An R Interface to Hail

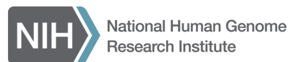
Michael Lawrence (Genentech)

May 31, 2019

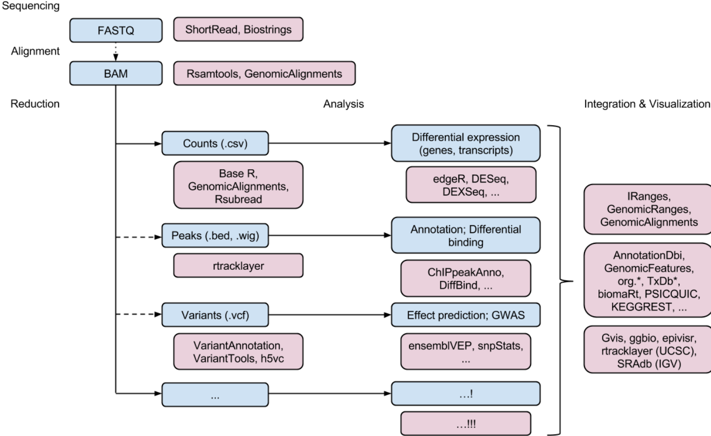
- A platform and language for distributed genomics on Apache Spark
- Provides:
  - General tabular data manipulation,
  - Specialized, scalable data structures stored in Parquet, and
  - Efficient implementation of domain-specific algorithms in C++
- Designed to be interfaced through high-level languages
  - Python interface comes out of the box

Bioconductor is an R platform for integrative genomics

- Started 2002
- Led by Martin Morgan
- Core infrastructure maintained by about 8 people, based in Roswell Park CRC in Buffalo, NY
- >2000 R software packages that form a unified platform
- Well-used and respected.
  - 53k unique IP downloads / month.
  - 21,700 PubMedCentral citations.

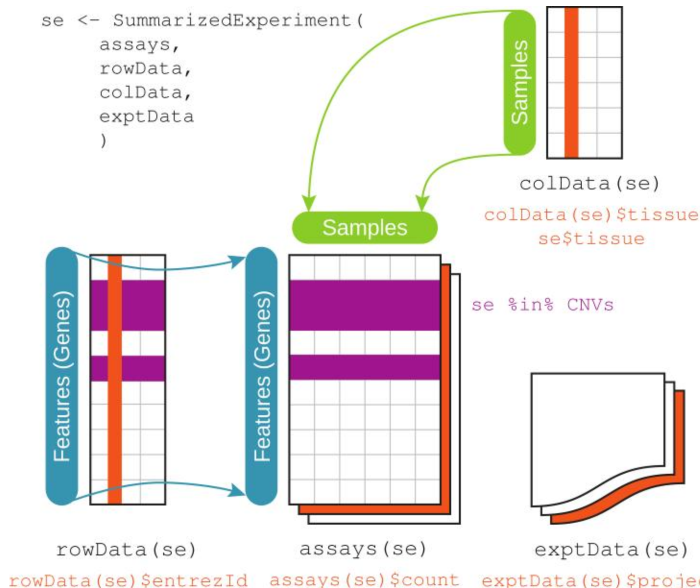


Bioconductor is an integrated platform



Hail and Bioconductor share their central data type

A SummarizedExperiment (Bioc) or MatrixTable (Hail)



The hailr package

hailR

HailDataFrame, HailExperiment, HailPromise

SparkObject

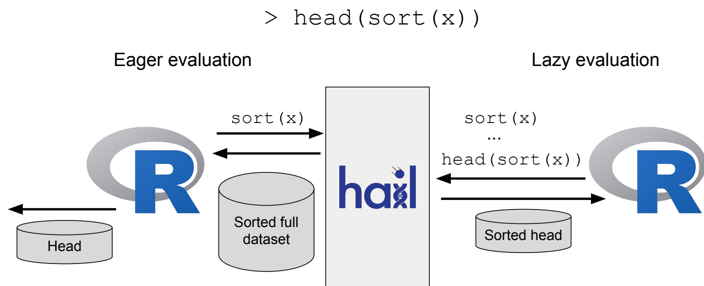
SparkDriver

sparklyr | SparkR? | Other?

