Living in Copious* Data with Vector Functional Programming

And Now For Something Completely Different

Dave Thomas

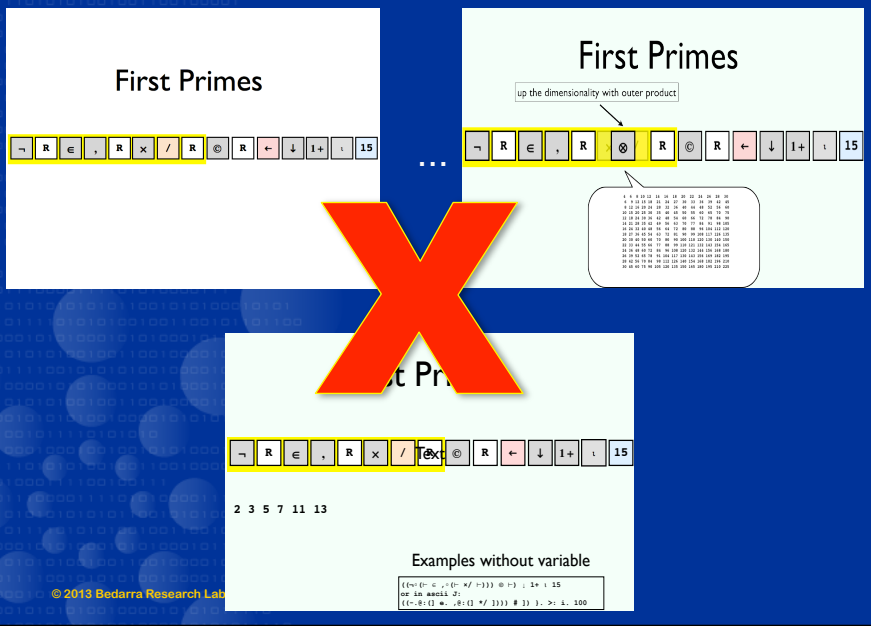
www.davethomas.net

■ Bedarra Research Labs ■ GOTO/YOW! Conferences

■ Carleton University ■ Queensland University of Technology ■

* Dean Wampler

Outline

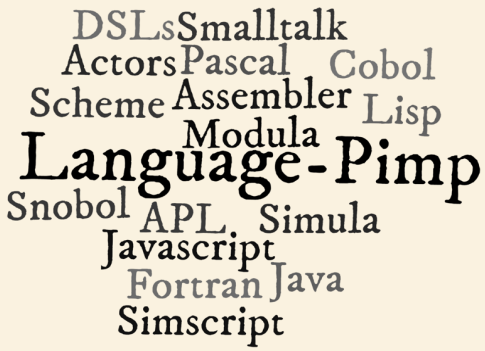


Outline

1. Big Data for Thinkers (Motivation)
2. Why we decided on Vector FP?
3. Copious Data – Oh! No YES! SQL
4. A Taste of Vector FP in q

Only Slightly Functional Old Country Programmer

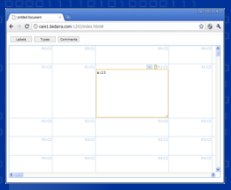
Confessed Objectaholic and Imperative Stateful! Sinner

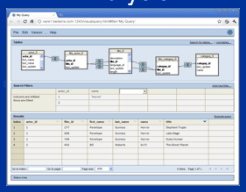


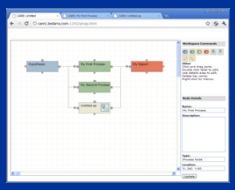
26	COBOL	0.441%
27	Fortran	0.414%
28	Scheme	0.378%
29	D	0.364%
30	Erlang	0.349%
31	Haskell	0.330%
32	Logo	0.327%
33	Scala	0.326%
34	Prolog	0.316%
35	Scratch	0.295%
36	Tcl	0.245%
37	F#	0.238%
38	NXT-G	0.232%
39	Smalltalk	0.228%
40	APL	0.226%
41	C shell	0.223%
42	Go	0.222%
43	Forth	0.214%

