pure type systems

```
fn (x : t) ⇒ e
```

```
e1(e2)
```

$$\lambda (x : \tau)\, e$$

$$e_1 (e_2)$$

```
fn (x : 'a) ⇒ e
```

```
(e1 : t1 → t2)(e2)
```

$$\Lambda(t)\,\lambda(x : t)\,e$$

$$e_1[t_1 \to t_2](e_2)$$

$$
\begin{aligned}
\tau &::= t \\
&\mid \tau_1 \to \tau_2 \\
&\mid \forall t.\, \tau
\end{aligned}
\qquad
\begin{aligned}
e &::= x \\
&\mid \lambda (x : \tau)\, e \\
&\mid e_1 (e_2) \\
&\mid \Lambda (t)\, e \\
&\mid e[\tau]
\end{aligned}
$$

System F

second-order logic

τ	proposition
e	proof
$\rightarrow$	$\Longrightarrow$
λ	proof assumption
application	proof application
$\forall$	$\forall$
Λ	proof quantification
type application	proof specialization
take 312!	$\exists$

	term / term	type / term
abstraction	$\lambda(x:\tau)\,e$	$\Lambda(t)\,e$
application	$e_1(e_2)$	$e[\tau]$
type of abs	$\tau_1 \to \tau_2$	$\forall t.\,\tau$

type constructors

```
'a list

'a -> 'b

Functor f

Map k v
```

type constructors

`'a list`	`list : * -> *`
`'a -> 'b`	`-> : * -> * -> *`
`Functor f`	`Functor : * -> *`
`Map k v`	`Map : * -> * -> *`

kinds

$$\kappa ::= *$$

kind of types

$$\mid \kappa_1 \to \kappa_2$$

not the same as arrow type!

$$
\begin{aligned}
\tau &::= t \\
&\mid \tau_1 \to \tau_2 \\
&\mid \forall (t : \kappa)\, \tau \\
&\mid \lambda (t : \kappa)\, \tau \\
&\mid \tau_1\ \tau_2
\end{aligned}
\qquad
\begin{aligned}
e &::= x \\
&\mid \lambda (x : \tau)\, e \\
&\mid e_1 (e_2) \\
&\mid \Lambda (t : \kappa)\, e \\
&\mid e[\tau]
\end{aligned}
$$

different!

system F_ω

system F_ω

foundation for ML-style type/module systems

as powerful as higher-order arithmetic!

	term / term	type / term	type / type
abs	$\lambda (x : \tau)\, e$	$\Lambda (t)\, e$	$\lambda (t : \kappa)\, \tau$
app	$e_1 (e_2)$	$e[\tau]$	$\tau_1\ \tau_2$
“type”	$\tau_1 \to \tau_2$	$\forall t.\, \tau$	$\kappa_1 \to \kappa_2$

