Typed Closure Conversion for the Calculus of Constructions

William J. Bowman, Amal Ahmed

Typed Closure Conversion
for the

Calculus of Constructions

Core calculus on which Coq is built

Dependent types

And high-assurance software

Dependent types

High-assurance software using dependent types

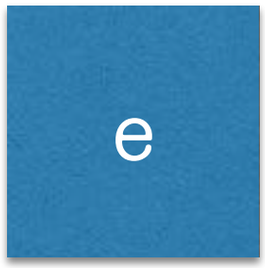
Verified in Coq!

- CompCert
- CertiKOS
- Vellvm
- RustBelt
- CertiCrypt

...

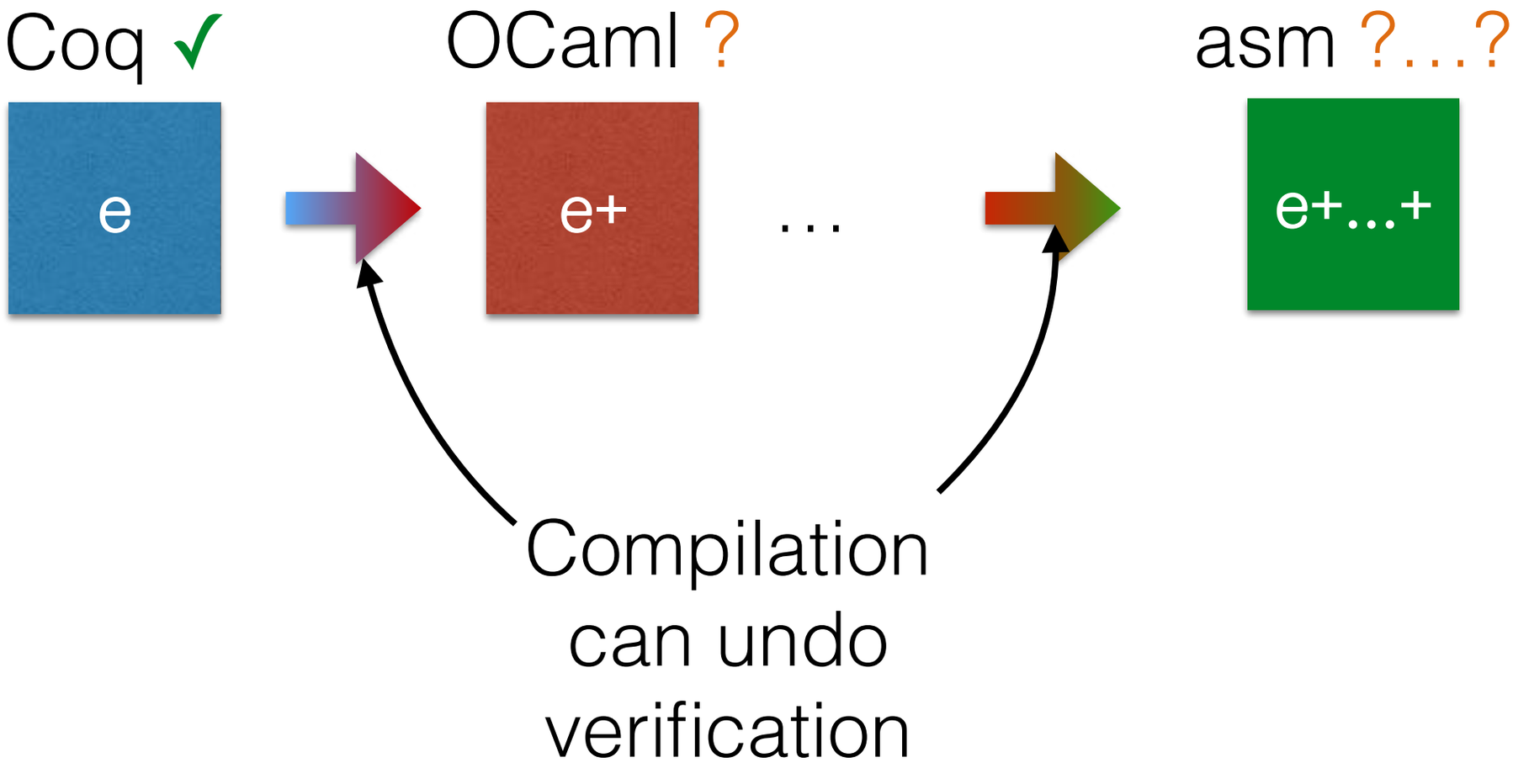
Story of a verified program

Coq



Story of a verified program

Story of a verified program



Compiler correctness!

A correct compilation story

Verify that the program we run is the program we verified

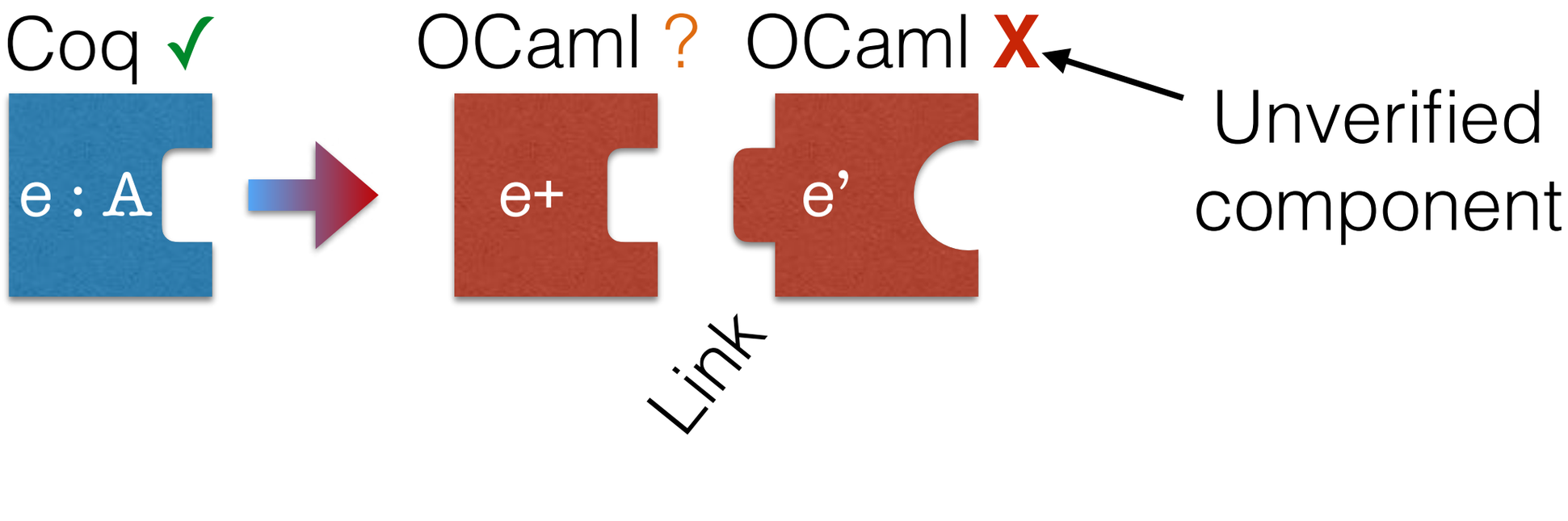
Compiler correctness is not the whole story

