# Universal Composability is Robust Compilation

Marco Patrignani[1] Robert Künneman[2] Riad S. Wahby[3] Ethan Cecchetti[4]




January 2024 Toplas, journal first

1 UNITRENTO 2 CISPA HELMHOLTZ CENTER FOR INFORMATION SECURITY 3 Carnegie Mellon University 4 WISCONSIN UNIVERSITY OF WISCONSIN–MADISON

# DISCLAIMER

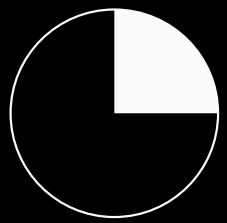
This film has been modified as follows from its original version: it has been formatted to fit your screen.

# DISCLAIMER

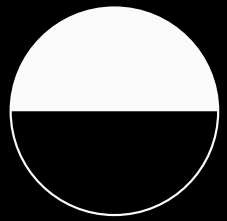
pl != crypto

# A Magic Trick

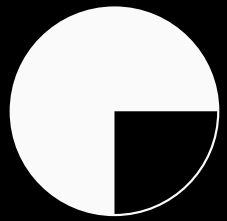
# A Magic Trick



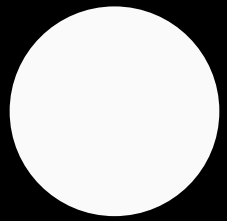
# A Magic Trick



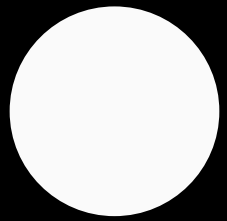
# A Magic Trick



# A Magic Trick



# A Magic Trick



Spoiler: there are $0 + \epsilon$ hands raised

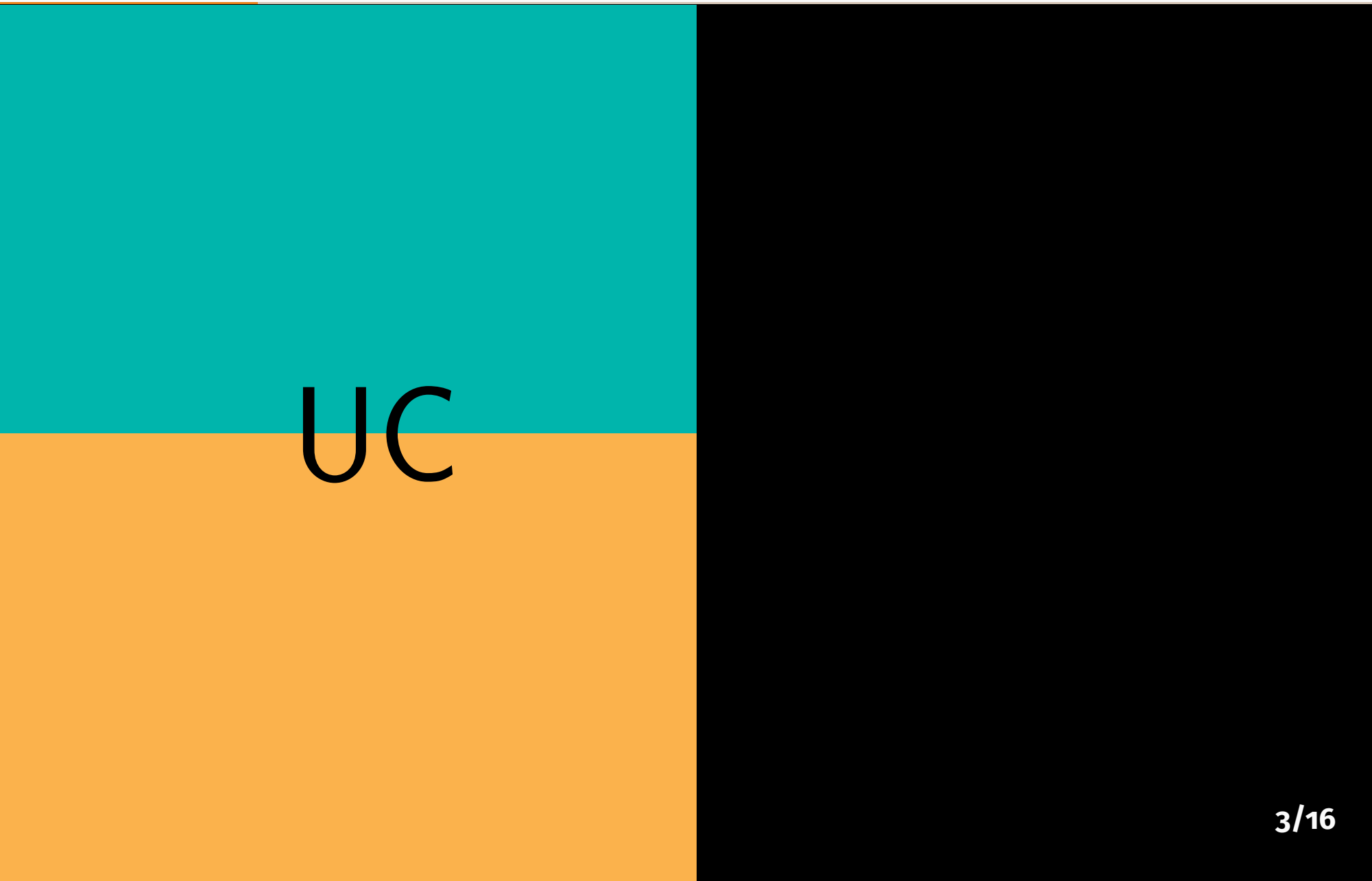
# Motivation and the Journey

# Motivation and the Journey

UC

$RC$

# **Fields:** UC



# **Universal Composability:** UC

- gold standard for proving security of protocols under concurrent composition

# **Universal Composability:** UC

- gold standard for proving security of protocols under concurrent composition
- overcomes security of protocol composition

# **Universal Composability:** UC

- gold standard for proving security of protocols under concurrent composition
- overcomes security of protocol composition
- many flavours: UC[1], SaUCy [2], iUC [3], . . .