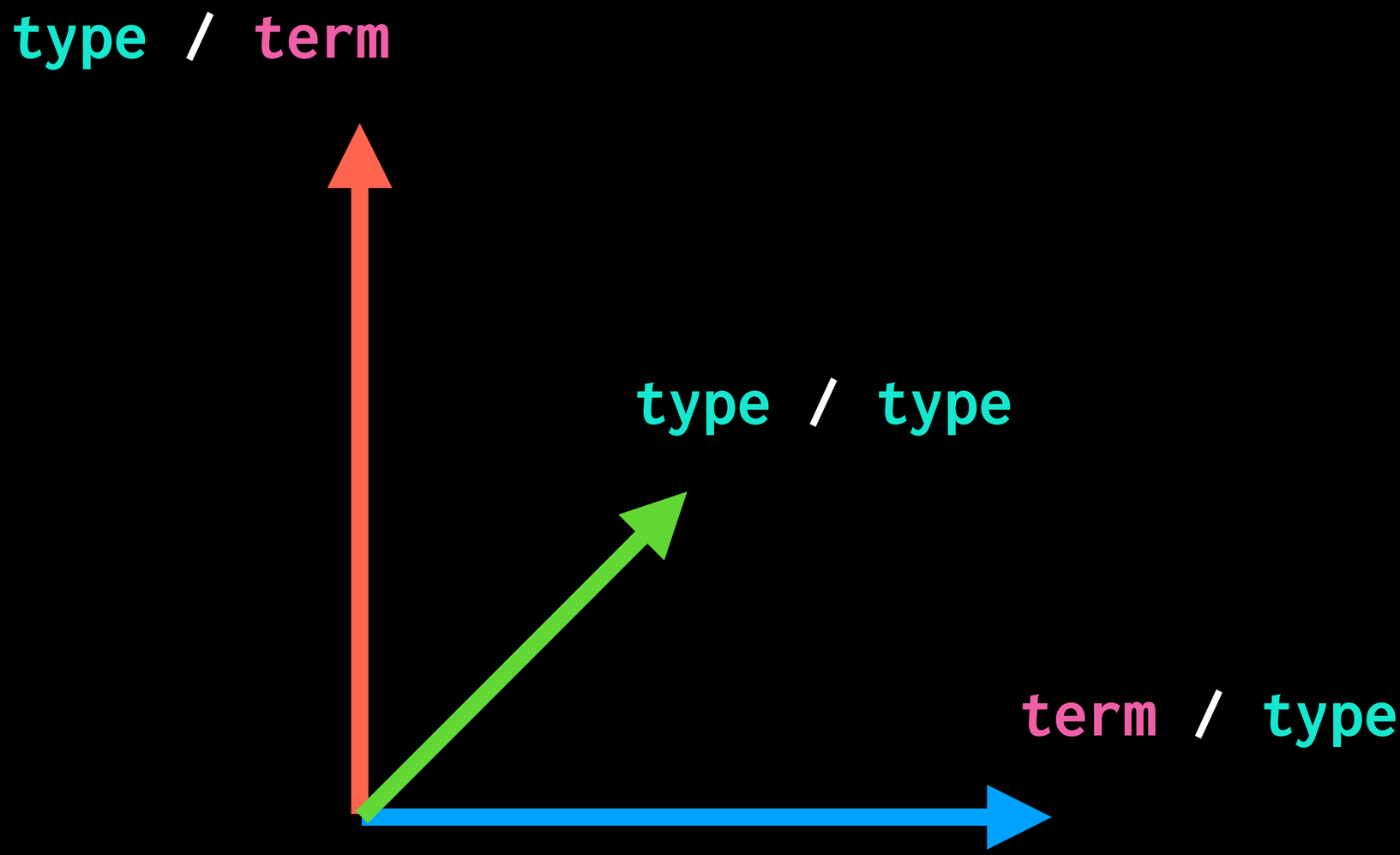
dependent types

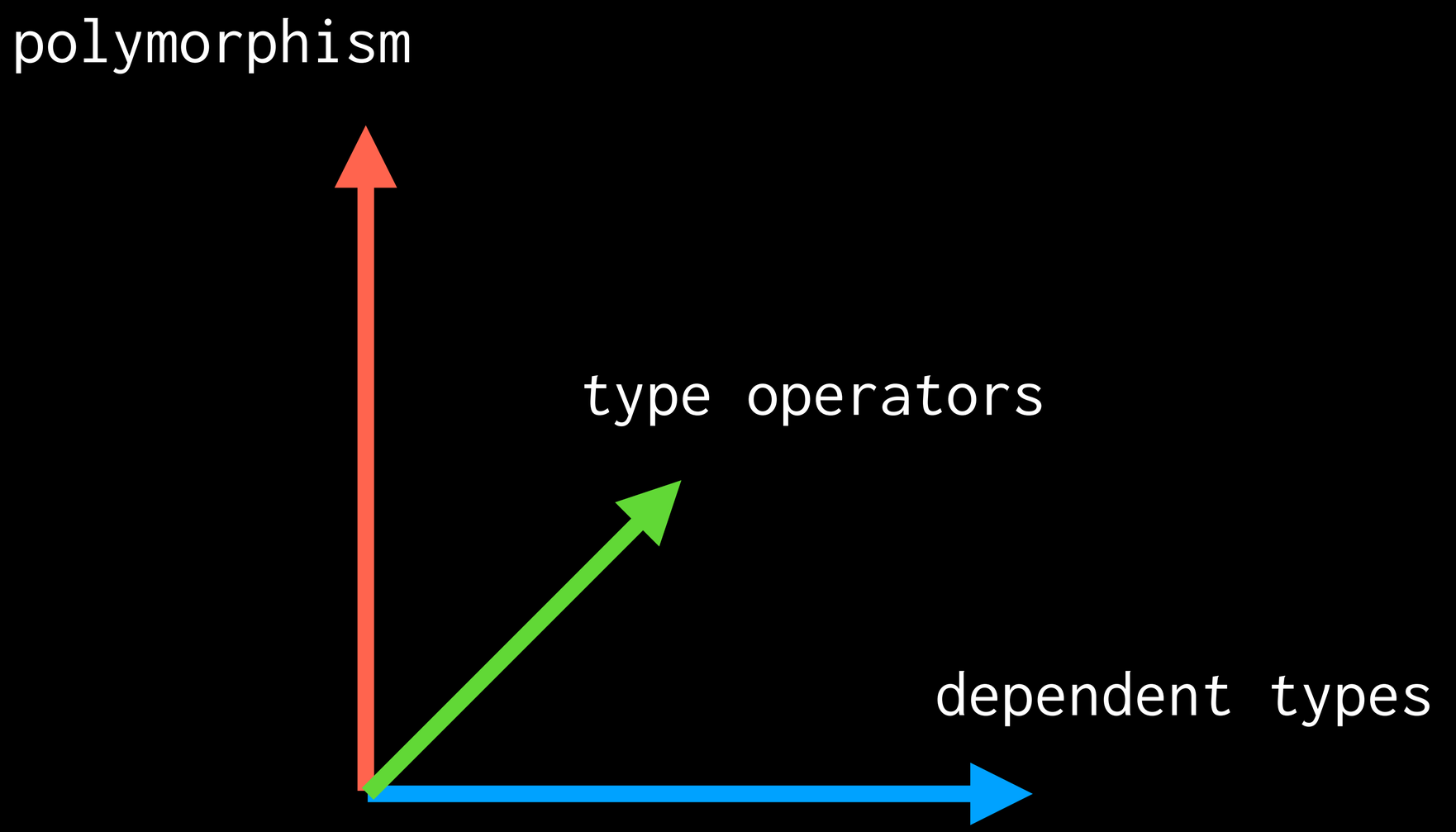
types that depend on values

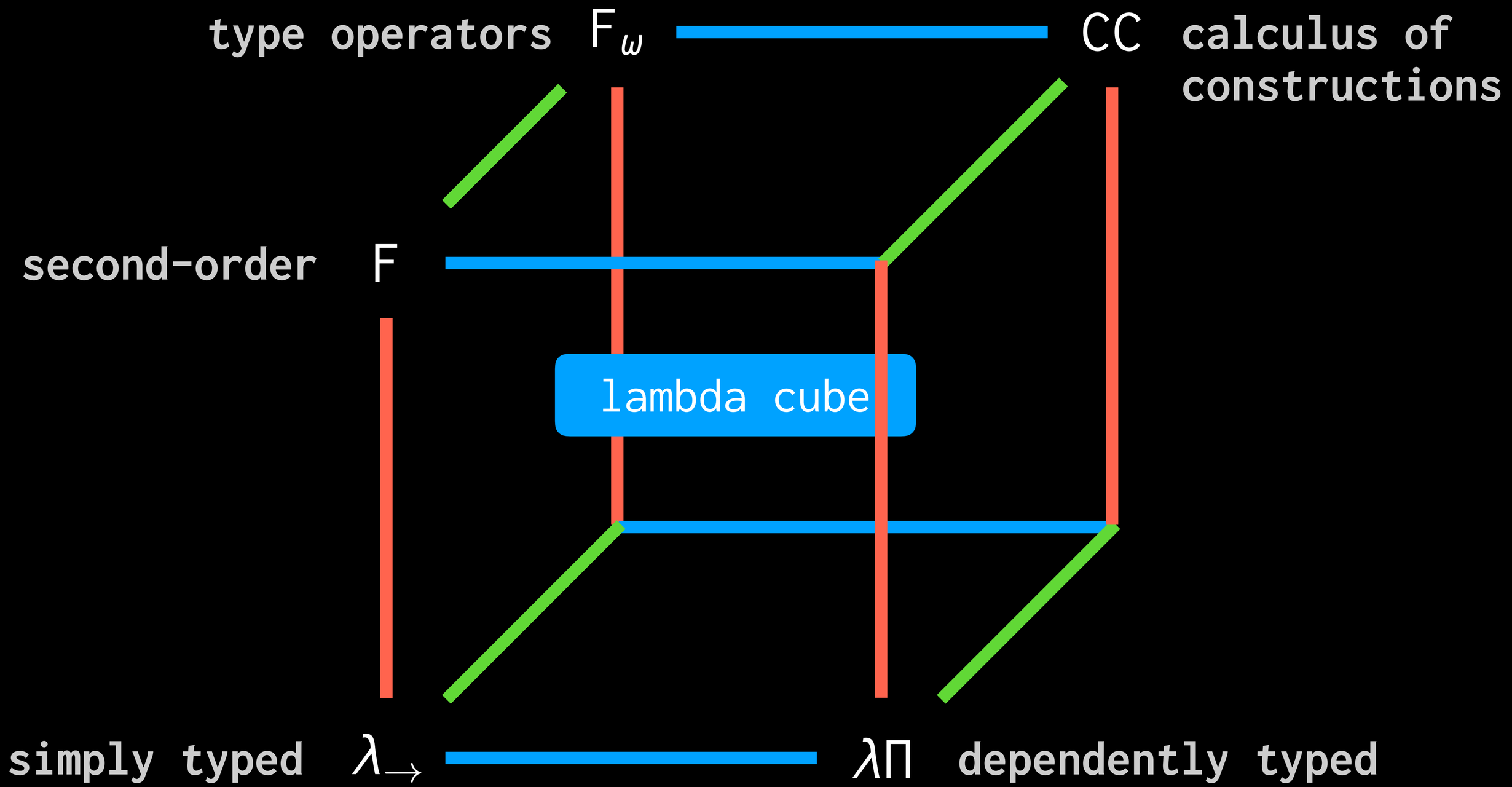
remember from last time?

	term / term	type / term	type / type	term / type
abs	$\lambda (x : \tau)\, e$	$\Lambda (t)\, e$	$\lambda (t : \kappa)\, \tau$	$\lambda (x : \tau)\, e$
app	$e_1 (e_2)$	$e[\tau]$	$\tau_1\ \tau_2$	$e_1 (e_2)$
“type”	$\tau_1 \to \tau_2$	$\forall t.\, \tau$	$\kappa_1 \to \kappa_2$	$\Pi (x : \tau_1)\, \tau_2$

plus a kind of type families


type / term
type / type
term / type


polymorphism
type operators
dependent types


type operators F_ω
CC calculus of constructions
second-order F
lambda cube
simply typed $\lambda_\rightarrow$
$\lambda\Pi$ dependently typed

calculus of constructions

λΠ is foundation for logical framework LF

CC is “F_ω with dependent types”,
or “λΠ with quantification”

foundation of Coq theorem prover

can supplement with (co)inductive types

$e : \tau$

$\tau : \kappa$

$\kappa : \mathtt{Kind}$

what if we went deeper?

$$t ::= s$$ sorts

$$\mid x$$

abstraction $$\mid \lambda (x : t)\, t$$

application $$\mid t_1\ t_2$$

