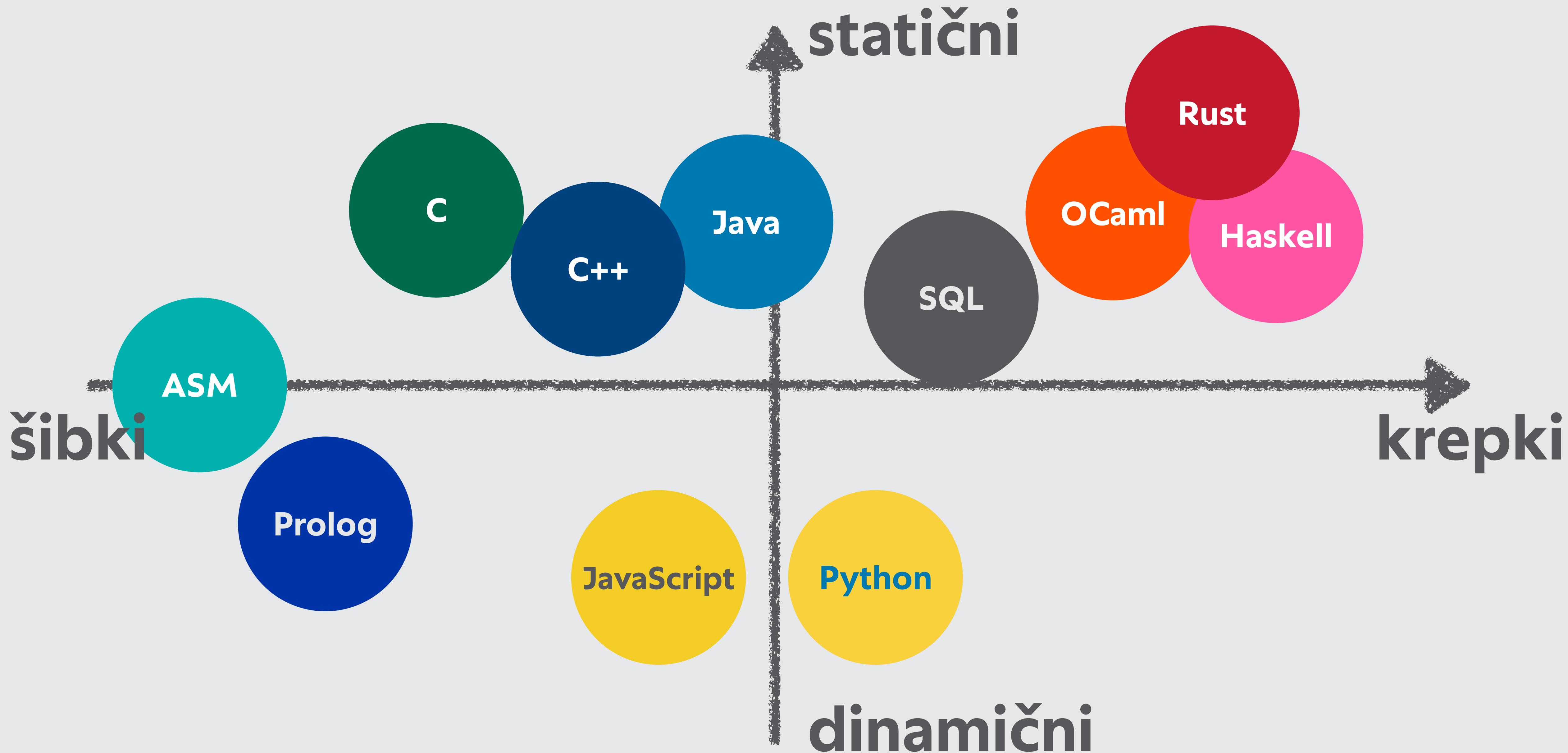


# Abstrakcija

prejšnjič...

