OSA CON 24

Presto2.0 Native(C++) Engine at Meta and IBM

Aditi Pandit
Principal Engineer (Ahana/IBM)
LinkedIn

Amit Dutta
Software Engineer (Meta)
LinkedIn

November 19-21, 2024

Talk Outline

- Introduction to Presto
- Presto Native engine : what and why ?
- Production experience at Meta
- IBM watsonx.data
- Project roadmap and Call for participation

Introduction to Presto

presto

Fast and reliable Open-source SQL query engine for Data analytics and Open Data LakeHouse.

Meta

300PB data lakehouse
1K daily active users
30K queries/day

Uber

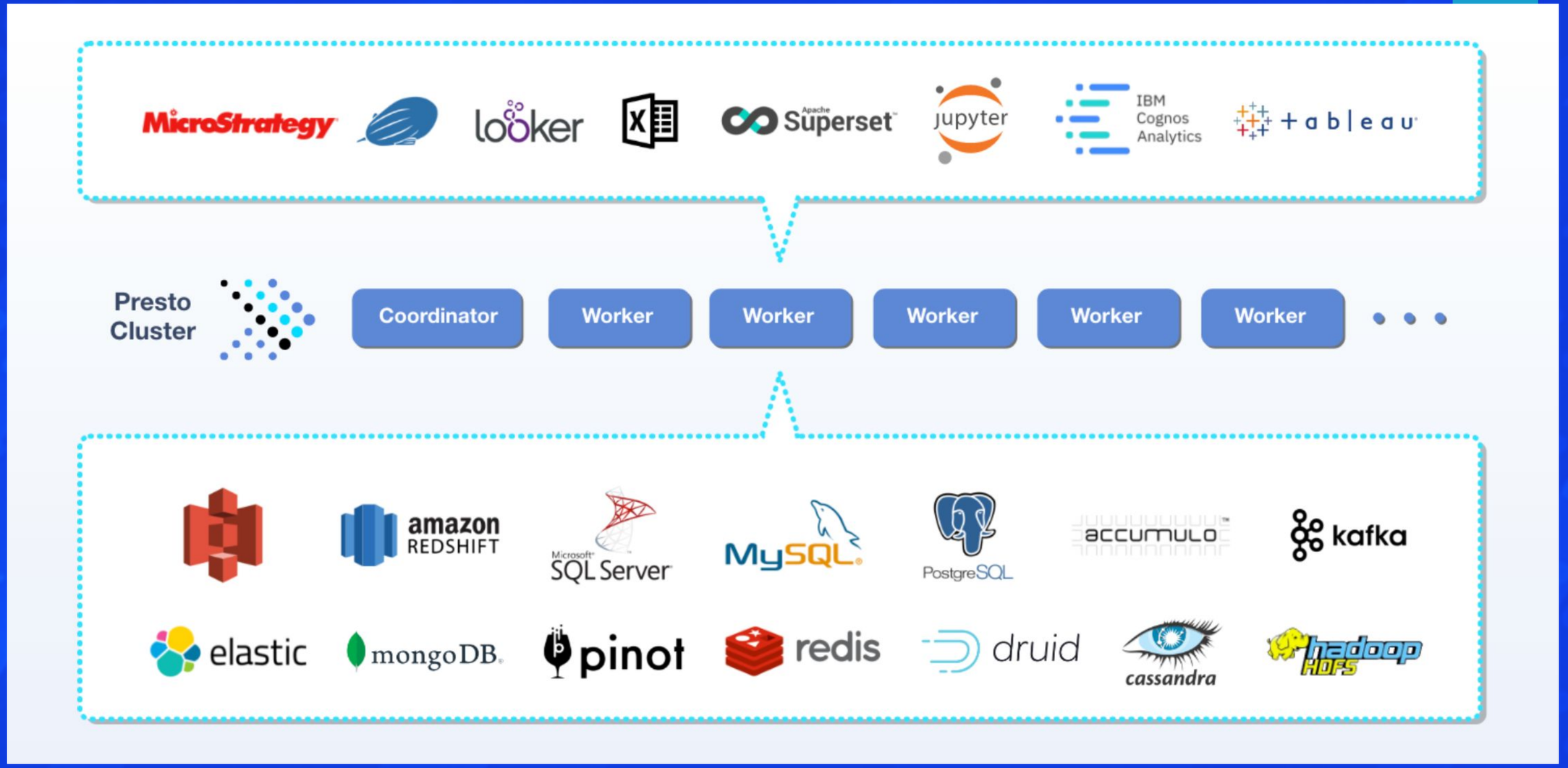
2 regions
20 clusters
8K nodes

7K weekly active users
100M+ queries/day
50PB HDFS bytes read/day

ByteDance

10K+ compute cores
1M queries/day

Presto Ecosystem



Presto has a vibrant Open Source community

Presto Users & Contributors

Meta Uber IBM Bolt Alibaba ALLUXIO

carbon كرتونة CARTONA etisalat by e& intel Hewlett Packard Enterprise blinkit

SECURONIX metropolis intuit adroitts. platform 24

ByteDance Tencent 腾讯 twilio RIPPLING

Presto Native Engine

What and Why ?

Presto/Presto Native Engine

Presto Circa 2020 : Latency/Efficiency/Scalability/Reliability going beyond warehouse analytics to the Open Data Lakehouse for diverse use-cases
Presto: A decade of SQL Analytics at Meta (Sigmod 2023)

