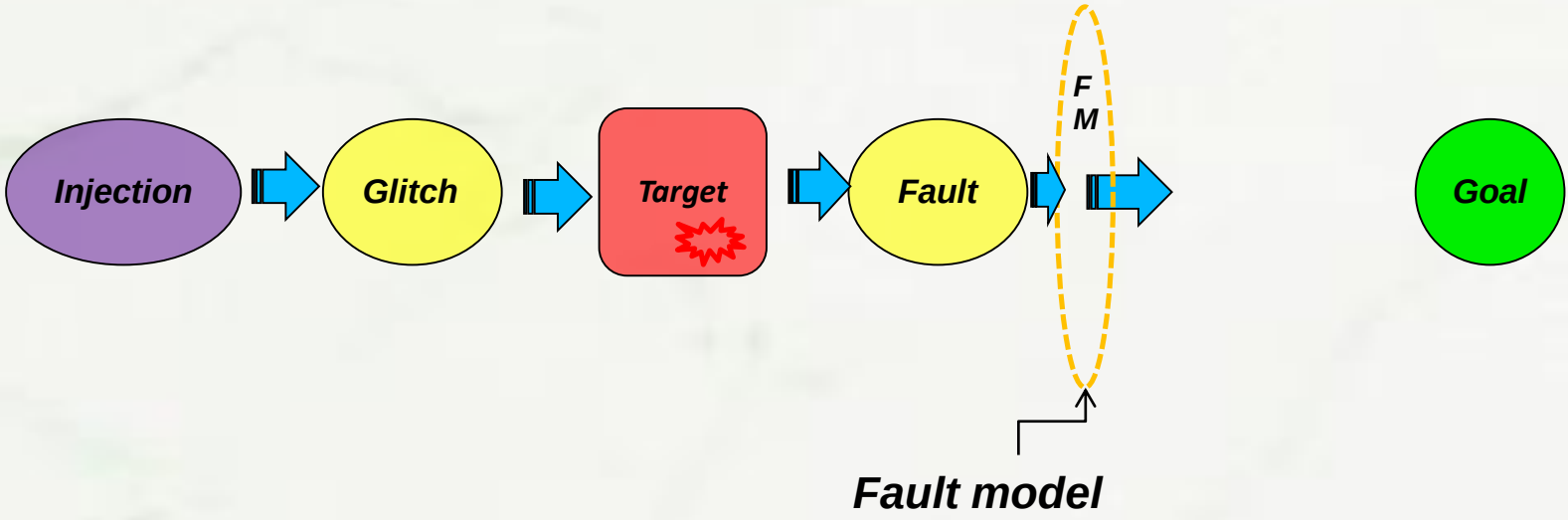


- *Fault:* the "changes" introduced by the glitch
- Can occur at multiple levels
  - *Physical, Logic, Architecture, Software,…*

Not all faults are created equal…

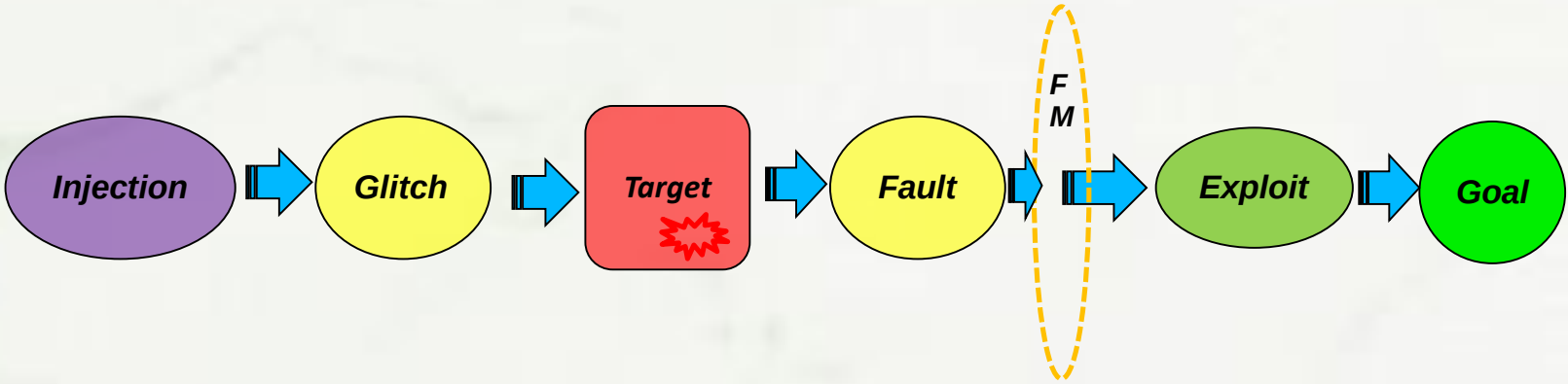
- Multiple different faults may occur, that:
  - Leave the system unstable
  - Have no visible effect
  - Have no "*attack-relevant*" effect
- Which are the "interesting" faults?
  - i.e. that can be leveraged for a successful attack

Fault Model



- *Fault Model:* defines the relevant set of faults
  - Faults that can be leveraged into an exploit
  - E.g.: faults that cause instruction skipping
    - Can be multiple. At multiple levels.

