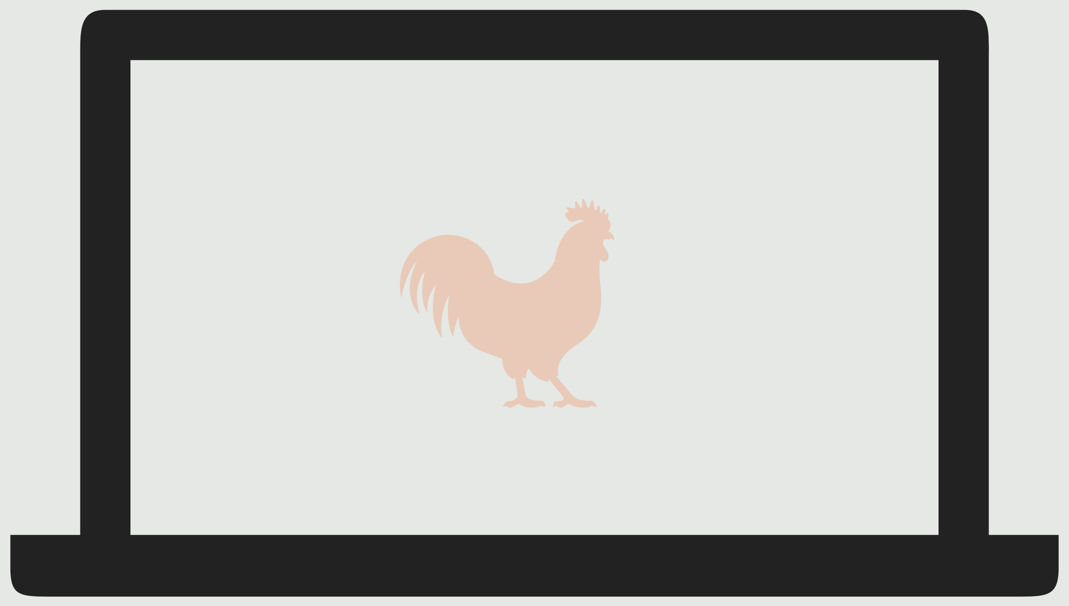
Syntax and Semantics of Type Theory

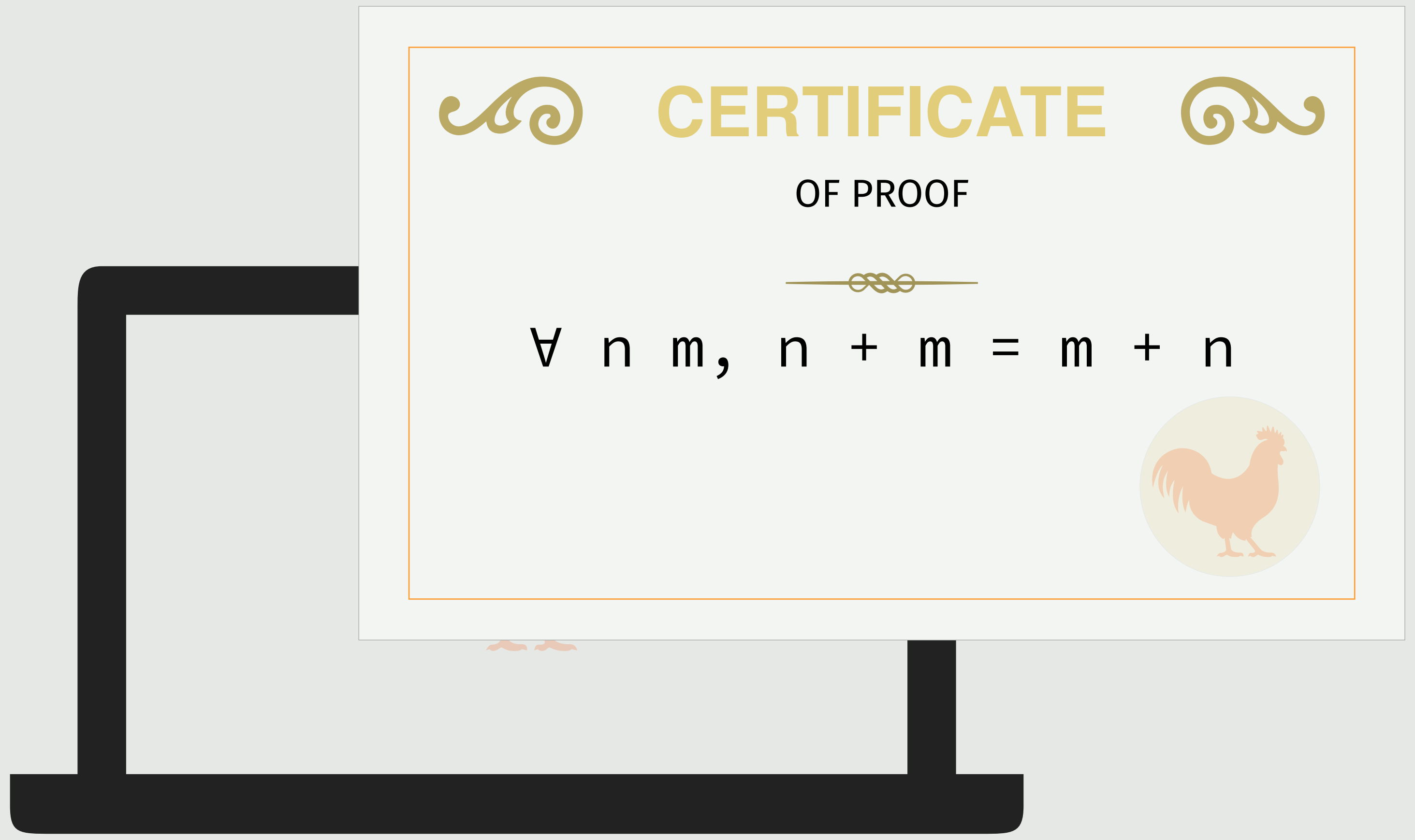
MetaCoq

Sound and complete type checking for Coq, in Coq

Théo **Winterhalter**

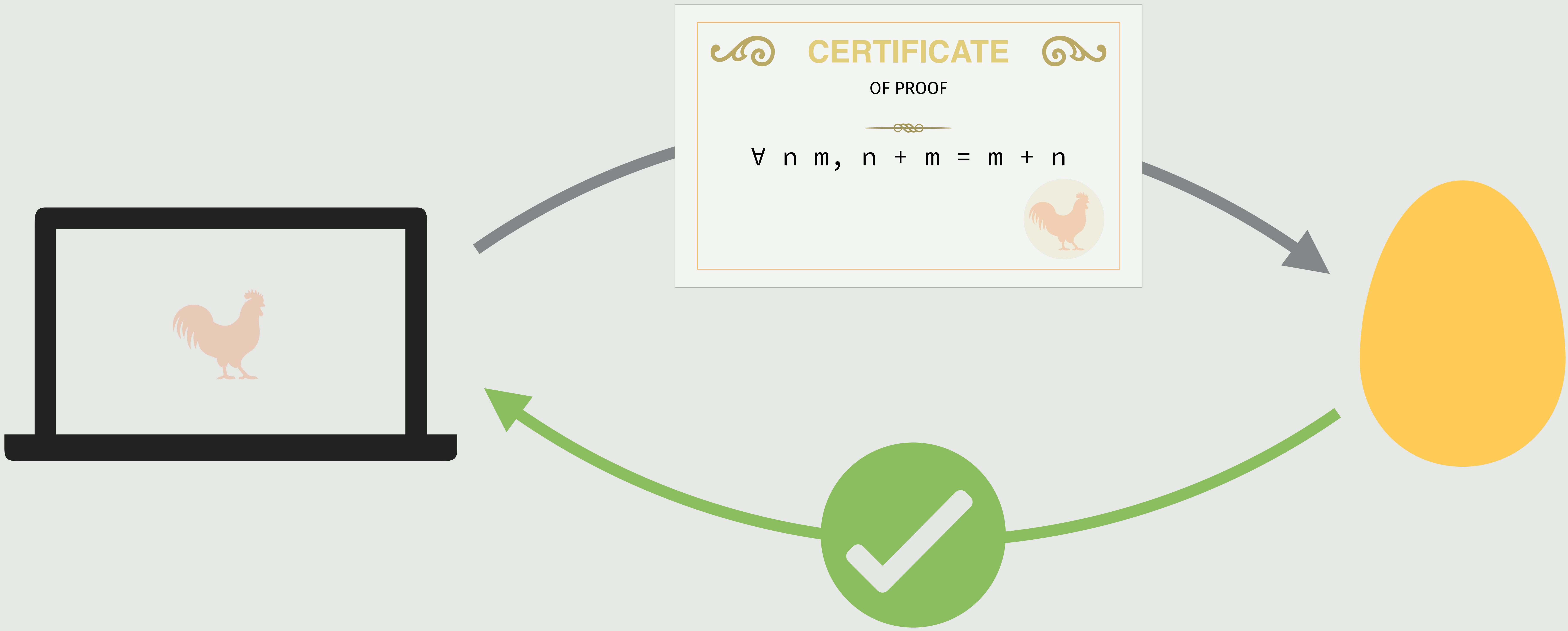
and the MetaCoq team



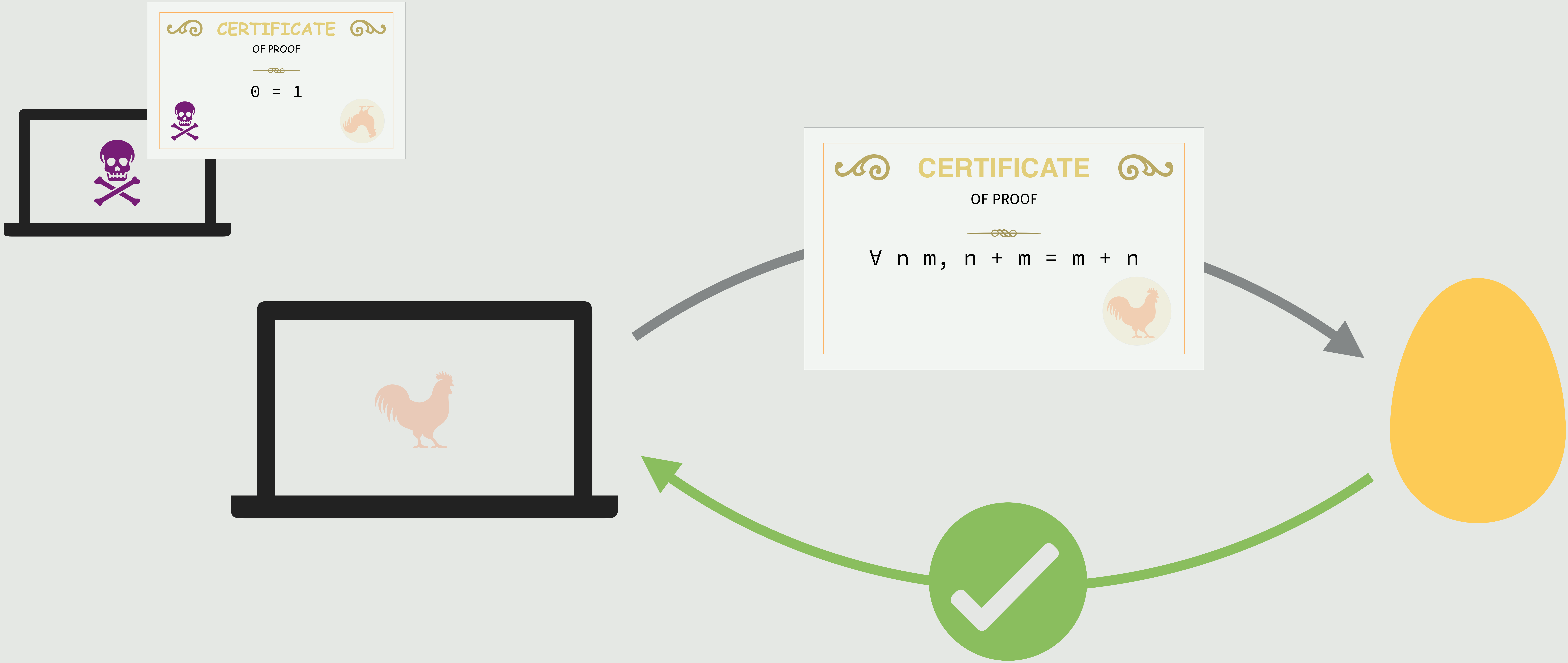

CERTIFICATE
OF PROOF
∀ n m, n + m = m + n

Under the hood

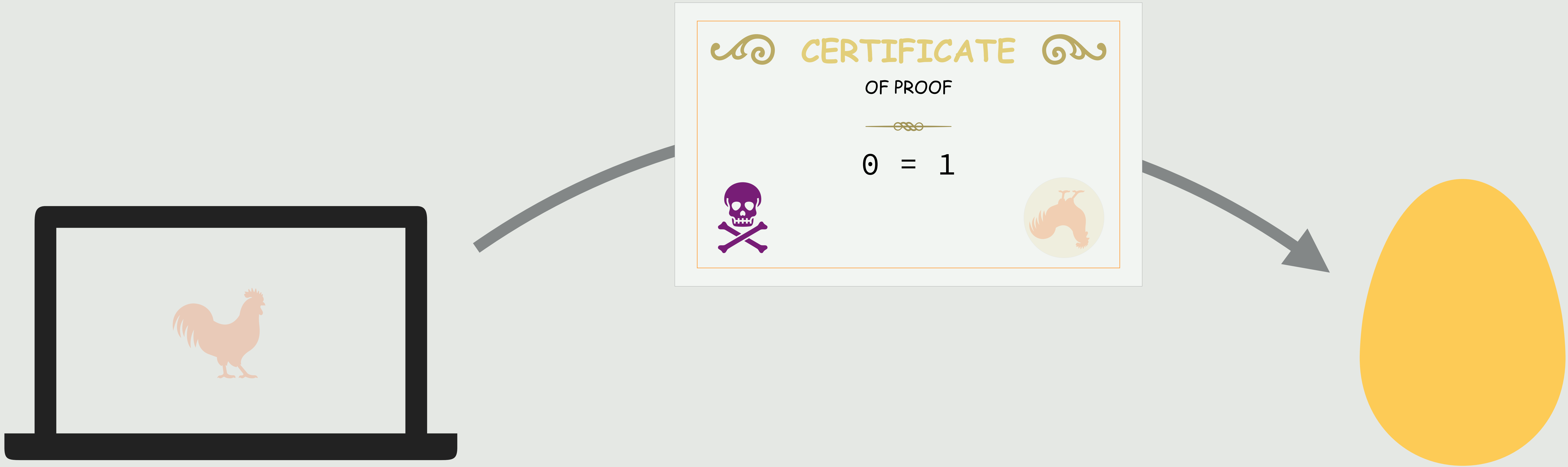
Under the hood



Under the hood



Under the hood

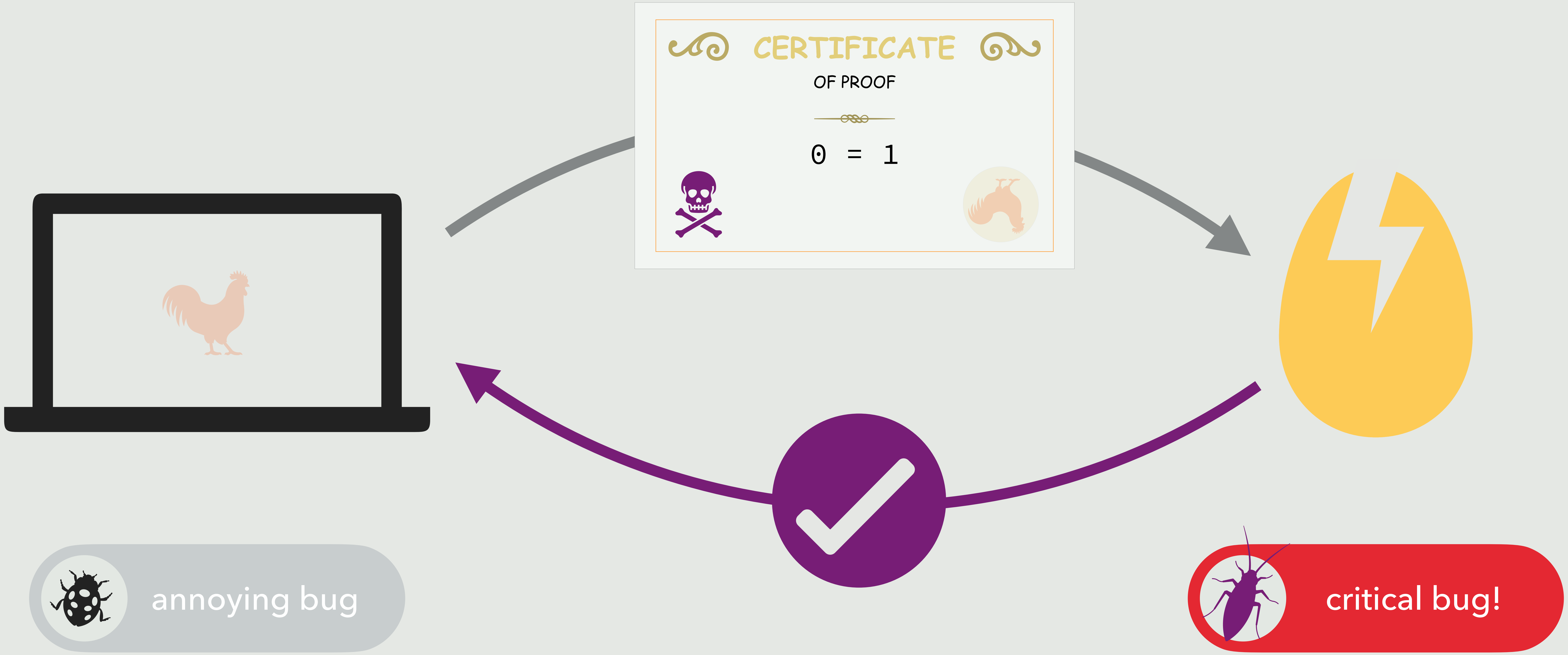


