

My Group's Journey in Secure Compilation

Cătălin Hrițcu, MPI-SP



My companions on this journey:

Carmine Abate, Cezar-Constantin Andrici, Sven Argo, Arthur Azevedo de Amorim,
Jonathan Baumann, Roberto Blanco, Ștefan Ciobâcă, Adrien Durier, Akram El-Korashy,
Boris Eng, Ana Nora Evans, Guglielmo Fachini, Deepak Garg, Aïna Linn Georges,
Théo Laurent, Dongjae Lee, Guido Martínez, Marco Patrignani, Benjamin Pierce,
Exequiel Rivas, Basile Schlosser, Marco Stronati, Éric Tanter, Jérémy Thibault,
Andrew Tolmach, Théo Winterhalter, ...



Germany

Poland

Ukraine

Moldova

Suceava

Romania

Bulgaria







MSc and PhD in Saarbrücken (2005-2011)



MSc and PhD in Saarbrücken (2005-2011)



MSc and PhD in Saarbrücken (2005-2011)

- **1st semester** there: learned **functional programming**, **semantics of programming languages**, started doing **research**
 - fell in love with all these things; but wanted something practical

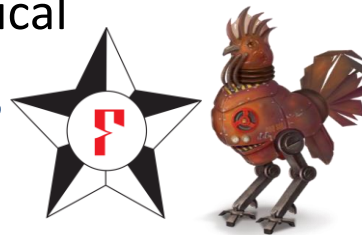
MSc and PhD in Saarbrücken (2005-2011)

- **1st semester** there: learned **functional programming**, **semantics of programming languages**, started doing **research**
 - fell in love with all these things; but wanted something practical
- **PhD on verification tools for cryptographic protocols**
 - these tools are precursors of the F^* verification system



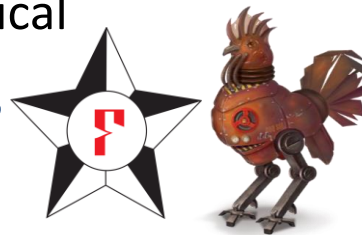
MSc and PhD in Saarbrücken (2005-2011)

- **1st semester** there: learned **functional programming**, **semantics of programming languages**, started doing **research**
 - fell in love with all these things; but wanted something practical
- **PhD on verification tools for cryptographic protocols**
 - these tools are precursors of the F^* verification system
 - learned mechanized proofs in **Rocq** using **Software Foundations** book



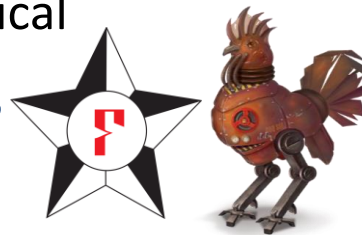
MSc and PhD in Saarbrücken (2005-2011)

- **1st semester** there: learned **functional programming**, **semantics of programming languages**, started doing **research**
 - fell in love with all these things; but wanted something practical
- **PhD on verification tools for cryptographic protocols**
 - these tools are precursors of the F^* verification system
 - learned mechanized proofs in **Rocq** using **Software Foundations** book
- **Made many friends during my studies, and I met my wife**



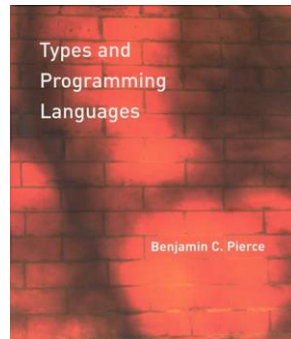
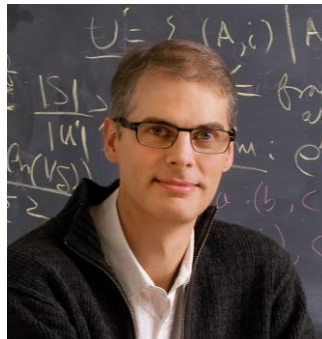
MSc and PhD in Saarbrücken (2005-2011)

- **1st semester** there: learned **functional programming**, **semantics of programming languages**, started doing **research**
 - fell in love with all these things; but wanted something practical
- **PhD on verification tools for cryptographic protocols**
 - these tools are precursors of the F^* verification system
 - learned mechanized proofs in **Rocq** using **Software Foundations** book
- **Made many friends during my studies, and I met my wife**



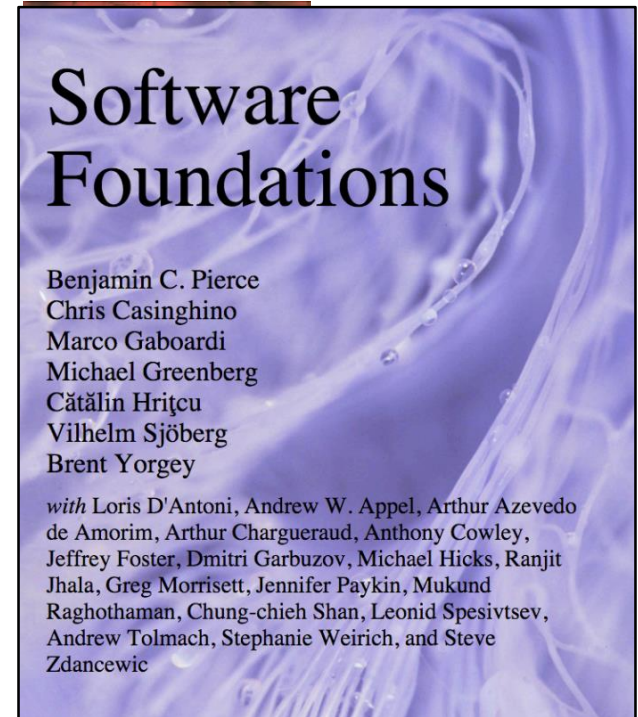
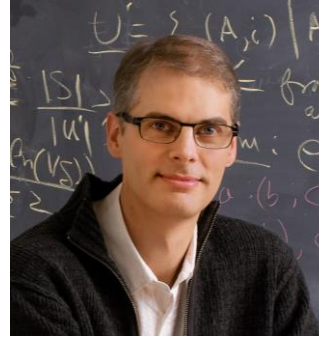
PostDoc at UPenn (2011-2013)

- with **Benjamin Pierce**



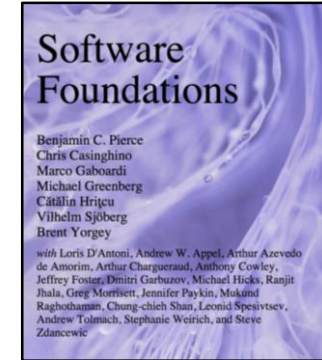
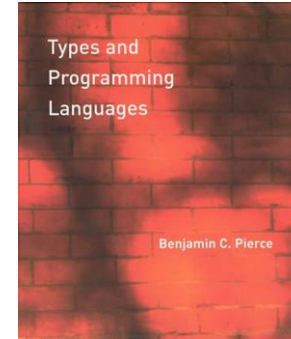
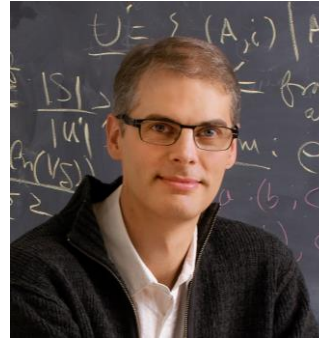
PostDoc at UPenn (2011-2013)

- with **Benjamin Pierce**
- got to **teach** with him



PostDoc at UPenn (2011-2013)

- with **Benjamin Pierce**
- got to **teach** with him
- **broadened research:**
DARPA CRASH/SAFE project

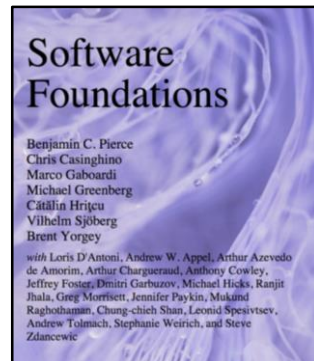
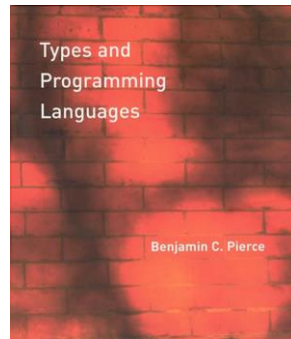
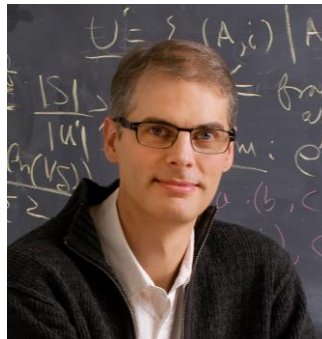


PostDoc at UPenn (2011-2013)

- with **Benjamin Pierce**
- got to **teach** with him
- **broadened research:**

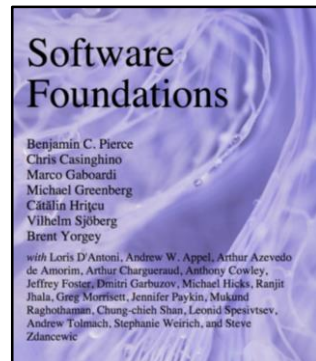
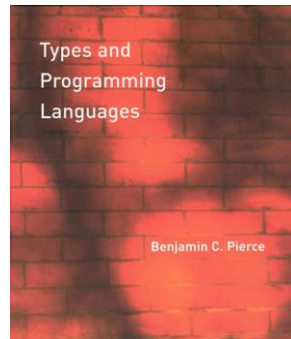
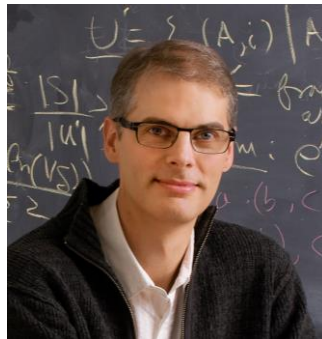
DARPA CRASH/SAFE project

– built more secure computer without legacy constraints



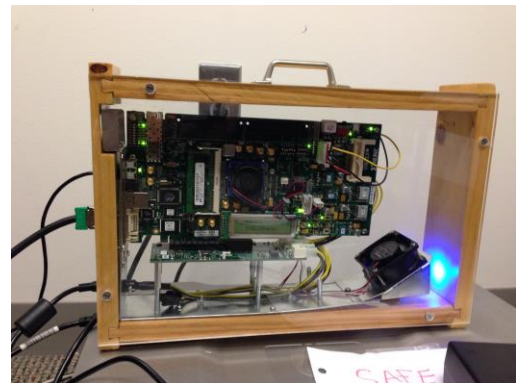
PostDoc at UPenn (2011-2013)

- with **Benjamin Pierce**
- got to **teach** with him
- **broadened research:**



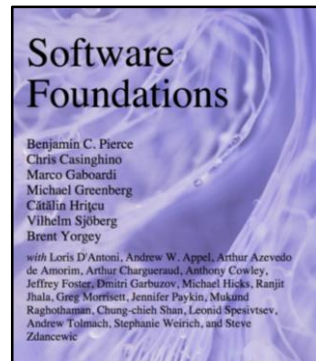
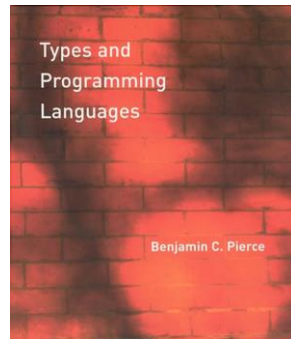
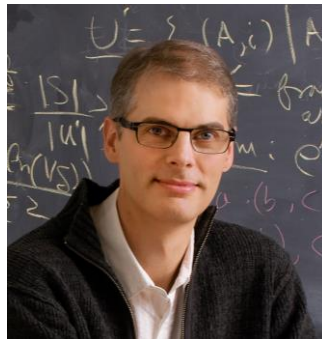
DARPA CRASH/SAFE project

- built more secure computer without legacy constraints
- clean-slate HW-SW co-design of a capability machine / tagged architecture



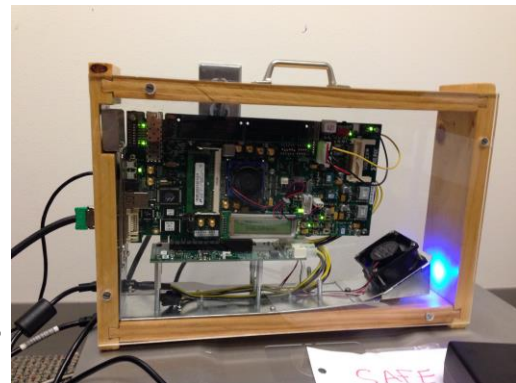
PostDoc at UPenn (2011-2013)

- with **Benjamin Pierce**
- got to **teach** with him
- **broadened research:**



DARPA CRASH/SAFE project

- built more secure computer without legacy constraints
- clean-slate HW-SW co-design of a capability machine / tagged architecture
- **Learned a lot:** programming languages, security, compilers, hardware, testing, ...



