Secure Compilation: Formal Foundations and (Some) Applications

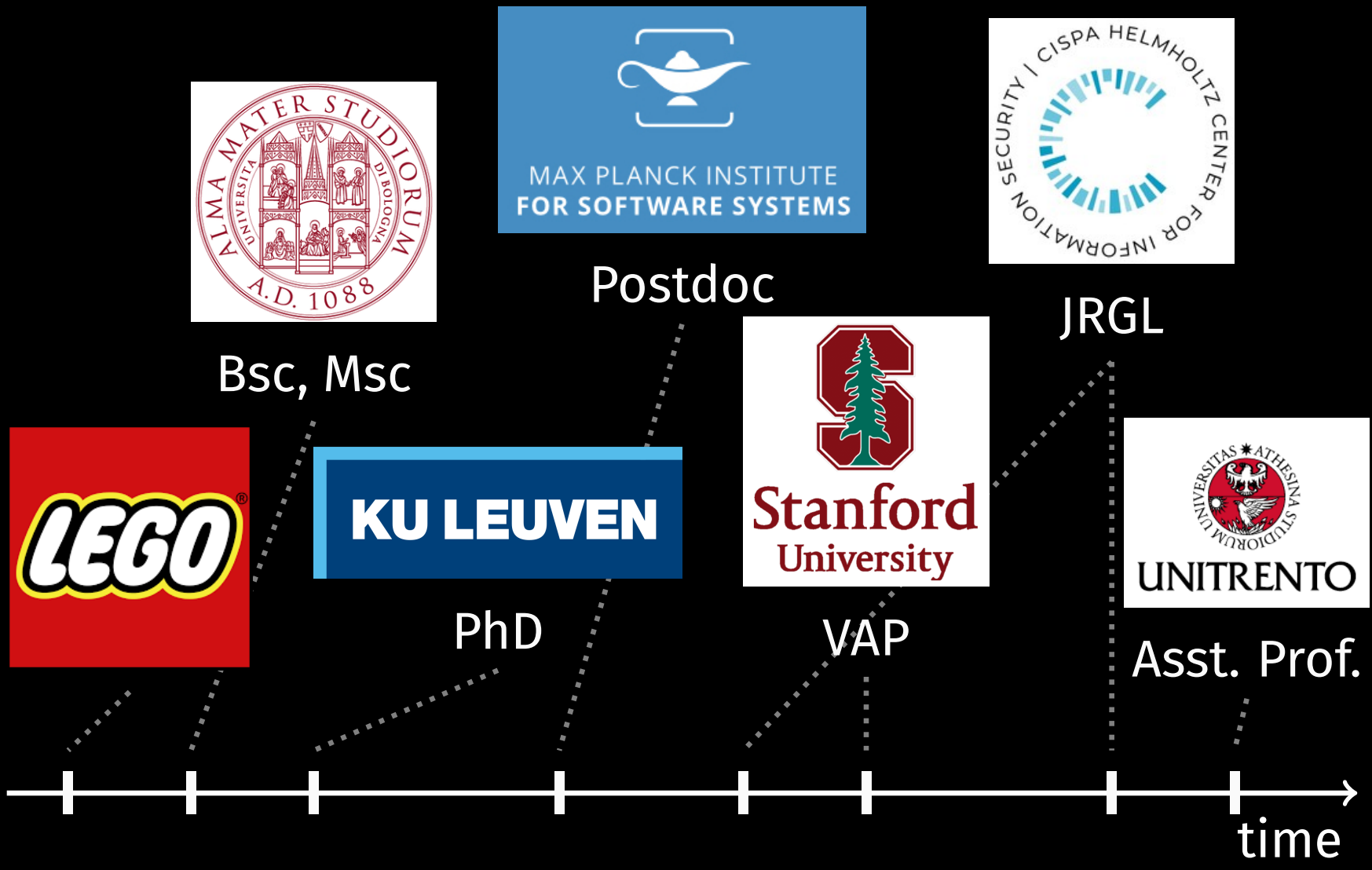
Marco Patrignani[1]

03 April 2024

1 UNIVERSITÀ DI TRENTO

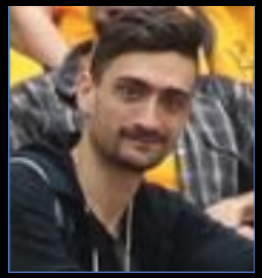
Who Am I ?

Marco Patrignani



Special Thanks to:

(wrt the contents of this talk)















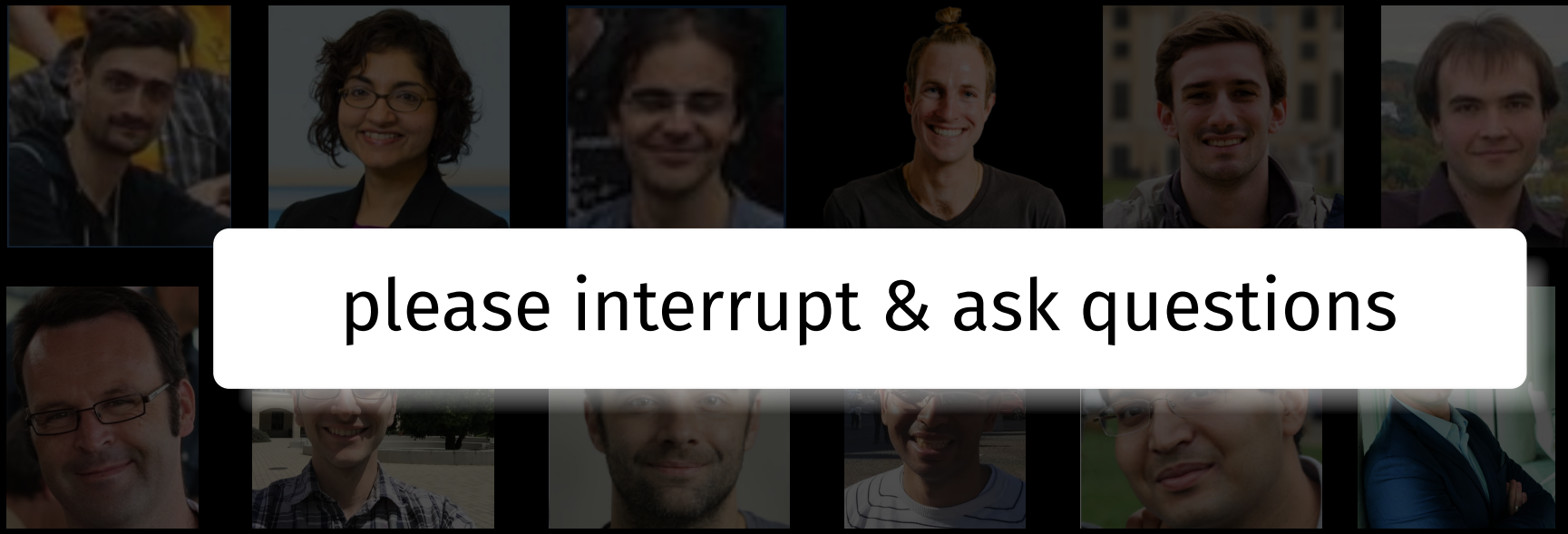






Special Thanks to:

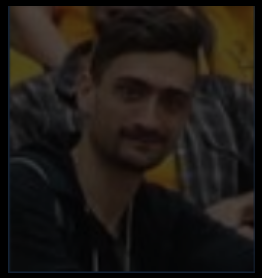
(wrt the contents of this talk)



please interrupt & ask questions

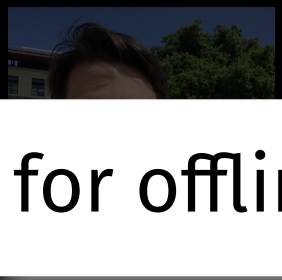
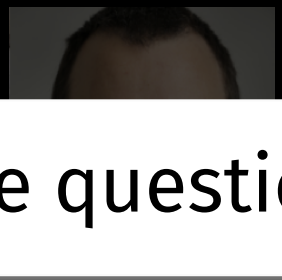
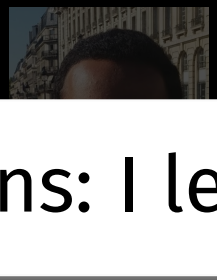
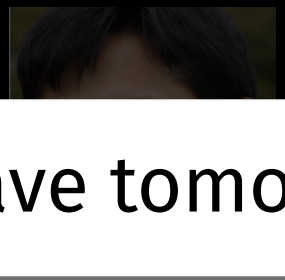
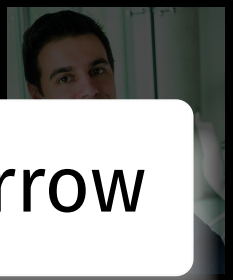
Special Thanks to:

(wrt the contents of this talk)



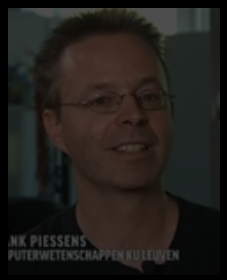




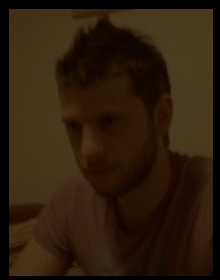
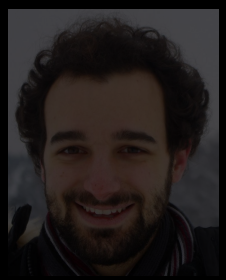







for offline questions: I leave tomorrow






Foundations of Secure Compilation

Programming Languages: Pros and Cons

Good PLs ( , 

,  ,  , ...) provide:

- helpful abstractions to write secure code

Programming Languages: Pros and Cons

Good PLs ( , 

,  ,  , ...) provide:

- helpful abstractions to write secure code

but

- when compiled ($[\![\cdot]\!]$) and linked with adversarial target code

Programming Languages: Pros and Cons

Good PLs ( , 

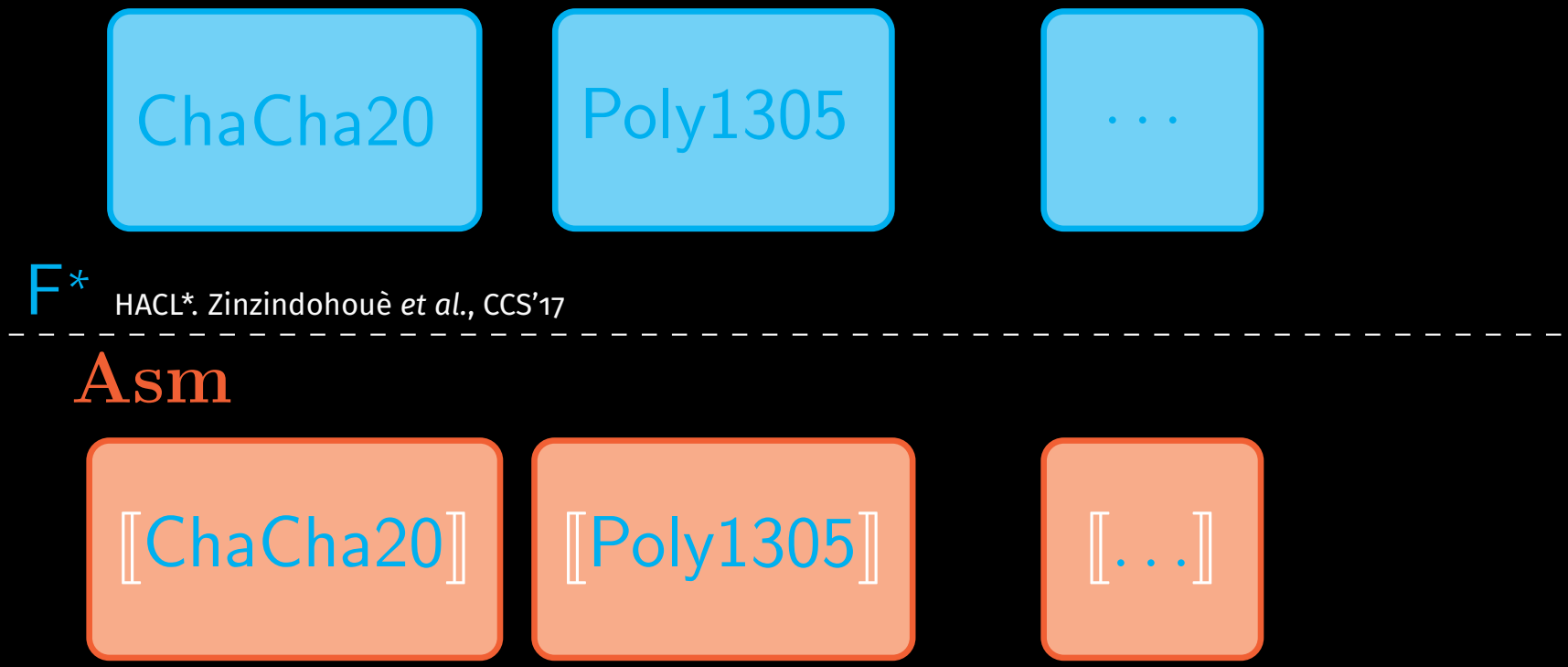
,  ,  , ...) provide:

- helpful abstractions to write secure code

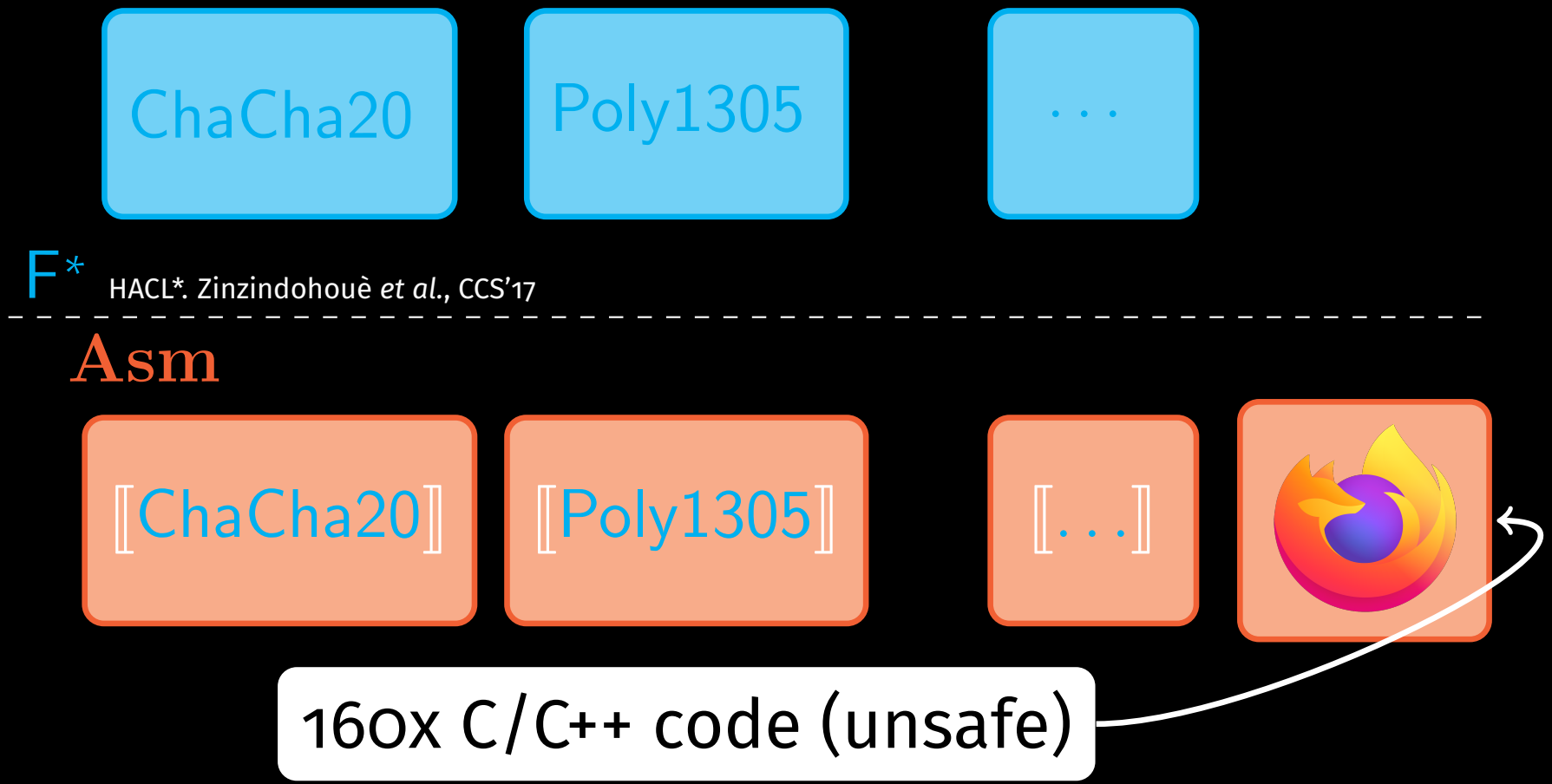
but

- when compiled ($[\![\cdot]\!]$) and linked with adversarial target code
- these abstractions are NOT enforced

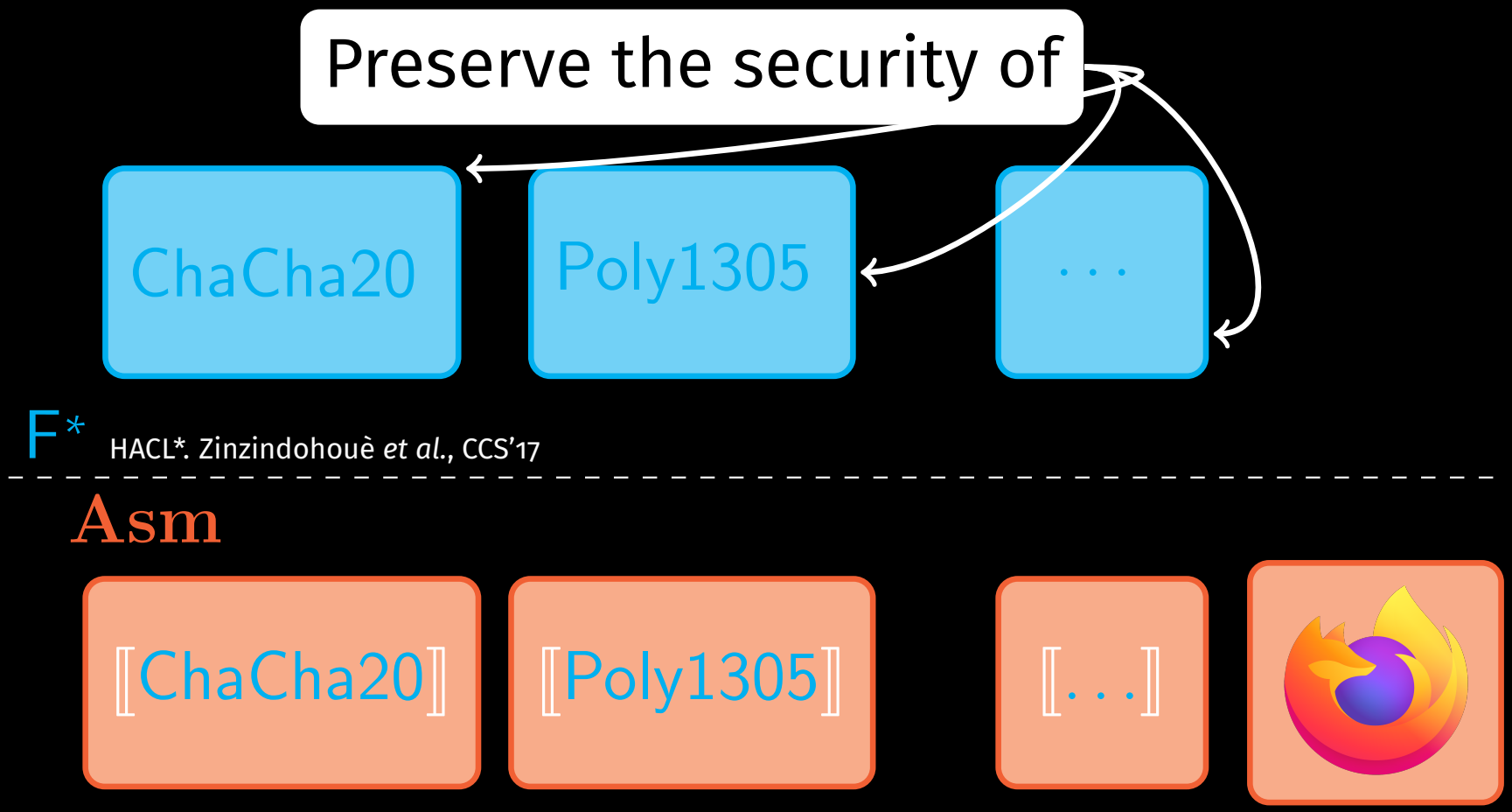
Secure Compilation: Example



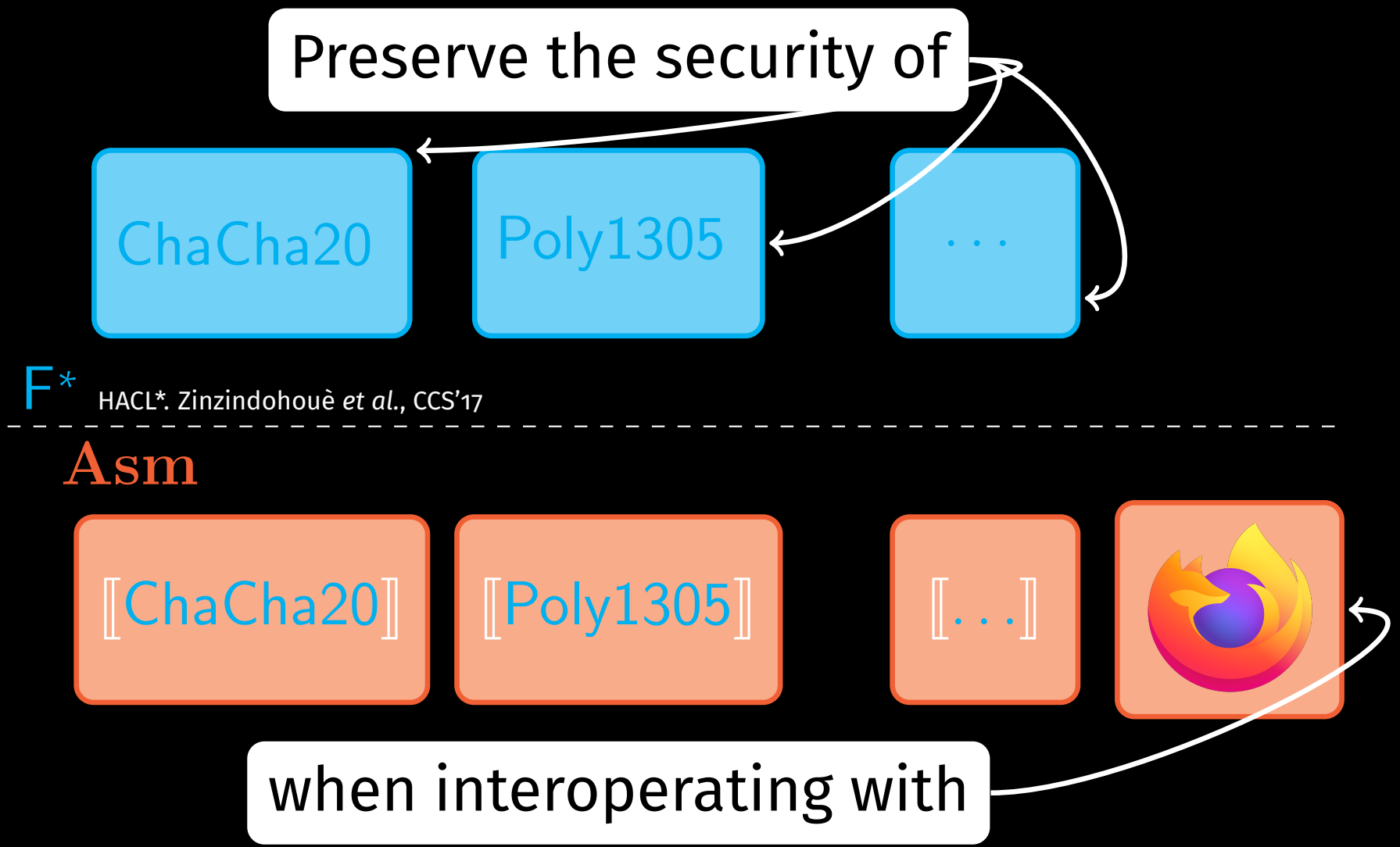
Secure Compilation: Example



Secure Compilation: Example



Secure Compilation: Example



Secure Compilation: Example

Correct compilation

ChaCha20 | Poly1305 | . . .

F* HACL*. Zinzindohouè *et al.*, CCS'17

Asm

⟦ChaCha20⟧ | ⟦Poly1305⟧ | ⟦. . .⟧

Secure Compilation: Example

Secure compilation

ChaCha20 | Poly1305 | ...

F* HACL*. Zinzindohouè *et al.*, CCS'17

Asm

⟦ChaCha20⟧ | ⟦Poly1305⟧ | ⟦...⟧

Secure Compilation: Example

Enable source-level security reasoning

ChaCha20 | Poly1305 | . . .