Under the hood

Under the hood

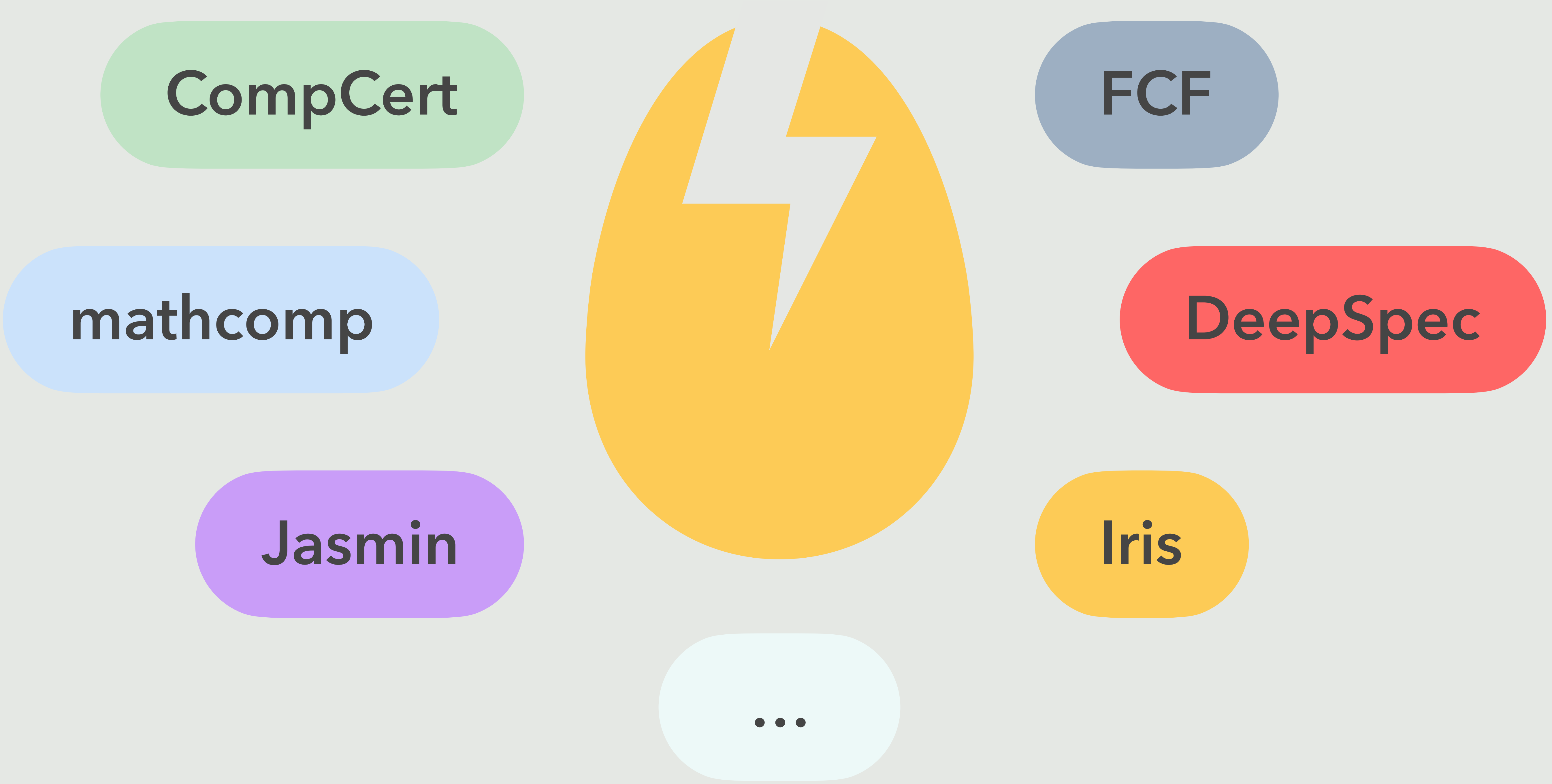
Under the hood

Under the hood

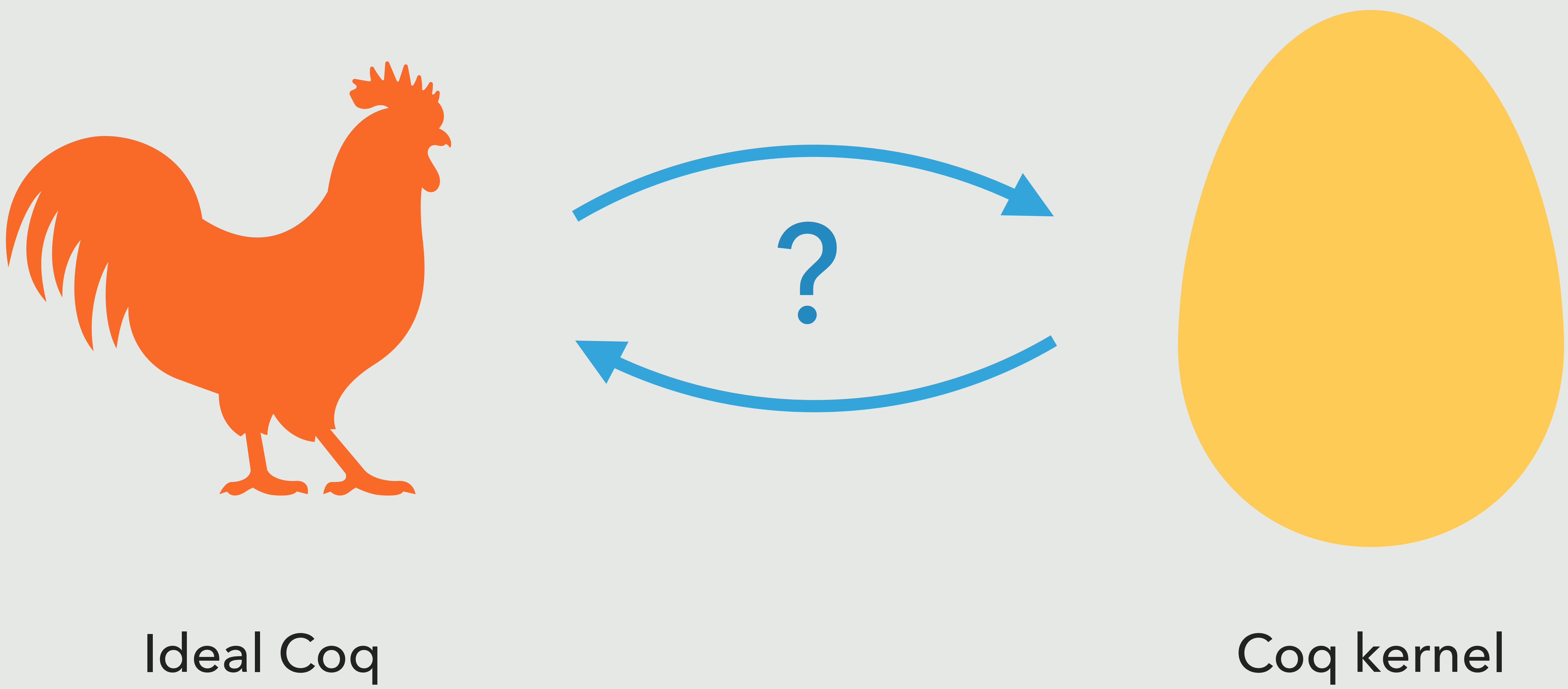


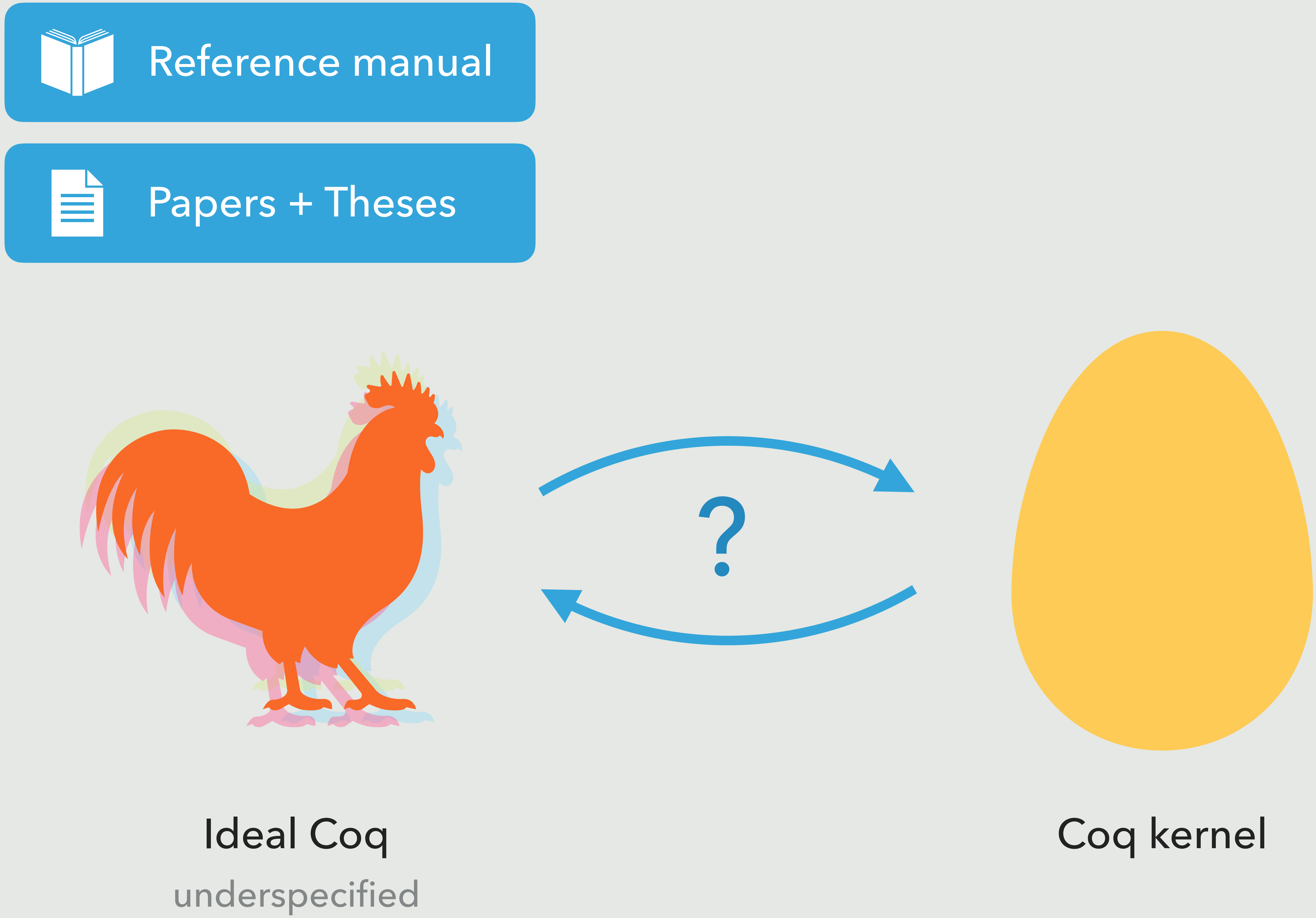
1 **critical bug** every year

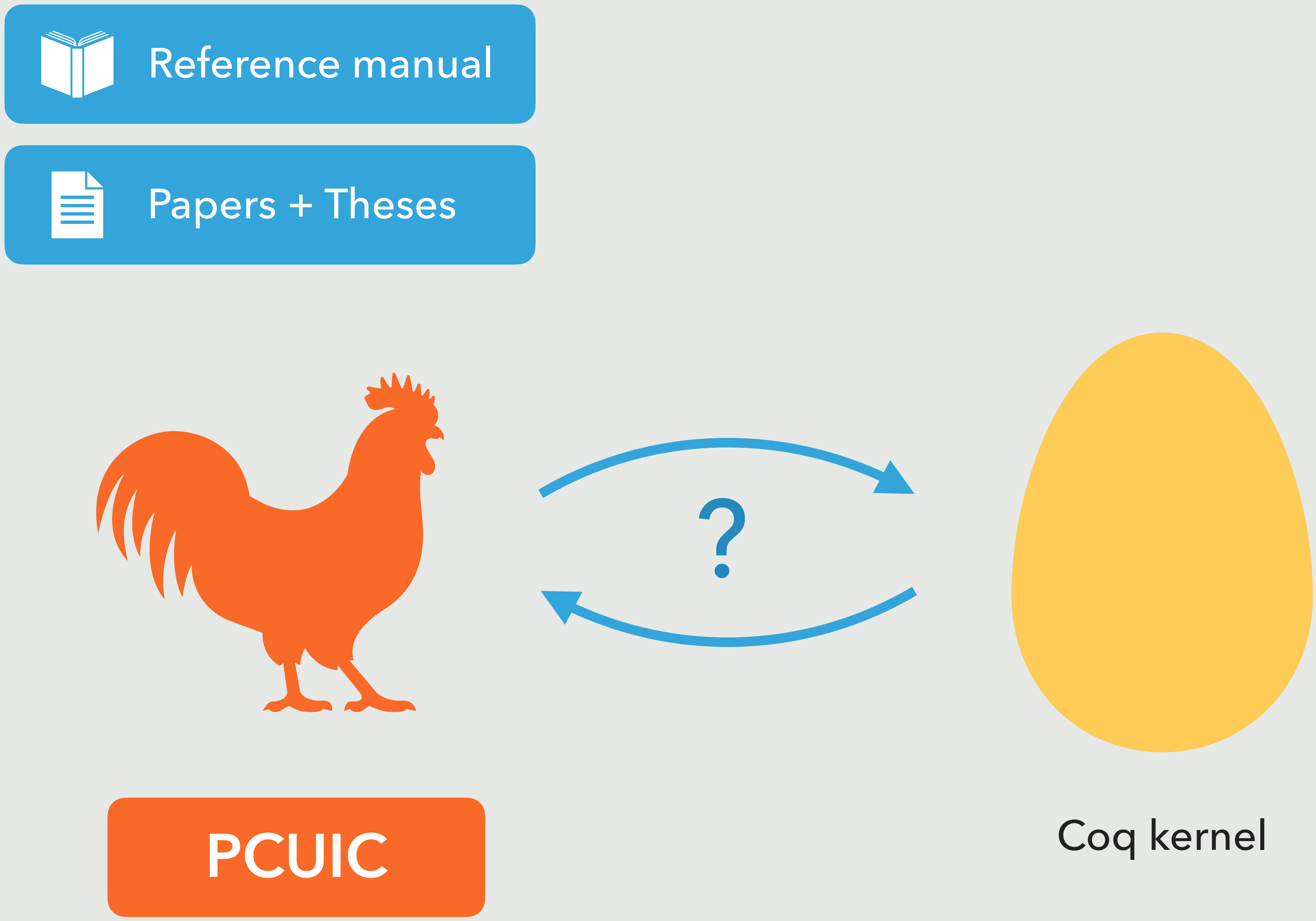
Coq ecosystem



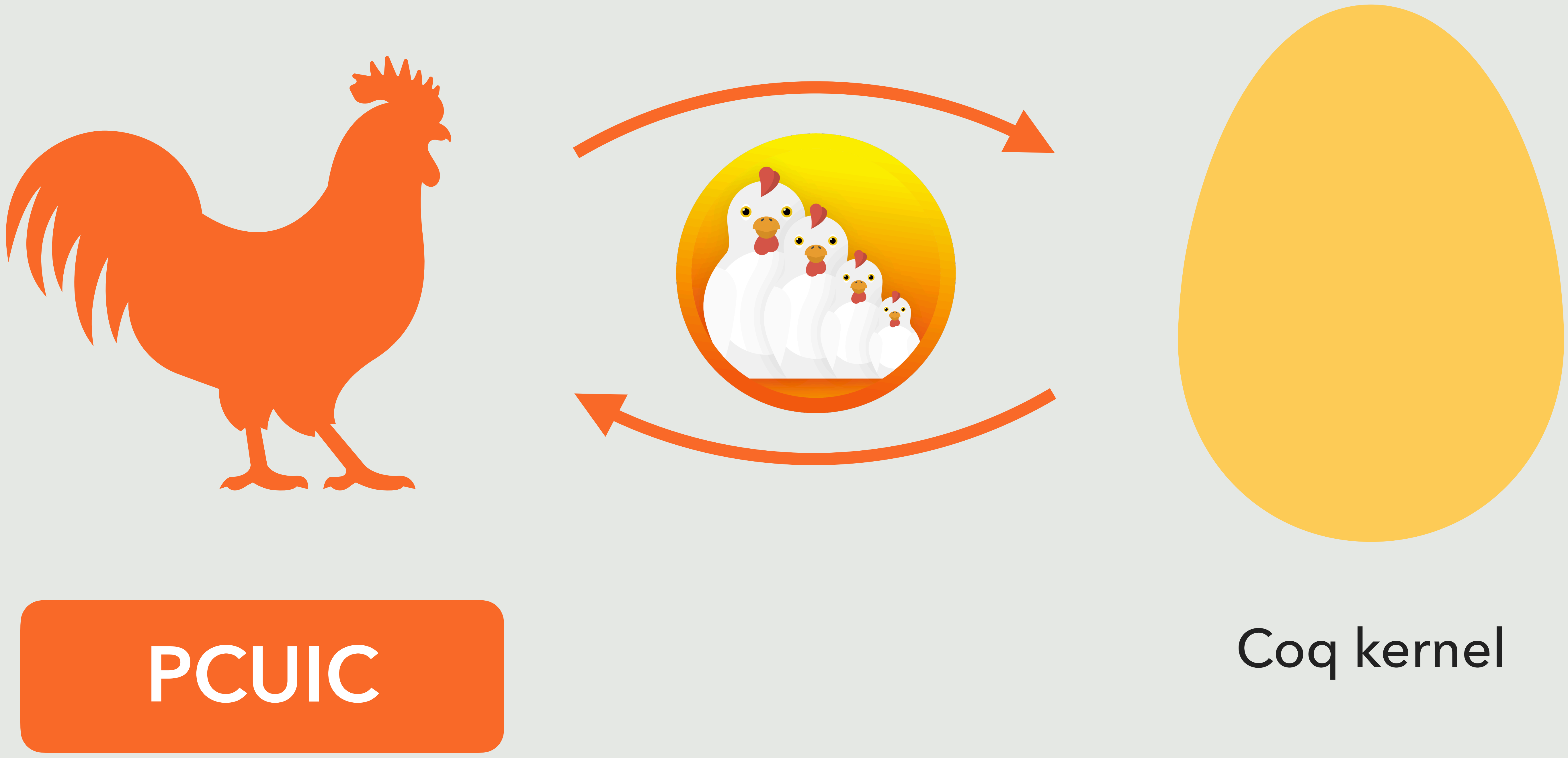
Coq ecosystem


?
Ideal Coq
Coq kernel

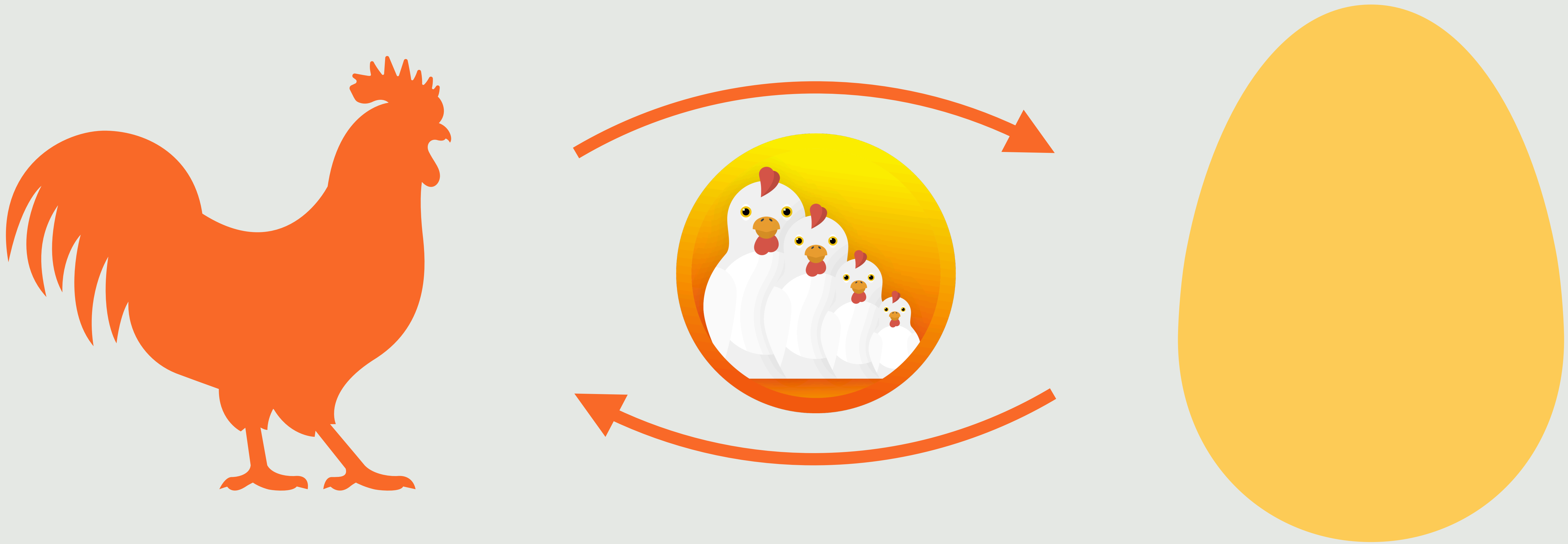

Reference manual
Papers + Theses
?
Ideal Coq
underspecified
Coq kernel


Reference manual
