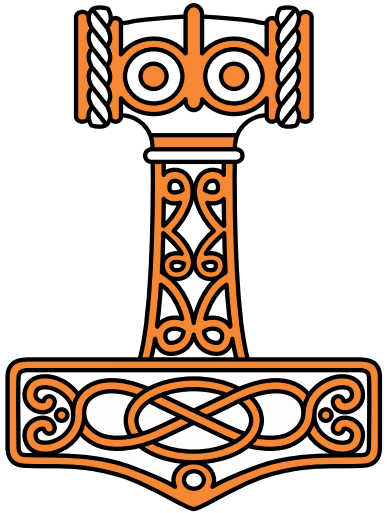
DYALOG

Olhão 2022

Filling the Core Language Gaps



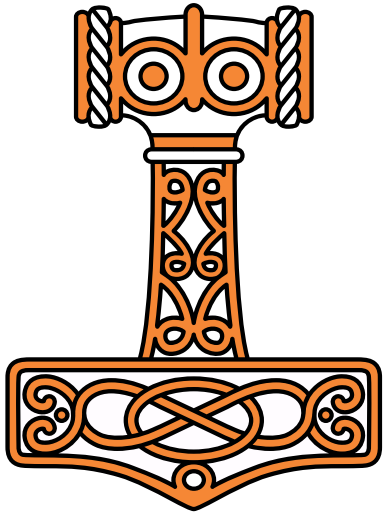
Adám Brudzewsky

DRAFT PROPOSAL

DXLOG

Chiao 2022

Filling the Core Language Gaps



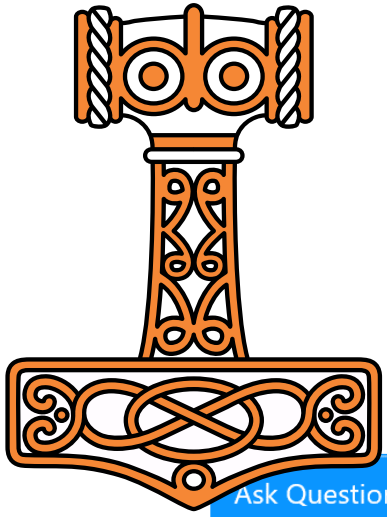
Adám Brudzewsky

DYALOG

Chiao 2022

DRAFT PROPOSAL

Filling the Core Language Gaps



Indexing with nested vectors in APL

Asked 2 years, 3 months ago Modified 2 years, 2 months ago Viewed 132 times

Ask Question

3

I have a vector of vectors that contain some indices, and a character vector which I want to use them on.

```
A←(1 2 3)(3 2 1)
B←'ABC'
```

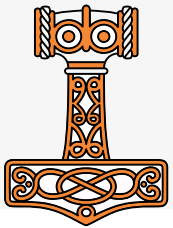
3

I have a vector of vectors that contain some

```
A←(1 2 3)(3 2 1)
B←'ABC'
```

I have tried:

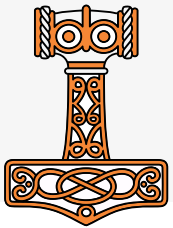
```
      B[A]
RANK ERROR
      B[A]
        ∧
```

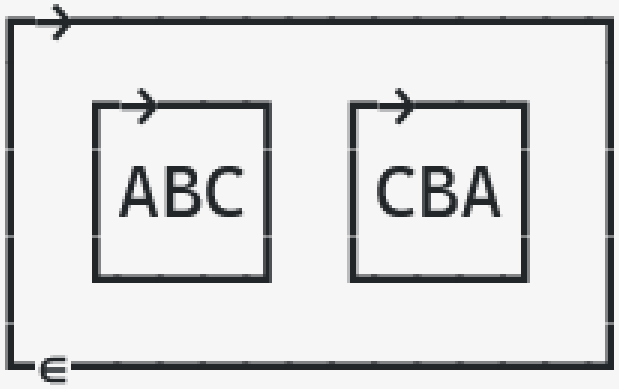
```
      A□B
LENGTH ERROR
      A□B
        ^
```

and

```
      A□B
LENGTH ERROR
      A□¨B
        ^
```



I would like

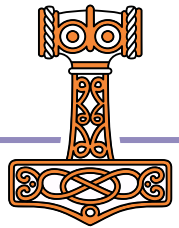


to be returned, but if i need to find another way,

indexing apl

Indexing with Nested Vectors

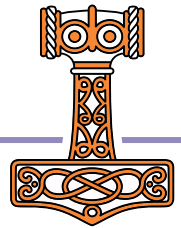
3 Answers:



Indexing with Nested Vectors

3 Answers:

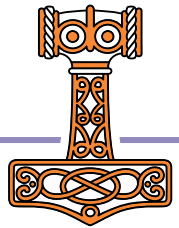
- `(⊂¨A)[]¨⊂B`



Indexing with Nested Vectors

3 Answers:

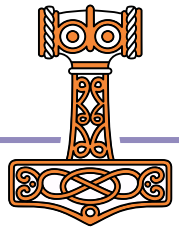
- `(⊂¨A)[]¨⊂B`
- `{B[⍵]}⍤⍤⍤⍤A`



Indexing with Nested Vectors

3 Answers:

- `(⊂¨A)[]¨⊂B`
- `{B[⍵]}⍤⍤⍤⍤A`
- *Don't do that!*



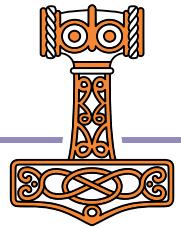
Indexing with Nested Vectors

3 Answers:

- `(⊂¨A)[]¨⊂B`
- `{B[⍵]}⌸A`
- *Don't do that!*

Possibilities:

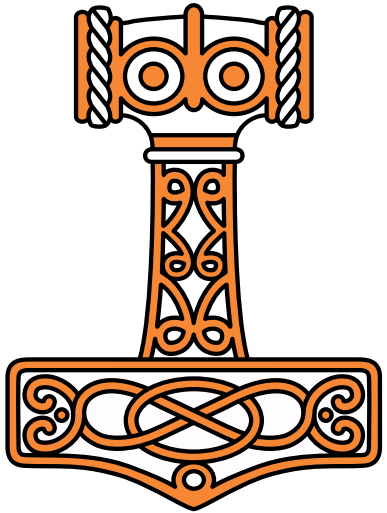
- `A[]⌸⊂⊂B`
- `[]∘B¨⊂¨A`
- `A(⊂⍤⊣[]⊢)¨⊂B`



DYALOG

Olhão 2022

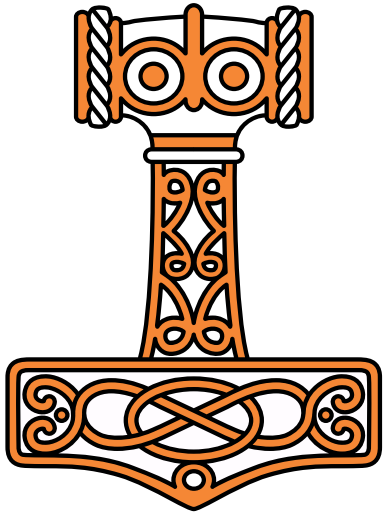
Filling the Core Language Gaps



DYALOG

Olhão 2022

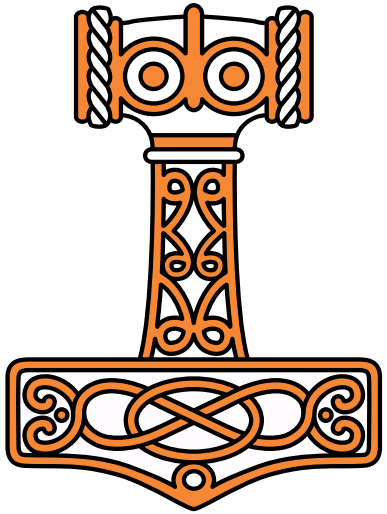
Core Language



DYALOG

Olhão 2022

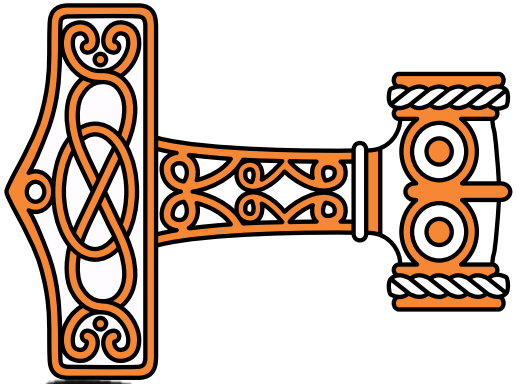
Core Language

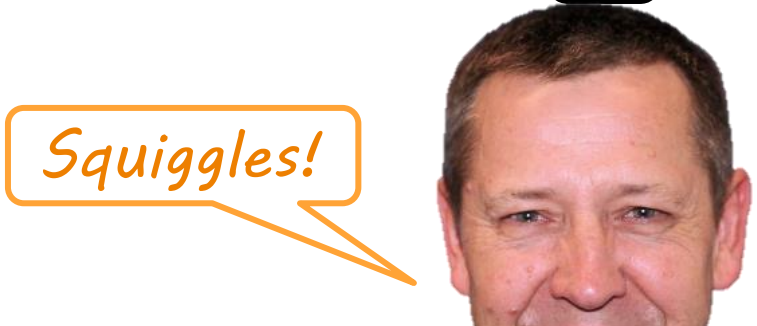


DYALOG

Olhão 2022

Core Language





Core Language

Core Language

Data Transformation

Core Language

Data Transformation

Function Application

Core Language

Data Transformation

Function Application

Function Composition

Core Language

Data Transformation $X \times Y$ ϕY $X \square Y$

Function Application

Function Composition

Core Language

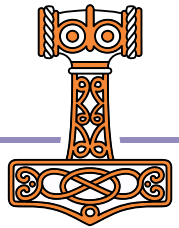
Data Transformation $X \times Y$ ϕY $X \square Y$

Function Application $f \not$ $f \overset{\cdot}{*} g$ $f \overset{\cdot}{\circ} k$

Function Composition

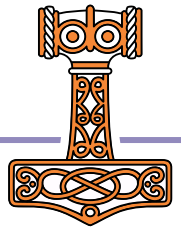
Core Language

Data Transformation	X × Y	ϕ Y	X [] Y
Function Application	f ⁄	f ⁑ g	f ⊙ k
Function Composition	f ⊙ g	f ⊚ g	f ∘ g



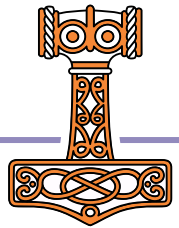
Core Language

Data Transformation	X × Y	ϕ Y	X [] Y
Function Application	f ≠	f ⁑ g	f ⊙ k
Function Composition	f ⊙ g	f ⊚ g	f ∘ g



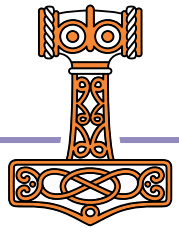
Data Transformation

Indexing



Data Transformation

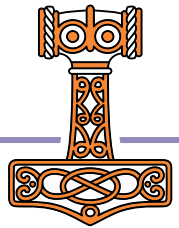
Simple Indexing



Data Transformation

Simple Indexing

Choose Indexing

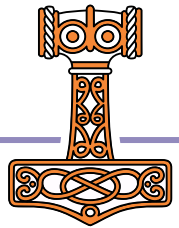


Data Transformation

Simple Indexing

Choose Indexing

Reach Indexing

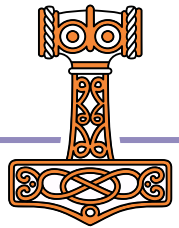


Data Transformation

- Simple Indexing

  Choose Indexing

  Reach Indexing



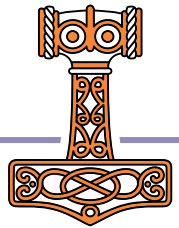
Data Transformation

- Simple Indexing

  Choose Indexing

  Reach Indexing

```
t←3 8⍴⎕A
```



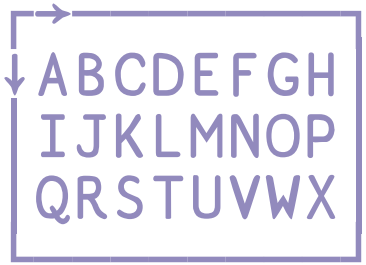
Data Transformation

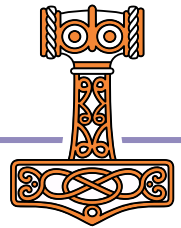
- Simple Indexing

  Choose Indexing

  Reach Indexing

```
t←3 8⍴⎕A
```





Data Transformation

- Simple Indexing

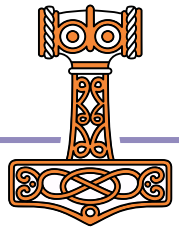
  Choose Indexing

  Reach Indexing

```
t←3 8⍴⎕A
```

```
t[2;4]
```

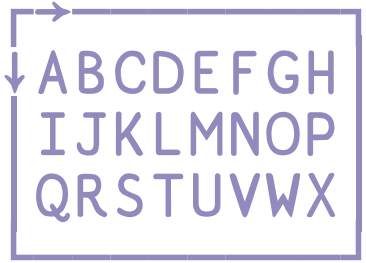
```
→
↓ABCDEFGH
 IJKLMNOP
 QRSTUVWX
```



Data Transformation

- Simple Indexing
  Choose Indexing
  Reach Indexing

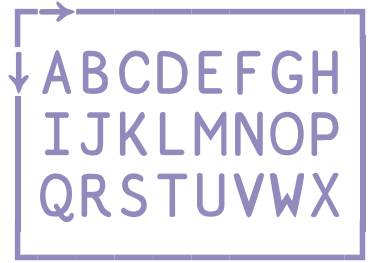
```
t←3 8⍴⎕A
t[2;4]
L
```

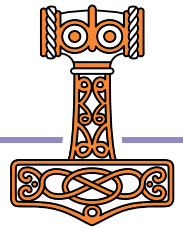


Data Transformation

- Simple Indexing
  Choose Indexing
  Reach Indexing

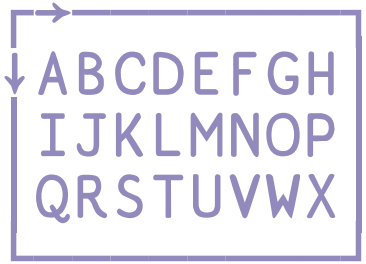
```
t←3 8⍴⎕A
ABCDEFGH
IJKLMNOP
QRSTUVWX

t[2;4]
L

2 4⌷t
L
```





Data Transformation

- Simple Indexing
  Choose Indexing
  Reach Indexing

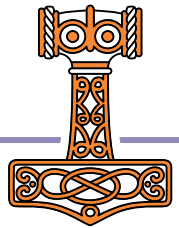



```
      t←3 8⍴⎕A
      t[2 1;4 1 7]
LIO
DAG

      (2 1)(4 1 7)⌷t
LIO
DAG
```

Data Transformation

- Simple Indexing
  Choose Indexing
  Reach Indexing

```
p←8⍴⍳A
```



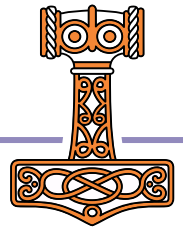
Data Transformation

- Simple Indexing

  Choose Indexing

  Reach Indexing

```
p←8⍴p⌷A
```



Data Transformation

- Simple Indexing

  Choose Indexing

  Reach Indexing


```
p←8⍴p⍳'A'
```

```
p[2]
```

Data Transformation

- Simple Indexing
  Choose Indexing
  Reach Indexing


```
p←8⍴p⎕A
p[2]
B
```

Data Transformation

- Simple Indexing
  Choose Indexing
  Reach Indexing


```
      p←8ρ⎕A
      p[2]
B
      2⌷p
B
```

Data Transformation

- Simple Indexing
  Choose Indexing
  Reach Indexing


```
      p←8⍴p⍞A

      p[2 1 7]
BAG
```

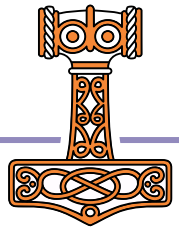
Data Transformation

- Simple Indexing
  Choose Indexing
  Reach Indexing


```
      p←8⍴p⌷A
      p[2 1 7]
BAG
      2 1 7?p
BAG
```

Data Transformation

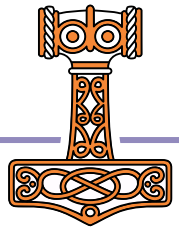
- Simple Indexing
- Choose Indexing
- Reach Indexing



Data Transformation

- Simple Indexing
- Choose Indexing
- Reach Indexing

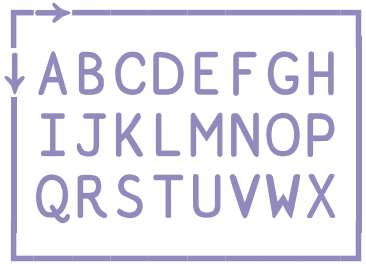
```
t←3 8⍴⎕A
```

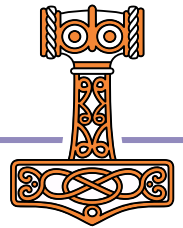


Data Transformation

- Simple Indexing
- Choose Indexing
- Reach Indexing

```
t←3 8⍴⎕A
```



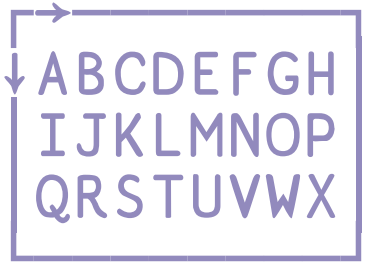


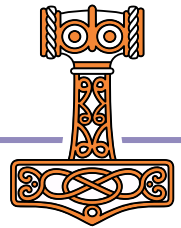
Data Transformation

- Simple Indexing
- Choose Indexing
- Reach Indexing

```
t←3 8⍴⎕A
```

```
t[⊂1 8]
```

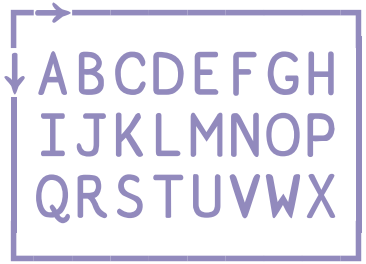




Data Transformation

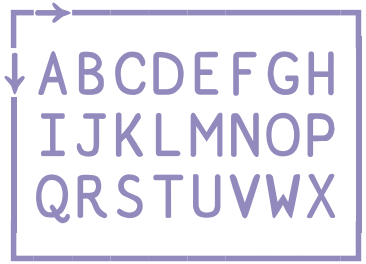
- Simple Indexing
- Choose Indexing
- Reach Indexing

```
t←3 8⍴⎕A

t[⊂1 8]

H
```



