

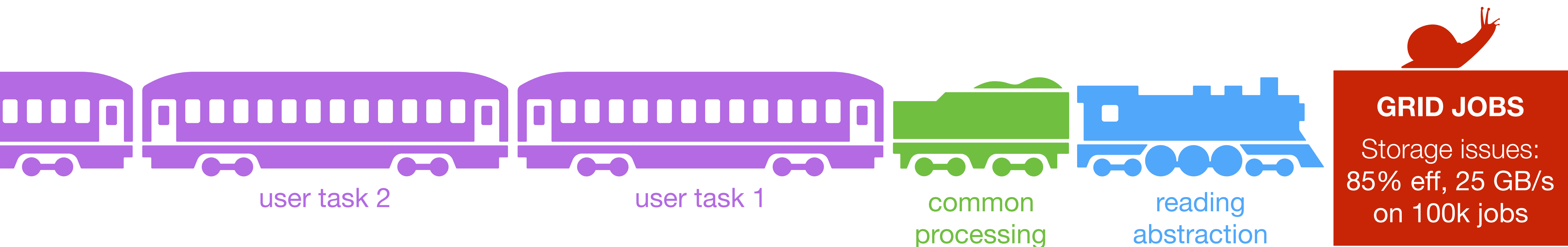
The ALICE Analysis Framework for LHC Run 3

Dario Berzano

CERN and INFN Torino · for the ALICE collaboration

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 github.com/alisw/AlPhysics, 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

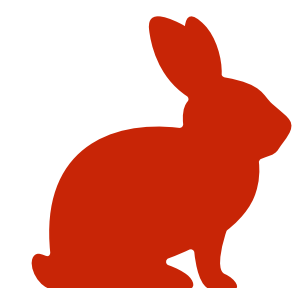
decompression

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)



more user tasks

more common processing

faster storage and reading abstraction

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

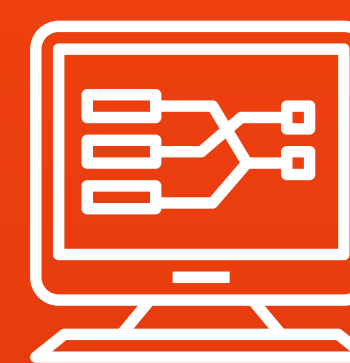


Low deserialization cost

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

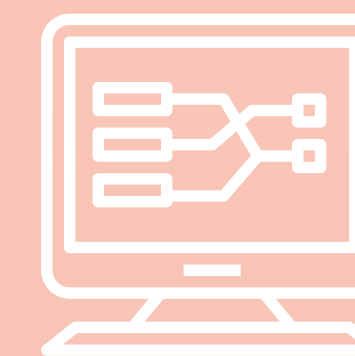


Low deserialization cost

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

Analysis Facility test setup at GSI: benchmarks

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

test conditions

data on Lustre uniformly distributed over the OSSes

Run 2 analysis framework

2500 cores process data at an aggregate speed of 32 GB/s

pure data transfer

one node/one OSS: 1.2 GB/s,
1500 parallel nodes: 600 GB/s

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).

600 GB/s

average aggregated upper limit on the GSI analysis facility

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

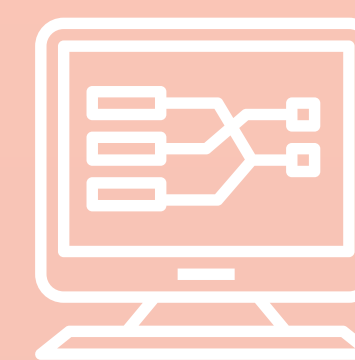


Low deserialization cost

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

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

timeframe length

~23 ms worth of data taking
(~1k Pb-Pb MB collisions)

