Game semantics for fragments of ML

Andrzej Murawski

University of Leicester

Nikos Tzevelekos

Queen Mary, London

What this talk is about

What is game semantics for programs
generating new/fresh resources ?

(e.g. references, exceptions, channels, etc.)

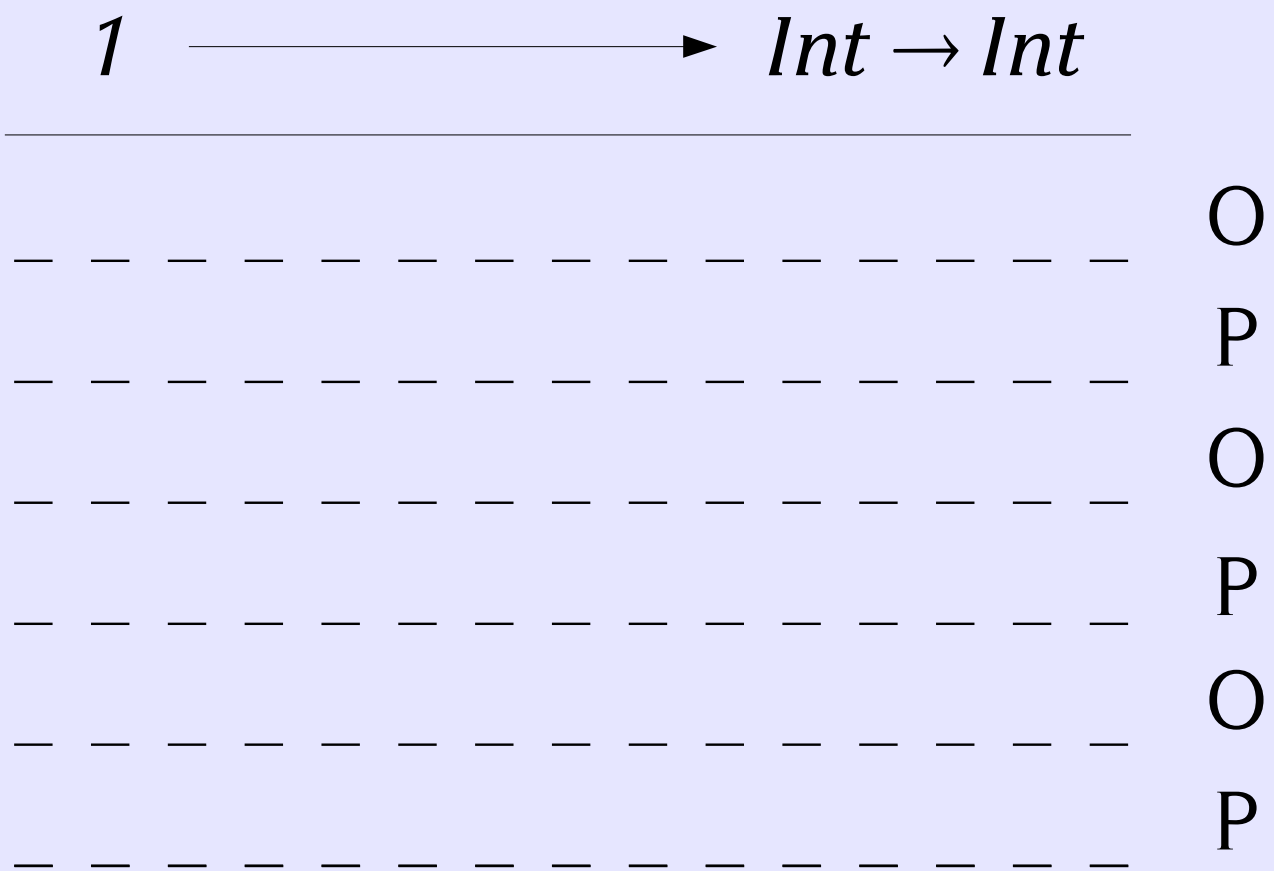
Model resources as names,
obtain a taxonomy of nominal game models

Game Semantics

- Computation is modelled as a 2-player game between:
  - *Proponent (the program)*
  - *Opponent* (the environment)

