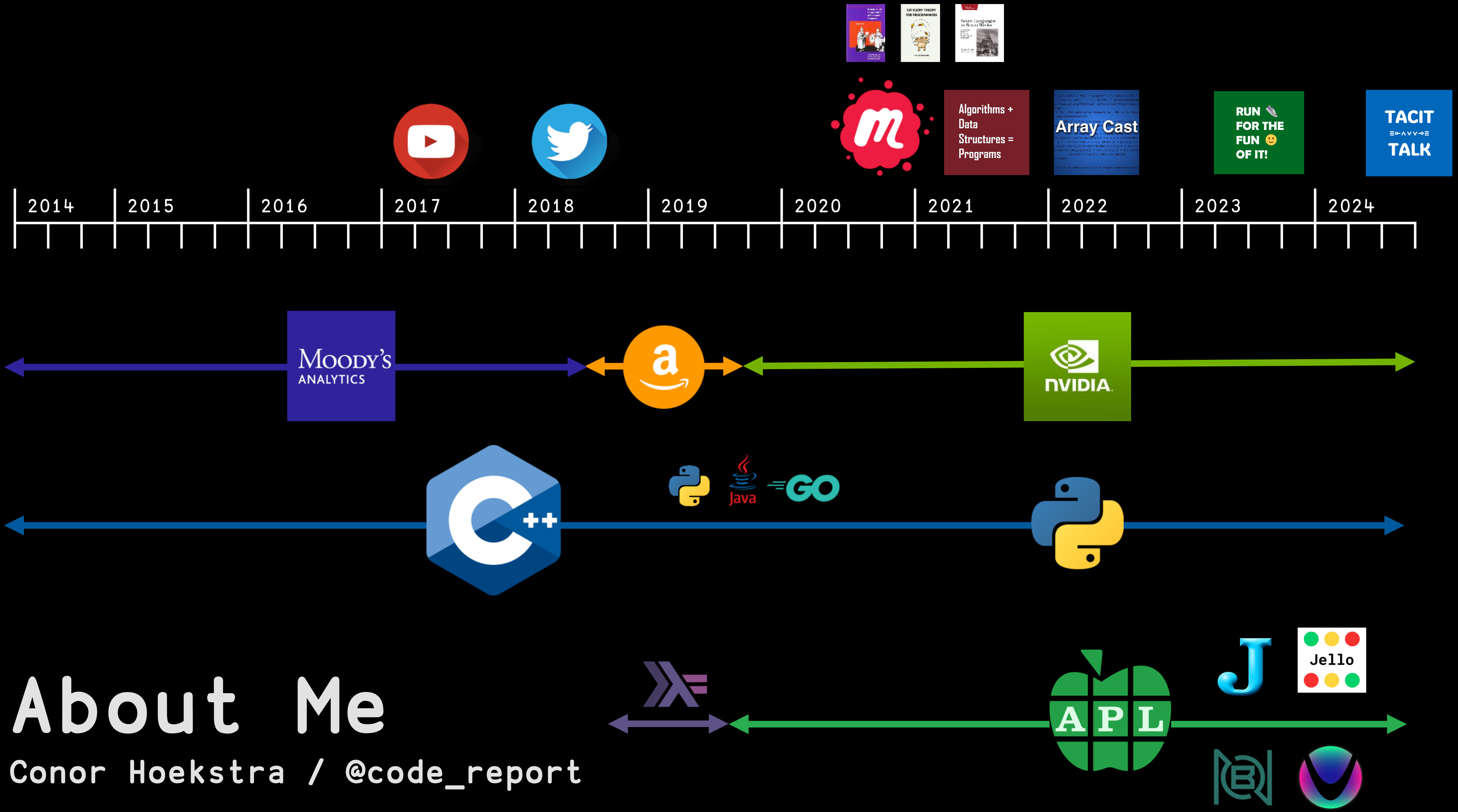
The Power of Function Composition

Conor Hoekstra

code_report | codereport

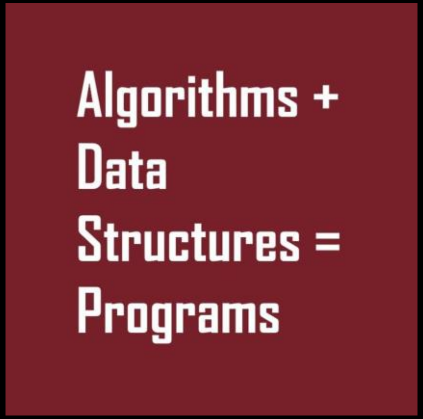


About Me

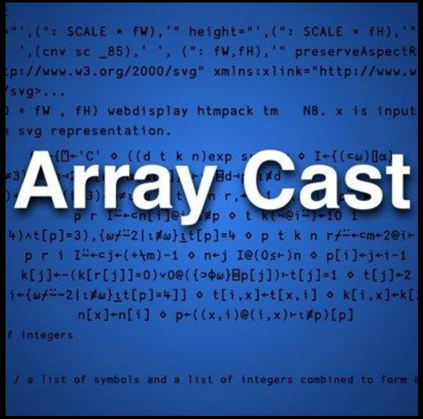
Conor Hoekstra / @code_report

344 Videos

39 (27) Talks



185 Episodes
@adspthepodcast



80 Episodes
@arraycast

2 Episodes
@codereport

18 Episodes
@conorhoekstra

https://github.com/codereport/Content

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*

What is a function only function (FOF)?

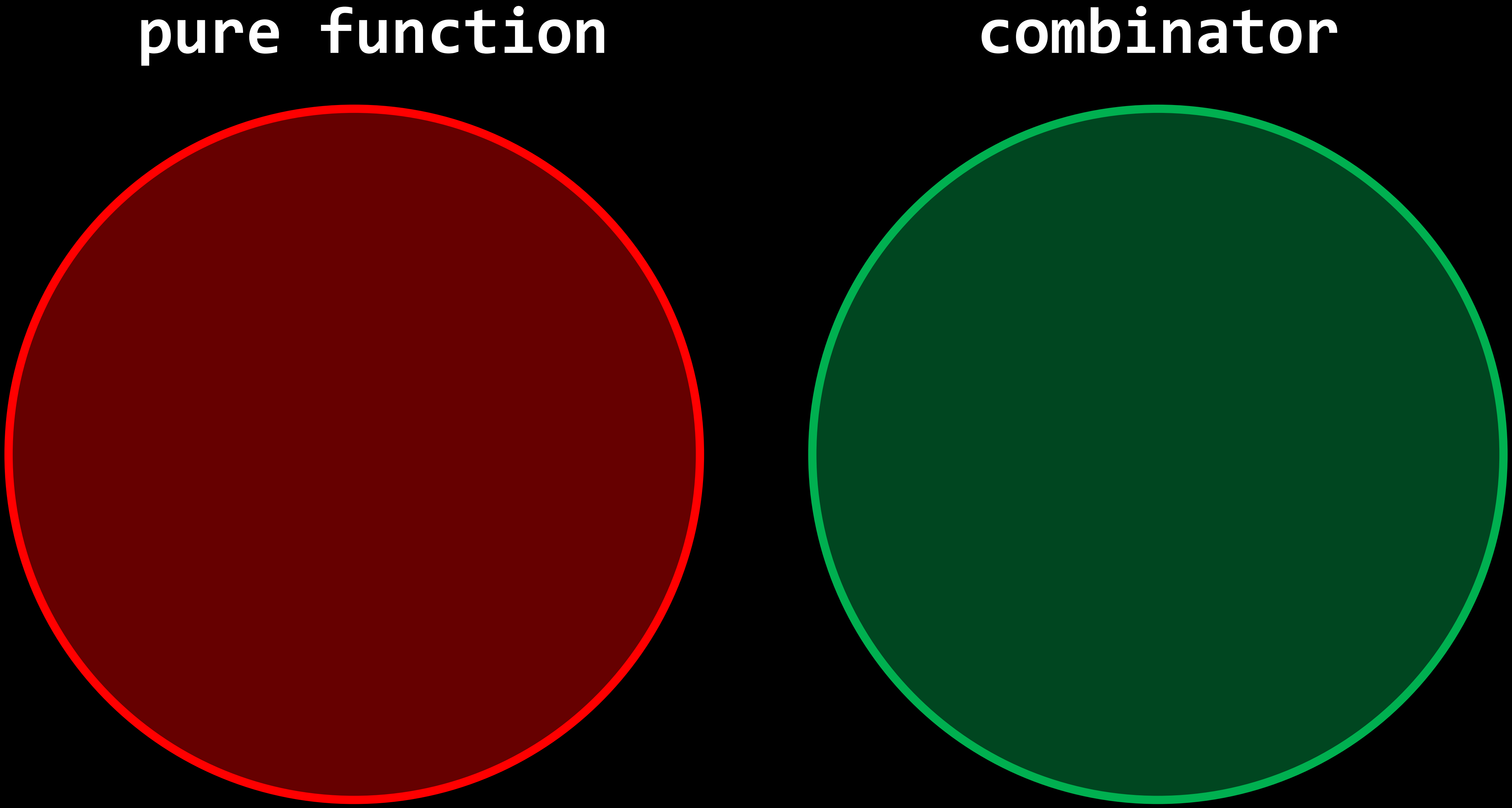
What is a combinator?

