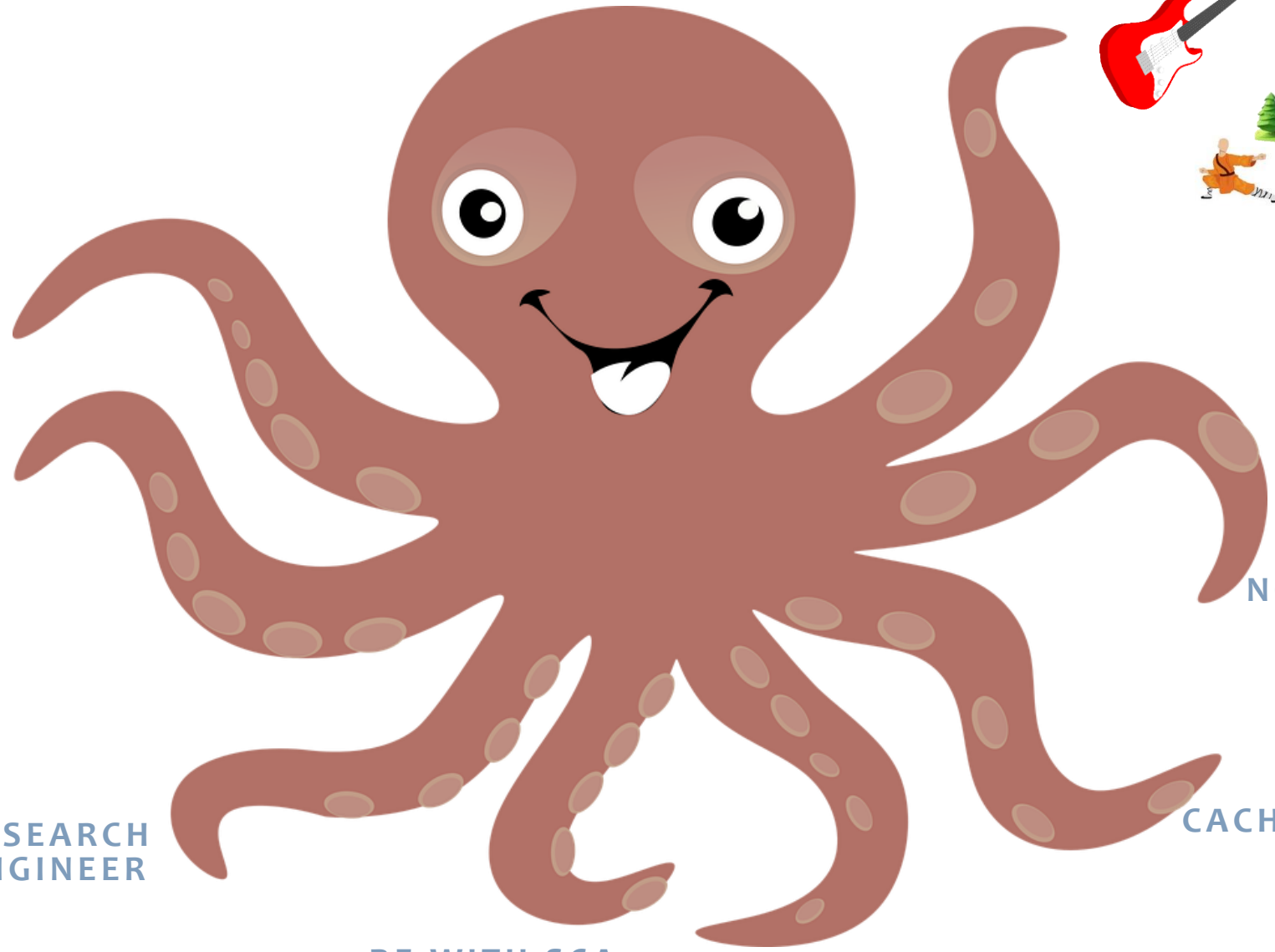
Université de Limoges

SIDE-CHANNEL BASED DISASSEMBLY ON COMPLEX PROCESSORS: FROM MICROACHITECTURAL CHARACTERIZATION TO PROBABILISTIC MODELS

Julien Maillard [1], Thomas Hiscock [1], Maxime Lecomte [1], Christophe Clavier [2]

1. CEA LETI, Université Grenoble Alpes, France
2. XLIM, University of Limoges, France

$ whoami



RESEARCH ENGINEER

RE WITH SCA

PROBABILISTIC SCA ATTACKS

CACHE ATTACKS

NON VOLATILE MEMORY FORENSICS

Context

- **Embedded devices** everywhere (approximately 19 billion today)[1]
- These devices manipulate secrets:
  - Cryptography
  - Personal data
  - Intellectual property

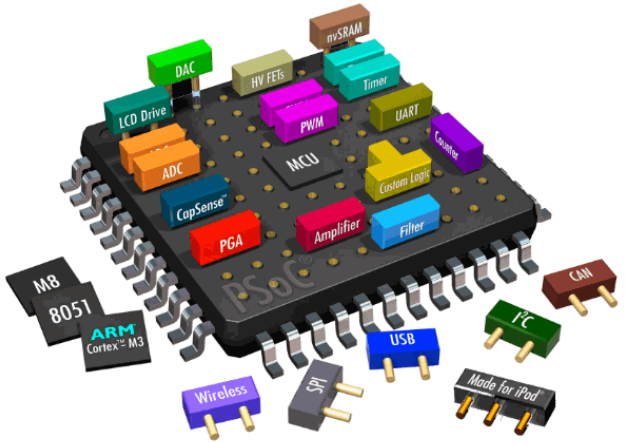
- Malicious actors want to retrieve these secrets
- Need to study an quantize the threat: **cybersecurity**

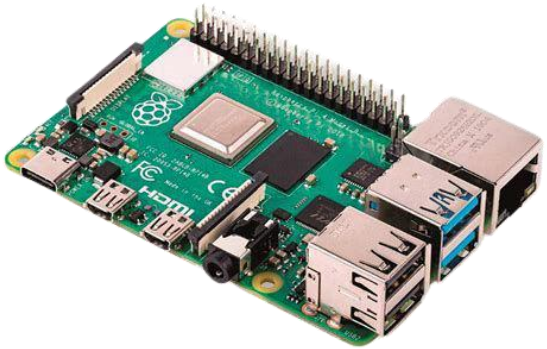
1 : iot-analytics.com

System-on-Chip

- Many features packed on the same chip

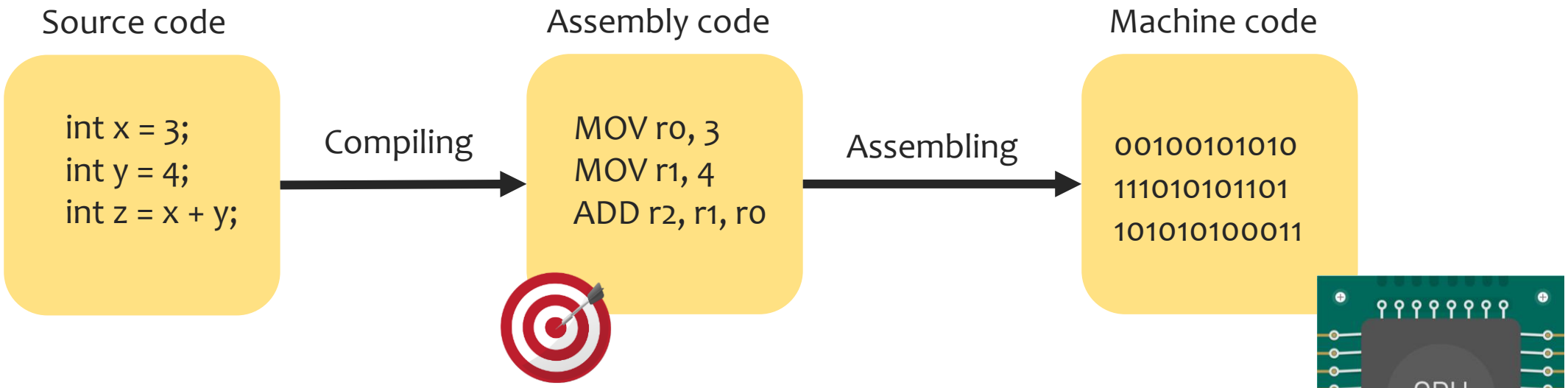
- Ubiquitous in embedded systems:
  - Automotive
  - Avionics
  - Smartphones…





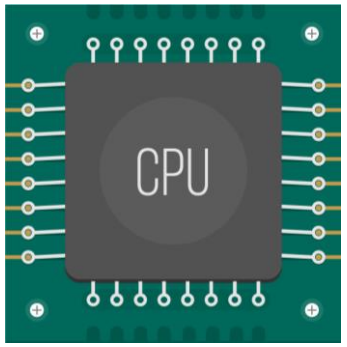
SoC image source: https://microcontrollerslab.com/system-on-chip-soc-introduction/

CPU & Assembly code

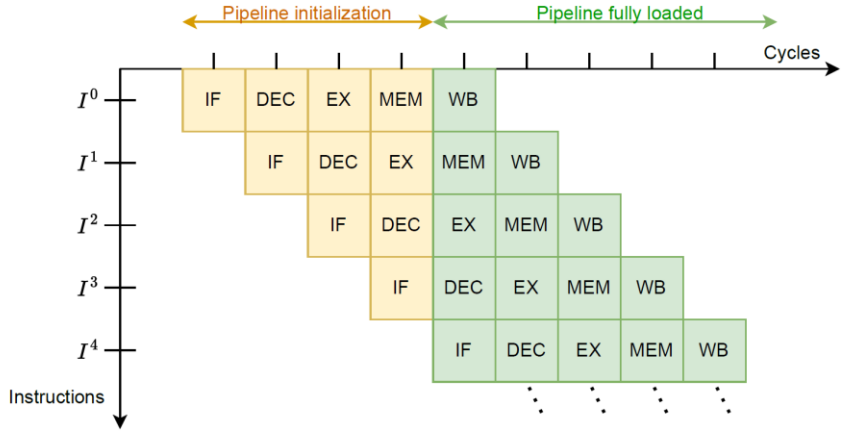


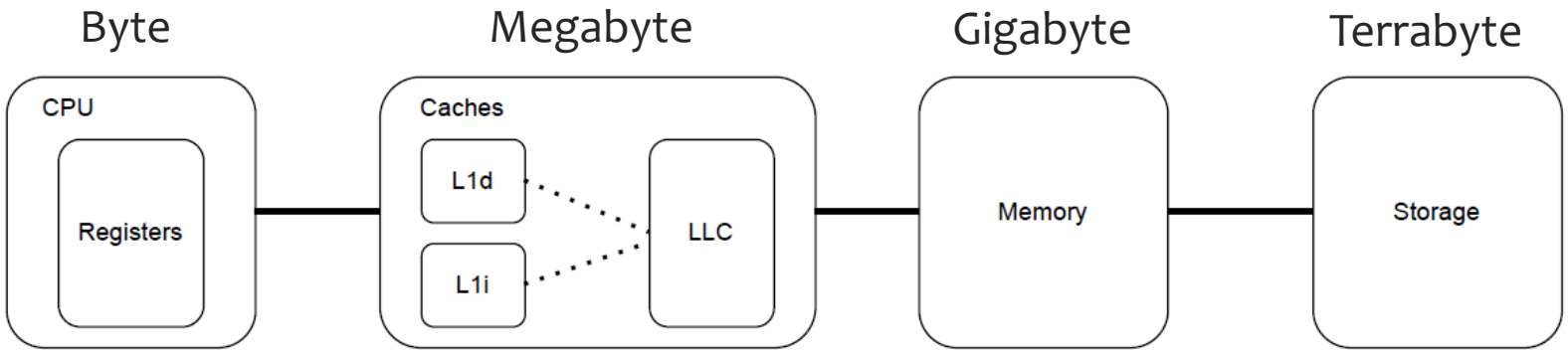
- A processor implements an Instruction Set Architecture (ISA)
- RISC-V, X86, ARM…
- CISC and RISC

CPU features



- Memory hierarchy
- Functional units (ALU, Load/Store Unit, …)
- Pipelining
- Branch prediction
- Program flow optimizations