product $$\mid \Pi (x : t)\, t$$

Type, Kind, …

$*, \square, \triangle, \ldots$

Sorts $\mathcal{S}$

$$\frac{\Gamma \vdash T : s}{\Gamma, x : T \vdash x : T}\ (\mathsf{Var})$$

$$\frac{\Gamma, x : S \vdash t : T \quad \Gamma \vdash \Pi\,(x : S)\,T : s}{\Gamma \vdash \lambda\,(x : S)\,t : \Pi\,(x : S)\,T}\ (\mathsf{Abs})$$

$$\frac{\Gamma \vdash t_1 : \Pi\,(x : S)\,T \quad \Gamma \vdash t_2 : S}{\Gamma \vdash t_1\ t_2 : [t_2/x]T}\ (\mathsf{App})$$

$$\frac{\Gamma \vdash T : s}{\Gamma, x : T \vdash x : T}\ (\text{Var})$$

$$\frac{\Gamma, x : S \vdash t : T \quad \Gamma \vdash \Pi\,(x : S)\,T : s}{\Gamma \vdash \lambda\,(x : S)\,t : \Pi\,(x : S)\,T}\ (\text{Abs})$$

$$\frac{\Gamma \vdash t_1 : \Pi\,(x : S)\,T \quad \Gamma \vdash t_2 : S}{\Gamma \vdash t_1\ t_2 : [t_2/x]T}\ (\text{App})$$

$$\frac{\Gamma \vdash T : s}{\Gamma, x : T \vdash x : T}\ (\text{Var})$$

need to check validity

$$\frac{\Gamma, x : S \vdash t : T \qquad \Gamma \vdash \Pi\,(x : S)\,T : s}{\Gamma \vdash \lambda\,(x : S)\,t : \Pi\,(x : S)\,T}\ (\text{Abs})$$

$$\frac{\Gamma \vdash t_1 : \Pi\,(x : S)\,T \qquad \Gamma \vdash t_2 : S}{\Gamma \vdash t_1\ t_2 : [t_2/x]T}\ (\text{App})$$

