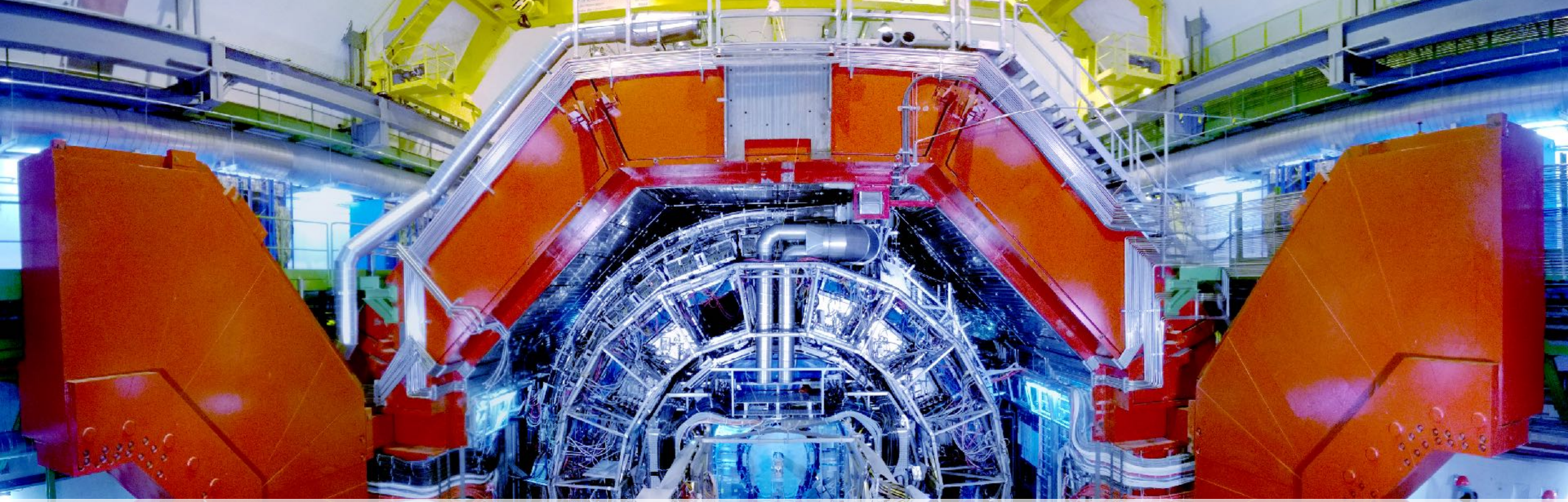


ALICE Analysis in Run 3

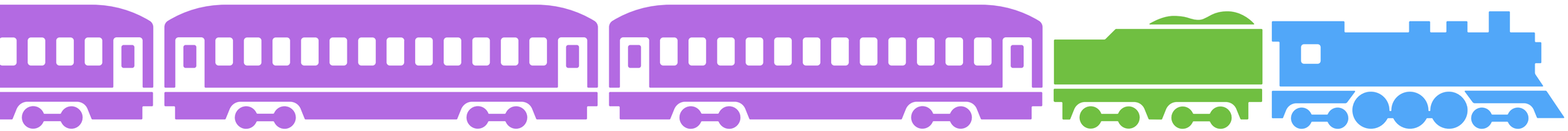
Dario Berzano

for the ALICE collaboration

Computing in High-Energy and Nuclear Physics 2018 - Sofia, July 9, 2018

ALICE Analysis Trains 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:** reconstructed (ESD + friends) or analysis-specific (AOD + deltaAOD) → TTrees
- **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



5 MB/s

avg per train job processing rate on Grid

We need to do much better in Run 3

- **Requirement for Run 3:** estimated 5 PB every 12 hours, on average 100 GB/s
- **Retain concepts that work:** analysis trains • centralized code • abstraction framework

What slows down our analysis the most?

slow storage

forget the Grid: provide for 2 or 3 analysis facilities, 50k cores each, local storage, fast network

decompression

use better compression algorithms and levels, parallelize decompression

deserialization

flat data structures with numbers, cross-reference using numeric indices

more coarse-grained user tasks

more common processing

faster storage and reading abstraction

100 GB/s

required **overall** processing rate on Analysis Facilities

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 interface

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 interface

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

current analysis framework

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

pure data transfer

1500 cores provide 1.2 GB/s per core: 1.8 TB/s aggregated

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).

