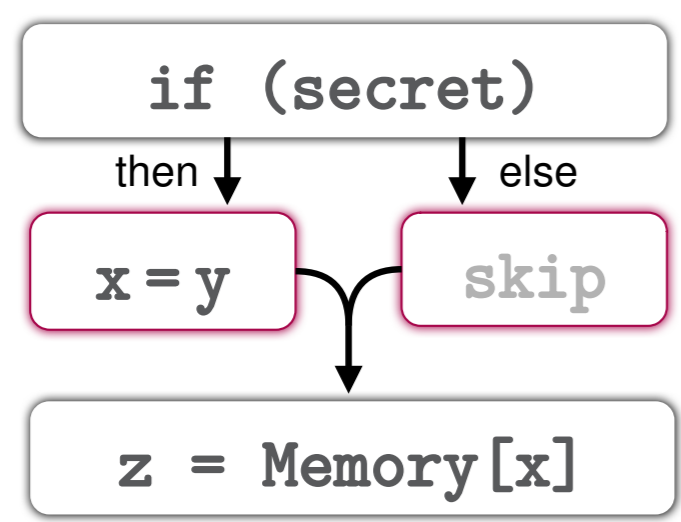


Context

Timing vulnerabilities caused by behavior depending on a secret

```
x = 0, y = 64
if (secret){
  x = y
}
z = Memory[x]
```

