



pure type systems

fn (x : t) \Rightarrow e

e1(e2)

$\lambda(x : \tau) e$ $e_1(e_2)$

fn (x : 'a) \Rightarrow e

(e1 : t1 \Rightarrow t2) (e2)

$$\Lambda(t) \lambda(x : t) e$$
$$e_1 [t_1 \rightarrow t_2] (e_2)$$

$\tau ::= t$ $| \tau_1 \rightarrow \tau_2$ $| \forall t. \tau$ $e ::= x$ $| \lambda (x : \tau) e$ $| e_1 (e_2)$ $| \Lambda (t) e$ $| e[\tau]$

system F

second-order logic

	τ	proposition
	e	proof
	\rightarrow	\implies
	λ	proof assumption
application		proof application
	\forall	\forall
	\wedge	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

$\tau_1 \rightarrow \tau_2$

$\forall t. \tau$

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

$K ::= *$

$| K_1 \rightarrow K_2$

kinds

$K ::= *$

kind of types

$| K_1 \rightarrow K_2$

not the same as arrow type!

$\tau ::= t$	$e ::= x$
$\tau_1 \rightarrow \tau_2$	$\lambda(x : \tau) e$
$\forall(t : \kappa) \tau$	$e_1(e_2)$
$\lambda(t : \kappa) \tau$	$\Lambda(t : \kappa) e$
$\tau_1 \tau_2$	$e[\tau]$

system F_ω

$\tau ::= t$ $e ::= x$ $|\ \tau_1 \rightarrow \tau_2$ $|\ \lambda(x : \tau) e$ $|\ \forall(t : \kappa) \tau$ $|\ e_1(e_2)$

different!

 $|\ \lambda(t : \kappa) \tau$ $|\ \Lambda(t : \kappa) e$ $|\ \tau_1 \ \tau_2$ $|\ e[\tau]$

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” $\tau_1 \rightarrow \tau_2$ $\forall t. \tau$ $\kappa_1 \rightarrow \kappa_2$

dependent types

types that depend on values

remember from last time?

term / term

type / term

type / type

term / type

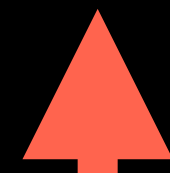
abs $\lambda(x : \tau) e$ $\Lambda(t) e$ $\lambda(t : \kappa) \tau$ $\lambda(x : \tau) e$

app $e_1(e_2)$ $e[\tau]$ $\tau_1 \tau_2$ $e_1(e_2)$

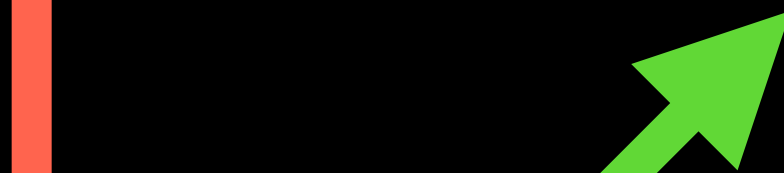
“type” $\tau_1 \rightarrow \tau_2$ $\forall t. \tau$ $\kappa_1 \rightarrow \kappa_2$ $\prod(x : \tau_1) \tau_2$

