Secure Compilation

an extensive introduction

Marco Patrignani

9^{th} October 2017

CISPA
Center for IT-Security, Privacy
and Accountability


MAX PLANCK INSTITUTE
FOR SOFTWARE SYSTEMS
CISPA-Stanford Center
FOR CYBERSECURITY
KU LEUVEN
ALMA MATER STUDIORUM
A.D. 1088
ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA
1
2
3
4
Germany
Belgium
Luxembourg
Slovenia
Croatia
Bruges
Antwerp
Düsseldorf
Leipzig
Dresden
Görlitz
Brussels
Cologne
Chemnitz
Gera
Lille
Amiens
Saint-Quentin
Reims
Wiesbade
Nuremberg
Prague
Saarbrü
Karlsruhe
Strasbourg
Munich
Linz
Freiburg
Konstanz
Basel
Zurich
Innsbr
Dijon
Besançon
Bourges
Chalon-sur-Saône
Clermont-Ferrand
Lyon
Milan
Venice
Trieste
Rijeka
Padua
Grenoble
Turin
Valence
Parma
Asti
Genoa
Bologna
Ravenna
Pula
Avignon

Collaborators 








Contents

What is Secure Compilation?

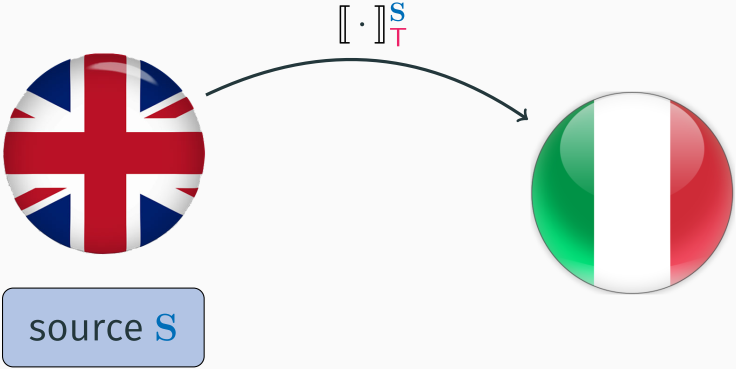
Compilation

Compilation

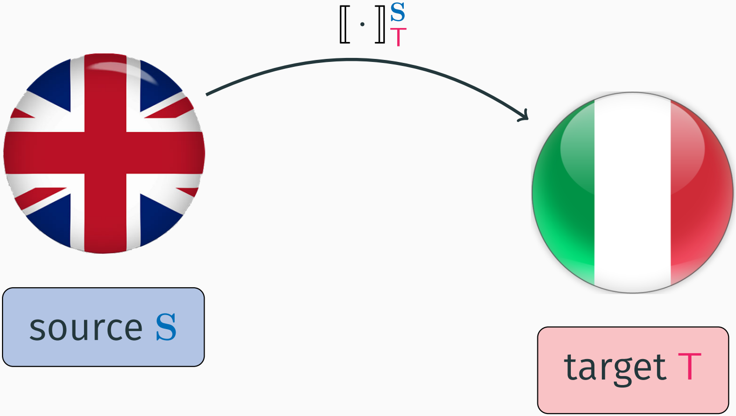
Compilation

source S

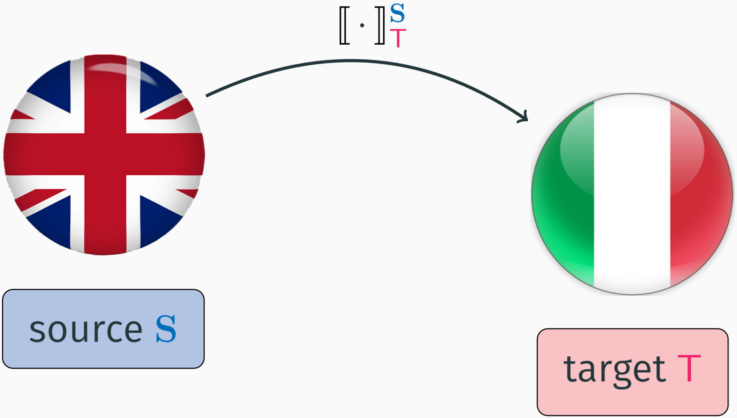
Compilation