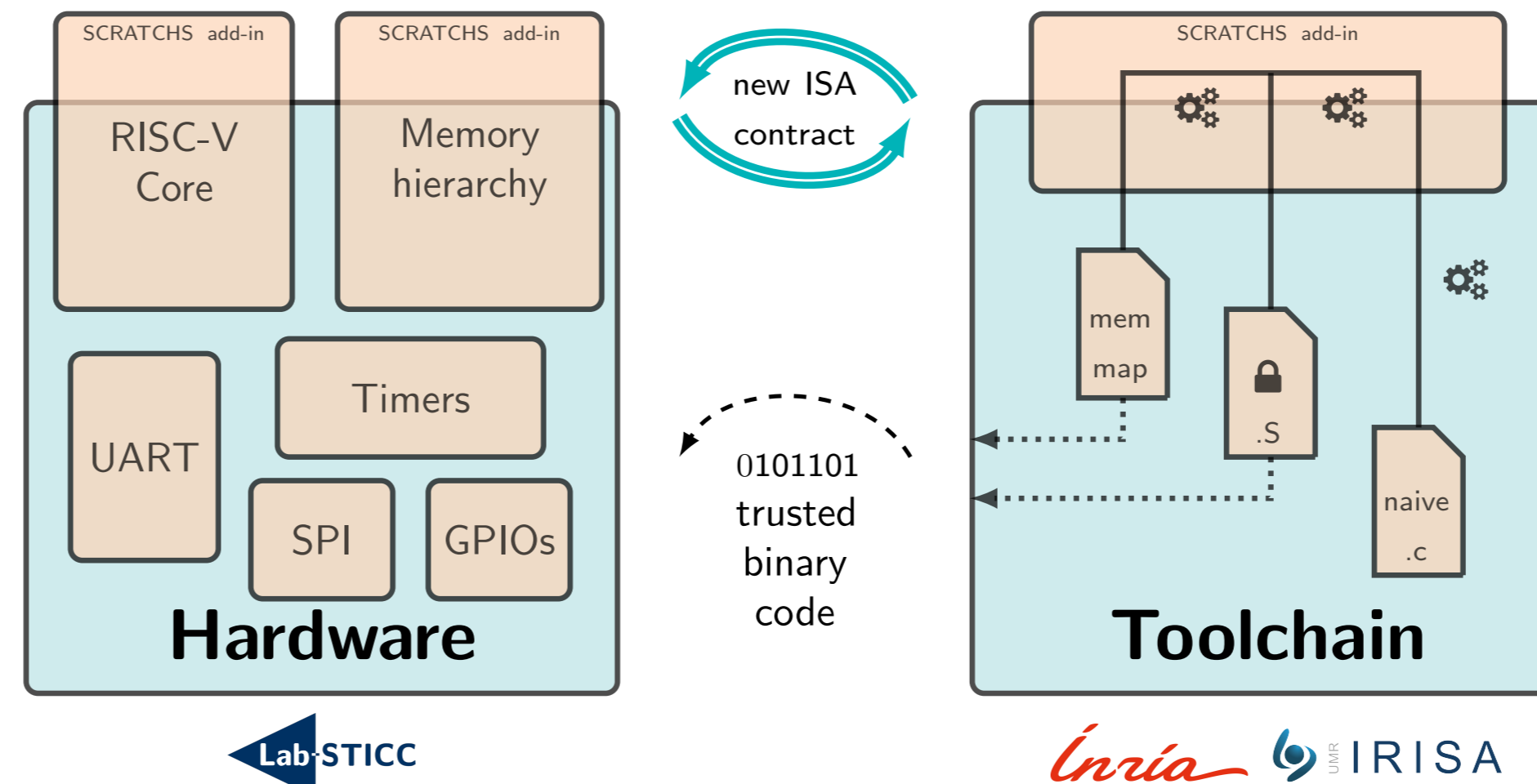


Timing differences:
 • in branching
 • in memory access
 → Secret exposed

Attacker: observes time ⇒ deduces secret

- ▶ Behavior duration depends on resource usage (like memory access).
- ▶ Timing is observable when resource usage is shared between the victim and the attacker.