combinator: a lambda expression
containing no free variables

combinator: a function that deals
only in its arguments

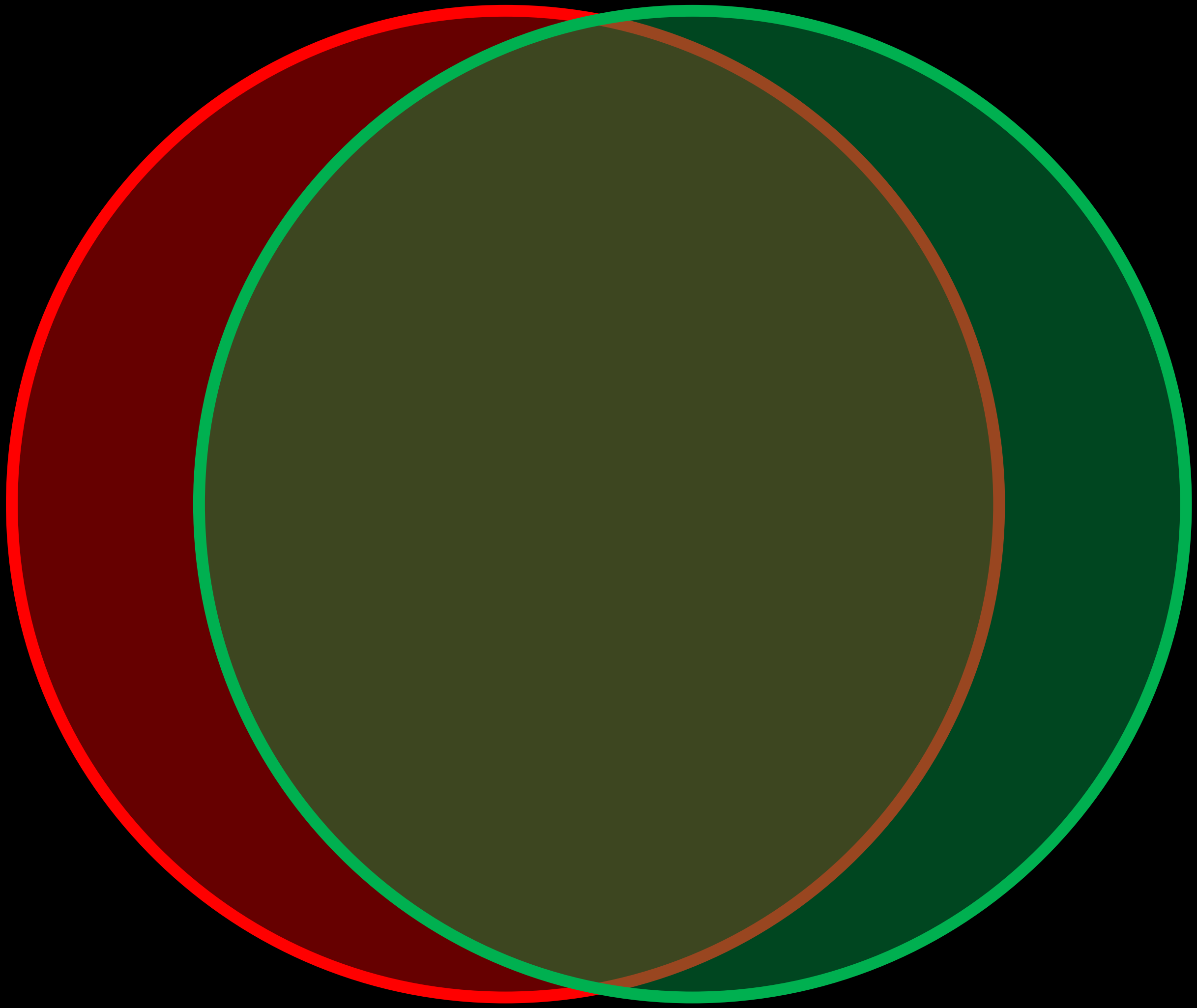
combinator = pure function?

