Apache Arrow
and the Future of Data Frames

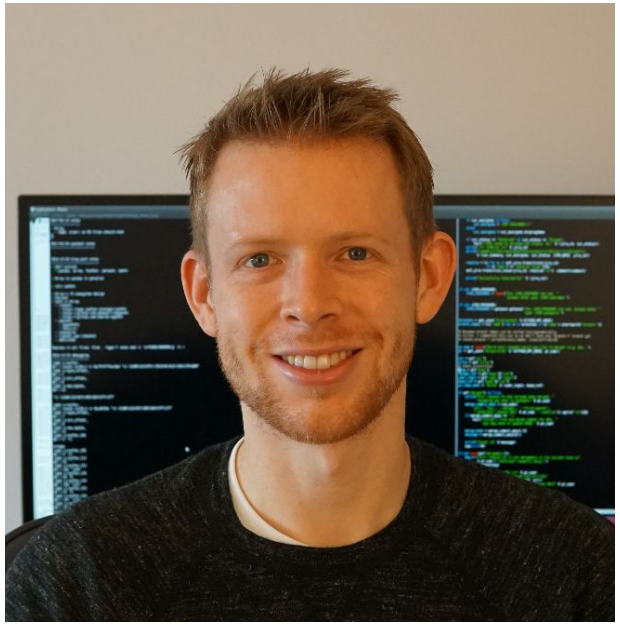
July 8, 2020 -- ACM TechTalks

Wes McKinney

URSA LABS

RStudio

Wes McKinney



- Director of **Ursa Labs**, not-for-profit dev group working on **Apache Arrow**
- Created Python **pandas** project (~2008), lead developer/maintainer until 2013
- PMC Apache Arrow, Apache Parquet, ASF Member
- Wrote ***Python for Data Analysis*** (1e 2012, 2e 2017)
- Formerly: Two Sigma, Cloudera, DataPad, AQR

Career Theme

Programming interfaces for data preparation, analytics, and feature engineering

Some Partners

- https://ursalabs.org
- Apache Arrow-powered Data Science Tools
- Funded by corporate partners
- Built in collaboration with RStudio

What exactly ***is*** a **data frame**?

Is it...

- A **data structure**?
- Or an **API**?

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
Mazda RX4	21.0	6	160.0	110	3.90	2.620	16.46	0	1	4	4
Mazda RX4 Wag	21.0	6	160.0	110	3.90	2.875	17.02	0	1	4	4
Datsun 710	22.8	4	108.0	93	3.85	2.320	18.61	1	1	4	1
Hornet 4 Drive	21.4	6	258.0	110	3.08	3.215	19.44	1	0	3	1
Hornet Sportabout	18.7	8	360.0	175	3.15	3.440	17.02	0	0	3	2
Valiant	18.1	6	225.0	105	2.76	3.460	20.22	1	0	3	1
Duster 360	14.3	8	360.0	245	3.21	3.570	15.84	0	0	3	4
Merc 240D	24.4	4	146.7	62	3.69	3.190	20.00	1	0	4	2
Merc 230	22.8	4	140.8	95	3.92	3.150	22.90	1	0	4	2
Merc 280	19.2	6	167.6	123	3.92	3.440	18.30	1	0	4	4
Merc 280C	17.8	6	167.6	123	3.92	3.440	18.90	1	0	4	4
Merc 450SE	16.4	8	275.8	180	3.07	4.070	17.40	0	0	3	3
Merc 450SL	17.3	8	275.8	180	3.07	3.730	17.60	0	0	3	3
Merc 450SLC	15.2	8	275.8	180	3.07	3.780	18.00	0	0	3	3
Cadillac Fleetwood	10.4	8	472.0	205	2.93	5.250	17.98	0	0	3	4
Lincoln Continental	10.4	8	460.0	215	3.00	5.424	17.82	0	0	3	4

One definition...

A "**data frame**" is

… a **programming interface**

… for expressing **data manipulations**

… on **tabular datasets**

… in a **general programming language**

… and whose primary modality is **analytical**

Compared with SQL-based systems

Data frames *often*

… use imperative / procedural constructs

… offer access to internal structure

… expose operations outside of traditional relational algebra

… have stateful semantics

Data structures: different approaches

- In R, the “data frame” data structure is part of the language
- Other projects implement their own (e.g. pandas)
- Some projects may not use any data structures (e.g. compiling operations to SQL)

Computational engines

- Algorithms implemented against project-specific data structures
- Coupling between API and algorithms common
- New data frame libraries must often develop new implementations

Data Access

- Data must be “loaded” (deserialized) into data structures in order to be manipulated / analyzed
- Each data source requires a data structure-specific “loader” implementation
- Examples: network protocols, file formats, etc.

