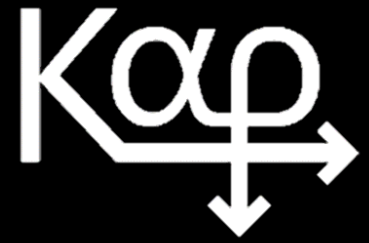
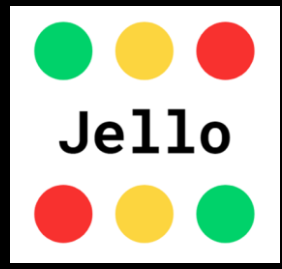
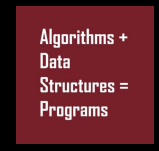
Composition Intuition II





Conor Hoekstra

code_report | codereport

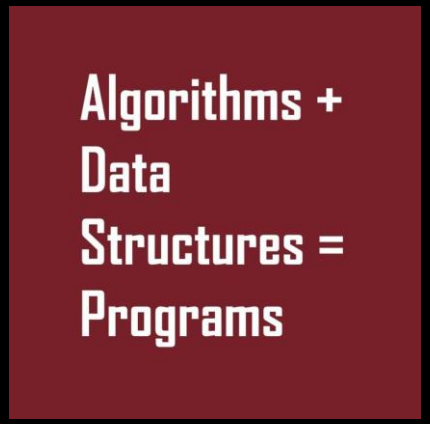


About Me

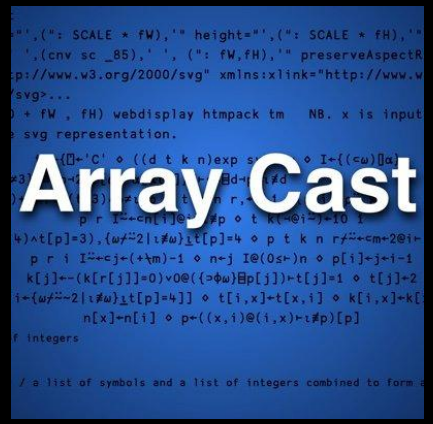
Conor Hoekstra / @code_report

348 Videos

40 (28) Talks



192 Episodes
@adspthepodcast



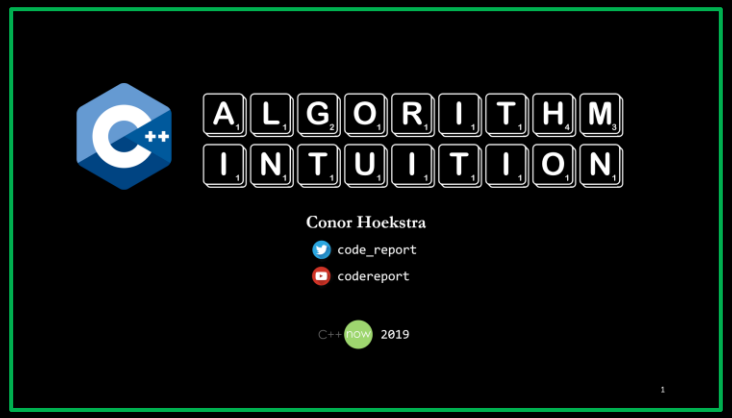
84 Episodes
@arraycast

3 Episodes
@codereport

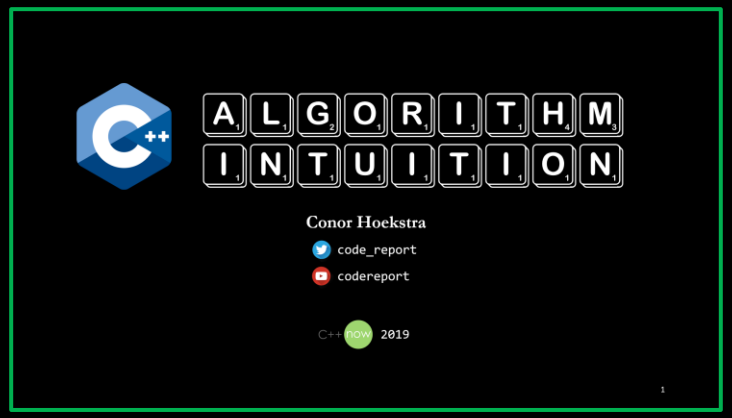
19 Episodes
@conorhoekstra

https://github.com/codereport/Content

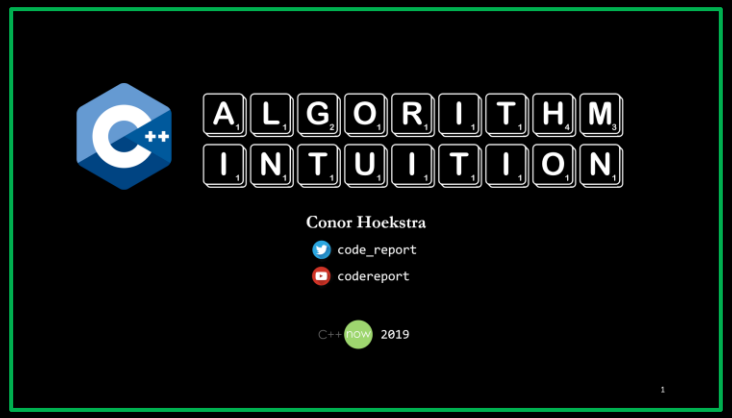
The Algorithm Intuition Trilogy



The Algorithm Intuition Trilogy



The Algorithm Intuition Trilogy

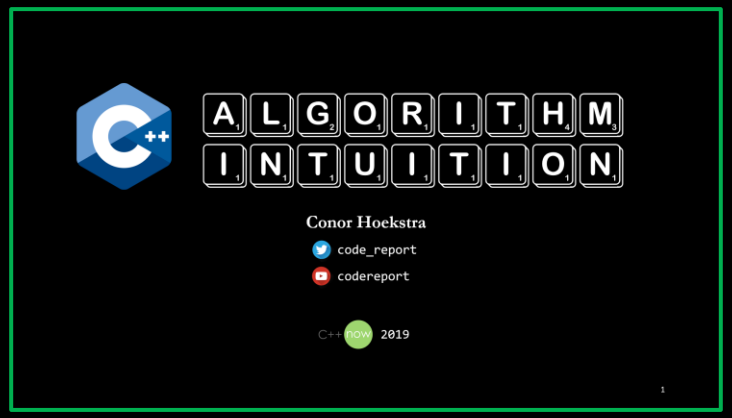


C++ now 2019

Meeting C++ 2019

Cpp North 2022

The Algorithm Intuition Trilogy

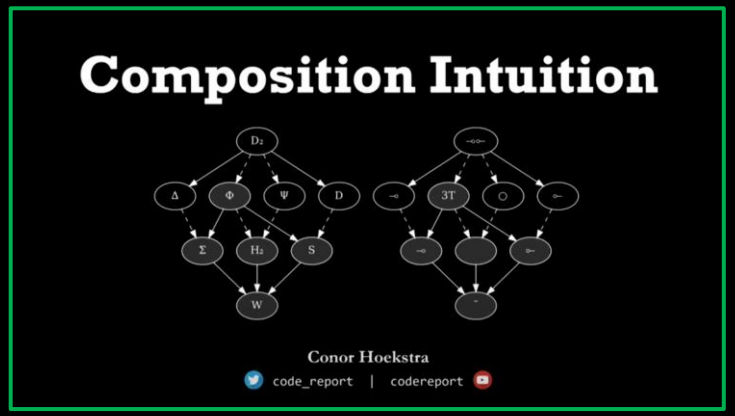


C++ now 2019

Meeting C++ 2019

Cpp North 2022

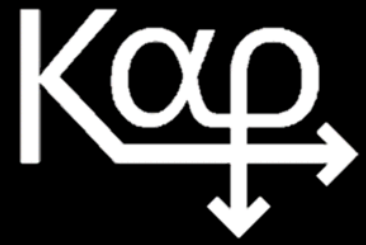
The Composition Intuition Trilogy

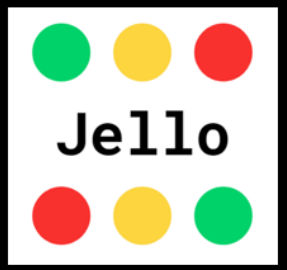


Cpp North 2023

Cpp North 2024

Composition Intuition II





Conor Hoekstra

code_report | codereport

Agenda

Warm Up
Combinator Recap++
Example #1
Example #2

Warm Up

Warm Up

Sort Letters

https://theweeklychallenge.org/blog/perl-weekly-challenge-279

Task 1: Sort Letters

You are given two arrays, `@letters` and `@weights`.

Write a script to sort the given array `@letters` based on the `@weights`.

