DYALOG
Eastbourne 2026

Slowly Getting Faster

Asher Harvey-Smith



The short version

- I have been working on the performance of:
  - Stencil (⌺)
  - Set functions (∈, ∪, etc)
  - CSV import (⎕CSV)
  - Matrix multiplication (+ . ×)
  - Array deserialisation

Stencil (⌺)

```
      3 3⍴⎕A
ABC
DEF
GHI
```

```
      {⊂⍵}⌺3 3 3 3⍴⎕A
```

AB DE	ABC DEF	BC EF
AB DE GH	ABC DEF GHI	BC EF HI
DE GH	DEF GHI	EF HI

Stencil (⌺)

- Special cases are optimised
  - `{+/,⍵}⌺`, etc
- General case is written in APL

Stencil (⍰)

```
{
  ⎕io←0
  ⎕ml←1
  (s m)←↓(⍉1 ,⍤¯1)⍤(2>≢⍴⍵⍵)⊢⍵⍵
  ws←(≢s)↑⍴⍵
  n←⌈m÷⍨ws-⍨2|s
  j←⊃∘.,/⊂⍤0⍤1⊢⍤(m×⍳⍤n)∘.+⍤⍳⍤s
  c←⌊2÷⍨s-1
  y←(ws+2×c)↑(-ws+c)↑⍵
  d←0⌈((m×n-1)+s-1)-ws+c-1
  p←0~⍨⍤0⌈c-  m×⍳⍤c
  q←0~⍨⍤0⌊d-⍨⍤⍤m×⍳⍤d
  i←,⍤⍤(1=≢s)⊃∘.,/p,⍤((n-(≢⍤p)+≢⍤q)⍴⍤0),⍤q
  i ⍺⍺{(⊃⍺) ⍺⍺ (⊃⍵)⎕y}⍤0⊢j
}
```

Stencil (⌺)

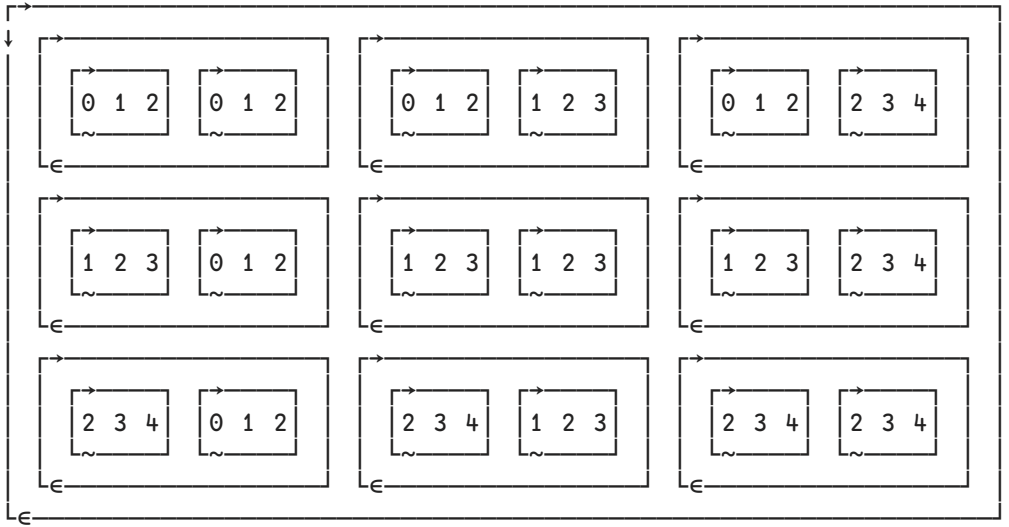
```
{
  ⎕io←0
  ⎕ml←1
  (s m)←↓(⍳1,¨⍤-)⍤(2>≢⍴⍵⍵)⊢⍵⍵
  ws←(≢s)↑⍴⍵
  n←⌈m÷¨ws-⍤2⊢s
  j←⊃∘.,/⊂⍤0¨↓¨(m×⍳¨n)∘.+¨⍳¨s
  c←⌊2÷¨s-1
  y←(ws+2×c)↑(-ws+c)↑⍵
  d←0⌈((m×n-1)+s-1)-ws+c-1
  p←0~¨⍨¨0⌈c-   m×⍳¨c
  q←0~¨⍨¨0⌊d-¨⍤⍣¨m×⍳¨d
  i←,¨⍤⍤(1=≢s)⊃∘.,/p,¨((n-(≢¨p)+≢¨q)⍴¨0),¨q
  i ⍺⍺{(⊃⍺) ⍺⍺ (⊃⍵)⌷y}⍤0⊢j
}
```

```
┌→──────┐
↓1 2 3  │
 4 5 6  │
 7 8 9  │
└~──────┘
```

```
┌→──────────┐
↓0 0 0 0 0  │
 0 1 2 3 0  │
 0 4 5 6 0  │
 0 7 8 9 0  │
 0 0 0 0 0  │
└~──────────┘
```

