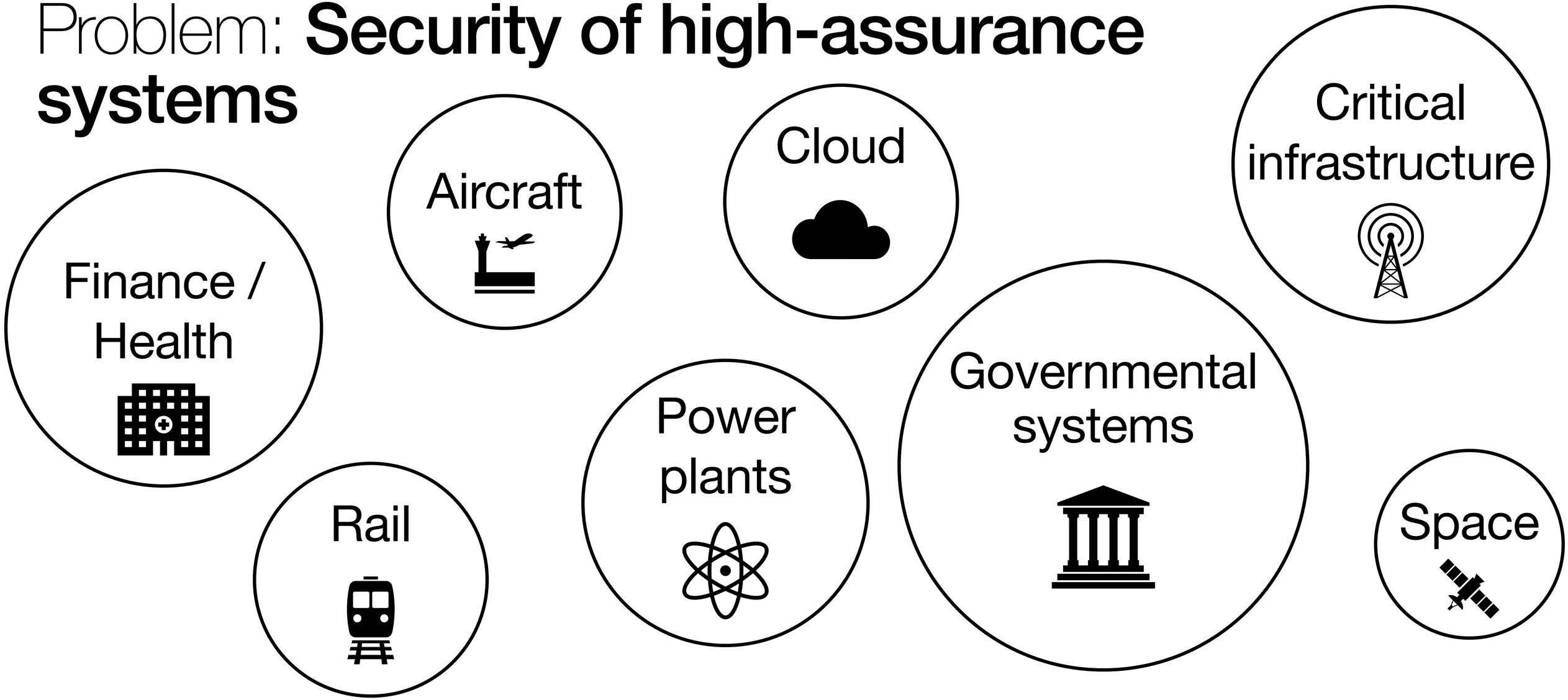
July 23-25, 2024
NIST FMCP 2024

Towards formal verification of the confidential computing framework for RISC-V

Wojciech Ozga	Lennard Gäher	**Guerney D.H. Hunt**	**Avraham Shinnar**	Elaine R. Palmer	Michael V. Le	Silvio Dragone
IBM Research Zürich	*IBM Research Zürich & MPI-SWS, Germany*	*IBM Thomas J. Watson Research Center*	*IBM Thomas J. Watson Research Center*	*IBM Thomas J. Watson Research Center*	*IBM Thomas J. Watson Research Center*	*IBM Research Zürich*

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

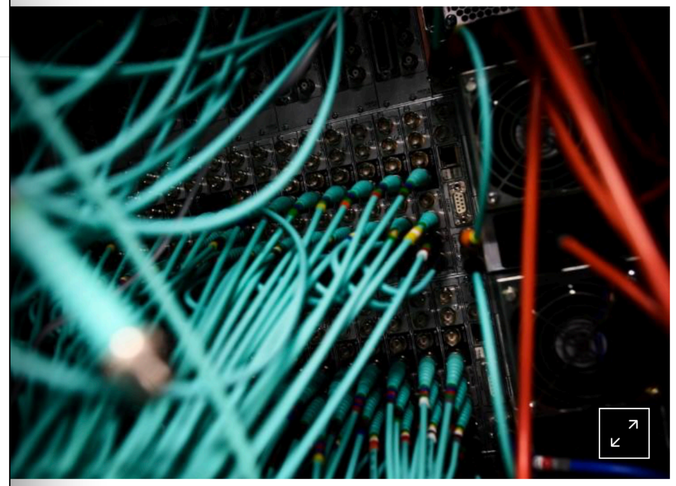
INTERNET NEWS

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

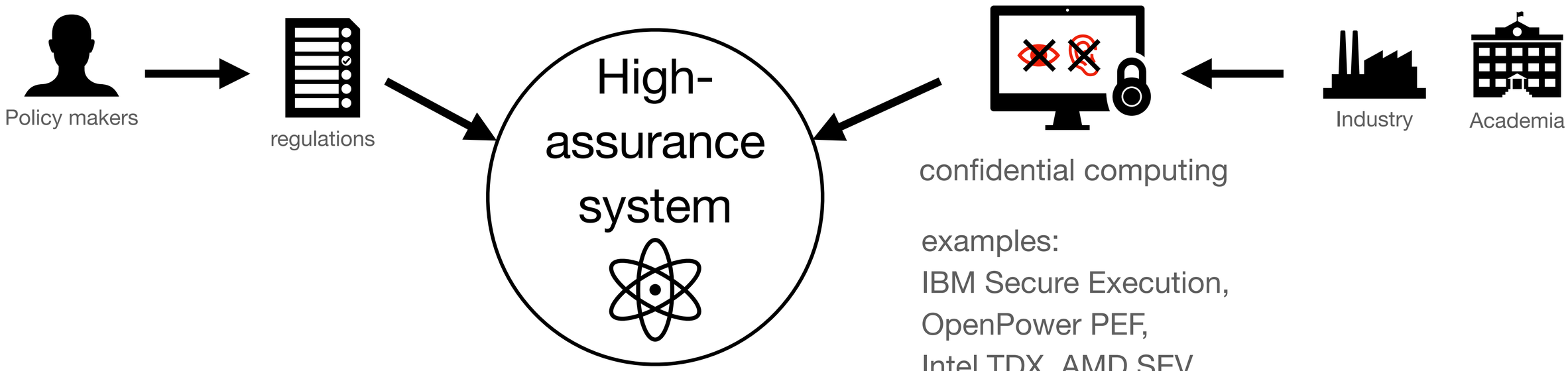
a do Ministério expõe dados pessoais de mais de 200 milhões de brasileiros

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?**



examples:
IBM Secure Execution,
OpenPower PEF,
Intel TDX, AMD SEV,
Intel SGX, Keystone
…

Security-critical systems are subject to **regulations**. Certification requires some form of verification. **Formal methods/verification** is one approach.

Goal: **Build an open-source formally verified confidential computing technology.**

Agenda

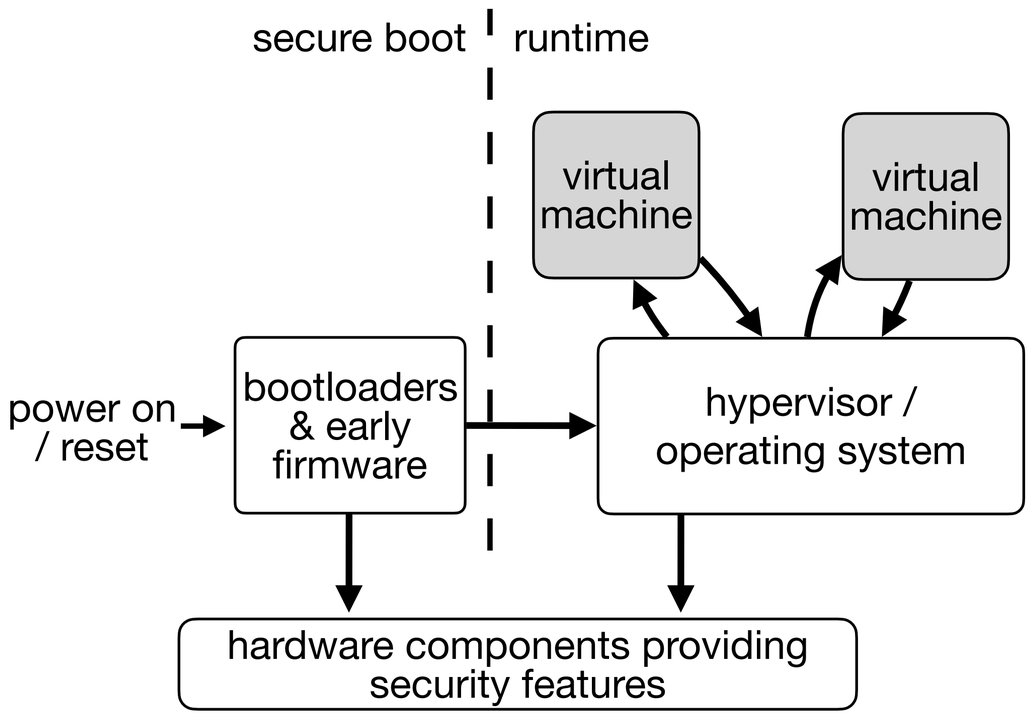
Part I - Confidential Computing Architecture

- Traditional vs confidential computing architecture
- Canonical architecture
- ACE: Implementation for RISC-V

Part II - Formal Verification

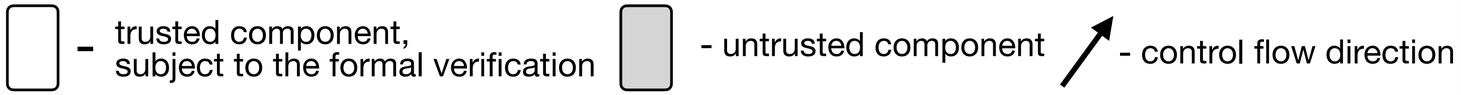
- Methodology & verification approach
- What has to be proven?
- Demo
- Towards proving security properties