Data Transformation

- Simple Indexing
- Choose Indexing
- Reach Indexing

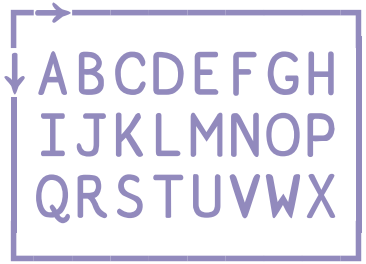


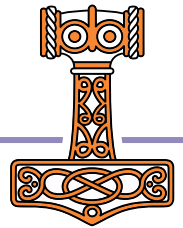

```
t←3 8⍴⎕A

t[⊂1 8]
H

1 8⌷t
H
```

Data Transformation

- Simple Indexing
- Choose Indexing
- Reach Indexing

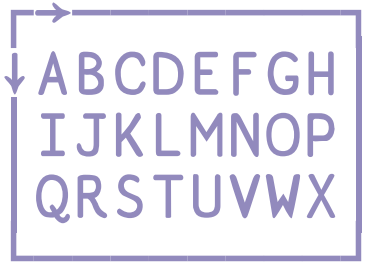
```
t←3 8⍴⎕A

t[(1 8)(2 7)]

HO
```





Data Transformation

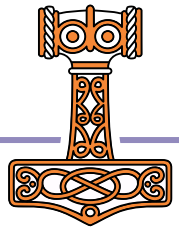
- Simple Indexing
- Choose Indexing
- Reach Indexing




```
      t←3 8⍴⎕A

      t[(1 8)(2 7)]
HO

      (1 8)(2 7)⍰t
HO
```

Data Transformation

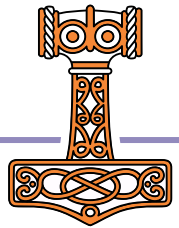
- Simple Indexing
- Choose Indexing
- Reach Indexing



Data Transformation

- Simple Indexing
- Choose Indexing
- Reach Indexing

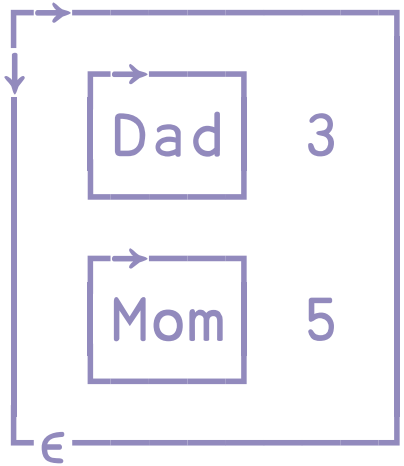
```
s←'Dad' 'Mom',⍪3 5
```



Data Transformation

- Simple Indexing
- Choose Indexing
- Reach Indexing

```
s←'Dad' 'Mom',⍪3 5
```

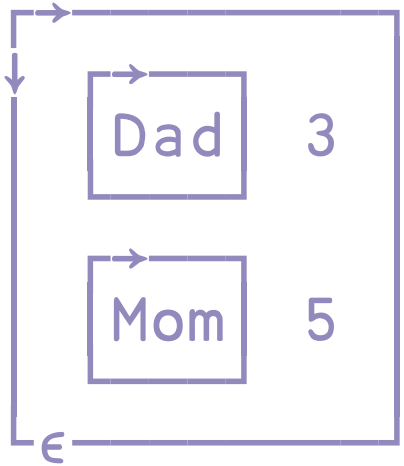


Data Transformation

- Simple Indexing
- Choose Indexing
- Reach Indexing

```
s←'Dad' 'Mom',⍪3 5
```

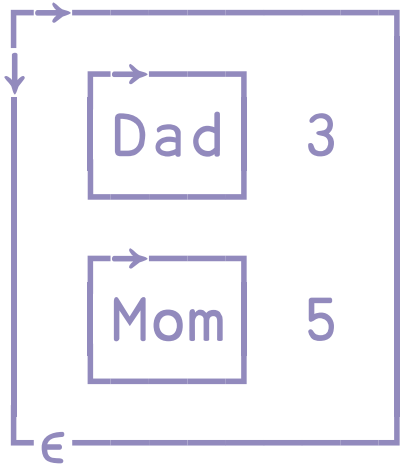
```
s[⊂(2 1)3]
```



Data Transformation

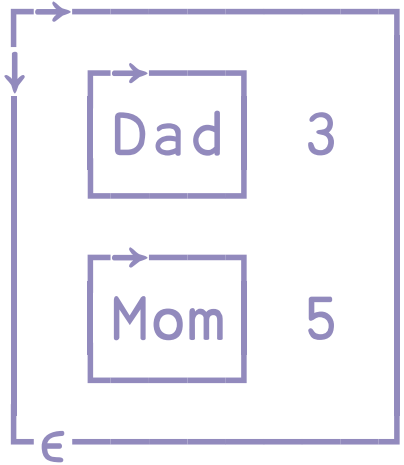
- Simple Indexing
- Choose Indexing
- Reach Indexing

```
s←'Dad' 'Mom',⍪3 5
s[⊂(2 1)3]
m
```



Data Transformation

- Simple Indexing
- Choose Indexing
- Reach Indexing

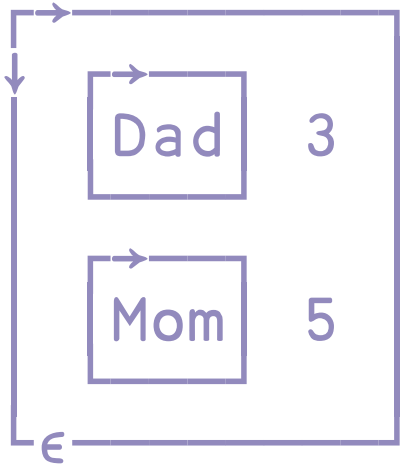



```
s←'Dad' 'Mom',⍪3 5
s[⊂(2 1)3]
m
(2 1)3⊃s
m
```

Data Transformation

- Simple Indexing
- Choose Indexing
- Reach Indexing

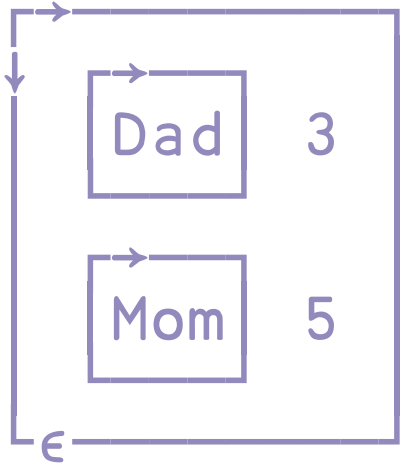
```
s←'Dad' 'Mom',⍪3 5
s[⊂(2 1)3]
m
```

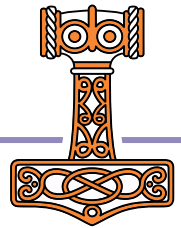


Data Transformation

- Simple Indexing
- Choose Indexing
- Reach Indexing

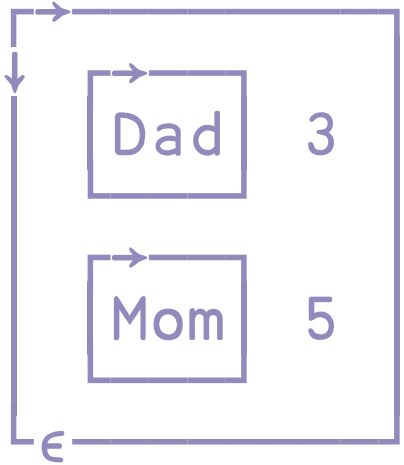
```
s←'Dad' 'Mom',⍪3 5

s[((2 1)3)((1 1)2)]

ma
```





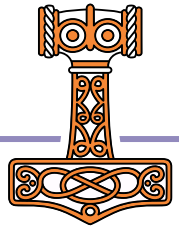
Data Transformation

- Simple Indexing
- Choose Indexing
- Reach Indexing




```
s←'Dad' 'Mom',⍪3 5

s[((2 1)3)((1 1)2)]
ma

((2 1)3)((1 1)2)⍰s
ma
```

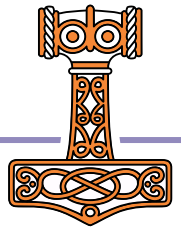
Data Transformation



Data Transformation

Select

$$X \supseteq Y$$



Indexing with Nested Vectors

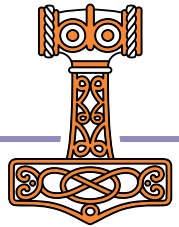
3 Answers:

- `(⊂¨A)[]¨⊂B`
- `{B[⍵]}⍤⍤⍤⍤A`
- *Don't do that!*

Possibilities:

- `A[]⍤⍤⍤⍤⊂⊂B`
- `[]∘B¨⊂¨A`
- `A(⊂⍤⊣[]⊢)¨⊂B`

With Select `X⊇Y`:



Indexing with Nested Vectors

3 Answers:

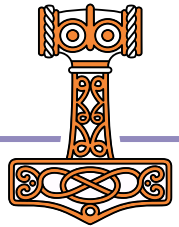
- `(⊂¨A)[]¨⊂B`
- `{B[⍵]}⌸A`
- *Don't do that!*

Possibilities:

- `A[]⌸⊂⊂B`
- `[]∘B¨⊂¨A`
- `A(⊂⍤⊣[]⊢)¨⊂B`

With Select `X⊇Y`:

- `A⊇¨⊂B`



Indexing with Nested Vectors

3 Answers:

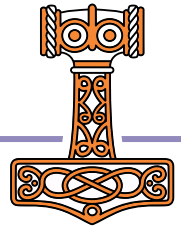
- `(⊂¨A)[]¨⊂B`
- `{B[⍵]}⌸A`
- *Don't do that!*

Possibilities:

- `A[]⌸⊂⊂B`
- `[]∘B¨⊂¨A`
- `A(⊂⍤⊣[]⊢)¨⊂B`

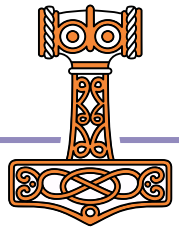
With Select `X⊇Y`:

- `A⊇¨⊂B`
- `⊇∘B¨A`



Data Transformation

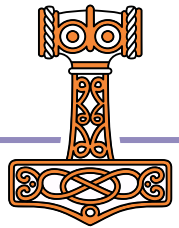
`X[]Y` Index



Data Transformation

X⌷Y Index

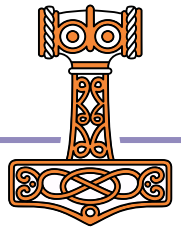
- `Sort ← (⊂⍤⍋⌷⊢)`



Data Transformation

X⌷Y Index

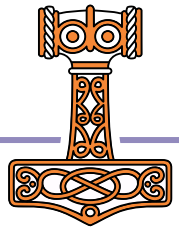
- `Sort ← (⊂⍤⍋⌷⊢)`
- `Sorts ← ⌷⍨∘⊂∘⍋⍨` `⍝ "sort Y by X"`



Data Transformation

X⌷Y Index

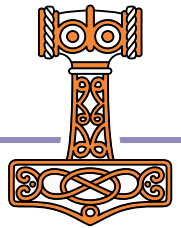
- `Sort ← (⊂⍤⍋⌷⊢)`
- `Sorts ← ⌷⍨∘⊂∘⍋⍨` `⍝ "sort Y by X"`
- `Shuffle ← (⊂⍤?⍨∘≢⌷⊢)`



Data Transformation

X⌷Y Index

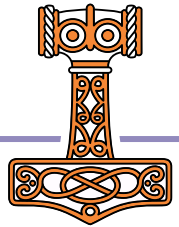
- `Sort  ← (⊂⍤⍋⌷⊢)`
- `Sorts ← ⌷⍨∘⊂∘⍋⍨  ⍝ "sort Y by X"`
- `Shuffle ← (⊂⍤?⍨∘≢⌷⊢)`
- `Grade ← ((⊂bounds∘⍳)⌷grades⍨)`



Data Transformation

X⊒Y Select

- Sort ← (⊂⍤⍋⌷⊢)
- Sorts ← ⌷⍨∘⊂∘⍋⍨ ⍝ "sort Y by X"
- Shuffle ← (⊂⍤?⍨∘≢⌷⊢)
- Grade ← ((⊂bounds∘⍳)⌷grades⍨)

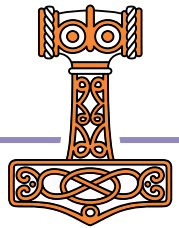


Data Transformation

X⊒Y Select/Permute

- Sort ← (⊂⍤⍋⌷⊢)
- Sorts ← ⌷⍨∘⊂∘⍋⍨ ⍝ "sort Y by X"
- Shuffle ← (⊂⍤?⍨∘≢⌷⊢)

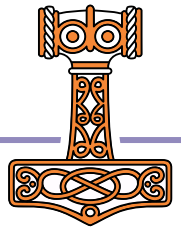
  Grade ← ((⊂bounds∘⍳)⌷grades⍨)



Data Transformation

X⊒Y Select/Permute

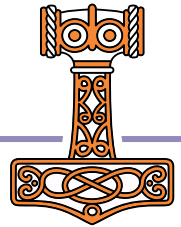
- `Sort  ← (  ⍋⊒⊢)`
- `Sorts ← ⊒⍨  ∘⍋⍨    ⍝ "sort Y by X"`
- `Shuffle ← (  ?⍨∘≠⊒⊢)`
- `Grade  ← (  bounds∘⍳ ⊒grades⍨)`



Data Transformation

X⊒Y Select/Permute

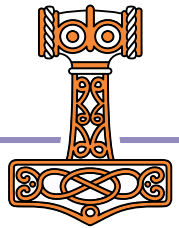
- Sort ← (⍋⊒⊢)
- Sorts ← ⊒⍨∘⍋⍨ ⍝ "sort Y by X"
- Shuffle ← (?⍨∘≠⊒⊢)
- Grade ← (bounds∘⍳⊒grades⍨)



