

Tipi

prejšnjič...

Rekurzivne definicije smo izenačili z negibnimi točkami

$$f = \lambda n . \text{if } n = 0 \text{ then } 1 \text{ else } n \cdot f(n - 1)$$

$$\Phi = \lambda g . (\lambda n . \text{if } n = 0 \text{ then } 1 \text{ else } n \cdot g(n - 1))$$

$$f = \Phi f$$

Videli smo, kako rekurzivne funkcije dobimo z **iteracijo**

$$\Phi = \lambda g . \lambda n . \text{if } n = 0 \text{ then } 1 \text{ else } n \cdot g(n - 1)$$

$$\begin{aligned} & f_0 \\ & \Phi(f_0) \\ & \Phi(\Phi(f_0)) \\ & \Phi(\Phi(\Phi(f_0))) \end{aligned}$$

Spoznali smo vsestransko uporabne **rekurzivne tipe**

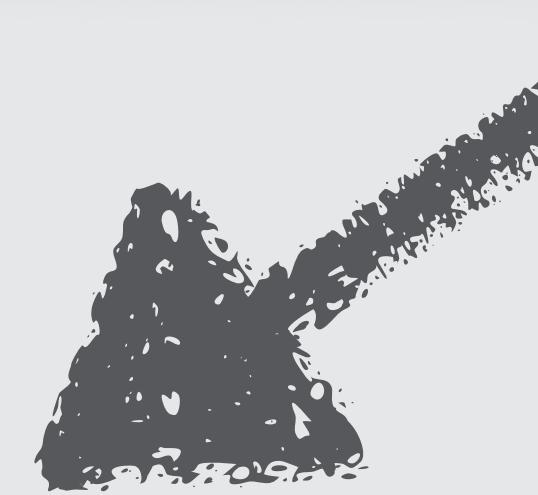
$$e ::= n \mid e_1 + e_2 \mid e_1 - e_2 \mid e_1 * e_2 \mid e_1 / e_2 \mid - e$$

```
type expression =
| Numeral of int
| Plus of expression * expression
| Minus of expression * expression
| Times of expression * expression
| Divide of expression * expression
| Negate of expression
```

S **koinduktivnimi** tipi smo predstavili **neskončne** podatkovne strukture

```
type seznam = Nil | Cons of int * seznam
```

```
Cons (1, Cons (2, Cons (3, Cons (4, ...))))
```



ne



da

induktivni tipi

koinduktivni tipi

práštěvila
v Haskellu

Sistemi tipov

Programski jeziki imajo različno **krepke** sisteme tipov

OCaml

```
# (42, "foo", false)
- : int * string * bool = ...
```

Python

```
>>> type((42, 'foo', False))
<class 'tuple'>
```

strojna koda

0x2a 0x2a 0x00

Statični tipi se preverijo med prevajanjem, **dinamični** med izvajanjem

OCaml

```
# let je_majhen x =
  if x < 10 then "Da" else false
```

Error: This expression has type bool but an expression was expected of type string

Python

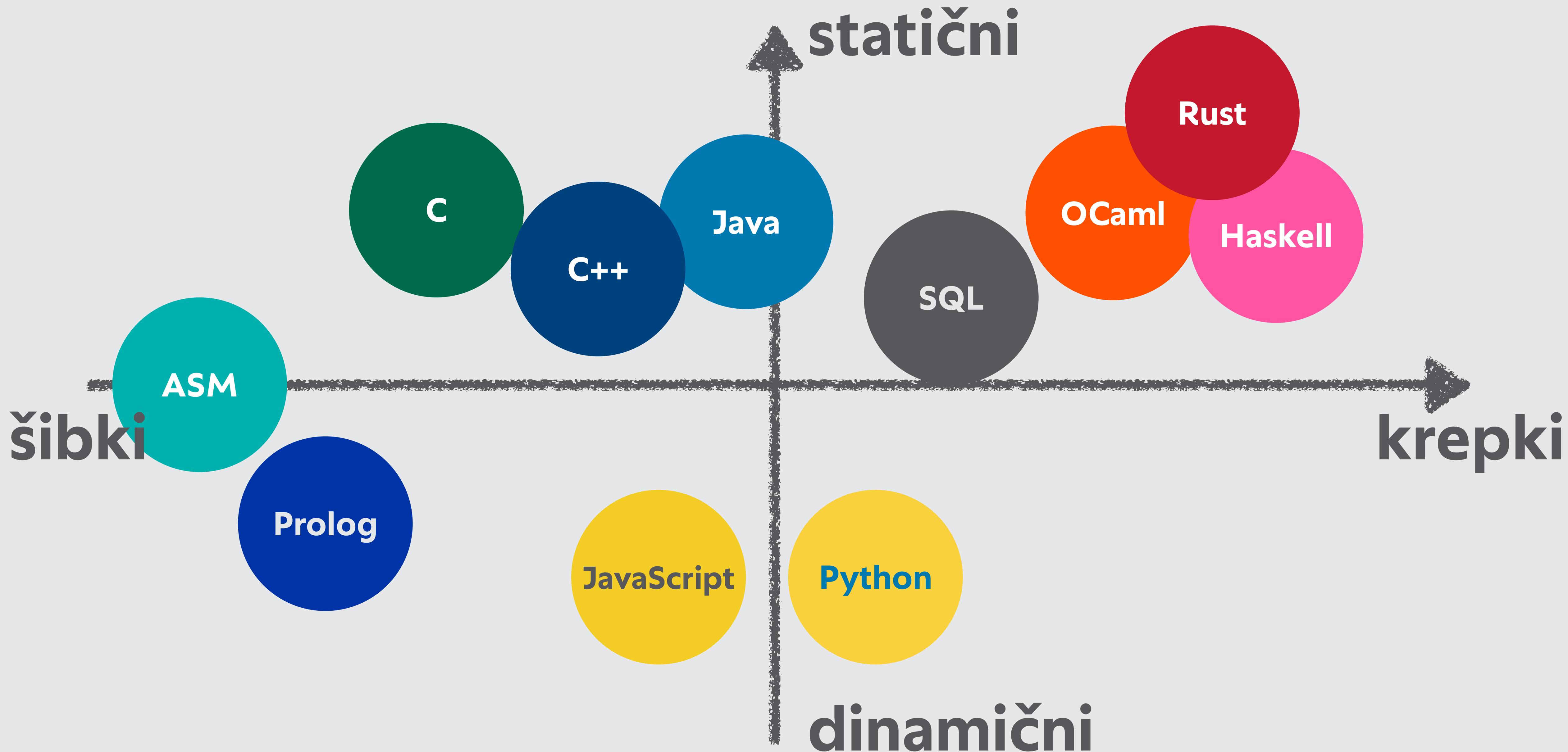
```
>>> def je_majhen(x):
...     return "Da" if x < 10 else False
>>> je_majhen(5) + "!"
"Da!"
>>> je_majhen(15) + "!"
```

