Type-Preserving CPS Translation of Σ and Π Types is Not Not Possible

William J. Bowman, Nick Rioux,
Youyou Cong, Amal Ahmed

Type-Preserving CPS Translation of Σ and Π Types is Not Not Possible

William J. Bowman, Nick Rioux,
Youyou Cong, Amal Ahmed

Dependent types

What are they good for?

Dependent types

What are they good for?

Well.

Dependent types

What are they good for?

Well.

Everything, apparently

Verified in Coq!

- CompCert
- CertiKOS
- Vellvm
- RustBelt
- CertiCrypt

...

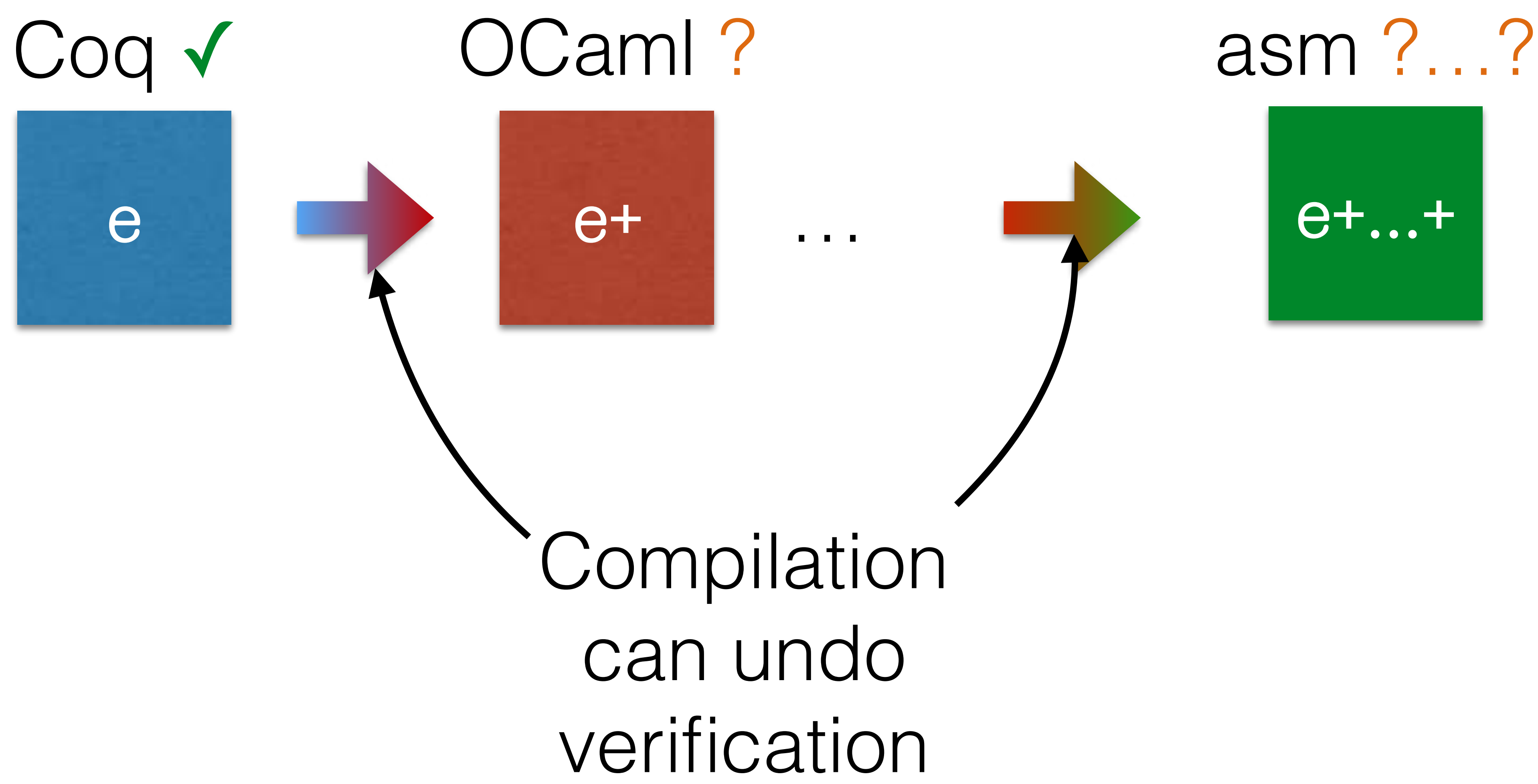
Story of a verified program

Coq

e

Story of a verified program

Story of a verified program



CertiCoq

Compiler correctness!

A correct compilation story

Verify that the program we run is the program we verified

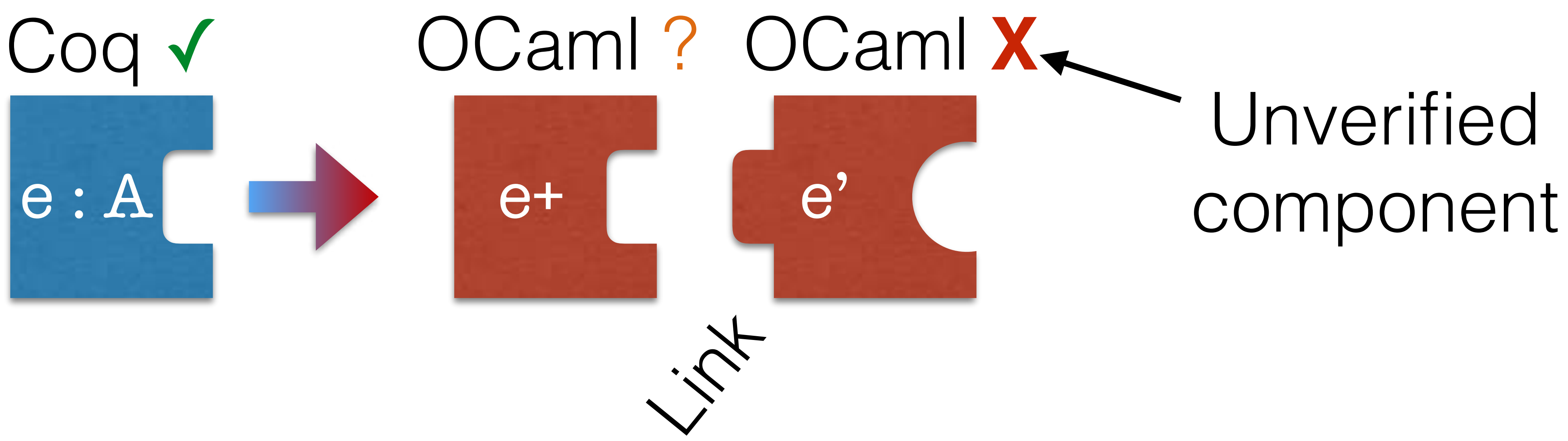
Compiler correctness
is not the whole story

Correctness is the
“whole program” story

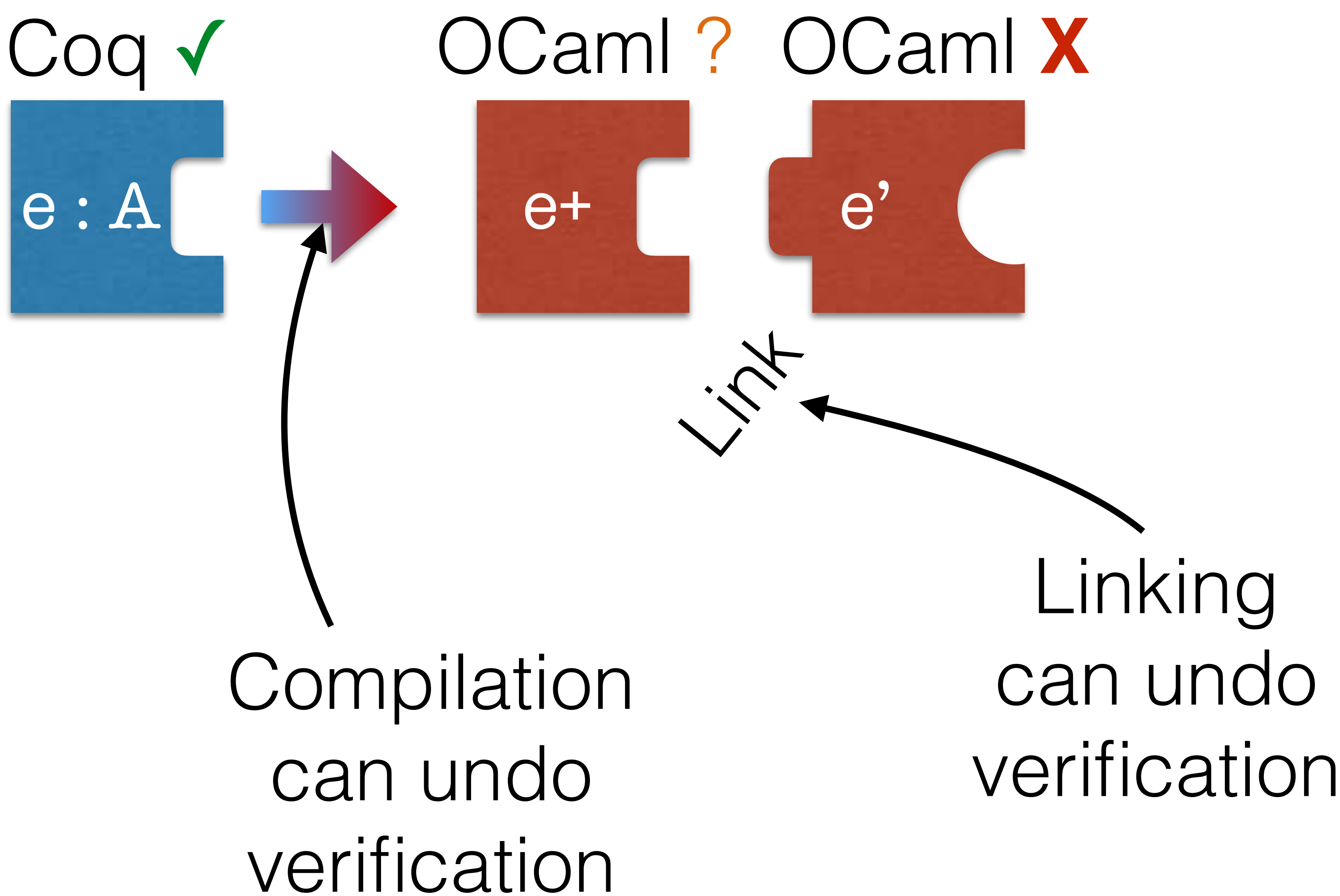
Why type preservation?

