Efficient Formally Secure Compilers to a Tagged Architecture

Cătălin Hrițcu

Inria Paris

Prosecco team

5 year vision

ERC SECOMP: https://secure-compilation.github.io

Computers are insecure

- **devastating low-level vulnerabilities**
- **teasing out 2 important security problems:**

  1. inherently insecure low-level languages

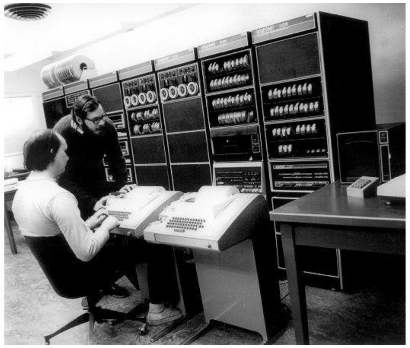
  – **memory unsafe**: any buffer overflow can be catastrophic allowing remote attackers to gain complete control

  2. unsafe interoperability with lower-level code

  – even code written in **safer languages** has to interoperate with **insecure low-level libraries**

  – **unsafe interoperability:** high-level safety guarantees lost

How did we get here?



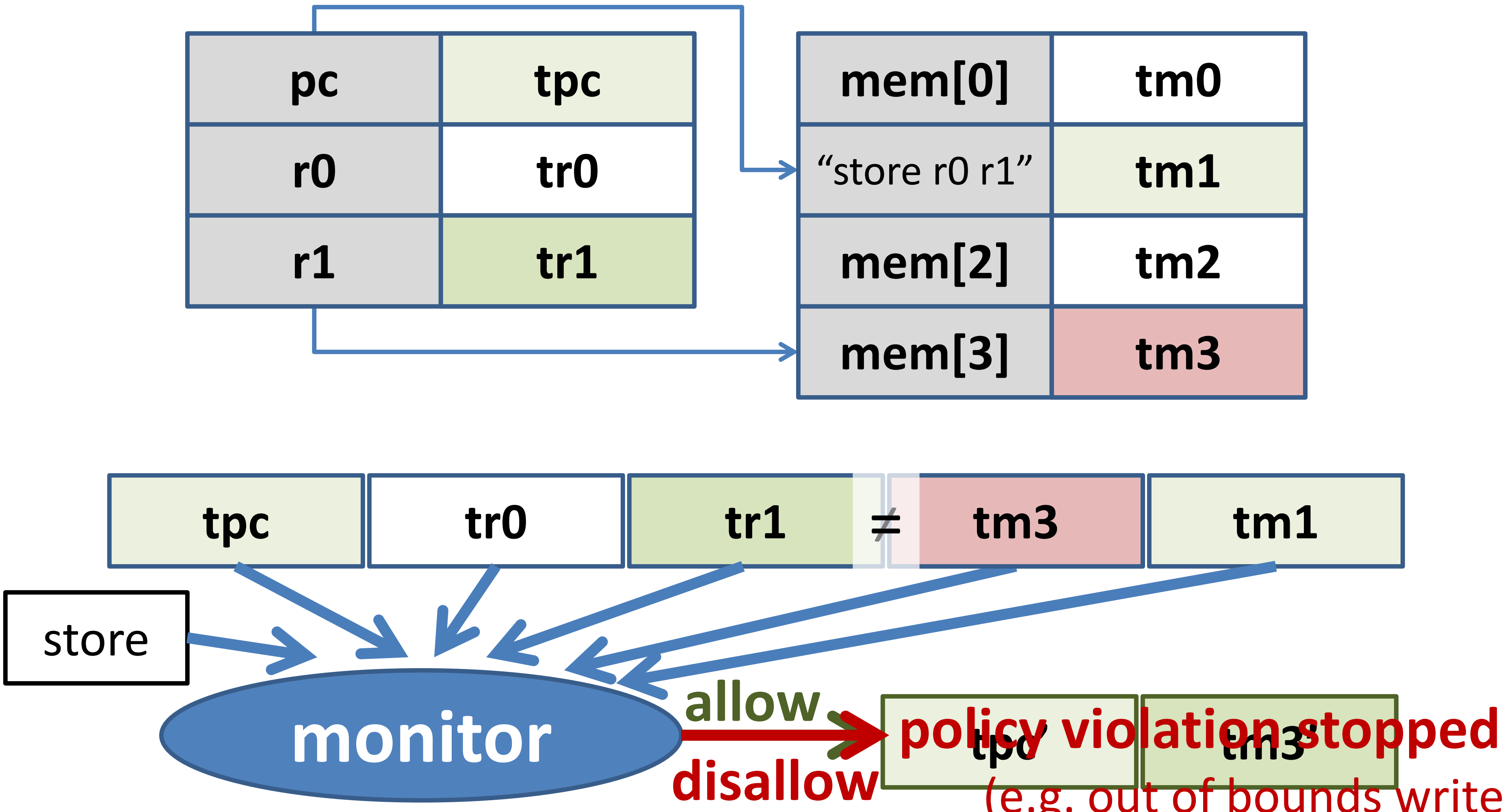
- **programming languages, compilers, and hardware architectures**
  - designed in an era of **scarce hardware resources**
  - too often **trade off security for efficiency**
- **the world has changed** (2017 vs 1972*)
  - security matters, hardware resources abundant
  - time to revisit some tradeoffs