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

TypeError: unsupported operand type(s) for +: 'bool' and 'str'

Različni sistemi tipov služijo različnim namenom



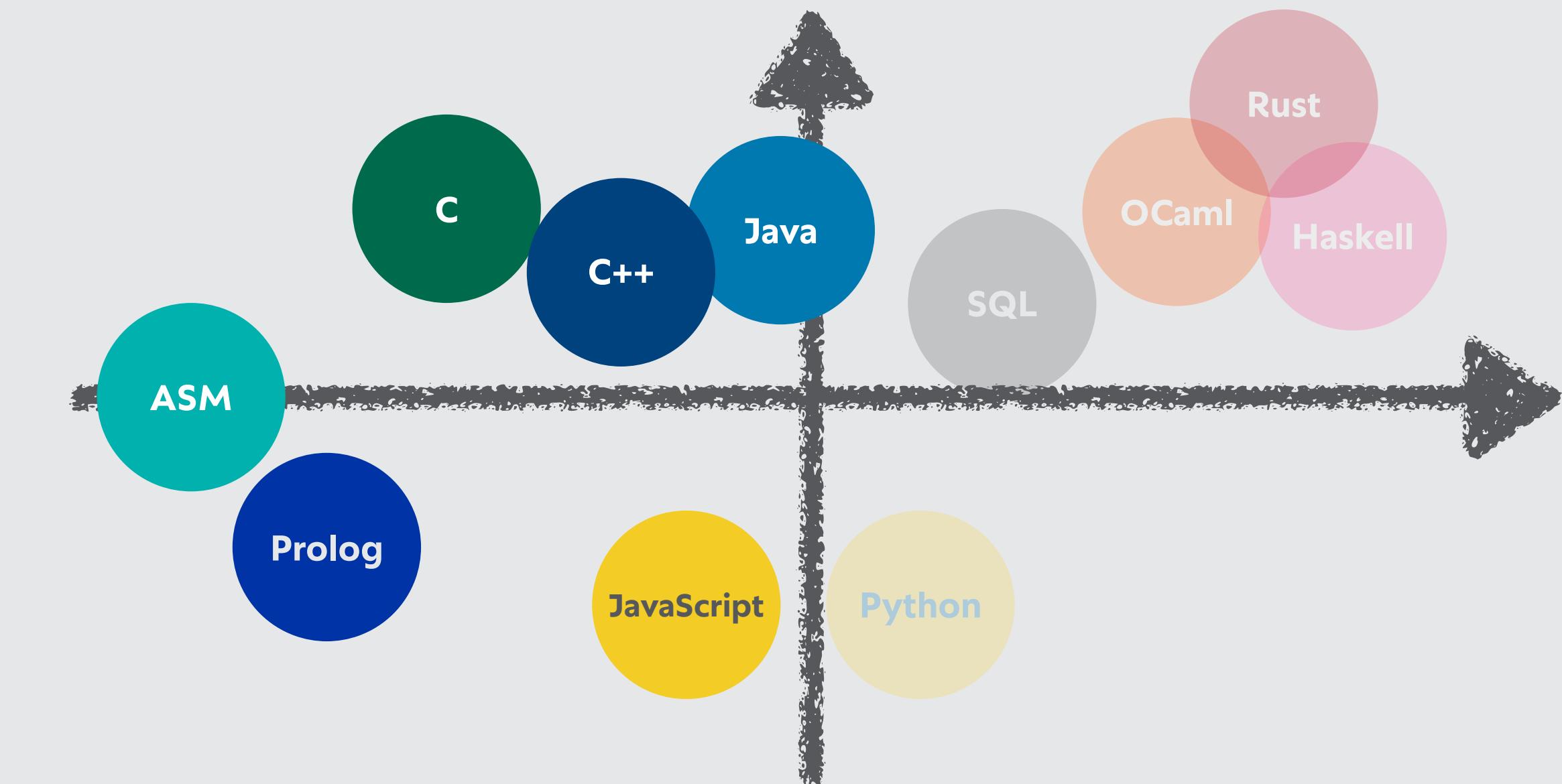
Šibkejši tipi nudijo večjo fleksibilnost

