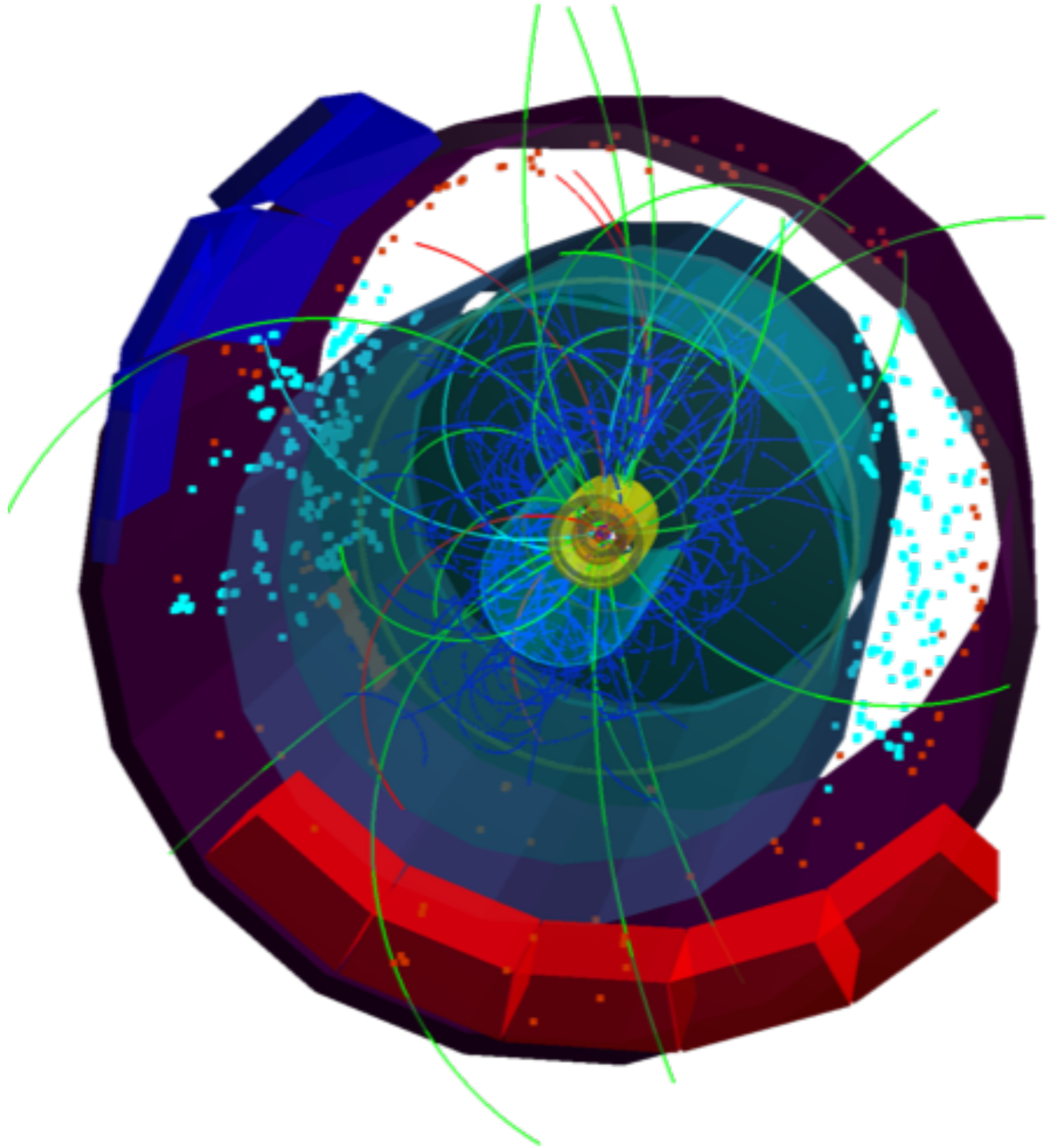
THE O[2] PROJECT

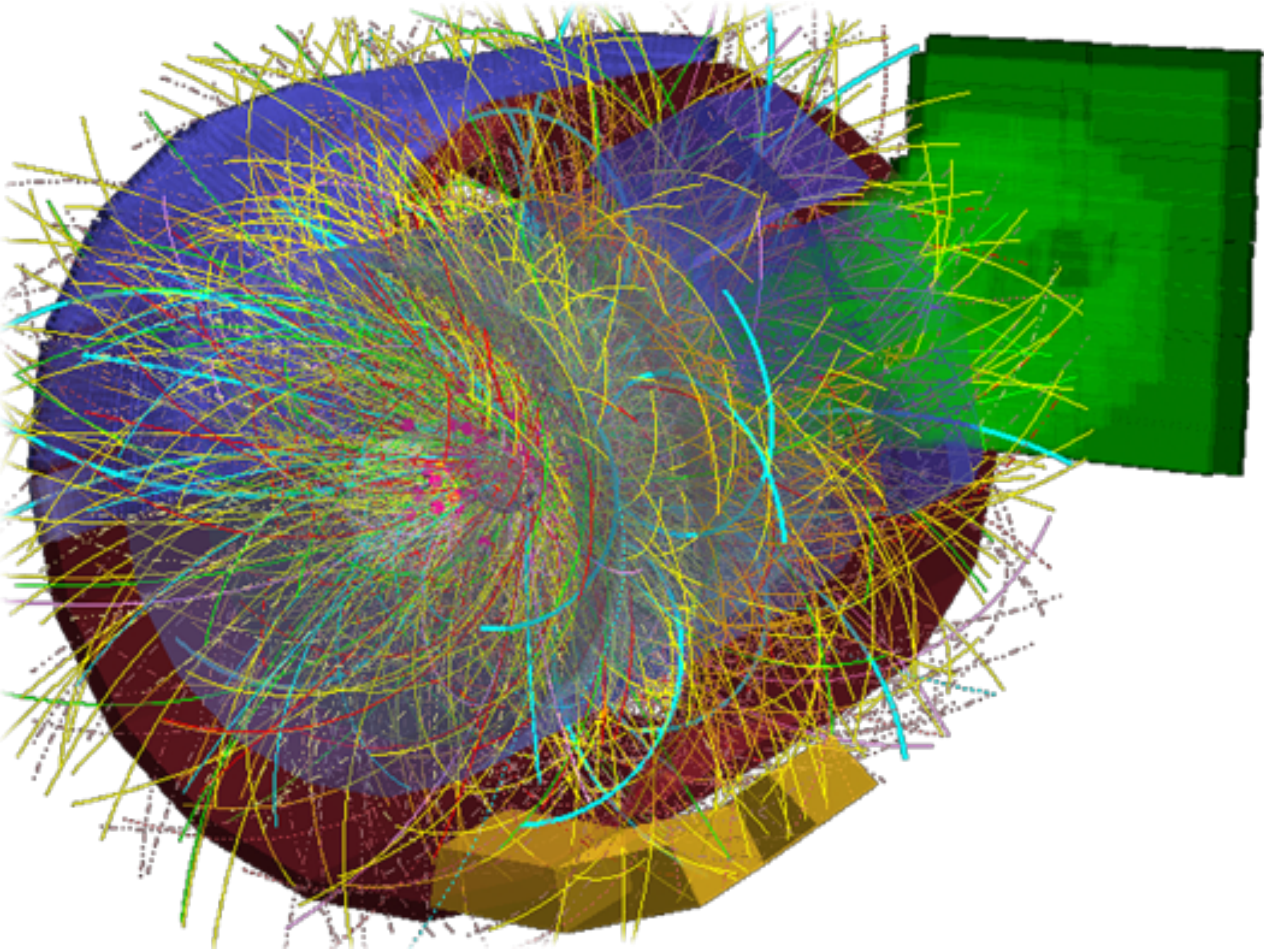
Giulio Eulisse (CERN)

ALICE IN RUN 2

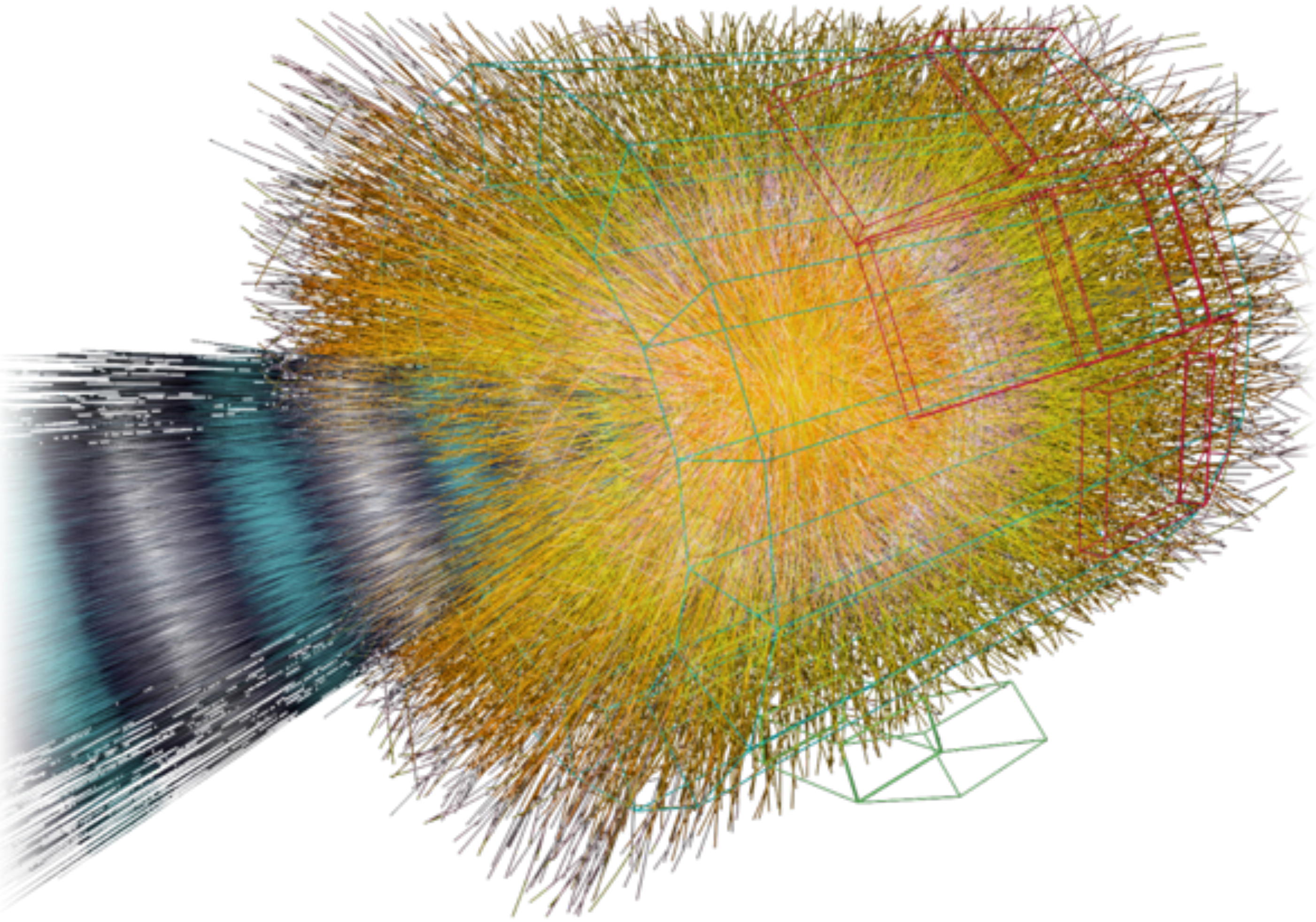
From O(1) kHz single events…



p - p



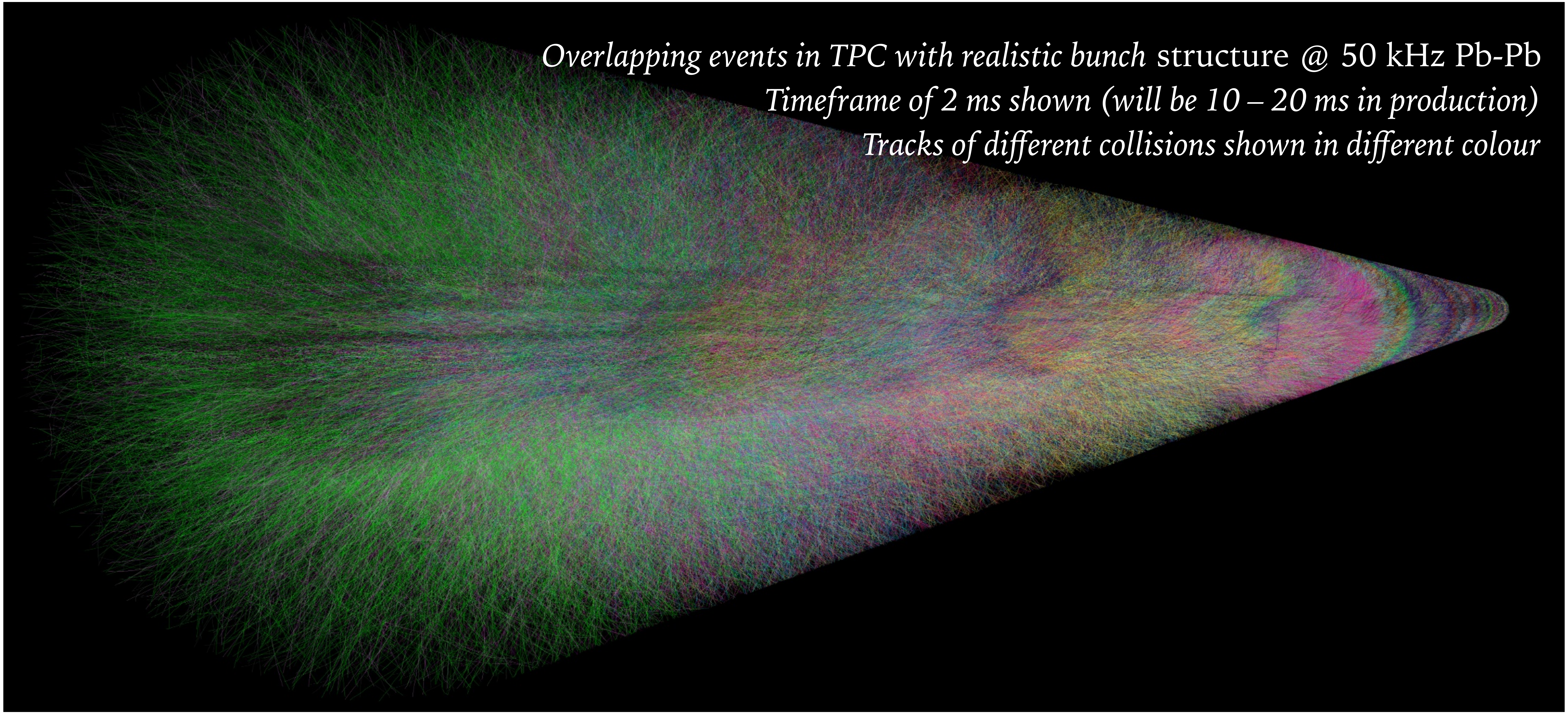
p - Pb



Pb - Pb

ALICE IN RUN 3

...to 50kHz of continuous readout data.



Challenges for ALICE in Run 3

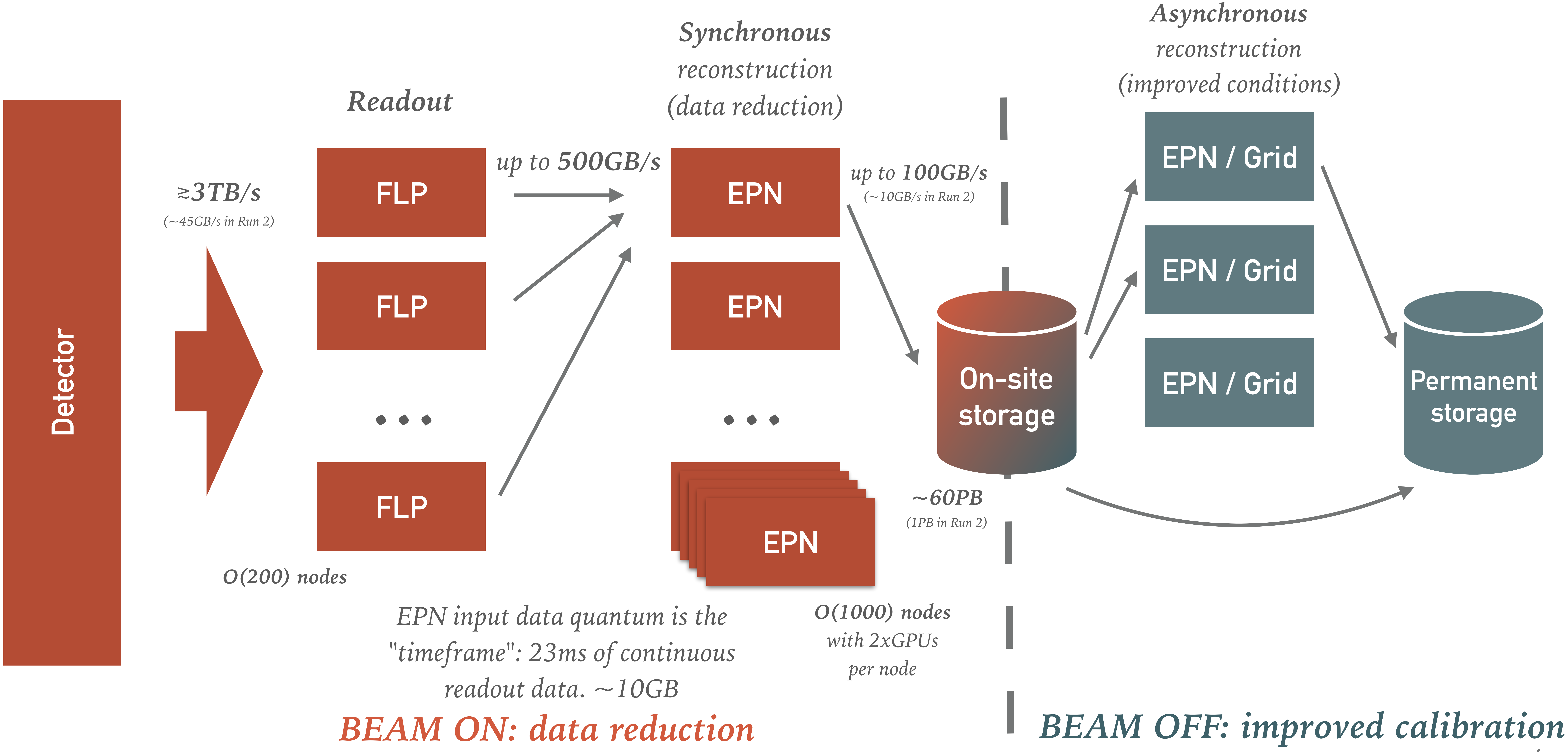
- ***Reconstruct 50x more*** *events online (minbias events only).*
- ***Store 50x*** *more events (needs TPC compression factor 20x compared to Run 2 raw data size)*
- *Reconstruct TPC data in* ***continuous readout*** *in combination with data coming from triggered detectors.*
- *All of the above while deploying a* ***completely new detector readout*** *and while performing* ***substantial upgrades to the detector itself*** *(new ITS, new GEM for TPC readout, ...).*
- *All of the above in a "flat budget" scenario...*

Designing a new Computing Architecture For ALICE in Run 3: ALICE O2

ALICE can cope with the challenges of Run 3 only by a radical redesign of its software and computing architecture.

