



HAL
open science

Securing a High-Level Language Virtual Machine Through its ISA: Pharo as a Case Study

Quentin Ducasse, Pascal Cotret, Loïc Lagadec

► **To cite this version:**

Quentin Ducasse, Pascal Cotret, Loïc Lagadec. Securing a High-Level Language Virtual Machine Through its ISA: Pharo as a Case Study. GDR SOC², Jun 2021, Rennes, France. hal-04542157

HAL Id: hal-04542157

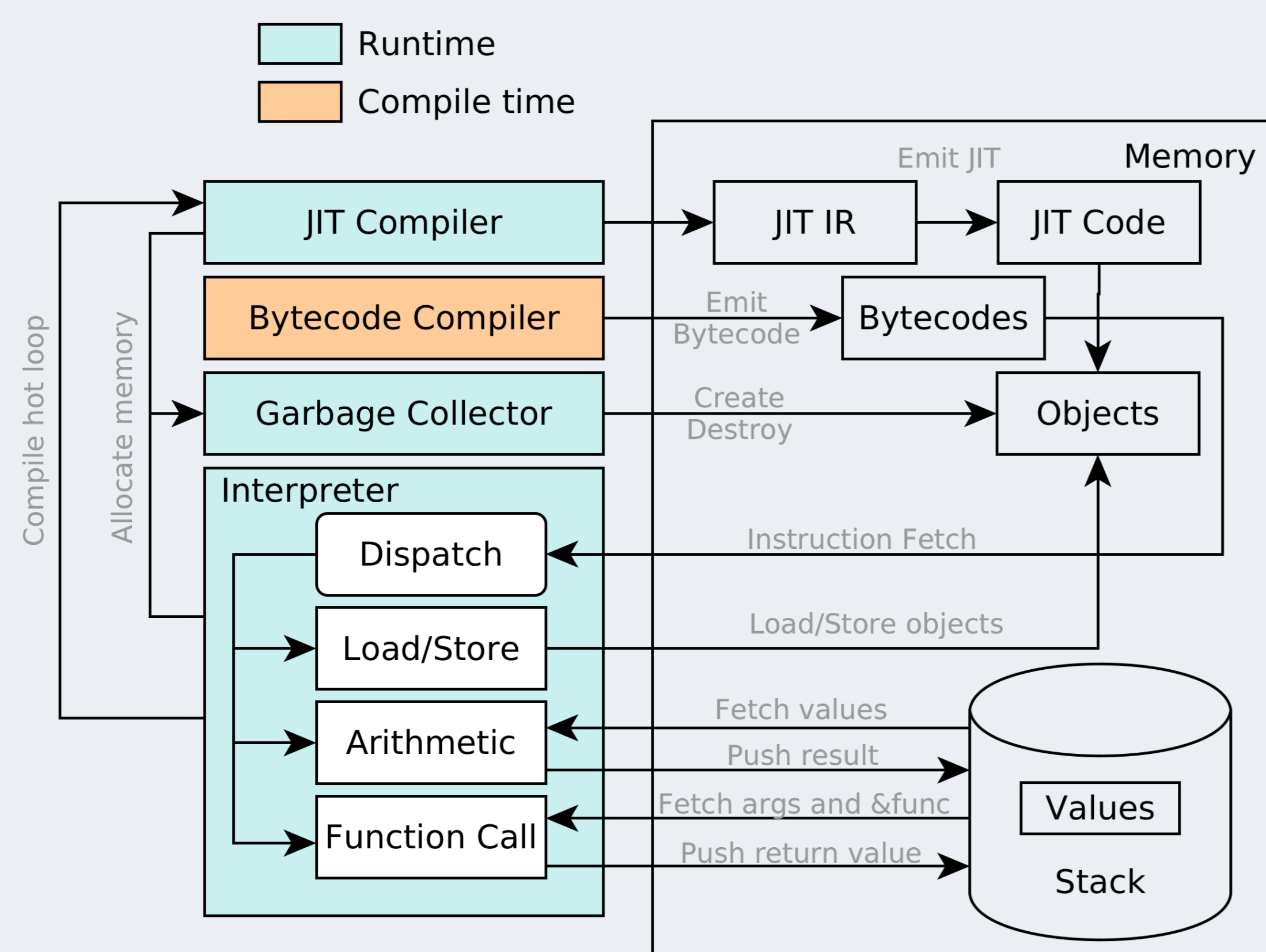
<https://hal.science/hal-04542157>

Submitted on 11 Apr 2024