$$\frac{\Gamma \vdash T : s}{\Gamma, x : T \vdash x : T}\ (\text{Var})$$

$$\frac{\Gamma, x : S \vdash t : T \quad \Gamma \vdash \Pi\,(x : S)\,T : s}{\Gamma \vdash \lambda\,(x : S)\,t : \Pi\,(x : S)\,T}\ (\text{Abs})$$

$$\frac{\Gamma \vdash t_1 : \Pi\,(x : S)\,T \quad \Gamma \vdash t_2 : S}{\Gamma \vdash t_1\ t_2 : [t_2/x]T}\ (\text{App})$$

$$\frac{\Gamma \vdash t : T \quad T \equiv T' \quad \Gamma \vdash T' : s}{\Gamma \vdash t : T'} \text{ (Conv)}$$

$$\frac{\Gamma \vdash S : s_i \quad \Gamma, x : S \vdash T : s_j}{\Gamma \vdash \Pi\,(x : S)\,T : s_j}\ (\mathtt{Pi})$$

for combinations of s_i and s_j

$$(s_i, s_j) \in \mathcal{R}$$

$\lambda_{\to}$	$\{ \quad (*, *) \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad \}$
$\lambda\Pi$	$\{ \quad (*, *), \quad (*, \Box) \qquad\qquad\qquad\qquad\qquad\qquad \}$
F	$\{ \quad (*, *), \qquad\qquad\qquad (\Box, *) \qquad\qquad\qquad \}$
F_ω	$\{ \quad (*, *), \qquad\qquad\qquad (\Box, *), \quad (\Box, \Box) \quad \}$
CC	$\{ \quad (*, *), \quad (*, \Box), \quad (\Box, *), \quad (\Box, \Box) \quad \}$

