Meta

Sponsor Talk

CIDR'23

Pedro Pedreira - pedroerp@fb.com
Software Engineer

1/9/23

Velox: Meta's Unified Execution Engine

CIDR'23

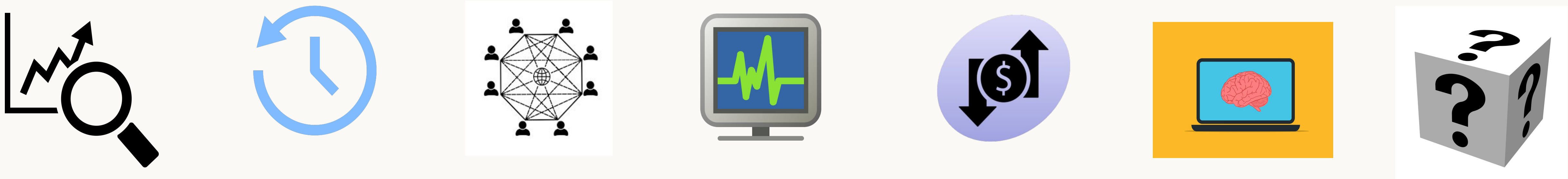
Pedro Pedreira - pedroerp@fb.com
Software Engineer

1/9/23

Motivation

User Workload Variety

Analytics Realtime Graph Monitoring Transactional ML ...?



Engine Specialization

"One size does not fit all"

Analytics

- Presto
- Spark
- Saber
- Cubrick

Realtime Infra

- XStream
- Scribe
- FBETL

Graph

- DIGraph

Monitoring

- Scuba
- ODS
- Logarithm

Transactional

- MySQL
- RocksDB
- XSQL

ML

- TorchArrow/PyTorch
- F3
- Koski

Engine Specialization

The flipside

- Very limited reusability.
- Duplicates efforts and forces engineers to reinvent the wheel.
- Hard to maintain and enhance.
  - Where do we optimize?
- Exposes inconsistencies to end-users.
- Hurts our capacity to move fast and innovate.

Analytics

- Presto
- Spark
- Saber
- Cubrick

Realtime Infra

- XStream
- Scribe
- FBETL

Graph

- DIGraph

Monitoring

- Scuba
- ODS
- Logarithm

Transactional

- MySQL
- RocksDB
- XSQLm

ML

- TorchArrow/PyTorch
- F3
- Koski

Through the Looking Glass 🔍

Different, but not really...

	Presto	Spark	XStream	Scuba	Cubrick	Koski	F3
Language Frontend	*Presto SQL*	*HQL / DataSet API*	*UPM*	*Scuba SQL*	*Cubrick SQL*	*Koski DataFrame*	*F3 DSL*
IR	*Presto IR*	*Spark IR*	*UPM IR*	*Scuba IR*	*Cubrick IR*	*Koski IR*	*F3 IR*
Optimizer	*Presto Optimizer*	*Catalyst*	*XStream Optimizer*	*Scuba Optimizer*	*Cubrick Optimizer*	*Koski Optimizer*	*F3 Optimizer*
Execution Runtime	*Presto Runtime*	*Spark Runtime*	*XStream Runtime*	*Scuba Runtime*	*Cubrick Runtime*	*DPP*	-
Execution Engine	*Presto*	*Spark*	*XStream Codegen*	*Scuba*	*Cubrick*	*Koski*	*F3 DAG*

Through the Looking Glass 🔍

Different, but not really...

	Presto	Spark	XStream	Scuba	Cubrick	Koski	F3
Language Frontend	*Presto SQL*	*HQL / DataSet API*	*UPM*	*Scuba SQL*	*Cubrick SQL*	*Koski DataFrame*	*F3 DSL*
IR	*Presto IR*	*Spark IR*	*UPM IR*	*Scuba IR*	*Cubrick IR*	*Koski IR*	*F3 IR*
Optimizer	*Presto Optimizer*	*Catalyst*	*XStream Optimizer*	*Scuba Optimizer*	*Cubrick Optimizer*	*Koski Optimizer*	*F3 Optimizer*
Execution Runtime	*Presto Runtime*	*Spark Runtime*	*XStream Runtime*	*Scuba Runtime*	*Cubrick Runtime*	*DPP*	-
Execution Engine	*Presto*	*Spark*	*XStream Codegen*	*Scuba*	*Cubrick*	*Koski*	*F3 DAG*

All very similar!

Through the Looking Glass

Different, but not really...

