# CSC Report – Foundations of Secure Compilation

Marco Patrignani[1,2]

23$^{rd}$ June 2021

1 CISPA HELMHOLTZ CENTER FOR INFORMATION SECURITY 2 Stanford University

# Talk Outline

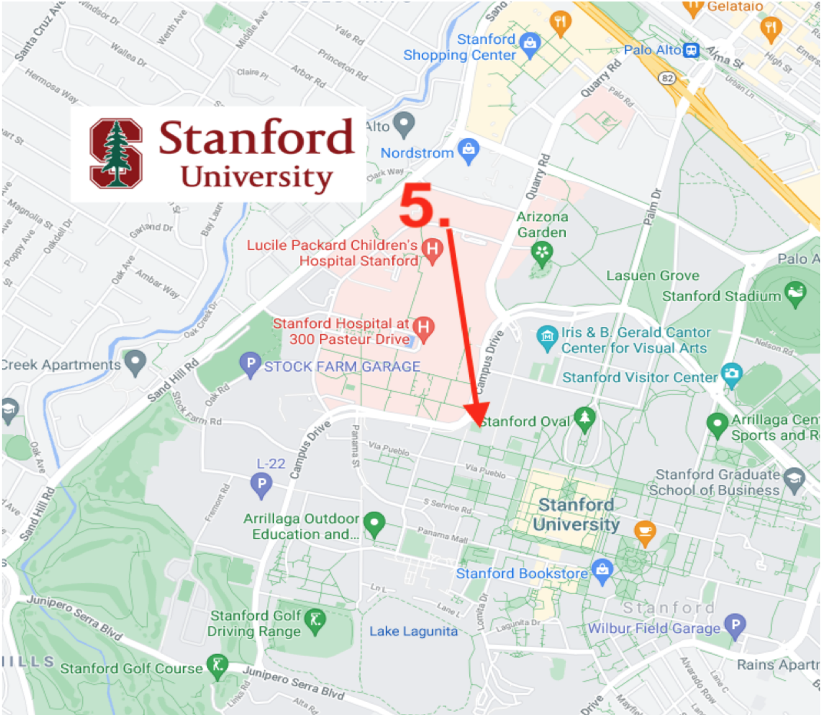
My Stanford Experience

Foundations of Secure Compilation

Future Outlook

# My Stanford Experience

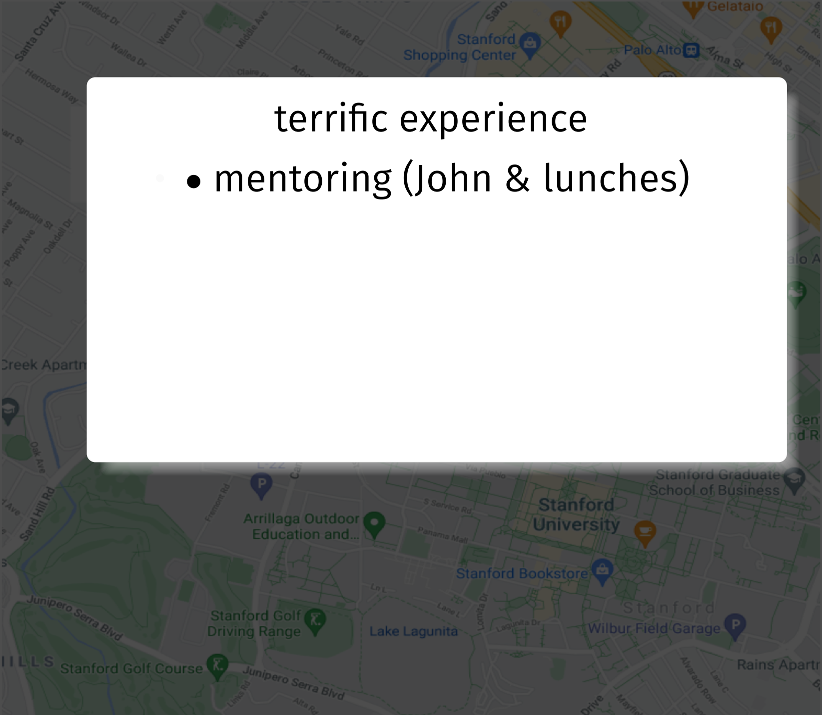
MAX PLANCK INSTITUTE
FOR SOFTWARE SYSTEMS
4- 6
CISPA-Stanford Center
FOR CYBERSECURITY
KU LEUVEN
3
ALMA MATER STUDIORUM
A.D. 1088
ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA
2
1
Bruges
Antwerp
Düsseldorf
Leipzig
Dresden
Görlitz
Brussels
Cologne
Germany
Chemnitz
Gera
Liberec
Lille
Amiens
Saint-Quentin
Luxembourg
Wiesbade
Prague
Czech Rep
Nuremberg
Reims
Karlsruhe
Strasbourg
Munich
Linz
Freiburg
Konstanz
Basel
Zurich
Dijon
Besançon
Bourges
Chalon-sur-Saône
Slovenia
Trieste
Venice
Rijeka
Padua
Croatia
Milan
Clermont-Ferrand
Lyon
Grenoble
Turin
Valence
Asti
Parma
Bologna
Pula
Genoa
Ravenna
Zadar
Avignon