Traditional (Non-Confidential) Computing Systems



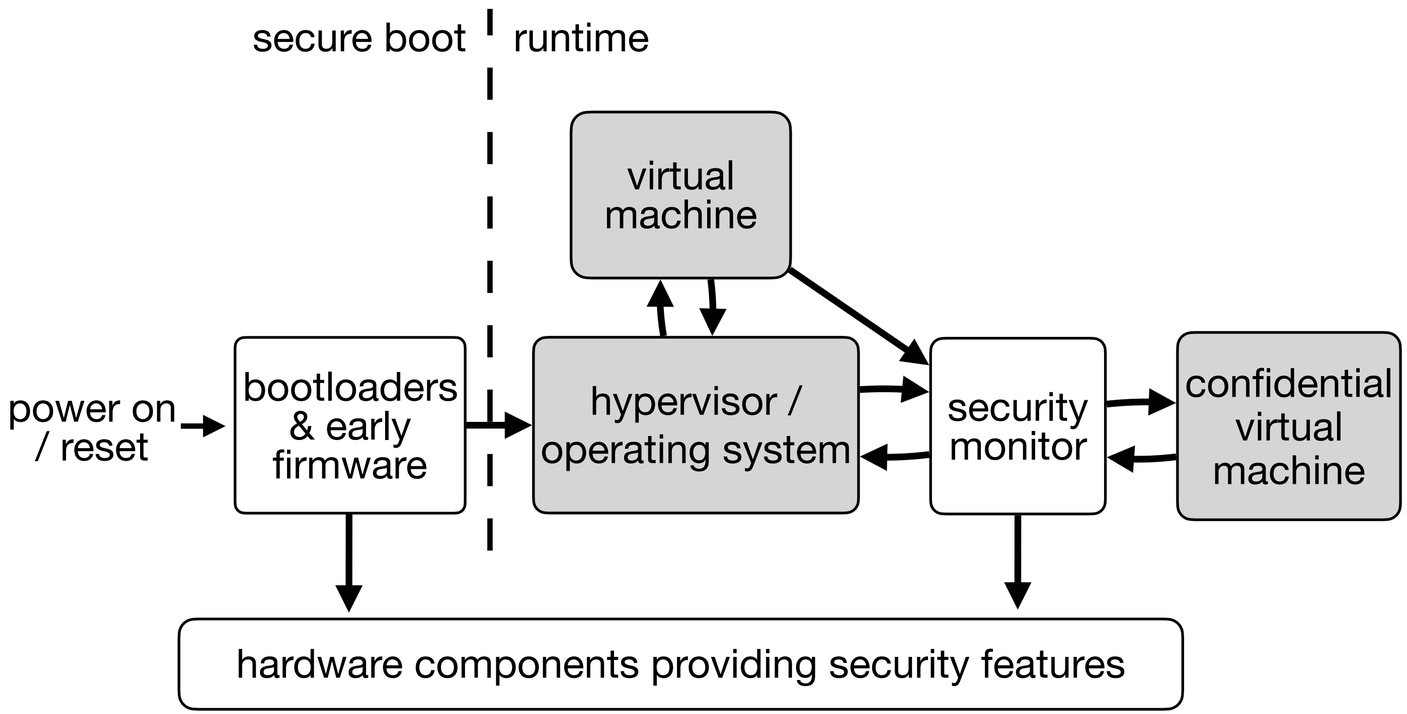
Security guarantees:

- Isolate virtual machines and hypervisor from other virtual machines
- Hypervisor, firmware, drivers, and system administrator are trusted.
- Linux-based hypervisor consists of more than 10 millions lines of code written in unsafe language.



Confidential Computing

is a technology that provides infrastructure to run computations confidentially.



- trusted component, subject to the formal verification
- untrusted component
- control flow direction

Minimal security guarantees:

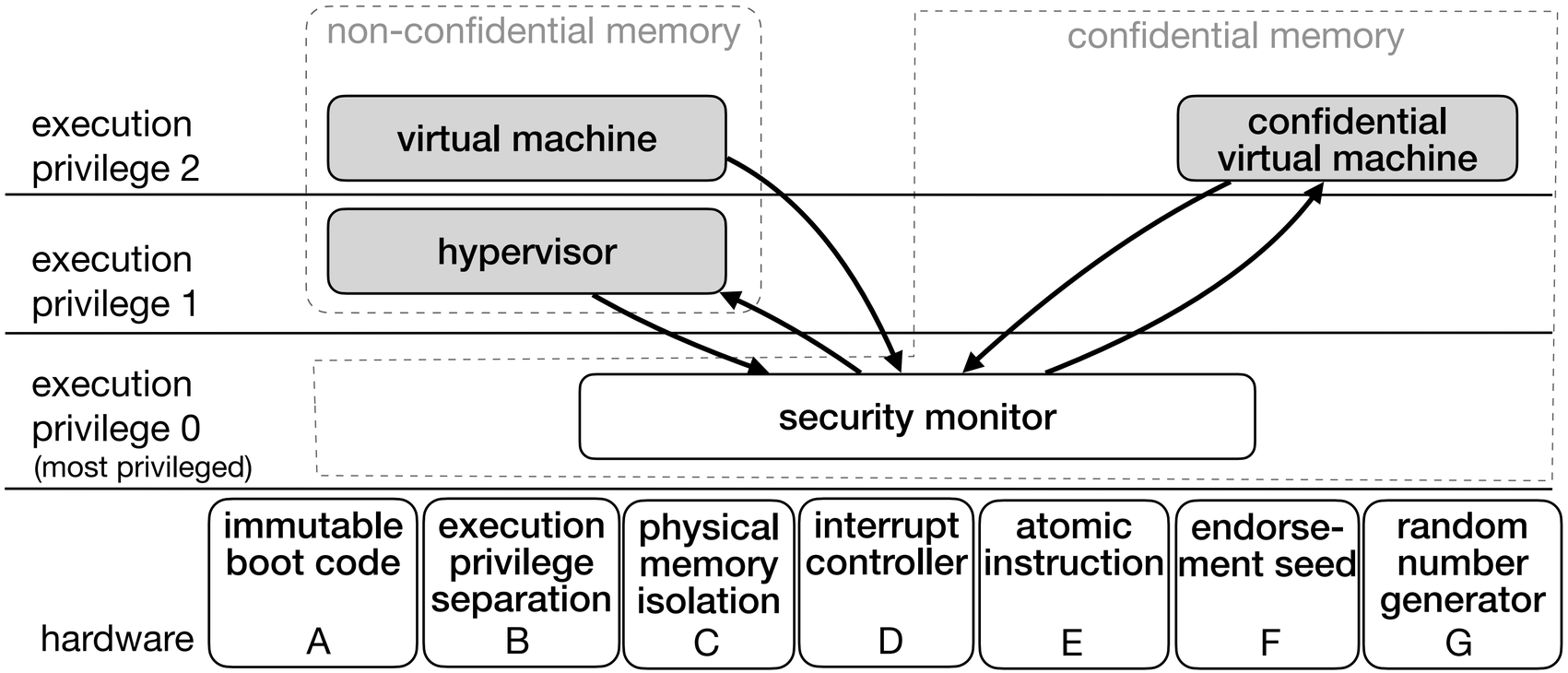
- (Confidentiality), integrity of code and its execution.
- Confidentiality and integrity of data.
- No availability guarantees.
- Guarantees to runtime state (no leaks via architectural state or when information stored in the main memory).

Threat model:

- Software-level adversary controlling hypervisor, other VMs, confidential VMs, peripheral devices except for the protected confidential VM
- Protections against hardware-level adversary include, for example, memory encryption.

Canonical Architecture

Is a (threat model dependent) set of hardware and software components sufficient to build a minimalistic but functional **processor-independent confidential computing architecture**.



Hardware components:

A. **Immutable boot code**
enables integrity- and authenticity-enforced boot of the security monitor.

B. **Execution privilege separation**
enables partitioning software to create, assign, and enforce roles and access control.

C. **Physical memory isolation**
allows isolating memory regions by setting and enforcing memory access control.

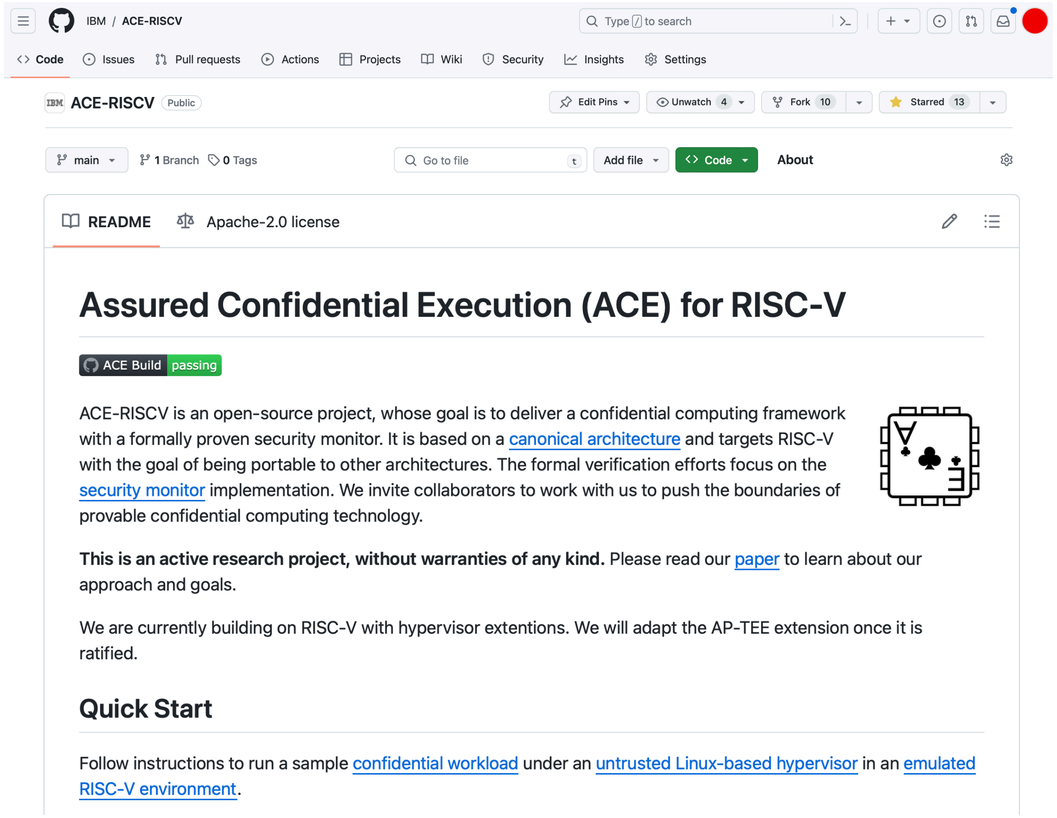
D. **Interrupt controller**
enables signaling and execution flow between execution privileges.

E. **Atomic instruction**
required on multi-core processors to implement synchronization primitives.

F. **Endorsement seed**
required for attestation, used to derive attestation key.

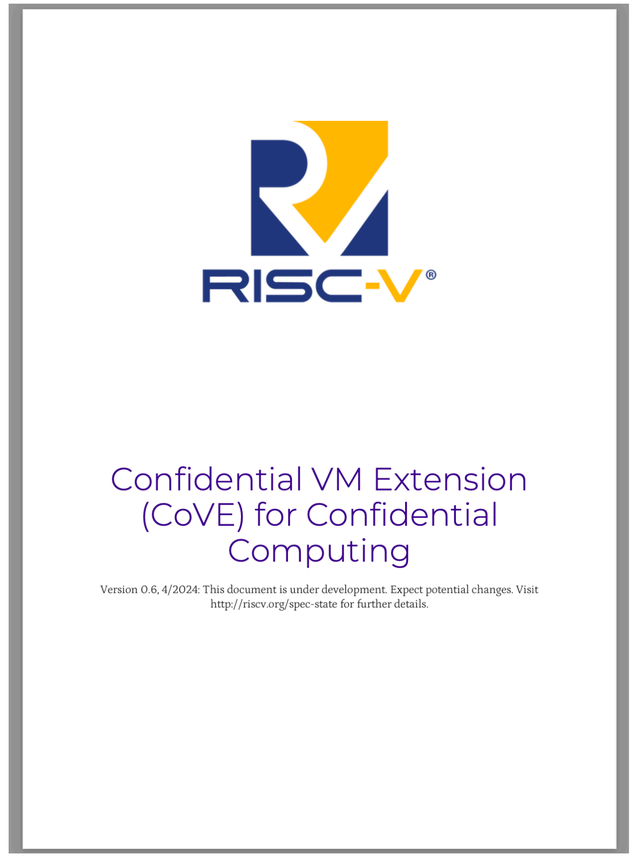
https://github.com/

IBM/ACE-RISCV



https://github.com/

riscv-non-isa/riscv-ap-tee



Agenda

Part I - Confidential Computing Architecture

- Traditional vs confidential computing architecture
- Canonical architecture
- ACE: Implementation for RISC-V

Part II - Formal Verification

- Methodology & verification approach
- What has to be proven?
- Demo
- Towards proving security properties

Where are formal methods used?