hail

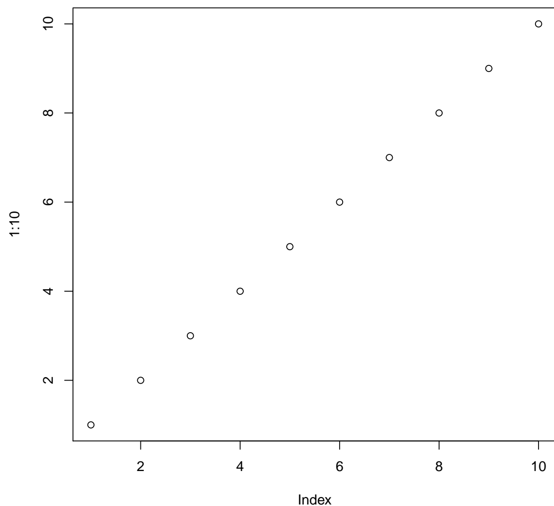
Apache Spark

hailr pushes compute to Hail via *lazy evaluation*



R already uses lazy evaluation

```
plot(1:10)
```



R function call arguments are promises

```r
fun <- function(arg) substitute(arg)
fun(1:10)
```

```
1:10
```

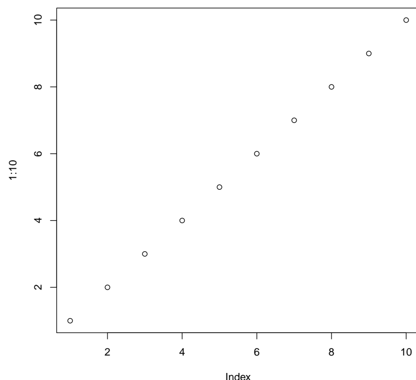
Lazy evaluation delays work until absolutely necessary

```
fun <- function(arg) {
    z <- arg
    substitute(z)
}
fun(1:10)
```

```
[1]  1  2  3  4  5  6  7  8  9 10
```

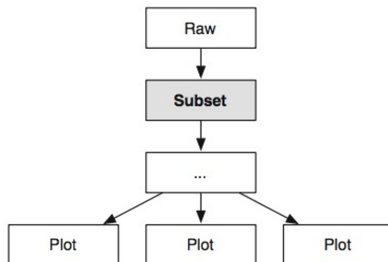
It often pays to be lazy

- Provenance capture
- Interactive graphics pipelines, streaming
- Responsiveness through asynchronicity
- Optimization
- Compact representations
- Querying databases and files
- Distributed computing



It often pays to be lazy

- Provenance capture
- Interactive graphics pipelines, streaming
- Responsiveness through asynchronicity
- Optimization
- Compact representations
- Querying databases and files
- Distributed computing



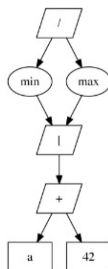
It often pays to be lazy

- Provenance capture
- Interactive graphics pipelines, streaming
- Responsiveness through asynchronicity
- Optimization
- Compact representations
- Querying databases and files
- Distributed computing

```
# time = 0:00.000
future(trainModel(Sonar, "Class")) %...>%
  print() # time = 0:15.673
# time = 0:00.062
```

It often pays to be lazy

- Provenance capture
- Interactive graphics pipelines, streaming
- Responsiveness through asynchronicity
- Optimization
- Compact representations
- Querying databases and files
- Distributed computing



It often pays to be lazy

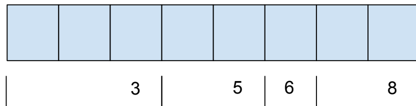
- Provenance capture
- Interactive graphics pipelines, streaming
- Responsiveness through asynchronicity
- Optimization
- Compact representations
- Querying databases and files
- Distributed computing

New in R 3.5:

```
system.time(1:1e12)
```

```
user   system elapsed
   0        0       0
```

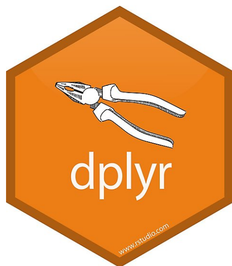
CompressedList in S4Vectors:



`splitAsList(1:8, rep(1:4, c(3, 2, 1, 2))`

It often pays to be lazy

- Provenance capture
- Interactive graphics pipelines, streaming
- Responsiveness through asynchronicity
- Optimization
- Compact representations
- Querying databases and files
- Distributed computing

