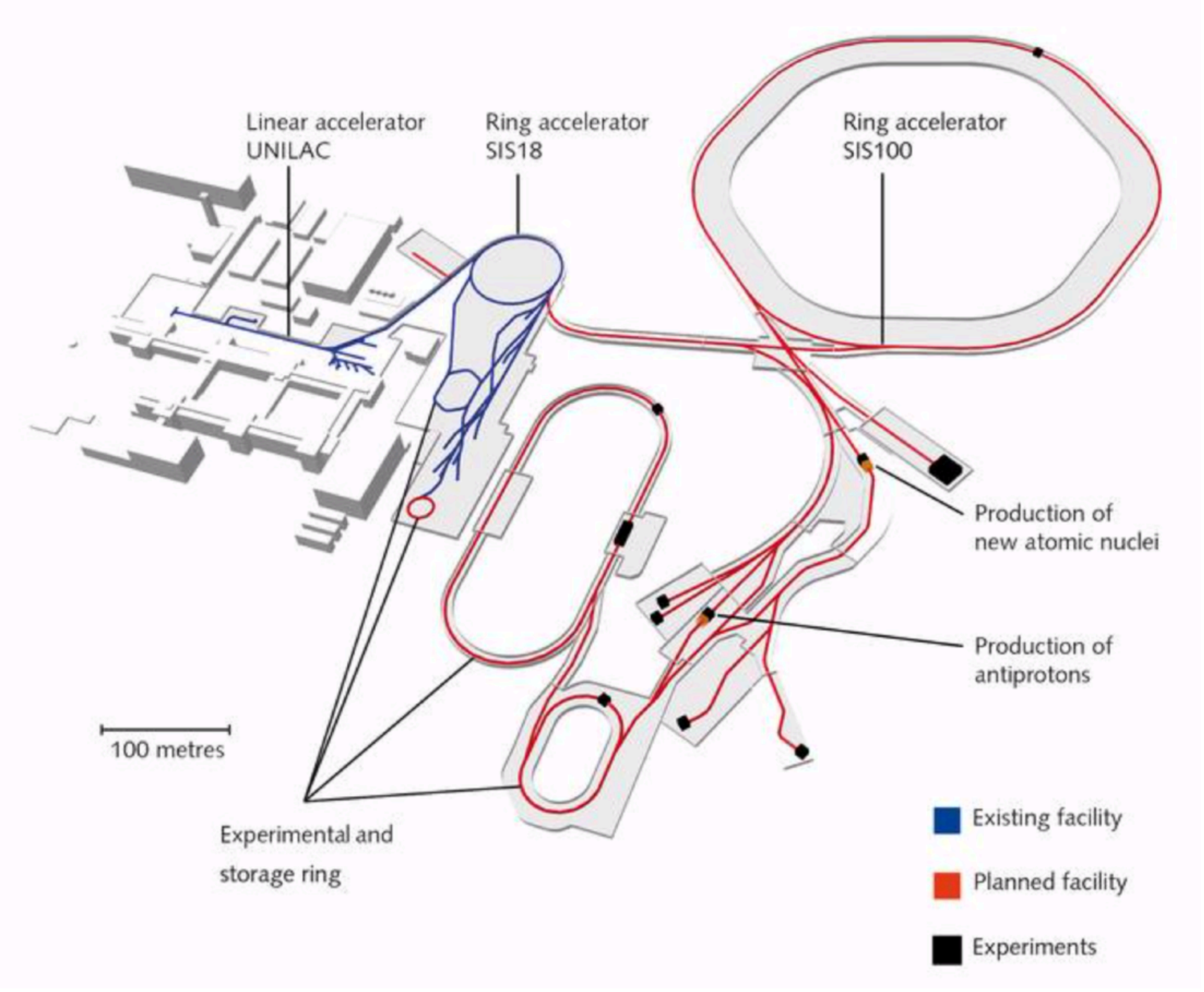
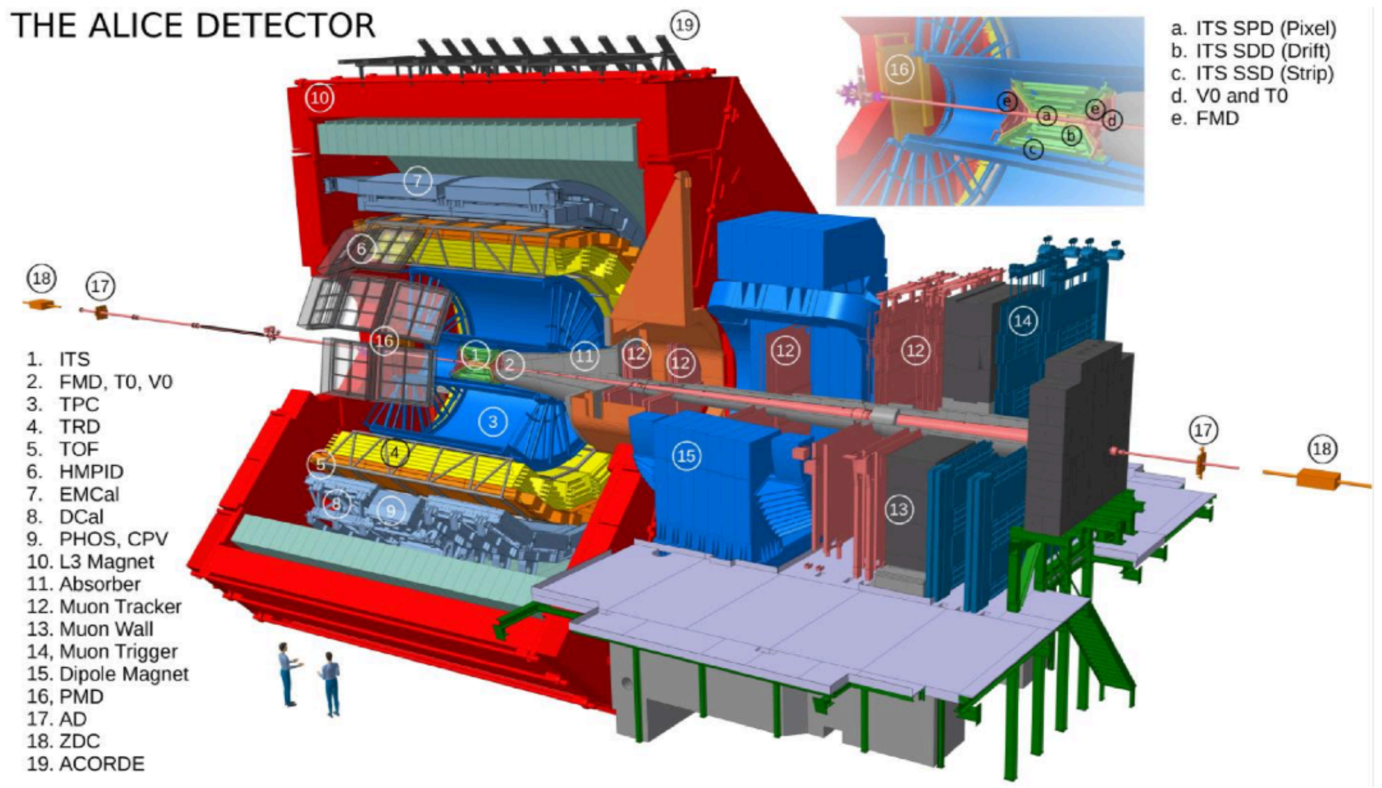
ALICE SOFTWARE FRAMEWORK FOR RUN 3

Giulio Eulisse - CERN

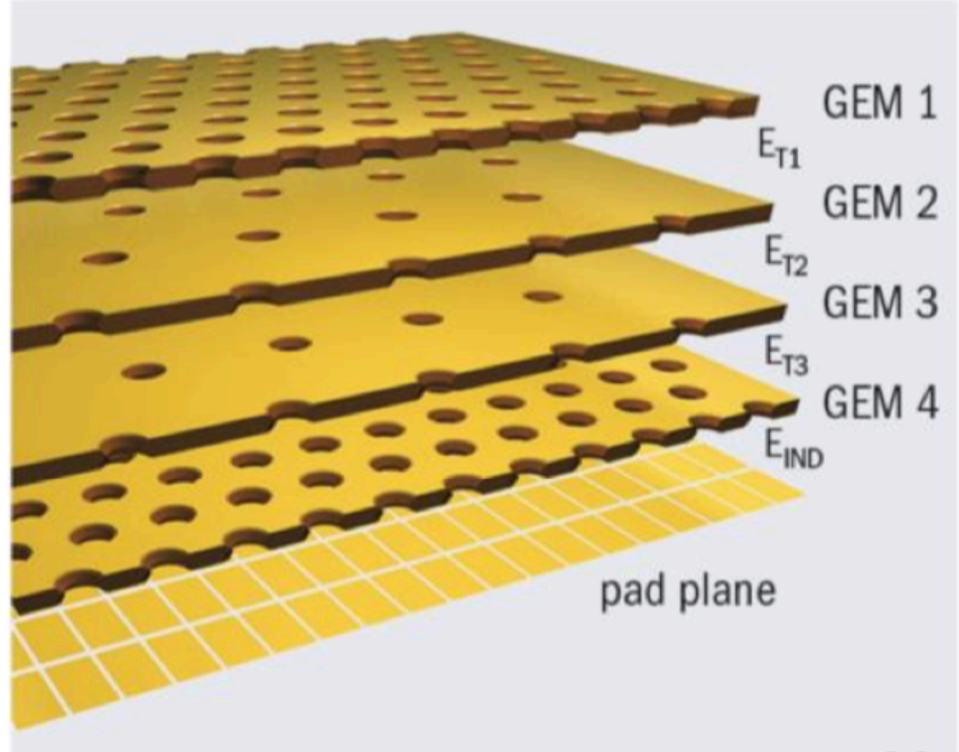
ALICE – FAIR FRAMEWORK COLLABORATION

- **Goal:** develop and support common software solutions for the Run3 of the ALICE LHC experiment and the upcoming experiments at the Facility for Antiproton and Ion Research in Europe (FAIR) being built at GSI.
- Based on the experiences of **ALICE HLT** in Run1 / Run2 and the of the **FairRoot** framework.
- One of the examples of fruitful collaboration on Software Frameworks & Toolkits in HEP.
- I modestly contribute to it as part of the CERN ALICE Team, in particular to the so called Data Processing Layer.