Exploit

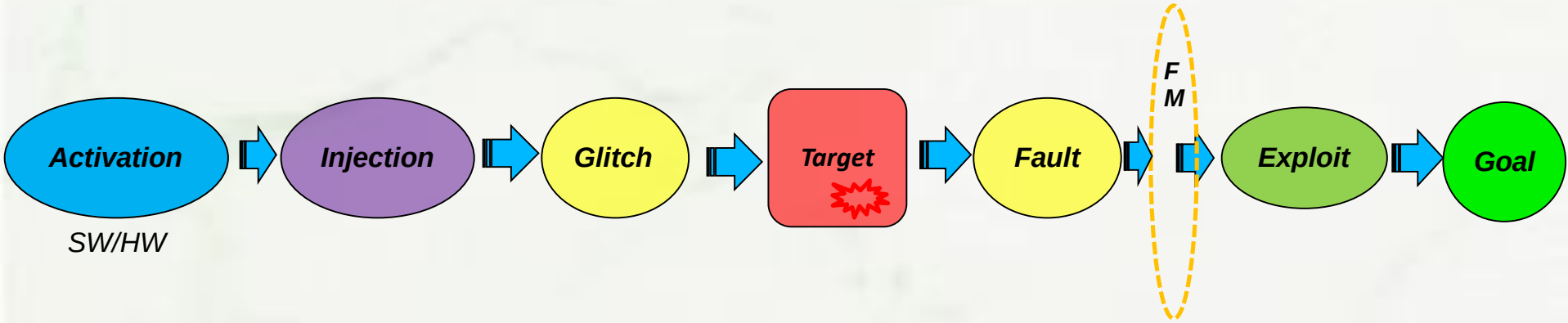


- *Exploit:* leverages faults within the fault model
  - *Preparation:* Image layout, Shellcodes, staging,…
  - *Execution:* Fault in adjacent rows → Memory de-dup → Shared page → Cross-VM attacks

“Activation” (?)

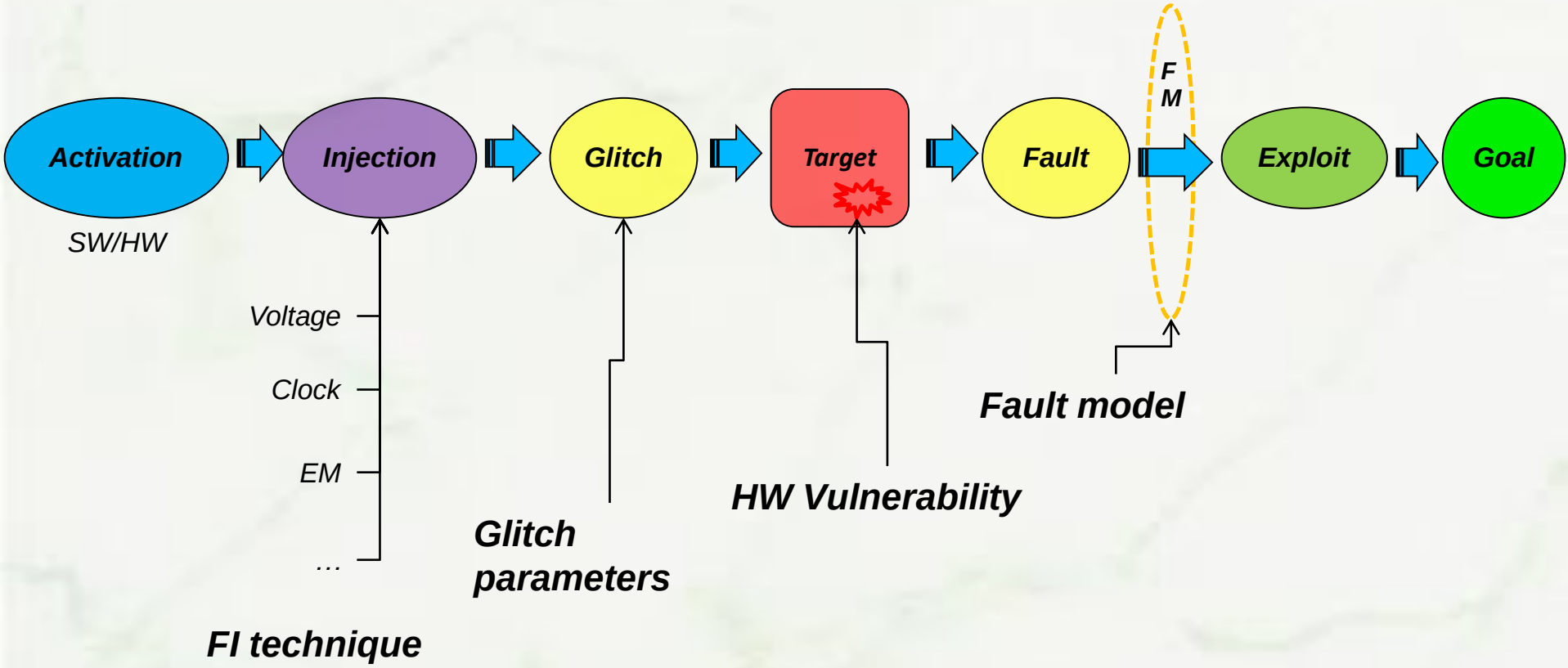
- In some cases injection does not require physical access:
  - SW may be used for “activating” injection
- ***Injection still occurs at physical level***
- “Activation” performed by SW:
  - By controlling HW subsystems/interfaces
- Examples:
  - CLKSCREW:
    - SW manipulates Clock + Voltage *(Injection)* subsystems via DFVS
  - Rowhammer:
    - SW causes EM interference (*Injection)* between memory cells by continuously accessing rows
    - *You can do this from Javascript!*

Activation



- *Activation:* HOW the technique is controlled
  - *HW activation*: Physical VCC/EM/CLK injection, …
  - *SW activation*: via DVFS, SW memory reads, PLCs (Stuxnet?)