$$\frac{}{\Gamma \vdash s_i : s_j}\ (\text{Start})$$

for combinations of s_i and s_j

$$(s_i, s_j) \in \mathcal{A}$$

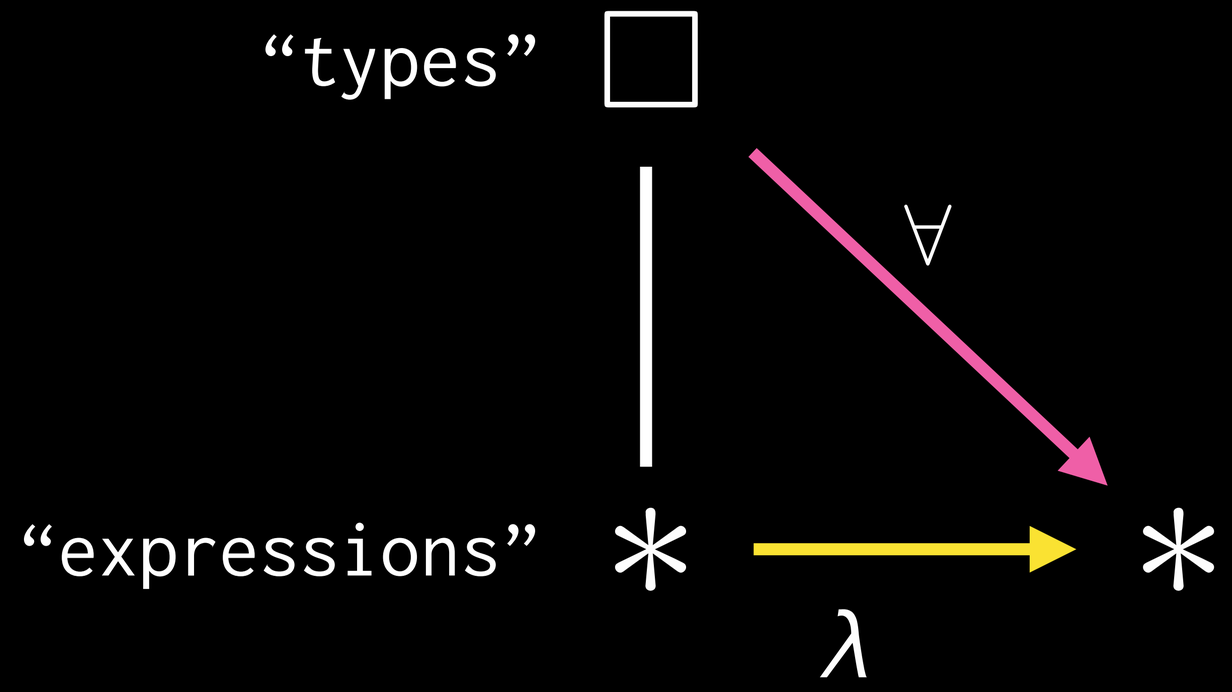
$$\mathcal{A} = (*, \square)$$

$$(\mathcal{S}, \mathcal{A}, \mathcal{R})$$

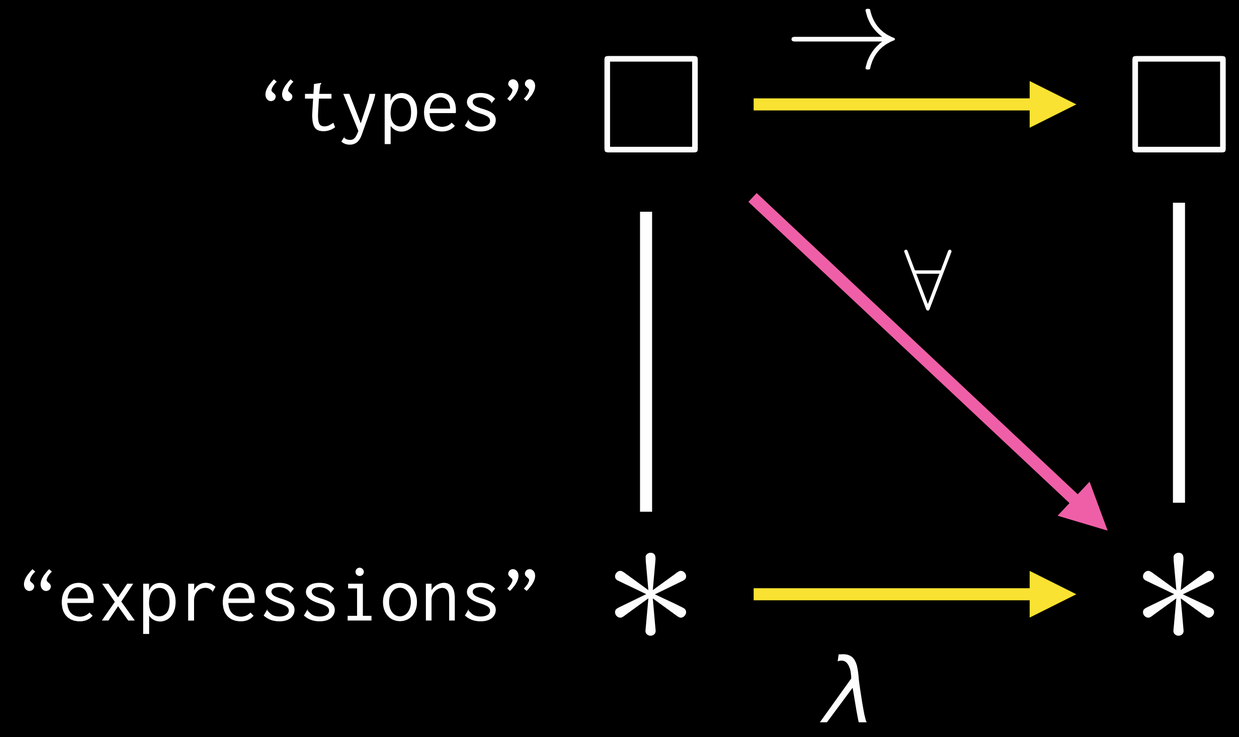
uniquely identifies a λ-calculus

$$\lambda_{\to} = (\{*, \Box\}, \{(*, \Box)\}, \{(*, *)\})$$

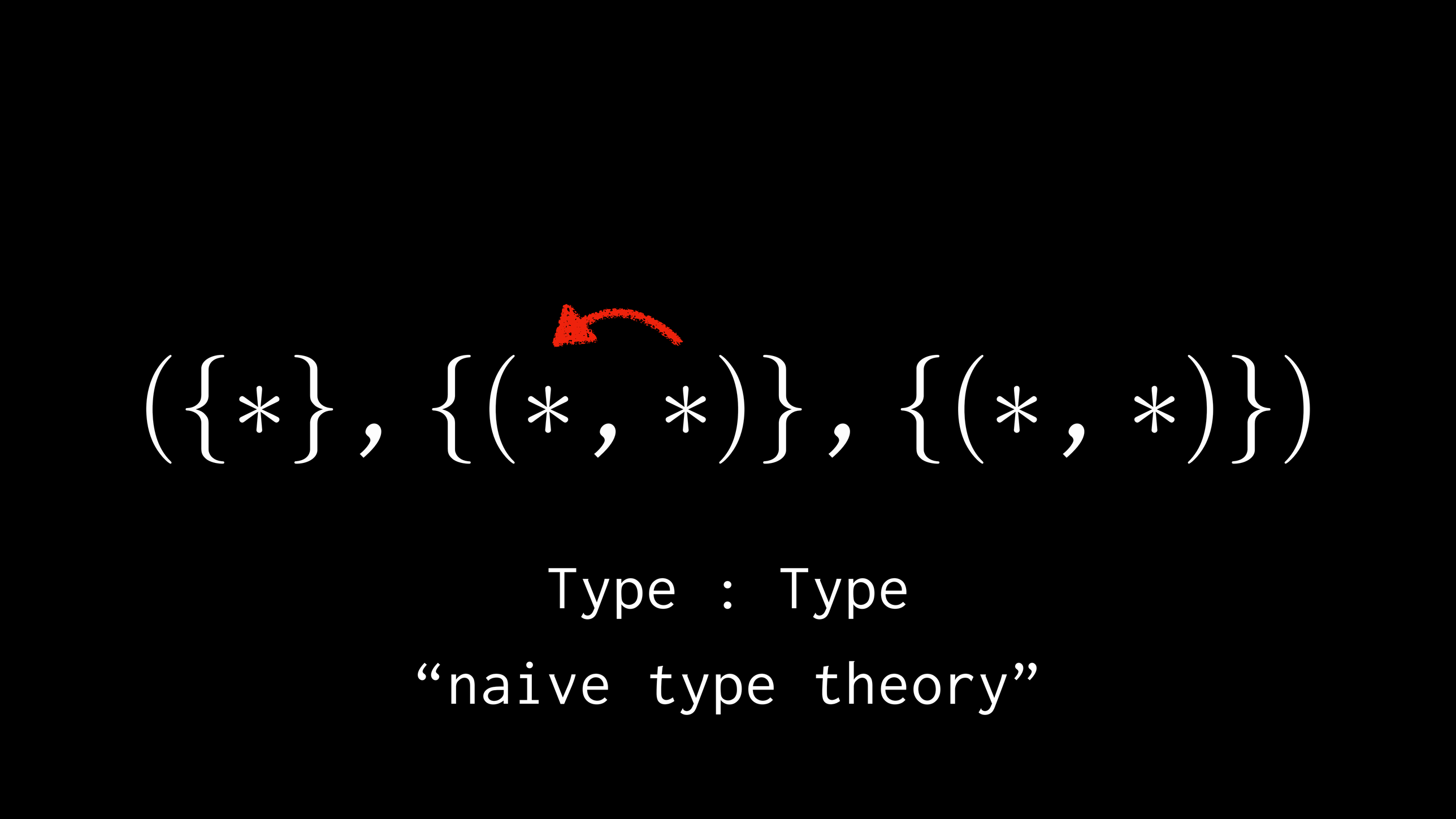
$$\mathsf{F} = (\{*, \Box\}, \{(*, \Box)\}, \{(*, *), (\Box, *)\})$$



system F



system F_ω


({*}, {(*, *)}, {(*, *)})
Type : Type
“naive type theory”