Indexing with Nested Vectors

I have

```
A←(1 2 3)(3 2 1) ⋄ B←'ABC'
```

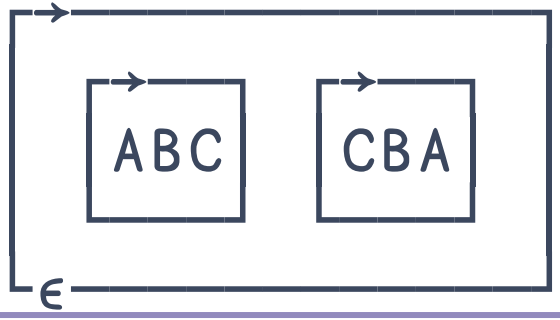


Indexing with Nested Vectors

I have

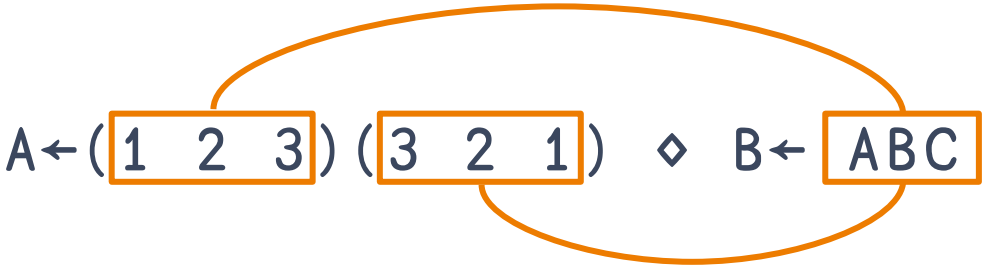
```
A←(1 2 3)(3 2 1) ⋄ B←'ABC'
```

I would like

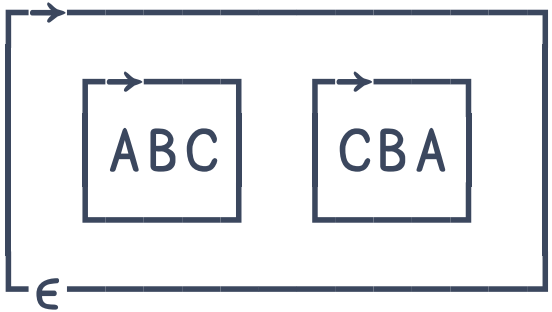


Indexing with Nested Vectors

I have

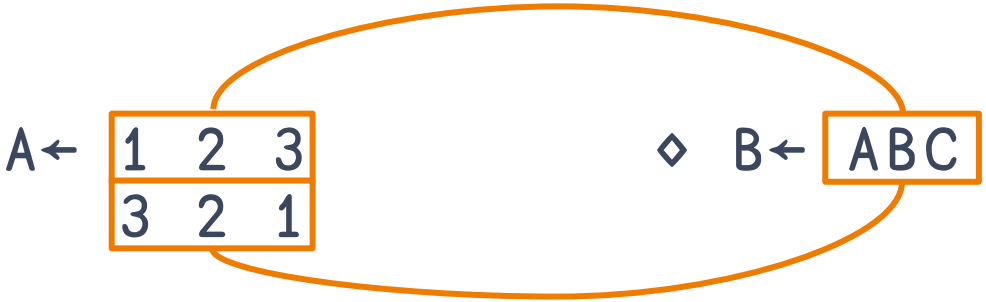


I would like

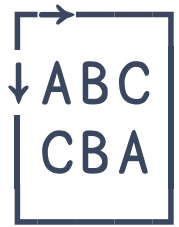


Indexing with Nested Vectors

I have

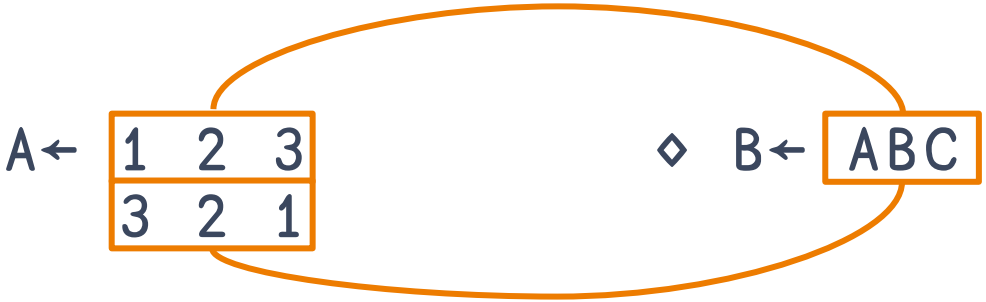


I would like

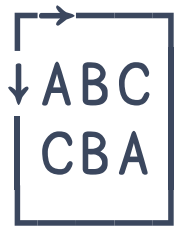


Indexing with Nested Vectors

I have



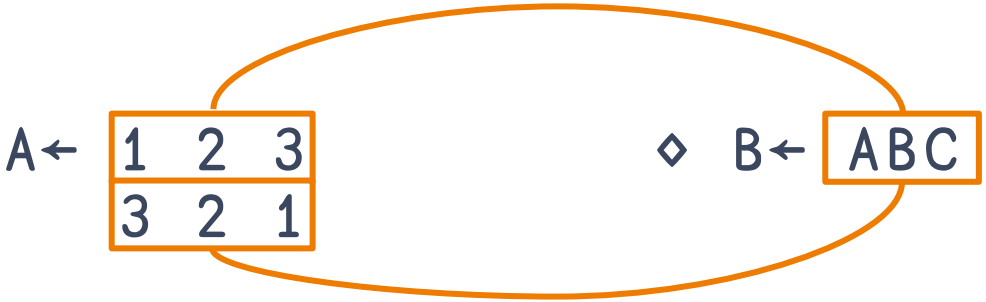
I would like



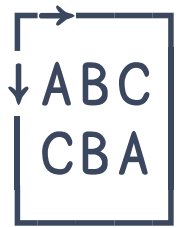

```
B[A]
```

Indexing with Nested Vectors

I have



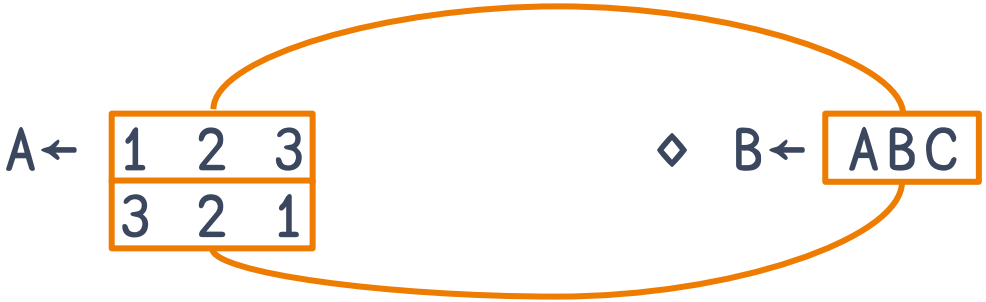
I would like



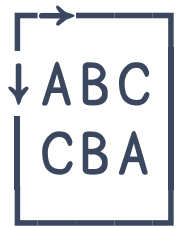
$$A \supseteq B$$

Indexing with Nested Vectors

I have



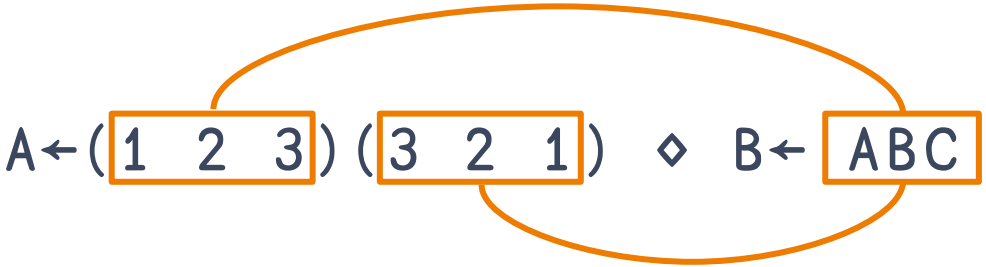
I would like



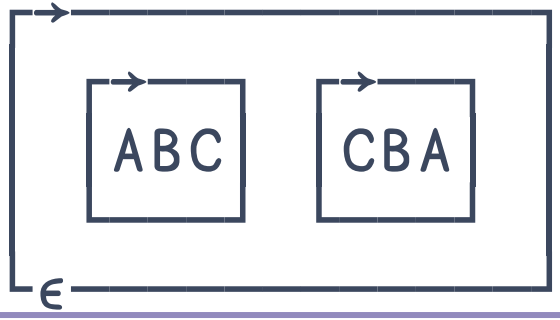

```
A(⊇⍤1)B
```

Function Application

I have

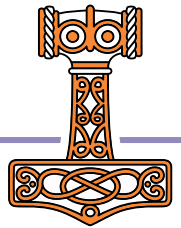


I would like




```
A(⍷?1)B
```

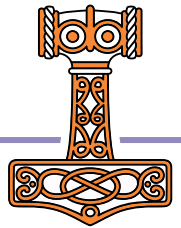
Function Application



Function Application

Depth

fök

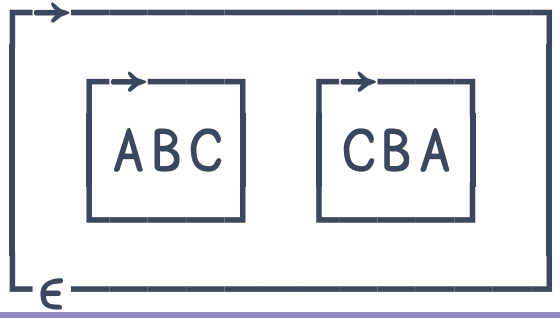


Function Application

I have

```
A←(1 2 3)(3 2 1) ⋄ B← ABC
```

I would like



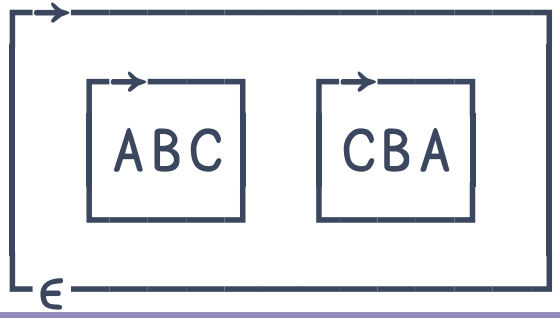

```
A(⍳⍤1)B
```

Function Application

I have

```
A←(1 2 3)(3 2 1) ⋄ B← ABC
```

I would like

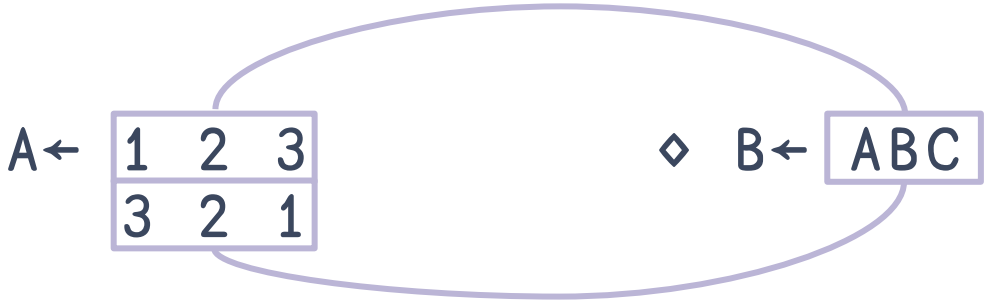



```
A(⍉⍤1)B
```

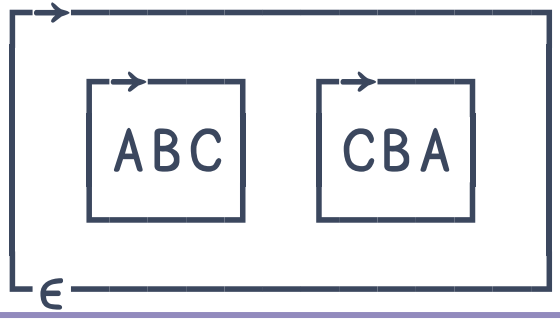
Watch this!

Function Application

I have



I would like



A (⍤ 1) B

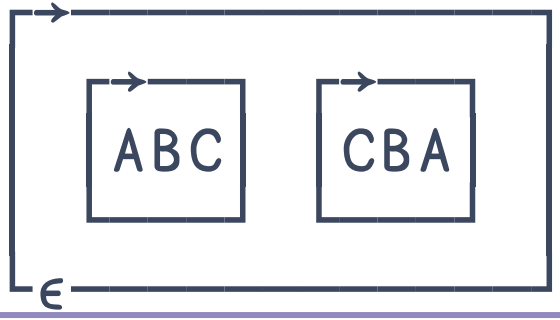
Watch this!

Function Application

I have

```
A←(1 2 3)(3 2 1) ⋄ B←ABC
```

I would like

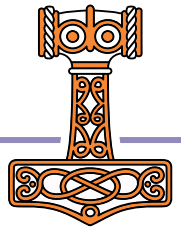



```
A(⍉⍤1)B
```

Watch this!

Function Application

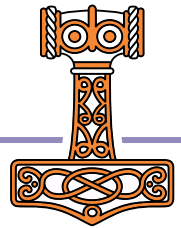
```
Fp←!
Fp 4 (5 6)
```



Function Application

```
Fp←!
Fp 4 (5 6)
```

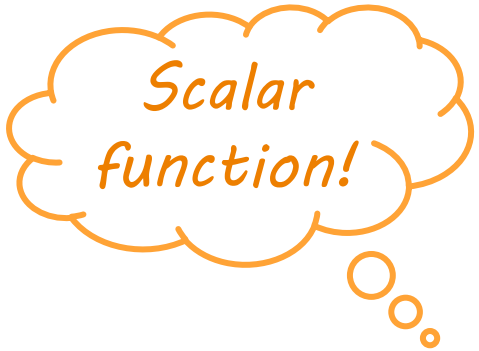
24	120 720

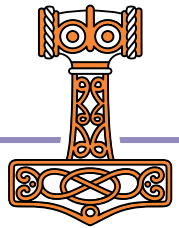


Function Application

```
Fp←!
Fp 4 (5 6)
```

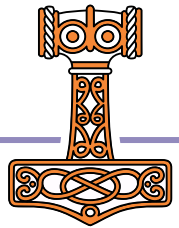
24	120 720





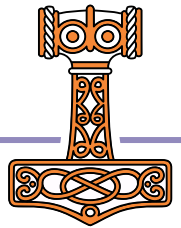
Function Application

```
F d←{×/⍳⍵}
```



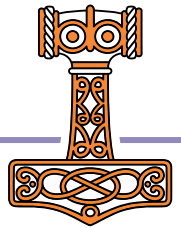
Function Application

```
      Fd←{×/⍳⍵}
      Fd 4 (5 6)
DOMAIN ERROR
Fd[0] Fd←{×/⍳⍵}
                ^
```



Function Application

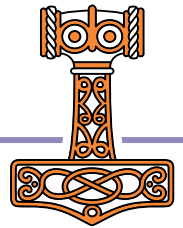
```
      Fd←{×/⍳⍵}
      Fd 4 (5 6)
DOMAIN ERROR
Fd[0] Fd←{×/⍳⍵}
                ^
      Fs←{×/⍳⍵}⍤0
```



Function Application

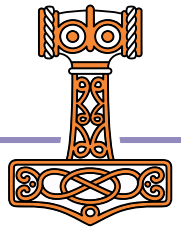
```
      Fd←{×/⍳⍵}
      Fd 4 (5 6)
DOMAIN ERROR
Fd[0] Fd←{×/⍳⍵}
                ^
      Fs←{×/⍳⍵}⍤⍤0
      Fs 4 (5 6)
```

24	120 720



Function Application

```
      □←t1←'hi'
hi
```

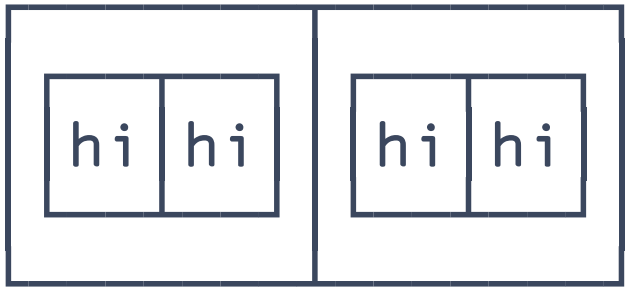


Function Application

```
⍞←t2←2⍴⊂t1←'hi'
```

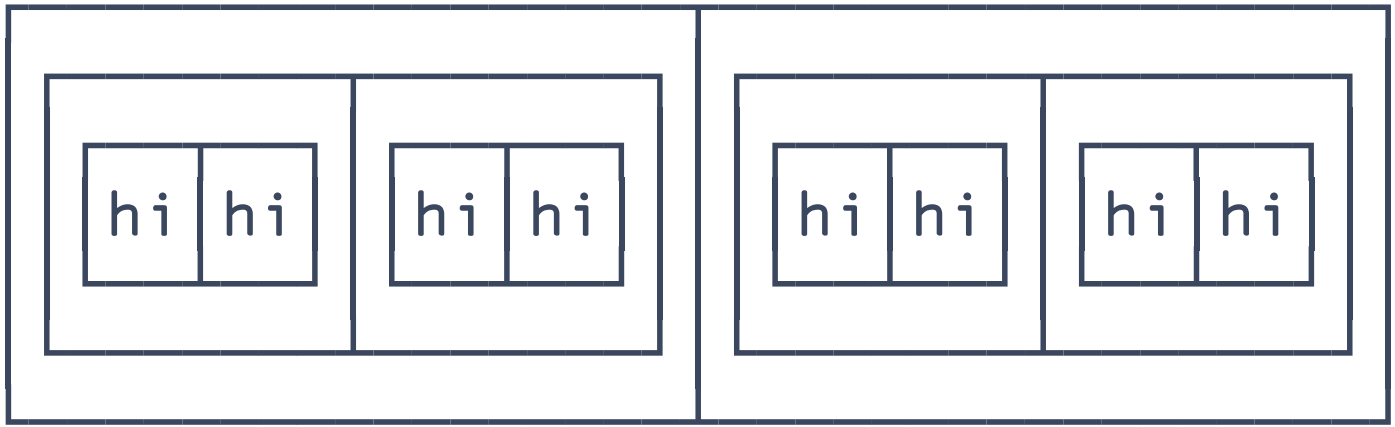
Function Application

```
⍞←t3←2⍴⊂t2←2⍴⊂t1←'hi'
```



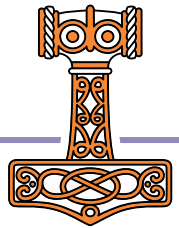
Function Application

```
⍞←t4←2⍴⊂t3←2⍴⊂t2←2⍴⊂t1←'hi'
```



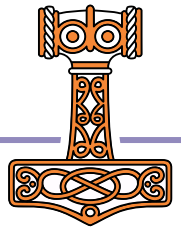
Function Application

```
t4←2⍴⊂t3←2⍴⊂t2←2⍴⊂t1←'hi'
```



Function Application

```
      t4←2⍴⊂t3←2⍴⊂t2←2⍴⊂t1←'hi'

      '^.'⎕R'\u&'⊢t1
Hi
```

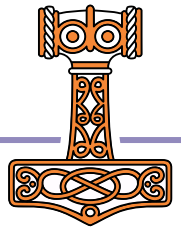


Function Application

```
t4←2⍴⊂t3←2⍴⊂t2←2⍴⊂t1←'hi'

'^.'⌷R'\u&'⊢t2
```

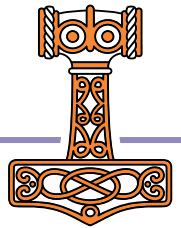
Function Application

```
      t4←2⍴⊂t3←2⍴⊂t2←2⍴⊂t1←'hi'

      '^.'⍞R'\u&'⊢t3
DOMAIN ERROR: Invalid input source
      '^.'⍞R'\u&'⊢t3
          ^
```



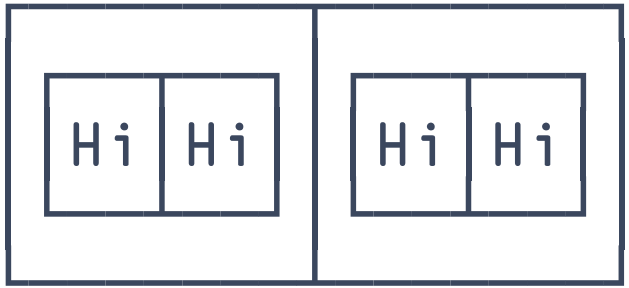
Function Application

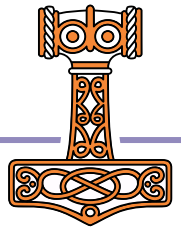
```
t4←2⍴⊂t3←2⍴⊂t2←2⍴⊂t1←'hi'

'^.'⍞R'\u&'⍤2⊢t3
```



Function Application

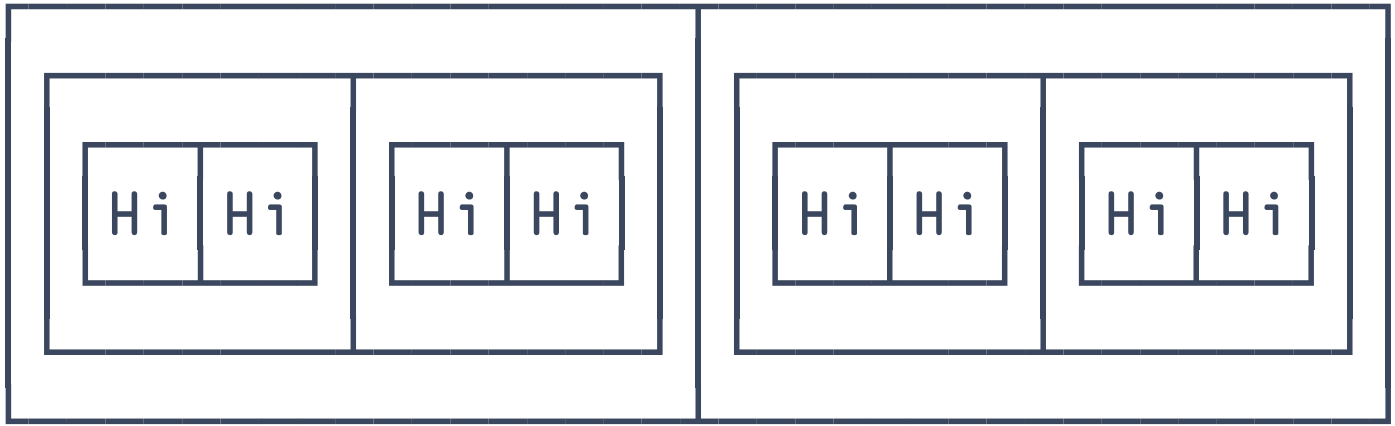
```
t4←2⍴⊂t3←2⍴⊂t2←2⍴⊂t1←'hi'

'^.'⍞R'\u&'⍤2⊢t3
```