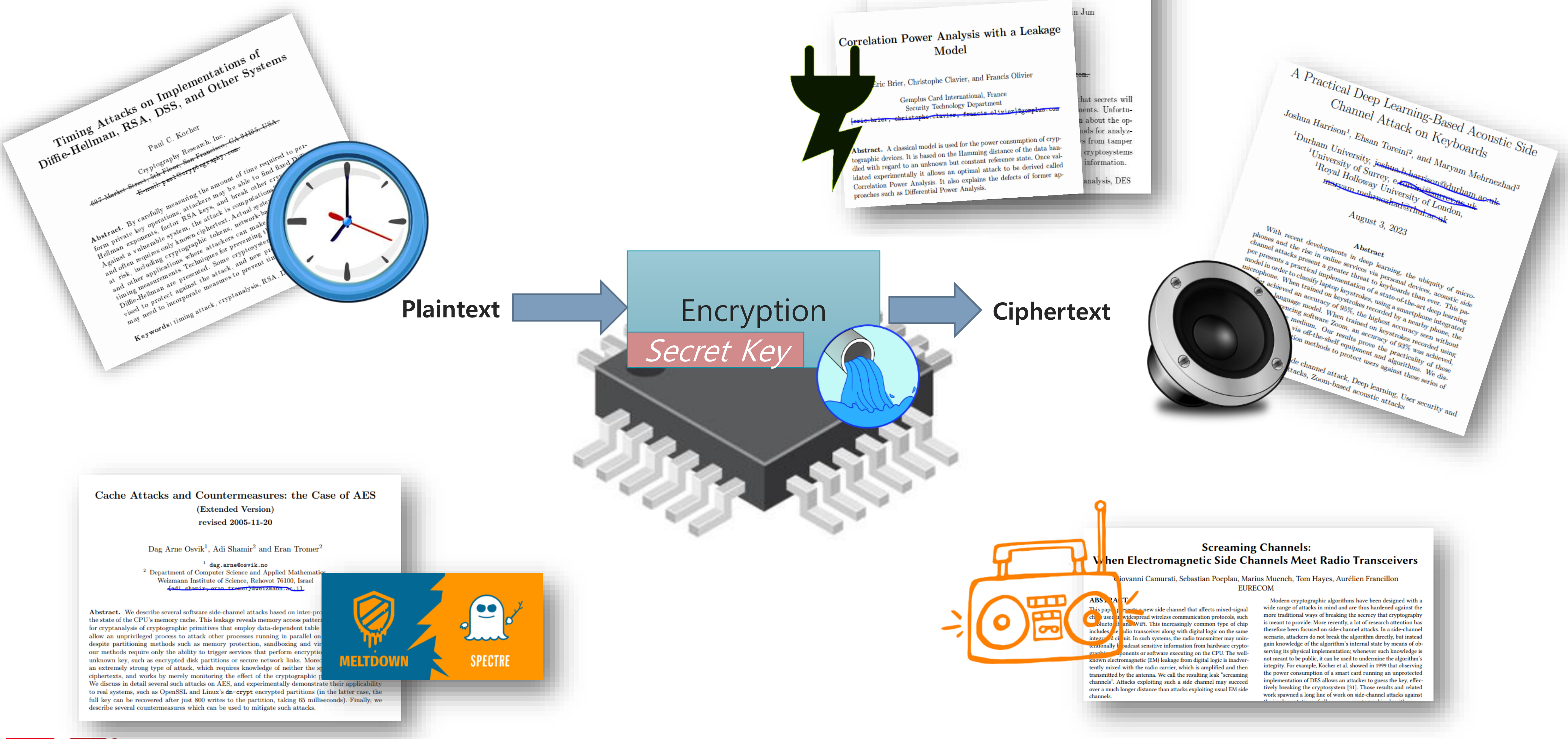


Side-Channel Attacks: general idea

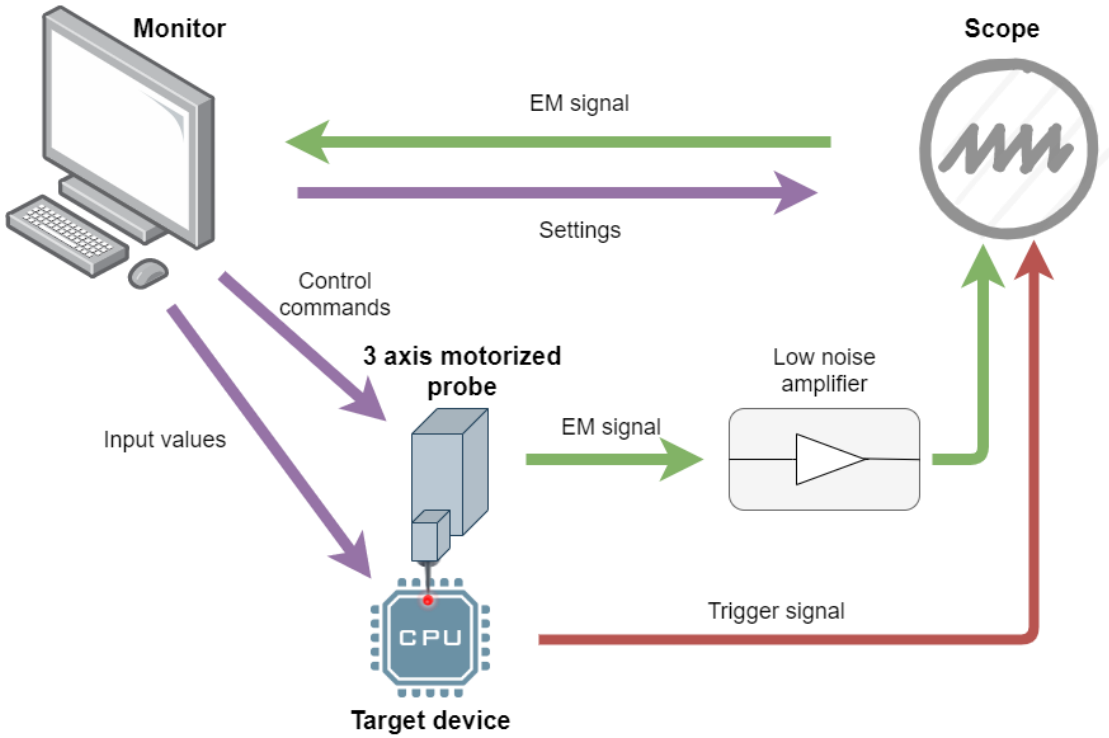
- Specification:
  - 4 digit password
  - 12^4 = 20 736 possibilities

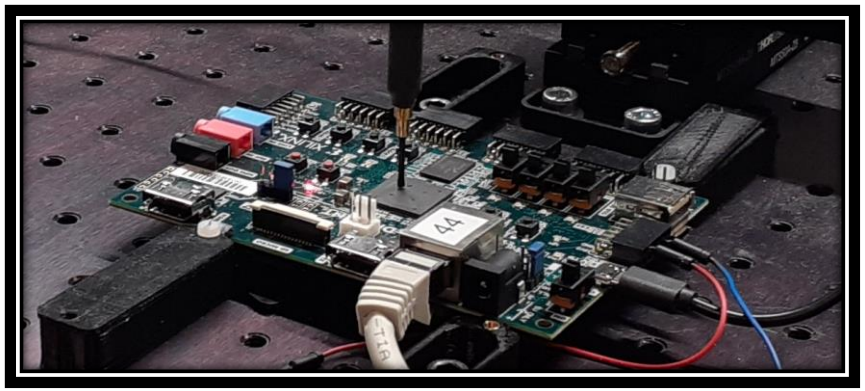
- Reality:
  - 4! = 24 possibilities

Side-Channels



Electromagnetic SCA





- Pros:
  - Non invasive
  - Allows local measurements
- Challenges:
  - Probe placement is hard
  - Long measurement campaigns

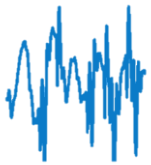
SCBD: motivation with toy example

Algorithm 1: Password check toy example.

Input: input $\in \mathcal{B}^5$, password $\in \mathcal{B}^5$

Result: state correct $\in$ {true, false}

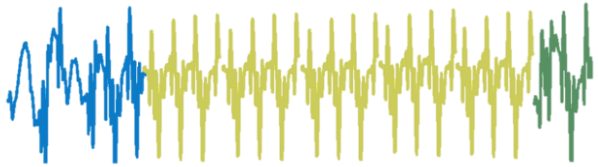
```
1  valid_chars ← 0
2  dummy ← 5
3  state ← false
4  for i from 1 to 5 do
5      if input[i] = password[i] then
6          valid_chars ← valid_chars + 1
7      else
8          dummy ← dummy − 1
9  if valid_chars = 5 then
10     state ← true
11 return state
```



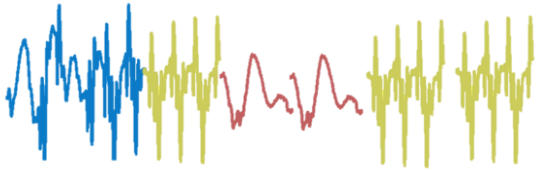
Can I gain information on the code by observing side-channel leakage?

: pizza

Input: pizza



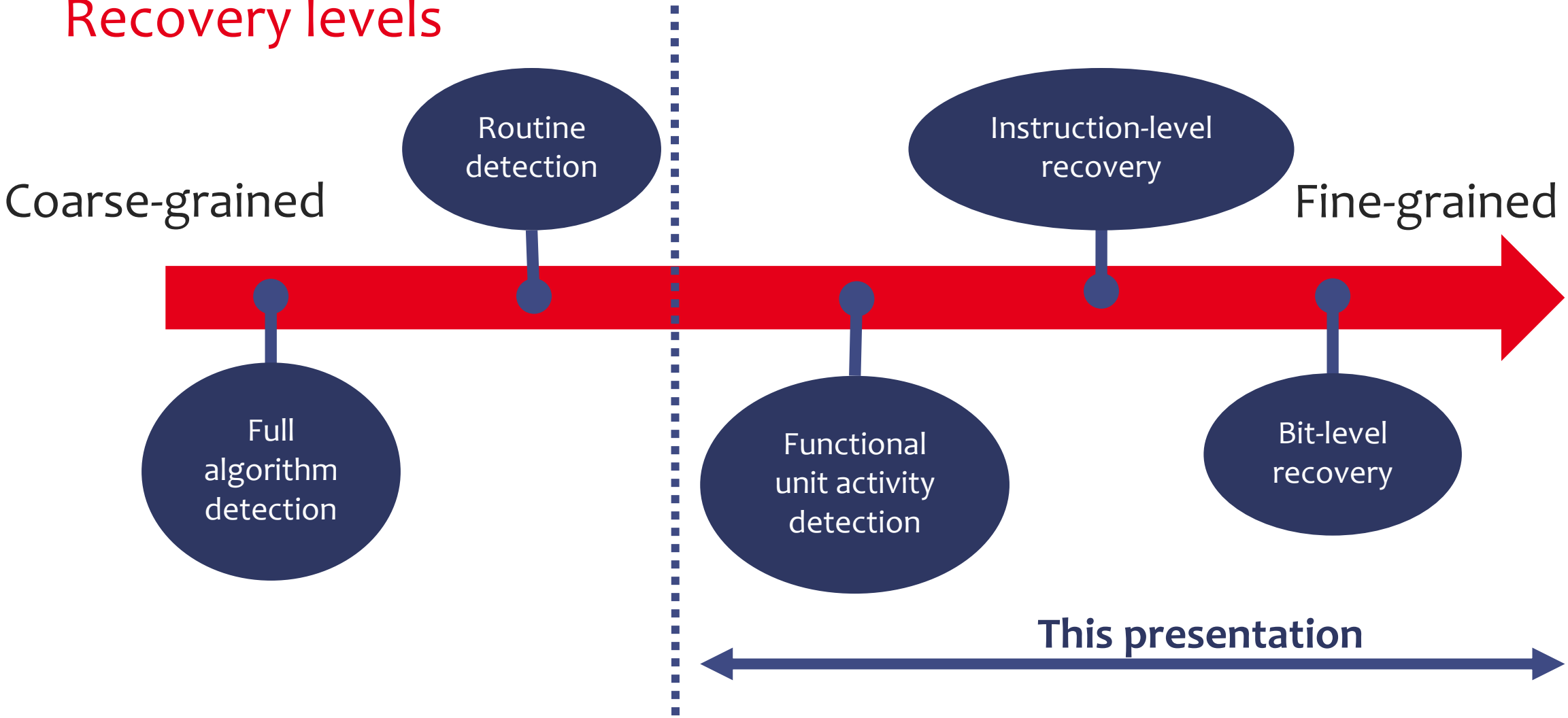
Input: polza



SCBD vs SCA against cryptography

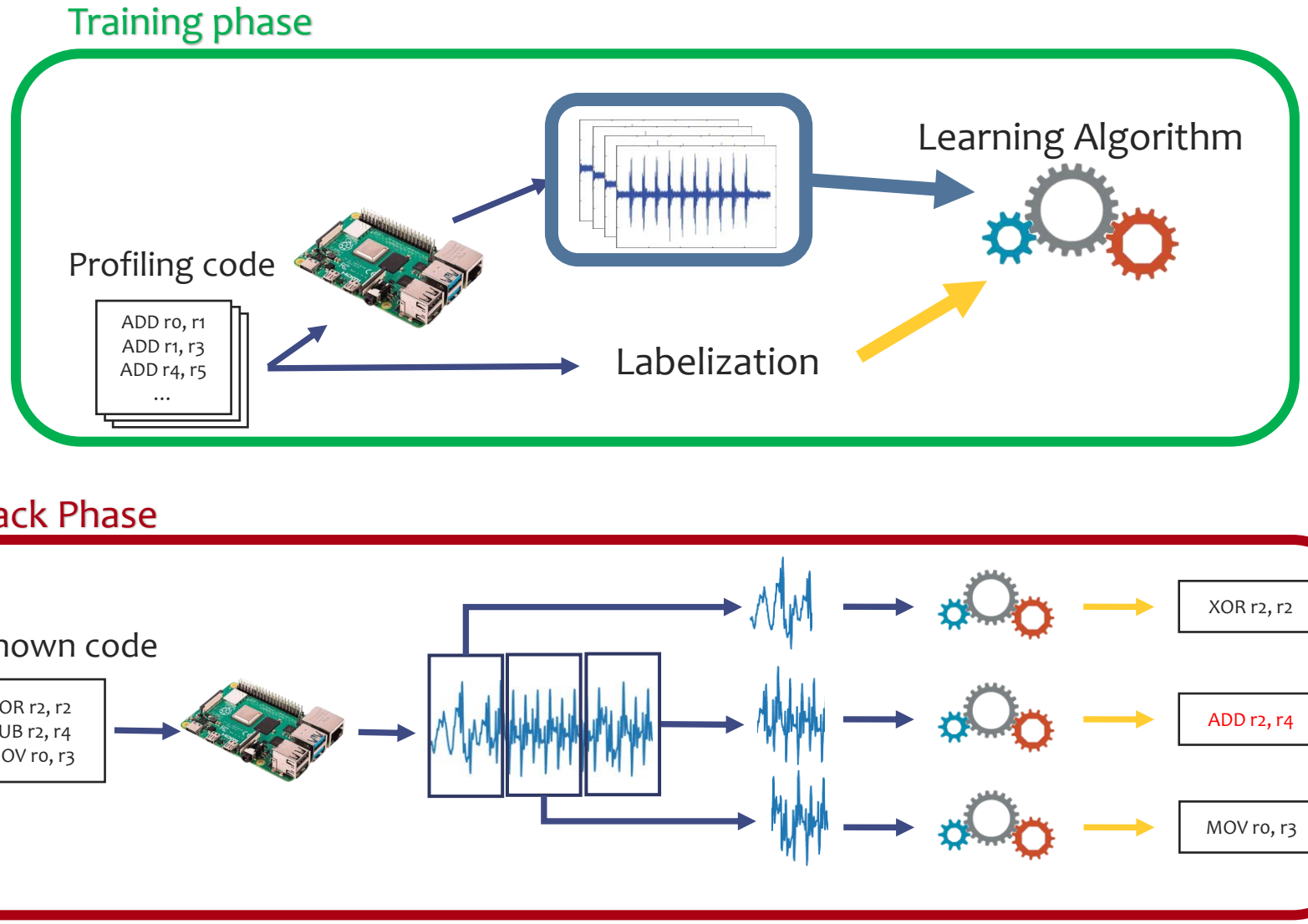
- Crypto SCA:
  - Targets cryptographic keys
  - Powerful non-profiled attacks
  - Randomly sampled secrets
  - Mostly varying inputs
- SCBD:
  - Detect RoI in traces, reconstruct code
  - Often profiled attack
  - Dependent target variables
  - Low variability
  - High repetability

Recovery levels

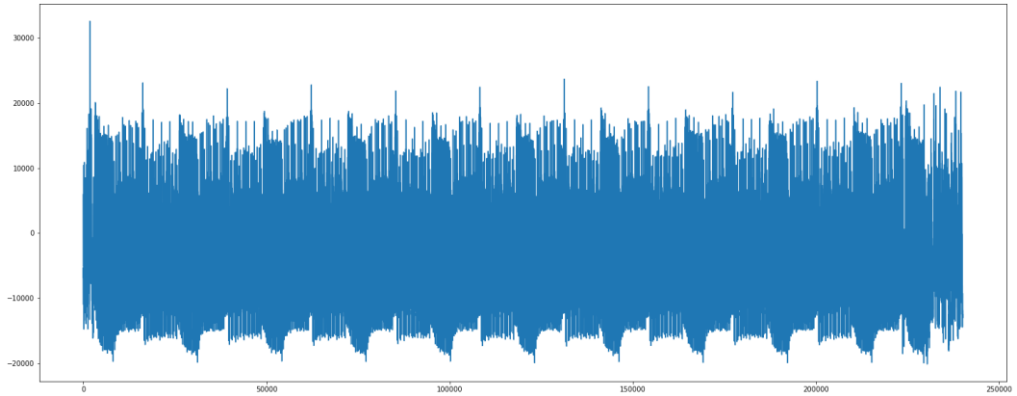


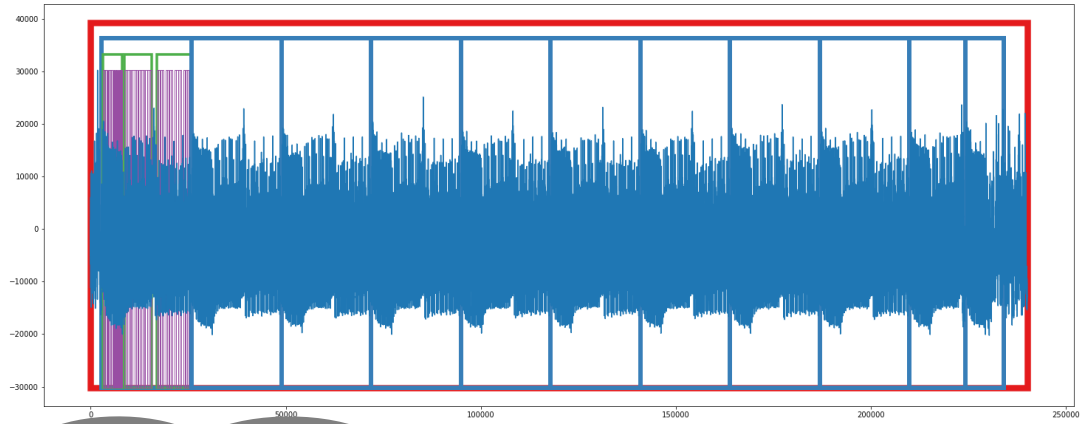
What recovery levels are achievable on a SoC?