Stanford
University
5.
Lucile Packard Children's Hospital Stanford
Stanford Hospital at 300 Pasteur Drive
STOCK FARM GARAGE
Stanford Shopping Center
Nordstrom
Palo Alto
Arizona Garden
Lasuen Grove
Stanford Stadium
Iris & B. Gerald Cantor Center for Visual Arts
Stanford Visitor Center
Stanford Oval
Stanford Graduate School of Business
Stanford University
Stanford Bookstore
Wilbur Field Garage
Lake Lagunita
Stanford Golf Driving Range
Stanford Golf Course
Arrillaga Outdoor Education and...
L-22
Gelataio
Sand Hill Rd
Campus Drive
Quarry Rd
Palm Dr
Junipero Serra Blvd
Alma St

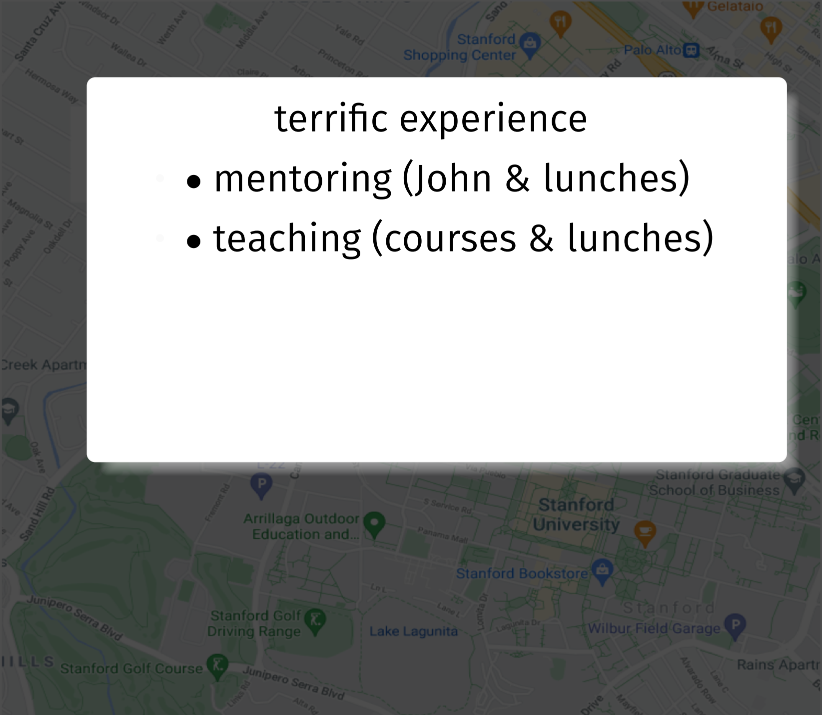
terrific experience

- mentoring (John & lunches)



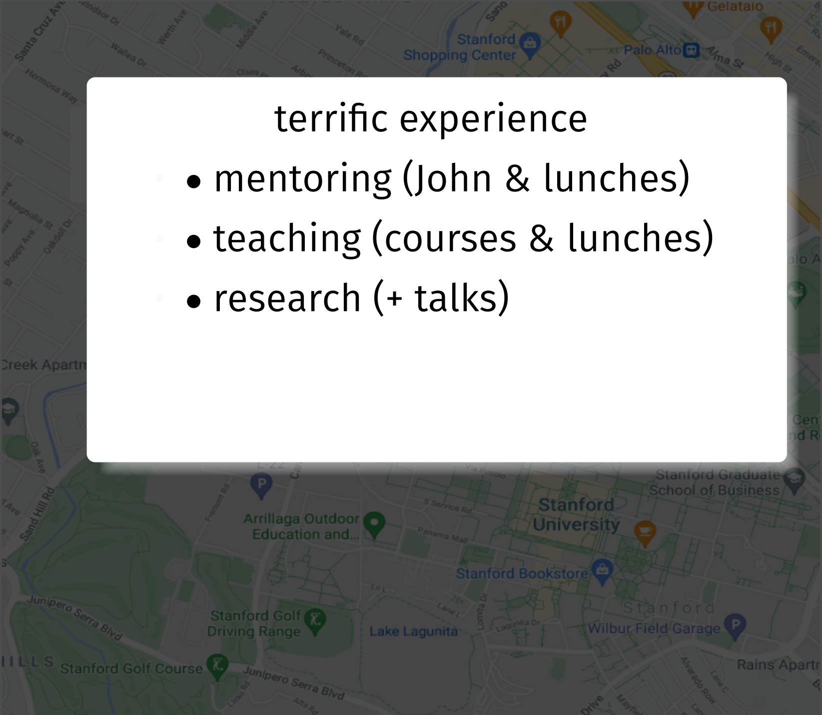
terrific experience

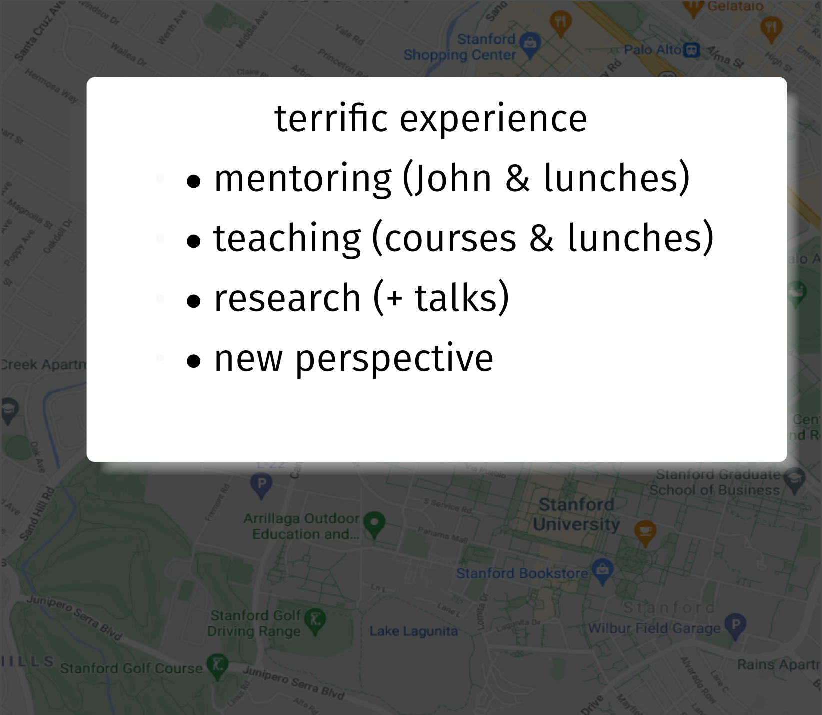
- mentoring (John & lunches)
- teaching (courses & lunches)



