Compositional Compiler Correctness and Secure Compilation

Where we are & where we want to be

Amal Ahmed

Northeastern University & Inria Paris

Compiler Verification since 2006…

Leroy '06 : Formal certification of a compiler back-end or: programming a compiler with a proof assistant.

Lochbihler '10 : Verifying a compiler for Java threads.

Myreen '10 : Verified just-in-time compiler on x86.

Sevcik et al.'11 : Relaxed-memory concurrency and verified compilation.

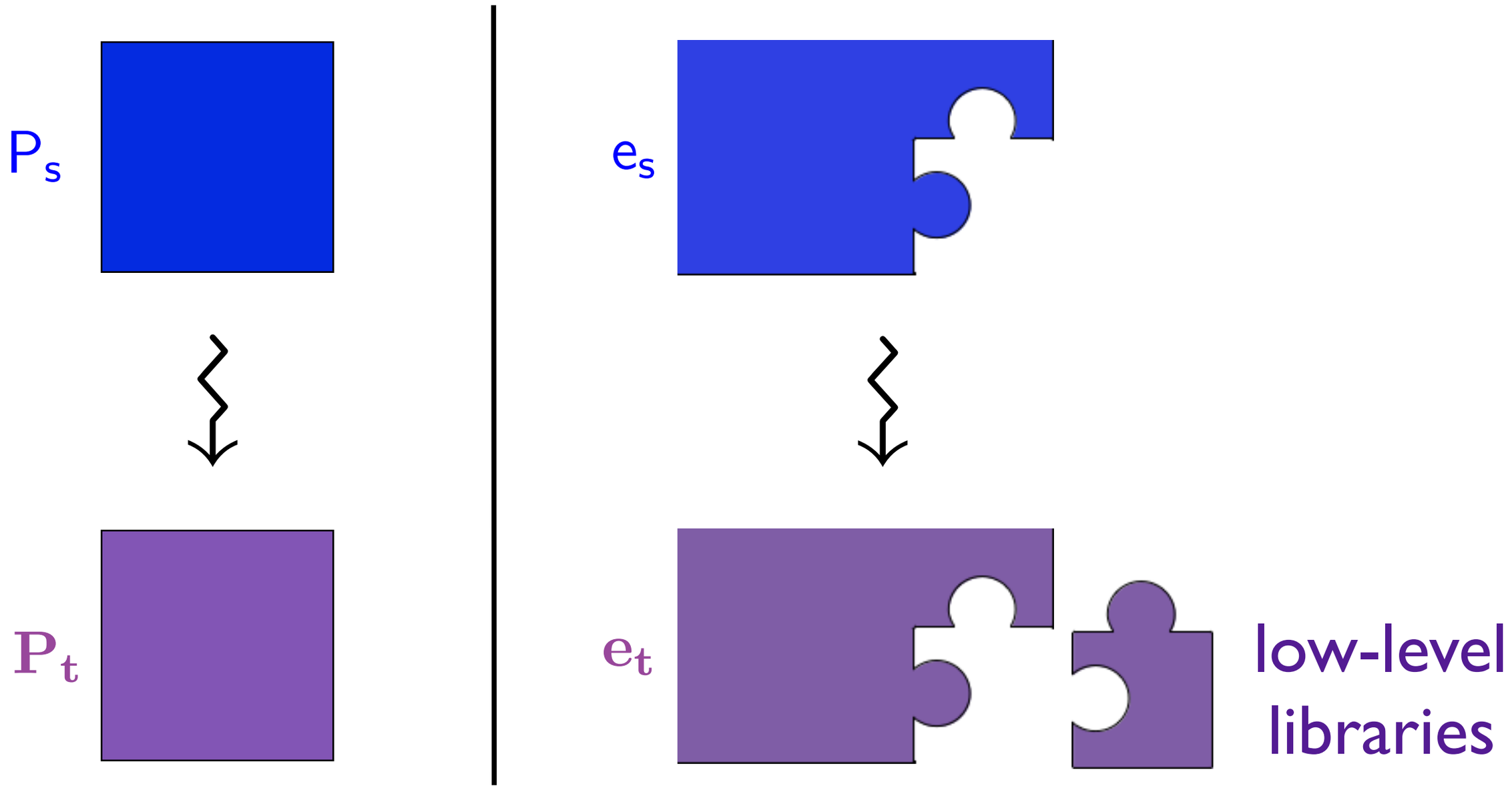
Zhao et al.'13 : Formal verification of SSA-based optimizations for LLVM

Kumar et al.'14 : CakeML: A verified implementation of ML

⋮

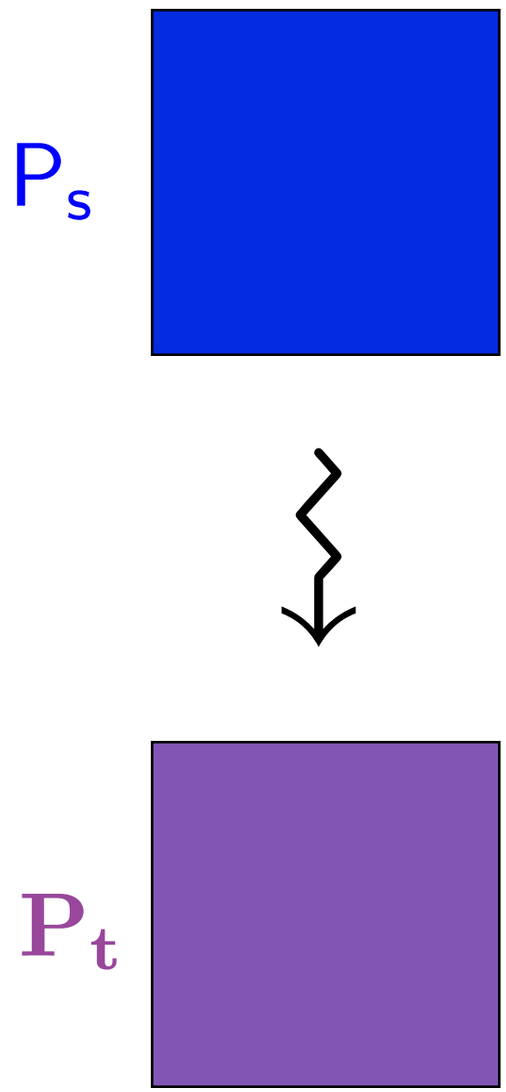
Problem: Whole-Program Assumption

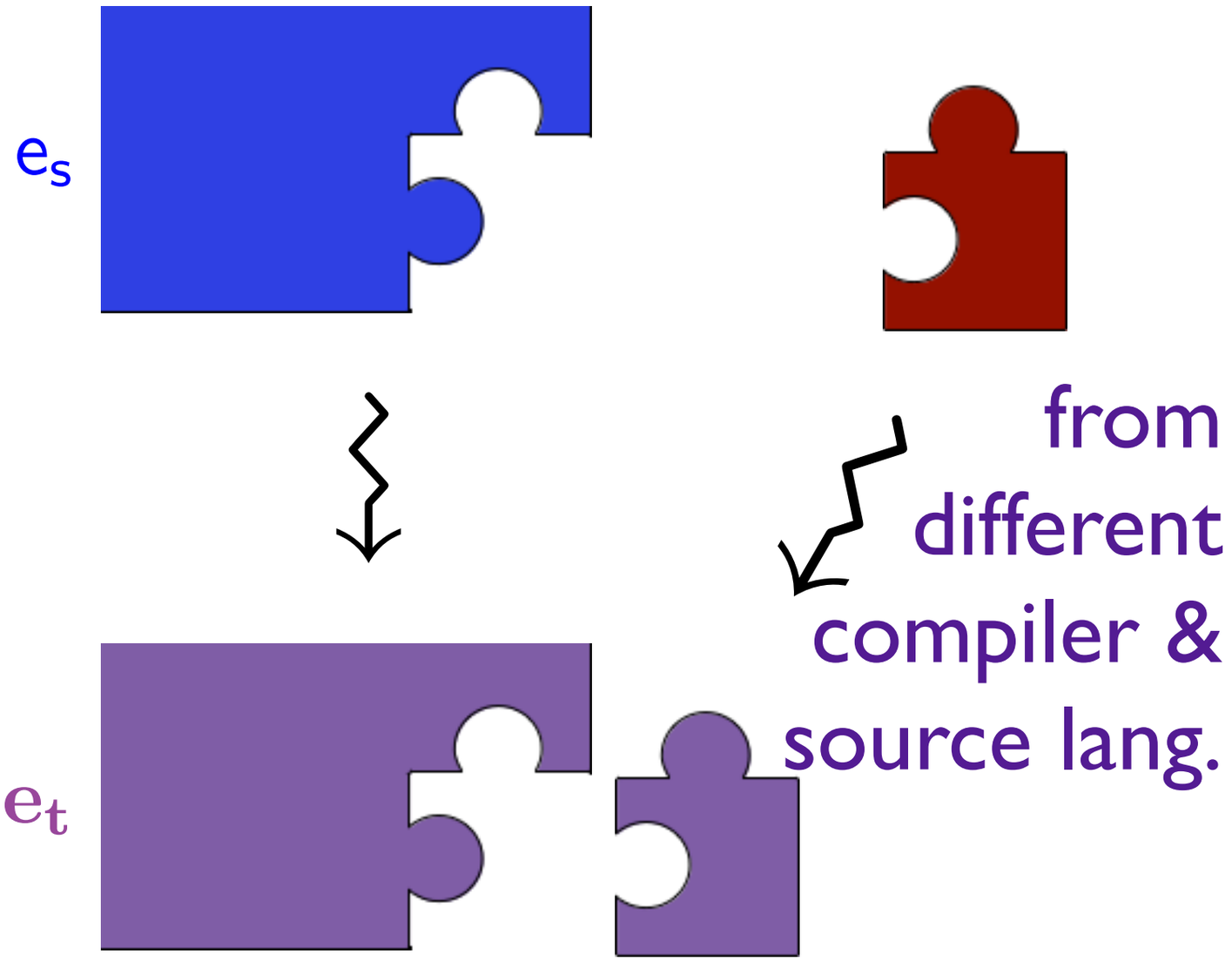
Correct compilation guarantee only applies to whole programs!



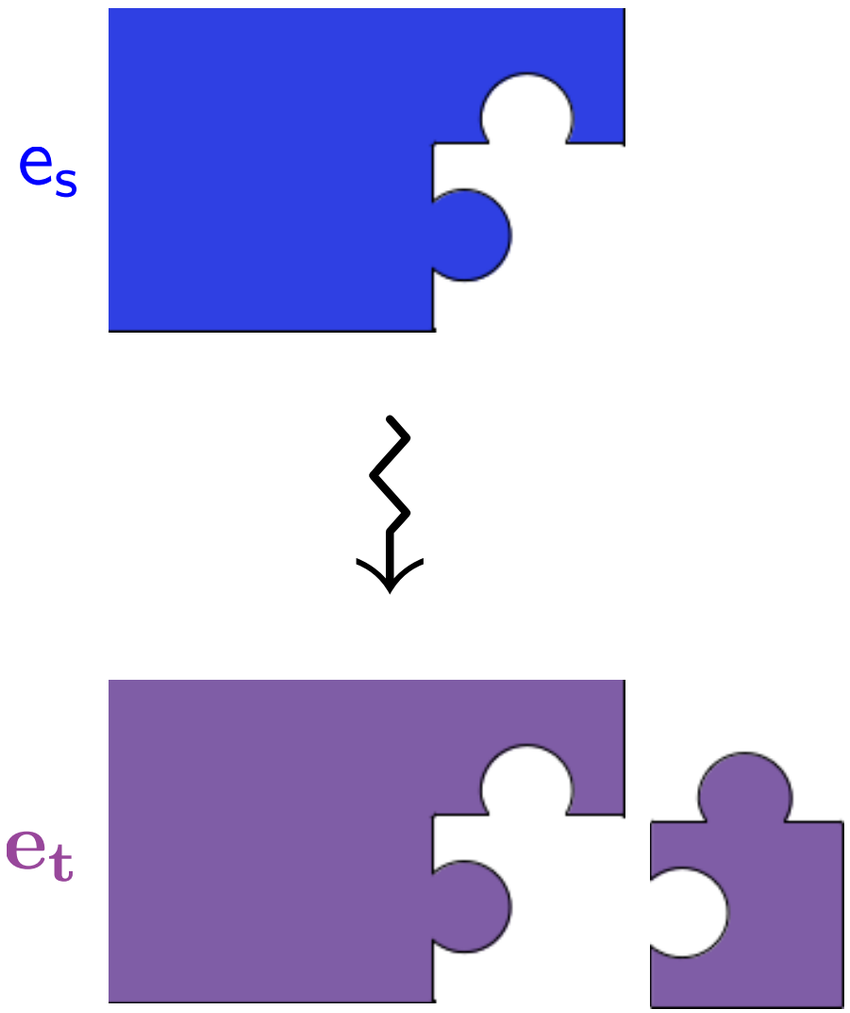
Problem: Whole-Program Assumption

Correct compilation guarantee only applies to whole programs!





“Compositional” Compiler Verification



This Talk…

- why specifying compositional compiler correctness is hard
- survey recent results
- generic CCC theorem and lessons for formalizing linking & verifying multi-pass compilers

Compiler Correctness

$$s \rightsquigarrow t \implies s \approx t$$

compiles to

behave the same

Compiler Correctness

$$s \rightsquigarrow t \implies s \approx t$$

expressed how?

Whole-Program Compiler Correctness

$$P_s \rightsquigarrow P_t \implies P_t \sqsubset P_s$$

behavior refinement

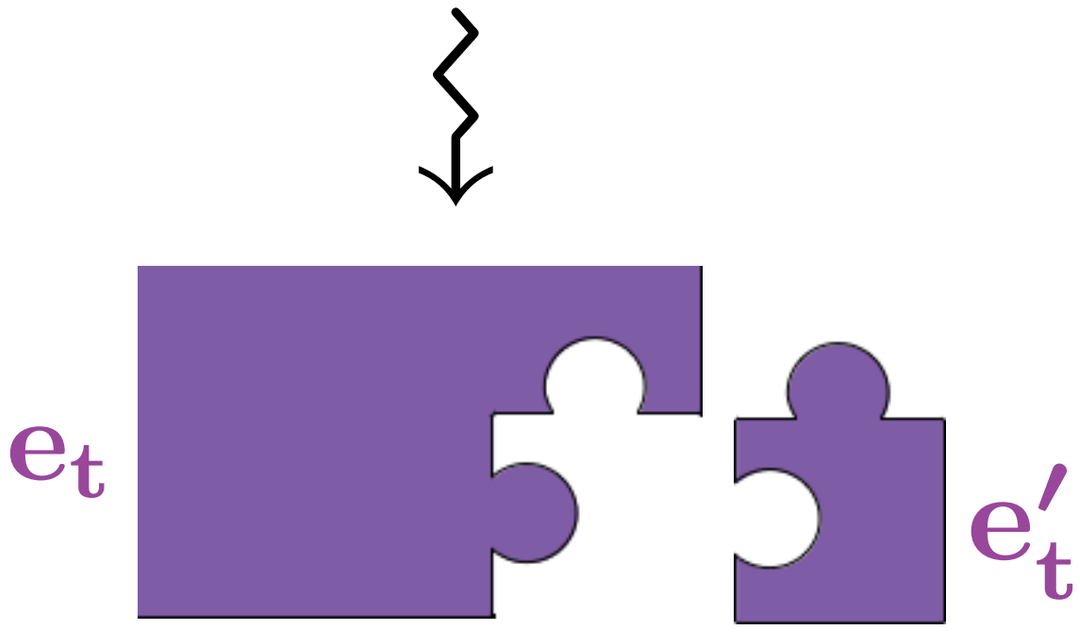
$$\forall n.\ \ P_t \overset{T_t}{\longmapsto}{}^{n}\ P_t' \implies$$

$$\exists m.\ \ P_s \overset{T_s}{\longmapsto}{}^{m}\ P_s' \ \ \wedge\ \ T_t \simeq T_s$$

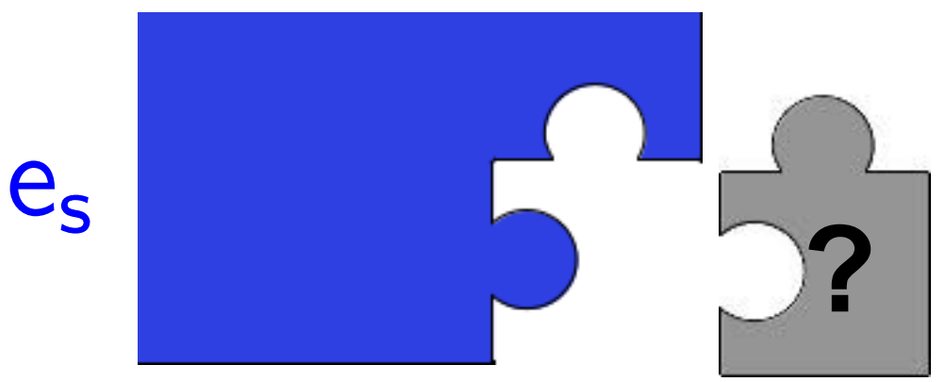
Correct Compilation of Components?

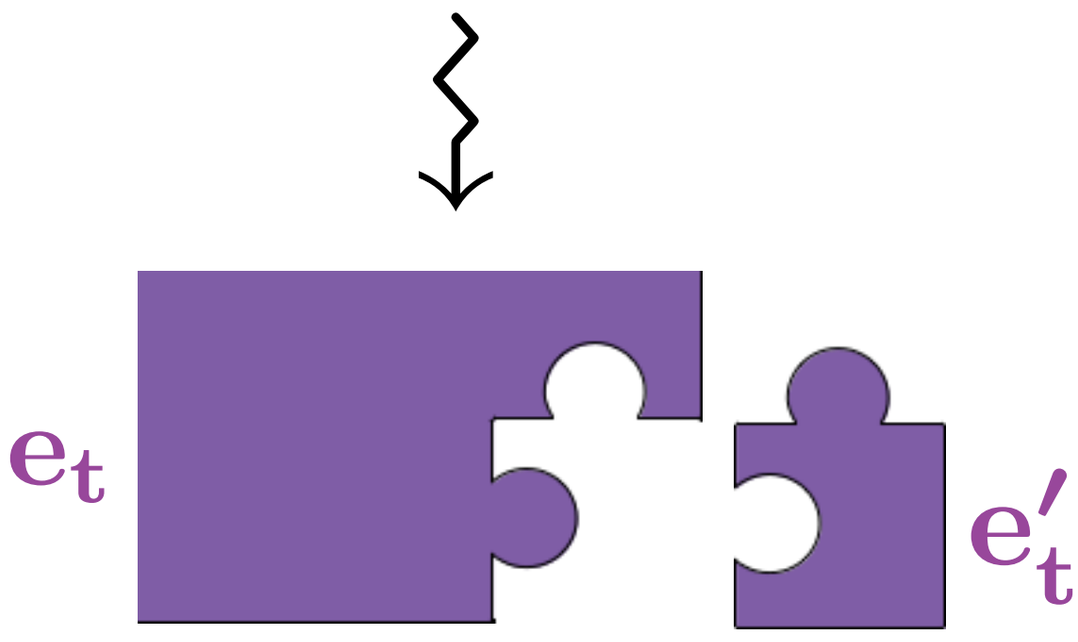
$e_S \approx e_T$

expressed how?



Correct Compilation of Components?

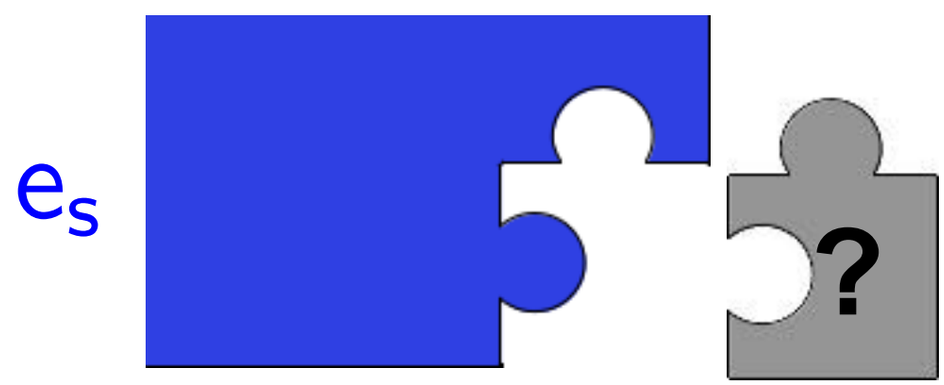


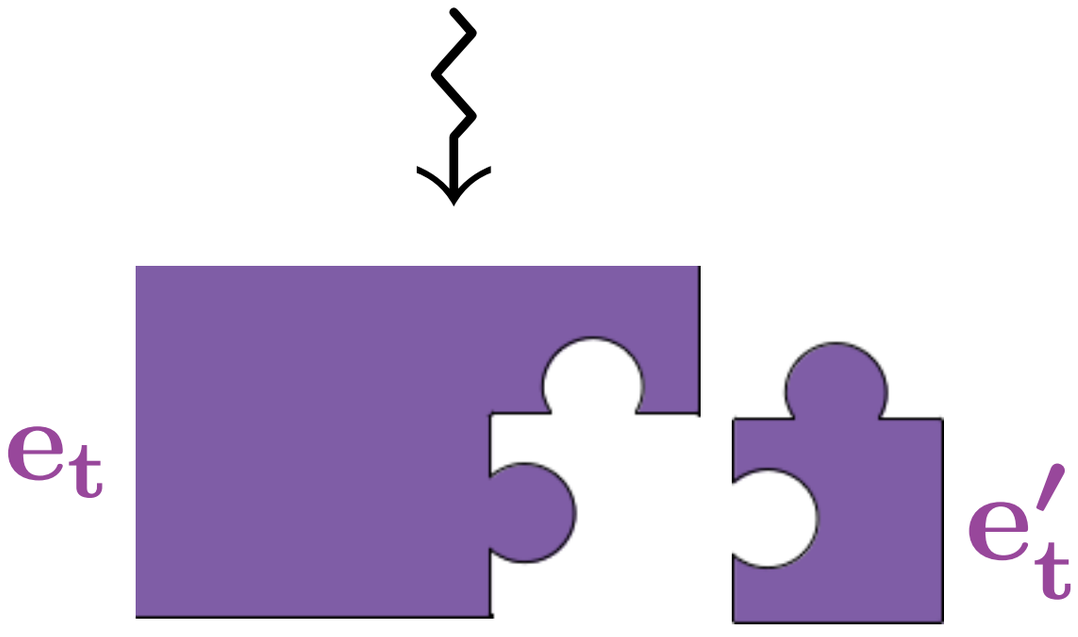


$e_S \approx e_T$

expressed how?

“Compositional” Compiler Correctness

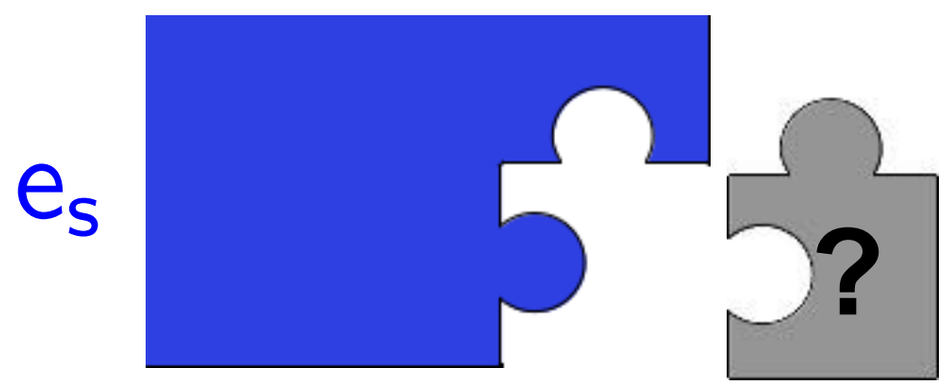


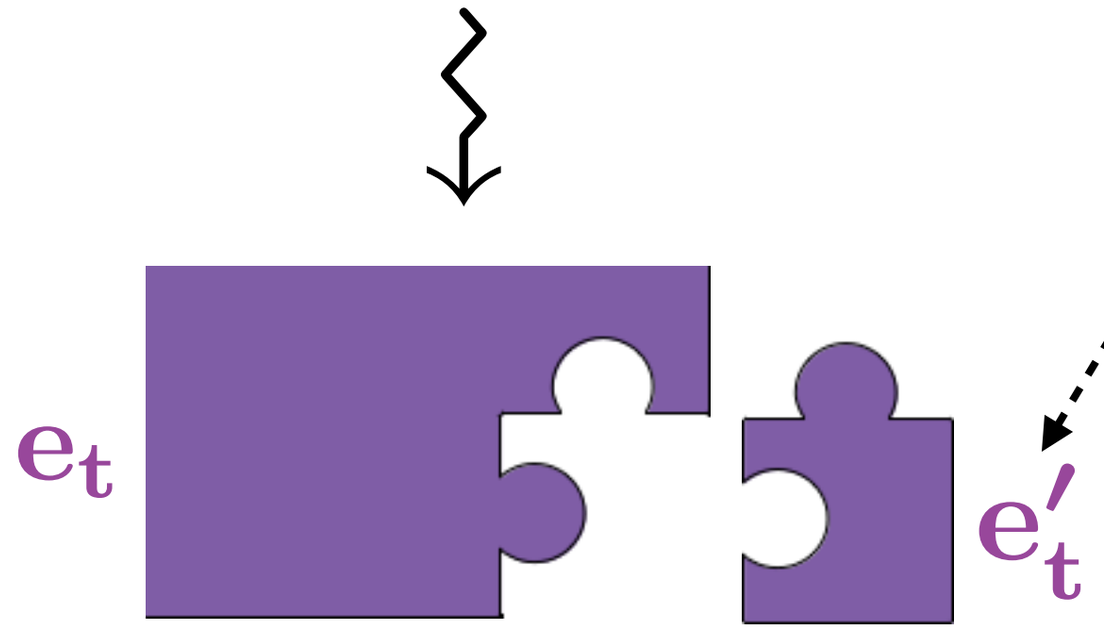


$e_S \approx e_T$

expressed how?

