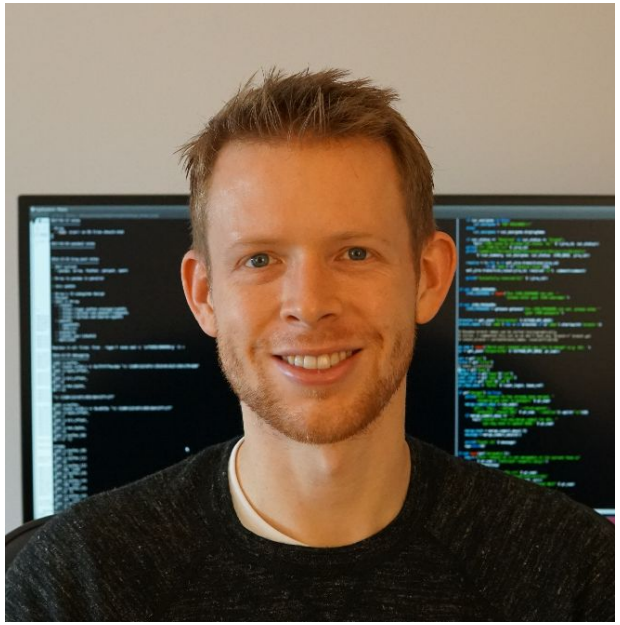
Apache Arrow

Cross-Language Development Platform for In-Memory Analytics

Wes McKinney

DataEngConf Barcelona 2018

Wes McKinney



- 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 Co-founder / CEO of DataPad (acquired by Cloudera in 2014)
- Other OSS work: Ibis, Feather, Apache Kudu, statsmodels

https://ursalabs.org

- Raise money to support full-time open source developers
- Grow **Apache Arrow** ecosystem
- Build cross-language, portable computational libraries for data science
- Build relationships across industry

People

Leadership

Wes McKinney

Director

Wes created the pandas project in 2008 and wrote the book *Python for Data Analysis*, helping popularize the use of Python for data science. He is a Member of The Apache Software Foundation, and is a PMC member for Apache Arrow and Apache Parquet. He was formerly the CEO and co-founder of DataPad. He is the managing director of Ursa Labs.

Hadley Wickham

Advisor

Hadley is the creator of many of the most widely-used R packages for data science, such as ggplot2, dplyr, and many others. He has written several books about R, such as R for Data Science and Advanced R. Hadley is the Chief Scientist at RStudio. He is a technical advisor for Ursa Labs on R language support and general API design and usability.

Initial Sponsors and Partners

Prospective sponsors / partners,
please reach out: info@ursalabs.org

Open standards: why do they matter?

- Simplify system architectures
- Reduce ecosystem fragmentation
- Improve interoperability
- Reuse more libraries and algorithms

Example open standards

- Human-readable semi-structured data: XML, JSON
- Structured data query language: SQL
- Binary storage formats (with metadata)
  - NetCDF
  - HDF5
  - Apache Parquet, ORC
- Serialization / RPC protocols
  - Apache Avro
  - Protocol buffers
- Not an open standard: Excel, CSV (grrrr)

Standardizing in-memory data

- Best example: strided ndarray / tensor memory (NumPy / Fortran-compatible)
- Why?
  - Zero-overhead memory sharing between libraries in-memory and processes via shared memory
  - Reuse algorithms
  - Reuse IO / storage code

Tables and data frames

- Notoriously not based on open standards
- Vary widely in supported data types (e.g. nested data)
- Where are they found?
  - Internals of SQL databases
  - Big data systems (Apache Spark, Apache Hive)
  - In-memory data frame libraries: Python (pandas), R (base, data.table), Julia (DataFrames.jl)
- We say “data frame” but the byte-level RAM layout varies greatly from system-to-system

Many data frame projects

- These are the workhorse libraries of data engineering, data preparation, and feature engineering for AI/ML
- Little code reuse across projects
- Uneven performance and features

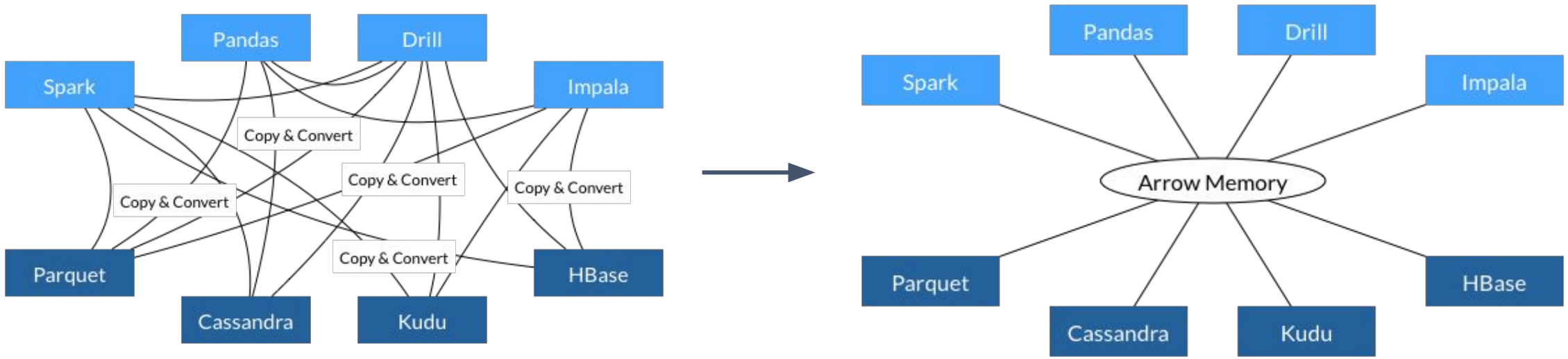
What’s driving the fragmentation?