Papers + Theses
?
PCUIC
Coq kernel

MetaCoq



MetaCoq

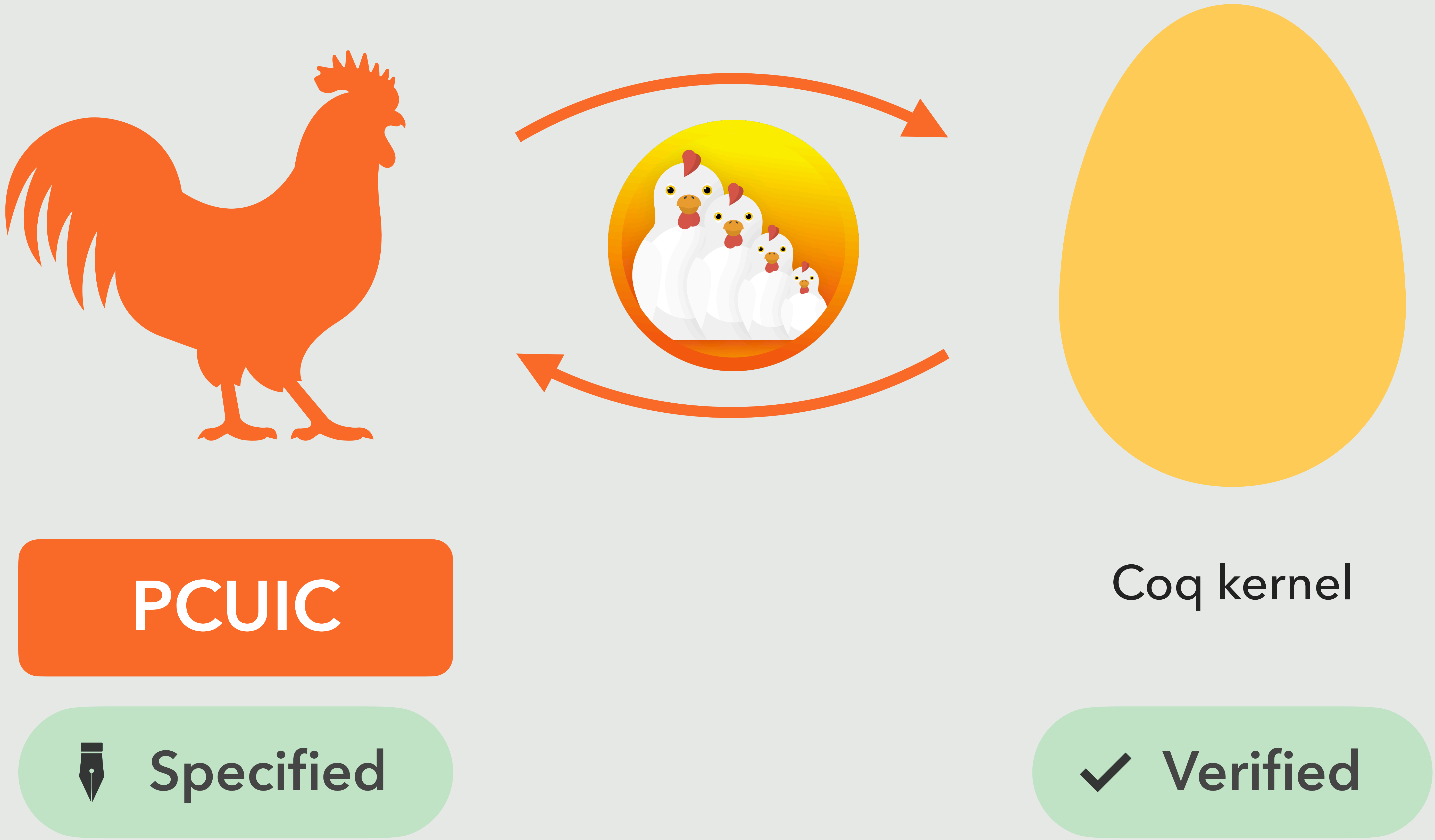


PCUIC

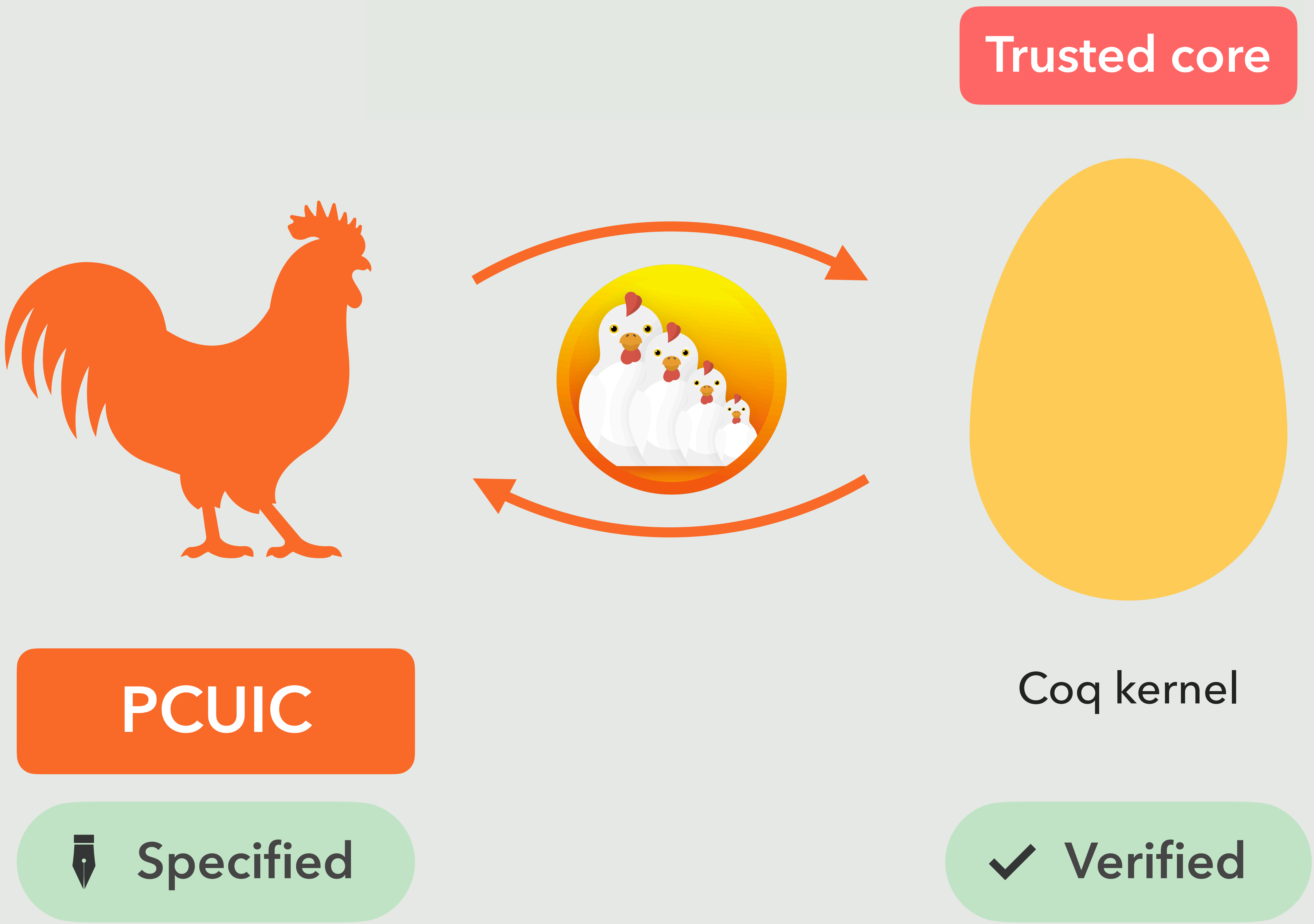
Specified

Coq kernel

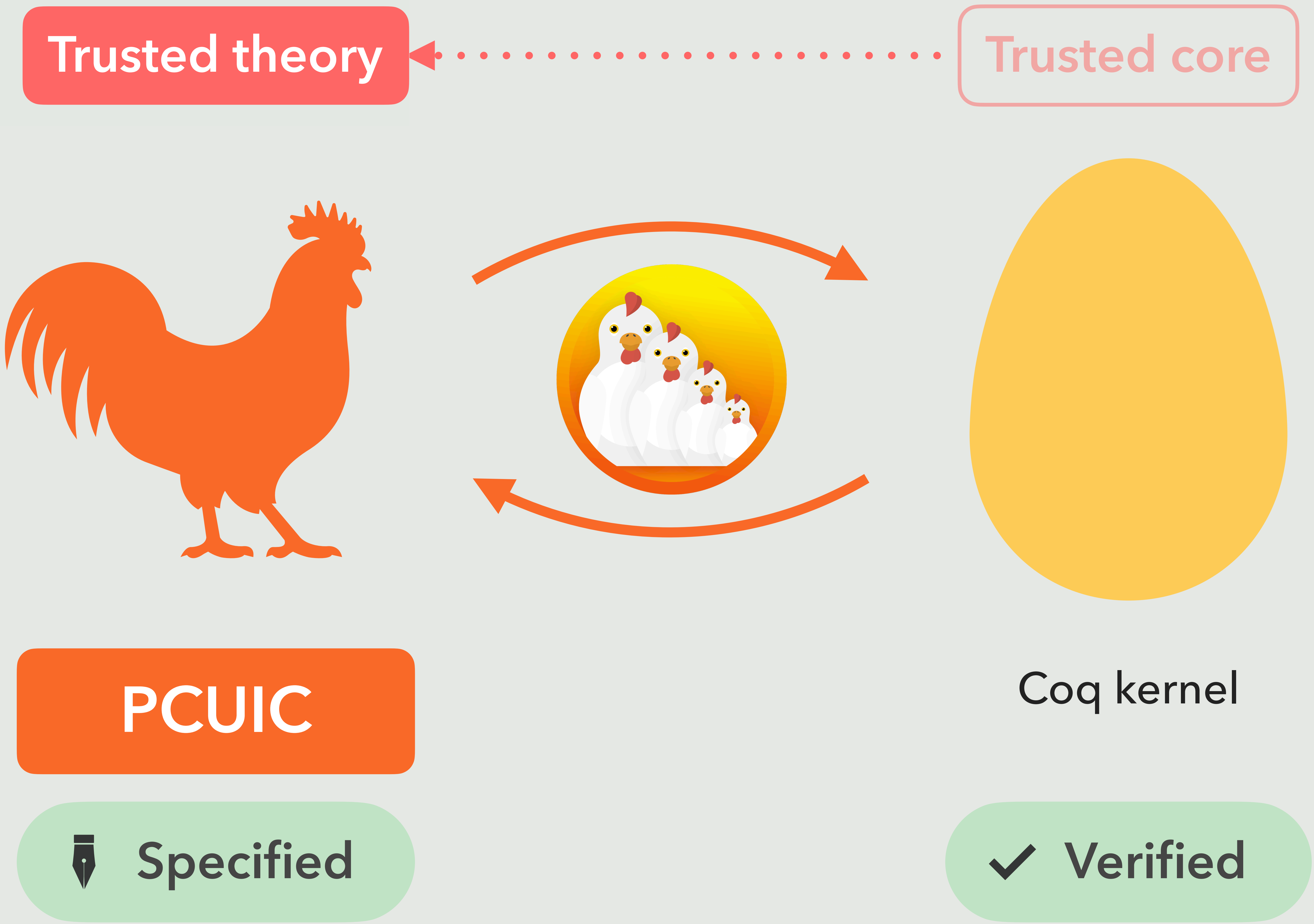
MetaCoq

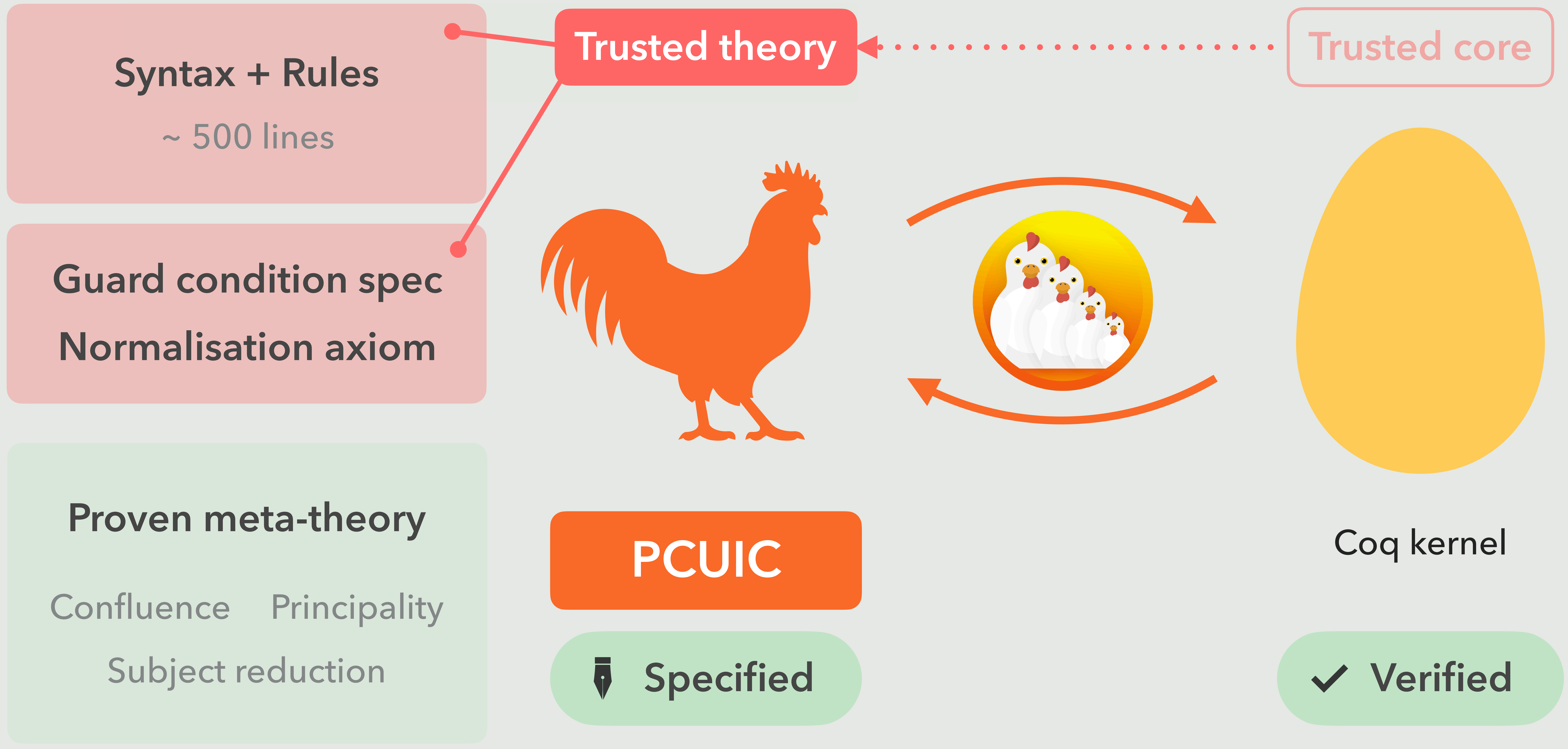


MetaCoq



Shifting trust




Syntax + Rules
~ 500 lines
Guard condition spec
Normalisation axiom
Proven meta-theory
Confluence
Principality
Subject reduction
Trusted theory
Trusted core
PCUIC
Specified
Coq kernel
Verified