* "...the number of UNIX installations has grown to 10, with more expected..."

-- *Dennis Ritchie and Ken Thompson, June 1972*

Key enabler: Micro-Policies

software-defined, hardware-accelerated, tag-based monitoring



software monitor's decision is hardware cached

Micro-policies are cool!

- **low level + fine grained**: unbounded per-word metadata, checked & propagated on each instruction
- **flexible**: tags and monitor defined by software
- **efficient**: software decisions hardware cached
- **expressive**: complex policies for secure compilation
- **secure** and **simple** enough to verify security in Coq
- **real**: FPGA implementation on top of RISC-V

Expressiveness

Way beyond MPX, SGX, SSM, etc

- information flow control (IFC) [POPL'14]
- monitor self-protection
- protected compartments
- dynamic sealing
- heap memory safety
- code-data separation
- control-flow integrity (CFI)
- taint tracking
- ...

Verified (in Coq) [Oakland'15]

Evaluated (<10% runtime overhead) [ASPLOS'15]

Micro-Policies team

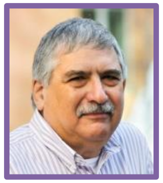
- **Formal methods** & **architecture** & **systems**
- **Current team**:
  - *Inria Paris*: **Cătălin Hrițcu, Guglielmo Fachini, Marco Stronati, Théo Laurent**
  - *UPenn*: **André DeHon**, **Benjamin Pierce, Arthur Azevedo de Amorim**, **Nick Roessler**
  - *Portland State*: **Andrew Tolmach**
  - *MIT:* **Howie Shrobe, Stelios Sidiroglou-Douskos**
  - *Industry*: **Draper Labs**
- **Spinoff of past project: DARPA CRASH/SAFE (2011-2014)**









