

# Noninterference for Free

William J. Bowman and Amal Ahmed



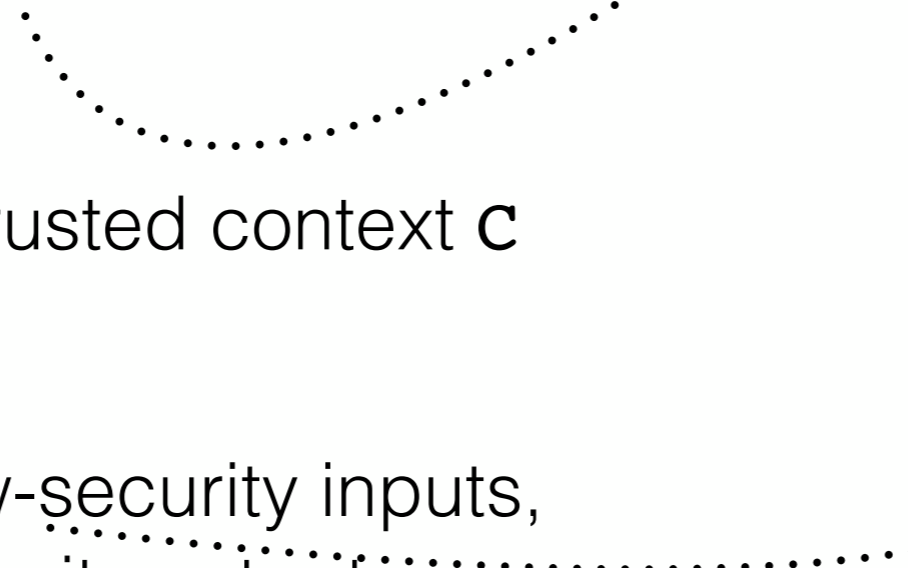
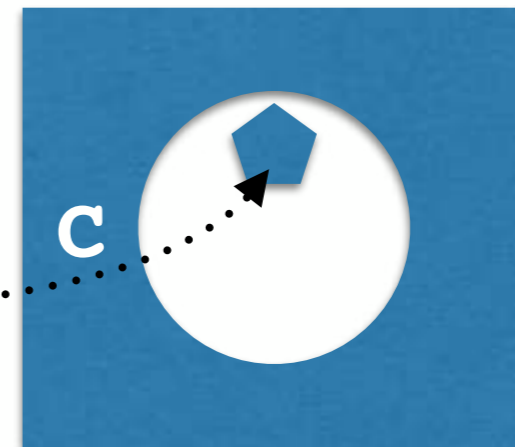
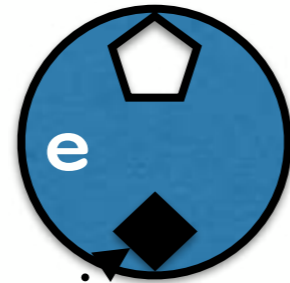
# Let's write a secure program

- Want to write a component  $e$  (browser)
- Manages high-security data (passwords)

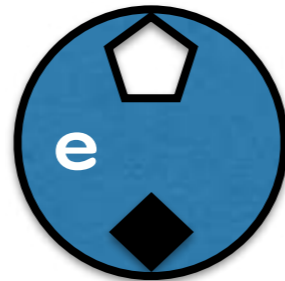
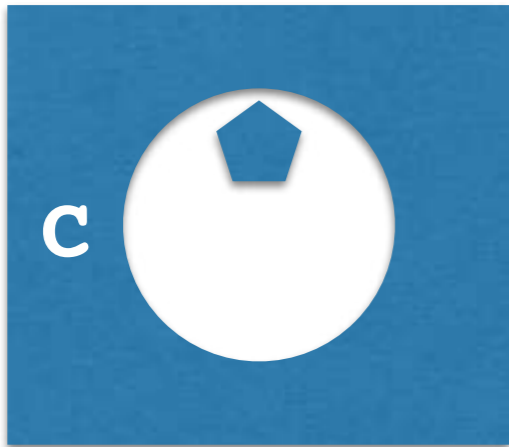


# Let's write a secure program

- Want to write a component **e** (browser)
- Manages high-security data (passwords)
- Links with untrusted context **c** (plugins)
- **c** provides low-security inputs, reads low-security outputs

