Combinatory Logic and Combinators in Array Languages

Conor Hoekstra

ARRAY 2022

June 13, 2022

Slide deck can be found at: github.com/codereport/Content

2014
2015
2016
2017
2018
2019
2020
2021
2022

Algorithms + Data Structures = Programs

Array Cast

MOODY'S ANALYTICS

NVIDIA

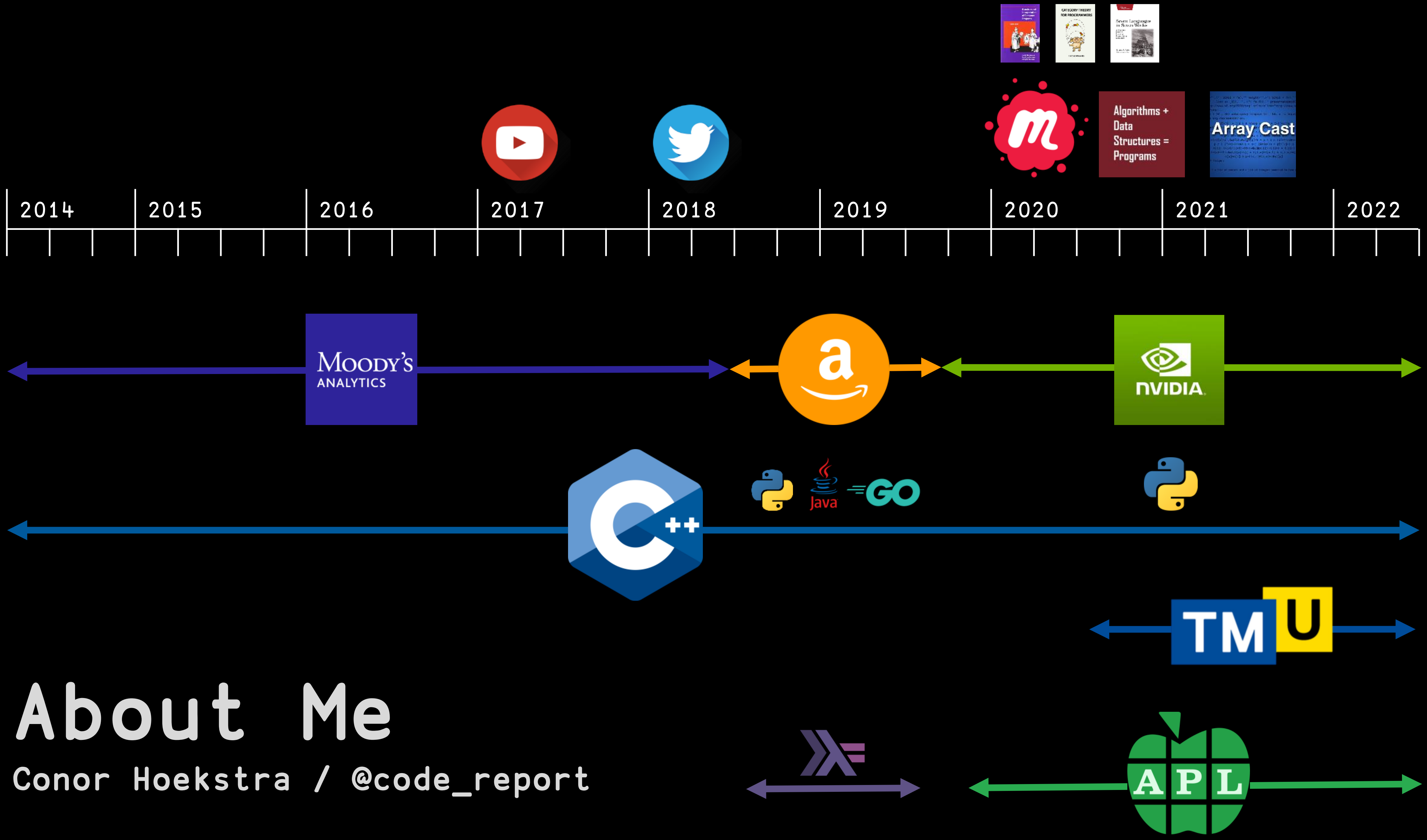
Java

GO

APL

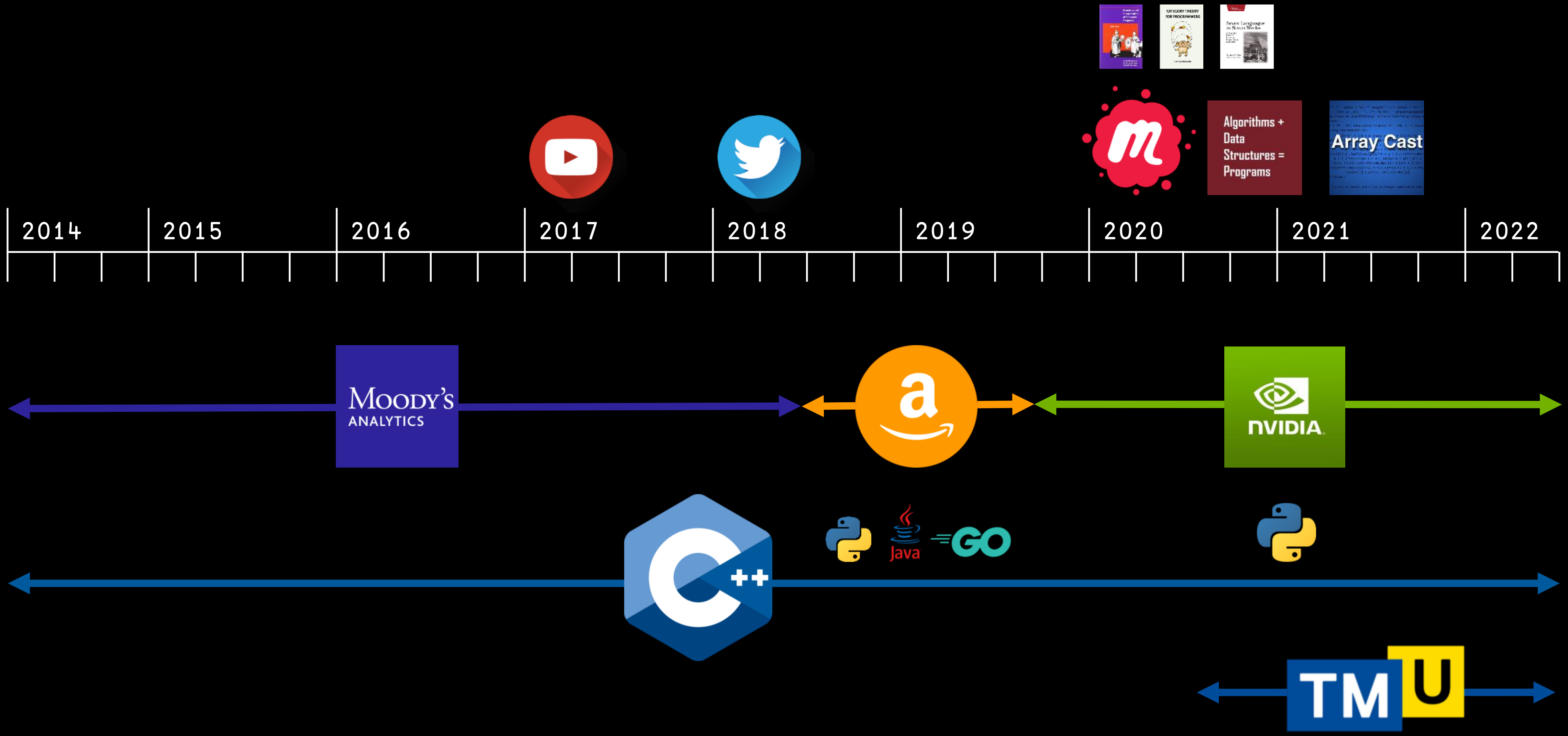
About Me

Conor Hoekstra / @code_report



About Me

Conor Hoekstra / @code_report



About Me

Conor Hoekstra / @code_report

1.
2. APL
3.
4.
5.

- Introduction
- A Brief Introduction to Array Languages
- A Brief History of Array Languages
- A Brief History of Combinatory Logic
- Combinator Specializations
- Evolution of Combinatory Logic in Array Languages
- Combinators in APL, BQN and J
- The Power of Combinators in Array Languages
- Examples
- Summary

