Secure Compilation

an extensive introduction

Marco Patrignani

9th October 2017

CISPA
Center for IT-Security, Privacy and Accountability

2017-10-09 Secure Compilation

Secure Compilation

an extensive introduction

Marco Patrignani
9th October 2017

CISPA
Center for IT-Security, Privacy and Accountability

so a language note i presume you all watch got do you watch it with subtitles? turn the m off, do yourselves a favour

this is to say, that if you don't understand stuff or if you have questiosn just ask – even in italian

2017-10-09 Secure Compilation

Collaborators

Collaborators

Contents

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

Compilation

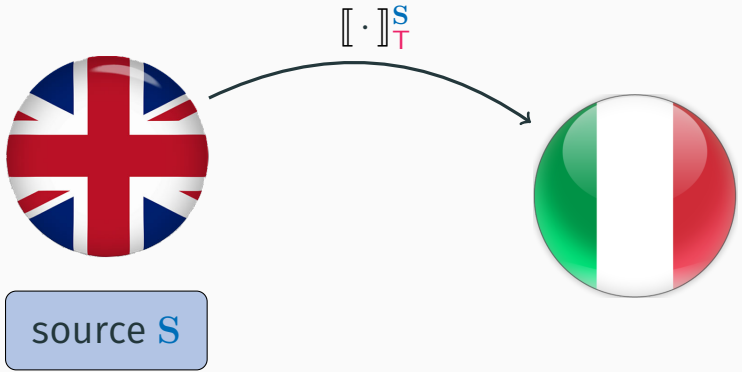
Compilation

source S

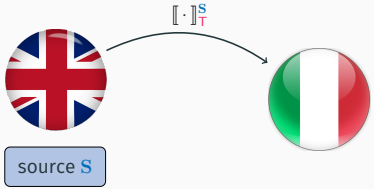
Compilation

source S

Compilation

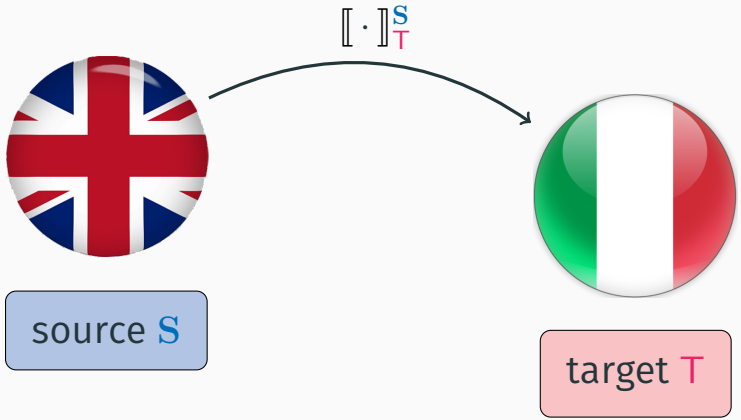


Compilation

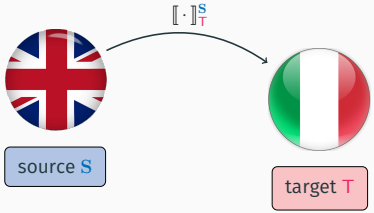


your brains are currently translating from english to italian

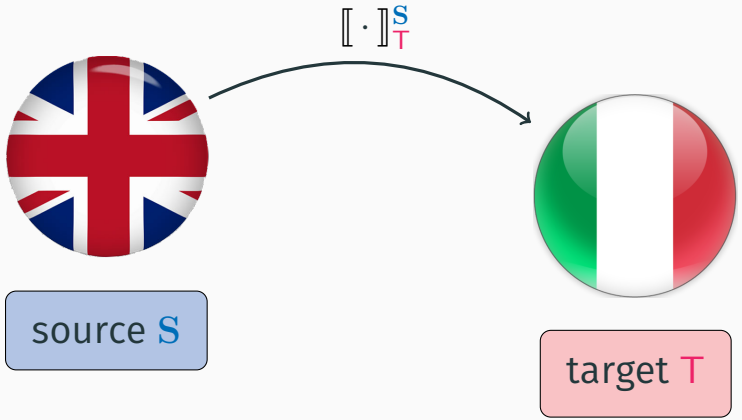
Compilation



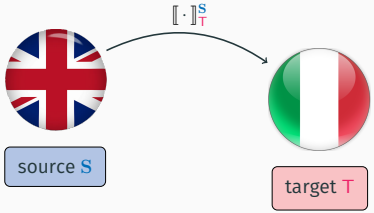
Compilation



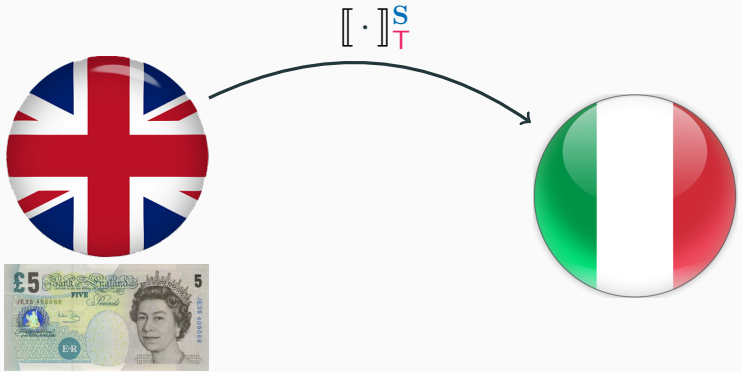
Correct Compilation



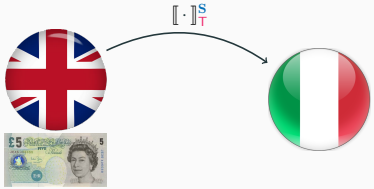
Correct Compilation



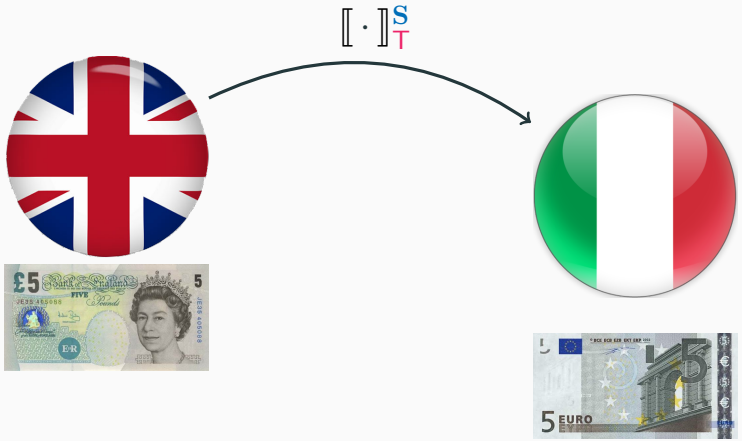
Correct Compilation



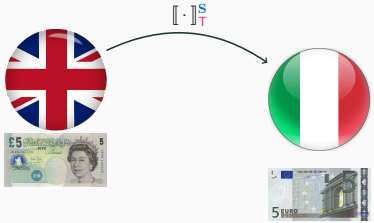
Correct Compilation



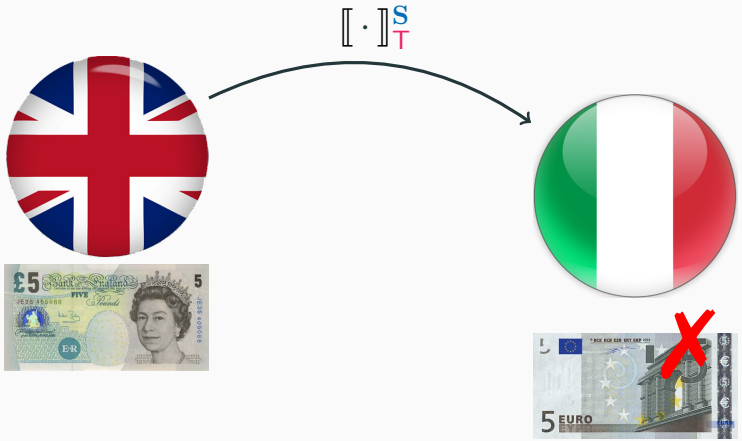
Correct Compilation