“Compositional” Compiler Correctness





$e_S \approx e_T$

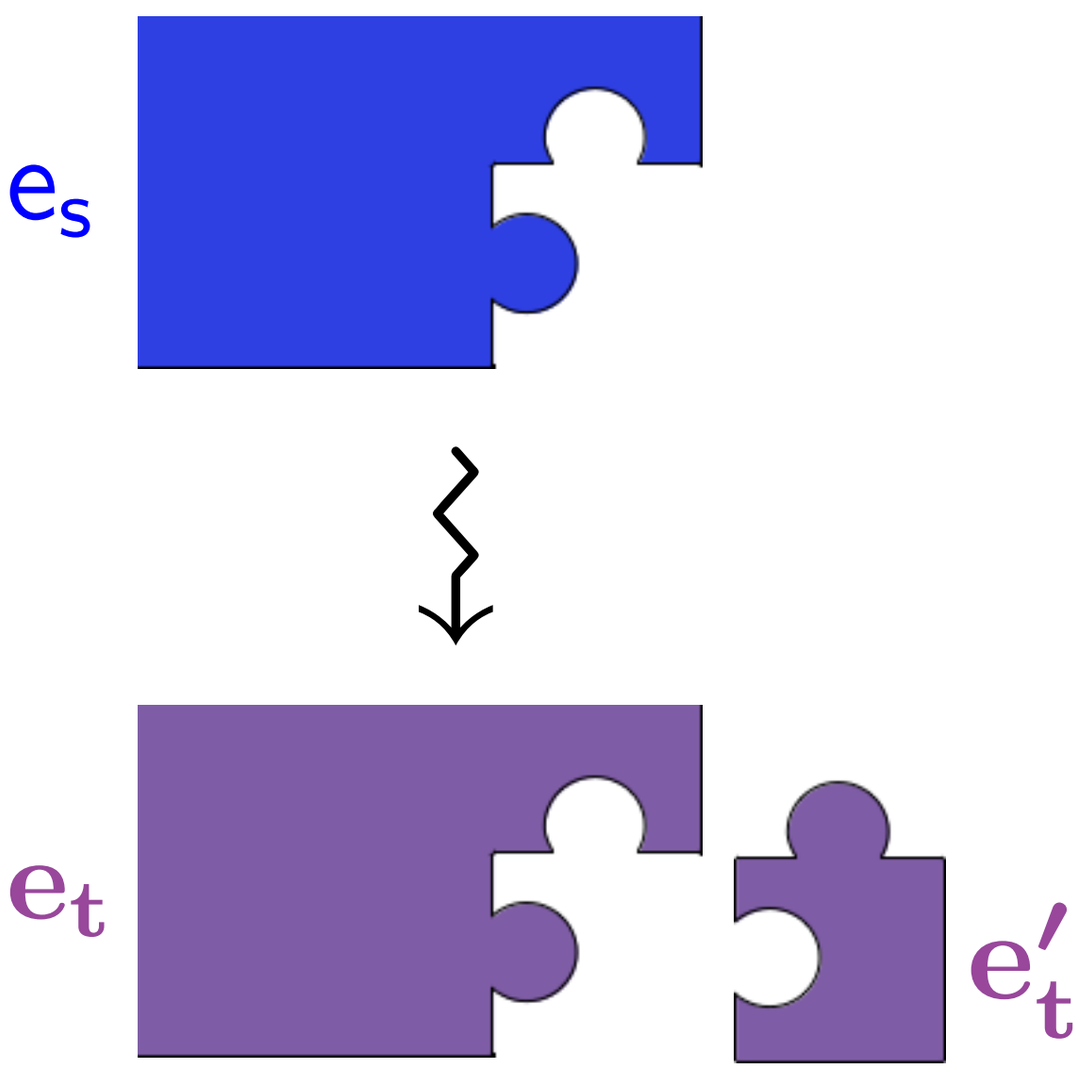
expressed how?

Produced by
- *same compiler,*
- *diff compiler for S,*
- *compiler for diff lang R,*
- *R that’s **very** diff from S?*

Behavior expressible in S?

“Compositional” Compiler Correctness

If we want to verify realistic compilers…



$e_S \approx e_T$

Definition should:

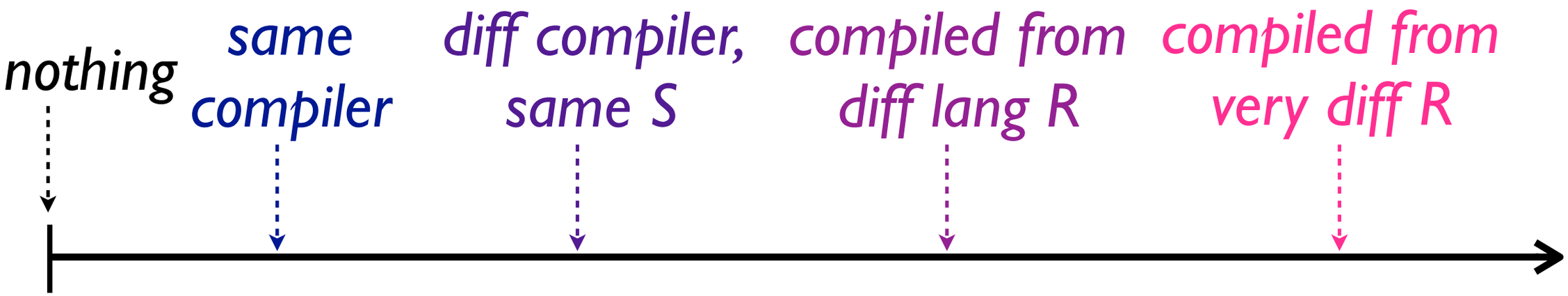
- permit **linking** with target code of arbitrary provenance
- support verification of **multi-pass** compilers

Next: Survey of State of the Art

- Survey of "compositional" compiler correctness results
  - how to express $e_S \approx e_T$

- How does the choice affect:
  - what we can **link** with *(horizontal compositionality)*
  - how we check if some e'_t is okay to link with
  - effort required to prove *transitivity* for **multi-pass** compilers *(vertical compositionality)*
  - effort required to have confidence in theorem statement

What we can link with



CompCert

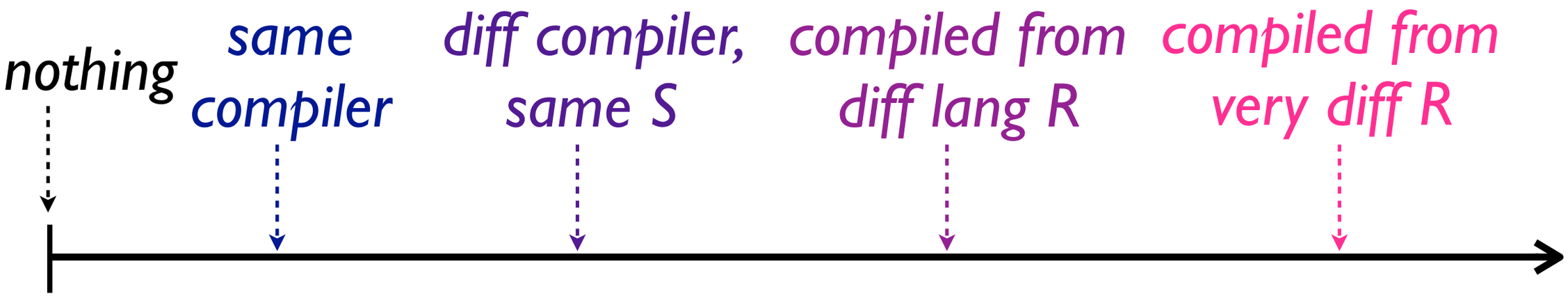
SepCompCert
Kang et al.'16

Pilsner
Neis et al.'15

Compositional CompCert
Stewart et al.'15

Multi-language ST
Perconti-Ahmed'14

What we can link with



CompCert

SepCompCert
Kang et al.'16

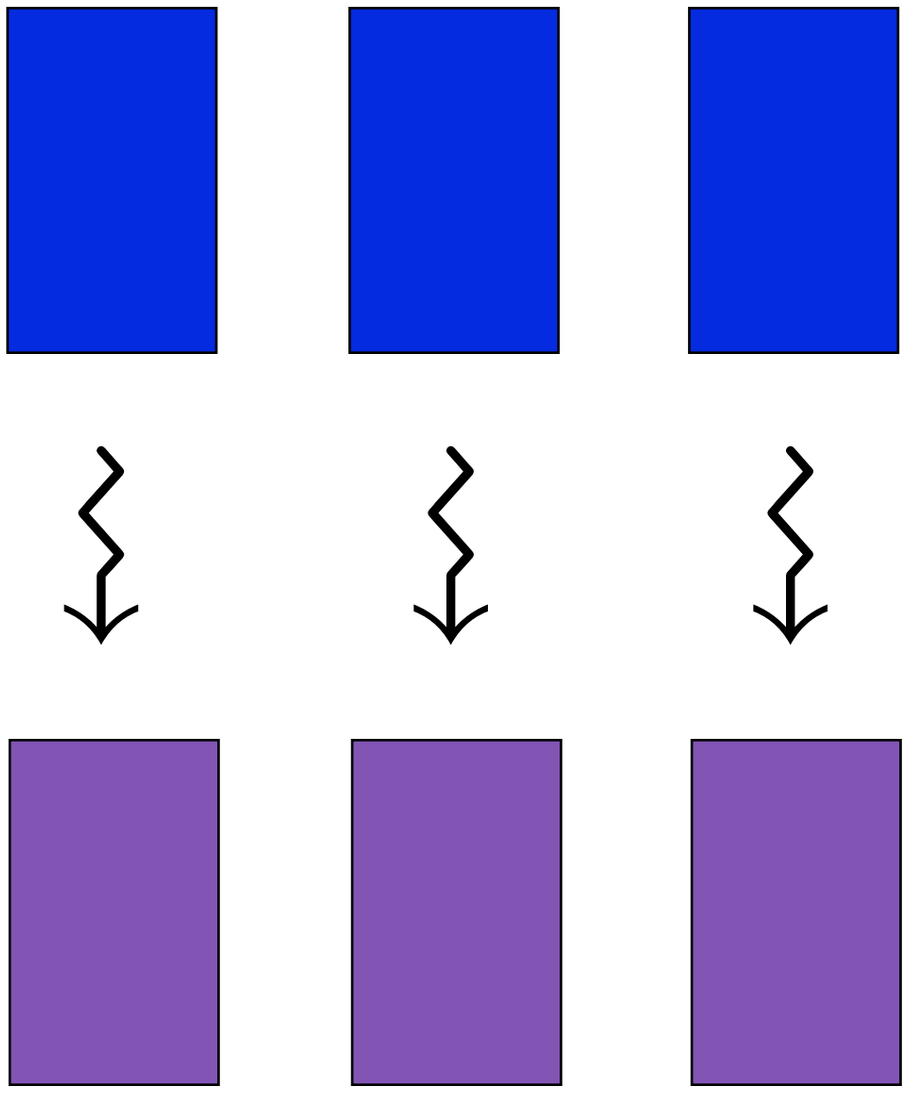
Pilsner
Neis et al.'15

Compositional CompCert
Stewart et al.'15

Multi-language ST
Perconti-Ahmed'14

Approach: Separate Compilation

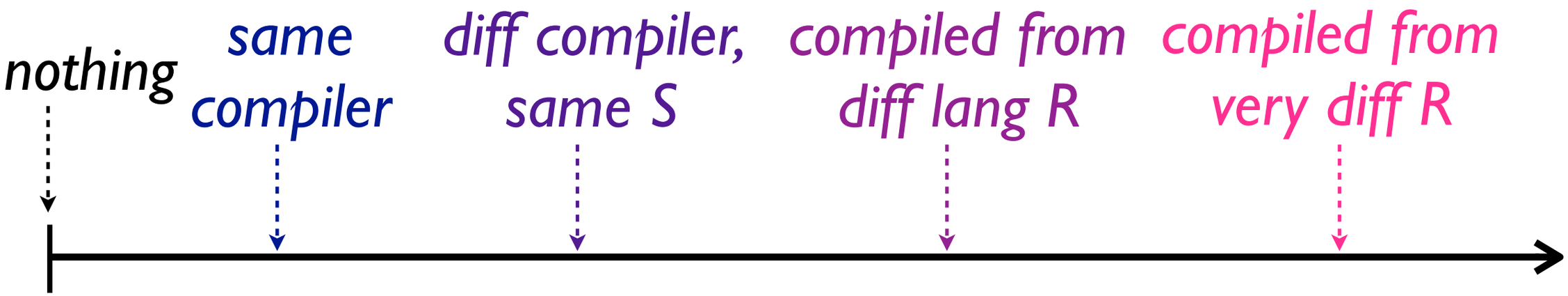
SepCompCert
[Kang et al.'16]



Level A correctness:
exactly same compiler

Level B correctness:
can omit some intra-language (RTL) optimizations

What we can link with



CompCert

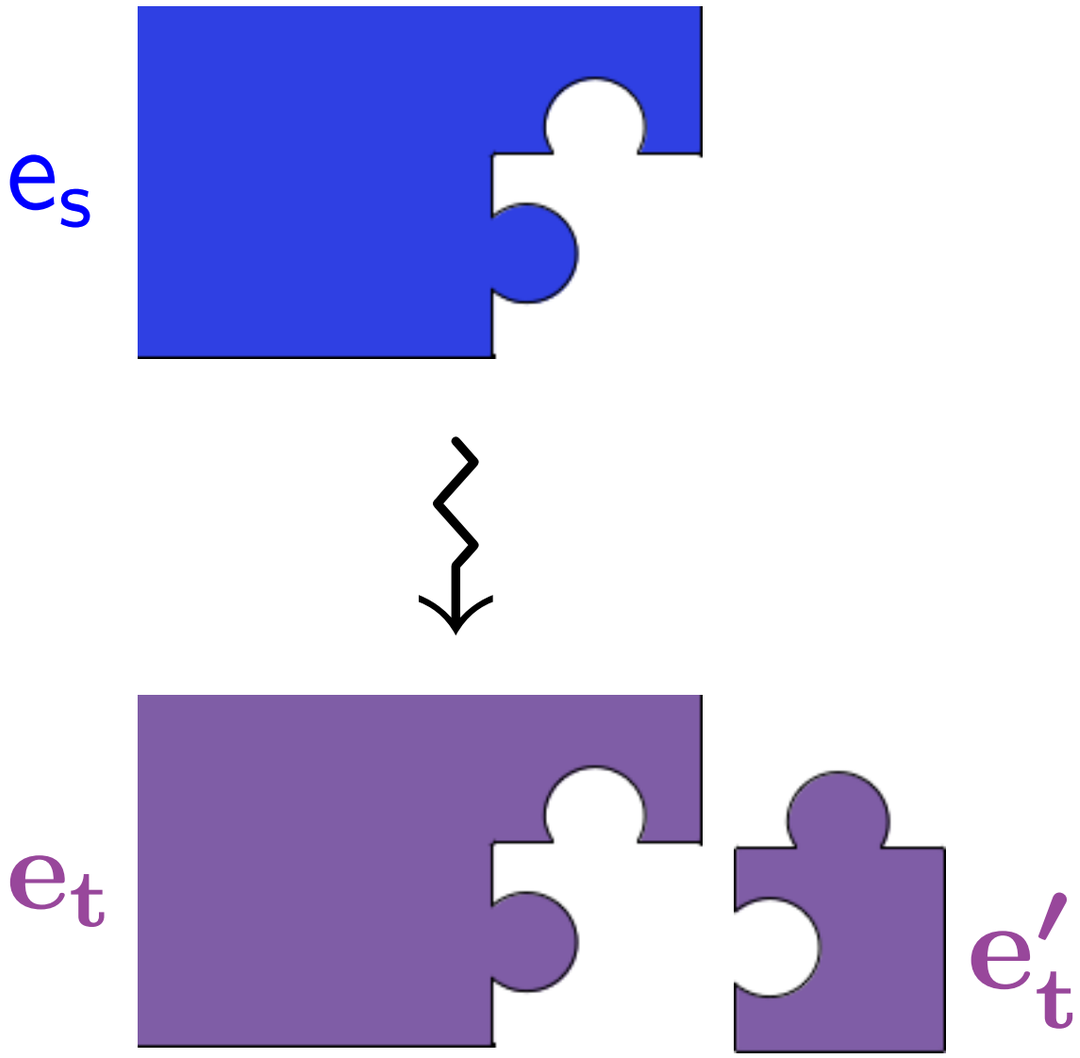
SepCompCert
Kang et al.'16

Pilsner
Neis et al.'15

Compositional CompCert
Stewart et al.'15

Multi-language ST
Perconti-Ahmed'14

Approach: Cross-Language Relations



Cross-language relation

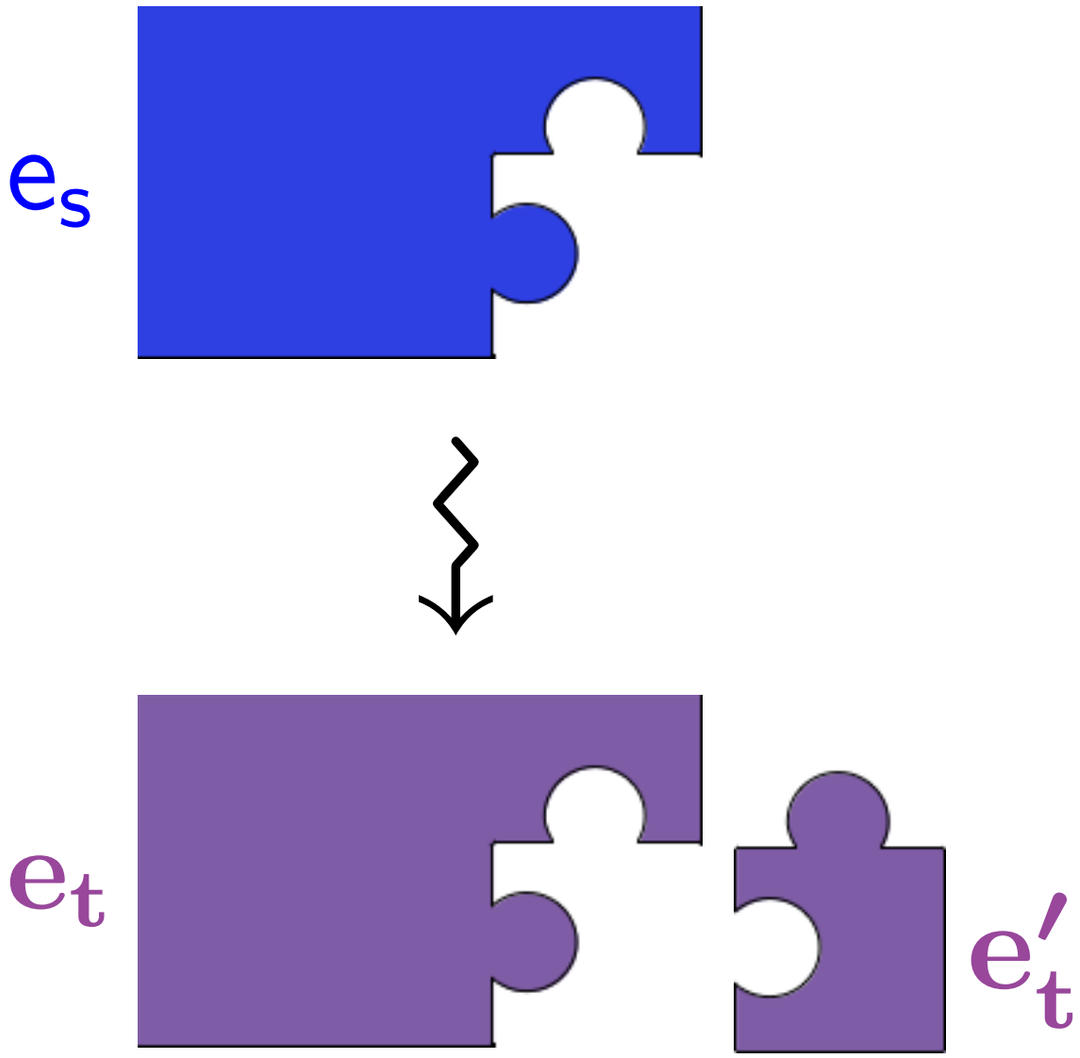
$e_S \approx e_T$

Compiling ML-like langs:

Logical relations

- *[Benton-Hur ICFP'09]*
- *[Hur-Dreyer POPL'11]*

Approach: Cross-Language Relations



Cross-language relation

$e_S \approx e_T$

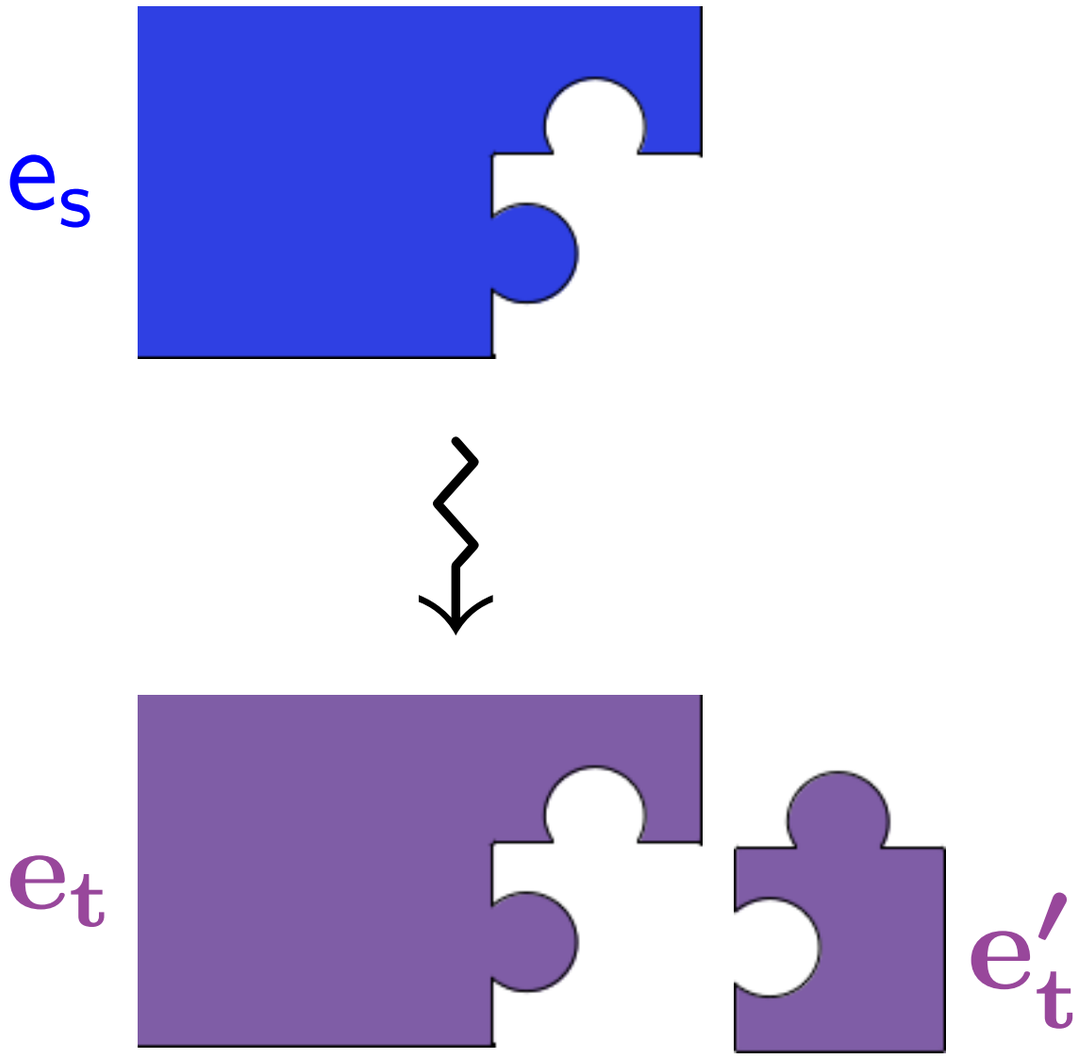
Compiling ML-like langs:

Logical relations

- [illegible]
- [illegible]

No transitivity!

Approach: Cross-Language Relations



Cross-language relation

$e_S \approx e_T$

Compiling ML-like langs:

Logical relations

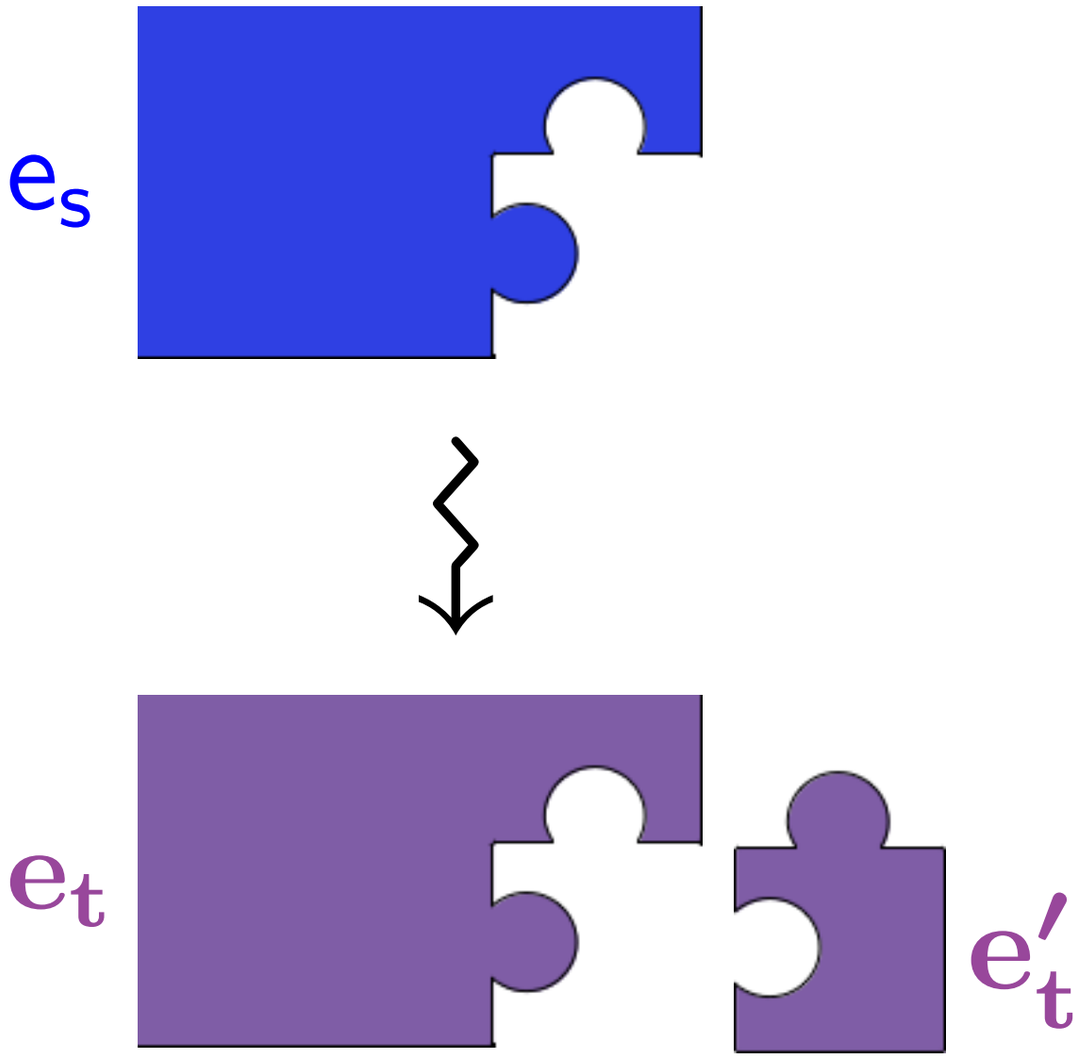
- [illegible]
- [illegible]

No transitivity!