- ❑ Introduction
- ❑ A Brief Introduction to Array Languages
- ❑ A Brief History of Array Languages
- ❑ A Brief History of Combinatory Logic
- ❑ Combinator Specializations
- ❑ Evolution of Combinatory Logic in Array Languages
- ❑ Combinators in APL, BQN and J
- ❑ The Power of Combinators in Array Languages
- ❑ Examples
- ❑ Summary

First 10 Odd Numbers

$\iota 10$

1 2 3 4 5 6 7 8 9 10

$$2\times\iota 10$$

2 4 6 8 10 12 14 16 18 20

```
¯1+2×⍳10
1 3 5 7 9 11 13 15 17 19
```

Multiplication Table

$\iota 10$

1 2 3 4 5 6 7 8 9 10

```
(⍳10)∘.×⍳10
```

1	2	3	4	5	6	7	8	9	10
2	4	6	8	10	12	14	16	18	20
3	6	9	12	15	18	21	24	27	30
4	8	12	16	20	24	28	32	36	40
5	10	15	20	25	30	35	40	45	50
6	12	18	24	30	36	42	48	54	60
7	14	21	28	35	42	49	56	63	70
8	16	24	32	40	48	56	64	72	80
9	18	27	36	45	54	63	72	81	90
10	20	30	40	50	60	70	80	90	100

```
∘.×⍨⍳10
```

