"Using Dockers for Online TPC Data handling"

A.KRYLOV

MPD SOFTWARE GROUP

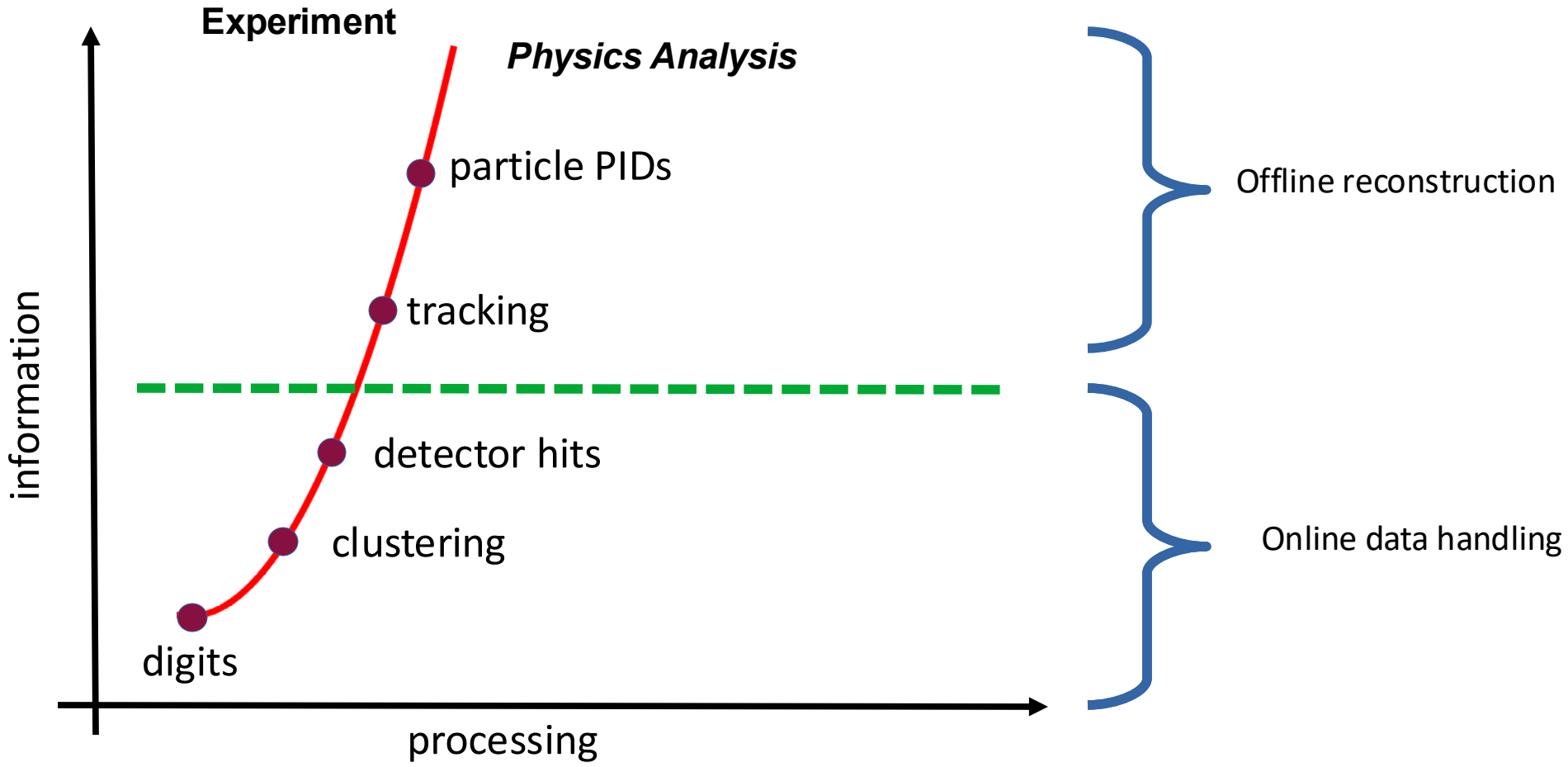
JINR/LHEP

Email: avkrylov@jinr.ru

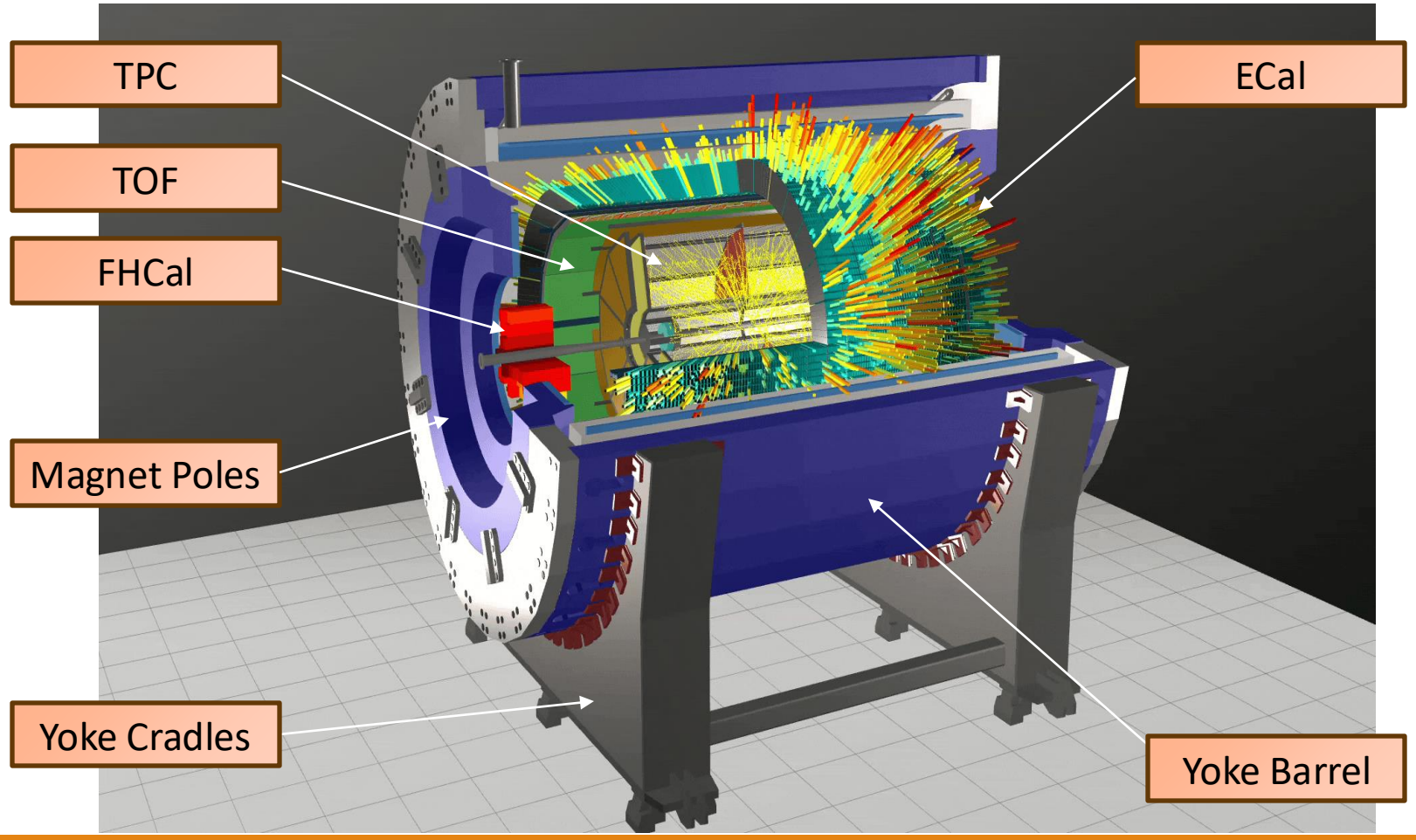
Outline

Experiment data chain
MPD Event Display
Online Tpc Clustering
Microservice software design pattern
Docker technology
ZeroMQ messaging library
QA histograms task

Experiment data chain



1-st stage MPD Detectors



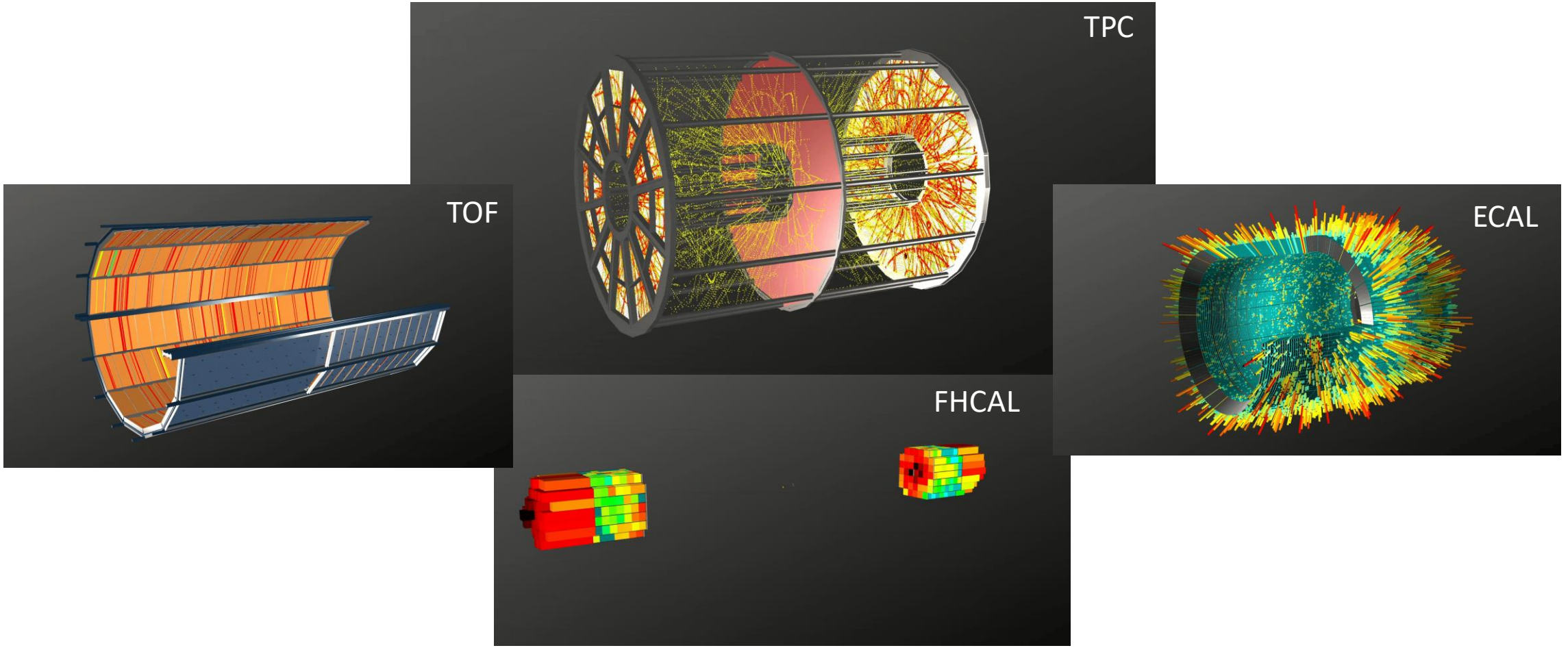
MPD Detectors:

- **TPC** - Time Projector Chamber
- TOF - Time of Flight
- ECAL - Electromagnetic Calorimeter
- FFD – Fast Forward Detector
- FHCAL - Forward Hadron Calorimeter

TPC is the main tracking detector of the MPD central barrel. It is a well-known detector for 3-dimensional tracking and particle identification for high multiplicity events.