Many-layered inefficiencies

- Most data frames are effectively “islands” with a hard serialization barrier
- Many non-reusable implementations of the same algorithms
- Limited collaboration across projects and programming languages

The Great CSV Showdown: Julia vs Python vs R

The very first task in any data analysis workflow is simply reading the data, and this absolutely must be done quickly and efficiently so the more interesting work can begin. Across many industries and domains, the CSV file format is king for storing and sharing tabular data. Loading CSVs fast and robustly is crucial, and it must scale well across a wide variety of file sizes, data types, and shapes. This post compares the performance for reading 8 different real-world datasets across three different CSV parsers: R's fread, Pandas' read_csv, and Julia's CSV.jl. Each of these was chosen as the "best in class" CSV parser in each R, Python and Julia, respectively. Multithreading is essential to reach peak performance with today's computers, but only one language (Julia) was able to consistently and efficiently use multiple cores.

https://towardsdatascience.com/the-great-csv-showdown-julia-vs-python-vs-r-aa77376fb96

What do we want?

- Reduce or remove serialization overhead associated with data access
- Reuse algorithms and IO utilities across data frame projects
- Promote collaboration and unify developer efforts

APACHE

ARROW

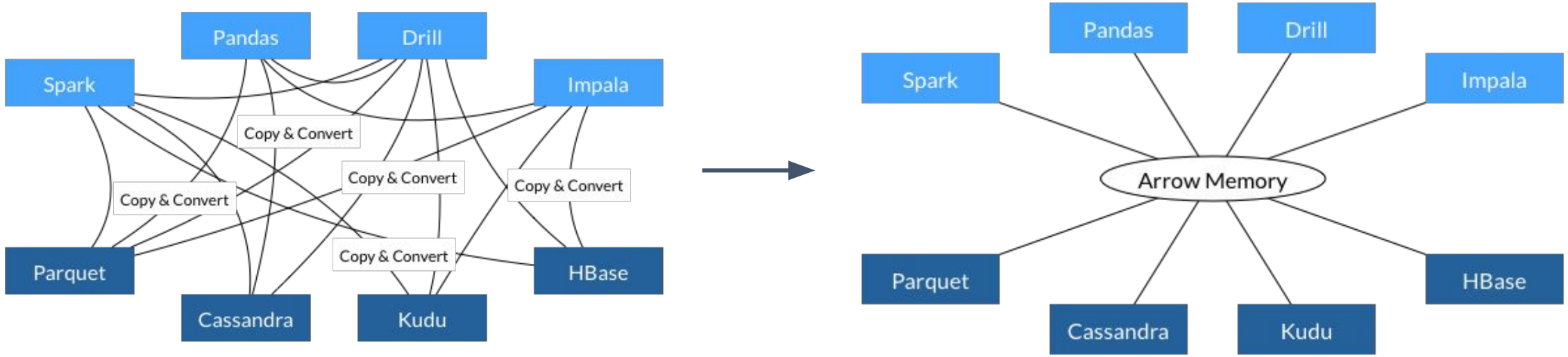
Apache Arrow

- Open source community project launched in 2016
- Intersection of database systems, big data, and data science tools
- Purpose: Language-independent open standards and libraries to accelerate and simplify in-memory computing
- **https://github.com/apache/arrow**

Personal motivations

- Improve interoperability problems with other data processing systems
- Standardize data structures used in data frame implementations
- Promote collaboration and code reuse across libraries and programming languages