[1] Canetti. 2001. “Universally composable security”

[2] Liao *et al.* 2019. “ILC: A Calculus for Composable, Computational Cryptography”

[3] Camenisch *et al.* 2019 “iUC: Flexible Universal Composability Made Simple”

# Universal Composability: UC

- gold standard for proving security of protocols under concurrent composition
- overcomes security of protocol composition
- many flavours: UC[1], SaUCy [2], iUC [3], . . .

This work: axiomatic formalisation, geared towards the newer theories SaUCy and iUC

---

[1] Canetti. 2001. "Universally composable security"

[2] Liao *et al.* 2019. "ILC: A Calculus for Composable, Computational Cryptography"

[3] Camenisch *et al.* 2019 "iUC: Flexible Universal Composability Made Simple"

# UC **Base Notions: ITMs** [4]

- protocols Π (using concrete crypto)

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

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

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

[4] From: Canetti, Fischlin. 2001. "Universally Composable Commitments"

# UC **Base Notions: ITMs** [4]

- protocols $\Pi$ (using concrete crypto)

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

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

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

- functionalities $\mathcal{F}$ (using abstract notions)

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

[4] From: Canetti, Fischlin. 2001. "Universally Composable Commitments"

# UC **Base Notions: ITMs** [4]

- protocols $\Pi$ (using concrete crypto)

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

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

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

- functionalities $\mathcal{F}$ (using abstract notions)

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

- attackers $\mathcal{A}$ & $\mathcal{S}$ (corrupting parties etc.)

[4] From: Canetti, Fischlin. 2001. “Universally Composable Commitments”

# UC **Base Notions: ITMs** [4]

- protocols Π (using concrete crypto)

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

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

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

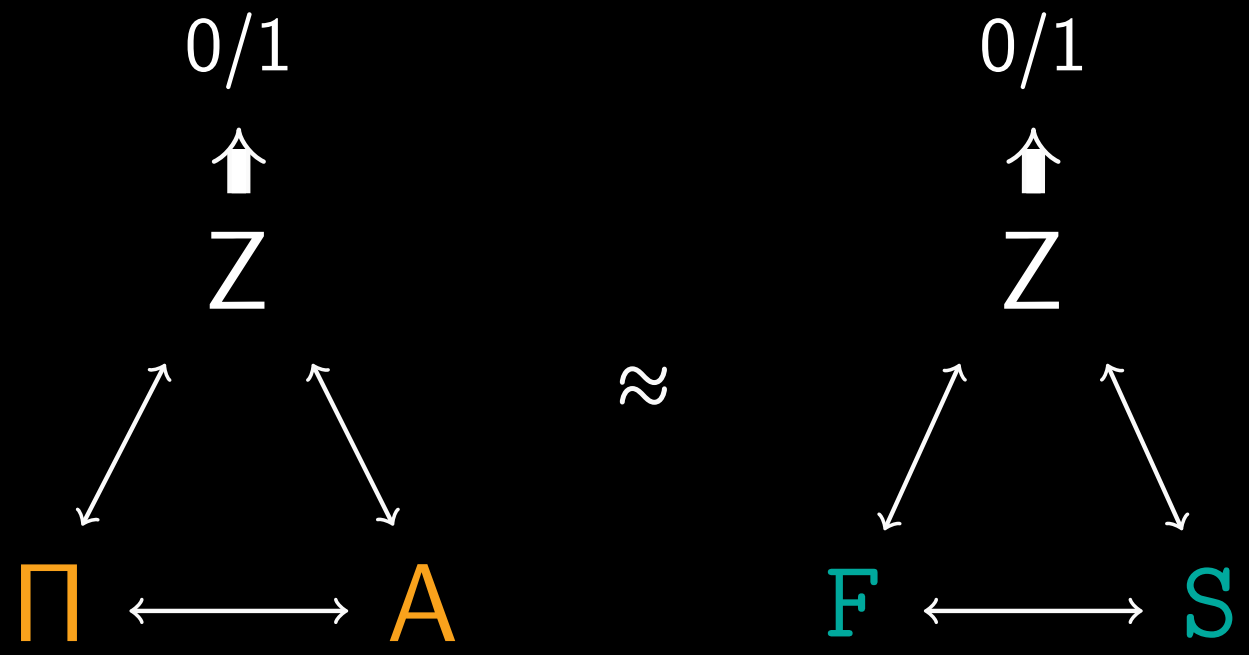
- functionalities F (using abstract notions)

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

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

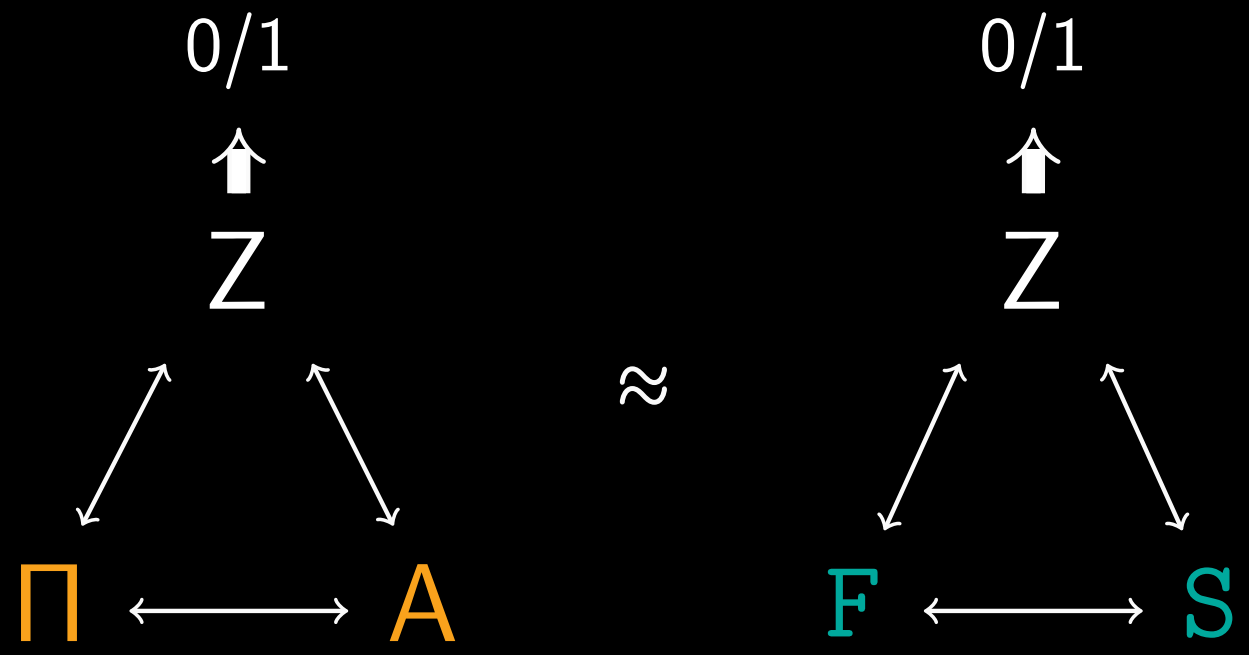
[4] From: Canetti, Fischlin. 2001. "Universally Composable Commitments"