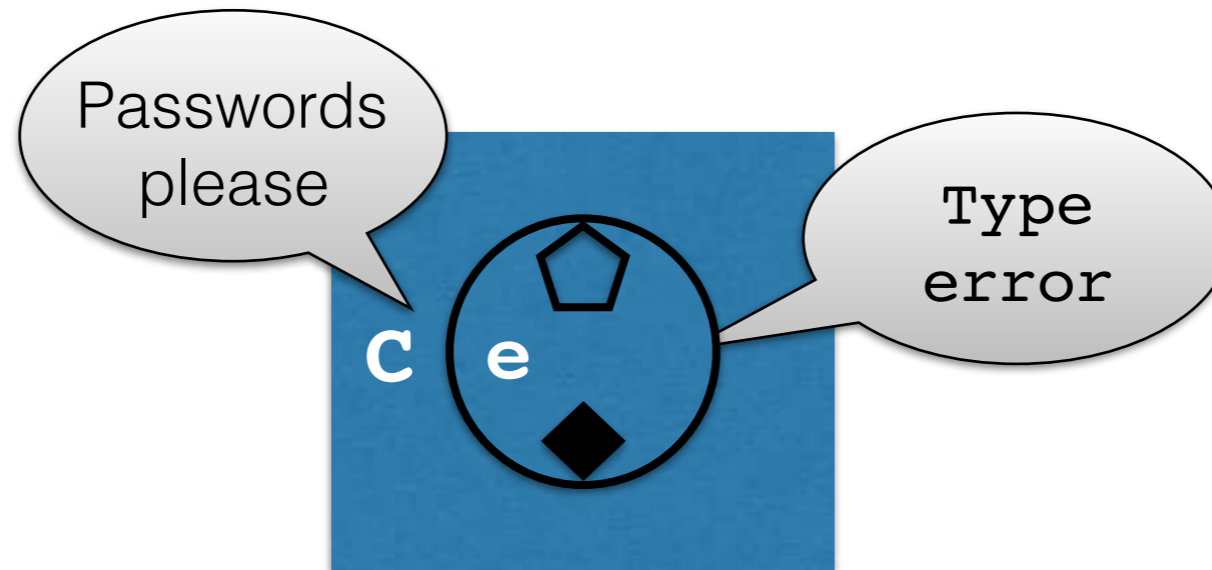


# Language-based security!



Using language-based security,  
we statically rule out attacks

# Language-based security!

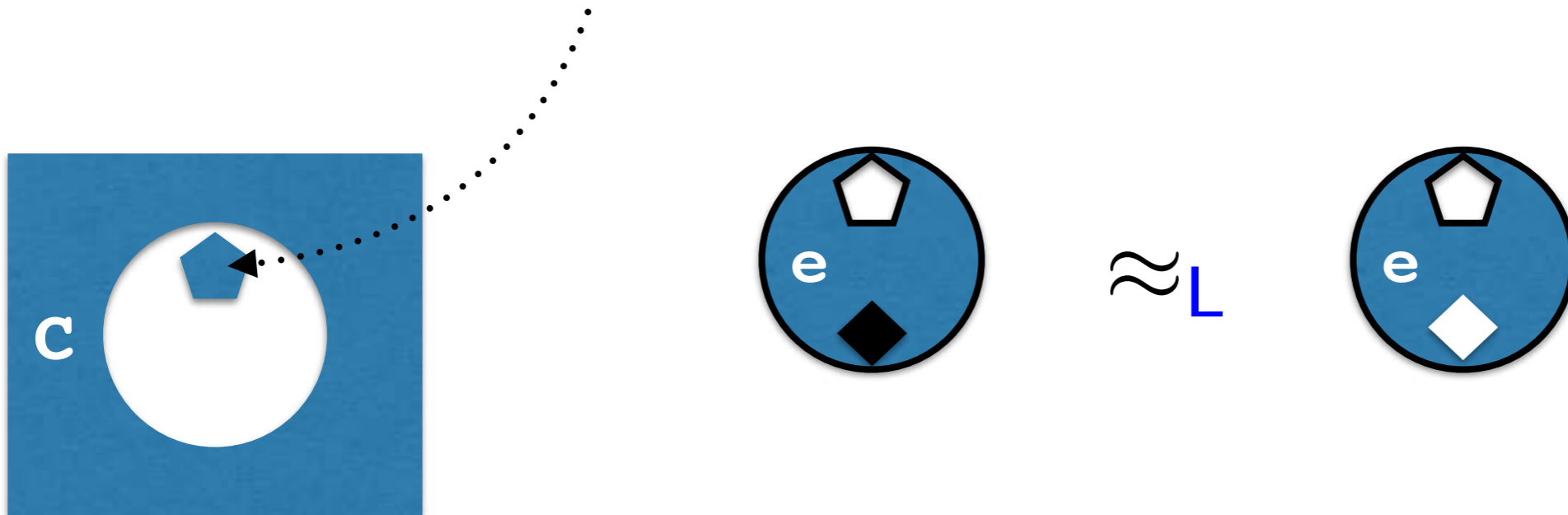


Using language-based security,  
we statically rule out attacks

# Noninterference

**Noninterference** is an **equivalence property**  
of any well-typed term  $e$ :

Given **same** low-level (**public**) inputs,

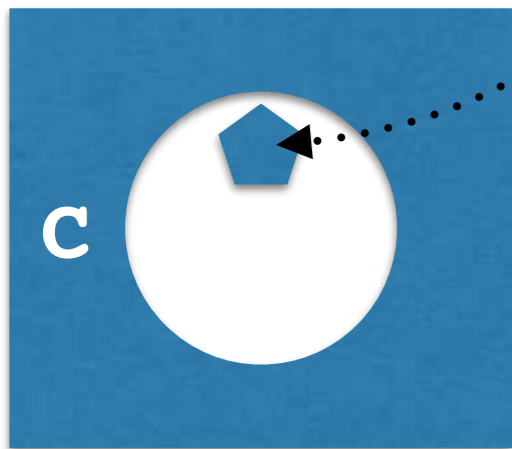


# Noninterference

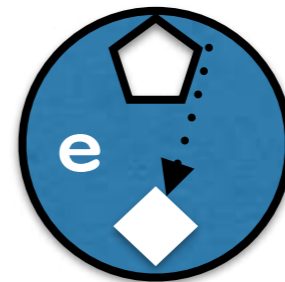
**Noninterference** is an **equivalence property**  
of any well-typed term  $e$ :

Given **same** low-level (**public**) inputs,

and **different** high-level (**private**) inputs



$\approx_L$

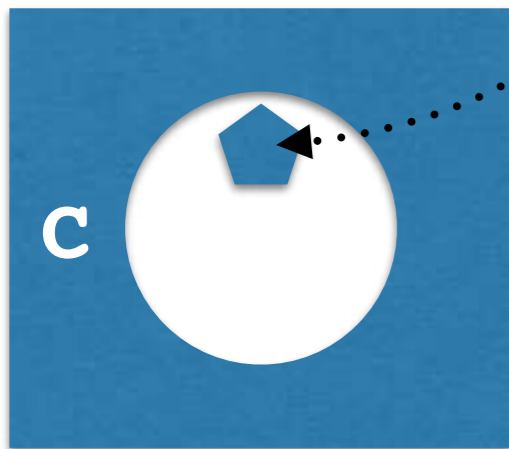


# Noninterference

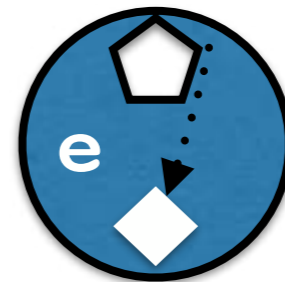
**Noninterference** is an **equivalence property** of any well-typed term  $e$ :

Given **same** low-level (**public**) inputs,

and **different** high-level (**private**) inputs



$\approx_L$



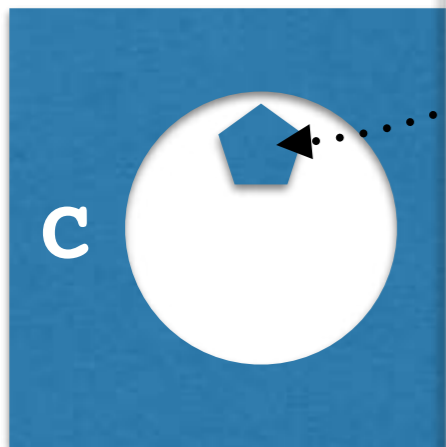
low-level outputs are indistinguishable



# Noninterference

**Noninterference** is an **equivalence property**  
of any well typed term  $e$ :

Given **same**



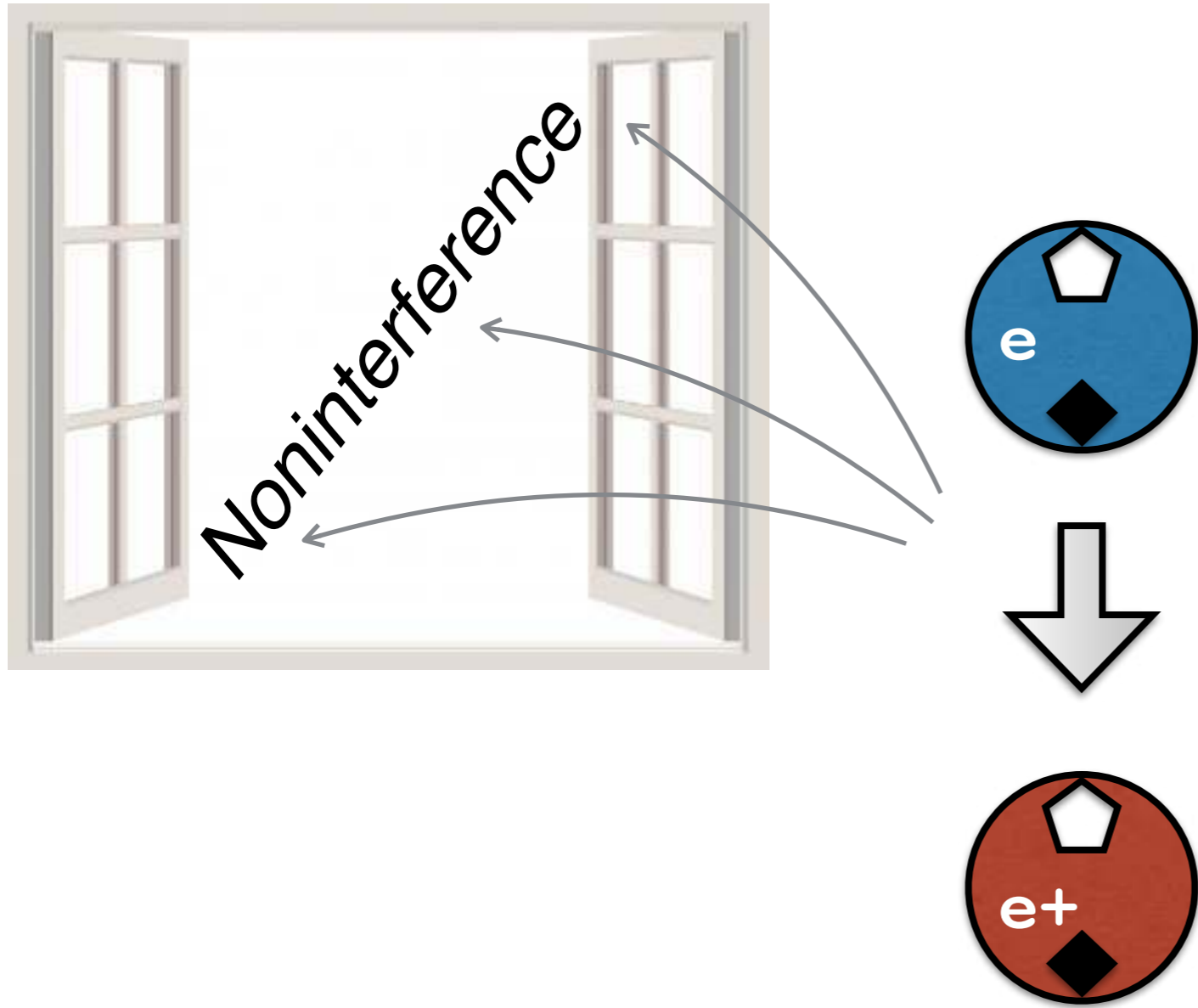
**vate**) inputs

Security Solved!