Defragmenting Data



Not just any data structures

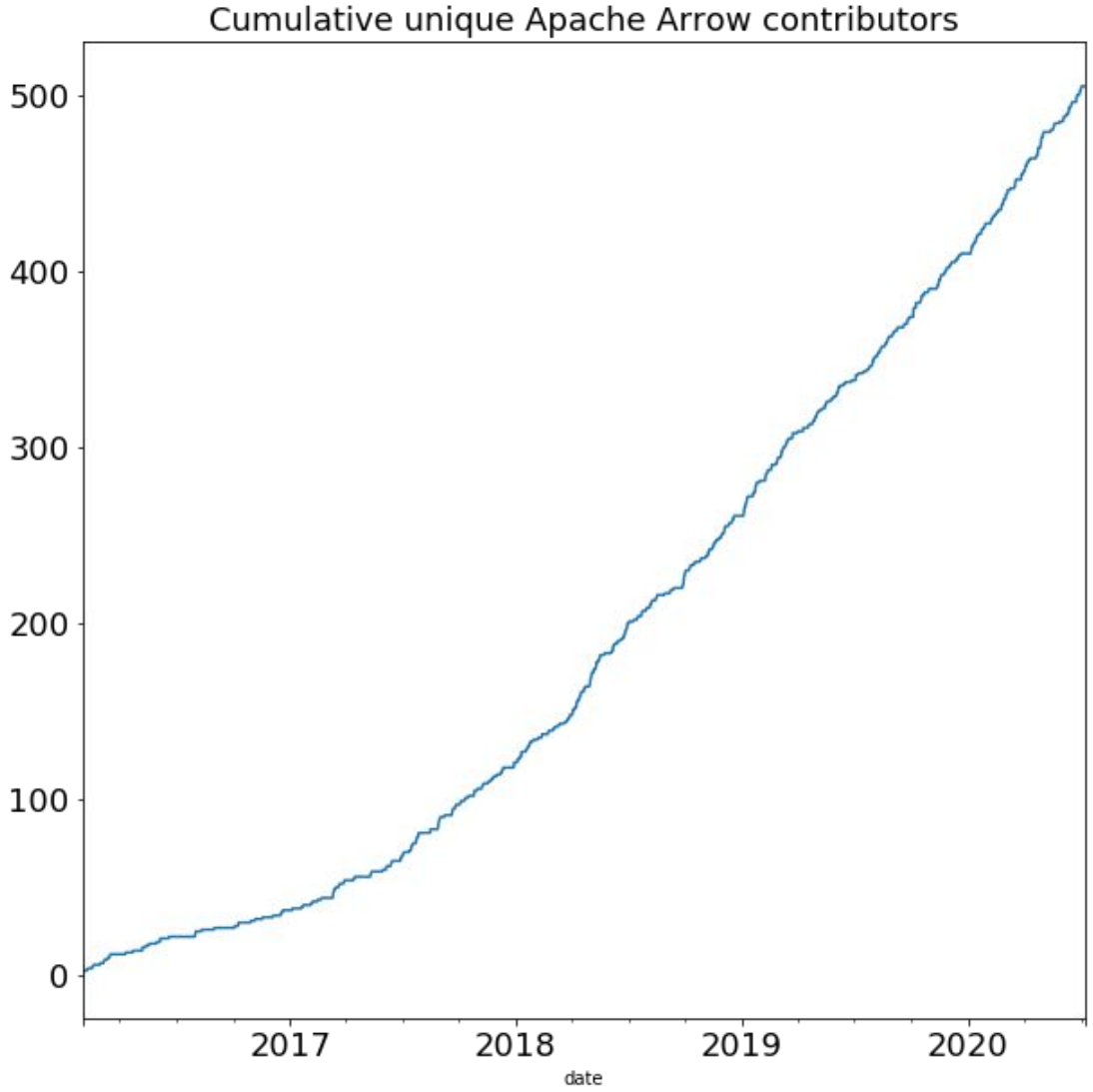
- Address fundamental computational issues in “legacy” data structures:
  - Limited data types
  - Excessive memory consumption
  - Poor processing efficiency for non-numeric types
  - Accommodate larger-than-memory datasets

Apache Arrow Project Overview

- Language-agnostic **in-memory columnar format** for analytical query engines, data frames
- **Binary protocol** for IPC / RPC
- **“Batteries included” development platform** for building data processing applications

2020 Development Status

- 17 major releases
- Over 500 unique contributors
- Over 50M package installs in 2019
- ASF roster: 52 committers, 29 PMC members
- 11 programming languages represented



Upcoming Arrow 1.0.0 release

- Columnar format and protocol formally declared stable with backward/forward compatibility guarantees
- Libraries moving to Semantic Versioning

Arrow and the Future of Data Frames

- As more data sources offer Arrow-based data access, it will make sense to process Arrow *in situ* rather than converting to some other data structure
- Analytical systems will generally grow more efficient the more “Arrow-native” they become

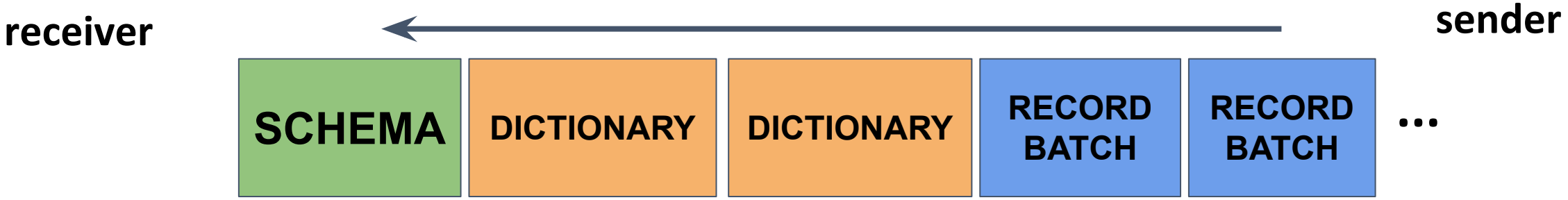
Arrow Columnar Format and Binary Protocol

Arrow’s Columnar Memory Format

- Runtime memory format for analytical query processing
  - Ideal companion to columnar storage like Apache Parquet
- “Fully shredded” columnar, supports flat and nested schemas
- Organized for cache-efficient access on CPUs/GPUs
- Optimized for data locality, SIMD, parallel processing
- Accommodates both random access and scan workloads