CompCert

Formally verified C compiler

HACL*

High assurance cryptographic primitives

seL4

Verified microkernel

IBM HSM

Certified Hardware Security Module

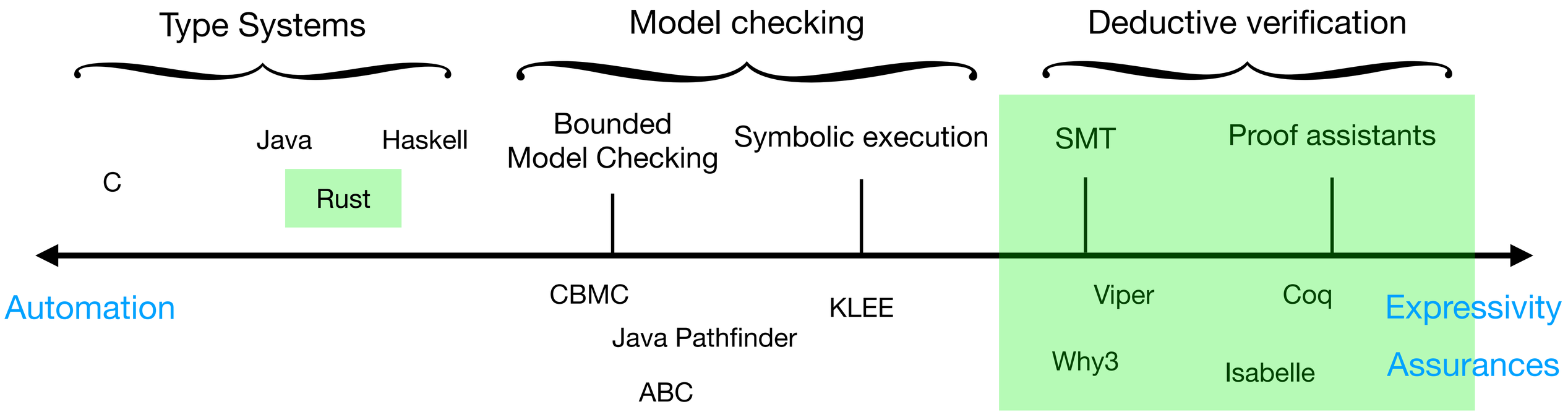
IBM Formal ML

Formalised probability theory

SLAM

Property checks for Windows drivers

Techniques in formal verification (non-exhaustive)



We use a two-pronged approach

The Rust programming language provides safety

Systems programming
with zero-cost abstractions
for memory management

Growing ecosystem and
increasing popularity

Brings modern programming paradigms
to systems programming

Aims to provide memory safety for free(*):

- no null-pointer accesses
- no use-after-free
- no data races
- …

(*) more work if you use unsafe code

Deductive verification using RefinedRust

PLDI'24

Goal: verify memory safety (of unsafe code) & functional correctness

Automatic translation
Rust ⇒ Radium

Proof automation
guiding application of typing rules

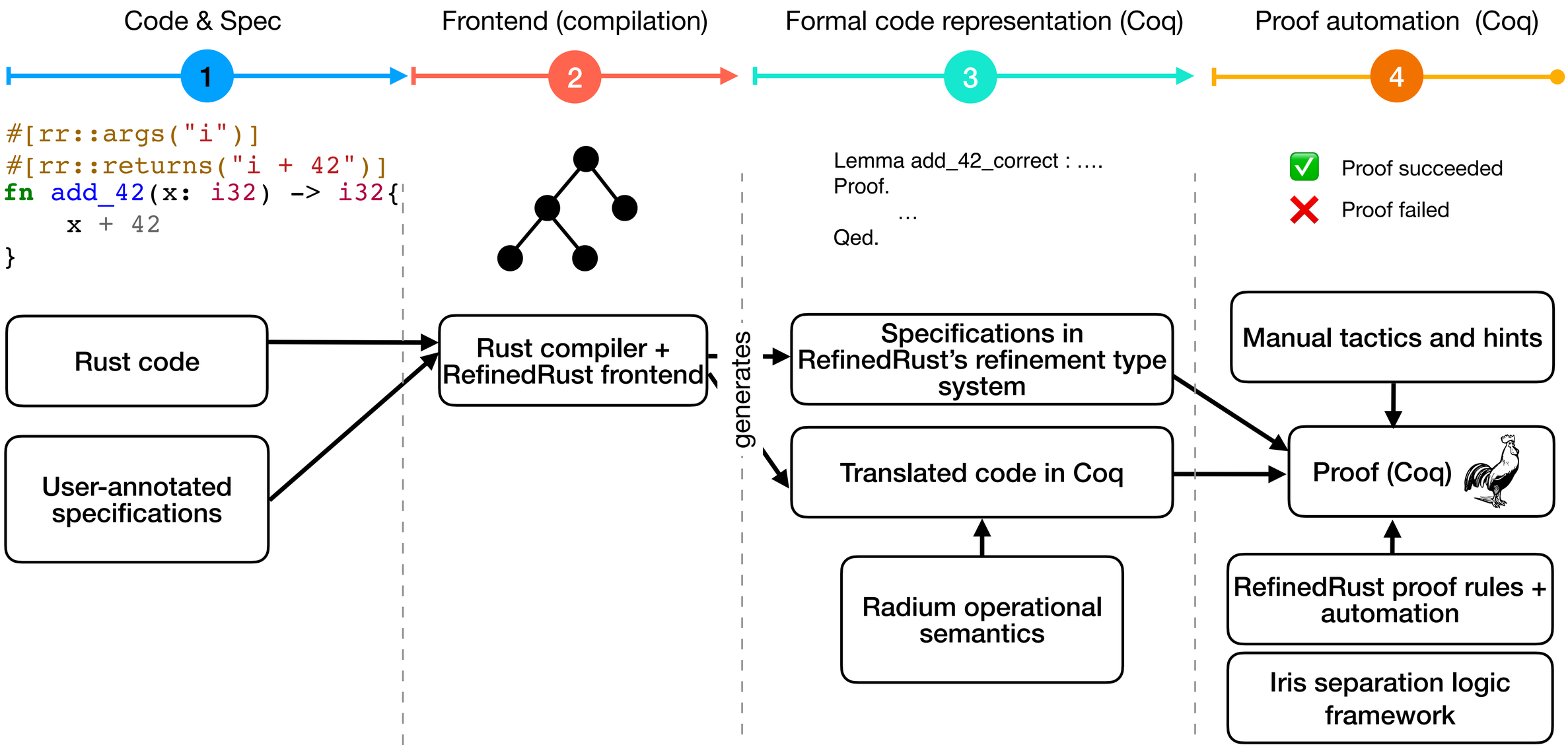
Formal model of Rust:
Radium operational semantics

Refinement type system
with semantic soundness proof

Coq proof assistant 

RefinedRust: A Type System for High-Assurance Verification of Rust Programs. PLDI'24.
Lennard Gäher, Michael Sammler, Ralf Jung, Robbert Krebbers, and Derek Dreyer.

Architecture of RefinedRust



Designed to be run continuously in CI

How practical is RefinedRust?

- Verified Vec library
- Published paper

Verified parts of
security monitor

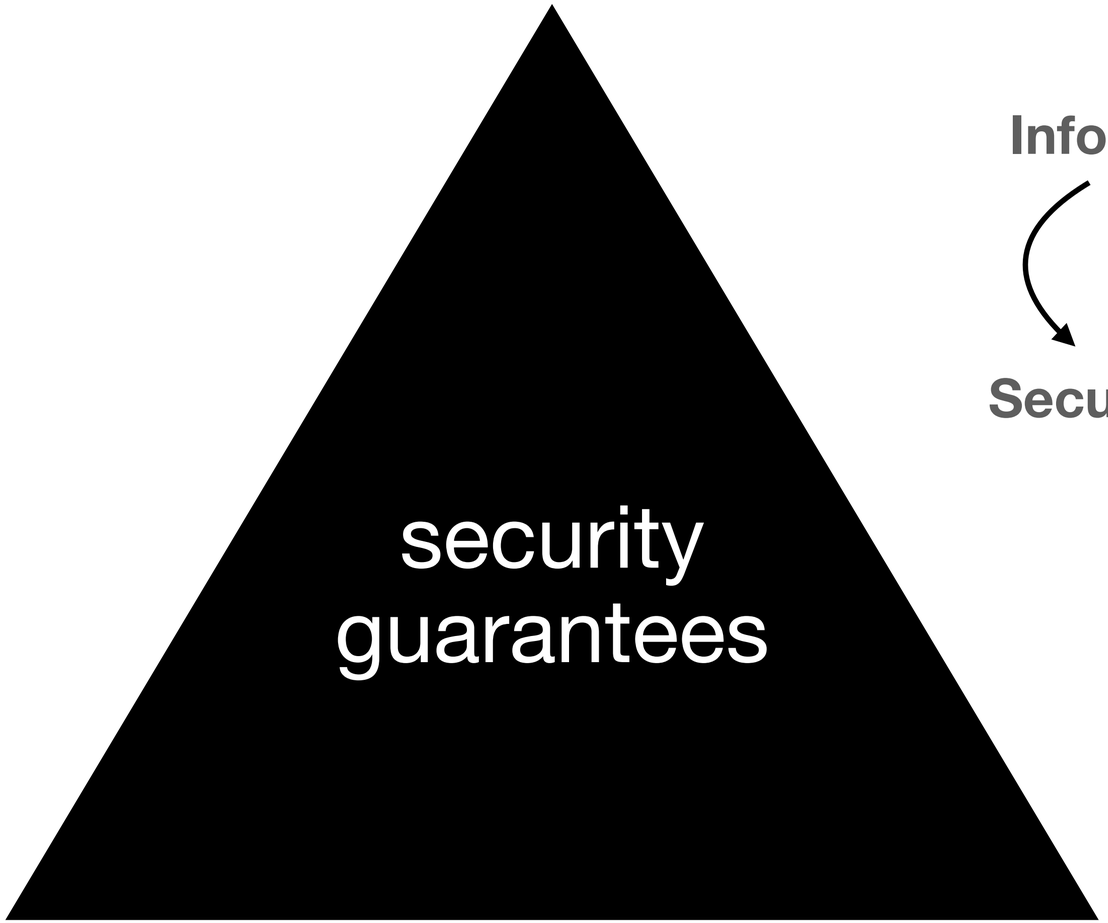
- Supports traits (≈ Rust interfaces)
- Supports closures (≈ Rust anonymous functions)
- Supports multi-module crates
- Runs in continuous integration (CI)
- Supports axiomatising Rust standard library
- Supports mixed manual-automatic proofs
- Supports Iris ghost state
- Supports statics, char, …
- Improved performance & documentation
- …. much more!

