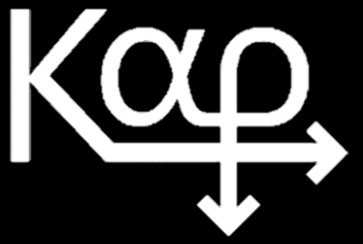
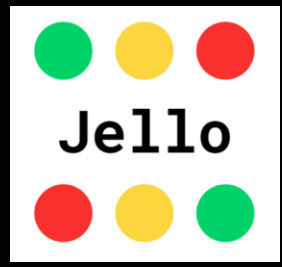






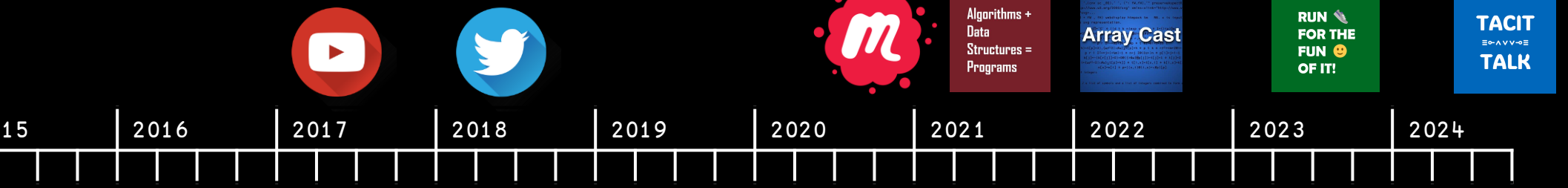
The Power of Function Composition

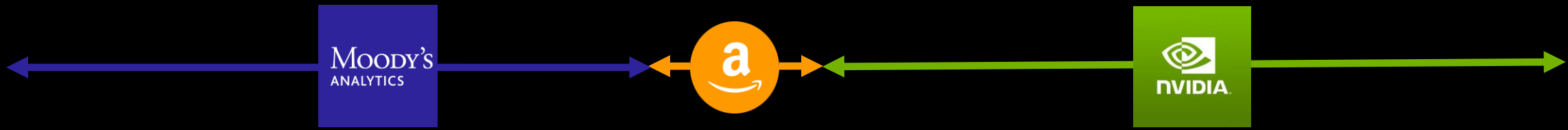


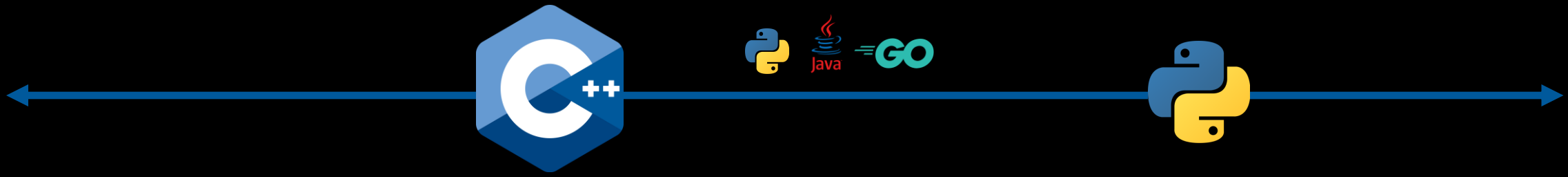



Conor Hoekstra

code_report | codereport

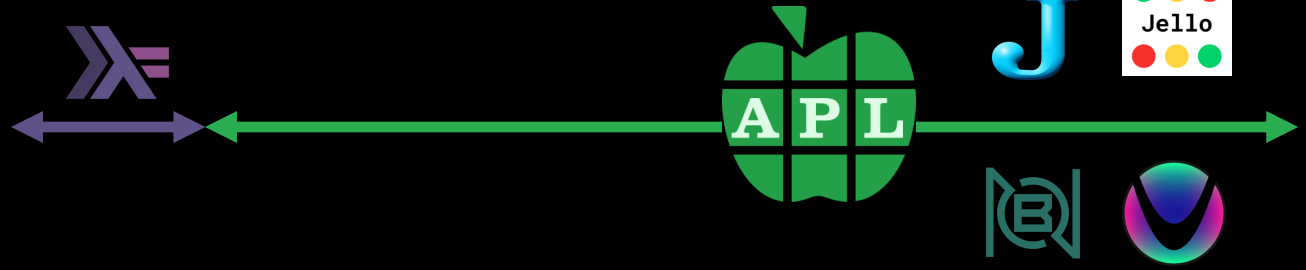






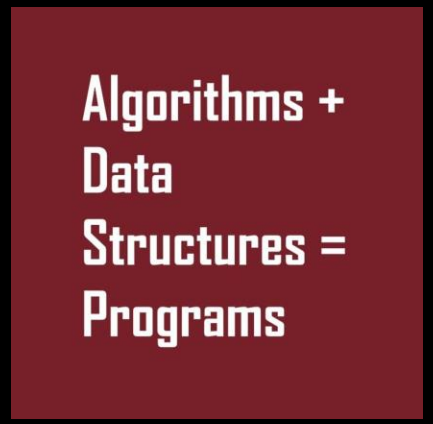
About Me

Conor Hoekstra / @code_report

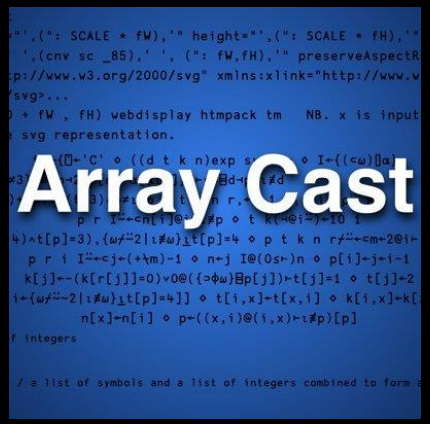


351 Videos

40 (28) Talks



203 Episodes
@adspthepodcast



89 Episodes
@arraycast

5 Episodes
@codereport

20 Episodes
@conorhoekstra

code_report

@code_report · 60.8K subscribers · 352 videos

Welcome to the code_report YouTube channel. ...more

twitter.com/code_report and 3 more links

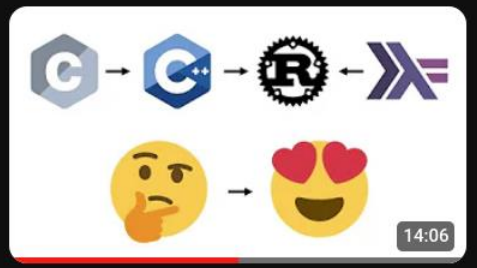
Customize channel | Manage videos

Home | Videos | Live | Podcasts | Playlists | Community

Latest | Popular | Oldest

1 Problem, 24 Programming Languages

375K views • 1 year ago



From C ➡️ C++ ➡️ Rust

170K views • 1 year ago

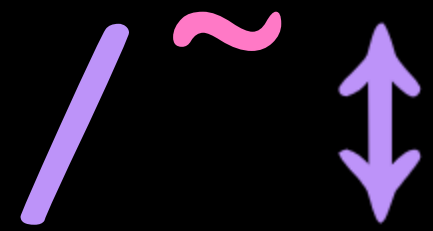
1 Problem, 16 Programming Languages (C++ vs Rust vs Haskell vs Python vs APL...)