plus a kind of
type families

type / term



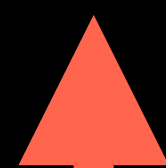
type / type



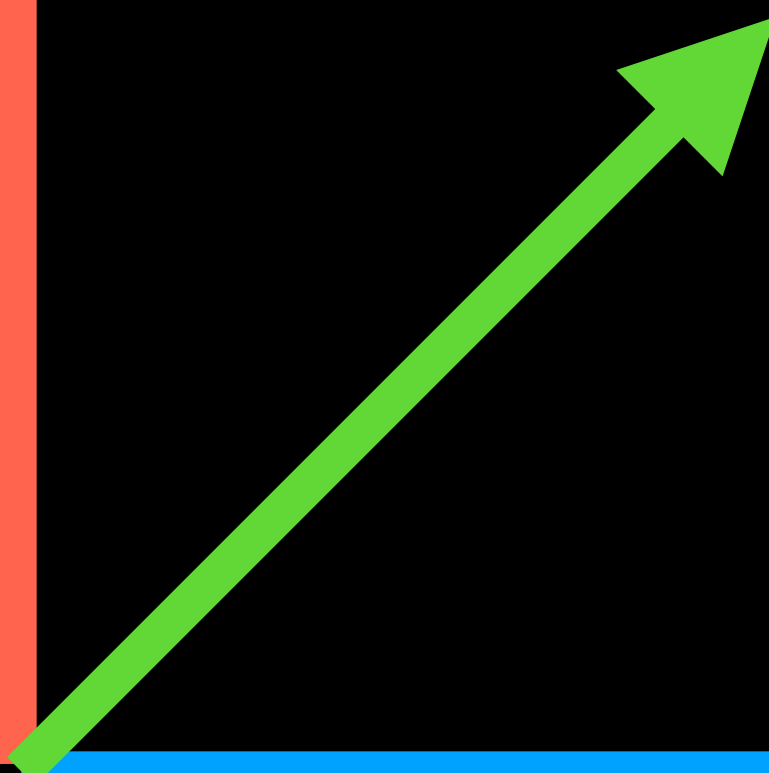
term / type



polymorphism

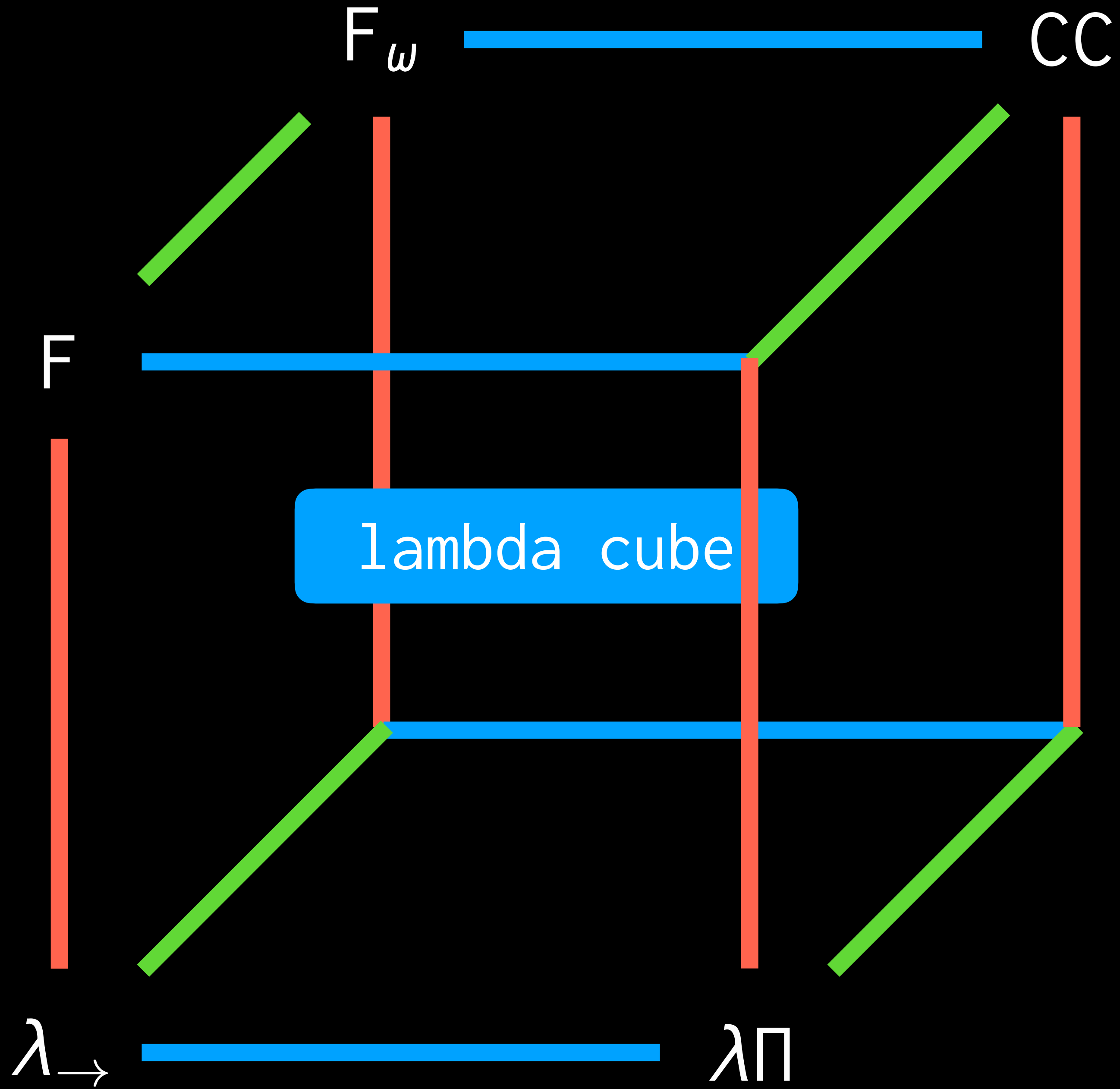


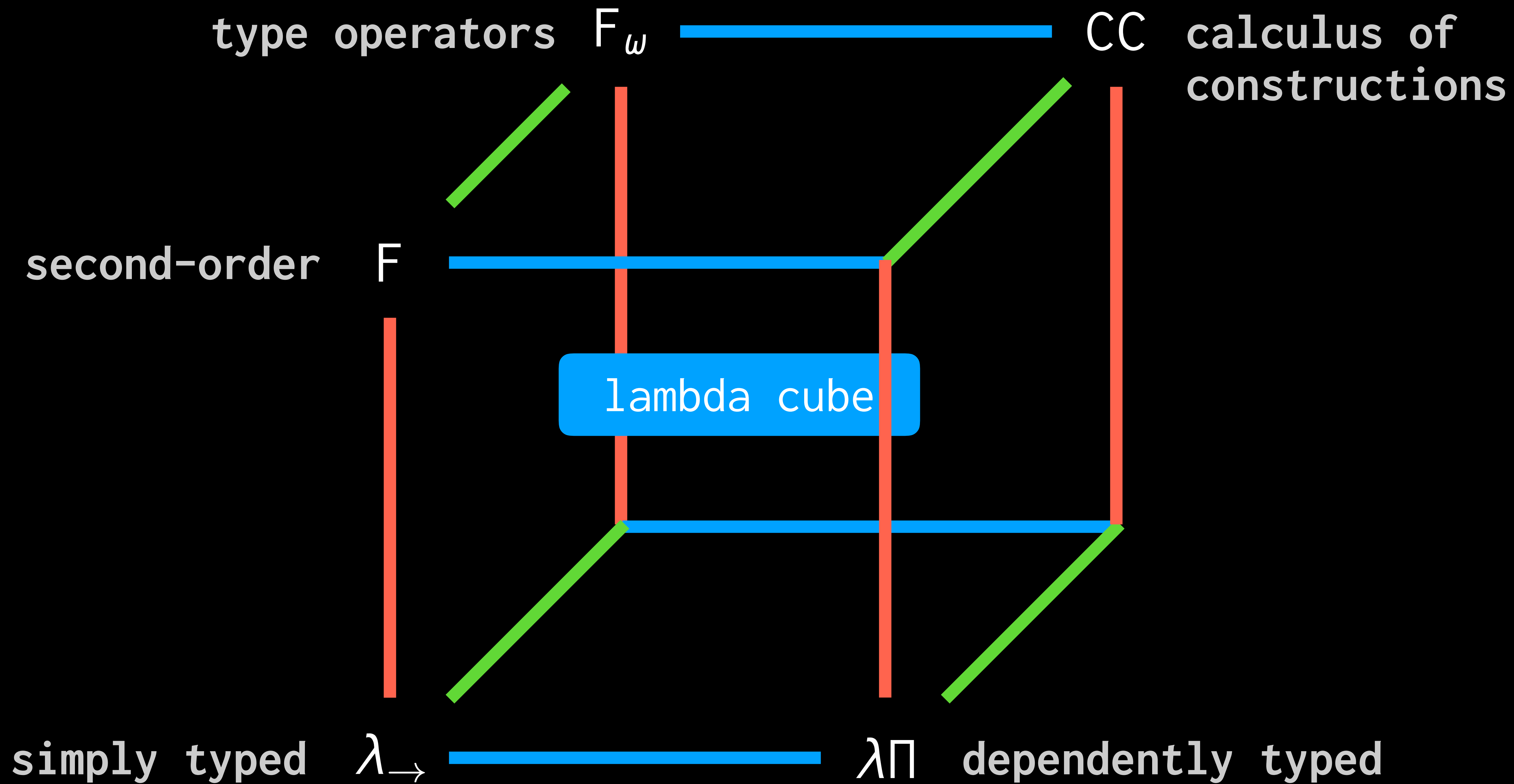
type operators



dependent types







calculus of constructions

$\lambda\Pi$ is foundation for logical framework **LF**

CC is “ F_ω with dependent types”,
or “ $\lambda\Pi$ with quantification”

foundation of **Coq** theorem prover

can supplement with **(co)inductive types**

$e : \tau$

$\tau : \kappa$

$\kappa : \text{Kind}$

what if we went deeper?

$t ::= s$

sorts

| x

| $\lambda (x : t) t$

| $t_1 t_2$

| $\Pi (x : t) t$

$t ::= s$

sorts

x

abstraction

$\lambda (x : t) t$

application

$t_1 t_2$

product

$\prod (x : t) t$

Type, Kind, ...

$*$, \square , \triangle , ...

Sorts \mathcal{S}

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

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

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

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

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

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

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

need to check validity

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

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

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

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

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

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

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

for combinations of s_i and s_j

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

λ_{\rightarrow}	$\{ (*, *) \}$
λ_{Π}	$\{ (*, *), (*, \square) \}$
F	$\{ (*, *), (\square, *) \}$
F_{ω}	$\{ (*, *), (\square, *), (\square, \square) \}$
CC	$\{ (*, *), (*, \square), (\square, *), (\square, \square) \}$

