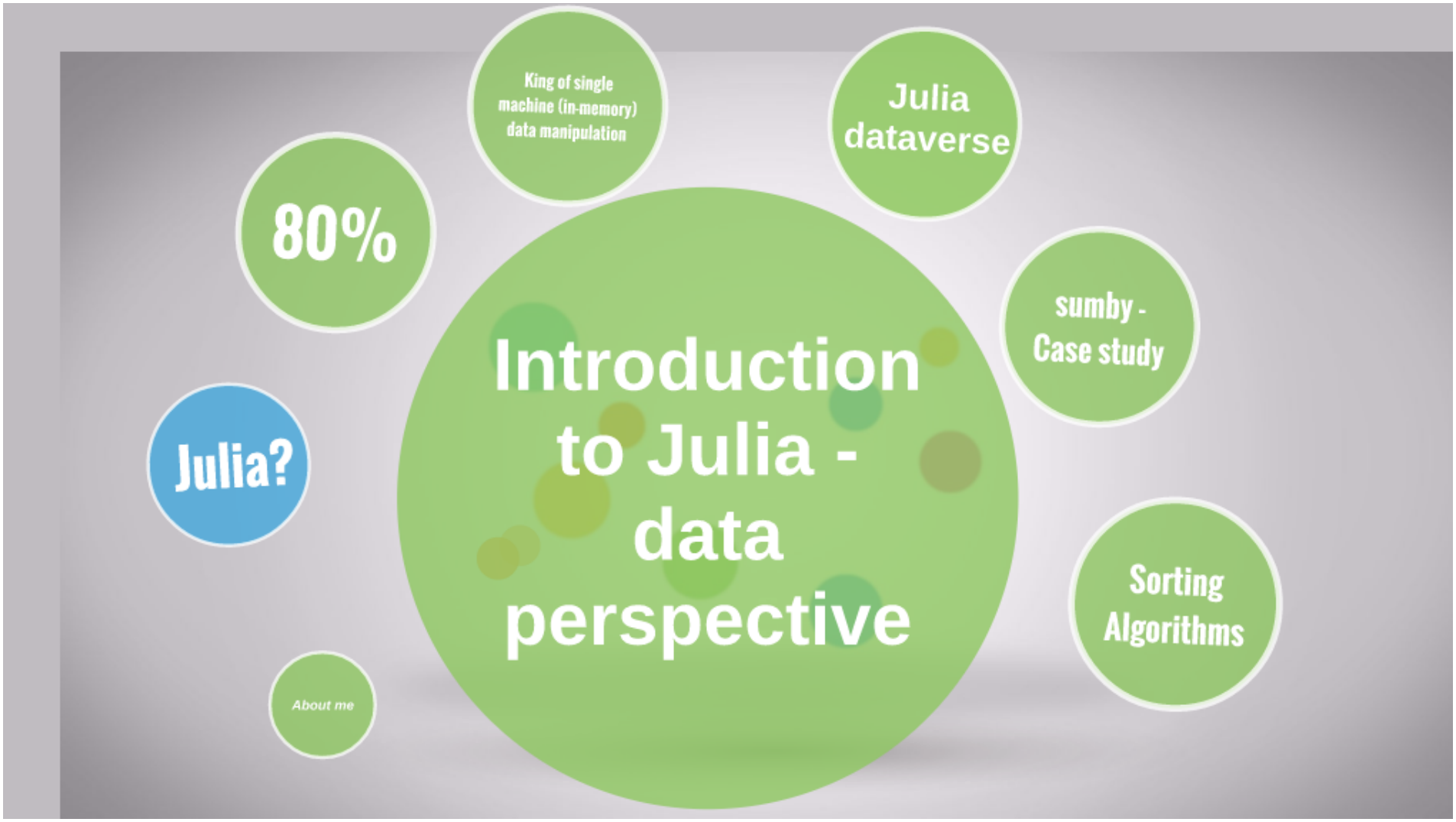
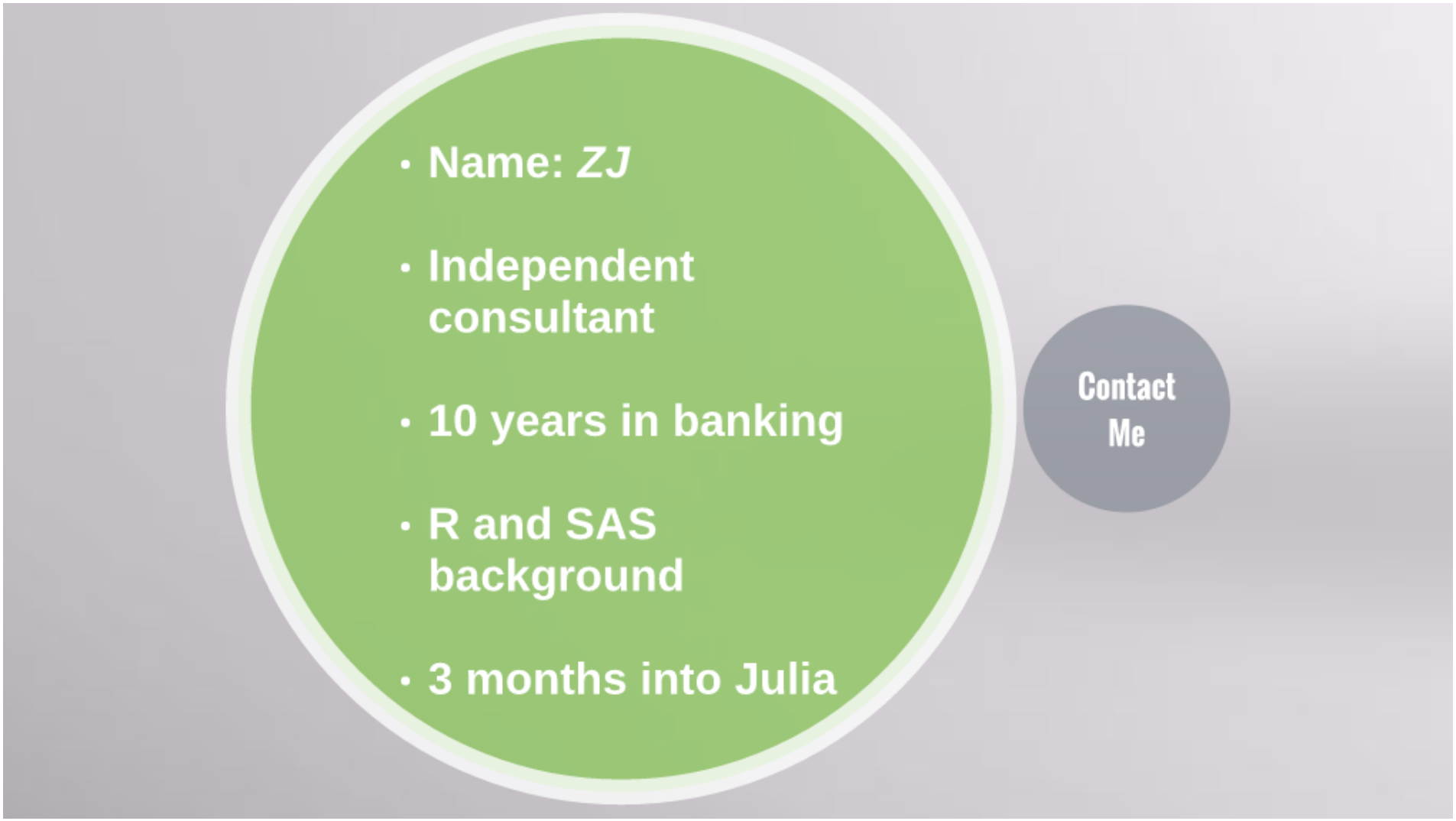

King of single machine (in-memory) data manipulation
Julia dataverse
80%
sumby - Case study
Introduction to Julia - data perspective
Julia?
Sorting Algorithms
About me