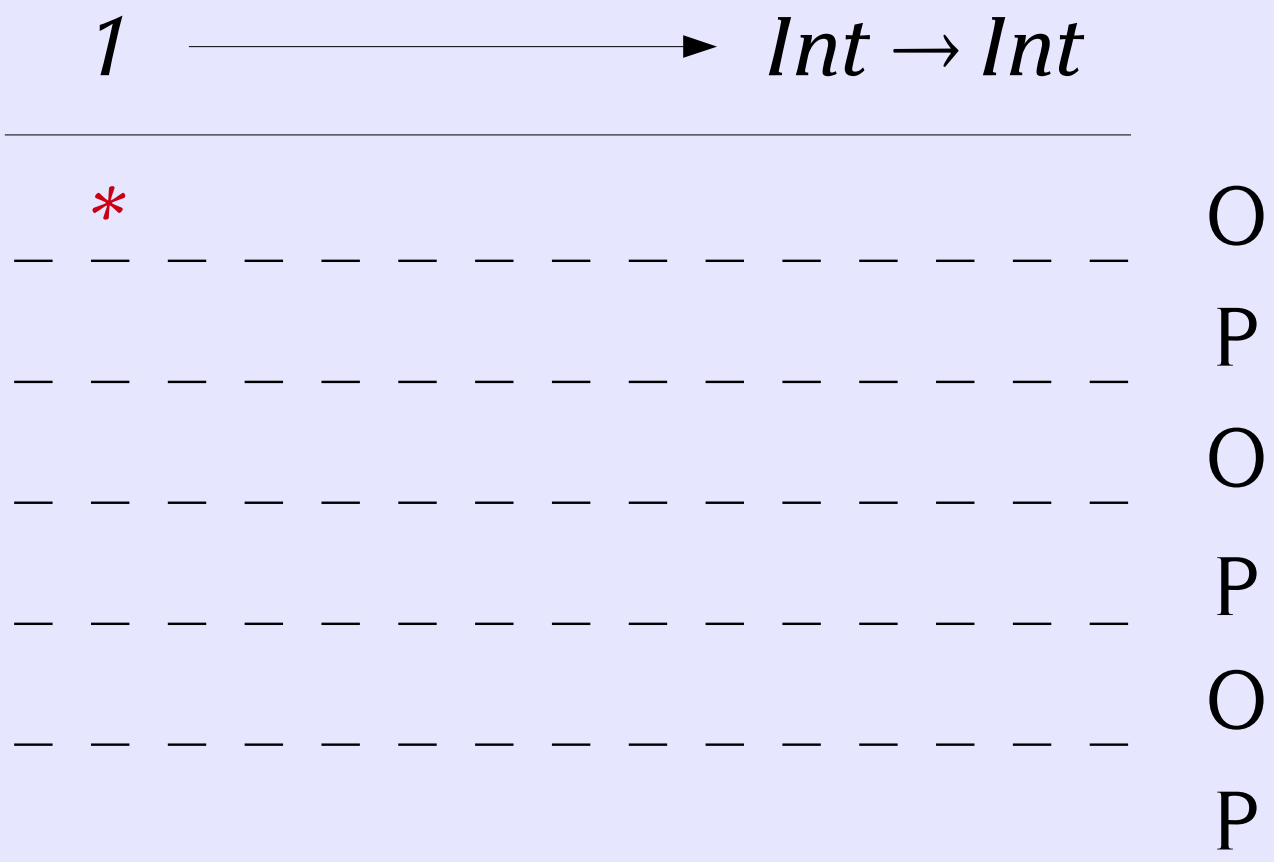
Example

$\lambda x.x+1 : \texttt{int} \to \texttt{int}$



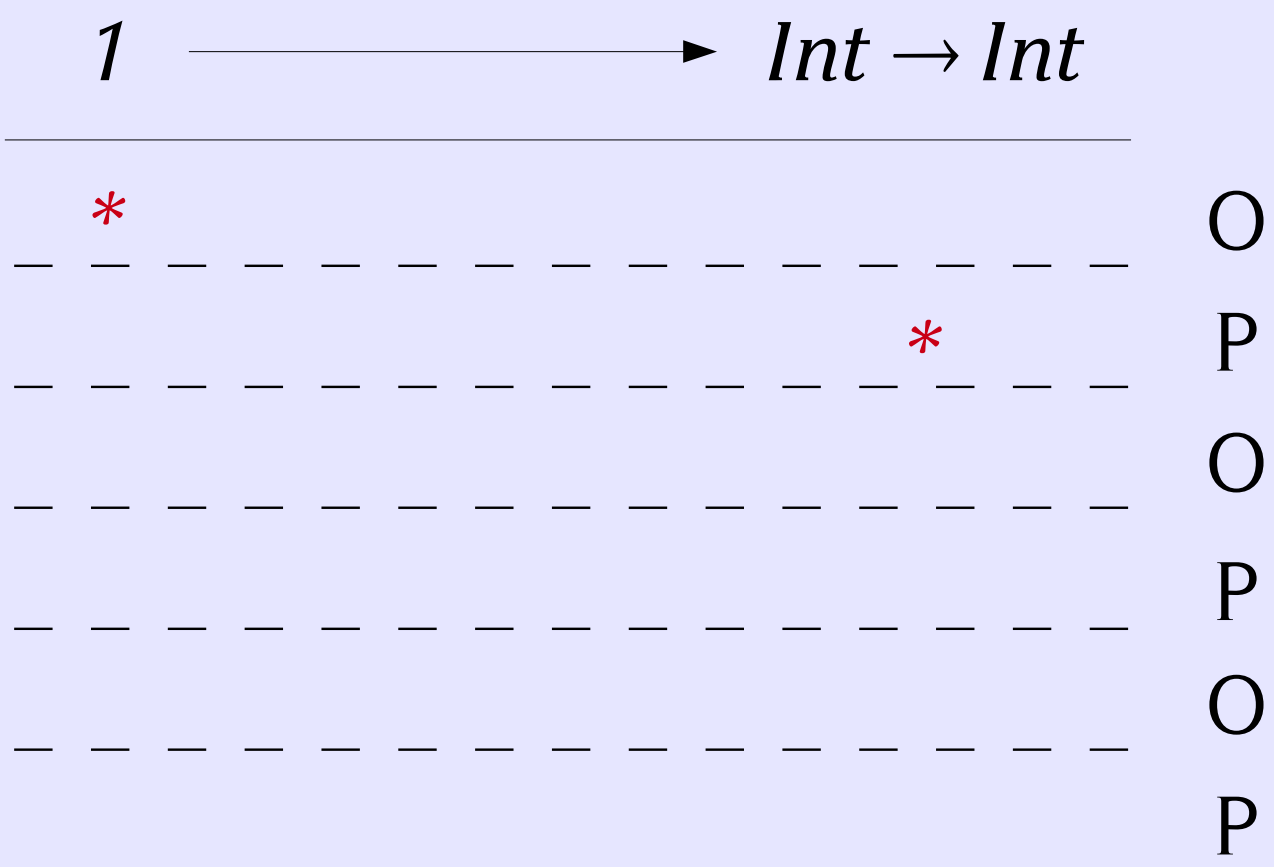
Example

$\lambda x.x+1 : \texttt{int} \to \texttt{int}$



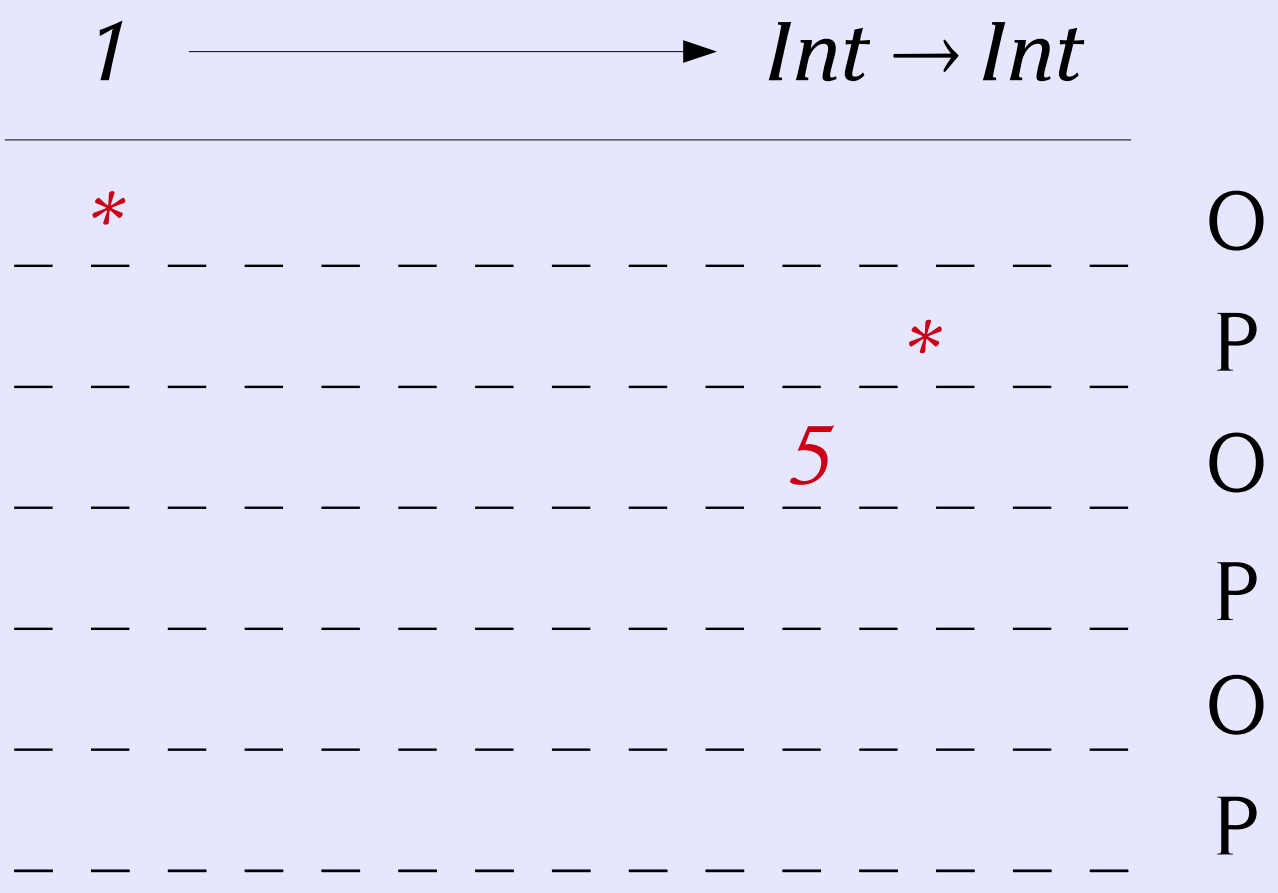
Example

$\lambda x.x+1 : \texttt{int} \to \texttt{int}$



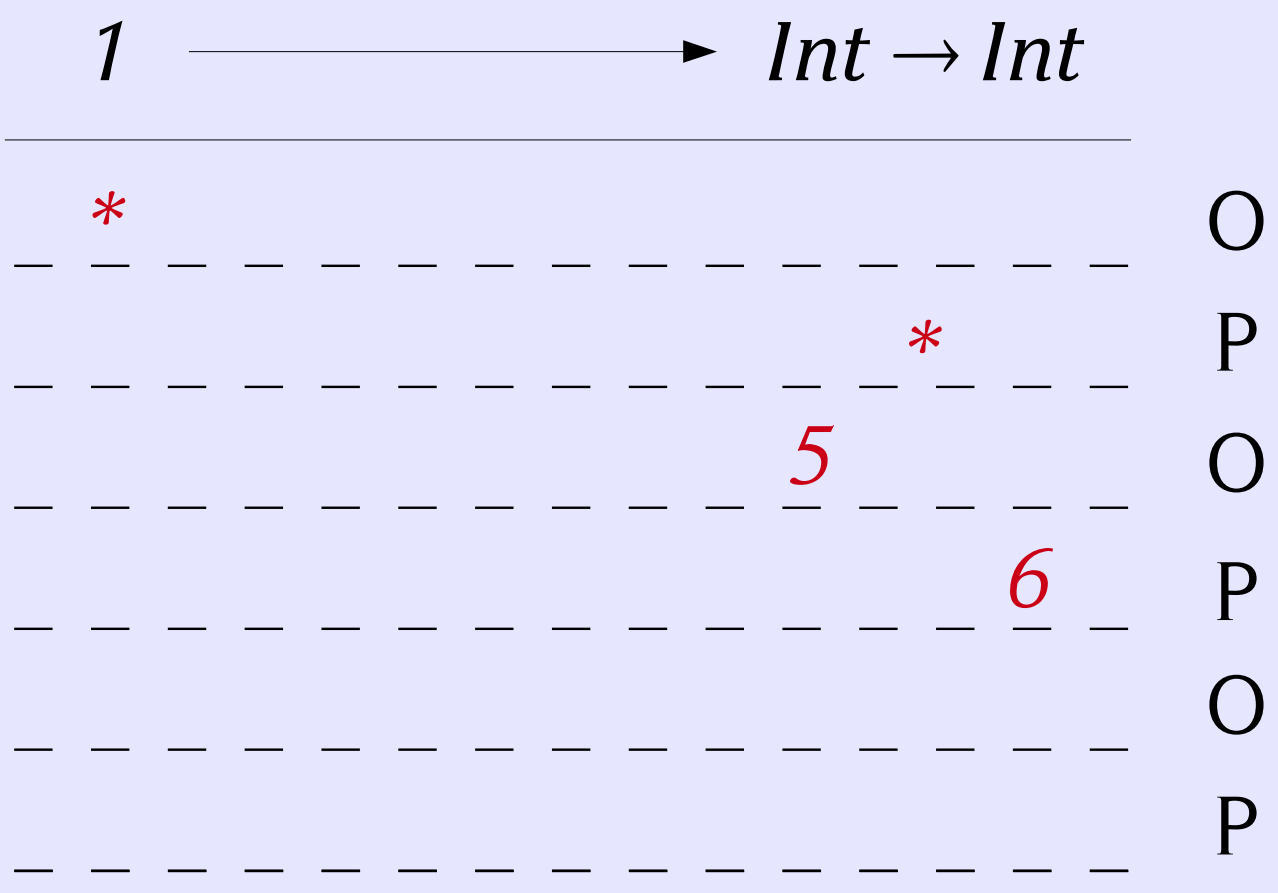
Example

$\lambda x.x+1 : \texttt{int} \to \texttt{int}$



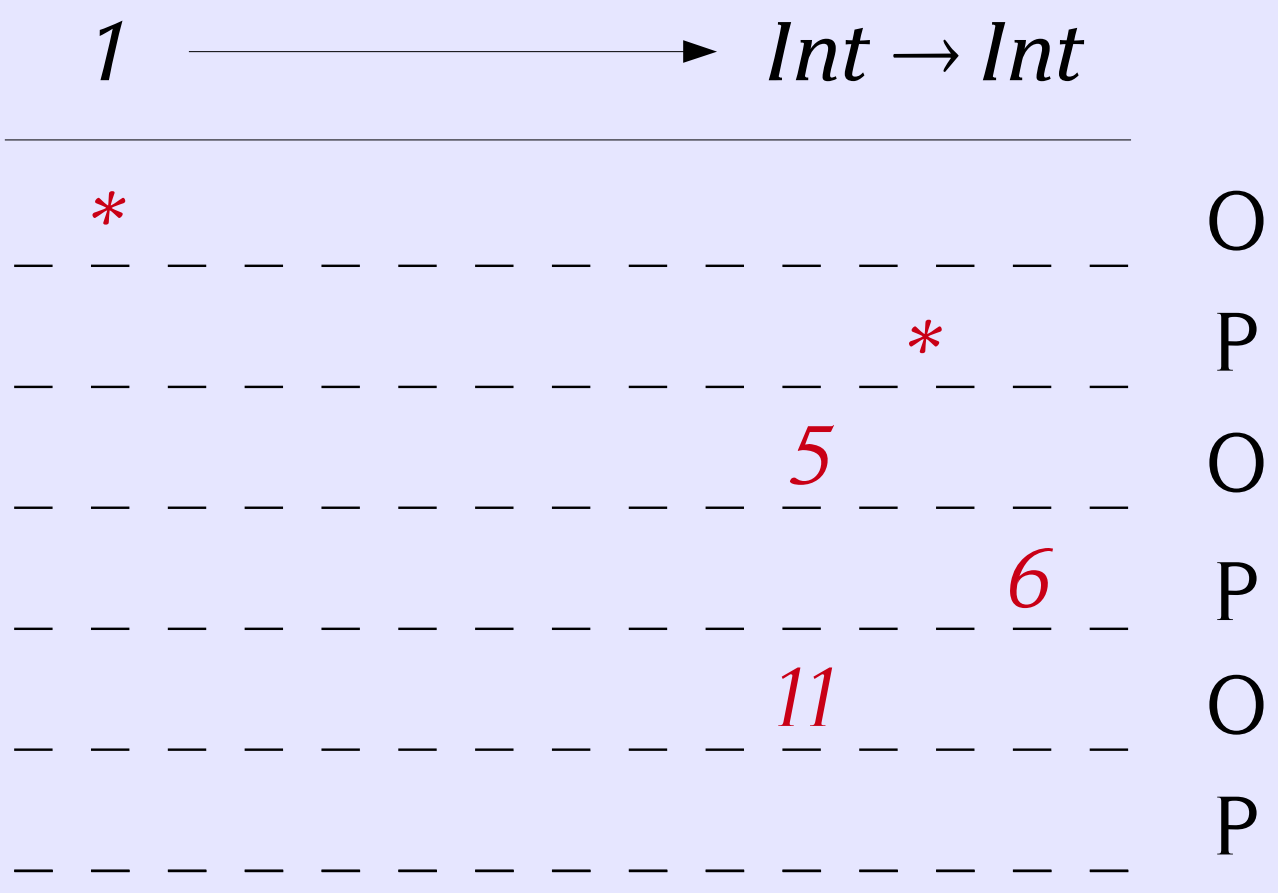
Example

$\lambda x.x+1 : \texttt{int} \to \texttt{int}$



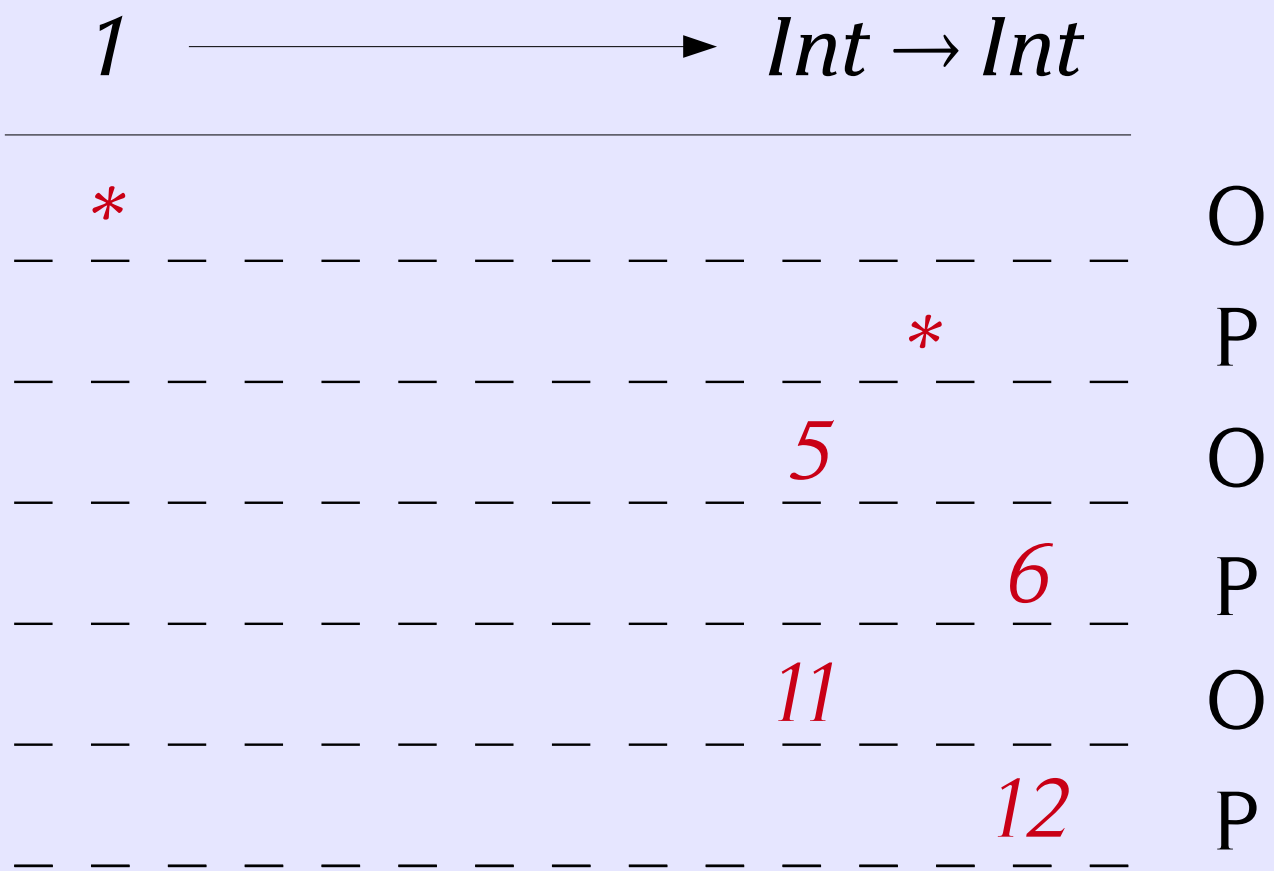
Example

$\lambda x.x+1 : \texttt{int} \to \texttt{int}$



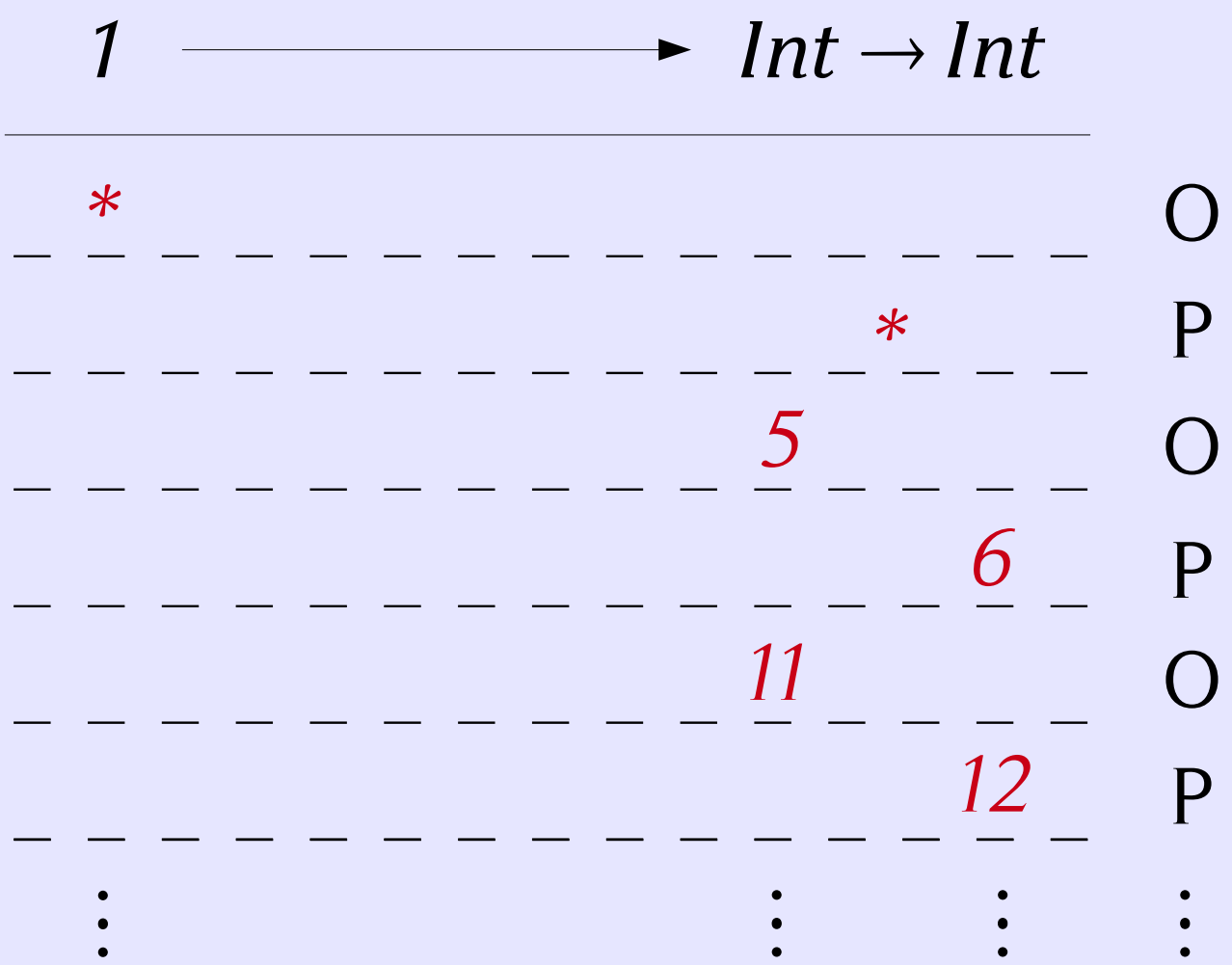
Example

$\lambda x.x{+}1 : \texttt{int} \to \texttt{int}$



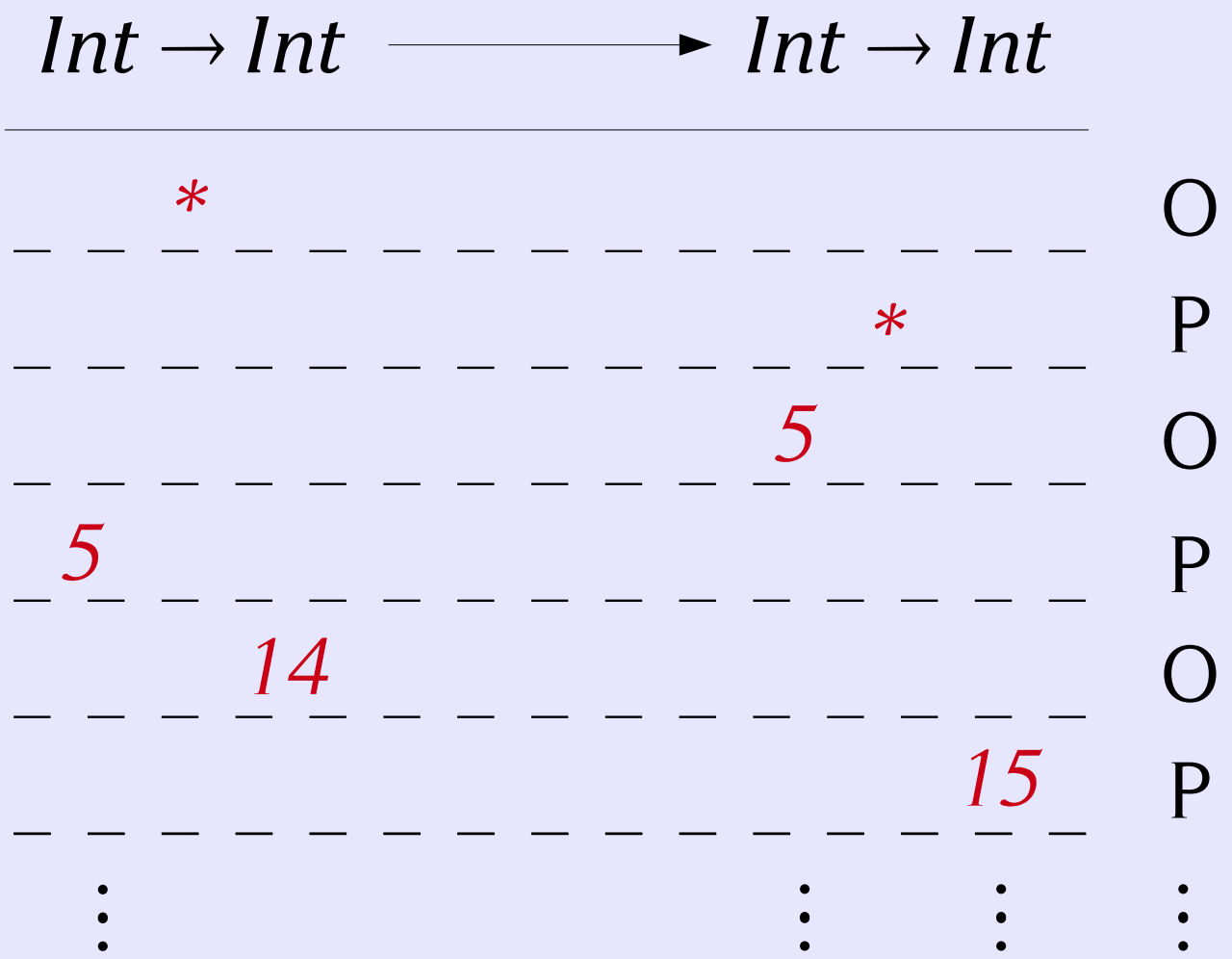
Example

$\lambda x.x+1 : \texttt{int} \to \texttt{int}$



Example

$$\texttt{f : int} \to \texttt{int} \vdash \texttt{λx.f(x)+1 : int} \to \texttt{int}$$

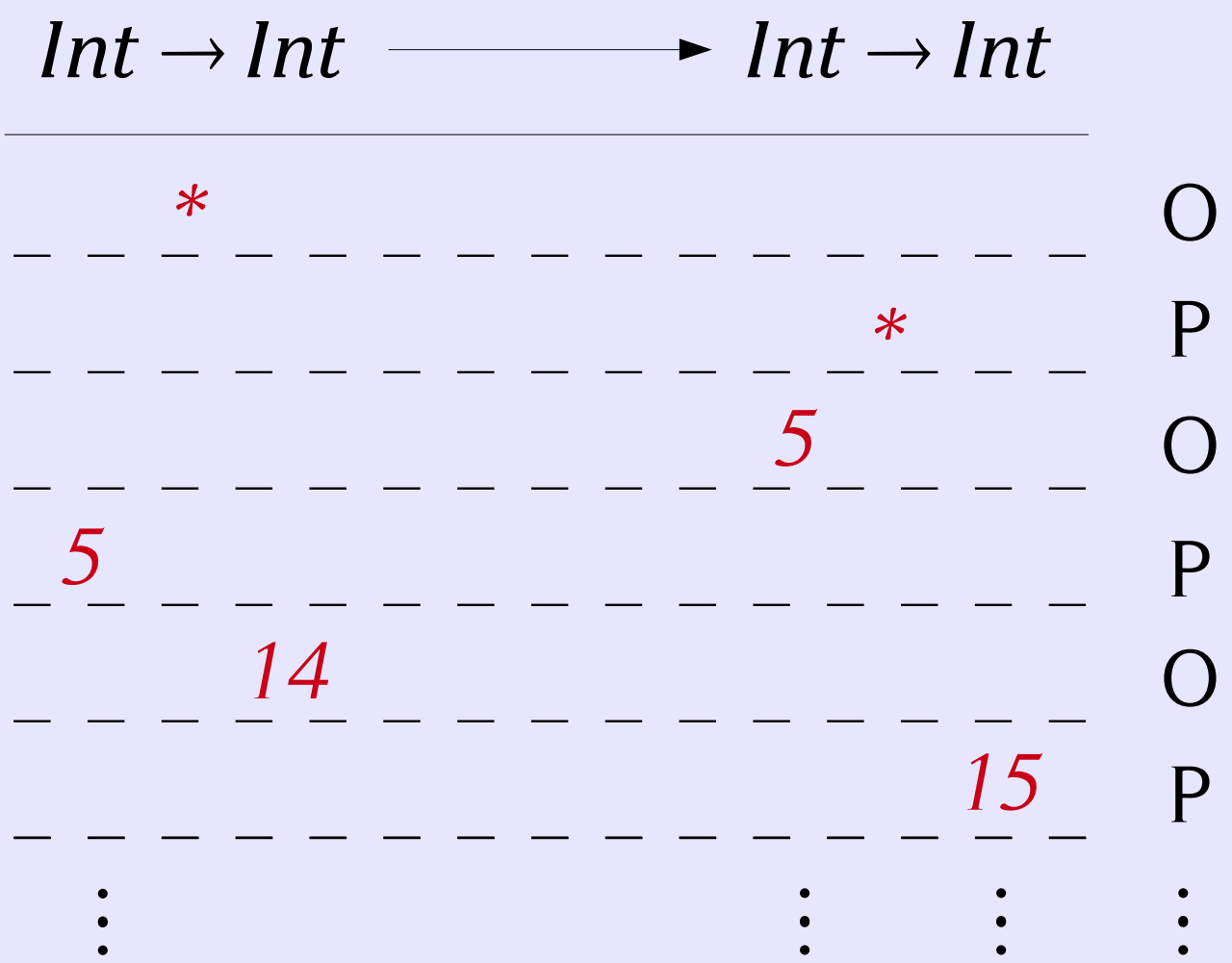


Game Semantics

- Computation is modelled as a 2-player game between:
  - *Opponent* (the environment)
  - *Proponent* (the program)
- Qualitative games (≠ Game Theory)
- Programs = *strategies* for Proponent
- Families (i.e. *categories*) of games

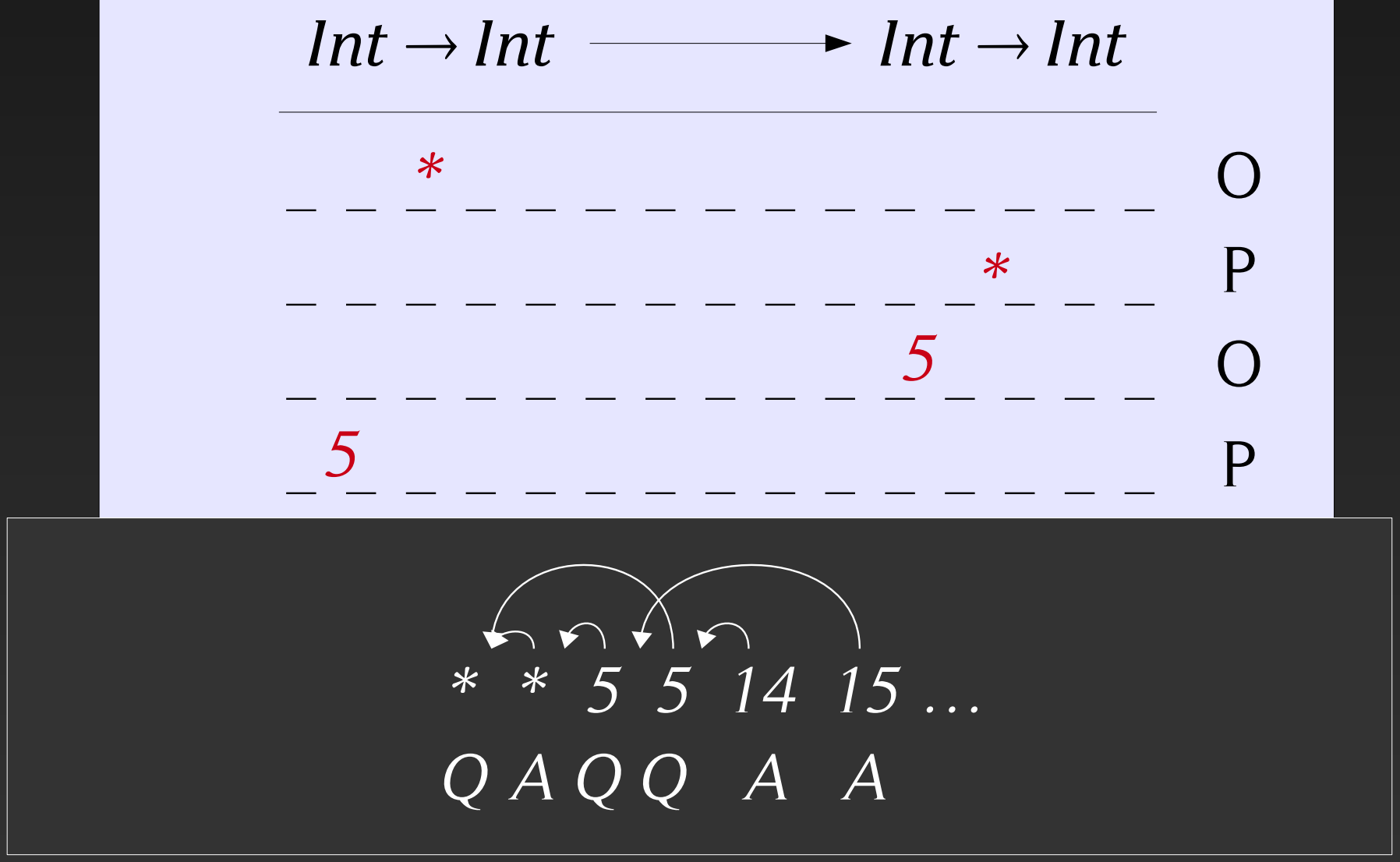
Pointers

$$\texttt{f : int} \to \texttt{int} \vdash \lambda\texttt{x.f(x)+1 : int} \to \texttt{int}$$

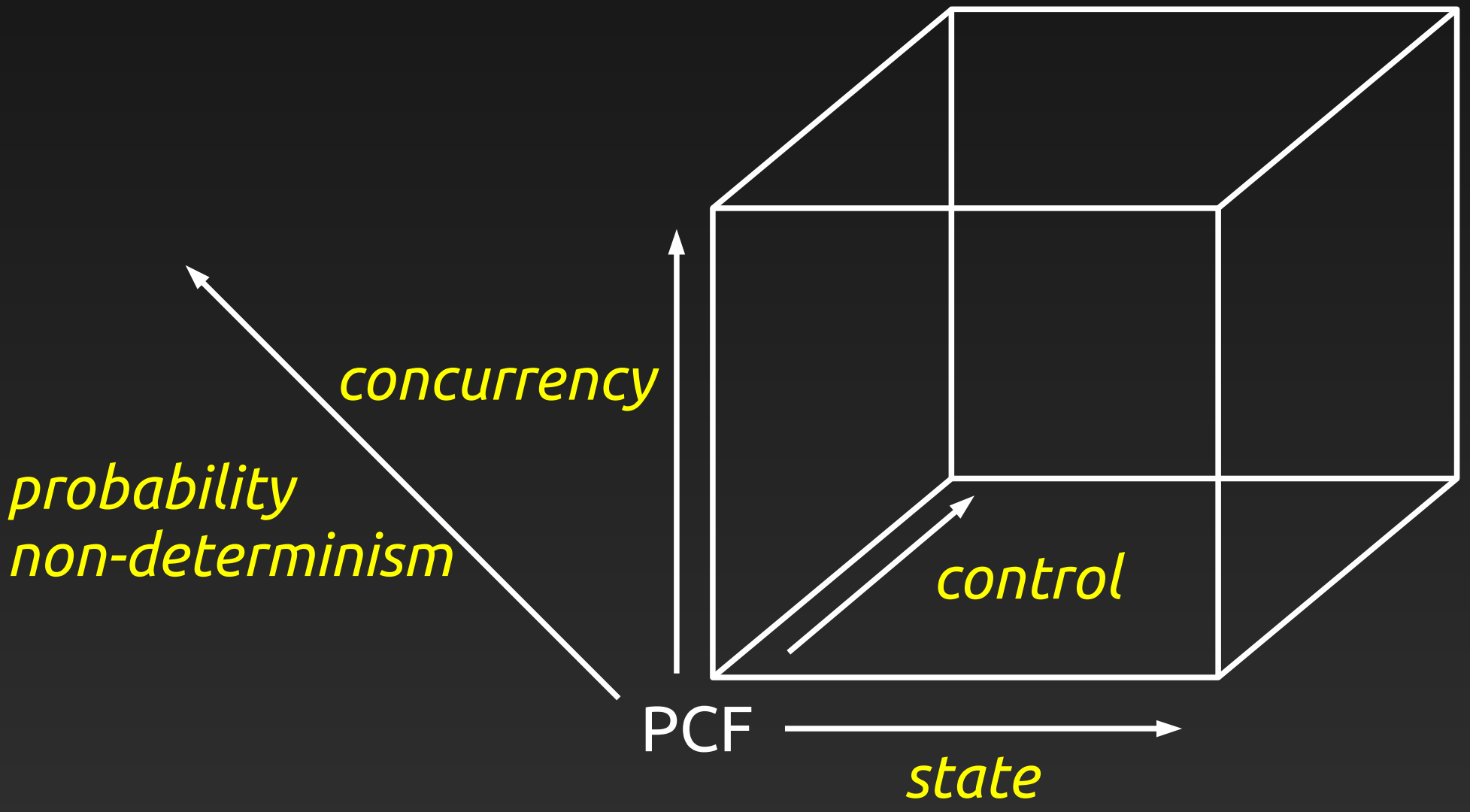


Pointers

$$f : \texttt{int} \to \texttt{int} \vdash \lambda x.f(x)+1 : \texttt{int} \to \texttt{int}$$



Abramsky's cube (90's)



move at each axis: relax constraints

Two ways to model references

Reynolds

- *Idealized Algol (1978)*

References are *pairs:*

$$\texttt{intref} = (\texttt{com} \to \texttt{int}) \times (\texttt{int} \to \texttt{com})$$

$$\longmapsto (\mathbf{1} \to \mathbf{Z}) \times (\mathbf{Z} \to \mathbf{1})$$

Two ways to model references

Reynolds

- *Idealized Algol (1978)*

References are *pairs:*

$$\texttt{intref} = \texttt{(com} \to \texttt{int)x(int} \to \texttt{com)}$$

$$\longmapsto \texttt{(1} \to \texttt{Z)x(Z} \to \texttt{1)}$$

- Theoretically attractive
- but: `mkvar(λx.3,λx.())` (*bad variables*)

Two ways to model references

Reynolds
- *Idealized Algol (1978)*

References are *pairs:*

$\texttt{intref} = (\texttt{com} \to \texttt{int}) \times (\texttt{int} \to \texttt{com})$

