Building Rich, High Performance Tools for Practical Data Analysis

Wes McKinney
@wesmckinn

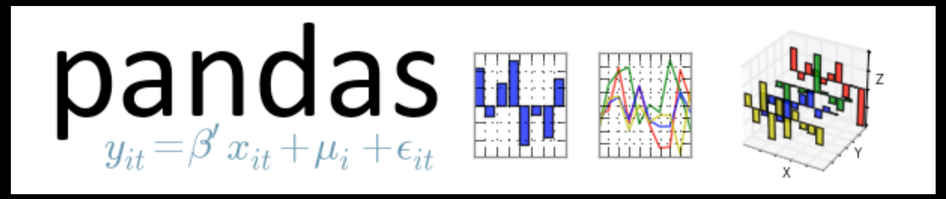
Lambda Foundry, Inc.

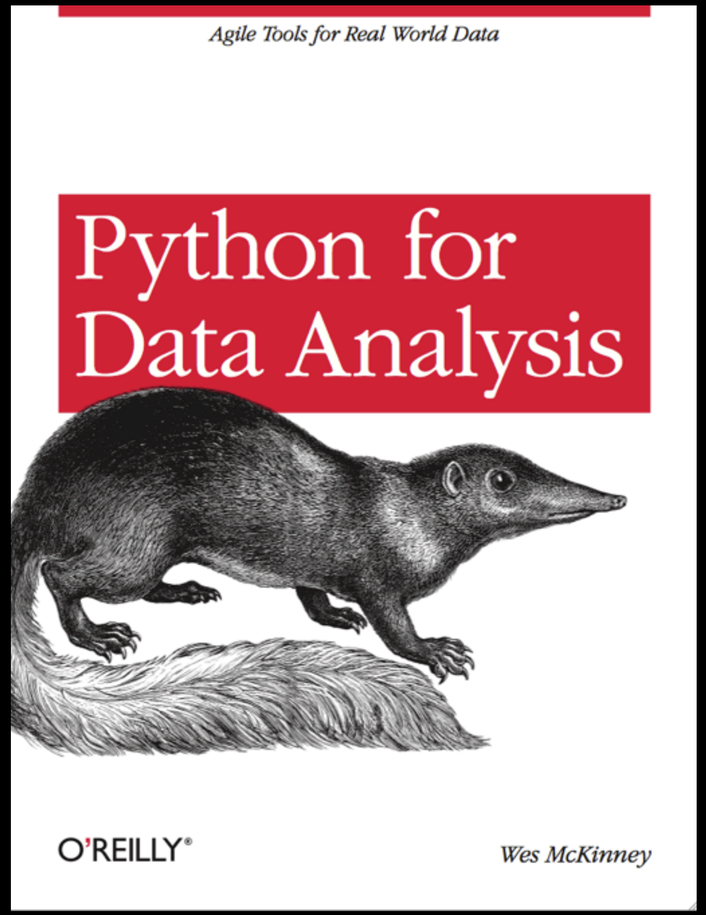
Talk Overview

- Background
- Goals
- Key Ingredients
- Examples

My Background

- MIT ’07, AQR 2007-2010, LF 2012-
- Lead developer of pandas (Python library)