Stencil (⌹)

```
{
  ⎕io←0
  ⎕ml←1
  (s m)←↓(⍳1,¨¯,)⍤(2>≢⍴⍵⍵)⊢⍵⍵
  ws←(≢s)↑⍴⍵
  n←⌈m÷¨ws-~2|s
  j←⊃∘.,/⊂⍤0¨↓¨(m×⍳¨n)∘.+¨⍳¨s
  c←⌊2÷¨s-1
  y←(ws+2×c)↑(-ws+c)↑⍵
  d←0⌈((m×n-1)+s-1)-ws+c-1
  p←0~¨¨0⌈c-   m×⍳¨c
  q←0~¨¨0⌊d-¨⍤¨m×⍳¨d
  i←,¨⍤(1=≢s)⊃∘.,/p,¨((n-(≢¨p)+≢¨q)⍴¨0),¨q
  i ⍺⍺{(⊃⍺) ⍺⍺ (⊃⍵)⌷y}⍤0⊢j
}
```



Stencil (⍰)

```
{
  ⎕io←0
  ⎕ml←1
  (s m)←↓(⍉1 ,⍤¯1)⍤(2>≢⍴⍵⍵)⊢⍵⍵
  ws←(≢s)↑⍴⍵
  n←⌈m÷⍨ws-~2|s
  j←⊃∘.,/⊂⍤0⍤⍳⍤(m×⍳⍤n)∘.+⍤⍳⍤s
  c←⌊2÷⍨s-1
  y←(ws+2×c)↑(-ws+c)↑⍵
  d←0⌈((m×n-1)+s-1)-ws+c-1
  p←0~⍨⍤0⌈c-   m×⍳⍤c
  q←0~⍨⍤0⌊d-⍨⍤⍤m×⍳⍤d
  i←,⍤⍤(1=≢s)⊃∘.,/p,⍤((n-(≢⍤p)+≢⍤q)⍴⍤0),⍤q
  i ⍺⍺{(⊃⍺) ⍺⍺ (⊃⍵)⎕⍵}⍤0⊢j
}
```

Stencil (⍰)

- Write the general case in C
- Avoid interpreter overhead
- Avoid allocating intermediate values
- Document the algorithm for non-APLers

Stencil (⌧)

Mind Boggling Performance

Posted on March 17, 2026 by Brian Becker

Or is it Minding Boggle Performance?

Better late than never? This was a blog post I started to write during COVID-19 and now I've finally gotten around to finishing it.

In the 2019 APL Problem Solving Competition, we presented a problem to solve the Boggle game . In Boggle, a player tries to make as many words as possible from contiguous letters in a 4×4



Stencil (⌺)

```
[1] inds←⍳⍴board
[2] neighbors←(,inds)∘∩¨↓inds∘.+(,¯2+⍳3 3)~⊂0 0
```

“You might recognize line [2] as a *stencil*-like (⌺) operation. [...] As it turns out, for this application, **stencil is slower**.”

```
{⊂(,α↓ω)~ω[2;2]}⌺3 3⊢inds
```

Stencil (⍁)

```
i←⍳4 4
```

Unoptimised:

```
(,i)∘∩⍤↓i∘.+(,¯2+⍳3 3)~⊂0 0 → 2.0E¯5 |     0% ⎕⎕⎕⎕⎕⎕⎕
{⊂(,α↓ω)~ω[2;2]}(⍁3 3)i      → 4.3E¯5 | +113% ⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕
```

Optimised:

```
(,i)∘∩⍤↓i∘.+(,¯2+⍳3 3)~⊂0 0 → 2.0E¯5 |     0% ⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕
{⊂(,α↓ω)~ω[2;2]}(⍁3 3)i      → 1.3E¯5 |  -38% ⎕⎕⎕⎕⎕⎕⎕⎕⎕
```