Because we do not write whole programs.

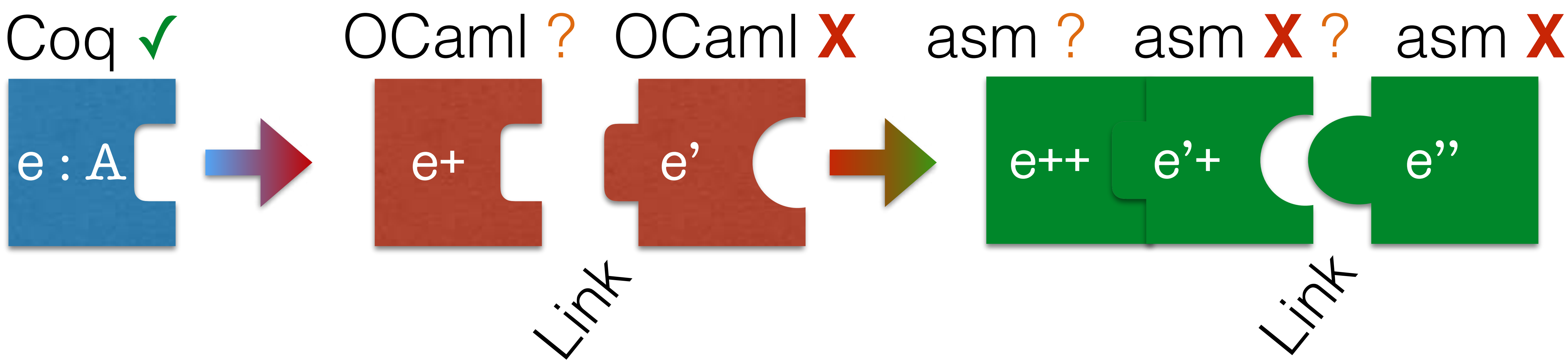
Story of a verified *component*



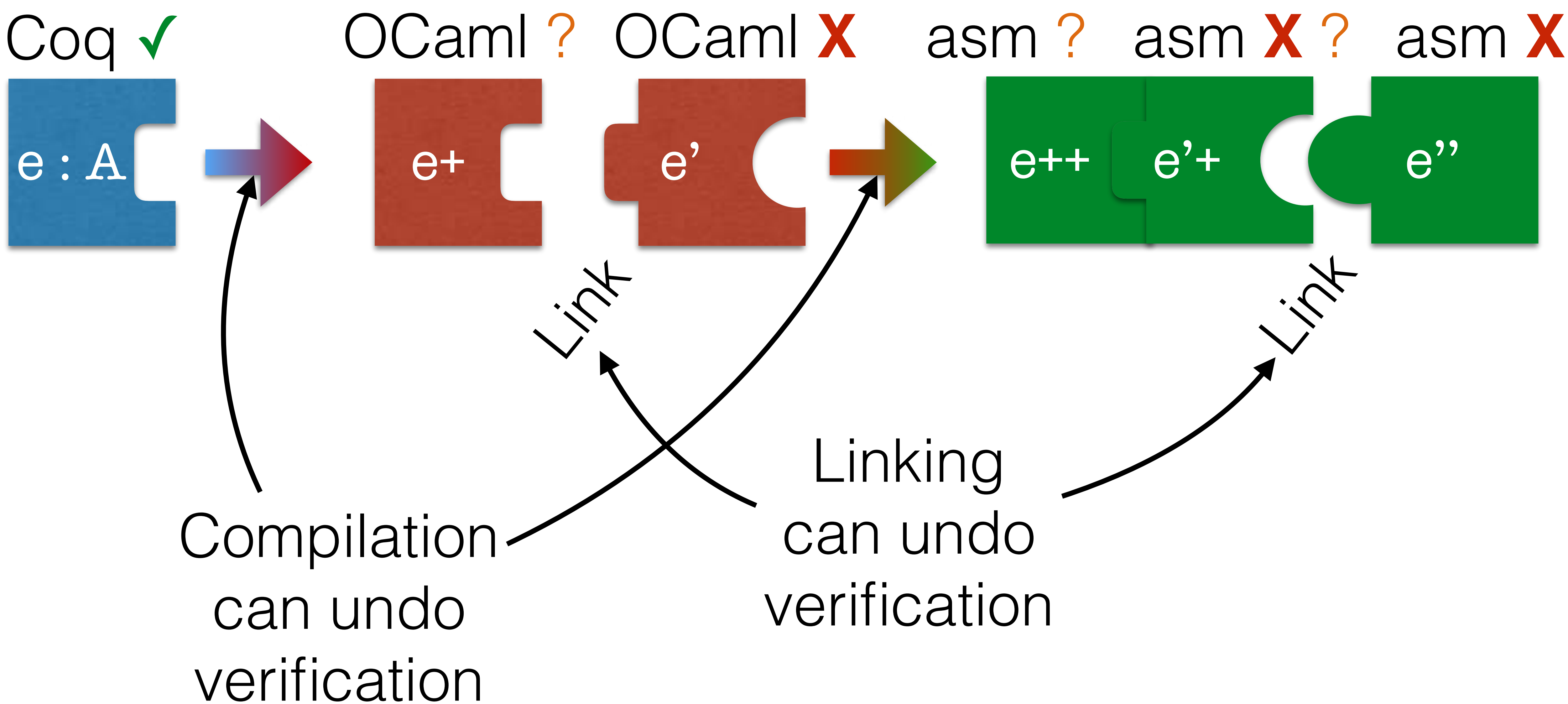
Story of a verified *component*



Story of a verified *component*



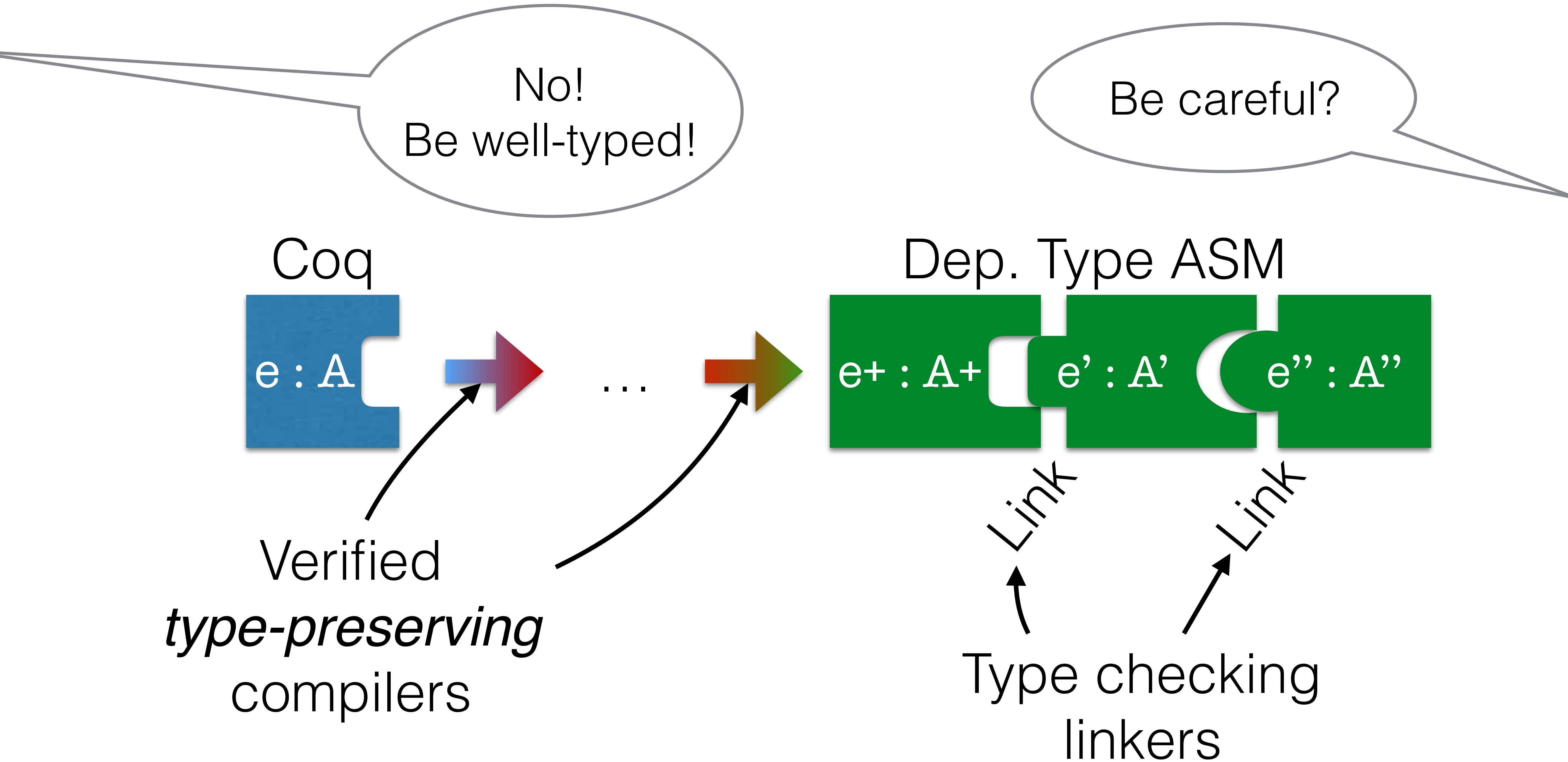
Story of a verified *component*



```
> 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
e : A
…
Dep. Type ASM
e+ : A+
e’ : A’
e’’ : A’’
Link
Link
Verified
type-preserving
compilers
Type checking
linkers

Great, so let’s
preserve dependent types

Type Preservation 101

From System F to Typed Assembly Language

GREG MORRISETT and DAVID WALKER
Cornell University
KARL CRARY
Carnegie Mellon University
and
NEAL GLEW
Cornell University

"a sequence of type-preserving transformations, including CPS …"

language abstractions, such as closures, tuples, and user-defined abstract data typ
system ensures that well-typed programs cannot violate these abstractions. In addi
ing constructs admit many low-level compiler optimizations. Our translation to TA
as a sequence of type-preserving transformations, including CPS and closure conve

Type Preservation 101

From System F to Typed Assembly Language

GRE **CPS Translating Inductive and Coinductive Types**

Corn

KAR [Extended Abstract]

Carn

and Gilles Barthe Tarmo Uustalu*

NEA

Cornell University

"*No* [CPS] translation *is possible* along the same lines for small Σ-types and sum types with dependent case analysis."

ing constructs admit many low-level compiler optimizations. Our translation to TA
as a sequence of type-preserving transformations, including CPS and closure conve

Type Preservation 101

From System F to Typed Assembly Language

GRE

Corn

KAR

Carn

and

NEA

Cornell University

CPS Translating Inductive and Coinductive Types

[Extended Abstract]

Gilles Barthe

Tarmo Uustalu*

“*No* [CPS] translation ***is possible*** <u>along the same lines</u> for small Σ-types and sum types with dependent case analysis.”

ing constructs admit many low-level compiler optimizations. Our translation to TA
as a sequence of type-preserving transformations, including CPS and closure conve

not-not is not possible

Why would CPS be hard?

Intuitively, CPS translates every

expression of type $\mathtt{A}$

into a

computation of type $(\mathtt{A} \to \bot) \to \bot$

i.e. $(\neg\neg\mathtt{A})$

Why would CPS be hard?

Equivalence of expressions

$$e \equiv e' : A$$

$$\text{IF} \quad \mathrm{eval}(e) \equiv \mathrm{eval}(e') : A$$

Why would CPS be hard?

Equivalence of expressions

$$e \equiv e' : A$$

$$\text{IF} \quad \mathrm{eval}(e) \equiv \mathrm{eval}(e') : A$$

Becomes

Equivalence of computations

$$c \equiv c' : \neg\neg A$$

$$\text{IF} \quad c\ ? \equiv c'\ ?' : \bot$$

Why would CPS be hard?

Because deciding equivalence of computations is hard.

Equivalence of expressions

$$\begin{array}{ll} & e \equiv e' : A \\ \text{IF} & \mathrm{eval}(e) \equiv \mathrm{eval}(e') : A \end{array}$$

Becomes

Equivalence of computations