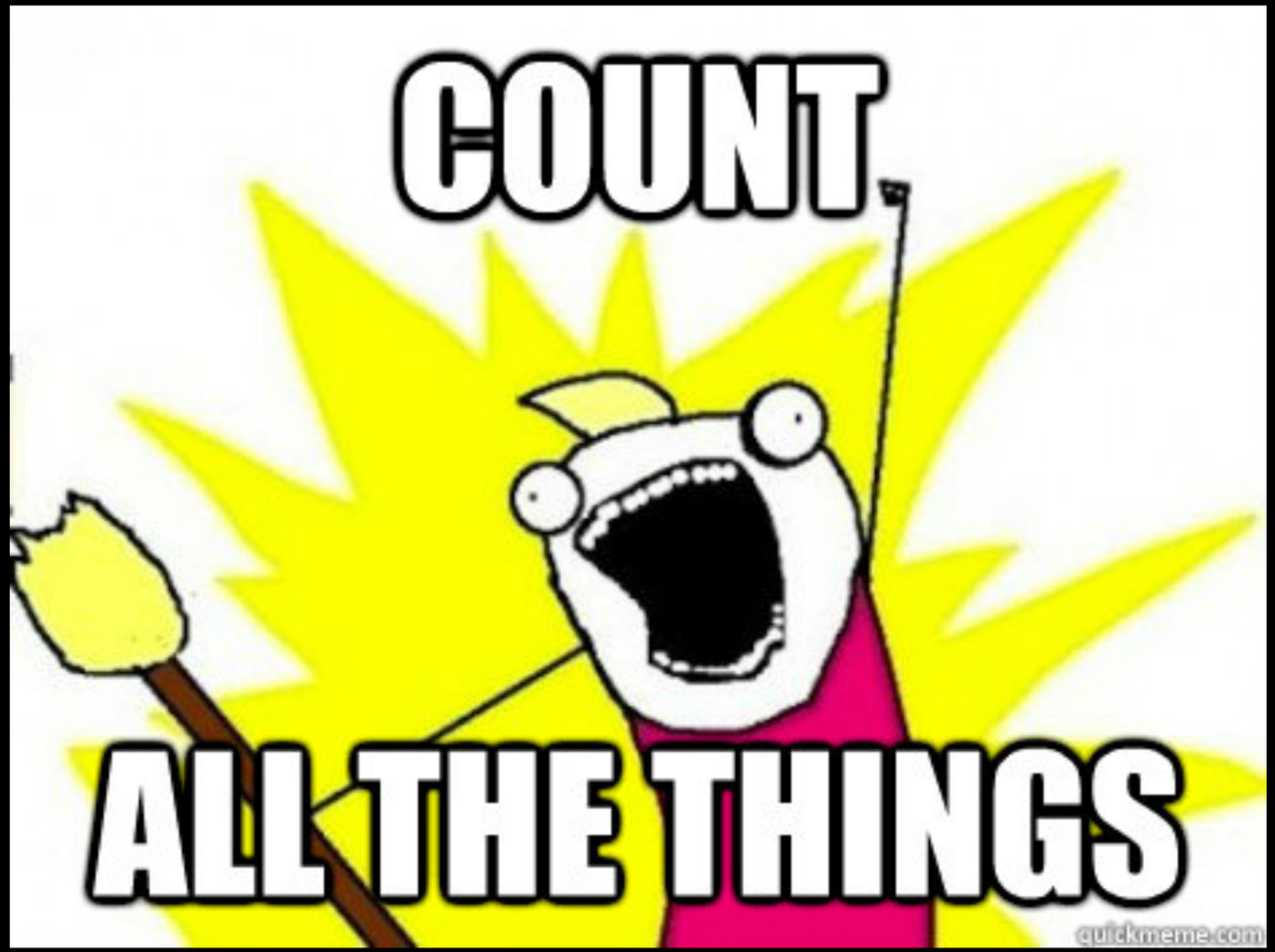


Goals

Data Tooling

Goals

Big™ and Small™ Data


COUNT
ALL THE THINGS
quickmeme.com

Goals

Simplify Data Wrangling

Goals

User interface design

User interface design

i.e. “how do I do what I need?”

FORGETS BREAK STATEMENT
RUNS $10,000 MAP REDUCE JOB
quickmeme.com

API Design

API Design

- “Fits your brain”
- “All my functions have 17 arguments”

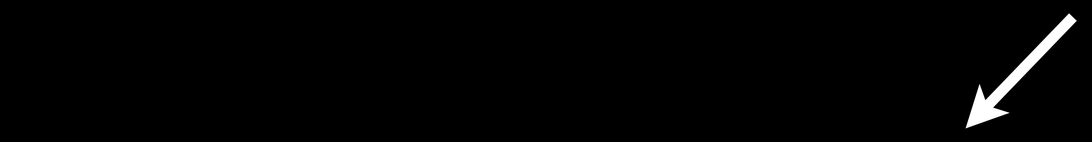
Syntax Matters

API Design

```
((($:@(<#[),(=#[),$:@(>#[))({~ ?@#))^:(1<#)

{f:*x@1?#x;:[0=#x;x;,/(_f x@&x<f;x@&x=f;_f x@&x>f)]}
```

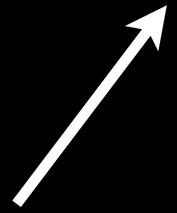
API Design

J



```
((($:@(<#[),(=#[),$:@(>#[))({~ ?@#))^:(1<#)
```

```
{f:*x@1?#x;:[0=#x;x;,/(_f x@&x<f;x@&x=f;_f x@&x>f)]}
```



