Chair of Aerodynamics and Fluid Mechanics
Department of Mechanical Engineering
Technical University of Munich

Probabilistic Programming for Scientific Discovery

Lecture 4

Ludger Paehler
Lviv Data Science Summer School

July 30, 2020

Table of Contents

Interacting with Scientific Simulators: The Engineering Challenge

- Merlin
- Reverb
- Apache Arrow
- XLA

Outline

Interacting with Scientific Simulators: The Engineering Challenge

- Merlin
- Reverb
- Apache Arrow
- XLA

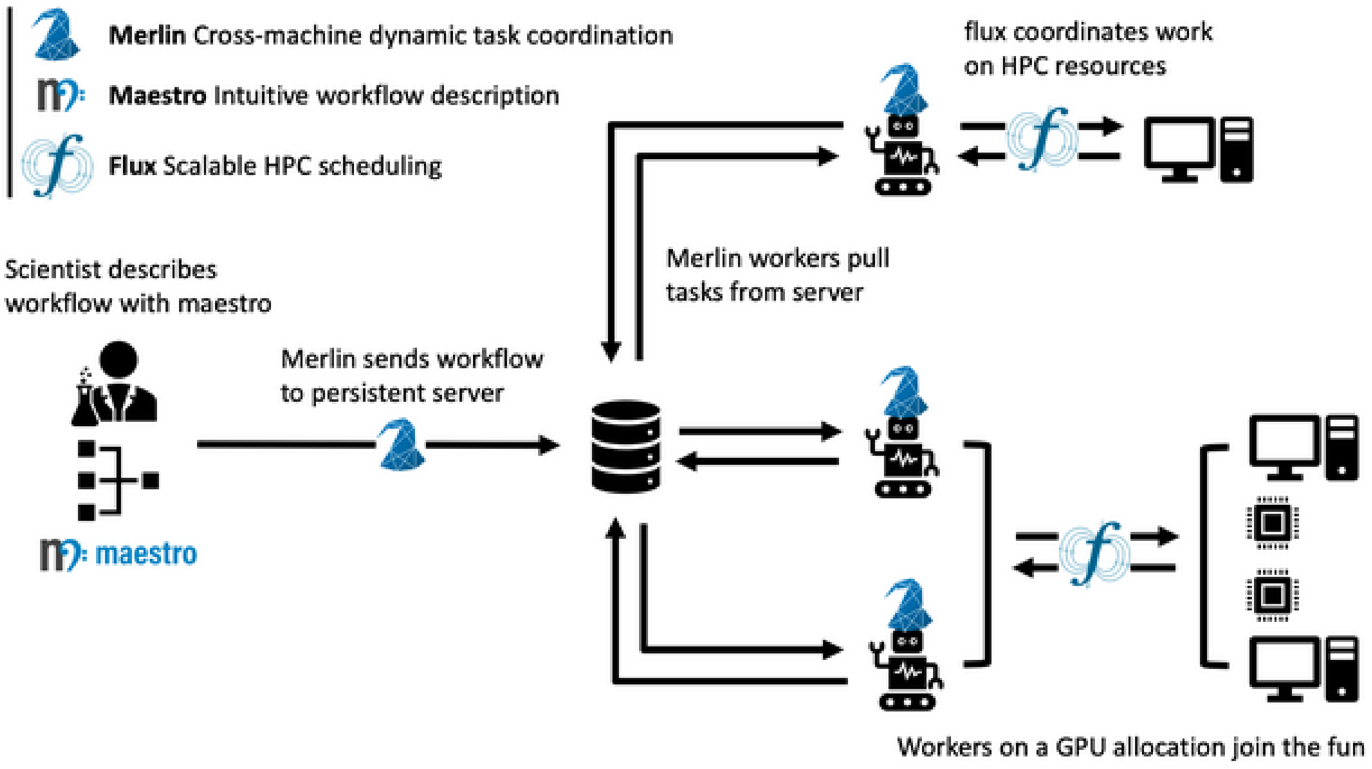
The Challenge

- When scaling our application/workflows to scientific applications we are faced with a multitude of problems:
  - Data movement
  - Amount of data
  - Concurrent processes, which might be prone to crashing
  - Application portability
- This is the point where probabilistic programming turns into an HPC engineering problem, where we have choose our frameworks wisely to get the most performance out of the hardware
- Offshot: Will also run much more efficient/faster on small computers