Researcher at Inria Paris (2013-2020)

- **Loved to have again a lot of freedom / scientific independence**
 - started advising interns, PhD students, PostDocs (i.e. building a group)

Researcher at Inria Paris (2013-2020)

- **Loved to have again a lot of freedom / scientific independence**
 - started advising interns, PhD students, PostDocs (i.e. building a group)
- **Complete redesign of F* with Nik Swamy et al (MSR & Inria)**
 - verification system combining a **proof assistant** (à la Rocq) with **SMT-based automation** (à la Hoare Logic)



Researcher at Inria Paris (2013-2020)

- **Loved to have again a lot of freedom / scientific independence**
 - started advising interns, PhD students, PostDocs (i.e. building a group)
- **Complete redesign of F* with Nik Swamy et al (MSR & Inria)**
 - verification system combining a **proof assistant** (à la Rocq) with **SMT-based automation** (à la Hoare Logic)
 - **interesting PL design** [POPL'16,'17,'18,'20,'24, ICFP'17,'19, ...]



Researcher at Inria Paris (2013-2020)

- **Loved to have again a lot of freedom / scientific independence**
 - started advising interns, PhD students, PostDocs (i.e. building a group)
- **Complete redesign of F* with Nik Swamy et al (MSR & Inria)**
 - verification system combining a **proof assistant** (à la Rocq) with **SMT-based automation** (à la Hoare Logic)
 - **interesting PL design** [POPL'16,'17,'18,'20,'24, ICFP'17,'19, ...]
 - **verified code shipping in Firefox, Linux, Windows, Python, ...**
 - crypto libraries (HACL*, EverCrypt), parsers and printers (EverParse), ...



Researcher at Inria Paris (2013-2020)

- **Loved to have again a lot of freedom / scientific independence**
 - started advising interns, PhD students, PostDocs (i.e. building a group)
- **Complete redesign of F* with Nik Swamy et al (MSR & Inria)**
 - verification system combining a **proof assistant** (à la Rocq) with **SMT-based automation** (à la Hoare Logic)
 - **interesting PL design** [POPL'16,'17,'18,'20,'24, ICFP'17,'19, ...]
 - **verified code shipping in Firefox, Linux, Windows, Python, ...**
 - crypto libraries (HACL*, EverCrypt), parsers and printers (EverParse), ...
- **Started QuickChick tool: property-based testing in Rocq**
 - [ICFP'13, ITP'15, JFP'16, POPL'17, ...]



Inria Paris: From tagged HW to secure compilation

- I also continued collaborating with Benjamin Pierce et al on **programming our tagged HW architecture**

Inria Paris: From tagged HW to secure compilation

- I also continued collaborating with Benjamin Pierce et al on **programming our tagged HW architecture**
 - **verified micro-policies** for information-flow control, memory safety, compartmentalization, control-flow integrity [POPL'14, ASPLOS'15, SP'15, ...]

Inria Paris: From tagged HW to secure compilation

- I also continued collaborating with Benjamin Pierce et al on **programming our tagged HW architecture**
 - **verified micro-policies** for information-flow control, memory safety, compartmentalization, control-flow integrity [POPL'14, ASPLOS'15, SP'15, ...]
 - but **it's very hard to talk about security by looking only at the ASM level**

Inria Paris: From tagged HW to secure compilation

- I also continued collaborating with Benjamin Pierce et al on **programming our tagged HW architecture**
 - **verified micro-policies** for information-flow control, memory safety, compartmentalization, control-flow integrity [POPL'14, ASPLOS'15, SP'15, ...]
 - but **it's very hard to talk about security by looking only at the ASM level**
- Developers think about security in terms of the **abstractions provided by their programming languages**

Inria Paris: From tagged HW to secure compilation

- I also continued collaborating with Benjamin Pierce et al on **programming our tagged HW architecture**
 - **verified micro-policies** for information-flow control, memory safety, compartmentalization, control-flow integrity [POPL'14, ASPLOS'15, SP'15, ...]
 - but **it's very hard to talk about security by looking only at the ASM level**
- Developers think about security in terms of the **abstractions provided by their programming languages**
 - big problem is that **normal compilers don't enforce those abstractions**

Inria Paris: From tagged HW to secure compilation

- I also continued collaborating with Benjamin Pierce et al on **programming our tagged HW architecture**
 - **verified micro-policies** for information-flow control, memory safety, compartmentalization, control-flow integrity [POPL'14, ASPLOS'15, SP'15, ...]
 - but **it's very hard to talk about security by looking only at the ASM level**
- Developers think about security in terms of the **abstractions provided by their programming languages**
 - big problem is that **normal compilers don't enforce those abstractions**
 - ~2015 started working on **secure compilation to protect these abstractions all the way down** (initially to our tagged architecture)

Inria Paris: From tagged HW to secure compilation

- I also continued collaborating with Benjamin Pierce et al on **programming our tagged HW architecture**
 - **verified micro-policies** for information-flow control, memory safety, compartmentalization, control-flow integrity [POPL'14, ASPLOS'15, SP'15, ...]
 - but **it's very hard to talk about security by looking only at the ASM level**
- Developers think about security in terms of the **abstractions provided by their programming languages**
 - big problem is that **normal compilers don't enforce those abstractions**
 - ~2015 started working on **secure compilation to protect these abstractions all the way down** (initially to our tagged architecture)
 - in 2017 this lead to ERC Starting Grant SECOMP, project still going strong

Moved to Bochum in April 2020



Moved to Bochum in April 2020

- "Interesting" start in Bochum with 1st pandemic wave

Opportunity to contribute to growing
MPI-SP into a top international institute



Opportunity to contribute to growing MPI-SP into a top international institute

Went from 2 to 12 research groups (and still growing):

- Christof Paar
- Gilles Barthe
- Peter Schwabe
- Asia Biega
- Clara Schneidewind
- Marcel Böhme
- Yixin Zou
- Abraham Mhaidli
- Mia Cha
- Jana Hofmann
- Carmela Troncoso
- ...



Things got much better after Corona

Things got much better after Corona

- For instance, I got **back into running after ~14 years of break**

Things got much better after Corona

- For instance, I got **back into running after ~14 years of break**
 - Running with colleagues at MPI-SP, joined the RUB running group, and also running again with my wife



Things got much better after Corona

- For instance, I got **back into running after ~14 years of break**
 - Running with colleagues at MPI-SP, joined the RUB running group, and also running again with my wife
 - Hobby that combines sports, socializing, and practicing German

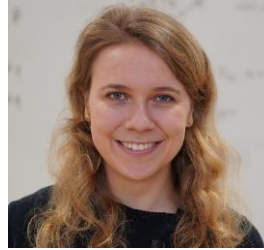


Things got much better after Corona

- For instance, I got **back into running after ~14 years of break**
 - Running with colleagues at MPI-SP, joined the RUB running group, and also running again with my wife
 - Hobby that combines sports, socializing, and practicing German
- Ran my 2nd half-marathon a month ago in Duisburg
 - 20 minutes faster than 18 years ago, and this time I didn't get injured

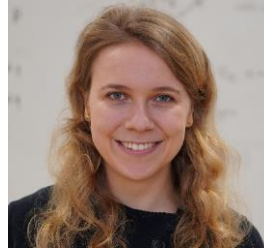
After Corona also teaching in CS@RUB

After Corona also teaching in CS@RUB



with Roberto Blanco, Clara Schneidewind, Jana Hofmann

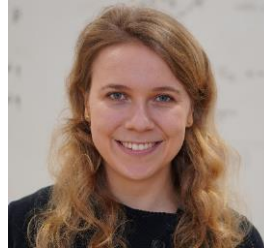
After Corona also teaching in CS@RUB



with Roberto Blanco, Clara Schneidewind, Jana Hofmann

1. Proofs are Programs - gentle introduction to mechanized proofs in Rocq

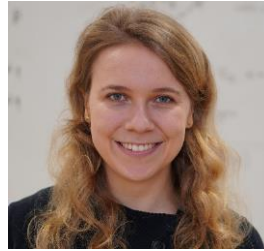
After Corona also teaching in CS@RUB



with Roberto Blanco, Clara Schneidewind, Jana Hofmann

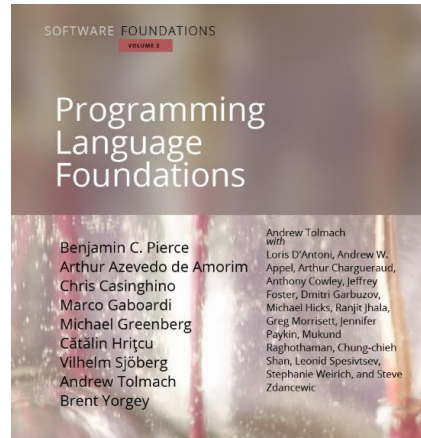
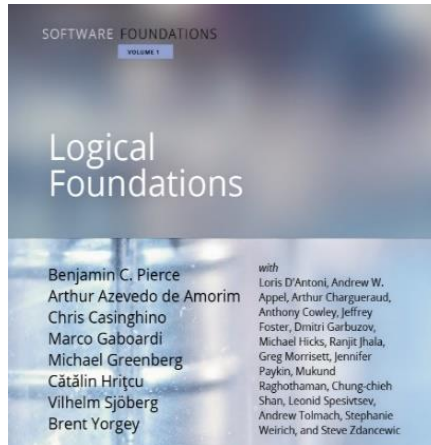
- 1. Proofs are Programs** - gentle introduction to mechanized proofs in Rocq
- 2. Foundations of Programming Languages, Verification, and Security**

After Corona also teaching in CS@RUB

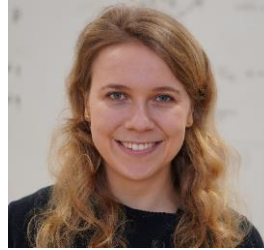


with Roberto Blanco, Clara Schneidewind, Jana Hofmann

- 1. Proofs are Programs** - gentle introduction to mechanized proofs in Rocq
- 2. Foundations of Programming Languages, Verification, and Security**

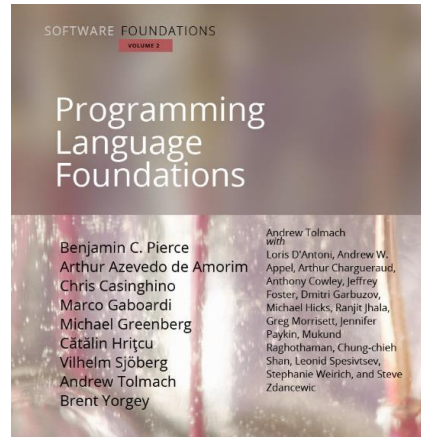
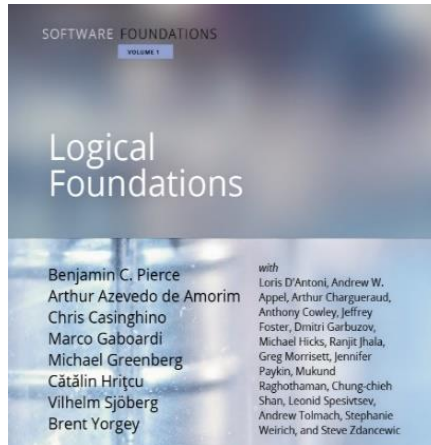