- *New architecture based on the experience accumulated in the ALICE HLT during Run 1 / Run 2.*
- *Focus on online data compression, only keeping raw data and AOD objects, **trading computational cost for storage**.*
- ***Simplified Data Model** in order to improve I/O performance.*
- ***Appropriately chosen algorithms** providing higher throughput for negligible loss of physics performance. Algorithms tuned for vectorisation / exploitation of hardware accelerators (GPUs).*
- *Ability to port software components in a **gradual manner.***
- ***Close collaboration with the physics community** in order to organise analysis efforts.*
- ***Close collaboration with GSI and FAIR** experiments on a common software stack (**ALFA**).*

ALICE in Run 3: Point 2



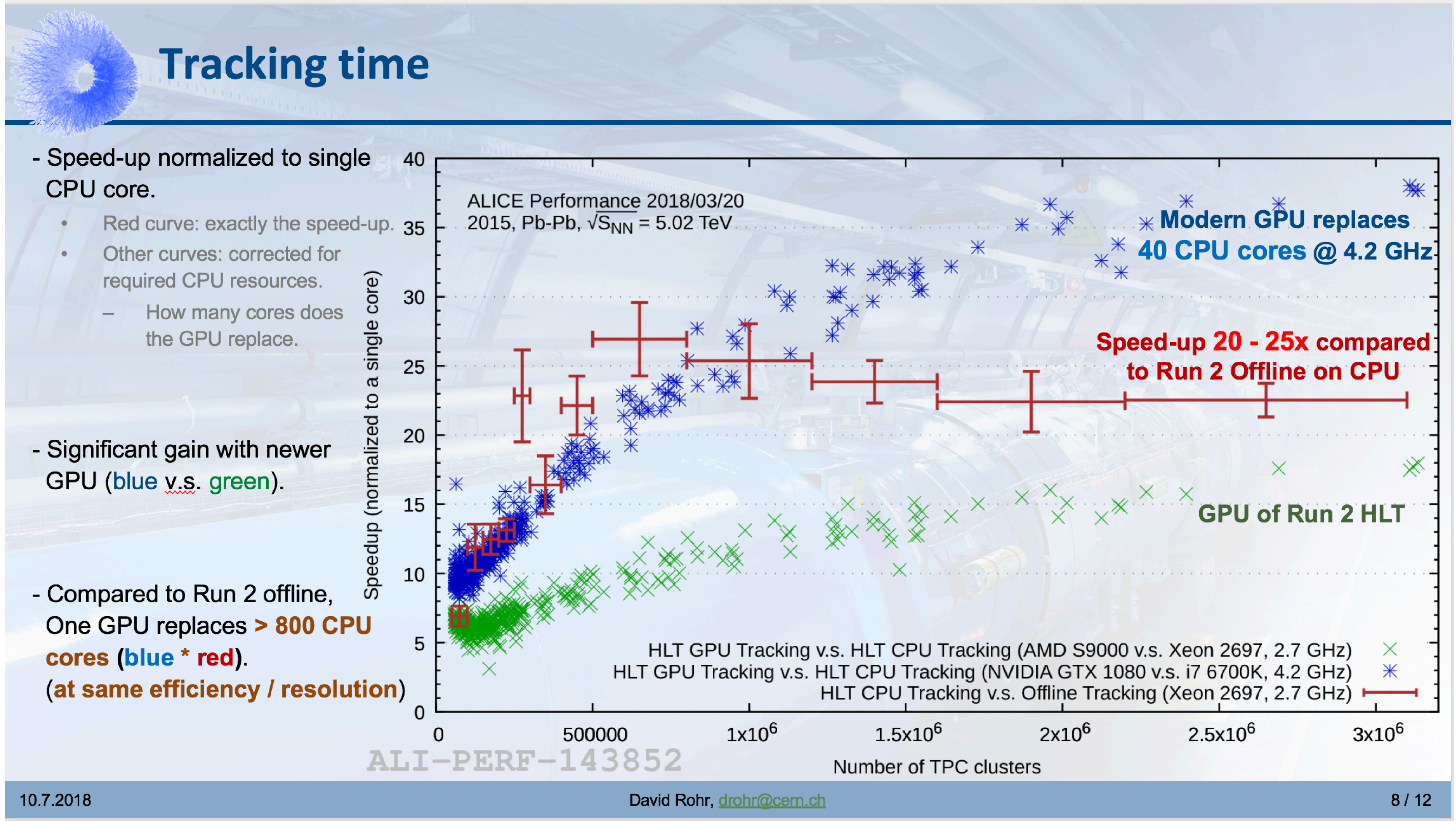
TIMEFRAME

Data quantum will not be the event, but the "Timeframe".

- *~23ms worth of data taking in continuous readout. Equivalent to 1000 collisions. Atomic unit.*
- *~10GB after timeframe building. Vast majority in TPC clusters.*
- *Compressed to ~2GB after asynchronous reconstruction, thanks to track-model-compression, storing clusters instead of ADC values, tailored fixed point integer format, logarithmic precision, entropy encoding.*
- *50x the number of collisions of Run 2.*
- *All MinBias. We need to (lossly) compress information, not filter it.*

Synchronous Reconstruction: GPUs as First Class Citizens

Synchronous processing will actually be possible thanks to GPU utilisation for TPC tracking. One modern GPU replaces 40 CPU cores. Changing the algorithm gives an additional 20x - 25x speedup with comparable quality.

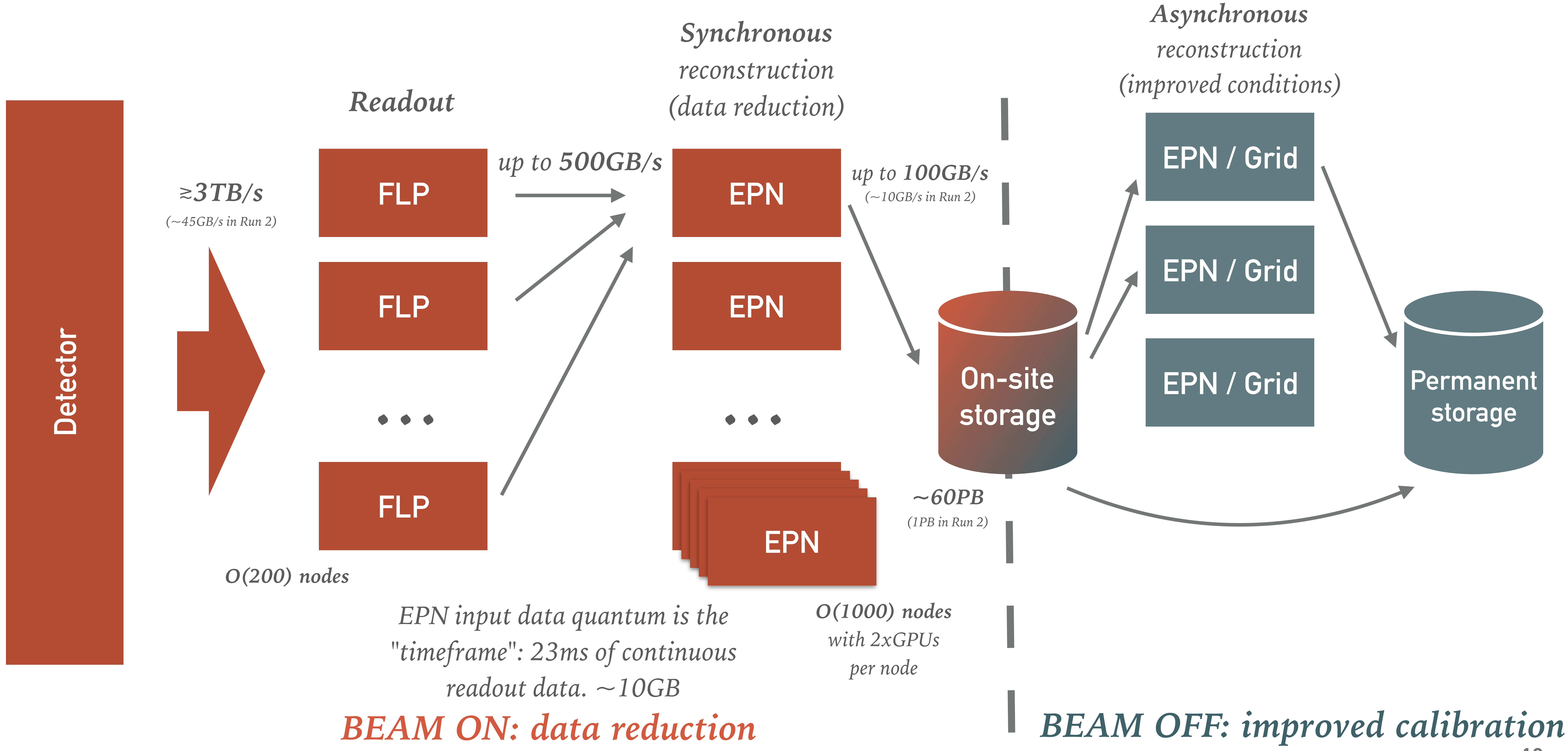


David Rohr
@CHEP 2018

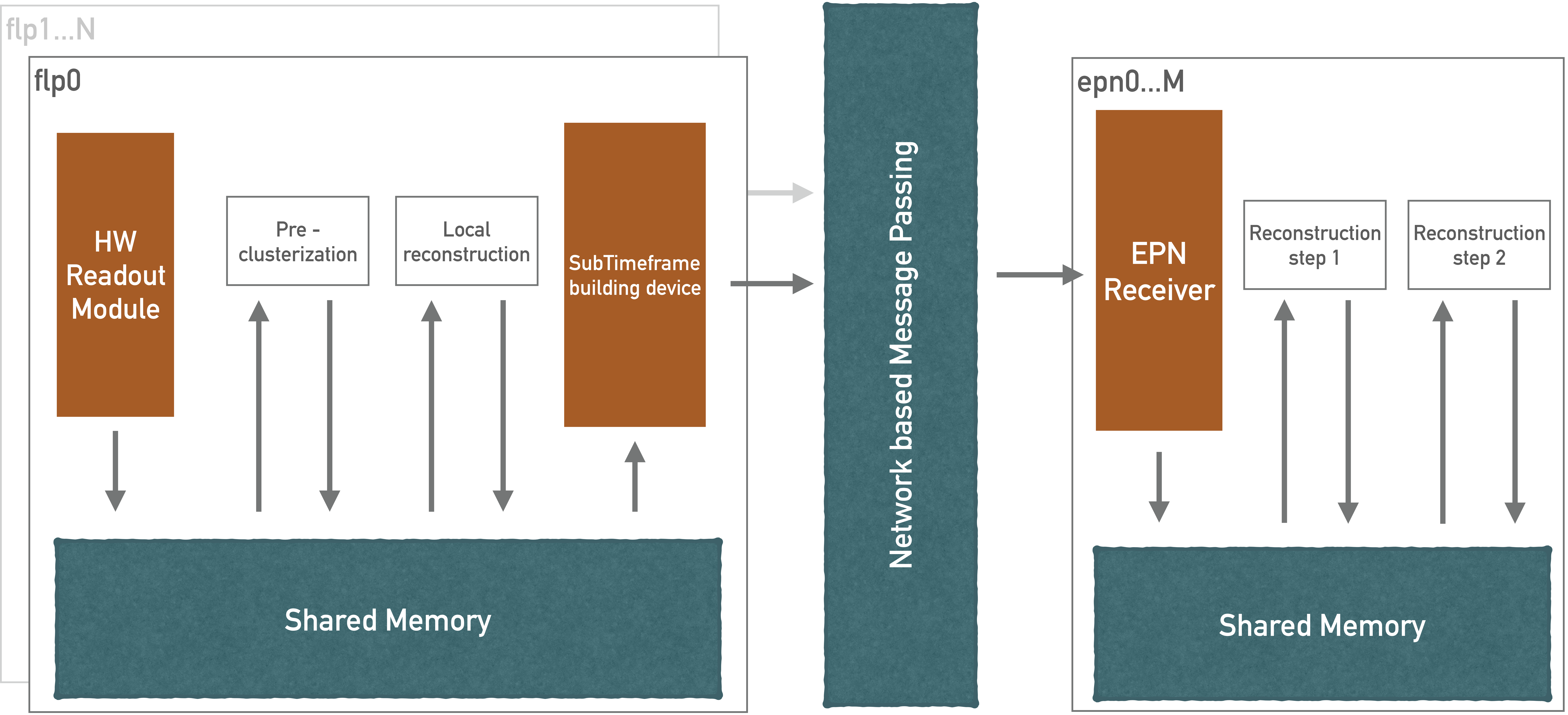
Asynchronous Reconstruction

- Follows the Data Taking period
- **2 processing cycles** per data taking year, with increasingly sophisticated calibration and improved reconstruction software.
- **Single persistent analysis object output - Analysis Object Data**
- Processing on O^2 Facility + T0 (70% of CTF volume), T1 (30% of CTF volume).
- **After 2nd cycle CTF will remain only on tape** (removed from the disk buffer) and any subsequent cycle will have to wait until LHC LS.

ALICE in Run 3: Point 2



Transport Level System Architecture



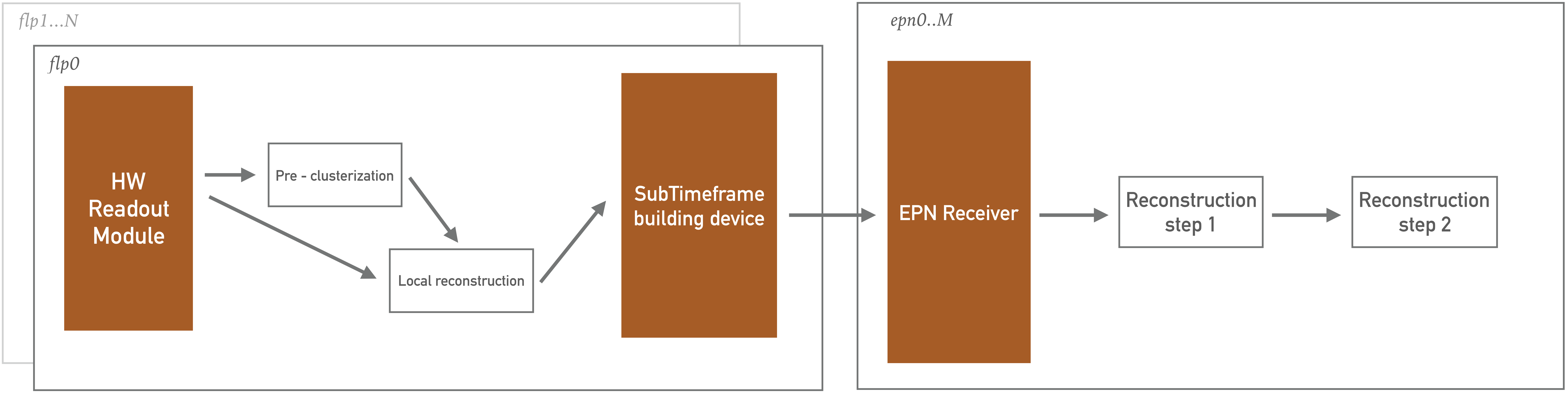
ALICE O2: Software Framework in one slide

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.*

Why FairMQ?

- **Separation of Concerns:** *From the architectural point of view, it allows ALICE to factor out data transport from the system description.*



- **Performant transport:** *collaboration with FAIR experiments and GSI allows sharing of highly skilled engineers to work on the performance critical parts related to transport.*

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.*

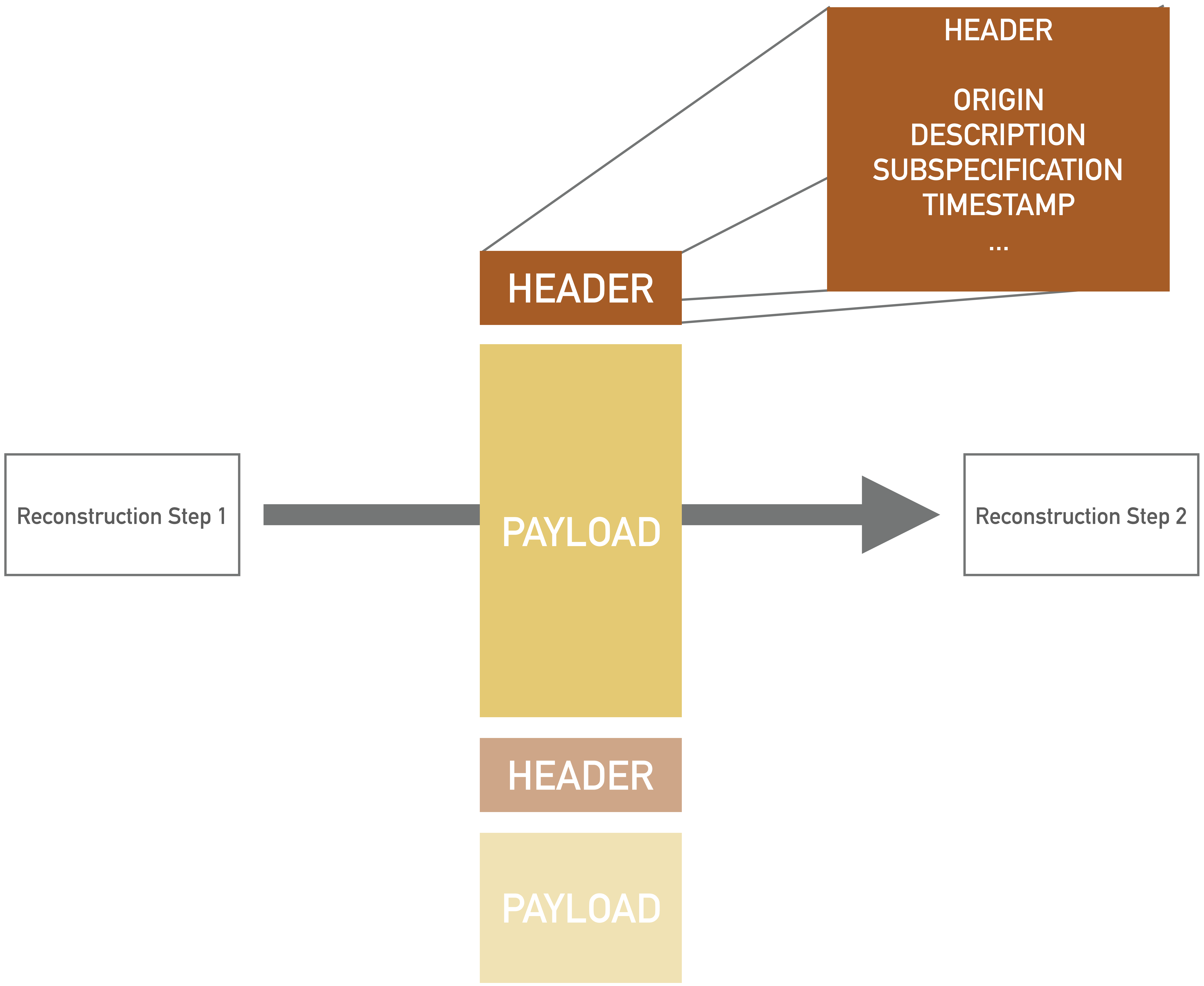
02 Data Model

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: **some columnar data format**. Multiple solutions being investigated.*

O2 Data Model



Messages being exchanged in O2 have a (header, payload) structure where the header describes the contents of the subsequent payload.

