ScootR: Scaling R Dataframes on Dataflow Systems

Andreas Kunft[1] Lukas Stadler[2] Daniele Bonetta[2] Cosmin Basca[2] Jens Meiners[1]

Sebastian Breß[1] Tilmann Rabl[1] Juan Fumero[3] Volker Markl[2]

Technische Universität Berlin[1] Oracle Labs[2] University of Manchester[3]

SPONSORED BY THE

Federal Ministry of Education and Research

DFG

ORACLE

R gained increased traction

- Dynamically typed, open-source language
- Rich support for analytics & statistics

R gained increased traction

- Dynamically typed, open-source language
- Rich support for analytics & statistics

But

- Standalone R is not well suited for out-of-core data loads

Analytics pipelines often work on large amounts of raw data

- Dataflow engines (DF), e.g., Apache Flink and Spark, scale-out
- Provide rich support for user-defined functions (UDFs)

Analytics pipelines often work on large amounts of raw data

- Dataflow engines (DF), e.g., Apache Flink and Spark, scale-out
- Provide rich support for user-defined functions (UDFs)

But

- R users are often unfamiliar with DF APIs and concepts

Combine the usability of R with the scalability of dataflow engines

- Goals
- From functions calls to an operator graph
- Approaches to execute R UDFs
- Our Approach: ScootR
- Evaluation

GOALS

1. **Provide data.frame API with *natural* feeling**

   - `df    <- select(df, count = flights, distance)`
   - `df$km <- df$miles * 1.6`
   - `df    <- apply(df, func)`

GOALS

1. **Provide data.frame API with *natural* feeling**

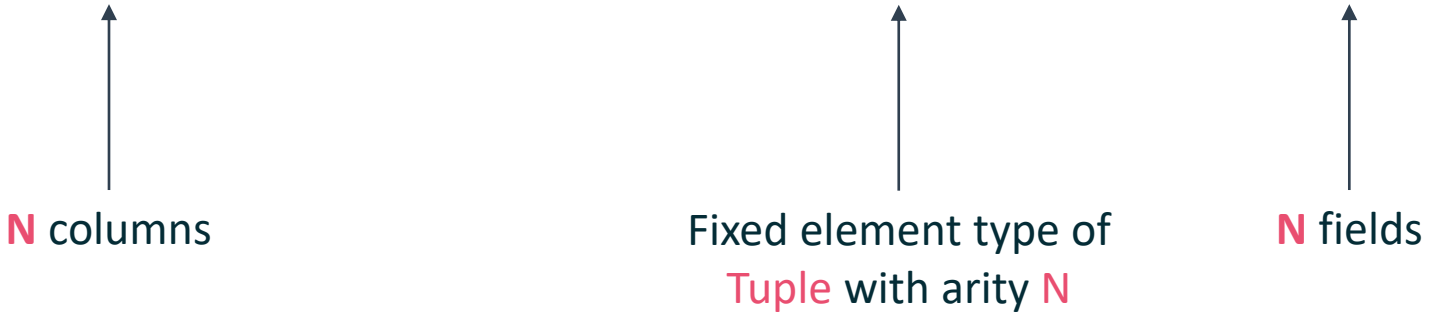
   - `df    <- select(df, count = flights, distance)`
   - `df$km <- df$miles * 1.6`
   - `df    <- apply(df, func)`

2. **Achieve comparable performance to native dataflow API**

From function calls to an operator graph

MAPPING DATA TYPES

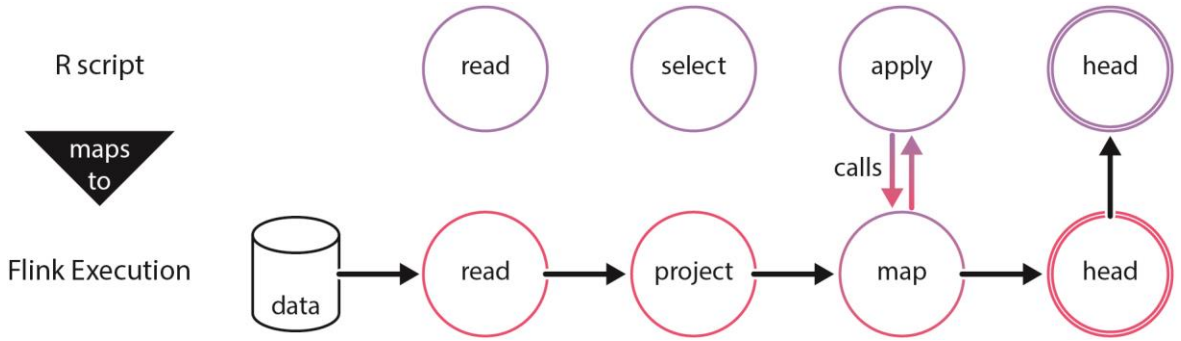
- R `data.frame(`T_1`,`T_2`,…,`T_N`)` as Flink `DataSet<TupleN<`T_1`,`T_2`,…,`T_N`>>`



- E.g., `data.frame(integer, character)` as `DataSet<Tuple2<Integer, String>>`

MAPPING R FUNCTIONS TO OPERATORS

- Functions on data.frames *lazily* build an operator graph

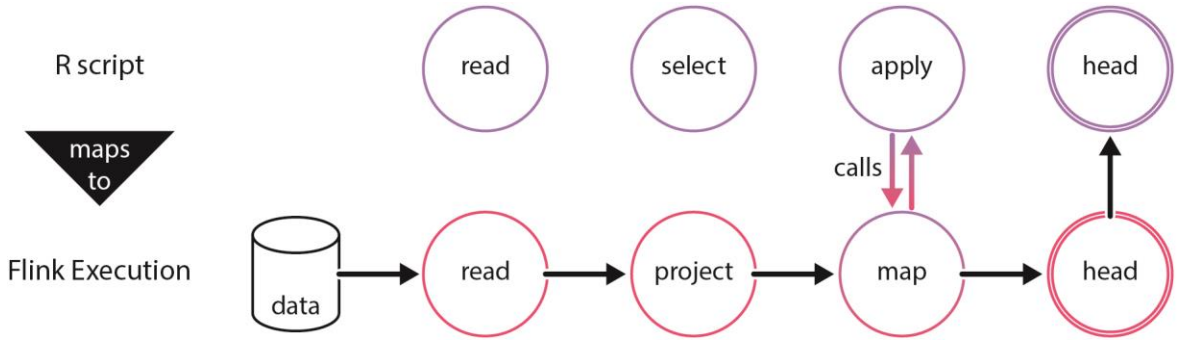


MAPPING R FUNCTIONS TO OPERATORS

- Functions on data.frames *lazily* build an operator graph

1. Functions w/o UDFs are handled before execution, e.g., a `select` function is mapped to a `project` operator

```
select(df$id, df$arrival) to ds.project(1, 3)
```