terrific experience

- mentoring (John & lunches)
- teaching (courses & lunches)
- research (+ talks)



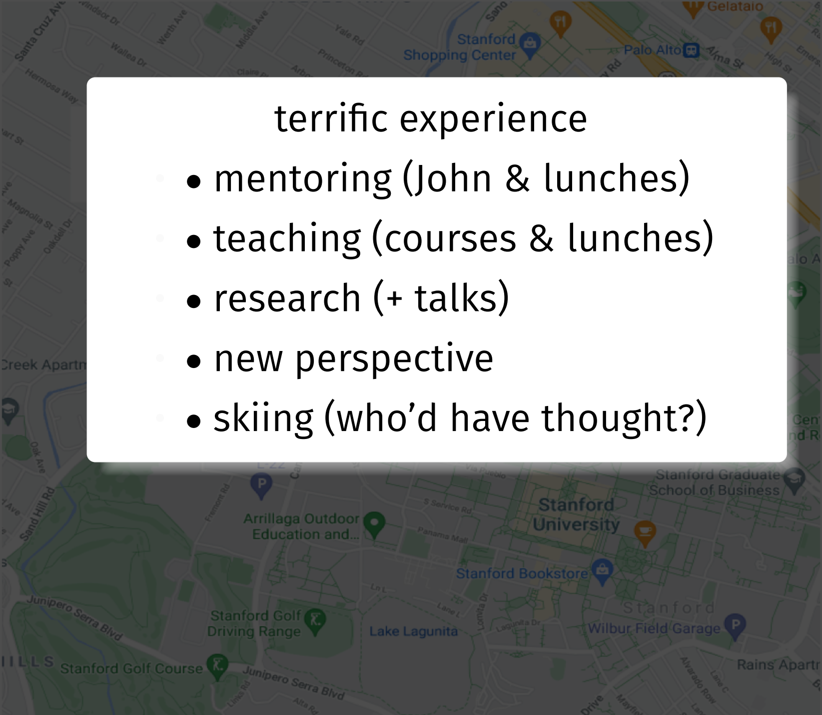



terrific experience

- mentoring (John & lunches)
- teaching (courses & lunches)
- research (+ talks)
- new perspective

terrific experience

- mentoring (John & lunches)
- teaching (courses & lunches)
- research (+ talks)
- new perspective
- skiing (who’d have thought?)



# Foundations of Secure Compilation

# Outline

1. Motivation behind SC
2. history of SC
3. our contributions to the foundations of SC
4. current and future applications

# Special Thanks to:

(wrt the contents of this talk)


Carmine Abate


Amal Ahmed


Roberto Blanco


Stefan Ciobaca


Dave Clarke


Dominique Devriese


Akram El-Korashy


Deepak Garg


Marco Guarnieri


Catalin Hritcu


Robert Künnemann


Frank Piessens


Eric Tanter


Jeremy Thibault

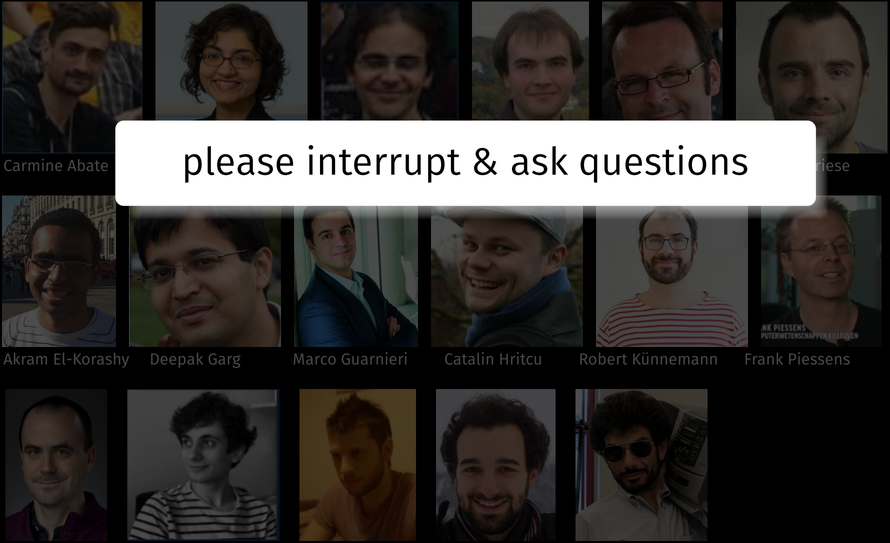

Stelios Tsampas


Marco Vassena


Riad Wahby

# Special Thanks to:

**(wrt the contents of this talk)**



Carmine Abate

please interrupt & ask questions

Akram El-Korashy · Deepak Garg · Marco Guarnieri · Catalin Hritcu · Robert Künnemann · Frank Piessens