1.8 TB/s

avg aggregated **setup limit** on the GSI analysis facility

See dedicated talk by S. Fleischer, K. Schwarz [link]

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 interface

Reuse standard interfaces

Declarative paradigm

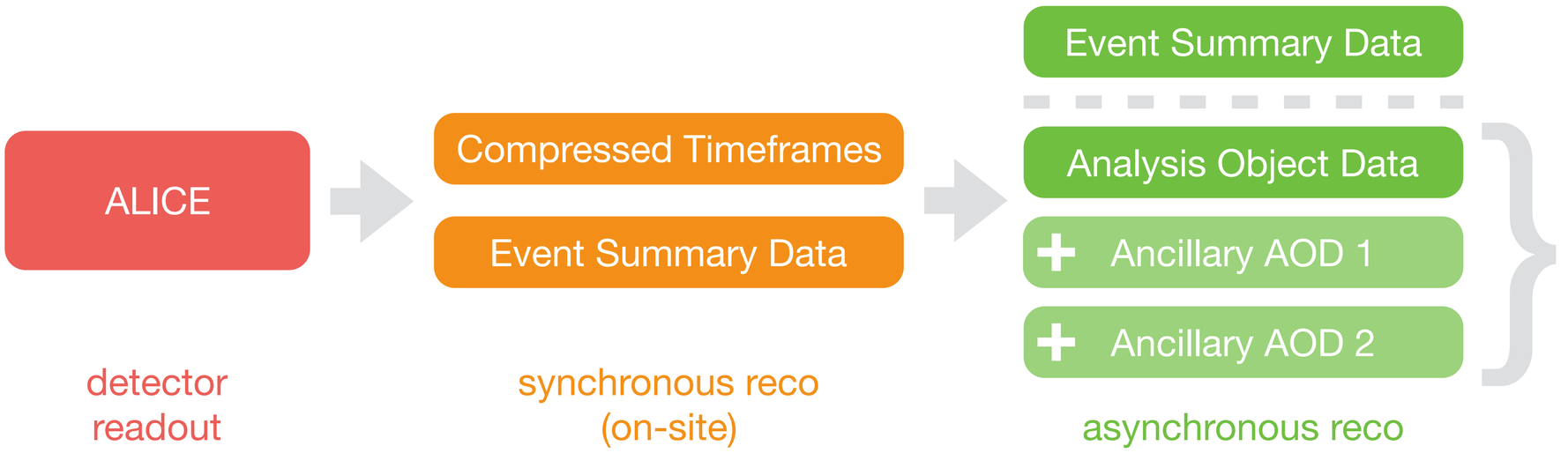
Optimize common operations

Timeframes

Base processing unit for ALICE in Run 3 is the timeframe and no longer events

length	size	triggerless
~23 ms worth of Physics (~1k Pb-Pb MB collisions)	~10 GB readout → ~2 GB after live reconstruction	continuous data taking without triggers

Data flow from the detector to analysis objects



each format (CTF, ESD, AOD) features a different flavor of timeframes

shipped to the Analysis Facilities

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

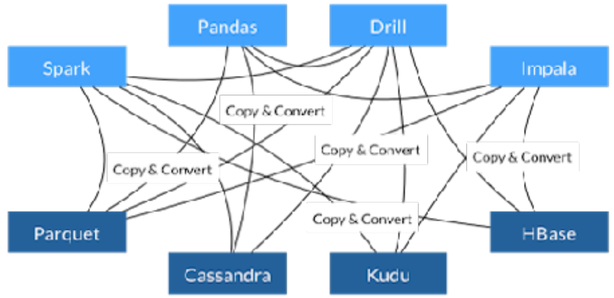
- **Simple, flat:** numbers only (no classes), use tables, cross-reference via numeric indices
- **Columnar:** SoA in-memory structure for better growing/shrinking and vectorization
- **Extensible:** base format is immutable, but easily extensible because it's SoA
- **Chunked:** a single timeframe can be divided in smaller units processable in parallel
- **Zero size for null objects:** filtered-out fields do not use RAM memory
- **Recompute, don't store:** do not store everything because recomputing may be cheaper
- **No data restructuring:** disk → memory → network should use similar representations