$$\begin{array}{ll} & c \equiv c' : \neg\neg A \\ \text{IF} & c\ \mathbf{?} \equiv c'\ \mathbf{?'} : \bot \end{array}$$

not-not is not possible

now with MATH!

Goal: *Type-preserving* CPS translation

Theorem. (Type Preservation)

If $e : A$

then translates to

$e+ : A+$

Goal: *Type-preserving* CPS translation

Problem: For CBN, ***Σ types*** (strong dependent pairs)

$$\frac{\Gamma \vdash e : \Sigma\, x : A.\, B}{\Gamma \vdash \mathrm{snd}\; e : B[\mathrm{fst}\; e/x]}\ [\textsc{Snd}]$$

Barthe and Uustalu 2002

`e` is pair of an `A` and a `B`

$$\frac{\Gamma \vdash \boxed{e} : \Sigma\, x : \boxed{A}.\ \boxed{B}}{\Gamma \vdash \mathsf{snd}\ e : B[\mathsf{fst}\ e / x]}\ [\textsc{Snd}]$$

$\mathtt{e}$ is pair of an $\mathtt{A}$ and a $\mathtt{B}$

$$\frac{\Gamma \vdash e : \Sigma\, x : A.\, B}{\Gamma \vdash \mathsf{snd}\ e : B[\mathsf{fst}\ e/x]}\ [\textsc{Snd}]$$

where
$\mathtt{B}$ (a type) can refer to
$\mathtt{x}$ (a term var. standing for $(\mathtt{fst\ e})$)

`e` is pair of an `A` and a `B`

$$\frac{\Gamma \vdash e : \Sigma\, x : A.\, B}{\Gamma \vdash \mathsf{snd}\ e : B[\mathsf{fst}\ e / x]}\ [\textsc{Snd}]$$

where

`B` (a type) can refer to

`x` (a term var. standing for `(fst e)`)

Goal: *Type-preserving* CPS translation

$$\frac{\Gamma \vdash e : \Sigma\, x : A.\, B}{\Gamma \vdash \mathsf{snd}\; e : B[\mathsf{fst}\; e / x]}\;[\textsc{Snd}]$$

$$\mathbf{y} := e^{+}; \mathbf{snd}\; \mathbf{y}$$

Barthe and Uustalu 2002

Goal: *Type-preserving* CPS translation

$$\frac{\Gamma \vdash e : \Sigma\, x : A.\, B}{\Gamma \vdash \mathsf{snd}\ e : B[\mathsf{fst}\ e / x]}\ [\textsc{Snd}]$$

Need:

$$\mathbf{y} := e^{+}; \mathbf{snd}\,\mathbf{y} : B^{+}[(\mathsf{fst}\ e)^{+} / \mathbf{x}]$$

Barthe and Uustalu 2002

Goal: *Type-preserving* CPS translation

$$\frac{\Gamma \vdash e : \Sigma\, x : A.\, B}{\Gamma \vdash \mathsf{snd}\ e : B[\mathsf{fst}\ e / x]}\ [\textsc{Snd}]$$

Need:

$$\mathbf{y} := e^{+}; \mathbf{snd}\,\mathbf{y} : B^{+}[(\mathsf{fst}\ e)^{+} / \mathbf{x}]$$

Have:

$$\mathbf{y} := e^{+}; \mathbf{snd}\,\mathbf{y} : B^{+}[\mathbf{fst}\,\mathbf{y} / \mathbf{x}]$$

Barthe and Uustalu 2002

Goal: *Type-preserving* CPS translation

$$\frac{\Gamma \vdash e : \Sigma\, x : A.\, B}{\Gamma \vdash \mathsf{snd}\ e : B[\mathsf{fst}\ e/x]}\ [\textsc{Snd}]$$

Need:

$$\mathbf{y} := e^{+}; \mathbf{snd}\ \mathbf{y} : B^{+}[(\mathsf{fst}\ e)^{+}/\mathbf{x}]$$

Have:

$$\mathbf{y} := e^{+}; \mathbf{snd}\ \mathbf{y} : B^{+}[\mathbf{fst}\ \mathbf{y}/\mathbf{x}]$$

Suffices:

$$\mathbf{fst}\ \mathbf{y} \equiv (\mathsf{fst}\ e)^{+}$$

Barthe and Uustalu 2002

Goal: *Type-preserving* **CPS** translation

Instead of $\mathbf{y} := e^{+}; \mathbf{snd}\,\mathbf{y} : B^{+}[(\mathrm{fst}\ e)^{+}/\mathbf{x}]$

We have $e^{+}\ (\boldsymbol{\lambda}\,\mathbf{y} : (\Sigma\,\mathbf{x} : A^{+}.\,B^{+}).\,(\mathbf{snd}\,\mathbf{y}))$

$\boxed{\text{Suffices:}}$

$\mathbf{fst}\,\mathbf{y} \equiv (\mathrm{fst}\ e)^{+}$

Goal: *Type-preserving* **CPS** translation

Instead of $\mathbf{y} := e^{+}; \mathbf{snd}\,\mathbf{y} : B^{+}[(\mathrm{fst}\ e)^{+}/\mathbf{x}]$