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



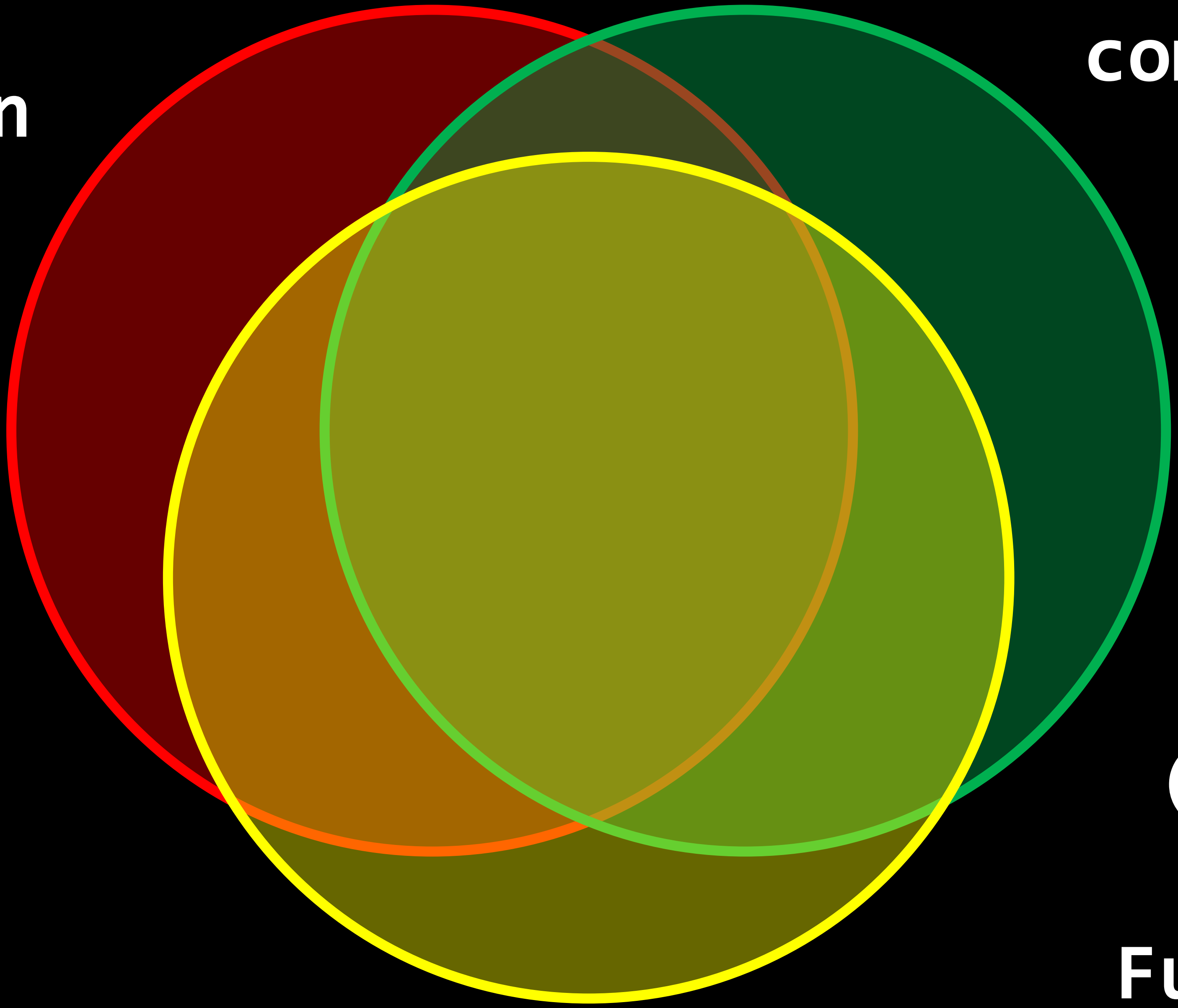
pure function

combinator



pure function

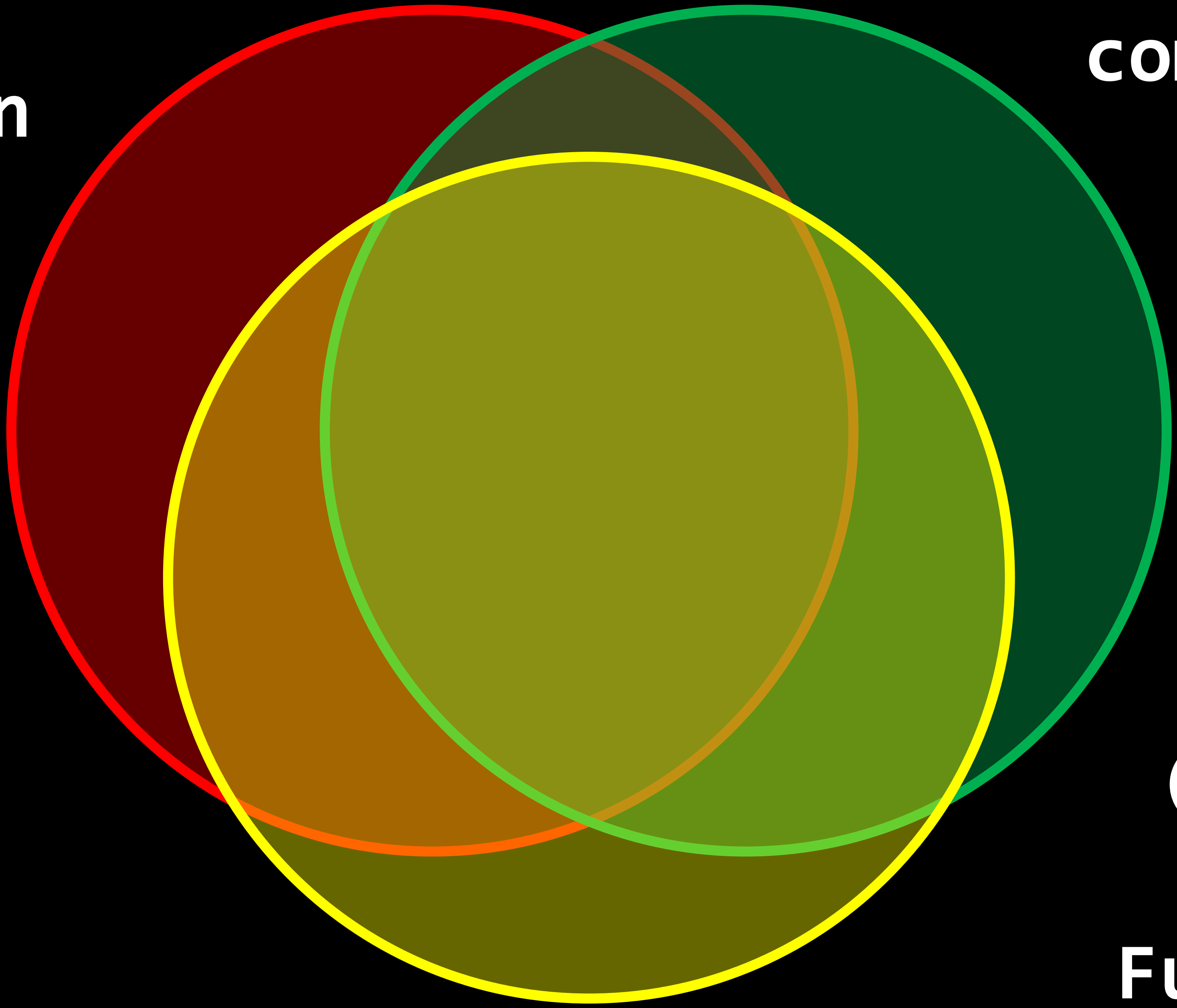
combinator



pure function

combinator

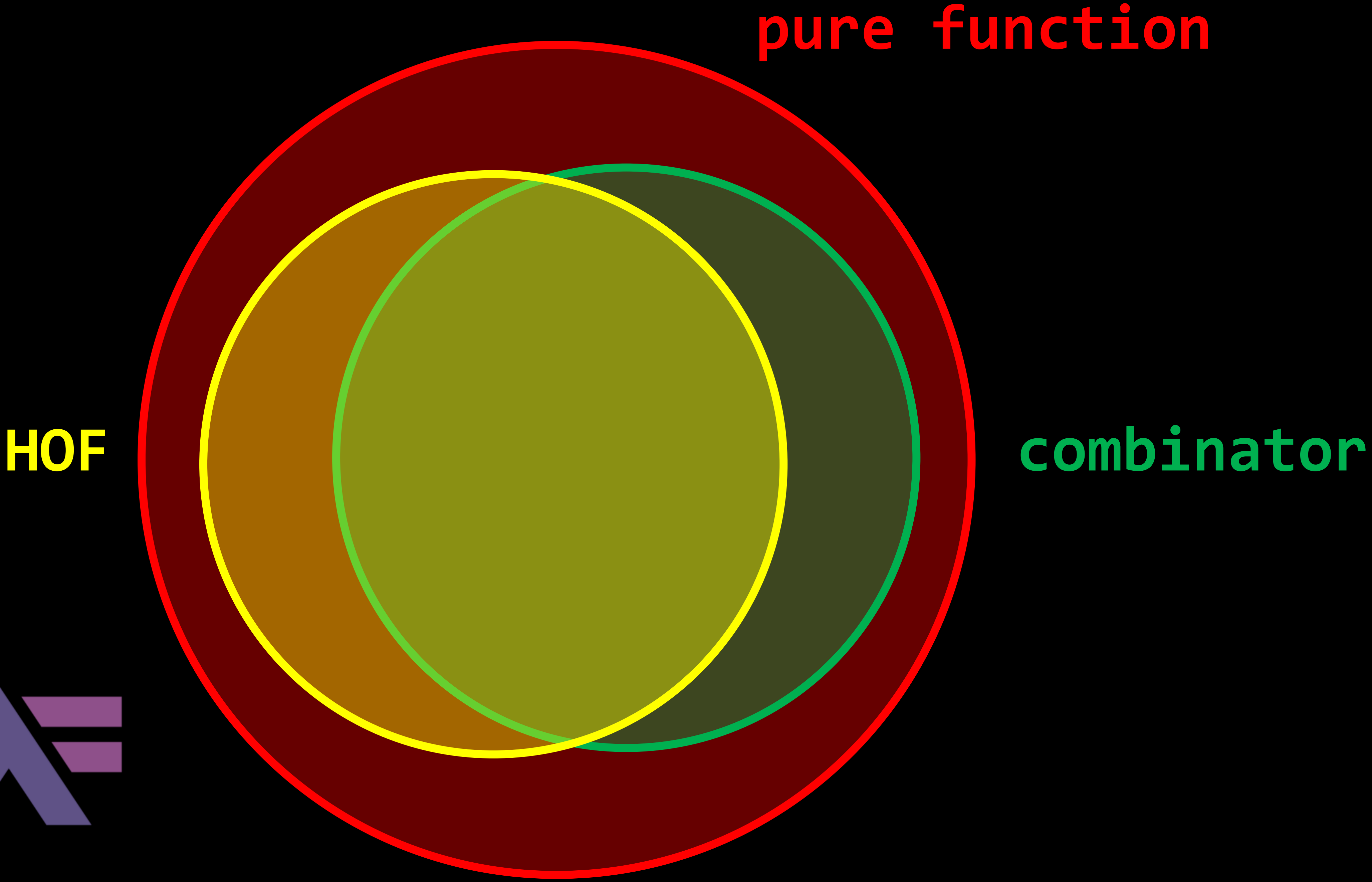
HOF (Higher Order Function)