Syntax + Rules

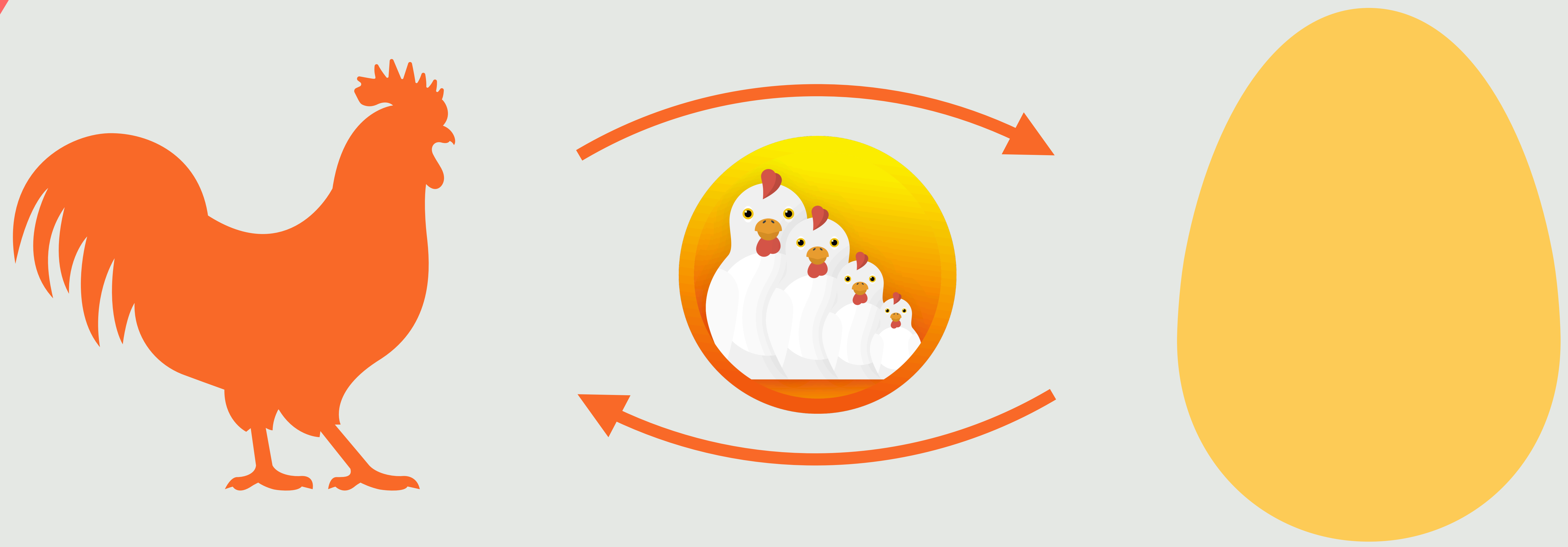
~ 500 lines

Guard condition spec

Normalisation axiom

Trusted theory

Trusted core



Proven meta-theory

Confluence Principality

Subject reduction

PCUIC

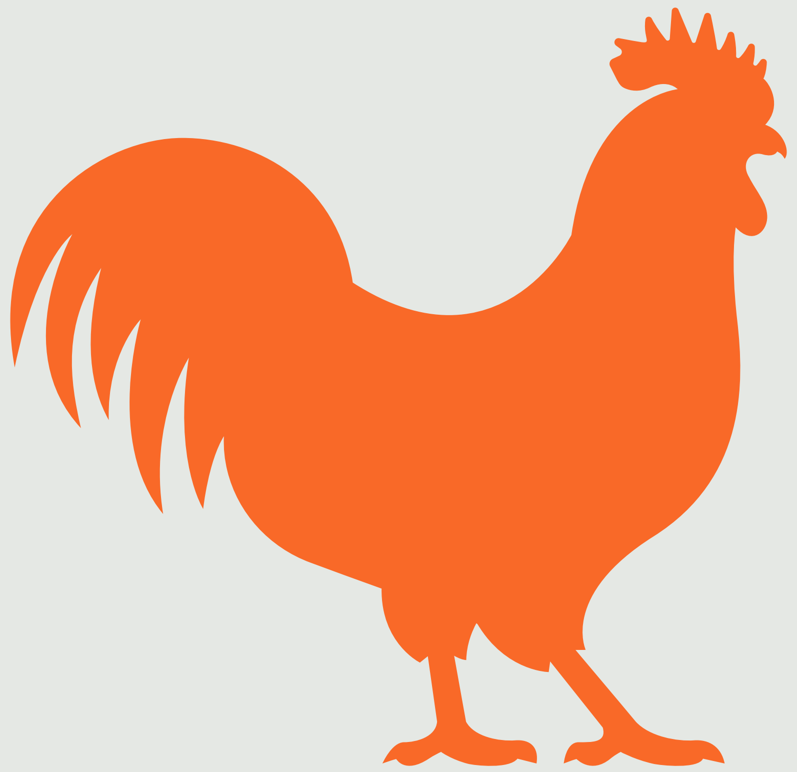
Specified

Coq kernel

✓ Verified

PCUIC

Syntax + Rules



PCUIC

Syntax + Rules

λ-calculus

`λ (x : A), t : ∀ (x : A), B`

PCUIC

Syntax + Rules

λ-calculus

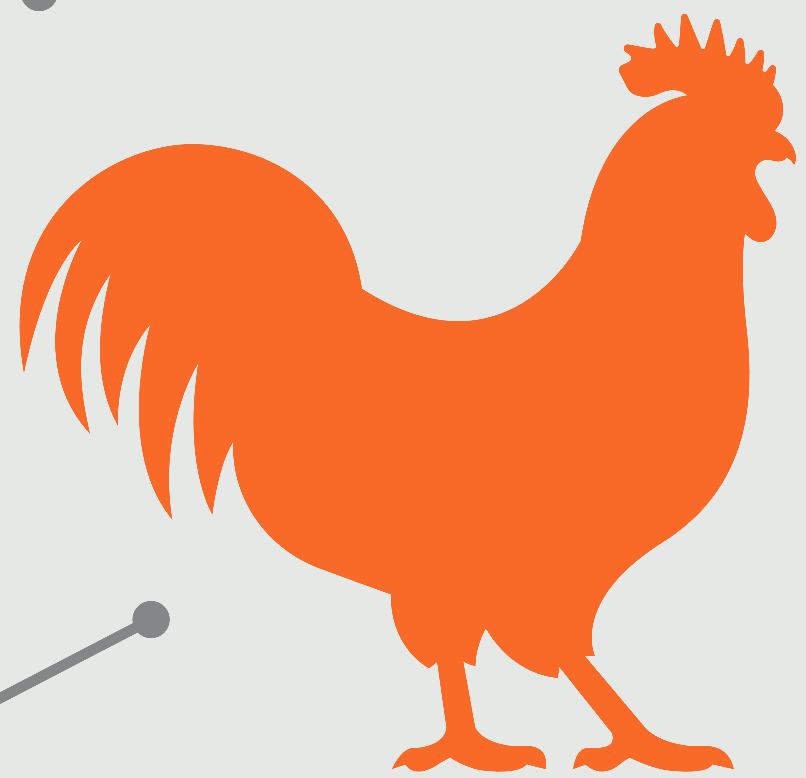
```
λ (x : A), t : ∀ (x : A), B
```

General (co-)inductive types

```
Inductive nat := 0 | S (n : nat).
```

Pattern-matching and (co-)fixed-points

```
match, fix, cofix
```



PCUIC

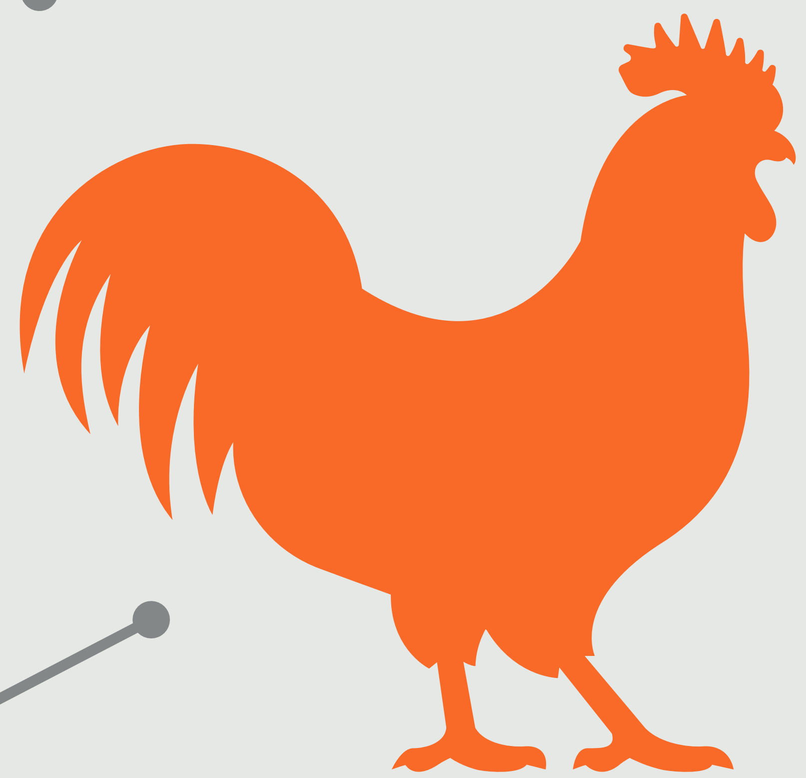
Syntax + Rules

λ-calculus

`λ (x : A), t : ∀ (x : A), B`

Universe polymorphism

`Type@{i}, Set, Prop`



General (co-)inductive types

`Inductive nat := 0 | S (n : nat).`

Pattern-matching and (co-)fixed-points

`match, fix, cofix`

PCUIC

Syntax + Rules

λ-calculus

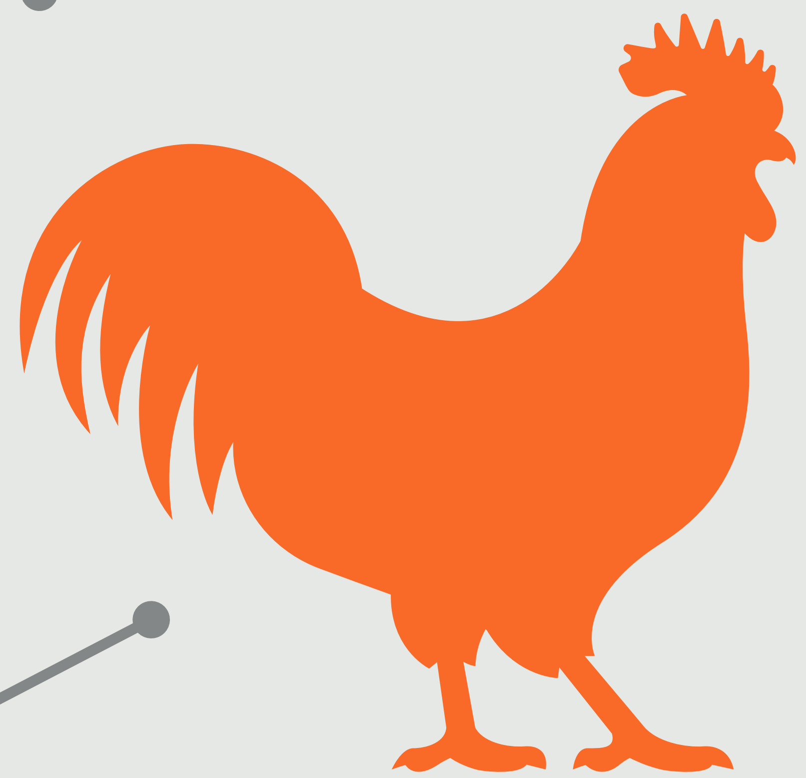
`λ (x : A), t : ∀ (x : A), B`

General (co-)inductive types

`Inductive nat := 0 | S (n : nat).`

Pattern-matching and (co-)fixed-points

`match, fix, cofix`



Universe polymorphism

`Type@{i}, Set, Prop`

Local definitions

`let x := u in t`

PCUIC

Syntax + Rules

λ-calculus

`λ (x : A), t : ∀ (x : A), B`

General (co-)inductive types

`Inductive nat := 0 | S (n : nat).`

Pattern-matching and (co-)fixed-points

`match, fix, cofix`

Universe polymorphism

`Type@{i}, Set, Prop`

Local definitions

`let x := u in t`

Axioms

`Axiom falso : ∀ A, A`

PCUIC

Syntax + Rules

λ-calculus

`λ (x : A), t : ∀ (x : A), B`

Universe polymorphism

`Type@{i}, Set, Prop`

General (co-)inductive types

`Inductive nat := 0 | S (n : nat).`

Local definitions

`let x := u in t`

Pattern-matching and (co-)fixed-points

`match, fix, cofix`

Axioms

`Axiom falso : ∀ A, A`

Template polymorphism

`list nat : Set`
`list Type : Type`

η-equalities

`λ x, f x ≡ f`
`(p.1, p.2) ≡ p`

Modules

`Module`
`Module Type`

Definitional proof irrelevance

`SProp`

PCUIC

Syntax

PCUIC

Syntax

```
Inductive term :=
| tRel (n : nat)
| tSort (u : Universe.t)
| tProd (na : name) (A B : term)
| tLambda (na : name) (A t : term)
| tApp (u v : term)
| tInd (ind : inductive) (ui : Instance.t)
| tConstruct (ind : inductive) (n : nat) (ui : Instance.t)
| (* … *)
```

```
Inductive term :=
| tRel (n : nat)
| tSort (u : Universe.t)
| tProd (na : name) (A B : term)
| tLambda (na : name) (A t : term)
| tApp (u v : term)
| tInd (ind : inductive) (ui : Instance.t)
| tConstruct (ind : inductive) (n : nat) (ui : Instance.t)
| (* … *)
```

```
λ(P : Prop). P
```

becomes

```
tLambda (nNamed "P") (tSort prop) (tRel 0)
```

PCUIC

Judgments

Cumulativity $\Sigma \; ; \; \Gamma \vdash u \leq v$

Typing $\Sigma \; ; \; \Gamma \vdash t : A$

PCUIC

Judgments

Cumulativity $\Sigma \; ; \; \Gamma \vdash u \leq v$

Typing $\Sigma \; ; \; \Gamma \vdash t : A$

Global environment

axioms

definitions

inductive types

universes

PCUIC

Judgments

Cumulativity $\Sigma \; ; \; \Gamma \vdash u \leq v$

Typing $\Sigma \; ; \; \Gamma \vdash t : A$

Global environment

axioms definitions

inductive types universes

Local environment

assumptions $x : A$

definitions $x := u$

PCUIC

Σ ; Γ ⊢ u ≤pb v

```
Inductive cumulSpec0 Σ Γ pb : term → term → Type :=
| cumul_Sort : ∀ s s',
    compare_universe pb Σ s s' →
    Σ ; Γ ⊢ tSort s ≤pb tSort s’
| cumul_beta : ∀ na t b a,
    Σ ; Γ ⊢ tApp (tLambda na t b) a ≤pb b {0 := a}
| cumul_App : ∀ t t' u u’,
    Σ ; Γ ⊢ t ≤pb t’ →
    Σ ; Γ ⊢ u ≤pb u’ →
    Σ ; Γ ⊢ tApp t u ≤pb tApp t’ u’
| (* … *)
```

PCUIC

$\Sigma \; ; \; \Gamma \vdash u \leq_{pb} v$

```
cumul_Sym : ∀ t u,
  Σ ; Γ ⊢ u ≤Conv t →
  Σ ; Γ ⊢ t ≤pb u
```

PCUIC

$\Sigma\ ;\ \Gamma \vdash u \leq_{pb} v$

```
cumul_Sym : ∀ t u,
  Σ ; Γ ⊢ u ≤Conv t →
  Σ ; Γ ⊢ t ≤pb u
```

```
Σ ; Γ ⊢ u ≤ v := Σ ; Γ ⊢ u ≤Cumul v
Σ ; Γ ⊢ u = v := Σ ; Γ ⊢ u ≤Conv v
```

PCUIC

$\Sigma \; ; \; \Gamma \vdash u \leq_{pb} v$

```
cumul_Sym : ∀ t u,
  Σ ; Γ ⊢ u ≤Conv t →
  Σ ; Γ ⊢ t ≤pb u
```

```
Σ ; Γ ⊢ u ≤ v := Σ ; Γ ⊢ u ≤Cumul v
Σ ; Γ ⊢ u = v := Σ ; Γ ⊢ u ≤Conv v
```

```
compare_universe Cumul Σ s s’
```

all valuations satisfying the constraints of Σ verify s ≤ s’

PCUIC

$\Sigma ; \Gamma \vdash u \leq_{pb} v$

```
cumul_Sym : ∀ t u,
  Σ ; Γ ⊢ u ≤Conv t →
  Σ ; Γ ⊢ t ≤pb u
```

```
Σ ; Γ ⊢ u ≤ v := Σ ; Γ ⊢ u ≤Cumul v
Σ ; Γ ⊢ u = v := Σ ; Γ ⊢ u ≤Conv v
```

```
compare_universe Cumul Σ s s’
```

all valuations satisfying the constraints of Σ verify s ≤ s’

⚠ equality / cumulativity is untyped

$\Sigma ; \Gamma \vdash t : A$

$\Sigma ; \Gamma \vdash u \leq_{pb} v$

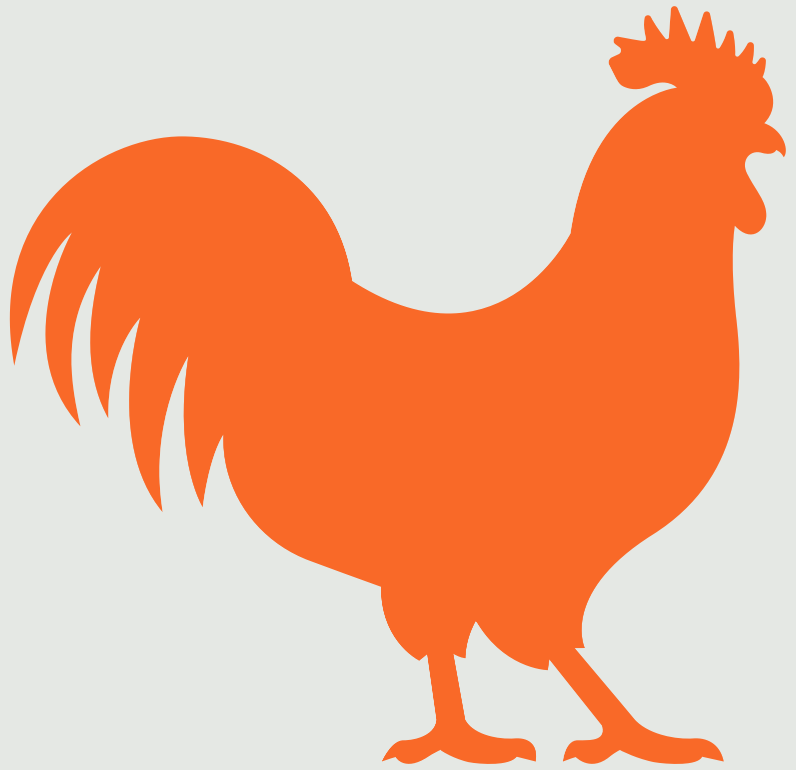
Syntax + Rules

~ 500 lines

Guard condition spec

Normalisation axiom

Trusted theory



Proven meta-theory

Confluence Principality

Subject reduction

PCUIC

Specified

$\Sigma \; ; \; \Gamma \vdash t : A$

$\Sigma \; ; \; \Gamma \vdash u \leq_{pb} v$

Syntax + Rules

~ 500 lines

Guard condition spec

Normalisation axiom

Trusted theory

Proven meta-theory

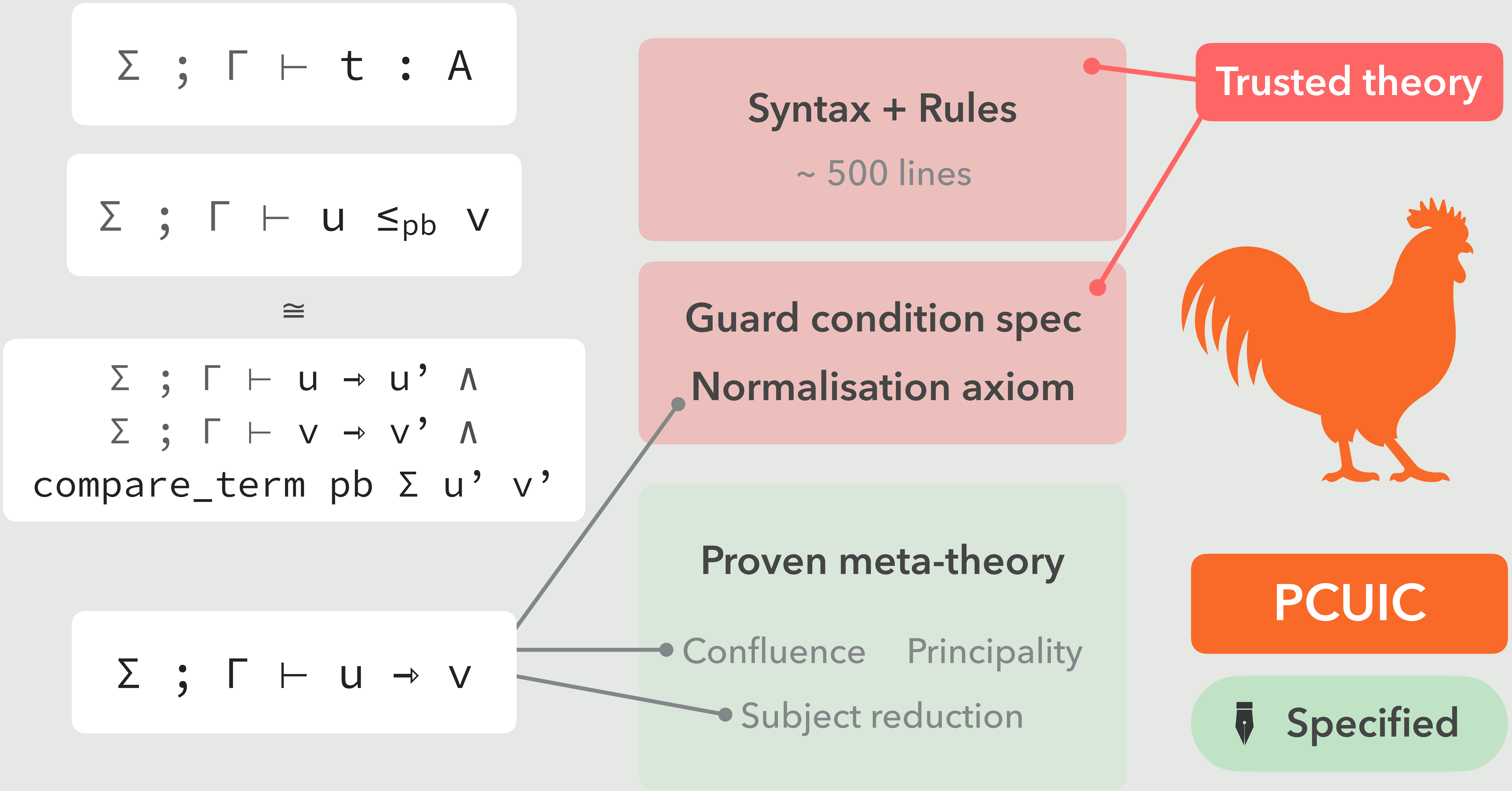
Confluence Principality

Subject reduction

$\Sigma \; ; \; \Gamma \vdash u \rightarrow v$

PCUIC

Specified


Σ ; Γ ⊢ t : A
Σ ; Γ ⊢ u ≤pb v
≅
Σ ; Γ ⊢ u → u' ∧
Σ ; Γ ⊢ v → v' ∧
compare_term pb Σ u' v'
Σ ; Γ ⊢ u → v
Syntax + Rules
~ 500 lines
Guard condition spec
Normalisation axiom
Proven meta-theory
Confluence
Principality
Subject reduction
Trusted theory
PCUIC
Specified

$\Sigma \; ; \; \Gamma \vdash t : A$

$\Sigma \; ; \; \Gamma \vdash u \leq_{pb} v$

$\cong$

$\Sigma \; ; \; \Gamma \vdash u \to u' \;\wedge$
$\Sigma \; ; \; \Gamma \vdash v \to v' \;\wedge$
compare_term pb Σ u' v'