Example 1

```
Input: @letters = ('R', 'E', 'P', 'L')
       @weights = (3, 2, 1, 4)
Output: PERL
```

Example 2

```
Input: @letters = ('A', 'U', 'R', 'K')
       @weights = (2, 4, 1, 3)
Output: RAKU
```

Example 3

```
Input: @letters = ('O', 'H', 'Y', 'N', 'P', 'T')
       @weights = (5, 4, 2, 6, 1, 3)
Output: PYTHON
```

3 2 1 4

R E P L

1 2 3 4

P E R L

PERL

4 5 6 7 8 3 2 1

N O R T H P P C

1 2 3 4 5 6 7 8

C P P N O R T H

CPPNORTH

```cpp
auto sort_string(std::span<int> nums, std::string_view chars) -> std::string {
  // ...
}
```

```cpp
auto sort_string(std::span<int> nums, std::string_view chars) -> std::string {
  auto pairs = std::vector<std::pair<int, char>>(nums.size());
  auto str   = std::string(nums.size(), ' ');
  // ...
}
```

```cpp
auto sort_string(std::span<int> nums, std::string_view chars) -> std::string {
  auto pairs = std::vector<std::pair<int, char>>(nums.size());
  auto str   = std::string(nums.size(), ' ');
  std::transform(nums.cbegin(), nums.cend(), chars.begin(), pairs.begin(),
                 [](auto i, auto c) { return std::pair{i, c};});
  // ...
}
```

```cpp
auto sort_string(std::span<int> nums, std::string_view chars) -> std::string {
  auto pairs = std::vector<std::pair<int, char>>(nums.size());
  auto str   = std::string(nums.size(), ' ');
  std::transform(nums.cbegin(), nums.cend(), chars.begin(), pairs.begin(),
                 [](auto i, auto c) { return std::pair{i, c};});
  std::sort(pairs.begin(), pairs.end());
  // ...
}
```

```cpp
auto sort_string(std::span<int> nums, std::string_view chars) -> std::string {
  auto pairs = std::vector<std::pair<int, char>>(nums.size());
  auto str   = std::string(nums.size(), ' ');
  std::transform(nums.cbegin(), nums.cend(), chars.begin(), pairs.begin(),
                 [](auto i, auto c) { return std::pair{i, c};});
  std::sort(pairs.begin(), pairs.end());
  std::transform(pairs.cbegin(), pairs.cend(), str.begin(),
                 [](auto p) { return std::get<1>(p); });
  // ...
}
```

```cpp
auto sort_string(std::span<int> nums, std::string_view chars) -> std::string {
  auto pairs = std::vector<std::pair<int, char>>(nums.size());
  auto str   = std::string(nums.size(), ' ');
  std::transform(nums.cbegin(), nums.cend(), chars.begin(), pairs.begin(),
                 [](auto i, auto c) { return std::pair{i, c};});
  std::sort(pairs.begin(), pairs.end());
  std::transform(pairs.cbegin(), pairs.cend(), str.begin(),
                 [](auto p) { return std::get<1>(p); });
  return str;
}
```

```cpp
auto sort_string(std::span<int> nums, std::string_view chars) -> std::string {
  auto pairs = std::vector<std::pair<int, char>>(nums.size());
  auto str   = std::string(nums.size(), ' ');
  std::ranges::transform(nums, chars, pairs.begin(),
                         [](auto i, auto c) { return std::pair{i, c}; });
  std::ranges::sort(pairs);
  std::ranges::transform(pairs, str.begin(),
                         [](auto p) { return std::get<1>(p); });
  return str;
}
```

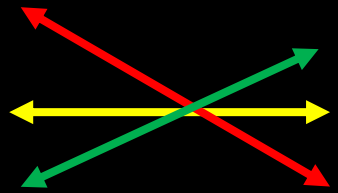
```cpp
namespace rv = std::ranges::views; // std::views
namespace v3 = ranges;

auto sort_string(std::span<int> nums, std::string_view chars) -> std::string {
  auto pairs = rv::zip(nums, chars) | v3::to<std::vector>;
  std::ranges::sort(pairs);
  return rv::transform(pairs, [](auto p) { return std::get<1>(p); }) |
         v3::to<std::string>;
}
```

```cpp
namespace rv = std::ranges::views; // std::views
namespace v3 = ranges;

auto sort_string(std::span<int> nums, std::string_view chars) -> std::string {
  return rv::zip(nums, chars)                                    //
         | v3::to<std::vector>                                   //
         | ranges::action::sort                                  //
         | rv::transform([](auto p) { return std::get<1>(p); }) //
         | v3::to<std::string>;
}
```

```cpp
namespace rv = std::ranges::views; // std::views
namespace v3 = ranges;

auto sort_string(std::span<int> nums,
                 std::string_view chars) -> std::string {
  return rv::zip(nums, chars)   //
         | v3::to<std::vector>  //
         | ranges::action::sort //
         | rv::transform(_snd)  //
         | v3::to<std::string>;
}
```

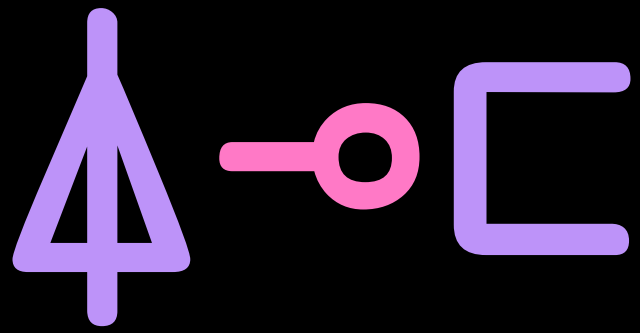
```
sortString = map snd .: sortOn fst .: zip
```

```cpp
auto sort_string(std::vector<int> nums,
                 std::string_view chars)
                -> std::string {
  return rv::zip(nums, chars)   //
         | v3::to<std::vector>  //
         | ranges::action::sort //
         | rv::transform(_snd)  //
         | v3::to<std::string>;
}
```

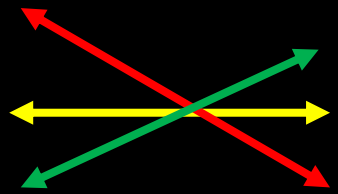


```
sortString
=   map snd
.:  sortOn fst
.:  zip
```

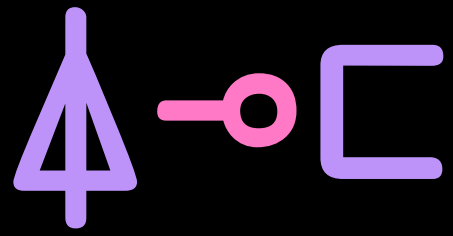
$$\{(\exists x)\sqsubset x\}$$



```cpp
auto sort_string(std::vector<int> nums,
                 std::string_view chars)
                -> std::string {
  return rv::zip(nums, chars)   //
         | v3::to<std::vector>  //
         | ranges::action::sort //
         | rv::transform(_snd)  //
         | v3::to<std::string>;
}
```



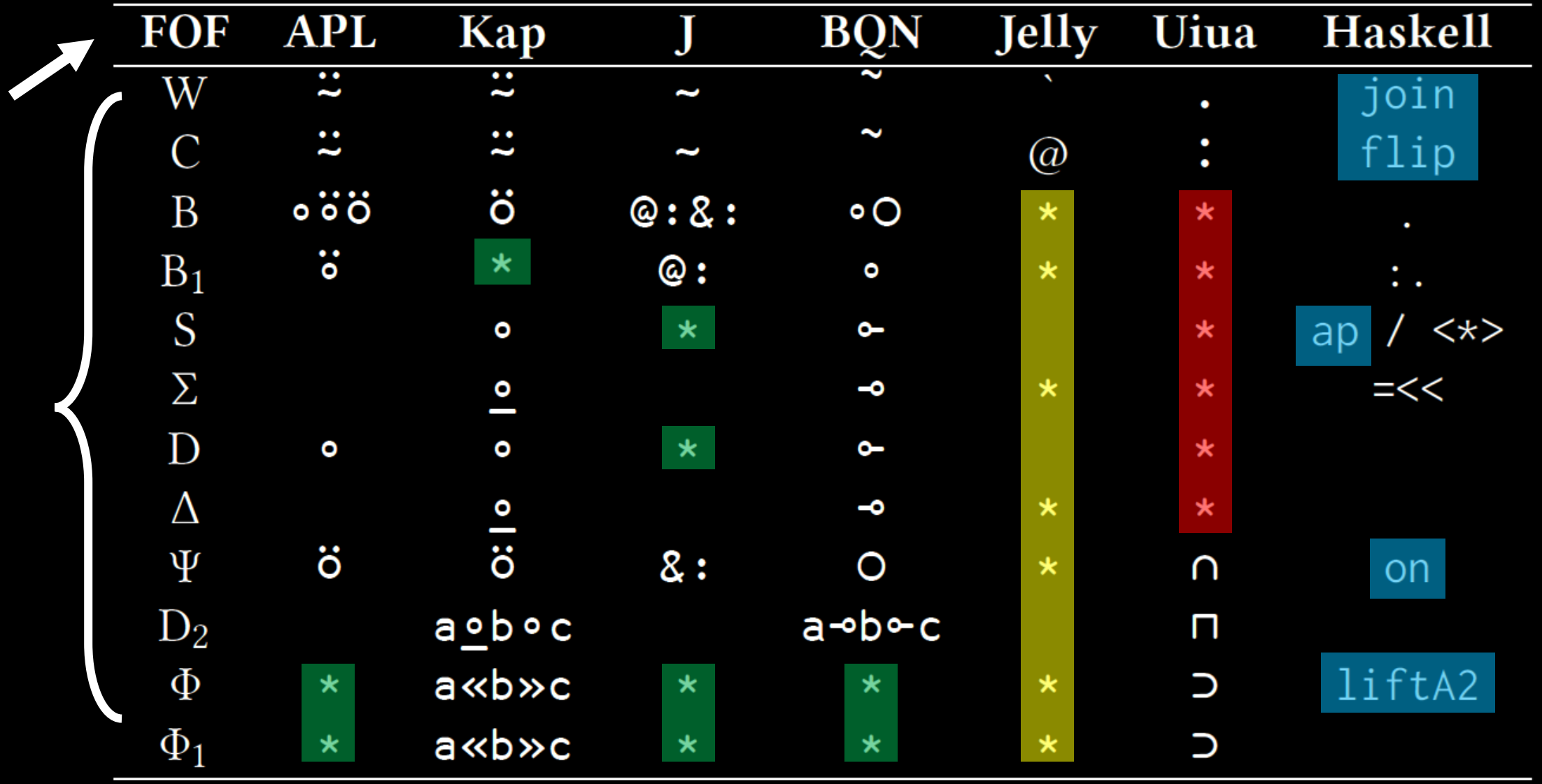
```
sortString
=   map snd
.:  sortOn fst
.:  zip
```