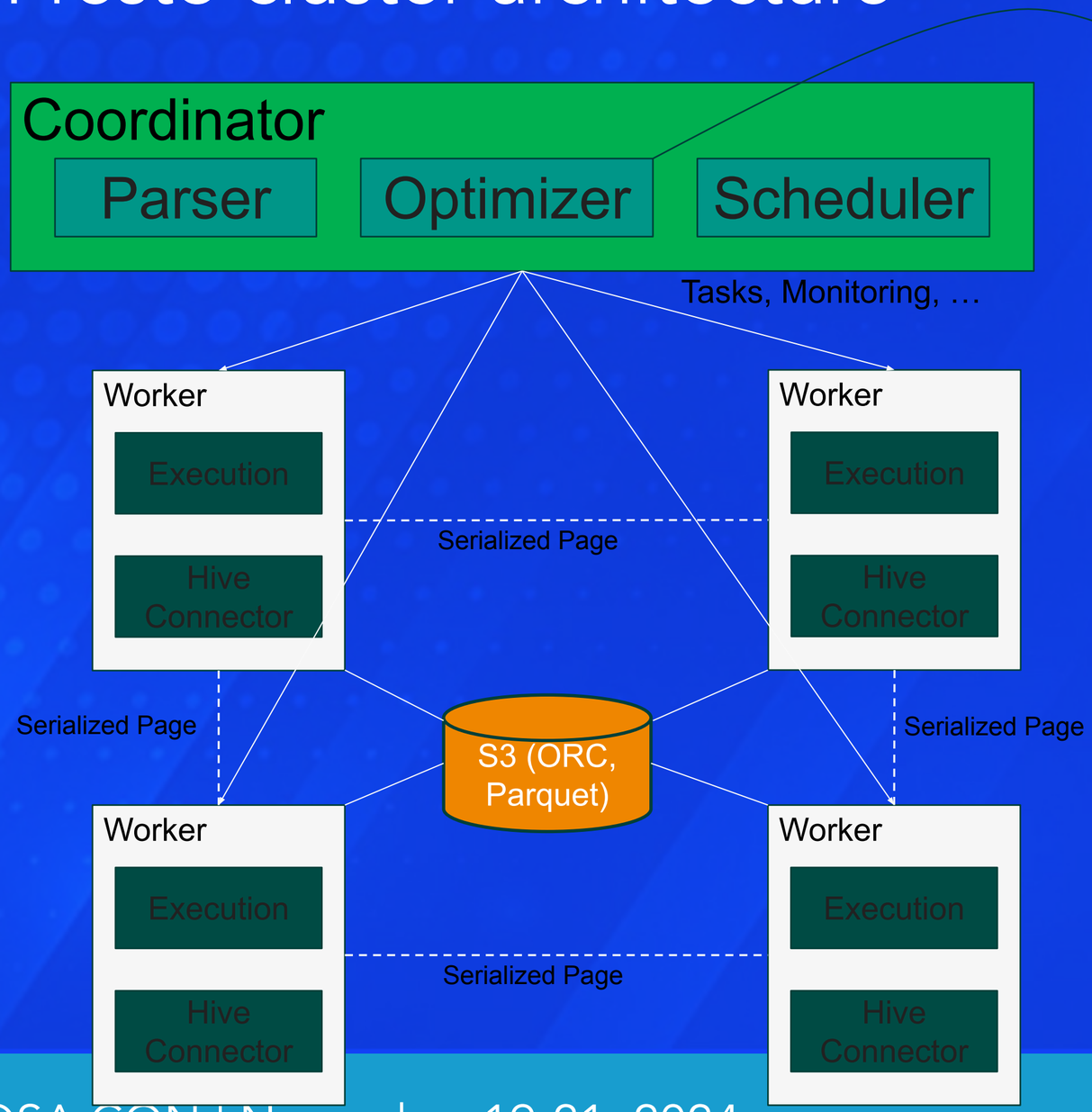
Project Prestissimo : Java -> C++ Query eval engine.
First-class Vectorization, Runtime optimizations and in-built memory management

Similar initiatives: Photon(Databricks), Gluten(Intel/Meta), Project Hummingbird(Trino) - Java but first-class vectorization

Prestissimo

- Prestissimo is a full re-write of the Presto worker in C++
- Drop in replacement for Presto Java worker
- Leverages Velox ref: Velox Metas Unified Execution Engine (VLDB 2022)
- Velox is a library of data processing primitives with in-built memory management