SCBD: General Idea

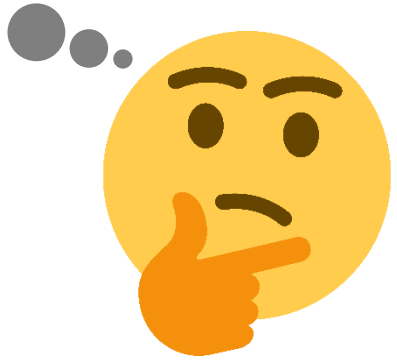


SCBD Paradigms: Coarse-grained & routine detection



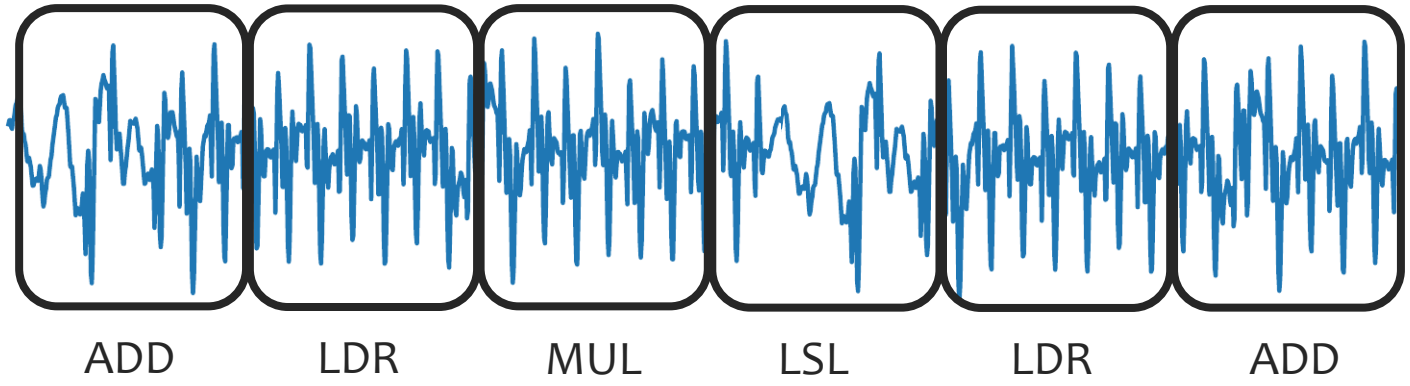


10 rounds, 4 routines per round, 16 times the same pattern per routine…



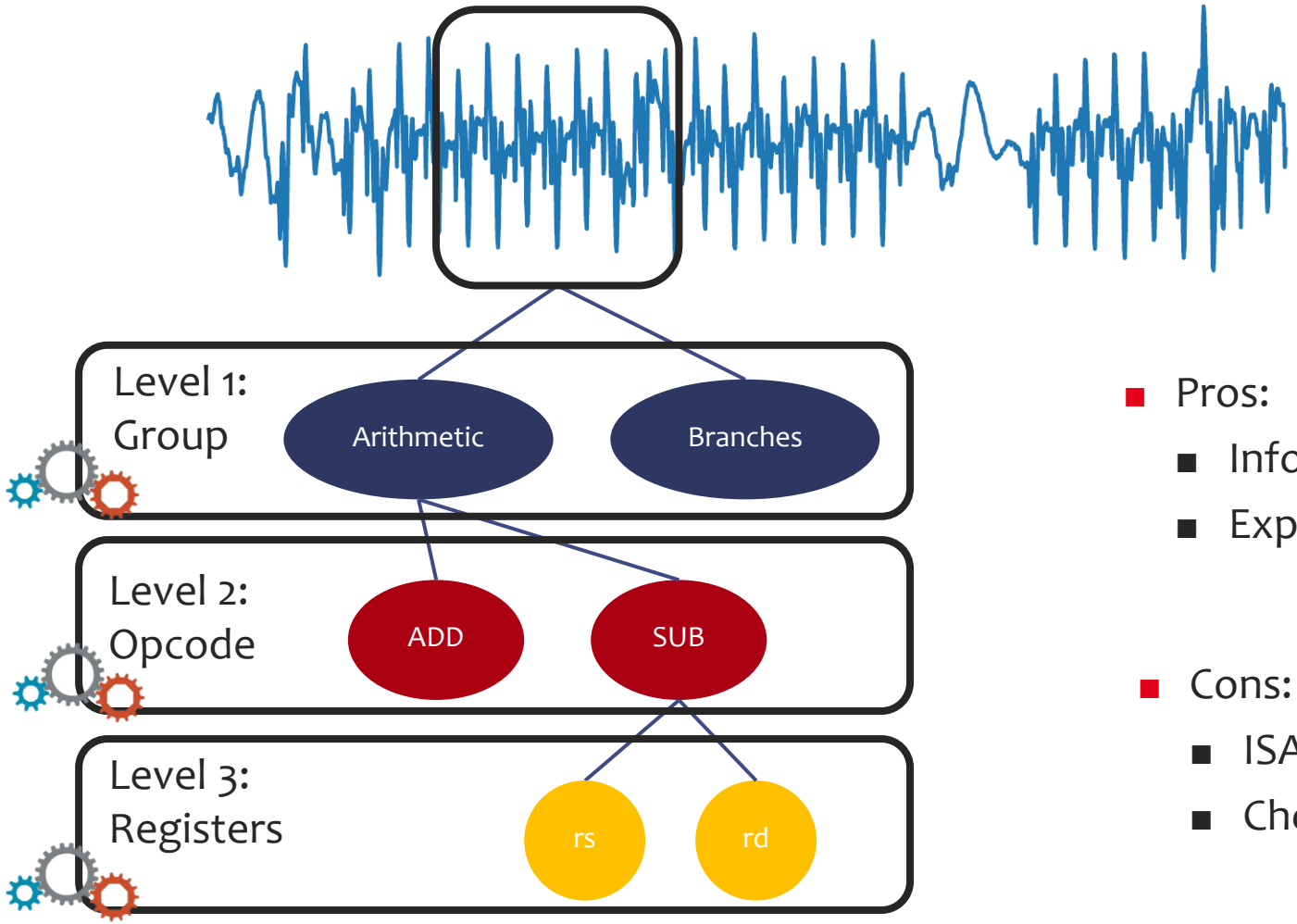
It’s an AES !

SCBD Paradigms: Opcode-level



- Pros:
  - Exploits the pipeline front-end and middle-end leakages
- Cons:
  - ISA-specific
  - Partial information on the instruction
  - Needs to manipulate many classes

SCBD Paradigms: Hierarchical



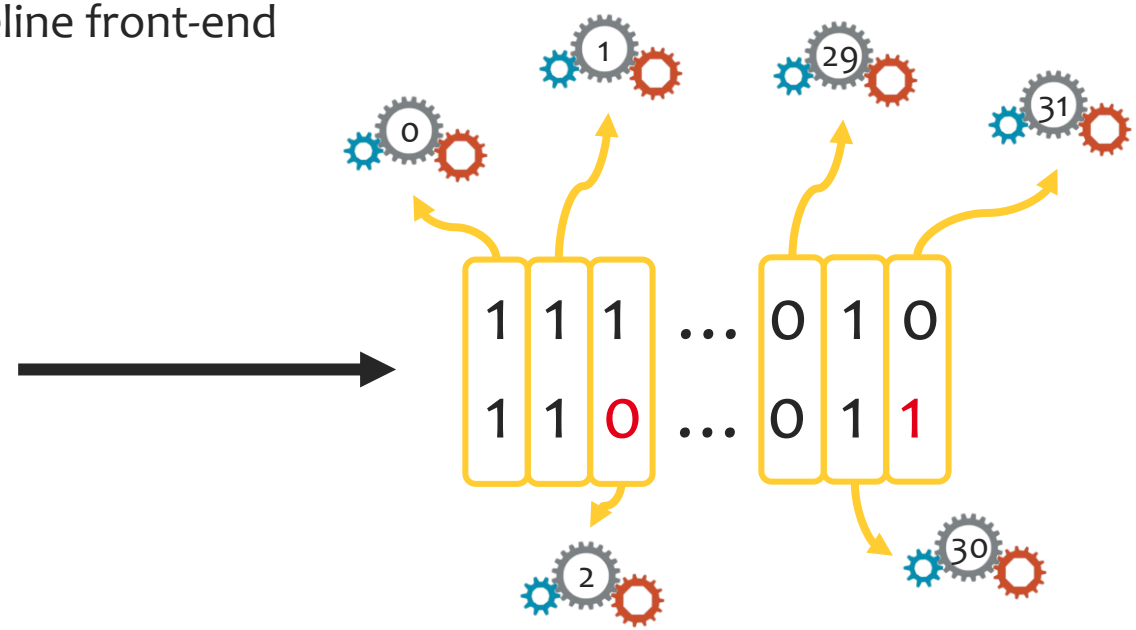
- Pros:
  - Information is gained, even at top level
  - Exploits leakages in all the pipeline
- Cons:
  - ISA-specific, with possibly limited coverage
  - Choices are made on each branch of the tree

SCBD Paradigms: Bit-Level

- Focuses on machine code representation
- (Mostly) exploits leakages in the pipeline front-end

ADD r0, r1, r2

SUB r1, r1, r2



- Pros:
  - Few dependent on the ISA
  - Scalable to large ISAs
  - Easier to generate training datasets
- Cons:
  - Front-end pipeline leakages can be slight
  - Can output invalid instructions

SCBD: Existing work

Paper	Granularity	Device	Method	Accuracy (Test code)	Accuracy (real code)
[VWG07]	Bytecode	Java Card	Pattern Matching	>90%	
[EPW10]	Opcode	PIC16F	LDA + HMM	71.1 %	50.8%
[MMM14]	Opcode	ATMEGA163	PCA + KNN	100% (39 selected opcodes)	
[Str+15]	Opcode	PIC16F	LDA, KNN, multi-positions	96.4%	87.4%
[Par+18]	Hierarchical	ATMEGA328p	CWT + KL-divergence	99.03%	
[VMA20]	Hierarchical	PIC16F	CWT + PCA, random forest	99.5%	93.3%
[VMA20]	Hierarchical	ARM Cortex-M3	CWT + PCA, random forest	98%	80.2%
[Fen+22]	Hierarchical	32-bit RISC-V	Sparse dictionary learning and MLP	93.01% (simulation)	93.16%
[CLH19]	Bit-level	PIC16F	LDA + QDA	95%	
[Fen+22]	Bit-level	32-bit RISC-V	MLP	95.16% (simulation)	

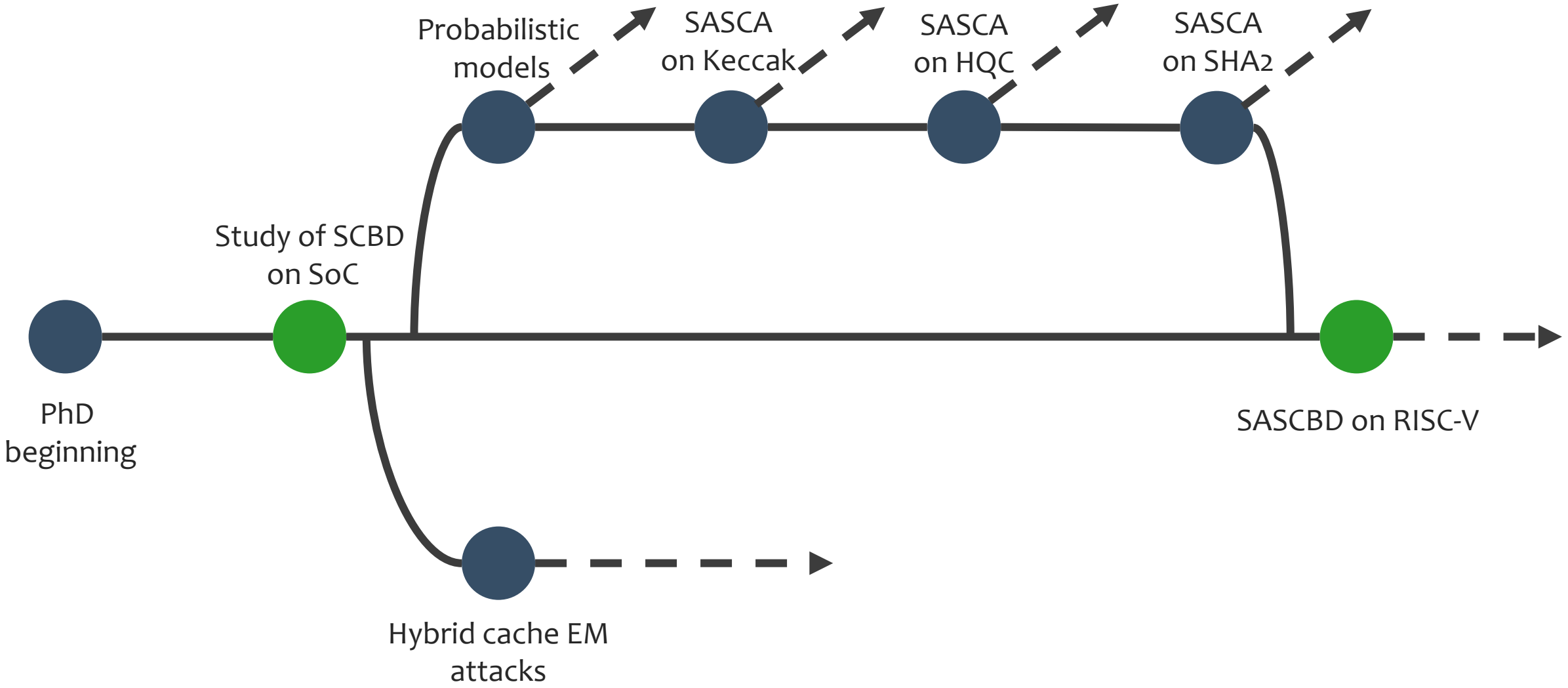
Research Questions

- Remarks:
  - All methods have their pros and cons
  - They are not **mutually exclusive**
  - They **have not been** tested on complex CPU architectures
  - Accuracy as an evaluation metric (what is done when imperfect?)
  - **Knowledge on the code** is not (often) exploited

- **RQ1:** What type of information can be **retrieved** on a SoC via Side-Channel Analysis ?