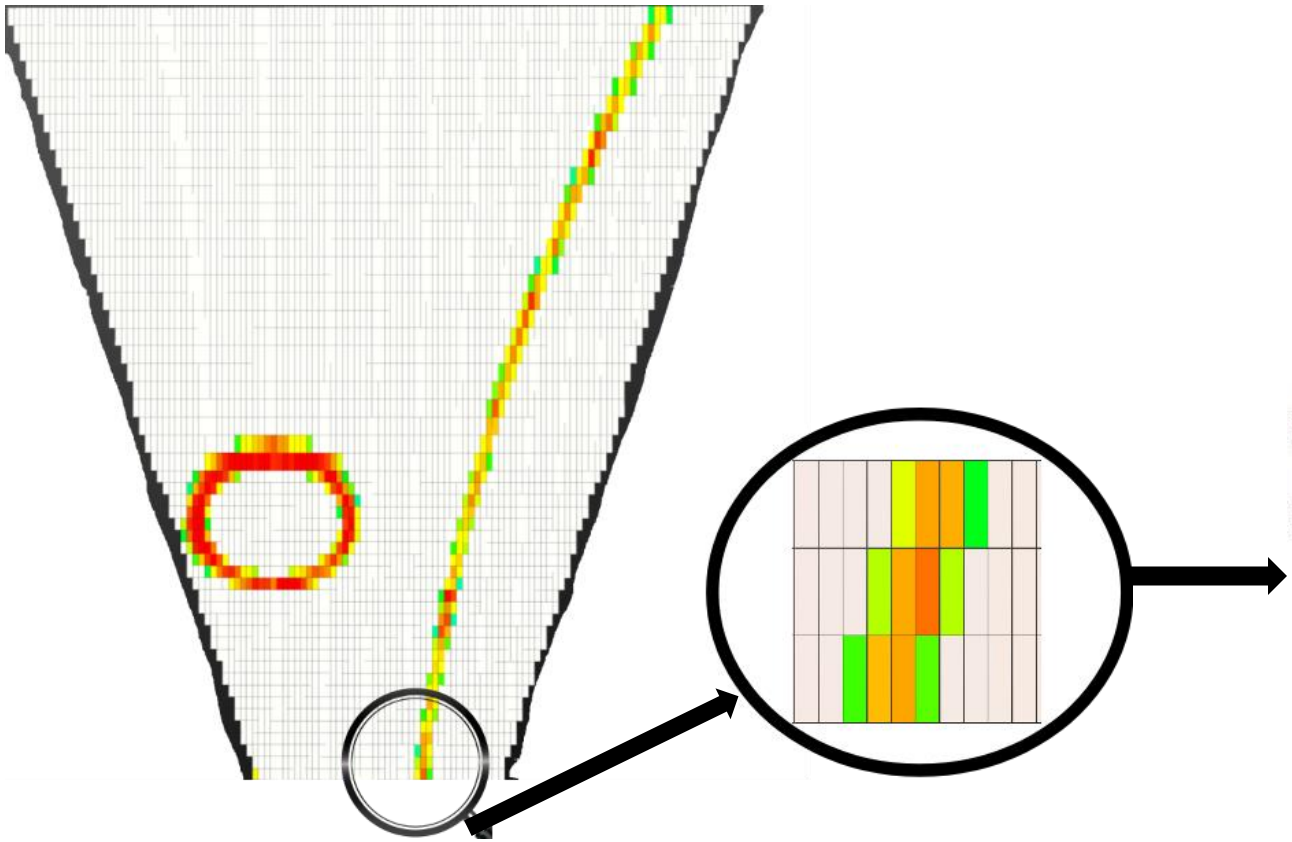
Web address of a prototype of the event display: https://mpd-edsrv.jinr.ru/

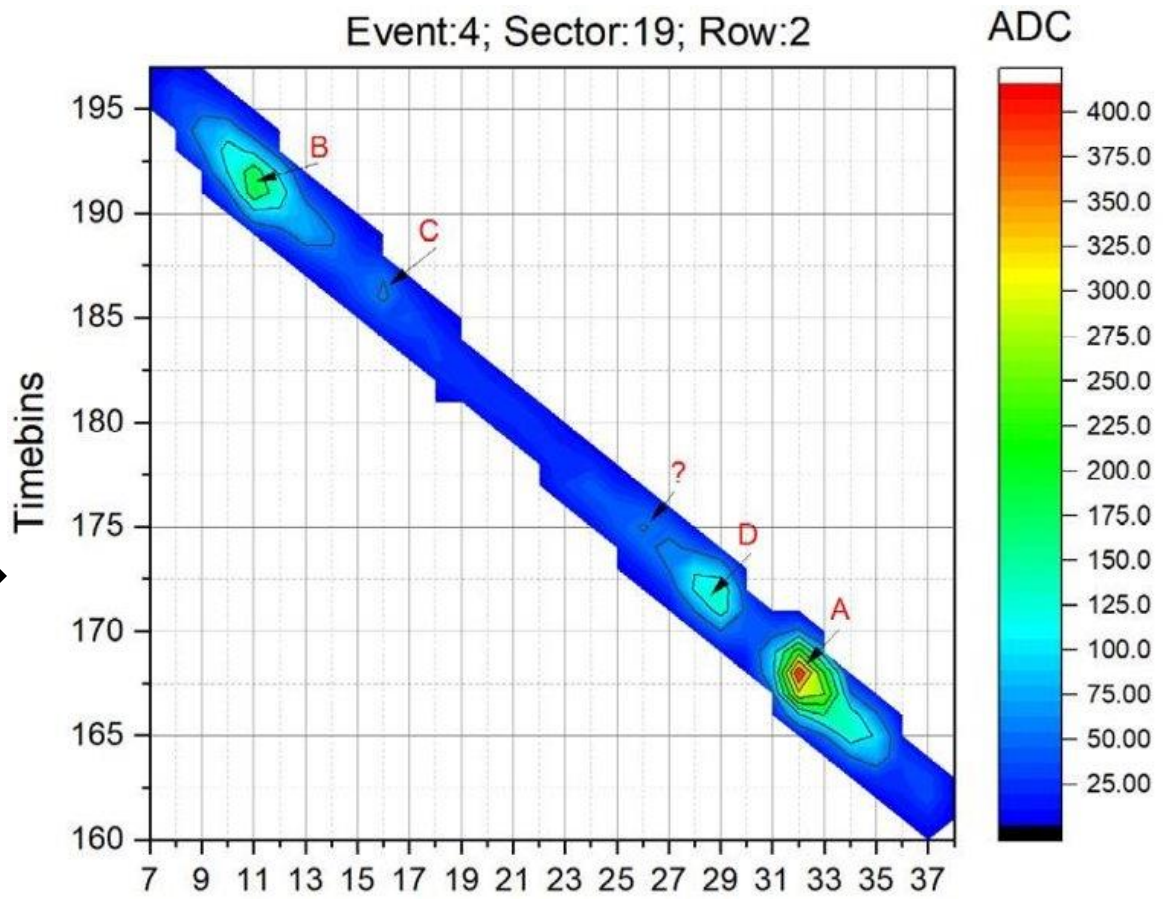
MPD Detectors response



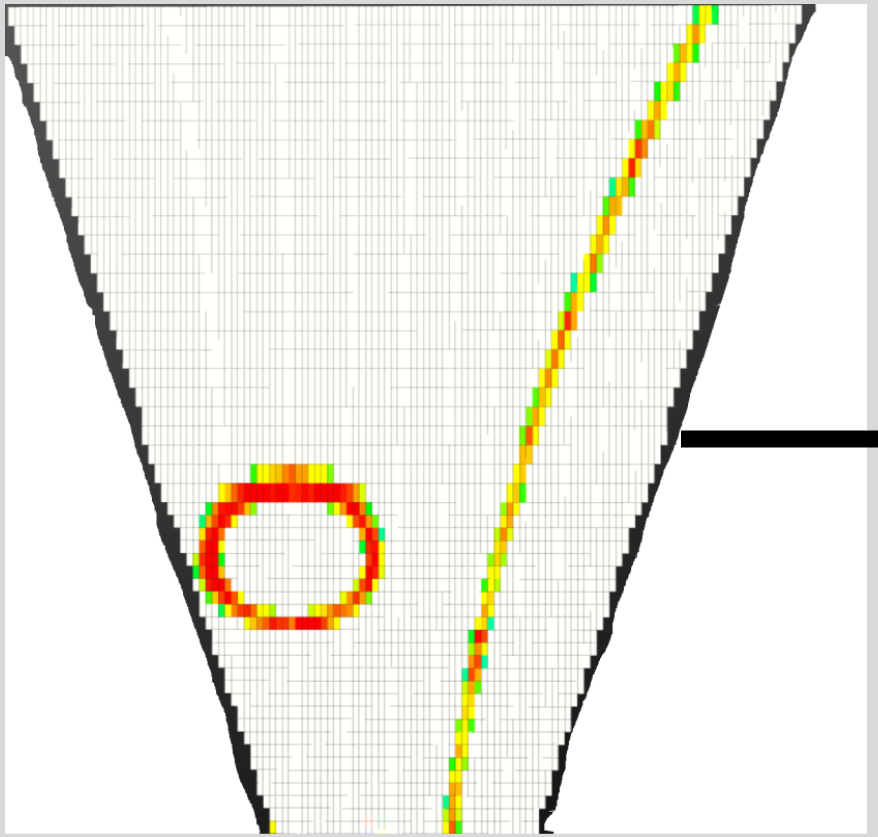
Web address of a prototype of the event display: https://mpd-edsrv.jinr.ru/

