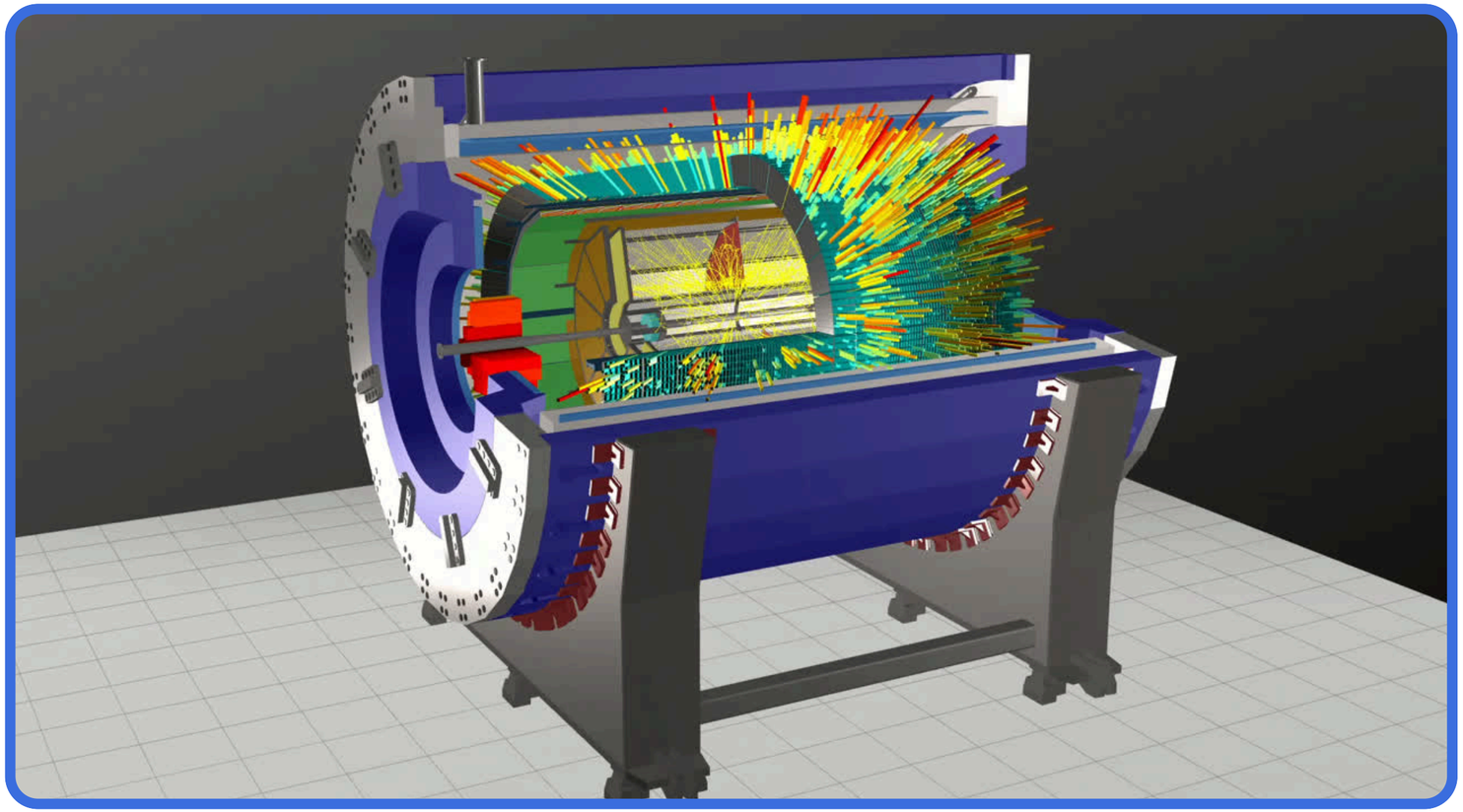
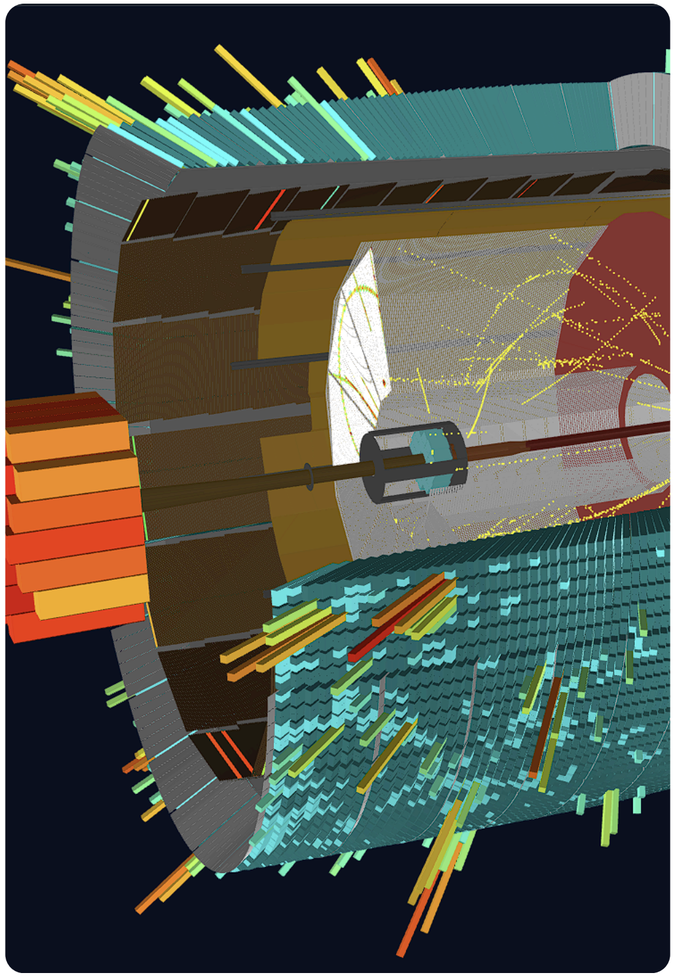
# MPD Online Data Monitoring



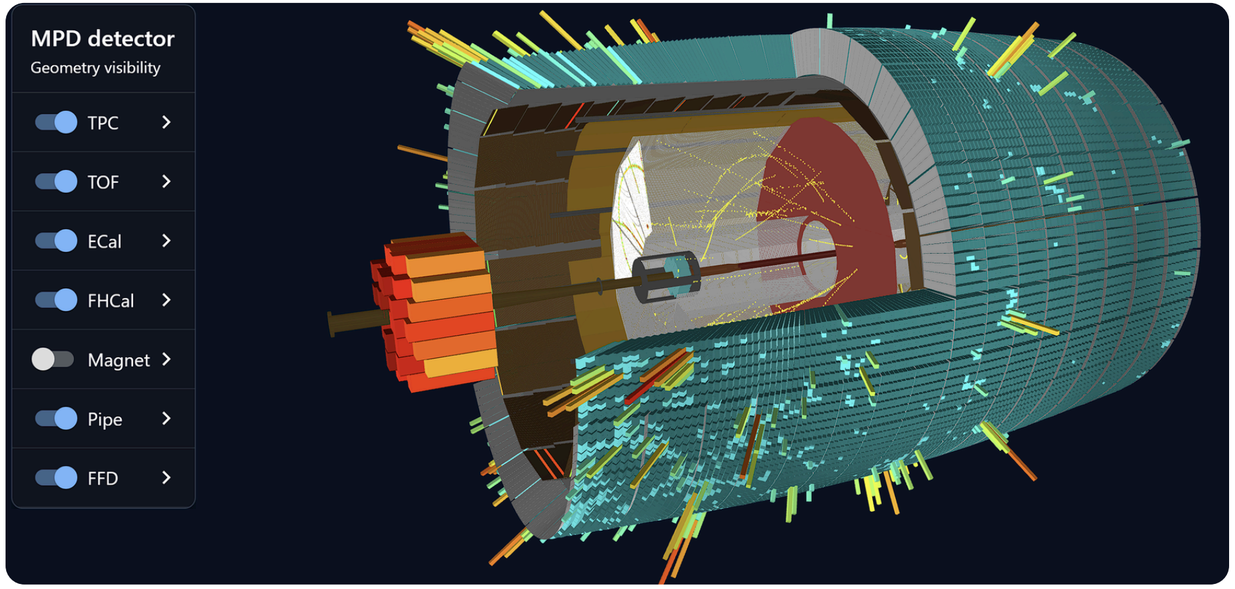
**A. Krylov, JINR/LHEP**
avkrylov@jinr.ru

# Overview



- Online Event Display
- What is behind it?
- Decoder C++ library
- Microservice architecture
- Kubernetes cluster preparation
- Future plans