- ***Origin*** *represents the Detector or Component that first created the message (e.g. TPC)*
- ***Description*** *is the data type of the payload (e.g. CLUSTERS),*
- ***Subspecification*** *can be used to encode extra information (e.g. TPC sectors)*
- ***Timestamp*** */* ***Timerange*** *indicates the Timeframe it belongs to.*

ALICE O2: Software Framework in one slide

Data Layer: O2 Data Model

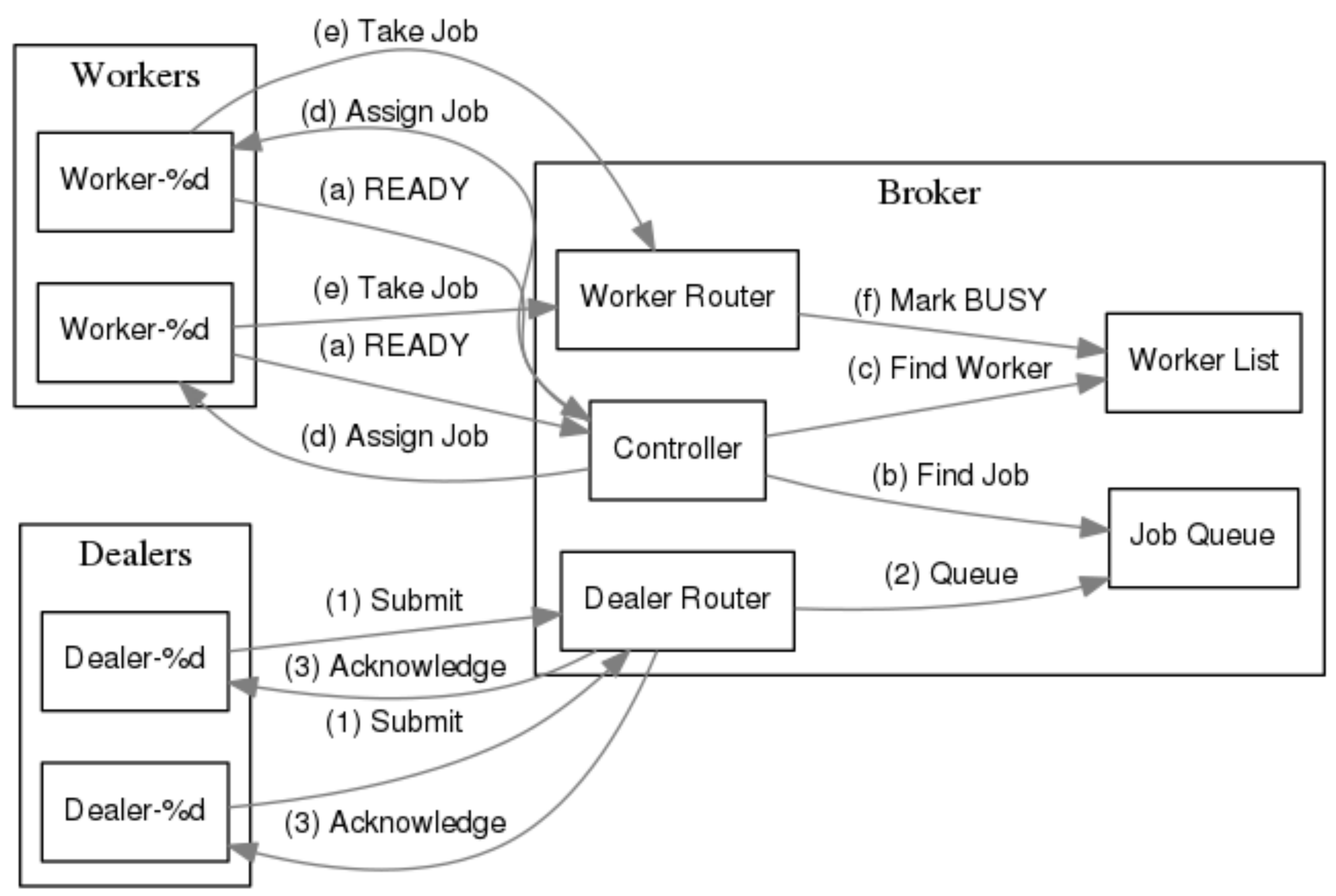
Message passing aware data model. Support for multiple backends:

- **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.*

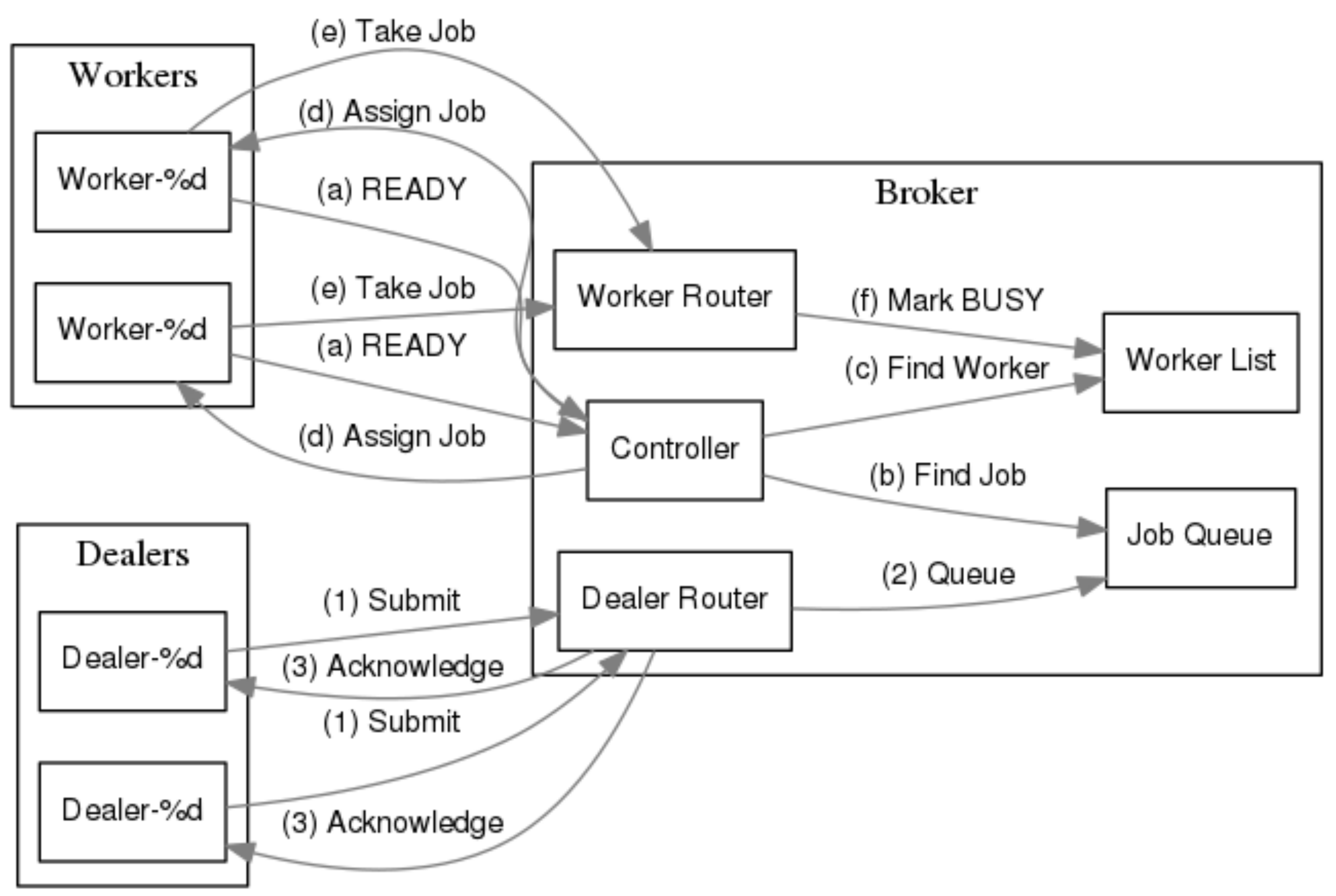
DISTRIBUTED SYSTEMS ARE HARD

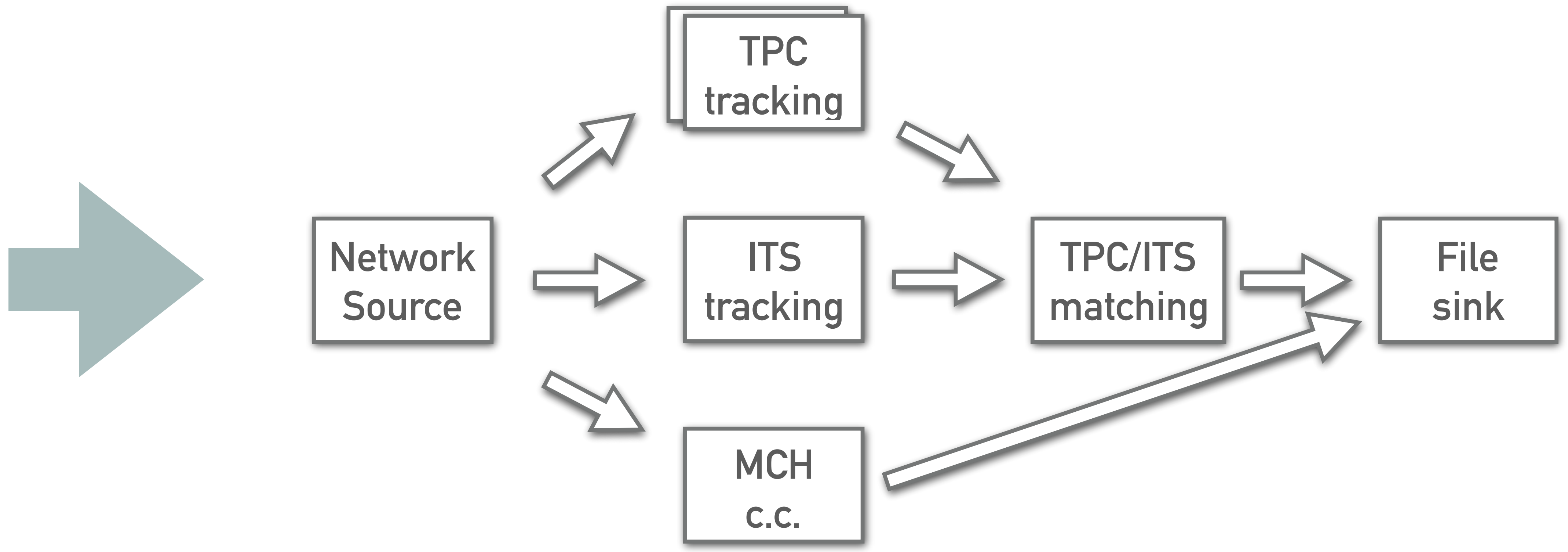


There are only **two hard problems** in distributed systems:

2. Exactly-once delivery
1. Guaranteed order of messages
2. Exactly-once delivery

Distributed Systems Are Hard





There are only **two hard problems** in distributed systems:

2. Exactly-once delivery
1. Guaranteed order of messages
2. Exactly-once delivery

Since too many people did not get the joke, we started thinking how to simplify this for the user, as a result we decided to build a **data flow engine (pipelines!)** on top of our distributed system backend.

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

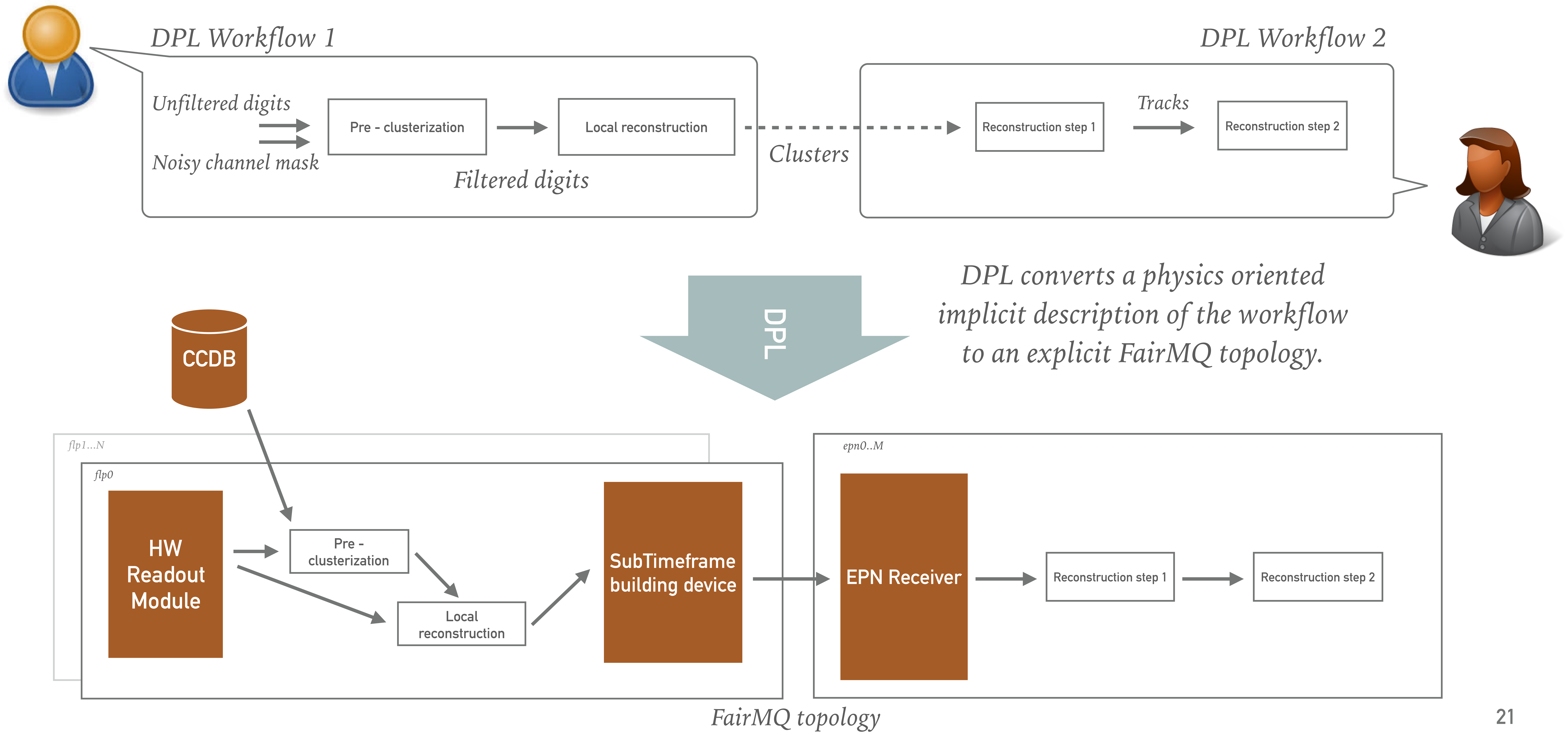
Message passing aware data model. Support for multiple backends:

- **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 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.*

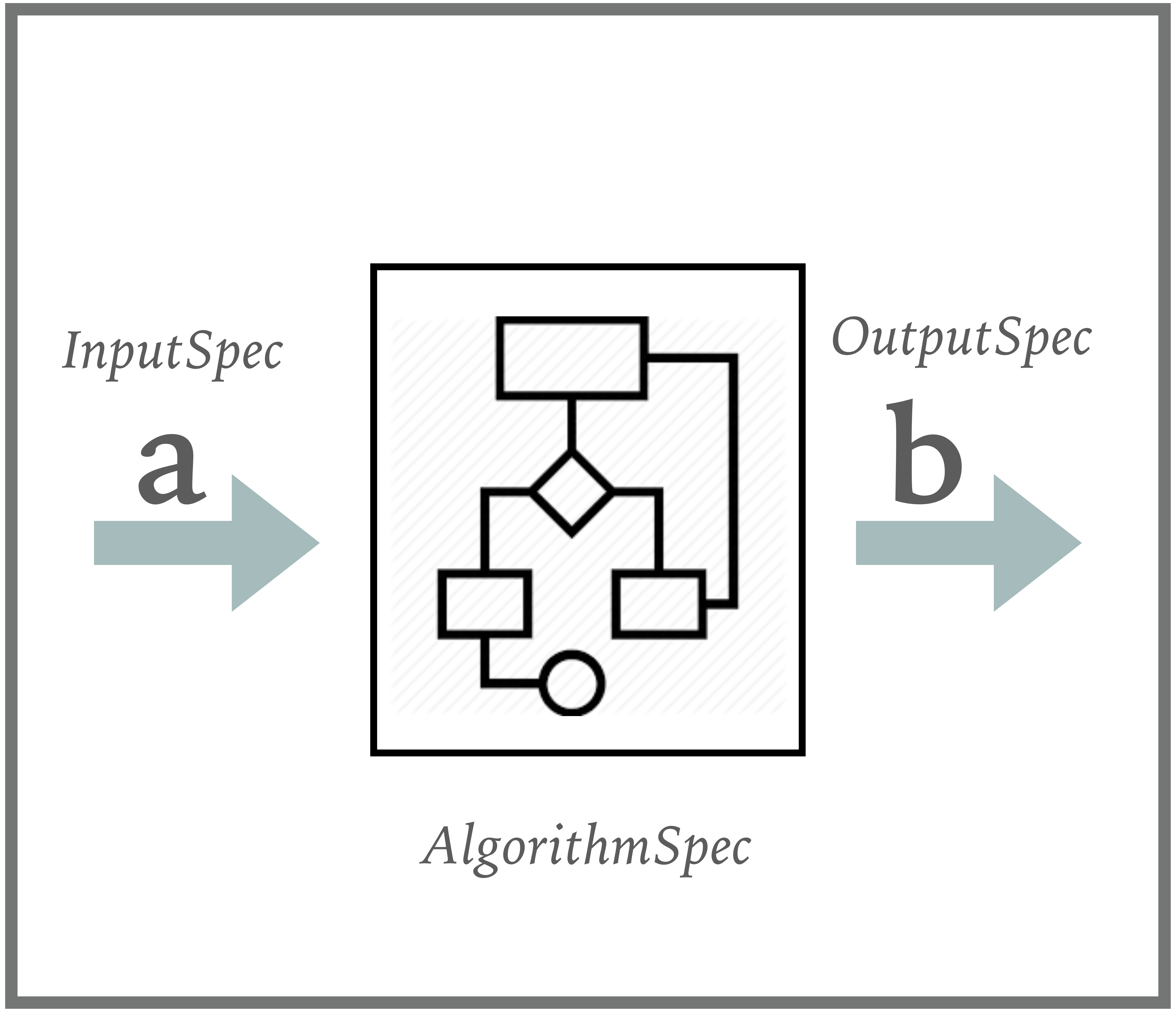
DPL: Implicit Workflow Definition



DPL: Building Block

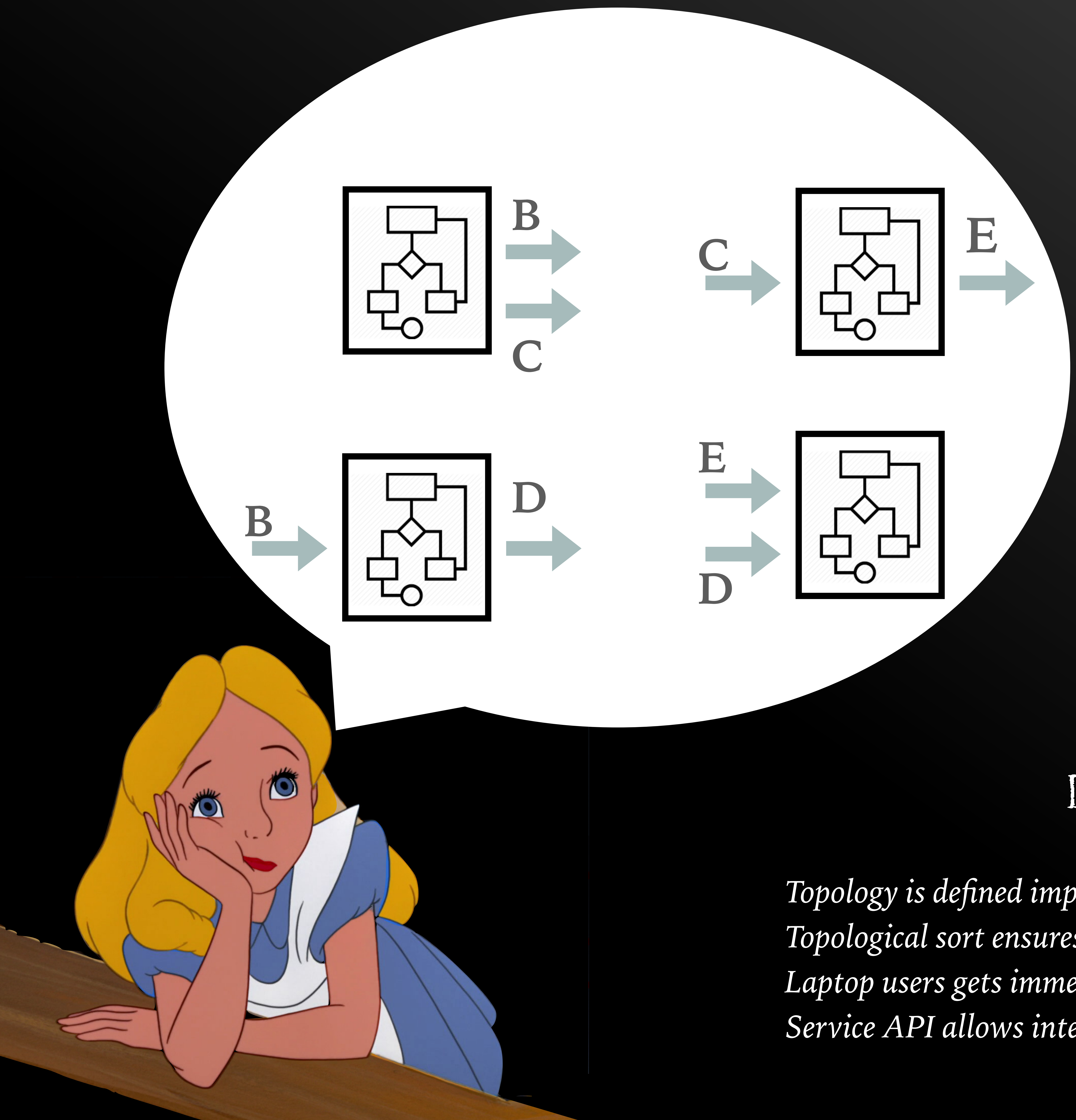
A **DataProcessorSpec** *defines a pipeline stage as a building block.*

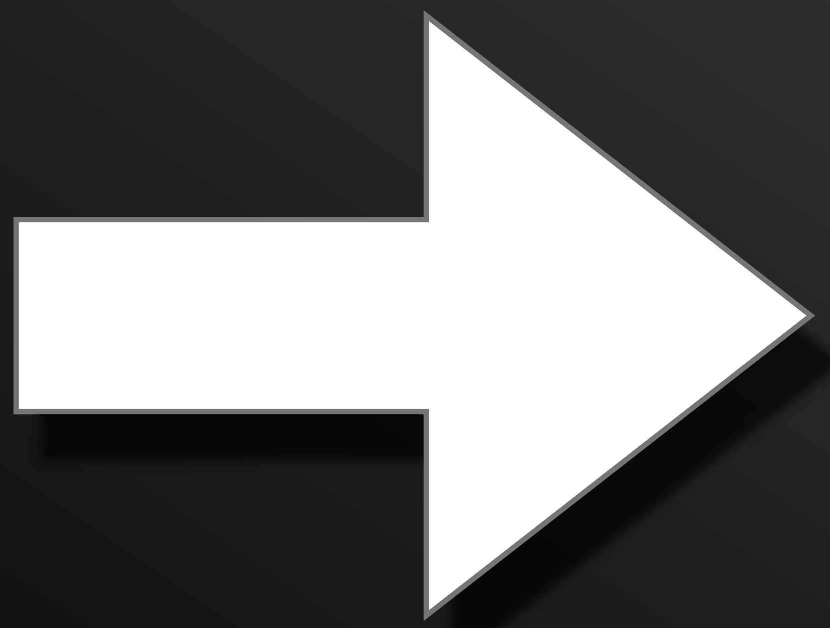
- *Specifies* ***inputs and outputs*** *in terms of the O2 Data Model descriptors.*
- *Provide an implementation of how to act on the inputs to produce the output.*
- *Advanced user can express possible data or time parallelism opportunities.*