Merlin

- Manager for HPC-focuses simulation workflow, which can include multiple supercomputers, cloud components, as well as heterogeneous clusters.
- Made to scale to multiple hundred million simulations at a time
  - (Broken down into individual batches of simulations)
- Can automatically take advantage of spare computing capacity
- Persistent in case of individual servers failing
- Can be likened to a microservices-focussed architecture for HPC simulations

Merlin



Merlin Cross-machine dynamic task coordination

Maestro Intuitive workflow description

Flux Scalable HPC scheduling

Scientist describes workflow with maestro

Merlin sends workflow to persistent server

Merlin workers pull tasks from server

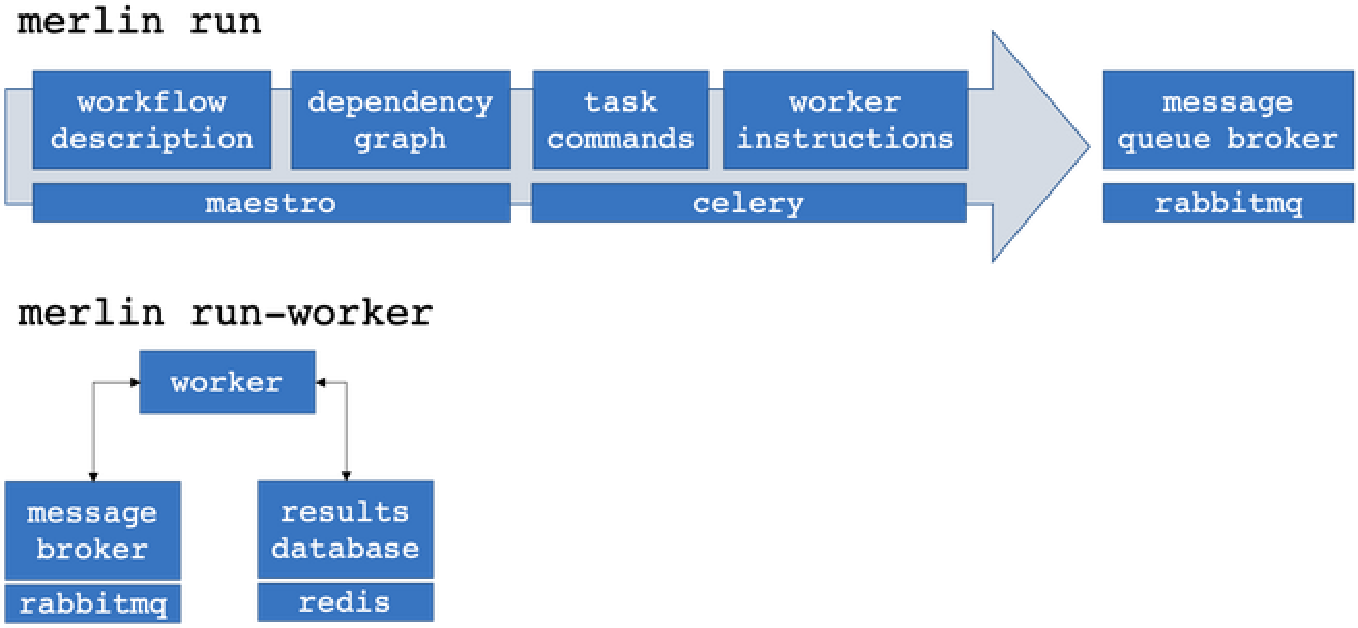
flux coordinates work on HPC resources

Workers on a GPU allocation join the fun

Merlin

- Asynchronous task queuing library (Merlin)
- Dynamic task queuing (Celery)
- Description of our model is contained in a central workflow file
- Resilience and support for multiple users and queues (RabbitMQ)
- Scalability and fast database access (Redis)

Merlin


merlin run
workflow description
dependency graph
task commands
worker instructions
maestro
celery
message queue broker
rabbitmq
merlin run-worker
worker
message broker
rabbitmq
results database
redis