DRAPER

SECOMP grand challenge

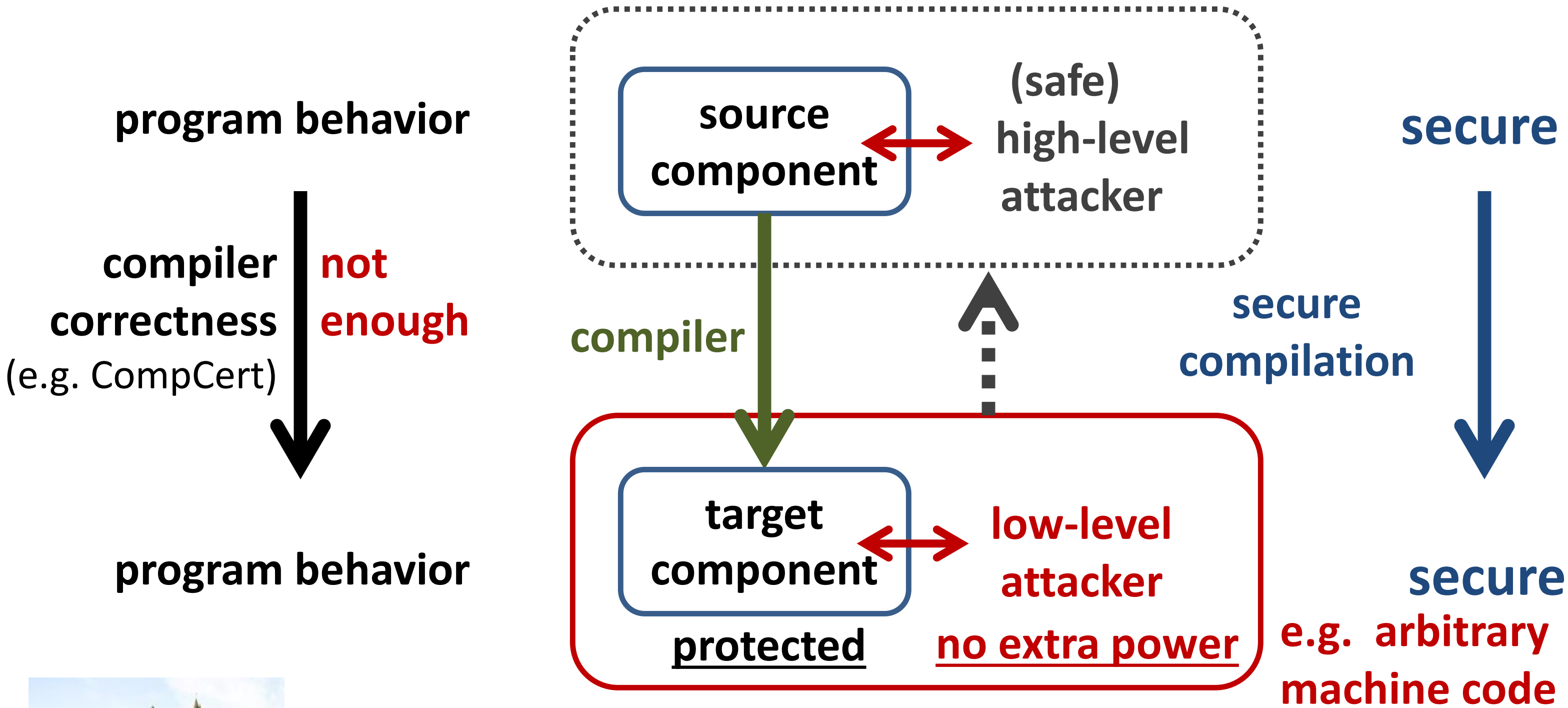
Use micro-policies to build **the first efficient formally secure compilers** for **realistic programming languages**

1. **Provide secure semantics for low-level languages**
   – C with protected components and memory safety
2. **Enforce secure interoperability with lower-level code**
   – ASM, C, and Low*

[= safe C subset embedded in F* for verification]