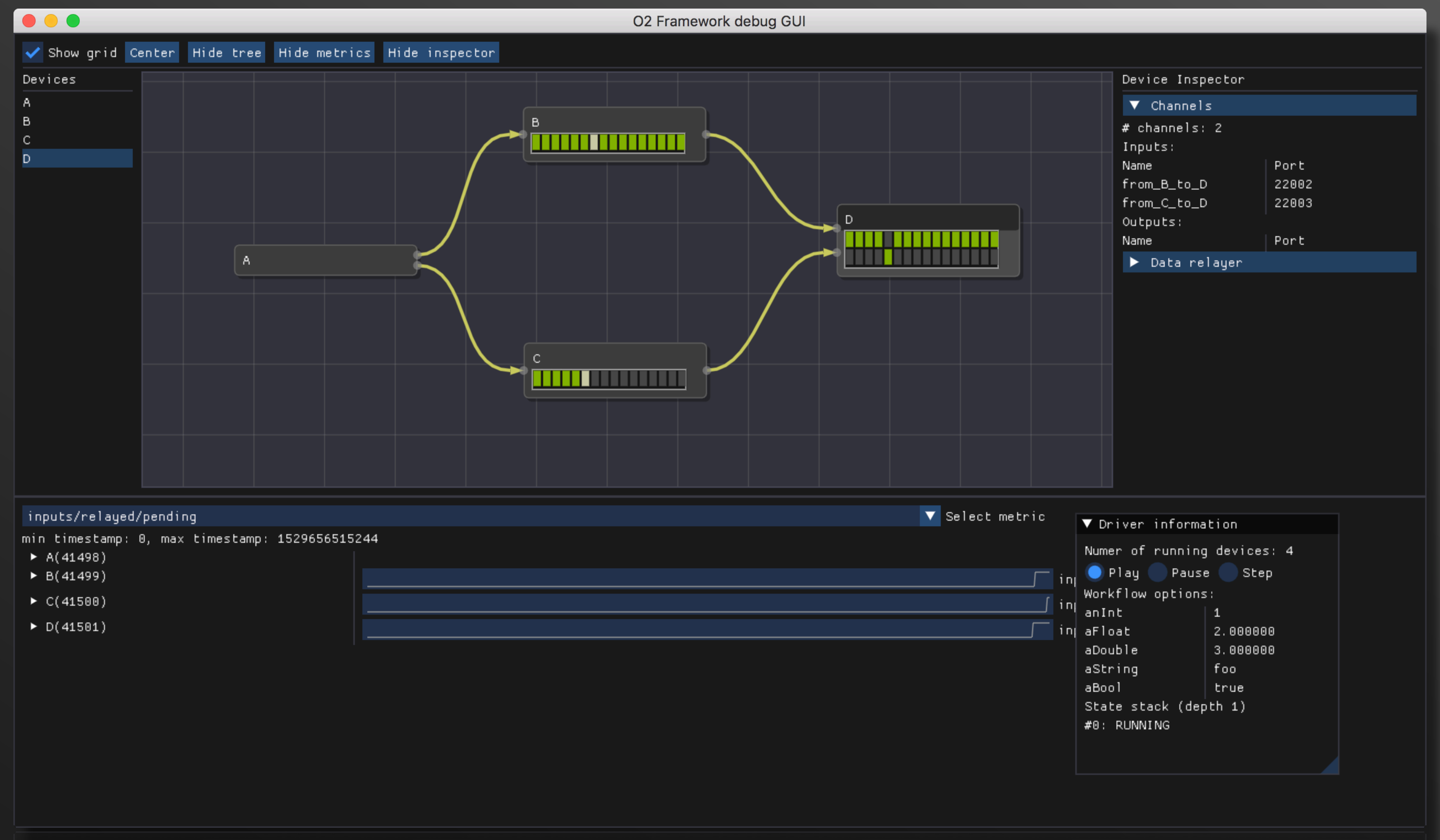


DataProcessorSpec

DATA PROCESSING LAYER: IMPLICIT TOPOLOGY

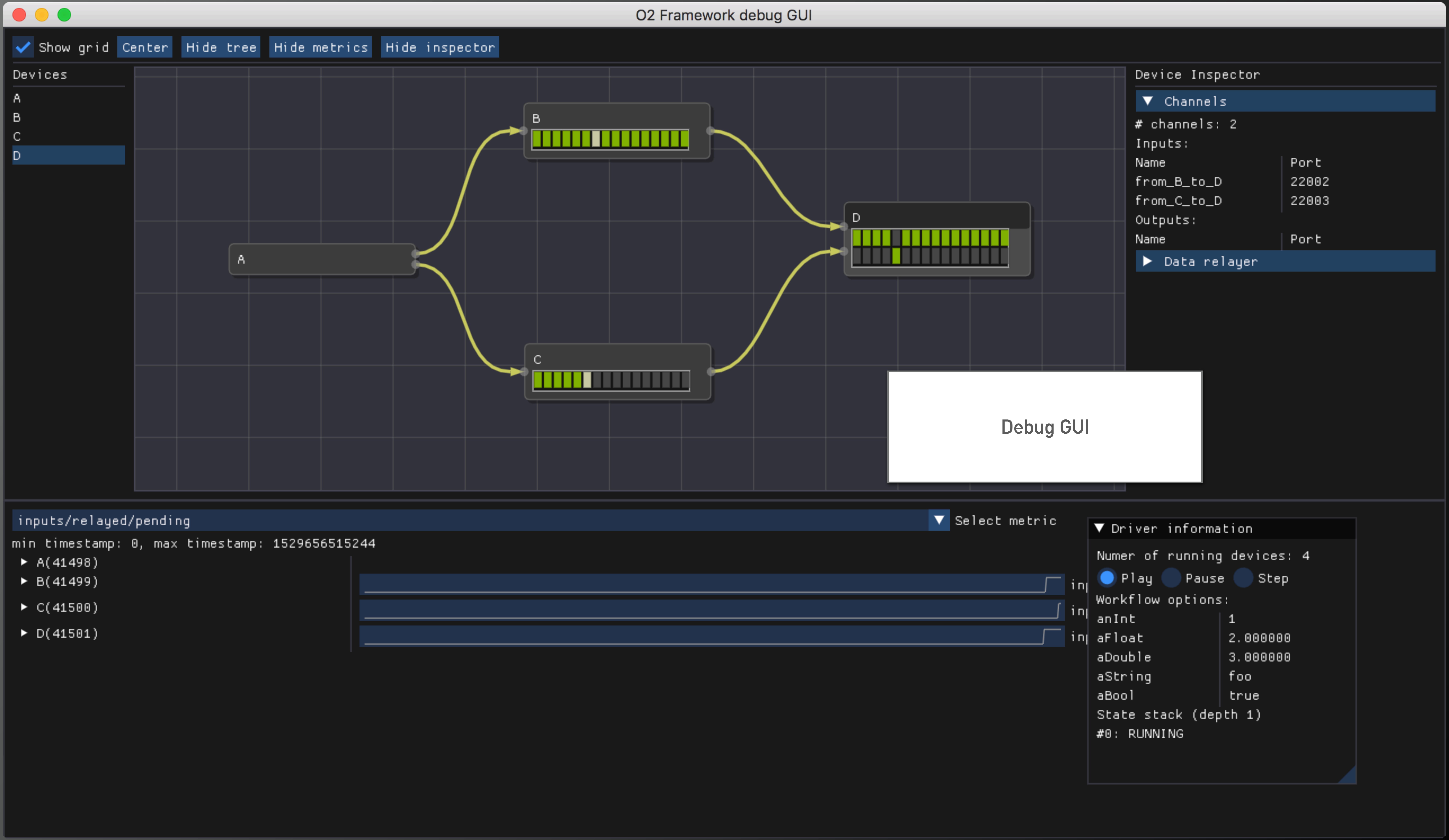


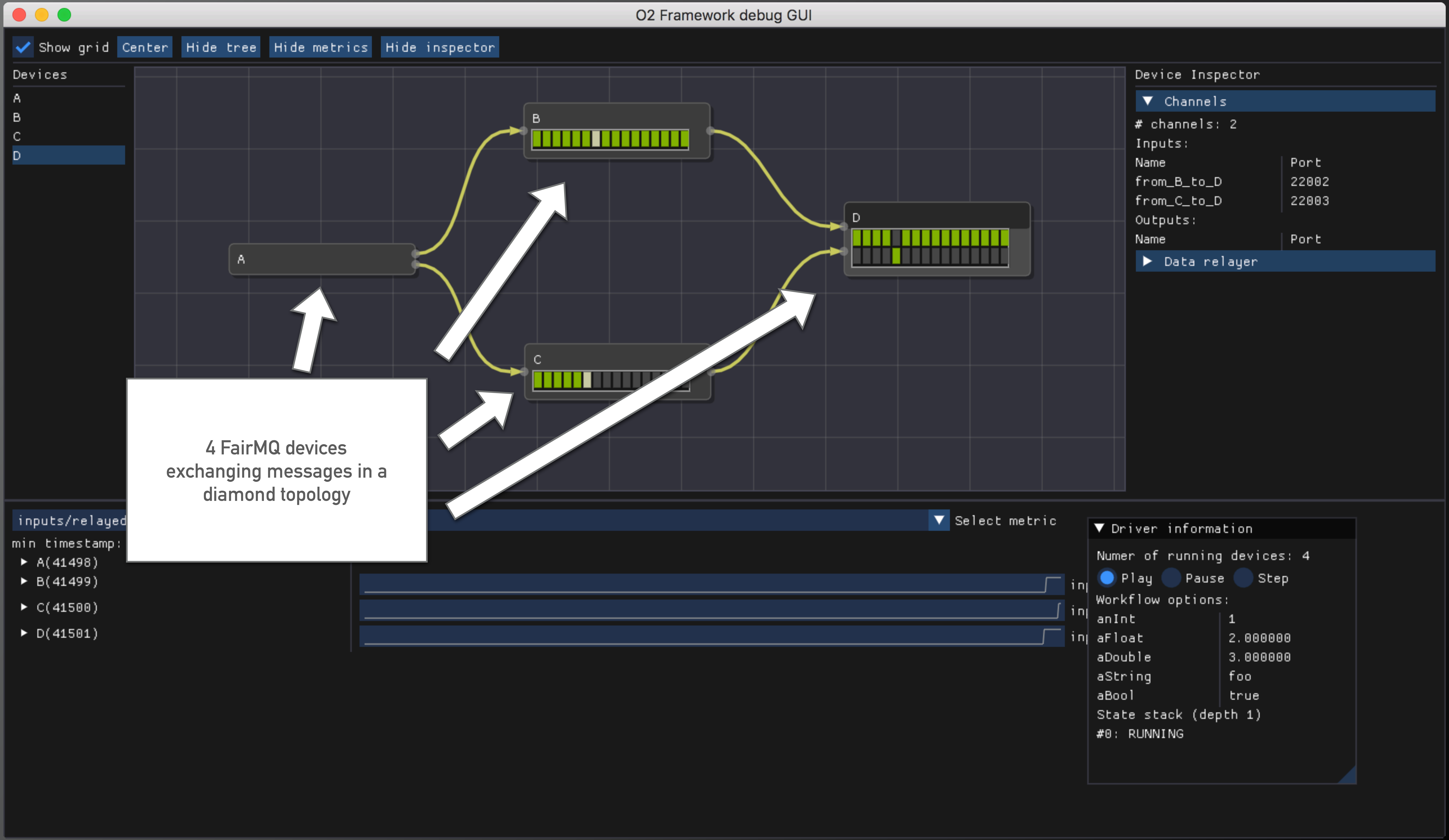


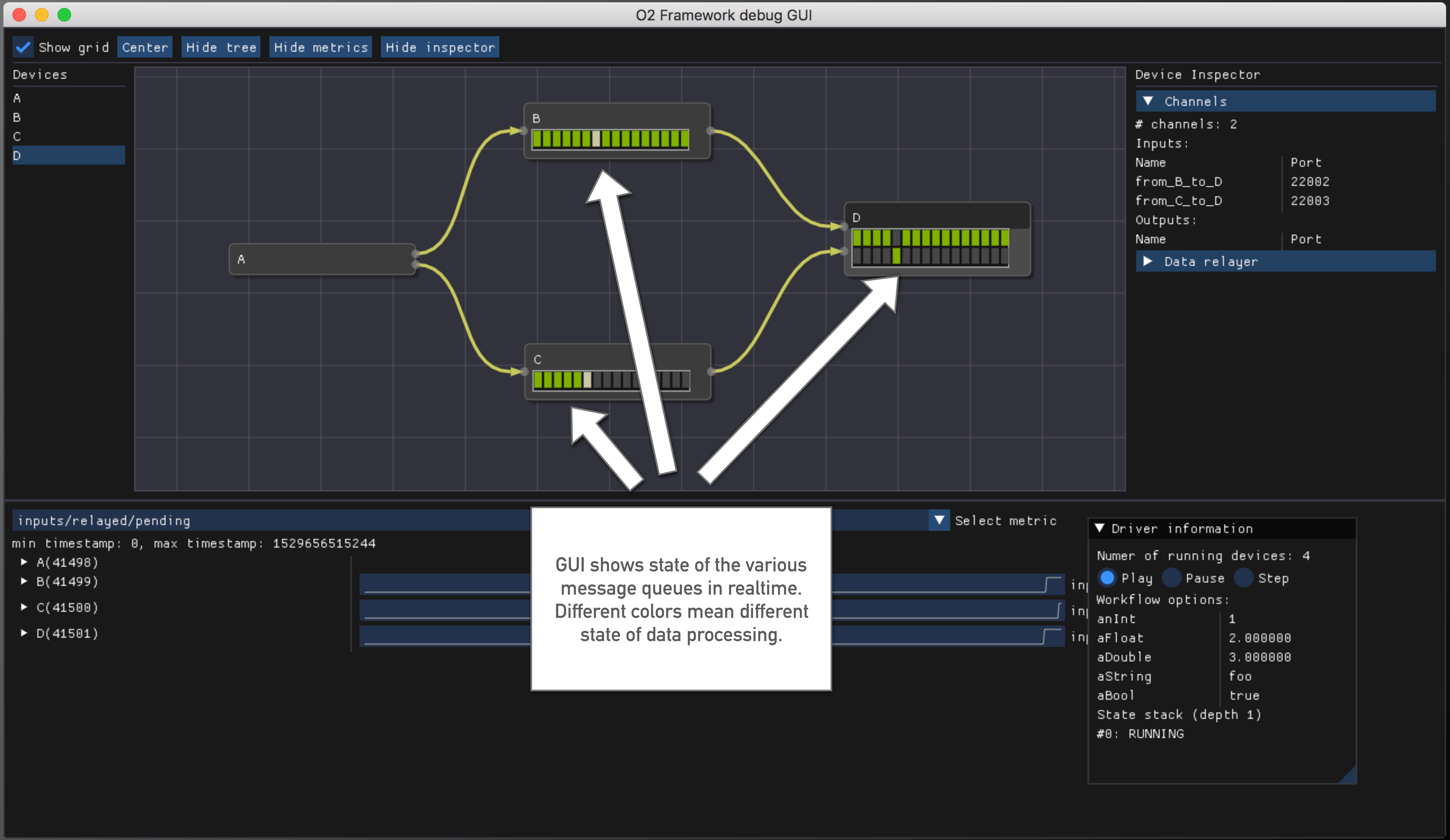


Data Processing Layer

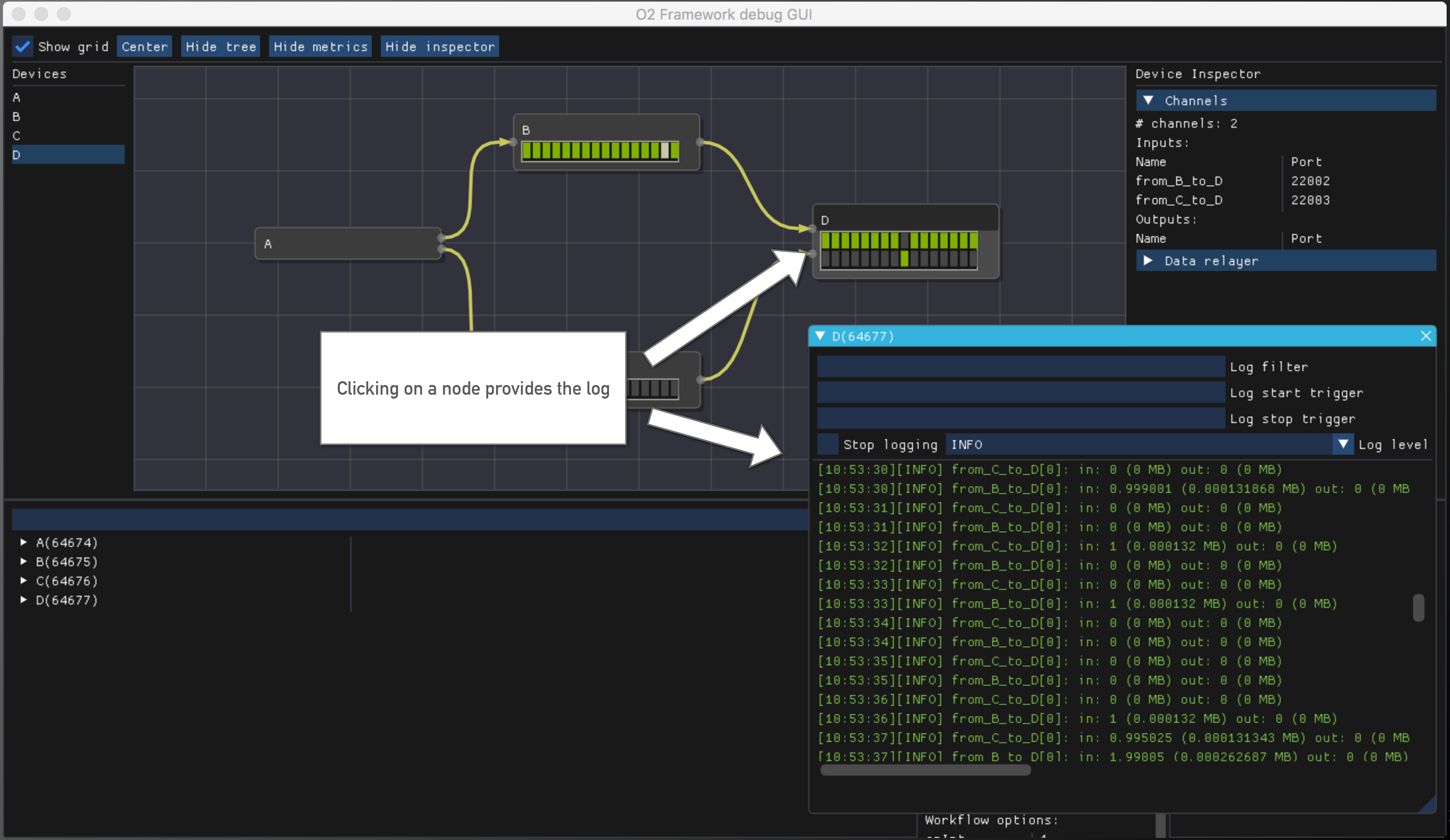
Topology is defined implicitly.
Topological sort ensures a viable dataflow is constructed (no cycles!).
Laptop users gets immediate feedback through the debug GUI.
Service API allows integration with non data flow components (e.g. Control)