Merlin

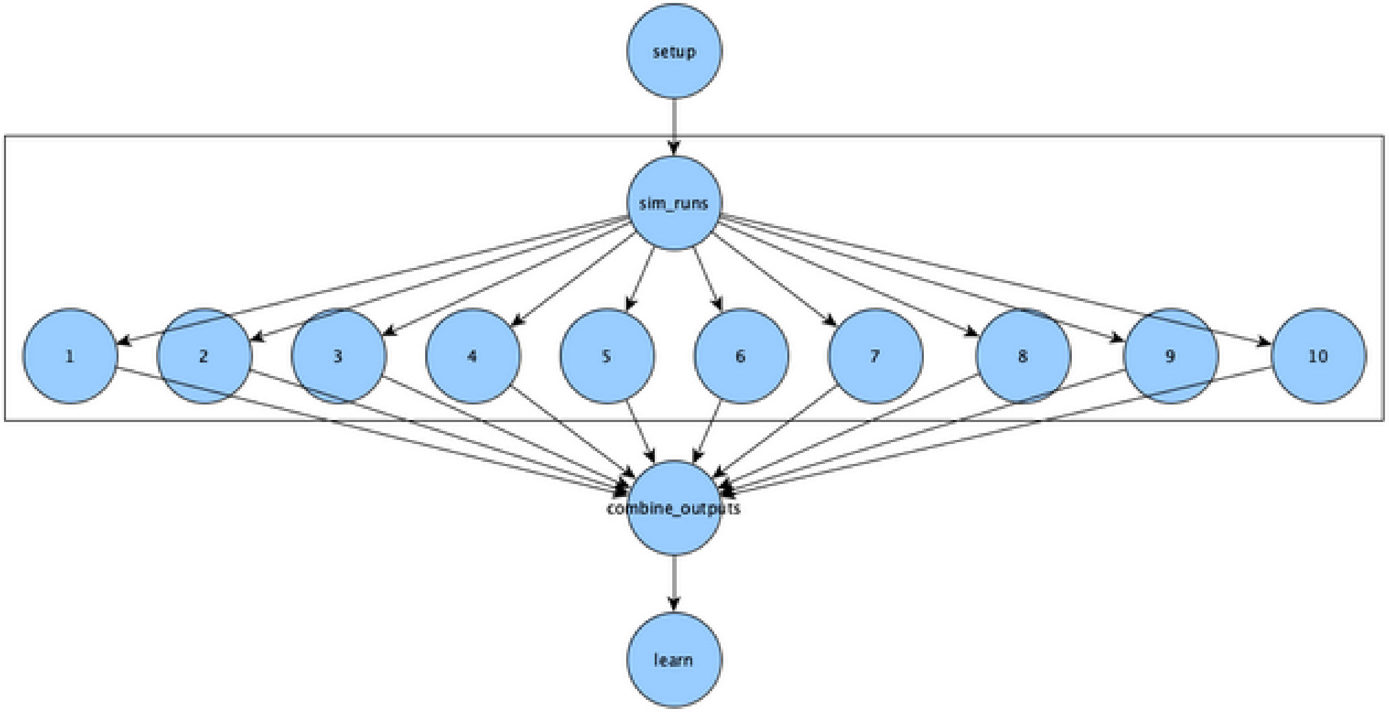
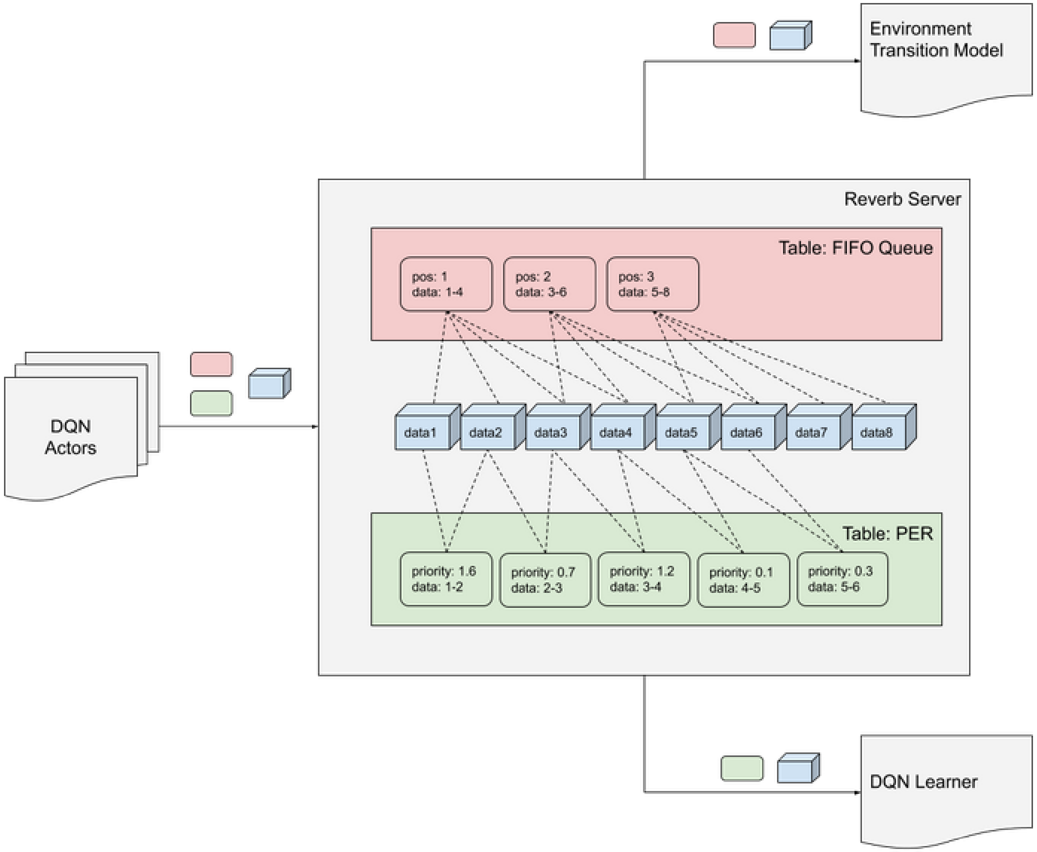