Secure Compilation

holy grail of preserving security all the way down



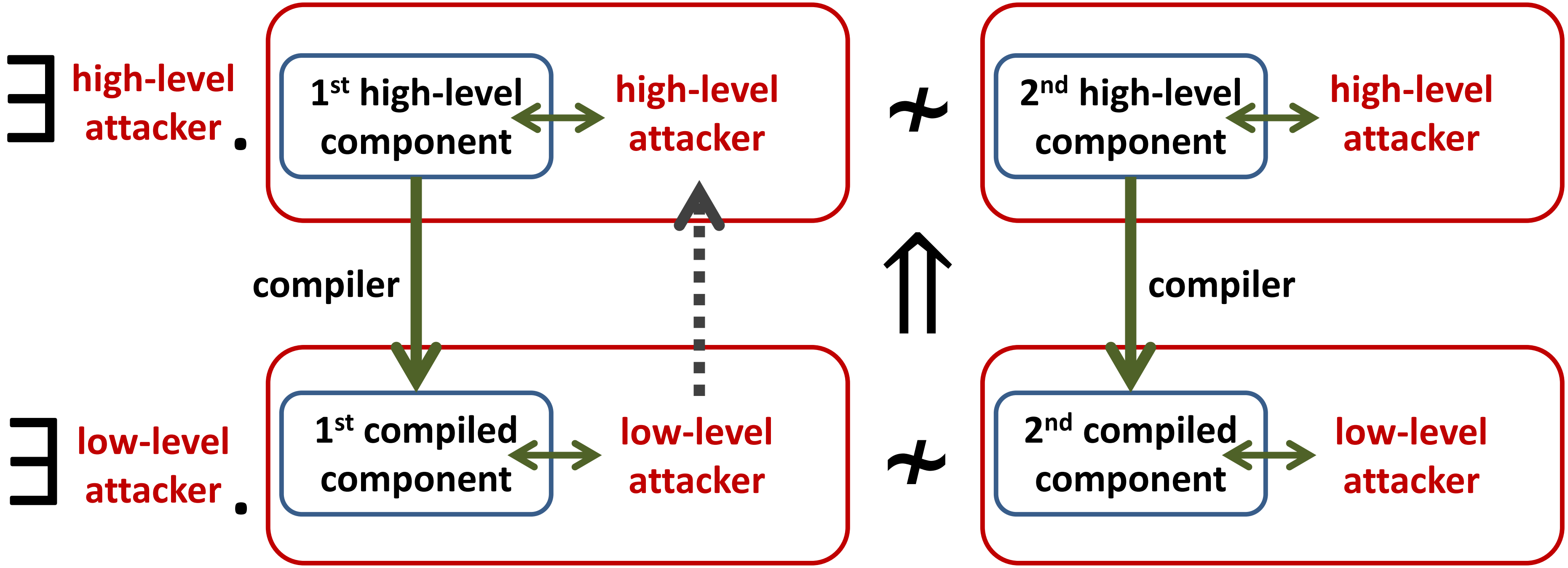
Benefit: sound security reasoning in the source language

forget about compiler chain (linker, loader, runtime system)

forget that libraries are written in a lower-level language

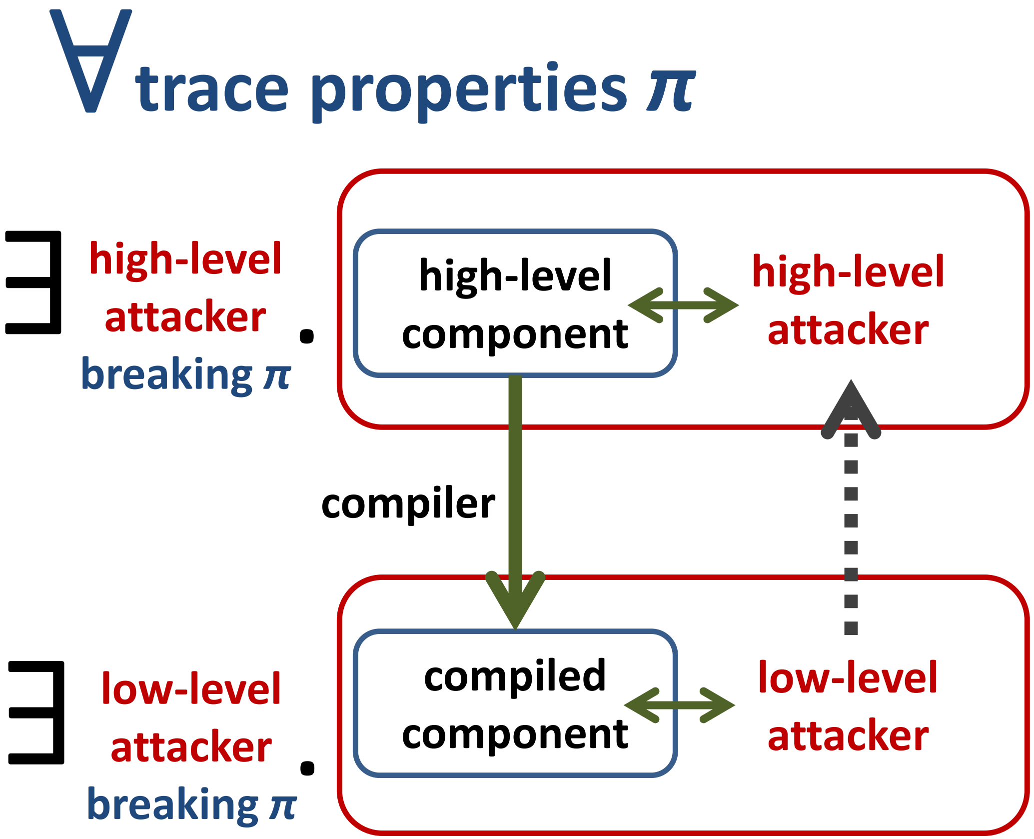
Our original secure compilation target:
fully abstract compilation

