Proving Capability Safety in the Presence of Indirect Sentries

Bastien Rousseau, Aina-Linn Georges, Jean Pichon-Pharabod, Dominique Devriese, Lars Birkedal

POCL 2024

- Capabilities

  → enable mutually distrustful programs to work together

- Functions calls are challenging

  → design
  → formally reason about

- "Encapsulated closures" at assembly level

Closures using Capabilities (1/2)

- Sealed pairs

  ⊕ expressive
  ⊕ follow principle of least privilege
  ⊖ hard to formally reason about
  ⊖ number of otypes (CHERIoT)

- Sentry (or Entry)

  ⊕ mechanized proof of capability safety [Georges et al.]
  ⊖ trampoline code
  → breaks principle of least privilege

Closures using Capabilities (2/2)

- Indirect Sentries

  ⊙ closures with fixed entry point and context
  ⊕ no trampoline code
  → enables principle of least privilege

In this work:

formal study of indirect sentries

Our Contributions

- Cerise (mini-CHERI) with Indirect Sentries

  → validate expected reasoning principles
  → validate expected security guarantees
  → capability safety (stronger than capability monotonicity)

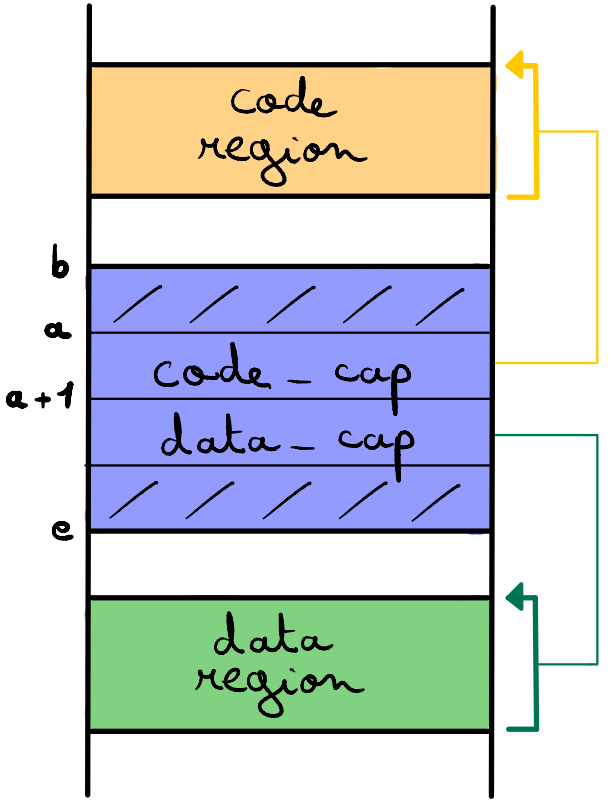
- Prove robust safety of a secure calling convention

  → no trampoline code
  → machine-checked proofs in Coq 

What are Indirect Sentries ?

points-to-pair

(IE, b, e, a)

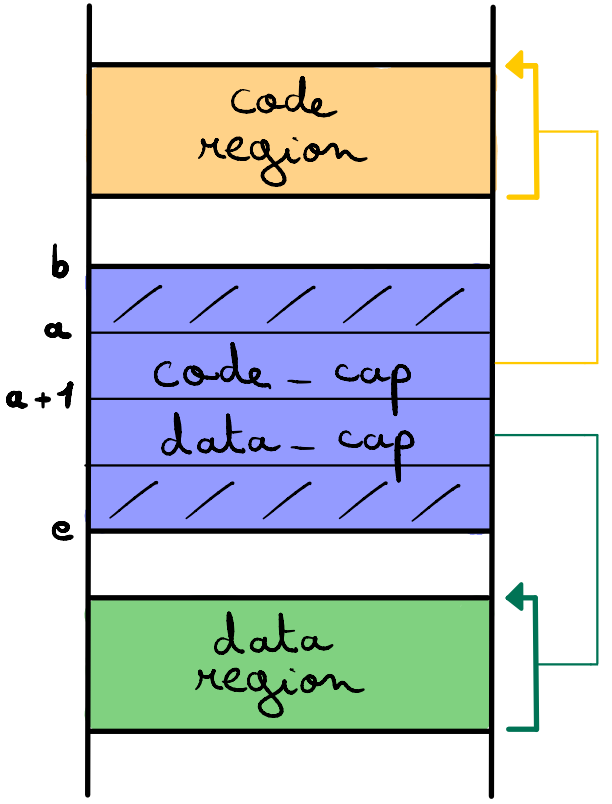


What are Indirect Sentries ?

points-to-pair

(IE, b, e, a)

Indirect Sentries permission

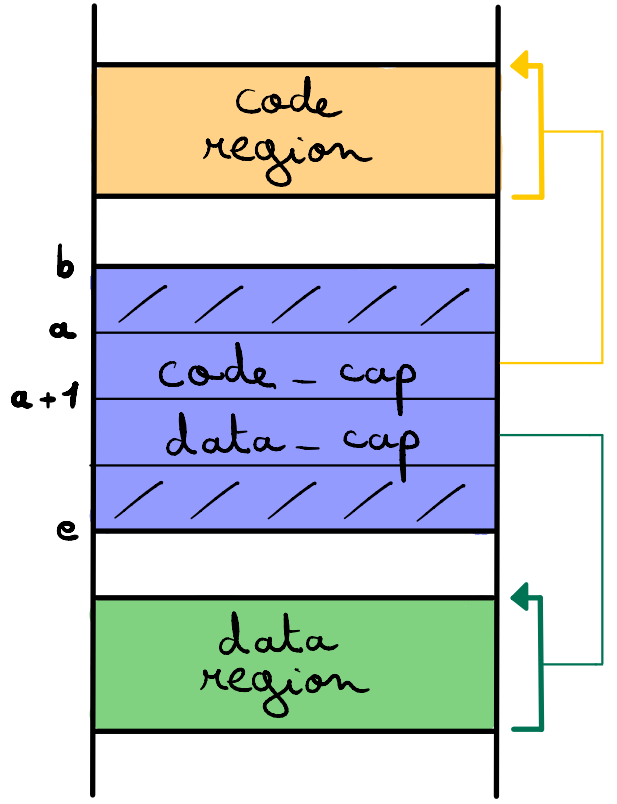


What are Indirect Sentries ?

points-to-pair

bounds

(IE, b, e, a)

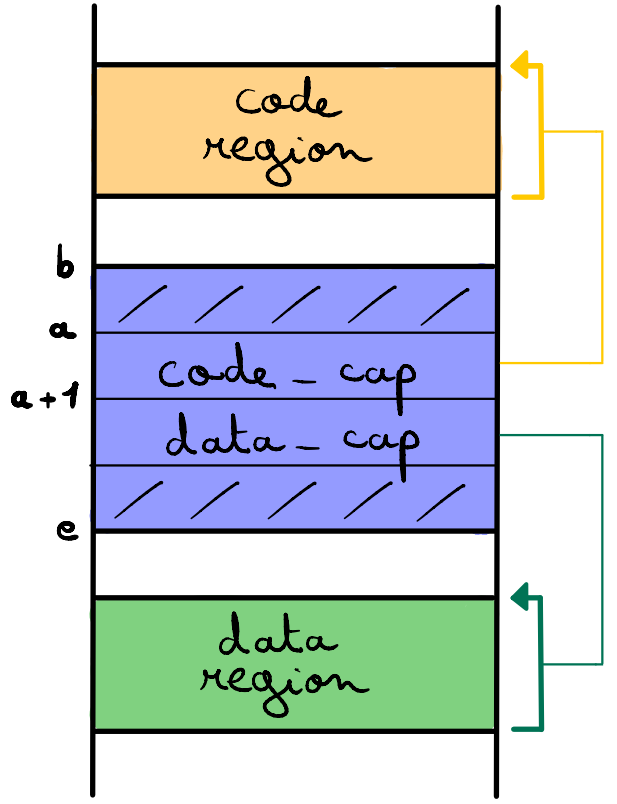


What are Indirect Sentries ?

points-to-pair

(IE, b, e, a)