# Različni sistemi tipov služijo različnim namenom



# 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
```

# 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

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

$\lambda x . x x$

$$f = \lambda n.$$

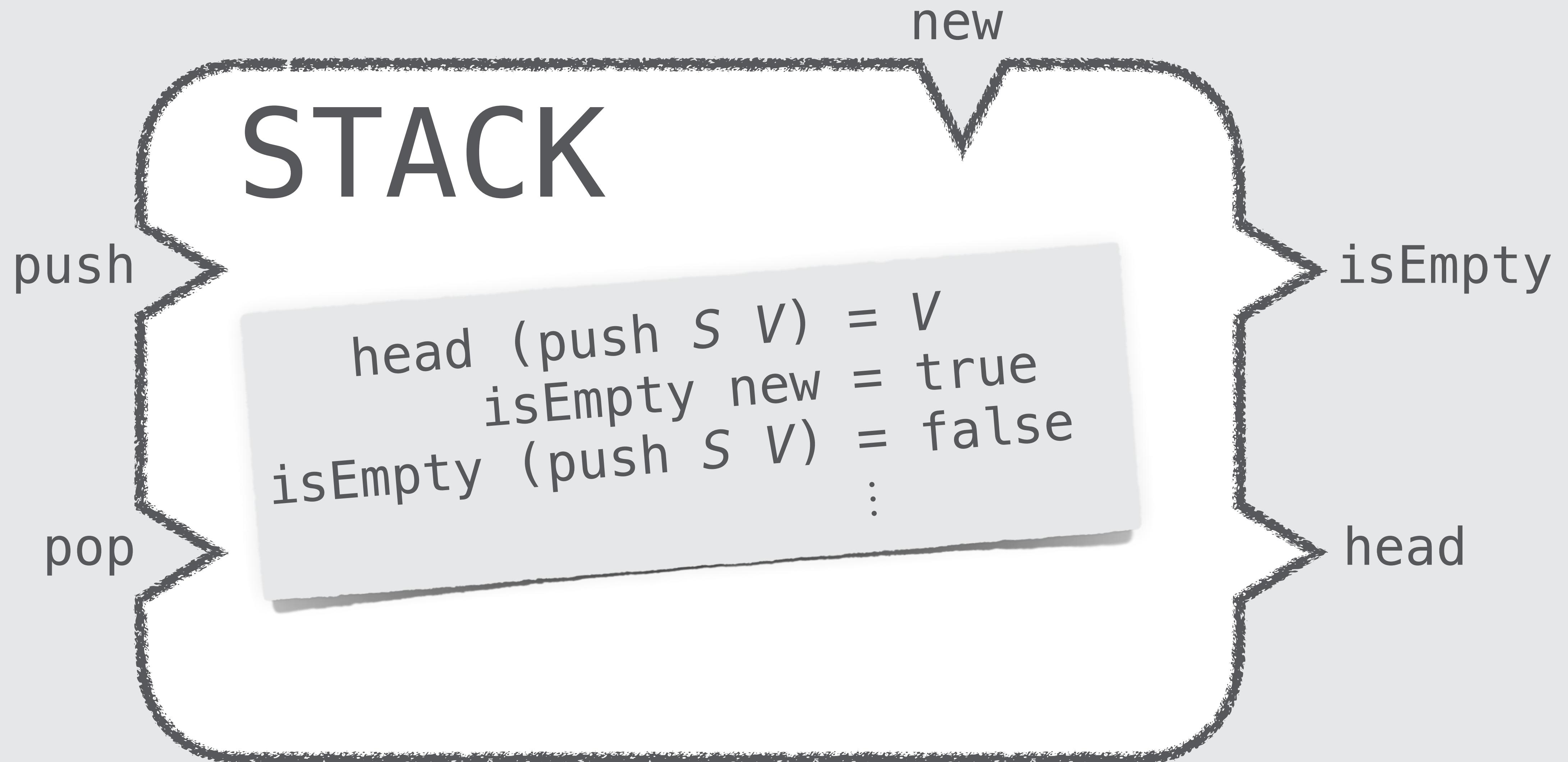
if  $n = 0$  then 1

else  $n * f(n - 1)$

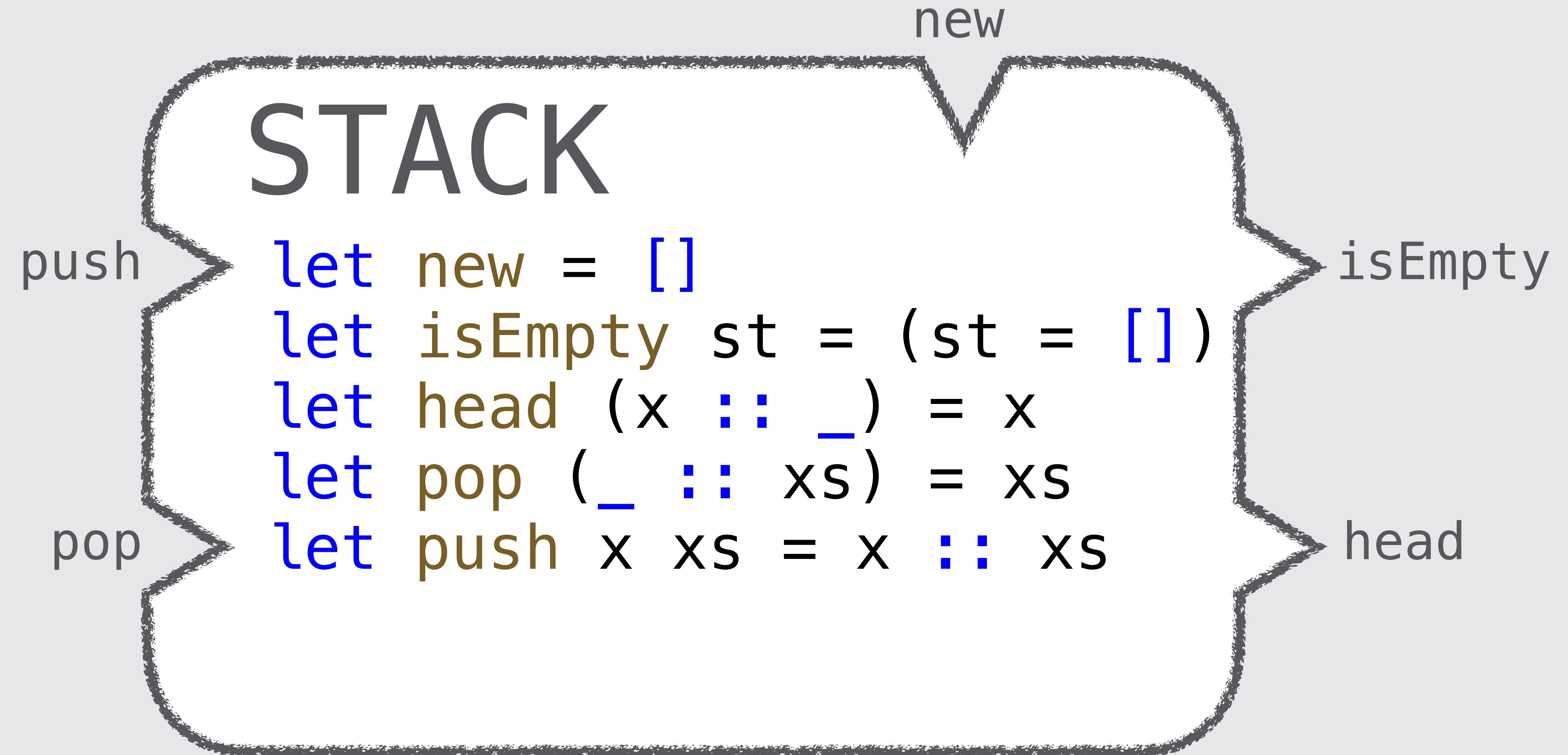
**izpeljava tipov v jeziku  
pohy**

# Specifikacije & implementacija

# Specifikacija opisuje zahteve po delovanju



# Implementacija je izdelek, ki zadošča specifikaciji



Tudi **matematične strukture** imajo specifikacijo in implementacije