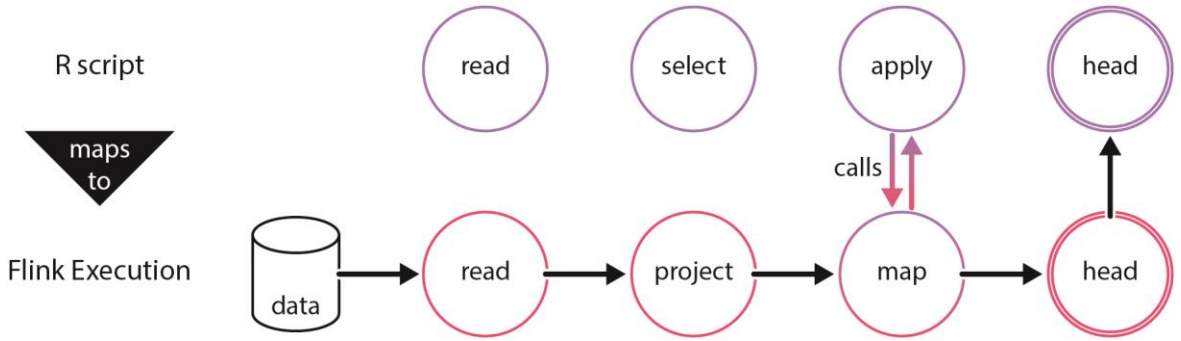


MAPPING R FUNCTIONS TO OPERATORS

- Functions on data.frames *lazily* build an operator graph

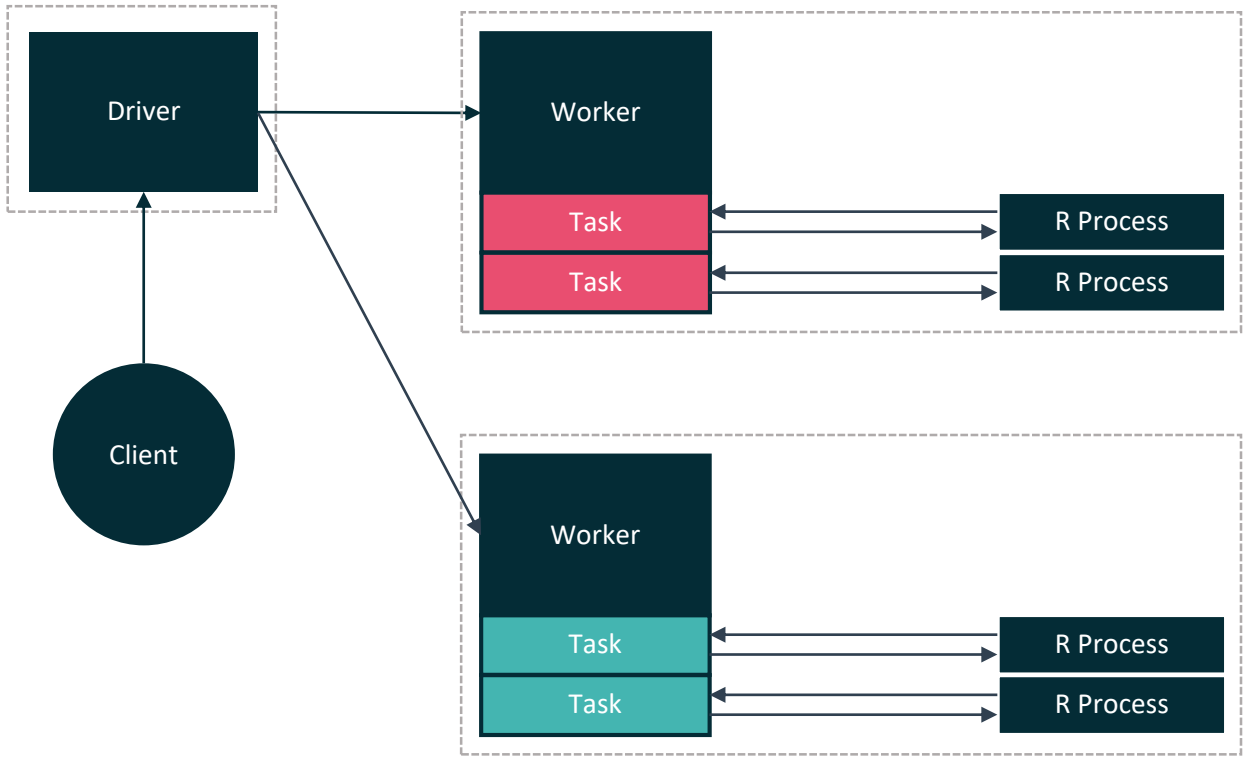
1. Functions w/o UDFs are handled before execution