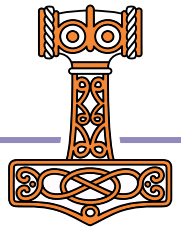
Function Application

```
t4←2⍴⊂t3←2⍴⊂t2←2⍴⊂t1←'hi'

'^.'⌷R'\u&'ö2⊢t4
```

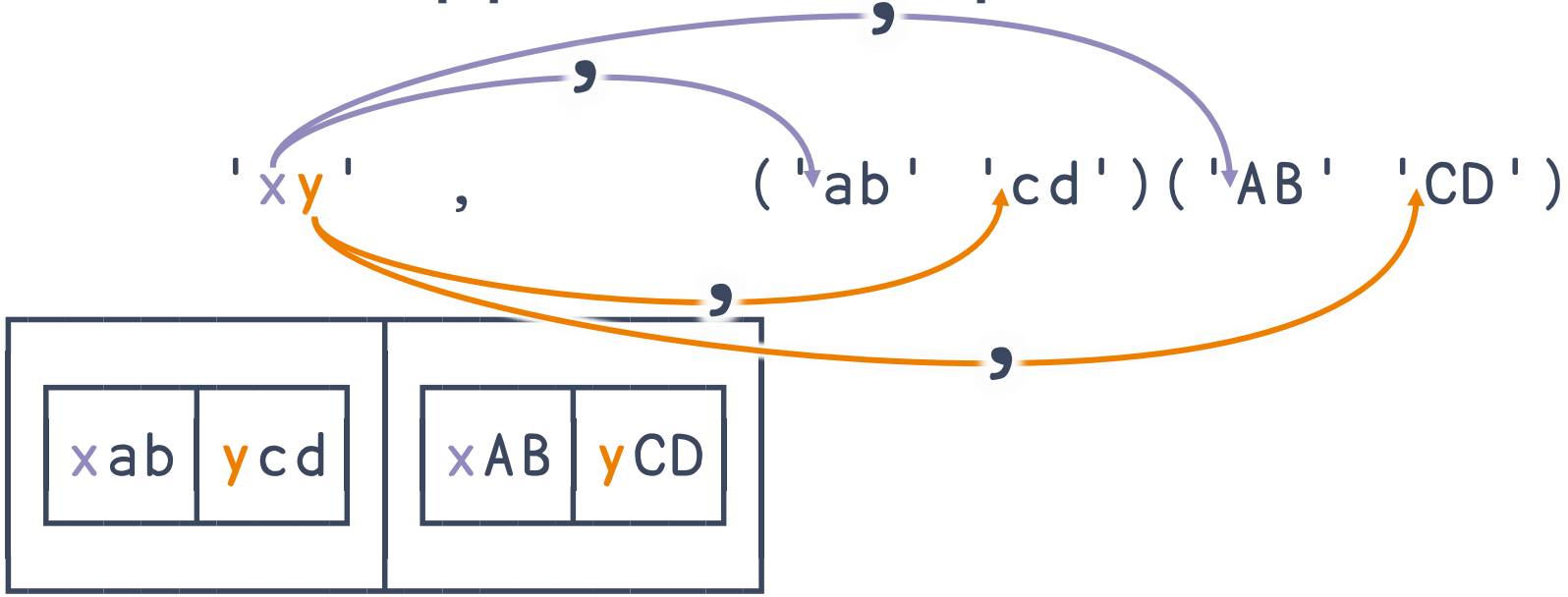


Function Application

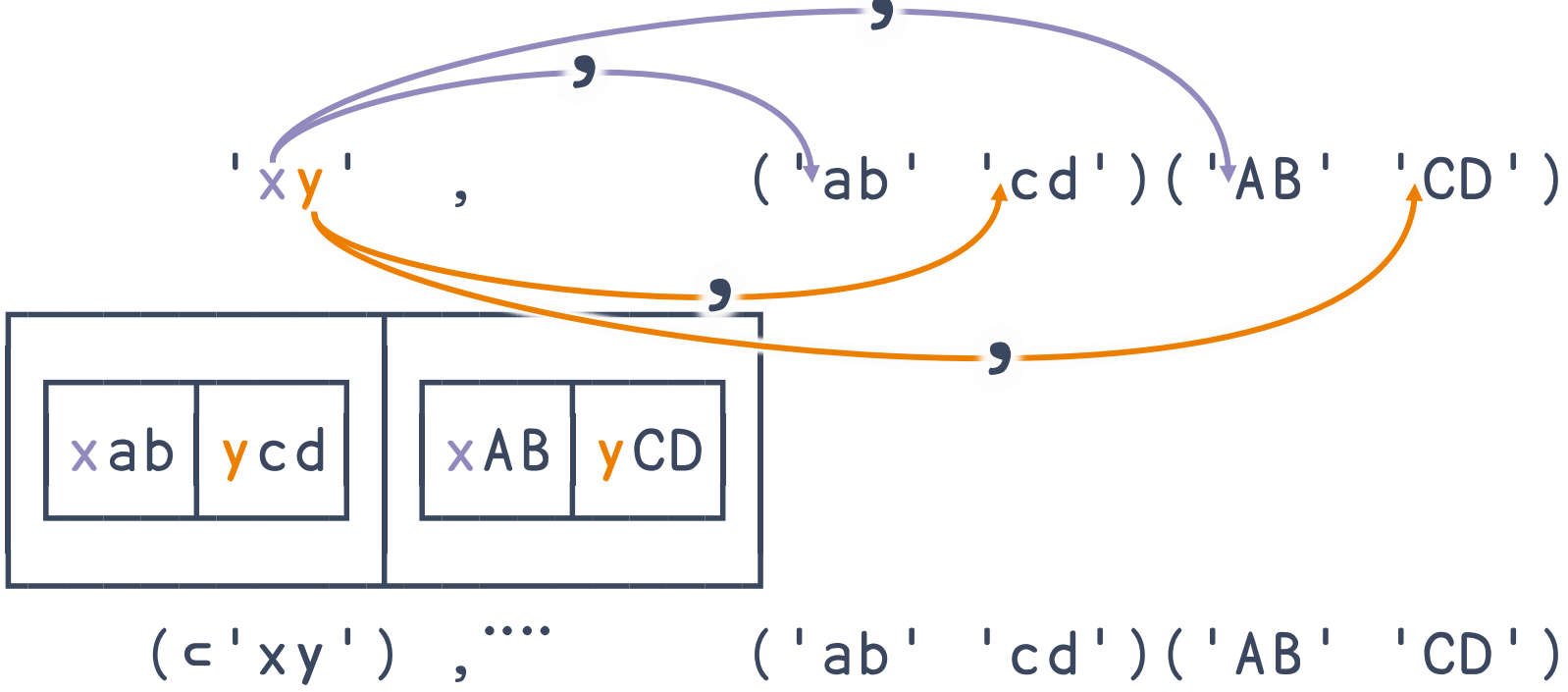
```
t4←2ρ⊂t3←2ρ⊂t2←2ρ⊂t1←'hi'

'^.'⎕R'\u&'⍤2⊢t4
{2≤|≡⍵:∇¨⍵ ⋄ '^.'⎕R'\u&'⊢⍵}
```



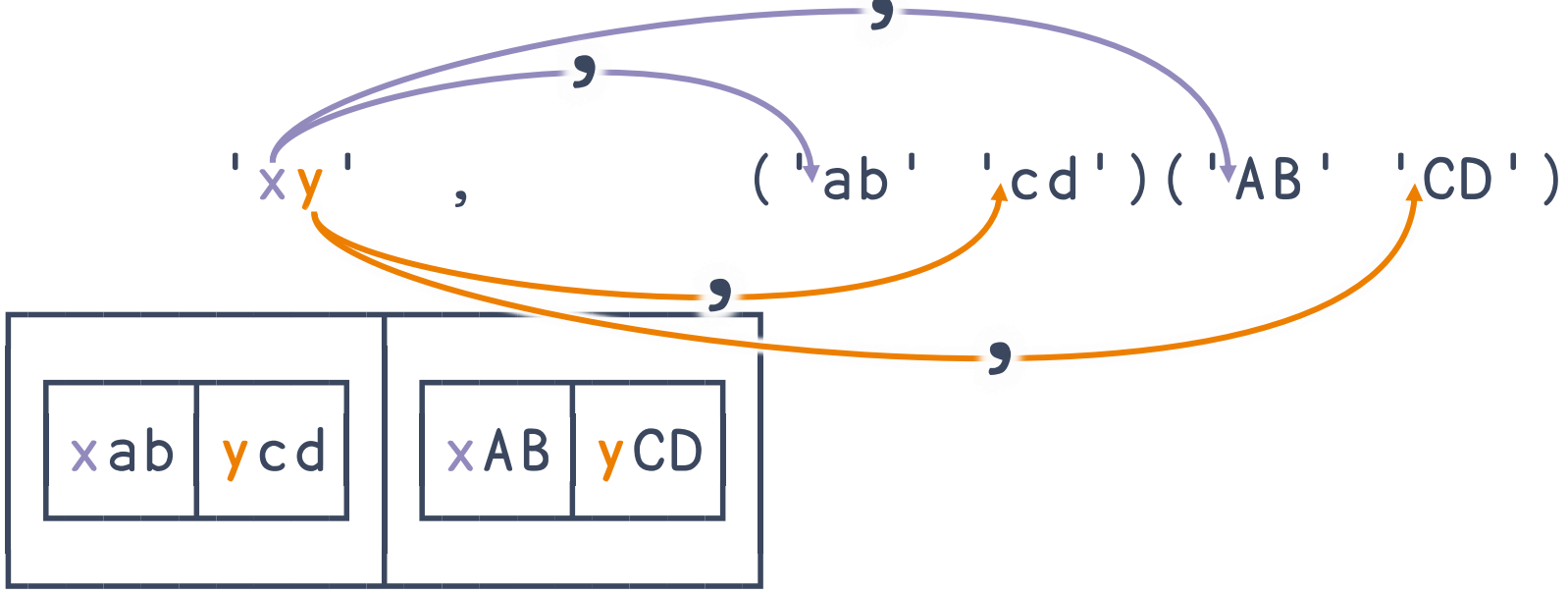
Function Application – quiz



Function Application

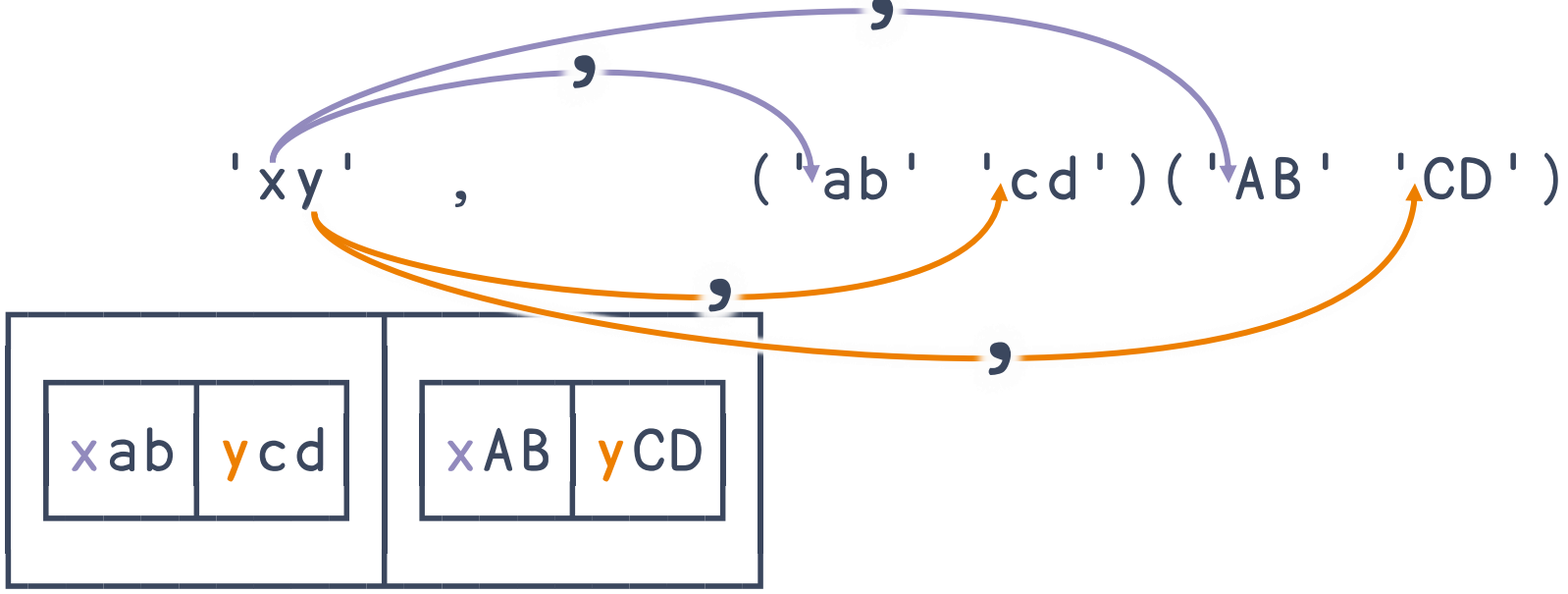


Function Application



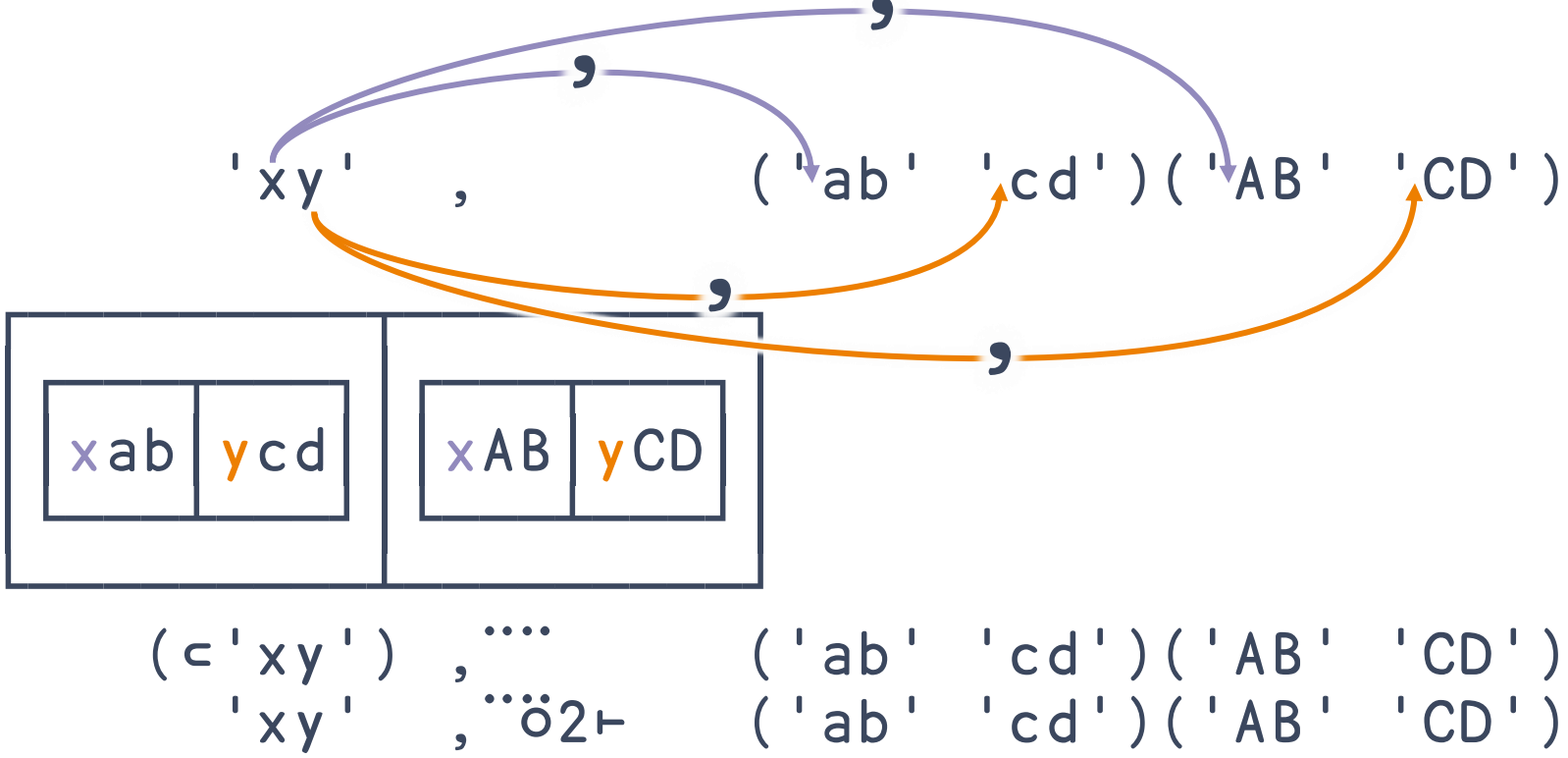

```
(⊂'xy') ,⋯⋯      ('ab' 'cd')('AB' 'CD')
  'xy' (,¨⍤1 2) ('ab' 'cd')('AB' 'CD')
```

Function Application




```
(⊂'xy') ,⍤⍤      ('ab' 'cd')('AB' 'CD')
  'xy' (,⍤⍤2)     ('ab' 'cd')('AB' 'CD')
```