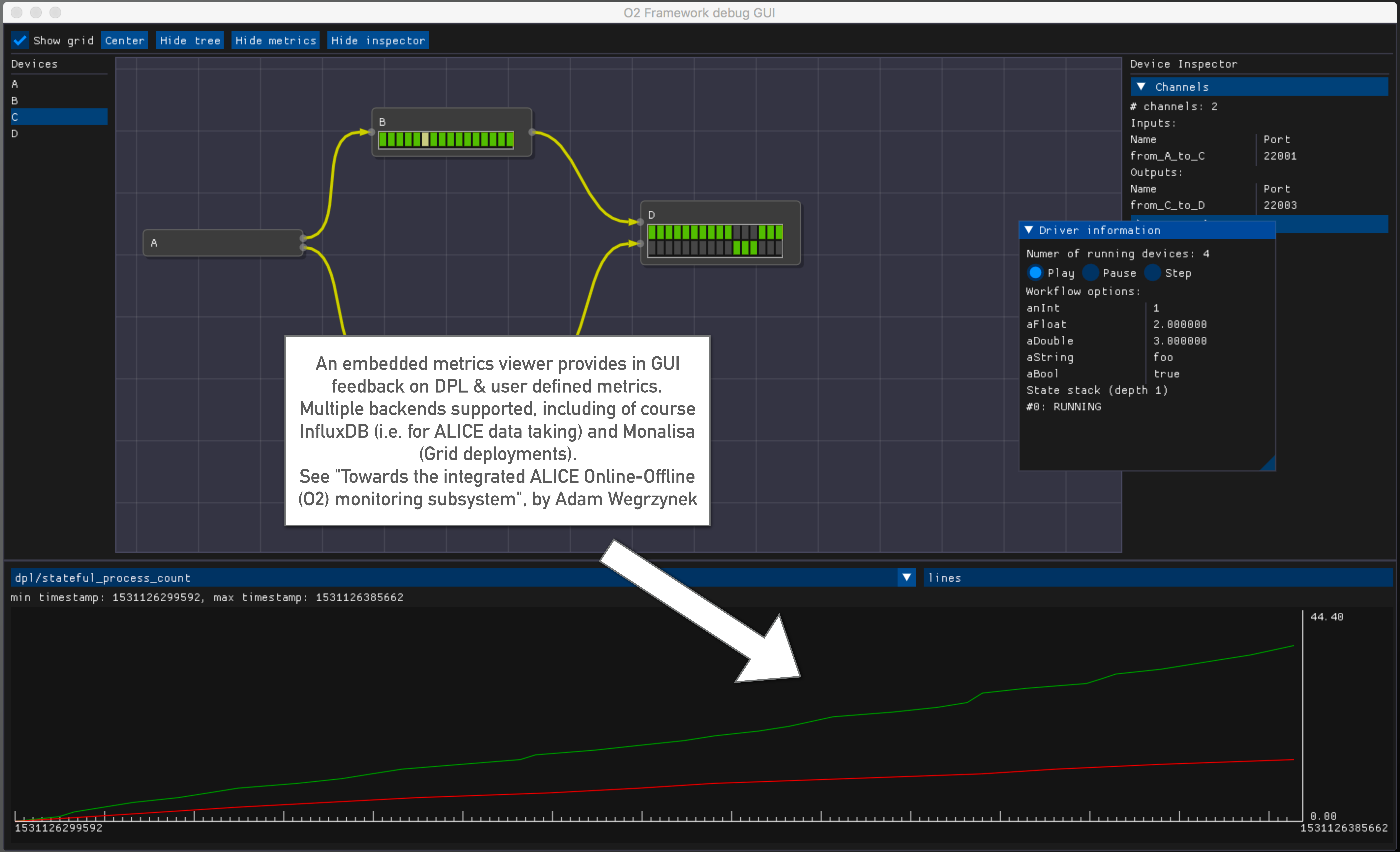

O2 Framework debug GUI
Show grid
Center
Hide tree
Hide metrics
Hide inspector
Devices
A
B
C
D
Device Inspector
Channels
channels: 2
Inputs:
Name
Port
from_B_to_D
22002
from_C_to_D
22003
Outputs:
Name
Port
Data relayer
inputs/relayed/pending
Select metric
min timestamp: 0, max timestamp: 1529656515244
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
B
D
A
Clicking on a node provides the log
Device Inspector
Channels
channels: 2
Inputs:
Name
Port
from_B_to_D
22002
from_C_to_D
22003
Outputs:
Name
Port
Data relayer
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:

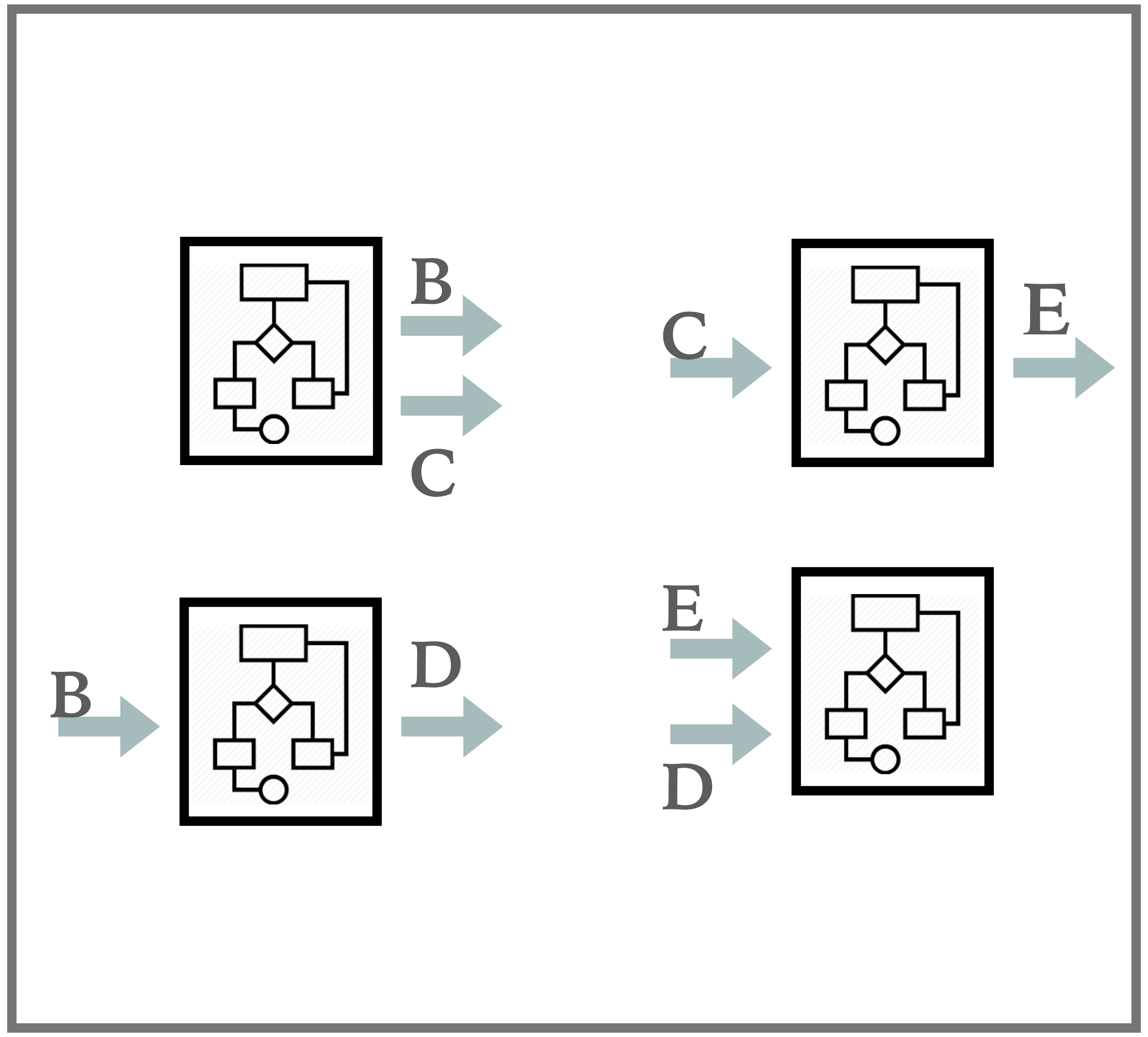


An embedded metrics viewer provides in GUI feedback on DPL & user defined metrics. Multiple backends supported, including of course InfluxDB (i.e. for ALICE data taking) and Monalisa (Grid deployments).
See "Towards the integrated ALICE Online-Offline (O2) monitoring subsystem", by Adam Wegrzynek

```
#include "Framework/runDataProcessing.h"

using namespace o2::framework;

AlgorithmSpec simplePipe(std::string const &what) {
  return AlgorithmSpec{ [what](ProcessingContext& ctx) {
    auto bData = ctx.outputs().make<int>(OutputRef{what}, 1);
  } };
}

WorkflowSpec defineDataProcessing(ConfigContext const&specs) {
  return WorkflowSpec{
  { "A", Inputs{}, {OutputSpec{{"a1"}, "TST", "A1"}, OutputSpec{{"a2"}, "TST", "A2"}},
    AlgorithmSpec{
      [](ProcessingContext &ctx) {
       auto aData = ctx.outputs().make<int>(OutputRef{ "a1" }, 1);
       auto bData = ctx.outputs().make<int>(OutputRef{ "a2" }, 1);
      }
    }
  },
  { "B", {InputSpec{"x", "TST", "A1"}}, {OutputSpec{{"b1"}, "TST", "B1"}}, simplePipe("b1")},
  { "C", {InputSpec{"x", "TST", "A2"}}, {OutputSpec{{"c1"}, "TST", "C1"}}, simplePipe("c1")},
  { "D", {InputSpec{"b", "TST", "B1"}, InputSpec{"c", "TST", "C1"}}, Outputs{},
    AlgorithmSpec{[](ProcessingContext &ctx) {}}
  }
  };
}
```

The previous example (GUI included) requires 27 user's SLOC.

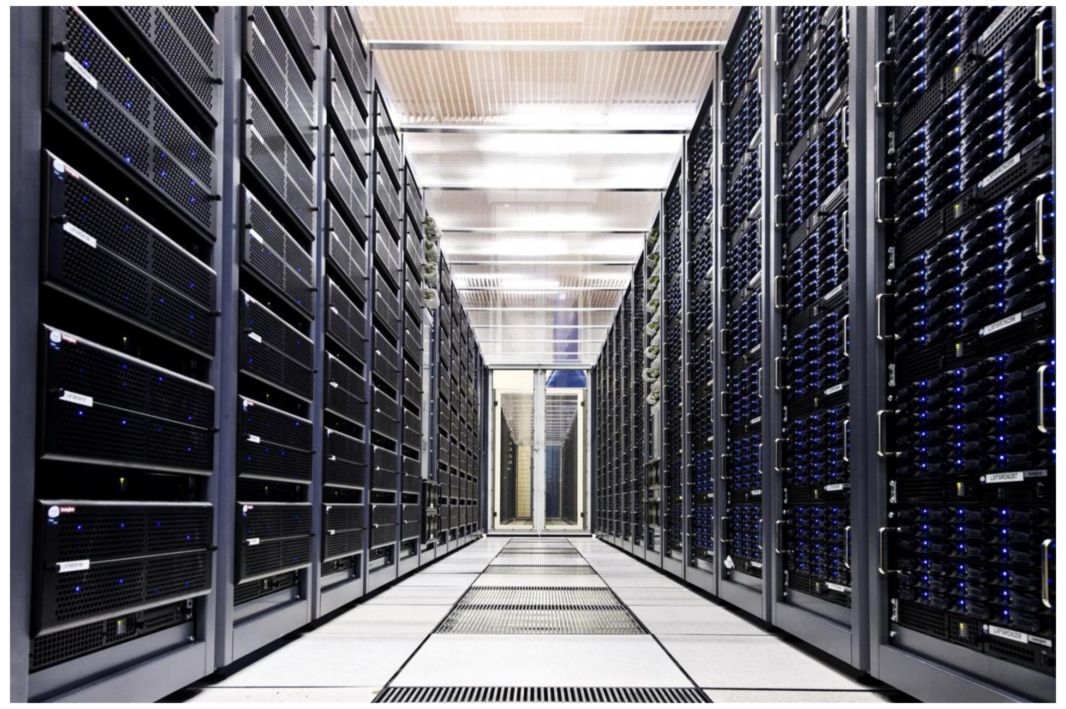
Support for multiple deployment strategies.



Compiles into a single executable for the laptop user.

```
<topology id="o2-dataflow">
 <decltask id="A">
  <exe reachable="true">../bin/o2DiamondWorkflow --id A ...</exe>
 </decltask>
 <decltask id="B">
  <exe reachable="true">../bin/o2DiamondWorkflow --id B ...</exe>
 </decltask>
 <decltask id="C">
  <exe reachable="true">../bin/o2DiamondWorkflow --id C ...</exe>
 </decltask>
 <decltask id="D">
   <exe reachable="true">../bin/o2DiamondWorkflow --id D ...</exe>
 </decltask>
</topology>
```

Generates DDS configuration for deployment on a farm.



Integration with O2 Control system.

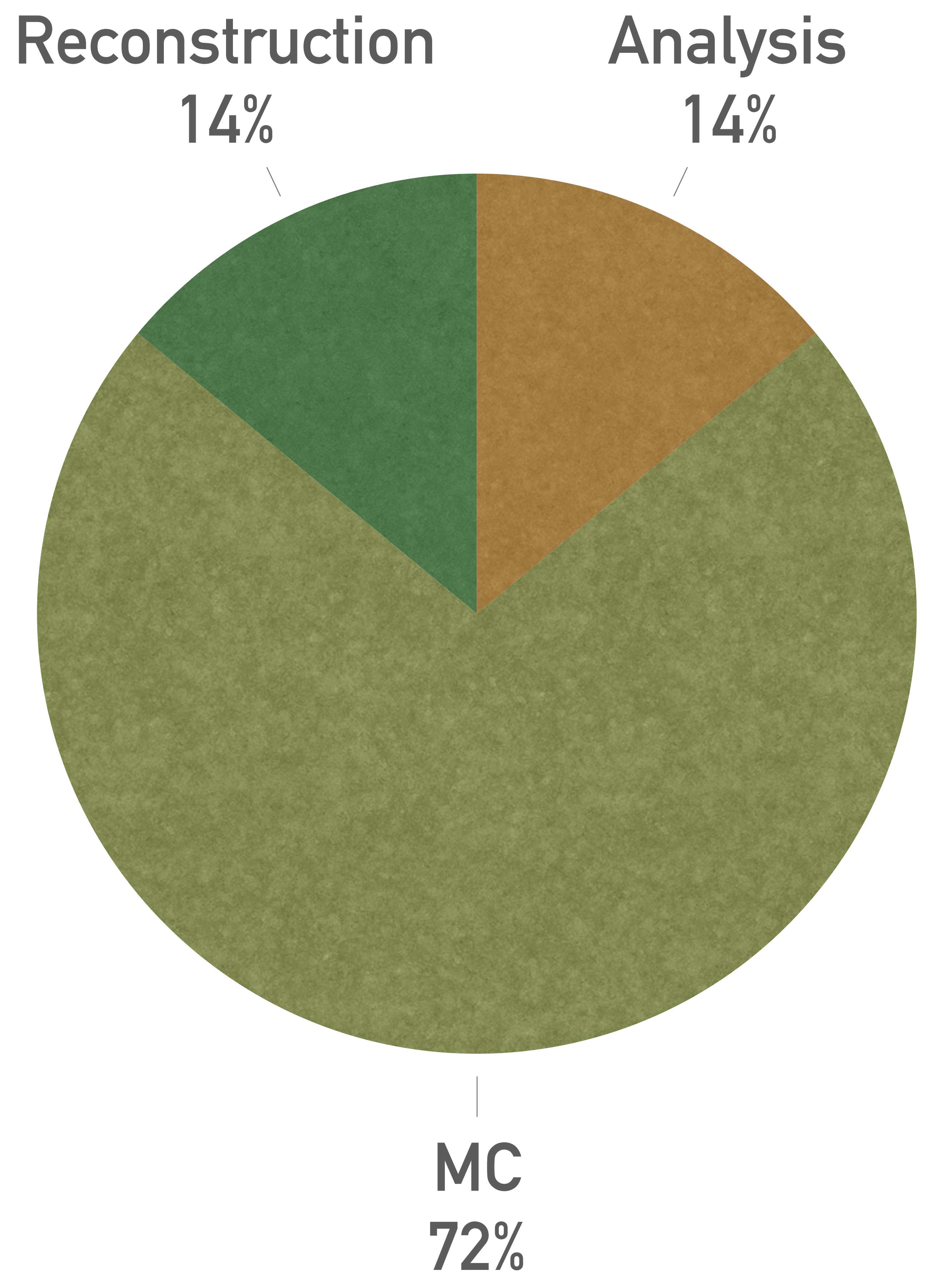
Analysis Model: Run 2

In order to offset the costs of reading data, ALICE has as strong tradition of organised analysis (i.e. trains):

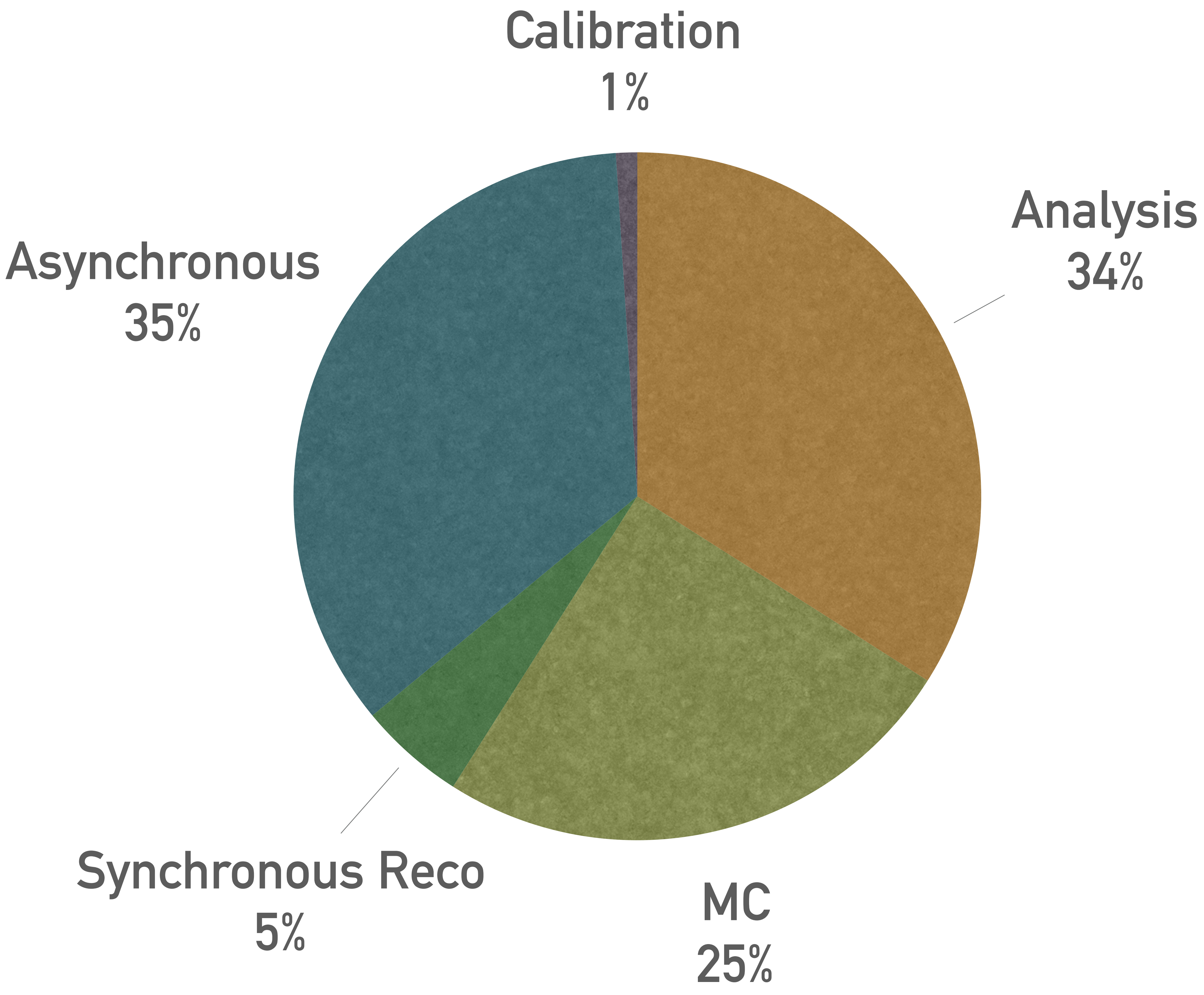
- Users provide "wagons", organised in "trains". Trains run on the Grid.
- Data is read only once per train, wagons get applied to it.
- Data is kept in a generic C++ object store, backed by ROOT, as you know.
- Slow sites / site issues is what dominates performance.