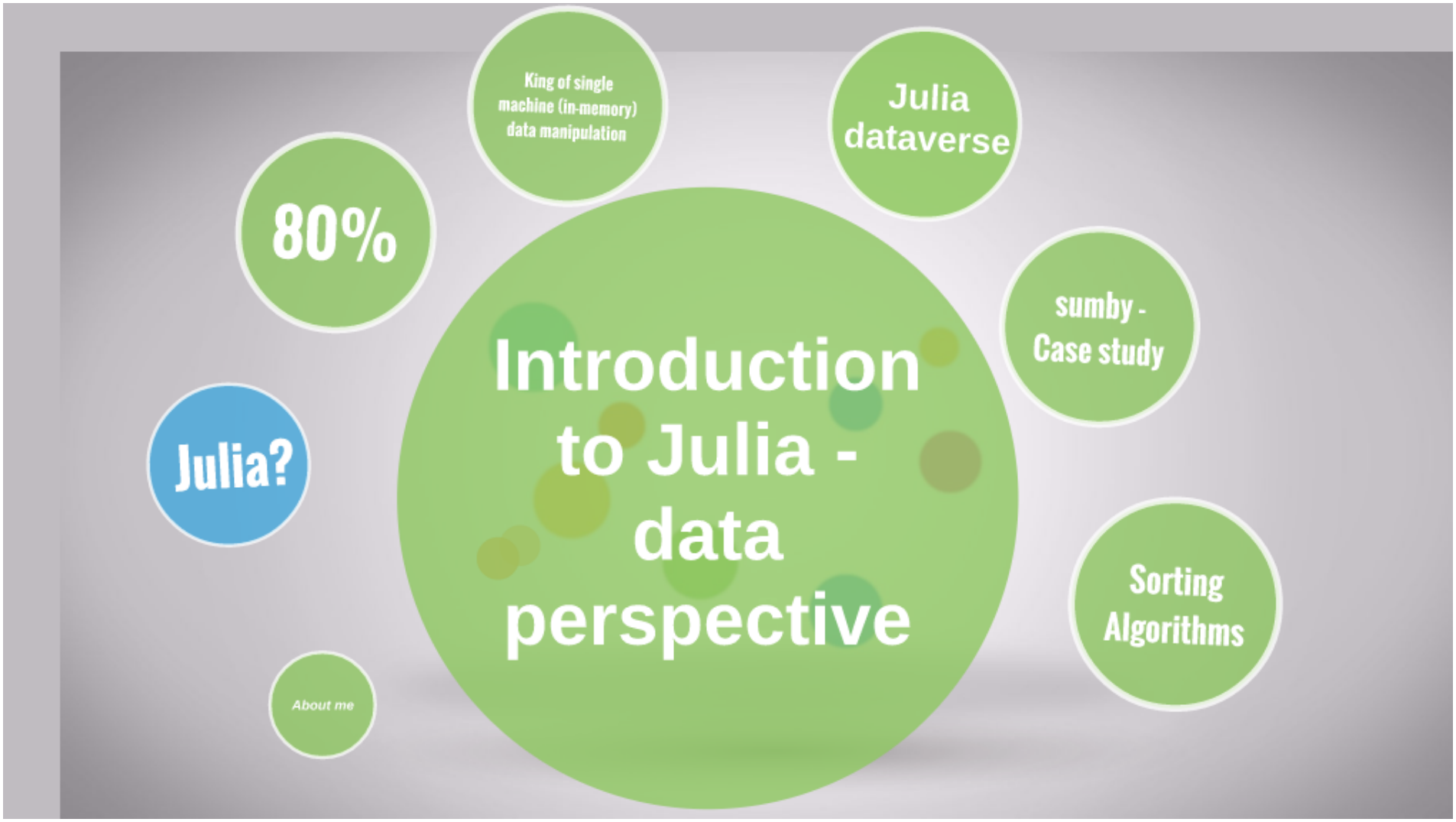

• Name: *ZJ*
• Independent consultant
• 10 years in banking
• R and SAS background
• 3 months into Julia
Contact Me

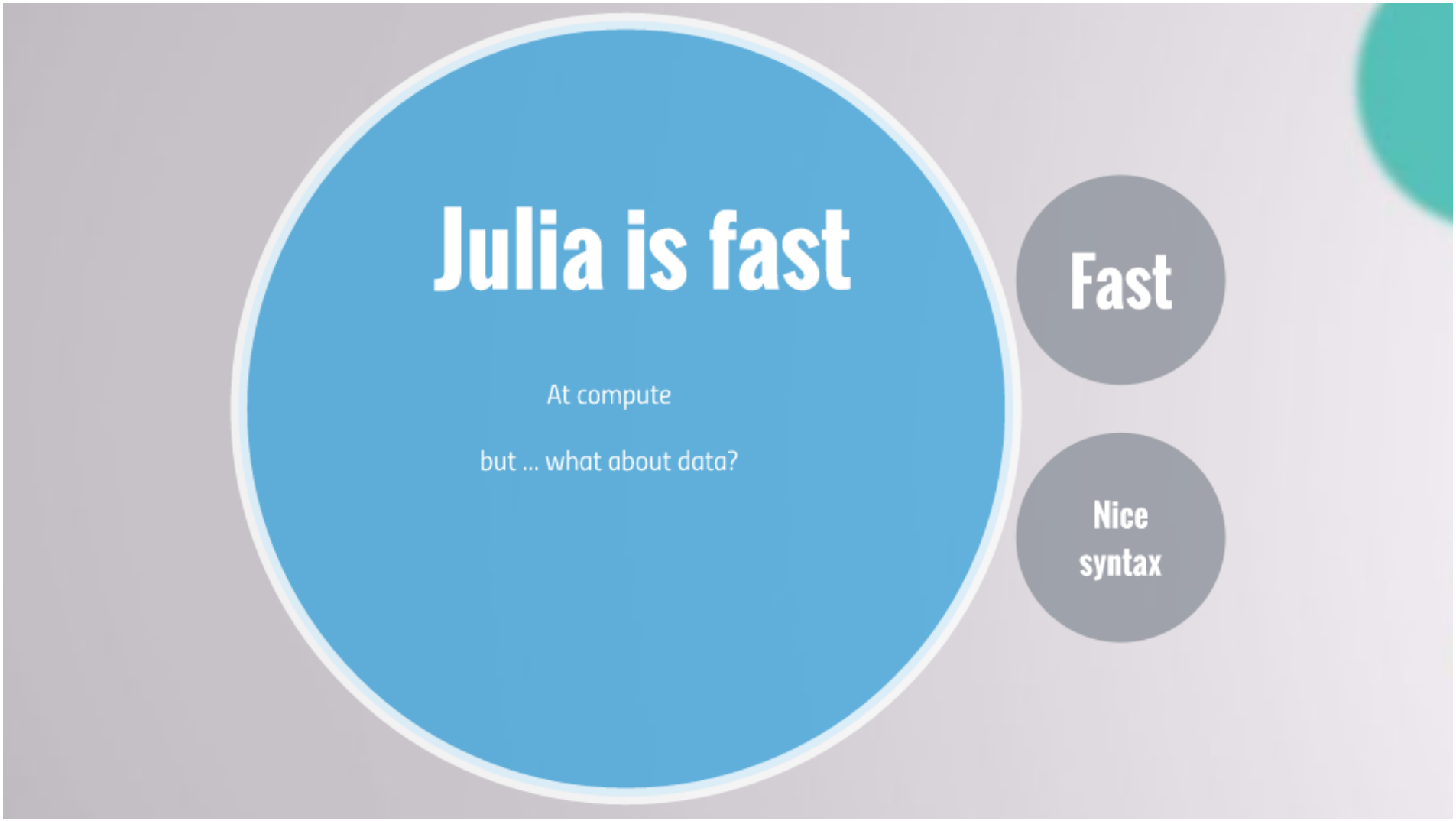
Contact Me

LinkedIn: https://linkedin.com/in/daizj

Github: @xiaodaigh

Email: dzj@analytixware.com


King of single machine (in-memory) data manipulation
Julia dataverse
80%
sumby - Case study
Introduction to Julia - data perspective
Julia?
Sorting Algorithms
About me


Julia is fast
At compute
but ... what about data?
Fast
Nice syntax

Fast

- Why is Julia fast?
- Why is C fast?
- Why are R & Python slow?

Why is C fast?

- C is compiled
  - What does it mean?
- A function in C knows
  - the type of variables it uses and
  - the type of outputs
- Based on these information one can optimise code
- In general the more you know about something the better you can optimize algorithms (e.g. categorical sumby - demoed later)

Why are R & Python slow?

- R & Python are interpreted not compiled
- A function in R or Python does not have the same level of "knowledge" of a C function

Julia is fast because

- It pretends to be R & Python
- But specialises its compiled code to the type of inputs - like C

```
testmul(a,b) = a*b;          > testmul

testmul(10, 100)             1000
testmul("b","c")             "bc"

@code_warntype testmul(10, 100)   ✓
```

Julia compiles the same function to specialized code depending on type of input

```
@code_warntype testmul(10, 100)
# Variables:
#   #self# <optimized out>
#   a::Int64
#   b::Int64
#
# Body:
#   begin
#       return (Base.mul_int)(a::Int64, b::Int64)::Int64
#   end::Int64

@code_warntype testmul("b","c")
# Variables:
#   #self# <optimized out>
#   a::String
#   b::String
#
# Body:
#   begin
#       return $(Expr(:invoke, MethodInstance for string(::String, ::Vararg
#   end::String
```

Implications for writing performant code

Type-stability is important
- **Types: Integers, Strings, Floats, etc.**

If Julia can't infer type then it will use ANY as type
- **Speed might reduce to R & Python**