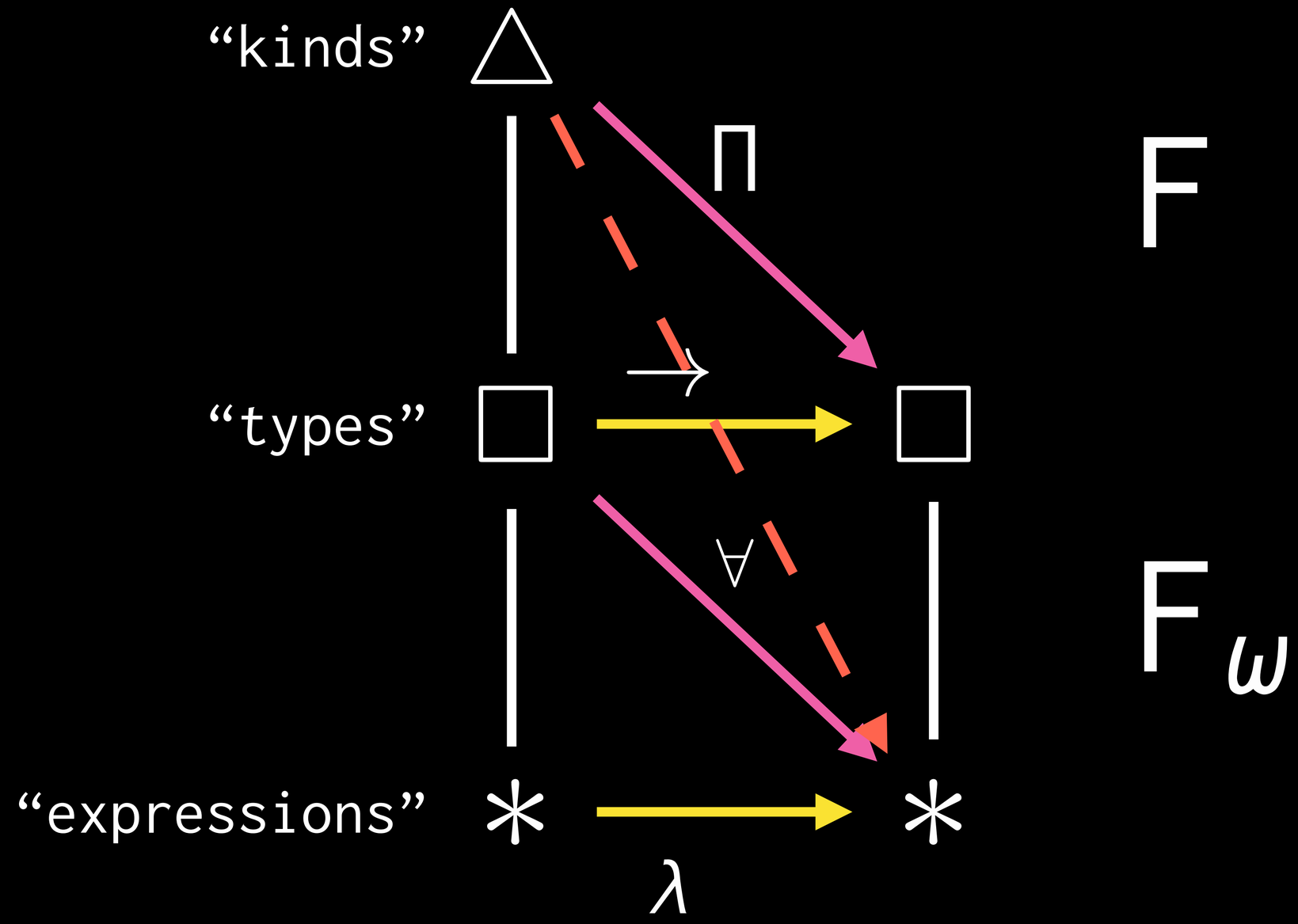
system U

$$(\{*, \square, \triangle\},$$
$$\{(*, \square), (\square, \triangle)\},$$
$$\{(*, *), (\square, *), (\square, \square), (\triangle, \square), (\triangle, *)\})$$

no cyclic dependency

system U^-

$(\{*, \square, \triangle\},$
$\{(*, \square), (\square, \triangle)\},$
$\{(*, *), (\square, *), (\square, \square), (\triangle, \square), (\triangle, *)\})$


“kinds”
“types”
“expressions”
⊓
→
∀
λ
F
F_ω

Girard (1972)

system U is inconsistent!