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*

combinator: a function that deals only in its arguments

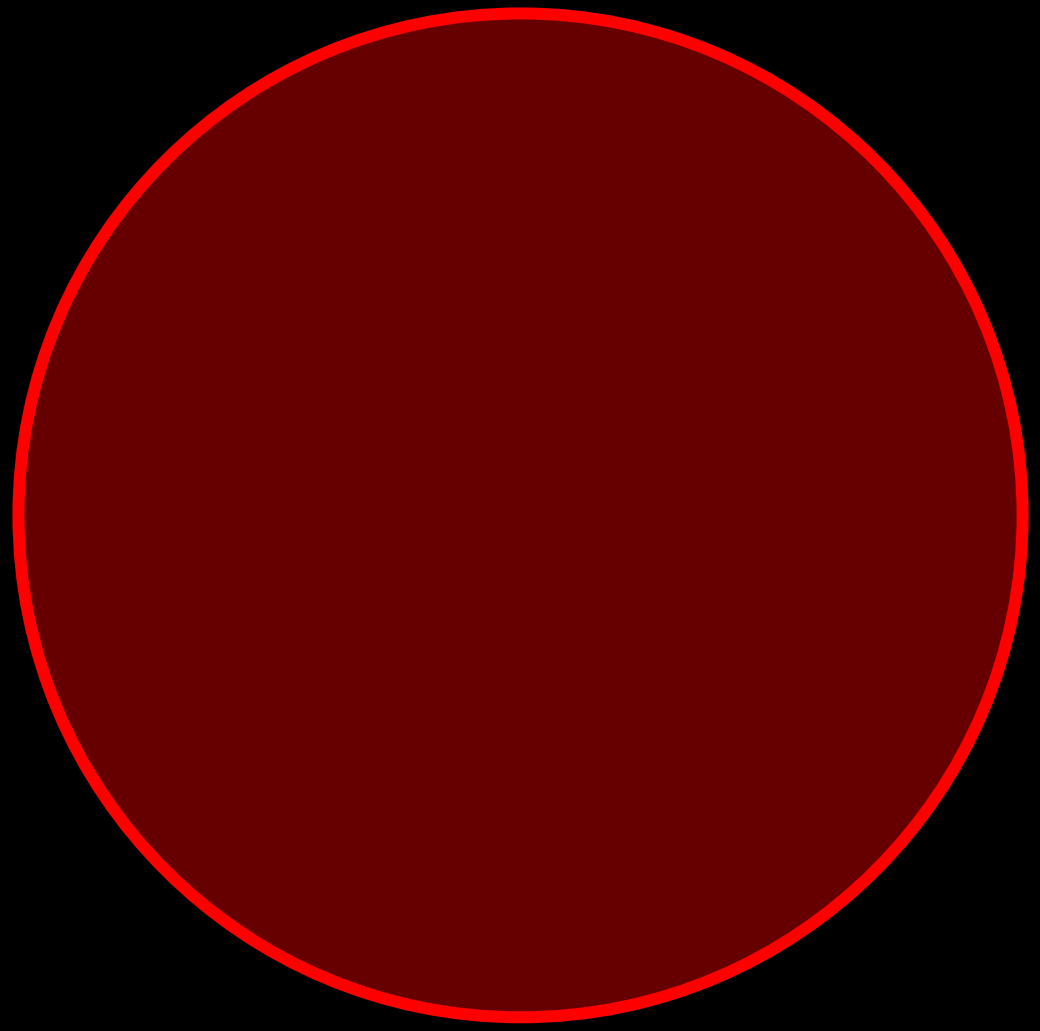
CL combinator: a combinator from Combinatory Logic

