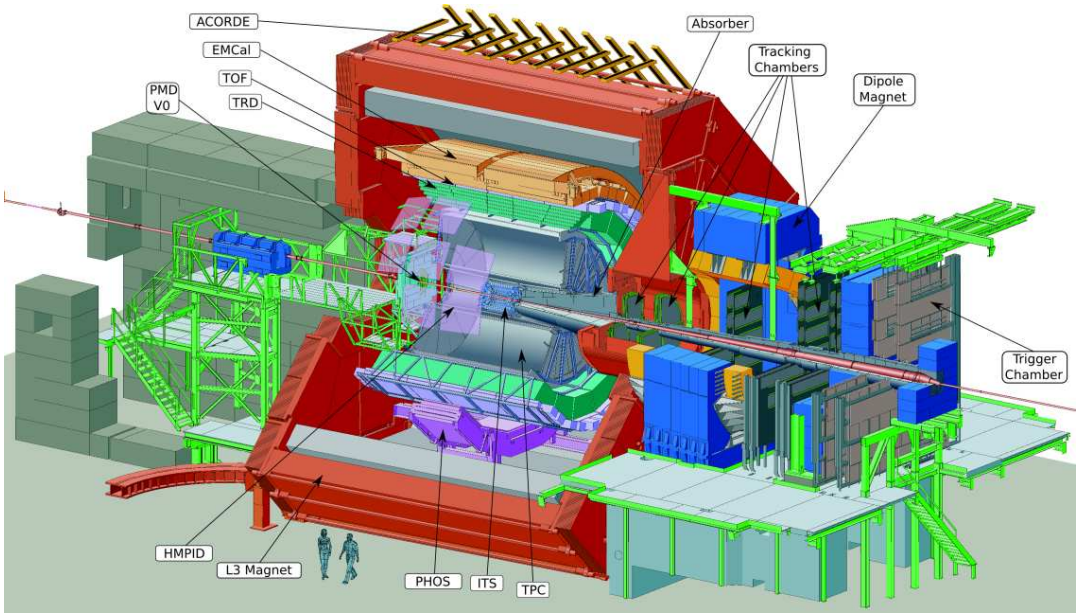
Data handling in the ALICE O2 event processing

Matthias Richter
for the ALICE collaboration

Department of Physics, University of Oslo
CERN - European Organization for Nuclear Research

Outline

- Online Data Processing in ALICE Run 3
- Data Organization
- Processing Pipelines
- Data Access
- Summary



ALICE Run 3 online data processing

A challanging goal ...

Inspect all Pb-Pb collisions at min bias rate of 50 kHz to provide access to rare physics probes. Continuous processing of data stream of 3.4 TByte/s.

- 14 different detector systems

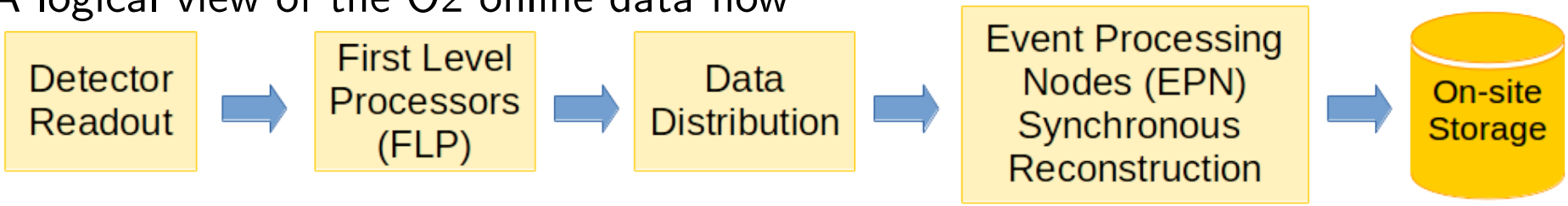
The ALICE O^2 facility: A combined online-offline system

- Distributed system with $\sim$ 1800 nodes
- Processing of a continuous, trigger-less stream of data segmented into so-called **Timeframes**
- Individual compute tasks are implemented as processes - *"Devices"*
- Unified data transport between processes using *FairMQ*

see M. Al-Turany, # ...