$\longmapsto (\mathbf{1} \to \mathbf{Z}) \times (\mathbf{Z} \to \mathbf{1})$

- Theoretically attractive
- but: `mkvar(λx.3,λx.())` (*bad variables*)

Pitts & Stark
- *nu-calculus (1993)*

References are *names:*

$\texttt{intref} = \texttt{base type}$

$\longmapsto \mathbf{N}$ (names)

Two ways to model references

Reynolds
- *Idealized Algol (1978)*

References are *pairs:*

$\texttt{intref} = (\texttt{com} \to \texttt{int}) \times (\texttt{int} \to \texttt{com})$

$\longmapsto (\mathbf{1} \to \mathbf{Z}) \times (\mathbf{Z} \to \mathbf{1})$

- Theoretically attractive
- but: `mkvar(λx.3,λx.())` (*bad variables*)

Pitts & Stark
- *nu-calculus (1993)*

References are *names:*

$\texttt{intref} = \texttt{base type}$

$\longmapsto \mathbf{N}$ (names)

- Notion of *resource (name):*
  - atomic values
  - infinitely many
  - comparable for equality

Two ways to model references exceptions channels …

Reynolds
- *Idealized Algol (1978)*

References are *pairs:*

$\texttt{intref} = (\texttt{com} \to \texttt{int}) \times (\texttt{int} \to \texttt{com})$

$\longmapsto (\mathbf{1} \to \mathbf{Z}) \times (\mathbf{Z} \to \mathbf{1})$

- Theoretically attractive
- but: $\texttt{mkvar}(\lambda x.3, \lambda x.())$ (*bad variables*)

Pitts
- *nu-*

References are *names:*

$\texttt{intref} = \texttt{base type}$

$\longmapsto \mathbf{N}$ (names)

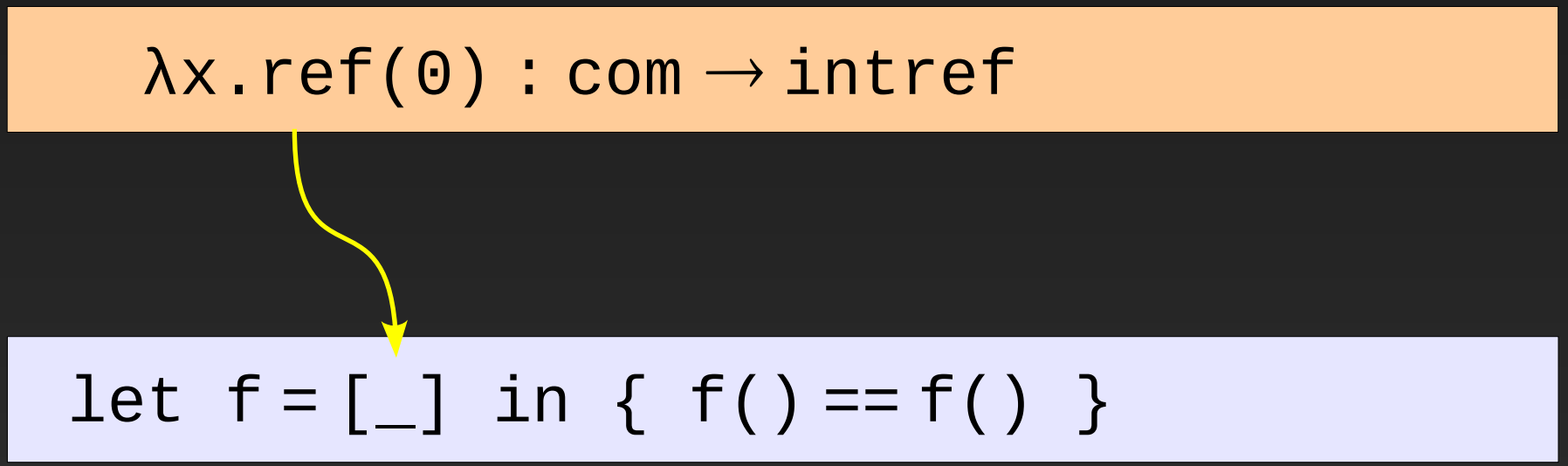
- Notion of *resource (name):*
  - atomic values
  - infinitely many
  - comparable for equality

Computation with names

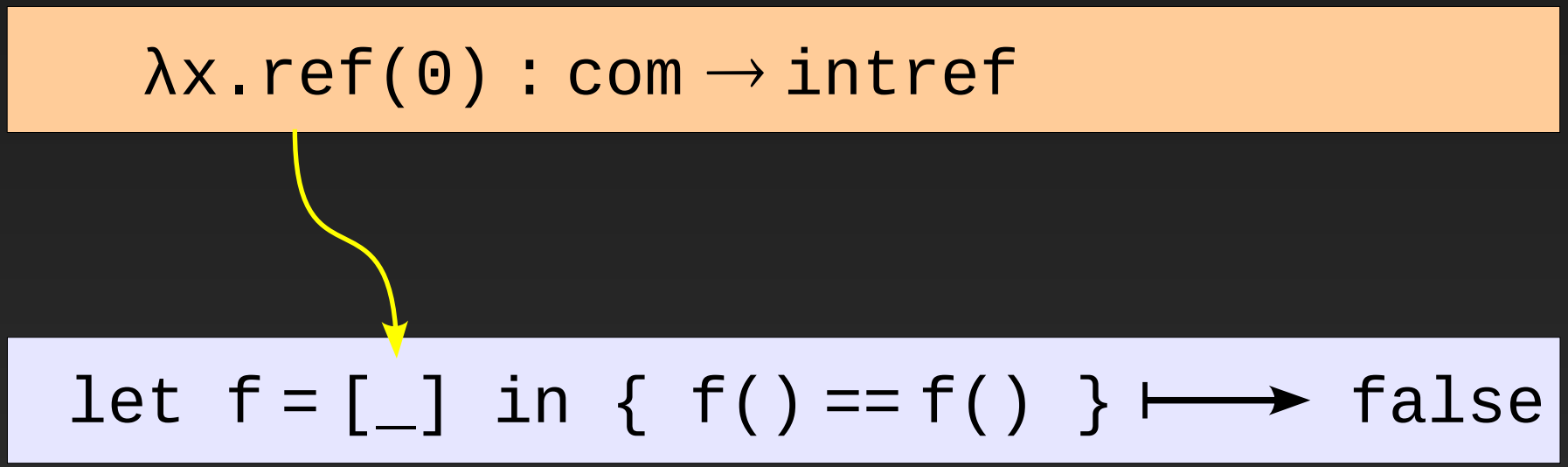
$\lambda x.\mathtt{ref}(0) : \mathtt{com} \to \mathtt{intref}$

```
let f = [_] in { f() == f() }
```

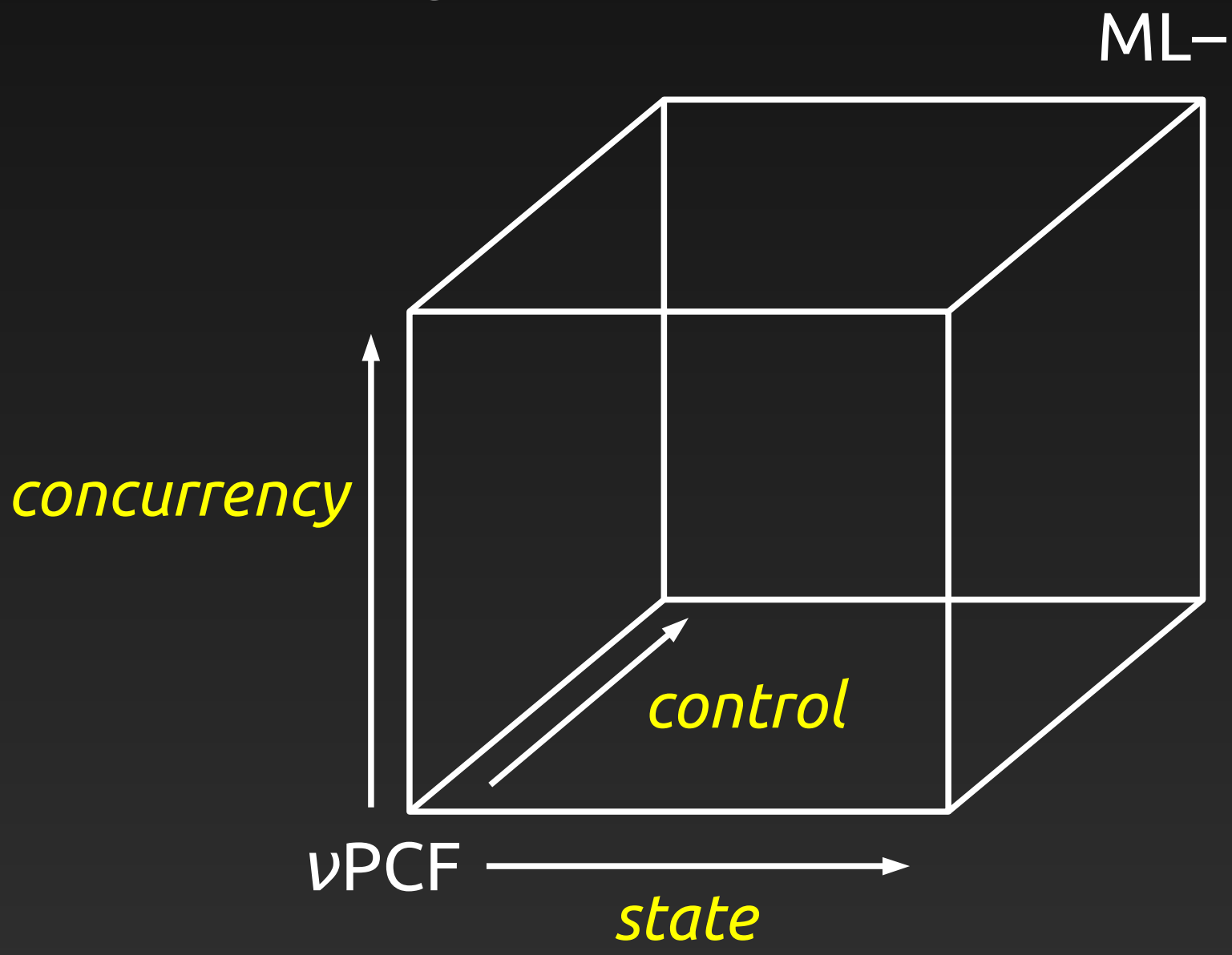
Computation with names



Computation with names



A nominal game cube



νPCF

- A ::= com | int | ref | A_f $\quad\quad$ A_f ::= $A \to A$

νPCF

- $A ::= \texttt{com} \mid \texttt{int} \mid \texttt{ref} \mid A_f \qquad A_f ::= A \to A$
- $\texttt{()}:\texttt{com},\ i:\texttt{int},\ \texttt{if}:\texttt{int}\to A\to A\to A,\ \texttt{Y}:(A_f\to A_f)\to A_f$

  …
- $\Gamma, x{:}A \vdash x{:}A \qquad \dfrac{\Gamma, x{:}A \vdash s{:}B}{\Gamma \vdash \lambda x.s{:}A\to B} \qquad \dfrac{\Gamma \vdash s{:}A\to B,\ t{:}A}{\Gamma \vdash st{:}B}$
- $\Gamma \vdash \texttt{ref()}:\texttt{ref} \qquad \dfrac{\Gamma \vdash s{:}\texttt{ref},\ t{:}\texttt{ref}}{\Gamma \vdash s == t : \texttt{int}}$

Nominal games

$\lambda x.\mathtt{ref}() : \mathtt{com} \to \mathtt{ref}$

```
let f = [_] in { f() == f() }
```

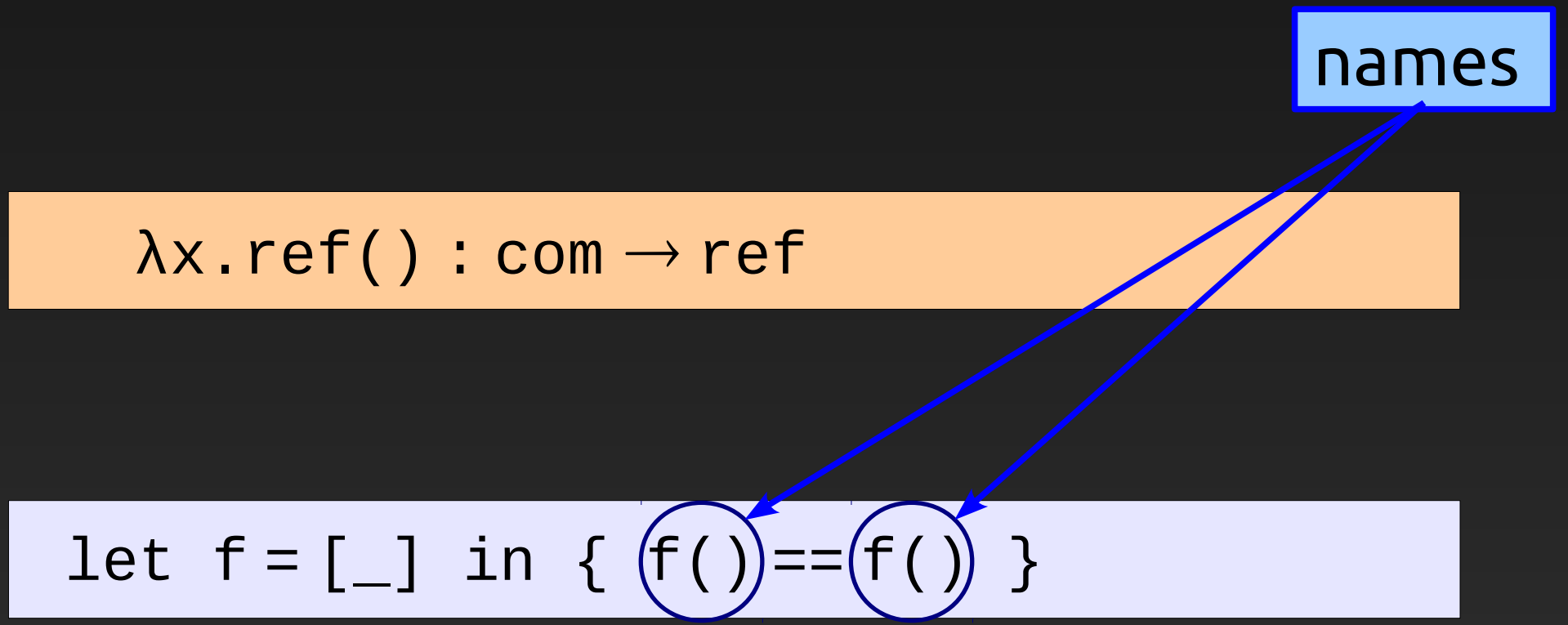
Nominal games

$\lambda x.\mathtt{ref}() : \mathtt{com} \to \mathtt{ref}$

```
let f = [_] in { f()==f() }
```