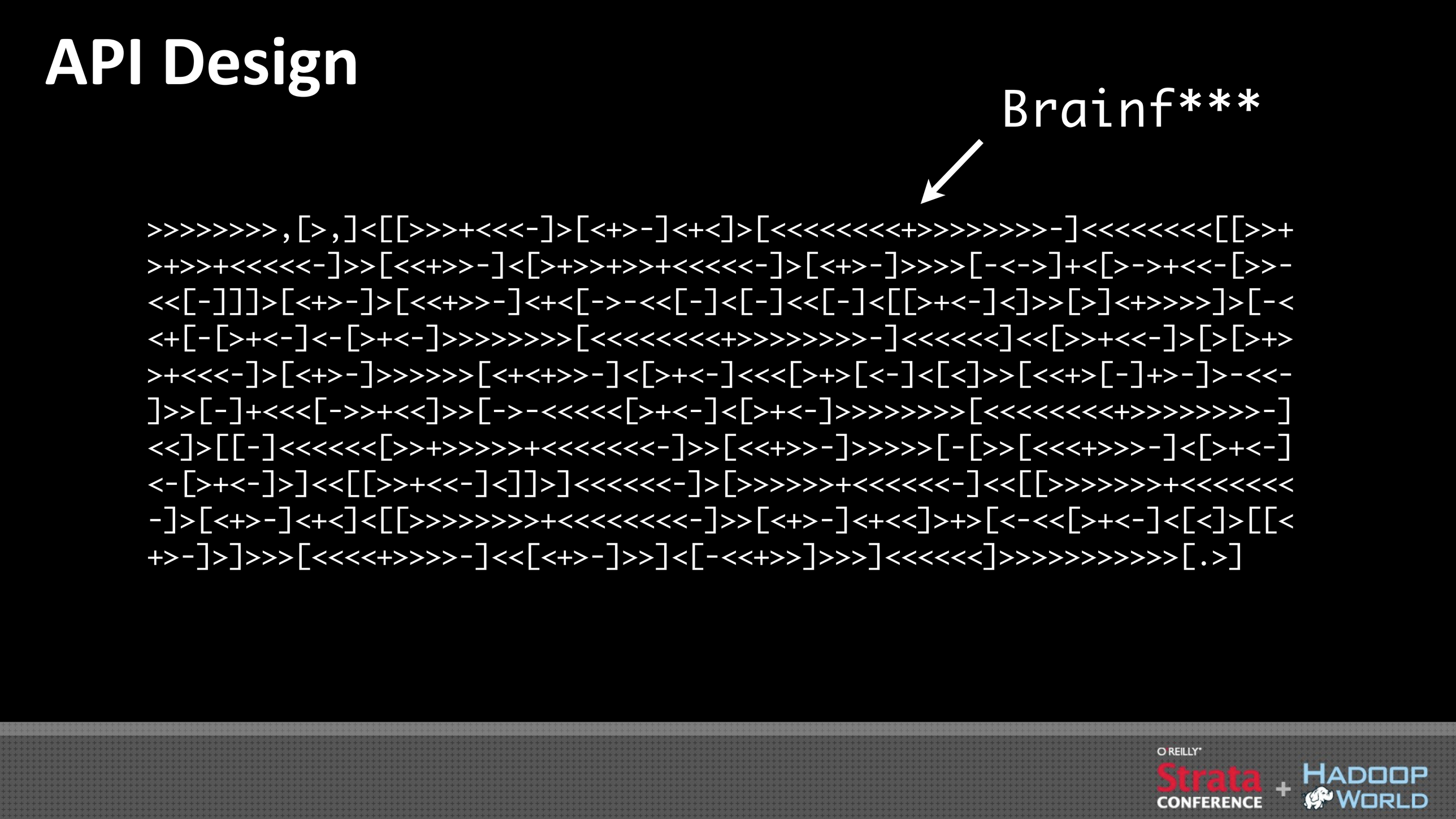
K/Kona

API Design

```
>>>>>>>>,[>,]<[[>>>+<<<-]>[<+>-]<+<]>[<<<<<<<<<+>>>>>>>>>-]<<<<<<<<[[>>+
>+>>+<<<<<-]>>[<<+>>-]<[>+>>+>>+<<<<<-]>[<+>-]>>>>[-<->]+<[>->+<<-[>>-
<<[-]]]>[<+>-]>[<<+>>-]<+<[->-<<[-]<[-]<<[-]<[[>+<-]<]>>[>]<+>>>>]>[-<
<+[-[>+<-]<-[>+<-]>>>>>>>>[<<<<<<<<<+>>>>>>>>>-]<<<<<<]<<[>>+<<-]>[>[>+>
>+<<<-]>[<+>-]>>>>>>[<+<+>>-]<[>+<-]<<<[>+>[<-]<[<]>>[<<+>[-]+>-]>-<<-
]>>[-]+<<<[->>+<<]>>[->-<<<<<[>+<-]<[>+<-]>>>>>>>>[<<<<<<<<<+>>>>>>>>>-]
<<]>[[-]<<<<<<[>>+>>>>>+<<<<<<<-]>>[<<+>>-]>>>>>[-[>>[<<<+>>>-]<[>+<-]
<-[>+<-]>]<<[[>>+<<-]<]]>]<<<<<<-]>[>>>>>>+<<<<<<-]<<[[>>>>>>>+<<<<<<<
-]>[<+>-]<+<]<[[>>>>>>>>+<<<<<<<<<-]>>[<+>-]<+<<]>+>[<-<<[>+<-]<[<]>[[<
+>-]>]>>>[<<<<+>>>>-]<<[<+>-]>>]<[-<<+>>]>>>]<<<<<]>>>>>>>>>>>>[.>]
```

API Design

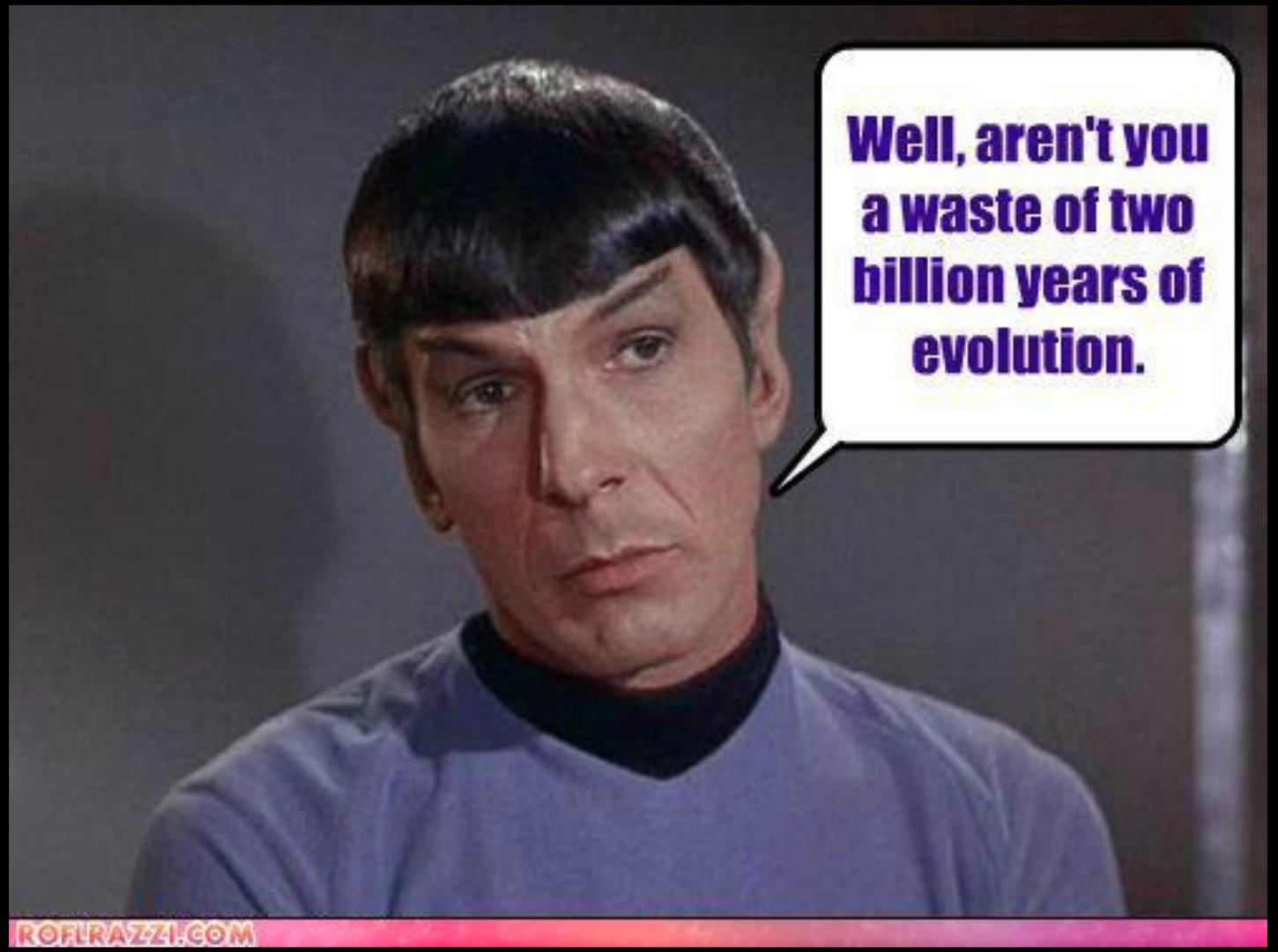
Brainf***



```
>>>>>>>>,[>,]<[[>>>+<<<-]>[<+>-]<+<]>[<<<<<<<<<+>>>>>>>>>-]<<<<<<<<[[>>+
>+>>+<<<<<-]>>[<<+>>-]<[>+>>+>>+<<<<<-]>[<+>-]>>>>[-<->]+<[>->+<<-[>>-
<<[-]]]>[<+>-]>[<<+>>-]<+<[->-<<[-]<[-]<<[-]<[[>+<-]<]>>[>]<+>>>>]>[-<
<+[-[>+<-]<-[>+<-]>>>>>>>>[<<<<<<<<<+>>>>>>>>>-]<<<<<<]<<[>>+<<-]>[>[>+>
>+<<<-]>[<+>-]>>>>>>[<+<+>>-]<[>+<-]<<<[>+>[<-]<[<]>>[<<+>[-]+>-]>-<<-
]>>[-]+<<<[->>+<<]>>[->-<<<<<[>+<-]<[>+<-]>>>>>>>>[<<<<<<<<<+>>>>>>>>>-]
<<]>[[-]<<<<<<[>>+>>>>>+<<<<<<<-]>>[<<+>>-]>>>>>[-[>>[<<<+>>>-]<[>+<-]
<-[>+<-]>]<<[[>>+<<-]<]]>]<<<<<<-]>[>>>>>>+<<<<<<-]<<[[>>>>>>>+<<<<<<<
-]>[<+>-]<+<]<[[>>>>>>>>+<<<<<<<<<-]>>[<+>-]<+<<]>+>[<-<<[>+<-]<[<]>[[<
+>-]>]>>>[<<<<+>>>>-]<<[<+>-]>>]<[-<<+>>]>>>]<<<<<]>>>>>>>>>>>>[.>]
```

