

# SECOMP

## Efficient Formally Secure Compilers to a Tagged Architecture

Cătălin Hrițcu  
INRIA Paris



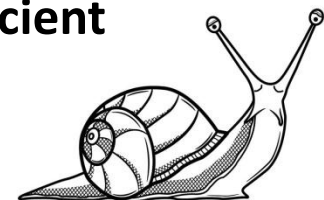
European Research Council  
**fresh grant**



**5 year vision**

# The problem: devastating low-level attacks

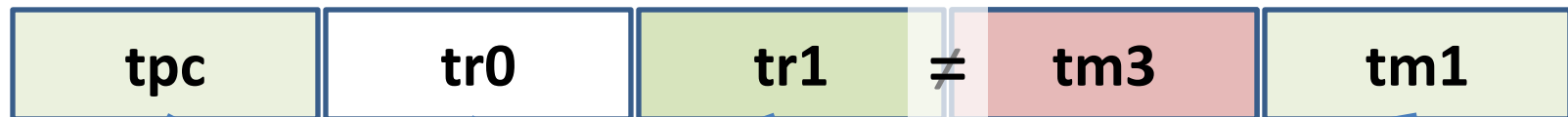
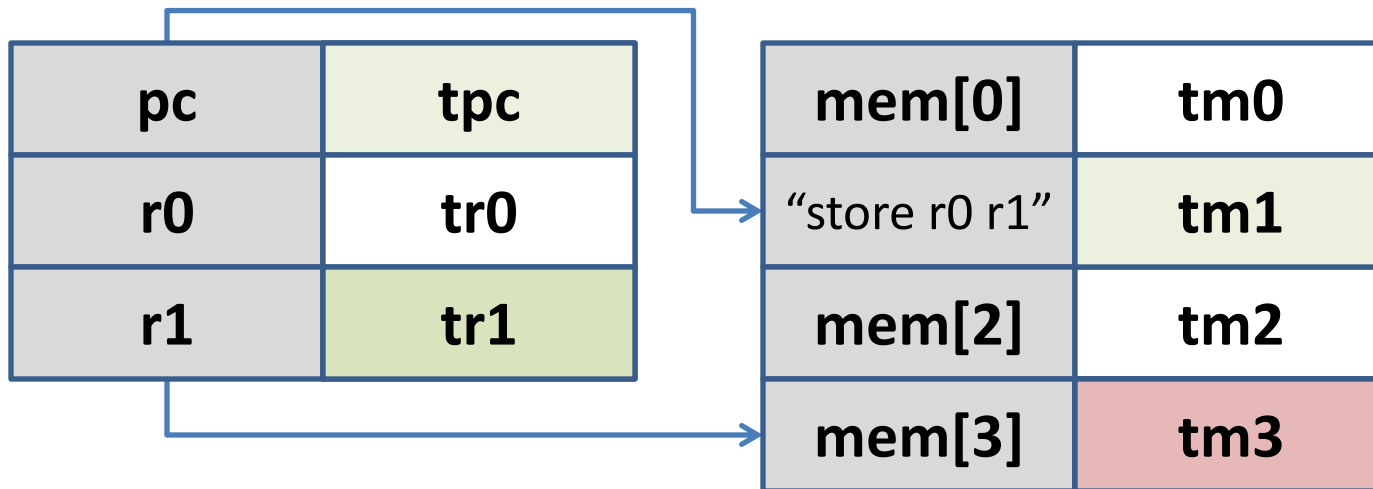
- **1. inherently insecure low-level languages (C, C++)**
  - **memory unsafe**: any buffer overflow can be catastrophic allowing remote attackers to gain complete control
- **2. unsafe interoperability with lower-level code**
  - even code written in **safer high-level languages (Java, C#, OCaml)** has to interoperate with **insecure low-level libraries (C, C++, ASM)**
  - **unsafe interoperability**: all high-level safety guarantees lost
- **Today's languages & compilers plagued by low-level attacks**
  - main culprit: **hardware** provides no appropriate security mechanisms
  - fixing this purely in software would be way **too inefficient**





