

# Secure Compilation

an extensive introduction

---

Marco Patrignani

9<sup>th</sup> October 2017



Center for IT-Security, Privacy  
and Accountability



KU LEUVEN

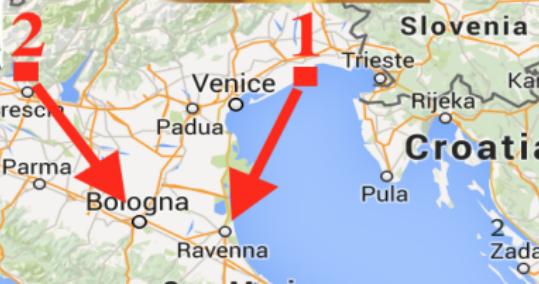


CISPA-Stanford Center

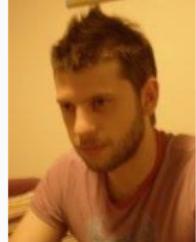
FOR CYBERSECURITY



ALMA MATER STUDIORUM  
UNIVERSITÀ DI BOLOGNA



# Collaborators



# Contents

What is Secure Compilation?

Secure Compilation Criteria

Programming Languages Techniques for Secure  
Compilation

Security Architectures for Secure Compilation

# **What is Secure Compilation?**

---

# Compilation

# Compilation

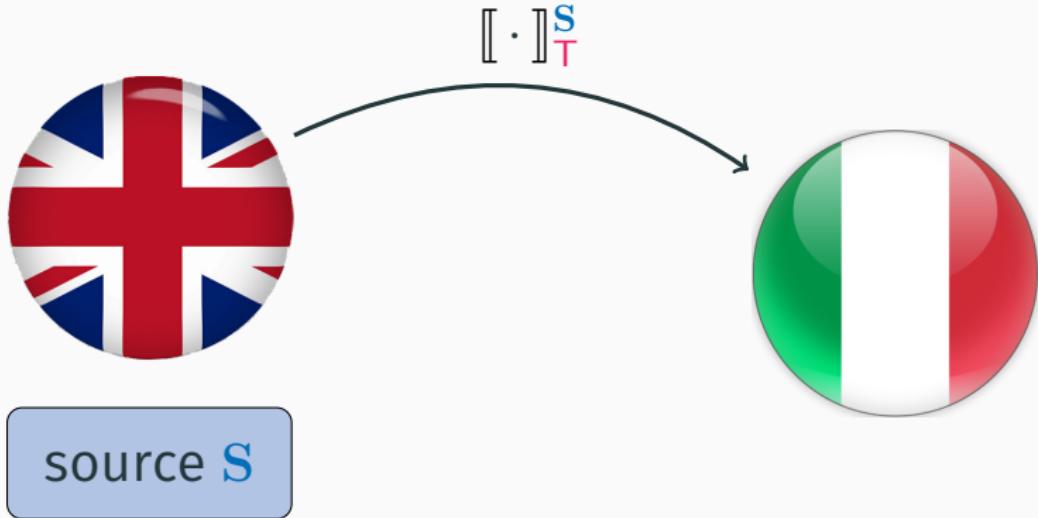


# Compilation

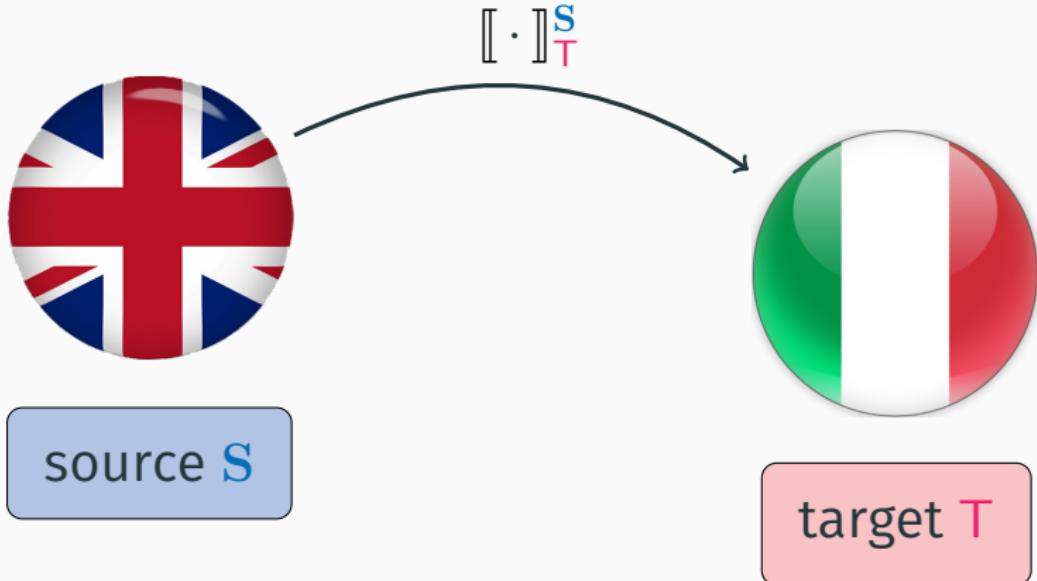


source S

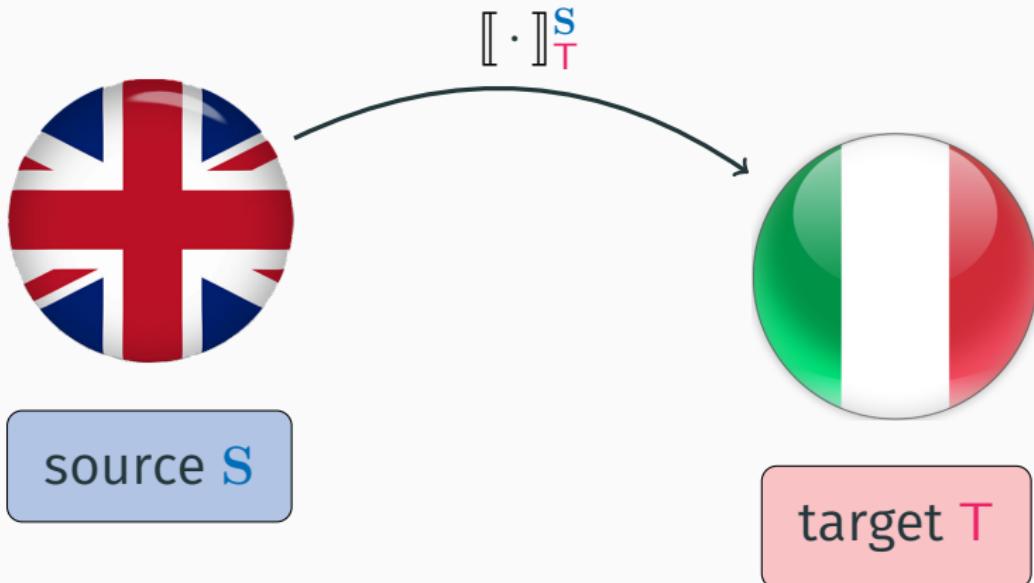
# Compilation



# Compilation



# Correct Compilation



# Correct Compilation



[[ · ]]  
S  
T



# Correct Compilation



[[ · ]]  
S  
T



# Correct Compilation



[[ · ]]  
S  
T

A curved arrow points from the British flag towards the Italian flag, with the text "[[ · ]] S T" positioned above it.

# Correct Compilation



[[ · ]]  
**S**  
**T**



# Correct Compilation



[[ · ]]  
S  
T

A curved black arrow points from the British flag towards the Italian flag.

# Secure Compilation



[[ · ]]  
S  
T



# Secure Compilation



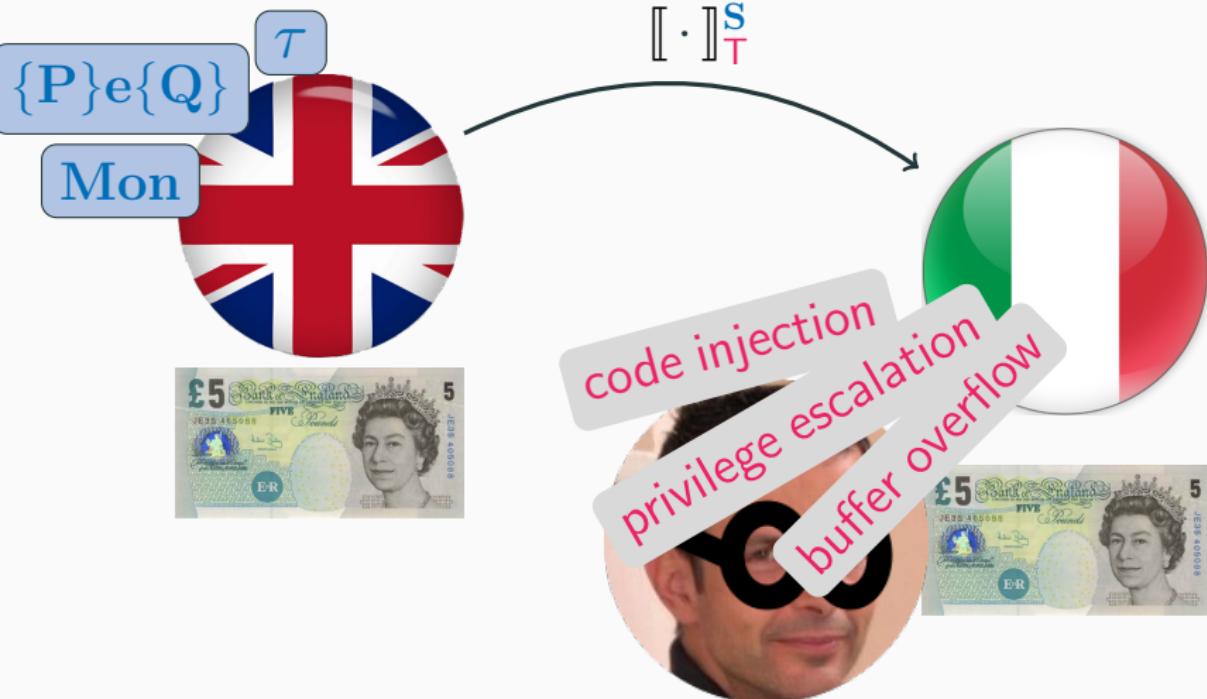
$\ll \cdot \gg^{\textcolor{blue}{S}}_{\textcolor{pink}{T}}$