## specifikacija

množica  $G$

enota  $e : G$

inverz  $\square^{-1} : G \rightarrow G$

množenje  $\square \bullet \square : G \times G \rightarrow G$

## signatura

## aksiomi

$$x \bullet (y \bullet z) = (x \bullet y) \bullet z$$

$$x \bullet e = e \bullet x = x$$

$$x \bullet x^{-1} = x^{-1} \bullet x = e$$

## implementacije

$$G = \mathbb{Z}$$

$$e = 0$$

$$n^{-1} = -n$$

$$m \bullet n = m + n$$

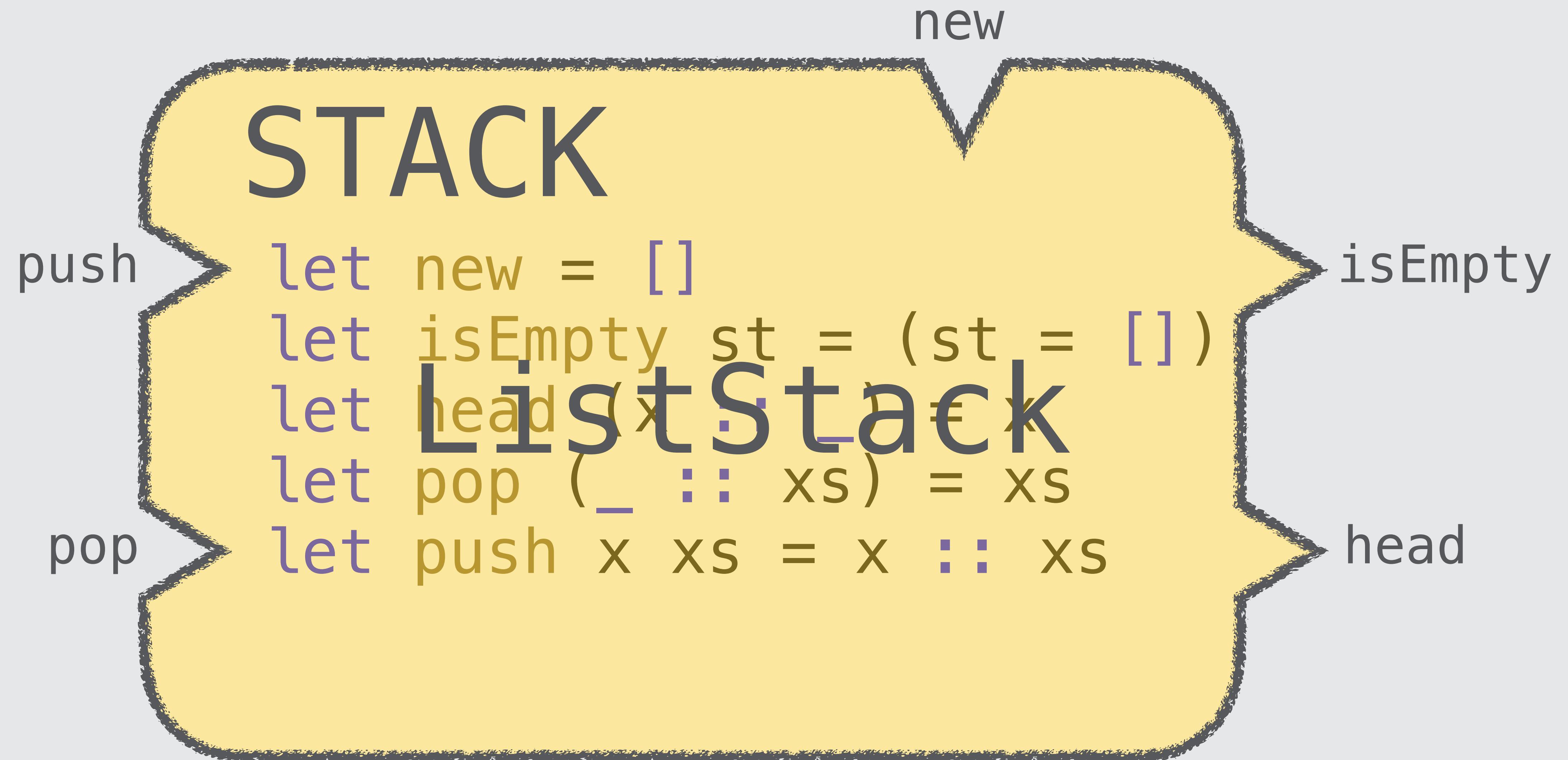
$$G = \mathbb{R}^+$$

$$e = 1$$

$$x^{-1} = 1/x$$

$$x \bullet y = xy$$

# Abstraktna implementacija skriva svoje podrobnosti



# Prednosti abstraktnih implementacij

## varnost

- robustnost