Presto cluster architecture



Java co-ordinator and workers

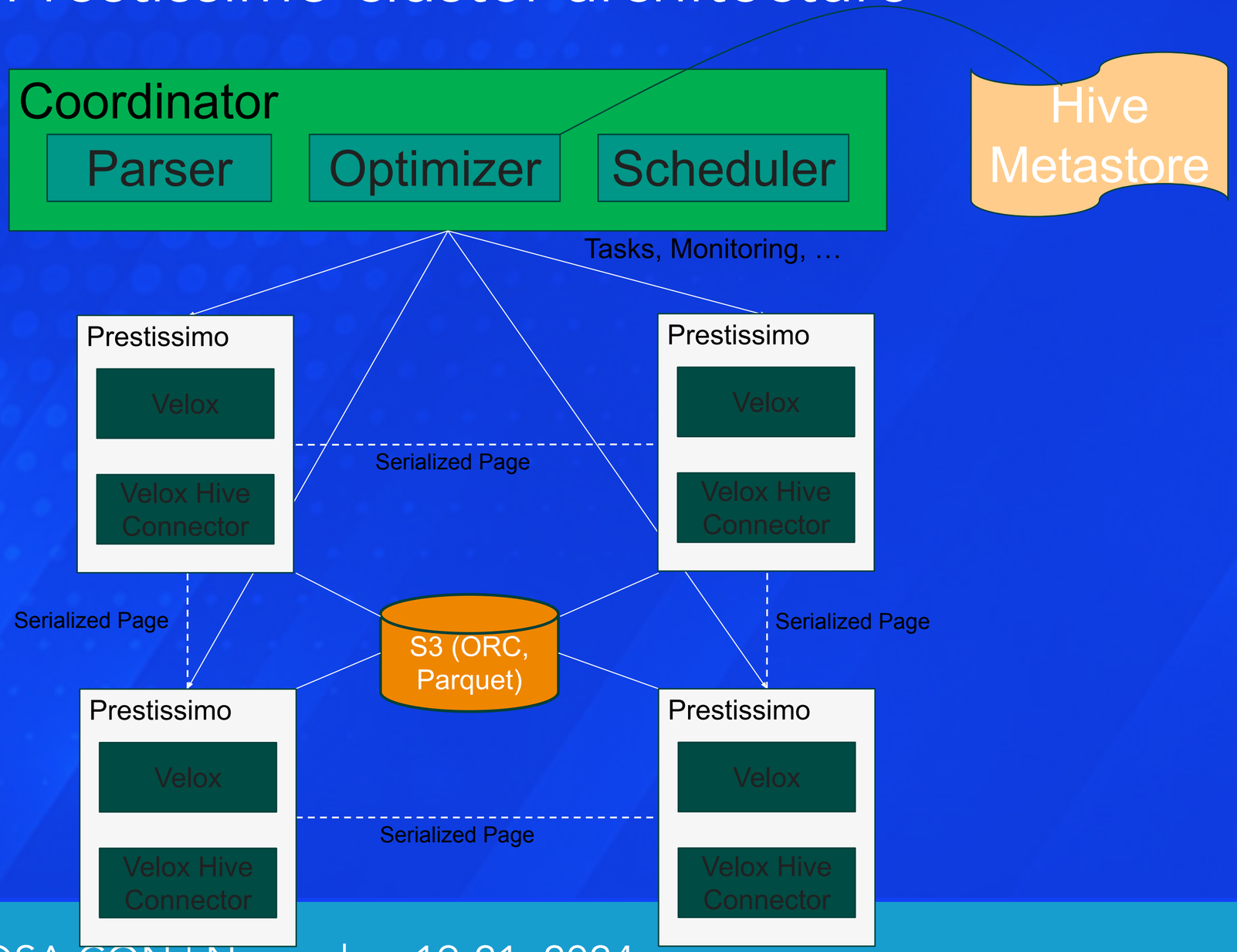
GC pauses

Performance cliffs

Operational difficulties

Hard to control bare metal level performance

Prestissimo cluster architecture



C++ worker

Built over the Velox library

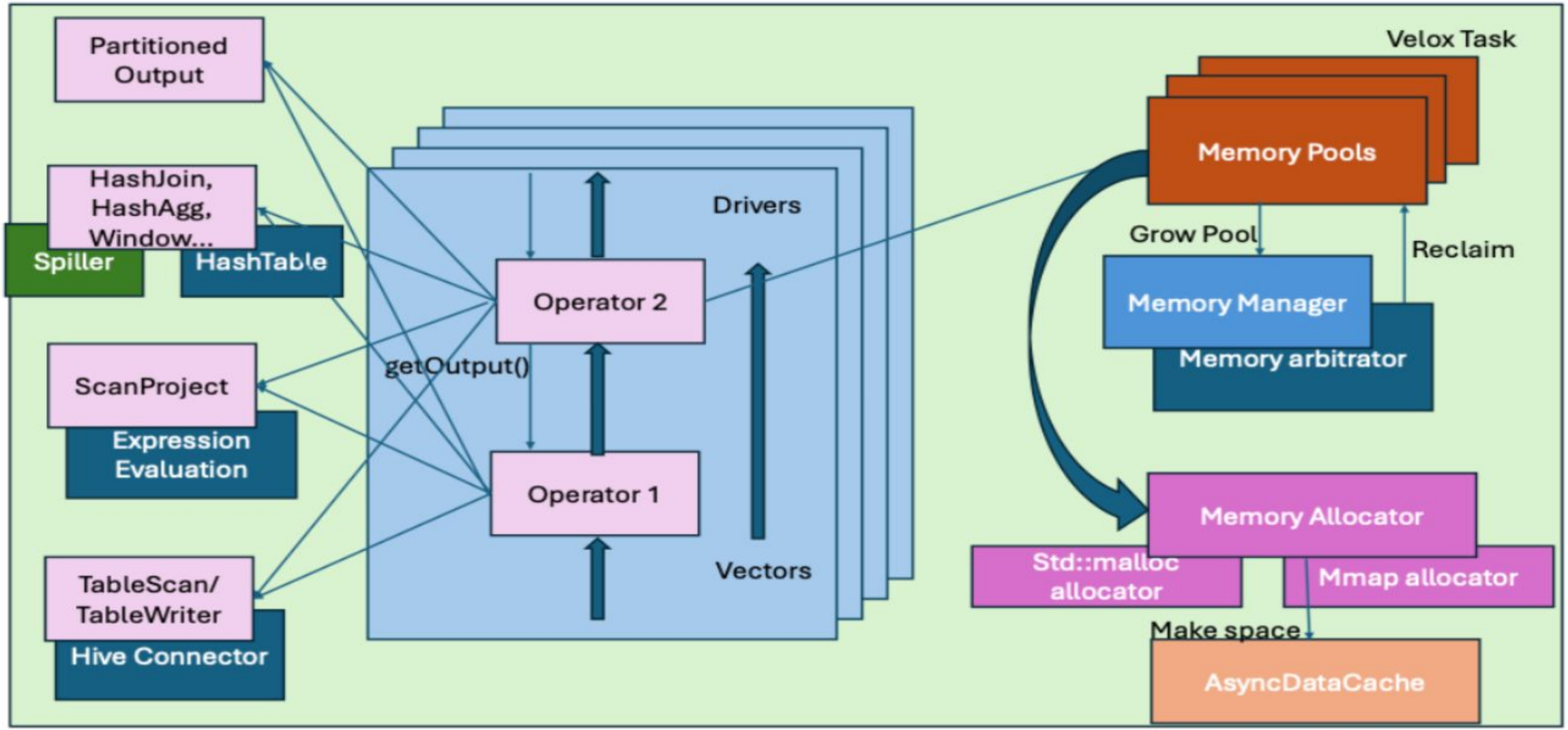
SIMD

Runtime optimizations

Smart I/O prefetching/caching