Avoid switching the type of variables within functions

Avoid returning different types of outputs on the same type of inputs

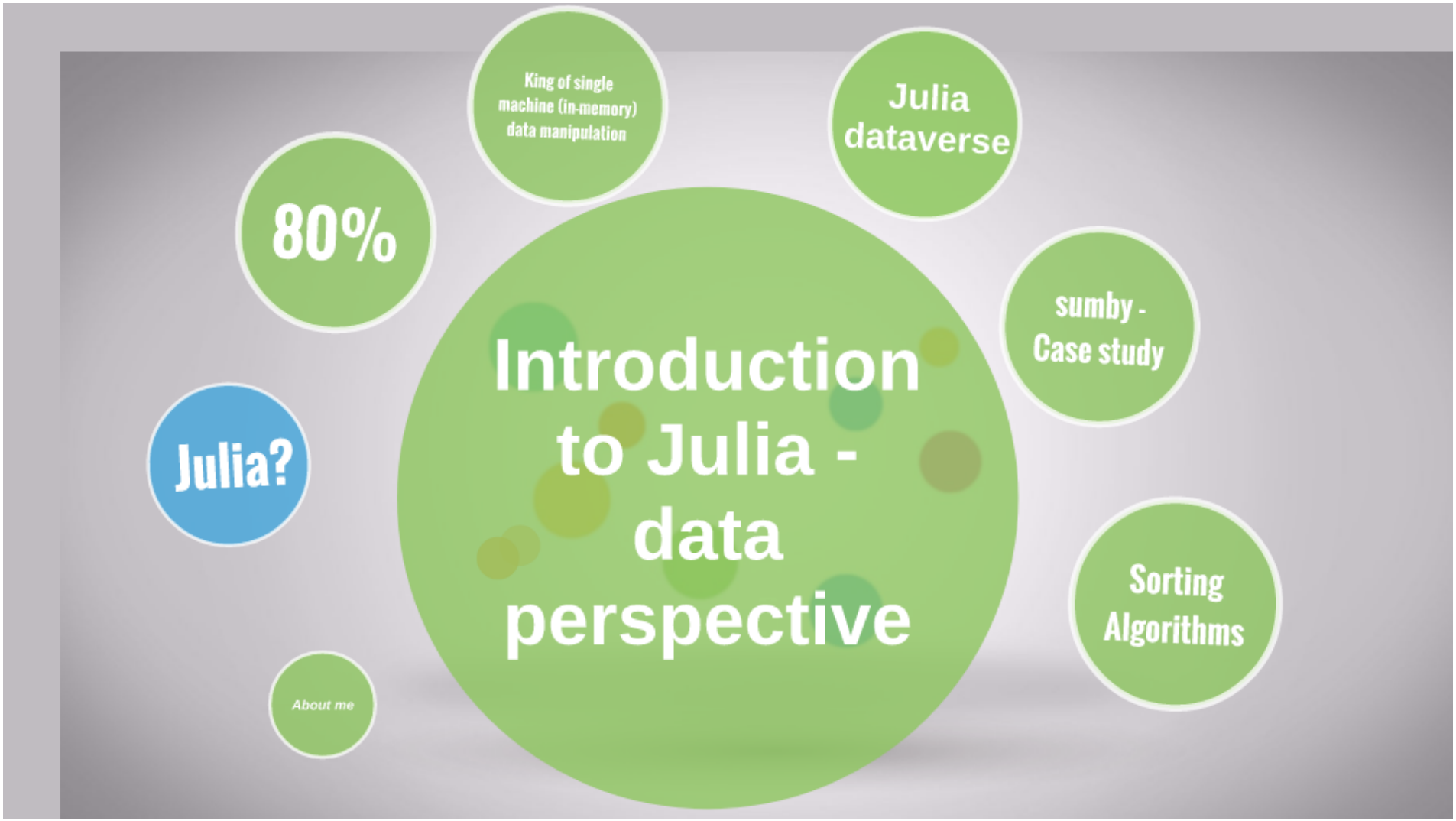
How does this impact DataFrames?