Online Data Processing in ALICE Run 3

A logical view of the O2 online data flow



- First Level Processor (FLP) - 270 FLPs in the facility
  - Hosts Common Readout Unit (CRU) to receive detector raw data
  - Creation of the metadata to facilitate navigation within data sets
  - Local synchronous processing tasks
  - SubTimeframe building
- Data Distribution - 500 GByte/s switching network
  - Service runnning on FLPs and EPNs and using the network
- Event Processing Node (EPN) - 1500 EPNs in the facility
  - Aggregation of full Timeframes for processing
  - Synchronous Timeframe reconstruction
  - Apply pipe-lined set of algorithms to data set

Requirements and Tasks for Data Handling

ALICE O^2 serves multiple tasks: raw data processing, reconstruction, data quality control, analysis

It's difficult to describe all data upfront; the system is under development in agile manner and new data structures will emerge

In a distributed, multi-process system we have to ensure:

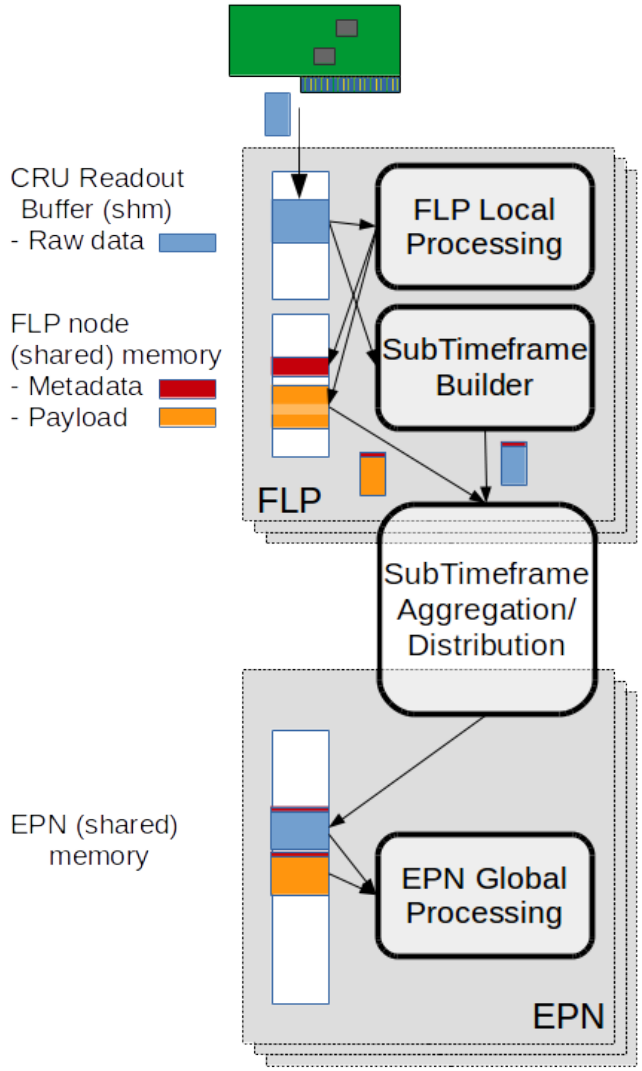
- Consistent annotation scheme for all types of data
- Annotate raw data without copying or parsing
- Allow for accumulation of data
- Support for addons to the annotation

Data Annotation

Goal: Identify all types of data in a unified way

raw data, temporary data in synchronous reconstruction, data published into asynchronous reconstruction/analysis

- Data are annotated with identifier and additional meta data
- Data annotations are flexible in size and content, and can be extended over lifetime of project
- CRU readout buffer is entirely reserved for the CRU-Readout communication, no reserved regions for annotations
- Avoid copies of data $\rightarrow$ Annotations sent as a separate block
- Currently using ability of grouping messages by the transport; this is no restriction, just a tool



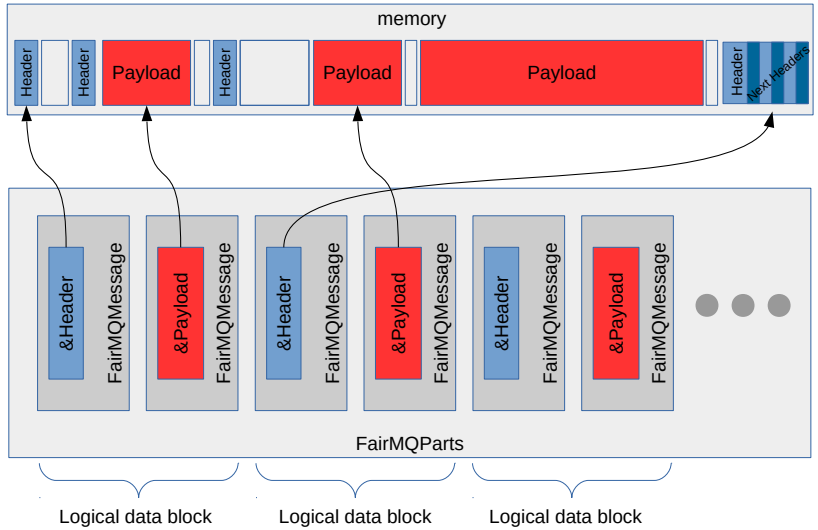
schematic view of data flow with inserted annotations