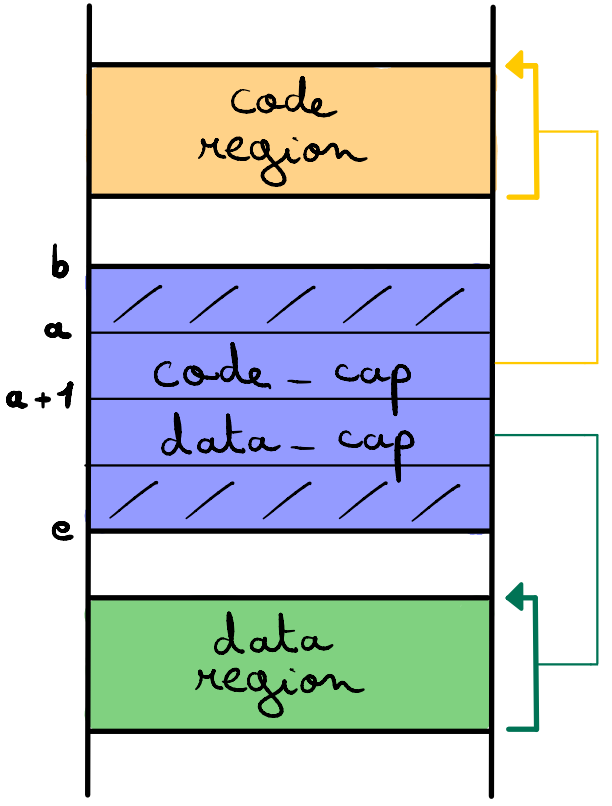
address



What are Indirect Sentries ?

points-to-pair

(IE, b, e, a)



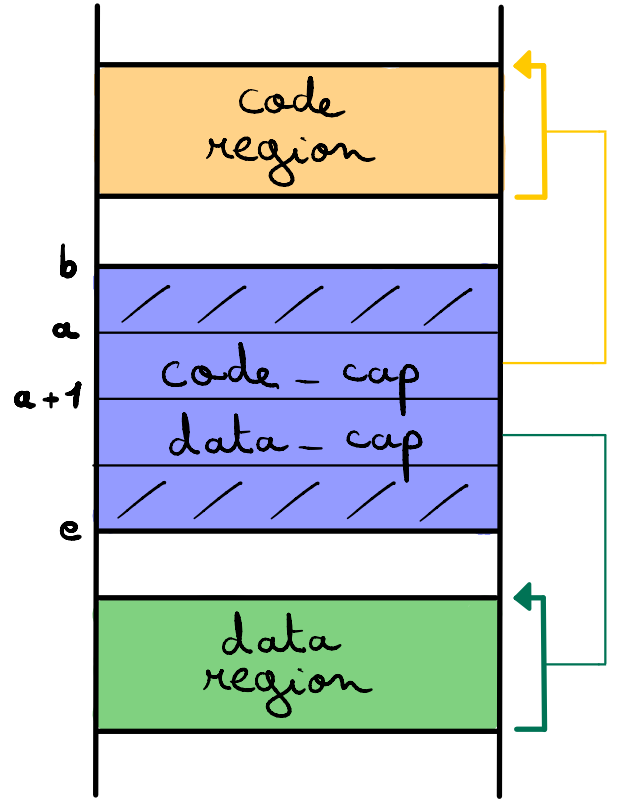
- immutable
- can only be jumped to

What are Indirect Sentries ?

points-to-pair

Jumps to

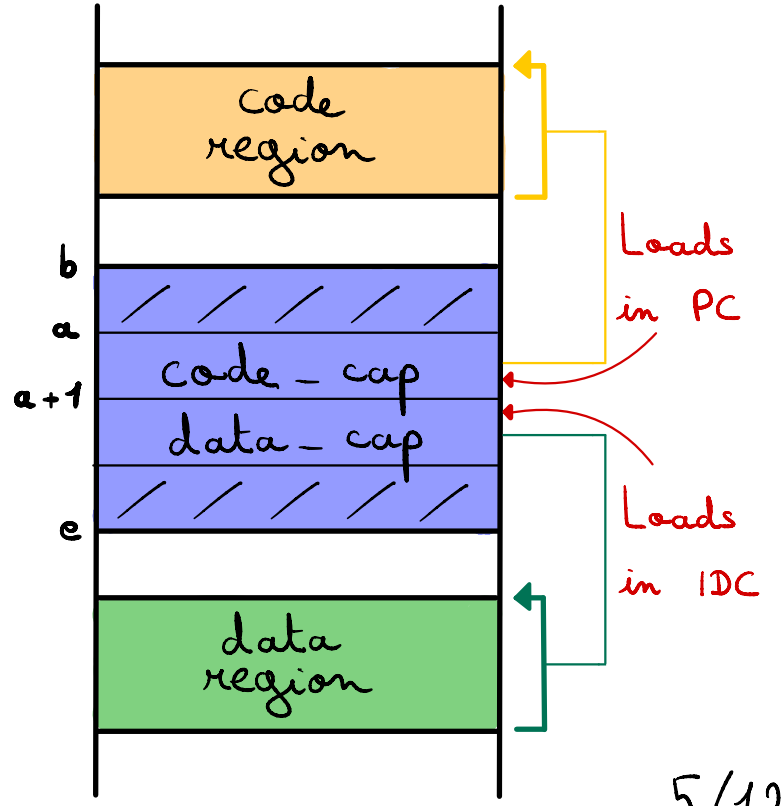
(IE, b, e, a)



What are Indirect Sentries ?

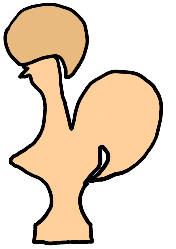
points-to-pair

Jumps to
(IE, b, e, a)



METHODOLOGY

extending [Georges et al.]