We have $e^{+}\ (\boldsymbol{\lambda}\,\mathbf{y} : (\Sigma\,\mathbf{x} : A^{+}.\,B^{+}).\,(\mathbf{snd}\,\mathbf{y}))$

To type check a *lambda*, must assume $\mathbf{y}$ is arbitrary

Suffices:

$\mathbf{fst}\,\mathbf{y} \equiv (\mathrm{fst}\ e)^{+}$

Goal: *Type-preserving* **CPS** translation

Instead of $\mathbf{y} := e^{+}; \mathbf{snd}\,\mathbf{y} : B^{+}[(\mathrm{fst}\; e)^{+}/\mathbf{x}]$

We have $e^{+}\ (\boldsymbol{\lambda}\,\mathbf{y} : (\Sigma\,\mathbf{x} : A^{+}.\, B^{+}).\, (\mathbf{snd}\,\mathbf{y}))$

To type check a *lambda*,
must assume $\mathbf{y}$ is ***arbitrary***

and type check the body

$\boxed{\text{Suffices:}}$

$\mathbf{fst}\,\mathbf{y} \equiv (\mathrm{fst}\; e)^{+}$

Need:

$$\frac{\mathbf{y} : \Sigma\, \mathbf{x} : A^{+} .\, B^{+} \vdash (\mathbf{snd}\, \mathbf{y}) : B^{+}[(\mathsf{fst}\ e)^{+} / \mathbf{x}]}{e^{+}\ (\boldsymbol{\lambda}\, \mathbf{y} : (\Sigma\, \mathbf{x} : A^{+} .\, B^{+}) .\, (\mathbf{snd}\, \mathbf{y}))}$$

Suffices:

$$\mathbf{fst}\, \mathbf{y} \equiv (\mathsf{fst}\ e)^{+}$$

Need:

$$\frac{\mathbf{y} : \Sigma\, \mathbf{x} : A^{+} .\, B^{+} \vdash (\mathbf{snd}\, \mathbf{y}) : B^{+}[(\mathsf{fst}\ e)^{+} / \mathbf{x}]}{e^{+}\ (\lambda\, \mathbf{y} : (\Sigma\, \mathbf{x} : A^{+} .\, B^{+}).\, (\mathbf{snd}\, \mathbf{y}))}$$

Suffices:

$$\mathbf{fst}\ \mathbf{y} \equiv (\mathsf{fst}\ e)^{+}$$

Intuitively,

$\mathbf{y} \equiv \text{value-of}(e^{+})$

Need:

$$\frac{\mathbf{y} : \Sigma\, \mathbf{x} : A^{+} .\, B^{+} \vdash (\mathbf{snd}\, \mathbf{y}) : B^{+}[(\mathsf{fst}\; e)^{+} / \mathbf{x}]}{e^{+}\; (\boldsymbol{\lambda}\, \mathbf{y} : (\Sigma\, \mathbf{x} : A^{+} .\, B^{+}) .\, (\mathbf{snd}\, \mathbf{y}))}$$

Suffices:

$$\mathbf{fst}\; \mathbf{y} \equiv (\mathsf{fst}\; e)^{+}$$

Intuitively,

$$\mathbf{y} \equiv \text{value-of}(e^{+})$$

and

$$\mathbf{fst}\; \text{value-of}(e^{+}) \equiv (\mathsf{fst}\; e)^{+}$$

Need:

$$\frac{\mathbf{y} : \Sigma\, \mathbf{x} : A^{+} .\, B^{+} \vdash (\mathbf{snd}\, \mathbf{y}) : B^{+}[(\mathsf{fst}\ e)^{+} / \mathbf{x}]}{e^{+}\ (\boldsymbol{\lambda}\, \mathbf{y} : (\Sigma\, \mathbf{x} : A^{+} .\, B^{+}).\, (\mathbf{snd}\, \mathbf{y}))}$$

Suffices:

$$\mathbf{fst}\, \mathbf{y} \equiv (\mathsf{fst}\ e)^{+}$$

Intuitively,

$$\mathbf{y} \equiv \text{value-of}(e^{+})$$

and

$$\mathbf{fst}\ \text{value-of}(e^{+}) \equiv (\mathsf{fst}\ e)^{+}$$

Hence

$$\mathbf{fst}\, \mathbf{y} \equiv (\mathsf{fst}\ e)^{+}$$

Need:

$$\frac{\mathbf{y} : \Sigma\, \mathbf{x} : A^+ .\, B^+ \vdash (\mathbf{snd}\, \mathbf{y}) : B^+[(\mathrm{fst}\ e)^+ / \mathbf{x}]}{e^+\ (\lambda\, \mathbf{y} : (\Sigma\, \mathbf{x} : A^+ .\, B^+).\, (\mathbf{snd}\, \mathbf{y}))}$$

Suffices:

$$\mathbf{fst}\ \mathbf{y} \equiv (\mathrm{fst}\ e)^+$$

Intuitively,

$$\mathbf{y} \equiv \text{value-of}(e^+)$$

and

$$\mathbf{fst}\ \text{value-of}(e^+) \equiv (\mathrm{fst}\ e)^+$$

1. What is the “value-of” a CPS’d computation?
2. How do we remember that value in a continuation?

not not-not is