code injection  
privilege escalation  
buffer overflow



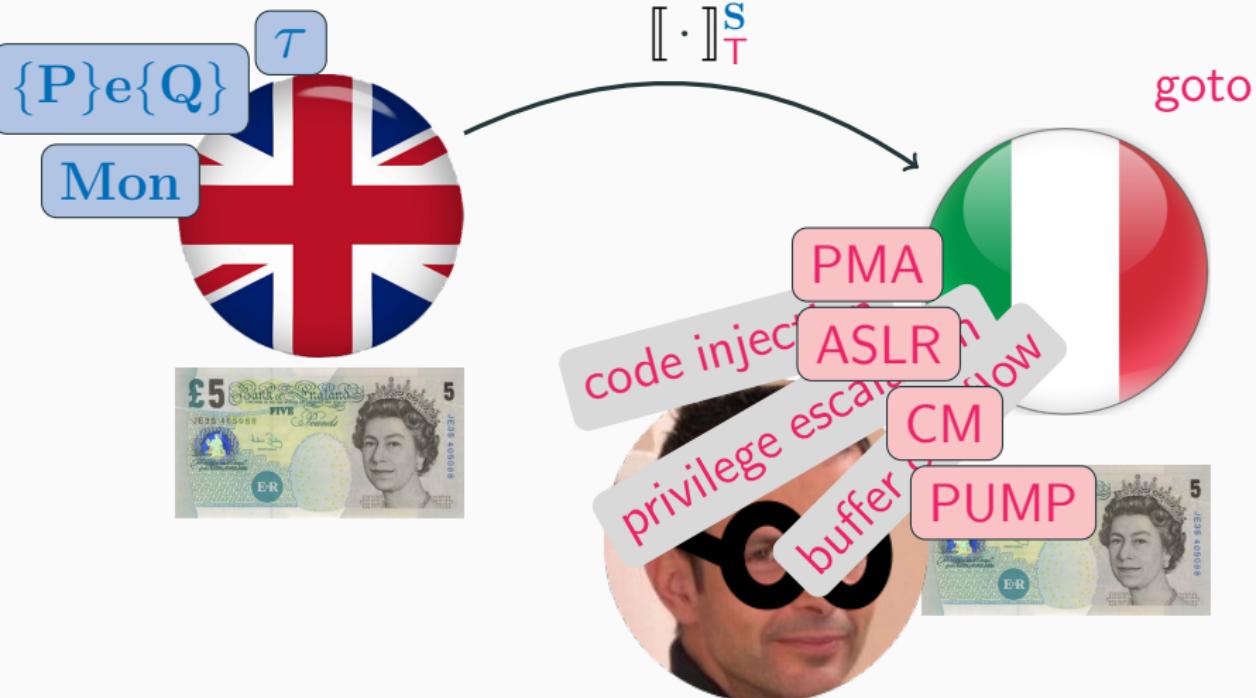
# Secure Compilation



# Secure Compilation

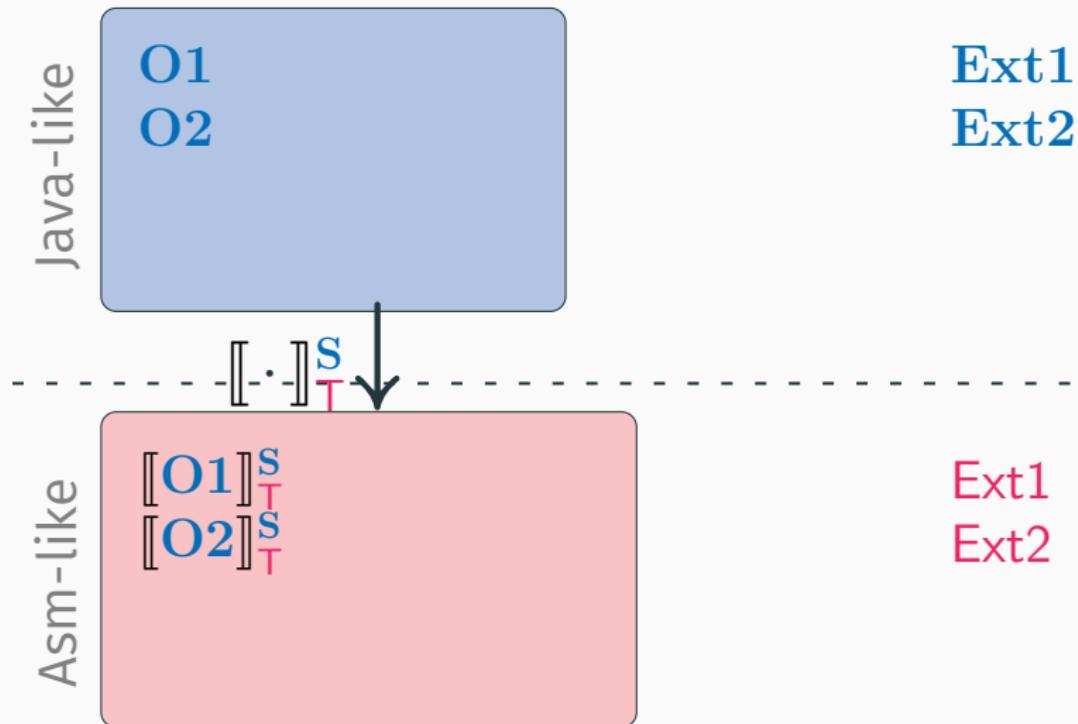


# Secure Compilation



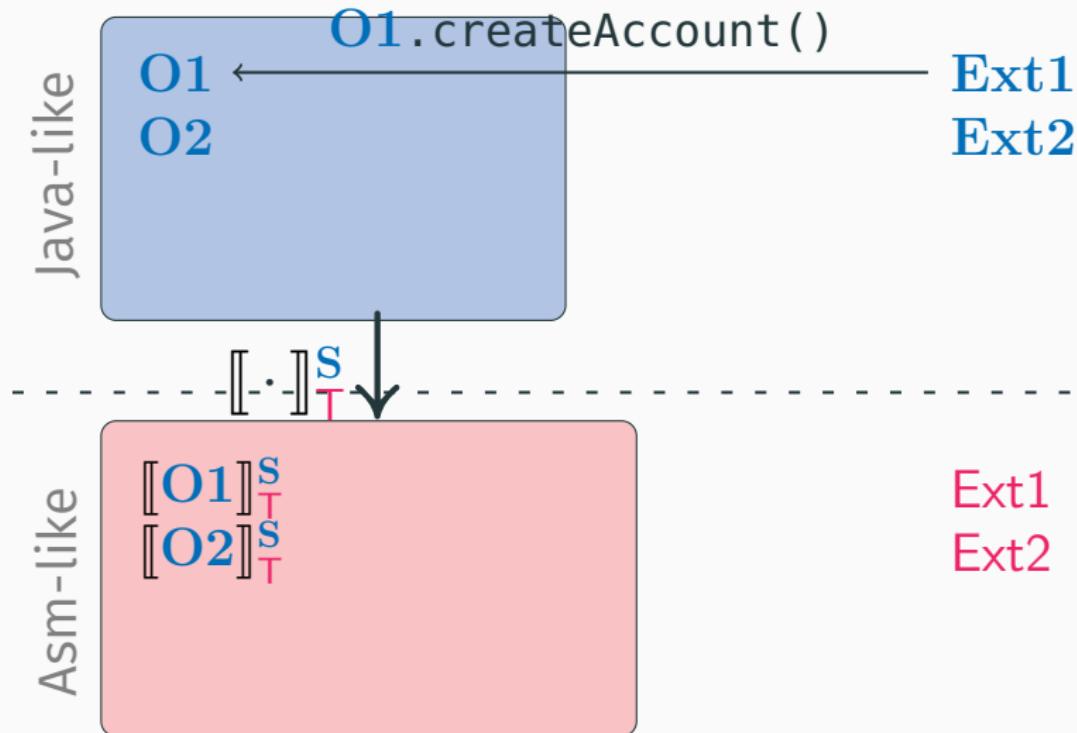
# Memory Allocation Issues

Patrignani et al.'15'16



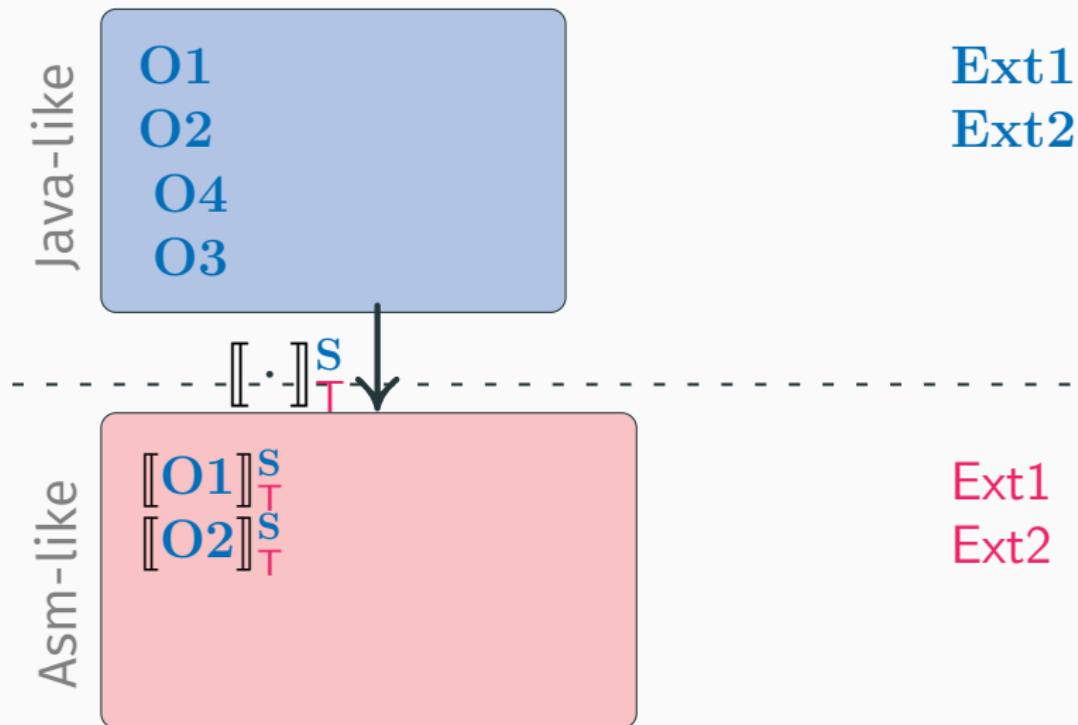
# Memory Allocation Issues

Patrignani et al.'15'16



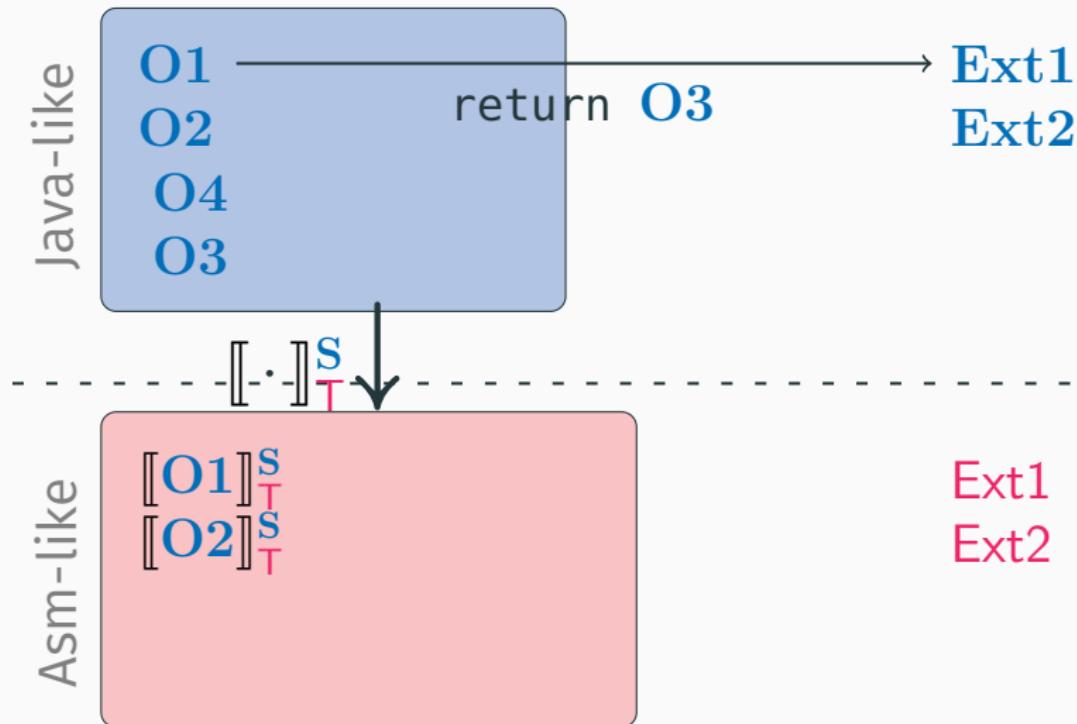
# Memory Allocation Issues

Patrignani et al.'15'16



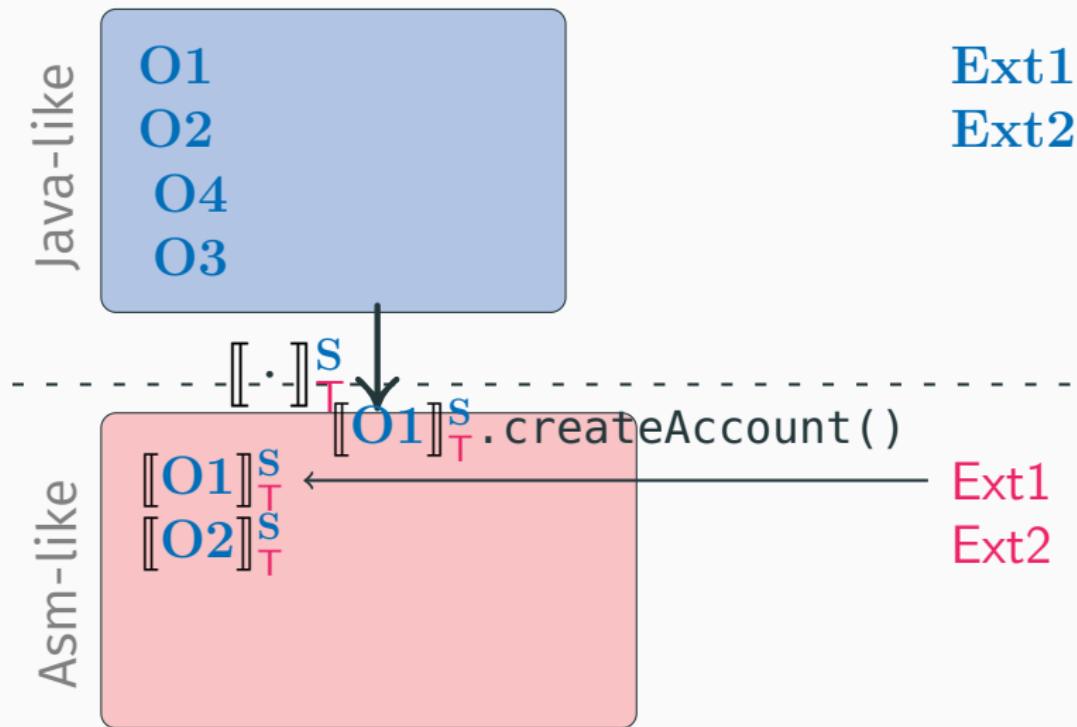
# Memory Allocation Issues

Patrignani et al.'15'16



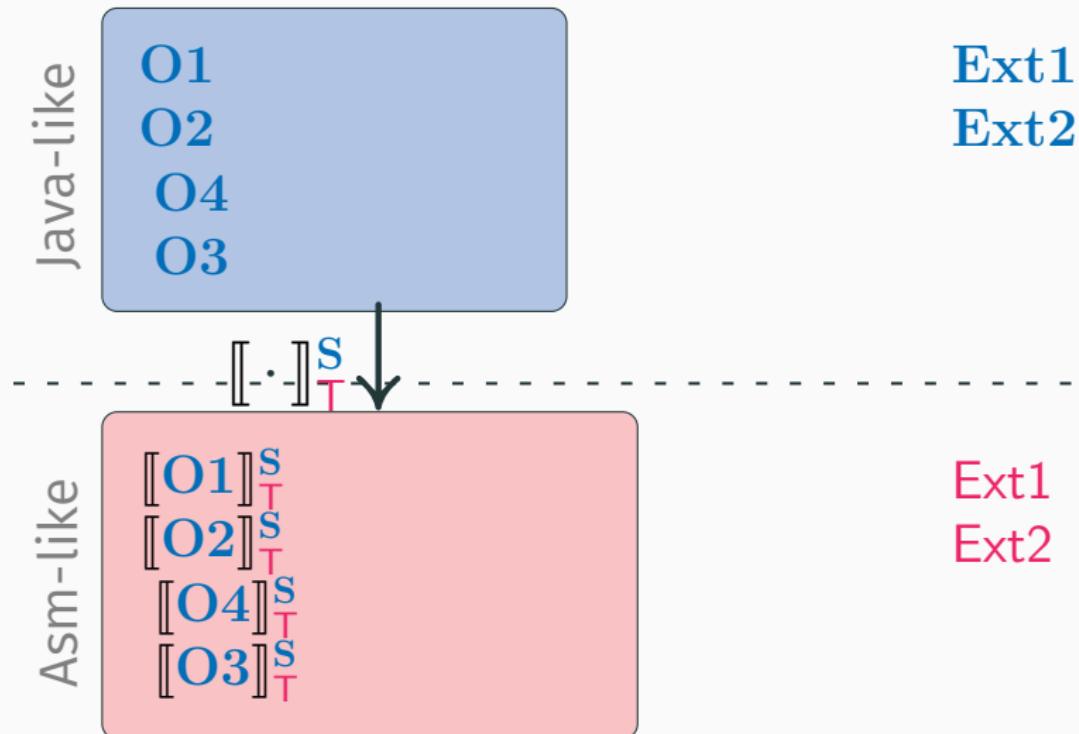
# Memory Allocation Issues

Patrignani et al.'15'16



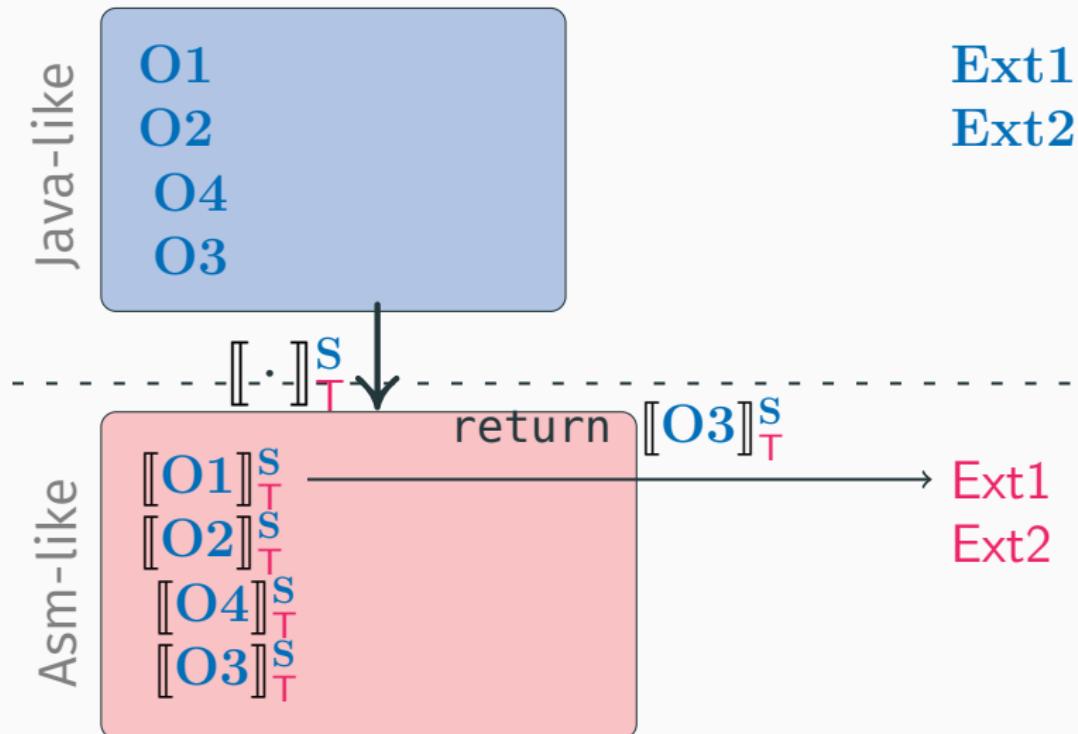
# Memory Allocation Issues

Patrignani et al.'15'16