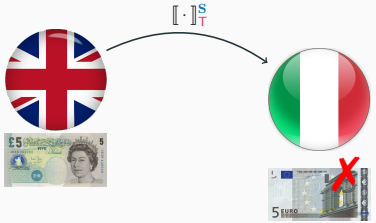
Correct Compilation



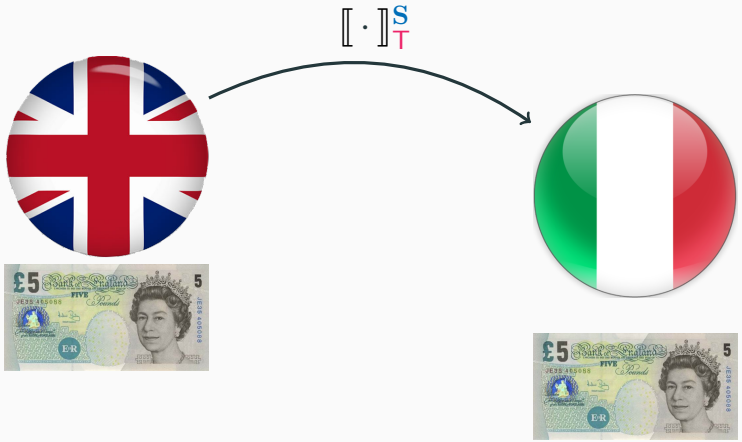
Correct Compilation



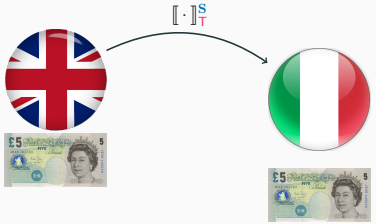
Correct Compilation



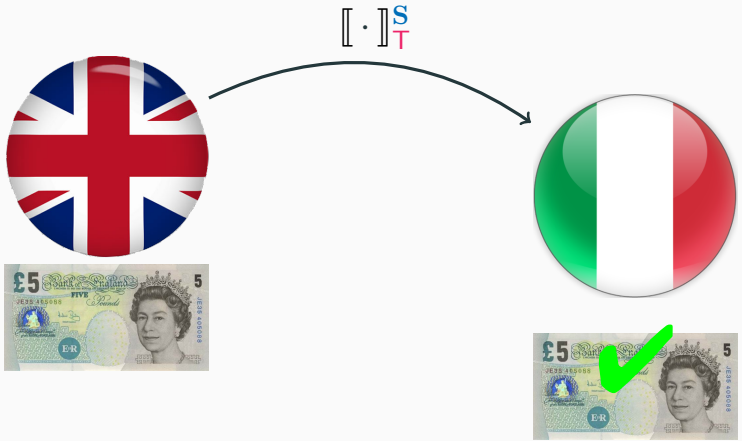
Correct Compilation



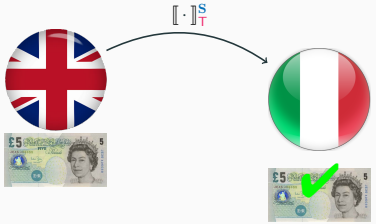
Correct Compilation



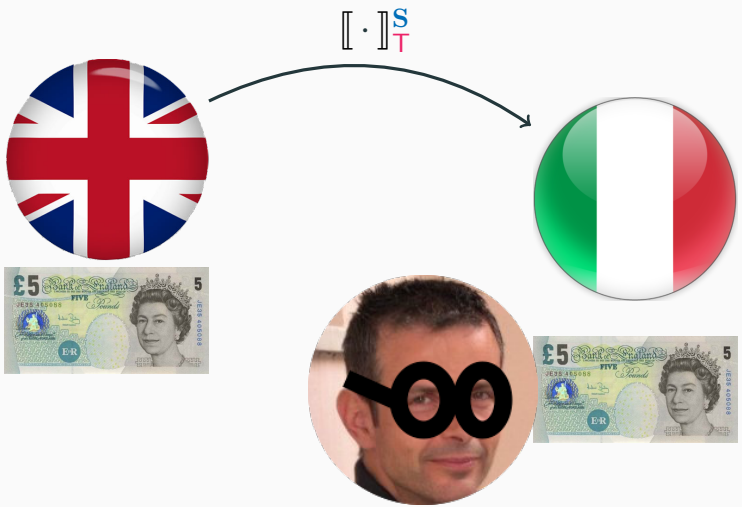
Correct Compilation



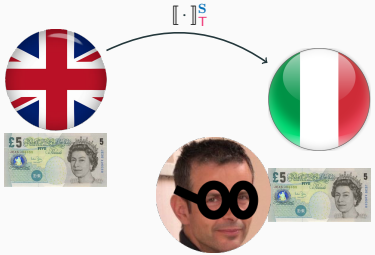
Correct Compilation



Secure Compilation

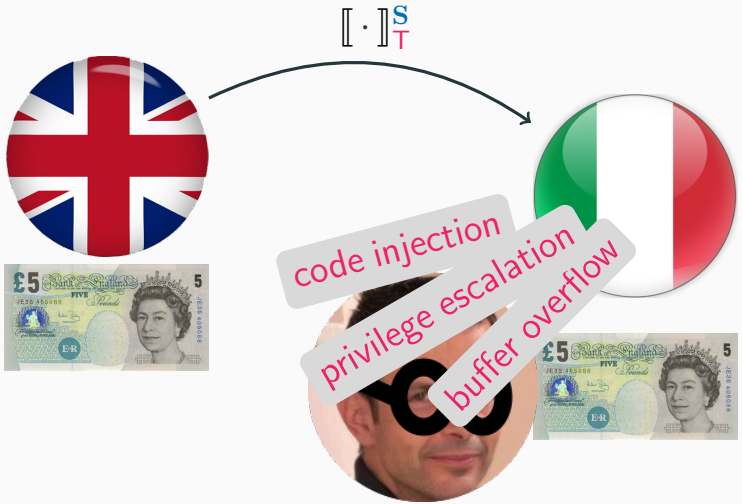


Secure Compilation



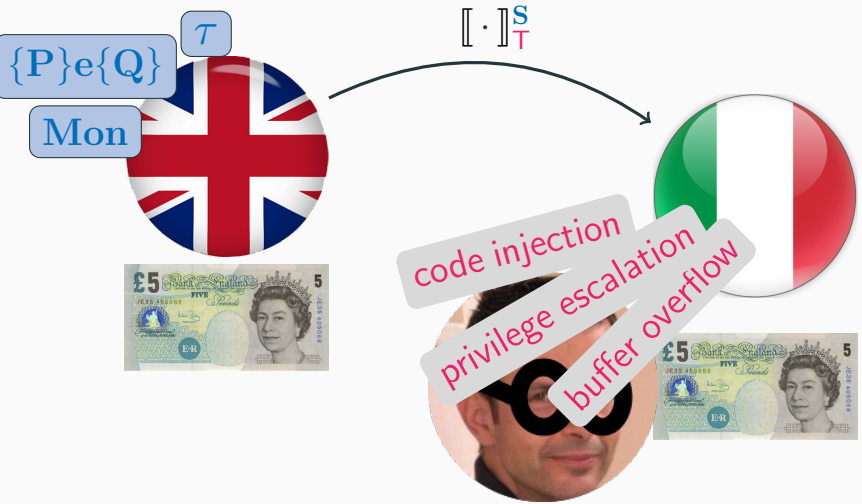
exploits: buffer overflow, privilege escalation

Secure Compilation

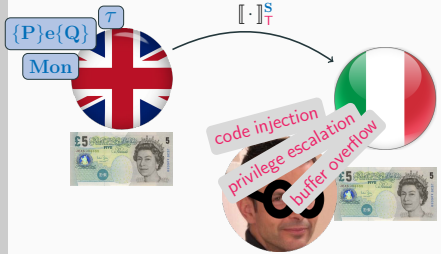


Secure Compilation

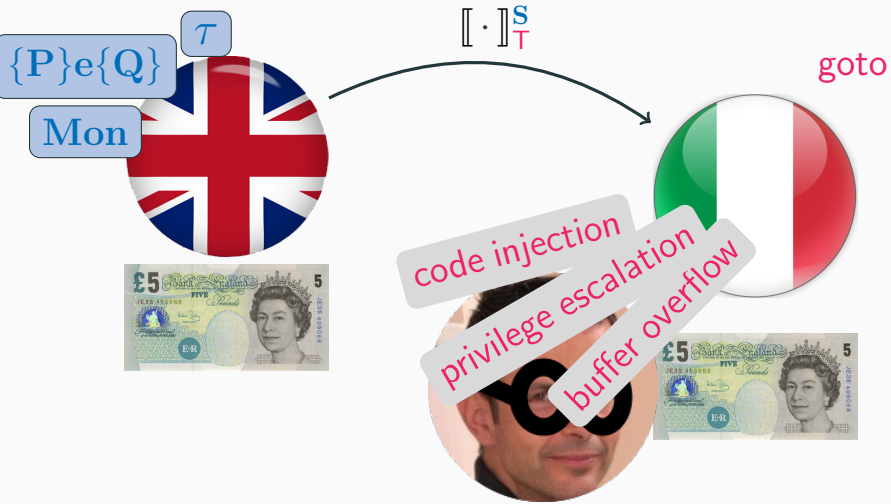
Secure Compilation



Secure Compilation

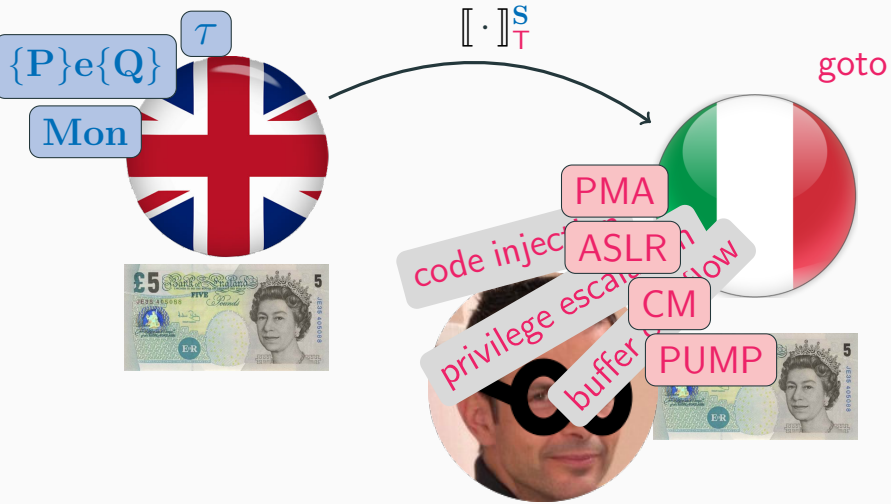


Secure Compilation

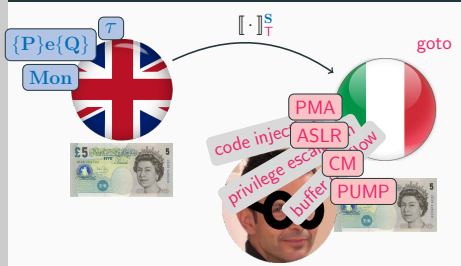


Secure Compilation

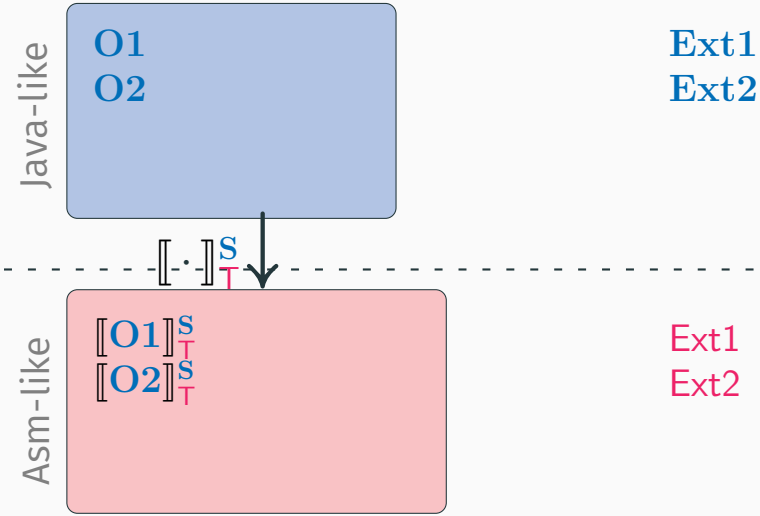
Secure Compilation



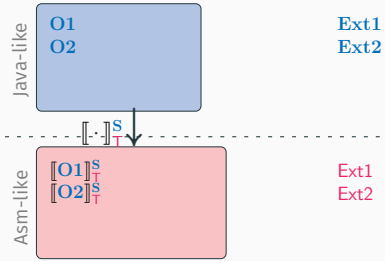
Secure Compilation



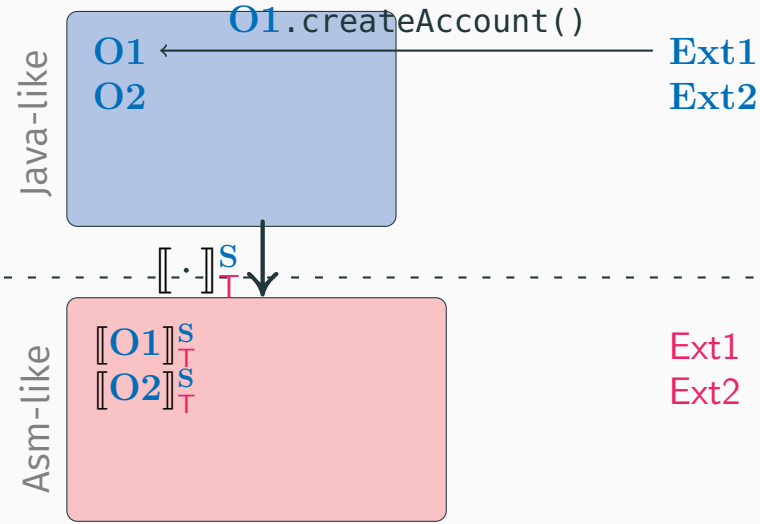
Memory Allocation Issues Patrignani *et al.*'15'16



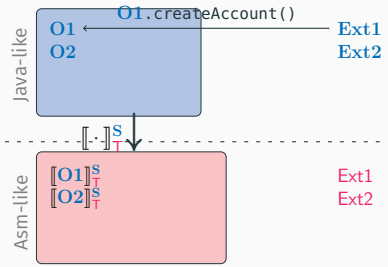
Memory Allocation Issues Patrignani *et al.*'15'16



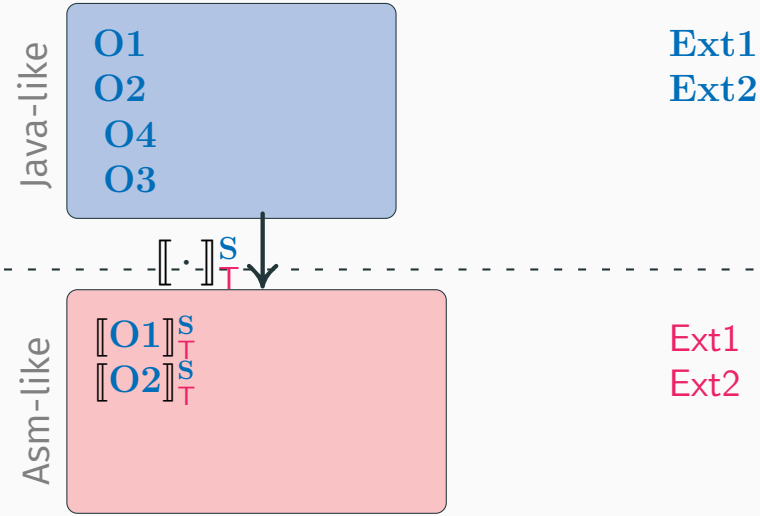
Memory Allocation Issues Patrignani *et al.*'15'16



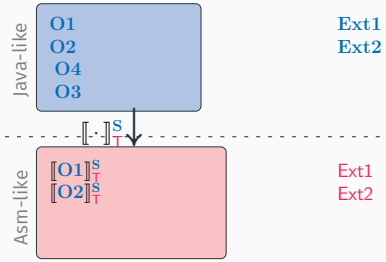
Memory Allocation Issues Patrignani *et al.*'15'16