not not possible

a different CPS

~~not not not~~ is

~~not not~~ possible

CPS with answer type polymorphism

Double negation CPS

$\mathtt{c} : \neg\neg\mathtt{A}$

Polymorphic CPS

$\mathtt{c} : \forall\, \alpha.\, (\mathtt{A} \rightarrow \alpha) \rightarrow \alpha$

CPS with answer type polymorphism

Double negation CPS

$\mathtt{c} : \neg\neg A$

$\mathtt{c}\ (\lambda\ (\mathtt{v})\ \ldots) : \bot$

Polymorphic CPS

$\mathtt{c} : \forall\, \alpha.\ (A \to \alpha) \to \alpha$

$\mathtt{c}\ B\ (\lambda\ (\mathtt{v})\ \ldots) : B$

CPS with answer type polymorphism

Double negation CPS

$\mathtt{c} : \neg\neg A$

$\mathtt{c}\ (\lambda\ (\mathtt{v})\ \ldots) : \boxed{\bot}$

Polymorphic CPS

$\mathtt{c} : \forall\, \alpha.\ (A \to \alpha) \to \alpha$

$\mathtt{c}\ B\ (\lambda\ (\mathtt{v})\ \ldots) : \boxed{B}$

1. The value of a CPS'd computation

The key is equivalence

Before

	$c \equiv c' : \neg\neg A$
IF	$c\ ?\ \equiv c'\ ?' : \perp$

The key is equivalence

Before

	$c \equiv c' : \neg\neg A$
IF	$c\ ? \equiv c'\ ?' : \bot$

After

	$c \equiv c' : \forall\, \alpha.\ (A \to \alpha) \to \alpha$
IF	$c\ A\ id \equiv c'\ A\ id : A$

The key is equivalence

Before

$$c \equiv c' : \neg\neg A$$

$$\text{IF} \quad c\ \textbf{?} \equiv c'\ \textbf{?}' : \bot$$

Meaningless garbage

After

$$c \equiv c' : \forall \alpha.\ (A \to \alpha) \to \alpha$$

$$\text{IF} \quad c\ A\ \mathrm{id} \equiv c'\ A\ \mathrm{id} : A$$

Meaningful underlying *value* of type A

Add to our typed CPS target language

$$\frac{}{\Gamma \vdash (e_1\ B\ (\lambda\, x : A.\ e_2)) \equiv (\lambda\, x : A.\ e_2)\ (e_1\ A\ \mathbf{id})}\ [\equiv\text{-}\textsc{Cont}]$$

Based on
"continuation shuffling"
aka
"parametricity condition"
aka
"naturality"

When computation e_1

$$\frac{}{\Gamma \vdash (e_1\ B\ (\lambda\, x : A.\ e_2)) \equiv (\lambda\, x : A.\ e_2)\ (e_1\ A\ \mathbf{id})}\ [\equiv\text{-}\textsc{Cont}]$$

When computation e_1
is applied to answer type B
and continuation

$$\frac{}{\Gamma \vdash (e_1 \; B \; (\lambda\, x : A.\, e_2)) \equiv (\lambda\, x : A.\, e_2)\; (e_1 \; A \; \mathbf{id})} \; [\equiv\text{-}\textsc{Cont}]$$

When computation $\mathtt{e_1}$
is applied to answer type $\mathtt{B}$
and continuation

$$\frac{}{\Gamma \vdash (\mathbf{e}_1\ \mathbf{B}\ (\lambda\, \mathbf{x} : \mathbf{A}.\ \mathbf{e}_2)) \equiv (\lambda\, \mathbf{x} : \mathbf{A}.\ \mathbf{e}_2)\ (\mathbf{e}_1\ \mathbf{A}\ \mathbf{id})}\ [\equiv\text{-}\textsc{Cont}]$$

"Shuffle" this into
continuation, as a function

When computation $\mathtt{e_1}$
is applied to answer type $\mathtt{B}$
and continuation

$$\frac{}{\Gamma \vdash (\mathbf{e_1}\ \mathbf{B}\ (\lambda\, \mathbf{x} : \mathbf{A}.\ \mathbf{e_2})) \equiv (\lambda\, \mathbf{x} : \mathbf{A}.\ \mathbf{e_2})\ (\mathbf{e_1}\ \mathbf{A}\ \mathbf{id})}\ [\equiv\text{-}\textsc{Cont}]$$

"Shuffle" this into
continuation, as a function,
applied to
value of $\mathtt{e_1}$