Parametric inter-language simulations (PILS)

- *[Neis et al. ICFP'15]*

Approach: Cross-Language Relations



Cross-language relation

$e_S \approx e_T$

Compiling ML-like langs:

Logical relations

- [illegible]

No transitivity!

Parametric inter-language simulations (PILS)

- [illegible]

Prove transitivity, but requires effort!

Cross-Language Relation (Pilsner)

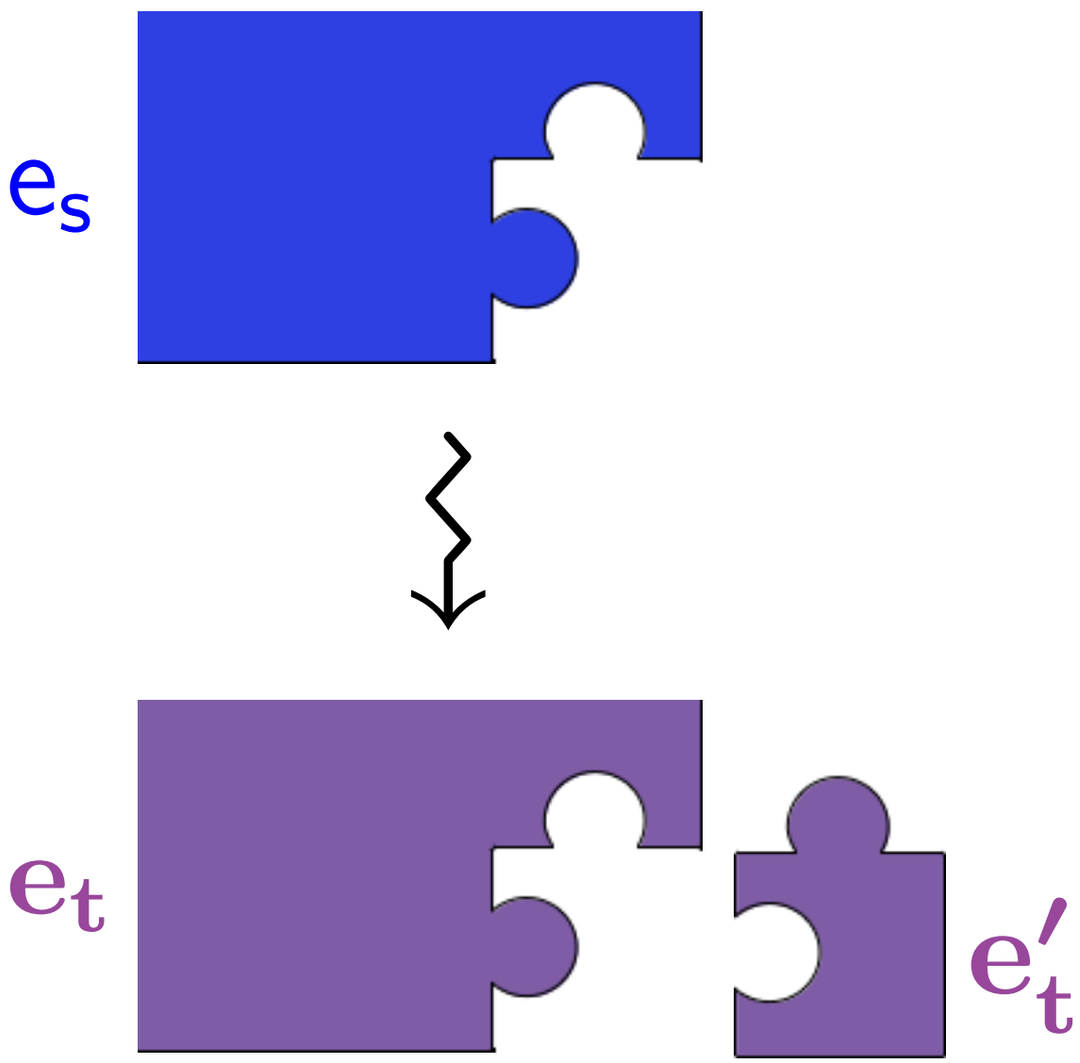
$$\mathsf{x} : \tau' \vdash \mathsf{e_s} : \tau \rightsquigarrow \mathbf{e_t} \implies \mathsf{x} : \tau' \vdash \mathsf{e_s} \simeq \mathbf{e_t} : \tau$$

cross-language relation

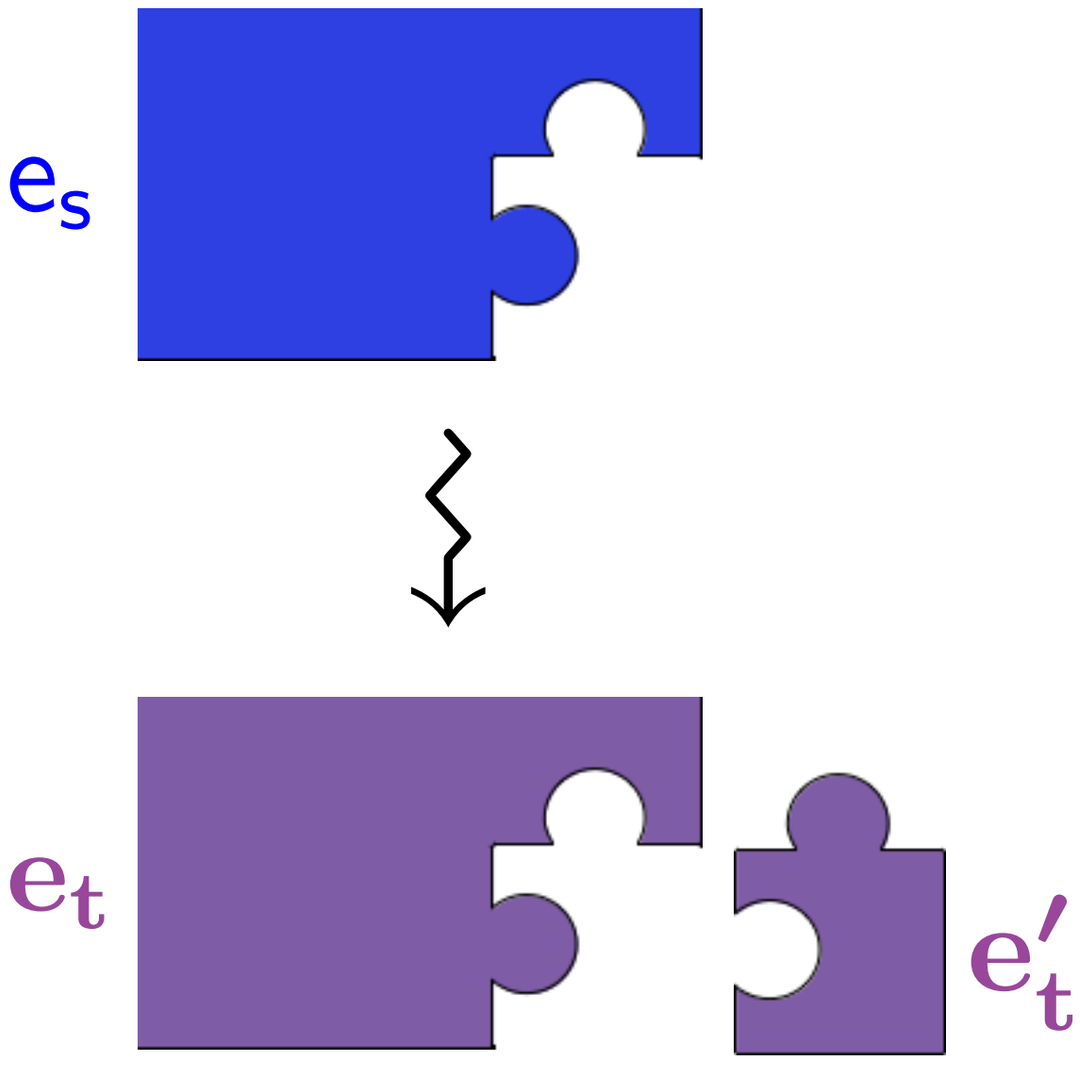
$$\forall \mathsf{e'_s}, \mathbf{e'_t}.\ \vdash \mathsf{e'_s} \simeq \mathbf{e'_t} : \tau' \implies \vdash \mathsf{e_s}[\mathsf{e'_s}/\mathsf{x}] \simeq \mathbf{e_t}[\mathbf{e'_t}/\mathbf{x}] : \tau$$

Cross-Language Relation (Pilsner)

Have $\mathsf{x} : \tau' \vdash \mathsf{e_s} \simeq \mathbf{e_t} : \tau$



Cross-Language Relation (Pilsner)



Have $x : \tau' \vdash e_s \simeq e_t : \tau$

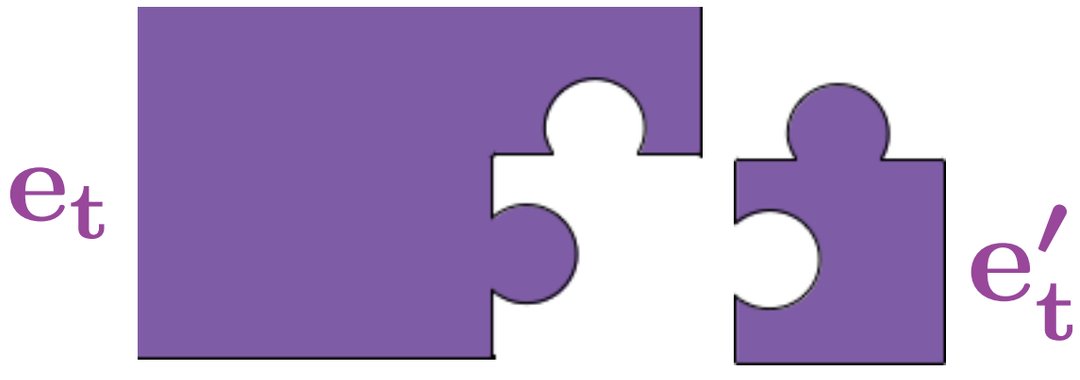
Does the compiler correctness theorem permit linking with e'_t *?*

Cross-Language Relation (Pilsner)

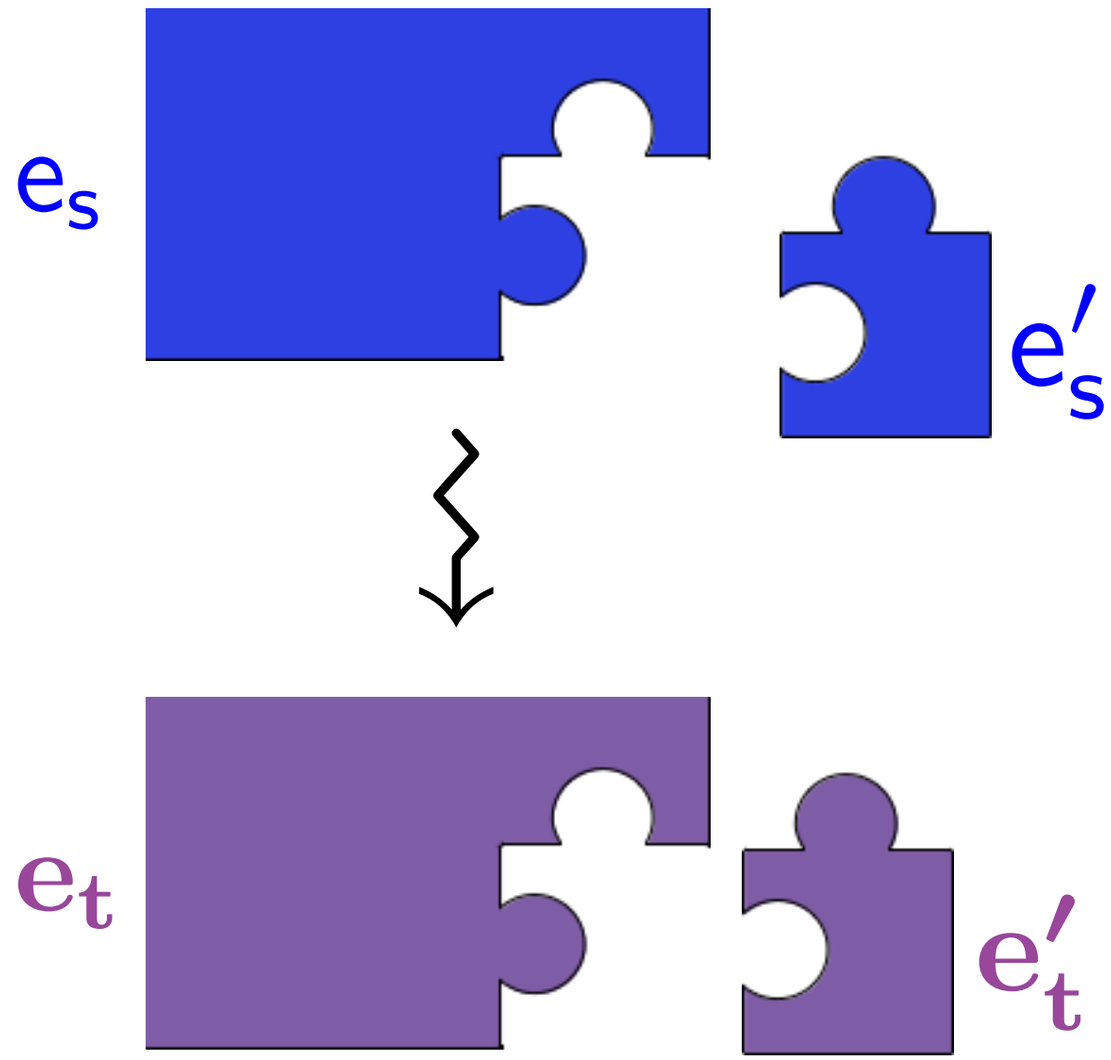
Have $\mathsf{x} : \tau' \vdash \mathsf{e_s} \simeq \mathbf{e_t} : \tau$

cross-language relation

$$\forall \mathsf{e'_s}, \mathbf{e'_t}.\ \vdash \mathsf{e'_s} \simeq \mathbf{e'_t} : \tau' \implies \vdash \mathsf{e_s}[\mathsf{e'_s}/\mathsf{x}] \simeq \mathbf{e_t}[\mathbf{e'_t}/\mathbf{x}] : \tau$$



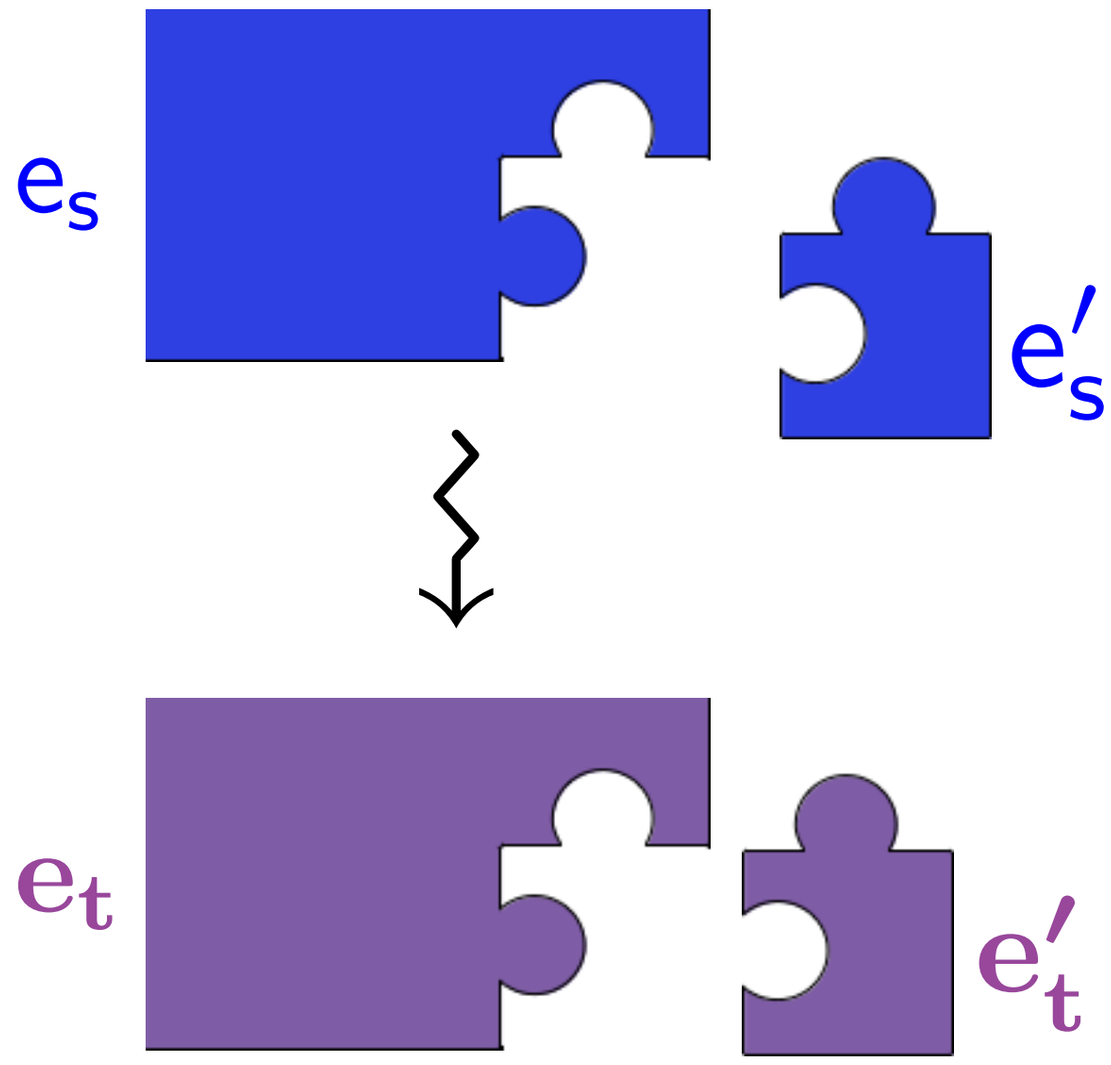
Cross-Language Relation (Pilsner)



Have $x : \tau' \vdash e_s \simeq \mathbf{e_t} : \tau$

$\vdash e'_s \simeq \mathbf{e'_t} : \tau'$

Cross-Language Relation (Pilsner)

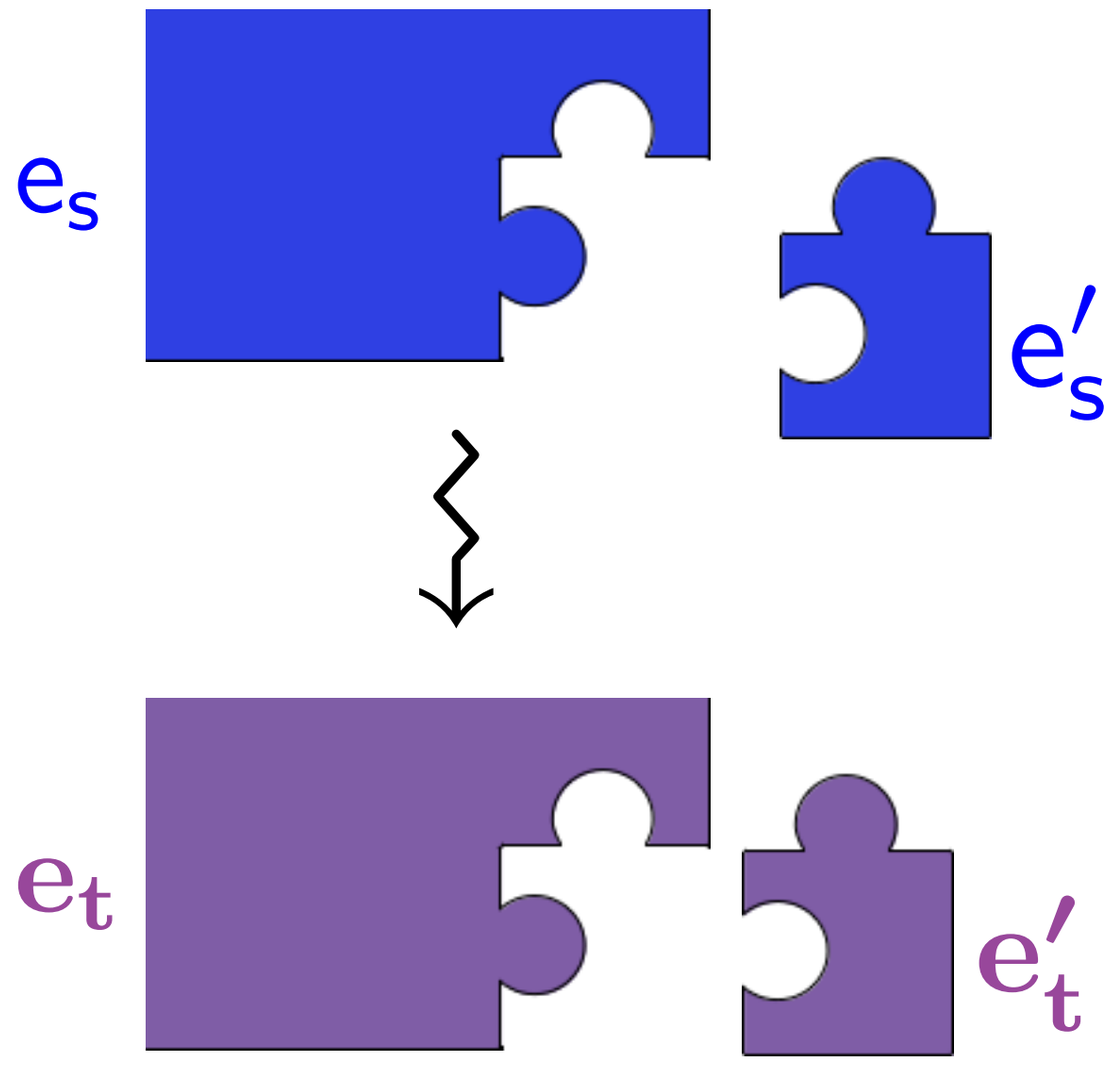


Have $x : \tau' \vdash e_s \simeq \mathbf{e_t} : \tau$

$\vdash e'_s \simeq \mathbf{e'_t} : \tau'$

$\therefore \vdash e_s[e'_s/x] \simeq \mathbf{e_t}[\mathbf{e'_t}/\mathbf{x}] : \tau$

Cross-Language Relation (Pilsner)



Have $x : \tau' \vdash e_s \simeq e_t : \tau$

$\vdash e'_s \simeq e'_t : \tau'$

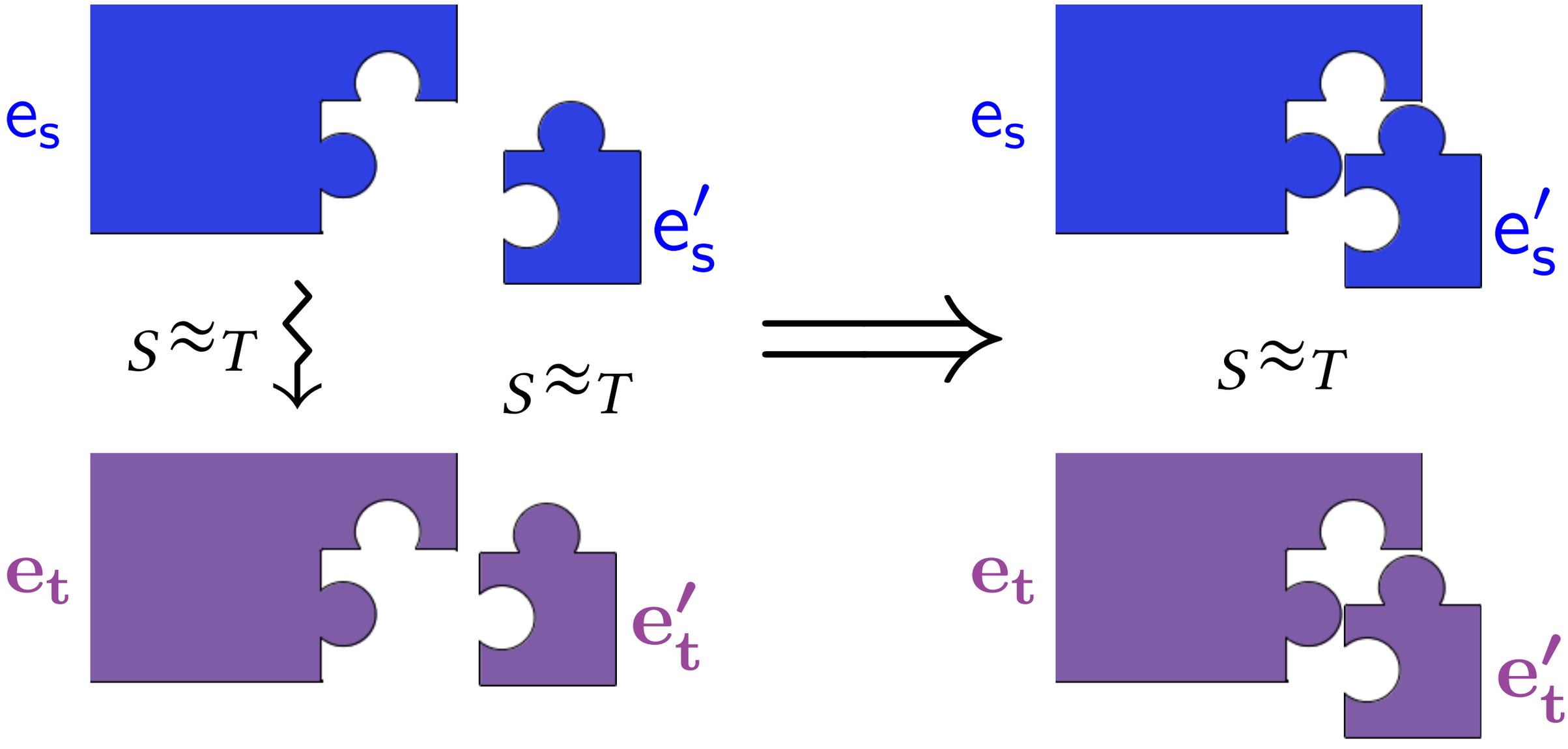
$\therefore \vdash e_s[e'_s/x] \simeq e_t[e'_t/x] : \tau$

- Need to come up with e'_s -- not feasible in practice!
- Cannot link with e'_t whose behavior cannot be expressed in source.