Correctness is the “whole program” story

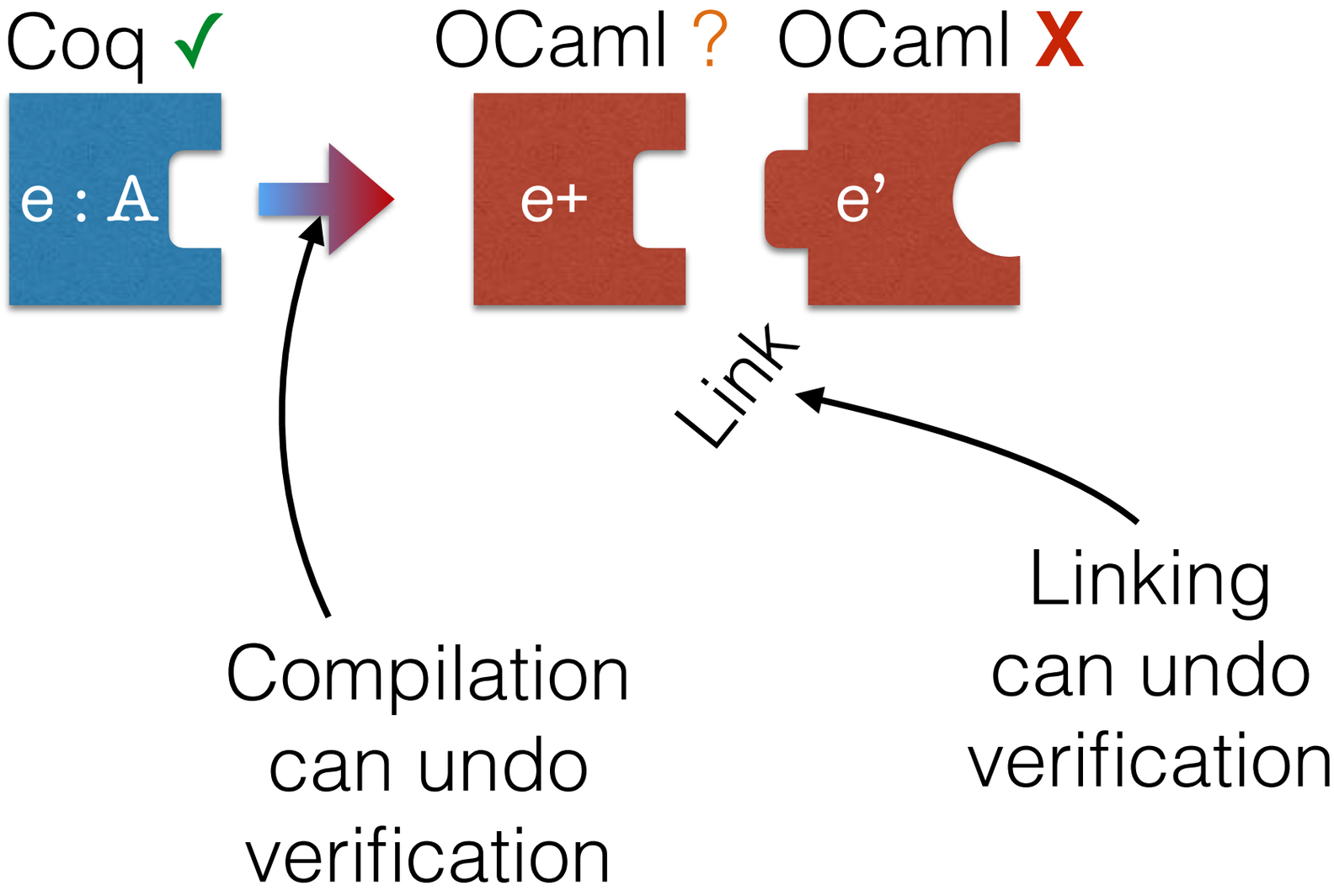
No one* writes whole programs

* Okay, well, not most people.

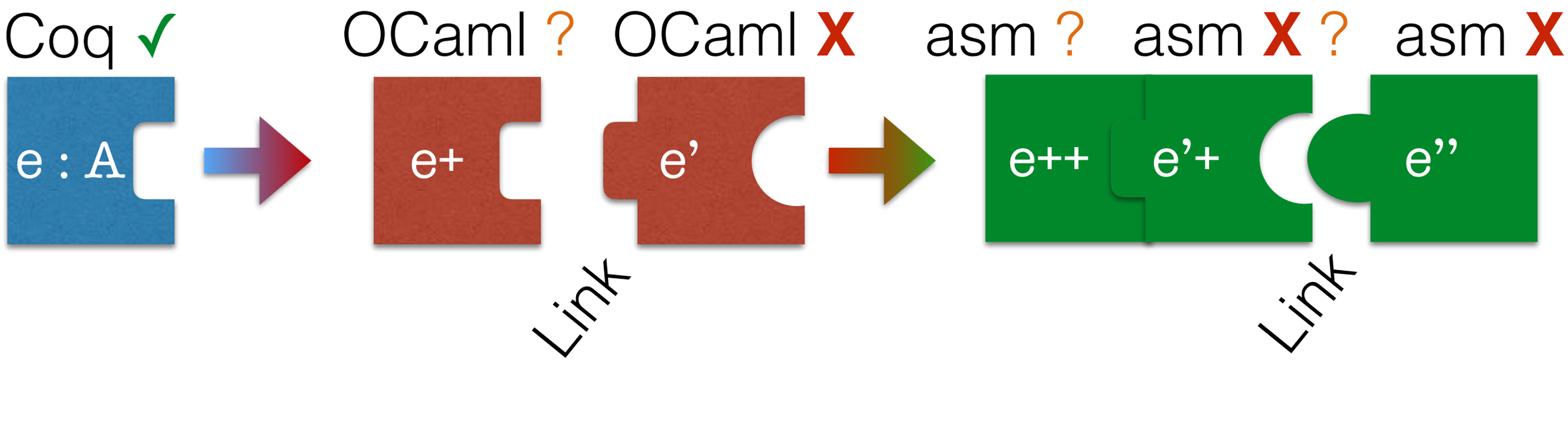
Story of a verified *component*



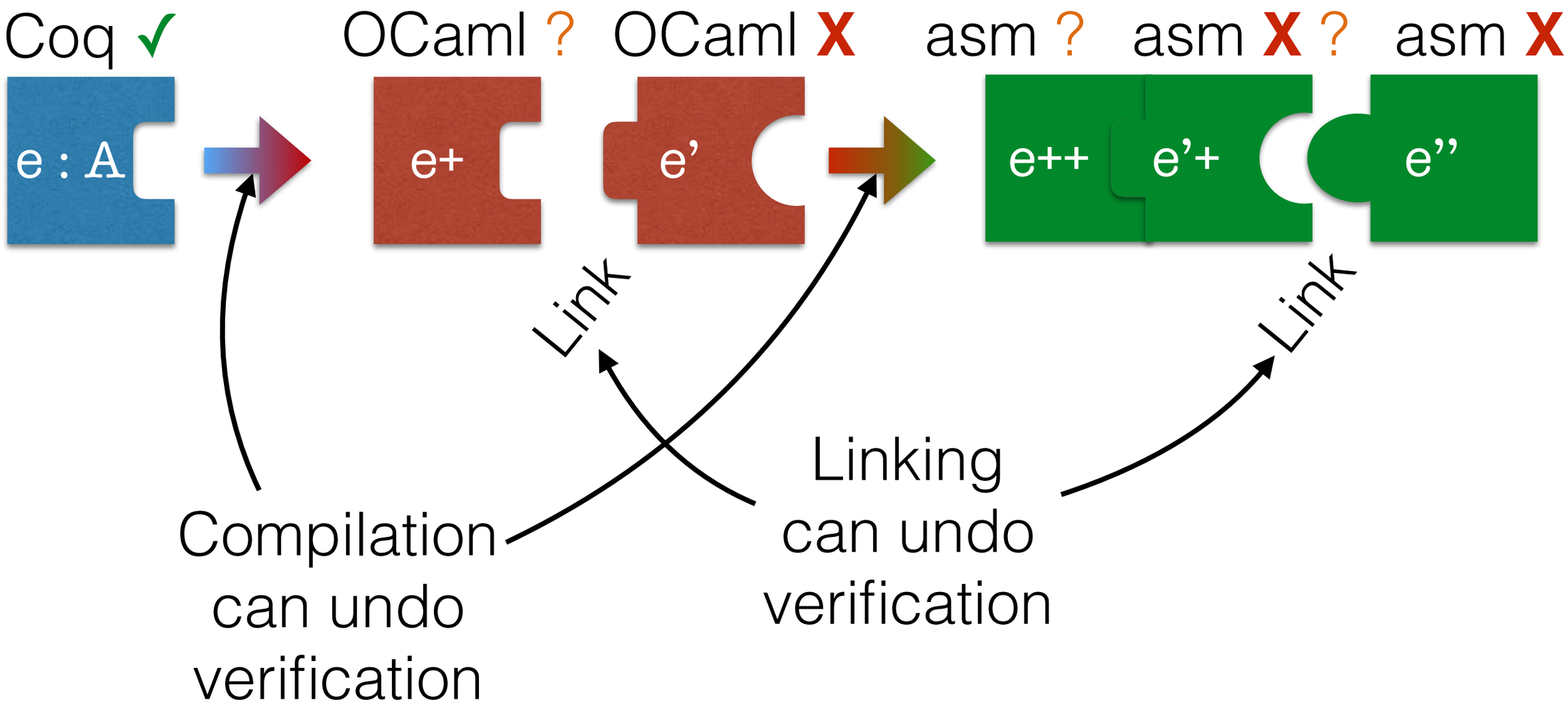
Story of a verified *component*



Story of a verified *component*



Story of a verified *component*



```
> coqc verified.v

> link verified.ml unverified.ml

> ocaml verified.ml
[1]    43185 segmentation fault (core dumped)
ocaml verified.ml
```

Be careful?