- Tight coupling between front end and back end
- Feudal nature of open source software communities
- Incompatible memory representations
  - Cannot share **data** without serialization
  - Cannot share **algorithms** because implementation depends on memory representation

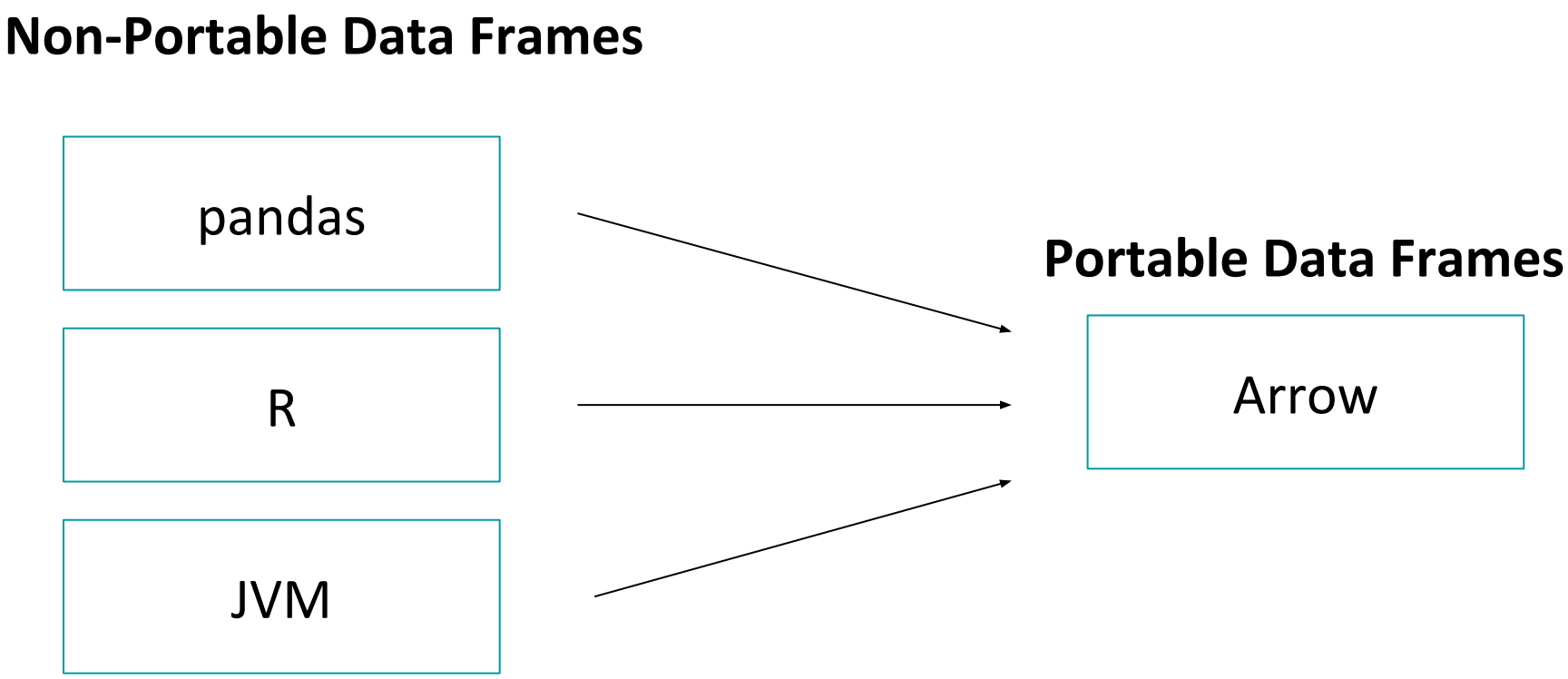
Apache Arrow

- OSS Community initiative conceived in 2015
- Intersection of big data, database systems, and data science tools
- Key idea: Language agnostic open standards to accelerate in-memory computing
- **https://github.com/apache/arrow**