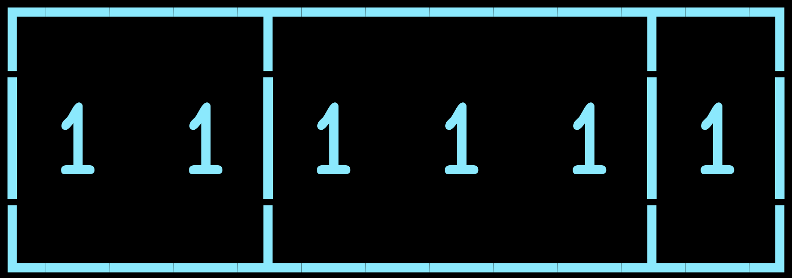
1	2	3	4	5	6	7	8	9	10
2	4	6	8	10	12	14	16	18	20
3	6	9	12	15	18	21	24	27	30
4	8	12	16	20	24	28	32	36	40
5	10	15	20	25	30	35	40	45	50
6	12	18	24	30	36	42	48	54	60
7	14	21	28	35	42	49	56	63	70
8	16	24	32	40	48	56	64	72	80
9	18	27	36	45	54	63	72	81	90
10	20	30	40	50	60	70	80	90	100

Maximum Consecutive Ones

```
vec ← 1 1 0 1 1 1 0 0 0 1
```

vec⊆vec

```
┌───┬─────┬─┐
│1 1│1 1 1│1│
└───┴─────┴─┘
```


1 1
1 1 1
1

```
≠¨⊆⍨vec
```

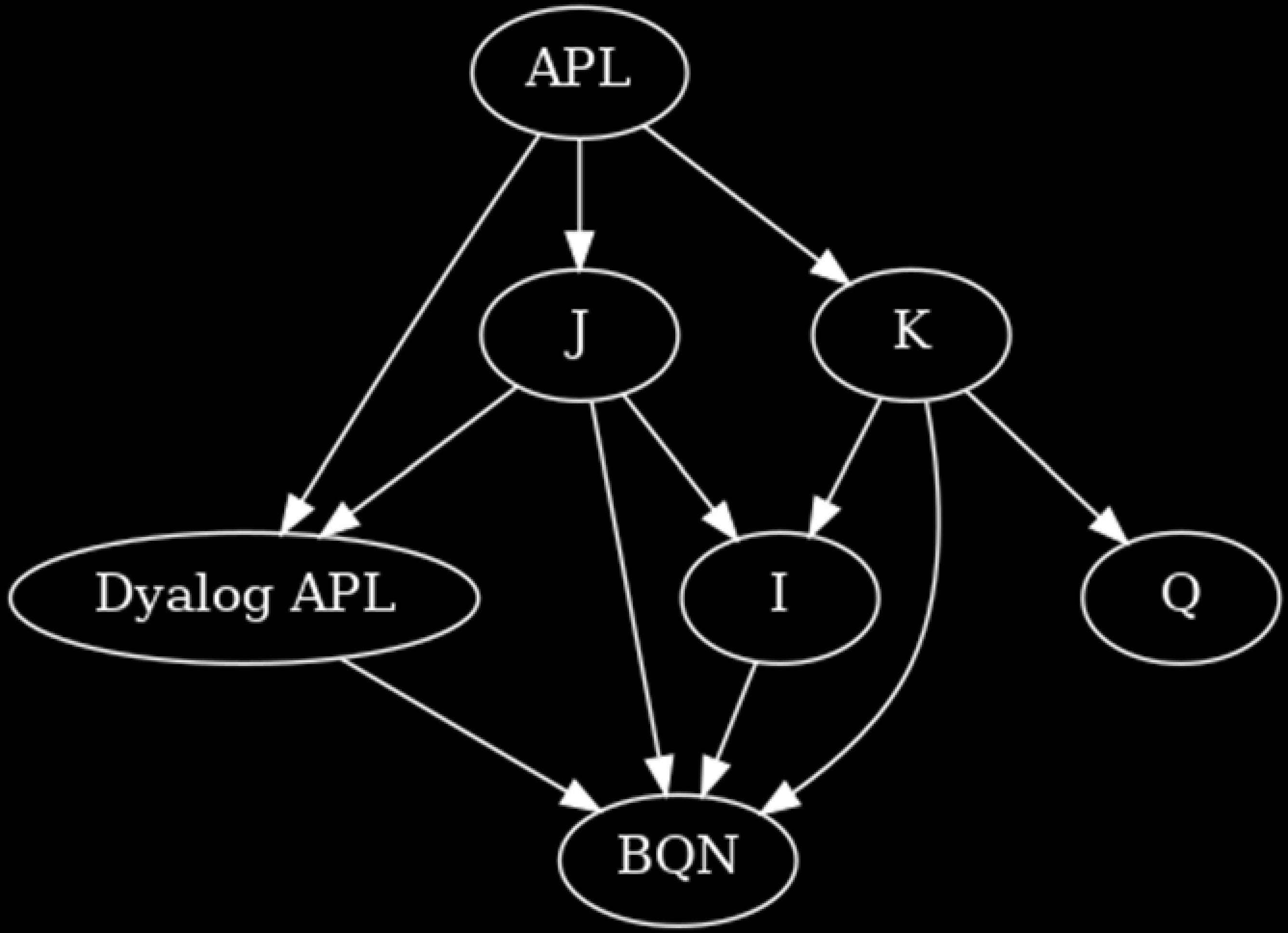
```
2 3 1
```

```
⌈/≢¨⊆⍨vec

3
```

- [ ] Introduction
- [x] A Brief Introduction to Array Languages
- [ ] A Brief History of Array Languages