# Key enabler: Micro-Policies

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



store



allow  
disallow



**policy violation stopped!**  
(e.g. out of bounds write)

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**: hardware caching, <10% overhead
  - heap safety, control-flow integrity, taint tracking
- **expressive**: complex policies for secure compilation
- **secure** and **simple** enough to verify security in Coq
- **real**: FPGA implementation on top of RISC-V



**DRAPER**  
**bluespec**<sup>®</sup>

[Oakland '13 & '15, POPL '14, ASPLOS '15]

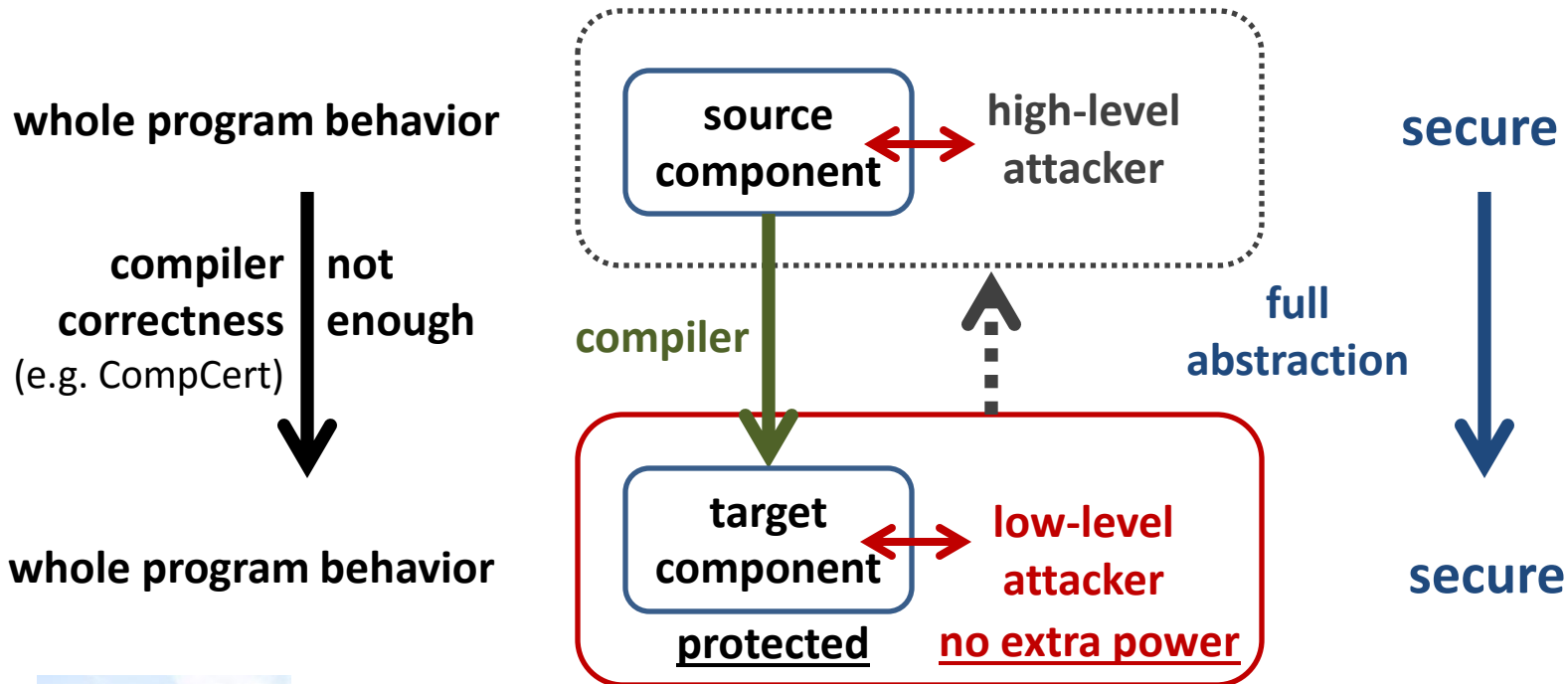
# SECOMP grand challenge

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 F\* [F\* = ML + verification]