$\Sigma \; ; \; \Gamma \vdash u \to v$

Syntax + Rules

~ 500 lines

Guard condition spec

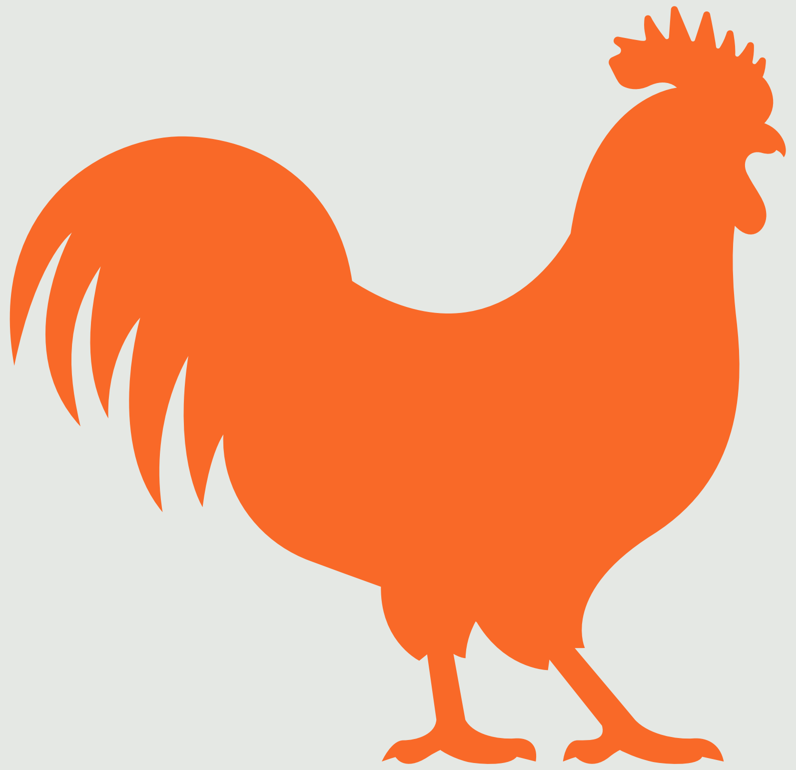
Normalisation axiom

Proven meta-theory

Confluence Principality

Subject reduction

Trusted theory



PCUIC

Specified

PCUIC

Meta-theory

Subject Reduction

If $\Sigma ; \Gamma \vdash u \rightarrow v$ and $\Sigma ; \Gamma \vdash u : A$

then $\Sigma ; \Gamma \vdash v : A$

PCUIC

Meta-theory

Weakening / Substitution …

Validity … Confluence

Subject Reduction

If $\Sigma ; \Gamma \vdash u \to v$ and $\Sigma ; \Gamma \vdash u : A$

then $\Sigma ; \Gamma \vdash v : A$

PCUIC

Meta-theory

Weakening / Substitution

…

Validity

Confluence

Subject Reduction

If $\Sigma \; ; \; \Gamma \vdash u \rightarrow v$ and $\Sigma \; ; \; \Gamma \vdash u : A$

then $\Sigma \; ; \; \Gamma \vdash v : A$

Parellel reduction

Following Tait, Martin-Löf and Takahashi

$$\rightarrow \;\subset\; \Rrightarrow \;\subset\; \rightarrow^*$$

Can reduce all immediate reducts in one step

Parellel reduction

Following Tait, Martin-Löf and Takahashi

$$\rightarrow \;\subset\; \Rrightarrow \;\subset\; \rightarrow^*$$

Can reduce all immediate reducts in one step

```
(S a) + ((λx. x + b) 0) ⇛ S (a + (0 + b))
```

Parellel reduction

Following Tait, Martin-Löf and Takahashi

$$\to \;\subset\; \Rrightarrow \;\subset\; \to^*$$

Can reduce all immediate reducts in one step

```
(S a) + ((λx. x + b) 0) ⇛ S (a + (0 + b))
```

but also

```
(S a) + ((λx. x + b) 0) ⇛ (S a) + (0 + b)
```

The triangle property

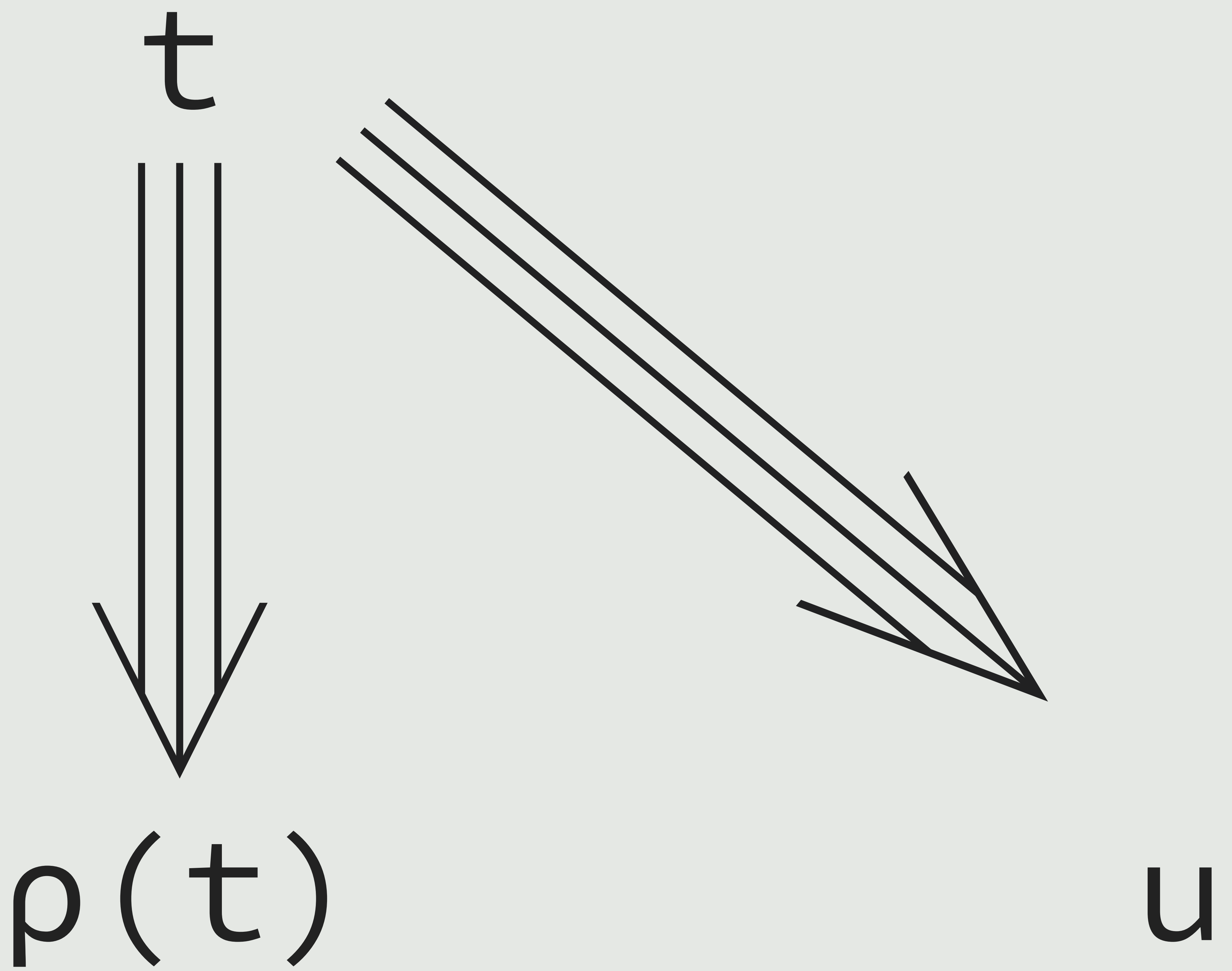
Optimal one-step parallel reduction

$$t$$

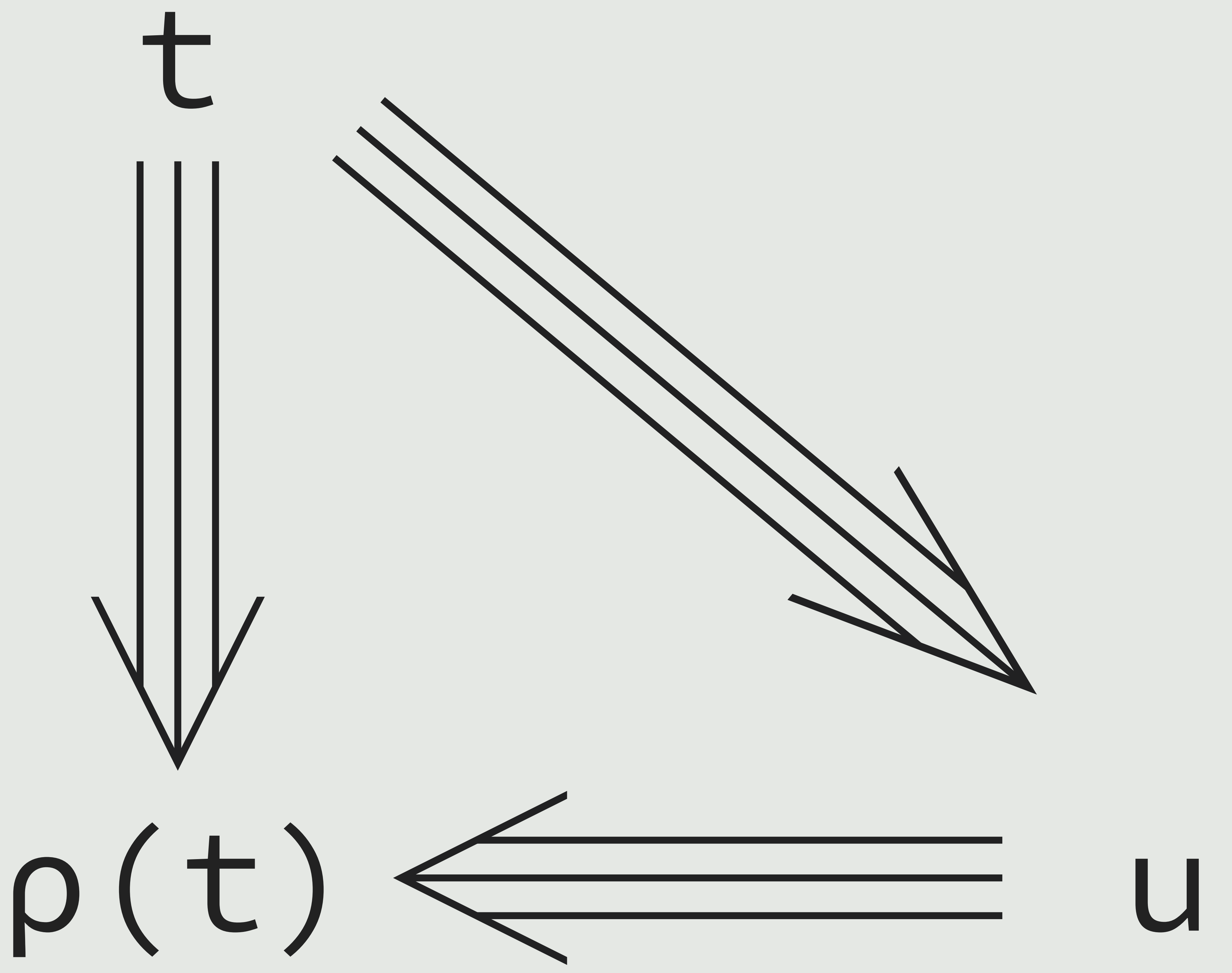
$$\Downarrow$$

$$\rho(t)$$

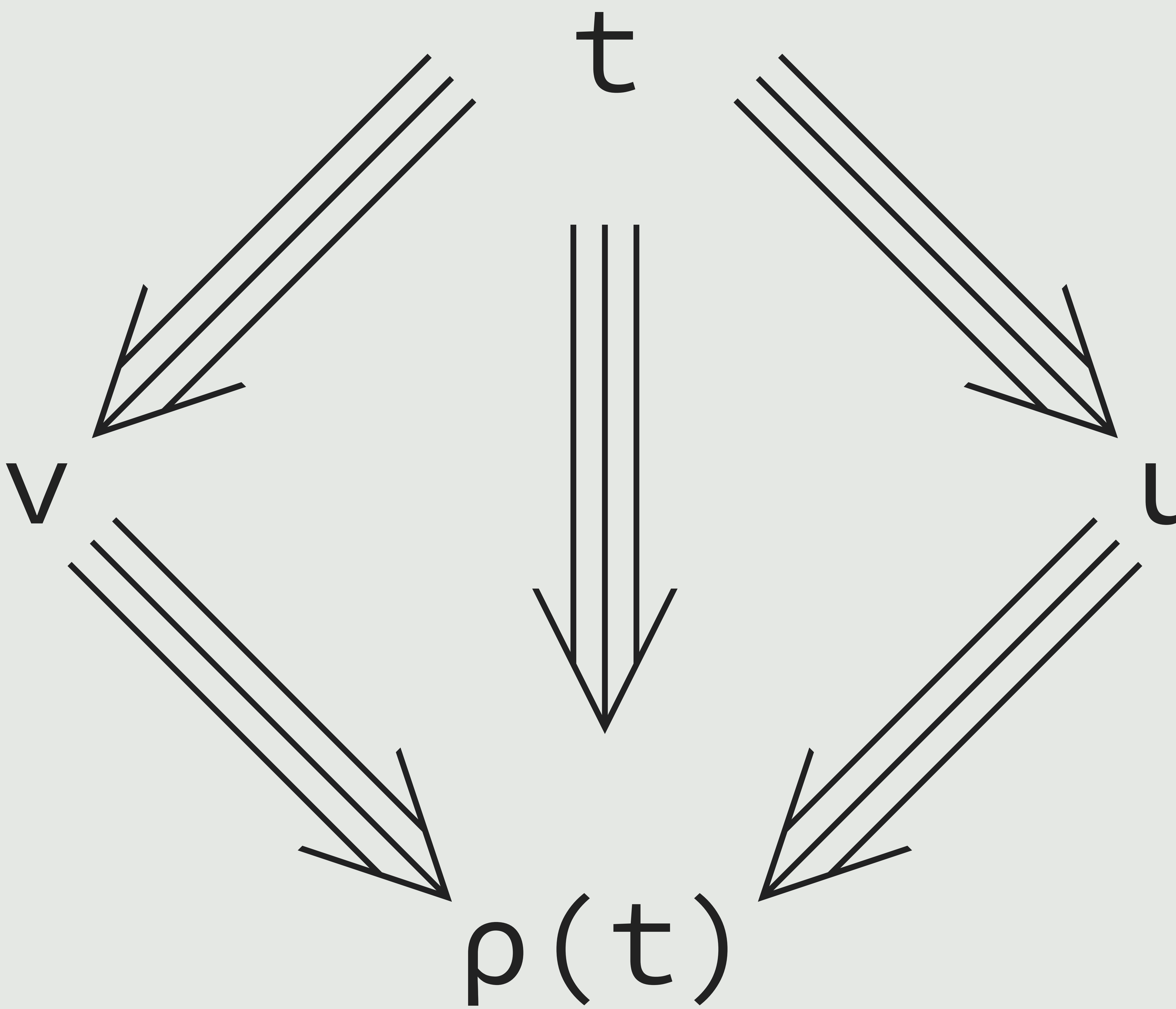
The triangle property



The triangle property

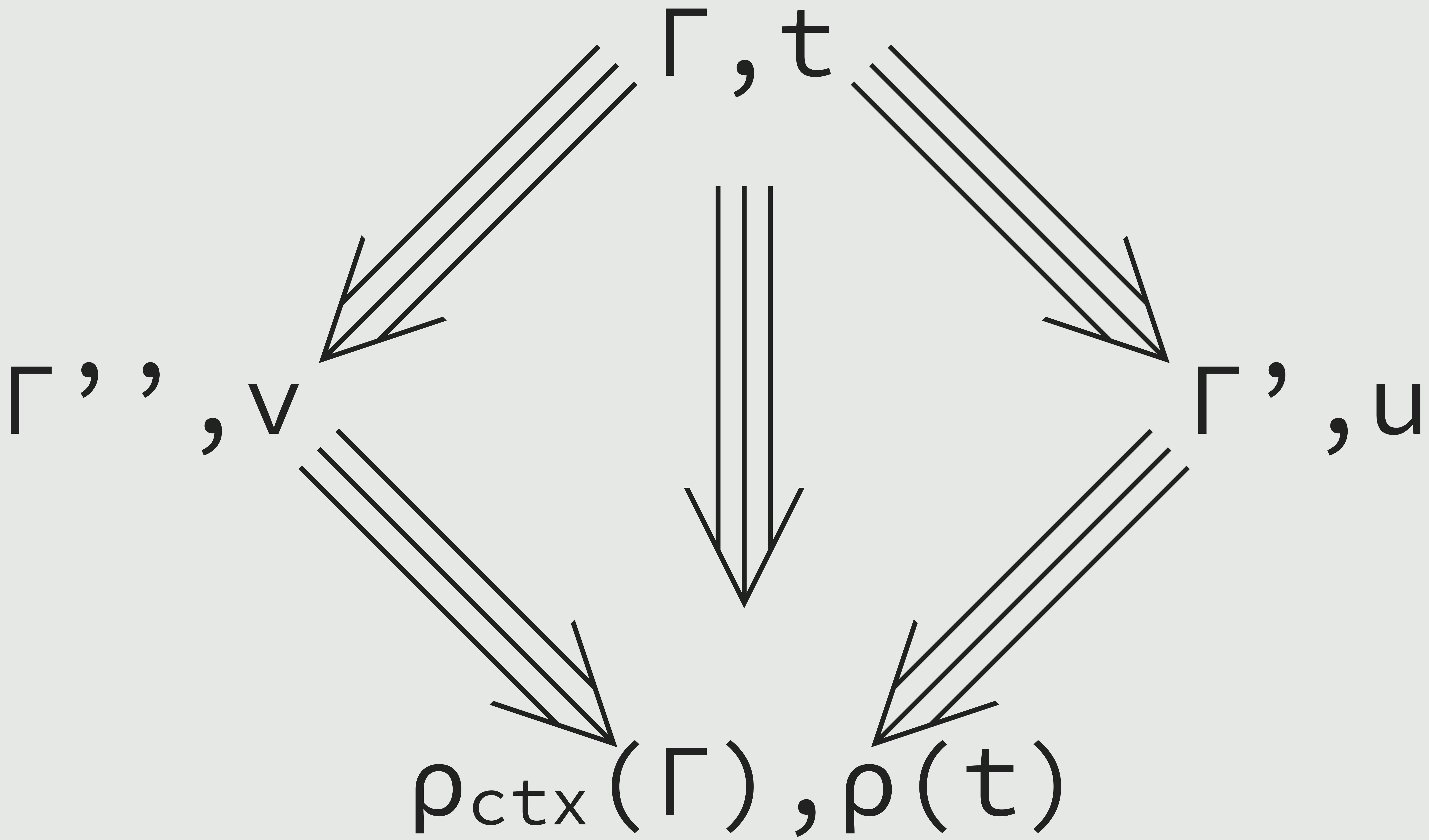


The triangle property



The triangle property

Accounting for local definitions



Syntax + Rules

~ 500 lines

Guard condition spec

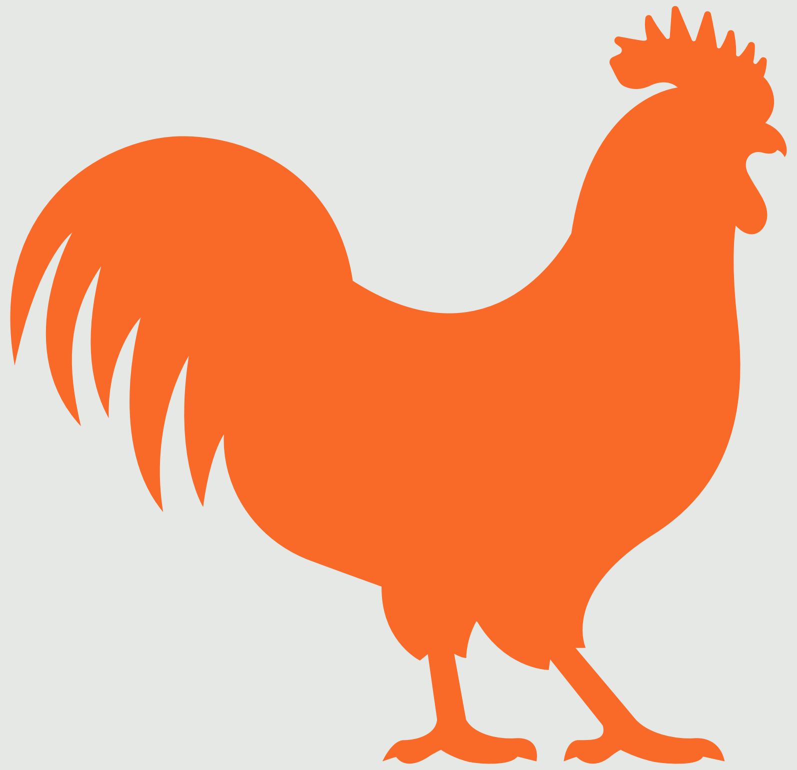
Normalisation axiom

Trusted theory

Proven meta-theory

Confluence Principality

Subject reduction



PCUIC

Specified

oracles such that

$$\Sigma \; ; \; \Gamma \vdash \_ \leftarrow \_$$

is well-founded
for well-typed terms

Syntax + Rules

~ 500 lines

Guard condition spec

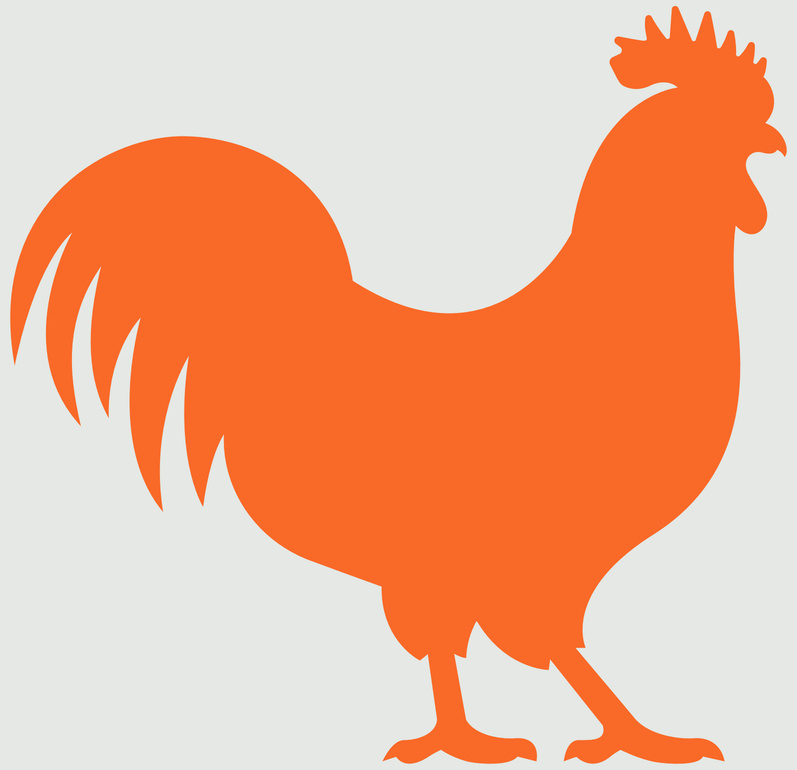
Normalisation axiom

Trusted theory

Proven meta-theory

Confluence Principality

Subject reduction



PCUIC

Specified

oracles such that

$$\Sigma \,;\, \Gamma \vdash \_ \leftarrow \_$$

is well-founded
for well-typed terms

```
Axiom normalisation :
  ∀ Σ Γ t,
    wf_ext Σ →
    welltyped Σ Γ t →
    Acc (cored Σ Γ) t.
```