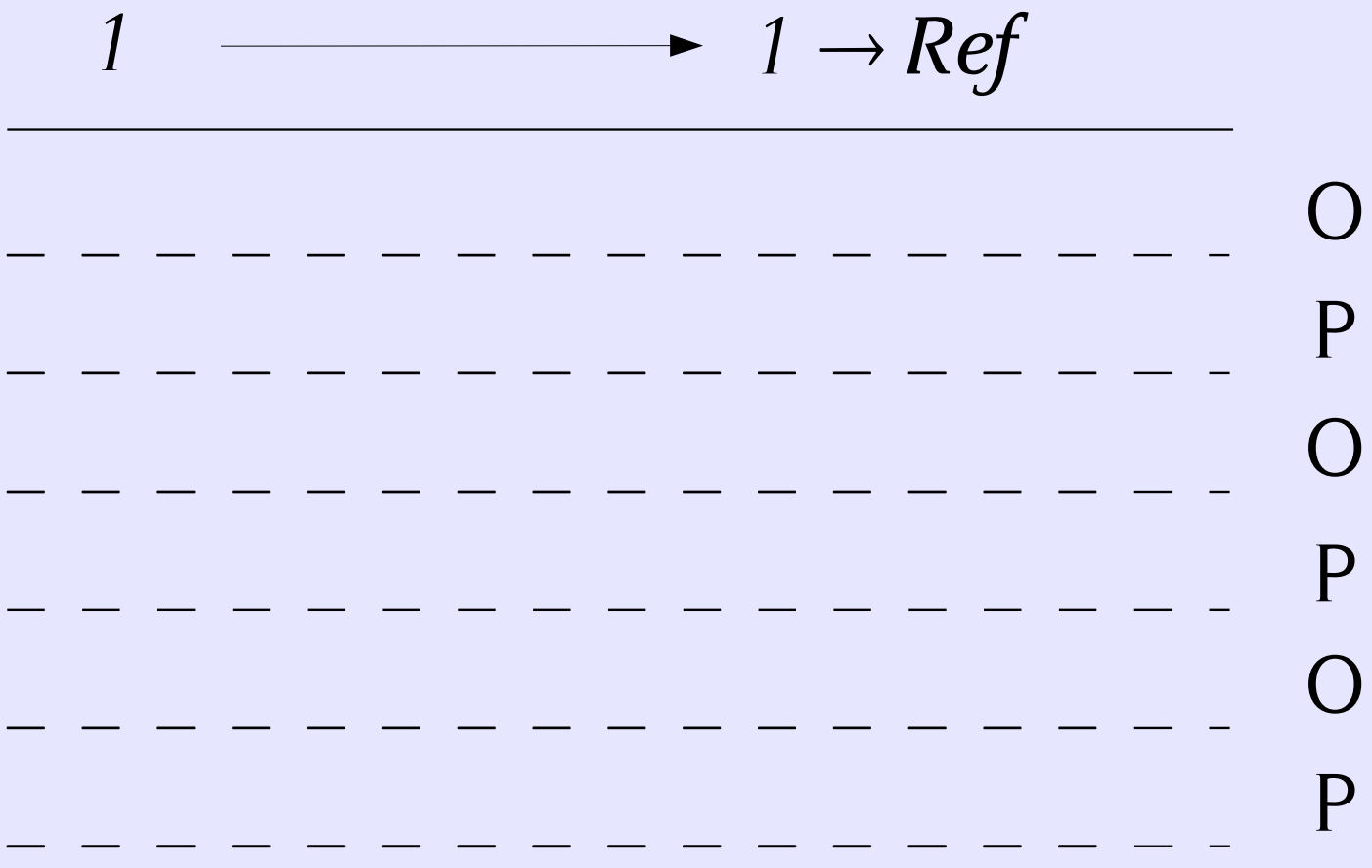
Nominal games

Games in nominal sets



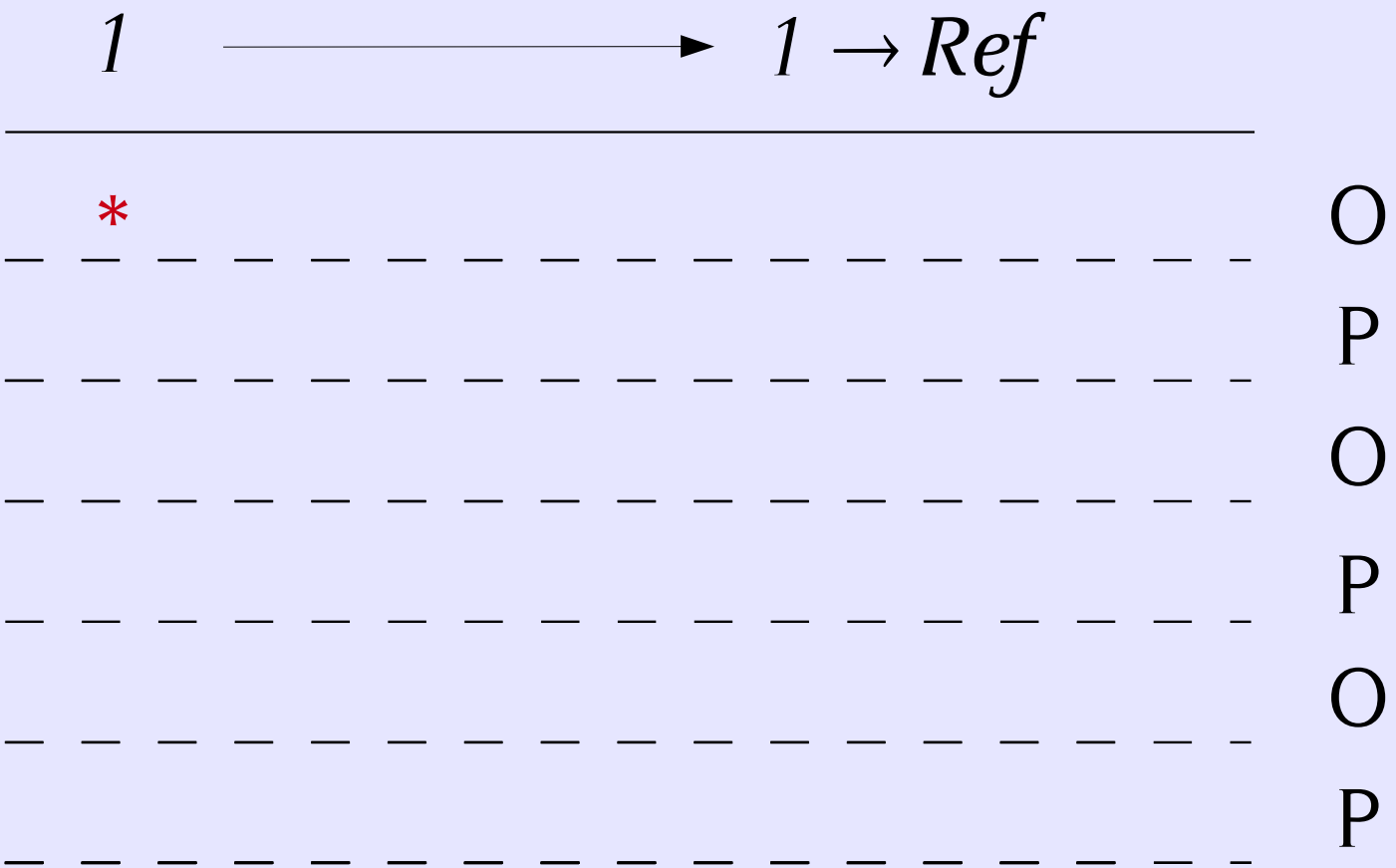
Examples

$\lambda x.\mathtt{ref}() : \mathtt{com} \to \mathtt{ref}$



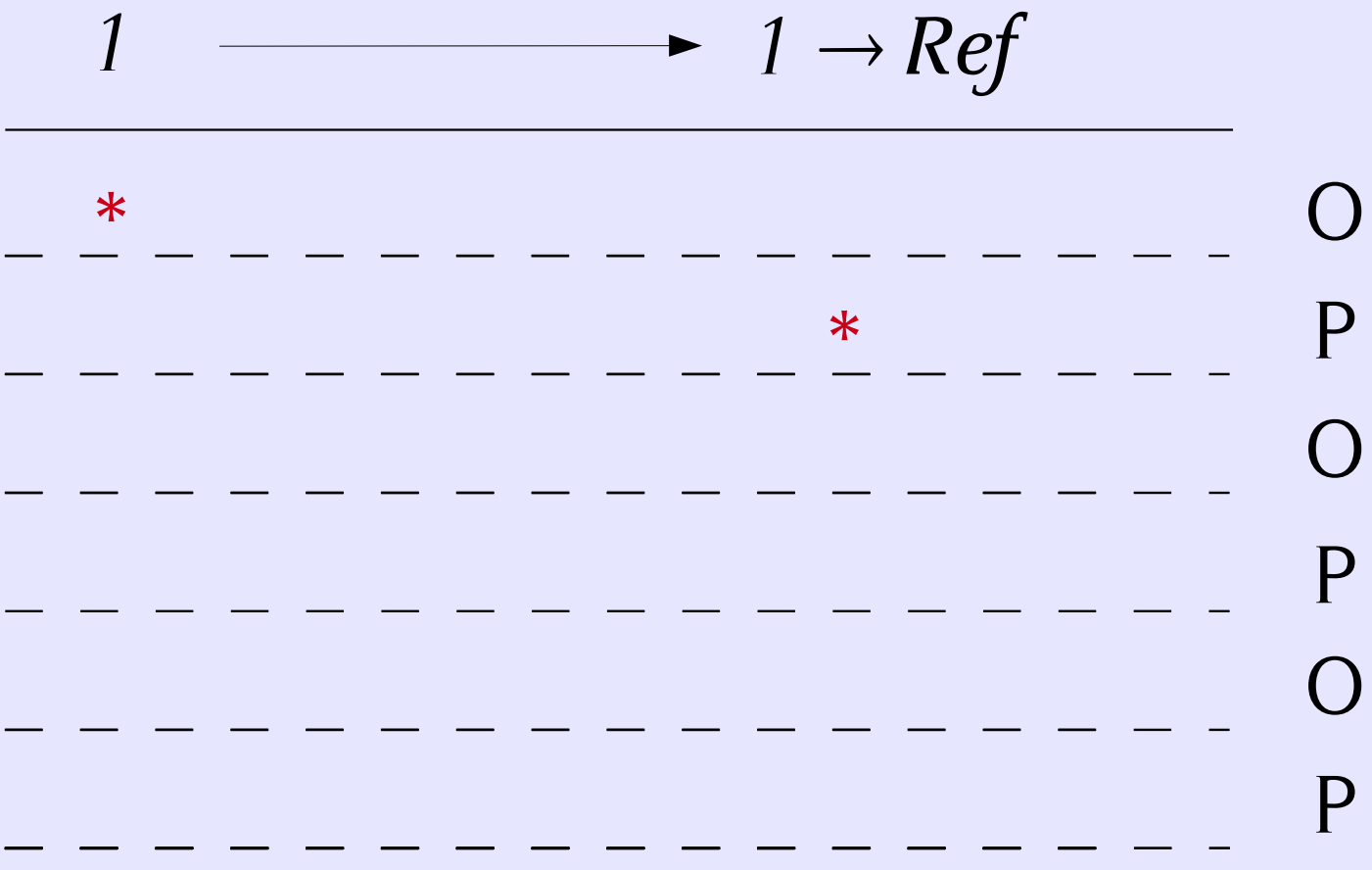
Examples

$\lambda x.\mathtt{ref}() : \mathtt{com} \to \mathtt{ref}$



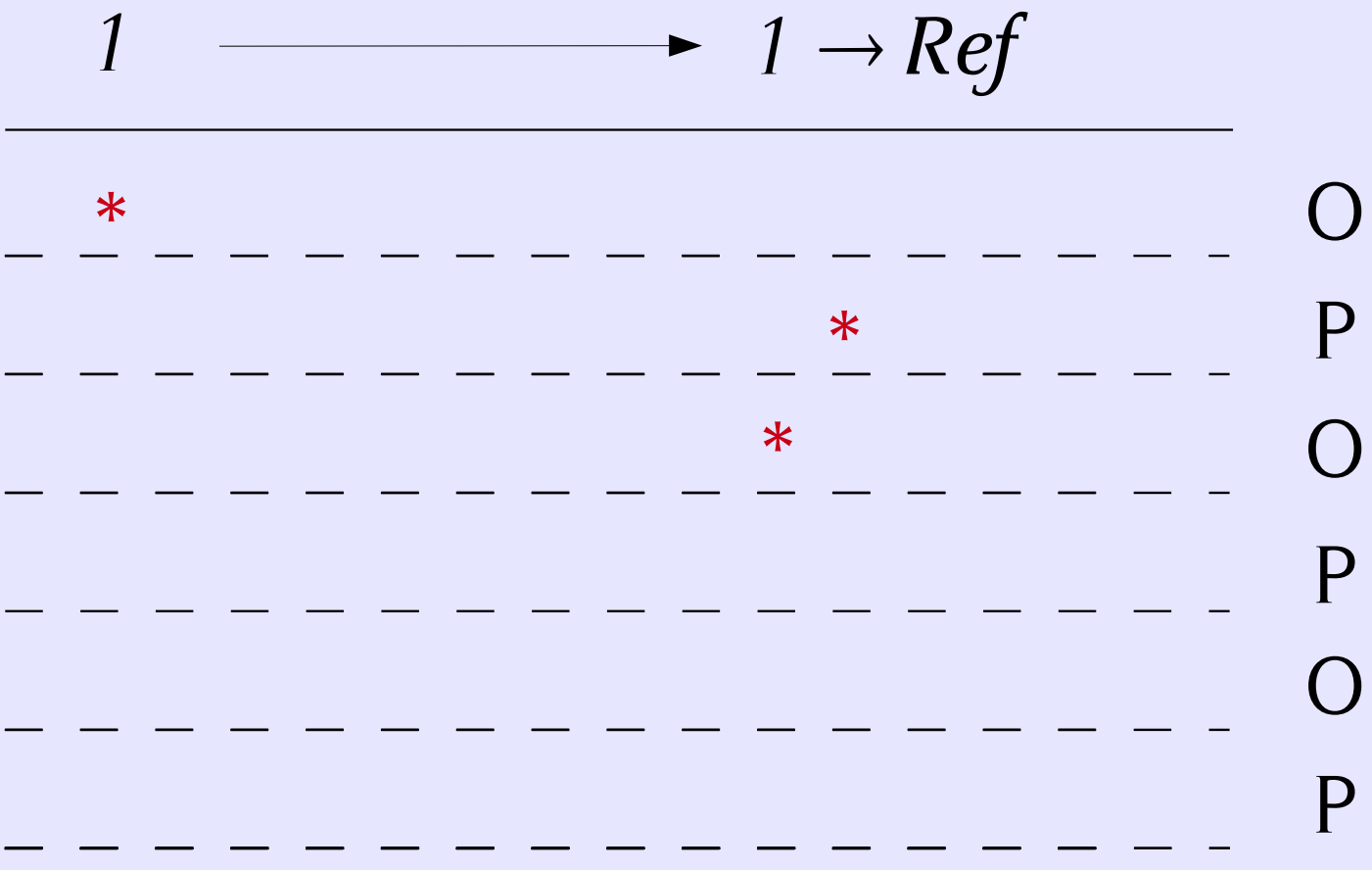
Examples

$\lambda x.\mathtt{ref}() : \mathtt{com} \to \mathtt{ref}$



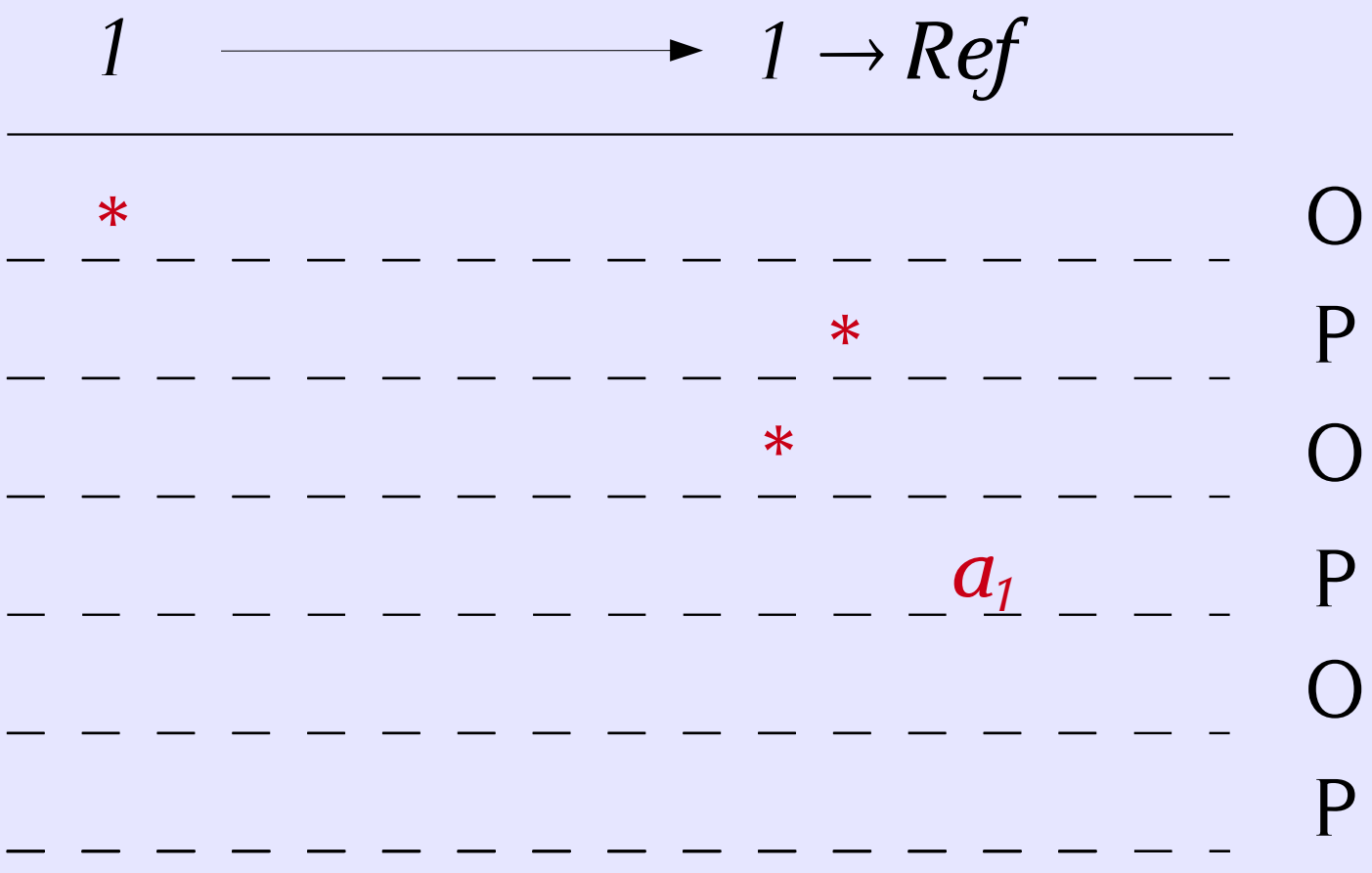
Examples

$\lambda x.\mathtt{ref}() : \mathtt{com} \to \mathtt{ref}$



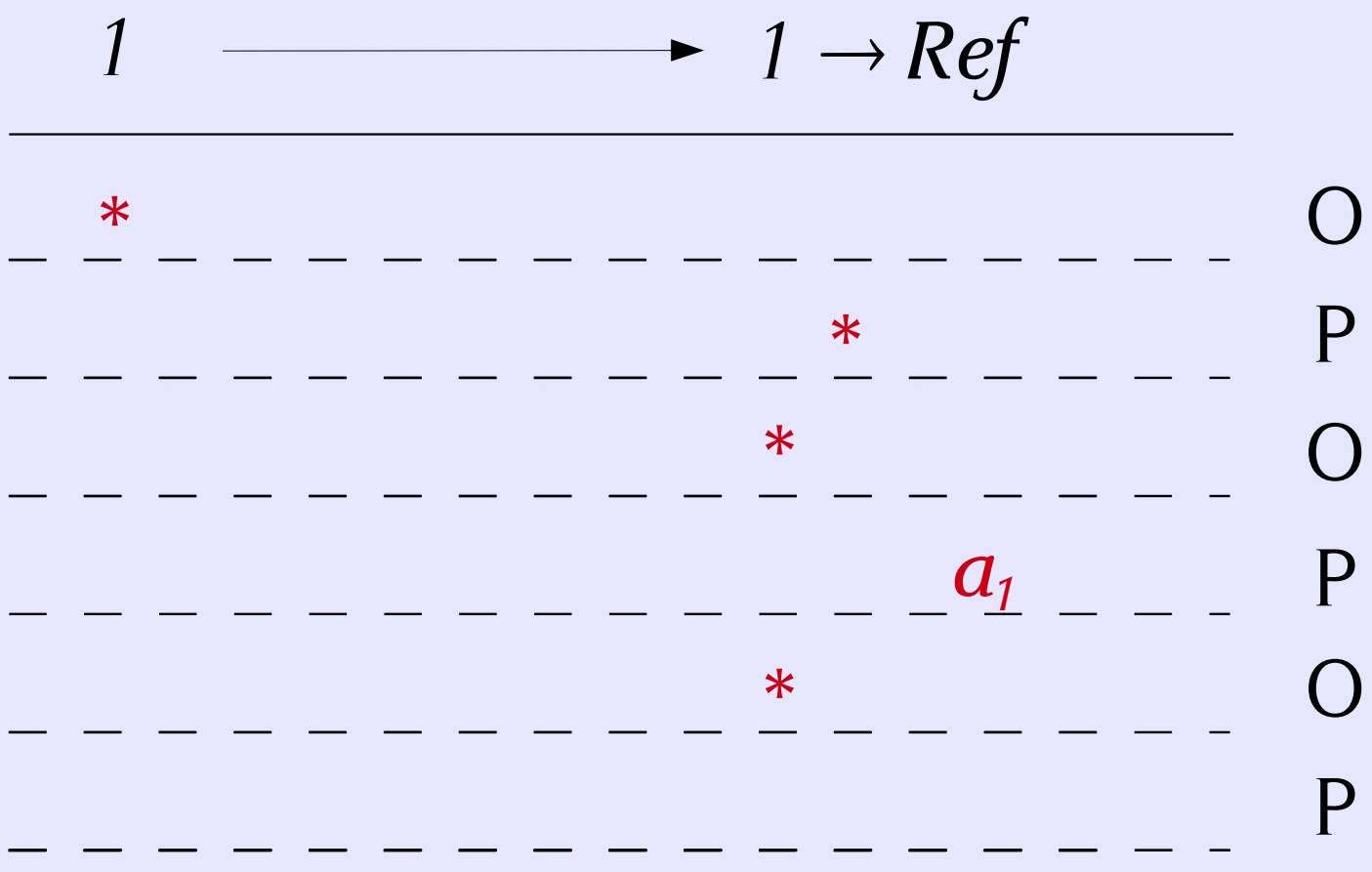
Examples

$\lambda x.\mathtt{ref}() : \mathtt{com} \to \mathtt{ref}$



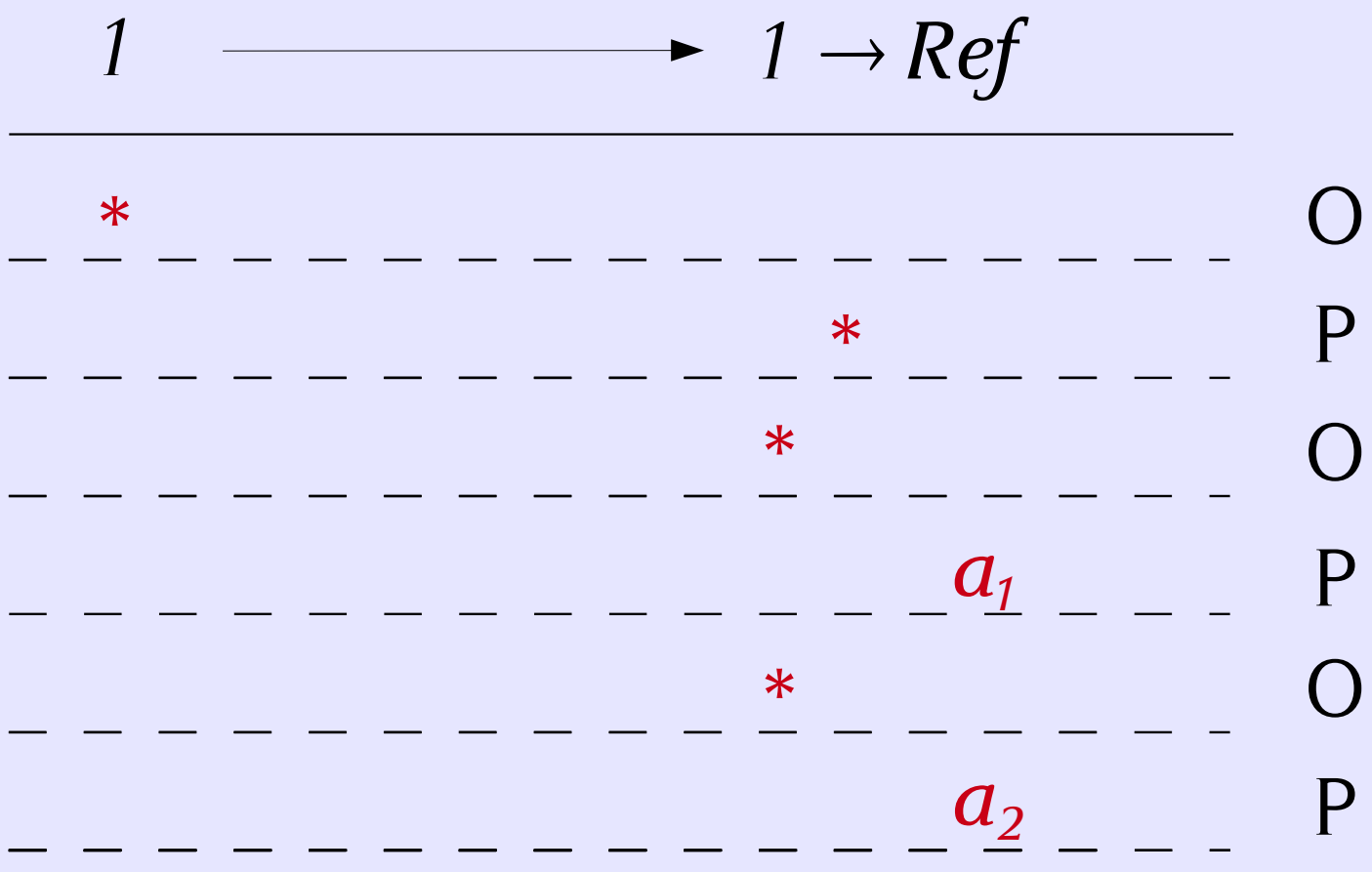
Examples

$\lambda x.\mathtt{ref}() : \mathtt{com} \to \mathtt{ref}$



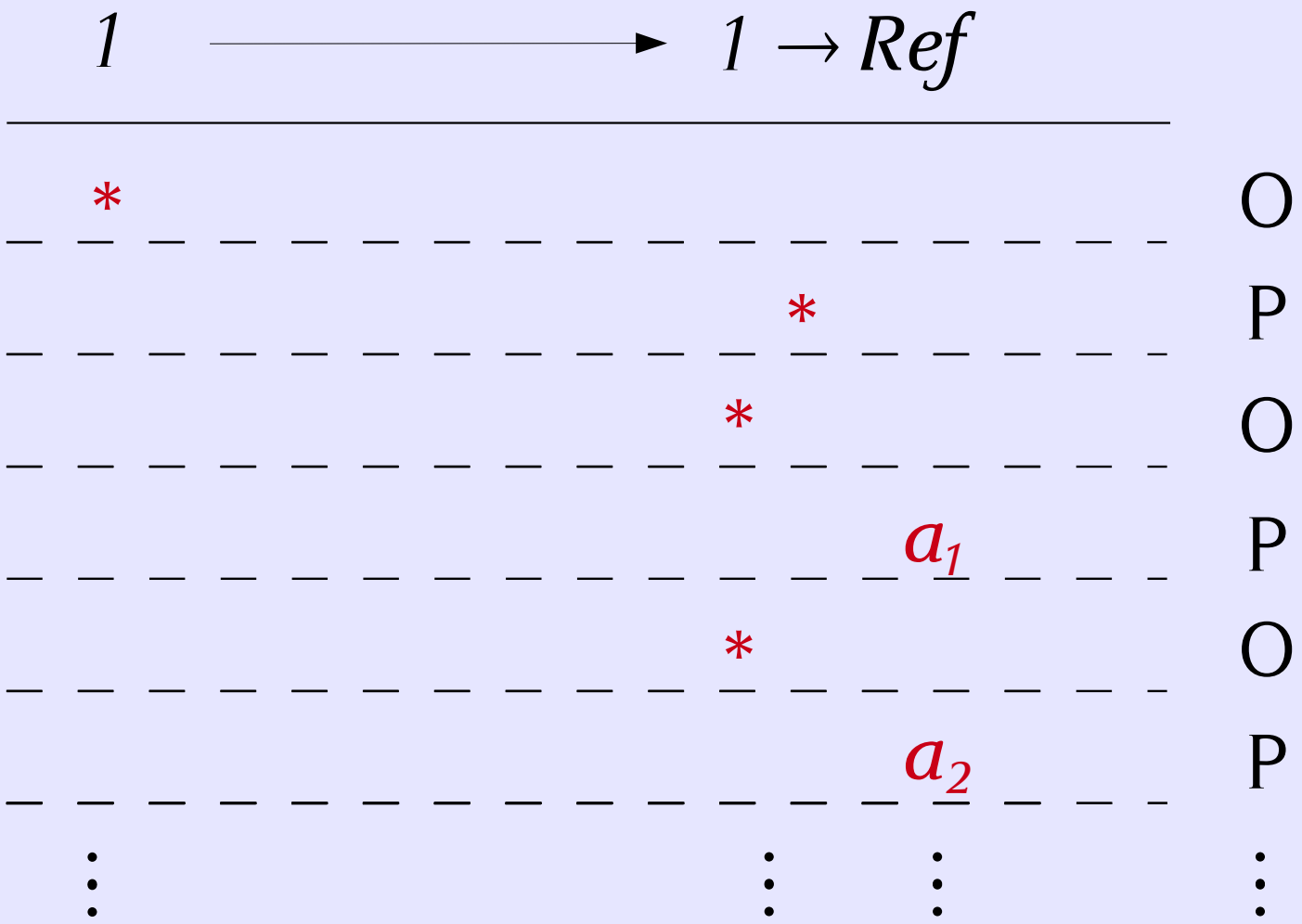
Examples

$\lambda x.\mathtt{ref}() : \mathtt{com} \to \mathtt{ref}$



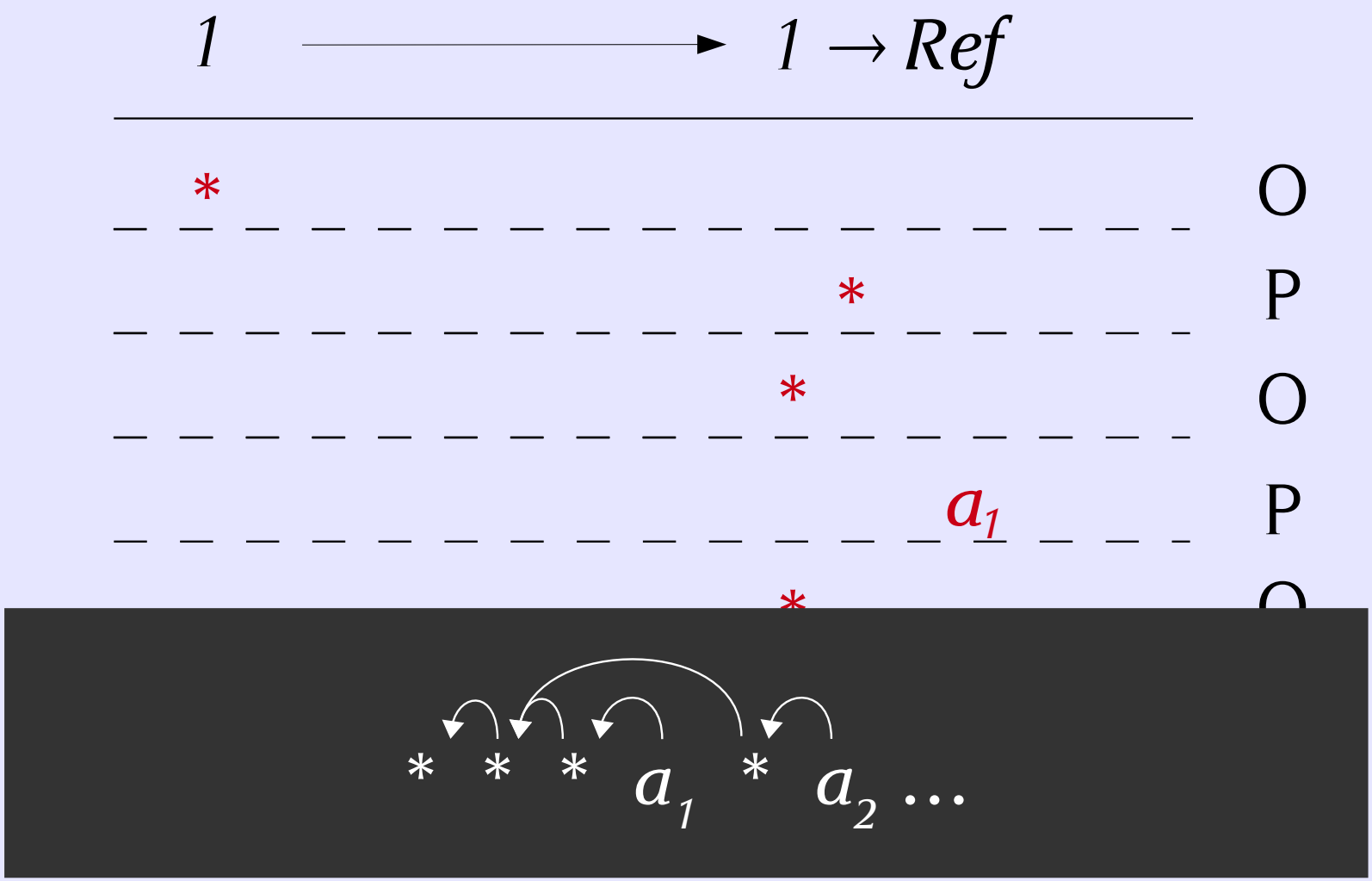
Examples

$\lambda x.\mathtt{ref}() : \mathtt{com} \to \mathtt{ref}$



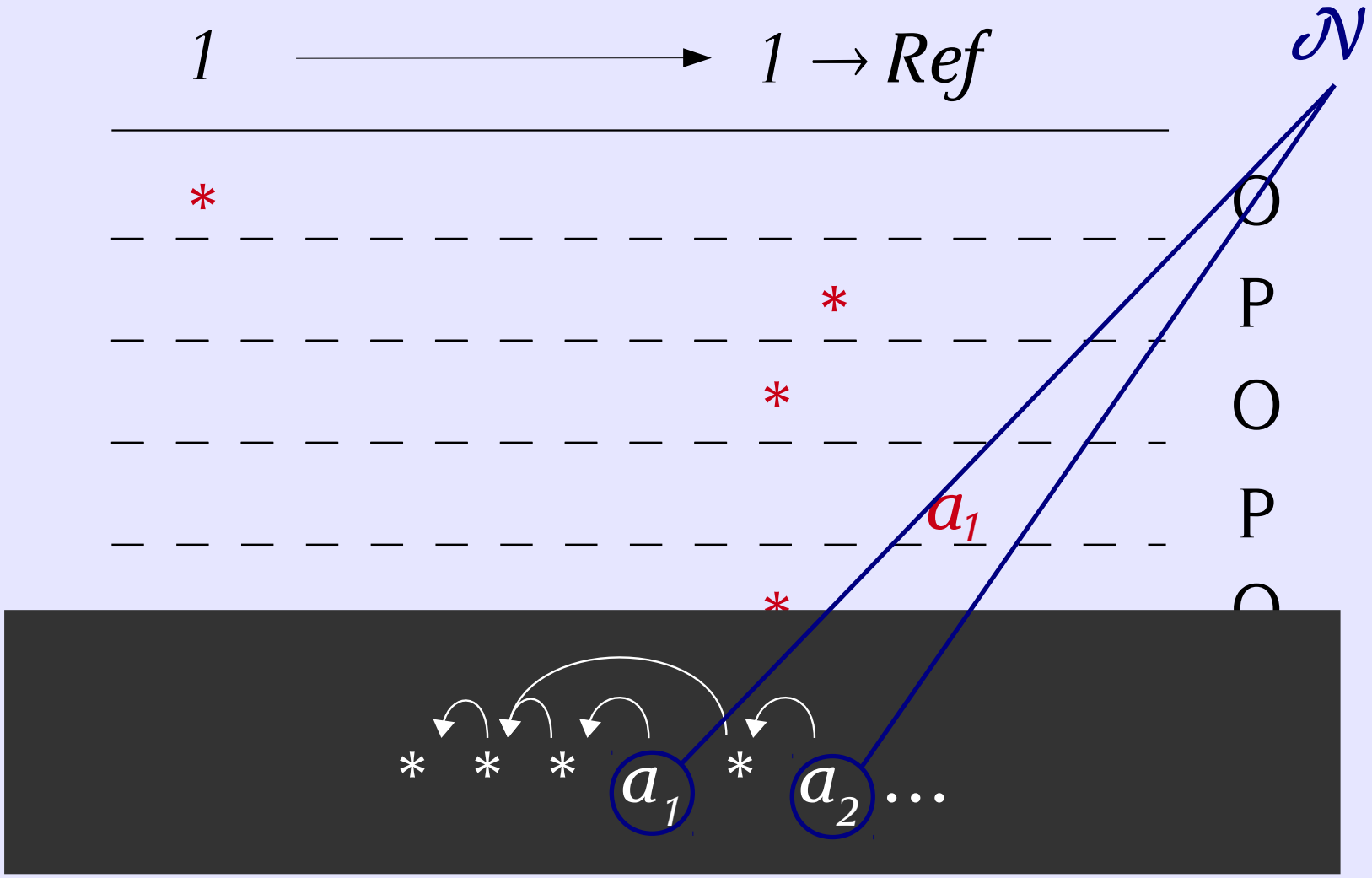
Examples

$\lambda x.\mathtt{ref}() : \mathtt{com} \to \mathtt{ref}$



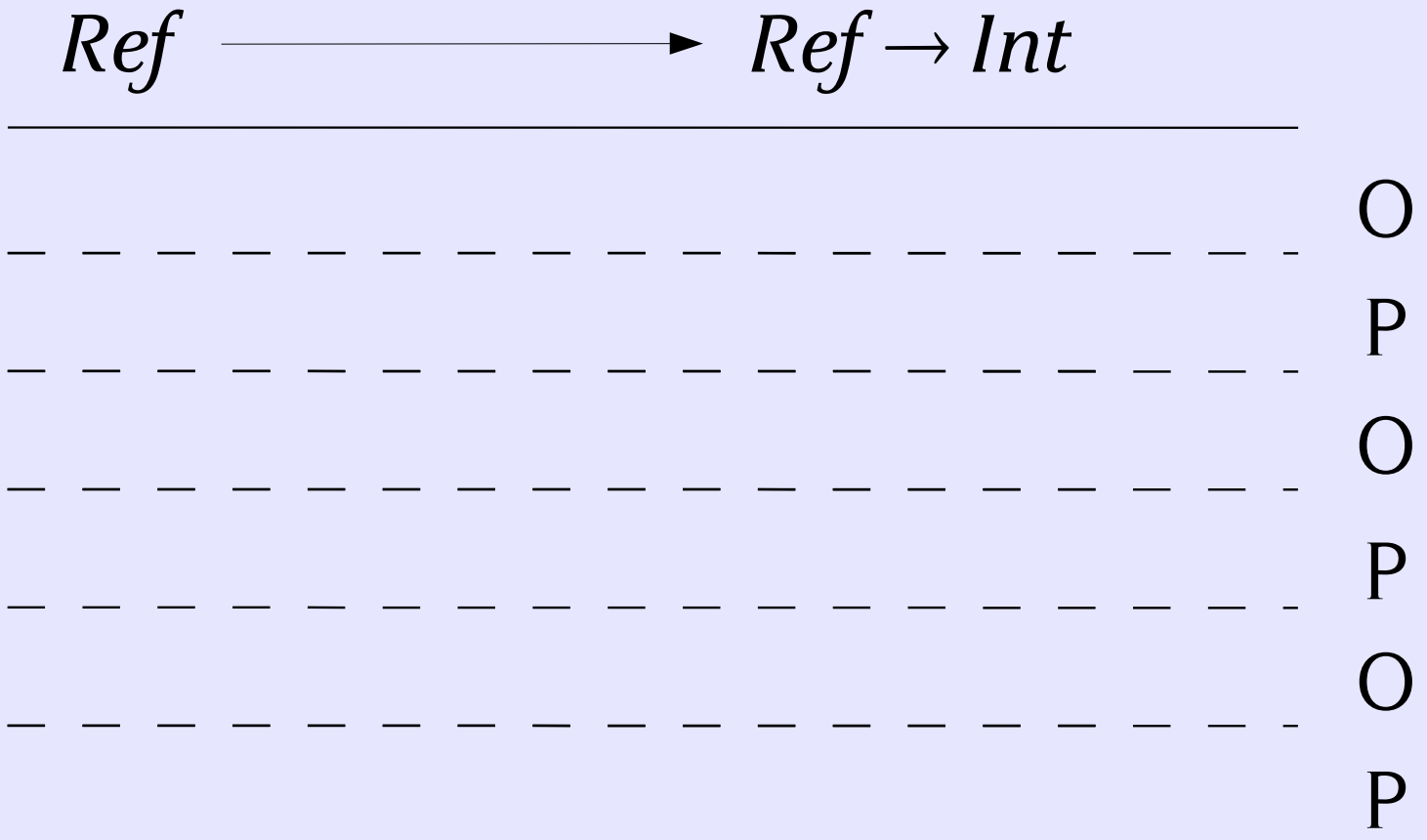
Examples

$\lambda x.\mathtt{ref}() : \mathtt{com} \to \mathtt{ref}$



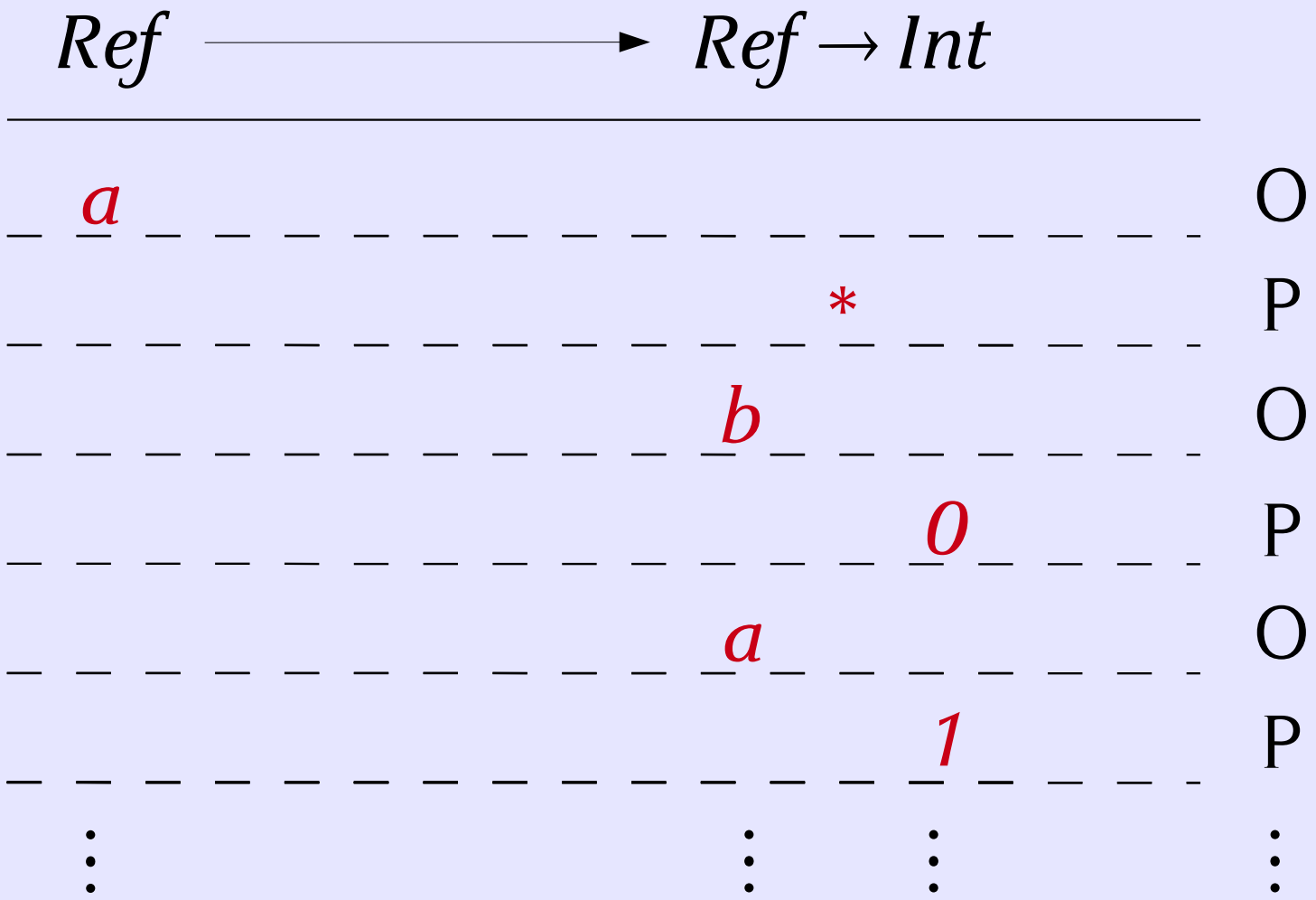
Examples