`10 + 3.14`

`10 + false`

`10 + "foo"`

```
if "abc".isdigit:  
    print("abc je številka")
```

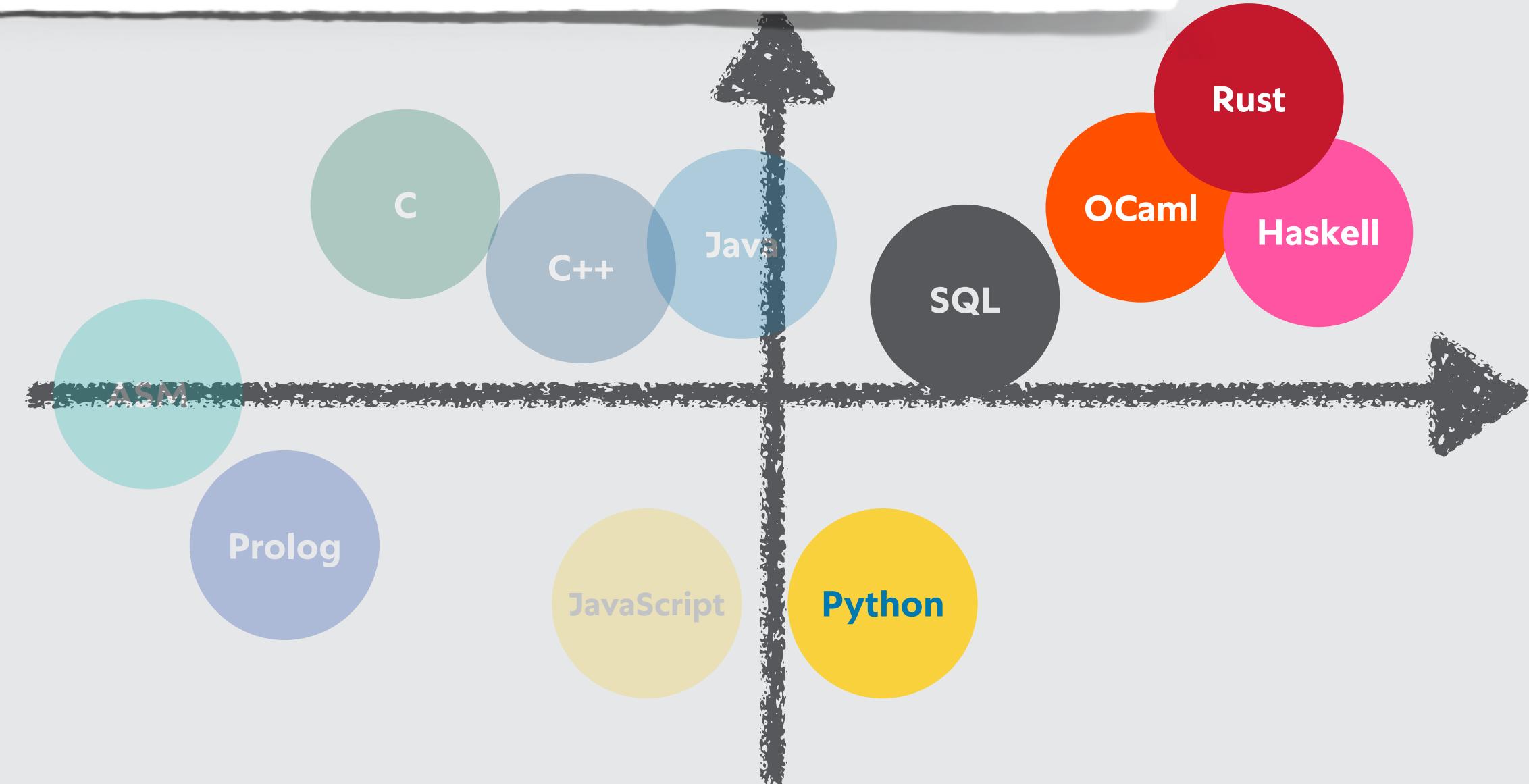


Krepkejši tipi zagotavljajo varnost med izvajanjem ohranitev

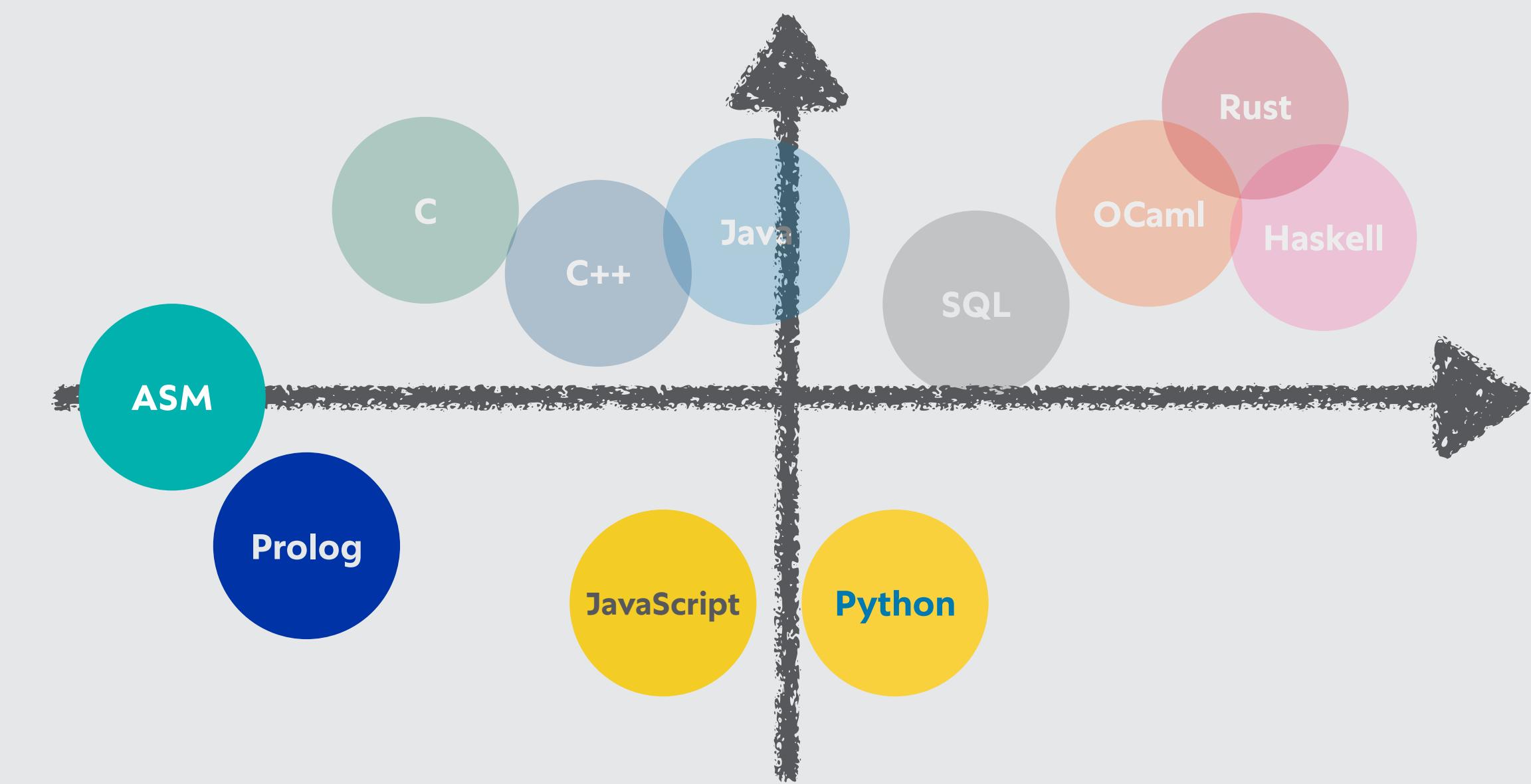
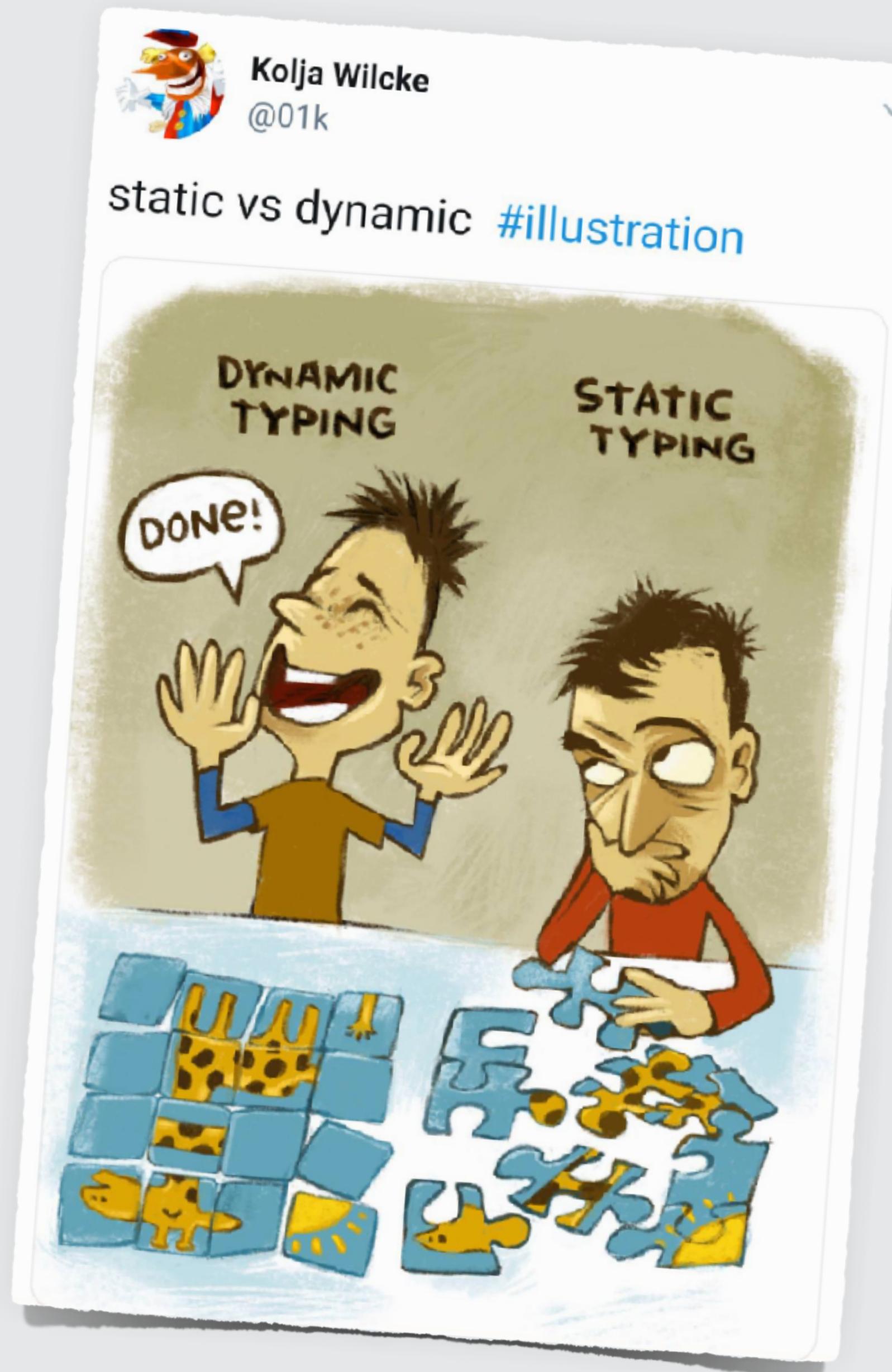
$$(\vdash e : \tau \wedge e \mapsto e') \implies \vdash e' : \tau$$