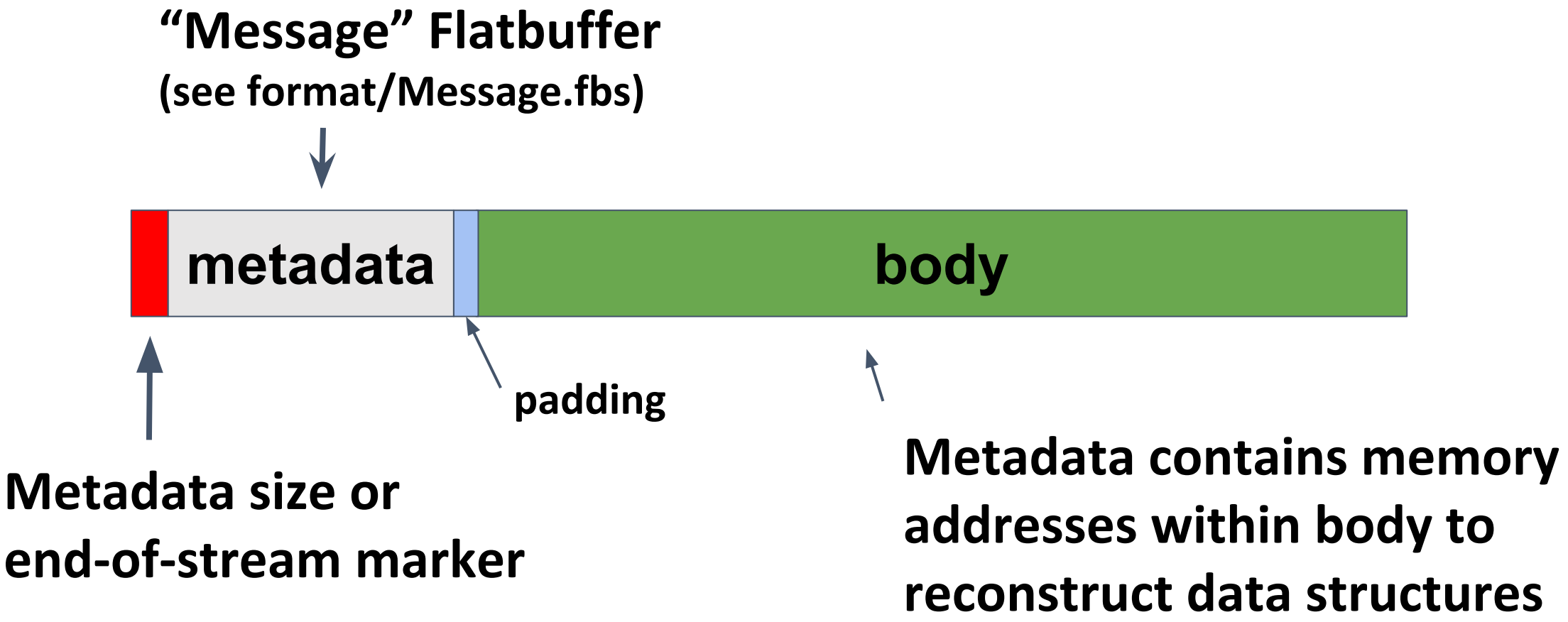
Arrow Binary Protocol

- **Record batch:** ordered collection of named arrays
- Streaming wire format for transferring datasets between address spaces
- Intended for both IPC / shared memory and RPC use cases



Encapsulated protocol (“IPC”) messages

- Serialization wire format suitable for stream-based parsing



Columnar Format Future Directions

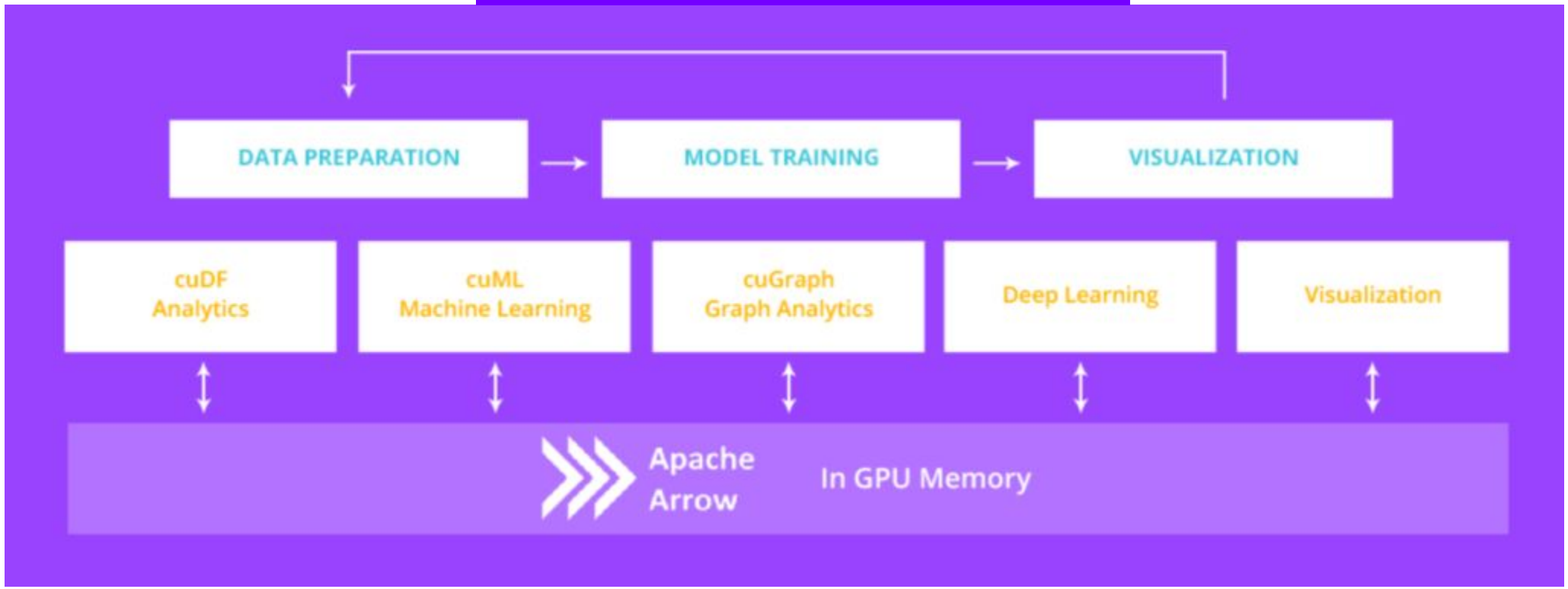
- In-memory encoding, compression, sparseness
  - e.g. run-length encoding
  - See mailing list discussions, we need your feedback!
- Expansion of logical types