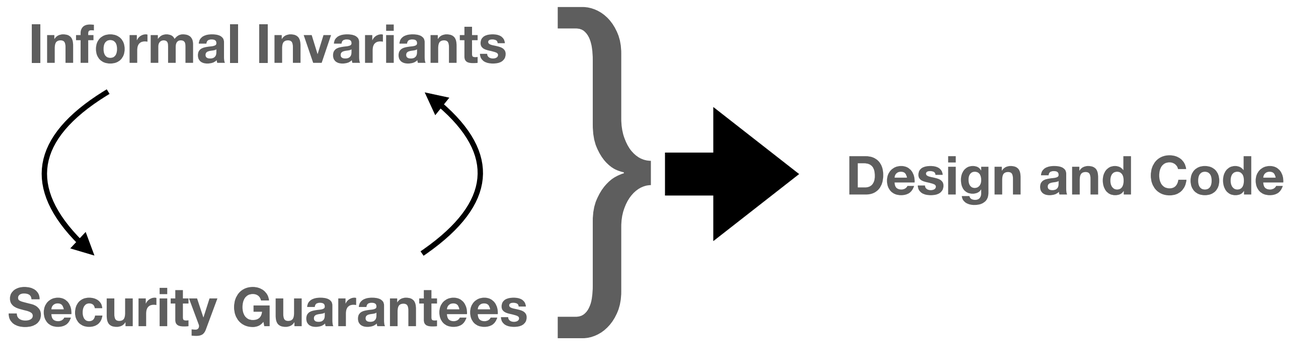
RefinedRust vs other Rust verification approaches

	Tech	Unsafe code	Traits & closures	Verified proofs	Automated	User-friendly	Extensible	Concurrency Aware
RefinedRust	Coq	✅	✅	✅	~	~	✅	✅
Prusti	SMT	Support varies and is evolving			✅	✅	Support varies and is evolving	
Creusot	SMT				✅	✅		
Aeneas	Mixed							
Verus	SMT				✅	✅		
Kani	Model Checking				✅	✅		

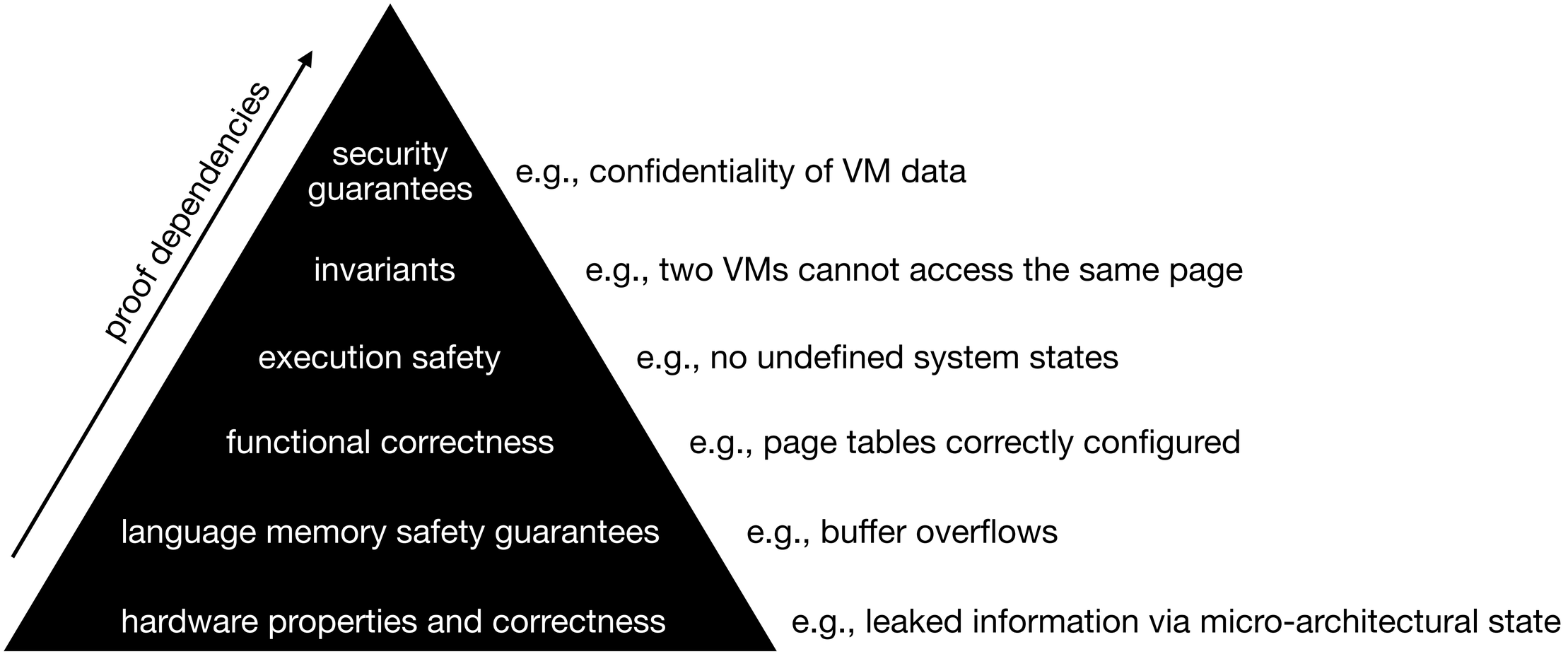
Future Work: combine advantages of different systems

ACE: What has to be proven?

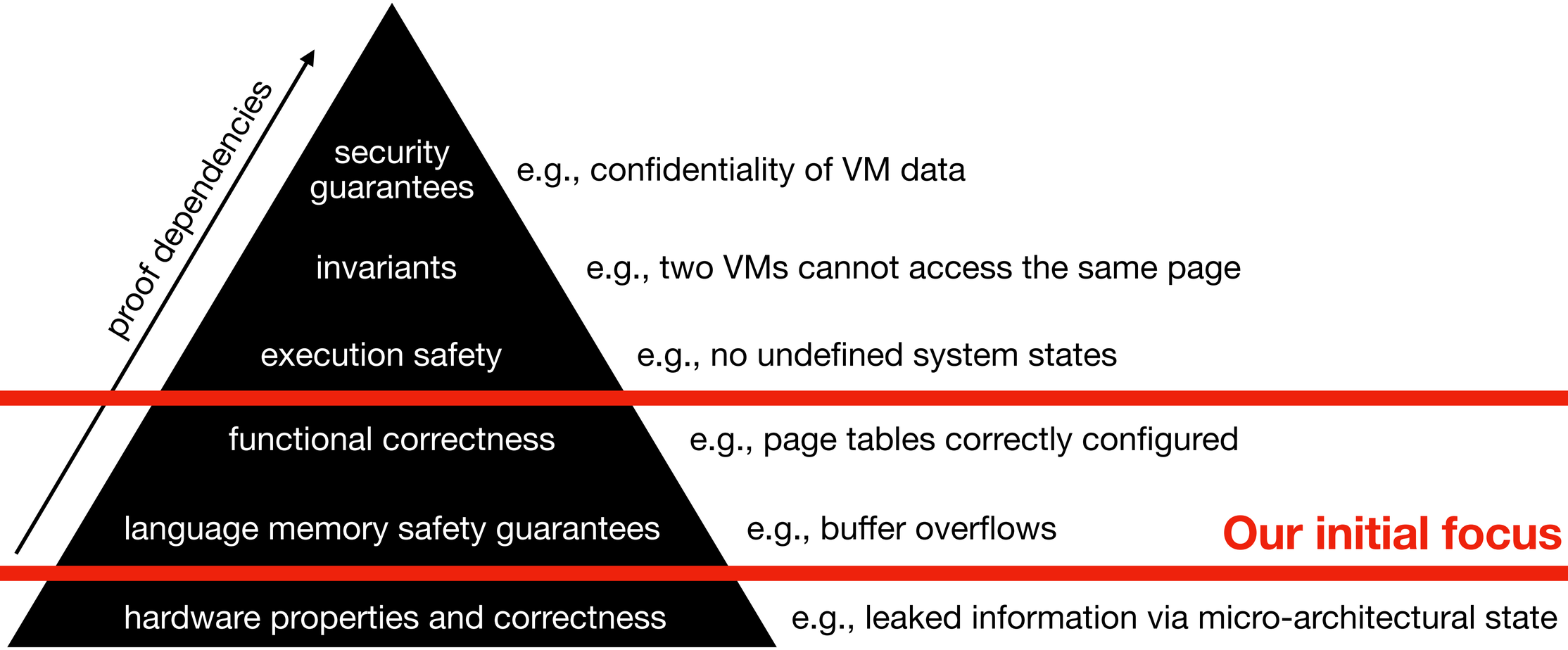




ACE: What has to be proven?

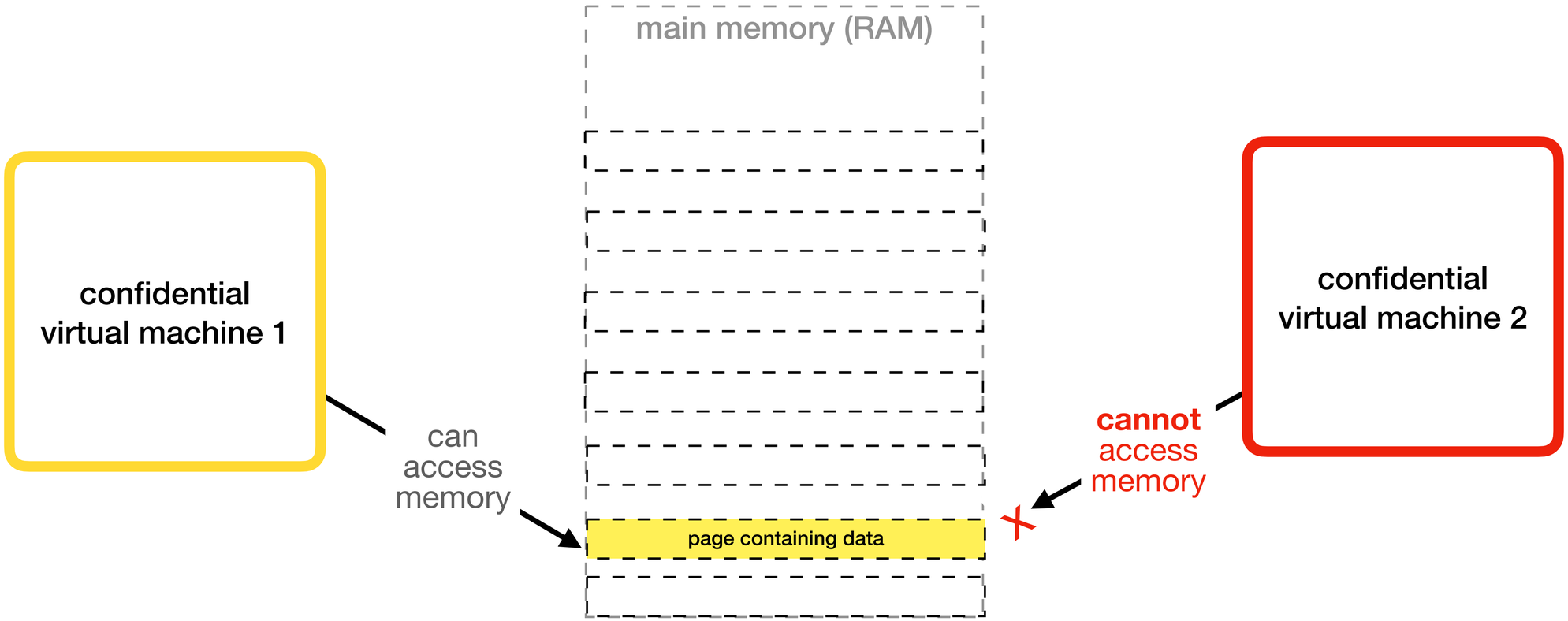


ACE: What has to be proven?