F* HACL*. Zinzindohouè *et al.*, CCS'17

Asm

⟦ChaCha20⟧ | ⟦Poly1305⟧ | ⟦. . .⟧

Quest for Foundations

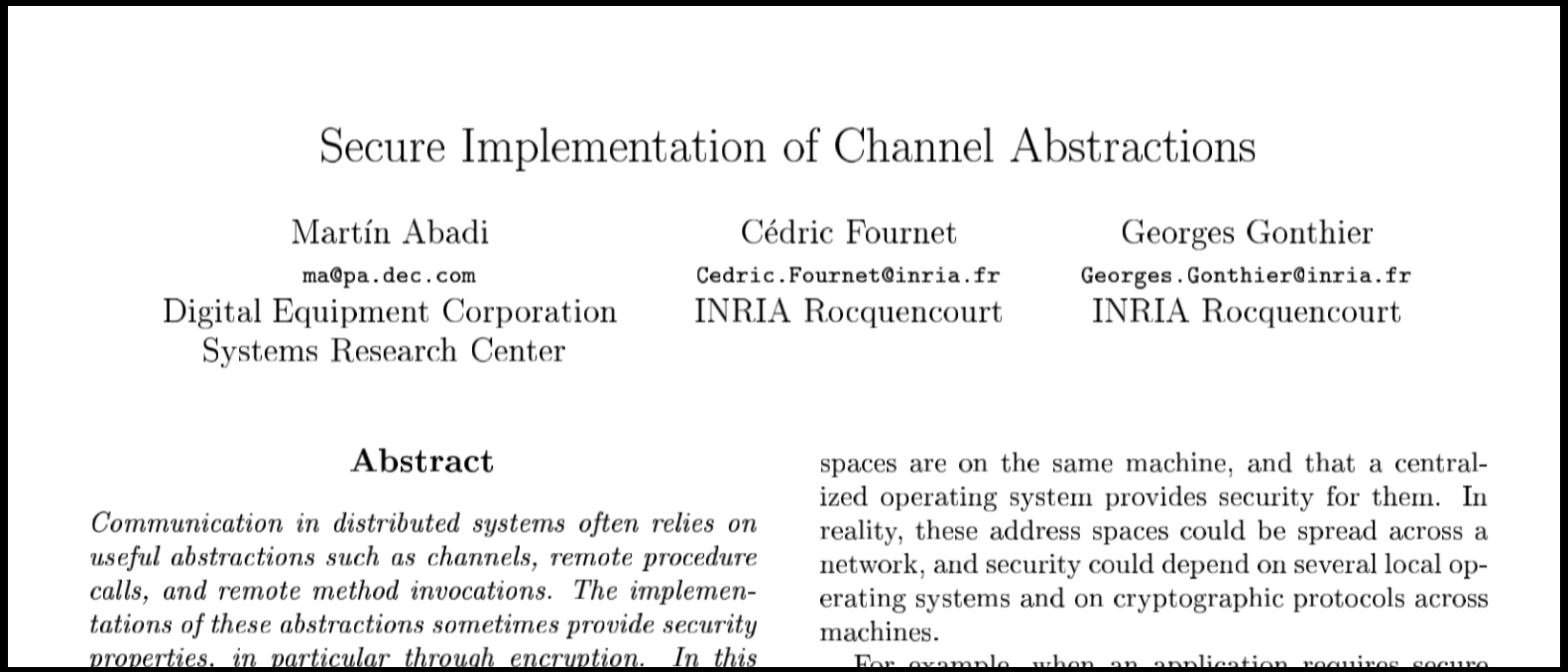
What does it mean for a compiler to be secure?

Quest for Foundations

What does it mean for a compiler to be secure?

Analogous questions are answered for type systems, correct compilation, …

Once Upon a Time in Process Algebra



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

Challenge: define that their implementation of secure channels via cryptography was secure

Once Upon a Time in Process Algebra

Fully Abstract Compilation (FAC)

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

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

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

network, and security could depend on several local operating systems and on cryptographic protocols across machines.

Challenge: define that their implementation of secure channels via cryptography was secure

Fully Abstract Compilation Influence

ACM CSUR'19

Fully Abstract Compilation to JavaScript

Secure Implementations for Typed Session Abstractions

Typed Closure Conversion Preserves Observational Equivalence

Amal Ahmed Matthias Blume
Toyota Technological Institute at Chicago
{amal,blume}@tti-c.org

Fully-Abstract Compilation by Approximate Back-Translation

Authentication primitives and their compilation

Martín Abadi* Cédric Fournet Georges G
Bell Labs Research Microsoft Research INRIA Roc
Lucent Technologies

On Protection by Layout Randomization

Beyond Good and Evil

Formalizing the Security Guarantees of Compartmentalizing Compilation

Secure Compilation of Object-Oriented Components to Protected Module Architectures

Marco Patrignani, Dave Clarke, and Frank Piessens

iMinds-DistriNet, Dept. Computer Sci

A Secure Compiler for ML Module

Local Memory via Layout Randomization

James Riely

Corin Pitcher Julian Rathke
University of Southampton

An Equivalence-Preserving CPS Translation via Multi-Language Semantics *

Amal Ahmed

Matthias Blume
Google
blume@google.com

Secure Compilation to Protected Module Architectures

Marco Patrig
Dept. Comput
and Dave C

Fully Abstract Compilation via Universal Embedding *

On Modular and Fully-Abstract Compil

- FAC: useful for language expressiveness

but complex and with an unclear security implication

- FAC: useful for language expressiveness
  but complex and with an unclear security implication
- Challenge: easier/more efficient/more precise alternatives

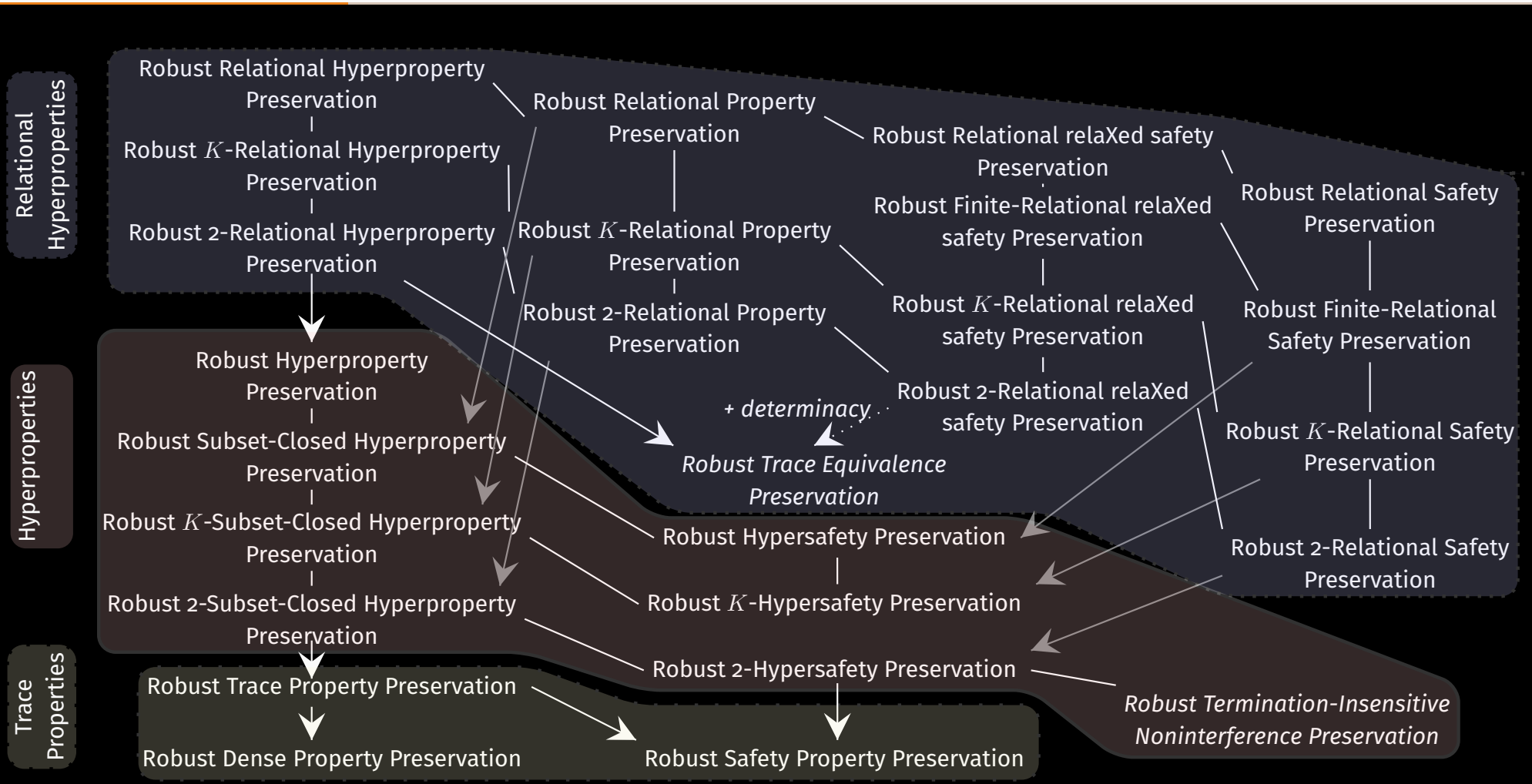
- · FAC: useful for language expressiveness
  but complex and with an unclear security implication
- · Challenge: easier/more efficient/more precise alternatives

  preserve classes of (hyper)properties

Clarkson & Schneider JCS '10

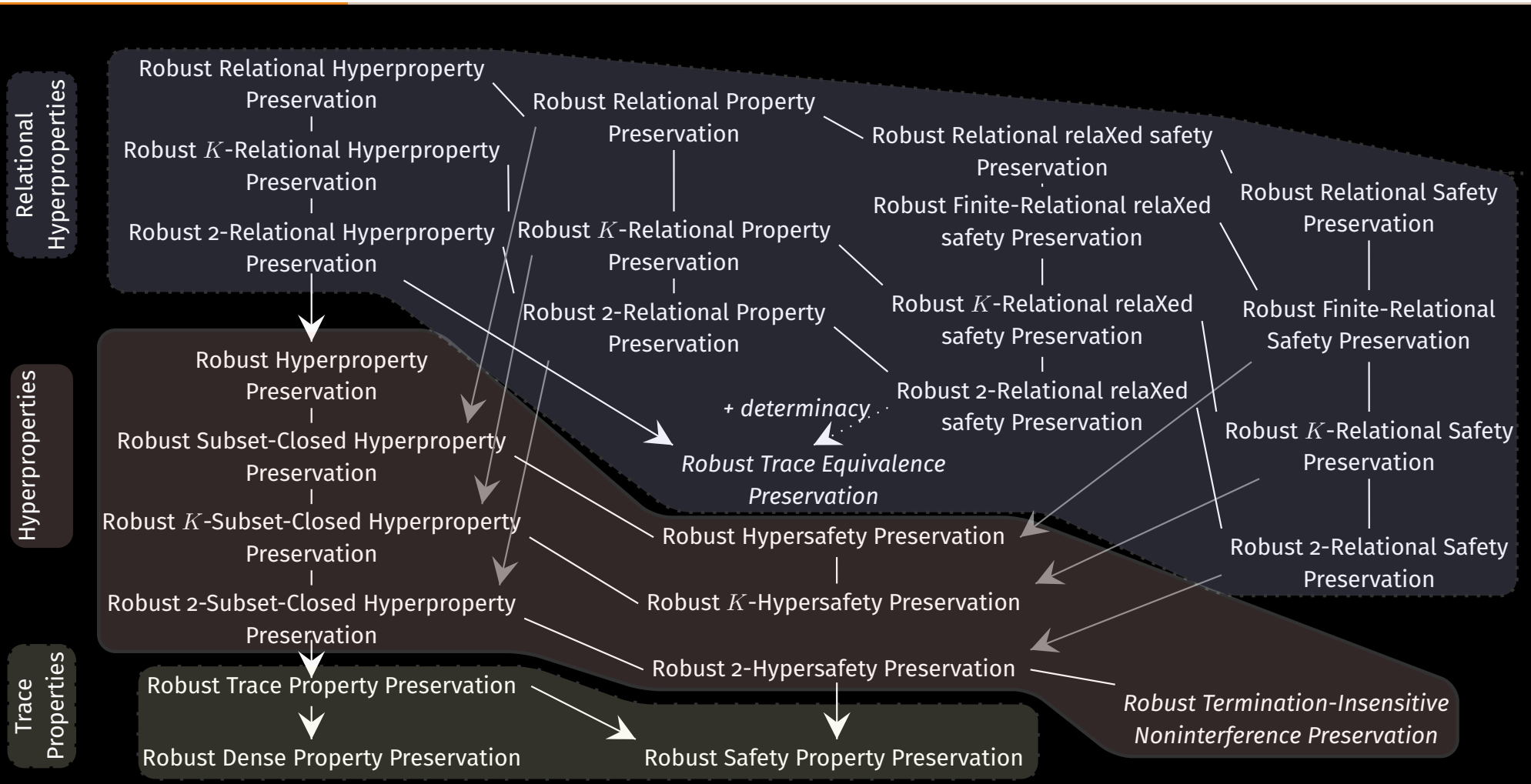
Robust Compilation (RC) Criteria

CSF'19, ESOP'20, Toplas'21



Robust Compilation (RC) Criteria

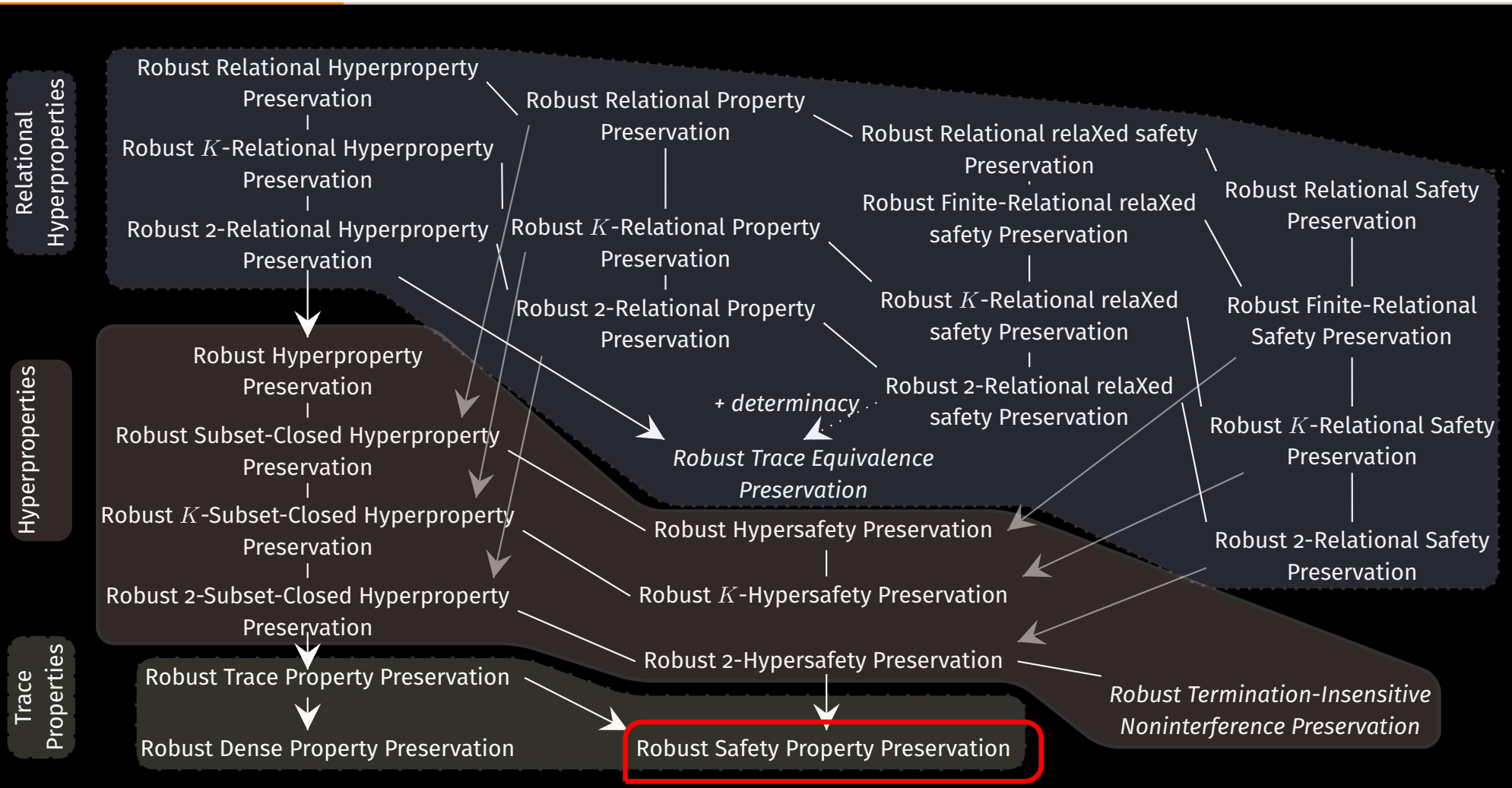
CSF'19, ESOP'20, Toplas'21



Tradeoffs for code efficiency, security guarantees, proof complexity

Robust Compilation (RC) Criteria

CSF'19, ESOP'20, Toplas'21



Tradeoffs for code efficiency, security guarantees, proof complexity

Robust Criteria: Intuition

Each point has two equivalent criteria:

- $\mathrm{Property} - \mathrm{ful}$:
  - \+ clearly tells what class it preserves

Robust Criteria: Intuition

Each point has two equivalent criteria:

- $\mathrm{Property} - \mathrm{ful}$:
  - \+ clearly tells what class it preserves
  - \- harder to prove

Robust Criteria: Intuition

Each point has two equivalent criteria:

- Property – ful :
  - \+ clearly tells what class it preserves
  - \- harder to prove
- Property – free :
  - \+ easier to prove

Robust Criteria: Intuition

Each point has two equivalent criteria:

- $\mathrm{Property} - \mathrm{ful}$:
  - \+ clearly tells what class it preserves
  - \- harder to prove
- $\mathrm{Property} - \mathrm{free}$:
  - \+ easier to prove
  - \- unclear what security classes are preserved

In Depth Example: RSC

$[\![\cdot]\!]$ = compiler $[\![\cdot]\!] : \mathrm{RSP} \stackrel{\mathrm{def}}{=}$

In Depth Example: RSC

$[\![\cdot]\!]$ = compiler
π / π = set of traces

$$[\![\cdot]\!] : \mathrm{RSP} \stackrel{\mathrm{def}}{=} \forall \pi \approx \pi \in Safety.$$

In Depth Example: RSC

$[\![\cdot]\!]$ = compiler
π / π = set of traces
P = partial program

$$[\![\cdot]\!] : \mathrm{RSP} \stackrel{\text{def}}{=} \forall \pi \approx \pi \in Safety.\ \forall \mathrm{P}.$$

In Depth Example: RSC

$[\![\cdot]\!]$ = compiler
π / π = set of traces
P = partial program
A / $\mathbf{A}$ = attacker
t / $\mathbf{t}$ = trace of events

$$[\![\cdot]\!] : \mathrm{RSP} \stackrel{\text{def}}{=} \forall \pi \approx \pi \in Safety.\ \forall \mathrm{P}.$$
$$\text{if } (\forall \mathrm{A}, \mathrm{t}.$$

In Depth Example: RSC

$[\![\cdot]\!]$ = compiler
π / π = set of traces
P = partial program
A / $\mathbf{A}$ = attacker
t / $\mathbf{t}$ = trace of events
$[\cdot]$ = linking
$\rightsquigarrow$ / $\rightsquigarrow$ = trace semantics

$$[\![\cdot]\!] : \mathrm{RSP} \stackrel{\text{def}}{=} \forall \pi \approx \pi \in \mathit{Safety}.\ \forall \mathrm{P}.$$
$$\text{if } (\forall \mathrm{A}, \mathrm{t}.\ \mathrm{A}\,[\mathrm{P}] \rightsquigarrow \mathrm{t}$$

In Depth Example: RSC

$[\![\cdot]\!]$ = compiler
π / π = set of traces
P = partial program
A / $\mathbf{A}$ = attacker
t / $\mathbf{t}$ = trace of events
$[\cdot]$ = linking
$\rightsquigarrow$ / $\rightsquigarrow$ = trace semantics

$$[\![\cdot]\!] : \mathrm{RSP} \stackrel{\text{def}}{=} \forall \pi \approx \pi \in \mathit{Safety}.\ \forall \mathrm{P}.$$
$$\text{if } (\forall \mathrm{A}, \mathrm{t}.\ \mathrm{A}[\mathrm{P}] \rightsquigarrow \mathrm{t} \Rightarrow \mathrm{t} \in \pi)$$

In Depth Example: RSC

$[\![\cdot]\!]$ = compiler
π / π = set of traces
P = partial program
A / $\mathbf{A}$ = attacker
t / $\mathbf{t}$ = trace of events
$[\cdot]$ = linking
$\rightsquigarrow$ / $\rightsquigarrow$ = trace semantics

$$[\![\cdot]\!] : \mathrm{RSP} \stackrel{\text{def}}{=} \forall \pi \approx \pi \in \mathit{Safety}.\ \forall \mathrm{P}.$$
$$\text{if } (\forall \mathrm{A}, \mathrm{t}.\ \mathrm{A}[\mathrm{P}] \rightsquigarrow \mathrm{t} \Rightarrow \mathrm{t} \in \pi)$$
$$\text{then } (\forall \mathbf{A}, \mathbf{t}.$$

In Depth Example: RSC

$[\![\cdot]\!]$ = compiler
π / π = set of traces
P = partial program
A / $\mathbf{A}$ = attacker
t / $\mathbf{t}$ = trace of events
$[\cdot]$ = linking
$\rightsquigarrow$ / $\rightsquigarrow$ = trace semantics

$$[\![\cdot]\!] : \mathrm{RSP} \stackrel{\text{def}}{=} \forall \pi \approx \pi \in \mathit{Safety}.\ \forall \mathrm{P}.$$
$$\text{if } (\forall \mathrm{A}, \mathrm{t}.\ \mathrm{A}[\mathrm{P}] \rightsquigarrow \mathrm{t} \Rightarrow \mathrm{t} \in \pi)$$
$$\text{then } (\forall \mathbf{A}, \mathbf{t}.\ \mathbf{A}[[\![\mathrm{P}]\!]] \rightsquigarrow \mathbf{t} \Rightarrow$$

In Depth Example: RSC

$[\![\cdot]\!]$ = compiler
π / π = set of traces
P = partial program
A / $\mathbf{A}$ = attacker
t / $\mathbf{t}$ = trace of events
$[\cdot]$ = linking
$\rightsquigarrow$ / $\rightsquigarrow$ = trace semantics

$$[\![\cdot]\!] : \mathrm{RSP} \stackrel{\text{def}}{=} \forall \pi \approx \pi \in \mathit{Safety}.\ \forall \mathrm{P}.$$
$$\text{if } (\forall \mathrm{A}, \mathrm{t}.\ \mathrm{A}[\mathrm{P}] \rightsquigarrow \mathrm{t} \Rightarrow \mathrm{t} \in \pi)$$
$$\text{then } (\forall \mathbf{A}, \mathbf{t}.\ \mathbf{A}[[\![\mathrm{P}]\!]] \rightsquigarrow \mathbf{t} \Rightarrow \mathbf{t} \in \pi)$$

In Depth Example: RSC

$[\![\cdot]\!]$ = compiler
π / π = set of traces
P = partial program
A / $\mathbf{A}$ = attacker
t / $\mathbf{t}$ = trace of events
$[\cdot]$ = linking
$\rightsquigarrow$ / $\rightsquigarrow$ = trace semantics

$$[\![\cdot]\!] : \mathrm{RSP} \stackrel{\text{def}}{=} \forall \pi \approx \pi \in \mathit{Safety}.\ \forall \mathrm{P}.$$
$$\text{if } (\forall \mathrm{A}, \mathrm{t}.\ \mathrm{A}[\mathrm{P}] \rightsquigarrow \mathrm{t} \Rightarrow \mathrm{t} \in \pi)$$
$$\text{then } (\forall \mathbf{A}, \mathbf{t}.\ \mathbf{A}[[\![\mathrm{P}]\!]] \rightsquigarrow \mathbf{t} \Rightarrow \mathbf{t} \in \pi)$$

$$[\![\cdot]\!] : \mathrm{RSC} \stackrel{\text{def}}{=}$$

In Depth Example: RSC

$[\![\cdot]\!]$ = compiler
π / π = set of traces
P = partial program
$A / \mathbf{A}$ = attacker
$t / \mathbf{t}$ = trace of events
$[\cdot]$ = linking
$\rightsquigarrow / \rightsquigarrow$ = trace semantics
$m / \mathbf{m}$ = prefix of a trace

$$[\![\cdot]\!] : \mathrm{RSP} \stackrel{\text{def}}{=} \forall \pi \approx \pi \in \mathit{Safety}.\ \forall \mathsf{P}.$$
$$\text{if } (\forall \mathsf{A}, t.\ \mathsf{A}[\mathsf{P}] \rightsquigarrow t \Rightarrow t \in \pi)$$
$$\text{then } (\forall \mathbf{A}, \mathbf{t}.\ \mathbf{A}[[\![\mathsf{P}]\!]] \rightsquigarrow \mathbf{t} \Rightarrow \mathbf{t} \in \pi)$$

$$[\![\cdot]\!] : \mathrm{RSC} \stackrel{\text{def}}{=} \forall \mathsf{P}, \mathbf{A}, \mathbf{m}.$$