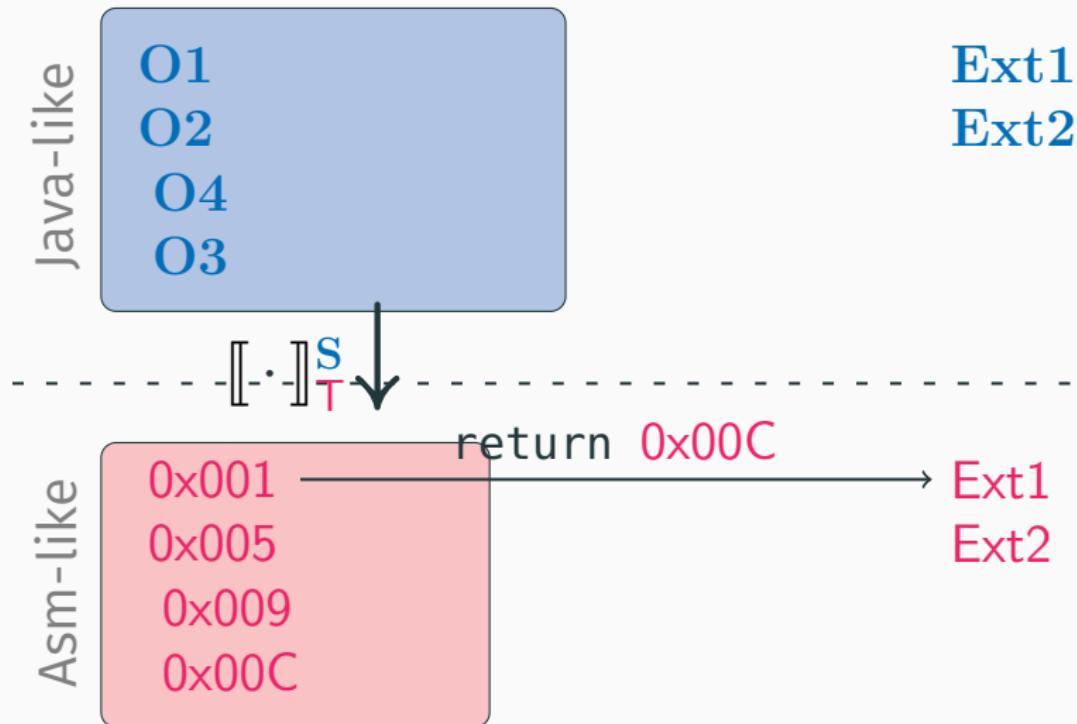
# Memory Allocation Issues

Patrignani et al.'15'16



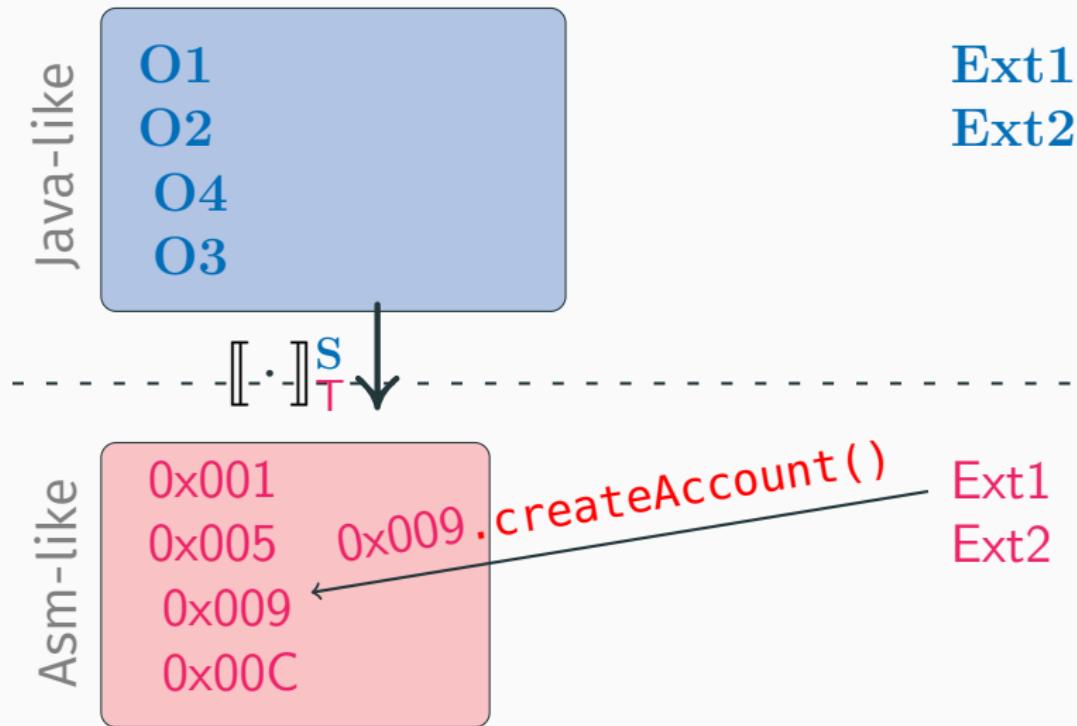
# Memory Allocation Issues

Patrignani et al.'15'16



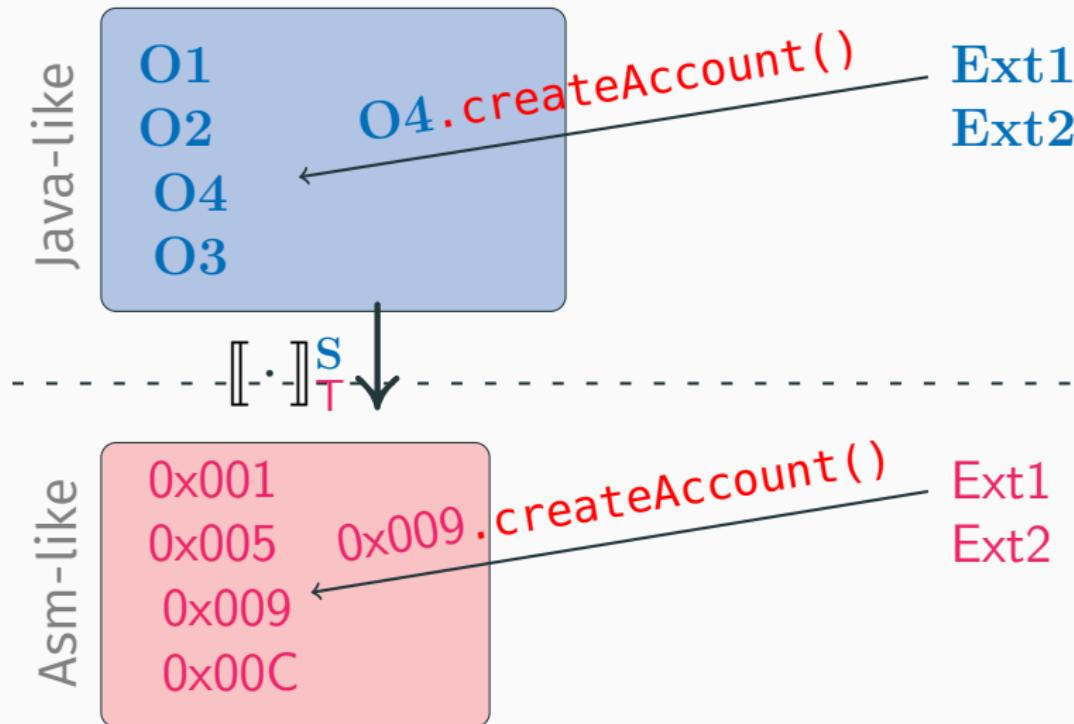
# Memory Allocation Issues

Patrignani et al.'15'16



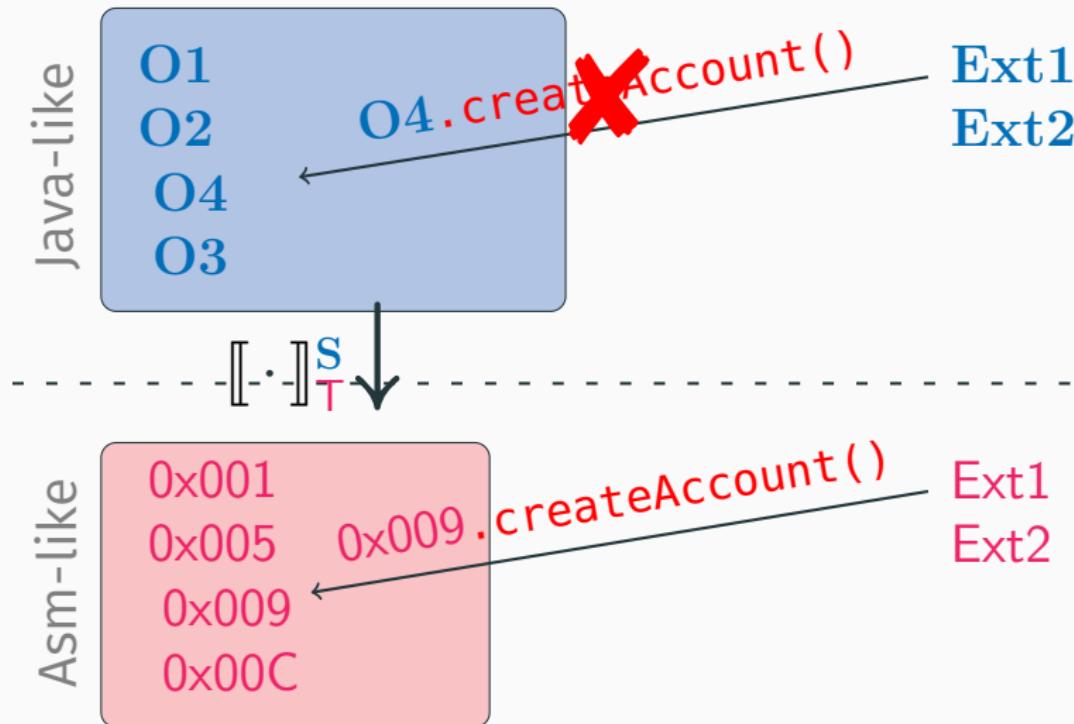
# Memory Allocation Issues

Patrignani et al.'15'16



# Memory Allocation Issues

Patrignani et al.'15'16



Issue: Oid guessing  
**Solution:** keep a map  
from Oid to random  
numbers

Asm-like

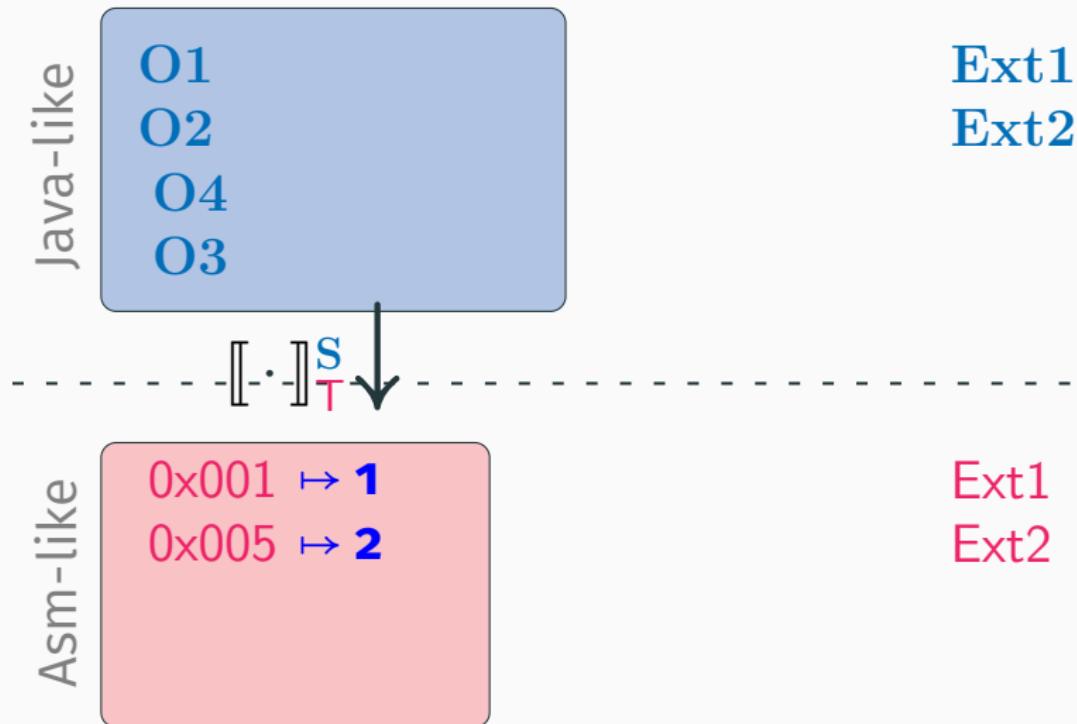
0x005

Ext1  
Ext2

Ext1  
Ext2

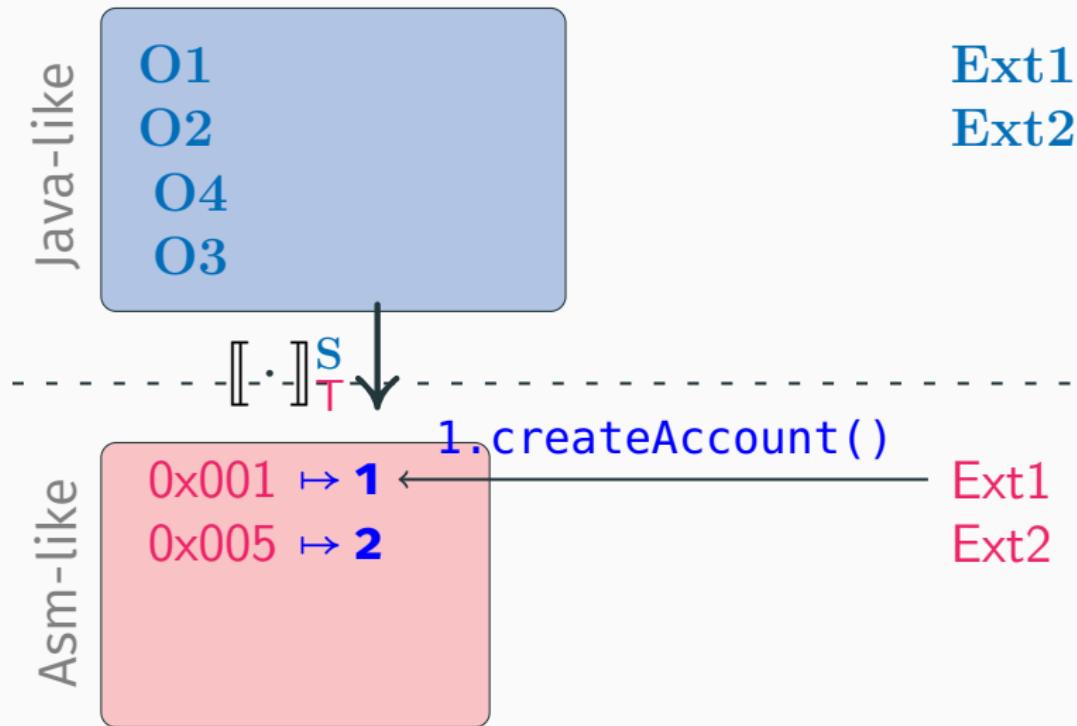
# Memory Allocation Issues

Patrignani et al.'15'16



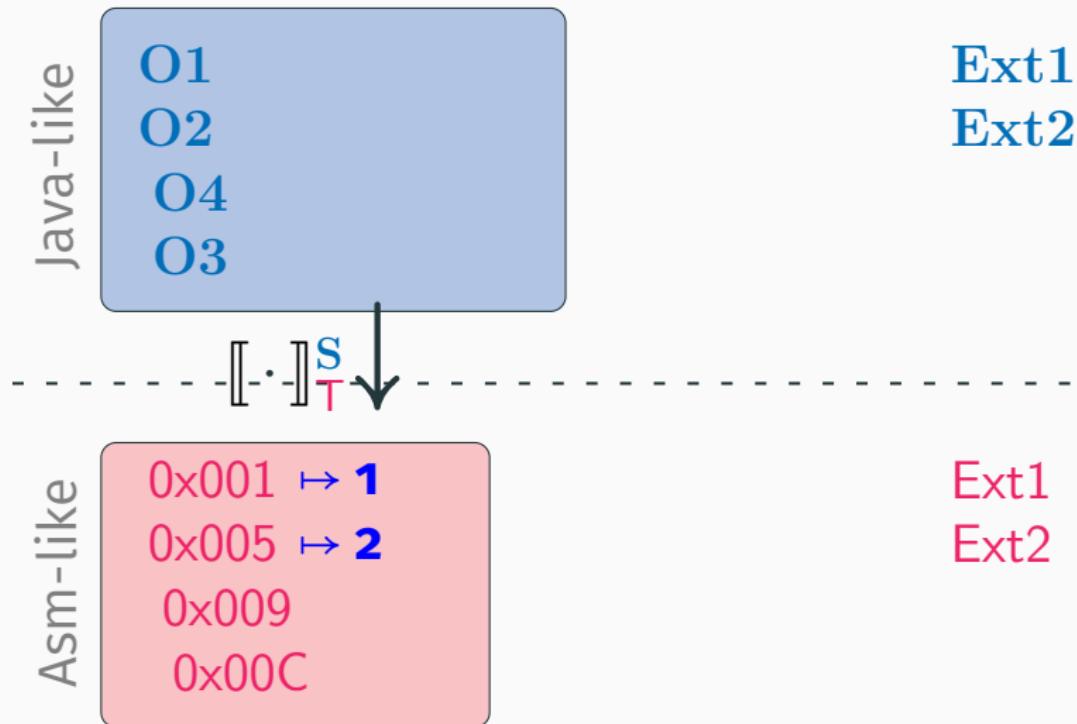
# Memory Allocation Issues

Patrignani et al.'15'16