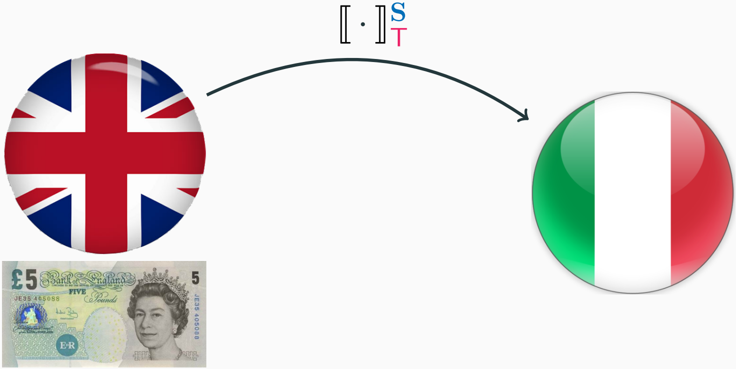
Compilation



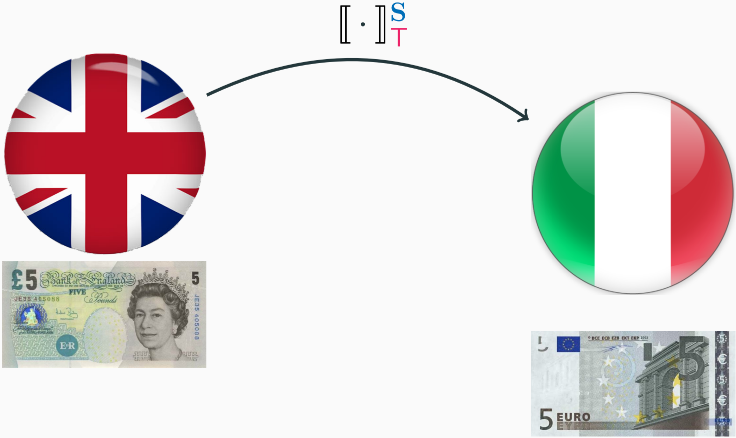
Correct Compilation



Correct Compilation

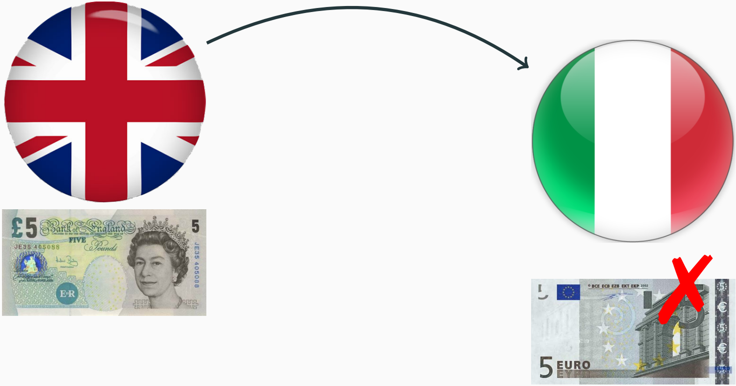


Correct Compilation



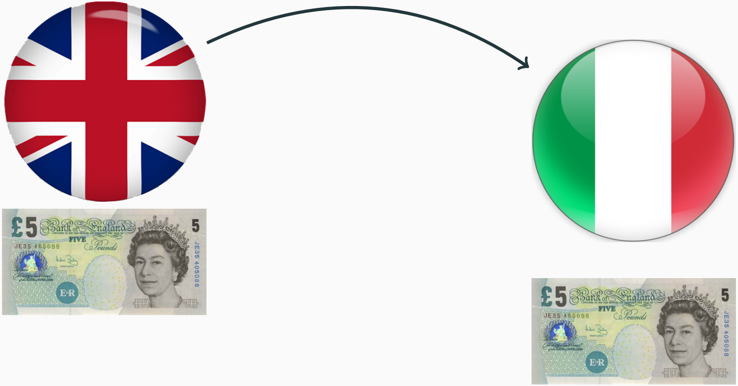
Correct Compilation

$[\![\cdot]\!]^{S}_{T}$

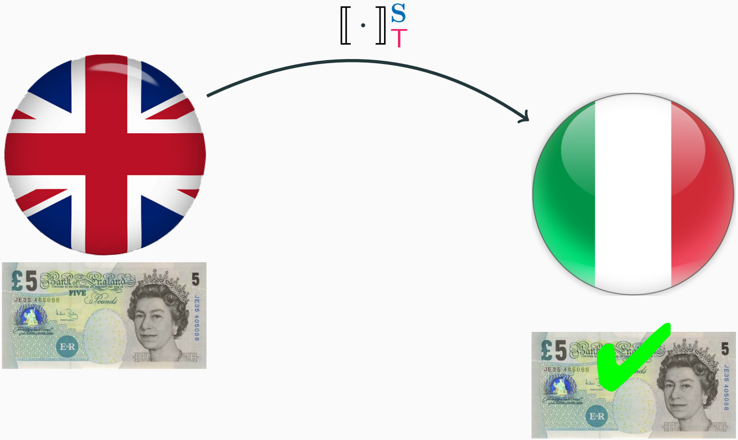


Correct Compilation

$[\![\cdot]\!]^{\mathsf{S}}_{\mathsf{T}}$



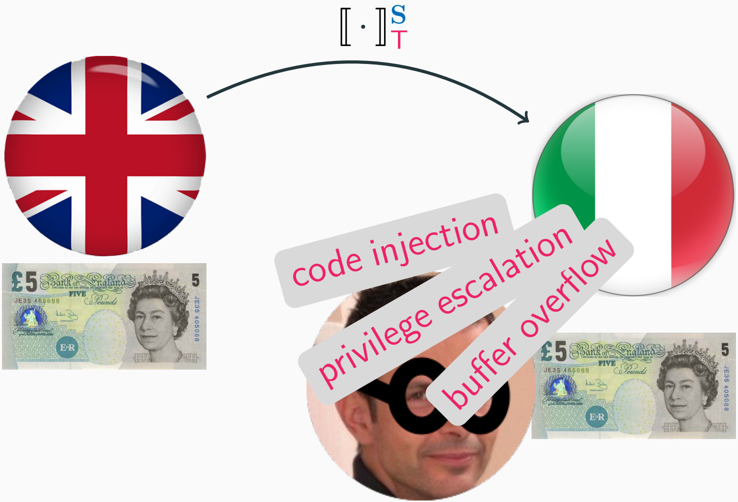
Correct Compilation



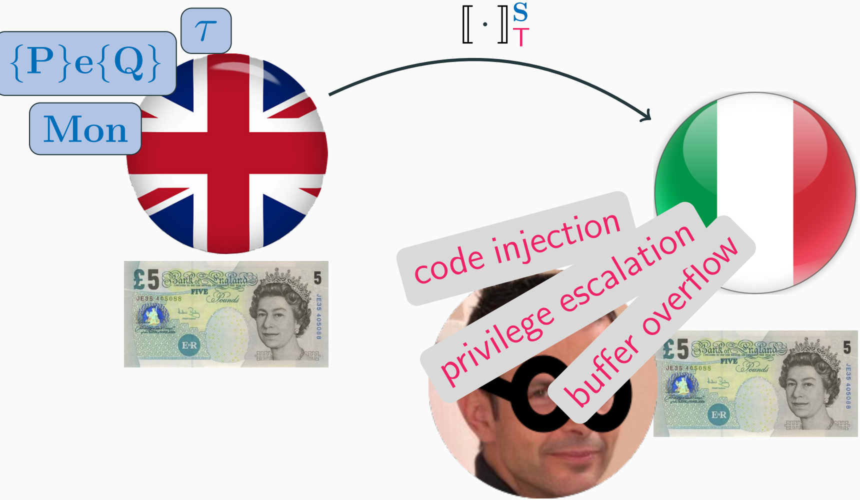
Secure Compilation

$[\![\cdot]\!]^{\mathbf{S}}_{\mathsf{T}}$

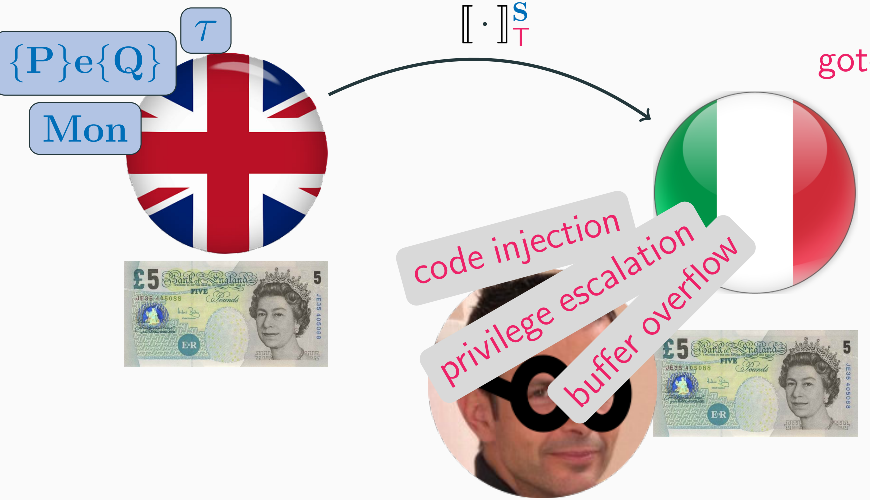
Secure Compilation



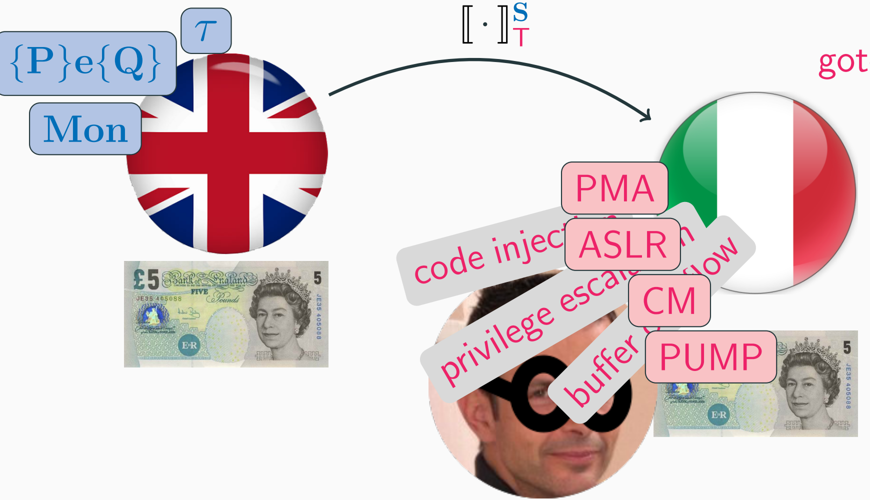
Secure Compilation



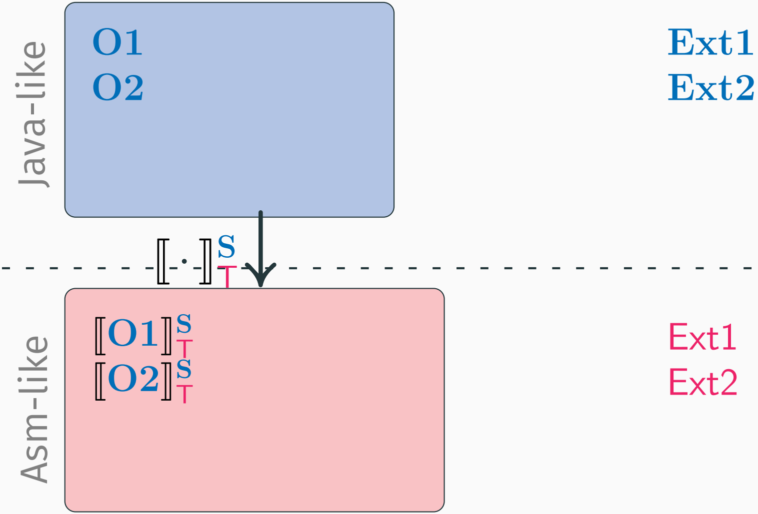
Secure Compilation



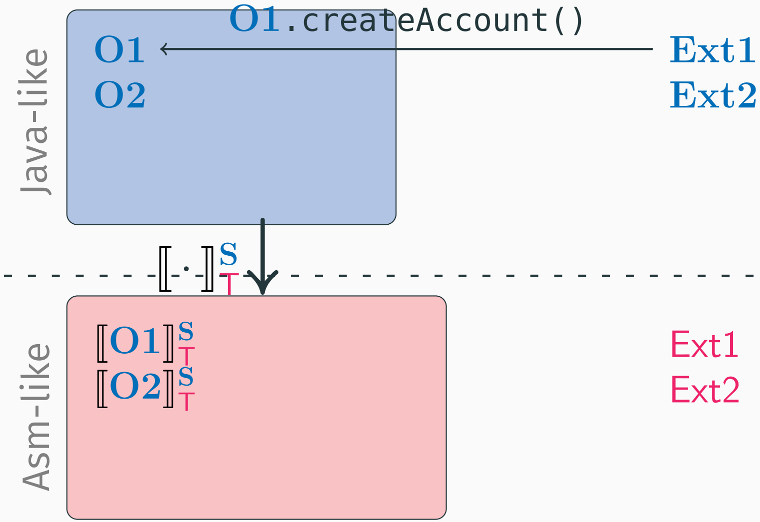
Secure Compilation



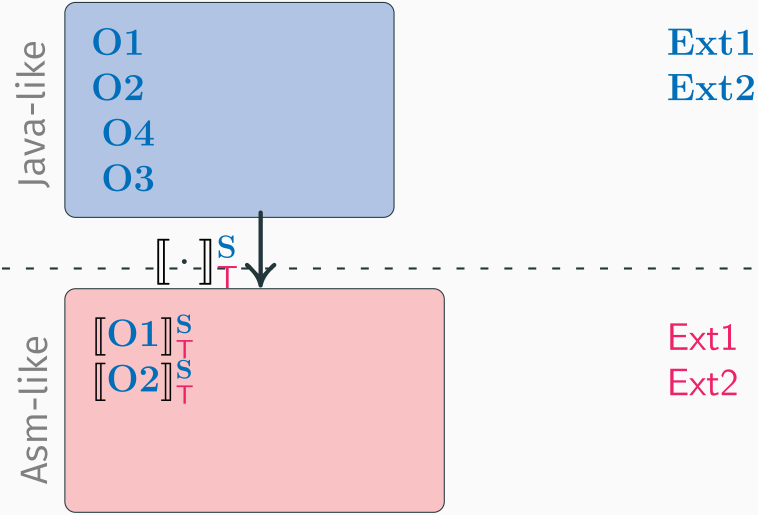
Memory Allocation Issues Patrignani *et al.*'15'16



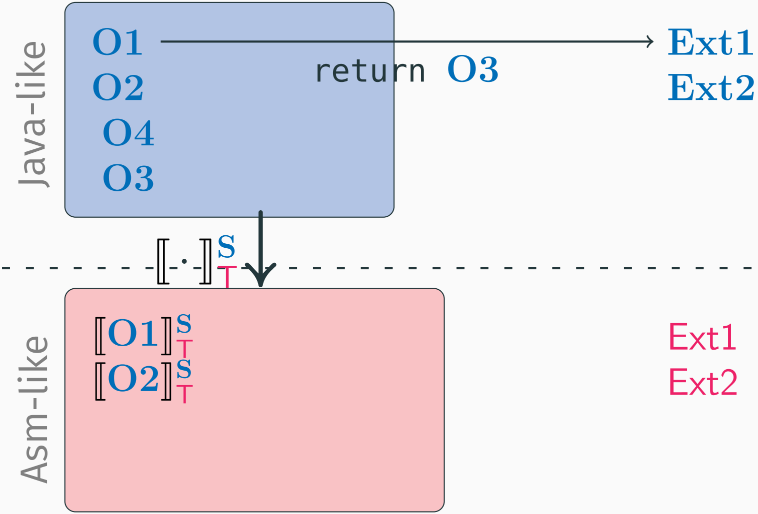
Memory Allocation Issues Patrignani *et al.*'15'16