- **RQ2:** How to efficiently **combine** gathered information in order to make **inferences at different scales** ?

Overview of contributions



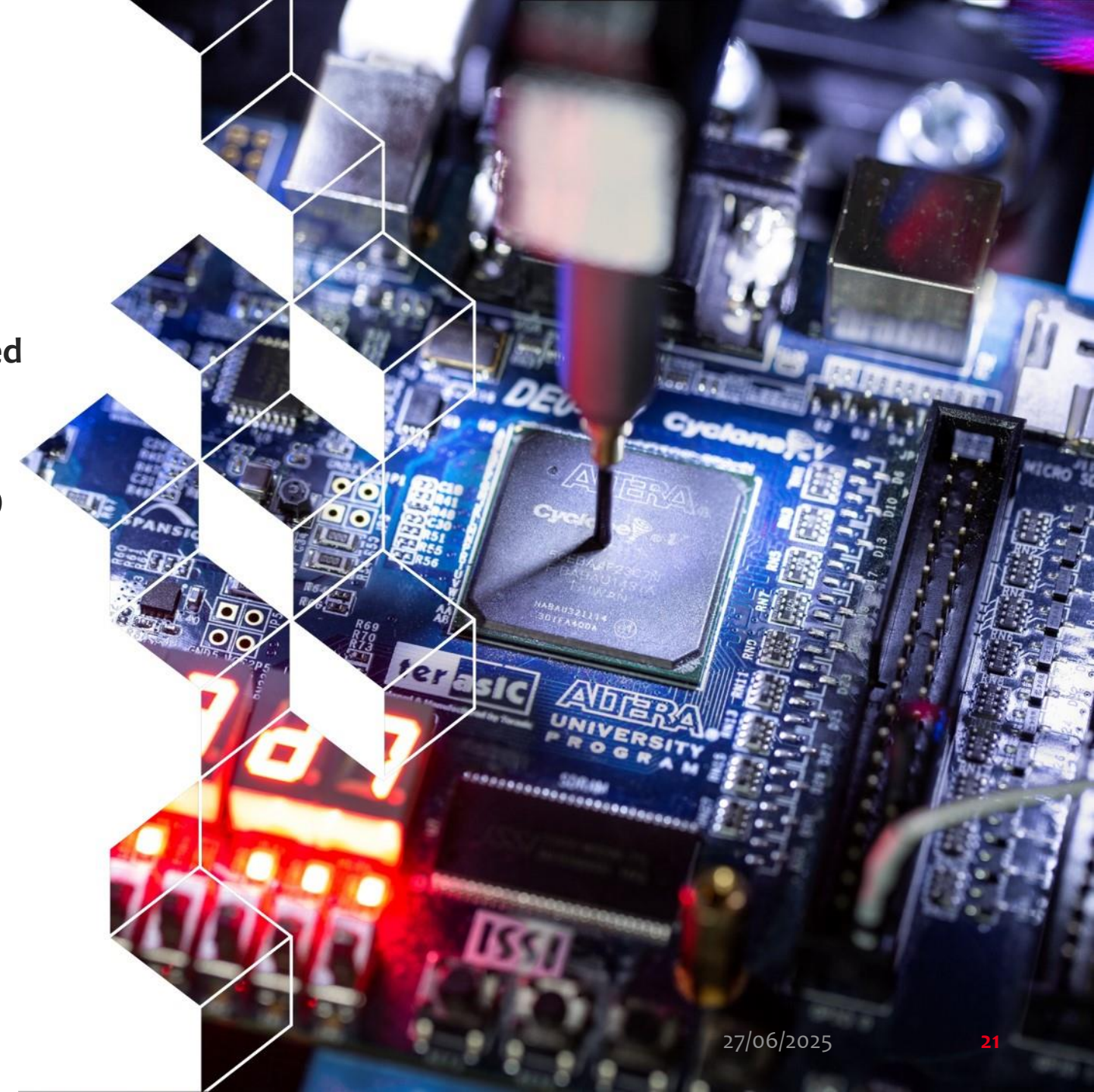
Outline

1. Study of SCBD on SoC

   RQ1: What type of information can be **retrieved** on a SoC via Side-Channel Analysis ?

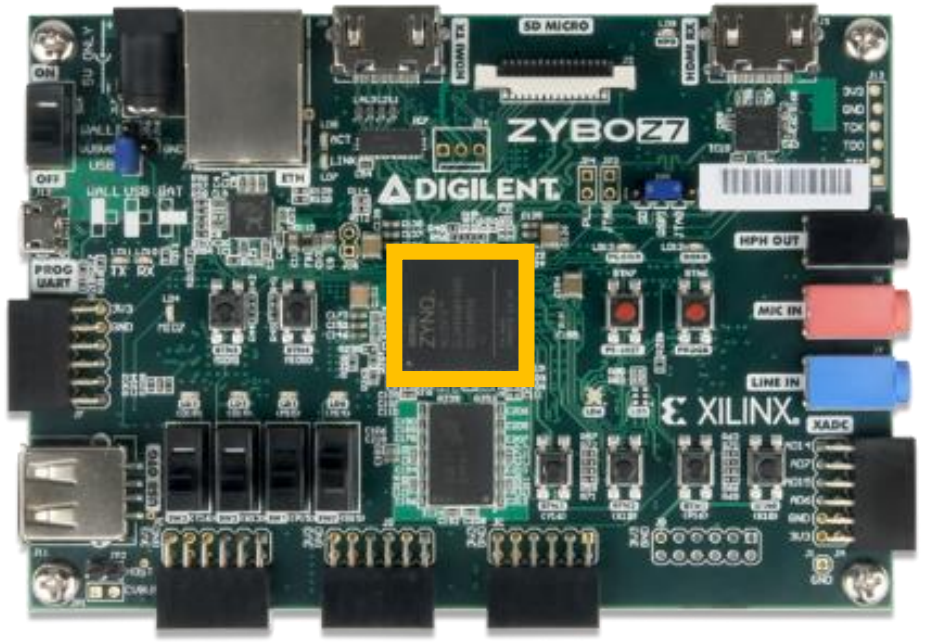
2. Probabilistic models for SCBD

3. Conclusion & future work



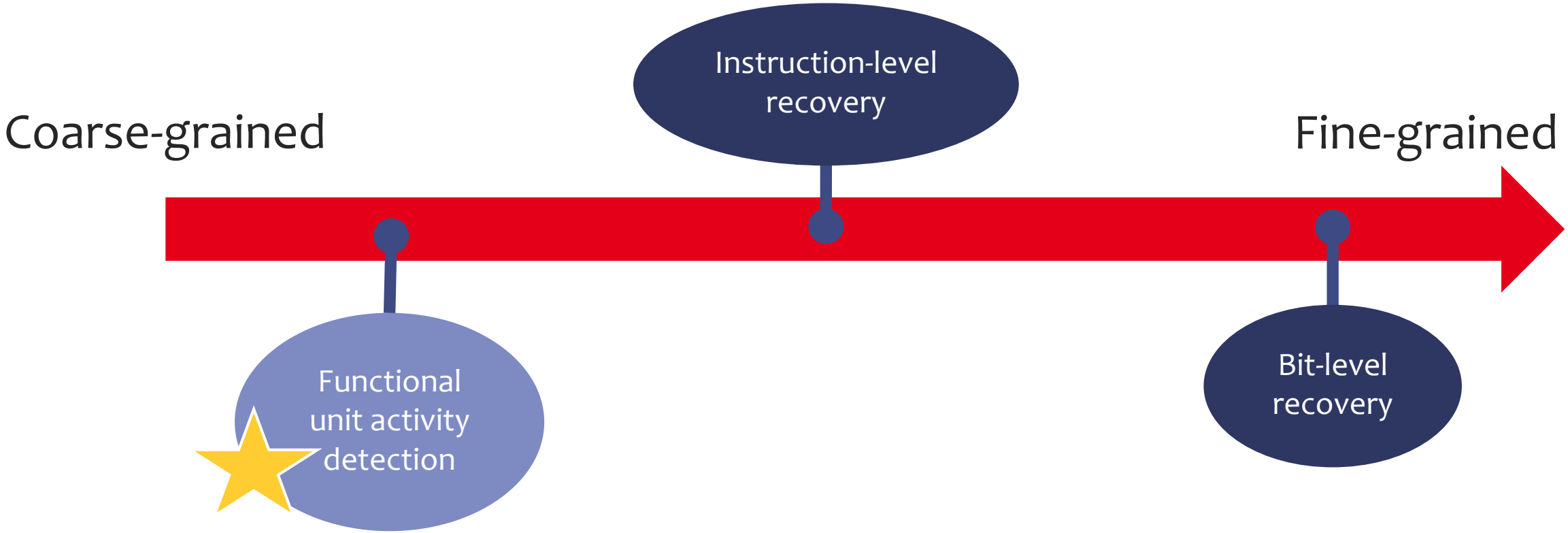
Device under test

- Digilent Zybo Z7 board
- CPU Cortex-A9:
  - Dual-core
  - 667 MHz
  - Superscalar
  - Out-of-Order
  - Dynamic branch prediction
  - Cache memory (2 levels)
- ISA: ARM (32-bit) and Thumb (32/16 bits)



Mitigating noise & jitter sources

Noise / Jitter sources	Mitigation strategy
Pipelining	Surround code with NOPs
Multi-issuing, Out-of-Order execution	Data dependencies
Branch-prediction, memory hierarchy	Code warmup
DVFS	Heavy workload
Address unconsistency	Store characterization code in a buffer

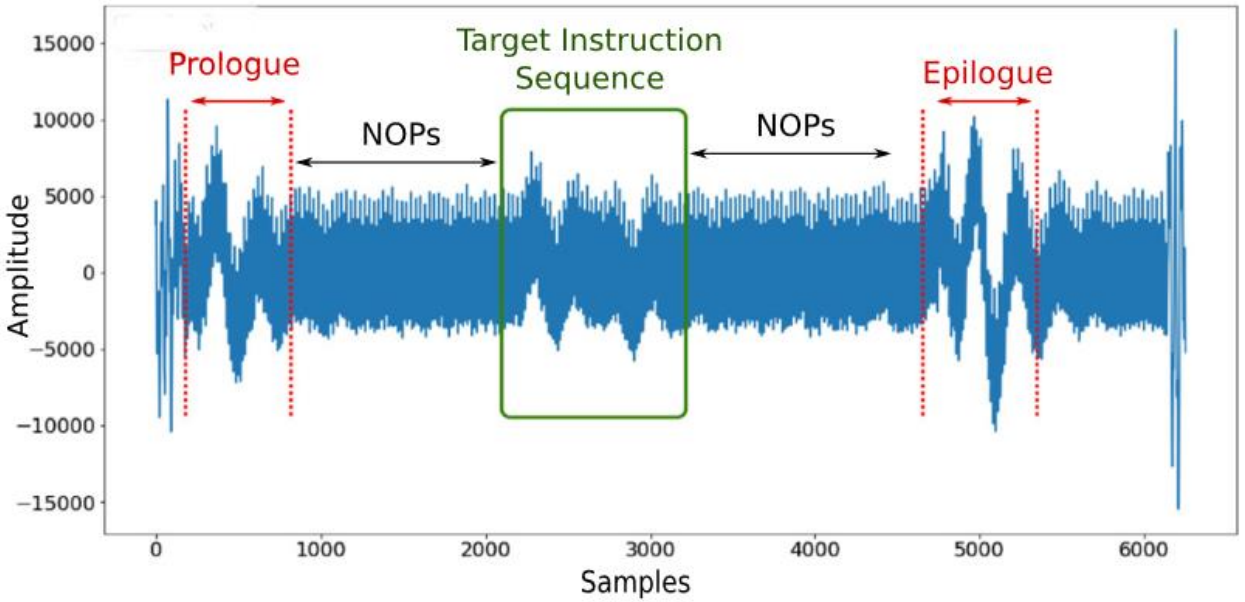


Can we recognize the functional unit used by an instruction sequence?