“A user interface can handle only so much complexity or it becomes unusable.”

Guido van Rossum

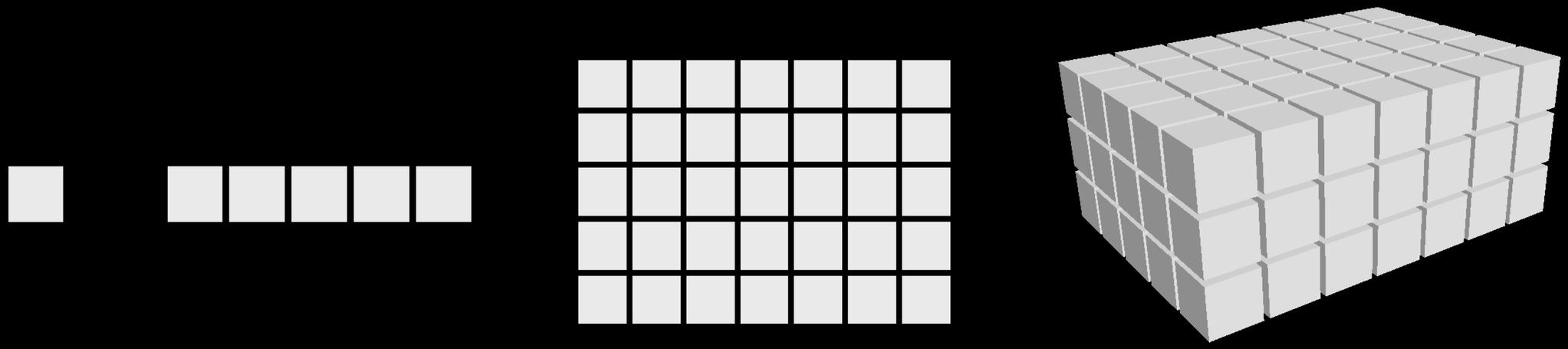

Well, aren't you a waste of two billion years of evolution.
ROFLRAZZI.COM

Key Ingredients

- API / User interface design
- Data types
- Arrays
- Data structures
- Efficient Algorithms

Arrays

- Blob of bytes as multidimensional array
- All elements same type
- Implementations: NumPy (Python), J / APL, R, MATLAB, ...



Scalar
Rank 0

Vector
Rank 1

Matrix
Rank 2

Cube/Hypercube
Rank > 2

Arrays

- Loopless programming
  - Vectorization
  - Broadcasting

- No-copy views

(Some) Basic data types

- Integer
- Floating point / Complex
- Characters and Strings (ASCII or Unicode)
- Date and time
- Box / “Object”

(Some) Basic data types

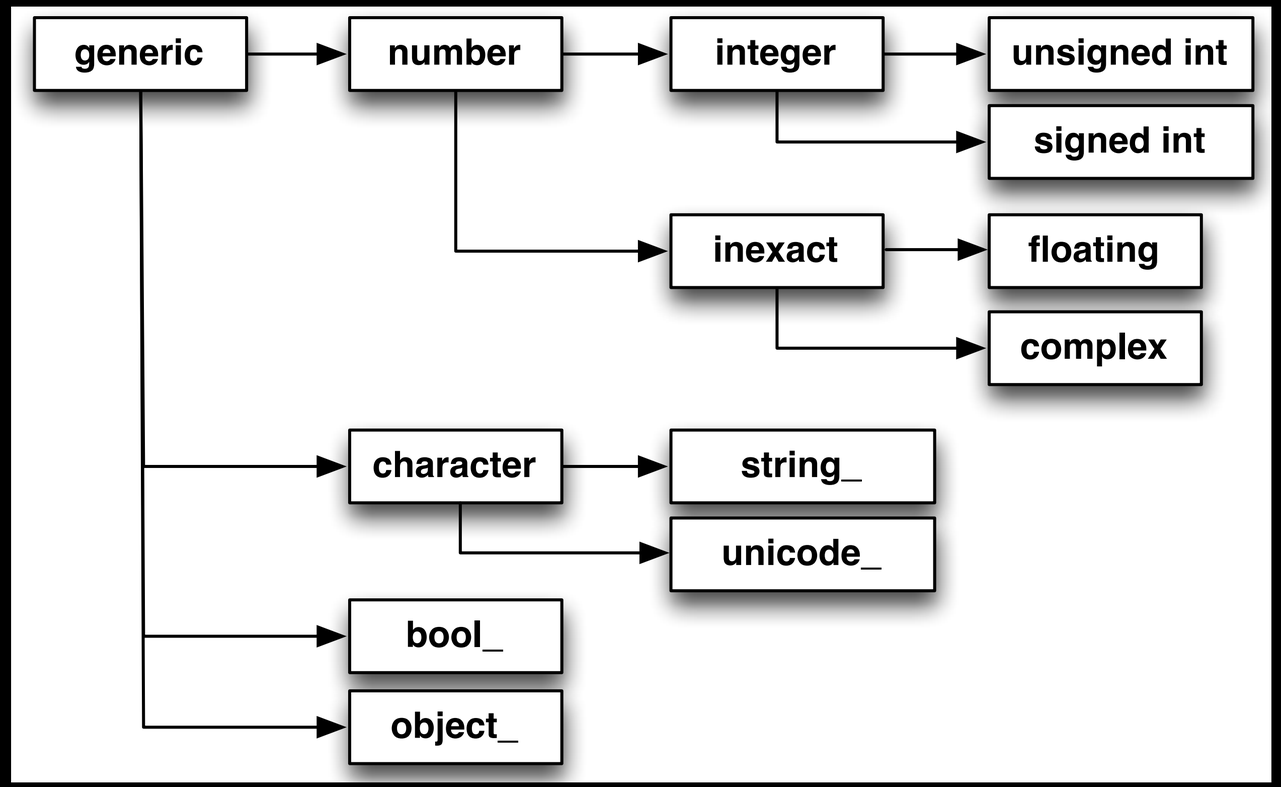
- Integer
- Floating point / Complex
- Characters and Strings (ASCII or Unicode)
- Date and time
- Box / “Object”
- Categorical / enumeration (R: “factors”)
- Record or structure (one or more of the above)