Defragmenting Data Access

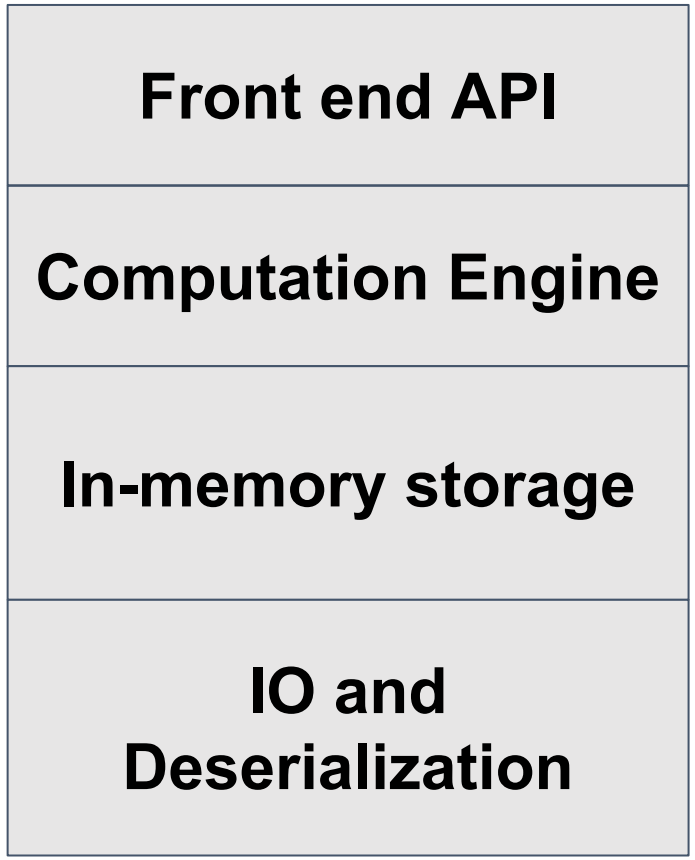


“Portable” Data Frames



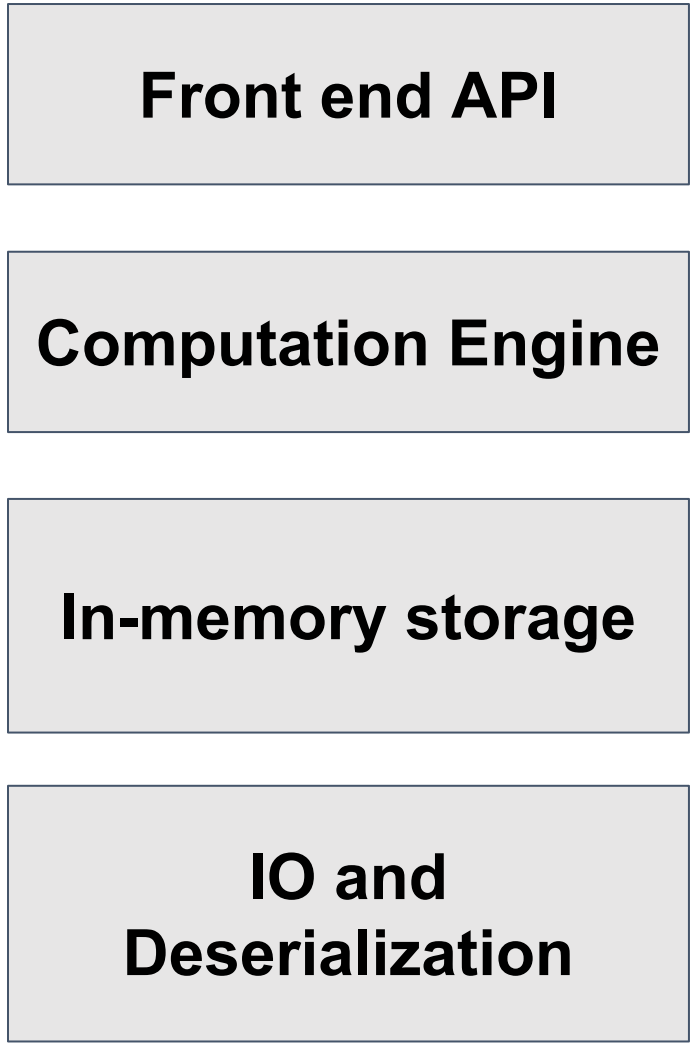
Share **data** and **algorithms** at ~zero cost