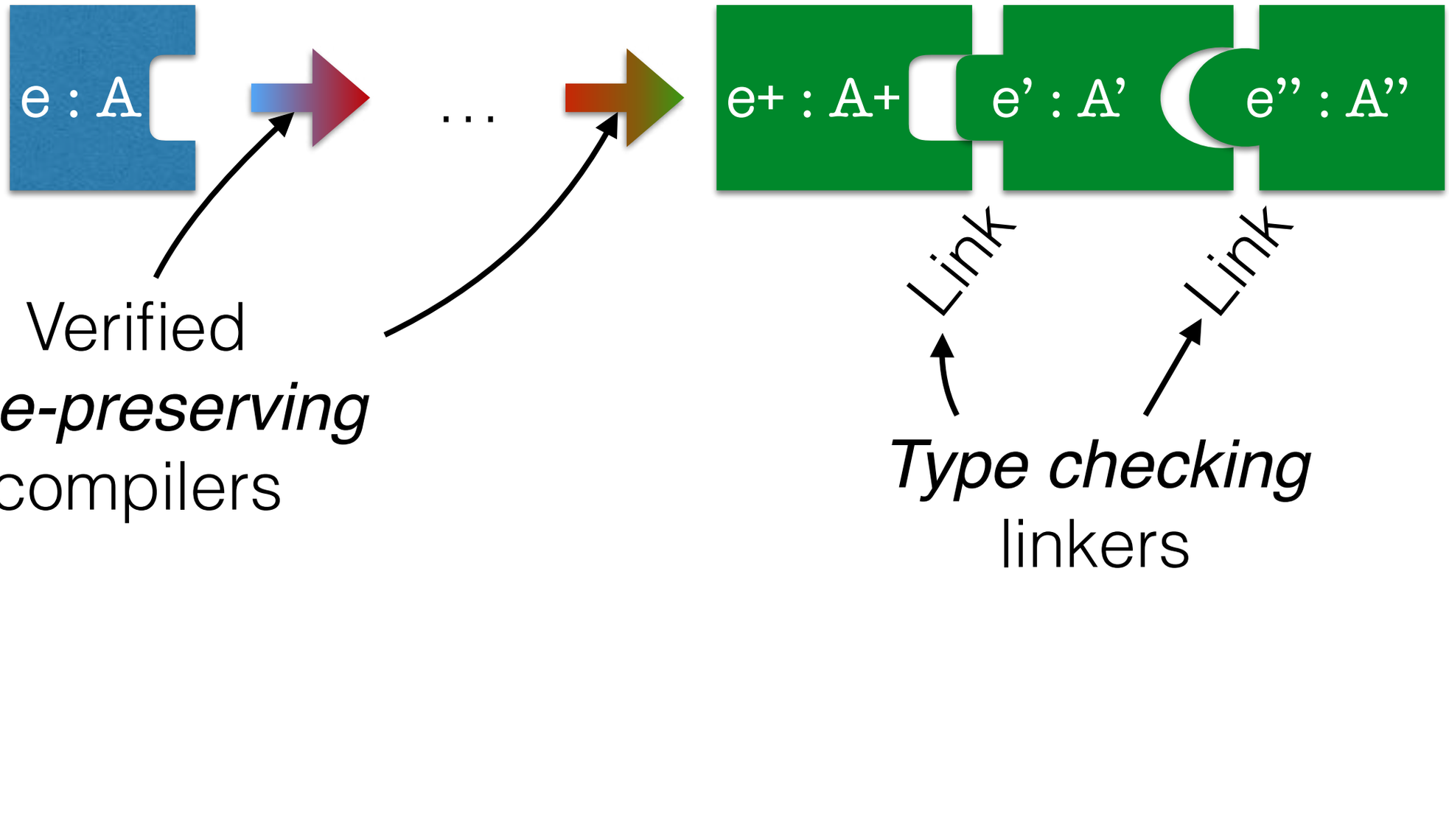
```
> coqc verified.v

> link verified.ml unverified.ml

> ocaml verified.ml
[1]    43185 segmentation fault (core dumped)
ocaml verified.ml
```

No!
Be well-typed!

Be careful?

Coq

Dep. Type ASM



Verified
type-preserving
compilers

Type checking
linkers

Typed Closure Conversion for the Calculus of Constructions

A standard compiler pass for functional languages

A type-preserving compiler

Continuation-Passing Style (CPS)

Closure Conversion

Allocation

Code generation

Morrisett, Walker, Crary, Glew 1998

A type-preserving compiler

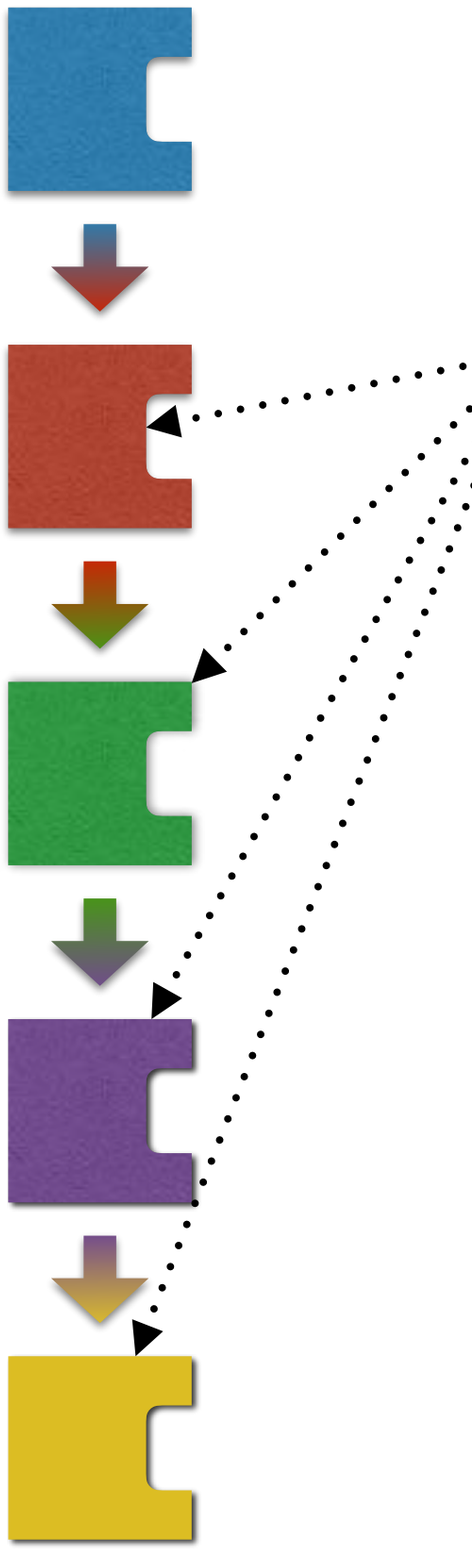
Theorem. (Type Preservation)