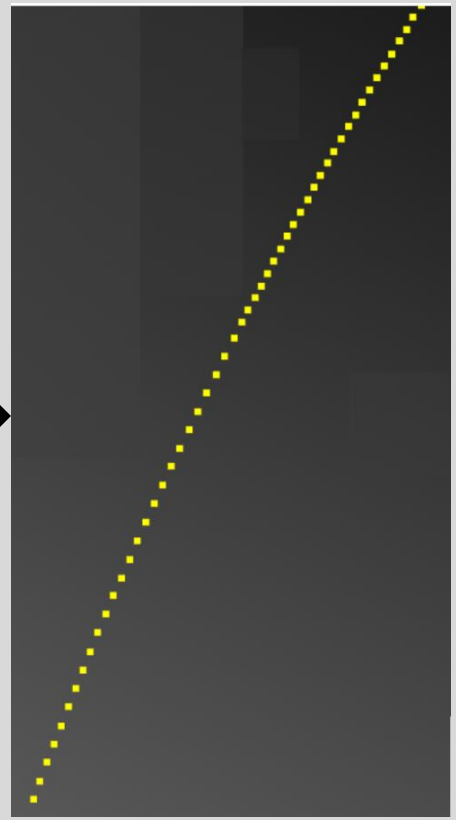
Online TPC Clustering

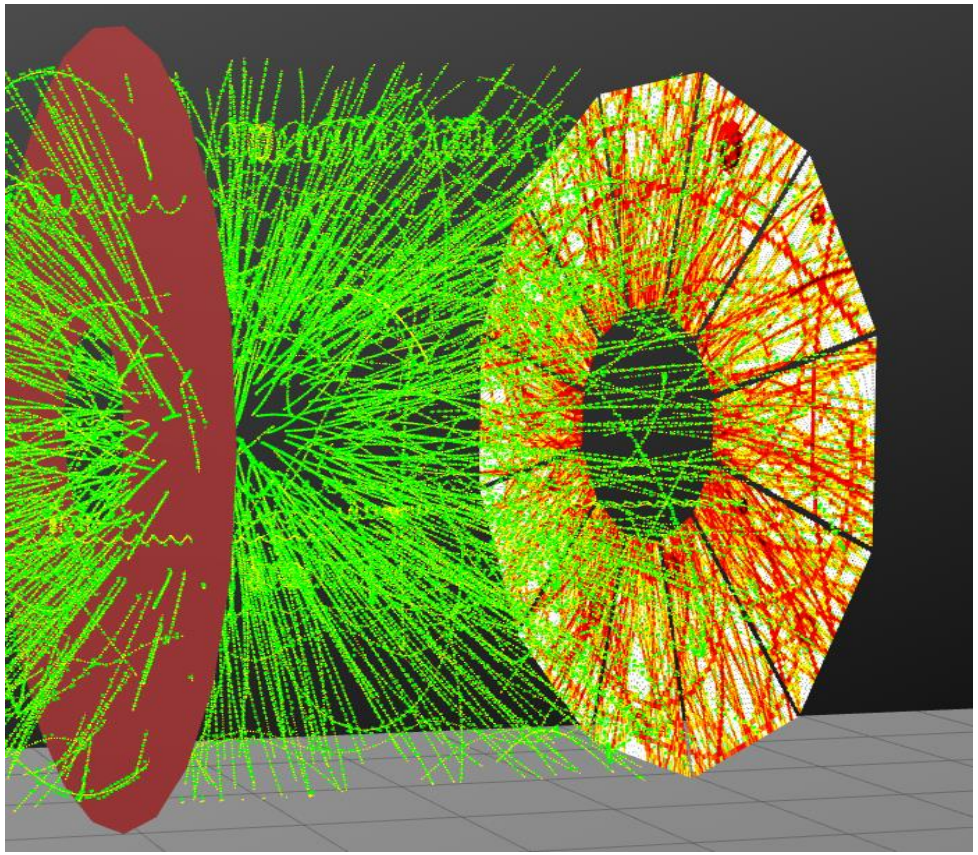




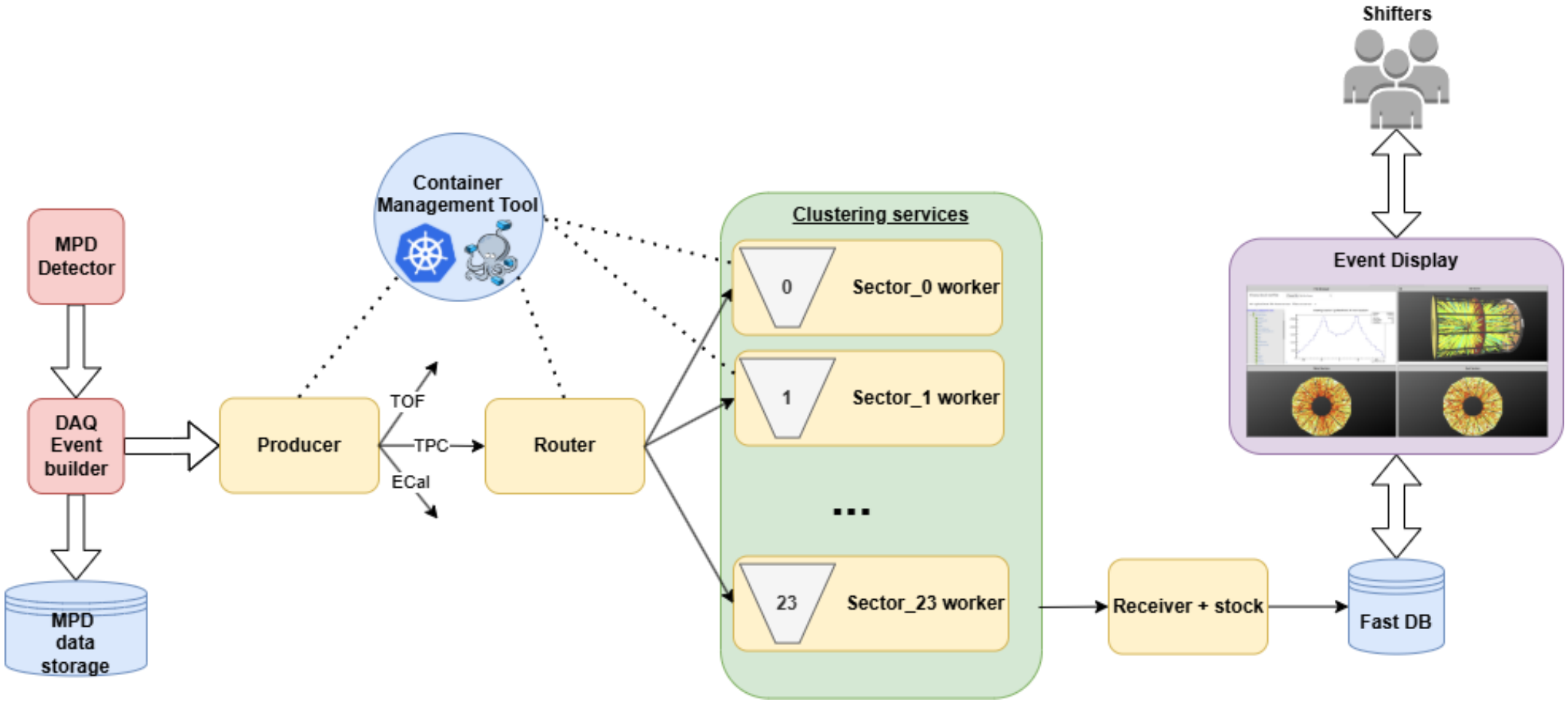
Online TPC Clustering





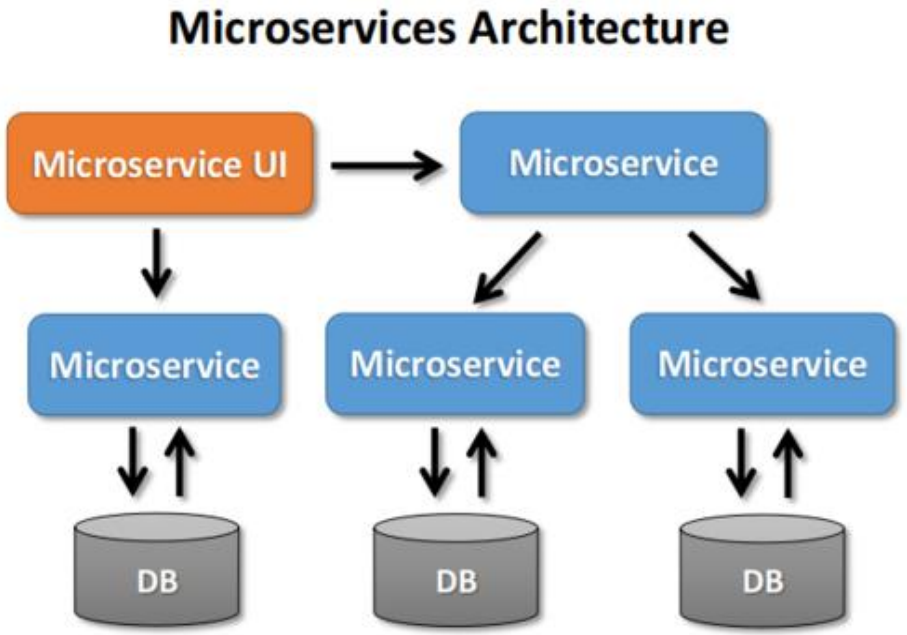


TPC Event Processing



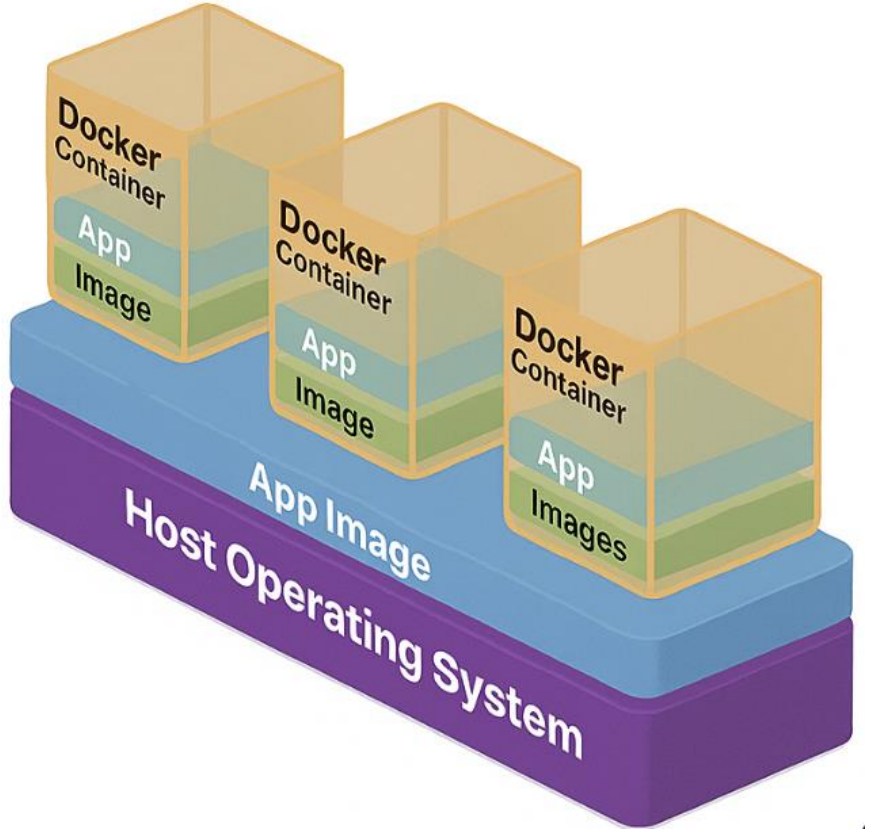
Microservices architecture

- **Microservices** - is an architectural pattern that organizes an application into a collection of loosely coupled, fine-grained services.
- **Microservice** is a software design pattern, while **Docker** is a tool for implementing and managing containerized applications.
- In contrast to the traditional monolithic approach of a large, tightly coupled application, **microservices** provide a **cloud-native** architectural framework.