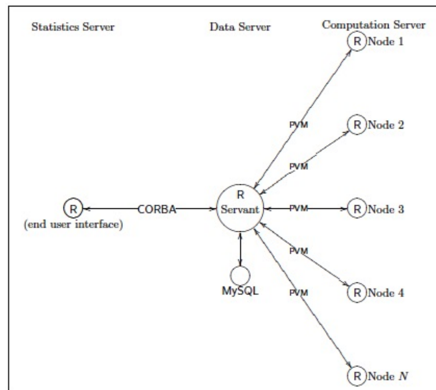


DelayedArray

platforms all | downloads top 5% | posts 3 / 0.3 / 0.7 / 0 | in Bioc 1.5 years

It often pays to be lazy

- Provenance capture
- Interactive graphics pipelines, streaming
- Responsiveness through asynchronicity
- Optimization
- Compact representations
- Querying databases and files
- Distributed computing



Deferred data structures allow for eager evaluation

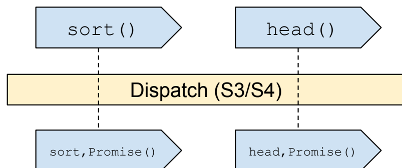
For some promise "`x`":

```
> head(sort(x))
```

Deferred data structures allow for eager evaluation

For some promise “x”:

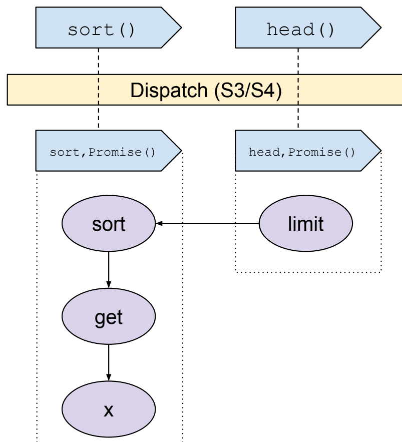
```
> head(sort(x))
```



Deferred data structures allow for eager evaluation

For some promise "x":

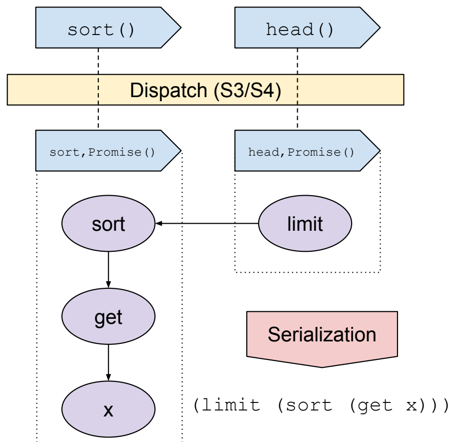
```
> head(sort(x))
```



Deferred data structures allow for eager evaluation

For some promise "x":

```
> head(sort(x))
```

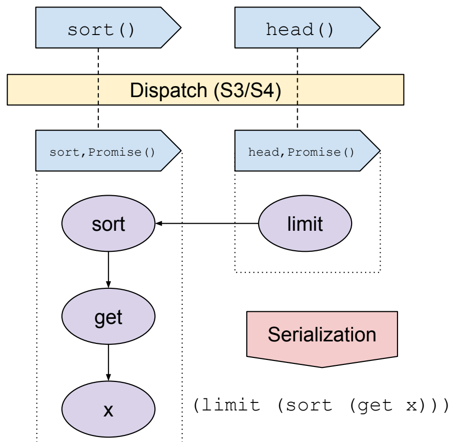



```
(limit (sort (get x)))
```

Deferred data structures allow for eager evaluation

For some promise “x”:

```
> head(sort(x))
```



Zhang, Herodotou, Yang (2009) RIOT: I/O-Efficient Numerical Computing without SQL. https://arxiv.org/abs/0909.1766

Programmability is fundamental

DSLs should rest on programmatic API:

```
> df %>% arrange(x) %>% head()
```

DSL implementation

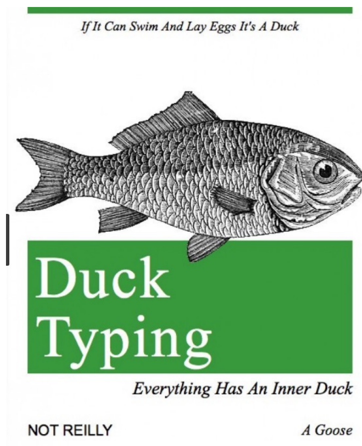
sort() head()