# Online Event Display



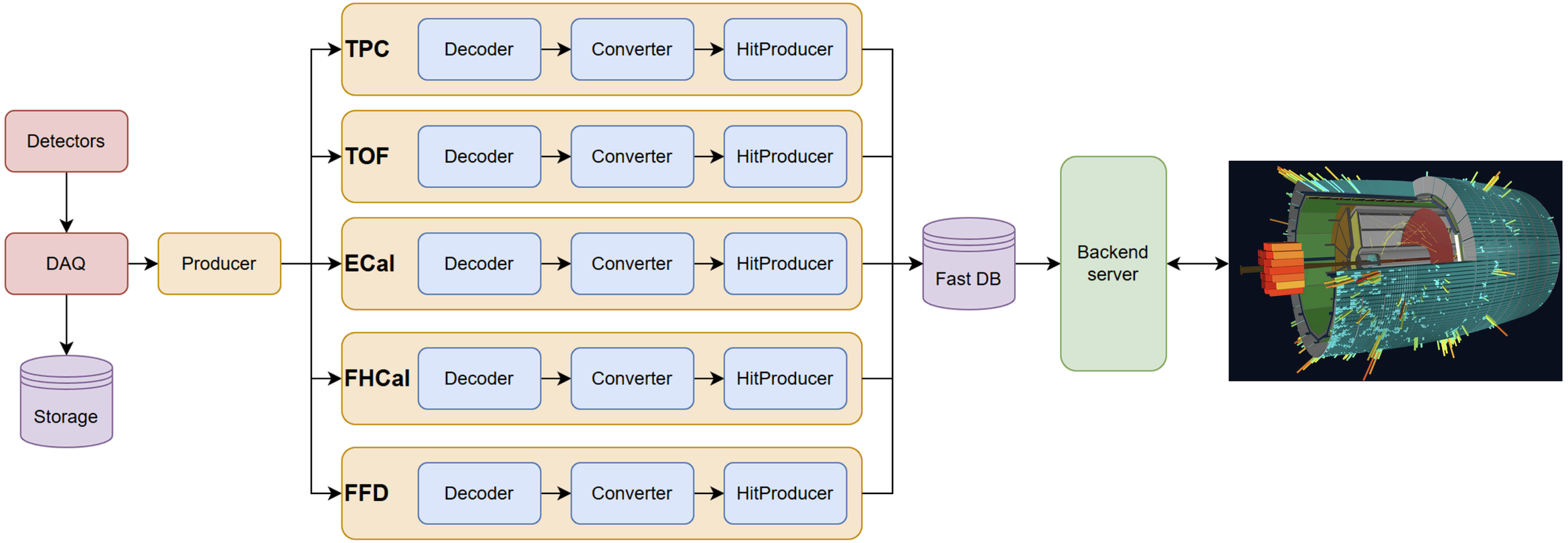

**3D Visualization**
comprehensive information on the progress of the experiment

**Online Monitoring**
online monitoring of experimental data and event reconstruction

**Data Quality Assurance**
detector anomalies and potential processing problems

**Detector Analysis**
event information across subsystems (TPC, TOF, ECAL, FHCAL)

# What is behind Event Display?




**Per-detector split**
DAQ raw data are divided by detector

**Common chain**
Decoder → Converter → Hit Producer for every detector

**Fast DB → Event Display**
results are stored in Valkey and served to the display

# Decoder Library

**01**

**Current state**

Each detector team keeps its own decoder (MPDROOT, ROOT, C++)

**02**

**Problem**

ROOT/MPDROOT code **does not meet online performance** and integration/modularity needs

**03**

**Requirement**

One library in standard C++ without ROOT or MPDROOT dependencies

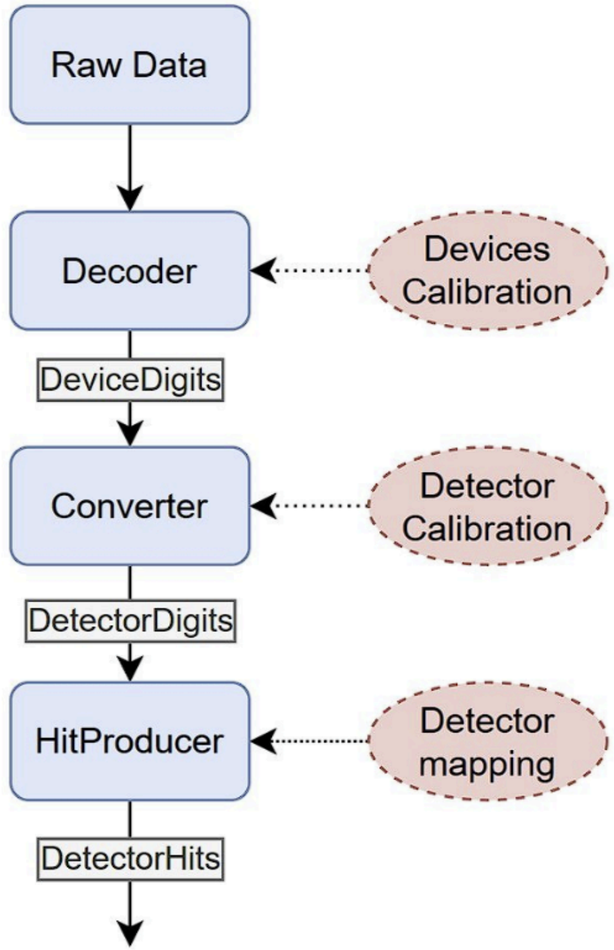
**04**

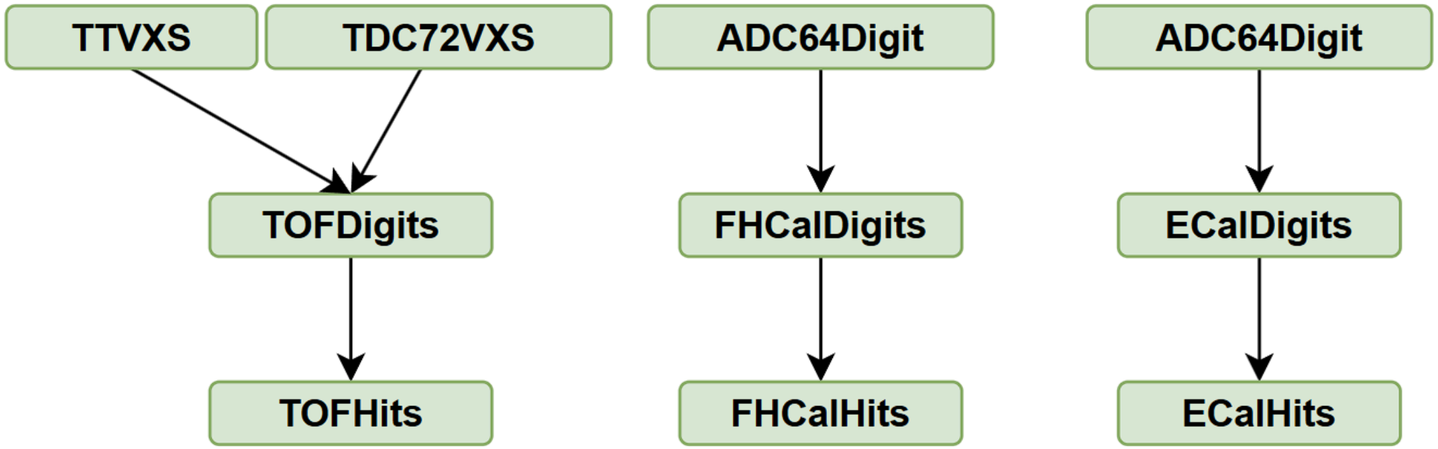
**Design**

Factory pattern: new detectors plug in their own decoding algorithm

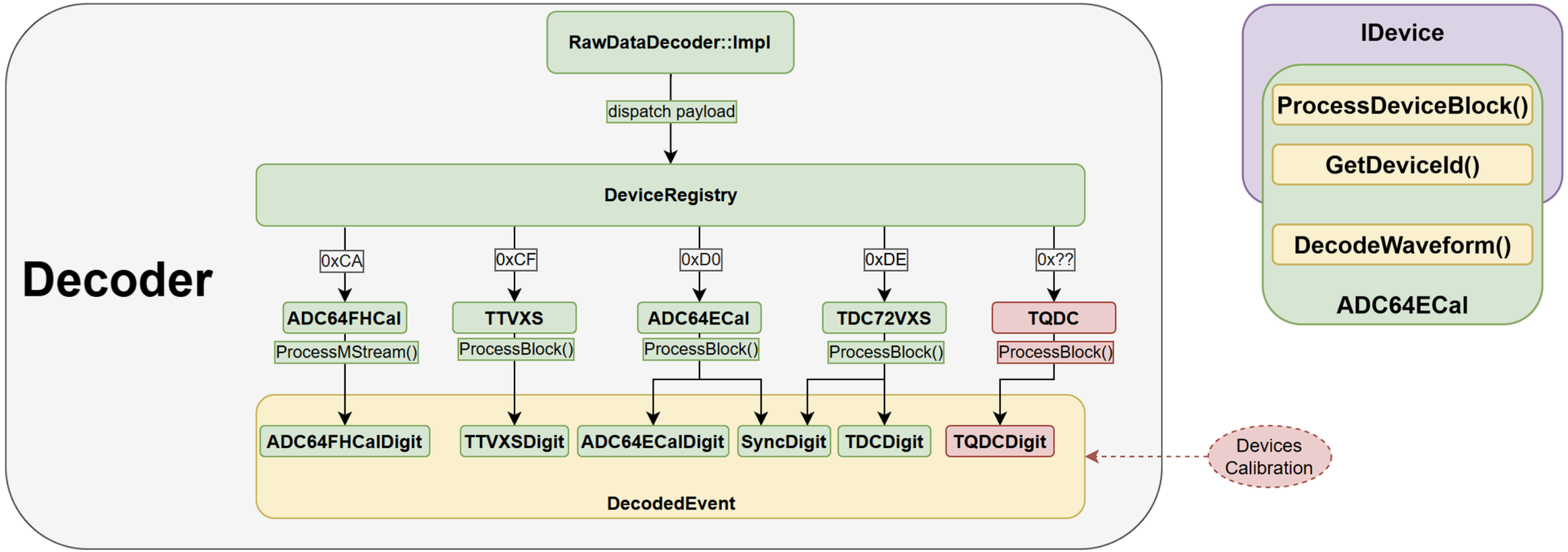
# Decoder Pipeline

- **Decoder**: Parses raw binary data into device-specific records (electronic digits).
- **Converter**: Processes decoded signals into detector-specific digits. (detector digits).
- **Hit Producer**: Applies calibration and detector-specific algorithms to reconstruct actual detector hits.