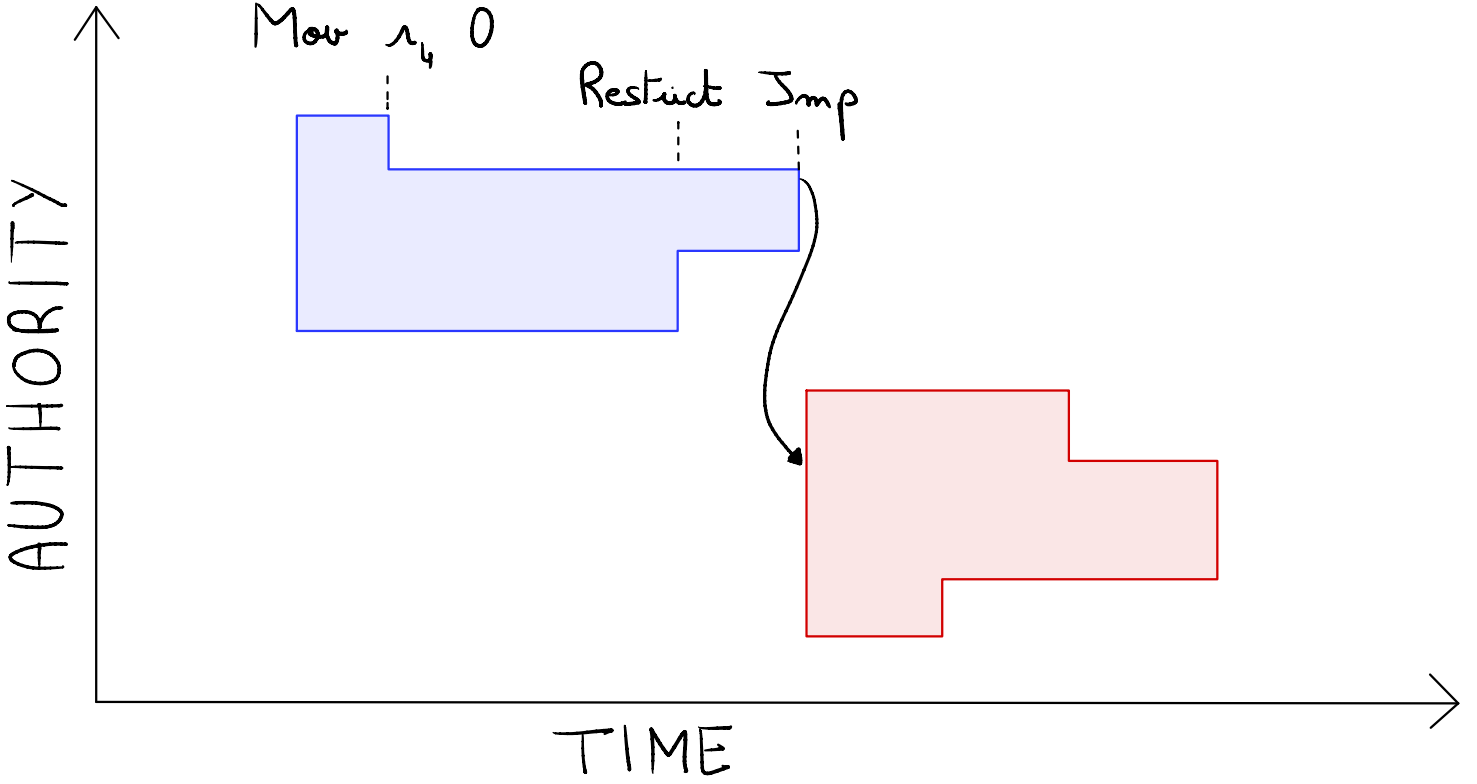


1) Operational Semantics
   - → mini - CHERI, more tractable proofs

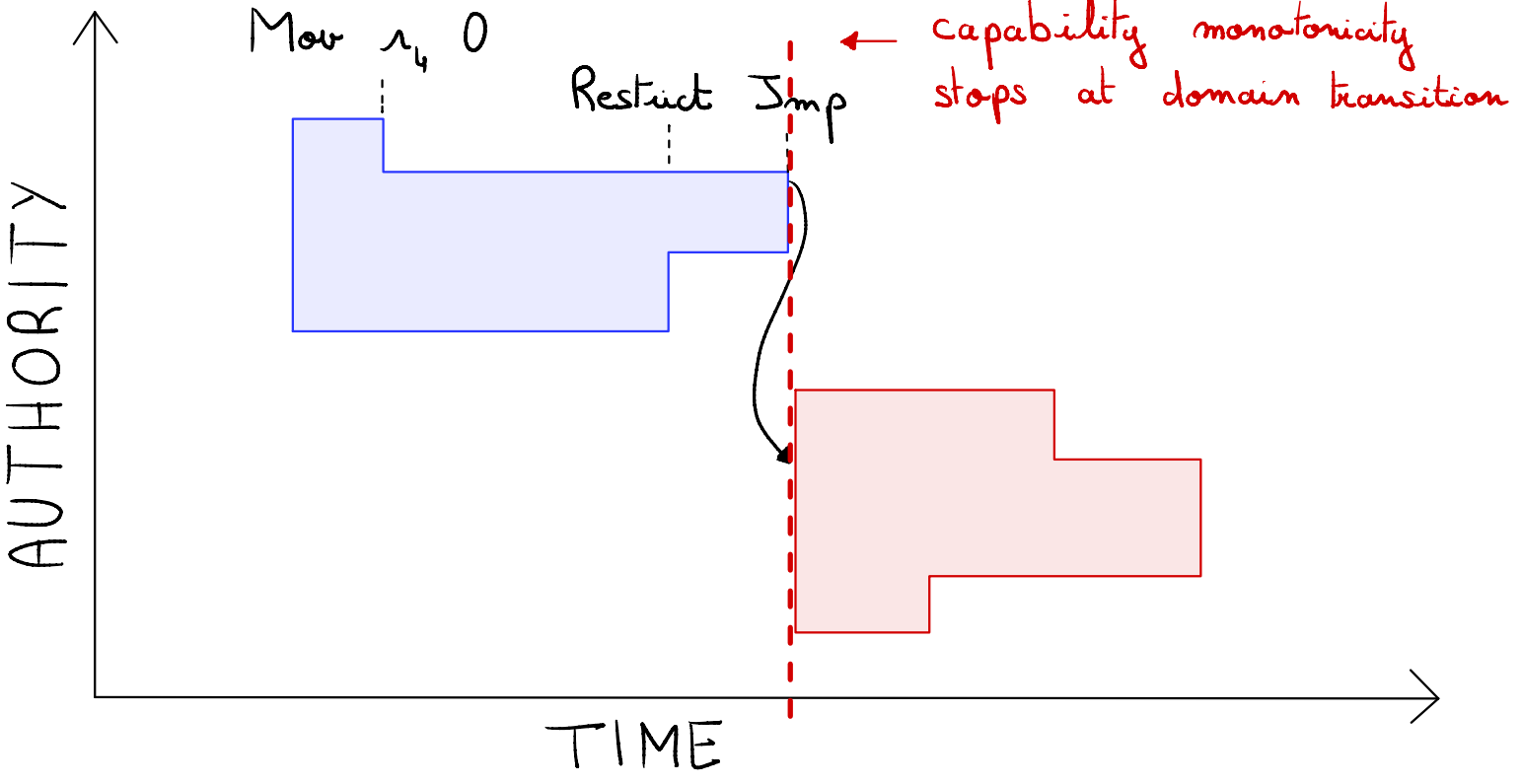
2) Program Logic
   - → reason about <u>known</u> code
   - → close to Cerise (built on top of the Iris separation logic)