- ▶ Countermeasures already exist (resources partitioning, Constant-Time programming), but are often costly.

SCRATCHS



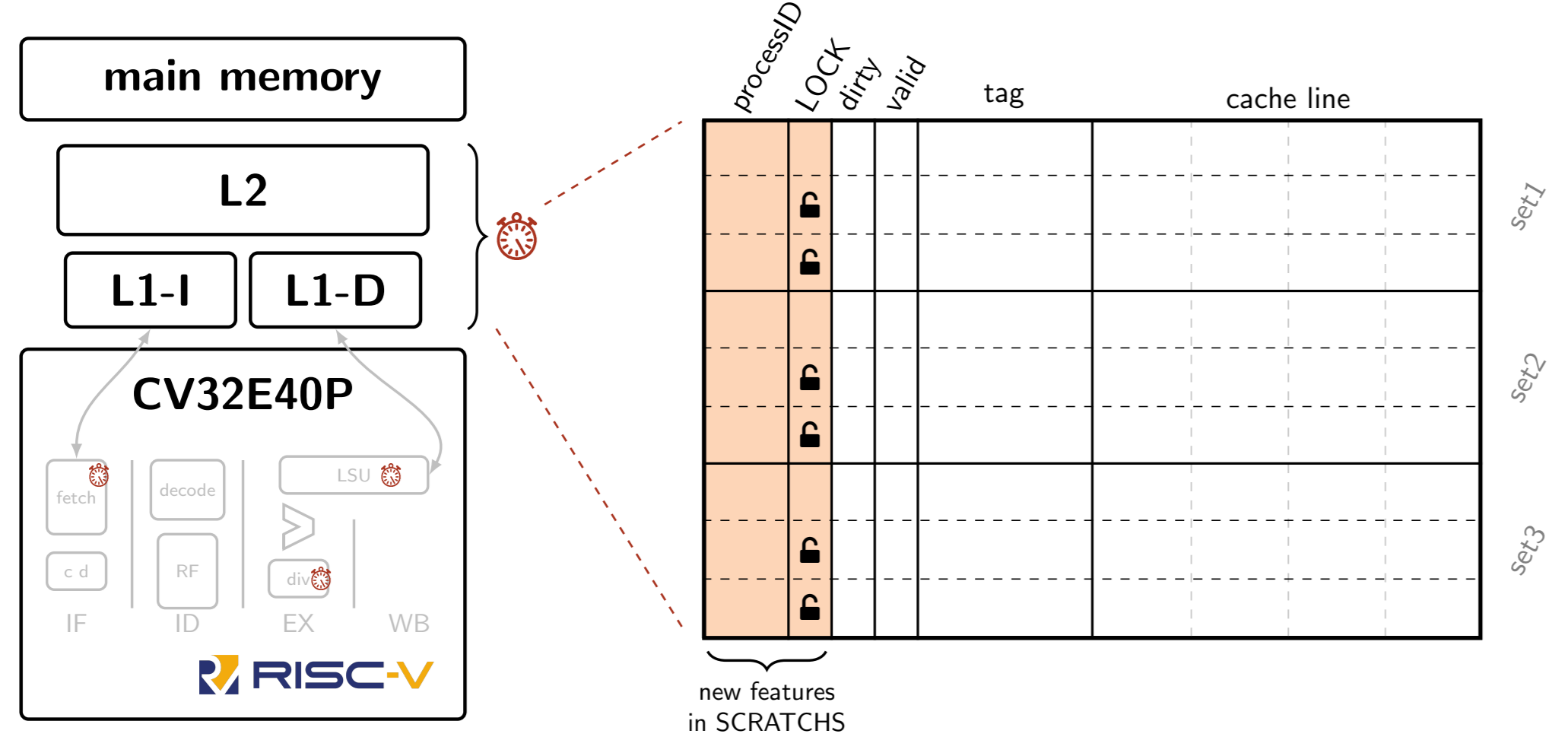
SCRATCHS's goal is to **co-design** a **RISC-V processor** and a **compiler toolchain**:

- ▶ Immune sensitive code to **timing side-channel attacks**.
- ▶ Minimal overhead on the micro-architecture.
- ▶ Considering a small-scale embedded system.

- ▶ **Hardware** implements **security mechanisms**.
- ▶ **Compiler** produces binaries able to use these mechanisms to be **side-channel resistant**.

Cache lines locking mechanism

Memory hierarchy and some functional units temporal behaviors (e.g. ALU, LSU, division or branching) can leak **information**.



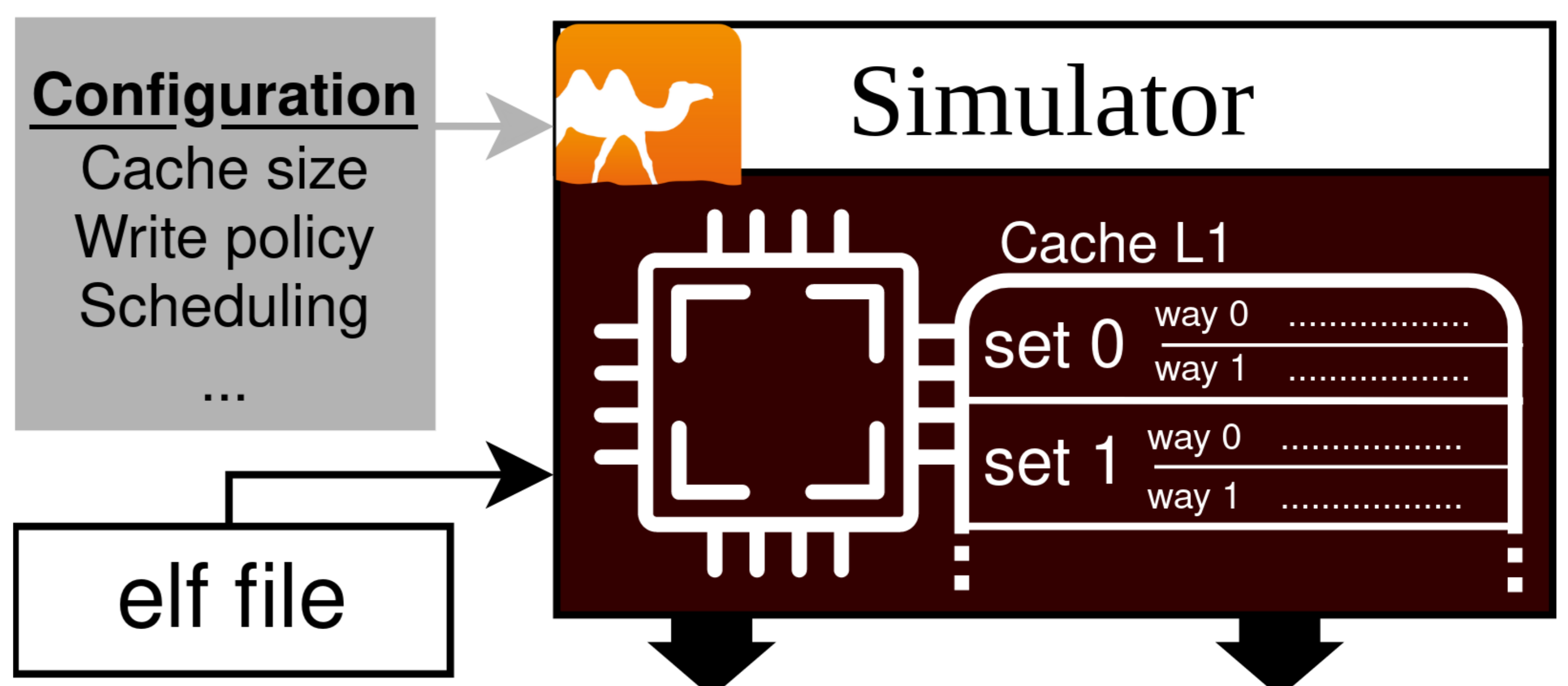
We identify three sources of leakage on the CV32E40P RISC-V processor:

Leak	Solutions
Division and modulo op.	→ Constant-time mode through a CSR
Non-aligned data requests	→ Solved by compiler toolchain
Cache accesses (L1, L2, TLB...)	→ New lock and unlock instructions

lock/unlock mechanism:

- ▶ The cache line is locked in cache until the locking process issues an unlock operation
- ▶ At least one way of the cache is kept available to other processes' data
- ▶ Implement lock on skewed randomized cache to augment security level
- ▶ Low overhead targeting FPGA (<3% on registers and LUTs)

Simulation and leakages



Program	Abstract leakage	Concrete leakage
int a = b + c;	[•]	[•]
int a = array[i];	cache set(&array + i)	cache miss
lock(&array + i);	cache set(&array + i)	cache hit
int a = array[i];	[•]	cache hit

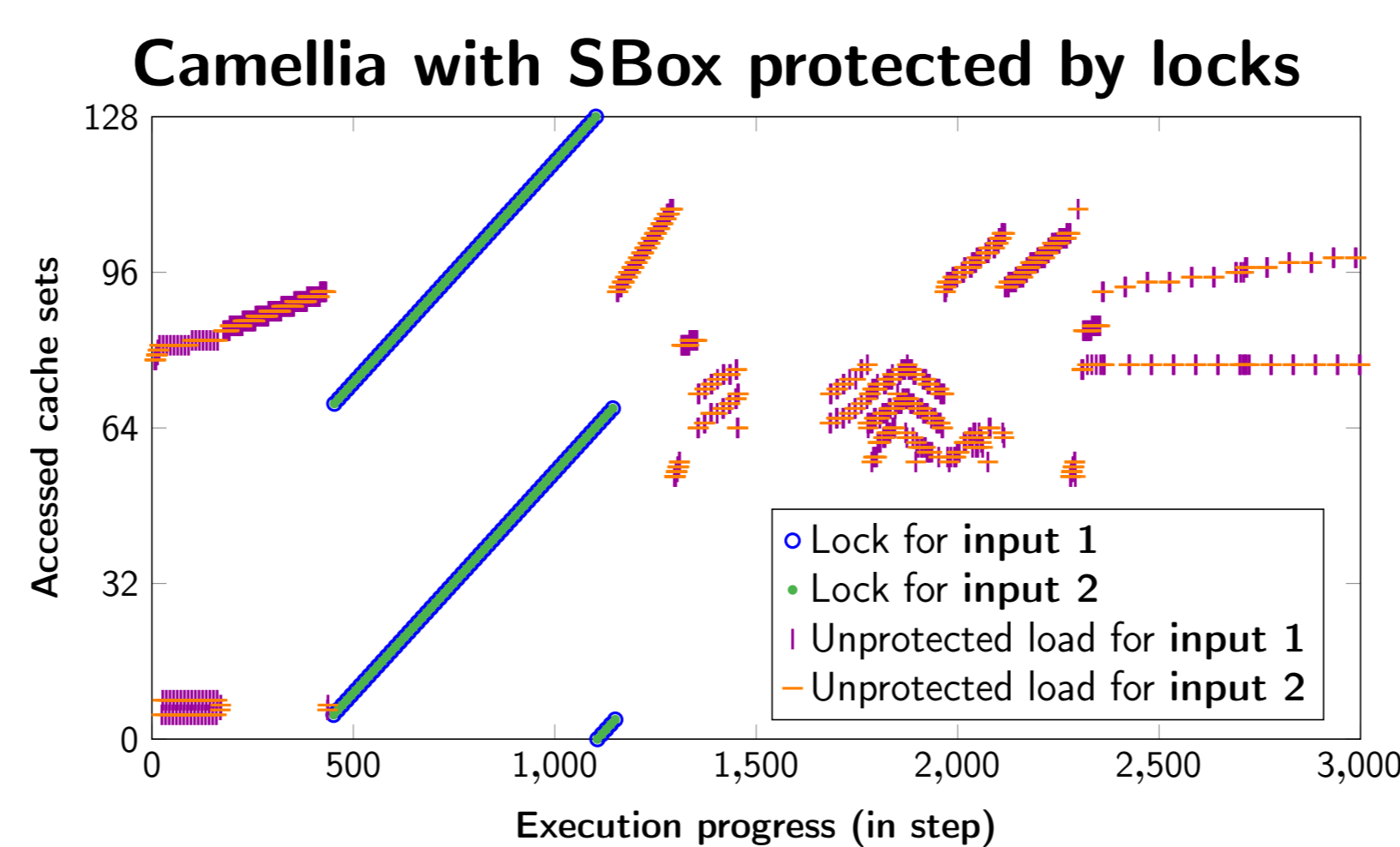
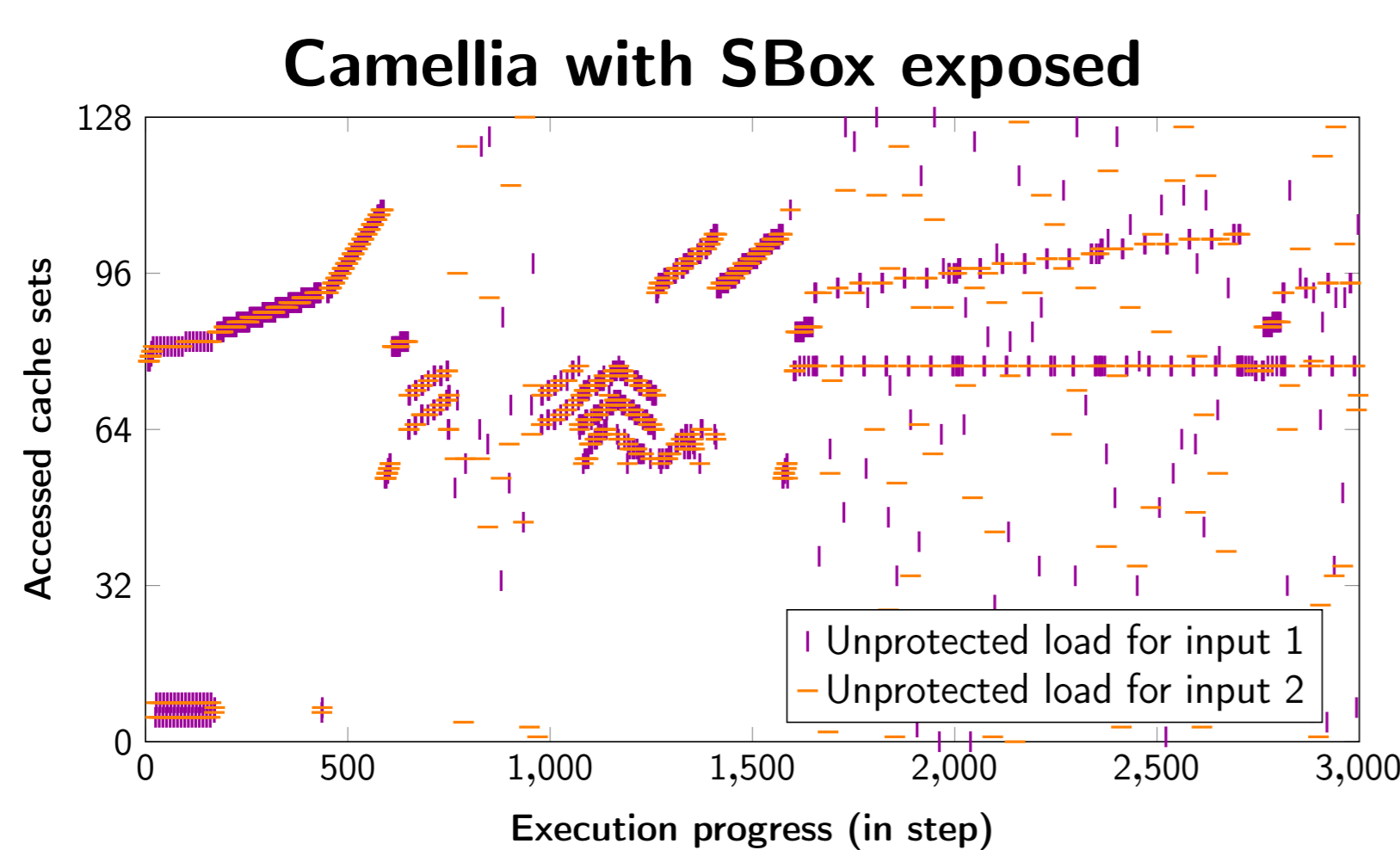
Abstract leakage: What could be seen, depends on current program and inputs
Concrete leakage: What is observed. Depends on current instruction and cache state.