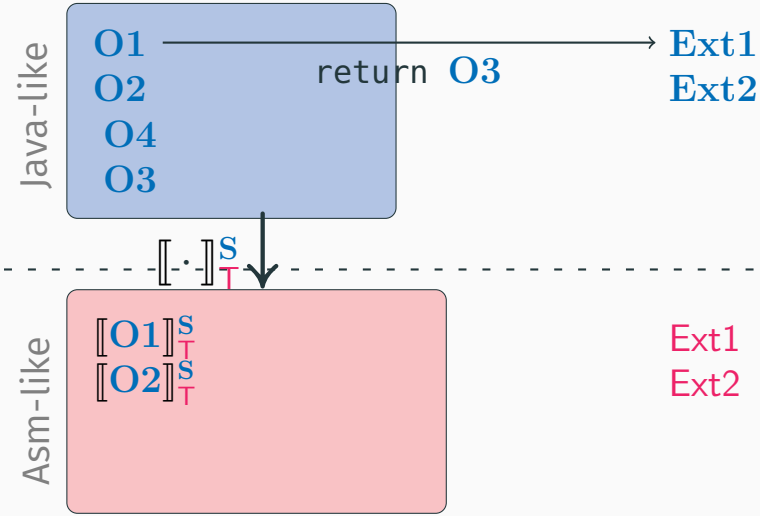
Memory Allocation Issues Patrignani *et al.*'15'16



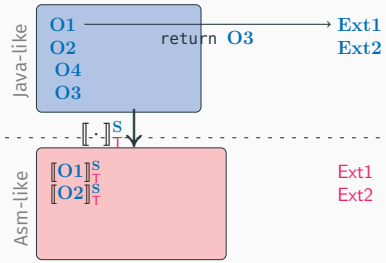
Memory Allocation Issues Patrignani *et al.*'15'16



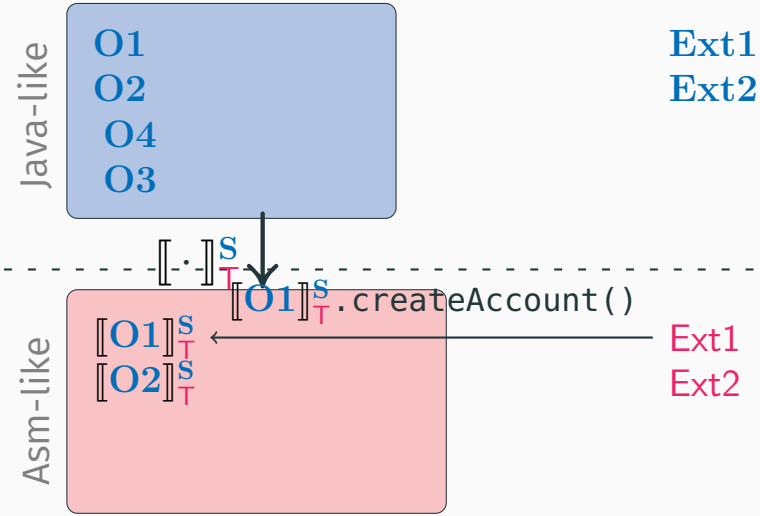
Memory Allocation Issues Patrignani *et al.*'15'16



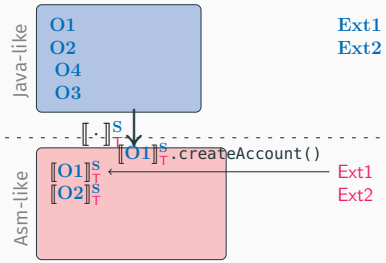
Memory Allocation Issues Patrignani *et al.*'15'16



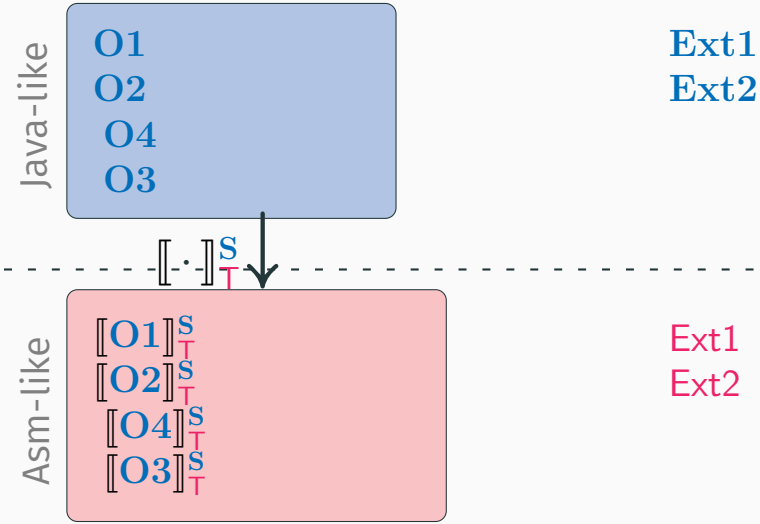
Memory Allocation Issues Patrignani *et al.*'15'16



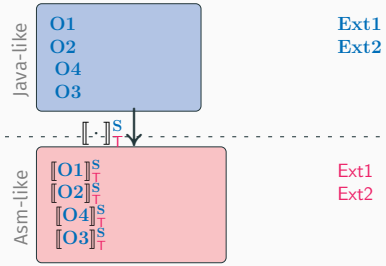
Memory Allocation Issues Patrignani *et al.*'15'16



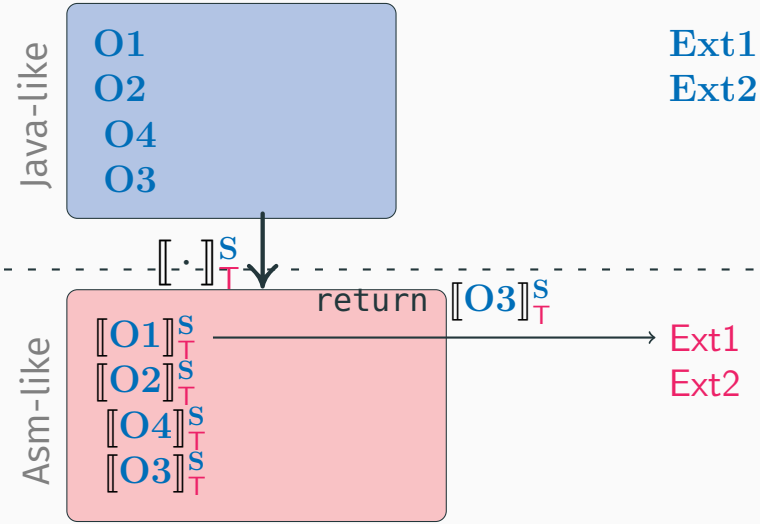
Memory Allocation Issues Patrignani *et al.*'15'16



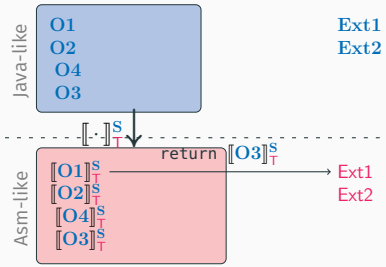
Memory Allocation Issues Patrignani *et al.*'15'16



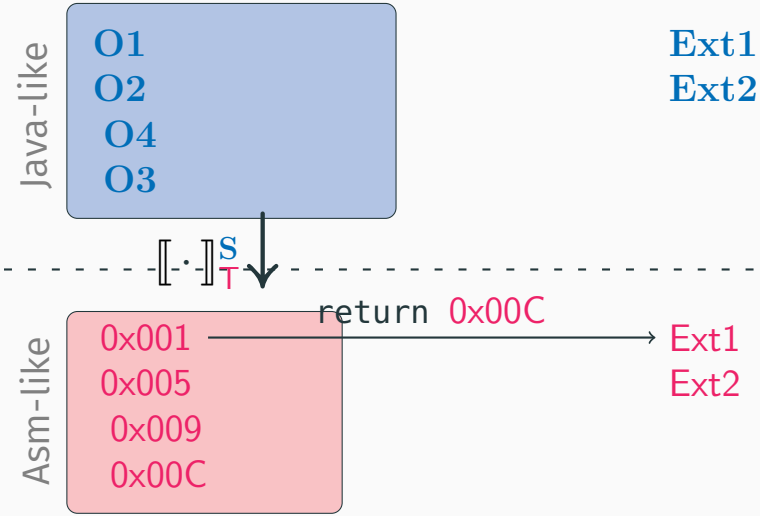
Memory Allocation Issues Patrignani *et al.*'15'16



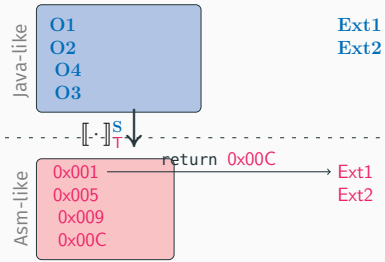
Memory Allocation Issues Patrignani *et al.*'15'16



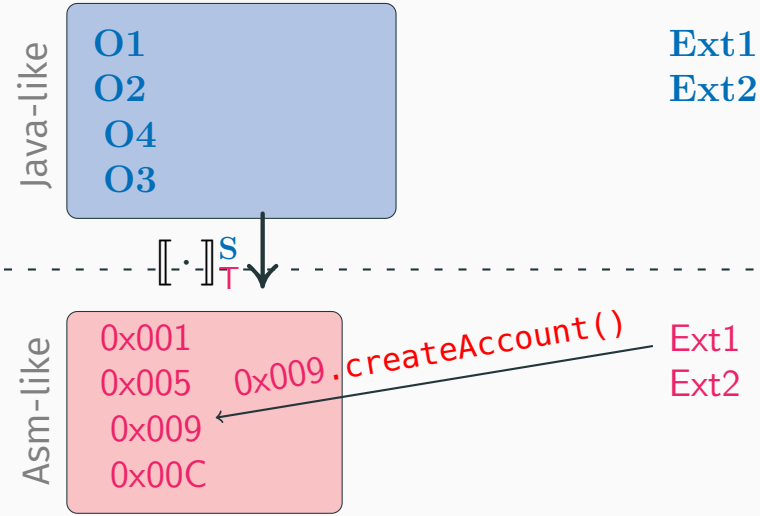
Memory Allocation Issues Patrignani *et al.*'15'16



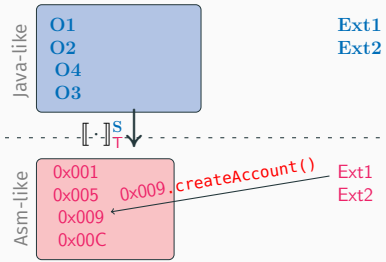
Memory Allocation Issues Patrignani *et al.*'15'16



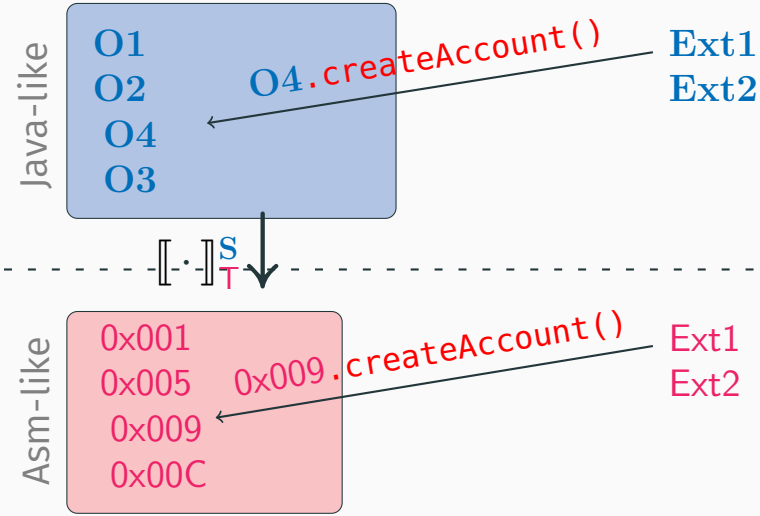
Memory Allocation Issues Patrignani *et al.*'15'16



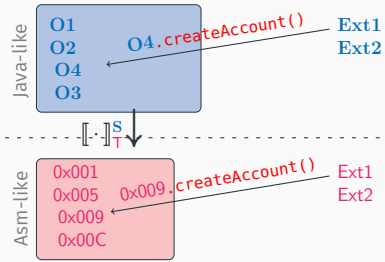
Memory Allocation Issues Patrignani *et al.*'15'16