(preservation of observational equivalence)



Problems: (1) **very hard to *realistically* achieve**
(hopeless against timing side channels;
more realistic: preservation of noninterference)
(2) **very difficult to prove** ……

Our new first target: robust compilation



- **robust satisfaction preserved** (adversarial context)
- **gives up** on confidentiality (relational/hyper properties)
  - more robust to side channels
- **conjectures:**
  - **stronger** than (compositional) compiler correctness
  - **weaker** than full abstraction + compiler correctness
- **less extensional** than FA

Advantages: **easier to realistically achieve and prove**

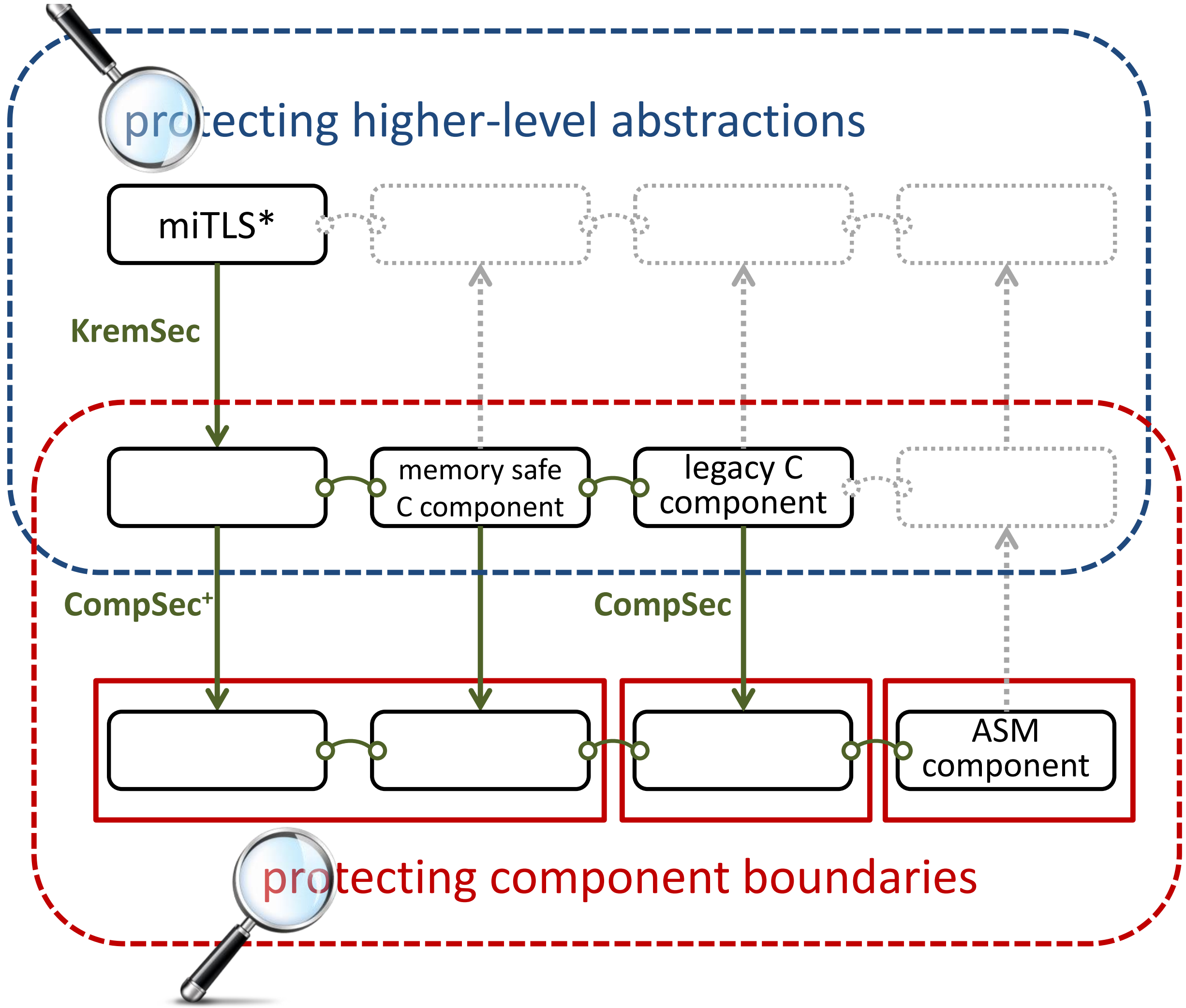
still useful: preservation of **invariants** and other **integrity properties**