If $e : A$

then $e^+ : A^+$ translates to

⋮

A type-preserving compiler

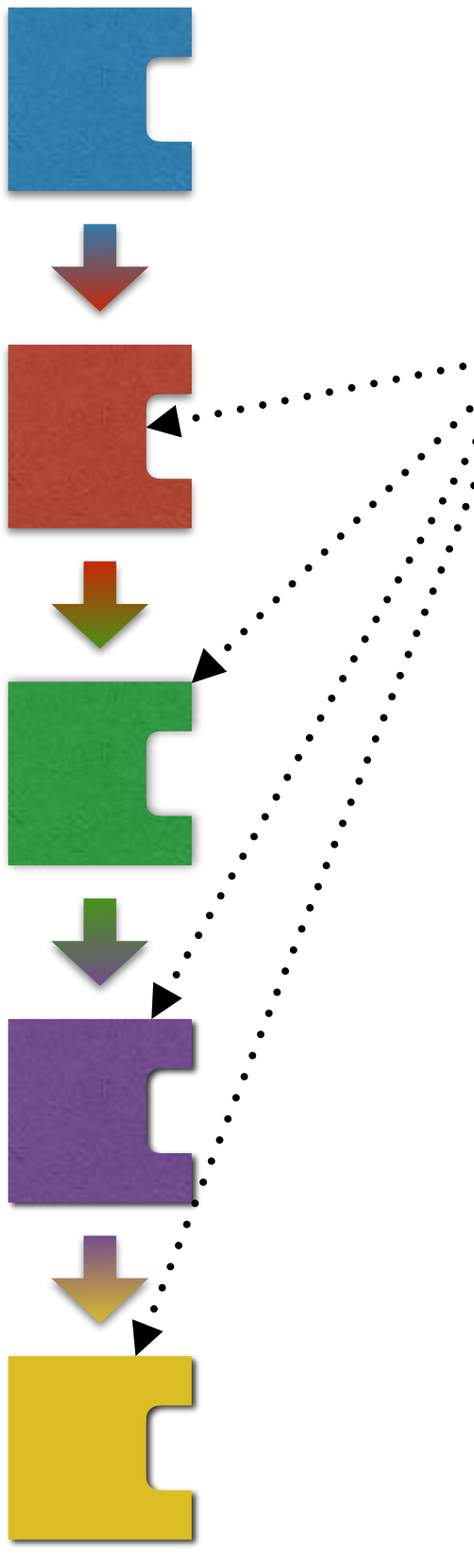


Design typed intermediate language

Prove soundness, decidability, etc

A dependent-type-preserving compiler

- Move from ***functional, compositional*** to ***global, stateful, instruction based***
- ***Which axioms*** can we use (e.g parametricity, impredicativity)



Design typed intermediate language

Prove soundness, decidability, etc

Brief history of preserving *dependent* types

Continuation-Passing Style (CPS)

1999 2002 ◂…. 2018

Impossibility result

Un-impossibility result

Brief history of preserving *dependent* types

Continuation-Passing Style (CPS)

1999 2002 ◂…. 2018

Impossibility result

Un-impossibility result

Key insights:

- Past work doesn't scale in the obvious way for dependent type theory

Type-Preserving Closure Conversion

Closure Conversion

Key problem:

- Which ***axioms*** does past work rely on, and can we use them?

Type-Preserving Closure Conversion

The standard (non-dependent) type-preserving translation

Goal: Translate $\Gamma \vdash f : A \to B$

Type-Preserving Closure Conversion

The standard (non-dependent) type-preserving translation

Goal: Translate $\Gamma \vdash f : A \to B$

Takes 1 argument, of type A

Type-Preserving Closure Conversion

The standard (non-dependent) type-preserving translation

Goal: Translate $\Gamma \vdash f : A \to B$

Takes 1 argument, of type A
returns result of type B

Type-Preserving Closure Conversion

The standard (non-dependent) type-preserving translation

Goal: Translate $\Gamma \vdash f : A \to B$

And can refer to lexical variables

Type-Preserving Closure Conversion

The standard (non-dependent) type-preserving translation

Goal: Translate $\Gamma \vdash f : A \to B$

To: $\Gamma^+ \vdash \langle \Gamma^+, f^+ \rangle : \mathit{Closure}(A^+ \to B^+)$

Pair of ***data*** and a ***code pointer***. (object)

Type-Preserving Closure Conversion

The standard (non-dependent) type-preserving translation

Goal: Translate $\Gamma \vdash f : A \to B$

To: $\Gamma^+ \vdash \langle \Gamma^+, f^+ \rangle : \mathit{Closure}(A^+ \to B^+)$

Where: $\cdot \vdash f^+ : \Gamma^+ \to A^+ \to B^+$

Code pointers are ***closed except formal arguments*** (can be heap allocated).

Type-Preserving Closure Conversion

The standard (non-dependent) type-preserving translation

How do we implement a ***typed closure***?

$$\Gamma^+ \vdash \langle \Gamma^+, f^+ \rangle : Closure(A^+ \to B^+)$$

Type-Preserving Closure Conversion

The standard (non-dependent) type-preserving translation

How do we implement a ***typed closure***?

$$\Gamma^+ \vdash \langle \Gamma^+, f^+ \rangle : Closure(A^+ \to B^+)$$

1. Not as pairs:

$$\Gamma^+ \vdash \langle \Gamma^+, f^+ \rangle : Pair(\Gamma^+, \Gamma^+ \to A^+ \to B^+)$$

$$\mathbf{apply\ c\ x} \stackrel{\text{def}}{=} \mathbf{(snd\ c)\ (fst\ c)\ x}$$

How do we implement a *typed closure*?

1. Not as pairs:

$$\Gamma^+ \vdash \langle \Gamma^+, f^+ \rangle : Pair(\Gamma^+, \Gamma^+ \to A^+ \to B^+)$$

$$\mathbf{apply\ c\ x} \stackrel{\text{def}}{=} (\mathbf{snd\ c})\,(\mathbf{fst\ c})\,\mathbf{x}$$

✓ Equal

$$z : Nat \vdash f : Nat \to Nat$$

$$\vdash f' : Nat \to Nat$$

How do we implement a *typed closure*?

1. Not as pairs:

$$\Gamma^+ \vdash \langle \Gamma^+, f^+ \rangle : Pair(\Gamma^+, \Gamma^+ \to A^+ \to B^+)$$

$$\mathbf{apply\ c\ x} \overset{\text{def}}{=} (\mathbf{snd\ c})\,(\mathbf{fst\ c})\,\mathbf{x}$$

✓ Equal

$$z : Nat \vdash f : Nat \to Nat$$

$$\vdash f' : Nat \to Nat$$

X Not equal

$$\vdash \langle z, f^+ \rangle : Pair((z : \mathbf{Nat}), (z : \mathbf{Nat}) \to \mathbf{Nat} \to \mathbf{Nat})$$

$$\vdash \langle (), f'^+ \rangle : Pair((), () \to \mathbf{Nat} \to \mathbf{Nat})$$

How do we implement a *typed closure*?

1. Not as pairs:

$$\Gamma^+ \vdash \langle \Gamma^+, f^+ \rangle : Pair(\Gamma^+, \Gamma^+ \to A^+ \to B^+)$$

$$\mathbf{apply\ c\ x} \stackrel{\text{def}}{=} \mathbf{(snd\ c)\ (fst\ c)\ x}$$

$$\mathbf{extract\_hidden\_data\ c} \stackrel{\text{def}}{=} \mathbf{fst\ c}$$

X Not Secure

Type-Preserving Closure Conversion

The standard (non-dependent) type-preserving translation

How do we implement a ***typed closure***?

$$\Gamma^+ \vdash \langle \Gamma^+, f^+ \rangle : Closure(A^+ \to B^+)$$

1. Not as pairs:

$$\Gamma^+ \vdash \langle \Gamma^+, f^+ \rangle : Pair(\Gamma^+, \Gamma^+ \to A^+ \to B^+)$$

$$\mathbf{apply\ c\ x} \overset{\text{def}}{=} \mathbf{(snd\ c)\ (fst\ c)\ x}$$

Type-Preserving Closure Conversion

The standard (non-dependent) type-preserving translation

How do we implement a ***typed closure***?

$$\Gamma^+ \vdash \langle \Gamma^+, \mathsf{f}^+ \rangle : \mathit{Closure}(\mathsf{A}^+ \to \mathsf{B}^+)$$

1. Not as pairs:

$$\Gamma^+ \vdash \langle \Gamma^+, \mathsf{f}^+ \rangle : \mathit{Pair}(\Gamma^+, \Gamma^+ \to \mathsf{A}^+ \to \mathsf{B}^+)$$

$$\mathbf{apply\ c\ x} \stackrel{\text{def}}{=} \mathbf{(snd\ c)\ (fst\ c)\ x}$$

2. As *existential* pairs:

$$\Gamma^+ \vdash \langle \Gamma^+, \mathsf{f}^+ \rangle : \exists \alpha.(\alpha, (\alpha \to \mathsf{A}^+ \to \mathsf{B}^+))$$

$$\mathbf{apply\ c\ x} \stackrel{\text{def}}{=} \mathbf{unpack}\langle \alpha, \mathsf{p} \rangle \mathbf{in\ (snd\ c)\ (fst\ c)\ x}$$

How do we implement a *typed closure*?