In Depth Example: RSC

$[\![\cdot]\!]$ = compiler
π / π = set of traces
P = partial program
$A / \mathbf{A}$ = attacker
$t / \mathbf{t}$ = trace of events
$[\cdot]$ = linking
$\rightsquigarrow / \rightsquigarrow$ = trace semantics
$m / \mathbf{m}$ = prefix of a trace

$$[\![\cdot]\!] : \mathrm{RSP} \stackrel{\text{def}}{=} \forall \pi \approx \pi \in \mathit{Safety}.\ \forall \mathsf{P}.$$
$$\text{if } (\forall \mathsf{A}, \mathsf{t}.\ \mathsf{A}[\mathsf{P}] \rightsquigarrow \mathsf{t} \Rightarrow \mathsf{t} \in \pi)$$
$$\text{then } (\forall \mathbf{A}, \mathbf{t}.\ \mathbf{A}[[\![\mathsf{P}]\!]] \rightsquigarrow \mathbf{t} \Rightarrow \mathbf{t} \in \pi)$$

$$[\![\cdot]\!] : \mathrm{RSC} \stackrel{\text{def}}{=} \forall \mathsf{P}, \mathbf{A}, \mathbf{m}.$$
$$\text{if } \mathbf{A}[[\![\mathsf{P}]\!]] \rightsquigarrow \mathbf{m}$$

In Depth Example: RSC

$[\![\cdot]\!]$ = compiler
π / π = set of traces
P = partial program
$A / \mathbf{A}$ = attacker
$t / \mathbf{t}$ = trace of events
$[\cdot]$ = linking
$\rightsquigarrow / \rightsquigarrow$ = trace semantics
$m / \mathbf{m}$ = prefix of a trace

$$[\![\cdot]\!] : \mathrm{RSP} \stackrel{\text{def}}{=} \forall \pi \approx \pi \in \mathit{Safety}.\ \forall \mathsf{P}.$$
$$\text{if } (\forall \mathsf{A}, t.\ \mathsf{A}[\mathsf{P}] \rightsquigarrow t \Rightarrow t \in \pi)$$
$$\text{then } (\forall \mathbf{A}, \mathbf{t}.\ \mathbf{A}[[\![\mathsf{P}]\!]] \rightsquigarrow \mathbf{t} \Rightarrow \mathbf{t} \in \pi)$$

$$[\![\cdot]\!] : \mathrm{RSC} \stackrel{\text{def}}{=} \forall \mathsf{P}, \mathbf{A}, \mathbf{m}.$$
$$\text{if } \mathbf{A}[[\![\mathsf{P}]\!]] \rightsquigarrow \mathbf{m}$$
$$\text{then } \exists \mathsf{A},$$

In Depth Example: RSC

$[\![\cdot]\!]$ = compiler
π / π = set of traces
P = partial program
$A / \mathbf{A}$ = attacker
$t / \mathbf{t}$ = trace of events
$[\cdot]$ = linking
$\rightsquigarrow / \rightsquigarrow$ = trace semantics
$m / \mathbf{m}$ = prefix of a trace

$$[\![\cdot]\!] : \mathrm{RSP} \stackrel{\text{def}}{=} \forall \pi \approx \pi \in \mathit{Safety}.\ \forall P.$$
$$\text{if } (\forall A, t.\ A[P] \rightsquigarrow t \Rightarrow t \in \pi)$$
$$\text{then } (\forall \mathbf{A}, \mathbf{t}.\ \mathbf{A}[[\![P]\!]] \rightsquigarrow \mathbf{t} \Rightarrow \mathbf{t} \in \pi)$$

$$[\![\cdot]\!] : \mathrm{RSC} \stackrel{\text{def}}{=} \forall P, \mathbf{A}, \mathbf{m}.$$
$$\text{if } \mathbf{A}[[\![P]\!]] \rightsquigarrow \mathbf{m}$$
$$\text{then } \exists A, m \approx \mathbf{m}.$$

In Depth Example: RSC

$[\![\cdot]\!]$ = compiler
π / π = set of traces
P = partial program
$A / \mathbf{A}$ = attacker
$t / \mathbf{t}$ = trace of events
$[\cdot]$ = linking
$\rightsquigarrow / \rightsquigarrow$ = trace semantics
$m / \mathbf{m}$ = prefix of a trace

$$[\![\cdot]\!] : \mathrm{RSP} \stackrel{\text{def}}{=} \forall \pi \approx \pi \in \mathit{Safety}.\ \forall P.$$
$$\text{if } (\forall A, t.\ A[P] \rightsquigarrow t \Rightarrow t \in \pi)$$
$$\text{then } (\forall \mathbf{A}, \mathbf{t}.\ \mathbf{A}[[\![P]\!]] \rightsquigarrow \mathbf{t} \Rightarrow \mathbf{t} \in \pi)$$

$$[\![\cdot]\!] : \mathrm{RSC} \stackrel{\text{def}}{=} \forall P, \mathbf{A}, \mathbf{m}.$$
$$\text{if } \mathbf{A}[[\![P]\!]] \rightsquigarrow \mathbf{m}$$
$$\text{then } \exists A, m \approx \mathbf{m}.\ A[P] \rightsquigarrow m$$

Secure Compilation Threat Model

- robust, active attacker ($\forall \mathbf{A}$)

robust safety works, e.g., Swasey *et al.* OOPSLA'17, Sammler *et al.* POPL'20

Secure Compilation Threat Model

- robust, active attacker ($\forall \mathbf{A}$)

robust safety works, e.g., Swasey *et al.* OOPSLA'17, Sammler *et al.* POPL'20

- in-language expressible attacker

Secure Compilation Threat Model

- robust, active attacker ($\forall \mathbf{A}$)

robust safety works, e.g., Swasey *et al.* OOPSLA'17, Sammler *et al.* POPL'20

- in-language expressible attacker
- trace-based security behaviour ($\mathrm{m}/\mathbf{m}$)

Secure Compilation Threat Model

- robust, active attacker ($\forall \mathbf{A}$)
- i
- trace-based security behaviour ($\mathrm{m}/\mathbf{m}$)

What can we do with these foundations?

Talk Outline

Robust Memory Safety — POPL'23

Robust Cryptographic Constant Time — (wip)

Micro-architectural Attacks (Spectre) — CCS'21

Security Architectures

(e.g., Cheri/ARM Morello, Sancus/Intel SGX, ...) — Toplas'15, CSF'21, ...

Mechanise Cryptographic Proofs — CSF'24 + wip

Conclusion

Robust Memory Safety

POPL'23

Memory Safety (Untyped, Intra-Object)

Memory Safety (Untyped, Intra-Object)

- add colours+shades to pointers & memory

Memory Safety (Untyped, Intra-Object)

- add colours+shades to pointers & memory
- check colour+shade when using pointers

Memarian *et al.* POPL'19, Azevedo de Amorim *et al.* POST'18

Memory Safety (Untyped, Intra-Object)

- add colours+shades to pointers & memory
- check colour+shade when using pointers

Memarian *et al.* POPL'19, Azevedo de Amorim *et al.* POST'18

F	F	F	F	F	F	F

Memory Safety (Untyped, Intra-Object)

- add colours+shades to pointers & memory
- check colour+shade when using pointers

Memarian *et al.* POPL'19, Azevedo de Amorim *et al.* POST'18

```
alloc(4)
```

Memory Safety (Untyped, Intra-Object)

- add colours+shades to pointers & memory
- check colour+shade when using pointers

Memarian *et al.* POPL'19, Azevedo de Amorim *et al.* POST'18

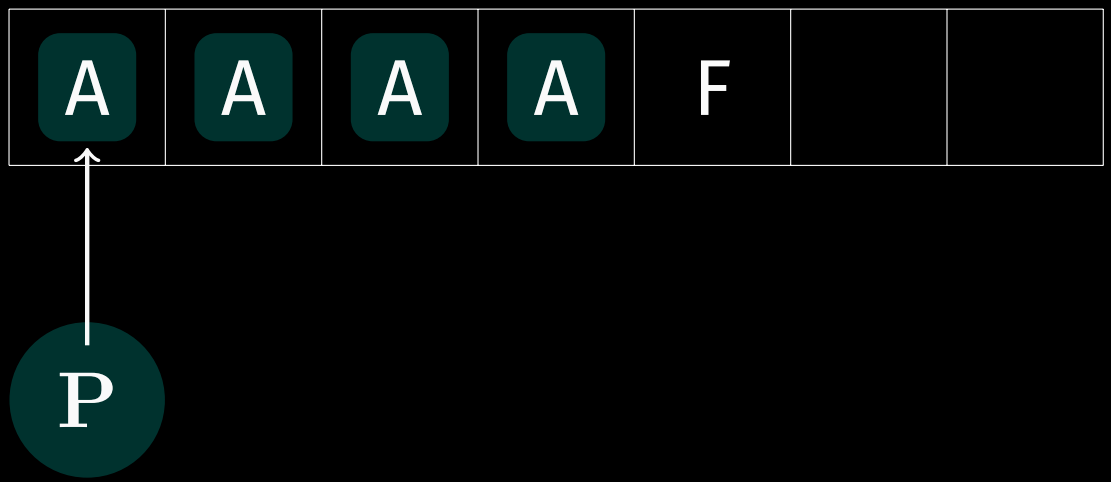
```
alloc(4)
```

Memory Safety (Untyped, Intra-Object)

- add colours+shades to pointers & memory
- check colour+shade when using pointers

Memarian *et al.* POPL'19, Azevedo de Amorim *et al.* POST'18

```
alloc(4)
```

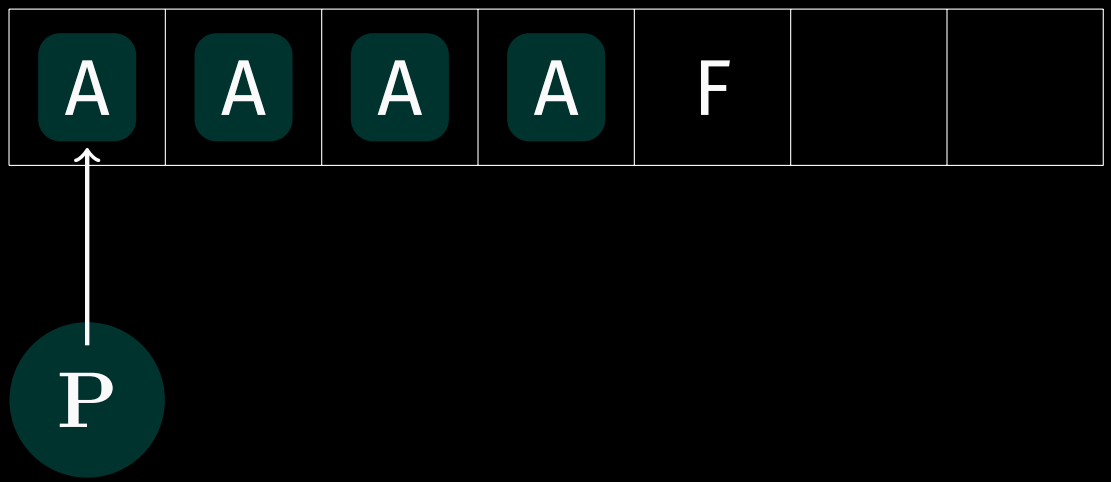


Memory Safety (Untyped, Intra-Object)

- add colours+shades to pointers & memory
- check colour+shade when using pointers

Memarian *et al.* POPL'19, Azevedo de Amorim *et al.* POST'18

```
alloc(4)
alloc(1+1)
```

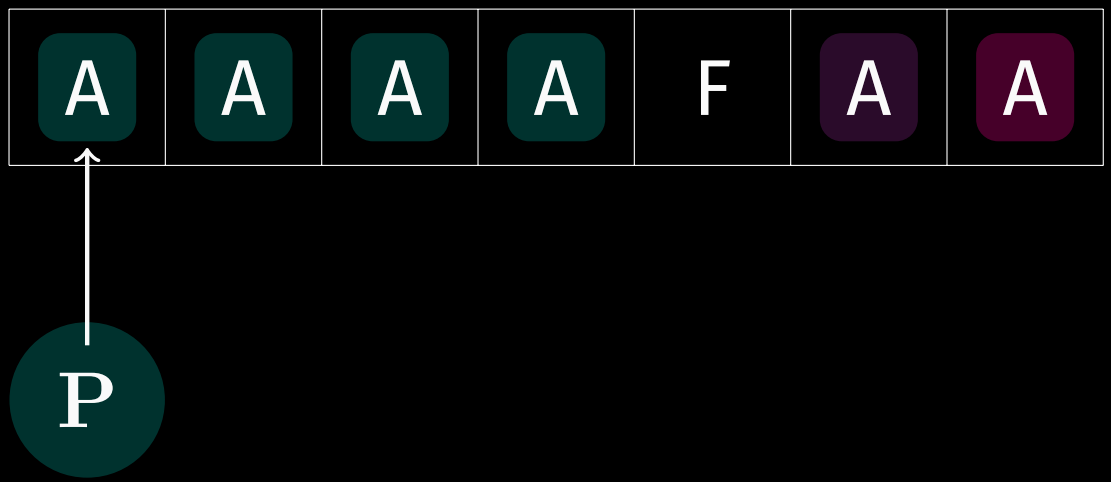


Memory Safety (Untyped, Intra-Object)

- add colours+shades to pointers & memory
- check colour+shade when using pointers

Memarian *et al.* POPL'19, Azevedo de Amorim *et al.* POST'18

```
alloc(4)
alloc(1+1)
```



Memory Safety (Untyped, Intra-Object)

- add colours+shades to pointers & memory
- check colour+shade when using pointers

Memarian *et al.* POPL'19, Azevedo de Amorim *et al.* POST'18

```
alloc(4)
alloc(1+1)
```

Memory Safety (Untyped, Intra-Object)

- add colours+shades to pointers & memory
- check colour+shade when using pointers

Memarian *et al.* POPL'19, Azevedo de Amorim *et al.* POST'18

```
alloc(4)
alloc(1+1)
 read(P)
```

Memory Safety (Untyped, Intra-Object)

- add colours+shades to pointers & memory
- check colour+shade when using pointers

Memarian *et al.* POPL'19, Azevedo de Amorim *et al.* POST'18

ok

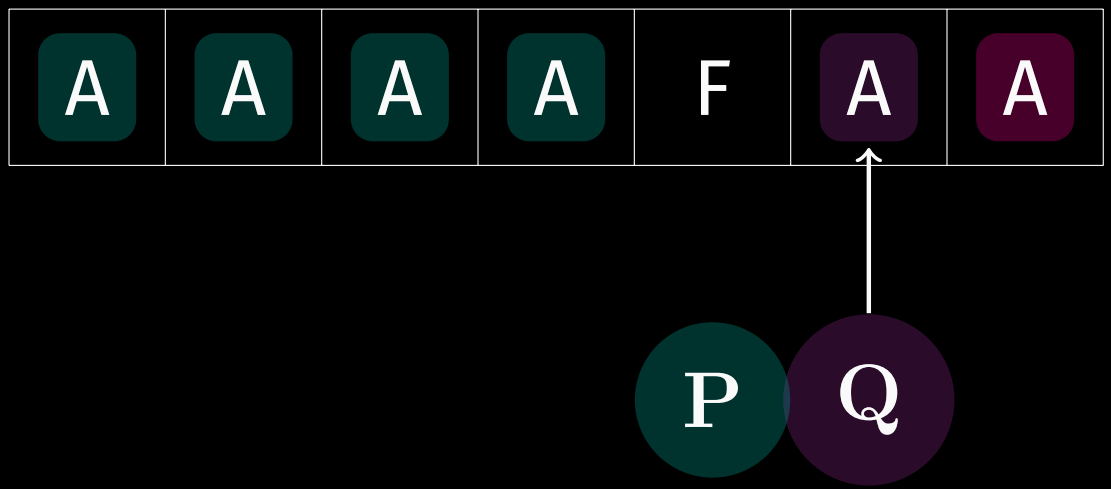
```
alloc(4)
alloc(1+1)
 read(P)
```

Memory Safety (Untyped, Intra-Object)

- add colours+shades to pointers & memory
- check colour+shade when using pointers

Memarian *et al.* POPL'19, Azevedo de Amorim *et al.* POST'18

```
alloc(4)
alloc(1+1)
 read(P)
```



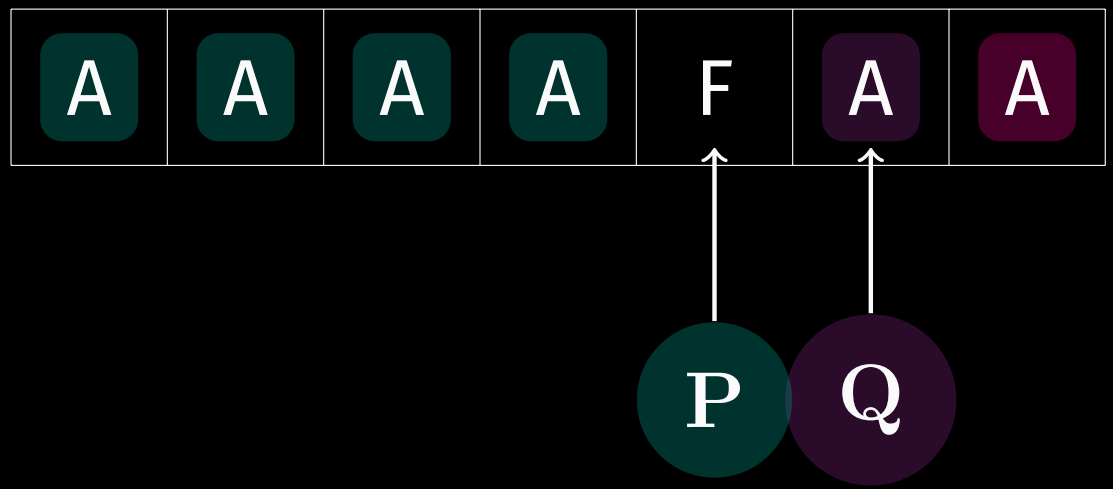
Memory Safety (Untyped, Intra-Object)

- add colours+shades to pointers & memory
- check colour+shade when using pointers

Memarian *et al.* POPL'19, Azevedo de Amorim *et al.* POST'18

NO

```
alloc(4)
alloc(1+1)
 read(P)
 read(P)
```



Memory Safety (Untyped, Intra-Object)

- add colours+shades to pointers & memory
- check colour+shade when using pointers

Memarian *et al.* POPL'19, Azevedo de Amorim *et al.* POST'18

```
alloc(4)
alloc(1+1)
 read(P)
write(P)
```

Memory Safety (Untyped, Intra-Object)

- add colours+shades to pointers & memory
- check colour+shade when using pointers

Memarian *et al.* POPL'19, Azevedo de Amorim *et al.* POST'18

NO

```
alloc(4)
alloc(1+1)
read(P)
write(P)
```

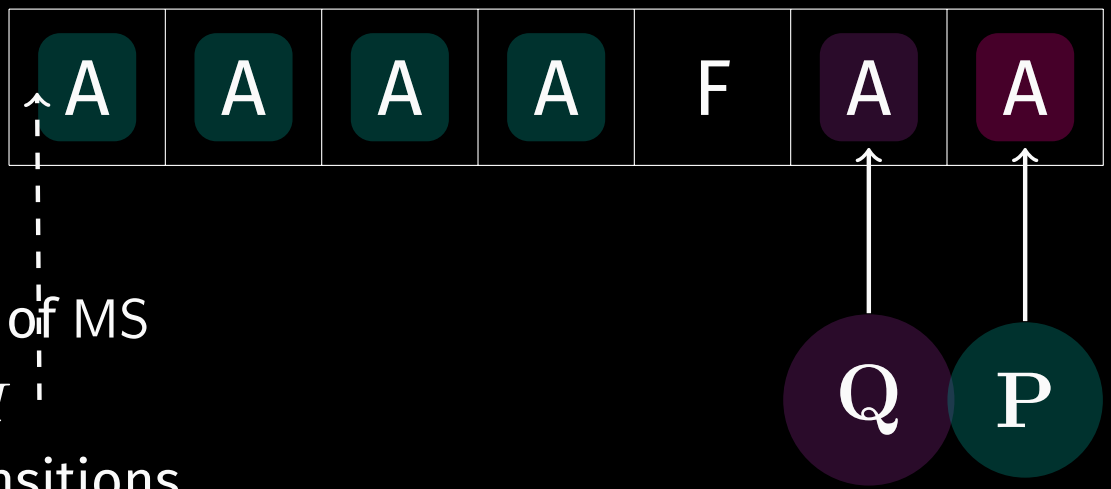
Memory Safety (Untyped, Intra-Object)

- add colours+shades to pointers & memory
- check colour+shade when using pointers

Memarian *et al.* POPL'19, Azevedo de Amorim *et al.* POST'18

NO

```
alloc(4)
alloc(1+1)
read(P)
write(P)
```



Monitor encoding of MS
with state M
and actions for transitions

Memory-Safe WebAssembly (`MSWAsm`)

- WAsm:
  - inter-sandboxes MS

Memory-Safe WebAssembly (MSWAsm)

- WAsm:
  - inter-sandboxes MS
  - intra-sandbox vulnerability

Memory-Safe WebAssembly (MSWAsm)

- WAsm:
  - inter-sandboxes MS
  - intra-sandbox vulnerability
- MSWAsm: segment memory indexed by Cheri-like pointers

Watson *et al.* S&P'15

Memory-Safe WebAssembly (MSWAsm)

- WAsm:
  - inter-sandboxes MS
  - intra-sandbox vulnerability
- MSWAsm: segment memory indexed by Cheri-like pointers

  Watson *et al.* S&P'15
  - handles:

    $\langle \mathrm{base}, \mathrm{length}, \mathrm{offset}, \mathrm{isCorrupted}, \mathrm{id} \rangle$

Memory-Safe WebAssembly (MSWAsm)

- WAsm:
  - inter-sandboxes MS
  - intra-sandbox vulnerability
- MSWAsm: segment memory indexed by Cheri-like pointers Watson *et al.* S&P'15
  - handles:
    $\langle \mathrm{base}, \mathrm{length}, \mathrm{offset}, \mathrm{isCorrupted}, \mathrm{id} \rangle$
- segment instructions:
  - segment_alloc, segment_free

Memory-Safe WebAssembly (MSWAsm)

- WAsm:
  - inter-sandboxes MS
  - intra-sandbox vulnerability
- MSWAsm: segment memory indexed by Cheri-like pointers Watson *et al.* S&P'15
  - handles:
    $\langle \mathrm{base}, \mathrm{length}, \mathrm{offset}, \mathrm{isCorrupted}, \mathrm{id} \rangle$
- segment instructions:
  - segment_alloc, segment_free
  - segment_read, segment_write

Memory-Safe WebAssembly (MSWAsm)

- WAsm:
  - inter-sandboxes MS
  - intra-sandbox vulnerability
- MSWAsm: segment memory indexed by Cheri-like pointers Watson *et al.* S&P'15
  - handles:
    $\langle \mathrm{base}, \mathrm{length}, \mathrm{offset}, \mathrm{isCorrupted}, \mathrm{id} \rangle$
- segment instructions:
  - segment_alloc, segment_free
  - segment_read, segment_write
  - handle_add, handle_slice

Compiling from C to MSWAsm

- pointer becomes handle

Compiling from `C` to `MSWAsm`

- pointer becomes handle
- dereference becomes segment_read

Compiling from `C` to `MSWAsm`

- pointer becomes handle
- dereference becomes segment_read
- write becomes segment_write

Compiling from C to MSWAsm

- pointer becomes handle
- dereference becomes segment_read
- write becomes segment_write
- pointer arithmetic becomes handle_add

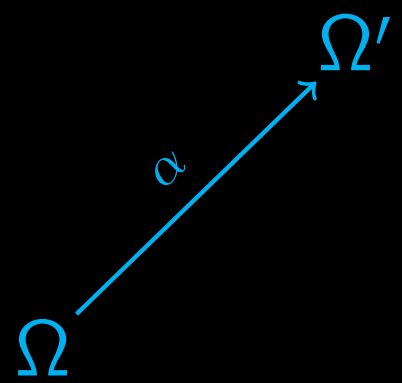
Compiling from C to MSWAsm

- pointer becomes handle
- dereference becomes segment_read
- write becomes segment_write
- pointer arithmetic becomes handle_add
- field access becomes handle_slice

Compiler Properties

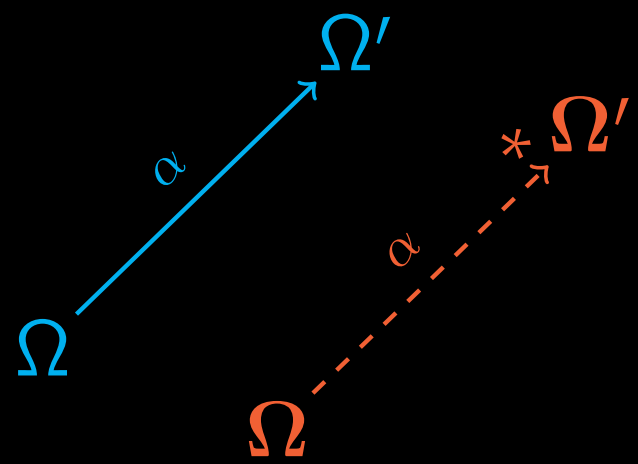
Ω = source state

α / α = trace action



Compiler Properties

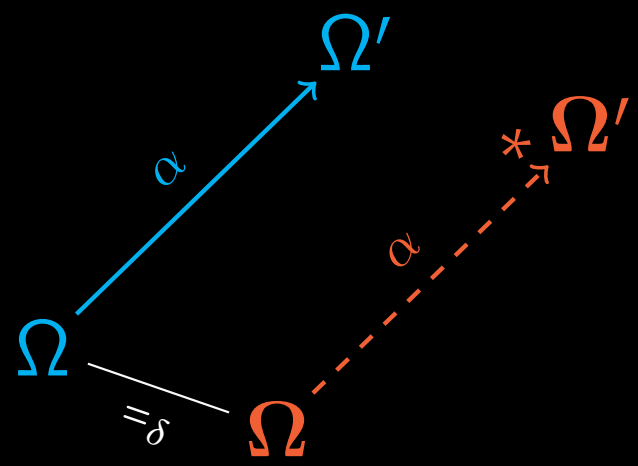
Ω = source state
Ω = compiled state
α / α = trace action