Motivation - Tools for Thinkers

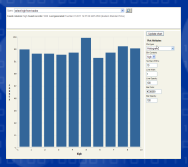
Analysis

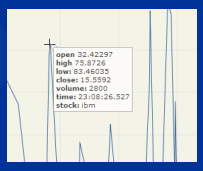


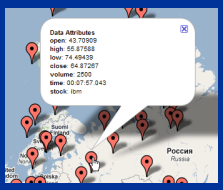


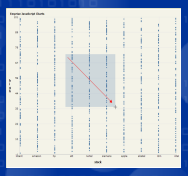


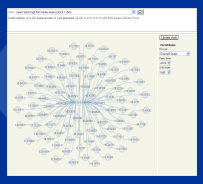
Results











Thinker Data

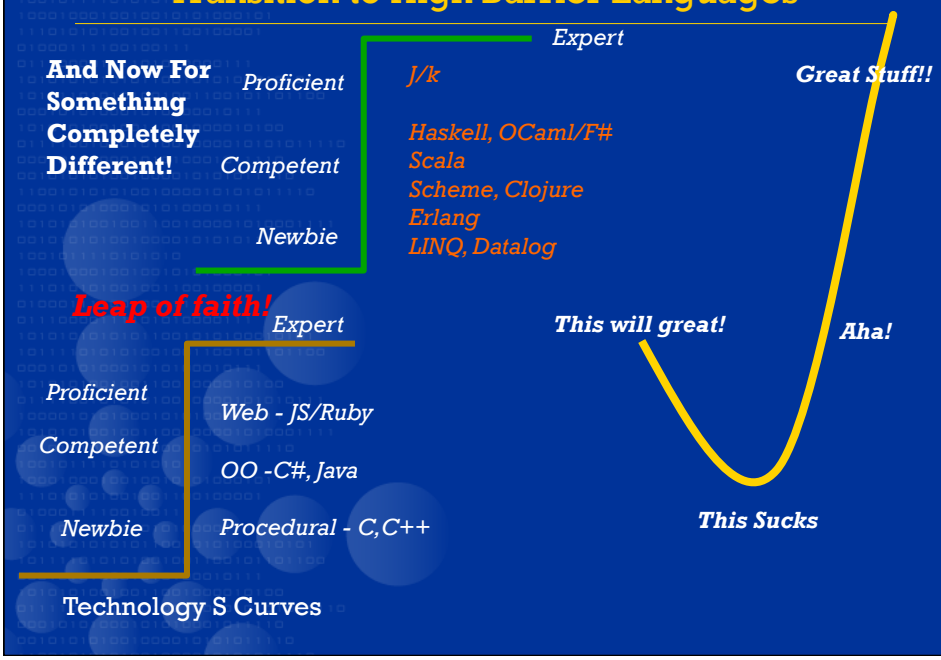
- Small (sensors) to Large Terabytes to Petabytes
- Semi Structured and Mixed Media
- Temporal (Nanoseconds to Decades) and Times Intervals
- Missing Data
- Out of Range Data – sensor failure, errors
- Uncertain Data (likely, highly unlikely)

Data Intensive Computing

All roads lead to some form of Functional CRUD

- Applied Functional Programming (aka Super CRUD)
- SQL + Functions + Streams – e.g. Greenplum …
- NoSQL Databases – Dictionaries on Steroids (Big Table, CouchDB…)
- Map Reduce Data Parallel
- Hybrid JVM, CLR/LINQ functional languages F#, Scala, Clojure
- Vector Functional Programming
- Reactive Programming Rx, FlapJack
- Haskell Array Fusion

Transition to High Barrier Languages



Top Down Data Language Design Choices

1. A few elegant and simple abstractions
2. A dynamic object model and garbage collector
3. Everything is an object (list , set)…

!! the first N < 5 implementations will suck in space and time

!! interop with native HW will trail HW

!! implementers will spend decades trying to make elegant => fast

4. Language is extended by libraries in the same language

!! the libraries will often be bloated and of variable quality, with changing APIs …

!! new libraries will interact with JIT runtime

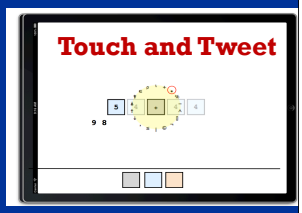
Bottom Up Data Language Design Choices

1. Needs to be fast !	Hence needs to be close to the metal in terms of runtime types and data structures
2. Needs to be small (compact)	
3. Needs basic safety	Hence must pay for nulls, index range checking…
4. Needs to support massive data	Hence needs to be value versus reference based and needs to support data parallel and simple actor concurrency
5. Needs scalable concurrency	
6. It will be a challenge to design a normal developer language (i.e. the GPU problem)	Hence needs an expert language and DSLs for specific application users