2. Functions w/ UDFs call **R functions** during execution

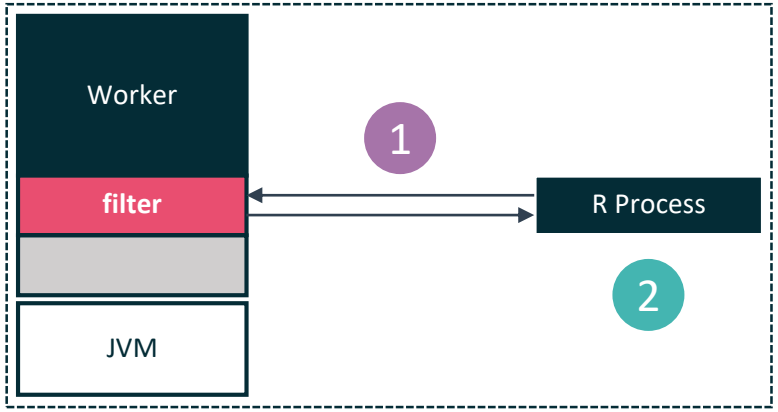


Approaches to execute R UDFs

INTER PROCESS COMMUNICATION (IPC)



INTER PROCESS COMMUNICATION (IPC)




```
filter <- function(df) {
  df$language == 'english'
}
```

1. Communication + Serialization (R <> Java)
2. JVM and R compete for memory

SOURCE-TO-SOURCE TRANSLATION (STS)

- Translate **restricted** set of functions to native dataflow API
- Constant translation overhead, but native execution performance

SOURCE-TO-SOURCE TRANSLATION (STS)

- E.g., STS translation in SparkR to Spark's Scala Dataframe API:

```
df <- filter(df,
  df$language == 'english'
)
```

```
val df = df.filter($"language" === "english")
```

```
df$km <- df$miles * 1.6
```

```
val df = df.withColumn("km", $"miles" * 1.6)
```

Inter Process Communication

+ Execute arbitrary R code

- Data serialization
- Data exchange
- Java and R process compete for memory

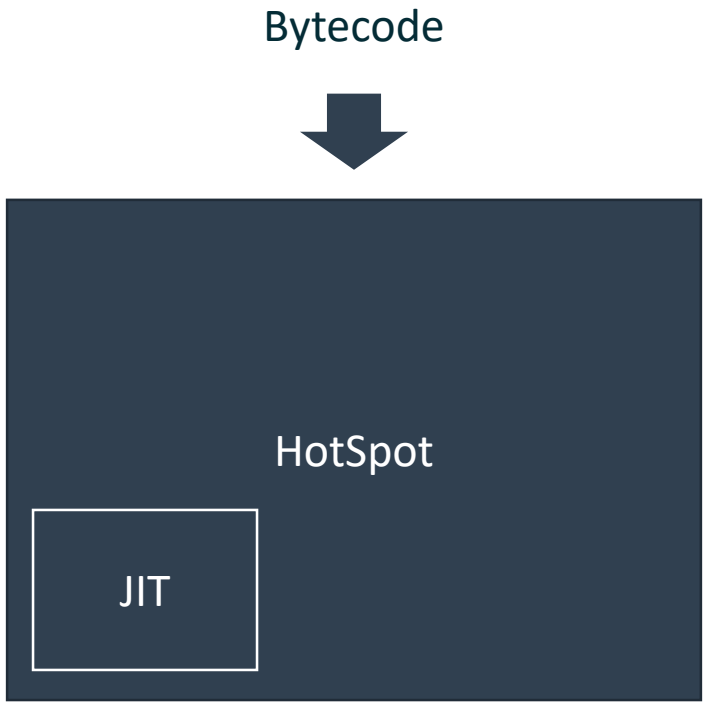
Source-to-source translation

+ Native performance

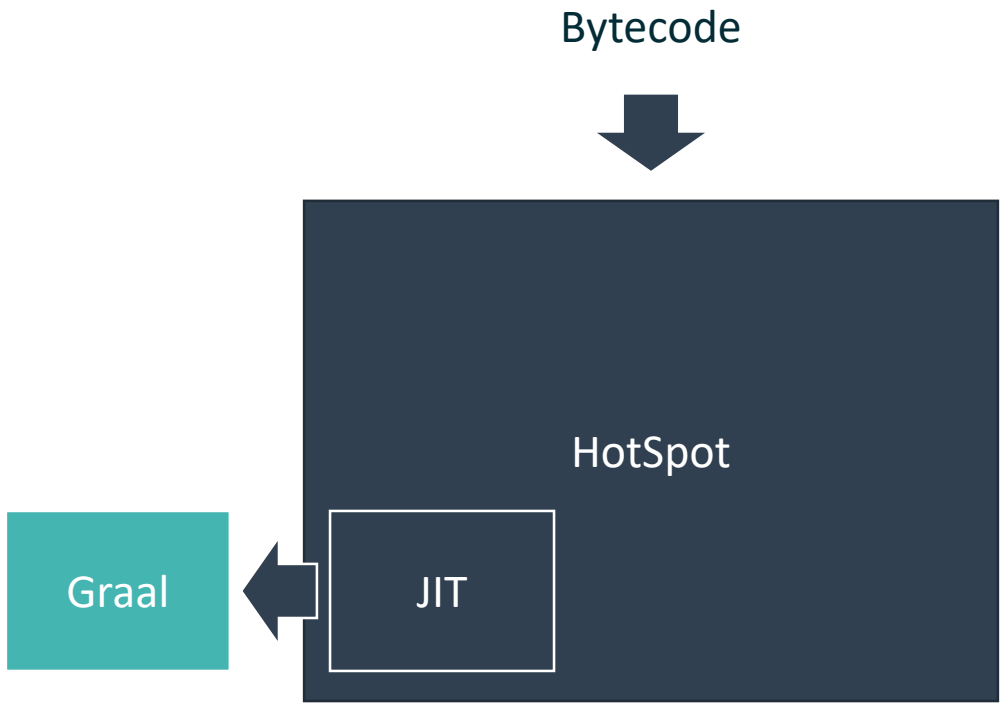
- Restricted to a language subset or requires to build full-fledged compiler