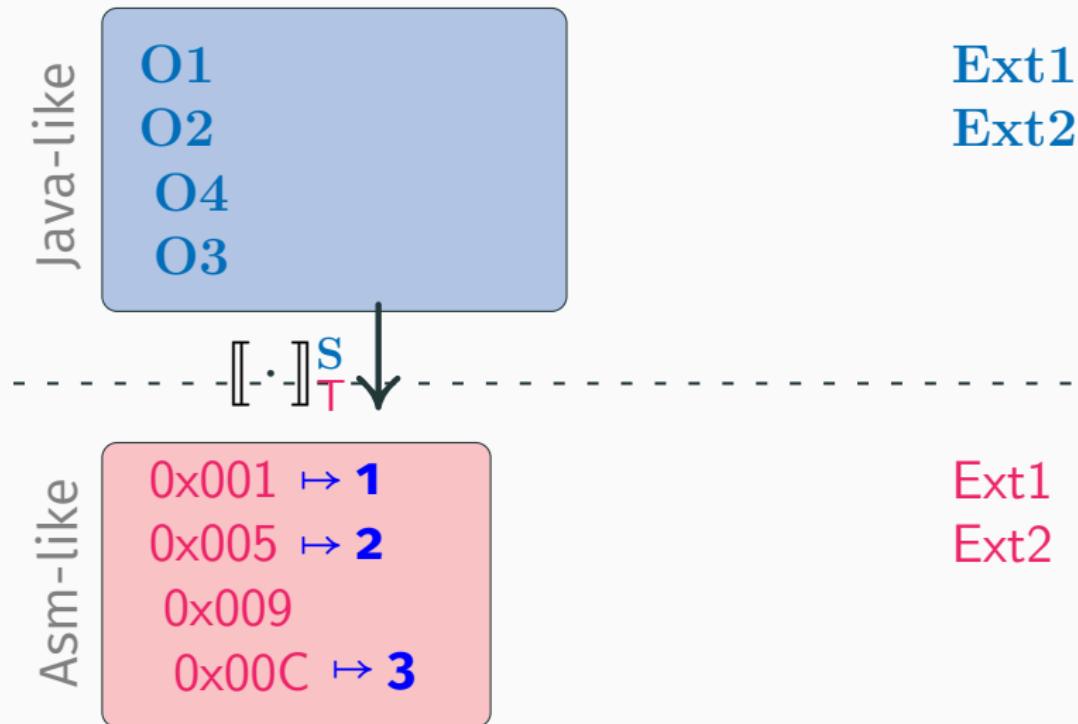
# Memory Allocation Issues

Patrignani et al.'15'16



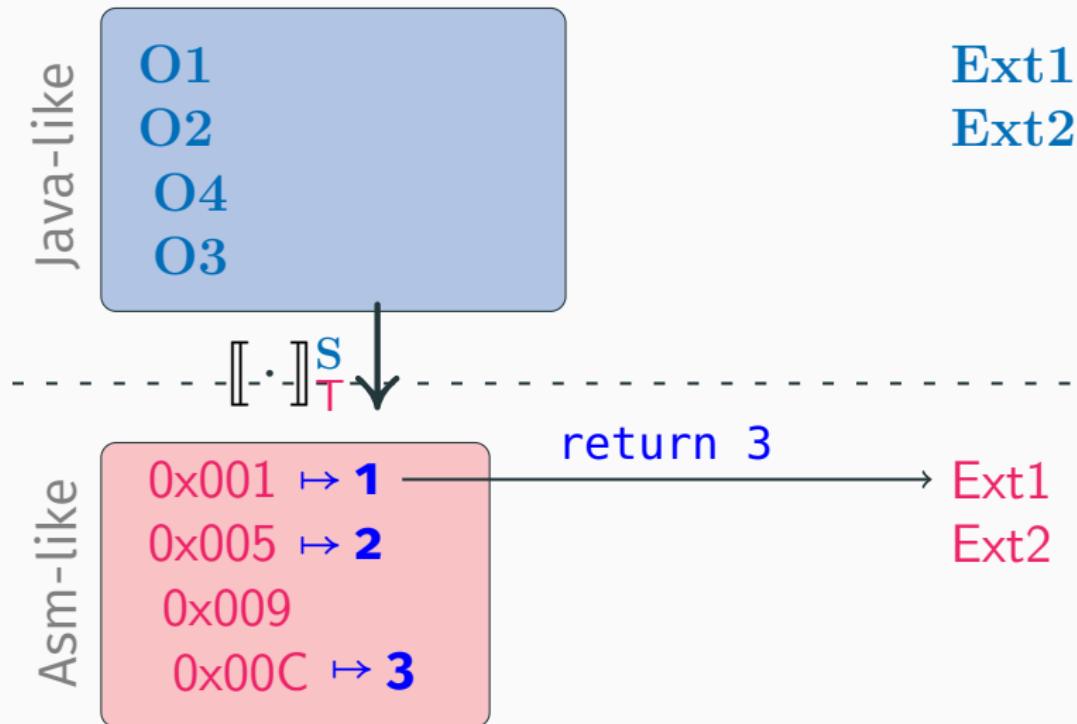
# Memory Allocation Issues

Patrignani et al.'15'16



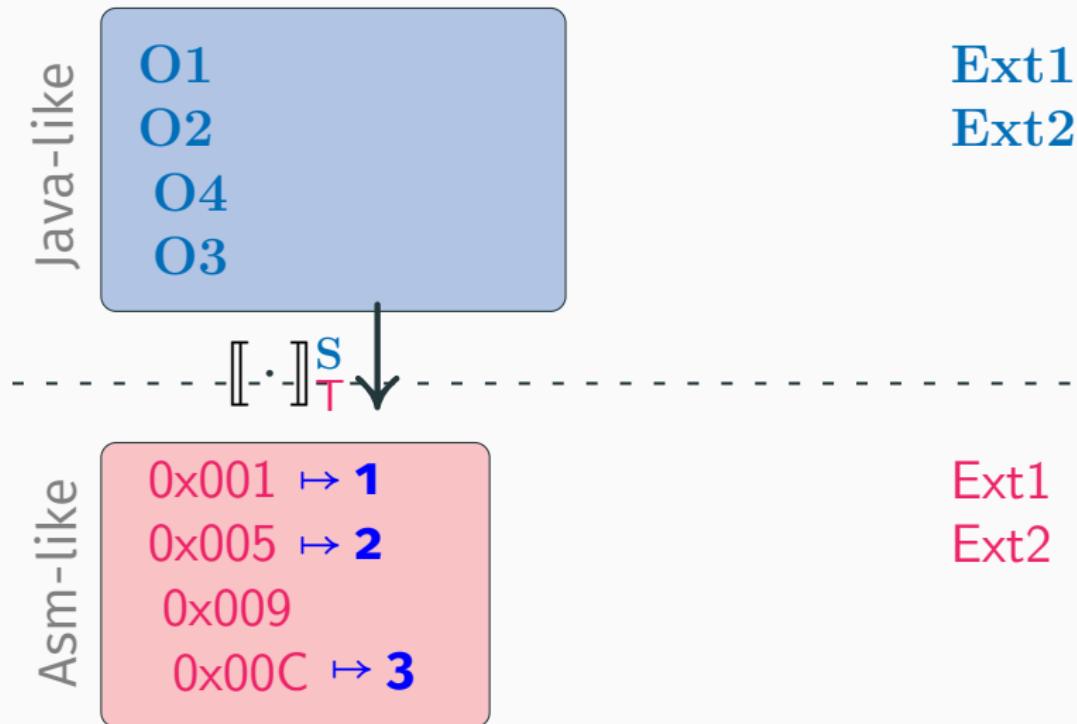
# Memory Allocation Issues

Patrignani et al.'15'16



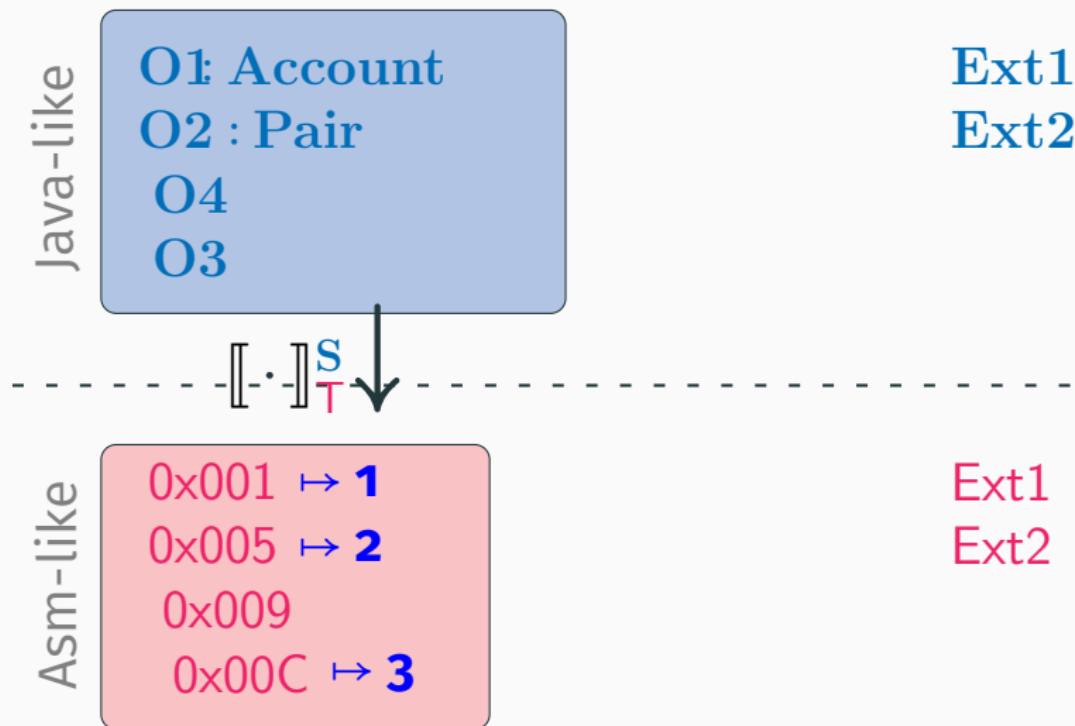
# Memory Allocation Issues

Patrignani et al.'15'16



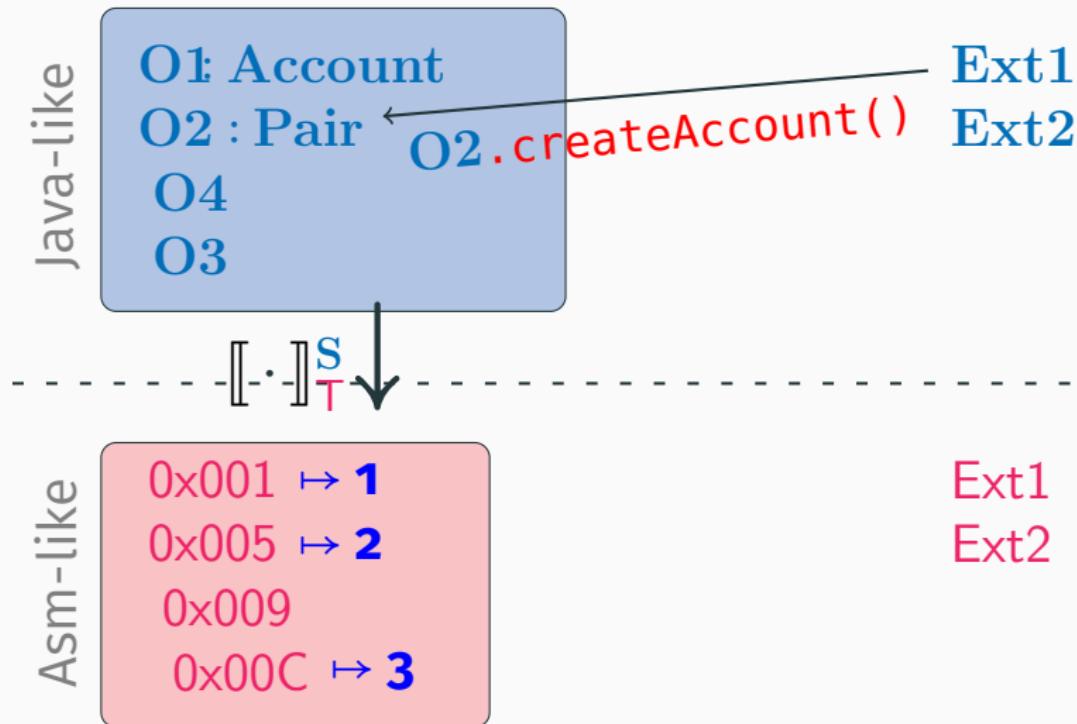
# Memory Allocation Issues

Patrignani et al.'15'16



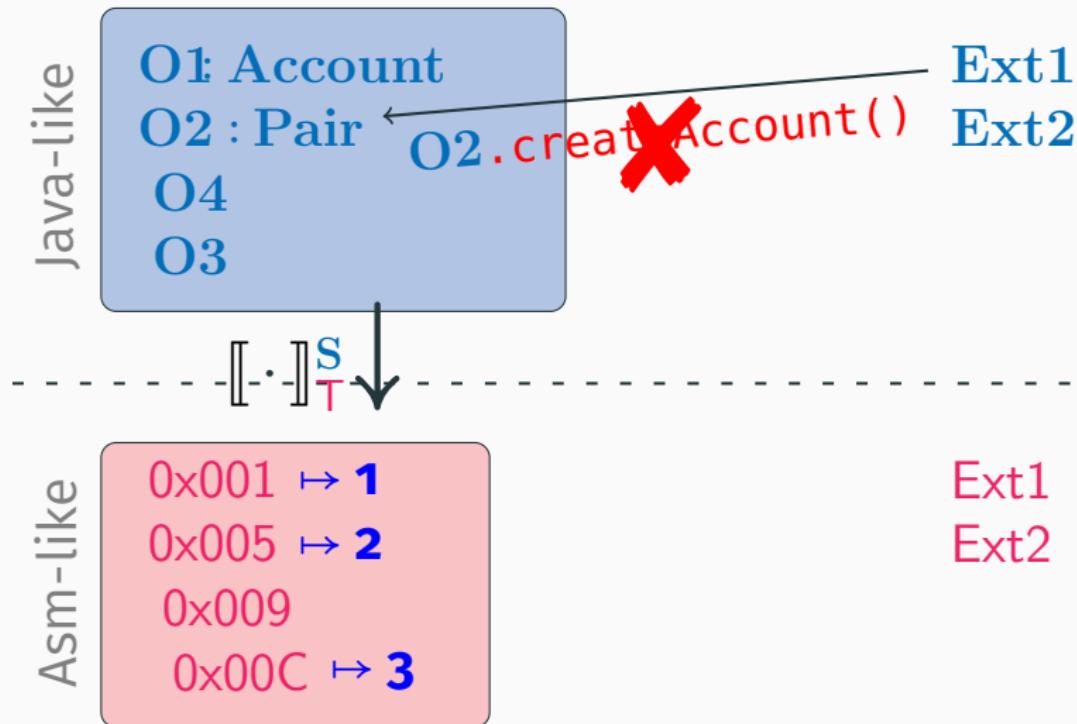
# Memory Allocation Issues

Patrignani et al.'15'16



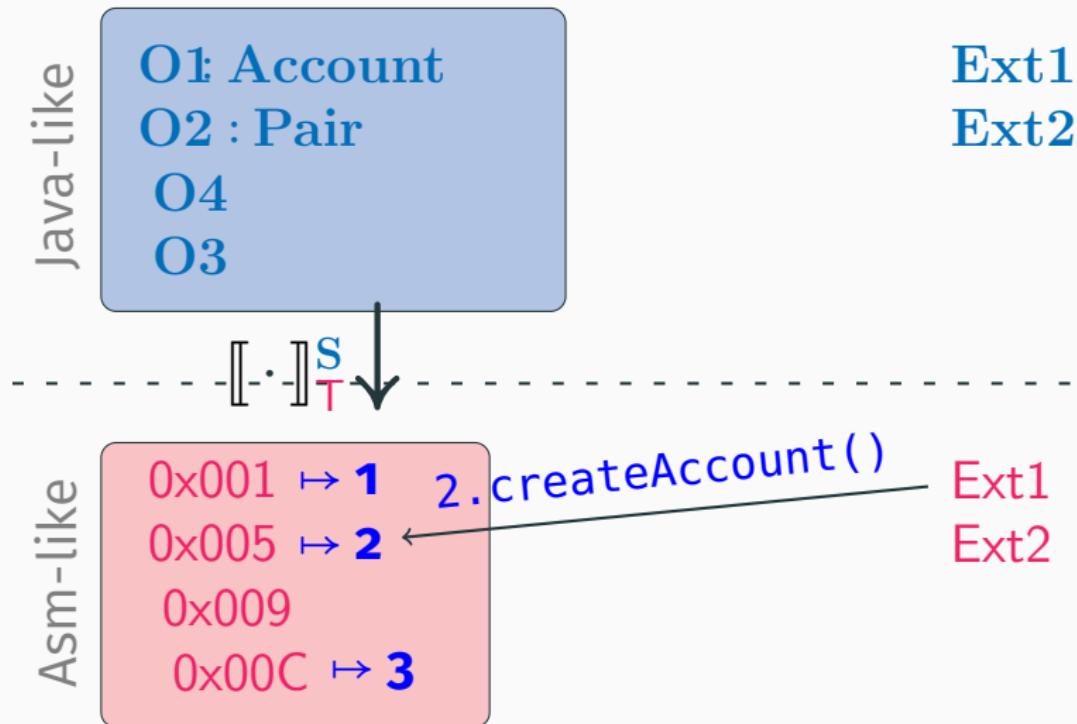
# Memory Allocation Issues

Patrignani et al.'15'16



# Memory Allocation Issues

Patrignani et al.'15'16



e

O1: Account

Issue: type violation  