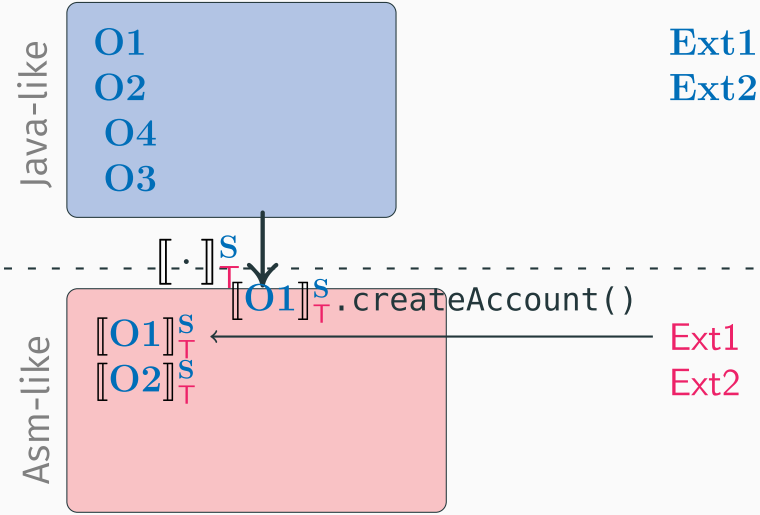
Memory Allocation Issues Patrignani *et al.*'15'16



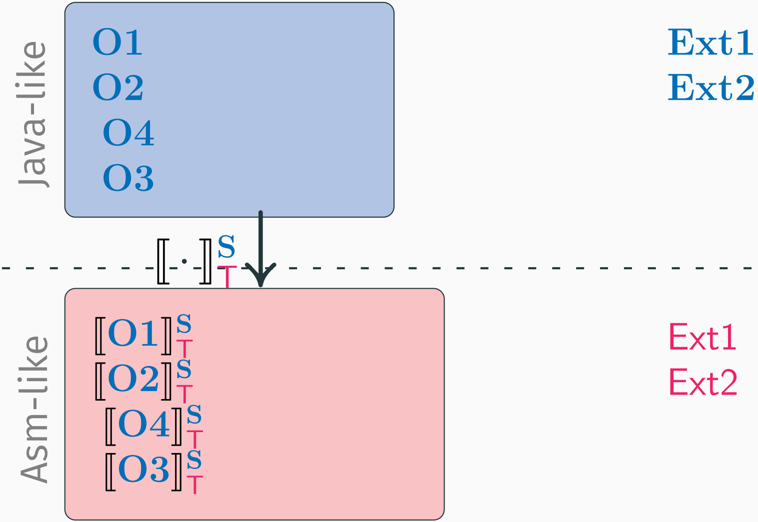
Memory Allocation Issues Patrignani *et al.*'15'16



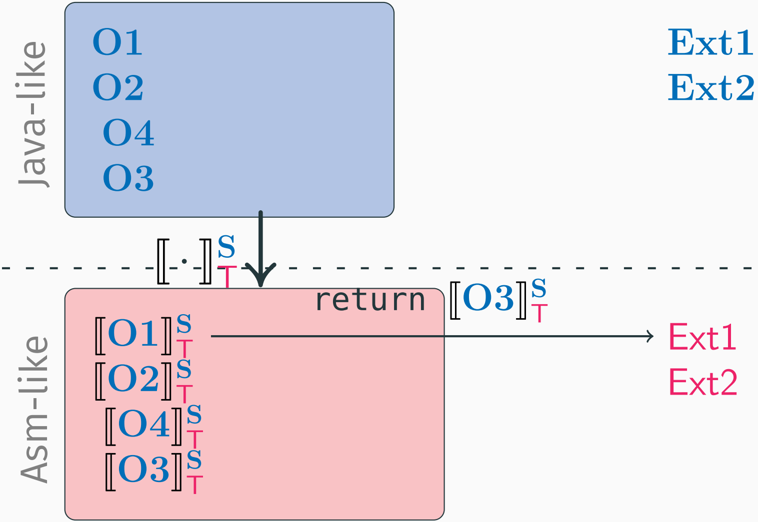
Memory Allocation Issues Patrignani *et al.*'15'16



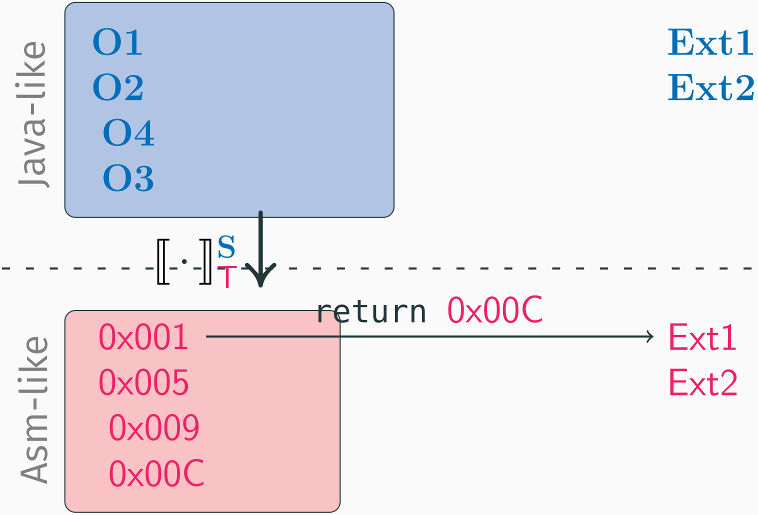
Memory Allocation Issues Patrignani *et al.*'15'16



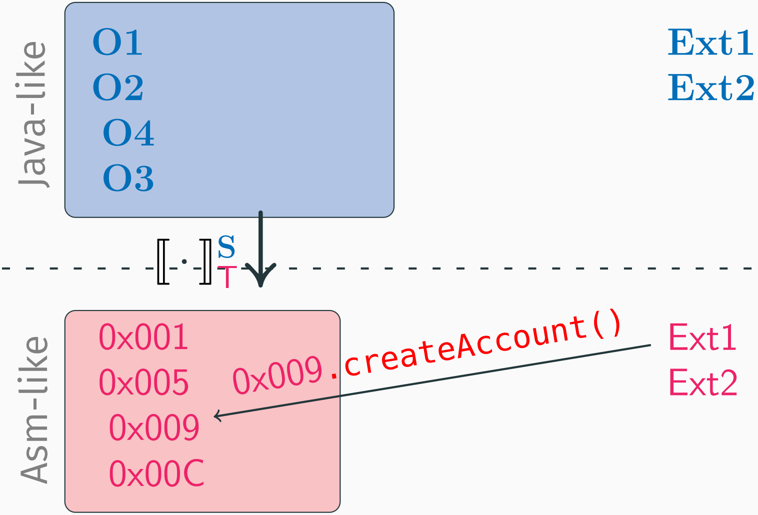
Memory Allocation Issues Patrignani *et al.*'15'16



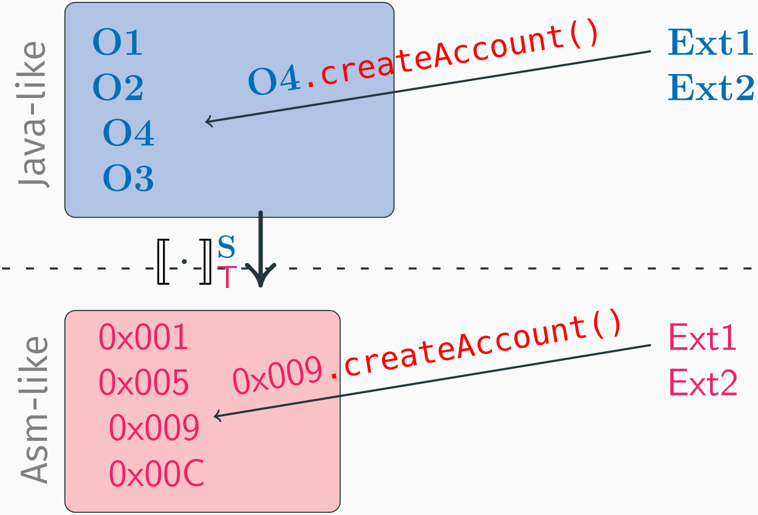
Memory Allocation Issues Patrignani *et al.*'15'16



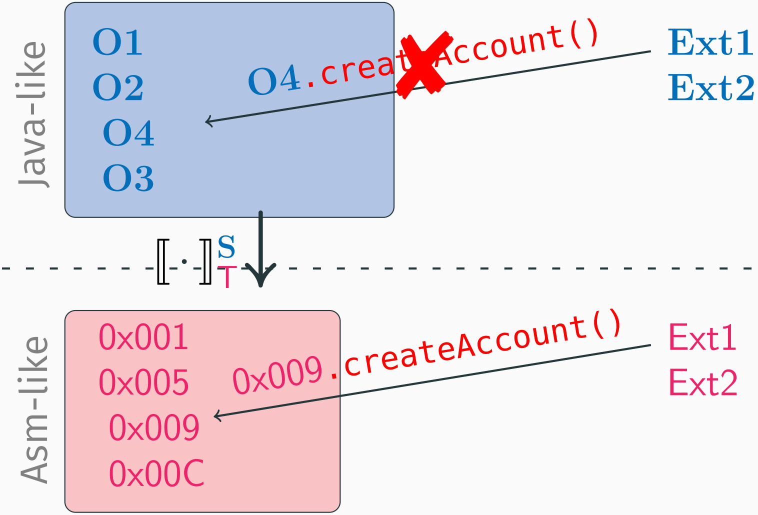
Memory Allocation Issues Patrignani *et al.*'15'16