$x : \texttt{ref} \vdash \lambda y.(x == y) : \texttt{ref} \to \texttt{int}$



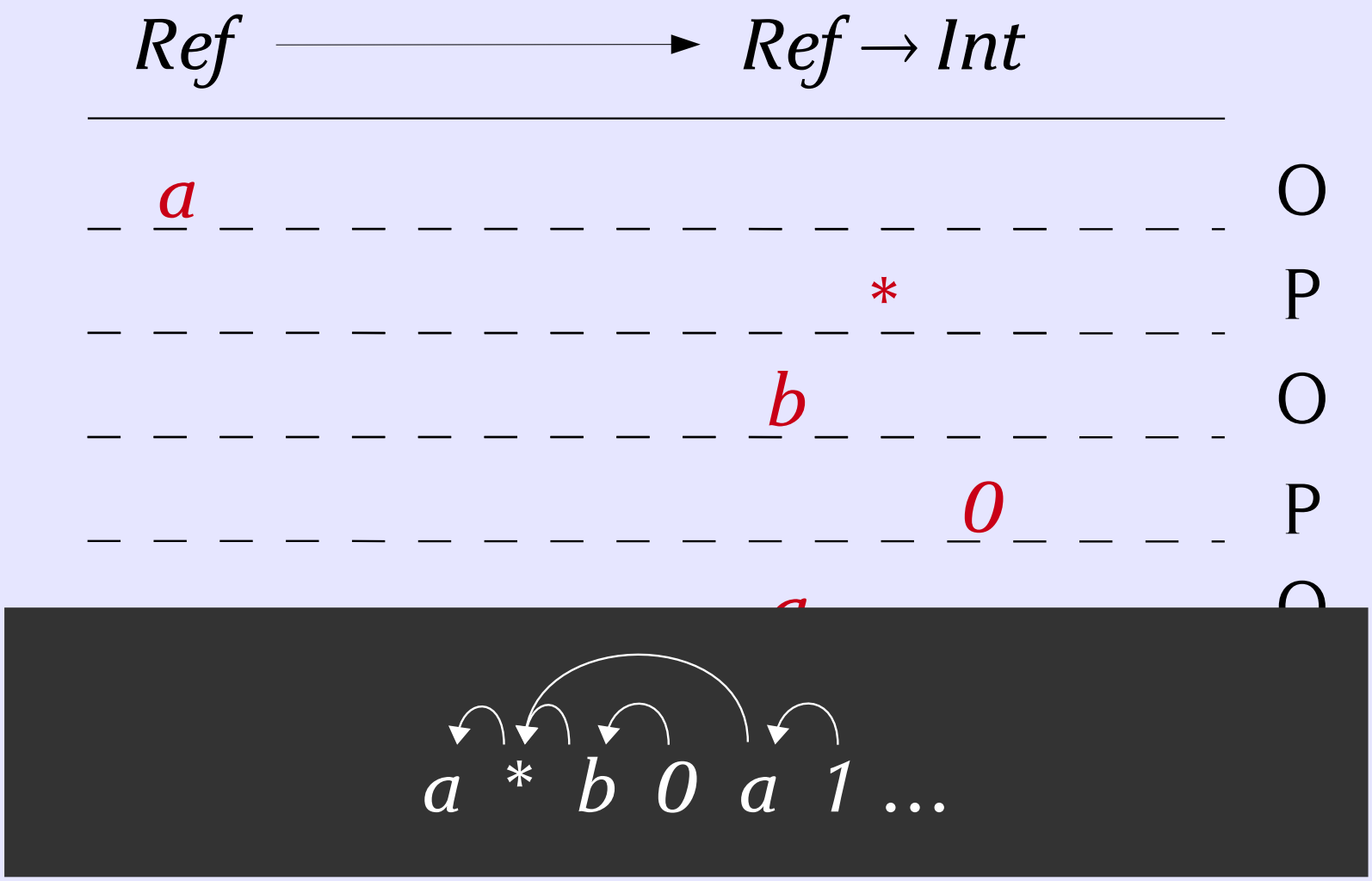
Examples

$x : \mathtt{ref} \vdash \lambda y.(x == y) : \mathtt{ref} \to \mathtt{int}$



Examples

$x : \mathtt{ref} \vdash \lambda y.(x == y) : \mathtt{ref} \to \mathtt{int}$

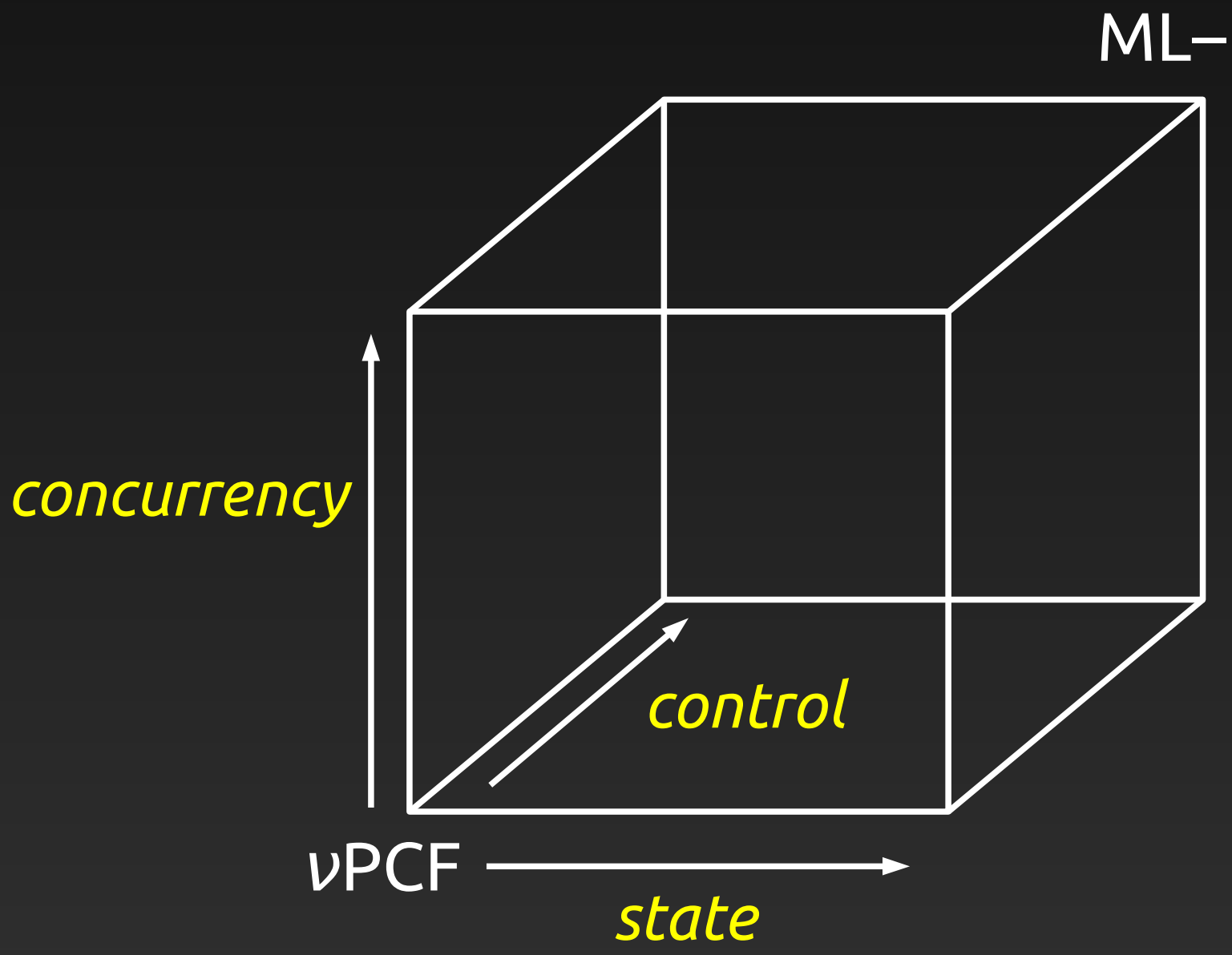


Games for νPCF

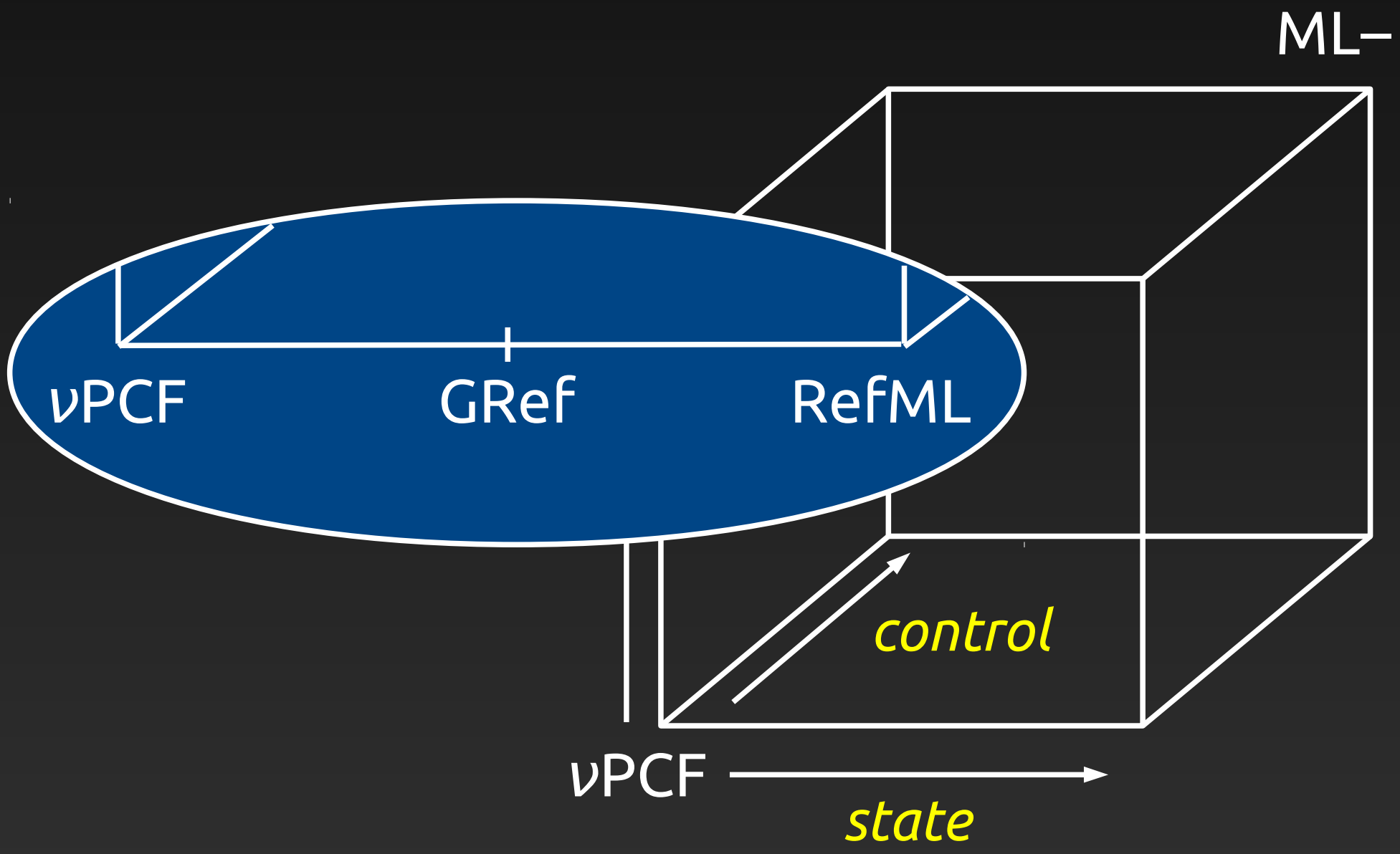
- Moves with names
- Nominal saturation

- Determinism
- Alternation
- Well-bracketing
- Visibility
- Innocence

Add state



Add state



GRef: full ground store

- A ::= `com` | `ref`^{i}`int` | A_f $\quad\quad$ A_f ::= $A \to A$

GRef: full ground store

- $A ::= \texttt{com} \mid \texttt{ref}^i\,\texttt{int} \mid A_f \qquad A_f ::= A \to A$
- $\texttt{()}:\texttt{com},\ i:\texttt{int},\ \texttt{if}:\texttt{int}\to A\to A\to A,\ \texttt{Y}:(A_f\to A_f)\to A_f$

  …
- $$\frac{\Gamma \vdash s:\texttt{ref}^i\,\texttt{int}}{\Gamma \vdash \texttt{ref}(s):\texttt{ref}^{i+1}\,\texttt{int}} \qquad \frac{\Gamma \vdash s,t:\texttt{ref}^{i+1}\,\texttt{int}}{\Gamma \vdash s == t:\texttt{int}}$$
- $$\frac{\Gamma \vdash s:\texttt{ref}^{i+1}\,\texttt{int}}{\Gamma \vdash !s:\texttt{ref}^i\,\texttt{int}} \qquad \frac{\Gamma \vdash s:\texttt{ref}^{i+1}\,\texttt{int},\ t:\texttt{ref}^i\,\texttt{int}}{\Gamma \vdash s:=t:\texttt{com}}$$