3) Logical Relation
   - → reason about <u>unknown</u> code
   - → challenging

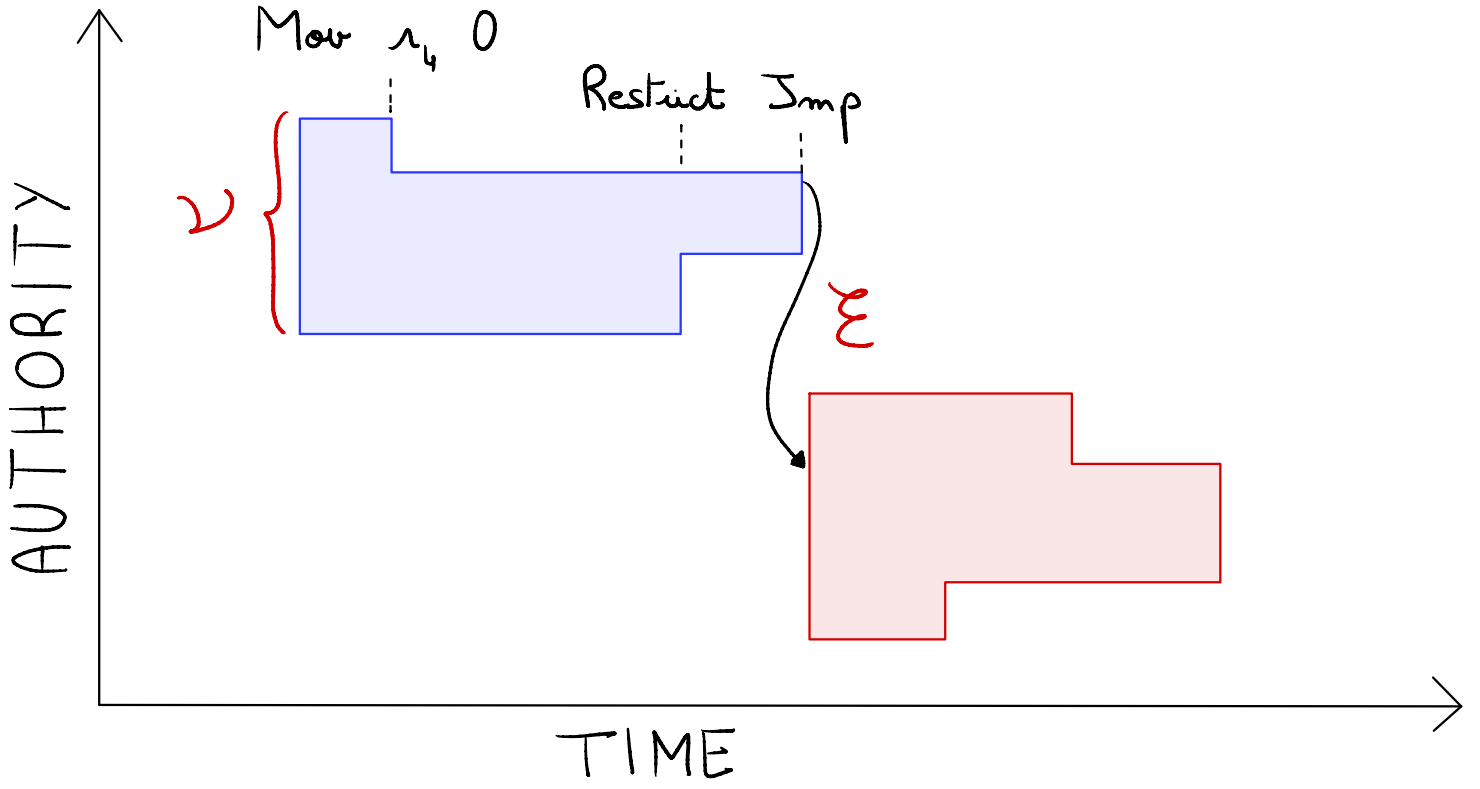
Logical Relation



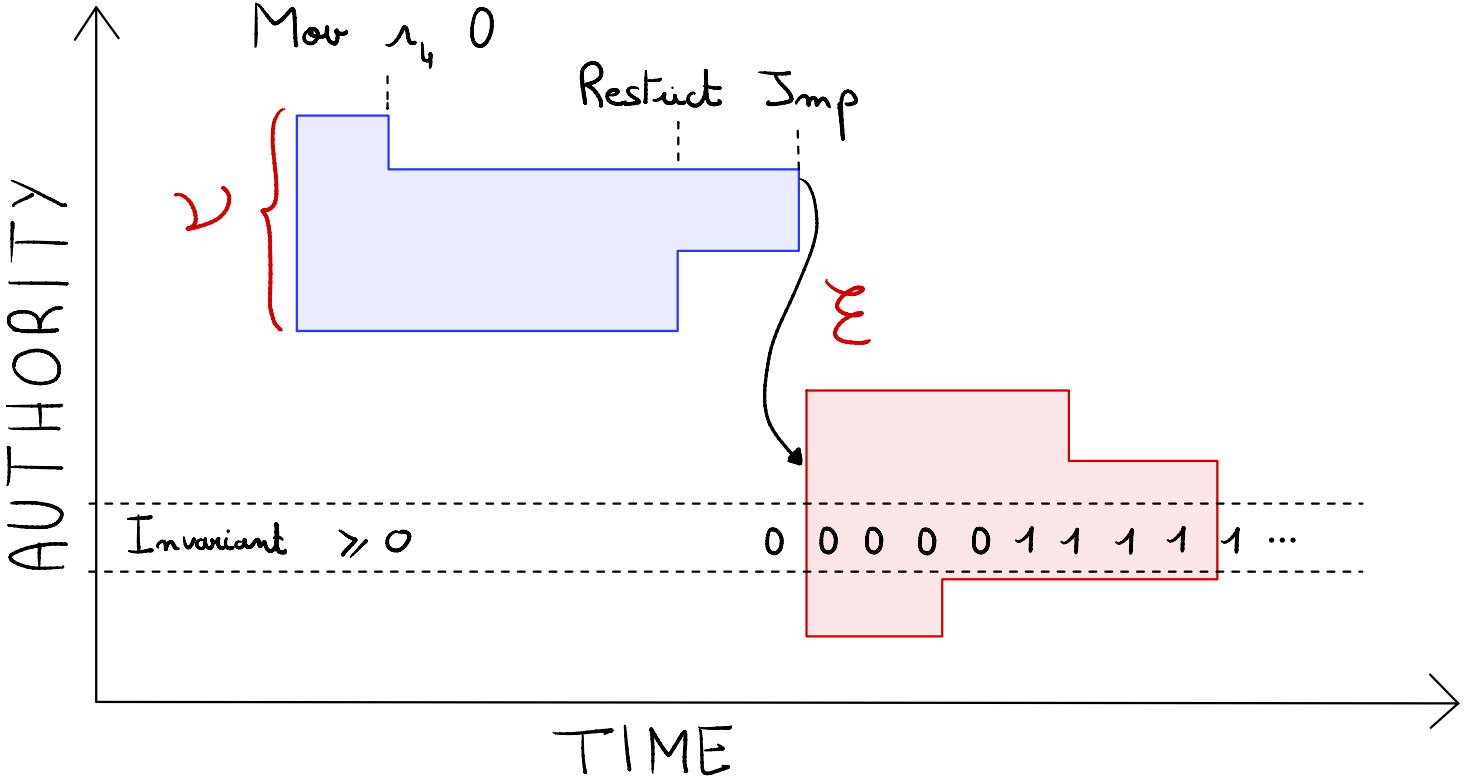
Logical Relation



Logical Relation



Logical Relation



Logical Relation with IE

- Intends to capture capability safety
- $\mathcal{V}(w)$: safe-to-share
  → an adversary using w cannot break existing invariants

  user-defined ↓
- $\mathcal{E}(w_1, w_2)$: safe-to-execute
  → if $PC \mapsto w_1$, $IDC \mapsto w_2$ and $r \mapsto \mathcal{V}(w)$,
  then executing the machine does not break existing invariants

- Jumping to unknown code, with safe registers, is safe

Logical Relation with IE

$$\mathcal{E}(w) \triangleq \forall w'.\, \mathcal{V}(w') \twoheadrightarrow \mathcal{E}_{\mathcal{G}}(w, w')$$

$$\mathcal{E}_{\mathcal{G}}(w_1, w_2) \triangleq \left\{ w_1 \;;\; \underset{\{(r,w)\mid (r,w)\in regs\}}{\ast} \mathsf{idc} \Rrightarrow w_2 \ast \; r \Rrightarrow w \ast \mathcal{V}(w) \right\} \rightsquigarrow \bullet$$

$\mathcal{V}(w)$

Regular Sentries

$$\mathcal{V}(z) \triangleq \text{True for } z \in \mathbb{Z}$$

$$\mathcal{V}(\mathrm{O}, -, -, -) \triangleq \text{True}$$

$$\mathcal{V}(\mathrm{E}, b, e, a) \triangleq \triangleright\, \Box\, \mathcal{E}(\mathrm{RX}, b, e, a)$$

$$\mathcal{V}(\mathrm{RW/RWX}, b, e, -) \triangleq \ast_{a\in[b,e)} \boxed{\exists w.\, a \mapsto w \ast \mathcal{V}(w)}$$

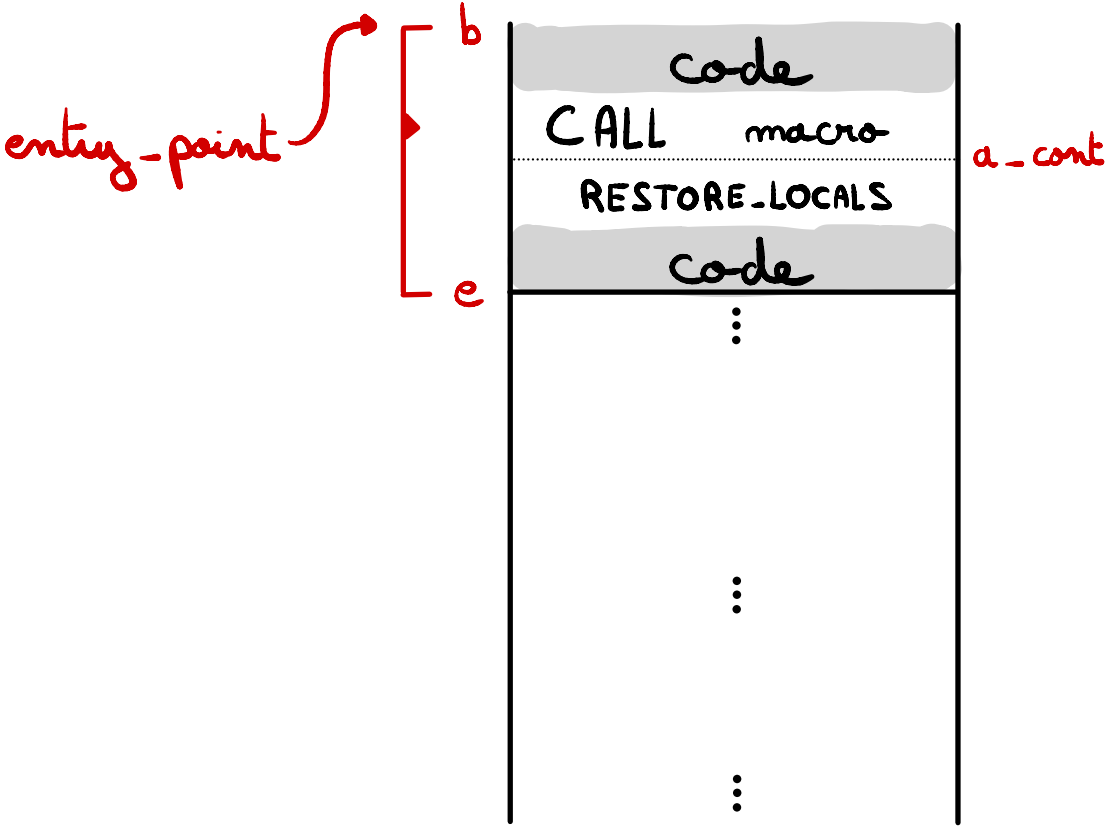
$$\mathcal{V}(\mathrm{RO/RX}, b, e, -) \triangleq \ast_{a\in[b,e)} \exists P.\, \boxed{\exists w.\, a \mapsto w \ast P(w)} \ast \triangleright\, \Box\, (\forall w.\, P(w) \twoheadrightarrow \mathcal{V}(w)) \ast \mathit{persistent\_cond}\ P$$

Indirect Sentries

$$\mathcal{V}(\mathrm{IE}, b, e, a) \triangleq \text{(1)}\ \lceil b \le a < a+1 < e \rceil \twoheadrightarrow \left(\begin{array}{l} \text{(2)}\ \exists P_1, P_2.\, \mathit{persistent\_cond}\ P_1 \ast \mathit{persistent\_cond}\ P_2 \ast \\ \text{(3a)}\ \boxed{\exists w_1.\, a \mapsto w_1 \ast P_1(w_1)}^{\,a} \ast \\ \text{(3b)}\ \boxed{\exists w_2.\, (a+1) \mapsto w_2 \ast P_2(w_2)}^{\,(a+1)} \ast \\ \text{(4)}\ \forall w_1, w_2.\, \triangleright\, \Box\, (P_1(w_1) \ast (P_2(w_2) \vee \mathit{is\_int}(w_2)) \twoheadrightarrow \mathcal{E}_{\mathcal{G}}(w_1, w_2)) \end{array} \right)$$