Stencil (⍁)

```
i←⍳100 100

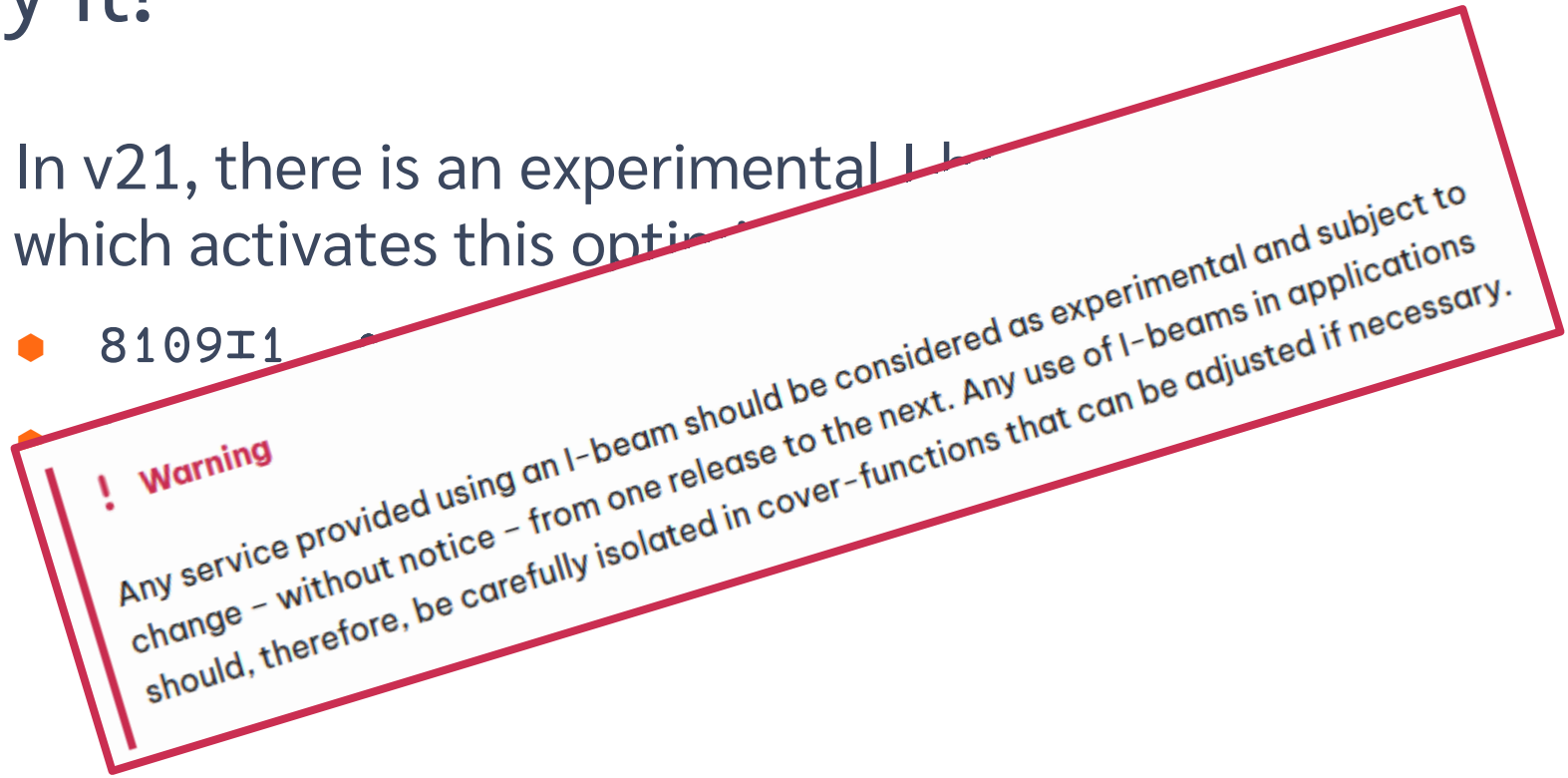
(,i)∘⍉¨↓i∘.+(,¯2+⍳3 3)~⊂0 0 → 3.8E0   |     0% ⬚⬚⬚⬚⬚⬚⬚⬚⬚⬚⬚⬚⬚⬚⬚⬚
{⊂(,⍺↓⍵)~⍵[2;2]}(⍁3 3)i      → 1.8E¯2 | -100%
```

Try it!

- In v21, there is an experimental I-beam which activates this optimisation
  - `8109⌶1` ⍝ switch it on
  - `8109⌶0` ⍝ switch it off

Try it!