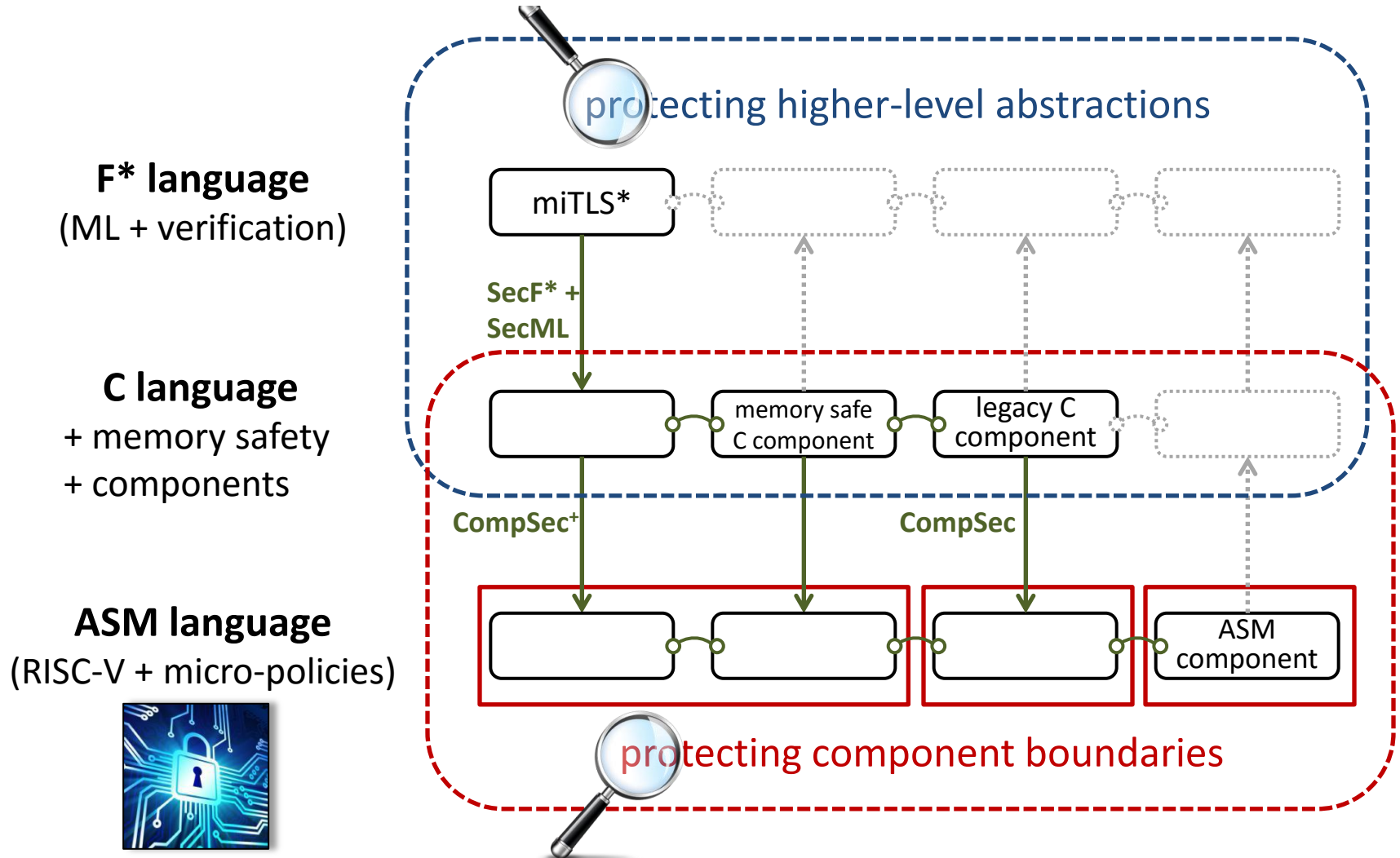
# Formally verify: full abstraction

holy grail of secure compilation, enforcing abstractions 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

