January 10, 2024
SVD'24 workshops, Neuchâtel

# Towards The Formal Verification of Security Monitor For Confidential Computing

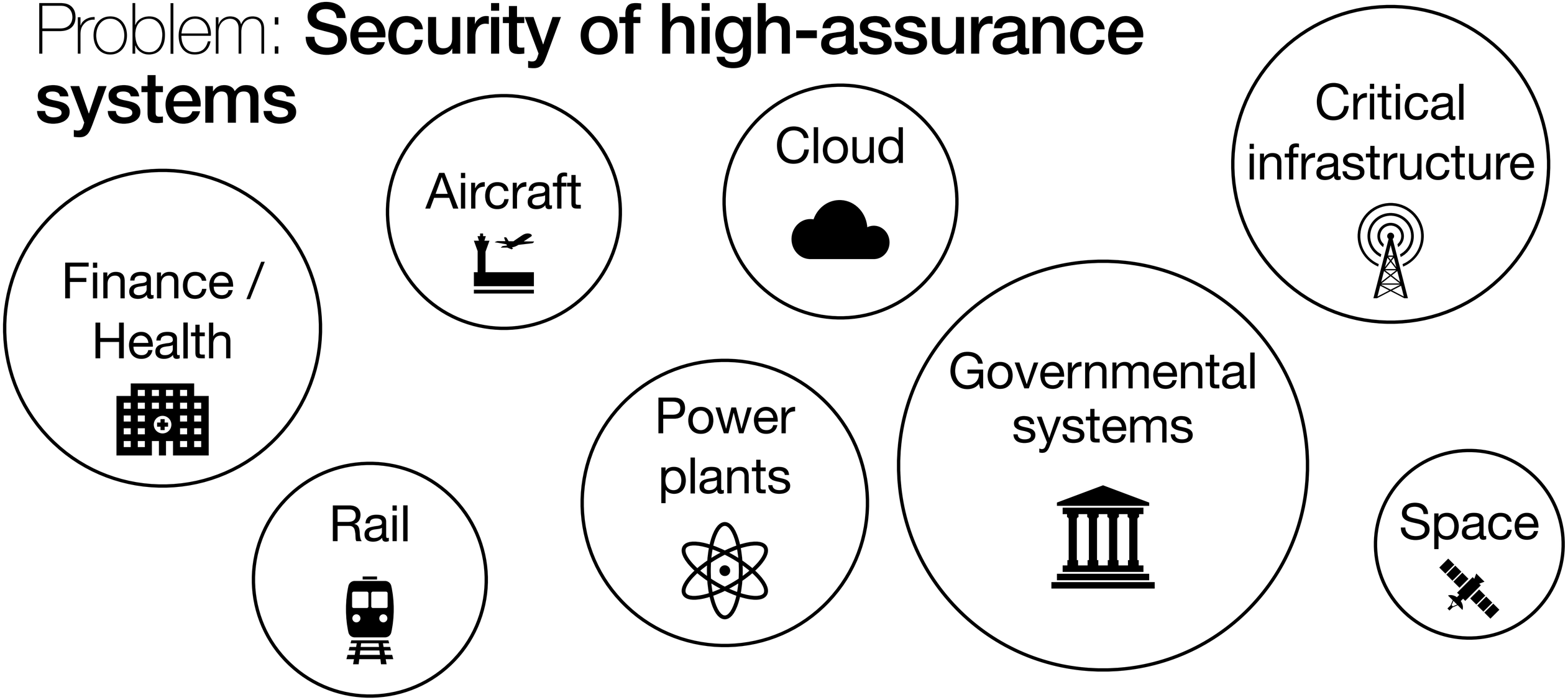
`Goal, Methodology, Demo, Challenges`

Lennard Gäher
IBM Research - Zürich
MPI-SWS, Germany
lennard.gaeher@ibm.com

Wojciech Ozga
IBM Research - Zürich
woz@zurich.ibm.com

How much does your life and security depend on computers?

# Problem: **Security of high-assurance systems**




Successful attacks on **high-assurance systems** might lead to catastrophe, social disturbances, political instability.

SIGN IN

The Register

{* SECURITY *}

# No big deal... Kremlin hackers 'jumped air-gapped networks' to pwn US power utilities

'Hundreds' of intrusions, switch could be pulled anytin

Richard Chirgwin

80

The US Department of Homeland Security government hackers of penetrating America

Uncle Sam's finest reckon Moscow's agent networks within US electric utilities – to the have virtually pressed the off switch in cont Yanks, and plunged America into darkness

The hackers, dubbed Dragonfly and Energe 2016, and continued throughout 2017 and i

INSIDER

Log in Subscribe

HOME > TECH

# The hackers that attacked a major US oil pipeline say it was only for money — here's what to know about DarkSide

Natasha Dailey May 10, 2021, 5:49 PM


npr

# A 'Worst Nightmare' Cyberattack: The Untold Story Of The SolarWinds Hack

April 16, 2021 · 10:05 AM ET

Heard on All Things Considered


An NPR investigation into the SolarWinds attack reveals a hack unlike any other, launched by a sophisticated adversary intent on exploiting the soft underbelly of our digital lives.

*Zoë van Dijk for NPR*

REUTERS

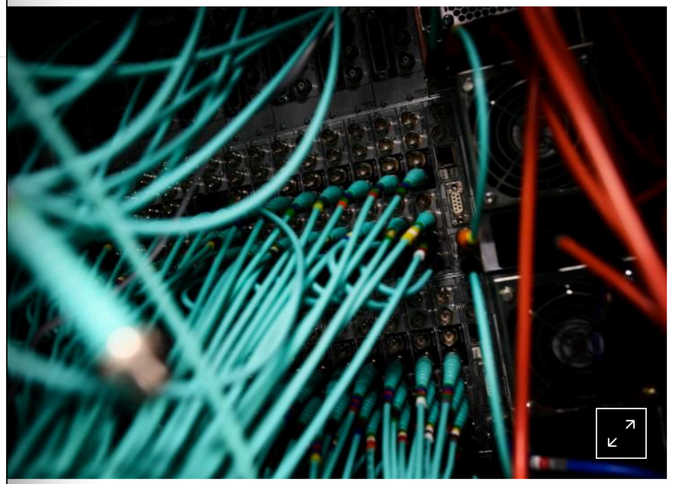
INTERNET NEWS

JULY 10, 2017 / 1:57 PM / UPDATED 5 YEARS AGO

# Foreign hackers probe European critical infrastructure networks: sources

By Mark Hosenball

LONDON (Reuters) - Cyber attackers are regularly trying to attack data networks connected to critical national infrastructure systems around Europe, according to current and former European government sources with knowledge of the issue.



and computers are seen inside a data centre at an office in the heart of the financial in London, Britain May 15, 2017. REUTERS/Dylan Martinez

The New York Times

# *Hackers Are Targeting Nuclear Faci Homeland Security Dept. and F.B.I.*

The Wolf Creek Nuclear power plant in Kansas in 2000. The corporation that runs the plant was targeted by hackers. David Eulitt/Capital Journal, via Associated Press

By Nicole Perlroth

July 6, 2017

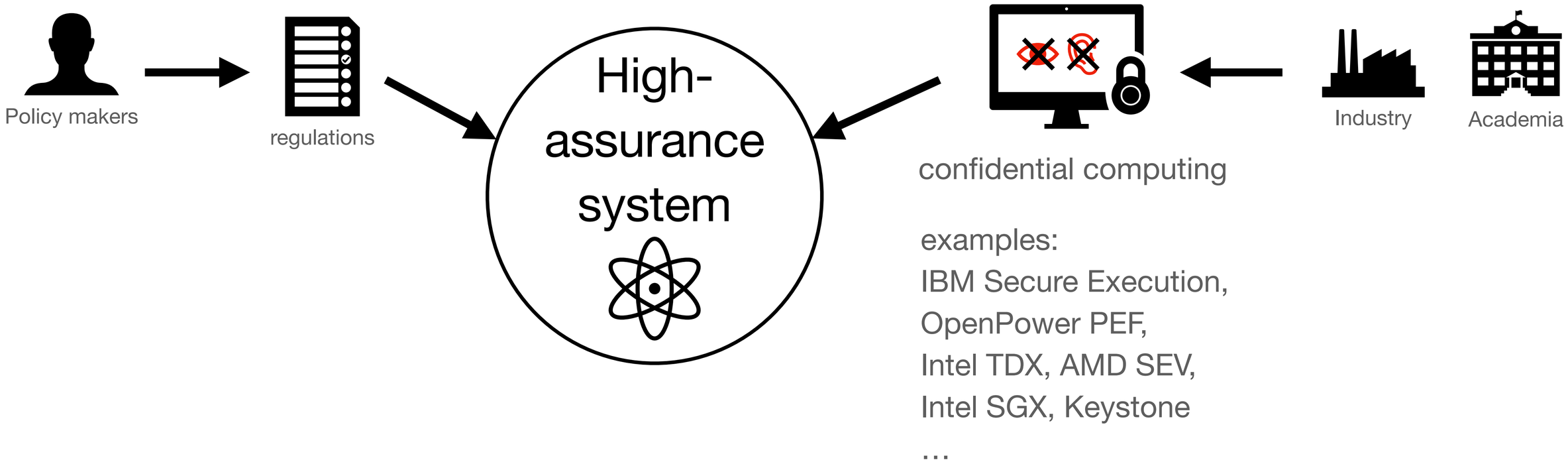
# a do Ministério expõe dados pessoais de mais de 200 milhões de brasileiros

Erro em sistema federal de registro de casos de covid permitiu acesso, durante seis meses, a informações pessoais de todos os brasileiros cadastrados no SUS e clientes de plano de saúde

Fabiana Cambricoli, O Estado de S.Paulo

02 de dezembro de 2020 | 05h00

# Problem: **How to formally verify security properties of confidential computing systems?**




Security-critical systems are subject to **regulations** and require **formal verification**.