2. As ***existential*** pairs:

$$\Gamma^+ \vdash \langle \Gamma^+, f^+ \rangle : \exists \alpha.(\alpha, (\alpha \to A^+ \to B^+))$$

$$\mathbf{apply\ c\ x} \overset{\text{def}}{=} \mathbf{unpack}\langle \alpha, \mathbf{p} \rangle \mathbf{in}\ (\mathbf{snd\ c})\ (\mathbf{fst\ c})\ \mathbf{x}$$

✓ Equal

$$z : \mathsf{Nat} \vdash f : \boxed{\mathsf{Nat} \to \mathsf{Nat}}$$

$$\vdash f' : \boxed{\mathsf{Nat} \to \mathsf{Nat}}$$

✓ Equal

$$\vdash \langle (), f'^+ \rangle : \boxed{\exists \alpha.(\alpha, \alpha \to \mathbf{Nat} \to \mathbf{Nat})}$$

$$\vdash \langle z, f^+ \rangle : \boxed{\exists \alpha.(\alpha, \alpha \to \mathbf{Nat} \to \mathbf{Nat})}$$

How do we implement a *typed closure*?

2. As *existential* pairs:

$$\Gamma^+ \vdash \langle \Gamma^+, f^+ \rangle : \exists \alpha.(\alpha, (\alpha \to A^+ \to B^+))$$

$$\mathbf{apply\ c\ x} \overset{\text{def}}{=} \mathbf{unpack}\langle \alpha, \mathbf{p} \rangle \mathbf{in}\ (\mathbf{snd\ c})\ (\mathbf{fst\ c})\ \mathbf{x}$$

$$\mathbf{extract\_hidden\_data\ c} \overset{\text{def}}{=} \mathbf{fst\ c}$$

✗

✓ Secure

Not definable

Type-Preserving Closure Conversion

The standard (non-dependent) type-preserving translation

How do we implement a ***typed closure***?

$$\Gamma^+ \vdash \langle \Gamma^+, f^+ \rangle : Closure(A^+ \to B^+)$$

As ***existential*** pairs:

$$\Gamma^+ \vdash \langle \Gamma^+, f^+ \rangle : \exists \alpha.(\alpha, (\alpha \to A^+ \to B^+))$$

$$\mathbf{apply\ c\ x} \stackrel{\text{def}}{=} \mathbf{unpack}\langle \alpha, p \rangle \mathbf{in}\ (\mathbf{snd\ c})\ (\mathbf{fst\ c})\ \mathbf{x}$$

Dependent-Type-Preserving Closure Conversion

Goal: Translate $\Gamma \vdash f : \Pi\, x : A.\, B$

Dependent-Type-Preserving Closure Conversion

Goal: Translate $\Gamma \vdash f : \Pi\, x : A.\, B$

Takes 1 argument, **x**, of type **A**.

Dependent-Type-Preserving Closure Conversion

Goal: Translate $\Gamma \vdash f : \Pi\, x : A.\, B$

Returns result of type B.

Dependent-Type-Preserving Closure Conversion

Goal: Translate $\Gamma \vdash f : \Pi\, x : A.\, B$

Refer to lexical variables

Dependent-Type-Preserving Closure Conversion

Goal: Translate $\Gamma \vdash f : \Pi\, x : A.\, B$

And so can types:
types can ***depend*** on terms

Dependent-Type-Preserving Closure Conversion

Goal: Translate $\Gamma \vdash f : \Pi\, x : A.\, B$

The term-level 0

$$\mathrm{div} : \Pi\, x : \mathrm{Nat}.\, \Pi\, y : \mathrm{Nat}.\, \Pi\, p : y \neq 0.\, \mathrm{Int}$$

The term-level inequality function

Dependent-Type-Preserving Closure Conversion

Goal: Translate $\Gamma \vdash f : \Pi\, x : A.\, B$

Dependent-Type-Preserving Closure Conversion

Goal: Translate $\Gamma \vdash f : \Pi\, x : A.\, B$

To: $\Gamma^{+} \vdash \langle \Gamma^{+}, f^{+} \rangle : \mathit{Closure}(x : A^{+} \to B^{+})$

Where: $\cdot \vdash f^{+} : \Pi \Gamma^{+}.\, \Pi\, x : A^{+}.\, B^{+}$

Code pointers are ***closed***
(can be heap allocated)

Dependent-Type-Preserving Closure Conversion

How do we implement a ***dependently typed closure***?

$$\Gamma^+ \vdash \langle \Gamma^+, f^+ \rangle : Closure(x : A^+ \to B^+)$$

Hint: not as pairs

Dependent-Type-Preserving Closure Conversion

How do we implement a ***dependently typed closure***?

$$\Gamma^+ \vdash \langle \Gamma^+, f^+ \rangle : Closure(x : A^+ \to B^+)$$

Hint: not as pairs

Hint: existential pairs don't work either

Digression on the nature of existence

On existence

$$\frac{\Gamma, \alpha : \mathbf{Type} \vdash A : \mathbf{Type}}{\Gamma \vdash \exists\, \alpha : \mathbf{Type}.\, A : \mathbf{Type}}$$

Key problem:

- Which ***axioms*** does past work rely on, and can we use them?

On existence

If A is a Type,

$$\frac{\Gamma, \alpha : \mathbf{Type} \vdash A : \mathbf{Type}}{\Gamma \vdash \exists\, \alpha : \mathbf{Type}.\, A : \mathbf{Type}}$$

Then

On existence

If A is a Type,
under the assumption that α is a Type

$$\frac{\Gamma, \alpha : \mathbf{Type} \vdash A : \mathbf{Type}}{\Gamma \vdash \exists\, \alpha : \mathbf{Type}.\, A : \mathbf{Type}}$$

Then

On existence

If A is a Type,
under the assumption that α is a Type

$$\frac{\Gamma, \alpha : \mathbf{Type} \vdash A : \mathbf{Type}}{\Gamma \vdash \exists\, \alpha : \mathbf{Type}.\, A : \mathbf{Type}}$$

Then “there exists an α such that A holds of α” is a Type.

On existence

$$\frac{\Gamma, \alpha : \mathbf{Type} \vdash A : \mathbf{Type}}{\Gamma \vdash \exists\, \alpha : \mathbf{Type}.\, A : \mathbf{Type}}$$

Properties:

On existence

$$\frac{\Gamma, \alpha : \mathbf{Type} \vdash A : \mathbf{Type}}{\Gamma \vdash \exists\, \alpha : \mathbf{Type}.\, A : \mathbf{Type}}$$

Properties:

1. Existence is *impredicative*
(formed by quantifying over Types *including itself*)

On existence

$$\frac{\Gamma, \alpha : \mathbf{Type} \vdash A : \mathbf{Type}}{\Gamma \vdash \exists\, \alpha : \mathbf{Type}.\, A : \mathbf{Type}}$$

Properties:

1. Existence is *impredicative*
   (formed by quantifying over Types *including itself*)
2. Existence is *computationally relevant*
   (terms of this type can be used when running a program)

On existence

$$\frac{\Gamma, \alpha : \mathbf{Type} \vdash A : \mathbf{Type}}{\Gamma \vdash \exists\, \alpha : \mathbf{Type}.\, A : \mathbf{Type}}$$

Properties:

1. Existence is *impredicative*
   (formed by quantifying over Types *including itself*)
2. Existence is *computationally relevant*
   (terms of this type can be used when running a program)
3. Existence is *parametric* in α
   (terms cannot make inspect α at run-time)