Data types

Also, any values can be “missing” (NA)

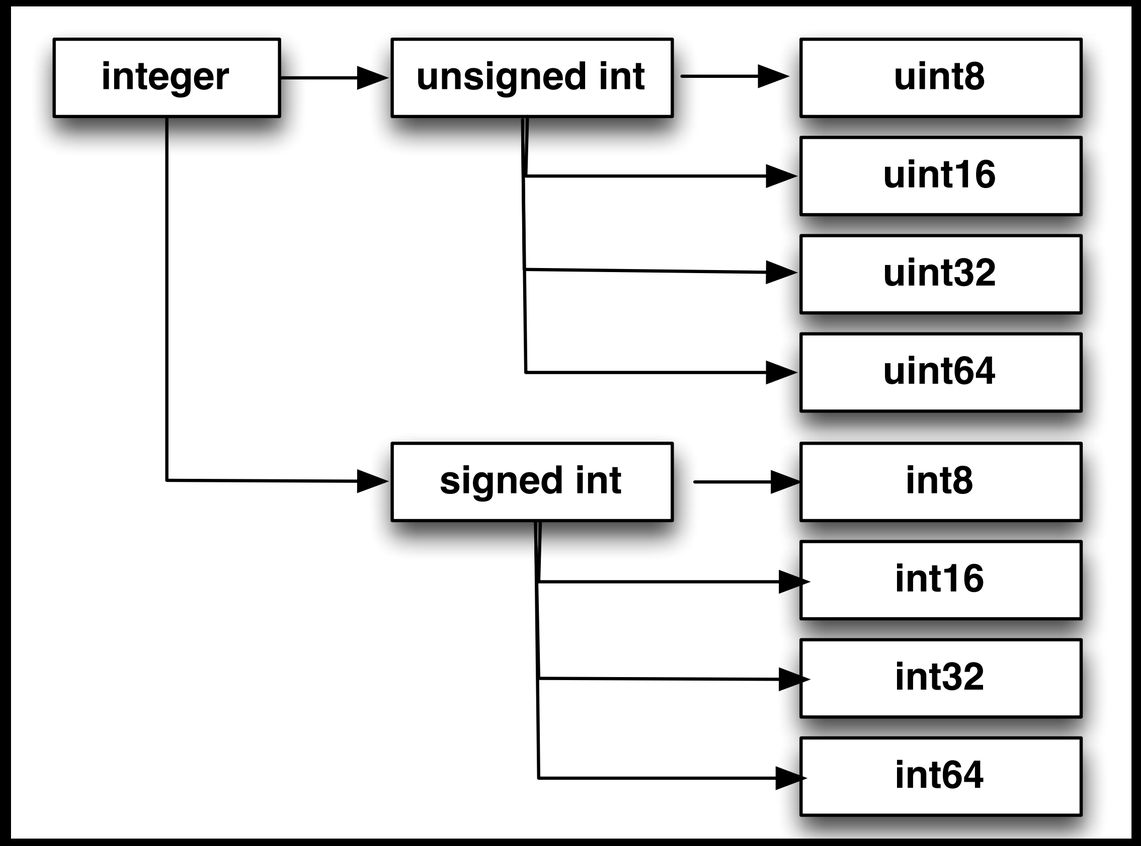
Risks

- Stuck in type soup



Risks

- Actually it’s more like



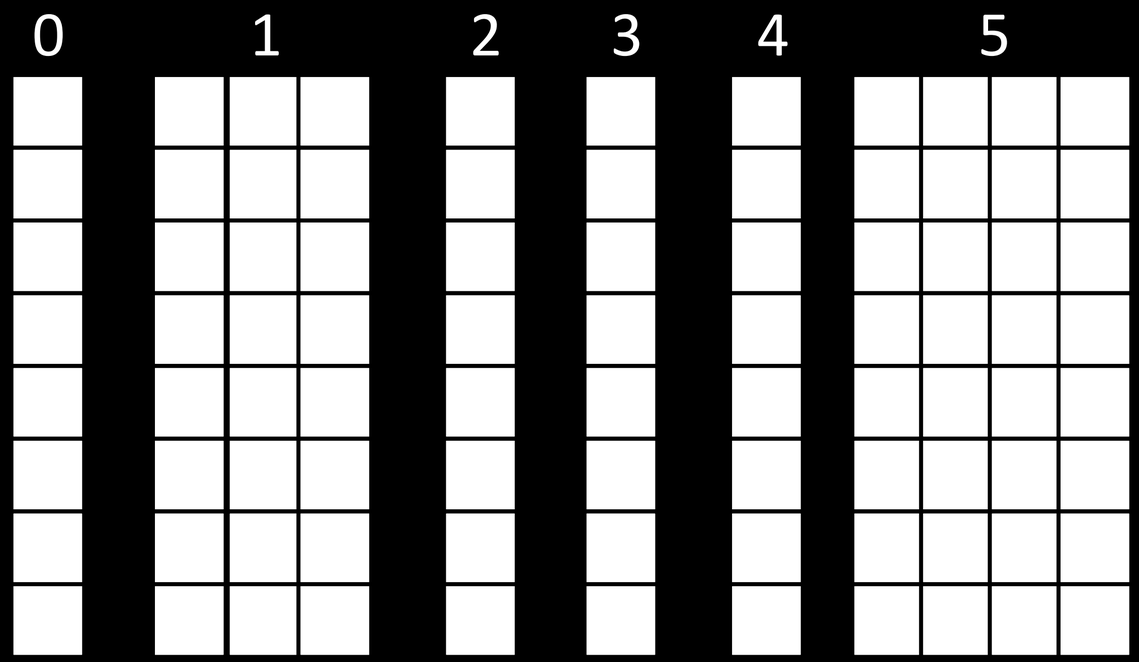
Data types and users

- Control over machine representation
  - Treacherous, but often necessary

- Simplicity vs. power and control

Tables

- A sequence of arrays, each with own data type



Tables

- A sequence of arrays, each with own data type