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

for combinations of s_i and s_j

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

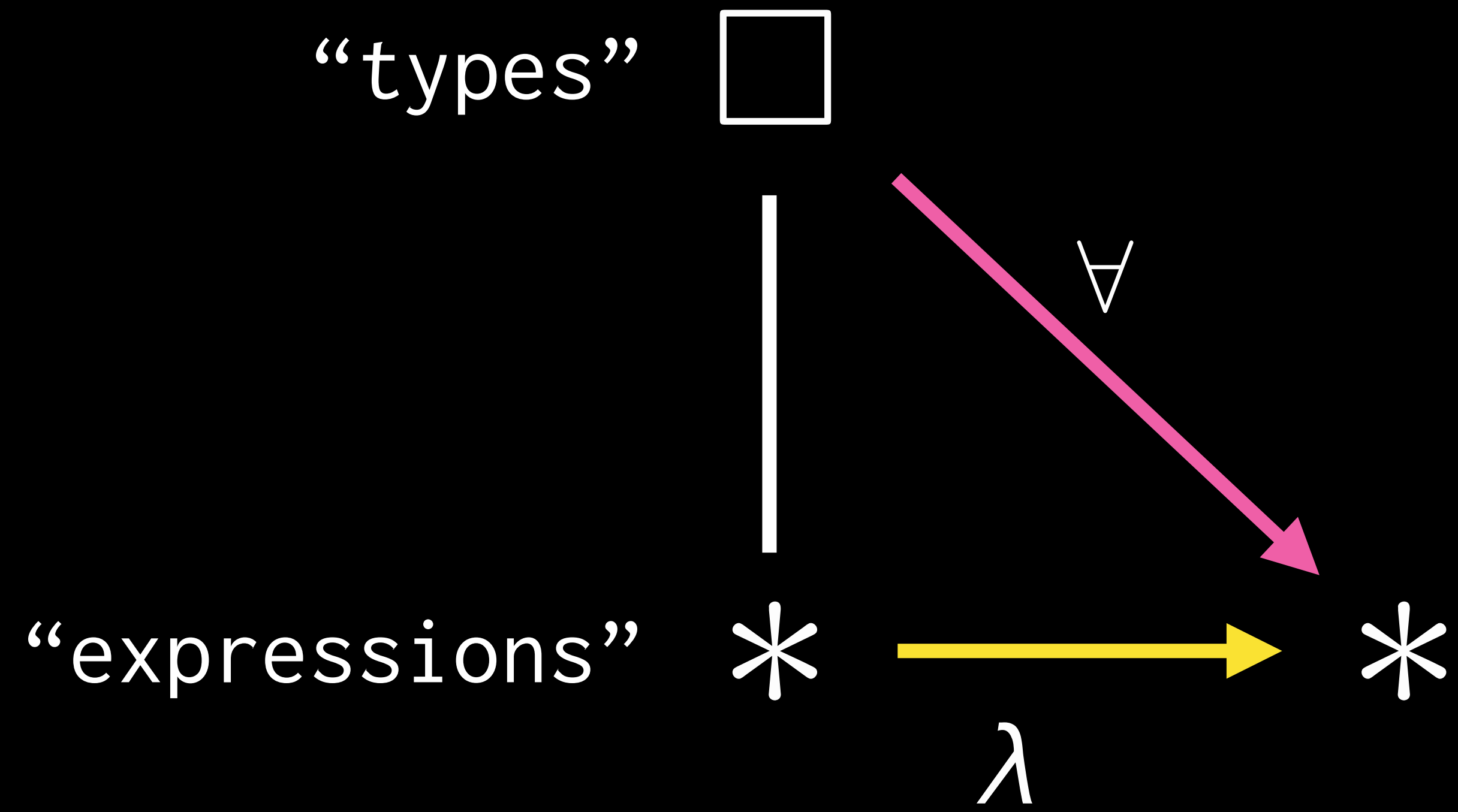
$$\mathcal{A} = (*, \square)$$

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

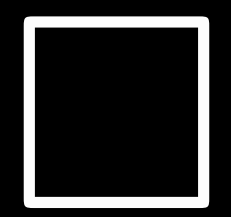
uniquely identifies a λ -calculus

$$\lambda_{\rightarrow} = (\{*, \square\}, \{(*, \square)\}, \{(*, *)\})$$

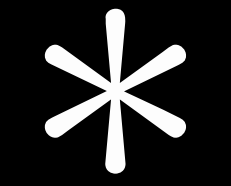
$$\mathbf{F} = (\{*, \square\}, \{(*, \square)\}, \{(*, *), (\square, *)\})$$



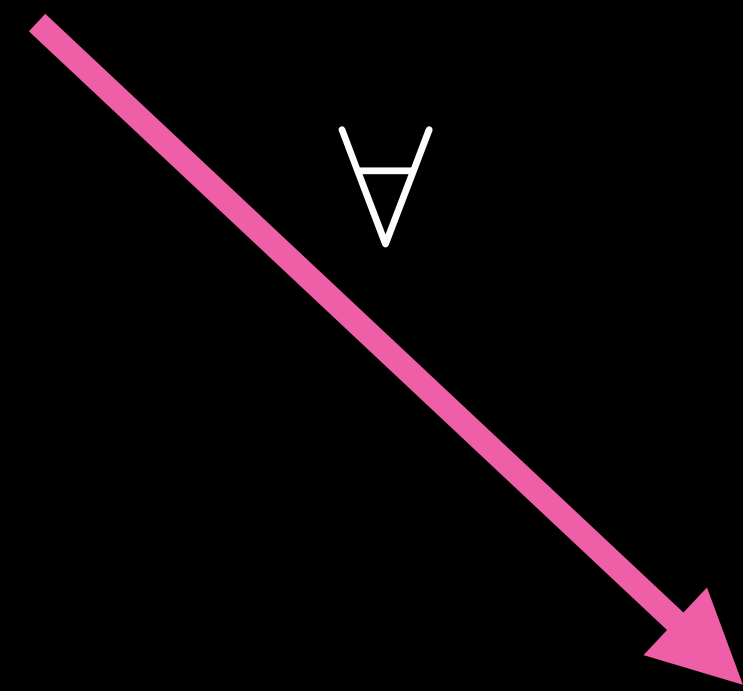
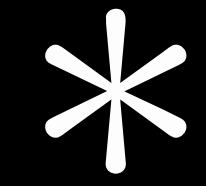
“types”



“expressions”