A reference model

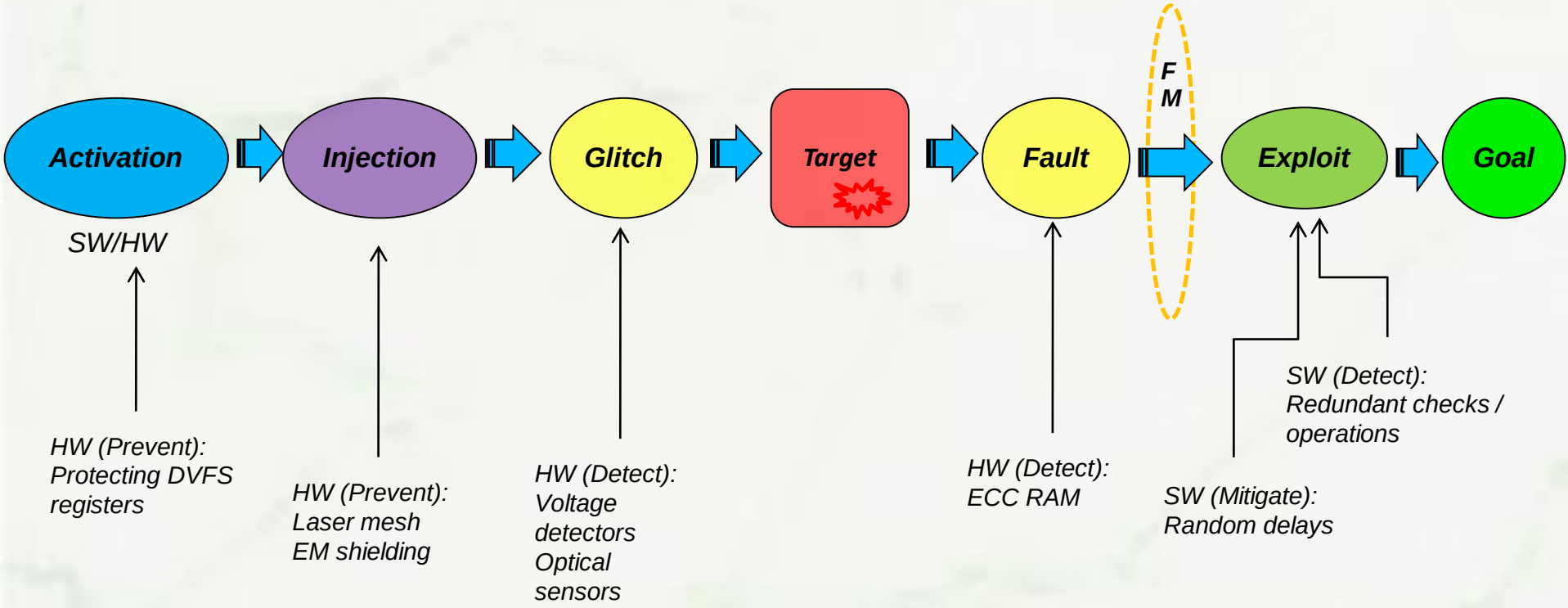


Applications

Possible uses

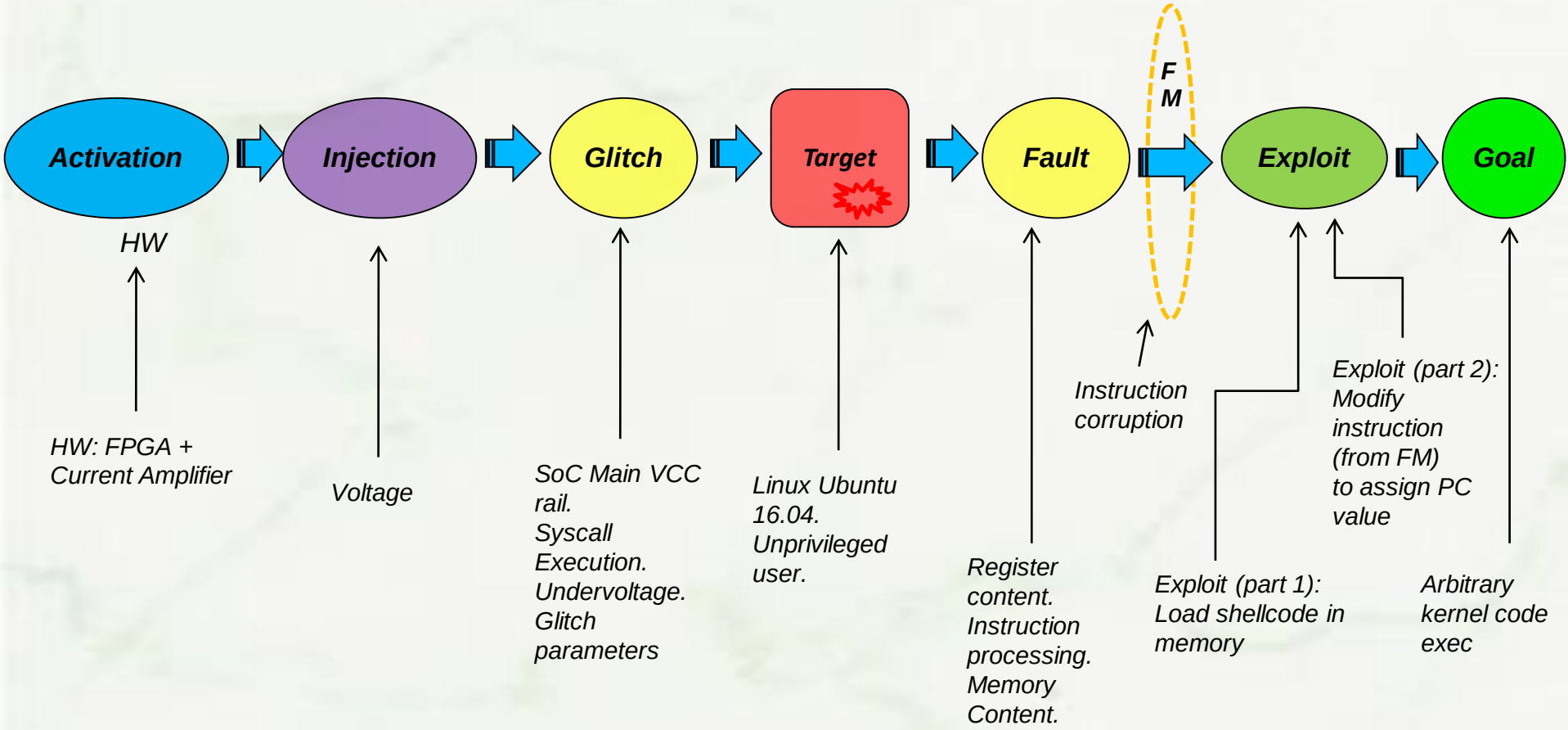
- Support for discussing:
  - Attacks
  - Countermeasures
- Visualization:
  - Attack strategies
  - Thought process
- Identification:
  - Commonalities and differences in attacks/defenses
  - Research trends