	Presto	Spark	XStream	Scuba	Cubrick	Koski	F3
Language Frontend	*Presto SQL*	*HQL / DataSet API*	*UPM*	*Scuba SQL*	*Cubrick SQL*	*Koski DataFrame*	*F3 DSL*
IR	*Presto IR*	*Spark IR*	*UPM IR*	*Scuba IR*	*Cubrick IR*	*Koski IR*	*F3 IR*
Optimizer	*Presto Optimizer*	*Catalyst*	*XStream Optimizer*	*Scuba Optimizer*	*Cubrick Optimizer*	*Koski Optimizer*	*F3 Optimizer*
Execution Runtime	*Presto Runtime*	*Spark Runtime*	*XStream Runtime*	*Scuba Runtime*	*Cubrick Runtime*	*DPP*	-
Execution Engine							

All very similar!

Velox

Velox

Meta

Velox

Velox Mission

Converge, Accelerate, and Unify execution engines across Meta and beyond

Velox Library Overview

- A generic **C++ database acceleration library.**
  - Generic APIs: from batch to interactive, to stream processing, to AI/ML workloads.
    - Key Concepts: **Modularity** and **Extensibility**.
  - C++: native code for maximum efficiency
    - **10x** cpp vs. java win (TPC-H Q1 and Q6 microbenchmarks).
  - State-of-art
    - Centralize all optimizations implemented in current engines.

Velox Library Overview (2)

- Database acceleration library vs. DBMS.
- Velox takes a fully optimized **physical plan** as input.
    - No frontend (SQL parser or dataframe layer)
    - No global optimizer.
- Though there's tons of **adaptivity.**
- Velox sits on the data-path
    - Everything that runs on a single server.
- No control plane.

Velox - Value Proposition

01
Efficiency and Latency

02
Consistency and Consolidation

03
Reusability

Use Cases

Velox - Use Cases

- Analytics:
  - Presto/Prestissimo - interactive
  - Spark//Gluten - batch
  - Saber - external analytics
- Realtime Infrastructure:
  - XStream - stream processing
  - FBETL/Morse - data warehouse and database ingestion
  - Scribe - log messaging system
- Transactional:
  - XSQL - distributed transaction processing
- Machine Learning:
  - TorchArrow/PyTorch - data preprocessing
  - F3 - feature engineering
  - XLDB/Koski - training

Velox - Open Source

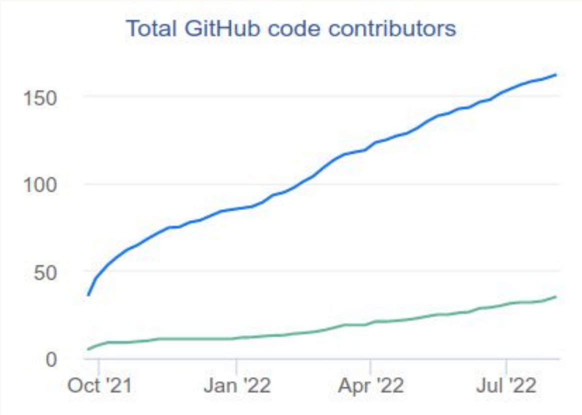
- Publicly announced in Oct 22!
  - https://engineering.fb.com/2022/08/31/open-source/velox/
- Available in github:
  - https://github.com/facebookincubator/velox
- VLDB'22:
  - *"Velox: Meta's Unified Execution Engine"*
- Fast growing open source community
  - +180 developers
  - Meta, Ahana, Intel, Voltron Data, ...