FOF: a combinator that only consumes AND produces functions

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

HOF: consumes OR produces a function

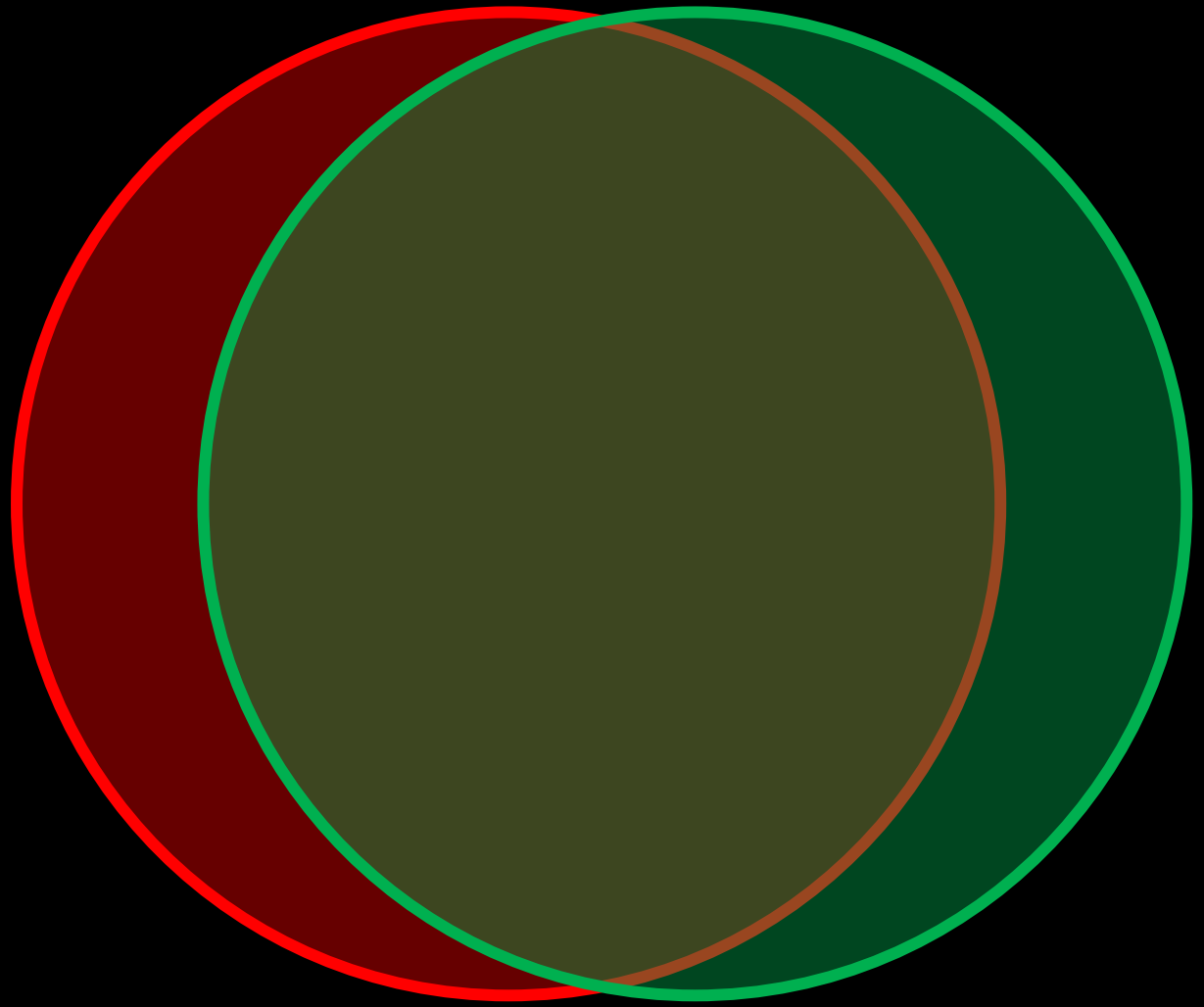
pure function



combinator

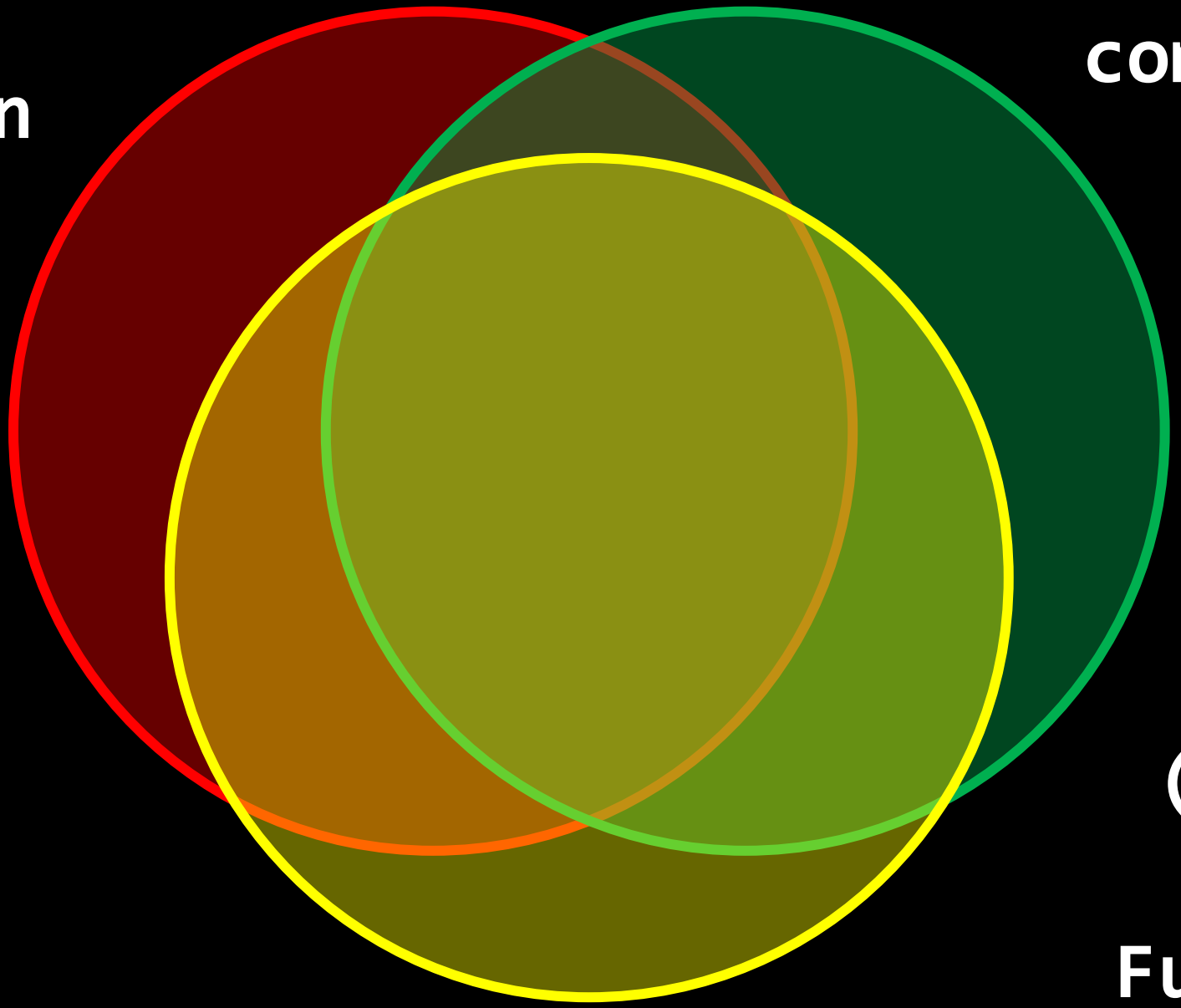
pure function

combinator

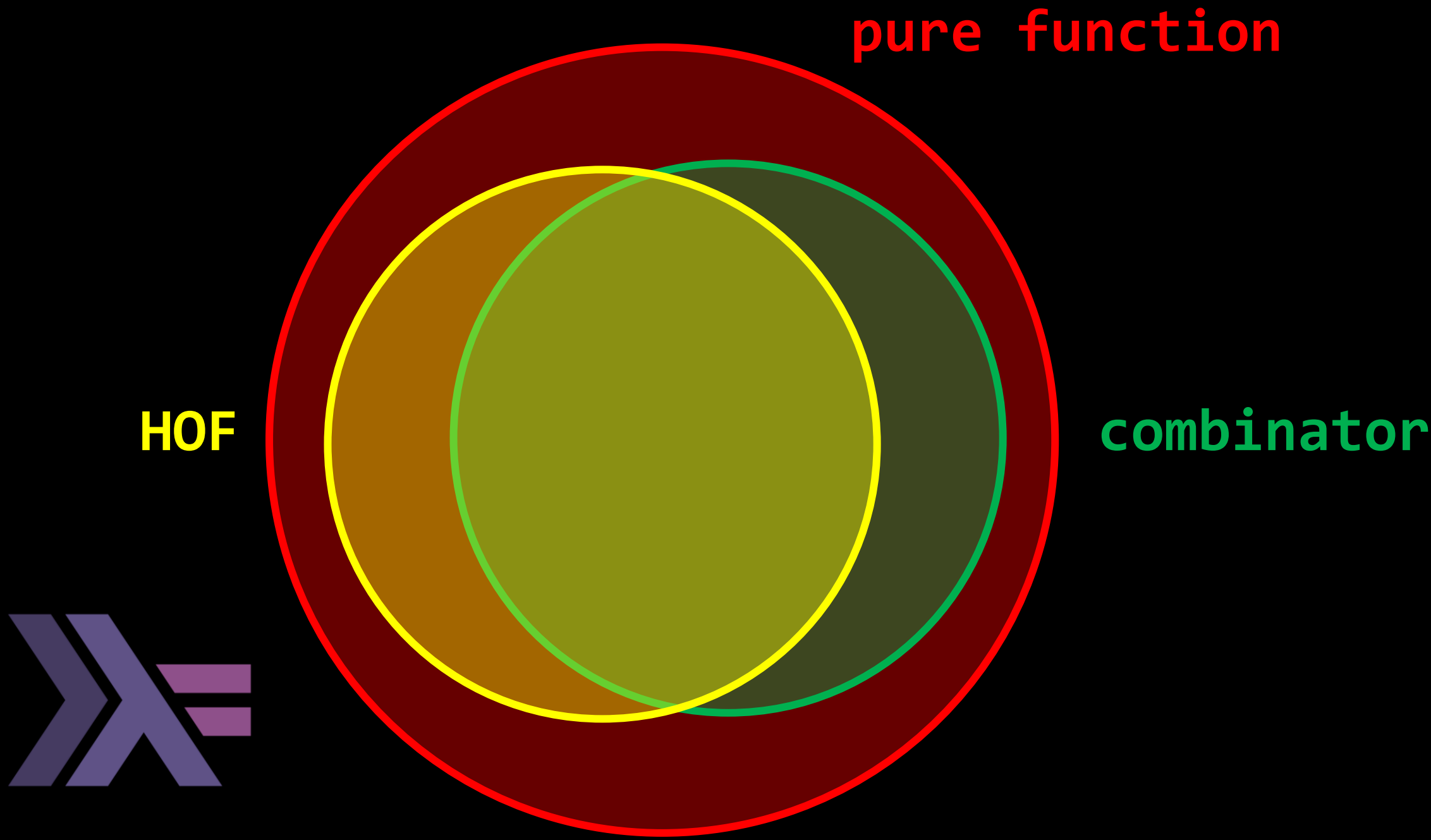


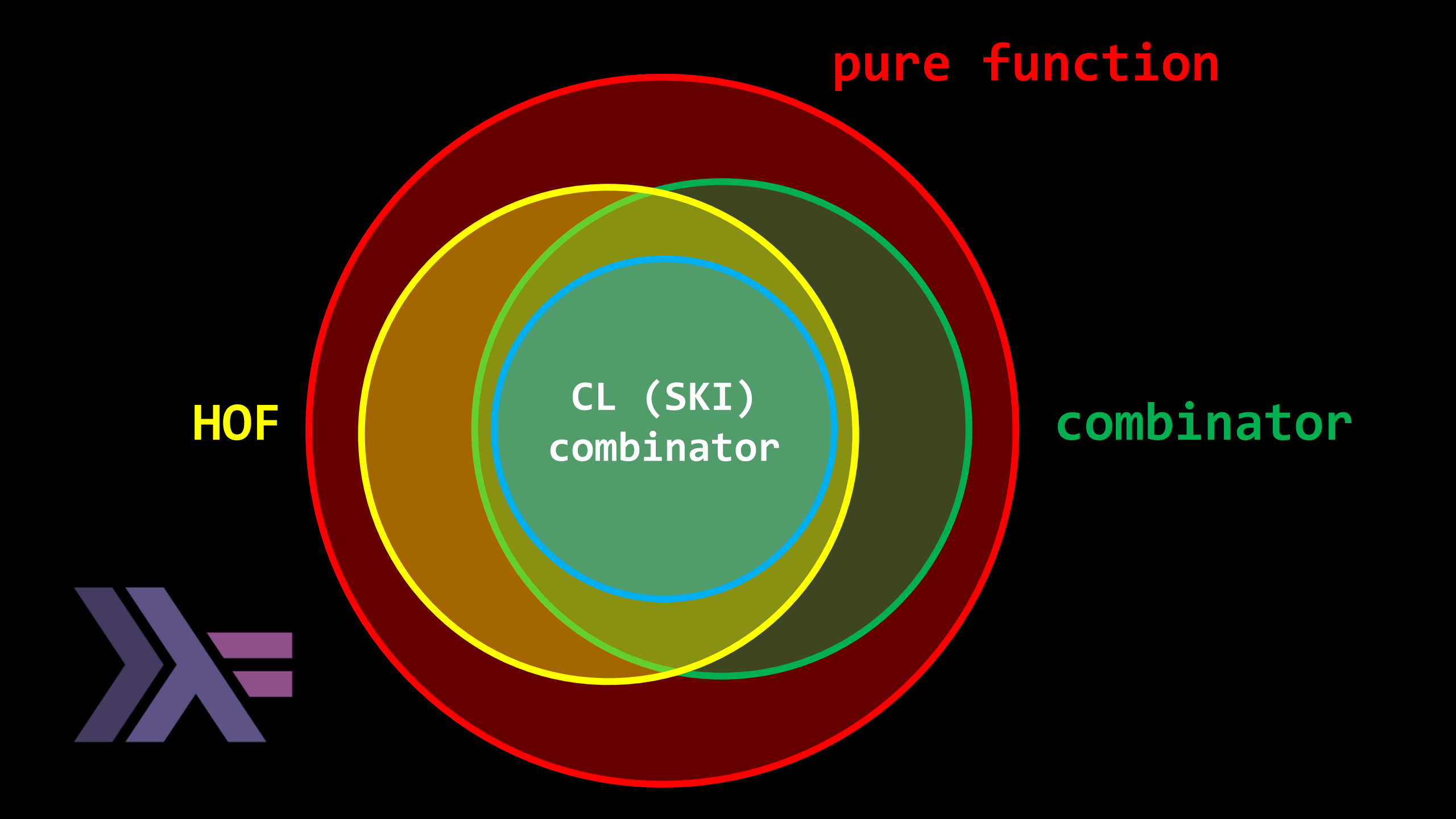
pure function

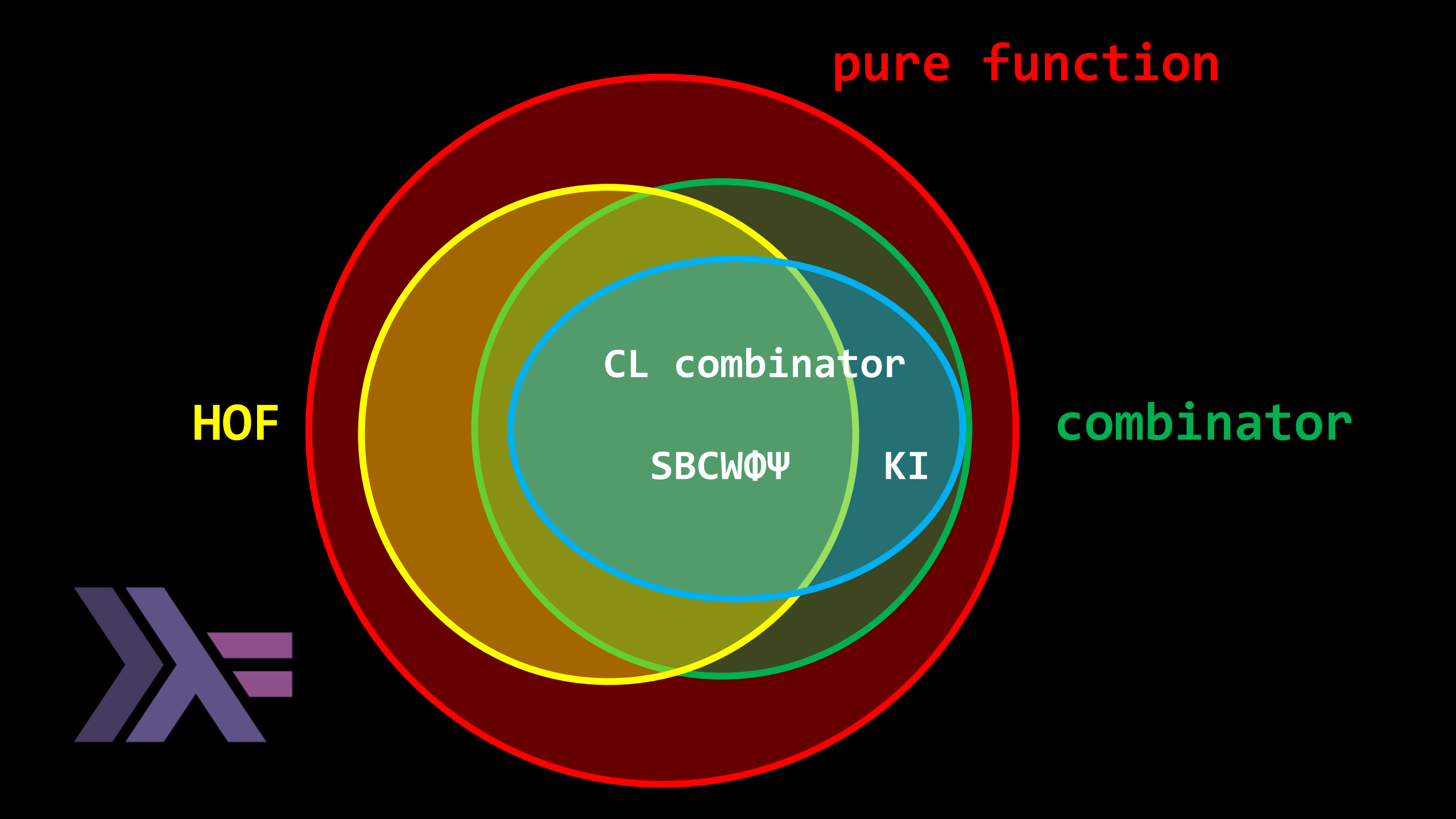
combinator

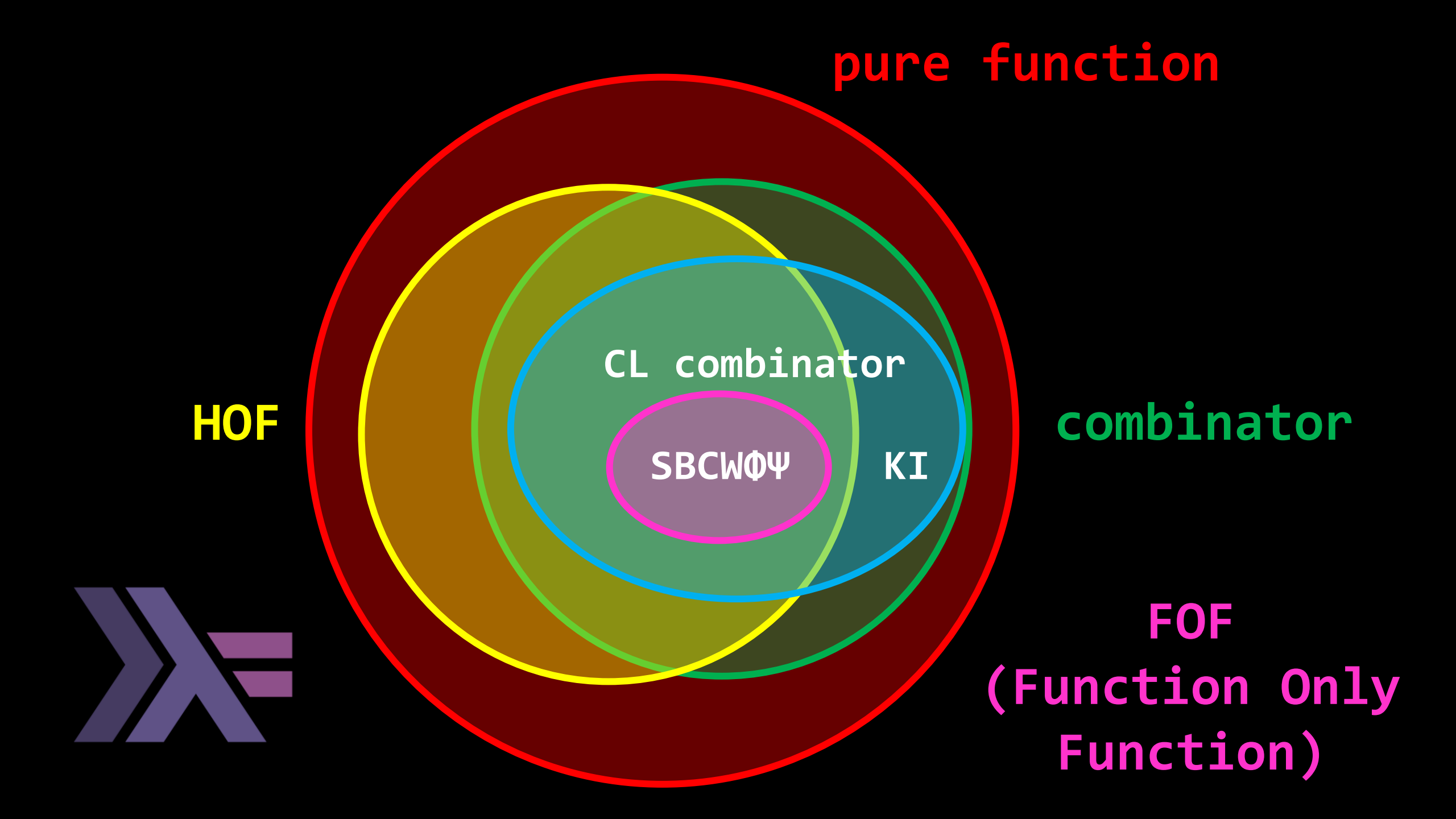


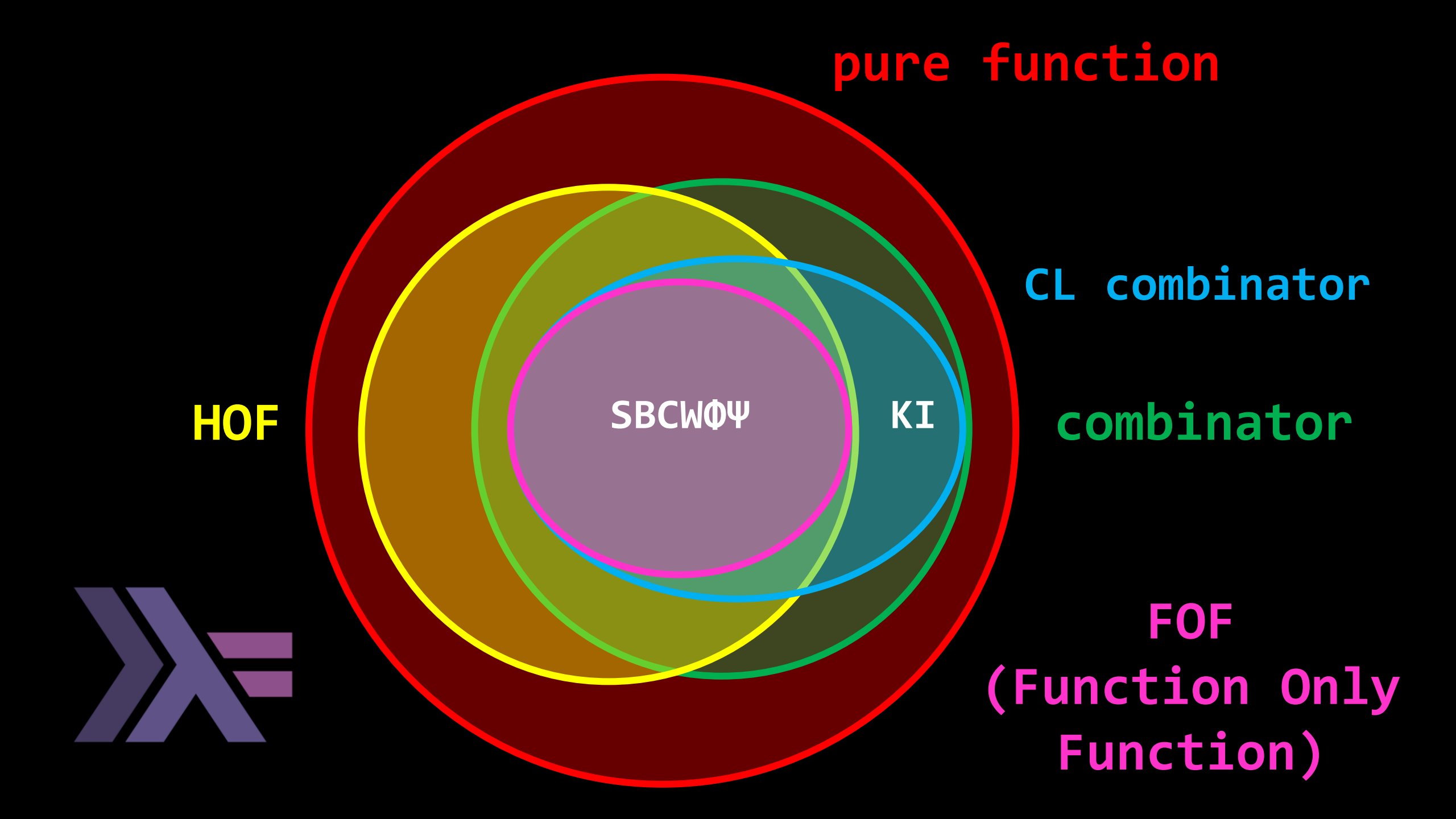
HOF (Higher Order Function)


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

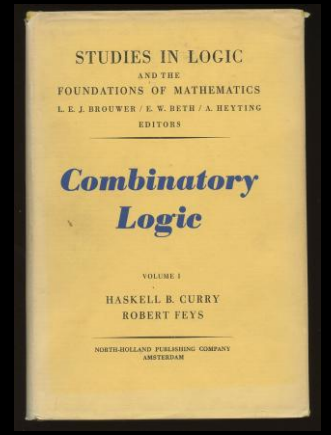