Data Model for Transport

A message is composed of multiple logical data blocks

A block is consisting of *header message* and *payload message*

Each individual message is a *FairMQMessage*



- All information necessary for the routing is in the *header message*
- *Header message* composed of variable number of headers in a *stack*
- Currently using *FaiMQParts* container to group messages
- *TODO: more on the header stack*

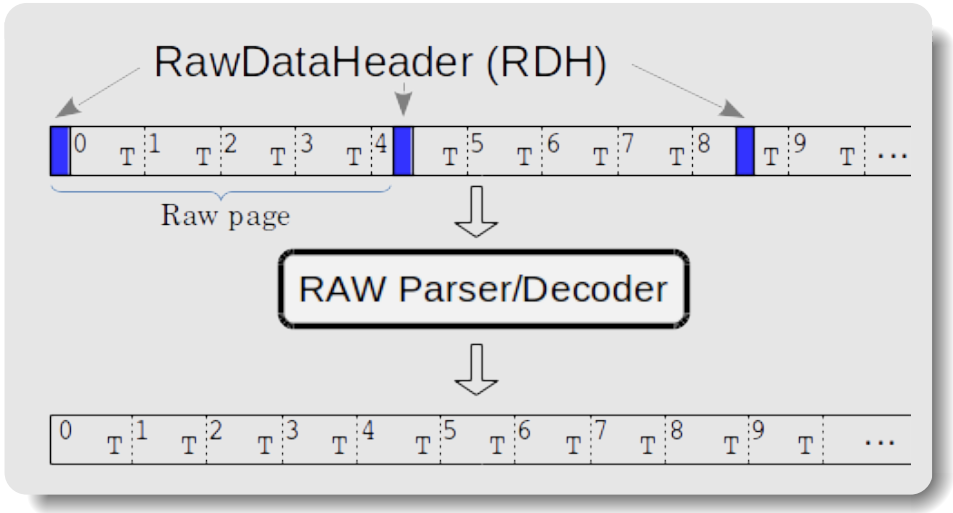
Data transfer between processes

Key Features:

- All data kept in memory, no intermediate files
- Unified protocol for data exchange
- Resource aware allocators
- Message queue transport and shared memory
- Serialization
- Abstraction to be independent from the underlying transport, no binding to a specific technology, simple exchange of transport method

Raw Data access

- Data written in *raw page format* into the FLP shared memory
- Each *raw page* starts with the `ReadoutDataHeader (RDH)`
- Raw pages have fixed size of 8 kB
- Data objects stay within pages or wrap over to next page



A generic RAW data parser/decoder, templated on `PageHeader` and `ElementType`

```
using PageHeaderType = o2::Header::RAWDataHeader;
// set up the decoder for some element type
RawParser<PageHeaderType, 8192, ElementType> RawParser;
RawParser parser(ptr, size);
for (const auto& element : parser) {
  // do something with element
}
```

- produces iterable view on data elements
- supports groups of data objects described by a `GroupHeader`
- support for multiple non-contiguous pages

$\Rightarrow$ can access data from the large variety of detectors in ALICE with a single simple implementation