Figure: Any workflow we can define as a directed-acyclic graph can be performed.

Reverb

- Experience replay system originally built for off-policy reinforcement learning
  - Can be adapted to probabilistic inference as well
- Reverb server consists of one or more tables, where tables also define sample and removal strategies, maximum item capacity, and a rate limiter.
- Multiple item items can refer to the same data element, as entries in tables only contain references to the data
- Data is deleted if there is no reference to it anymore

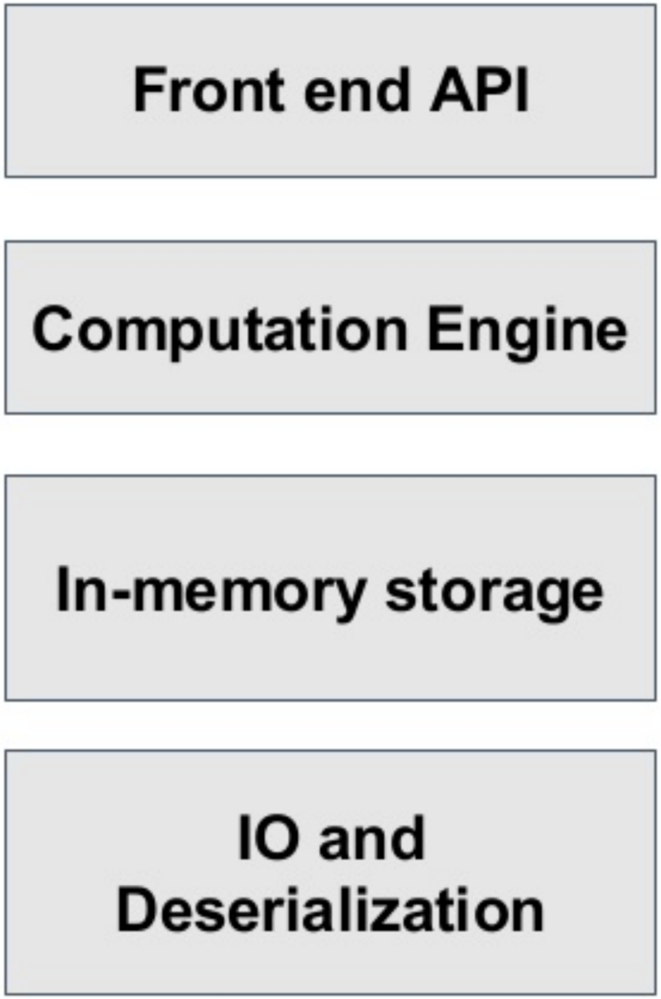
Reverb


Environment
Transition Model
Reverb Server
Table: FIFO Queue
pos: 1
data: 1-4
pos: 2
data: 3-6
pos: 3
data: 5-8
DQN
Actors
data1
data2
data3
data4
data5
data6
data7
data8
Table: PER
priority: 1.6
data: 1-2
priority: 0.7
data: 2-3
priority: 1.2
data: 3-4
priority: 0.1
data: 4-5
priority: 0.3
data: 5-6
DQN Learner