$$\vdash e : \tau \implies e = v \vee \exists e'. e \mapsto e'$$

napredek



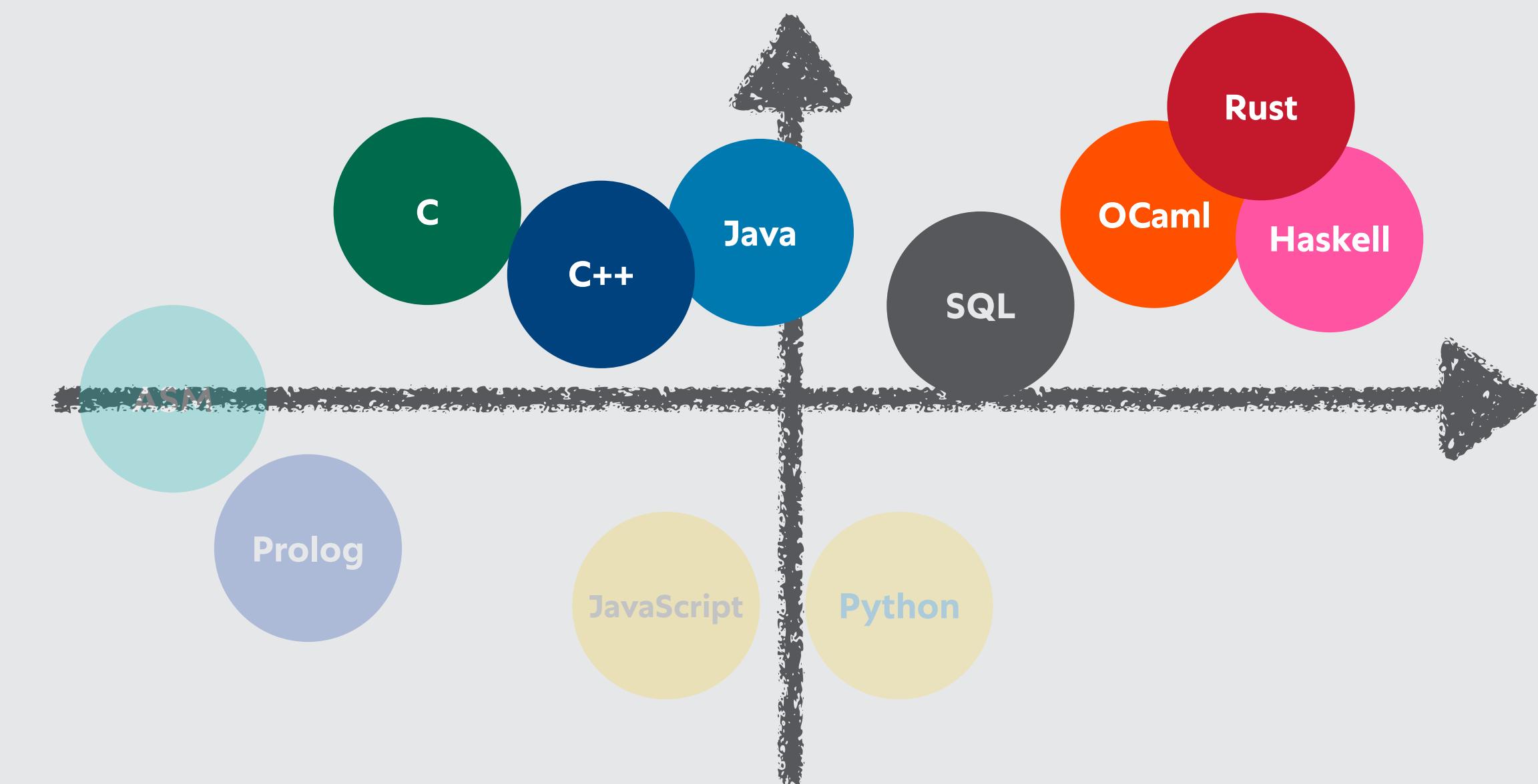
Dinamični tipi (na začetku) omogočajo hitrejše pisanje