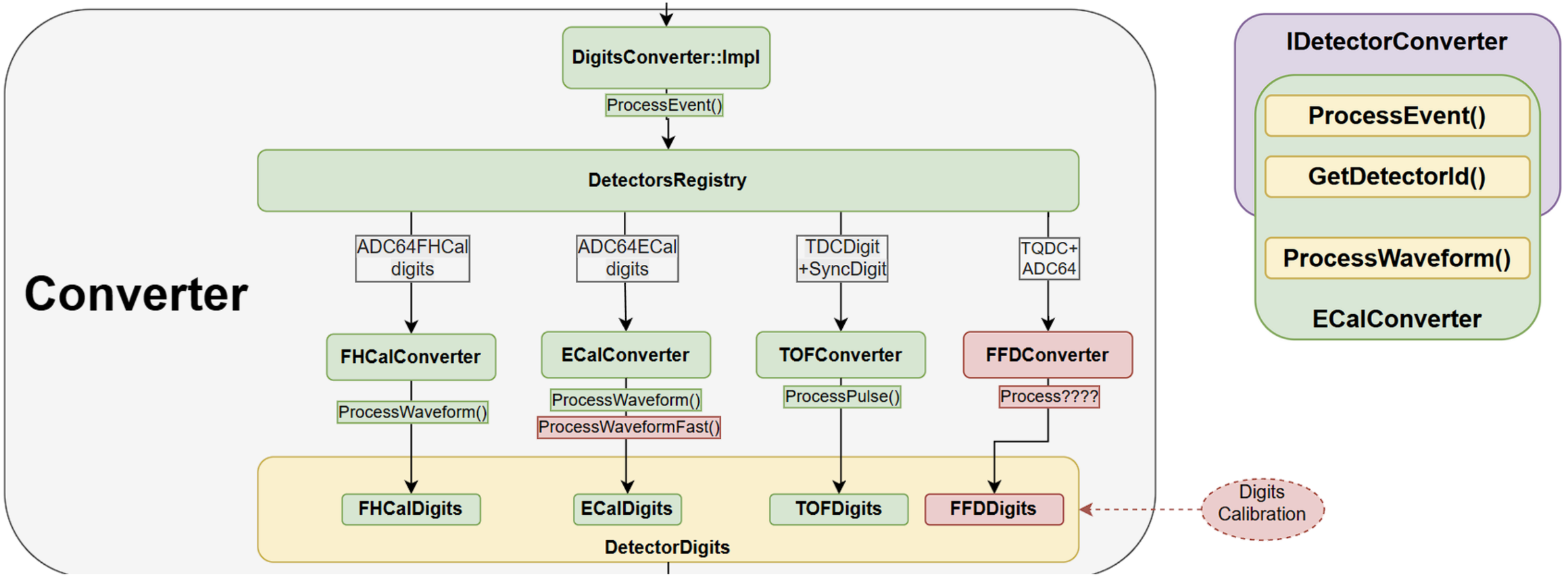
# Component: Decoder




**Common Interface (IDevice)**: Defines the operations that every device decoder must support.

**DeviceRegistry:** Stores decoder instances and selects the appropriate handler using the device ID.

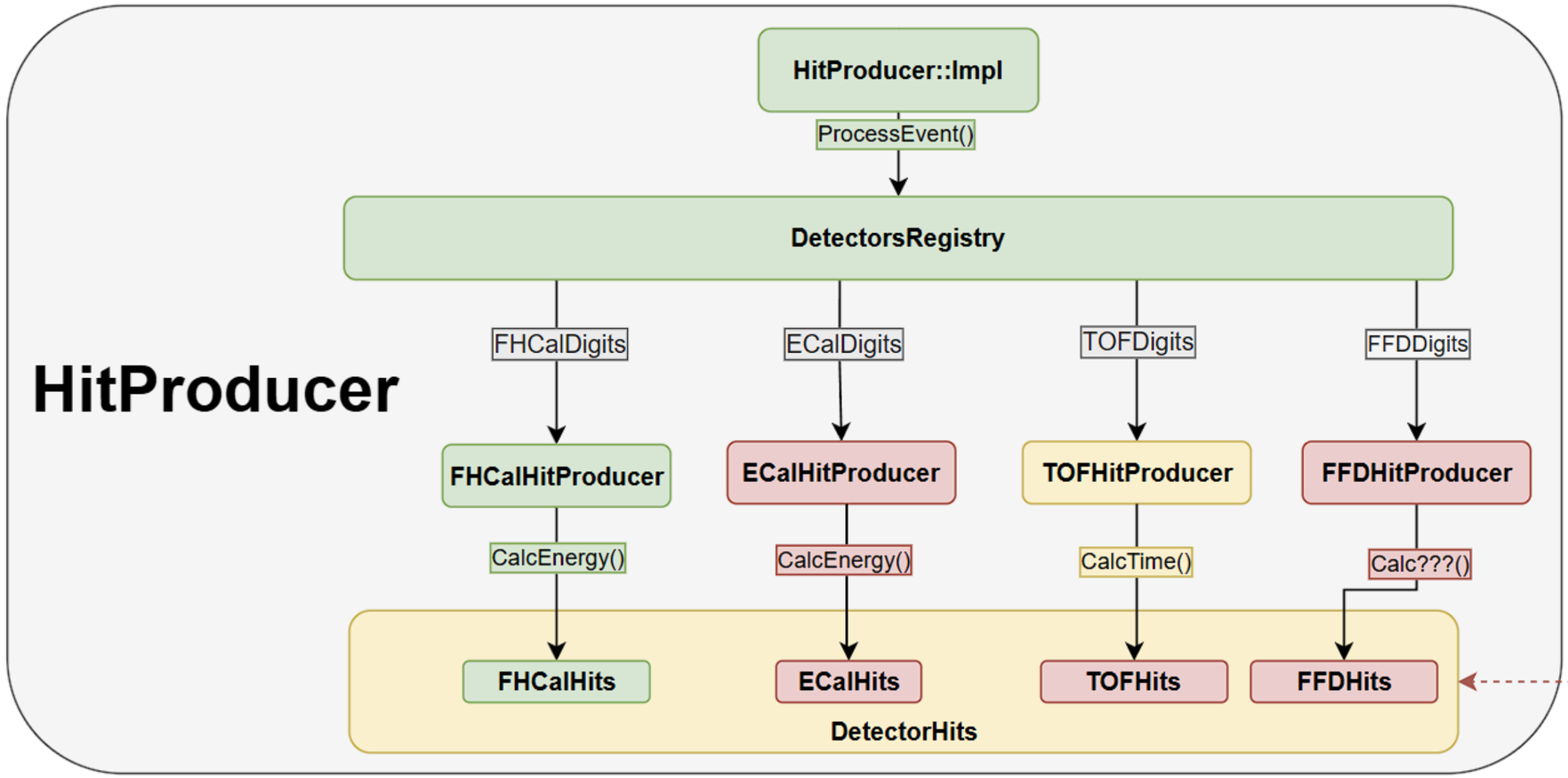
# Component: Converter

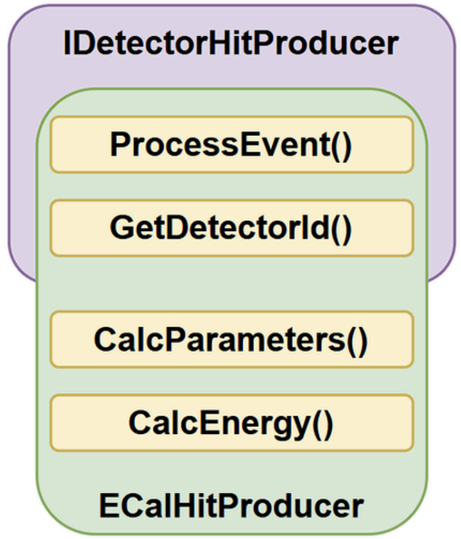



**Common Interface (IDetectorConverter):** Defines the operations that every detector converter must support.

**DetectorsRegistry:** Owns the detector converters and invokes them for each decoded event.

# Component: Hit Producer

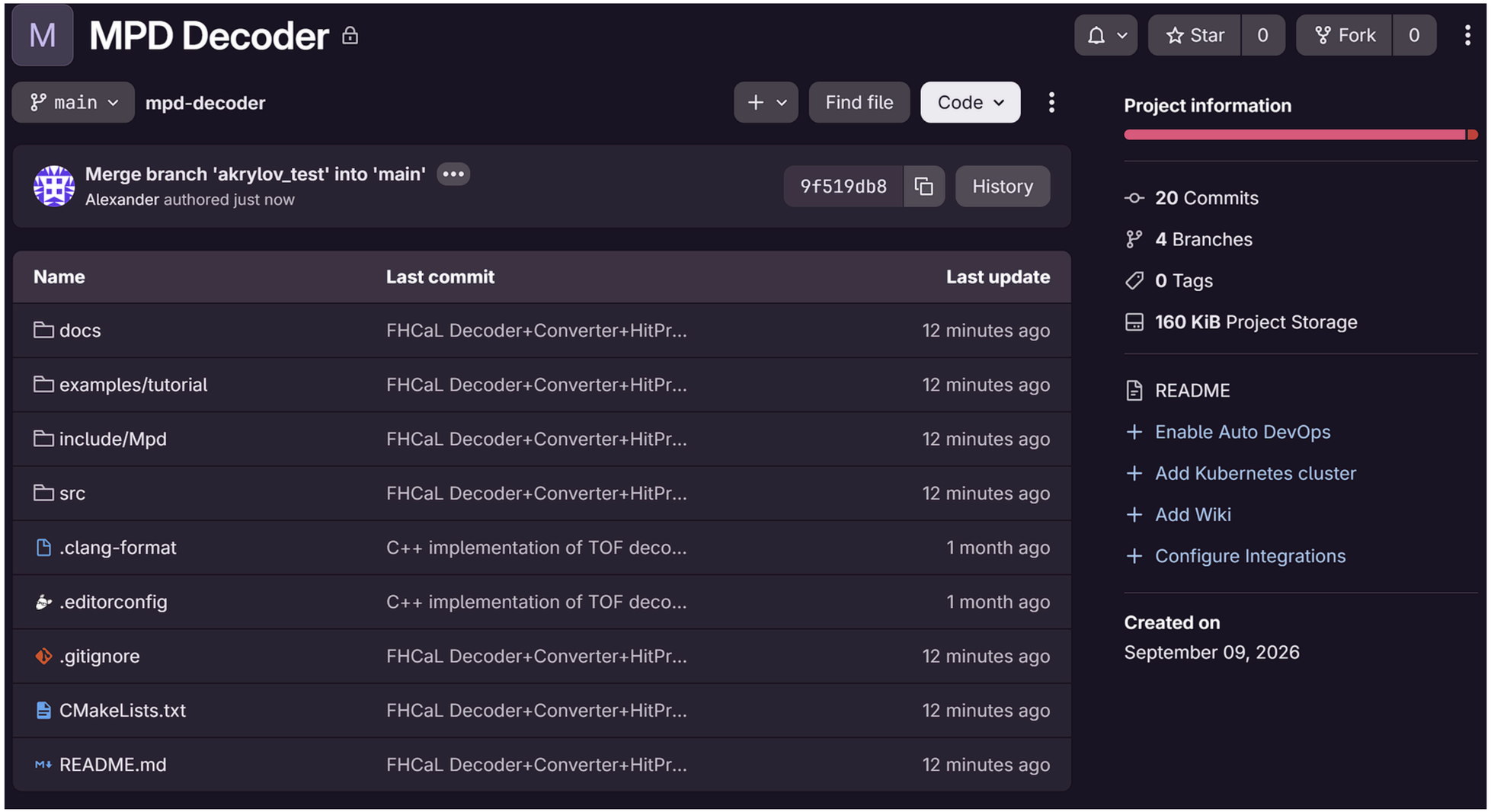



**Common Interface (IDetectorHitProducer):** Defines the operations that every detector producer must support.

**DetectorsRegistry:** Owns the detector producers and invokes them for each detector digits.

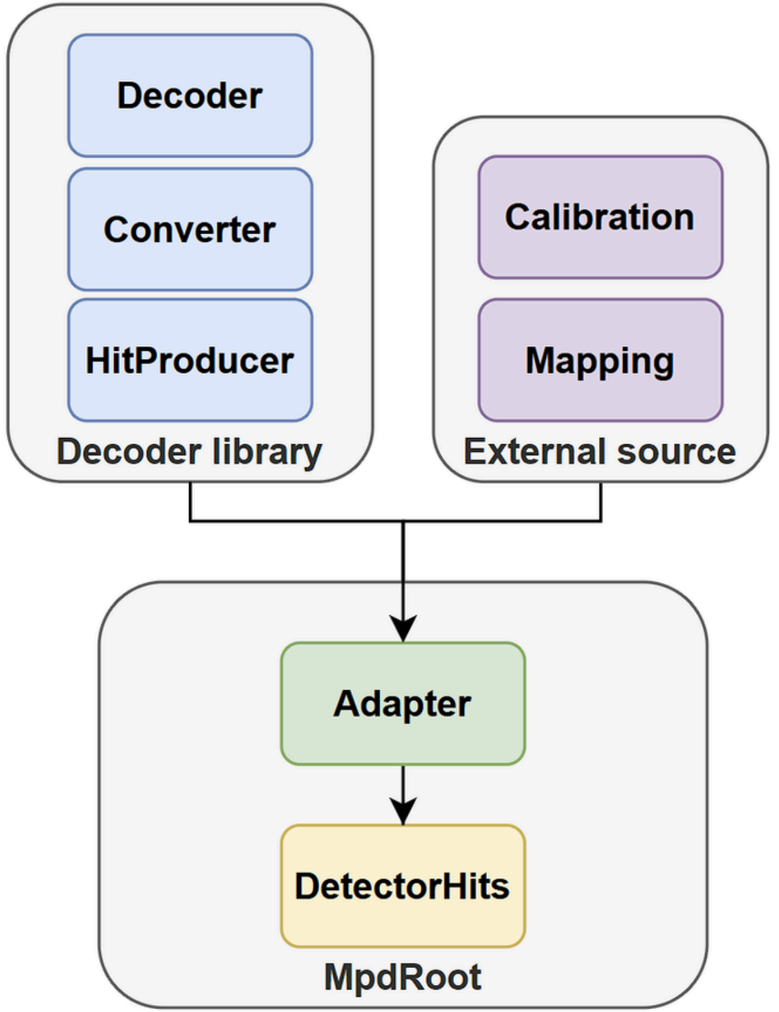