A common runtime for R and Java

BACKGROUND: TRUFFLE/GRAAL



BACKGROUND: TRUFFLE/GRAAL



BACKGROUND: TRUFFLE/GRAAL

BACKGROUND: TRUFFLE/GRAAL

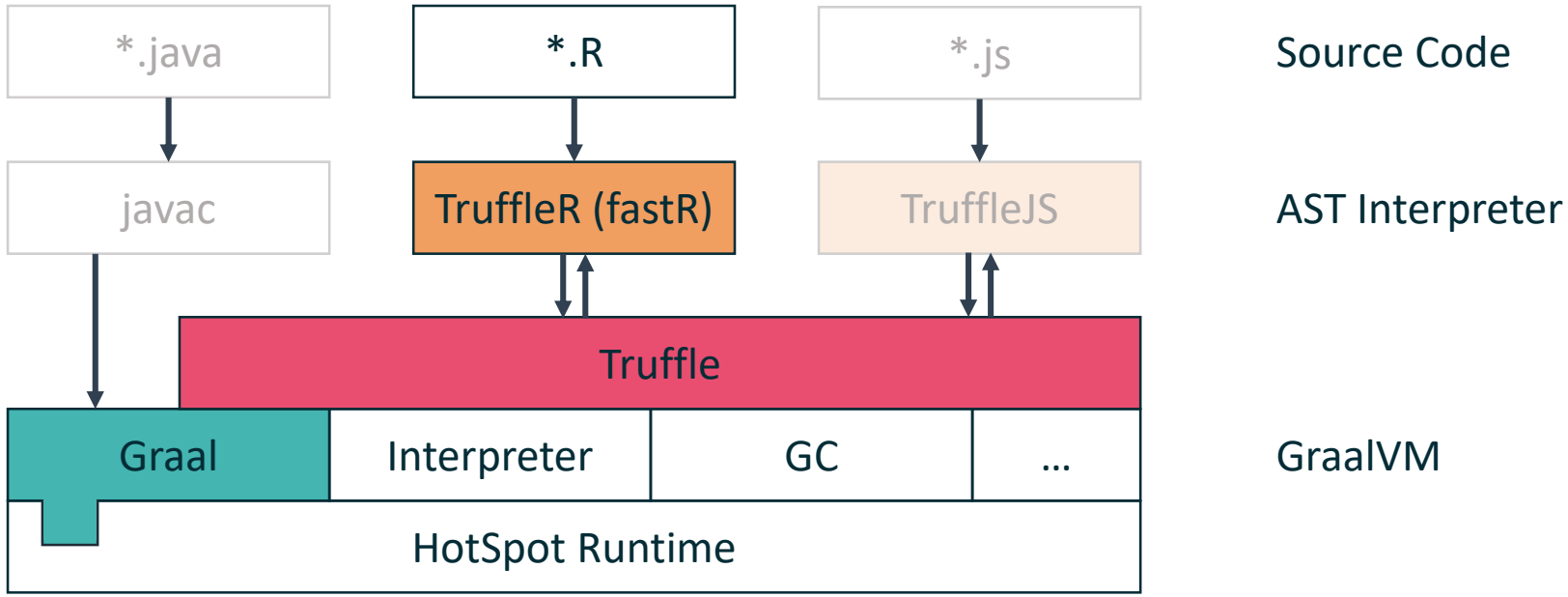
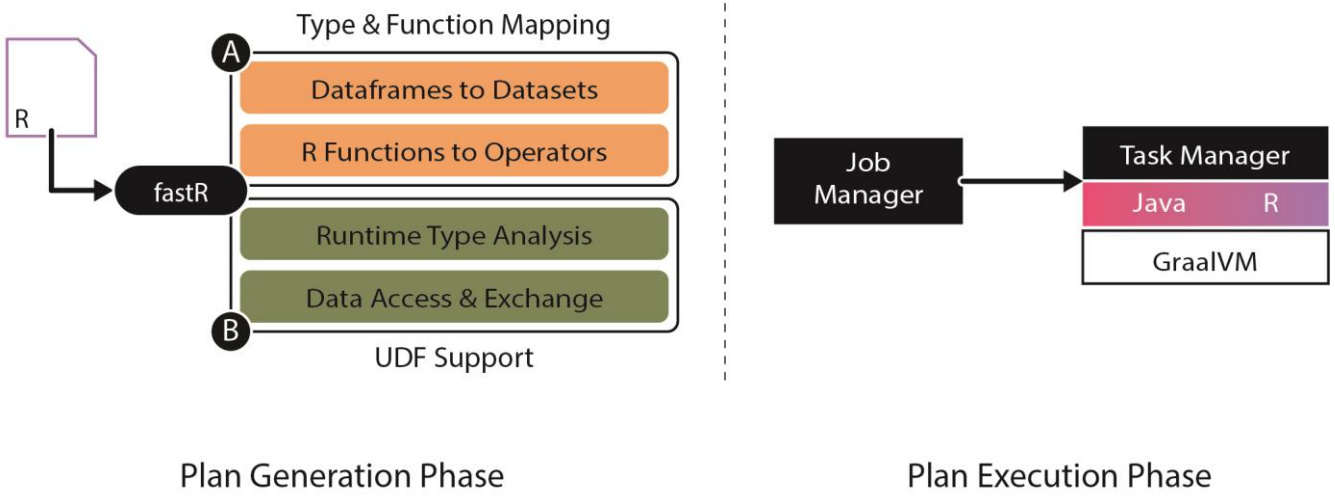


Figure based on: Grimmer, Matthias, et al. "High-performance cross-language interoperability in a multi-language runtime." *ACM SIGPLAN Notices*. Vol. 51. No. 2. ACM, 2015.