Why Functional Vector VMs vs Object VMs

- Array VMs vs. OVMs
  - Everything is boxed - No need for boxing and unboxing! … Simpler GC
  - Support for all native machine types
  - Virtual machine is smaller… can easily be held in instruction and data caches
  - Values are shared until modified
- Arrays are Column Stores => Table is a set of columns
  - Reduces the impedance between Objects and Records
  - Vectors are trivially and efficiently serialized
  - Vectors are machine values
  - Vector operations stream data through caches
- Array libraries use efficient algorithms code at machine level
- Arrays take less space than object collections
- Data parallelism is easily implemented

A Programming Language (APL) – Ken Iverson[1]



Extremely Concise Idiomatic[2,3] Programs (one liners)

Life is an array `life←{↑1 ⍵∨.∧3 4=+/,¯1 0 1∘.⊖¯1 0 1∘.⌽⊂⍵}`

Efficient generalized array operators e.g. +/ "plus reduce"; +.* …

Applicative Operator Style with Monadic and Dyadic Functions with "No Stinking Loops" [4]

1. Ken Iverson, Notational as a Tool for Thought, 1979 ACM Turning Award
2. Alan Perlis, In Praise of APL – A Language for Lyrical Programming, SIAM News, 1977 and Programming with Idioms in APL, APL Conference, 1979
3. D.B McIntyre, Language as an Intellectual Tool: From Hieroglyphics to APL, IBM 1991
4. Steve Apter, nostinkingloops.com

The Vector (Array) System

APL

Nial, A+
APL2

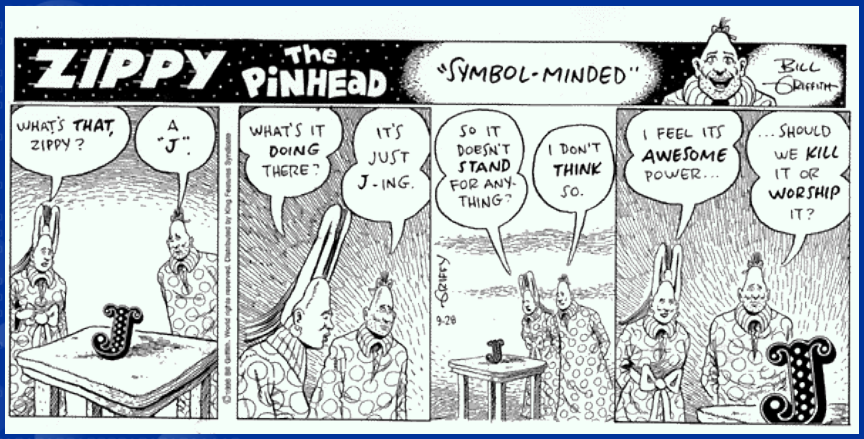
J

k

- APL – Array Programming – K. Iverson
- Nested Array Theory – T. Moore - APL2, Q'NIAL , A+
- Dyalog APL, APL 2000 – Unix and .NET - *R, Matlab, PyNumerics, Octave..*
- J - Vector FP "CISC" language
- k, k2+ksql, k3/kdb, k4/kdb+...
- Vector FP "RISC" language, q financial DSL for kdb+
- Julia

"Chasing Men That Who Stare Arrays" by Catherine Lathwell

Lyrical, Idiomatic, Shape Shifters, Transformers, Iron Chef, AlienCoders, Puzzlers, Oneliners, LineNoise Genertors, Anti Loop Conspirators, Secret Society …