- In v21, there is an experimental l
  which activates this opti
  - `8109I1`

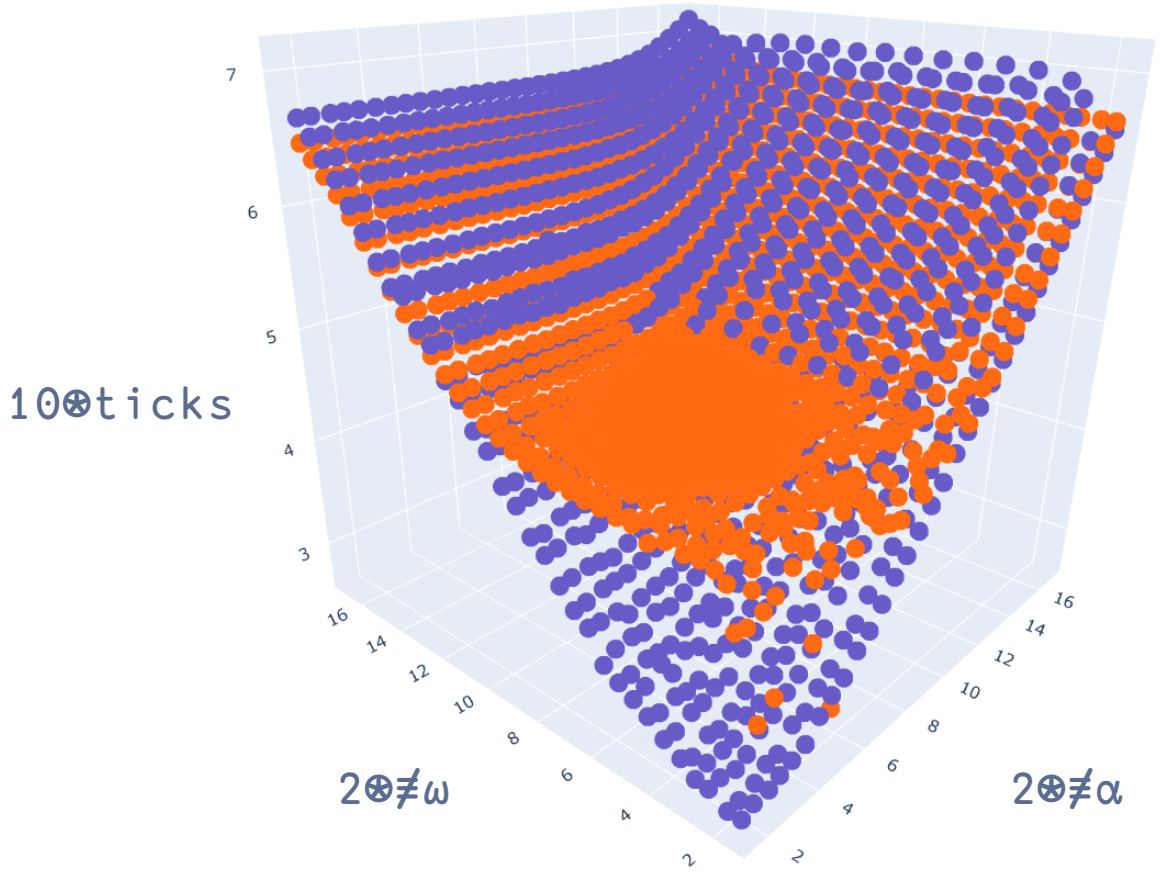


What’s cooking?

- ~~Stencil (⌺)~~
- Set functions (∈, ∪, etc)
- CSV import (⍞CSV)
- Matrix multiplication (+.×)
- Array deserialisation

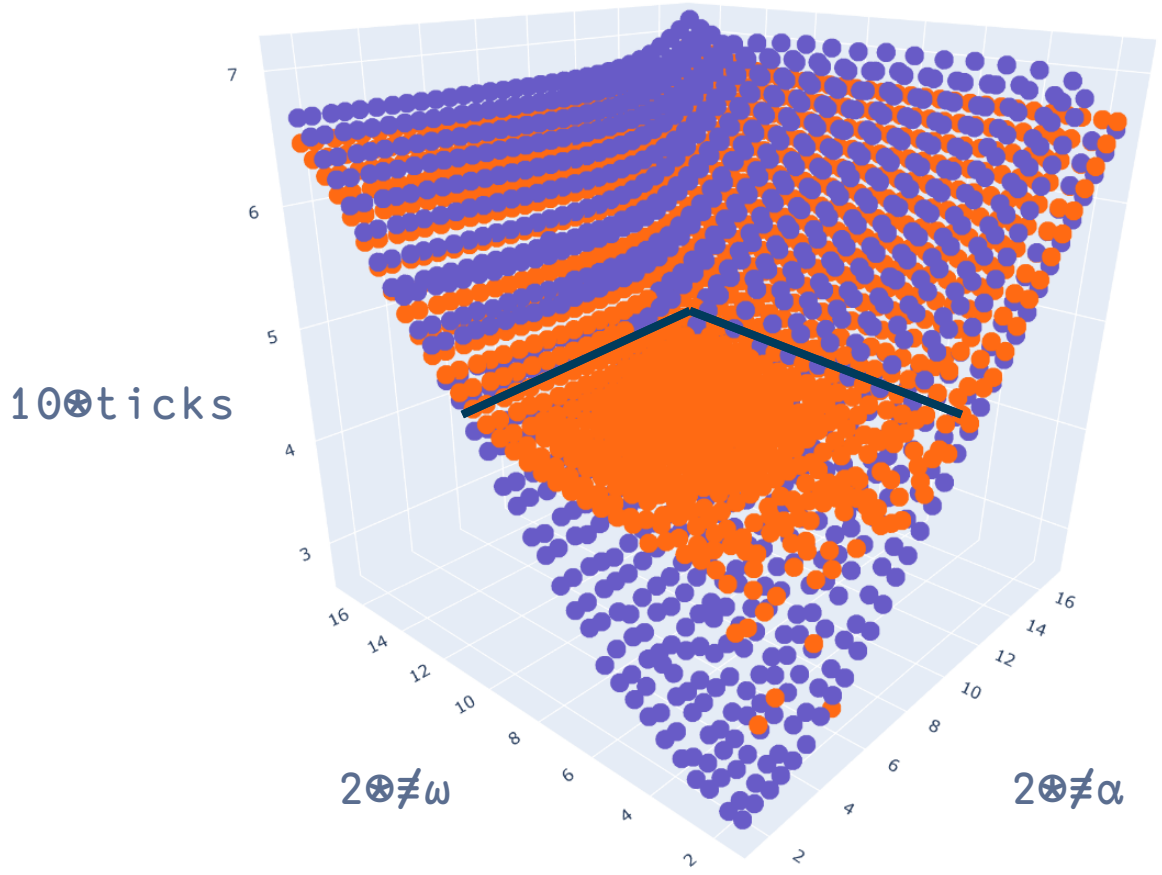
To lookup or to hash?

