CSC Report – Foundations of Secure Compilation

Marco Patrignani[1,2]

23rd June 2021

1 CISPA HELMHOLTZ CENTER FOR INFORMATION SECURITY

2 Stanford University

Talk Outline

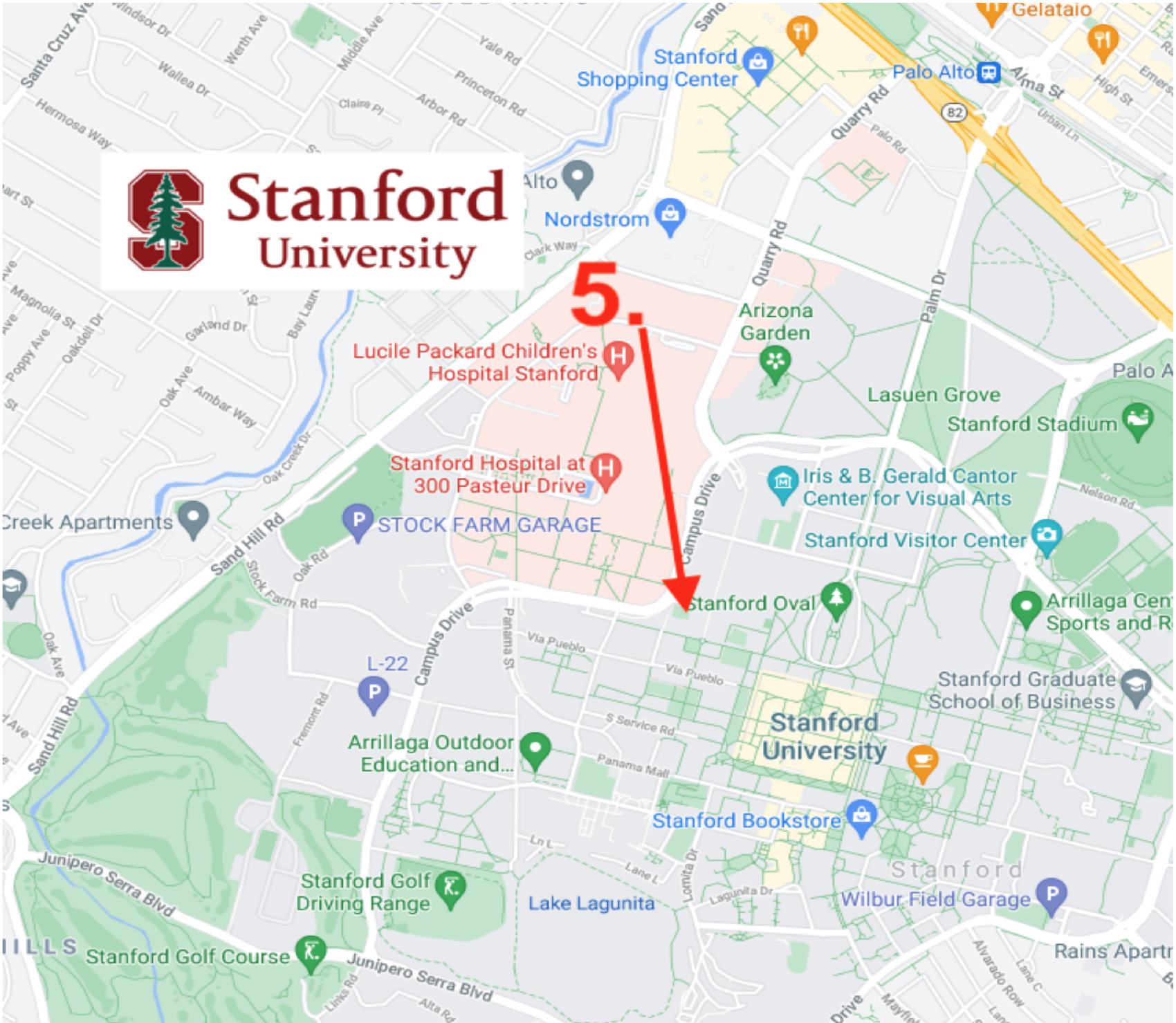
My Stanford Experience

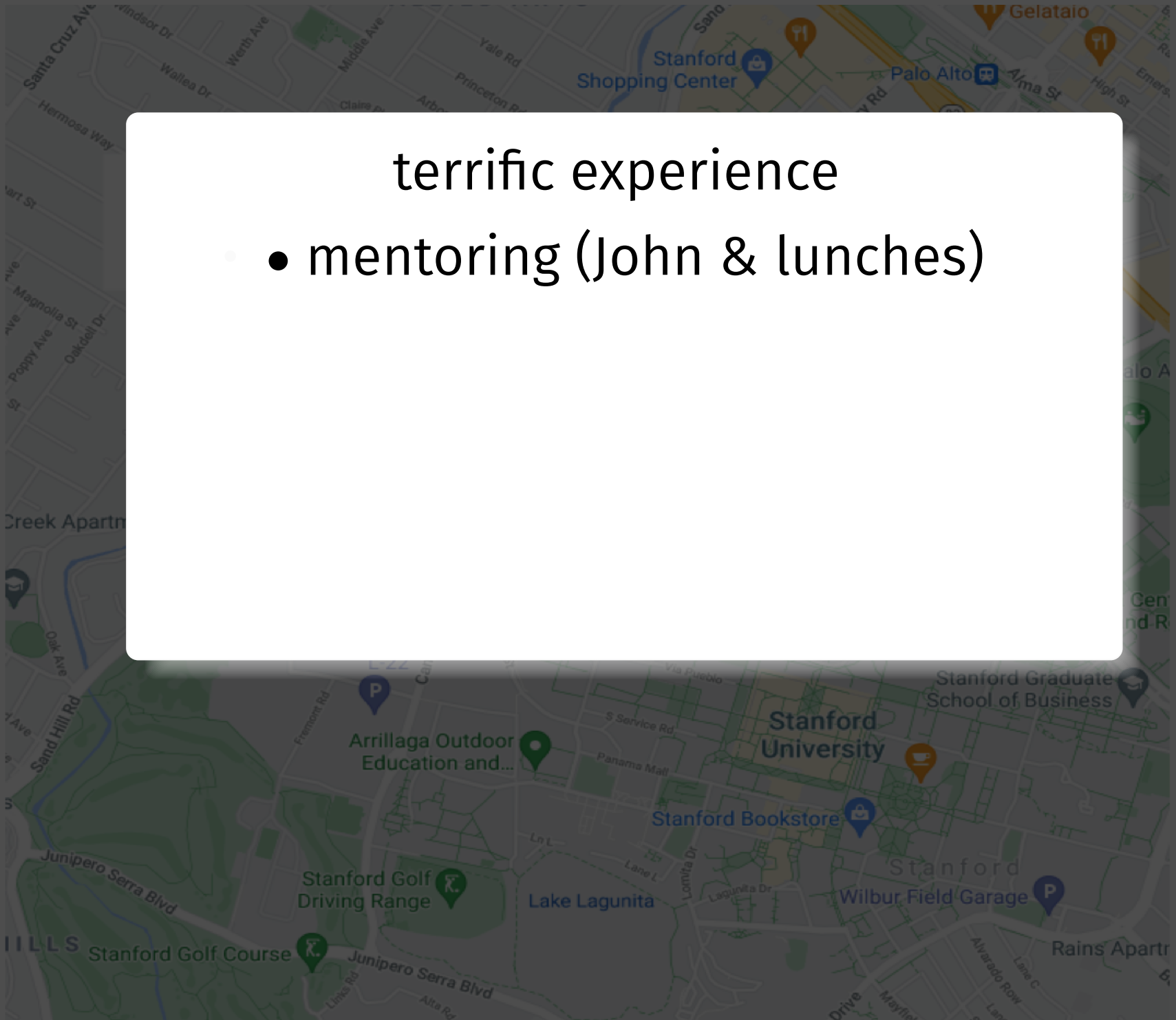
Foundations of Secure Compilation

Future Outlook

My Stanford Experience

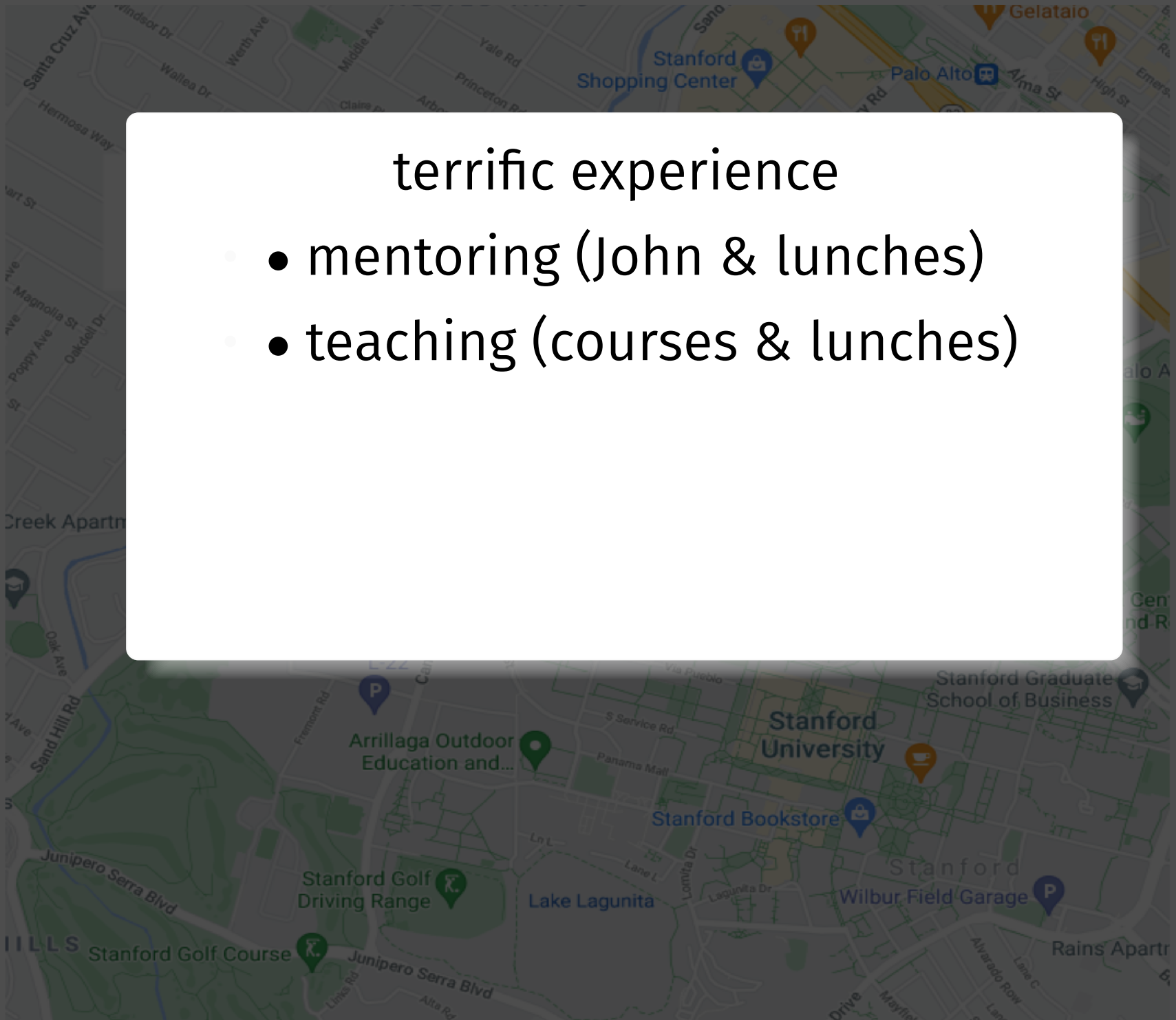

Germany
Belgium
Luxembourg
France
Czech Rep
Slovenia
Croatia
Bruges
Antwerp
Brussels
Lille
Amiens
Saint-Quentin
Reims
Düsseldorf
Cologne
Leipzig
Dresden
Görlitz
Chemnitz
Gera
Liberec
Prague
Wiesbade
Nuremberg
Karlsruhe
Strasbourg
Munich
Linz
Freiburg
Konstanz
Basel
Zurich
Dijon
Besançon
Bourges
Chalon-sur-Saône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