Solution: add dynamic typechecks

Asm-like

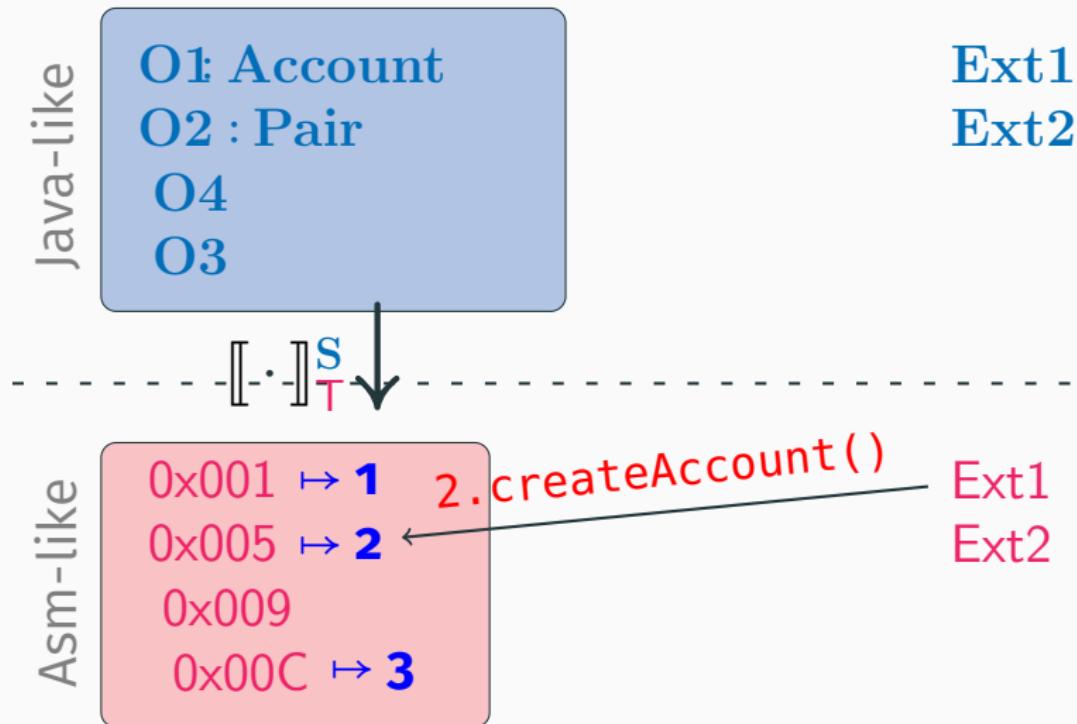
0x001 → 1  
0x005 → 2  
0x009  
0x00C → 3

Ext1  
Ext2

Ext1  
Ext2

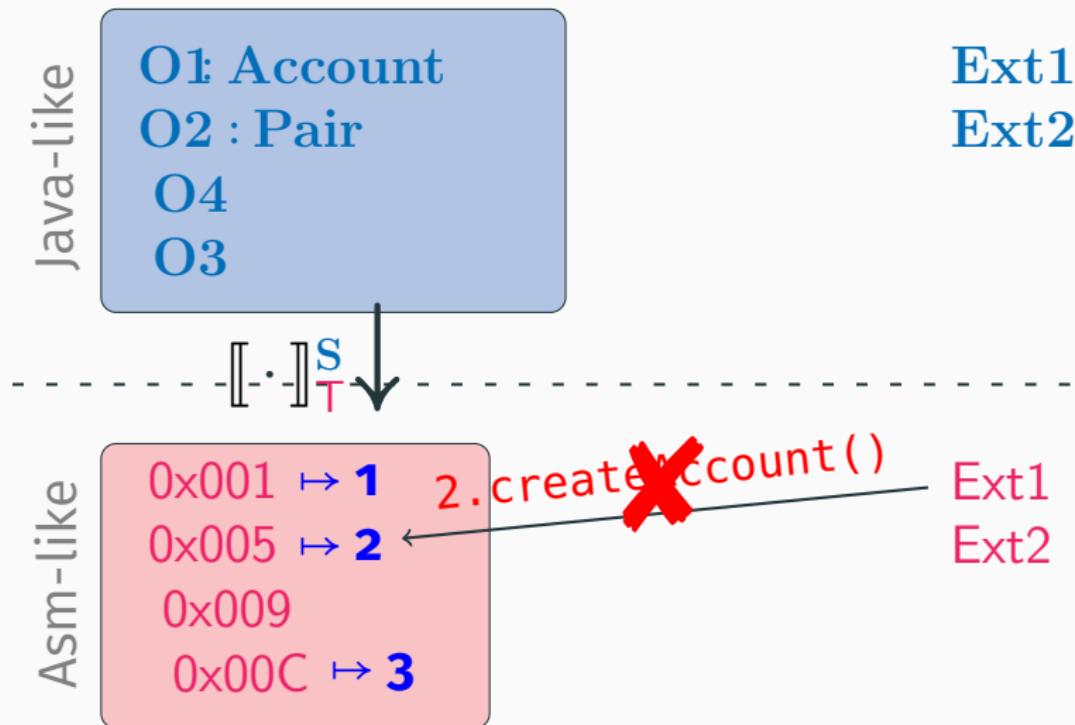
# Memory Allocation Issues

Patrignani et al.'15'16



# Memory Allocation Issues

Patrignani et al.'15'16



# Memory Allocation Issues

Patrignani et al.'15'16

like

O1: Account  
O2 : Pair

Ext1  
Ext2

Isolated memory regions  
e.g., SGX enclaves

Asm-like

0x001  $\mapsto$  1  
0x005  $\mapsto$  2  
0x009  
0x00C  $\mapsto$  3

Ext1  
Ext2

# Memory Allocation Issues

Patrignani et al.'15'16

- design
- implement



Asm-[

0x009

0x00C  $\mapsto$  3

Ext1  
Ext2

Ext1  
Ext2

# Memory Allocation Issues

Patrignani et al.'15'16

- design
- implement



- ...or?