- To execute the set functions (∈, ∪, etc) the interpreter builds tables
- Which *kind* of table depends on the situation



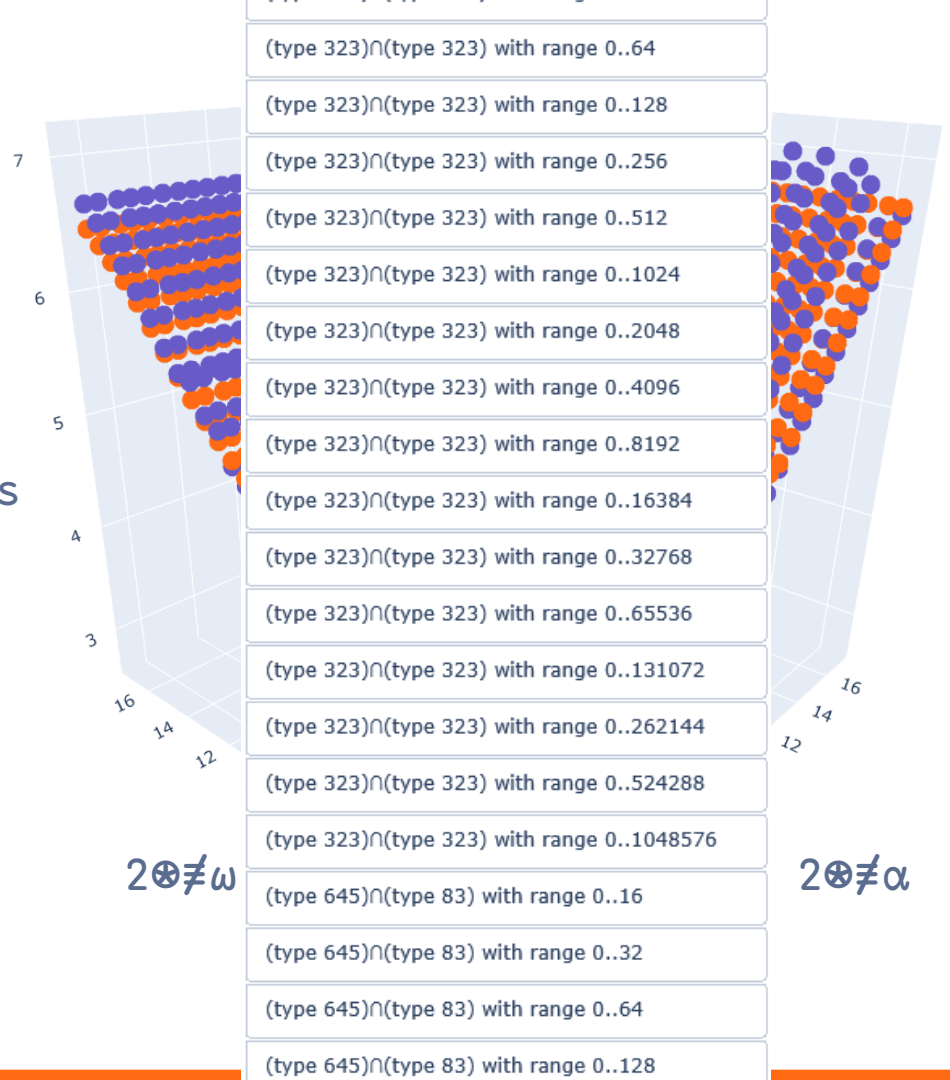
- Lookup table
- Hash table

Slowly Getting Faster



- Lookup table
- Hash table

Slowly Getting Faster



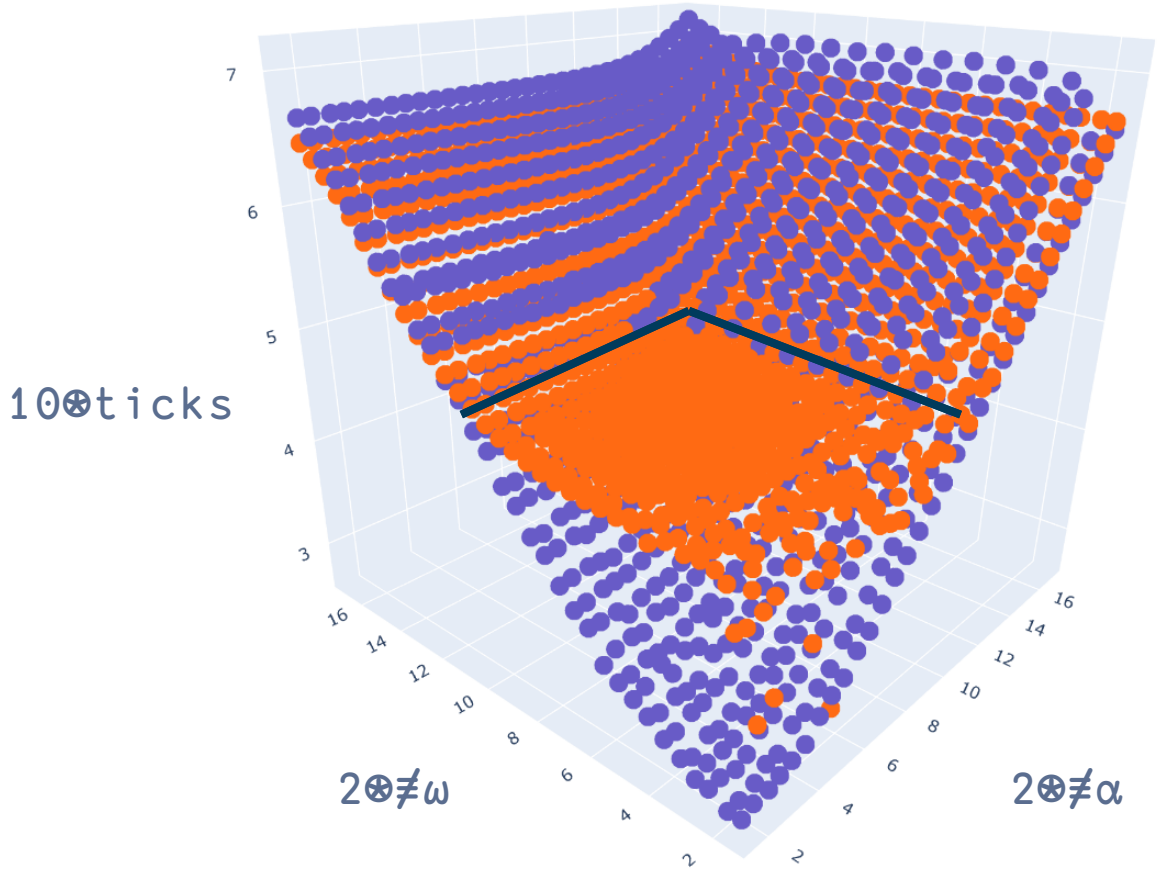
- Lookup table
- Hash table

Learning the boundary

- Based on parameters (array size, type, range), should we use a lookup table?
- Sounds like a logistic regression problem!

Learning the boundary

```
 2.9139   1.0664        0   -0.9579
 0.6245        0   1.7320   -1.2008
 2.0608   1.4608        0   -1.2244
 0.2911        0   1.3353   -0.9605
 2.2082   1.4739        0   -1.1737
-0.0353        0   1.5757   -1.0753
 2.4179   1.4169        0   -1.2149
 0.7878        0   1.6099   -1.2145
 1.7297   0.7576        0   -0.7023
 2.3594        0   1.2155   -0.9427
 2.2259   0.9155        0   -0.8793
 2.1737        0   1.4003   -1.1017
```



- Lookup table
- Hash table

Slowly Getting Faster

What’s cooking?