Characterization programs

```
[PROLOGUE]

NOP
...
NOP

ADD r0, r1, 1
ADD r2, r0, 1
ADD r3, r2, 1
ADD r1, r3, 1

NOP
...
NOP

[EPILOGUE]
```

Target code

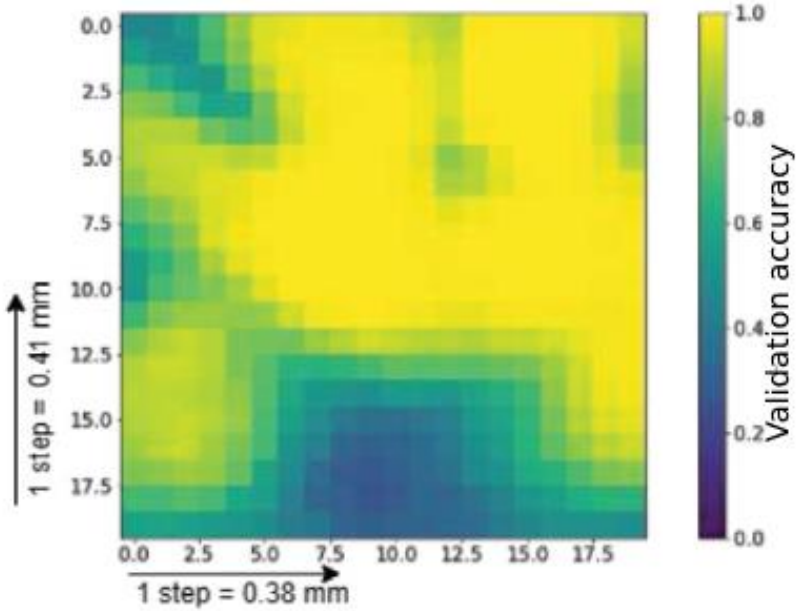


- 4 Functional units targeted:
  - ALU
  - ALU/MUL
  - Barrel shifters
  - Load/Store unit

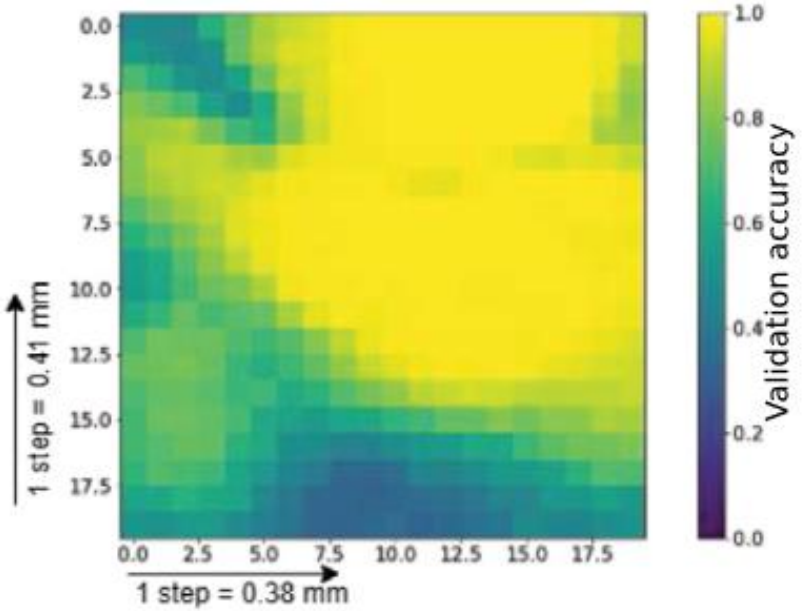
FU recognition: instruction sequence

- Use LDA classifier
- Investigate the spatial spreading

Core 1



Core 2



FU recognition: single instruction

[PROLOGUE]

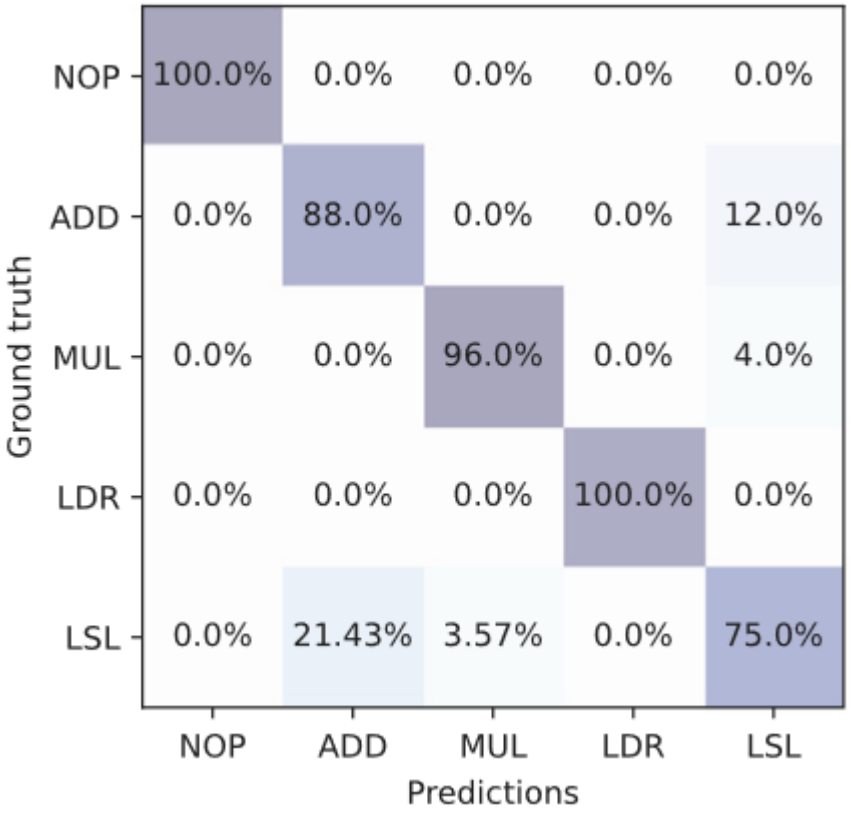
NOP
…
NOP

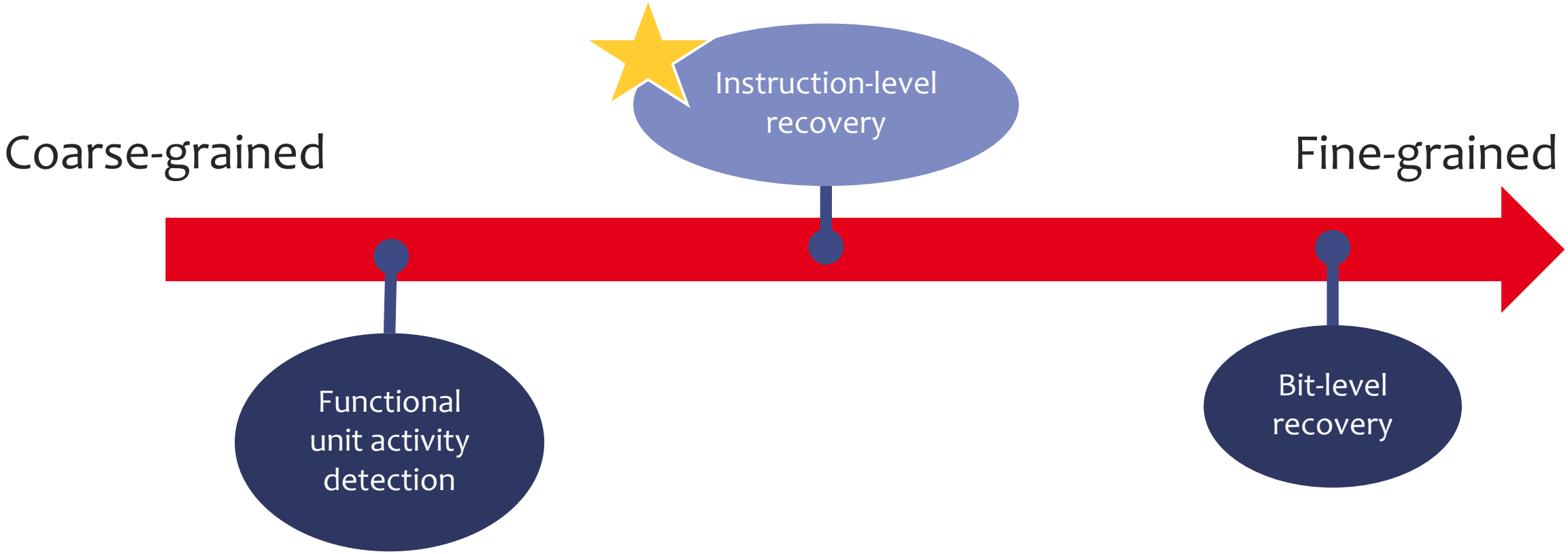
Target instruction

NOP
…
NOP

[EPILOGUE]

$$\mathcal{I} = \left\{ \begin{array}{l} \texttt{NOP} \\ \texttt{ADD r3, r2, r1} \\ \texttt{MUL r3, r2, r1} \\ \texttt{LDR r0, [r0, \#0]} \\ \texttt{LSL r3, r2, r1} \end{array} \right\}$$



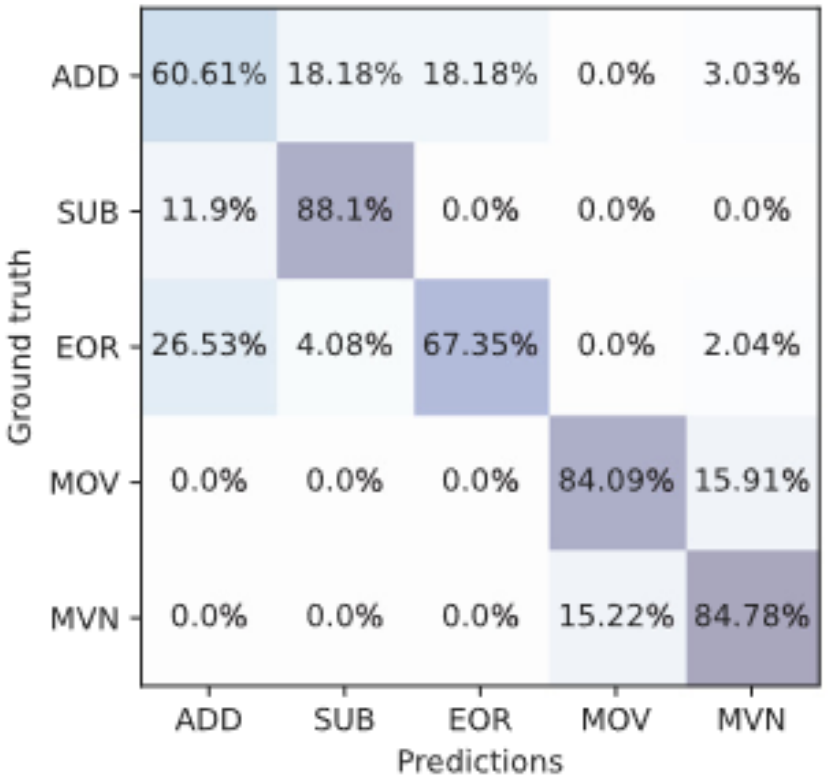


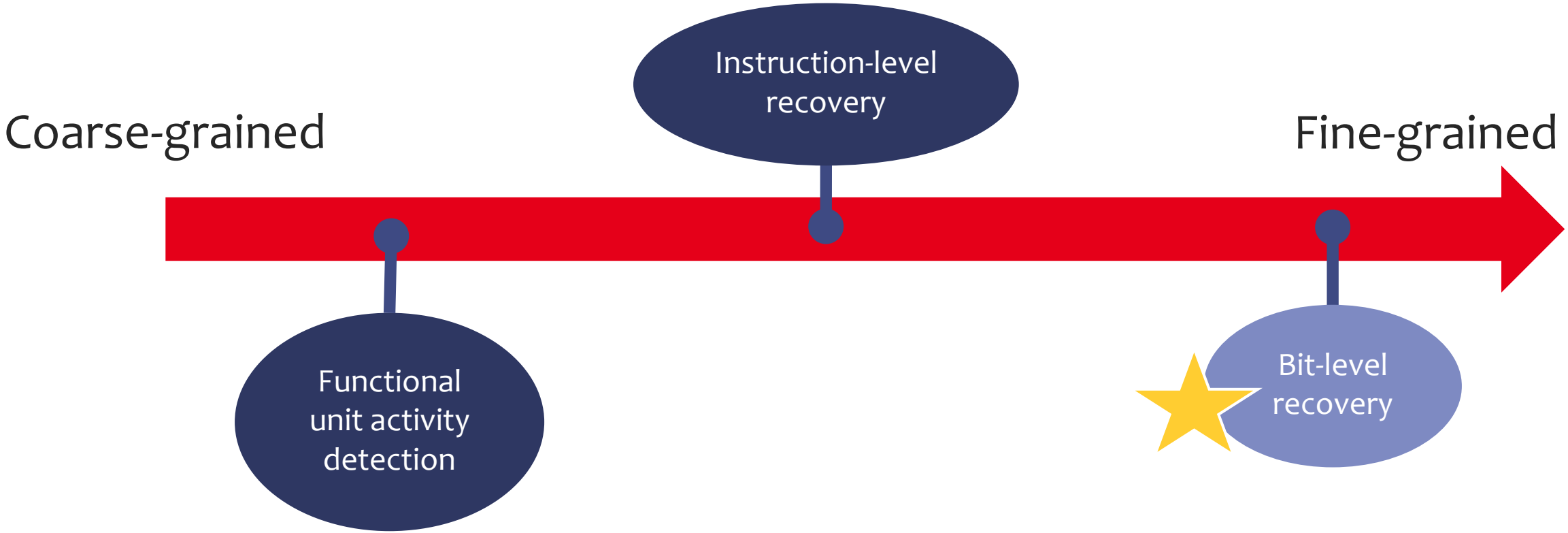
Can we distinguish two instructions going through the same functional unit?

Opcode recognition

- Take **ALU** related opcodes
- Target the **fetch** and/or **decoding** stages
- Randomize operands

$$\mathcal{I} = \left\{ \begin{array}{l} \texttt{ADD r2, r1, \#imm} \\ \texttt{SUB r2, r1, \#imm} \\ \texttt{EOR r2, r1, \#imm} \\ \texttt{MOV r1, \#imm} \\ \texttt{MVN r1, \#imm} \end{array} \right\}$$





Can we recognize the bits that encode an instruction?

Bit level recovery

- Target **fetch** or **prefetch** stages
- Target 8-bit immediates
- Variability on opcodes

$$\mathcal{I} = \left\{ \begin{array}{l} \texttt{ADD r2, r1, \#imm} \\ \texttt{SUB r2, r1, \#imm} \\ \texttt{EOR r2, r1, \#imm} \\ \texttt{MOV r1, \#imm} \\ \texttt{MVN r1, \#imm} \end{array} \right\}$$

Bit index	7	6	5	4	3	2	1	0
Accuracy	92.1%	64.5%	65.9%	50.5%	57.9%	53.3%	57.5%	50.0%

- Possible improvements:
  - Billions of traces
  - Depackaging
  - Multi-position

Coarse-grained

Fine-grained

Routine detection

Instruction-level recovery

Full algorithm detection

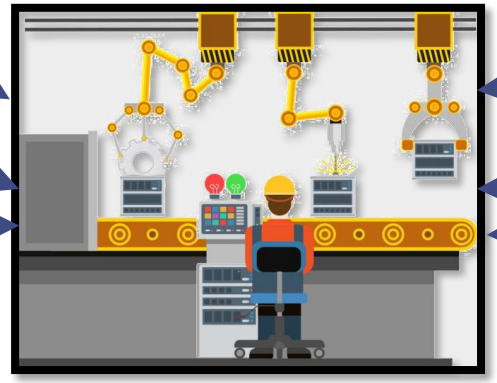
Functional unit activity detection

Bit-level recovery

Bit index	7	6	5	4	3	2	1	0
Accuracy	92.1%	64.5%	65.9%	50.5%	57.9%	53.3%	57.5%	50.0%

JPEG encoder?

Crypto code?