Syntax + Rules

~ 500 lines

Guard condition spec

Normalisation axiom

Trusted theory

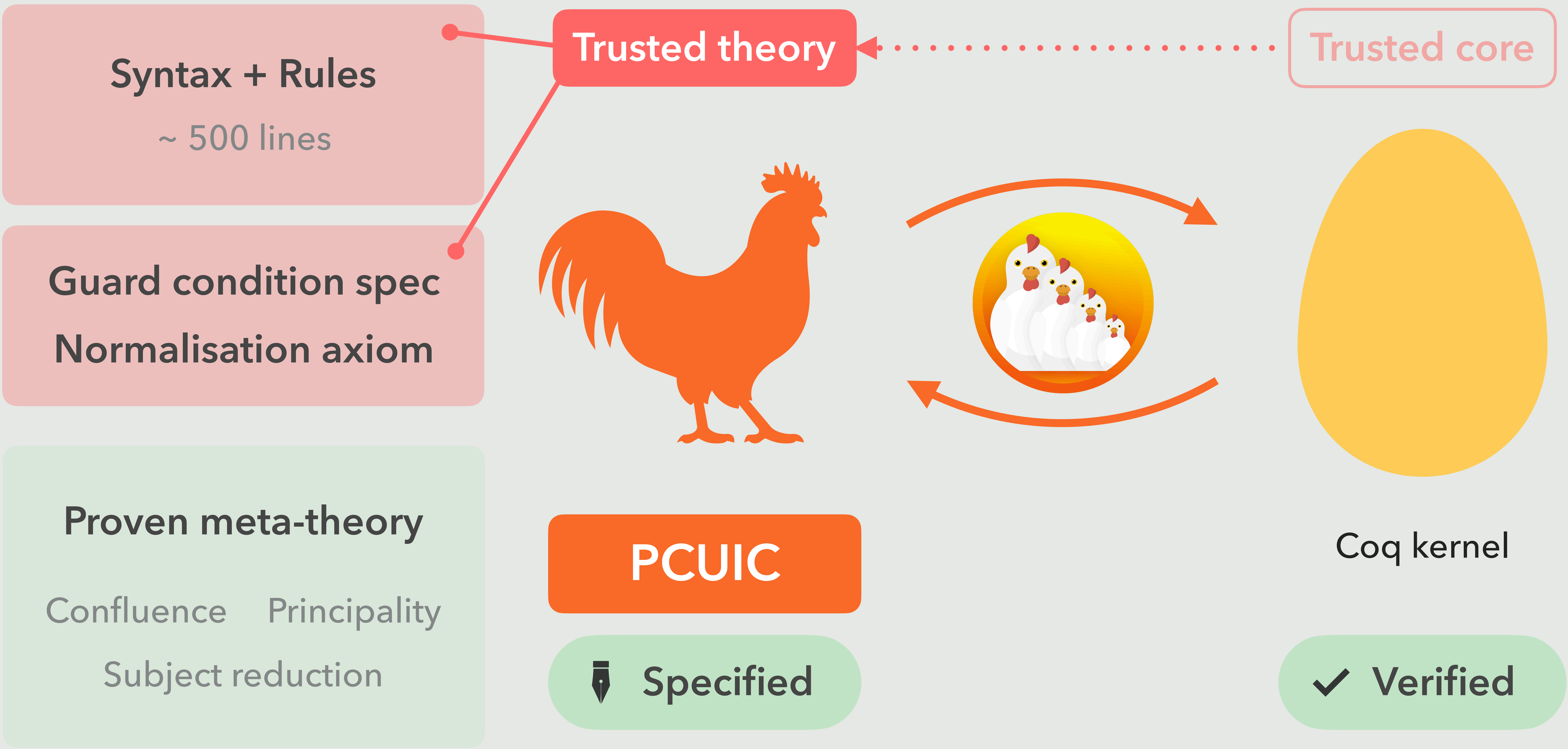
Proven meta-theory

Confluence Principality

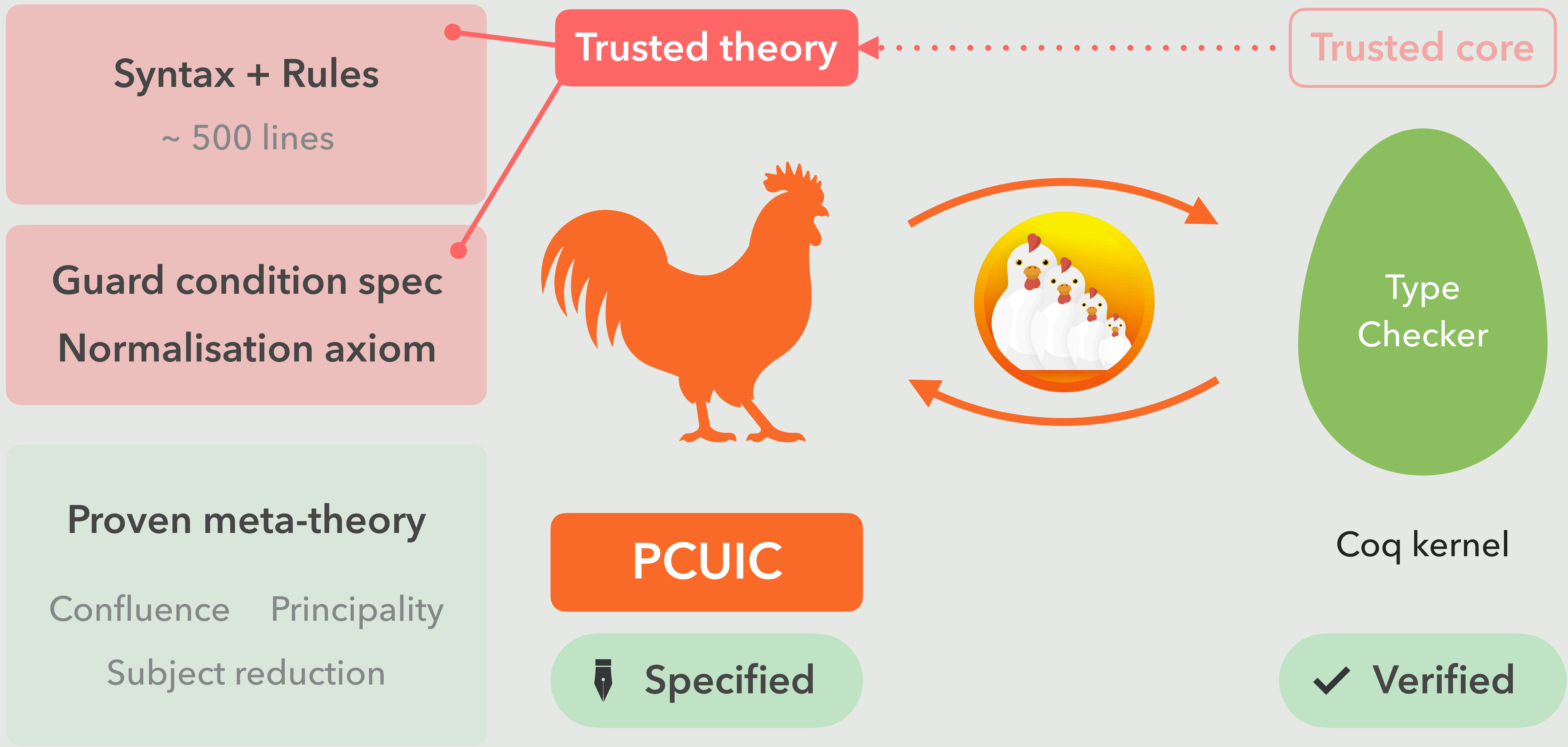
Subject reduction

PCUIC

Specified



Syntax + Rules
~ 500 lines
Guard condition spec
Normalisation axiom
Trusted theory
Trusted core
Proven meta-theory
Confluence
Principality
Subject reduction
PCUIC
Specified
Coq kernel
Verified



Syntax + Rules
~ 500 lines
Guard condition spec
Normalisation axiom
Proven meta-theory
Confluence
Principality
Subject reduction
Trusted theory
Trusted core
PCUIC
Specified
Type Checker
Coq kernel
Verified

```
check : ∀ Σ Γ t A, dec ‖ Σ ; Γ ⊢ t : A ‖
```

Verified
Type Checker

```
dec A := A + ¬ A
```

Inference

```
infer : ∀ Σ Γ t, dec (∑ A, ‖ Σ ; Γ ⊢ t : A ‖)
```

```
check : ∀ Σ Γ t A, dec ‖ Σ ; Γ ⊢ t : A ‖
```

Verified
Type Checker

```
dec A := A + ¬ A
```

Cumulativity checking

```
iscumul :
  ∀ pb Σ Γ u v,
    welltyped Σ Γ u →
    welltyped Σ Γ v →
    dec ∥ Σ ; Γ ⊢ u ≤pb v ∥
```

Inference

```
infer : ∀ Σ Γ t, dec (∑ A, ∥ Σ ; Γ ⊢ t : A ∥)
```

```
check : ∀ Σ Γ t A, dec ∥ Σ ; Γ ⊢ t : A ∥
```

Verified
Type Checker

```
dec A := A + ¬ A
```

Cumulativity checking

```
iscumul :
  ∀ pb Σ Γ u v,
    welltyped Σ Γ u →
    welltyped Σ Γ v →
    dec ∥ Σ ; Γ ⊢ u ≤pb v ∥
```

Inference

```
infer : ∀ Σ Γ t, dec (∑ A, ∥ Σ ; Γ ⊢ t : A ∥)
```

```
check : ∀ Σ Γ t A, dec ∥ Σ ; Γ ⊢ t : A ∥
```

```
Check t : A
```

Cumulativity checking

```
iscumul :
  ∀ pb Σ Γ u v,
    welltyped Σ Γ u →
    welltyped Σ Γ v →
    dec ∥ Σ ; Γ ⊢ u ≤pb v ∥
```

Inference

```
infer : ∀ Σ Γ t, dec (∑ A, ∥ Σ ; Γ ⊢ t : A ∥)
```

```
check : ∀ Σ Γ t A, dec ∥ Σ ; Γ ⊢ t : A ∥
```

```
Infer t
```

```
Check t : A
```

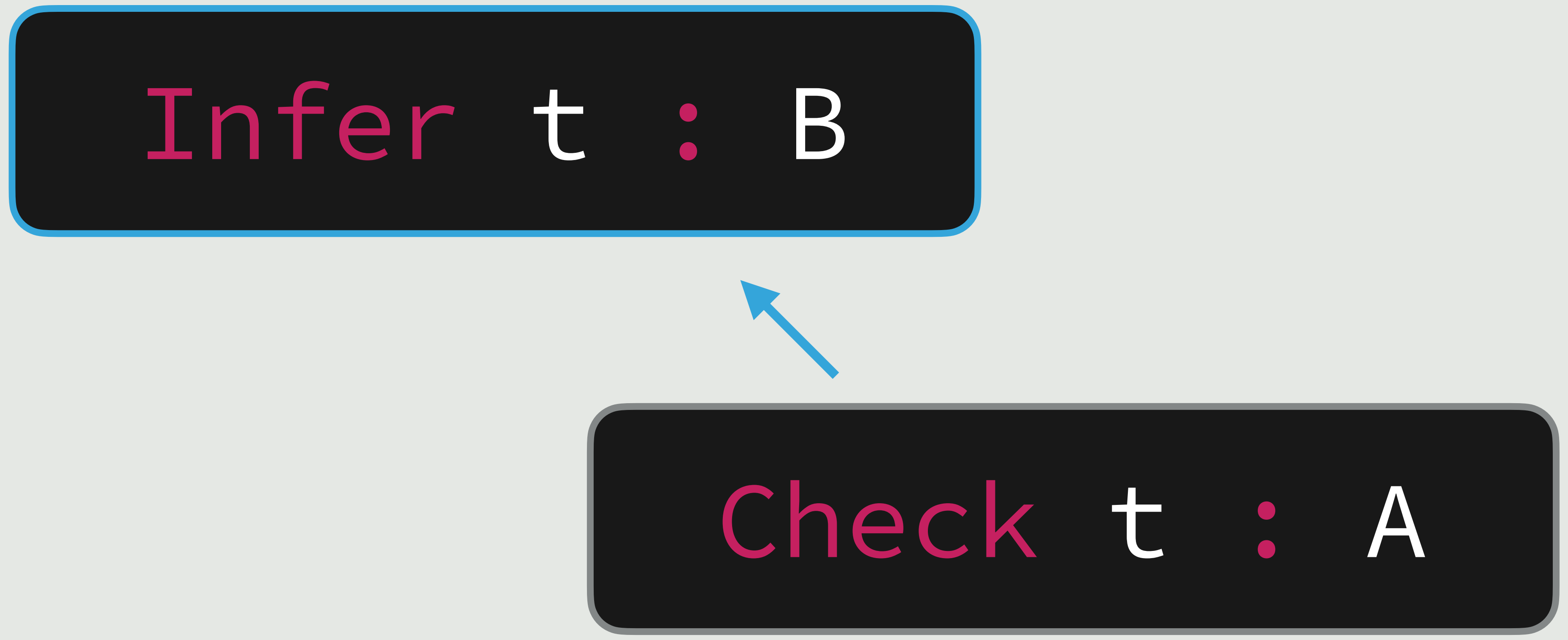
Cumulativity checking

```
iscumul :
  ∀ pb Σ Γ u v,
    welltyped Σ Γ u →
    welltyped Σ Γ v →
    dec ∥ Σ ; Γ ⊢ u ≤pb v ∥
```

Inference

```
infer : ∀ Σ Γ t, dec (∑ A, ∥ Σ ; Γ ⊢ t : A ∥)
```

```
check : ∀ Σ Γ t A, dec ∥ Σ ; Γ ⊢ t : A ∥
```



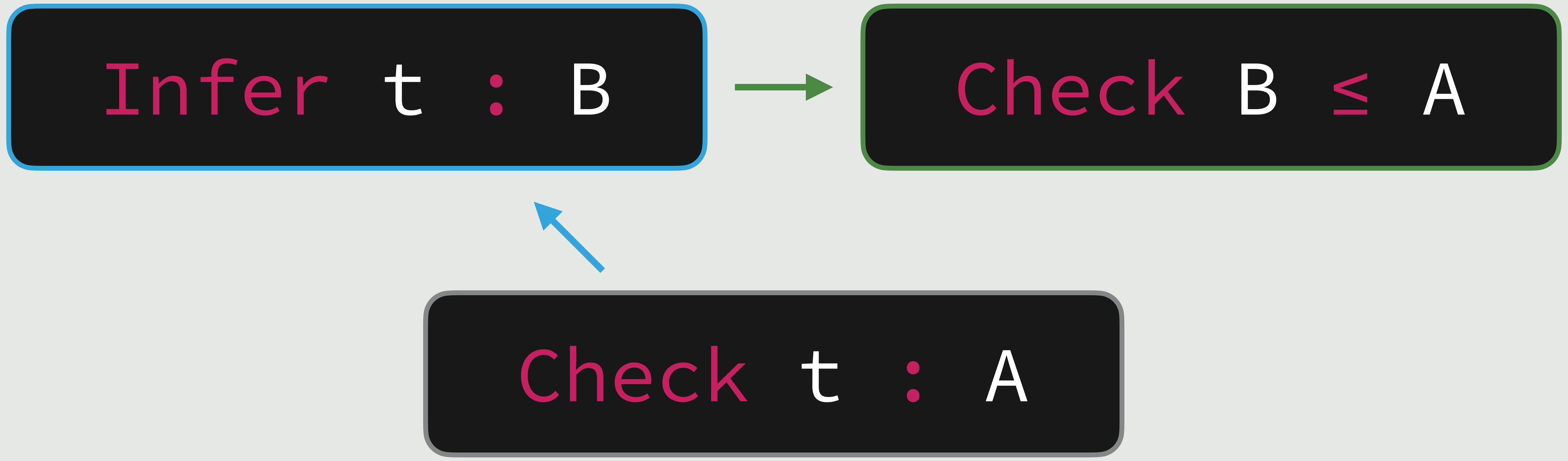
Cumulativity checking

```
iscumul :
  ∀ pb Σ Γ u v,
    welltyped Σ Γ u →
    welltyped Σ Γ v →
    dec ∥ Σ ; Γ ⊢ u ≤pb v ∥
```

Inference

```
infer : ∀ Σ Γ t, dec (∑ A, ∥ Σ ; Γ ⊢ t : A ∥)
```

```
check : ∀ Σ Γ t A, dec ∥ Σ ; Γ ⊢ t : A ∥
```