pure
function

combinator

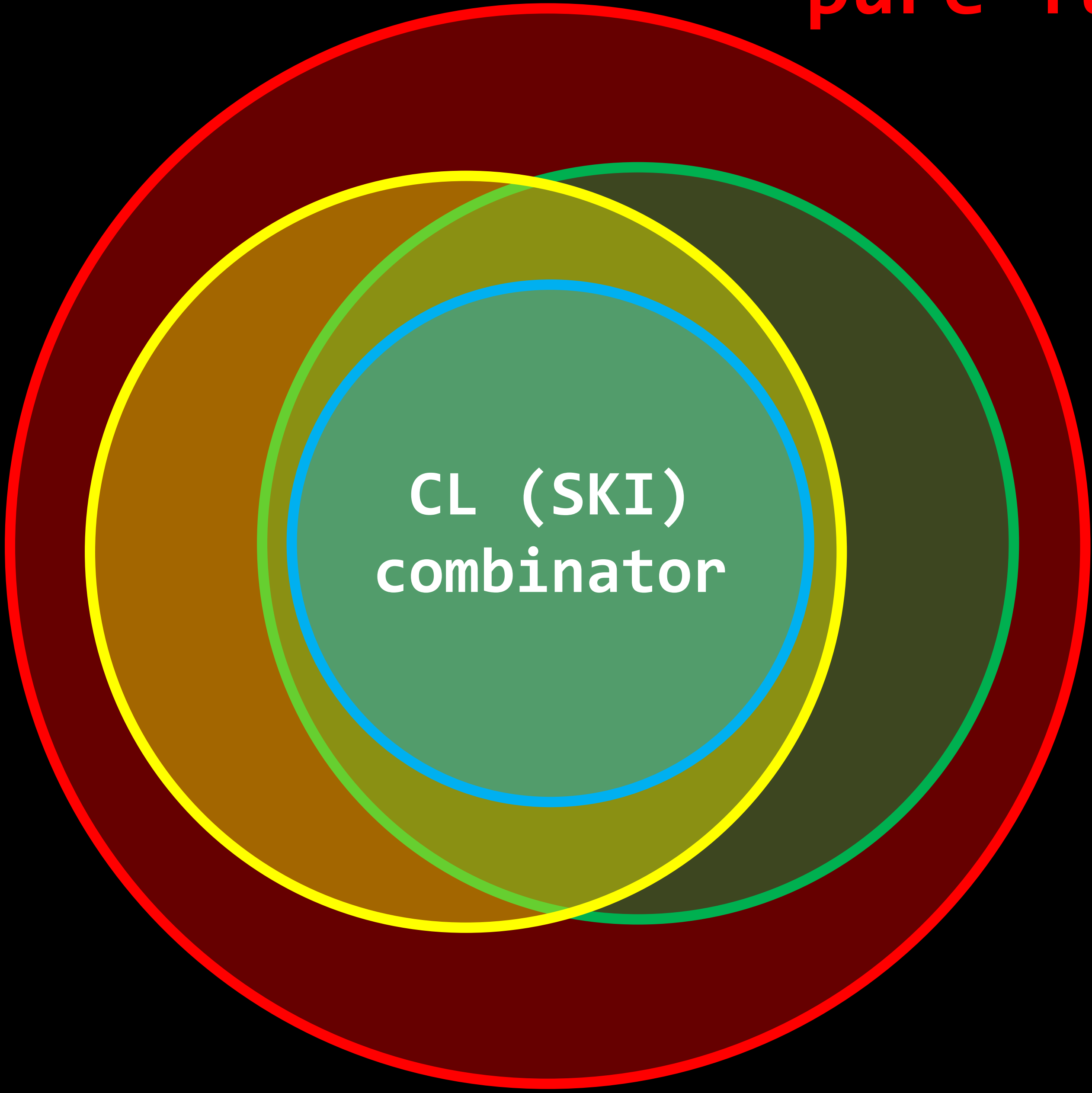
HOF
(Higher
Order
Function)