# **Perfect (!!)** UC




↔ represent communication channels

# **Perfect (!!)** UC

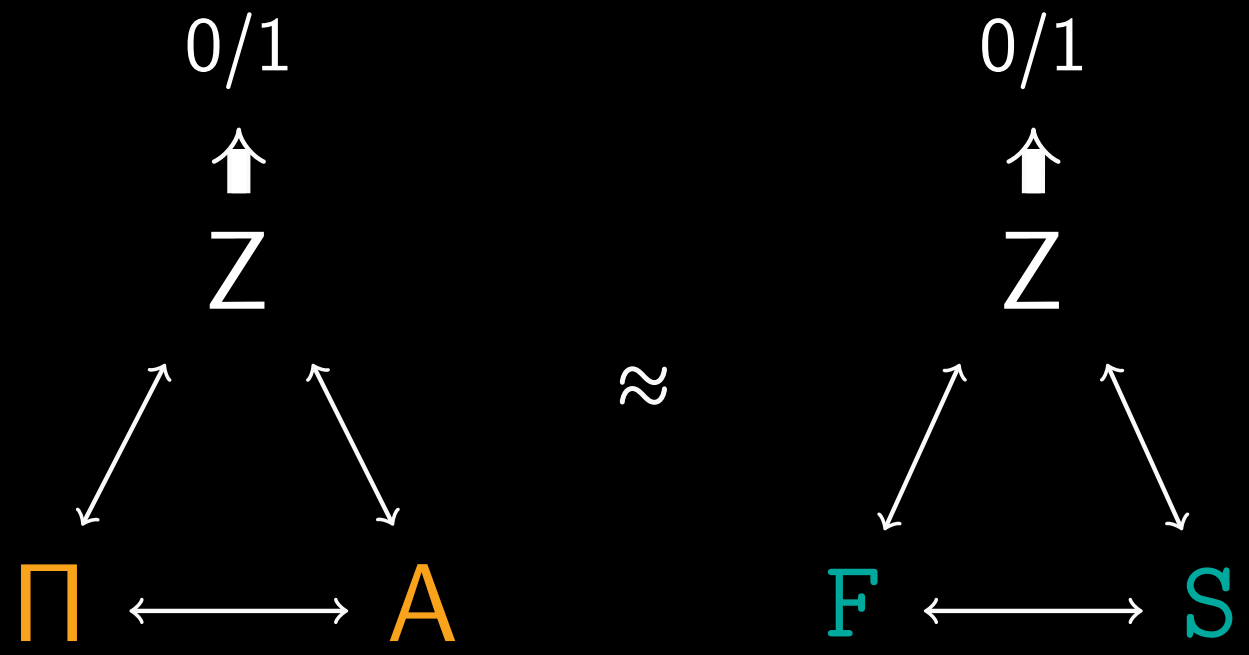



$\leftrightarrow$ represent communication channels

$$\Pi \vdash_{\mathrm{UC}} \mathtt{F} \stackrel{\text{def}}{=} \forall \mathsf{A}, \exists \mathtt{S}, \forall \mathsf{Z}.$$

$$\textsc{Exec}[\mathsf{Z}, \mathsf{A}, \Pi] \approx \textsc{Exec}[\mathsf{Z}, \mathtt{S}, \mathtt{F}]$$