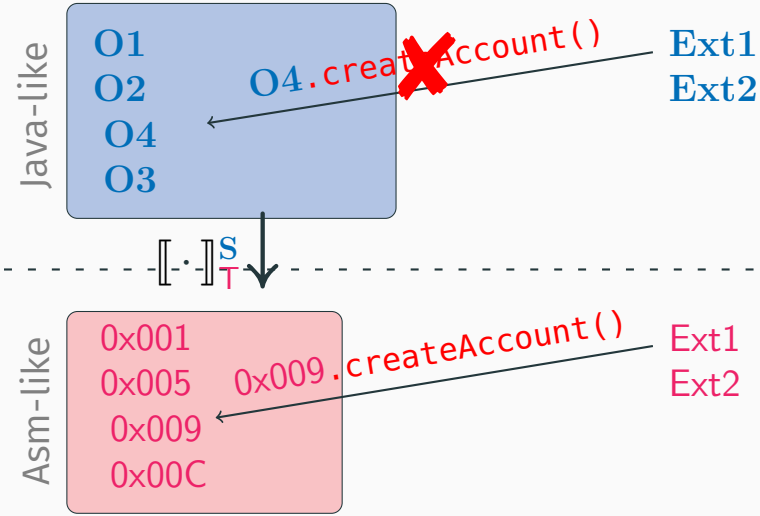
Memory Allocation Issues Patrignani *et al.*'15'16



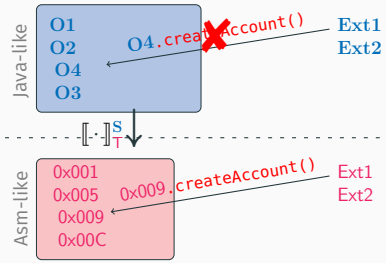
Memory Allocation Issues Patrignani *et al.*'15'16



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

Ext1
Ext2

Ext1
Ext2

Asm-lik

0x005

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

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

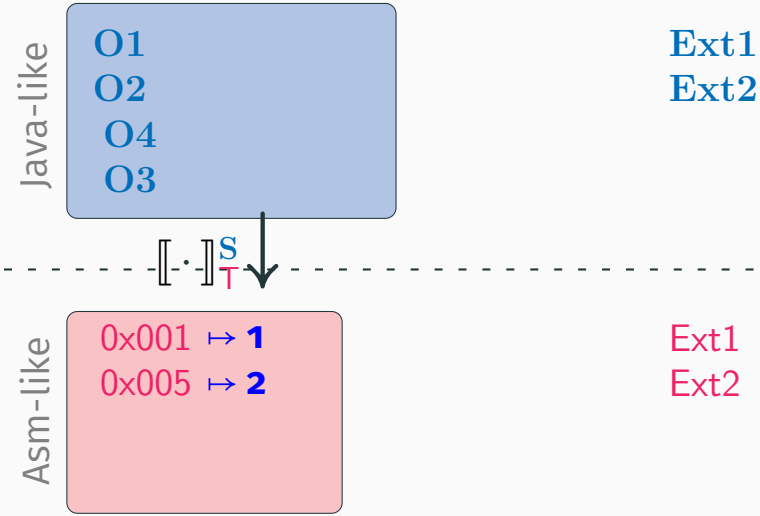
Ext1
Ext2

Ext1
Ext2

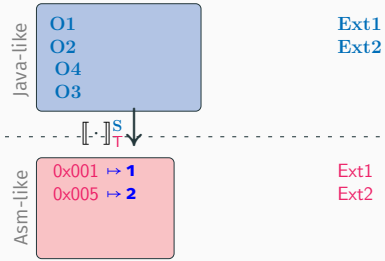
Asm-lik

0x005

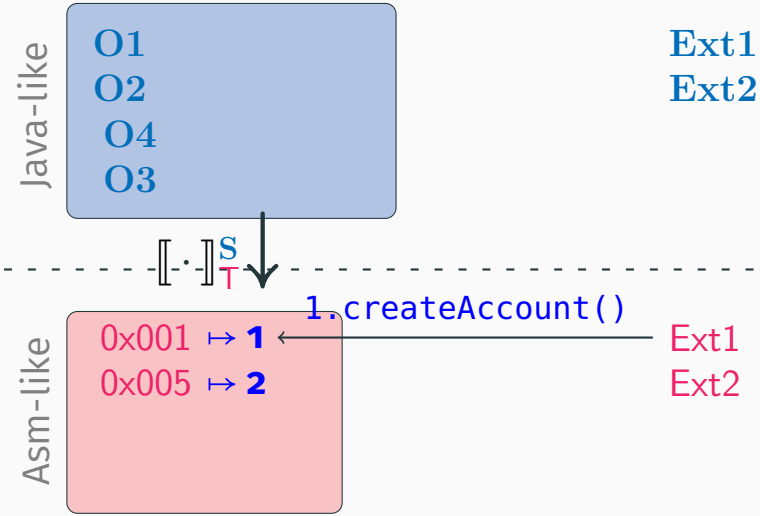
Memory Allocation Issues Patrignani *et al.*'15'16



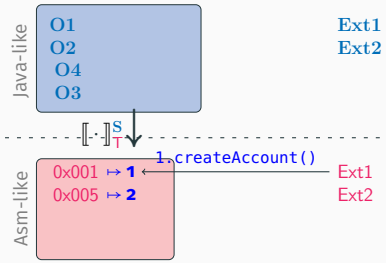
Memory Allocation Issues Patrignani *et al.*'15'16



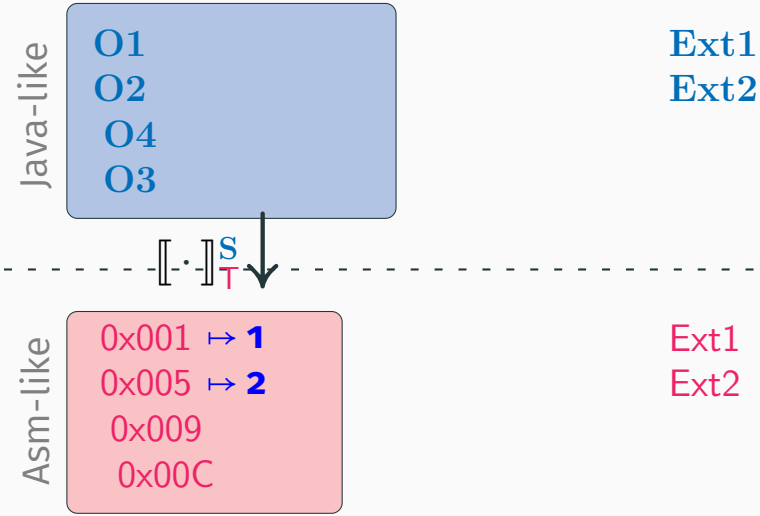
Memory Allocation Issues Patrignani *et al.*'15'16



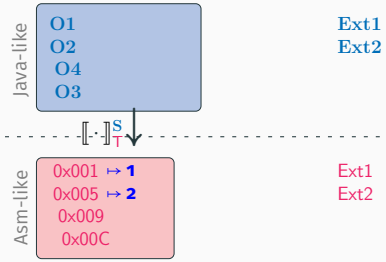
Memory Allocation Issues Patrignani *et al.*'15'16



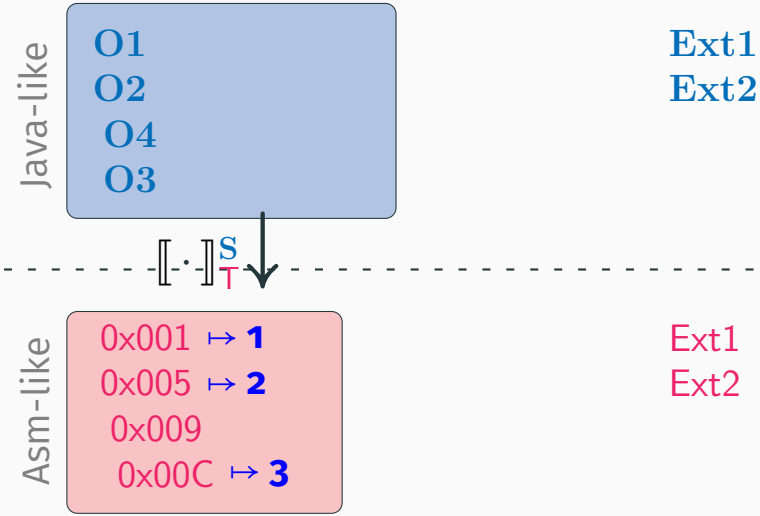
Memory Allocation Issues Patrignani *et al.*'15'16



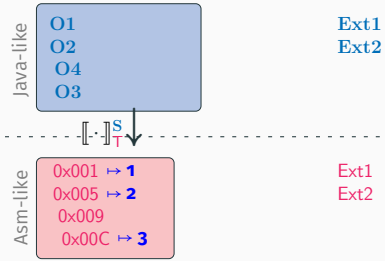
Memory Allocation Issues Patrignani *et al.*'15'16



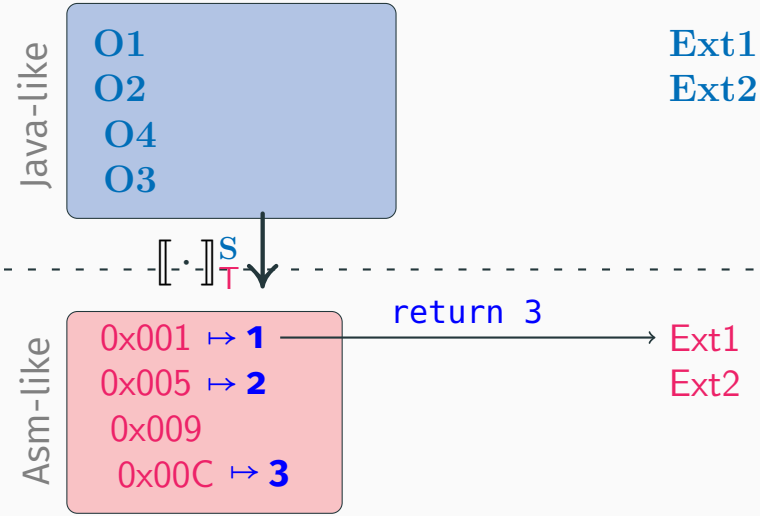
Memory Allocation Issues Patrignani *et al.*'15'16



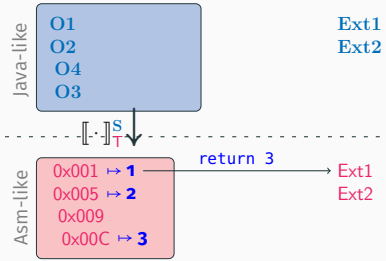
Memory Allocation Issues Patrignani *et al.*'15'16



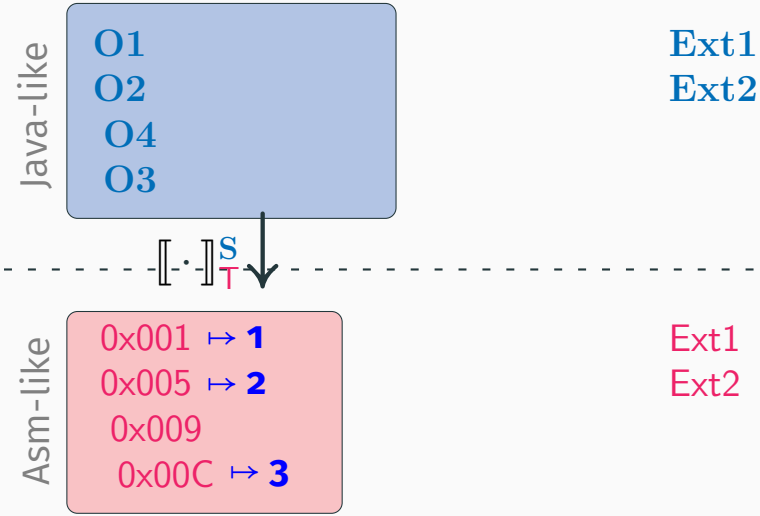
Memory Allocation Issues Patrignani *et al.*'15'16



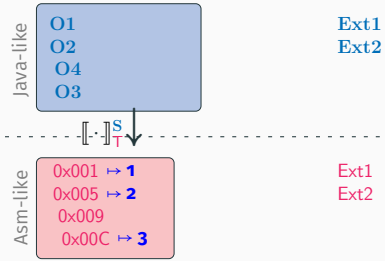
Memory Allocation Issues Patrignani *et al.*'15'16