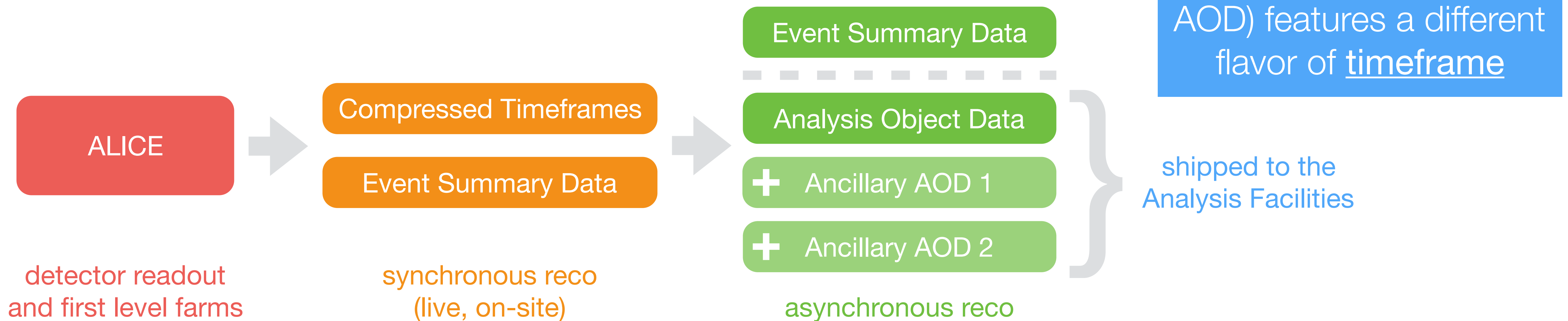
timeframe size

First Level Processors: ~10 GB
live reco: ~2 GB → AOD: ~1 GB

triggerless

continuous data taking
without triggers

Data flow from the detector to analysis objects



Analysis data format: requirements

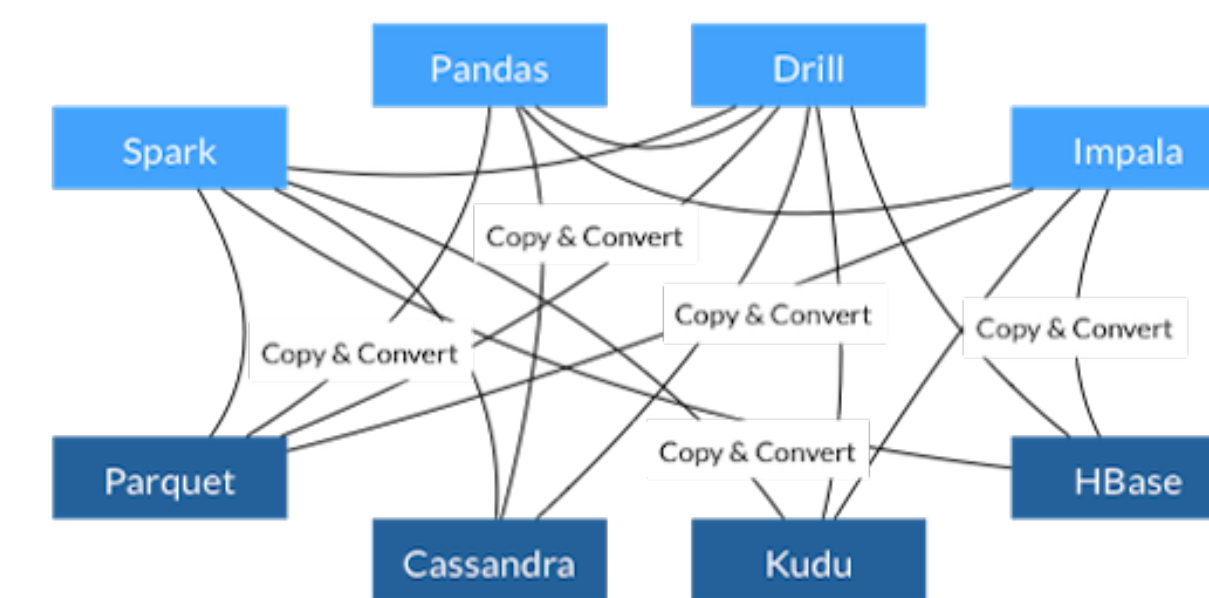
New data format should reduce as much as possible the cost of deserialization: some generality will be lost for the sake of improved speed