Resource Share Projection

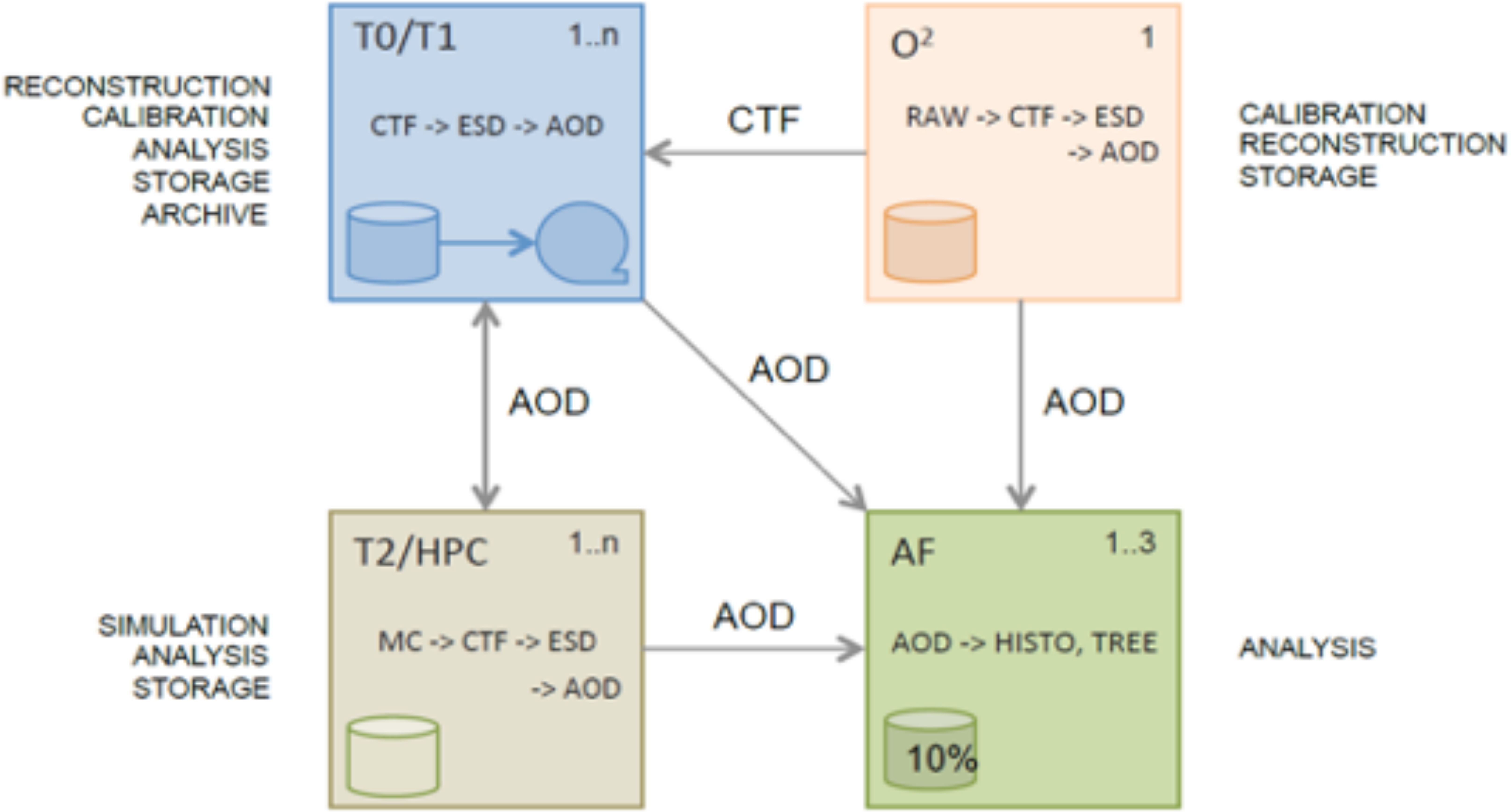
Today



Run 3+



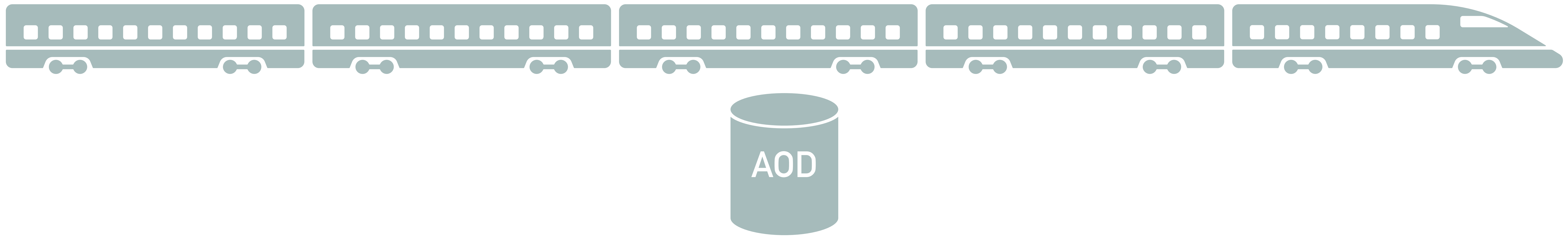
O2 Computing Model in one Slide



Analysis Model: Run 3

Solid foundations: *the idea of organised analysis will remain. Improve on the implementation.*

- ***x100*** *more collisions compared to present setup*
- *Do initial analysis on a fraction of the data on fewer, highly performant,* ***Analysis Facilities****.*
- *Full analysis on a reduced set of wagons on the Grid ⇒* ***Prioritise processing according to physics needs****.*
- ***Streamline data model****, reducing generality and features set to improved speed.*
- *Explore different* ***compression strategies*** *(e.g. LZ4, Zstd, custom compression code)*
- ***Recompute*** *quantities on the fly rather than storing them. CPU cycles are cheap.*
- *Goal is to have each Analysis Facility go through 5PB of AODs every 12 hours (~100GB/s).*



Requirements for the AOD Format

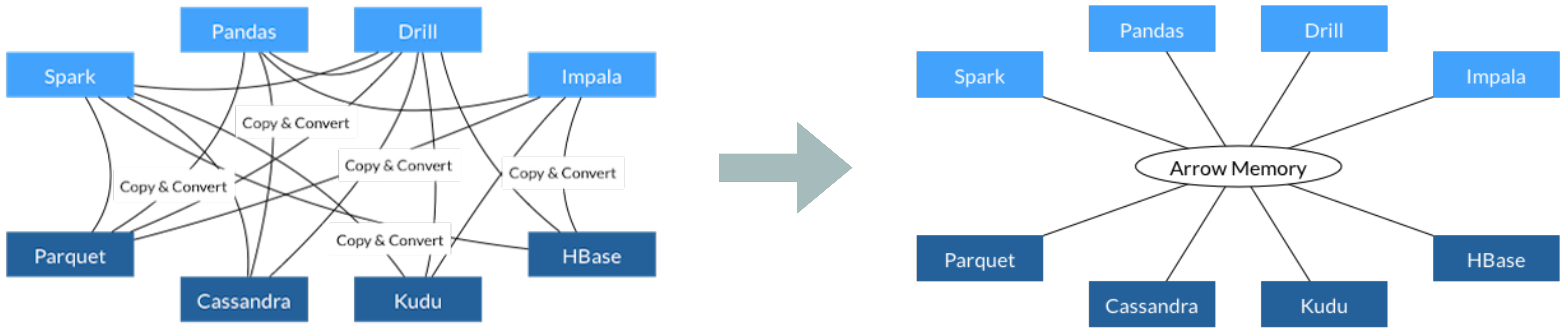
AOD's data format will have to play well with AliceO2 message passing, shared memory backed, distributed nature.

- **Zero-{Copy,Serialisation,Adjustments}**: *we want to be able to reuse data between processes.*
- **Growable**: *ability to extend columns on the fly.*
- **Prunable:** *ability to drop columns on the fly.*
- **Skimmable:** *ability to select only certain rows.*

Strategy: we are willing to lose some degree of generality for performance.

Apache Arrow: a possible solution for In-Memory Columnar Format

"Cross-language development platform for in-memory columnar data."



Well established. *Top-Level Apache project backed by key developers of a number of opensource projects:* ***Calcite****, Cassandra, Drill, Hadoop, HBase, Ibis, Impala, Kudu,* ***Pandas****,* ***Parquet****, Phoenix,* ***Spark****, and Storm.*

Very active. *119 contributors, https://github.com/apache/arrow*

O2 design friendly. *message passing / shared memory friendly. Support for zero-copy slicing, filtering.*

Apache Arrow: a few technical details

In memory column oriented storage. *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.*

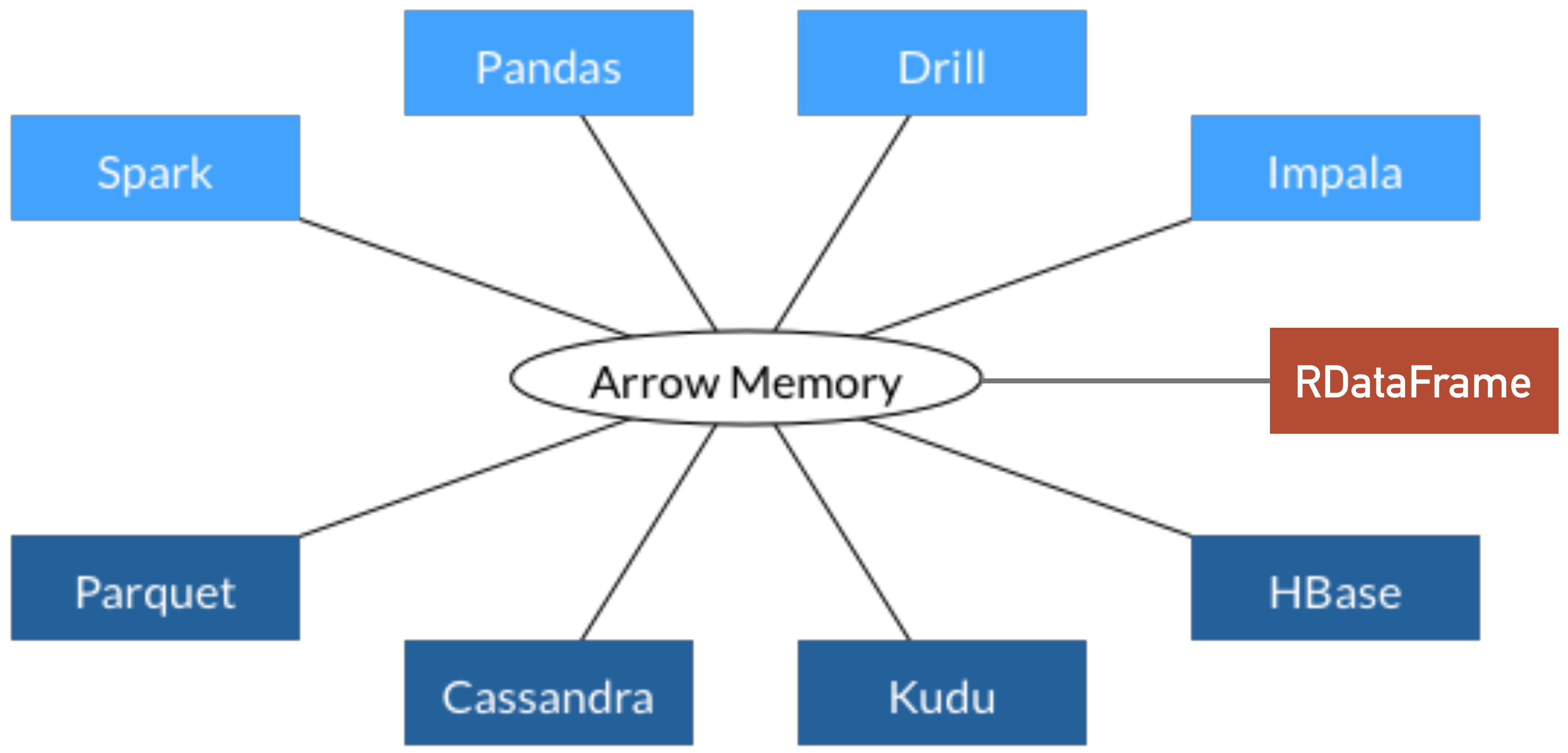
Nested types. *Usual basic types (int, float, ..). It's also possible (via the usual record shredding presented in Google's Dremel paper) to support nested types. E.g. a String is a List<Char>.*

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

Suitable for ALICE analysis needs?

Apache Arrow: Integration with ROOT

The main concern here is of course "how do I use this from ROOT"?



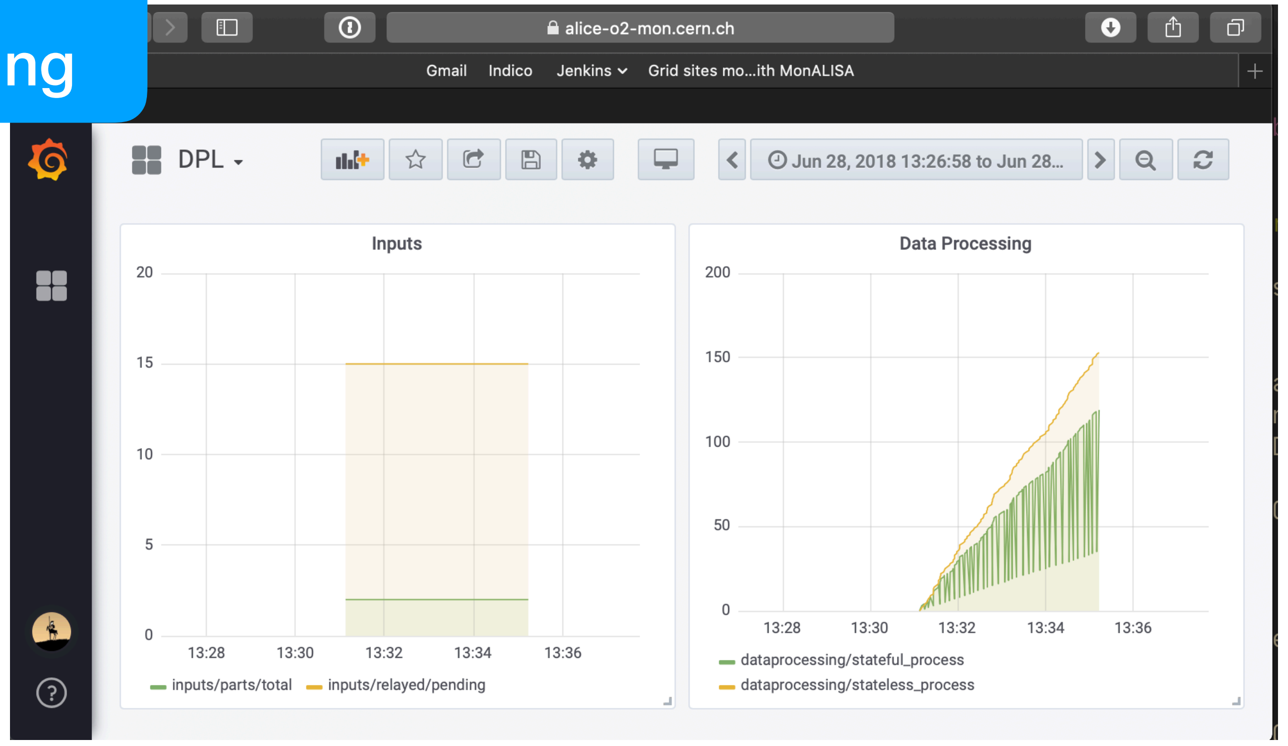
RDataFrame: *new component of ROOT for "declarative analysis". Modular design allows*

Initial integration of Arrow with ROOT has already been provided by ALICE and merged by the ROOT team.

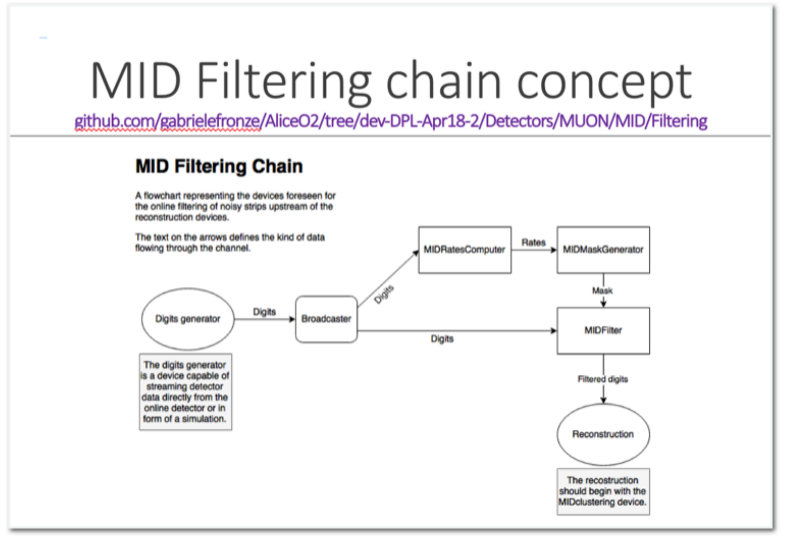
Bonus: *ROOT gets seamless integration with many OpenSource projects which you can mention to impress your friends and that make your CV look good to head-hunters.*