Engineering at Meta

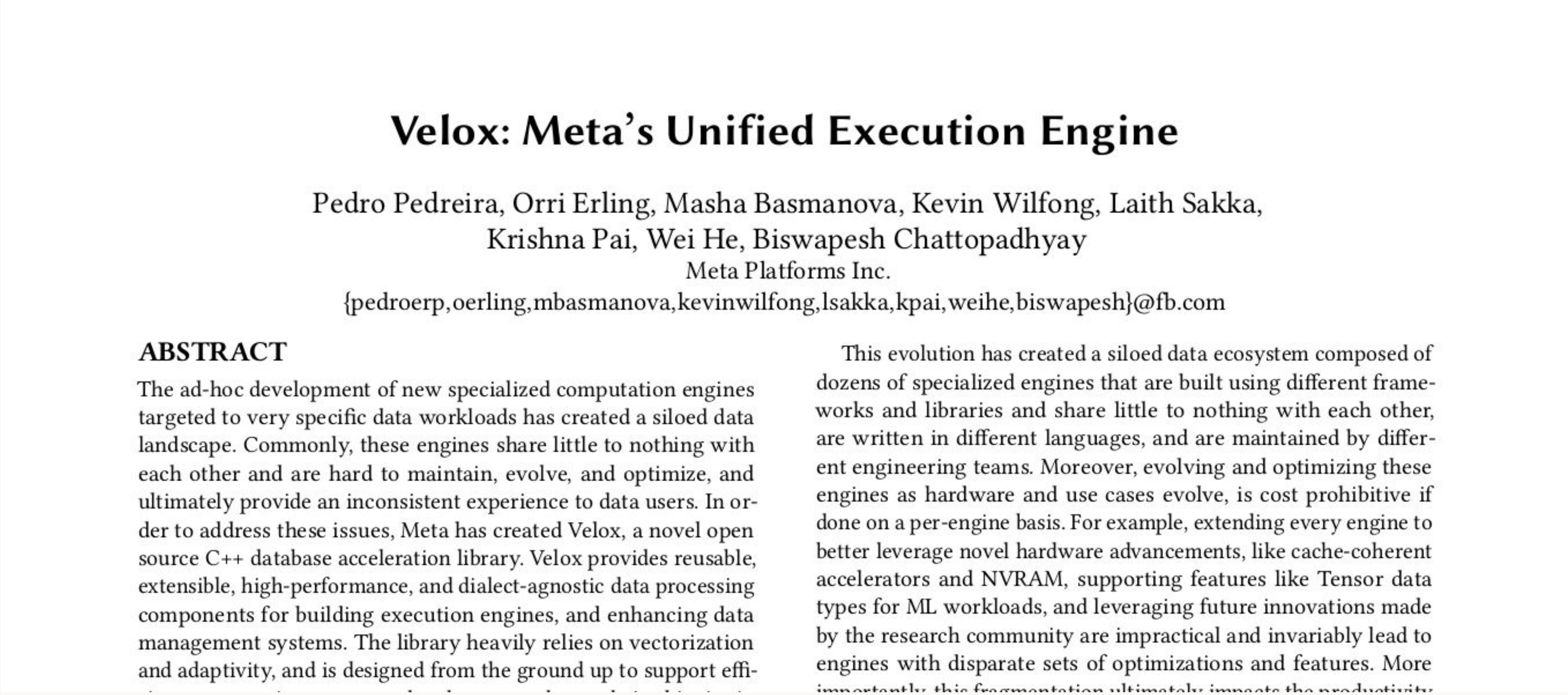
Open Source | Platforms | Infrastructure Systems | Physical Infrastructure | Video Engineering & AR/VR

POSTED ON AUGUST 31, 2022 TO DATA INFRASTRUCTURE, OPEN SOURCE

Introducing Velox: An open source unified execution engine



# of contributors


Velox: Meta's Unified Execution Engine

Pedro Pedreira, Orri Erling, Masha Basmanova, Kevin Wilfong, Laith Sakka,
Krishna Pai, Wei He, Biswapesh Chattopadhyay
Meta Platforms Inc.
{pedroerp,oerling,mbasmanova,kevinwilfong,lsakka,kpai,weihe,biswapesh}@fb.com

ABSTRACT

The ad-hoc development of new specialized computation engines targeted to very specific data workloads has created a siloed data landscape. Commonly, these engines share little to nothing with each other and are hard to maintain, evolve, and optimize, and ultimately provide an inconsistent experience to data users. In order to address these issues, Meta has created Velox, a novel open source C++ database acceleration library. Velox provides reusable, extensible, high-performance, and dialect-agnostic data processing components for building execution engines, and enhancing data management systems. The library heavily relies on vectorization and adaptivity, and is designed from the ground up to support effi-

This evolution has created a siloed data ecosystem composed of dozens of specialized engines that are built using different frameworks and libraries and share little to nothing with each other, are written in different languages, and are maintained by different engineering teams. Moreover, evolving and optimizing these engines as hardware and use cases evolve, is cost prohibitive if done on a per-engine basis. For example, extending every engine to better leverage novel hardware advancements, like cache-coherent accelerators and NVRAM, supporting features like Tensor data types for ML workloads, and leveraging future innovations made by the research community are impractical and invariably lead to engines with disparate sets of optimizations and features. More

Library Outline

Velox Library - Components Outline

- **Types:**
  - Scalar and nested data types, including structs, maps, arrays, tensors, and more.
- **Vectors:**
  - An *“Arrow-compatible”* columnar memory layout module.
- **Expression Eval:**
  - Fully vectorized expression evaluation engine built based on Vector-encoded data.
- **Functions:**
  - APIs for custom scalar (row-by-row and batch-by-batch) and aggregate functions.
- **Operators:**
  - Common data processing SQL operators (OrderBy, GroupBy, HashJoin, etc).
- **I/O:**
  - Pluggable file format encode/decoder, storage adapter, and network serializers.
- **Resource Management:**
  - Memory pools, arenas, thread/tasks, spilling, SSD and memory caching.