Analytic database architecture



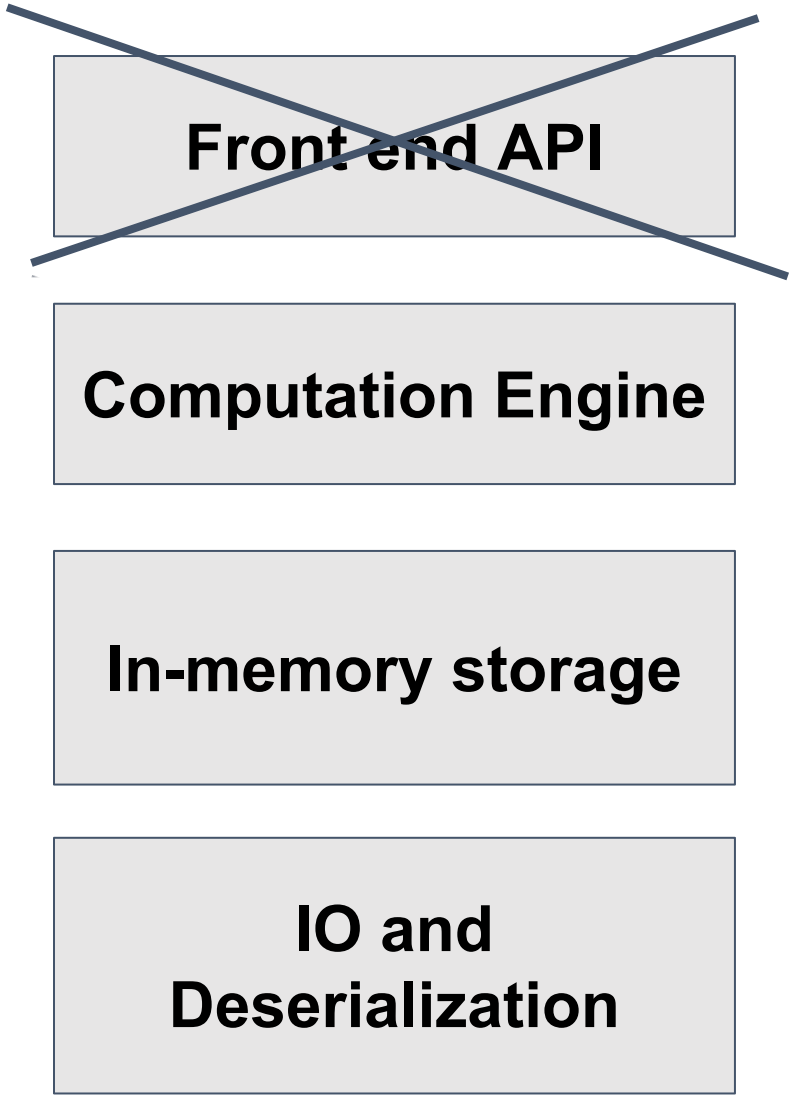
- Vertically integrated / “Black Box”
- Internal components do not have a public API
- Users interact with front end

Analytic database, deconstructed



- Components have public APIs
- Use what you need
- Different front ends can be developed

Analytic database, deconstructed



Arrow is front end agnostic


shutterstock
IMAGE ID: 552124468
www.shutterstock.com

Arrow Use Cases

- Data access
  - Read and write widely used storage formats
  - Interact with database protocols, other data sources
- Data movement
  - Zero-copy interprocess communication
  - Efficient RPC / client-server communications
- Computation libraries
  - Efficient in-memory / out-of-core data frame-type analytics
  - LLVM-compilation for vectorized expression evaluation