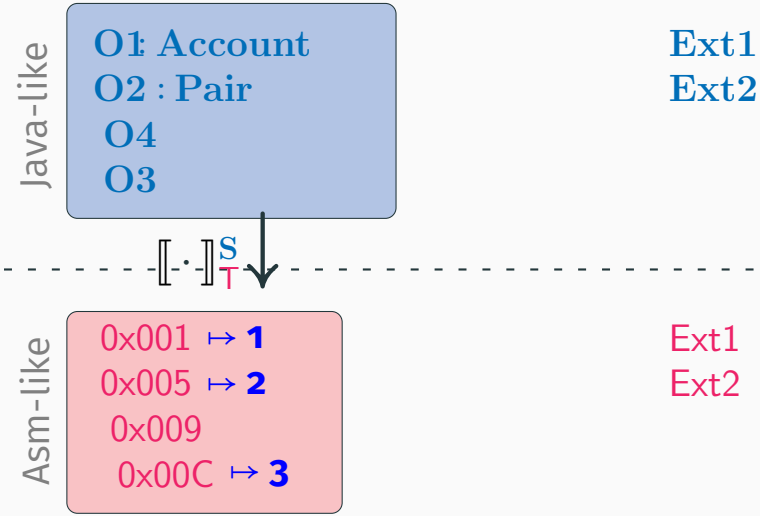
Memory Allocation Issues Patrignani *et al.*'15'16



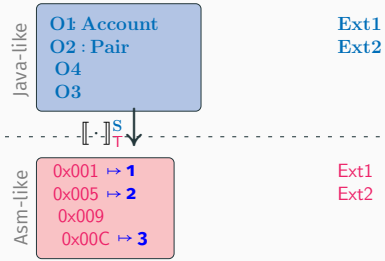
Memory Allocation Issues Patrignani *et al.*'15'16



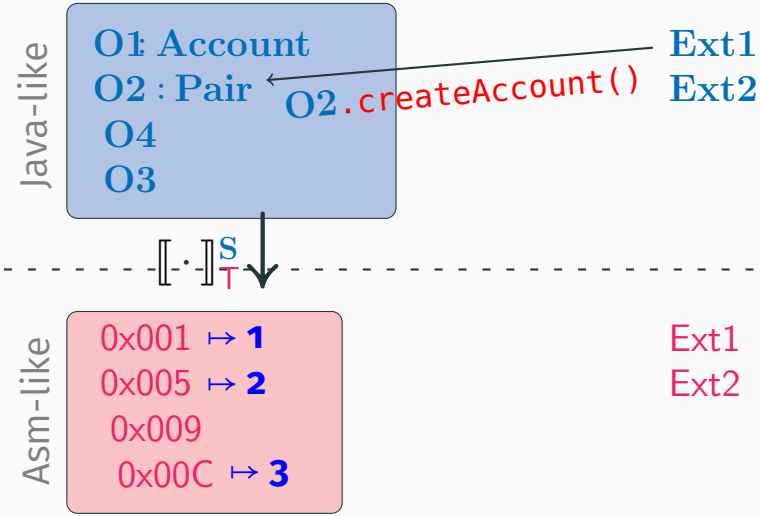
Memory Allocation Issues Patrignani *et al.*'15'16



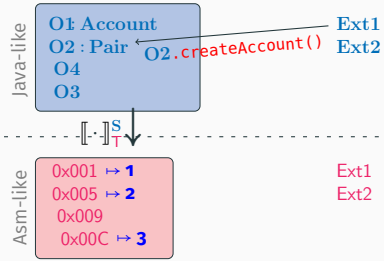
Memory Allocation Issues Patrignani *et al.*'15'16



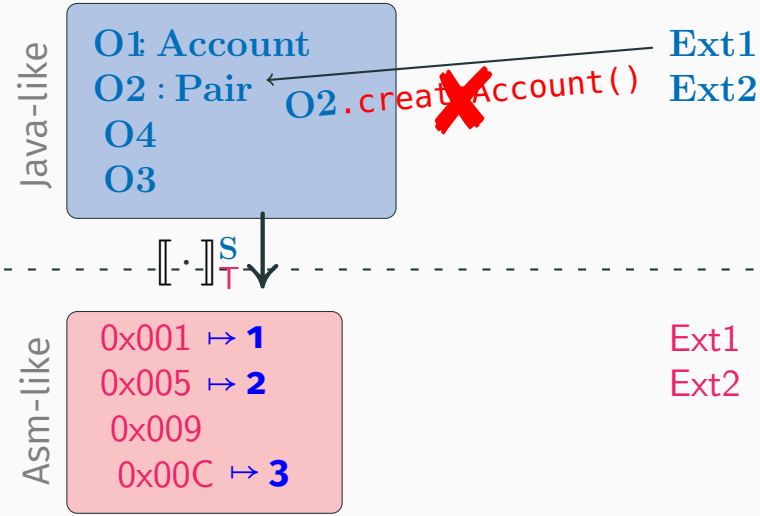
Memory Allocation Issues Patrignani *et al.*'15'16



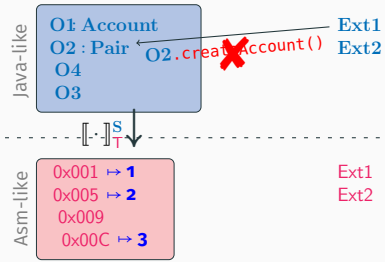
Memory Allocation Issues Patrignani *et al.*'15'16



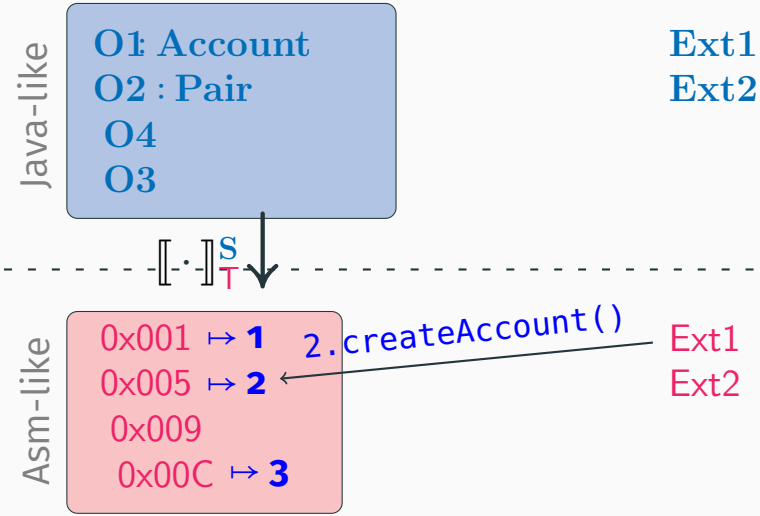
Memory Allocation Issues Patrignani *et al.*'15'16



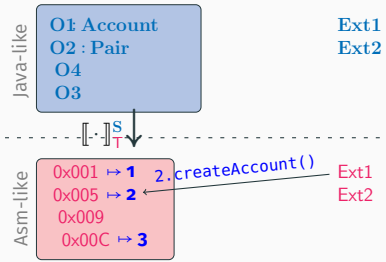
Memory Allocation Issues Patrignani *et al.*'15'16



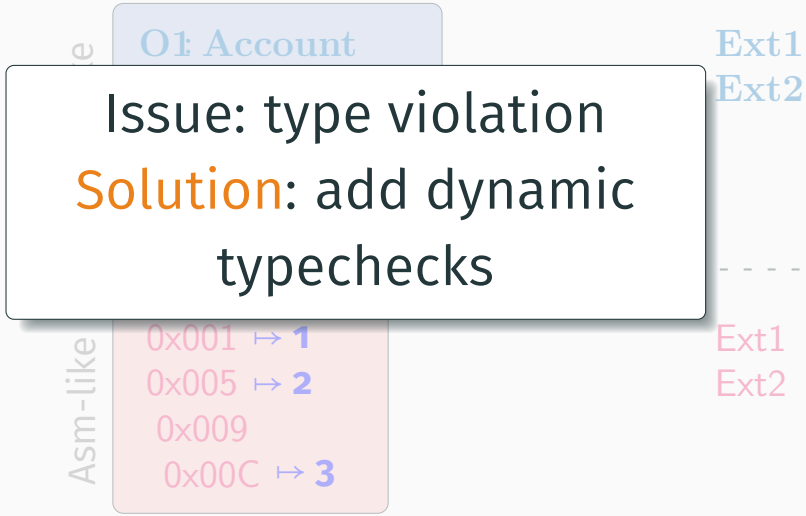
Memory Allocation Issues Patrignani *et al.*'15'16



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

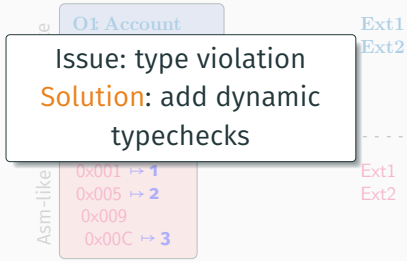


Memory Allocation Issues Patrignani *et al.*'15'16



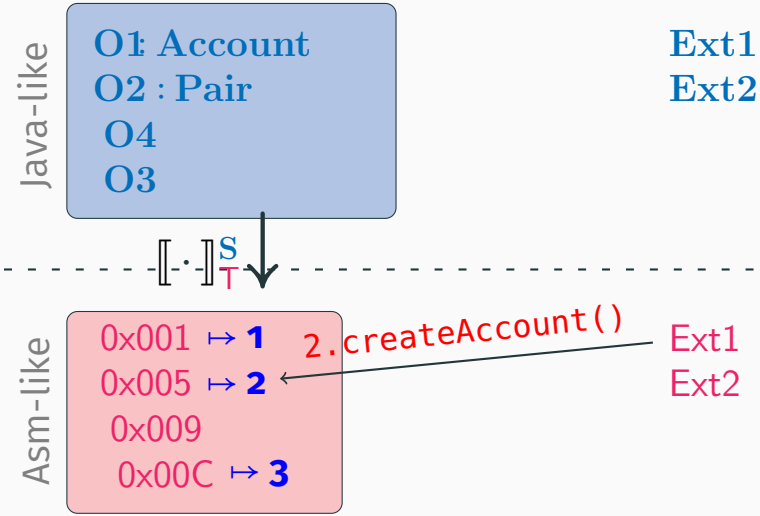
Issue: type violation
Solution: add dynamic typechecks

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

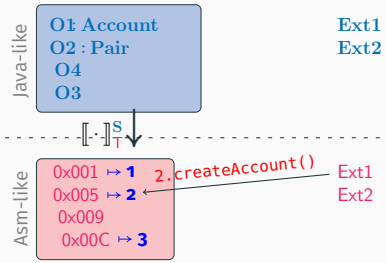


Issue: type violation
Solution: add dynamic typechecks

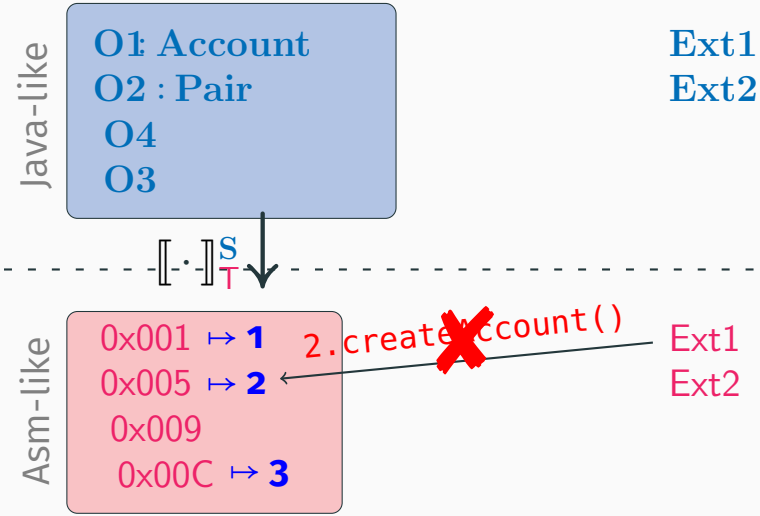
Memory Allocation Issues Patrignani *et al.*'15'16



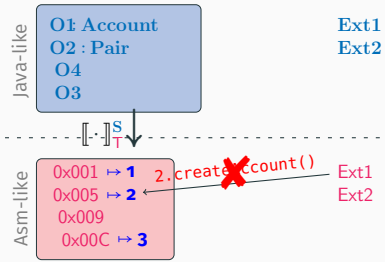
Memory Allocation Issues Patrignani *et al.*'15'16



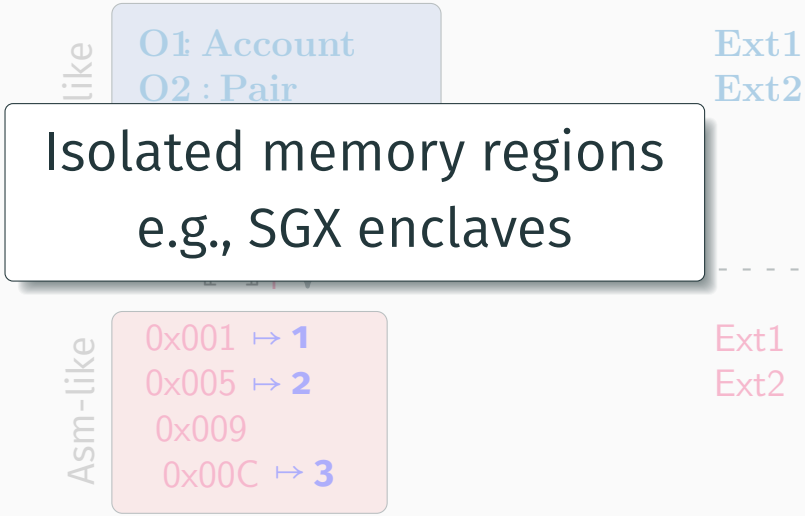
Memory Allocation Issues Patrignani *et al.*'15'16



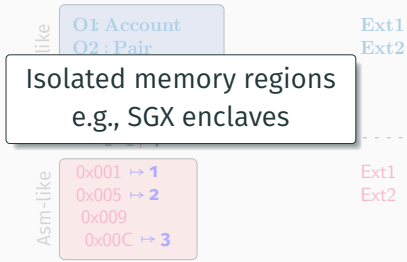
Memory Allocation Issues Patrignani *et al.*'15'16



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



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



efficiency issues are the price to pay

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

- design
- implement
- 

Ext1
Ext2

Ext1
Ext2

Asm-

0x009
0x00C ↦ 3

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

- design
- implement
- 

Ext1
Ext2

Ext1
Ext2

Asm-

0x009
0x00C ↦ 3

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

- design
- implement

-
- ...or?