Memory Allocation Issues Patrignani *et al.*'15'16



Memory Allocation Issues Patrignani *et al.*'15'16

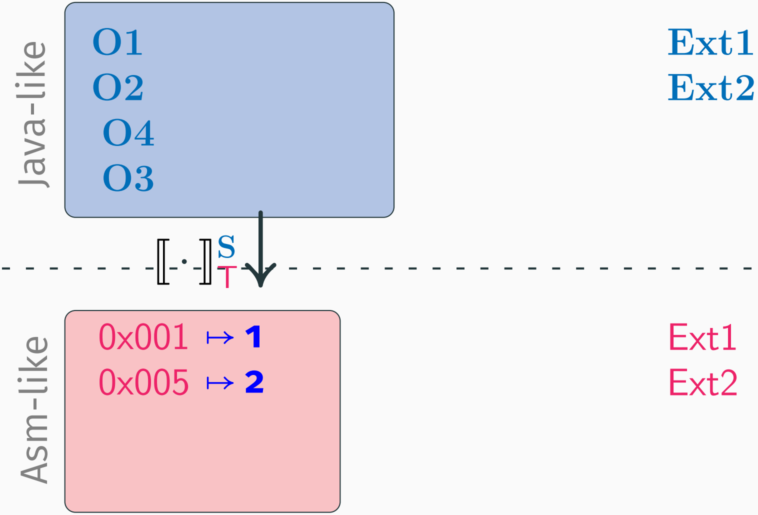


Memory Allocation Issues — Patrignani *et al.*'15'16

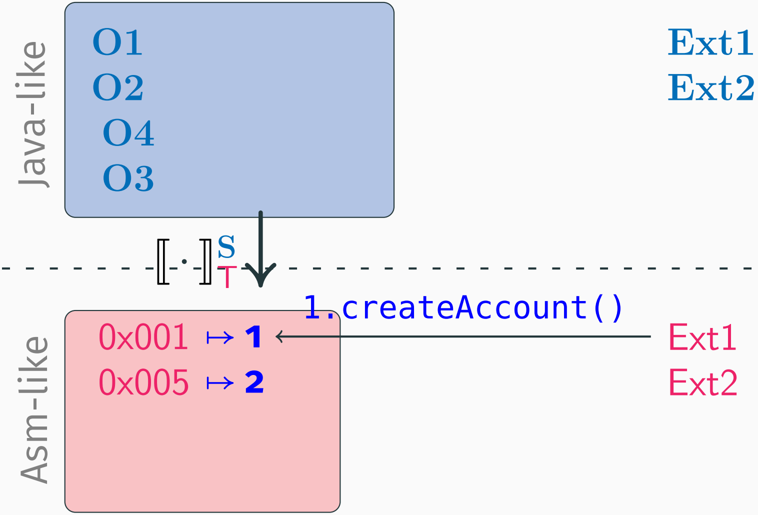
Issue: Oid guessing

Solution: keep a map from Oid to random numbers

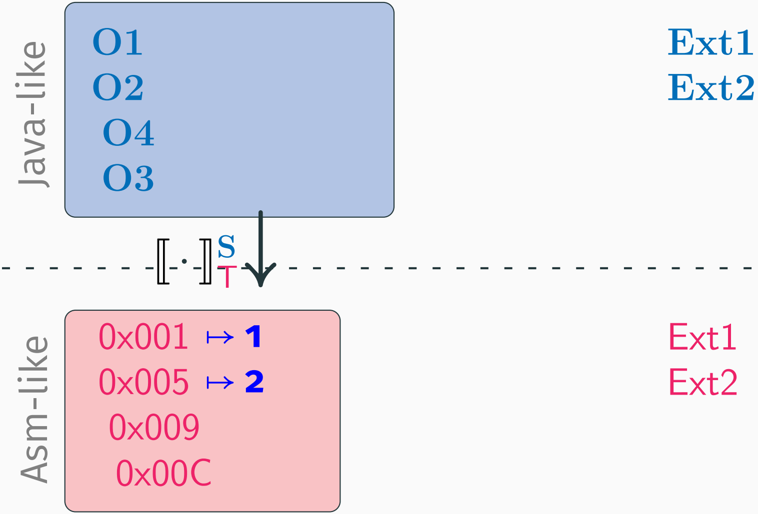
Memory Allocation Issues Patrignani *et al.*'15'16



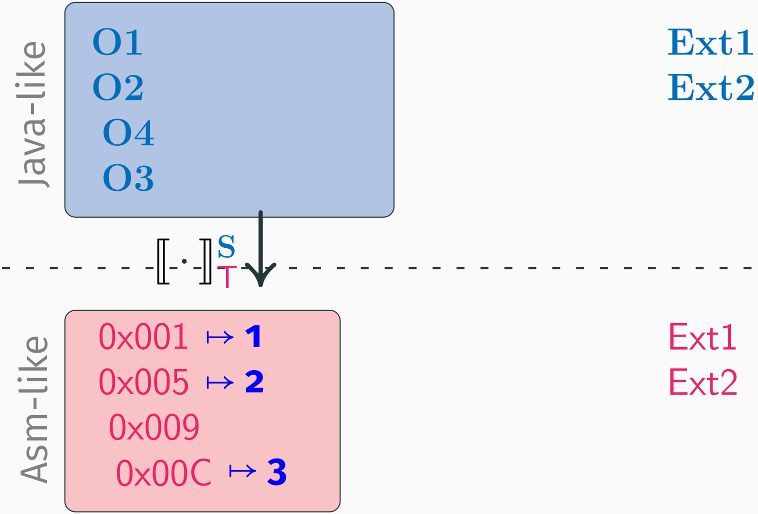
Memory Allocation Issues Patrignani *et al.*'15'16



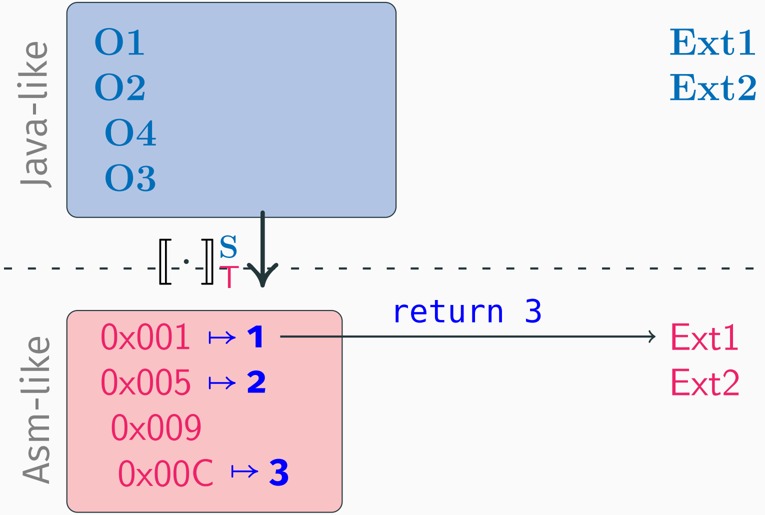
Memory Allocation Issues Patrignani *et al.*'15'16



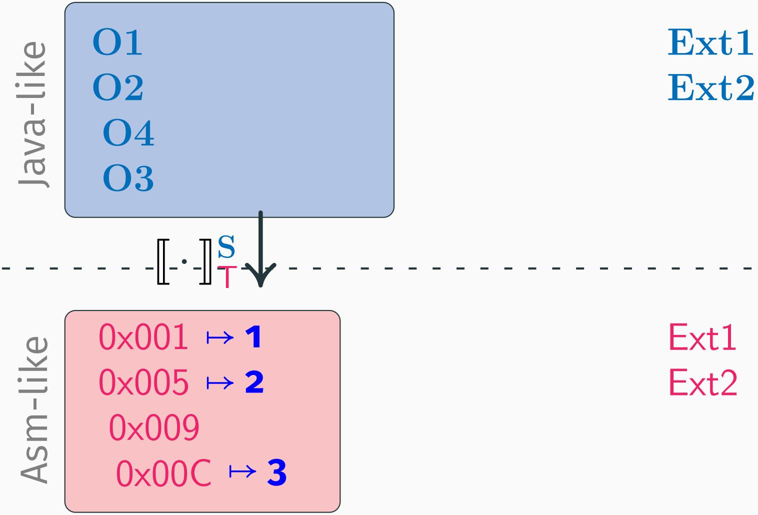
Memory Allocation Issues Patrignani *et al.*’15’16



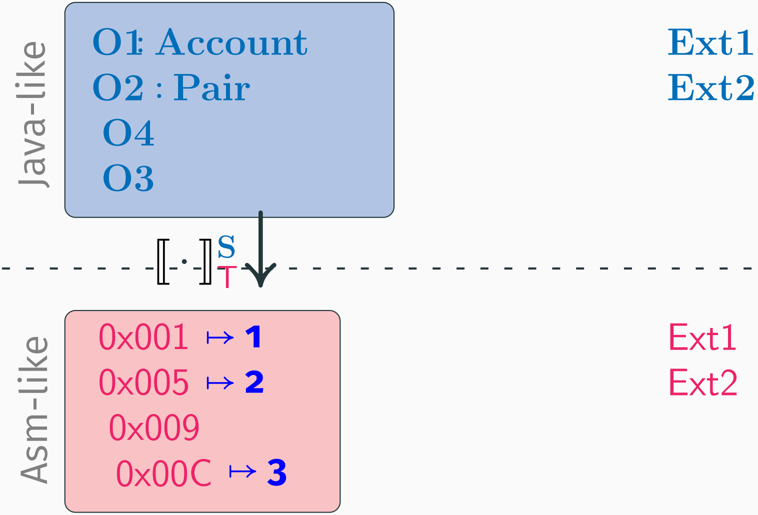
Memory Allocation Issues Patrignani *et al.*'15'16