- [ ] A Brief History of Combinatory Logic
- [ ] Combinator Specializations
- [ ] Evolution of Combinatory Logic in Array Languages
- [ ] Combinators in APL, BQN and J
- [ ] The Power of Combinators in Array Languages
- [ ] Examples
- [ ] Summary

- ☐ Introduction
- ☑ A Brief Introduction to Array Languages
- ☐ A Brief History of Array Languages
- ☐ A Brief History of Combinatory Logic
- ☐ Combinator Specializations
- ☐ Evolution of Combinatory Logic in Array Languages
- ☐ Combinators in APL, BQN and J
- ☐ The Power of Combinators in Array Languages
- ☐ Examples
- ☐ Summary


APL
J
K
Dyalog APL
I
Q
BQN

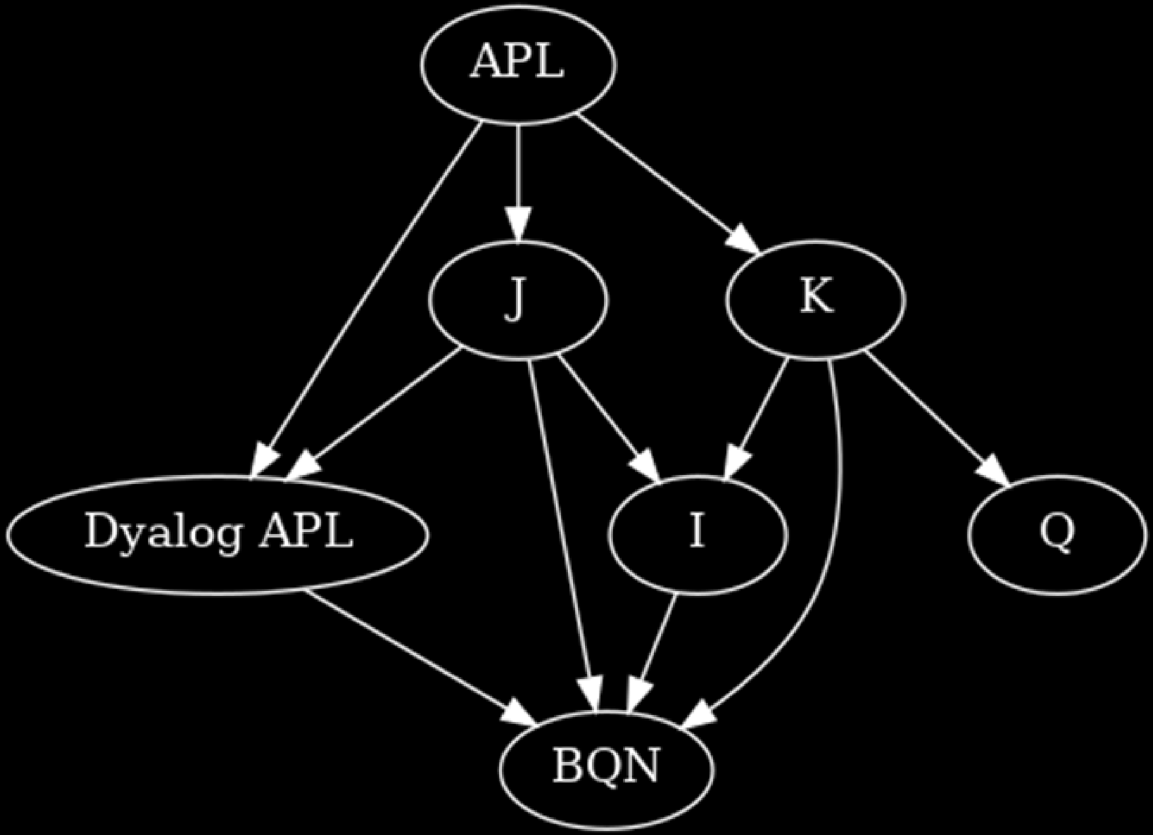


Table 1. Array languages timeline.