Goal: **Build an open-source formally verified security monitor for confidential computing.**

IBM / ACE-RISCV

Type / to search

Code | Issues | Pull requests | Actions | Projects | Wiki | Security | Insights | Settings

ACE-RISCV Public

Edit Pins | Unwatch 4 | Fork 10 | Starred 13

main | 1 Branch | 0 Tags | Go to file t | Add file | Code | About

README | Apache-2.0 license

# Assured Confidential Execution (ACE) for RISC-V

ACE Build passing

ACE-RISCV is an open-source project, whose goal is to deliver a confidential computing framework with a formally proven security monitor. It is based on a canonical architecture and targets RISC-V with the goal of being portable to other architectures. The formal verification efforts focus on the security monitor implementation. We invite collaborators to work with us to push the boundaries of provable confidential computing technology.

**This is an active research project, without warranties of any kind.** Please read our paper to learn about our approach and goals.

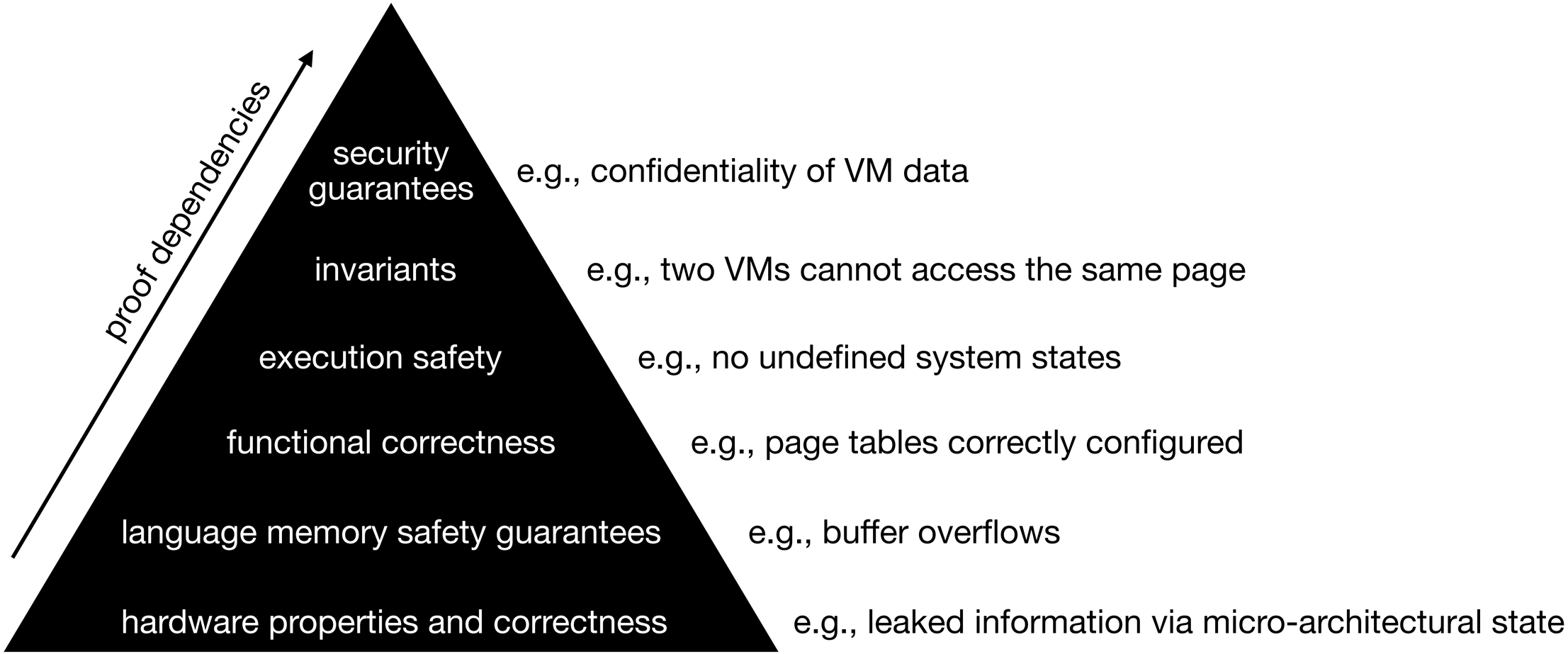
We are currently building on RISC-V with hypervisor extentions. We will adapt the AP-TEE extension once it is ratified.

## Quick Start

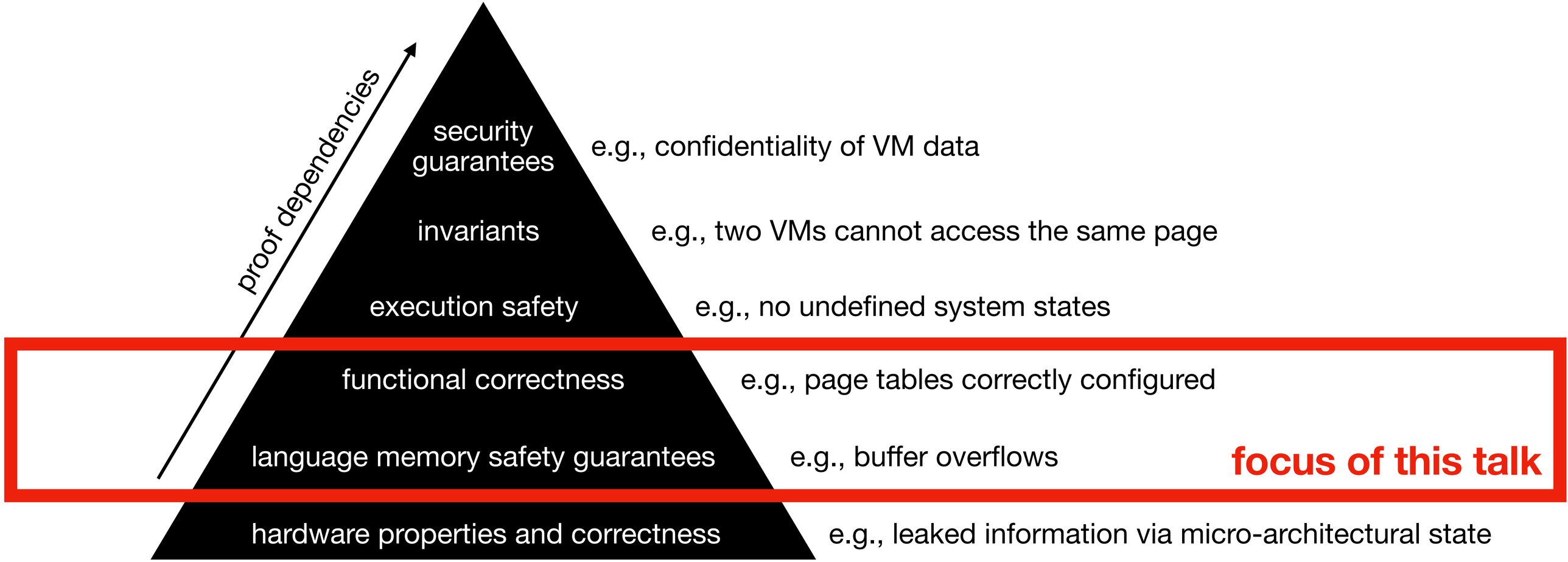
Follow instructions to run a sample confidential workload under an untrusted Linux-based hypervisor in an emulated RISC-V environment.

**https://github.com/IBM/ACE-RISCV**

# What has to be proven?



# What has to be proven?



# Rust provides memory safety guarantees

**Safe Rust**

- Memory safety,
- Type system providing ownership, borrowing, lifetimes.

**Unsafe Rust**

- Enables C-style pointer accesses
- Gives no memory safety guarantees.

Read more: https://tiemoko.com/blog/blue-team-rust/

# Verification using RefinedRust

Presented at RW'23

**Goal**: verify memory safety & functional correctness & panic-freedom

**Automatic translation** from Rust (MIR) into Radium

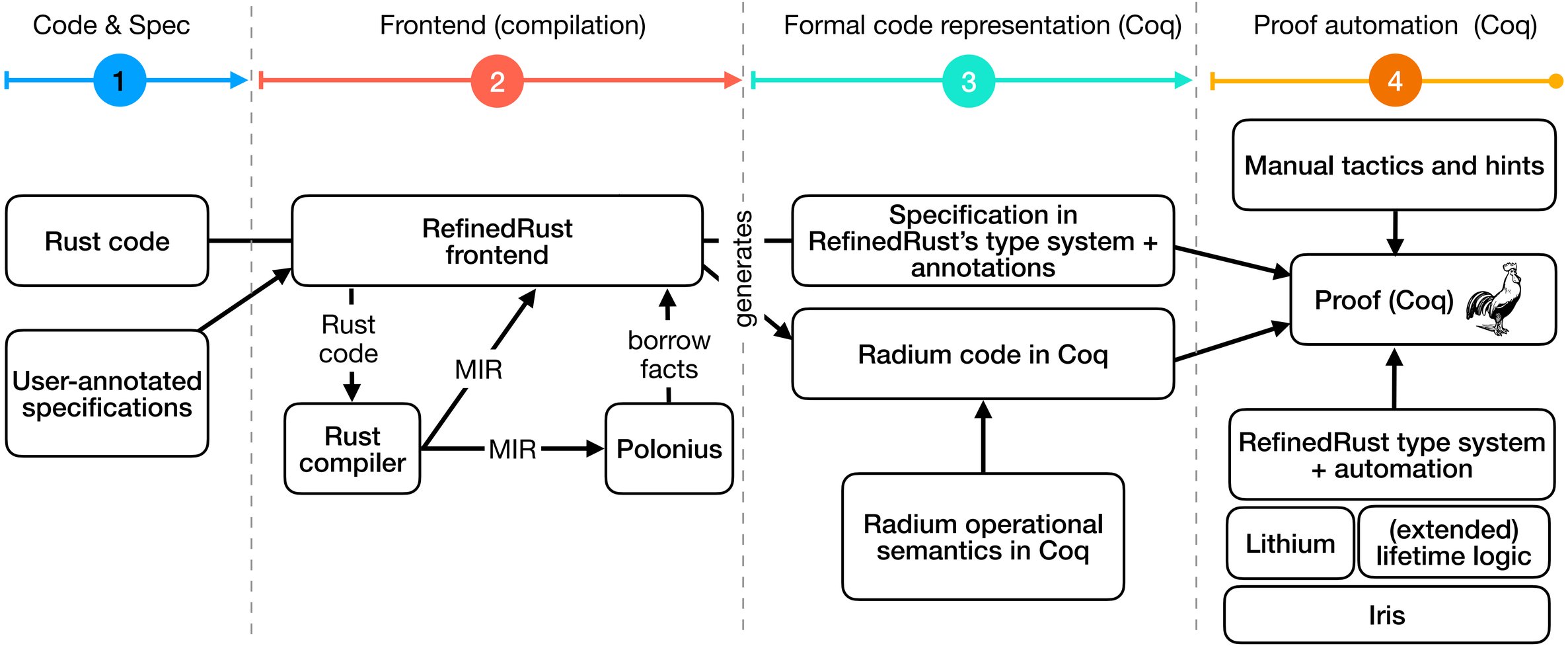
**Proof automation** using the Lithium separation logic engine