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



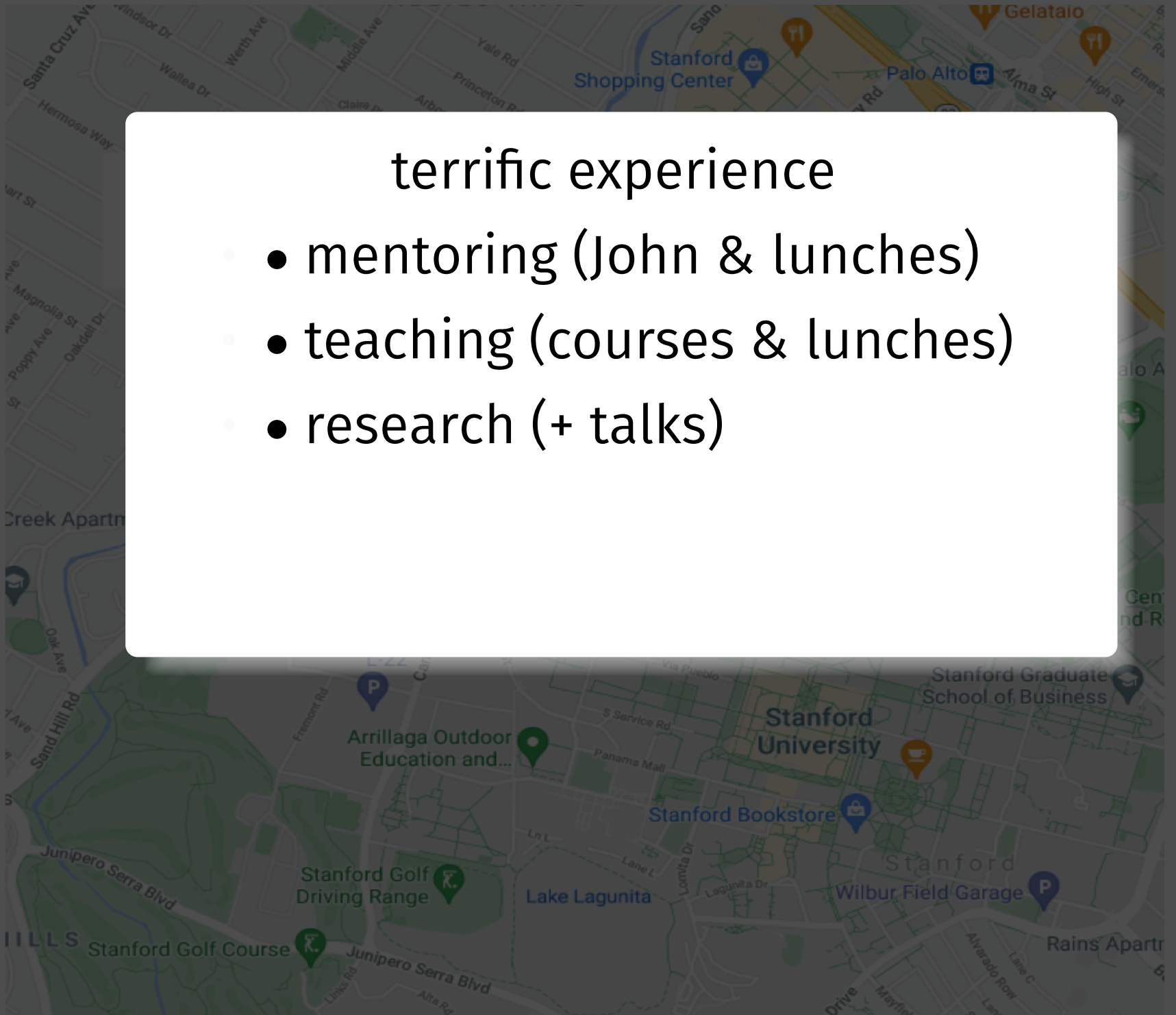
terrific experience

- mentoring (John & lunches)