Docker Ecosystem and Containerization

- **Docker** is open-source platform that enables developers to build, deploy, run, update and manage containers.
- Each docker is like a **separate Linux system**, so you have a complete operating system running within a single machine.
- Dockers can be easily replicated and orchestrated, enabling rapid scaling of the applications in cloud and cluster environments.



Examples of CERN's use Docker in experiments

- **CERN:** At CERN we provide base Docker images for the supported Linux distributions. They can be found in both CERN's Docker registry or on https://hub.docker.com/u/cern. These images can be used as the base for your own images, by extending its content with all your requirements.
- **CMS:** The CMS experiment provides guides and resources for running CMS analysis code using Docker, including mounting local file areas for data sharing. (https://opendata.cern.ch/docs/cms-guide-docker)
- **ATLAS:** The ATLAS utilizes Docker for distributing its analysis software (Athena) and provides base images for analyses, published on CERN's GitLab container registry. (https://atlas-software.docs.cern.ch/athena/containers/intro-docker)
- **DELPHI:** The DELPHI experiment uses Docker containers to run its legacy software stack, including CERNLIB applications. (https://opendata.cern.ch/docs/delphi-guide-docker)
- **ALICE**: CERN's ALICE experiment utilizes Docker containers to facilitate the execution of Grid jobs and provide a consistent software environment across various computing sites. (https://github.com/aphecetche/docker-alice-online)
- and more…

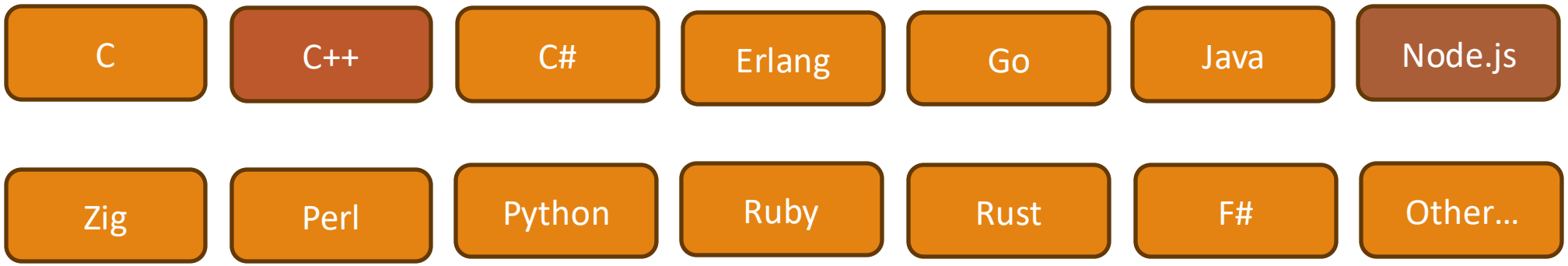
ØMQ
ZeroMQ

ZeroMQ

ZeroMQ (also spelled **ØMQ** or **ZMQ**) is a high-performance asynchronous messaging library, aimed at use in distributed or concurrent applications.

ZeroMQ supports common messaging patterns (*pub/sub, request/reply, client/server and others*) over a variety of transports (*tcp, ipc, in-process, multicast, websocket and more…*)

The philosophy of **ZeroMQ** starts with the zero. The zero is for zero broker (**ZeroMQ** is brokerless), zero latency, zero cost (it's free), and zero administration.



ZeroMQ TPC Messaging Topology

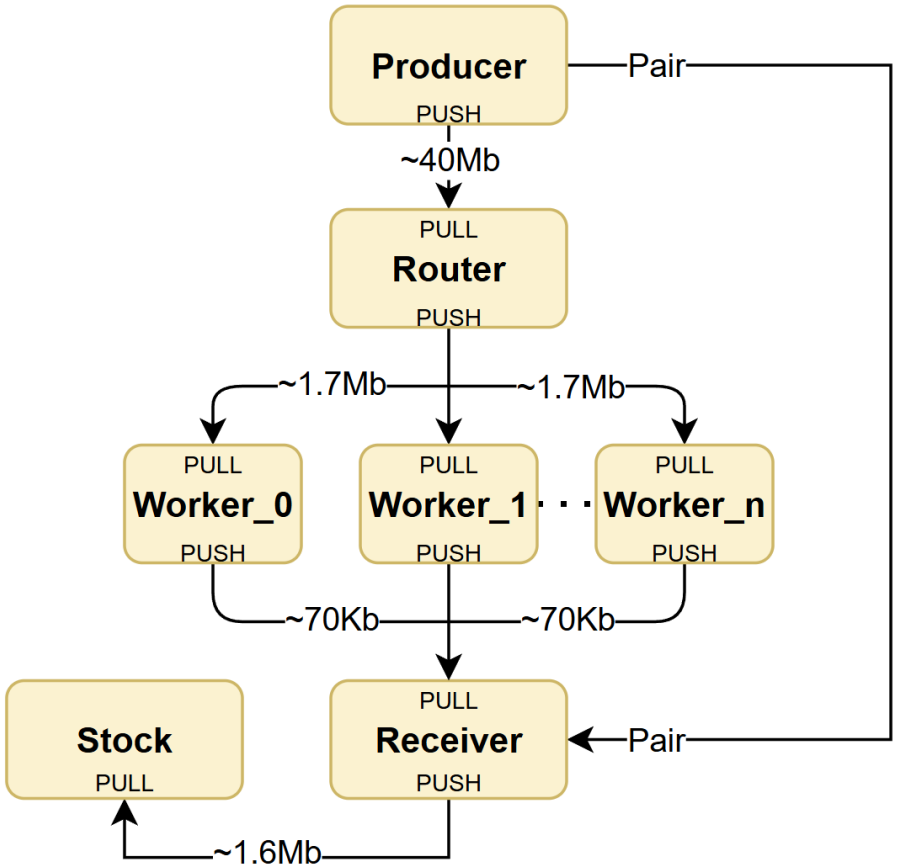
❑**Producer** - extracts MPD detector data from the DAQ event builder stream and send TPC detector data to router.

❑**Router** - distributes TPC detector data packages to a worker by sectors.

❑**Worker** - perform clustering and processing of TPC detector responses.

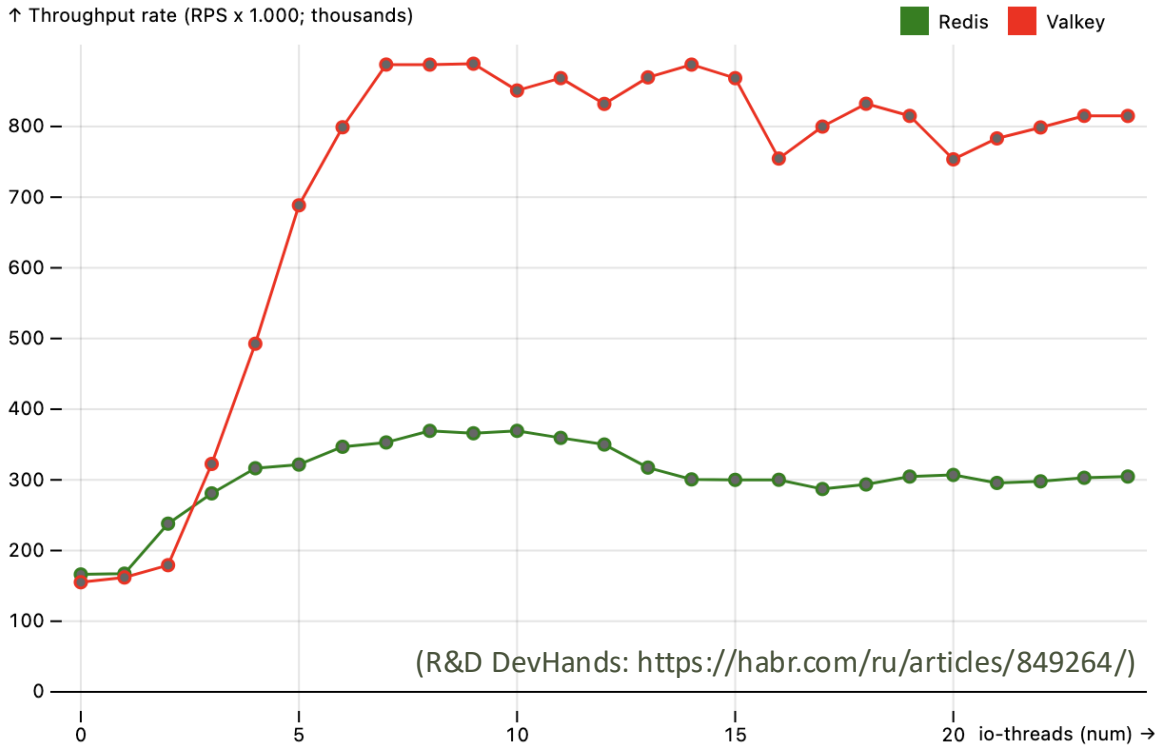
❑**Receiver** - collects all worker outputs and builds QA histograms per event.

❑**Stock** - accumulates QA histograms across events and updates Valkey DB with current status.



Valkey in memory database

Valkey is an open source (BSD-3 Clause License) high-performance key-value datastore that supports a variety of workloads such as caching, message queues, and can act as a primary database.



In-memory structure stores all the data served from memory and uses RAM for storage. They offer a unique data model and high performance that supports various data structures like string, list, sets, hash, which it uses as a database cache or message broker. It is also called like **Data Structure Server**.