Visualizing countermeasures

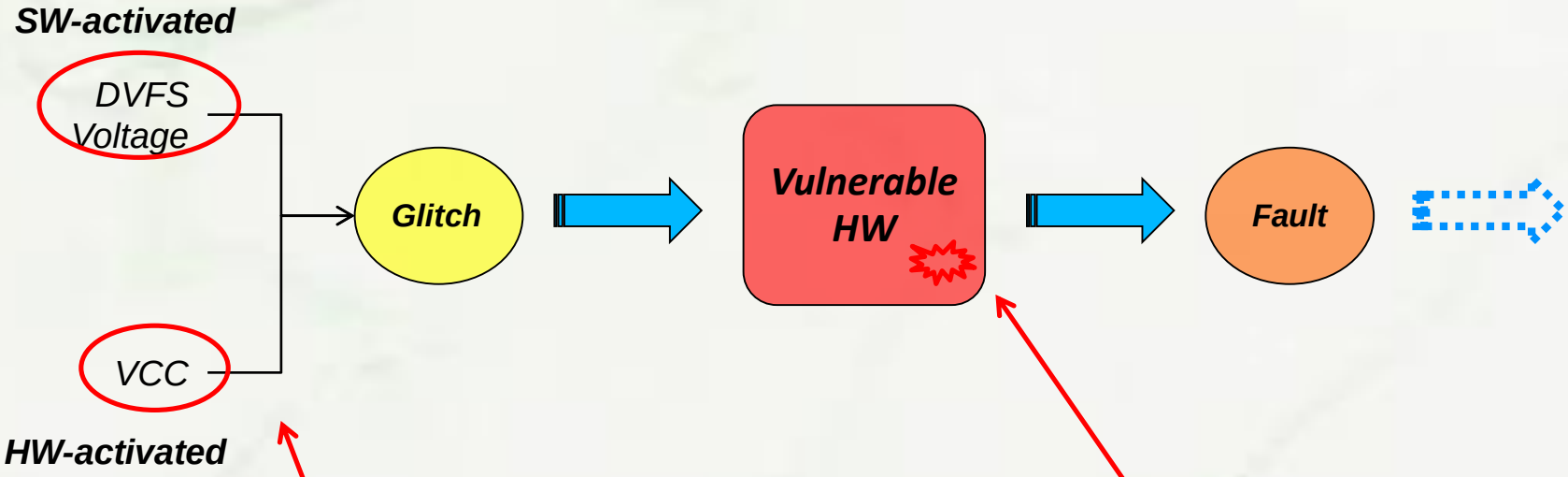


- *Observations:*
  - SW countermeasures: *depend on FM, focus on exploit*
  - HW countermeasures: (mostly) FM independent, focus before exploit

Visualizing attacks: Linux privilege escalation



Creating “new” attacks?