pure function

HOF

combinator

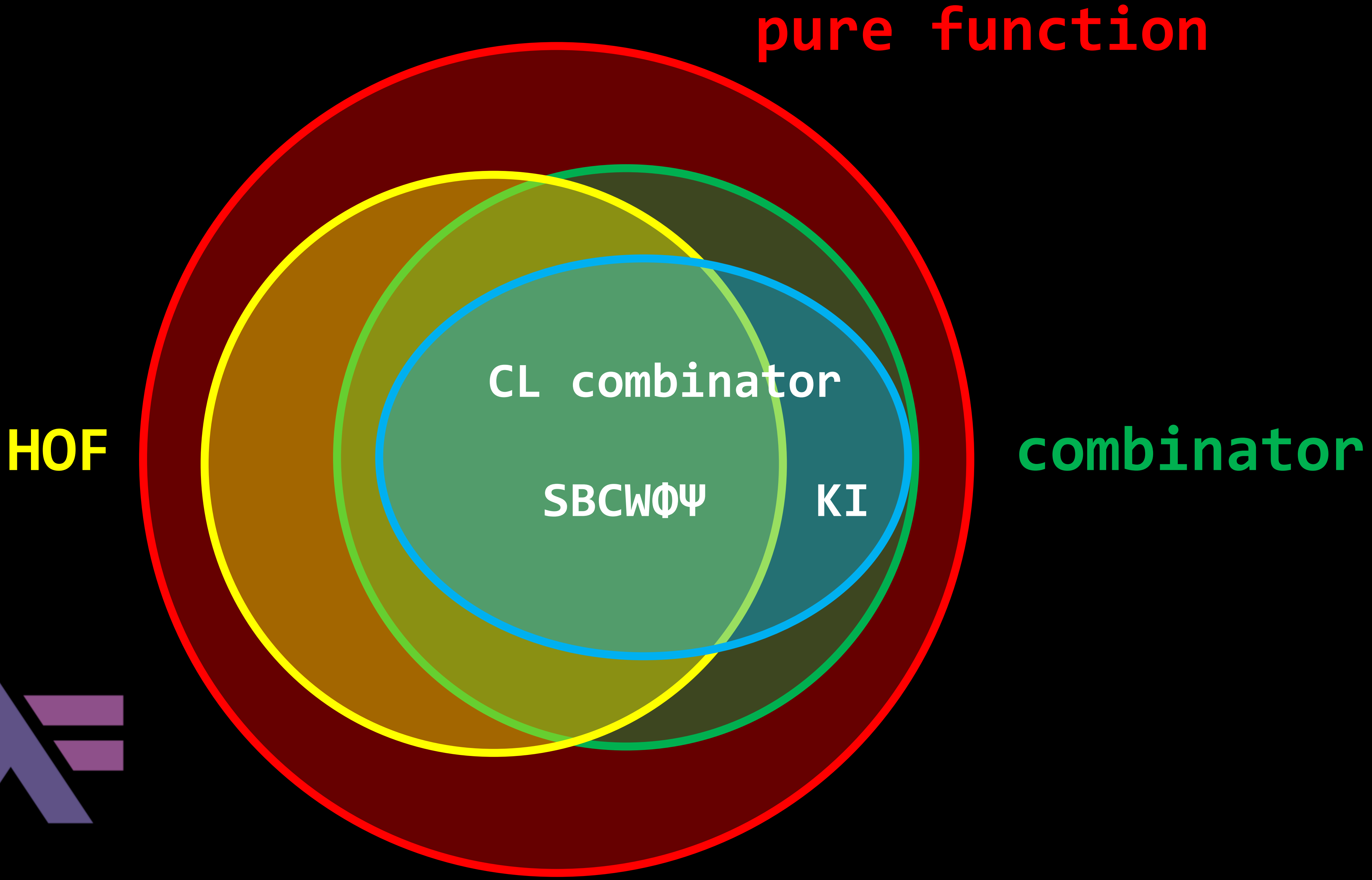


pure function

HOF

CL (SKI)
combinator

combinator



pure function

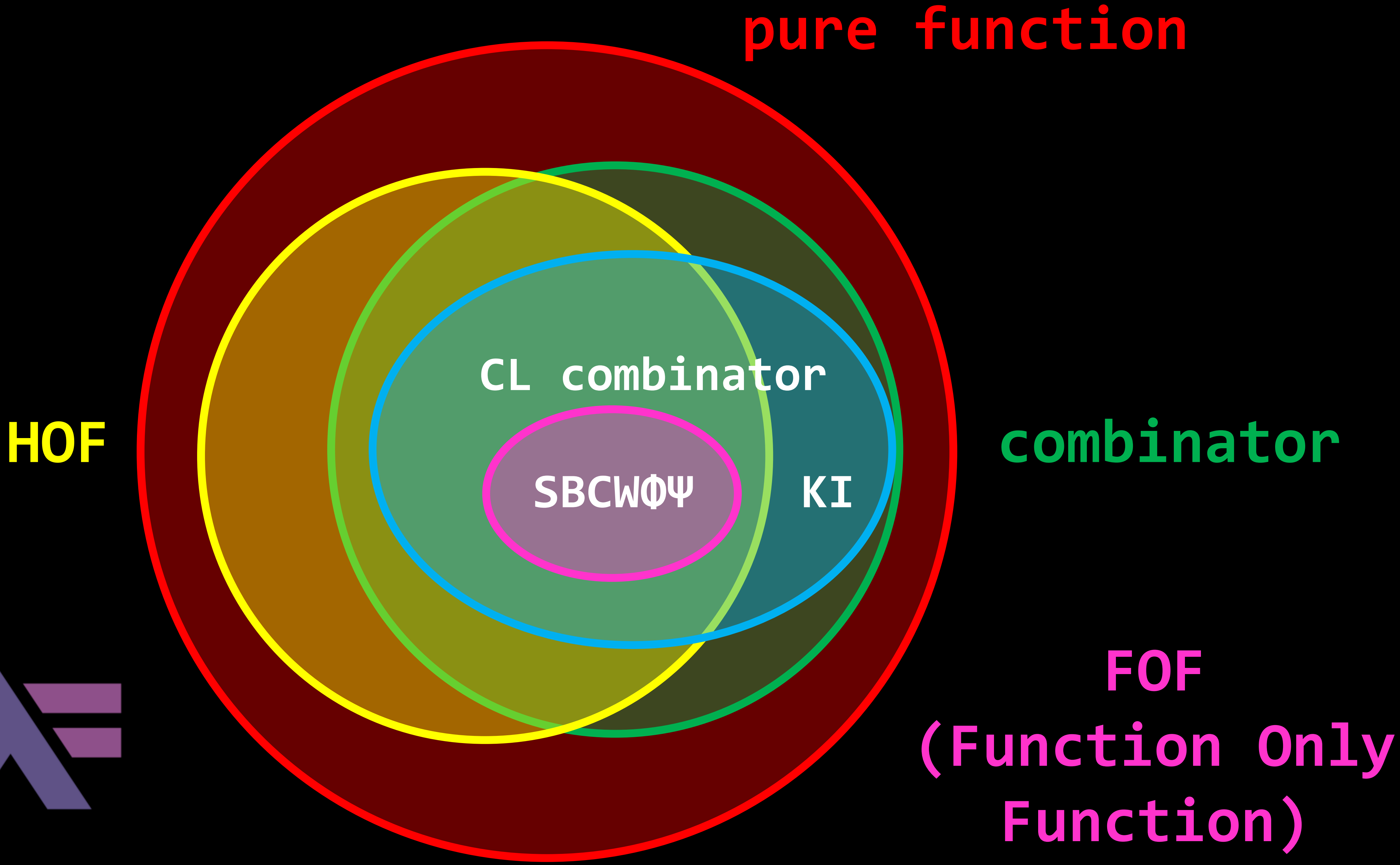
HOF

CL combinator

SBCWΦΨ

KI

combinator



pure function

HOF

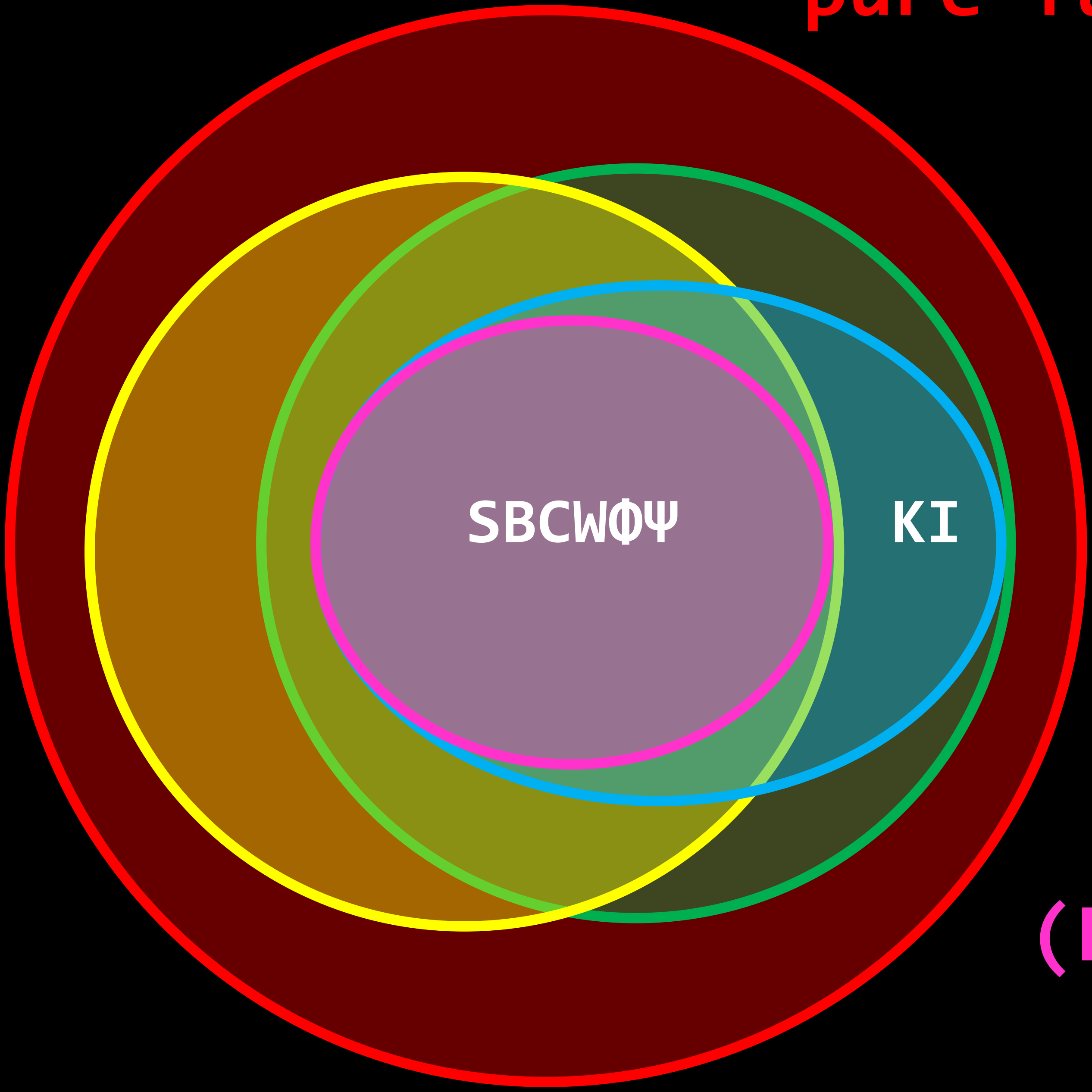
CL combinator

SBCWΦΨ

KI

combinator

FOF
(Function Only
Function)



pure function

HOF

SBCWΦΨ

KI

CL combinator

combinator

FOF
(Function Only
Function)

combinator: a function that deals only in its arguments

CL combinator: a combinator from the Combinatory Logic

FOF: a combinator that only consumes AND produces functions

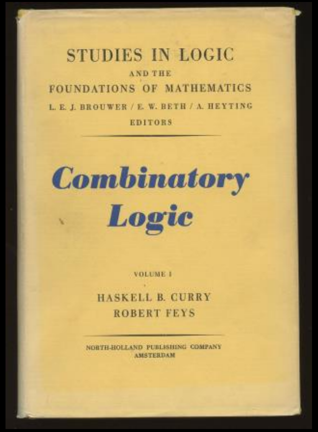
pure function: same input = same output / no side effects