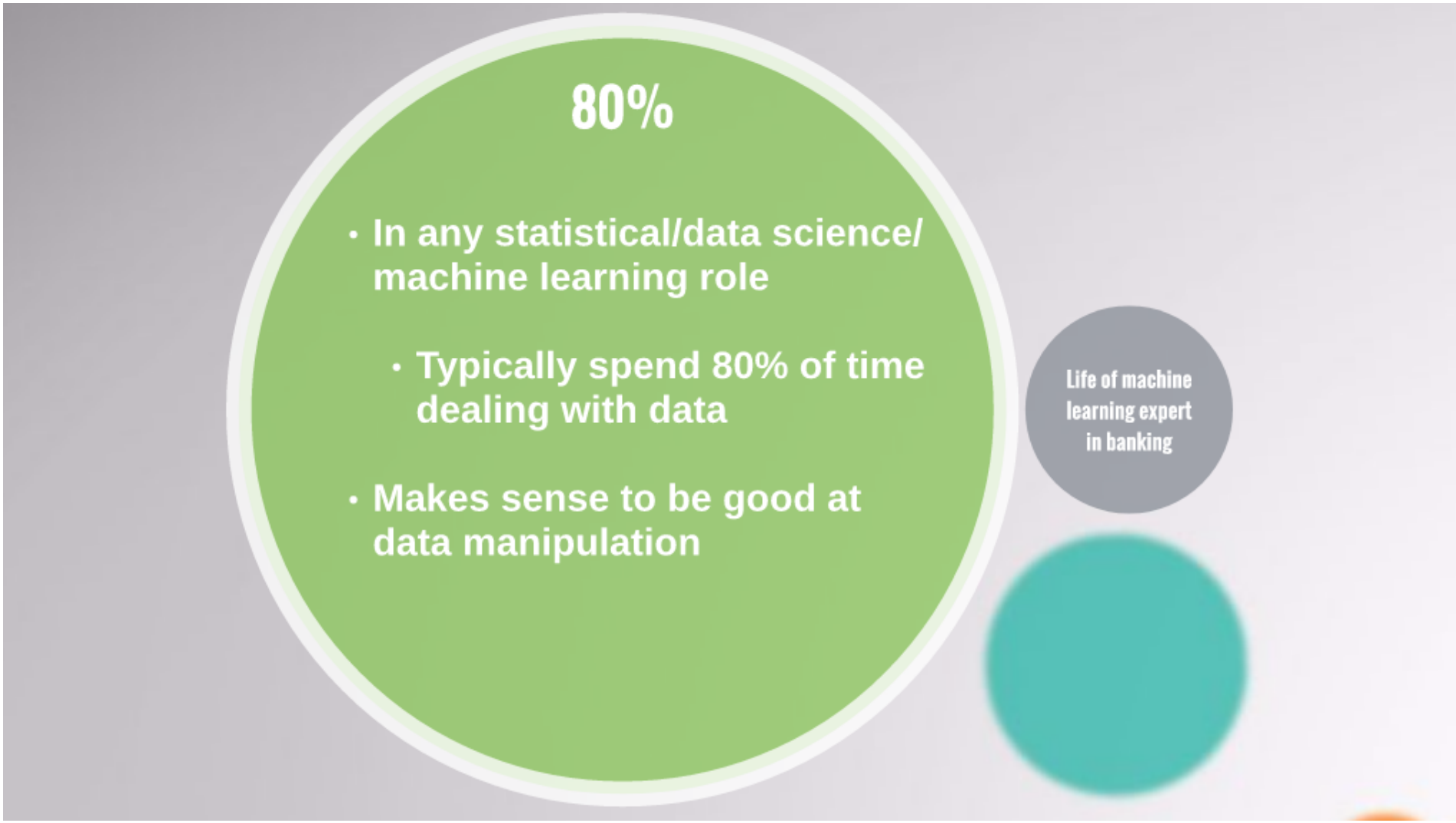
Nice syntax

Python inspired

Easy to read

Easy to write

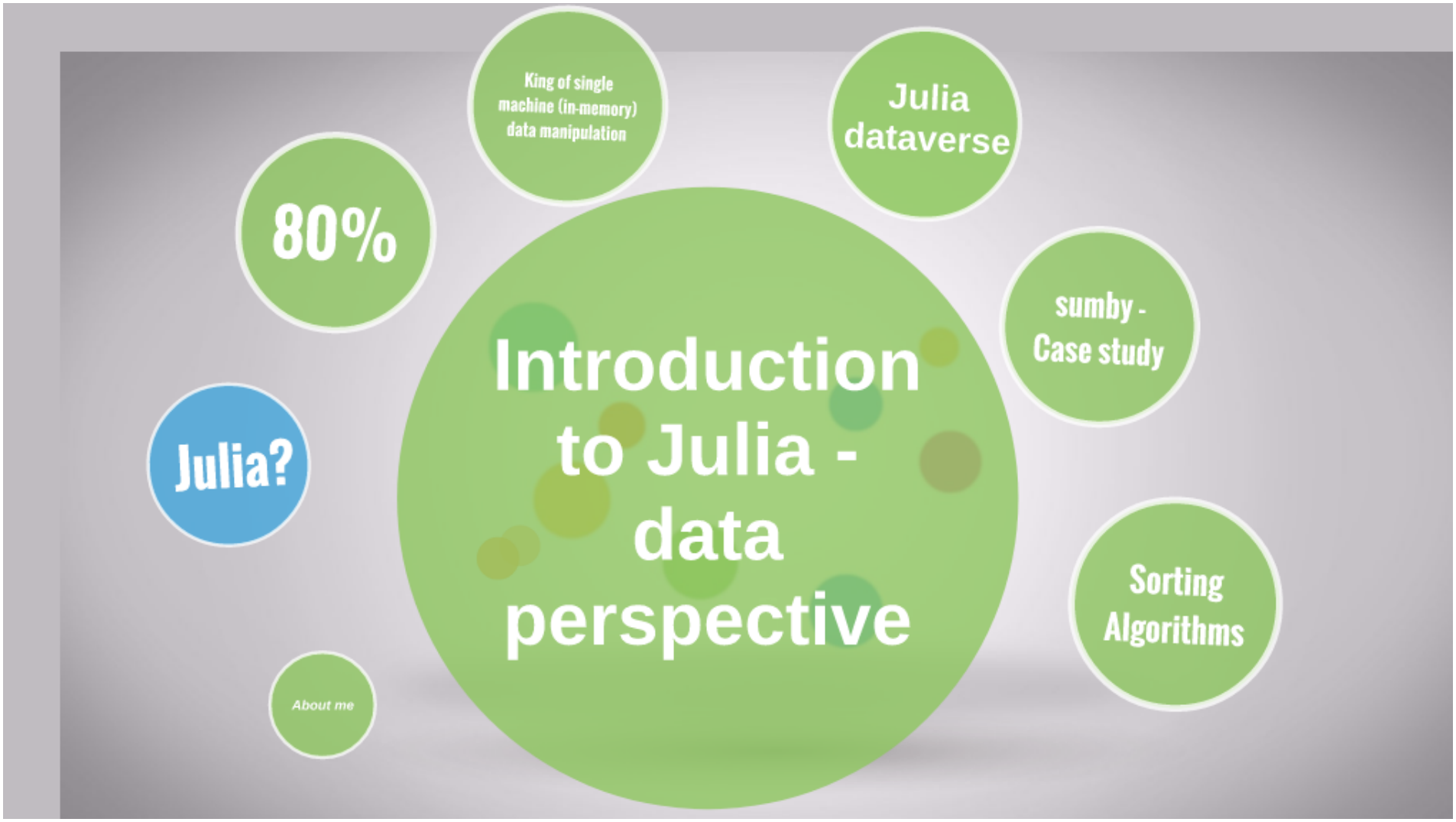

King of single machine (in-memory) data manipulation
Julia dataverse
80%
sumby - Case study
Introduction to Julia - data perspective
Julia?
Sorting Algorithms
About me

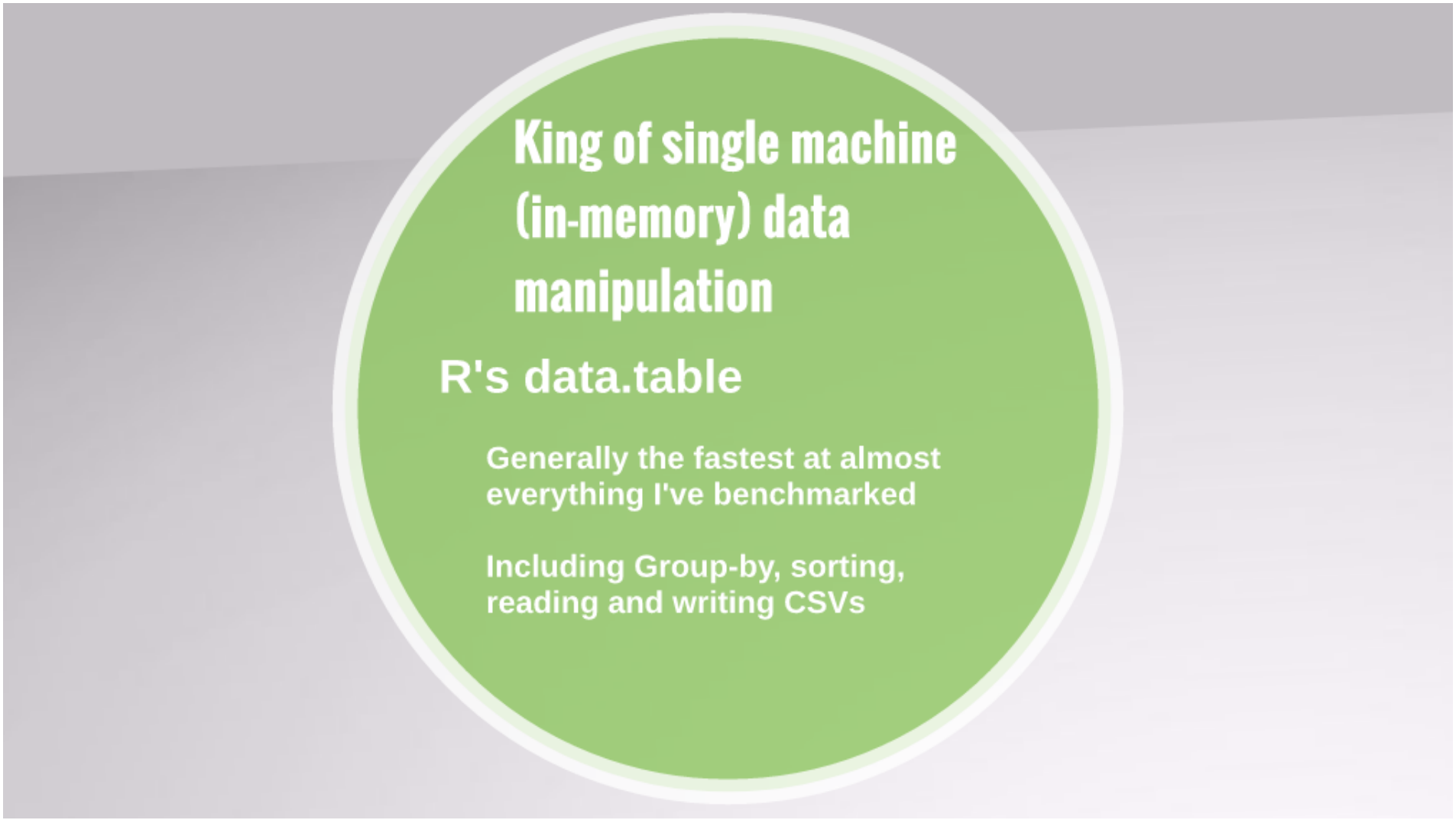

80%
• In any statistical/data science/ machine learning role
• Typically spend 80% of time dealing with data
• Makes sense to be good at data manipulation
Life of machine learning expert in banking

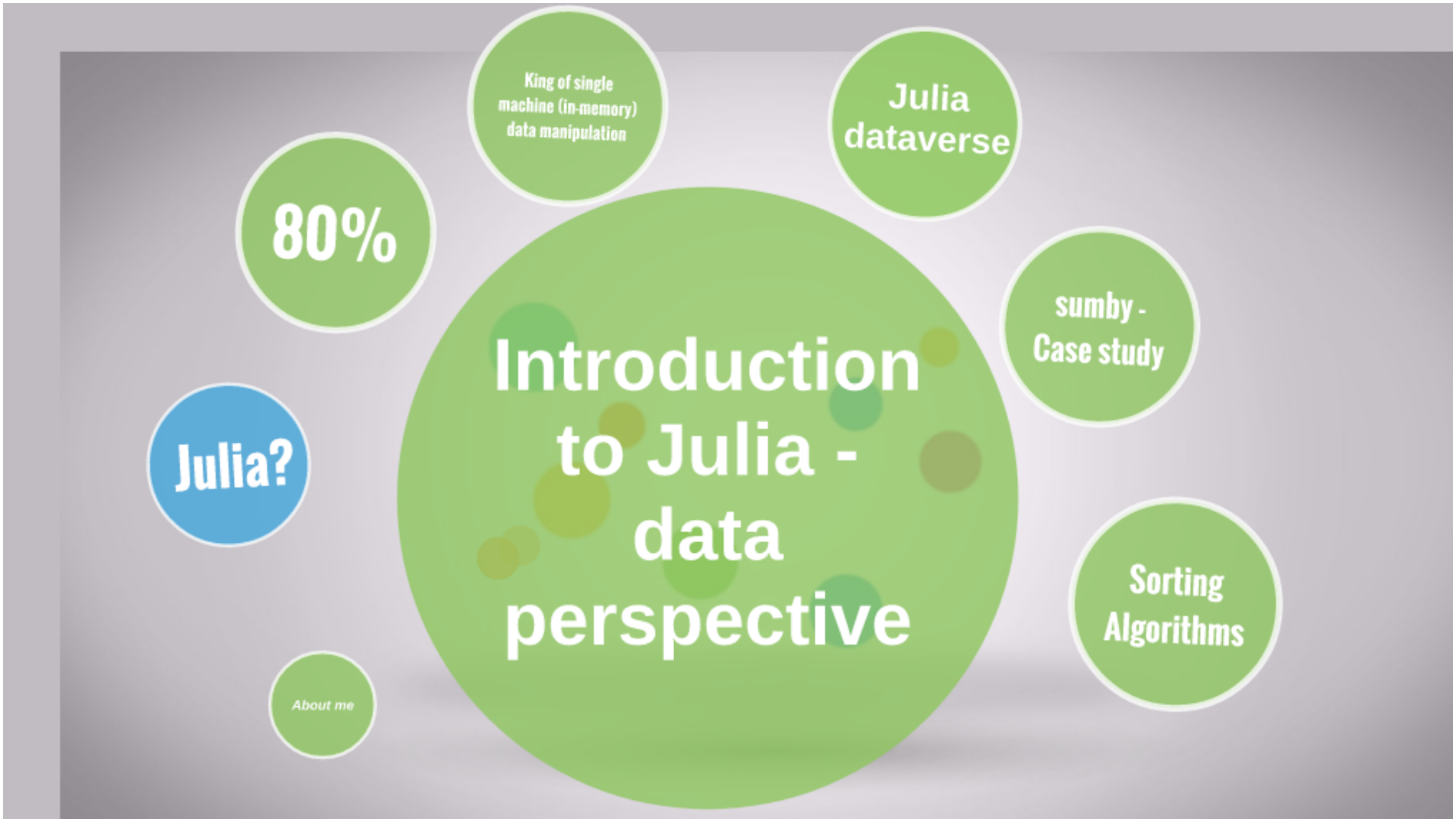
Life of machine learning expert in banking