$$\frac{}{\Gamma \vdash (\mathbf{e}_1\ \mathbf{B}\ (\lambda\, \mathbf{x} : \mathbf{A}.\ \mathbf{e}_2)) \equiv (\lambda\, \mathbf{x} : \mathbf{A}.\ \mathbf{e}_2)\ (\mathbf{e}_1\ \mathbf{A}\ \mathbf{id})}\ [\equiv\text{-}\textsc{Cont}]$$

$$\text{value-of}(\mathbf{e}_1) \stackrel{\text{def}}{=} \mathbf{e}_1\ \mathbf{A}\ \mathbf{id}$$

Modeling ≡-Cont

Parametricity Translation

CC w/ ≡-Cont

Extensional CC

Prove extensional ≡-Cont
as a free theorem.

Toward Type Preservation

Need:

$$\frac{\mathbf{y} : \Sigma\, \mathbf{x} : A^+ .\, B^+ \vdash (\mathbf{snd}\,\mathbf{y}) : B^+[(\mathsf{fst}\ e)^+ / \mathbf{x}]}{e^+\ (\lambda\, \mathbf{y} : (\Sigma\, \mathbf{x} : A^+ .\, B^+).\, (\mathbf{snd}\,\mathbf{y}))}$$

Suffices:

$$\mathbf{fst}\,\mathbf{y} \equiv (\mathsf{fst}\ e)^+$$

Intuitively,

$$\mathbf{y} \equiv e^+\ \mathbf{id}$$

and

$$\mathbf{fst}\,(e^+\ \mathbf{id}) \equiv (\mathsf{fst}\ e)^+$$

Toward Type Preservation

Need: :

$\mathrm{y} : \Sigma$ ... $(\mathrm{fst}\ e)^+$

$(\mathrm{fst}\ e)^+ = e^+\ (\lambda\, \mathrm{y}.\, \mathrm{fst}\, \mathrm{y})$

By ≡-Cont

$$\equiv (\lambda\, \mathrm{y}.\, \mathrm{fst}\, \mathrm{y})\ (e^+\ \mathrm{id})$$

$$\equiv \mathrm{fst}\,(e^+\ \mathrm{id})$$

Intuitively,

$\mathrm{y} \equiv e^+\ \mathrm{id}$

$\mathrm{fst}\,(e^+\ \mathrm{id}) \equiv (\mathrm{fst}\ e)^+$

1. The value of a CPS'd computation

$$\frac{}{\Gamma \vdash (\mathbf{e_1}\ \mathbf{B}\ (\lambda\, \mathbf{x} : \mathbf{A}.\ \mathbf{e_2})) \equiv (\lambda\, \mathbf{x} : \mathbf{A}.\ \mathbf{e_2})\ (\mathbf{e_1}\ \mathbf{A}\ \mathbf{id})}\ [\equiv\text{-}\textsc{Cont}]$$

$$\text{value-of}(\mathbf{e_1}) \stackrel{\text{def}}{=} \mathbf{e_1}\ \mathbf{A}\ \mathbf{id}$$

2. Remember value when jumping to a continuation

Need:

$$\frac{\mathbf{y} : \Sigma\, \mathbf{x} : A^{+}.\, B^{+} \vdash (\mathbf{snd}\, \mathbf{y}) : B^{+}[(\mathsf{fst}\ e)^{+} / \mathbf{x}]}{e^{+}\ (\boldsymbol{\lambda}\, \mathbf{y} : (\Sigma\, \mathbf{x} : A^{+}.\, B^{+}).\, (\mathbf{snd}\, \mathbf{y}))}$$

Suffices:

$$\mathbf{fst}\, \mathbf{y} \equiv (\mathsf{fst}\ e)^{+}$$

Need:

$$\frac{\mathbf{y} : \Sigma\, \mathbf{x} : A^{+} .\, B^{+} \vdash (\mathbf{snd}\; \mathbf{y}) : B^{+}[(\mathsf{fst}\; e)^{+}/\mathbf{x}]}{e^{+}\; (\boldsymbol{\lambda}\, \mathbf{y} : (\Sigma\, \mathbf{x} : A^{+} .\, B^{+}).\, (\mathbf{snd}\; \mathbf{y}))}$$

Suffices:

$$\mathbf{fst}\; \mathbf{y} \equiv (\mathsf{fst}\; e)^{+}$$

Intuitively,

$$\mathbf{y} \equiv \text{value-of}(e^{+})$$

Need:

$$\frac{\mathbf{y} : \Sigma\, \mathbf{x} : A^+ .\, B^+ \vdash (\mathbf{snd}\ \mathbf{y}) : B^+[(\mathrm{fst}\ e)^+/\mathbf{x}]}{e^+\ (\lambda\, \mathbf{y} : (\Sigma\, \mathbf{x} : A^+ .\, B^+).\ (\mathbf{snd}\ \mathbf{y}))}$$

Suffices:

$$\mathbf{fst}\ \mathbf{y} \equiv (\mathrm{fst}\ e)^+$$

Intuitively,

$$\mathbf{y} \equiv \text{value-of}(e^+)$$

But formally, $\mathbf{y}$ is arbitrary

What if we make this equivalence

$$\mathbf{y} \equiv \text{value-of}(e^{+})$$

$$\frac{\mathbf{y} : \Sigma\, \mathbf{x} : A^{+}.\, B^{+} \vdash (\mathbf{snd}\ \mathbf{y}) : B^{+}[(\mathrm{fst}\ e)^{+} / \mathbf{x}]}{e^{+}\ (\boldsymbol{\lambda}\, \mathbf{y} : (\Sigma\, \mathbf{x} : A^{+}.\, B^{+}).\, (\mathbf{snd}\ \mathbf{y}))}$$