Some success stories

NVIDIA

RAPIDS

Open GPU Data Science



Introducing Pandas UDF for PySpark

How to run your native Python code with PySpark, fast.

by Li Jin

Posted in ENGINEERING BLOG | October 30, 2017

This is a guest community post from Li Jin, a software engineer at Two Sigma Investments, LP in New York. This blog is also posted on Two Sigma

Try this notebook in Databricks

UPDATE: This blog was updated on Feb 22, 2018, to include some changes.

This blog post introduces the Pandas UDFs (a.k.a. Vectorized UDFs) feature in the upcoming Apache Spark 2.3 release that substantially improves the performance and usability of user-defined functions (UDFs) in Python.

https://databricks.com/blog/2017/10/30/introducing-vectorized-udfs-for-pyspark.html

Announcing google-cloud-bigquery Version 1.17.0: Query Results to DataFrame 31x Faster with Apache Arrow

Tim Swast Follow

Jul 28, 2019 · 4 min read

https://medium.com/google-cloud/announcing-google-cloud-bigquery-version-1-17-0-1fc428512171

snowflake

PRODUCT SOLUTIONS PRICING CUSTOMERS PAI

FETCHING QUERY RESULTS FROM SNOWFLAKE JUST GOT A LOT FASTER WITH APACHE ARROW

FEB 12, 2020 | 4 MIN READ

Author: Harsha Kapre | Contributing Authors: Andong Zhan and Haowei Yu

Snowflake News

We took our first step toward the adoption of Apache Arrow with the release of our latest JDBC and Python clients. Fetching result sets over these clients now leverages the Arrow columnar format to avoid the overhead previously associated with serializing and deserializing Snowflake data structures which are also in columnar format.

https://www.snowflake.com/blog/fetching-query-results-from-snowflake-just-got-a-lot-faster-with-apache-arrow/