- Common simplification: only 1-dimensional arrays

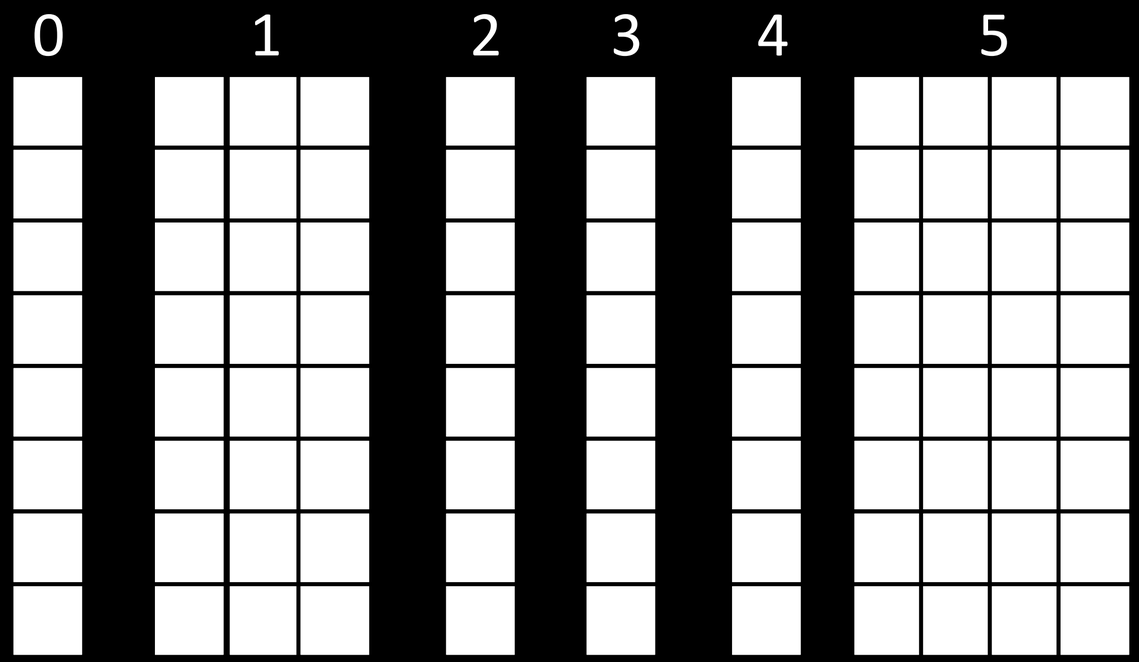


Table and Array Axis Labeling

- One or more categorical arrays per axis
- Numerous uses

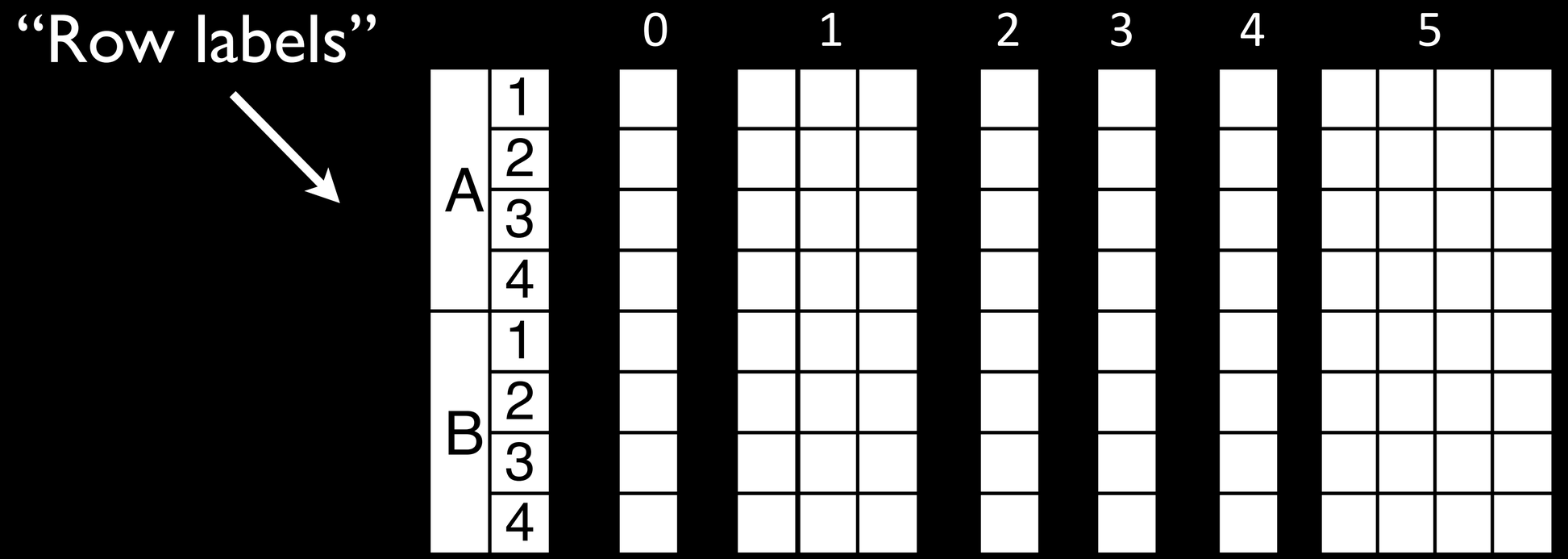


Table and Array Axis Labeling

	day	Fri		Sat		Sun		Thur	
		sum	len	sum	len	sum	len	sum	len
time	smoker								
Dinner	No	8.25	3	139.63	45	180.57	57	3	1
	Yes	27.03	9	120.77	42	66.82	19	NA	NA
Lunch	No	3	1	NA	NA	NA	NA	117.32	44
	Yes	13.68	6	NA	NA	NA	NA	51.51	17