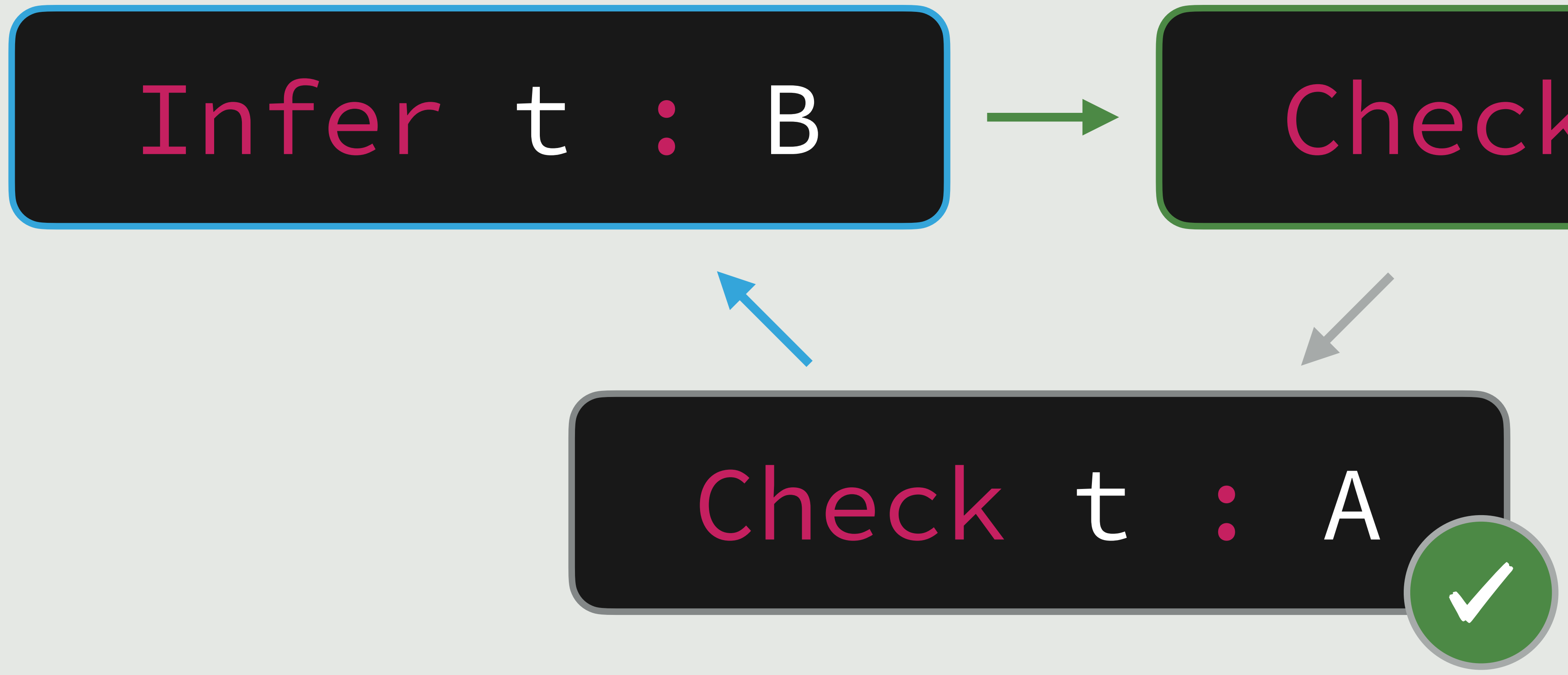
Cumulativity checking

```
iscumul :
  ∀ pb Σ Γ u v,
    welltyped Σ Γ u →
    welltyped Σ Γ v →
    dec ∥ Σ ; Γ ⊢ u ≤pb v ∥
```

Inference

```
infer : ∀ Σ Γ t, dec (∑ A, ∥ Σ ; Γ ⊢ t : A ∥)
```

```
check : ∀ Σ Γ t A, dec ∥ Σ ; Γ ⊢ t : A ∥
```



Cumulativity checking

```
iscumul :
  ∀ pb Σ Γ u v,
    welltyped Σ Γ u →
    welltyped Σ Γ v →
    dec ∥ Σ ; Γ ⊢ u ≤pb v ∥
```

Inference

```
infer : ∀ Σ Γ t, dec (∑ A, ∥ Σ ; Γ ⊢ t : A ∥)
```

```
check : ∀ Σ Γ t A, dec ∥ Σ ; Γ ⊢ t : A ∥
```

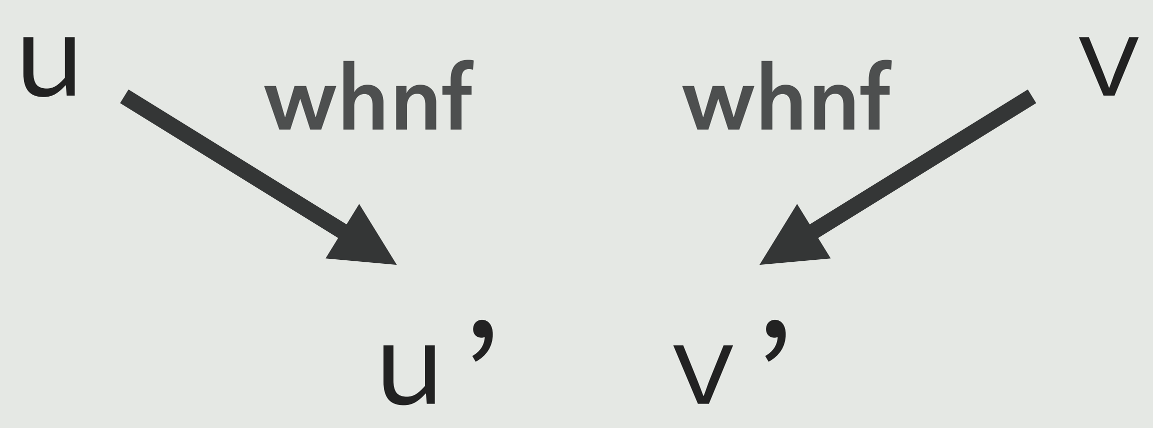
Cumulativity checking

```
iscumul :
  ∀ pb Σ Γ u v,
    welltyped Σ Γ u →
    welltyped Σ Γ v →
    dec ∥ Σ ; Γ ⊢ u ≤pb v ∥
```

Compare heads and proceed recursively

Cumulativity checking

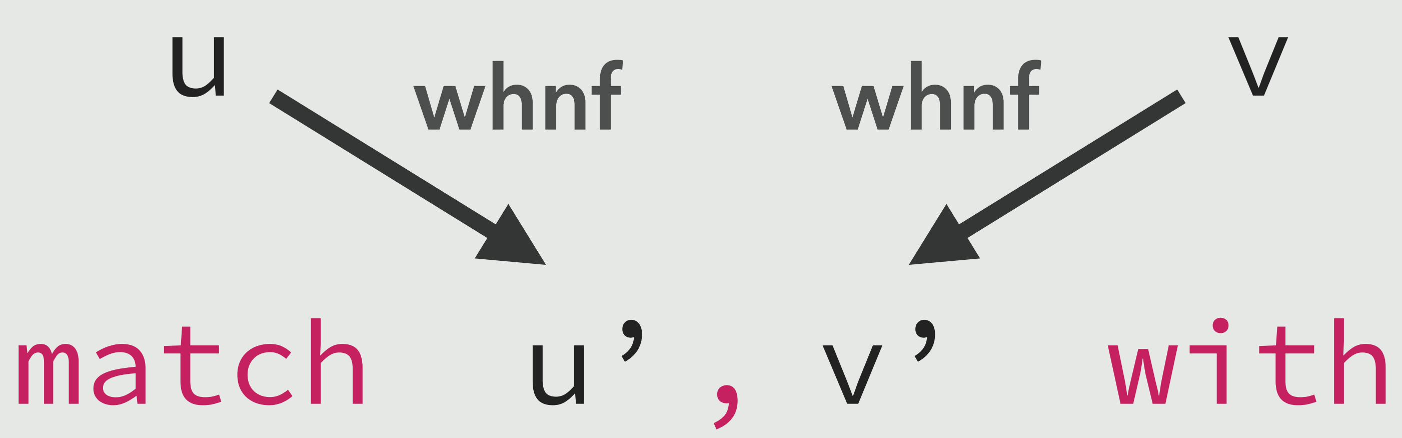
```
iscumul :
  ∀ pb Σ Γ u v,
    welltyped Σ Γ u →
    welltyped Σ Γ v →
    dec ‖ Σ ; Γ ⊢ u ≤pb v ‖
```



Compare heads and proceed recursively

Cumulativity checking

```
iscumul :
  ∀ pb Σ Γ u v,
    welltyped Σ Γ u →
    welltyped Σ Γ v →
    dec ‖ Σ ; Γ ⊢ u ≤pb v ‖
```



Compare heads and proceed recursively

Cumulativity checking

```
iscumul :
  ∀ pb Σ Γ u v,
    welltyped Σ Γ u →
    welltyped Σ Γ v →
    dec ‖ Σ ; Γ ⊢ u ≤pb v ‖
```

Compare heads and proceed recursively

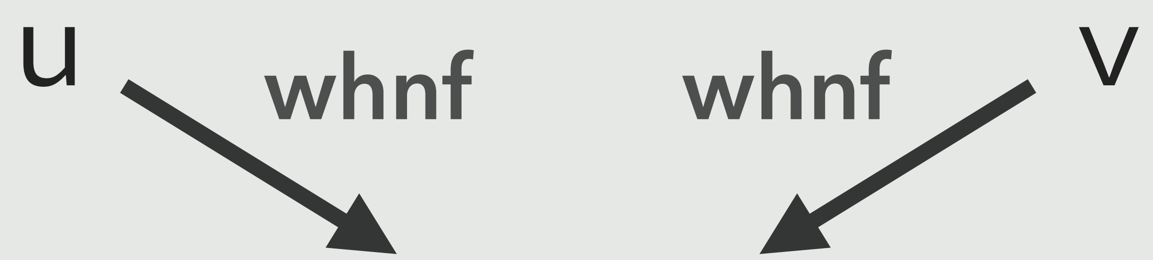
```
u  whnf        whnf  v

match  u' , v'  with
| tProd A0 B0, tProd A1 B1 ⇒
```

Cumulativity checking

```
iscumul :
  ∀ pb Σ Γ u v,
    welltyped Σ Γ u →
    welltyped Σ Γ v →
    dec ‖ Σ ; Γ ⊢ u ≤pb v ‖
```

💡 Compare heads and proceed recursively




```
match  u' , v'  with
| tProd A₀ B₀, tProd A₁ B₁ ⇒
  iscumul pb Σ Γ A₀ A₁ _ _ ;
```

Cumulativity checking

```
iscumul :
  ∀ pb Σ Γ u v,
    welltyped Σ Γ u →
    welltyped Σ Γ v →
    dec ‖ Σ ; Γ ⊢ u ≤pb v ‖
```

Compare heads and proceed recursively

```
u    whnf       whnf    v
        ↘         ↙
match   u' , v'   with
| tProd A0 B0, tProd A1 B1 ⇒
  iscumul pb Σ Γ A0 A1 _ _ ;
  iscumul pb Σ (Γ,A0) B0 B1 _ _
```

Cumulativity checking

```
iscumul :
  ∀ pb Σ Γ u v,
    welltyped Σ Γ u →
    welltyped Σ Γ v →
    dec ‖ Σ ; Γ ⊢ u ≤pb v ‖
```

Compare heads and proceed recursively

```
u   whnf      whnf   v
        ↘         ↙
match  u’, v’  with
| tProd A₀ B₀, tProd A₁ B₁ ⇒
  iscumul pb Σ Γ A₀ A₁ _ _ ;
  iscumul pb Σ (Γ,A₀) B₀ B₁ _ _
| (* … *)
end
```

Cumulativity checking

```
iscumul :
  ∀ pb Σ Γ u v,
    welltyped Σ Γ u →
    welltyped Σ Γ v →
    dec ∥ Σ ; Γ ⊢ u ≤pb v ∥
```

Compare heads and proceed recursively

u —whnf→ u'   v —whnf→ v'

```
match  u' , v'  with
| tProd A0 B0, tProd A1 B1 ⇒
  iscumul pb Σ Γ A0 A1 _ _ ;
  iscumul pb Σ (Γ,A0) B0 B1 _ _
| (* … *)
end
```

Termination

Cumulativity checking

```
iscumul :
  ∀ pb Σ Γ u v,
    welltyped Σ Γ u →
    welltyped Σ Γ v →
    dec ∥ Σ ; Γ ⊢ u ≤pb v ∥
```

💡 Compare heads and proceed recursively

u —whnf→ u'   v —whnf→ v'

```
match  u', v'  with
| tProd A₀ B₀, tProd A₁ B₁ ⇒
  iscumul pb Σ Γ A₀ A₁ _ _ ;
  iscumul pb Σ (Γ,A₀) B₀ B₁ _ _
| (* … *)
end
```

Termination

Typing

Cumulativity checking

```
iscumul :
  ∀ pb Σ Γ u v,
    welltyped Σ Γ u →
    welltyped Σ Γ v →
    dec ‖ Σ ; Γ ⊢ u ≤pb v ‖
```

Compare heads and proceed recursively

u —whnf→ u' , v' ←whnf— v

```
match  u', v'  with
| tProd A0 B0, tProd A1 B1 ⇒
  iscumul pb Σ Γ A0 A1 _ _ ;
  iscumul pb Σ (Γ,A0) B0 B1 _ _
| (* … *)
end
```

Termination

Typing

Cumulativity checking

```
iscumul :
  ∀ pb Σ Γ u v,
    welltyped Σ Γ u →
    welltyped Σ Γ v →
    dec ‖ Σ ; Γ ⊢ u ≤pb v ‖
```

Compare heads and proceed recursively

u —whnf→ u'   v —whnf→ v'

```
match  u', v'  with
| tProd A0 B0, tProd A1 B1 ⇒
  iscumul pb Σ Γ A0 A1 _ _ ;
  iscumul pb Σ (Γ,A0) B0 B1 _ _
| (* … *)
end
```

Soundness + Completeness

Termination

Typing

Cumulativity checking

```
iscumul :
  ∀ pb Σ Γ u v,
    welltyped Σ Γ u →
    welltyped Σ Γ v →
    dec ∥ Σ ; Γ ⊢ u ≤pb v ∥
```

💡 Compare heads and proceed recursively

u —whnf→ u'   v —whnf→ v'

```
match  u', v'  with
| tProd A₀ B₀, tProd A₁ B₁ ⇒
  iscumul pb Σ Γ A₀ A₁ _ _ ;
  iscumul pb Σ (Γ,A₀) B₀ B₁ _ _
| (* … *)
end
```

Soundness + Completeness

$A_0 \neq A_1 \to \text{tProd}\ A_0\ B_0 \neq \text{tProd}\ A_1\ B_1$

Termination

Typing

Cumulativity checking

```
iscumul :
  ∀ pb Σ Γ u v,
    welltyped Σ Γ u →
    welltyped Σ Γ v →
    dec ‖ Σ ; Γ ⊢ u ≤pb v ‖
```

Compare heads and proceed recursively

u —whnf→ u'   v —whnf→ v'

```
match  u' , v'  with
| tProd A₀ B₀, tProd A₁ B₁ ⇒
  iscumul pb Σ Γ A₀ A₁ _ _ ;
  iscumul pb Σ (Γ,A₀) B₀ B₁ _ _
| (* … *)
end
```

Soundness + Completeness

Termination

Typing

$A_0 \neq A_1 \to \mathrm{tProd}\ A_0\ B_0 \neq \mathrm{tProd}\ A_1\ B_1$

Cumulativity checking

```
iscumul :
  ∀ pb Σ Γ u v,
    welltyped Σ Γ u →
    welltyped Σ Γ v →
    dec ‖ Σ ; Γ ⊢ u ≤pb v ‖
```

Compare heads and proceed recursively

u —whnf→ u'   v —whnf→ v'

```
match  u', v'  with
| tProd A₀ B₀, tProd A₁ B₁ ⇒
  iscumul pb Σ Γ A₀ A₁ _ _ ;
  iscumul pb Σ (Γ,A₀) B₀ B₁ _ _
| (* … *)
end
```

Soundness + Completeness

$A_0 \neq A_1 \rightarrow \text{tProd}\ A_0\ B_0 \neq \text{tProd}\ A_1\ B_1$

Termination

SN

Typing

Cumulativity checking

```
iscumul :
  ∀ pb Σ Γ u v,
    welltyped Σ Γ u →
    welltyped Σ Γ v →
    dec ‖ Σ ; Γ ⊢ u ≤pb v ‖
```

Compare heads and proceed recursively

u whnf   whnf v

```
match  u', v'  with
| tProd A₀ B₀, tProd A₁ B₁ ⇒
  iscumul pb Σ Γ A₀ A₁ _ _ ;
  iscumul pb Σ (Γ,A₀) B₀ B₁ _ _
| (* … *)
end
```

Soundness + Completeness

$A_0 \neq A_1 \to \mathrm{tProd}\ A_0\ B_0 \neq \mathrm{tProd}\ A_1\ B_1$

Termination

SN

Typing

SR + context conversion

Cumulativity checking

Need algo sucht that: $u' \leftarrow u$ u' whnf

```
iscumul :
  ∀ pb Σ Γ u v,
    welltyped Σ Γ u →
    welltyped Σ Γ v →
    dec ‖ Σ ; Γ ⊢ u ≤pb v ‖
```

Compare heads and proceed recursively

u —whnf→ u'   v —whnf→ v'

```
match u', v' with
| tProd A0 B0, tProd A1 B1 ⇒
  iscumul pb Σ Γ A0 A1 _ _ ;
  iscumul pb Σ (Γ,A0) B0 B1 _ _
| (* … *)
end
```