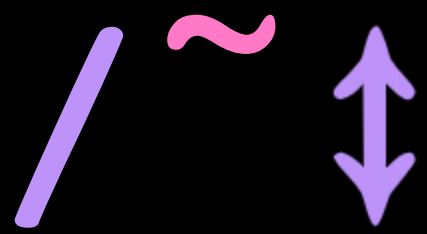
158K views • 3 years ago

Functional vs Array Programming

131K views • 3 years ago

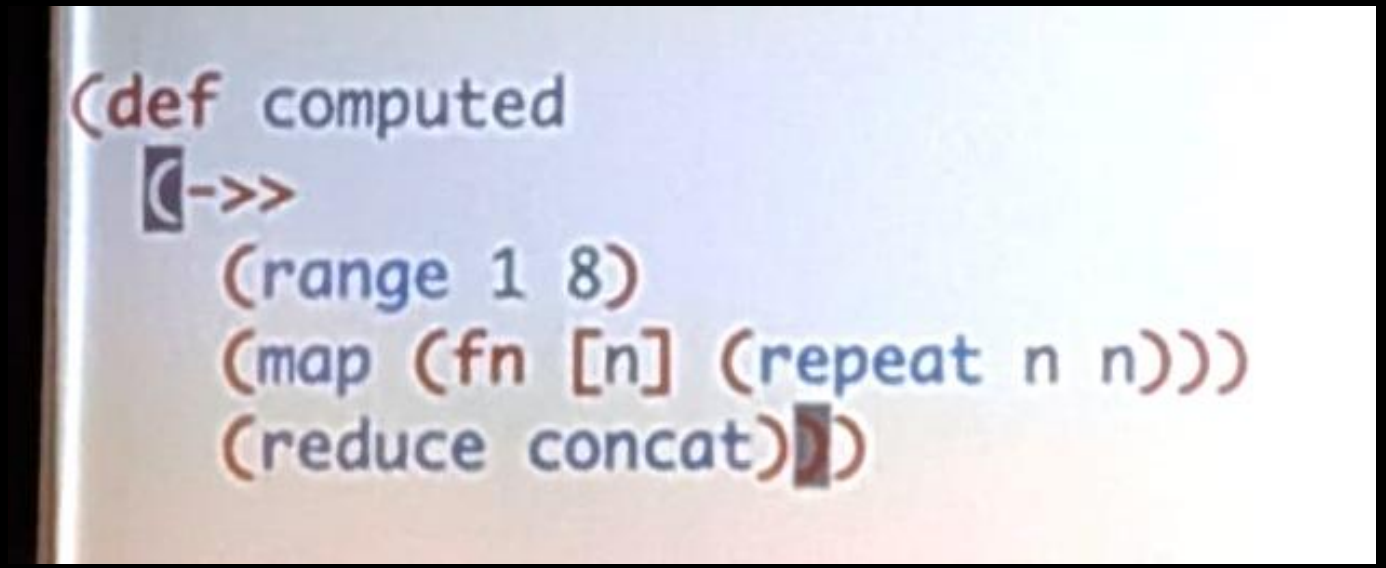
https://github.com/codereport/Content



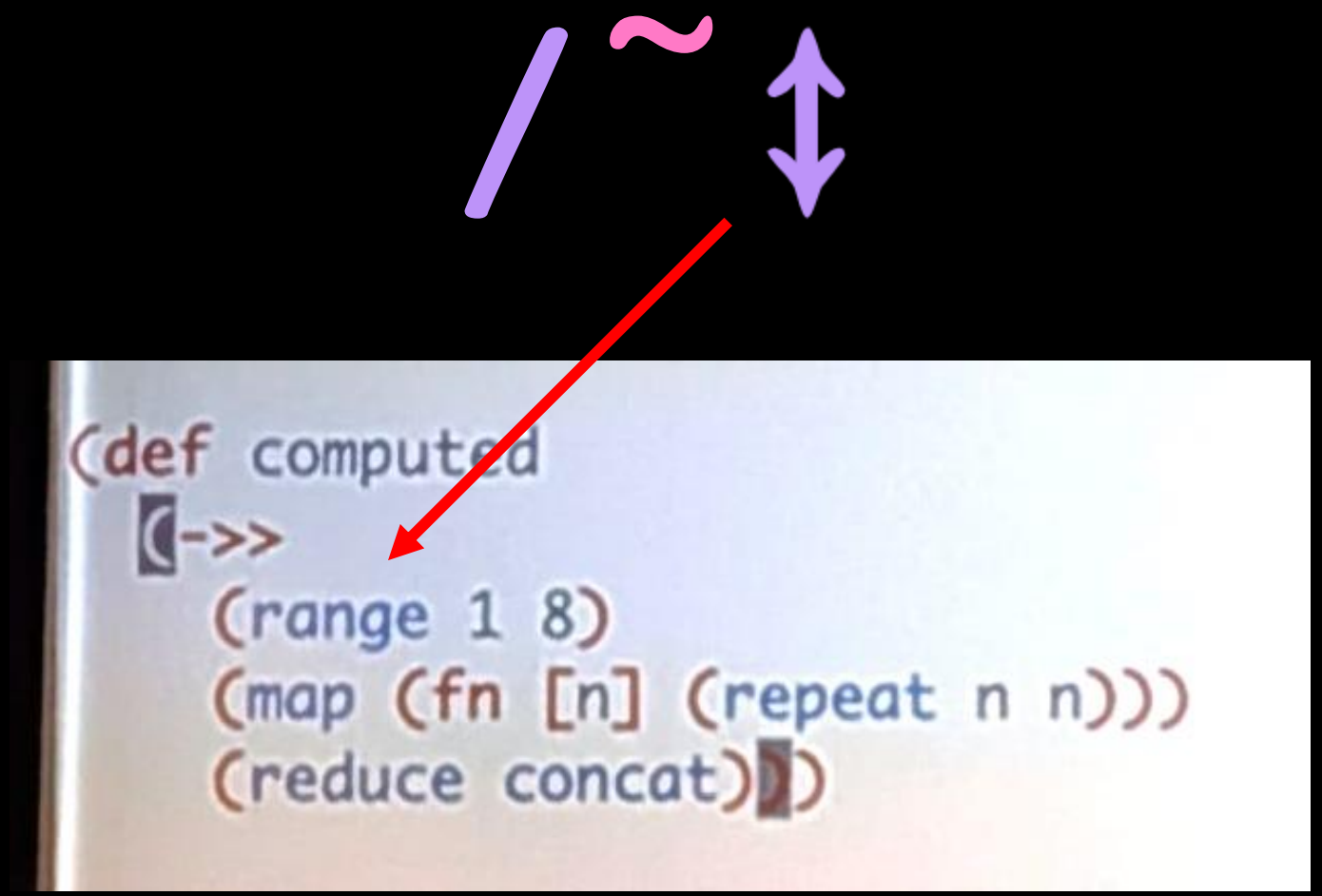


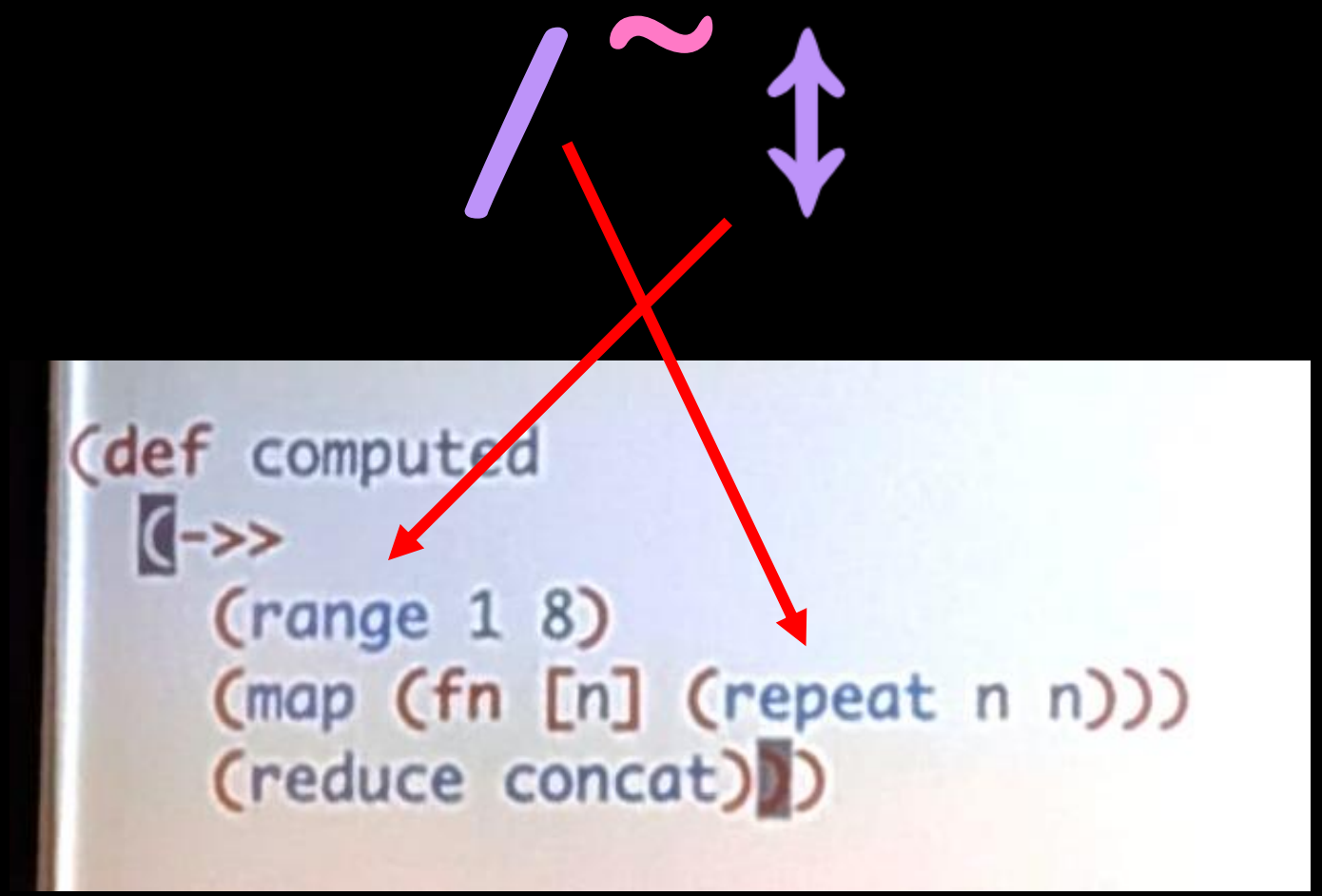
```
(def computed
  (->>
    (range 1 8)
    (map (fn [n] (repeat n n)))
    (reduce concat)))
```

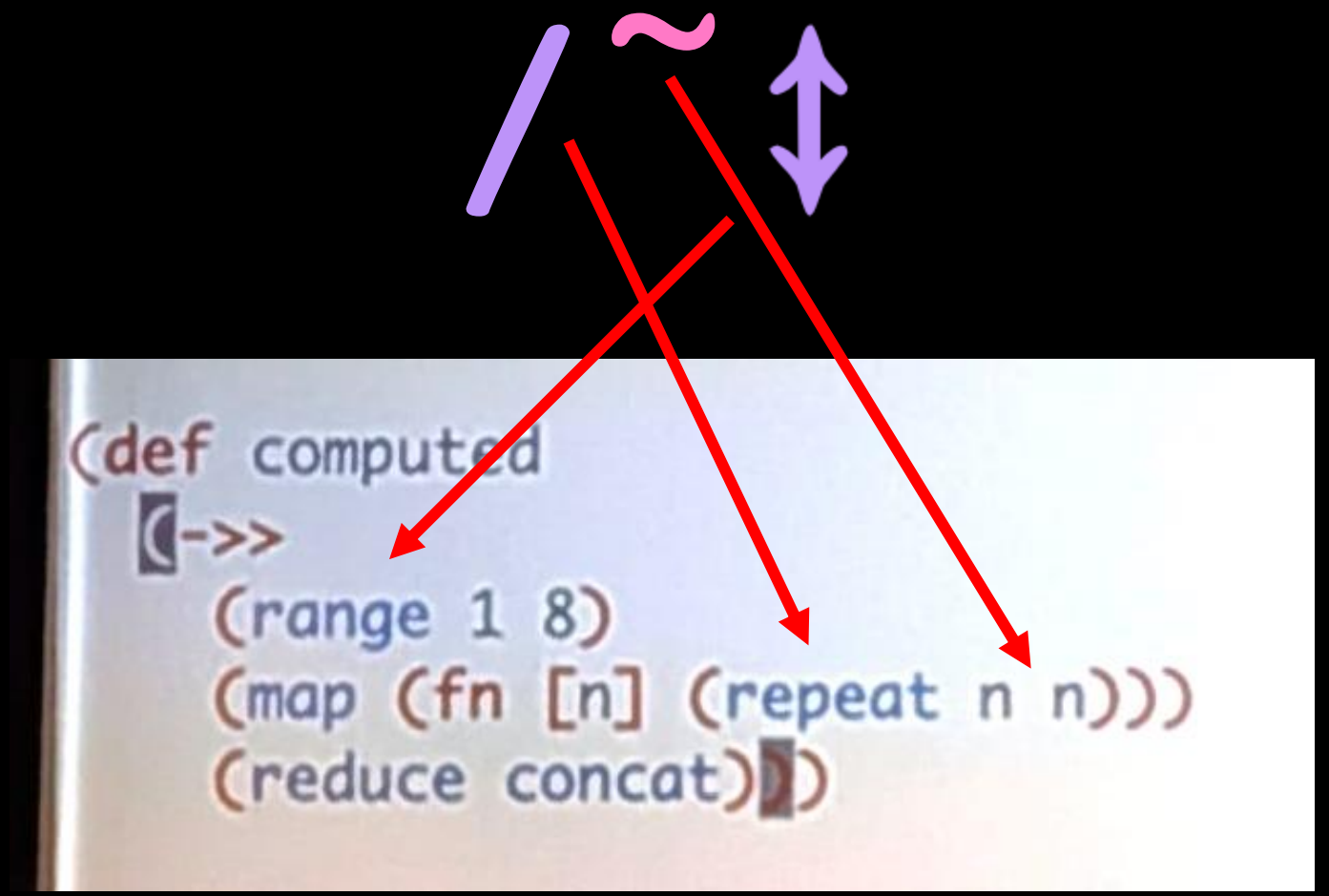
/~↕



```
(def computed
  (->>
    (range 1 8)
    (map (fn [n] (repeat n n)))
    (reduce concat)))
```


(def computed
(->>
(range 1 8)
(map (fn [n] (repeat n n)))
(reduce concat)))


(def computed
(->>
(range 1 8)
(map (fn [n] (repeat n n)))
(reduce concat)))


(def computed
(->>
(range 1 8)
(map (fn [n] (repeat n n)))
(reduce concat)))