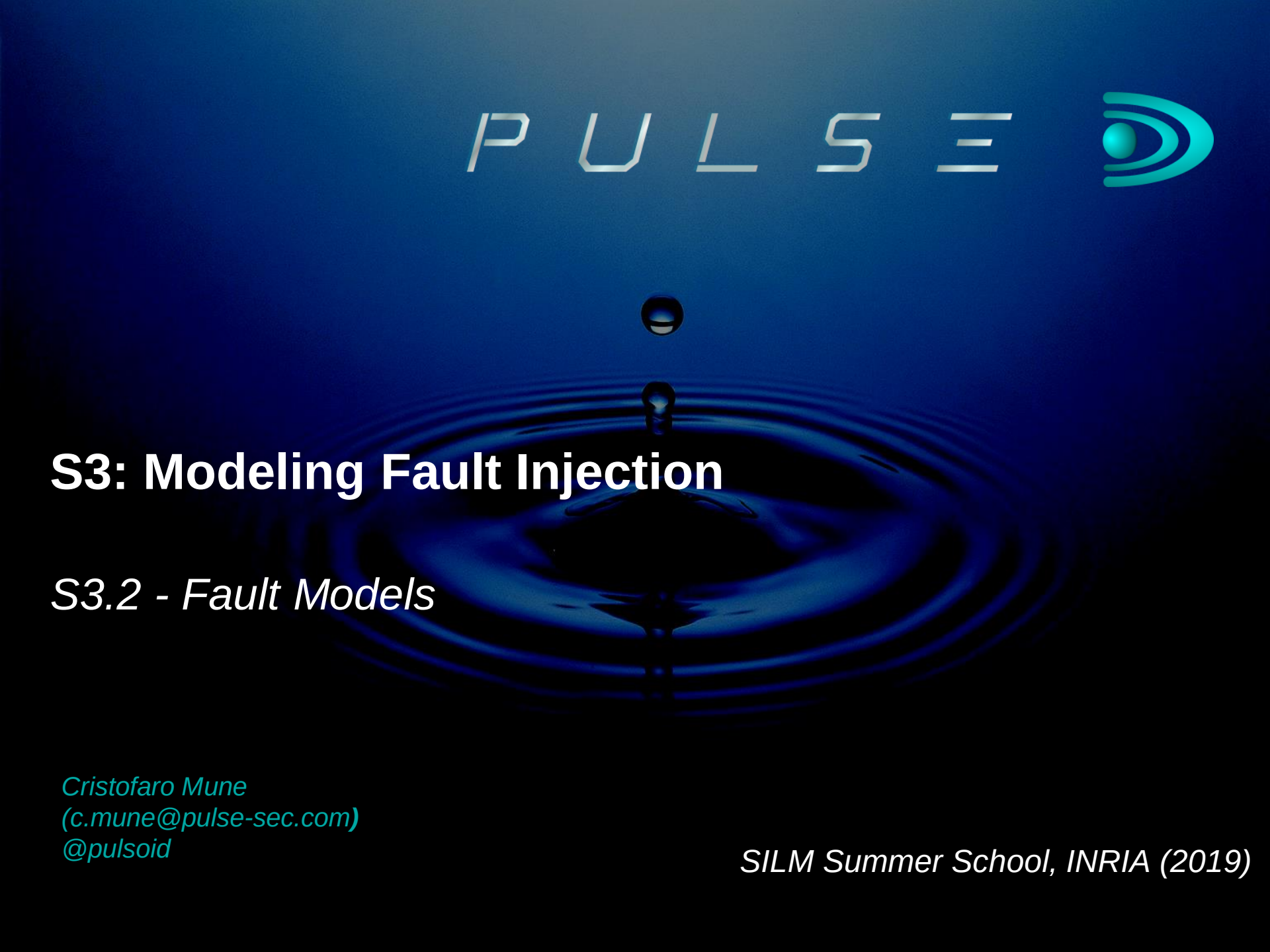
May different activations…

…trigger the ***same vulnerability?***

Conclusions

Conclusions

- FI attacks much more than:
  - Select a technique and..
  - *“Magic happens”*
- Proper modeling may allow for:
  - Improving methodologies
  - Countermeasures design:
    - *Scope*, placement, Relevance/effectiveness
  - *Creating new attacks ‘modularly’*
    - Identify uncovered attack vectors
    - New creative attacks for known injection techniques
  - *Injection-independent attacks*



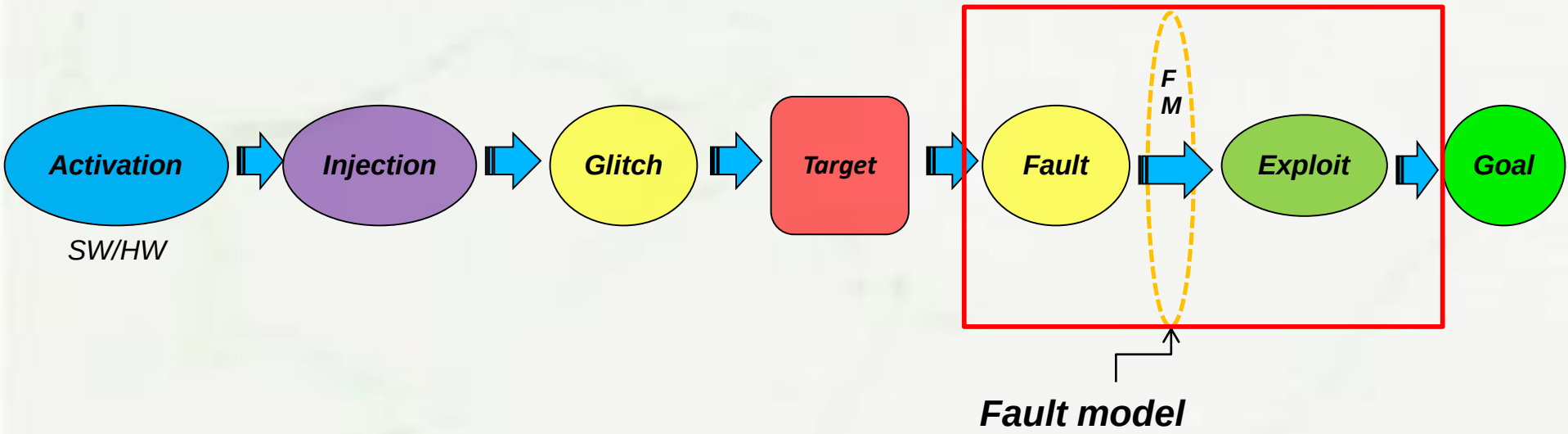
S3: Modeling Fault Injection

S3.2 - Fault Models

Cristofaro Mune
(c.mune@pulse-sec.com)
@pulsoid

SILM Summer School, INRIA (2019)

Question

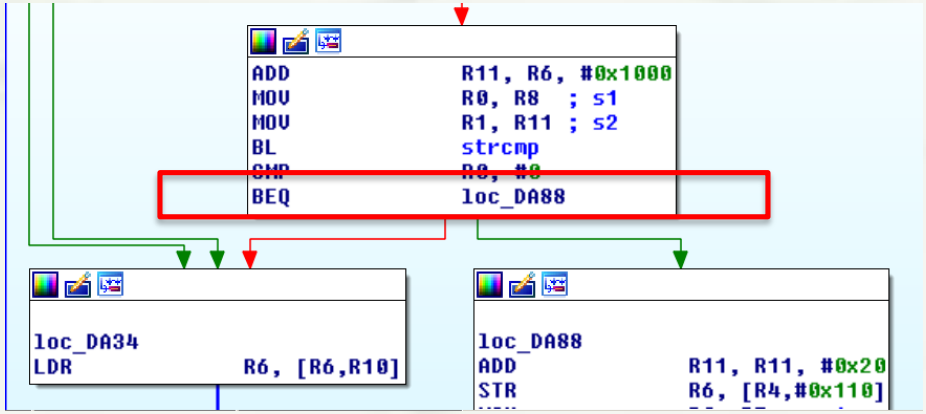


Which faults are useful and exploitable?