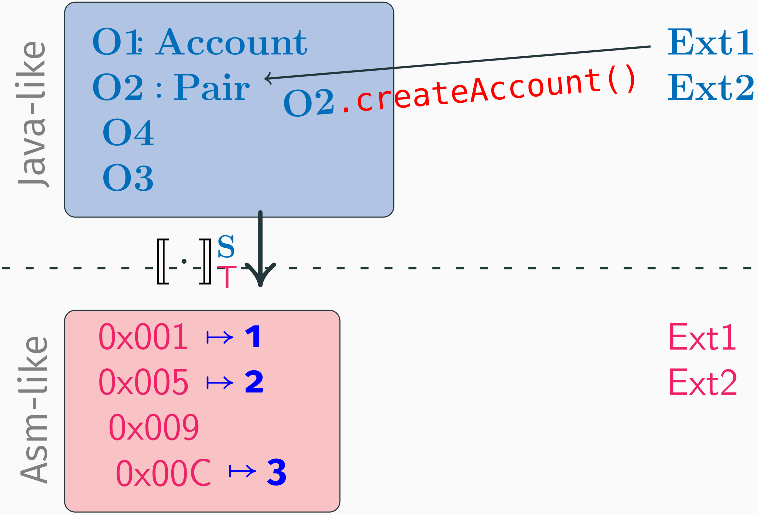
Memory Allocation Issues Patrignani *et al.*'15'16



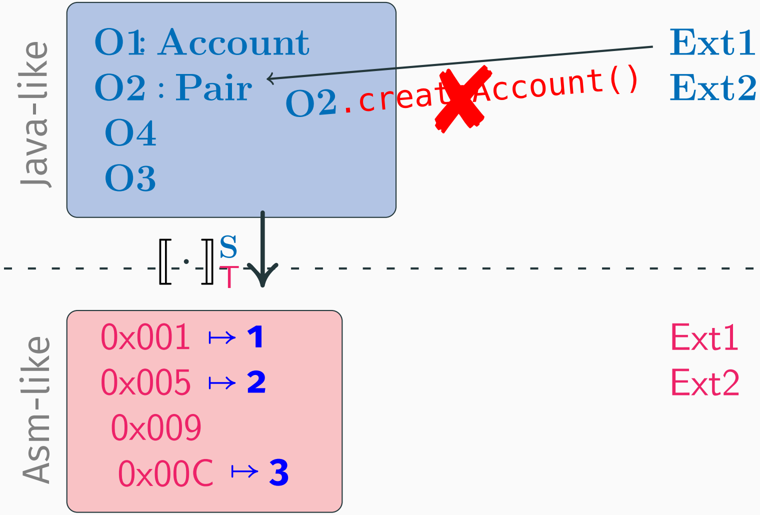
Memory Allocation Issues Patrignani *et al.*'15'16



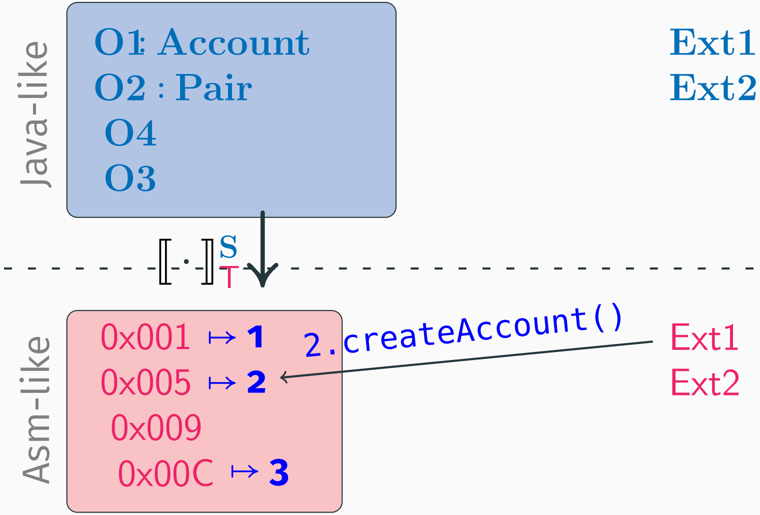
Memory Allocation Issues Patrignani *et al.*’15’16



Memory Allocation Issues Patrignani *et al.*'15'16



Memory Allocation Issues Patrignani *et al.*'15'16



Memory Allocation Issues Patrignani *et al.*'15'16

O1: Account

Ext1

Ext2

Issue: type violation

Solution: add dynamic typechecks

Asm-like

0x001 ↦ **1**

0x005 ↦ **2**

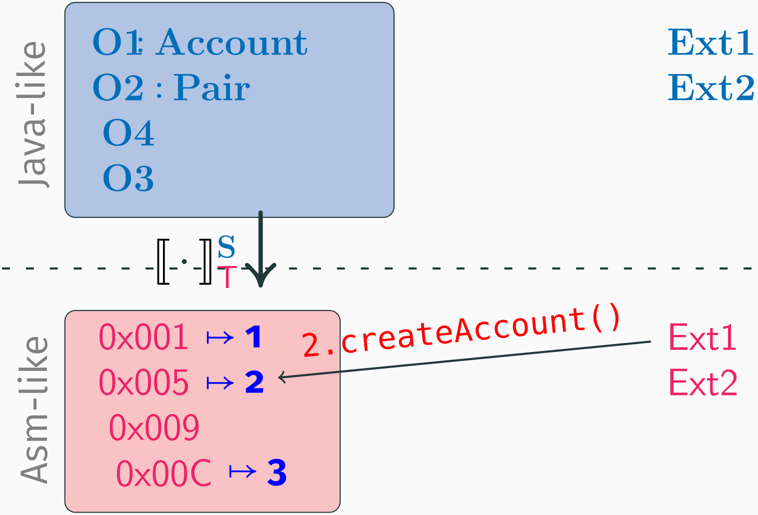
0x009

0x00C ↦ **3**

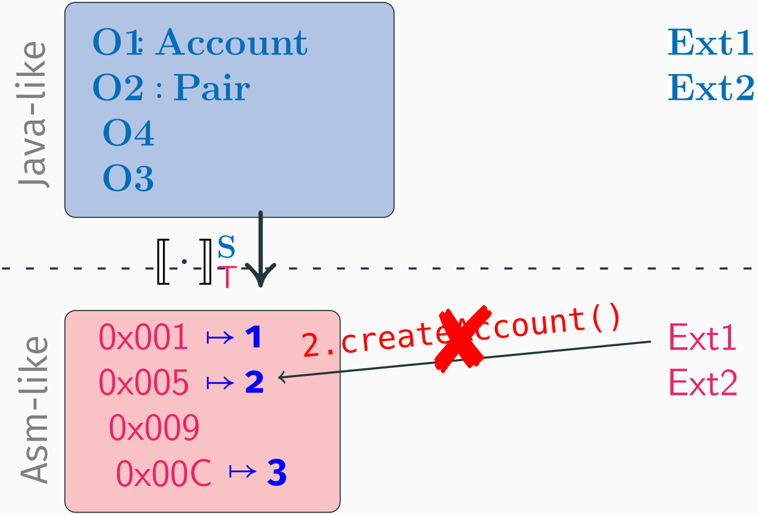
Ext1

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 ↦ **1**
0x005 ↦ **2**
0x009
0x00C ↦ **3**

Ext1
Ext2

Memory Allocation Issues Patrignani *et al.*’15’16

- design
- implement
- 

Ext1
Ext2

Ext1
Ext2

Asm-

0x009
0x00C $\mapsto$ **3**

Memory Allocation Issues Patrignani *et al.*'15'16

- design
- implement
- 
- ...or?

Ext1
Ext2

Ext1
Ext2

Asm-U

0x009
0x00C ↦ **3**

Guarantees

- How do we know we are right?

Guarantees

- How do we know we are right?
- How can we know that $[\![\cdot]\!]^{\mathbf{S}}_{\mathsf{T}}$ is secure?

Guarantees

Prove

$[\![\cdot]\!]^{\mathbf{S}}_{\mathsf{T}}$ to attain a secure compilation criterion

Guarantees

- How do we know we are right?
- How can we know that $[\![\cdot]\!]^{\mathbf{S}}_{\mathsf{T}}$ 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
Digital Equipment Corporation
Systems Research Center

Cédric Fournet
Cedric.Fournet@inria.fr
INRIA Rocquencourt

Georges Gonthier
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,*

$$P \approx Q \quad \textit{if and only if} \quad \mathcal{E}nv\,[\,[\![P]\!]\,] \approx \mathcal{E}nv\,[\,[\![Q]\!]\,]$$

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

The r … e):

wha … e

compilers?

Fully Abstract Compilation (FAC)

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

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

Fully Abstract Compilation Influence

Fully Abstract Compilation Influence

Why

does Fully Abstract Compilation entail **security**?

Because

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

Compiler Full Abstraction

$$\mathbf{x} = \mathbf{1};$$
$$\mathbf{x}{++};$$
$$\mathbf{x}$$

$$\mathbf{x} = \mathbf{0};$$
$$\mathbf{x}{+=}\ \mathbf{2};$$
$$\mathbf{x}$$

Compiler Full Abstraction

$$\begin{array}{l} \mathbf{x} = \mathbf{1}; \\ \mathbf{x}{+}{+}; \\ \mathbf{x} \end{array} \quad = \quad \begin{array}{l} \mathbf{x} = \mathbf{0}; \\ \mathbf{x}{+}{=}\ \mathbf{2}; \\ \mathbf{x} \end{array}$$

Compiler Full Abstraction

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

$$\Updownarrow$$

$$\begin{array}{ll} \mathsf{loadi}\ \mathsf{r}_0\ 1 & \mathsf{loadi}\ \mathsf{r}_0\ 0 \\ \mathsf{inc}\ \mathsf{r}_0 & \mathsf{addi}\ \mathsf{r}_0\ 2 \\ \mathsf{ret}\ \mathsf{r}_0 & \mathsf{ret}\ \mathsf{r}_0 \end{array}$$

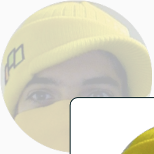
Compiler Full Abstraction

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

$$\Updownarrow$$

$$\begin{array}{lcl} \mathsf{loadi}\ \mathsf{r}_0\ 1 & & \mathsf{loadi}\ \mathsf{r}_0\ 0 \\ \mathsf{inc}\ \mathsf{r}_0 & = & \mathsf{addi}\ \mathsf{r}_0\ 2 \\ \mathsf{ret}\ \mathsf{r}_0 & & \mathsf{ret}\ \mathsf{r}_0 \end{array}$$

Compiler Full Abstraction



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

 and  have different powers!

inc r_0 ret r_0 = addi r_0 2 ret r_0

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

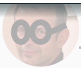
Why is FAC Secure?