Function Composition

Function Composition

1. Operators
2. Functions
3. Trains
4. Chains
5. Stacks*

FOF	APL	Kap	J	BQN	Jelly	Uiua	Haskell
W	⍨	⍨	~	˜	`	.	join
C	⍨	⍨	~	˜	@	:	flip
B	∘⍤⍤	⍤	@:&:	∘○	*	*	.
B_1	⍤	*	@:	∘	*	*	.:
S		∘	*	⊸		*	ap / <*>
Σ		⍛∘		⊸	*	*	=<<
D	∘	∘	*	⊸		*	
Δ		⍛∘		⊸	*	*	
Ψ	⍤	⍤⍨	&:	○	*	∩	on
D_2		a⍛∘b∘c		a⊸b⊸c		⊓	
Φ	*	a«b»c	*	*	*	⊃	liftA2
$Φ_1$	*	a«b»c	*	*	*	⊃	

Operators Functions Trains Chains Stacks*

FOF	APL	Kap	J	BQN	Jelly	Uiua	Haskell
W	⍨	⍨	~	˜	`	.	join
C	⍨	⍨	~	˜	@	:	flip
B	∘⍤⍤	⍤	@:&:	∘○	*	*	.
B_1	⍤	*	@:	∘	*	*	.:
S		∘	*	⊸		*	ap / <*>
Σ		⍥∘		⊸	*	*	=<<
D	∘	∘	*	⊸		*	
Δ		⍥∘		⊸	*	*	
Ψ	⍤	⍤⍨	&:	○	*	∩	on
D_2		a∘b∘c		a⊸b⊸c		⊓	
Φ	*	a«b»c	*	*	*	⊃	liftA2
$Φ_1$	*	a«b»c	*	*	*	⊃	

Operators Functions Trains Chains Stacks*

FOF	APL	Kap	J	BQN	Jelly	Uiua	Haskell
W	⍨	⍨	~	~	`	.	join
C	⍨	⍨	~	~	@	:	flip
B	∘⍥⍥	⍥	@:&:	∘○	*	*	.
B_1	⍥	*	@:	∘	*	*	.:
S		∘	*	⍟		*	ap / <*>
Σ		⍛∘		⍟⊸	*	*	=<<
D	∘	∘	*	⍟		*	
Δ		⍛∘		⍟⊸	*	*	
Ψ	⍥	⍥	&:	○	*	∩	on
D_2		a⍛∘b∘c		a⍟⊸b⊸c		⊓	
Φ	*	a«b»c	*	*	*	⊃	liftA2
$Φ_1$	*	a«b»c	*	*	*	⊃	

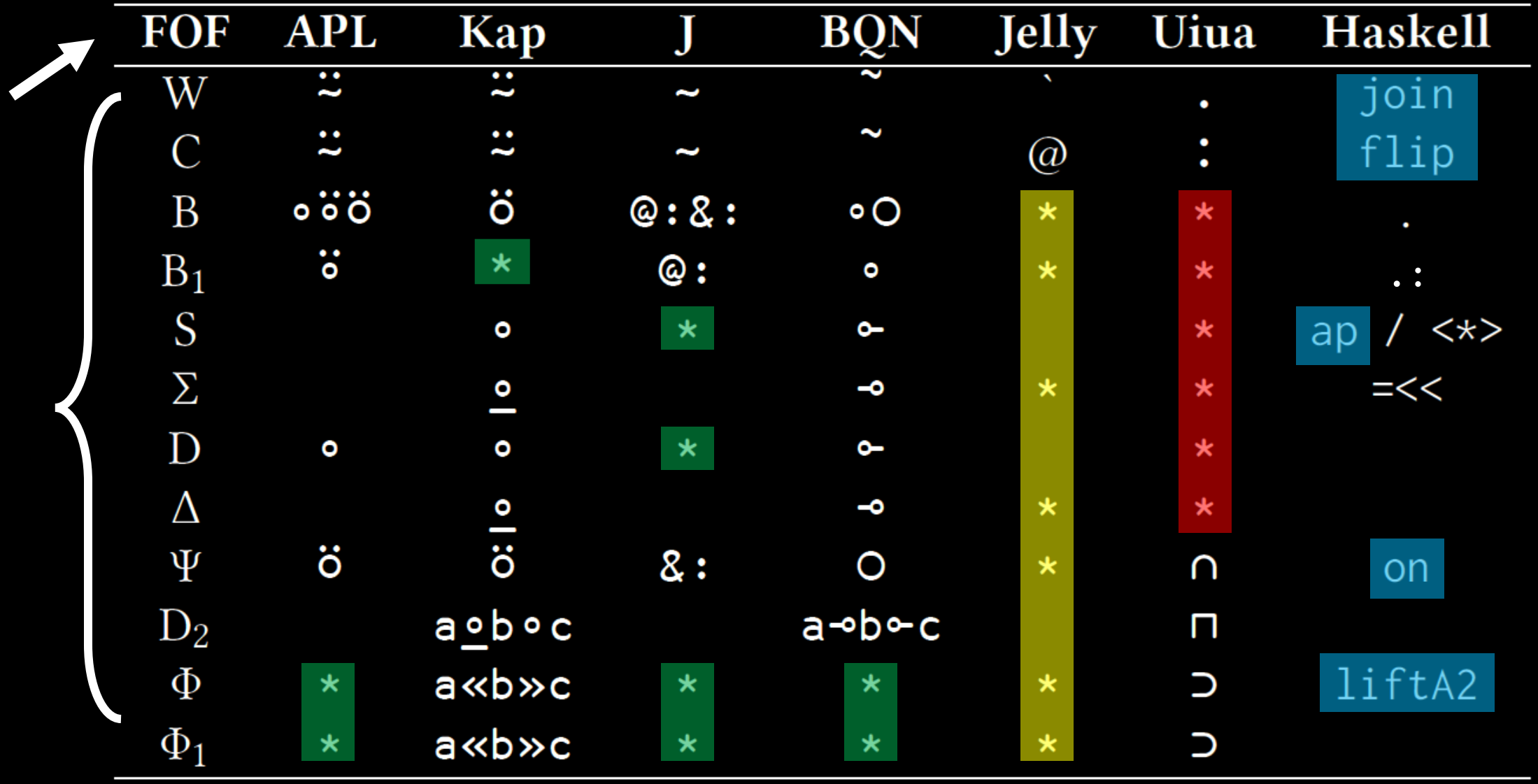
Operators Functions Trains Chains Stacks*

FOF	APL	Kap	J	BQN	Jelly	Uiua	Haskell
W	⍨	⍨	~	˜	`	.	join
C	⍨	⍨	~	˜	@	:	flip
B	∘⍥⍥	⍥	@:&:	∘○	*	*	.
B_1	⍥	*	@:	∘	*	*	.:
S		∘	*	⊸		*	ap / <*>
Σ		⍛∘		⊸	*	*	=<<
D	∘	∘	*	⊸		*	
Δ		⍛∘		⊸	*	*	
Ψ	⍥	⍥:	&:	○	*	∩	on
D_2		a⍛∘b∘c		a⊸b⊸c		⊓	
Φ	*	a«b»c	*	*	*	⊃	liftA2
$Φ_1$	*	a«b»c	*	*	*	⊃	

Operators Functions Trains Chains Stacks*

FOF	APL	Kap	J	BQN	Jelly	Uiua	Haskell
W	⍨	⍨	~	˜	`	.	join
C	⍨	⍨	~	˜	@	:	flip
B	∘⍥⍥	⍥	@:&:	∘○	*	*	.
B_1	⍥	*	@:	∘	*	*	.:
S		∘	*	⊸		*	ap / <*>
Σ		⍛∘		⊸	*	*	=<<
D	∘	∘	*	⊸		*	
Δ		⍛∘		⊸	*	*	
Ψ	⍥	⍥	&:	○	*	∩	on
D_2		a⍛∘b∘c		a⊸b⊸c		⊓	
Φ	*	a«b»c	*	*	*	⊃	liftA2
$Φ_1$	*	a«b»c	*	*	*	⊃	

Operators Functions Trains Chains Stacks*


FOF APL Kap J BQN Jelly Uiua Haskell
W ¨˜ ¨˜ ˜ ˜ ` . join
C ¨˜ ¨˜ ˜ ˜ @ : flip
B ∘¨∘¨ ∘¨ @:&: ∘○ * * .
B1 ¨ * @: ∘ * * .:
S ∘ * ⍥ * ap / <*>
Σ ∘‿∘ ⍥ * * =<<
D ∘ ∘ * ⍥ *
Δ ∘‿∘ ⍥ * *
Ψ ¨ ¨‿¨ &: ○ * ∩ on
D2 a∘‿b∘c a⍥b⍥c ⊓
Φ * a«b»c * * * ⊃ liftA2
Φ1 * a«b»c * * * ⊃
Operators Functions Trains Chains Stacks*

combinator: a function that deals only in its arguments

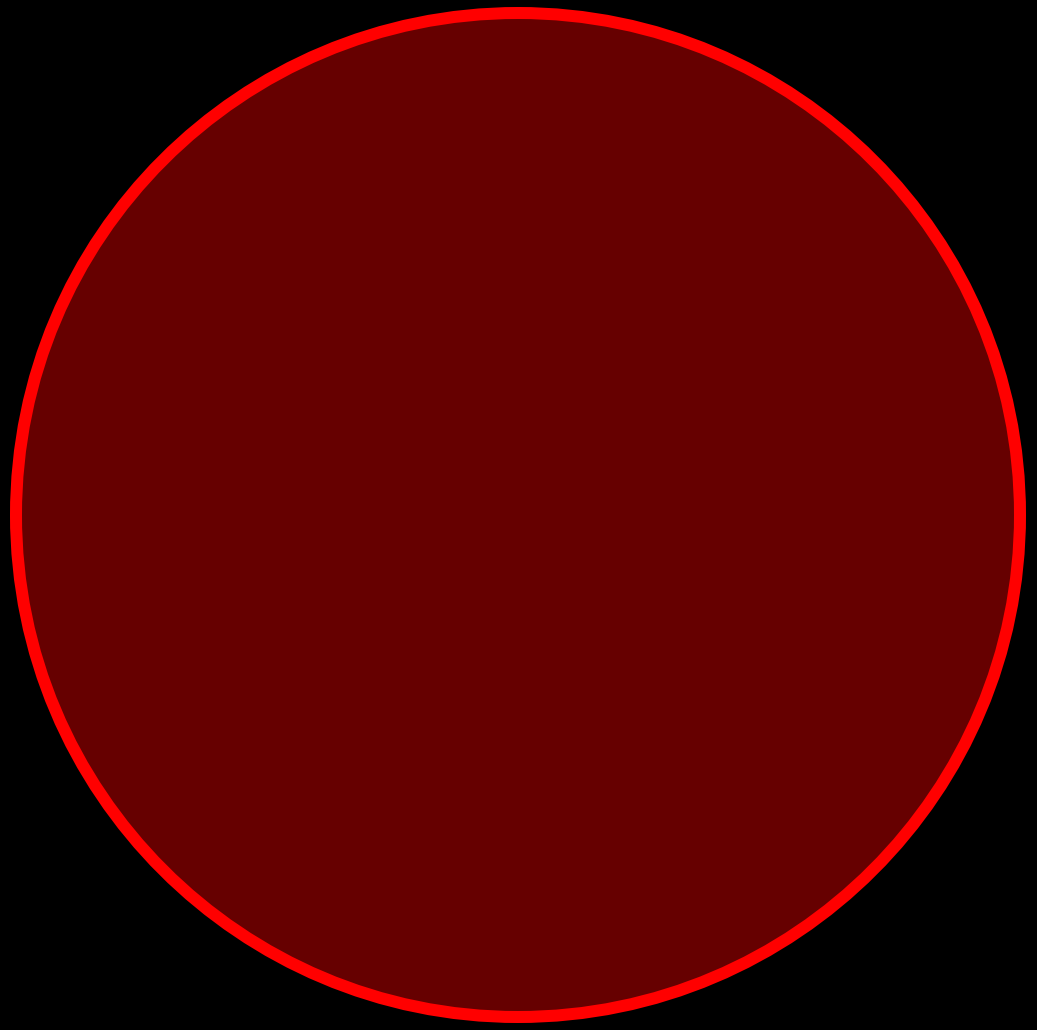
CL combinator: a combinator from Combinatory Logic