Management tool platforms

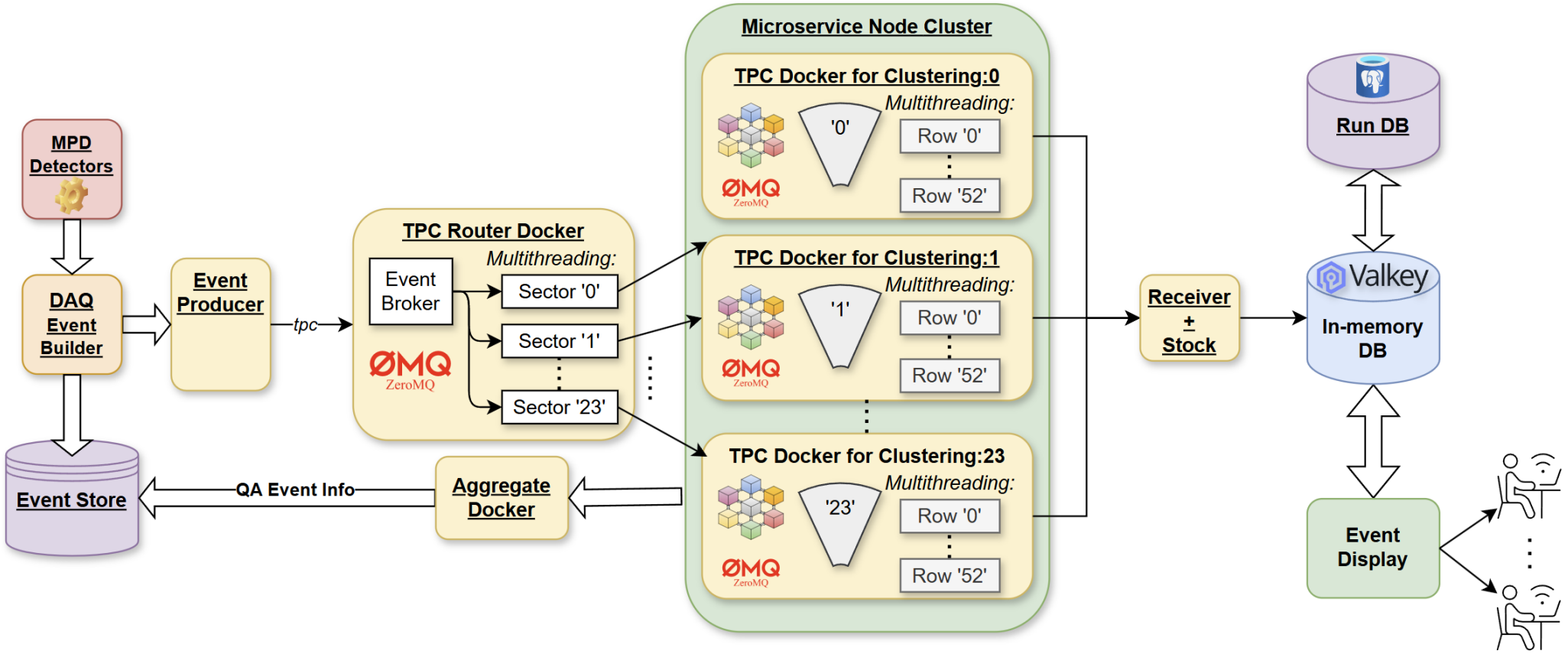
Docker Compose is a simple tool for defining and running multi-container applications. It is the key to unlocking a streamlined and efficient development and deployment experience.

Kubernetes, also known as **K8s**, is the most popular container orchestration platform, providing automation for deployment, scaling, and managing complex, multi-container workloads.

Apache Mesos is an open-source platform that allows organizations to manage and orchestrate distributed systems and applications. It's a powerful microservice orchestration tool that can deploy and manage complex microservice architectures in a highly efficient and scalable way.

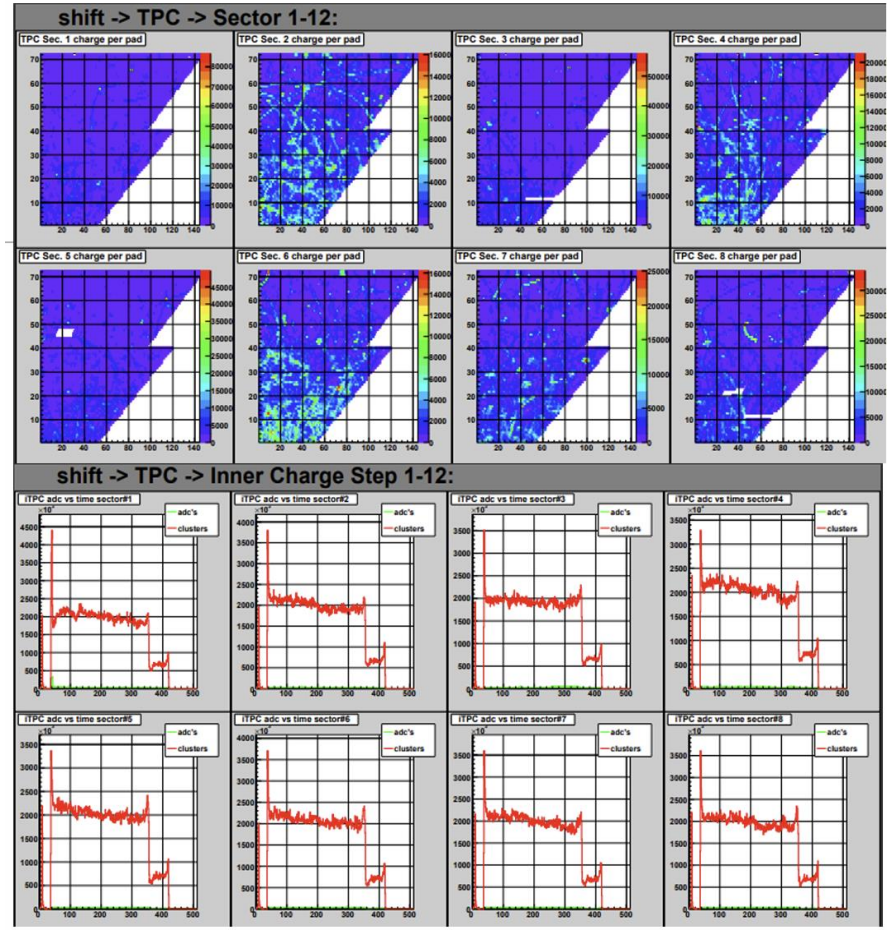
Netflix Conductor is an open-source microservice orchestration engine that provides a workflow and coordination service for microservices-based applications.

TPC Event Processing

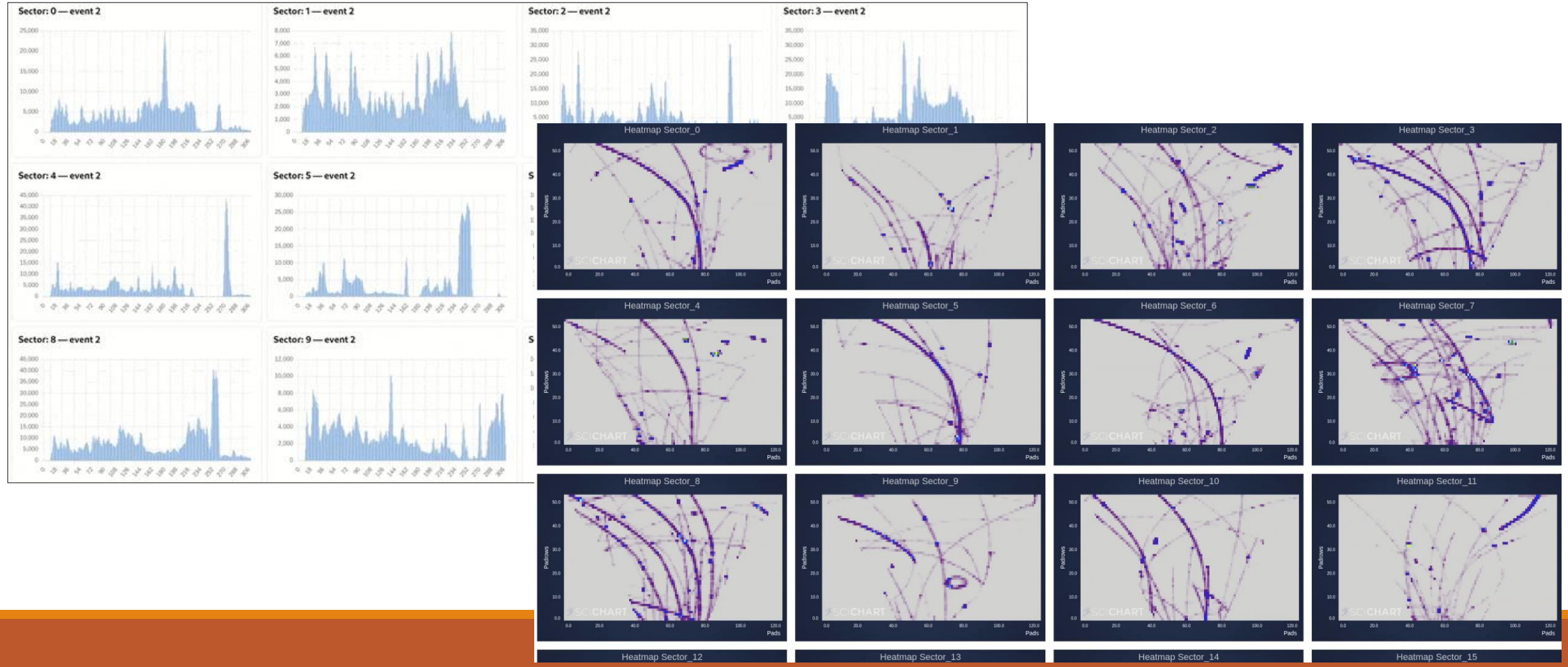


Quality assurance histograms for TPC

- ❑Inner pads ADC per sector – 24 histograms
- ❑Outer pads ADC per sector – 24 histograms
- ❑Inner pads ADC per timebucket – 24 histograms
- ❑Outer pads ADC per timebucket – 24 histograms
- ❑Inner pads ADC for current event – 24 histograms
- ❑Outer pads ADC for current event – 24 histograms
- ❑General clusters information – 6 histograms
- ❑**Total number of histograms > 150**