Memory Arbitration features

Velox at a glance



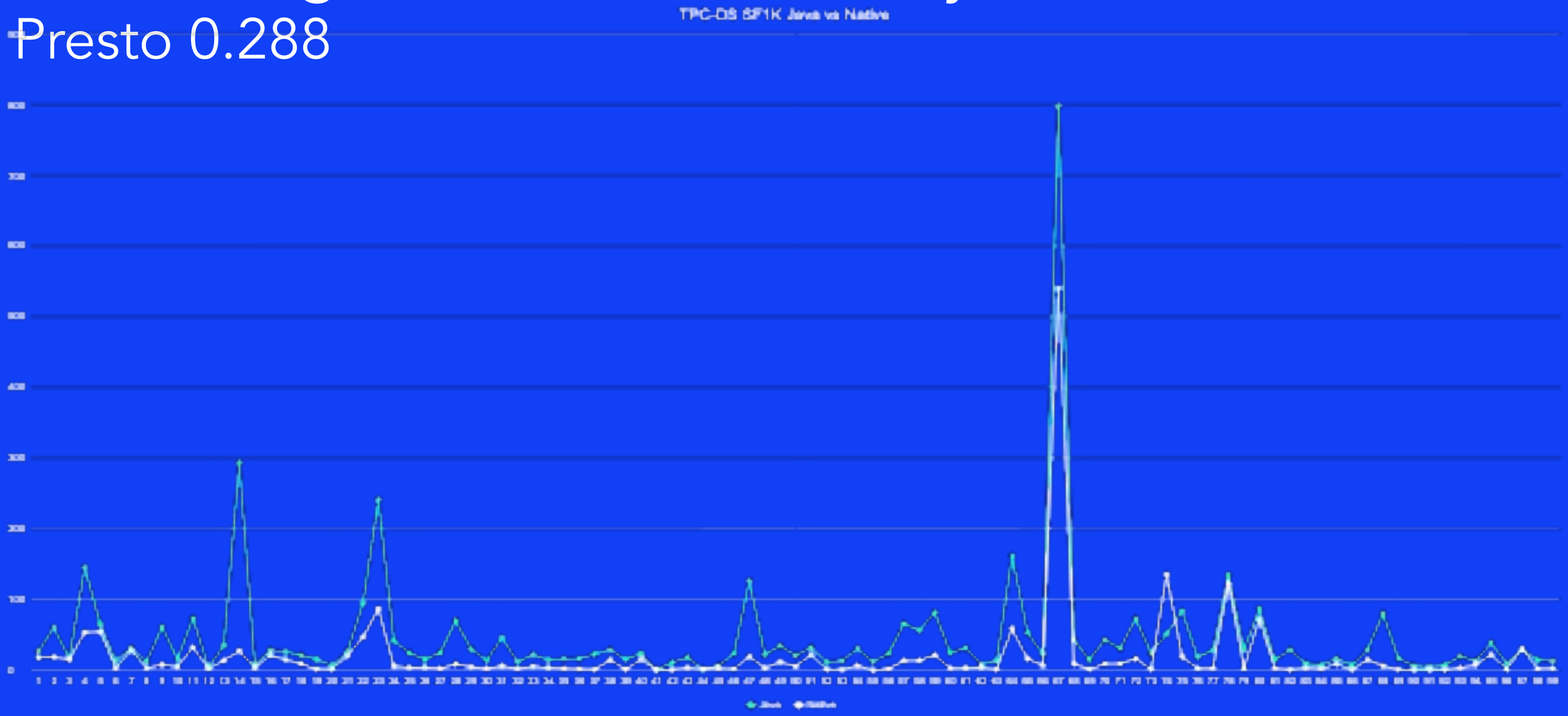
Benefits of Prestissimo/Velox

- Huge performance boost
  - Better numbers overall. Query processing can be done with much smaller clusters.
- Avoids performance cliffs
  - No Java processes, JVM or Garbage collection.
  - Memory management and SIMD are explicitly controlled in C++.
  - Memory arbitration improves efficiency.