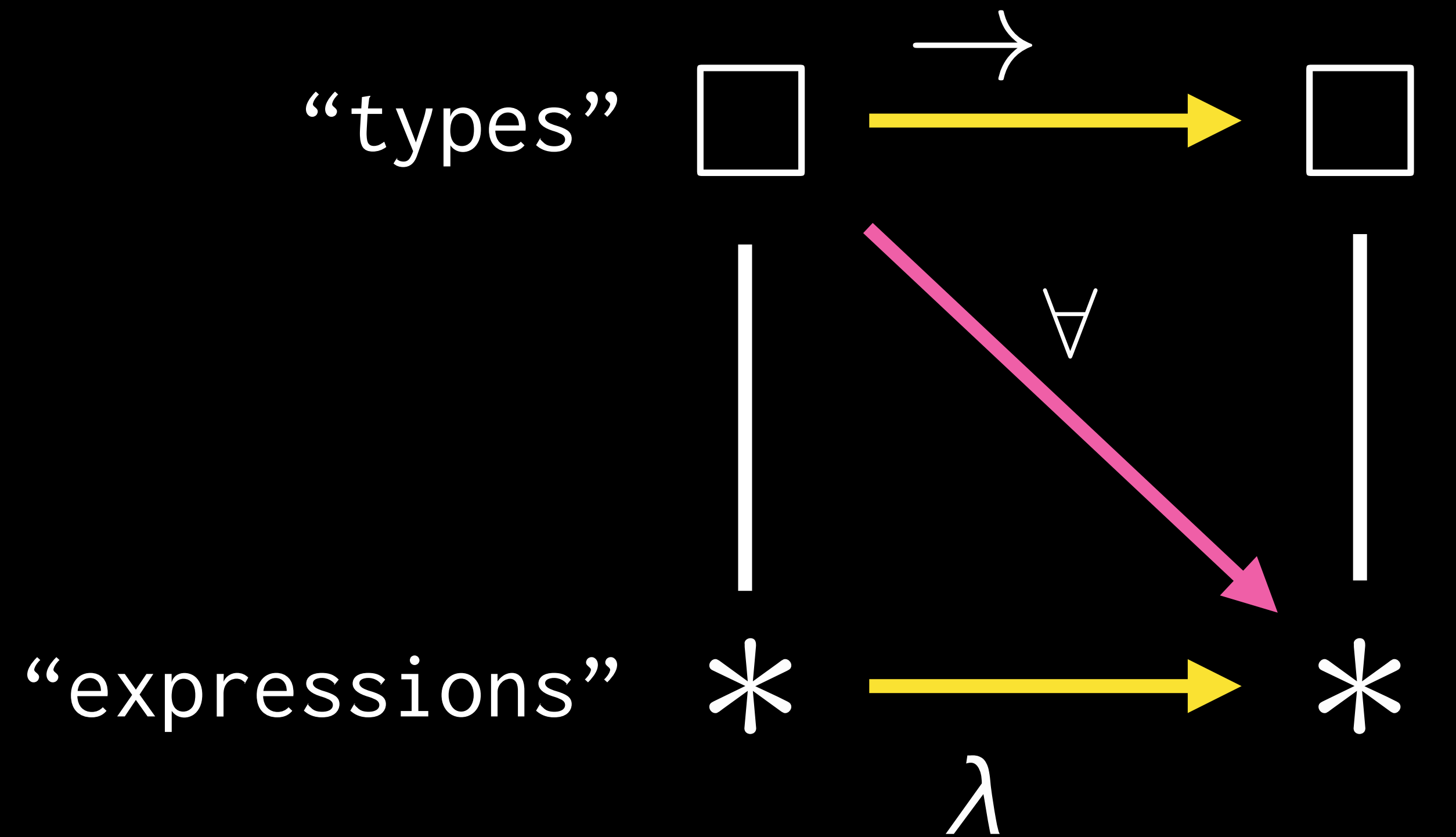


λ



\forall

system F



system F_ω

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

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

Type : Type

“naive type theory”

system U

$$\begin{aligned} & (\{*, \square, \triangle\}, \\ & \{(*, \square), (\square, \triangle)\}, \\ & \{(*, *), (\square, *), (\square, \square), (\triangle, \square), (\triangle, *)\}) \end{aligned}$$