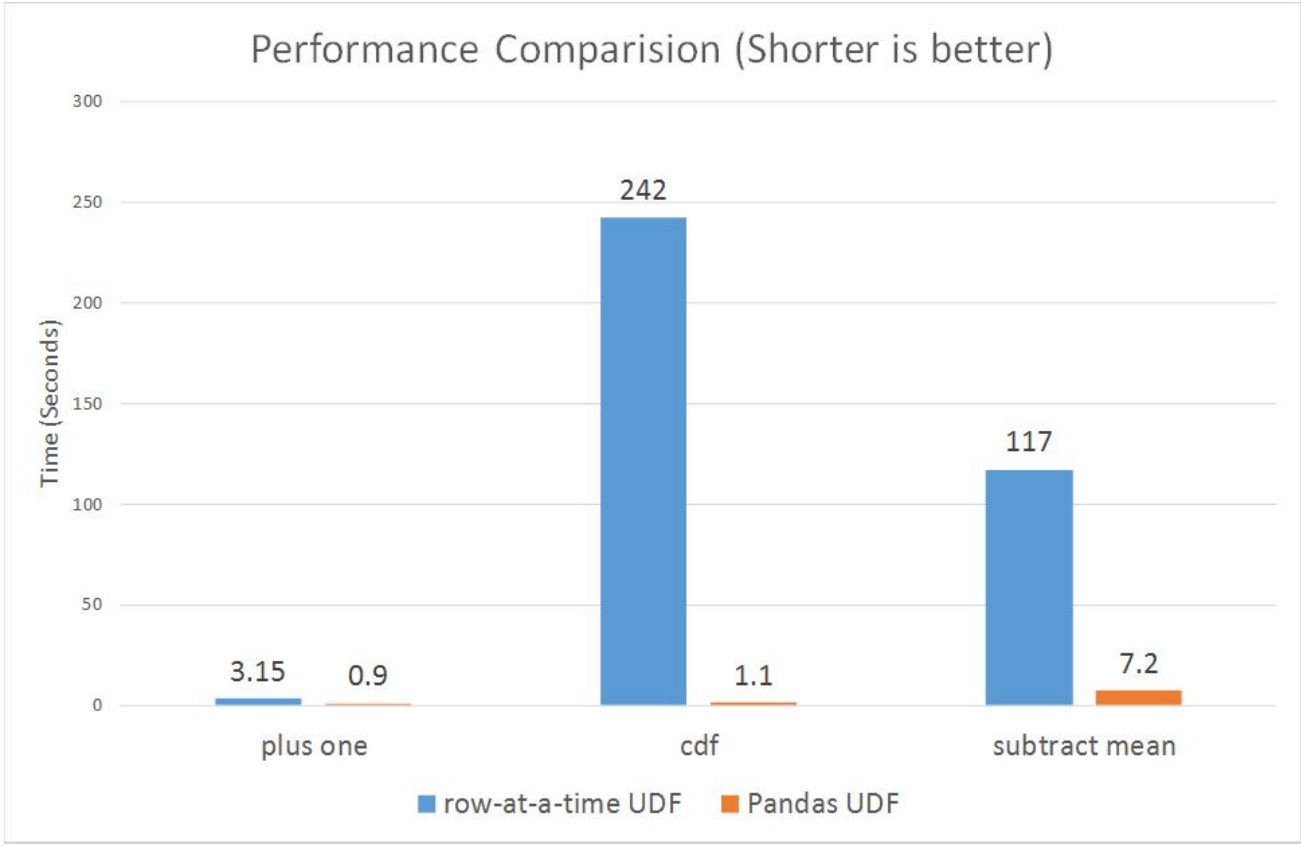
Arrow’s Columnar Memory Format

- Runtime memory format for analytical query processing
  - Companion to serialization tech like Apache {Parquet, ORC}
- “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

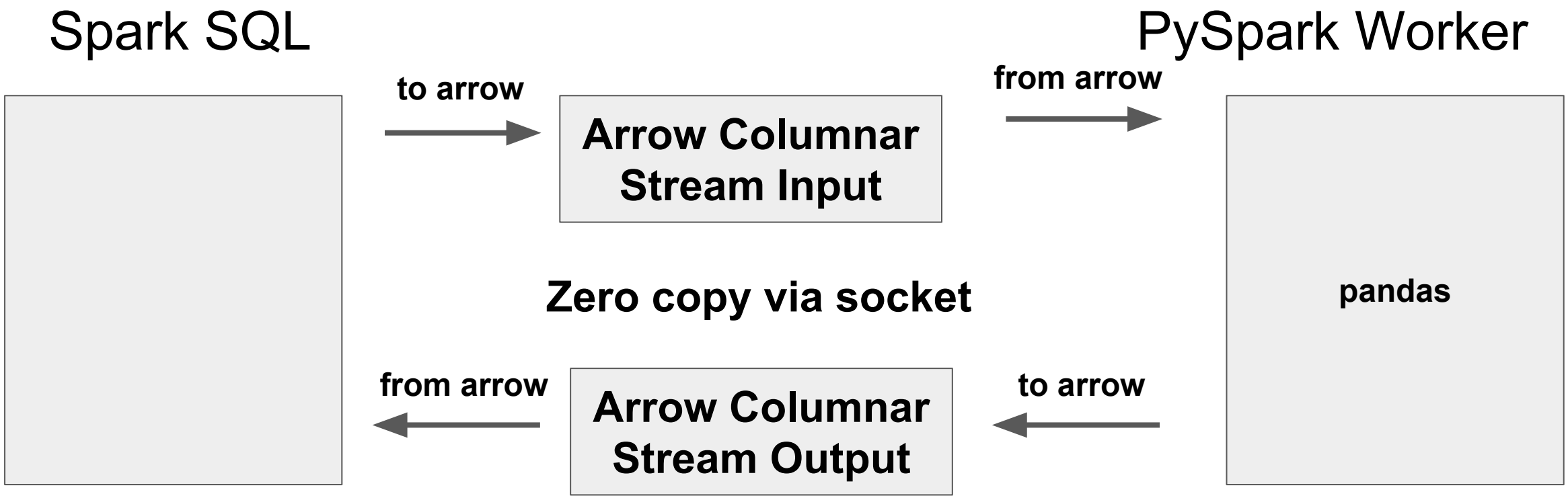
Arrow-accelerated Python + Apache Spark

- Joint work with Li Jin from Two Sigma, Bryan Cutler from IBM
- Vectorized user-defined functions, fast data export to pandas

```
import pandas as pd
from scipy import stats
@pandas_udf('double')
def cdf(v):
    return pd.Series(stats.norm.cdf(v))
df.withColumn('cumulative_probability',
              cdf(df.v))
```