Apache Arrow

- OSS Community initiative conceived in 2015
- Sits right at the intersection of big data, database systems, and data science tools
- Core of the project: Language agnostic open standards to accelerate in-memory computing
- Components have public APIs
- Arrow is front-end agnostic
- Dataframe backend for NVIDIA Rapids

https://github.com/apache/arrow



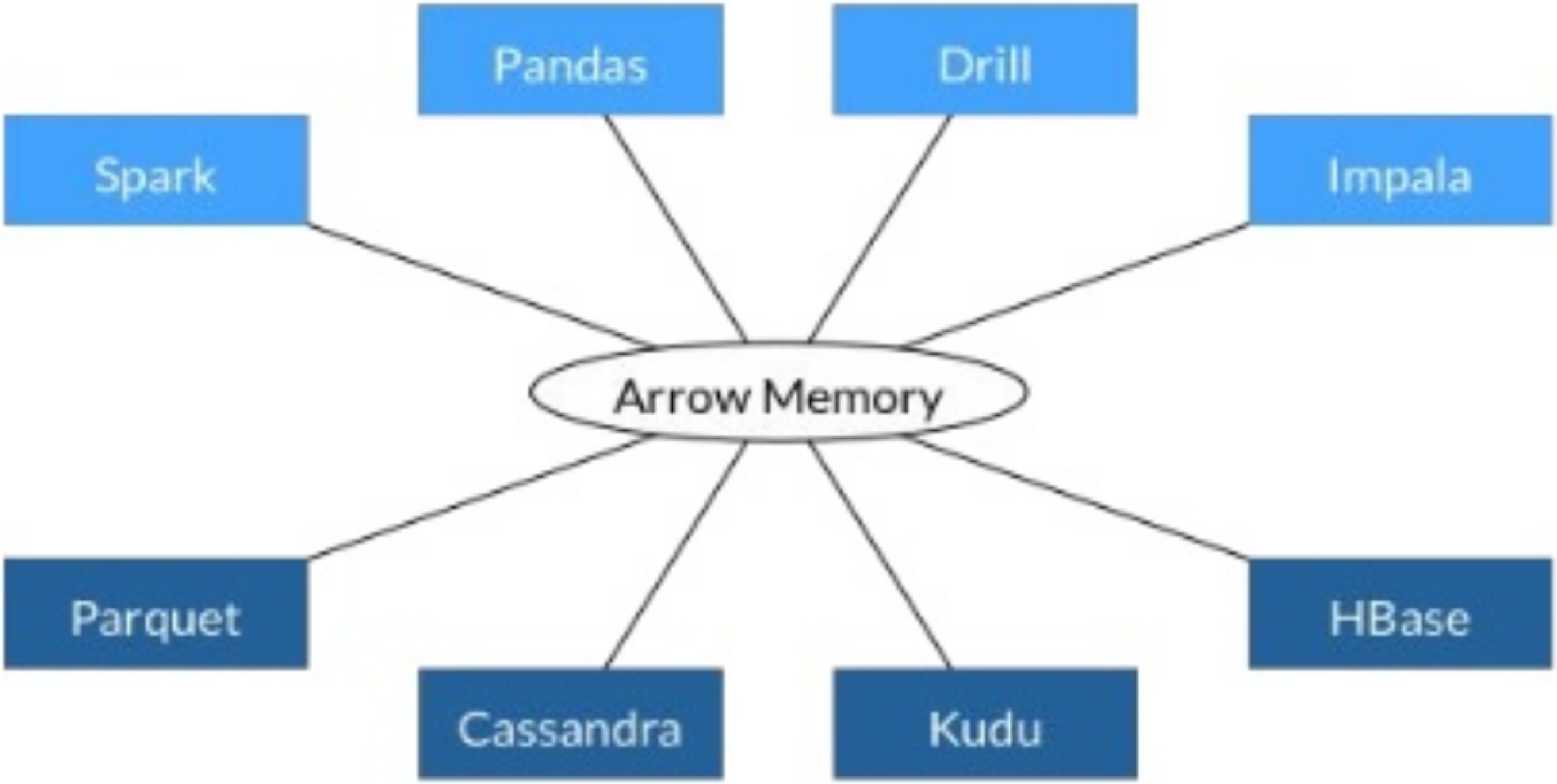
Front end API

Computation Engine