- Easier to build and operate at scale
  - Reusable and extensible primitives across data engines (like Spark).
  - Performance can be better understood.

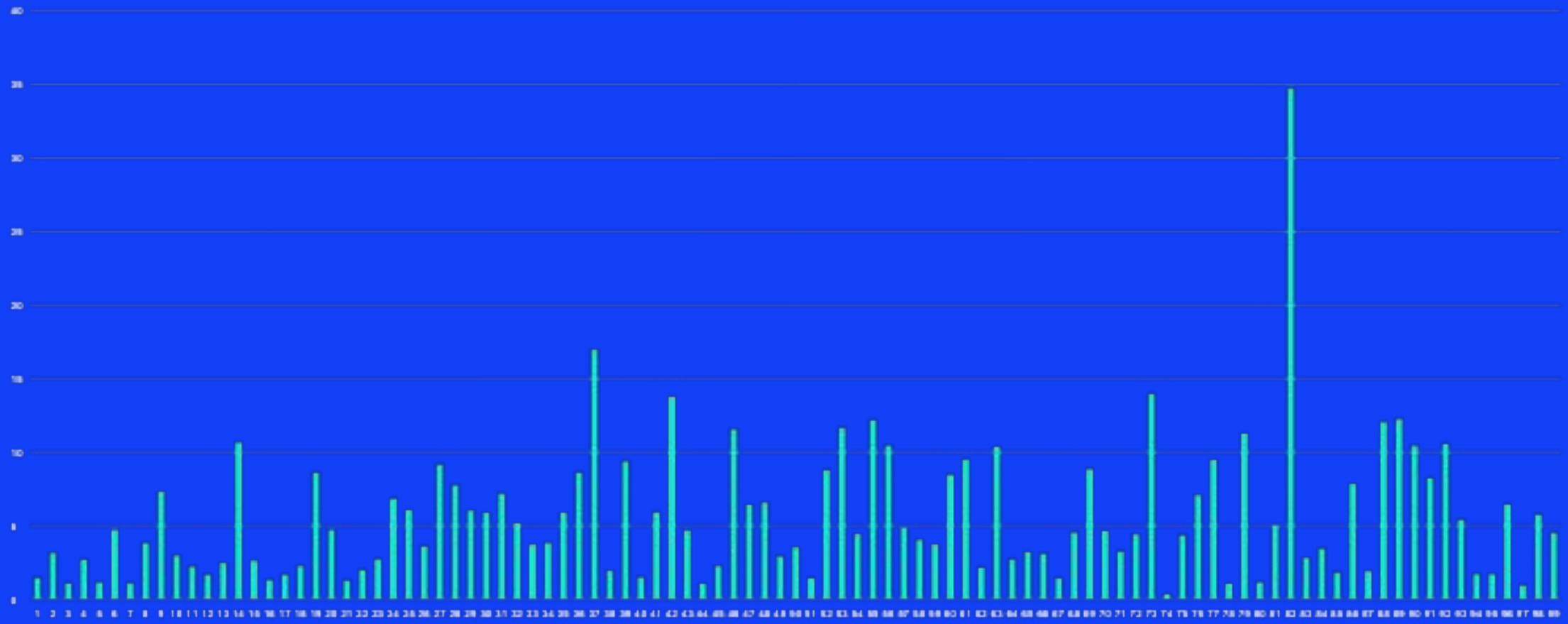
Some TPC-DS perf graphs ☺️

IBM blog

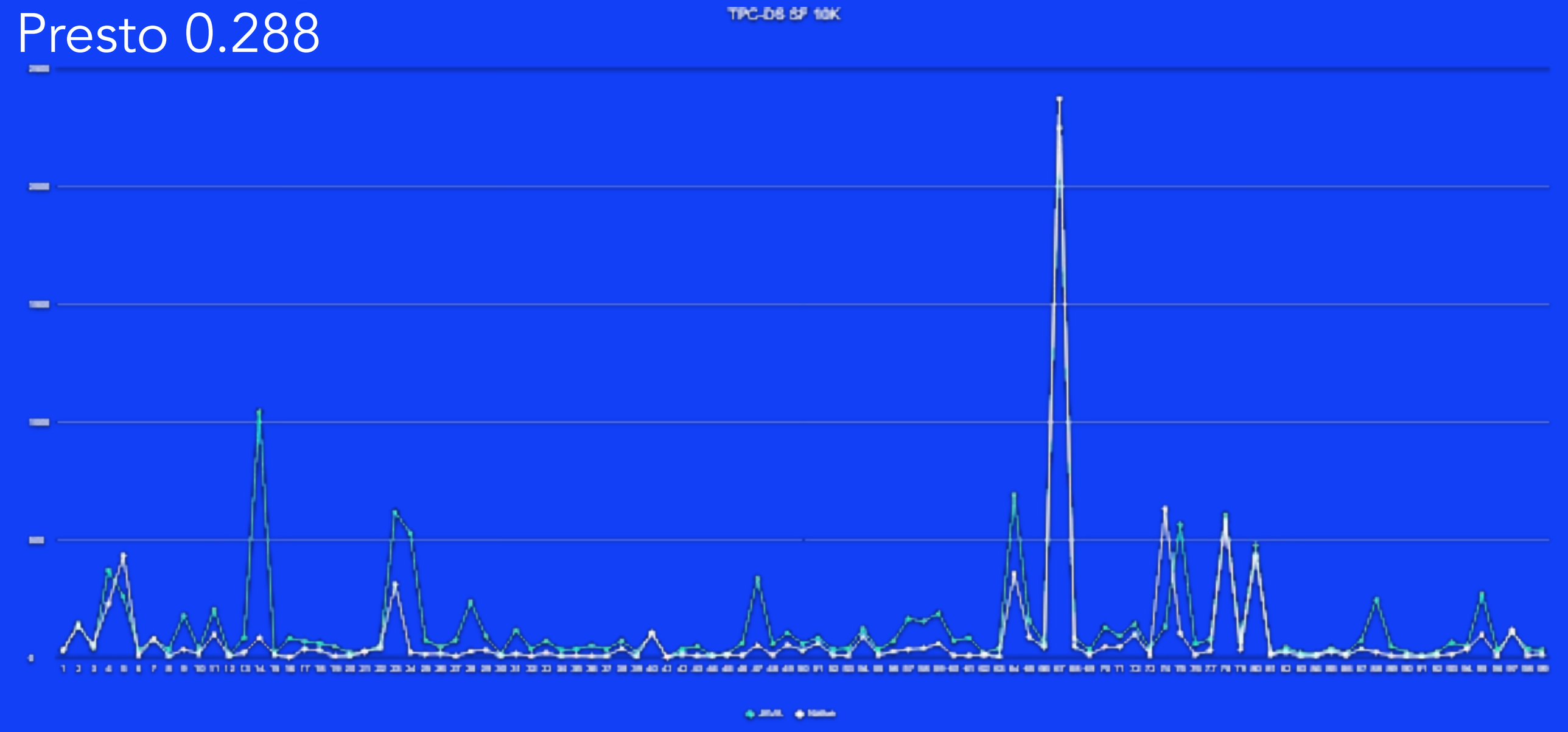
TPC-DS 1K run (Native : 31.2 min, Java : 1.25 hr)
8 r6i.4xlarge (vCPU: 16, Memory: 128GB)
Presto 0.288