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



STUDIES IN LOGIC
AND THE
FOUNDATIONS OF MATHEMATICS
L. E. J. BROUWER / E. W. BETH / A. HEYTING
EDITORS

Combinatory Logic

VOLUME I
HASKELL B. CURRY
ROBERT FEYS

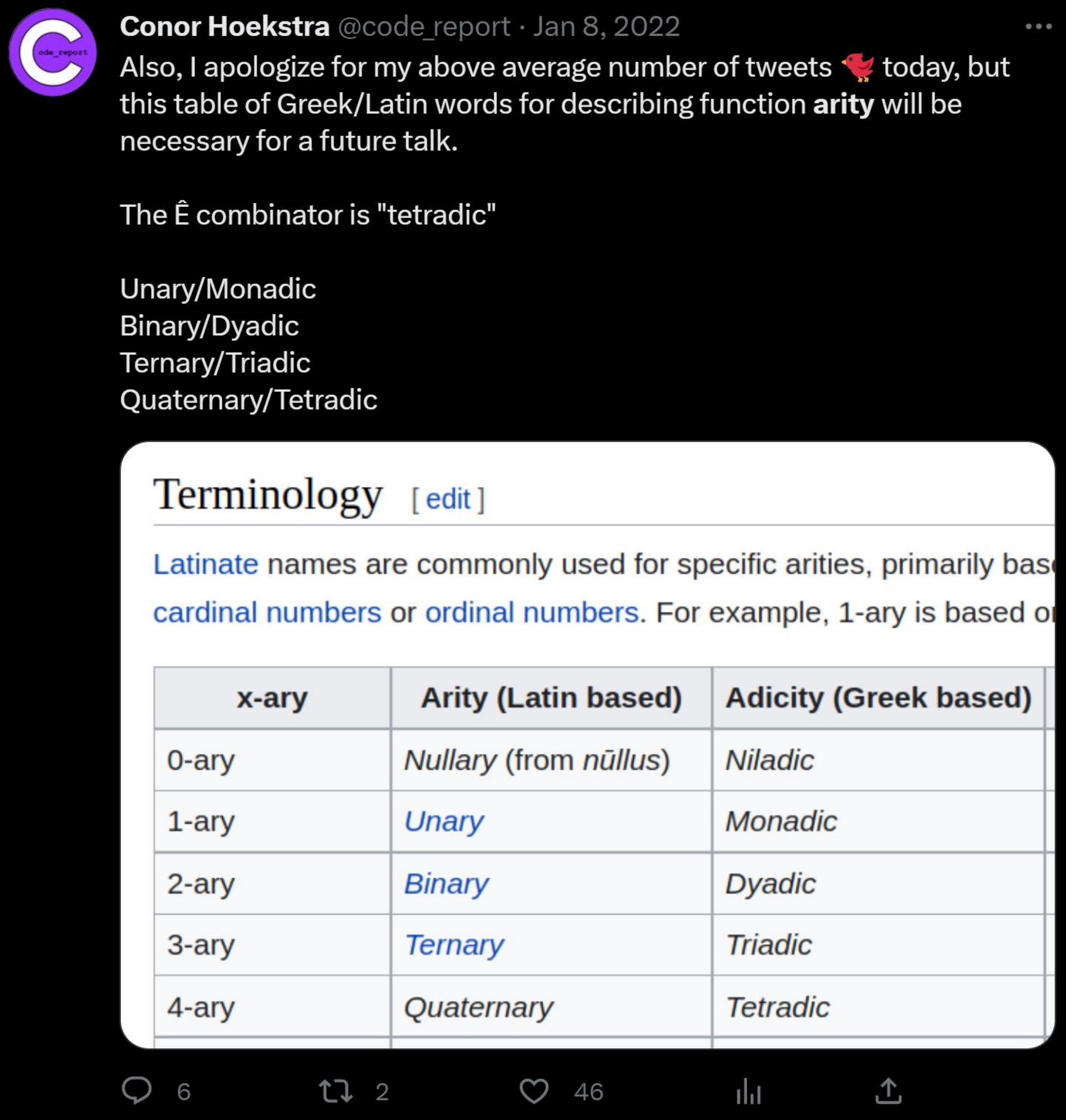
NORTH-HOLLAND PUBLISHING COMPANY
AMSTERDAM

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

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

```python
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]]

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

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

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

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

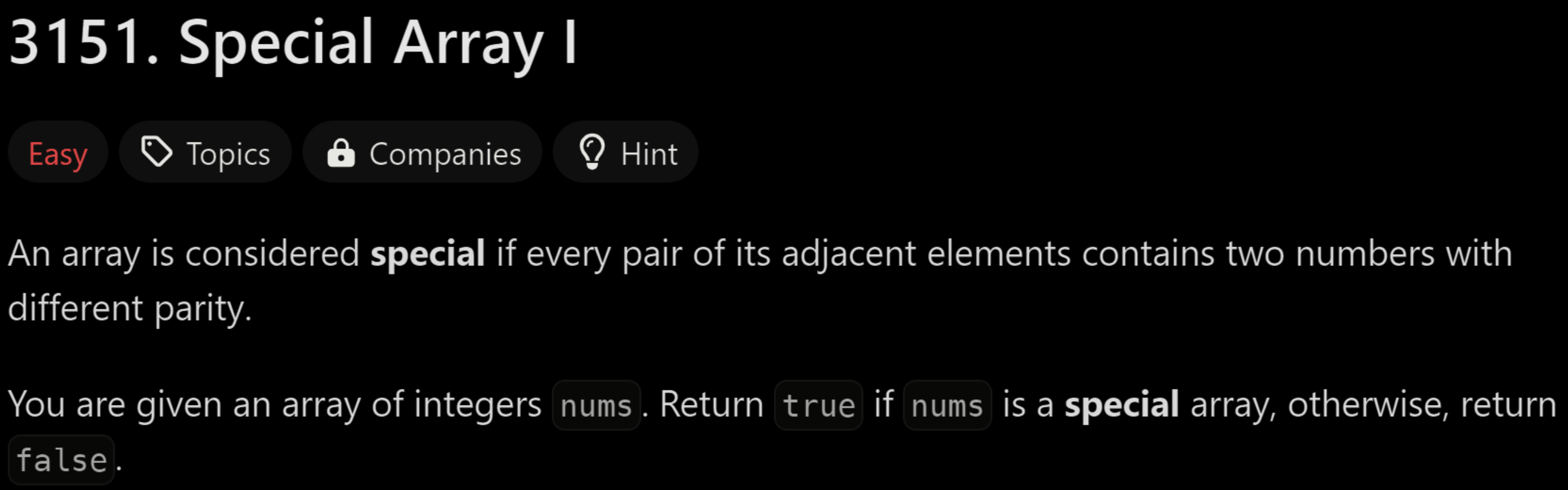
```python
def i  (x):       return x
def k  (x, y):    return x
def ki (x, y):    return y
def s  (f, g):    return lambda x:    f(x, g(x))
def b  (f, g):    return lambda x:    f(g(x))
def c  (f):       return lambda x, y: f(y, x)
def w  (f):       return lambda x:    f(x, x)
def d  (f, g):    return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

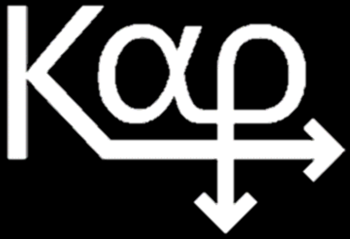
4 3	3 1	1 6

0 1	1 1	1 0

1 0 1

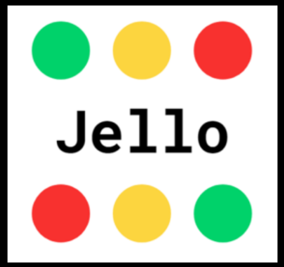
0

APL


Kap

B

J


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
```

```
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:]))
```

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