- **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

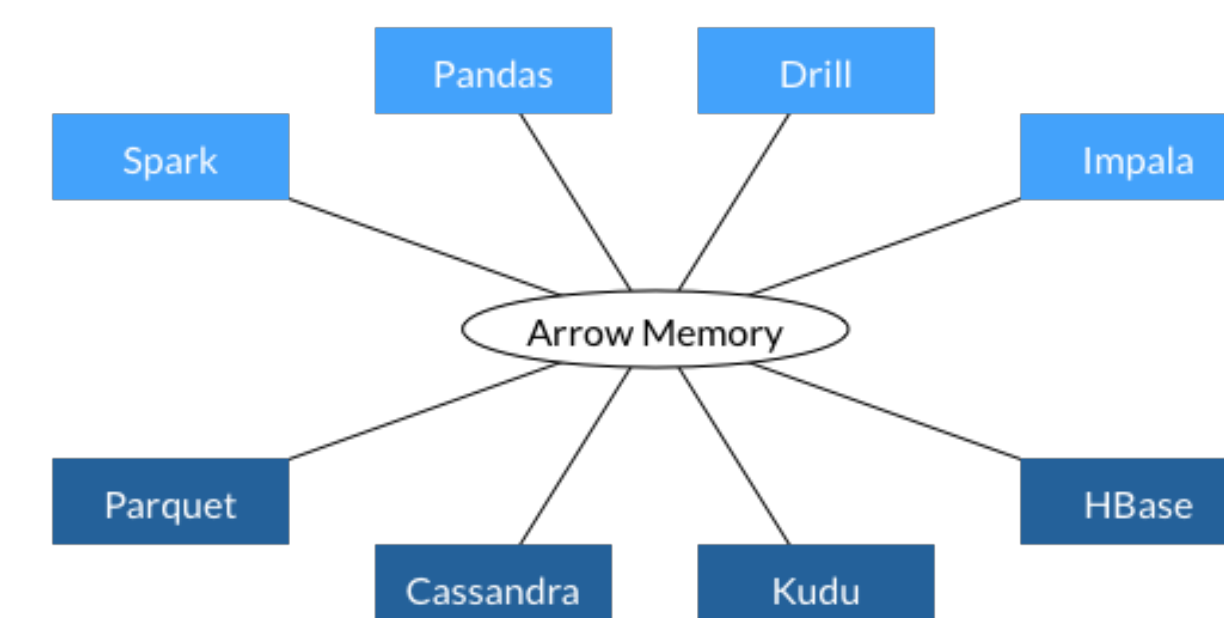
We have been experimenting with Apache Arrow: in-memory columnar data format targeting memory efficiency and cross-language compatibility

- **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



Apache Arrow
arrow.apache.org



Development areas

Analysis facilities



Only analyze local data

Fast local storage and network

Allow inter-nodes communication

Data format

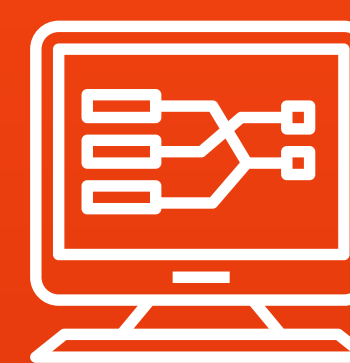


Low deserialization cost

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 O² framework and Data Processing Layer