SCOOTR: FASTR + FLINK



SCOOTR OVERVIEW

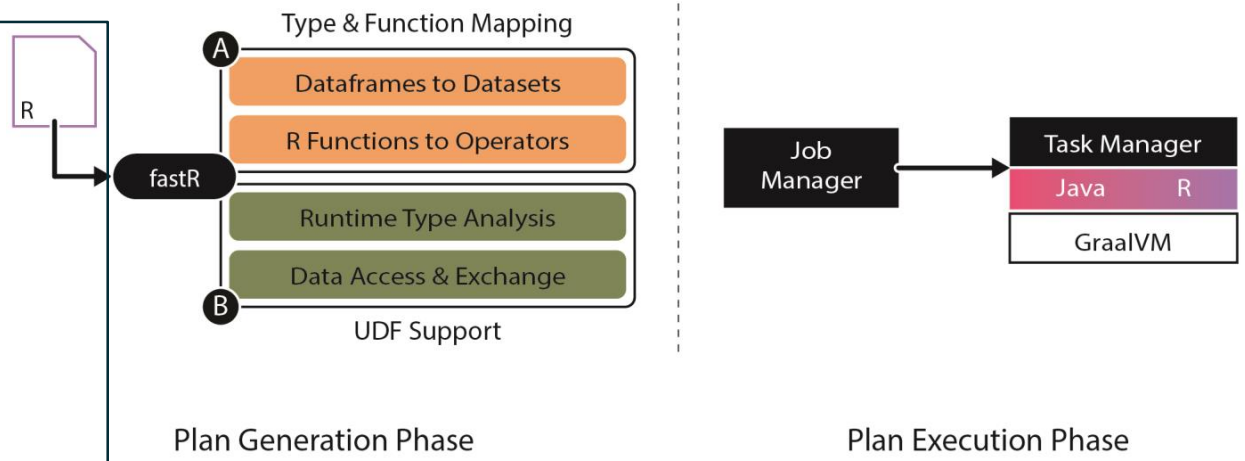
```
flink.init(SERVER, PORT)
flink.parallelism(DOP)

df <- flink.readdf(SOURCE,
  list("id", “body“, …),
  list(character, character, …)
)

words <- function(df) {
  len <- length(strsplit(df$body, " ")[[1]])
  list(df$id, df$body, len)
}

df    <- flink.apply(df, words)

flink.writeAsText(df, SINK)
flink.execute()
```



SCOOTR OVERVIEW

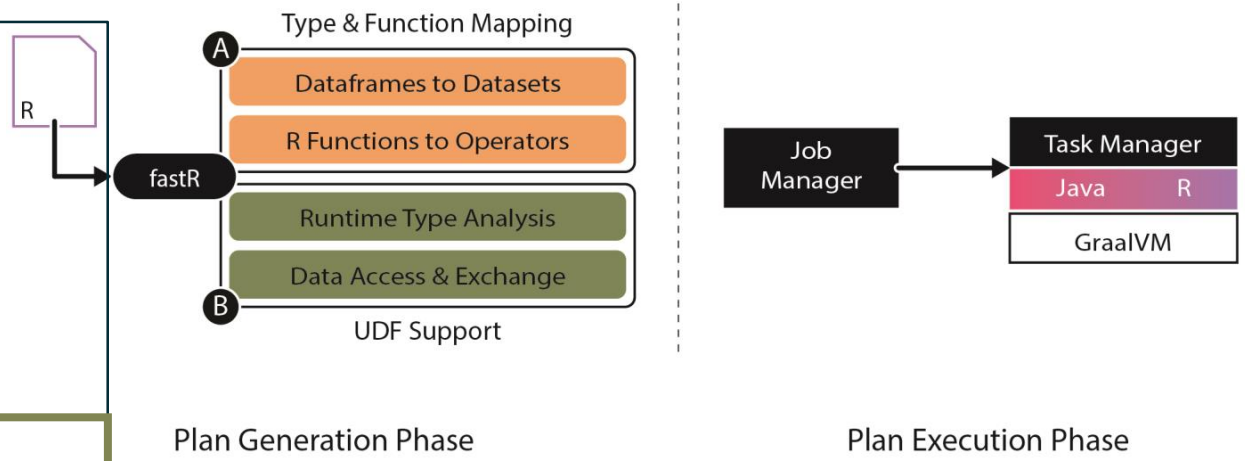
```
flink.init(SERVER, PORT)
flink.parallelism(DOP)

df <- flink.readdf(SOURCE,
  list("id", “body“, …),
  list(character, character, …)
)

words <- function(df) {
  len <- length(strsplit(df$body, " ")[[1]])
  list(df$id, df$body, len)
}

df    <- flink.apply(df, words)

flink.writeAsText(df, SINK)
flink.execute()
```



Efficient data access in R UDFs

```
function(df) {
  len <- length(strsplit(df$body, " ")[[1]])
  list(df$id, df$body, len)
}
```

```
function(df) {
  len <- length(strsplit(df$body, " ")[[1]])
  list(df$id, df$body, len)
}
```

```
function(tuple) {
  len <- length(strsplit(tuple[[2]], " ")[[1]])
  list(tuple[[1]], tuple[[2]], len)
}
```

---

1. Dataframe *proxy* keeps track of **columns** and provides efficient access

```
function(df) {
  len <- length(strsplit(df$body, " ")[[1]])
  list(df$id, df$body, len)
}
```

```
function(tuple) {
  len <- length(strsplit(tuple[[2]], " ")[[1]])
  flink.tuple(tuple[[1]], tuple[[2]], len)
}
```

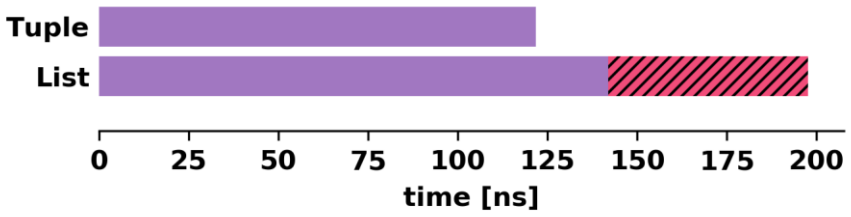
---

1. Dataframe *proxy* keeps track of **columns** and provides efficient access

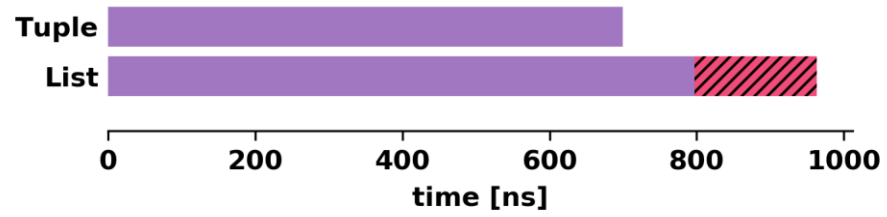
2. Rewrite to directly instantiate a Flink **tuple** instead of an R **list**