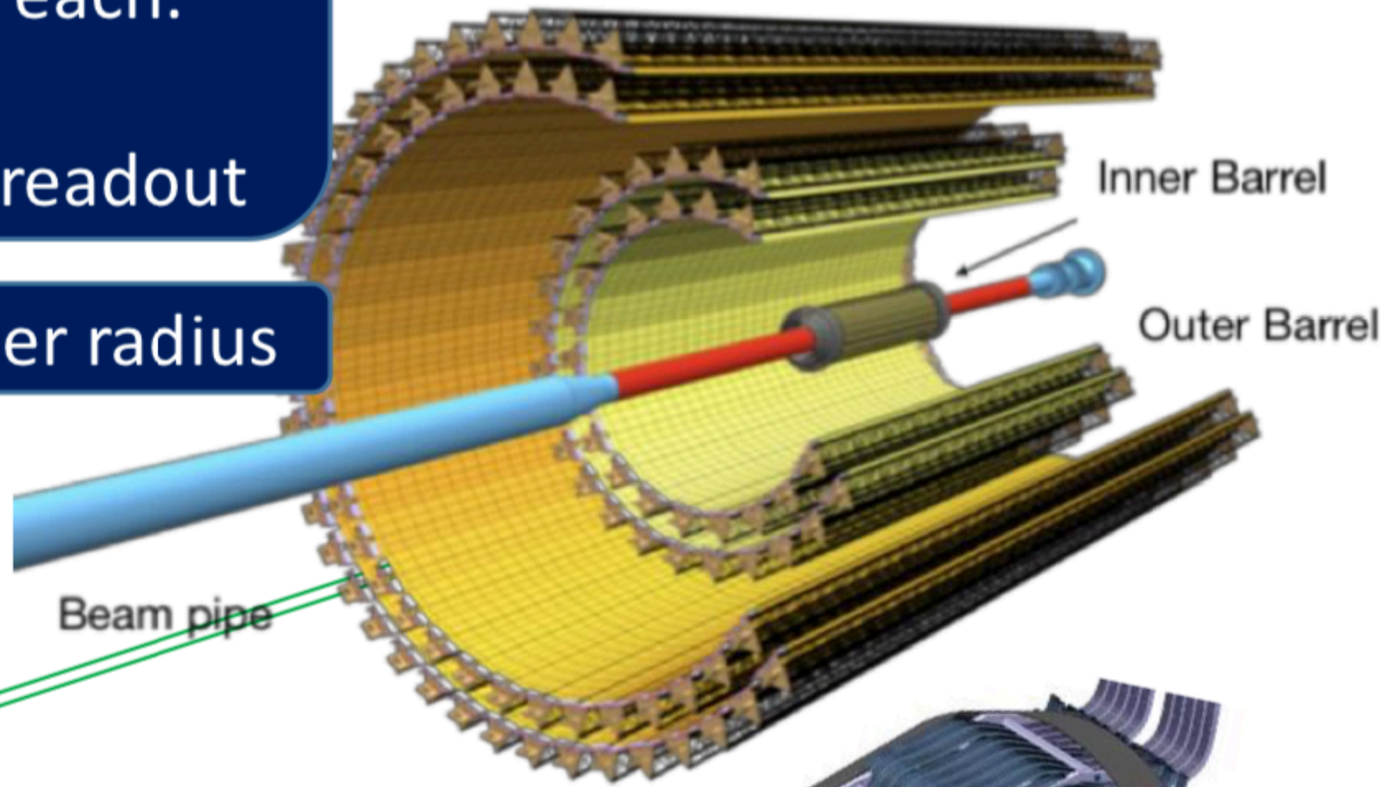
ALICE HW upgrades



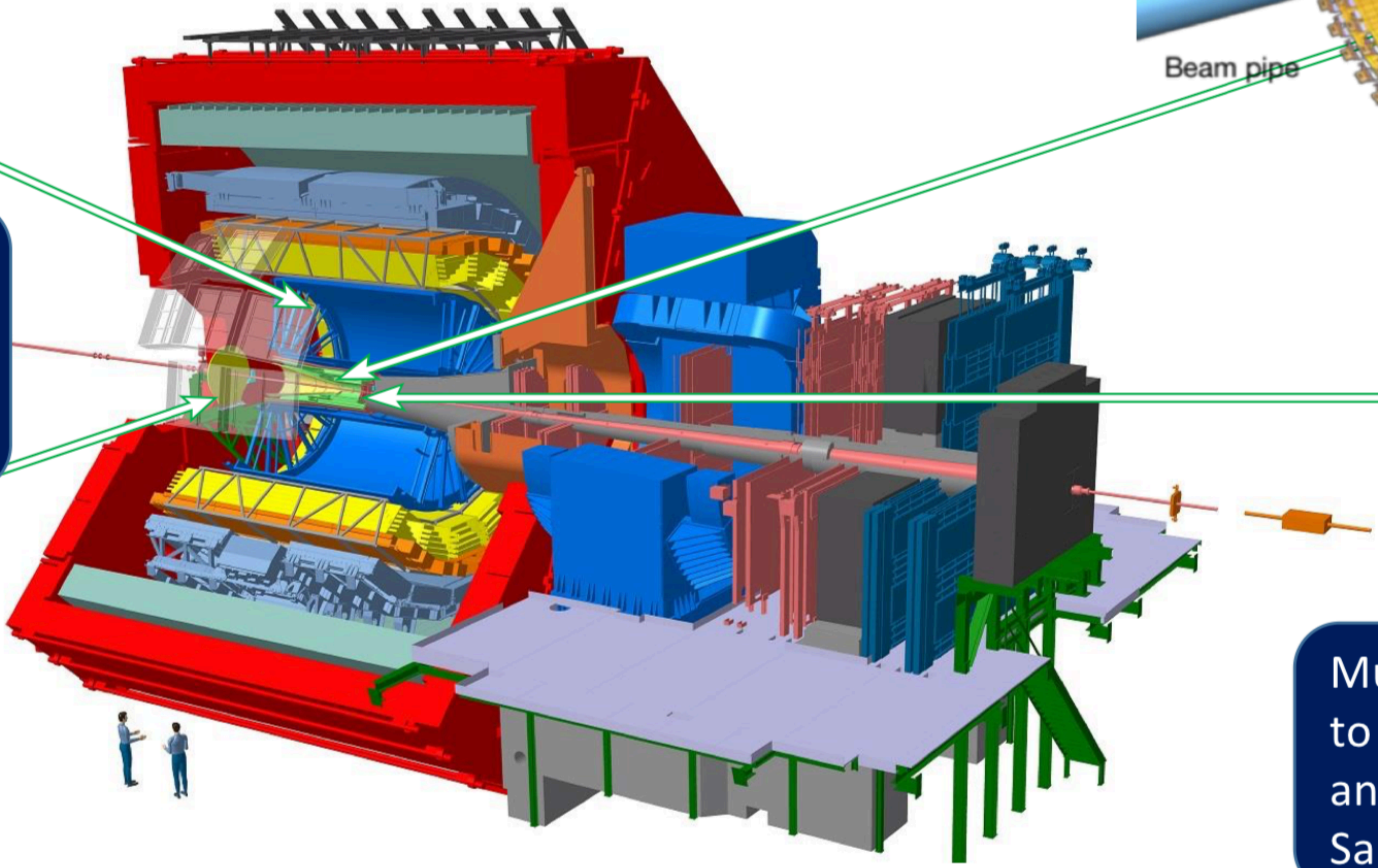
TPC MWPC readout → 4 layer GEM
(Intrinsic ion backflow ~99% blocking)
5MHz continuous sampling

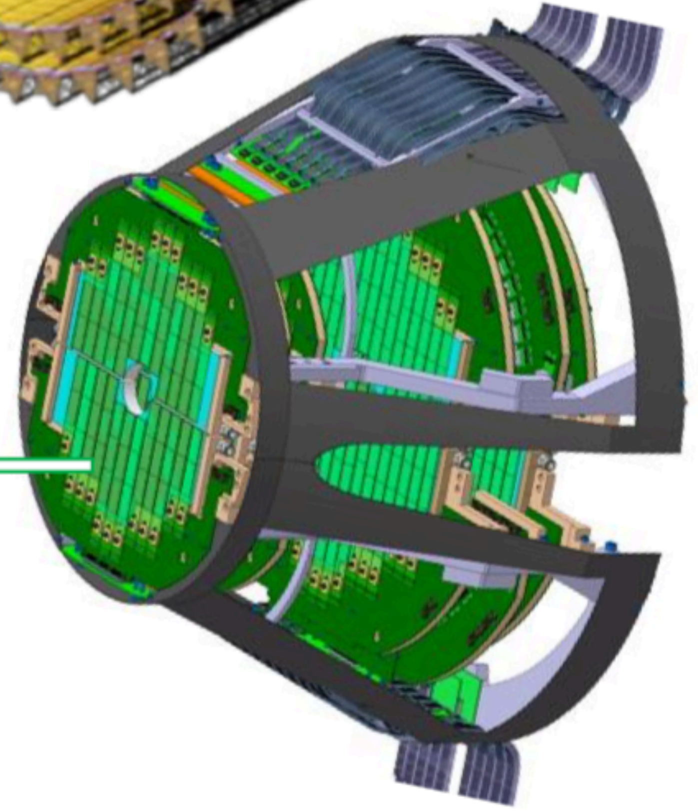
New Si Inner Tracker: 10 m^2 of MAPS with 29x27μm^2 pixel size
3 inner layers ~0.3% X0 each.
Closer to the beam
50-500 kHz continuous readout

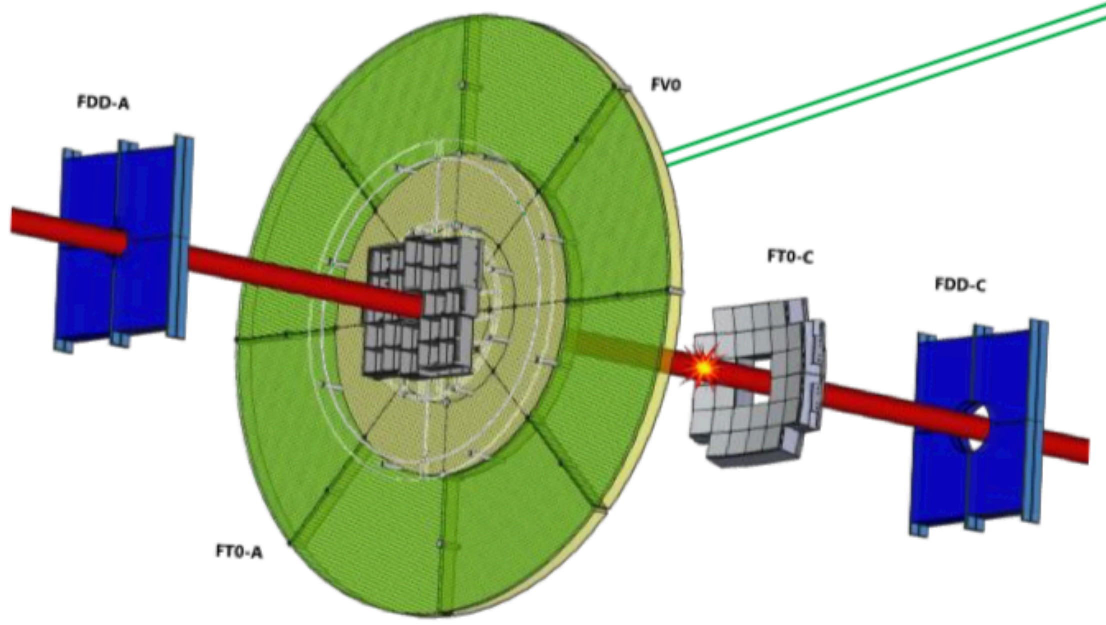
New beam pipe of smaller radius



Fast Interaction Trigger (FIT) detector
Scintillator (FV0, FDD) + Cerenkov (FT0) detectors to provide Min.Bias trigger for detectors with triggered R/O

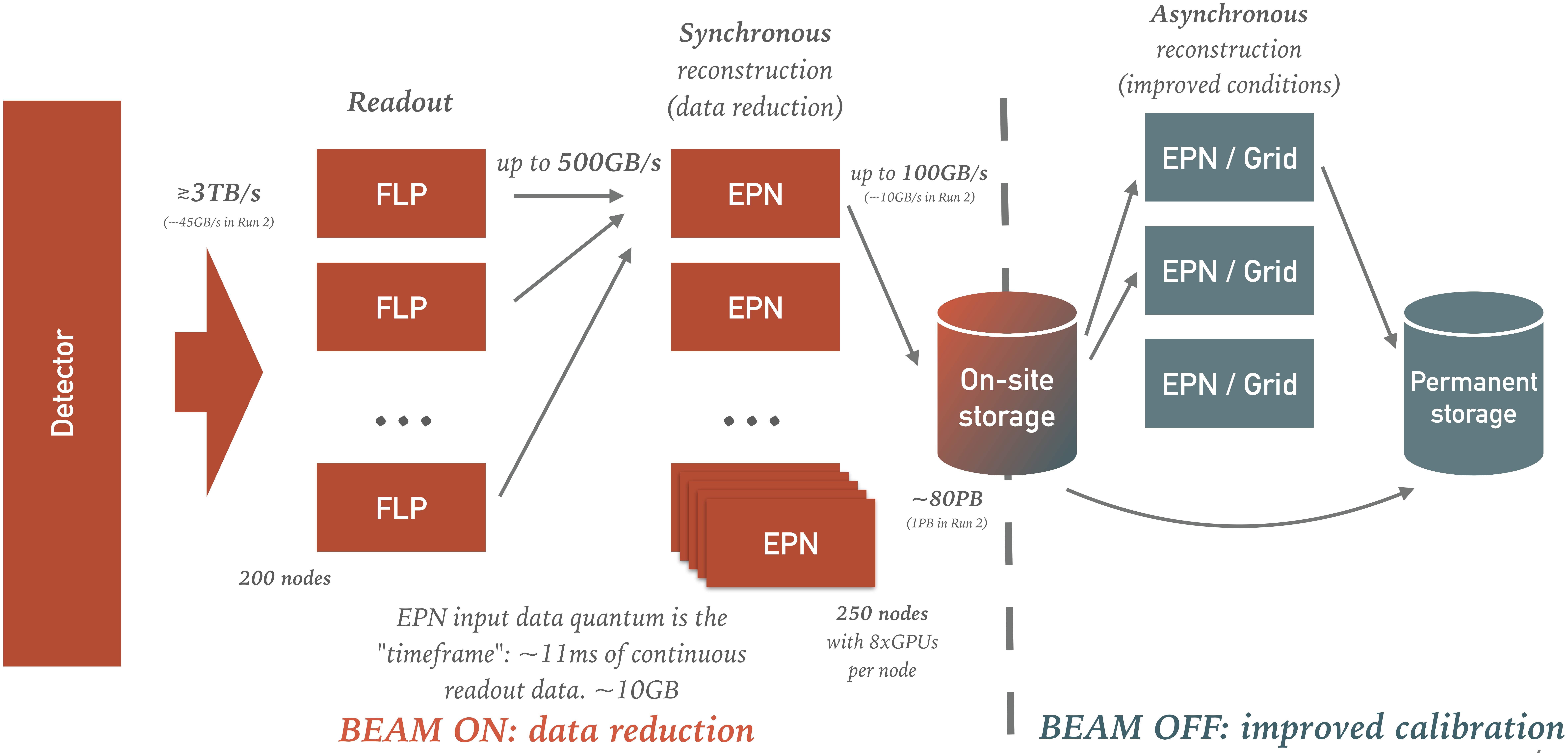






Muon Forward Tracker
to match muons before and after the absorber.
Same Si chips as new ITS

ALICE in Run 3: Point 2



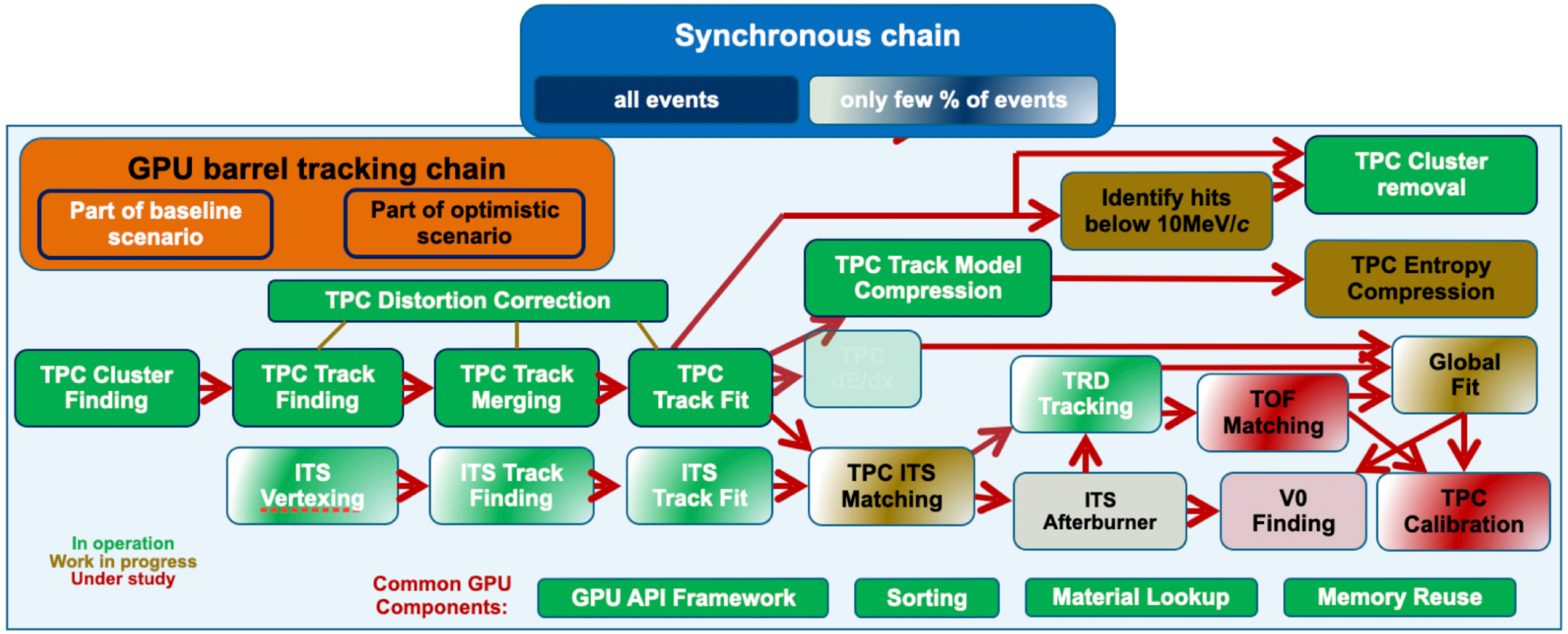
SYNCHRONOUS RECONSTRUCTION: GPUS AS FIRST CLASS CITIZENS

Synchronous processing requires GPU utilisation for TPC tracking. One modern GPU replaces 40 CPU cores. Changing the algorithm gives an additional 20x - 25x speedup. ***GPUs provide a 4x total benefit in terms of cost.***

ALICE will use ~250 dual AMD Rome for a total of 64 cores, each equipped with 8 AMD MI50 32 GB GPUs. 1500 GPUs needed to process @ 50 kHz, 20% margin.

Besides TPC tracking, baseline foresees running most of ITS tracking on the GPU. ***99% of the computing in synchronous phase already running on the GPU.***

*Same source code can targeted to support different GPU middlewares (**AMD HIP**, nVIDIA CUDA, OpenCL) or CPU (mostly for debugging and validation).*



Asynchronous Reconstruction

Follows the PbPb data taking, interleaved with pp. **Two processing cycles** per data taking year.