Language	Year
APL	1966
Dyalog APL 1.0	1983
J	1990
K	1994
Q	2003
I	2012
BQN	2020
Dyalog APL 18.0	2020

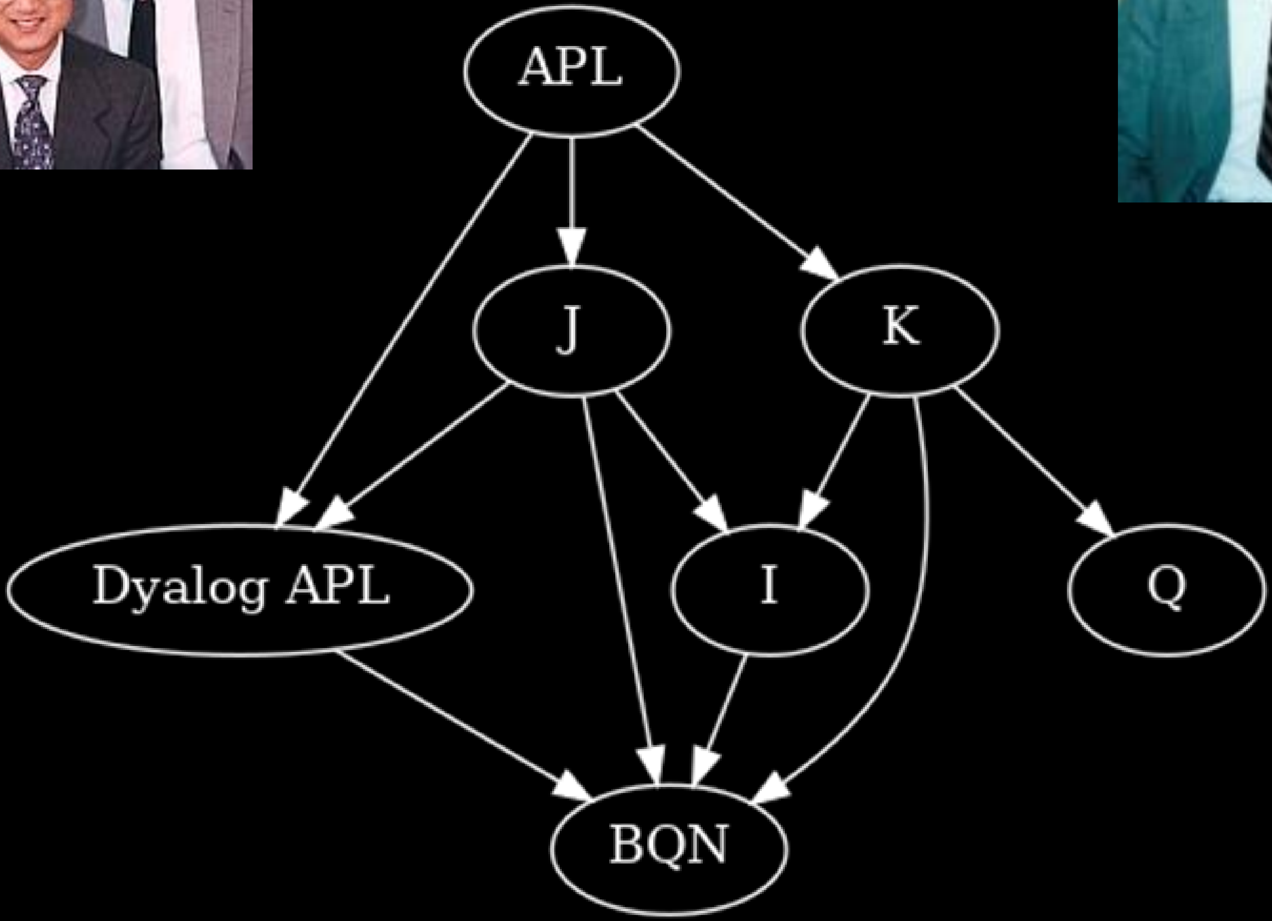


Table 1. Array languages timeline.