IMPACT OF DIRECT TYPE ACCESS

- From `list(...)` to `flink.tuple(...)`
- Avoids additional pass over R list to create Flink tuple
- Up to 1.75*x* performance improvement



Output w/ arity 2



Output w/ arity 19

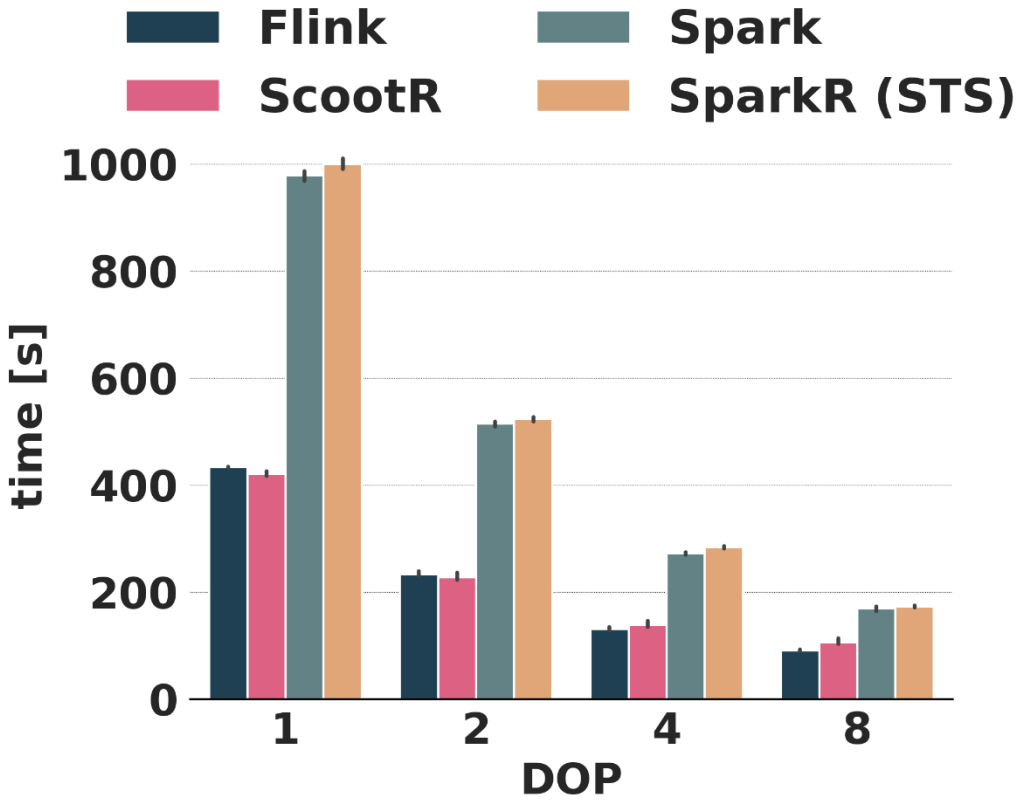
Purple is function execution, **pink** (hatched) conversion from list to tuple