On existence

$$\frac{\Gamma, \alpha : \mathbf{Type} \vdash A : \mathbf{Type}}{\Gamma \vdash \exists\, \alpha : \mathbf{Type}.\, A : \mathbf{Type}}$$

Properties:

1. Existence is *impredicative*
   (formed by quantifying over Types *including itself*)
2. Existence is *computationally relevant*
   (terms of this type can be used when running a program)
3. Existence is *parametric* in α
   (terms cannot make inspect α at run-time)
4. Existence can be used in *large elimination*
   (we can compute *types from term* of existential types)

Existence is incompatible with reality

$$\frac{\Gamma, \alpha : \mathbf{Type} \vdash A : \mathbf{Type}}{\Gamma \vdash \exists\, \alpha : \mathbf{Type}.\, A : \mathbf{Type}}$$

We could add this axiom, *in theory*, but *in reality*:

Existence is incompatible with reality

$$\frac{\Gamma, \alpha : \mathbf{Type} \vdash A : \mathbf{Type}}{\Gamma \vdash \exists\, \alpha : \mathbf{Type}.\, A : \mathbf{Type}}$$

We could add this axiom, *in theory*, but *in reality*:

1. Some type theories do not admit *impredicativity*
(Agda, Coq for computational Types (by default))

Existence is incompatible with reality

$$\frac{\Gamma, \alpha : \mathbf{Type} \vdash A : \mathbf{Type}}{\Gamma \vdash \exists\, \alpha : \mathbf{Type}.\, A : \mathbf{Type}}$$

We could add this axiom, *in theory*, but *in reality*:

1. Some type theories do not admit *impredicativity* (Agda, Coq for computational Types (by default))
2. Some type theories do not admit *parametricity* (Agda, Coq by default)

Existence is incompatible with reality

$$\frac{\Gamma, \alpha : \mathbf{Type} \vdash A : \mathbf{Type}}{\Gamma \vdash \exists\, \alpha : \mathbf{Type}.\, A : \mathbf{Type}}$$

We could add this axiom, *in theory*, but *in reality*:

1. Some type theories do not admit *impredicativity* (Agda, Coq for computational Types (by default))
2. Some type theories do not admit *parametricity* (Agda, Coq by default)
3. *Incompatible* with *set theory*:
    1. impredicativity + large elimination + excluded middle *implies False*
    2. impredicativity + relevance + excluded middle *implies False*

Dependent-Type-Preserving Closure Conversion

How do we implement a ***dependently typed closure***?

$$\Gamma^+ \vdash \langle \Gamma^+, f^+ \rangle : Closure(x : A^+ \to B^+)$$

1. not as pairs
2. not as existential pairs

Dependent-Type-Preserving Closure Conversion

How do we implement a ***dependently typed closure***?

$$\Gamma^+ \vdash \langle \Gamma^+, f^+ \rangle : Closure(x : A^+ \to B^+)$$

1. not as pairs
2. not as existential pairs
3. as …. closures

Pros:

- Works

Cons:

- Require proof of consistency
- Unclear how past work (optimizations?) applies

Closure Axioms

Closure Axioms

Well-typed code pointer

$$\frac{\cdot, \mathbf{n} : \mathbf{A}', \mathbf{x} : \mathbf{A} \vdash \mathbf{e} : \mathbf{B}}{\Gamma \vdash \boldsymbol{\lambda}\, \mathbf{n} : \mathbf{A}', \mathbf{x} : \mathbf{A}.\, \mathbf{e} : \mathbf{Code}\,(\mathbf{n} : \mathbf{A}', \mathbf{x} : \mathbf{A}).\, \mathbf{B}}\ [\textsc{Code}]$$

Closure Axioms

Well-typed code pointer

$$\frac{\cdot, \mathbf{n} : \mathbf{A}', \mathbf{x} : \mathbf{A} \vdash \mathbf{e} : \mathbf{B}}{\Gamma \vdash \lambda\, \mathbf{n} : \mathbf{A}', \mathbf{x} : \mathbf{A}.\, \mathbf{e} : \mathbf{Code}\,(\mathbf{n} : \mathbf{A}', \mathbf{x} : \mathbf{A}).\, \mathbf{B}}\ [\textsc{Code}]$$

A code pointer,
pointing to **e**,
which expects two arguments, **n** and **x**

Closure Axioms

Well-typed code pointer

$$\frac{\cdot, \mathbf{n} : A', \mathbf{x} : A \vdash \mathbf{e} : B}{\Gamma \vdash \lambda\, \mathbf{n} : A', \mathbf{x} : A.\, \mathbf{e} : \mathbf{Code}\,(\mathbf{n} : A', \mathbf{x} : A).\, B}\ [\textsc{Code}]$$

Has a dependent code type
(A can depend on n,
B can depend on n and x)

Closure Axioms

Well-typed code pointer

$$\frac{\cdot, \mathbf{n} : A', \mathbf{x} : A \vdash \mathbf{e} : B}{\Gamma \vdash \lambda\, \mathbf{n} : A', \mathbf{x} : A.\, \mathbf{e} : \mathbf{Code}\,(\mathbf{n} : A', \mathbf{x} : A).\, B}\ [\textsc{Code}]$$

When **e** is a well-typed block of code

Closure Axioms

Well-typed code pointer

$$\frac{\cdot, \mathbf{n} : \mathbf{A}', \mathbf{x} : \mathbf{A} \vdash \mathbf{e} : \mathbf{B}}{\Gamma \vdash \lambda\, \mathbf{n} : \mathbf{A}', \mathbf{x} : \mathbf{A}.\, \mathbf{e} : \mathbf{Code}\,(\mathbf{n} : \mathbf{A}', \mathbf{x} : \mathbf{A}).\, \mathbf{B}}\ [\textsc{Code}]$$

When **e** is a well-typed block of code
with references only to declared arguments **n** and **x**.

Closure Axioms

Well-typed closure

$$\frac{\Gamma \vdash e : \mathbf{Code}\,(n : A', x : A).\, B \qquad \Gamma \vdash e' : A'}{\Gamma \vdash \langle\!\langle e, e' \rangle\!\rangle : \Pi\, x : A[e'/n].\, B[e'/n]}\ [\textsc{Clo}]$$

A closure has code pointer e, data e’

Closure Axioms

Well-typed closure

$$\frac{\Gamma \vdash e : \mathbf{Code}\,(n : A', x : A).\, B \qquad \Gamma \vdash e' : A'}{\Gamma \vdash \langle\langle e, e' \rangle\rangle : \Pi\, x : A[e'/n].\, B[e'/n]}\ [\textsc{Clo}]$$

With the data e’ substituted for n in the types (dependent closure type)

Closure Axioms

Well-typed code pointer

$$\frac{\cdot, \mathbf{n} : \mathbf{A}', \mathbf{x} : \mathbf{A} \vdash \mathbf{e} : \mathbf{B}}{\Gamma \vdash \boldsymbol{\lambda}\, \mathbf{n} : \mathbf{A}', \mathbf{x} : \mathbf{A}.\, \mathbf{e} : \mathbf{Code}\,(\mathbf{n} : \mathbf{A}', \mathbf{x} : \mathbf{A}).\, \mathbf{B}}\ [\textsc{Code}]$$

Well-typed closure

$$\frac{\Gamma \vdash \mathbf{e} : \mathbf{Code}\,(\mathbf{n} : \mathbf{A}', \mathbf{x} : \mathbf{A}).\, \mathbf{B} \qquad \Gamma \vdash \mathbf{e}' : \mathbf{A}'}{\Gamma \vdash \langle\!\langle \mathbf{e}, \mathbf{e}' \rangle\!\rangle : \Pi\, \mathbf{x} : \mathbf{A}[\mathbf{e}'/\mathbf{n}].\, \mathbf{B}[\mathbf{e}'/\mathbf{n}]}\ [\textsc{Clo}]$$

Closure Axioms

Well-typed code pointer

We show

1. Logical Consistency
   Orthogonal to impredicativity, relevance, parametricity, large elimination, and set theory
2. Type safety
   IR programs *type* and *memory safe*; *validate* compiler output and *enforce safe linking* via *type checking*