After Corona also teaching in CS@RUB

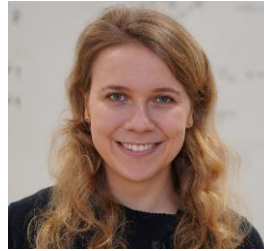


with Roberto Blanco, Clara Schneidewind, Jana Hofmann

1. **Proofs are Programs** - gentle introduction to mechanized proofs in Rocq
2. **Foundations of Programming Languages, Verification, and Security**



After Corona also teaching in CS@RUB



with Roberto Blanco, Clara Schneidewind, Jana Hofmann

1. **Proofs are Programs** - gentle introduction to mechanized proofs in Rocq
2. **Foundations of Programming Languages, Verification, and Security**

Information flow control:

- static and dynamic enforcement

Preventing timing side channels:

- cryptographic constant time
- speculative constant time

Relational Hoare Logic:

- program equivalence and security



Formally Verified Security group at MPI-SP

– PhD students: **Cezar Andrici** and **Jérémy Thibault**

Formally Verified Security group at MPI-SP

- PhD students: **Cezar Andrici** and **Jérémy Thibault**
- PostDocs: **Yonghyun Kim** (officially starting with us tomorrow),
Rob Blanco (starting as Associate Prof at TU Eindhoven tomorrow)

Formally Verified Security group at MPI-SP

- PhD students: **Cezar Andrici** and **Jérémy Thibault**
- PostDocs: **Yonghyun Kim** (officially starting with us tomorrow),
Rob Blanco (starting as Associate Prof at TU Eindhoven tomorrow)
- Interns: **Basile Schlosser**, **Jonathan Baumann** (ENS Paris-Saclay),
Julay Leatherman-Brooks (Portland State University, starting soon)

Formally Verified Security group at MPI-SP

- PhD students: **Cezar Andrici** and **Jérémy Thibault**
- PostDocs: **Yonghyun Kim** (officially starting with us tomorrow),
Rob Blanco (starting as Associate Prof at TU Eindhoven tomorrow)
- Interns: **Basile Schlosser**, **Jonathan Baumann** (ENS Paris-Saclay),
Julay Leatherman-Brooks (Portland State University, starting soon)
- Some recent alumni:
 - **Adrien Durier** (ex PostDoc, Associate Professor, Uni. Paris-Saclay)
 - **Théo Winterhalter** (ex PostDoc, Tenured Researcher, Inria Saclay)

Formally Verified Security group at MPI-SP

- PhD students: **Cezar Andrici** and **Jérémy Thibault**
- PostDocs: **Yonghyun Kim** (officially starting with us tomorrow),
Rob Blanco (starting as Associate Prof at TU Eindhoven tomorrow)
- Interns: **Basile Schlosser**, **Jonathan Baumann** (ENS Paris-Saclay),
Julay Leatherman-Brooks (Portland State University, starting soon)
- Some recent alumni:
 - **Adrien Durier** (ex PostDoc, Associate Professor, Uni. Paris-Saclay)
 - **Théo Winterhalter** (ex PostDoc, Tenured Researcher, Inria Saclay)
 - **Carmin Abate** (ex PhD student, Researcher, Barkhausen Institute)
 - **Guido Martínez** (ex PhD student, Research Engineer, Microsoft Research)

Formally Verified Security group at MPI-SP

- PhD students: **Cezar Andrici** and **Jérémy Thibault**
- PostDocs: **Yonghyun Kim** (officially starting with us tomorrow),
Rob Blanco (starting as Associate Prof at TU Eindhoven tomorrow)
- Interns: **Basile Schlosser**, **Jonathan Baumann** (ENS Paris-Saclay),
Julay Leatherman-Brooks (Portland State University, starting soon)
- Some recent alumni:
 - **Adrien Durier** (ex PostDoc, Associate Professor, Uni. Paris-Saclay)
 - **Théo Winterhalter** (ex PostDoc, Tenured Researcher, Inria Saclay)
 - **Carmin Abate** (ex PhD student, Researcher, Barkhausen Institute)
 - **Guido Martínez** (ex PhD student, Research Engineer, Microsoft Research)
 - **Aïna Linn Georges** (ex Intern, PostDoc, MPI-SWS)
 - **Dongjae Lee** (ex Intern, PhD student, MIT)

My Group's Journey in Secure Compilation



**Good programming languages provide
helpful abstractions for writing more secure code**

Good programming languages provide helpful abstractions for writing more secure code

- structured control flow, procedures, modules, types, interfaces, correctness and security specifications, ...

Good programming languages provide helpful abstractions for writing more secure code

- structured control flow, procedures, modules, types, interfaces, correctness and security specifications, ...
- **suppose we have a secure source program ...**
 - For instance formally verified in F^* (e.g. EverCrypt verified crypto library)
 - Or a program written in safe Rust or OCaml



- What happens when we compile such a secure source program and link it with adversarial target code?

- What happens when we compile such a secure source program and link it with adversarial target code?



- What happens when we compile such a secure source program and link it with adversarial target code?



- What happens when we compile such a secure source program and link it with adversarial target code?
 - not just hypothetical: verified code often linked with unverified code, safe OCaml and Rust often linked with C/C++/ASM code (e.g. libraries)



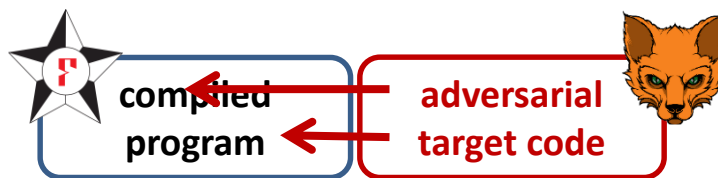
- What happens when we compile such a secure source program and link it with **adversarial target code**?
 - **not just hypothetical**: verified code often linked with unverified code, safe OCaml and Rust often linked with C/C++/ASM code (e.g. libraries)
 - target-level code can be buggy, vulnerable, compromised, malicious



- What happens when we compile such a secure source program and link it with adversarial target code?
 - **not just hypothetical**: verified code often linked with unverified code, safe OCaml and Rust often linked with C/C++/ASM code (e.g. libraries)
 - target-level code can be buggy, vulnerable, compromised, malicious
 - **currently: all abstractions and source-level guarantees are lost**

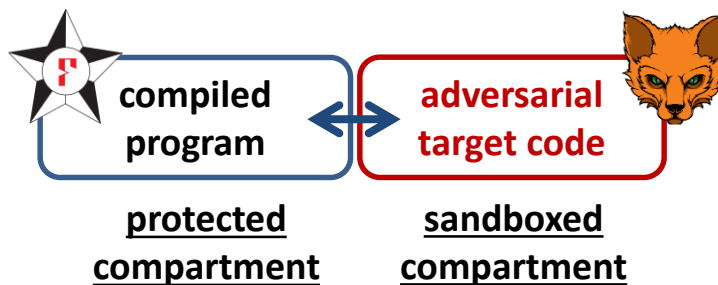


- What happens when we compile such a secure source program and link it with adversarial target code?
 - not just hypothetical: verified code often linked with unverified code, safe OCaml and Rust often linked with C/C++/ASM code (e.g. libraries)
 - target-level code can be buggy, vulnerable, compromised, malicious
 - currently: all abstractions and source-level guarantees are lost
 - lower-level attacks become possible: break control flow, memory safety, etc.



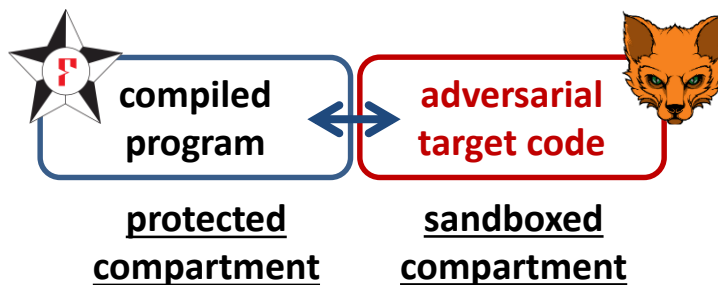
Secure Compilation

- What happens when we compile such a secure source program and link it with **adversarial target code**?
 - **not just hypothetical**: verified code often linked with unverified code, safe OCaml and Rust often linked with C/C++/ASM code (e.g. libraries)
 - target-level code can be buggy, vulnerable, compromised, malicious
 - **currently: all abstractions and source-level guarantees are lost**
 - **lower-level attacks become possible: break control flow, memory safety, etc.**



Secure Compilation of Secure Source Programs

- What happens when we compile such a secure source program and link it with adversarial target code?
 - not just hypothetical: verified code often linked with unverified code, safe OCaml and Rust often linked with C/C++/ASM code (e.g. libraries)
 - target-level code can be buggy, vulnerable, compromised, malicious
 - currently: all abstractions and source-level guarantees are lost
 - lower-level attacks become possible: break control flow, memory safety, etc.



Secure Compilation of Secure Source Programs

- Protect source-level abstractions all the way down
even against linked adversarial target code

Secure Compilation of Secure Source Programs

- **Protect source-level abstractions all the way down even against linked adversarial target code**
 - **various enforcement mechanisms for sandboxing untrusted code:**
capability machines, tagged architectures, software-fault isolation (SFI), ...
 - shared responsibility: compiler, linker, loader, OS, HW

Secure Compilation of Secure Source Programs

- **Protect source-level abstractions all the way down even against linked adversarial target code**
 - various enforcement mechanisms for sandboxing untrusted code: capability machines, tagged architectures, software-fault isolation (SFI), ...
 - shared responsibility: compiler, linker, loader, OS, HW
- **This is very challenging:**
 - the originally proposed formal criterion was fully abstract compilation [Martín Abadi, Protection in programming-language translations. 1999]
 - very difficult to enforce and very difficult to prove
 - (in)famous wrong full abstraction conjecture survived decades [Eijiro Sumii and Benjamin Pierce POPL'04, Dominique Devriese et al. POPL'18]
 - 250 pages of proof on paper even for toy compilers

Secure Compilation of Vulnerable Source Programs

Secure Compilation of Vulnerable Source Programs

- Insecure languages like C enable **devastating vulnerabilities**



Secure Compilation of Vulnerable Source Programs

- Insecure languages like C enable **devastating vulnerabilities**
 - **undefined behavior** pervasive in C: buffer overflows, use after frees, double frees, invalid type casts, various concurrency bugs, ...



Secure Compilation of Vulnerable Source Programs

- Insecure languages like C enable **devastating vulnerabilities**
 - **undefined behavior** pervasive in C: buffer overflows, use after frees, double frees, invalid type casts, various concurrency bugs, ...
 - **undefined behavior** also present in unsafe Rust, OCaml, ...



Secure Compilation of Vulnerable Source Programs

- Insecure languages like C enable **devastating vulnerabilities**
 - **undefined behavior** pervasive in C: buffer overflows, use after free, double frees, invalid type casts, various concurrency bugs, ...
 - **undefined behavior** also present in unsafe Rust, OCaml, ...
- Yet even the C language does provide some useful abstractions:
 - structured control flow, procedures, pointers to shared memory



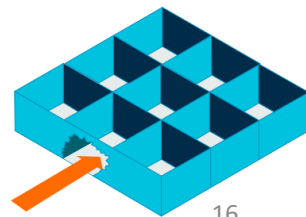
Secure Compilation of Vulnerable Source Programs

- Insecure languages like C enable **devastating vulnerabilities**
 - **undefined behavior** pervasive in C: buffer overflows, use after frees, double frees, invalid type casts, various concurrency bugs, ...
 - **undefined behavior** also present in unsafe Rust, OCaml, ...
- Yet even the C language does provide some useful abstractions:
 - structured control flow, procedures, pointers to shared memory
 - **but not enforced during compilation for programs with UB: all guarantees are lost!**



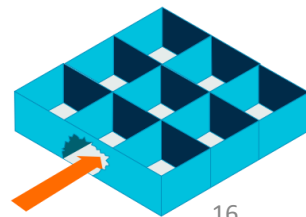
Secure Compilation of Vulnerable Source Programs

- Insecure languages like C enable **devastating vulnerabilities**
 - **undefined behavior** pervasive in C: buffer overflows, use after frees, double frees, invalid type casts, various concurrency bugs, ...
 - **undefined behavior** also present in unsafe Rust, OCaml, ...
- Yet even the C language does provide some useful abstractions:
 - structured control flow, procedures, pointers to shared memory
 - **but not enforced during compilation for programs with UB: all guarantees are lost!**
 - we add one more abstraction to C: **fine-grained compartments that can naturally interact**