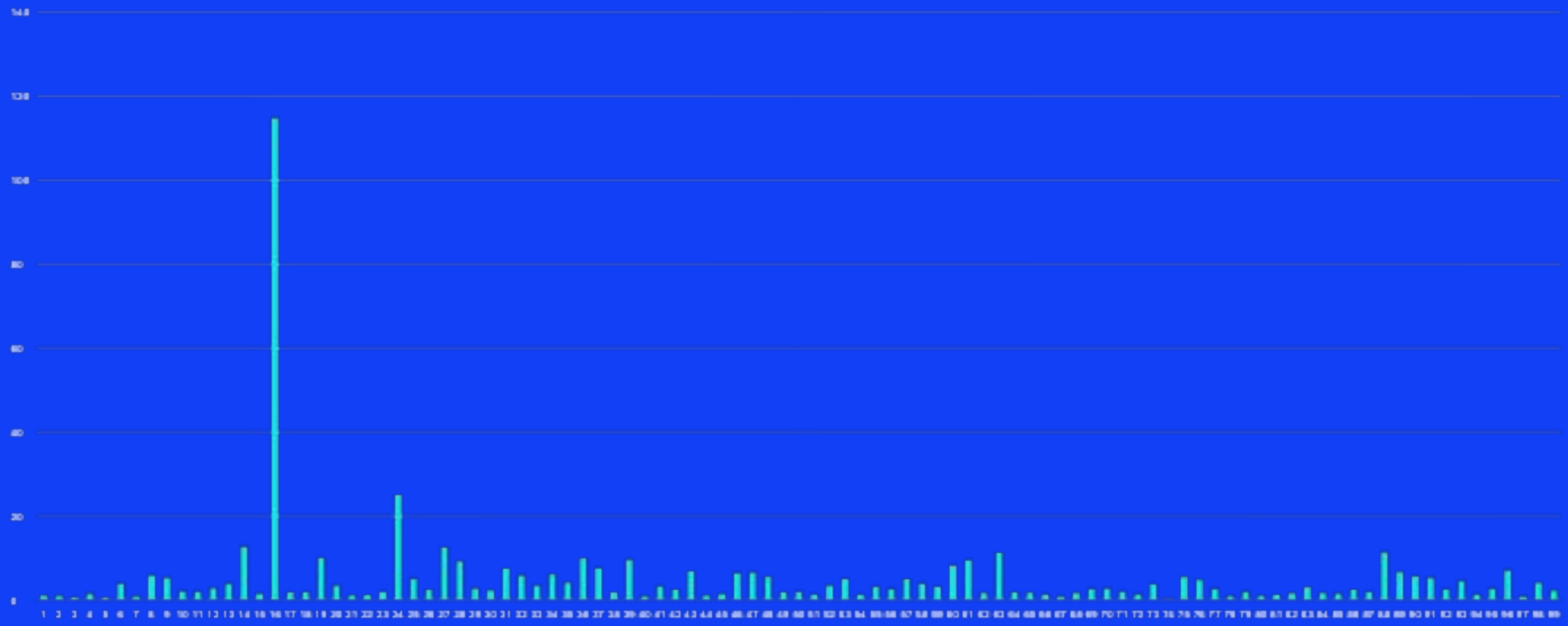
Duration ratio (Java/Native time) graph



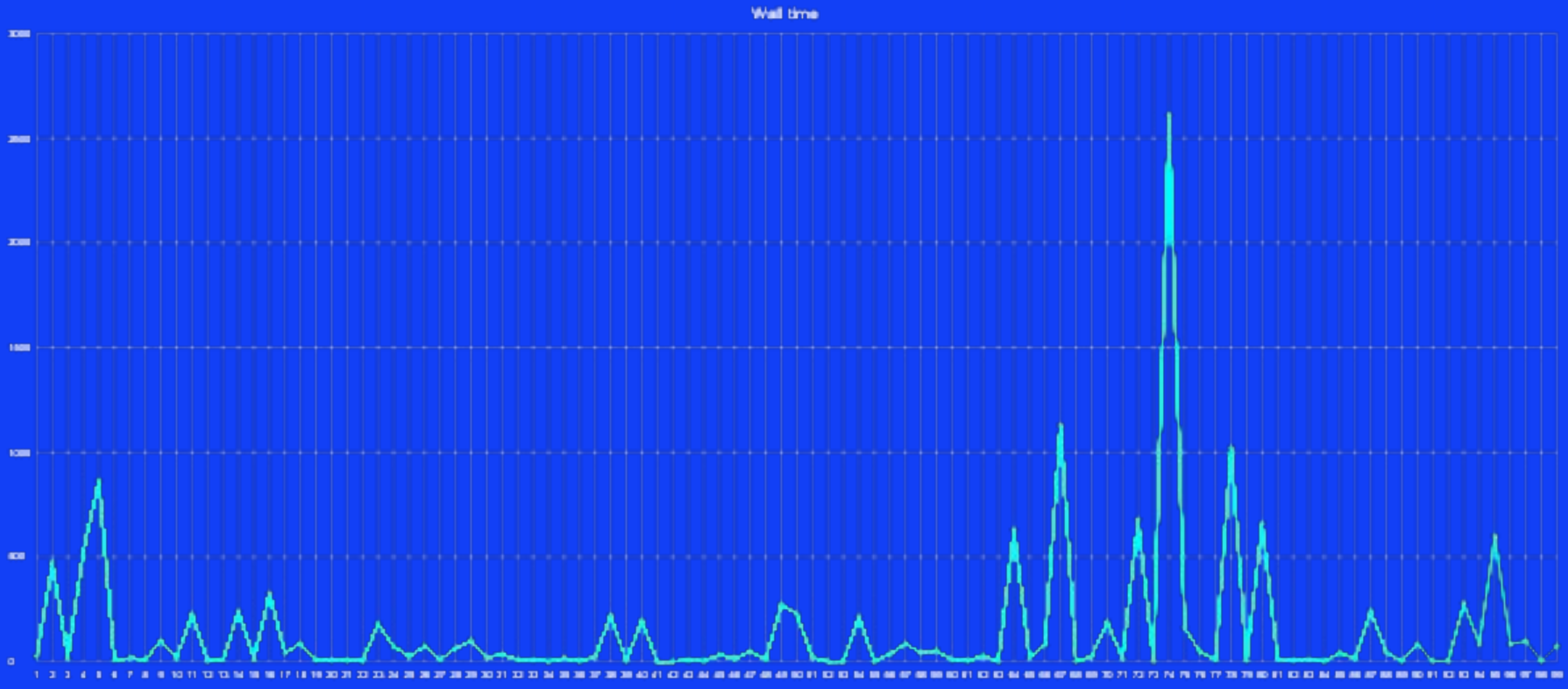
TPC-DS 10K run (Native: 2.19 h Java: 3.80 h)
16 r6i.8xlarge (vCPU 32, Memory 256 GB)
Presto 0.288



Duration ration (Java/Native time)



TPC-DS 100K run (Native: 3.95 hours)
48 r6i.16xlarge(vCPU: 64, Memory: 512 GB)
Presto 0.288



Native engine in Meta Production

Presto at Meta - Historical context

- 2015 - 2017 Hive Migration and External Analytics
- 2018 - 2019 Interactive and Batch, Linux foundation
- 2020 - 2021 Caching, Elastic Compute, Efficiency
- 2022 - Present Native execution

Presto batch workload at Meta

- Variety of workload using all Presto features
  - Internal Functions
  - Internal Connectors
- Workload wall time between 5 minutes to 1h
  - Periodic
  - Ad-hoc
  - Tiring
- Writers
- Exchange
- Large Batch Mode
- Faster Execution
- Limited Hardware and Customer expectation

Presto batch workload at Meta

- Evolving requirements
- Machine Learning
  - File formats
- SQL on Machine learning file formats
  - Periodic and Adhoc
  - Presto
- Same deployment for traditional and new formats

Presto batch workload at Meta

- Correctness
- Performance
- Release and Deployments
  - Changing the engines while flying the plane.

Presto batch workload at Meta - Correctness

- Decade worth of fixes
- Source of truth for all computing platforms
- Backward compatibility of results
- Verification Strategy
  - Wide variety of workload
  - Understand workload to verify correctness
    - Deterministic
    - Not deterministic
    - Templated patterns [Same query, different parameters]
    - Ad-hoc
    - Retryable - Non Retryable
  - Behavior changes
    - Json parsing
    - Regex (Joni vs Re2)
    - Consolidation
  - Verify fast to fix bugs

Presto batch workload at Meta - Performance

- Main driving factor for the complete re-write
- Define key metrics
  - Ratio = Sum of Presto CPU/Exec time / Sum of Prestissimo CPU/Exec time
  - Take timings for queries that are successful in both systems
  - Use this along with success rate by CPU and Count
    - Percentage and absolute values

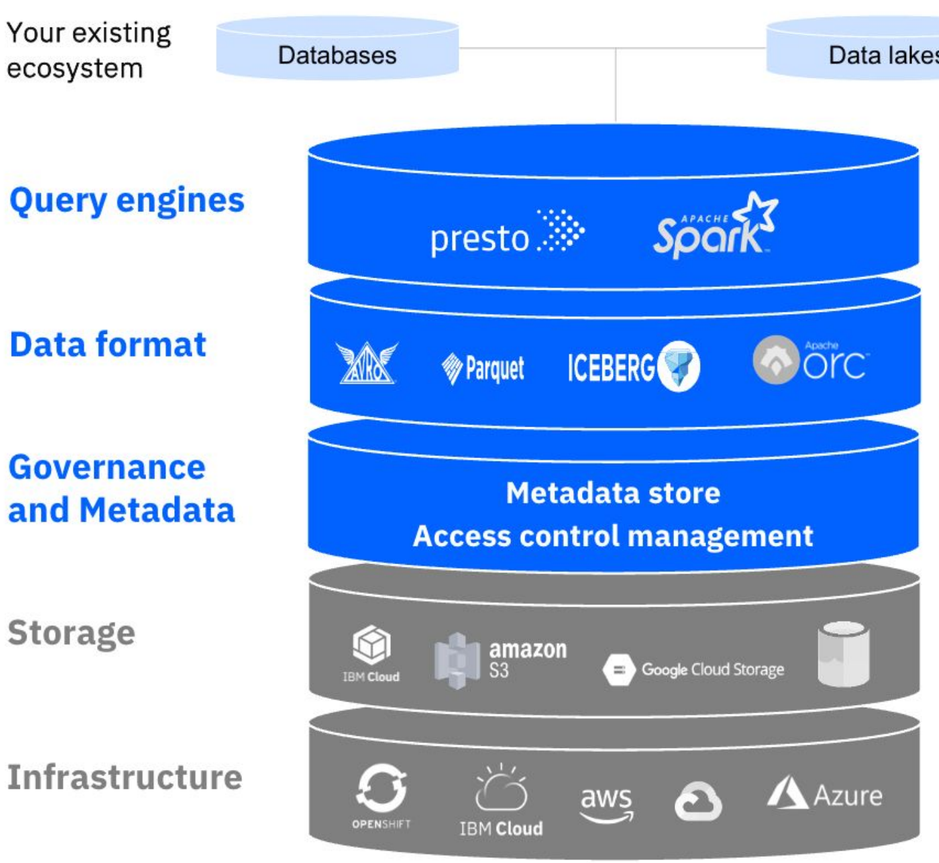
Presto batch workload at Meta - Performance

Presto batch workload at Meta - Highlights

- Reliable, maintainable and Fast execution
- Fantastic Customer experience
- ~3-4X better in CPU Time
- ~2X Execution Time
- 60% less hardware OR 3X throughput
- ~30% workload running in native engine