-  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?

1. c
2. i
3. i
4. r
5. w

confidentiality:

$$\mathbf{P1} = \mathbf{P2} \iff [\![\mathbf{P1}]\!]^{\mathbf{S}}_{\mathsf{T}} = [\![\mathbf{P2}]\!]^{\mathbf{S}}_{\mathsf{T}}$$

- $\mathbf{P1}$ and $\mathbf{P2}$ have different secrets
- but they are equivalent

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

Why is FAC Secure?

1. c
2. i
3. i
4. r
5. w

confidentiality:

$$\mathbf{P1} = \mathbf{P2} \iff [\![\mathbf{P1}]\!]^{\mathbf{S}}_{\mathsf{T}} = [\![\mathbf{P2}]\!]^{\mathbf{S}}_{\mathsf{T}}$$

- $\mathbf{P1}$ and $\mathbf{P2}$ have different secrets
- but they are equivalent
- $[\![\mathbf{P1}]\!]^{\mathbf{S}}_{\mathsf{T}}$ and $[\![\mathbf{P2}]\!]^{\mathbf{S}}_{\mathsf{T}}$ also have different secrets
- but they are equivalent

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

Why is FAC Secure?

1. c
2. i
3. i
4. r
5. v

confidentiality:

$$\mathbf{P1} = \mathbf{P2} \iff [\![\mathbf{P1}]\!]^{S}_{T} = [\![\mathbf{P2}]\!]^{S}_{T}$$

- $\mathbf{P1}$ and $\mathbf{P2}$ have different secrets
- but they are equivalent
- $[\![\mathbf{P1}]\!]^{S}_{T}$ and $[\![\mathbf{P2}]\!]^{S}_{T}$ 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
4. memory allocation
5. well-bracketed control flow

- FAC preserves these properties

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

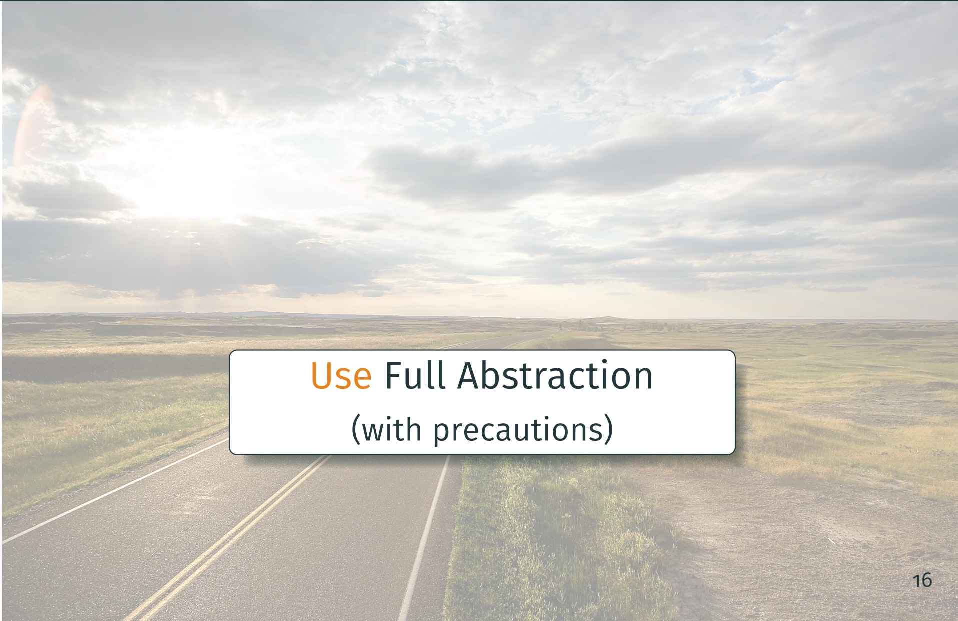
- No support for separate compilation [Patrignani *et al.*’16, Juglaret *et al.*’16]
- No support for undefined behaviour [Juglaret *et al.*’16]
- Costly to enforce

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



Use Full Abstraction

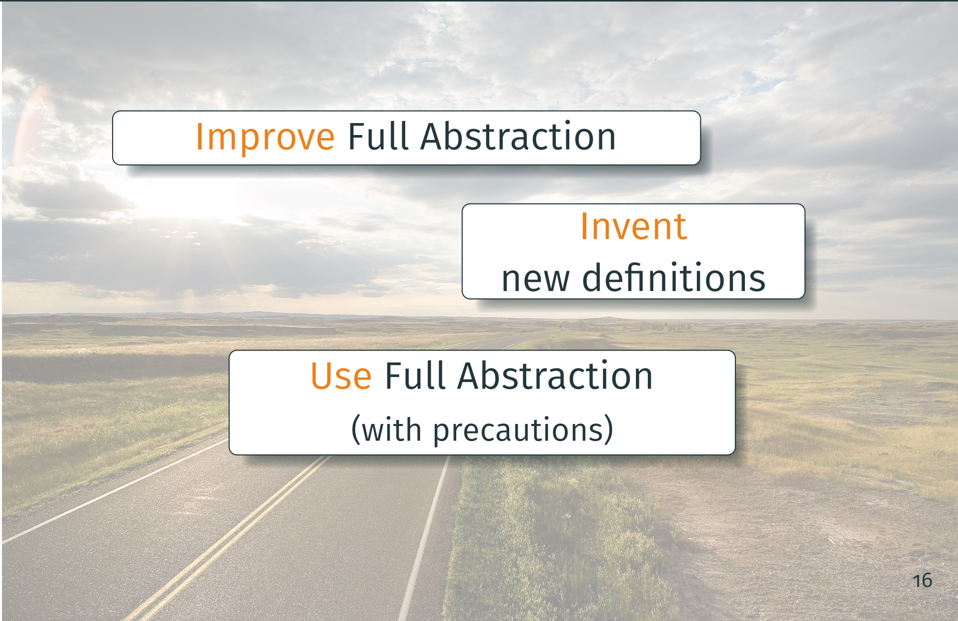
(with precautions)

Perspective on Foundations

Invent
new definitions

Use Full Abstraction
(with precautions)

Perspective on Foundations



Improve Full Abstraction

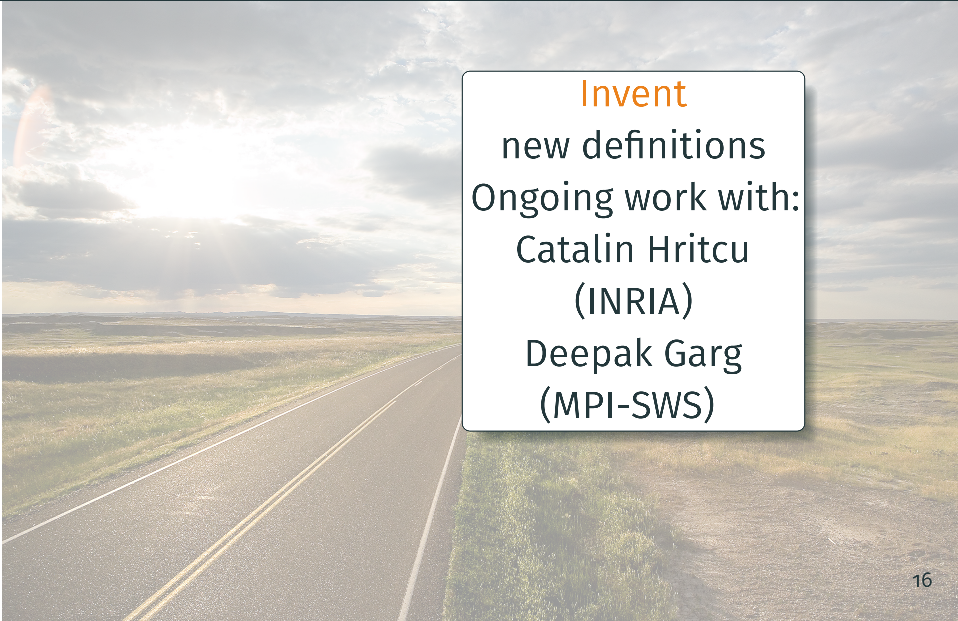
Invent
new definitions

Use Full Abstraction
(with precautions)

Perspective on Foundations

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

$$\mathbf{P1} \;\simeq_{ctx}\; \mathbf{P2}$$

$$\Updownarrow$$

$$[\![\mathbf{P1}]\!]^{\mathsf{S}}_{\mathsf{T}} \simeq_{ctx} [\![\mathbf{P2}]\!]^{\mathsf{S}}_{\mathsf{T}}$$

Proving FAC

$$\mathbf{P1} \quad \simeq_{ctx} \quad \mathbf{P2}$$

$$\Uparrow$$

$$[\![\mathbf{P1}]\!]_{\mathsf{T}}^{\mathbf{S}} \simeq_{ctx} [\![\mathbf{P2}]\!]_{\mathsf{T}}^{\mathbf{S}}$$

Proving FAC

$$\mathbf{P1} \simeq_{ctx} \mathbf{P2}$$

$$\Downarrow$$

$$[\![\mathbf{P1}]\!]^{\mathrm{S}}_{\mathsf{T}} \simeq_{ctx} [\![\mathbf{P2}]\!]^{\mathrm{S}}_{\mathsf{T}}$$

Proving FAC

$$\mathbf{P1} \simeq_{ctx} \mathbf{P2}$$

$$\Downarrow$$

$$\forall \mathbb{C}. \mathbb{C}[\llbracket \mathbf{P1} \rrbracket^{\mathrm{S}}_{\mathrm{T}}]\Downarrow \iff \mathbb{C}[\llbracket \mathbf{P2} \rrbracket^{\mathrm{S}}_{\mathrm{T}}]\Downarrow$$