Soundness + **Completeness**   **Termination**   **Typing**

$A_0 \neq A_1 \rightarrow \text{tProd } A_0\ B_0 \neq \text{tProd } A_1\ B_1$

SN

SR + context conversion

Weak head reduction

Goal

Input

u

Output

v

u ⇾ v

```
weak_head_reduce : ∀ (u : term), ∑ (v : term), u ⇾ v
```

Weak head reduction

Example

Input u

Output v u ⇾ v

foo 0

Weak head reduction

Example

Input **u**

Output **v** **u ⇾ v**

```
Definition foo := λ(x:nat). x.
```

```
foo 0
```

Weak head reduction

Example

Input u

Output v u ⇾ v

```
Definition foo := λ(x:nat). x.
```

```
foo 0
```

Weak head reduction

Example

Input u

Output v u ⇾ v

```
Definition foo := λ(x:nat). x.
```

```
foo 0
```

Weak head reduction

Example

Input u

Output v u ⇾ v

```
Definition foo := λ(x:nat). x.
```

foo 0

foo ⟶ λ(x:nat).x

Weak head reduction

Example

Input u

Output v | u ⇝ v

```
Definition foo := λ(x:nat). x.
```

```
λ(x:nat).x 0
```

foo ⟶ λ(x:nat).x

Weak head reduction

Example

Input u

Output v u → v

```
Definition foo := λ(x:nat). x.
```

0

foo ⟶ λ(x:nat).x

Weak head reduction

Example

Input u

Output v u → v

```
Definition foo := λ(x:nat). x.
```

0

foo 0 ⟶ (λ(x:nat).x) 0 ⟶ 0

Weak head reduction

Termination

Input

u

Output

v

u ⇝ v

Weak head reduction

Termination

Input

u π_1

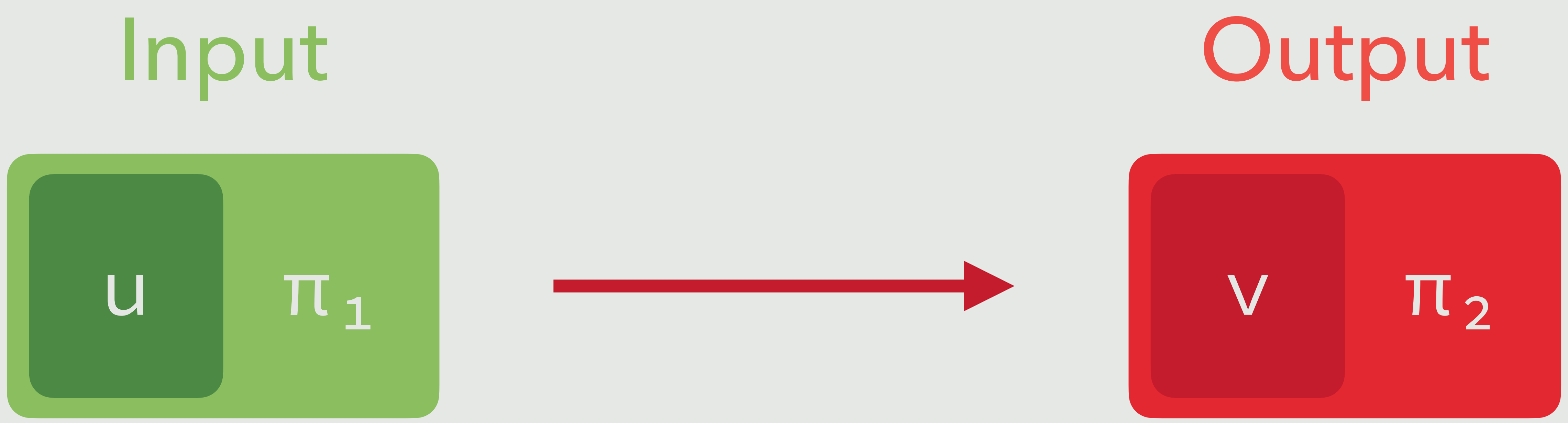
Output

v π_2

$u \to v$

Weak head reduction

Termination



Weak head reduction

Termination

```
foo 0
```

Weak head reduction

Termination

`foo 0` `foo 0` `λ(x:nat).x 0` `0`

Weak head reduction

Termination

(λ(x:nat).x) 0 ⟶ 0

Weak head reduction

Termination

foo 0 ⟶ (λ(x:nat).x) 0

(λ(x:nat).x) 0 ⟶ 0

Weak head reduction

Termination

`foo 0 ⟶ (λ(x:nat).x) 0`

`foo 0 ⊐ foo`

`(λ(x:nat).x) 0 ⟶ 0`

Weak head reduction

Termination

`foo 0 ⟶ (λ(x:nat).x) 0`

`foo 0 ⊐ foo`

`(λ(x:nat).x) 0 ⟶ 0`

Weak head reduction

Termination

`foo 0` ⟶ `(λ(x:nat).x) 0`

`foo 0` | `foo` `0` | `λ(x:nat).x` `0` | `0`

`foo 0 ⊐ foo`

`(λ(x:nat).x) 0` ⟶ `0`

1 Lexicographic order of ← and ⊏

Weak head reduction

Termination

foo 0 ⟶ (λ(x:nat).x) 0

foo 0

foo 0

λ(x:nat).x 0

0

foo 0 ⊐ foo

and foo 0 = foo 0

(λ(x:nat).x) 0 ⟶ 0

1 Lexicographic order of ← and ⊏

Weak head reduction

Termination

1 Lexicographic order of $\leftarrow$ and $\sqsubset$

Weak head reduction

Termination

`p.1`

1 Lexicographic order of $\leftarrow$ and $\sqsubset$

Weak head reduction

Termination

`p.1`

1 Lexicographic order of $\leftarrow$ and $\sqsubset$

Weak head reduction

Termination

p

Lexicographic order of $\leftarrow$ and $\sqsubset$

Weak head reduction

Termination

`p.1`

`p`

1 Lexicographic order of $\leftarrow$ and $\sqsubset$

Weak head reduction

Termination

p.1 p

$p.1 \sqsupset p$

1 Lexicographic order of $\leftarrow$ and $\sqsubset$

Weak head reduction

Termination

p.1

p

$p.1 \sqsupset p$

but $p.1 \neq p$

1 Lexicographic order of $\leftarrow$ and $\sqsubset$

Weak head reduction

Termination

`p.1` > `p.1`

`p.1 ⊐ p`

and `p.1 = p.1`

1 Lexicographic order of ← and ⊏

Weak head reduction

Termination

fix f (n:nat). t end n

1 Lexicographic order of $\leftarrow$ and $\sqsubset$

Weak head reduction

Termination

```
fix f (n:nat). t end n
```

1 Lexicographic order of ← and ⊏

Weak head reduction

Termination

```
fix f (n:nat). t end n
```

1 Lexicographic order of ← and ⊏

Weak head reduction

Termination

```
fix f (n:nat). t end n
```

Lexicographic order of ← and ⊏

Weak head reduction

Termination

```
fix f (n:nat). t end n
```

▼

```
fix f (n:nat). t end n
```

1 Lexicographic order of ← and ⊏

Weak head reduction

Termination

```
fix f (n:nat). t end n
```

▼

```
fix f (n:nat). t end n
```

~~Lexicographic order of $<$ and $\sqsubset$~~

Weak head reduction

Termination

~~Lexicographic order of $\to$ and $\sqsubset$~~

Weak head reduction

Termination

Lexicographic order of ← and an order on positions

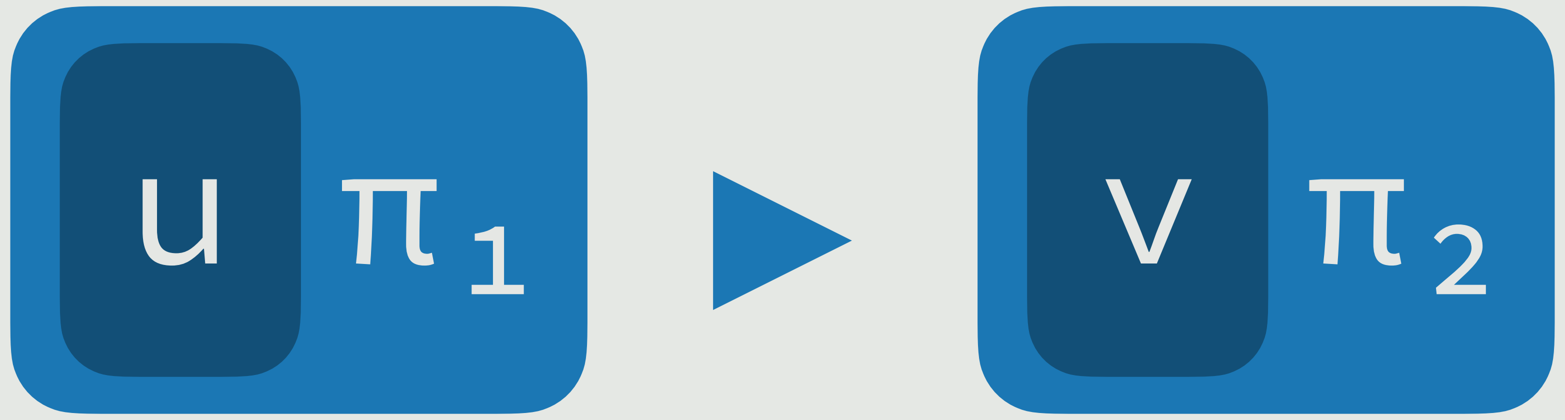
Weak head reduction

Termination

Lexicographic order of $\leftarrow$ and an order on positions

Weak head reduction

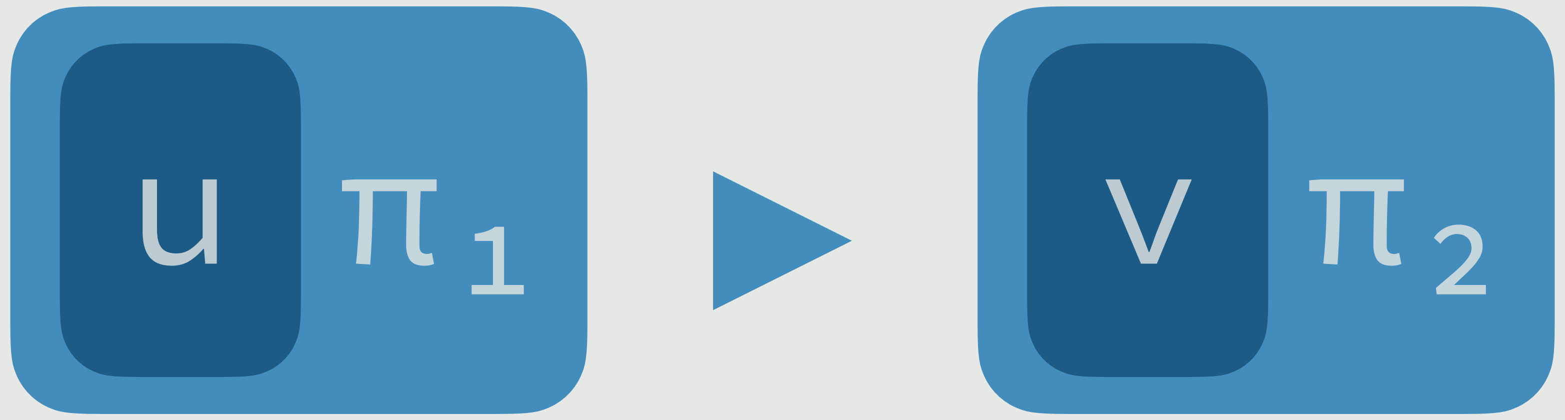
Termination



Lexicographic order of $\leftarrow$ and an order on positions

Weak head reduction

Termination

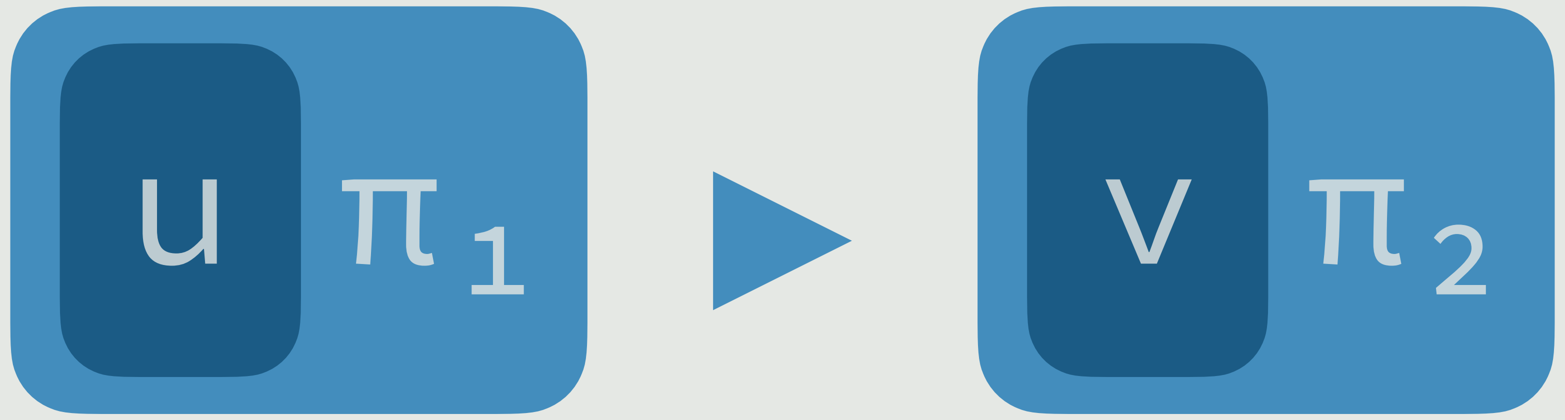


$$\langle\, u\ \pi_1\ ,\ \texttt{stack\_pos}\ u\ \pi_1 \,\rangle > \langle\, v\ \pi_2\ ,\ \texttt{stack\_pos}\ v\ \pi_2 \,\rangle$$

Lexicographic order of ← and an order on positions

Weak head reduction

Termination

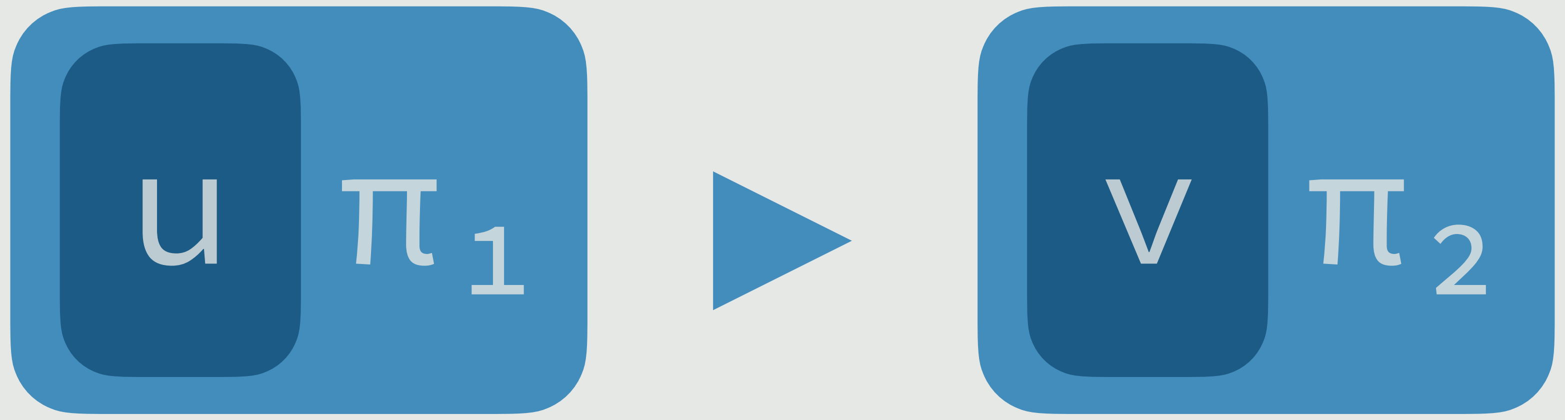


$$\langle\ u\ \pi_1\ ,\ \underbrace{\texttt{stack\_pos}\ u\ \pi_1}_{\texttt{pos}\ (u\ \pi_1)}\ \rangle > \langle\ v\ \pi_2\ ,\ \texttt{stack\_pos}\ v\ \pi_2\ \rangle$$

2 Lexicographic order of ← and an order on positions

Weak head reduction

Termination

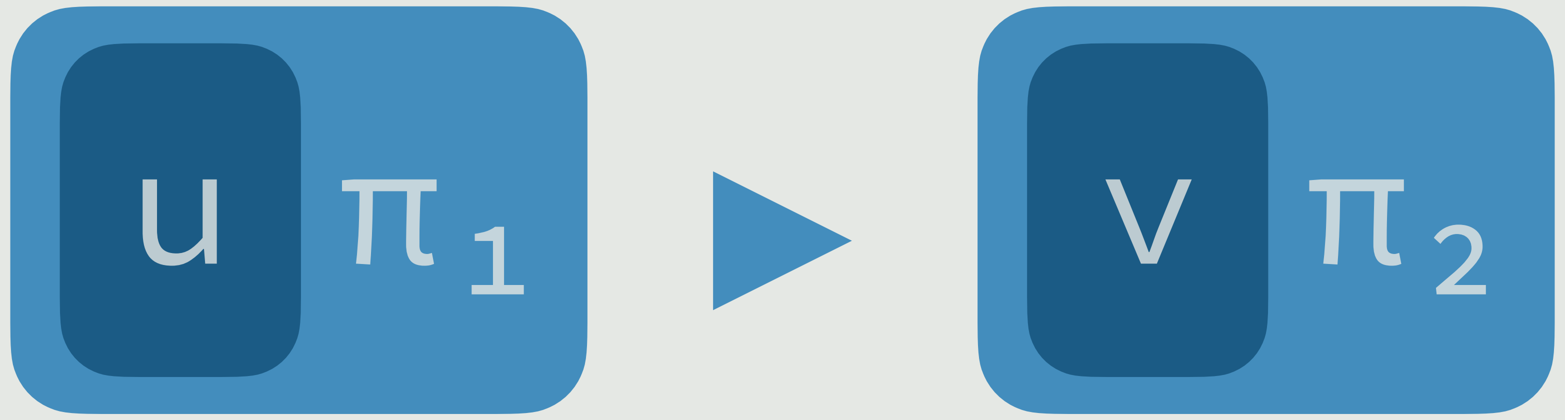


$$\langle\ u\ \pi_1\ ,\ \underbrace{\texttt{stack\_pos}\ u\ \pi_1}_{\texttt{pos}\ (u\ \pi_1)}\ \rangle > \langle\ v\ \pi_2\ ,\ \underbrace{\texttt{stack\_pos}\ v\ \pi_2}_{\texttt{pos}\ (v\ \pi_2)}\ \rangle$$

Lexicographic order of ← and an order on positions

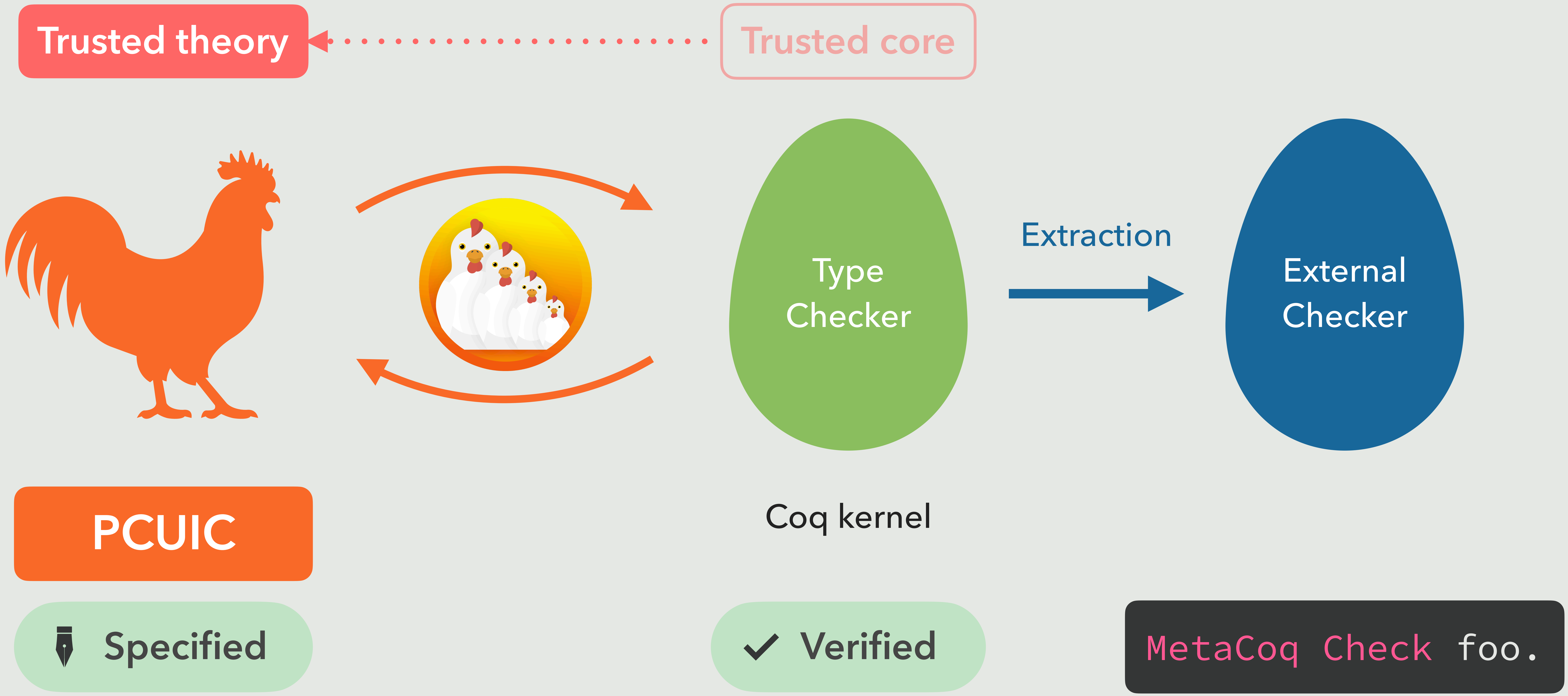
Weak head reduction

Termination



$$\langle\, u\ \pi_1\ ,\ \underbrace{\texttt{stack\_pos}\ u\ \pi_1}_{\texttt{pos}\ (u\ \pi_1)}\,\rangle > \langle\, v\ \pi_2\ ,\ \underbrace{\texttt{stack\_pos}\ v\ \pi_2}_{\texttt{pos}\ (v\ \pi_2)}\,\rangle$$

Dependent lexicographic order of ← and an order on positions



Trusted theory
Trusted core
Type Checker
Extraction
External Checker
PCUIC
Coq kernel
Specified
Verified
MetaCoq Check foo.

Conclusion and perspectives

MetaCoq
+ Rewrite rules

Extensible MetaCoq

Coq checks itself

MiniCoq kernel?

Guard condition at the source of a lot of bugs

Incompleteness found in the implementation!