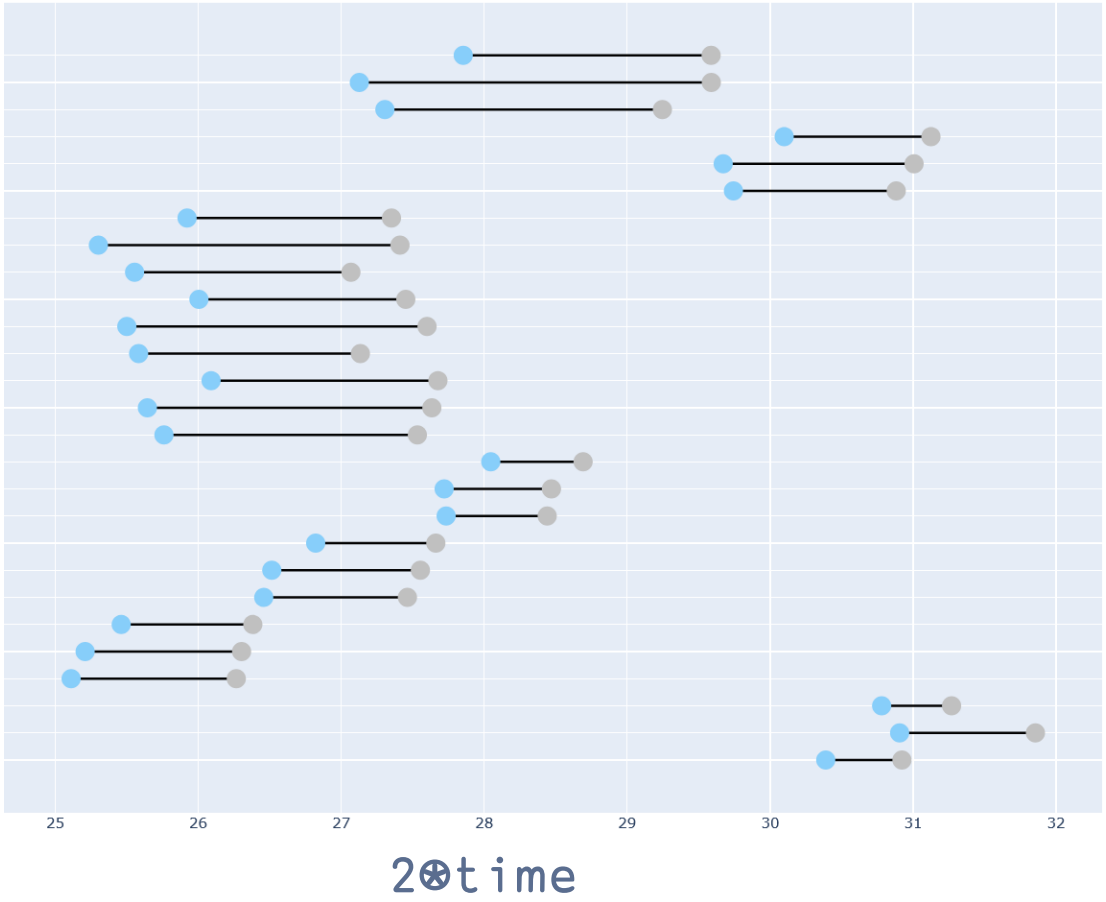
- ~~Stencil (⌺)~~
- ~~Set functions (∈, ∪, etc)~~
- CSV import (⎕CSV)
- Matrix multiplication (+ . ×)
- Array deserialisation

CSV has a lot of options

- Source – file name, tie number, vector, vector of vectors
- Description – encoding or code-page
- Column types – ignored, character, numeric (4 ways to handle bad values)
- Header indicator – Yes/no
- Decimal mark – `'.'` or `','`
- Double quotes – Yes/no
- Escape character – What, if any
- Invert – Which format, if any
- Fill value
- Quote character – What, if any
- Ragged – Yes/no
- Separator
- Thousands – What, if any
- Trim – Yes/no
- Widths – What, if any

CSV import (CSV)

- CSV needs to handle all these options, which has a performance cost
- It would be nice to use special code which only handles what it needs to
- I tried this, and it's ~2× as fast as the existing CSV code



Old implementation

New implementation

Slowly Getting Faster

CSV import (□CSV)

- We can do even better than that!
  - I think an order of magnitude is possible
- Slowest parts are now:
  - Parsing numbers
  - File reads

What’s cooking?

- ~~Stencil (⌺)~~
- ~~Set functions (∈, ∪, etc)~~
- ~~CSV import (⎕CSV)~~
- Matrix multiplication (+ . ×)
- Array deserialisation

Matrix multiplication (+ . ×)

- We already have a fairly fast + . ×
- There has been a **lot** of research into fast matrix multiplication
- These researchers have kindly made their work open-source

Matrix multiplication (+ . ×)

```
      F←{x+.×y⊣NNNNI⍵}

      x←1E6×?200 200⍴0
      y←1E6×?200 200⍴0

      cmpx 'F 0' 'F 1'
F 0 → 3.0E¯3 |   0% ■■■■■■■■■■■■■■■■■■■■■■
F 1 → 2.3E¯4 | -93% ■
```

What’s cooking?