# Git repository



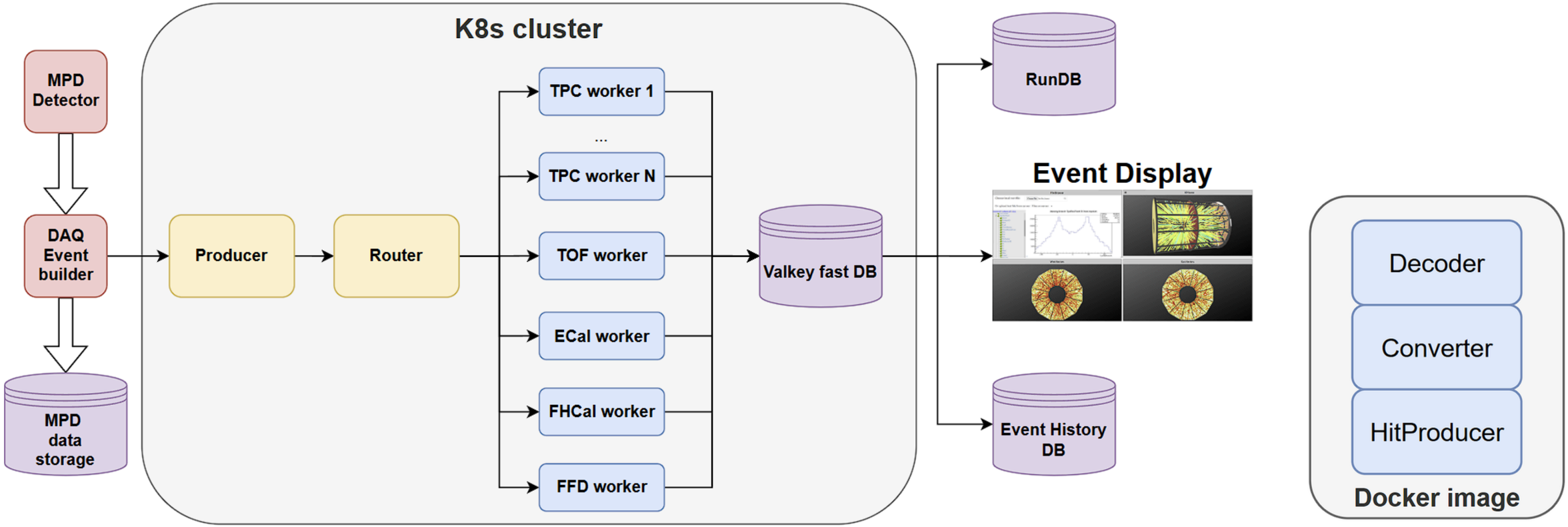
# MpdRoot compatibility

- A dedicated adapter will convert FHCalHit objects into MPDROOT's MpdZdcDigi format using externally supplied calibration and mapping.
- This enables integration with existing MPDROOT analysis workflows while keeping the decoder library independent of ROOT and FairRoot.

```
for (const FHCalHit& hit : producer.hits().FHCal) {
        auto* digi = new(output[i]) MpdZdcDigi();
        digi->SetDetectorID(hit.GetDetectorID());
        digi->SetModuleID(hit.GetModuleID());
        digi->SetChannelID(hit.GetChannelID());
        digi->AddEloss(energy);
        digi->SetELossReco(energy);
    }
```



