ALICE Analysis Trains on the Grid in Run 2

- **What the user does:** ROOT C++ Analysis Task code processing a single event
- **Abstraction layer:** run that code locally, on Grid, on PROOF
- **Input data:** reco (ESD + friends) or analysis (AOD + deltaAOD) → TTrees + calibration data
- **Analysis repo:** all user code on alisw/AliPhysics, built centrally, on CVMFS
- **Analysis trains:** read once, process many times, benefit from common processing
- **Operations:** user tasks assembled in wagons by operators: reduce human error
Longer, faster trains on Analysis Facilities in Run 3

- **Dedicated Analysis Facilities**: process 5 PB every 12 hours each, on average 100 GB/s
- **Retain concepts that work**: analysis trains • centralized code • abstraction framework

**What slows down our analysis the most?**

- **storage performance**: forget the Grid: provide for 2 or 3 analysis facilities, **20k cores** each, fast local storage and network
- **decompression**: not limiting in Run 2, bottleneck in Run 3: use better compression algorithms and parallelize
- **deserialization**: flat data structures with numbers, cross-reference using numerical indices

**Writing not an issue**: output = small trees/histograms (< 100 MB per analysis)

**ANALYSIS FACILITIES**

Dedicated and dense, do more with less: aim at > 95% efficiency
Development areas

**Analysis facilities**
- Only analyze local data
- Fast local storage and network
- Allow inter-nodes communication

**Data format**
- Low deserialization cost
- Efficient in-memory store
- Optimized decompression

**Workflow handling**
- Allow for non-linear workflows
- Nodes subscribe to data
- Use network and shared memory

**User-facing API**
- Reuse standard interfaces
- Declarative paradigm
- Optimize common operations
Development areas

**Analysis facilities**
- Only analyze local data
- Fast local storage and network
- Allow inter-nodes communication

**Data format**
- Low deserialization cost
- Efficient in-memory store
- Optimized decompression

**Workflow handling**
- Allow for non-linear workflows
- Nodes subscribe to data
- Use network and shared memory

**User-facing API**
- Reuse standard interfaces
- Declarative paradigm
- Optimize common operations
ALICE has a test Analysis Facility at GSI (Darmstadt, DE). Run 3 requirements are fulfilled and allows for testing, while currently running Run 2 jobs as a Tier-2.

## Tests on the current GSI facility

<table>
<thead>
<tr>
<th>test conditions</th>
<th>Run 2 analysis framework</th>
<th>pure data transfer</th>
</tr>
</thead>
<tbody>
<tr>
<td>data on Lustre uniformly distributed over the OSSes</td>
<td>2500 cores process data at an aggregate speed of 32 GB/s</td>
<td>one node/one OSS: 1.2 GB/s, 1500 parallel nodes: 600 GB/s</td>
</tr>
</tbody>
</table>

Current network and storage can easily sustain transfer rates way above our 100 GB/s limit, we don’t have an hardware limitation: analysis framework and user code are our only limits (as seen by the “old” framework speed).

Dedicated talk:🔗 A prototype for the ALICE Analysis Facility at GSI (Thu, 12:00, T8)
Development areas

**Analysis facilities**
- Only analyze local data
- Fast local storage and network
- Allow inter-nodes communication

**Data format**
- Low deserialization cost
- Efficient in-memory store
- Optimized decompression

**Workflow handling**
- Allow for non-linear workflows
- Nodes subscribe to data
- Use network and shared memory

**User-facing API**
- Reuse standard interfaces
- Declarative paradigm
- Optimize common operations
**Timeframes**

**Base unit for ALICE Run 3 is the **timeframe**: no reco events but **vertices and tracks****

<table>
<thead>
<tr>
<th><strong>timeframe length</strong></th>
<th><strong>timeframe size</strong></th>
<th><strong>triggerless</strong></th>
</tr>
</thead>
</table>
| ~23 ms worth of data taking  
(~1k Pb-Pb MB collisions) | First Level Processors: ~10 GB  
live reco: ~2 GB → **AOD**: ~1 GB | continuous data taking  
without triggers |

**Data flow from the detector to analysis objects**

- ALICE
  - detector readout and first level farms
  - synchronous reco
    - (live, on-site)

- Compressed Timeframes

- Event Summary Data

- Analysis Object Data
  - Ancillary AOD 1
  - Ancillary AOD 2

- asynchronous reco

Each format (CTF, ESD, AOD) features a different flavor of **timeframe**

shipped to the Analysis Facilities

Dario.Berzano@cern.ch - CHEP 2018 - The ALICE Analysis Framework for LHC Run 3
New data format should reduce as much as possible the cost of deserialization: some generality will be lost for the sake of improved speed

- **Simple and flat:** numbers only (no classes), tables cross-referenced via numeric indices

- **Columnar:** immutable base format, efficiently grow/shrink and vectorize

- **Chunked:** single timeframe is large (~1 GB): store in chunks to allow parallel processing

- **Memory efficient:** zero copy, zero size for null values, recompute may be better than storing

- **Data not restructured:** disk → memory → network should use similar representations
We have been experimenting with Apache Arrow: in-memory columnar data format targeting memory efficiency and cross-language compatibility

- **Leverages vectorization** and fits our other requirements
- **Units**: data organized in Tables, made of immutable Columns. Columns shared among tables (no copy)
- **Memory management**: Columns backed by Buffers, which allow for custom Memory Pools
- **Meant for interoperability**: allows for data exchange within the Apache ecosystem and outside, widely supported
- **Fits the ALICE Run 3 data model** based on message passing