Compiler Properties

Ω = source state
Ω = compiled state
α / α = trace action

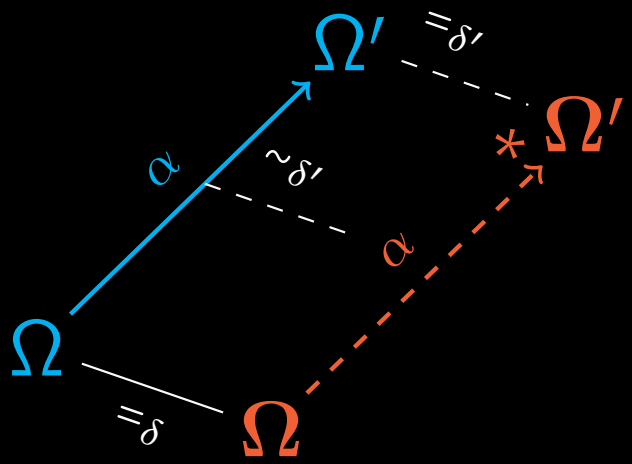
δ = partial bijection



Compiler Properties

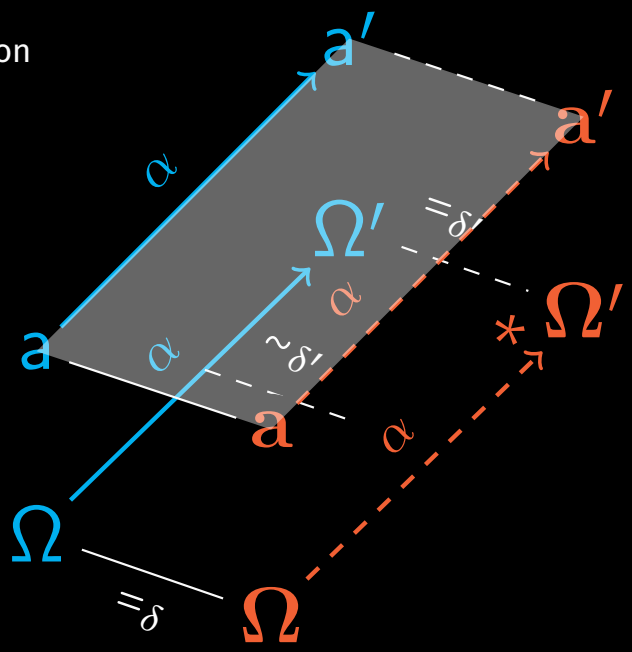
Ω = source state
Ω = compiled state
α / α = trace action

δ = partial bijection



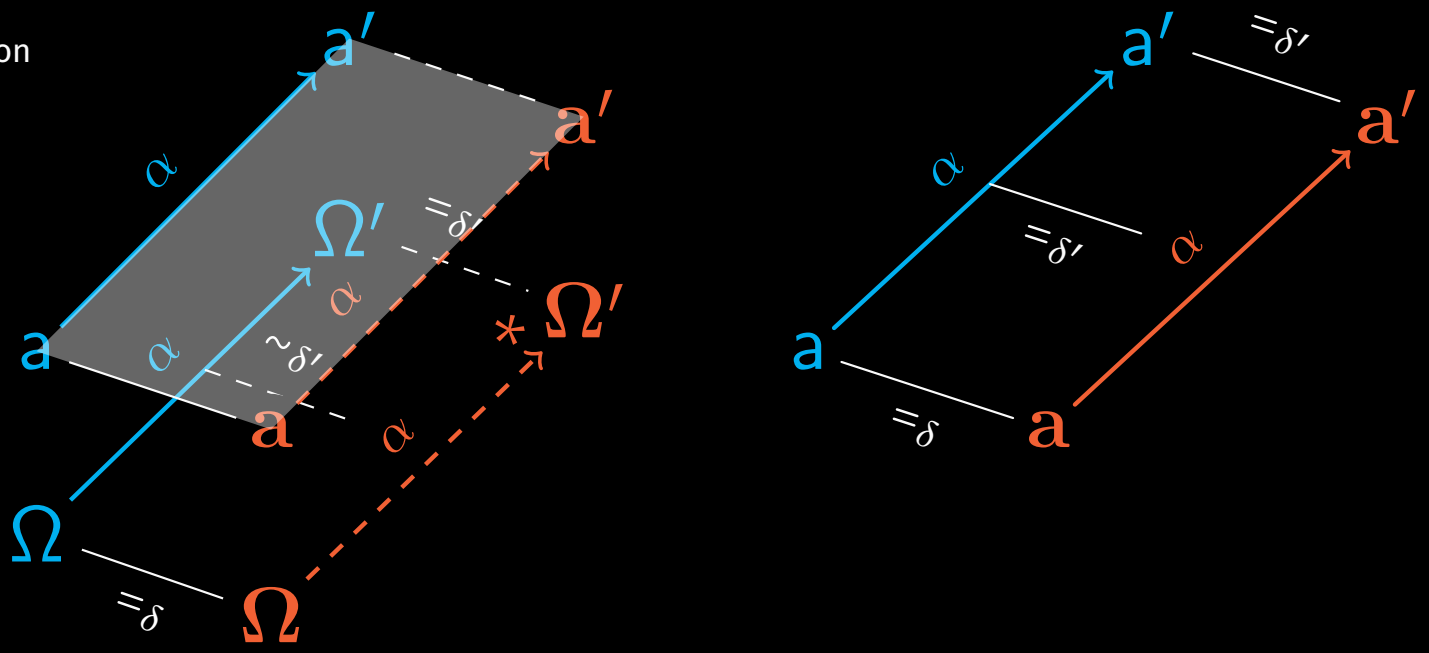
Compiler Properties

Ω = source state
$\mathbf{\Omega}$ = compiled state
α/α = trace action
a/$\mathbf{a}$ = allocator
δ = partial bijection



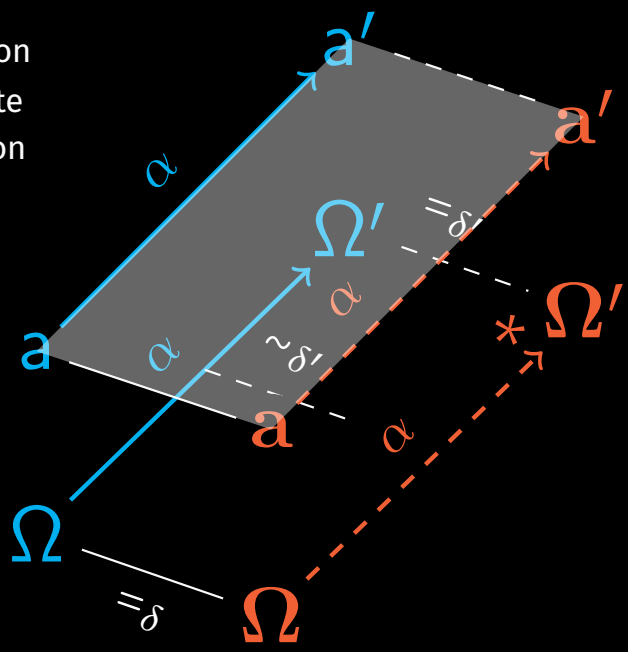
Compiler Properties

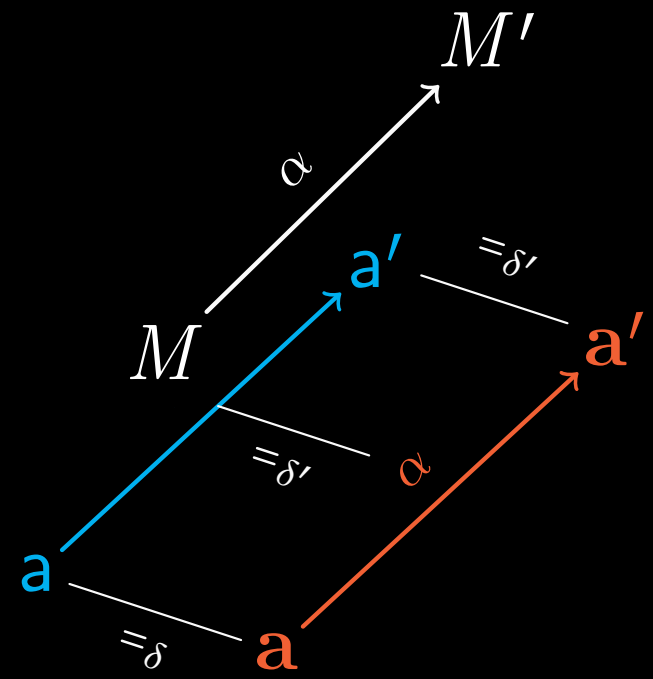
Ω = source state
$\mathbf{\Omega}$ = compiled state
α/α = trace action
a/$\mathbf{a}$ = allocator
δ = partial bijection



Compiler Properties

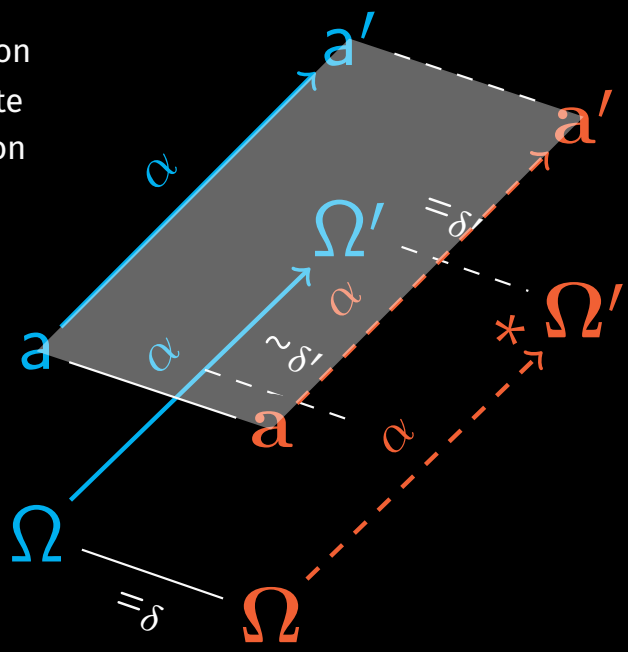
Ω = source state
$\mathbf{\Omega}$ = compiled state
α/α = trace action
a/$\mathbf{a}$ = allocator
δ = partial bijection
M = monitor state
α = monitor action

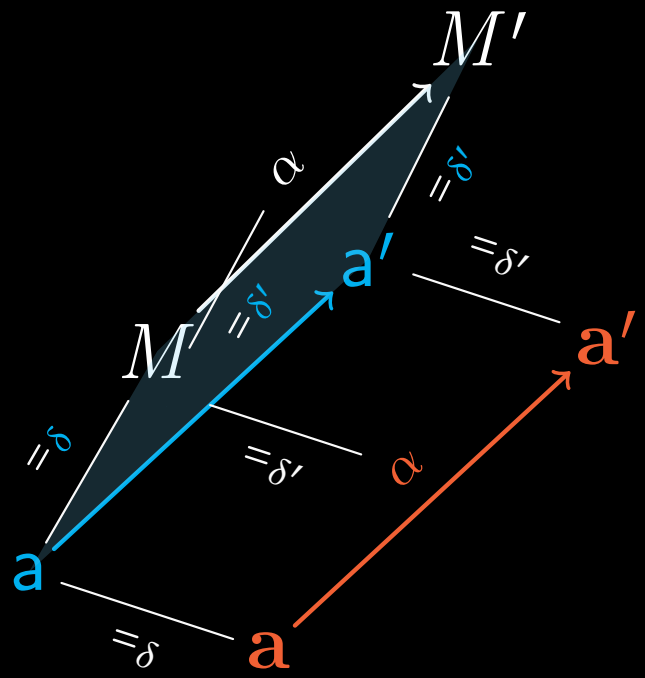




Compiler Properties

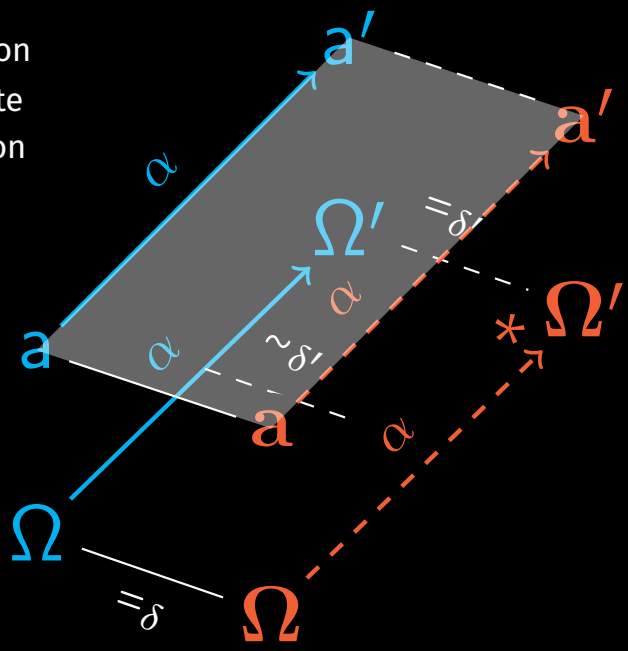
Ω = source state
$\mathbf{\Omega}$ = compiled state
α/α = trace action
a/$\mathbf{a}$ = allocator
δ = partial bijection
M = monitor state
α = monitor action

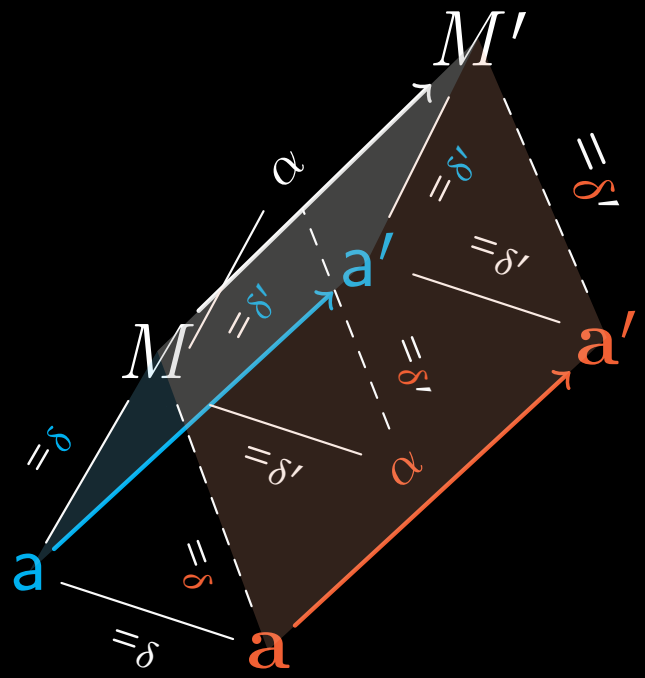




Compiler Properties

Ω = source state
$\mathbf{\Omega}$ = compiled state
α/α = trace action
a/$\mathbf{a}$ = allocator
δ = partial bijection
M = monitor state
α = monitor action

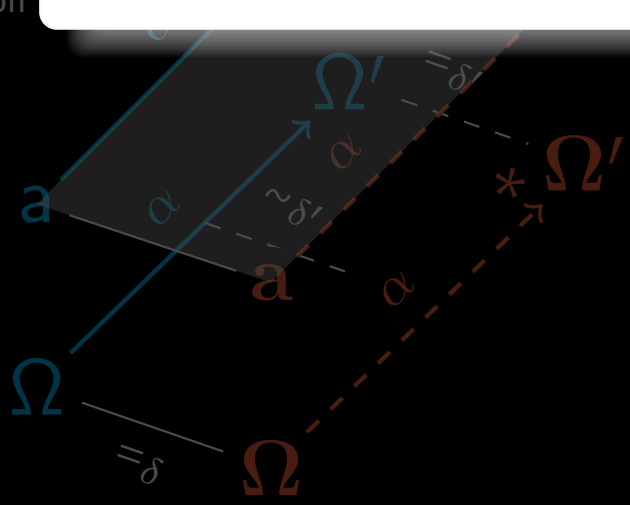


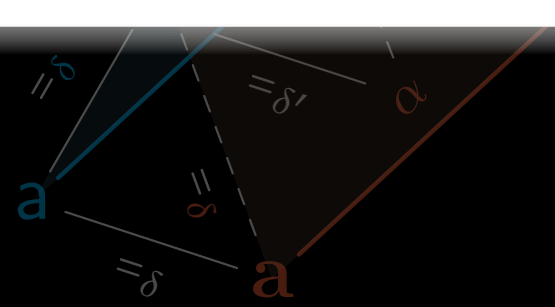


Compiler Properties

PRO: proved MS preservation, MS enforcement

Ω = source state
Ω = compiled state
α / α = trace action
a / a = allocator
δ = partial bijection
M = monitor state
α = monitor action



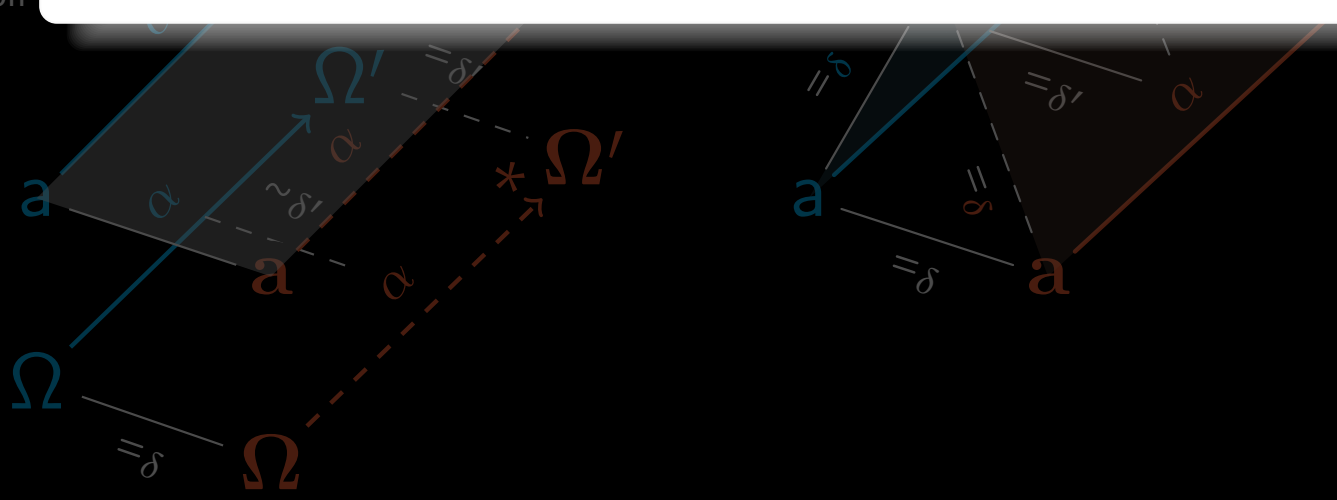


Compiler Properties

PRO: proved MS preservation, MS enforcement

CON: not really RSC (no $\forall \mathbf{A}$)

Ω = source state
Ω = compiled state
α / α = trace action
$\mathrm{a} / \mathbf{a}$ = allocator
δ = partial bijection
M = monitor state
α = monitor action



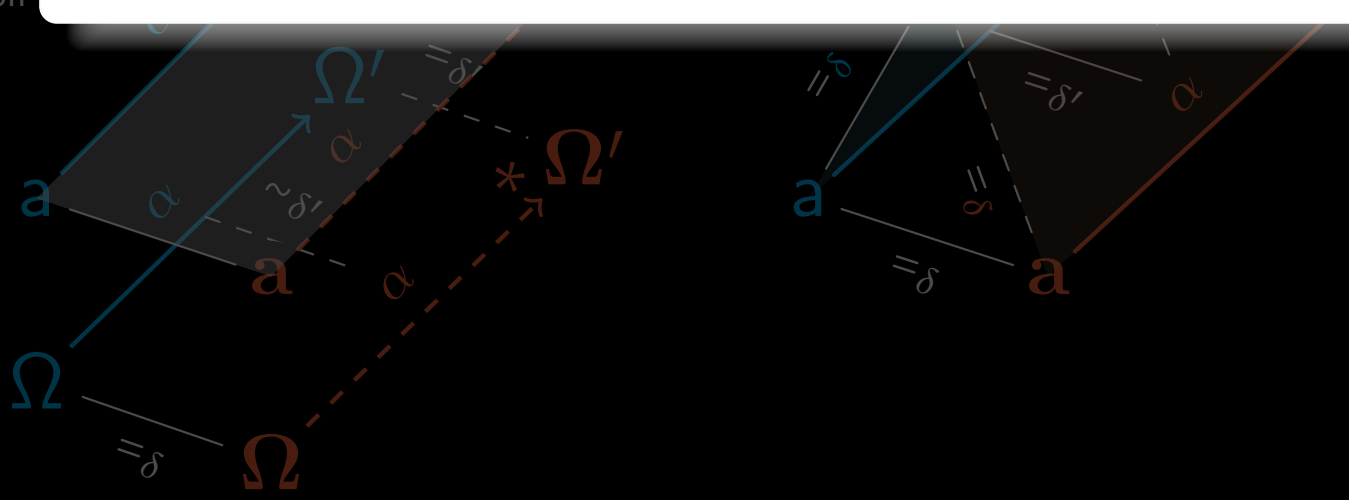
Compiler Properties

PRO: proved MS preservation, MS enforcement

CON: not really RSC (no $\forall \mathbf{A}$)

Challenge: how to ensure $\mathbf{A}$ actions do not affect MS?