pure function: same input = same output / no side effects

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*

6 2 46

[[end of digression]]

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

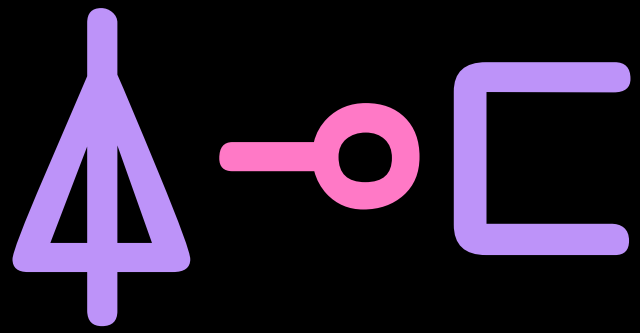
```
def b(f, g):
    return lambda x: f(g(x))
```

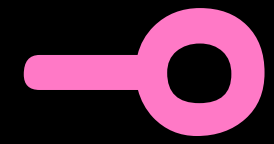
```
def c(f):
    return lambda x, y: f(y, x)
```

```
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))
```

Revisiting the Warmup



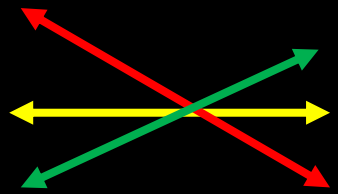


```
_Delta_ ← {(𝔽w)𝔾x}
 # Before (Dyadic)
 # Reverse Hook (I)
```

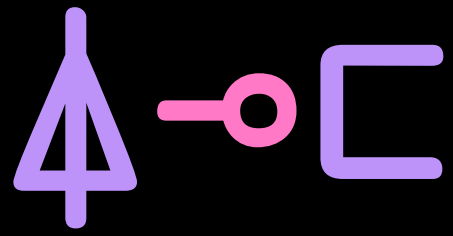