*Prototype based on Arrow: other solutions being investigated too*
## Development areas

<table>
<thead>
<tr>
<th>Analysis facilities</th>
<th>Data format</th>
<th>Workflow handling</th>
<th>User-facing API</th>
</tr>
</thead>
<tbody>
<tr>
<td>Only analyze local data</td>
<td>Low deserialization cost</td>
<td>Allow for non-linear workflows</td>
<td>Reuse standard interfaces</td>
</tr>
<tr>
<td>Fast local storage and network</td>
<td>Efficient in-memory store</td>
<td>Nodes subscribe to data</td>
<td>Declarative paradigm</td>
</tr>
<tr>
<td>Allow inter-nodes communication</td>
<td>Optimized decompression</td>
<td>Use network and shared memory</td>
<td>Optimize common operations</td>
</tr>
</tbody>
</table>
**ALICE O² framework and Data Processing Layer**

**ALICE O²: Offline/Online**
Offline/Online share the same processing framework: do the same for Analysis

**Message-passing parallelism**
Processes (devices) exchanging data via ZeroMQ/shared memory: user code less error prone

**Data Processing Layer**
O² component allowing to specify the data flow (how inputs/outputs are connected) declaratively

---

**Example of analysis workflow**
Boxes are O² devices (independent processes)

- **Unzip**
- **Preprocess**
- **Task1**
- **Task2**
- **Task3**
- **Merger1**
- **Output1**
- **Merger2**
- **Output2**
- **Merger3**
- **Output3**

Uncompress in parallel on different processes

Dedicated talk:🔗 Evolution of the ALICE Software Framework for LHC Run 3 (Tue, 14:15, T5)
Development areas

**Analysis facilities**
- Only analyze local data
- Fast local storage and network
- Allow inter-nodes communication

**Data format**
- Low deserialization cost
- Efficient in-memory store
- Optimized decompression

**Workflow handling**
- Allow for non-linear workflows
- Nodes subscribe to data
- Use network and shared memory

**User-facing API**
- Reuse standard interfaces
- Declarative paradigm
- Optimize common operations
User-facing API, or the new ALICE Analysis Task

- **Current analysis:** very simple abstraction, only one degree of freedom: user writes a function for “processing” an event (whatever “processing” means)

- **RDataFrame-based prototype:** more declarative and concise, user relinquishes strict control to the framework for some automatic optimization (lazy execution, IMT…)

**ROOT::RDataFrame**

```cpp
ROOT::RDataFrame d(input).Filter(criteria).Foreach(\lambda)
```

independent from source (ROOT Trees, Arrow…) • has Implicit Multithreading capabilities

ALICE contribution to RDataFrame allowing for Apache Arrow Tables as source

[github link](https://github.com/root-project/root#1712) (merged)

Current prototype based on RDataFrame: other solutions being investigated as well
Analysis framework prototype: stream Arrow data

- **Apache Arrow**
  Split/compose tables, serialize, deserialize

- **O² Data Processing Layer**
  Stream only subscribed columns to tasks, manage parallel decompression

- **ROOT::RDataFrame**
  User writes a lambda completely independent from the used data format and transport backends

Analysis framework prototype is ready for testing
Power users can now start writing analysis code to refine framework and data format

Complete example available: o2ArrowColumnStreamer
Analysis framework prototype: decompress in parallel

- **LZ4 default compression**
  Recommended option

- **Single reader**
  20 MB-large blocks

- **Time-based parallelism**
  DPL parallelizes by pushing timeframes in round robin to automatic task clones

Tests run on an old 2014 Ivy Bridge EP-based CPU: Intel® Xeon® E5-2697 v2 @ 2.70GHz

<table>
<thead>
<tr>
<th>simple read test</th>
<th>LZ4 decompression</th>
<th>on a single node</th>
</tr>
</thead>
<tbody>
<tr>
<td>aggregated max 2.2 GB/s on single node (upper limit)</td>
<td>1x → 560 MB/s</td>
<td>4 cores used for decompressing: use the rest for analysis</td>
</tr>
<tr>
<td>4x → 2.2 GB/s (plateau)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Complete example available: o2ParallelDecompressor
Easily exploit multiple dimensions of parallelism: the most relevant of them come for free from the framework, without any extra knowledge by the user

- **Decompression and analysis tasks using multiple processes**
  Each read timeframe is sent to multiple decompressor processes via message passing

- **ROOT’s Implicit Multithreading and Apache Arrow chunks**
  Each Arrow timeframe is large enough to make it worth to divide it in chunks: RDataFrame is capable of parallelizing over them if desired (IMT on)

- **Single instruction, multiple data**
  Arrow’s columnar in-memory format makes it possible to vectorize certain operations (in some cases this happens automatically at compile time)

- **Repeat the same topology for every input file**
  For each input file, spawn one of the whole aforementioned topologies
Conclusions

• **Early stages: we needed a full stack working prototype to involve power users**
  Current ALICE analysis tasks cannot be easily converted, we need to try something new

• **Compartmentalize user code and data format/backend**
  Optimized “declared” operations (filtering), change framework without affecting user code

• **Built on top of the unifying O² processing framework**
  Development is going in parallel with the Data Processing Layer

• **Retain the general successful idea of the ALICE analysis trains model**
  It worked because it factored out critical parts; we want to factor out even more
Thanks!