Radium **operational semantics** for Rust

**Refinement type system** with semantic model inspired by RustBelt

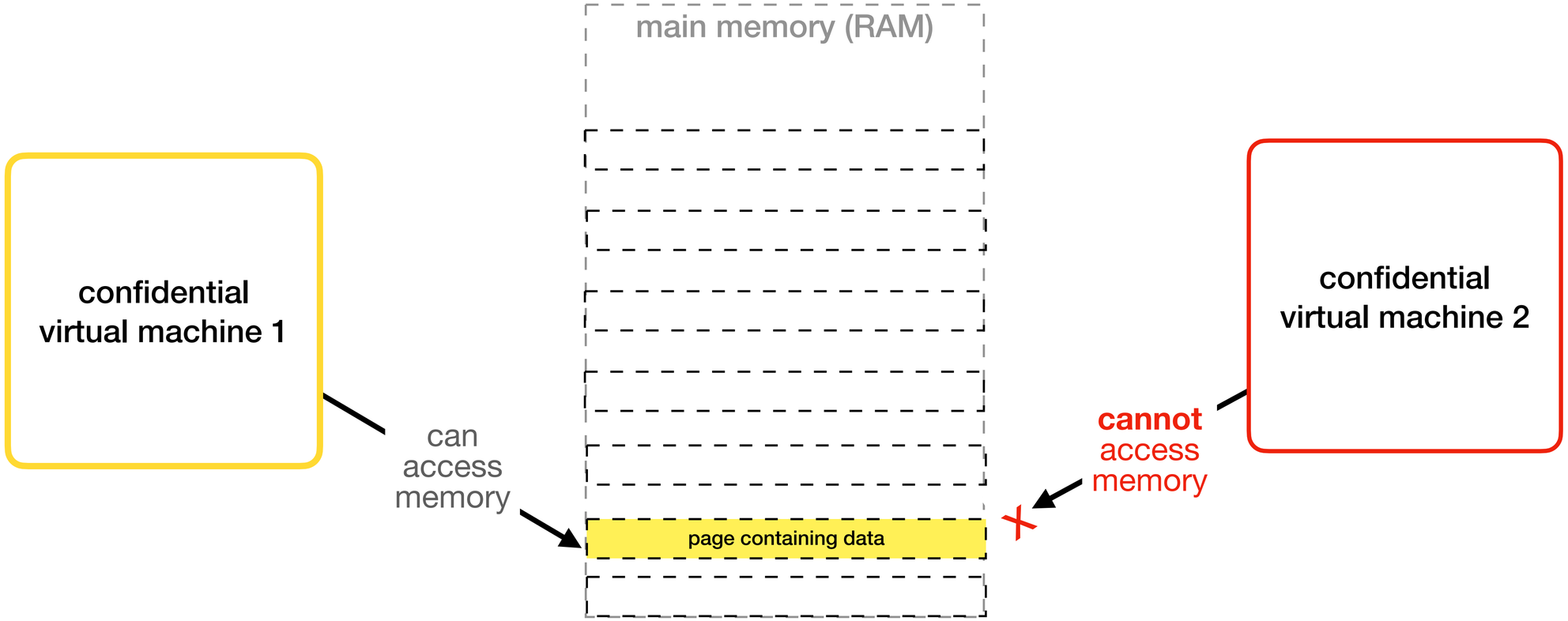
Coq proof assistant

# Architecture of RefinedRust



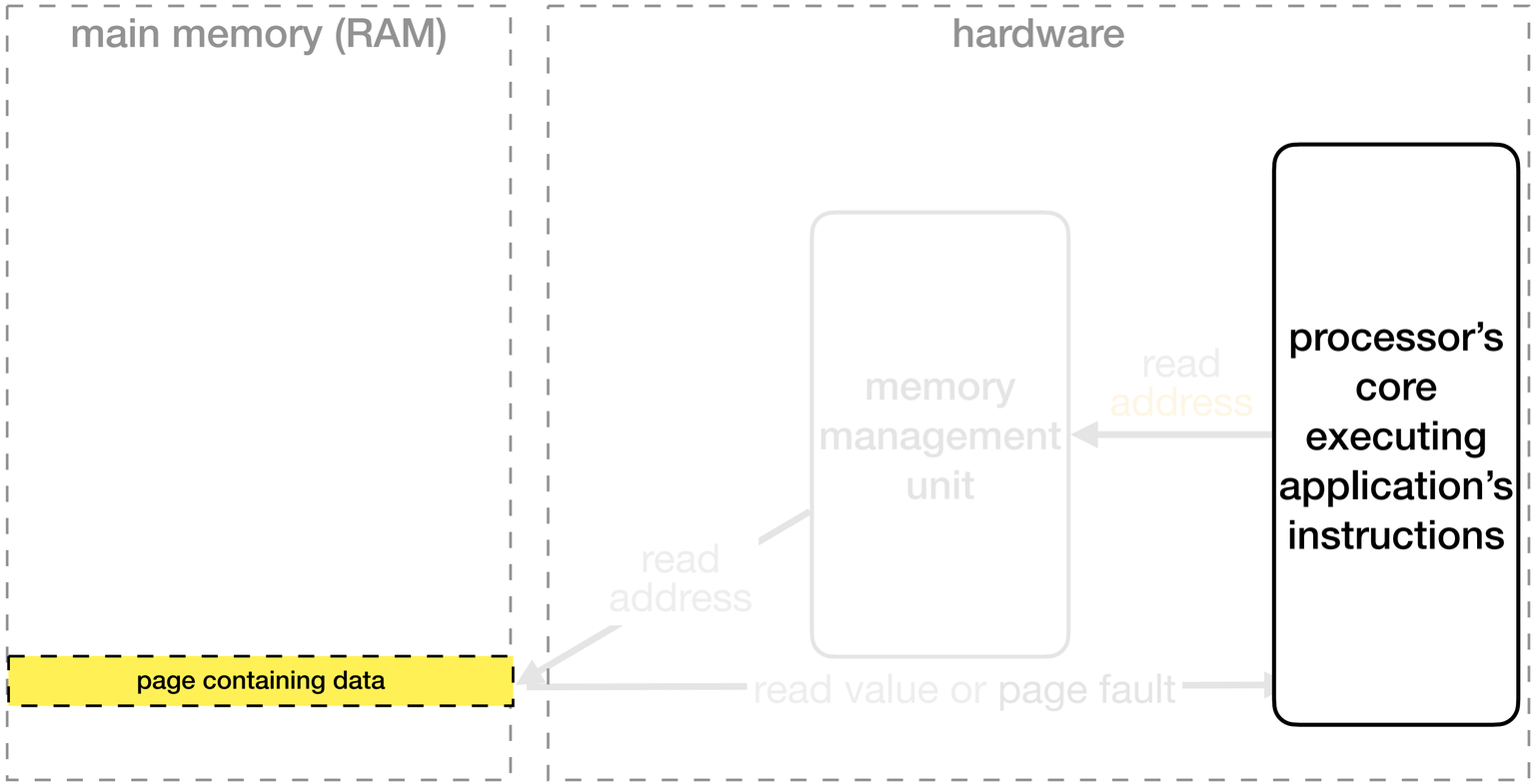
# Example: **Memory Allocation**

Goal: To prove that two different confidential VMs cannot access the same physical memory region in the confidential memory.