Function Application



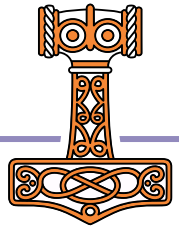
Core Language

Data Transformation

Function Application

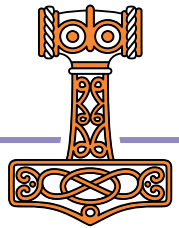
Function Composition

fök



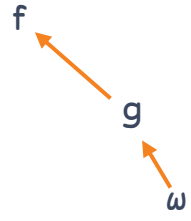
Core Language

Data Transformation	X × Y	ϕ Y	X [] Y	X ⊒ Y
Function Application	f ≠	f ⁑ g	f ⍤ k	f ⍥ k
Function Composition	f ⍤ g	f ⍥ g	f ∘ g	

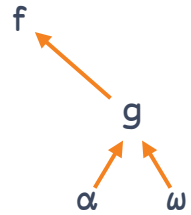


Function Composition

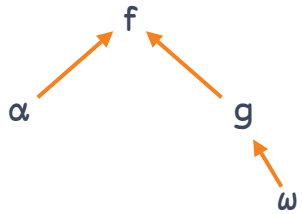
f ⍤ g f ∘ g f ⍥ g



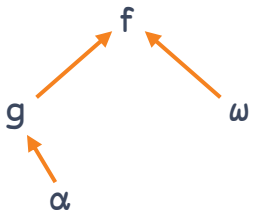
f ⍤ g



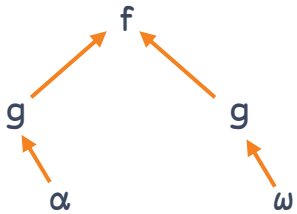
f ∘ g

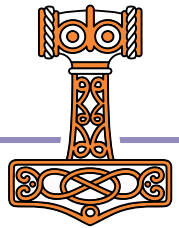


f ⍨ ∘ g ⍨



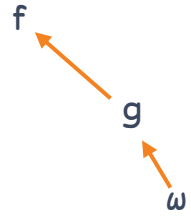
f ⍥ g



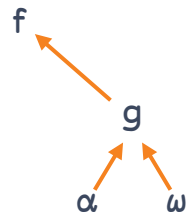


Function Composition

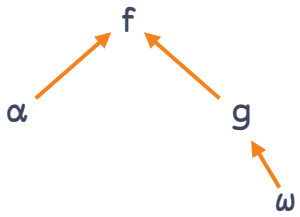
f ⍤ g f ∘ g f ⍥ g



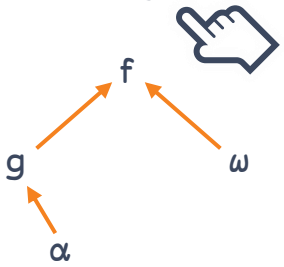
f ⍤ g



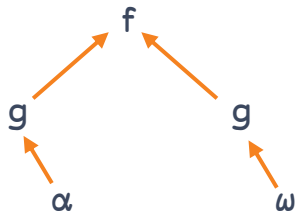
f ∘ g

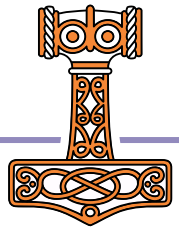


f ⍨ ∘ g ⍨



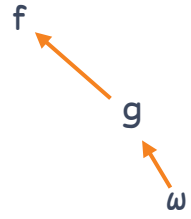
f ⍥ g



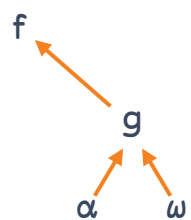


Function Composition

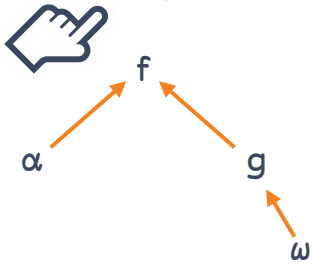
f ⍤ g f ∘ g f ⍥ g



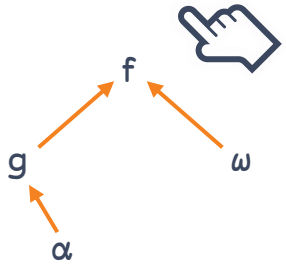
f ⍤ g



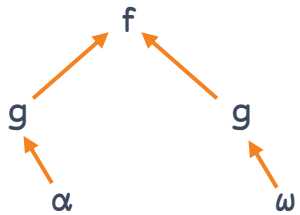
f ∘ g

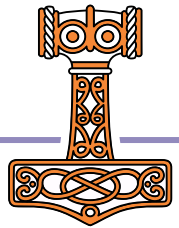


f ⍨ ∘ g ⍨



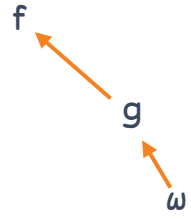
f ⍥ g



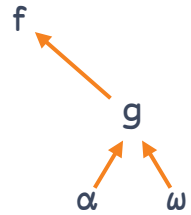


Function Composition

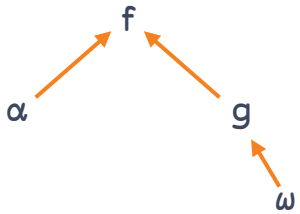
f ⍤⍤ g f ∘ g f ⍥ g



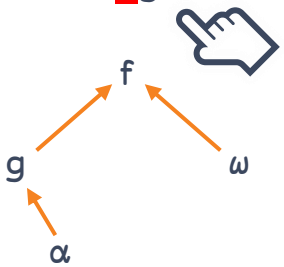
f ⍤⍤ g



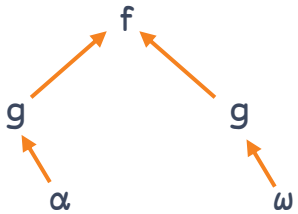
f ∘ g

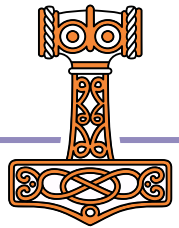


f ? g



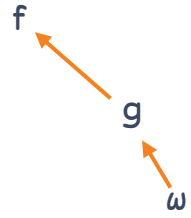
f ⍥ g



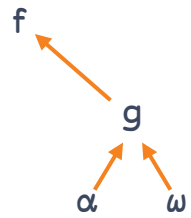


Function Composition

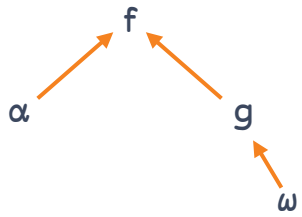
f ⍤ g f ∘ g f ⍥ g



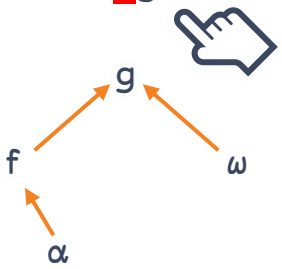
f ⍤ g



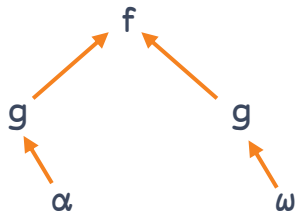
f ∘ g

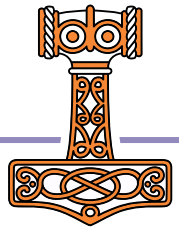


f ? g

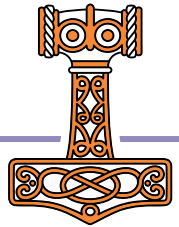


f ⍥ g





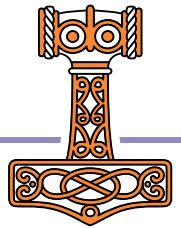
Function Composition



Function Composition

Behind

$f \circ g$

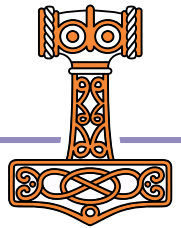


Function Composition

Behind

$$f \circ g$$

$$X \to (f \circ g) \to Y$$

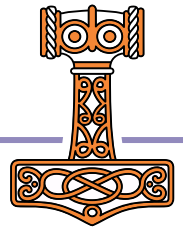


Function Composition

Behind

$$f \circ g$$

$$X \xrightarrow{\quad f \quad} Y \xrightarrow{\quad g \quad} Z$$

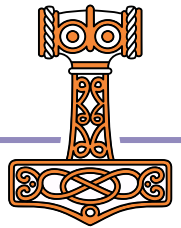


Function Composition

Behind

$$f \circ g$$

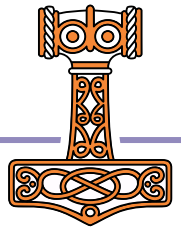
$$(f\ X)\ g\ Y$$



Function Composition

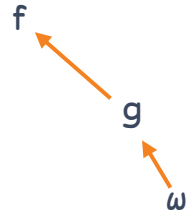
Behind

$f \circ g$

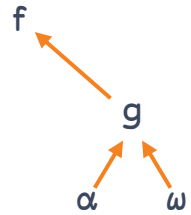


Function Composition

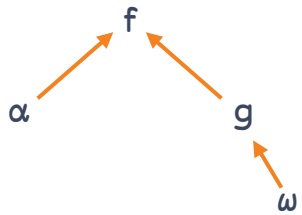
f ⚬̈ g f ⚬ g f Ö g



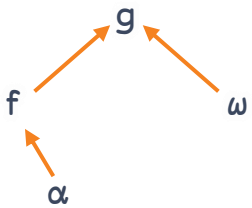
f ⚬̈ g



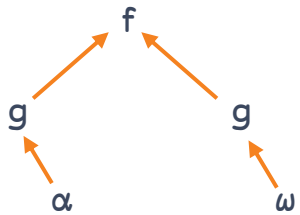
f ⚬ g

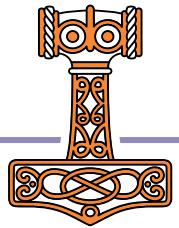


f ⚬ g



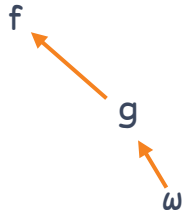
f Ö g



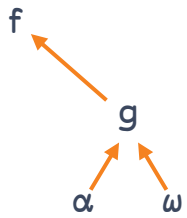


Function Composition

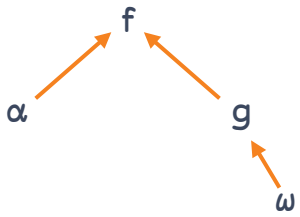
f ⍤ g f ∘ g f ⍥ g



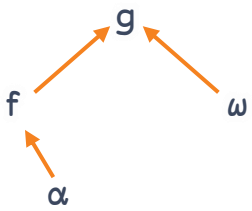
f ⍤ g



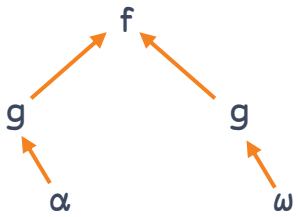
f ∘ g

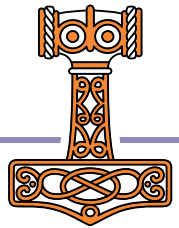


f ∘ g



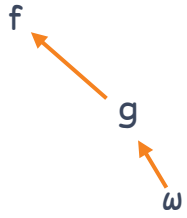
f ⍥ g



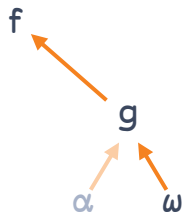


Function Composition

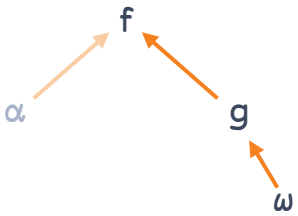
f ⍤ g f ∘ g f ⍥ g



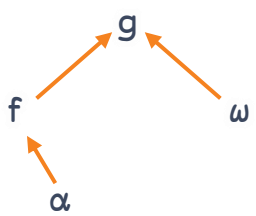
f ⍤ g



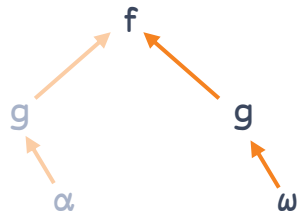
f ∘ g

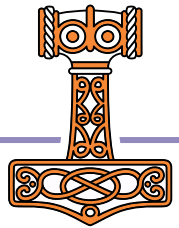


f ∘ g



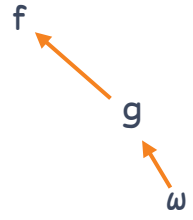
f ⍥ g



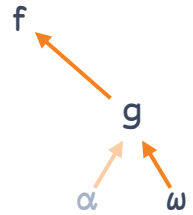


Function Composition

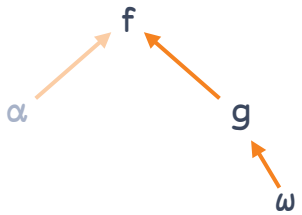
f ⚬̈ g f ⚬ g f ⊙̈ g



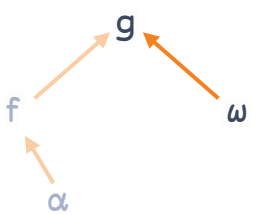
f ⚬̈ g



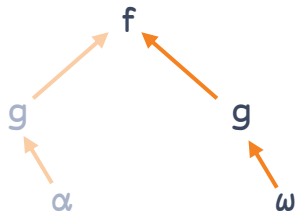
f ⚬ g

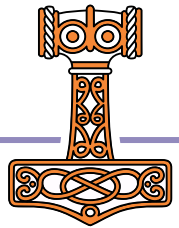


f ⚬ g

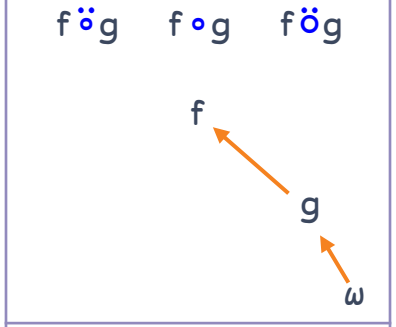


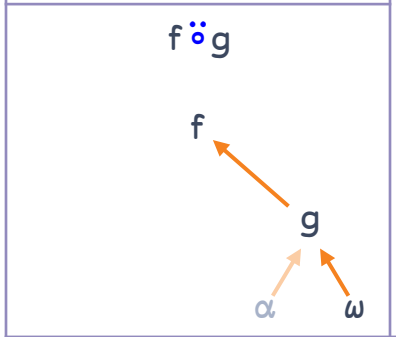
f ⊙̈ g



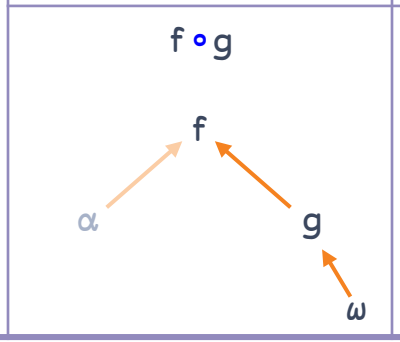


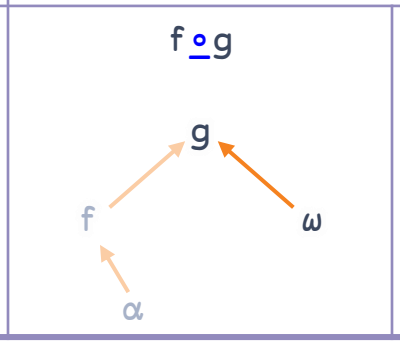
Function Composition

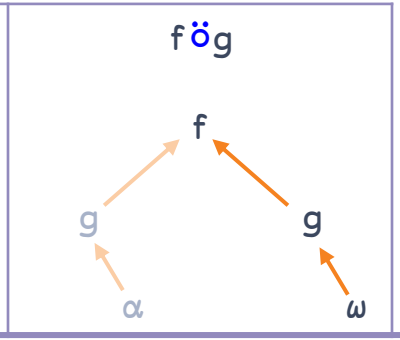




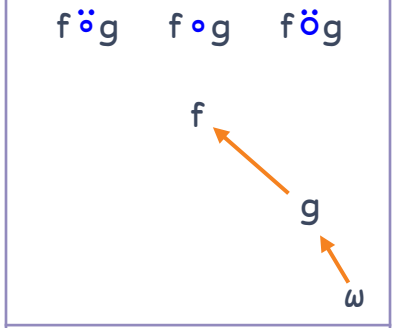
f ∘ g ω ⇔ g ω

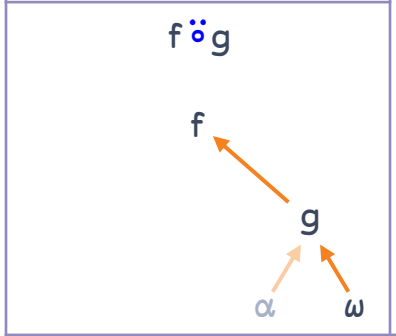




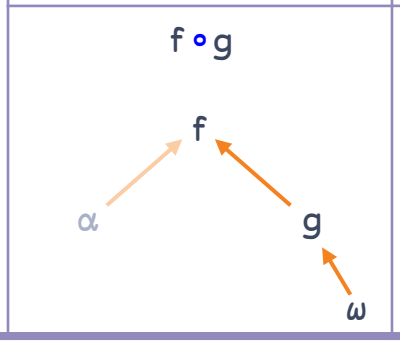


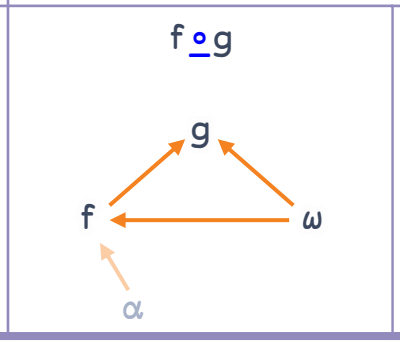
Function Composition

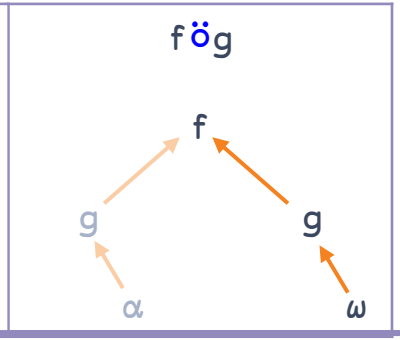




$$f \underline{\circ} g \;\; \omega \;\Leftrightarrow\; f \underline{\circ} g \tilde{\ddot{\omega}}$$







Essays/Hook Conjunction?