system U

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

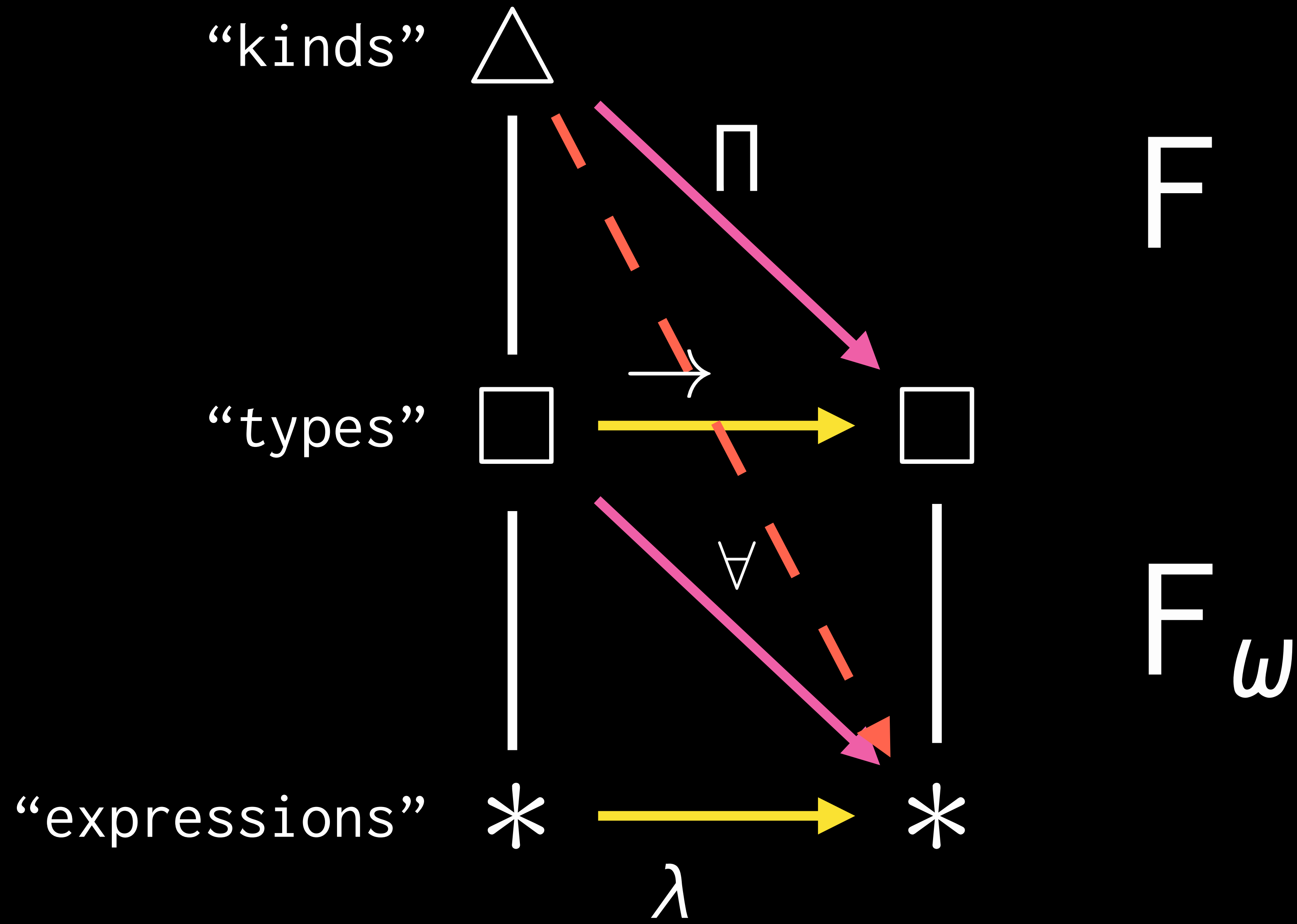
$\{(*, \square), (\square, \triangle)\},$

no cyclic dependency

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

system U^-

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



Girard (1972)