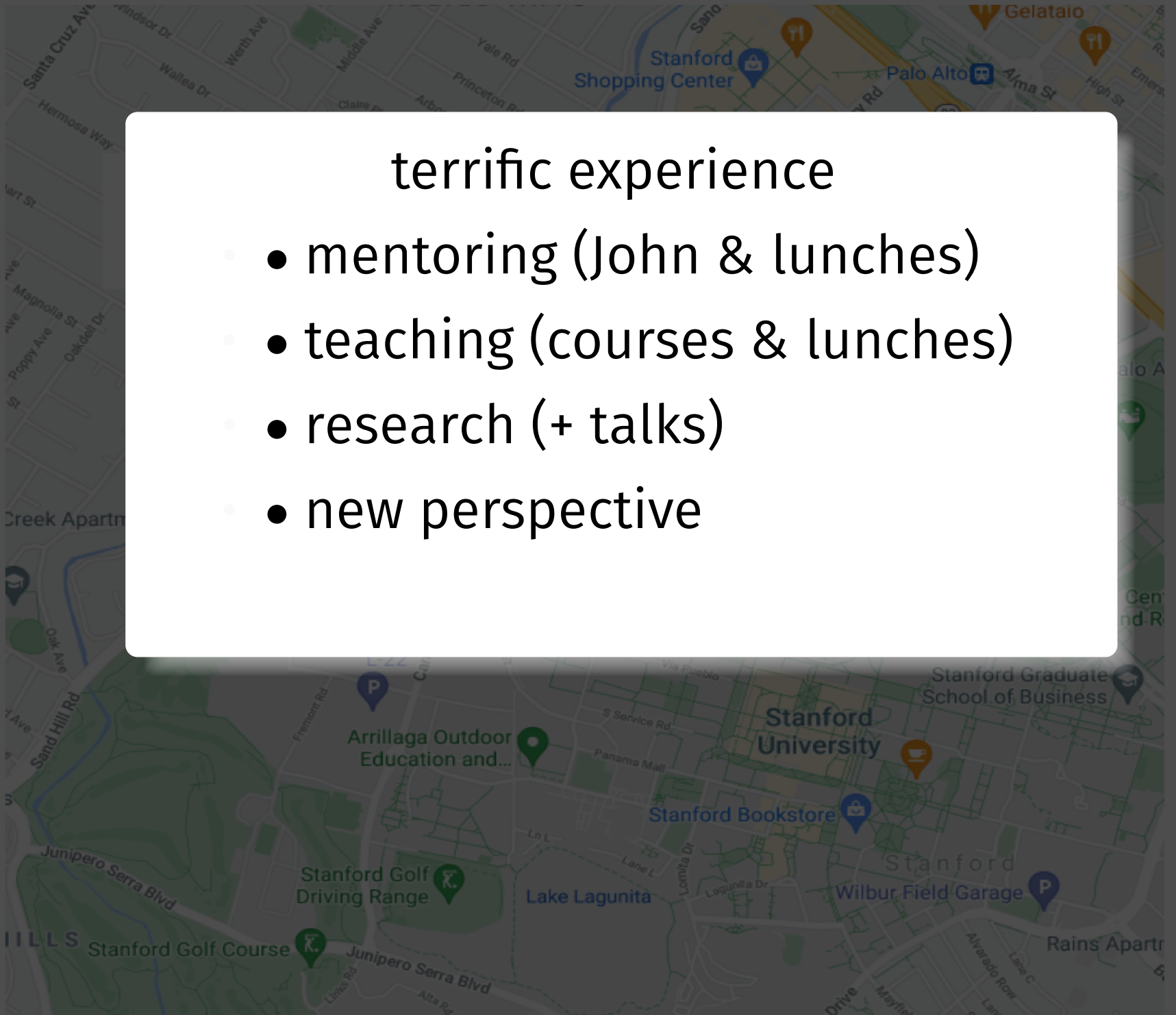
terrific experience

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



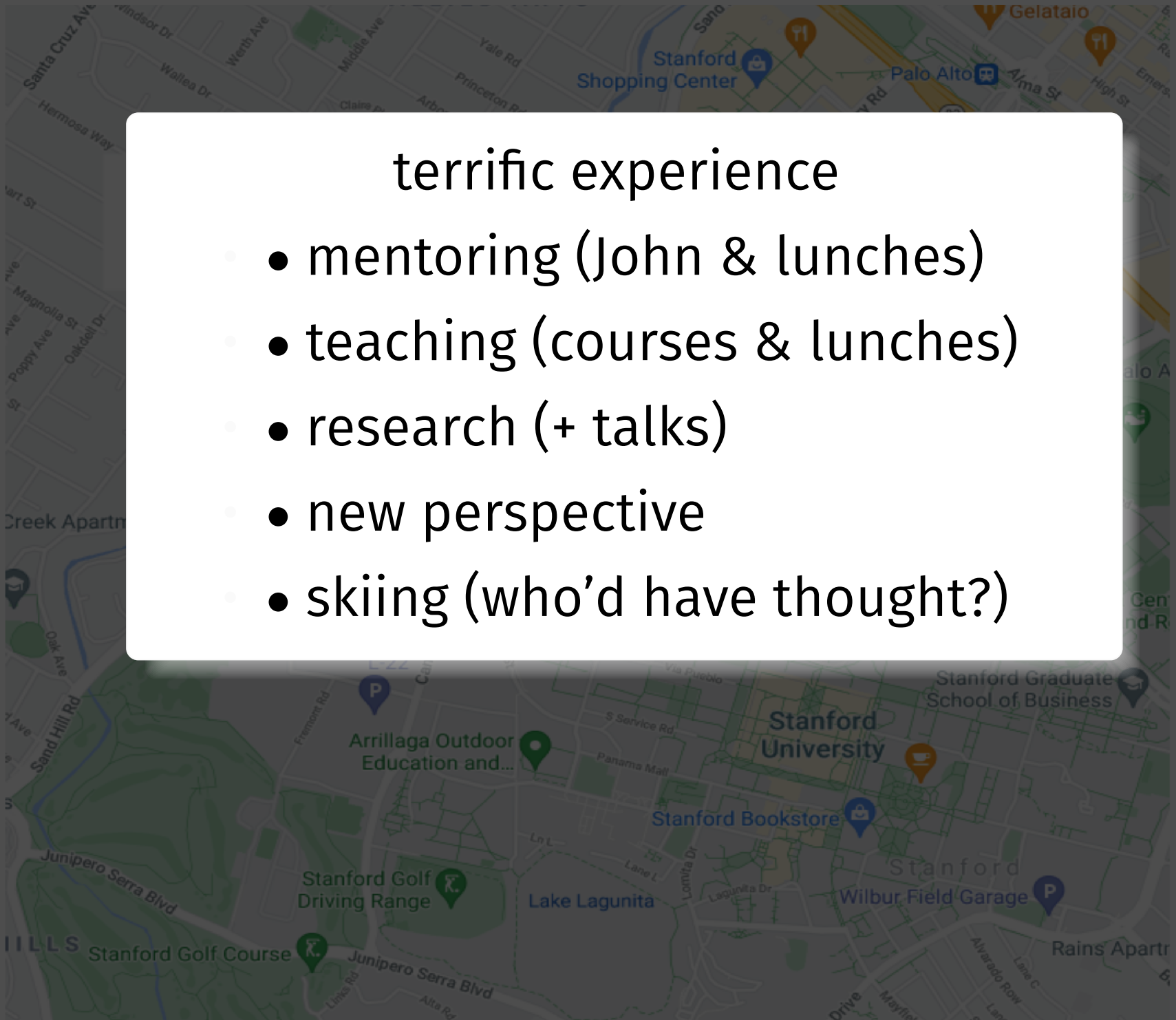
terrific experience

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



terrific experience

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



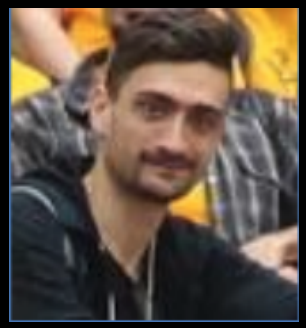
terrific experience

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

Foundations of Secure Compilation

Special Thanks to:

(wrt the contents of this talk)









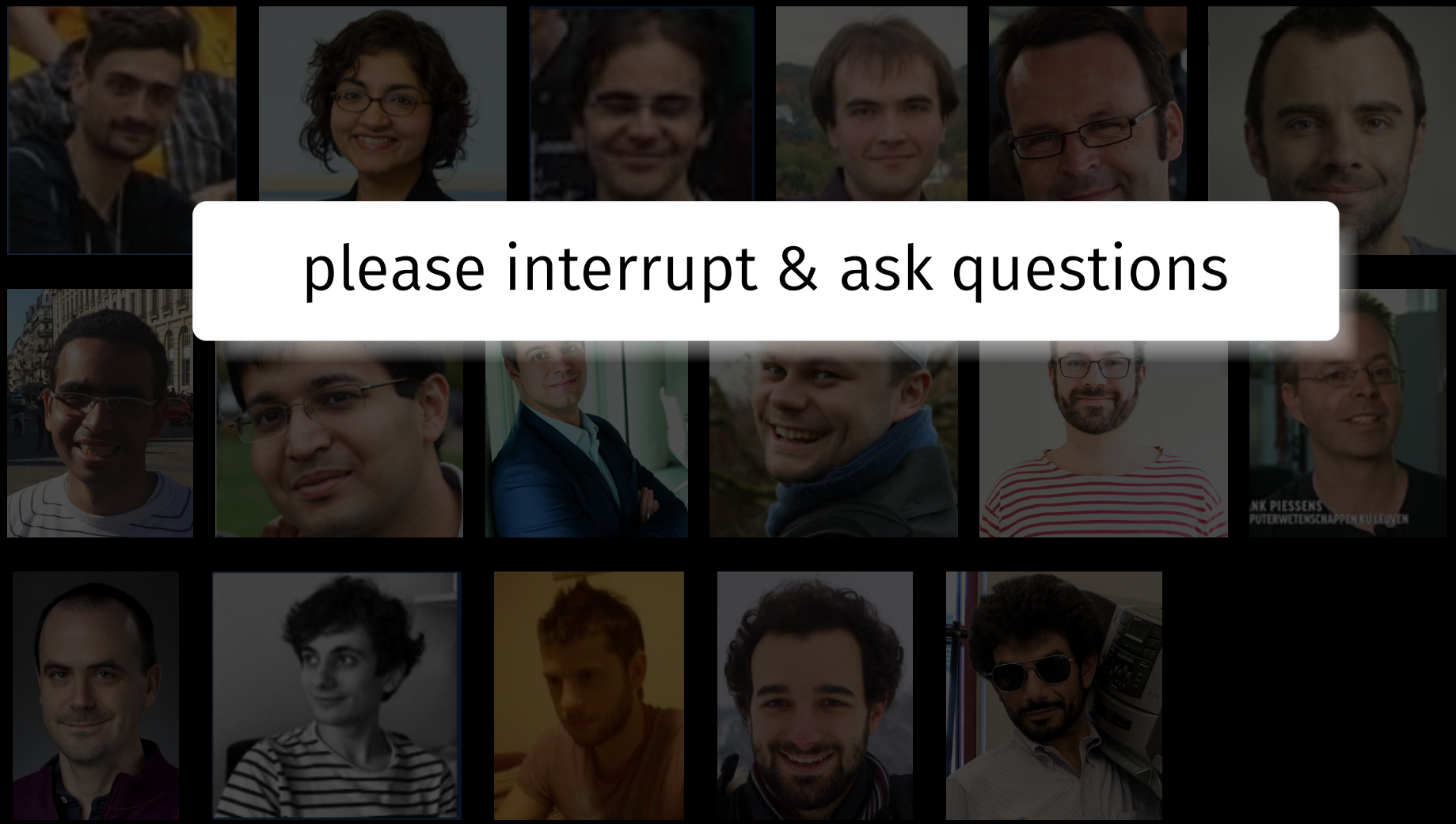







Special Thanks to:

(wrt the contents of this talk)