In-memory storage

IO and Deserialization

Apache Arrow



Apache Arrow

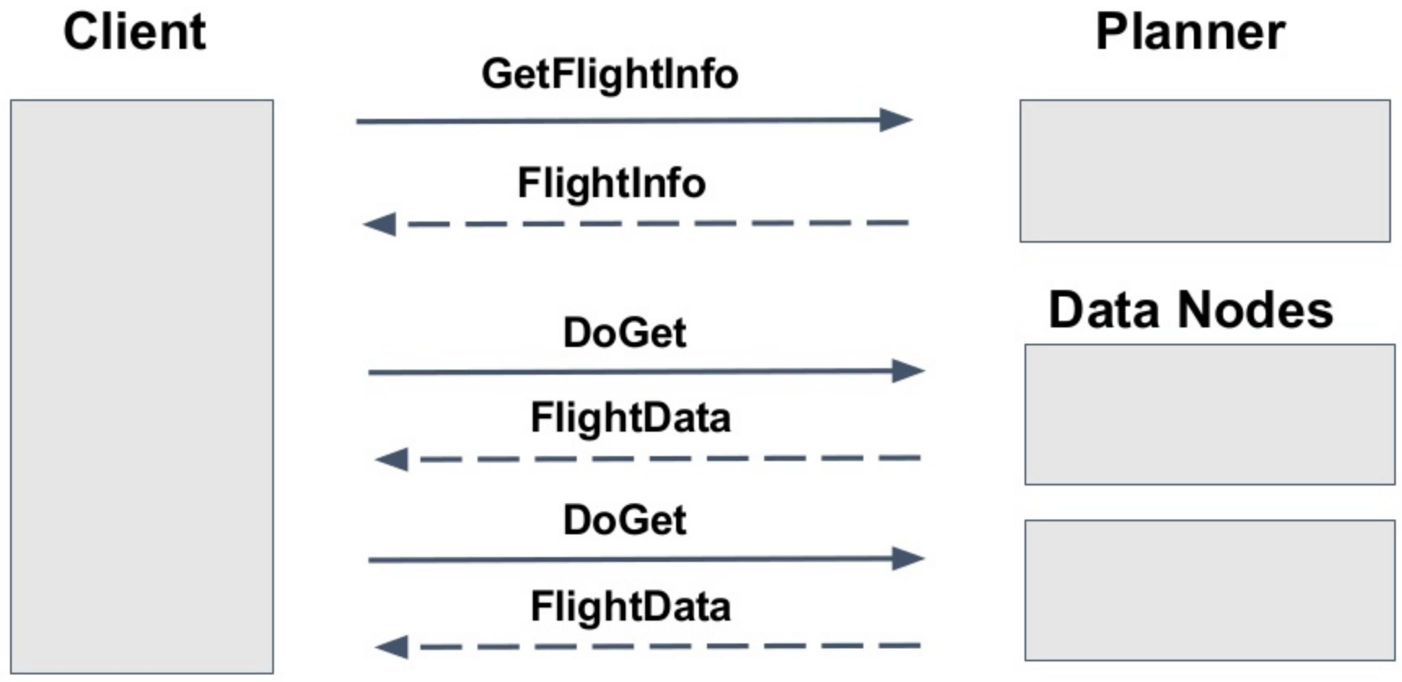
- Applied to eliminate serialization overhead in data interchange
- Improve CPU/GPU in-memory processing efficiency
- Simplify architectures
- Organized for cache-efficient access on CPUs/GPUs
- Optimized for data locality, SIMD, and parallel processing
- In-memory encoding, compression, and sparseness are in the works

