DYALOG
Glasgow 2016

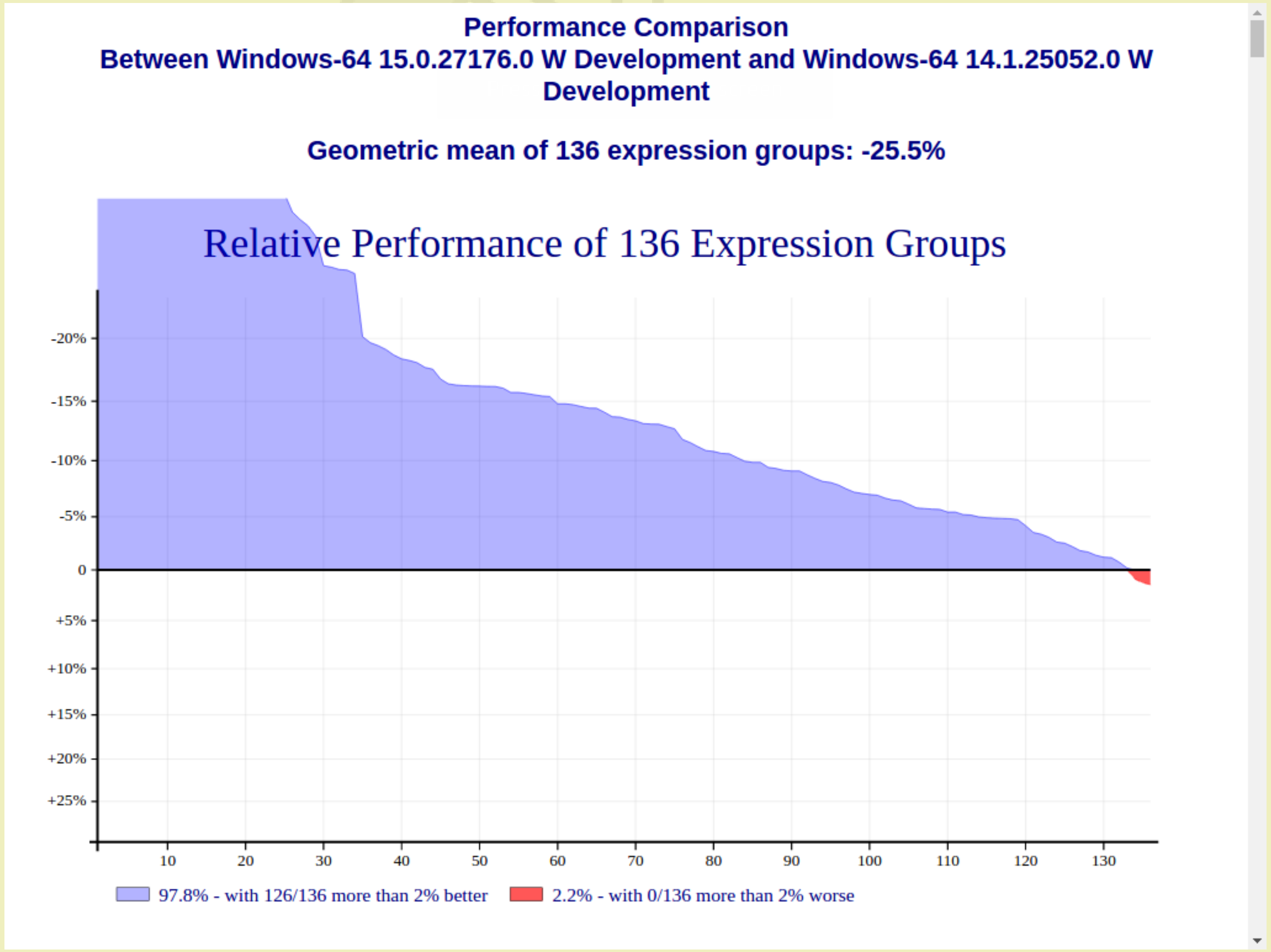
# Performance: The Neverending Story

Jay Foad

## Agenda

- Version 15.0
- Version 16.0
- … and beyond!