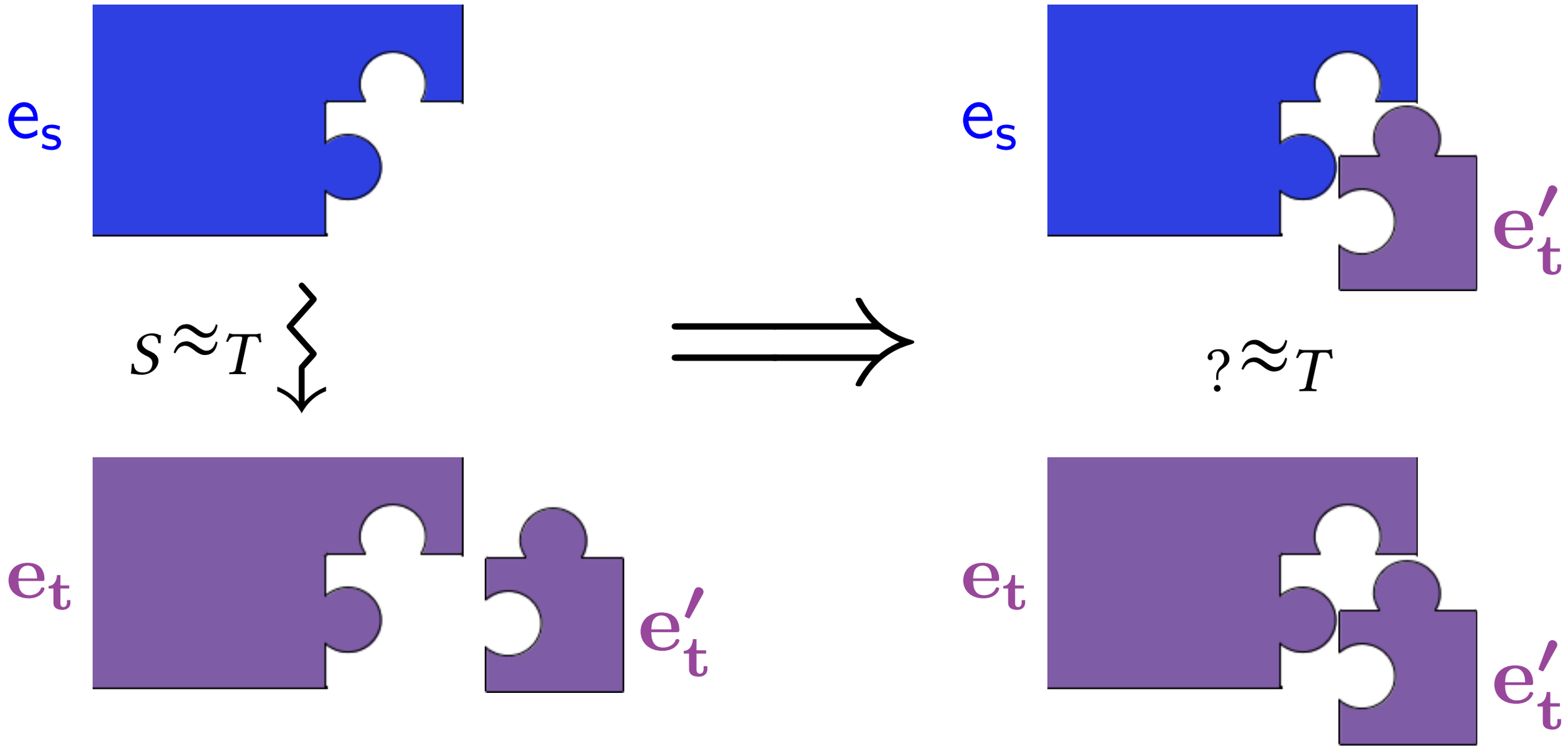
Horizontal Compositionality

Linking

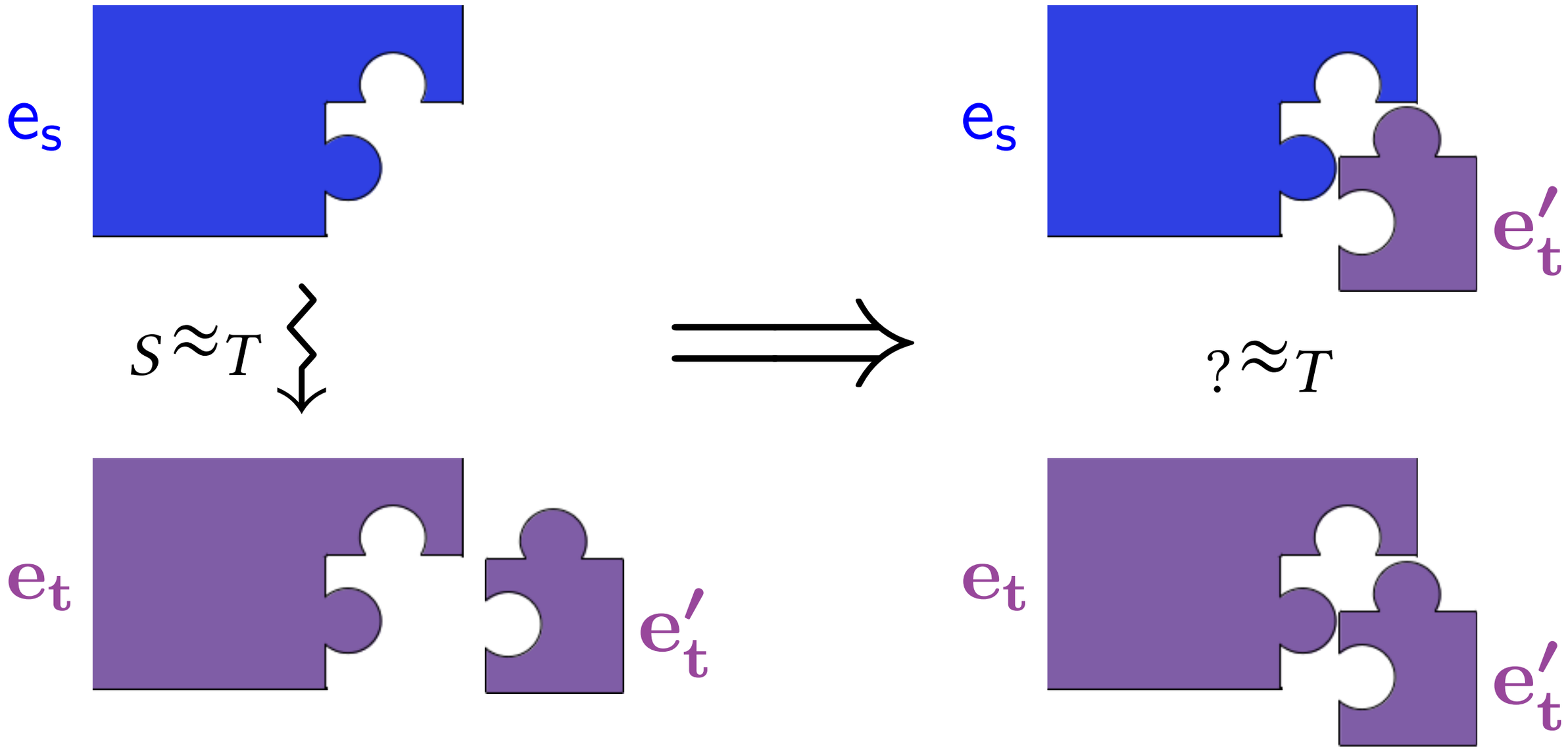
Horizontal Compositionality



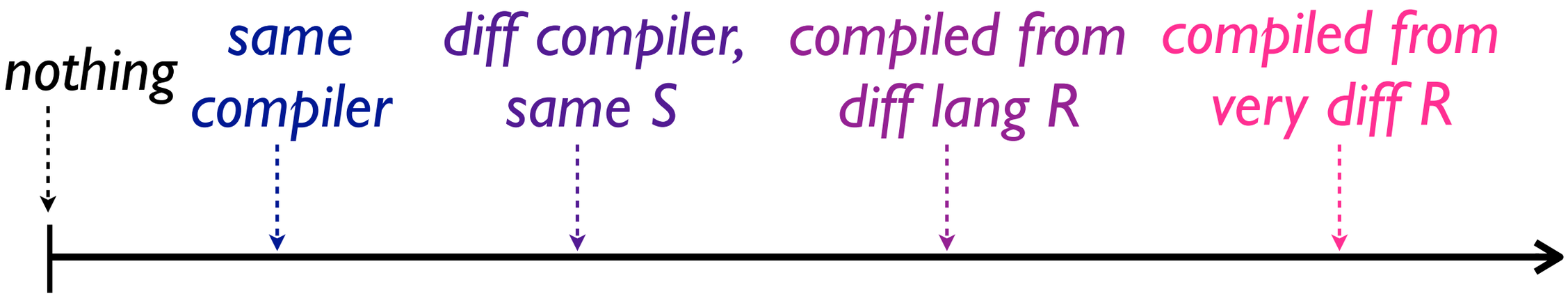
Linking



Source-Independent Linking



What we can link with



CompCert

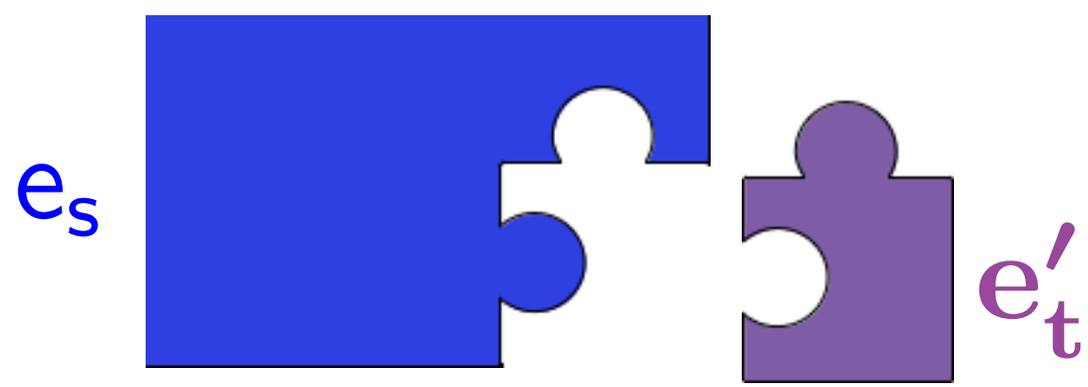
SepCompCert
Kang et al.'16

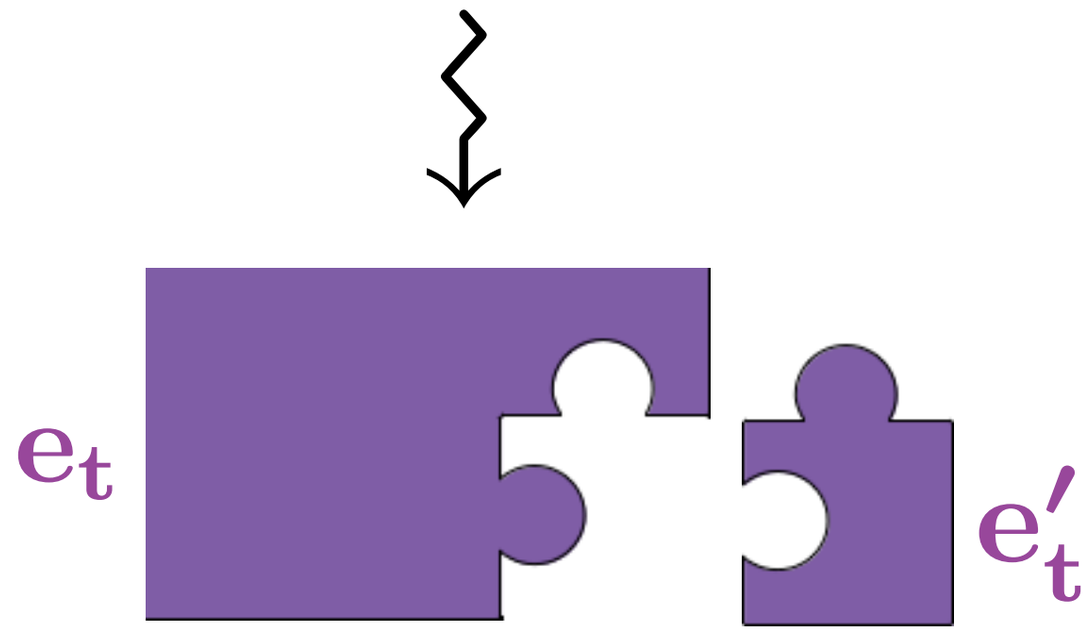
Pilsner
Neis et al.'15

Compositional CompCert
Stewart et al.'15

Multi-language ST
Perconti-Ahmed'14

Correct Compilation of Components?



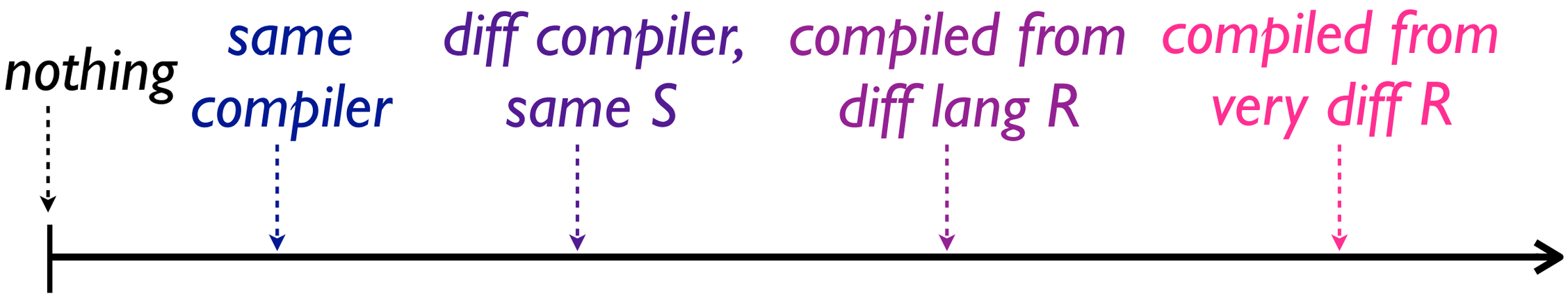


$e_S \approx e_T$

expressed how?

Need a semantics
of source-target
interoperability:
- *interaction semantics*
- *source-target multi-language*

What we can link with



CompCert

SepCompCert
Kang et al.'16

Pilsner
Neis et al.'15

Compositional CompCert
Stewart et al.'15

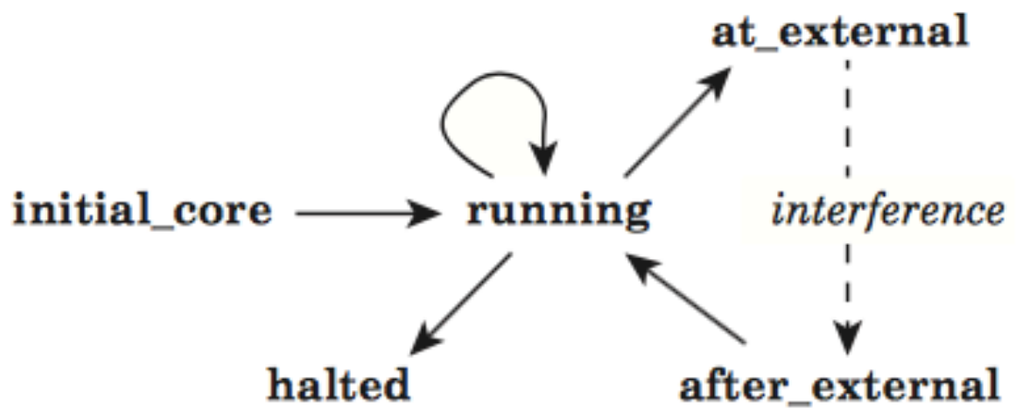
Multi-language ST
Perconti-Ahmed'14

Approach: Interaction Semantics

Compositional CompCert

[Stewart et al.'15]

- Language-independent linking



$$
\begin{array}{l}
\mathsf{Semantics}\ (G\ C\ M : \mathsf{Type}) : \mathsf{Type} \triangleq \\
\left\{\begin{array}{lcl}
\mathsf{initial\_core} & : & G \to \mathcal{V} \to \mathsf{list}\ \mathcal{V} \to \mathsf{option}\ C \\
\mathsf{at\_external} & : & C \to \mathsf{option}\ (\mathcal{F} \times \mathsf{list}\ \mathcal{V}) \\
\mathsf{after\_external} & : & \mathsf{option}\ \mathcal{V} \to C \to \mathsf{option}\ C \\
\mathsf{halted} & : & C \to \mathsf{option}\ \mathcal{V} \\
\mathsf{corestep} & : & G \to C \to M \to C \to M \to \mathsf{Prop}
\end{array}\right.
\end{array}
$$

Figure 2. Interaction semantics interface. The types G (global environment), C (core state), and M (memory) are parameters to the interface. $\mathcal{F}$ is the type of external function identifiers. $\mathcal{V}$ is the type of CompCert values.

Approach: Interaction Semantics

Compositional CompCert

[Stewart et al.'15]

- Language-independent linking
- Structured simulation: support rely-guarantee relationship between the different languages while retaining vertical compositionality

Approach: Interaction Semantics

Compositional CompCert

[Stewart et al.'15]

- Language-independent linking
  - uniform CompCert memory model across all languages
  - not clear how to scale to richer source langs (e.g., ML), compilers with different source/target memory models
- Structured simulation: support rely-guarantee relationship between the different languages while retaining vertical compositionality

Approach: Interaction Semantics

Compositional CompCert

[Stewart et al.'15]

- Language-independent linking

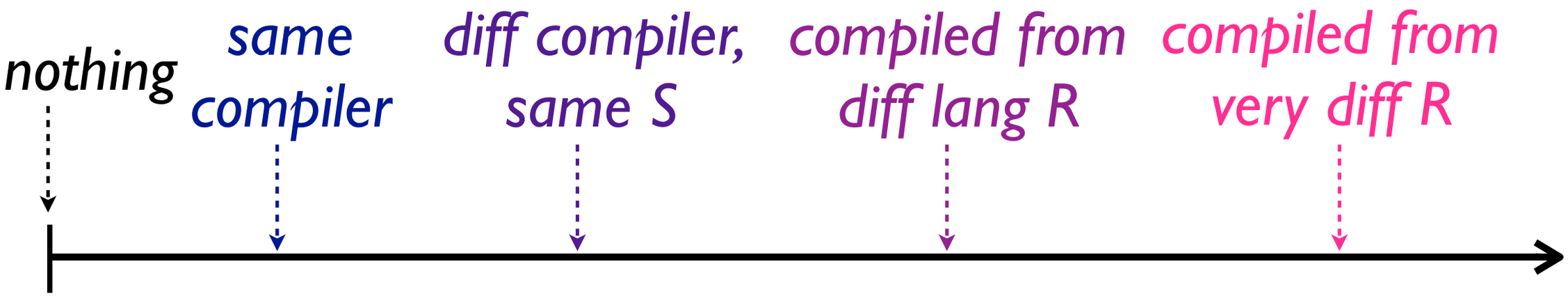
  - uniform CompCert memory model across all languages

  - not clear how to scale to richer source langs (e.g., ML), compilers with different source/target memory models

- Structured simulation: support rely-guarantee relationship between the different languages while retaining vertical compositionality

  - transitivity relies on compiler passes performing restricted set of memory transformations

What we can link with



CompCert

SepCompCert
Kang et al.'16

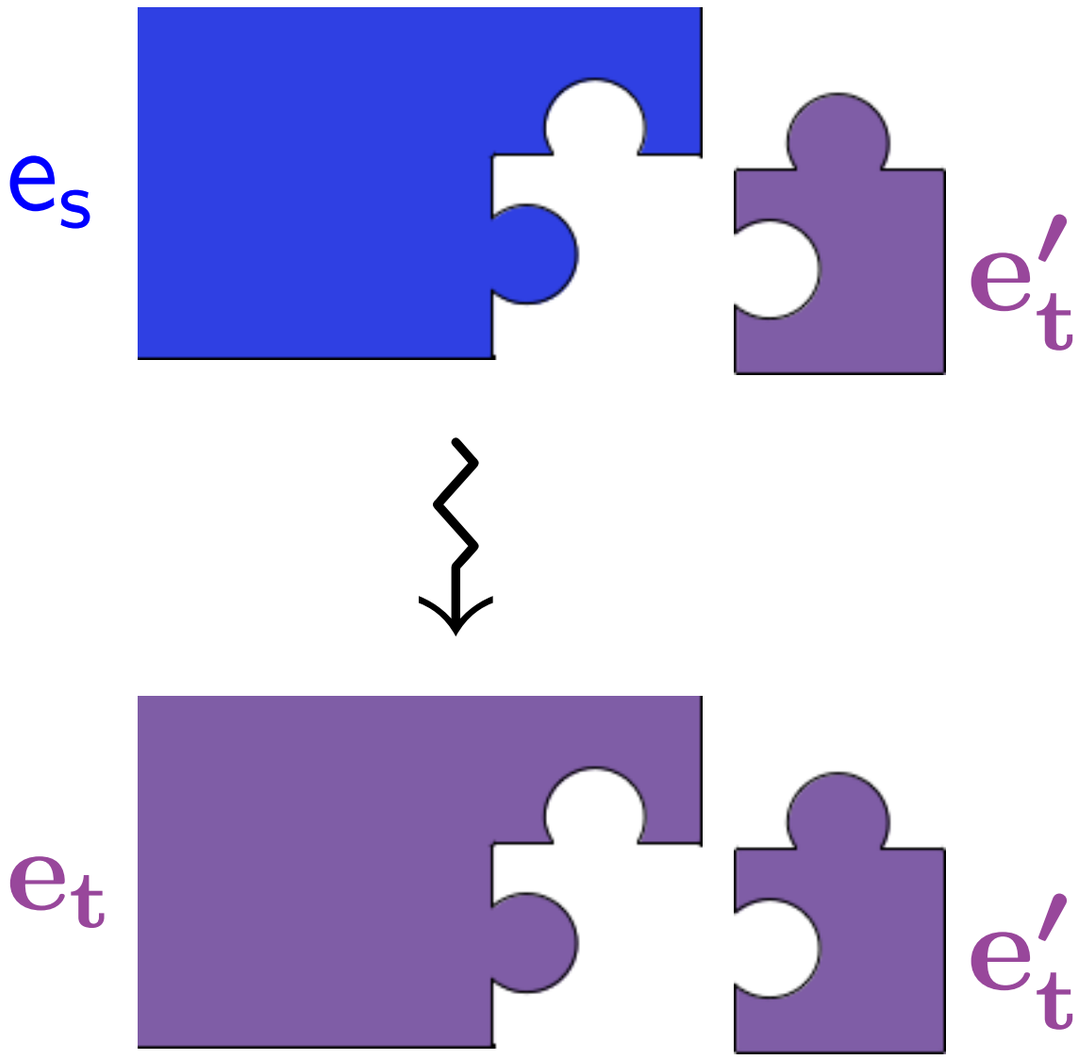
Pilsner
Neis et al.'15

Compositional CompCert
Stewart et al.'15

Multi-language ST
Perconti-Ahmed'14

Approach: Source-Target Multi-lang.

[Perconti-Ahmed'14]



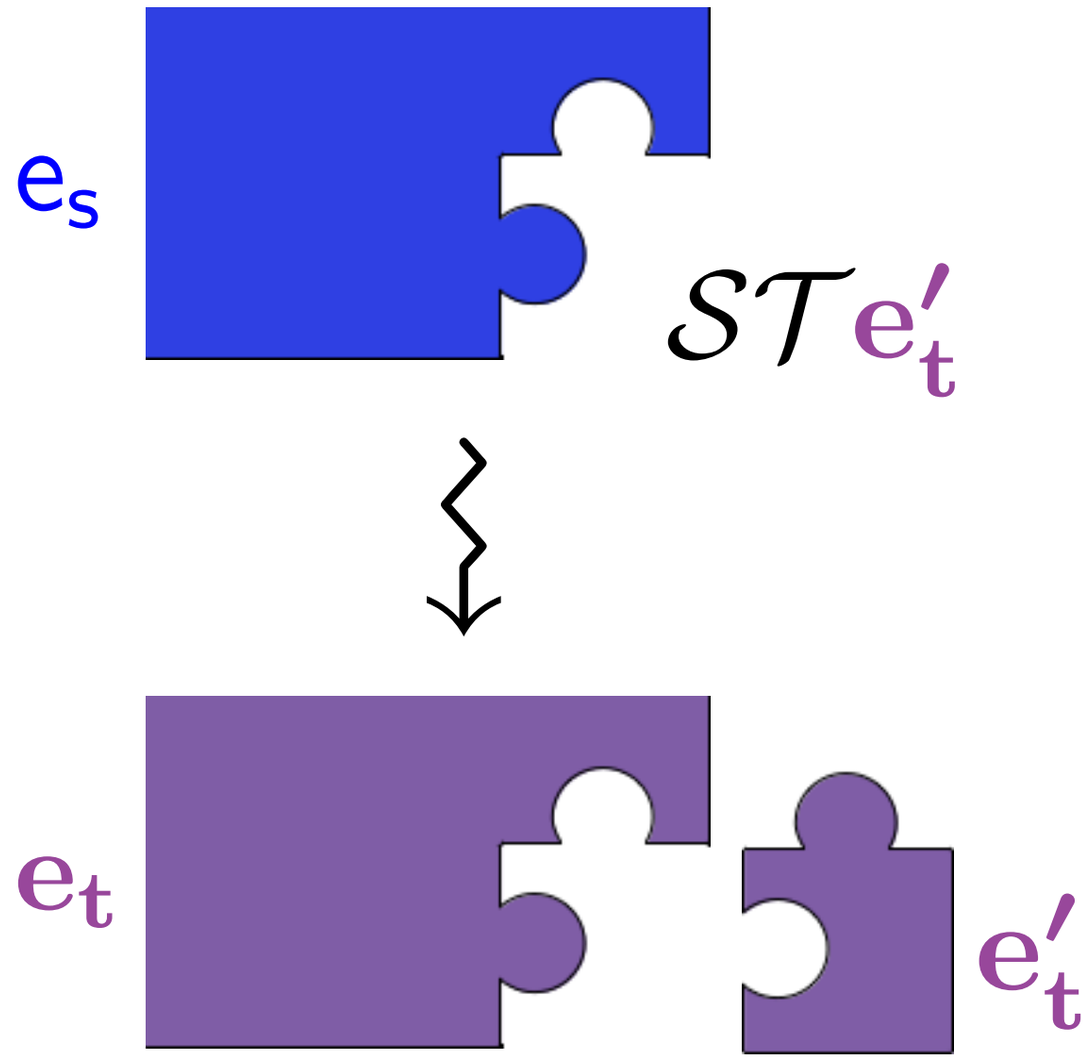
Specify semantics of source-target interoperability:

$\mathcal{ST}\,e_t \qquad \mathcal{TS}\,e_s$

Multi-language semantics: a la Matthews-Findler '07

Approach: Source-Target Multi-lang.