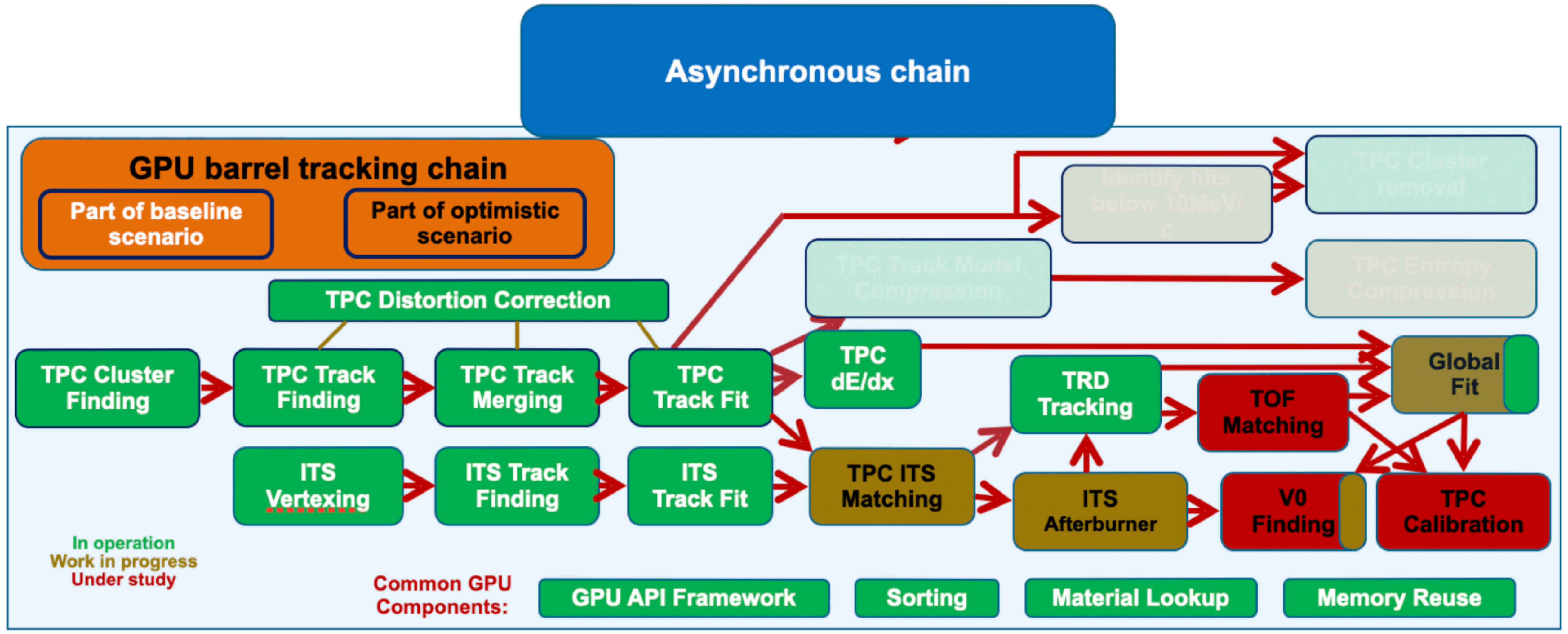
Processing on EPN farm (2/3 CTF volume) and the Grid (1/3).

Currently over 80% of the CPU - equivalent computing time running on GPUs. GPU usage is crucial to effectively use EPN farm when not taking data.

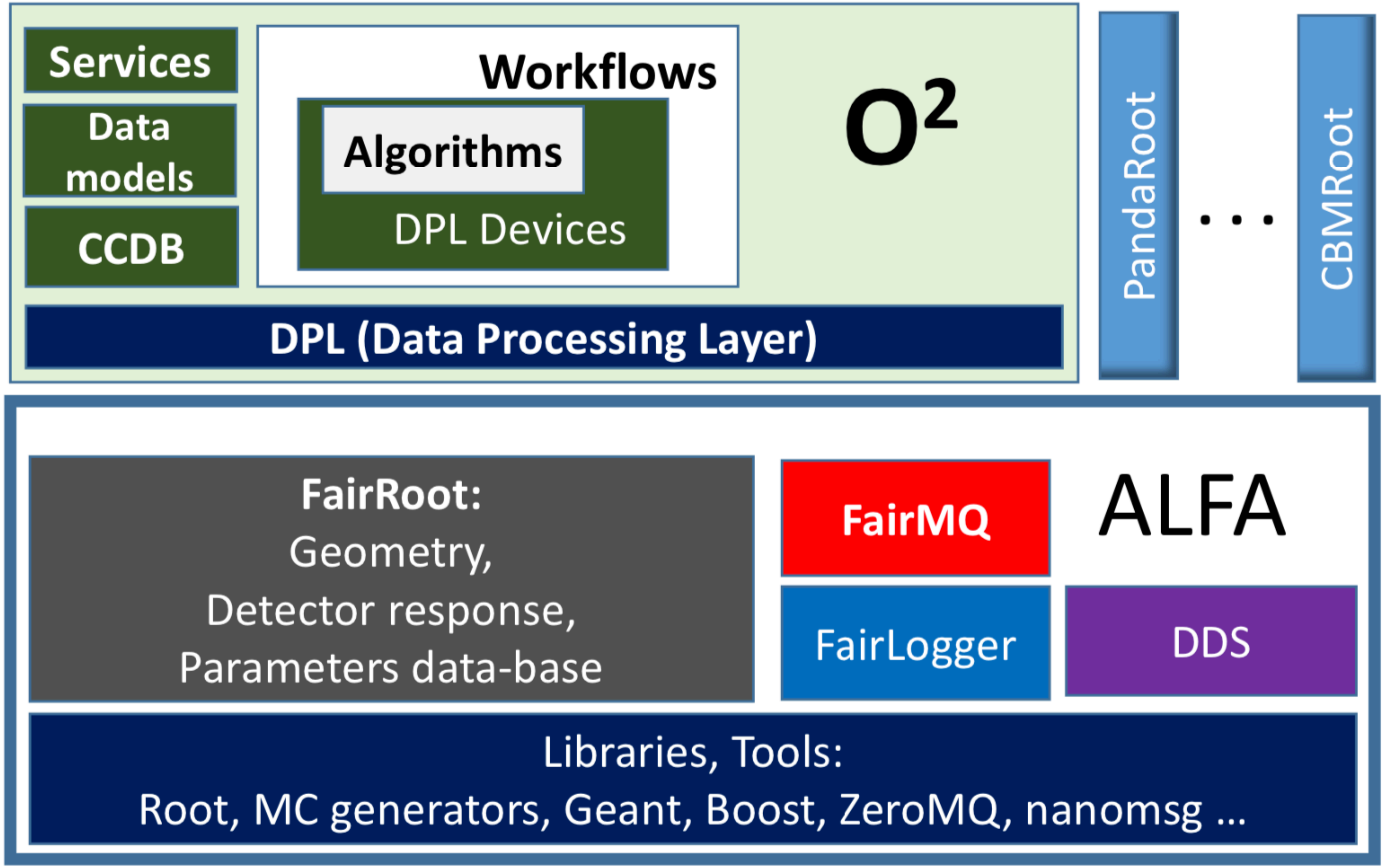
After 2nd cycle CTF will remain only on tape. Any subsequent cycle will have to wait until LHC LS.

Single persistent analysis object output - Analysis Object Data. All the analysis will have to be performed on such data and the associated derived objects.

20 PB of EOS disk cache already benchmarked and ready for commissioning.



THE BIG PICTURE



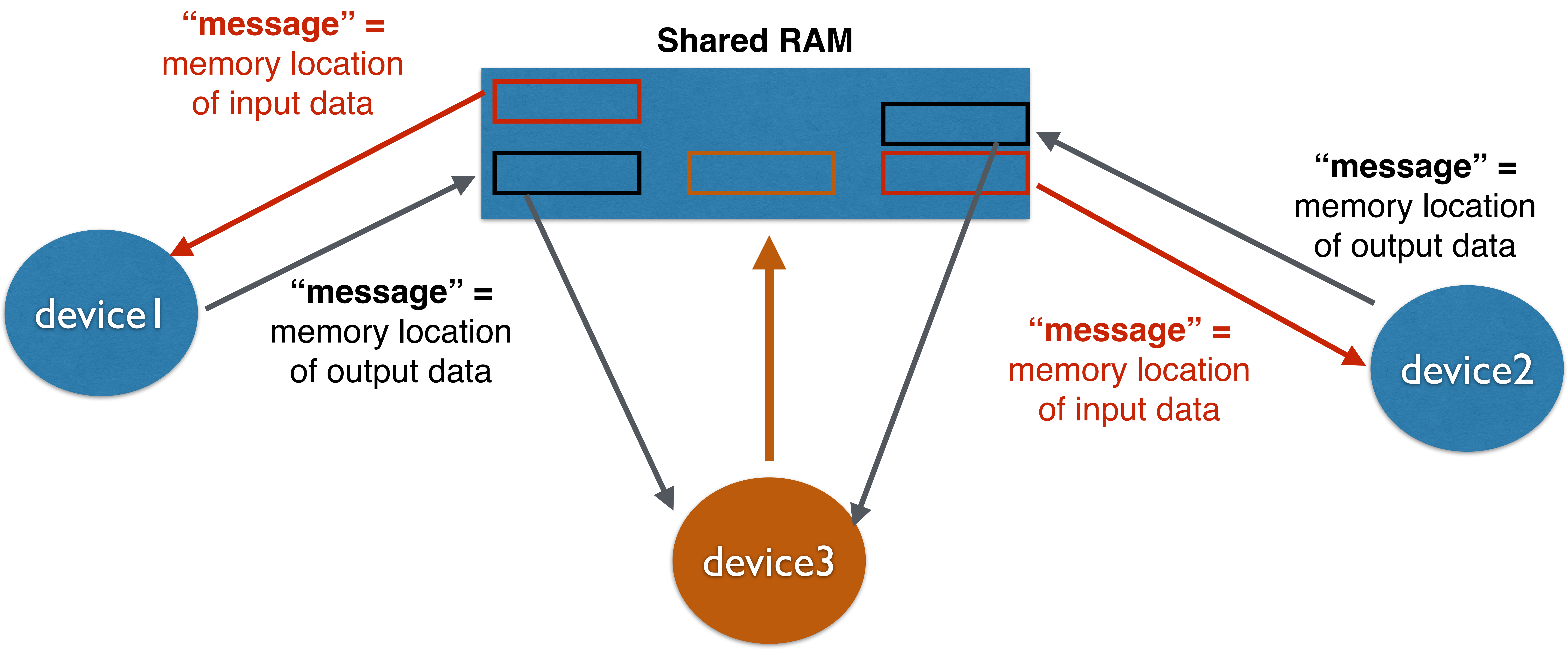
ALICE O2: Software Framework in one slide

Transport Layer: ALFA / FairMQ[1]

- **Standalone processes** *for deployment flexibility.*
- **Message passing** *as a parallelism paradigm.*
- **Shared memory** *backend for reduced memory usage and improved performance.*

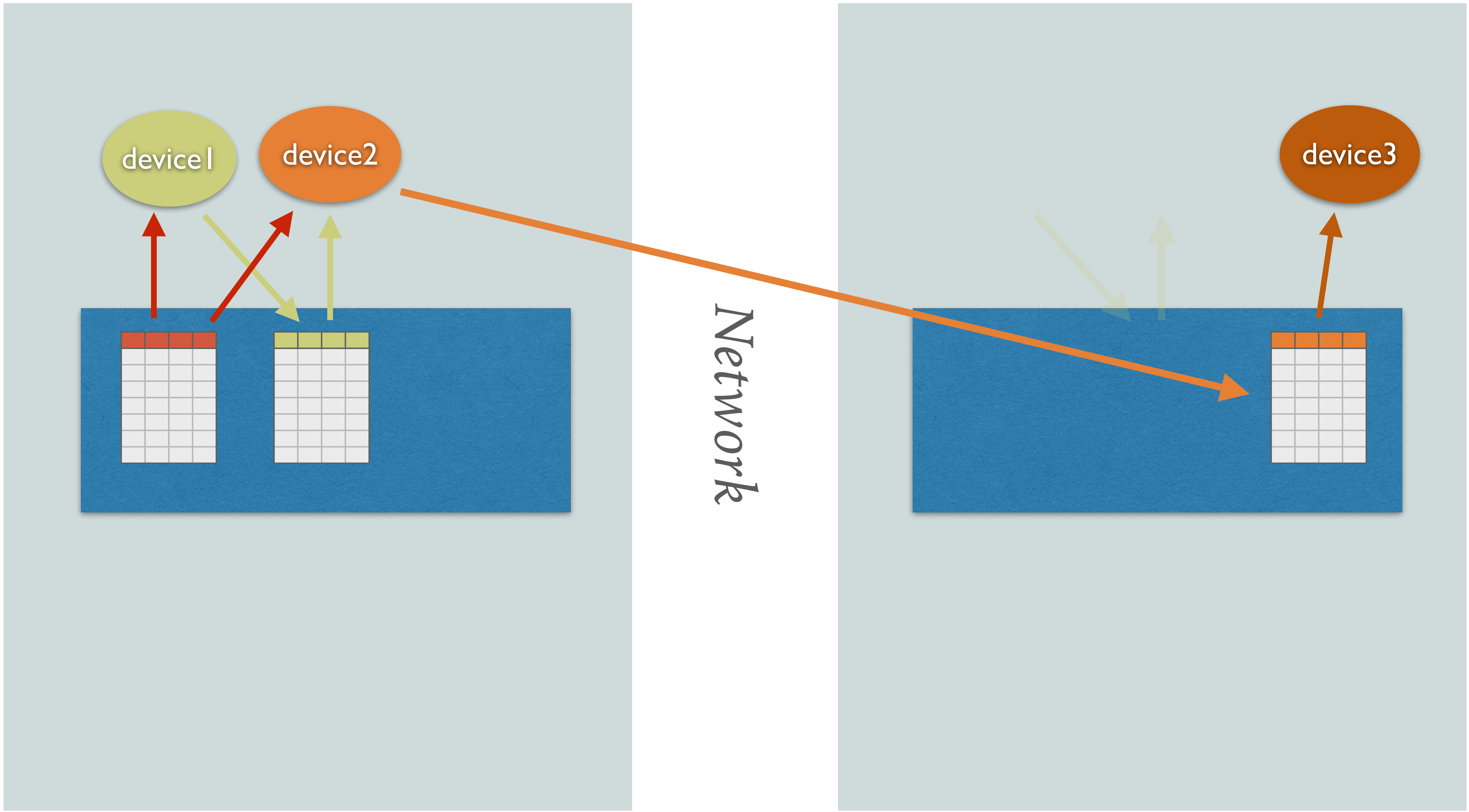
ALFA / FairMQ Framework: General Idea

Data processing happens in separate processes, called **devices**, exchanging data via a shared memory backed **Message Passing** paradigm.



ALFA / FairMQ Framework: General Idea

Seamless and homogeneous support for multi-node setups using one of the network enabled message passing backends, e.g. InfiniBand with RDMA.



SOFTWARE STACK BEHIND FAIRMQ

- Support for multiple message passing OpenSource libraries: **ZeroMQ,** nanomsg.
- In-house developed C++ bindings (**FairRootGroup/asiofi**) to OFI libfabric for InfiniBand support.
- Adoption of **boost::interprocess** for the shared memory backend.
- Support for multiple message serialisation (or not) protocols. Transport is agnostic about actual message content, allowing implementor to use their preferred technology: protobuf, flatbuffers, detector specific, **Apache Arrow**.
- State machine with pluggable support for deployment / control services: DDS, O^2 AliECS, PMIx or "standalone".