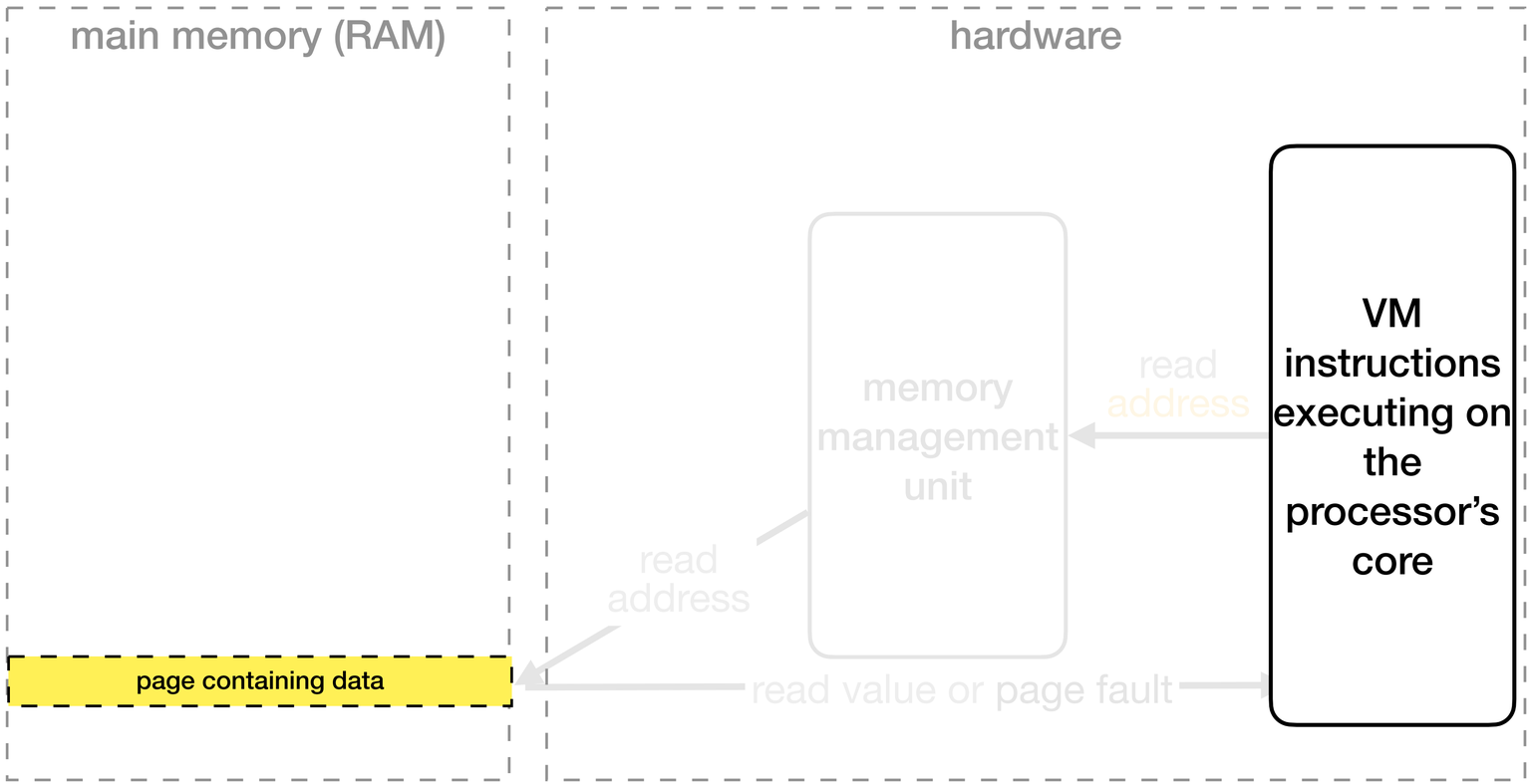
Example: **Memory Allocation**

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



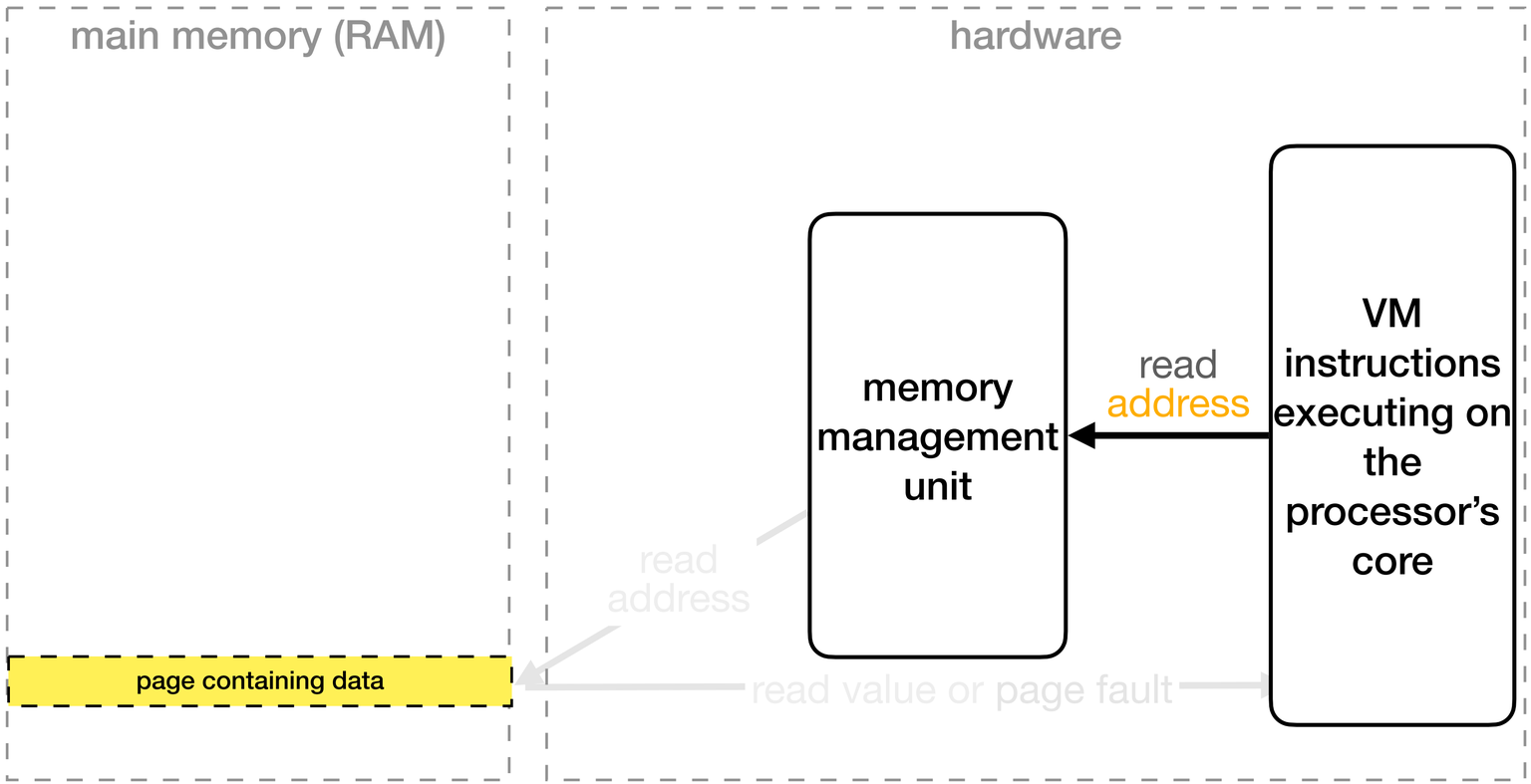
Example: **Memory Allocation**

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



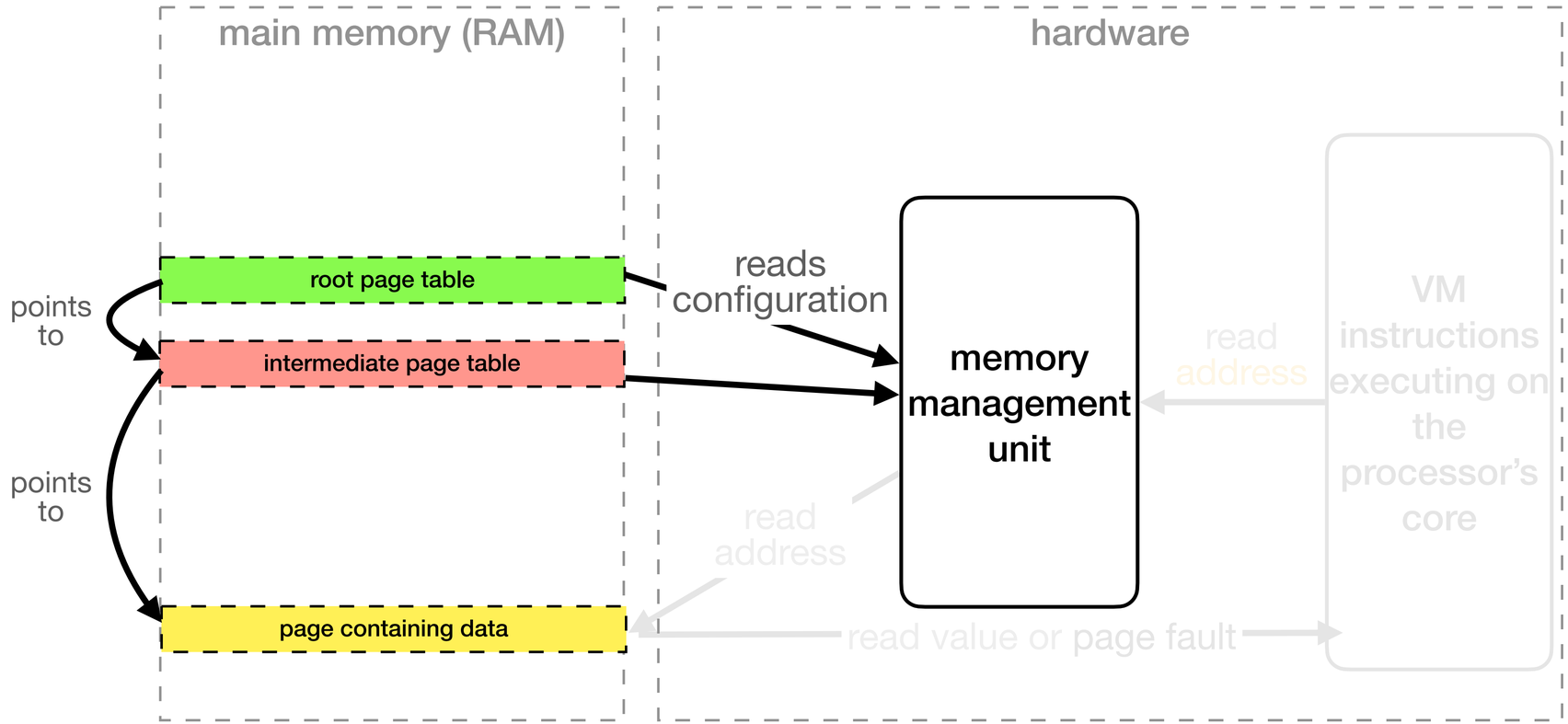
Example: **Memory Allocation**

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