Statični tipi (med drugim) omogočajo hitrejše izvajanje

[**(42, true), (5, false), (1, false)**]]

[**42, true, 5, false**]



Sistemi tipov se razlikujejo tudi po podpori **polimorfizmu**

primeri

Java

OCaml

Haskell

C

parametrični

compose, length, +, <, toString
map, reverse

generiki

('a -> 'b) -> 'a list -> 'b list

(a -> b) -> List a -> List b



ad-hoc

preobteževanje



razredi tipov
(konec leta)



Izpeljava tipov

Tipe lahko jezik preveri ali izpelje

Java

```
public static <A, B> List<B> map(List<A> lst, Function<A, B> f) {  
    ...  
}
```

OCaml

```
let rec map f = function  
| [] -> []  
| hd :: tl -> f hd :: map f tl  
  
val map : ('a -> 'b) -> 'a list -> 'b list
```

Izpeljavo in **preverjanje** lahko kombiniramo

brez anotacij

```
let rec map f = function
| [] -> []
| hd :: tl -> f hd :: map f tl
```

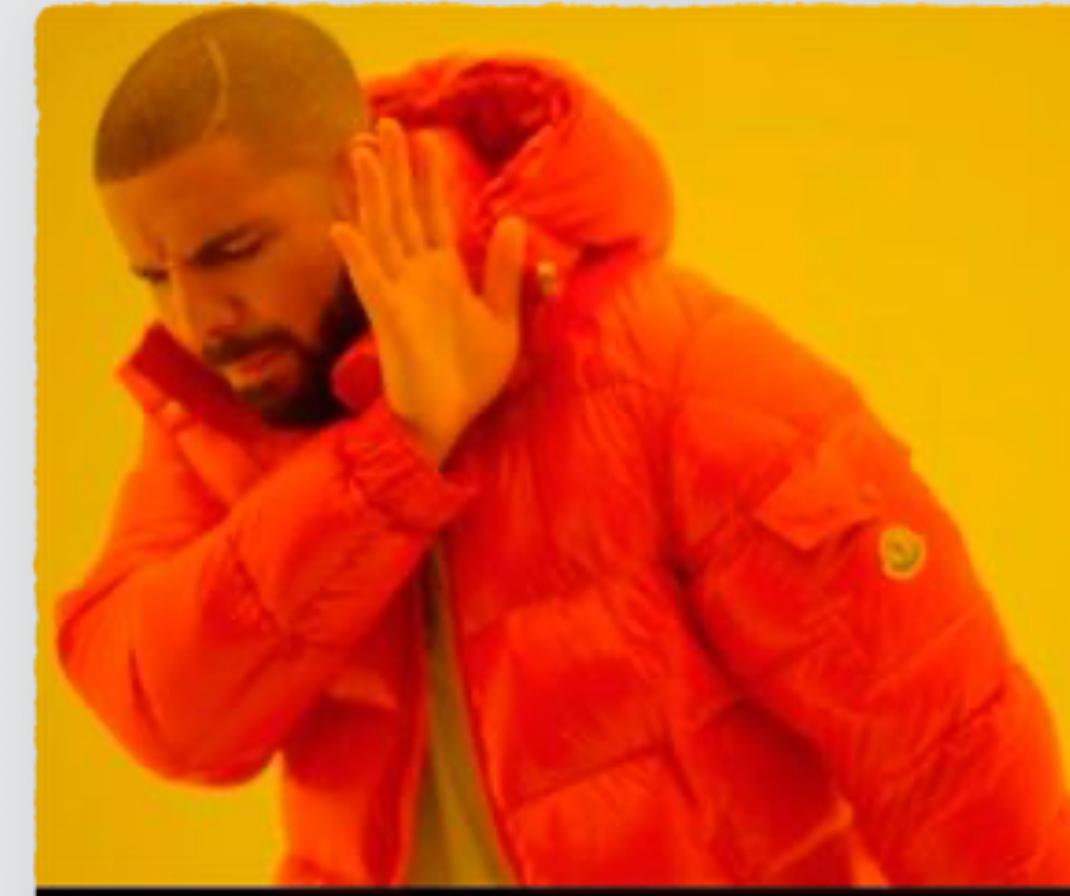
```
val map : ('a -> 'b) -> 'a list -> 'b list
```

z anotacijami

```
let rec map (f : int -> string) = function
| [] -> []
| hd :: tl -> f hd :: map f tl
```

```
val map : (int -> string) -> int list -> string list
```

Pri izpeljavi želimo najti **glavni tip** programa

$$\lambda x. \lambda y. x$$

$$\begin{array}{l} \text{int} \rightarrow \text{int} \rightarrow \text{int} \\ \text{int} \rightarrow \text{bool} \rightarrow \text{int} \\ \alpha \rightarrow \alpha \rightarrow \alpha \end{array}$$

$$\alpha \rightarrow \beta \rightarrow \alpha$$

Hindley-Milnerjev algoritem poteka v dveh fazah

$$\lambda f.f(f\ 0)$$

1. nastavimo
tip & enačbe

$$\begin{aligned}\alpha &= \gamma \rightarrow \beta \\ \gamma &= \text{int} \\ \gamma &= \beta\end{aligned}$$

$$\begin{aligned}\alpha &\mapsto (\text{int} \rightarrow \text{int}) \\ \beta &\mapsto \text{int} \\ \gamma &\mapsto \text{int}\end{aligned}$$