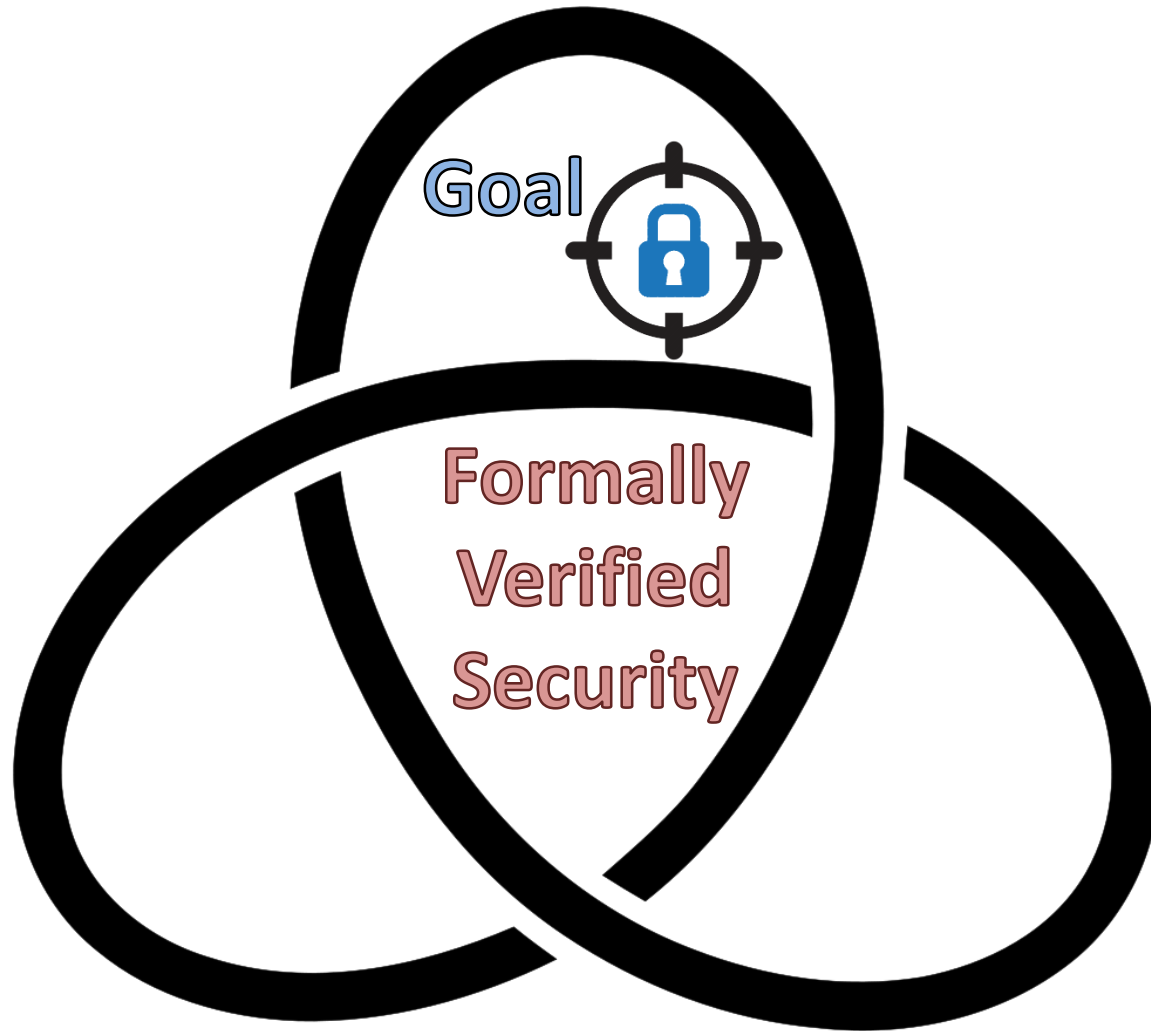


Secure Compilation of Vulnerable Source Programs

- Insecure languages like C enable **devastating vulnerabilities**
 - **undefined behavior** pervasive in C: buffer overflows, use after frees, double frees, invalid type casts, various concurrency bugs, ...
 - **undefined behavior** also present in unsafe Rust, OCaml, ...
- Yet even the C language does provide some useful abstractions:
 - structured control flow, procedures, pointers to shared memory
 - **but not enforced during compilation for programs with UB: all guarantees are lost!**
 - we add one more abstraction to C: **fine-grained compartments that can naturally interact**
- **Secure compilation chain that protects these abstractions**
 - all the way down, at compartment boundaries (hopefully more efficient than removing UB)
 - against compartments dynamically compromised by undefined behavior
 - using the same kind of enforcement mechanisms for **compartmentalization**



Formally Verified Security







Secure Compilation





1. Security Goal



1. Security Goal



OCaml

- Question A:

What does it mean to securely compile a secure source program **against linked adversarial target-level code**?





1. Security Goal



OCaml



- **Question A:**

What does it mean to securely compile a secure source program **against linked adversarial target-level code**?

– e.g. simple verified web server, linked with unverified libraries [POPL'24]

Preserving security against adversarial contexts



Preserving security **against adversarial contexts**

\forall security property π



Preserving security **against adversarial contexts**

\forall security property π



Preserving security **against adversarial contexts**

\forall security property π



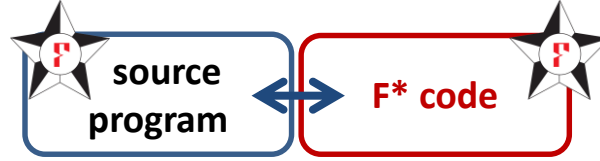
satisfies π



Preserving security against adversarial contexts

\forall security property π

\forall F^* code

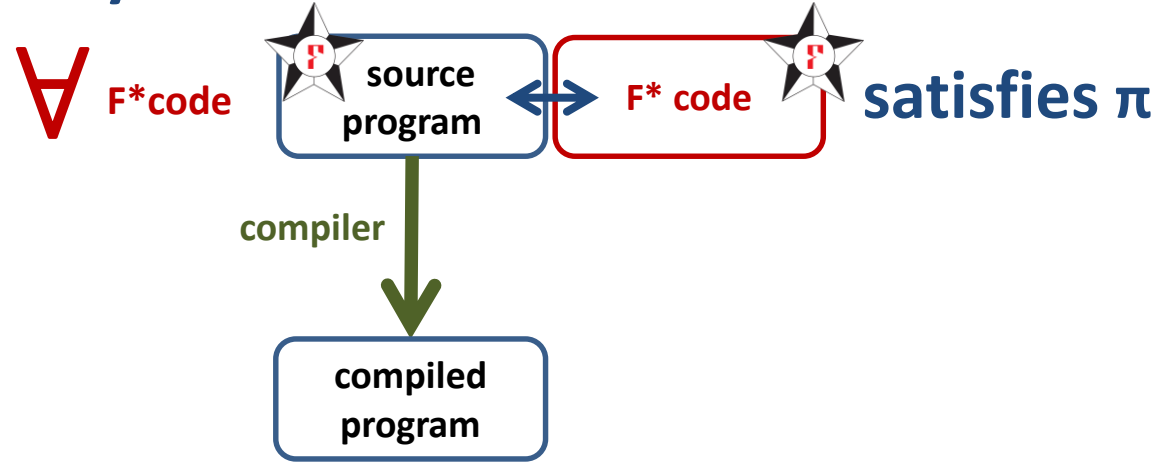


satisfies π



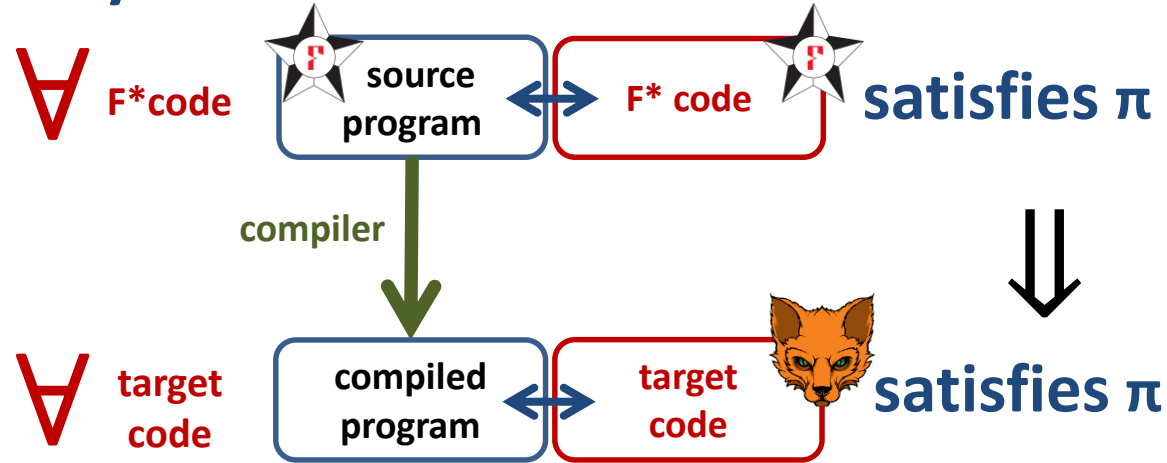
Preserving security against adversarial contexts