Apache Arrow

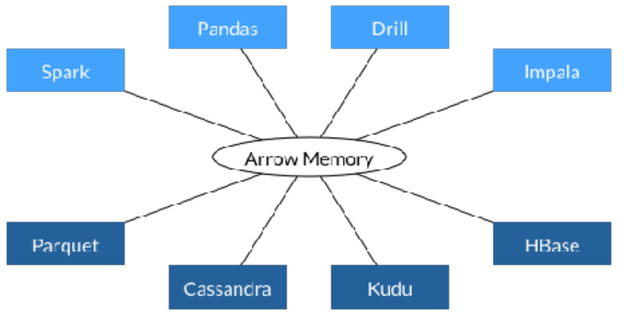
We have been experimenting with Apache Arrow: in-memory, language independent columnar data format. Has an on-disk companion too (Apache Parquet)

- **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 interface

Reuse standard interfaces

Declarative paradigm

Optimize common operations

ALICE O² framework and Data Processing Layer

ALICE O²: Offline/Online

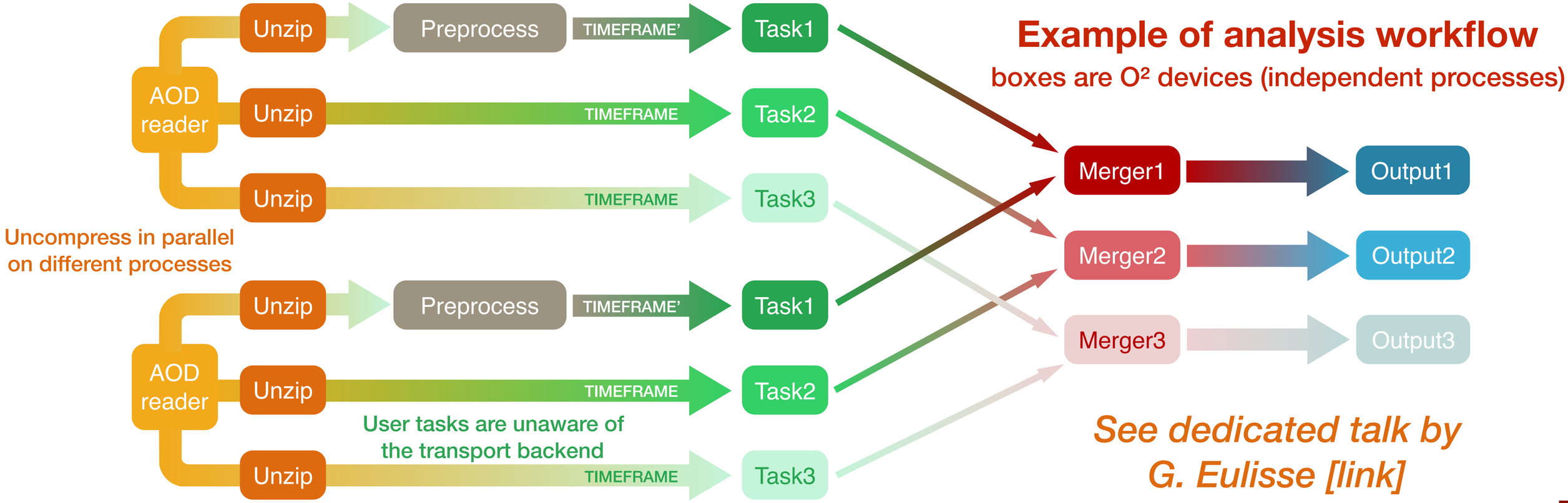
Offline/Online share the same processing framework: do the same for Analysis