- ~~Stencil (⌺)~~
- ~~Set functions (∈, ∪, etc)~~
- ~~CSV import (⍞CSV)~~
- ~~Matrix multiplication (+ . ×)~~
- Array deserialisation

In v20 we got Array Notation

```
('vector' ◇ 1 2 3)

['high'
 'rank'
 'array']

(name: 'space' ◇ ⎕IO: 0)
```

Array Notation Import/Export

- In a future version, the interpreter will expose ⎕REPR and ⎕EVAL
- This will speed up applications using array notation as a serialisation format
  - Like Link!

```
(
 (
  'rowid'
  'species'
  'island'
  'bill_length_mm'
  'bill_depth_mm'
  'flipper_length_mm'
  'body_mass_g'
  'sex'
  'year'
 )
 [
  ('1'◇) 'Adelie' 'Torgersen' '39.1' '18.7' '181' '3750' 'male'   '2007'
  ('2'◇) 'Adelie' 'Torgersen' '39.5' '17.4' '186' '3800' 'female' '2007'
  ('3'◇) 'Adelie' 'Torgersen' '40.3' '18'   '195' '3250' 'female' '2007'
  ('4'◇) 'Adelie' 'Torgersen' 'NA'   'NA'   'NA'  'NA'   'NA'     '2007'
  ('5'◇) 'Adelie' 'Torgersen' '36.7' '19.3' '193' '3450' 'female' '2007'
  ('6'◇) 'Adelie' 'Torgersen' '39.3' '20.6' '190' '3650' 'male'   '2007'
  ('7'◇) 'Adelie' 'Torgersen' '38.9' '17.8' '181' '3625' 'female' '2007'
  ('8'◇) 'Adelie' 'Torgersen' '39.2' '19.6' '195' '4675' 'male'   '2007'
  ('9'◇) 'Adelie' 'Torgersen' '34.1' '18.1' '193' '3475' 'NA'     '2007'
  '10'   'Adelie' 'Torgersen' '42'   '20.2' '190' '4250' 'NA'     '2007'
  '11'   'Adelie' 'Torgersen' '37.8' '17.1' '186' '3300' 'NA'     '2007'
  …
```

```
      Deserialise←⎕SE.Dyalog.Array.Deserialise

      cmpx 'NNNNItxt' '⍎txt' 'Deserialise txt'
NNNNItxt         → 1.6E¯4 |     0% ⎕
⍎txt             → 1.0E¯3 |  +534% ⎕⎕⎕⎕⎕
Deserialise txt → 3.5E¯3 | +2076% ⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕
```


Slowly Getting Faster

What’s cooking?

- Stencil (⌺)
- Set functions (∈, ∪, etc)
- CSV import (⍓CSV)
- Matrix multiplication (+.×)
- Array deserialisation

Thank you!

- Stencil (⌺)
- Set functions (∈, ∪, etc)
- CSV import (⍞CSV)
- Matrix multiplication (+.×)
- Array deserialisation

```
LR←{
    ⍝ Logistic regression via gradient descent
    ⍝ α: Optional number of iterations for gradient descent
    ⍝ ω: Pair of
    ⍝    1. Matrix of features: nfeatures×nsamples
    ⍝    2. Boolean vector of labels
    ⍝ ←: Vector of weights (including a constant intercept)

    ⍝ Arguments
    α←1E4     ⍝ default number of steps
    n←α       ⍝ number of steps
    e←1       ⍝ initial step size
    (x y)←ω   ⍝ features and labels
    x←1⍪x     ⍝ include a constant row

    ⍝ Log-Likelihood to be maximised
    ⍝ ny and sy extracted for performance
    ny←~y
    sy←¯1*ny
    LL←{+/⍟ny+sy×÷1+*-ω+.×x}

    ⍝ One gradient descent step
    ⍝ Step sizes are controlled with backtracking line search and the Armijo condition
    GD←{
        11::ω                                    ⍝ on overflow, pass through last good result
        th←ω                                     ⍝ parameters so far
        g←+/x(×⍤1)y-÷1+*-th+.×x                  ⍝ gradient of LL here
        Armijo←{                                 ⍝ does step size improve LL enough?
            11::0                                ⍝  if we overflow, then no
            (LL th+ω×g)≥(LL th)+0.0001×ω×g+.×g
        }
        Armijo e: th+e×g                         ⍝ if so, done with new parameters
        e⊢←0.5×⍣Armijo e                         ⍝ else, half it until it does
        th+e×g                                   ⍝ new parameters
    }

    ⍝ Perform n steps of gradient descent
    (GD⍣n)0/⍤≠x
}
```


Slowly Getting Faster