ALICE O²: Offline/Online

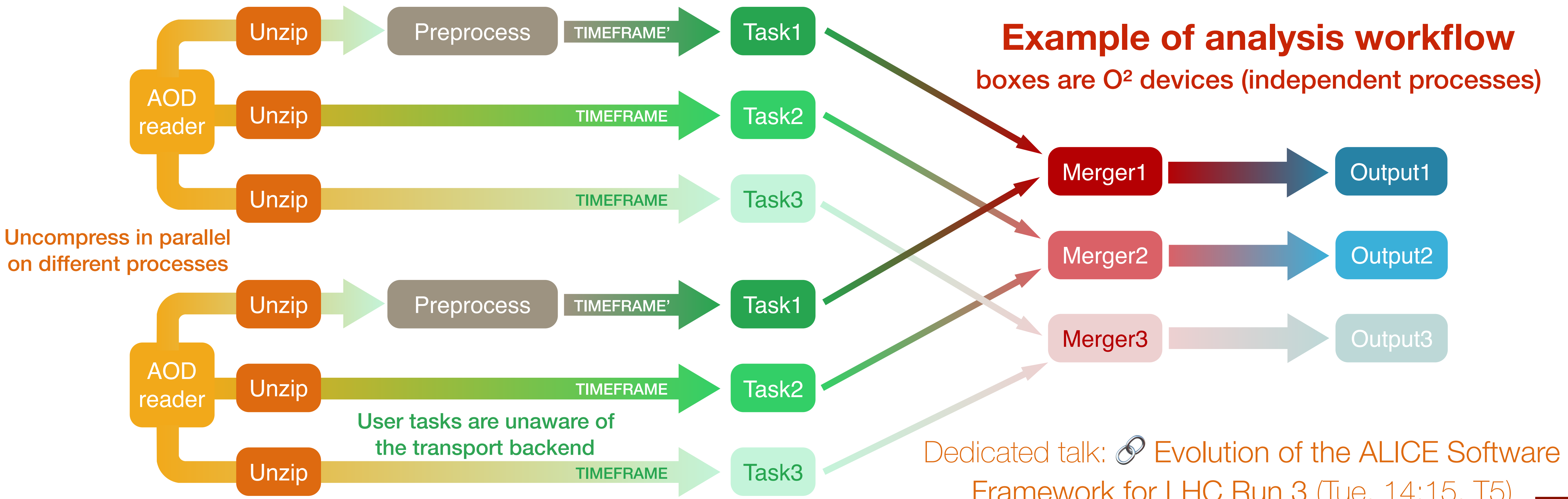
Offline/Online share the same processing framework: do the same for Analysis