Our claim:
 All information of **Concrete leakage** is deductible from **Abstract leakage** ⇒ Non interference of input in **Abstract leakage** means no timing attack possible ⇒ Security guarantee for a program possible (for a given input space)

Results

Security evaluation

We can protect symmetric encryption algorithms (AES, Camellia, etc.) that use SBox (lookup table)

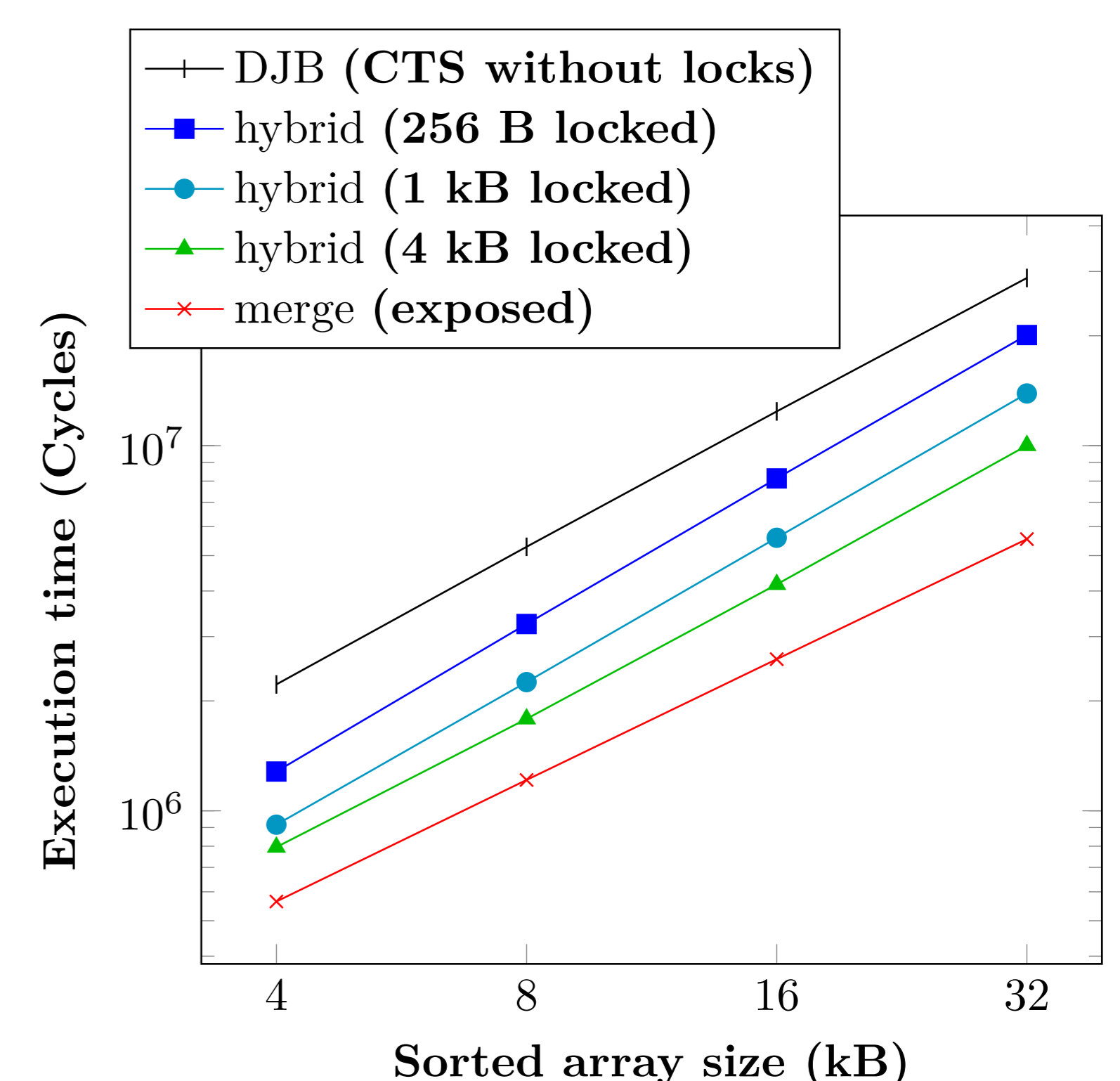


These figures display memory accesses that are visible to a potential attacker.

Variation of visible accesses depending on the input means the input is exposed to timing attacks.

Performance evaluation

Comparison between sorting algorithms



[1] N. Gaudin, J-L. Hatchikian-Houdot et al., "Work in Progress: Thwarting Timing Attacks in Microcontrollers using Fine-grained Hardware Protections", 2023 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW), Delft, Netherlands, 2023, pp. 304-310, doi: 10.1109/EuroSPW59978.2023.00038.
 [2] J-L. Hatchikian-Houdot et al., "Formal Hardware/Software Models for Cache Locking Enabling Fast and Secure Code", European Symposium on Research in Computer Security (ESORICS), 2024, doi: 10.1007/978-3-031-70896-1_8.
 [3] N. Gaudin et al., "A Fine-Grained Dynamic Partitioning Against Cache-Based Timing Attacks via Cache Locking", IEEE Computer Society Annual Symposium on VLSI (ISVLSI), 2024, doi: 10.1109/ISVLSI51997.2024.00001