\forall security property π



Preserving security against adversarial contexts

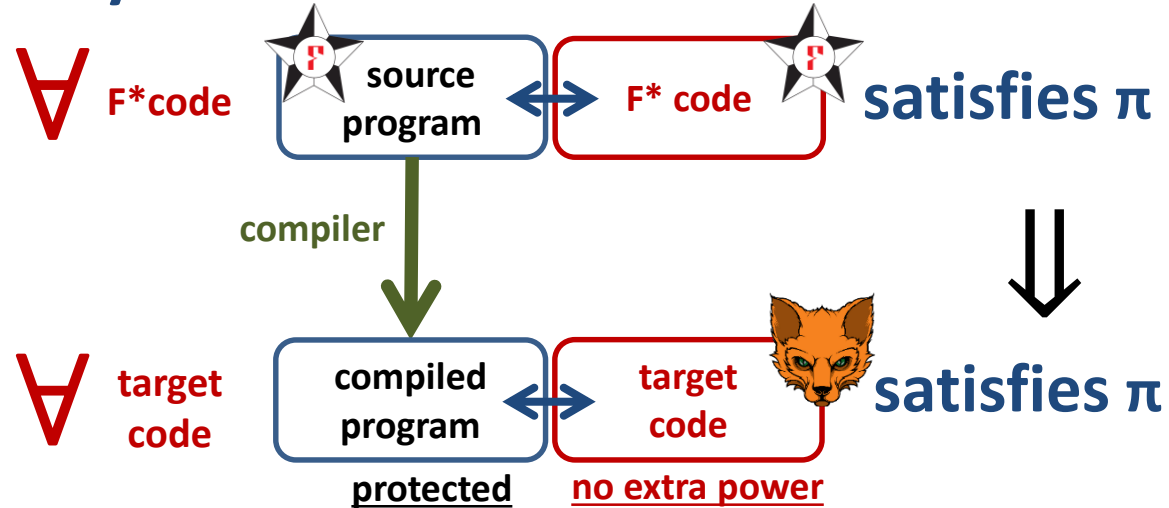
\forall security property π



Preserving security against adversarial contexts

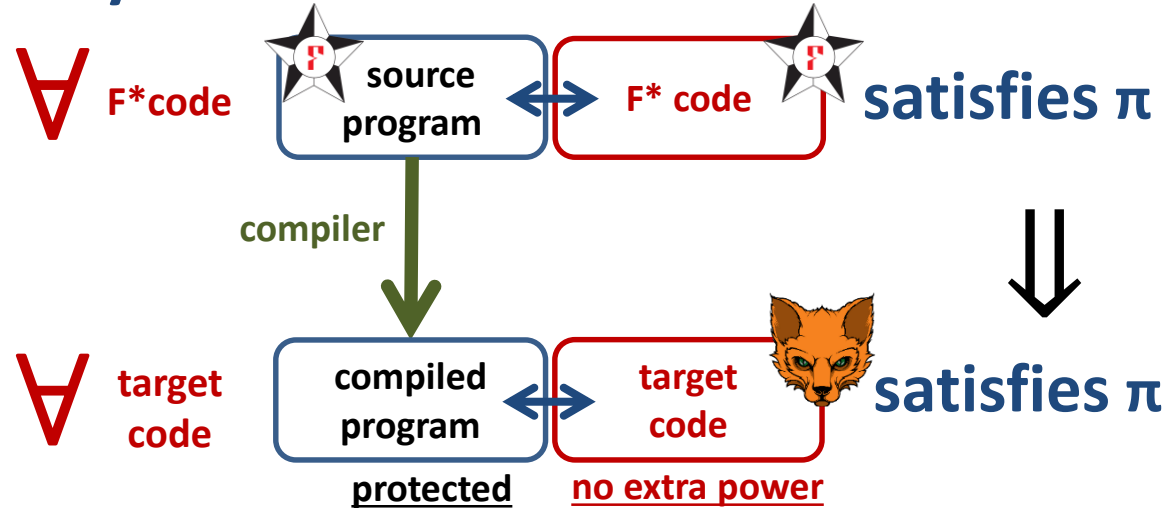


\forall security property π



Preserving security against adversarial contexts

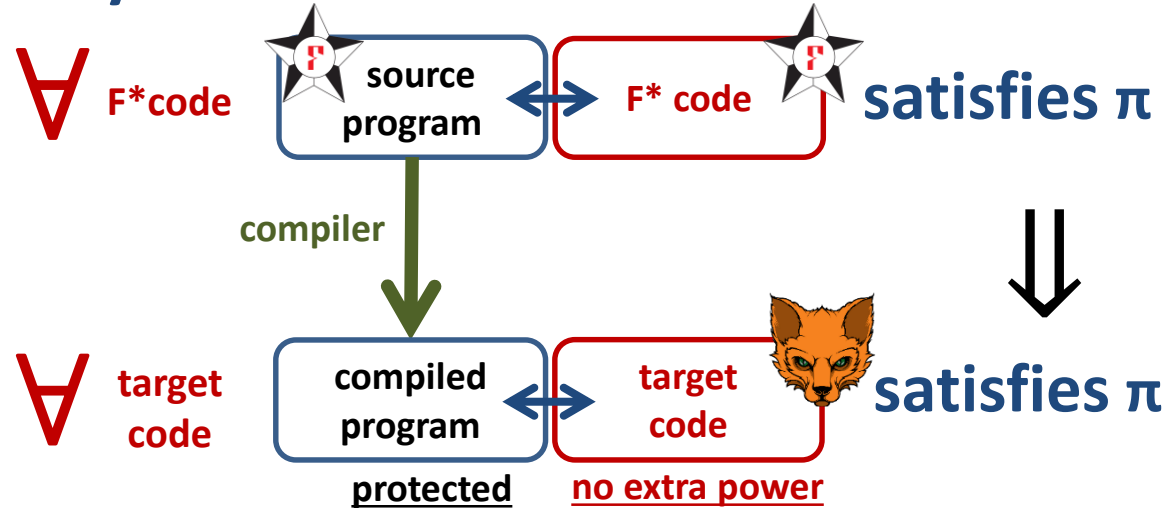
\forall security property π



Where π can e.g. be "the web server's private key is not leaked"

Preserving security against adversarial contexts

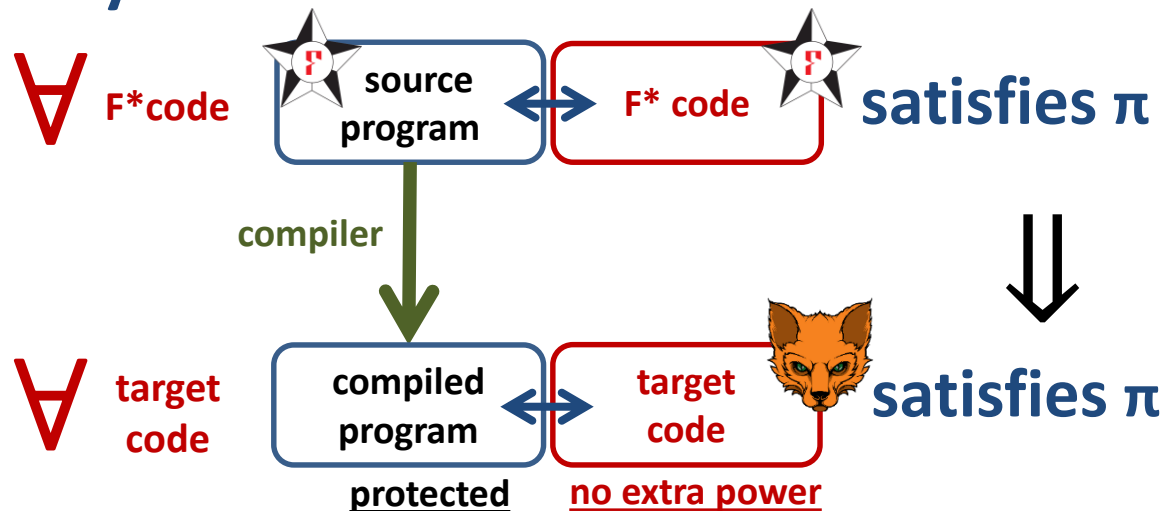
\forall security property π



Where π can e.g. be "the web server's private key is not leaked"

Preserving security against adversarial contexts

\forall security property π



Where π can e.g. be "the web server's private key is not leaked"

We explored many classes of properties one can preserve this way ...

Journey Beyond Full Abstraction [CSF'19, ESOP'20, TOPLAS'21]

Journey Beyond Full Abstraction [CSF'19, ESOP'20, TOPLAS'21]

trace properties
(safety & liveness)

Journey Beyond Full Abstraction [CSF'19, ESOP'20, TOPLAS'21]

hyperproperties
(noninterference)

trace properties
(safety & liveness)

Journey Beyond Full Abstraction [CSF'19, ESOP'20, TOPLAS'21]

**relational
hyperproperties**
(trace equivalence)

hyperproperties
(noninterference)

trace properties
(safety & liveness)

Journey Beyond Full Abstraction [CSF'19, ESOP'20, TOPLAS'21]

**relational
hyperproperties
(trace equivalence)**

Robust Relational Hyperproperty
Preservation (**RrHP**)

Robust K-Relational Hyperproperty
Preservation (**RKrHP**)

Robust 2-Relational Hyperproperty
Preservation (**R2rHP**)

Robust Relational
Property Preservation (**RrTP**)

Robust K-Relational
Property Preservation (**RKrTP**)

Robust 2-Relational
Property Preservation (**R2rTP**)

Robust Relational
XSafety Preservation (**RrSP**)

Robust Finite-Relational
XSafety Preservation (**RFrSC**)

Robust K-Relational
XSafety Preservation (**RKrSP**)

Robust 2-Relational
XSafety Preservation (**R2rSP**)

+ *determinacy*

*Robust Trace Equivalence
Preservation (RTEP)*

**hyperproperties
(noninterference)**

Robust Hyperproperty
Preservation (**RHP**)

Robust Subset-Closed Hyperproperty
Preservation (**RSCHC**)

Robust K-Subset-Closed Hyperproperty
Preservation (**RKSCHP**)

Robust 2-Subset-Closed Hyperproperty
Preservation (**R2SCHP**)

Robust Hypersafety
Preservation (**RHSC**)

Robust K-Hypersafety
Preservation (**RKHSP**)

Robust 2-Hypersafety
Preservation (**R2HSP**)

*Robust Termination-Insensitive
Noninterference Preservation
(RTINIP)*

**trace properties
(safety & liveness)**

Robust Trace Property
Preservation (**RTP**)

Robust Dense Property
Preservation (**RDP**)

Robust Safety Property
Preservation (**RSP**)

Journey Beyond Full Abstraction [CSF'19, ESOP'20, TOPLAS'21]

**relational
hyperproperties
(trace equivalence)**

Robust Relational Hyperproperty
Preservation (**RrHP**)

Robust K-Relational Hyperproperty
Preservation (**RKrHP**)

Robust 2-Relational Hyperproperty
Preservation (**R2rHP**)

Robust Hyperproperty
Preservation (**RHP**)

Robust Subset-Closed Hyperproperty
Preservation (**RSCHC**)

Robust K-Subset-Closed Hyperproperty
Preservation (**RKSCHP**)

Robust 2-Subset-Closed Hyperproperty
Preservation (**R2SCHP**)

Robust Trace Property
Preservation (**RTP**)

Robust Dense Property
Preservation (**RDP**)

**trace properties
(safety & liveness)**

Robust Relational
Property Preservation (**RrTP**)

Robust K-Relational
Property Preservation (**RKrTP**)

Robust 2-Relational
Property Preservation (**R2rTP**)

Robust Hypersafety
Preservation (**RHSC**)

Robust K-Hypersafety
Preservation (**RKHSP**)

Robust 2-Hypersafety
Preservation (**R2HSP**)

Robust Safety Property
Preservation (**RSP**)

No one-size-fits-all security criterion

More secure



Robust Relational
XSafety Preservation (**RrSP**)

Robust Finite-Relational
XSafety Preservation (**RfRSC**)

Robust K-Relational
XSafety Preservation (**RKrSP**)

Robust 2-Relational
XSafety Preservation (**R2rSP**)

+ determinacy

Robust Trace Equivalence
Preservation (**RTEP**)

Robust Termination-Insensitive
Noninterference Preservation
(**RTINIP**)

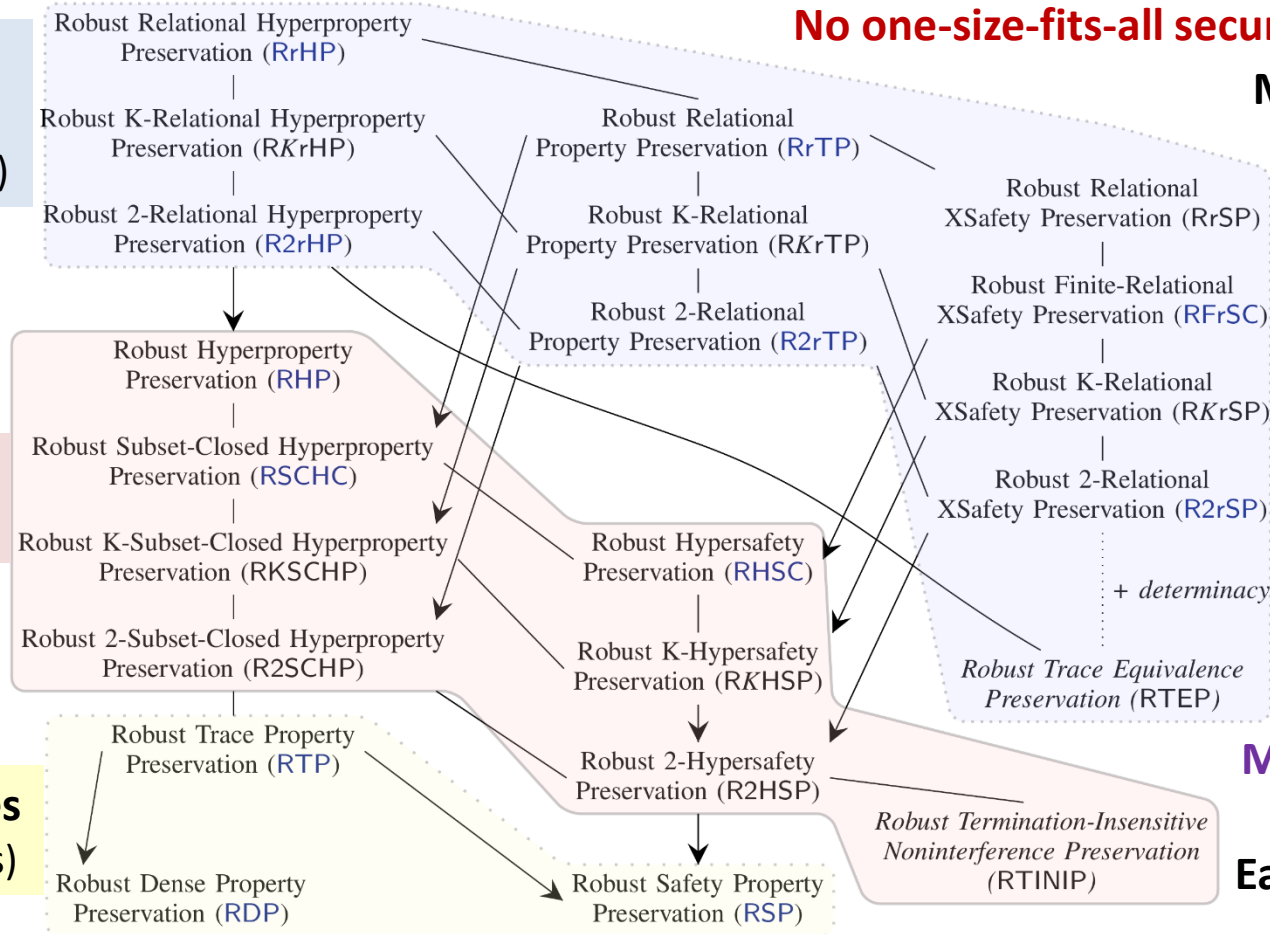
**More efficient
to enforce
Easier to prove**