please interrupt & ask questions

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 ( , 

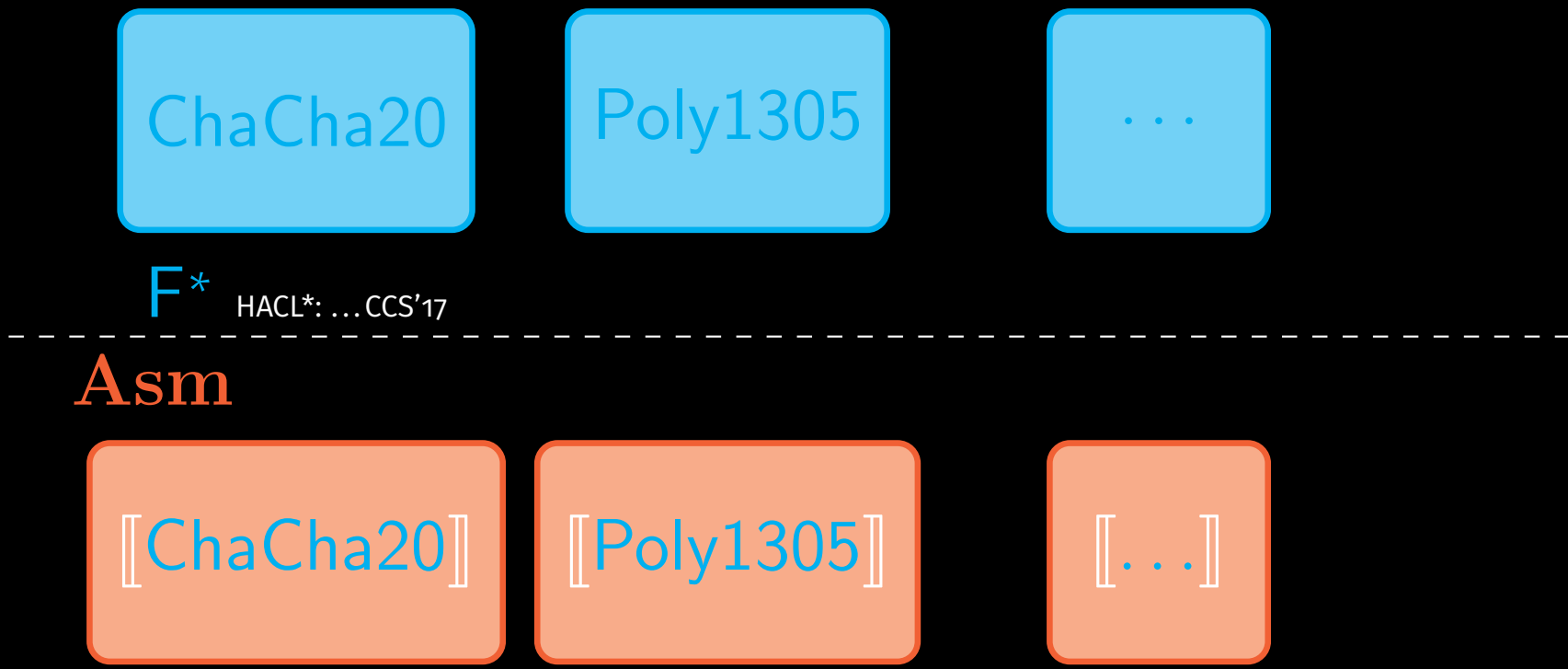
,  ,  , ...) provide:

- helpful abstractions to write secure code

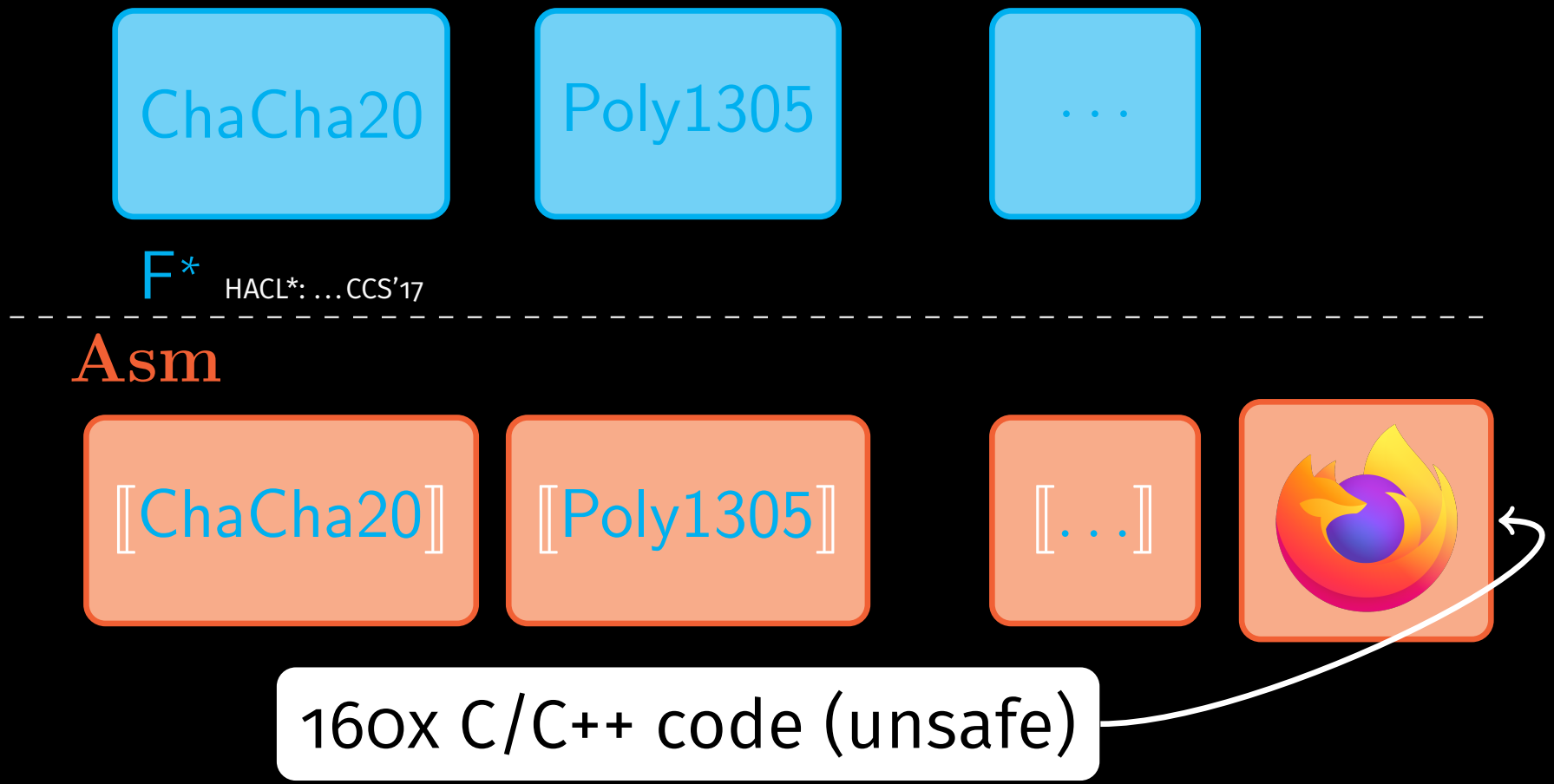
but

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

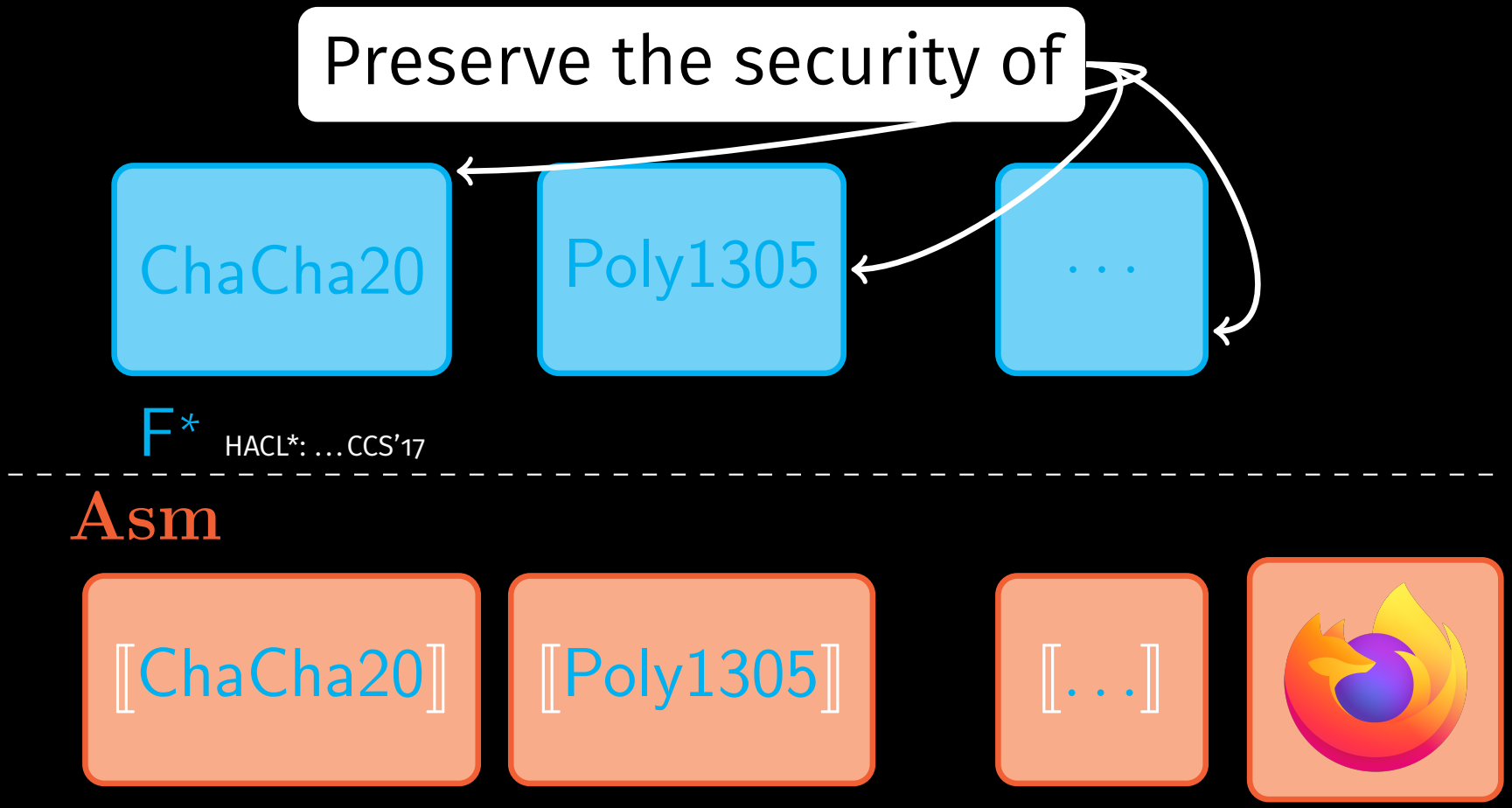
Secure Compilation: Example



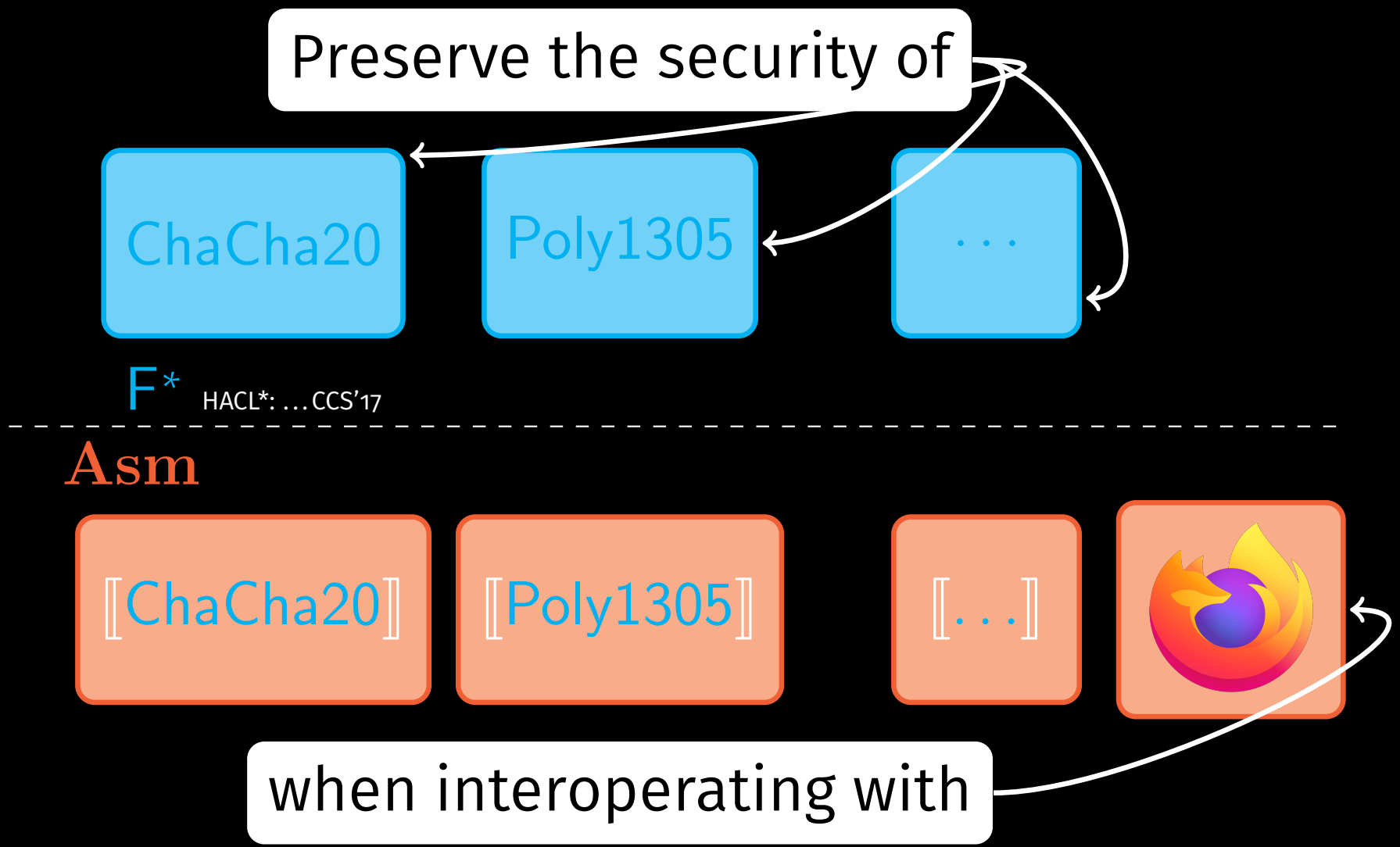
Secure Compilation: Example



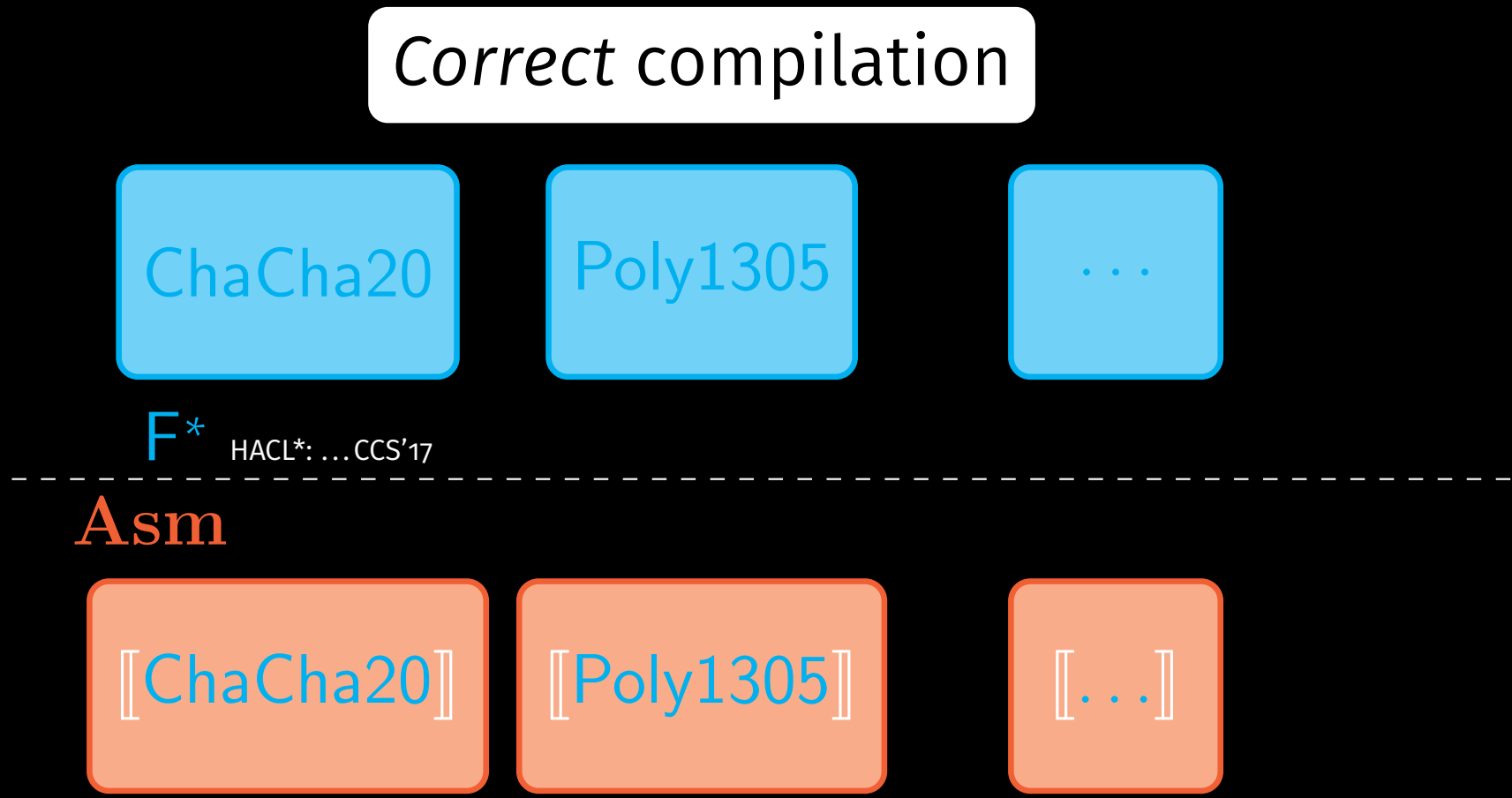
Secure Compilation: Example



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⟧ | ⟦. . .⟧ | 

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?

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,*