- zagotavljanje invariant



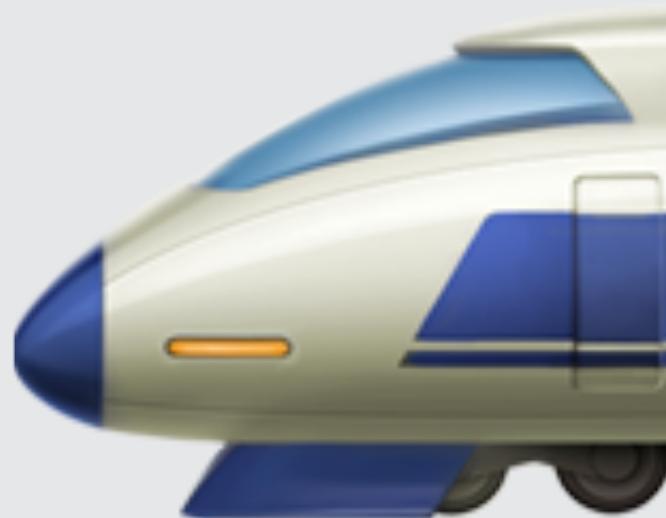
## lažji razvoj

- hkratno delo
- delitev odgovornosti



## hitrost

- ločeno prevajanje
- lažje optimizacije



## preglednost

- skrivanje detajlov
- organizacija kode

Jeziki specifikacijam in implementacijam nudijo **različno podporo**

## specifikacija implementacija abstrakcija

**Java**

vmesniki

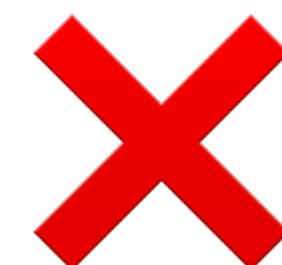
razredi

private  
public  
protected

**C**

\*.h

\*.c



**Python**



\*.py

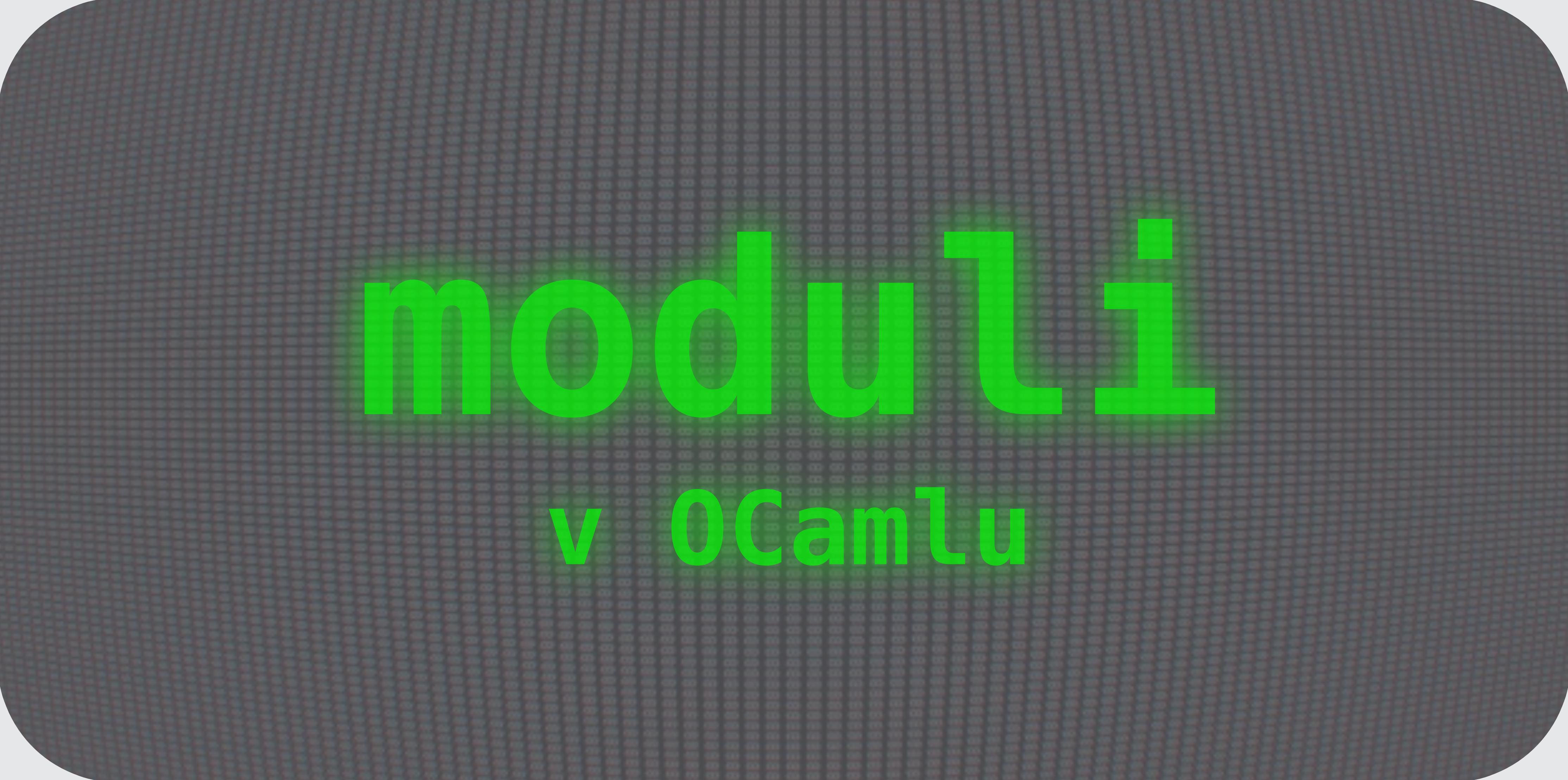
\_ime\_metode 😊

**OCaml**

signature  