Language	**Year**
APL	1966
Dyalog APL 1.0	1983
J	1990
K	1994
Q	2003
I	2012
BQN	2020
Dyalog APL 18.0	2020

- [ ] Introduction
- [x] A Brief Introduction to Array Languages
- [x] A Brief History of Array Languages
- [ ] A Brief History of Combinatory Logic
- [ ] Combinator Specializations
- [ ] Evolution of Combinatory Logic in Array Languages
- [ ] Combinators in APL, BQN and J
- [ ] The Power of Combinators in Array Languages
- [ ] Examples
- [ ] Summary

- ❑ Introduction
- ☑ A Brief Introduction to Array Languages
- ☑ A Brief History of Array Languages
- ❑ A Brief History of Combinatory Logic
- ❑ Combinator Specializations
- ❑ Evolution of Combinatory Logic in Array Languages
- ❑ Combinators in APL, BQN and J
- ❑ The Power of Combinators in Array Languages
- ❑ Examples
- ❑ Summary

Moses Schönfinkel
1988 - 1942

Moses Schönfinkel
1988 - 1942

Haskell Curry
1900-1982

Year	Author(s)	Title	Introduced
1924	Schönfinkel	On the building blocks of mathematical logic	S K I B C
1929	Curry	An Analysis of Logical Substitution	W
1930	Curry	The Foundations of Combinatory Logic	B_n (B_1, B_2, …)
1931	Curry	The Universal Quantifier in Combinatory Logic	Ψ, Φ_n (Φ, Φ_1, …)
1958	Curry and Feys	Combinatory Logic: Volume I	

Table 2. The elementary combinators.

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

Table 3. Combinators and lambda expressions.

Combinator	Lambda Expression
I	λa.a
K	λab.a
W	λab.abb
C	λabc.acb
B	λabc.a(bc)
S	λabc.ac(bc)
D	λabcd.ab(cd)
B_1	λabcd.a(bcd)
Ψ	λabcd.a(bc)(bd)
Φ	λabcd.a(bd)(cd)
D_2	λabcde.a(bd)(ce)
E	λabcde.ab(cde)
Φ_1	λabcde.a(bde)(cde)
$\hat{E}$	λabcdefg.a(bde)(cfg)

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

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

- [ ] Introduction
- [x] A Brief Introduction to Array Languages
- [x] A Brief History of Array Languages
- [x] A Brief History of Combinatory Logic
- [ ] Combinator Specializations
- [ ] Evolution of Combinatory Logic in Array Languages
- [ ] Combinators in APL, BQN and J
- [ ] The Power of Combinators in Array Languages
- [ ] Examples
- [ ] Summary

- [ ] Introduction
- [x] A Brief Introduction to Array Languages
- [x] A Brief History of Array Languages
- [x] A Brief History of Combinatory Logic
- [ ] Combinator Specializations
- [ ] Evolution of Combinatory Logic in Array Languages
- [ ] Combinators in APL, BQN and J
- [ ] The Power of Combinators in Array Languages
- [ ] Examples
- [ ] Summary

Phrasal Forms

K. E. Iverson
Toronto

E. E. McDonnell
I. P. Sharp Associates
Palo Alto

NOTE: In this paper we use the linguistic terms *verb* and *pronoun* interchangeably with the mathematical terms *function* and *variable.*

INTRODUCTION

In standard APL [ISO88] certain forms are ungrammatical, and new definitions could be adopted for them without conflict. Such definitions we shall call *phrasal forms* [AHD76]. For example, if *b* and *c* are pronouns, the phrase *b c* is meaningless, and in APL2 [IBM85] the definition (⊂ *b*), (⊂ *c*), where ⊂ is the APL2 *enclose* function, is adopted for it.

VERB RANK

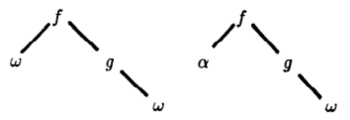
The notion of *verb rank*, first introduced by Iverson [Iv78], later elaborated by Keenan [Ke79], and further evolved by Whitney [Wh84], has been adopted by Iverson in his Dictionary [Iv87]. It refers to the rank of the subarrays of an argument which are the cells to which the verb aplies. For example, the cells that negate applies to are items, and items are ranlk zero objects, and thus we say the rank of negate is zero. Similarly, the cells to which reverse applies are lists, or rank ome objects, and thus we say the rank of reverse is one. Not only primitive verbs, but also derived and defined verbs have rank. The idea is a powerful one, producing great simplifications, and so we define the ranks of the new constructions we describe herein.

DEFINITIONS

In the following definitions, *f*, *g*, and *h* denote verbs and α and ω denote pronouns.

HOOK. A *hook* is denoted by *f g* and is defined formally by identities and informally by hook-shaped diagrams as follows:

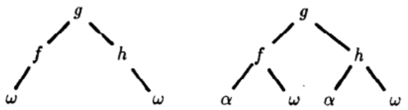
$$(fg)\omega \leftrightarrow \omega f g \omega; \qquad \alpha(fg)\omega \leftrightarrow \alpha f g \omega$$



The rank of the hook *f g* is the maximum of the ranks of *f* and *g*. Note that the verb is used monadically.