Common Features of Vector Languages

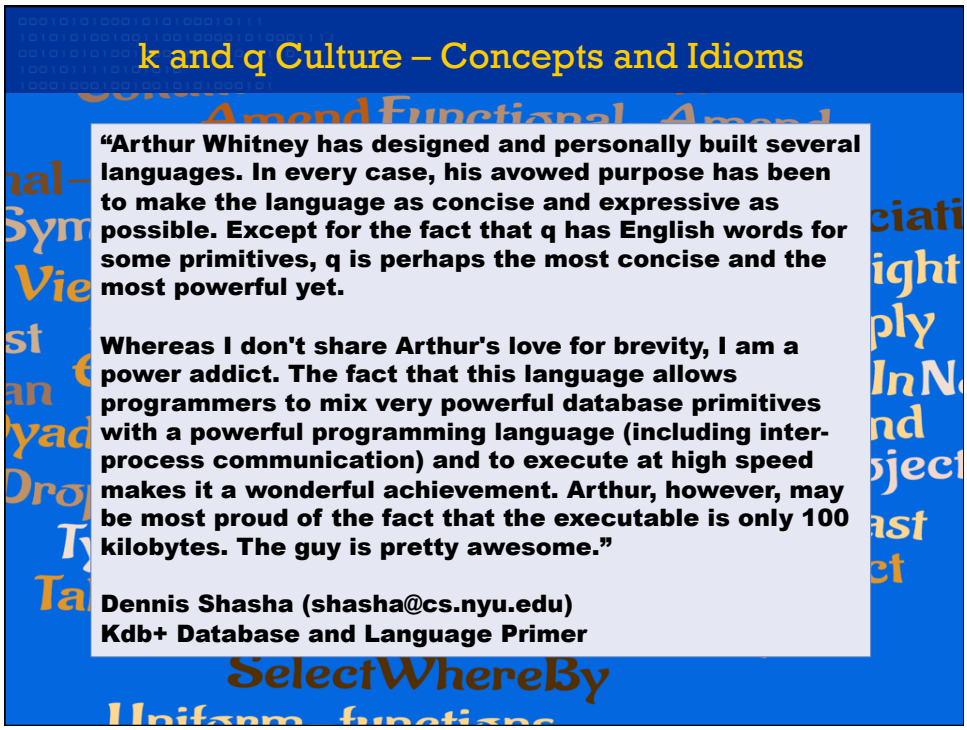
- core data type is an array or vector
- functional with assignment (no lazy eval)
- large library of built in generalized operations
- local and global scope (not lexically scoped)
- most functions are monadic or dyadic
- composition of functions/operations
- execution is right to left no precedence
  "left of right"

Each language has significant differences in their array model operations and library!

How to learn an array language?

1. Assume it will take weeks not hours or days?
2. Don't write programs, write short expressions
3. Learn the nous, verb and adverbs by doing idioms (Kata of Arrays)
4. Focus on expressions as data /shape transformers
5. Write many small functions, don't focus on smallest!
6. Start with normal control structures, later move away from loops


k and q Culture – Concepts and Idioms
Amend
Functional–Amend
Over
NameSpace
Reduce
Index
Vector
Nulls
Monadic
Global
Fill
LeftofRight
View
GroupBy
Join
Each
Enlist
Verbs
Infinities
Shape
Apply
EachRight
PointerChasing
asof
IPC
Local
Juxtaposition
Table
Rand
Flip
Find
Functions
Cut
Project
Drop
Adverbs
Count
EachLeft
Cast
Type
Distinct
Reshape
InnerProduct
Take
Enumeration
Dictionary
SelectWhereBy


k and q Culture – Concepts and Idioms
"Arthur Whitney has designed and personally built several languages. In every case, his avowed purpose has been to make the language as concise and expressive as possible. Except for the fact that q has English words for some primitives, q is perhaps the most concise and the most powerful yet.
Whereas I don't share Arthur's love for brevity, I am a power addict. The fact that this language allows programmers to mix very powerful database primitives with a powerful programming language (including inter-process communication) and to execute at high speed makes it a wonderful achievement. Arthur, however, may be most proud of the fact that the executable is only 100 kilobytes. The guy is pretty awesome."
Dennis Shasha (shasha@cs.nyu.edu)
Kdb+ Database and Language Primer
SelectWhereBy

q and KDB+ Syntax

Nouns – variables, constants, functions .

```
a, 42.42; (“xyz”;39;`Fred); myfunction[x;y;z];
```

Verbs - primitive symbols and infix e.g a+b ; a infixfcn b;

Note for readability juxtaposition any be used monadic functions; indexing; and function projections

```
a i  is short for a[i];
mymonadic `apple is short for mymonadic[`apple];
mymonadic:mydyadic 42 ;
```

Adverbs - modify dyadic functions and verbs to new verbs. Each (') modifies dyadic functions and verbs to apply to the items of lists instead of the lists themselves. For example,

```
1 2 3,4 5 / join 12345
(1 2 3;"abcd"),'(4 5;"e") / Join-Each (,')
(1 2 3 4 5;"abcde")
```


SQL + Functional Vector Programming

```
select date by stock, myavgs[3; mydouble price]
    from trade
```

mydouble and myavgs

are user defined functions

```
select from t where c by g
```

is syntactic sugar for `?[t;c;g]`

KDB+ Vector Functional Application Platform

Vector Functional Language k

- Comprehensive Data Types and Collections
- Function and Verb Composition with adverbs

```
q) count (1 2 3;"hello")  2
q) count each (1 2 3;"hello") 3 5
```