Fault Models

Traditional fault models

Control flow corruption
by "skipping instructions"



Data corruption
by "flipping bits"

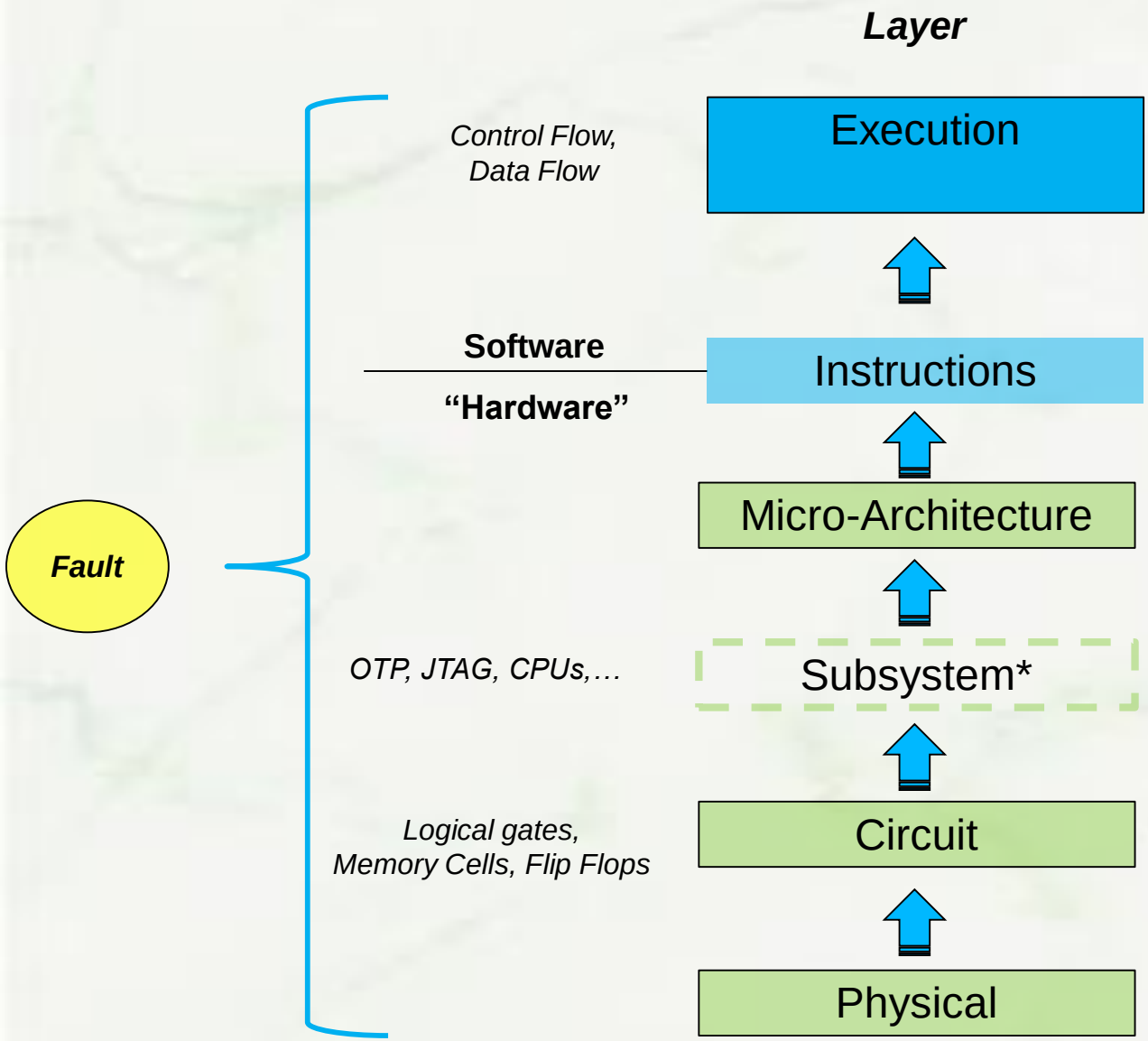
```
000c8420h: D0 EF AA FB 43 4D 33 85 45 F9 02 7F 50 3C 9F A8
000c8430h: 51 A3 40 8F 92 9D 38 F5 BC B6 DA 21 10 FF F3 D2
000c8440h: CD 0C 13 EC 5F 97 44 17 C4 A7 7E 3D 64 5D 19 73
000c8450h: 60 81 4F DC 22 2A 90 88 46 EE B8 14 DE 5E 0B DB
000c8460h: E0 32 3A 0A 49 06 24 5C C2 D3 AC 62 91 95 E4 79
000c8470h: E7 C8 37 6D 8D D5 4E A9 6C 56 F4 EA 65 7A AE 08
000c8480h: BA 78 25 2E 1C A6 B4 C6 E8 DD 74 1F 4B BD 8B 8A
000c8490h: 70 3E B5 66 48 03 F6 0E 61 35 57 B9 86 C1 1D 9E
```