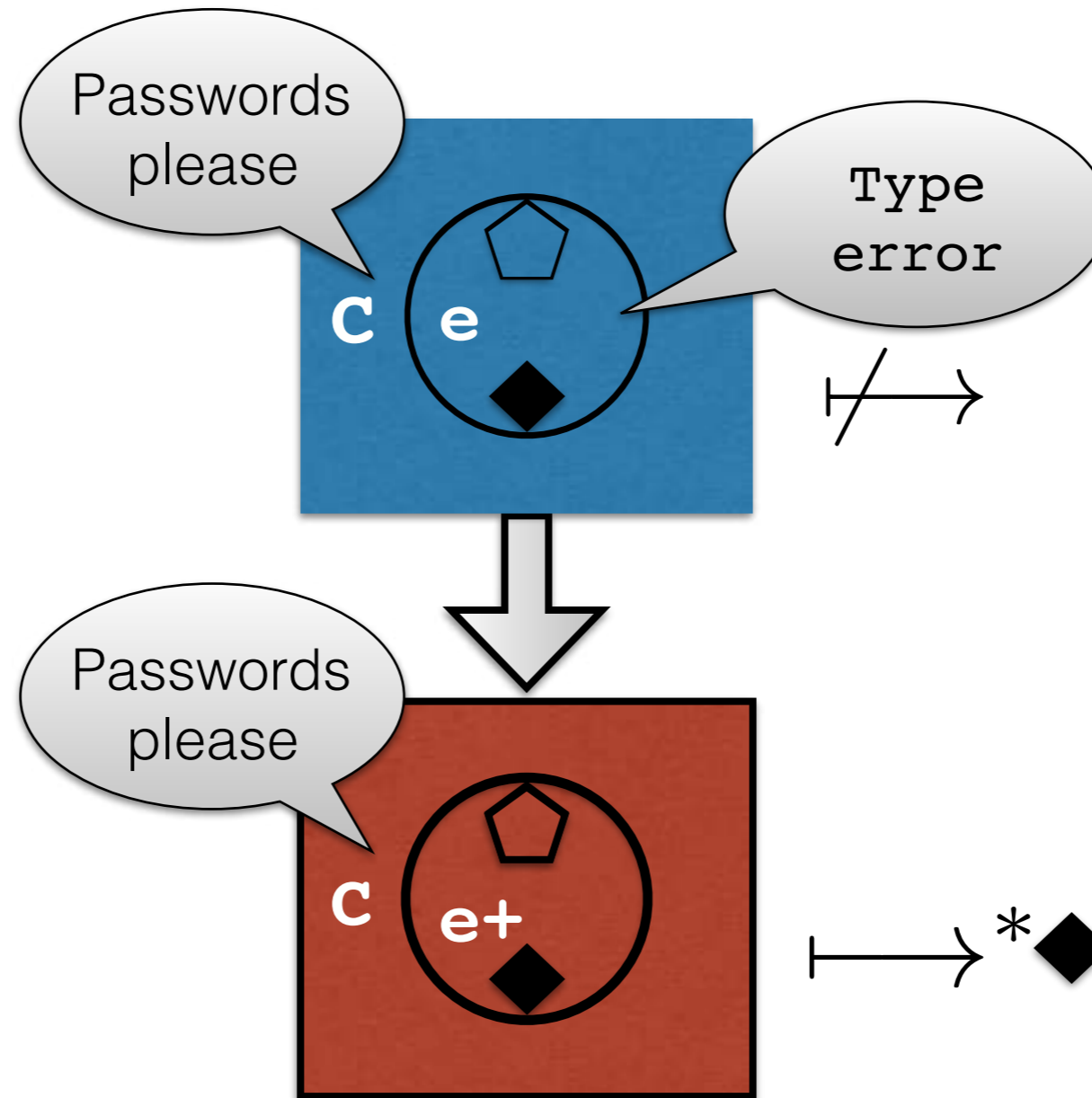
low-level outputs are indistinguishable

WRONG

# Because compilers

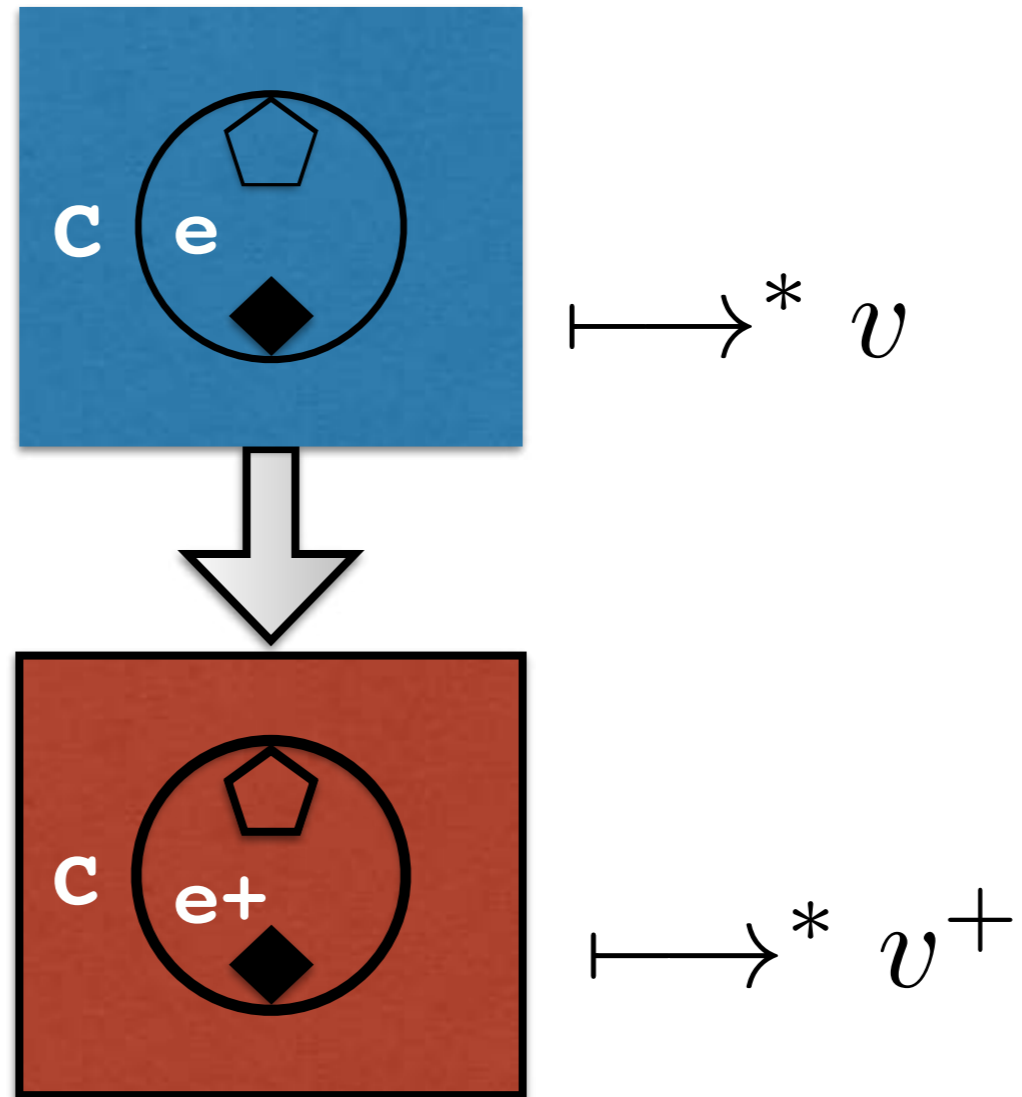


# Because compilers



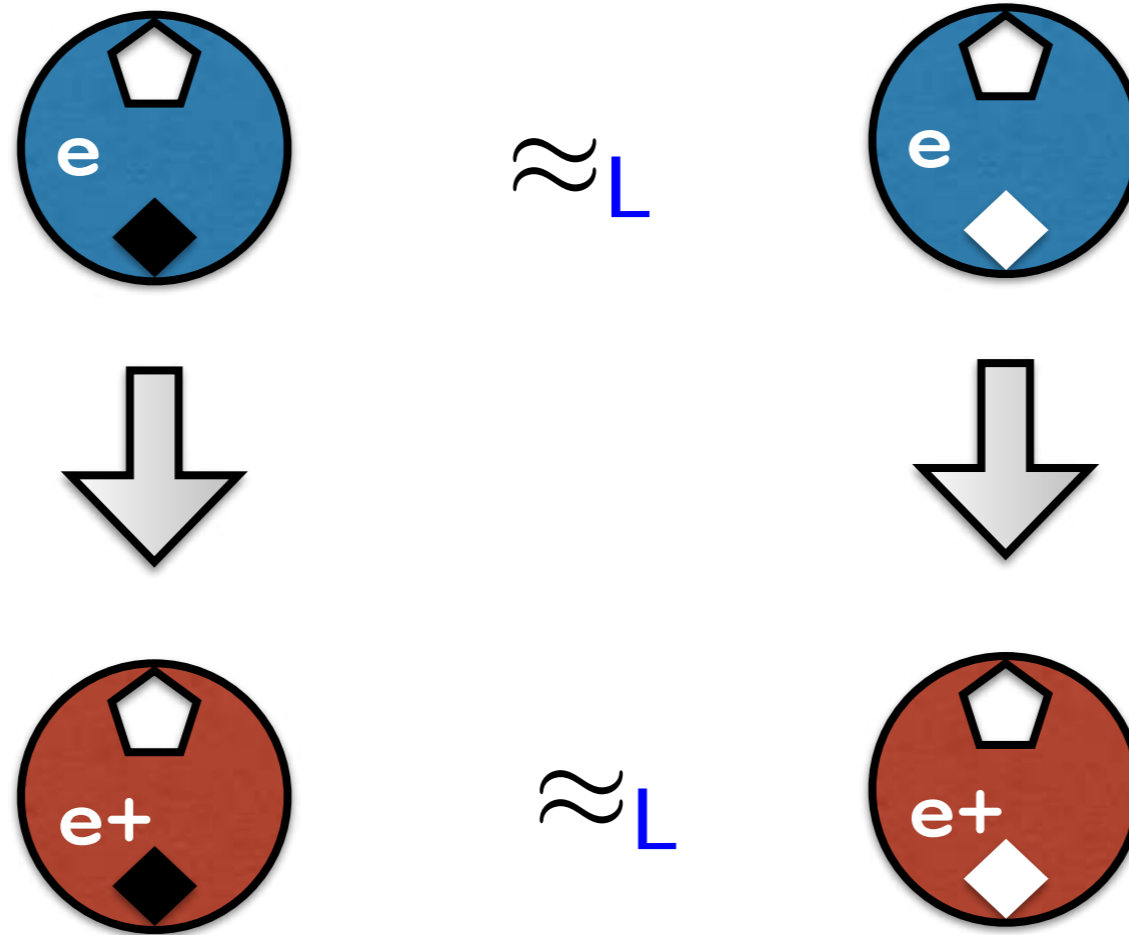
# “Correct”

Even if the compiler is proven “correct” ...