vstavimo rešitev v tip

$(\text{int} \rightarrow \text{int}) \rightarrow \text{int}$

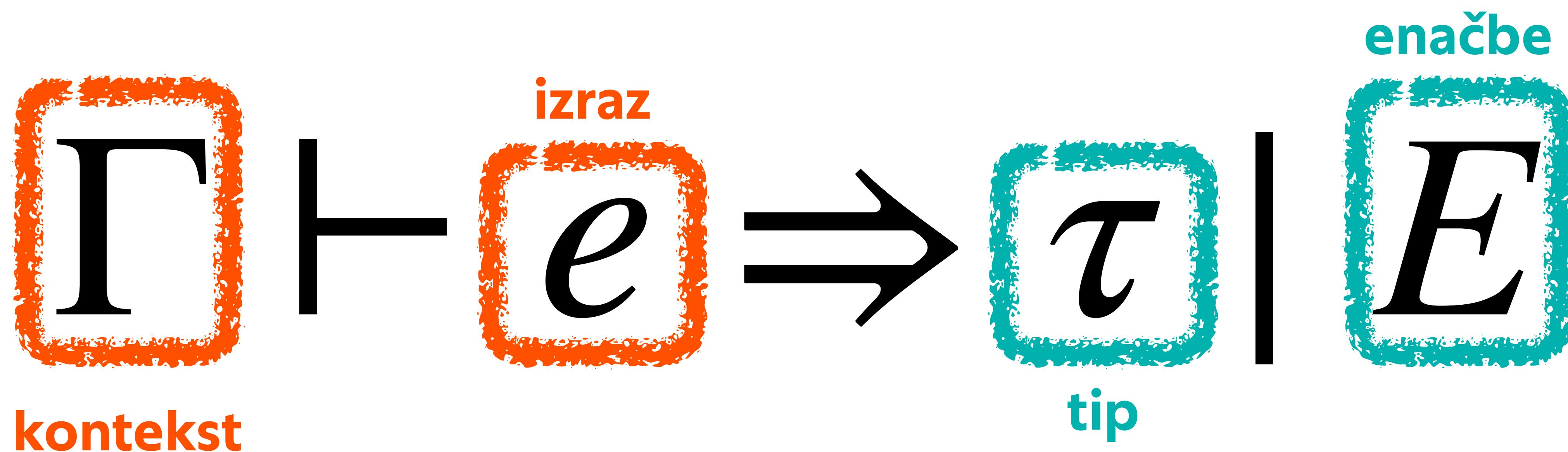
2. rešimo
enačbe

Roger Hindley, 1939–

Robin Milner, 1934–2010

© Computer
Laboratory
University of
Cambridge

✓ prvi fazi **nastavimo** tip ter **enačbe** med neznankami



$x : \text{bool}, y : \text{int}, z : \alpha$

Tip konstant določimo enostavno

$$\Gamma \vdash \text{true} \Rightarrow \text{bool} \mid \emptyset$$
$$\Gamma \vdash \text{false} \Rightarrow \text{bool} \mid \emptyset$$
$$\Gamma \vdash n \Rightarrow \text{int} \mid \emptyset$$

Tip spremenljivk preberemo iz konteksta

$$\Gamma \vdash x \Rightarrow \Gamma(x) \mid \emptyset$$

Pri **aritmetičnih/logičnih izrazih** dobimo enačbe za podizraze

$$\frac{\Gamma \vdash e_1 \Rightarrow \tau_1 | E_1 \quad \Gamma \vdash e_2 \Rightarrow \tau_2 | E_2}{\Gamma \vdash e_1 + e_2 \Rightarrow \text{int} | E_1, E_2, \tau_1 = \text{int}, \tau_2 = \text{int}}$$

$$\frac{\Gamma \vdash e_1 \Rightarrow \tau_1 | E_1 \quad \Gamma \vdash e_2 \Rightarrow \tau_2 | E_2}{\Gamma \vdash e_1 < e_2 \Rightarrow \text{bool} | E_1, E_2, \tau_1 = \text{int}, \tau_2 = \text{int}}$$

$$x : \alpha, y : \beta \vdash x < y + 1 \Rightarrow ??? \mid ???$$

Izpeljave bomo pisali **na krajše**

$$\frac{\frac{\Gamma \vdash x \Rightarrow \alpha \mid \emptyset}{\Gamma \vdash x \Rightarrow \alpha \mid \emptyset} \quad \frac{\frac{\Gamma \vdash y \Rightarrow \beta \mid \emptyset \quad \Gamma \vdash 1 \Rightarrow \text{int} \mid \emptyset}{\Gamma \vdash y + 1 \Rightarrow \text{int} \mid \beta = \text{int}, \text{int} = \text{int}}}{\Gamma \vdash x < y + 1 \Rightarrow \text{bool} \mid \beta = \text{int}, \text{int} = \text{int}, \alpha = \text{int}, \text{int} = \text{int}}$$

$$\frac{\frac{\frac{\text{bool}}{\text{int}}}{x \leftarrow \underbrace{y + 1}_{\text{int}}} \quad \alpha \quad \beta \quad \text{int}}{\alpha \quad \beta \quad \text{int}}$$

$\beta = \text{int}$
 $\text{int} = \text{int}$
 $\alpha = \text{int}$
 $\text{int} = \text{int}$

Pri pogojnem stavku se morata tipa vej ujemati

$$\frac{\Gamma \vdash e \Rightarrow \tau \mid E \quad \Gamma \vdash e_1 \Rightarrow \tau_1 \mid E_1 \quad \Gamma \vdash e_2 \Rightarrow \tau_2 \mid E_2}{\Gamma \vdash \text{if } e \text{ then } e_1 \text{ else } e_2 \Rightarrow \tau_1 \mid E, E_1, E_2, \tau = \text{bool}, \tau_1 = \tau_2}$$

Pri pari h zdržimo tipe in enačbe podizrazov

$$\frac{\Gamma \vdash e_1 \Rightarrow \tau_1 | E_1 \quad \Gamma \vdash e_2 \Rightarrow \tau_2 | E_2}{\Gamma \vdash (e_1, e_2) \Rightarrow \tau_1 \times \tau_2 | E_1, E_2}$$

Pri projekcijah je ena komponenta tipa nedoločena

$$\frac{\Gamma \vdash e \Rightarrow \tau \mid E}{\Gamma \vdash \text{fst } e \Rightarrow \alpha \mid E, \tau = \alpha \times \beta}$$

$$\frac{\Gamma \vdash e \Rightarrow \tau \mid E}{\Gamma \vdash \text{snd } e \Rightarrow \beta \mid E, \tau = \alpha \times \beta}$$