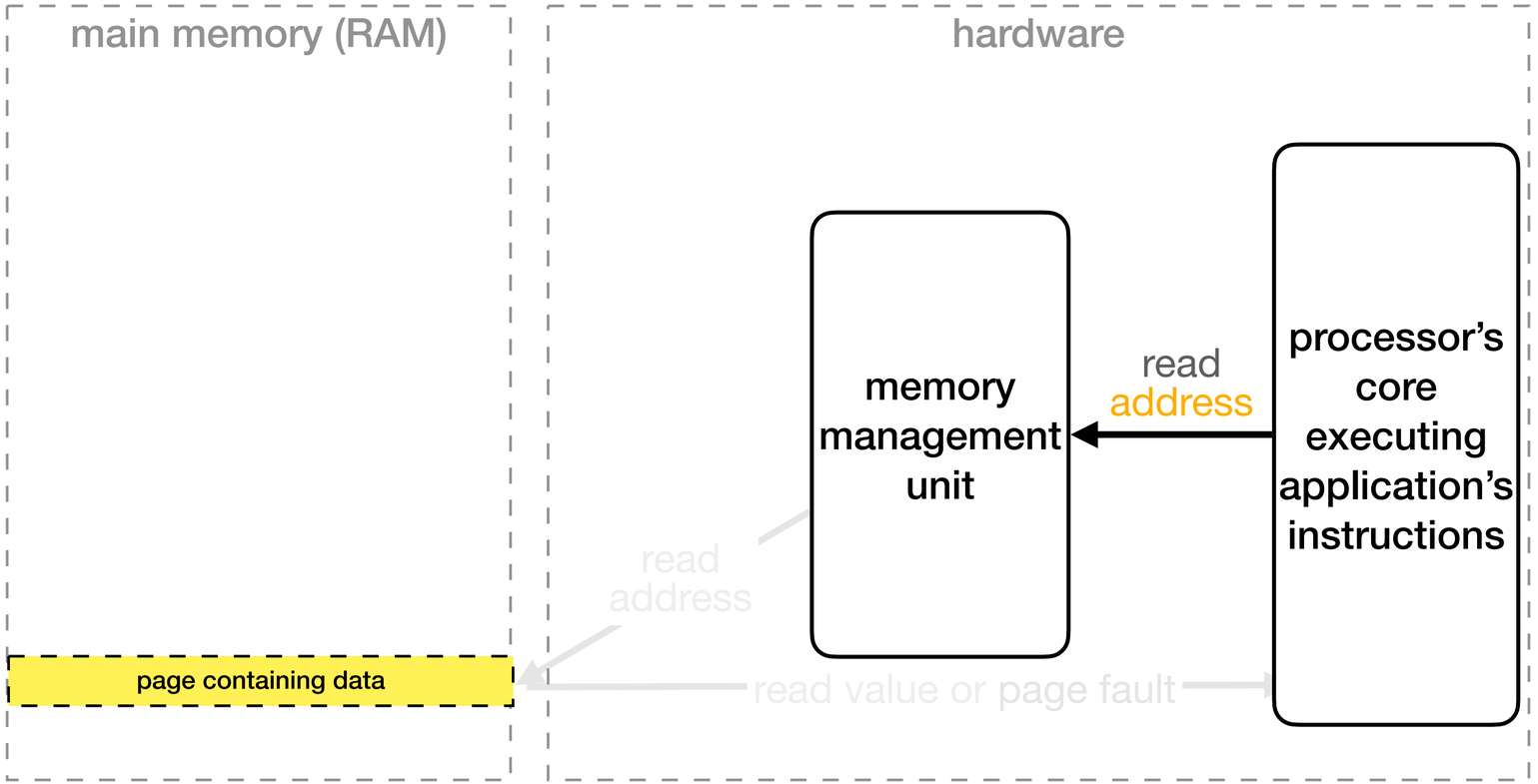
# Example: **Memory Allocation**

Goal: To prove that two different confidential VMs cannot access the same physical memory region in the confidential memory.



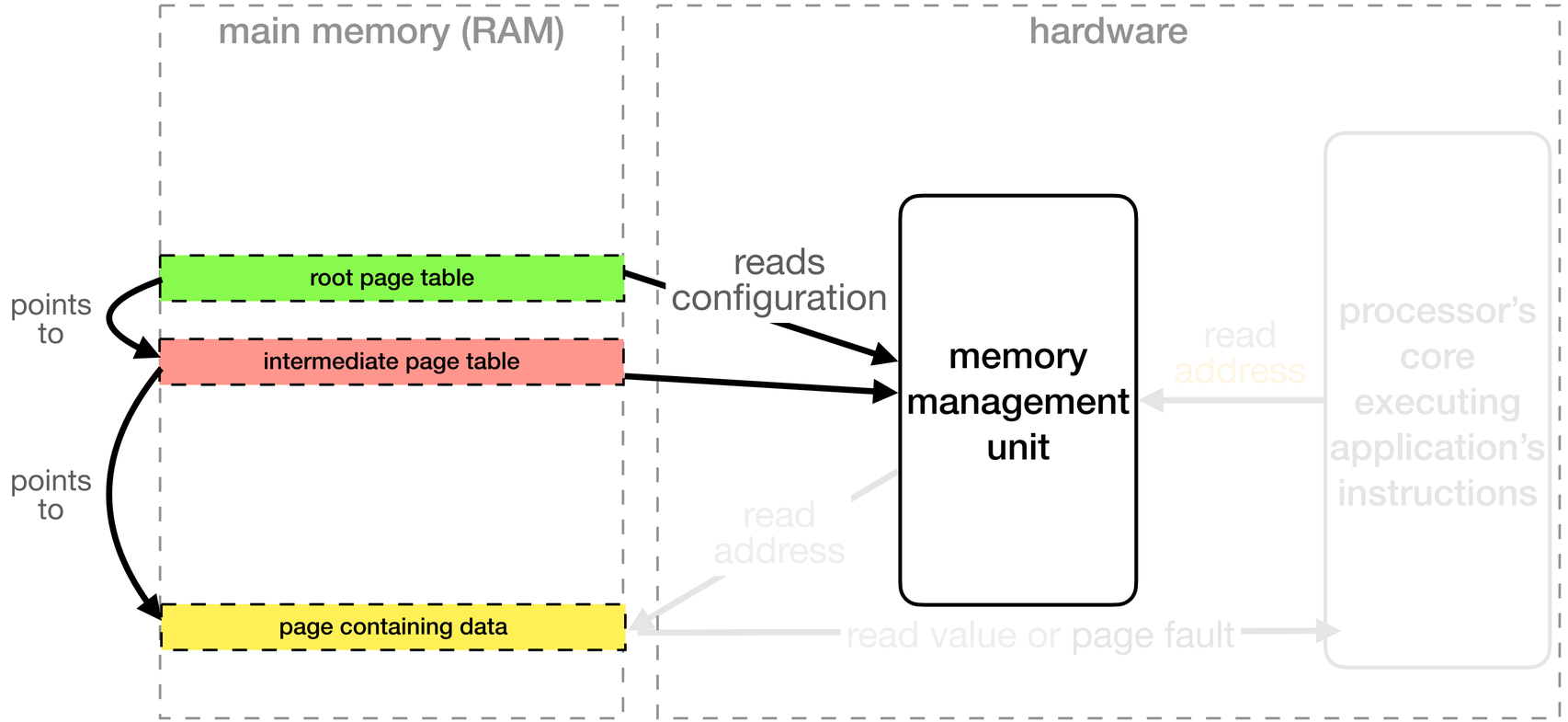
# Example: **Memory Allocation**

Goal: To prove that two different confidential VMs cannot access the same physical memory region in the confidential memory.



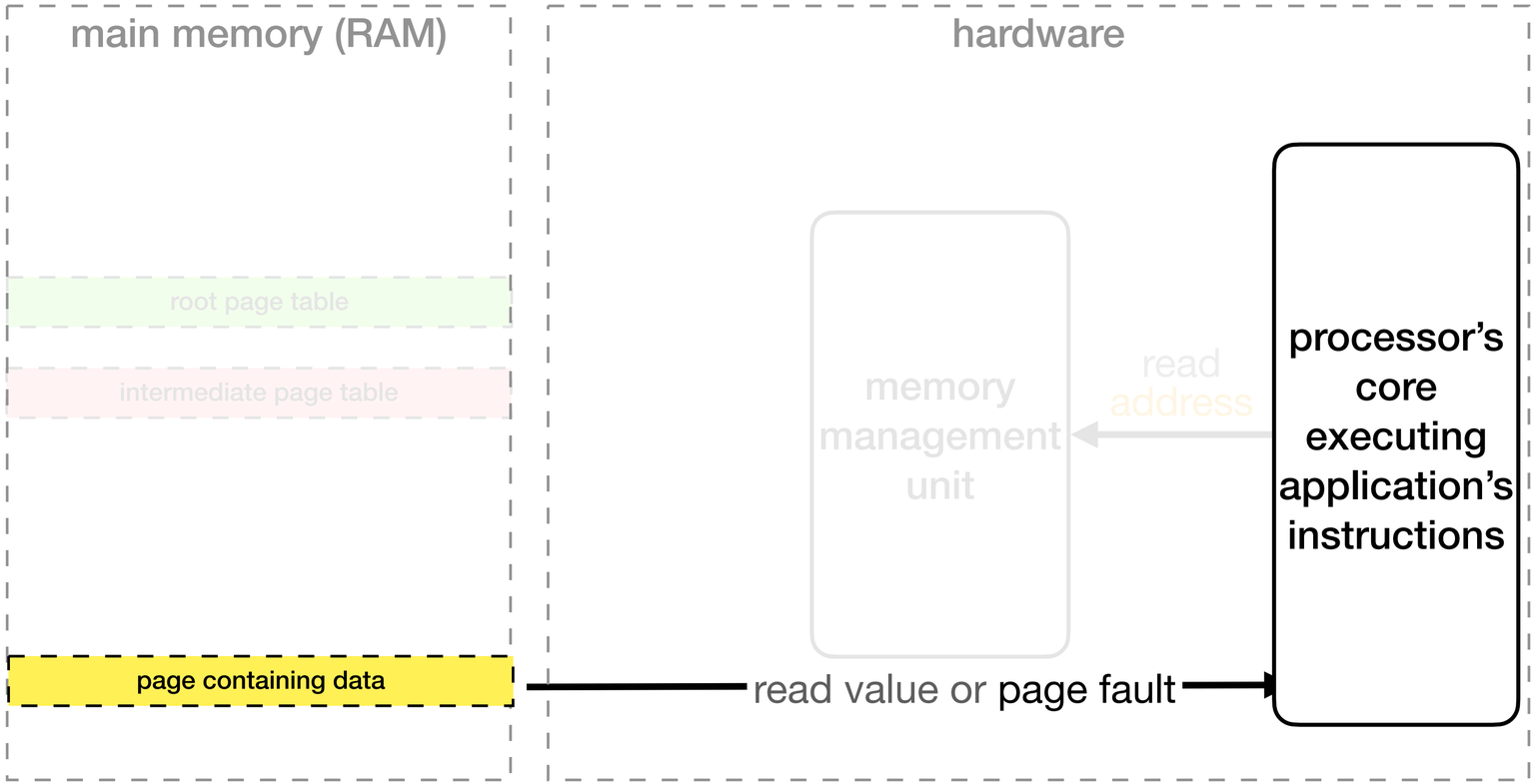
# Example: **Memory Allocation**

Goal: To prove that two different confidential VMs cannot access the same physical memory region in the confidential memory.