IBM watsonx.data

Overview of the key components of IBM watsonx.data (data lakehouse) : multiple query engines, open table formats and built-in enterprise governance



IBM watsonx.data

Multiple engines such as Presto and Spark that provide **fast, reliable, and efficient processing of big data** at scale

Vendor agnostic open formats for analytic data sets, allowing different engines to access and share the same data, at the same time

Built-in governance that is compatible with existing solutions, including IBM Knowledge Catalog

Cost effective, simple object storage available across hybrid-cloud and multi-cloud environments

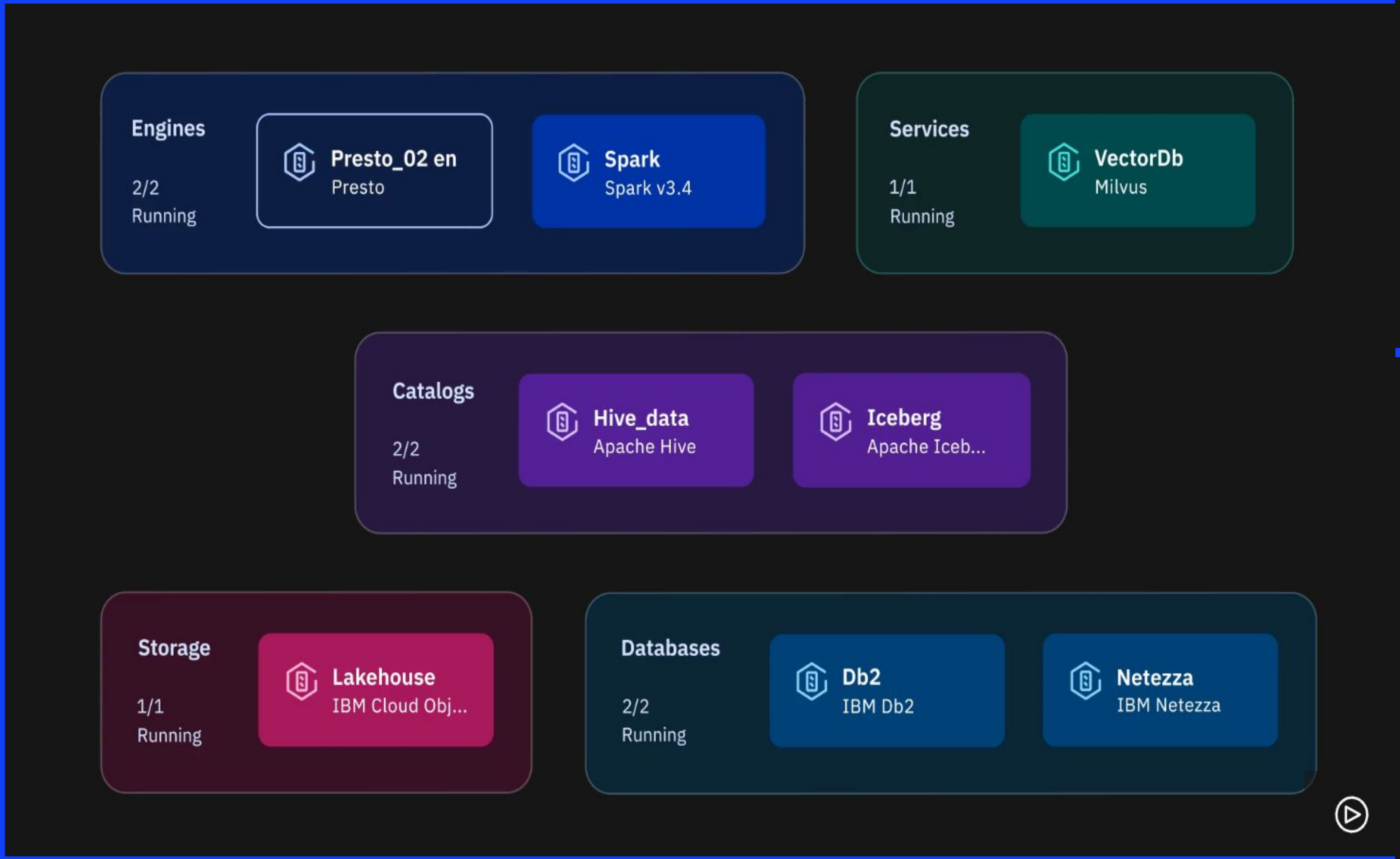
Hybrid-cloud deployments and workload portability across hyperscalers and on-prem with Red Hat OpenShift