Pri **funkcijah** kontekst razširimo s svežim parametrom

$$\frac{\Gamma, x : \alpha \vdash e \Rightarrow \tau \mid E}{\Gamma \vdash \lambda x . e \Rightarrow \alpha \rightarrow \tau \mid E}$$

$$\lambda b x . \text{if } b \text{ then } x \text{ else } 0$$

Pri aplikaciji mora biti tip prvega podizraza funkcijski

$$\frac{\Gamma \vdash e_1 \Rightarrow \boxed{\tau_1} | E_1 \quad \Gamma \vdash e_2 \Rightarrow \boxed{\tau_2} | E_2}{\Gamma \vdash e_1 e_2 \Rightarrow \boxed{\alpha} | E_1, E_2, \boxed{\tau_1 = \tau_2 \rightarrow \alpha}}$$

$$\lambda fx.f(fx)$$

$$\lambda pfx.\ \text{if}\ px\ \text{then}\ fx\ \text{else}\ x$$

Pri **rekurzivnih definicijah** nastavimo rekurzivno enačbo

$$\frac{\Gamma, f : \alpha \vdash e \Rightarrow \tau \mid E}{\Gamma \vdash (\text{rec } f. e) \Rightarrow \tau \mid E, \alpha = \tau}$$

$$f = \lambda n.$$

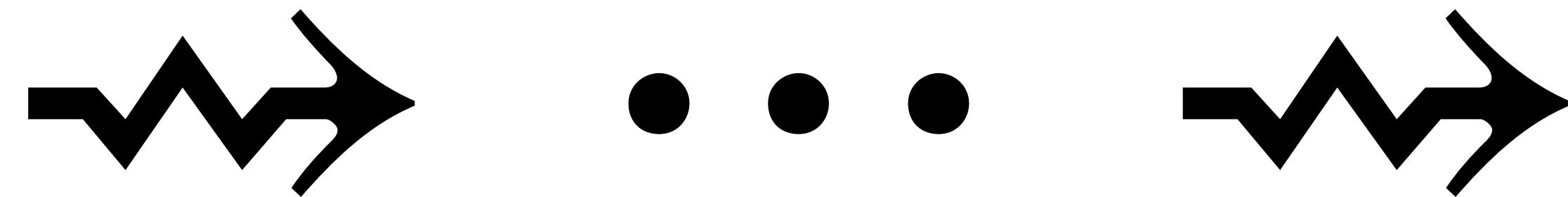
if $n = 0$ then 1

else $n * f(n - 1)$

nastavjanje enačb
v jeziku
poly

V drugi fazi iz enačb postopoma dobimo **rešitev**

$$\alpha = \text{int}, \beta = \alpha \rightarrow \alpha$$



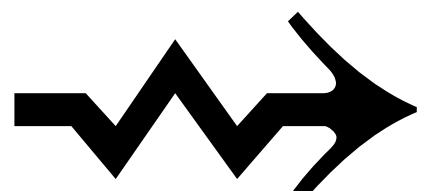
$$\alpha \mapsto \text{int}, \beta \mapsto \text{int} \rightarrow \text{int}$$

Enačbe postopoma **združujemo**, dokler jih ne zmanjka



$$\rightsquigarrow \langle E' | R' \rangle$$

$$\langle E | \emptyset \rangle$$



...



$$\langle \emptyset | R' \rangle$$

Enačbe, kjer se **obe strani ujemata**, lahko zavržemo

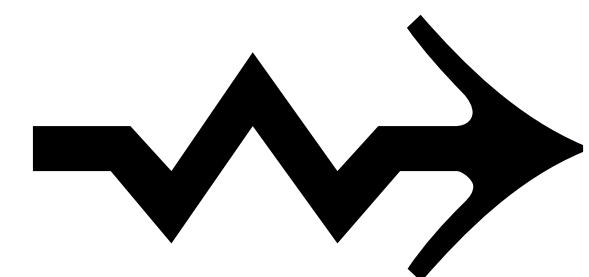
$$\langle \sigma = \sigma | E | R \rangle$$



$$\langle E | R \rangle$$

Enačbo med produktoma razbijemo na dve enačbi

$$\langle \sigma_1 \times \tau_1 = \sigma_2 \times \tau_2, E | R \rangle$$



$$\langle \sigma_1 = \sigma_2, \tau_1 = \tau_2, E | R \rangle$$

Podobno razbijemo enačbo med funkcijskima tipoma

$$\langle \sigma_1 \rightarrow \tau_1 = \sigma_2 \rightarrow \tau_2, E | R \rangle$$

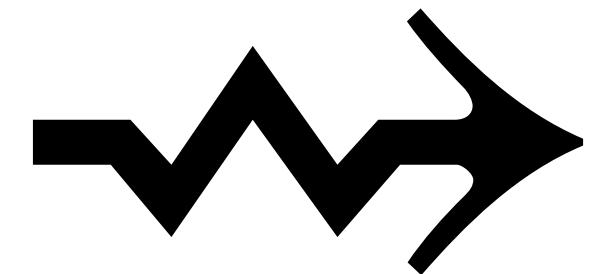


$$\langle \sigma_1 = \sigma_2, \tau_1 = \tau_2, E | R \rangle$$

Če ima enačba na **eni strani parameter**, jo dodamo* med **rešitve**

$$\langle \alpha = \tau | E | R \rangle$$

če α ne nastopa v τ



$$\langle E[\tau/\alpha] | \alpha \mapsto \tau, R[\tau/\alpha] \rangle$$

V ostalih primerih enačbe nimajo rešitve

	a	bool	int	$\sigma' \times \tau'$	$\sigma' \rightarrow \tau'$
a	+	+	+	!!	!!
bool	+	+	✗	✗	✗
int	+	✗	+	✗	✗
$\sigma \times \tau$!!	✗	✗	🚫%	✗
$\sigma \rightarrow \tau$!!	✗	✗	✗	🚫%