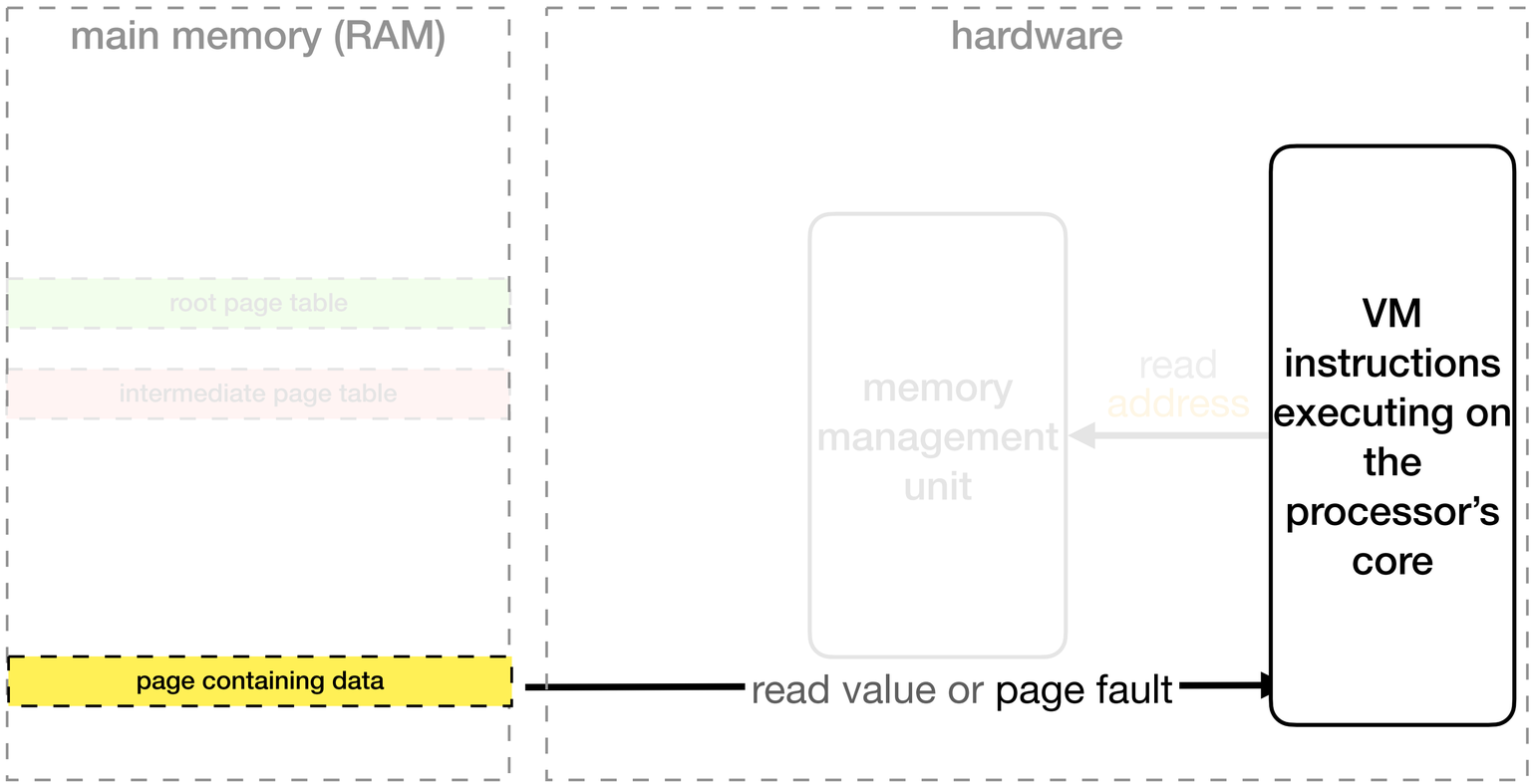
Example: **Memory Allocation**

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



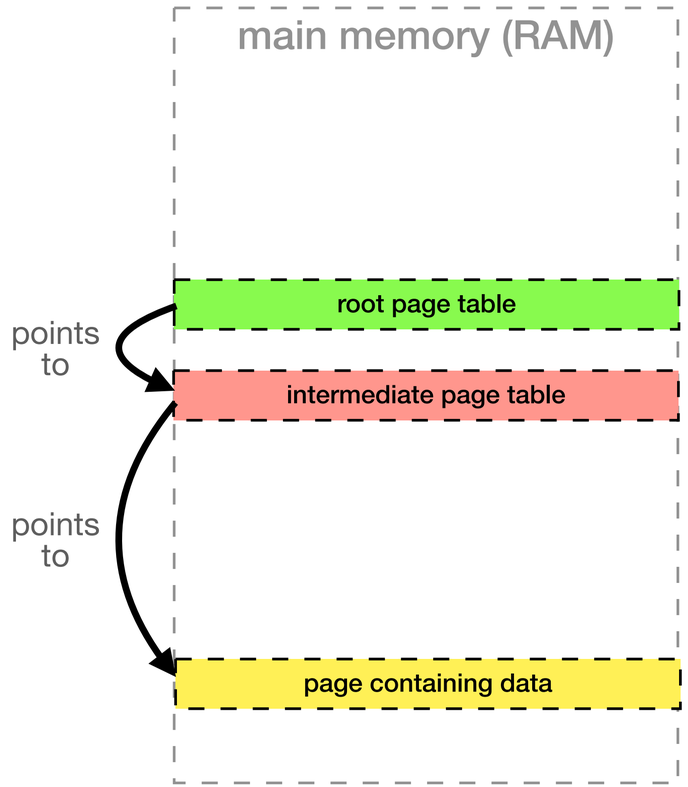
Example: **Memory Allocation**

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.

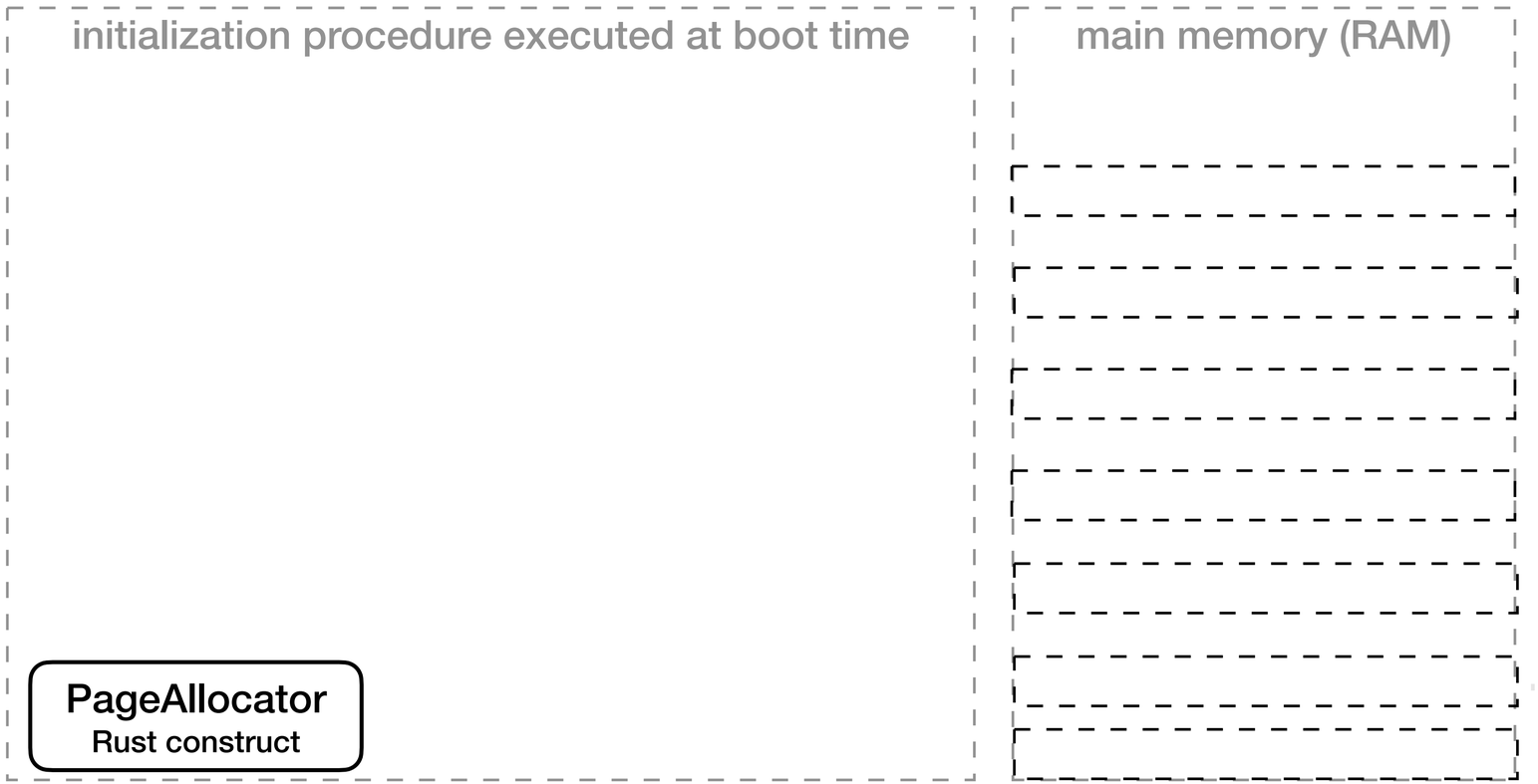


We must formally verify the functional correctness of the page table configuration.