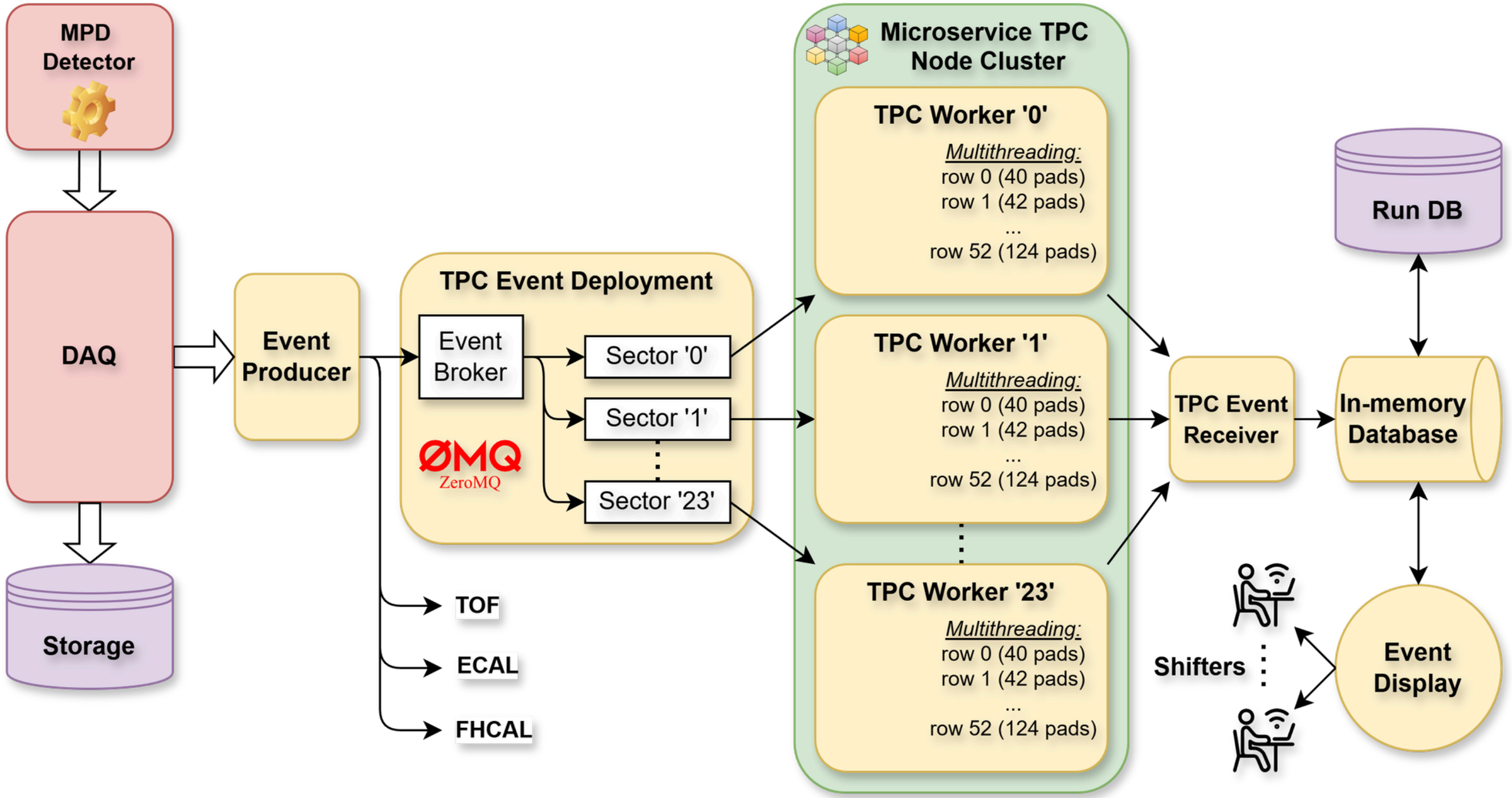
# Data Pipeline architecture



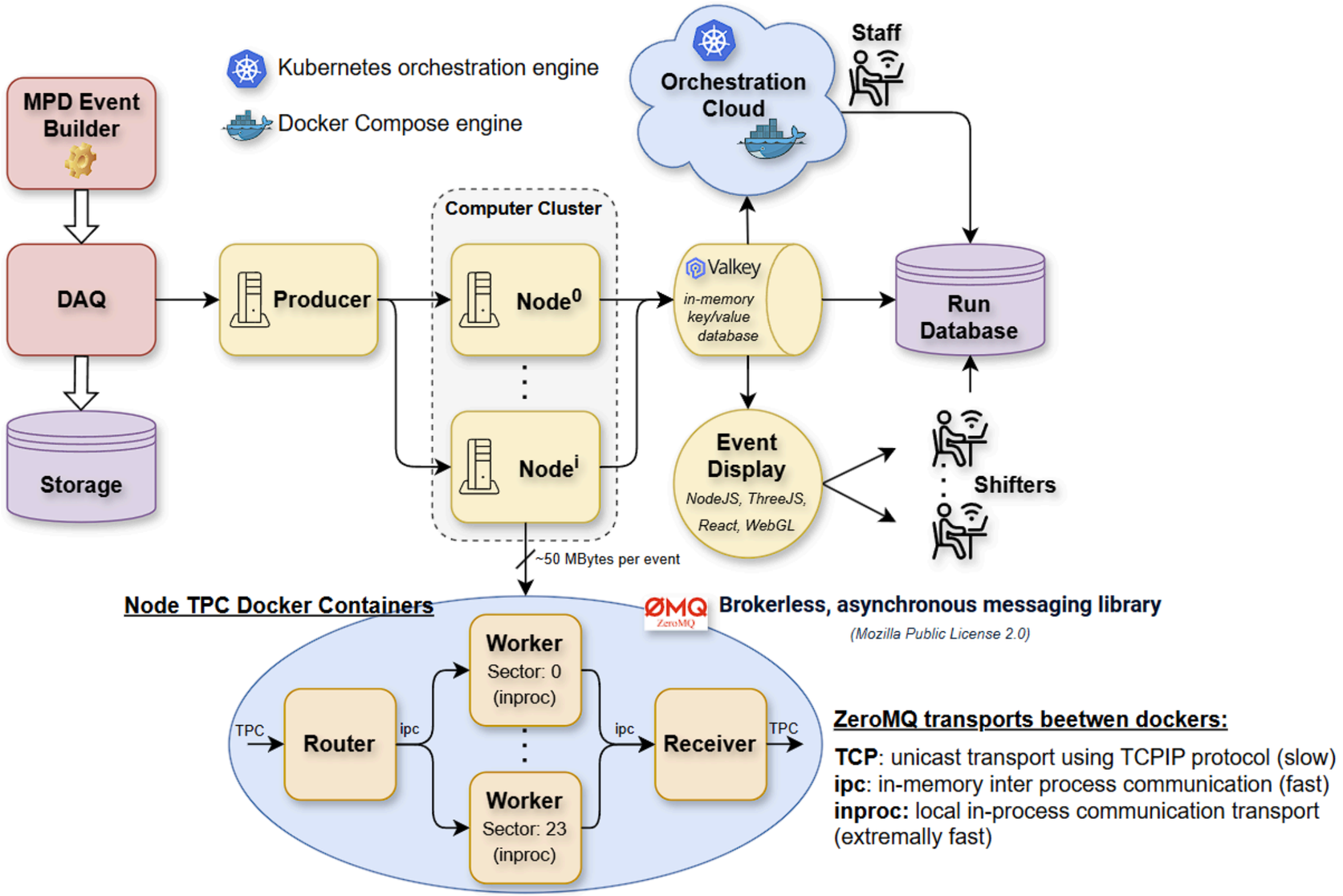

**Per-detector workers** run as microservices in the **Kubernetes** (K8s) cluster. TPC uses N parallel workers, other - a few or just one.

**One Docker image**
Contains Decoder, Converter and Hit Producer

# TPC Pipeline architecture



# TPC pipeline new approach



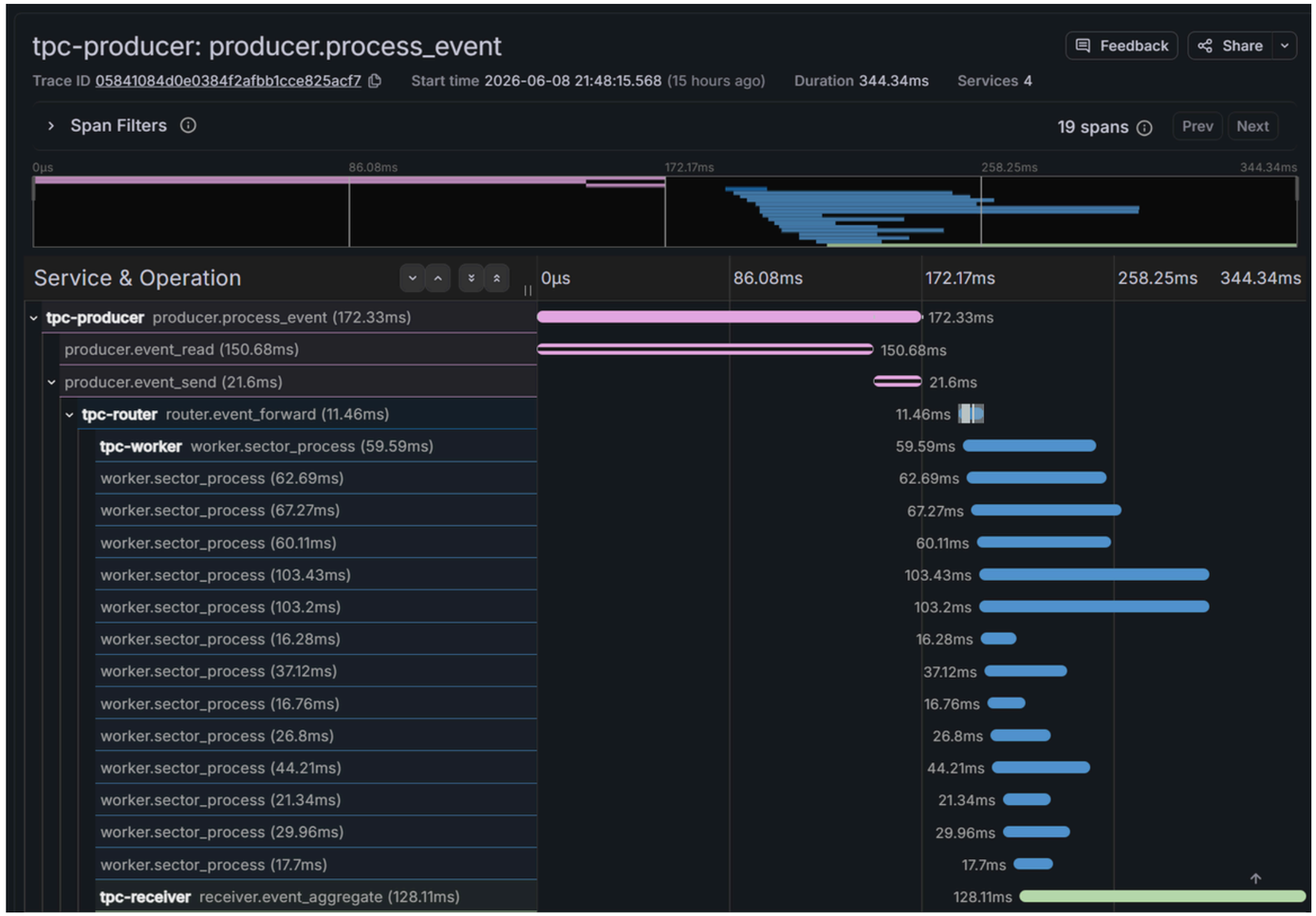
# TPC Pipeline Monitoring

Without any real data source (DAQ or its digital twin) system can be tested on simulation data (UrQMD generator + ROOT).