Ω = source state
Ω = compiled state
α / α = trace action
$\mathrm{a} / \mathbf{a}$ = allocator
δ = partial bijection
M = monitor state
α = monitor action



Robust Cryptographic Constant Time

(wip)

(Robust) Cryptographic Constant Time

- larger trace model than MS:

(Robust) Cryptographic Constant Time

- larger trace model than MS:
  - memory accesses (as for MS)
  - and timing-relevant operations

(Robust) Cryptographic Constant Time

- larger trace model than MS:
  - memory accesses (as for MS)
  - and timing-relevant operations
- (in)formally RCT: …

(Robust) Cryptographic Constant Time

- larger trace model than MS:
  - memory accesses (as for MS)
  - and timing-relevant operations
- (in)formally RCT: …
  no secret-dependent operations

Bernstein ’15, Barbosa *et al.* S&P’21

Compiler Preserving RCT

- Goal: protect a crypto library from any application using it

Compiler Preserving RCT

- Goal: protect a crypto library from any application using it
- crypto developers already zero out memory before calling apps (e.g., Libsodium)

Compiler Preserving RCT

- Goal: protect a crypto library from any application using it
- crypto developers already zero out memory before calling apps (e.g., Libsodium)
- Challenge: crypto devs must make their code CT

Compiler Preserving RCT

- Goal: protect a crypto library from any application using it
- crypto developers already zero out memory before calling apps (e.g., Libsodium)
- Challenge: crypto devs must make their code CT
- Solution: devise CT code

e.g., Bacelar Almeida *et al.* CCS'17

Compiler Preserving RCT

- Goal: protect a crypto library from any application using it
- crypto developers already zero out memory before calling apps (e.g., Libsodium)
- Challenge: crypto devs must make their code CT
- Solution: devise CT code e.g., Bacelar Almeida *et al.* CCS'17
- Challenge: crypto devs do not know where their code is used

Compiler Preserving RCT

- Goal: protect a crypto library from any application using it
- crypto developers already zero out memory before calling apps (e.g., Libsodium)
- Challenge: crypto devs must make their code CT
- Solution: devise CT code e.g., Bacelar Almeida *et al.* CCS'17
- Challenge: crypto devs do not know where their code is used
- Solution: use a compiler that preserves RCT

Micro-architectural Attacks (Spectre)

CCS'21

Speculative Semantics & SNI

$\mathbf{void\ f\ (int\ x) \mapsto if(x < A.size)\ \{y = B[A[x]]\}}$

run 1: A.size = 16, A[128] = 3

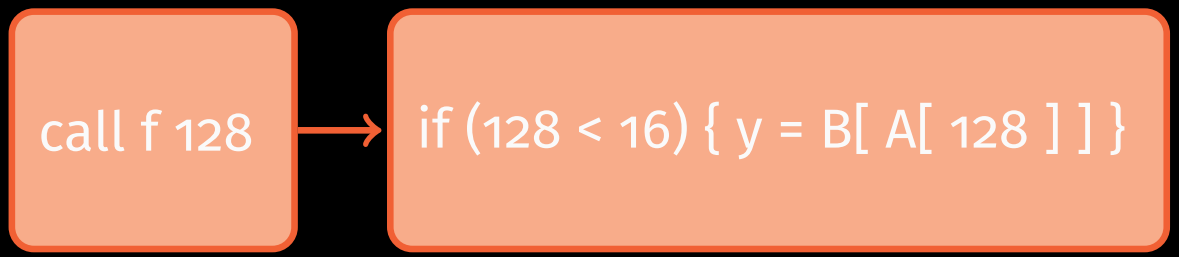
call f 128

Speculative Semantics & SNI

Guarnieri *et al.* S&P'21

$\mathbf{void\ f\ (int\ x) \mapsto if(x < A.size)\ \{y = B[A[x]]\}}$

run 1: A.size = 16, A[128] = 3

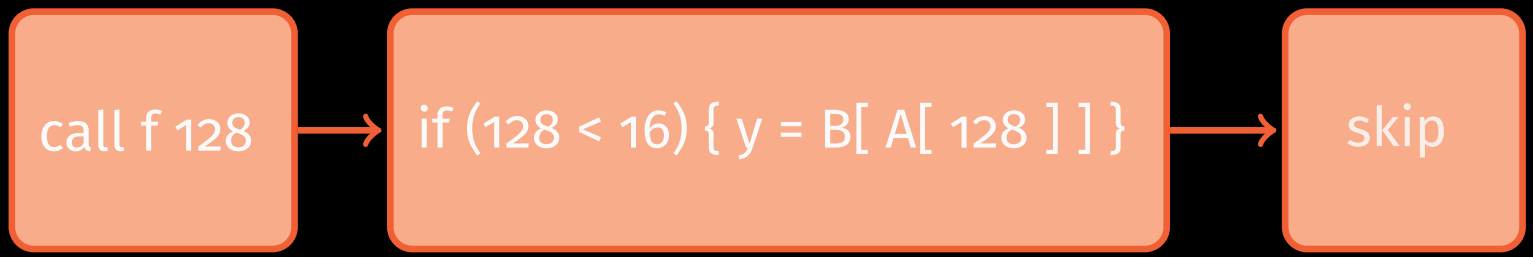


Speculative Semantics & SNI

Guarnieri *et al.* S&P'21

$\mathbf{void\ f\ (int\ x) \mapsto if(x < A.size)\ \{y = B[A[x]]\}}$

run 1: A.size = 16, A[128] = 3

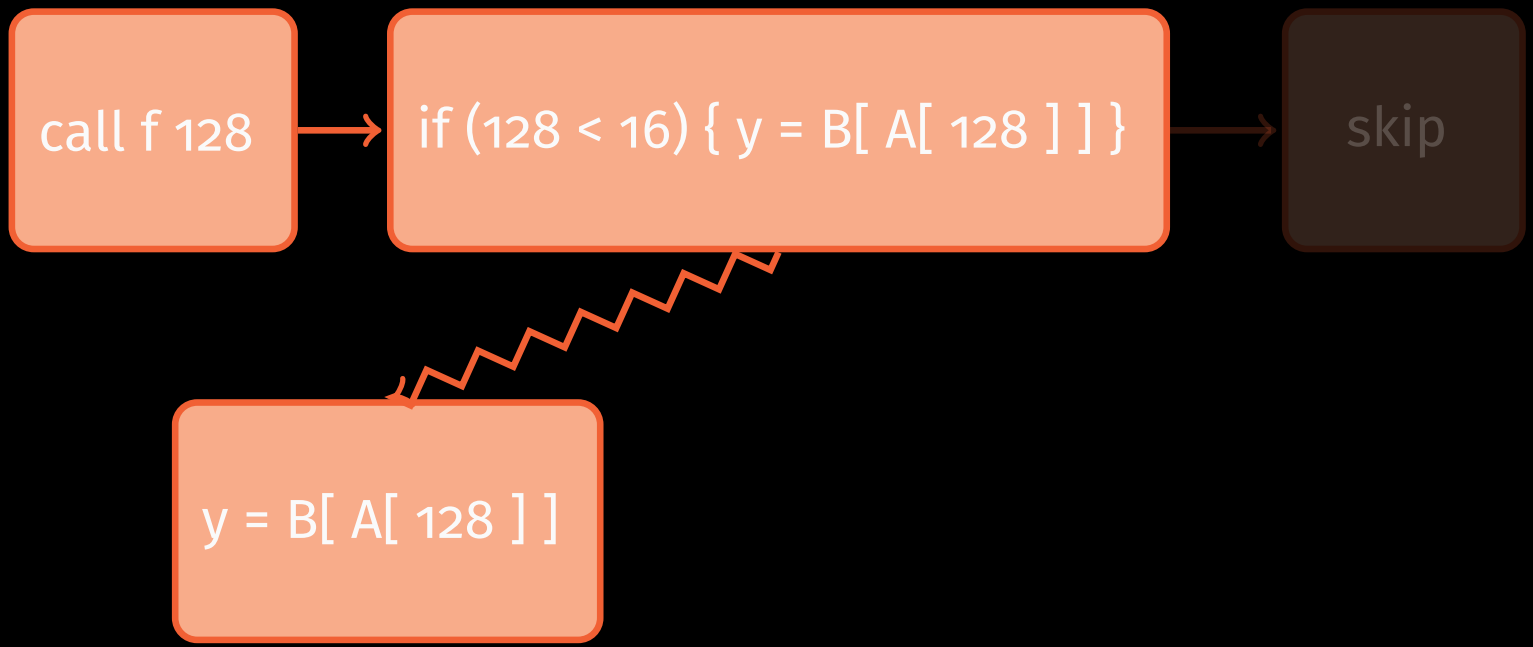


Speculative Semantics & SNI

Guarnieri *et al.* S&P'21

$\mathbf{void\ f\ (int\ x) \mapsto if(x < A.size)\ \{y = B[A[x]]\}}$

run 1: A.size = 16, A[128] = 3

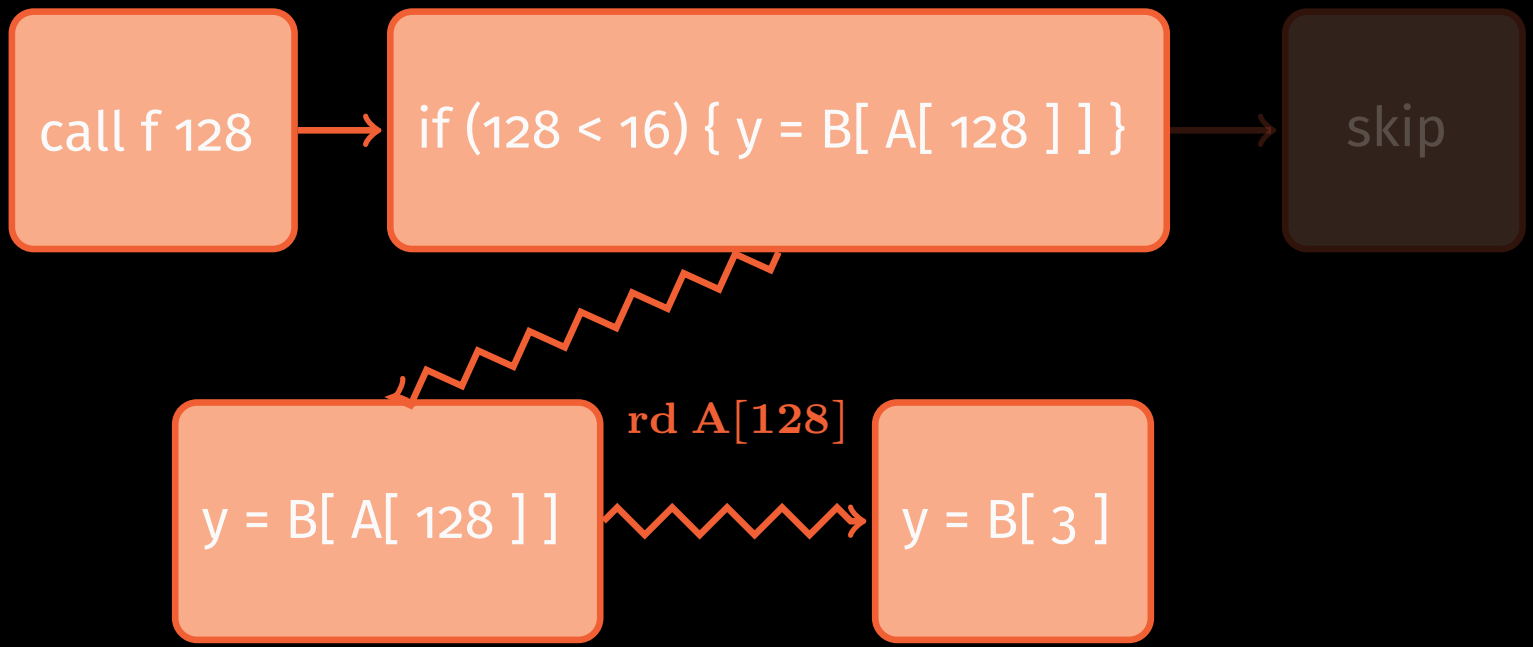


Speculative Semantics & SNI

Guarnieri *et al.* S&P'21

$\mathbf{void\ f\ (int\ x) \mapsto if(x < A.size)\ \{y = B[A[x]]\}}$

run 1: A.size = 16, A[128] = 3

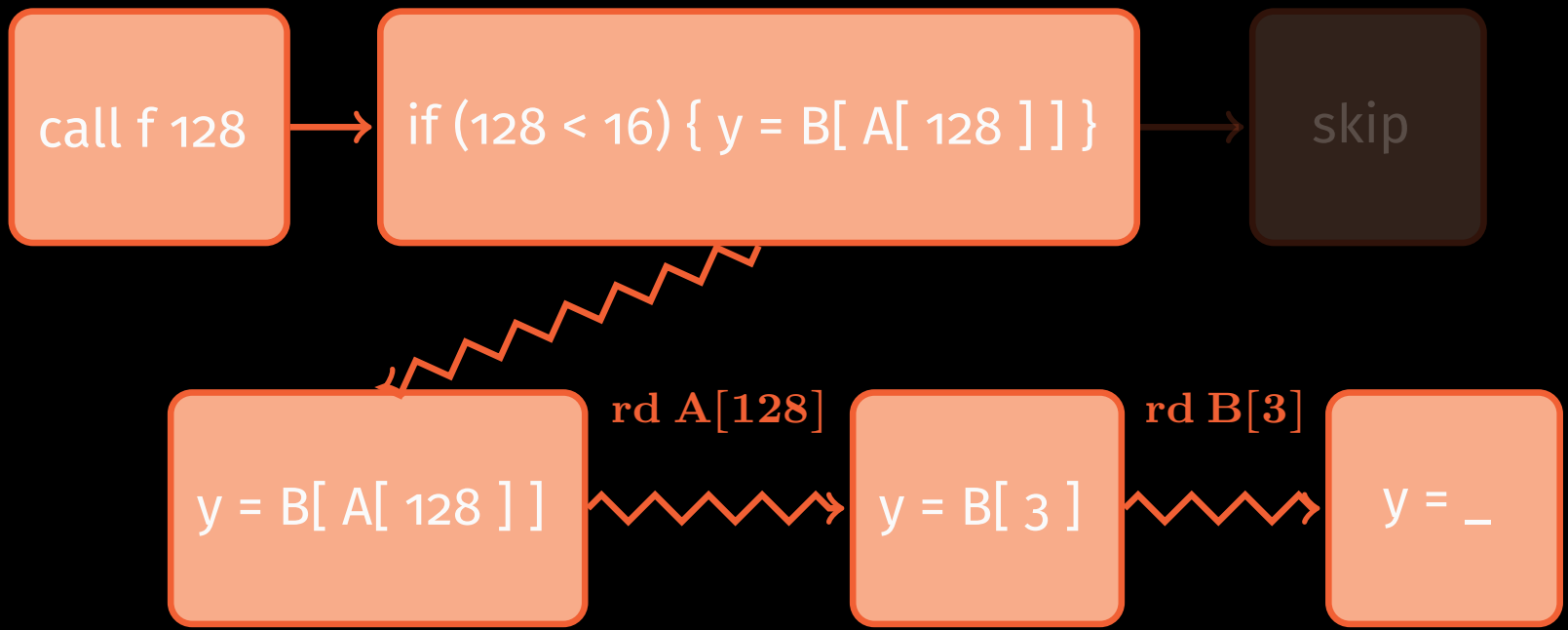


Speculative Semantics & SNI

Guarnieri *et al.* S&P'21

$\mathbf{void\ f\ (int\ x) \mapsto if(x < A.size)\ \{y = B[A[x]]\}}$

run 1: A.size = 16, A[128] = 3

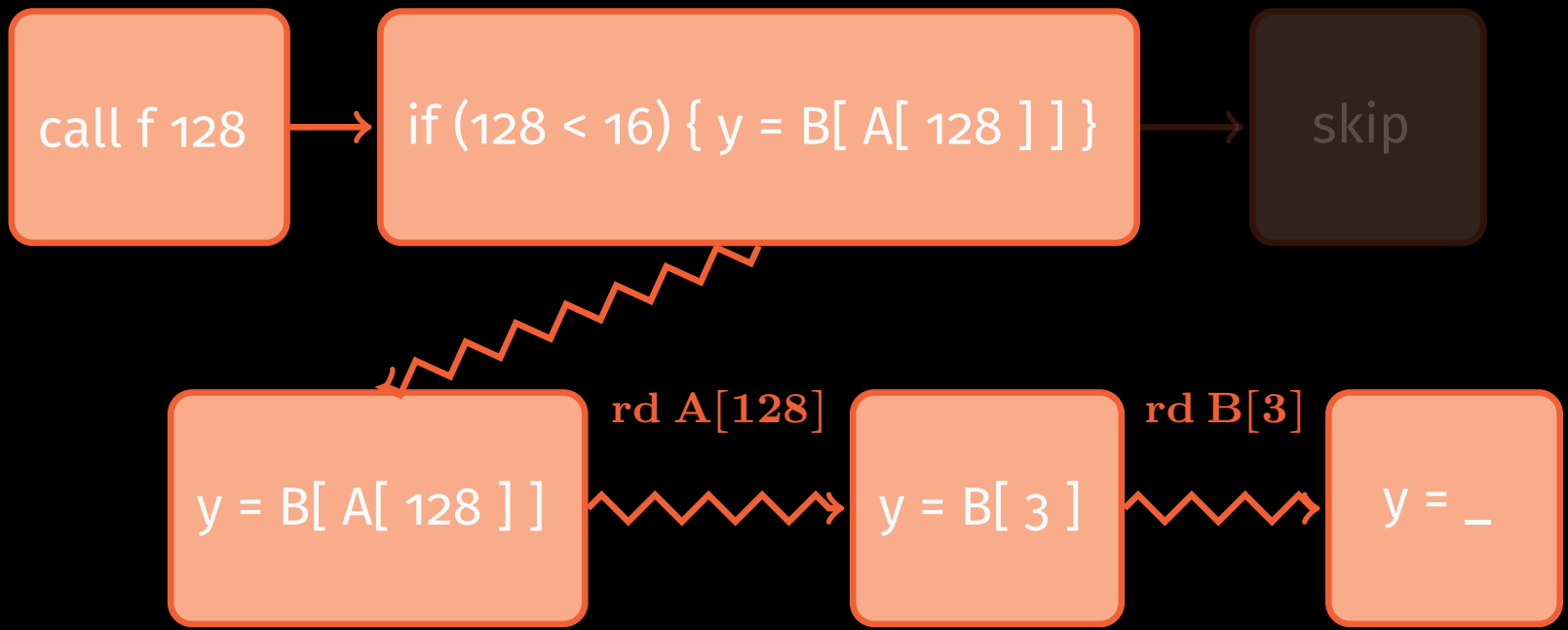


Speculative Semantics & SNI

Guarnieri *et al.* S&P'21

$\mathbf{void\ f\ (int\ x) \mapsto if(x < A.size)\ \{y = B[A[x]]\}}$

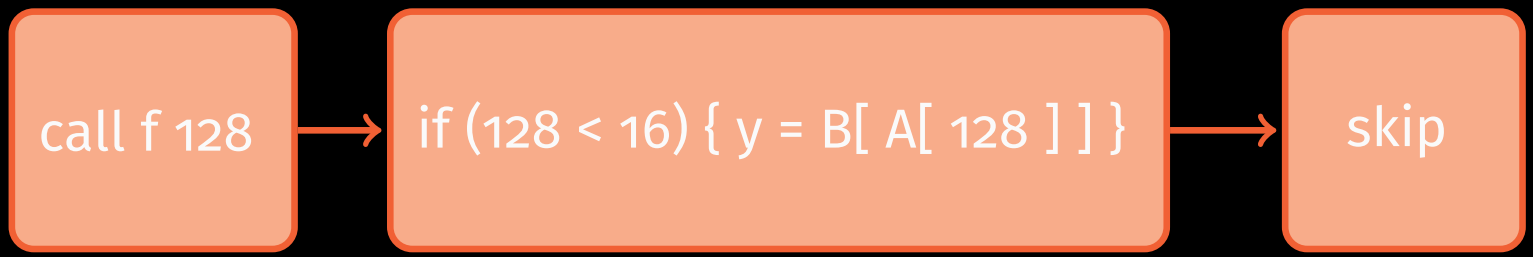
run 1: A.size = 16, A[128] = 3



Speculative Semantics & SNI

$\mathbf{void\ f\ (int\ x) \mapsto if(x < A.size)\ \{y = B[A[x]]\}}$

run 1: A.size = 16, A[128] = 3



trace 1: $\mathbf{rd\ A[128]}$ $\mathbf{rd\ B[3]}$

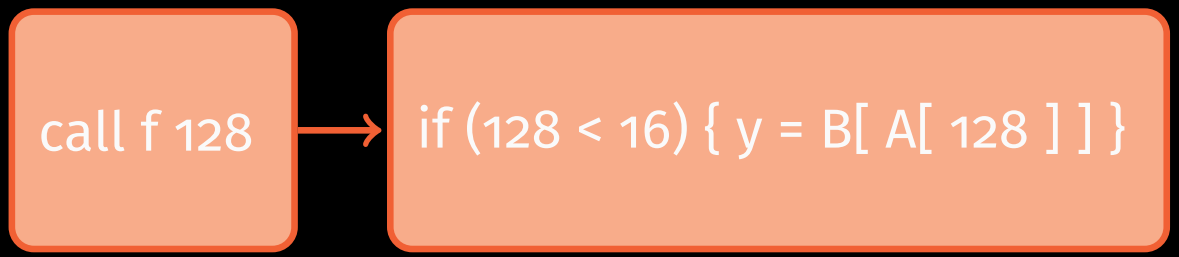
Speculative Semantics & SNI

Guarnieri *et al.* S&P'21

$\mathbf{void\ f\ (int\ x) \mapsto if(x < A.size)\ \{y = B[A[x]]\}}$

run 1: A.size = 16, A[128] = 3

run 2: A[128] = 7 different H values



trace 1: $\mathbf{rd\ A[128]}$ $\mathbf{rd\ B[3]}$

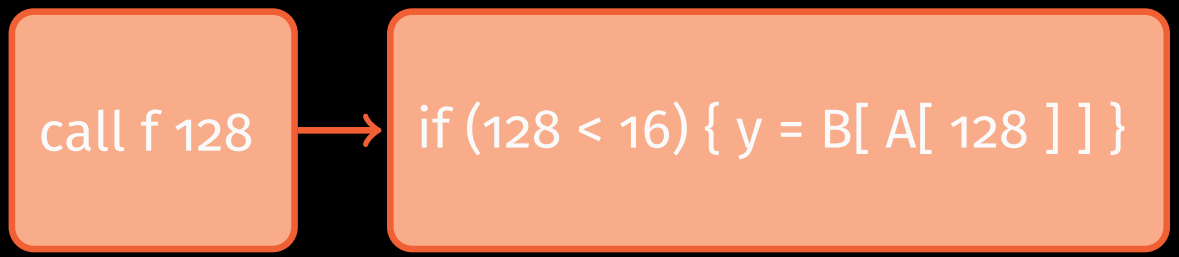
Speculative Semantics & SNI

Guarnieri *et al.* S&P'21

$\mathbf{void\ f\ (int\ x) \mapsto if(x < A.size)\ \{y = B[A[x]]\}}$

run 1: A.size = 16, A[128] = 3

run 2: A[128] = 7 different H values



trace 1: $\mathbf{rd\ A[128]}$ $\mathbf{rd\ B[3]}$

$\mathbf{rd\ A[128]}$

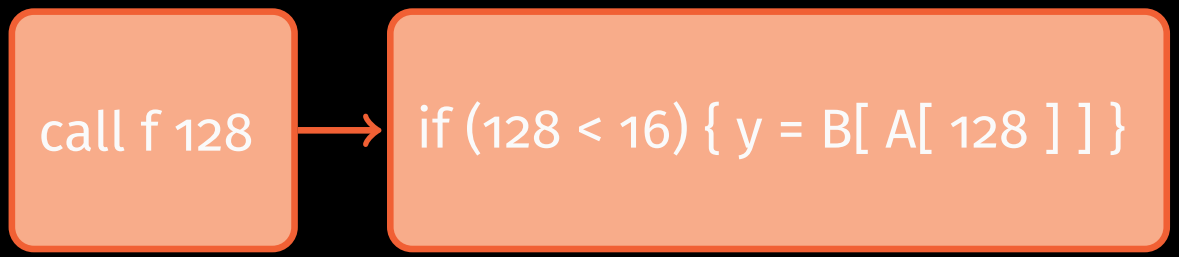
Speculative Semantics & SNI

Guarnieri *et al.* S&P'21

$\mathbf{void\ f\ (int\ x) \mapsto if(x < A.size)\ \{y = B[A[x]]\}}$

run 1: A.size = 16, A[128] = 3

run 2: A[128] = 7 different H values



trace 1: $\mathbf{rd\ A[128]}$ $\mathbf{rd\ B[3]}$

$\mathbf{rd\ A[128]}$ $\mathbf{rd\ B[7]}$

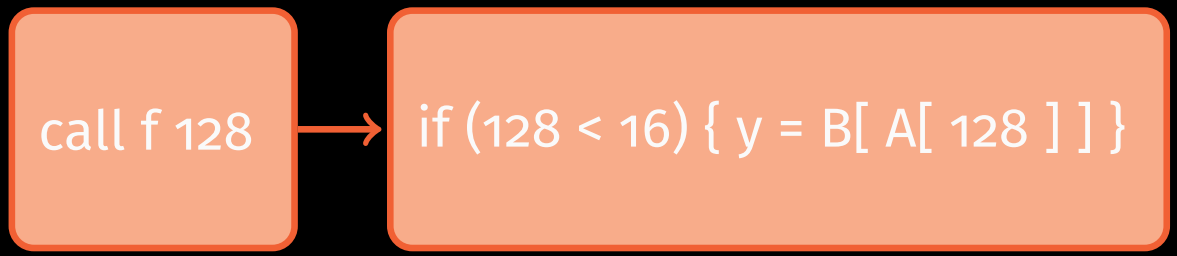
Speculative Semantics & SNI

Guarnieri *et al.* S&P'21

$\mathbf{void\ f\ (int\ x) \mapsto if(x < A.size)\ \{y = B[A[x]]\}}$

run 1: A.size = 16, A[128] = 3

run 2: A[128] = 7 different H values



trace 1: $\mathbf{rd\ A[128]}$ $\mathbf{rd\ B[3]}$ different traces

trace 2: $\mathbf{rd\ A[128]}$ $\mathbf{rd\ B[7]}$ $\Rightarrow$ SNI violation

Speculative Semantics & SNI

Guarnieri *et al.* S&P'21

A program is SNI ($\vdash \mathbf{P} : \mathbf{SNI}$) if, given two runs from low-equivalent states:

- assuming the non-speculative traces are low-equivalent
- then the speculative traces are also low-equivalent

call f

trace 1: $\mathbf{rd\ A[128]}$ $\mathbf{rd\ B[3]}$
trace 2: $\mathbf{rd\ A[128]}$ $\mathbf{rd\ B[7]}$

different traces $\Rightarrow$ SNI violation

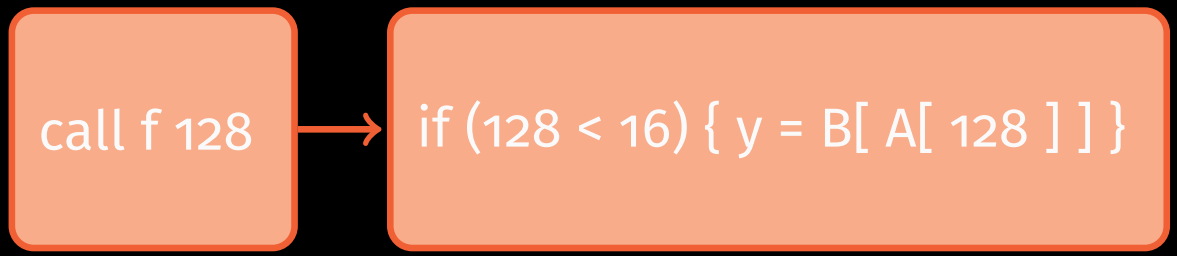
Speculative Semantics & SNI

Guarnieri *et al.* S&P'21

$\mathbf{void\ f\ (int\ x) \mapsto if(x < A.size)\ \{y = B[A[x]]\}}$

run 1: A.size = 16, A[128] = 3

run 2: A[128] = 7 different H values



trace 1: $\mathbf{rd\ A[128]}$ $\mathbf{rd\ B[3]}$ different traces

trace 2: $\mathbf{rd\ A[128]}$ $\mathbf{rd\ B[7]}$ $\Rightarrow$ SNI violation

Problems Problems Problems ...