Journey Beyond Full Abstraction [CSF'19, ESOP'20, TOPLAS'21]

**relational
hyperproperties**
(trace equivalence)

hyperproperties
(noninterference)

trace properties
(safety & liveness)
only integrity



No one-size-fits-all security criterion

More secure



**More efficient
to enforce
Easier to prove**

Journey Beyond Full Abstraction [CSF'19, ESOP'20, TOPLAS'21]

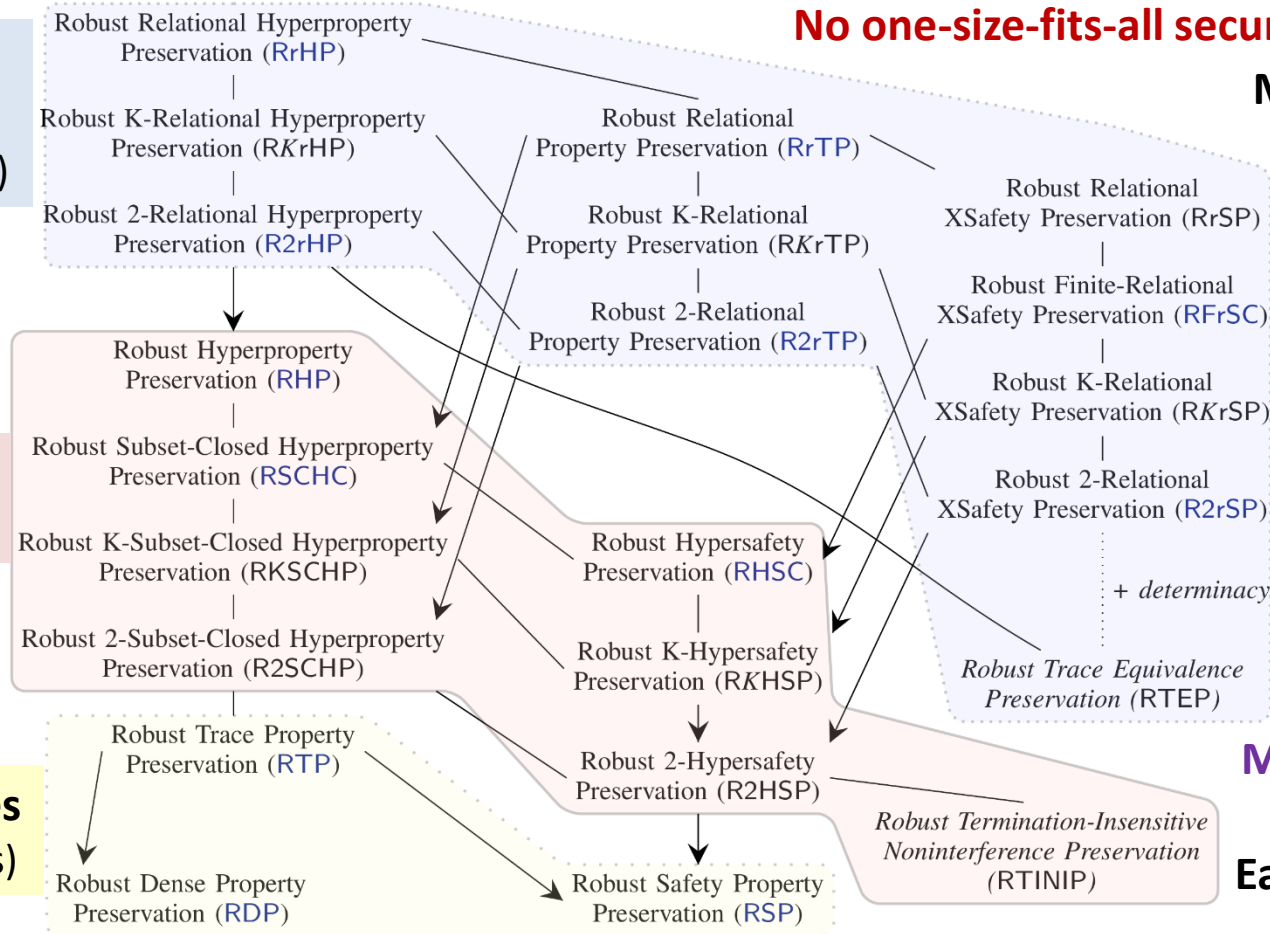
**relational
hyperproperties**
(trace equivalence)

hyperproperties
(noninterference)

+ data confidentiality

trace properties
(safety & liveness)