ALICE O2: Software Framework in one slide

Transport Layer: ALFA / FairMQ[1]

- **Standalone processes** *for deployment flexibility.*
- **Message passing** *as a parallelism paradigm.*
- **Shared memory** *backend for reduced memory usage and improved performance.*

ALICE O2: Software Framework in one slide

Data Layer: O2 Data Model

Message passing aware data model. Support for multiple backends:

- **Simplified, zero-copy** *format optimised for performance and direct GPU usage. Useful e.g. for TPC reconstruction on the GPU.*
- **ROOT based serialisation.** *Useful for QA and final results.*
- **Apache Arrow based.** *Useful as backend of the analysis ntuples and for integration with other tools.*

Transport Layer: ALFA / FairMQ[1]

- **Standalone processes** *for deployment flexibility.*
- **Message passing** *as a parallelism paradigm.*
- **Shared memory** *backend for reduced memory usage and improved performance.*

O2 DATA MODEL

A timeframe is a collection of (header, payload) pairs. Headers defines the type of data. Different header types can be stacked to store extra metadata (mimicking a Type hierarchy structure). Both header and payloads should be usable in a ***message passing*** *environment.*

Different payloads might have ***different serialisation strategies****. E.g.:*

- *TPC clusters / tracks:* ***flat POD data with relative indexes****, well suitable for GPU processing.*
- *QA histograms:* ***serialised ROOT*** *histograms.*
- *AOD:* ***columnar data format based on Arrow****.*

ALICE O2: Software Framework in one slide

Data Processing Layer (DPL)

Abstracts away the hiccups of a distributed system, presenting the user a familiar "Data Flow" system.

- ***Reactive-like design*** *(push data, don't pull)*
- ***Declarative Domain Specific Language*** *for topology configuration (C++17 based).*
- ***Integration*** *with the rest of the production system, e.g. Monitoring, Logging, Control.*
- ***Laptop mode****, including graphical debugging tools.*

Data Layer: O2 Data Model

Message passing aware data model. Support for multiple backends:

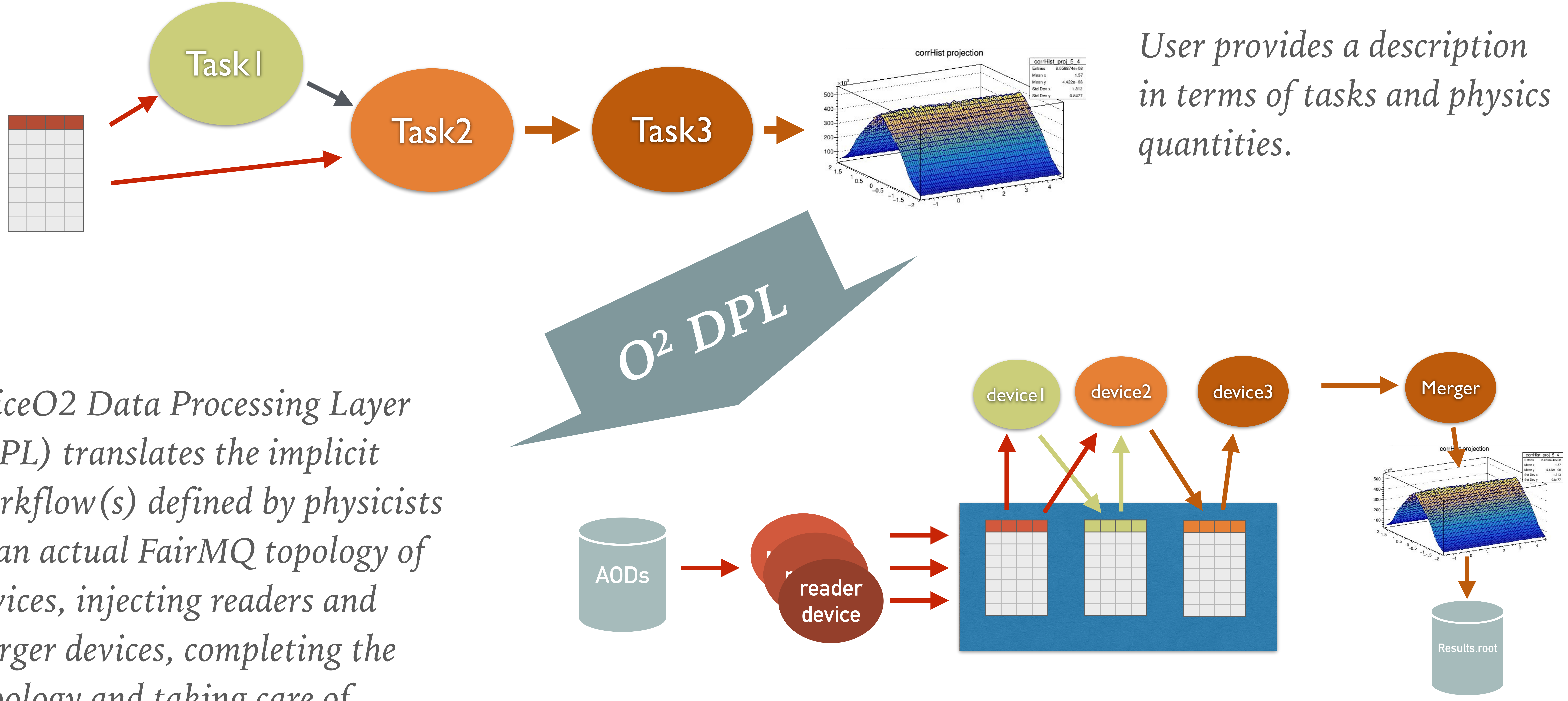
- **Simplified, zero-copy** *format optimised for performance and direct GPU usage. Useful e.g. for TPC reconstruction on the GPU.*
- **ROOT based serialisation.** *Useful for QA and final results.*
- **Apache Arrow based.** *Useful as backend of the analysis ntuples and for integration with other tools.*

Transport Layer: ALFA / FairMQ[1]

- **Standalone processes** *for deployment flexibility.*
- **Message passing** *as a parallelism paradigm.*
- **Shared memory** *backend for reduced memory usage and improved performance.*

AliceO2 Data Processing Layer

User provides a description in terms of tasks and physics quantities.

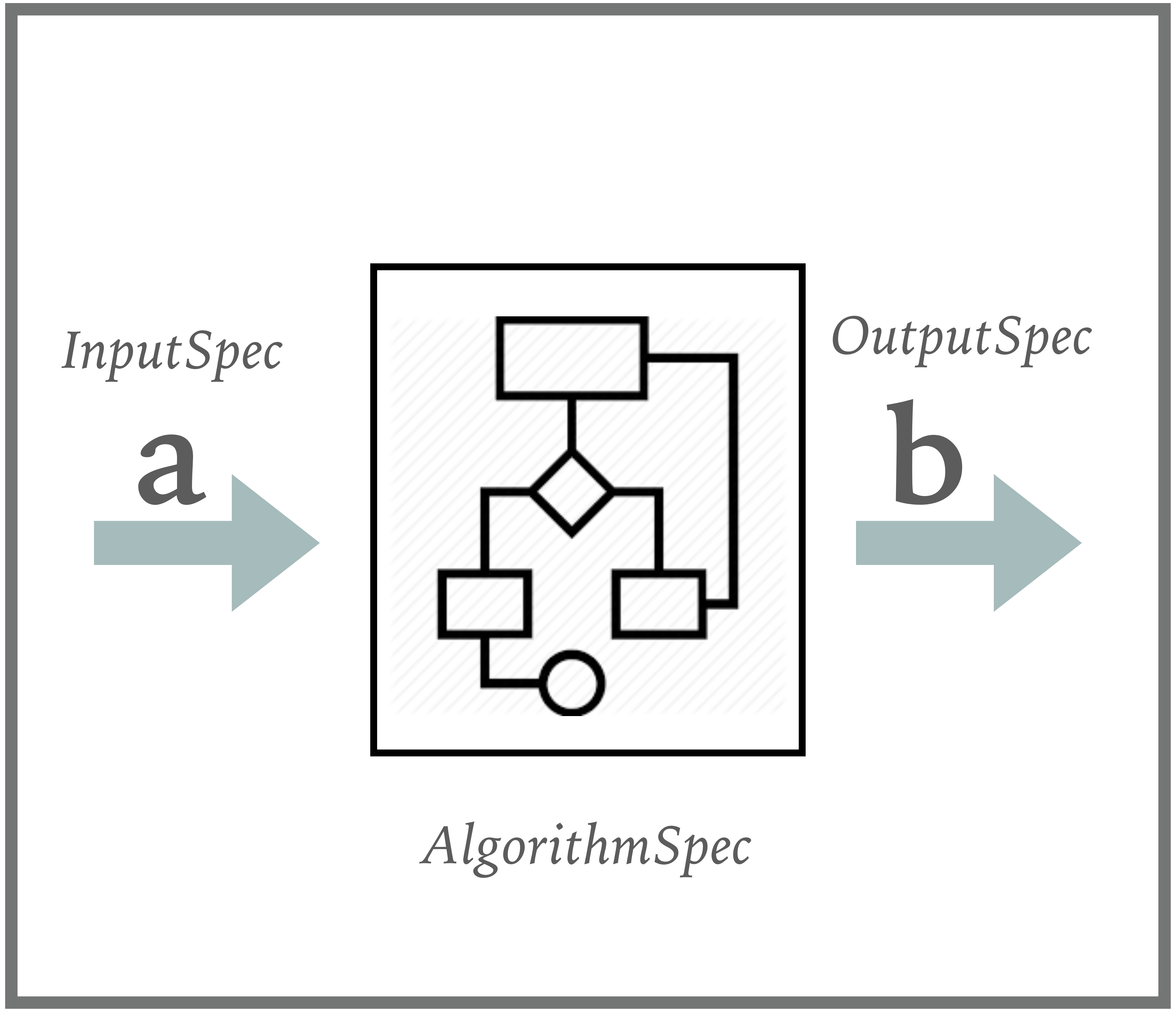


AliceO2 Data Processing Layer (DPL) translates the implicit workflow(s) defined by physicists to an actual FairMQ topology of devices, injecting readers and merger devices, completing the topology and taking care of parallelism / rate limiting.

DATA PROCESSING LAYER: BUILDING BLOCK

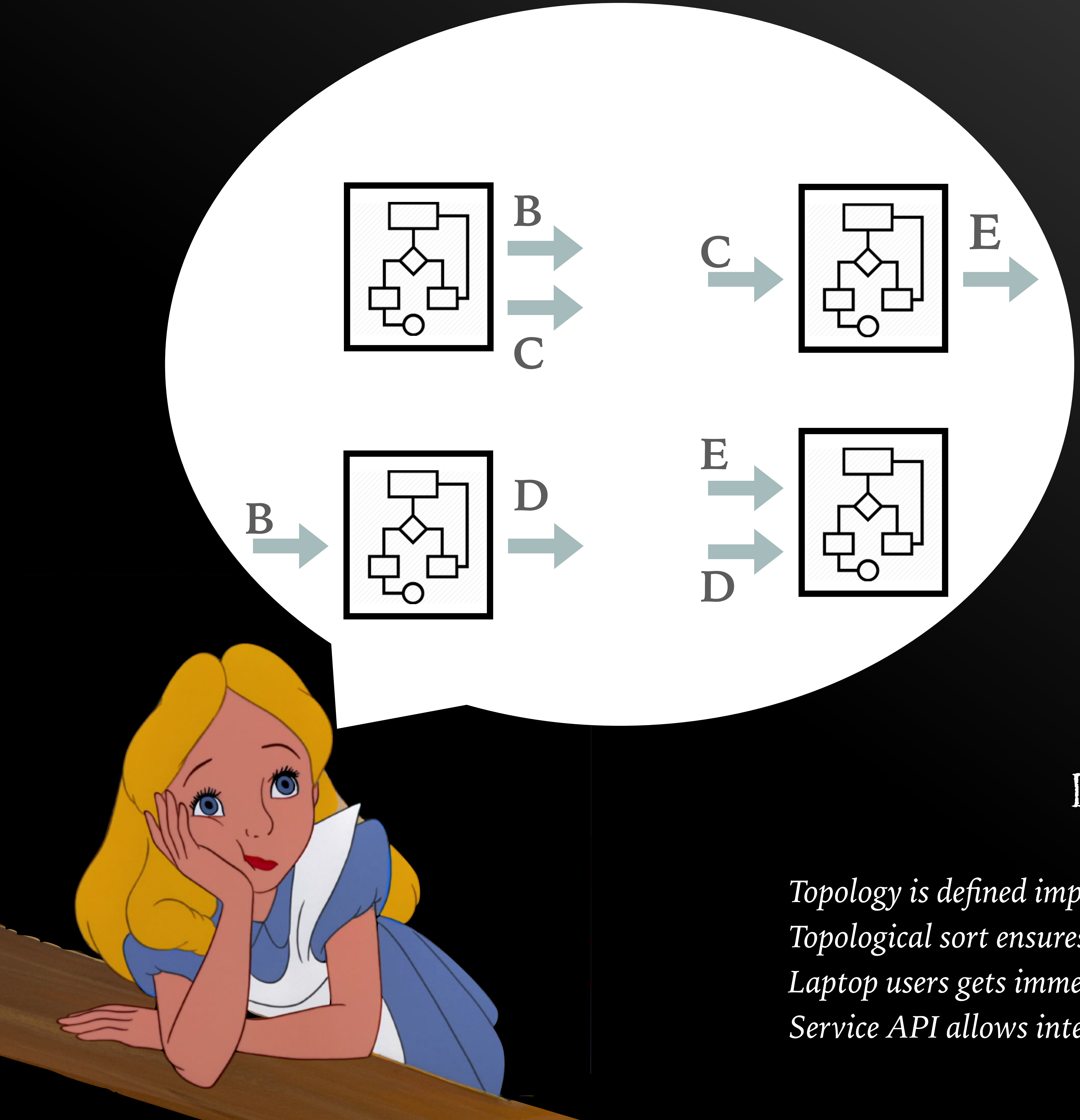
A DataProcessorSpec *defines a pipeline stage as a building block.*