Games with store

- Moves with store

$$Moves : \quad m^{S}$$

$$S = \{ (a,4), (b,c), (c,3), \ldots \}$$

- Frugality

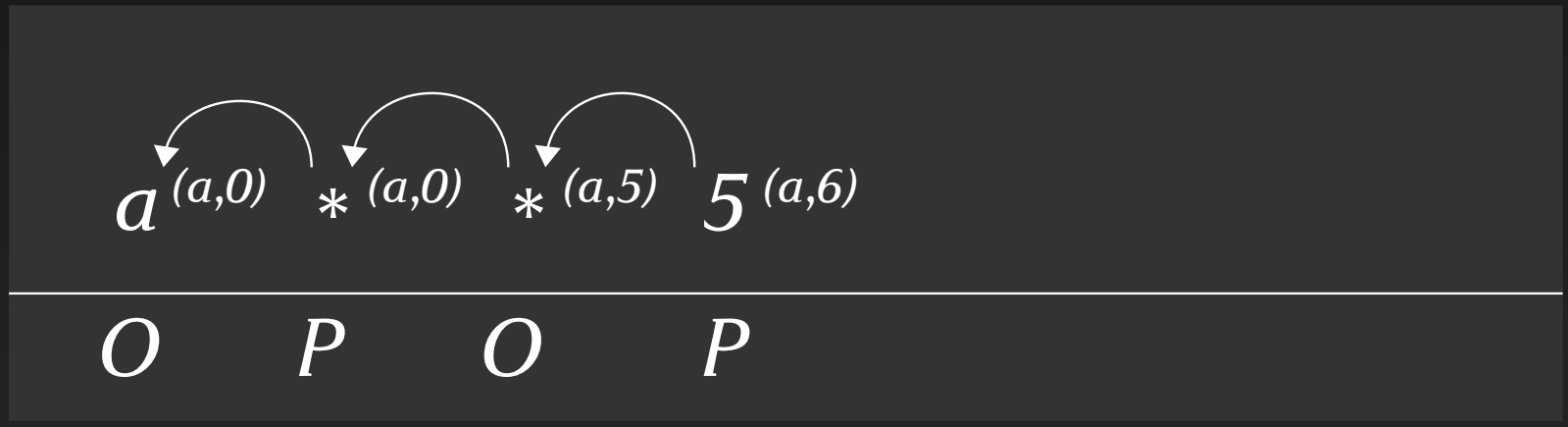
$$\ldots m^{\{(a,v), \ldots\}} \Rightarrow \ldots a^{S} \ldots m^{\{(a,v), \ldots\}}$$

Break innocence

$$\texttt{x} : \texttt{ref}\ \texttt{int} \vdash \lambda\texttt{y}.\ \texttt{x++; !x-1} : \texttt{com} \to \texttt{int}$$

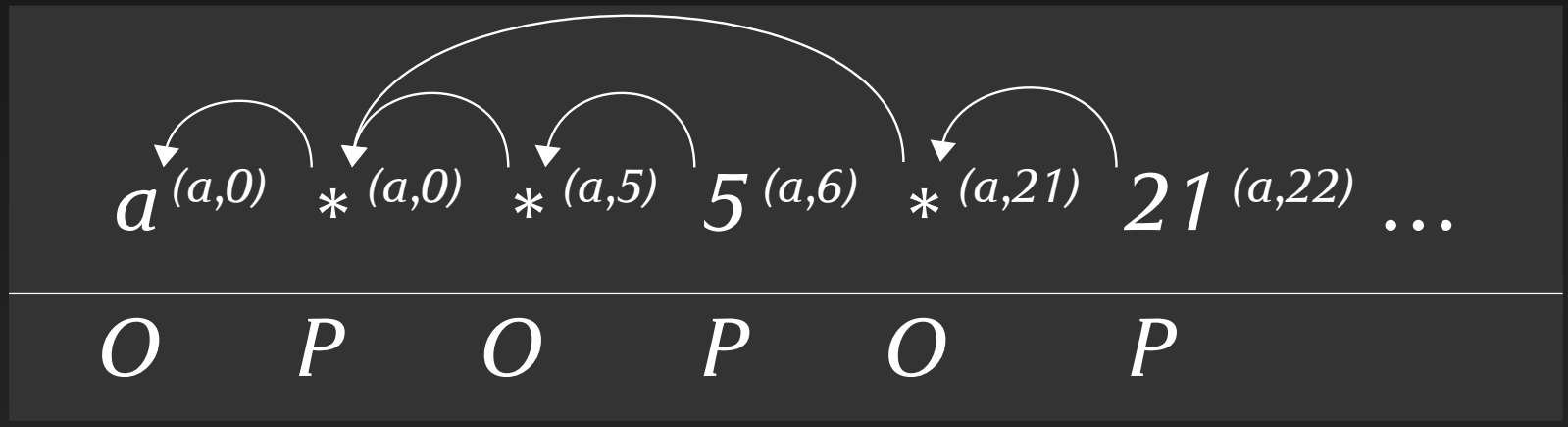
Break innocence

$\texttt{x : ref int} \vdash \texttt{λy. x++; !x-1 : com} \to \texttt{int}$



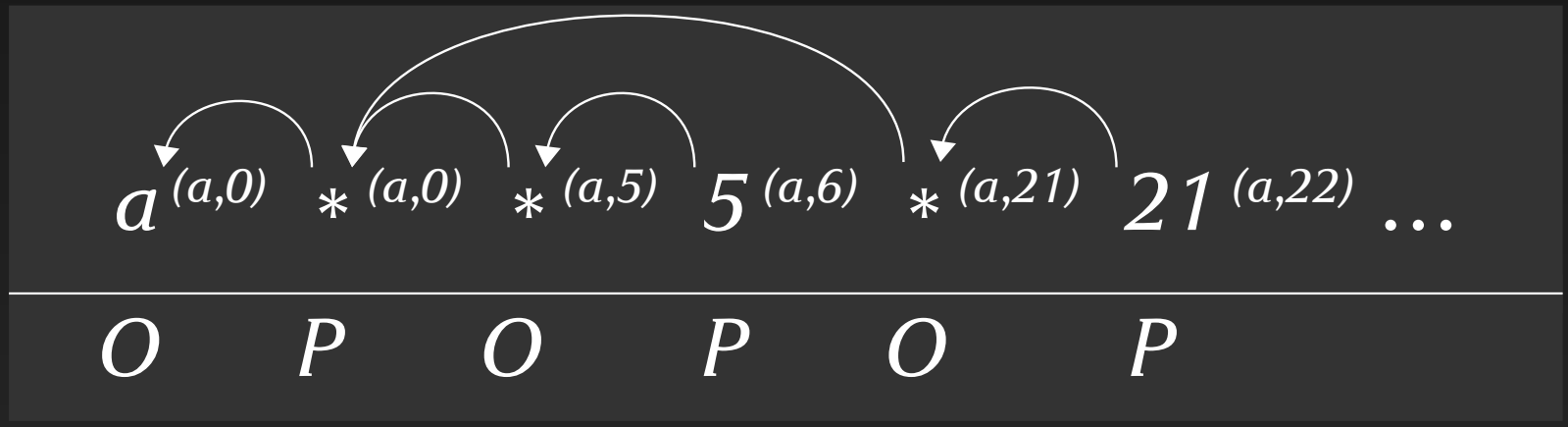
Break innocence

$\texttt{x : ref int} \vdash \texttt{λy. x++; !x-1 : com} \to \texttt{int}$



Break innocence

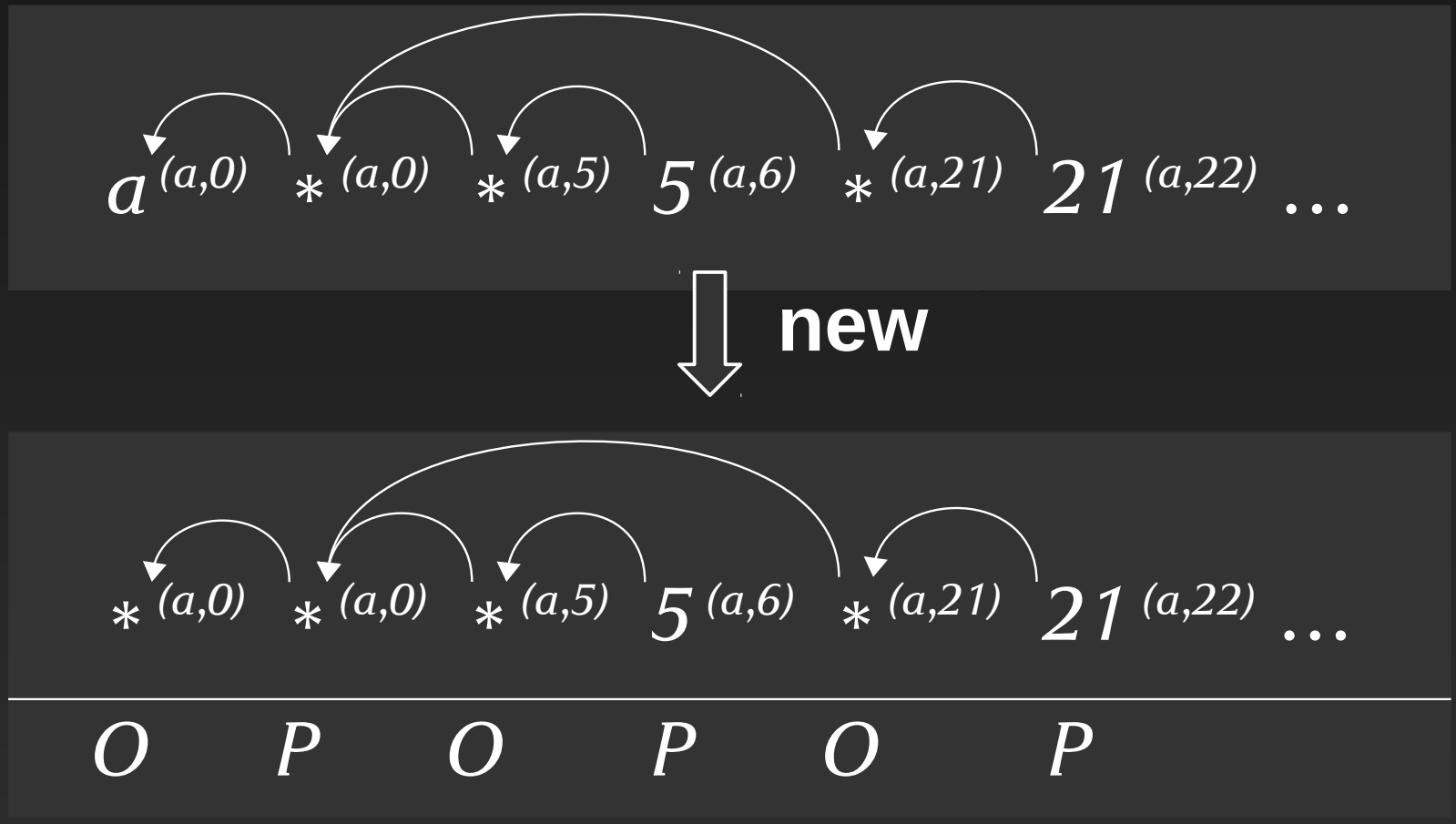
$\texttt{x : ref int} \vdash \texttt{λy. x++; !x-1 : com} \to \texttt{int}$



$\vdash \texttt{let x=ref(0) in λy. x++; !x-1 : com} \to \texttt{int}$

Break innocence

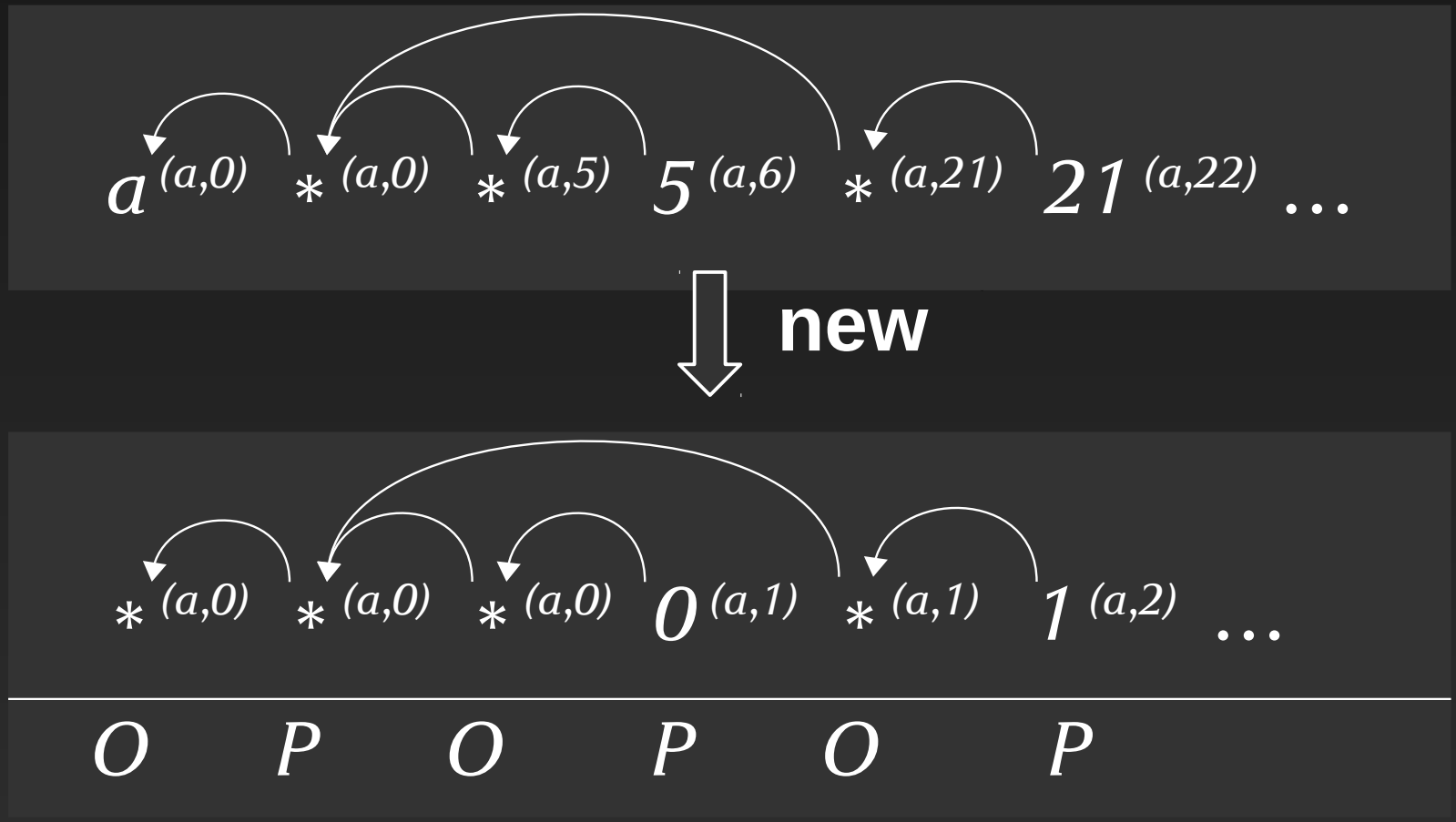
$x : \mathtt{ref}\ \mathtt{int} \vdash \lambda y.\ x{+}{+};\ !x - 1 : \mathtt{com} \to \mathtt{int}$



$\vdash \mathtt{let}\ x = \mathtt{ref}(0)\ \mathtt{in}\ \lambda y.\ x{+}{+};\ !x - 1 : \mathtt{com} \to \mathtt{int}$

Break innocence

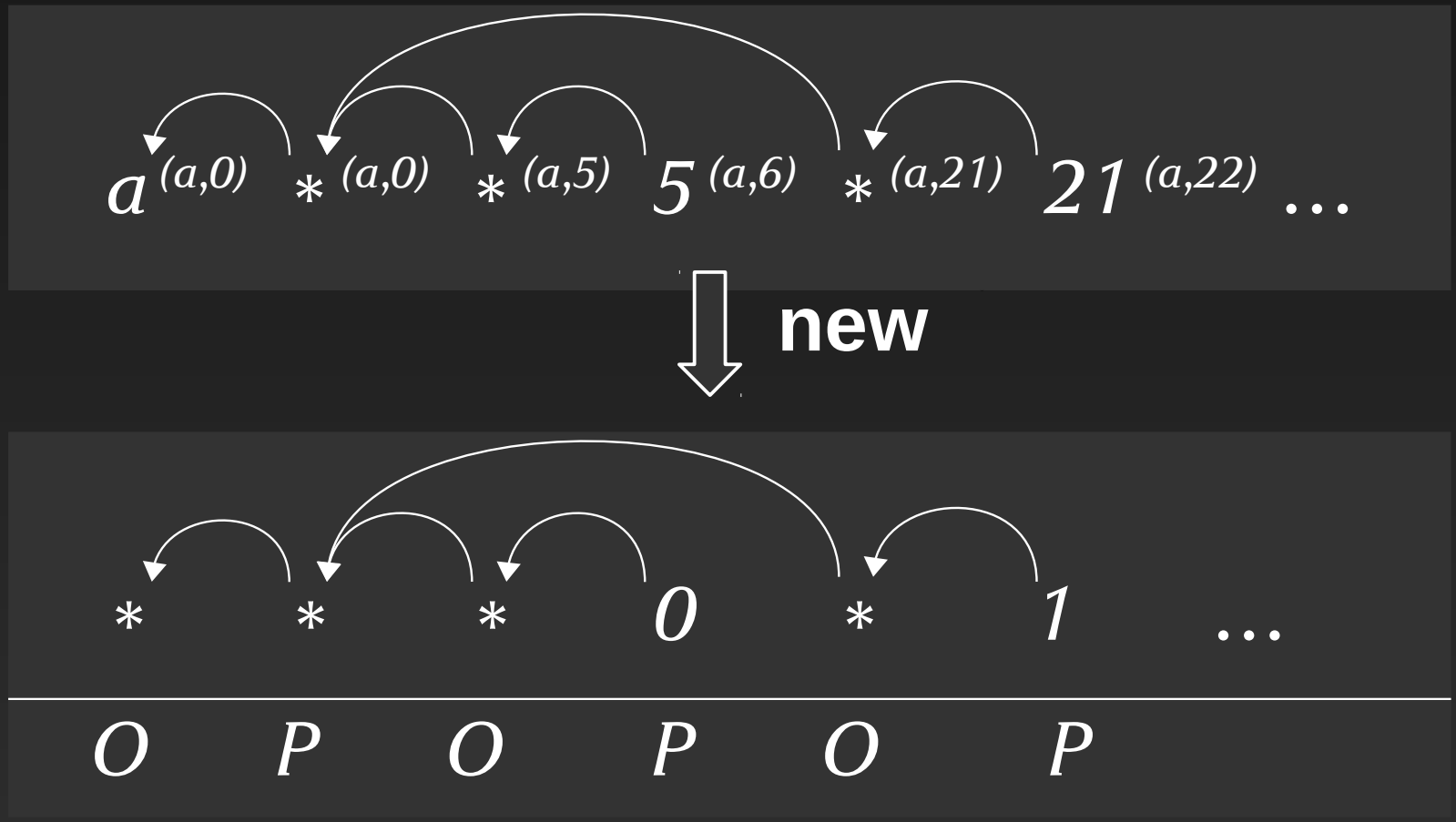
$\texttt{x : ref int} \vdash \lambda\texttt{y. x++; !x-1 : com} \to \texttt{int}$