# Perfect (!!) UC (computational UC in Künneman et al. CSF'24)

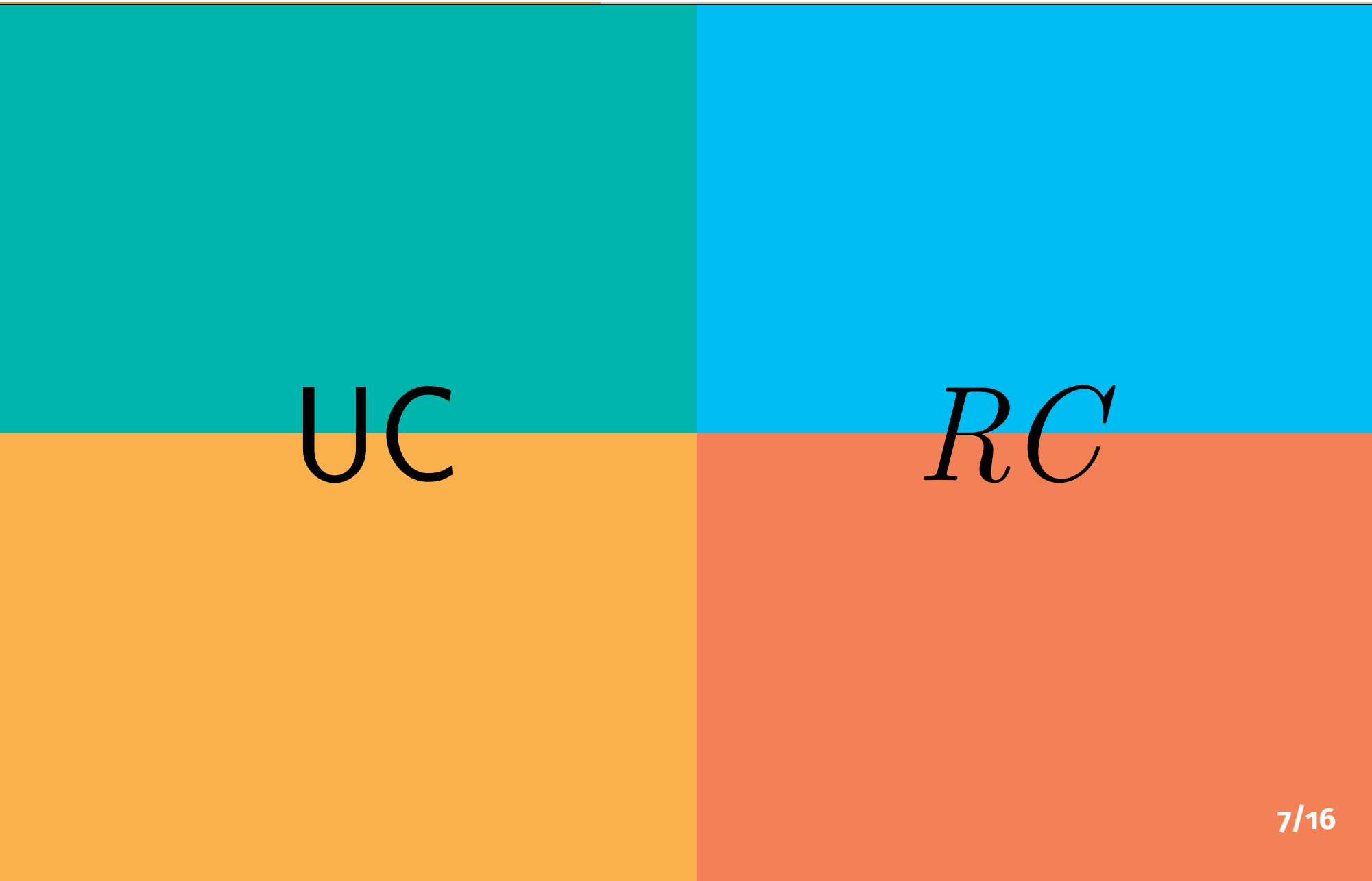



↔ represent communication channels

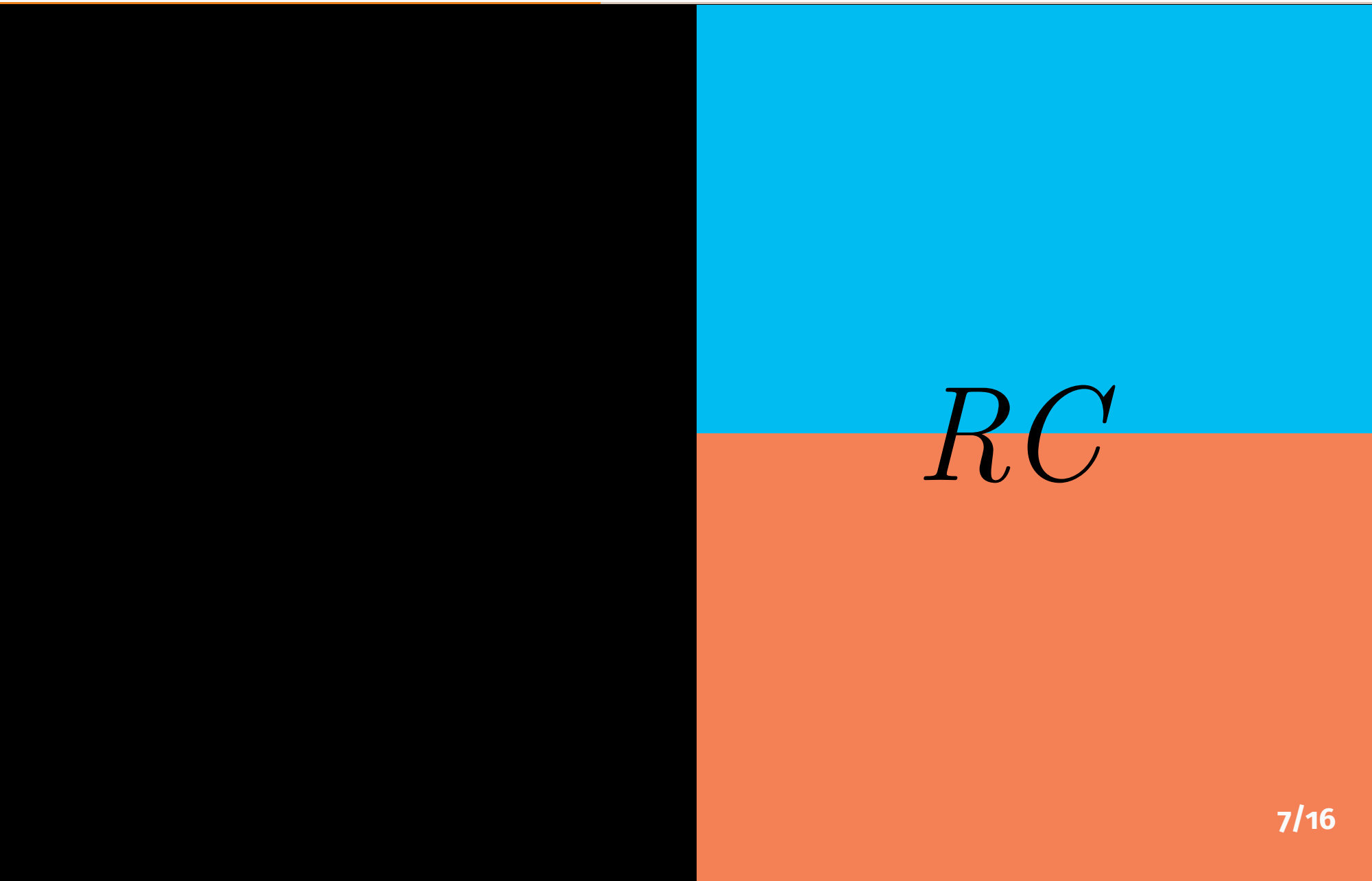
$$\Pi \vdash_{\mathrm{UC}} \mathtt{F} \stackrel{\mathrm{def}}{=} \forall \mathsf{A}, \exists \mathtt{S}, \forall \mathsf{Z}.$$

$$\textsc{Exec}[\mathsf{Z}, \mathsf{A}, \Pi] \approx \textsc{Exec}[\mathsf{Z}, \mathtt{S}, \mathtt{F}]$$