Instruction set

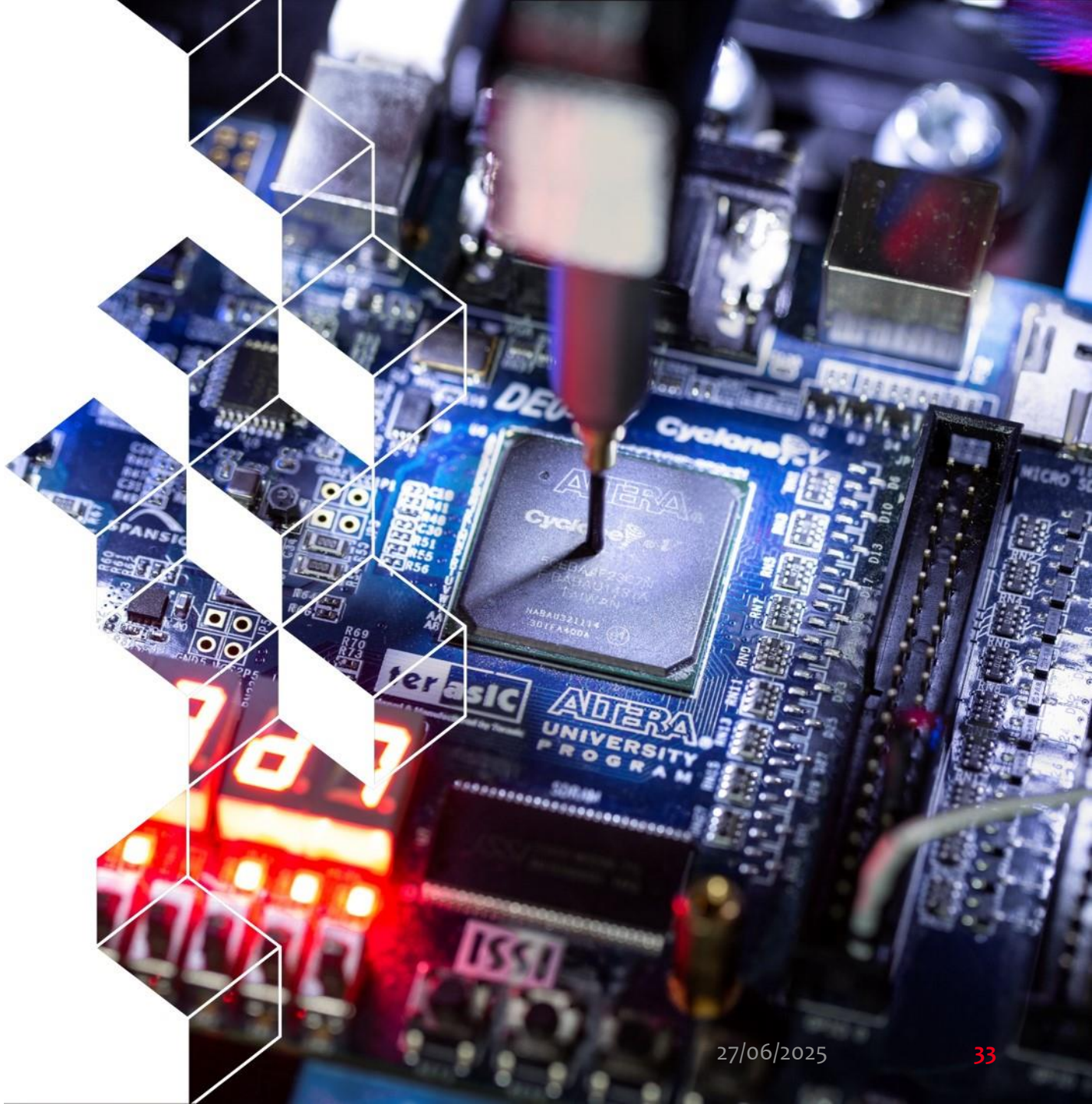
Outline

1. Study of SCBD on SoC

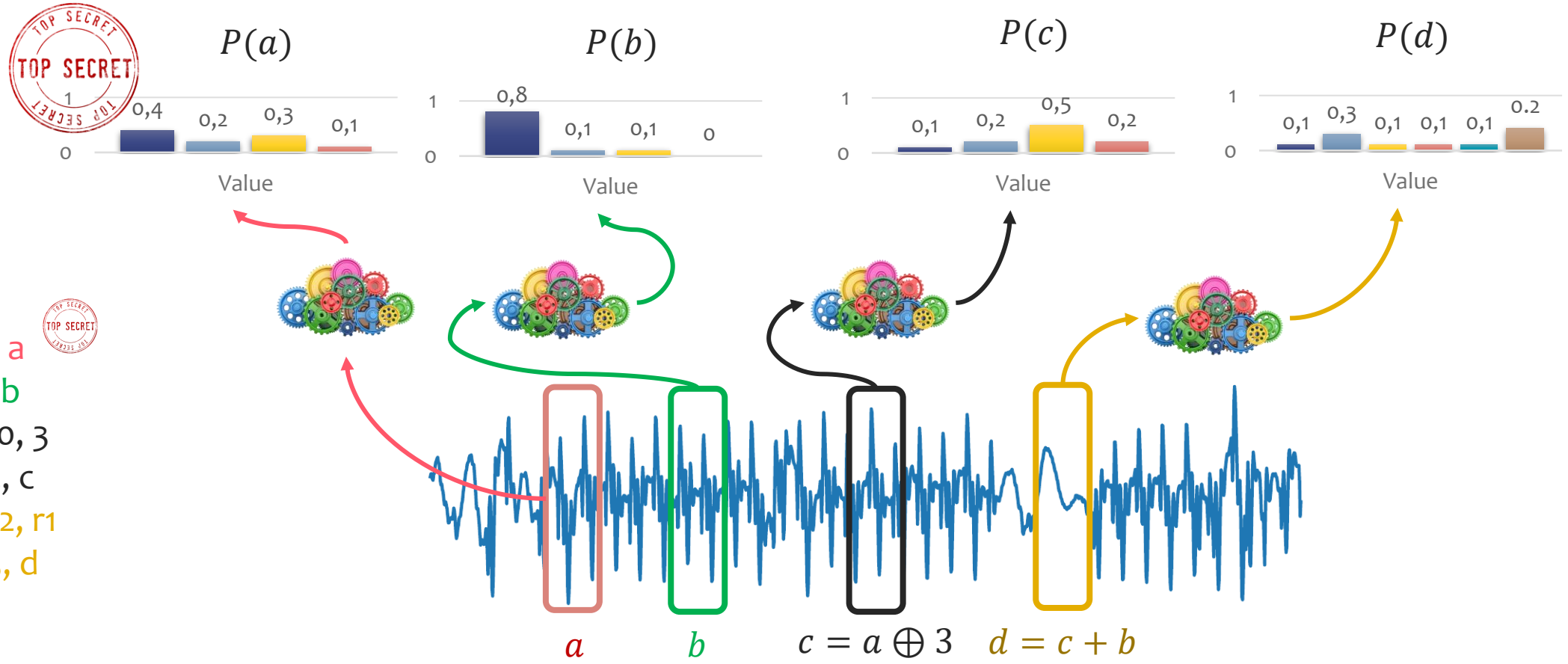
2. Probabilistic models for SCBD

   RQ2: How to efficiently combine gathered information in order to make inferences at different scales?

3. Conclusion & future work



SASCA: General idea



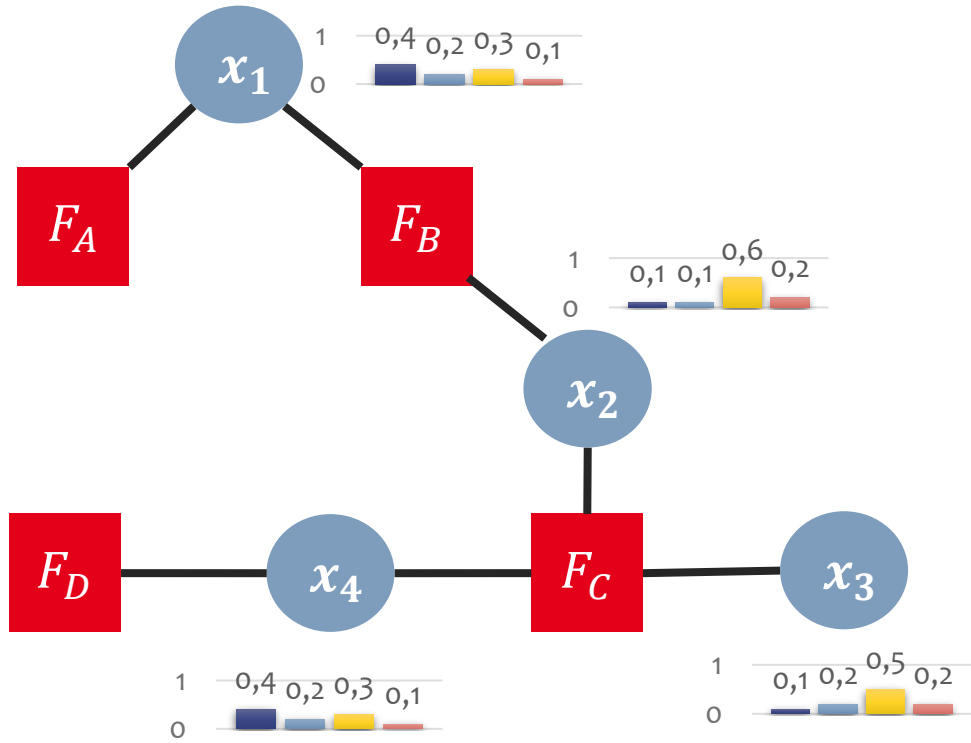
Question: How to efficiently compute $P(a \mid b, c, d)$?

Factor Graphs & Belief Propagation (BP)

- Break **a high dimensional space** problem into **smaller dimensions**
- Tailored to manipulate **probability distributions**

- **Configuration space:** $S = A_1 \times A_2 \times A_3 \times \ ... \times A_n$
- **Variables:** $(x_1, x_2, x_3, \dots, x_n) \in S$
- **Problem:** $g(x_1, x_2, x_3, \dots, x_n) \in R$
- **Goal:** We want to compute marginals for each x_i
- Key idea: break g into $g = \prod_{j \in J} F_j(X_j)$
- A factor graph:
  - Is a bipartite graph that expresses factorization
  - Has a variable nodes for each x_i
  - Has factor nodes for each F_j

$$g(x_1, x_2, x_3, x_4) = F_A(x_1)F_B(x_1, x_2)F_C(x_2, x_3, x_4)F_D(x_4)$$

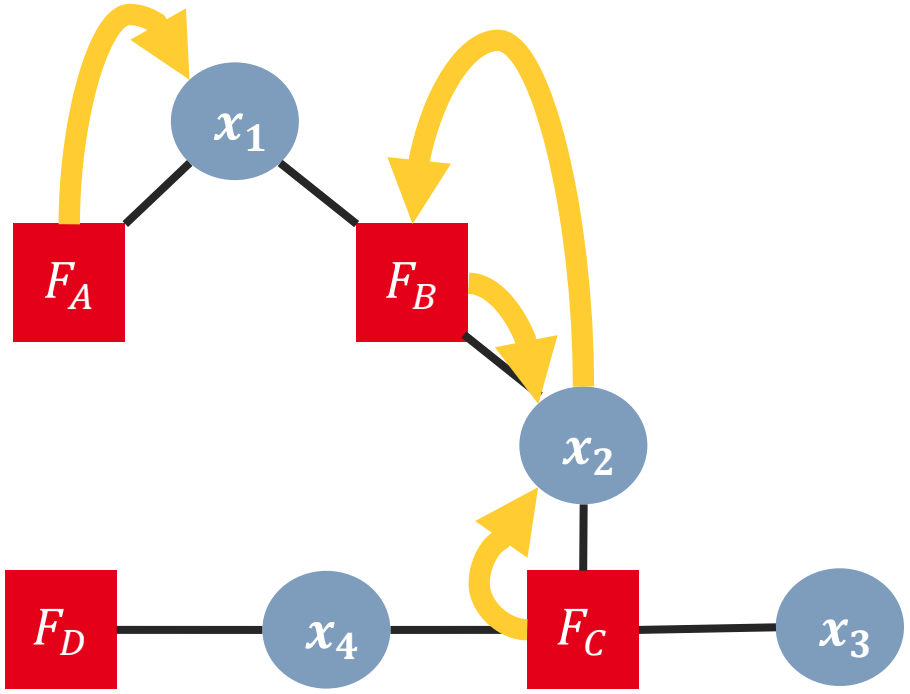


- Several crypto algorithms have been attacked with SASCA

Message passing and Sum-Product

$$P(x) = \frac{1}{Z} \prod_{F \in n(x)} \mu_{F \to x}(x)$$

Marginal computation: computes the product of incoming messages



$$\mu_{x \to F}(x) = \prod_{F' \in n(x) \setminus \{F\}} \mu_{F' \to x}(x)$$

Variable to factor: send the product of messages from other neighboring factors

$$\mu_{F \to x}(x) = \sum_{\sim \{x\}} \left(f(X) \prod_{x' \in n(F) \setminus \{x\}} \mu_{x' \to F}(x') \right)$$

Factor to variable: compute the sum of the product of a function f with the messages received from neighbors

Proved exact on tree-like graphs!

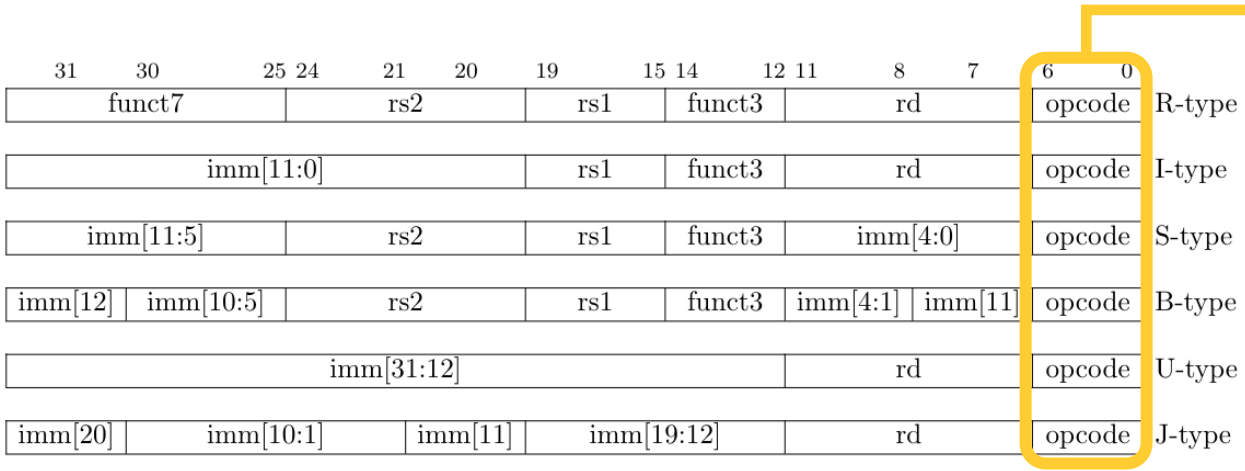
(But iterative message passing can be done otherwise)

SASCA framework

- Several challenges arise when attempting BP
- Implemented a generic framework:
  - Multiple supports for variables
  - Optimized reprensentation for 10+ factors
  - Numerically stable
  - Scalable
  - Flexible
- Leakage simulation:
  - **Theoretical** (Hamming weight leakage model…)
  - **Real case** oriented (Prediction matrices)
- **Monitoring / visualization** tools (entropy, rank)
- **Convergence handling:** Damping, early stopping…

- Faster than SCAlib's implementation
- Used in several publications *[Mai+24b], [Goy+24], [Bai+24]*
- We can use it to combine information on RISC-V instructions (**SASCBD**)