```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

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

```
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

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

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

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)))
```

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

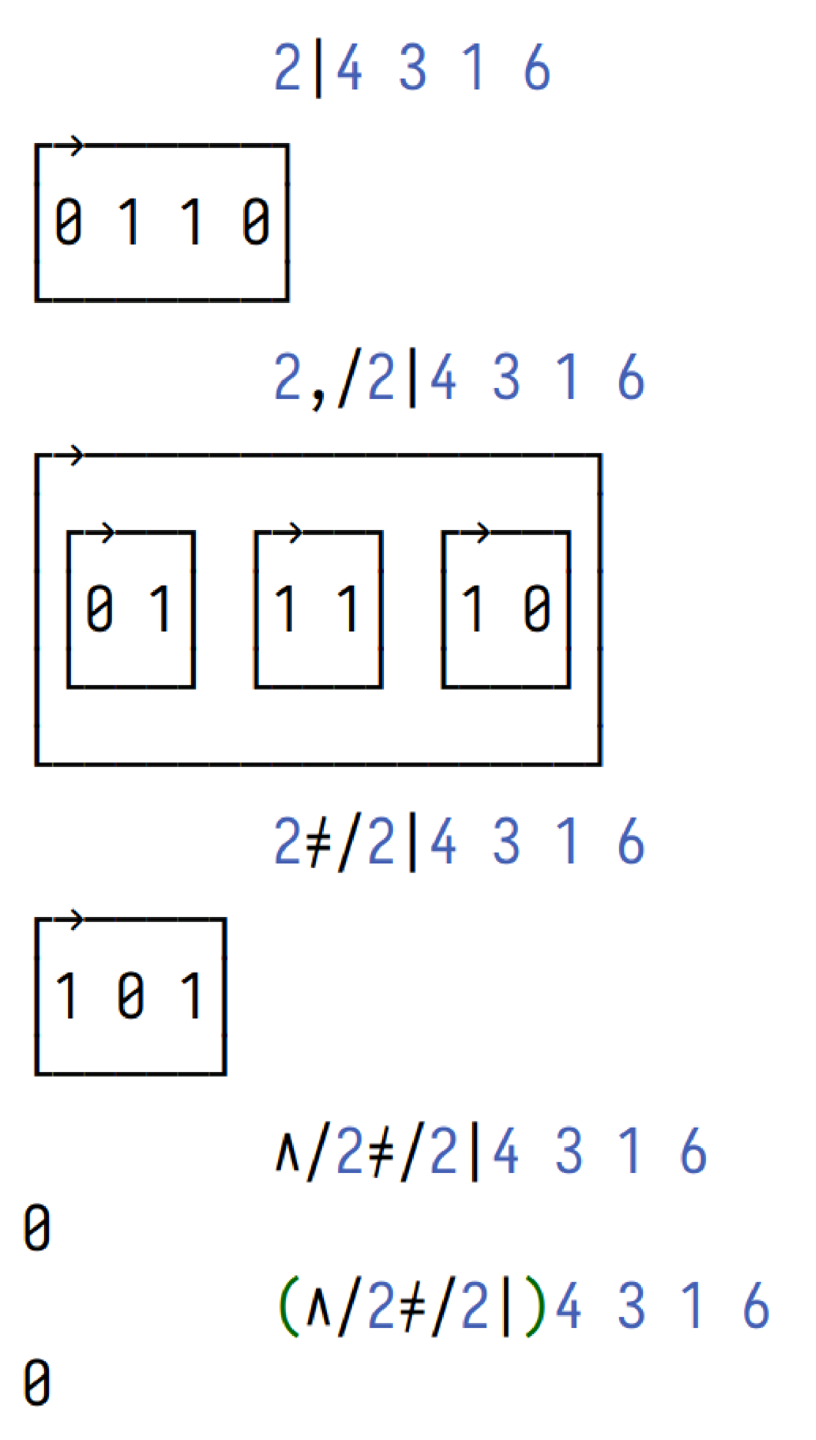
APL, BQN, J, Kap

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



```
        2|4 3 1 6
┌→──────┐
│0 1 1 0│
└───────┘
        2,/2|4 3 1 6
┌→────────────────┐
│┌→──┐ ┌→──┐ ┌→──┐│
││0 1│ │1 1│ │1 0││
│└───┘ └───┘ └───┘│
└─────────────────┘
        2≠/2|4 3 1 6
┌→────┐
│1 0 1│
└─────┘
        ∧/2≠/2|4 3 1 6
0
        (∧/2≠/2|)4 3 1 6
0
```

Jelly

Function Composition

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

Jello

```
> [4,1,6]
```

🟢🟡🔴 Jello 🔴🟡🟢

```
> [4,1,6] :: odd?
                Ḃ
   Ḃ   [4,1,6] ➡️  [0, 1, 0]
```

🟢🟡🔴 Jello 🔴🟡🟢

```
> [4,1,6] :: odd? differ
                Ḃ     Ď
   Ḃ   [4,1,6] ➡️  [0, 1, 0]
   ḂĎ  [4,1,6] ➡️  [1, 1]
```

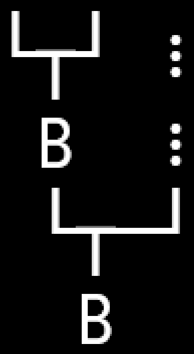
●●● Jello ●●●

```
> [4,1,6] :: odd? differ all
              Ḃ     Ḋ     Ạ
   Ḃ   [4,1,6] ➡  [0, 1, 0]
   ḂḊ  [4,1,6] ➡  [1, 1]
   ḂḊẠ [4,1,6] ➡  1
```

●●● Jello ●●●

```
> [4,1,6] :: odd? differ all
             Ḃ      Ď      Ạ
   Ḃ   [4,1,6] ➡  [0, 1, 0]
   ḂĎ  [4,1,6] ➡  [1, 1]
   ḂĎẠ [4,1,6] ➡  1
```

This is a 1-1-1 monadic chain (BB)



🟢🟡🔴 Jello 🔴🟡🟢

```
> [4,1,6] :: odd? ≠ prior and fold
              Ḃ   n    ṕ    a   /
```

and fold can be replaced with all

☝️🥳 algorithm advisor 🥳☝️

≠ prior can be replaced with differ

☝️🥳 algorithm advisor 🥳☝️

```
Ḃ     [4,1,6] ➡️  [0, 1, 0]
Ḃnṕ   [4,1,6] ➡️  [1, 1]
Ḃnṕa/ [4,1,6] ➡️  1
 This is a 1-2-q-2-q monadic chain (BB)
           ⁝ └┬┘ └┬┘
           ⁝ h₁  h₁
           └┬─┘   ⁝
            B     ⁝
            └──┬──┘
               B
```

Uiua

Function Composition

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

Kap	`∧/2≠/2|`
Dyalog APL	`∧/2≠/2|⊢`
Uiua	`/↧≡/≠◫2◿2`
BQN	`∧´·≠´˘2↕2|⊢`
J	`[: *./ 2~:/\ 2|[`
Jello	`odd? differ all`