FORK. A *fork* is denoted by *f g h* and is defined formally by identities and informally by forking diagrams as follows:

$$(fgh)\omega \leftrightarrow (f\omega)g(h\omega); \qquad \alpha(fgh)\omega \leftrightarrow (\alpha f\omega)g(\alpha h\omega)$$



The rank of the fork *f g h* is the maximum of the ranks of *f* and *g*. Note that the central verb is used dyadically or monadically according to whether the fork is applied to two arguments or one. Parenthesis are required around hook and fork forms only to avoid ambiguity.

DISCUSSION OF HOOK AND FORK

HOOK. In combinatory logic one of the most useful primitive combinators is designated by **S** [Sch24]. Curry defines **S**fgx in prefix notation to be $fx(gx)$[CuFeCr74]. In common mathematical infix notation this would be given by $(x)f(g(x))$, which one can write in APL as $xfgx$, and this is the hook form $(fg)x$. The combinatory logician appreciates this form because of its great expressiveness: it can be shown that **S**, along with **K**, the *constancy* combinator, suffice to define all other combinators of interest [Ro50]. (The constancy combinator **K** is defined in infix notation so that *c***K***x* has the value *c* for all *x*.) Users of APL will appreciate the hook for the same reasons.

For example, **+÷** adds the reciprocal of the right argument to the left argument, a form used in describing continued fractions. Thus (+÷)\3 7 16 ¯294 gives the first four convergents to pi, which, to nine decimals, are 3, 3.1412857143, 3.14159292, and 3.141592654. Further, =⌊ is a proposition that tests whether its argument is an integer, the number of primes less than positive integer ω is approximately (÷⍟)ω; and to decompose a number ω into numerator and denominator, one can write (÷∨/)ω, 1 (where ∨ is the *greatest common divisor).*

FORK. The forks $f + h$ and $f \times h$ and $f \div h$ provide formal treatment of the identical but informal phrases used in mathematics [e.g. Ef89] for the sum and product and quotient,

d further evolved by
erson in his Dictionary
arrays of an argument
ies. For example, the
d items are ranlk zero
gate is zero. Similarly,
, or rank ome objects,
ne. Not only primitive
s have rank. The idea
plifications, and so we
ns we describe herein.

enote verbs and α and

is defined formally by
diagrams as follows:

$\omega \leftrightarrow \alpha f g \omega$

and g. Note that the central verb is used dyadically or monadically according to whether the fork is applied to two arguments or one. Parenthesis are required around hook and fork forms only to avoid ambiguity.

DISCUSSION OF HOOK AND FORK

HOOK. In combinatory logic one of the most useful primitive combinators is designated by **S** [Sch24]. Curry defines **S**fgx in prefix notation to be $fx(gx)$[CuFeCr74]. In common mathematical infix notation this would be given by $(x)f(g(x))$, which one can write in APL as $xfgx$, and this is the hook form $(fg)x$. The combinatory logician appreciates this form because of its great expressiveness: it can be shown that **S**, along with **K**, the *constancy* combinator, suffice to define all other combinators of interest [Ro50]. (The constancy combinator **K** is defined in infix notation so that c**K**x has the value c for all x.) Users of APL will appreciate the hook for the same reasons.

For example, +÷ adds the reciprocal of the right argument to the left argument, a form used in describing continued fractions. Thus (+÷)\3 7 16 ¯294 gives the first four convergents to pi, which, to nine decimals, are 3, 3.1412857143, Curry [Cu31] defines a *formalizing combinator*, Φ, in prefix notation, such that $\Phi fghx$ means $f(gx)(hx)$. In common mathematical infix notation this would be designated by $(g(x))f(h(x))$. An example of this form is $\Phi{+}sin^2cos^2\theta$, meaning $sin^2\theta + cos^2\theta$. The fork $(f\ g\ h)\omega$ has the same meaning, namely $(f\omega)g(h\omega)$. Curry named this the *formalizing combinator* because of its role in defining formal implication in terms of ordinary implication.

Iverson and Whitney have made several earlier suggestions of ways to achieve what the fork form provides: the *scalar operators* of [Iv78], [Iv79a], [Iv 79b], the *til* operator of [Iv82], the *union* and *intersection* conjunctions of [Iv87], and the *yoke* adverb of [Iv88]. Benkard [Bk87] has also suggested a way to achieve the meaning of this form, in his proposal for $\uparrow g/(f\ h)\alpha\ \omega$, using the notion of *function pair* (↑ is APL2's *first* function). The present proposal has significant advantages over these earlier ones.

Table 4. 2 and 3-trains in APL, 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	BQN	B and B_1	Φ and Φ_1

Table 5. History of combinators in Dyalog APL.