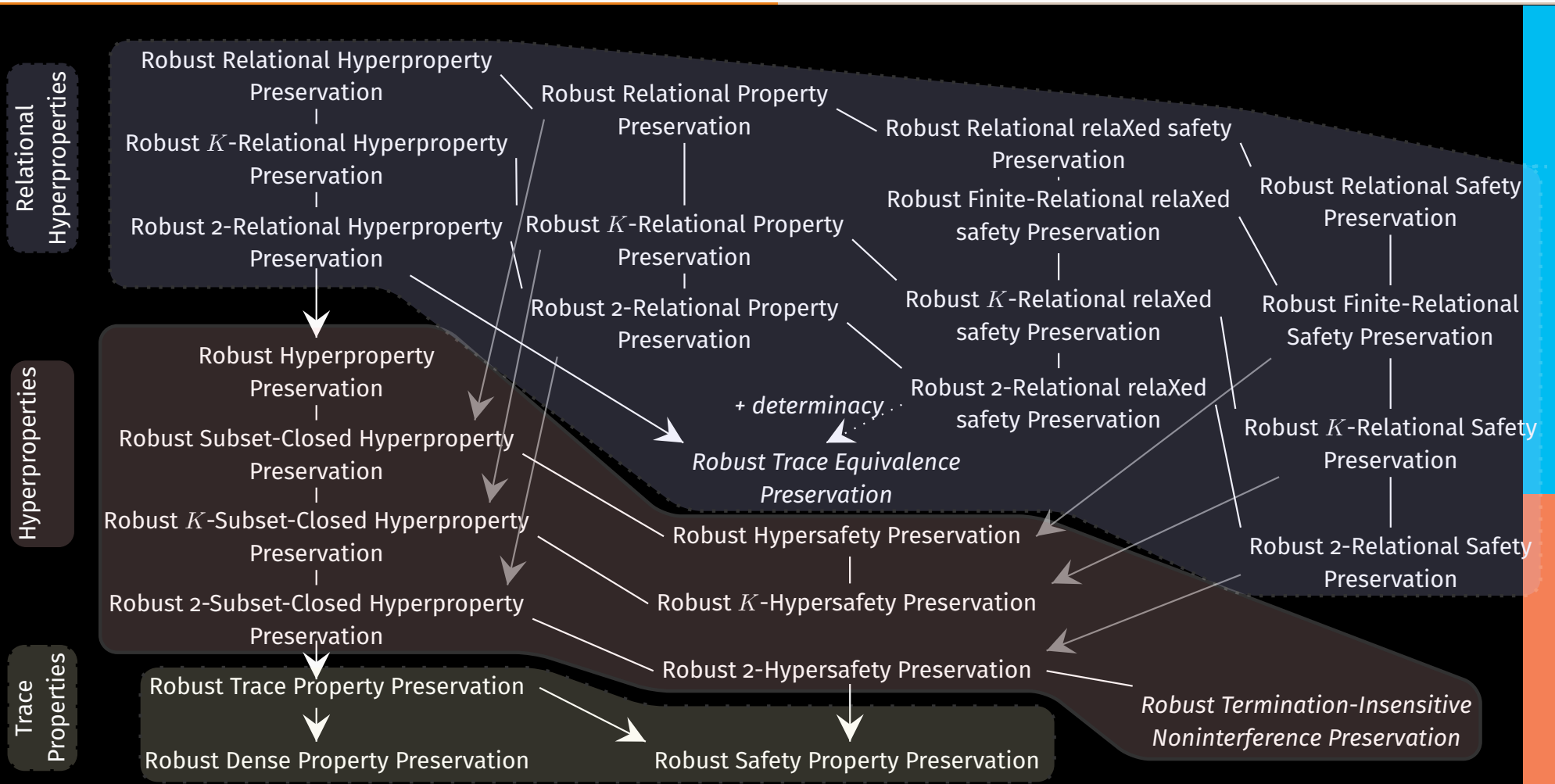
# Fields



# Fields: $RC$

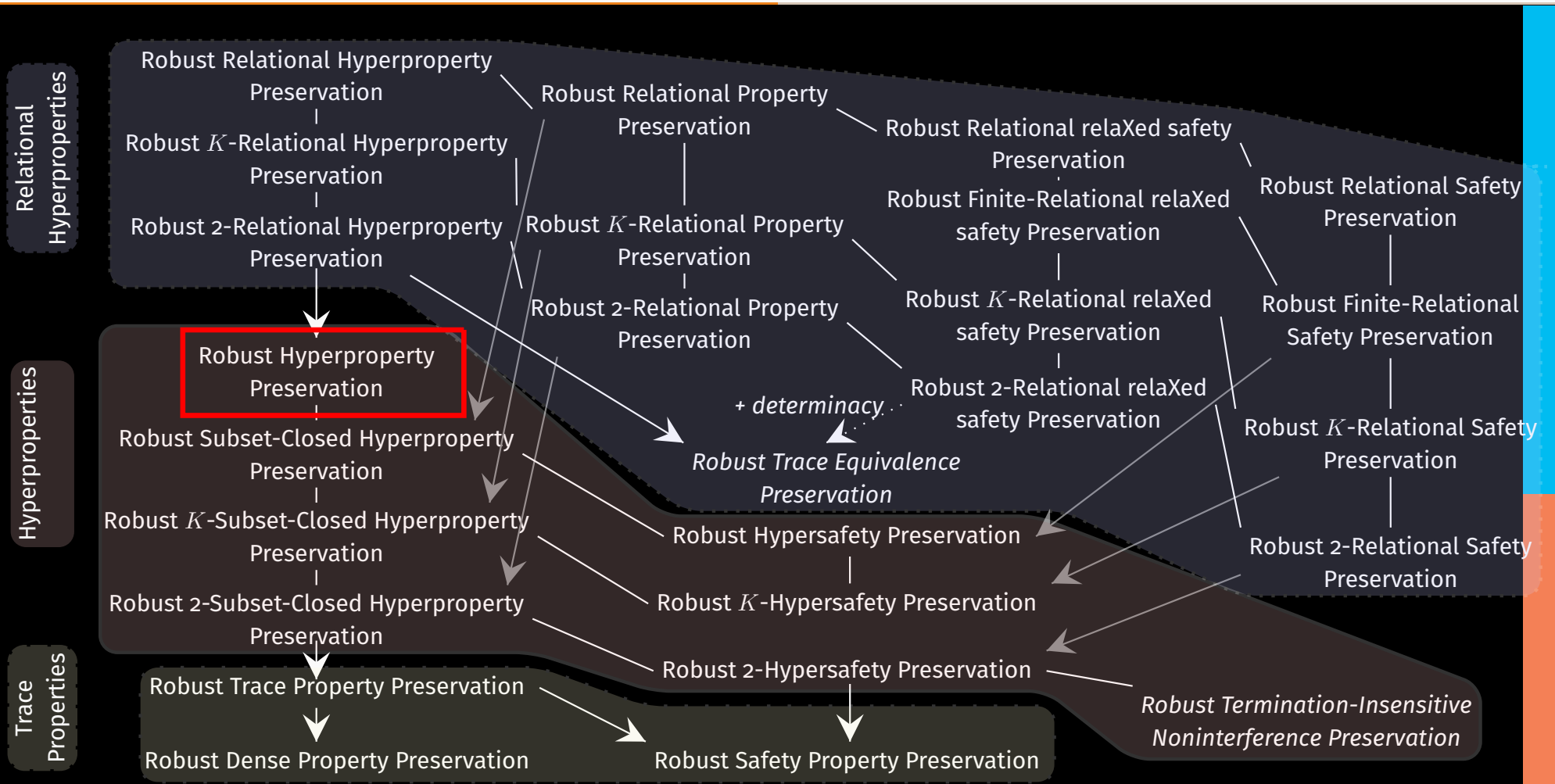


# Robust Compilation [5]




[5] Abate *et al.* 2019. "Journey Beyond Full Abstraction …"

# Robust Compilation [5]




[5] Abate *et al.* 2019. "Journey Beyond Full Abstraction …"

# Robust Hyperproperty Preservation: $RHC$

For any language $S$ and $\mathbf{T}$

$$\begin{array}{ccccc} & \overline{t} & & \overline{t} & \\ & \rightsquigarrow & & \rightsquigarrow & \\ [\![P]\!] & \bowtie & \mathbf{A} \iff P & \bowtie & A \end{array}$$

# Robust Hyperproperty Preservation: $RHC$

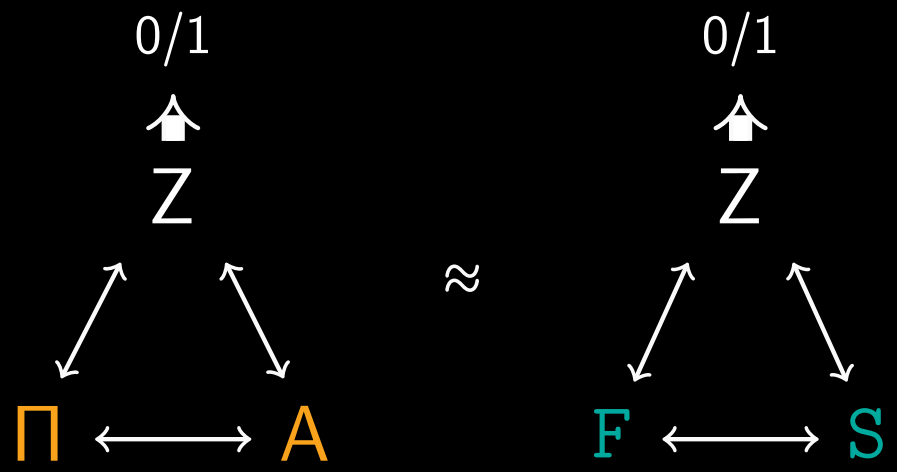