Eric Tanter · Jeremy Thibault · Stelios Tsampas · Marco Vassena · Riad Wahby

# Programming Languages: Pros and Problems

Good PLs (, , , , …) provide:

- helpful abstractions to write secure code

# Programming Languages: Pros and Problems

Good PLs (, , , , ...) provide:

- helpful abstractions to write secure code

but

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

# Programming Languages: Pros and Problems

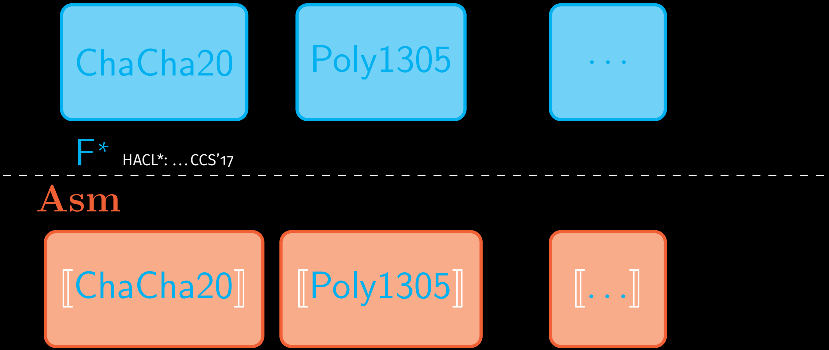
Good PLs (, , , , …) provide:

- helpful abstractions to write secure code

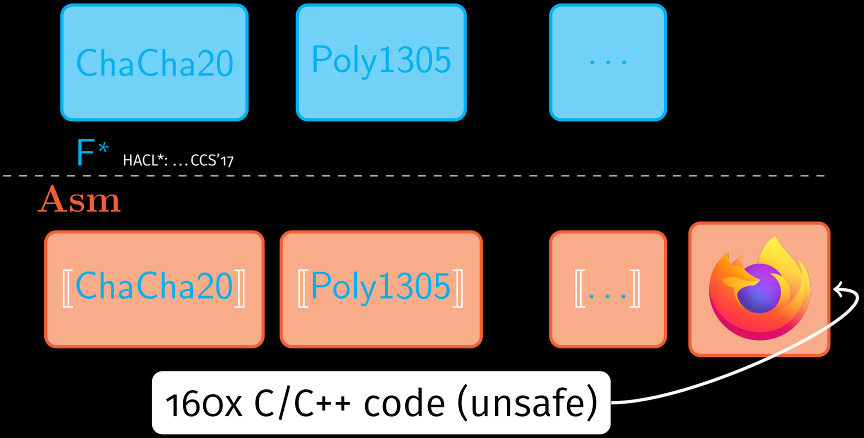
but

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

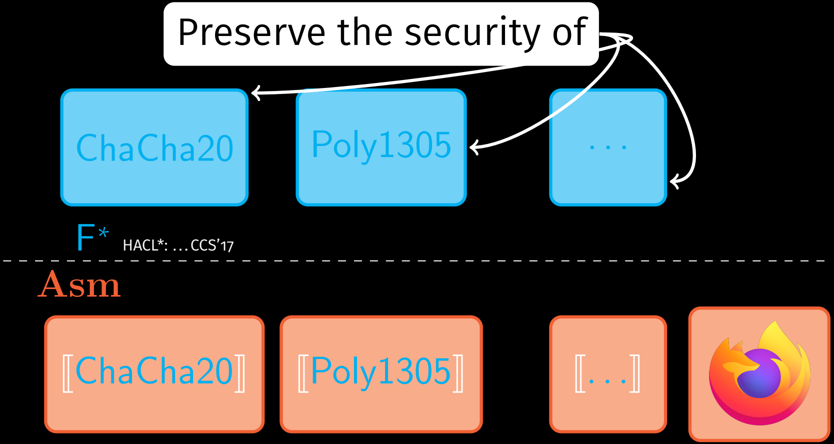
# Secure Compilation: Example



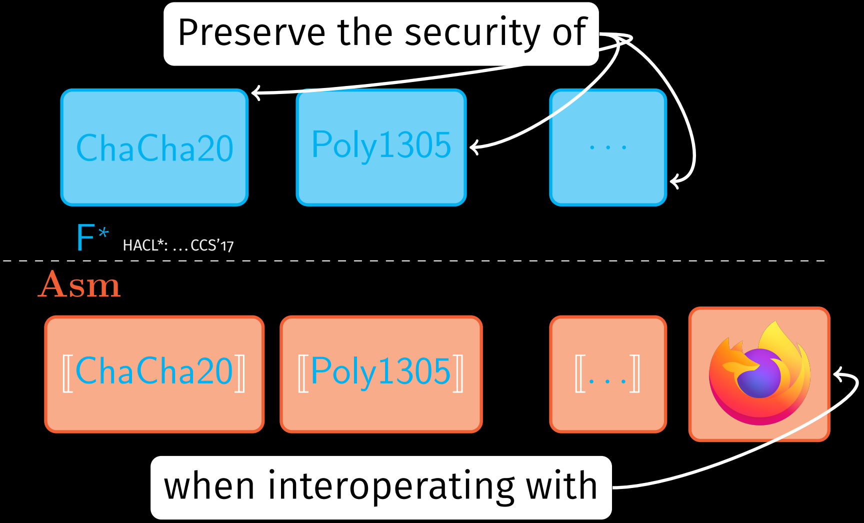
# Secure Compilation: Example



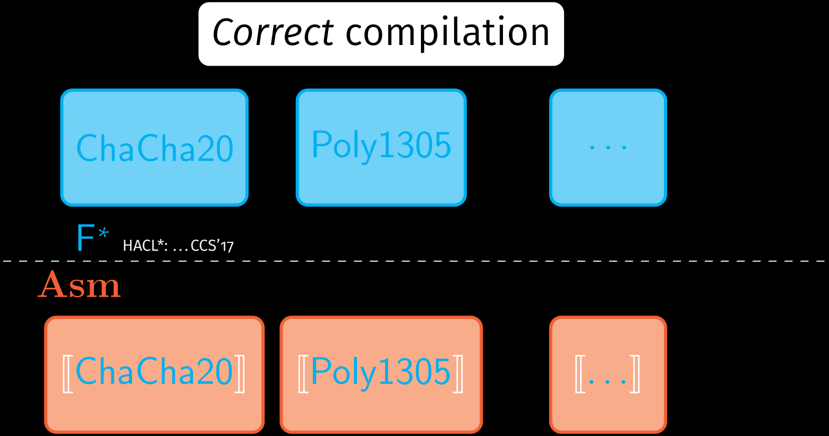
# Secure Compilation: Example



# Secure Compilation: Example



# Secure Compilation: Example



# Secure Compilation: Example

**Secure** compilation

ChaCha20

Poly1305

. . .

F* HACL*: . . . CCS'17

Asm

⟦ChaCha20⟧

⟦Poly1305⟧

⟦. . .⟧

# Secure Compilation: Example

Enable source-level security reasoning

ChaCha20 | Poly1305 | ...

F* HACL*: ...CCS'17

Asm

⟦ChaCha20⟧ | ⟦Poly1305⟧ | ⟦...⟧

What does it mean for a compiler to be secure?

# What does it mean for a compiler to be secure?

Known for type systems, CC but not for SC

# 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

**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 $Q$,*

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

# Once Upon a Time in Process Algebra

they needed a definition 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 $Q$,*

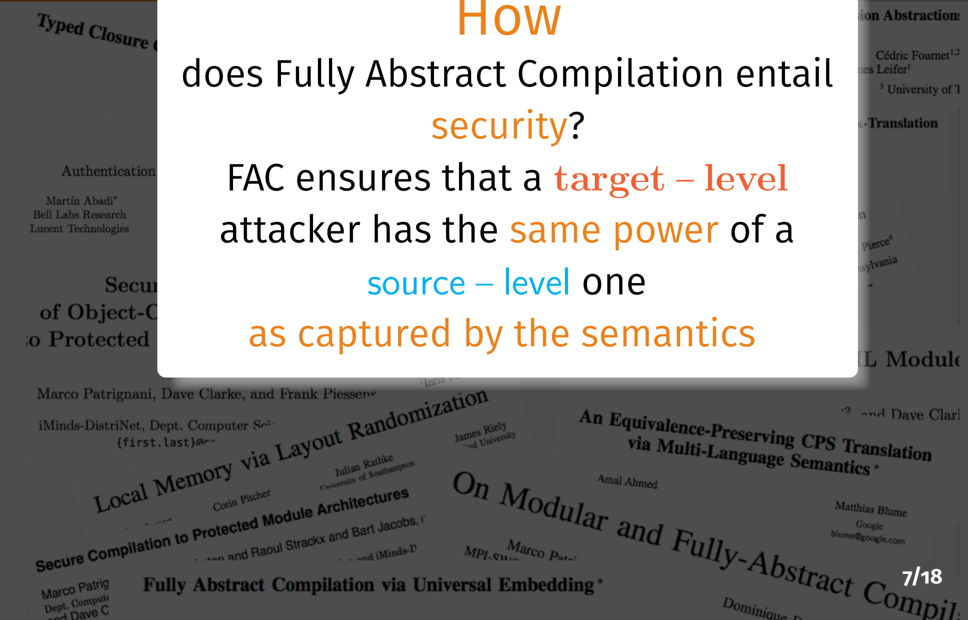
$$P \approx Q \quad \textit{if and only if} \quad \mathcal{E}nv\,[\,[\![P]\!]\,] \approx \mathcal{E}nv\,[\,[\![Q]\!]\,]$$

# Fully Abstract Compilation Influence

# Fully Abstract Compilation Influence

**How** does Fully Abstract Compilation entail **security**?

# Fully Abstract Compilation Influence




## How

does Fully Abstract Compilation entail **security**?

FAC ensures that a $\mathbf{target-level}$ attacker has the same power of a source – level one

as captured by the semantics

# Fully Abstract Compilation: Definition

$$\mathsf{P}_1 \quad \simeq_{ctx} \quad \mathsf{P}_2$$

$$\Updownarrow$$

$$[\![\mathsf{P}_1]\!] \simeq_{ctx} [\![\mathsf{P}_2]\!]$$

# Fully Abstract Compilation: Definition

$$\mathsf{P}_1 \quad \simeq_{ctx} \quad \mathsf{P}_2$$

$$\Uparrow$$

$$[\![\mathsf{P}_1]\!] \ \simeq_{ctx} \ [\![\mathsf{P}_2]\!]$$

# Fully Abstract Compilation: Definition

$$P_1 \simeq_{ctx} P_2$$

$$\Downarrow$$

$$[\![P_1]\!] \simeq_{ctx} [\![P_2]\!]$$

# Fully Abstract Compilation: Definition

$$\mathsf{P}_1 \quad \simeq_{ctx} \quad \mathsf{P}_2$$

$$\Downarrow$$

$$\forall \mathbf{A}.\, \mathbf{A}\left[\llbracket \mathsf{P}_1 \rrbracket\right]\Downarrow \iff \mathbf{A}\left[\llbracket \mathsf{P}_2 \rrbracket\right]\Downarrow$$