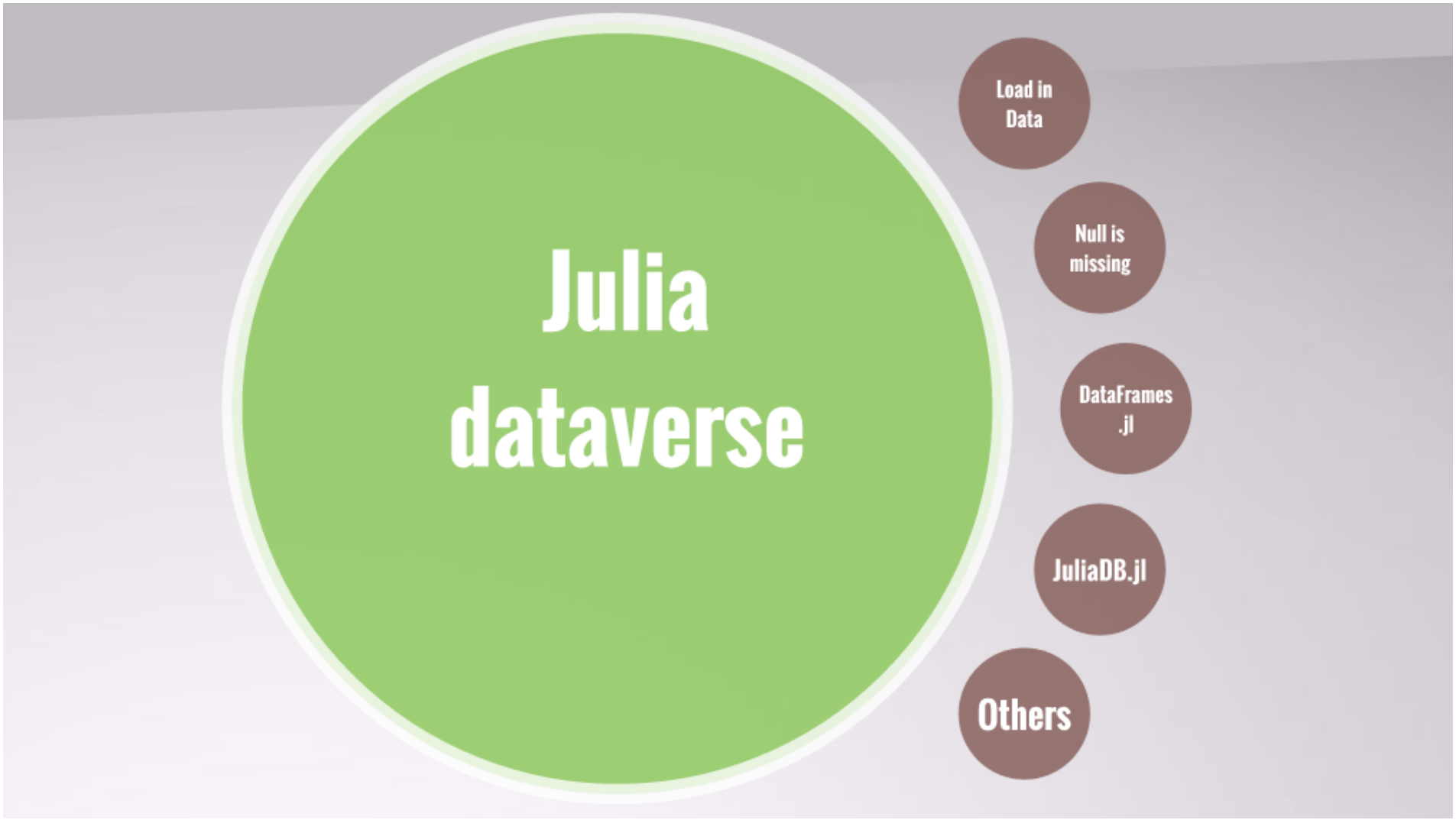
"Because finance data is so tricky, Amen suggests that one of the best uses for machine learning in a financial context is simply the cleaning and processing of data."

https://news.efinancialcareers.com/au-en/301629/machine-learning-finance/


King of single machine (in-memory) data manipulation
Julia dataverse
80%
sumby - Case study
Introduction to Julia - data perspective
Julia?
Sorting Algorithms
About me


King of single machine (in-memory) data manipulation
R's data.table
Generally the fastest at almost everything I've benchmarked
Including Group-by, sorting, reading and writing CSVs


King of single machine (in-memory) data manipulation
Julia dataverse
80%
sumby - Case study
Introduction to Julia - data perspective
Julia?
Sorting Algorithms
About me


Julia
dataverse
Load in Data
Null is missing
DataFrames .jl
JuliaDB.jl
Others

Load in Data

JDBC & ODBC & HDF5

CSV.jl

TextParse.jl

uCSV.jl
- Commendable for Elm-inspired error messages

Feather.jl

Null is missing

- R has NA which is used for missing values
- Julia has missing

```
using Missings | ✓

1 + missing | missing

true | missing | true

true & missing | missing
```

Should use Missings.missing to represent null values

Nullable and Nulls are outdated

Should use Missings.missing to represent null values

```
1 + missing # missing

true & missing # missing

true | missing # true
```

Nullable and Nulls are outdated

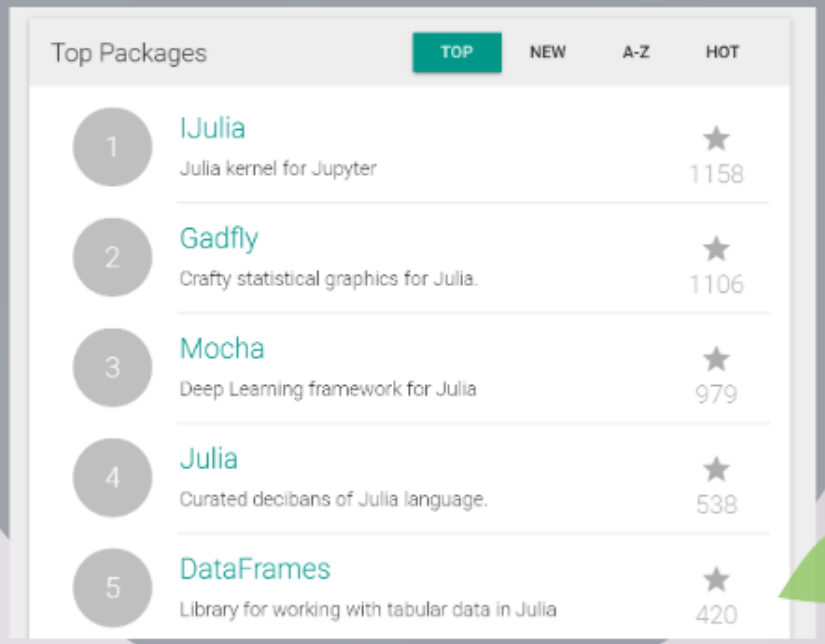
- Literature before Dec 2017 may refer to Nullable or Nulls
- Those are obsolete
- Use Missings.jl's missing instead