For any language $S$ and $\mathbf{T}$

$$\begin{array}{ccc} & \overline{t} & \\ & \rightsquigarrow & \\ [\![P]\!] & \bowtie \ \mathbf{A} & \iff P \ \bowtie \ A \end{array} \qquad \begin{array}{c} \overline{t} \\ \rightsquigarrow \\ \\ \end{array}$$

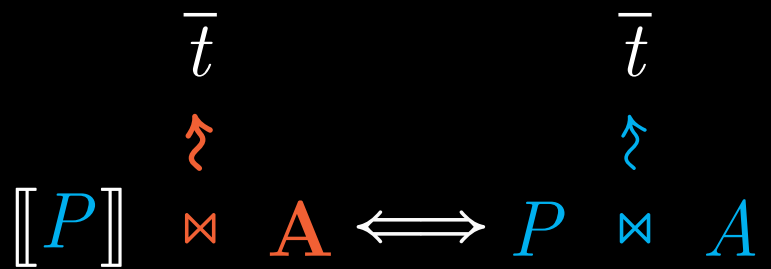
$$[\![\cdot]\!] \vdash RHC \stackrel{\text{def}}{=} \forall P, \mathbf{A}. \exists A. \forall \overline{t}.$$
$$\mathbf{A} \bowtie [\![P]\!] \rightsquigarrow \overline{t} \iff A \bowtie P \rightsquigarrow \overline{t}$$

# A Closer Look

$$\Pi \vdash_{\mathsf{UC}} \mathrm{F} \stackrel{\text{def}}{=} \forall \mathrm{A}, \exists \mathrm{S}, \forall \mathrm{Z}.$$