FOF: a combinator that only consumes **AND** produces functions

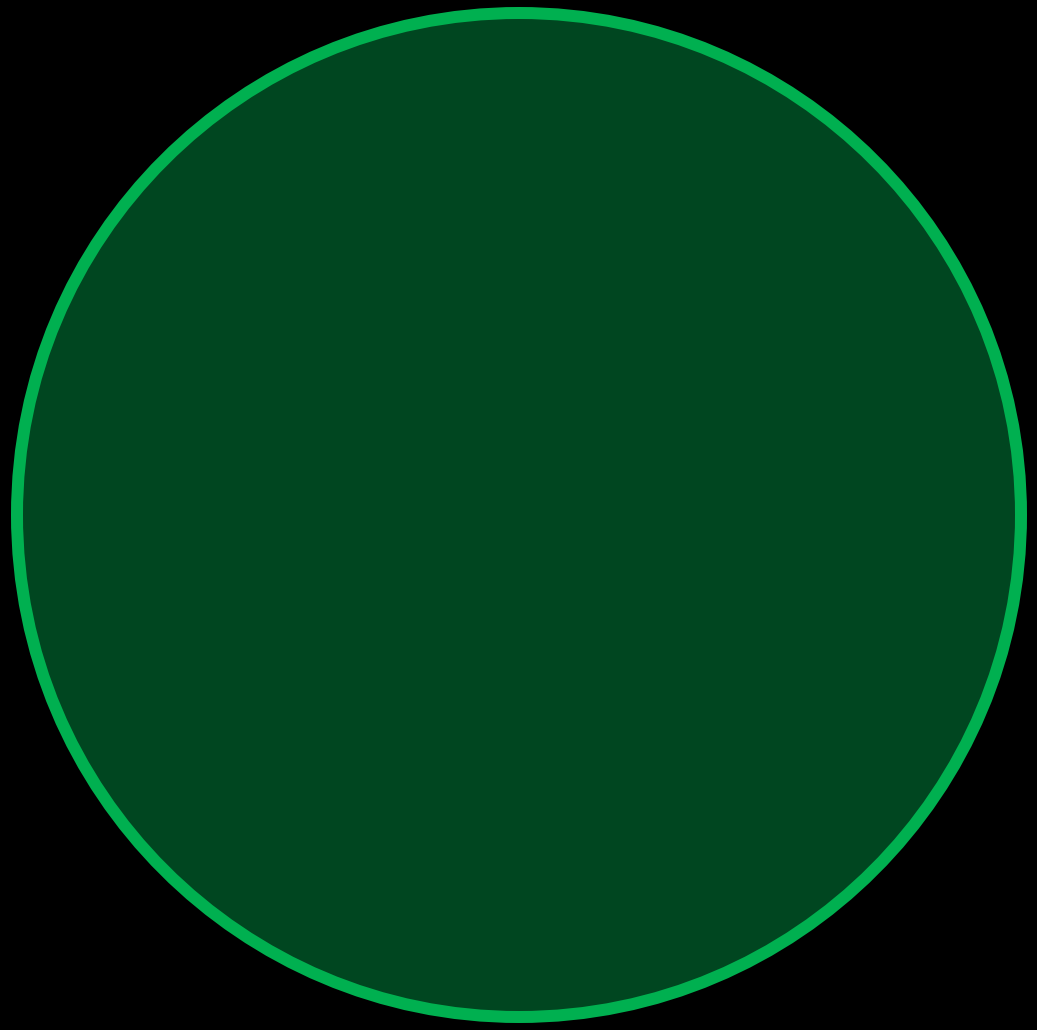
pure function: same input = same output / no side effects