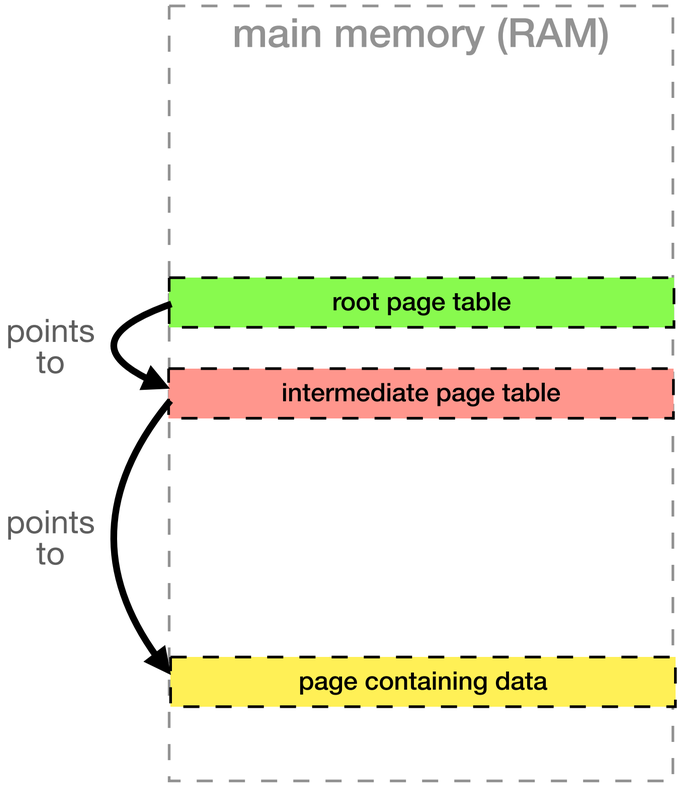
# Example: **Memory Allocation**

Goal: To prove that two different confidential VMs cannot access the same physical memory region in the confidential memory.



# Example: **Memory Allocation**

Goal: To prove that two different confidential VMs cannot access the same physical memory region in the confidential memory.

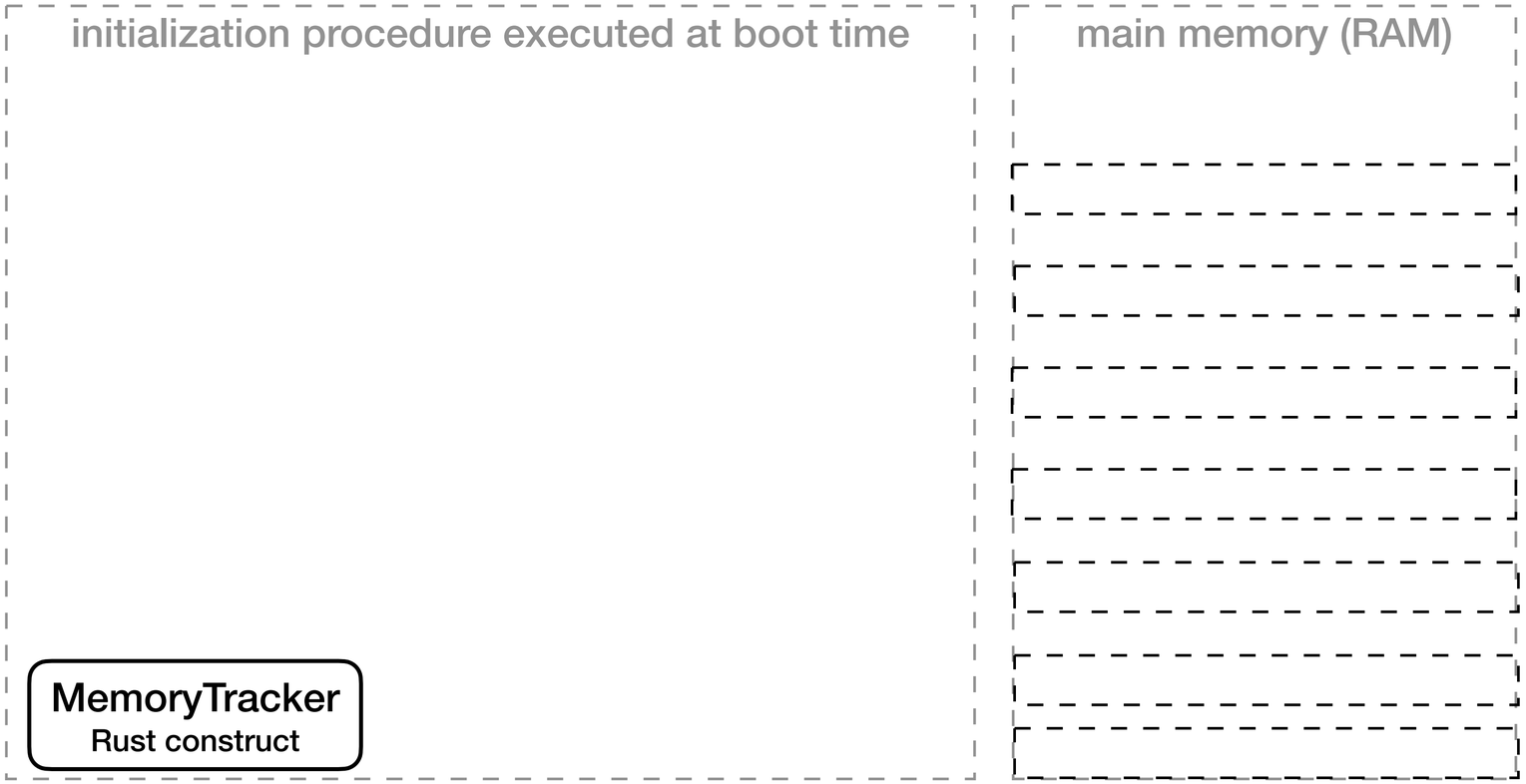



We must formally verify the functional correctness of the page table configuration.

Let's leverage Rust's type system with its ownership and memory safety guarantees!

# Example: **Memory Allocation**

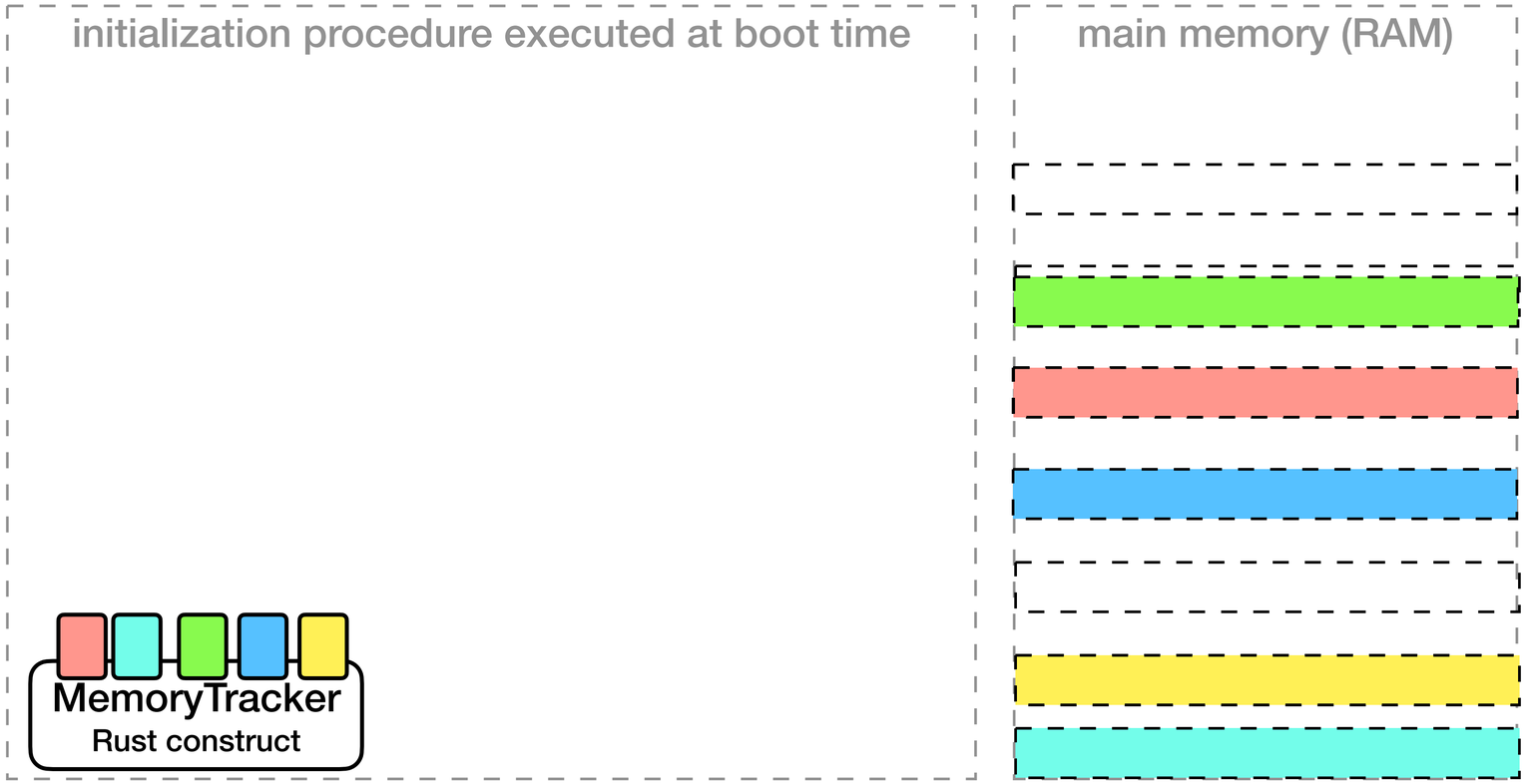
Goal: To prove that two different confidential VMs cannot access the same physical memory region in the confidential memory.




```rust
pub struct MemoryTracker {
    pages: Vec<Page<UnAllocated>>,
}
```

# Example: **Memory Allocation**

Goal: To prove that two different confidential VMs cannot access the same physical memory region in the confidential memory.