## Version 15.0



Performance Comparison
Between Windows-64 15.0.27176.0 W Development and Windows-64 14.1.25052.0 W Development

Geometric mean of 136 expression groups: -25.5%

## Version 15.0

- PQA graphs look better than ever (best increase we have ever *measured*)
- Due to a combination of:
  - C compiler upgrades
  - Lots of individual optimisations
- Also occasional new performance features
  - E.g. `8⌶` (Inverted table index of) in version 14.1

## Version 15.0 Hashed arrays

- I-beam to mark an array as a potential and likely left argument to dyadic `⍳` (and the other set functions)
- Better than the old `A∘⍳` system
- Hash table is updated by:
  - Append idiom `,←`
  - Chop idiom `↓⍨←`

## Version 15.0 Hashed arrays

Old way:

```
f ← A∘⍳
f x ⋄ f y ⋄ f z
```

New way:

```
B ← 1500⌶ A
B⍳x ⋄ y∊B ⋄ ∪B
B ,← ⍳10 ⋄ B ↓⍨← ¯5
```

## Version 15.0 Chop idiom

- Fastest way of trimming a vector
- Works in place (like the append idiom)
- Also works on leading axis of any array

```
vec ↓⍨← ¯2   ⍝ chop last 2 items
mat ↓⍨← ¯3   ⍝ chop last 3 rows
```

## Version 16.0

- Random bits
- Namespace refs
- Selective assignment
- Boolean algorithms
- DECF representation

# Version 16.0 Random bits

Previously:

```
      ⎕IO←0
      cmpx'?1E6⍴2'
  ?1E6⍴2    → 4.5E¯3 |   0% ⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕
```

New default and optimisations in version 16.0:

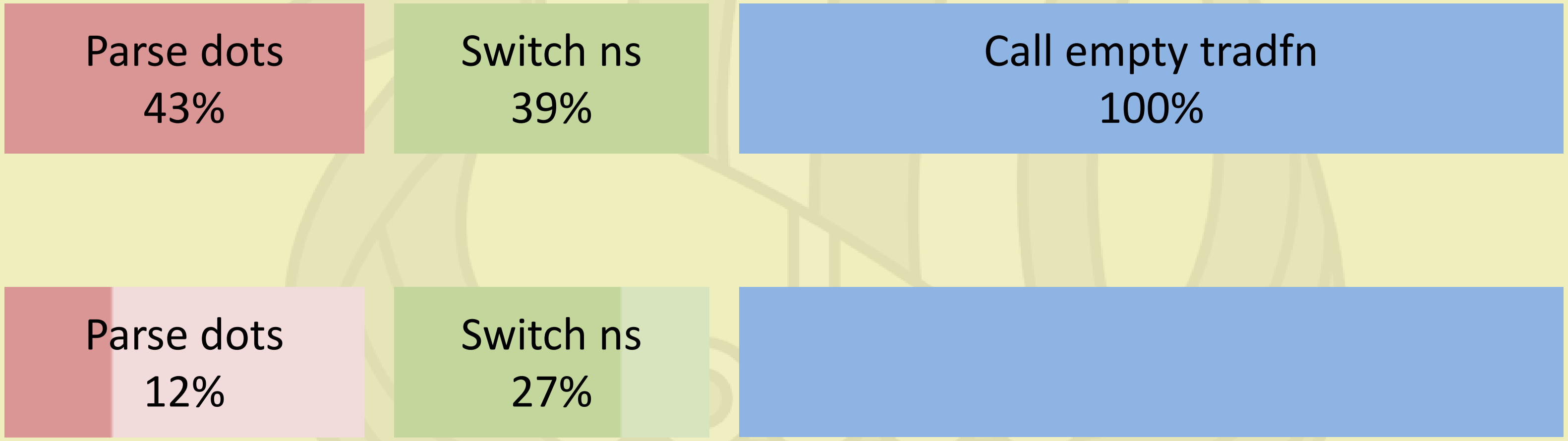
```
      ⎕RL←⍬
      cmpx'?1E6⍴2' '1E6(?⍴)2'
  ?1E6⍴2    → 2.1E¯4 |    0% ⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕
* 1E6(?⍴)2 → 7.0E¯5 | -68% ⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕⎕
```

## Version 16.0 Namespace refs

Calling a function *in a namespace*

```
      ns.foo 99
```

has an 82% penalty



Penalty reduced to 39%

# Version 16.0 Selective assignment

Selective assignment is not an efficient way to modify a few items in a large array A:

```
        (2↑A)←99
   ((⊂2 4)⌷A)←99
```

... because we generate an index array for the whole of A.
(Factor of 2 when A has 1000 items.
 Factor of 1000 when A has 1E6 items.)