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

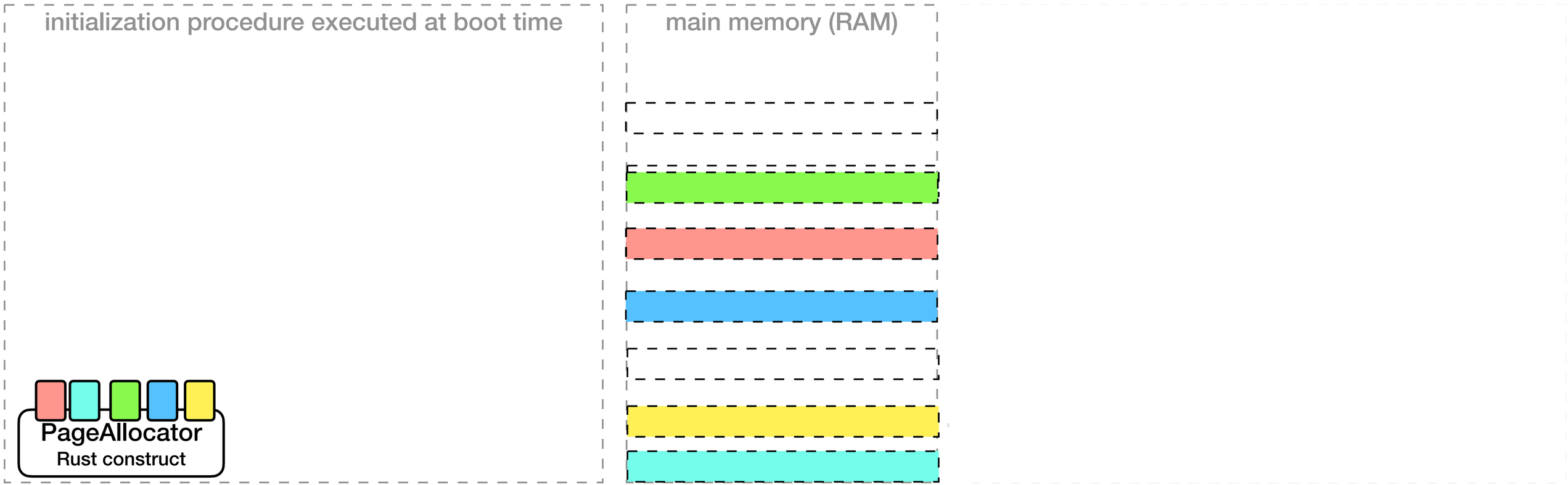
Example: **Memory Allocation**

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



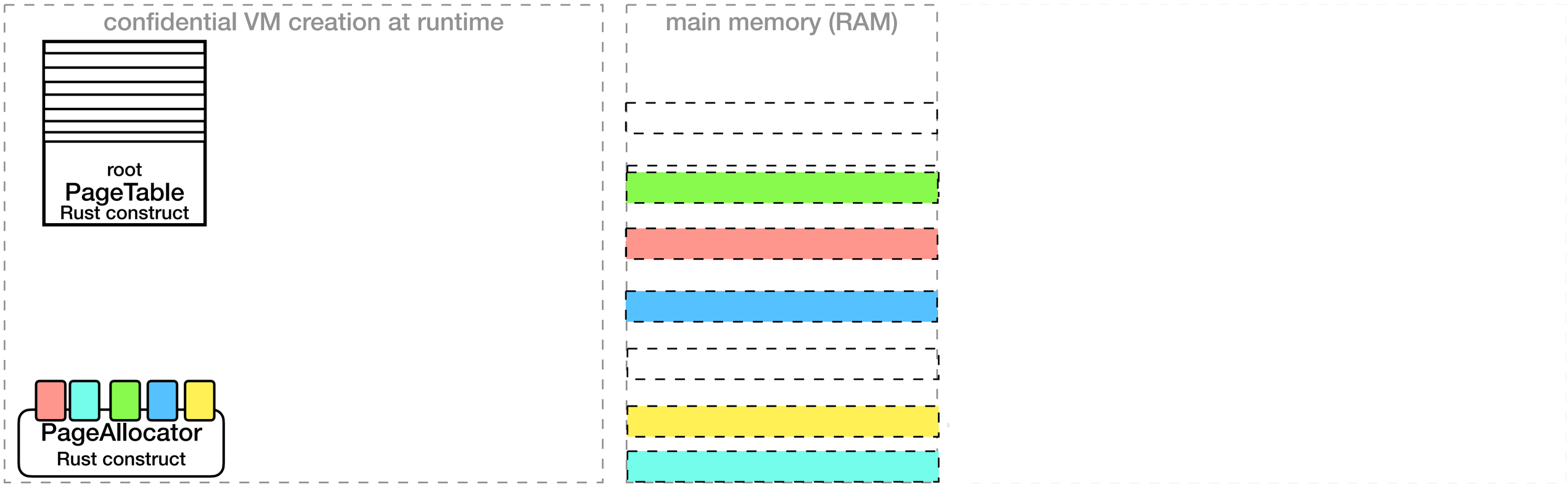
Example: **Memory Allocation**

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



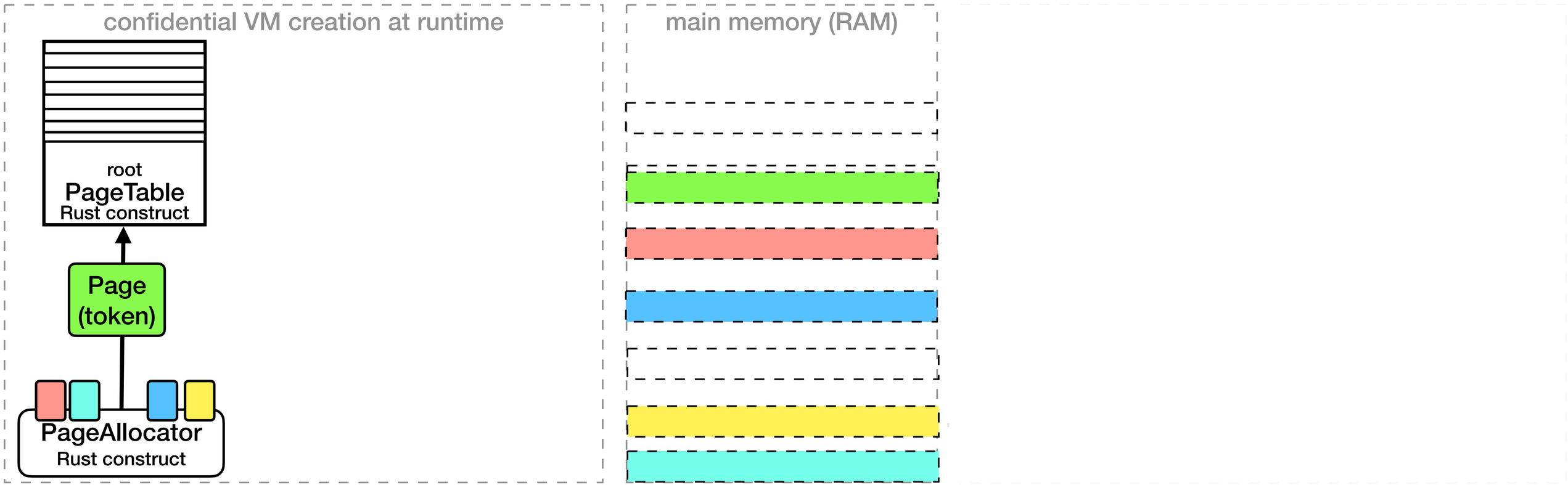
Example: **Memory Allocation**

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



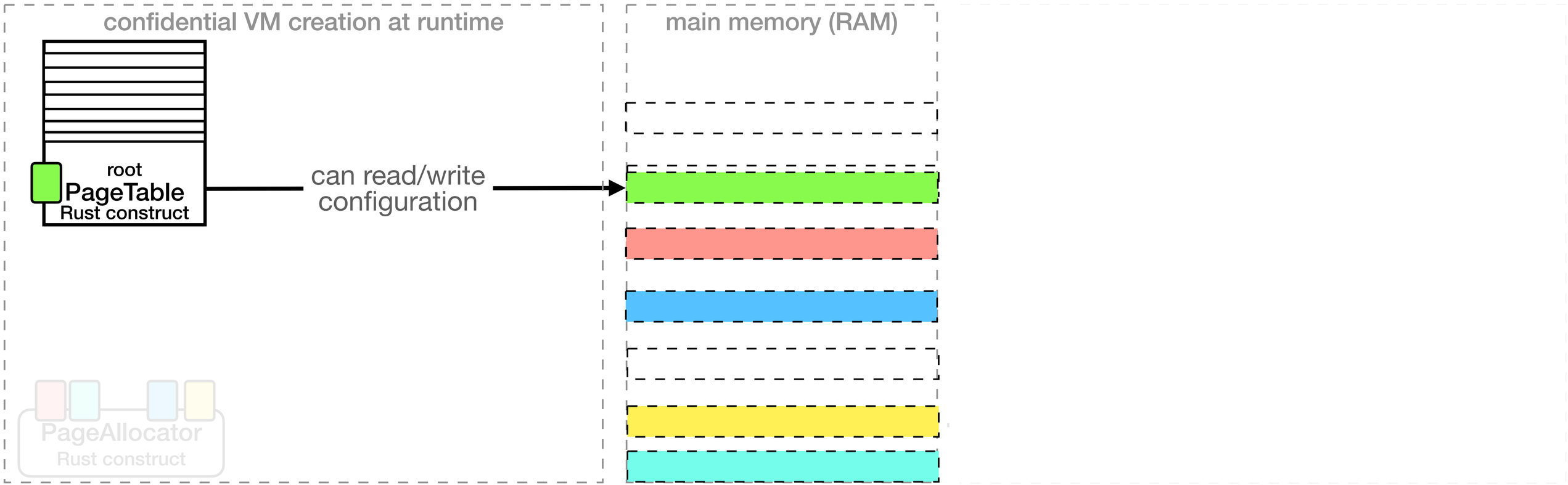
Example: **Memory Allocation**

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



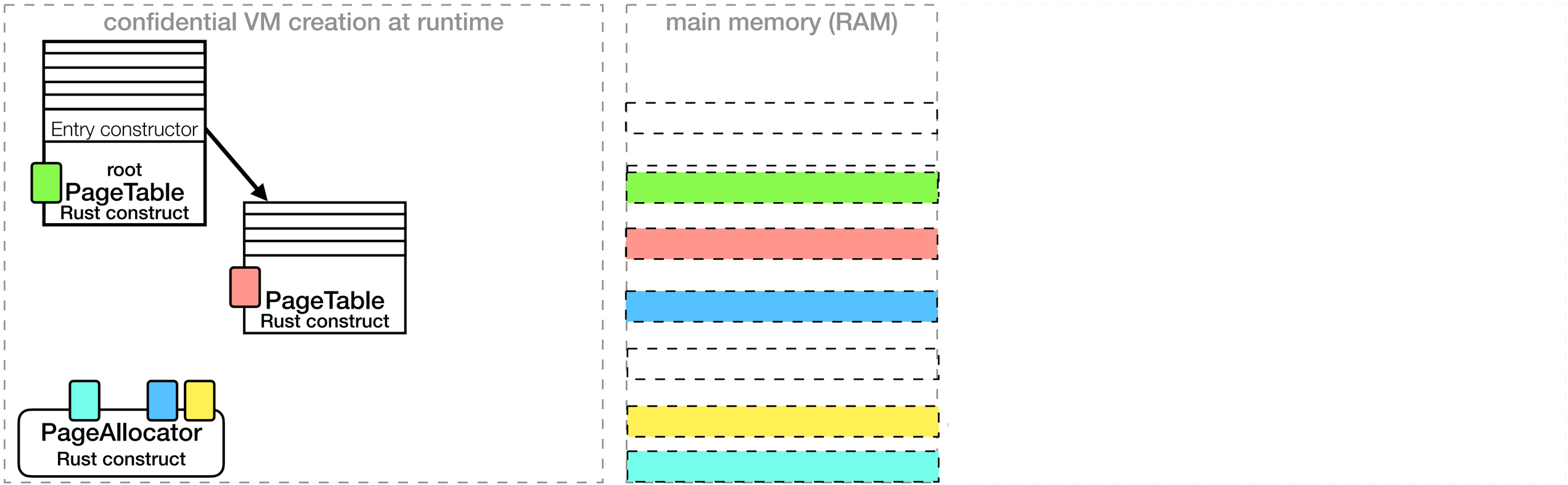
Example: **Memory Allocation**

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



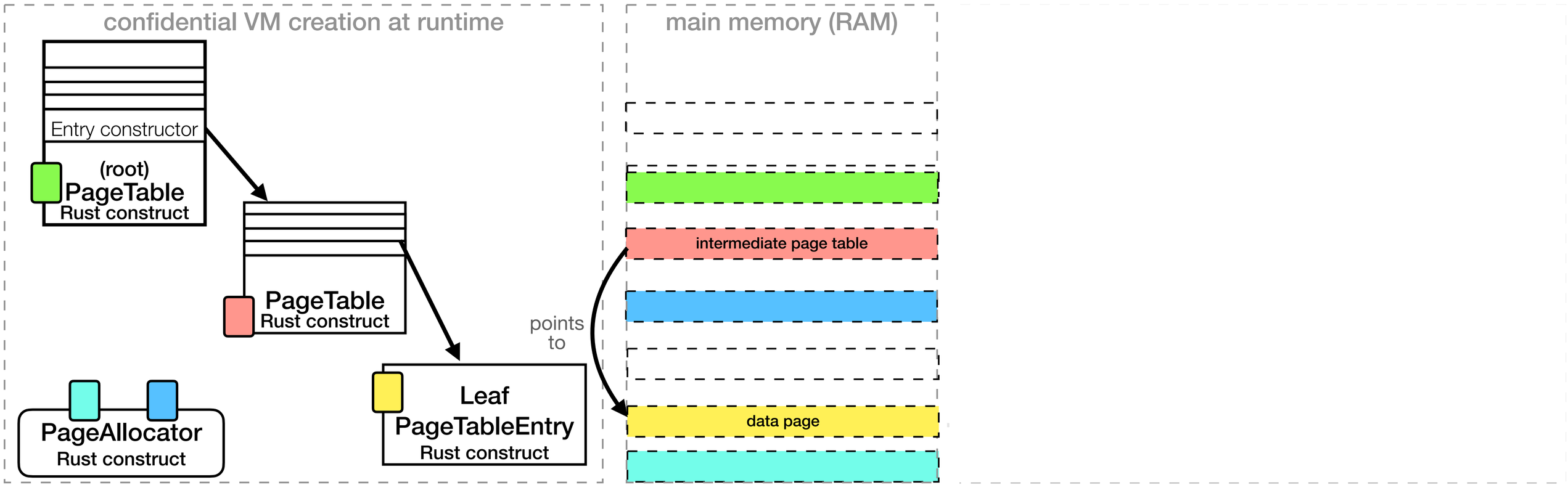
Example: **Memory Allocation**

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.