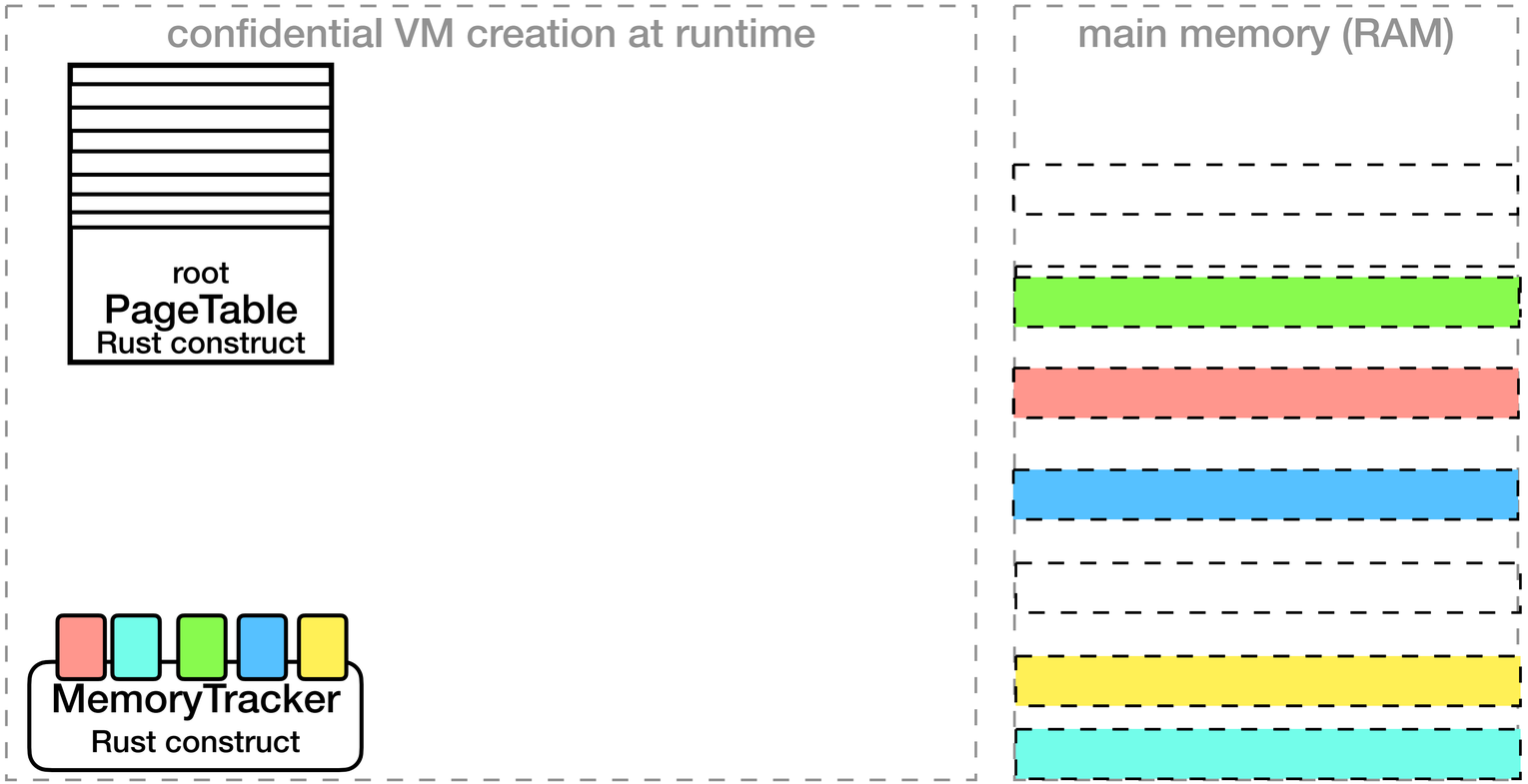



```rust
pub struct MemoryTracker {
    pages: Vec<Page<UnAllocated>>,
}

pub struct Page {
    address: usize,
    size: PageSize,
}
```

# Example: **Memory Allocation**

Goal: To prove that two different confidential VMs cannot access the same physical memory region in the confidential memory.

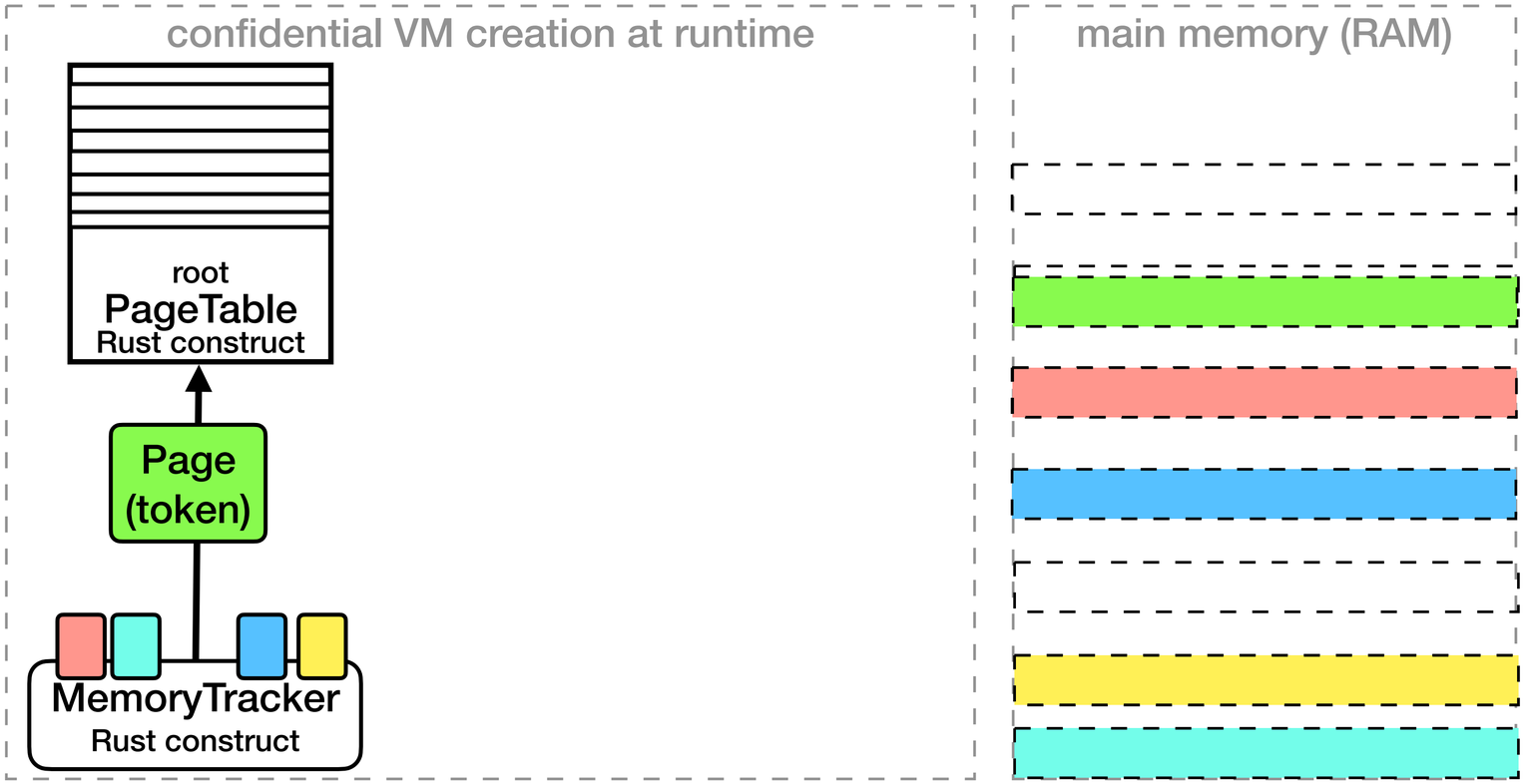



```rust
pub(super) struct PageTable {
    configuration_page: Page<Allocated>,
    entries: Vec<PageTableEntry>,
}

pub(super) enum PageTableEntry {
    Pointer(PageTable, PageTableConfiguration),
    Leaf(Page<Allocated>, PageTableConfiguration,
        PageTablePermission),
    NotValid,
}
```

# Example: **Memory Allocation**

Goal: To prove that two different confidential VMs cannot access the same physical memory region in the confidential memory.