This has been fixed for Squad ⌷ indexing
We hope to fix it for Take/Drop ↑↓ and Compress Bool/
Maybe others, as time permits

## Version 16.0 Boolean algorithms

Coming next…

(U08) A Compendium of SIMD Boolean Array Algorithms in APL

Robert Bernecky (Snake Island Research)

Word-at-a-time algorithms for =\ and ≠\

```
      {⍵/⍨q∨≠\q←⍵='"'} 'Bob "SIMD" Bernecky'
"SIMD"
```

## Version 16.0 DECF representation

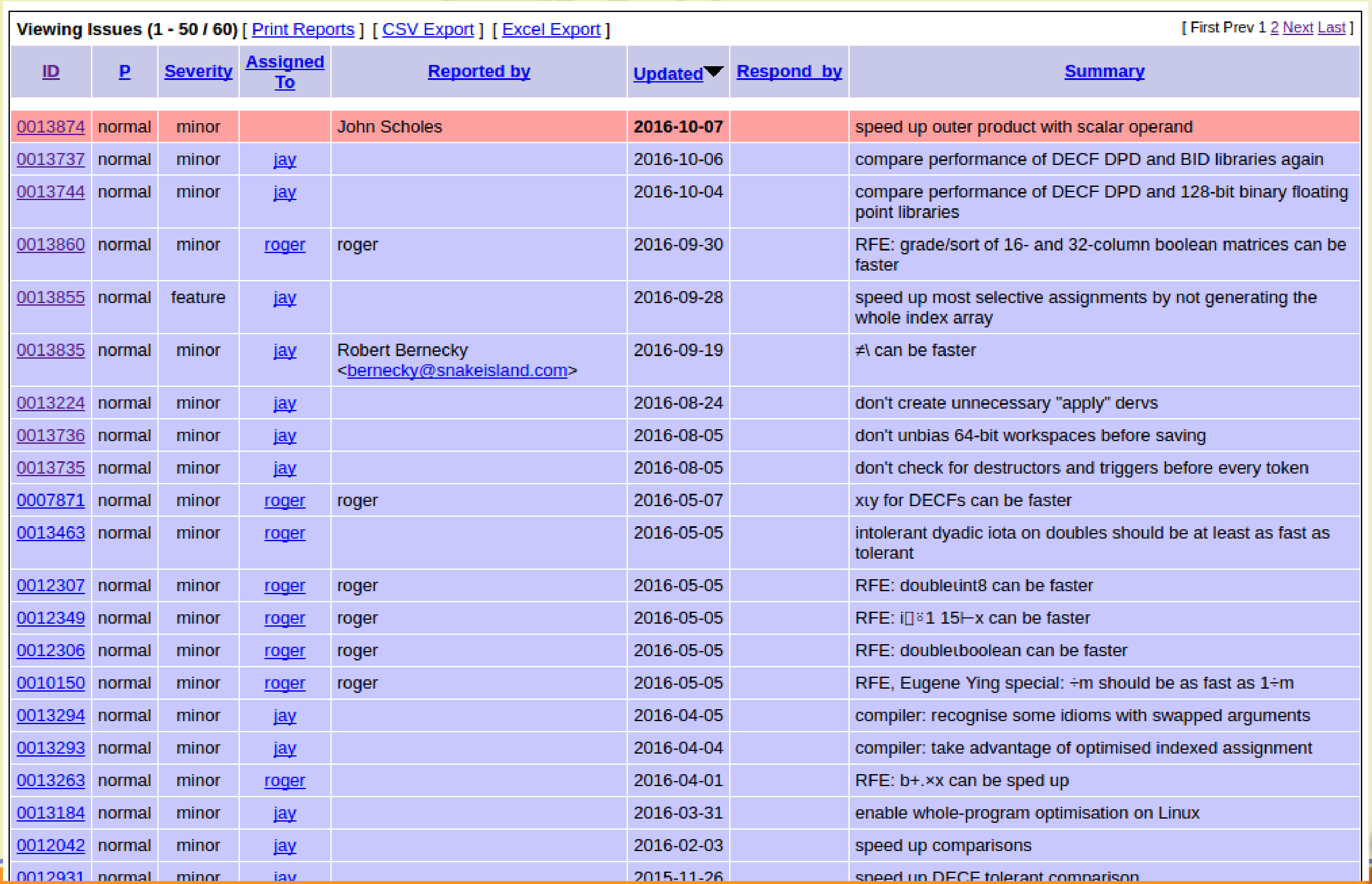
128-bit Decimal floating point

- Current representation is DPD:
  good for formatting
- Alternative is BID:
  good for calculations (2x faster)

Or we could do 128-bit *Binary* floating point

(another 2x faster for calculations)

# The future



Viewing Issues (1 - 50 / 60) [ Print Reports ] [ CSV Export ] [ Excel Export ] [ First Prev 1 2 Next Last ]

| ID | P | Severity | Assigned To | Reported by | Updated | Respond by | Summary |
|---|---|---|---|---|---|---|---|
| 0013874 | normal | minor | | John Scholes | **2016-10-07** | | speed up outer product with scalar operand |
| 0013737 | normal | minor | jay | | 2016-10-06 | | compare performance of DECF DPD and BID libraries again |
| 0013744 | normal | minor | jay | | 2016-10-04 | | compare performance of DECF DPD and 128-bit binary floating point libraries |
| 0013860 | normal | minor | roger | roger | 2016-09-30 | | RFE: grade/sort of 16- and 32-column boolean matrices can be faster |
| 0013855 | normal | feature | jay | | 2016-09-28 | | speed up most selective assignments by not generating the whole index array |
| 0013835 | normal | minor | jay | Robert Bernecky <bernecky@snakeisland.com> | 2016-09-19 | | ≠\ can be faster |