# Are there Alternatives to FAC?

- FAC is not precise about security

# Are there Alternatives to FAC?

- FAC is not precise about security
- this affects efficiency and proof complexity

# Are there Alternatives to FAC?

- FAC is not precise about security
- this affects efficiency and proof complexity
- in certain cases we want easier/more efficient alternatives

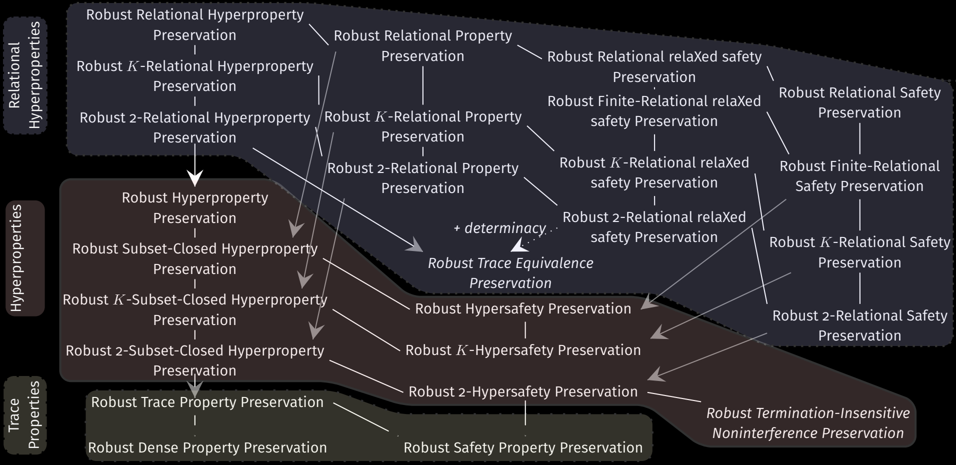
# Are there Alternatives to FAC?

- FAC is not precise about security
- this affects efficiency and proof complexity
- in certain cases we want easier/more efficient alternatives

preserve classes of security (hyper)properties

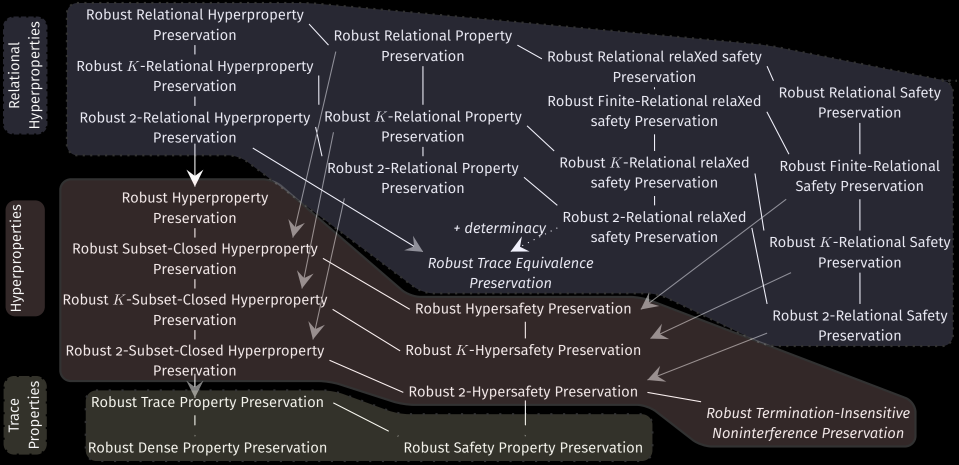
# Robust Compilation Criteria

CSF'19, ESOP'20, ACM Toplas'21



# Robust Compilation Criteria

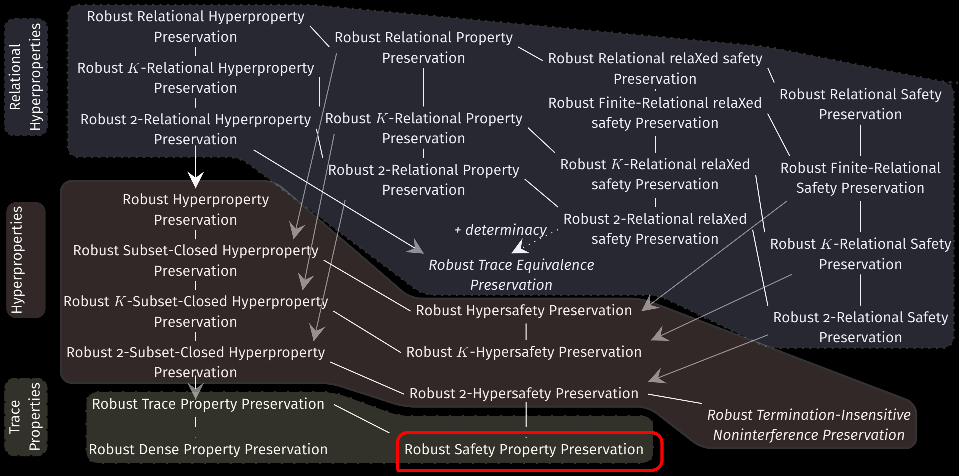
CSF'19, ESOP'20, ACM Toplas'21




Tradeoffs for code efficiency, security guarantees, proof complexity

# Robust Compilation Criteria

CSF'19, ESOP'20, ACM Toplas'21




Tradeoffs for code efficiency, security guarantees, proof complexity

# Robust Criteria: Intuition

Each point has two equivalent criteria:

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

# Robust Criteria: Intuition

Each point has two equivalent criteria:

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

# Robust Criteria: Intuition

Each point has two equivalent criteria:

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

# Robust Criteria: Intuition

Each point has two equivalent criteria:

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

# Robust Criteria: Intuition

Each point has two equivalent criteria:

- $\mathbf{Property-ful}$:
  - \+ clearly tells what class it preserves
  - \- harder to prove
- $\mathbf{Property-free}$:
  - \+ easier to prove
  - \- unclear what security classes are preserved
  - = akin to some crypto statements (UC)

# In Depth Example: RSC



$[\![\cdot]\!]$ = compiler

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

# In Depth Example: RSC

ESOP'19, ACM Toplas'21

$[\![\cdot]\!]$ = compiler

$\pi$ / $\pi$ = set of traces

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

# In Depth Example: RSC

ESOP'19, ACM Toplas'21

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

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

# In Depth Example: RSC



$[\![\cdot]\!]$ = compiler
$\pi$/$\pi$ = set of traces
P = partial program
A/**A** = attacker
t/**t** = trace of events

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

$$\text{if } (\forall \mathsf{A}, \mathsf{t}.$$

# In Depth Example: RSC

ESOP'19, ACM Toplas'21

$[\![\cdot]\!]$ = compiler
$\pi / \pi$ = set of traces
P = partial program
A / **A** = attacker
t / **t** = trace of events
$[\cdot]$ = linking
$\rightsquigarrow / \rightsquigarrow$ = trace semantics

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

$$\text{if } (\forall \mathsf{A}, \mathsf{t}.\ \mathsf{A}\left[\mathsf{P}\right] \rightsquigarrow \mathsf{t}$$

# In Depth Example: RSC

ESOP'19, ACM Toplas'21

$[\![\cdot]\!]$ = compiler
$\pi / \pi$ = set of traces
P = partial program
A / **A** = attacker
t / **t** = trace of events
$[\cdot]$ = linking
$\rightsquigarrow / \rightsquigarrow$ = trace semantics

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

$$\text{if } (\forall \mathsf{A}, \mathsf{t}.\ \mathsf{A}[\mathsf{P}] \rightsquigarrow \mathsf{t} \Rightarrow \mathsf{t} \in \pi)$$

# In Depth Example: RSC

ESOP'19, ACM Toplas'21

$[\![\cdot]\!]$ = compiler
$\pi$/$\pi$ = set of traces
P = partial program
A/**A** = attacker
t/**t** = trace of events
$[\cdot]$ = linking
$\rightsquigarrow$/$\rightsquigarrow$ = trace semantics

$$[\![\cdot]\!] : \mathsf{RSP} \overset{\mathsf{def}}{=} \forall \pi \approx \pi \in \mathit{Safety}.\ \forall \mathsf{P}.$$

$$\text{if } (\forall \mathsf{A}, \mathsf{t}.\ \mathsf{A}[\mathsf{P}] \rightsquigarrow \mathsf{t} \Rightarrow \mathsf{t} \in \pi)$$

$$\text{then } (\forall \mathbf{A}, \mathbf{t}.$$

# In Depth Example: RSC



$[\![\cdot]\!]$ = compiler
$\pi / \pi$ = set of traces
P = partial program
A / **A** = attacker
t / **t** = trace of events
$[\cdot]$ = linking
$\rightsquigarrow / \rightsquigarrow$ = trace semantics

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

$$\text{if } (\forall \mathsf{A}, \mathsf{t}.\ \mathsf{A}[\mathsf{P}] \rightsquigarrow \mathsf{t} \Rightarrow \mathsf{t} \in \pi)$$

$$\text{then } (\forall \mathbf{A}, \mathbf{t}.\ \mathbf{A}[[\![\mathsf{P}]\!]] \rightsquigarrow \mathbf{t} \Rightarrow$$

# In Depth Example: RSC

ESOP'19, ACM Toplas'21

$[\![\cdot]\!]$ = compiler
$\pi$ / $\pi$ = set of traces
P = partial program
A / **A** = attacker
t / **t** = trace of events
$[\cdot]$ = linking
$\rightsquigarrow$ / $\rightsquigarrow$ = trace semantics

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

$$\text{if } (\forall \mathsf{A}, \mathsf{t}.\ \mathsf{A}[\mathsf{P}] \rightsquigarrow \mathsf{t} \Rightarrow \mathsf{t} \in \pi)$$

$$\text{then } (\forall \mathbf{A}, \mathbf{t}.\ \mathbf{A}[[\![\mathsf{P}]\!]] \rightsquigarrow \mathbf{t} \Rightarrow \mathbf{t} \in \pi)$$

# In Depth Example: RSC



$[\![\cdot]\!]$ = compiler
$\pi/\pi$ = set of traces
P = partial program
A/**A** = attacker
t/**t** = trace of events
$[\cdot]$ = linking
$\rightsquigarrow/\rightsquigarrow$ = trace semantics

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

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

# In Depth Example: RSC

ESOP'19, ACM Toplas'21

$[\![\cdot]\!]$ = compiler
$\pi / \pi$ = 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