# Equivalence Preserving

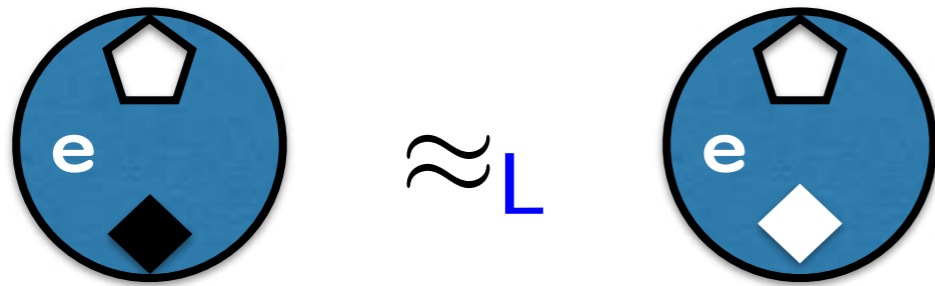
It may not preserve equivalences, e.g., noninterference.



How do we preserve  
noninterference?

# How do we preserve noninterference?

Folklore suggests noninterference can be



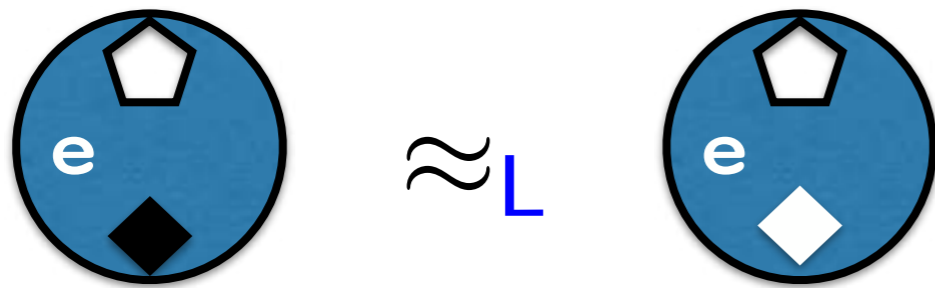
captured by parametricity





# How do we preserve noninterference?

Folklore suggests noninterference can be



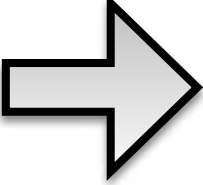
captured by parametricity



[1] Tse & Zdancewic, *Translating Dependency into Parametricity*, ICFP 2004

[2] Shikuma & Igarashi, *Proving Noninterference by a Fully Complete ...*, ASIAN 2006

# Languages

Source: DCC <sup>[1]</sup>  Target: System F $\omega$

- Captures dependency analyses
  - e.g. Information-flow security
- STLC + Lattice of monads
- Parametricity
- Type constructors, i.e., higher-order polymorphism

[1] Abadi *et al.*, *The Core Calculus of Dependency*, POPL 1999

# Source Language

DCC (Core Calculus of Dependency)

Monad protects  
data based on label


$\Gamma \vdash e_1 : T_\ell s_1$

# Source Language

DCC (Core Calculus of Dependency)

$$\eta_H \text{ true} \approx_L \eta_H \text{ false} : T_H \text{ bool}$$

Monad protects  
data based on label

$$\Gamma \vdash e_1 : T_\ell s_1$$


# Source Language

DCC (Core Calculus of Dependency)

$$\frac{\Gamma \vdash e_1 : T_\ell s_1 \quad \Gamma, x : s_1 \vdash e_2 : s_2}{\Gamma \vdash \text{bind } x = e_1 \text{ in } e_2 : s_2}$$

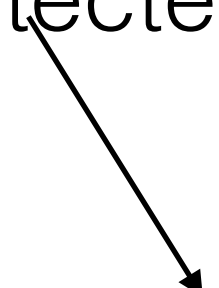
Term containing  
private data

Continuation using private data

# Source Language

DCC (Core Calculus of Dependency)

Promise that result is protected

$$\frac{\Gamma \vdash e_1 : T_\ell s_1 \quad \Gamma, x : s_1 \vdash e_2 : s_2 \quad \ell \preceq s_2}{\Gamma \vdash \text{bind } x = e_1 \text{ in } e_2 : s_2}$$


# Source Language

DCC (Core Calculus of Dependency)

Promise that result is protected

For example...

$L \not\leq \text{bool}$

$H \leq 1$

$\ell \leq s_2$

A diagram consisting of a rounded rectangular box with a light gray gradient and a black border. Inside the box, the text "For example..." is centered at the top. Below it, two expressions are written in blue: "L not less-or-equal to bool" on the left and "H less-or-equal to 1" on the right. To the right of the box, the expression "l less-or-equal to s2" is written in blue. A black arrow points from the top-right corner of the box to the "l" in the expression "l less-or-equal to s2". A horizontal black line is positioned below the expression "l less-or-equal to s2".

# Source Language

DCC (Core Calculus of Dependency)

Promise that result is protected

For example...

$L \not\leq \text{bool}$

$L \leq T_L s$

$H \leq 1$

$H \not\leq T_L s$

$\ell \leq s_2$





# Source Language

DCC (Core Calculus of Dependency)

Promise that result is protected

For example...

$L \not\leq \text{bool}$

$L \leq T_L s$

$L \leq T_H s$

$H \leq 1$

$H \not\leq T_L s$

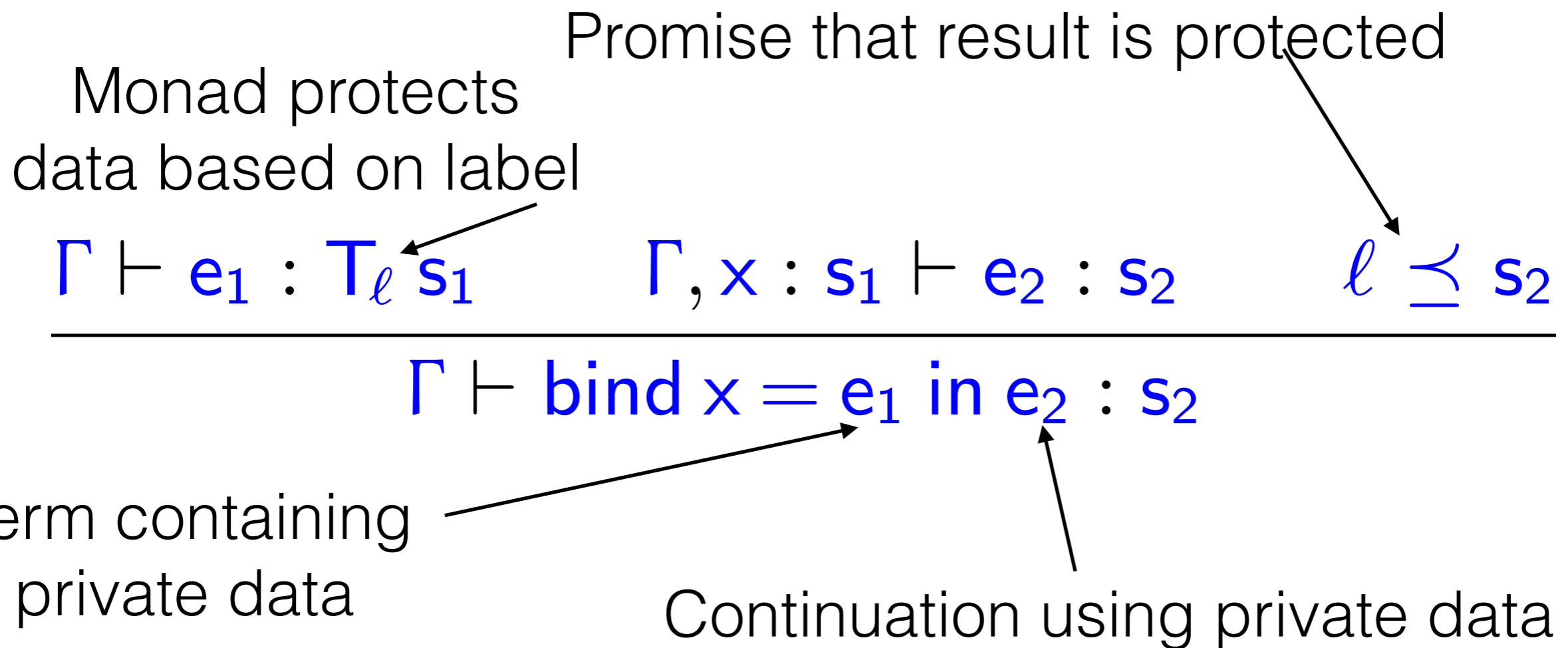
$H \leq T_H s$

$\ell \leq s_2$



# Source Language

DCC (Core Calculus of Dependency)



# Translation, Briefly

How are types translated?