# SECOMP: achieving full abstraction at scale





# Protecting component boundaries

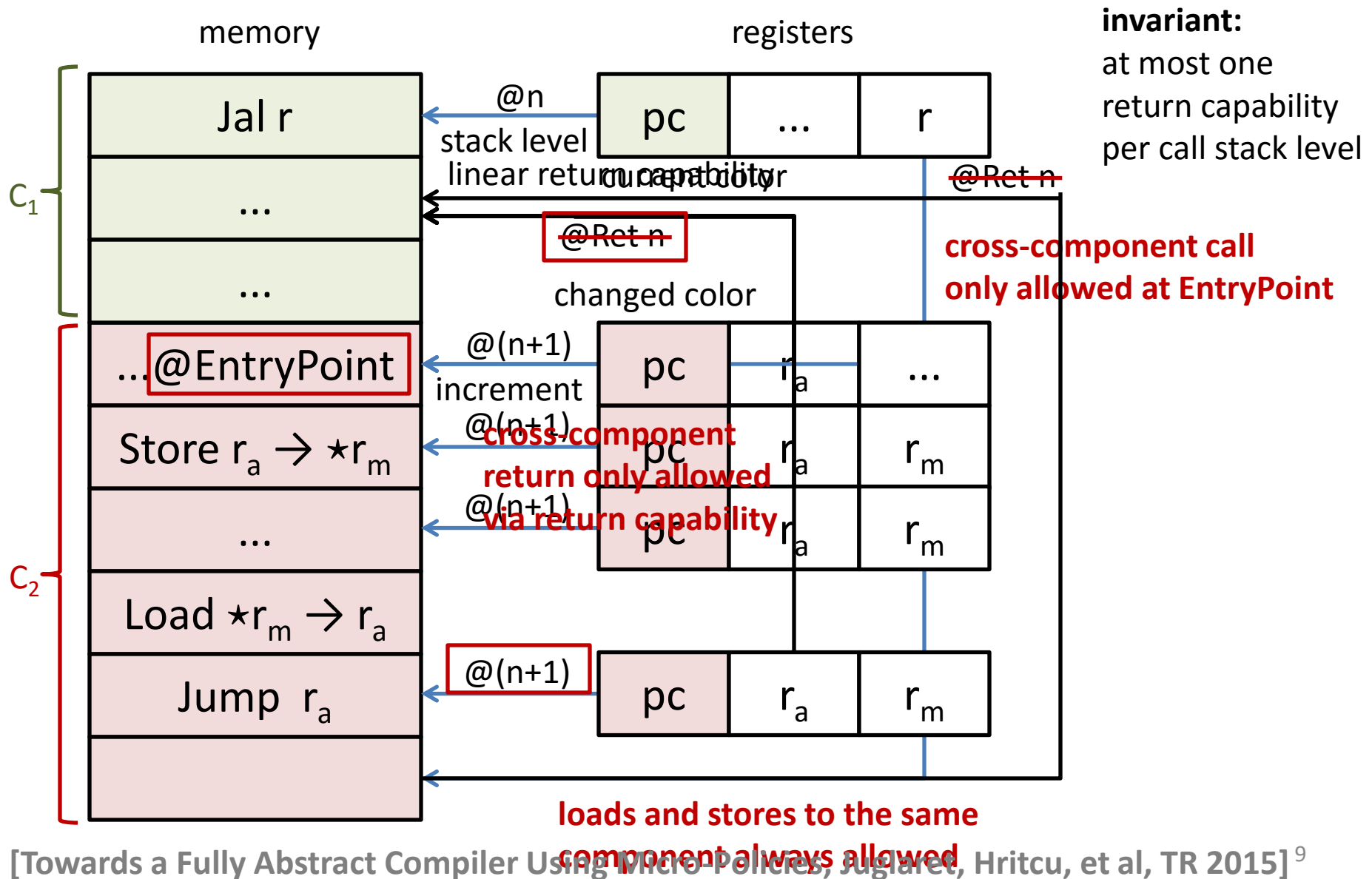
- **Add mutually distrustful components to C**
  - interacting only via **strictly enforced interfaces**
- **CompSec compiler chain** (based on CompCert)
  - propagate interface information to produced binary
- **Micro-policy simultaneously enforcing**
  - component separation
  - type-safe procedure call and return discipline
- **Interesting attacker model**
  - extending full abs. to mutual distrust + unsafe source