result known as *Girard's paradox*

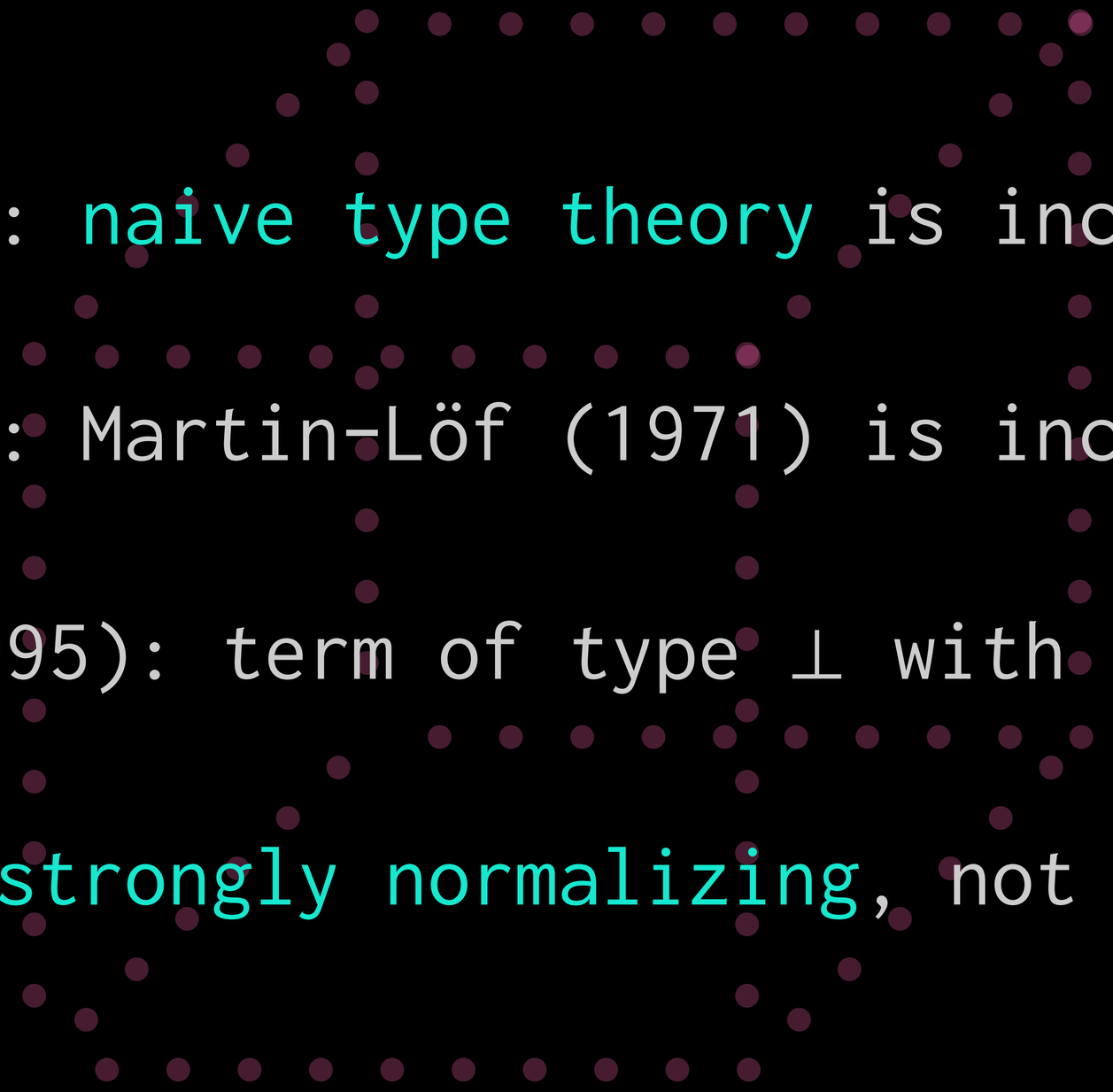
basis: every type is inhabited, including $\bot$

Girard’s paradox

analog of Russell’s paradox

but not the same: no *type comprehension*

embedding of *Burali-Forti paradox* in U



corollary: naive type theory is inconsistent

corollary: Martin-Löf (1971) is inconsistent

Hurkens (1995): term of type ⊥ with length 2039

lambda cube strongly normalizing, not all PTSs are

We consider the following universe:

$$\mathcal{U} \equiv \Pi\mathcal{X} : \Box.\,((\wp\wp\mathcal{X}{\to}\mathcal{X}){\to}\wp\wp\mathcal{X})$$

For each term t of type $\wp\wp\mathcal{U}$, we define a term of type $\mathcal{U}$:

$$\tau t \equiv \Lambda\mathcal{X} : \Box.\,\lambda f : (\wp\wp\mathcal{X}{\to}\mathcal{X}).\,\lambda p : \wp\mathcal{X}.\,(t\ \lambda x : \mathcal{U}.\,(p\ (f\ (\{x\ \mathcal{X}\}\ f))))$$

For each term s of type $\mathcal{U}$, we define a term of type $\wp\wp\mathcal{U}$:

$$\sigma s \equiv (\{s\ \mathcal{U}\}\ \lambda t : \wp\wp\mathcal{U}.\,\tau t)$$

(So we do not consider σ and τ as *terms*.)

We define normal terms of type $\wp\mathcal{U}$ and $\mathcal{U}$, respectively:

$$\Delta \equiv \lambda y : \mathcal{U}.\,\neg\forall p : \wp\mathcal{U}.\,[(\sigma y\ p) \Rightarrow (p\ \tau\sigma y)]$$
$$\Omega \equiv \text{the normal form of } \tau\ \lambda p : \wp\mathcal{U}.\,\forall x : \mathcal{U}.\,[(\sigma x\ p) \Rightarrow (p\ x)]$$