$$1^+ = \mathbf{1}$$

$$\text{bool}^+ = \mathbf{bool}$$

$$(s_1 \rightarrow s_2)^+ = s_1^+ \rightarrow s_2^+$$

$$(T_\ell s)^+ = ?$$

# Translation, Briefly

$$\frac{\Gamma \vdash e_1 : T_l s_1 \quad \Gamma, x : s_1 \vdash e_2 : s_2 \quad l \preceq s_2}{\Gamma \vdash \text{bind } x = e_1 \text{ in } e_2 : s_2}$$

Idea: CPS the monad + constrain continuation result

$$(T_l s)^+ = \forall \beta :: *. (s^+ \rightarrow \beta) \rightarrow \beta$$

s.t.  $l \preceq \beta$

# Translation, Briefly

$$\frac{\Gamma \vdash e_1 : T_l s_1 \quad \Gamma, x : s_1 \vdash e_2 : s_2 \quad l \preceq s_2}{\Gamma \vdash \text{bind } x = e_1 \text{ in } e_2 : s_2}$$

$$(T_l s)^+ = \forall \beta :: *. ([l \preceq \beta] \times (s^+ \rightarrow \beta)) \rightarrow \beta$$

# Translation, Briefly

$$\frac{\Gamma \vdash e_1 : T_l s_1 \quad \Gamma, x : s_1 \vdash e_2 : s_2 \quad l \preceq s_2}{\Gamma \vdash \text{bind } x = e_1 \text{ in } e_2 : s_2}$$

$$(T_l s)^+ = \forall \beta :: *. ([l \preceq \beta] \times (s^+ \rightarrow \beta)) \rightarrow \beta$$

$$[[l \preceq s^+]] = (\alpha_{\preceq} \alpha_l s^+)$$

# Translation, Briefly

$$\frac{\Gamma \vdash e_1 : T_l s_1 \quad \Gamma, x : s_1 \vdash e_2 : s_2 \quad l \preceq s_2}{\Gamma \vdash \text{bind } x = e_1 \text{ in } e_2 : s_2}$$

$$(T_l s)^+ = \forall \beta :: *. ((\alpha_{\preceq} \alpha_l \beta) \times (s \rightarrow \beta)) \rightarrow \beta$$

**data**  $(\alpha_{\preceq} \alpha_l s^+)$  **where**

**Unit\_P**  $::: (\alpha_{\preceq} \alpha_l 1)$

...

**Monad\_P**  $::: [l \sqsubseteq l'] \rightarrow (\alpha_{\preceq} \alpha_l (T_{l'} s)^+)$

# Translation Summary

$$1^+ = \mathbf{1}$$

$$\text{bool}^+ = \mathbf{bool}$$

$$(s_1 \rightarrow s_2)^+ = s_1^+ \rightarrow s_2^+$$

$$(\mathbf{T}_\ell s)^+ = \forall \beta :: *. ((\alpha_{\leq} \alpha_\ell \beta) \times (s \rightarrow \beta)) \rightarrow \beta$$



# Translation Summary

$$1^+ = \mathbf{1}$$

$$\text{bool}^+ = \mathbf{bool}$$

$$(s_1 \rightarrow s_2)^+ = s_1^+ \rightarrow s_2^+$$

$$(\mathbf{T}_\ell s)^+ = \forall \beta :: *. ((\alpha_{\leq} \ \alpha_\ell \ \beta) \times (s \rightarrow \beta)) \rightarrow \beta$$

---

Invariants:

$$\vdash s$$

$$\alpha_{\leq}, \alpha_\ell, \dots \vdash s^+$$

# Proving Noninterference Preservation

# Proving Equivalence Preservation

# Equiv. Preservation is Hard

To show

$$\lambda x : s. e \approx \lambda x : s. e'$$

implies

$$\lambda x : s^+ . e^+ \approx \lambda x : s^+ . e'^+$$

# Equiv. Preservation is Hard

To *show*

$$\lambda x : s. e \approx \lambda x : s. e'$$

implies

$$\lambda x : s^+ . e^+ \approx \lambda x : s^+ . e'^+$$

Assume  $\mathbf{m}_1 \approx \mathbf{m}_2 : s^+$

Show  $e^+[\mathbf{m}_1/\mathbf{x}] \approx e'^+[\mathbf{m}_2/\mathbf{x}]$

# Equiv. Preservation is Hard

To *show*

$$\lambda x : s. e \approx \lambda x : s. e'$$

implies

$$\lambda x : s^+ . e^+ \approx \lambda x : s^+ . e'^+$$

Assume  $\mathbf{m}_1 \approx \mathbf{m}_2 : s^+$

Show  $e^+[\mathbf{m}_1/\mathbf{x}] \approx e'^+[\mathbf{m}_2/\mathbf{x}]$

Want to proceed as follows:

- By assumption,  $\lambda x : s. e \approx \lambda x : s. e'$

# Equiv. Preservation is Hard

To show

$$\lambda x : s. e \approx \lambda x : s. e'$$

implies

$$\lambda x : s^+ . e^+ \approx \lambda x : s^+ . e'^+$$

Assume  $\mathbf{m}_1 \approx \mathbf{m}_2 : s^+$

Show  $e^+[\mathbf{m}_1/x] \approx e'^+[\mathbf{m}_2/x]$

Want to proceed as follows:

- By assumption,  $\lambda x : s. e \approx \lambda x : s. e'$
- Hence,  $e[\mathbf{m}_1/x] \approx e'[\mathbf{m}_2/x]$

# Equiv. Preservation is Hard

To show

$$\lambda x : s. e \approx \lambda x : s. e'$$

implies

$$\lambda x : s^+ . e^+ \approx \lambda x : s^+ . e'^+$$

Assume  $\mathbf{m}_1 \approx \mathbf{m}_2 : s^+$

Show  $e^+[\mathbf{m}_1/x] \approx e'^+[\mathbf{m}_2/x]$

Want to proceed as follows:

- By assumption,  $\lambda x : s. e \approx \lambda x : s. e'$
- Hence,  $e[\mathbf{m}_1/x] \approx e'[\mathbf{m}_2/x]$
- By induction



# Equivalence Reflection?

How do we say that since  $\mathbf{m}_1 \approx \mathbf{m}_2 : \mathbf{s}^+$

$\Downarrow$   $\Downarrow$

there must exist  $\mathbf{e}_1 \quad \mathbf{e}_2 : \mathbf{s}$

# Equivalence Reflection?

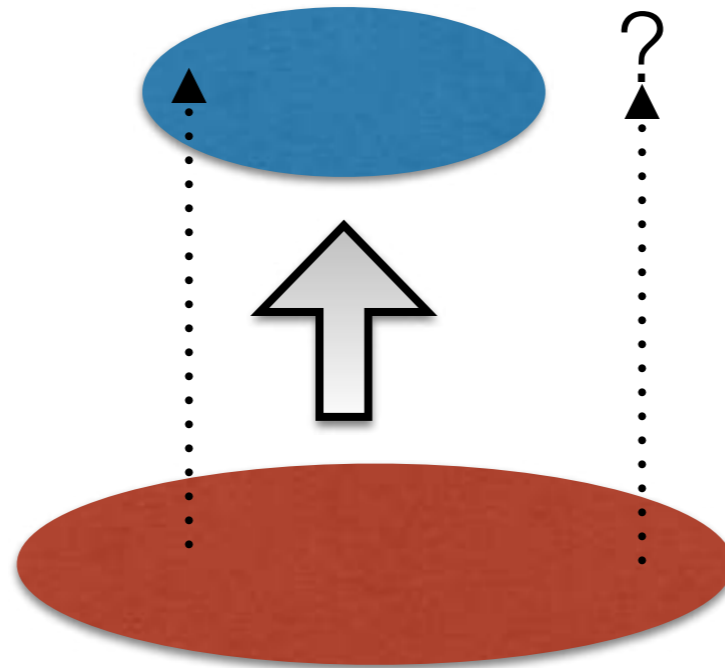
How do we say that since  $\mathbf{m}_1 \approx \mathbf{m}_2 : \mathbf{s}^+$

$\Downarrow \quad \Downarrow$

there must exist  $\mathbf{e}_1 \approx \mathbf{e}_2 : \mathbf{s}$

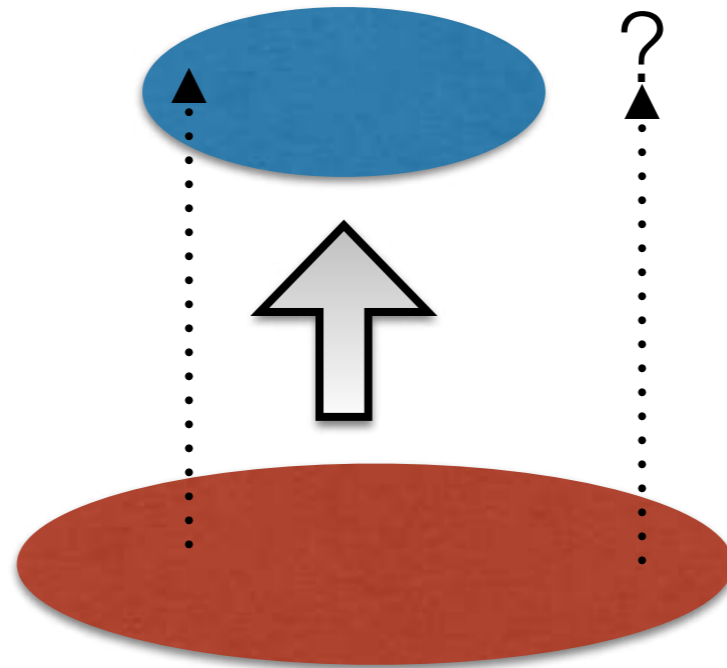
# Back-translation

How can we possibly back-translate  
*a more expressive* target language?