SECOMP: achieving secure compilation at scale

Low* language
(safe C subset in F*)

C language
+ components
+ memory safety

ASM language
(RISC-V + micro-policies)



Protecting component boundaries

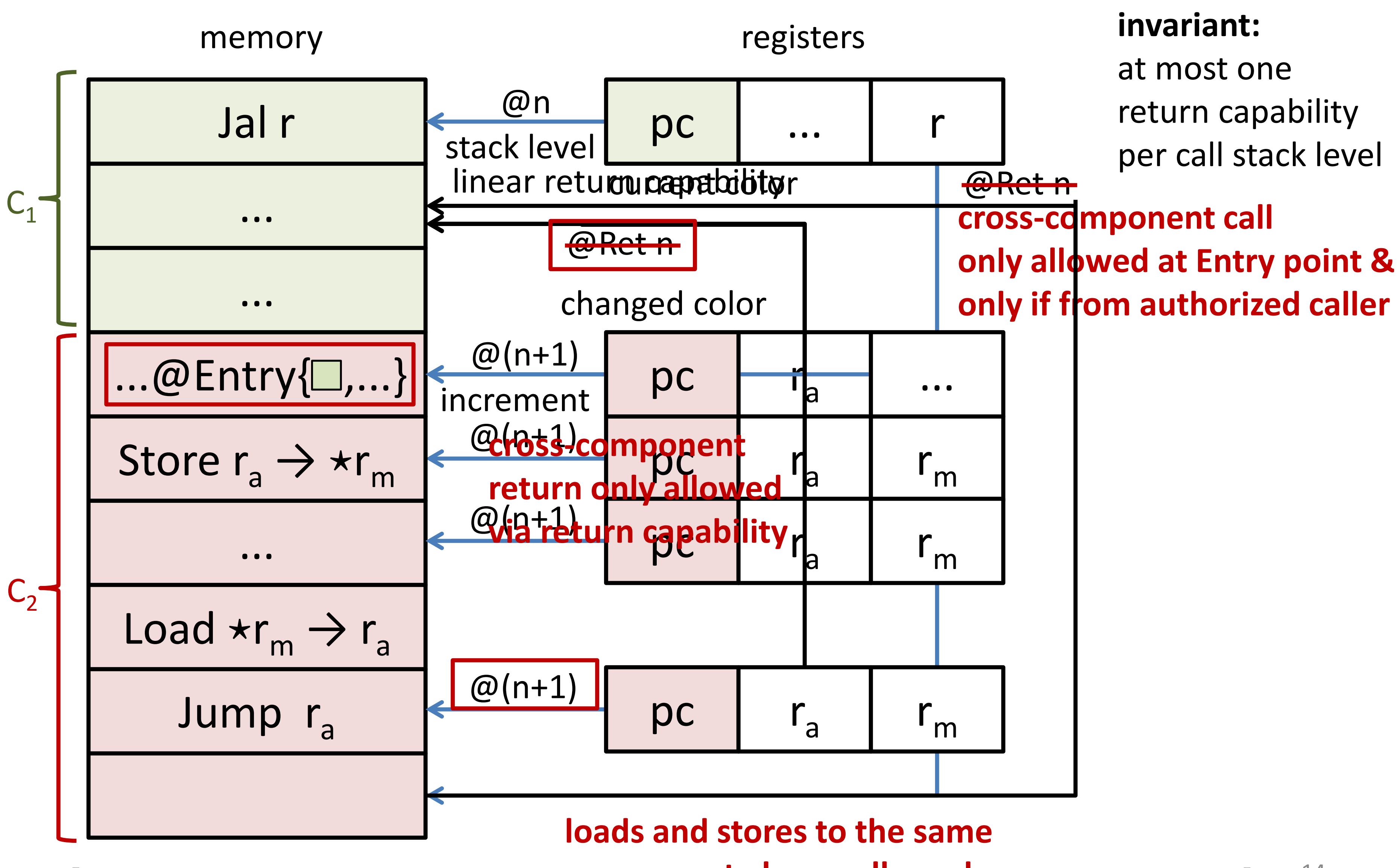
- **Add mutually distrustful components to C**
  - interacting only via **strictly enforced interfaces**