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



# Our (Isabelle'd) Connection

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

# Our (Isabelle'd) Connection

| UC | | $RC$ | |
|---|---|---|---|
| protocol | $\Pi$ | $[\![P]\!]$ | compiled program |
| concrete attacker | $\mathtt{A}$ | $\mathbf{A}$ | target context |
| ideal functionality | $\mathtt{F}$ | $P$ | source program |
| simulator | $\mathtt{S}$ | $A$ | source context |
| environment, output | $\mathsf{Z}$, 0/1 | $\overline{t}$, $\rightsquigarrow$ | trace, semantics |
| communication | $\leftrightarrow$ | $\bowtie$ | linking |
| probabilistic equiv. | $\approx$ | $\Longleftrightarrow$ | trace equality |
| human translation $\Pi \to \mathtt{F}$ | | $[\![\cdot]\!]\colon P \to \mathbf{P}$ compiler | |

# Why Should You Care?

Prove $RHC$ via UC

(e.g., Viaduct ...Acay et al PLDI'21)

# Why Should You Care?

Prove $RHC$ via UC

(e.g., Viaduct ...Acay et al PLDI'21)

Admittedly, less explored,

(is there more?)

# Why Should You Care?

# Why Should You Care?

Mechanise UC proofs with program analysis tools

(Deepsec, Cryptoverif, Squirrel, etc)

# Why Should You Care?

Mechanise UC proofs with program analysis tools

(Deepsec, Cryptoverif, Squirrel, etc)

as in computer-aided crypto

# How? The 1-bit Commitment Example

- Write protocol and `functionality` as Deepsec processes

# How? The 1-bit Commitment Example

- Write protocol and `functionality` as Deepsec processes
- Start building the missing `ideal processes` (90%) using:

# How? The 1-bit Commitment Example

- Write protocol and `functionality` as Deepsec processes
- Start building the missing `ideal processes` (90%) using:
  - backtranslation (from secure compilation)
  - and dummy attacker theorem (from this work)

# How? The 1-bit Commitment Example

- Write protocol and `functionality` as Deepsec processes
- Start building the missing `ideal processes` (90%) using:
  - backtranslation (from secure compilation)
  - and dummy attacker theorem (from this work)
- Fill the `missing lines` (4!)

# How? The 1-bit Commitment Example

- Write protocol and `functionality` as Deepsec processes
- Start building the missing `ideal processes` (90%) using:
  - backtranslation (from secure compilation)
  - and dummy attacker theorem (from this work)
- Fill the `missing lines` (4!)
- Wrap real and `ideal` processes with an environment proxy to regulate scheduling

# How? The 1-bit Commitment Example

- Write protocol and `functionality` as Deepsec processes
- Start building the missing `ideal processes` (90%) using:
  - backtranslation (from secure compilation)
  - and dummy attacker theorem (from this work)
- Fill the `missing lines` (4!)
- Wrap real and `ideal` processes with an environment proxy to regulate scheduling
- Add the missing `lines` for adaptive corruption (binding or hiding, not both)

# What Will You Find in the (64 pp) Paper?

1. Axiomatised UC semantics

# What Will You Find in the (64 pp) Paper?

1. Axiomatised UC semantics
2. Isabelle'd the connection

# What Will You Find in the (64 pp) Paper?

1. Axiomatised UC semantics
2. Isabelle'd the connection
3. Formalised conditions to use the connection with any language

# What Will You Find in the (64 pp) Paper?

1. Axiomatised UC semantics
2. Isabelle'd the connection
3. Formalised conditions to use the connection with any language
4. Formalised composition axioms

# What Will You Find in the (64 pp) Paper?

1. Axiomatised UC semantics
2. Isabelle'd the connection
3. Formalised conditions to use the connection with any language
4. Formalised composition axioms
5. Mechanised UC proof for 1-bit commitment for static & adaptive corruption

# What Will You Find in the (64 pp) Paper?

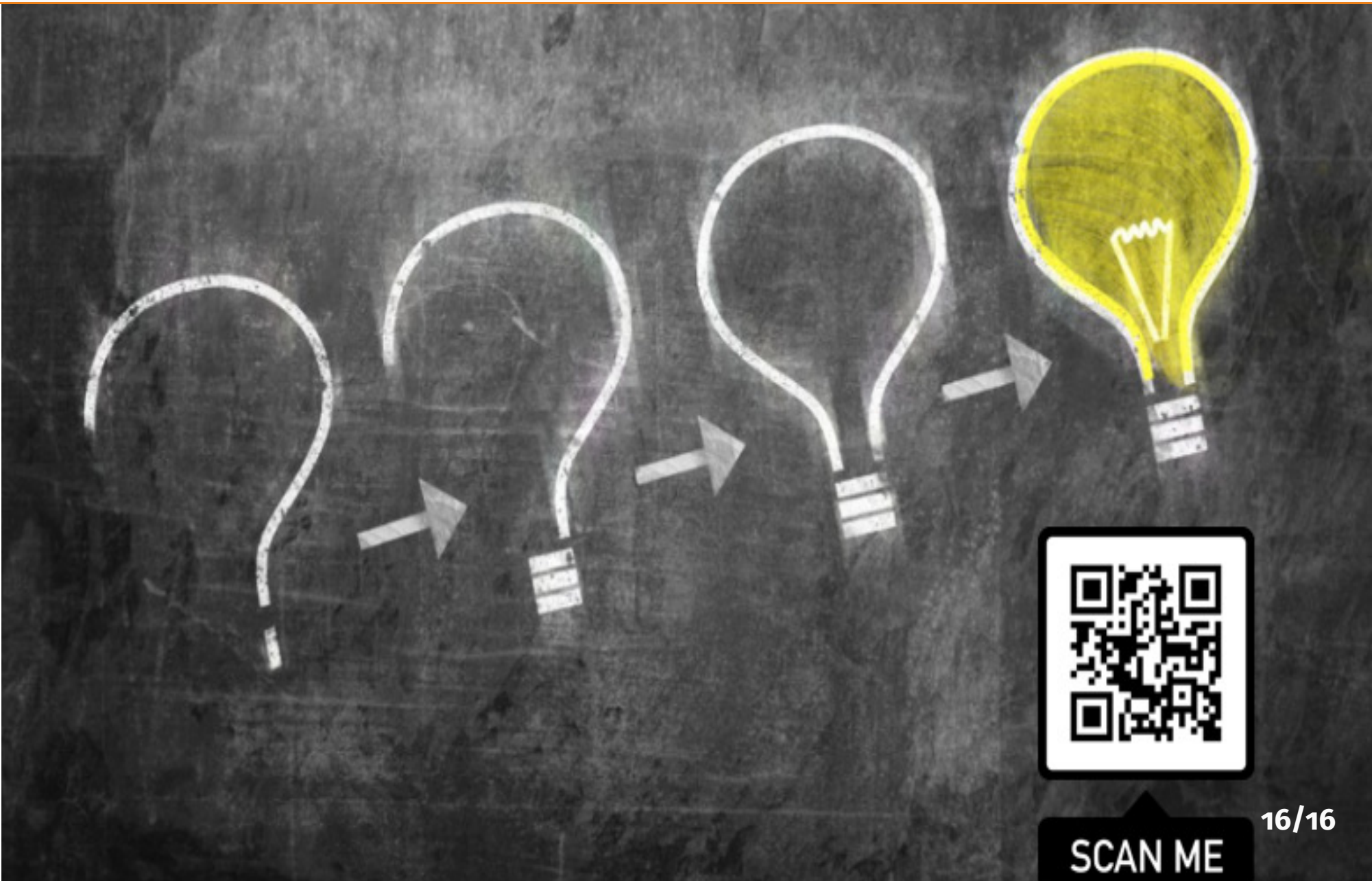
1. Axiomatised UC semantics
2. Isabelle'd the connection
3. Formalised conditions to use the connection with any language
4. Formalised composition axioms
5. Mechanised UC proof for 1-bit commitment for static & adaptive corruption
6. A lot of insights