# Enrich Source?

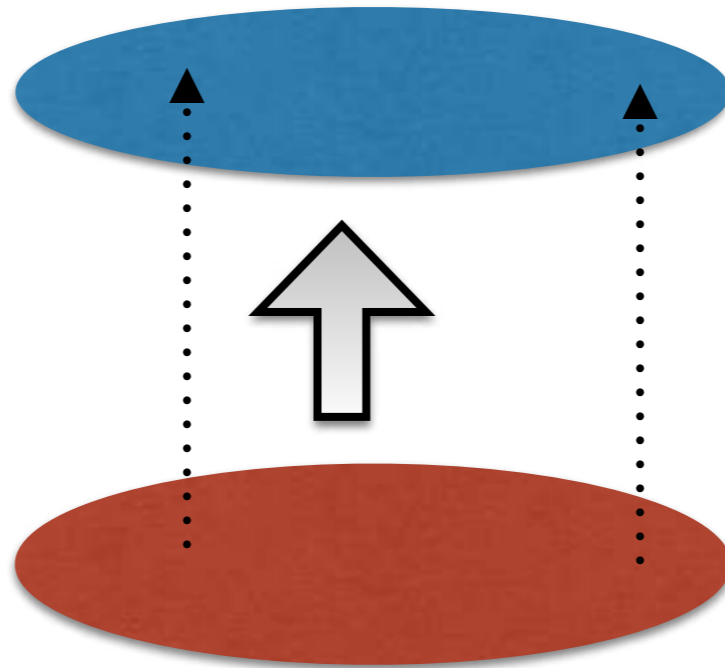
How can we possibly back-translate  
*a more expressive* target language?



We could enrich the source

# Enrich Source?

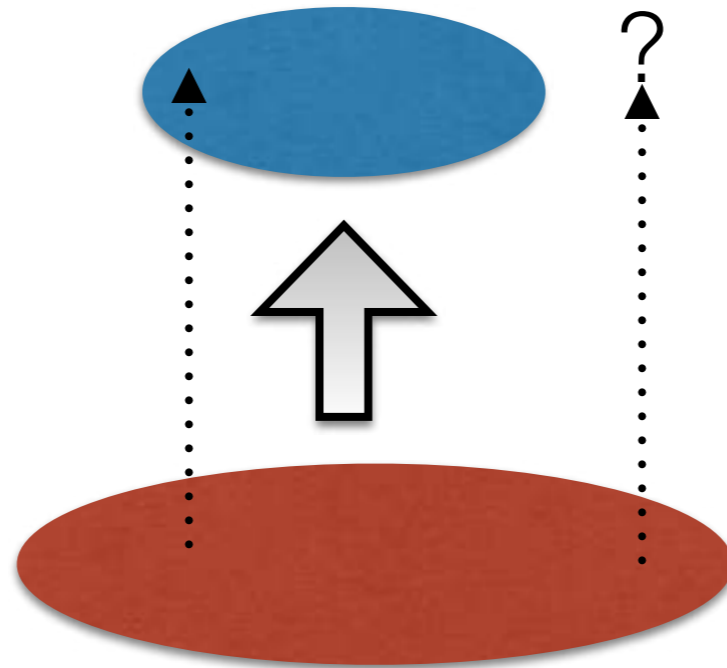
How can we possibly back-translate  
*a more expressive* target language?



We could enrich the source

# Impoverish Target?

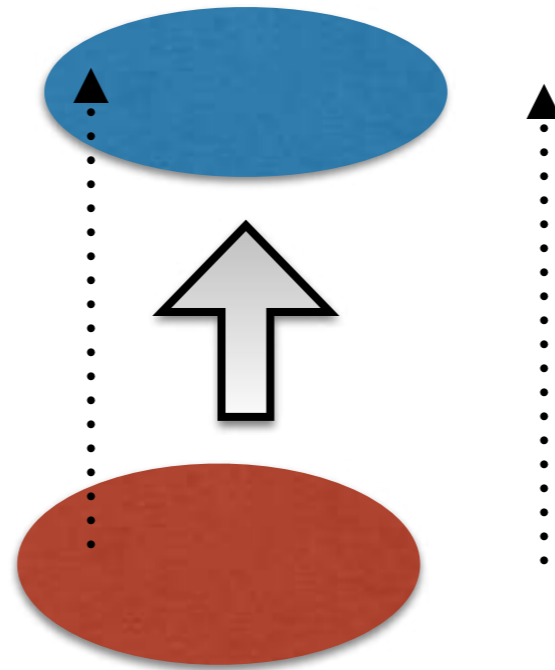
How can we possibly back-translate  
*a more expressive* target language?



Or impoverish the target

# Impoverish Target?

How can we possibly back-translate  
*a more expressive* target language?

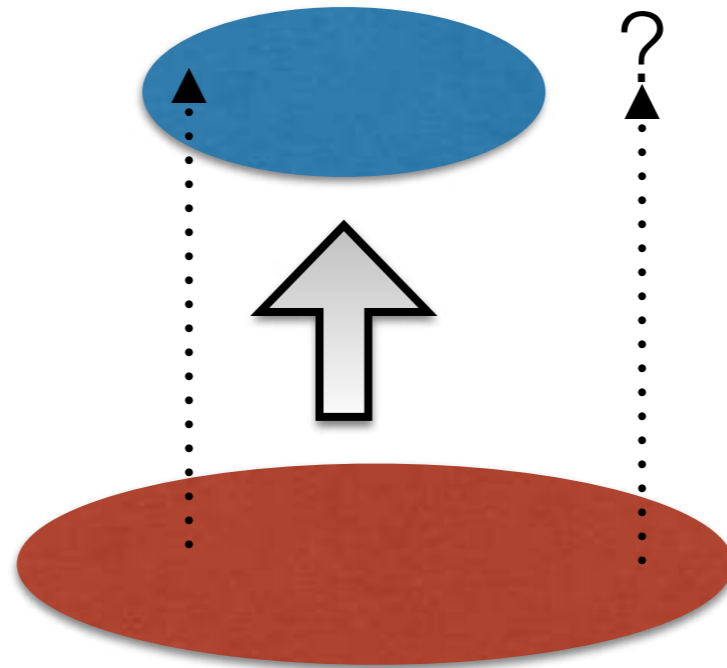


Or impoverish the target

# None of the above

How can we possibly back-translate  
*a more expressive* target language?

Neither is satisfying



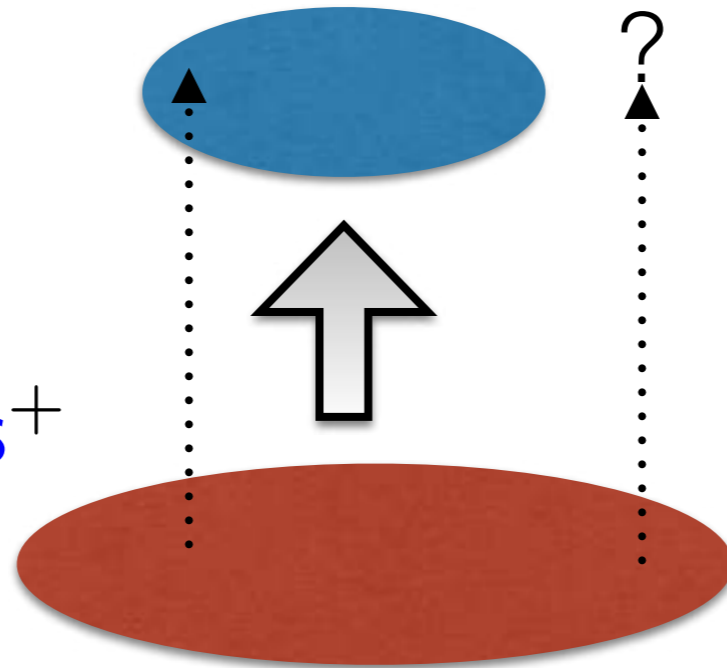


# Be clever

How can we possibly back-translate  
*a more expressive* target language?