Dispatch (S3/S4)

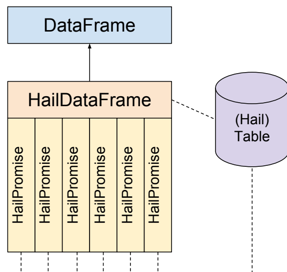
sort,Promise() head,Promise()

Bioconductor containers rely on base API to be generic

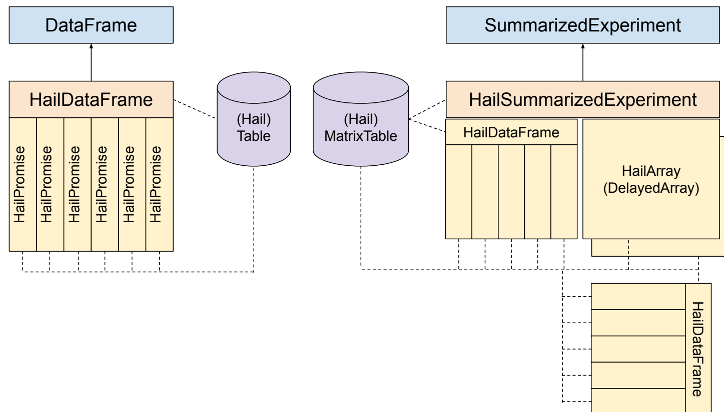
- Bioconductor containers assume elements implement key functions from the base API
  - *DataFrame* allows anything "vector-like" to be a column
  - *SummarizedExperiment* allows anything "matrix-like" to hold assay values
- Since our promises implement the base API, they just work
- But we still want to map DataFrame operations to Hail Table operations



hailr is a hierarchical extension of Bioconductor



hailr is a hierarchical extension of Bioconductor



We can load data into Hail

Directly from a text file:

```
library(hailr)
data_dir <- system.file("extdata", package="hailr")
tsv1 <- file.path(data_dir, "kt_example1.tsv")
df <- readHailDataFrameFromText(tsv1, header=TRUE)
```

Copying from an R data.frame:

```
df <- copy(read.table(tsv1, header=TRUE), hail())
```

. . . and get it back out

```
df
```

```
HailDataFrame with 4 rows and 8 columns
            ID            HT            SEX             X             Z
  <Int32Promise> <Int32Promise> <StringPromise> <Int32Promise> <Int32Promise>
1             1            65               M             5             4
2             2            72               M             6             3
3             3            70               F             7             3
4             4            60               F             8             2
            C1            C2            C3
  <Int32Promise> <Int32Promise> <Int32Promise>
1             2            50             5
2             2            61             1
3            10            81            -5
4            11            90           -10
```

```
df$ID
```

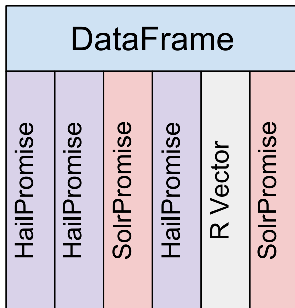
```
[1] 1 2 3 4
```

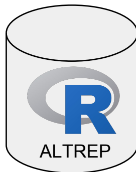
A glimpse into the compiler

```
as.character(df$ID@expr)
```