Evaluation

APPLY FUNCTION MICROBENCHMARK

- Airline On-Time Performance Dataset (2005 – 2016)
  CSV, 19 columns, **9.5GB**

- UDF: `df$km <- df$miles * 1.6`

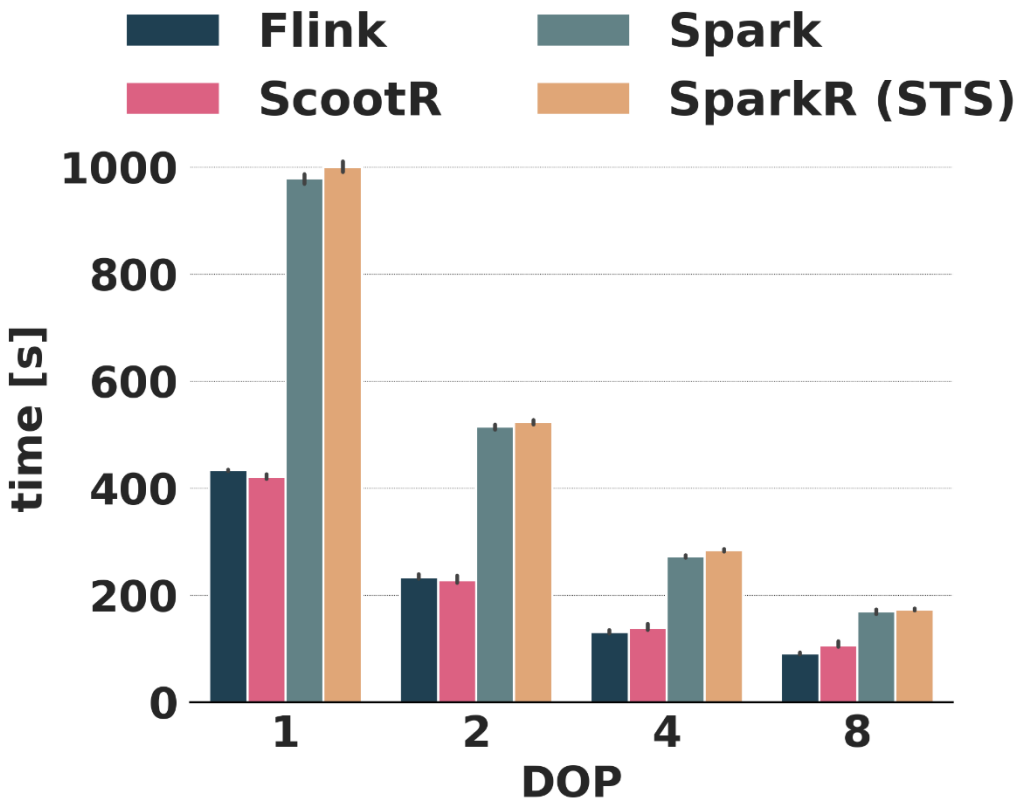


APPLY FUNCTION MICROBENCHMARK

- Airline On-Time Performance Dataset (2005 – 2016)
  CSV, 19 columns, **9.5GB**

- UDF: `df$km <- df$miles * 1.6`

ScootR and **SparkR (STS)** achieve near native performance



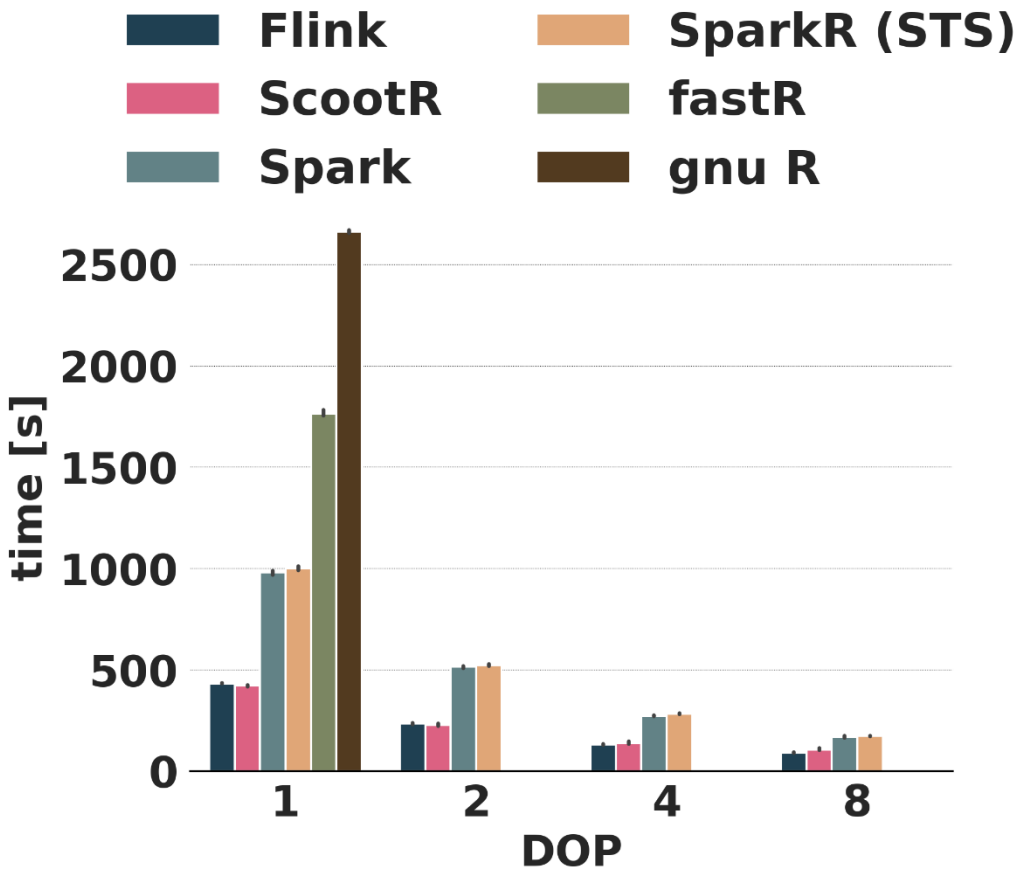
APPLY FUNCTION MICROBENCHMARK

- Airline On-Time Performance Dataset (2005 – 2016)
  CSV, 19 columns, **9.5GB**

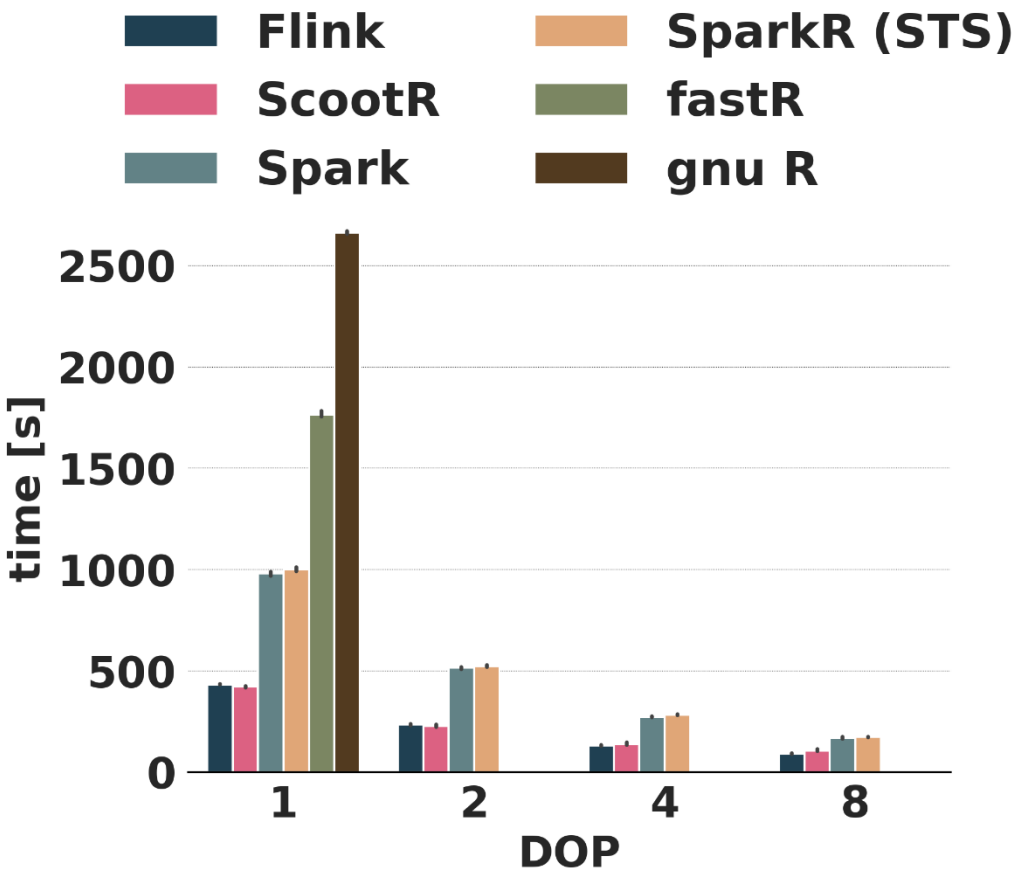
- UDF: `df$km <- df$miles * 1.6`

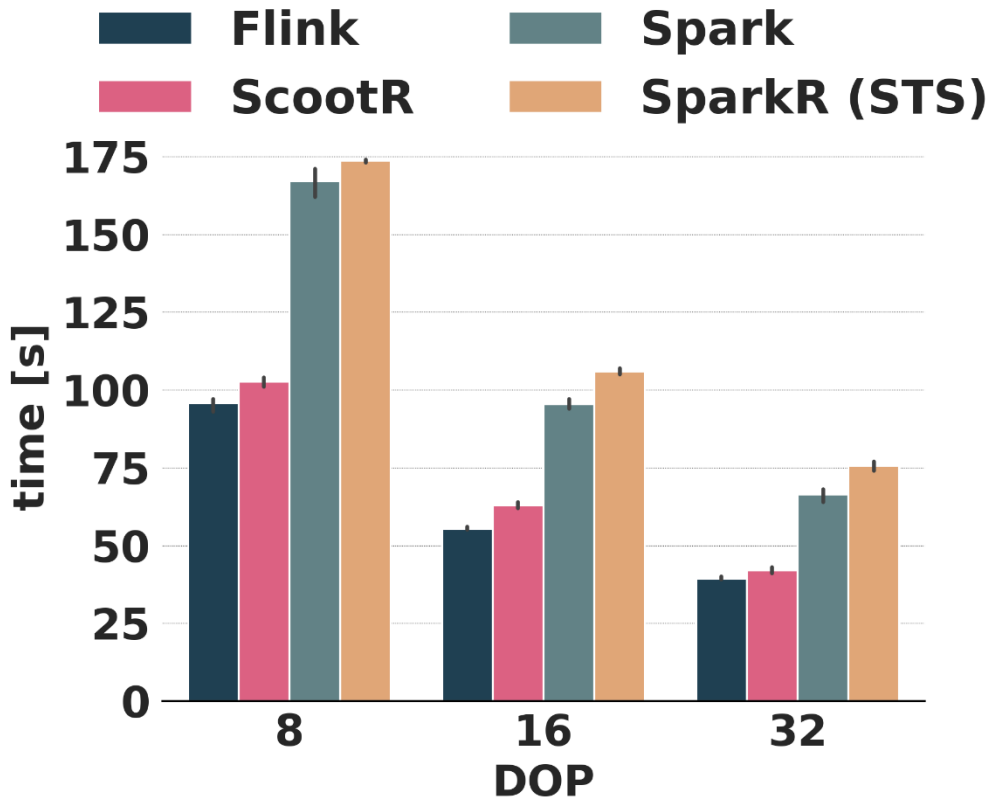