Ext1
Ext2

Ext1
Ext2

Asm-

0x009
0x00C ↦ 3

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

- design
- implement

-
- ...or?

Ext1
Ext2

Ext1
Ext2

Asm-

0x009
0x00C ↦ 3

Guarantees

- How do we know we are right?

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

- 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

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

- 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

Guarantees

Show
the security implications
of the criterion

Secure Compilation Criteria

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

From the join-calculus to the sjoin-calculus

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

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

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.

From the join-calculus to the sjoin-calculus

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

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

secure compilation is a broad field that received contributions from many researchers, many of which are in this room today.

around the end of the 90's came a line of works that contributed greatly to the idea we have nowadays of Secure Compilation,

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

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 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,*

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

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,

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

many works using FA have been made

and FA has also been studied by Parrow that tells us when it's possible

Fully Abstract Compilation Influence

Fully Abstract Compilation Influence

Fully Abstract Compilation Influence

Why
does Fully Abstract Compilation entail security?

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

Because

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

Compiler Full Abstraction

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

Compiler Full Abstraction

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

1. let's look at FA, informaly

Compiler Full Abstraction

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

$\Updownarrow$

```
loadi r0 1    loadi r0 0
inc r0        addi r0 2
ret r0        ret r0
```


Compiler Full Abstraction

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

$\Updownarrow$

```
loadi r0 1    loadi r0 0
inc r0        addi r0 2
ret r0        ret r0
```

Compiler Full Abstraction

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

$\Updownarrow$

```
loadi r0 1    loadi r0 0
inc r0   =    addi r0 2
ret r0        ret r0
```

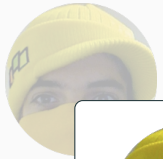
Compiler Full Abstraction

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

$\Updownarrow$

```
loadi r0 1    loadi r0 0
inc r0   =    addi r0 2
ret r0        ret r0
```

Compiler Full Abstraction



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

and

have different powers!

```
inc r0    =    addi r0 2
ret r0         ret r0
```

Compiler Full Abstraction

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

and

have different powers!

```
inc r0    =    addi r0 2
ret r0         ret r0
```

1. note that the observers have different powers!
   the target one is not subject to what happens in the source, so if S has types, he's not subject to them if the language allows him to, he could also jump in mid code!
   but if FA holds, either those bad things cannot arise/are prevented or they are also doable in the source

Why is FAC Secure?

-  is an attacker linking or injecting target code

Why is FAC Secure?

-  is an attacker linking or injecting target code

1. the difference in observers power is at the root of the reasons why FA is a good def for SC
   2 reasons: what attacks are prevented
   what properties are secured

Why is FAC Secure?

-  is an attacker linking or injecting target code

Why is FAC Secure?

-  is an attacker linking or injecting target code

1. First reason is: what is FA protecting from target level attacker

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

1. First reason is: what is FA protecting from target level attacker

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 source constructs
- the co-implied equalities reduce  to 

1. First reason is: what is FA protecting from target level attacker

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?
- is an attacker linking or injecting target code
- 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. 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

1. Second reason: what is FA preserving

Why is FAC Secure?

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?

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?

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?

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.
2.
3.
4.
5.

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
- so the secret does not leak

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

Why is FAC Secure?

1.
2.
3.
4.
5.

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
- 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. 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. [illegible]
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

- FAC preserves these properties

Why is FAC Secure?

1. confidentiality
2. integrity
3. [illegible]
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

- FAC preserves these properties

2017-10-09

Secure Compilation

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]

1. Is that it? Well, no, fa has a number of shortcomings, unfortunately, which we started unveiling only recently.

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]

1. first off, when studied in an untyped target setting, it enforces nothing about modularity

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]

1. secondly, it does not support languages with undefined behaviour

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

1. some security properties are not upheld by it for example a simple declassification policy ... ... though someone may say that it's a matter of changing the kind of equivalence to be preserved

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]

Not All That Glitters is Gold

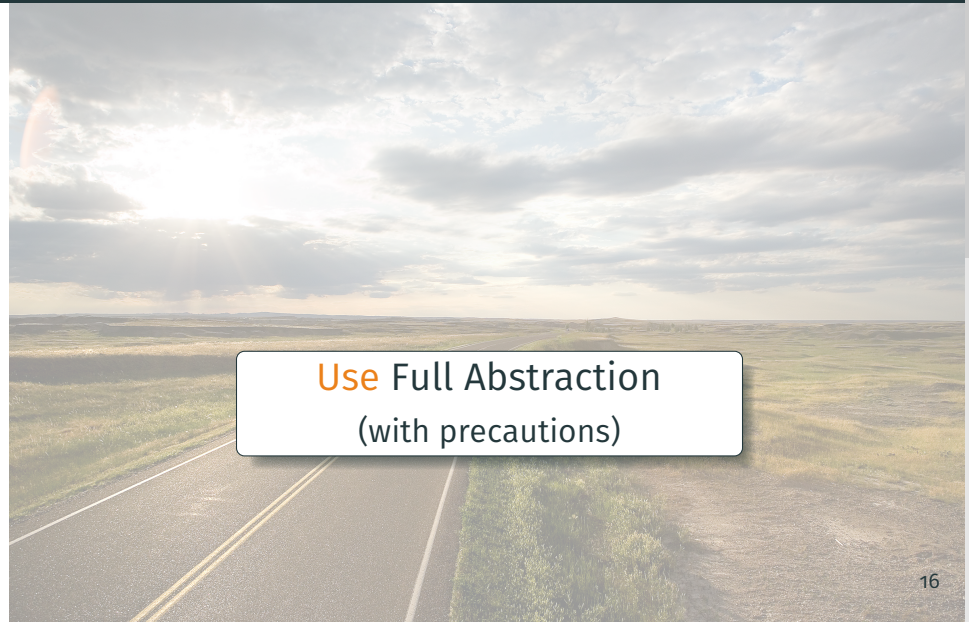
- No support for separate compilation [Patrignani *et al.*'16, Juglaret *et al.*'16]
- No support for undefined behaviour [Juglaret *et al.*'16]
- Costly to enforce
- Preserves hypersafety under certain conditions [Patrignani and Garg '17]

1. finally, the list of security properties it preserves is obtained by example,
   it is not clear if there should be more to that list, or what is definitively not there.
   for this, we have a paper under submission that defines what (a specific form of) fa preserves in terms of classes of hyperproperties, but i'll not go into details here

Perspective on Foundations

Perspective on Foundations

Perspective on Foundations



Perspective on Foundations

Use Full Abstraction
(with precautions)

1. So: what now?
   this has potential.
   we can keep using FA as we've done and in similar settings: it's fine!

Perspective on Foundations

Perspective on Foundations

Invent
new definitions

Use Full Abstraction
(with precautions)

1. So: what now?
   this has potential.
   we can keep using FA as we've done and in similar settings: it's fine!

Perspective on Foundations

Perspective on Foundations

Improve Full Abstraction

Invent
new definitions

Use Full Abstraction
(with precautions)

1. some of us are discussing more specific statements of SC just tailored to specific properties of interest

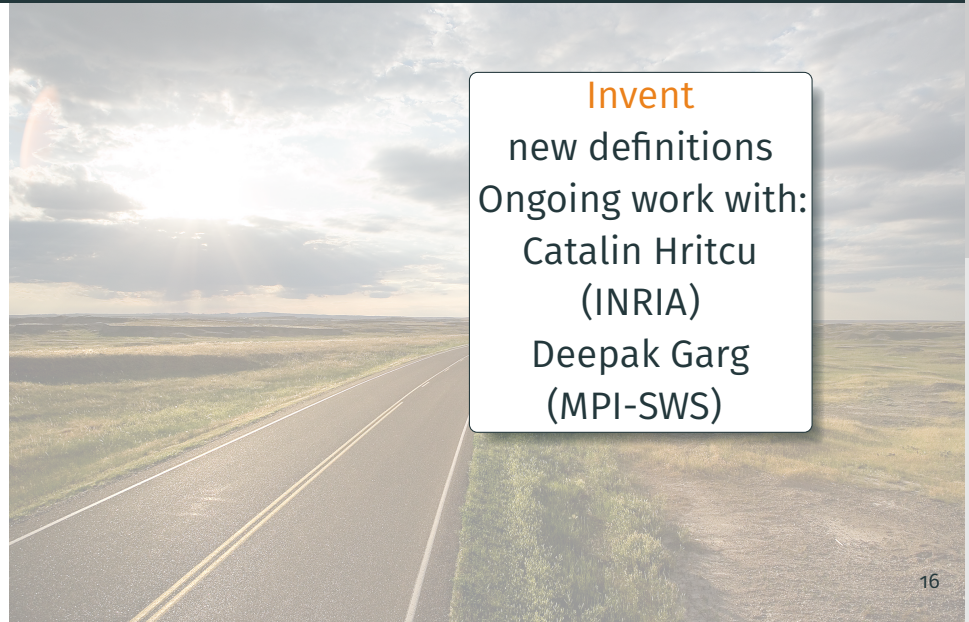
Perspective on Foundations

Perspective on Foundations

Invent
new definitions

1. some of us have already started looking into this: catalin's SCC, our TPC, though they all show a tight connection with fa

Perspective on Foundations



Invent
new definitions
Ongoing work with:
Catalin Hritcu
(INRIA)
Deepak Garg
(MPI-SWS)

Perspective on Foundations

Invent
new definitions
Ongoing work with:
Catalin Hritcu
(INRIA)
Deepak Garg
(MPI-SWS)

1. FA is not bad,
   but it can be improved:
   that's what we're after when we look at the foundations of this field

What More does Secure Compilation Offer?

What More does Secure Compilation Offer?

What More does Secure Compilation Offer?

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

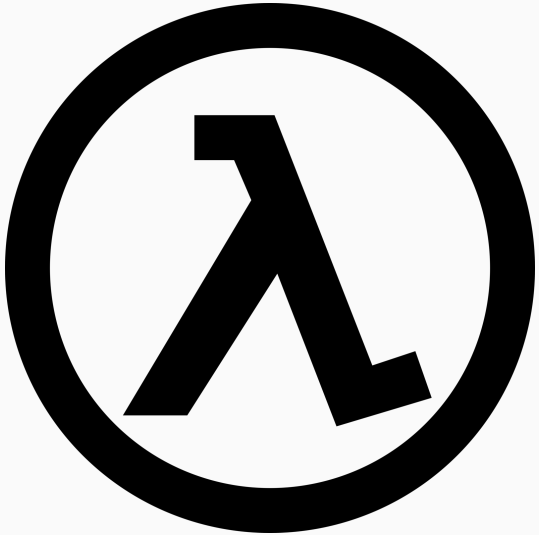
What More does Secure Compilation Offer?