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



Development areas

Analysis facilities

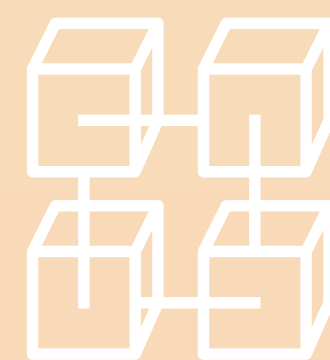


Only analyze local data

Fast local storage and network

Allow inter-nodes communication

Data format

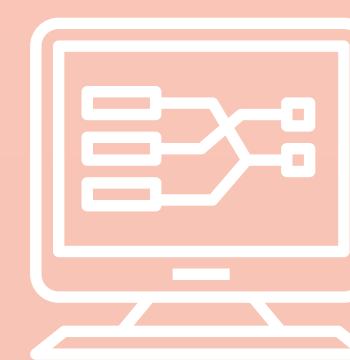


Low deserialization cost

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

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

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

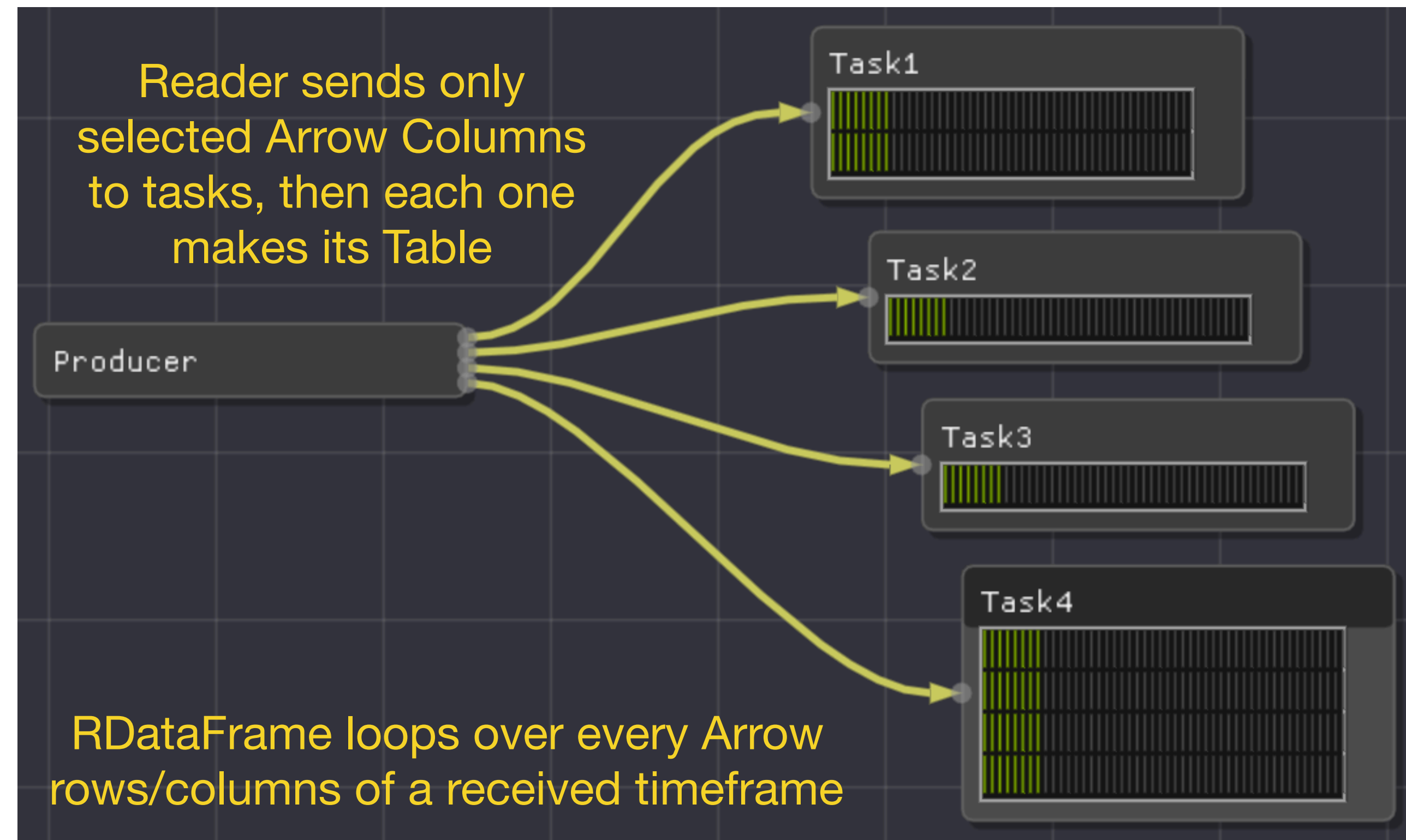
ALICE contribution to RDataFrame allowing for Apache Arrow Tables as source

 [root-project/root#1712](https://github.com/root-project/root/pull/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



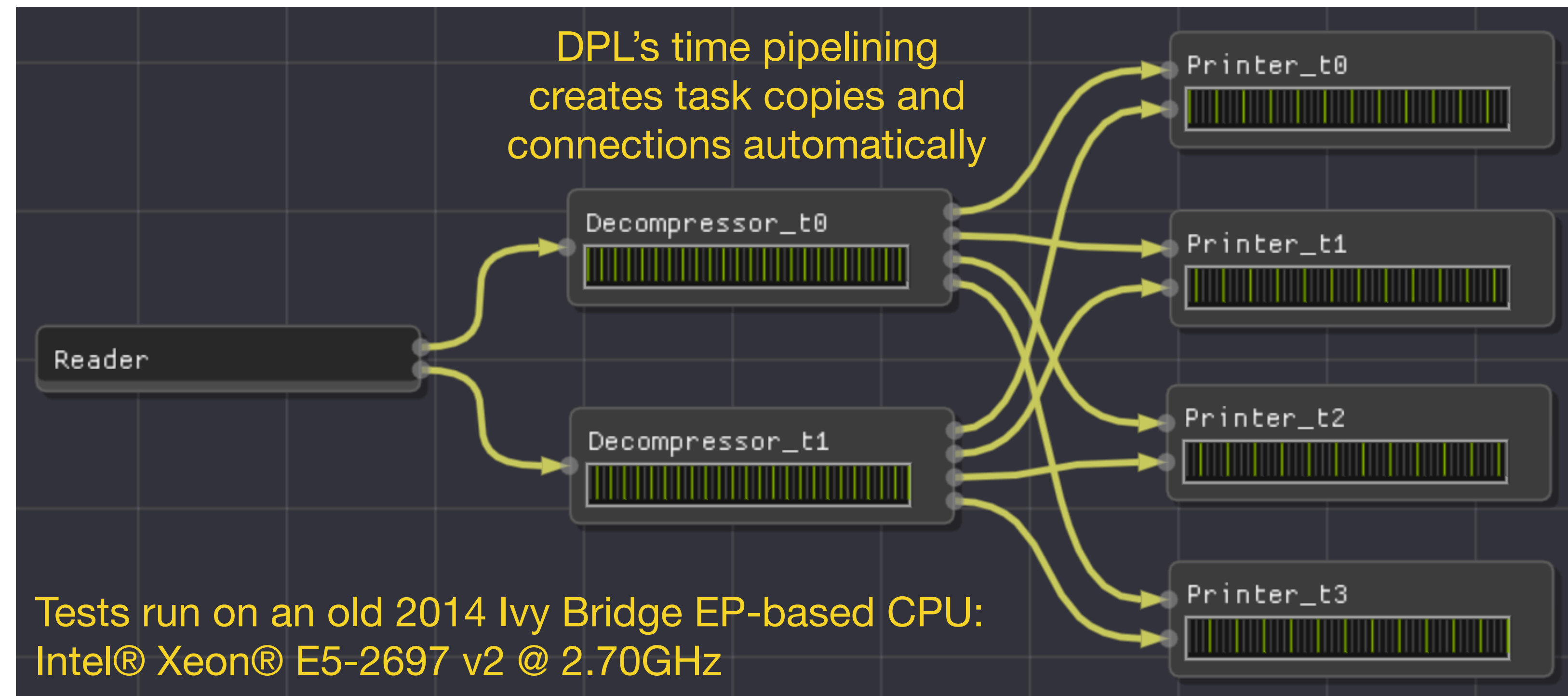
Analysis framework prototype is ready for testing