- **CompSec compiler chain** (based on CompCert)
  - propagate interface information to produced binary
- **Micro-policy simultaneously enforcing**
  - component separation
  - type-safe procedure call and return discipline

- **Interesting attacker model**
  - mutual distrust, unsafe source language

Ongoing work, started with Yannis Juglaret et al

Protected components micro-policy

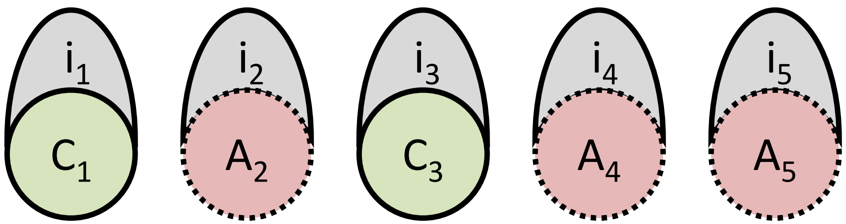


[Towards a Fully Abstract Compiler Using Micro-Policies, Juglaret et al, TR 2015]

Mutual-distrust attacker model

(more interesting compared to vanilla FA or RC)

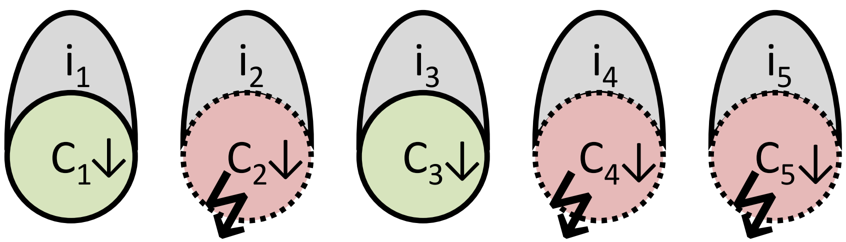
∀compromise scenarios *s*. ∀scenario-indexed trace properties π.