part of this typing rule

$$\frac{\mathbf{y} = e^{+}\ \mathbf{id} : \Sigma\,\mathbf{x} : A^{+}.\,B^{+} \vdash (\mathbf{snd}\,\mathbf{y}) : B^{+}[(\mathsf{fst}\ e)^{+}/\mathbf{x}]}{e^{+}\ \mathbf{@}\ (\boldsymbol{\lambda}\,\mathbf{y} : (\Sigma\,\mathbf{x} : A^{+}.\,B^{+}).\,(\mathbf{snd}\,\mathbf{y}))}$$

“Jump-to” form

Jumping to a continuation is not just application.

$$\frac{\Gamma \vdash e : \Pi\, \alpha : *. (A \to \alpha) \to \alpha \qquad \Gamma \vdash B : * \qquad \Gamma, x = e\; A\; \mathrm{id} \vdash e' : B}{\Gamma \vdash e \mathbin{@} B\, (\lambda\, x : A.\, e') : B}\ [\text{T-Cont}]$$

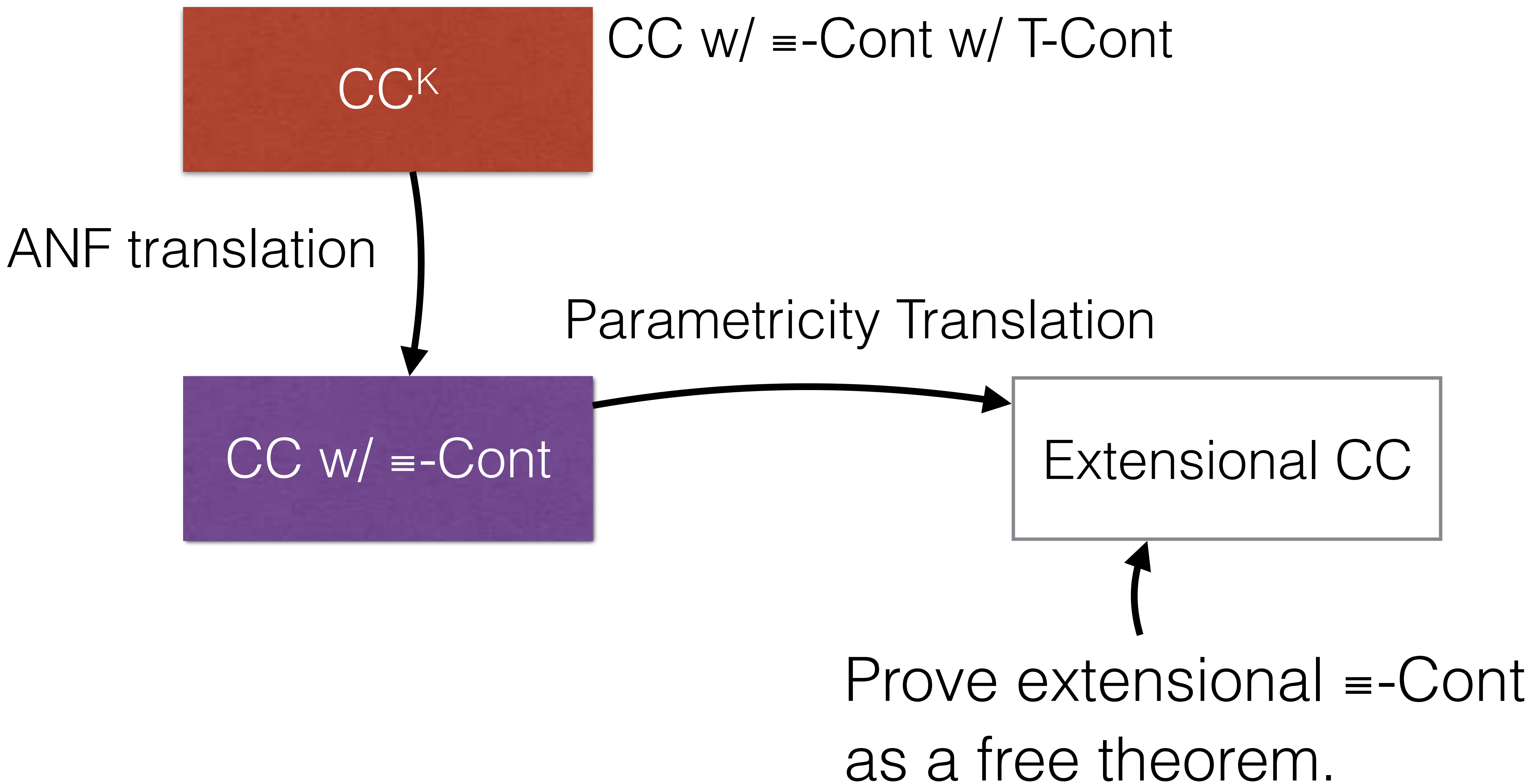
Jumping to a continuation is not just application.

$$\frac{\Gamma \vdash \boxed{e : \Pi\, \alpha : *.\, (A \to \alpha) \to \alpha} \qquad \Gamma \vdash B : * \qquad \Gamma, x = e\ A\ \mathrm{id} \vdash e' : B}{\Gamma \vdash e\ @\ B\ (\lambda\, x : A.\, e') : B}\ [\textsc{T-Cont}]$$

Jumping to a continuation is not just application.

$$\frac{\Gamma \vdash e : \Pi\, \alpha : *. (A \to \alpha) \to \alpha \qquad \Gamma \vdash B : * \qquad \Gamma, \boxed{x = e\ A\ \mathrm{id}} \vdash \boxed{e'} : B}{\Gamma \vdash e\ @\ B\ \boxed{(\lambda\, x : A.\, e')} : B}\ [\text{T-Cont}]$$

Modeling T-Cont



2. Remember value when jumping to a continuation

$$\frac{\Gamma \vdash e : \Pi\, \alpha : *.\, (A \to \alpha) \to \alpha \qquad \Gamma \vdash B : * \qquad \Gamma, x = e\ A\ \mathrm{id} \vdash e' : B}{\Gamma \vdash e\ @\ B\ (\lambda\, x : A.\, e') : B}\ [\textsc{T-Cont}]$$

Type Preservation

Need:

$$\frac{\mathbf{y} = e^{+}\ \mathbf{id} : \Sigma\, \mathbf{x} : A^{+}.\, B^{+} \vdash (\mathbf{snd}\,\mathbf{y}) : B^{+}[(\mathsf{fst}\ e)^{+}/\mathbf{x}]}{e^{+}\ @\ (\boldsymbol{\lambda}\,\mathbf{y} : (\Sigma\, \mathbf{x} : A^{+}.\, B^{+}).\, (\mathbf{snd}\,\mathbf{y}))}$$

Suffices:

$$\mathbf{fst}\,\mathbf{y} \equiv (\mathsf{fst}\ e)^{+}$$

Type Preservation

Need:

$$\frac{\mathbf{y} = e^{+}\ \mathbf{id} : \Sigma\, \mathbf{x} : A^{+}.\, B^{+} \vdash (\mathbf{snd}\, \mathbf{y}) : B^{+}[(\mathsf{fst}\ e)^{+} / \mathbf{x}]}{e^{+}\ @\ (\lambda\, \mathbf{y} : (\Sigma\, \mathbf{x} : A^{+}.\, B^{+}).\, (\mathbf{snd}\, \mathbf{y}))}$$

Suffices:

$$\mathbf{fst}\ \mathbf{y} \equiv (\mathsf{fst}\ e)^{+}$$

By T-Cont,

$$\mathbf{y} \equiv e^{+}\ \mathbf{id}$$

Type Preservation

Need:

$$\frac{\mathbf{y} = e^{+}\ \mathbf{id} : \Sigma\, \mathbf{x} : A^{+}.\, B^{+} \vdash (\mathbf{snd}\ \mathbf{y}) : B^{+}[(\mathrm{fst}\ e)^{+} / \mathbf{x}]}{e^{+}\ @\ (\lambda\, \mathbf{y} : (\Sigma\, \mathbf{x} : A^{+}.\, B^{+}).\, (\mathbf{snd}\ \mathbf{y}))}$$

Suffices:

$$\mathbf{fst}\ \mathbf{y} \equiv (\mathrm{fst}\ e)^{+}$$

By T-Cont,

$$\mathbf{y} \equiv e^{+}\ \mathbf{id}$$

By earlier (uses ≡-Cont)

$$\mathbf{fst}\,(e^{+}\ \mathbf{id}) \equiv (\mathrm{fst}\ e)^{+}$$

Type Preservation

Need:

$$\frac{\mathbf{y} = e^{+}\ \mathbf{id} : \Sigma\, \mathbf{x} : A^{+}.\, B^{+} \vdash (\mathbf{snd}\ \mathbf{y}) : B^{+}[(\mathsf{fst}\ e)^{+} / \mathbf{x}]}{e^{+}\ @\ (\lambda\, \mathbf{y} : (\Sigma\, \mathbf{x} : A^{+}.\, B^{+}).\, (\mathbf{snd}\ \mathbf{y}))}$$

Suffices:

$$\mathbf{fst}\ \mathbf{y} \equiv (\mathsf{fst}\ e)^{+}$$

By T-Cont,

$$\mathbf{y} \equiv e^{+}\ \mathbf{id}$$

By earlier (uses ≡-Cont)

$$\mathbf{fst}\ (e^{+}\ \mathbf{id}) \equiv (\mathsf{fst}\ e)^{+}$$

Hence

$$\mathbf{fst}\ \mathbf{y} \equiv (\mathsf{fst}\ e)^{+}$$

Call-by-value

CBV CPS fails, even for Π

$$\frac{\Gamma \vdash e_1 : \Pi\, x : A.\, B \qquad \Gamma \vdash e_2 : A}{\Gamma \vdash e_1\ e_2 : B[e_2/x]}\ [\textsc{App}]$$

CBV CPS fails, even for Π

$$\frac{\Gamma \vdash e_1 : \Pi\, x : A.\, B \qquad \Gamma \vdash e_2 : A}{\Gamma \vdash e_1\ e_2 : B[e_2/x]}\ [\textsc{App}]$$

Need:

$$\mathbf{y}_1 := e_1^+; \mathbf{y}_2 := e_2^+; \mathbf{y}_1\ \mathbf{y}_2 : B^+[\boxed{e_2^+}/\mathbf{x}]$$

Have:

$$\mathbf{y}_1 := e_1^+; \mathbf{y}_2 := e_2^+; \mathbf{y}_1\ \mathbf{y}_2 : B^+[\boxed{\mathbf{y}_2}/\mathbf{x}]$$

But Π is fine is CBN

$$\frac{\Gamma \vdash e_1 : \Pi\, x : A.\, B \qquad \Gamma \vdash e_2 : A}{\Gamma \vdash e_1\ e_2 : B[e_2/x]}\ [\textsc{App}]$$

CBV:

$$\mathbf{y}_1 := e_1^+; \mathbf{y}_2 := e_2^+; \mathbf{y}_1\ \mathbf{y}_2 : B^+[e_2^+/\mathbf{x}]$$

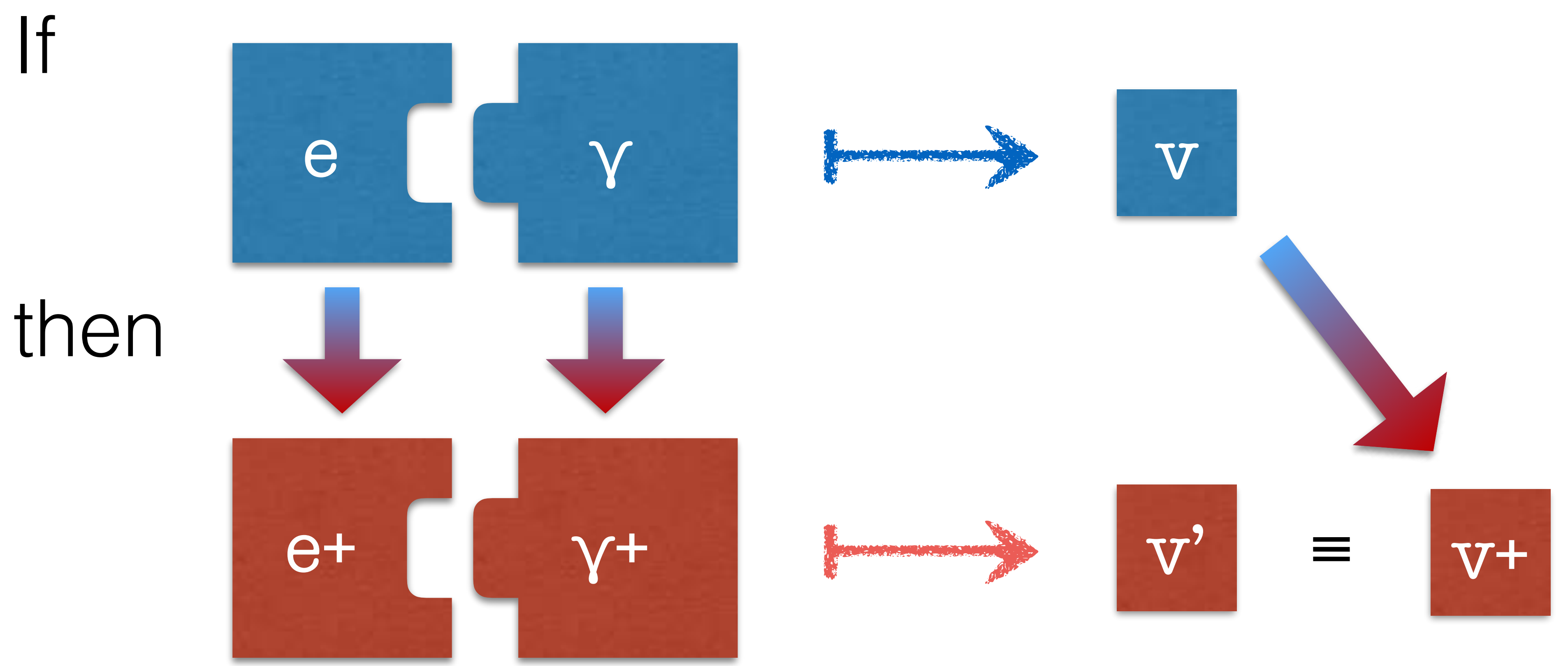
X

CBN:

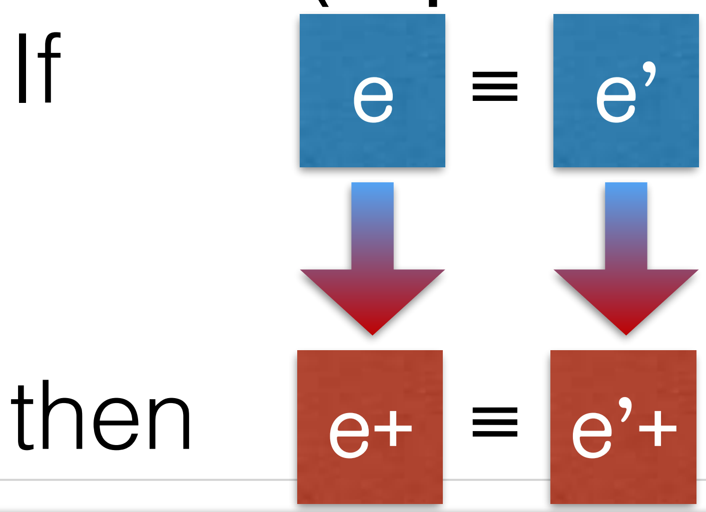
$$\mathbf{y}_1 := e_1^+; \mathbf{y}_1\ e_2^+ : B^+[e_2^+/\mathbf{x}] \quad \checkmark$$

Compiler Correctness, Too

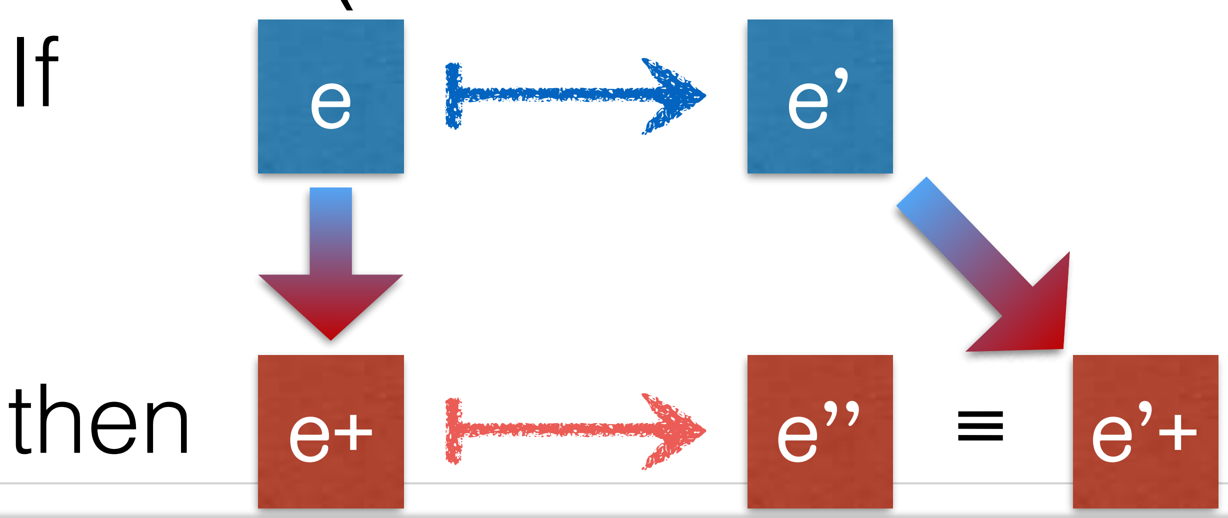
Theorem. (Correctness of Separate Compilation)



Lemma. (Equivalence Preservation)

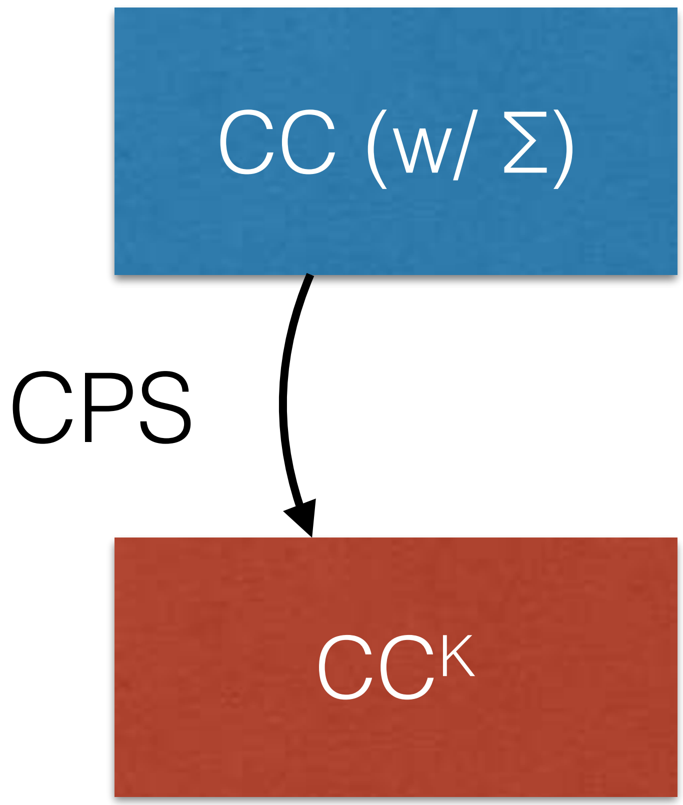


Lemma. (Preservation of Reduction Sequences)

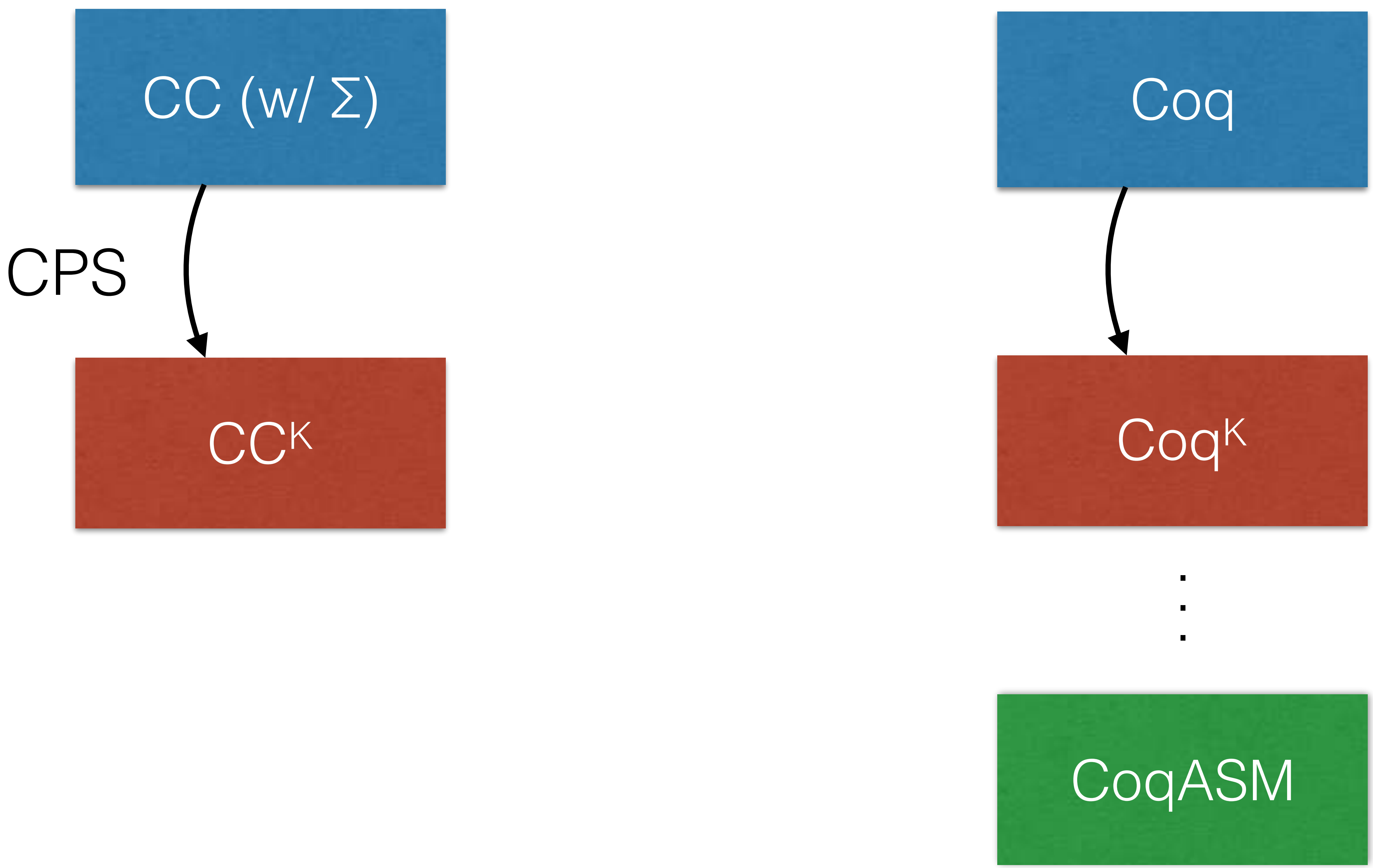


Scaling to Coq

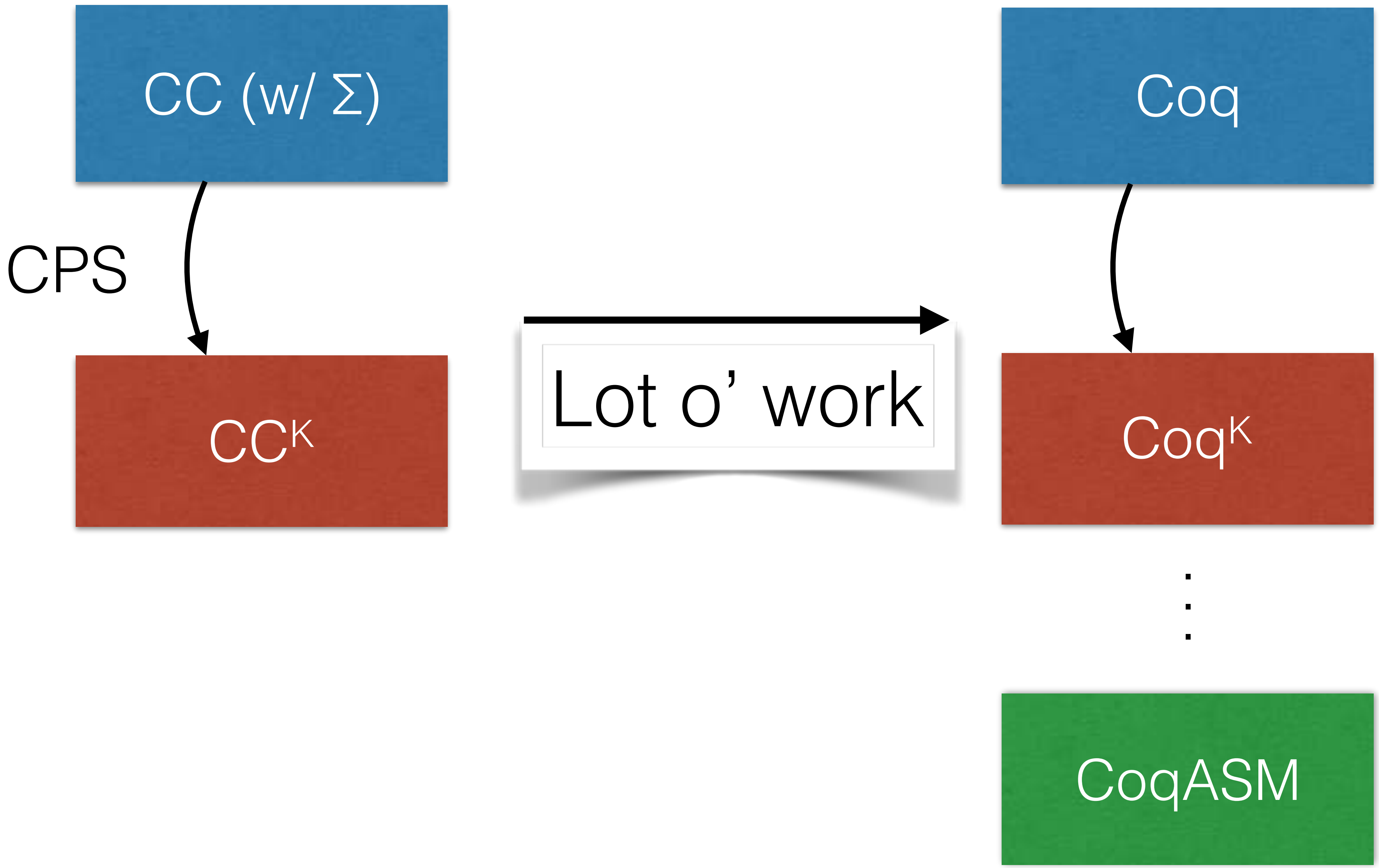
Our compiler so far



Future Work



Future Work



Dependent Case

In 2002, Barthe and Uustalu also prove:

- *No type-preserving CPS translation can exist* with dependent case analysis on sum types.

Dependent Case

In 2002, Barthe and Uustalu also prove:

- *No type-preserving CPS translation* can exist* with dependent case analysis on sum types.

* that admits `call/cc` in certain contexts.

Dependent Case

In 2002, Barthe and Uustalu also prove:

- ***No type-preserving CPS translation* can exist*** with dependent case analysis on sum types.

Answer type polymorphism disallows **call/cc**,
in *any* compiled code.

* that admits **call/cc** in certain contexts.

Dependent Case

In 2002, Barthe and Uustalu also prove:

- ***No type-preserving CPS translation* can exist*** with dependent case analysis on sum types.

Answer type polymorphism disallows **`call/cc`**, in *any* compiled code.

We sketch extending our translation to dependent case

* that admits **`call/cc`** in certain contexts.

Type-Preserving CPS Translation of Σ and Π Types is ~~Not Not~~ **Possible**

williamjbowman.com/#cps-sigma

1. The "value-of" a CPS'd computation and
2. "remember" that value when jumping to a continuation.