Closure Conversion

$$(\Pi\, x : A.\, B)^+ \stackrel{\text{def}}{=} \Pi\, x : A^+.\, B^+$$

$$(\lambda\, x : A.\, e)^+ \stackrel{\text{def}}{=} \langle\langle \; , \langle x_i \ldots \rangle \rangle\rangle$$

where $x_i : A_i \ldots$ are the free variables of e and A

Pair code pointer with free variables, as explicit data (standard)

Closure Conversion

$$(\Pi\, x : A.\, B)^+ \stackrel{\text{def}}{=} \mathbf{\Pi\, x : A^+.\, B^+}$$

$$(\lambda\, x : A.\, e)^+ \stackrel{\text{def}}{=} \langle\!\langle (\mathbf{\lambda}\, (\mathbf{n} : \quad), \mathbf{x} : \quad).$$

$$\mathbf{let}\ \langle \mathbf{x}_i \ldots \rangle = \mathbf{n}\ \mathbf{in}\ \mathbf{e}), \langle \mathbf{x}_i \ldots \rangle \rangle\!\rangle$$

where $x_i : A_i \ldots$ are the free variables of e and A

Bind free variables in ***term*** by projecting from ***data argument*** (standard)

Closure Conversion

$$(\Pi\, x : A.\, B)^+ \stackrel{\text{def}}{=} \mathbf{\Pi\, x : A^+.\, B^+}$$

$$(\lambda\, x : A.\, e)^+ \stackrel{\text{def}}{=} \mathbf{\langle\!\langle (\lambda\, (n : \ldots), x : \text{let } \langle x_i \ldots \rangle = n \text{ in } A^+).\ \text{let } \langle x_i \ldots \rangle = n \text{ in } e), \langle x_i \ldots \rangle \rangle\!\rangle}$$

where $x_i : A_i \ldots$ are the free variables of e and A

Bind free variables in *type annotation* (*dependency*)

Closure Conversion

$$(\Pi\, x : A.\, B)^+ \stackrel{\text{def}}{=} \mathbf{\Pi}\, \mathbf{x} : A^+.\, B^+$$

$$(\lambda\, x : A.\, e)^+ \stackrel{\text{def}}{=} \langle\!\langle (\boldsymbol{\lambda}\, (\mathbf{n} : \Sigma\, (\mathbf{x}_i : A_i^+ \ldots),\, \mathbf{x} : \mathbf{let}\, \langle \mathbf{x}_i \ldots \rangle = \mathbf{n}\, \mathbf{in}\, A^+).\, \mathbf{let}\, \langle \mathbf{x}_i \ldots \rangle = \mathbf{n}\, \mathbf{in}\, \mathbf{e}),\, \langle \mathbf{x}_i \ldots \rangle \rangle\!\rangle$$

where $x_i : A_i \ldots$ are the free variables of e and A

Bind each $\mathbf{x}_i$ in the type annotation for $\mathbf{x}_{i+1}$ (*dependency*)

Type Preservation

Theorem. (Type Preservation)

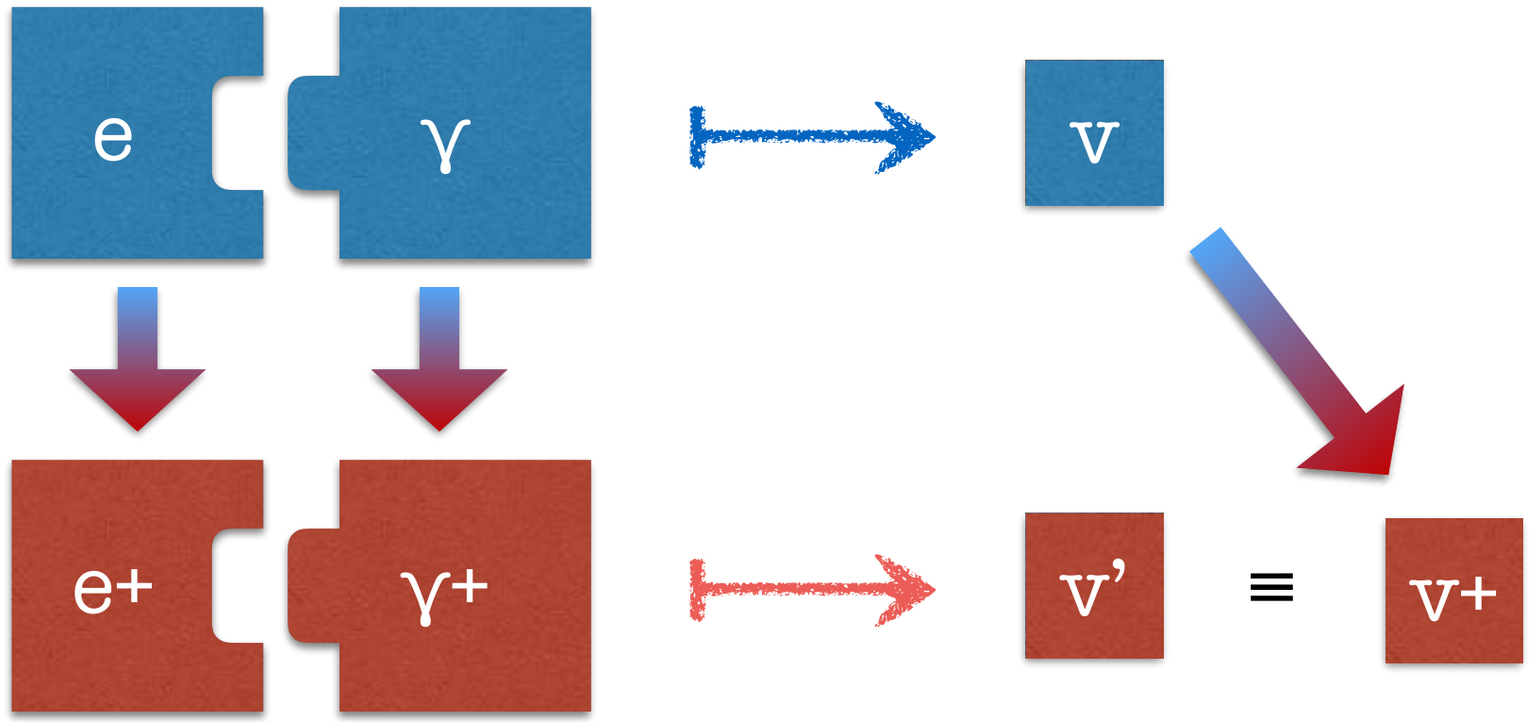
If $e : A$

then $e^+ : A^+$ translates to

Type Preservation

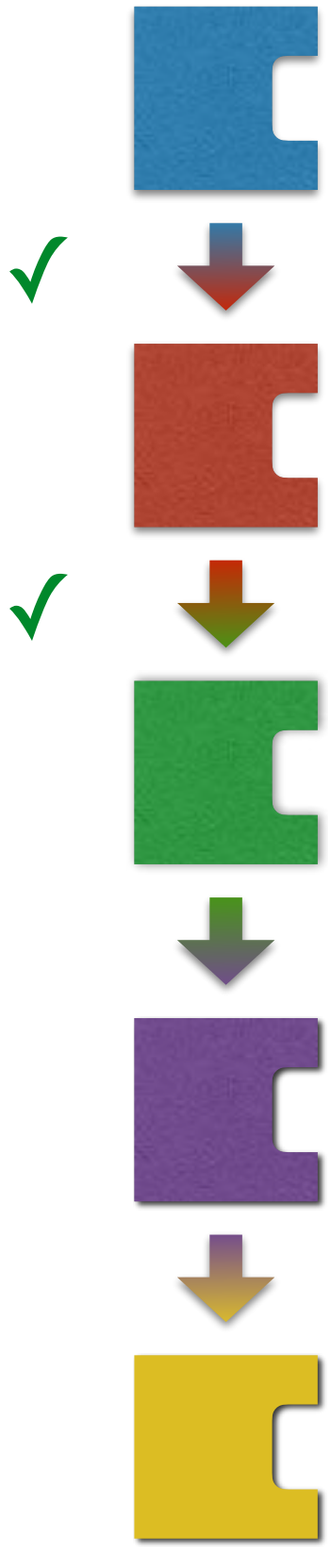
Theorem. (Correctness of Separate Compilation)