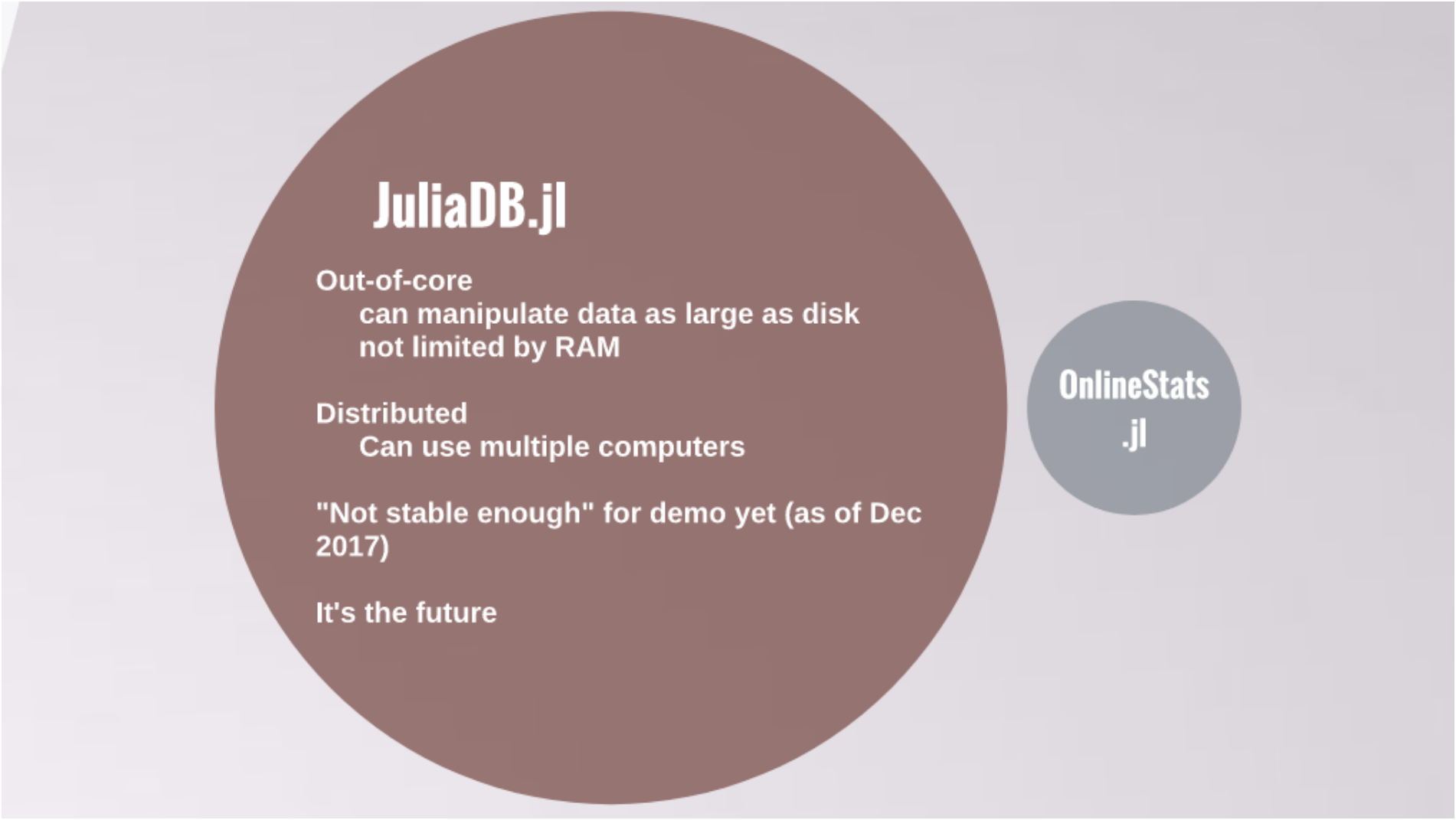
DataFrames.jl

- Most Starred repo on dealing with tabular data
- But is it "type-stable"?
- Can only deal with data in memory

5th most starred

5th most starred




JuliaDB.jl
Out-of-core
can manipulate data as large as disk
not limited by RAM
Distributed
Can use multiple computers
"Not stable enough" for demo yet (as of Dec 2017)
It's the future
OnlineStats .jl

OnlineStats.jl

- Algorithms that work on one chunk of data at a time
  - avoid RAM limit issues and
  - enable parallelism
- Allows Julia to eventaully take on SAS

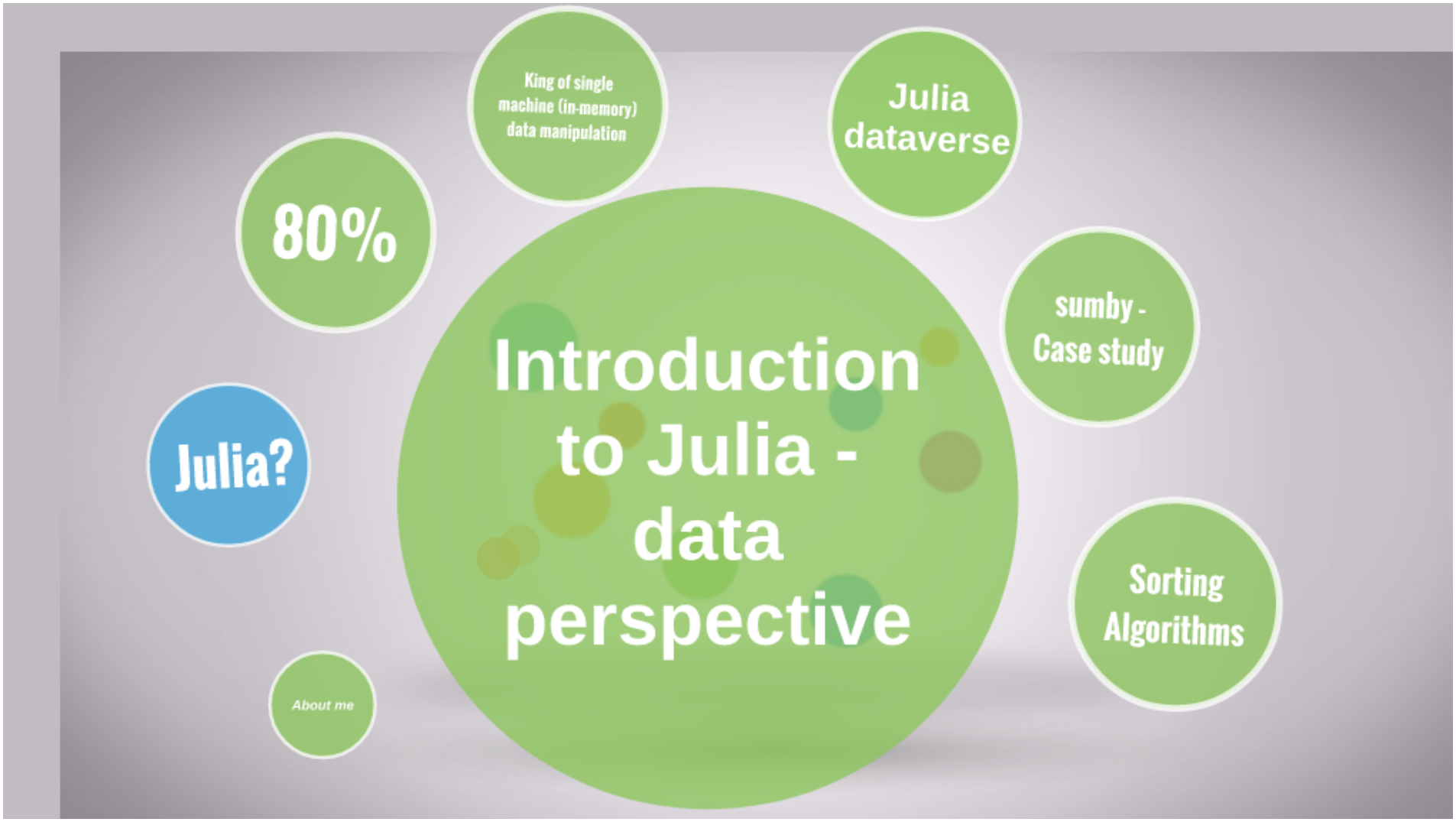
Others

ODBC & JDBC supported by JuliaComputing

IterableTables.jl
- convert between table-types easily

IndexedTables.jl
- type-stable? tables

Query.jl
- a LINQ or dplyr-esque framework
- seems "slow"


King of single machine (in-memory) data manipulation
Julia dataverse
80%
sumby - Case study
Introduction to Julia - data perspective
Julia?
Sorting Algorithms
About me

sumby - Case study

The only benchmark published by data.table is on "grouping"
https://github.com/Rdatatable/data.table/wiki/Benchmarks-:-Grouping

In SQL:

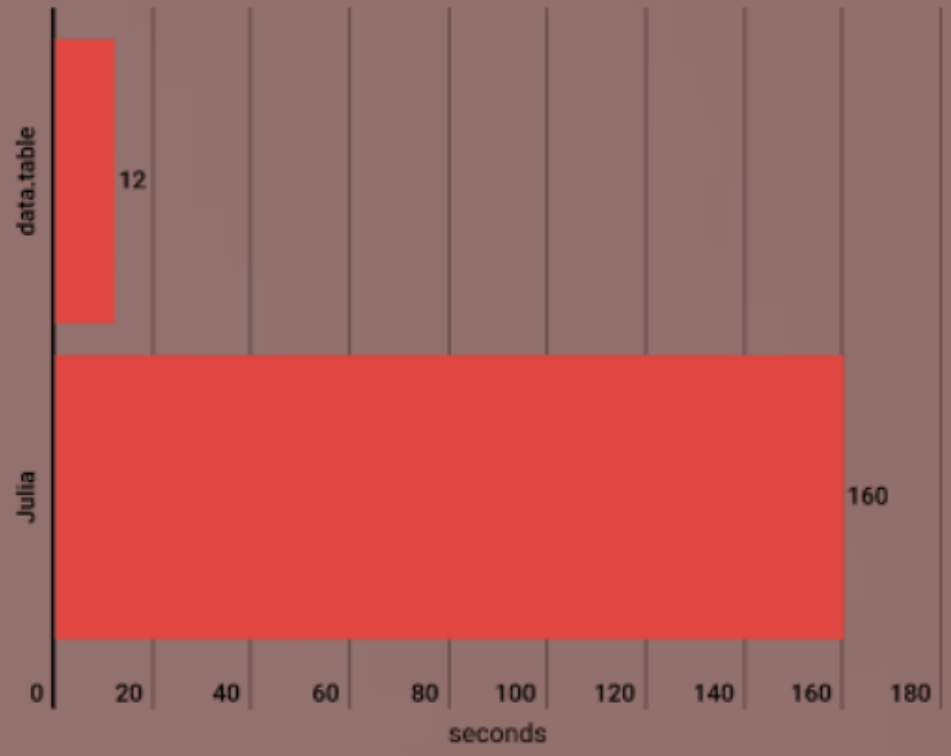
```
select
    sum(val)
from
    table
group by
    grp_col
```