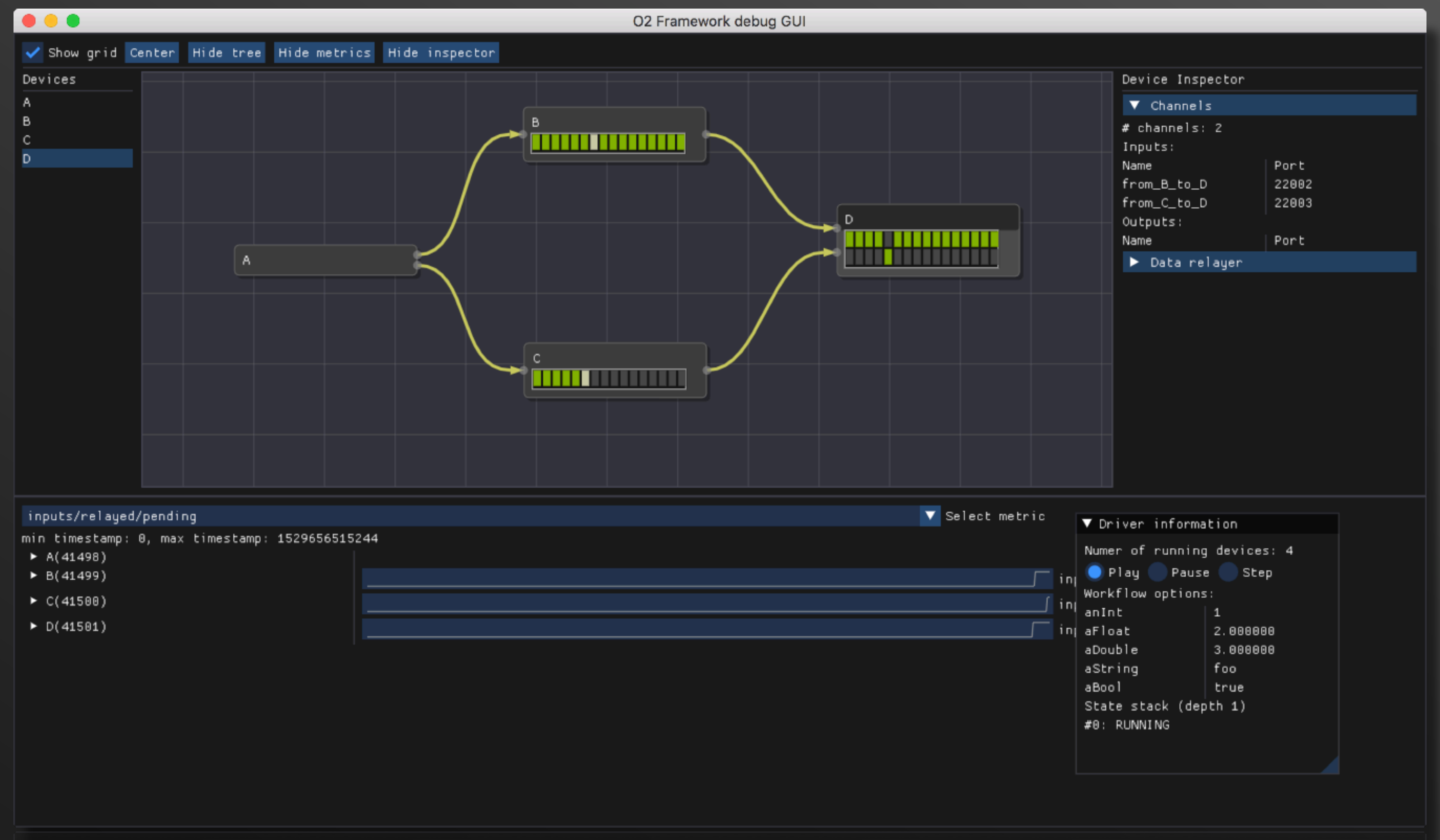
- *Specifies* ***inputs and outputs*** *in terms of the O2 Data Model descriptors.*
- *Provide an implementation of how to act on the inputs to produce the output.*
- *Advanced user can express possible data or time parallelism opportunities.*



DataProcessorSpec

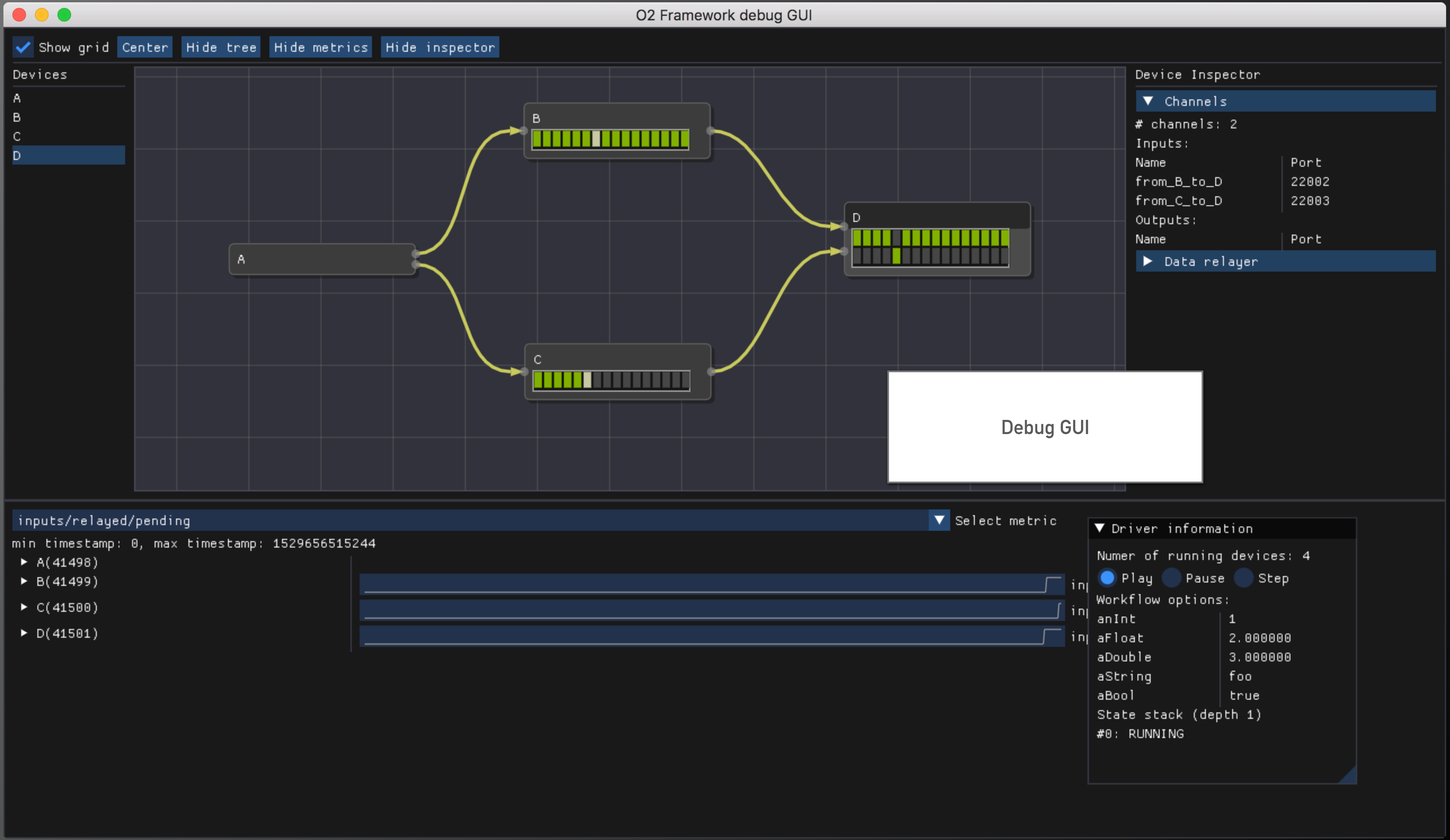
DATA PROCESSING LAYER: IMPLICIT TOPOLOGY



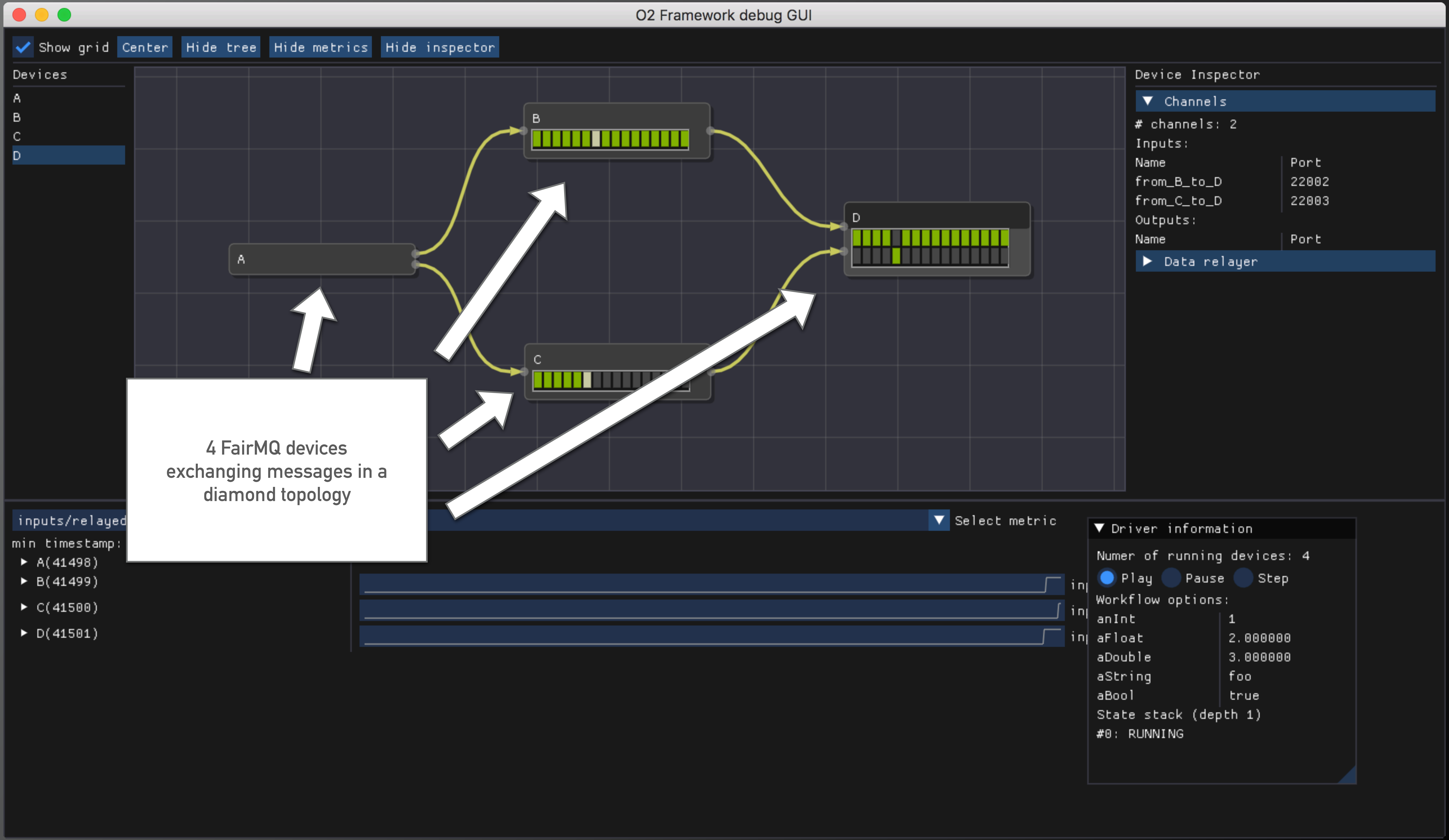


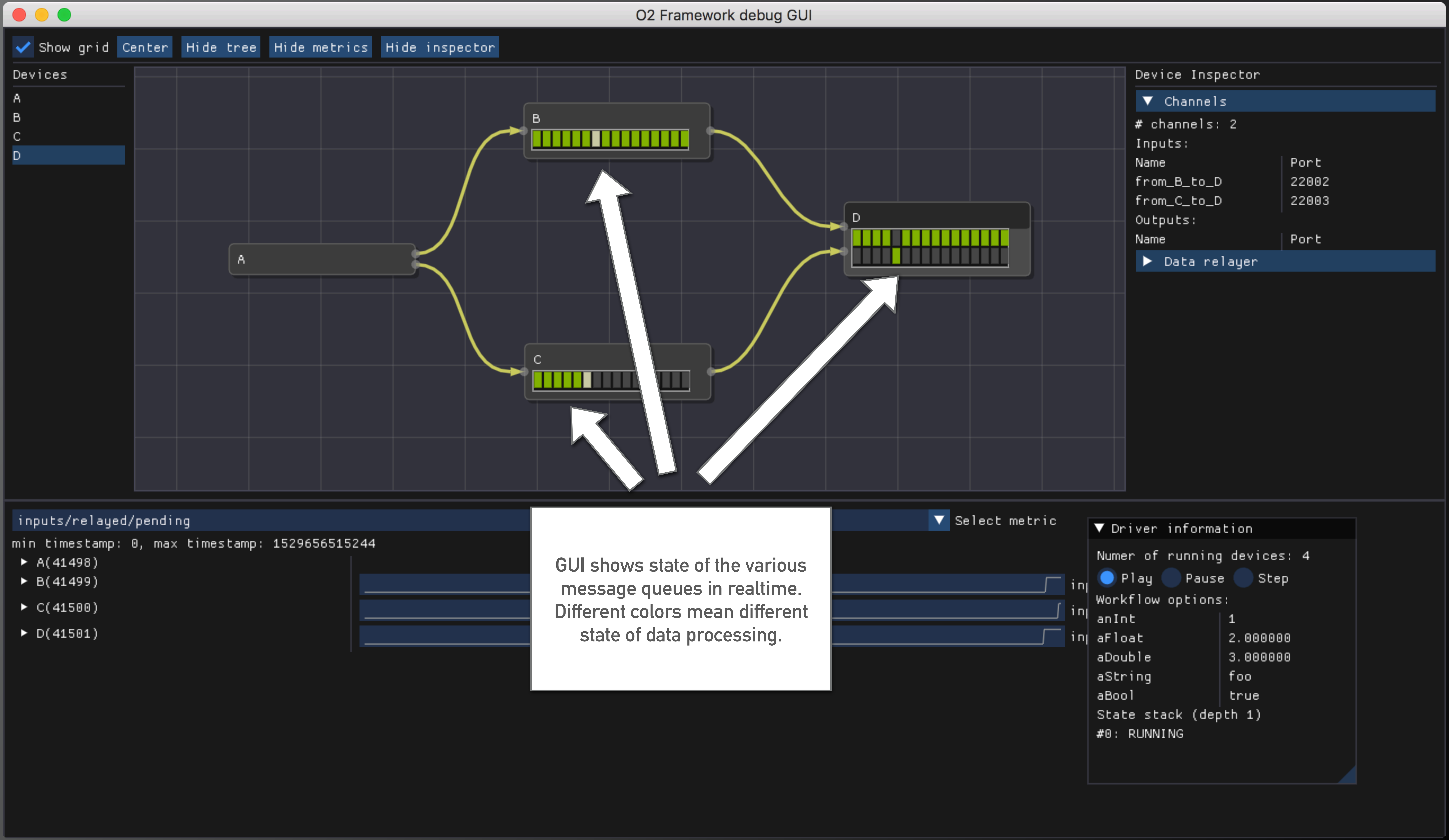
Data Processing Layer

Topology is defined implicitly.
Topological sort ensures a viable dataflow is constructed (no cycles!).
Laptop users gets immediate feedback through the debug GUI.
Service API allows integration with non data flow components (e.g. Control)