SASCBD: RISC-V Specifications



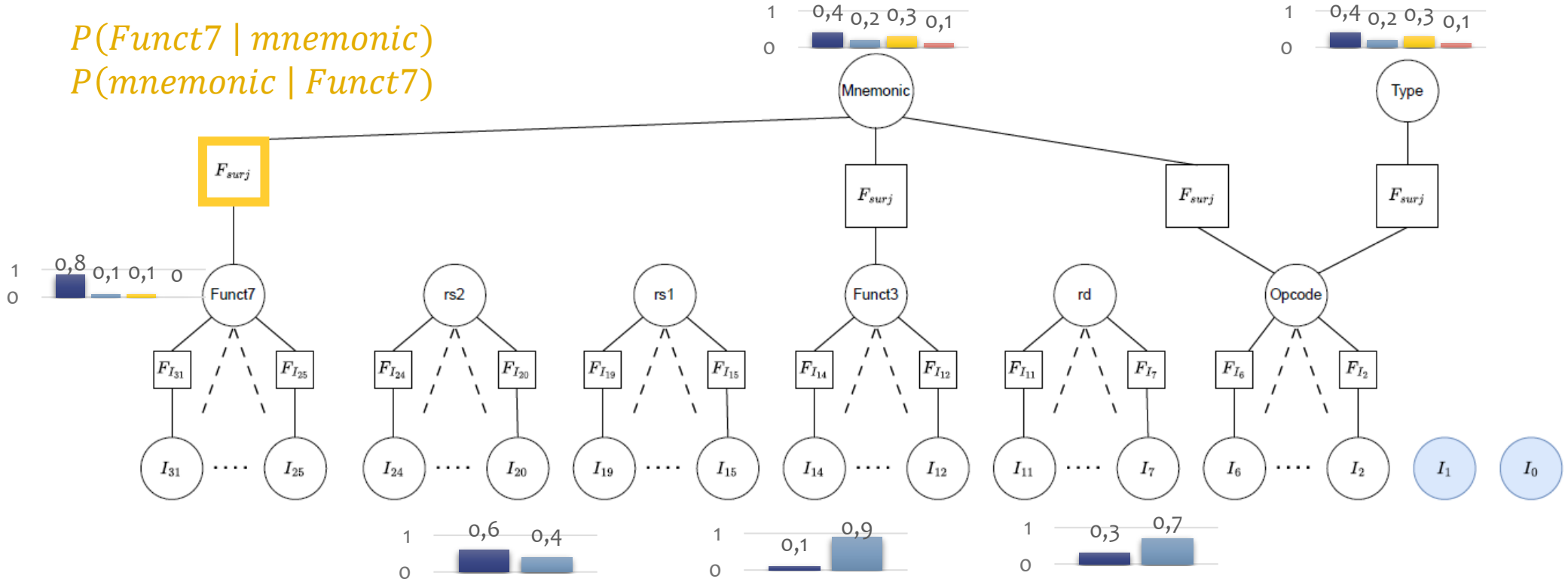
inst[4:2] inst[6:5]	000	001	010	011	100	101	110	111 (> 32b)
00	LOAD	LOAD-FP		MISC-MEM	OP-IMM	AUIPC	OP-IMM-32	
01	STORE	STORE-FP		AMO	OP	LUI	OP-32	
10	MADD	MSUB	NMSUB	NMADD	OP-FP			
11	BRANCH	JALR		JAL	SYSTEM			

- 32-bit instructions
- Multiple fields varying with encoding type
- Mnemonic (*ADD, SUB, BGE …*) = *opcode + Funct3 + Funct7*
- We want to represent this encoding with a factor graph:
  - Represent conditional probabilities on fields
  - Discard impossible values

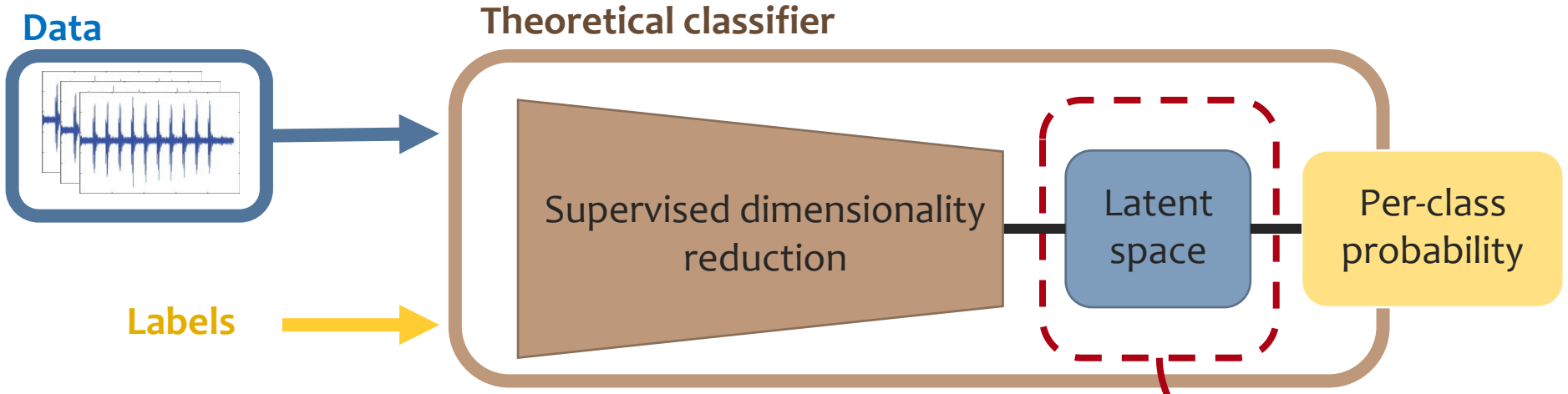
RISC-V instruction encodings, extracted from [RIS]

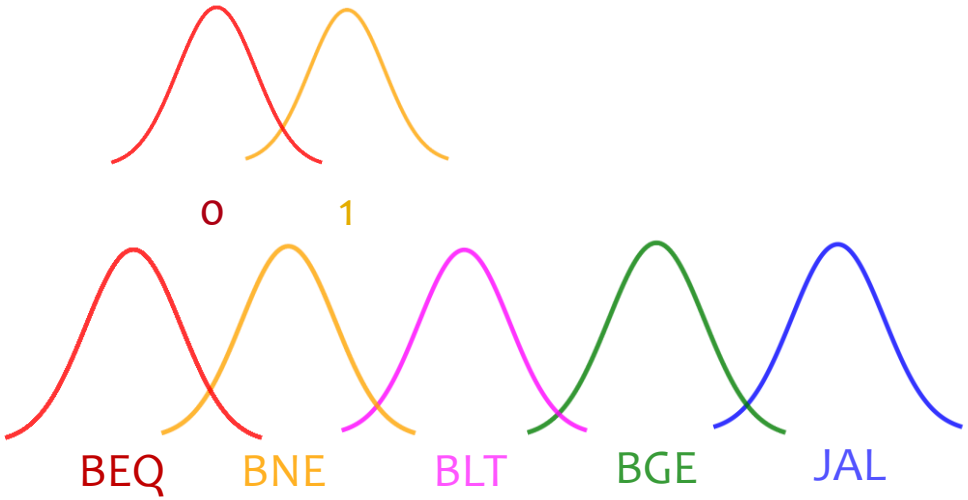
SASCBD : Building RISC-V instruction graph

$P(Funct7 \mid mnemonic)$
$P(mnemonic \mid Funct7)$



SASCBD: Simulation Parameters



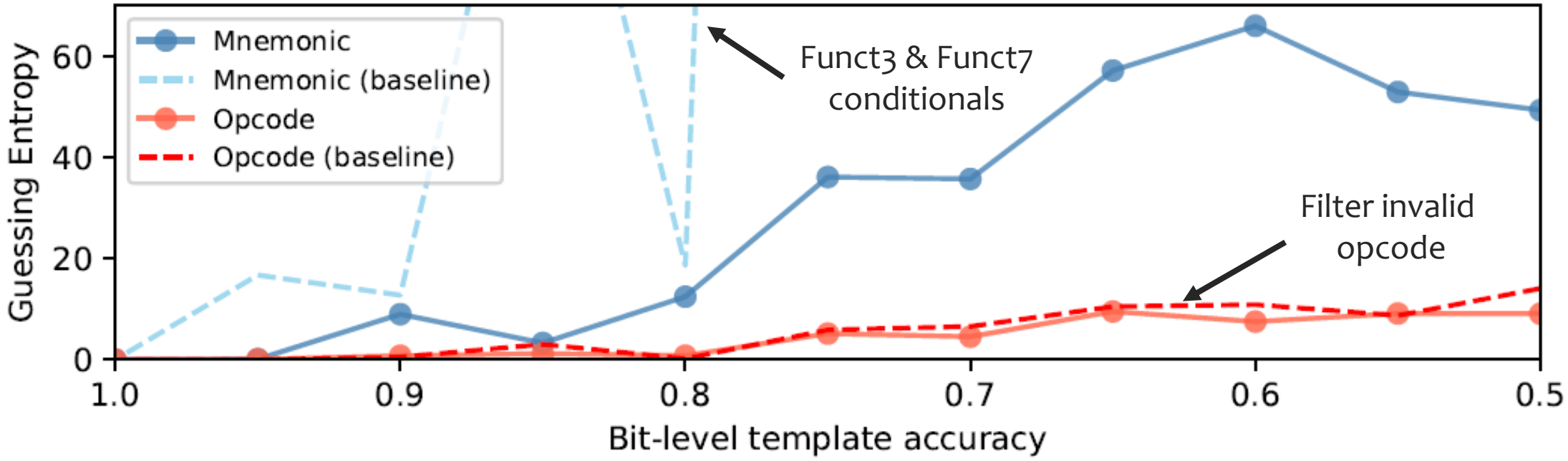


Bit or mnemonic — Noise parameter

$$l = x + \beta,\ \beta \sim \mathcal{N}(0, \sigma^2)$$

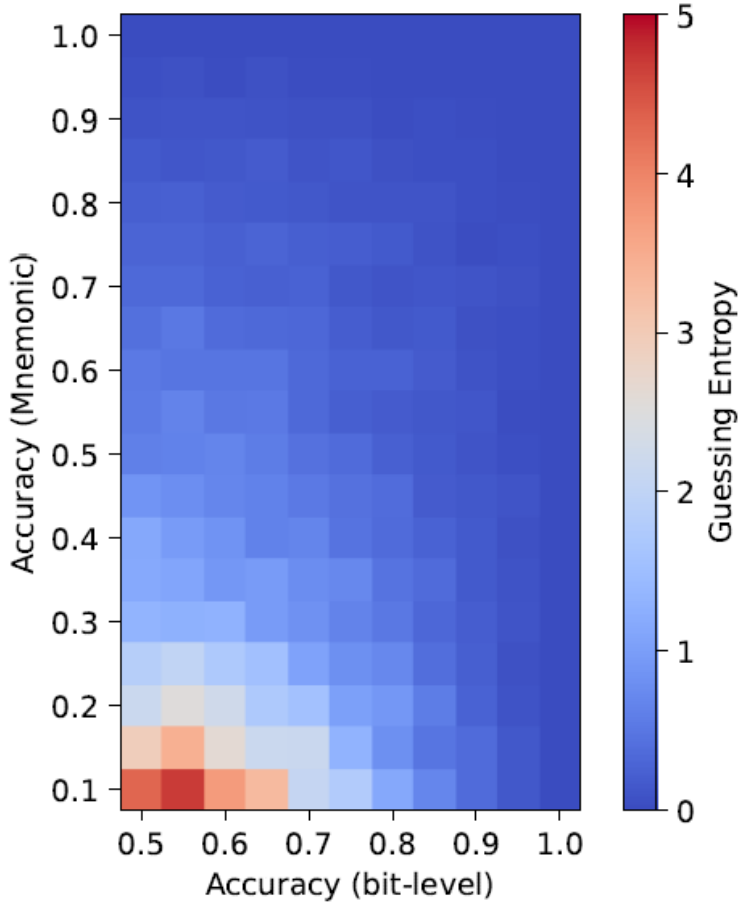
SASCBD: ISA aware bit-level approach

- Problem: *Bit-level SCBD can output invalid instructions*
- Simulate leakages on bits
- Evaluate **guessing entropy** on **opcode** and **mnemonic**



SASCBD: Aggregating multiple templates

- Problem: *A hierarchical paradigm requires to **make a choice** at each level*
- Problem: *Multiple **non-exclusive** paradigms can be used*
- Simulate leakages on bits and on mnemonics
- Evaluate guessing entropy of **mnemonic** recovery

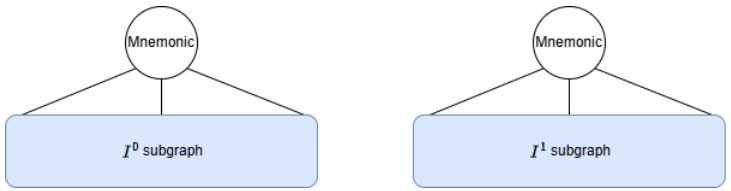


SASCBD: Exploiting code properties

- Problem: *Knowledge on the code is poorly exploited by existing SCBD techniques*
- Consider a sequence of instructions
- Leakage on the bits
- Consider 3 variants: Base, Count and Join