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

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

# In Depth Example: RSC

ESOP'19, ACM Toplas'21

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

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

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

# In Depth Example: RSC

ESOP'19, ACM Toplas'21

$[\![\cdot]\!]$ = compiler
$\pi$ / $\pi$ = 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

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

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

# In Depth Example: RSC

ESOP'19, ACM Toplas'21

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

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

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

# In Depth Example: RSC

ESOP'19, ACM Toplas'21

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

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

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

# Understanding RSC

RSP/RSC:

- adaptable to reason about complex features: concurrency, undefined behaviour

# Understanding RSC

RSP/RSC:

- adaptable to reason about complex features: concurrency, undefined behaviour

RSP:

- provable if source is robustly-safe

# Understanding RSC

RSP/RSC:

- adaptable to reason about complex features: concurrency, undefined behaviour

RSP:

- provable if source is robustly-safe

RSC:

- easiest backtranslation proof

# RSC - FAC

Both:

# RSC - FAC

Both:

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

# RSC - FAC

Both:

- robust ($\forall \mathbf{A}$)
- rely on program semantics

# RSC - FAC

Both:

- robust ($\forall \mathbf{A}$)
- rely on program semantics

FAC:

# RSC - FAC

Both:

- robust ($\forall \mathbf{A}$)
- rely on program semantics

FAC:

- yields a language result POPL'21

# RSC - FAC

Both:

- robust ($\forall \mathbf{A}$)
- rely on program semantics

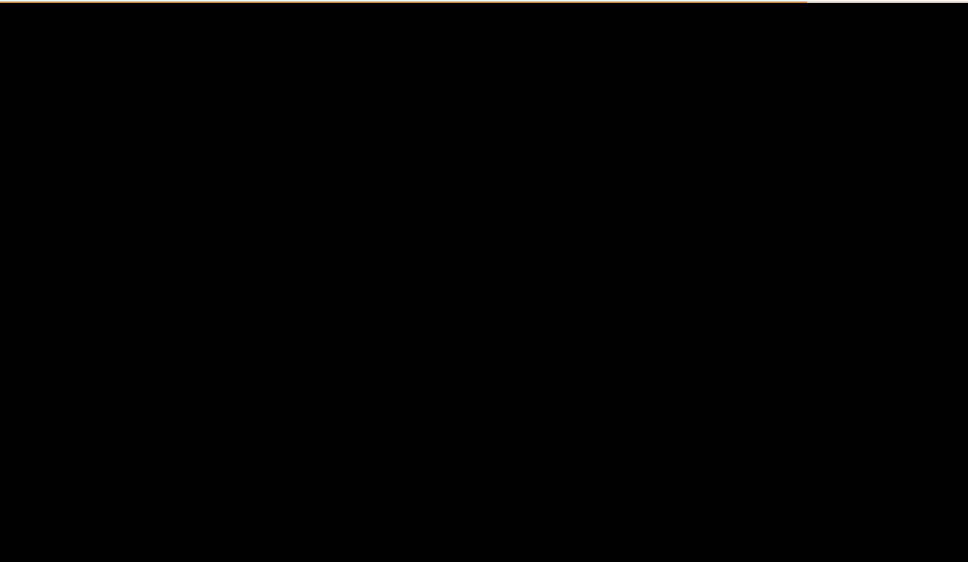
FAC:

- yields a language result POPL'21

RSC/RSP:

- extends the semantics ($\rightsquigarrow$) to focus on security

# Is There More?



# Is There More?

Some still unknown foundations include:

# Is There More?

Some still unknown foundations include:

- optimisation

# Is There More?

Some still unknown foundations include:

- optimisation
- composition (multipass & linking)

# Robust(ly Safe) Compilation at Work

Instantiate RSC to specific properties

- absence of speculation leaks CCS'21

# Robust(ly Safe) Compilation at Work

Instantiate RSC to specific properties

- absence of speculation leaks CCS'21
- memory safety preservation (spatial, temporal)

# Robust(ly Safe) Compilation at Work

Instantiate RSC to specific properties

- absence of speculation leaks CCS'21
- memory safety preservation (spatial, temporal)
- constant-time preservation

# Robust(ly Safe) Compilation at Work

Instantiate RSC to specific properties

- absence of speculation leaks CCS'21
- memory safety preservation (spatial, temporal)
- constant-time preservation
- ...

# Future Outlook

# More Secure Compilation

# More Secure Compilation

- secure compilation for Spectre V2+
  (w. Imdea, Cispa)

# More Secure Compilation

- secure compilation for Spectre V2+ (w. Imdea, Cispa)
- secure compilation to webassembly (w. UCSD, Harvard, Cispa)

# More Secure Compilation

- secure compilation for Spectre V2+
  (w. Imdea, Cispa)
- secure compilation to webassembly
  (w. UCSD, Harvard, Cispa)
- secure compilation & universal composability
  (w. Stanford, Cispa)

# More Secure Compilation

- secure compilation for Spectre V2+
  (w. Imdea, Cispa)
- secure compilation to webassembly
  (w. UCSD, Harvard, Cispa)
- secure compilation & universal composability
  (w. Stanford, Cispa)
- secure compilation for linear languages
  (w. Novi / FB)

# More Secure Compilation

- secure compilation for Spectre V2+ (w. Imdea, Cispa)
- secure compilation to webassembly (w. UCSD, Harvard, Cispa)
- secure compilation & universal composability (w. Stanford, Cispa)
- secure compilation for linear languages (w. Novi / FB)
- …(some PL too, w. Stanford, KU Leuven)

# Questions?