Recall:

Assume  $\mathbf{m}_1 \approx \mathbf{m}_2 : \mathbf{s}^+$

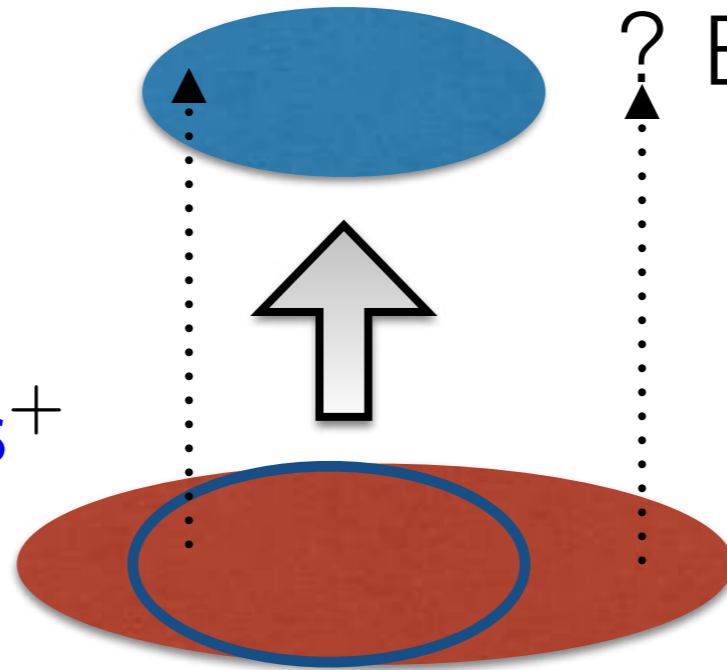


# Be clever

How can we possibly back-translate  
*a more expressive* target language?

Recall:

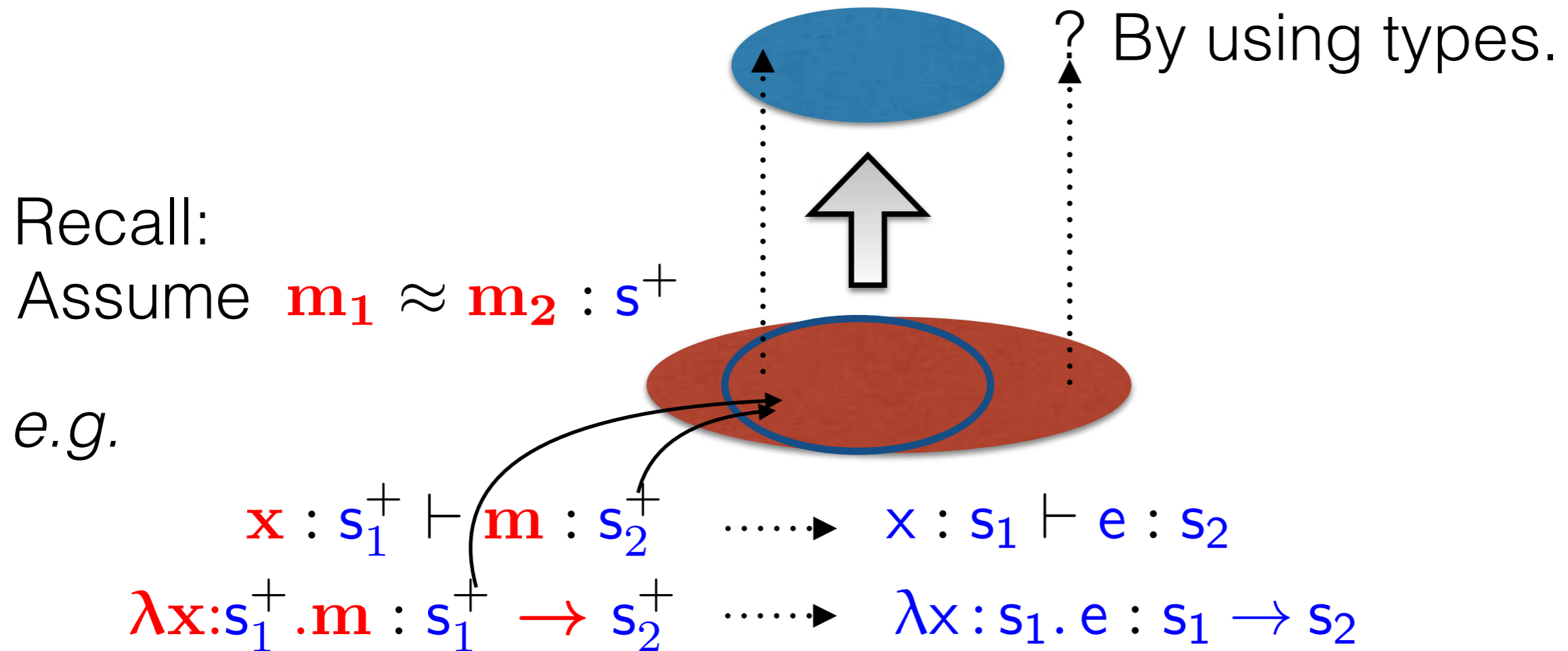
Assume  $\mathbf{m}_1 \approx \mathbf{m}_2 : s^+$



? By using types.

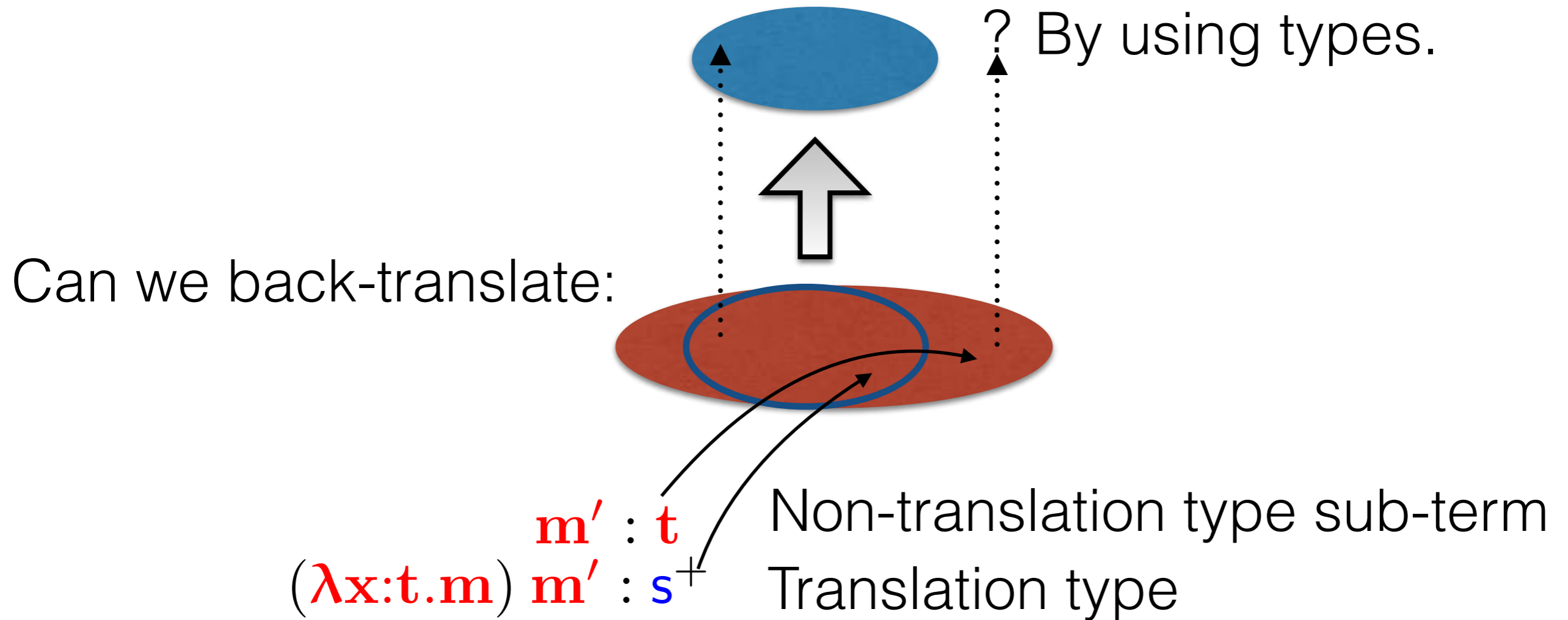
# Be clever

How can we possibly back-translate  
*a more expressive* target language?



# Be clever

How can we possibly back-translate  
*a more expressive* target language?

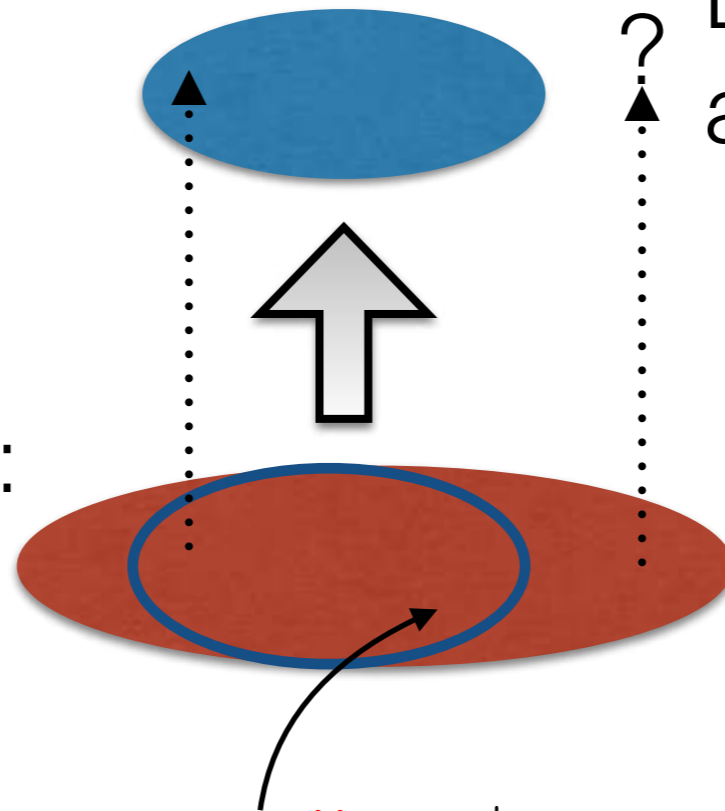


# Be cleverer

How can we possibly back-translate  
*a more expressive* target language?