Proving FAC

$$\mathbf{P1} \quad \simeq_{ctx} \quad \mathbf{P2}$$

$$\Downarrow$$

$$[\![\mathbf{P1}]\!]^{\mathbf{S}}_{\mathsf{T}} \;\stackrel{\mathsf{T}}{=}\; [\![\mathbf{P2}]\!]^{\mathbf{S}}_{\mathsf{T}}$$

Proving FAC

$$\mathbf{P1} \quad \simeq_{ctx} \quad \mathbf{P2}$$

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

$$[\![\mathbf{P1}]\!]_{\mathsf{T}} \simeq [\![\mathbf{P2}]\!]_{\mathsf{T}}$$

Proving FAC

$$\mathbf{P1} \ \simeq_{ctx} \ \mathbf{P2}$$

$$\Downarrow$$

$$[\![\mathbf{P1}]\!]^{\mathsf{S}}_{\mathsf{T}} \approx [\![\mathbf{P2}]\!]^{\mathsf{S}}_{\mathsf{T}}$$

Proving FAC

$$\mathbf{P1} \simeq_{ctx} \mathbf{P2}$$

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

Bugliesi *et al.*'07

Adao *et al.*'06

Fournet *et al.*'13

$$[\![\mathbf{P1}]\!]_{\mathsf{T}} \approx [\![\mathbf{P2}]\!]_{\mathsf{T}}$$

Proving FAC

$$\mathbf{P1} \simeq_{ctx} \mathbf{P2}$$

$$\Downarrow$$

$$[\![\mathbf{P1}]\!]^{\mathsf{S}}_{\mathsf{T}} \sim_{\mathsf{n}} [\![\mathbf{P2}]\!]^{\mathsf{S}}_{\mathsf{T}}$$

Proving FAC

$$\mathbf{P1} \quad \simeq_{ctx} \quad \mathbf{P2}$$

$$\Updownarrow$$

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

$$[\![\mathbf{P1}]\!]^{\mathsf{S}}_{\mathsf{T}} \sim_{\mathsf{n}} [\![\mathbf{P2}]\!]^{\mathsf{S}}_{\mathsf{T}}$$

Proving FAC with Logical Relations

$$\mathbf{P1} \simeq_{ctx} \mathbf{P2}$$

approx. compiler security

$$[\![\mathbf{P1}]\!]^{\mathsf{S}}_{\mathsf{T}} \overset{?}{\simeq}_{ctx} [\![\mathbf{P2}]\!]^{\mathsf{S}}_{\mathsf{T}}$$

Proving FAC with Logical Relations

$$\mathbf{P1} \simeq_{ctx} \mathbf{P2}$$

$$\mathbb{C}\left[[\![\mathbf{P1}]\!]^{\mathsf{S}}_{\mathsf{T}}\right]\Downarrow_{\mathsf{n}} \quad \overset{?}{\Rightarrow} \quad \mathbb{C}\left[[\![\mathbf{P2}]\!]^{\mathsf{S}}_{\mathsf{T}}\right]\Downarrow_{\_}$$

$$[\![\mathbf{P1}]\!]^{\mathsf{S}}_{\mathsf{T}} \overset{?}{\simeq}_{ctx} [\![\mathbf{P2}]\!]^{\mathsf{S}}_{\mathsf{T}}$$

approx. compiler security

Proving FAC with Logical Relations

$$\mathbf{P1} \simeq_{ctx} \mathbf{P2}$$

$\langle\langle \mathbb{C} \rangle\rangle_{\mathbf{n}} \sim_n \mathbb{C}$

$\mathbf{P1} \sim_{\_} [\![\mathbf{P1}]\!]^{\mathsf{S}}_{\mathsf{T}}$

$\Uparrow$ (1)

$$\mathbb{C}\left[[\![\mathbf{P1}]\!]^{\mathsf{S}}_{\mathsf{T}}\right]\Downarrow_{\mathsf{n}} \overset{?}{\Rightarrow} \mathbb{C}\left[[\![\mathbf{P2}]\!]^{\mathsf{S}}_{\mathsf{T}}\right]\Downarrow_{\_}$$

$$[\![\mathbf{P1}]\!]^{\mathsf{S}}_{\mathsf{T}} \overset{?}{\simeq}_{ctx} [\![\mathbf{P2}]\!]^{\mathsf{S}}_{\mathsf{T}}$$

$\Downarrow$ approx. compiler security

Proving FAC with Logical Relations

$$\mathbf{P1} \simeq_{ctx} \mathbf{P2}$$

$$\langle\langle \mathbb{C} \rangle\rangle_{\mathbf{n}} [\mathbf{P1}] \Downarrow_{\_}$$

$$\langle\langle \mathbb{C} \rangle\rangle_{\mathbf{n}} \sim_n \mathbb{C}$$

$$\mathbf{P1} \sim_{\_} [\![\mathbf{P1}]\!]^{\mathbf{S}}_{\mathsf{T}}$$

(1)

$$\mathbb{C}\left[[\![\mathbf{P1}]\!]^{\mathbf{S}}_{\mathsf{T}}\right] \Downarrow_{\mathsf{n}} \overset{?}{\Rightarrow} \mathbb{C}\left[[\![\mathbf{P2}]\!]^{\mathbf{S}}_{\mathsf{T}}\right] \Downarrow_{\_}$$

$$[\![\mathbf{P1}]\!]^{\mathbf{S}}_{\mathsf{T}} \overset{?}{\simeq}_{ctx} [\![\mathbf{P2}]\!]^{\mathbf{S}}_{\mathsf{T}}$$

approx. compiler security

Proving FAC with Logical Relations

$$\mathbf{P1} \simeq_{ctx} \mathbf{P2}$$

$$\langle\!\langle \mathbb{C} \rangle\!\rangle_{\mathbf{n}} [\mathbf{P1}] \Downarrow\_ \underset{(2)}{\Rightarrow} \langle\!\langle \mathbb{C} \rangle\!\rangle_{\mathbf{n}} [\mathbf{P2}] \Downarrow\_$$

$$\langle\!\langle \mathbb{C} \rangle\!\rangle_{\mathbf{n}} \sim_n \mathbb{C}$$

$$\mathbf{P1} \sim\_ [\![\mathbf{P1}]\!]^{\mathsf{S}}_{\mathsf{T}}$$

(1)

$$\mathbb{C}\left[[\![\mathbf{P1}]\!]^{\mathsf{S}}_{\mathsf{T}} \right] \Downarrow_{\mathsf{n}} \overset{?}{\Rightarrow} \mathbb{C}\left[[\![\mathbf{P2}]\!]^{\mathsf{S}}_{\mathsf{T}} \right] \Downarrow\_$$

$$[\![\mathbf{P1}]\!]^{\mathsf{S}}_{\mathsf{T}} \overset{?}{\simeq}_{ctx} [\![\mathbf{P2}]\!]^{\mathsf{S}}_{\mathsf{T}}$$

approx. compiler security

Proving FAC with Logical Relations

$$\mathbf{P1} \simeq_{ctx} \mathbf{P2}$$

$$\langle\langle \mathbb{C} \rangle\rangle_{\mathbf{n}}[\mathbf{P1}] \Downarrow\_ \quad \underset{(2)}{\Rightarrow} \quad \langle\langle \mathbb{C} \rangle\rangle_{\mathbf{n}}[\mathbf{P2}] \Downarrow\_$$

$$\langle\langle \mathbb{C} \rangle\rangle_{\mathbf{n}} \sim_n \mathbb{C} \qquad\qquad \langle\langle \mathbb{C} \rangle\rangle_{\mathbf{n}} \sim\_ \mathbb{C}$$

$$\mathbf{P1} \sim\_ [\![\mathbf{P1}]\!]^{\mathsf{S}}_{\mathsf{T}} \quad (1) \qquad (3) \quad \mathbf{P2} \sim\_ [\![\mathbf{P2}]\!]^{\mathsf{S}}_{\mathsf{T}}$$

$$\mathbb{C}\left[[\![\mathbf{P1}]\!]^{\mathsf{S}}_{\mathsf{T}}\right] \Downarrow_{\mathsf{n}} \quad \overset{?}{\Rightarrow} \quad \mathbb{C}\left[[\![\mathbf{P2}]\!]^{\mathsf{S}}_{\mathsf{T}}\right] \Downarrow\_$$

$$[\![\mathbf{P1}]\!]^{\mathsf{S}}_{\mathsf{T}} \overset{?}{\simeq}_{ctx} [\![\mathbf{P2}]\!]^{\mathsf{S}}_{\mathsf{T}}$$

approx. compiler security

Proving FAC with Logical Relations

$$\mathbf{P1} \simeq_{ctx} \mathbf{P2}$$

$\mathbf{P1} \sim [\![\mathbf{P1}]\!]^{\mathsf{S}}_{\mathsf{T}}$ is obtained with standard techniques

Benton *et al.*'09'10

Hur *et al.*'11

Neis *et al.*'15

$$\mathbb{C}\left[[\![\mathbf{P1}]\!]^{\mathsf{S}}_{\mathsf{T}}\right]\Downarrow_{\mathsf{n}} \overset{?}{\Rightarrow} \mathbb{C}\left[[\![\mathbf{P2}]\!]^{\mathsf{S}}_{\mathsf{T}}\right]\Downarrow_{\_}$$

$$[\![\mathbf{P1}]\!]^{\mathsf{S}}_{\mathsf{T}} \overset{?}{\simeq}_{ctx} [\![\mathbf{P2}]\!]^{\mathsf{S}}_{\mathsf{T}}$$

approx. compiler security

Proving FAC with Logical Relations

$\langle\langle \mathbb{C} \rangle\rangle_{\mathbf{n}} \sim \mathbb{C}$ requires

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

$[\![\mathbf{P1}]\!]^{\mathsf{S}}_{\mathsf{T}} \overset{?}{\simeq}_{ctx} [\![\mathbf{P2}]\!]^{\mathsf{S}}_{\mathsf{T}}$

approx. compiler security

Proving FAC with Logical Relations

$\langle\langle \mathbb{C} \rangle\rangle_{\mathbf{n}} \sim \mathbb{C}$ requires

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

$$[\![\mathbf{P1}]\!]^{\mathsf{S}}_{\mathsf{T}} \overset{?}{\simeq}_{ctx} [\![\mathbf{P2}]\!]^{\mathsf{S}}_{\mathsf{T}}$$

approx. compiler security

Proving FAC with Logical Relations

$\langle\langle \mathbb{C} \rangle\rangle_{\mathbf{n}} \sim \mathbb{C}$ 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