[Perconti-Ahmed'14]



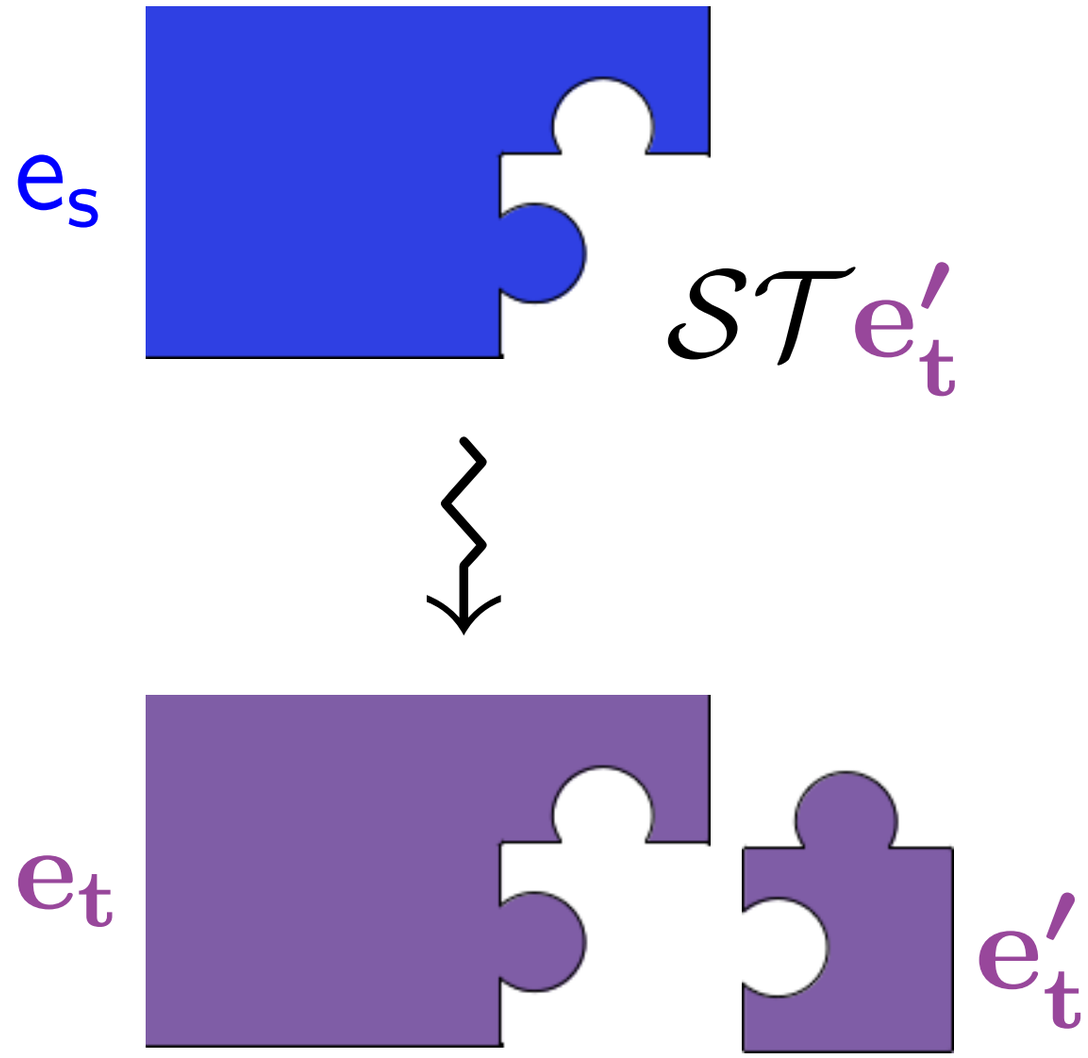
Specify semantics of source-target interoperability:

$\mathcal{ST}\,e_t$ $\quad$ $\mathcal{TS}\,e_s$

Multi-language semantics: a la Matthews-Findler '07

Approach: Source-Target Multi-lang.

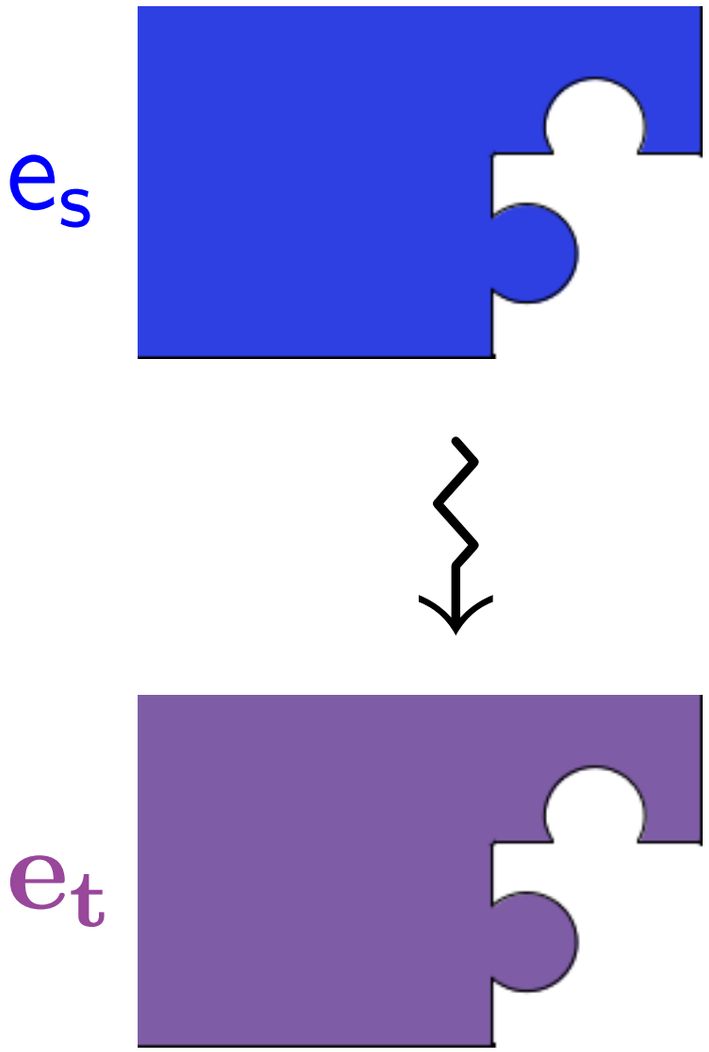
[Perconti-Ahmed'14]



$$\mathcal{TS}(e_s\ (\mathcal{ST}\,e'_t)) \approx^{ctx} e_t\ e'_t$$

Approach: Source-Target Multi-lang.

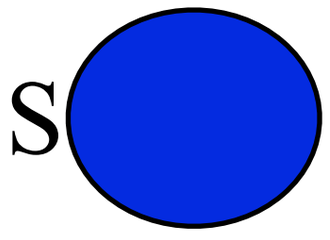
[Perconti-Ahmed'14]

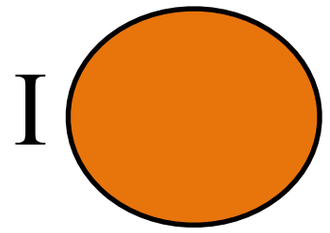


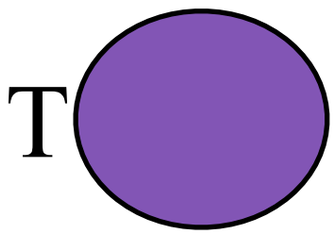
$$e_S \approx e_T \stackrel{\text{def}}{=\!=}$$