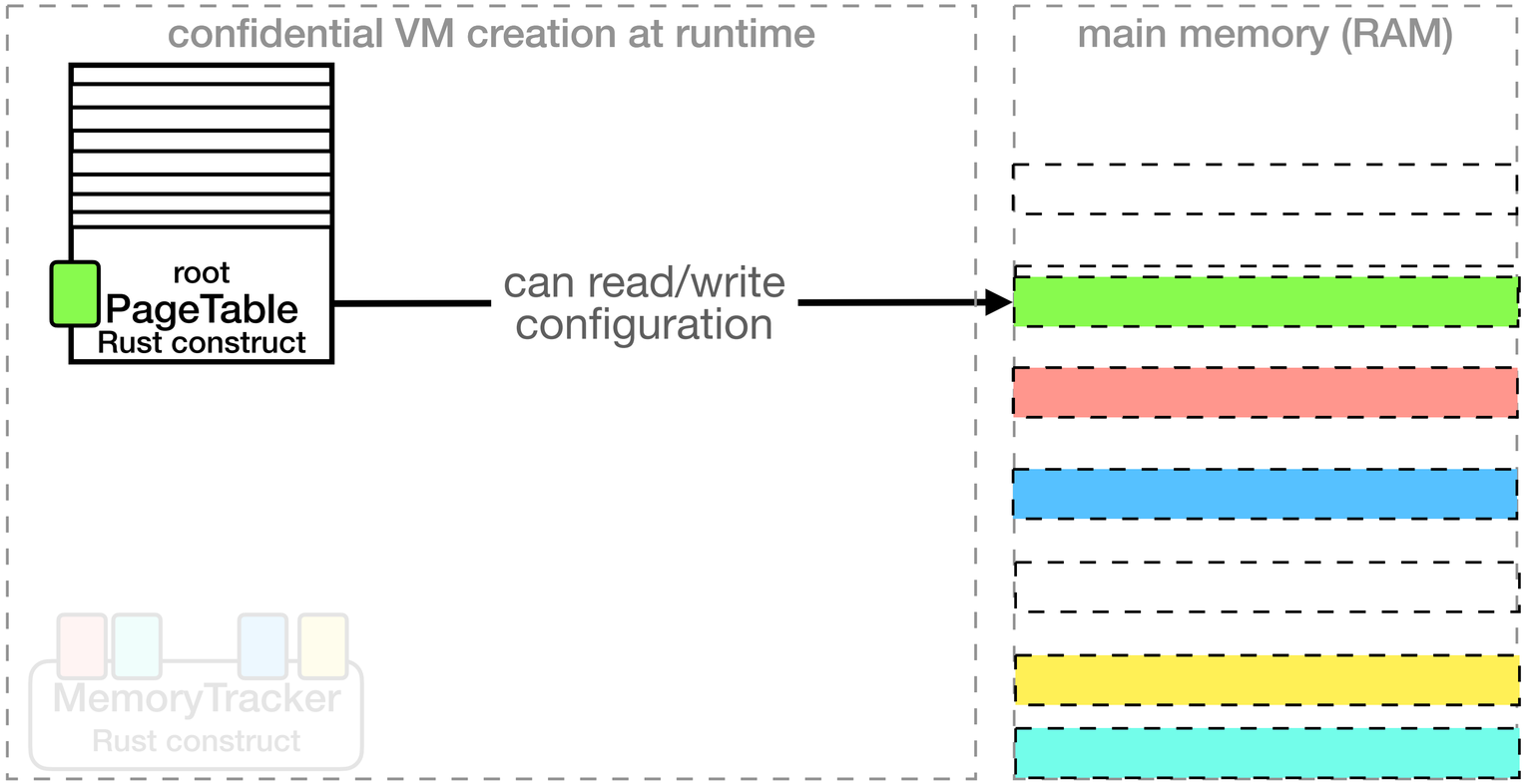

```rust
pub(super) struct PageTable {
    configuration_page: Page<Allocated>,
    entries: Vec<PageTableEntry>,
}
```

```rust
pub struct Page {
    address: usize,
    size: PageSize,
}
```

```rust
pub struct MemoryTracker {
    pages: Vec<Page<UnAllocated>>,
}
```

# Example: **Memory Allocation**

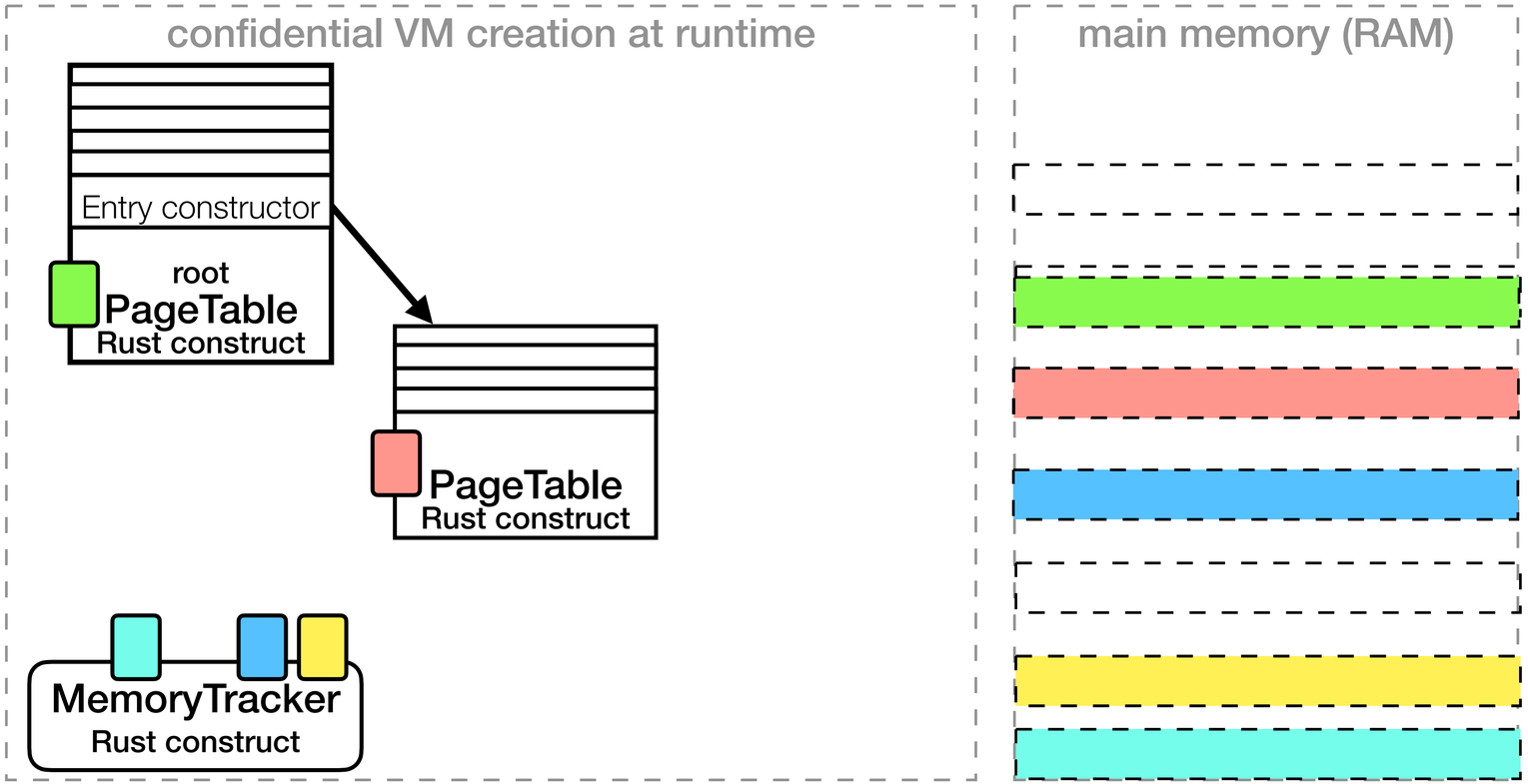
Goal: To prove that two different confidential VMs cannot access the same physical memory region in the confidential memory.




```rust
impl Page {
    pub fn write(&self, offset: usize, value: usize) {
        assert!(offset + core::mem::size_of::<usize>() <= self.size.in_bytes());
        unsafe {
            ((self.address.usize() + offset) as *mut usize).write_volatile(value)
        };
    }
}
```

# Example: **Memory Allocation**

Goal: To prove that two different confidential VMs cannot access the same physical memory region in the confidential memory.

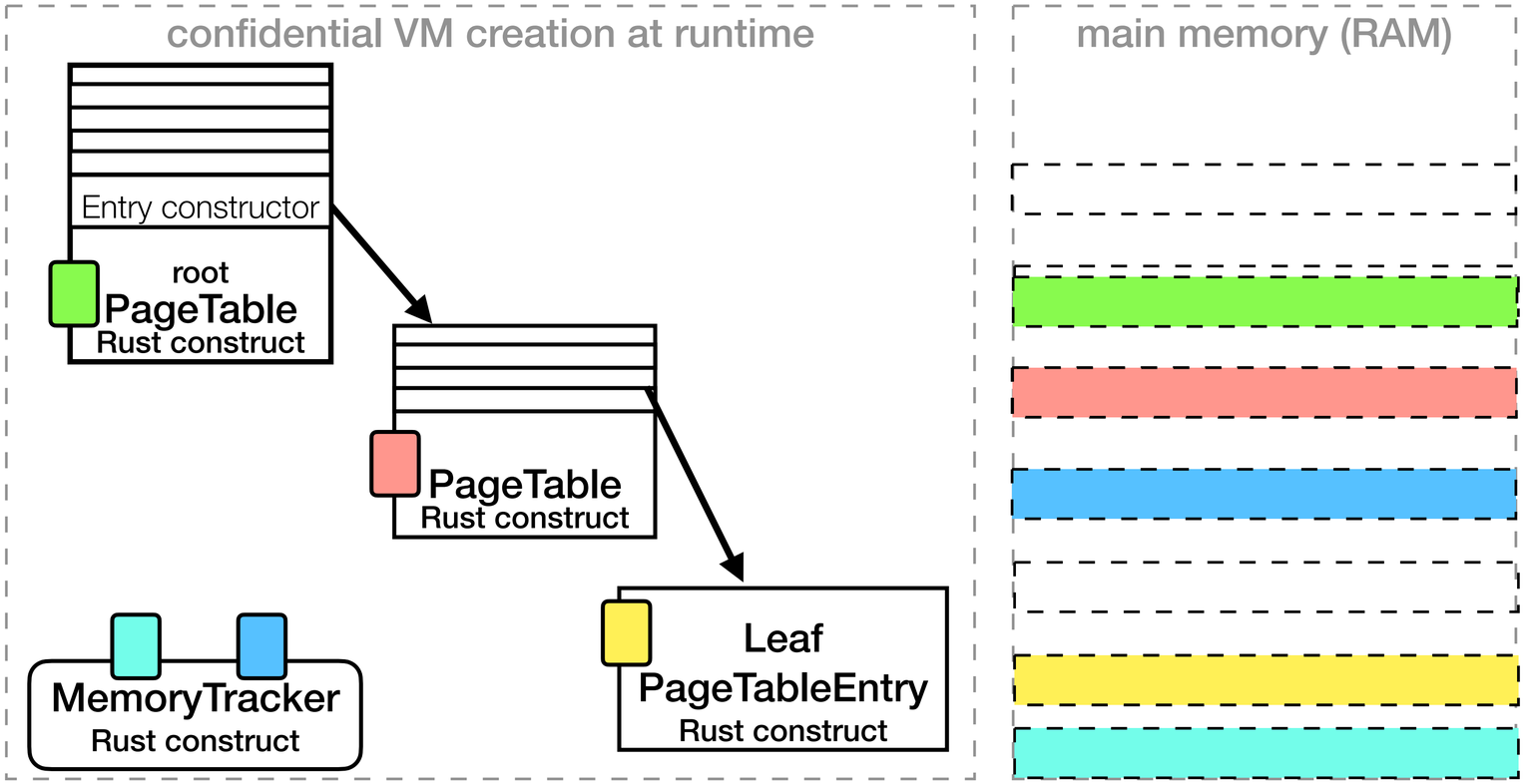



```rust
pub(super) struct PageTable {
    configuration_page: Page<Allocated>,
    entries: Vec<PageTableEntry>,
}

pub(super) enum PageTableEntry {
    Pointer(PageTable, PageTableConfiguration),
    Leaf(Page<Allocated>, PageTableConfiguration,
        PageTablePermission),
    NotValid,
}
```

# Example: **Memory Allocation**

Goal: To prove that two different confidential VMs cannot access the same physical memory region in the confidential memory.