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

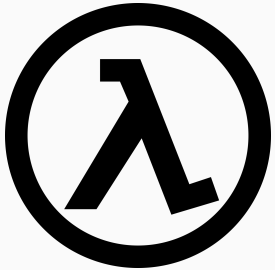
Programming Languages Techniques for Secure Compilation

Programming Languages Techniques for Secure Compilation

What PL Want



What PL Want



1. as we said, sc sits at the verge of PL and security. from the PL perspective, what is there to be challenged for SC?
   In this part of the talk, i'll focus on proof techniques for proving FA and the reason is that other definitions that we've seen also need to capture the additional power of target-level programs/contexts and reason about that. while the "how" it shows up

What PL Want

- better proof techniques

What PL Want

- better proof techniques

Proving FAC

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

$$\Updownarrow$$

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

Proving FAC

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

$$\Updownarrow$$

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

1. to prove fa, we need to prove the 2 parts of its coimplication
   if you don't see the observer don't worry, it's buried in here

Proving FAC

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

$$\Uparrow$$

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

Proving FAC

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

$$\Uparrow$$

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

1. <= is easy, it follows from compiler correctness

Proving FAC

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

$$\Downarrow$$

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

Proving FAC

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

$$\Downarrow$$

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

1. => is the hard one. let us now look a bit more in detail why is this difficult and how can we do this

Proving FAC

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

$$\Downarrow$$

$$\forall \mathbb{C}.\mathbb{C}[[\![\mathbf{P1}]\!]^{\mathbf{S}}_{\mathsf{T}}]\Downarrow \Longleftrightarrow \mathbb{C}[[\![\mathbf{P2}]\!]^{\mathbf{S}}_{\mathsf{T}}]\Downarrow$$

Proving FAC

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

$$\Downarrow$$

$$\forall \mathbb{C}.\mathbb{C}[[\![\mathbf{P1}]\!]^{\mathbf{S}}_{\mathsf{T}}]\Downarrow \Longleftrightarrow \mathbb{C}[[\![\mathbf{P2}]\!]^{\mathbf{S}}_{\mathsf{T}}]\Downarrow$$

1. well, here's the deal: we need to prove something over all target contexts, which is a notoriously complex feat to attain so we do what any sensible person would do, replace ceqT with something else

Proving FAC

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

$$\Downarrow$$

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

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

1. and now there are 2 schools of thought
   - trace equivalence
   - logical relations

Proving FAC

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

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

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

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

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

1. and now there are 2 schools of thought
   - trace equivalence
   - logical relations

Proving FAC

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

$$\Downarrow$$

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

Proving FAC

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

$$\Downarrow$$

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

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

$$[\![\mathbf{P1}]\!]_{\top} \sim [\![\mathbf{P2}]\!]_{\top}$$

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
$[\![\mathbf{P1}]\!]_{\top} \sim [\![\mathbf{P2}]\!]_{\top}$

Proving FAC

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

$$\Downarrow$$

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

Proving FAC

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

$$\Downarrow$$

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

Proving FAC

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

$$\Downarrow$$

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

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

Proving FAC

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

$$\Downarrow$$

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

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

Proving FAC with Logical Relations

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

approx. compiler security

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

Proving FAC with Logical Relations

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

approx. compiler security

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

1. Logical relations: what we need to prove in this case is
   start from P1,relate it and its context to source counterparts
   obtain that P2 terminates in the source
   relate this to P2 terminating in the target

Proving FAC with Logical Relations

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

approx. compiler security

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

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

Proving FAC with Logical Relations

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

approx. compiler security

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

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

1. Logical relations: what we need to prove in this case is
   start from P1,relate it and its context to source counterparts
   obtain that P2 terminates in the source
   relate this to P2 terminating in the target

Proving FAC with Logical Relations

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

$$\langle\!\langle \mathbb{C} \rangle\!\rangle_{\mathbf{n}} \sim_n \mathbb{C}$$
$$\mathbf{P1} \sim_{\_} [\![\mathbf{P1}]\!]^{\mathbf{S}}_{\mathsf{T}}$$

(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

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

$$\langle\!\langle \mathbb{C} \rangle\!\rangle_{\mathbf{n}} \sim_n \mathbb{C}$$
$$\mathbf{P1} \sim_{\_} [\![\mathbf{P1}]\!]^{\mathbf{S}}_{\mathsf{T}}$$

(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

1. Logical relations: what we need to prove in this case is
   start from P1,relate it and its context to source counterparts
   obtain that P2 terminates in the source
   relate this to P2 terminating in the target

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

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

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

$$\langle\!\langle \mathbb{C} \rangle\!\rangle_{\mathbf{n}} \sim_n \mathbb{C}$$
$$\mathbf{P1} \sim_{\_} [\![\mathbf{P1}]\!]^{\mathbf{S}}_{\mathsf{T}}$$

(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

1. Logical relations: what we need to prove in this case is
   start from P1,relate it and its context to source counterparts
   obtain that P2 terminates in the source
   relate this to P2 terminating in the target

Proving FAC with Logical Relations

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

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

$$\langle\!\langle \mathbb{C} \rangle\!\rangle_{\mathbf{n}} \sim_n \mathbb{C}$$
$$\mathbf{P1} \sim_{\_} [\![\mathbf{P1}]\!]^{\mathbf{S}}_{\mathsf{T}}$$

(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

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

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

$$\langle\!\langle \mathbb{C} \rangle\!\rangle_{\mathbf{n}} \sim_n \mathbb{C}$$
$$\mathbf{P1} \sim_{\_} [\![\mathbf{P1}]\!]^{\mathbf{S}}_{\mathsf{T}}$$

(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

1. Logical relations: what we need to prove in this case is
   start from P1,relate it and its context to source counterparts
   obtain that P2 terminates in the source
   relate this to P2 terminating in the target

Proving FAC with Logical Relations

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

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

$$\begin{array}{c} \langle\!\langle \mathbb{C} \rangle\!\rangle_{\mathbf{n}} \sim_{n} \mathbb{C} \\ \mathbf{P1} \sim_{\_} [\![\mathbf{P1}]\!]^{\mathbf{S}}_{\mathsf{T}} \end{array} \quad \Uparrow (1) \qquad (3) \Downarrow \quad \begin{array}{c} \langle\!\langle \mathbb{C} \rangle\!\rangle_{\mathbf{n}} \sim_{\_} \mathbb{C} \\ \mathbf{P2} \sim_{\_} [\![\mathbf{P2}]\!]^{\mathbf{S}}_{\mathsf{T}} \end{array}$$

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

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

approx. compiler security

Proving FAC with Logical Relations

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

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

$$\begin{array}{c} \langle\!\langle \mathbb{C} \rangle\!\rangle_{\mathbf{n}} \sim_{n} \mathbb{C} \\ \mathbf{P1} \sim_{\_} [\![\mathbf{P1}]\!]^{\mathbf{S}}_{\mathsf{T}} \end{array} \quad \Uparrow (1) \qquad (3) \Downarrow \quad \begin{array}{c} \langle\!\langle \mathbb{C} \rangle\!\rangle_{\mathbf{n}} \sim_{\_} \mathbb{C} \\ \mathbf{P2} \sim_{\_} [\![\mathbf{P2}]\!]^{\mathbf{S}}_{\mathsf{T}} \end{array}$$

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

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

approx. compiler security

1. Logical relations: what we need to prove in this case is
   start from P1,relate it and its context to source counterparts
   obtain that P2 terminates in the source
   relate this to P2 terminating in the target

Proving FAC with Logical Relations

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

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

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

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

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

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

Benton *et al.*'09'10

Hur *et al.*'11

Neis *et al.*'15

approx. compiler security

Proving FAC with Logical Relations

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

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

Benton *et al.*'09'10

Hur *et al.*'11

Neis *et al.*'15

approx. compiler security

1. so we need to define a relation for normal terms for relating programs

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

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

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

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

approx. compiler security

1. need to define a pseudo type and a relation at that pseudotype for back-translated terms

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

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

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

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

approx. compiler security

1. for example, we proved a fa compiler from stlc+fix to ulc

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

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

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

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

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

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

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

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

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

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

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

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

approx. compiler security

Security Architectures for Secure Compilation

Security Architectures for Secure Compilation

Security Architectures for SC

Security Architectures for SC

Security Architectures for SC

Security Architectures:

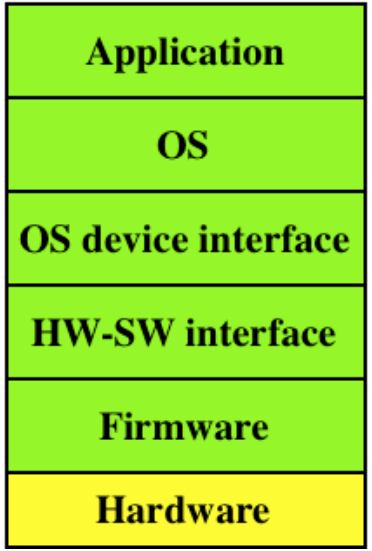
Security Architectures for SC

Security Architectures:

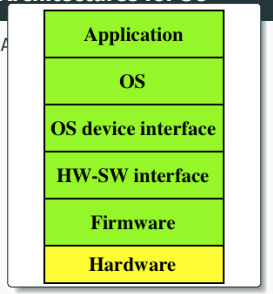
1. albeit formally, we devise compilers and we want them to be written and used.
   so let's change perspective and after looking at how to ensure that these compilers are secure,
   let' look at how to write them.

Security Architectures for SC

Security A



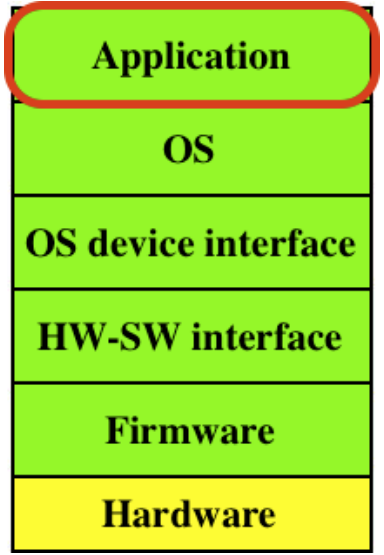
Security Architectures for SC
Security A



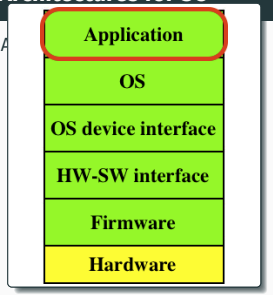
1. a large number of SC results have been made possible due to the developments of the security community
   SGX Enclaves (aka PMA), ASLR, TAL, PUMP. Cheri are all examples of security architectures which support secure compilation and whose development allowed us to devise sc for them

Security Architectures for SC

Security A



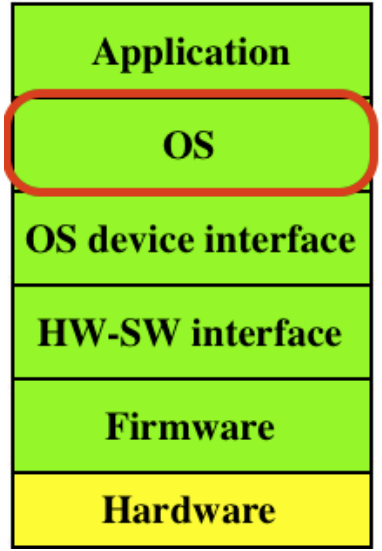
Security Architectures for SC
Security A



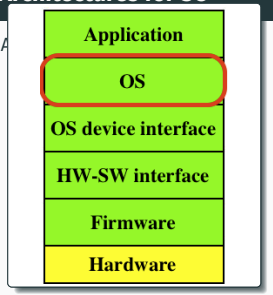
1. a large number of SC results have been made possible due to the developments of the security community
   SGX Enclaves (aka PMA), ASLR, TAL, PUMP. Cheri are all examples of security architectures which support secure compilation and whose development allowed us to devise sc for them

Security Architectures for SC

Security A



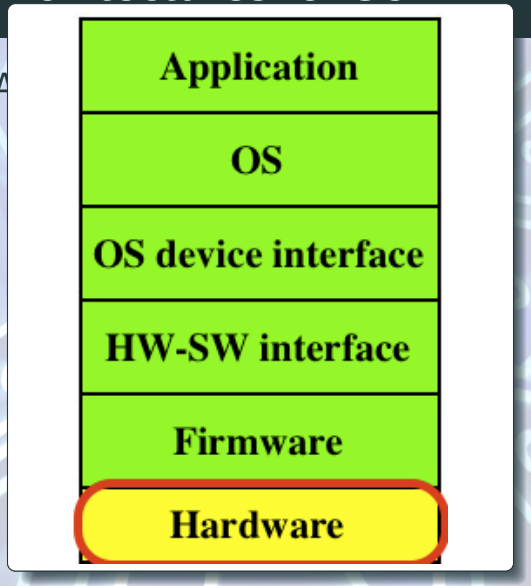
Security Architectures for SC
Security A