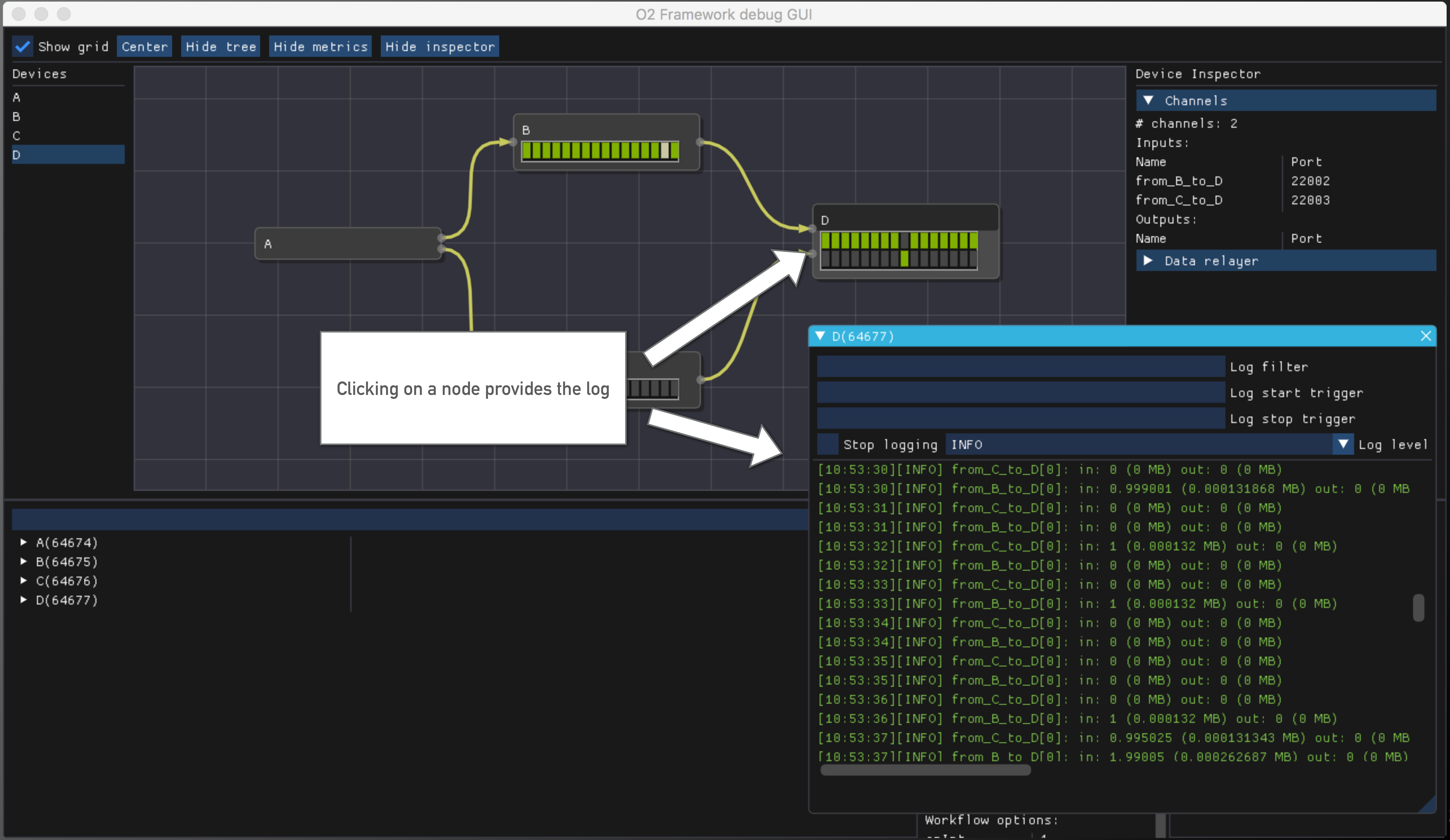

O2 Framework debug GUI
Show grid
Center
Hide tree
Hide metrics
Hide inspector
Devices
A
B
C
D
B
A
D
C
Device Inspector
Channels
channels: 2
Inputs:
Name
Port
from_B_to_D
22002
from_C_to_D
22003
Outputs:
Name
Port
Data relayer
inputs/relayed/pending
Select metric
min timestamp: 0, max timestamp: 1529656515244
A(41498)
B(41499)
C(41500)
D(41501)
Driver information
Numer of running devices: 4
Play
Pause
Step
Workflow options:
anInt
1
aFloat
2.000000
aDouble
3.000000
aString
foo
aBool
true
State stack (depth 1)
#0: RUNNING

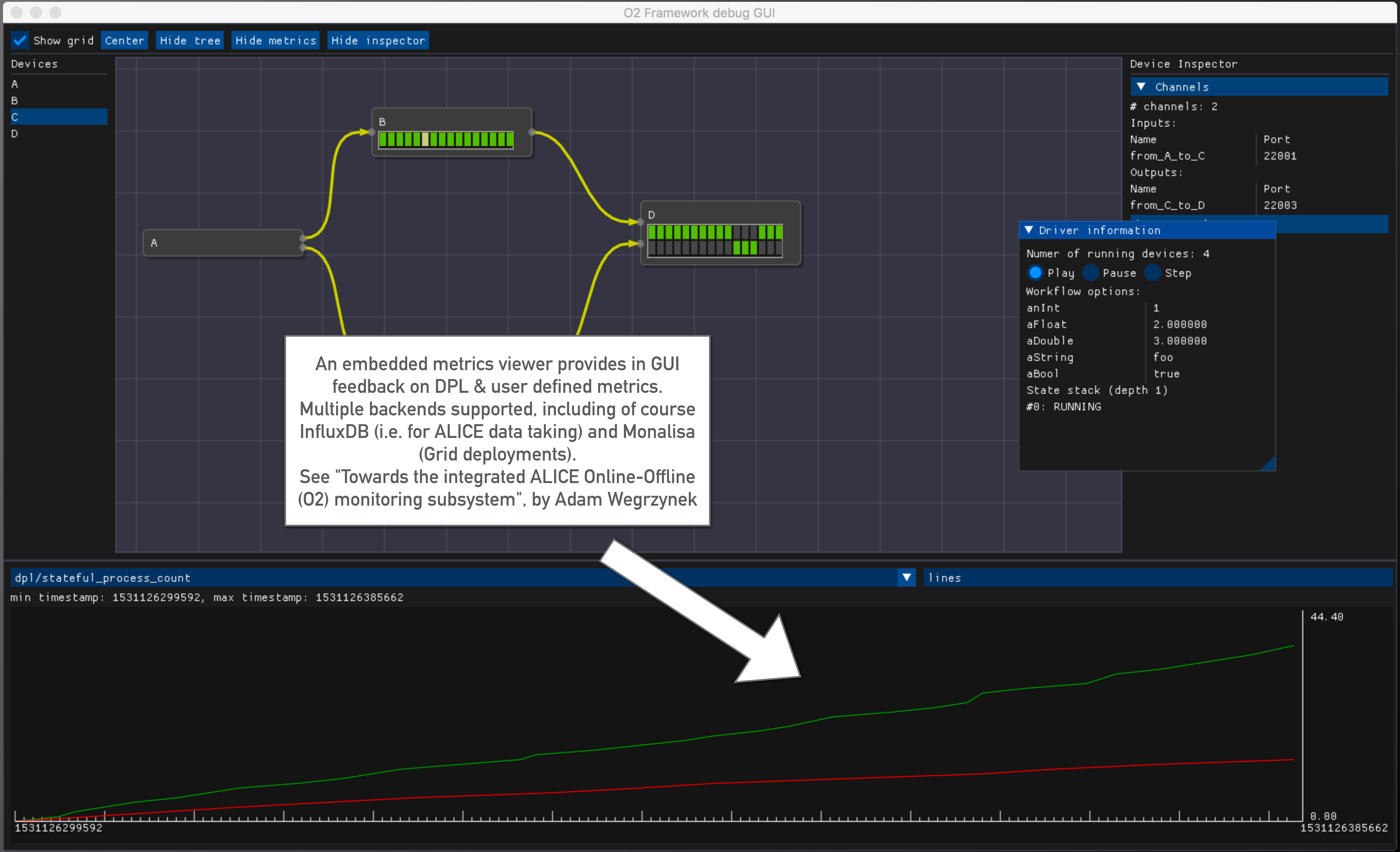
Debug GUI


O2 Framework debug GUI
Show grid
Center
Hide tree
Hide metrics
Hide inspector
Devices
A
B
C
D
Device Inspector
Channels
channels: 2
Inputs:
Name Port
from_B_to_D 22002
from_C_to_D 22003
Outputs:
Name Port
Data relayer
4 FairMQ devices exchanging messages in a diamond topology
inputs/relayed
Select metric
min timestamp:
A(41498)
B(41499)
C(41500)
D(41501)
Driver information
Numer of running devices: 4
Play Pause Step
Workflow options:
anInt 1
aFloat 2.000000
aDouble 3.000000
aString foo
aBool true
State stack (depth 1)
#0: RUNNING



GUI shows state of the various message queues in realtime. Different colors mean different state of data processing.


O2 Framework debug GUI
Show grid
Center
Hide tree
Hide metrics
Hide inspector
Devices
A
B
C
D
Device Inspector
Channels
channels: 2
Inputs:
Name Port
from_B_to_D 22002
from_C_to_D 22003
Outputs:
Name Port
Data relayer
Clicking on a node provides the log
D(64677)
Log filter
Log start trigger
Log stop trigger
Stop logging
INFO
Log level
[10:53:30][INFO] from_C_to_D[0]: in: 0 (0 MB) out: 0 (0 MB)
[10:53:30][INFO] from_B_to_D[0]: in: 0.999001 (0.000131868 MB) out: 0 (0 MB
[10:53:31][INFO] from_C_to_D[0]: in: 0 (0 MB) out: 0 (0 MB)
[10:53:31][INFO] from_B_to_D[0]: in: 0 (0 MB) out: 0 (0 MB)
[10:53:32][INFO] from_C_to_D[0]: in: 1 (0.000132 MB) out: 0 (0 MB)
[10:53:32][INFO] from_B_to_D[0]: in: 0 (0 MB) out: 0 (0 MB)
[10:53:33][INFO] from_C_to_D[0]: in: 0 (0 MB) out: 0 (0 MB)
[10:53:33][INFO] from_B_to_D[0]: in: 1 (0.000132 MB) out: 0 (0 MB)
[10:53:34][INFO] from_C_to_D[0]: in: 0 (0 MB) out: 0 (0 MB)
[10:53:34][INFO] from_B_to_D[0]: in: 0 (0 MB) out: 0 (0 MB)
[10:53:35][INFO] from_C_to_D[0]: in: 0 (0 MB) out: 0 (0 MB)
[10:53:35][INFO] from_B_to_D[0]: in: 0 (0 MB) out: 0 (0 MB)
[10:53:36][INFO] from_C_to_D[0]: in: 0 (0 MB) out: 0 (0 MB)
[10:53:36][INFO] from_B_to_D[0]: in: 1 (0.000132 MB) out: 0 (0 MB)
[10:53:37][INFO] from_C_to_D[0]: in: 0.995025 (0.000131343 MB) out: 0 (0 MB
[10:53:37][INFO] from_B_to_D[0]: in: 1.99005 (0.000262687 MB) out: 0 (0 MB)
A(64674)
B(64675)
C(64676)
D(64677)
Workflow options:



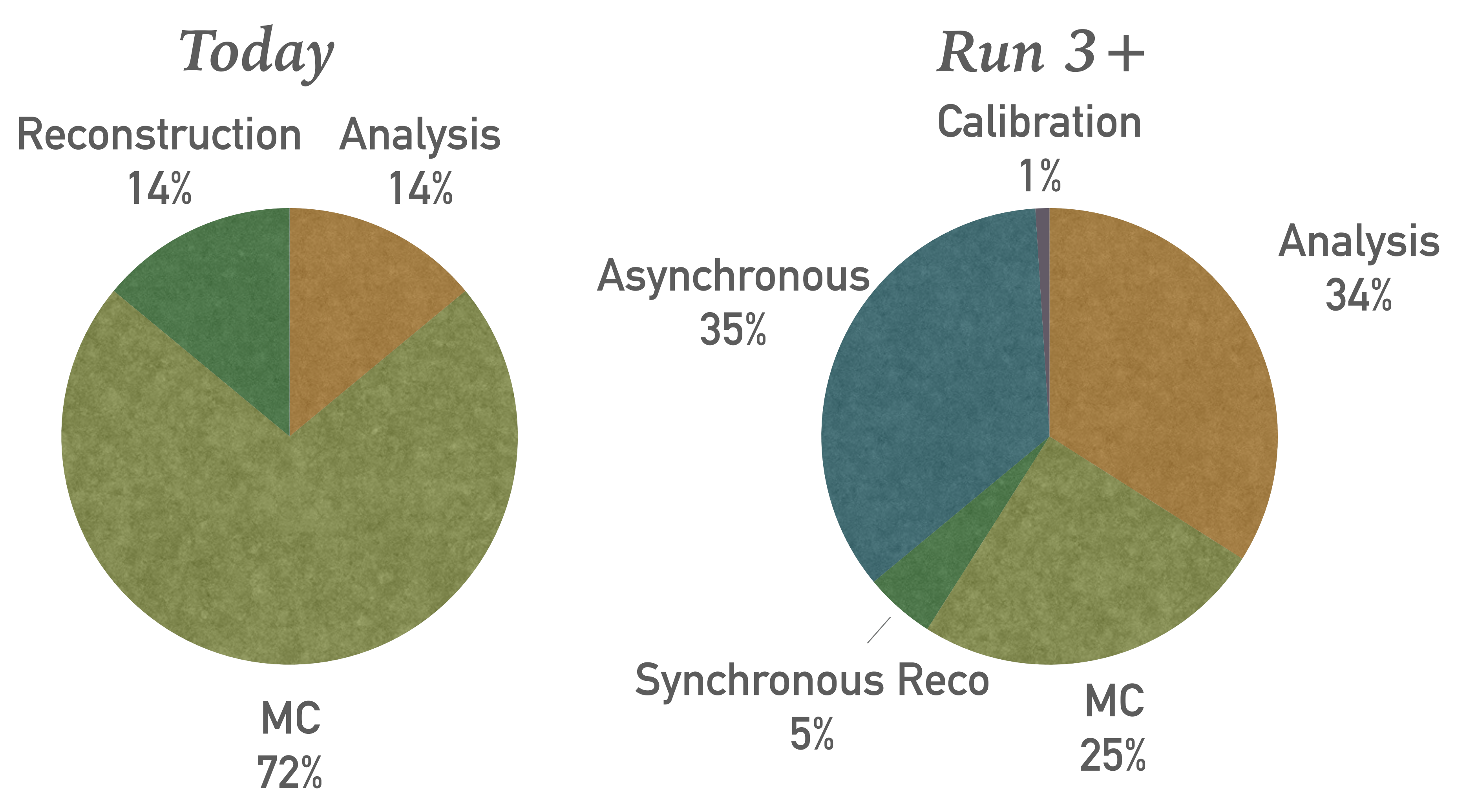
An embedded metrics viewer provides in GUI feedback on DPL & user defined metrics. Multiple backends supported, including of course InfluxDB (i.e. for ALICE data taking) and Monalisa (Grid deployments).
See "Towards the integrated ALICE Online-Offline (O2) monitoring subsystem", by Adam Wegrzynek