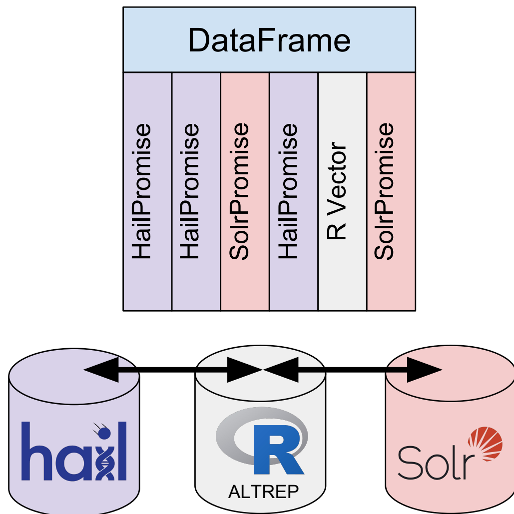
```
[1] "(GetField ID (Ref row))"
```

Abstractions enable mixed evaluation



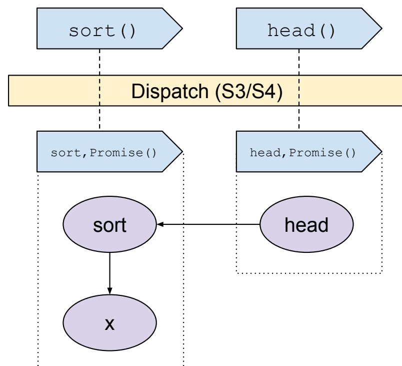


Abstractions enable mixed evaluation



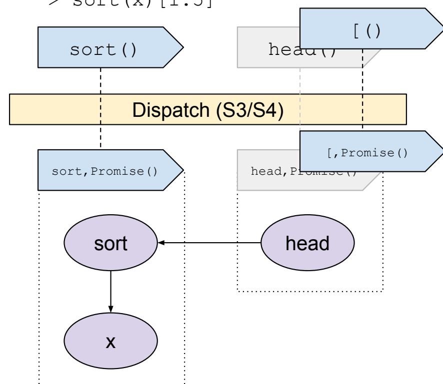
Looking forward: generalized, integrated compute

Intermediate algebra, optimization with backend-informed cost model



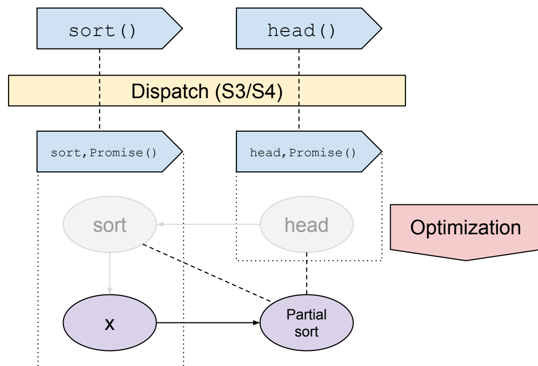
Looking forward: generalized, integrated compute

Intermediate algebra, optimization with backend-informed cost model



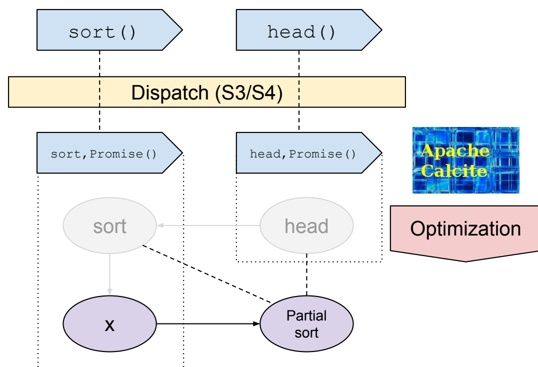
Looking forward: generalized, integrated compute

Intermediate algebra, optimization with backend-informed cost model



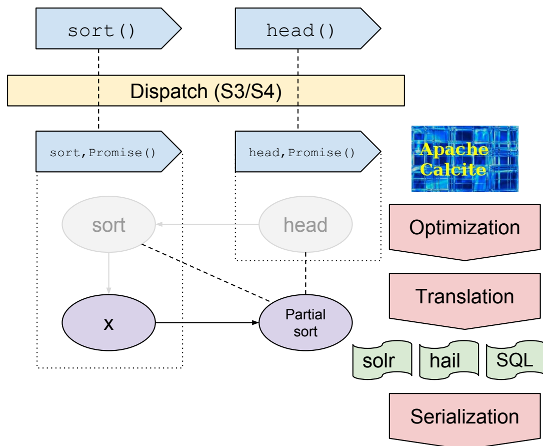
Looking forward: generalized, integrated compute

Intermediate algebra, optimization with backend-informed cost model



Looking forward: generalized, integrated compute

Intermediate algebra, optimization with backend-informed cost model



Acknowledgements

- Javier Luraschi and Kevin Kuo (responsive sparklyr support)
- Samuel Macêdo (recent contributions)
- Cotton Seed (leader of Hail project)