influxdata Products Solutions Developers Pricing

Apache Arrow, Parquet, Flight and Their Ecosystem are a Game Changer for OLAP

By Paul Dix / April 16, 2020 / Community, Developer / 1 Comment

Estimated read time: 11 minutes

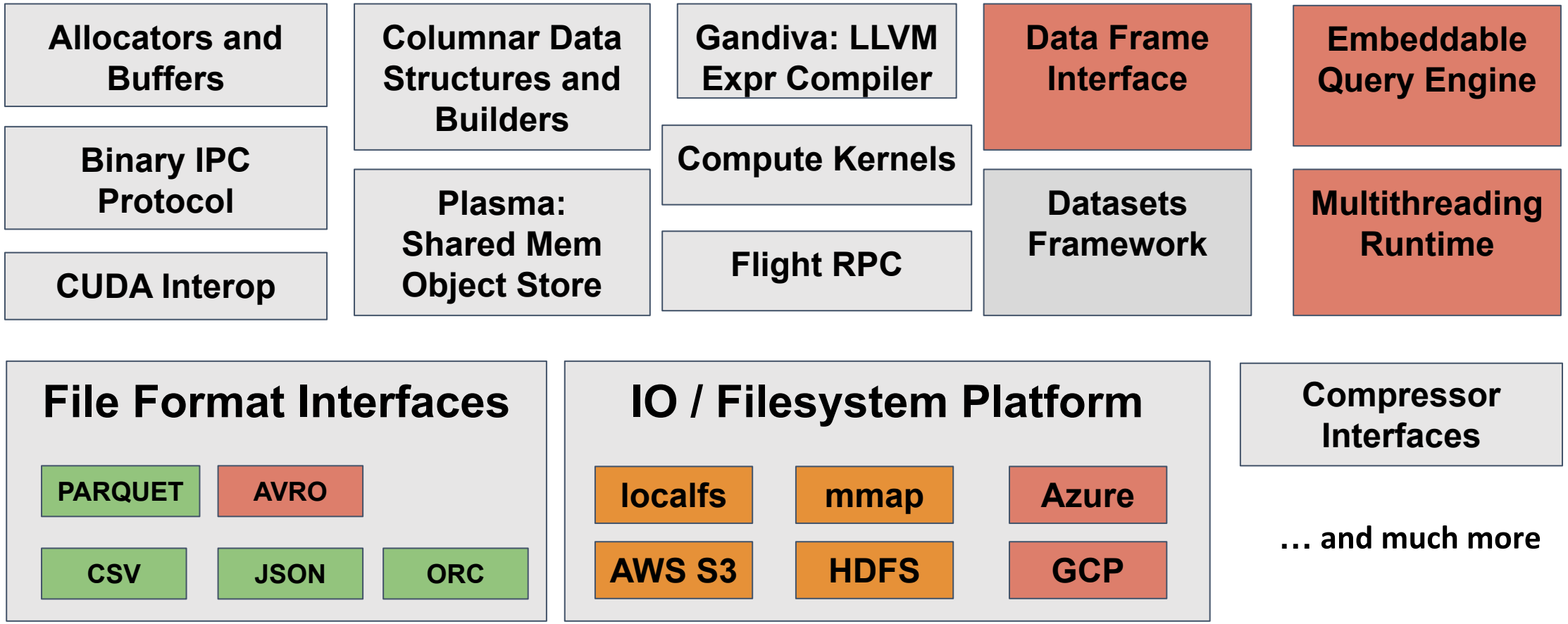
Apache Arrow, a specification for an in-memory columnar data format, and associated projects: Parquet for compressed on-disk data, Flight for highly efficient RPC, and other projects for in-memory query processing will likely shape the future of OLAP and data warehousing systems. This will mostly be driven by the promise of interoperability between projects, paired with massive performance gains for pushing and pulling data in and out of big data systems. With object storage like S3 as the common data lake, OLAP projects need a common data API, which Parquet represents. For data science and query workloads, they need a common RPC that is optimized for pulling many millions of records to do more complex analytical and machine learning tasks.

In this post, I'll cover each of these areas and why I think the Apache Arrow umbrella of projects represents the common API around which current and future big data, OLAP, and data warehousing projects will collaborate and innovate. I'll conclude with some thoughts on where these projects are and where things might be going.

https://www.influxdata.com/blog/apache-arrow-parquet-flight-and-their-ecosystem-are-a-game-changer-for-olap/