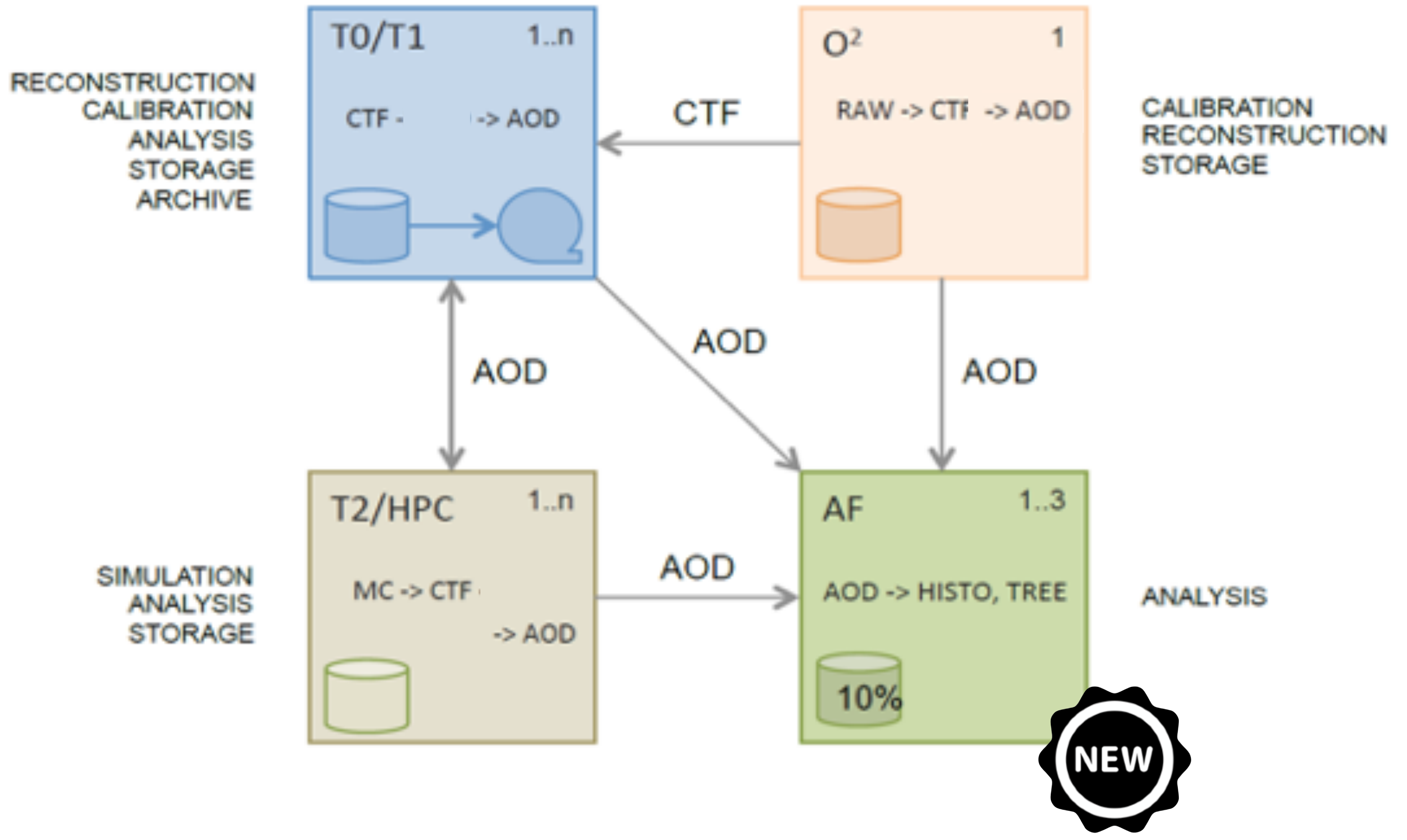

Shared Input
GPU Processing
Shared CPU Processing
Shared CPU Processing
NUMA Domain 1
NUMA Domain 2
Output
internal-dpl-clock
raw-file-reader
internal-dpl-clock
readout-proxy
tpc-tracker_t0
tpc-tracker_t1
tpc-tracker_t2
tpc-tracker_t3
its-stf-decoder_t0
its-stf-decoder_t1
its-stf-decoder_t2
mft-stf-decoder
MIDRawDecoder
ft0-datareader-dpl
tof-compressor-0
its-tracker
tof-compressed-decoder
MIDRawAggregator
ft0-reconstructor
mft-entropy-encoder
tpc-entropy-encoder_t0
tpc-entropy-encoder_t1
tpc-entropy-encoder_t2
tpc-entropy-encoder_t3
tpc-entropy-encoder_t4
tpc-entropy-encoder_t5
tpc-entropy-encoder_t6
ft0-entropy-encoder
its-entropy-encoder
MIDClusterizer
TOFClusterer
itstpc-track-matcher_t0
itstpc-track-matcher_t1
itstpc-track-matcher_t2
itstpc-track-matcher_t3
itstpc-track-matcher_t4
itstpc-track-matcher_t5
itstpc-track-matcher_t6
mid-entropy-encoder
tof-entropy-encoder
MIDTracker
TOFRecoWorkflow
tpc-track-interpolation
internal-dpl-global-binary-file-sink

Analysis Model: Run 3

Solid foundations: *the idea of organised analysis (trains) will stay. Improve on the implementation.*

- ***x100*** *more collisions compared to present setup,* ***AOD only.***
- *Initial analysis of 10% of the data at fewer* ***Analysis Facilities,*** *highly performant in terms of data access.*
- *Full analysis of a validated set of wagons on the Grid*
  ⇒ ***Prioritise processing according to physics needs.***
- ***Streamline data model****, trade generality for speed, flatten data structures.*
- ***Recompute*** *quantities on the fly rather than storing them. CPU cycles are cheap.*
- ***Produce highly targeted ntuples*** *(in terms of information needed and selected events of interest) to reduce turnaround for some key analysis.*
- ***Goal is to have each Analysis Facility go through the equivalent of 5PB of AODs every 12 hours (~100GB/s).***





Building an Analysis Framework for the Years to Come

Homogeneity: *use the same message passing architecture which will be used for data taking to ensure homogeneity, integration and provide easy access to parallelism for the analysis tasks.*

Fast: *simplify the Analysis Data Model to achieve higher performance (e.g. via reducing I/O cost, vectorisation) for critical usecases.*

Familiar: *hide as much as possible the internal details and expose an API which provides a classic Object Oriented "feeling".*

Modern: *follow developments in ROOT and provide an easy way to access modern ROOT tools like RDataFrame.*

Open to the rest of the world: *consider integration with external analysis frameworks (e.g. Python Pandas) and ML toolkits (e.g. Tensorflow) as a requirement.*

Data Model for Analysis

Flat tables: in order to minimise the I/O cost and improve vectorisation / parallelism opportunity data will be organised in memory as column-wise collections (Tables) holding the various entities. Frontend API will still allow for nested collections but the backend will map them to a set of chunked columns.

Relational: relationships between entities are expressed in a relational manner (e.g. via indexes between tables) or as optional values (optimised via a bitmask). Frontend will still allow references, however pointers are banned from the backend.

Shared memory / message passing friendly: if we want our analysis framework to be a good citizen in the O2 world, we need the data model and the backend to be optimised for shared memory backed message passing, so that we are not hit by serialisation / deserialisation costs.

Apache Arrow: a few technical details

In-memory column oriented storage (think TTrees, but shared memory friendly). *Full description: https://arrow.apache.org/docs/memory_layout.html. Data is organized in Tables. Tables are made of Columns. Columns are (<metadata>, Array). An Array is backed by one or multiple Buffers.*

Nullable fields. *An extra bitmap can optionally be provided to tell if a given slot in a column is occupied.*

Nested types. *Usual basic types (int, float, ..). It's also possible (via the usual record shredding presented in Google's Dremel paper) to support nested types. E.g. a String is a List<Char>.*

No (generic) polymorphism. *The type in an array can be nested, but there is no polymorphisms available (can be faked via nullable fields & unions).*

Gandiva: JIT compiled, vectorised, query engine now available in upstream.

Now used in production for Run 3 Analysis.

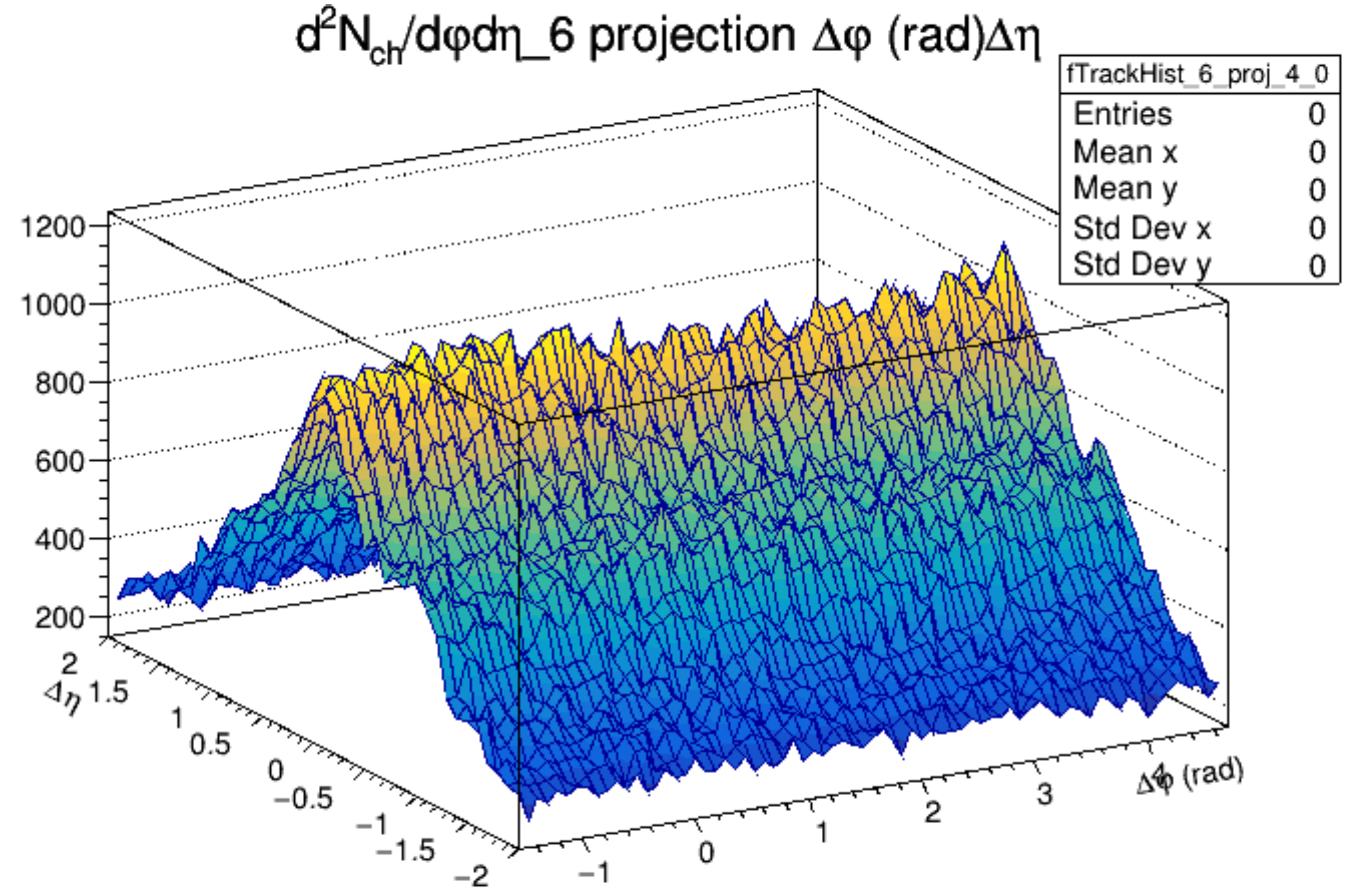
A Trivial Analysis

- Define a standalone workflow
- Define an AnalysisTask
- Define outputs, filters, partitions.
- Subscribe to the tracks for a given timeframe
- Compute (e.g.) φ from the propagation parameters
- Fill a plot

```
#include "Framework/runDataProcessing.h"
#include "Framework/AnalysisTask.h"
#include "Framework/AnalysisDataModel.h"
#include <TH1F.h>

using namespace o2;
using namespace o2::framework;

struct ATask : AnalysisTask
{
  OutputObj<TH1F> hPhi{TH1F("phi", "Phi", 100, 0, 2 * M_PI)};
  Filter ptFilter = aod::track::pt > 1;
  Partition pos = aod::track::x >= 0;

  void process(aod::Tracks const& tracks)
  {
    for (auto& track : pos(tracks)) {
      float phi = asin(track.snp()) + track.alpha() + M_PI;
      hPhi->Fill(phi);
    }
  }
};

WorkflowSpec defineDataProcessing(ConfigContext const&)
{
  return WorkflowSpec{
    adaptAnalysisTask<ATask>("mySimpleTrackAnalysis", 0)
  };
}
```

...AND ONE STEP BEYOND...

```
void process(aod::Collision const& collision, aod::Tracks const& tracks)
{
  LOGF(info, "Tracks for collision: %d", tracks.size());

  for (auto it1 = tracks.begin(); it1 != tracks.end(); ++it1)
  {
    auto& track1 = *it1;
    if (track1.pt() < 0.5)
      continue;

    double eventValues[3];
    eventValues[0] = track1.pt();
    eventValues[1] = collision.v0mult();;
    eventValues[2] = collision.positionZ();

    same->GetEventHist()->Fill(eventValues, AliUEHist::kCFStepReconstructed);
    //mixed->GetEventHist()->Fill(eventValues, AliUEHist::kCFStepReconstructed);

    for (auto it2 = it1 + 1; it2 != tracks.end(); ++it2)
    {
      auto& track2 = *it2;
      if (track1 == track2)
        continue;
      if (track2.pt() < 0.5)
        continue;

      double values[6];

      values[0] = track1.eta() - track2.eta();
      values[1] = track1.pt();
      values[2] = track2.pt();
      values[3] = collision.v0mult();

      values[4] = track1.phi() - track2.phi();
      if (values[4] > 1.5 * TMath::Pi())
        values[4] -= TMath::TwoPi();
      if (values[4] < -0.5 * TMath::Pi())
        values[4] += TMath::TwoPi();

      values[5] = collision.positionZ();

      same->GetTrackHist()->Fill(values, AliUEHist::kCFStepReconstructed);
      //mixed->GetTrackHist()->Fill(values, AliUEHist::kCFStepReconstructed);
    }
  }
}
```



Courtesy of Jan-Fiete Grosse-Oetringhaus

Strategy underneath the Example

This is of course something very trivial, but it well illustrates the pursued strategy:

- **Task based API**: *reproduce run 1 and 2 analysis task concept to make transition easier. Task members are extracted to define outputs, filters, selections.*
- ***O²* DPL underpinnings**: *this is actually an O² DPL workflow, heavy-lifting to map it to the message passing topology is taken care of by the framework.*
- **Type-safe**: *users subscribe to the inputs they need, in a type safe manner.* `aod::Tracks` *is a an actual type, which the DPL maps automatically to messages matching the associated Data Header.*
- **Arrow Skins**: *users are exposed to a familiar Imperative / "Object Oriented" API to access physics objects. In reality they act on an Apache Arrow backed* ***AoSoA*** *data store, on top of which the Framework allows to construct "Skins". Similar to LHCb SOAContainer or CMS FWCore/SOA.*
- **Generic:** *the signature of the* ***process*** *method is what drives the subscription to data (via template magic). E.g. to get all the tracks associated to a given collision:*

```
void process(aod::Collision& collision, aod::Tracks const& tracks)
```

Arrow Skins: Data Definition Example

```cpp
#include "Framework/ASoA.h"

namespace o2::aod
{
namespace track
{
DECLARE_SOA_COLUMN(CollisionId, collisionId, int, "fID4Tracks");
DECLARE_SOA_COLUMN(Alpha, alpha, float, "fAlpha");
DECLARE_SOA_COLUMN(Snp, snp, float, "fSnp");
// ...
DECLARE_SOA_DYNAMIC_COLUMN(Phi, phi,
  [](float snp, float alpha) { return asin(snp) + alpha + M_PI; });
} // namespace track

using Tracks = soa::Table<track::CollisionId, track::Alpha,
                          /* ... */,
                          track::Snp, track::Tgl,
                          track::Phi<track::Snp, track::Alpha>>;

using Track = Tracks::iterator;
}
```

Column

The smallest component is the Column, which is a type mapped to a specific column name.