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

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 target – level attacker has the same power of a source – level one as captured by the semantics

Fully Abstract Compilation: Definition

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

$$\Updownarrow$$

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

Fully Abstract Compilation: Definition

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

$$\Uparrow$$

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

Fully Abstract Compilation: Definition

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

$$\Downarrow$$

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

Fully Abstract Compilation: Definition

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

$$\Downarrow$$

$$\forall \mathbf{A}.\, \mathbf{A}\,[\![\mathsf{P}_1]\!]\Downarrow \iff \mathbf{A}\,[\![\mathsf{P}_2]\!]\Downarrow$$

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

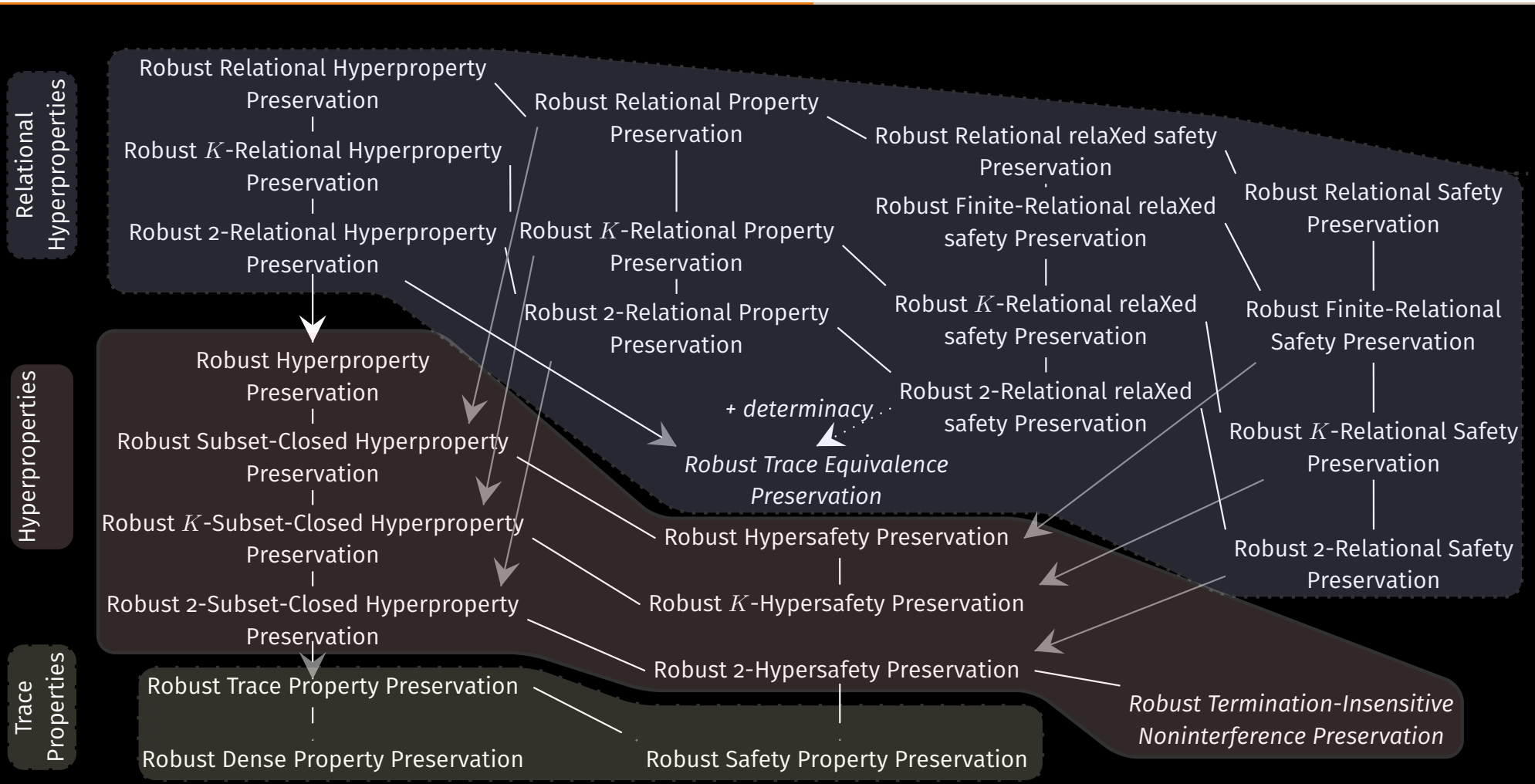
Are there Alternatives to FAC?