Secure Calling Convention

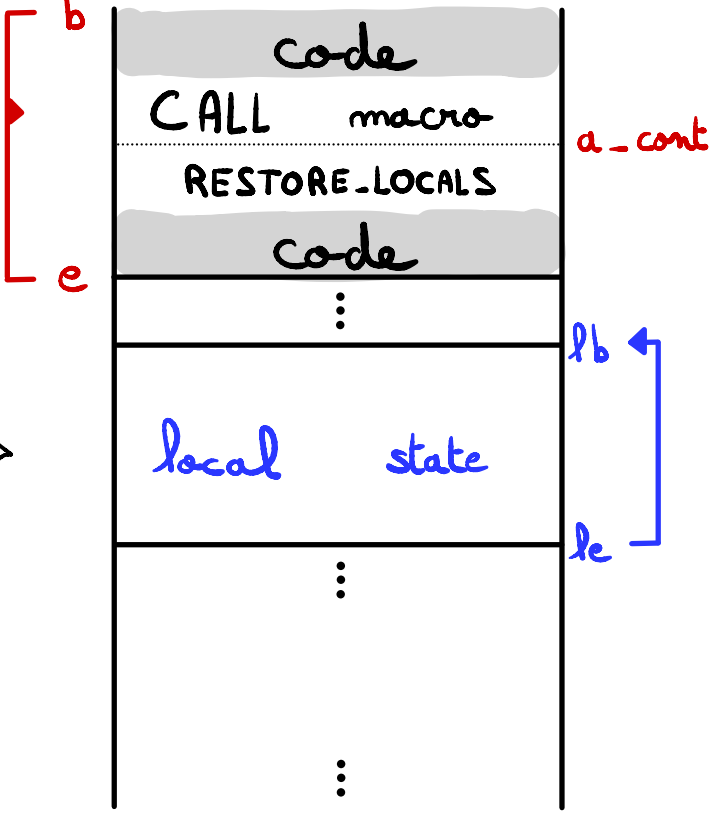
PC	(RX, b, e, -)
r_0	
⋮	⋮



Secure Calling Convention

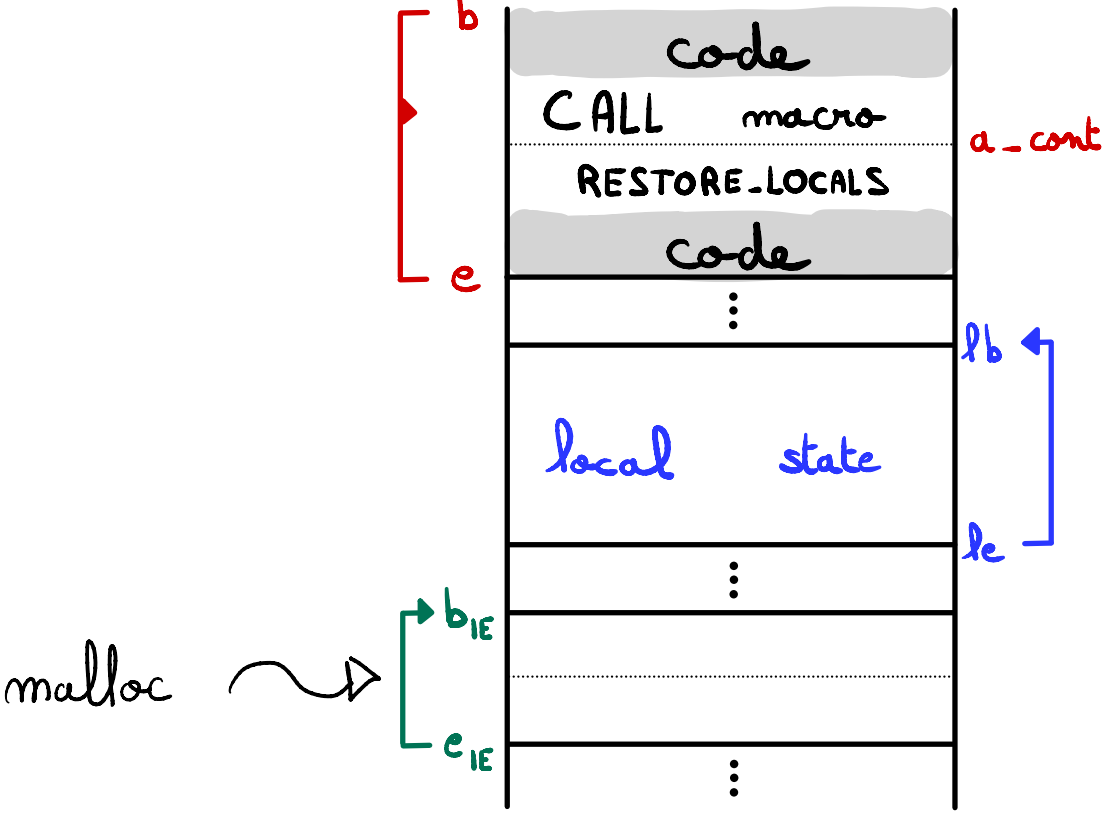
PC	(RX, b, e, -)
r_0	
⋮	⋮

malloc



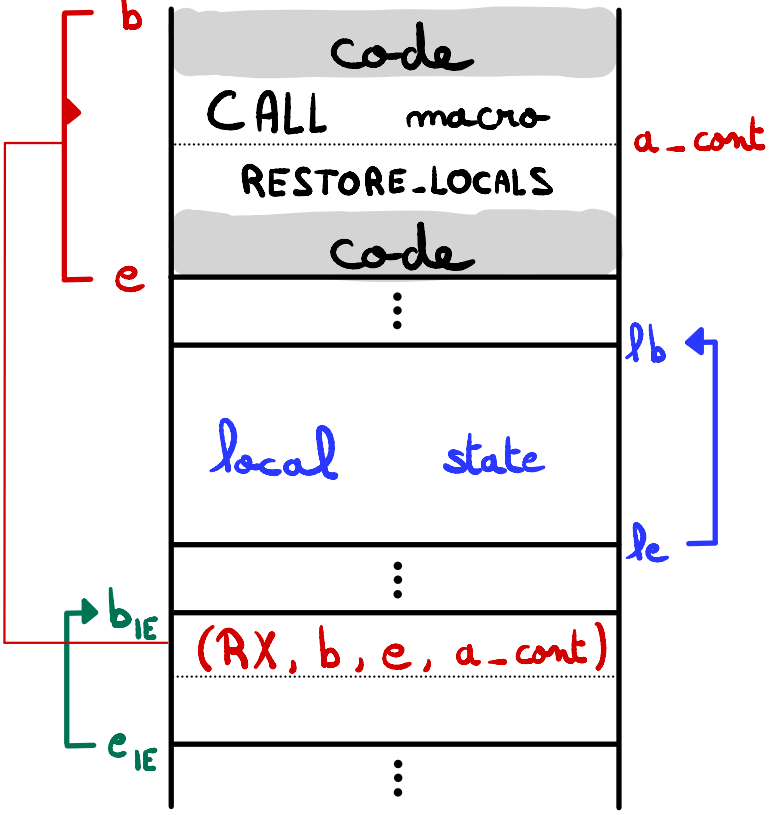
Secure Calling Convention

PC	(RX, b, e, -)
r_0	
⋮	⋮



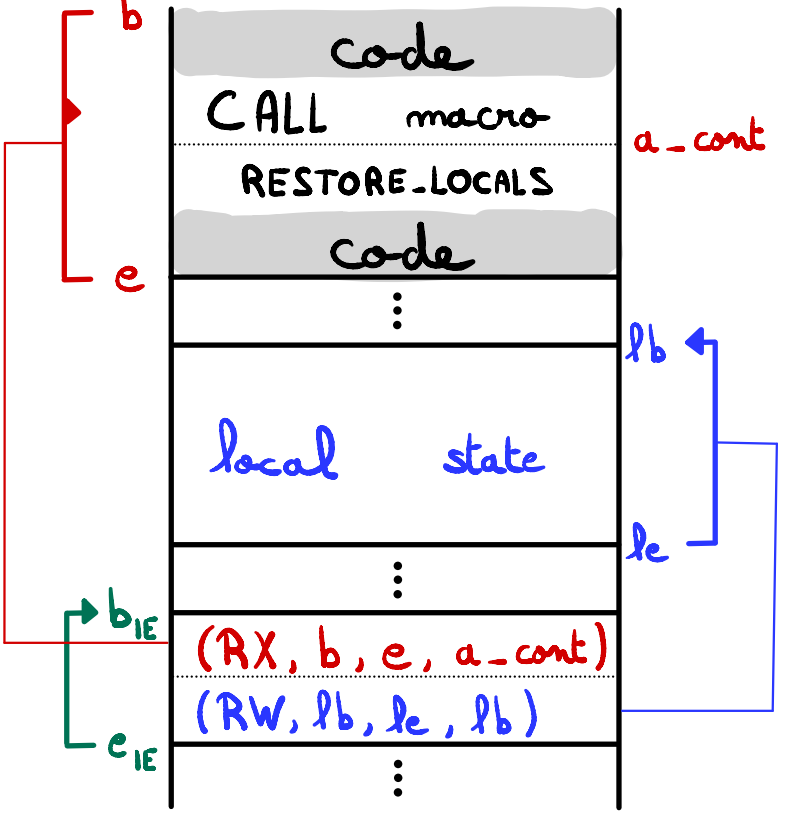