HOF: consumes **OR** produces a function

pure function

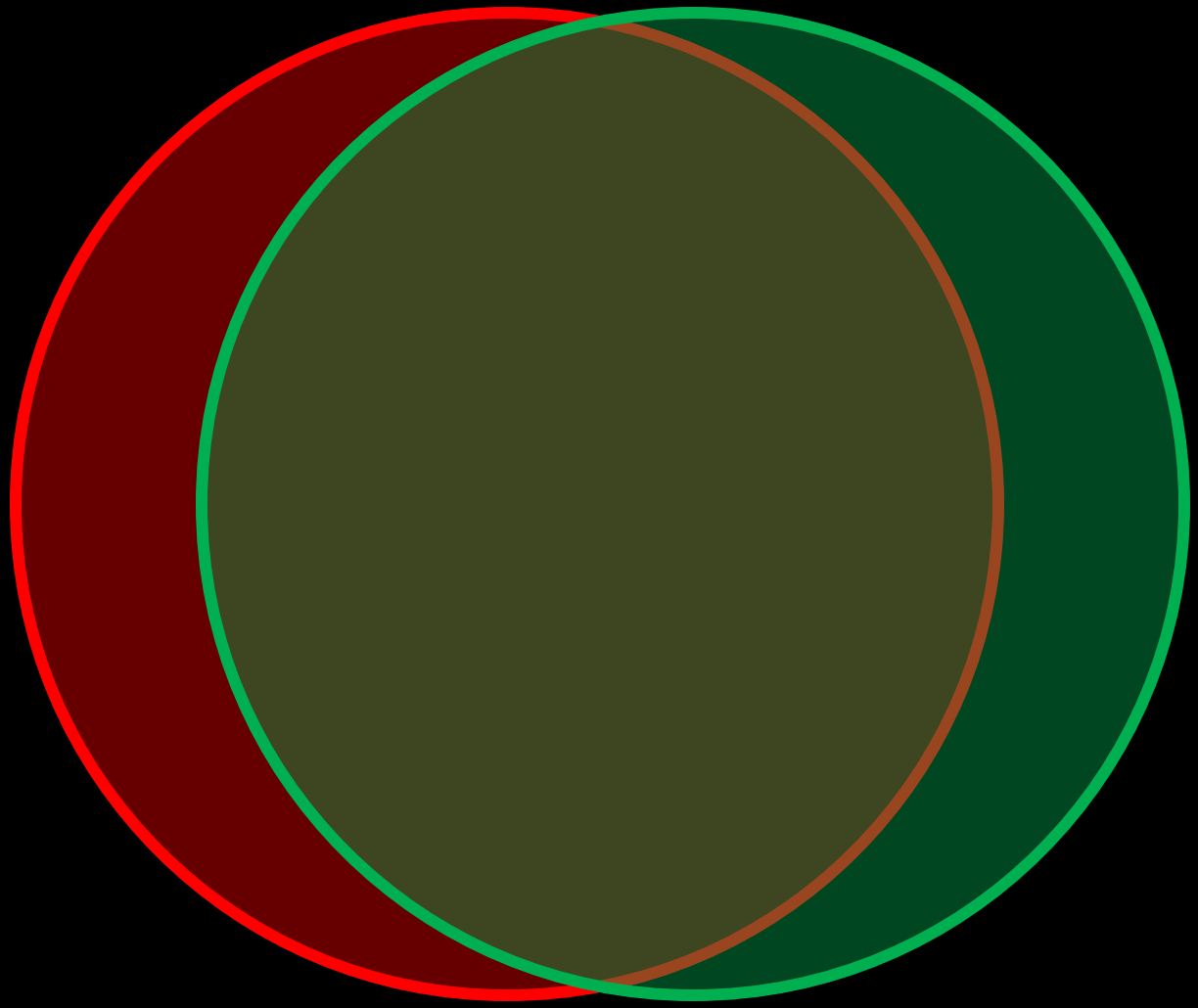


combinator



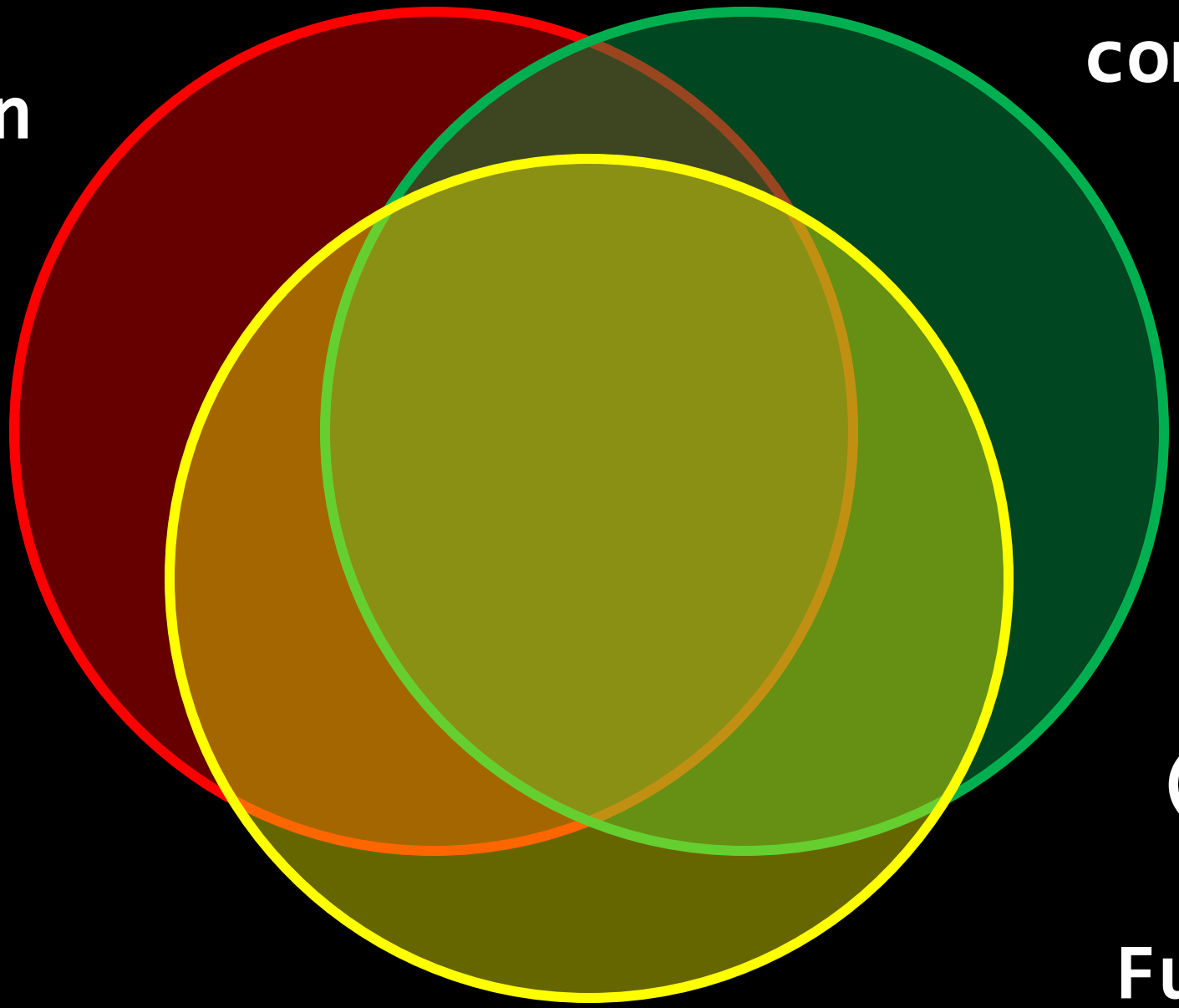
pure function

combinator

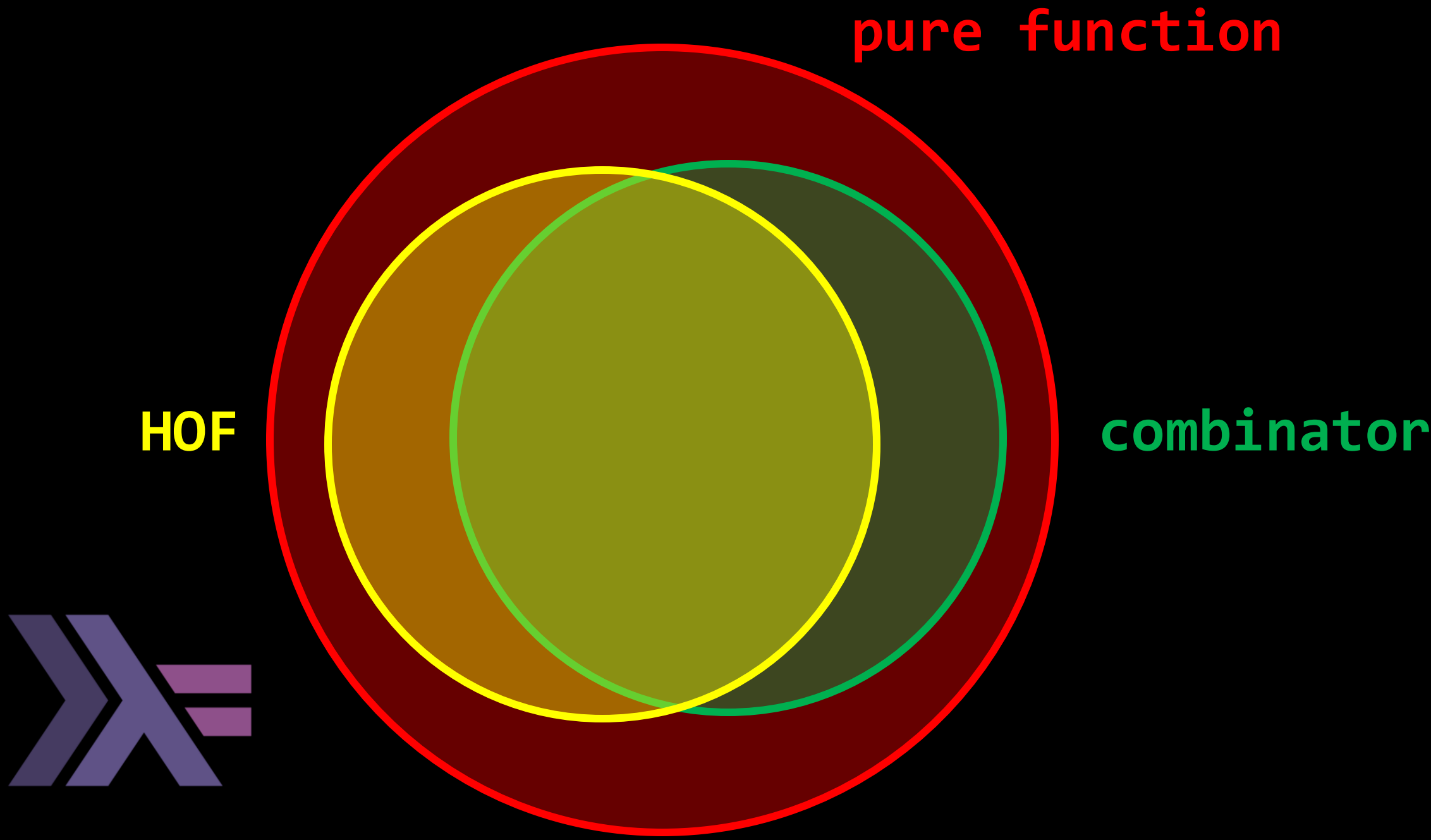


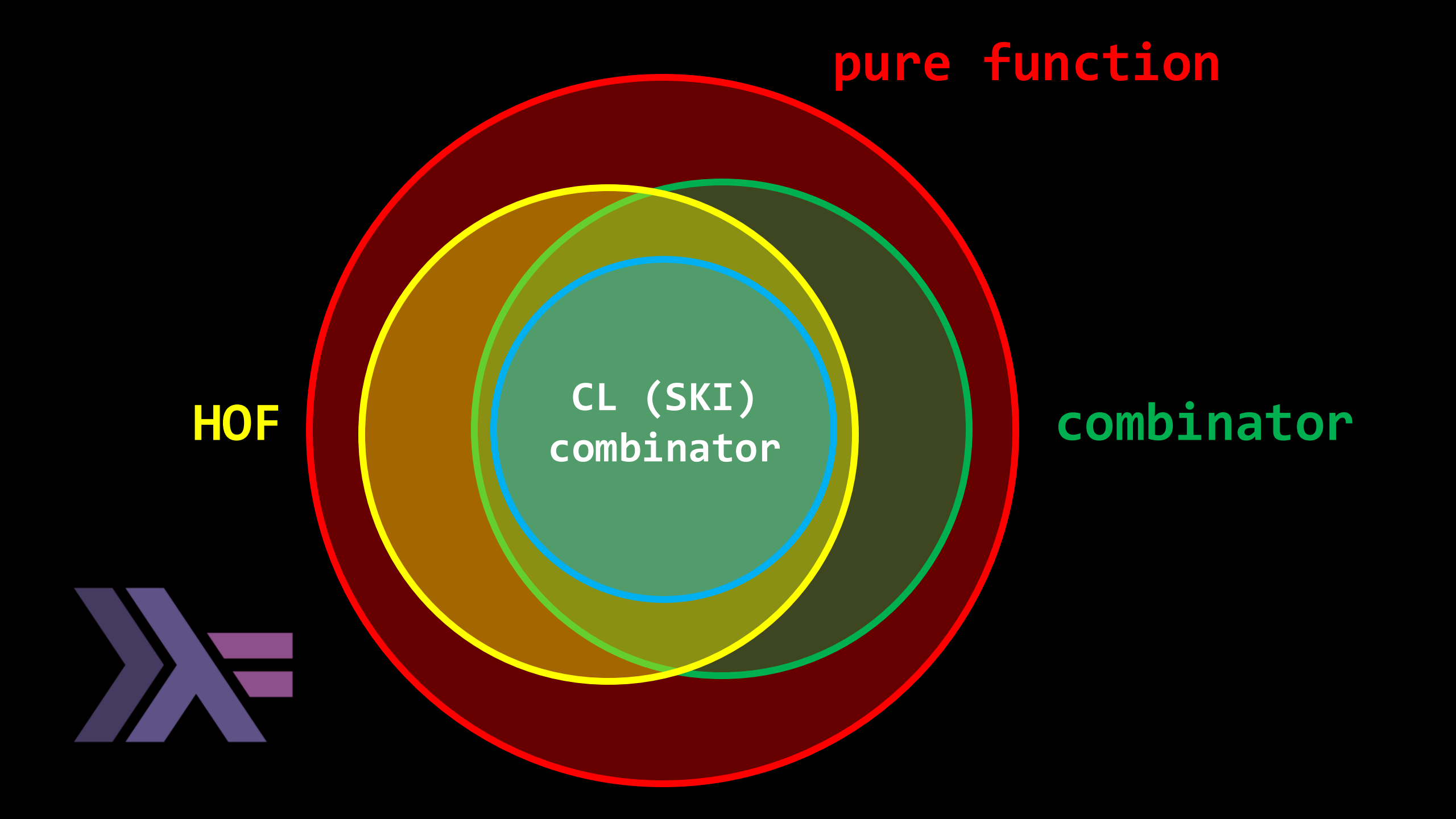
pure function

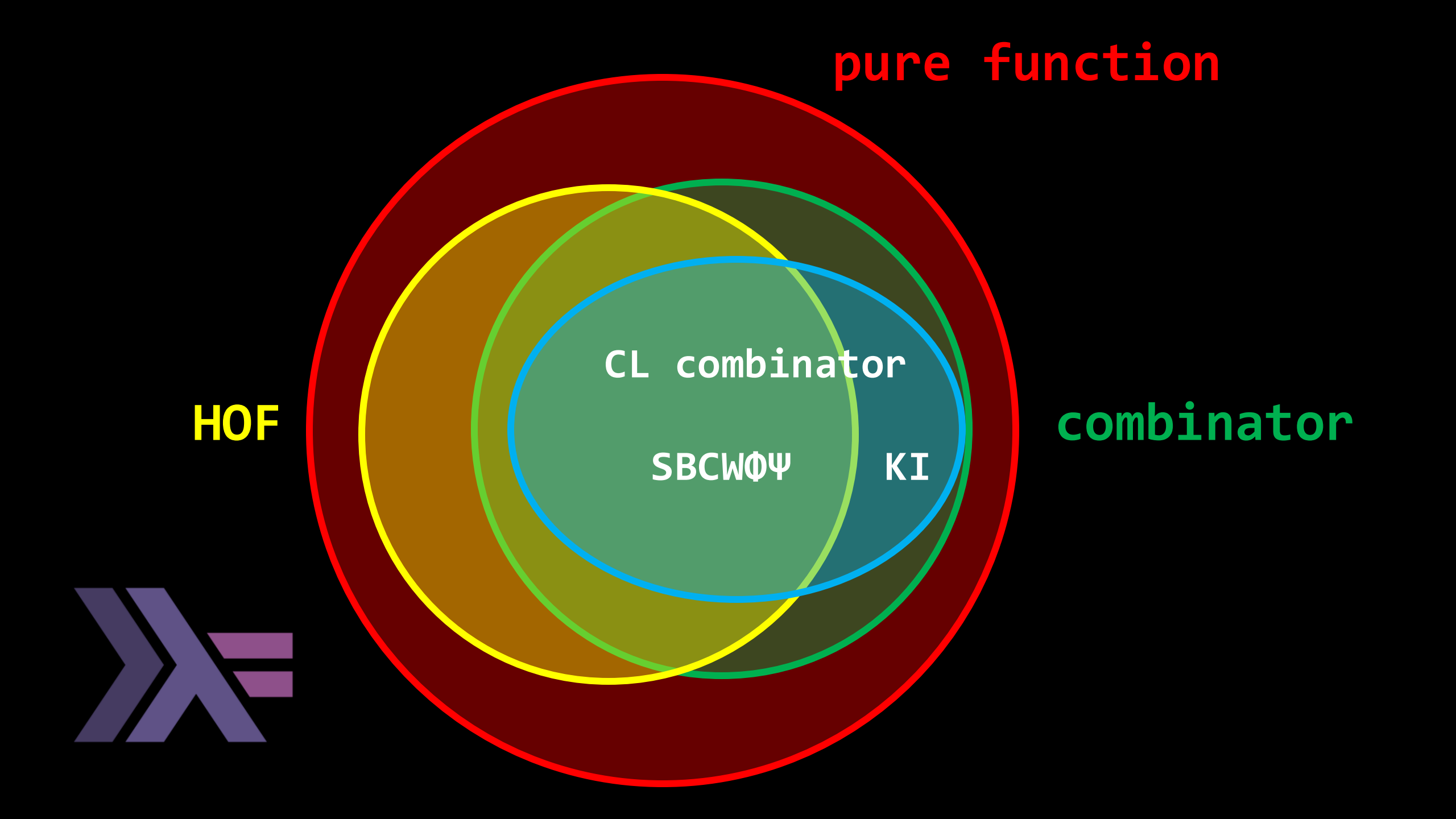
combinator

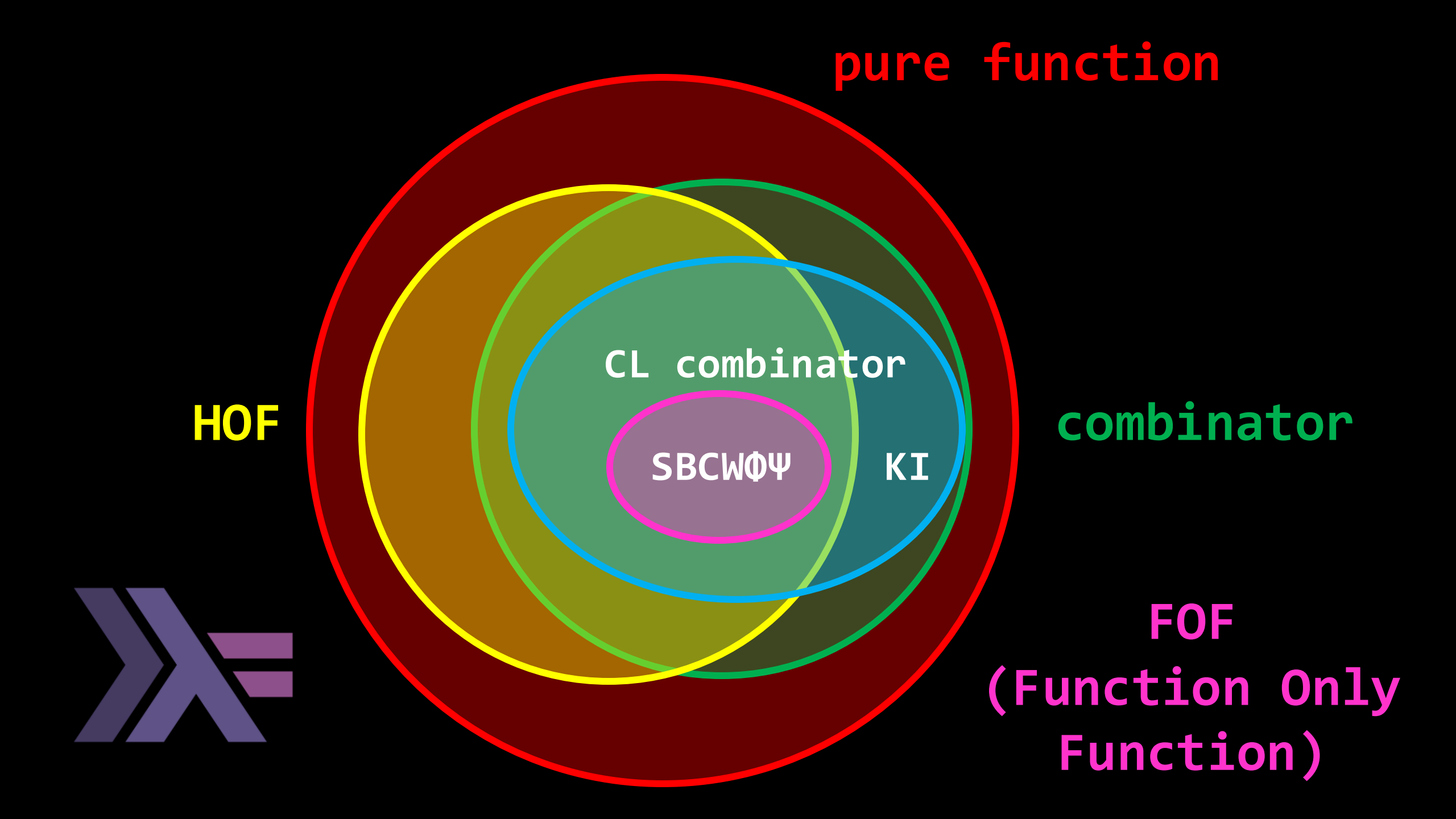


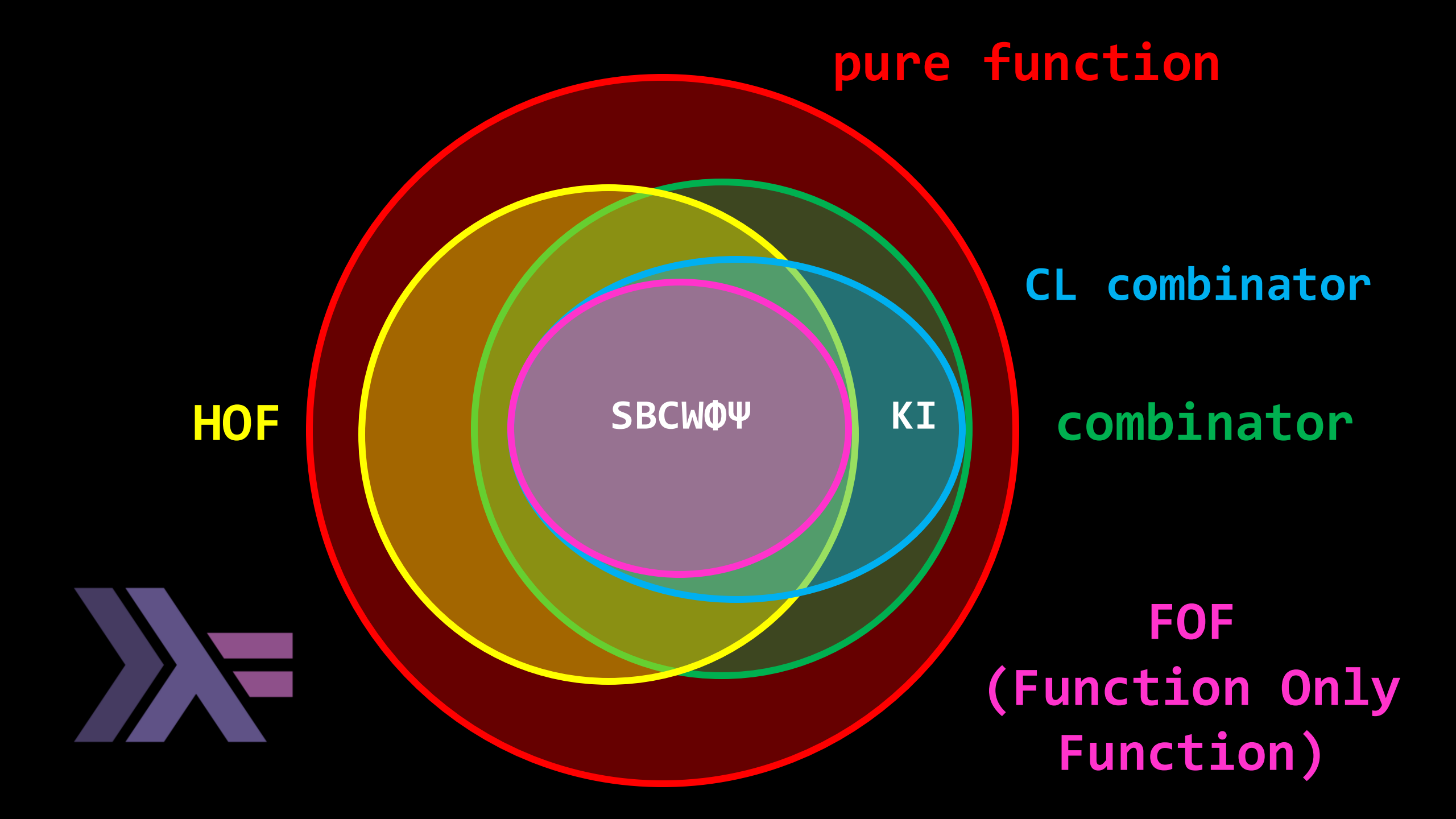
HOF (Higher Order Function)


pure function
HOF
combinator


pure function
HOF
combinator
CL (SKI)
combinator


pure function
HOF
combinator
CL combinator
SBCWΦΨ
KI


pure function
HOF
combinator
CL combinator
SBCWΦΨ
KI
FOF
(Function Only
Function)


pure function
CL combinator
combinator
HOF
SBCWΦΨ
KI
FOF
(Function Only
Function)

combinator: a function that deals only in its arguments

CL combinator: a combinator from Combinatory Logic

FOF: a combinator that only consumes **AND** produces functions

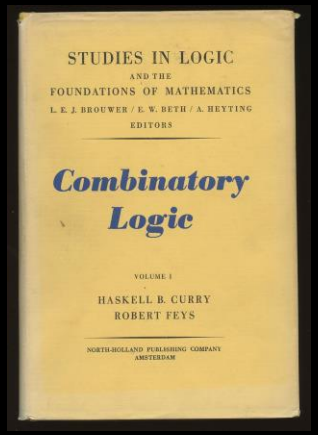
pure function: same input = same output / no side effects

HOF: consumes **OR** produces a function

Combinators

THE ELEMENTARY COMBINATORS

Combinator	Elementary Name
I	Elementary Identificator
C	Elementary Permutator
W	Elementary Duplicator
B	Elementary Compositor
K	Elementary Cancellator



```
def i(x):
    return x
```

```
def k(x, y):
    return x
```

```
def w(f):
    return lambda x: f(x, x)
```

[[digression]]

Conor Hoekstra @code_report · Jan 8, 2022

Also, I apologize for my above average number of tweets 🐦 today, but this table of Greek/Latin words for describing function **arity** will be necessary for a future talk.