Security Measures bypass

Cryptographic key extraction

One fault model for each attack

Reality: One glitch → Multiple faults



*Extension to [2018]: Yuce, Schaumont, Witteman

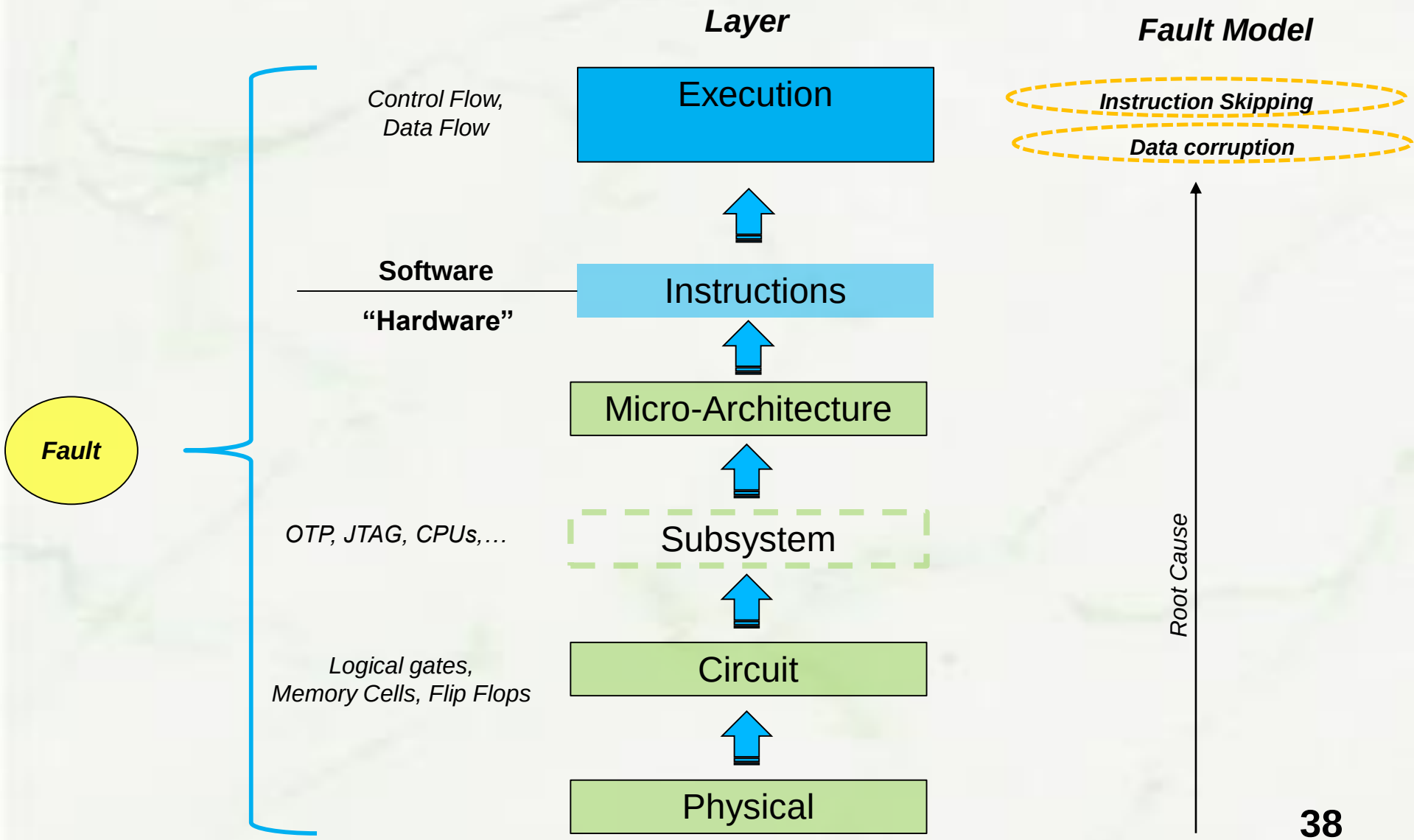
FaultS (!)

- Multiple faults occurs **at the same time**:
  - **With the same technique**
  - *With one single glitch*
  - *At multiple layers and locations in the system*

- *Each fault, at each layer, can be potentially exploitable*

- **Multiple attacks possible at the same time**
  - for the same Goal
  - with the same injection technique
  - *By selecting different [fault model:exploitation]*

Traditional Fault Models



A new fault model

*A generic one: **"instruction corruption"****

Single-bit (MIPS)

```
addi $t1, $t1, 8    00100001001010010000000000001000
addi $t1, $t1, 0    00100001001010010000000000000000
```

Multi-bit (ARM)

```
ldr w1, [sp, #0x8]  10111001010000000010111100001
str w7, [sp, #0x20] 10111001000000000100111110011
```

Remarks

- Limited control over which bit(s) will be corrupted
- Also *includes other fault models as sub-cases* (e.g. instruction skipping)

**[2016]: Timmers, Spruyt, Witteman*

Controlling PC (or SP)

- ARM32 has an interesting ISA
- *Program Counter (PC) is directly accessible*

Valid ARM instructions

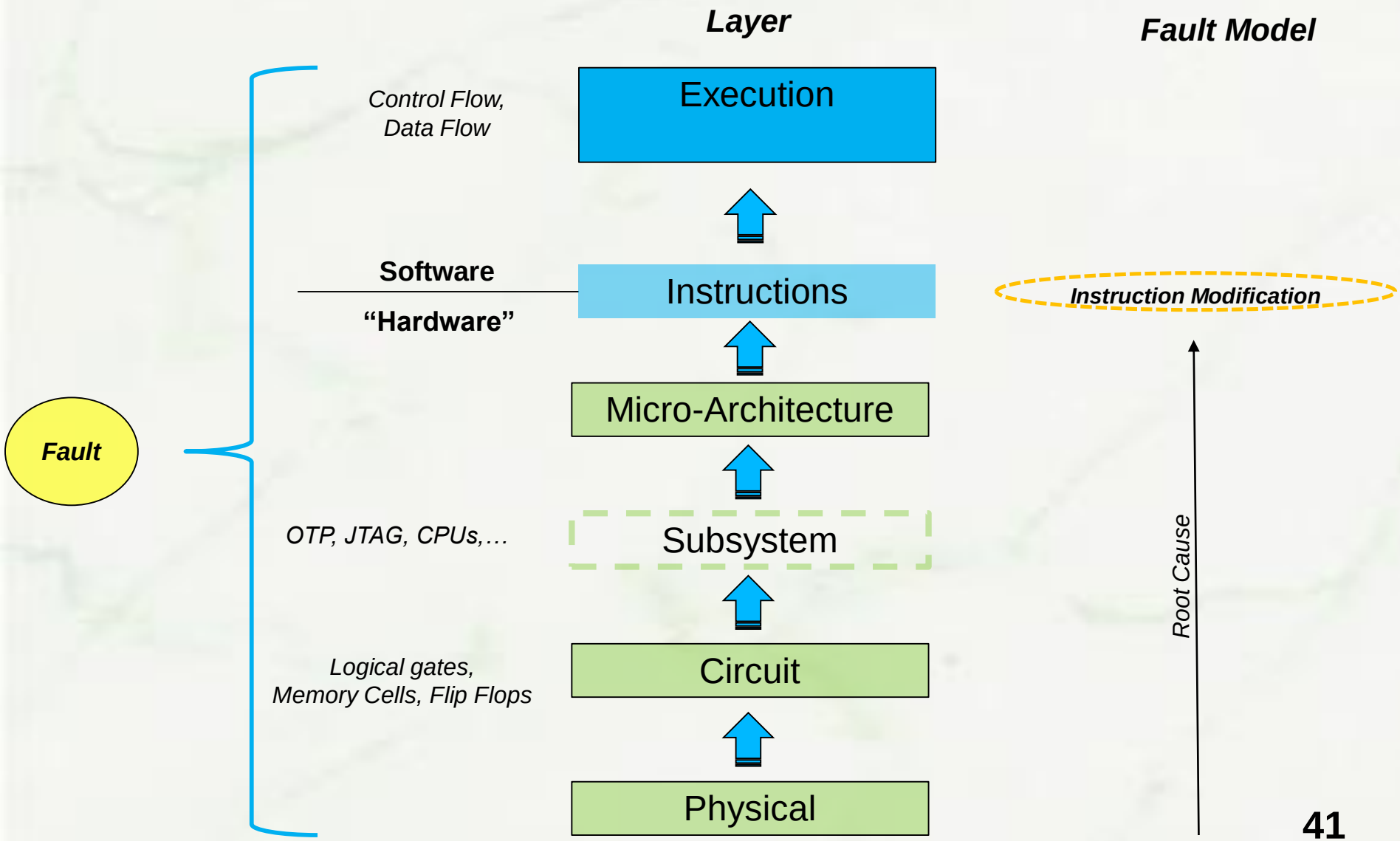
```
MOV r7,r1          00000001 01110000 10100000 11100001
EOR r0,r1          00000001 00000000 00100000 11100000
LDR r0,[r1]        00000000 00000000 10010001 11100101
LDMIA r0,{r1}      00000010 00000000 10010000 11101000
```