$[\![\mathbf{P1}]\!]^{\mathsf{S}}_{\mathsf{T}} \overset{?}{\simeq}_{ctx} [\![\mathbf{P2}]\!]^{\mathsf{S}}_{\mathsf{T}}$

approx. compiler security

Proving FAC with Logical Relations

$\langle\langle \mathbb{C} \rangle\rangle_{\mathbf{n}} \sim \mathbb{C}$ 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

$[\![\mathbf{P1}]\!]^{\mathsf{S}}_{\mathsf{T}} \overset{?}{\simeq}_{ctx} [\![\mathbf{P2}]\!]^{\mathsf{S}}_{\mathsf{T}}$

approx. compiler security

Proving FAC with Logical Relations

$\langle\langle \mathbb{C} \rangle\rangle_{\mathbf{n}} \sim \mathbb{C}$ 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

$[\![\mathbf{P1}]\!]^{\mathsf{S}}_{\mathsf{T}} \overset{?}{\simeq}_{ctx} [\![\mathbf{P2}]\!]^{\mathsf{S}}_{\mathsf{T}}$

approx. compiler security

Security Architectures for Secure Compilation

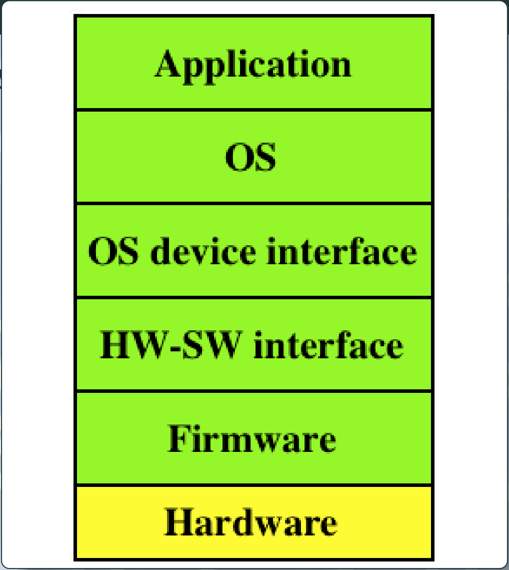
Security Architectures for SC

Security Architectures for SC

Security Architectures:

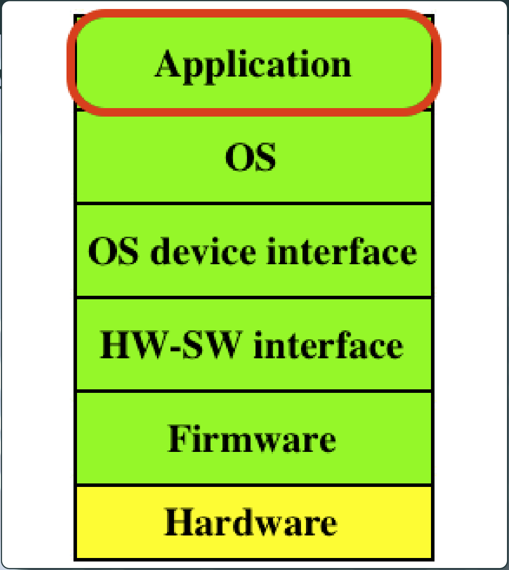
Security Architectures for SC

Security A



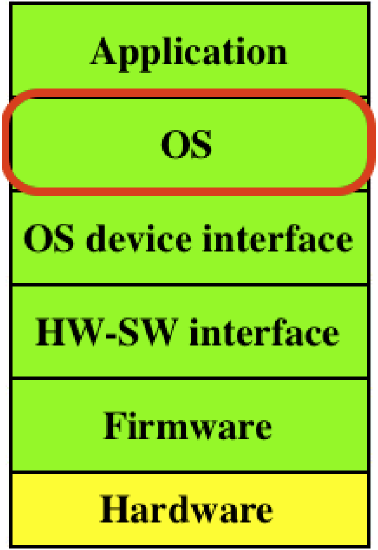
Security Architectures for SC

Security A



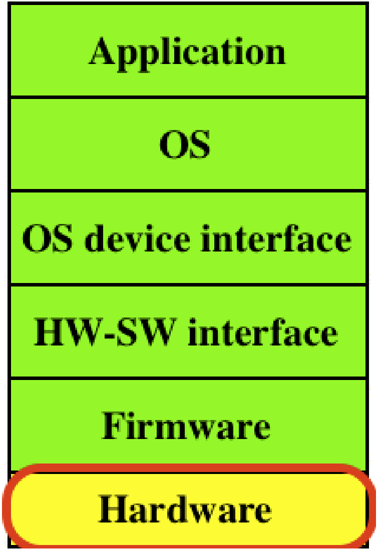
Security Architectures for SC

Security A



Security Architectures for SC

Security A



Security Architectures for SC

Security A

Reduced TCB

Efficiency

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

- Ha[...]
  Ca[...]

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

Capability Machines: Cheri

Capability Machines

- Ha[…]
  Ca[…]

Capability Mantra:

subjects

perform operations

on objects

if they have rights

instructions

Capability Machines: Cheri

Capability Machines

- Ha
  Ca

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

instructions

read/ write/
execute

Capability Machines: Cheri

Capability Machines

- Ha
  Ca

Capability Mantra:
subjects
perform operat
on objects
if they have rig

instructions

read/ write/
execute

address
ranges

Capability Machines: Cheri

Capability Machines

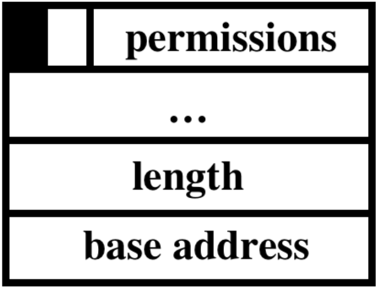
- Hardware support for fine-grained Capabilities
- Cheri (MIPS extension, FPGA) [Woodruff et al'14]

Capability Machines: Cheri

Capability Machines

- Hard
  Capa
- Cheri uff et al'14]

A Cheri capability



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, FPGA) [Woodruff et al'14]

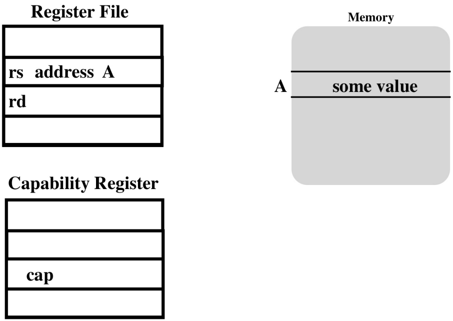
```
load rs rd cap
```

- Tagged memory
- Aligned memory
- Capability registers file
- Capability instructions

Capability Machines: Cheri

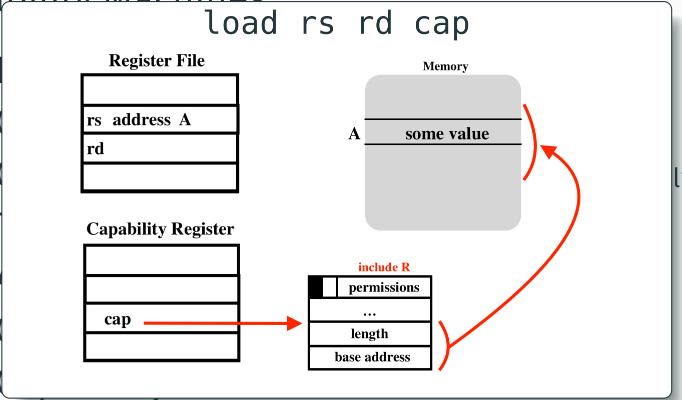
Capability Machines

`load rs rd cap`



Capability Machines: Cheri

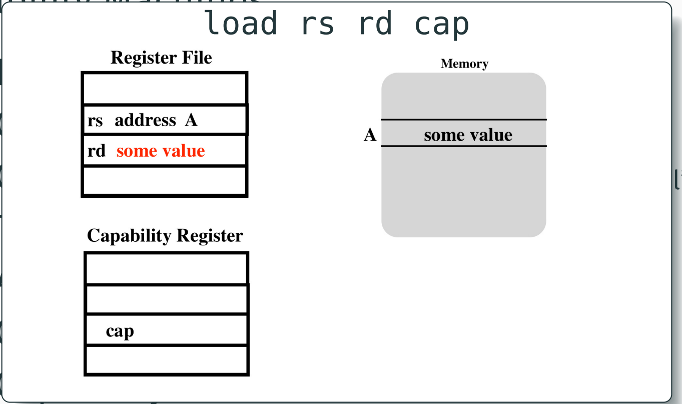
Capability Machines



[’14]

Capability Machines: Cheri

Capability Machines



['14]

Capability Machines: Cheri

Capability Machines

- Hardware support for fine-grained Capabilities
- Ch… et al'14]
- Tag…
- Aligned memory
- Capability registers file
- Capability instructions

Unforgeable capabilities at the hardware level

Capability Machines: Cheri

Capability Machines

- Hardware support for fine-grained Cap[...]
- Ch[...] [...]et al'14]
- Tag[...]
- Ali[...]
- Capability registers file
- Capability instructions

Unforgeable capabilities at the hardware level

Mature: has a FreeBSD port

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

- ide
- wr a
  ca
- ca ods:
  pu
- ca ts:
  sha
- su
  (runtime modification of accesses)

More efficient than existing results

CM and Secure Compilation

- ide
- wra a
  cap
- cap ods:
  pu
- cap ts:
  sha
- sup
  (runtime modification of accesses)

More efficient than existing results

Support unprecedented security paradigms

CM and Secure Compilation

- ide[...]
- wr[...] a
  ca[...]
- ca[...] ds:
  pu[...]
- ca[...] ts:
  sh[...]
- su[...]
  (runtime modification of accesses)

More efficient than existing results

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