Secure Calling Convention

PC	(RX, b, e, -)
r_0	
⋮	⋮

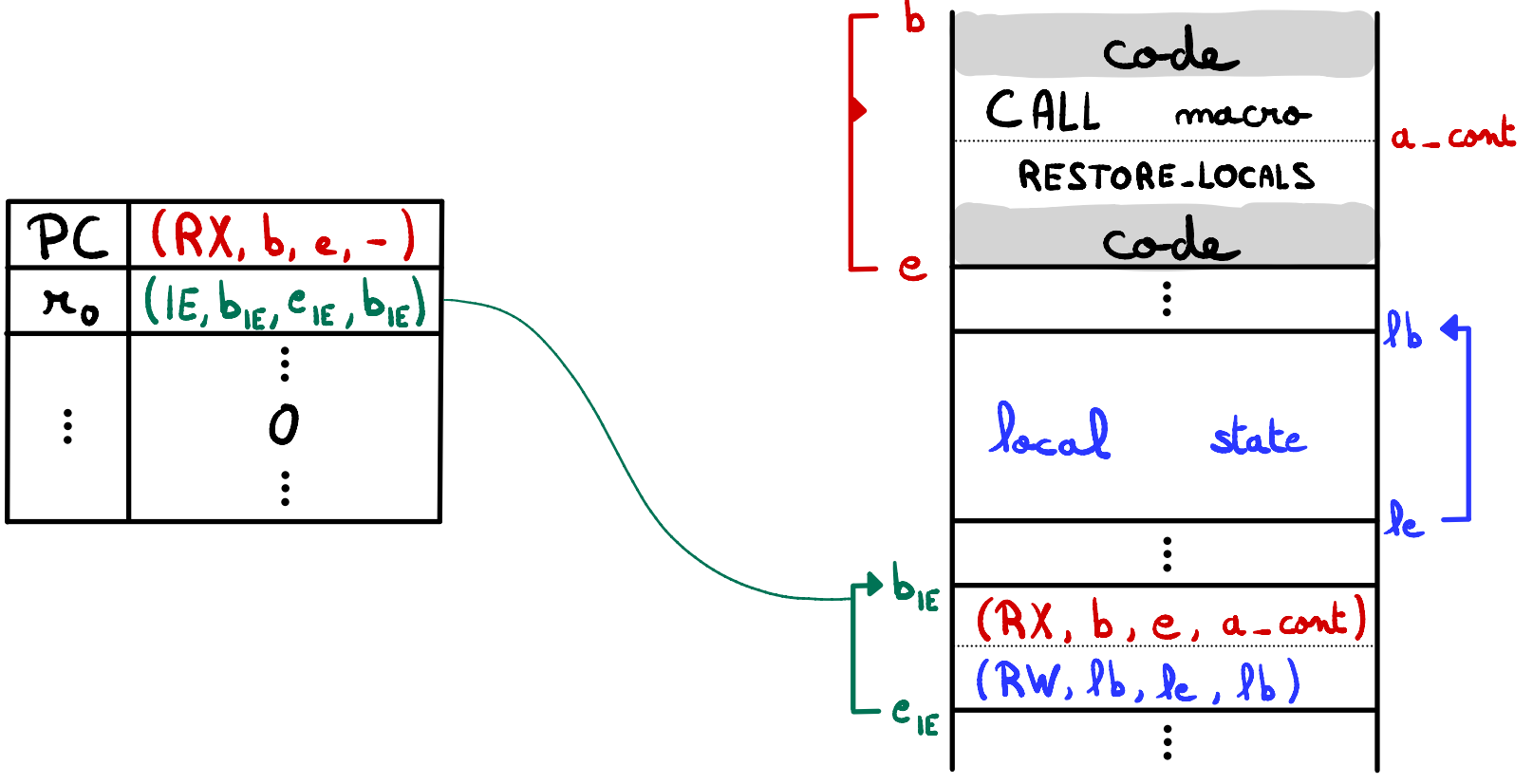


Secure Calling Convention

PC	(RX, b, e, -)
r_0	
⋮	⋮



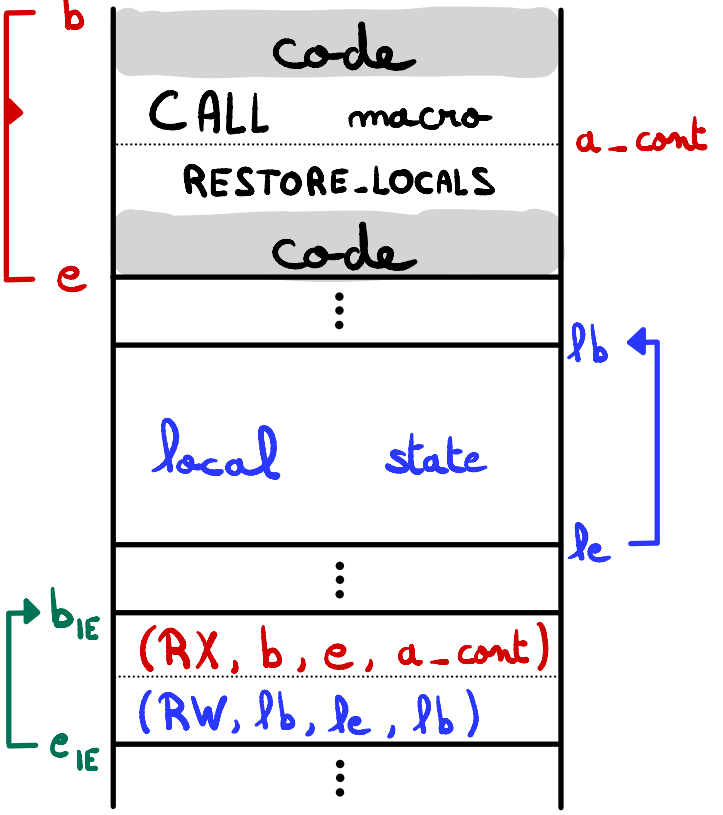
Secure Calling Convention



Secure Calling Convention

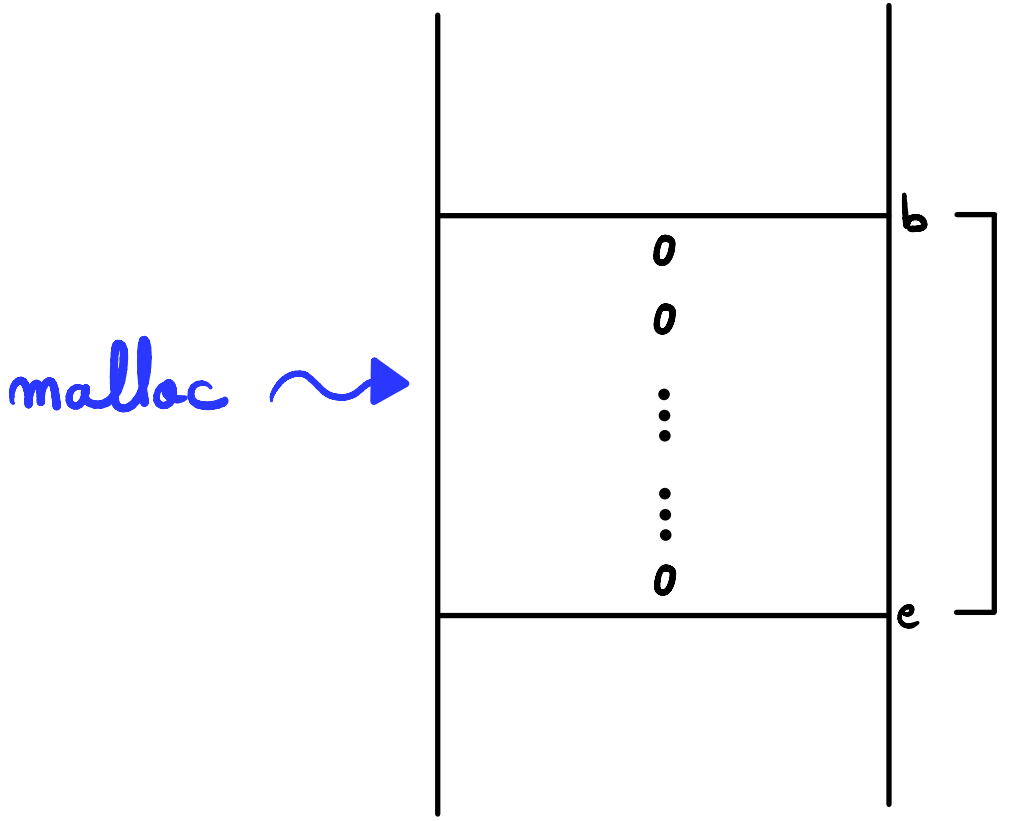
PC	(RX, b, e, -)
r_0	$(IE, b_{IE}, e_{IE}, b_{IE})$
⋮	⋮
	0
	⋮