The Ê combinator is "tetradic"

Unary/Monadic
Binary/Dyadic
Ternary/Triadic
Quaternary/Tetradic

Terminology [edit]

Latinate names are commonly used for specific arities, primarily bas
cardinal numbers or ordinal numbers. For example, 1-ary is based o

x-ary	Arity (Latin based)	Adicity (Greek based)
0-ary	*Nullary* (from *nūllus*)	*Niladic*
1-ary	*Unary*	*Monadic*
2-ary	*Binary*	*Dyadic*
3-ary	*Ternary*	*Triadic*
4-ary	*Quaternary*	*Tetradic*

[[end of digression]]

```
def w(f):
    return lambda x: f(x, x)
```

```
def b(f, g):
    return lambda x: f(g(x))
```

```
def c(f):
    return lambda x, y: f(y, x)
```

```
def s(f, g):
    return lambda x: f(x, g(x))
```

```
def i   (x):       return x
def k   (x, y):    return x
def ki  (x, y):    return y
def s   (f, g):    return lambda x:    f(x, g(x))
def b   (f, g):    return lambda x:    f(g(x))
def c   (f):       return lambda x, y: f(y, x)
def w   (f):       return lambda x:    f(x, x)
def d   (f, g):    return lambda x, y: f(x, g(y))
def b1  (f, g):    return lambda x, y: f(g(x, y))
def psi (f, g):    return lambda x, y: f(g(x), g(y))
def phi (f, g, h): return lambda x:    g(f(x), h(x))
```

Example

Example
Special Array

https://leetcode.com/problems/special-array-i/description/

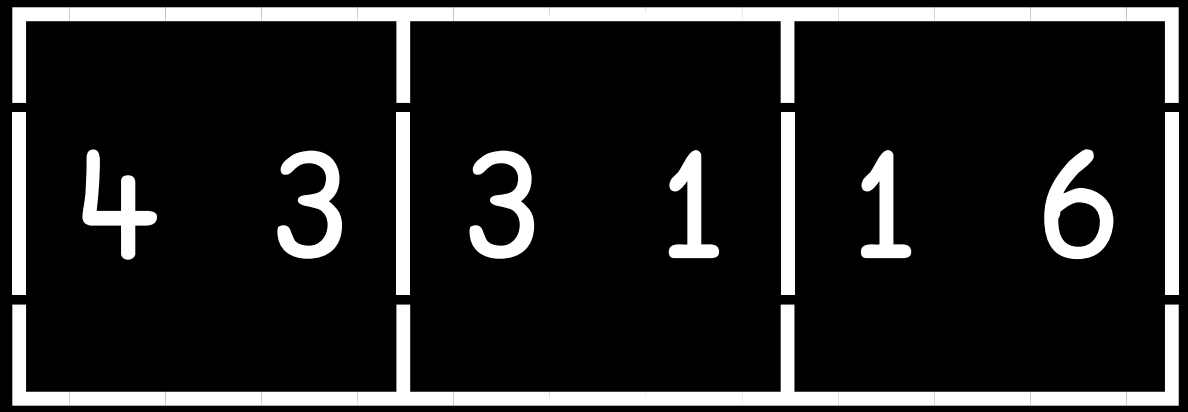
3151. Special Array I

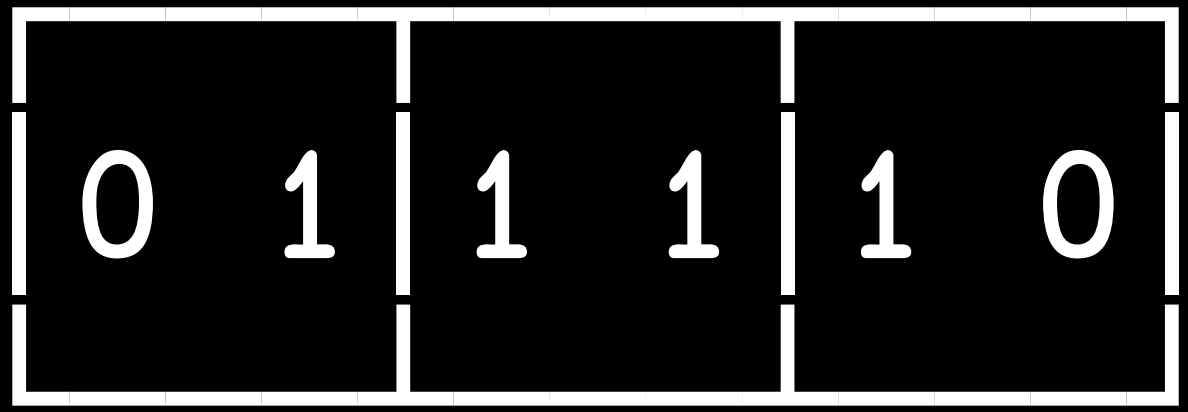
Easy | Topics | Companies | Hint

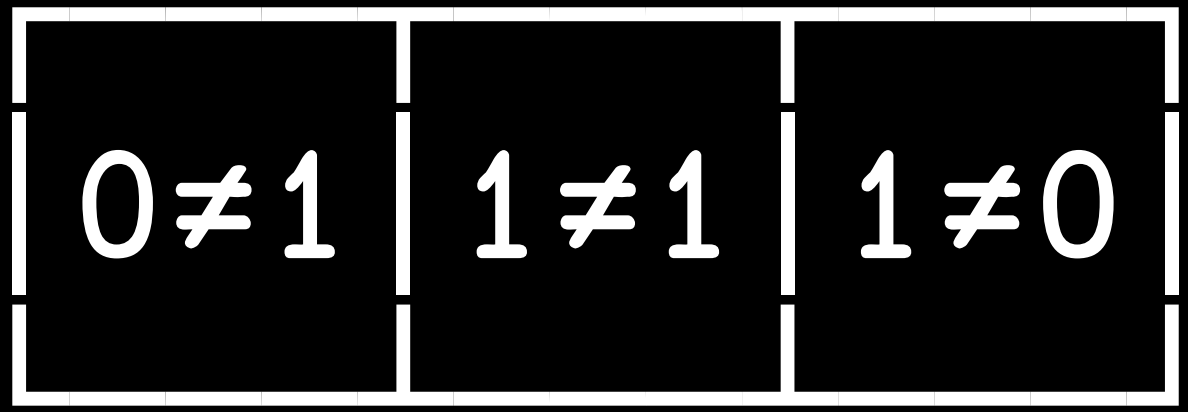
An array is considered **special** if every pair of its adjacent elements contains two numbers with different parity.

You are given an array of integers `nums`. Return `true` if `nums` is a **special** array, otherwise, return `false`.

4 3 1 6


4 3
3 1
1 6


0 1
1 1
1 0

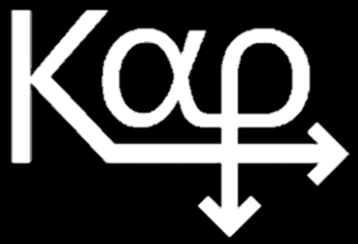

0≠1
1≠1
1≠0

1 0 1

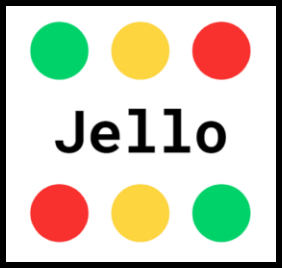
0





APL

Kαρ




Jello

Python

Function Composition

- ✅ Functions
- ❌ Operators
- ❌ Trains
- ❌ Chains
- ❌ Stacks

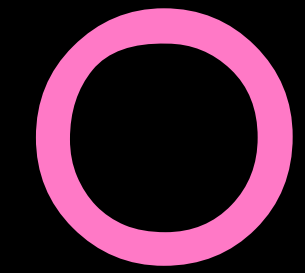
```python
def isArraySpecial(nums):
    for i in range(len(nums) - 1):
        if nums[i] % 2 == nums[i + 1] % 2:
            return False
    return True
```

```python
def isArraySpecial(nums):
    for x, y in zip(nums, nums[1:]):
        if x % 2 == y % 2:
            return False
    return True
```

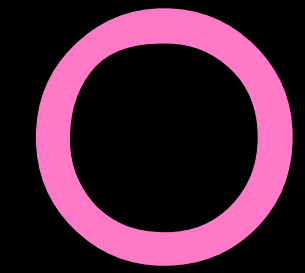
```python
def isArraySpecial(nums):
    return all(x % 2 != y % 2
               for x, y in zip(nums, nums[1:]))
```

```python
from dovekie import psi, odd
from operator import ne

def isArraySpecial(nums):
    return all(psi(ne, odd)(x, y)
               for x, y in zip(nums, nums[1:]))
```

```
def psi(f, g):
    return lambda x, y: f(g(x), g(y))
```



```
_Psi_ ← {(𝔾w)𝔽𝔾x}
      # Over
   # on (Haskell)
```



```
_Psi_ ← {(𝔾w)𝔽(𝔾x)}
      # Over
      # on (Haskell)
```

```python
from dovekie import psi, odd
from operator import ne

def isArraySpecial(nums):
    return all(psi(ne, odd)(x, y)
               for x, y in zip(nums, nums[1:]))
```

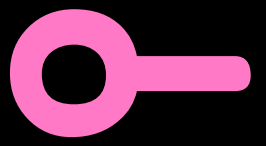
```python
from dovekie import psi, odd
from operator import ne

def isArraySpecial(nums):
    return all(map(psi(ne, odd), nums, nums[1:]))
```

Haskell

Function Composition

- ✅ Functions
- ✅ Operators
- ❌ Trains
- ❌ Chains
- ❌ Stacks

```
isArraySpecial = foldl1 (&&)
               . ((zipWith (/=)) <*> tail)
               . map odd
```



```
_S_ ← {𝕩𝔽(𝔾𝕩)}
# After (Monadic)
# Hook (J/I)
```

FOF	APL	Kap	J	BQN	Jelly	Uiua	Haskell
W	⍨	⍨	~	˜	`	.	join
C	⍨	⍨	~	˜	@	:	flip
B	∘⍥⍥	⍥	@:&:	∘○	*	*	.
B_1	⍥	*	@:	∘	*	*	.:
S		∘	*	⊸		*	ap / <*>
Σ		⍛∘		⊸	*	*	=<<
D	∘	∘	*	⊸		*	
Δ		⍛∘		⊸	*	*	
Ψ	⍥	⍥	&:	○	*	∩	on
D_2		a⍛∘b∘c		a⊸b⊸c		⊓	
Φ	*	a«b»c	*	*	*	⊃	liftA2
$Φ_1$	*	a«b»c	*	*	*	⊃	

Operators Functions Trains Chains Stacks*

```
isArraySpecial = foldl1 (&&)
               . ((zipWith (/=)) <*> tail)
               . map odd
```

```haskell
import Data.List.HT (mapAdjacent)

isArraySpecial = foldl1 (&&)
               . mapAdjacent (/=)
               . map odd
```

```haskell
import Data.List.HT (mapAdjacent)

isArraySpecial = and
               . mapAdjacent (/=)
               . map odd
```

```haskell
import Data.List.HT (mapAdjacent)
import Data.Function (on)

isArraySpecial = and
               . mapAdjacent (on (/=) odd)
```

```python
from dovekie import psi, odd
from operator import ne

def isArraySpecial(nums):
    return all(map(psi(ne, odd), nums, nums[1:]))
```

```python
from dovekie import psi, odd
from operator import ne



def isArraySpecial(nums):
    return all(map(psi(ne, odd), nums, nums[1:]))
```

```python
from dovekie import psi, odd
from operator import ne
from itertools import pairwise

def adjacentMap(xs, op):
    return [op(a, b) for a, b in pairwise(xs)]

def isArraySpecial(nums):
    return all(map(psi(ne, odd), nums, nums[1:]))
```

```python
from dovekie import psi, odd
from operator import ne
from itertools import pairwise

def adjacentMap(xs, op):
    return [op(a, b) for a, b in pairwise(xs)]

def isArraySpecial(nums):
    return all(adjacentMap(nums, psi(ne, odd)))
```

```python
from dovekie import odd
from operator import ne
from itertools import pairwise

def adjacentMap(xs, op):
    return [op(a, b) for a, b in pairwise(xs)]

def isArraySpecial(nums):
    return all(adjacentMap(map(odd, nums), ne))
```

Clojure

Function Composition

- ✅ Functions
- ✅ Operators
- ❌ Trains
- ❌ Chains
- ❌ Stacks

```clojure
(defn is-special-array [nums]
  (->> nums
       (partition 2 1)
       (map #(reduce not= %))
       (every? identity)))
```

```clojure
(defn is-special-array [nums]
  (->> nums
       (partition 2 1)
       (every? (fn [[a b]] not= a b))))
```

APL, BQN, J, Kap

Function Composition

- ❌ Functions
- ✅ Operators
- ✅ Trains
- ❌ Chains
- ❌ Stacks

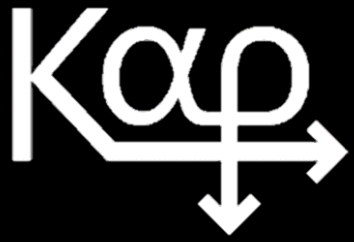


Table 4: 2 and 3-trains in APL, Kap, BQN and J.