If



then

Future Work

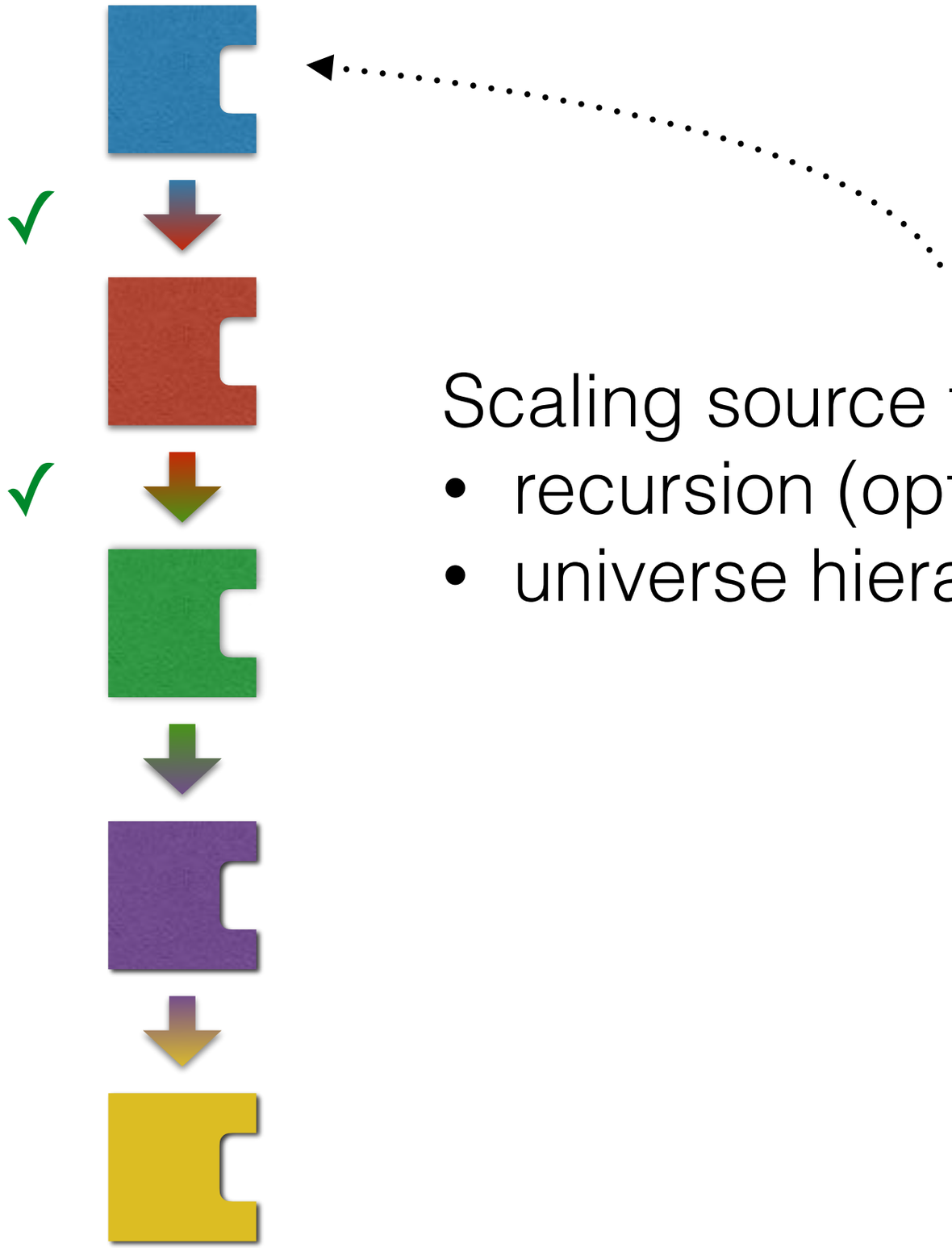


✓ Closure Conversion

Allocation

Code generation

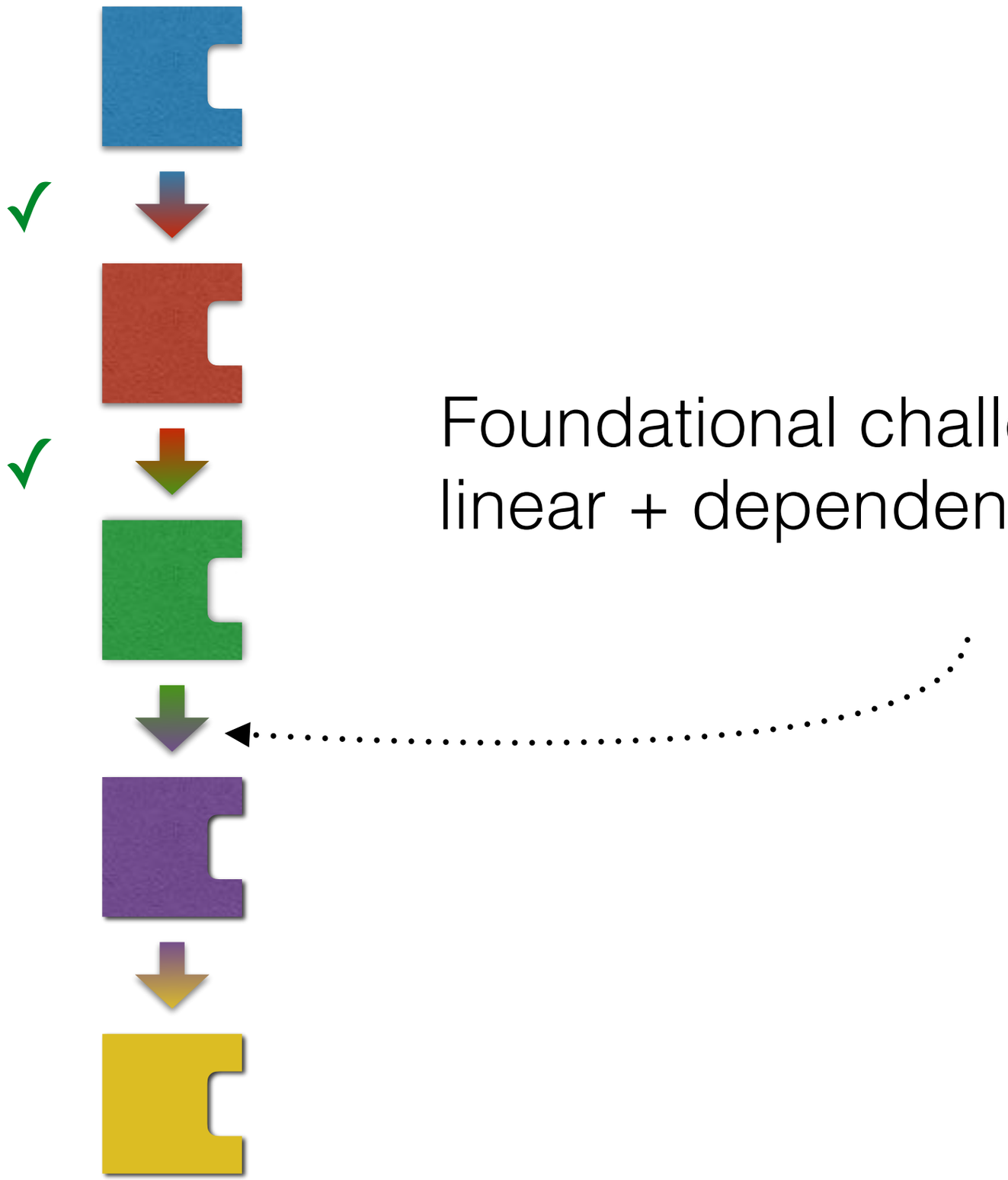
Future Work



Scaling source feature set:

- recursion (optimizing)
- universe hierarchy

Future Work



Foundational challenges:
linear + dependent types

Typed Closure Conversion of the Calculus of Constructions

https://williamjbowman.com/#cccc

Dependently typed closures require abandoning past work on type preservation (existential types) to be practical.