Table and Array Axis Labeling

	time	smoker	day	sum	len
0	Dinner	No	Fri	8.25	3
1	Dinner	No	Sat	139.63	45
2	Dinner	No	Sun	180.57	57
3	Dinner	No	Thur	3	1
4	Dinner	Yes	Fri	27.03	9

• • • •

This predicates on the existence of sane APIs

Some “primitive” operations

- Align / join / merge / “Vlookup”
- Concatenate
- Reshape, Stack/fold, Unstack/unfold, Pivot, Melt
- Subset: Drop rows, columns
- Elementwise transforms: map, replace, fill, string stuff
- Group By: Apply, Aggregate, Transform, Cut, ...
- Set operations: Unique, deduplicate, is-in

Example: concatenation

left (prices)

	AAPL	JNJ	SPX	XOM
2011-09-06	379.74	64.64	1165.24	71.15
2011-09-07	383.93	65.43	1198.62	73.65
2011-09-08	384.14	64.95	1185.90	72.82
2011-09-09	377.48	63.64	1154.23	71.01
2011-09-12	379.94	63.59	1162.27	71.84
2011-09-13	384.62	63.61	1172.87	71.65
2011-09-14	389.30	63.73	1188.68	72.64

right (volume)

	AAPL	JNJ	XOM
2011-09-06	18173500	15848300	25416300
2011-09-07	12492000	10759700	23108400
2011-09-08	14839800	15551500	22434800
2011-09-09	20171900	17008200	27969100
2011-09-12	16697300	13448200	26205800

Example: concatenation

```
result = concat([left, right], axis=1,
                keys=['price', 'volume'])
```

	price				volume		
	AAPL	JNJ	SPX	XOM	AAPL	JNJ	XOM
2011-09-06	379.74	64.64	1165.24	71.15	18173500	15848300	25416300
2011-09-07	383.93	65.43	1198.62	73.65	12492000	10759700	23108400
2011-09-08	384.14	64.95	1185.90	72.82	14839800	15551500	22434800
2011-09-09	377.48	63.64	1154.23	71.01	20171900	17008200	27969100
2011-09-12	379.94	63.59	1162.27	71.84	16697300	13448200	26205800
2011-09-13	384.62	63.61	1172.87	71.65	NaN	NaN	NaN
2011-09-14	389.30	63.73	1188.68	72.64	NaN	NaN	NaN

Example: stack/fold

```
result.stack(0)
```

		AAPL	JNJ	SPX	XOM
2011-09-06	price	379.74	64.64	1165.24	71.15
	volume	18173500.00	15848300.00	NaN	25416300.00
2011-09-07	price	383.93	65.43	1198.62	73.65
	volume	12492000.00	10759700.00	NaN	23108400.00
2011-09-08	price	384.14	64.95	1185.90	72.82
	volume	14839800.00	15551500.00	NaN	22434800.00
2011-09-09	price	377.48	63.64	1154.23	71.01
	volume	20171900.00	17008200.00	NaN	27969100.00
2011-09-12	price	379.94	63.59	1162.27	71.84
	volume	16697300.00	13448200.00	NaN	26205800.00
2011-09-13	price	384.62	63.61	1172.87	71.65
2011-09-14	price	389.30	63.73	1188.68	72.64

Under the hood

- Array / vector operations
- Efficient data movement
- Hash sets and tables
- Sorting algorithms
- Relational algebra

Example: group-by

- H. Wickham “The Split-Apply-Combine Strategy for Data Analysis”
- Beyond relational databases

```
SELECT key1, key2, key3,
       MEAN(value1), STD(value2)
FROM table
GROUP BY key1, key2, key2
```

Example: group-by

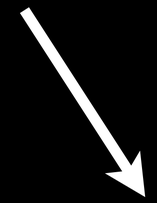
- In pseudocode

```
grouped = table.groupby(key_list)

result = grouped.apply(function)
```

Example: group-by

- In pseudocode

Arrays, functions, column names, …

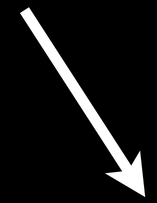


```
grouped = table.groupby(key_list)

result = grouped.apply(function)
```

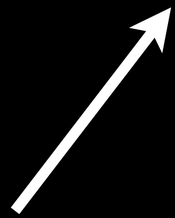
Example: group-by

- In pseudocode

Arrays, functions, column names, ...



```
grouped = table.groupby(key_list)

result = grouped.apply(function)
```



Preferably, any function accepting an array or table

Example: group-by

- In pseudocode

Arrays, functions, column names, ...