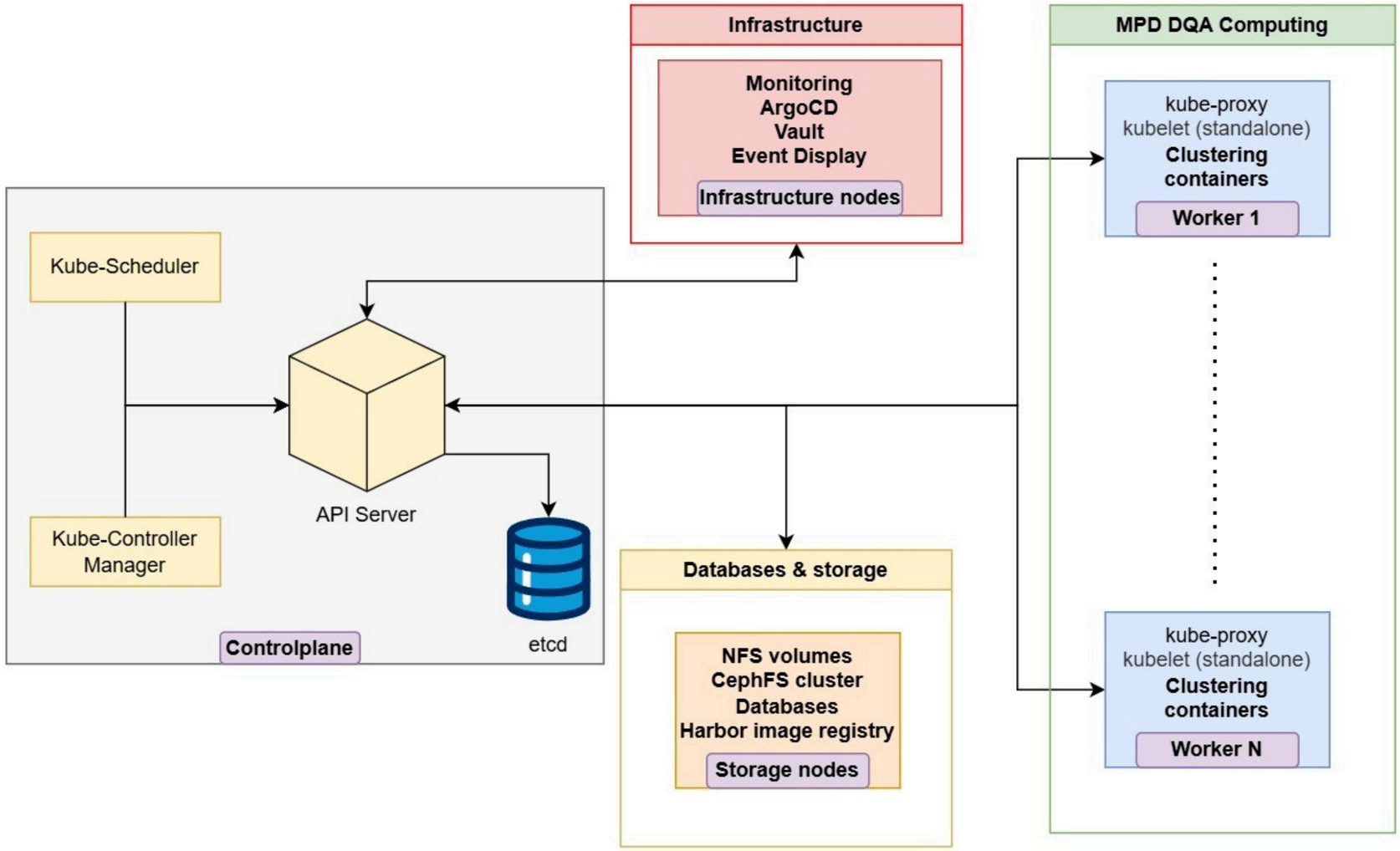
Excluding ROOT reading pipeline achieves **100 ms** in main part.

**Main difficulties**:

- Network (unnecessary hops, internode communication);
- Broker technology (ZeroMQ require testing of throughout possibilities)
- Processed event compilation (receiver awaits all parts of the event).



# K8s infrastructure




## Autoscaling

processing capacity follows the data flow via Horizontal Pod Autoscaler

## Observability

every component produces metrics, logs and traces for performance analysis

## GitOps

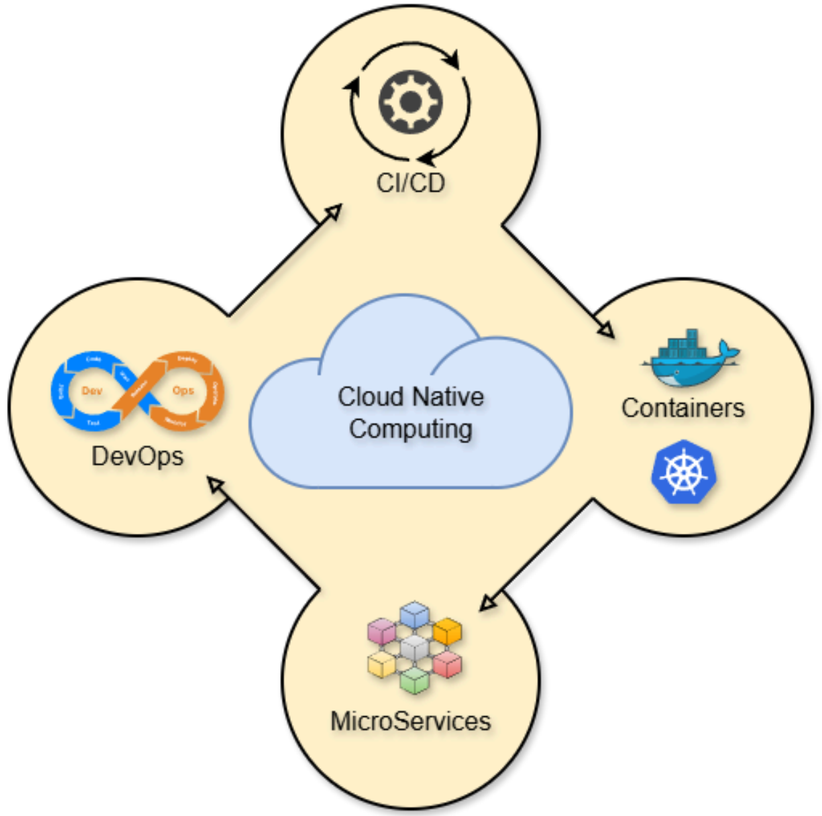
infrastructure and data processing services are deployed from Git (IaC)

# Infrastructure as Code (IaC)

Subsystems are managed through **ArgoCD** following the **GitOps** methodology.

**Main idea:** configuration and environments must be declarative and version controlled (Git repos).

The images for the online processingsystem are built via **GitLab CI/CD pipelines**, images/charts (artifacts) are stored in Harbor



# K8s: goals & challenges

During the test data processing must be defined and documented **technical and functional demands/targets** for the system in terms of Site Reliability Engineering (SRE):

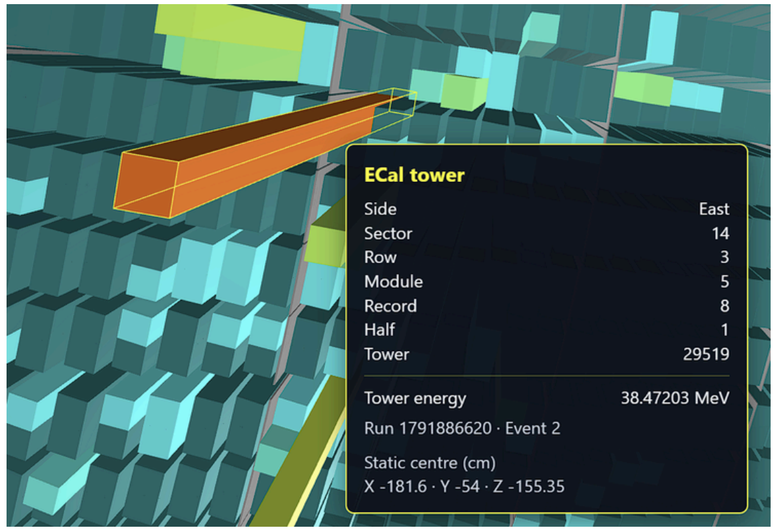
- **Service Level Objective** - target behavior expressed by acceptable value (or values interval) based on metrics;
- **Service Level Indicator** - metrics that indicates whether system goes well or not in terms of SLA;

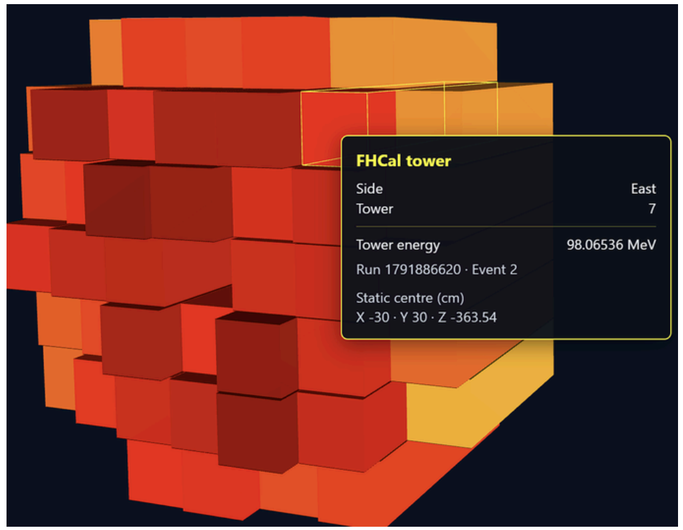
Achievement of this goals provides a proper state of whole system during the intensive online processing.

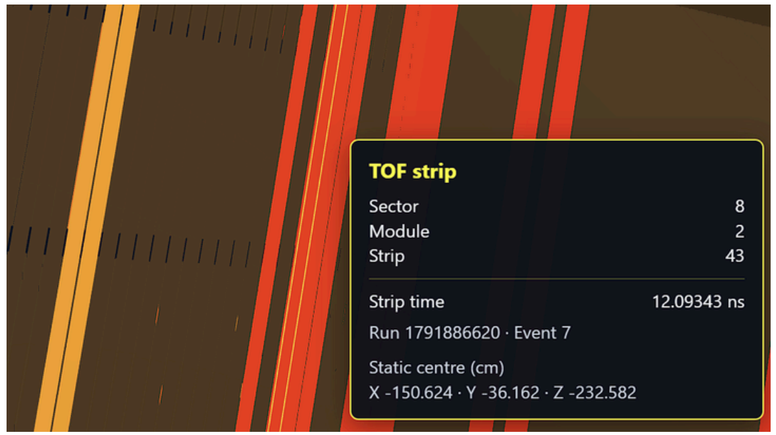
| Target | SLA | SLI* |
|---|---|---|
| Fast data processing | within 1-50 ms per event | Trace duration, P95 |
| Resource buffer | Utilization<=80%, other - buffer | CPU/RAM Utilization, Network I/O |

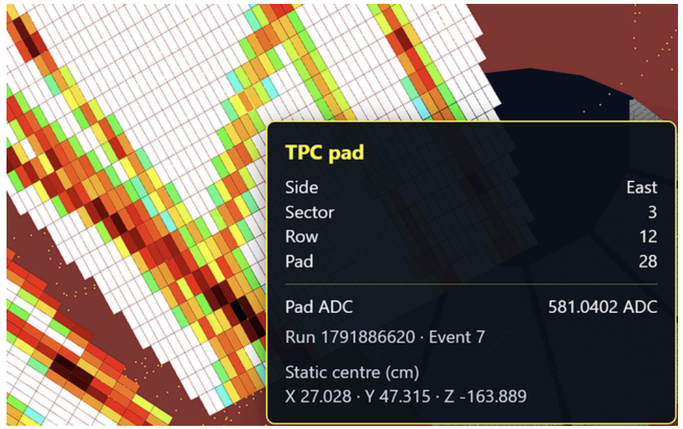
(*) Real telemetry and performance metrics can be obtained **ONLY** after MPD detector technical run