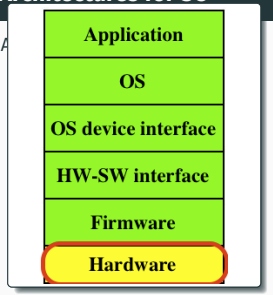
1. sec arch are needed because we simply can't SC to our pcs without having to trust huge codebases so we rely on special machines that give us additional security primitives for the compiler to rely on
   - isolation / randomisation / types / tags / capabilities
   these let us devise secure compilers that have a

Security Architectures for SC

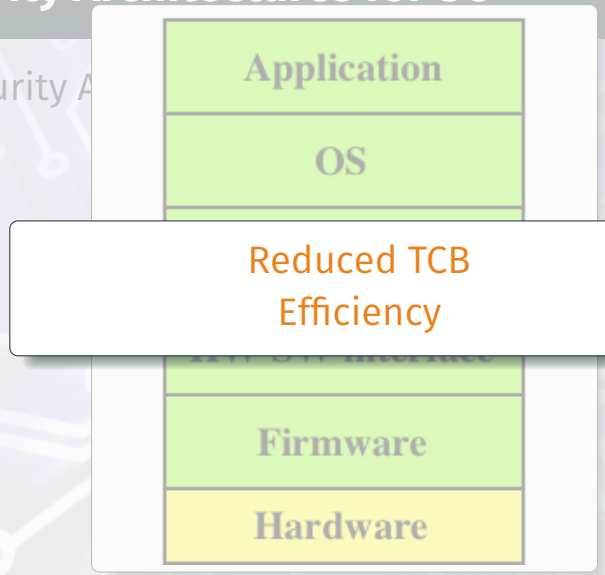
Security A



Security Architectures for SC
Security A



Security Architectures for SC



Reduced TCB
Efficiency

Security Architectures for SC

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
- 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]

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
- Hardware support for fine-grained Capabilities

1. now i want to discuss some of the most recent and interesting of these sec arch: cap machs
cap machs bring the capability principle to hardware, so they are directly usable at the assembly level

Capability Machines: Cheri

Capability Machines

- Ha
  Ca

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

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

capabilities: subjects perform operations on objects if they have the right to do it no right -> no operation

in this case subj = instructions, operations = r/w/x, objs are memory areas/addresses

so to load a word you need cap to R it,

to jump to an address you need cap to X it

Capability Machines: Cheri

Capability Machines

- Hardware support for fine-grained Ca

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

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

capabilities: subjects perform operations on objects if they have the right to do it no right -> no operation

in this case subj = instructions, operations = r/w/x, objs are memory areas/addresses

so to load a word you need cap to R it,

to jump to an address you need cap to X it

Capability Machines: Cheri

Capability Machines

- Hardware support for fine-grained Capability Mantra:

  subjects perform operations on objects if they have rights

  instructions

  read/ write/ execute

Capability Machines: Cheri
Capability Machines
Capability Mantra:
subjects perform operations on objects if they have rights
instructions
read/ write/ execute

capabilities: subjects perform operations on objects if they have the right to do it no right -> no operation

in this case subj = instructions, operations = r/w/x, objs are memory areas/addresses

so to load a word you need cap to R it,

to jump to an address you need cap to X it

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
Capability Mantra:
subjects
perform operat
on objects
if they have rig
instructions
read/ write/ execute
address ranges

capabilities: subjects perform operations on objects if they have the right to do it no right -> no operation

in this case subj = instructions, operations = r/w/x, objs are memory areas/addresses

so to load a word you need cap to R it,

to jump to an address you need cap to X it

Capability Machines: Cheri

Capability Machines

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

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

Cheri is a CM implementation that provides this principle and that is mature enough to support OSs

description of a cheri cap

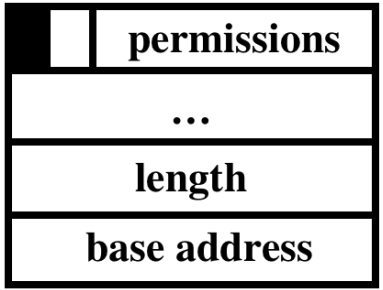
Capability Machines: Cheri

Capability Machines

- Hardw
  Capa
- Cheri

uff et al'14]

A Cheri capability



Capability Machines: Cheri
Capability Machines
- Hardw
  Capa
- Cheri
uff et al'14]
A Cheri capability
permissions
...
length
base address

Cheri is a CM implementation that provides this principle and that is mature enough to support OSs

description of a cheri cap

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

Cheri is a CM implementation that provides this principle and that is mature enough to support OSs

description of a cheri cap

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

```
load rs rd cap
```


Capability Machines: Cheri

Capability Machines

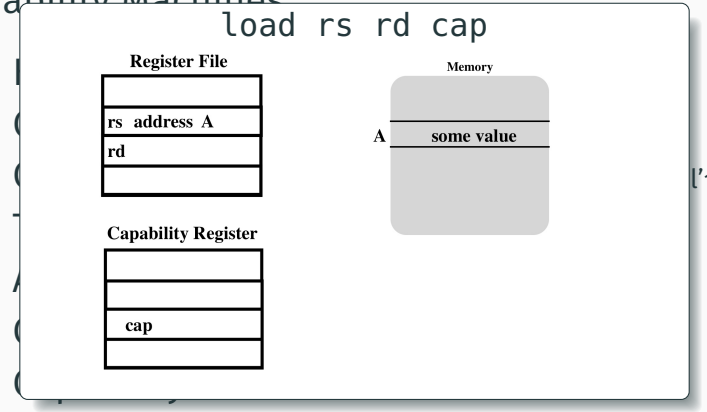
- Hardware support for fine-grained Capabilities
- Cheri (MIPS extension, FPGA) [Woodruff et al'14]
- Tagged memory
- Aligned memory
- Capability registers file
- Capability instructions

load rs rd cap

Capability Machines: Cheri

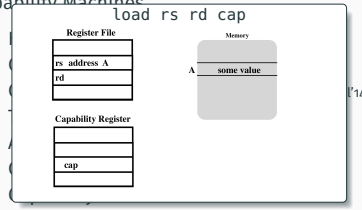
Capability Machines

- I
- C
- C
- T
- A
- C
- C



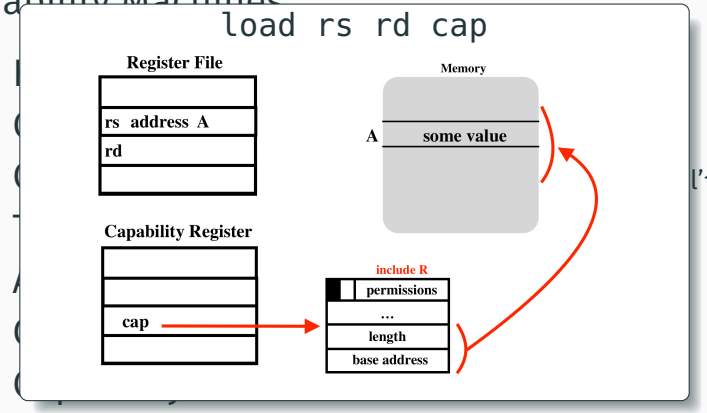
['14]

Capability Machines: Cheri



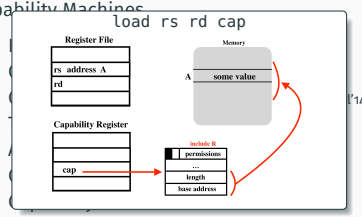
Capability Machines: Cheri

Capability Machines



[’14]

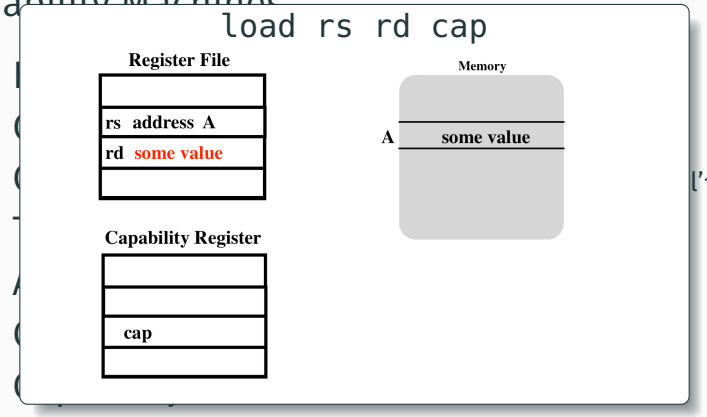
Capability Machines: Cheri



[’14]

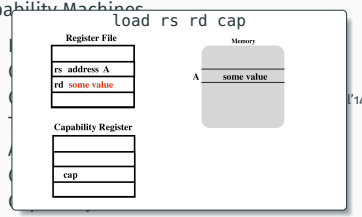
Capability Machines: Cheri

Capability Machines



['14]

Capability Machines: Cheri



['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 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 Capabilities
- Ch[illegible] et al'14]
- Tag[illegible]
- Aligned memory
- Capability registers file
- Capability instructions

Unforgeable capabilities at the hardware level
Mature: has a FreeBSD port

Capability Machines: Cheri

Capability Machines

- Hardware support for fine-grained Capabilities
- Ch[illegible] et al'14]
- Tag[illegible]
- Aligned memory
- 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

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

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

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

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

More efficient than existing results

CM and Secure Compilation
- identify secure compartments
- wra a
  cap
- cap ods:
  pu
- cap s:
  sha
- sup
  (runtime modification of accesses)
More efficient than existing results

CM and Secure Compilation

- identify secure compartments
- wra… a
  cap…
- cap… ods:
  pu…
- cap… s:
  sha…
- sup…
  (runtime modification of accesses)

More efficient than existing results

Support unprecedented security paradigms

CM and Secure Compilation

- identify secure compartments
- wra… a
  cap…
- cap… ods:
  pu…
- cap… s:
  sha…
- sup…
  (runtime modification of accesses)

More efficient than existing results

Support unprecedented security paradigms

CM and Secure Compilation

- identify secure compartments
- wra… a
  cap…
- cap… ods:
  pu…
- cap… s:
  sha…
- sup…
  (runtime modification of accesses)

More efficient than existing results

Support unprecedented security paradigms

Running! implemented by Tsampas

CM and Secure Compilation

- identify secure compartments
- wra… a
  cap…
- cap… ods:
  pu…
- cap… s:
  sha…
- sup…
  (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

Conclusion

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

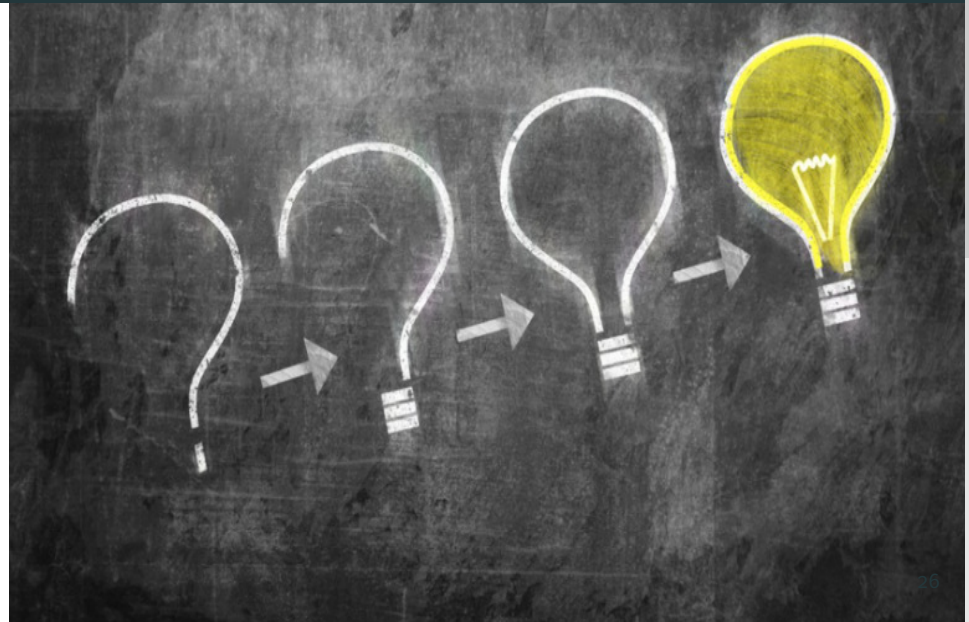
Conclusion

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

Conclusion

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

Conclusion



Conclusion