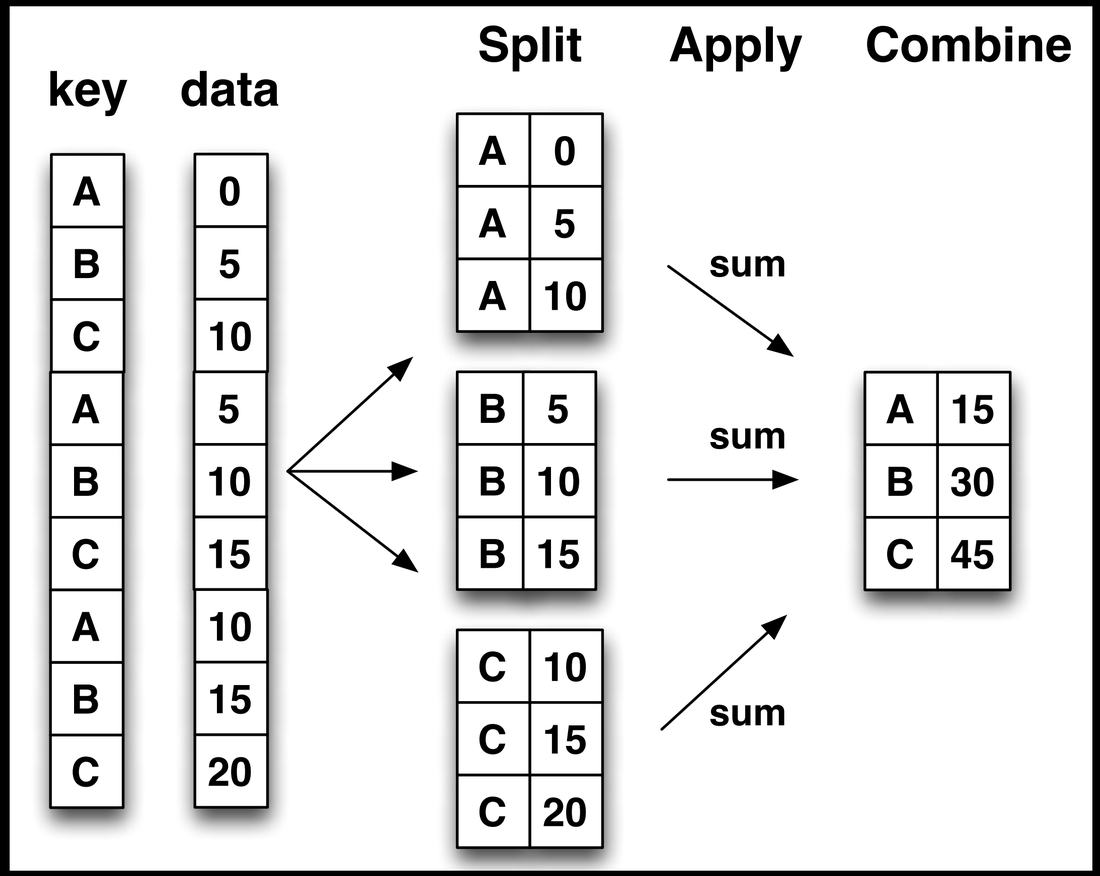
```
grouped = table.groupby(key_list)

result = grouped.apply(function)
```

Something useful?

Preferably, any function accepting an array or table

Example: group-by

- In pictures



```
by_size = table.groupby('size')
by_size.apply(topn, 'tip_pct', n=2)
```

	total_bill	tip	sex	smoker	day	time	size	tip_pct
1	16.99	1.01	Female	No	Sun	Dinner	2	0.059447
2	10.34	1.66	Male	No	Sun	Dinner	3	0.160542
3	21.01	3.50	Male	No	Sun	Dinner	3	0.166587
4	23.68	3.31	Male	No	Sun	Dinner	2	0.139780
5	24.59	3.61	Female	No	Sun	Dinner	4	0.146808

• • • •

		total_bill	tip	sex	smoker	day	time	size	tip_pct
size									
1	**68**	3.07	1.00	Female	Yes	Sat	Dinner	1	0.325733
	223	8.58	1.92	Male	Yes	Fri	Lunch	1	0.223776
2	**173**	7.25	5.15	Male	Yes	Sun	Dinner	2	0.710345
	179	9.60	4.00	Female	Yes	Sun	Dinner	2	0.416667
3	**215**	28.17	6.50	Female	Yes	Sat	Dinner	3	0.230742
	18	16.29	3.71	Male	No	Sun	Dinner	3	0.227747
4	**184**	23.17	6.50	Male	Yes	Sun	Dinner	4	0.280535
	64	18.29	3.76	Male	Yes	Sat	Dinner	4	0.205577
5	**186**	20.69	5.00	Male	No	Sun	Dinner	5	0.241663
	156	29.85	5.14	Female	No	Sun	Dinner	5	0.172194
6	**142**	34.30	6.70	Male	No	Thur	Lunch	6	0.195335
	144	27.05	5.00	Female	No	Thur	Lunch	6	0.184843

```
cuts = cut(table.size, [0, 3, 6])
table.groupby(cuts).apply(topn, 'tip_pct', n=4)
```

		total_bill	tip	sex	smoker	day	time	size	tip_pct
size									
(0, 3]	**173**	7.25	5.15	Male	Yes	Sun	Dinner	2	0.710345
	179	9.60	4.00	Female	Yes	Sun	Dinner	2	0.416667
	68	3.07	1.00	Female	Yes	Sat	Dinner	1	0.325733
	233	11.61	3.39	Male	No	Sat	Dinner	2	0.291990
(3, 6]	**184**	23.17	6.50	Male	Yes	Sun	Dinner	4	0.280535
	186	20.69	5.00	Male	No	Sun	Dinner	5	0.241663
	64	18.29	3.76	Male	Yes	Sat	Dinner	4	0.205577
	212	25.89	5.16	Male	Yes	Sat	Dinner	4	0.199305

GroupBy: inside

- Key arrays -> Categorical variables (hash table pass)
- Apply step
  - Simple functions: 1 pass through data
  - Generally: sort data (O(N)!), iterate through chunks
- Combine: don't lose information!

GroupBy: inside

- Key arrays -> Categorical variables (hash table pass)
- Apply step
  - Simple functions: 1 pass through data
  - Generally: sort data (O(N)!), iterate through chunks
- Combine: don’t lose information!

- Bonus: precompute (all the things)

Sources of performance / efficiency

- Optimal algorithms
- Cache-efficient memory layout
- “Boxing” overhead
- Minimal copying of data
- Elimination of temporary variables

Not all algo impls created equal

- Example: Unique
- Test case: 1M integers with 10K unique values

Not all algo impls created equal

- Example: Unique
- Test case: 1M integers with 10K unique values
- Shootout
  - R: 17.8 ms
  - Python-NumPy: 24.4 ms
  - Python-pandas: 10.1 ms
  - Julia: 170 ms (Set(values))

Not all algo impls created equal

- Example: Unique
- Test case: 1M integers with 10K unique values
- Shootout
  - R: 17.8 ms
  - Python-NumPy: 24.4 ms
  - Python-pandas: 10.1 ms
  - Julia: 170 ms (Set(values))
  - **J: 2.2 ms (~. values)**

Examples

TL;DR

Thanks!

On Twitter: @wesmckinn