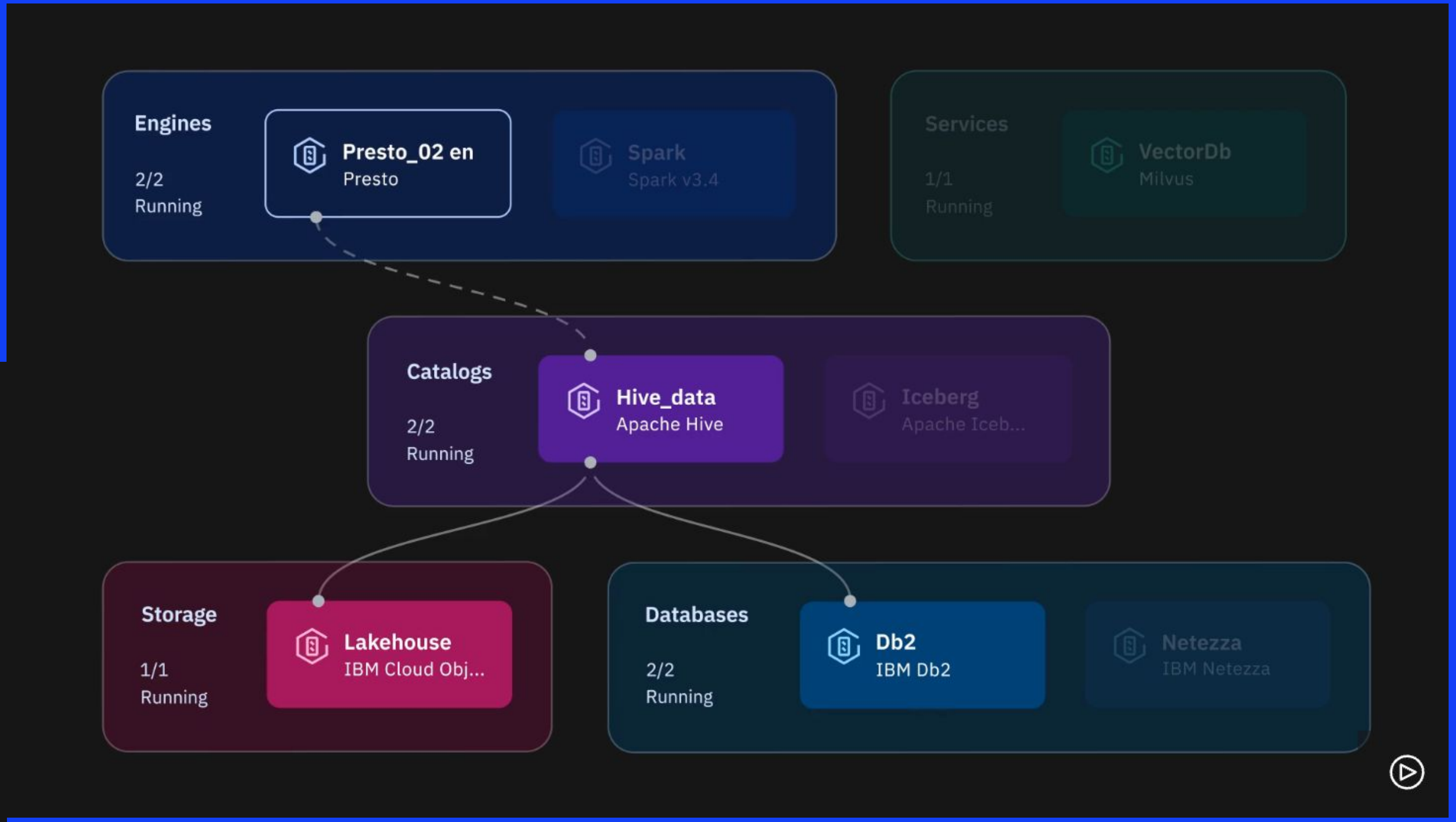
Core lakehouse functionality

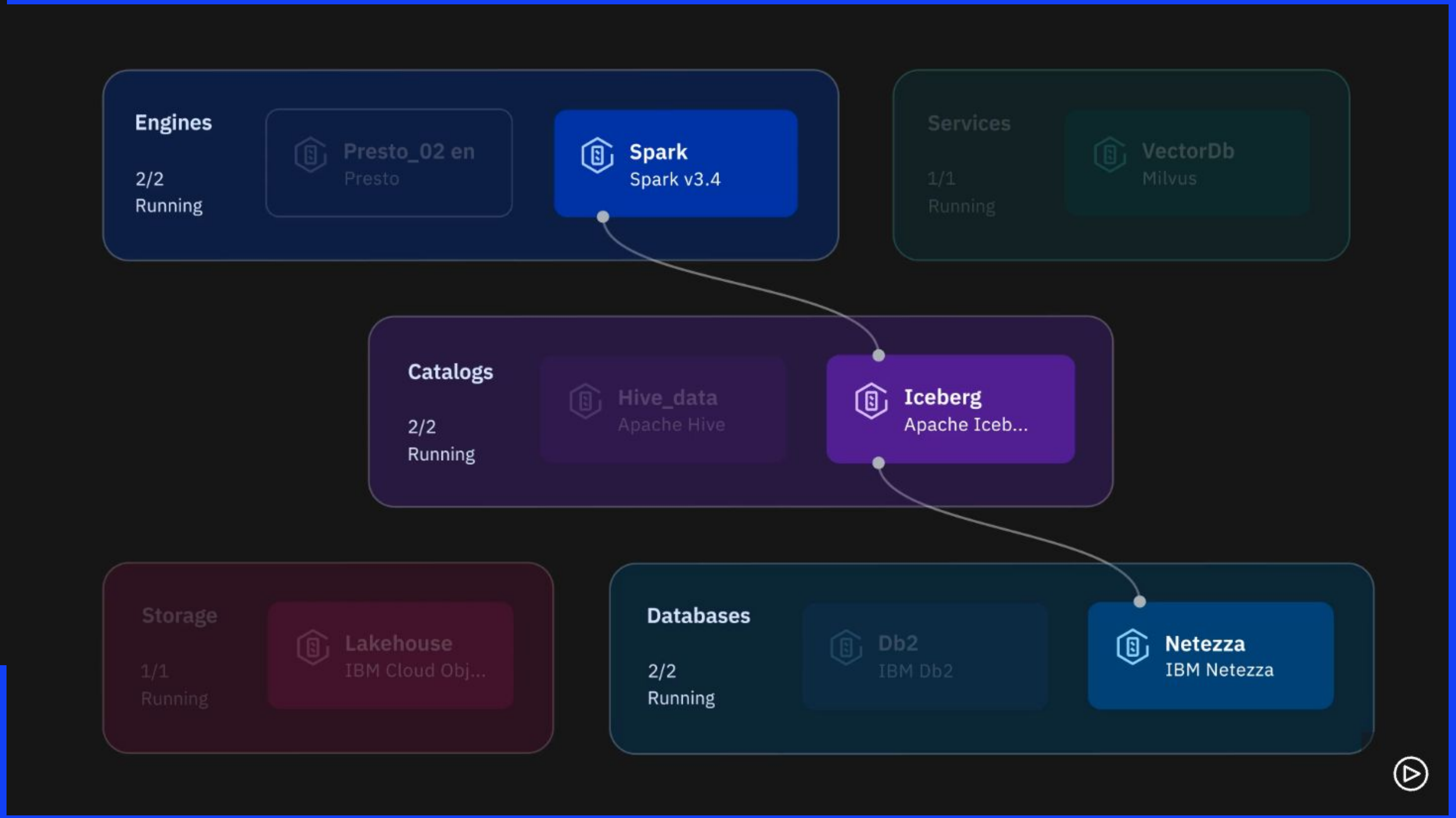
Ecosystem infrastructure

- **Optimize workload costs and performance using multi-engine functionality**
- **Access 100% of your data across databases and data lakes**
- **Ensure governance and reduce time to insight with centralized metadata and access management**
- **Reduce storage costs and facilitate data ingest**
- **Deploy on any infrastructure and optimize available resources**

watsonx.data Open Data Lakehouse







Project roadmap

Current status

- SQL Coverage : Some gaps remain
  - Types: char(n) type
  - Functions : PrestoSQL Coverage map
  - Select SQL complete. Write/Delete paths not tested
- Lakehouse : Parquet/Iceberg
  - Reader : Parquet/Iceberg
  - Writer : Parquet
- Connectors : Hive (for Iceberg as well), TPC-H/TPC-DS, Arrow Flight

Future roadmap

- Presto native side-car/SPI v2
  - Prestissimo has a split brain problem
  - Co-ordinator plans based on a catalog populated by the Java worker. Does not reflect the Native engine functions. Or session properties
  - Native side car aids the co-ordinator in planning for C++ worker
- Lakehouse connectors : Delta, Hudi
- Stability, stability and stability.
  - Fuzzers for correctness testing.
  - Performance.
  - Production hardening.

Q&A

Presto Native Worker Working Group

Prestissimo : GitHub, Slack, LinkedIn, Meetup

Velox : GitHub, Slack, Website