$$\lambda pfx. \text{ if } px \text{ then } fx \text{ else } x$$

$$f = \lambda n.$$

if $n = 0$ then 1

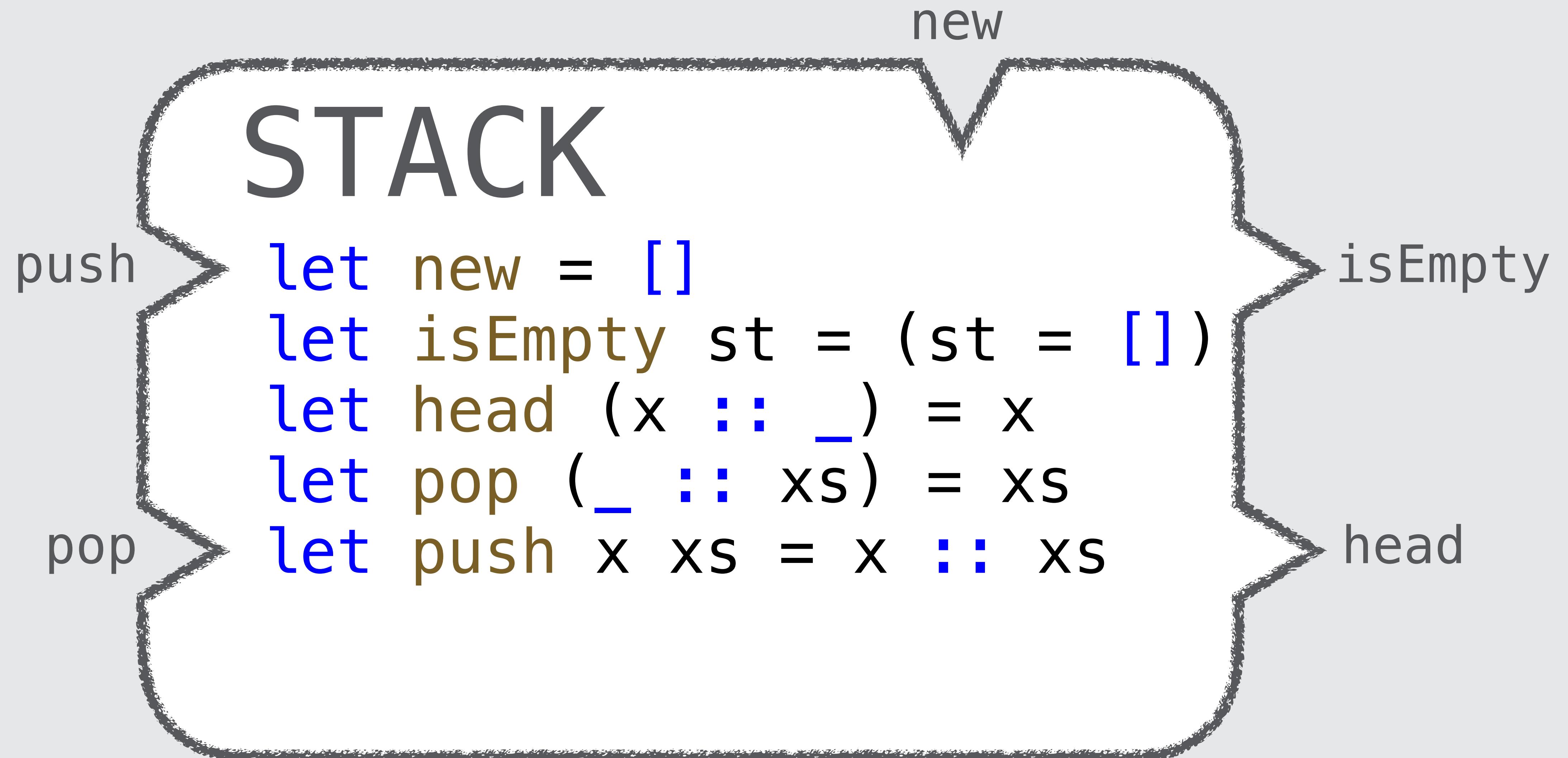
else $n * f(n - 1)$

$\lambda x . x x$

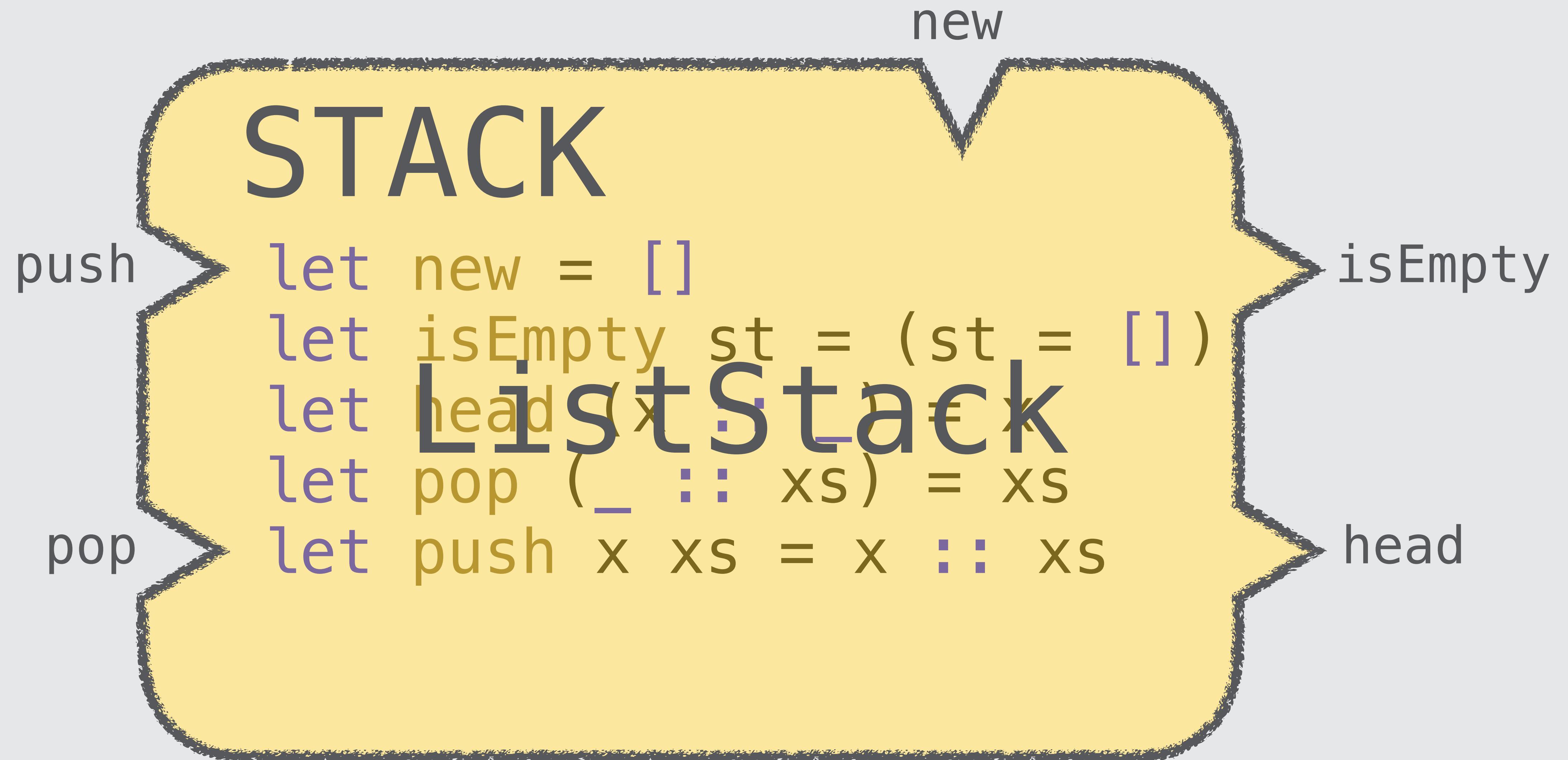
zdravovanje v jeziku
poly

prihodnjič...

Videli bomo, kako podajamo specifikacije in implementacije



Implementacije bomo skrivali z abstrakcijo



Ogledali si bomo generično programiranje

ListStack

DFS

ArrayList