system U is **inconsistent!**

result known as *Girard's paradox*

basis: every type is inhabited, including \perp

Girard's paradox

analog of Russell's paradox

but not the same: no *type comprehension*

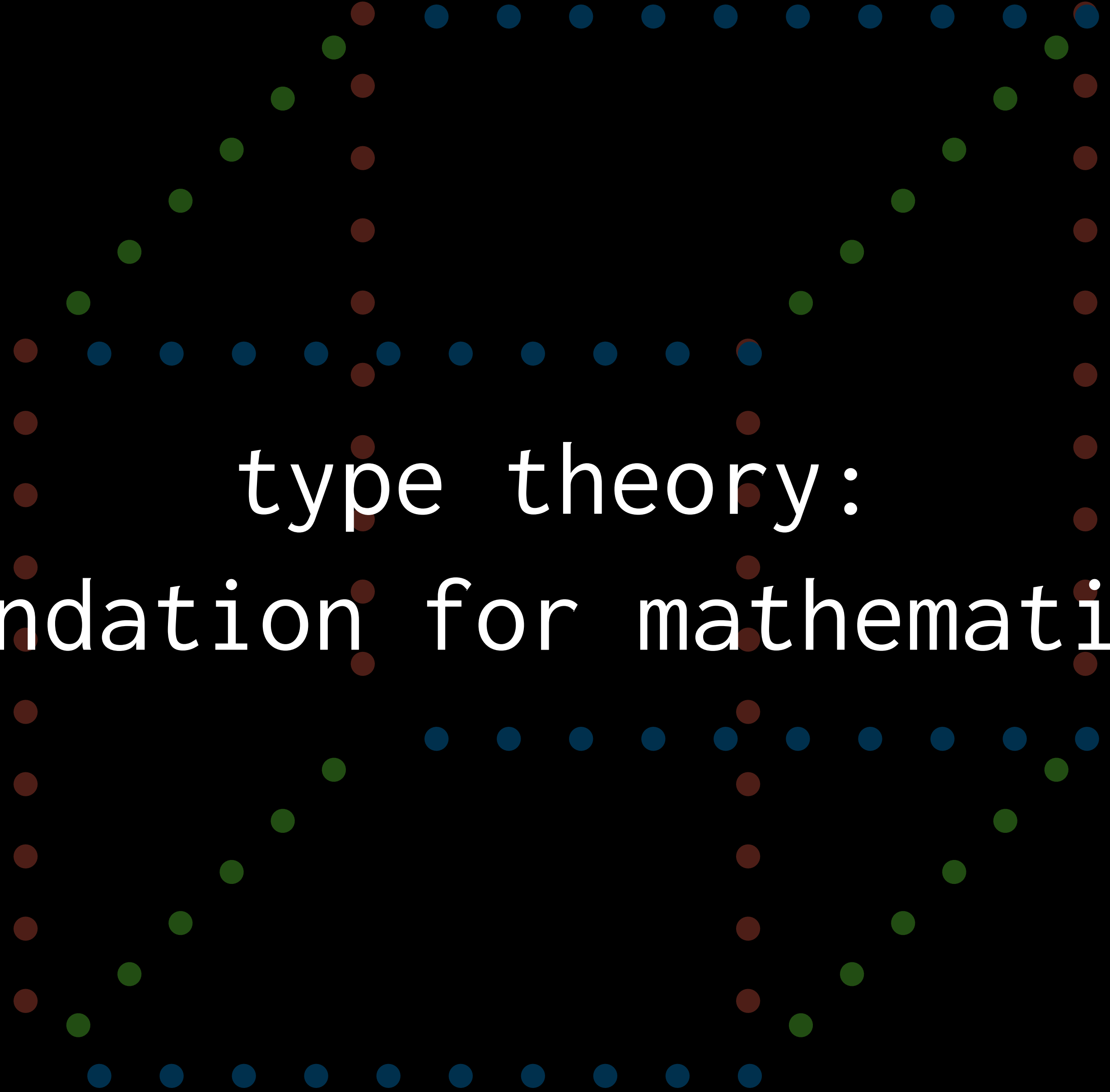
embedding of *Burali-Forti paradox* in U

corollary: naive type theory is inconsistent

corollary: Martin-Löf (1971) is inconsistent

Hurkens (1995): term of type \perp with length 2039

lambda cube strongly normalizing, not all PTSs are



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>