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Language Virtual Machines JIT Compilation



- **Bytecode Compiler:** transforms source code in a high-level intermediate representation, **bytecodes**
- **Interpreter:** process the bytecodes and monitor instruction *hotness*
- **JIT compiler:** compiles *hot* methods/blocks/constants into **native code**
- **Garbage Collector:** allocates and deallocates memory

Memory Protection and Process Isolation

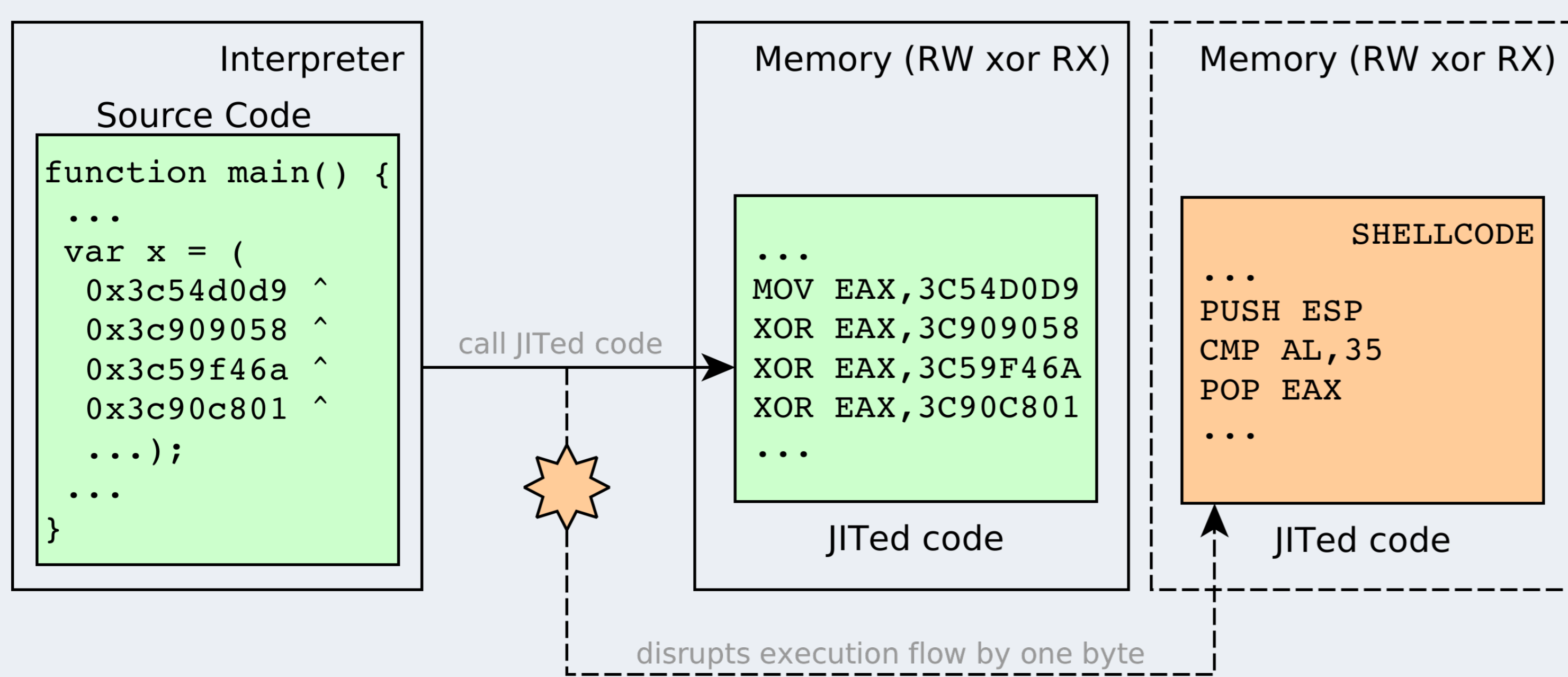
Memory protections are present at different **level** and **granularity**:

- **Memory Page Permissions:** *Read (R)*, *Write (W)* or *Execute (X)* granted to a given memory page (4kB)
- **Memory Protection Keys:** Userspace hardware mechanism to control page table permissions with key tags.
- **Domain Keys:** Per *process isolation* that can be split through several memory pages.
- **Trusted Execution Environments:** Hardware extension enabling isolated execution environments called *enclaves*.

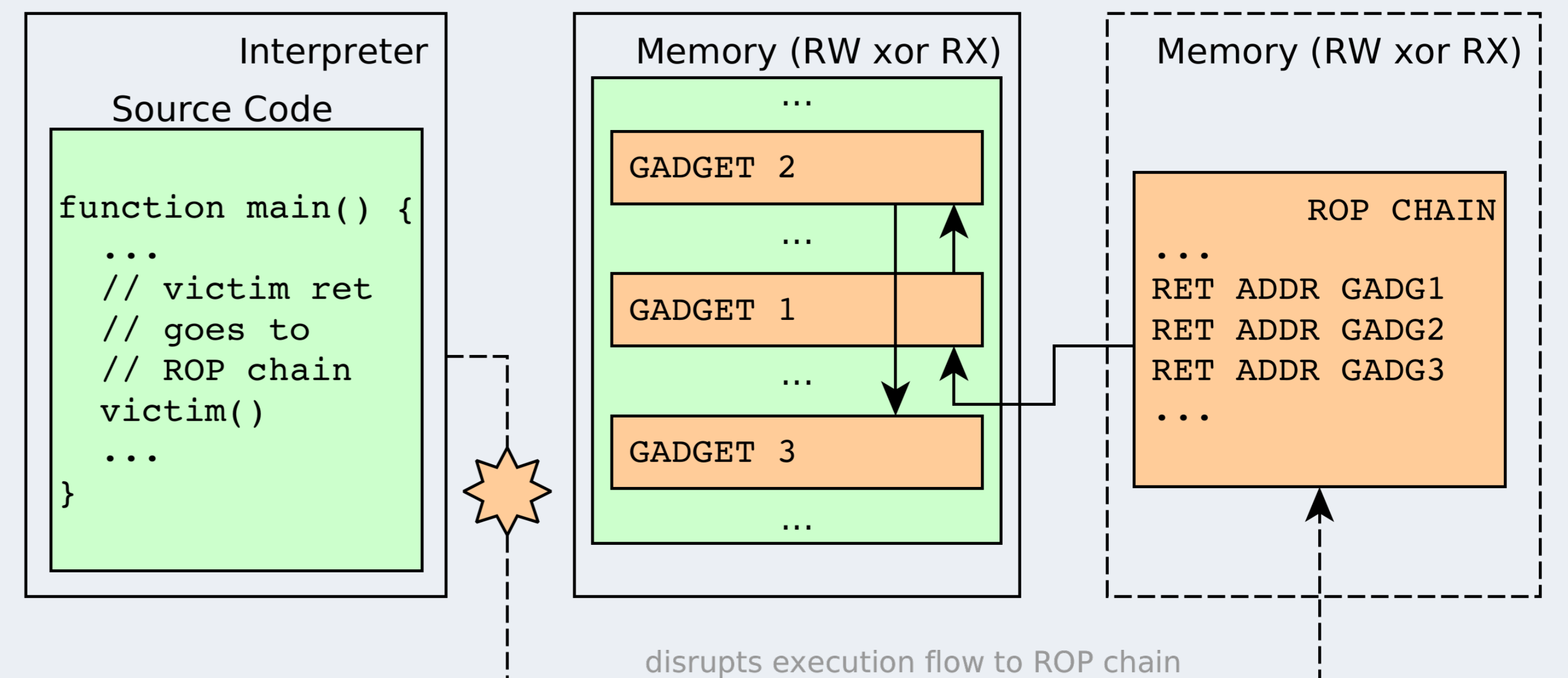
Manufacturers propose their TEEs such as **ARM Trustzone** or **Intel Software Guard Extension (SGX)** as well as **RISC-V Keystone** [1].