\*.mli

moduli  
\*.ml





**modul**  
**vocam**

# Generično programiranje

Dve abstraktni implementaciji iste specifikacije sta **zamenljivi**

DFS

ListStack

ArrayStack

V Javi generično programiranje dosežemo z **omejenimi generiki**

```
public class DFS<S extends Stack> {  
    ...  
}
```

funktörji  
vocabu

Projekt **MirageOS** s pomočjo funkторjev zgradi t.i. *unikernel*



**prihodnjič...**

S Hornovimi formulami bomo izražali računske probleme

$$\forall n . \text{vsota}(n, Z, n)$$

$$\forall k, m, n . \text{vsota}(k, m, n) \Rightarrow \text{vsota}(k, S(m), S(n))$$

$$\exists^? n . \text{vsota}(S(Z), S(Z), n)$$

$$\exists^? n . \text{vsota}(S(Z), n, S(S(Z)))$$

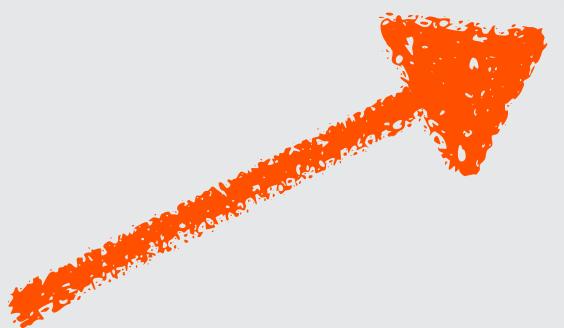
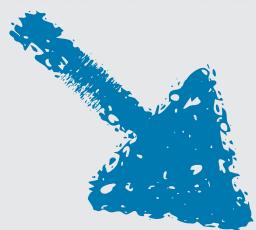
# Logične naloge bomo reševali z **združevanjem**

$$\forall n . \text{vsota}(n, Z, n)$$

$$\forall k, m, n . \text{vsota}(k, m, n) \Rightarrow \text{vsota}(k, S(m), S(n))$$

$$\text{vsota}(S(Z), n, S(S(Z)))$$

$$n' = Z$$



$$n = S(n') \wedge \text{vsota}(S(Z), n', S(Z))$$

# Začeli bomo spoznavati Prolog

```
vsota(X, z, X).  
vsota(X, s(Y), s(Z)) :-  
    vsota(X, Y, Z).  
  
?- vsota(s(z), s(z), N).  
N = s(s(z)).  
  
?- vsota(s(z), N, s(s(z))).  
N = s(z).
```