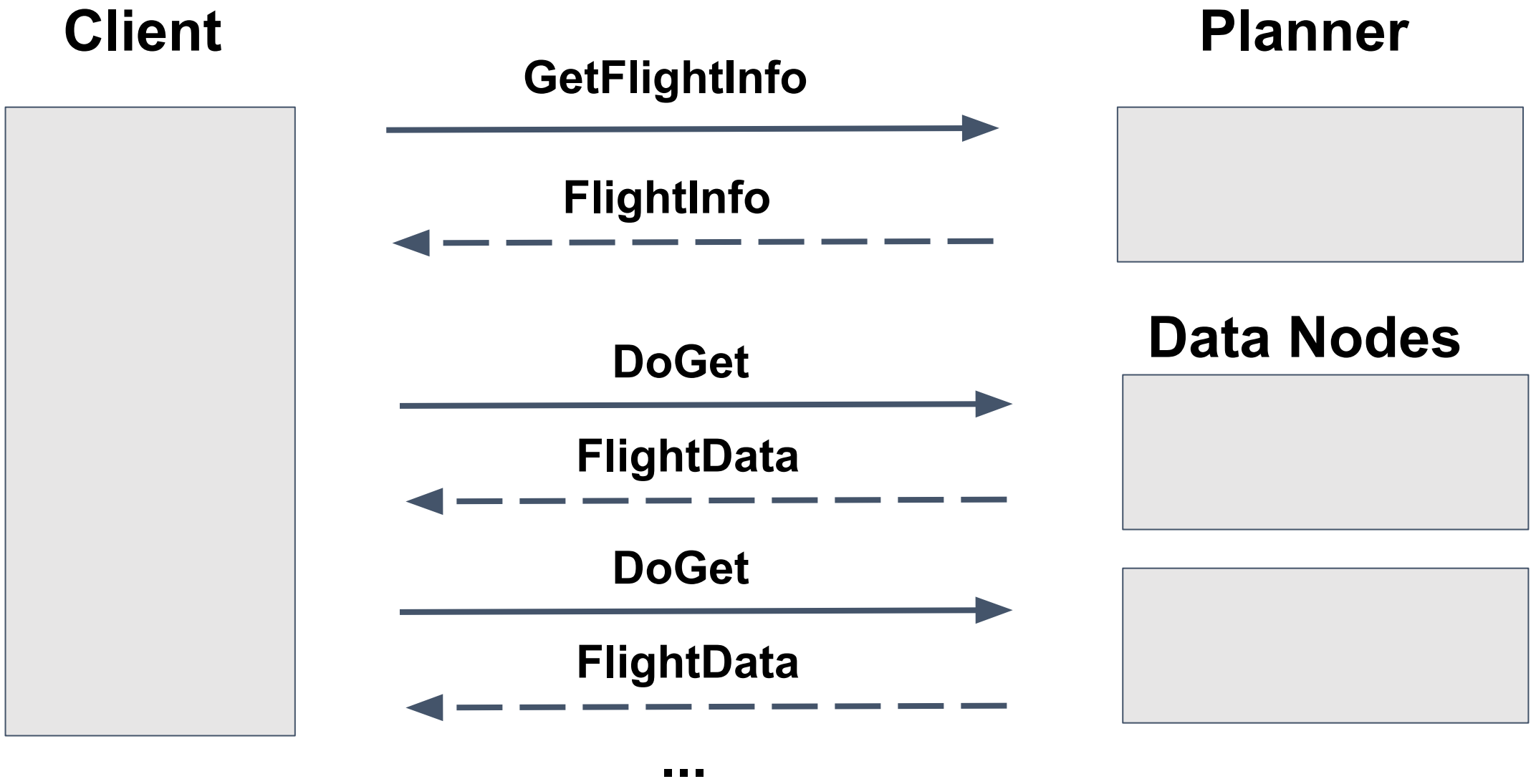
Arrow-accelerated Python + Apache Spark



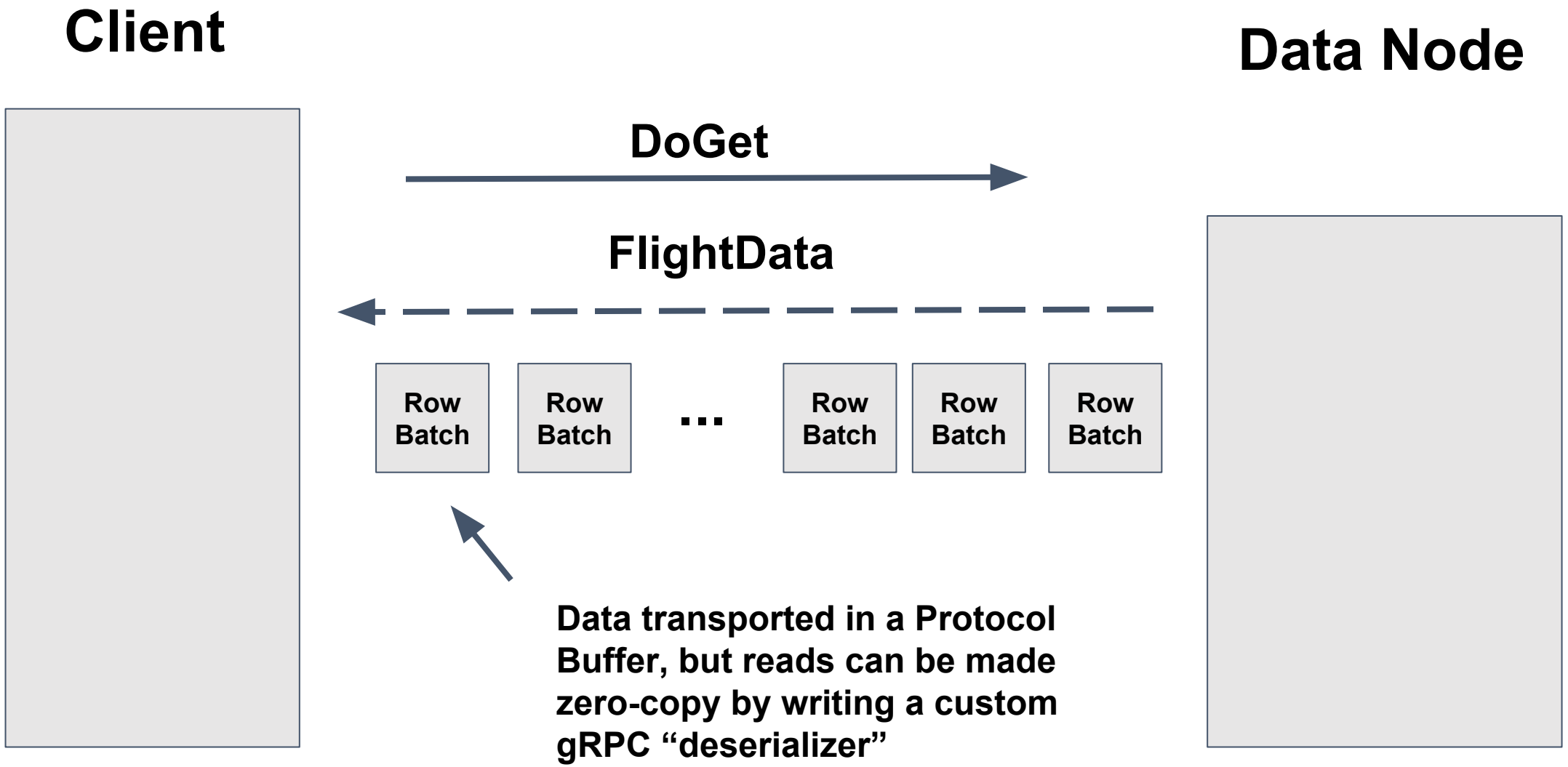
New work: Arrow Flight RPC Framework

- 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
  - Write Arrow memory directly onto outgoing gRPC buffer
  - Avoid any copying or deserialization

Arrow Flight - Parallel Get



Arrow Flight - Efficient gRPC transport



Flight: Static datasets and custom commands

- Support “named” datasets, and “command” datasets
- Commands are binary, and will be server-dependent
- Implement custom commands using general structured data serialization tools

Commands.proto

```
message SQLQuery {
  binary database_uri = 1;
  binary query = 2;