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



simple read test

aggregated max 2.2 GB/s
on single node (upper limit)

LZ4 decompression

1x → 560 MB/s
4x → 2.2 GB/s (plateau)

on a single node

4 cores used for decompressing:
use the rest for analysis

Complete example available: [o2ParallelDecompressor](#)

Exploiting dimensions of parallelism

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**

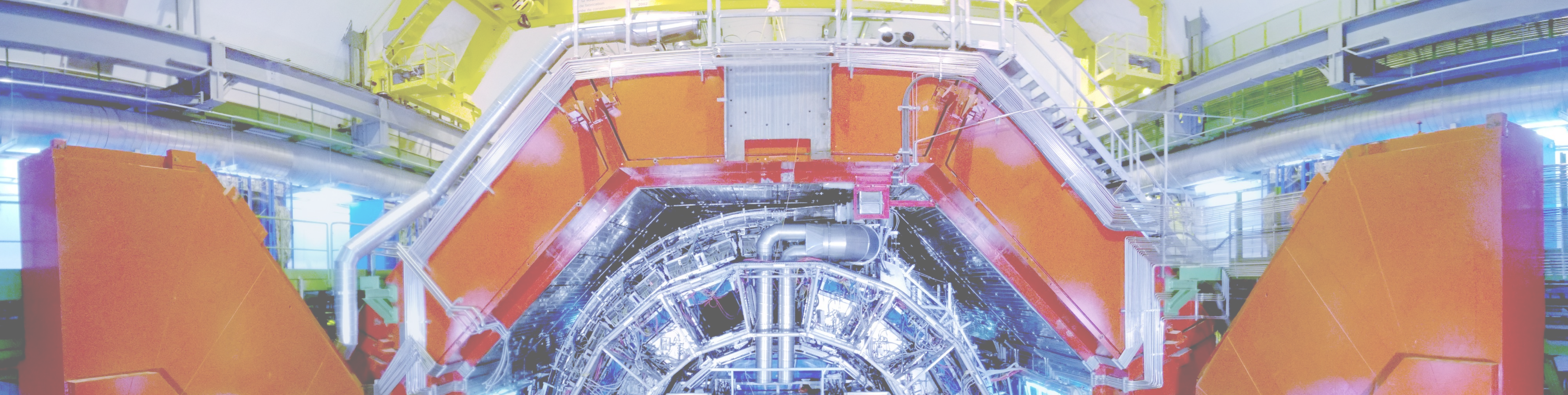
Arrow's columnar in-memory format makes it possible to vectorize certain operations (in some cases this happens automatically at compile time)

- **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!