# Web Event Display



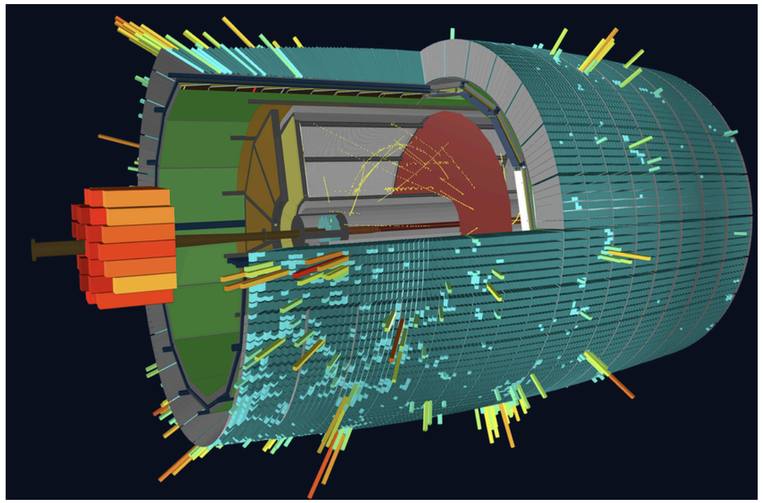



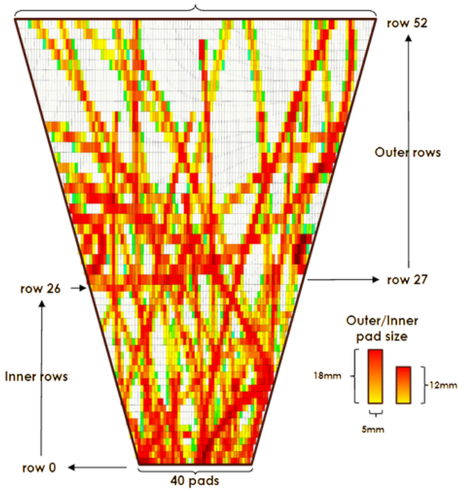

| | |
|---|---|
| **Backend** | Node.js |
| **Frontend** | React, ThreeJS |
| **Data source** | Valkey (Redis-like fast DB) |
| **Events** | MPDEvent |

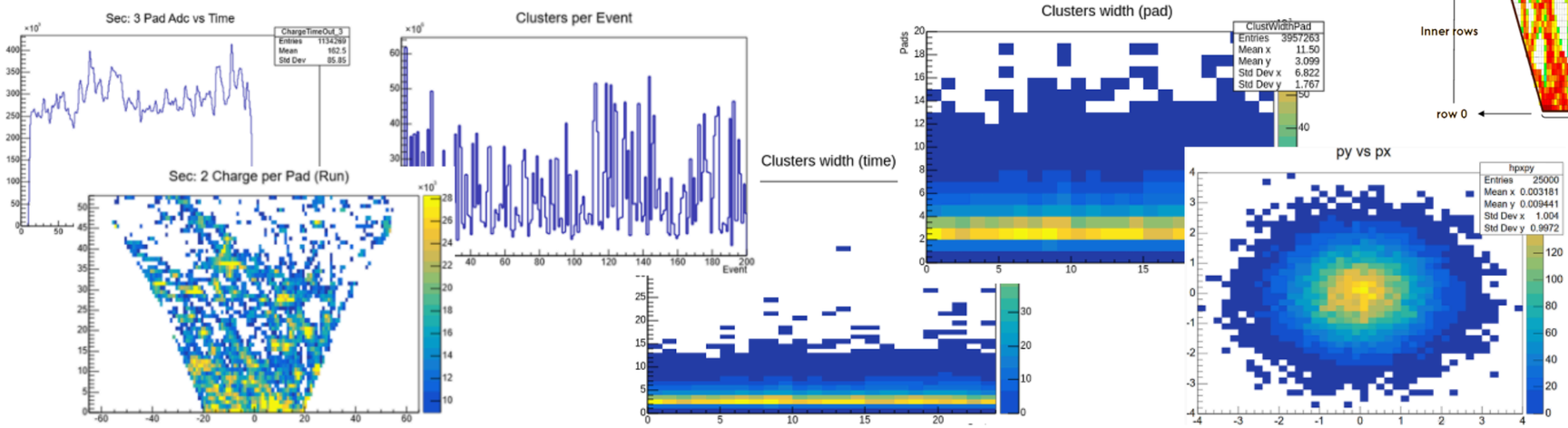
*Updated online as events arrive*



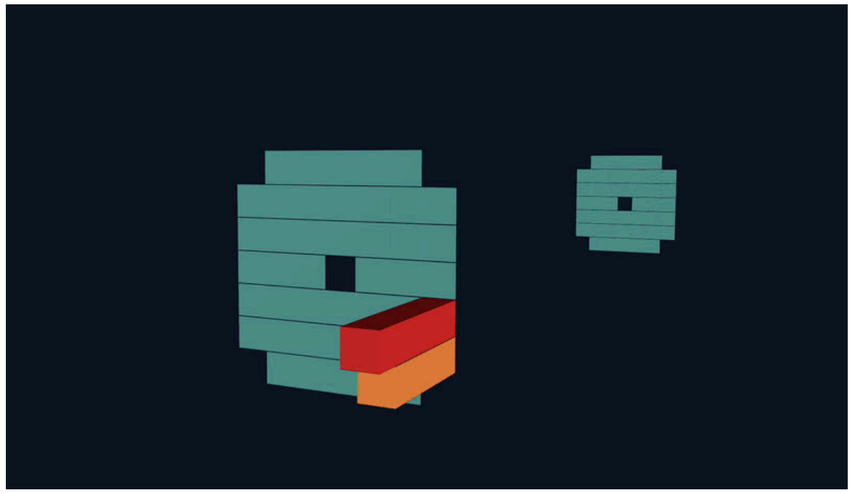
# TPC Data Quality Assurance (DQA)

- ADC distribution per sector – 24 histograms (for current run/event)
- ADC distribution per timebucket – 24 histograms (for current run/event)
- General cluster information per sector – 4 x 24 histograms
- General cluster information per event – 4 histograms
- and more…
- Total number: more then 150 histograms (**Only for TPC!!!)**