Timeframe structure

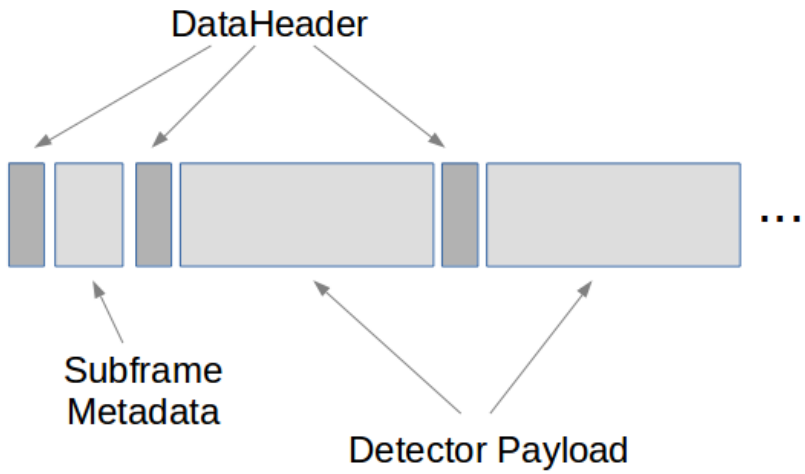
The *heartbeat trigger* structures the continuous data stream produced by the detector frontend.

- Heartbeat period is a multiple of the LHC orbit, natural time unit for timestamps in the online system.
- The timeframe duration is a power of two of the heartbeat period currently length is set to 256 HBs ($\sim$ 22 ms)

The data model allows to assemble data belonging to same Timeframe by simply accumulating data parts.

Format of the (sub) Timeframe:

- Sub timeframe is composed from multiple parts encapsulated in `FairMQMessage`
- The first pair describes general properties of the (S)TF



Data Processing Layer applied to the EPN processing

The Alice O^2 Data Processing Layer (DPL) is a framework providing declarative workflow definition

- Abstraction of a computation described by input, output and algorithm specification
- Declarative workflow based on data annotation,

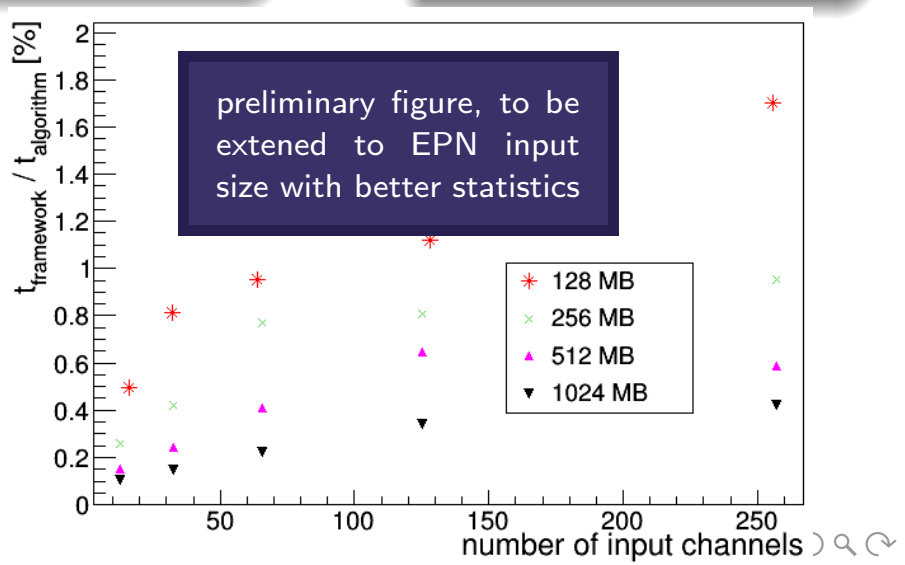
more on DPL in G. Eulisse, # ...

Define processors in terms of inputs, outputs and algorithm $\Rightarrow$ Connect processors in a workflow $\Rightarrow$ Deploy on computing resource

Checking the influence of the framework data handling to the total processing time

- Example process: TPC hardware cluster decoder as a fast algorithm
- Only simple data parsing and reformatting of data
- Tested for different input sizes and number of parts the input is split into

$\Rightarrow$ less than 2% consumed by framework



Process I/O

Framework provides specific I/O methods depending on type of data implemented as template specializations

Messageable types

- POD data structures
- trivially copyable types
- non-polymorphic
- no pointer members inside
- can be sent without copy

Serializable types

- serialize and flatten an object according to a schema
- always includes a copy
- ROOT serialization as example, many solutions on the market

```
auto podOutput = context.outputs().make<PODStruct>(OutputRef(...));
auto histogram = context.outputs().make<TH1>(OutputRef(...));
std::vector<SomeRootObjectType> data;
// ...
context.snapshot(OutputRef(...), data);
```