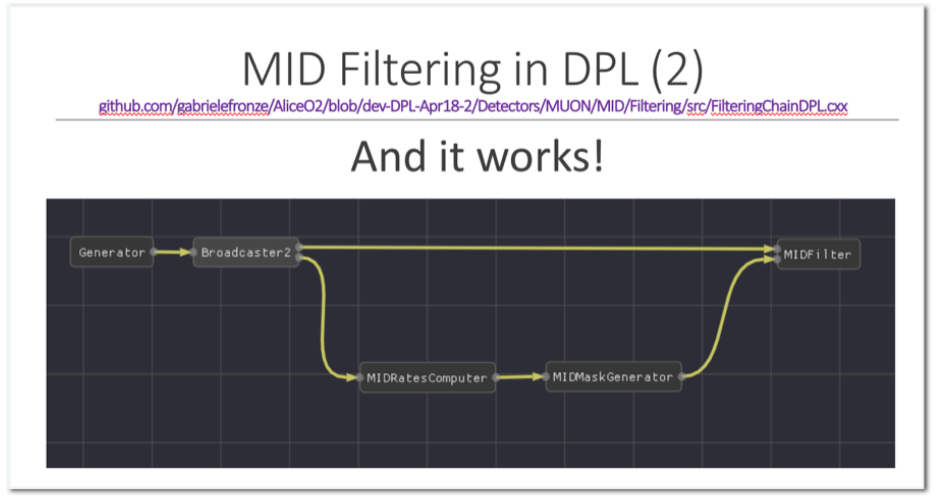
DPL AS AN INTEGRATION PLATFORM FOR O2

WP8 / Monitoring

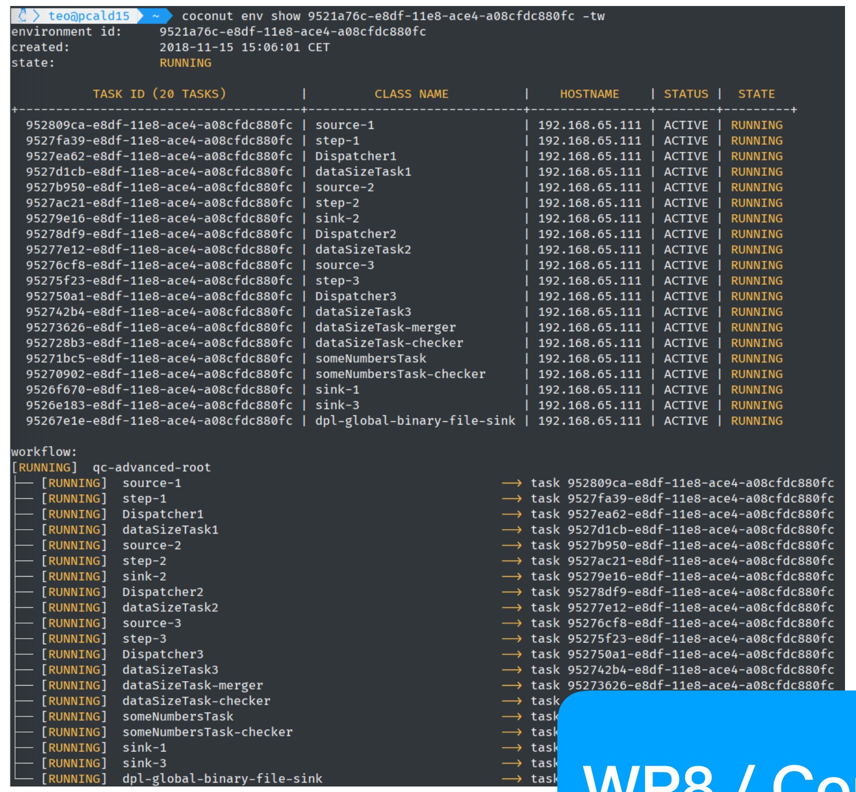


O2 Monitoring and InfoLogger integration



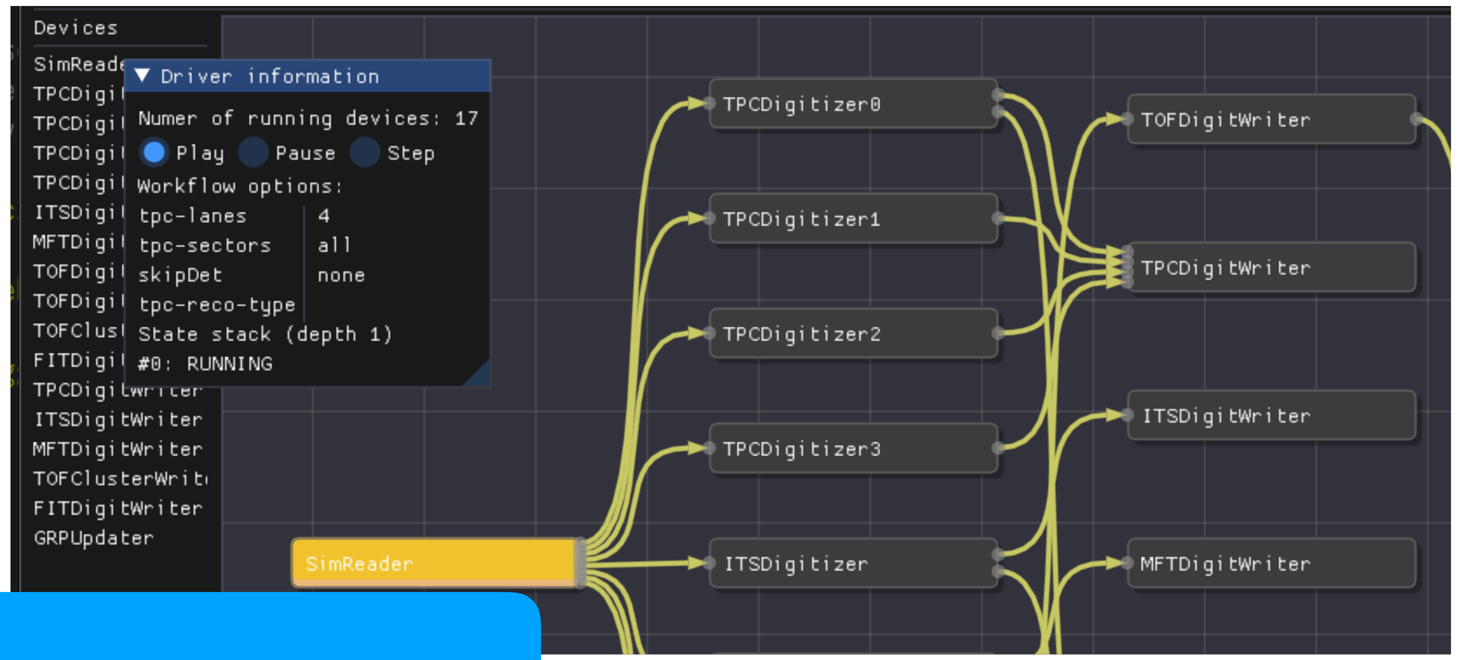


MID Filtering Chain



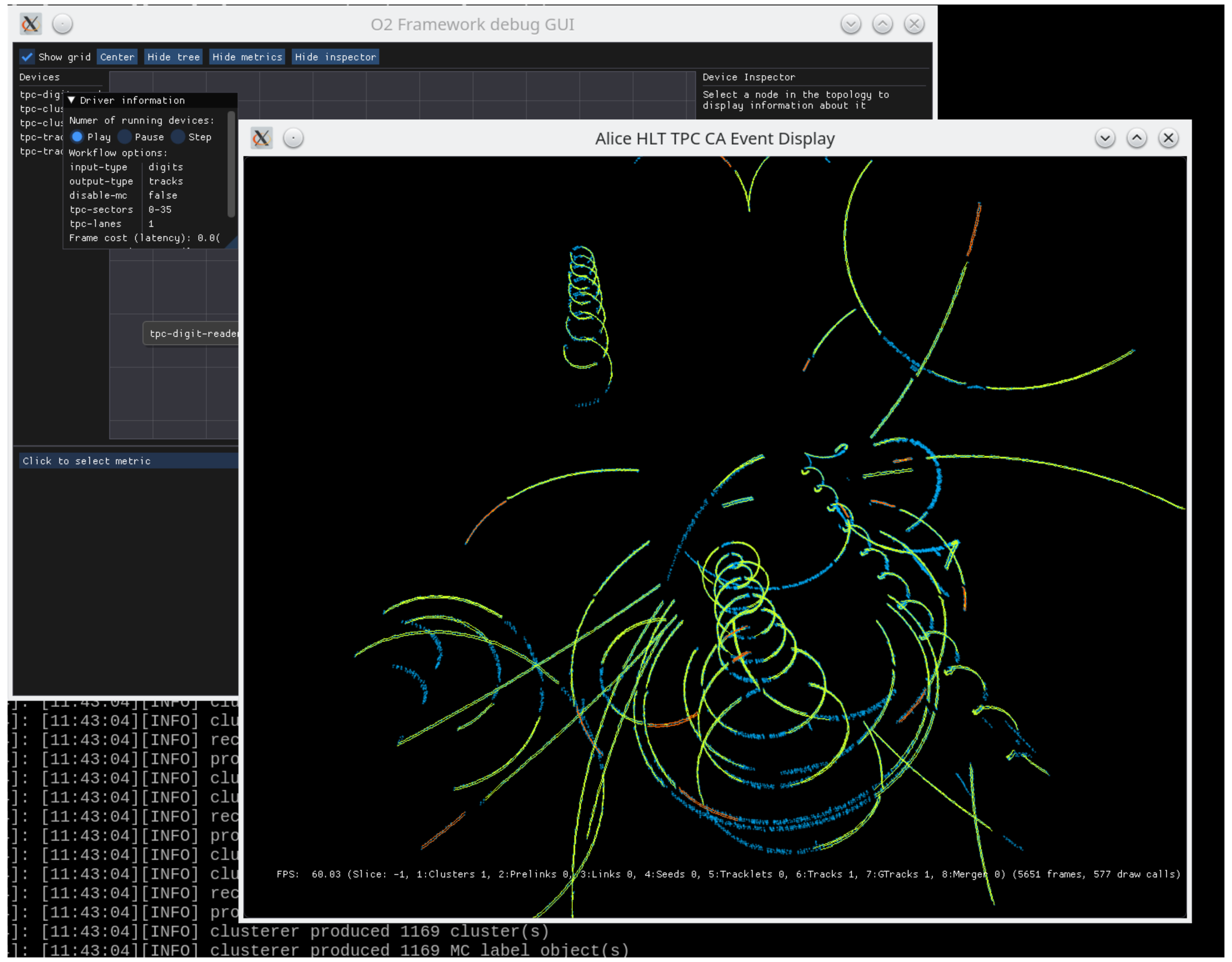
O2 AliECS

WP8 / Control

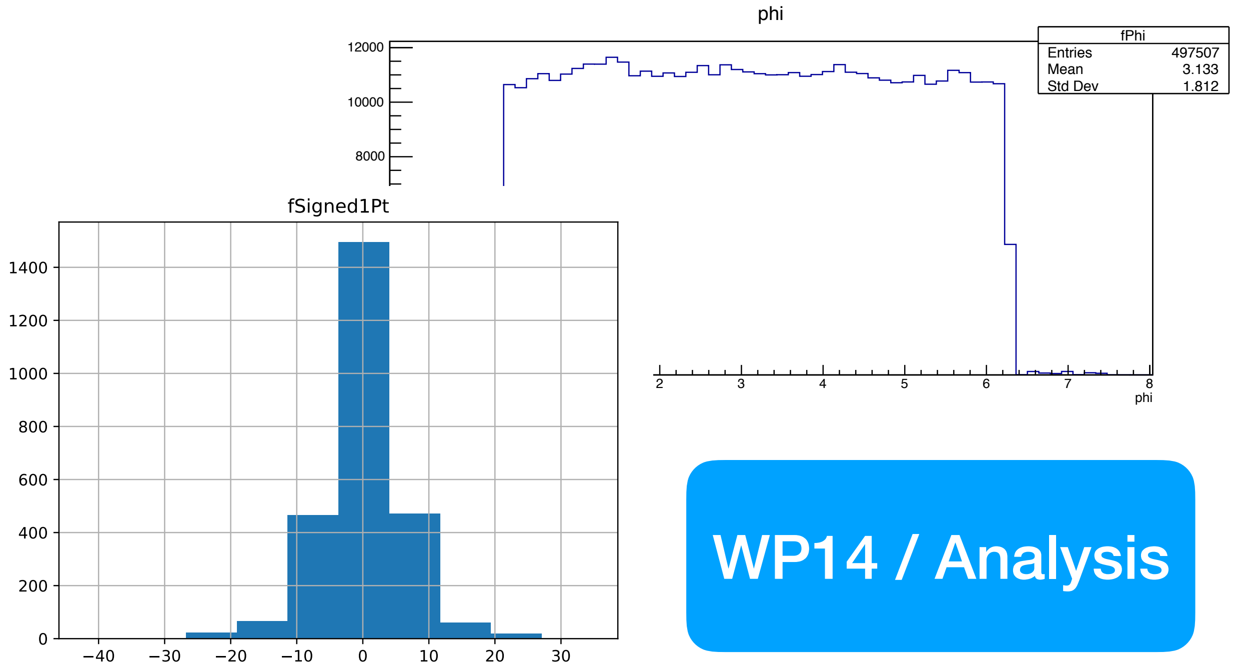


WP12 / Simulation

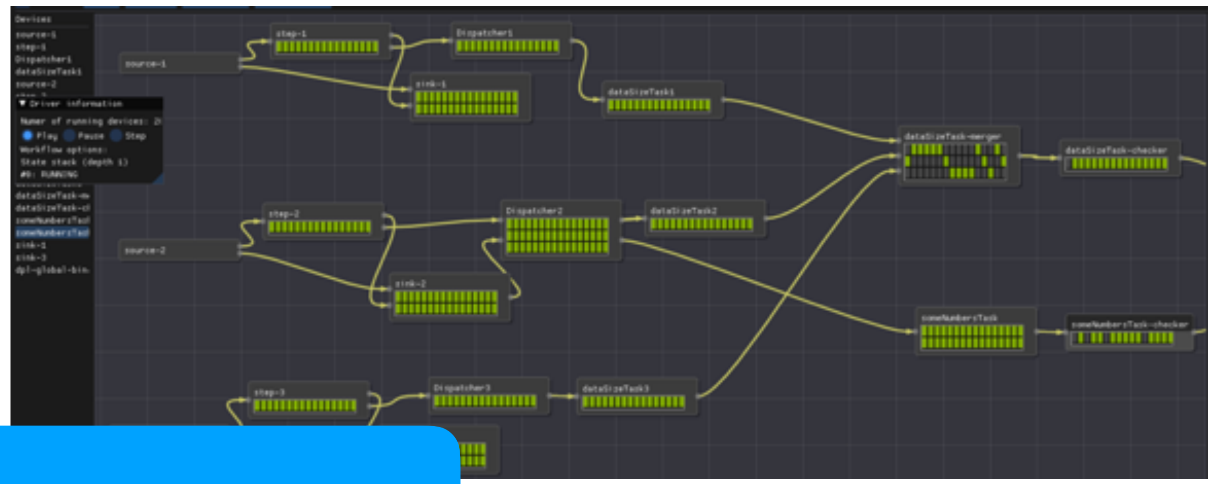
Digitization



TPC Event Displays

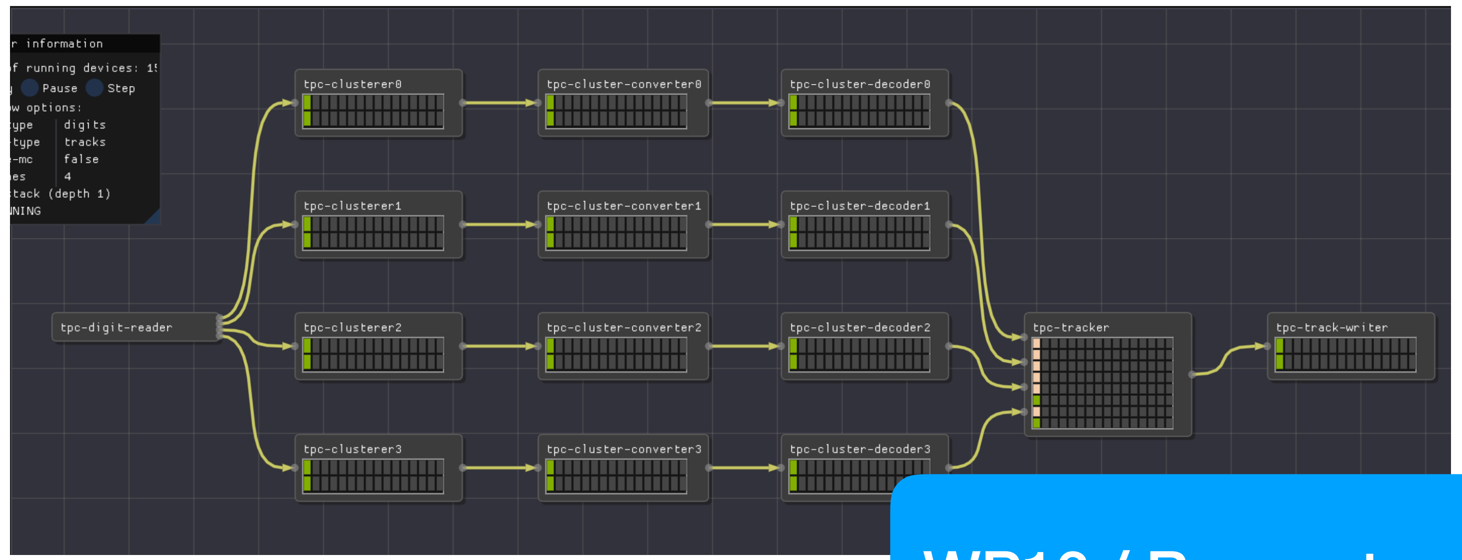


WP14 / Analysis



WP7 / Quality Control

DataSampling

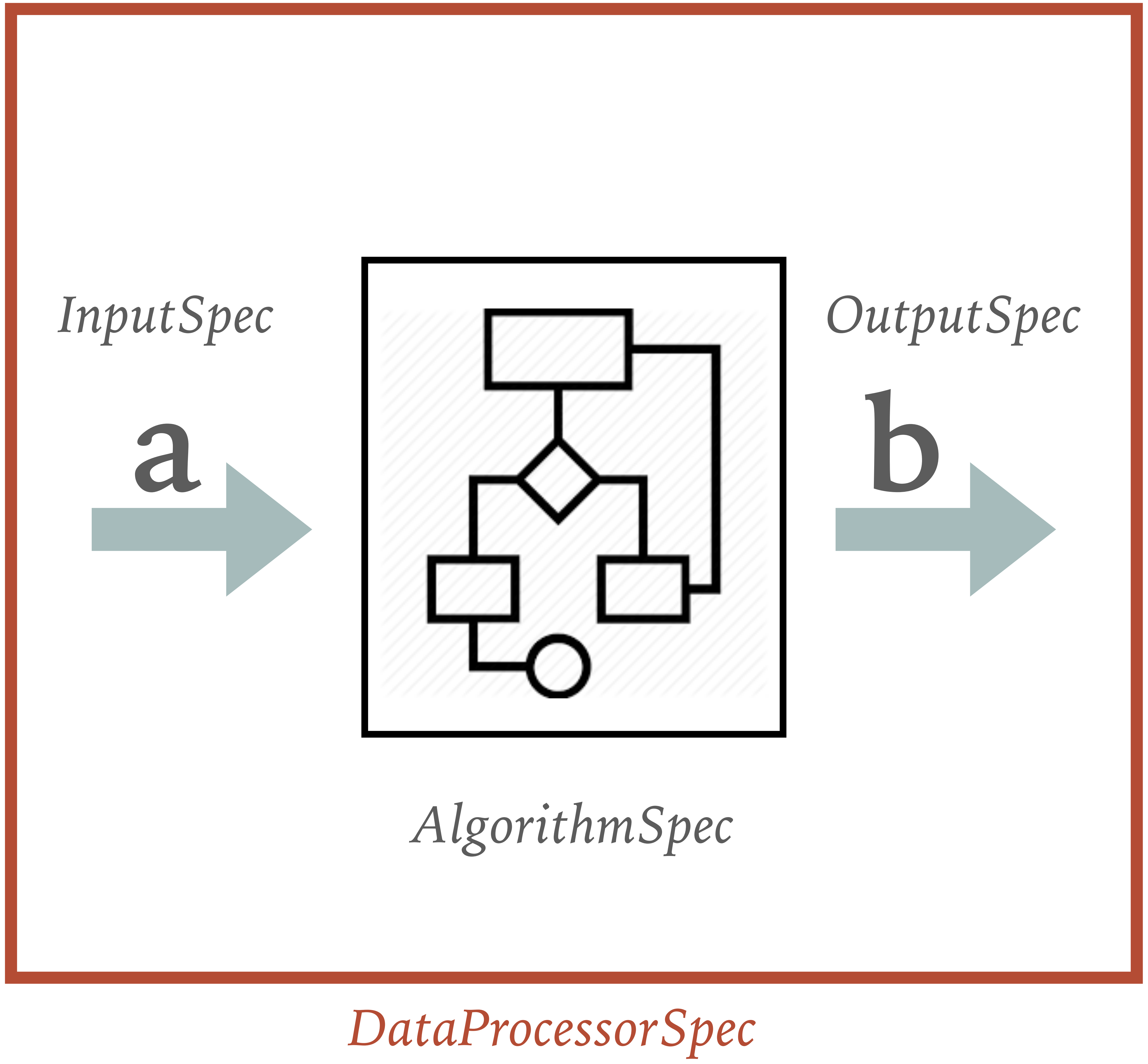


TPC reconstruction

WP13 / Reconstruction

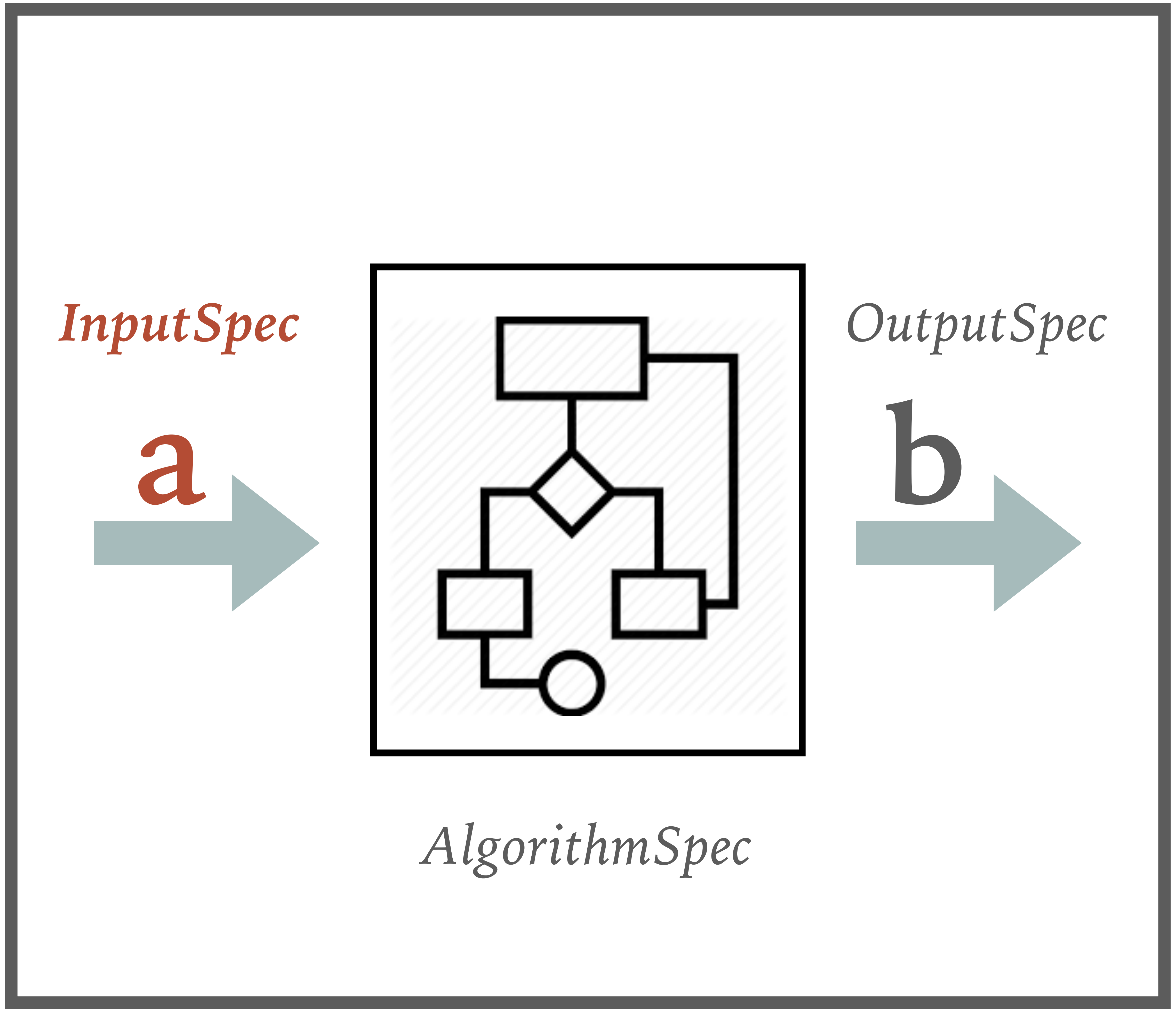
BACKUP

DATA PROCESSING LAYER: HOW




```
DataProcessorSpec{
  "A",
  Inputs{
   InputSpec{"a", "TPC", "CLUSTERS"}
  },
  Outputs{
   OutputSpec{{"b"}, "TPC", "TRACKS"}
  },
  AlgorithmSpec{
    [](ProcessingContext &ctx) {
       auto track = ctx.outputs().make<Track>(OutputRef{ "b" }, 1);
    }
  }
}
```