| 0013224 | normal | minor | jay | | 2016-08-24 | | don't create unnecessary "apply" dervs |
| 0013736 | normal | minor | jay | | 2016-08-05 | | don't unbias 64-bit workspaces before saving |
| 0013735 | normal | minor | jay | | 2016-08-05 | | don't check for destructors and triggers before every token |
| 0007871 | normal | minor | roger | roger | 2016-05-07 | | x⍳y for DECFs can be faster |
| 0013463 | normal | minor | roger | | 2016-05-05 | | intolerant dyadic iota on doubles should be at least as fast as tolerant |
| 0012307 | normal | minor | roger | roger | 2016-05-05 | | RFE: double⍳int8 can be faster |
| 0012349 | normal | minor | roger | roger | 2016-05-05 | | RFE: i⎕⍤1 15⊢x can be faster |
| 0012306 | normal | minor | roger | roger | 2016-05-05 | | RFE: double⍳boolean can be faster |
| 0010150 | normal | minor | roger | roger | 2016-05-05 | | RFE, Eugene Ying special: ÷m should be as fast as 1÷m |
| 0013294 | normal | minor | jay | | 2016-04-05 | | compiler: recognise some idioms with swapped arguments |
| 0013293 | normal | minor | jay | | 2016-04-04 | | compiler: take advantage of optimised indexed assignment |
| 0013263 | normal | minor | roger | | 2016-04-01 | | RFE: b+.×x can be sped up |
| 0013184 | normal | minor | jay | | 2016-03-31 | | enable whole-program optimisation on Linux |
| 0012042 | normal | minor | jay | | 2016-02-03 | | speed up comparisons |
| 0012931 | normal | minor | jay | | 2015-11-26 | | speed up DECF tolerant comparison |

## The future

- No shortage of work for Roger
- Squeeze more out of the C compilers
- More use of modern SIMD instructions (AVX2, POWER8)
- More to be done on namespace refs and similar targetted speed-ups