Live Coding

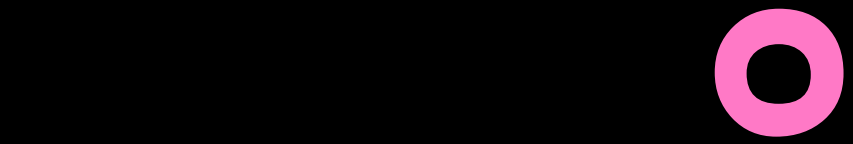
```
_Delta_ ← {(𝔽w)𝔾x} # ⊸

SortString ← ⍋ _Delta_ ⊏
```

```cpp
auto sort_string(std::vector<int> nums,
                 std::string_view chars)
                -> std::string {
  return rv::zip(nums, chars)   //
         | v3::to<std::vector>  //
         | ranges::action::sort //
         | rv::transform(_snd)  //
         | v3::to<std::string>;
}
```



```
sortString
=   map snd
.:  sortOn fst
.:  zip
```





```
_B1_ ← {𝔽𝕨𝔾𝕩}
# Atop (Dyadic)
```

Live Coding

```
_B1_ ← {𝔽w𝔾x} # ∘ & 2 Train
Zip  ← ⋈¨
Sort ← ∧
Snd  ← ⊑⌽

SortString ← Snd¨ Sort _B1_ Zip
```