Ongoing Work

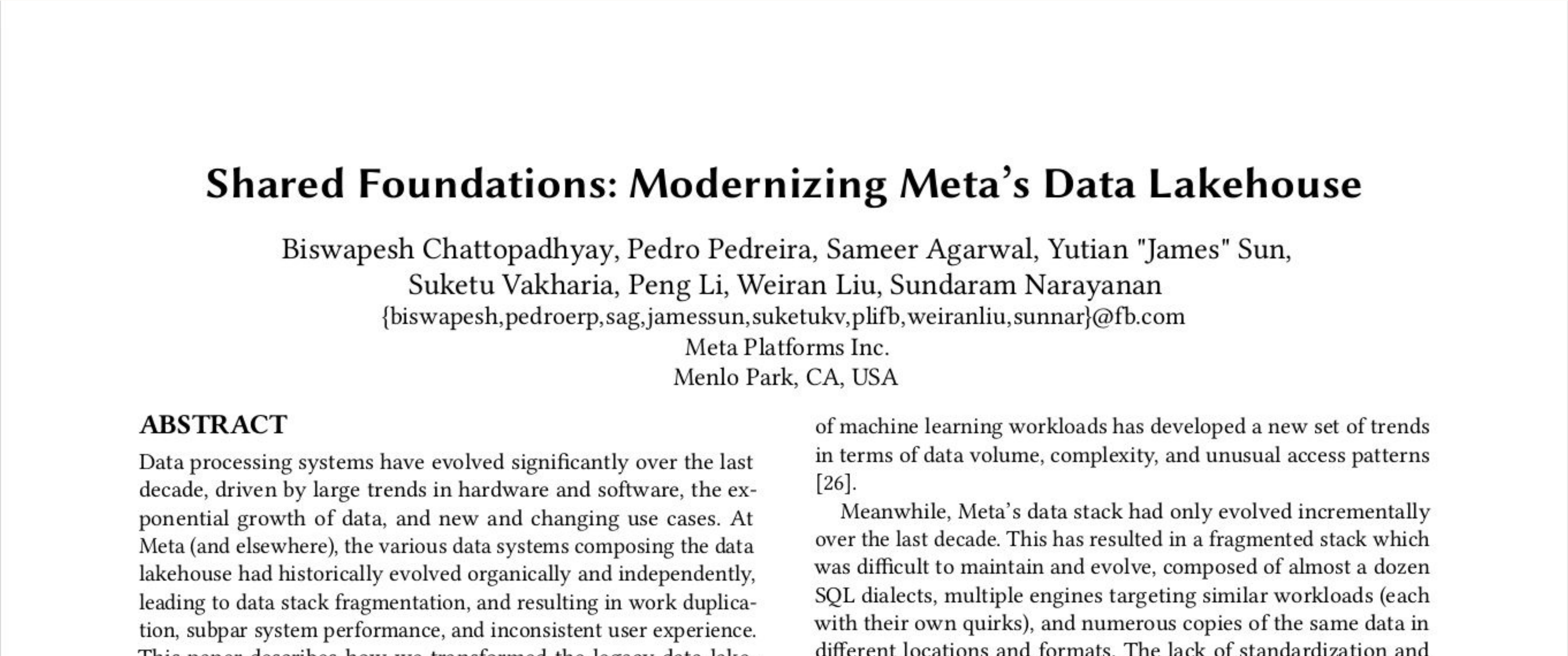
Ongoing Work - Where we need help

- Continue blurring the boundaries between Analytics and ML.
- Software and hardware co-evolution.
- Further componentization of the stack.
- More collaboration with academia!

Ongoing Work - Where we need help

- Continue blurring the boundaries between Analytics and ML.
- Software and hardware co-evolution.
- Further componentization of the stack.
- More collaboration with academia!

Tuesday 3:20pm @CIDR



Shared Foundations: Modernizing Meta's Data Lakehouse

Biswapesh Chattopadhyay, Pedro Pedreira, Sameer Agarwal, Yutian "James" Sun, Suketu Vakharia, Peng Li, Weiran Liu, Sundaram Narayanan
{biswapesh,pedroerp,sag,jamessun,suketukv,plifb,weiranliu,sunnar}@fb.com
Meta Platforms Inc.
Menlo Park, CA, USA

ABSTRACT

Data processing systems have evolved significantly over the last decade, driven by large trends in hardware and software, the exponential growth of data, and new and changing use cases. At Meta (and elsewhere), the various data systems composing the data lakehouse had historically evolved organically and independently, leading to data stack fragmentation, and resulting in work duplication, subpar system performance, and inconsistent user experience. This paper describes how we transformed the legacy data lake-

of machine learning workloads has developed a new set of trends in terms of data volume, complexity, and unusual access patterns [26].

Meanwhile, Meta's data stack had only evolved incrementally over the last decade. This has resulted in a fragmented stack which was difficult to maintain and evolve, composed of almost a dozen SQL dialects, multiple engines targeting similar workloads (each with their own quirks), and numerous copies of the same data in different locations and formats. The lack of standardization and

Thank you!

Q/A

Meta