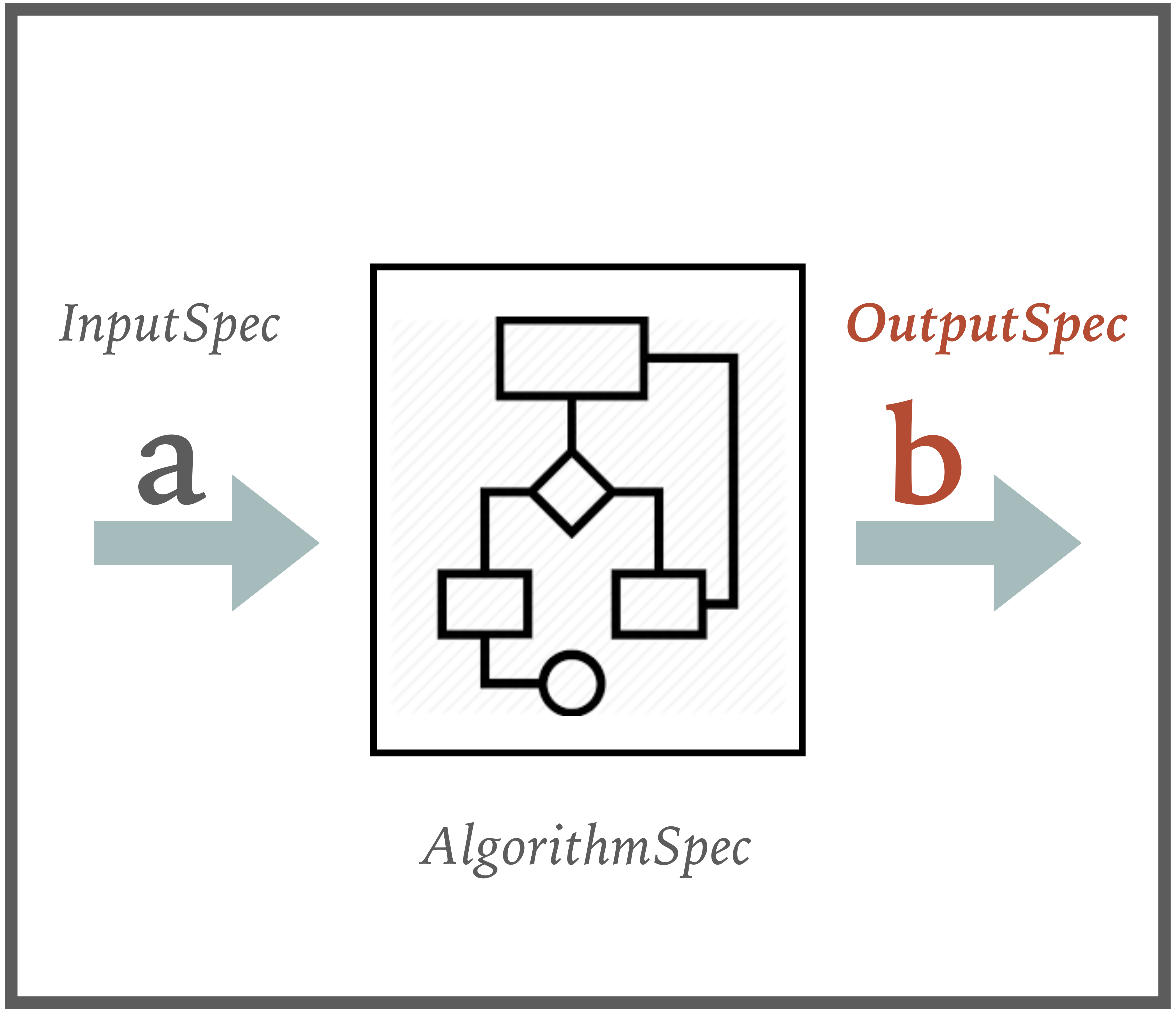
DATA PROCESSING LAYER: HOW



DataProcessorSpec

```
DataProcessorSpec{
  "A",
  Inputs{
   InputSpec{"a", "TPC", "CLUSTERS"}
  },
  Outputs{
   OutputSpec{{"b"}, "TPC", "TRACKS"}
  },
  AlgorithmSpec{
    [](ProcessingContext &ctx) {
       auto track = ctx.outputs().make<Track>(OutputRef{ "b" }, 1);
    }
  }
}
```

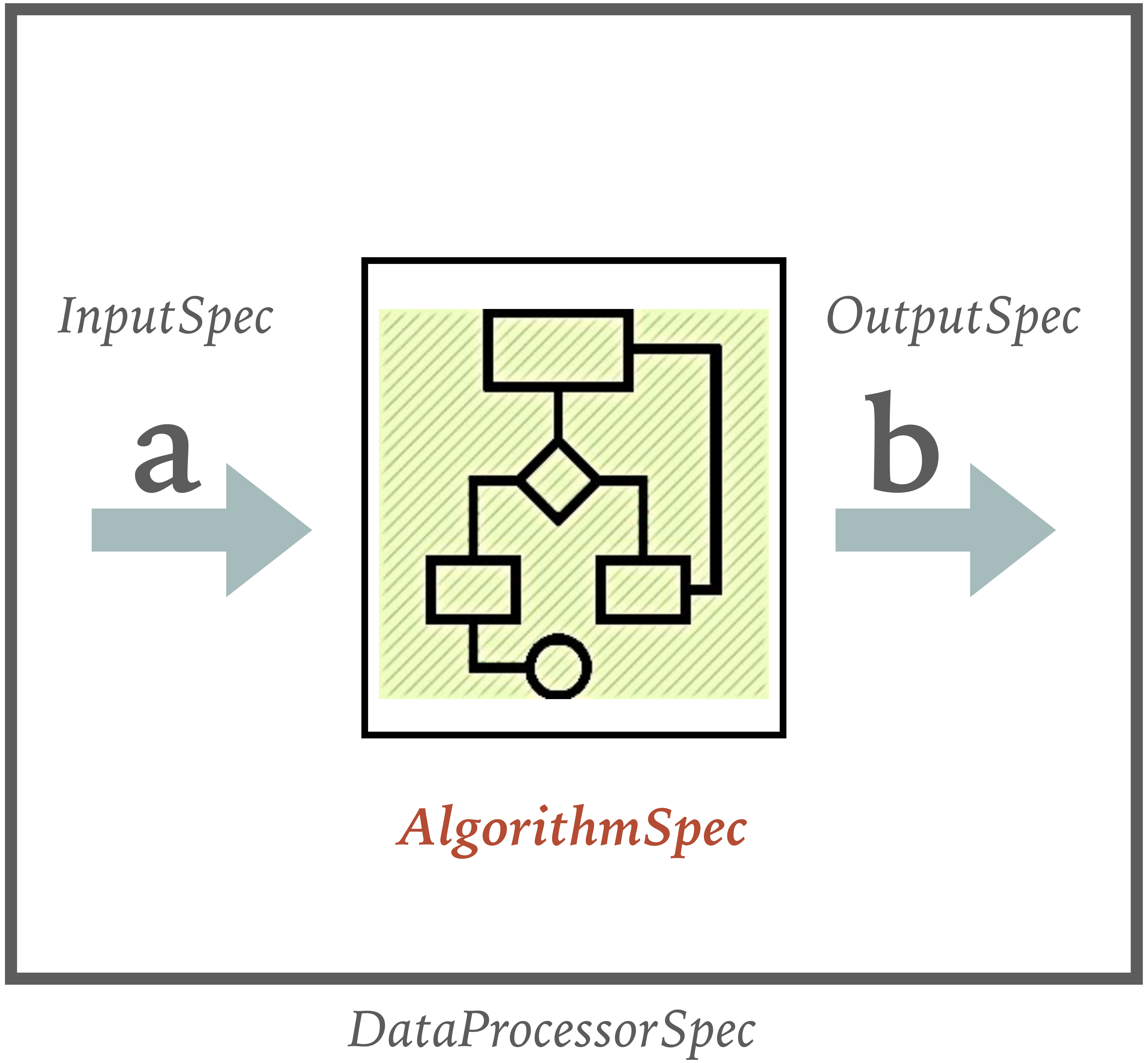
DATA PROCESSING LAYER: HOW



DataProcessorSpec

```
DataProcessorSpec{
  "A",
  Inputs{
   InputSpec{"a", "TPC", "CLUSTERS"}
  },
  Outputs{
   OutputSpec{{"b"}, "TPC", "TRACKS"}
  },
  AlgorithmSpec{
    [](ProcessingContext &ctx) {
       auto track = ctx.outputs().make<Track>(OutputRef{ "b" }, 1);
    }
  }
}
```

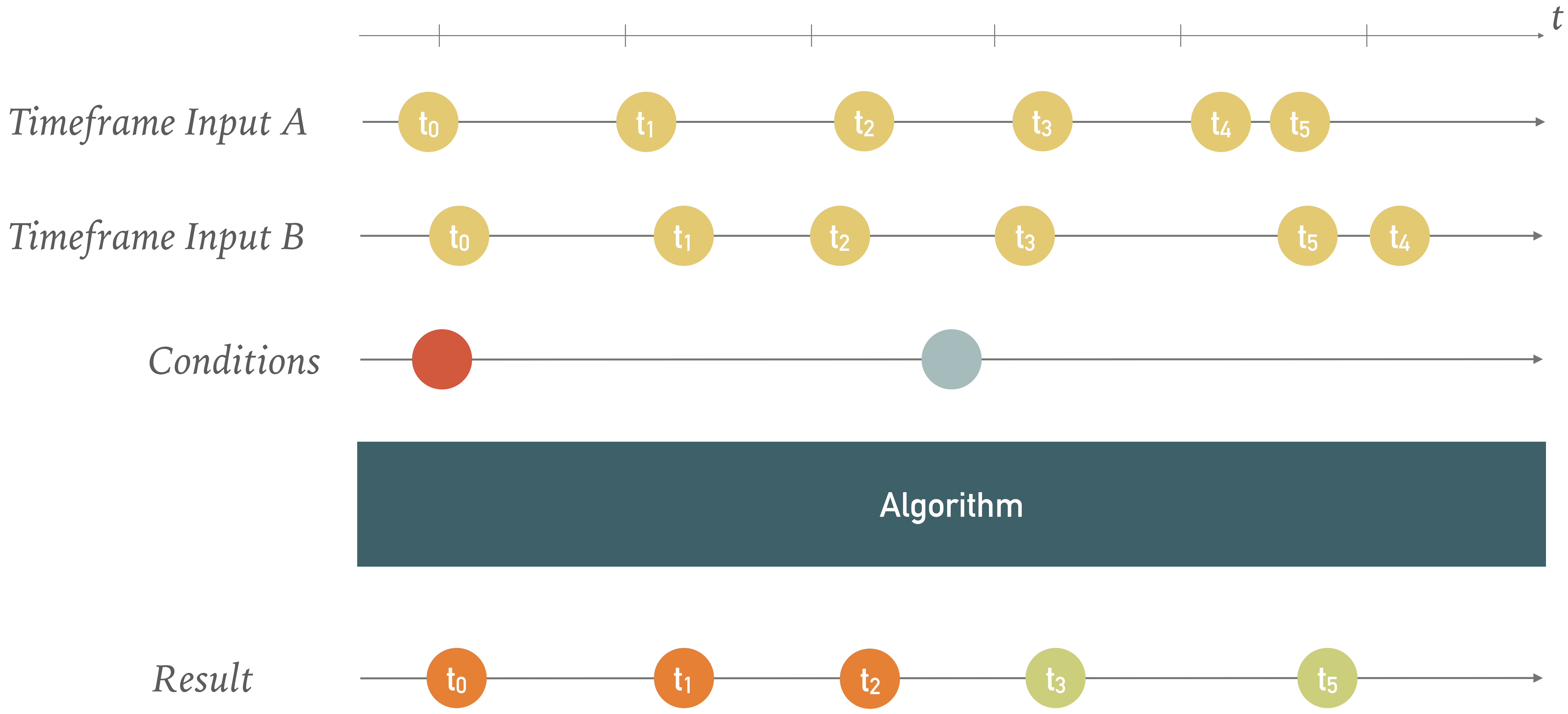
DATA PROCESSING LAYER: HOW



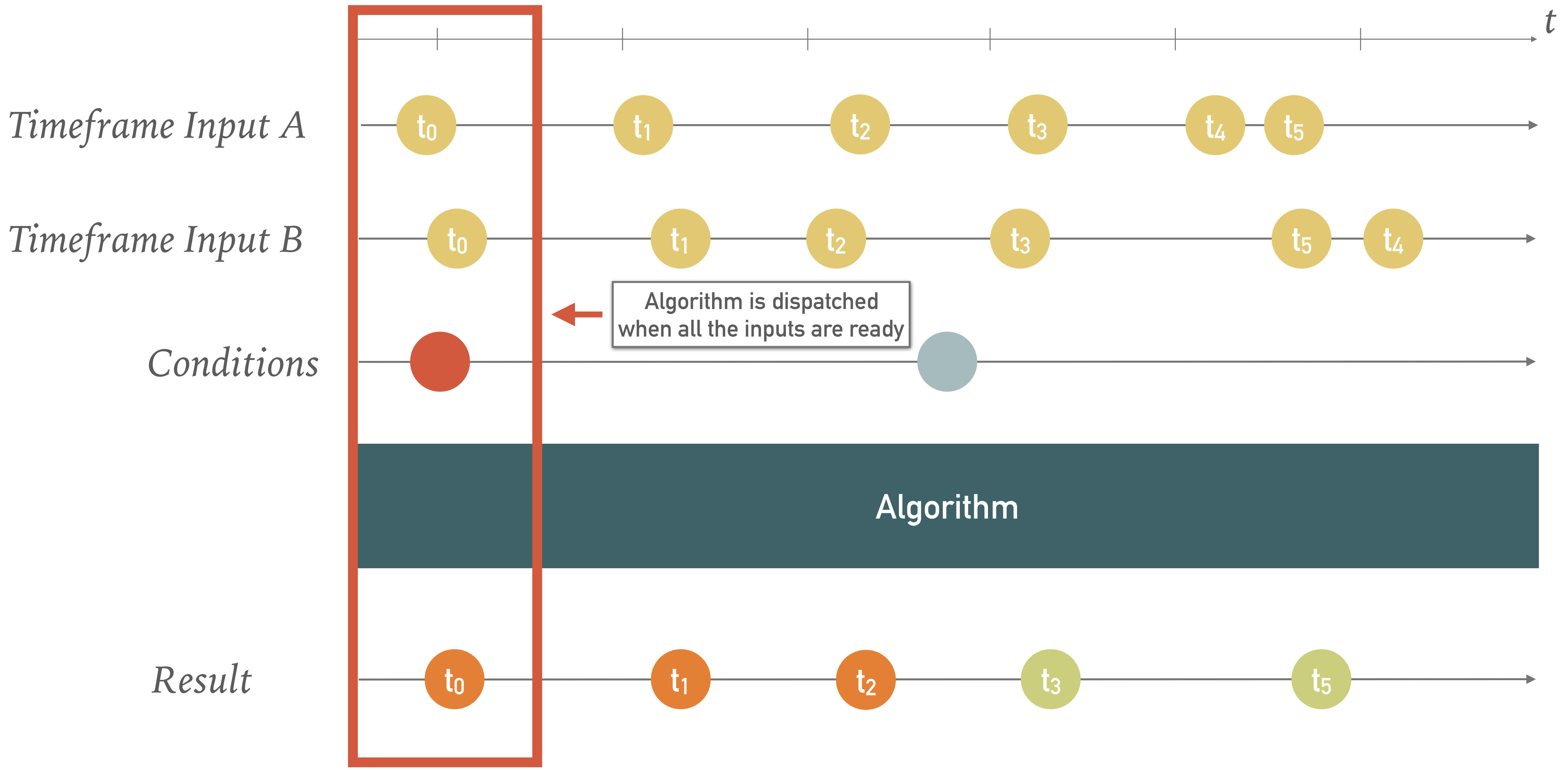

```
DataProcessorSpec{
  "A",
  Inputs{
   InputSpec{"a", "TPC", "CLUSTERS"}
  },
  Outputs{
   OutputSpec{{"b"}, "TPC", "TRACKS"}
  },
  AlgorithmSpec{
    [](ProcessingContext &ctx) {
       auto track = ctx.outputs().make<Track>(OutputRef{ "b" }, 1);
    }
  }
}
```

REACTIVE DESIGN

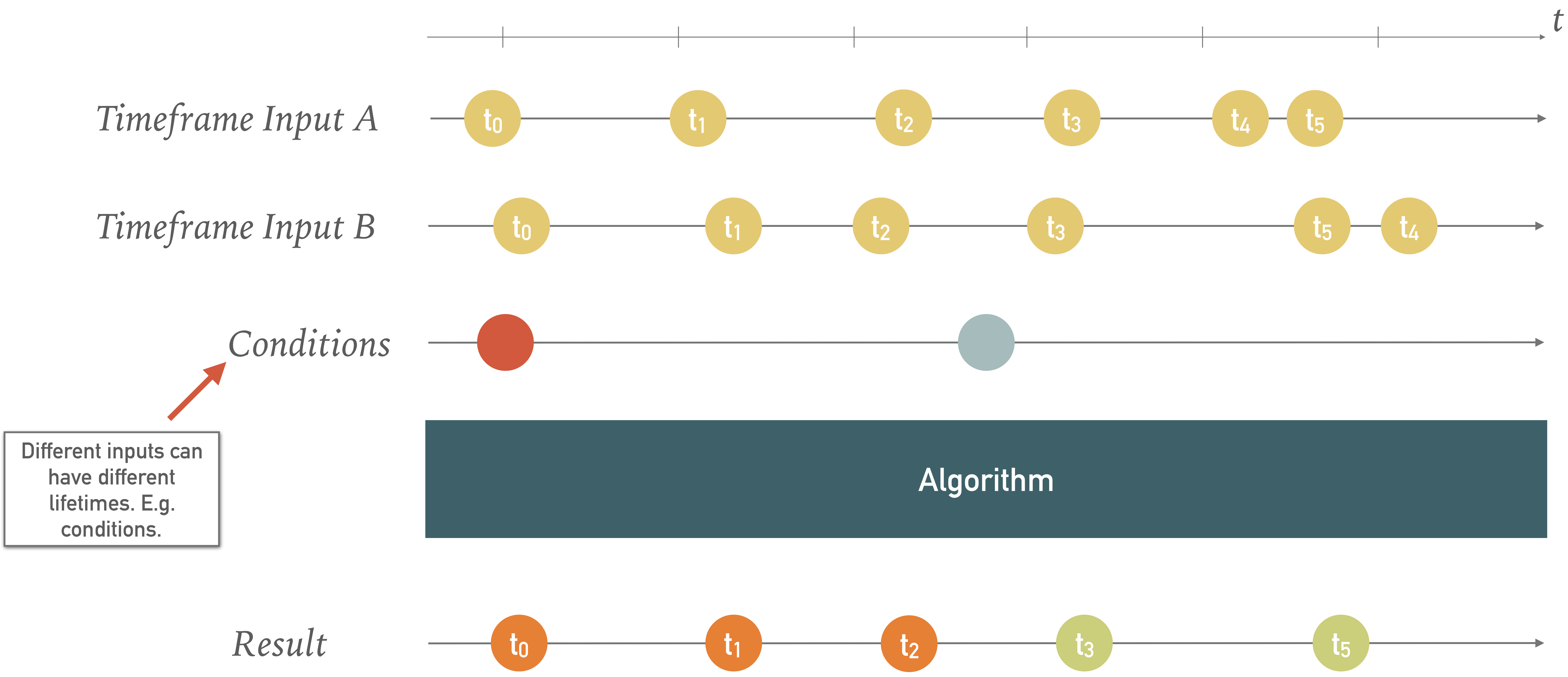
Data is described as pushed through the pipeline.



REACTIVE DESIGN



REACTIVE DESIGN



REACTIVE DESIGN