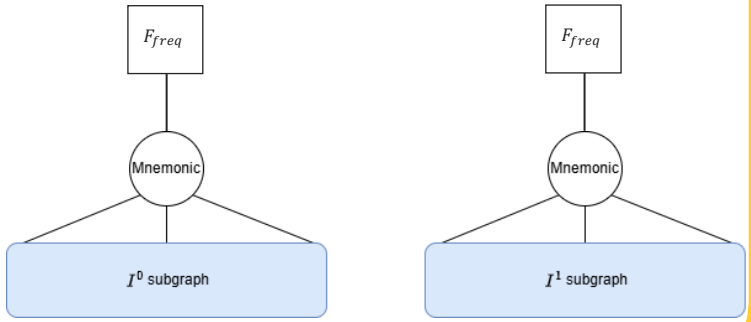
Base

- ISA knowledge only



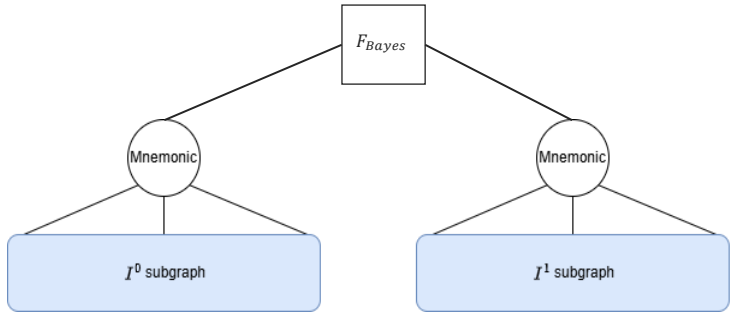
Count

- ISA knowledge
- Mnemonic frequency



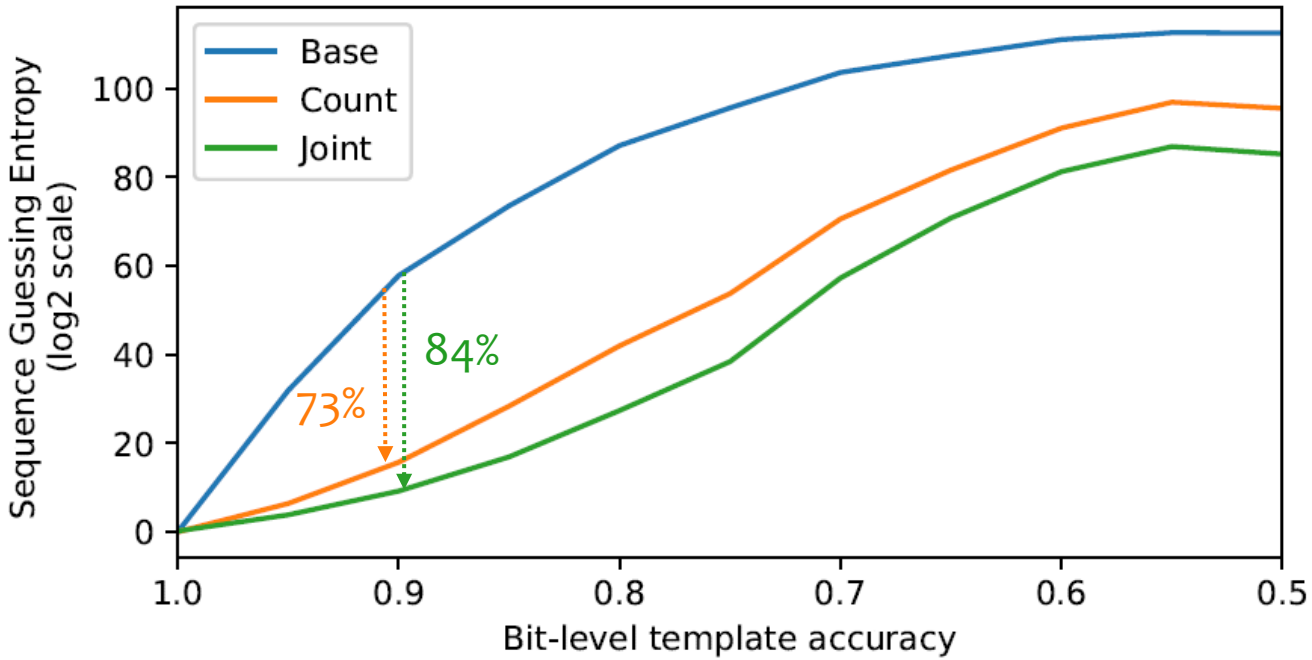
Joint

- ISA knowledge
- Mnemonic transition frequency



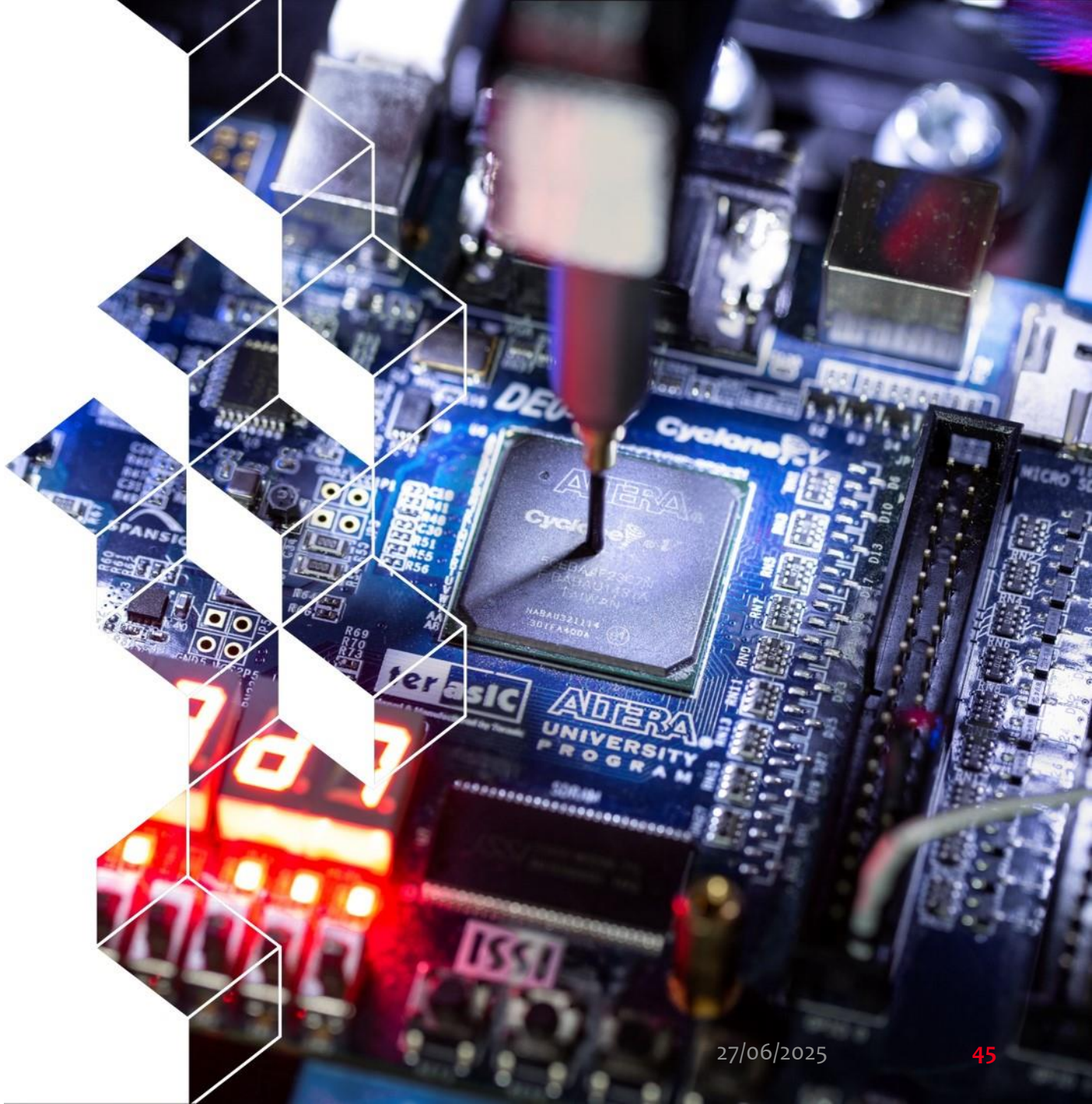
SASCBD: Exploiting code properties

- Measure **guessing entropy** on **mnemonic sequence**



Outline

1. Study of SCBD on SoC
2. Probabilistic models for SCBD
3. Conclusions & future work



Conclusions

- Investigated SCBD on SoC:
  - Information retrieval
  - Information aggregation
- SCBD can be done at coarse-grained scale
- Harder at fine-grained scale
- SASCBD: rely on probabilistic modeling
- Allows to:
  - Combine multiple paradigms
  - Exploit ISA structure
  - Exploit code properties

- But also:
  - New hybrid cache/EM attack paradigms
  - SASCA on hash functions (bootloaders, PQC…)
  - SASCA on HQC

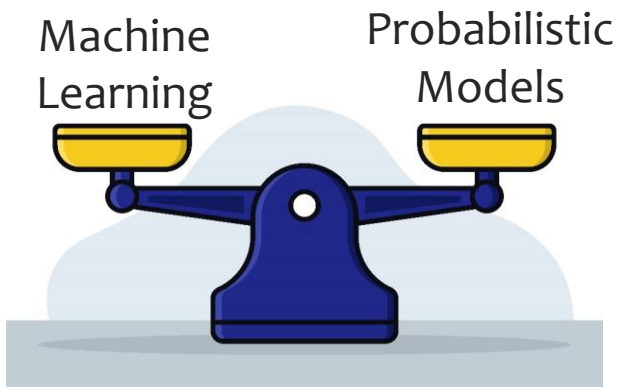
Further work

Information gathering

- Characterization without noise mitigations
- More powerful learning models
- Instruction set coverage:
  - Control-flow instructions
  - Floating point, special instructions…
- Apply this technique for:
  - Bootloader reverse-engineering
  - Fault injection helper, malware detection…

Information aggregation

- Apply SASCBD with real SoC measurements
- Better code statistics (static and dynamic analysis)
- Handle variable sized instruction sets
- Pushing the limits of SASCA:
  - Extend the scope of exact inference
  - Work on better approximates for loopy-BP



Publications

Main Works

- Julien Maillard, Thomas Hiscock, Maxime Lecomte, and Christophe Clavier. “Towards Fine-grained Side-Channel Instruction Disassembly on a System-on-Chip”. In: *2022 25th Euromicro Conference on Digital System Design (DSD)*. 2022, pp. 472–479
- Julien Maillard, Thomas Hiscock, Maxime Lecomte, and Christophe Clavier. “Side-channel disassembly on a system-on-chip: A practical feasibility study”. In: *Microprocessors and Microsystems* 101 (2023), p. 104904
- Julien Maillard, Thomas Hiscock, Maxime Lecomte, and Christophe Clavier. “Cache Side-Channel Attacks Through Electromagnetic Emanations of DRAM Accesses”. In: *Proceedings of the 21st International Conference on Security and Cryptography – SECRYPT* .INSTICC. SciTePress, 2024, pp. 262–273
- Julien Maillard, Thomas Hiscock, Maxime Lecomte, and Christophe Clavier. “Simulating SASCA on Keccak: Security Implications for Post-Quantum Cryptographic Schemes”. In: *Proceedings of the 21st International Conference on Security and Cryptography - SECRYPT*. INSTICC. SciTePress, 2024, pp. 518–527

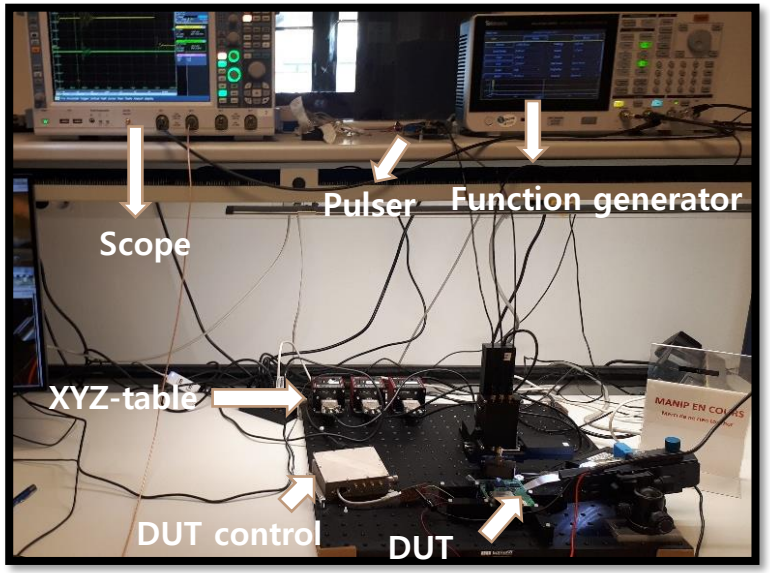
Side Works

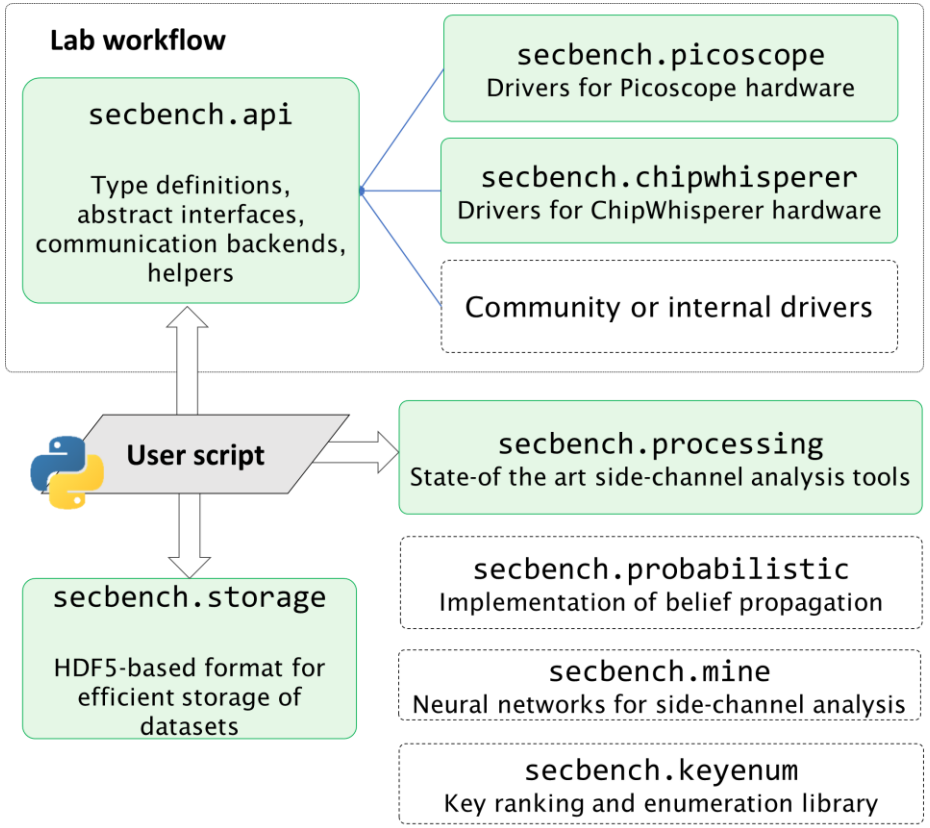
- Julien Maillard, Awaleh Houssein Meraneh, Modou Sarry, Christophe Clavier, Hélène Le Bouder, and Gaël Thomas. “Blind side-channel analysis on the Elephant LFSR Extended version”. In: *International Conference on Smart Business Technologies*. Springer. 2022, pp. 20–42
- Guillaume Goy, Julien Maillard, Philippe Gaborit, and Antoine Loiseau. “Single trace HQC shared key recovery with SASCA”. in: *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2024.2 (2024), pp. 64–87
- Chloé Baisse, Antoine Moran, Guillaume Goy, Julien Maillard, Nicolas Aragon, Philippe Gaborit, Maxime Lecomte, and Antoine Loiseau. “Secret and Shared Keys Recovery on Hamming Quasi-Cyclic with SASCA”. in: *Cryptology ePrint Archive* (2024) (submitted to DCC)
- Julie Godard, Nicolas Aragon, Philippe Gaborit, Antoine Loiseau, and Julien Maillard. “Single Trace Side-Channel Attack on the MPC-in-the-Head Framework”. In : *Cryptology ePrint Archive* (2024) (submitted to PQCrypto)

Secbench

Goal : provide a framework for conducting reproductible hardware security experiments

- Internal tool developed at CEA since 2017
- For researchers, used daily for security characterizations
- Python, Rust and C++ codebase
- Support for common platforms: Linux, Windows, MacOS





If you want to try Secbench

If you want to know more about my work ;)

Thanks!
Questions?

Images sources: Freepik

Contact: julien.maillard@cea.fr