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



Example of analysis workflow

boxes are O² devices (independent processes)

See dedicated talk by G. Eulisse [link]

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 interface

Reuse standard interfaces

Declarative paradigm

Optimize common operations

User interface: 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)
- **A more declarative approach:** allow user to specify how to filter data and preprocess it first, and then where to store the results, in a compact way
- **Transparent optimization:** if user relinquishes strict control over the program flow, we can optimize behind the scenes (better queries, lazy execution…)

ROOT::RDataFrame

```
ROOT::RDataFrame d(input).Filter(criteria).Foreach(λ)
```

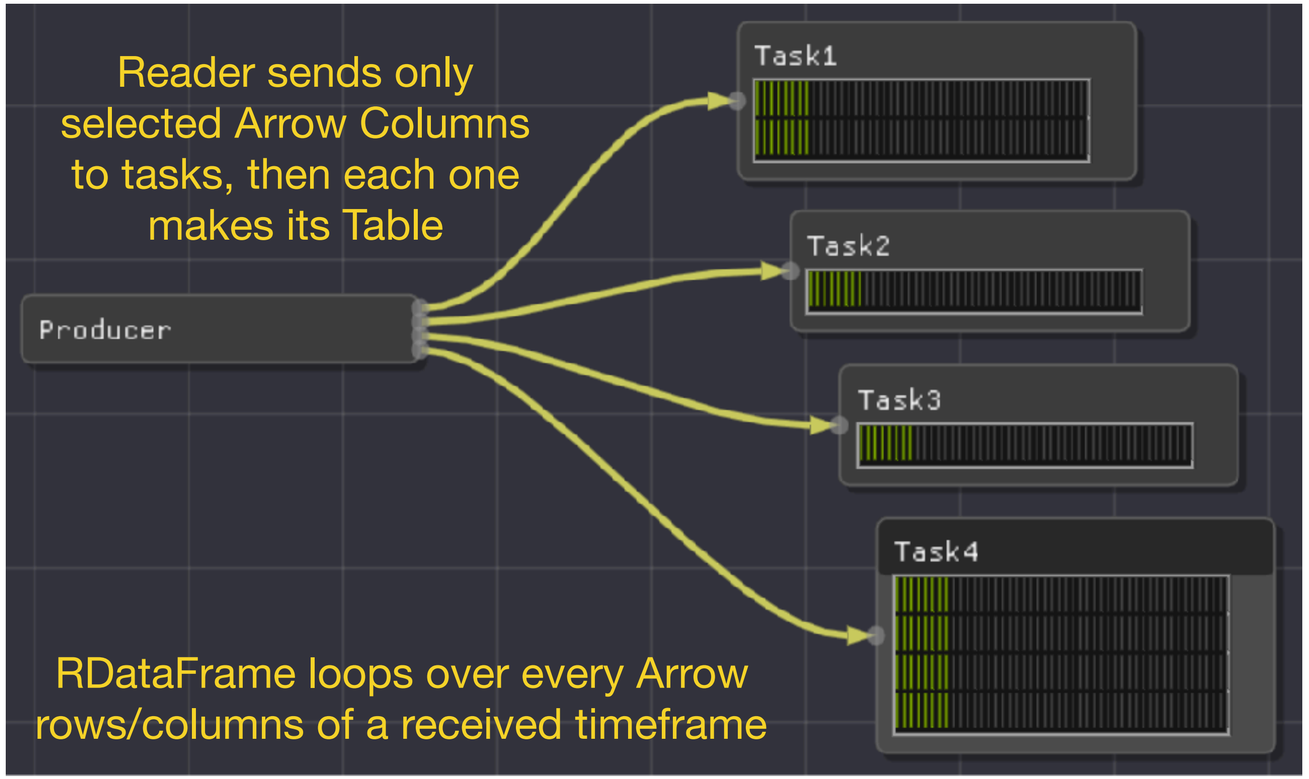
independent from source (ROOT Trees, Arrow…) • has Implicit Multithreading capabilities