$$e_S \approx^{ctx} \mathcal{ST}\, e_T$$

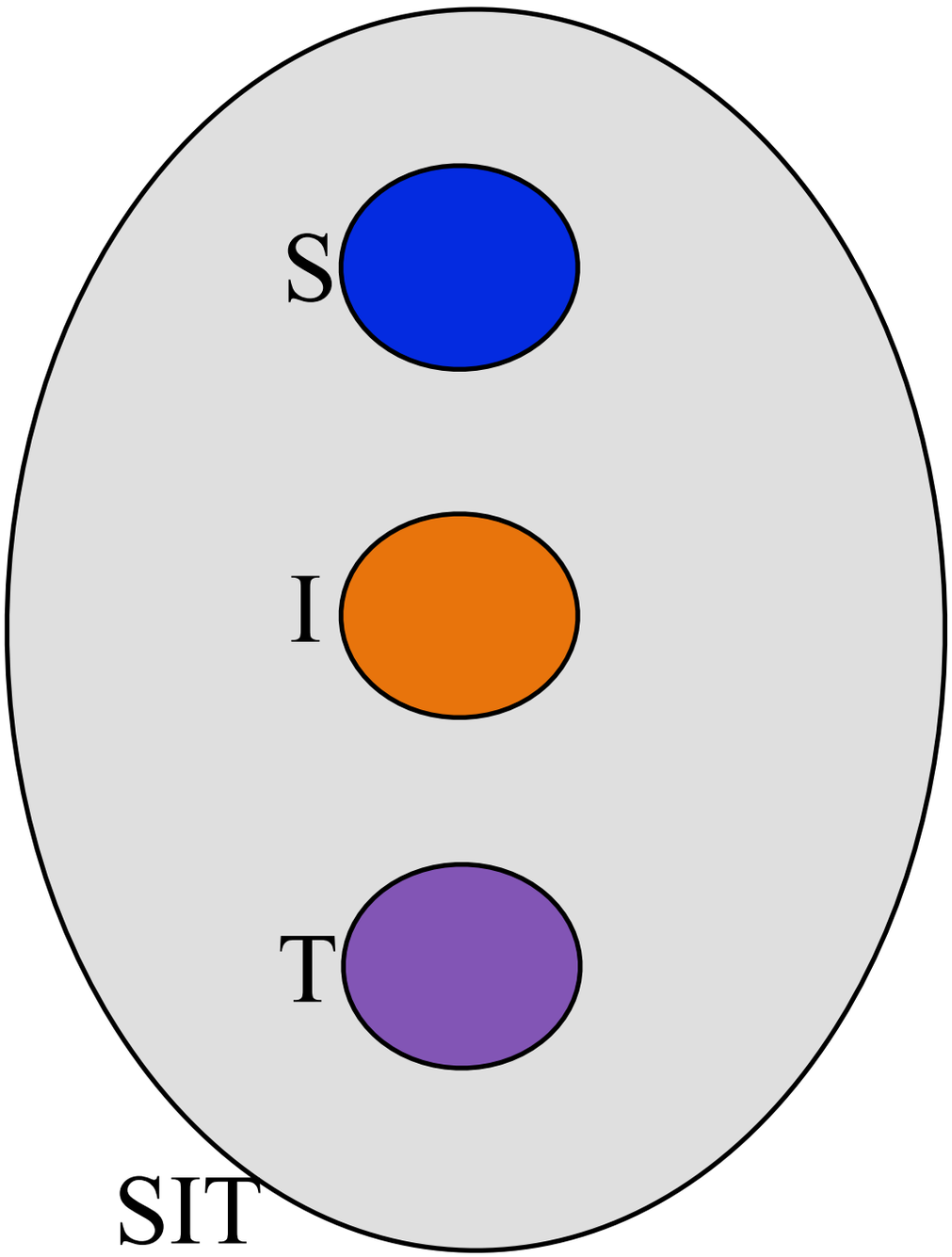
Multi-Language Semantics Approach



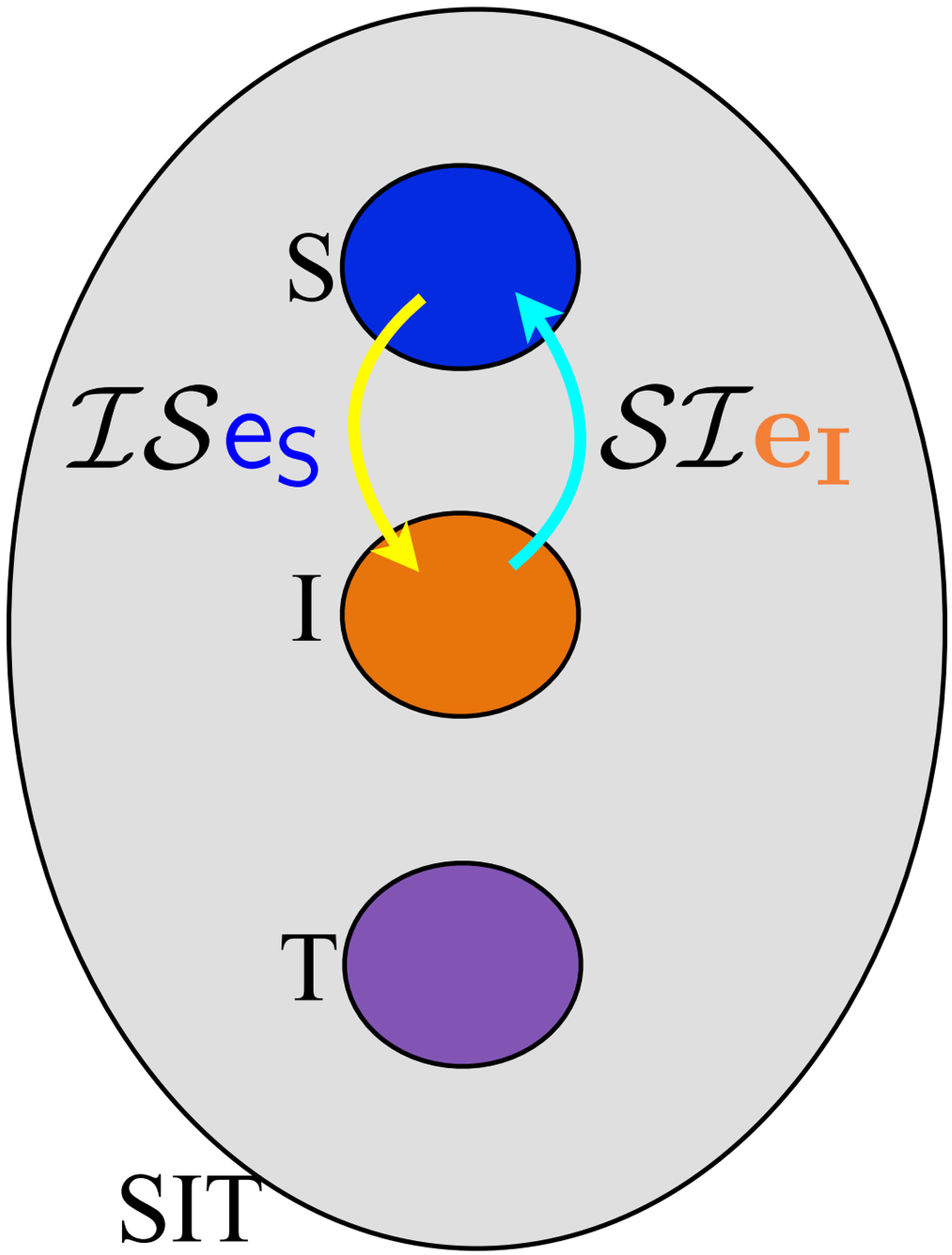




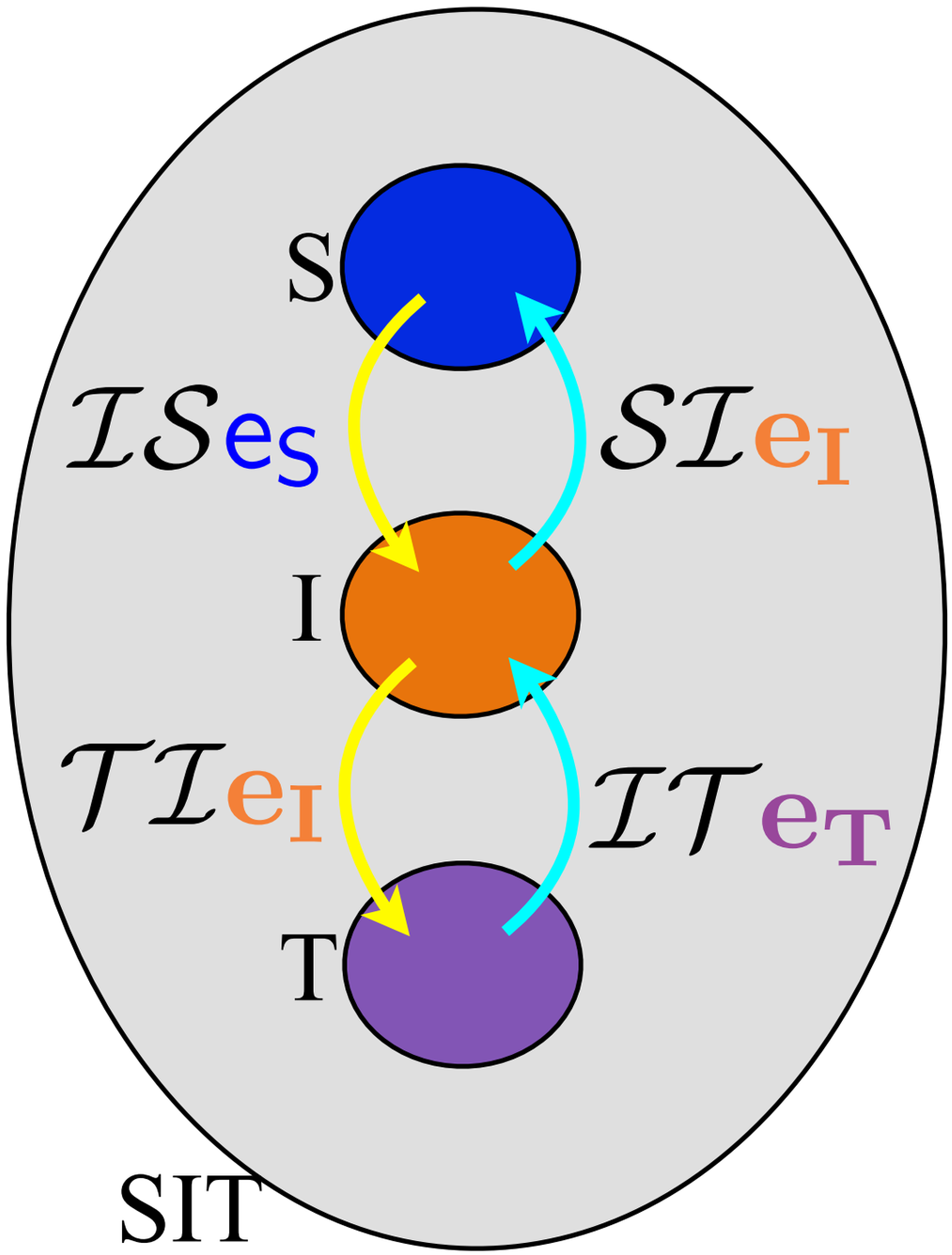
Multi-Language Semantics Approach



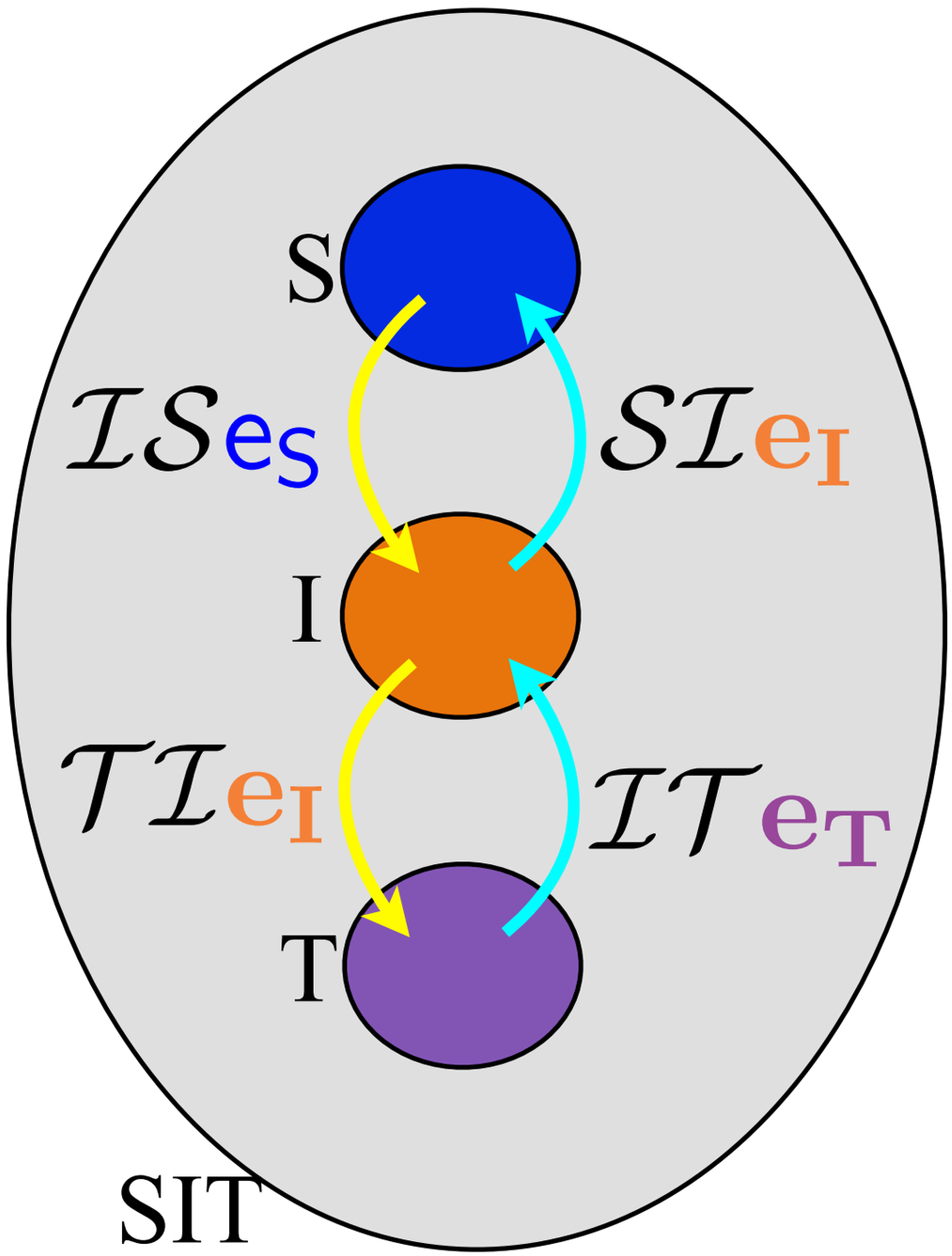
Multi-Language Semantics Approach



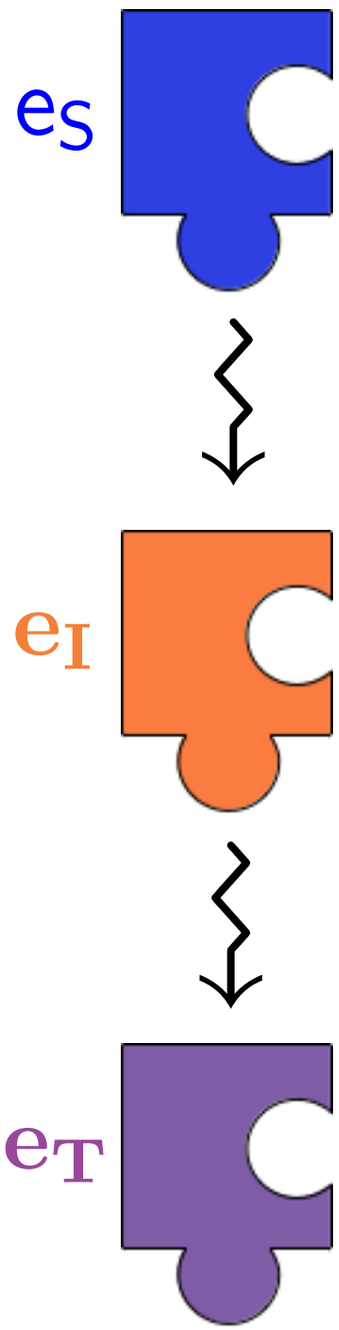
Multi-Language Semantics Approach



Multi-Language Semantics Approach



Compiler Correctness

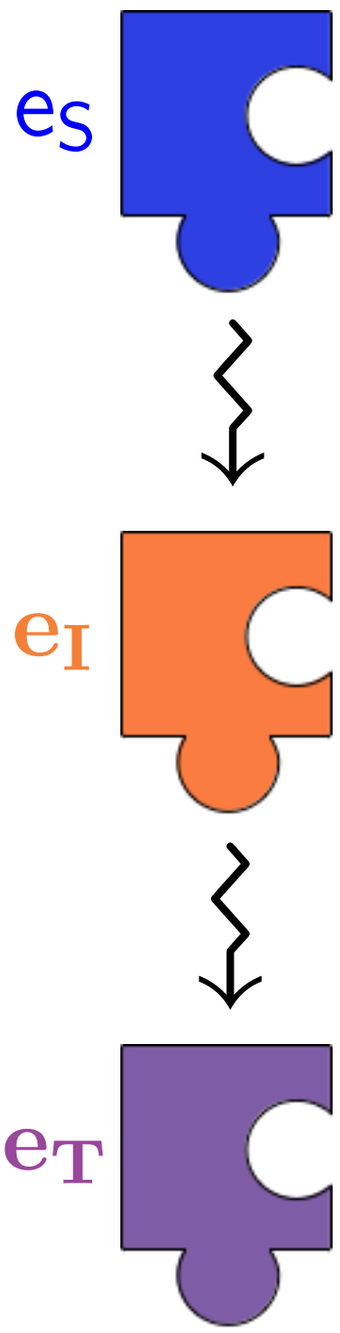


$\mathrm{e_S} \approx^{ctx} \mathcal{SI}\mathrm{e_I}$

$\mathrm{e_I} \approx^{ctx} \mathcal{IT}\mathrm{e_T}$

Multi-Lang. Approach: Multi-pass ✓

Compiler Correctness

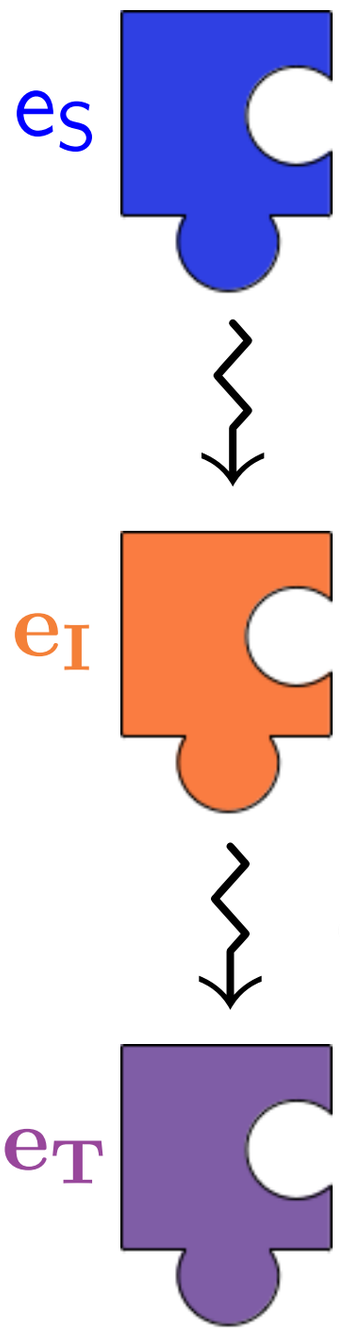


$$e_S \approx^{ctx} \mathcal{SI} e_I$$

$$e_I \approx^{ctx} \mathcal{IT} e_T$$

Multi-Lang. Approach: Multi-pass ✓

Compiler Correctness

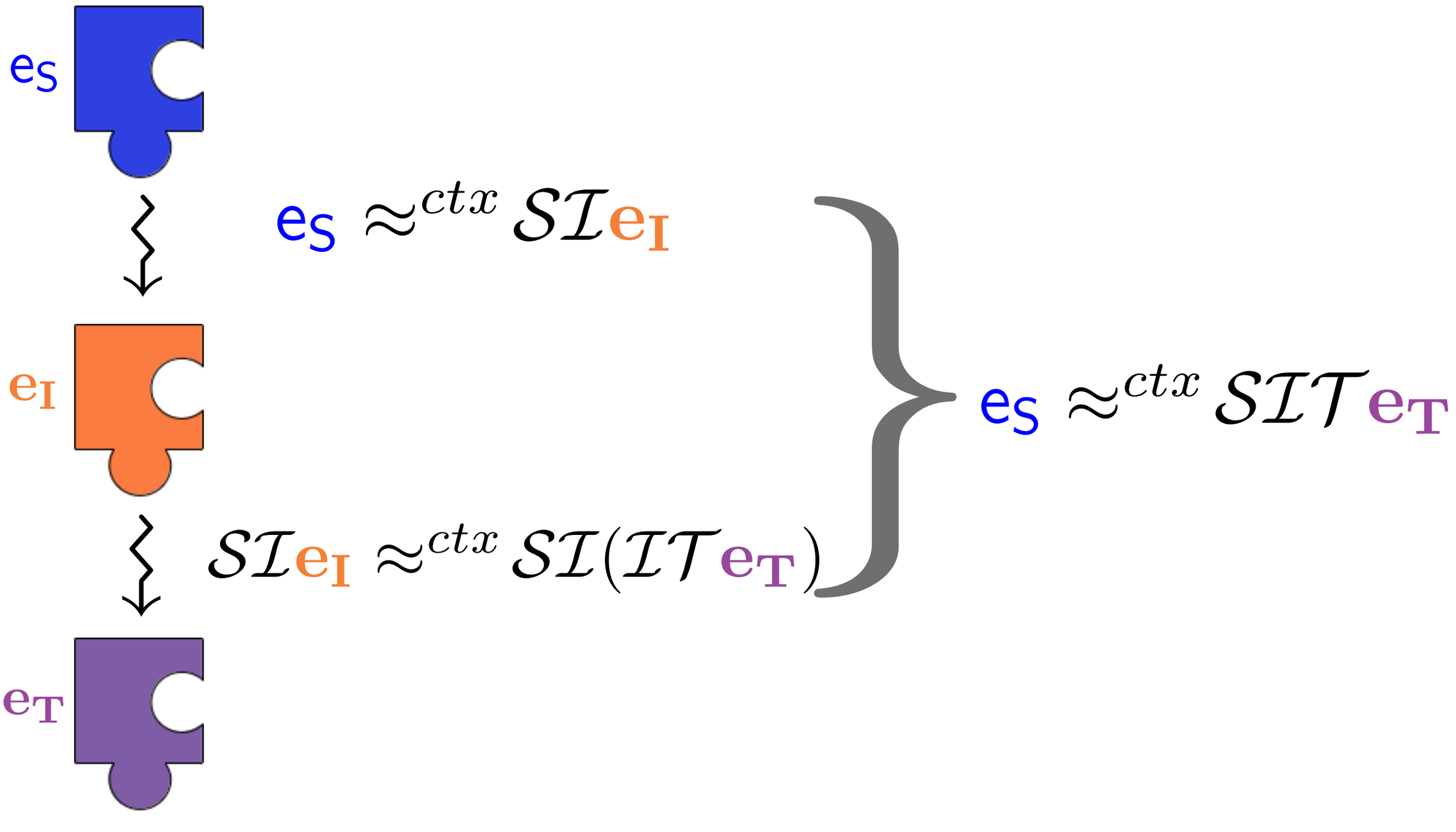


$$e_S \approx^{ctx} \mathcal{SI}e_I$$

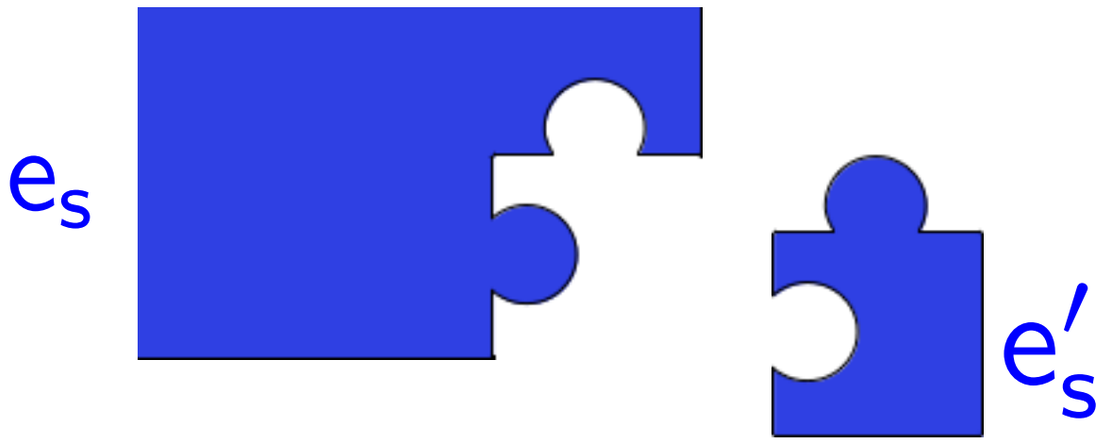
$$\mathcal{SI}e_I \approx^{ctx} \mathcal{SI}(\mathcal{IT}e_T)$$

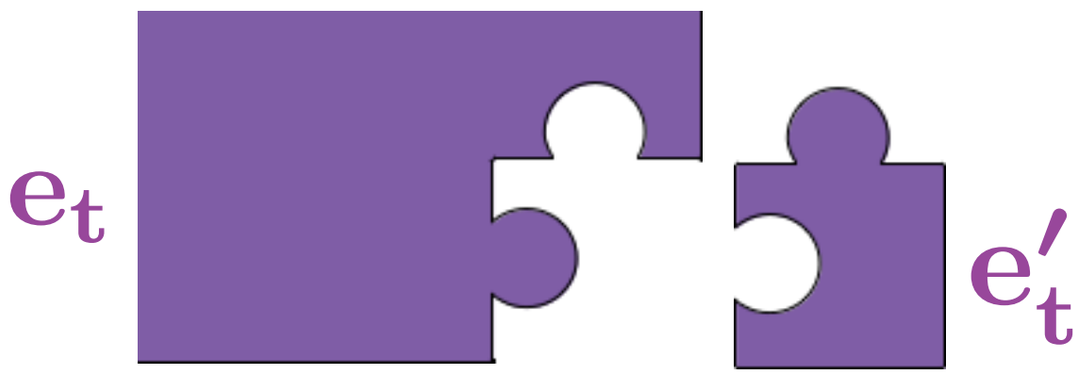
Multi-Lang. Approach: Multi-pass ✓

Compiler Correctness

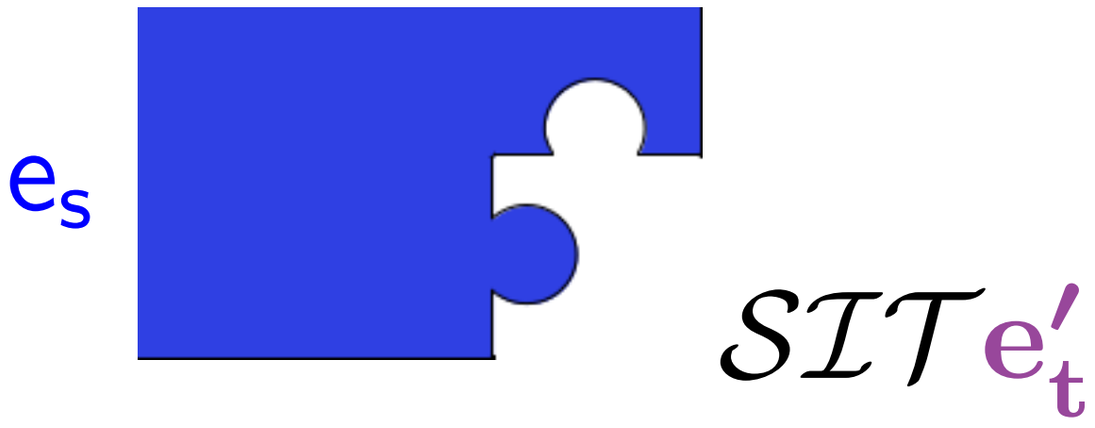


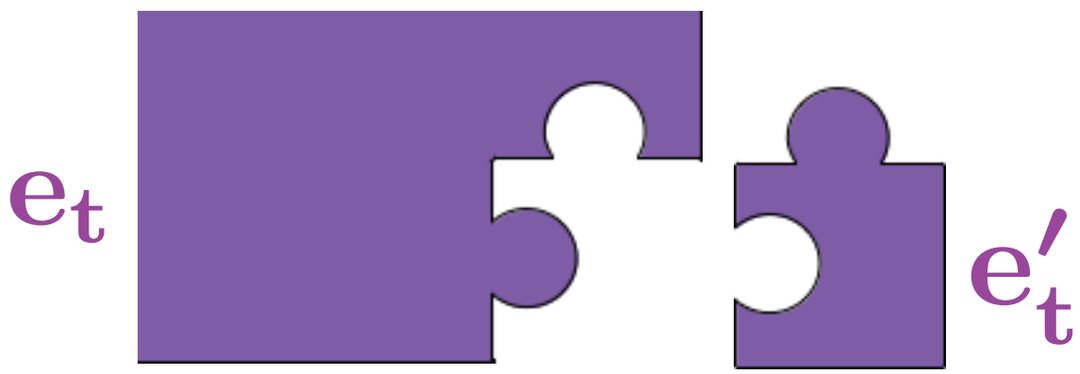
Multi-Lang. Approach: Linking ✓





Multi-Lang. Approach: Linking ✓



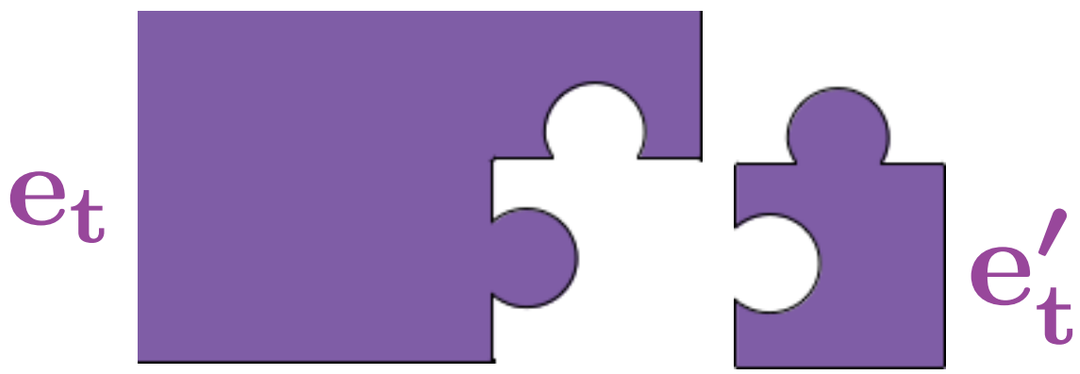


Multi-Lang. Approach: Linking ✓

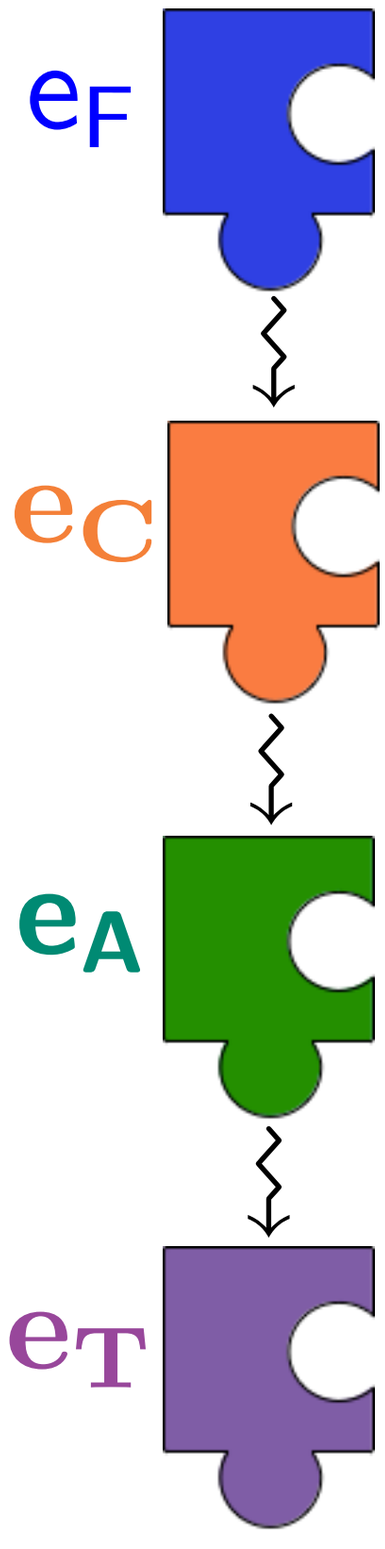
$$\mathcal{SIT}\,\mathbf{e'_t}$$

$$\mathcal{TIS}(\mathsf{e_s}\ (\mathcal{SIT}\,\mathbf{e'_t}))$$

$$\approx^{ctx}\ \mathbf{e_t}\ \mathbf{e'_t}$$



Compiler Correctness: F to TAL

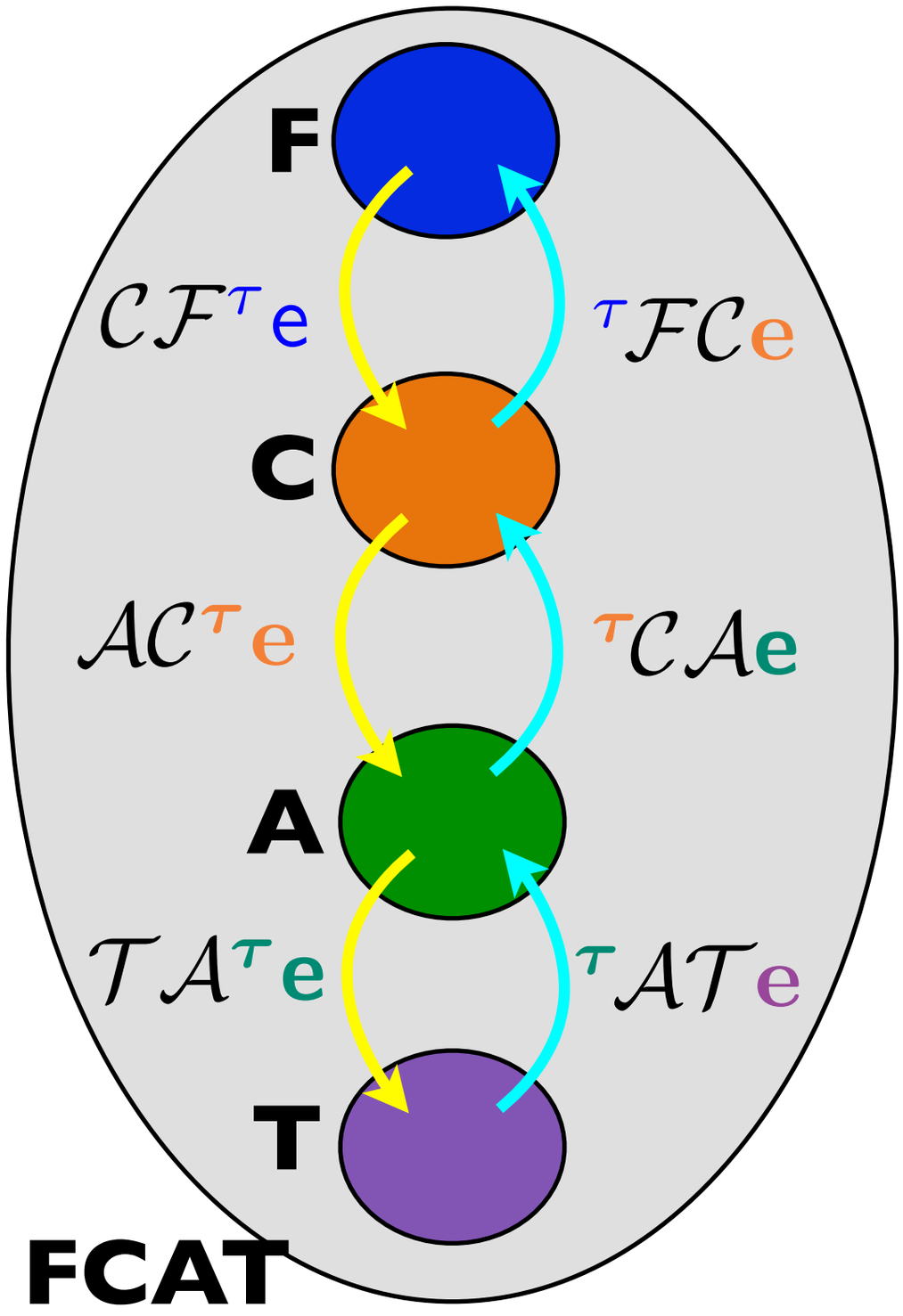


Closure Conversion τ^{C}

Allocation $\tau^{\mathcal{A}}$

Code Generation $\tau^{\mathcal{T}}$

Combined language **FCAT**



[Perconti-Ahmed'14]
[Patterson et al.'17]

- Boundaries mediate between

  $\tau \,\&\, \tau^{\mathcal{C}}$ $\tau \,\&\, \tau^{\mathcal{A}}$ $\tau \,\&\, \tau^{\mathcal{T}}$

- Operational semantics

  $\mathcal{C}\mathcal{F}^{\tau}\,\mathsf{e} \longmapsto^{*} \mathcal{C}\mathcal{F}^{\tau}\,\mathsf{v} \longmapsto \mathsf{v}$

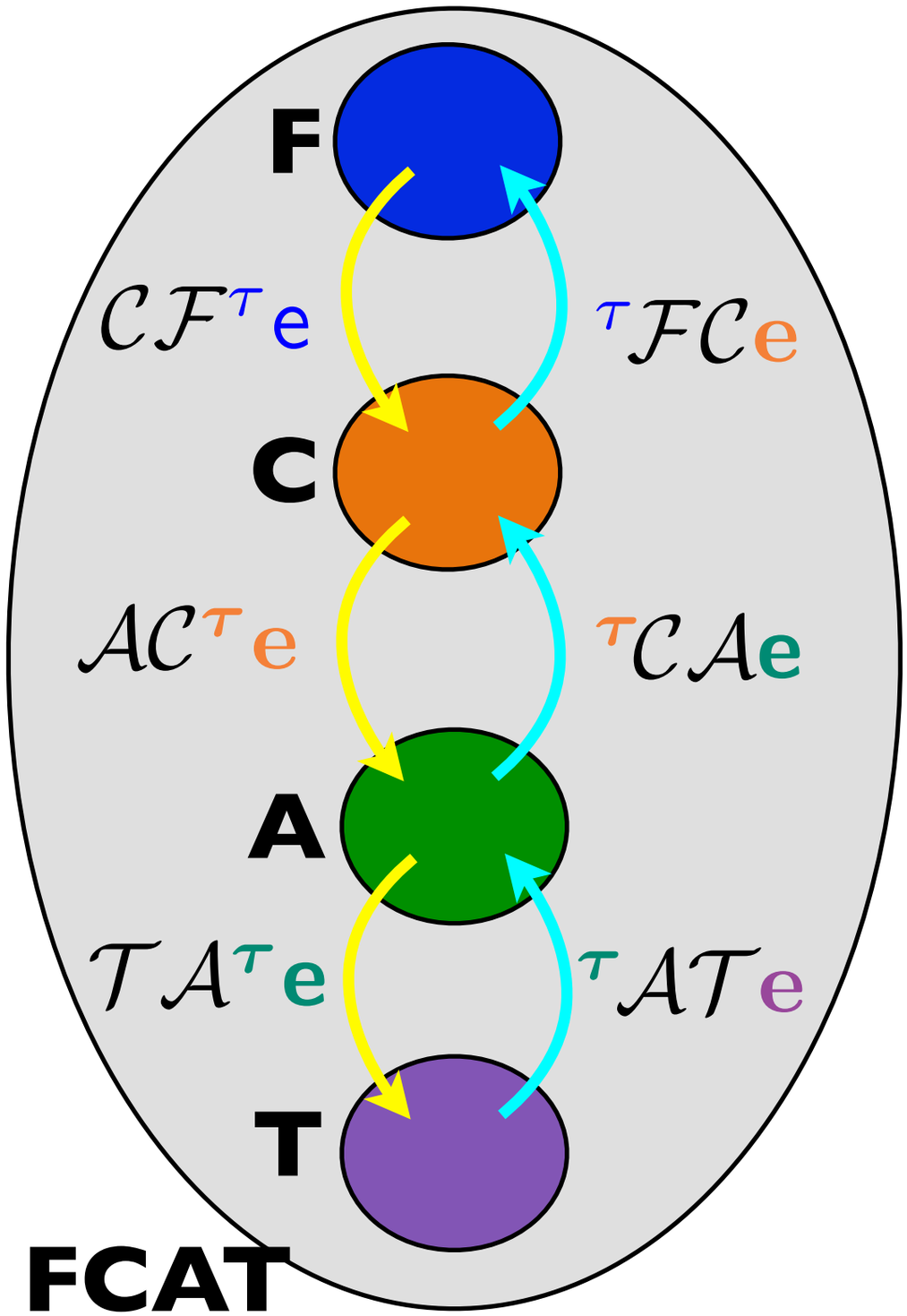
  ${}^{\tau}\mathcal{F}\mathcal{C}\,\mathsf{e} \longmapsto^{*} {}^{\tau}\mathcal{F}\mathcal{C}\,\mathsf{v} \longmapsto \mathsf{v}$

- Boundary cancellation

  ${}^{\tau}\mathcal{F}\mathcal{C}\mathcal{C}\mathcal{F}^{\tau}\,\mathsf{e} \approx^{ctx} \mathsf{e} : \tau$

  $\mathcal{C}\mathcal{F}^{\tau}\,{}^{\tau}\mathcal{F}\mathcal{C}\,\mathsf{e} \approx^{ctx} \mathsf{e} : \tau^{\mathcal{C}}$

Challenges

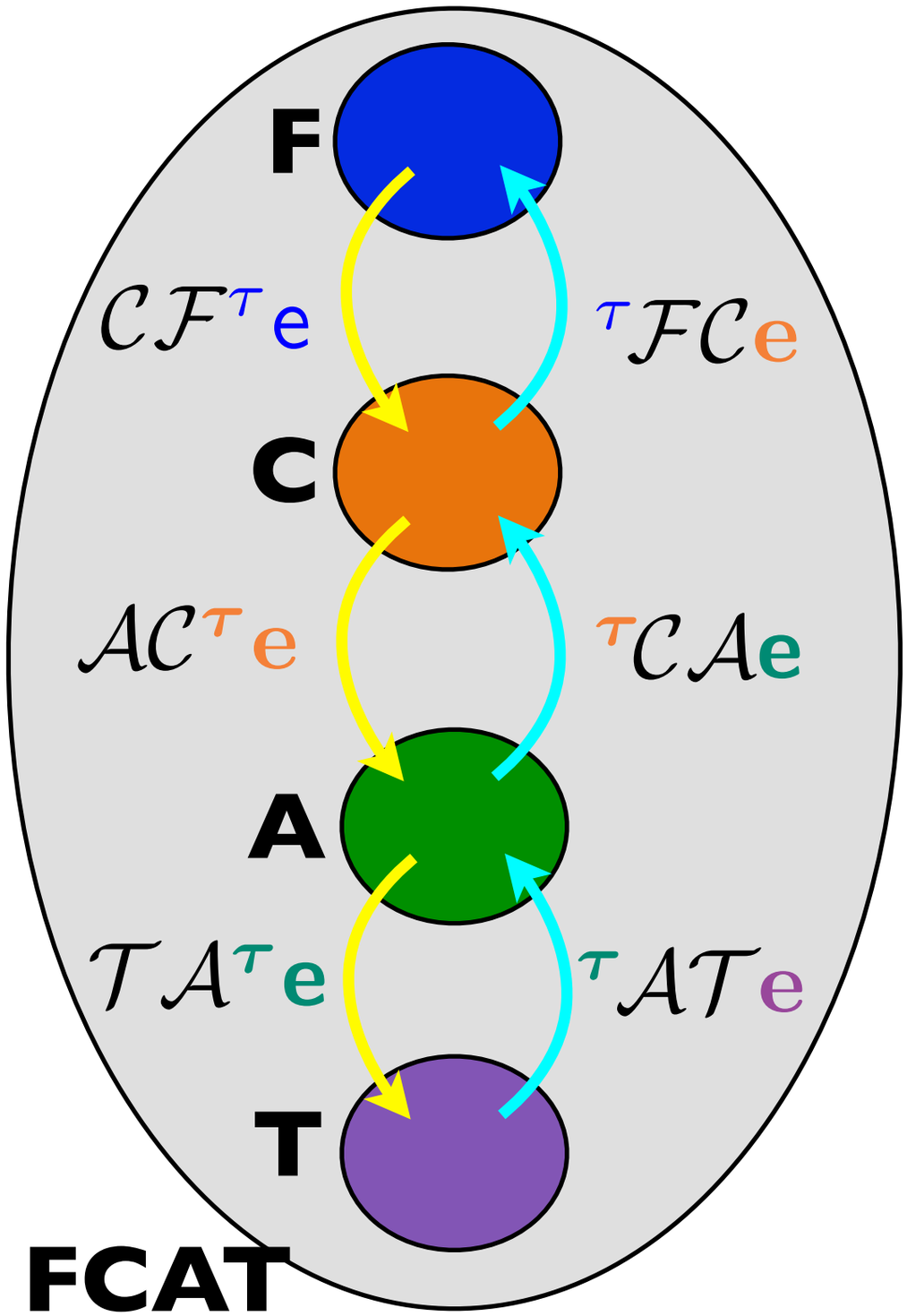


F+C: Interoperability semantics with type abstraction in both languages

C+A: Interoperability when compiler pass allocates code & tuples on heap

A+T: What is e? What is v? How to define contextual equiv. for TAL *components*? How to define logical relation?