$\vdash \texttt{let x=ref(0) in } \lambda\texttt{y. x++; !x-1 : com} \to \texttt{int}$

Break innocence

$\texttt{x : ref int} \vdash \texttt{λy. x++; !x-1 : com} \to \texttt{int}$



$\vdash \texttt{let x=ref(0) in λy. x++; !x-1 : com} \to \texttt{int}$

RefML: full higher-order store

- A ::= `com` | `int` | `ref` A | A_f $\qquad$ A_f ::= $A \to A$

RefML: full higher-order store

- A ::= `com` | `int` | `ref` A | A_f $\qquad$ A_f ::= $A \to A$
- `()`:`com`, i:`int`, `if`:`int`$\to A \to A \to A$, `Y`:$(A_f \to A_f) \to A_f$

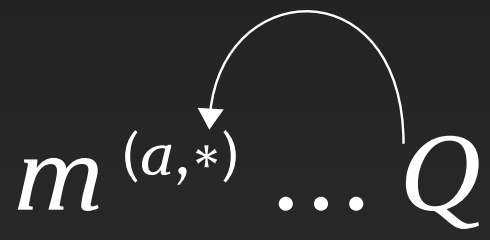
  …
- $$\frac{\Gamma \vdash s : A}{\Gamma \vdash \texttt{ref}(s) : \texttt{ref}\, A} \qquad \frac{\Gamma \vdash s, t : \texttt{ref}\ A}{\Gamma \vdash s == t : \texttt{int}}$$
- $$\frac{\Gamma \vdash s : \texttt{ref}\, A}{\Gamma \vdash\, !s : A} \qquad \frac{\Gamma \vdash s : \texttt{ref}\, A,\ t : A}{\Gamma \vdash s := t : \texttt{com}}$$

Games with HO-store

- Moves with HO-store

$$S = \{\ (a,4),\ (b,c),\ (c,3),\ (d,*),\ \ldots\ \}$$

- Justify by store

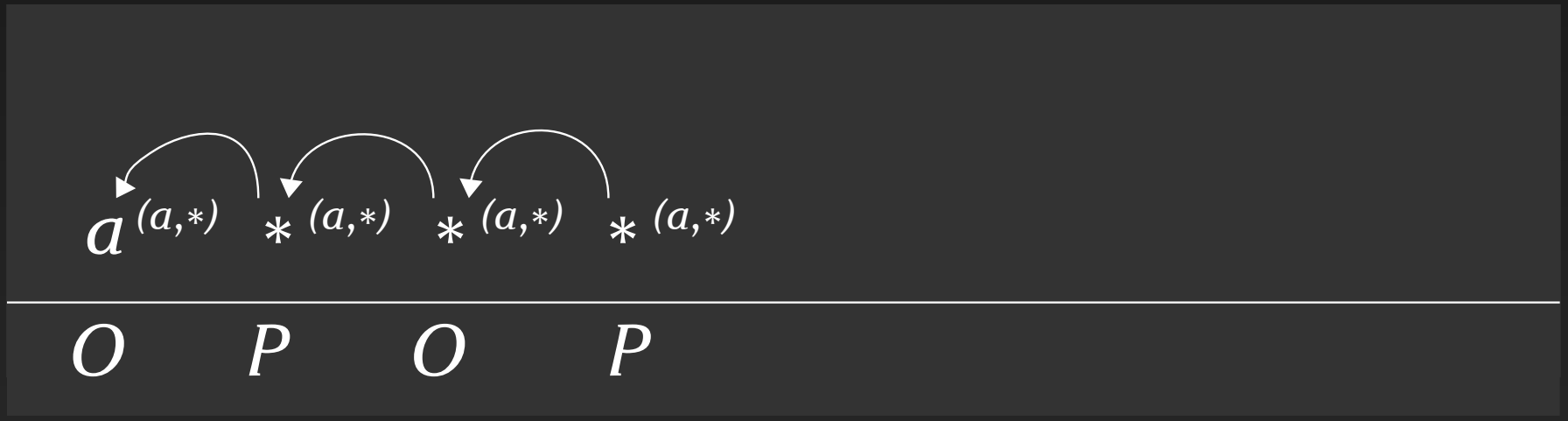


- Frugality

$$\ldots\ m^{\{(a,v),\ \ldots\}} \quad \Rightarrow \quad \ldots\ a^{S}\ \ldots\ m^{\{(a,v),\ \ldots\}}$$

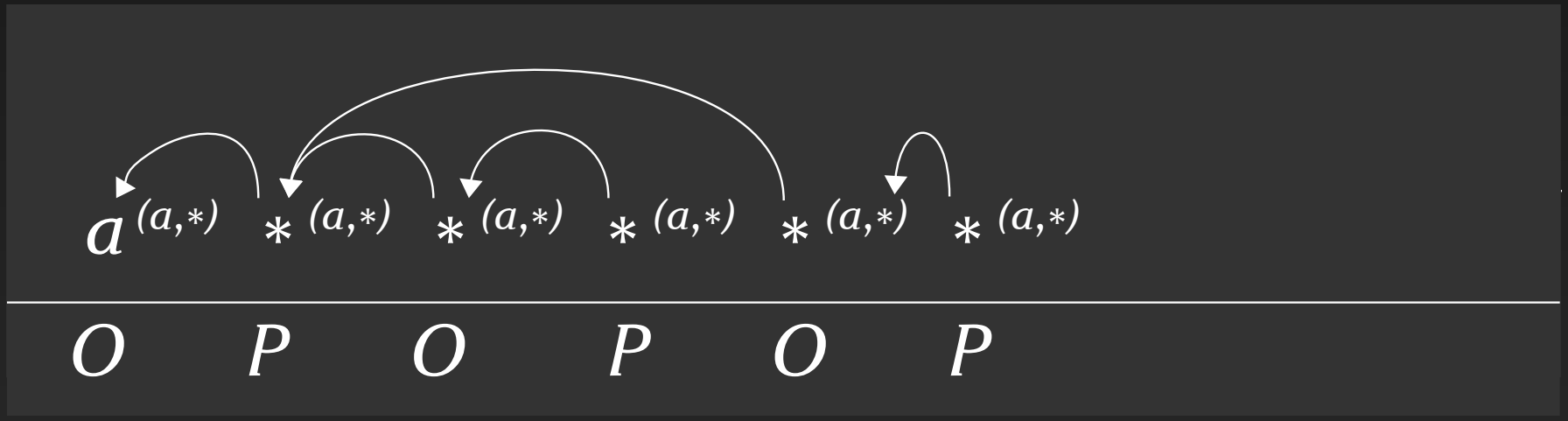
Break visibility

```
x : ref(com→com) ⊢ λy. first: x:=y
                       then: (!x)() : (com→com)→com
```



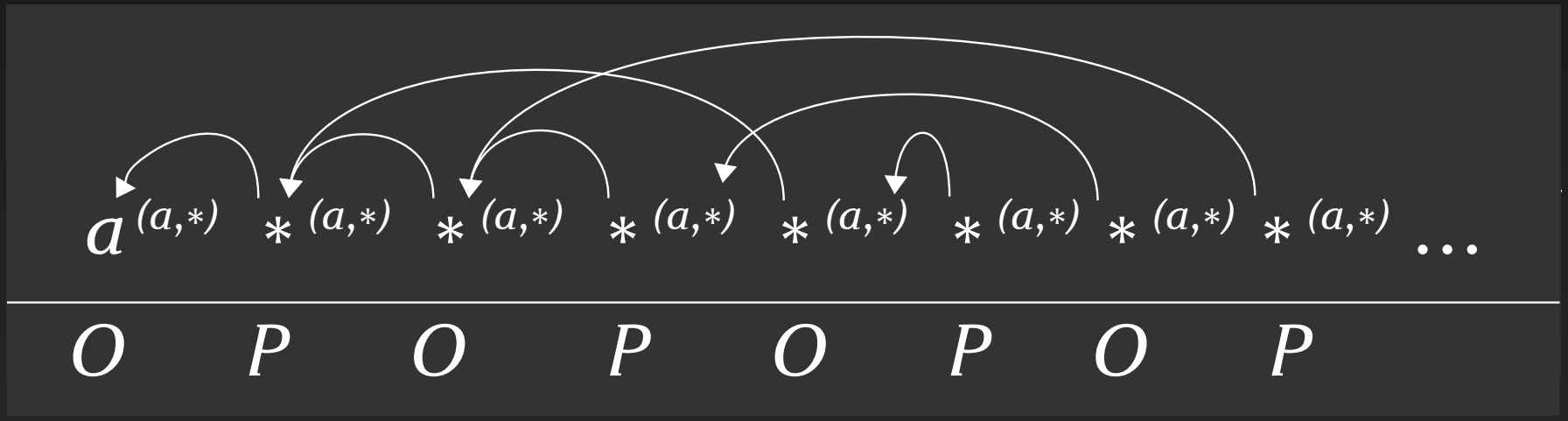
Break visibility

```
x : ref(com→com) ⊢ λy. first: x:=y
                       then: (!x)() : (com→com)→com
```



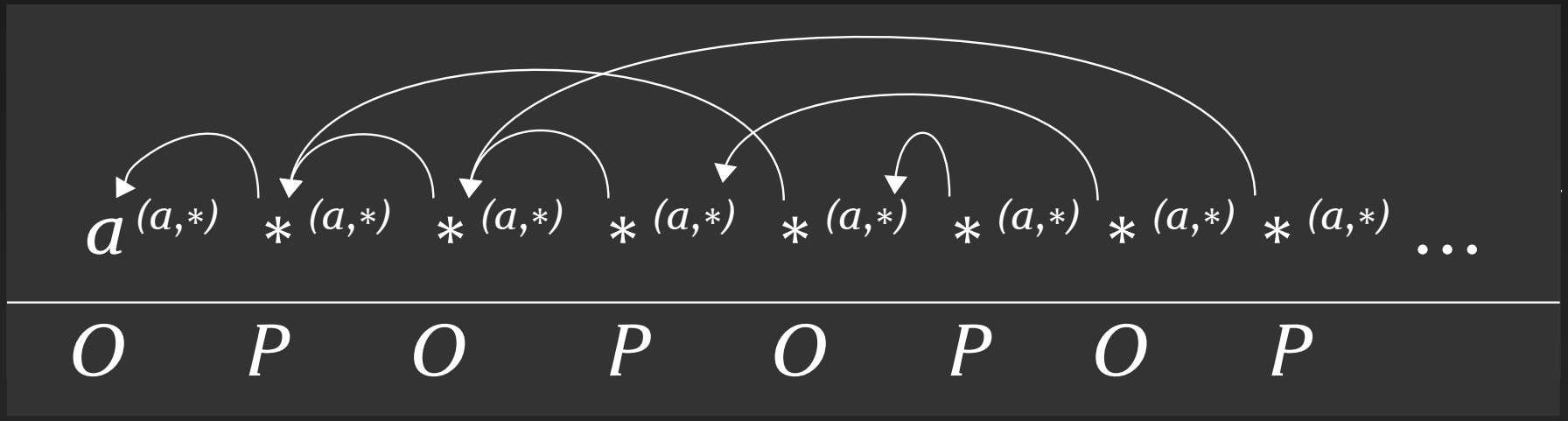
Break visibility

```
x : ref(com→com) ⊢ λy. first: x:=y
                       then: (!x)() : (com→com)→com
```



Break visibility

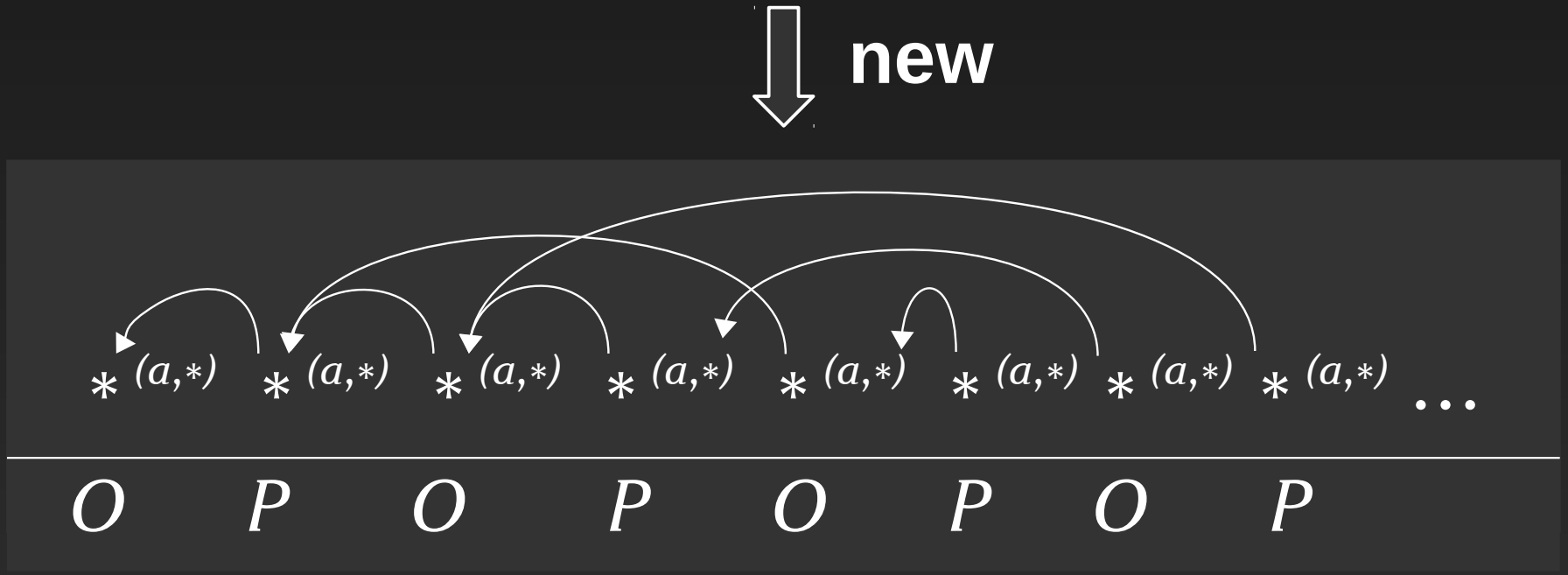
```
x : ref(com→com) ⊢ λy. first: x:=y
                     then: (!x)() : (com→com)→com
```




```
⊢ let x=ref(...) in λy. first: x:=y
                     then: (!x)() : (com→com)→com
```

Break visibility

```
x : ref(com→com) ⊢ λy. first: x:=y
                     then: (!x)() : (com→com)→com
```

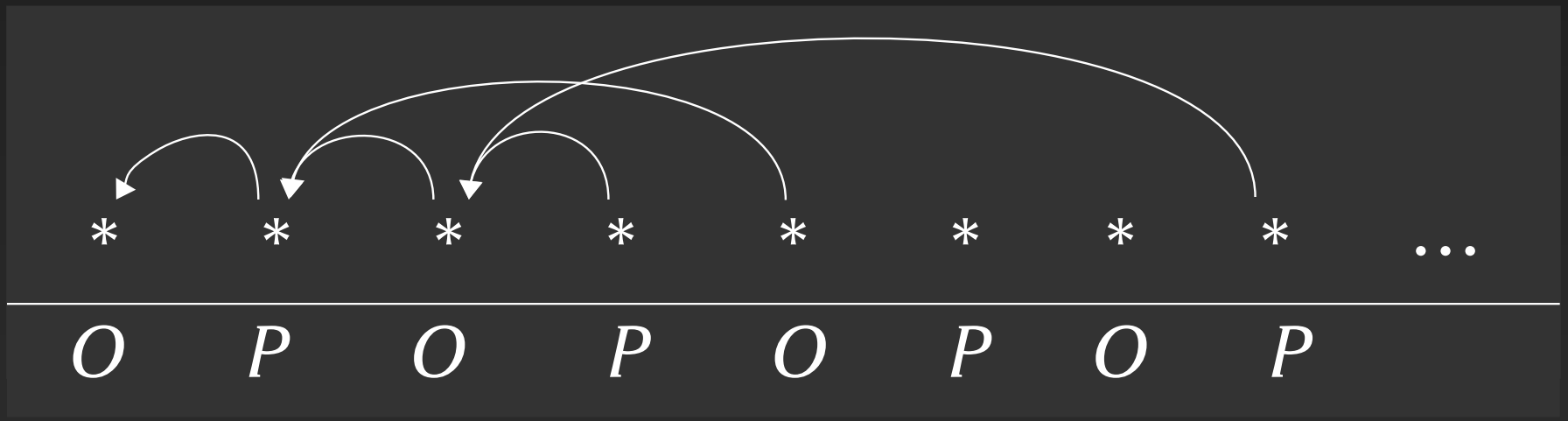



```
⊢ let x=ref(...) in λy. first: x:=y
                     then: (!x)() : (com→com)→com
```

Break visibility

```
x : ref(com→com) ⊢ λy. first: x:=y
                     then: (!x)() : (com→com)→com
```

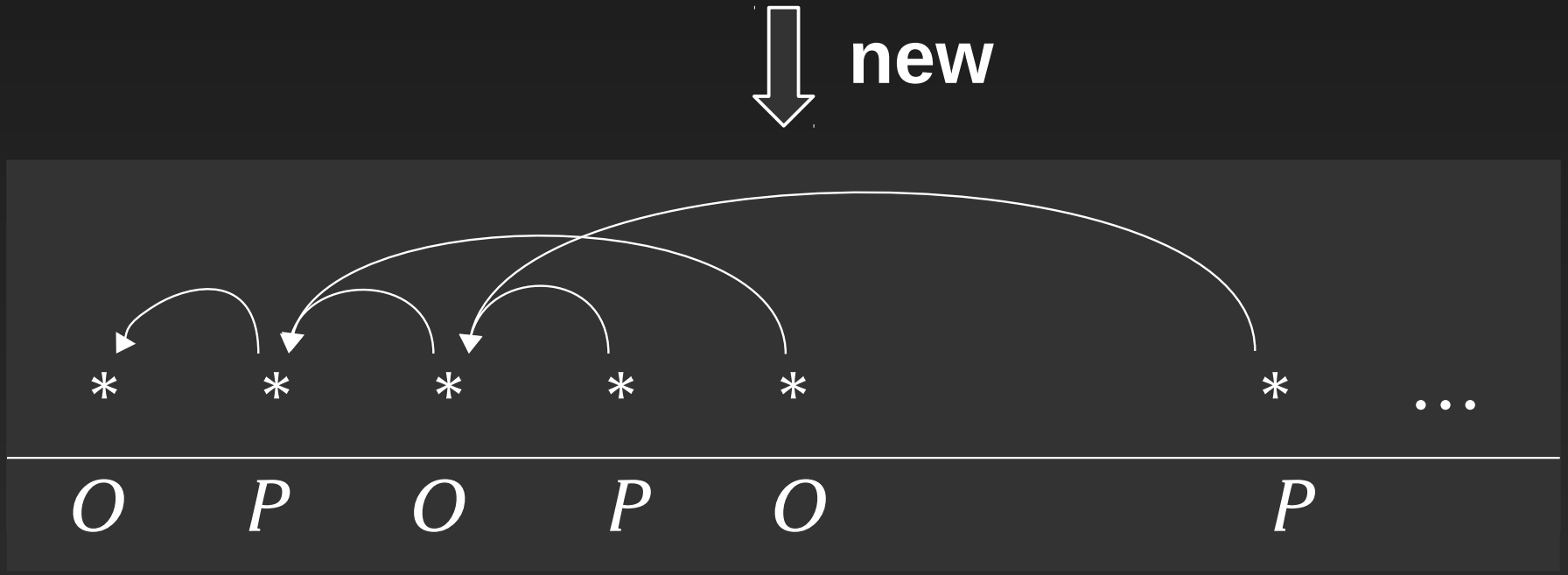
new




```
⊢ let x=ref(...) in λy. first: x:=y
                     then: (!x)() : (com→com)→com
```