Real time QA histograms demo



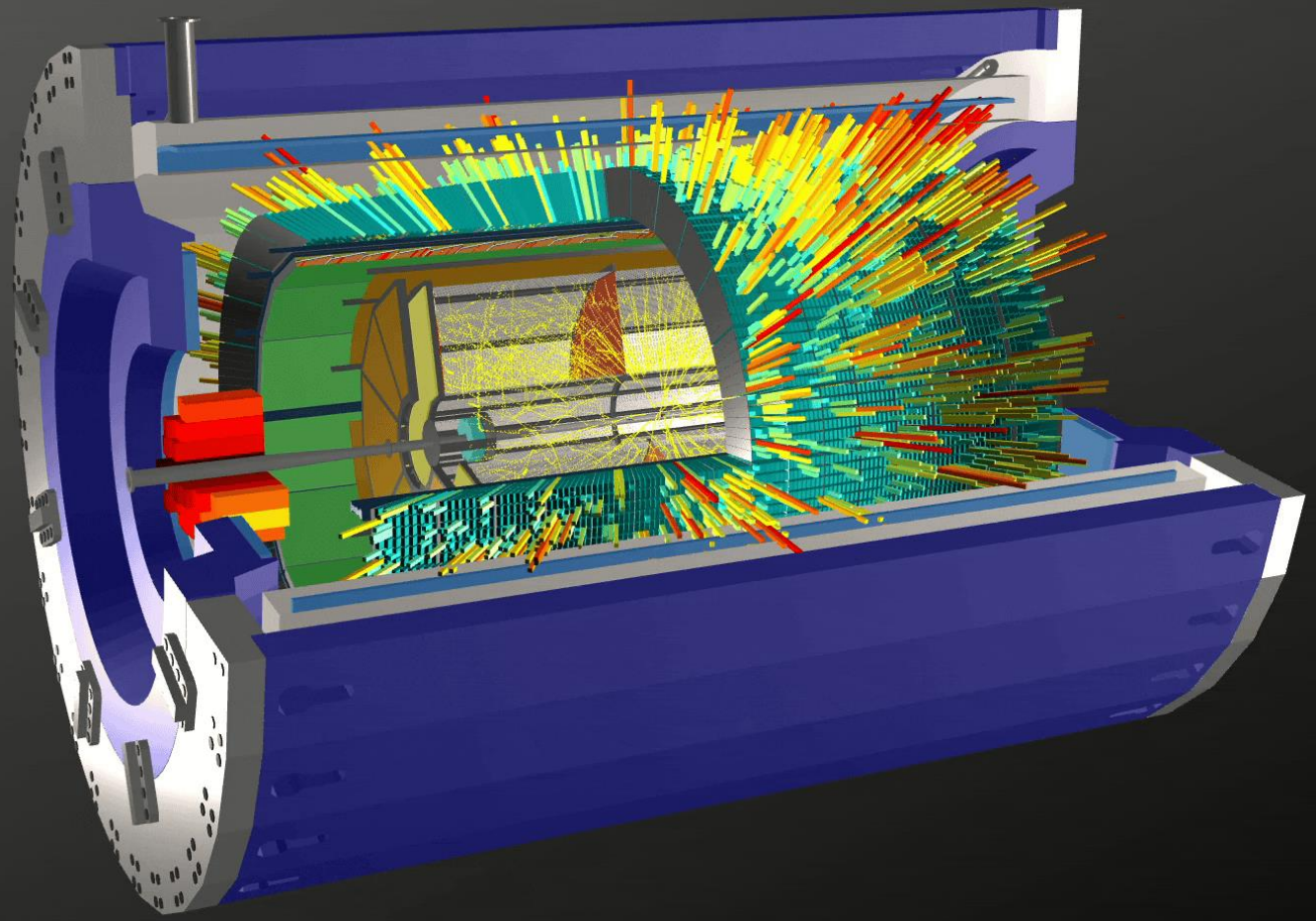
Future plans

1. Define Project Architecture
2. Explore and Choose Technologies (ZeroMQ, Valkey, Docker Compose)
3. Prototype the algorithms and containerize all components
4. Testing on QA histograms task
5. Testing on DAQ cluster (using multiple nodes)
6. Add logging and monitoring
7. Testing on real data

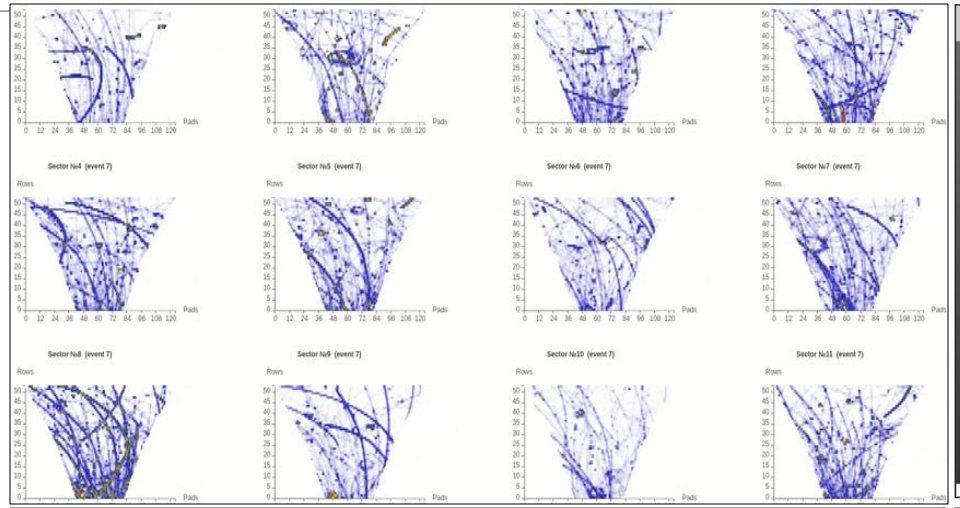
Summary

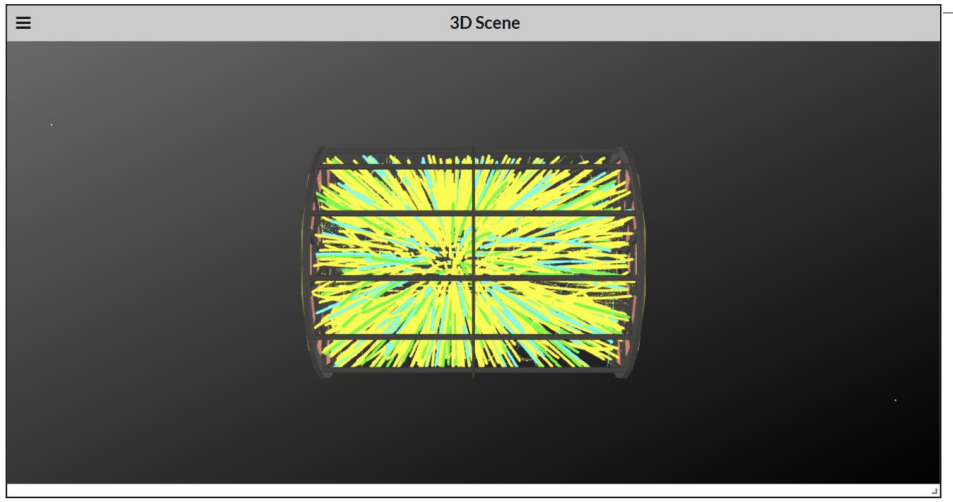
- Using distributed system (microservice architecture) is essential for such large experiment as MPD with a dataflow more than 1**00 Gb/s**.
- Technologies such as the **ZMQ** and **Docker Compose** make it possible to create a fault-tolerant distributed data processing system ready for further scaling.
- Current version of the data handling system processing is currently being tested on the QA task for TPC detector.
- The approximate event processing time will be less than 30 ms, which allows for the further use of this system in online mode during the experiment.