Asm-[  
  0x0000 ... -

0x009

0x00C  $\mapsto$  3

Ext1  
Ext2

Ext1  
Ext2

# Guarantees

- How do we know we are right?

# Guarantees

- How do we know we are right?
- How can we know that  $\llbracket \cdot \rrbracket_T^S$  is **secure**?

# Guarantees

Prove  
 $\llbracket \cdot \rrbracket_T^S$  to attain a secure  
compilation criterion

# Guarantees

- How do we know we are right?
- How can we know that  $\llbracket \cdot \rrbracket_T^S$  is **secure**?
- What do we mean with **secure**?

# Guarantees

Show  
the security implications  
of the criterion

# **Secure Compilation Criteria**

---

# The Origins of the Secure Compiler

## Secure Implementation of Channel Abstractions

Martín Abadi

[ma@pa.dec.com](mailto:ma@pa.dec.com)

Digital Equipment Corporation  
Systems Research Center

Cédric Fournet

[Cedric.Fournet@inria.fr](mailto:Cedric.Fournet@inria.fr)

INRIA Rocquencourt

Georges Gonthier

[Georges.Gonthier@inria.fr](mailto:Georges.Gonthier@inria.fr)

INRIA Rocquencourt

### Abstract

*Communication in distributed systems often relies on useful abstractions such as channels, remote procedure calls, and remote method invocations. The implementations of these abstractions sometimes provide security properties, in particular through encryption. In this*

spaces are on the same machine, and that a centralized operating system provides security for them. In reality, these address spaces could be spread across a network, and security could depend on several local operating systems and on cryptographic protocols across machines.

For example, when an application requires secure

**Theorem 1** *The compositional translation is fully-abstract, up to observational equivalence: for all join-calculus processes  $P$  and  $Q$ ,*

$$P \approx Q \quad \text{if and only if} \quad \mathcal{E}\text{nv}[\llbracket P \rrbracket] \approx \mathcal{E}\text{nv}[\llbracket Q \rrbracket]$$

From the join-calculus to  
the sjoin-calculus

# The Origins of the Secure Compiler

they needed a definition that their implementation of **secure channels** via **cryptography** was secure

# The Origins of the Secure Compiler

The main question they had (and we still have):

what are good **correctness criteria** for secure compilers?

# The Origins of the Secure Compiler

## Fully Abstract Compilation (FAC)

**Theorem 1** *The compositional translation is fully-abstract, up to observational equivalence: for all join-calculus processes  $P$  and  $Q$ ,*

$$P \approx Q \quad \text{if and only if} \quad \mathcal{E}\text{nv}[\llbracket P \rrbracket] \approx \mathcal{E}\text{nv}[\llbracket Q \rrbracket]$$

what

compilers?

# Fully Abstract Compilation Influence

## Fully Abstract Compilation to JavaScript

J.-Chen<sup>1</sup>, Pierre-Evariste Dagand<sup>2</sup>, Pierre-Yves Strub<sup>1</sup>, Benj<sup>1</sup>  
and MSR-INRIA<sup>1</sup>

strath.ac.uk pierre-yves@stru

## Secure Implementations for Typed Session Abstraction

Ricardo Corin<sup>1,2,3</sup>, Pierre-Malo Deniéou<sup>1,2</sup>, Cédric Fournet<sup>1,2</sup>  
Karthikayan Bhargavan<sup>1,2</sup>, James Leifer<sup>1</sup>

1 MSR-INRIA Joint Centre    2 Microsoft Research    3 University of Toulouse

Amal Ahmed<sup>1</sup>, Matthias Blume<sup>2</sup>  
Toyota Technological Institute at Chicago  
(amal.blume)@ttic.org

Authentication primitives and their compilation

Martín Abadi<sup>\*</sup>  
Bell Labs Research  
Lucent Technologies

Cédric Fournet  
Microsoft Research

Georges G. J. P. Plotkin<sup>†</sup>  
INRIA Rocquencourt

On Protection by Layout Randomization  
MARTÍN ABADI, Microsoft Research, Silicon Valley  
Santa Cruz, Collège de France  
GORDON D. PLOTKIN,<sup>†</sup>  
University of Edinburgh

## Secure Compilation of Object-Oriented Components to Protected Module Architectures

Marco Patrignani, Dave Clarke, and Frank Piessens<sup>\*</sup>

iMinds-DistriNet, Dept. Computer Science  
{first.last}@iminds.be

## Beyond Good and Evil

Formalizing the Security Guarantees of Compartmentalizing Compilation  
Yannis Juglarel<sup>1,2</sup>, Cătălin Hritcu<sup>1</sup>, Arthur Azevedo de Amorim<sup>3</sup>, Boris Eng<sup>1,3</sup>, Benjamin C. Pierce<sup>4</sup>  
<sup>1</sup>Inria Paris, <sup>2</sup>Université Paris Diderot (Paris 7), <sup>3</sup>Université Paris 8, <sup>4</sup>University of Pennsylvania

## A Secure Compiler for ML Modules

An Equivalence-Preserving CPS Translation  
via Multi-Language Semantics<sup>\*</sup>

On Modular and Fully-Abstract Compilers<sup>11</sup>  
Amal Ahmed  
Julian Rathke  
Corin Pitcher  
University of Southampton  
James Riely  
Johns Hopkins University  
Marco Patrignani  
MPI-Saarland  
Marco Patrignani  
Marco Patrignani  
Matthias Blume  
Google  
blume@google.com

Marco Patrignani  
Dept. Computer Science  
and Dave Clarke

## Fully Abstract Compilation via Universal Embedding\*

Secure Compilation to Protected Module Architectures  
Marco Patrignani<sup>\*</sup>, Raoul Strackx<sup>†</sup> and Bart Jacobs<sup>‡</sup>  
and iMinds-DistriNet<sup>\*</sup>

Marco Patrignani  
Dept. Computer Science  
and Dave Clarke

# Fully Abstract Compilation Influence

## Fully Abstract Compilation to JavaScript

J.-Chen<sup>1</sup> Pierre-Evariste Dagand<sup>2</sup> Pierre-Yves Strub<sup>1</sup> Benj<sup>3</sup>  
<sup>1,2</sup> MSR-INRIA<sup>1</sup>

## Secure Implementations for Typed Session Abstraction

Ricardo Corin<sup>1,2,3</sup> Pierre-Malo Deniéou<sup>1,2</sup> Cédric Fournet<sup>1,2</sup>  
Karthikeyan Bhargavan<sup>1,2</sup> James Leifer<sup>1</sup>

Why  
does Fully Abstract Compilation entail  
security?

Authentication

Martín Abadi<sup>\*</sup>  
Bell Labs Research  
Lucent Technologies

Secure Compilation  
of Object-Oriented Components  
to Protected Module Architectures

Marco Patrignani, Dave Clarke, and Frank Piessens

iMinds-DistriNet, Dept. Computer Science  
{first.last}@iminds.be

Formalizing the Security Guarantees  
Yannis Juglanet<sup>1,2</sup> Cătălin Hritcu<sup>1</sup> Arthur Azevedo de Amorim<sup>4</sup> Boris Féraud<sup>3</sup>  
<sup>1</sup>Inria Paris<sup>2</sup>Université Paris Diderot (Paris 7)<sup>3</sup>Université Paris 8<sup>4</sup>University of Pennsylvania

A Secure Compiler for ML Modules

•<sup>2</sup> and Dave Clarke  
An Equivalence-Preserving CPS Translation  
via Multi-Language Semantics<sup>\*</sup>

Fully Abstract Compilation via Universal Embedding\*

Marco Patrignani  
Dept. Computer Science  
and Dave Clarke

Julian Rathke  
University of Southampton  
Corin Pitcher  
University of Southampton

On Modular and Fully-Abstract Compilers  
Amal Ahmed  
Dominique Devriese  
Matthias Blume  
Google  
blume@google.com

Marco Patrignani  
Dept. Computer Science  
and Dave Clarke

Julian Rathke  
University of Southampton  
Corin Pitcher  
University of Southampton

On Modular and Fully-Abstract Compilers  
Amal Ahmed  
Dominique Devriese  
Matthias Blume  
Google  
blume@google.com

# Because

FAC ensures that a target-level attacker  
has the same power of a source-level  
one

# Compiler Full Abstraction



**x = 1;**

**x ++;**

**x**

**x = 0;**

**x+= 2;**

**x**

# Compiler Full Abstraction



$x = 1;$        $x = 0;$   
 $x ++;$        $= x += 2;$   
 $x$                  $x$

# Compiler Full Abstraction



$x = 1;$	$x = 0;$
$x ++;$	$= x += 2;$
$x$	$x$



loadi r <sub>0</sub> 1	loadi r <sub>0</sub> 0
inc r <sub>0</sub>	addi r <sub>0</sub> 2
ret r <sub>0</sub>	ret r <sub>0</sub>

# Compiler Full Abstraction


$$\begin{array}{ll} \mathbf{x = 1;} & \mathbf{x = 0;} \\ \mathbf{x ++;} & = \mathbf{x += 2;} \\ \mathbf{x} & \mathbf{x} \end{array}$$

$$\begin{array}{ll} \text{loadi } r_0 \ 1 & \text{loadi } r_0 \ 0 \\ \text{inc } r_0 & = \text{addi } r_0 \ 2 \\ \text{ret } r_0 & \text{ret } r_0 \end{array}$$


# Compiler Full Abstraction

$x = 1;$        $x = 0;$   
 $x ++;$        $= x += 2;$



and have different  
powers!

inc r<sub>0</sub>      = addi r<sub>0</sub> 2  
ret r<sub>0</sub>      ret r<sub>0</sub>



# Why is FAC Secure?

-  is an attacker linking or injecting **target** code

# Why is FAC Secure?

-  is an attacker linking or injecting **target** code

# Why is FAC Secure?

-  is an attacker linking or injecting **target** code
-  is not constrained by **source** constructs

# Why is FAC Secure?

-  is an attacker linking or injecting **target** code
-  is not constrained by **source** constructs
- the co-implied equalities reduce  to 

# Why is FAC Secure?

-  is an attacker linking or injecting **target** code
-  is not constrained by
  - FAC protects against these attacks
- the co-implied equalities reduce  to 

# Why is FAC Secure?

1. confidentiality
2. integrity
3. invariant definition
4. memory allocation
5. well-bracketed control flow

Agten *et al.*'12, Abadi and Plotkin '10, Jagadeesan *et al.*'11

# Why is FAC Secure?

confidentiality:

$$P1 = P2 \iff \llbracket P1 \rrbracket_T^S = \llbracket P2 \rrbracket_T^S$$

- **P1** and **P2** have **different** secrets
- but they are equivalent

Agten et al.'12, Abadi and Plotkin '10, Jagadeesan et al.'11

# Why is FAC Secure?

confidentiality:

$$P1 = P2 \iff \llbracket P1 \rrbracket_T^S = \llbracket P2 \rrbracket_T^S$$

- 1. •  $P1$  and  $P2$  have different secrets
- 2. • but they are equivalent
- 3. •  $\llbracket P1 \rrbracket_T^S$  and  $\llbracket P2 \rrbracket_T^S$  also have different secrets
- 4. • but they are equivalent
- 5. • but they are equivalent

Agten et al.'12, Abadi and Plotkin '10, Jagadeesan et al.'11

# Why is FAC Secure?

confidentiality:

$$P1 = P2 \iff \llbracket P1 \rrbracket_T^S = \llbracket P2 \rrbracket_T^S$$

- $P1$  and  $P2$  have **different** secrets
- but they are equivalent
- $\llbracket P1 \rrbracket_T^S$  and  $\llbracket P2 \rrbracket_T^S$  also have different secrets
- but they are equivalent
- so the secret **does not leak**

Agten et al.'12, Abadi and Plotkin '10, Jagadeesan et al.'11

# Why is FAC Secure?

1. confidentiality
2. integrity
3. invariant definition
4. memory allocation
5. well-bracketed control flow

If the source has it.

Agten *et al.*'12, Abadi and Plotkin '10, Jagadeesan *et al.*'11

# Why is FAC Secure?

1. confidentiality
2. integrity
3. i     • FAC preserves these properties
4. memory allocation
5. well-bracketed control flow

If the source has it.