Corrupted ARM instructions may *directly set PC

```
MOV pc,r1          00000001 11110000 10100000 11100001
EOR pc,r1          00000001 11110000 00101111 11100000
LDR pc,[r1]        00000000 11110000 10010001 11100101
LDMIA r0,{r1, pc}  00000010 10000000 10010000 11101000
```

Attack variations (SP-control) also affect other architectures

Fault Model


Layer
Fault Model
Control Flow,
Data Flow
Execution
Software
"Hardware"
Instructions
Instruction Modification
Micro-Architecture
Fault
OTP, JTAG, CPUs,…
Subsystem
Root Cause
Logical gates,
Memory Cells, Flip Flops
Circuit
Physical

Execution primitives…out of thin air

- ***ANY memory read*** can be *redirected to PC (or SP)*
  - *ANY memcpy()*
- ***PC*** *(or SP)* ***immediately assigned*** *with content from memory*
  - *Exploited* ***DIRECTLY*** *from Instruction Layer*

- *Following SW NOT executed*
  - ***SW countermeasures fully bypassed***

- ***A new target for FI:***
  - Security checks
  - Crypto algorithms
  - ...

Code Execution

Application: Kernel exec via FI

[User]: All registers set to target memory address

[User]: Syscall

[Kernel]: MOV instruction modified (operand)

[Kernel]: PC becomes destination register

Arbitrary PC control

…more to come…

(session 4)

Using Fault Models

Fault models: the Analytical approach

- Fault model focuses on *actual* effects:
  - Attempts to model *real faults* as accurately as possible
  - **Attempts to explain the '*how*' *and* '*why*' of the fault** for:
    - *understanding* fault propagation
    - *inferring* system behavior
  - *System complexity limits this approach*

HOW and WHY do faults happen?

Fault models: a Descriptive approach

- Fault model *assumes interesting* effects:
  - ***Measurements*** performed for verification
  - Focus only on '***if***' fault occurs and '***how often***'
  - Limited support for root cause identification:
    - No attempt to establish causation
  - *But provides exactly what is needed for an attack…*

THIS happens. With THIS frequency

Practical Exploitation

- Assume a fault model for a *specific layer*:
  - E.g. Instruction corruption
    - (for SP control)
- Create an exploit, assuming that such faults occur:
  - E.g. ROP based exploit
- ***Detect success***:
  - Measure frequency and useful related data
- If you have:
  - ***ONE*** success, you have an attack.
  - ***Multiple*** successes, you may have a **predictable attack.**

ONE may be enough

From an attacker perspective…

- Fault root cause is irrelevant
- ***Injection technique is*** *(mostly)* ***irrelevant***
  - *Only the actually introduced faults are*
- Understanding the fault propagation is irrelevant
- What happens at other layers is irrelevant

- *Attack relevant*:
  - System remains stable
  - Fault within chosen fault model, shows up in the desired layer
  - Exploit triggered
  - Success rate

In summary…

*Fault model predicts faults **actually happening** at the chosen layer?*

YES

Attack can be performed.

Considerations

Detecting success?

- Detection usually happens after exploit:
  - Not at fault occurrence

- Detection **DEPENDS *on exploit side effects***
  - Which **DEPENDS *on Fault Model***

- Important results may be missed if detection not aligned with fault model
- Example:
  - We may have been missing decades of instruction corruptions.
  - Detection was only focused on results of *“instruction skipping”* attacks…

Success rate…as attack effectiveness?

- Two attacks:
  - A) 1% success rate, 10 attempts per minute
  - B) 0,1% success rate, 1000 attempts per minute
- ***Which is better?***
  - A) yields success after 10 minutes in average
  - *B) yields success after 1 minute in average*
- **Success rate has no attack effectiveness meaning:**
  - Only gives fault frequency
- Better:
  - *Complement with attack speed*, or
  - Provide *average time for success*

Conclusions

Modeling

- ‘Offensive’ fault models:
  - Focus on measured effects only
  - Do not require understanding physics and system-level
  - Allow exploitation of faults at multiple layers

- New attacks may be possible:
  - With the same injection technique
  - By assuming a different Fault Model

Observations

- *“Make sure to detect successful attacks.”*
  - Detection depends on Fault Model and Exploit
  - You may be missing so many interesting faults…for years.

- *“Success rate alone is insufficient for assessing attack effectiveness”:*
  - Provide attack speed or average time for successful attacks

- ***“Faults are enabling attacks. Not how they are injected”:***
  - Attacks may be designed independently of injection techniques

Contacts

Cristofaro Mune

Product Security Consultant

c.mune@pulse-sec.com