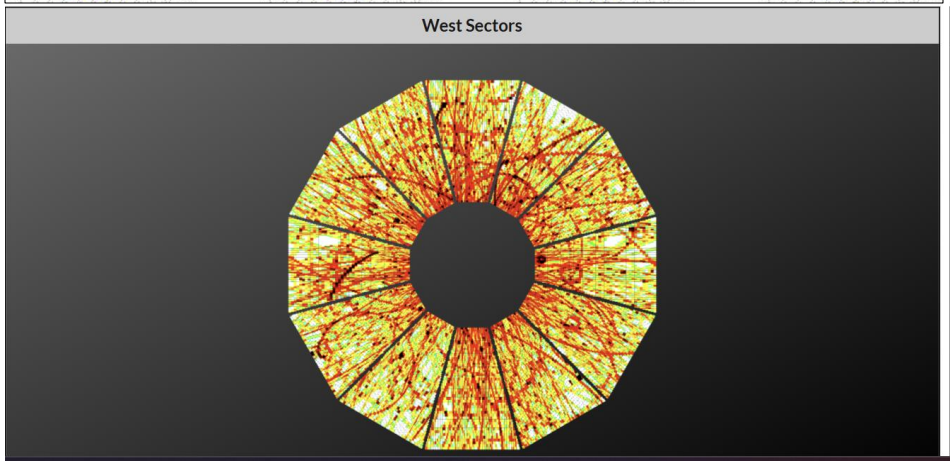
Thank you for your attention

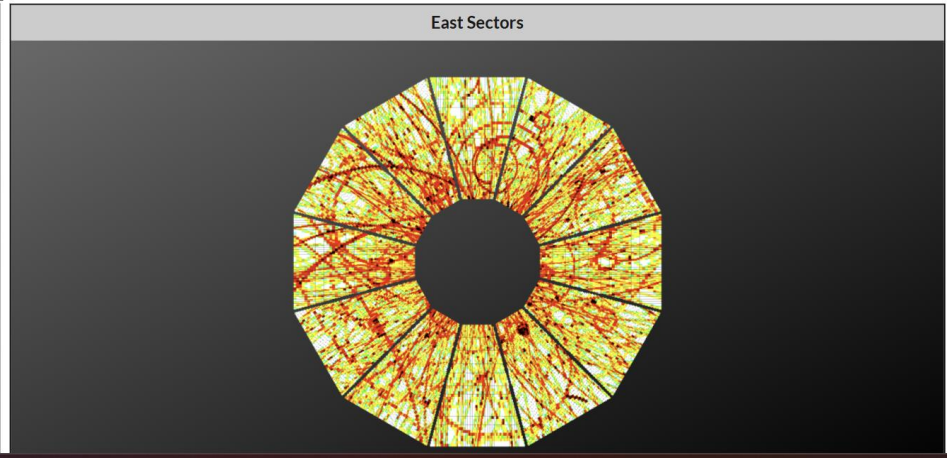


Event Display visualization

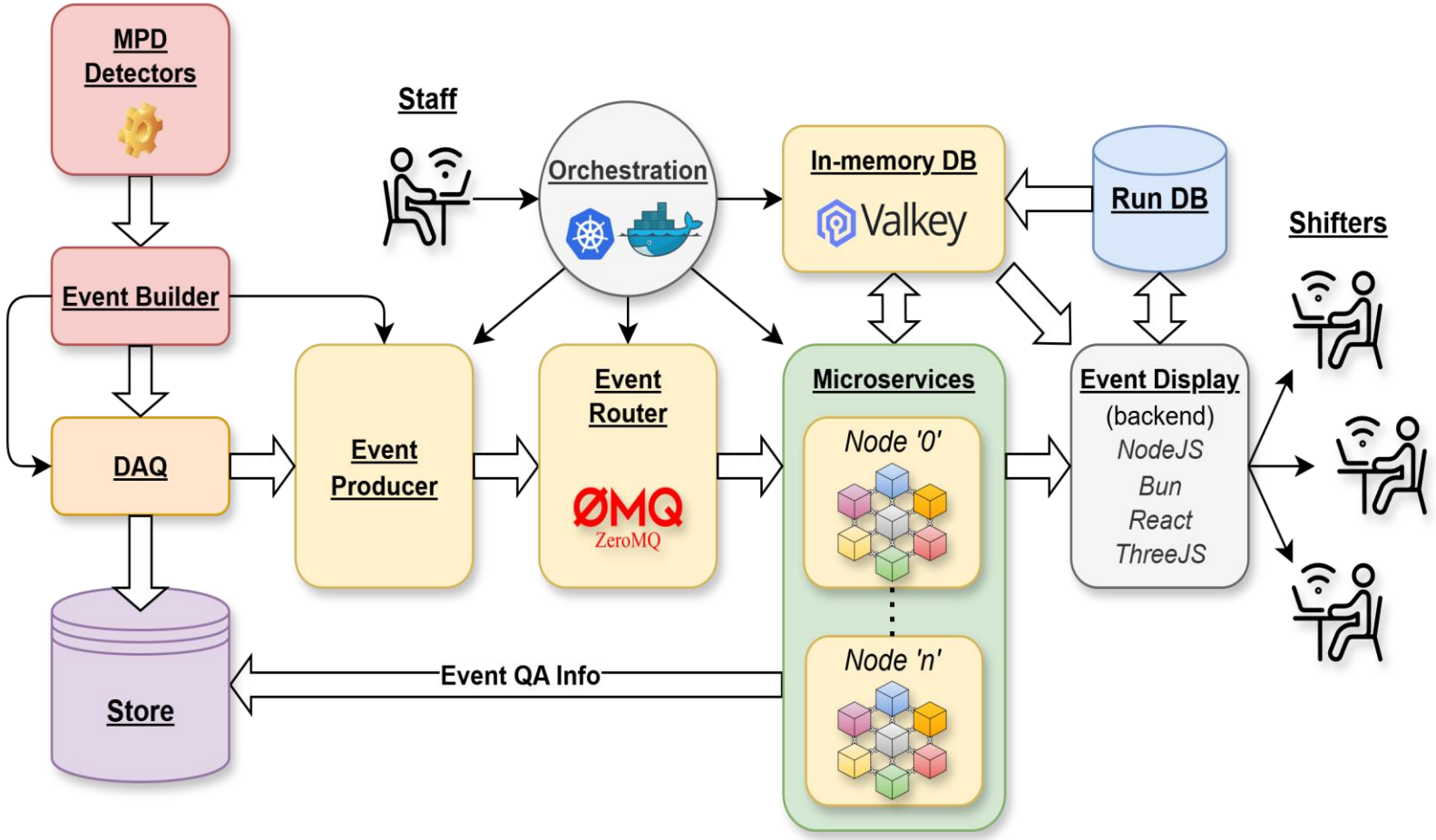








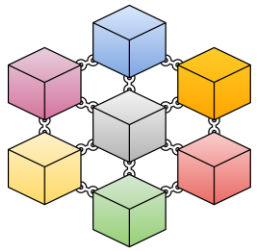
MPD Microservice Event Handling



ØMQ
ZeroMQ

ZeroMQ Messaging Patterns

- **Request-Reply** (REQ/REP), which connects a set of clients to a set of services. This is a remote procedure call and task distribution pattern.
- **Publish-Subscribe** (PUB/SUB), which connects a set of publishers to a set of subscribers. This is a data distribution pattern.
- **PIPELINE** (Push/Pool), which connects nodes in a fan-out/fan-in pattern that can have multiple steps and loops. This is a parallel task distribution and collection pattern.
- **WebSocket** (ZWS), which is starting after successful WebSockets handshake between the client and the server. ZWS message are binary websocket messages (the message opcode must be binary).
- **Exclusive Pair** (EXPAIR), which connects two sockets exclusively. This is a pattern for connecting two threads in a process, not to be confused with "normal" pairs of sockets.
- **Radio-Dish**, which used for one-to-many distribution of data from a single publisher to multiple subscribers in a fan out fashion *(draft state)*.
- and more …



What the microservices are?

Microservices - also known as the **microservice** architecture - is an architectural pattern that organizes an application into a collection of loosely coupled, fine-grained services that communicate through lightweight protocols.

This pattern is characterized by the ability to develop and deploy services independently, improving modularity, scalability, and adaptability.

In contrast to the traditional monolithic approach of a large, tightly coupled application, **microservices** provide a **cloud-native** architectural framework.

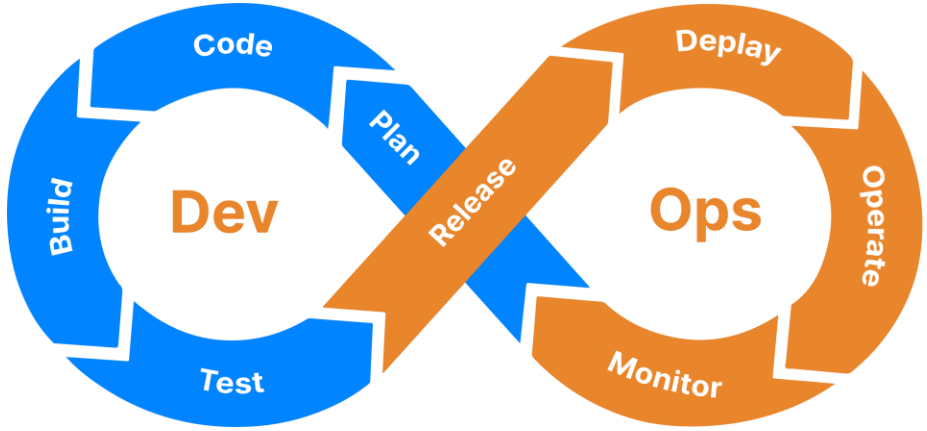
However, it introduces additional complexity, particularly in managing distributed systems and inter-service communication, making the initial implementation more challenging compared to a **monolithic** architecture.

Microservices-based applications have been a game changer. This approach to building applications comes with its own set of tools, technologies and best practices. It has compelling advantages that are now indispensable!

DevOps solution and CI/CD pipelines

DevOps is a philosophy, set of practices, and tools that combine software development (**Dev**) and IT operations (**Ops**) to improve software deployment and management.
DevOps may be one of the haziest terms in software development, but most of us agree that four basic activities make **DevOps** what it is:

- **CI/CD** pipelines
- Cloud infrastructure
- Test automation
- Configuration management



Some of these practices help accelerate, automate, and improve a specific phase. Others span several phases, helping teams create seamless processes that help improve productivity.
It brings both developers and operations together to form a smaller and active product development cycle. **DevOps** is a practice that every product should follow!

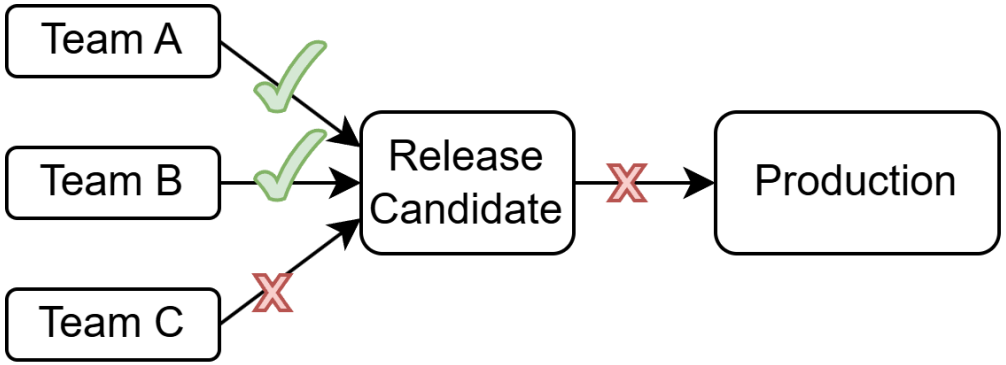
CI/CD pipeline for microservices architectures

Continuous **I**ntegration**/C**ontinuous **D**elivery and **D**eployment

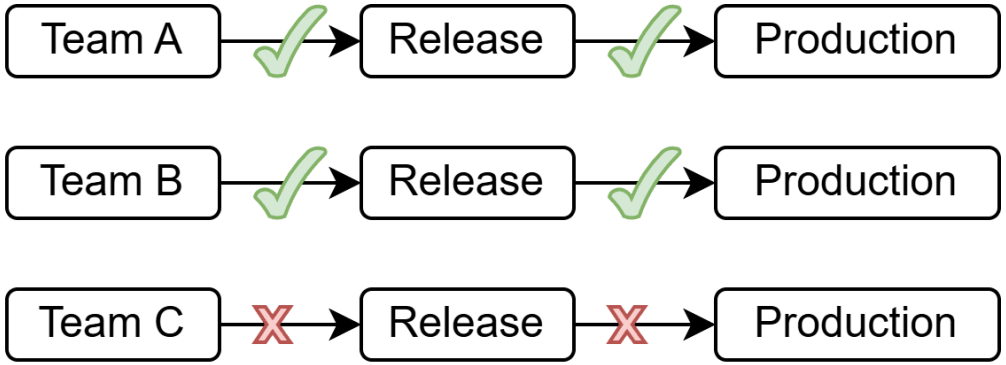
- **C**ontinuous **I**ntegration. Code changes are frequently merged into the main branch. Automated build and test processes ensure that code in the main branch is always production-quality. **CI** helps to catch bugs early in the development cycle, which makes them less expensive to fix.
- **C**ontinuous **D**elivery. Any code changes that pass the **CI** process are automatically published to a production-like environment. The goal is that your code should always be *ready to deploy* into production.
- **C**ontinuous **D**eployment. Code changes that pass the previous two steps are automatically deployed *into production*.

A robust CI/CD pipeline matters

In a traditional **monolithic** application, there is a single build pipeline whose output is the application executable.



Monolithic Application



Microservices

Following the **microservices** philosophy, there should never be a long release train where every team has to get in line.

Microservices and Dockers

Microservices represent an architectural approach where an application is structured as a collection of small, independently deployable services. Each microservice focuses on a specific capability, runs in its own process, and communicates with other services via APIs.

Microservice is a software design pattern, while **Docker** is a tool for implementing and managing containerized applications.

Docker is a platform for developing, shipping, and running applications using containerization.

Docker is often used to deploy microservices. Each microservice can be packaged into its own Docker container, providing isolation and simplifying deployment and scaling.

kubernetes

Kubernetes Orchestration

Kubernetes, also known as **K8s**, is the most popular container orchestration platform, providing automation for deployment, scaling, and managing complex, multi-container workloads.

Kubernetes provides you with:

- Service discovery, load balancing and scalability
- Storage orchestration
- Automated rollouts and rollbacks
- Automatic bin packing
- Self-healing for high availability
- Designed for extensibility
- and more...

The name **Kubernetes** originates from Greek, meaning helmsman.

K8s as an abbreviation results from counting the eight letters between the "K" and the "s".

Developed by Google and released in 2014, **Kubernetes** became one of the fastest-growing projects in open source software's history.