Protections are enforced on JIT engines using $W \oplus X$, **Data Execution Prevention (DEP)**, **Address Space Layout Randomization (ASLR)** or **Control-Flow Integrity (CFI)**. More specific protections have also been developed by integrating **MPK** [2] or **SGX** [3].

Attack on Virtual Machines and their JIT Engines



JIT Spraying [4] forces the JIT compilation of a **XOR chain** to produce a set of constants in JITed memory then disrupts the execution flow by one byte to reveal the hidden shellcode.



JIT-ROP [5] creates the payload from available **gadgets** in memory, either already present or JIT compiled directly by the attacker. The execution flow is disrupted to a return chain to launch the shellcode.

Pharo as a Case Study Virtual Machine for Security

Why choose Pharo?

- **Migration in Process or Completed** to other ISAs (ARM v8).
- **Simpler** than heavily engineered Java or JavaScript Virtual Machines.
- **Complex enough** to be in commercial use and require the security aspect.
- **High-level Language Test Harness** of hundreds of *ISA-agnostic* tests.

A VM implementation for a new ISA runs through three main phases:

- 1 **Unit-test** a **simulation** of the VM in the development environment itself.
- 2 **Emulate** the whole system on an **architecture emulator** (QEMU/Unicorn).
- 3 **Execute** the system on the **real hardware**.

Objectives

- Port the Pharo VM to the RISC-V ISA.
- Replay well-known attacks in a controlled environment.
- Design enclave behaviour on critical components using Keystone.
- Design instructions to handle isolation.
- Test the implementation in hardware (Beagle-V, CVA6).

Open Questions

- Are the attacks replayable on RISC-V?
- What defines a pertinent instruction for JIT engines?

References

- [1] D. Lee, D. Kohlbrenner, S. Shinde, K. Asanović, and D. Song, "Keystone: An open framework for architecting trusted execution environments," in *Proceedings of the Fifteenth European Conference on Computer Systems*, 2020, pp. 1–16.
- [2] T. Park, K. Dhondt, D. Gens, Y. Na, S. Volckaert, and M. Franz, "Nojitsu: Locking down javascript engines," in *Symposium on Network and Distributed System Security (NDSS)*, 2020.
- [3] T. Frassetto, D. Gens, C. Liebchen, and A.-R. Sadeghi, "Jitguard: hardening just-in-time compilers with SGX," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017, pp. 2405–2419.
- [4] D. Blazakis, "Interpreter exploitation: Pointer inference and JIT spraying," *BlackHat DC*, 2010.
- [5] K. Z. Snow, F. Monrose, L. Davi, A. Dmitrienko, C. Liebchen, and A.-R. Sadeghi, "Just-in-time code reuse: On the effectiveness of fine-grained address space layout randomization," in *2013 IEEE Symposium on Security and Privacy*. IEEE, 2013, pp. 574–588.

At a glance

- **Problem:** JIT engines are powerful but vulnerable pieces of software.
- **Approach:** involving the ISA in a high-level component provides more security guarantees.
- **Implementation:** Pharo VM on RISC-V adding instruction extensions and enclave support.