Benchmark: sum-by large number of groups

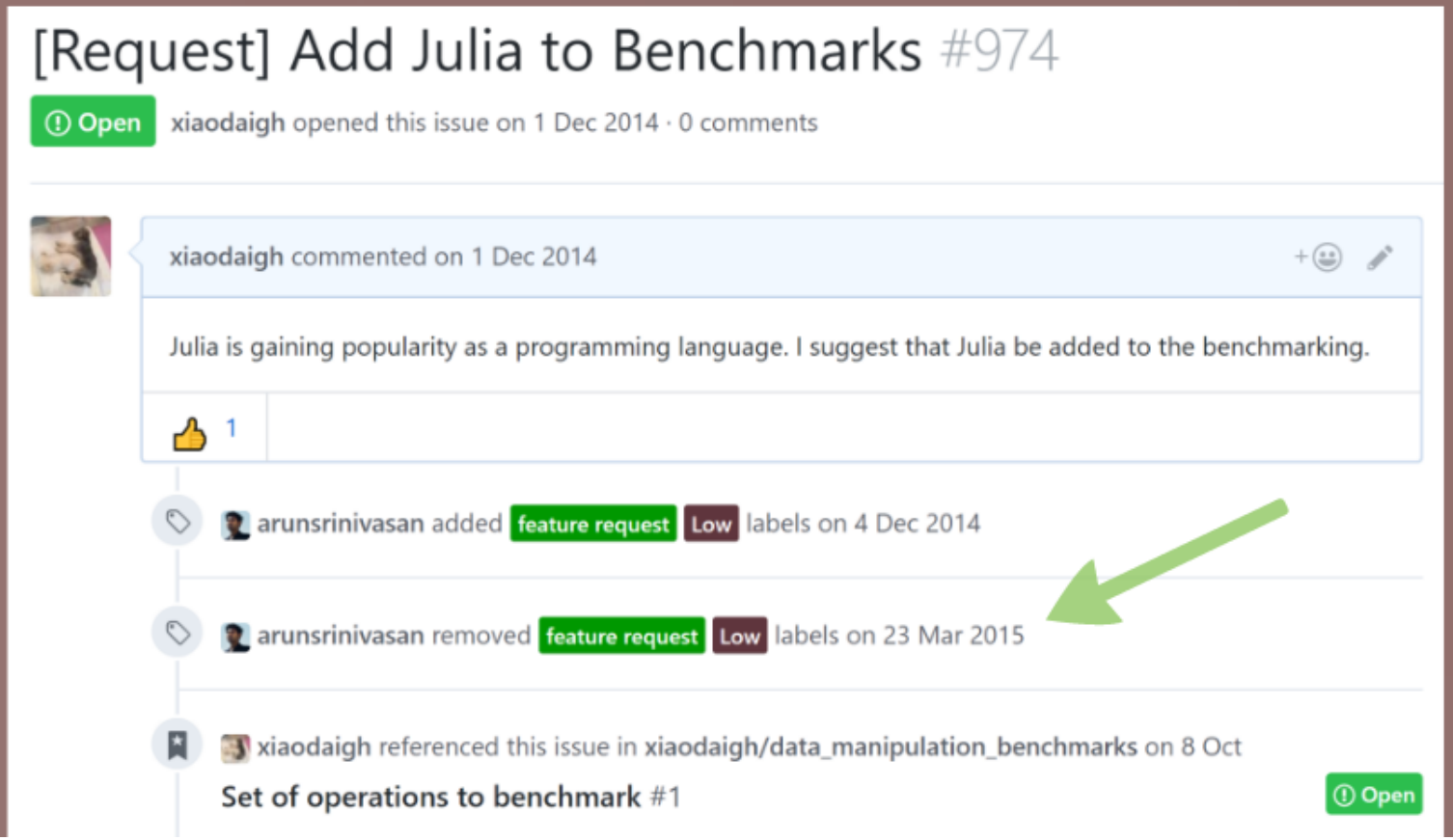
Can Julia be the fastest too?

Benchmark: sum-by large number of groups

- 200 million rows
- 2 million distinct groups



I did try to be lazy



data.table is C backed

Basically it's Julia vs R via C

data.table is generally hard to beat

Shout out to @MattDowle and @arun_sriniv + 20 others for such an amazing package

Shoulders of giants

Read somewhere that data.table uses radixsort

Found radixsort in SortingAlgorithms.jl

Radixsort was written entirely in Julia; so easy to pick-up

Re-purposed the algorithm for sum-by

Now it's implemented in FastGroupBy.jl

Radixsort & Multithreading

Can sort "bits" type in O(N) time

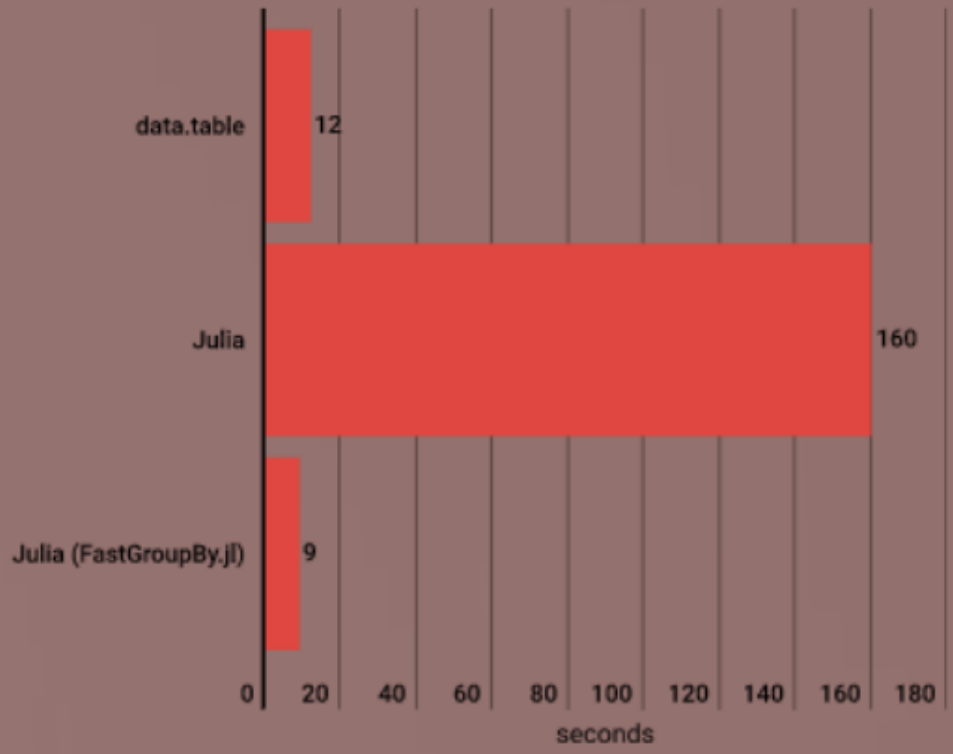
Uses more memory as it makes a copy of the original vector

It's not a comparison-based sort

Other comparison-based algorithms are O(N*logN)

Used multithreading to work on mutually-exclusive chunks of the array to further speed-up

Using FastGroupBy.jl



Demo

https://github.com/Julia-Sydney/Julia-Sydney-Talks/tree/master/20171207

Misc

- FastGroupBy.jl's sumby is *faster* when
  - the by-vector is categorical and we know the categories before hand
  - the by-vector is already sorted (or more precisely just grouped)

- FastGroupBy.jl's sumby is *slower* when
  - the vector is not 100 million+ in length (but "no par")
  - by-vector is not bits type e.g. String