}
```

GetFlightInfo RPC

```
type: CMD
cmd: <serialized command>
```

Flight: Custom actions

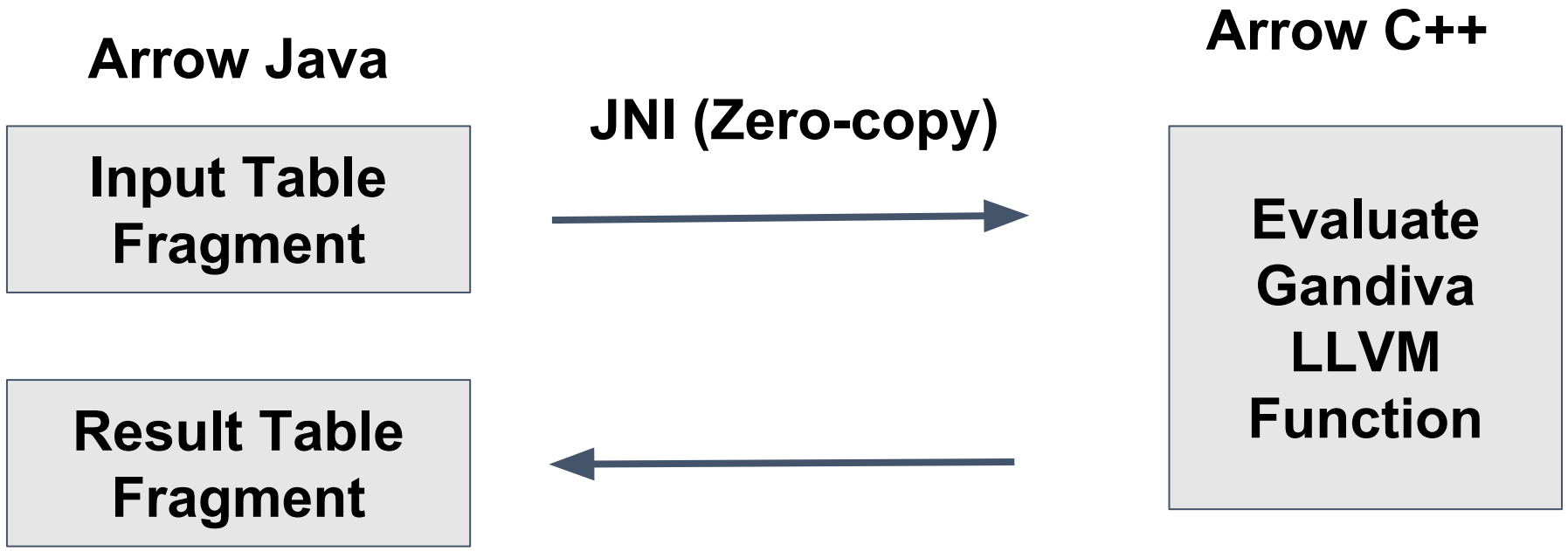
- Any request that is not a table pull or push, can be implemented as an “action”
- Actions return a stream of opaque binary data

Arrow: History and Status

- Community initiative started in 2016, initially backed by leading developers of ~13 major OSS data processing projects
- Project development status
  - Codebase 2.5 years old
  - > 190 distinct contributors
  - **10 major releases**
  - Some level of support in **10 programming languages** (C, C++, Go, Java, JavaScript, MATLAB, Python, R, Ruby, Rust)
  - Over **500K monthly installs** in Python alone