- Parallel Operations (Peach)

High Performance Relational Column Store (Memory and Disk)

Web Server, ODBC, IPC, C, Java, C# , R, Mattlab interfaces

Financial Vector Functional DSL q – SQL + Functions

- Find the volume weighted average price over 5 minute intervals for Intel

```
select vwap:quant wavg px by date, bucket:5 xbar
time.minute from trade where sym=`intc
```

Define 3 Tables – Book, Author, BookAuthor

```
author:([author:`king`hemingway`flaubert`lazere`shasha]
 address: `maine`expat`france`mamaroneck`newyork;
 area: `horror`suffering`psychology`journalism`puzzles)
book:([book: `forwhomthebelltolls`oldmanandthesea
 `shining`secretwindow`clouds`madambovary`sala
 language: `english`english`english`english`English
 `french`french`english; numprintings: 3 5 4 2 2 8 9 2)

bookauthor:([] author:`author
$`hemingway`hemingway`king`king`king
 `flaubert`flaubert`shasha`lazere;
 book:`book$`forwhomthebelltolls`oldmanandthesea`shining
 `secretwindow`clouds`madambovary`sala
 numfansinmillions: 20 20 50 50 30 60 30 0.02 0.02)
```

Implicit Joins and Groups reduce Code

```
/ SQL92:
SELECT DISTINCT bookauthor.author, book.language, author.address
FROM bookauthor, book author
 WHERE book.book = bookauthor.book AND
 author.author = bookauthor.author AND author.area = "psychology"
/ q
select distinct author,book.language,author.address
 from bookauthor where author.area = `psychology
/ SQL92:
SELECT book.language, SUM(numfansinmillions)
 FROM bookauthor, book
 WHERE bookauthor.book = book.book
 GROUP BY book.language
/ q implicit join and implicit group by
 select sum numfansinmillions by book.language from bookauthor
```


Word Count in Scalding

```
import com.twitter.scalding._
 class WordCountJob(args: Args) extends Job(args) {
   TextLine( args("input") )
     .read
     .flatMap('line -> 'word) {
       line: String =>
           line.trim.toLowerCase.split("\\W+")
     }
     .groupBy('word) { group => group.size('count) }
   }
    .write(Tsv(args("output")))
 }
```

Word Count in Hive and q

```
CREATE TABLE docs (line STRING);
LOAD DATA INPATH '/path/to/docs' INTO TABLE
docs;

CREATE TABLE word_counts AS
SELECT word, count(1) AS count FROM
(SELECT explode(split(line, '\W+'))
 AS word FROM docs) w GROUP BY wordORDER BY
word;
```

```
docs: ("This is my doc doc"; "This is another doc");

+/[{[doc] count each group lower " " vs doc} peach
docs]
```


Hive SQL vs q

```
SELECT t1.id, count(*) AS count
FROM  table1 t1 JOIN table2 t2 ON (t1.id = t2.id)
GROUP BY t1.id
ORDER BY count ASC;
```

```
/ Query 1: Find the counts for each id after joining two
tables, order by count

q1a: `cnt xasc 0! select  cnt: count t1col0 by id
  from (select from t2) ij `id xgroup select from t1
```

Hive SQL vs q

```
SELECT s.count, count(*) AS count2 FROM (
 SELECT t1.id, count(*) AS count  FROM  table1 t1
 JOIN table2 t2 ON (1.id = t2.id)
 GROUP BY t1.id ORDER BY count ASC) s
 GROUP BY s.count ORDER BY count2; // requires 5
map reduces
```

```
// Find the counts for each id after joining two
tables

q2a: count each group exec cnt from `cnt xasc 0!
select  cnt: count  t1col0 by id from (select from t2)
ij `id xgroup select from t1
```


Things to Love and Hate about q

Love

- Very productive short programs
- Efficient execution
- All you really need in a simple compact system
- No massive frameworks, associated code bulk and execution overhead

Frustrate (ate)

- Documentation and Error messages are minimal
- Several ways to say the same thing
- q is an "embedded" DSL which sometimes masks the regularity of k

Like most High Barrier Languages: It really grows on you if you stick with it long enough (~ 3 mths).

Think More! – Write Less Code ! !

High Barrier Small Surface Area ROI

>>

High Ceremony Large Surface Area Languages

Thanks!