Problem: Proving compiler preserves SNI is hard

Problems Problems Problems ...

Problem: Proving compiler preserves SNI is hard

Solution: overapproximate SNI with a novel property: speculative safety (SS)

Speculative Safety (SS): Taint Tracking

$\mathbf{void\ f\ (int\ x) \mapsto if(x < A.size)\ \{y = B[A[x]]\}}$

only 1 run needed: A.size=16, A[128]=3

integrity lattice: $S \subset U \quad S \sqcap U = S \quad U$ does not flow to S

call f 128

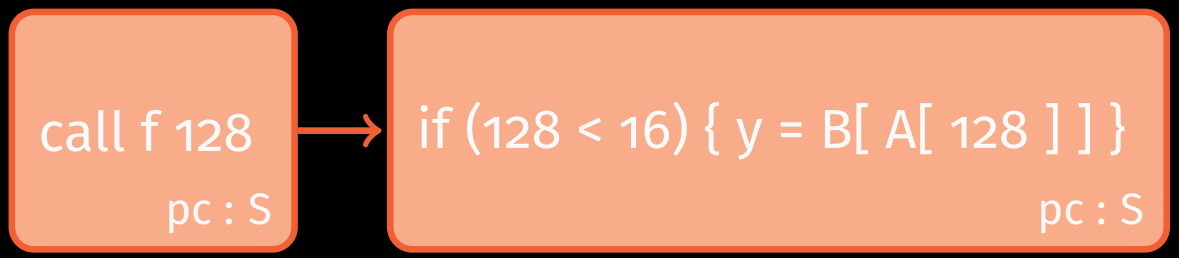
pc : S

Speculative Safety (SS): Taint Tracking

$\mathbf{void\ f\ (int\ x) \mapsto if(x < A.size)\ \{y = B[A[x]]\}}$

only 1 run needed: A.size=16, A[128]=3

integrity lattice: $S \subset U \quad S \sqcap U = S \quad U$ does not flow to S

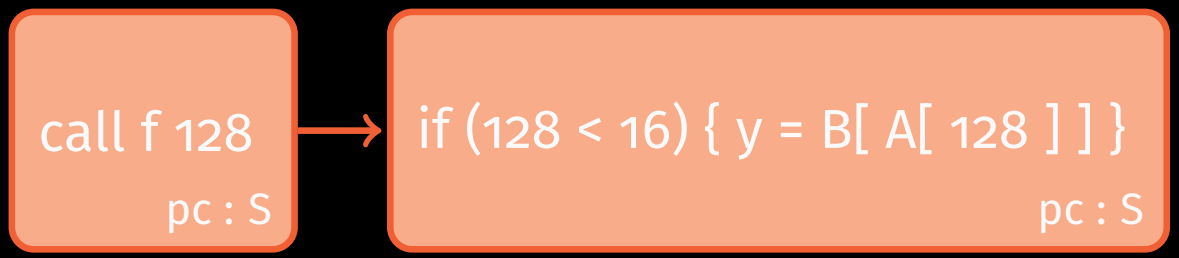


Speculative Safety (SS): Taint Tracking

$\mathbf{void\ f\ (int\ x) \mapsto if(x < A.size)\ \{y = B[A[x]]\}}$

only 1 run needed: A.size=16, A[128]=3

integrity lattice: $S \subset U \quad S \sqcap U = S \quad U$ does not flow to S

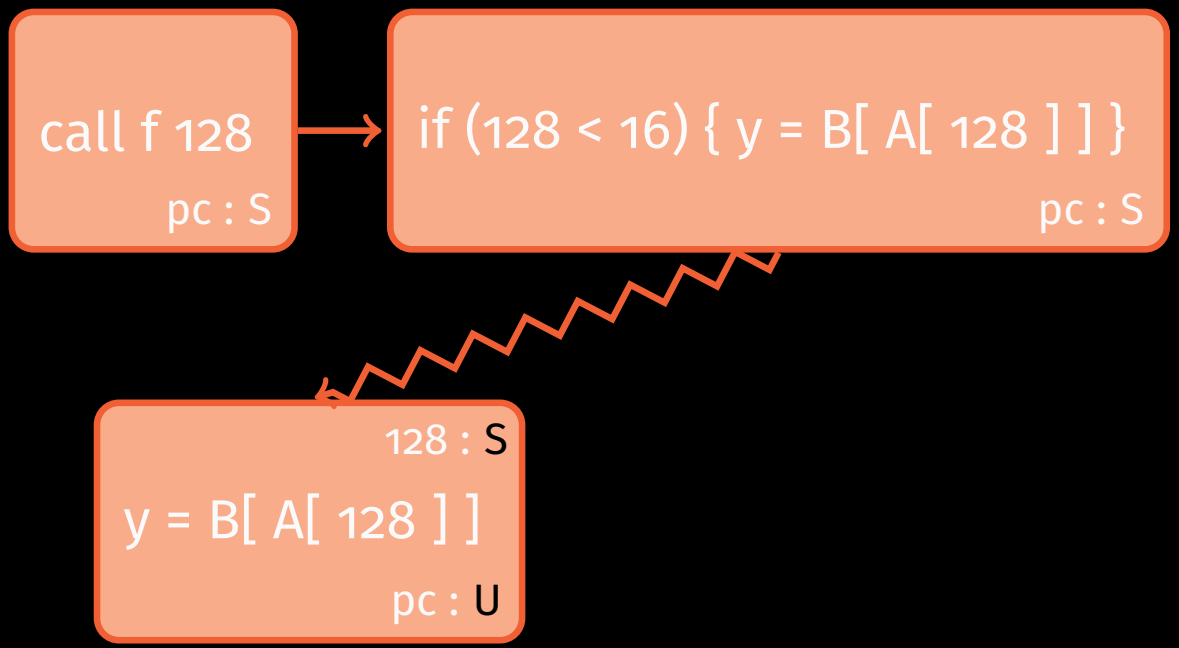


Speculative Safety (SS): Taint Tracking

$\mathbf{void\ f\ (int\ x) \mapsto if(x < A.size)\ \{y = B[A[x]]\}}$

only 1 run needed: A.size=16, A[128]=3

integrity lattice: $S \subset U \quad S \sqcap U = S \quad U$ does not flow to S

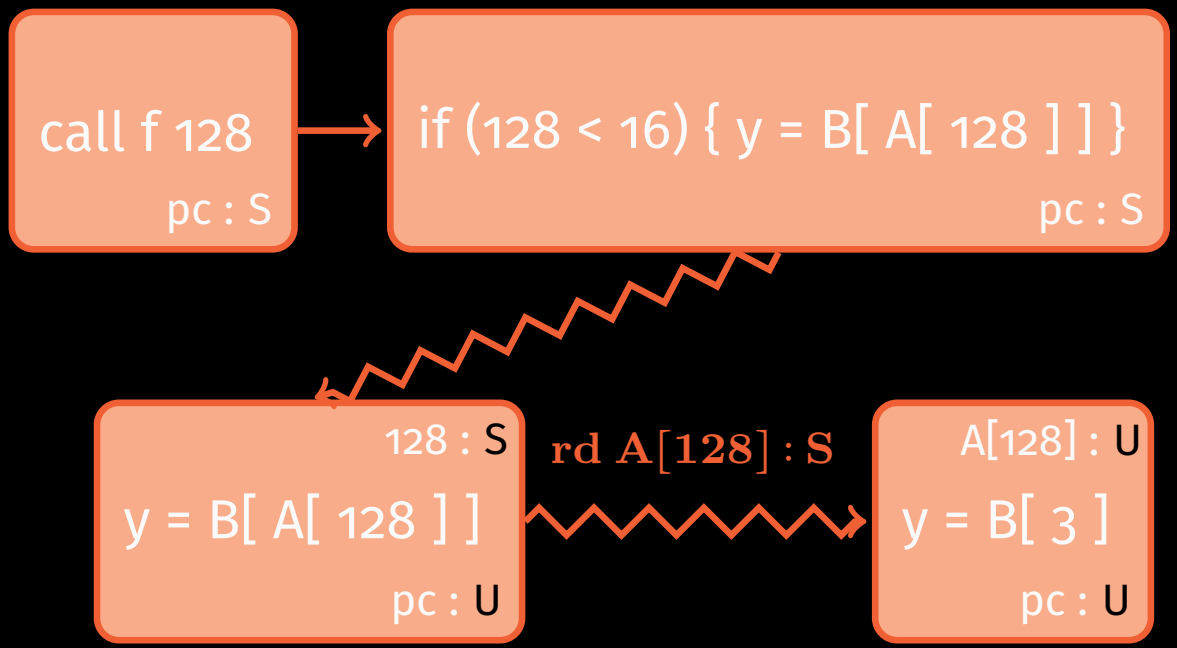


Speculative Safety (SS): Taint Tracking

$\mathbf{void\ f\ (int\ x) \mapsto if(x < A.size)\ \{y = B[A[x]]\}}$

only 1 run needed: A.size=16, A[128]=3

integrity lattice: $S \subset U \quad S \sqcap U = S \quad U$ does not flow to S

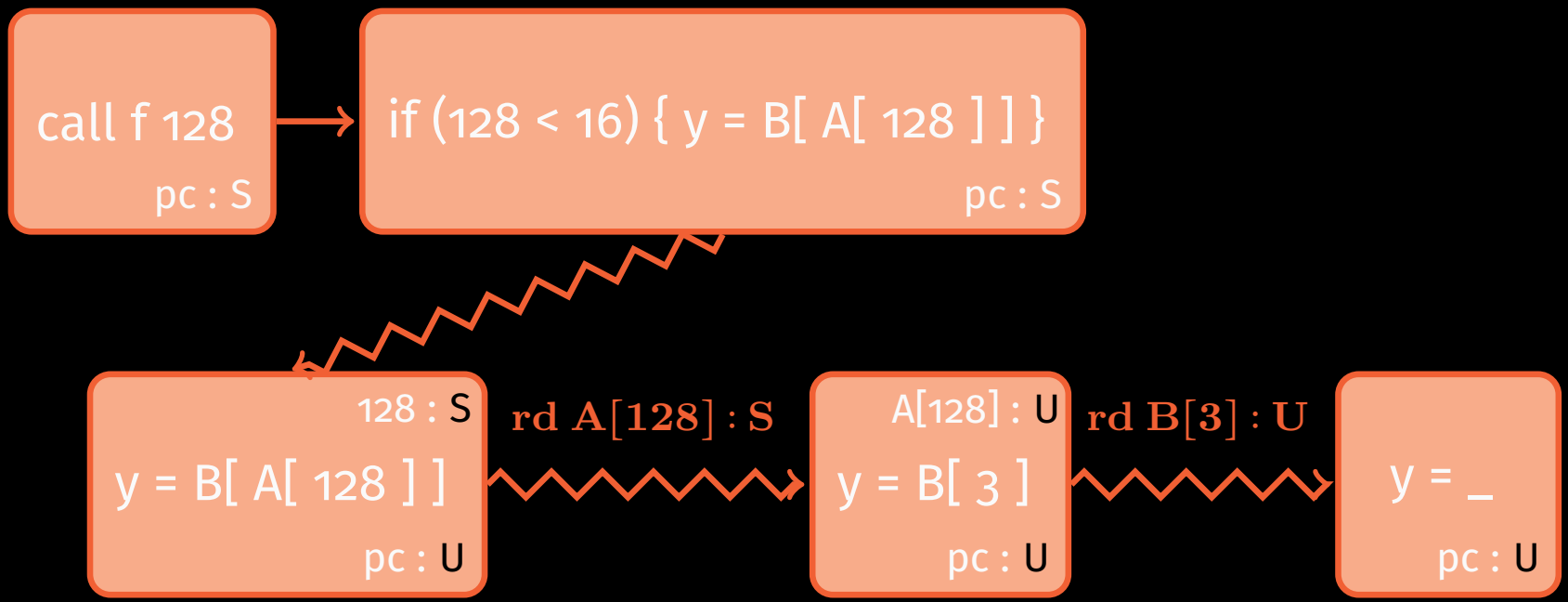


Speculative Safety (SS): Taint Tracking

$\mathbf{void\ f\ (int\ x) \mapsto if(x < A.size)\ \{y = B[A[x]]\}}$

only 1 run needed: A.size=16, A[128]=3

integrity lattice: $S \subset U \quad S \sqcap U = S \quad U$ does not flow to S

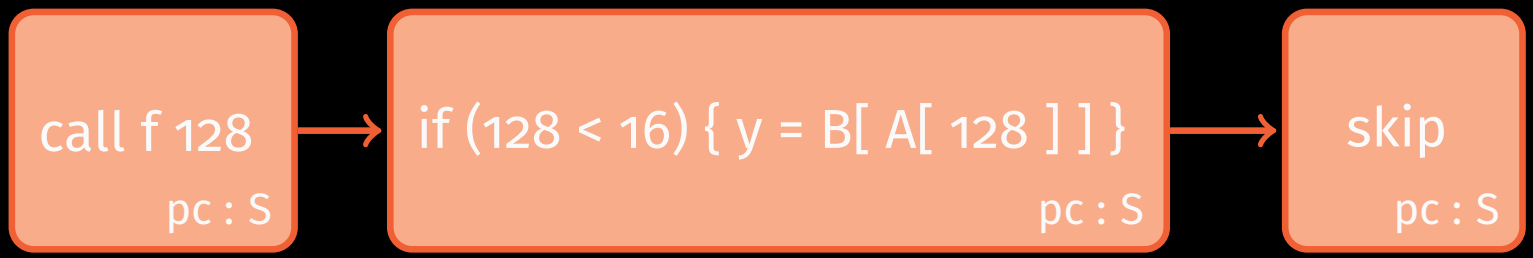


Speculative Safety (SS): Taint Tracking

$\mathbf{void\ f\ (int\ x) \mapsto if(x < A.size)\ \{y = B[A[x]]\}}$

only 1 run needed: A.size=16, A[128]=3

integrity lattice: $S \subset U \quad S \sqcap U = S \quad U$ does not flow to S



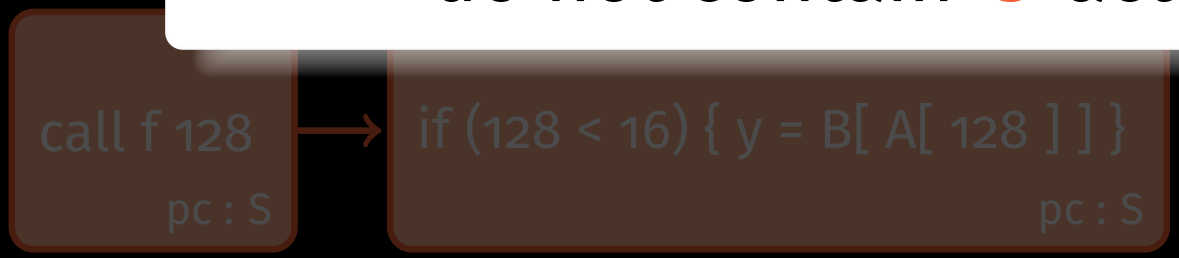
$\mathbf{rd\ A[128] : S}$ $\mathbf{rd\ B[3] : U}$

Speculative Safety (SS): Taint Tracking

$\mathbf{void\ f\ (int\ x) \mapsto if(x < A.size)\ \{y = B[A[x]]\}}$

only 1 run needed: A.size=16, A[128]=3

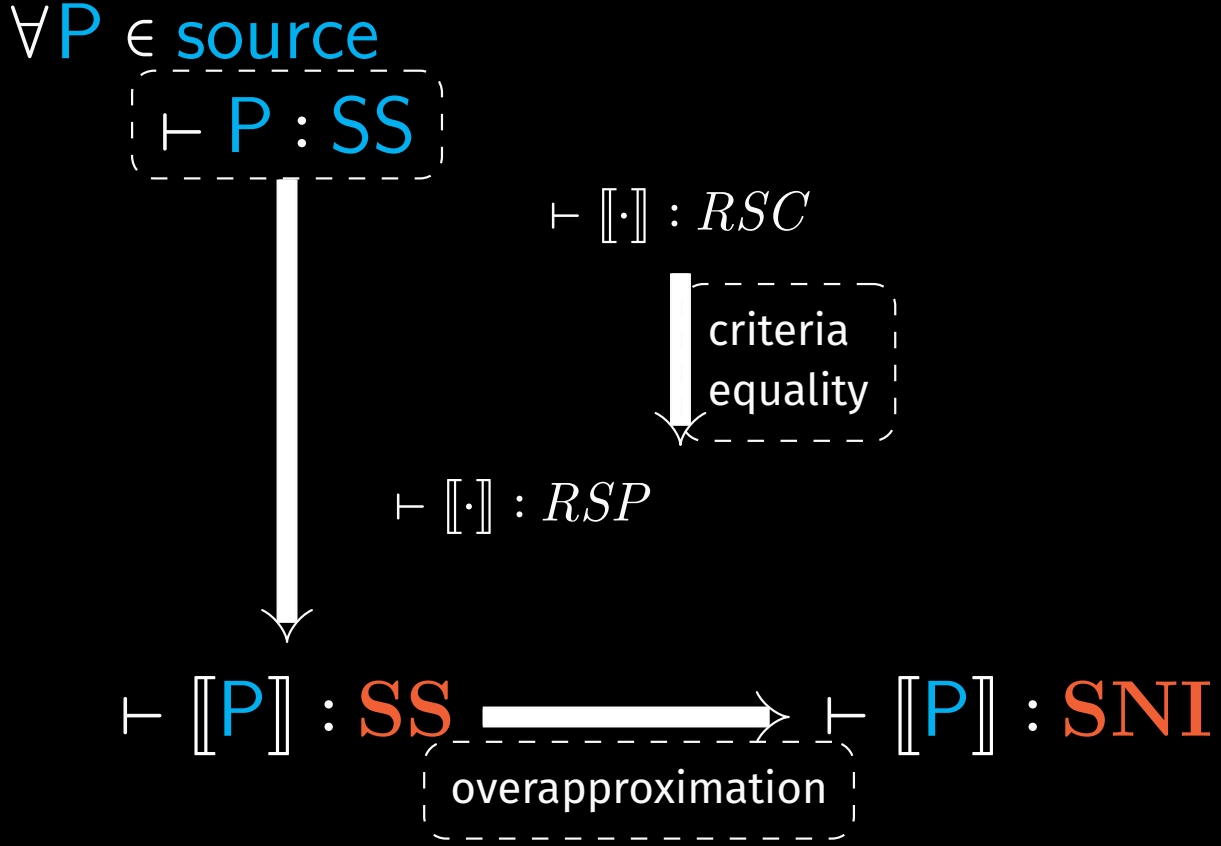
A program is SS ($\vdash \mathbf{P} : \mathbf{SS}$) if its traces do not contain $\mathbf{U}$ actions



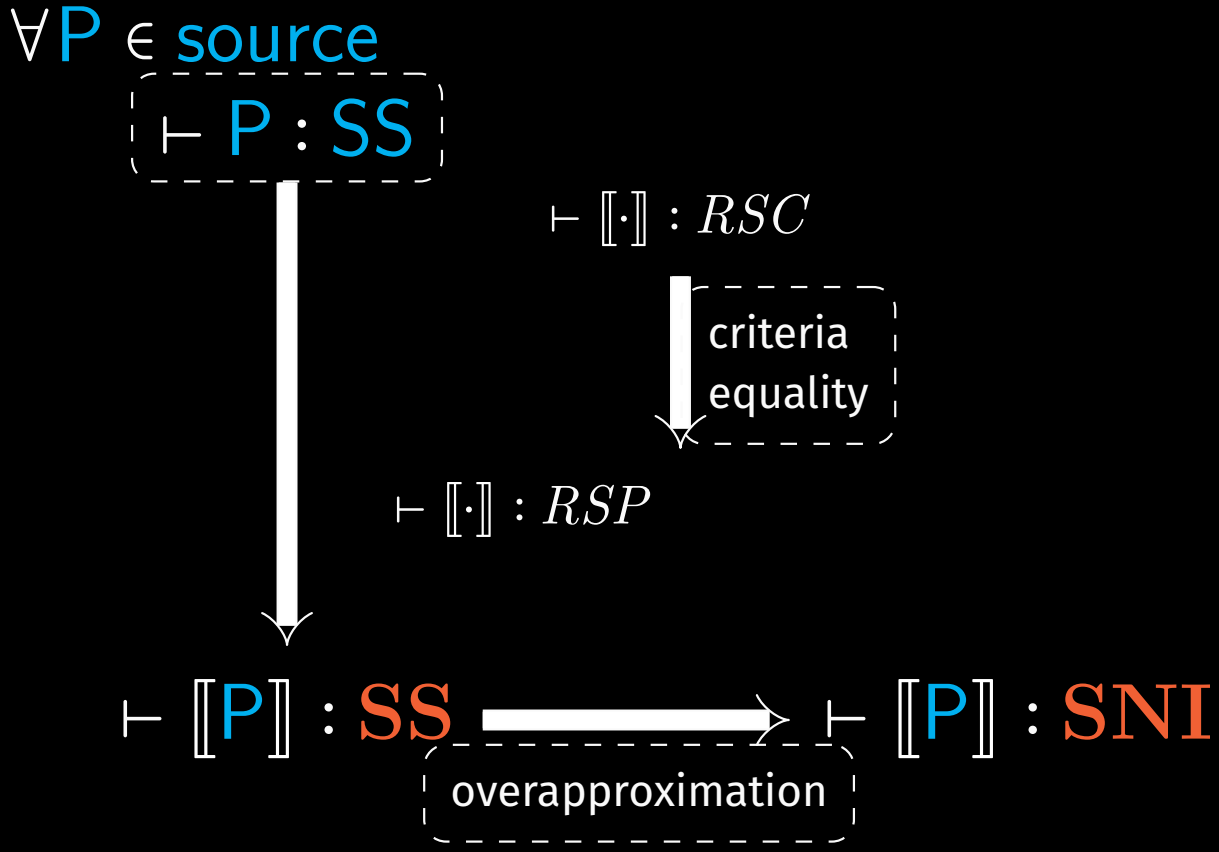
$\mathbf{rd\ A[128] : S}$ $\mathbf{rd\ B[3] : U}$

Secure Compilation Framework for Spectre

Secure Compilation Framework for Spectre

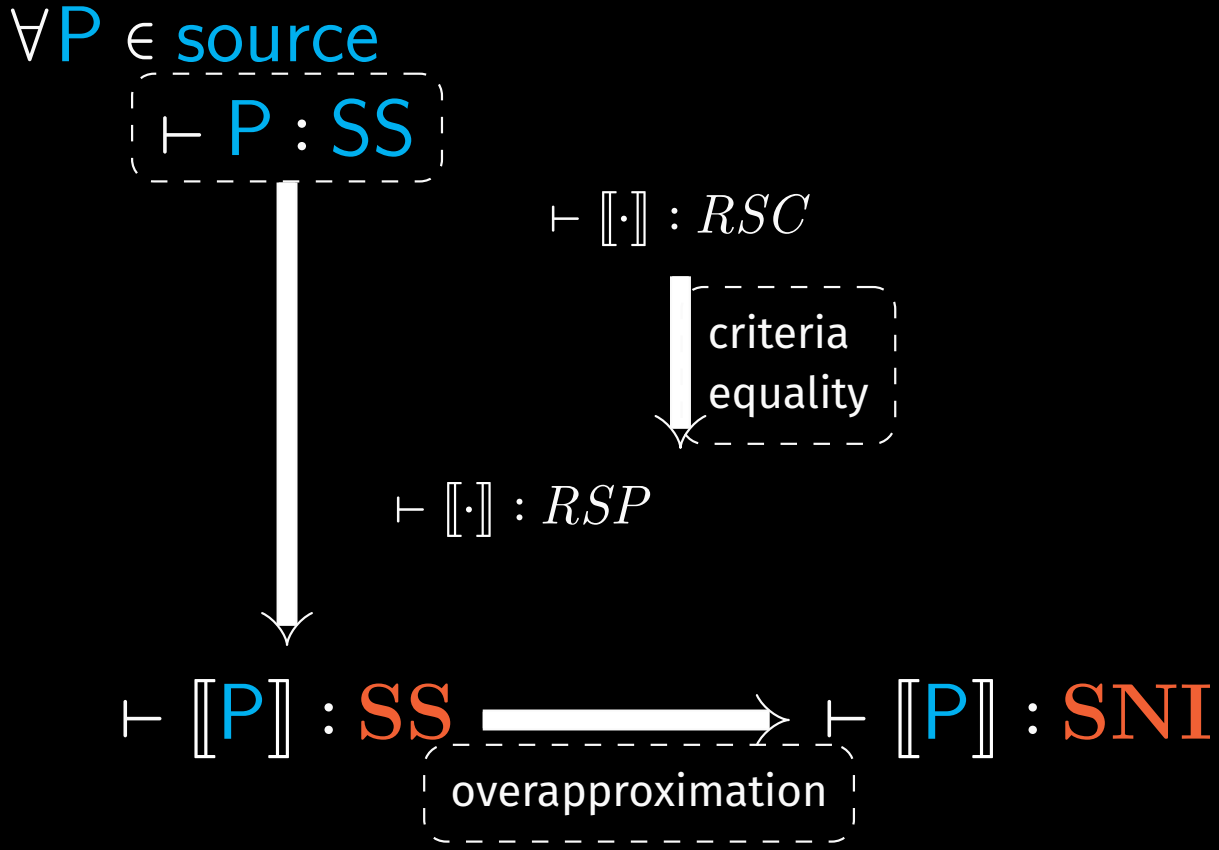


Secure Compilation Framework for Spectre



- dashed premises are already discharged

Secure Compilation Framework for Spectre



- dashed premises are already discharged
- to show security: simply prove RSC

RSSC **for** `lfence`

$$\text{void f(int x)} \mapsto \text{if(x < A.size)\{y = B[A[x]]\}} \qquad \text{// A.size=16, A[128]=3}$$

$$[\![\cdot]\!] = \mathbf{void\ f(int\ x) \mapsto if(x < A.size)\{lfence; y = B[A[x]]\}}$$