ν

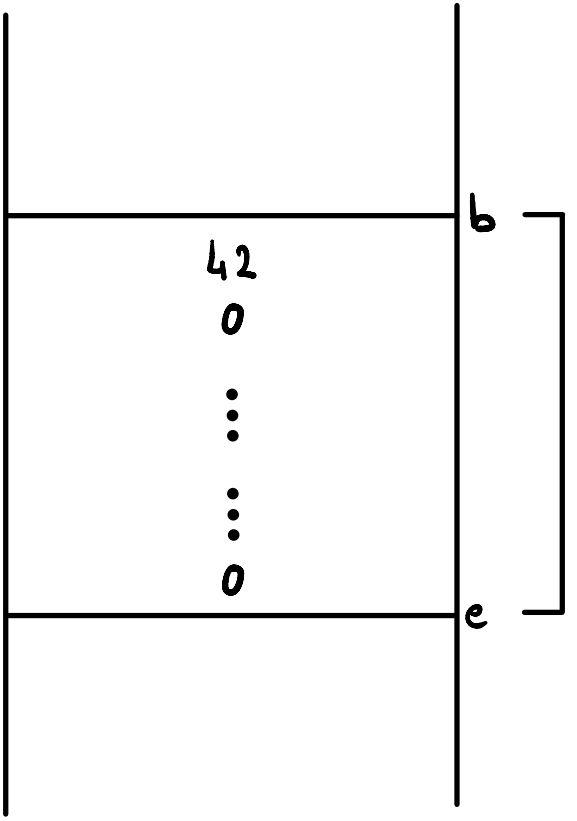


How do we prove that our calling convention is "SECURE"?