Example #1

Example #1
Special Array

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

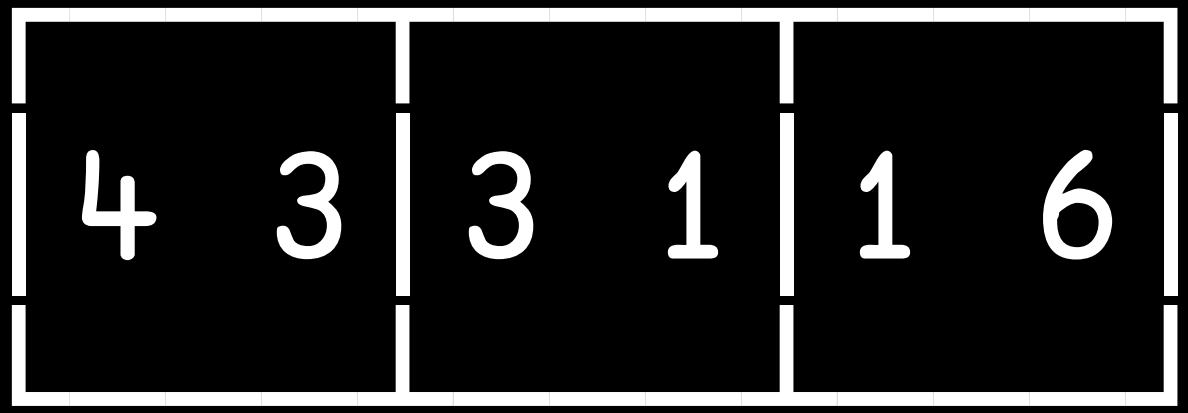
3151. Special Array I

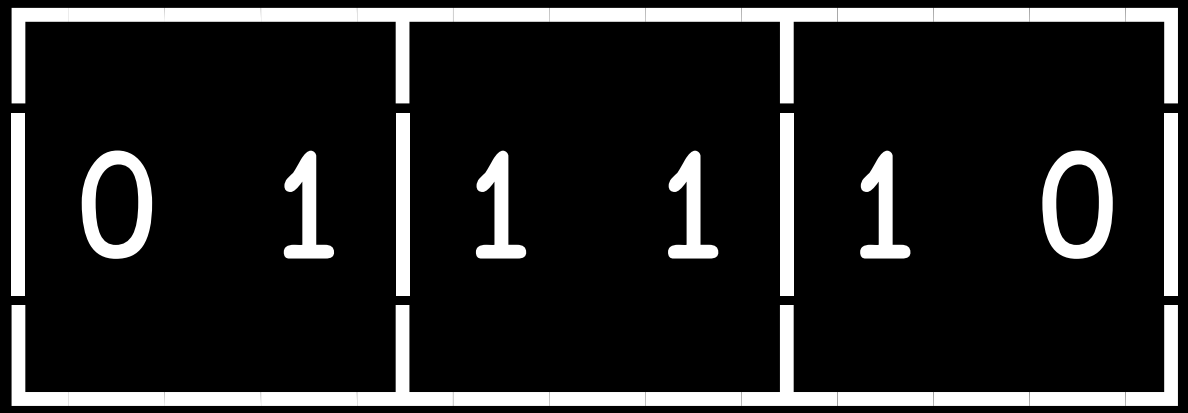
Easy | Topics | Companies | Hint

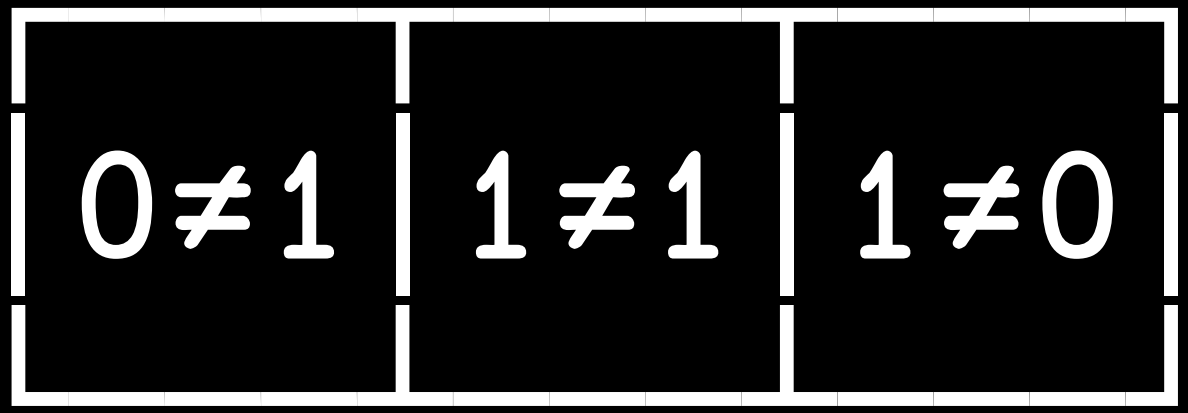
An array is considered **special** if every pair of its adjacent elements contains two numbers with different parity.

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

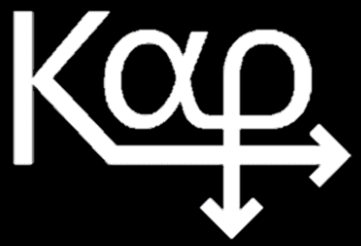

0 1
1 1
1 0

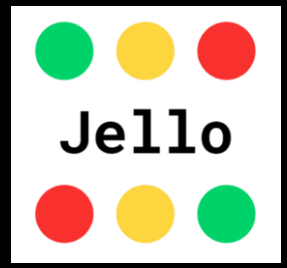

0≠1
1≠1
1≠0

1 0 1

0

APL


Kαρ


Jello

Python & C++

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:]))
```

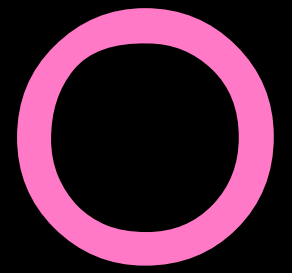
```cpp
namespace rv = std::ranges::views; // std::views

auto is_array_special(std::vector<int> nums) -> bool {
  return std::ranges::all_of(             //
      rv::zip(nums, nums | rv::drop(1)) //
          | rv::transform([](auto t) {
              auto const [a, b] = t;
              return a % 2 != b % 2;
            }),
      std::identity{});
}
```

```cpp
namespace rv = std::ranges::views; // std::views

auto is_array_special(std::vector<int> nums) -> bool {
  return std::ranges::all_of(
    rv::zip_transform(
       [](auto a, auto b) { return a % 2 != b % 2; },
       nums, nums | rv::drop(1)),
    std::identity{});
}
```

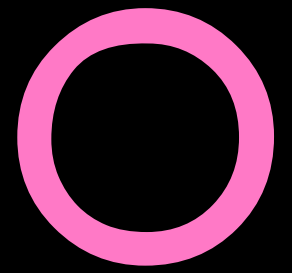
```cpp
namespace rv = std::ranges::views; // std::views

auto is_array_special(std::vector<int> nums) -> bool {
  return std::ranges::all_of(
    rv::adjacent_transform<2>(
      nums,
      [](auto a, auto b) { return a % 2 != b % 2; }),
    std::identity{});
}
```

```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_ ← {(𝔾x)𝔽𝔾x}
      # Over
    # on (Haskell)
```



```
_Psi_ ← {(𝔾𝕩)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:]))
```

```cpp
namespace rv = std::ranges::views; // std::views

auto is_array_special(std::vector<int> nums) -> bool {
  return std::ranges::all_of(
    rv::adjacent_transform<2>(
      nums,
      [](auto a, auto b) { return a % 2 != b % 2; }),
    std::identity{});
}
```

```cpp
namespace rv = std::ranges::views; // std::views

auto is_array_special(std::vector<int> nums) -> bool {
  return std::ranges::all_of(
    rv::adjacent_transform<2>(nums, _psi(_neq_, _odd)), _id);
}
```

```cpp
namespace rv = std::ranges::views; // std::views

auto is_array_special(std::vector<int> nums) -> bool {
  return std::ranges::all_of(
    rv::pairwise_transform(nums, _psi(_neq_, _odd)), _id);
}
```

```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)))
```

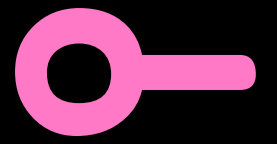
```python
from dovekie import odd
from operator import ne
from itertools import pairwise

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

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

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)
```

APL, BQN, J, Kap

Function Composition

❌ Functions

✅ Operators

✅ Trains

❌ Chains

❌ Stacks

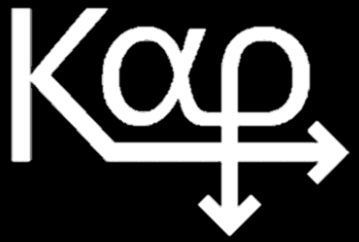


Table 4: 2 and 3-trains in APL, Kap, BQN and J.

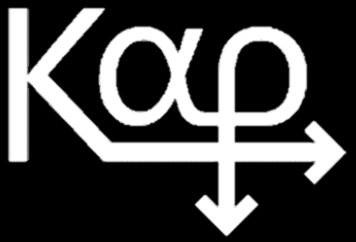
Year	Language	2-Train	3-Train
1990	J	S and D	Φ and Φ_1
2014	Dyalog APL	B and B_1	Φ and Φ_1
2020	Kap	B and B_1	-
2020	BQN	B and B_1	Φ and Φ_1

```
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

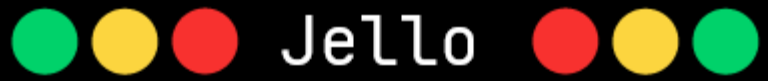
❌ Functions

✅ Operators

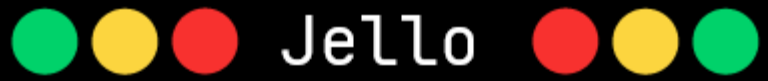
❌ Trains

✅ Chains

❌ Stacks

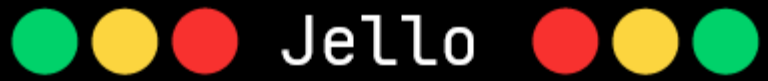



```
> [4,1,6]
```



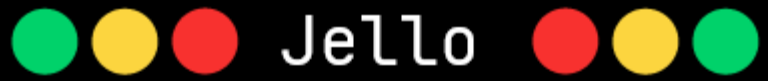

```
> [4,1,6] :: odd?
                Ḃ
    Ḃ    [4,1,6] ➡️  [0, 1, 0]
```

🟢🟡🔴 Jello 🔴🟡🟢



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

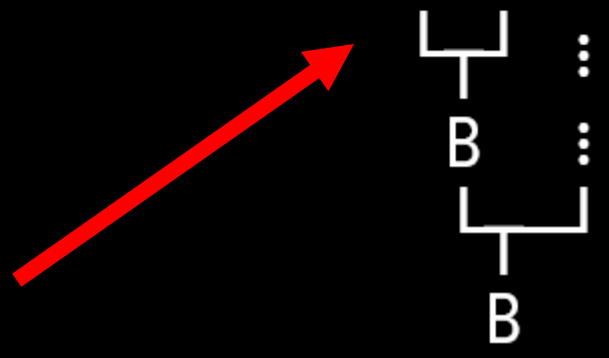
🟢🟡🔴 Jello 🔴🟡🟢



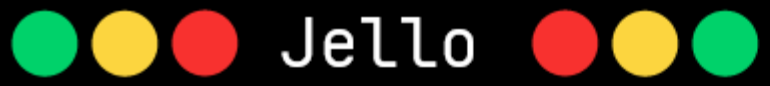
```
> [4,1,6] :: odd? differ all
                 Ḃ      Ď      Ạ
   Ḃ    [4,1,6] ➡  [0, 1, 0]
   ḂĎ   [4,1,6] ➡  [1, 1]
   ḂĎẠ  [4,1,6] ➡  1
```

Jello

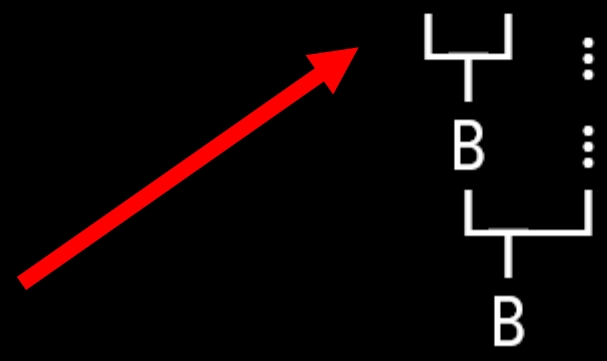
```
> [4,1,6] :: odd? differ all
                Ḃ      Ď      Ạ
  Ḃ    [4,1,6] ➡  [0, 1, 0]
  ḂĎ   [4,1,6] ➡  [1, 1]
  ḂĎẠ  [4,1,6] ➡  1
   This is a 1-1-1 monadic chain (BB)
```