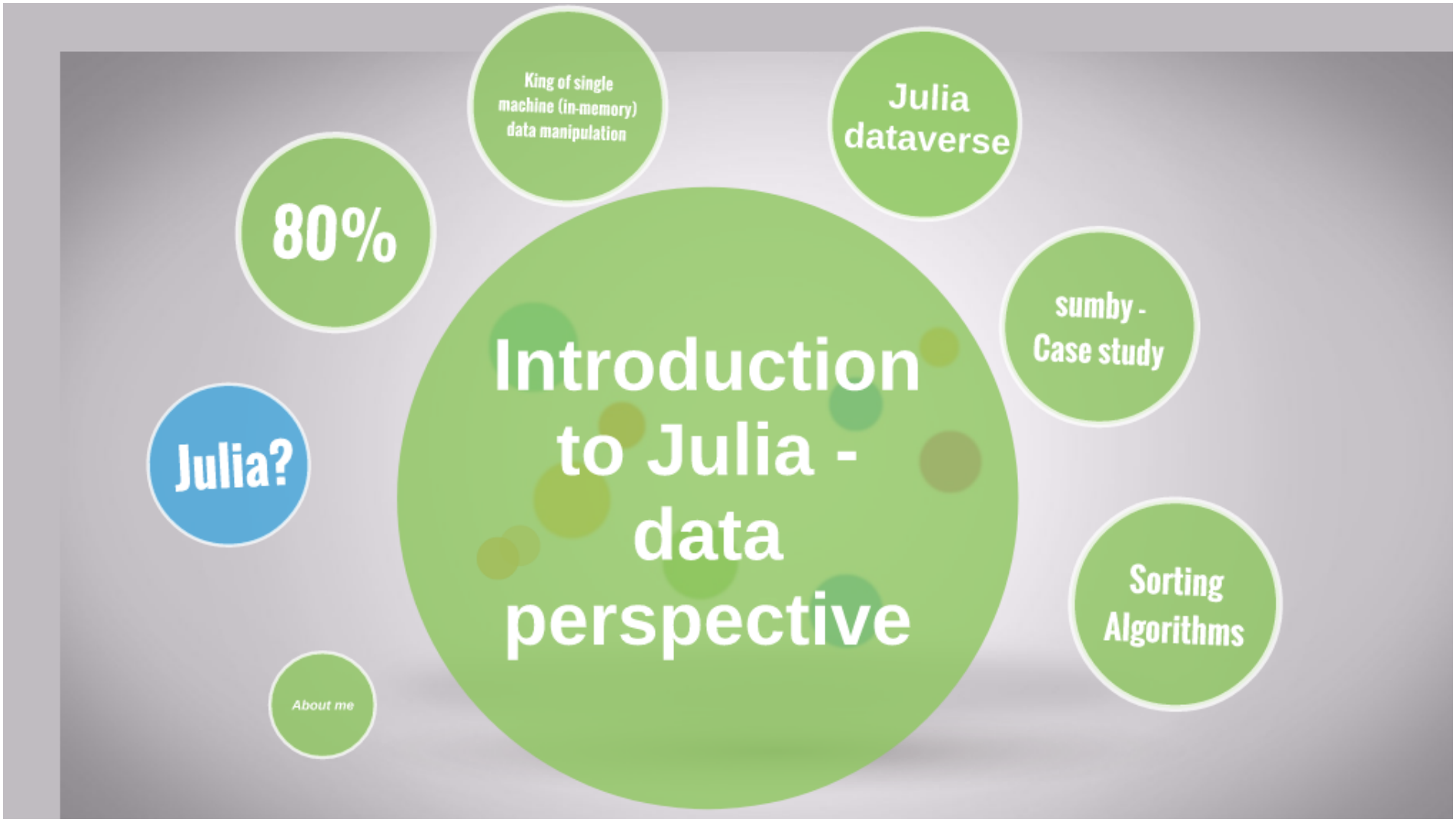
- Blog: https://www.codementor.io/zhuojiadai/an-empirical-study-of-group-by-strategies-in-julia-dagnosell

Can Julia be the fastest too?

I think it can but...

It will take time for people to find inefficiencies and implement better algorithms

We have data.table as the beacon


King of single machine (in-memory) data manipulation
Julia dataverse
80%
sumby - Case study
Introduction to Julia - data perspective
Julia?
Sorting Algorithms
About me

Sorting Algorithms

Julia's sorting algorithms are not yet well optimized

Comparison-based sort is O(nlogn)

- **let n be size fo input**
- **time_to_sort(n) = f(nlogn) for some *linear* function f**
- **time_to_sort(n) = a + b*nlogn; size of a and b matters!**

Julia comes with 3 sorting algorithms

- **Quicksort - One of best "on average"**
- **Mergesort - Fast but requires more memory**
- **InsertionSort - very quick for small data size**

Joking Algorithms

Random Sort - "Wanna play pick up 52 cards?

Sleep Sort - Interesting!

String sorting/grouping is a key pillar of data science

Group By operations is one of the most common

- Being able to groups items together is key

Group By in SQL

```
select
        grps,
        sum(val) as tot_val
  from
        some_table
  group by
        grps;
```

String-sorting in R very fast

R

- sort(x, method="radix")

Julia

- No string radix. I implemented radixsort(x)
- about 20% slower than R

Why is R so fast?

R's strings unlike Julia's are stored "centrally" in a cache (look up table)
Using a clever algorithm, it only needs to sort the unique strings

If the proportion of uniques to total is large; then R's advantage vanishes

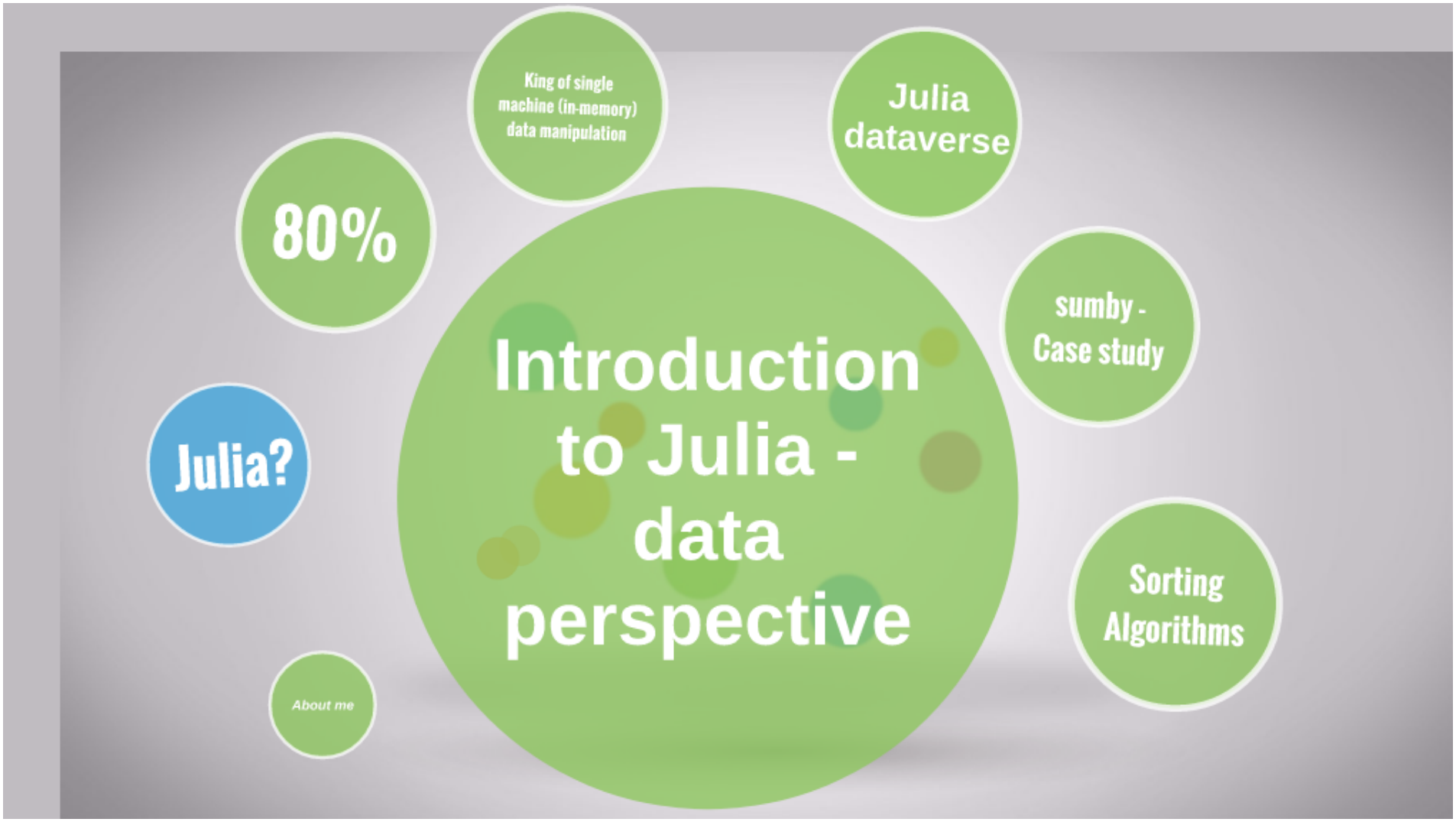
The future

- InternedStrings.jl - Similar to how srings are stored in R
- Use CategoricalArrays.jl - specialised fast algorithm for group-by
  - equivalent of R's factor type
  - Grouping known categories before hand
  - Can program fast paths for easy functions like sum
    - this is done in data.table

SortingAlgorithms.jl

SortingAlgorithms.jl provides 3 (soon to be 4) more

- TimShot - the default in Python, exploits common sequences
- HeapSort - uses a heapstructure
- RadixSort - R's default; very fast if implemented right
- StringRadixSort - My contributions sorts strings fast


King of single machine (in-memory) data manipulation
Julia dataverse
80%
sumby - Case study
Introduction to Julia - data perspective
Julia?
Sorting Algorithms
About me