Apache Arrow

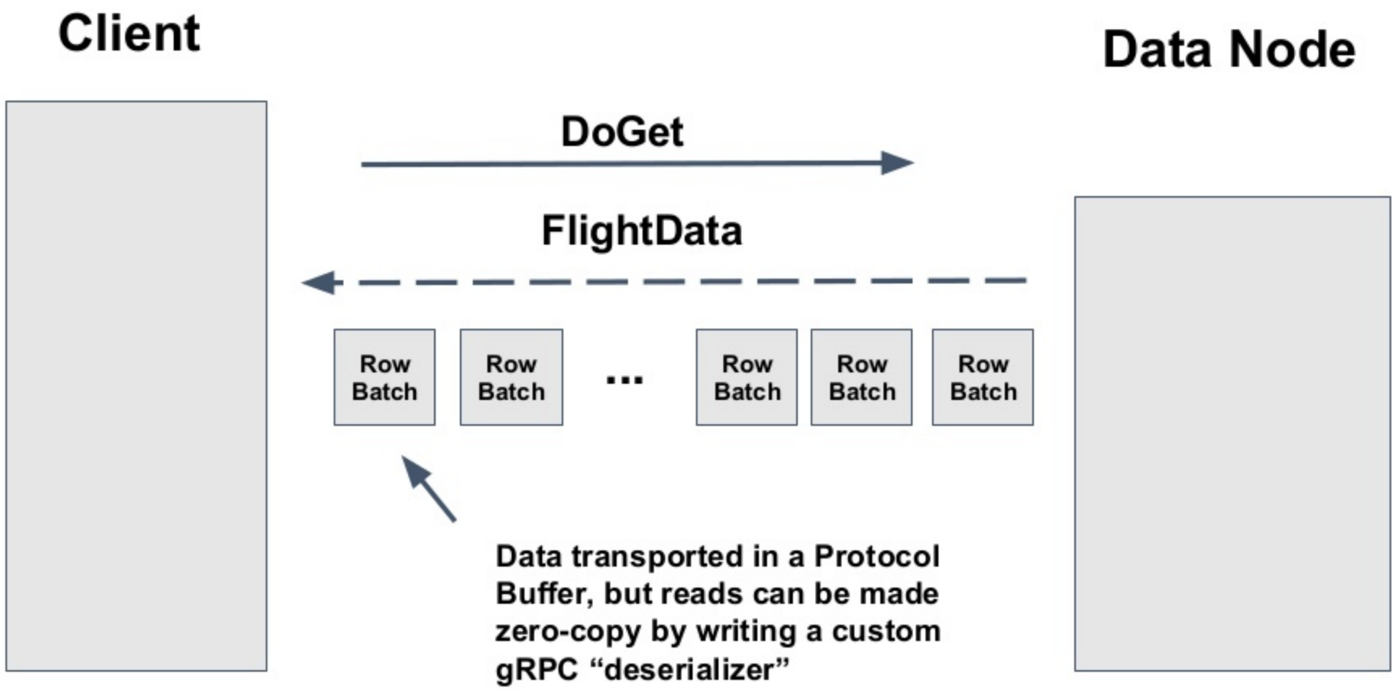
Apache Arrow Flight

- Framework to define custom data services and receive Arrow data natively
  - Avoids any copying or deserializatio
- Low-level optimizations
- Made for continous extreme-scale datastreams, i.e. also useful for HPC applications