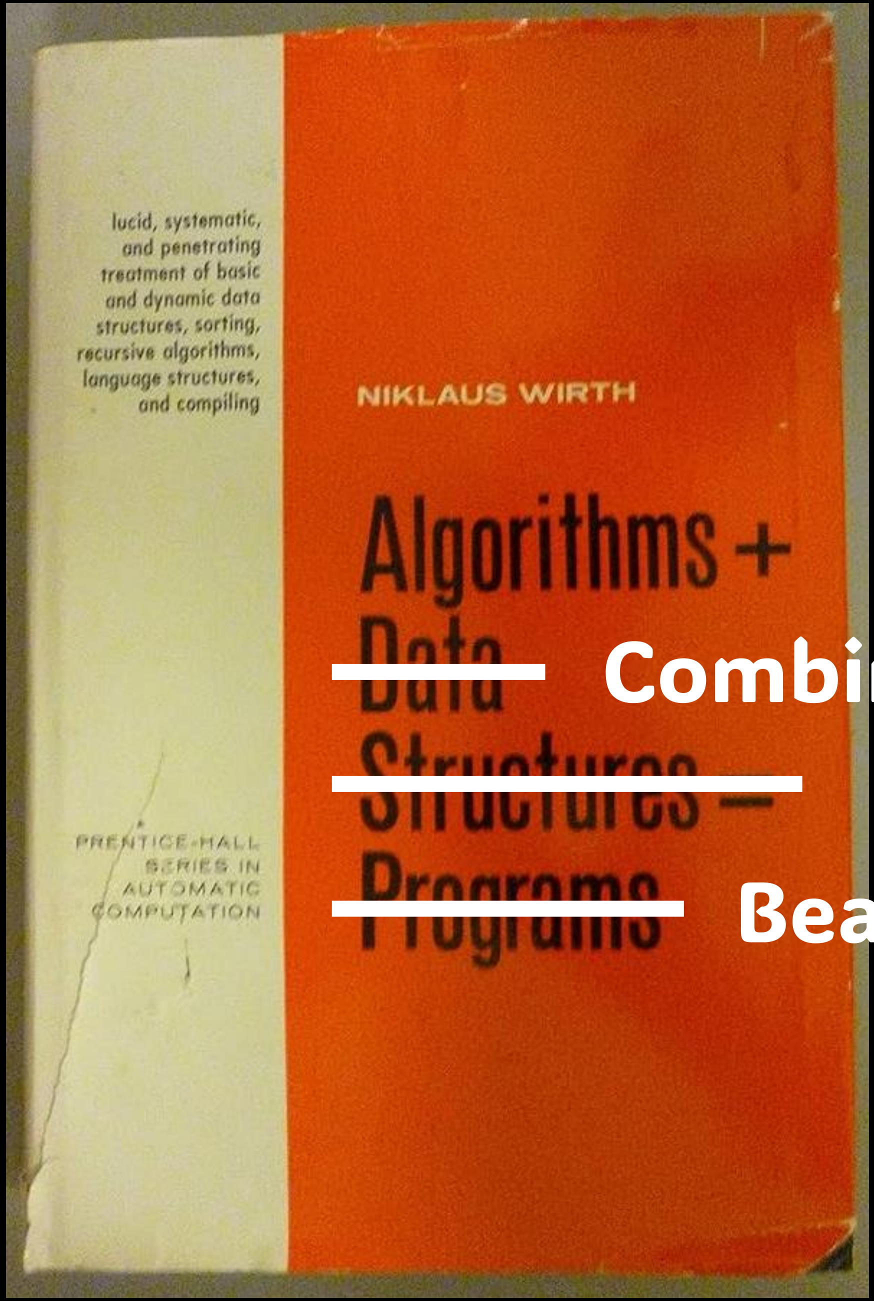
				APL	Kap	J	B	Jello	
Functions	✓	✓	✓	✗	✗	✗	✗	✗	✗
Operators	✗	✓	✓	✓	✓	✓	✓	✓	✓
Trains	✗	✗	✗	✓	✓	✓	✓	✗	✗
Chains	✗	✗	✗	✗	✗	✗	✗	✓	✗
Stacks	✗	✗	✗	✗	✗	✗	✗	✗	✓

In Conclusion

Kap	`∧/2≠/2|`
Dyalog APL	`∧/2≠/2|⊢`
Uiua	`/↧≡/≠◫2◿2`
BQN	`∧´·≠´˘2↕2|⊢`
J	`[: *./ 2~:/\ 2|[`
Jello	`odd? differ all`

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*


lucid, systematic, and penetrating treatment of basic and dynamic data structures, sorting, recursive algorithms, language structures, and compiling
NIKLAUS WIRTH
Algorithms +
~~Data~~
~~Structures =~~
~~Programs~~
PRENTICE-HALL SERIES IN AUTOMATIC COMPUTATION

Combinators =

Beautiful Code

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

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

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

```
●●● Jello ●●●

> [4,1,6] :: odd? differ all
              Ḃ     Ď     Ạ
   Ḃ   [4,1,6] ➡   [0, 1, 0]
   ḂĎ  [4,1,6] ➡   [1, 1]
   ḂĎẠ [4,1,6] ➡   1
    This is a 1-1-1 monadic chain (BB)
              └┬┘ ⋮
               B  ⋮
               └┬─┘
                B
```

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

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

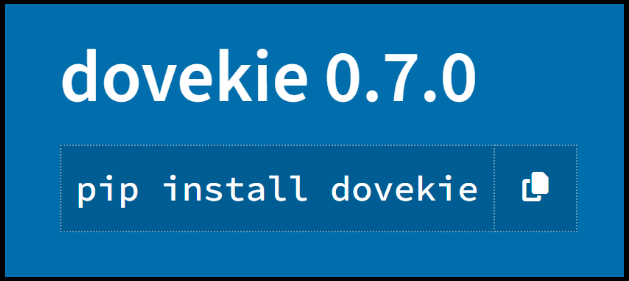
<table>
<tr><td>Kap</td><td>∧/2≠/2|</td></tr>
<tr><td>Dyalog APL</td><td>∧/2≠/2|⊢</td></tr>
<tr><td>Uiua</td><td>/↧≡/≠◫2◿2</td></tr>
<tr><td>BQN</td><td>∧´·≠´˘2↕2|⊢</td></tr>
<tr><td>J</td><td>[: *./ 2~:/\ 2|[</td></tr>
<tr><td>Jello</td><td>odd? differ all</td></tr>
</table>

www.combinatorylogic.com

Dovekie

Blackbird

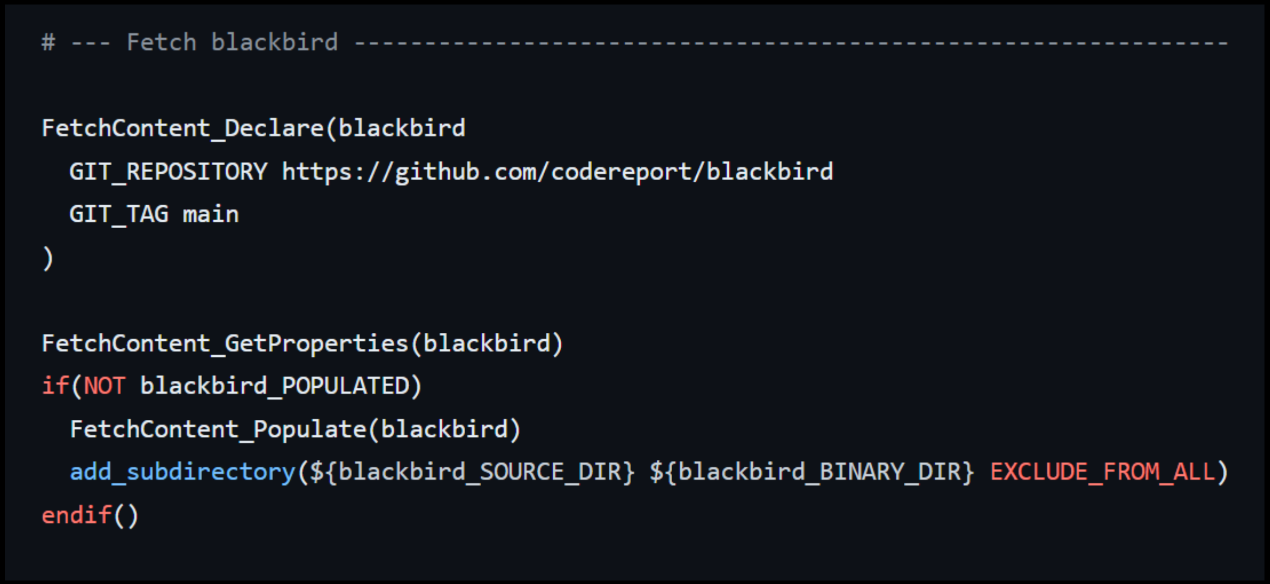
Dovekie



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()
```

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