Year	Version	Combinator	Spelling
1983	1.0	B, D, C	∘⍨
2003	10.0	W	⍨
2013	13.0	K	⊢
2014	14.0	B, B_1, Φ, Φ_1	trains
2020	18.0	B, B_1, Ψ, K	⍤⍥⍨

- [ ] Introduction
- [x] A Brief Introduction to Array Languages
- [x] A Brief History of Array Languages
- [x] A Brief History of Combinatory Logic
- [ ] Combinator Specializations
- [x] Evolution of Combinatory Logic in Array Languages
- [ ] Combinators in APL, BQN and J
- [ ] The Power of Combinators in Array Languages
- [ ] Examples
- [ ] Summary

- [ ] Introduction
- [x] A Brief Introduction to Array Languages
- [x] A Brief History of Array Languages
- [x] A Brief History of Combinatory Logic
- [ ] Combinator Specializations
- [x] Evolution of Combinatory Logic in Array Languages
- [ ] Combinators in APL, BQN and J
- [ ] The Power of Combinators in Array Languages
- [ ] Examples
- [ ] Summary

Table 14. Ψ combinator.

Language	Name	Symbol
APL	Over	⍥
BQN	Over	○
J	Appose	&:

Table 14. Ψ combinator.

Language	Name	Symbol
APL	Over	⍥
BQN	Over	○
J	Appose	&:

	APL		J
I	⊣⊢	⊣⊢	][
K	⊢	⊢	]
KI	⊣	⊣	[
S		⟜	2 Train
B	∘⍤⍥ 2T	∘○ 2T	@: &:
C	⍨	˜	~
W	⍨	˜	~
B_1	⍤ 2T	∘ 2T	@:
D	∘	⟜	2 Train
Ψ	⍥	○	&:
Φ	3 Train	3 Train	3 Train
$Φ_1$	3 Train	3 Train	3 Train
D_2		⊸⟜	

	APL		J
I	⊣⊢	⊣⊢	][
K	⊢	⊢	]
KI	⊣	⊣	[
S		⟜	2 Train
B	∘⍤⍥ 2T	∘○ 2T	@: &:
C	⍨	˜	~
W	⍨	˜	~
B_1	⍤ 2T	∘ 2T	@:
D	∘	⟜	2 Train
Ψ	⍥	○	&:
Φ	3 Train	3 Train	3 Train
$Φ_1$	3 Train	3 Train	3 Train
D_2		⊸⟜	

	APL	BQN	J
I	⊣⊢	⊣⊢	][
K	⊢	⊢	]
KI	⊣	⊣	[
S		⟜	2 Train
B	∘⍤⍥ 2T	∘○ 2T	@: &:
C	⍨	˜	~
W	⍨	˜	~
B_1	⍥ 2T	∘ 2T	@:
D	∘	⟜	2 Train
Ψ	⍥	○	&:
Φ	3 Train	3 Train	3 Train
Φ_1	3 Train	3 Train	3 Train
D_2		⊸⟜	

	APL		J
I	⊣⊢	⊣⊢	][
K	⊢	⊢	]
KI	⊣	⊣	[
S		⟜	2 Train
B	∘⍤⍥ 2T	∘○ 2T	@: &:
C	⍨	˜	~
W	⍨	˜	~
B_1	⍤ 2T	∘ 2T	@:
D	∘	⟜	2 Train
Ψ	⍥	○	&:
Φ	3 Train	3 Train	3 Train
$Φ_1$	3 Train	3 Train	3 Train
D_2		⊸⟜	
Σ		⊸	
Δ		⊸	

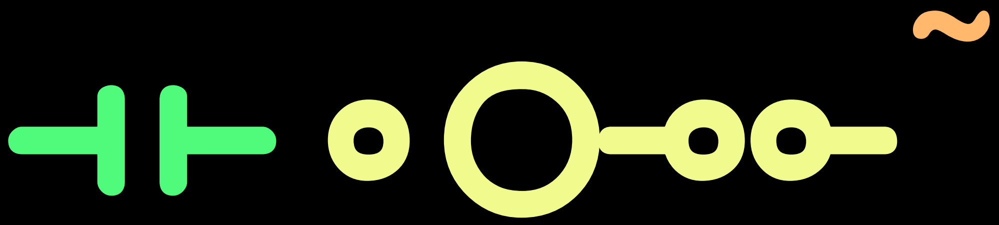
- ❑ Introduction
- ☑ A Brief Introduction to Array Languages
- ☑ A Brief History of Array Languages
- ☑ A Brief History of Combinatory Logic
- ❑ Combinator Specializations
- ☑ Evolution of Combinatory Logic in Array Languages
- ☑ Combinators in APL, BQN and J
- ❑ The Power of Combinators in Array Languages
- ❑ Examples
- ❑ Summary

∘ ⍤ ⍥ ⍨

\+ - × ÷

⊣⊢∘⍤⍥⍨

+−×÷



+ − × ÷

APL ⊣⊢∘⍤⍥⍨

⊣⊢∘○⊸⟜˜

Thank You!

Conor Hoekstra
choekstra@ryerson.ca
conorhoekstra@gmail.com
@code_report