Apache Arrow

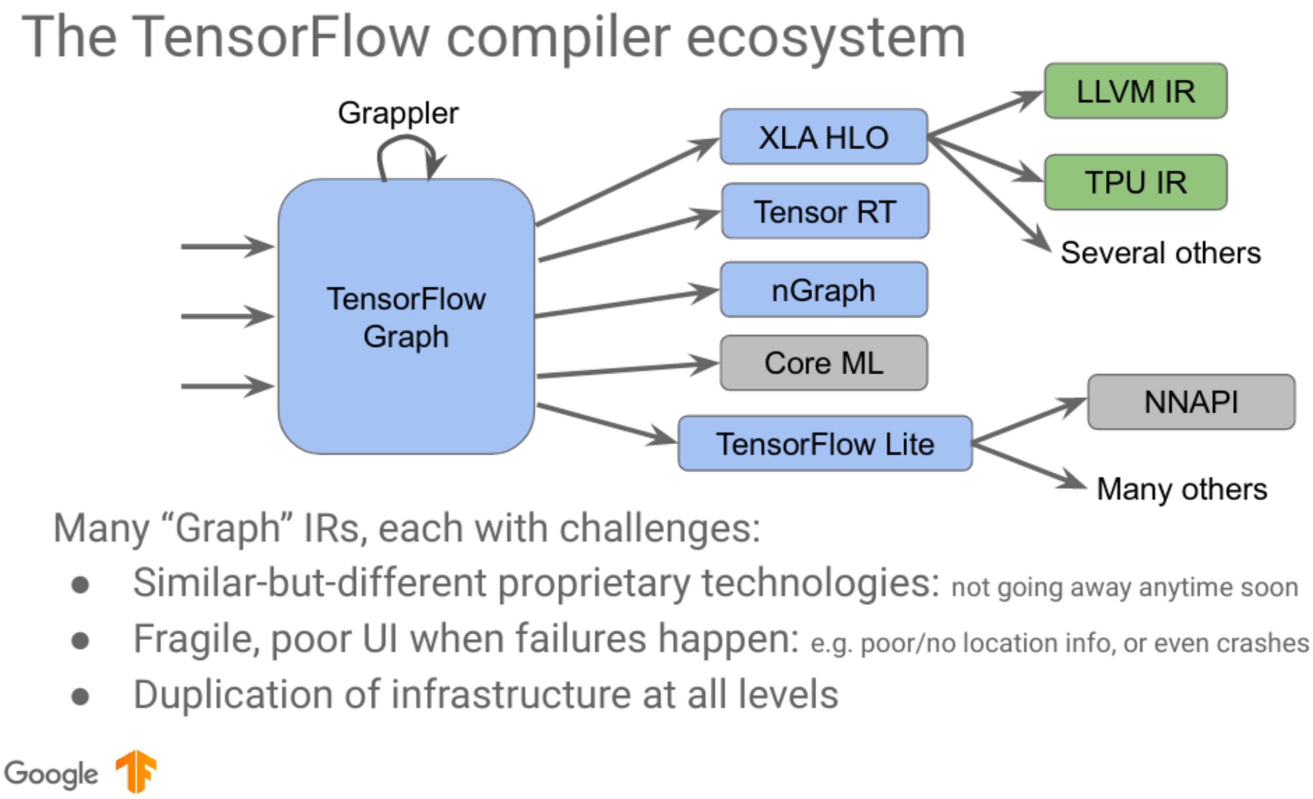


Apache Arrow



Data transported in a Protocol Buffer, but reads can be made zero-copy by writing a custom gRPC “deserializer”

Machine Learning Compilers



The TensorFlow compiler ecosystem

Many “Graph” IRs, each with challenges:

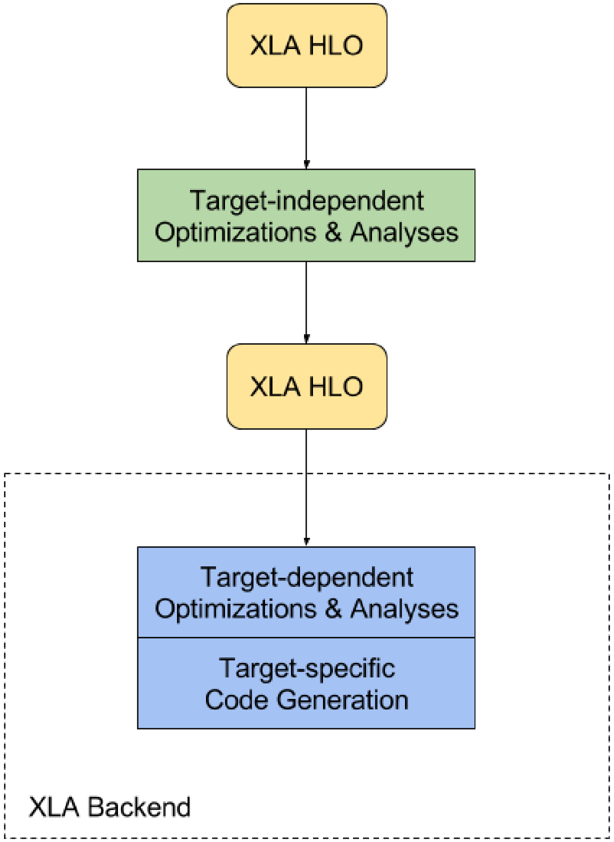
- Similar-but-different proprietary technologies: not going away anytime soon
- Fragile, poor UI when failures happen: e.g. poor/no location info, or even crashes
- Duplication of infrastructure at all levels

XLA

Purpose behind XLA:

- Improve the execution speed
- Improve memory usage
- Reduce reliance on custom operations
- Reduce mobile footprint
- Improve portability
  - Accelerator-producers just need to implement the XLA API, and our code will run seamlessly on accelerators of all types

XLA


XLA HLO
Target-independent Optimizations & Analyses
XLA HLO
Target-dependent Optimizations & Analyses
Target-specific Code Generation
XLA Backend

The Approach

- Data movement ⟶ Apache Arrow
- Amount of data ⟶ Apache Arrow + Compression
- Concurrent processes, which might be prone to crashing ⟶ Merlin
- Application portability ⟶ XLA or MLIR