≺ Essays

New Users
J code search (new)
NuVoc
J in a browser
Guides
System
Showcase
Library
Community
Recent changes
Prototype wiki

Tools
What links here
Related changes
Special pages
Permanent link
Page information
Cite this page

Print/export
Create a book
Download as PDF
Printable version

Hook is a 2-train, an isolated sequence of two verbs, introduced in APL by K.E. Iverson and E.E. McDonnell, *Phrasal Forms*, APL89, APL Quote-Quad, Volume 19, Number 4, 1989-08. It is defined as follows:

```
  (g h) y   ↔   y g h y
x (g h) y   ↔   x g h y
```

For example, the monad `(=<.)` is a test for integers and `(+%)/` computes a continued fraction -- `(+%)/20$1` is an approximation of the golden ratio. Hook is based on the S combinator of combinatory logic .

With over 17 years of hindsight, I believe it would have been better to use a conjunction (denoted by `h.`, say) to denote a hook rather than using a 2-train. Everything that can be done with the 2-train `(f g)` can be done with the conjunction `h.`, but `h.` does not require a special parsing rule.

The original motivation for assigning a meaning to a train of length 2 was so that a train of any length (greater than 1) would be interpreted: A train with odd length is a sequence of forks; a train with even length is either a hook (if of length 2) or a hook followed by a sequence of forks (if of length >2). Again with hindsight, the alternatives are:

0. Leave trains of even length uninterpreted -- just signal error.

1. Assign the "at" meaning to it:

```
    (g h) y   ↔  g   h y
  x (g h) y   ↔  g x h y
```

That is, the capped fork meaning. You'd probably still have the capped fork. Compare:

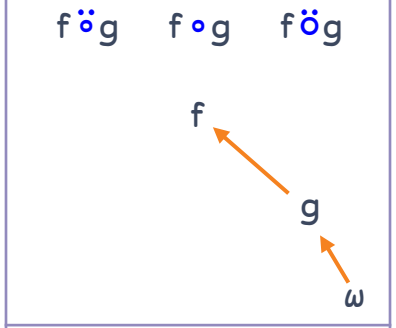
```
  [: f0 [: f1 f2 f3 f4
  (f0 (f1 (f2 f3 f4)))
```

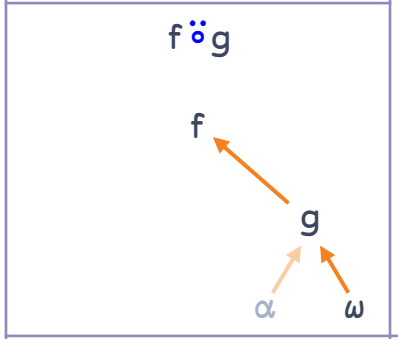
See also

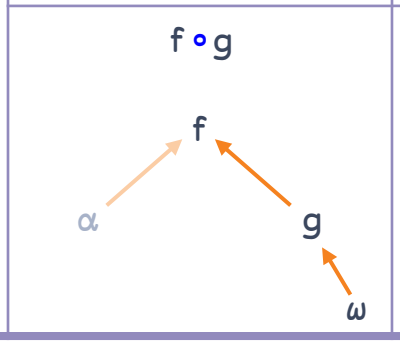
- Trains

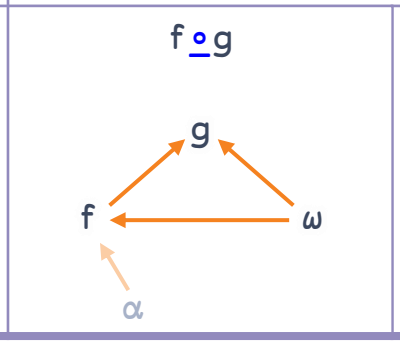
Contributed by Roger Hui.

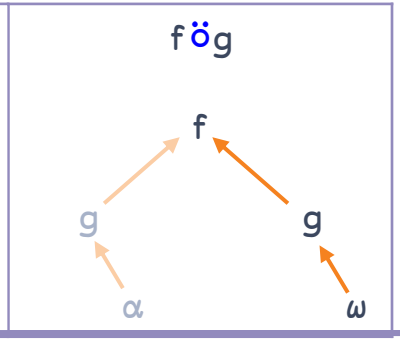
Function Composition







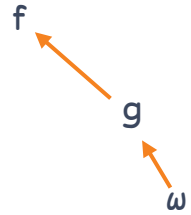




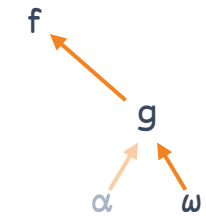
$$f \mathbin{\underline{\circ}} g \;\; \omega \;\Leftrightarrow\; f \mathbin{\underline{\circ}} g \mathbin{\ddot{\sim}} \omega$$