Framework allocates directly in the message memory and creates data annotation

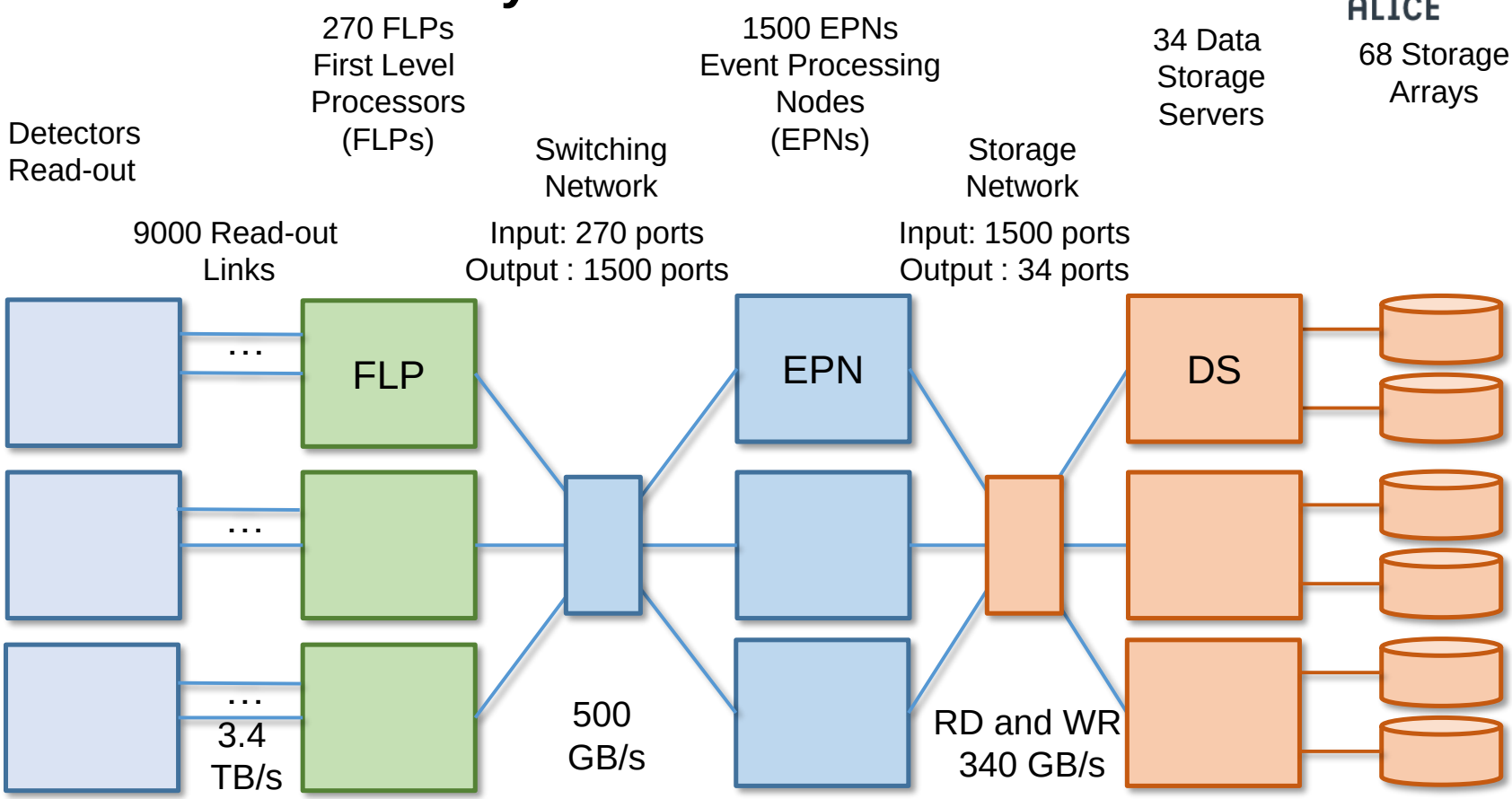
$\Rightarrow$ simple and coherent C++ I/O API

Summary

- Alice O^2 uses the ALFA software framework and message queue communication to distribute workload among many processes running on multiple compute nodes.
- ALICE O^2 implements efficient way of data annotation
- All types of data are consistently described within the annotation scheme
- The lightweight framework has a negligible contribution to the total processing time
- Data description is the basis for workflow-oriented definition of the compute topology

Backup slides

Hardware Facility



Unmodified raw data of all interactions - continuous read-out

Baseline correction, zero suppression cluster finder.

Data volume reduction by online tracking, Asynchronous processing

Data Storage: 1 year of compressed data

Further Performance Metrics