**Recent preliminary work, joint with Yannis Juglaret et al**

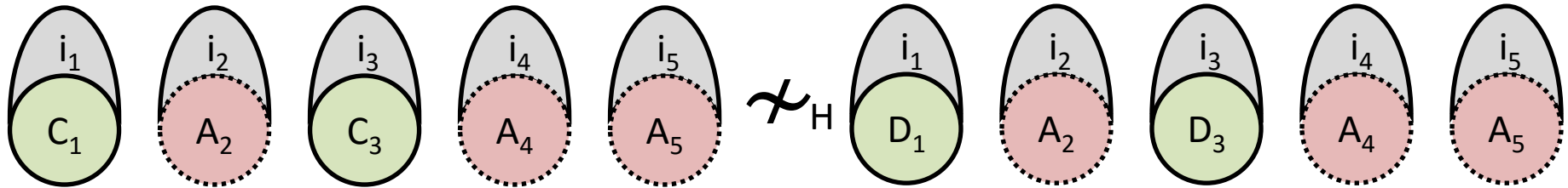


# Compartmentalization micro-policy

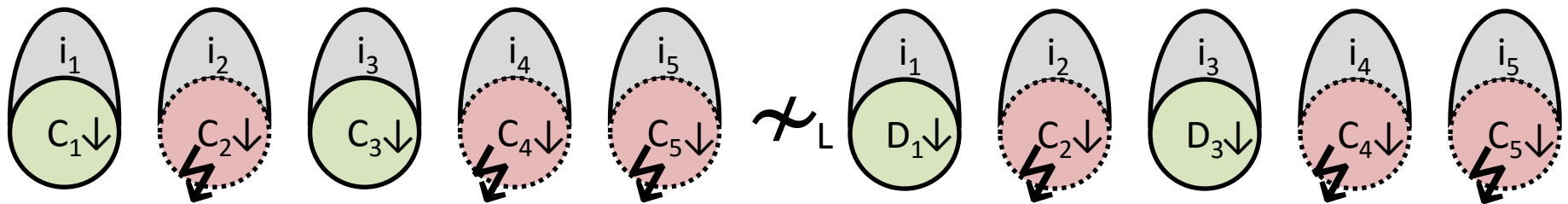


# Secure compartmentalizing compilation

$\forall$  compromise scenarios.



$\forall$  low-level attack from compromised C<sub>2</sub>↓, C<sub>4</sub>↓, C<sub>5</sub>↓  
 $\exists$  high-level attack from some fully defined A<sub>2</sub>, A<sub>4</sub>, A<sub>5</sub>



follows from “structured full abstraction  
for unsafe languages” + “separate compilation”

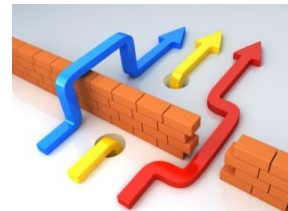
[Beyond Good and Evil, Juglaret, Hritcu, et al, CSF'16]



# Protecting higher-level abstractions



- **ML abstractions we want to enforce with micro-policies**
  - types, value immutability, opaqueness of closures, parametricity (dynamic sealing), GC vs malloc/free, ...
- **F\*: enforcing full specifications using micro-policies**
  - some can be turned into **contracts**, checked dynamically
  - fully abstract compilation of F\* to ML **trivial for ML interfaces** (because F\* allows and tracks effects, as opposed to Coq)
- **Limits of purely-dynamic enforcement**
  - functional purity, termination, relational reasoning
  - **push these limits further and combine with static analysis**



# Micro-policies:

## remaining fundamental challenges

- **Micro-policies for C and ML**
  - 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, ML, F\*)
- **Answering challenging fundamental questions**
  - attacker models, composition, micro-policies for C and ML
- **Achieving strong security properties like full abstraction**
  - + 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 ...
  - trying to build a community and looking for collaborators & students & PostDocs to try to make some of this real