Function Composition

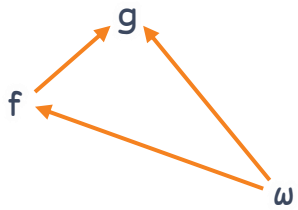
f ⍤ g f ∘ g f ⍥ g



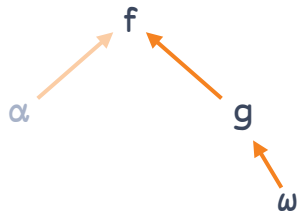
f ⍤ g



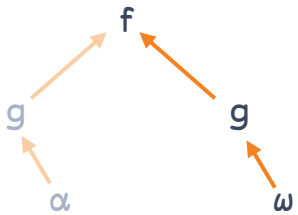
f ∘ g

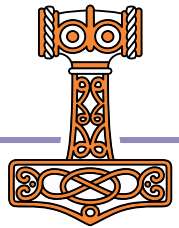


f ∘ g



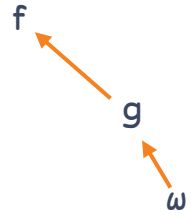
f ⍥ g



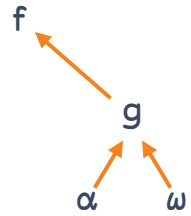


Function Composition

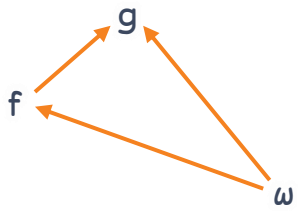
f ⊙ g f ∘ g f ⊙ g



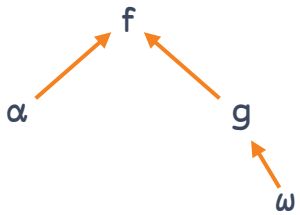
f ⊙ g



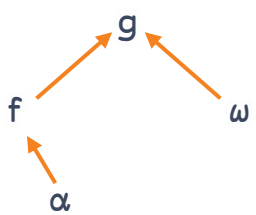
f ∘ g



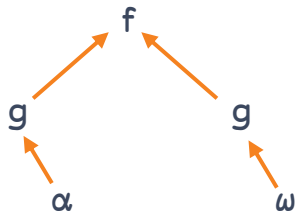
f ∘ g



f ∘ g

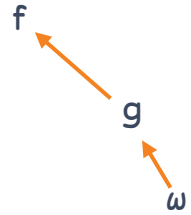


f ⊙ g

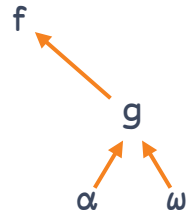


Function Composition

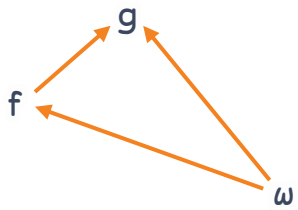
f ⍤ g f ∘ g f ⍥ g



f ⍤ g

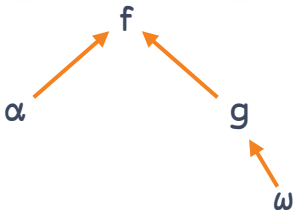


f ∘ g

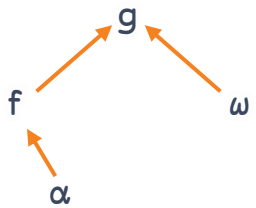


f ∘ g

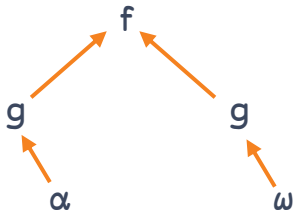
preprocess right

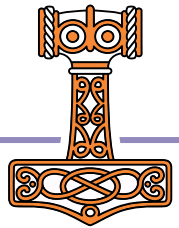


f ∘ g



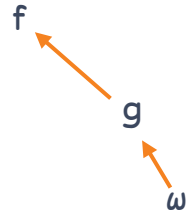
f ⍥ g



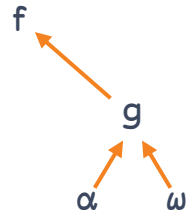


Function Composition

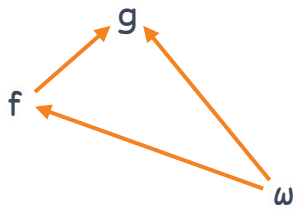
f⊙g f∘g f⊙g



f⊙g

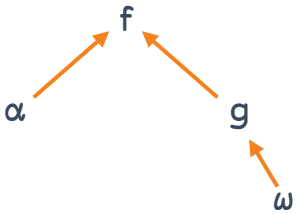


f∘g



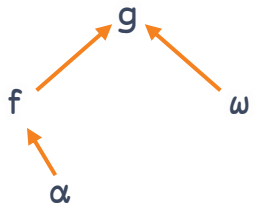
f∘g

preprocess right

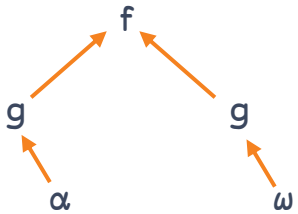


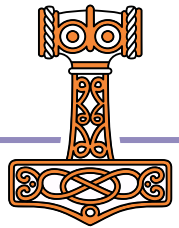
f∘g

preprocess left



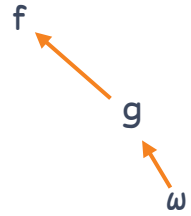
f⊙g



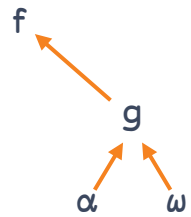


Function Composition

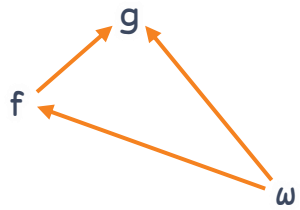
f ⍤ g f ∘ g f ⍥ g



f ⍤ g

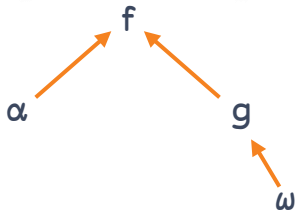


f ∘ g



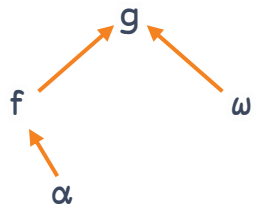
f ∘ g

preprocess right



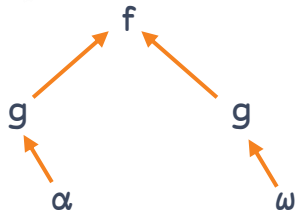
f ⍥ g

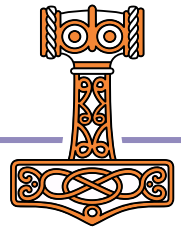
preprocess left



f ⍥ g

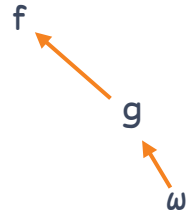
preprocess both





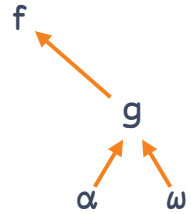
Function Composition

f ⍤ g f ∘ g f ⍥ g

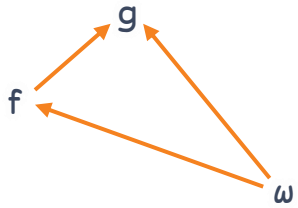


f ⍤ g

postprocess

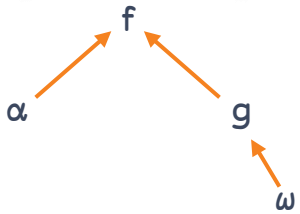


f ∘ g



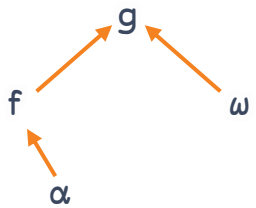
f ∘ g

preprocess right



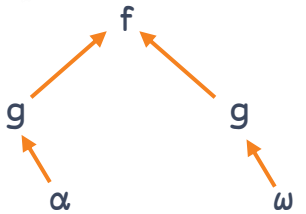
f ∘ g

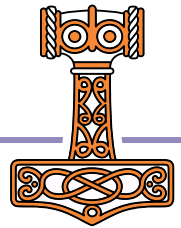
preprocess left



f ⍥ g

preprocess both

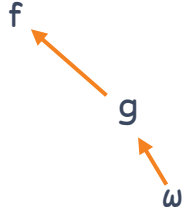




Function Composition

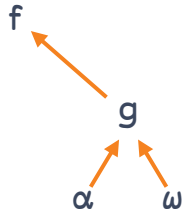
f⍤⍤g f∘g f⍥g

pre/postprocess

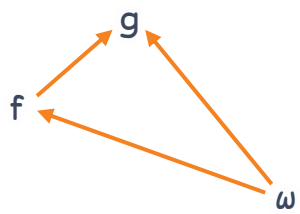


f⍤⍤g

postprocess

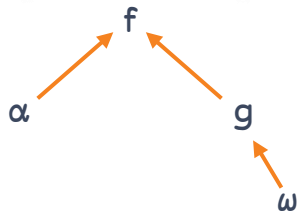


f∘⍥g



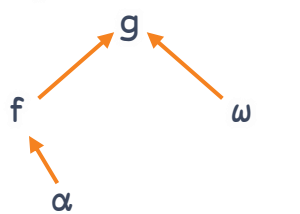
f∘g

preprocess right



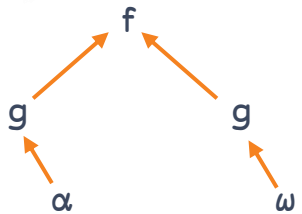
f⍥g

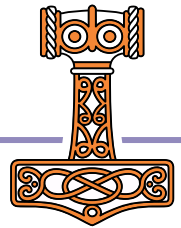
preprocess left



f⍥g

preprocess both

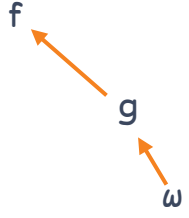




Function Composition

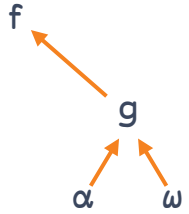
f⊙g f∘g f⊙g

pre/postprocess



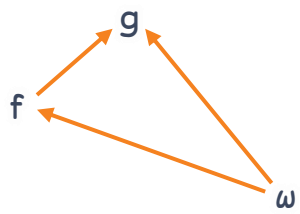
f⊙g

postprocess



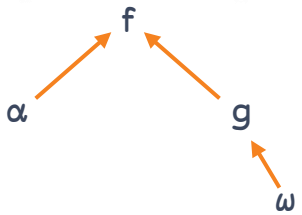
f∘g

preprocess left



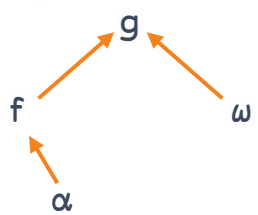
f∘g

preprocess right



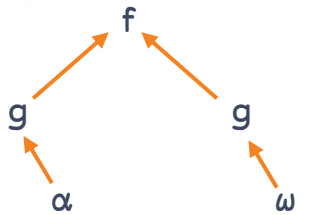
f∘g

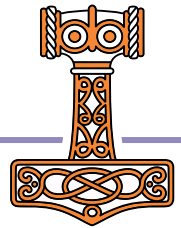
preprocess left



f⊙g

preprocess both





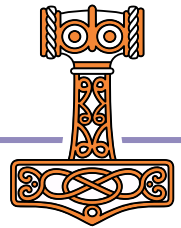
Function Composition

preprocess left

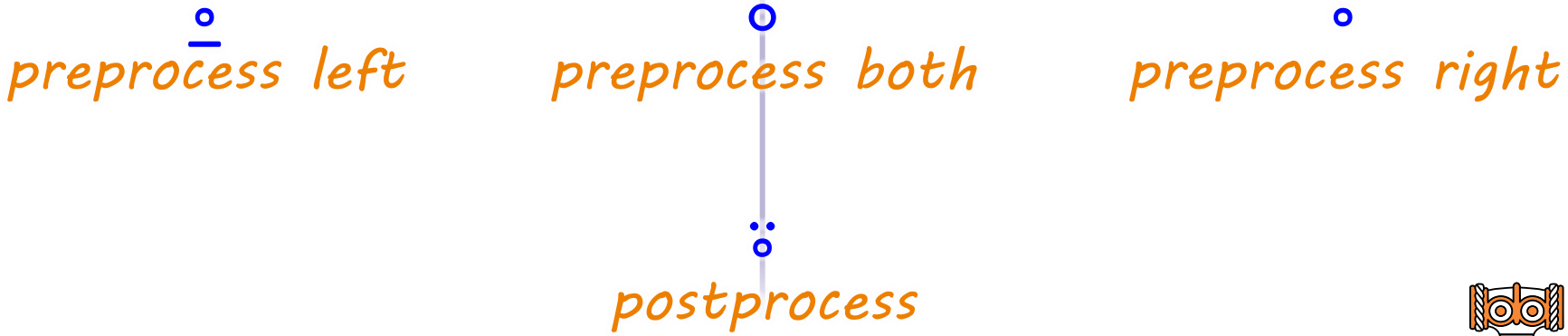
preprocess both

preprocess right

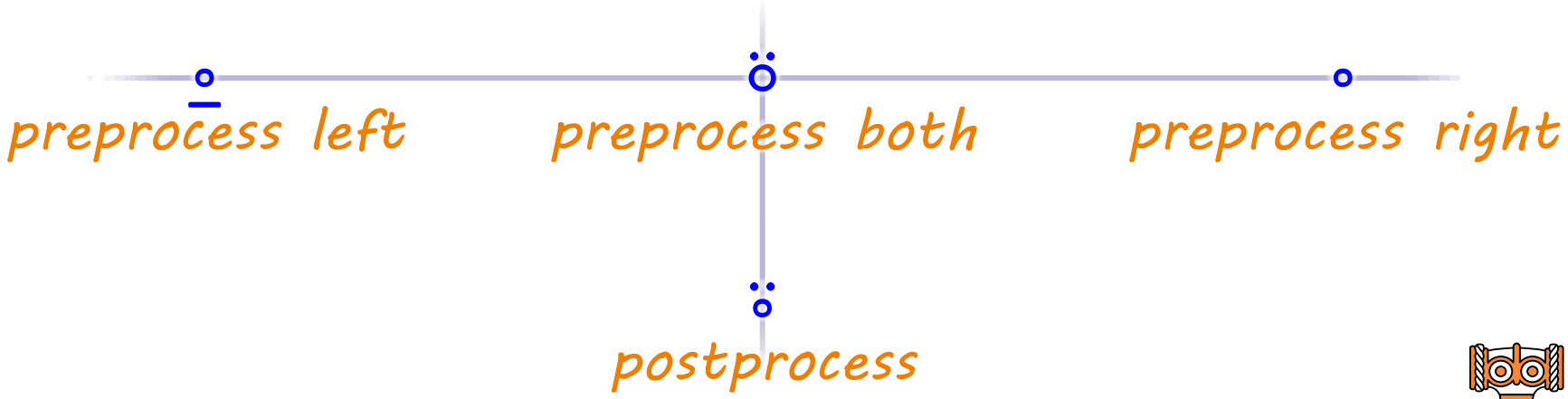
postprocess



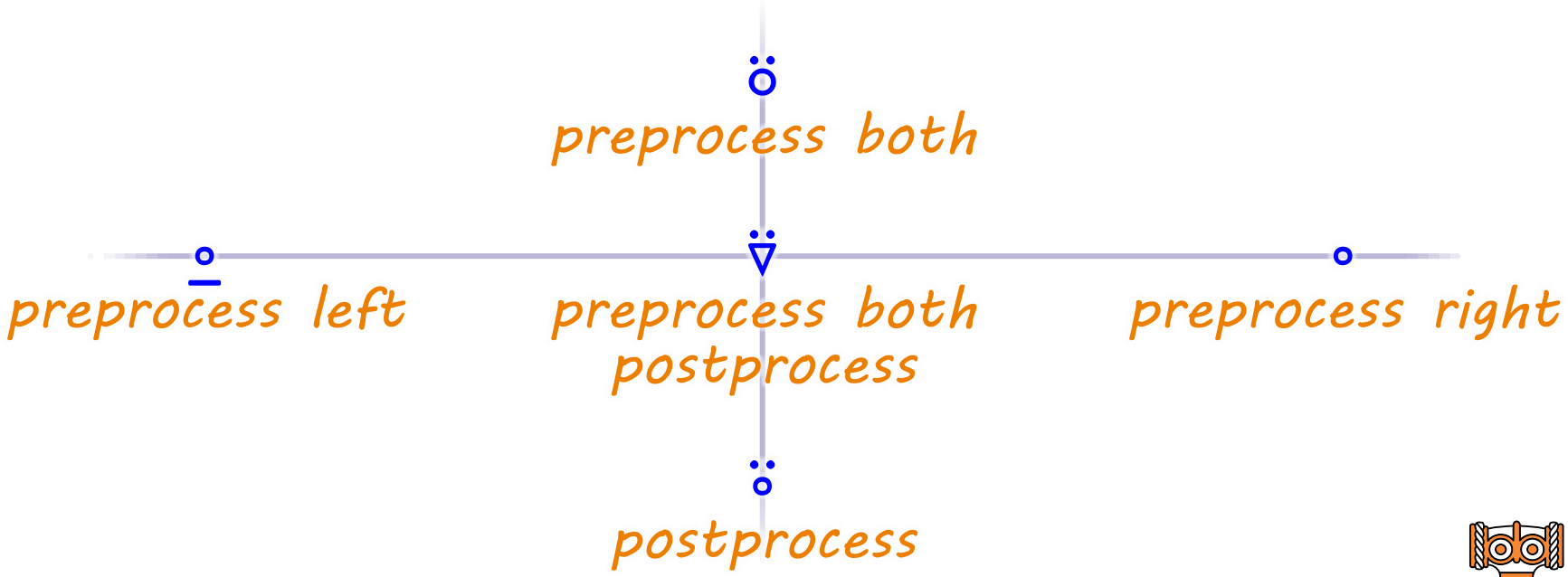
Function Composition



Function Composition

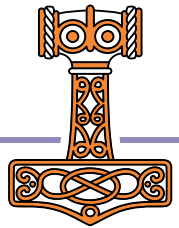


Function Composition



Function Composition

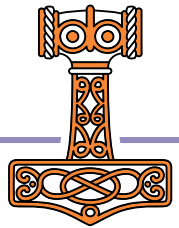
`f ∘ g` Behind with `X ⊇ Y` Select/Permute



Function Composition

`f∘g` Behind with `X⊇Y` Select/Permute

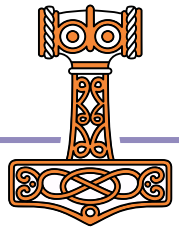
- `Sort ← (⍋⊇⊢)`



Function Composition

`f⊙g` Behind `X⊒Y` with `X⊒Y` Select/Permute

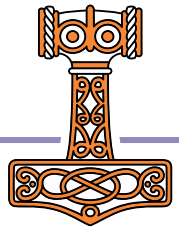
- `Sort ← ⍏⊙⊒`



Function Composition

`f⊙g` Behind with `X⊇Y` Select/Permute

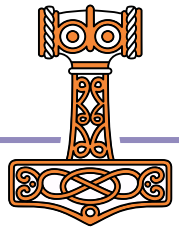
```
Sort  ← ⍋⊙⊇
Sorts ← ⊇⍨∘⍋⍨      ⍝ "sort Y by X"
```



Function Composition

`f ∘ g` Behind with `X ⊇ Y` Select/Permute

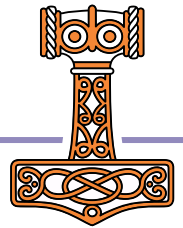
```
Sort  ← ⍋∘⊇
Sorts ← ⍋∘⊇  ⍝ "sort Y by X"
```



Function Composition

`f⊙g` Behind with `X⊇Y` Select/Permute

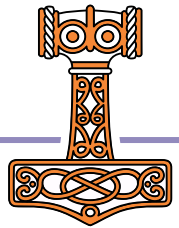
```
Sort    ← ⍏⊙⊇
Sorts   ← ⍏⊙⊇          ⍝ "sort Y by X"
Shuffle ← (?⍨⊙≠⊇⊢)
```



Function Composition

`f⊙g` Behind with `X⊇Y` Select/Permute

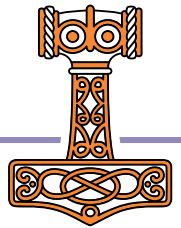
```
Sort    ← ⍏⊙⊇
Sorts   ← ⍏⊙⊇          ⍝ "sort Y by X"
Shuffle ← ?⍤⊙≠⊙⊇
```



Function Composition

`f⊙g` Behind

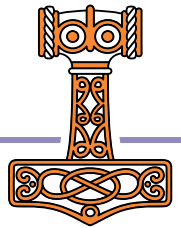
- `SameAsFirst ← ↄ⊙=`
- `HasDuplicates ← ∪⊙≡`
- `Palindrome ← ϕ⊙≡`
- `IsPermutation ← ⍋⍤⍋⊙≡`



Function Composition

`f∘g` Behind

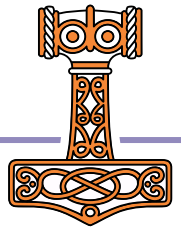
- `Integer ← ⌊∘=`
- `Split ← (⊃∘≠⊆⊢)`
- `Scale ← ⌈/∘(÷¨)`
- `Deviation ← (+⌿÷≢)∘(-¨)`



Function Composition

`f∘g` Behind

```
Integer   ← ⌊∘=
Split     ← (⊃∘≠⊆⊢)
Scale     ← ⌈/∘(÷¨)
Deviation ← (+⌿÷≠)∘(-¨)
```

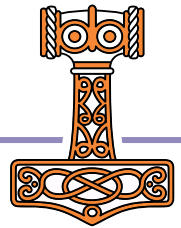


Function Composition

f⍥g Behind

```
Integer   ← ⌊⍥=
Split     ← (⊃⍥≠⊆⊢)
Scale     ← ⌈/⍥(÷⍨)
Deviation ← (+⌿÷≢)⍥(-⍨)
```

```
'/' (≠⊆⊢)  'hi/how/goes'
   (⊃⍥≠⊆⊢)'/hi/how/goes'
'/'(⊃⍥≠⊆⊢)'/hi/how/goes'
```

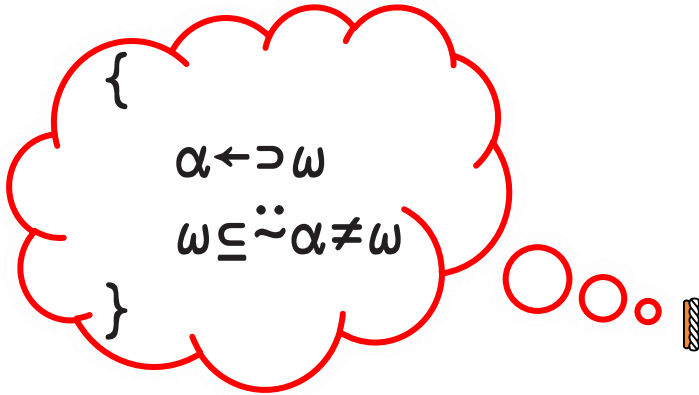


Function Composition

f⍥g Behind

```
Integer   ← ⌊⍥=
Split     ← (⊃⍥≠⊆⊢)
Scale     ← ⌈/⍥(÷⍨)
Deviation ← (+⌿÷≢)⍥(-⍨)
```

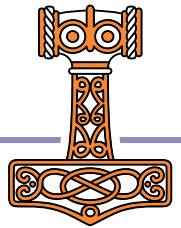
```
'/' (≠⊆⊢)  'hi/how/goes'
   (⊃⍥≠⊆⊢)'/hi/how/goes'
'/'(⊃⍥≠⊆⊢)'/hi/how/goes'
```



Function Composition

`f∘g` Behind

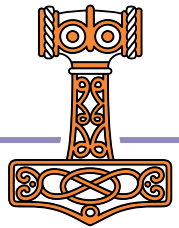
- `Integer ← ⌊∘=`
- `Split ← (⊃∘≠⊆⊢)`
- `Scale ← ⌈/∘(÷⍨)`
- `Deviation ← (+⌿÷≢)∘(-⍨)`



Function Composition

`f ∘ g` Behind

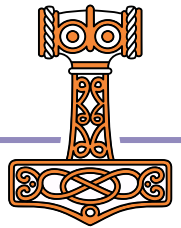
```
Filters ← ∘⌿
```



Function Composition

`f∘g` Behind

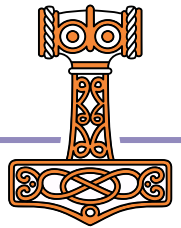
```
      Filters ← ∘⌿
      >∘5 Filters 2 7 1 8 2 8
7 8 8
```



Function Composition

f∘g Behind

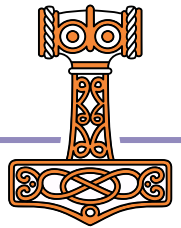
```
      Filters ← ∘⌿
      >∘5 Filters 2 7 1 8 2 8
7 8 8
      ⌽∘≡¨Filters 'racecar' 'racer' 'toot'
 racecar  toot
```



Function Composition

`f⍥g` Behind

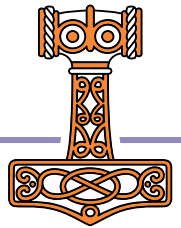
```
      >⍥5 ⍥≠ 2 7 1 8 2 8
7 8 8
      ⍞⍥≡¨⍥≠ 'racecar' 'racer' 'toot'
 racecar  toot
```



Function Composition

f⍥g Behind

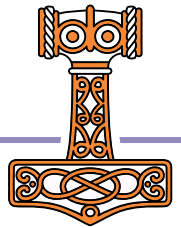
- `Whence ← ⍳⍥∈` `⍝ {(⍳⍺)∈⍵}`
- `InPoly ← ⍪⍥⊥` `⍝ {(⍪⍺)⊥⍵}`
- `Shapes ← ⍴⍥⍴` `⍝ {(⍴⍺)⍴⍵}`
- `ToFile ← ⊂⍥⎕NPUT` `⍝ {(⊂⍺)⎕NPUT ⍵}`



Function Composition

f ∘ g Behind

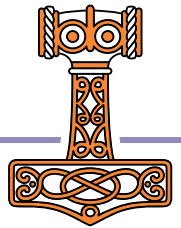
- FCat ← ⊸∘,
- RIndex ← ⊖∘⍳ ⍝ {(⊖⍺)⍳⍵}
- RDrop ← -∘↓ ⍝ {(-⍺)↓⍵}
- RndSfx ← -⍥?∘↑ ⍝ {(-?⍺)↑⍵}



Function Composition

f ∘ g Behind

- FCat ← ⊸∘,
  RIndex ← ⊖∘⍳ ⍝ {(⊖⍺)⍳⍵}
  RDrop ← -∘↓ ⍝ {(-⍺)↓⍵}
  RndSfx ← -⍥?∘↑ ⍝ {(-?⍺)↑⍵}



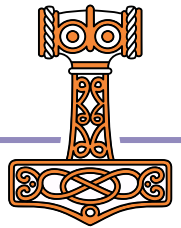
Function Composition

f⍥g Behind

- `FCat ← ⫢⍥,`

```
RIndex ← ⊖⍥⍳      ⍝ {(⊖⍺)⍳⍵}
RDrop  ← -⍥↓      ⍝ {(-⍺)↓⍵}
RndSfx ← -⍤?⍥↑    ⍝ {(-?⍺)↑⍵}
```

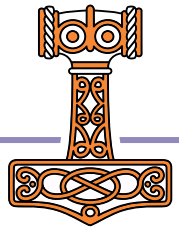
APL\360: `'abc';42;'def'`
Dyalog: `'abc',(⫢42),'def'`
19.0?: `'abc',42⫢⍥,'def'`



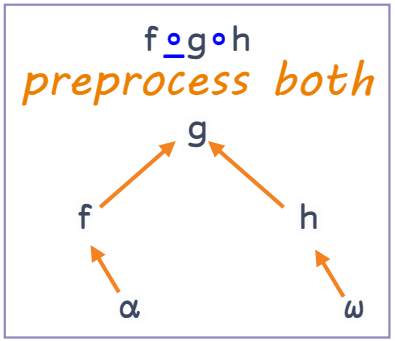
Function Composition

`f ∘ g` Behind

Split-compose `X f ∘ g ∘ h Y`



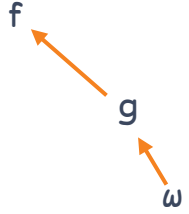
Function Composition



Function Composition

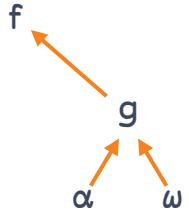
f⊘g f∘g f⊙g

pre/postprocess



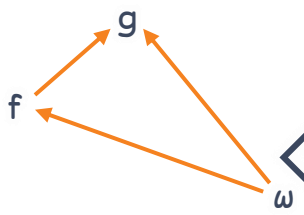
f⊘g

postprocess



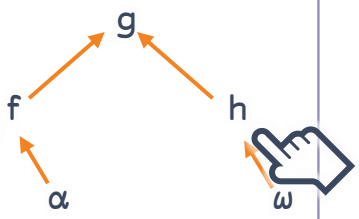
f∘g

preprocess left



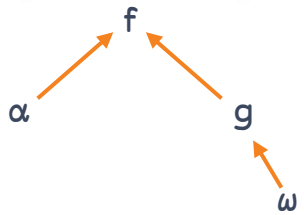
f∘g∘h

preprocess both



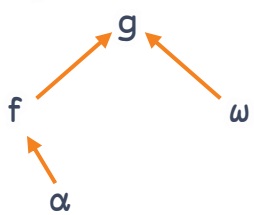
f∘g

preprocess right



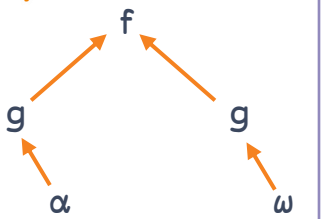
f∘g

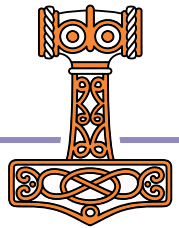
preprocess left



f⊙g

preprocess both

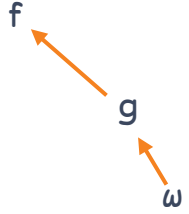




Function Composition

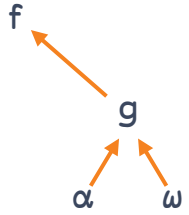
f⍥g f⍤g f⍥⍤g

pre/postprocess



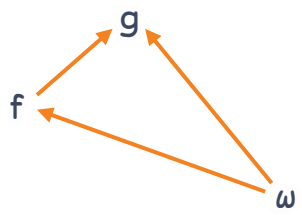
f⍥g

postprocess



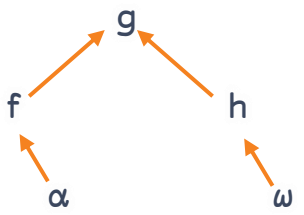
f⍤g

preprocess left



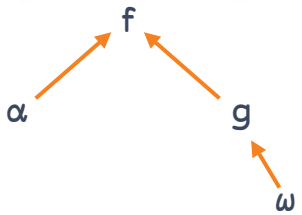
f⍤g∘h

preprocess both



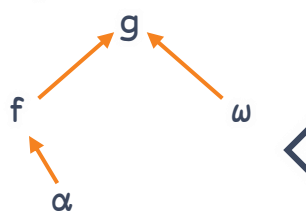
f∘g

preprocess right



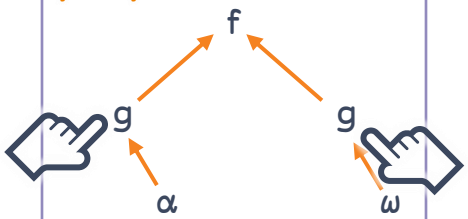
f⍤g

preprocess left



f⍥g

preprocess both

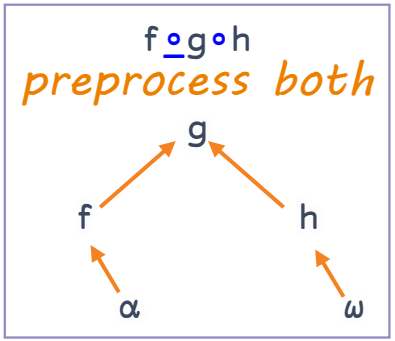


α g⍤f⍤g ω

⇔

α f⍥g ω

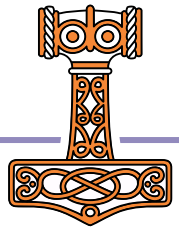
Function Composition



Function Composition

`f ∘ g` Behind

Split-compose `X f ∘ g ∘ h Y`

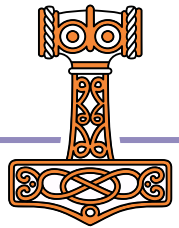


Function Composition

f ∘ g Behind

Split-compose X f ∘ g ∘ h Y

Pre-18.0: g ̈ ∘ f ̈ ∘ h
18.0: (f ̈ ⊣ g h ̈ ⊢)
19.0?: f ∘ g ∘ h

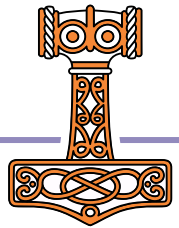


Function Composition

`f⊙g` Behind

Split-compose `X f⊙g∘h Y`

Pre-18.0: `((f⊣)g(h⊢))`
18.0: `(f⍤⊣ g h⍤⊢)`
19.0?: `f⊙g∘h`

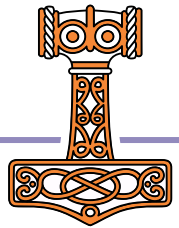


Function Composition

`f ∘ g` Behind

Split-compose `X f ∘ g ∘ h Y`

Hybrid mitigation `2 ∘ | ∘ /`



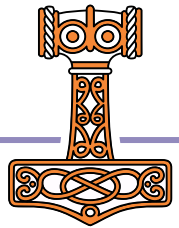
Function Composition

`f⍥g` Behind

Split-compose `X f⍥g∘h Y`

Hybrid mitigation `2∘| ⍥/`

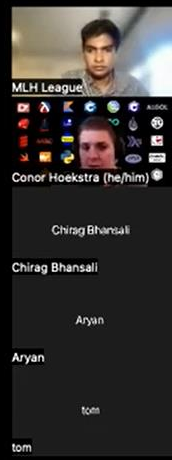
Pre-18.0: `(2∘|{⍺/⍵}⊢)`
18.0: `(2∘|⊢⍤⍥/⊢)`
19.0?: `2∘| ⍥/`