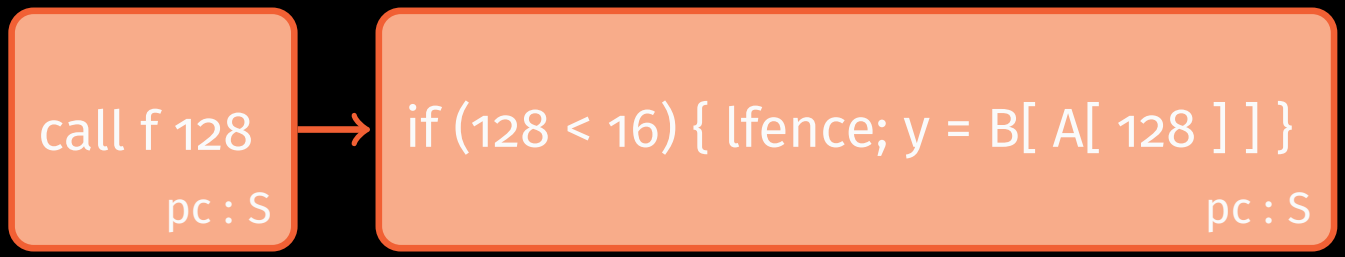
call f 128

pc : S

RSSC **for** `lfence`

$$\text{void f(int x)} \mapsto \text{if(x < A.size)\{y = B[A[x]]\}} \qquad \text{// A.size=16, A[128]=3}$$

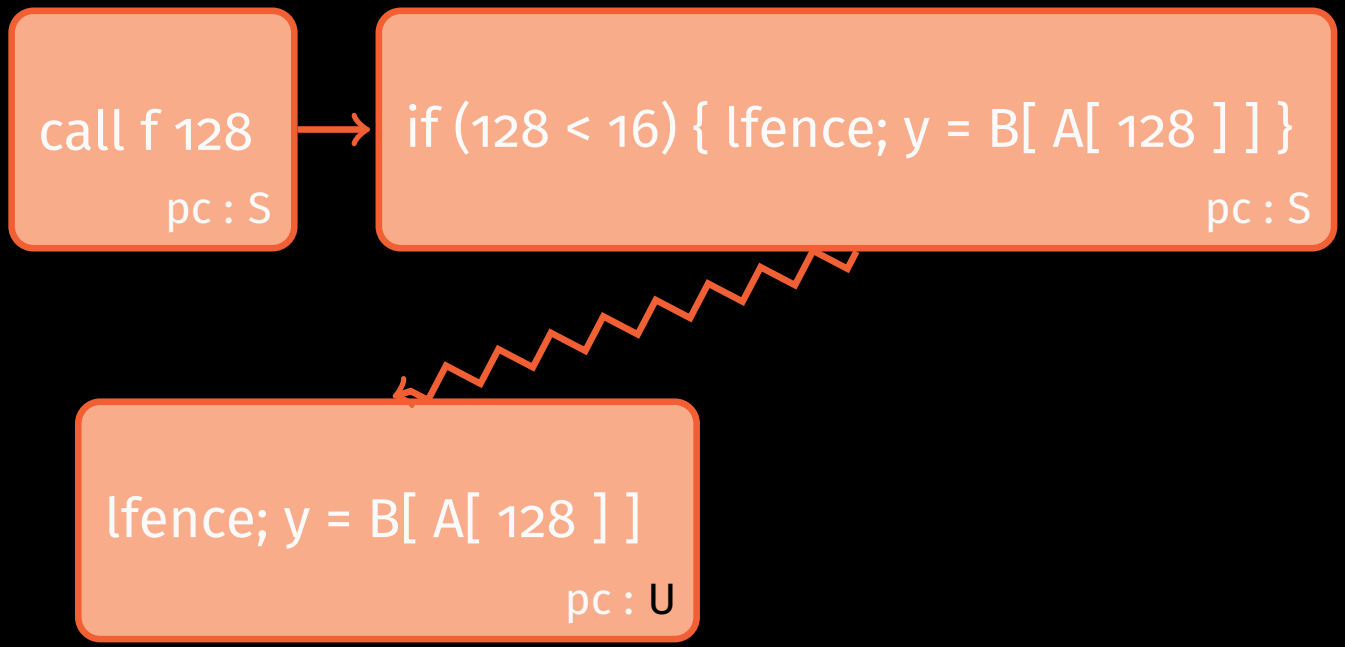
$$[\![\cdot]\!] = \mathbf{void\ f(int\ x) \mapsto if(x < A.size)\{lfence; y = B[A[x]]\}}$$



RSSC **for** `lfence`

void f(int x) ↦ if(x < A.size){y = B[A[x]]} // A.size=16, A[128]=3

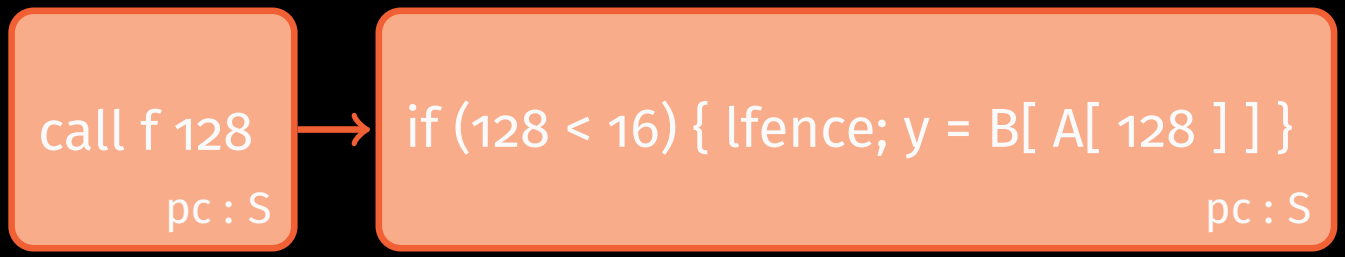
$[\![\cdot]\!]$ = $\mathbf{void\ f(int\ x) \mapsto if(x < A.size)\{lfence; y = B[A[x]]\}}$



RSSC **for** `lfence`

$$\text{void f(int x)} \mapsto \text{if(x < A.size)\{y = B[A[x]]\}} \qquad \text{// A.size=16, A[128]=3}$$

$$[\![\cdot]\!] = \mathbf{void\ f(int\ x) \mapsto if(x < A.size)\{lfence; y = B[A[x]]\}}$$

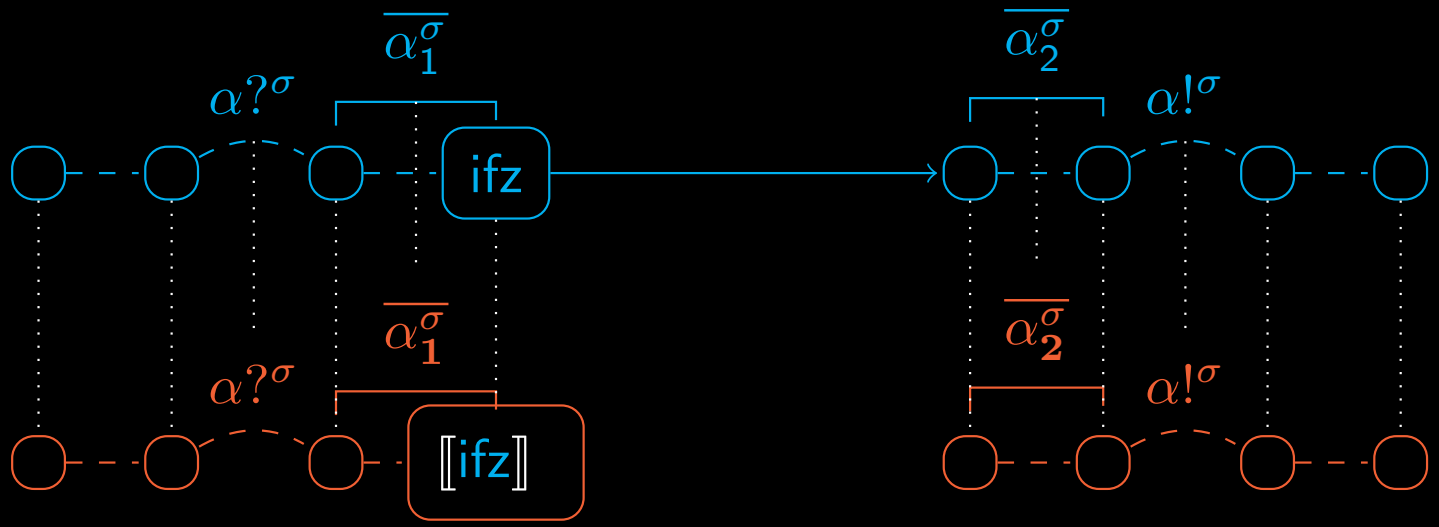


RSSC **for** `lfence`

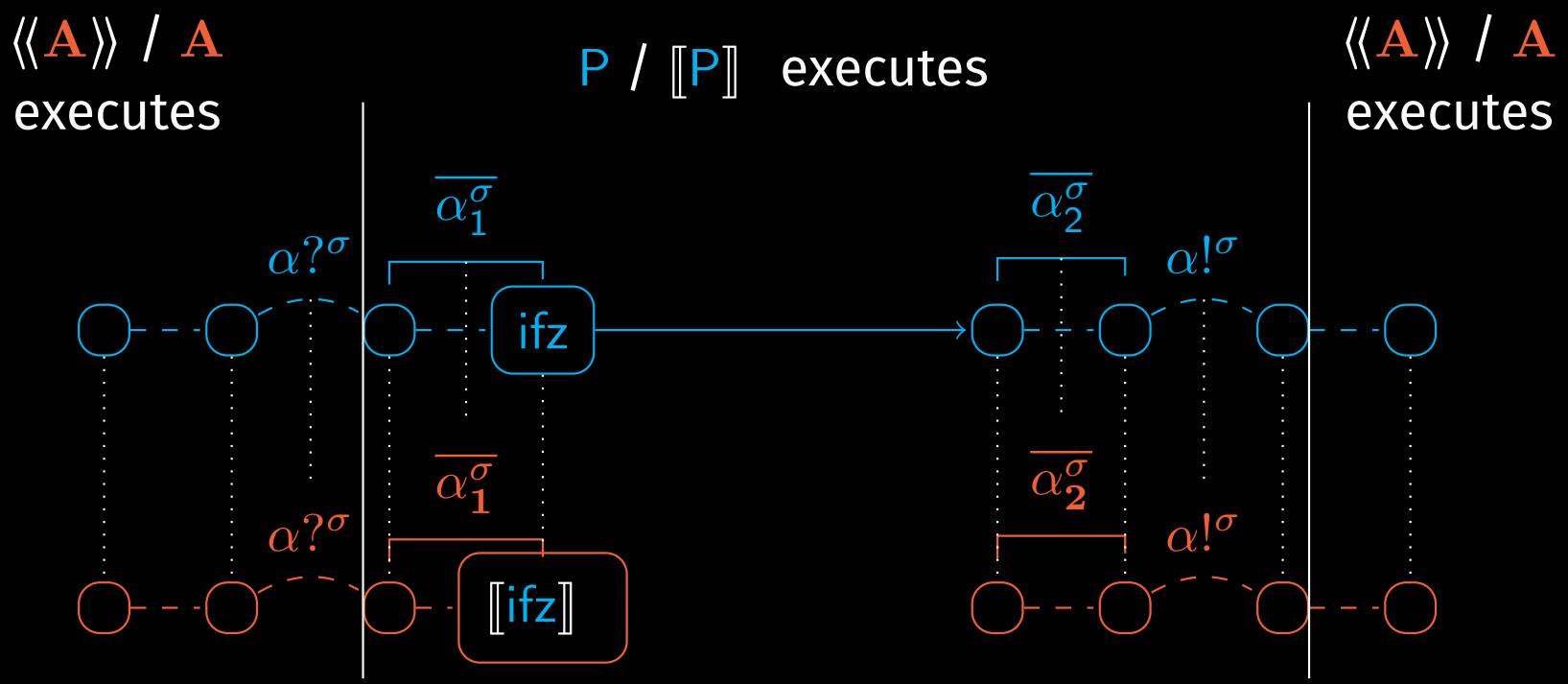
$$\text{void f(int x)} \mapsto \text{if(x < A.size)\{y = B[A[x]]\}} \qquad \text{// A.size=16, A[128]=3}$$

$$[\![\cdot]\!] = \mathbf{void\ f(int\ x) \mapsto if(x < A.size)\{lfence; y = B[A[x]]\}}$$

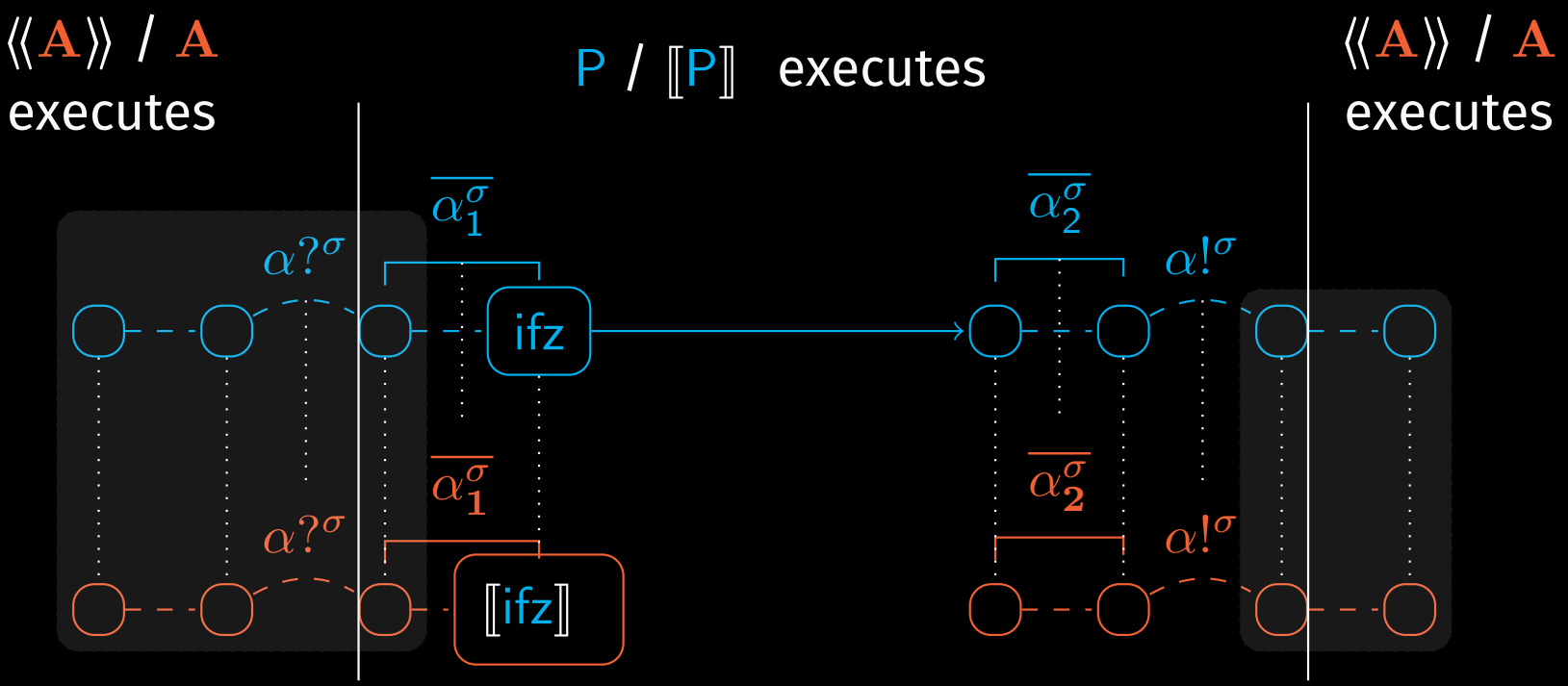
Proofs Insight



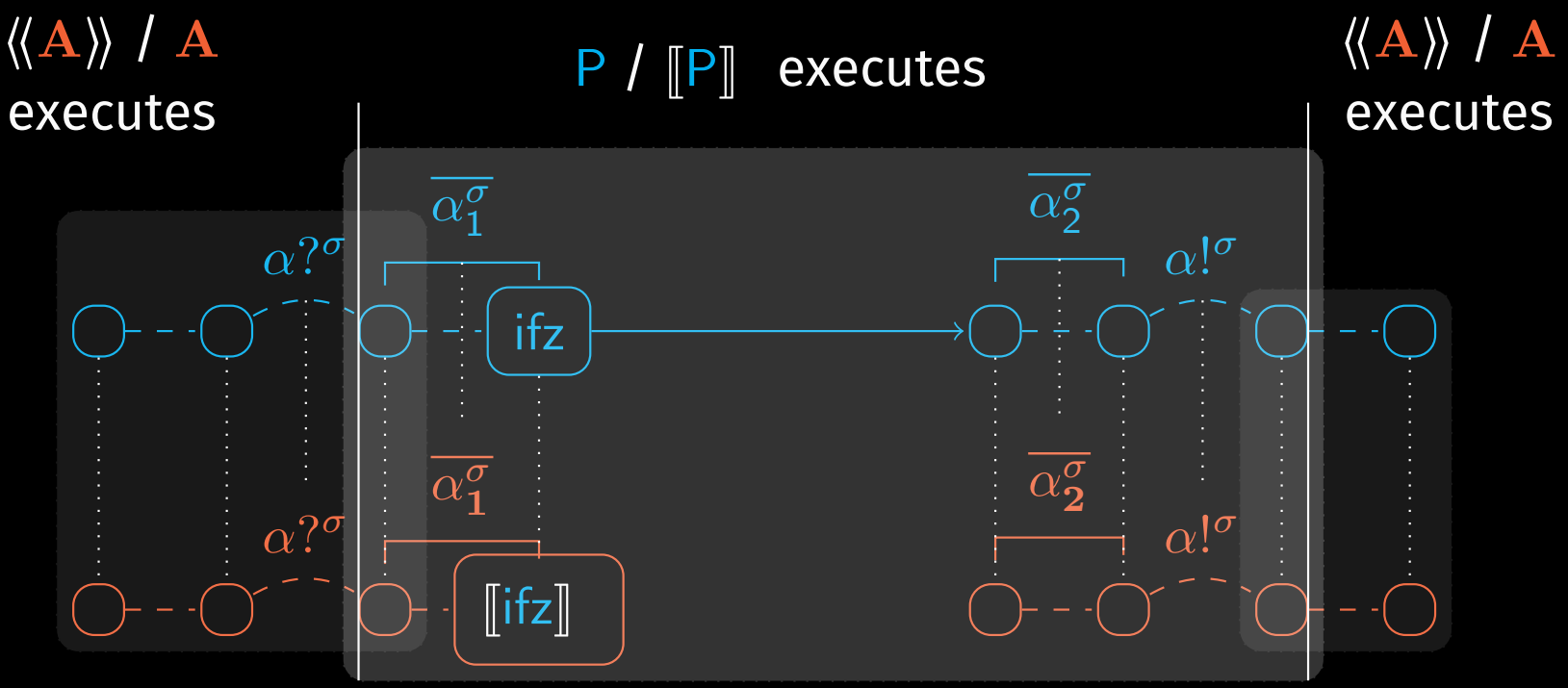
Proofs Insight



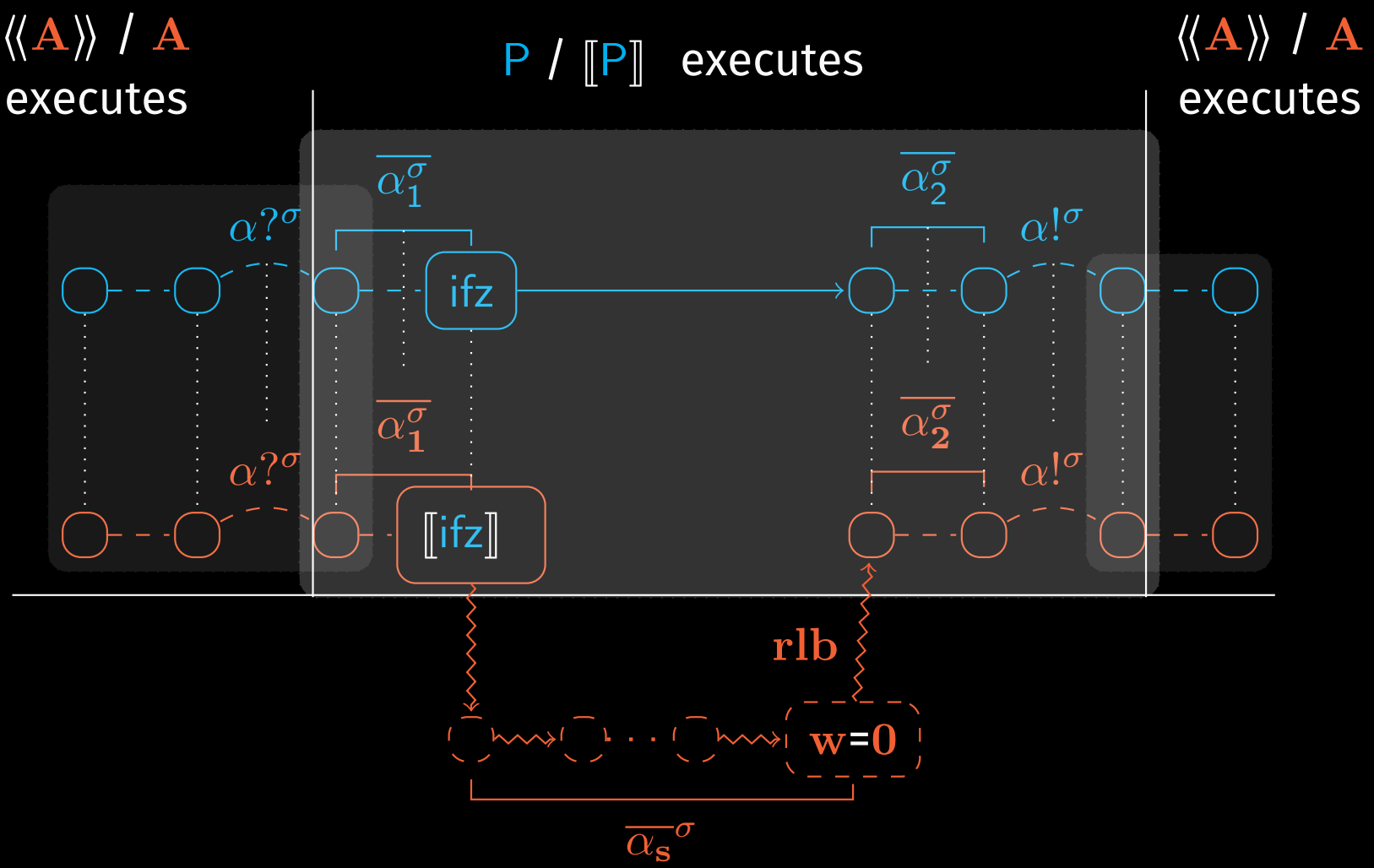
Proofs Insight



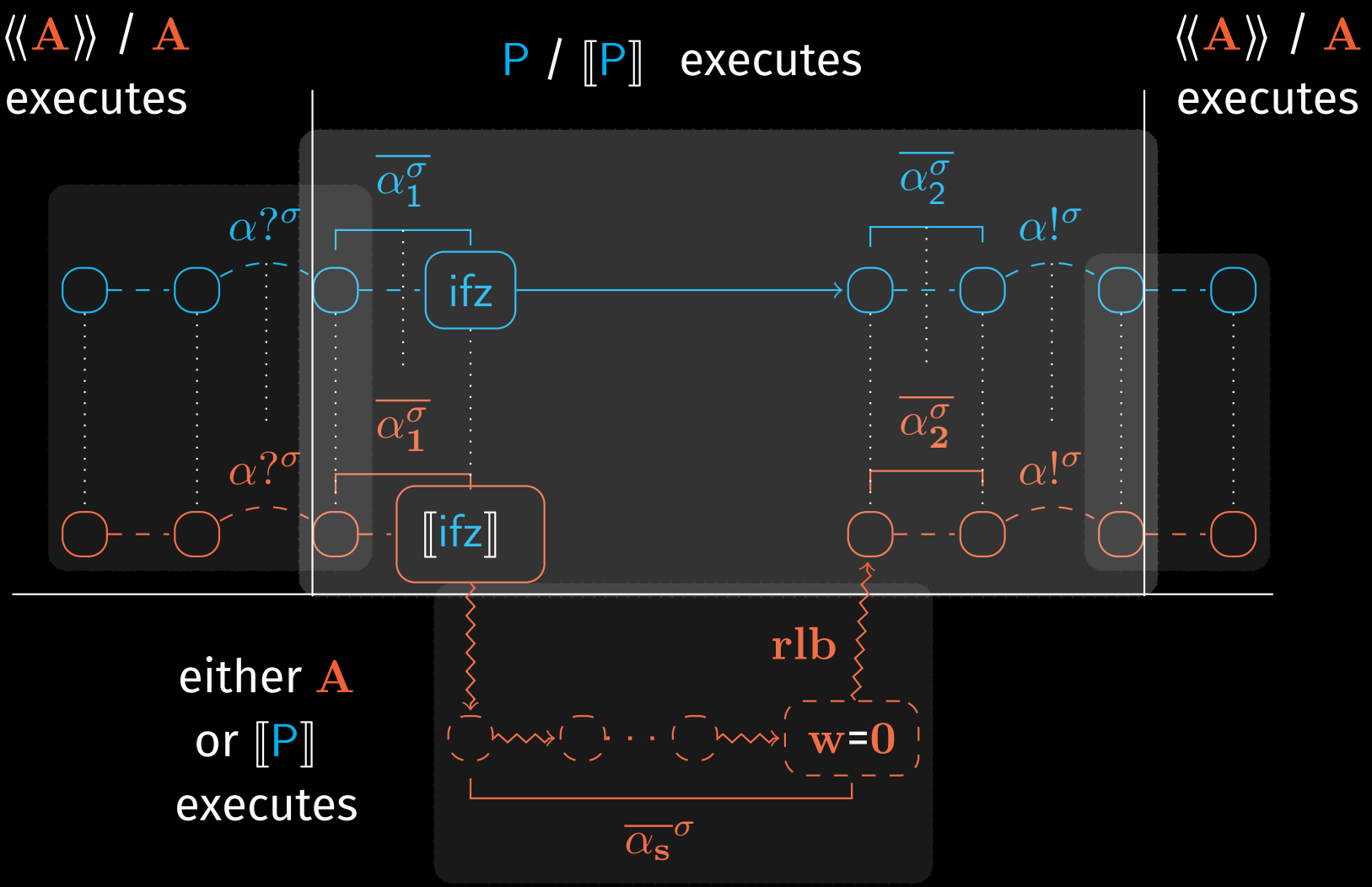
Proofs Insight



Proofs Insight



Proofs Insight



What Then?