ScootR and **SparkR (STS)** achieve near native performance

Both heavily outperform **gnu R** and **fastR**



APPLY FUNCTION MICROBENCHMARK: SCALABILITY





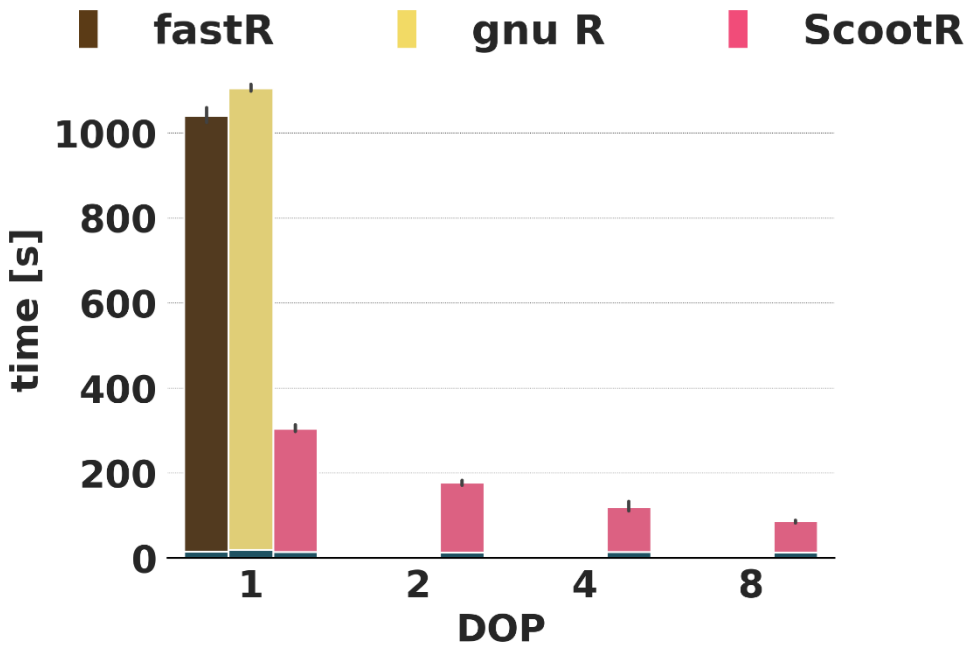
MIXED PIPELINE W/ PREPROCESSING AND ML

Pipeline:

- ● ● ● (Distributed) preprocessing of the dataset
- ● Data is collected locally and an generalized linear model is trained

Majority of the time is spent in preprocessing

ScootR is up to 11x faster than **gnu R** and **fastR**



RECAP

- ScootR provides **a data.frame API** in R **for Apache Flink**

- R and Flink run within the same runtime

  - Avoids serialization and data exchange
  - Avoids type conversion

> Achieves near native performance for a rich set of operators