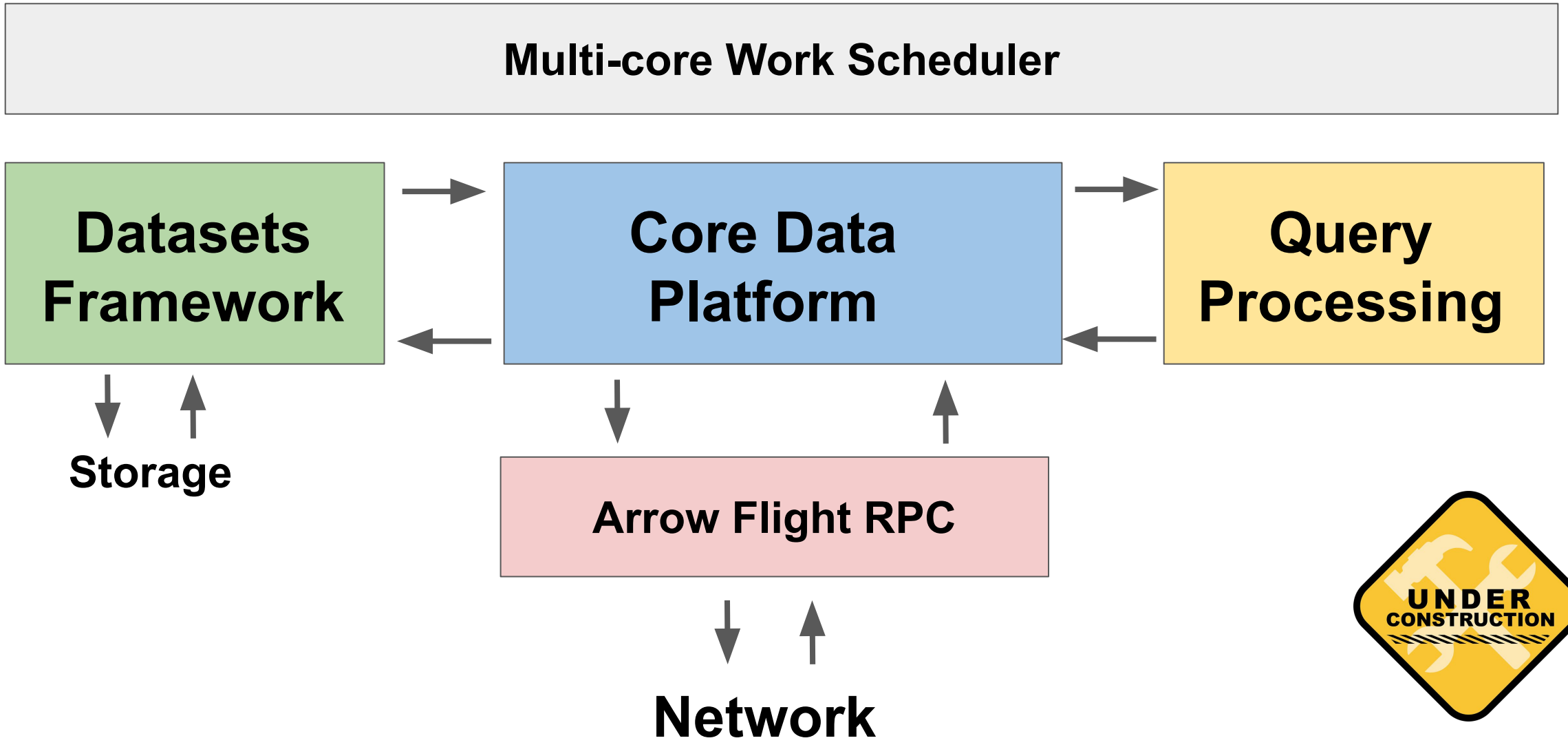
Inside Arrow development

Arrow C++ development platform



Red means planned / under construction work

Arrow C++ Platform



Example: use in R libraries

Can be a massive Arrow dataset

```
flights %>%
  group_by(year, month, day) %>%
  select(arr_delay, dep_delay) %>%
  summarise(
    arr = mean(arr_delay, na.rm = TRUE),
    dep = mean(dep_delay, na.rm = TRUE)
  ) %>%
  filter(arr > 30 | dep > 30)
```

dplyr verbs can be translated to Arrow computation graphs, executed by parallel runtime

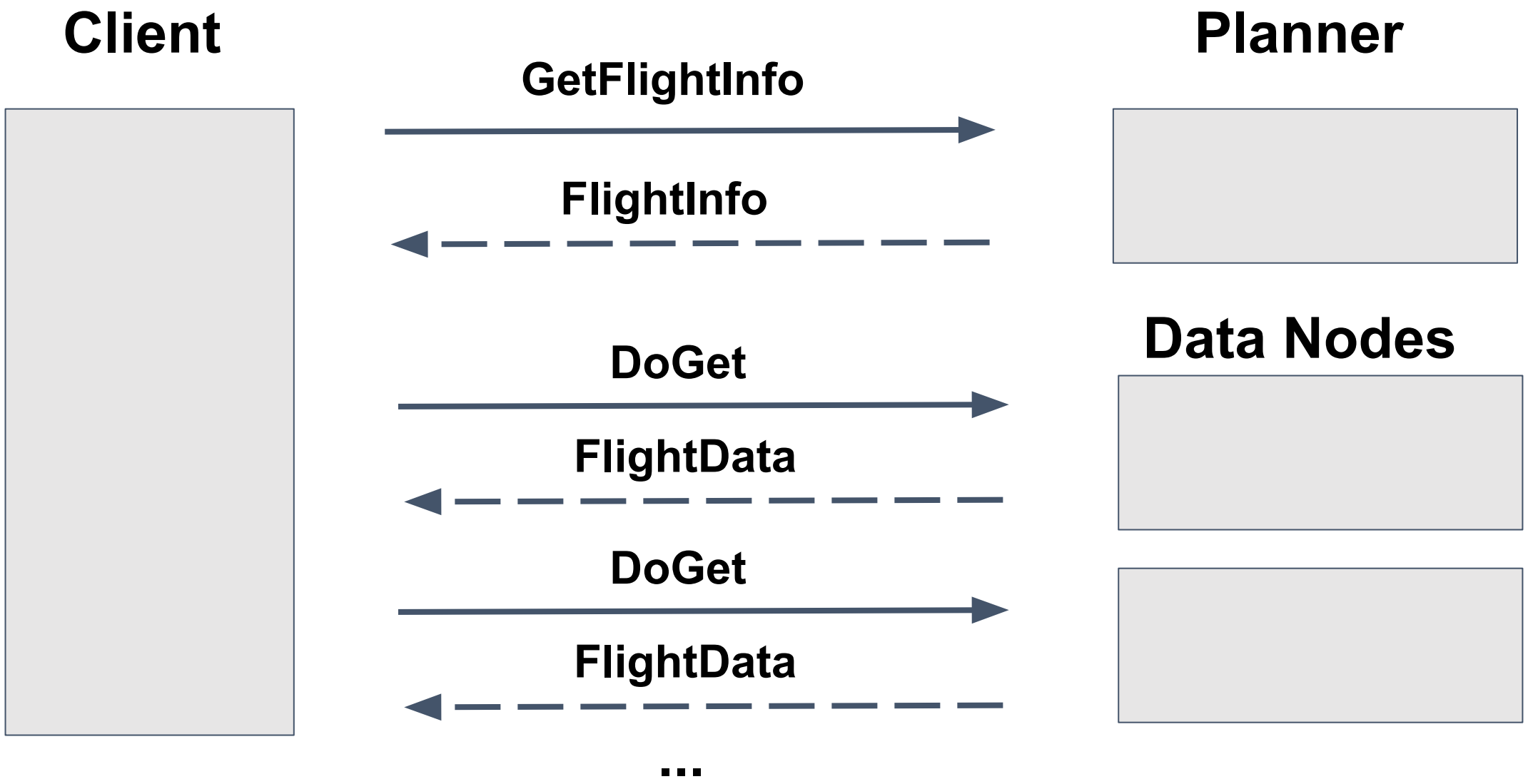
R expressions can be JIT-compiled with LLVM