violates $\pi(s)$

∃ high-level attack from some fully defined A_2, A_4, A_5

C_1 and C_3 fully defined



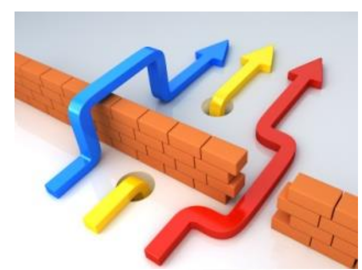
violates $\pi(s)$

∃ low-level attack from compromised $C_2\downarrow$, $C_4\downarrow$, $C_5\downarrow$

[Beyond Good and Evil, Juglaret, Hritcu, et al, CSF'16]

Protecting higher-level abstractions

- **Low*: enforcing specifications in C**
  - some can be turned into **contracts,** checked dynamically; **micro-policies can speed this up**
- **Limits of purely-dynamic enforcement**
  - functional purity, termination, relational reasoning
  - **push these limits further and combine with static analysis**



SECOMP focused on dynamic enforcement
but combining with static analysis can ...

- **improve efficiency**
  - **removing spurious dynamic checks**
  - e.g. turn off pointer checking for a statically memory safe component that never sends or receives pointers
- **improve transparency**
  - **allowing more safe behaviors**
  - e.g. statically detect which copy of linear return capability the code will use to return
  - in this case **unsound "static analysis" is fine**

Verification and testing

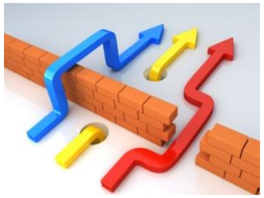
- So far most secure compilation work **on paper**
  - one can't verify an interesting compiler on paper
- SECOMP uses **proof assistants**: Coq and F*
- **Reduce effort**
  - more automation (e.g. based on SMT, like in F*)
  - integrate testing and proving (QuickChick and Luck)
- **Problem not just with scale of mechanization**
  - devising good **proof techniques** for secure compilation is a hot research topic of it's own

Remaining challenges for micro-policies

- **Micro-policies for C**
  - needed for vertical compiler composition
  - will put micro-policies in the hands of programmers
- **Secure micro-policy composition**
  - micro-policies are **interferent** reference monitors
  - one micro-policy's behavior can break another's guarantees
    - e.g. composing anything with IFC can leak

SECOMP in a nutshell

- **We need more secure languages, compilers, hardware**
- **Key enabler: micro-policies** (software-hardware protection)
- **Grand challenge: the first efficient formally secure compilers** for **realistic programming languages** (C and Low*)
- **Answering challenging fundamental questions**
  - properties/attacker models, proof techniques
  - secure composition, micro-policies for C



- **Achieving strong security properties**
  + testing and proving formally that this is the case
- **Measuring & lowering the cost of secure compilation**

- Most of this is **vaporware** at this point but ...
  - building a community, looking for collaborators, and hiring to make some of this real

BACKUP SLIDES

Collaborators & Community

- **Core team at Inria Paris**

  - Marco Stronati (PostDoc), Guglielmo Fachini and Théo Laurent (Interns)
  - Looking for excellent **interns**, **students**, **researchers**, and **engineers**
- **Traditional collaborators from Micro-Policies project**
  - UPenn, MIT, Portland State, Draper Labs
- **Other researchers working on secure compilation**
  - Deepak Garg (MPI-SWS), Frank Piessens (KU Leuven),
    Amal Ahmed (Northeastern), Cedric Fournet & Nik Swamy (MSR), …
- **Secure compilation meetings**
  - 1st at Inria Paris in Aug. 2016, 2nd at POPL in Jan. 2017, POPL workshop
  - Upcoming: Dagstuhl seminar on Secure Compilation, May 2018
  - **build larger research community, identify open problems,**
    bring together communities (HW, systems, security, PL, verification, ...)

Broad view on secure compilation

- **Different security goals / attacker models**
  - Fully abstract compilation and variants, **robust compilation**, noninterference preservation, ...
- **Different enforcement mechanisms**
  - **reference monitors**, secure hardware, static analysis, software rewriting, randomization, ...
- **Different proof techniques**
  - (bi)**simulation**, logical relations, multi-language semantics, embedded interpreters, ...