B → 1y11
B_1 → 2y12
S → 1y21
∑ → 1y12
D → 2y21
Δ → 2y21
Ψ → 2y21




```
> [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	→	1y11
B_1	→	2y12
S	→	1y21
Σ	→	1y12
D	→	2y21
Δ	→	2y21
Ψ	→	2y21

🟢🟡🔴 Jello 🔴🟡🟢

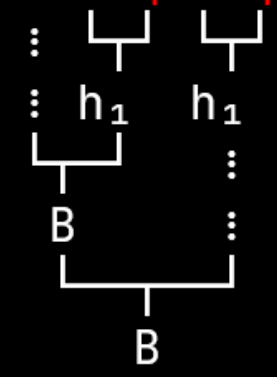
```
> [4,1,6] :: odd? ≠ prior and fold
             Ḃ   n    ṗ    a   /
```

and fold can be replaced with all
🤞🥳 algorithm advisor 🥳🤞

≠ prior can be replaced with differ
🤞🥳 algorithm advisor 🥳🤞

```
Ḃ      [4,1,6] ➡  [0, 1, 0]
Ḃnṗ    [4,1,6] ➡  [1, 1]
Ḃnṗa/  [4,1,6] ➡  1
```

This is a 1-2-q-2-q monadic chain (BB)



Uiua

Function Composition

❌ Functions
✅ Operators
❌ Trains
❌ Chains
✅ Stacks

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
```

Example #2

Example #2
Minimum Average

https://leetcode.com/problems/minimum-average-of-smallest-and-largest-elements

3194. Minimum Average of Smallest and Largest Elements

Easy | Topics | Companies | Hint

You have an array of floating point numbers `averages` which is initially empty. You are given an array `nums` of `n` integers where `n` is even.

You repeat the following procedure `n / 2` times:

- Remove the **smallest** element, `minElement`, and the **largest** element `maxElement`, from `nums`.
- Add `(minElement + maxElement) / 2` to `averages`.

Return the **minimum** element in `averages`.

7 8 3 4 15 13 4 1

1 3 4 4 7 8 13 15

1 3 4 4

15 13 8 7

16 16 12 11

11

5.5

```cpp
namespace rv = std::ranges::views; // std::views

auto min_avg(std::vector<int> nums) -> float {
  std::ranges::sort(nums);
  auto const n = nums.size() / 2;
  return std::ranges::min(
    rv::zip_transform(std::plus{},
                      rv::take(nums, n),
                      rv::drop(nums, n) | rv::reverse)
  | rv::transform([](auto x) { return x / 2.0; }));
}
```

```cpp
namespace rv = std::ranges::views; // std::views

auto min_avg(std::vector<int> nums) -> float {
  std::ranges::sort(nums);
  auto const n = nums.size() / 2;
  return std::ranges::min(
    rv::zip_transform(_add_,
                      rv::take(nums, n),
                      rv::drop(nums, n) | rv::reverse)
  | rv::transform(div_(2.0)));
}
```

```cpp
namespace rv = std::ranges::views; // std::views

auto min_avg(std::vector<int> nums) -> float {
  std::ranges::sort(nums);
  auto const n = nums.size() / 2;
  return std::ranges::min(
    rv::zip_transform(_b1(div_(2.0), _add_),
                      rv::take(nums, n),
                      rv::drop(nums, n) | rv::reverse));
}
```

```cpp
auto min_avg(std::vector<int> nums) -> float {
  std::ranges::sort(nums);
  auto const n = nums.size() / 2;
  return nums
      |> rv::zip_transform(_b1(div_(2.0), _add_),
                           rv::take($, n),
                           rv::drop($, n) | rv::reverse)
      |> std::ranges::min($);
}
```

```cpp
auto minimumAverage(std::vector<int> &nums) -> double {
    std::ranges::sort(nums);
    return std::transform_reduce(
                nums.begin(),
                nums.begin() + nums.size() / 2,
                nums.rbegin(),
                std::numeric_limits<int>::max(),
                std::ranges::min,
                std::plus{}) / 2.0;
}
```

Live Coding

$B \rightarrow 1y11$

$B_1 \rightarrow 2y12$

$S \rightarrow 1y21$

$\Sigma \rightarrow 1y12$

$D \rightarrow 2y21$

$\Delta \rightarrow 2y21$

$\Psi \rightarrow 2y21$

B → 1y11

B_1 → 2y12

S → 1y21

Σ → 1y12

D → 2y21

Δ → 2y21

Ψ → 2y21

Φ → 1y121

$Φ_1$ → 2y222

W → 1y2

C → 2y2

B → 1y11

B_1 → 2y12

S → 1y21

Σ → 1y12

D → 2y21

Δ → 2y21

Ψ → 2y21

Φ → 1y121

$Φ_1$ → 2y222

W → 1y2
C → 2y2
B → 1y11
B_1 → 2y12

} 2 Train

S → 1y21
Σ → 1y12
D → 2y21
Δ → 2y21
Ψ → 2y21
Φ → 1y121
$Φ_1$ → 2y222

} 3 Train

In Conclusion

Kap

```
∧/2≠/2|
```

Dyalog APL

```
∧/2≠/2|⊢
```

Uiua

```
/↧≡/≠⊞2⊿2
```

BQN

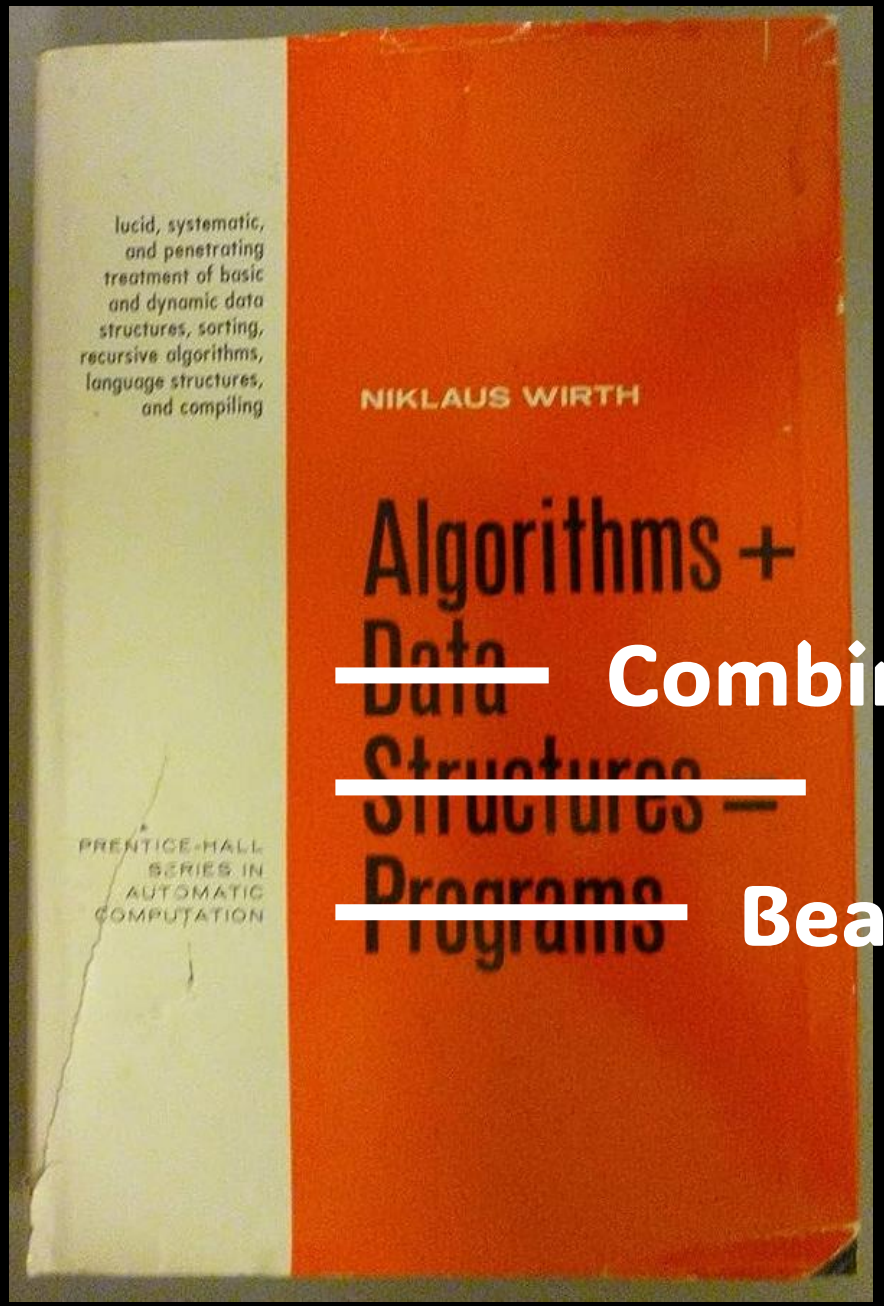
```
∧´·≠´˘2↕2|⊢
```

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*

Functions	✅	✅	✅	❌	❌	❌	❌	❌	❌
Operators	❌	❌	✅	✅	✅	✅	✅	✅	✅
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)))
```

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

code_report@mastodon.social

Questions?

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

Conor Hoekstra

code_report

codereport

code_report@mastodon.social