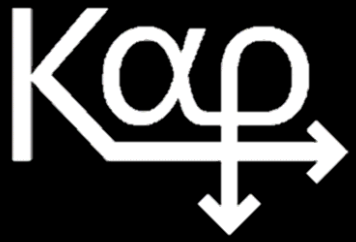
Year	Language	2-Train	3-Train
1990	J	S and D	Φ and Φ_1
2014	Dyalog APL	B and B_1	Φ and Φ_1
2020	Kap	B and B_1	-
2020	BQN	B and B_1	Φ and Φ_1

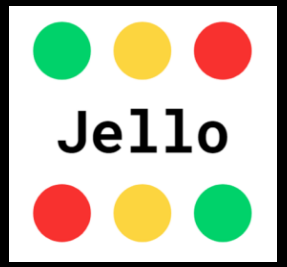
```
def i   (x):       return x
def k   (x, y):    return x
def ki  (x, y):    return y
def s   (f, g):    return lambda x:    f(x, g(x))
def b   (f, g):    return lambda x:    f(g(x))
def c   (f):       return lambda x, y: f(y, x)
def w   (f):       return lambda x:    f(x, x)
def d   (f, g):    return lambda x, y: f(x, g(y))
def b1  (f, g):    return lambda x, y: f(g(x, y))
def psi (f, g):    return lambda x, y: f(g(x), g(y))
def phi (f, g, h): return lambda x:    g(f(x), h(x))
```

```
def s   (f, g):    return lambda x:    f(x, g(x))
def b   (f, g):    return lambda x:    f(g(x))

def d   (f, g):    return lambda x, y: f(x, g(y))
def b1  (f, g):    return lambda x, y: f(g(x, y))

def phi(f, g, h): return lambda x:    g(f(x), h(x))
```

```
def b  (f, g):  return lambda x:    f(g(x))
def b1 (f, g):  return lambda x, y: f(g(x, y))

def phi (f, g, h): return lambda x:    g(f(x), h(x))
def phi1(f, g, h): return lambda x, y: g(f(x, y), h(x, y))
```

Live Coding





Jelly

Function Composition

- ❌ Functions
- ✅ Operators
- ❌ Trains
- ✅ Chains
- ❌ Stacks

Uiua

Function Composition

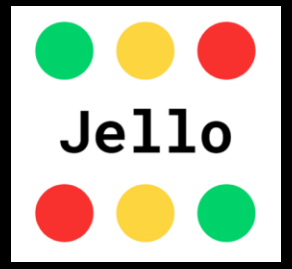
❌ Functions

✅ Operators

❌ Trains

❌ Chains

✅ Stacks



Live Coding

Kap

```
∧/2≠/2|
 B  B
```

Dyalog APL

```
∧/2≠/2|⊢
 B  Φ  Φ
```

Uiua

```
/↧≡/≠⊞2⊿2
```

BQN

```
∧´∘≠´˘2↕2|⊢
 B  B  Φ  Φ
```

J

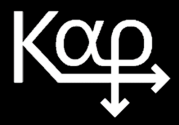
```
[: */ 2~:/\ 2|[
   B     Φ  Φ
```

Jello

```
odd? differ all
      B     B
```

									
Functions	✅	✅	✅	❌	❌	❌	❌	❌	❌
Operators	❌	✅	✅	✅	✅	✅	✅	✅	✅
Trains	❌	❌	❌	✅	✅	✅	✅	❌	❌
Chains	❌	❌	❌	❌	❌	❌	❌	✅	❌
Stacks	❌	❌	❌	❌	❌	❌	❌	❌	✅

In Conclusion



Kap

```
∧/2≠/2|
```

Dyalog APL

```
∧/2≠/2|⊢
```

Uiua

```
/↧≡/≠⊞2⊿2
```

BQN

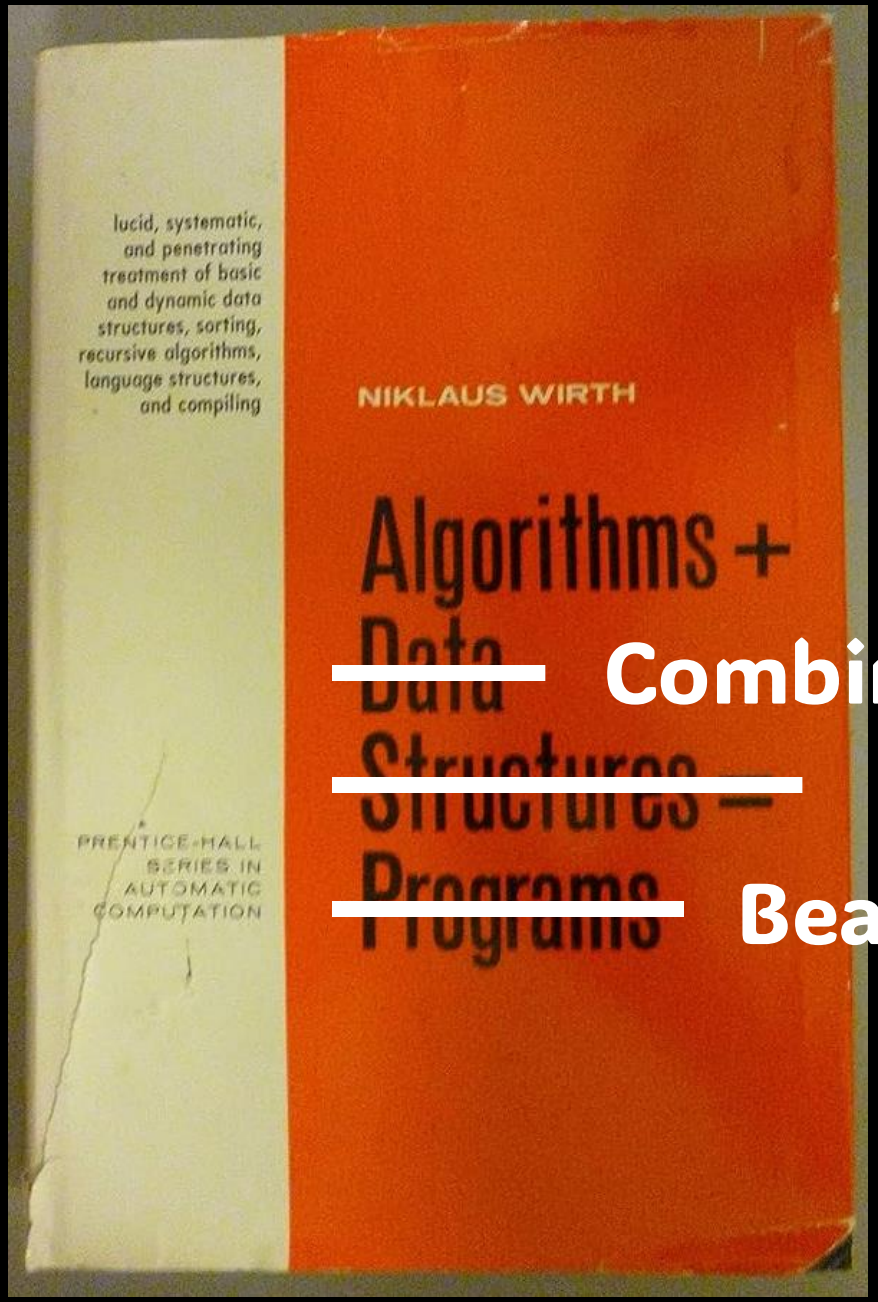
```
∧´⊸≠´˘2↕2|⊢
```

J

```
[: */ 2~:/\ 2|[
```

Jello

```
odd? differ all
```



~~Algorithms + Data Structures = Programs~~ **Combinators = Beautiful Code**

FOF	APL	Kap	J	BQN	Jelly	Uiua	Haskell
W	⍨	⍨	~	˜	`	.	join
C	⍨	⍨	~	˜	@	:	flip
B	∘⍥⍥	⍥	@:&:	∘○	*	*	.
B_1	⍥	*	@:	∘	*	*	.:
S		∘	*	⊸		*	ap / <*>
Σ		⍛∘		⊸	*	*	=<<
D	∘	∘	*	⊸		*	
Δ		⍛∘		⊸	*	*	
Ψ	⍥	⍥	&:	○	*	∩	on
D_2		a⍛∘b∘c		a⊸b⊸c		⊓	
Φ	*	a«b»c	*	*	*	⊃	liftA2
$Φ_1$	*	a«b»c	*	*	*	⊃	

Operators Functions Trains Chains Stacks*

```python
from dovekie import odd
from operator import ne

def isArraySpecial(nums):
    return all(adjacentMap(map(odd, nums), ne))
```

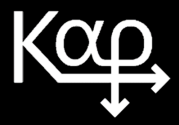
```python
from dovekie import odd

def isArraySpecial(nums):
    return all(differ(map(odd, nums)))
```

```python
from dovekie import odd

def isArraySpecial(nums):
    return all(differ(map(odd, nums)))
```

reduce map


```python
from dovekie import odd

def isArraySpecial(nums):
    return all(differ(map(odd, nums)))
```

reduce map



Kap

```
∧/2≠/2|
```

Dyalog APL

```
∧/2≠/2|⊢
```

Uiua

```
/↧≡/≠⊞2⊿2
```

BQN

```
∧´⊸≠´˘2↕2|⊢
```

J

```
[: */ 2~:/\ 2|[
```

Jello

```
odd? differ all
```

Online REPLS

Language	Link
APL	https://tryapl.org/
Kap	https://kapdemo.dhsdevelopments.com/clientweb2/
BQN	https://bqnpad.mechanize.systems/
J	https://jsoftware.github.io/j-playground/bin/html2/
Uiua	https://www.uiua.dev/pad

Dovekie

Blackbird

Dovekie

dovekie 0.7.0

```
pip install dovekie
```

Blackbird

```
# --- Fetch blackbird ------------------------------------------------------------

FetchContent_Declare(blackbird
  GIT_REPOSITORY https://github.com/codereport/blackbird
  GIT_TAG main
)

FetchContent_GetProperties(blackbird)
if(NOT blackbird_POPULATED)
  FetchContent_Populate(blackbird)
  add_subdirectory(${blackbird_SOURCE_DIR} ${blackbird_BINARY_DIR} EXCLUDE_FROM_ALL)
endif()
```

www.combinatorylogic.com

Thank You

https://github.com/codereport/Content/Talks

Conor Hoekstra

code_report

codereport

Questions?

https://github.com/codereport/Content/Talks

Conor Hoekstra

code_report

codereport