# What Will You Find in the (64 pp) Paper?

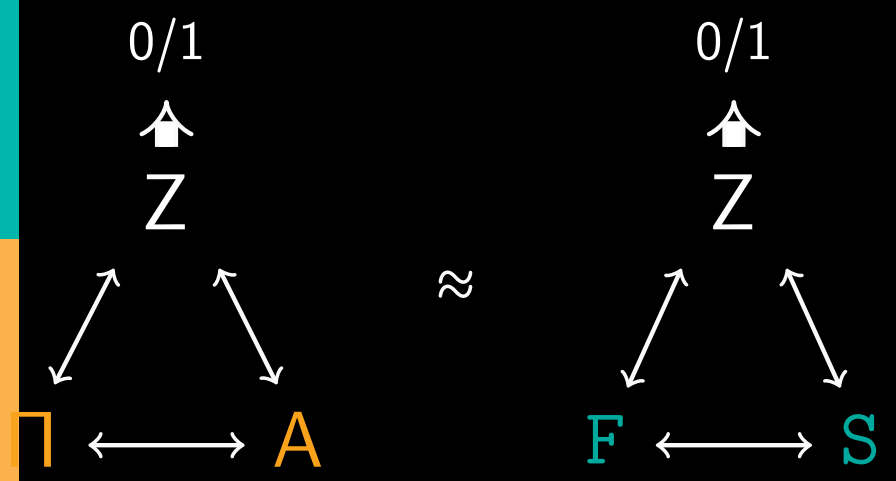
1. Axiomatised UC semantics
2. [illegible]
3. [illegible]
4. [illegible]
5. Mechanised UC proof for 1-bit commitment for static & adaptive corruption
6. A lot of insights

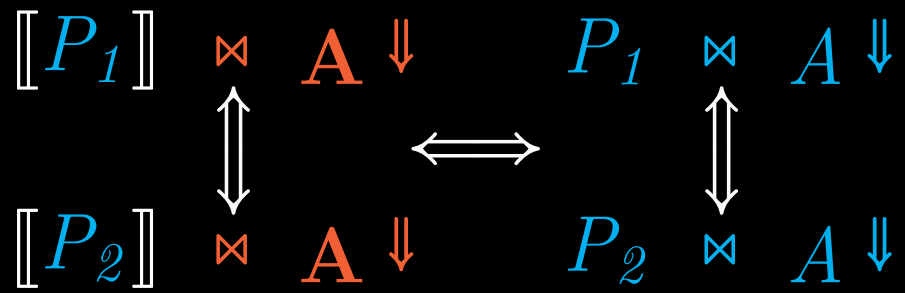
A security proof between ITMs is a UC proof,
a security proof between arbitrary languages is a RC one.

# Questions?



# The Full Abstraction (false) Conjecture




$FAC$ is relational, $RHC$ is propositional, like UC

# What is the Compiler?

- seemingly-degenerate (translate one concrete input to one concrete output)

# What is the Compiler?

- seemingly-degenerate (translate one concrete input to one concrete output)
- the connection works with any compiler!

# What is the Compiler?

- seemingly-degenerate (translate one concrete input to one concrete output)
- the connection works with any compiler!
- if only there were a protocol definition language ...(future work)

# Composition Operators

- Linking
- Program FFI
- Attacker FFI
- Complete FFI

# Composition Operators

- Linking
- Program FFI
- Attacker FFI
- Complete FFI
- Just program-level linking in any PL

# Composition Operators

- Linking
- Program FFI
- Attacker FFI
- Complete FFI
- Just program-level linking in any PL
- Must follow 3 (obvious) axioms

# The Dummy Attacker

- In UC, replace A with a dummy proxy

# The Dummy Attacker

- In UC, replace A with a dummy proxy
- 4 (obvious) Axioms provide the same theorem in $RC$

# The Dummy Attacker

- In UC, replace A with a dummy proxy
- 4 (obvious) Axioms provide the same theorem in $RC$
  thus, no need to do induction, just reason about the $source + simulator$ and $\mathbf{target}$ programs (with tools)