only integrity



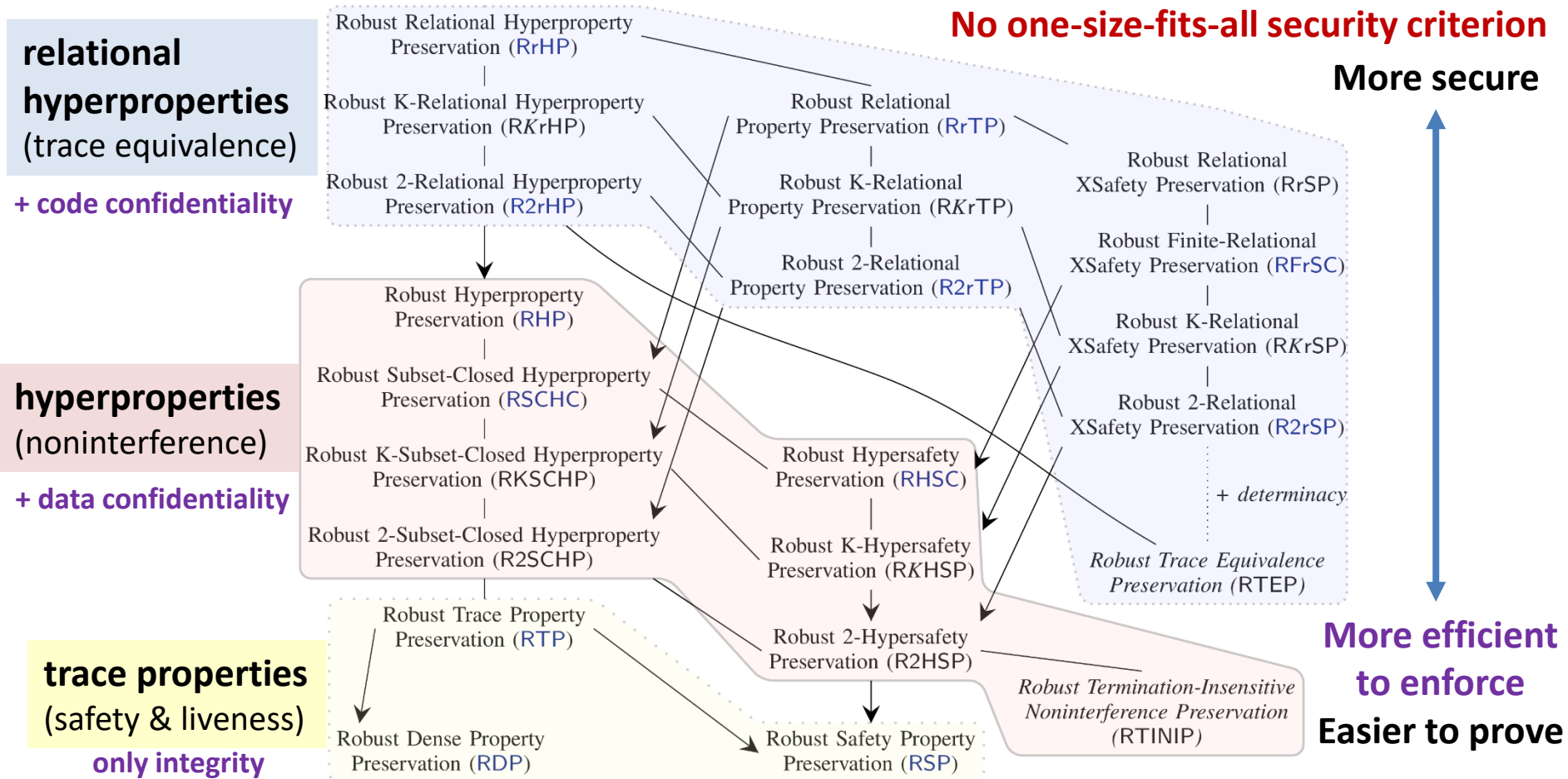
No one-size-fits-all security criterion

More secure

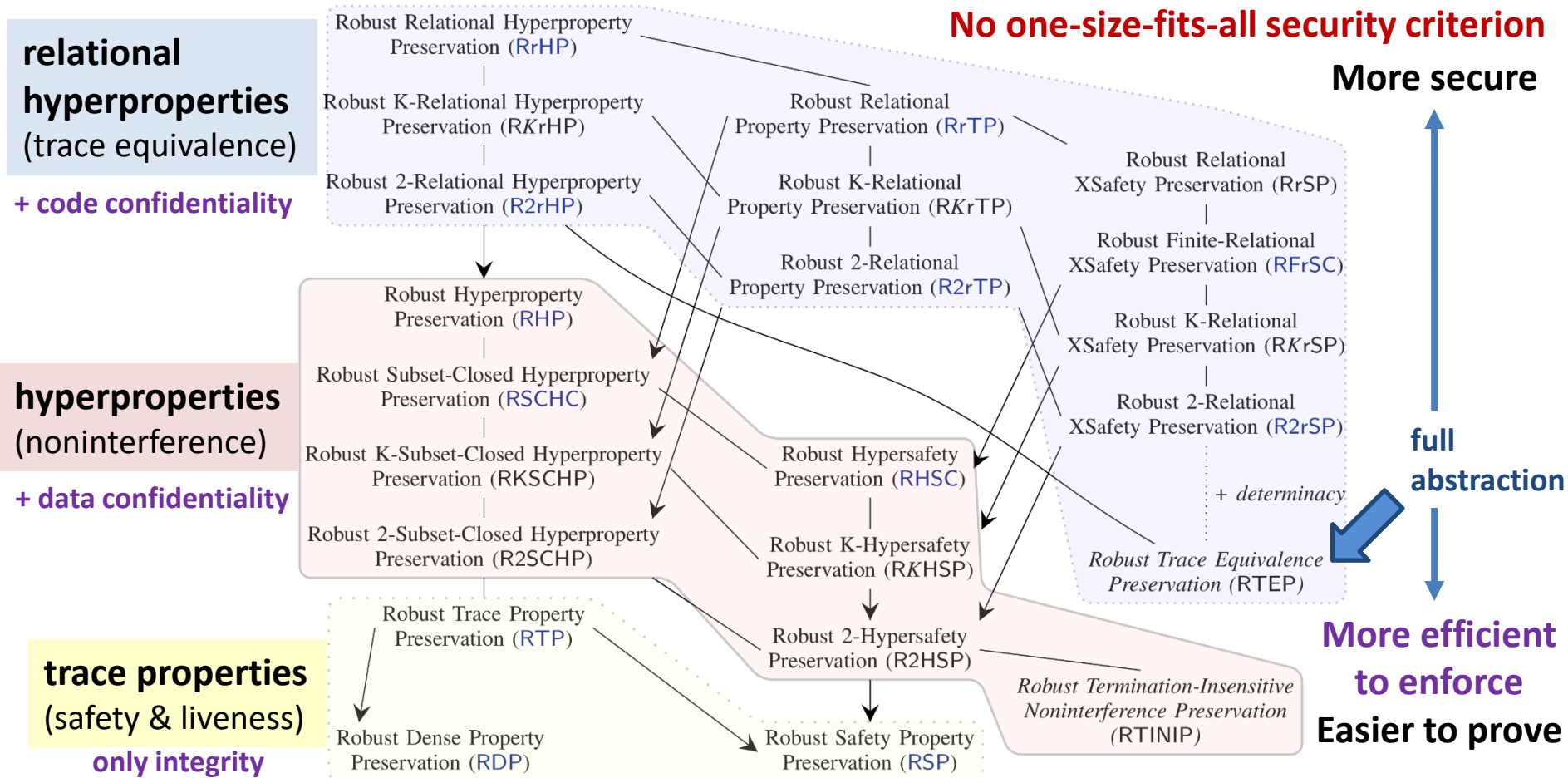


**More efficient
to enforce
Easier to prove**

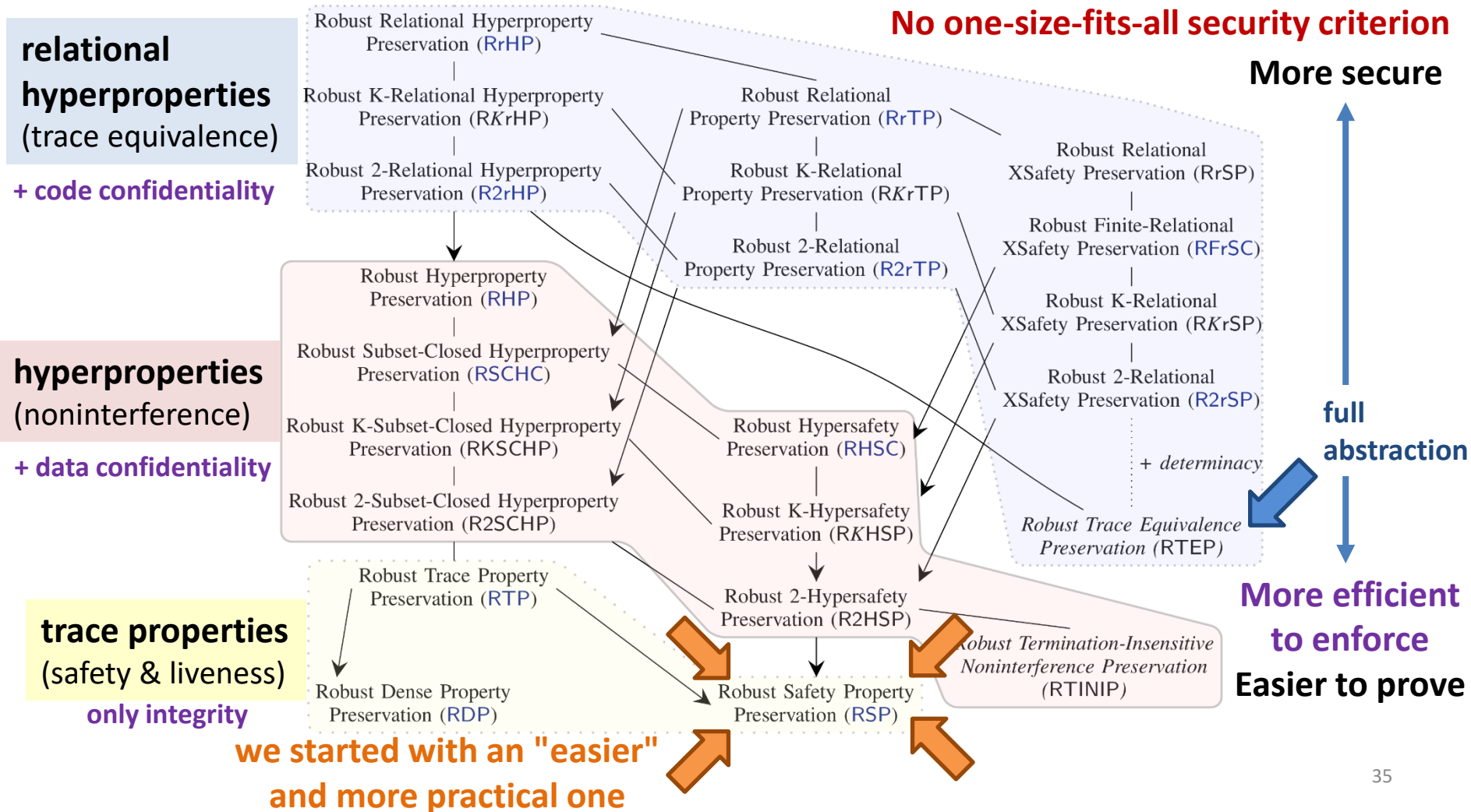
Journey Beyond Full Abstraction [CSF'19, ESOP'20, TOPLAS'21]



Journey Beyond Full Abstraction [CSF'19, ESOP'20, TOPLAS'21]



Journey Beyond Full Abstraction [CSF'19, ESOP'20, TOPLAS'21]





1. Security Goal

- Question A:

What does it mean to securely compile a secure source program **against linked adversarial target-level code**?



robust safety preservation



1. Security Goal

- Question A:

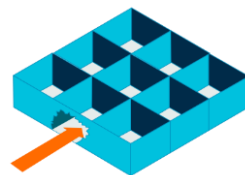
What does it mean to securely compile a secure source program **against linked adversarial target-level code**?



robust safety preservation

- Question B:

What does it mean for a compilation chain for vulnerable C compartments to be secure?





1. Security Goal

- Question A:

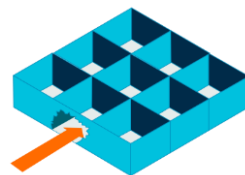
What does it mean to securely compile a secure source program **against linked adversarial target-level code**?



robust safety preservation

- Question B:

What does it mean for a compilation chain for vulnerable C compartments to be secure?



reduced this to a variant of robust safety preservation [CCS'18]

2. Security Enforcement



CompCert C
with compartments 



2. Security Enforcement



CompCert C
with compartments 



SECOMP: CompCert extended with secure compartments



2. Security Enforcement



CompCert C
with compartments 



CompCert RISC-V ASM
with compartments 

SECOMP: CompCert extended with secure compartments

magically secure semantics



2. Security Enforcement



CompCert C
with compartments 

SECOMP: CompCert extended with secure compartments

CompCert RISC-V ASM
with compartments 

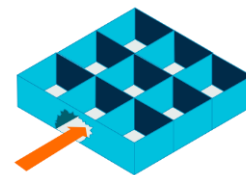
magically secure semantics



Software-Fault Isolation

vanilla ASM

2. Security Enforcement



CompCert C
with compartments 


SECOMP: CompCert extended with secure compartments

CompCert RISC-V ASM
with compartments 

magically secure semantics

Software-Fault Isolation

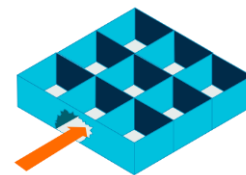
vanilla ASM

Micro-Policies: ASM
with programmable tags 

[POPL'14, S&P'15, ASPLOS'15,
POST'18, CCS'18, CSF'23]

Hardware-accelerated enforcement

2. Security Enforcement



CompCert C
with compartments 

SECOMP: CompCert extended with secure compartments


CompCert RISC-V ASM
with compartments 

magically secure semantics

Software-Fault Isolation

vanilla ASM

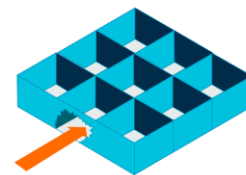
Done for simplified languages,
yet to be ported to RISC-V

Micro-Policies: ASM 
with programmable tags

[POPL'14, S&P'15, ASPLOS'15,
POST'18, CCS'18, CSF'23]

Hardware-accelerated enforcement

2. Security Enforcement



CompCert C
with compartments

SECOMP: CompCert extended with secure compartments

CompCert RISC-V ASM
with compartments

magically secure semantics

Software-Fault Isolation

vanilla ASM

Done for simplified languages,
yet to be ported to RISC-V

Micro-Policies: ASM
with programmable tags

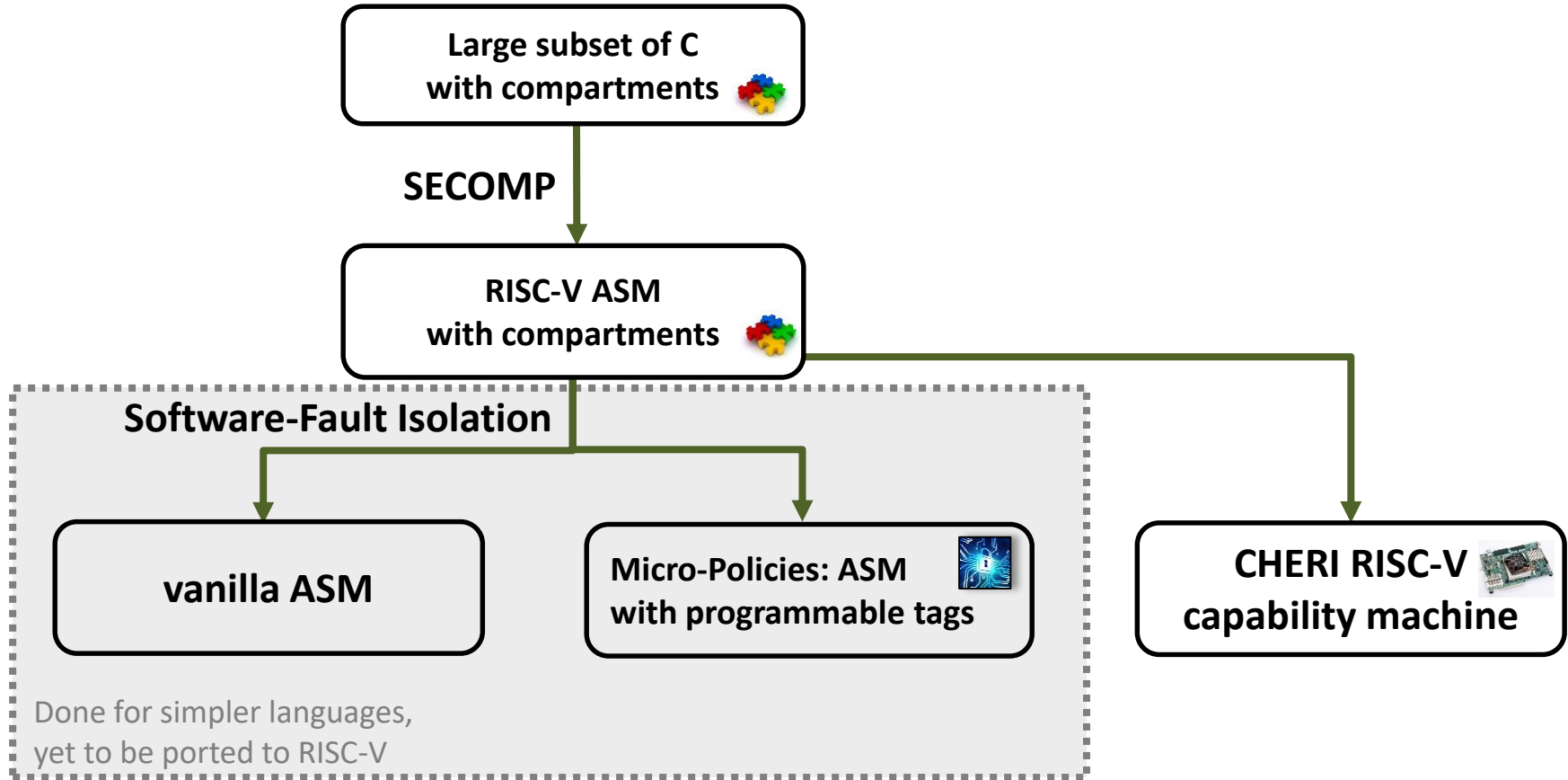
[POPL'14, S&P'15, ASPLOS'15,
POST'18, CCS'18, CSF'23]

CHERI RISC-V
capability machine

(inspiration for ARM Morello)

Hardware-accelerated enforcement

3. Secure Compilation Proofs in Rocq



3. Secure Compilation Proofs in Rocq

Machine-checked
proofs in Rocq



Large subset of C
with compartments



SECOMP

RISC-V ASM
with compartments



Software-Fault Isolation

vanilla ASM

Micro-Policies: ASM
with programmable tags



CHERI RISC-V
capability machine



Done for simpler languages,
yet to be ported to RISC-V

3. Secure Compilation Proofs in Rocq

Machine-checked proofs in Rocq



Large subset of C
with compartments



SECOMP

RISC-V ASM
with compartments



Scalable proof technique for secure compilation

- first applied to simpler languages [CCS'18, CSF'22]

Software-Fault Isolation

vanilla ASM

Micro-Policies: ASM
with programmable tags



CHERI RISC-V
capability machine



Done for simpler languages,
yet to be ported to RISC-V

3. Secure Compilation Proofs in Rocq

Machine-checked proofs in Rocq



Large subset of C
with compartments



SECOMP

RISC-V ASM
with compartments



Scalable proof technique for secure compilation

- first applied to simpler languages [CCS'18, CSF'22]
- then scaled up to C compartments [CCS'24]
 - reuses CompCert correctness proof (~130K LoC)
 - verified strong secure compilation property (+ ~43K LoC)

Software-Fault Isolation

vanilla ASM

Micro-Policies: ASM
with programmable tags



CHERI RISC-V
capability machine



Done for simpler languages,
yet to be ported to RISC-V

3. Secure Compilation Proofs in Rocq

Machine-checked proofs in Rocq



Large subset of C
with compartments



SECOMP

RISC-V ASM
with compartments



Scalable proof technique for secure compilation

- first applied to simpler languages [CCS'18, CSF'22]
- then scaled up to C compartments [CCS'24]
 - reuses CompCert correctness proof (~130K LoC)
 - verified strong secure compilation property (+ ~43K LoC)
- milestone in terms of realism!