- SNIv1, SNIv2, SNIv4, SNIv5

Kocher *et al.* S&P'19

What Then?

- SNIv1, SNIv2, SNIv4, SNIv5

Kocher *et al.* S&P'19

- Challenge: can the lfence compiler "mess" with SNIv2?

What Then?

- SNIv1, SNIv2, SNIv4, SNIv5

Kocher *et al.* S&P'19

- Challenge: can the lfence compiler "mess" with SNIv2?
- Challenge: can we compose lfence(SNIv1) and retpoline(SNIv5)?

Security Architectures

(e.g., Cheri/ARM Morello, Sancus/Intel SGX, . . .) Toplas'15, CSF'21, ...

Mechanise Cryptographic Proofs

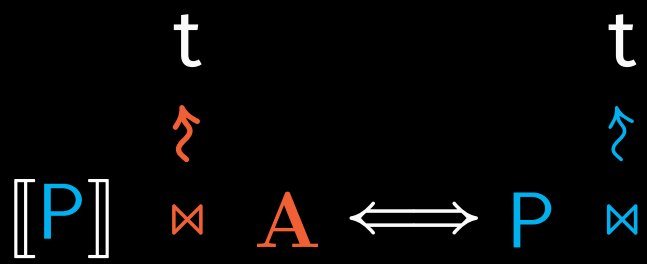
CSF'24 + wip

Robust Hyperproperty Preservation

$$[\![\cdot]\!] \vdash \mathsf{RHP} \stackrel{\text{def}}{=} \forall \mathsf{P}, \mathbf{A}.\ \exists \mathsf{A}.\ \forall t.$$

$$\mathbf{A}\,[[\![\mathsf{P}]\!]] \rightsquigarrow t \iff \mathsf{A}\,[\mathsf{P}] \rightsquigarrow t$$

Robust Hyperproperty Preservation



$$[\![\cdot]\!] \vdash \mathsf{RHP} \stackrel{\text{def}}{=} \forall \mathsf{P}, \mathbf{A}.\ \exists \mathsf{A}.\ \forall t.$$

$$\mathbf{A}\,[[\![\mathsf{P}]\!]] \rightsquigarrow t \iff \mathsf{A}\,[\mathsf{P}] \rightsquigarrow t$$

Universal Composability: UC

- gold standard for proving security of crypto protocols under concurrent composition

Universal Composability: UC

- gold standard for proving security of crypto protocols under concurrent composition
- overcome main drawback in protocol vulnerabilities: composition

Universal Composability: UC

- gold standard for proving security of crypto protocols under concurrent composition
- overcome main drawback in protocol vulnerabilities: composition
- many flavours: UC, SaUCy, iUC, ...

Canetti '01, Liao *et al.* PLDI'19, Camenisch *et al.* Asiacrypt'19

Universal Composability: UC

- gold standard for proving security of crypto protocols under concurrent composition
- overcome main drawback in protocol vulnerabilities: composition
- many flavours: UC, SaUCy, iUC, ...

Canetti '01, Liao *et al.* PLDI'19, Camenisch *et al.* Asiacrypt'19

This talk: generic flavour, geared towards the newer theories

UC Base Notions: ITMs

Canetti and Fischlin Crypto'01

- protocols Π (using concrete crypto)

commitment for $b \in \{0,1\}$ *with SID* sid*:*

compute $G_{pk_b}(r)$ for random $r \in \{0,1\}^n$
set $y = G_{pk_b}(r)$ for $b = 0$, or $y = G_{pk_b}(r) \oplus \sigma$ for $b = 1$
send $(\texttt{Com}, sid, y)$ to the receiver

Upon receiving $(\texttt{Com}, sid, y)$ from P_i, P_j outputs $(\texttt{Receipt}, sid, cid, P_i, P_j)$

UC Base Notions: ITMs

Canetti and Fischlin Crypto'01

- protocols Π (using concrete crypto)

commitment for $b \in \{0,1\}$ *with SID* sid*:*

compute $G_{pk_b}(r)$ for random $r \in \{0,1\}^n$
set $y = G_{pk_b}(r)$ for $b = 0$, or $y = G_{pk_b}(r) \oplus \sigma$ for $b = 1$
send $(\texttt{Com}, sid, y)$ to the receiver

Upon receiving $(\texttt{Com}, sid, y)$ from P_i, P_j outputs $(\texttt{Receipt}, sid, cid, P_i, P_j)$

- functionalities F (using abstract notions)

1. Upon receiving a value $(\texttt{Commit}, sid, P_i, P_j, b)$ from P_i, where $b \in \{0,1\}$, record the value b and send the message $(\texttt{Receipt}, sid, P_i, P_j)$ to P_j and $\mathcal{S}$. Ignore any subsequent Commit messages.

UC Base Notions: ITMs

Canetti and Fischlin Crypto'01

- protocols Π (using concrete crypto)

commitment for $b \in \{0,1\}$ *with SID* sid*:*

compute $G_{pk_b}(r)$ for random $r \in \{0,1\}^n$
set $y = G_{pk_b}(r)$ for $b = 0$, or $y = G_{pk_b}(r) \oplus \sigma$ for $b = 1$
send $(\texttt{Com}, sid, y)$ to the receiver

Upon receiving $(\texttt{Com}, sid, y)$ from P_i, P_j outputs $(\texttt{Receipt}, sid, cid, P_i, P_j)$

- functionalities F (using abstract notions)

1. Upon receiving a value $(\texttt{Commit}, sid, P_i, P_j, b)$ from P_i, where $b \in \{0,1\}$, record the value b and send the message $(\texttt{Receipt}, sid, P_i, P_j)$ to P_j and $\mathcal{S}$. Ignore any subsequent Commit messages.

- attackers A & S (corrupting parties etc.)

UC Base Notions: ITMs

Canetti and Fischlin Crypto'01

- protocols Π (using concrete crypto)

commitment for $b \in \{0,1\}$ *with SID* sid*:*

compute $G_{pk_b}(r)$ for random $r \in \{0,1\}^n$
set $y = G_{pk_b}(r)$ for $b = 0$, or $y = G_{pk_b}(r) \oplus \sigma$ for $b = 1$
send $(\texttt{Com}, sid, y)$ to the receiver

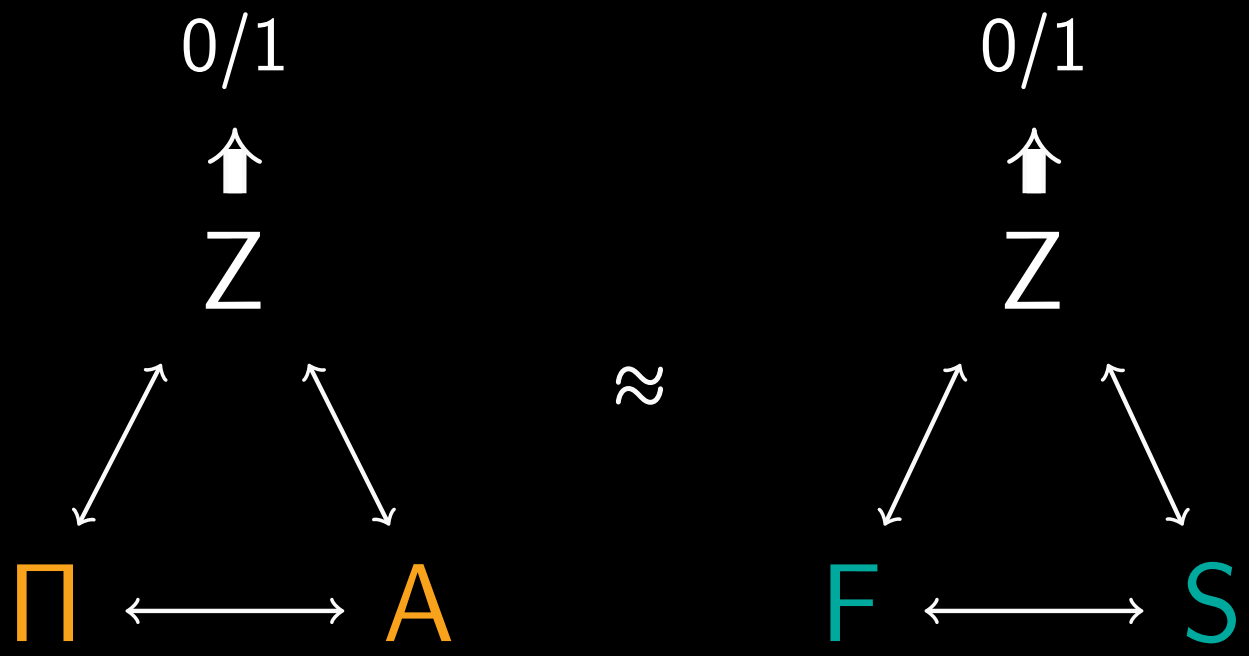
Upon receiving $(\texttt{Com}, sid, y)$ from P_i, P_j outputs $(\texttt{Receipt}, sid, cid, P_i, P_j)$

- functionalities F (using abstract notions)

1. Upon receiving a value $(\texttt{Commit}, sid, P_i, P_j, b)$ from P_i, where $b \in \{0,1\}$, record the value b and send the message $(\texttt{Receipt}, sid, P_i, P_j)$ to P_j and $\mathcal{S}$. Ignore any subsequent Commit messages.

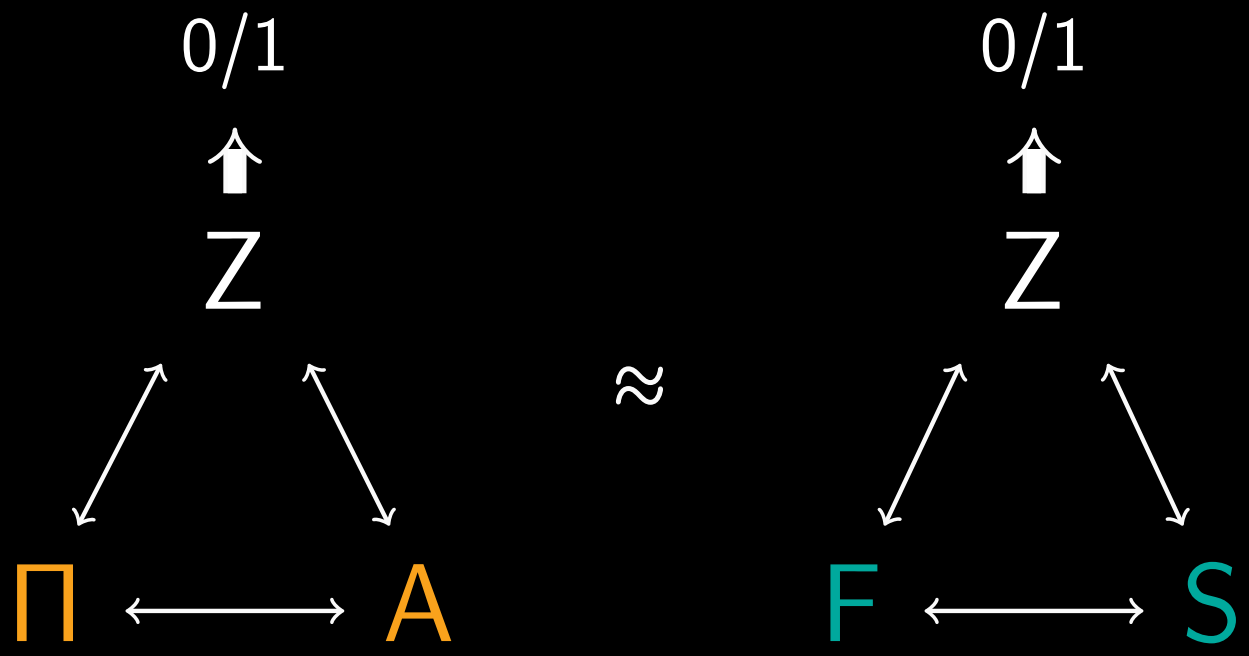
- attackers A & S (corrupting parties etc.)
- environments Z (objective witness)

UC (Semi-formally)



↔ represent communication channels

UC (Semi-formally)

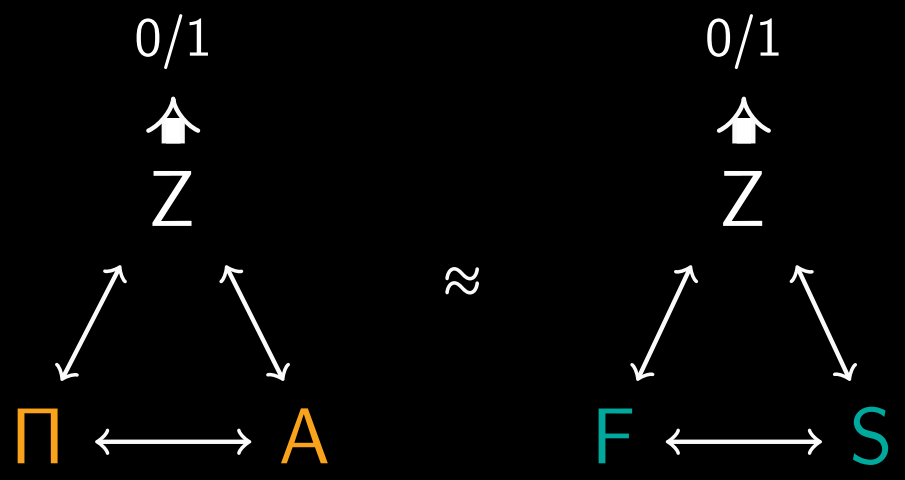


$\leftrightarrow$ represent communication channels

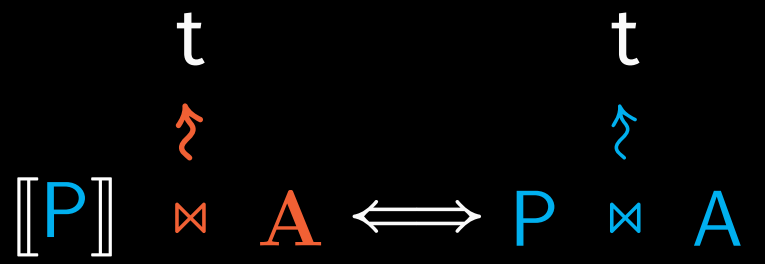
$$\Pi \vdash_{UC} \mathsf{F} \stackrel{\text{def}}{=} \forall \text{poly } \mathsf{A}, \exists \mathsf{S}, \forall \mathsf{Z}.$$
$$\text{Exec}[\mathsf{Z}, \mathsf{A}, \Pi] \approx \text{Exec}[\mathsf{Z}, \mathsf{S}, \mathsf{F}]$$

A Closer Look

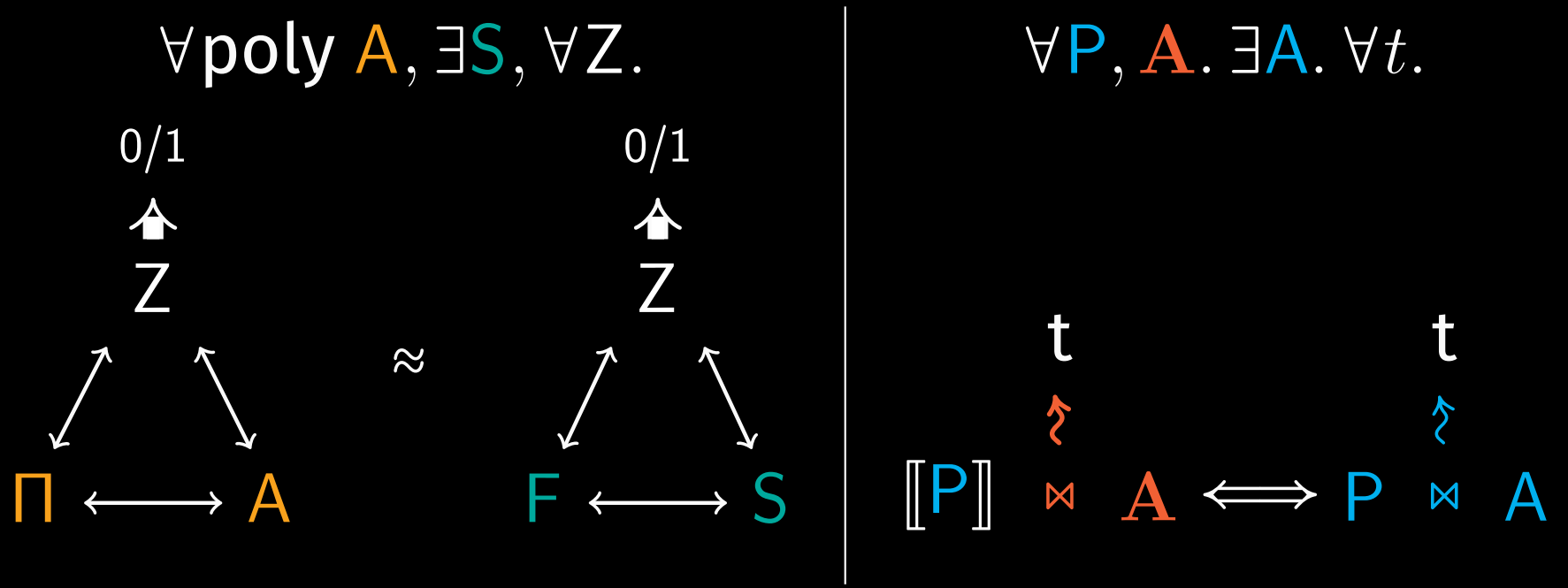
$\forall \text{poly}\ A, \exists S, \forall Z.$



$\forall P, \mathbf{A}. \exists A. \forall t.$



A Closer Look



Isabelle'd both perfect and computational UC

Analogy

UC			SC
protocol	Π	$[\![\mathsf{P}]\!]$	compiled program
concrete attacker	A	$\mathbf{A}$	target context
ideal functionality	F	P	source program
simulator	S	A	source context
environment, output	Z, 0/1	t, $\rightsquigarrow$	trace, semantics
communication	$\leftrightarrow$	[]	linking
probabilistic equiv.	$\approx$	$\Longleftrightarrow$	trace equality

Analogy

UC		SC	
protocol	Π	$[\![\mathrm{P}]\!]$	compiled program
concrete attacker	A	$\mathbf{A}$	target context
ideal functionality	F	P	source program
simulator	S	A	source context
environment, output	Z, 0/1	t, $\rightsquigarrow$	trace, semantics
communication	$\leftrightarrow$	$[]$	linking
probabilistic equiv.	$\approx$	$\Longleftrightarrow$	trace equality
human translation $\Pi \to \mathrm{F}$		$[\![\cdot]\!]\colon \mathrm{P} \to \mathbf{P}$ compiler	
general composition result		…	

Analogy Results

- transfer UC results from ITMs to any $\mathtt{S}$/$\mathtt{T}$

Analogy Results

- transfer UC results from ITMs to any $\mathtt{S}$/$\mathtt{T}$
- mechanise UC results as RHP results

Analogy Results

- transfer UC results from ITMs to any $\mathtt{S}/\mathtt{T}$
- mechanise UC results as RHP results known in computer-aided crypto Haagh *et al.* CSF'18

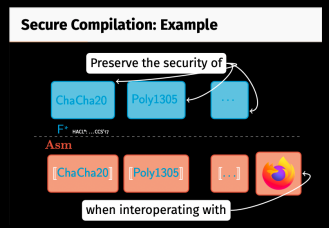
Analogy Results

- transfer UC results from ITMs to any $\mathtt{S}/\mathtt{T}$
- mechanise UC results as RHP results known in computer-aided crypto Haagh *et al.* CSF'18
- Mechanised UC for 1-Bit Commitment in `Deepsec` submission
- Mechanised UC for 1/2 Wireguard in `Cryptoverif` CSF'24

Conclusion

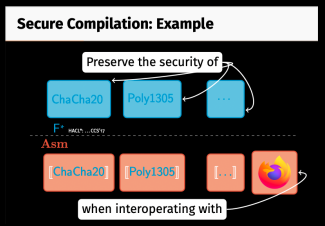
Conclusion

- secure compilation threat model

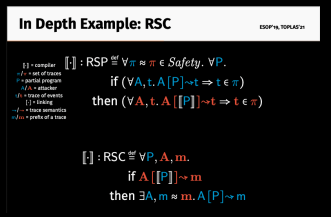


Conclusion

- secure compilation threat model

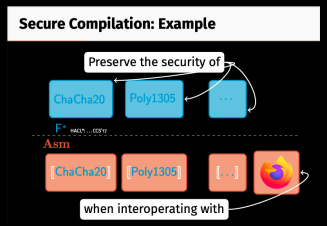


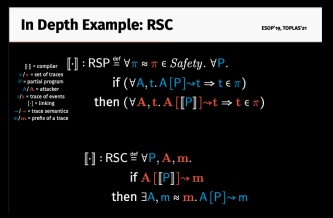
- formal foundations: RSC, RHP

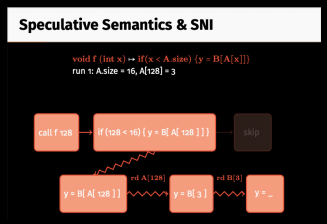


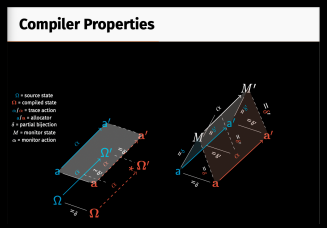
Conclusion

- secure compilation threat model
- formal foundations: RSC, RHP
- robust compilation use-cases (MS, CT, SNI)



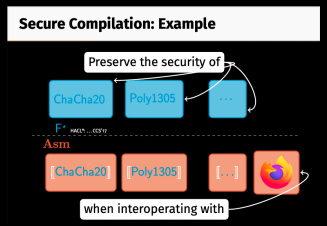


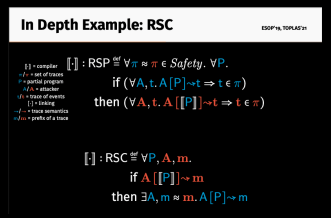


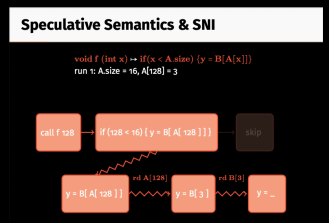


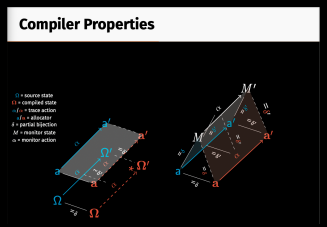
Conclusion

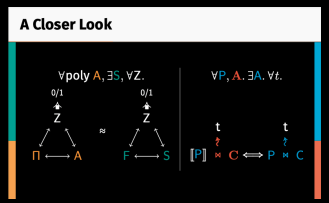
- secure compilation threat model
- formal foundations: RSC, RHP
- robust compilation use-cases (MS, CT, SNI)
- connection with UC











Future

- More foundations questions?

Future

- More foundations questions?
- SC for emerging security archs?

Future

- More foundations questions?
- SC for emerging security archs?
- SC for more properties?

Future

- More foundations questions?
- SC for emerging security archs?
- SC for more properties?
- SC for different languages?

Future

- More foundations questions?
- SC for emerging security archs?
- SC for more properties?
- SC for different languages?
- Other UC-like connections?

Future

- More foundations questions?
- SC for emerging security archs?
- SC for more properties?
- SC for different languages?
- Other UC-like connections?
- More mechanised UC protocols?

Future

- More foundations questions?
- S
- S
- SC for different languages?
- Other UC-like connections?
- More mechanised UC protocols?

Come to PRISC'25, co-located with POPL'25.

Questions?