ALICE contribution to TDataFrame transparently allowing Apache Arrow tables as source

root-project/root#1712

Analysis framework prototype: stream Arrow data

- **Apache Arrow + Parquet**
  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



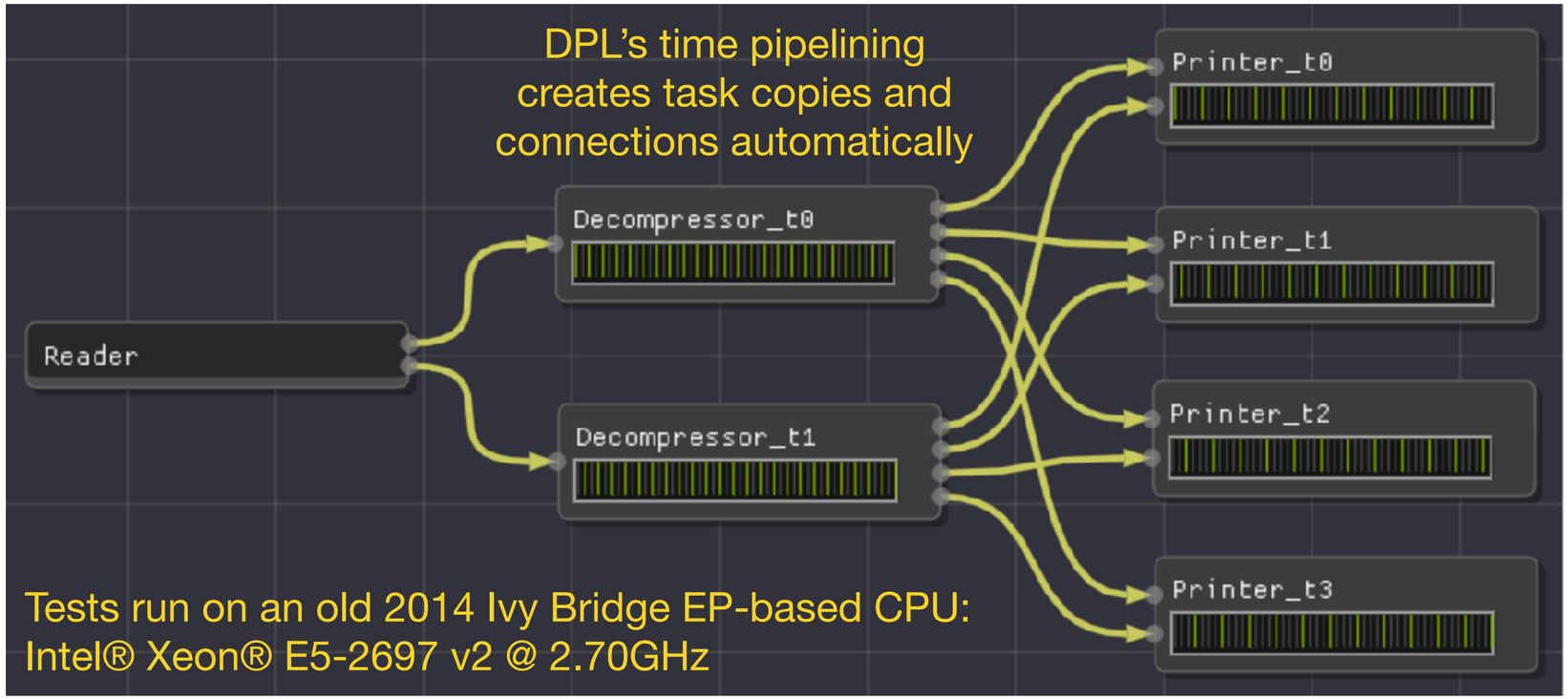
Analysis framework prototype is ready for testing

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

- **Decompress 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)
- **One topology per 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
- **Strategies to restrict user code domain for better performances/stability**
  Independent processes, declarativeness, expose as little as possible (TDataFrame fits)
- **Separate user code from data format and underlying backend**
  We want to experiment with data formats other than Arrow 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!