Software-Fault Isolation

vanilla ASM

Micro-Policies: ASM
with programmable tags



CHERI RISC-V
capability machine



Done for simpler languages,
yet to be ported to RISC-V

3. Secure Compilation Proofs in Rocq

Machine-checked proofs in Rocq



Large subset of C
with compartments



SECOMP

RISC-V ASM
with compartments



Scalable proof technique for secure compilation

- first applied to simpler languages [CCS'18, CSF'22]
- then scaled up to C compartments [CCS'24]
 - reuses CompCert correctness proof (~130K LoC)
 - verified strong secure compilation property (+ ~43K LoC)
- milestone in terms of realism!
 - optimizing C compiler with 19 passes

Software-Fault Isolation

vanilla ASM

Micro-Policies: ASM
with programmable tags



CHERI RISC-V
capability machine



Done for simpler languages,
yet to be ported to RISC-V

3. Secure Compilation Proofs in Rocq

Machine-checked
proofs in Rocq



Large subset of C
with compartments



SECOMP

RISC-V ASM
with compartments



Scalable proof technique for secure compilation

- first applied to simpler languages [CCS'18, CSF'22]
- then scaled up to C compartments [CCS'24]
 - reuses CompCert correctness proof (~130K LoC)
 - verified strong secure compilation property (+ ~43K LoC)
- milestone in terms of realism!
 - optimizing C compiler with 19 passes

Software-Fault Isolation

vanilla ASM

Micro-Policies: ASM
with programmable tags



CHERI RISC-V
capability machine



Done for simpler languages,
yet to be ported to RISC-V



Systematic testing

3. Secure Compilation Proofs in Rocq

Machine-checked
proofs in Rocq



Large subset of C
with compartments



SECOMP

RISC-V ASM
with compartments



Scalable proof technique for secure compilation

- first applied to simpler languages [CCS'18, CSF'22]
- then scaled up to C compartments [CCS'24]
 - reuses CompCert correctness proof (~130K LoC)
 - verified strong secure compilation property (+ ~43K LoC)
- milestone in terms of realism!
 - optimizing C compiler with 19 passes

Software-Fault Isolation

vanilla ASM

Micro-Policies: ASM
with programmable tags



CHERI RISC-V
capability machine



Done for simpler languages,
yet to be ported to RISC-V



Big verification challenge for the future

Ongoing work: better proof techniques

Ongoing work: better proof techniques

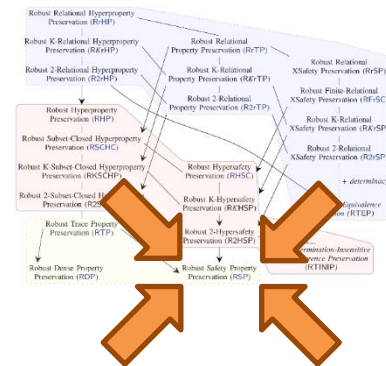
- **Verifying secure compilation for low-level backends**
 - they do the actual security enforcement, so very interesting
 - **such low-level proofs are conceptually very challenging though**

Ongoing work: better proof techniques

- **Verifying secure compilation for low-level backends**
 - they do the actual security enforcement, so very interesting
 - **such low-level proofs are conceptually very challenging though**
 - Basile Schlosser & Jérémy Thibault verifying micro-policies backend in a simplified setting; recently devised a **new proof technique** for this

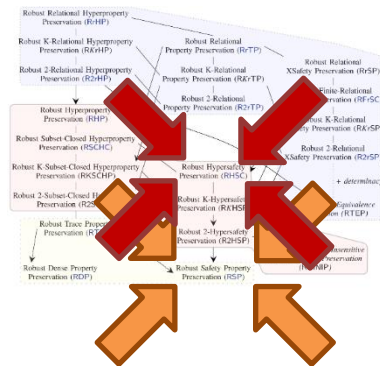
Ongoing work: better proof techniques

- **Verifying secure compilation for low-level backends**
 - they do the actual security enforcement, so very interesting
 - **such low-level proofs are conceptually very challenging though**
 - Basile Schlosser & Jérémy Thibault verifying micro-policies backend in a simplified setting; recently devised a **new proof technique** for this
- **Beyond preserving **safety** against adversarial contexts**



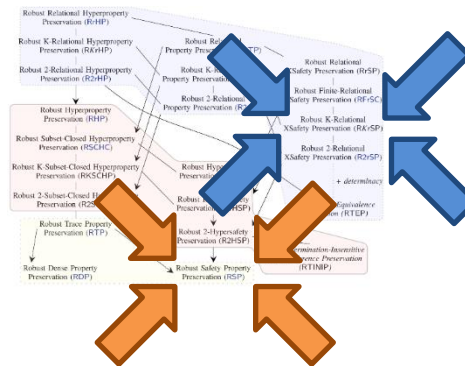
Ongoing work: better proof techniques

- Verifying secure compilation for low-level backends
 - they do the actual security enforcement, so very interesting
 - such low-level proofs are conceptually very challenging though
 - Basile Schlosser & Jérémy Thibault verifying micro-policies backend in a simplified setting; recently devised a **new proof technique** for this
- Beyond preserving **safety** against adversarial contexts
 - towards preserving **hyperproperties** (data confidentiality)



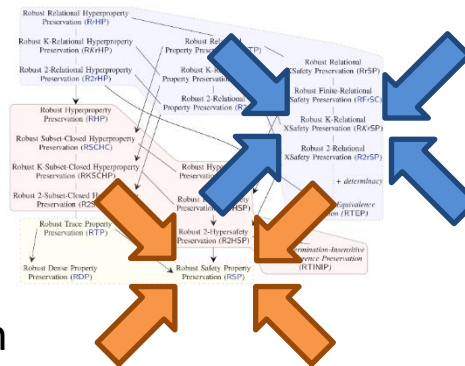
Ongoing work: better proof techniques

- **Verifying secure compilation for low-level backends**
 - they do the actual security enforcement, so very interesting
 - **such low-level proofs are conceptually very challenging though**
 - Basile Schlosser & Jérémy Thibault verifying micro-policies backend in a simplified setting; recently devised a **new proof technique** for this
- **Beyond preserving **safety** against adversarial contexts**
 - towards preserving **hyperproperties** (data confidentiality)
 - even **relational hyperproperties** (observational equivalence)



Ongoing work: better proof techniques

- **Verifying secure compilation for low-level backends**
 - they do the actual security enforcement, so very interesting
 - **such low-level proofs are conceptually very challenging though**
 - Basile Schlosser & Jérémy Thibault verifying micro-policies backend in a simplified setting; recently devised a **new proof technique** for this
- **Beyond preserving **safety** against adversarial contexts**
 - towards preserving **hyperproperties** (data confidentiality)
 - even **relational hyperproperties** (observational equivalence)
 - secure compilation criteria strictly stronger than full abstraction
 - can do this for CompCert, but won't hold for the backends
 - "Nanopass back-translation" [Jérémy Thibault et al, CSF'19, arXiv'25]



Enforcement beyond safety (challenging)

- Preserving **hypersafety** against adversarial contexts (e.g. data confidentiality)

Enforcement beyond safety (challenging)

- Preserving **hypersafety** against adversarial contexts (e.g. data confidentiality)
 - challenging at the lowest level: micro-architectural side-channels attacks



Enforcement beyond safety (challenging)

- Preserving **hypersafety** against adversarial contexts (e.g. data confidentiality)
 - challenging at the lowest level: micro-architectural side-channels attacks
 - compartments running in the same process, "universal read gadgets" easy



Enforcement beyond safety (challenging)

- Preserving **hypersafety** against adversarial contexts (e.g. data confidentiality)
 - challenging at the lowest level: micro-architectural side-channels attacks
 - compartments running in the same process, "universal read gadgets" easy
- Started looking into Spectre defenses compilers can insert



Enforcement beyond safety (challenging)

- Preserving **hypersafety** against adversarial contexts (e.g. data confidentiality)
 - challenging at the lowest level: micro-architectural side-channels attacks
 - compartments running in the same process, "universal read gadgets" easy
- Started looking into Spectre defenses compilers can insert
 - Speculative Load Hardening (implemented in LLVM + selective variant in Jasmin DSL)
 - enforces speculative constant time [Barthe et al, SP'23] (Security Foundations chapter)



Enforcement beyond safety (challenging)

- Preserving **hypersafety** against adversarial contexts (e.g. data confidentiality)
 - challenging at the lowest level: micro-architectural side-channels attacks
 - compartments running in the same process, "universal read gadgets" easy
- Started looking into Spectre defenses compilers can insert
 - Speculative Load Hardening (implemented in LLVM + selective variant in Jasmin DSL)
 - enforces speculative constant time [Barthe et al, SP'23] (Security Foundations chapter)
 - Rocq proofs that Ultimate SLH and our new Flexible SLH enforce relative security [CSF'25]



Enforcement beyond safety (challenging)

- Preserving **hypersafety** against adversarial contexts (e.g. data confidentiality)
 - challenging at the lowest level: micro-architectural side-channels attacks
 - compartments running in the same process, "universal read gadgets" easy
- Started looking into Spectre defenses compilers can insert
 - Speculative Load Hardening (implemented in LLVM + selective variant in Jasmin DSL)
 - enforces speculative constant time [Barthe et al, SP'23] (Security Foundations chapter)
 - Rocq proofs that Ultimate SLH and our new Flexible SLH enforce relative security [CSF'25]
 - Ongoing work: property-based testing for scaling this up to LLVM and x86/ARM

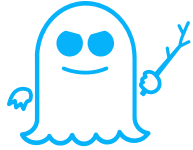


Enforcement beyond safety (challenging)

- Preserving **hypersafety** against adversarial contexts (e.g. data confidentiality)
 - challenging at the lowest level: micro-architectural side-channels attacks
 - compartments running in the same process, "universal read gadgets" easy
- Started looking into Spectre defenses compilers can insert
 - Speculative Load Hardening (implemented in LLVM + selective variant in Jasmin DSL)
 - enforces speculative constant time [Barthe et al, SP'23] (Security Foundations chapter)
 - Rocq proofs that Ultimate SLH and our new Flexible SLH enforce relative security [CSF'25]
 - Ongoing work: property-based testing for scaling this up to LLVM and x86/ARM
- Combining this with compartmentalization practically interesting
 - Especially for languages like Wasm, which are used for same-process isolation



Future Plans on Formally Secure Compilation



SPECTRE

Stronger Security Goals



**Preserve data confidentiality
against micro-architectural side-channel attacks,
for compartmentalized programs in F*, C, or Wasm**



Realistic Enforcement

**ARM Morello
capability machine**

Capability passing

Better Proof Techniques

Verify capability backend