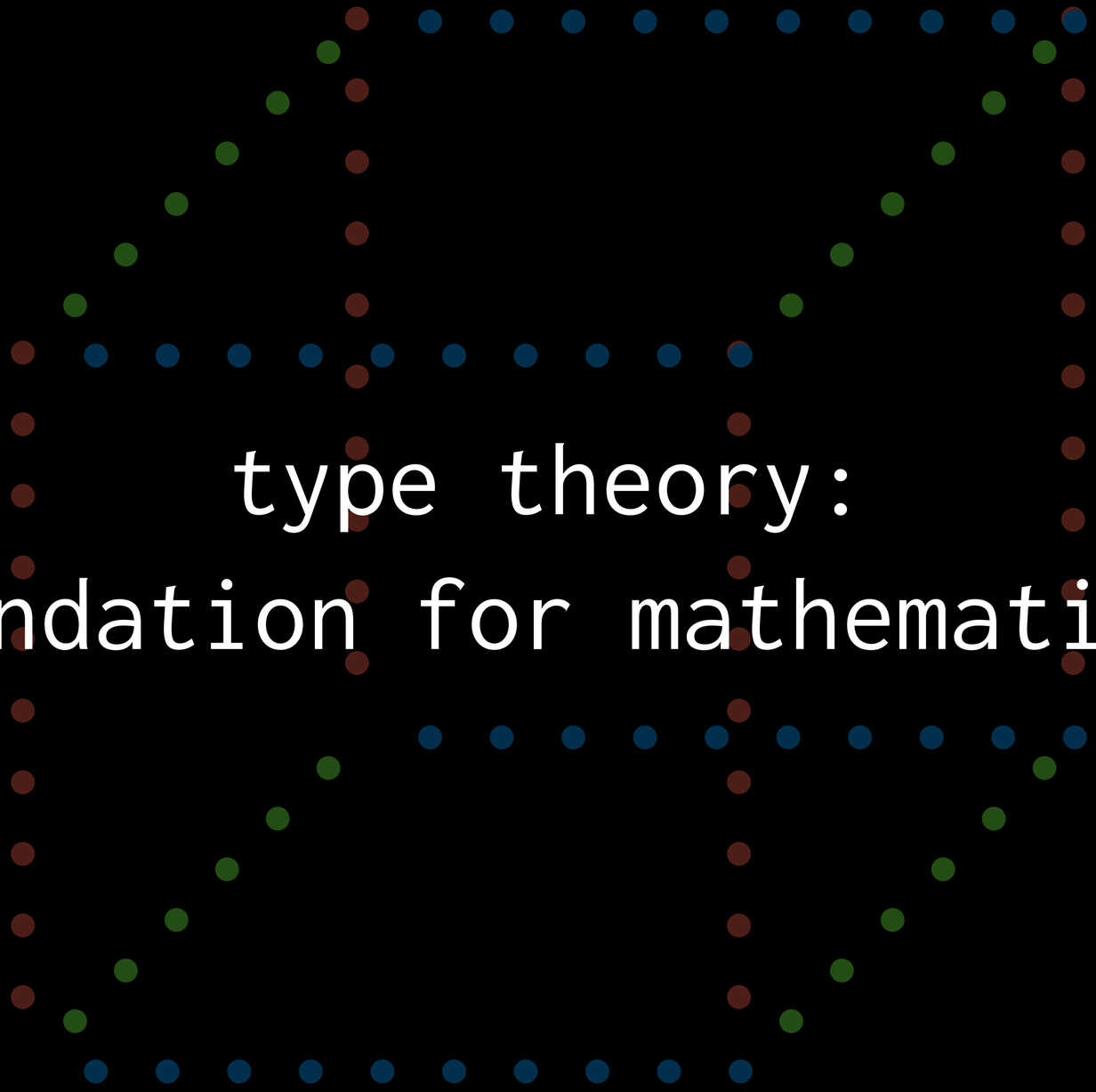
In other words, $\Omega \equiv \Lambda\mathcal{X} : \Box.\,\lambda f : (\wp\wp\mathcal{X}{\to}\mathcal{X}).\,\lambda p : \wp\mathcal{X}.\,\forall x : \mathcal{U}.\,[(\sigma x\ \lambda y : \mathcal{U}.\,(p\ (f\ (\{y\ \mathcal{X}\}\ f)))) \Rightarrow (p\ (f\ (\{x\ \mathcal{X}\}\ f)))]$.

We claim that the following is a term of type $\bot$ in $\lambda\mathrm{U}^-$:

$$[\mathsf{suppose}\ 0 : \forall p : \wp\mathcal{U}.\,[\forall x : \mathcal{U}.\,[(\sigma x\ p) \Rightarrow (p\ x)] \Rightarrow (p\ \Omega)].$$
$$[[\langle 0\ \Delta\rangle\ \mathsf{let}\ x : \mathcal{U}.\,\mathsf{suppose}\ 2 : (\sigma x\ \Delta).\,\mathsf{suppose}\ 3 : \forall p : \wp\mathcal{U}.\,[(\sigma x\ p) \Rightarrow (p\ \tau\sigma x)].$$
$$[[\langle 3\ \Delta\rangle\ 2]\ \mathsf{let}\ p : \wp\mathcal{U}.\,\langle 3\ \lambda y : \mathcal{U}.\,(p\ \tau\sigma y)\rangle\ ]]\ \mathsf{let}\ p : \wp\mathcal{U}.\,\langle 0\ \lambda y : \mathcal{U}.\,(p\ \tau\sigma y)\rangle]$$
$$\mathsf{let}\ p : \wp\mathcal{U}.\,\mathsf{suppose}\ 1 : \forall x : \mathcal{U}.\,[(\sigma x\ p) \Rightarrow (p\ x)].\,[\langle 1\ \Omega\rangle\ \mathsf{let}\ x : \mathcal{U}.\,\langle 1\ \tau\sigma x\rangle]]$$

Note that each subterm (except for the term itself) is normal. One easily verifies that (in the empty context) there is no normal term of type $\bot$ in $\lambda\mathrm{U}^-$. At the end of this article, we analyse the β-reduction behaviour of this proof term.

The proof is simple in the sense that it contains just 6 applications corresponding to *modus ponens*. In order to get an idea of the influence of abbreviations, one can also calculate the *length*: the total number of applications, abstractions, products, and occurrences of variables and sorts. For example, the terms abbreviated by $\bot$, $\mathcal{U}$, Δ, and Ω have length 3, 15, 241, and 145. The complete proof term has length 2039.



type theory:
foundation for mathematics?

further reading

seminal works:
http://ttic.uchicago.edu/~dreyer/course/papers/barendregt.pdf

http://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=EFA2A9409B12A54895CFD1EFBACEC439?doi=10.1.1.37.3153&rep=rep1&type=pdf

overviews:
ch. 2 of *Advanced Topics in Types and Programming Languages* (by Pierce)

https://ncatlab.org/nlab/show/pure+type+system

deeper dives:
https://people.mpi-sws.org/~skilpat/modsem/notes1.pdf

http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.56.7045&rep=rep1&type=pdf

http://www.cse.chalmers.se/research/group/logic/TypesSS05/Extra/miquel_sl3.pdf

https://github.com/scravy/types/blob/master/A%20Simplification%20of%20Girard's%20Paradox/hurkens95tlca.pdf