Conor Hoekstra's Opening Talk (https://youtu.be/8oKAHQsh1oM)

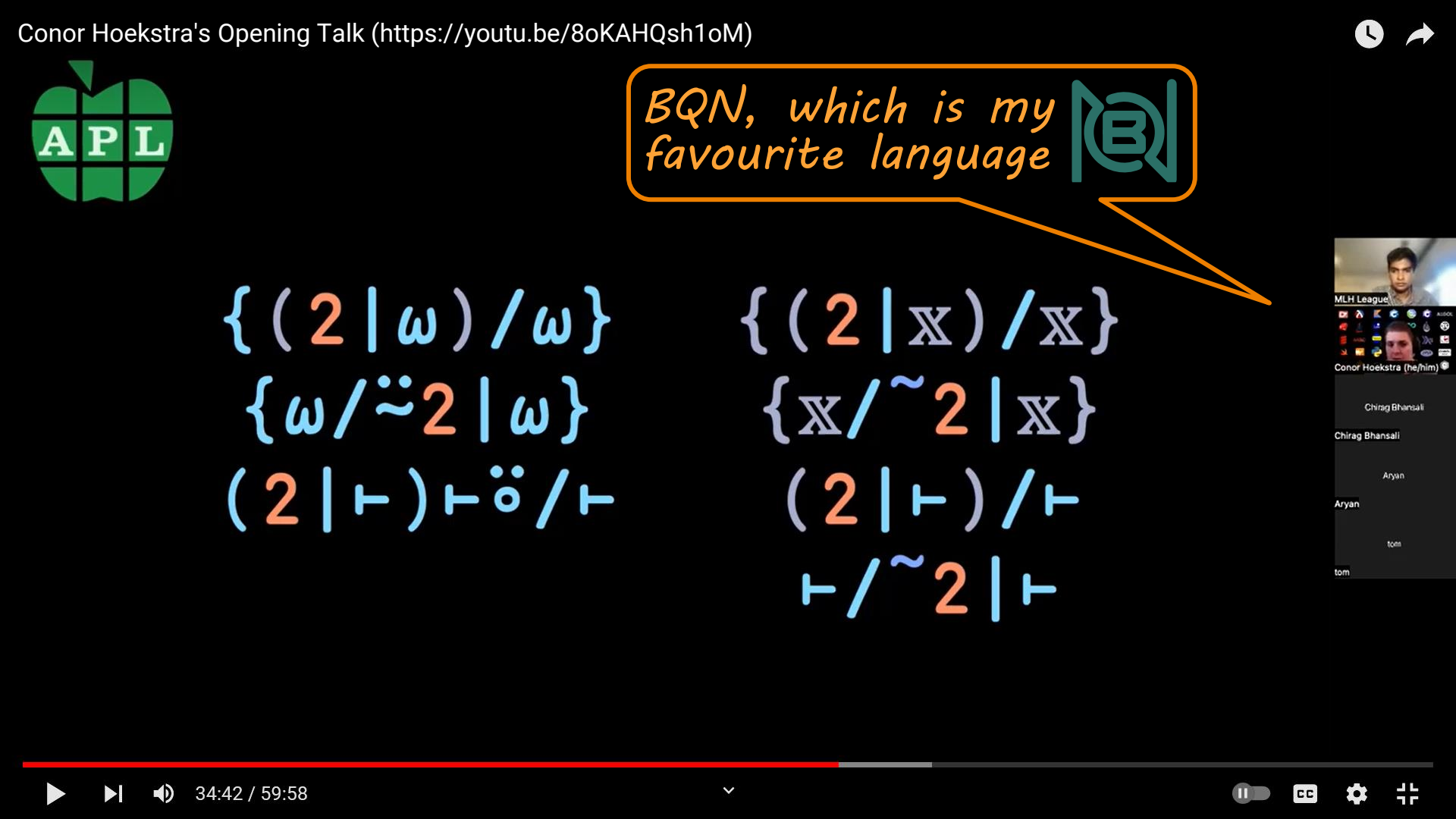
BQN, which is my favourite language

```
{(2|ω)/ω}    {(2|𝕩)/𝕩}
{ω/⁼˜2|ω}    {𝕩/˜2|𝕩}
(2|⊢)⊢⍥⍟/⊢    (2|⊢)/⊢
```

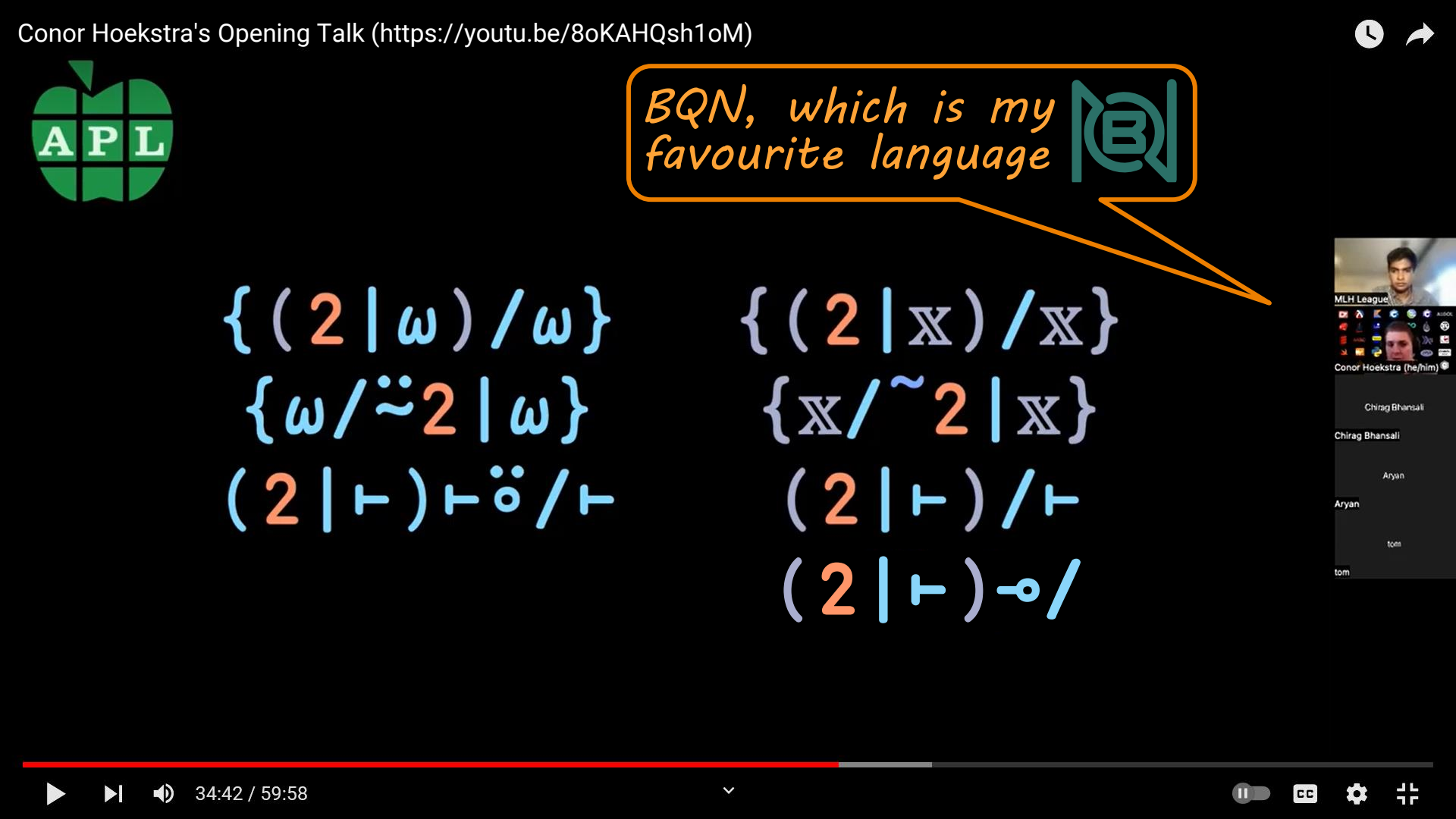


34:42 / 59:58

Conor Hoekstra's Opening Talk (https://youtu.be/8oKAHQsh1oM)



Conor Hoekstra's Opening Talk (https://youtu.be/8oKAHQsh1oM)

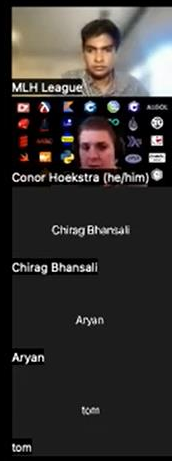


Conor Hoekstra's Opening Talk (https://youtu.be/8oKAHQsh1oM)

BQN, which is my favourite language

```
{(2|ω)/ω}
{ω/⁼2|ω}
(2|⊢)⊢⍥⊏/⊢
```

```
{(2|𝕩)/𝕩}
{𝕩/˜2|𝕩}
(2|⊢)/⊢
(2|⊢)⊸/
2⊸|⊸/
```



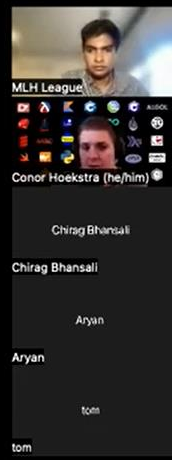
34:42 / 59:58

Conor Hoekstra's Opening Talk (https://youtu.be/8oKAHQsh1oM)

BQN, which is my favourite language

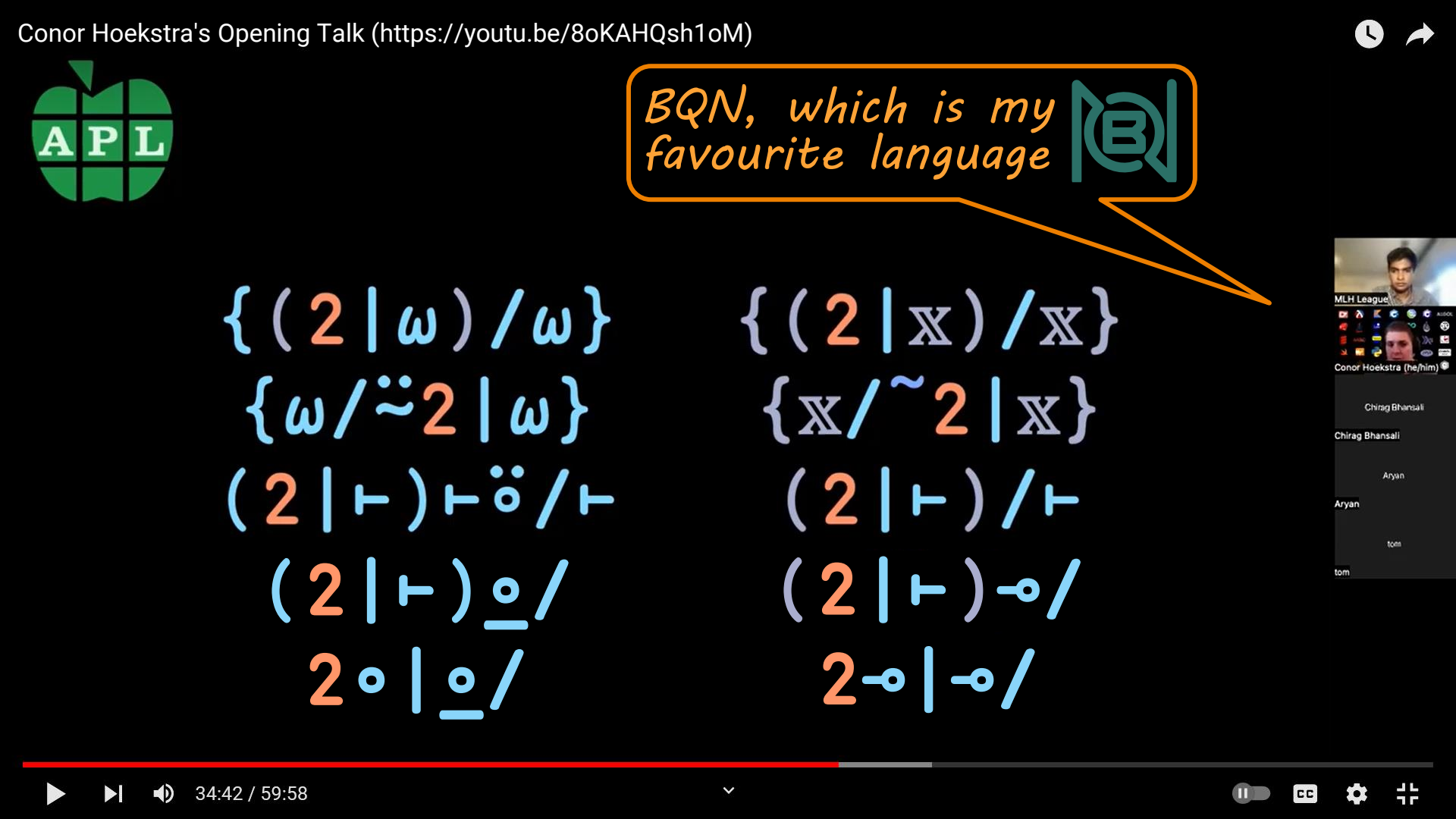
```
{(2|ω)/ω}
{ω/⁼2|ω}
(2|⊢)⊢⍥⊸/⊢
(2|⊢)⍥⊸/
```

```
{(2|𝕩)/𝕩}
{𝕩/˜2|𝕩}
(2|⊢)/⊢
(2|⊢)⊸/
2⊸|⊸/
```



34:42 / 59:58

Conor Hoekstra's Opening Talk (https://youtu.be/8oKAHQsh1oM)



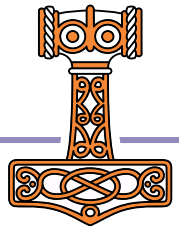
Function Composition

`f⍥g` Behind

Split-compose `X f⍥g∘h Y`

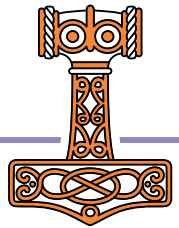
Hybrid mitigation `2∘| ⍥/`

Pre-18.0: `(2∘|{⍺/⍵}⊢)`
18.0: `(2∘|⊢⍤⍥/⊢)`
19.0?: `2∘| ⍥/`



Function Composition

$f \circ g$



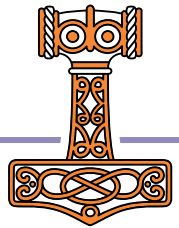
Core Language

Data Transformation

Function Application

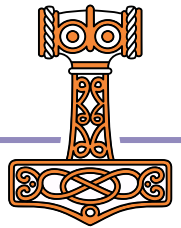
Function Composition

$f \circ g$



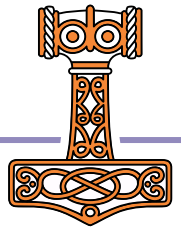
Core Language

Data Transformation	X × Y	ϕ Y	X [] Y	X ⊒ Y
Function Application	f ≠	f ⸭ g	f ⸭ k	f ⸭ k
Function Composition	f ⸭ g	f ⸭ g	f ∘ g	f ∘ g



Core Language

Data Transformation	Select	`Y[X;;]`	`X⊒Y`
Function Application	Depth	`X f⁼⁼⁼⁼⊂⊂Y`	`X f⍤k Y`
Function Composition	Behind	`(f X)g Y`	`X f∘g Y`



Core language

DRAFT PROPOSAL

Data Transformation Select `Y[X;;]` `X⍗Y`

apl.wiki/select

Function Application Depth `X f⁴⁴⁴⁴⊂⊂Y` `X f⍤k Y`

apl.wiki/depth_operator

Want it?

Function Composition Behind `(f X)g Y` `X f⍥g Y`

apl.wiki/behind

Questions?