Challenges



F+C: Interoperability semantics with type abstraction in both languages

C+A: Interoperability when compiler pass allocates code & tuples on heap

A+T: What is e? What is v? How to define contextual equiv. for TAL *components*? How to define logical relation?

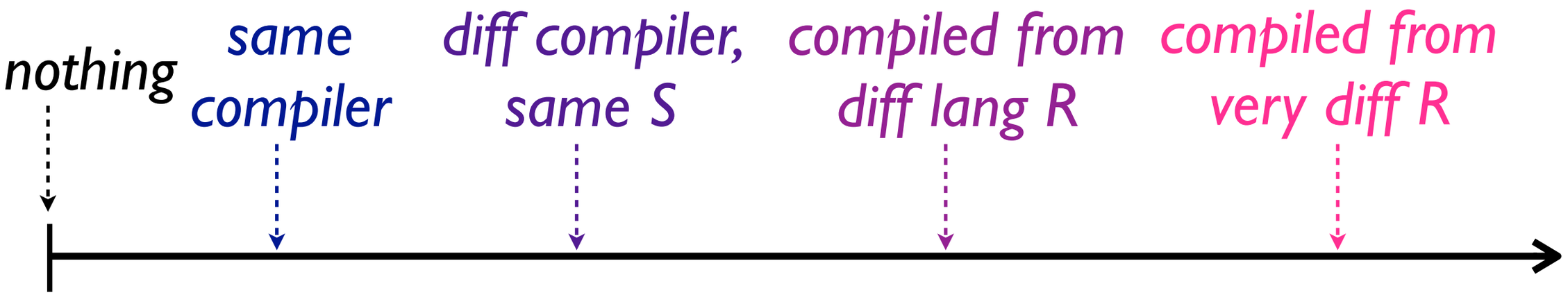
Central Challenge: interoperability between high-level (direct-style) language & assembly (continuation style)

FunTAL: Reasonably Mixing a Functional Language with Assembly *[Patterson et al. PLDI'17]*

CompCompCert vs. Multi-language

Transitivity:	
- structured simulations	- all passes use multi-lang $\approx^{ctx}$
Check okay-to-link-with:	
- satisfies CompCert memory model	- satisfies expected type (translation of source type)
Contexts:	
- semantic representation	- syntactic representation
Requires uniform memory model across compiler IRs?	
- yes	- no

What we can link with



CompCert

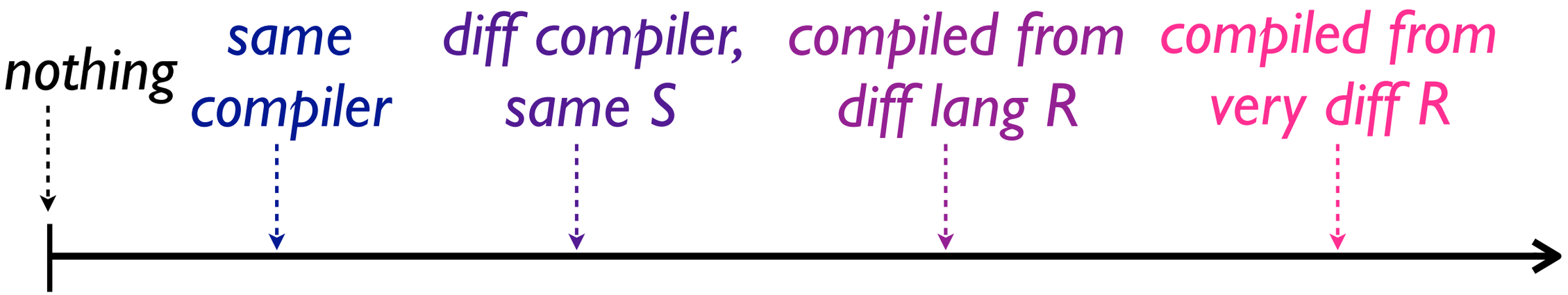
SepCompCert
Kang et al.'16

Pilsner
Neis et al.'15

Compositional CompCert
Stewart et al.'15

Allows linking with behavior inexpressible in S → *Multi-language ST*
Perconti-Ahmed'14

Proving Transitivity



CompCert

SepCompCert
Kang et al.'16

Pilsner
Neis et al.'15

Compositional CompCert
Stewart et al.'15

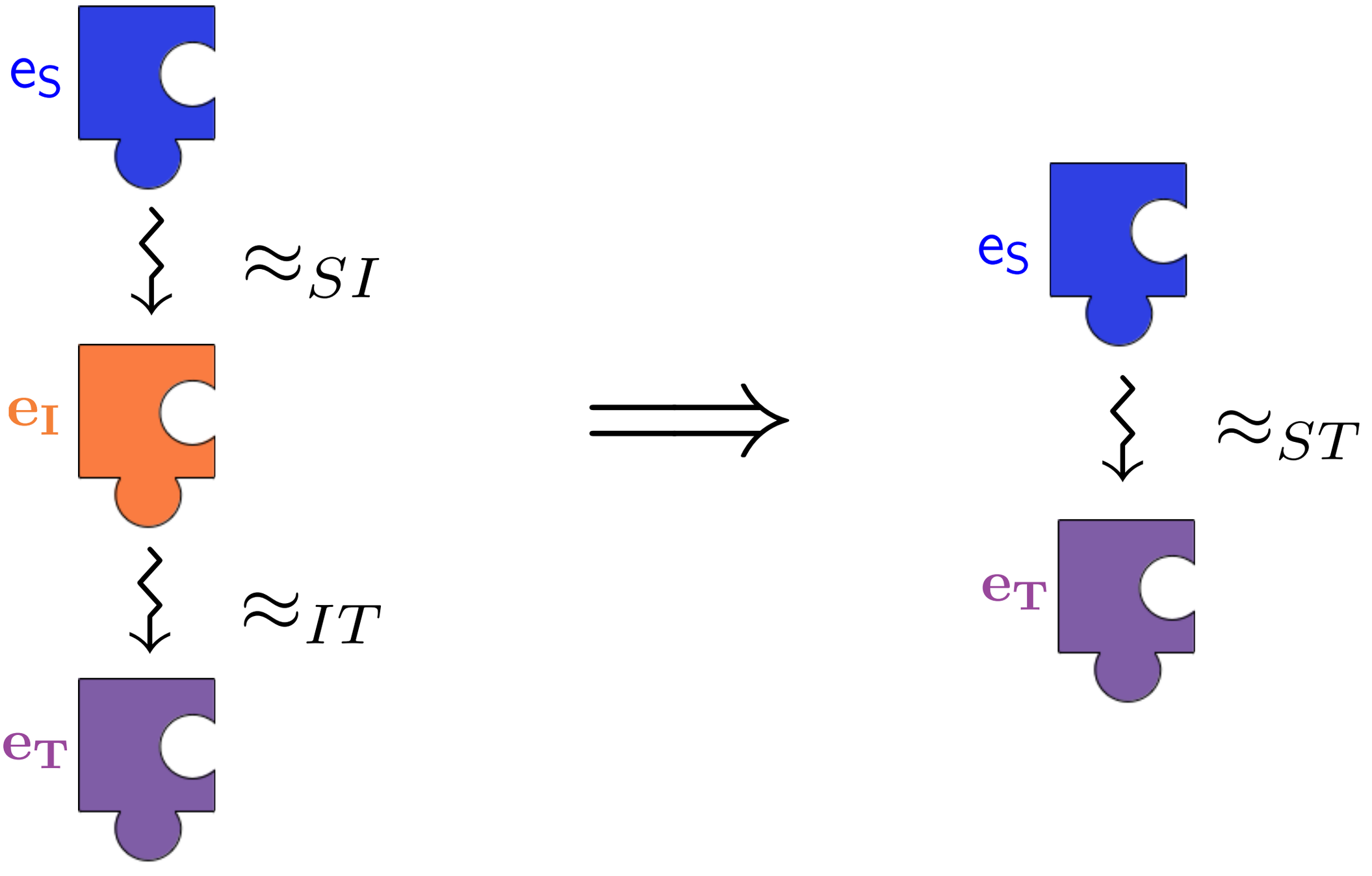
Multi-language ST
Perconti-Ahmed'14

Transitivity requires effort / engineering

Vertical Compositionality

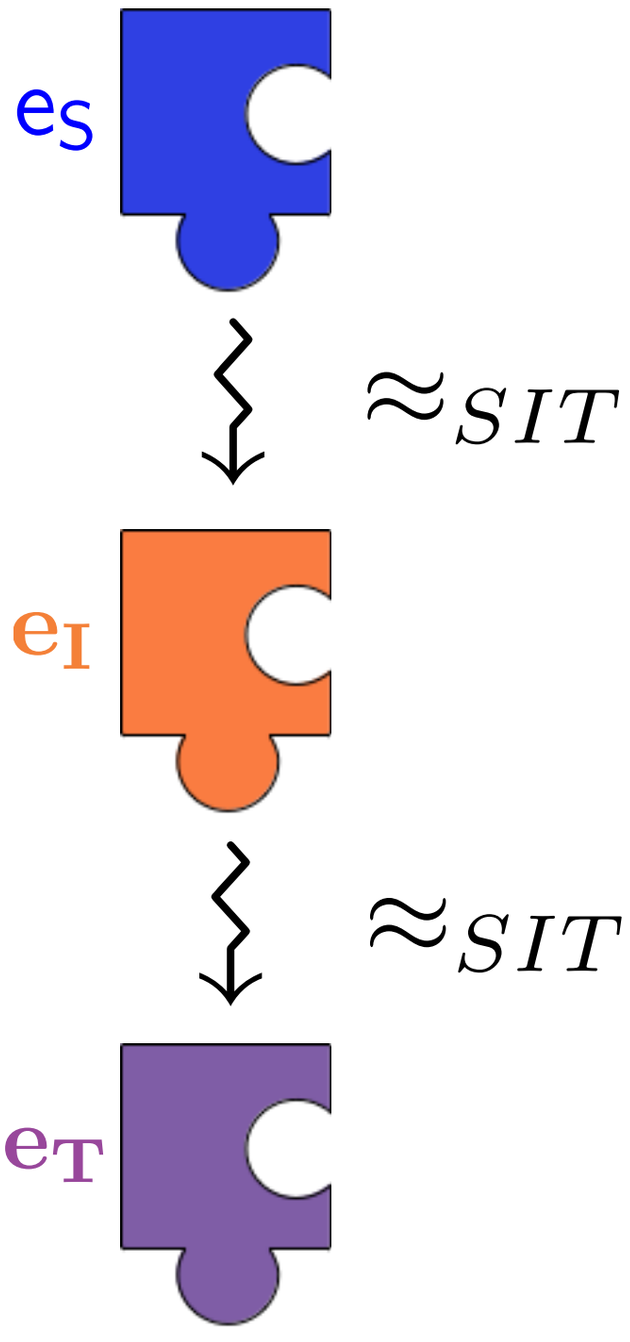
Transitivity

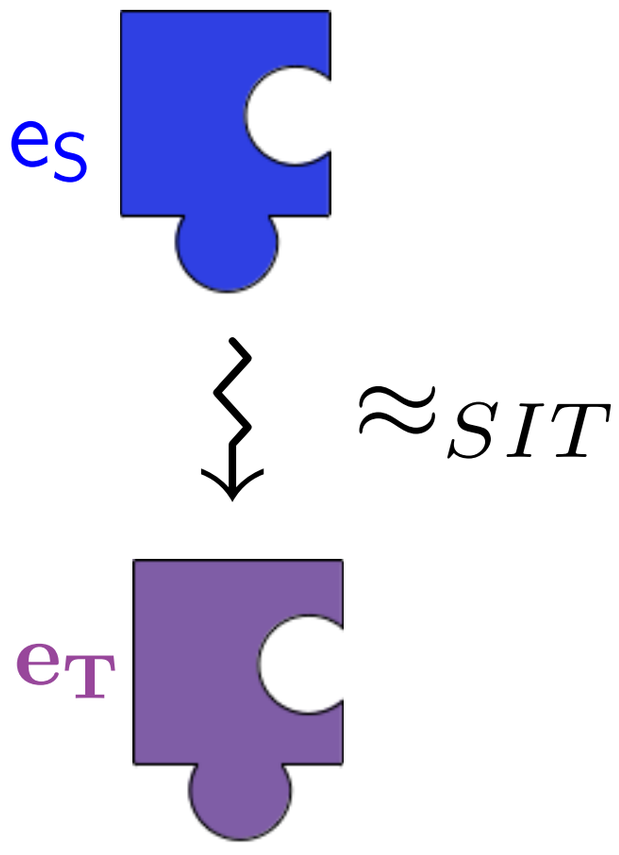
Vertical Compositionality



Transitivity

CompCompCert & Multi-lang





Horizontal Compositionality

Source-Independent Linking

Vertical Compositionality

Transitivity

Horizontal Compositionality

Pilsner
Neis et al.'15

Source-Independent Linking

Compositional CompCert
Stewart et al.'15

Multi-language ST
Perconti-Ahmed'14

Vertical Compositionality

Pilsner
Neis et al.'15

Transitivity

Compositional CompCert
Stewart et al.'15

Multi-language ST
Perconti-Ahmed'14

To Understand if Theorem is Correct…

Pilsner
Neis et al.'15

- source-target PILS

Compositional CompCert
Stewart et al.'15

Multi-language ST
Perconti-Ahmed'14

To Understand if Theorem is Correct…

Pilsner
Neis et al.'15

- source-target PILS

Compositional CompCert
Stewart et al.'15

- interaction semantics
& structured simulations

Multi-language ST
Perconti-Ahmed'14

- source-target multi-language

To Understand if Theorem is Correct…

Pilsner
Neis et al.'15

- source-target PILS

Compositional CompCert
Stewart et al.'15

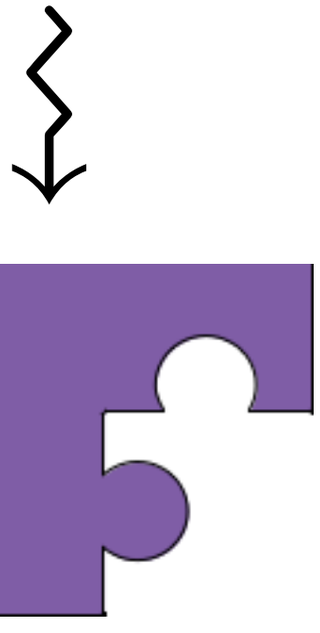
- interaction semantics
& structured simulations

Multi-language ST
Perconti-Ahmed'14

- source-target multi-language

Is there a generic CCC theorem?

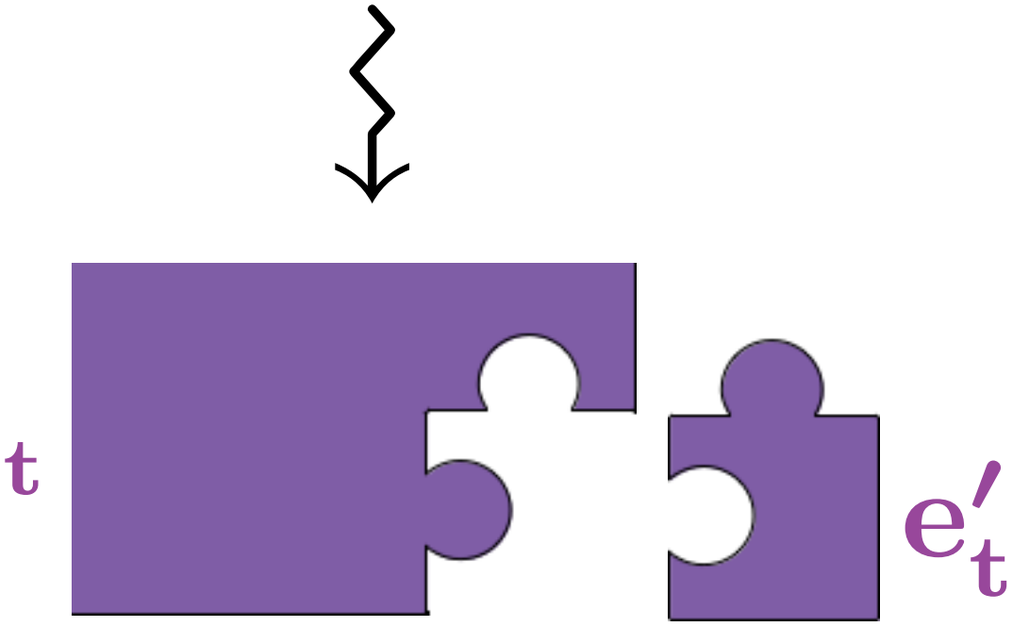
Generic CCC Theorem?



$e_S \approx e_T$

expressed how?

Generic CCC Theorem?



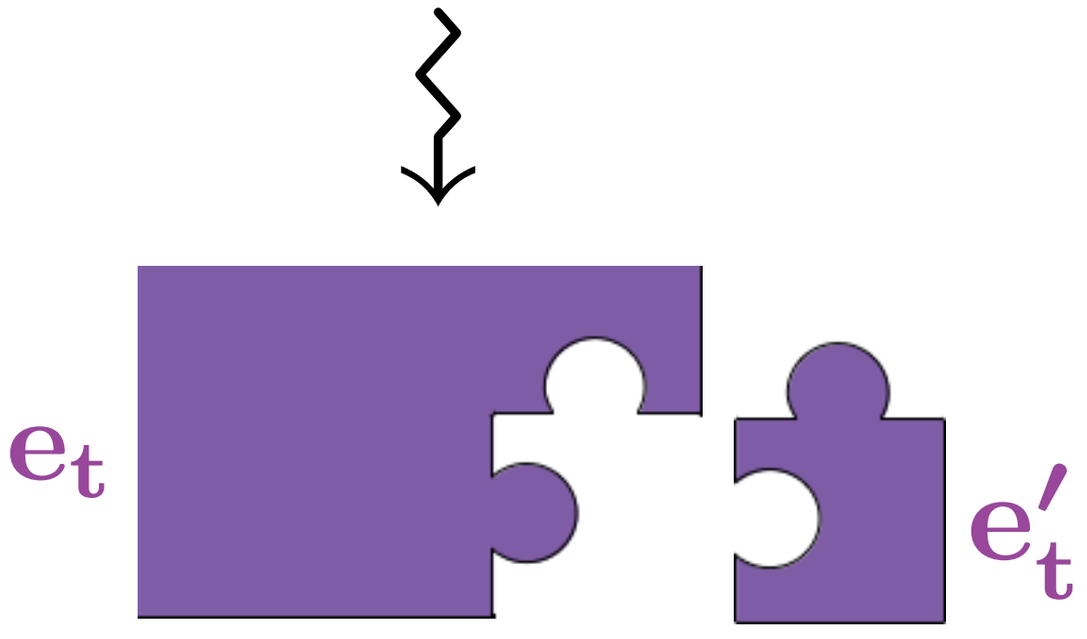
$e_S \approx e_T$

expressed how?

Generic CCC Theorem?

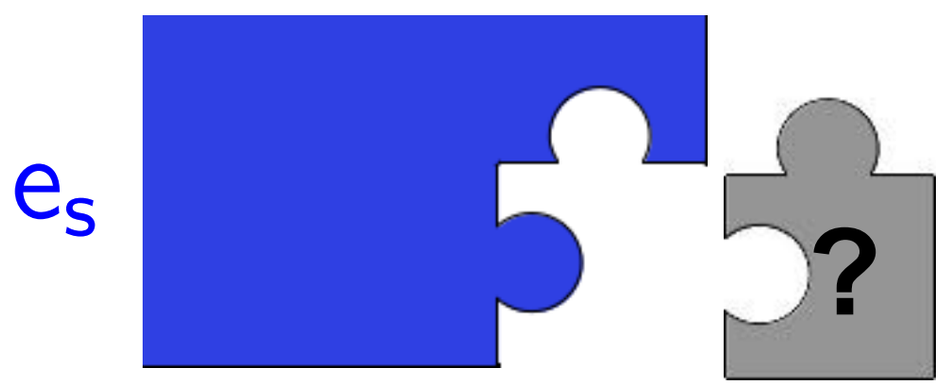
$e_S \approx e_T$

expressed how?



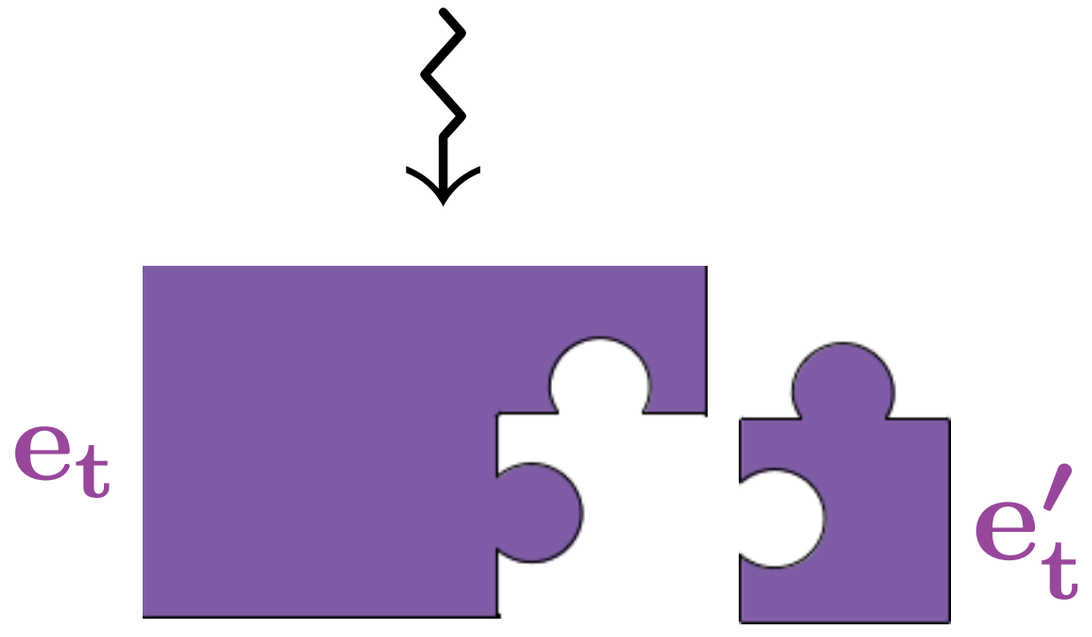
$\in \mathcal{L}$ ⟵ *linking set*

Generic CCC Theorem?



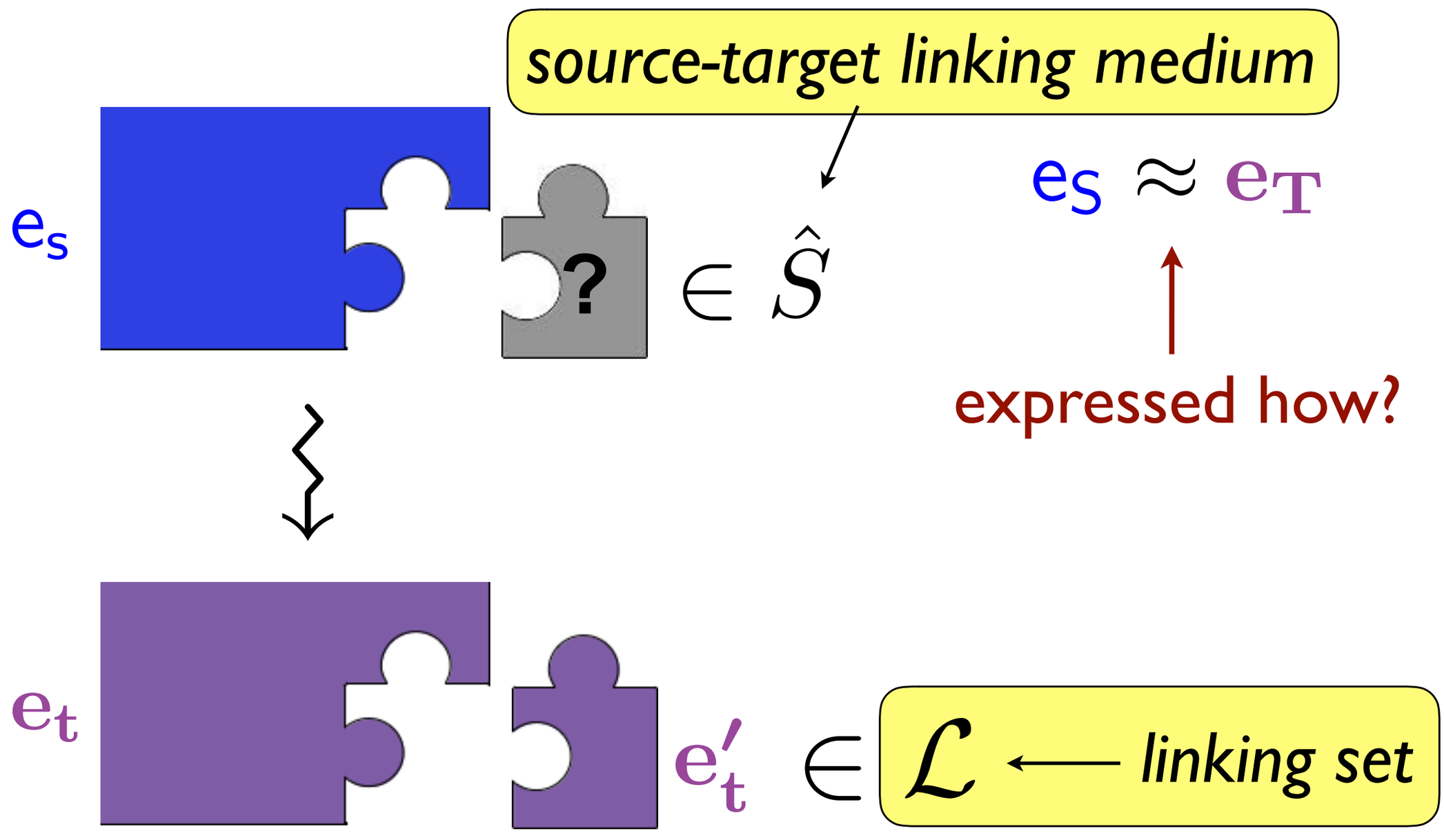
$e_S \approx e_T$

expressed how?