Arrow Flight RPC Framework

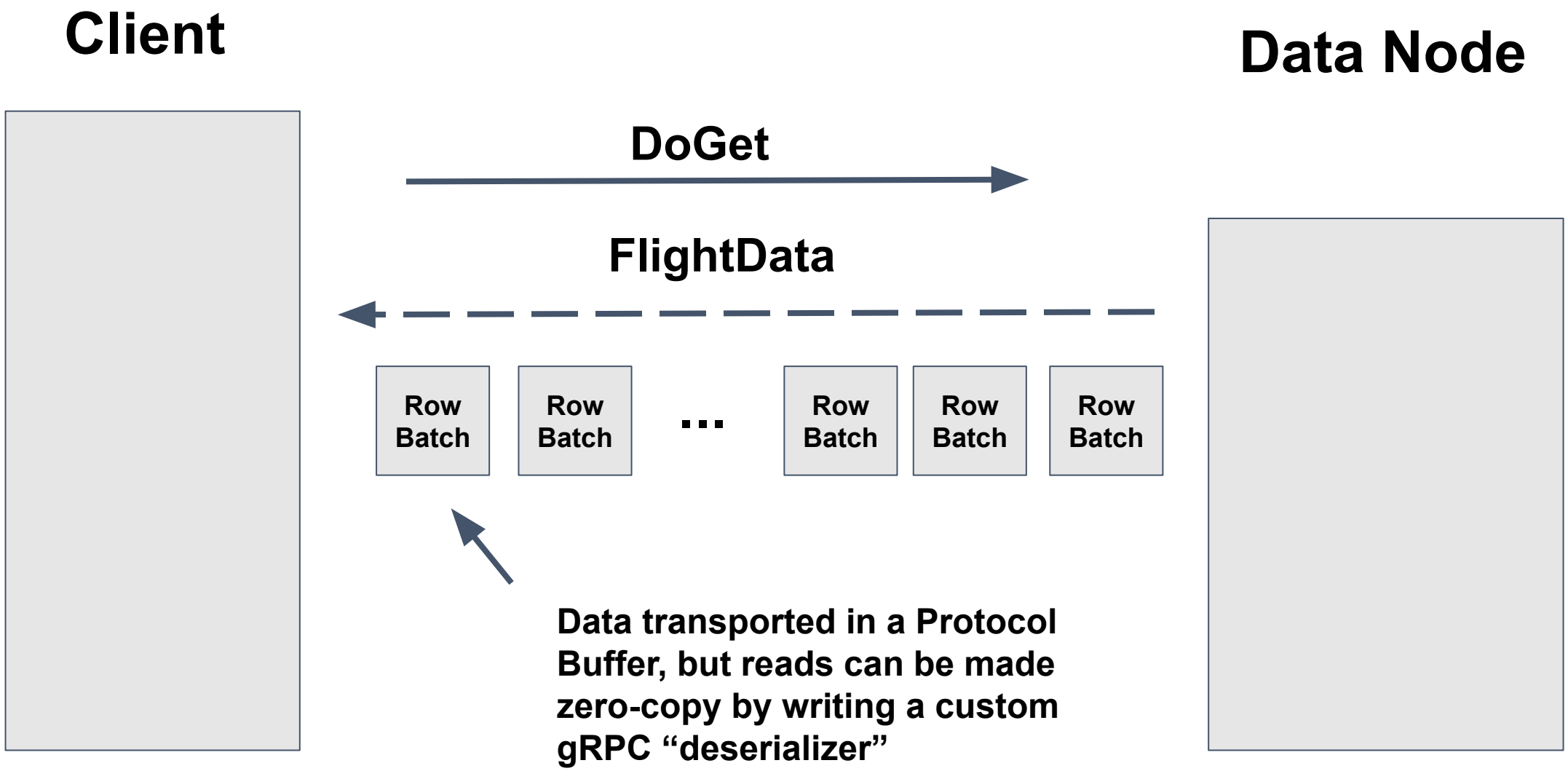
Arrow Flight Overview

- A gRPC-based framework for defining custom data services that send and receive Arrow columnar data natively
- Uses Protocol Buffers v3 for client protocol
- Pluggable command execution layer, authentication
- Low-level gRPC optimizations to avoid unnecessary serialization

Arrow Flight - Parallel Get



Arrow Flight - Efficient gRPC transport



Demo: Build simple Flight service in Python

Getting involved

- Join dev@arrow.apache.org
- Development https://github.com/apache/arrow