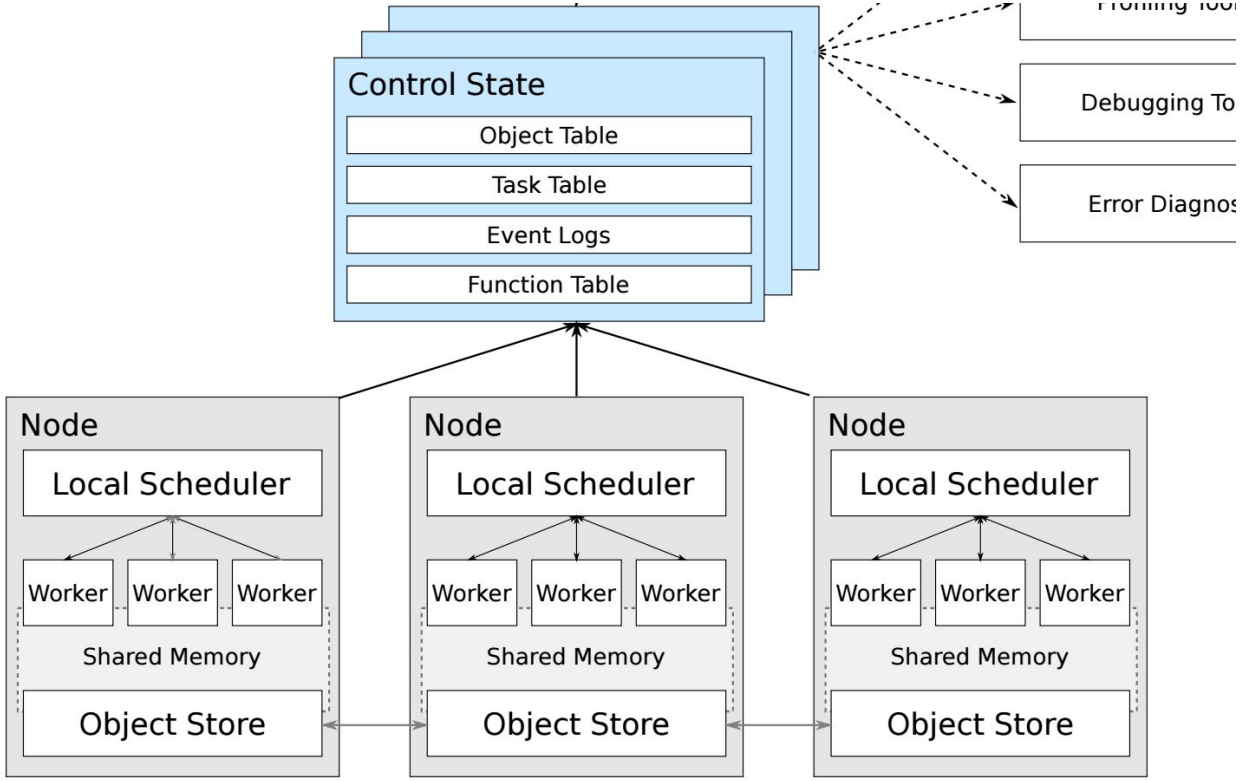
Example: Gandiva, Arrow-LLVM compiler

```
SELECT year(timestamp), month(timestamp), ...
FROM table
...
```



https://github.com/dremio/gandiva

Example use: Ray ML framework from Berkeley RISELab



- **Uses Plasma, shared memory-based object store originally developed for Ray**
- **Zero-copy reads of tensor collections**

Source: https://arxiv.org/abs/1703.03924

Arrow on the GPU

- NVIDIA-led GPU Open Analytics Initiative (http://gpuopenanalytics.com)
- “GPU DataFrame”: Arrow on the GPU
- Example: Execute Numba-compiled code on SQL results from MapD shared via CUDA IPC
- Plasma also supports GPU shared memory

Some Industry Contributors to Apache Arrow

ClearCode

UBER

BlueYonder

cloudera

databricks

Upcoming Roadmap

- Software development lifecycle improvements
- Data ingest / access / export
- Computational libraries (CPU + GPU)
- Expanded language support
- Richer RPC / messaging
- More system integrations

Computational libraries

- “Kernel functions” performing vectorized analytics on Arrow memory format
  - Select CPU or GPU variant based on data location
- Operator graphs (compose multiple operators)
- Subgraph compiler (using LLVM -- see Gandiva)
- Runtime engine: execute operator graphs

Data Access / Ingest

- Apache Avro
- Apache Parquet nested data support
- Apache ORC
- CSV
- JSON
- ODBC / JDBC
- … and likely other data access points

Arrow-powered Data Science Systems

- Portable runtime libraries, usable from multiple programming languages
- Decoupled front ends
- Companion to distributed systems like Dask, Ray

Getting involved

- Join dev@arrow.apache.org
- PRs to https://github.com/apache/arrow
- Learn more about the Ursa Labs vision for Arrow-powered data science: https://ursalabs.org/tech/