- 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

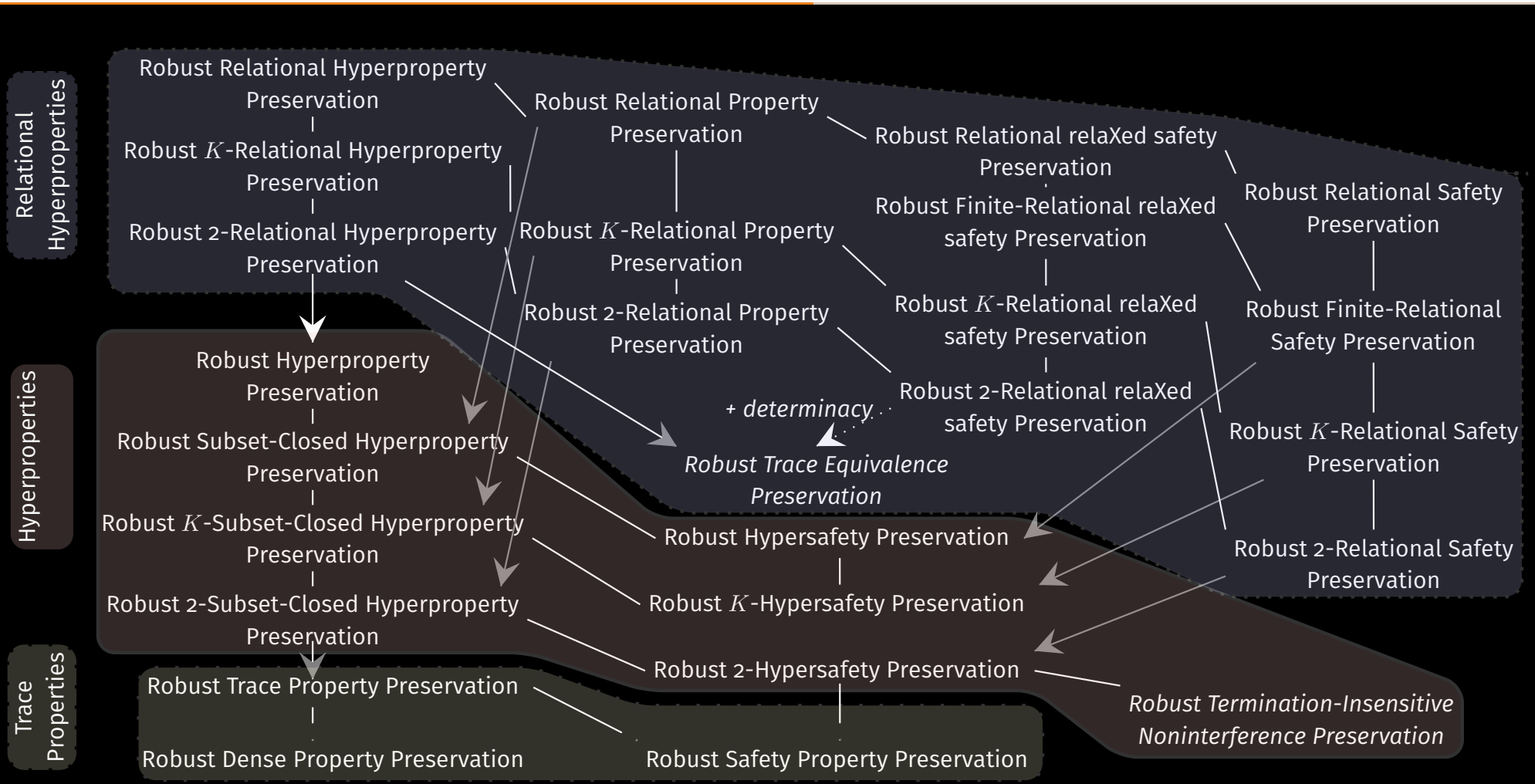
Robust Compilation Criteria

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



Robust Compilation Criteria

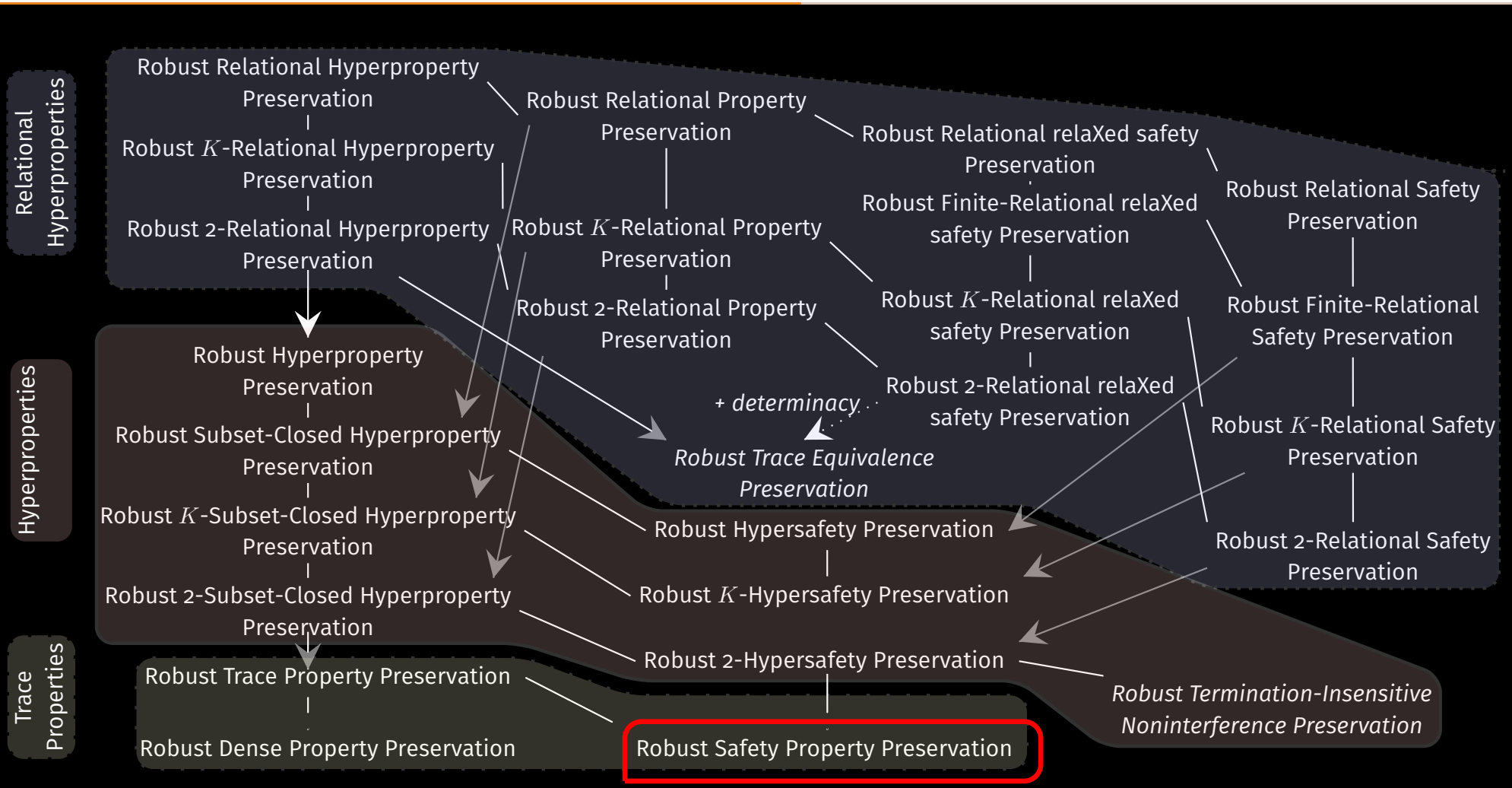
CSF'19, ESOP'20, ACM Toplas'21



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:

- $\mathrm{Property} - \mathrm{ful}$:
  - \+ 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:

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

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
  - = akin to some crypto statements (UC)

In Depth Example: RSC

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

In Depth Example: RSC

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

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

In Depth Example: RSC

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

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

In Depth Example: RSC

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

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

In Depth Example: RSC

$\llbracket\cdot\rrbracket$ = 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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

In Depth Example: RSC

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

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

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

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

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

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

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

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

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

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

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

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

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?