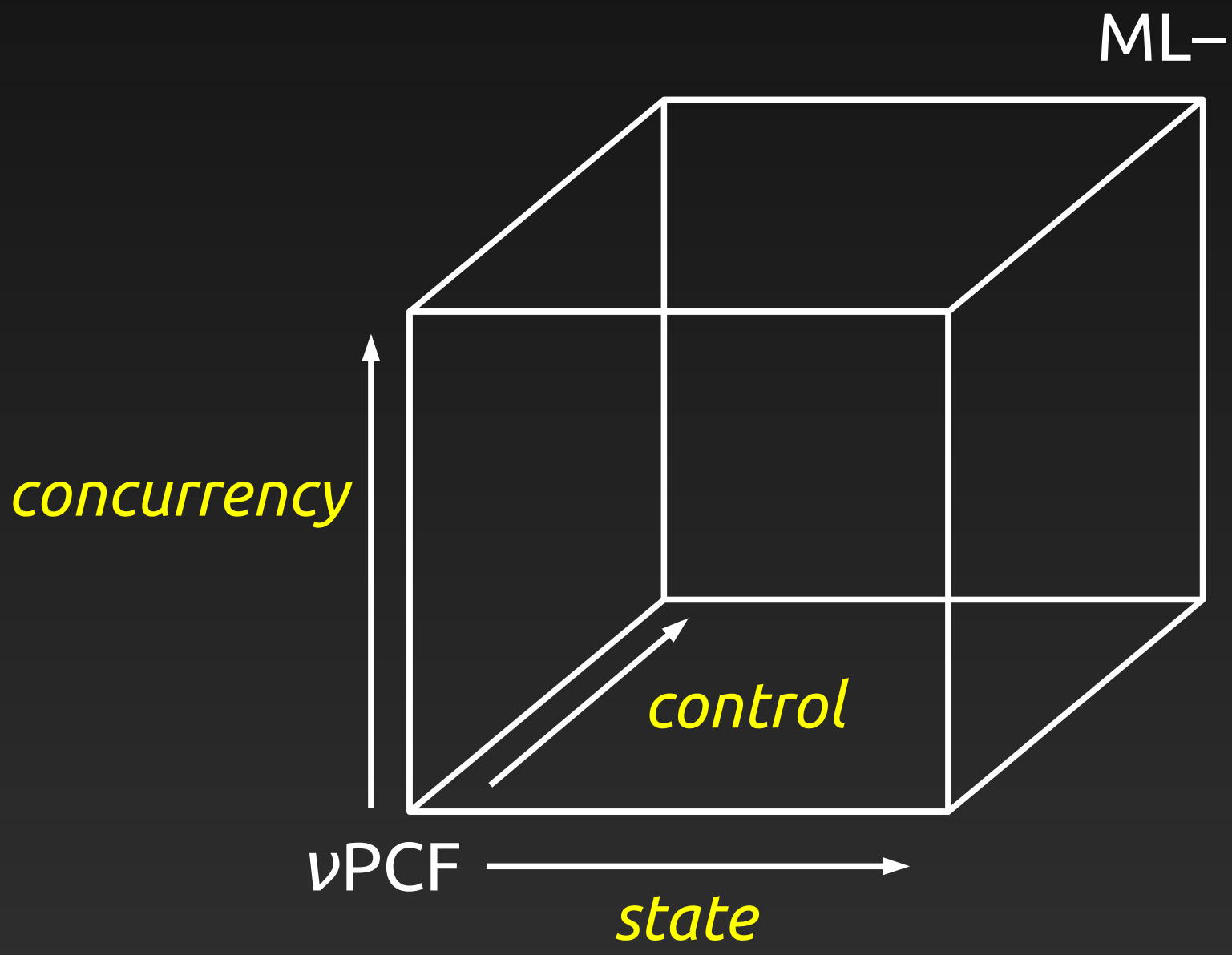
Break visibility

```
x : ref(com→com) ⊢ λy. first: x:=y
                     then: (!x)() : (com→com)→com
```

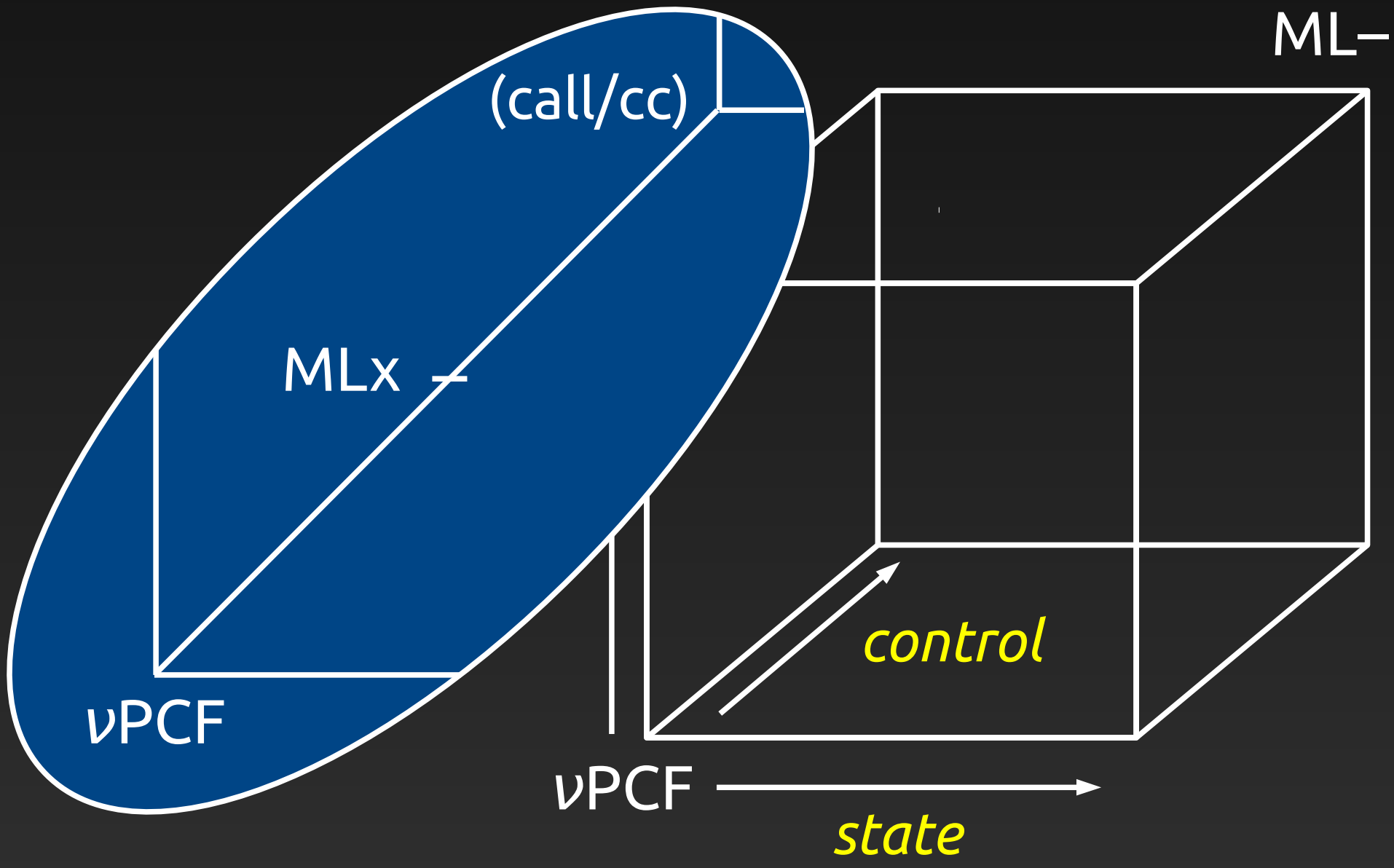



```
⊢ let x=ref(...) in λy. first: x:=y
                     then: (!x)() : (com→com)→com
```

Add control



Add control



MLx: simple exceptions

- A ::= `com` | `int` | `exc` | A_f $\qquad$ A_f ::= $A \to A$

MLx: simple exceptions

- A ::= com | int | exc | A_f $\qquad$ A_f ::= $A \to A$
- $():\mathtt{com}$, $i:\mathtt{int}$, $\mathtt{if}:\mathtt{int}\to A\to A\to A$, $\mathtt{Y}:(A_f\to A_f)\to A_f$

  …
- $\Gamma \vdash \mathtt{exc}():\mathtt{exc}$

$$\frac{\Gamma \vdash s:\mathtt{exc}}{\Gamma \vdash \mathtt{raise}\ s : A} \qquad \frac{\Gamma \vdash s:\mathtt{exc} \quad \Gamma \vdash t,t':A}{\Gamma \vdash \mathtt{try}\ t\ \mathtt{handle}\ s\ t' : A}$$

Games with exceptions

- Exception names

$$Moves = \{ \ldots, e, f, \ldots \}$$

- Raised exceptions

- Frugality

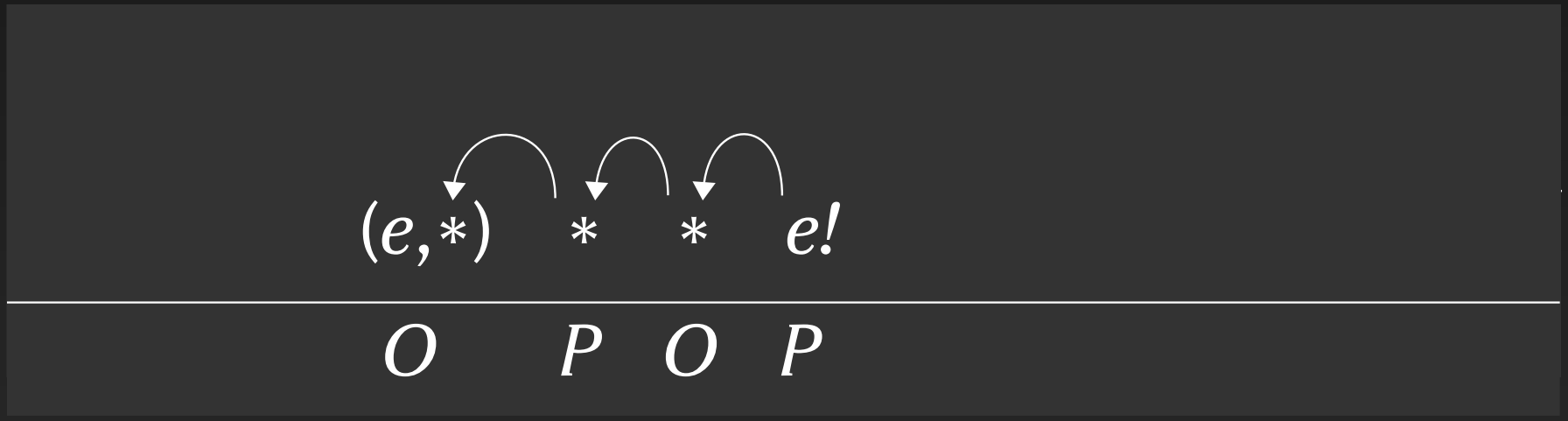
$$\ldots e! \quad \Rightarrow \quad \ldots e \ldots e!$$

Break well-bracketing

```
x:exc, f:(com→com)→com ⊢ try f(λz.raise x)
                         handle x () : com
```

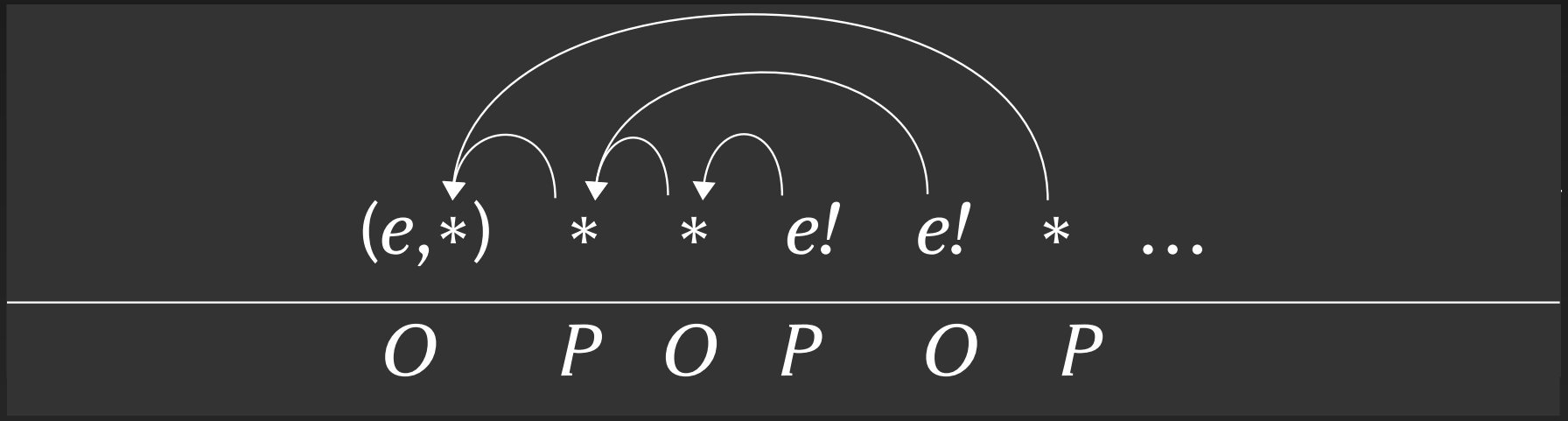
Break well-bracketing

```
x:exc, f:(com→com)→com ⊢ try f(λz.raise x)
                         handle x () : com
```



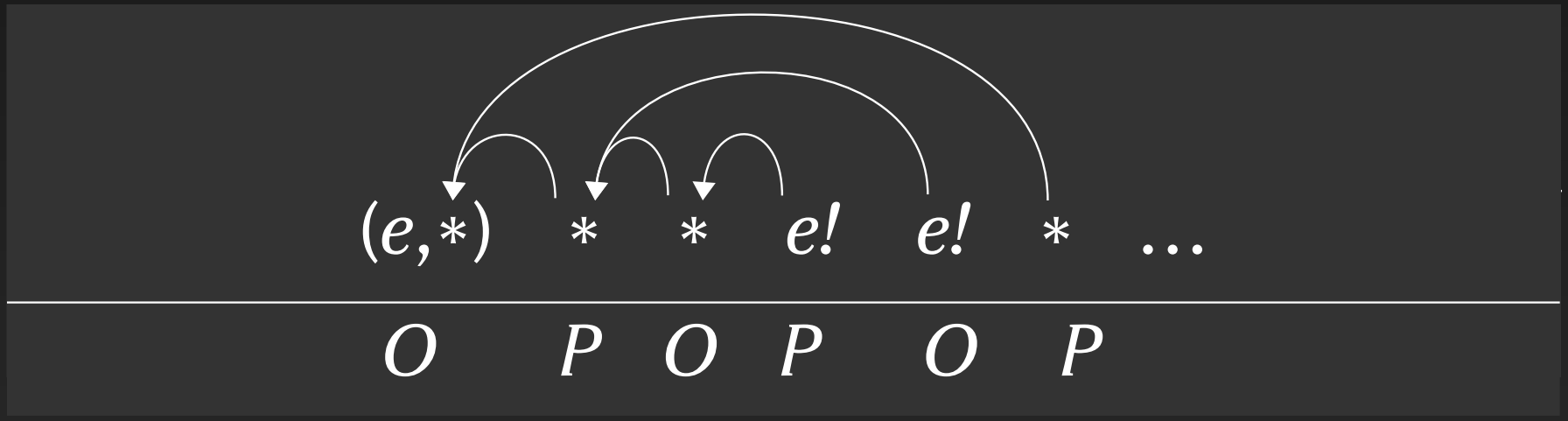
Break well-bracketing

```
x:exc, f:(com→com)→com ⊢ try f(λz.raise x)
                         handle x () : com
```



Break well-bracketing

```
x:exc, f:(com→com)→com ⊢ try f(λz.raise x)
                         handle x () : com
```

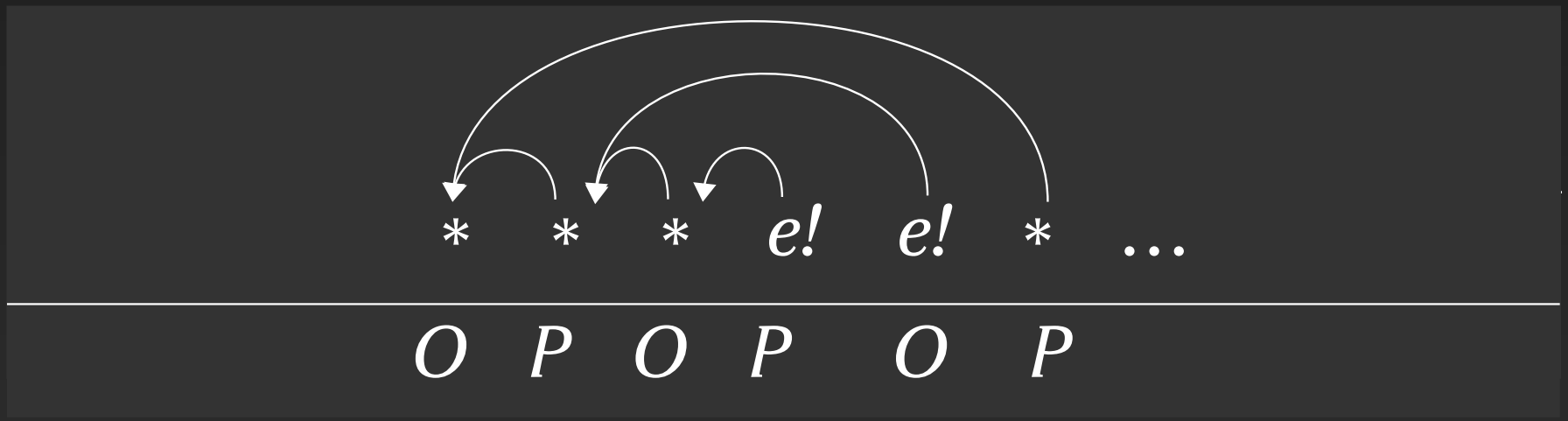



```
f:(com→com)→com ⊢ let x=exc() in
                  try f(λz.raise x) handle x () : com
```

Break well-bracketing

```
x:exc, f:(com→com)→com ⊢ try f(λz.raise x)
                         handle x () : com
```

⇩ **new**

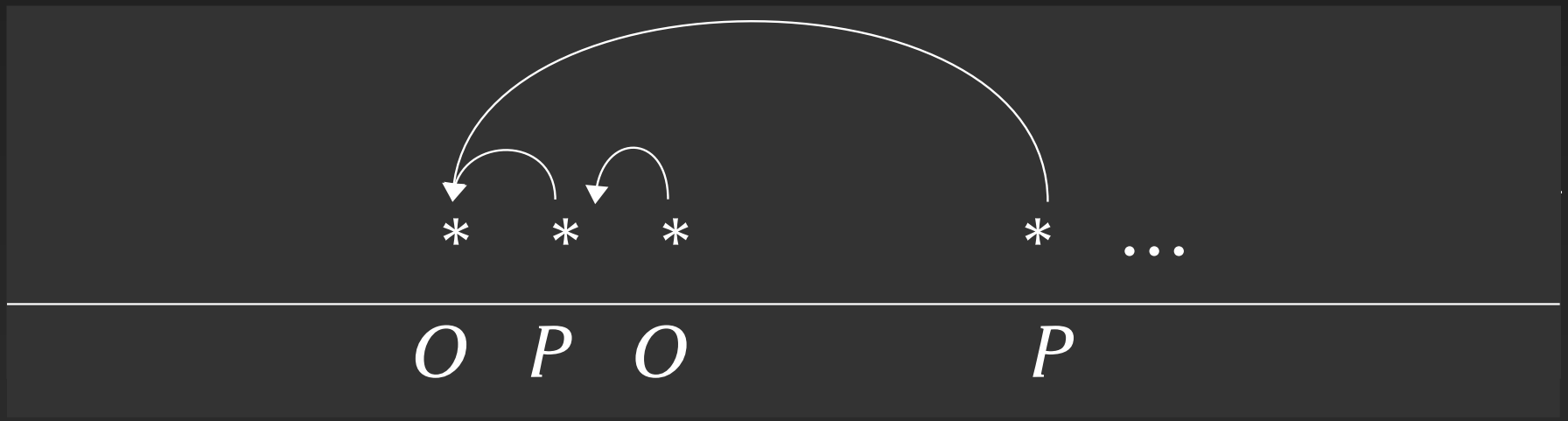



```
f:(com→com)→com ⊢ let x=exc() in
                  try f(λz.raise x) handle x () : com
```

Break well-bracketing

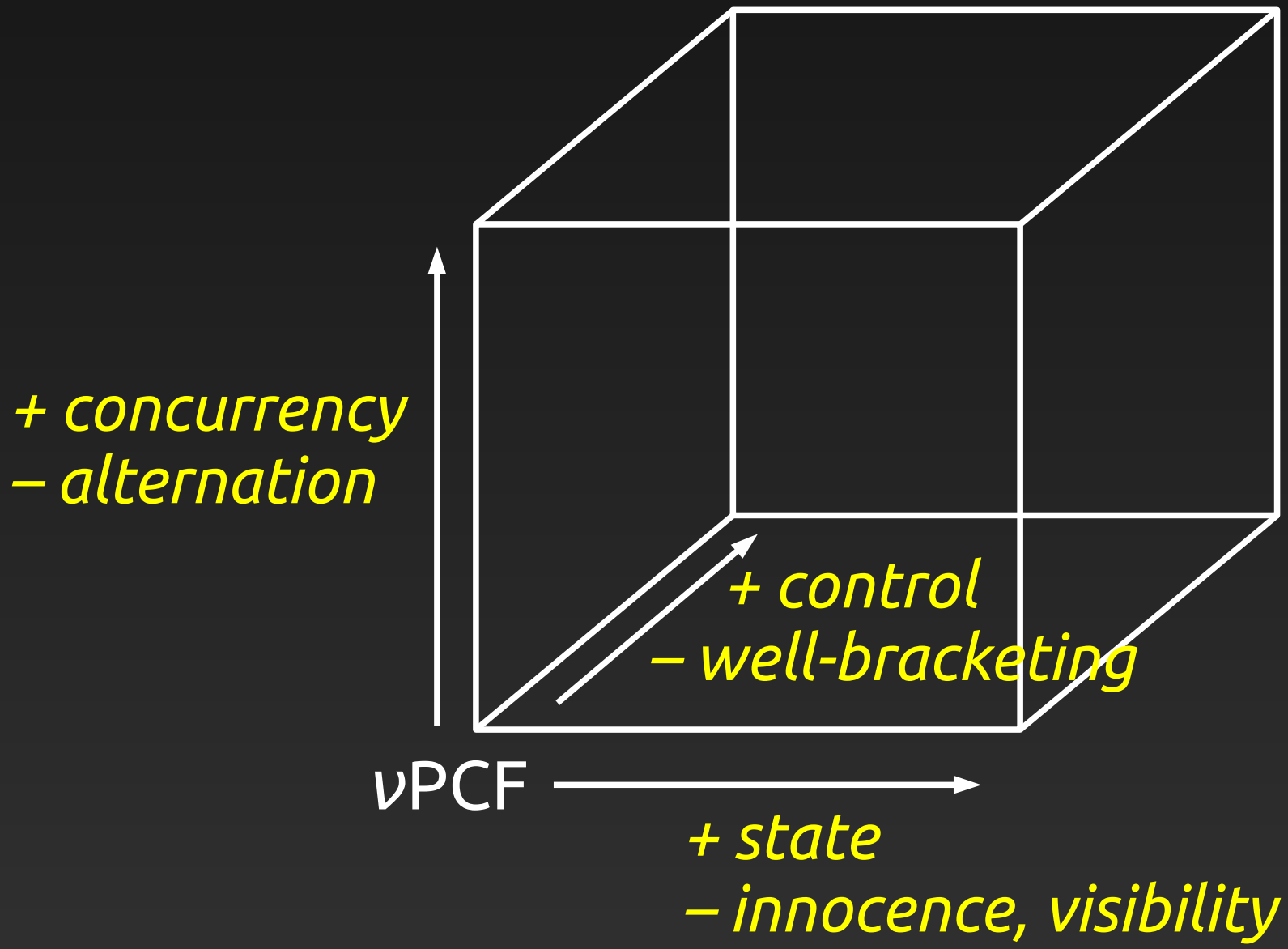
```
x:exc, f:(com→com)→com ⊢ try f(λz.raise x)
                         handle x () : com
```

new




```
f:(com→com)→com ⊢ let x=exc() in
                  try f(λz.raise x) handle x () : com
```

The nominal cube



References

- νPCF:

  Abramsky, Ghica, Murawski, Ong & Stark LICS'04
  Tz. LICS'07
- GRef:

  Laird APAL'08
  Murawski & Tz. ICALP'12
- RefML:

  Murawski & Tz. LICS'11
- conc.ML: Laird FSTTCS'06

Thanks!