Agten *et al.*'12, Abadi and Plotkin '10, Jagadeesan *et al.*'11



# Not All That Glitters is Gold

- No support for **separate compilation**

[Patrignani *et al.*'16, Juglaret *et al.*'16]



# Not All That Glitters is Gold

- No support for separate compilation

[Patrignani *et al.*'16, Juglaret *et al.*'16]



# Not All That Glitters is Gold

- No support for **separate compilation**  
[Patrignani *et al.*'16, Juglaret *et al.*'16]
- No support for **undefined behaviour** [Juglaret *et al.*'16]



# Not All That Glitters is Gold

- 
- A photograph of several gold-colored coins stacked in two piles. One pile is a vertical stack of coins, and the other is a more horizontal cluster. A black pen lies across the top of the vertical stack. The coins appear to be Canadian loonies.
- No support for **separate compilation**  
[Patrignani *et al.*'16, Juglaret *et al.*'16]
  - No support for **undefined behaviour** [Juglaret *et al.*'16]
  - **Costly** to enforce

# Not All That Glitters is Gold

- No support for **separate compilation**  
[Patrignani *et al.*'16, Juglaret *et al.*'16]
- No support for **undefined behaviour** [Juglaret *et al.*'16]
- **Costly** to enforce
- Preserves **hypersafety** under certain conditions [Patrignani and Garg '17]



# Perspective on Foundations



# Perspective on Foundations

A photograph of a paved road with double yellow lines receding into a vast, open landscape under a dramatic, cloudy sky.

Use Full Abstraction  
(with precautions)

# Perspective on Foundations

A landscape photograph showing a paved road with double yellow lines receding into a vast, open field under a dramatic, cloudy sky.

Invent  
new definitions

Use Full Abstraction  
(with precautions)

# Perspective on Foundations

Improve Full Abstraction

Invent  
new definitions

Use Full Abstraction  
(with precautions)

# Perspective on Foundations

A photograph of a long, straight asphalt road stretching into a vast, open landscape under a cloudy sky. The road has double yellow lines and is flanked by green grass and brown dirt. A white rectangular box with a drop shadow is positioned in the center-right of the image, containing the text.

Invent  
new definitions

# Perspective on Foundations

Invent

new definitions

Ongoing work with:

Catalin Hritcu  
(INRIA)

Deepak Garg  
(MPI-SWS)

# What More does Secure Compilation Offer?

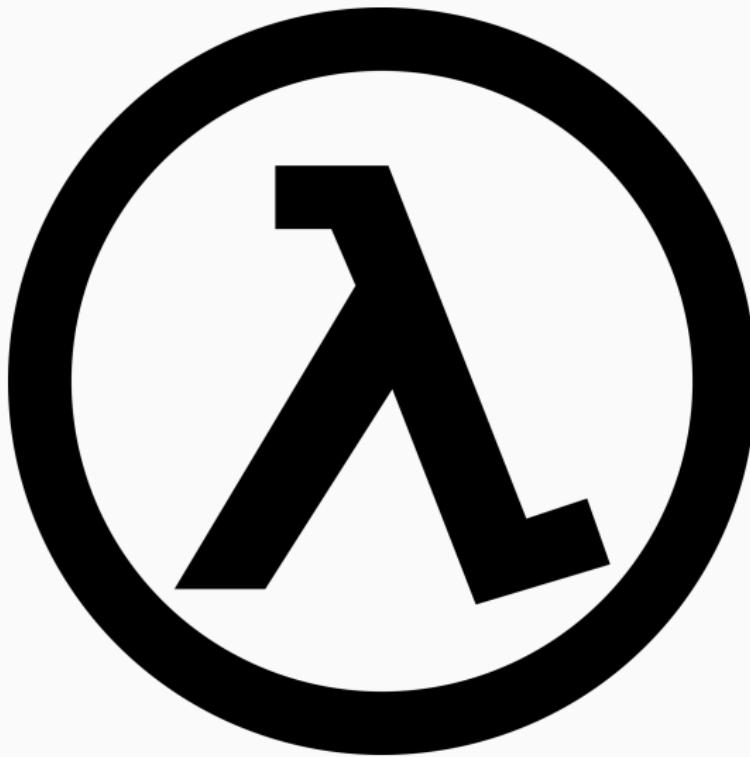
# What More does Secure Compilation Offer?

- study language techniques for proofs
- implement secure compilers to new security architectures

# **Programming Languages Techniques for Secure Compilation**

---

# What PL Want



# What PL Want

- better proof techniques

# Proving FAC

$$P1 \simeq_{ctx} P2$$



$$[P1]_T^S \simeq_{ctx} [P2]_T^S$$

# Proving FAC

$$P1 \simeq_{ctx} P2$$



$$[\![P1]\!]_T^S \simeq_{ctx} [\![P2]\!]_T^S$$

# Proving FAC

$$P1 \simeq_{ctx} P2$$



$$\llbracket P1 \rrbracket_T^S \simeq_{ctx} \llbracket P2 \rrbracket_T^S$$

# Proving FAC

$$P1 \simeq_{ctx} P2$$



$$\forall C. C[\llbracket P1 \rrbracket_T^S] \Downarrow \iff C[\llbracket P2 \rrbracket_T^S] \Downarrow$$



# Proving FAC

$$P1 \simeq_{ctx} P2$$



$$\llbracket P1 \rrbracket_T^S \dashv \llbracket P2 \rrbracket_T^S$$

# Proving FAC

$$P1 \simeq_{ctx} P2$$

Jagadeesan *et al.*'11,  
Agten *et al.*'12,  
Patrignani *et al.*'15'16,  
Juglaret *et al.*'16

$$\llbracket T \perp T \rrbracket_T = \llbracket T \triangleleft T \rrbracket_T$$

# Proving FAC

$$P1 \simeq_{ctx} P2$$



$$\llbracket P1 \rrbracket_T^S \approx \llbracket P2 \rrbracket_T^S$$

# Proving FAC

$$P1 \simeq_{ctx} P2$$

Abadi *et al.*'00'01'02'

Bugliesi *et al.*'07

Adao *et al.*'06

Fournet *et al.*'13

$$\llbracket T \vdash T \rrbracket \approx \llbracket T \triangleleft T \rrbracket$$

# Proving FAC

$$P1 \simeq_{ctx} P2$$



$$\llbracket P1 \rrbracket_T^S \sim_n \llbracket P2 \rrbracket_T^S$$

# Proving FAC

$$P1 \simeq_{ctx} P2$$

Ahmed *et al.*'8'11'14'15'16'17,  
Devriese *et al.*'16

$$\llbracket P1 \rrbracket_T^S \sim_n \llbracket P2 \rrbracket_T^S$$

# Proving FAC with Logical Relations

$$P1 \simeq_{ctx} P2$$

approx. compiler security

$$\llbracket P1 \rrbracket_T^S \simeq_{ctx}^? \llbracket P2 \rrbracket_T^S$$

# Proving FAC with Logical Relations

$$P1 \simeq_{ctx} P2$$

$$\begin{array}{ccc} C[\llbracket P1 \rrbracket_T^S] \Downarrow_n & \stackrel{?}{\Rightarrow} & C[\llbracket P2 \rrbracket_T^S] \Downarrow_- \\ \llbracket P1 \rrbracket_T^S \simeq_{ctx} ? \llbracket P2 \rrbracket_T^S \end{array}$$

approx. compiler security

# Proving FAC with Logical Relations

$$P1 \simeq_{ctx} P2$$

$$\begin{array}{c} \langle\langle C \rangle\rangle_n \sim_n C \\ P1 \sim_- [\![P1]\!]_T^S \end{array} \quad \begin{array}{c} \uparrow \\ (1) \end{array}$$

$$\begin{array}{ccc} C[\![P1]\!]_T^S \Downarrow_n & \stackrel{?}{\Rightarrow} & C[\![P2]\!]_T^S \Downarrow_- \\ [\![P1]\!]_T^S \simeq_{ctx} ? & & [\![P2]\!]_T^S \end{array}$$

approx. compiler security

# Proving FAC with Logical Relations

$$P1 \simeq_{ctx} P2$$

$$\langle\langle C \rangle\rangle_n [P1] \Downarrow_-$$

$$\begin{array}{l} \langle\langle C \rangle\rangle_n \sim_n C \\ P1 \sim_- [\![P1]\!]_T^S \end{array} \quad \begin{array}{c} \uparrow \\ (1) \end{array}$$

$$C[\![P1]\!]_T^S \Downarrow_n \stackrel{?}{\Rightarrow} C[\![P2]\!]_T^S \Downarrow_-$$

$$[\![P1]\!]_T^S \stackrel{?}{\simeq}_{ctx} [\![P2]\!]_T^S$$

approx. compiler security

# Proving FAC with Logical Relations

$$P1 \simeq_{ctx} P2$$

$$\langle\langle C \rangle\rangle_n [P1] \Downarrow_- \xrightarrow{(2)} \langle\langle C \rangle\rangle_n [P2] \Downarrow_-$$

$$\begin{array}{c} \langle\langle C \rangle\rangle_n \sim_n C \\ P1 \sim_- [\![P1]\!]_T^S \end{array} \quad \begin{array}{c} \uparrow \\ (1) \end{array}$$

$$\begin{array}{ccc} C[\![P1]\!]_T^S \Downarrow_n & \xrightarrow{?} & C[\![P2]\!]_T^S \Downarrow_- \\ [\![P1]\!]_T^S \simeq_{ctx} ? & & [\![P2]\!]_T^S \end{array}$$

approx. compiler security

# Proving FAC with Logical Relations

$$P1 \simeq_{ctx} P2$$

$$\langle\langle C \rangle\rangle_n[P1] \Downarrow_- \xrightarrow{(2)} \langle\langle C \rangle\rangle_n[P2] \Downarrow_-$$

$$\begin{array}{ccc} \langle\langle C \rangle\rangle_n \sim_n C & \uparrow(1) & \downarrow(3) \\ P1 \sim_- [\![P1]\!]_T^S & & \\ & & \langle\langle C \rangle\rangle_n \sim_- C \\ & & P2 \sim_- [\![P2]\!]_T^S \end{array}$$

$$C[\![P1]\!]_T^S \Downarrow_n \stackrel{?}{\Rightarrow} C[\![P2]\!]_T^S \Downarrow_-$$

$$[\![P1]\!]_T^S \simeq_{ctx} ? [\![P2]\!]_T^S$$

approx. compiler security

# Proving FAC with Logical Relations

$$P1 \simeq_{ctx} P2$$

$P1 \sim [P1]_T^S$  is obtained  
with standard  
techniques

Benton *et al.*'09'10

Hur *et al.*'11

Neis *et al.*'15

$$\mathbb{C}[P1]_T^S \Downarrow_n \stackrel{?}{\Rightarrow} \mathbb{C}[P2]_T^S \Downarrow_-$$

$$[P1]_T^S \simeq_{ctx} [P2]_T^S$$

approx. compiler security

# Proving FAC with Logical Relations

$\langle\!\langle C \rangle\!\rangle_n \sim C$  requires

- back-translation of terms
- reasoning at the type of  
back-translated terms

$\langle\!\langle C \rangle\!\rangle_n$

$P_1$

$$[\![P_1]\!]_T^S \stackrel{?}{\simeq}_{ctx} [\![P_2]\!]_T^S$$

approx. compiler security

# Proving FAC with Logical Relations

$\langle\!\langle C \rangle\!\rangle_n \sim C$  requires

- back-translation of terms
- reasoning at the type of back-translated terms
- needed for all kinds of back-translation

$$[\![P1]\!]_T^S \stackrel{?}{\simeq}_{ctx} [\![P2]\!]_T^S$$

approx. compiler security

# Proving FAC with Logical Relations

$\langle\!\langle C \rangle\!\rangle_n \sim C$  requires

- back-translation of terms
- reasoning at the type of back-translated terms
- needed for all kinds of back-translation
- needed for alternative criteria too

$$[P1]_T^S \stackrel{?}{\simeq}_{ctx} [P2]_T^S$$

approx. compiler security

$\langle\!\langle C \rangle\!\rangle_n$

$P1$

$P1$        $P2$

# Proving FAC with Logical Relations

$\langle\!\langle C \rangle\!\rangle_n \sim C$  requires

- back-translation of terms
- reasoning at the type of back-translated terms
- needed for all kinds of back-translation
- needed for alternative criteria too

$$[P1]_T^S \stackrel{?}{\simeq}_{ctx} [P2]_T^S$$

approx. compiler security

$\langle\!\langle C \rangle\!\rangle_n$

$P1$

$P1$        $P2$

# Proving FAC with Logical Relations

$\langle\!\langle C \rangle\!\rangle_n \sim C$  requires

- back-translation of terms
- reasoning at the type of back-translated terms
- needed for all kinds of back-translation
- needed for alternative criteria too

$$[P1]_T^S \stackrel{?}{\simeq}_{ctx} [P2]_T^S$$

approx. compiler security

$\langle\!\langle C \rangle\!\rangle_n$

$P1$

$P1$        $P2$

# **Security Architectures for Secure Compilation**

---

# Security Architectures for SC

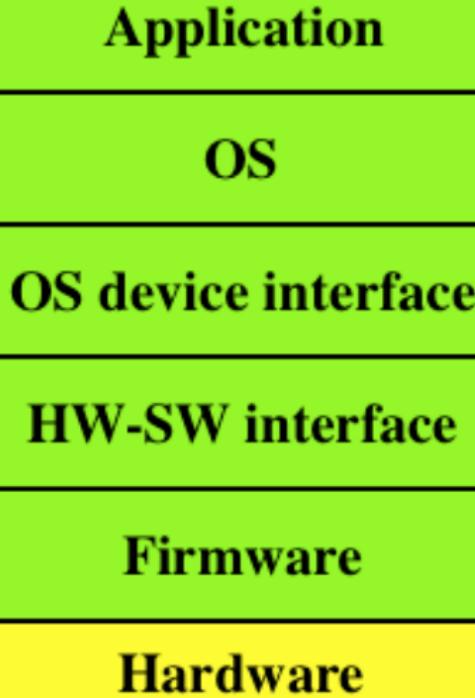


# Security Architectures for SC

Security Architectures:

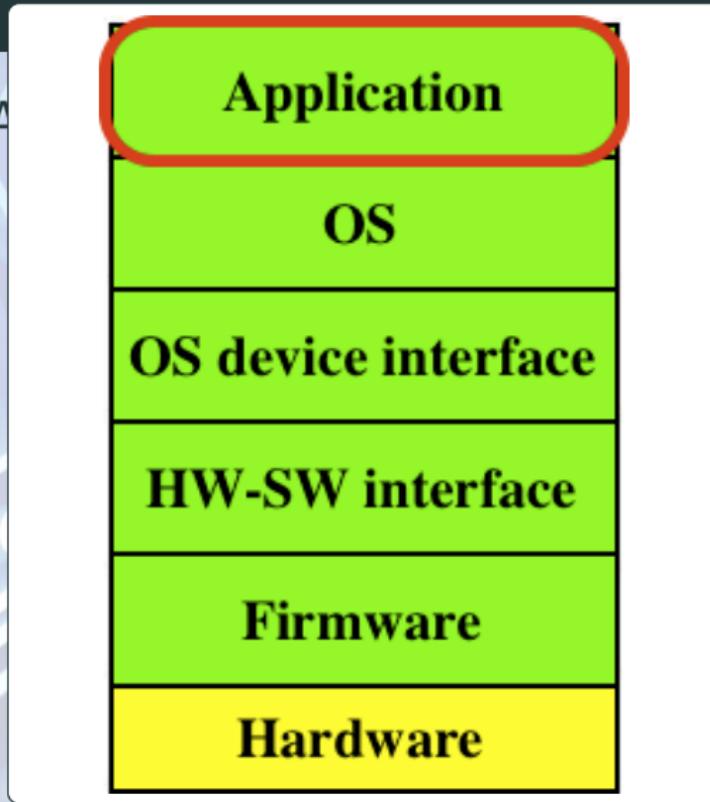
# Security Architectures for SC

Security A



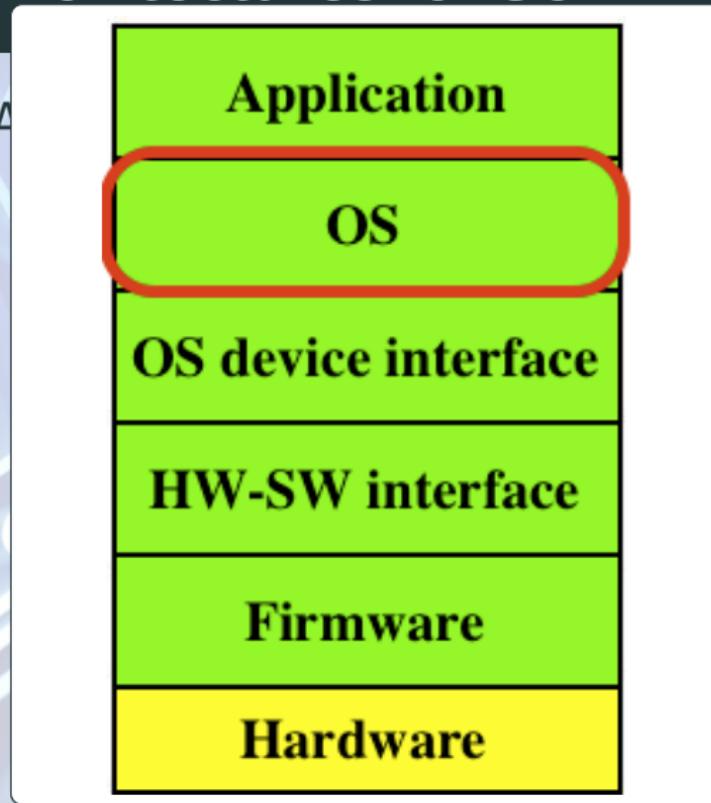
# Security Architectures for SC

Security A



# Security Architectures for SC

Security A



# Security Architectures for SC

Security A

**Application**

**OS**

**OS device interface**

**HW-SW interface**

**Firmware**

**Hardware**

# Security Architectures for SC

Security A

Application

OS

Reduced TCB  
Efficiency

Firmware

Hardware

# Security Architectures for SC

Security Architectures:

- ASLR
- Intel SGX-like enclaves
- Typed Assembly Languages
- Tagged Architectures (Pump)
- Capability Machines

# Security Architectures for SC

## Security Architectures:

- ASLR [Abadi & Plotkin '10, Jagadeesan *et al.*'11]
- Intel SGX-like enclaves [Agten *et al.*'12, Patrignani *et al.*'13,'16]
- Typed Assembly Languages [Ahmed *et al.*'14]
- Tagged Architectures (Pump) [Juglaret *et al.*'16]
- Capability Machines [Tsampas *et al.*'17, WIP]

# Capability Machines: Cheri

## Capability Machines

- Hardware support for **fine-grained** Capabilities

# Capability Machines: Cheri

## Capability Machines

- Handwritten notes for Cheri machines  
Cap

Capability Mantra:  
subjects  
perform operations  
on objects  
if they have rights

# Capability Machines: Cheri

## Capability Machines

- Handwritten notes for Cheri machines

Cap

Capability Mantra:  
subjects  
perform operations  
on objects  
if they have rights

instructions

# Capability Machines: Cheri

## Capability Machines

- Handwritten notes for Cheri machines

Cap

Capability Mantra:

subjects

perform operations

on objects

if they have rights

instructions

read/ write/  
execute

# Capability Machines: Cheri

## Capability Machines

- Handwritten notes for Cheri machines

Cap

Capability Mantra:

subjects

perform operations

on objects

if they have rights

instructions

read/ write/  
execute

address  
ranges

# Capability Machines: Cheri

## Capability Machines

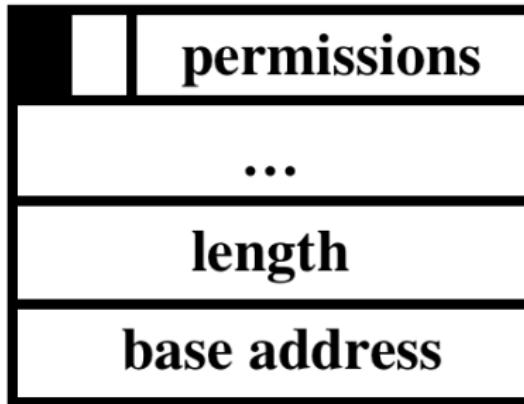
- Hardware support for **fine-grained** Capabilities
- Cheri (MIPS extension, FPGA) [Woodruff et al'14]

# Capability Machines: Cheri

## Capability Machines

- Hard
- Capa
- Cheri

A Cheri capability



[Buff et al'14]

# Capability Machines: Cheri

## Capability Machines

- Hardware support for **fine-grained** Capabilities
- Cheri (MIPS extension, FPGA) [Woodruff et al'14]
- Tagged memory
- Aligned memory
- Capability registers file
- Capability instructions

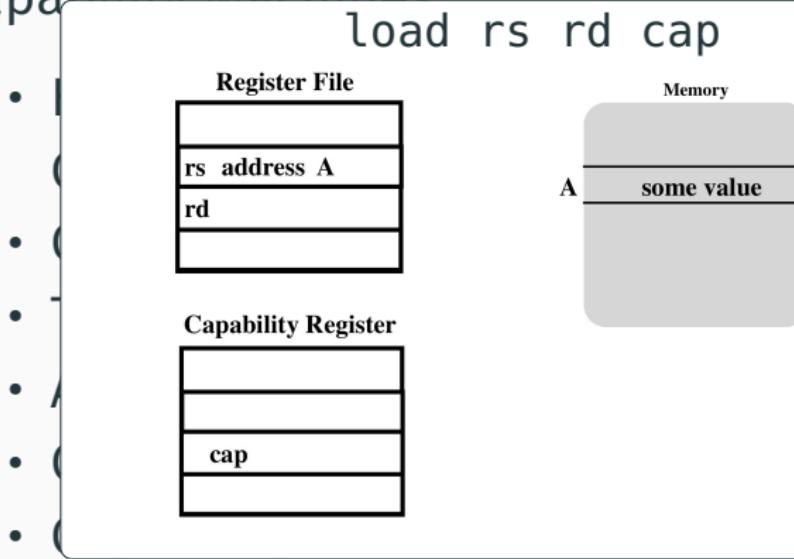
# Capability Machines: Cheri

## Capability Machines

- Hardware support for **fine-grained** Capabilities
- Cheri (MIPS extension EDGA) [Woodruff et al.'14]  
load rs rd cap
- Tagged memory
- Aligned memory
- Capability registers file
- Capability instructions

# Capability Machines: Cheri

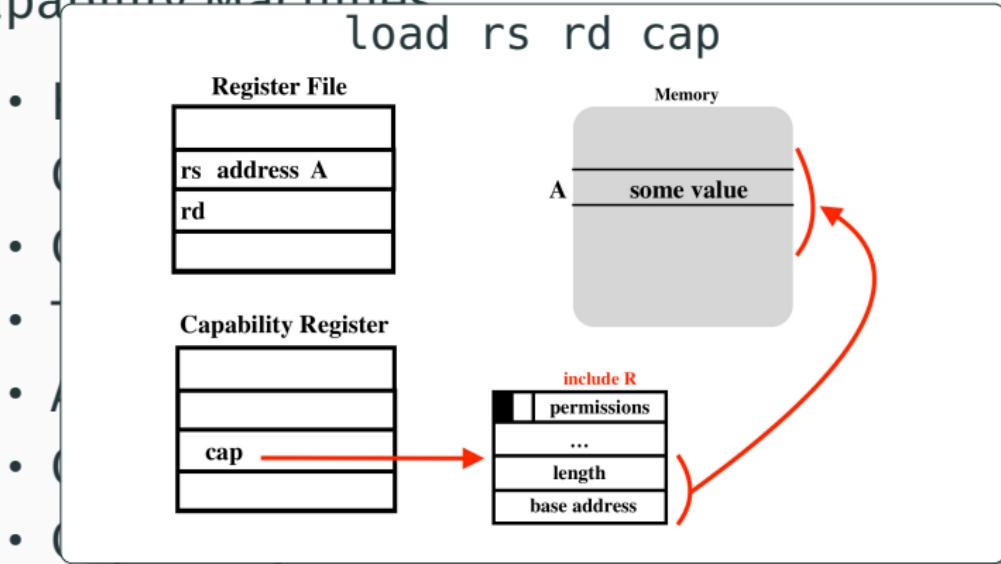
## Capability Machines



[l'14]

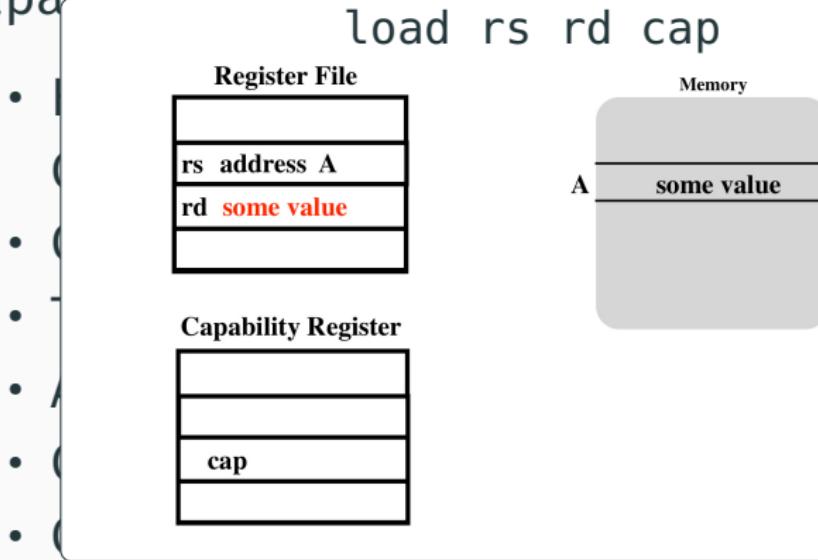
# Capability Machines: Cheri

## Capability Machines



# Capability Machines: Cheri

## Capability Machines



[l'14]

# Capability Machines: Cheri

## Capability Machines

- Hardware support for **fine-grained** capabilities
- Cheri [Kilby et al'14]
- Tags
- Aligned memory
- Capability registers file
- Capability instructions

**Unforgeable** capabilities at the hardware level

[Kilby et al'14]

# Capability Machines: Cheri

## Capability Machines

- Hardware support for **fine-grained** capabilities
- Cheri [Haller et al'14]
- Tags: Mature: has a FreeBSD port
- Aligned memory
- Capability registers file
- Capability instructions

# CM and Secure Compilation

- identify secure compartments

# CM and Secure Compilation

- identify secure compartments
- wrap compiled code in code and data capabilities: **isolation**

# CM and Secure Compilation

- identify secure compartments
- wrap compiled code in code and data capabilities: **isolation**
- capabilities regulate access to methods:  
**public/private**

# CM and Secure Compilation

- identify secure compartments
- wrap compiled code in code and data capabilities: **isolation**
- capabilities regulate access to methods:  
**public/private**
- capabilities regulate access to objects:  
**shared/local**

# CM and Secure Compilation

- identify secure compartments
- wrap compiled code in code and data capabilities: **isolation**
- capabilities regulate access to methods: **public/private**
- capabilities regulate access to objects: **shared/local**
- support **dynamic** security policies  
(runtime modification of accesses)

# CM and Secure Compilation

- identify secure compartments
  - wrap them in compartments
  - capability-based security
  - capability-based sharing
  - support runtime modification of accesses
- More efficient than existing approaches:**
- results

# CM and Secure Compilation

- identify secure compartments
  - wrap them in compartments
  - capture their access patterns
  - capture their security requirements
  - capture their sharing requirements
  - support runtime modification of accesses
- More efficient than existing approaches:**  
Support unprecedented security paradigms

# CM and Secure Compilation

- identify secure compartments
  - wrap them in compartments
  - capture access patterns
  - capture shared resources
  - support unprecedented security paradigms
  - support shared memory
  - support runtime modification of accesses
- More efficient than existing approaches:**
- Support unprecedented security paradigms:**
- Running! implemented by Tsampas**

# Conclusion

- motivations for secure compilation

# Conclusion

- motivations for secure compilation
- secure compilation criterion: fully abstract compilation

# Conclusion

- motivations for secure compilation
- secure compilation criterion: fully abstract compilation
- proof techniques for fully abstract compilation

# Conclusion

- motivations for secure compilation
- secure compilation criterion: fully abstract compilation
- proof techniques for fully abstract compilation
- secure compilation to capability machines

# Conclusion