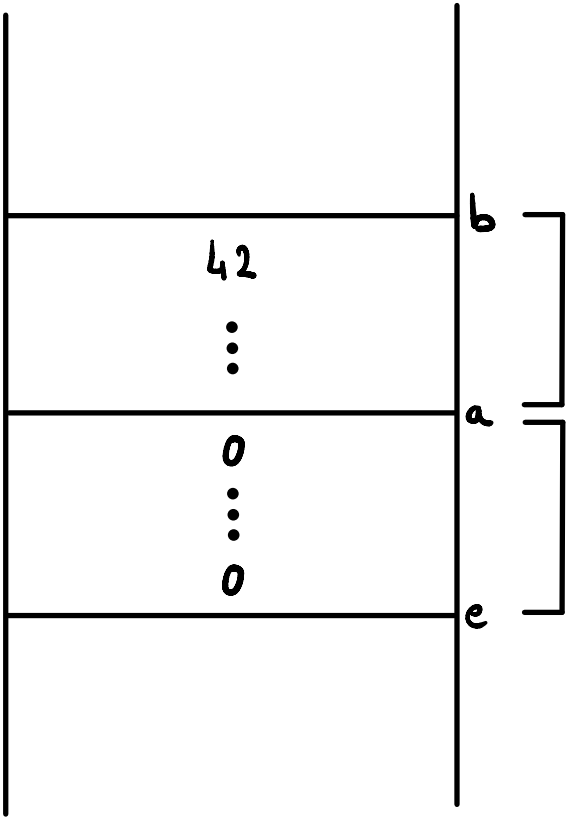
Sub-buffer Example



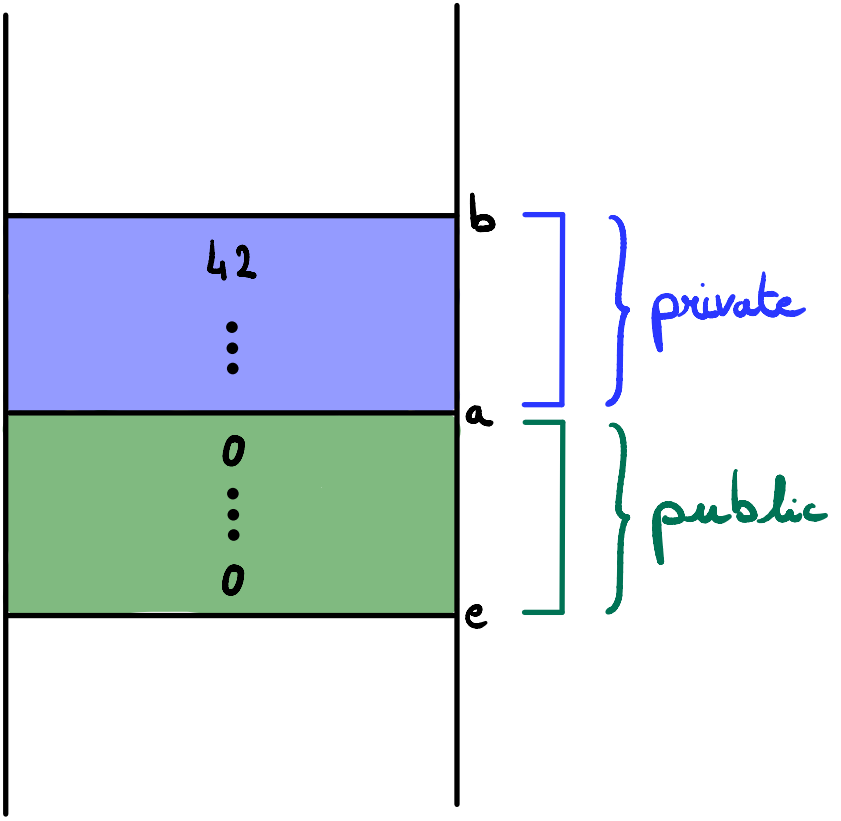
SUB-BUFFER EXAMPLE



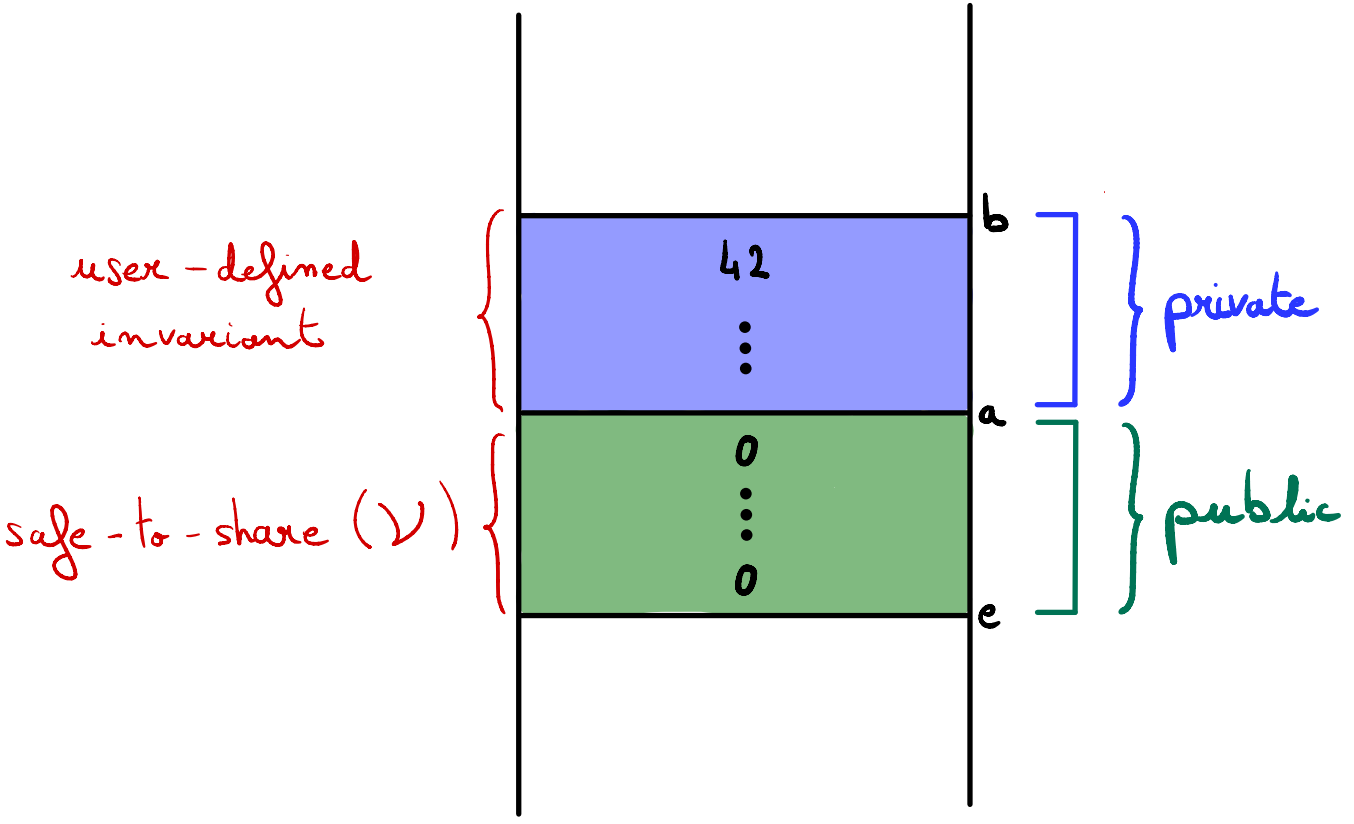
Sub-buffer Example



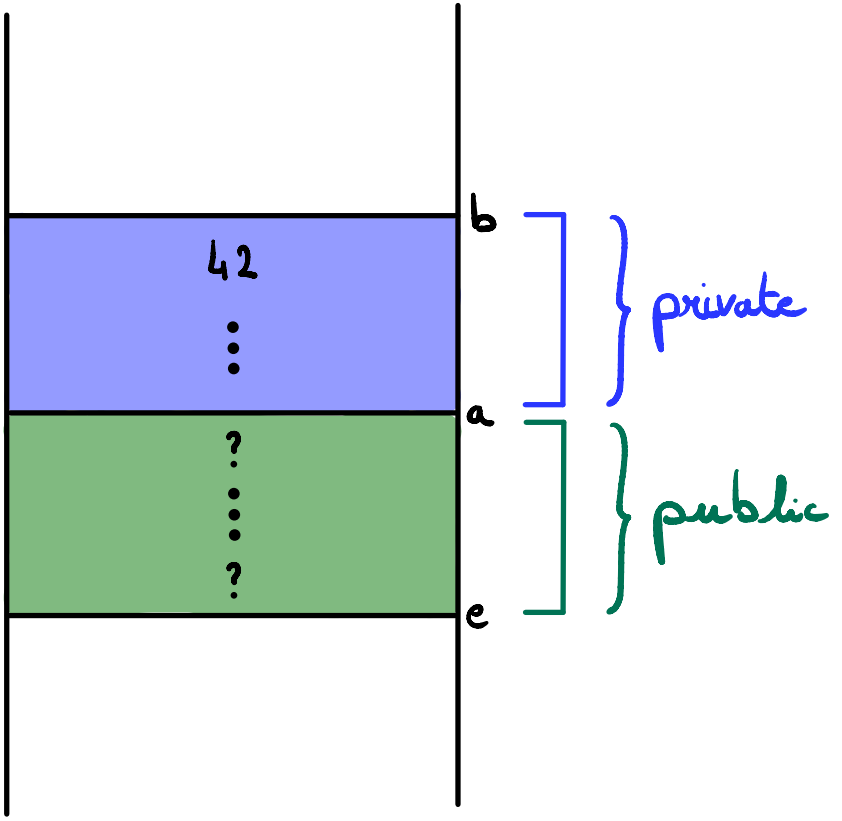
Sub-buffer Example



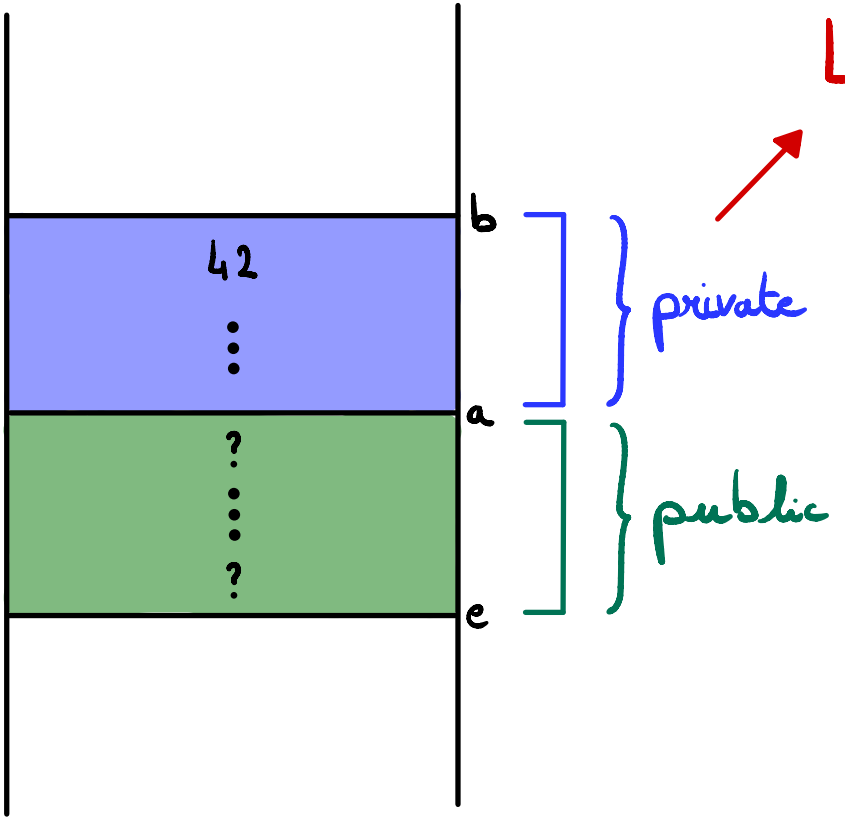
Sub-buffer Example



Sub-buffer Example



Sub-buffer Example



LSE !!

Summary

- Contributions

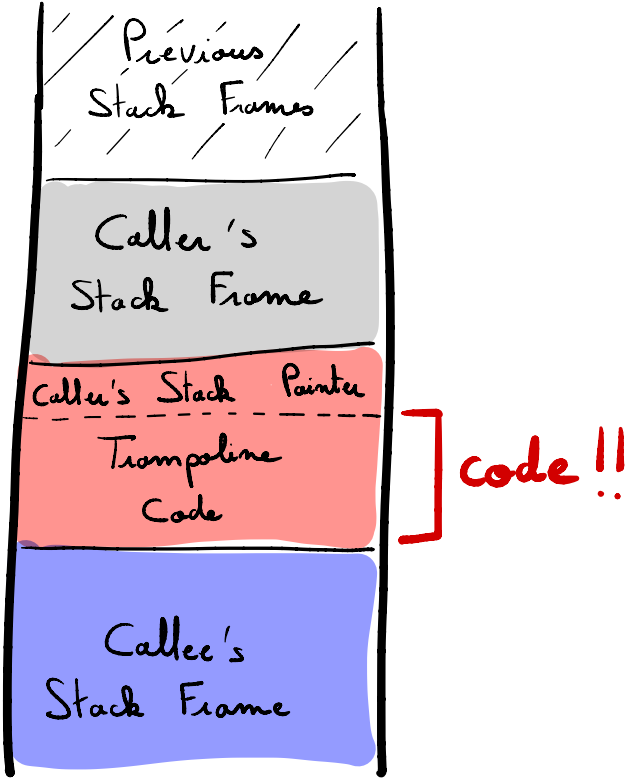
  → Cerise (mini-CHERI) with Indirect Sentries
  → Secure heap-based calling convention without trampoline code
  → Illustrate Robust Capability Safety on concrete examples

- Future Work

  → Combine with "Stack Cerise": locality bit, uninitialised and directed capabilities
  → Stack-based calling convention ensuring Well-Bracketed Control Flow

Backup Slides

Secure Calling Convention



- Trampoline code recovers caller's stack pointer
- Executable Stack → breaks **least privilege** principle
- Indirect Sentries **eliminate** the need of trampoline code

Secure Calling Convention