K0s vs K3s vs K8s

K0s, **K3s** and **K8s** are three different orchestration systems used to deploy and manage containers.

- **K0s** is a container native platform based on distributed systems, such as Apache Kafka.
  - It has a strong focus on stream processing and data-driven applications.
  - **K0s** was developed by **Google** and is used in many of its products.
  - This container-native platform was engineered to efficiently run containerized applications in a distributed computing environment. It can manage millions of containers and provide a reliable and scalable platform to handle enterprise-level workloads.
- **K3s** is a hybrid container orchestration platform that developed by **CoreOS**.
  - It is a lightweight platform that can deploy and run containers in any environment.
  - It is designed to run on bare metal, virtual machines or top of other cloud platforms like Amazon Web Services (AWS) and Google Cloud Platform (GCP).
- **K8s** is the most popular general-purpose container orchestration platform and the standard for modern containerized applications. It allows users to deploy and manage applications in various environments, including on-premise and cloud-based.

The choice between **K0s**, **K3s** and **K8s** depends on the user's specific requirements.

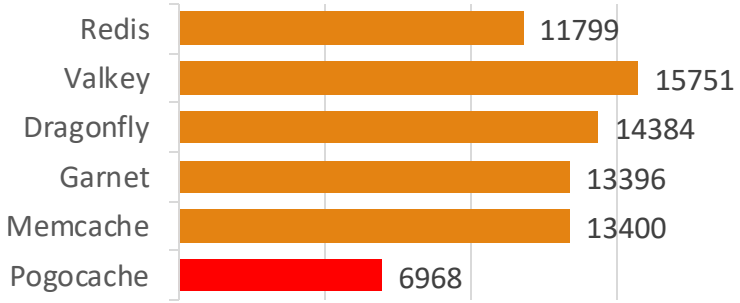
They all provide similar functionality, so the main difference is in their design, deployment, and installation.

POGOCACHE

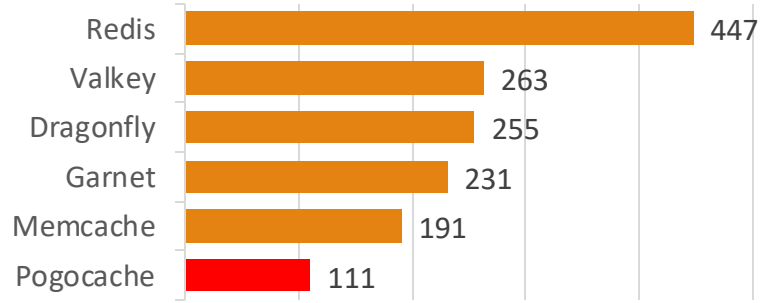
Pogocache caching software

Pogocache is fast caching software built from scratch with a focus on low latency and CPU efficiency and published by *Josh Baker* (https://github.com/tidwall) in July 2025 (current release v1.0.2).
Pogocache is faster than *Memcache*, *Valkey*, *Redis*, *Dragonfly*, and *Garnet*. It has the lowest latency per request, providing the quickest response times. **Pogocache** stores entries (key-value pairs) in a sharded hashmap.
Optionally instead of running **Pogocache** as a server-based program, the self-contained *pogocache.c* file can be compiled into existing software, bypassing the network and directly accessing the cache programmatically.

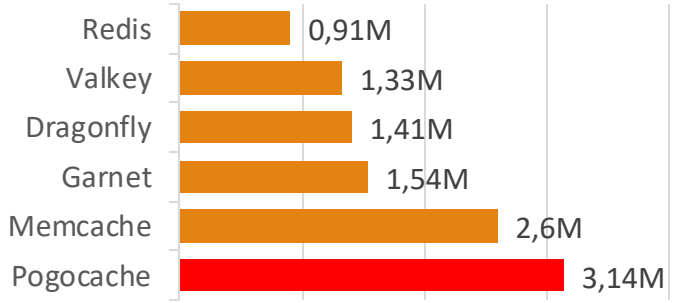
CPU cycles per operation
(less is better)



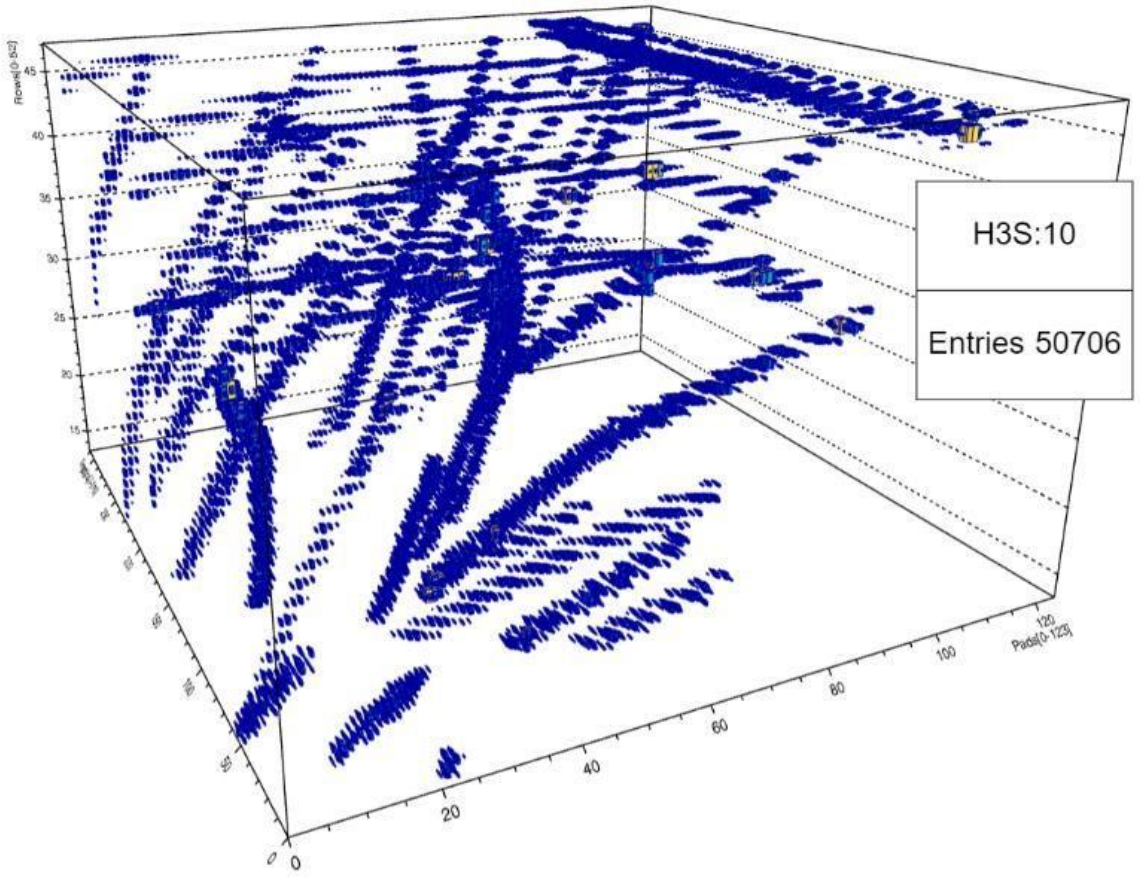
Duration per query, µs
(less is better)

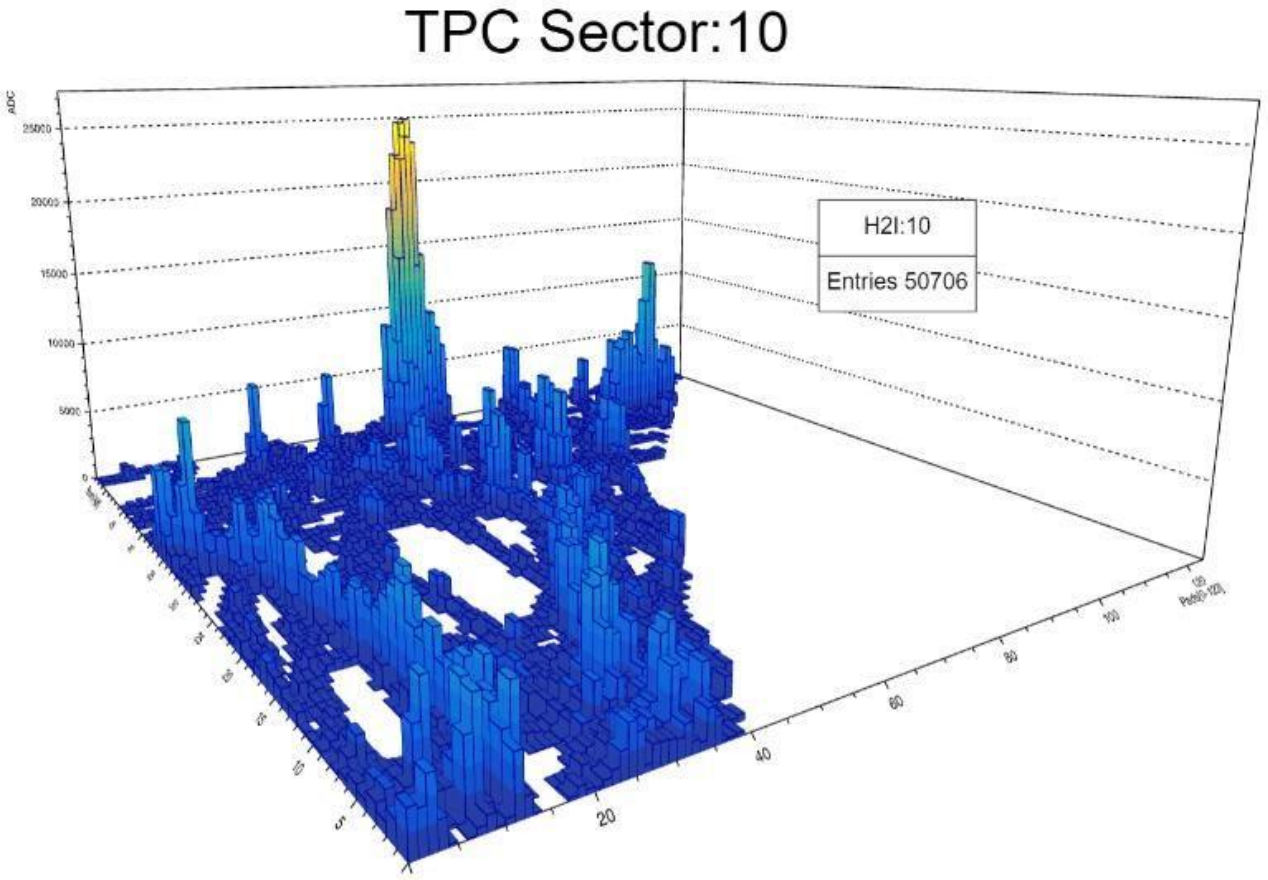


Queries per second
(more is better)



TPC Sector Digits before Clustering





Historical context for Kubernetes