Table

A Table is a generic union of Column types.

Dynamic Columns

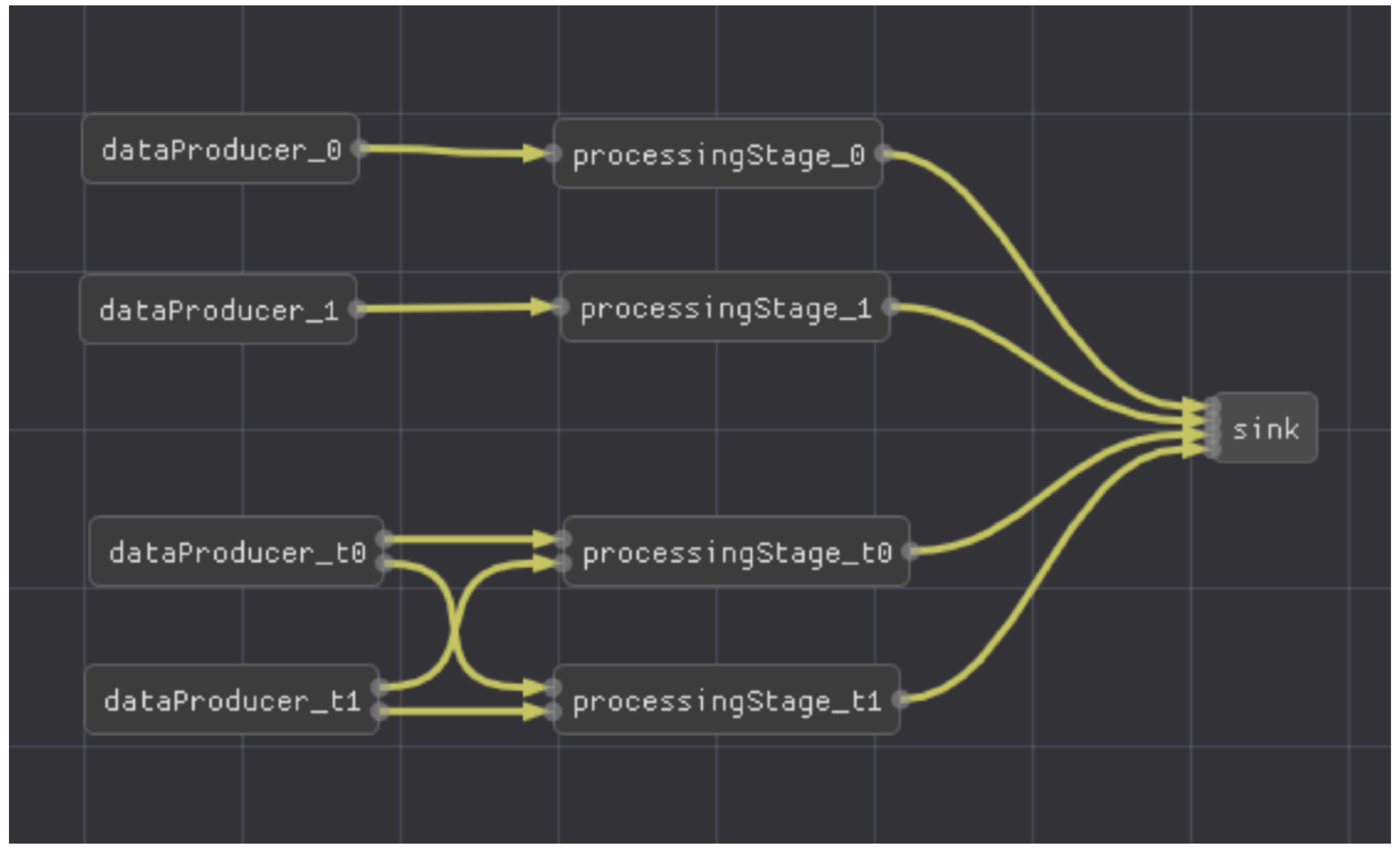
Non persistent (i.e. calculated) quantities can be associated with a table in the form of a so called dynamic column.

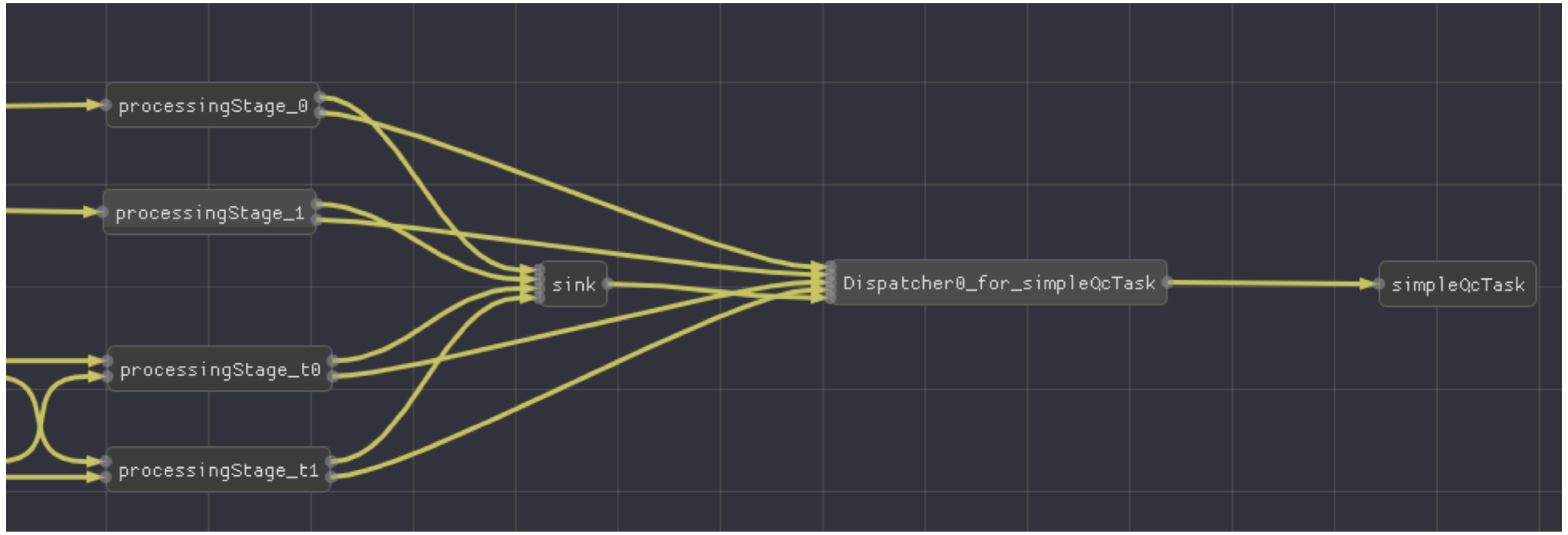
Object

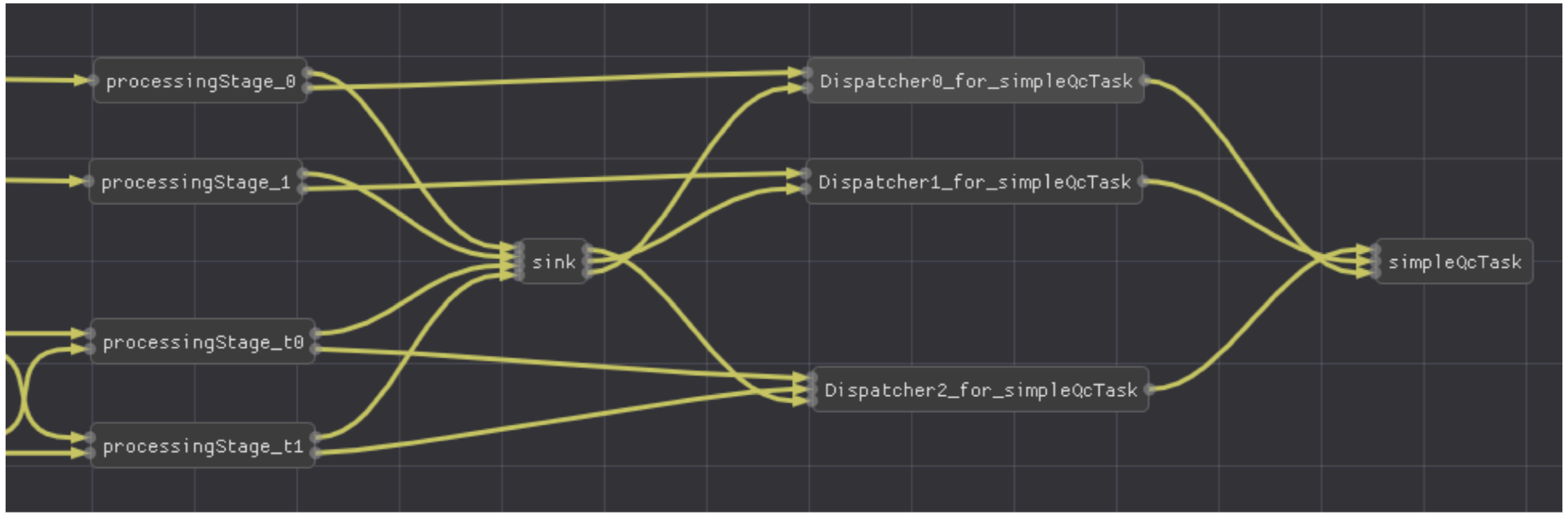
An object is actually an alias to the simultaneous iterators over the columns involved in a given table row.

COMPOSABLE WORKFLOWS

Declarative configuration allows for easy customisation: e.g. adding a (one or more) dispatchers for QA.







Workflows are executables. Piping on the shell multiple executables builds the closure workflow.

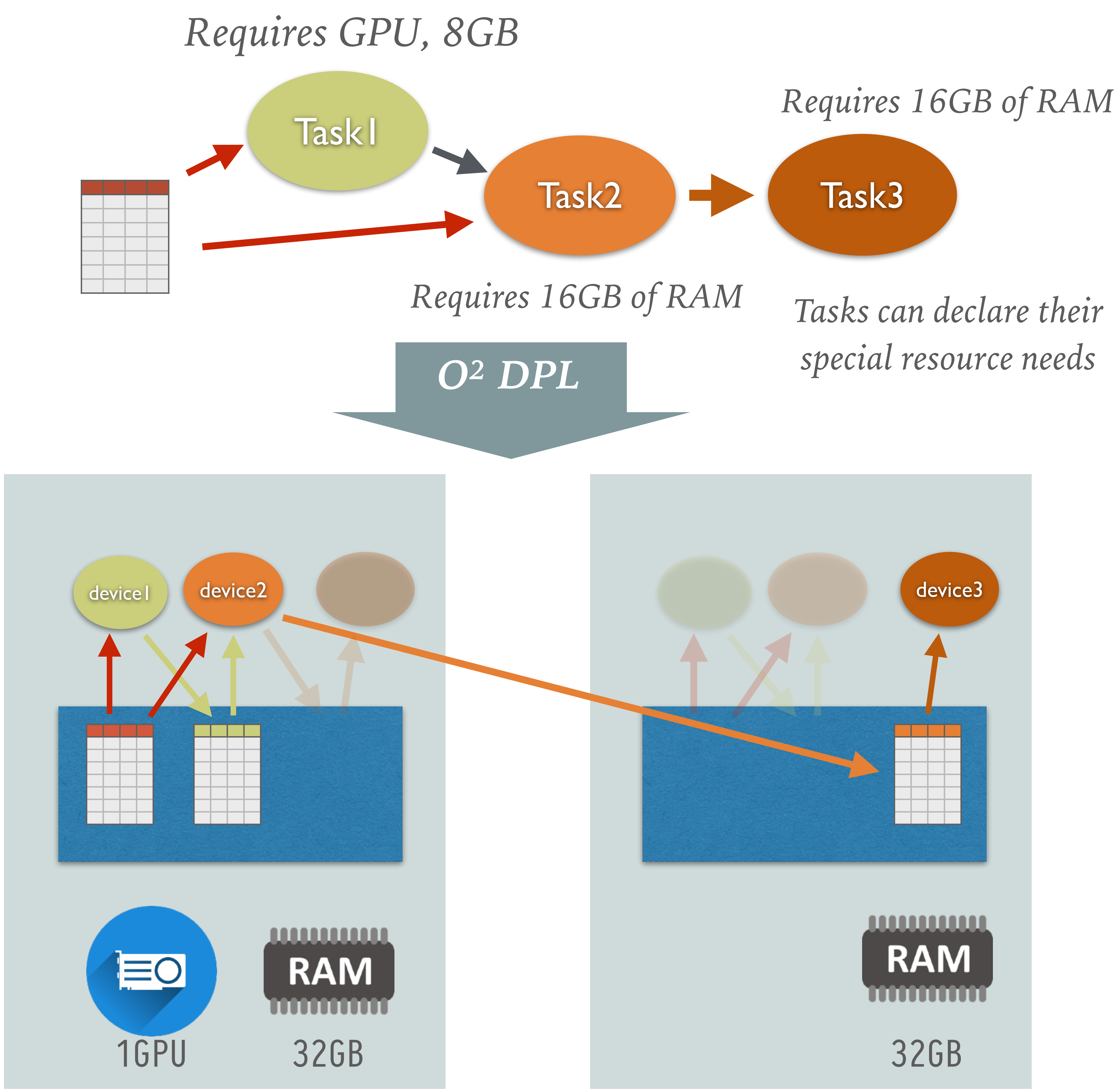
reconstruction-workflow | qc-workflow

Heterogeneous Computing Support

The mapping of an analysis workflow on top of a topology of message passing entities has the advantage to fit well physically / logically heterogeneous architectures.

Simple Multi Node support: *the current code can in particular already take advantage of multi-node setups (e.g. using Kubernetes ReplicaSet), without the need of an additional orchestrator entity. Each Replica knows the full topology and uses the same deterministic resource scheduling algorithm, resulting in seamless deployments for a low number of distinct nodes.*

Asymmetric nodes: *we are exploring using the same approach to model logically separated resources like GPU or NUMA.*



Resources can be either physically separated, or logically different domains within the same box.