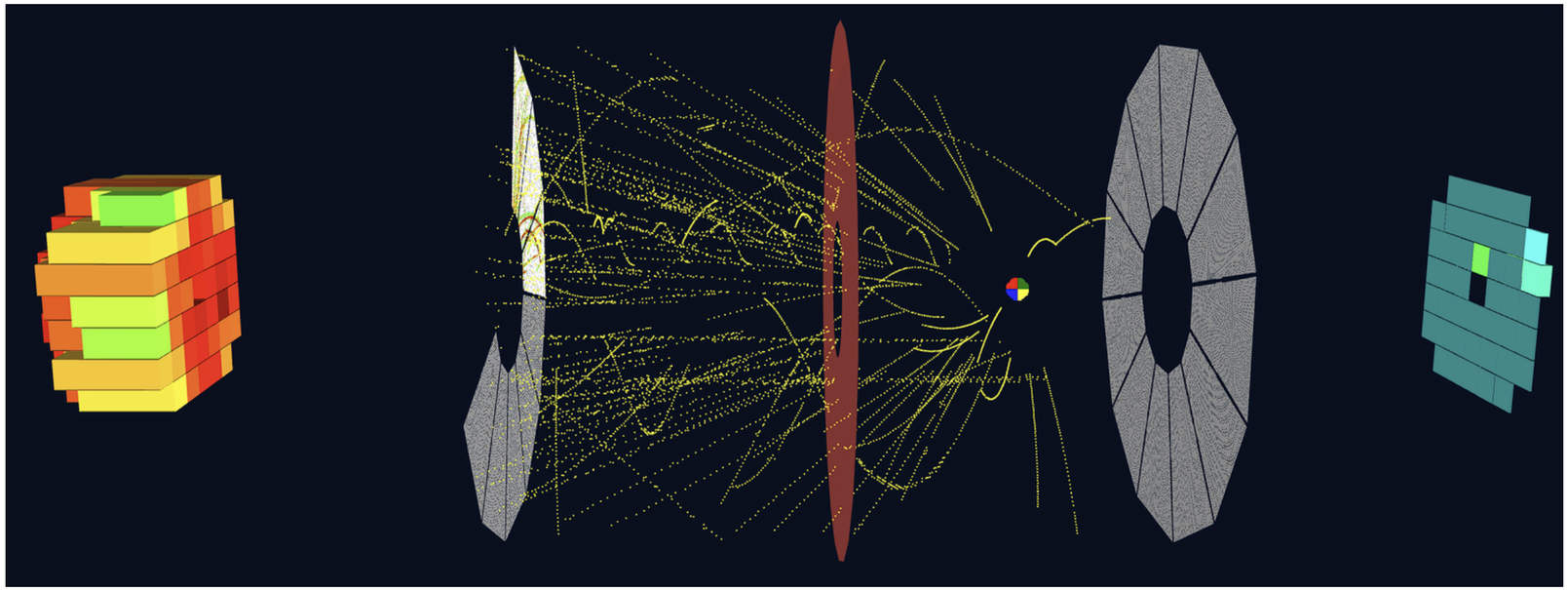


# Real data visualization



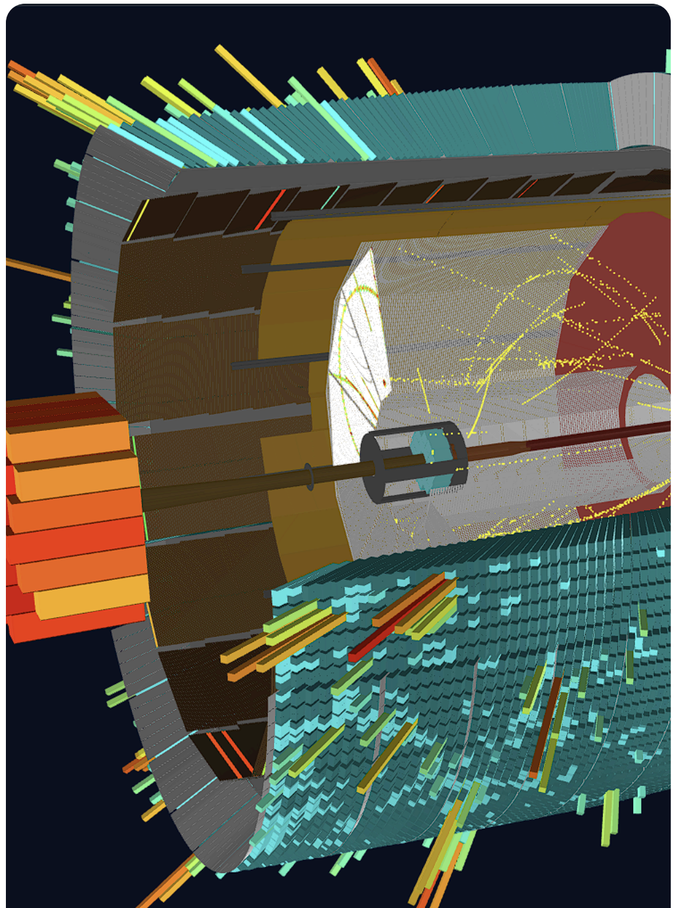


# Summary

- Event Display is ready to be tested on real MPD experiment data.
- Decoder library will be tested during technical run, validated and then implemented in mpdroot software. Examples and tutorials will be included.
- Decoder library git (Internal): https://git.jinr.ru/mpd-software/mpd-dqa/mpd-decoder
- Decoder library **must be validated by detector groups** before active usage.
- At the moment we are implementing new approach for microservices architecture to improve processing speed for online event MPD data handler of TPC detector.
- For the hundreds of DQA histograms we should also design a framework that visualizes them in a format suitable for operators and other staff in the Control Room.

# Future Plans



- TQDC decoder, FFD converter and hit producer
- Performance evaluation using real data
- Security development
- HA control plane, Ceph, Infra/Computing separation

# Thank you!

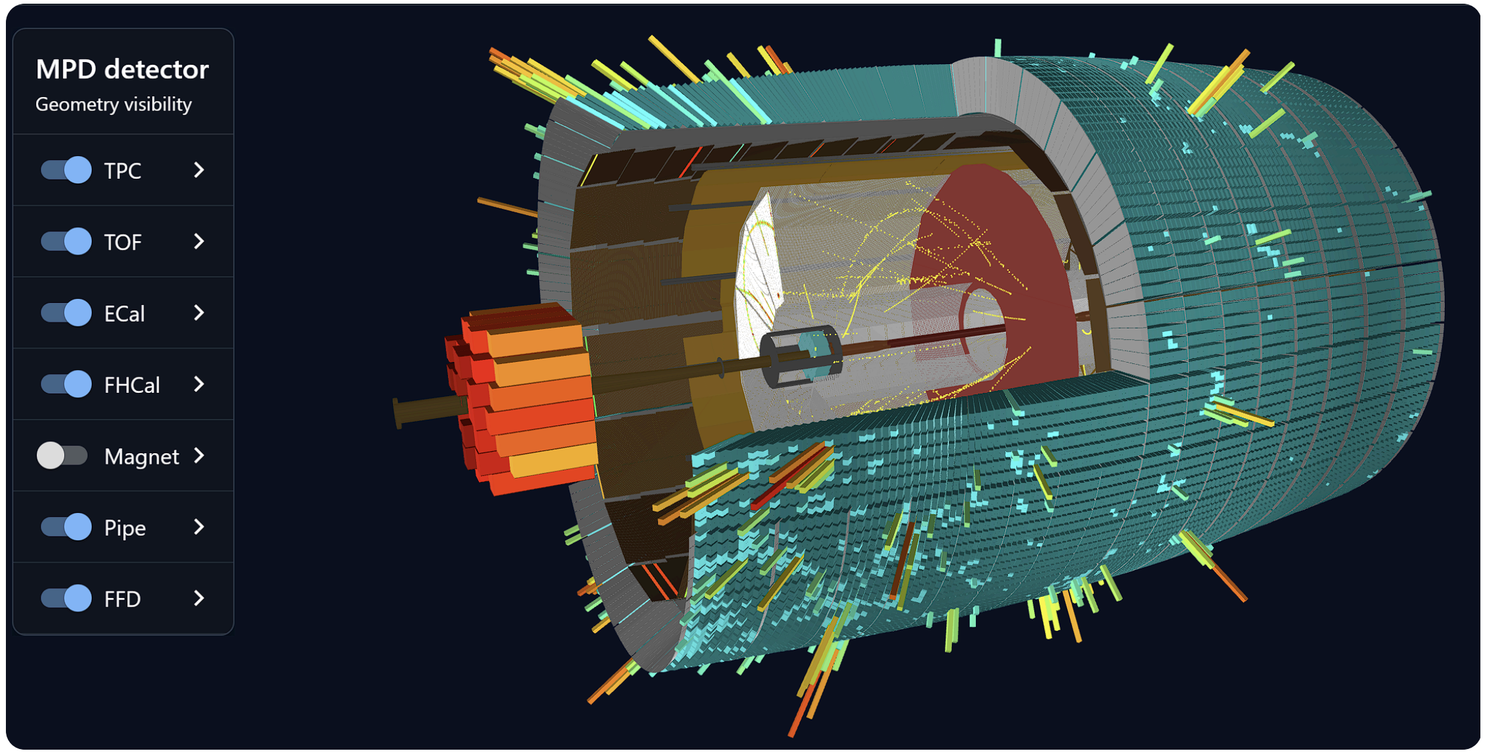
**Contact:**
avkrylov@jinr.ru

# K8s infrastructure: goals & challenges

| Target | SLA | SLI* |
|---|---|---|
| Fast data processing | within 50-150 ms per event | Trace duration, P95 |
| Resource buffer | Utilization<=80%, other - buffer | CPU/RAM Utilization, Network I/O |
| No-downtime | 99.99% of Run time | Time of up/down services, P95 |

(*) Real telemetry and performance metrics can be obtained **ONLY** after MPD detector technical run

# Web Event Display




| | |
|---|---|
| **Backend** | Node.js |
| **Frontend** | React |
| **Data source** | Valkey (Redis-like fast DB) |
| **Events** | MsgTpcEvent |
| **Histograms** | Hist1D / Hist2D |

*Updated online as events arrive*

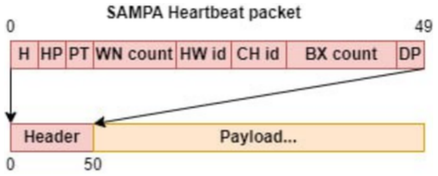
# TPC Sector Data

## SAMPA format

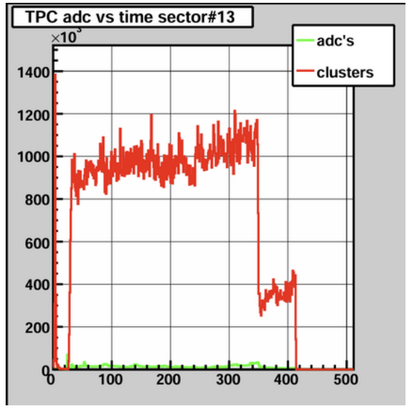
TPC raw data from the detector readout

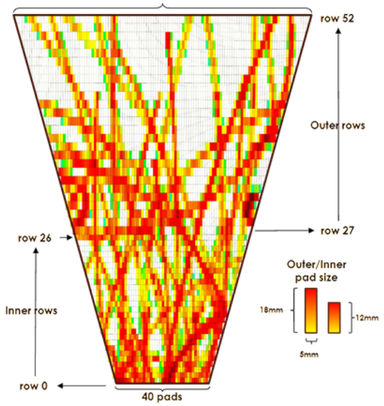
## TPC digits

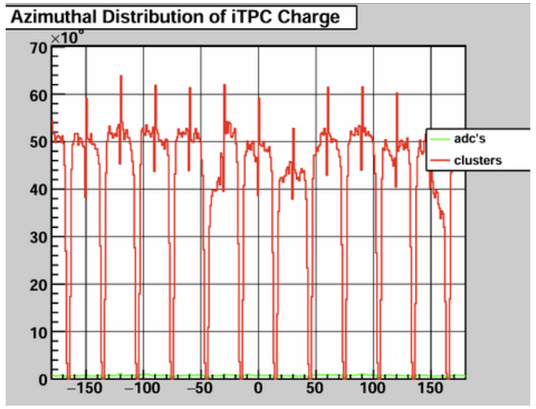
ADC signal of each pad after decoding, input for QA and clustering

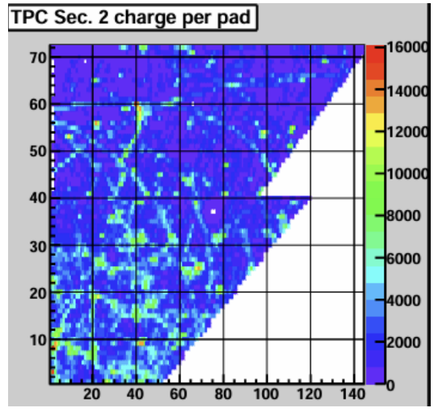



| Heartbeat Header | | |
|---|---|---|
| H | 6 | Hamming code |
| HP | 1 | Header parity bit (+ hamming code) |
| PT | 3 | Packet type |
| WN | 10 | Number of 10 bit words in data payload |
| HW | 4 | Hardware address id |
| CH | 5 | Channel address id |
| BX | 20 | Bunch-crossing counter (40MHz counter) |
| DP | 1 | Data payload parity bit |



# TPC Clustering

**Raw TPC digits → spatial hit candidates**

clusters already reveal the event structure and give a compact input for the event display and higher-level QA