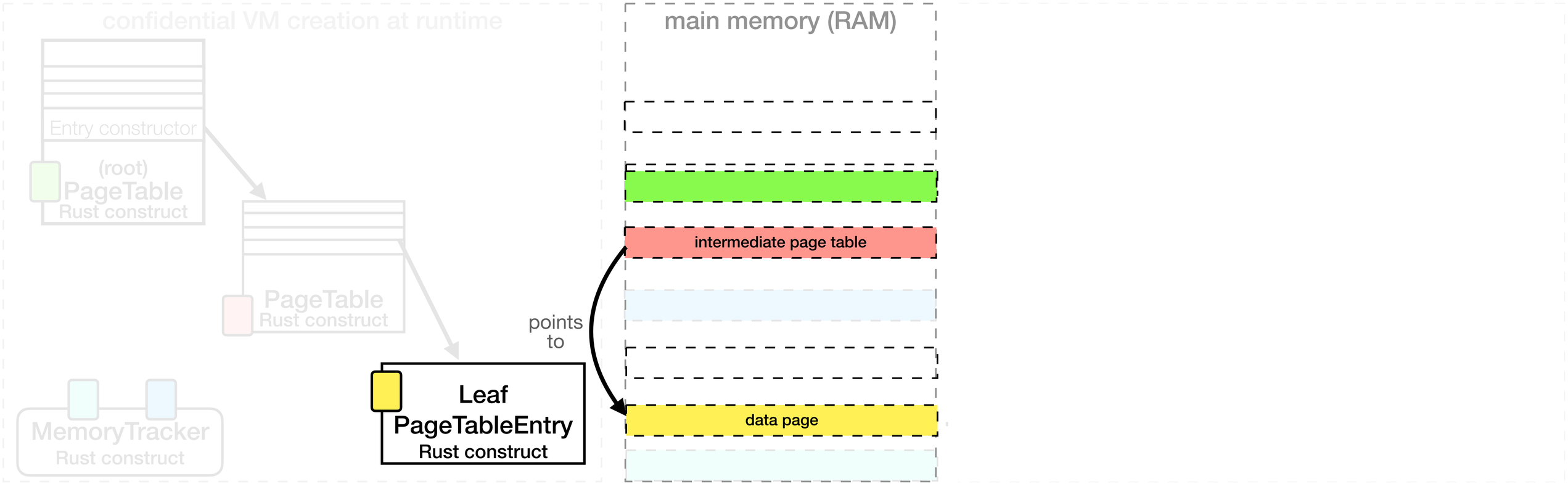



```rust
pub(super) struct PageTable {
    configuration_page: Page<Allocated>,
    entries: Vec<PageTableEntry>,
}

pub(super) enum PageTableEntry {
    Pointer(PageTable, PageTableConfiguration),
    Leaf(Page<Allocated>, PageTableConfiguration,
        PageTablePermission),
    NotValid,
}
```

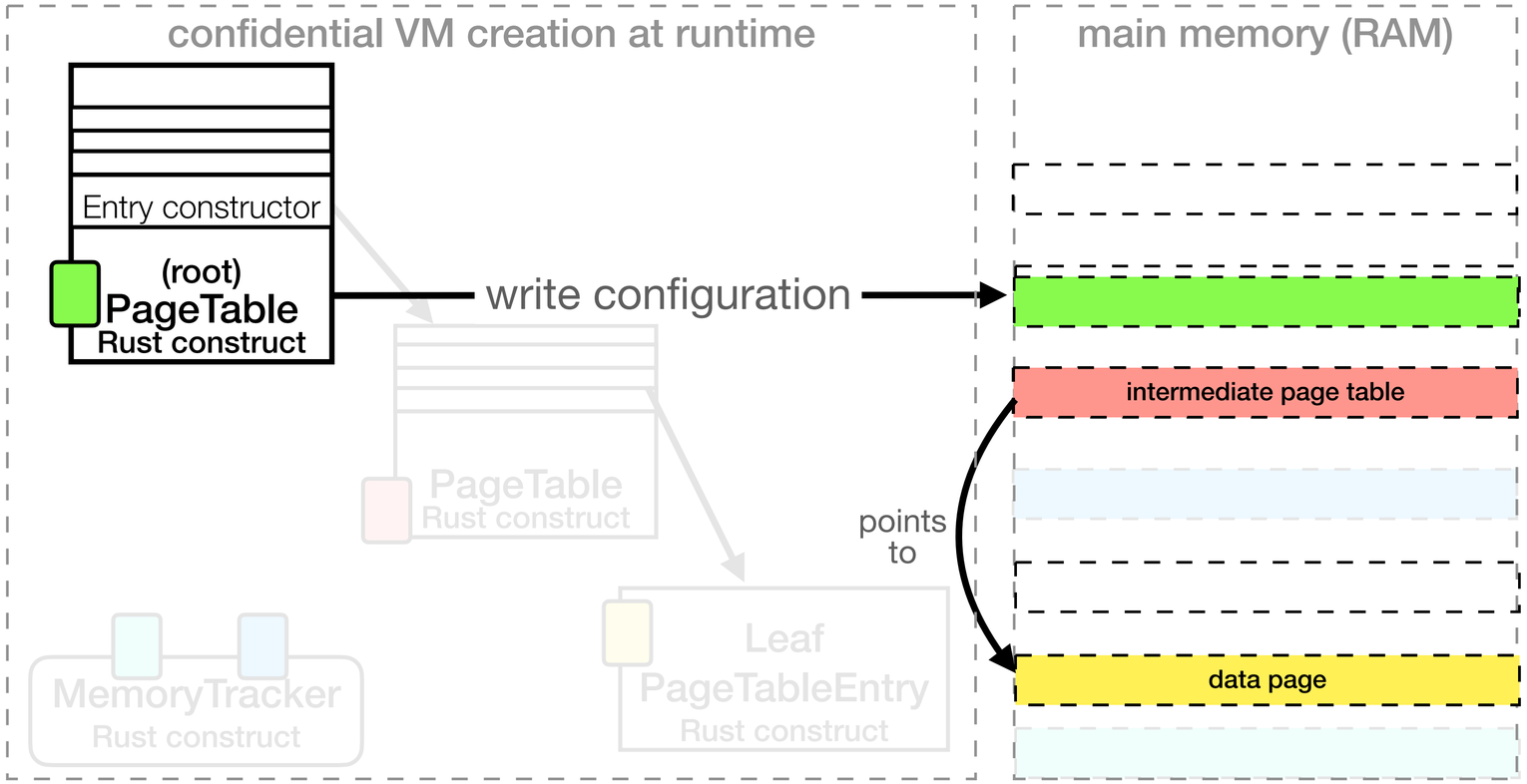
# Example: **Memory Allocation**

Goal: To prove that two different confidential VMs cannot access the same physical memory region in the confidential memory.



# Example: **Memory Allocation**

Goal: To prove that two different confidential VMs cannot access the same physical memory region in the confidential memory.




```rust
impl PageTable {
    fn set_entry(&mut self, index: usize, value: PageTableEntry) {
        self.configuration_page.write::<usize>(self.entry_size * index_in_page,
                                                                    value.encode())
    }
}

impl Page {
    pub fn write(&self, offset: usize, value: usize) {
        assert!(offset + core::mem::size_of::<usize>() <= self.size.in_bytes());
        unsafe {
            ((self.address.usize() + offset) as *mut usize).write_volatile(value)
        };
    }
}

impl PageTableEntry {
    pub fn encode(&self) -> usize {
        match self {
            PageTableEntry::Leaf(page, configuration, permissions) => {
                PageTableBits::Valid.mask()
                    | PageTableAddress::encode(page.address().usize())
                    | configuration.encode()
                    | permissions.encode()
            }
            _ => { ... }
```

# Demo: Verifying page tokens with RefinedRust

# Impressions on using Rust

- System engineering:
  - Sufficient ecosystem for writing embedded systems,
  - Need to be careful to not falling into C style programming with Rust when writing low-level code (hint: treat “unsafe” as a red flag and build safe abstraction over hardware).
- Formal verification
  - Ecosystem less mature than for C,
  - Rust limits the developer but it forces to write code in a way that is easier to prove. C gives more flexibility to the developer but the code is then more difficult to prove.

# Project status and future plans

- Implementation status:
  - Prototype running confidential VMs with VirtIO support
- Goal for the next months:
  - Add support for Linux-based VMs
  - Verify the core of ACE’s paging system
- Gradually adding more features to RefinedRust (including lots of usability improvements); long-term: add another backend for speed

# Open Questions

1. How to verify parts implemented in **assembly** and link with Rust verification?
2. How can we prove **security properties** on top of the functional verification?
   - Feasible thanks to Coq's expressive logic!
3. How can we make **unsafe Rust verification more scalable**?
   - RefinedRust's usability is nowhere near more mature provers for safe Rust (like Prusti, Creusot, etc.)

Thank you

Lennard Gäher & Wojciech Ozga