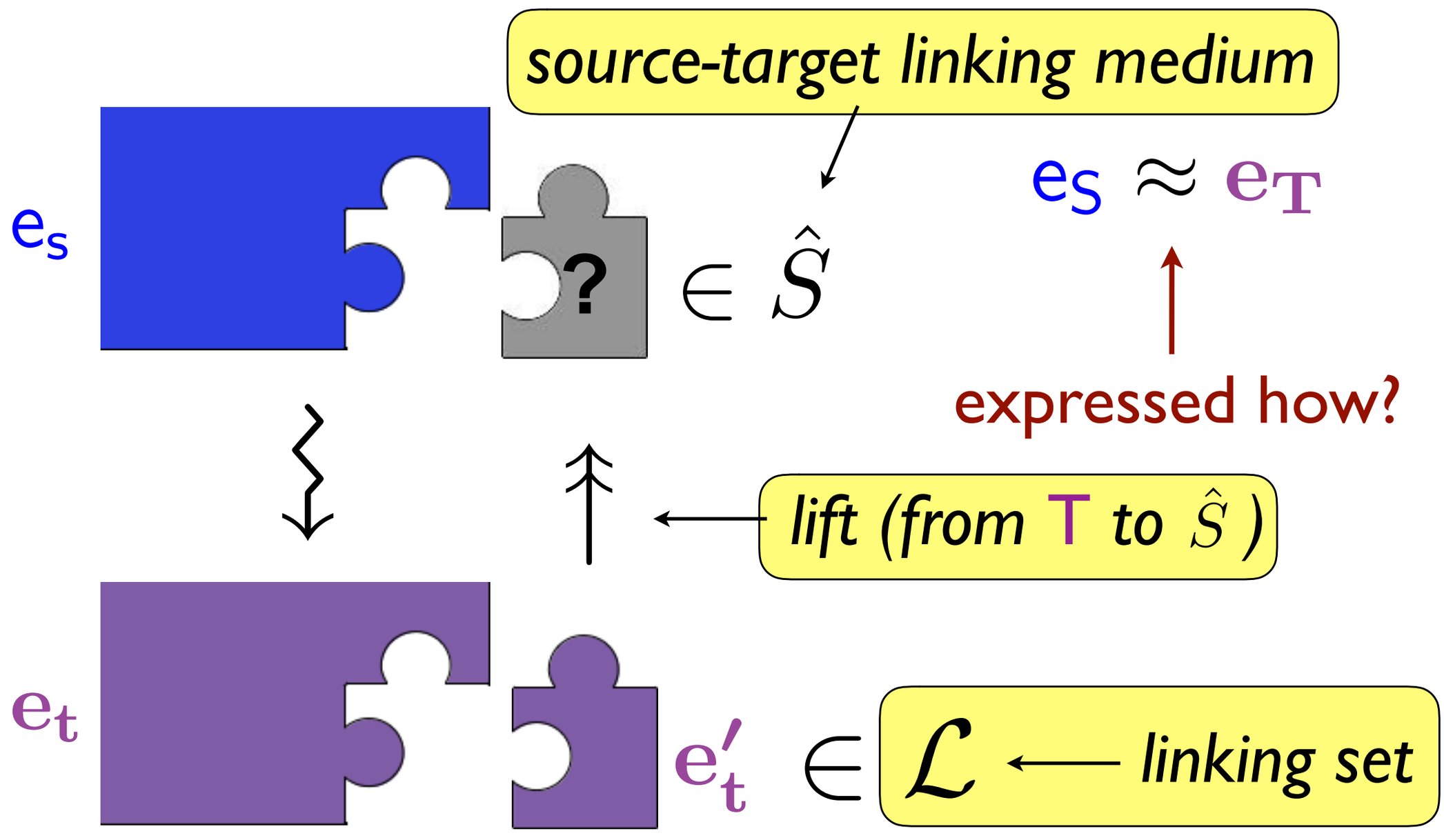


$\in \mathcal{L}$ ⟵ *linking set*

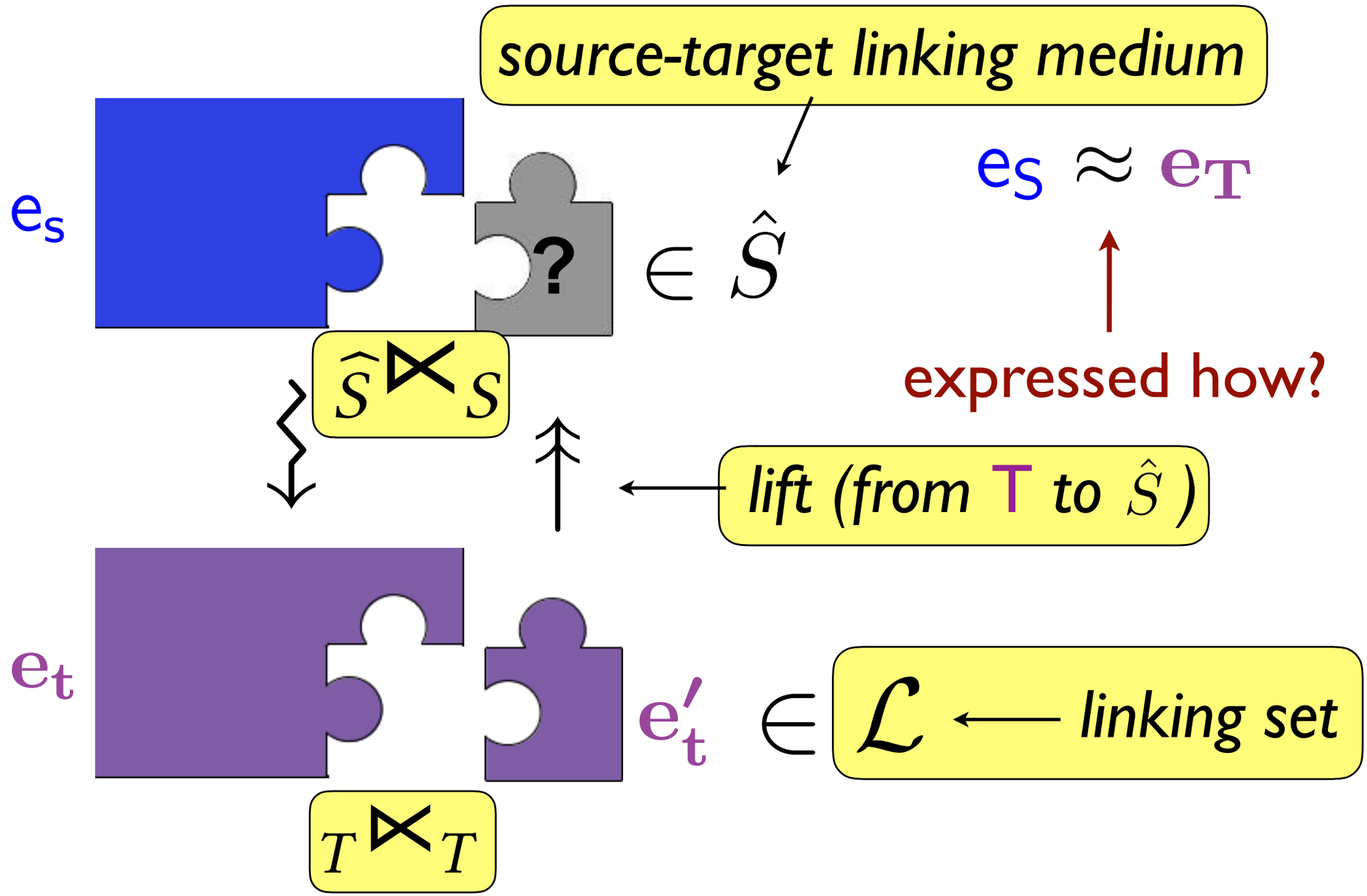
Generic CCC Theorem?



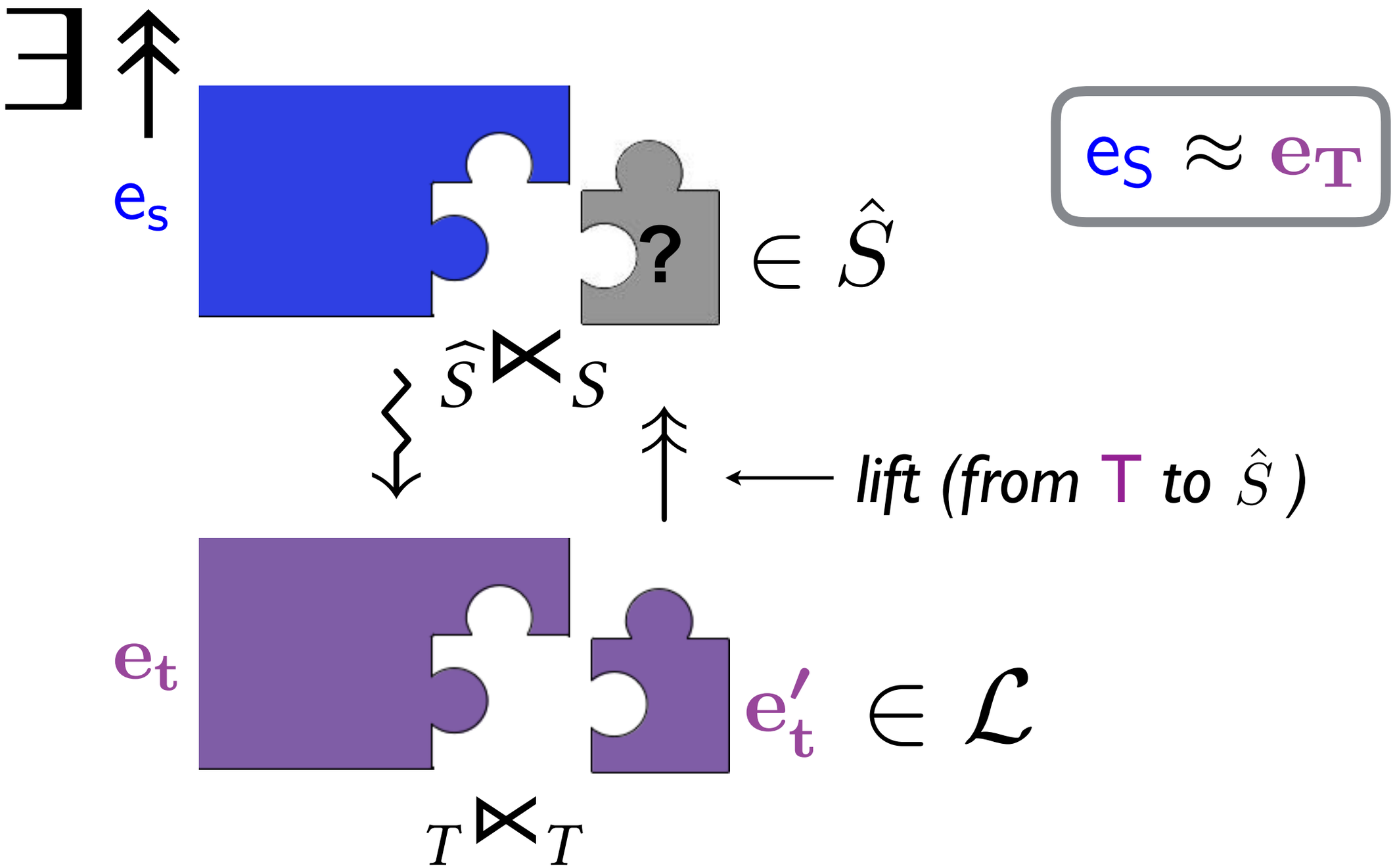
Generic CCC Theorem?



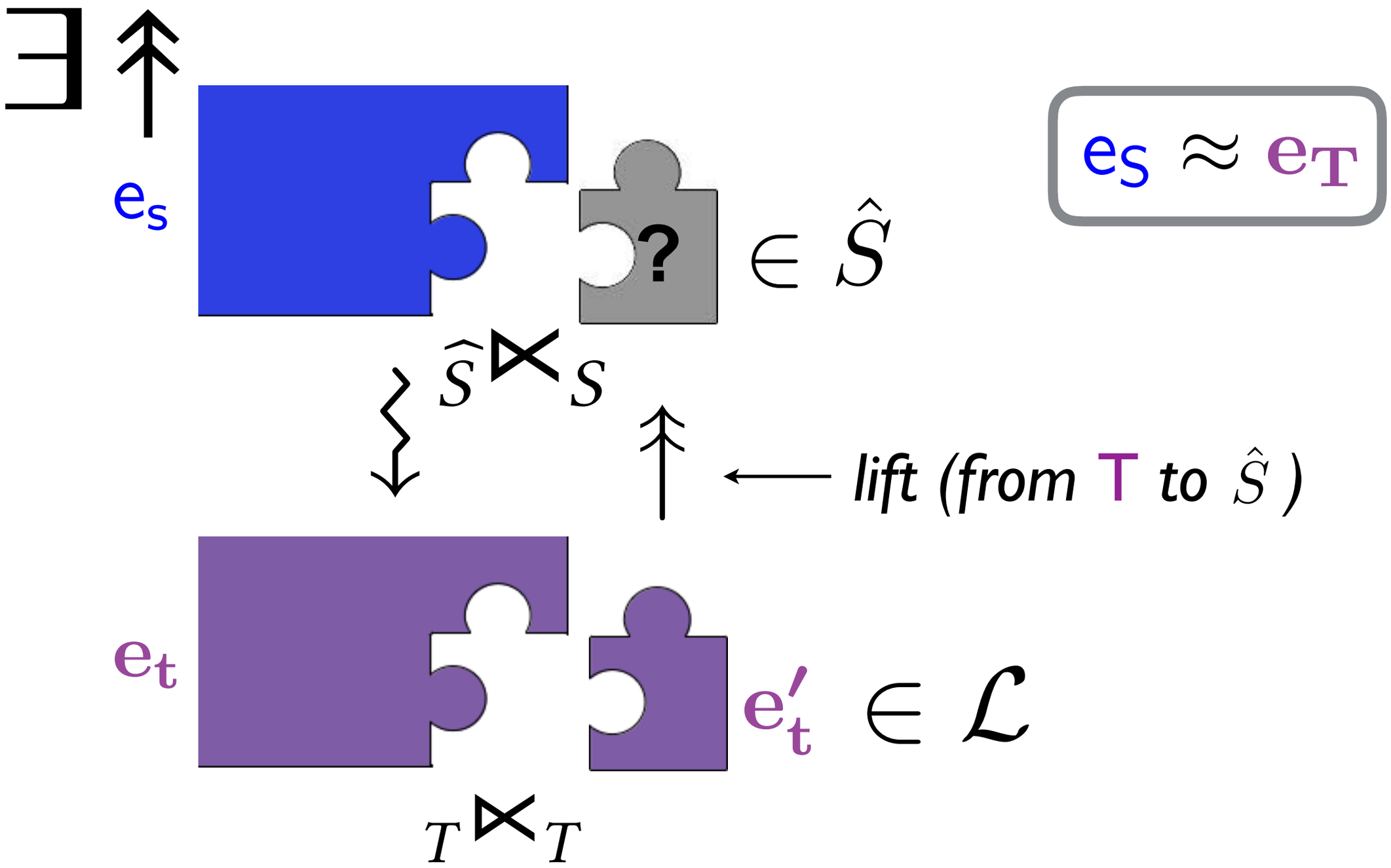
Generic CCC Theorem?



Generic CCC Theorem



Generic CCC Theorem



…and "lift" is inverse of "compile" on compiler output

Generic CCC Theorem

$$\exists \twoheaduparrow.\ \forall e_S, e_T.\ e_S \rightsquigarrow e_T \implies$$

$$\forall (e'_T, \varphi) \in \mathcal{L}.\ ok_{\ltimes}(e'_T, e_T) \implies$$

$$e'_T\ {}_T\ltimes_T\ e_T \implies {}_T\sqsubset_{\widehat{S}}\ \twoheaduparrow(e'_T, \varphi)\ \widehat{S}\ltimes_S\ e_S$$

…and "lift" is inverse of "compile" on compiler output

$$\forall (e'_T, \varphi) \in \mathcal{L}.\ \forall e_S, e_T.\ e_S \rightsquigarrow e_T \implies$$

$$e'_T\ {}_T\sqsubseteq_T\ e_T \implies \twoheaduparrow(e'_T, \varphi)\ \widehat{S}\sqsubseteq_S\ e_S$$

CCC Properties

Implies whole-program compiler correctness & correct separate compilation

Can be instantiated with different formalisms…

CCC with Pilsner

$\mathcal{L}$	$\{(e_T, \varphi) \mid \varphi = \text{source component } e_S \;\&\; \text{proof that } e_S \simeq e_T\}$
$\widehat{S}$	unchanged source language S
$\widehat{S}{\ltimes}_S$	unchanged source language linking
$\widehat{S}{\sqsubset}_S$	source language (whole program) observational equivalence
$\twoheaduparrow(\cdot)$	$\twoheaduparrow(e_T, (e_S, \_)) = e_S$

CCC with Multi-language

$\mathcal{L}$ $\{(e_T, \_) \mid$ where e_T is any target component $\}$

$\widehat{S}$ source-target multi-language ST

$\widehat{S}{\ltimes}_S$ $e\ {}_{ST}{\ltimes}_{ST}\ e_S$

$\widehat{S}{\sqsubset}_S$ run $\widehat{S}$ according to multi-lang ST, compare with running S

$\twoheaduparrow(\cdot)$ $\twoheaduparrow(e_T, \_) = \mathcal{ST}(e_T)$

Vertical Compositionality for Free

when $\Uparrow_{ST} = \Uparrow_{SI} \circ \Uparrow_{IT}$

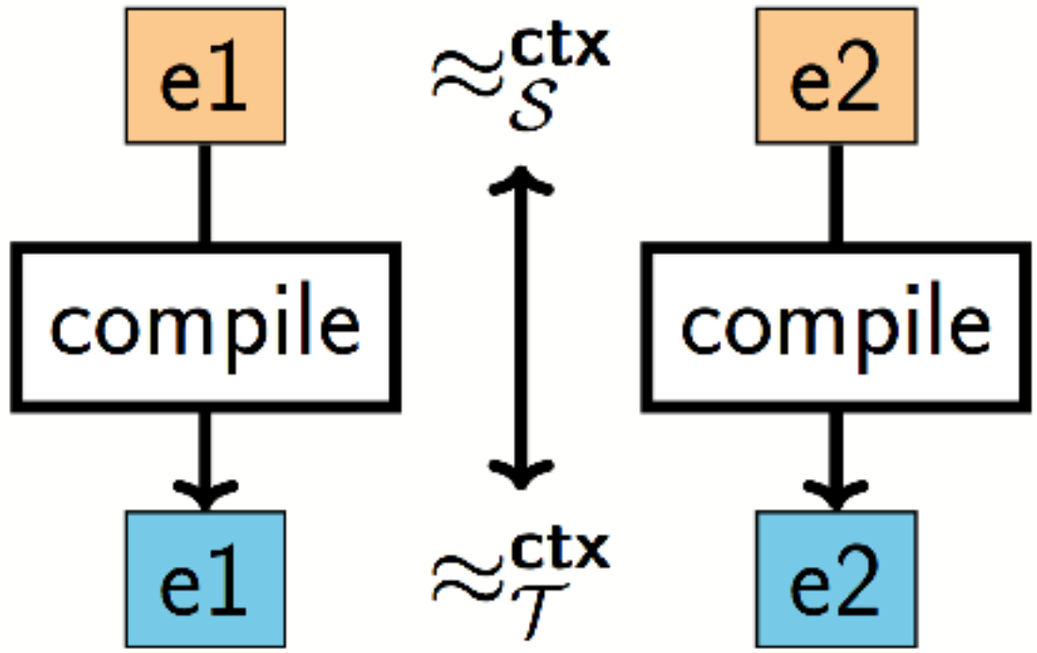
i.e., when lift $\Uparrow$ is a back-translation that maps every $e_T \in \mathcal{L}$ to some e_S

Fully abstract compilers have such back-translations!

Bonus of vertical comp: can verify different passes using different formalisms to instantiate CCC

Fully Abstract Compilers

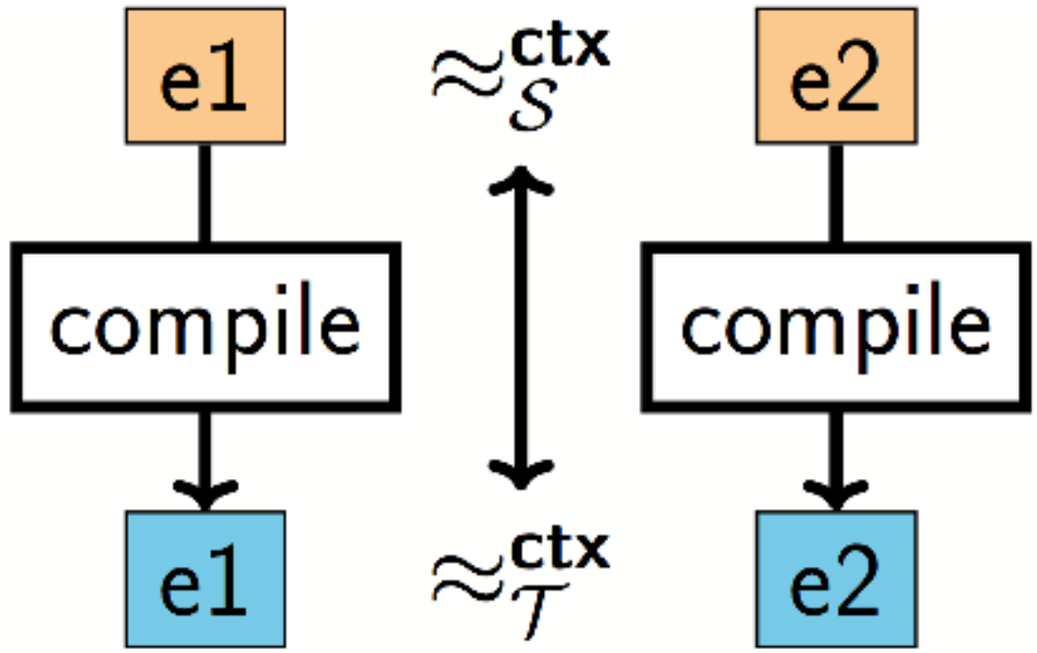
preserve equivalence



- ensure a compiled component does not interact with any target behavior that is inexpressible in S
- Do we want to link with behavior inexpressible in S? Or do we want fully abstract compilers?

Fully Abstract Compilers

preserve equivalence



- ensure a compiled component does not interact with any target behavior that is inexpressible in S
- Do we want to link with behavior inexpressible in S? Or do we want fully abstract compilers?

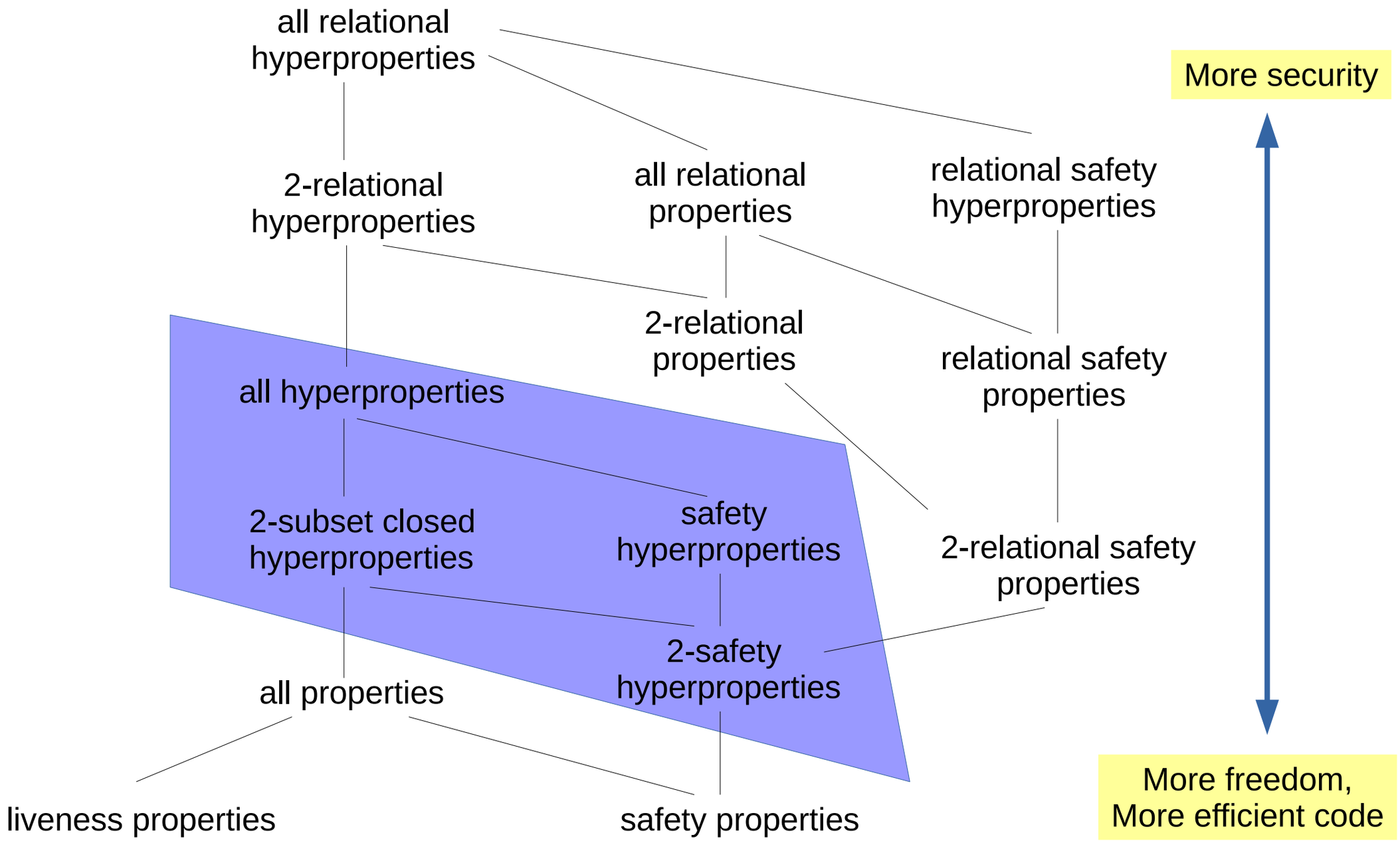
We want both!

Conclusion

- Common attacker model: low-level contexts that compiled code is linked with
- Compositional Compiler Correctness
  - allows linking with low-level code of arbitrary provenance
  - but source-independent linking and vertical compositionality are at odds

Relation to Deepak's talk...

A compiler is secure for X if it robustly preserves all X, where X is:



Relation to Deepak's talk...

Vertical compositionally needs back-translation that can only depend on A', or on A' and P, but not on traces t.

- Compiler security for all relational hyperproperties

  $\forall A'.\ \exists A.\ \forall P.\ \mathrm{Tr}(A' \,\|\, (P\downarrow)) = \mathrm{Tr}(A \,\|\, P)$

- Compiler security for all hyperproperties

  $\forall A'\ P.\ \exists A.\ \mathrm{Tr}(A' \,\|\, (P\downarrow)) = \mathrm{Tr}(A \,\|\, P)$

Relation to Deepak's talk...

Vertically compositional compiler correctness needs back-translation that can *only* depend on A', or on A' and P, but *NOT* on traces t.

Hence, *NOT* compatible with:

- Compiler security for safety

$$\forall A'\ P\ t.\ (t \in \mathrm{Fin}(A' \parallel (P\downarrow))) \Rightarrow \exists A.\ t \in \mathrm{Fin}(A \parallel P)$$

Vertically compositional compiler correctness seems unachievable for certain secure compilers -- those that wrap compiled code with fewer dynamic checks in an effort to attain "weaker but more efficient" secure compilation.