By using types  
and partial evaluation.

Can we back-translate:  
Not by induction



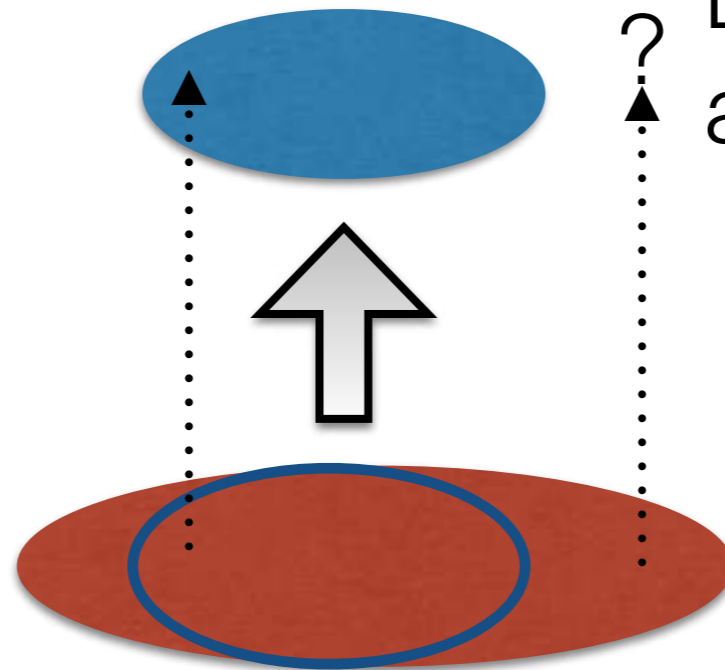
$$(\lambda x:t.m) m' \mapsto m'' : s^+$$

$m' : t$  is no more

# Be clevererer

How can we possibly back-translate  
*a more expressive* target language?

By using types  
and partial evaluation.



Not by induction

Therefore, must prove all terms are back-translatable.

# Proof of Equiv. Preservation

To *show*

$$\lambda x : s. e \approx \lambda x : s. e'$$

implies

$$\lambda x : s^+ . e^+ \approx \lambda x : s^+ . e'^+$$

Assume  $\mathbf{m}_1 \approx \mathbf{m}_2 : s^+$

Show  $e^+[\mathbf{m}_1/\mathbf{x}] \approx e'^+[\mathbf{m}_2/\mathbf{x}]$

How the proof proceeds:

# Proof of Equiv. Preservation

To *show*

$$\lambda x : s. e \approx \lambda x : s. e'$$

implies

$$\lambda x : s^+ . e^+ \approx \lambda x : s^+ . e'^+$$

Assume  $\mathbf{m}_1 \approx \mathbf{m}_2 : s^+$

Show  $e^+[\mathbf{m}_1/\mathbf{x}] \approx e'^+[\mathbf{m}_2/\mathbf{x}]$

How the proof proceeds:

- Back-translate  $\mathbf{m}_1 \approx \mathbf{m}_2 : s^+$  to  $e_1 \approx e_2 : s$



# Proof of Equiv. Preservation

To *show*

$$\lambda x : s. e \approx \lambda x : s. e'$$

implies

$$\lambda x : s^+ . e^+ \approx \lambda x : s^+ . e'^+$$

Assume  $\mathbf{m}_1 \approx \mathbf{m}_2 : \mathbf{s}^+$

Show  $e^+[\mathbf{m}_1/\mathbf{x}] \approx e'^+[\mathbf{m}_2/\mathbf{x}]$

How the proof proceeds:

- Back-translate  $\mathbf{m}_1 \approx \mathbf{m}_2 : \mathbf{s}^+$  to  $e_1 \approx e_2 : \mathbf{s}$
- By assumption,  $\lambda x : s. e \approx \lambda x : s. e'$

# Proof of Equiv. Preservation

To *show*

$$\lambda x : s. e \approx \lambda x : s. e'$$

implies

$$\lambda x : s^+ . e^+ \approx \lambda x : s^+ . e'^+$$

Assume  $\mathbf{m}_1 \approx \mathbf{m}_2 : s^+$

Show  $e^+[\mathbf{m}_1/\mathbf{x}] \approx e'^+[\mathbf{m}_2/\mathbf{x}]$

How the proof proceeds:

- Back-translate  $\mathbf{m}_1 \approx \mathbf{m}_2 : s^+$  to  $e_1 \approx e_2 : s$
- By assumption,  $\lambda x : s. e \approx \lambda x : s. e'$
- Hence,  $e[e_1/\mathbf{x}] \approx e'[e_2/\mathbf{x}]$

# Proof of Equiv. Preservation

To show

$$\lambda x : s. e \approx \lambda x : s. e'$$

implies

$$\lambda x : s^+ . e^+ \approx \lambda x : s^+ . e'^+$$

Assume  $\mathbf{m}_1 \approx \mathbf{m}_2 : s^+$

Show  $e^+[\mathbf{m}_1/x] \approx e'^+[\mathbf{m}_2/x]$

How the proof proceeds:

- Back-translate  $\mathbf{m}_1 \approx \mathbf{m}_2 : s^+$  to  $e_1 \approx e_2 : s$
- By assumption,  $\lambda x : s. e \approx \lambda x : s. e'$
- Hence,  $e[e_1/x] \approx e'[e_2/x]$
- By induction. QED.

# Proof of Equiv. Preservation

To show

$$\lambda x:s. e \approx \lambda x:s. e'$$

implies

$$\lambda x:s^+. e^+ \approx \lambda x:s^+. e'^+$$

Assume  $m_1 \approx m_2 : s^+$

Show  $e^+[m_1/x] \approx e'^+[m_2/x]$

EXCEPT

proceeds:

$$m_1 \approx m_2 : s^+$$

$$\lambda x:s. e \approx \lambda x:s. e'$$

$$e^+[e_2/x]$$

# Proof of Equiv. Preservation

To show

$$\lambda x : s. e \approx \lambda x : s. e'$$

implies

$$\lambda x : s^+. e^+ \approx \lambda x : s^+. e'^+$$

Assume

$$m_1 \approx m_2 : s^+$$

Show

$$e^+[m_1/x] \approx e'^+[m_2/x]$$

does:

$$m_1 \approx m_2 : s^+$$

$$\lambda x : s. e \approx \lambda x : s. e'$$

$$e^+[e_2/x]$$

EXCEPT

If you don't understand logical relations, you can stop listening for the next 4 slides.

# “Open” Logical Relations

Typically, logical relations are defined on closed terms/types.

Again recall: Assume  $\mathbf{m}_1 \approx \mathbf{m}_2 : s^+$

And:  $\alpha_{\leq}, \alpha_{\ell}, \dots \vdash s^+$

But translation types are only well-formed when open.

# “Open” Logical Relations

Again recall: Assume  $\mathbf{m}_1 \approx \mathbf{m}_2 : s^+$

And:  $\alpha_{\underline{s}}, \alpha_l, \dots \vdash s^+$

*i.e.*, to even *state* this assumption, the logical relation *must* leave these type variables open.

# “Open” Logical Relations

$$\cancel{m_1 \approx m_2 : s^+}$$

building on [1], we define  $m_1 \approx^\Sigma m_2 : s^+$

$$\Sigma = \begin{array}{l} \text{data } (\alpha_{\leq} \ \alpha_{\ell} \ s^+) \text{ where} \\ \text{Unit\_P} \ :: (\alpha_{\leq} \ \alpha_{\ell} \ 1) \\ \dots \\ \text{Monad\_P} \ :: (\alpha_{\leq} \ \alpha_{\ell} \ (T_{\ell} s)^+) \end{array}$$



# QED! (ish)

$$\cancel{m_1 \approx m_2 : s^+}$$

$$m_1 \approx^\Sigma m_2 : s^+$$

$\Sigma =$  **data**  $(\alpha_{\leq} \ \alpha_{\ell} \ s^+)$  **where**  
**Unit\_P**  $:: (\alpha_{\leq} \ \alpha_{\ell} \ 1)$   
...  
**Monad\_P**  $:: (\alpha_{\leq} \ \alpha_{\ell} \ (T_{\ell} s)^+)$

$$\llbracket \Sigma \rrbracket_L = ?$$

# Conclusion

- Language-based reasoning requires better compilers
- We have developed techniques for such compilers (specifically, for noninterference preservation)

<https://www.williamjbowman.com/papers#niforfree>