Page: Rust

2 implementations
```rust
pub trait PageState {}
```

1 implementation
```rust
pub struct UnAllocated {}
```
1 implementation
```rust
pub struct Allocated {}
```

```rust
impl PageState for UnAllocated {}
impl PageState for Allocated {}
```

5 implementations
```rust
pub struct Page<S: PageState> {
    address: ConfidentialMemoryAddress,
    size: PageSize,
    _marker: PhantomData<S>,
}
```

Page: RefinedRust

```
#[rr::refined_by("p" : "page")]
#[rr::invariant(#type "p.(page_loc)" : "<#> p.(page_val)"
                    @ "array_t (int usize_t)
                        (page_size_in_words_nat p.(page_sz))")]

#[rr::invariant("page_wf p")]
#[rr::context("onceG Σ memory_layout")]
#[rr::exists("MEMORY_CONFIG")]
#[rr::invariant(#iris "once_status \"MEMORY_LAYOUT\" (Some MEMORY_CONFIG)"

#[rr::invariant("MEMORY_CONFIG.(conf_start).2 ≤ p.(page_loc).2")]
#[rr::invariant("p.(page_loc).2 + (page_size_in_bytes_nat p.(page_sz))
                    < MEMORY_CONFIG.(conf_end).2")]
```

6 implementations

```
pub struct Page<S: PageState> {
```

Page: fields in RefinedRust

```rust
pub struct Page<S: PageState> {
    /// Specification: the `address` has mathematical value `l`.
    #[rr::field("p.(page_loc)")]
    address: ConfidentialMemoryAddress,
    /// Specification: the `size` has mathematical value `sz`.
    #[rr::field("p.(page_sz)")]
    size: PageSize,
    /// Specification: the `_marker` has no relevance for the verification.
    #[rr::field("tt")]
    _marker: PhantomData<S>,
}
```

Page read: Rust

```rust
impl<T: PageState> Page<T> {
```

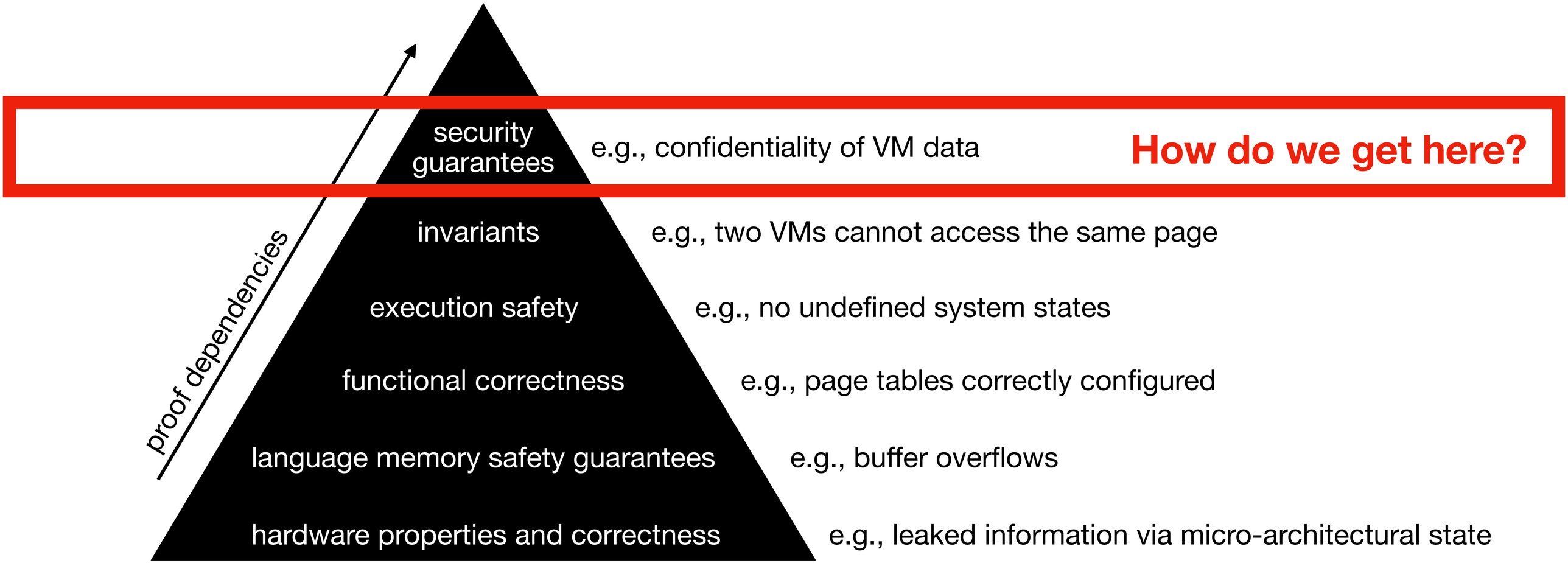
```rust
pub fn read(&self, offset_in_bytes: usize) -> Result<usize, Error> {
    assert!(offset_in_bytes % Self::ENTRY_SIZE == 0);
    let data: usize = unsafe {
        // Safety: below add results in a valid confidential memory address
        // because we ensure that it is within the page boundary and
        // page is guaranteed to be entirely inside the confidential memory.
        let pointer: ConfidentialMemoryAddress =
            self.address.add(offset_in_bytes,
                             upper_bound: self.end_address_ptr()).unwrap();
        // pointer is guaranteed to be in
        // the range <0;self.size()-size_of::(usize)>

        pointer.read_volatile()
    };
    Ok(data)
}
```

Page read: RefinedRust

```
impl<T: PageState> Page<T> {
```

```
#[rr::params("p", "off")]
#[rr::args("#p", "off")]
/// Precondition: the offset needs to be divisible by the size of usize.
#[rr::requires("H_off" : "(ly_size usize_t | off)%Z")]
/// Precondition: we need to be able to fit a usize at the offset
/// and not exceed the page bounds
#[rr::requires("H_sz" : "(off + ly_size usize_t ≤ page_size_in_bytes_Z p.(page_sz))%Z")]
/// Postcondition: there exists some value `x`...
#[rr::exists("x" : "Z", "off'" : "nat")]
/// ...where off is a multiple of usize
#[rr::ensures("(off = off' * ly_size usize_t)%Z")]
/// ...that has been read from the byte sequence `v` at offset `off`
#[rr::ensures("p.(page_val) !! off' = Some x")]
/// ...and we return an Ok containing the value `x`
#[rr::returns("Ok(#x)")]
pub fn read(&self, offset_in_bytes: usize) -> Result<usize, Error> {
```

ACE: What has to be proven?



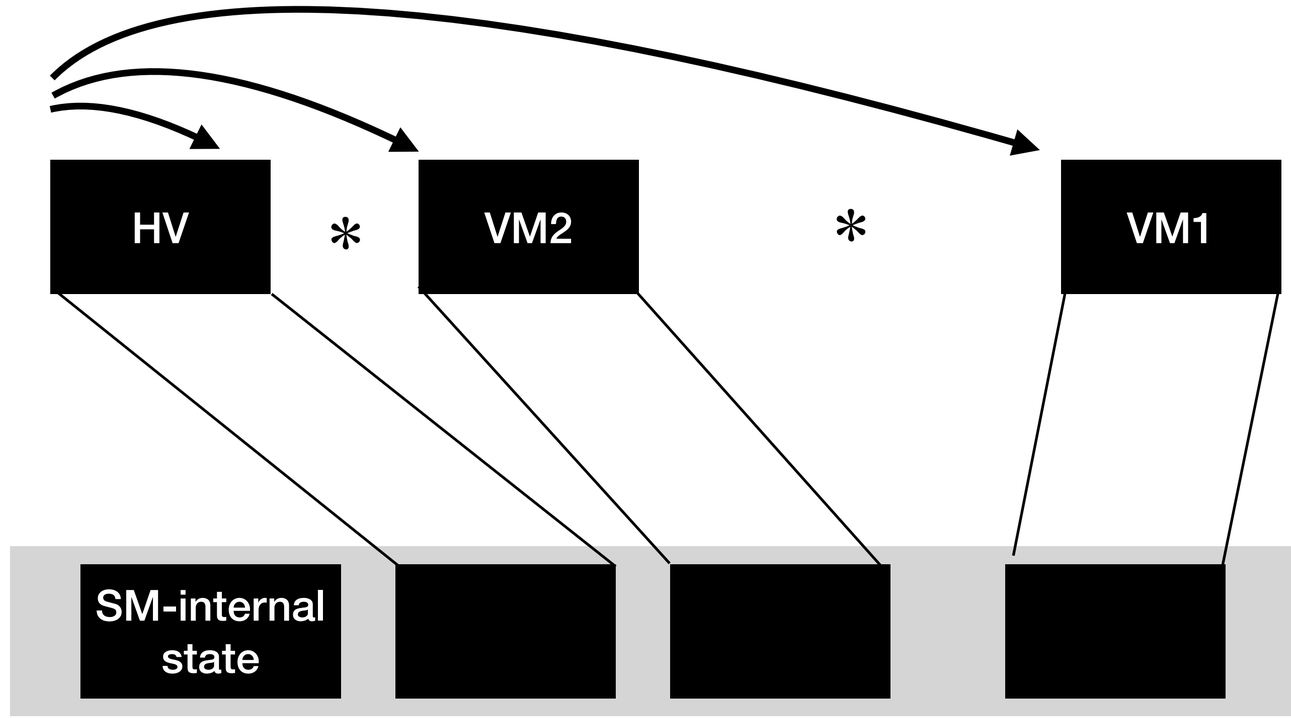
Towards security properties

- Goal: prove *non-interference (wrt. memory; not timing etc.)*
  - No secrets from confidential VMs are leaked
- In a realistic system: *relaxed* non-interference
  - Confidential VMs can selectively declassify information
  - e.g. by requesting to share a page

For now: focus on strict non-interference

Part I: Isolating memory regions

HW memory isolation
set up by
security monitor!



virtual memory view
of VMs

physical memory view
of security monitor

If we prove the page table setup correct, we know that
any process (a VM or the hypervisor) cannot access another process memory.

Part II: Proving non-interference for the security monitor

- The security monitor has access to the full physical memory
  - We cannot prove that it is physically isolated!
- Security monitor could open side channels:
  - Read memory of one VM and behave differently depending on the value

How to prove non-interference for the security monitor?

How do we prove non-interference?

- Typically: proved by relating two executions

$\forall s_1, s_1', s_2, s_2', i$. related $p_i\ s_1\ s_2 \rightarrow$ exec $p_i\ s_1\ s_1' \rightarrow$ exec $p_i\ s_2\ s_2' \rightarrow$ related $p_i\ s_1'\ s_2'$

where related states that the two states are equivalent on p_i's memory

- But our main verification of the security monitor (in RefinedRust) reasons about only one execution!

Standard trick: **information flow tracking**
Future Work: add information flow tracking to **RefinedRust**

Ongoing & Future Work

- Informal specification of invariants and security properties
- Work with RISC-V community on CoVE (AP-TEE) spec

- Canonical architecture design
- ACE prototype implementation in Rust

- Implementation of ACE for RISC-V with support for Linux VMs
- Formal specification of memory isolation subsystem
- Formally verified page tokens and partial proof for page allocator

- Formally verified hardware memory isolation,
- ACE compliance with RISC-V CoVE spec.
- Contribute to Linux kernel CoVE patches.

- Proving security properties (non-interference) using RefinedRust,
- Defining formal spec for a sample declassifier

Summary

- We developed a canonical architecture for confidential computing.
- We are implementing an open source security monitor based on the canonical architecture mapped to RISC-V called ACE.
- We defined the formal specification for the memory isolation subsystem.
- We have formally verified part of the memory isolation subsystem in RefinedRust.
- Developed viable approach to end-to-end verification of a system.

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

Thank you

Back up slides

Outcome

- Talk at CSF'23: "New Software Abstractions for Hardware Security Technology", Ascona, 1-4 October, 2023
- Paper at HASP@MICRO’23: “Towards a Formally Verified Security Monitor for VM-based Confidential Computing”
- Talk at Swiss Verification Day 2024, Neuchatel, 10 January, 2024
- RISC-V CoVE spec includes our architecture and use case for formally verified confidential computing (ratification in progress)
- Open source implementation available at GitHub (Rust + formal spec and verification) - work in progress
- RefinedRust: A Type System for High-Assurance Verification of Rust Programs. PLDI'24. Lennard Gäher, Michael Sammler, Ralf Jung, Robbert Krebbers, and Derek Dreyer