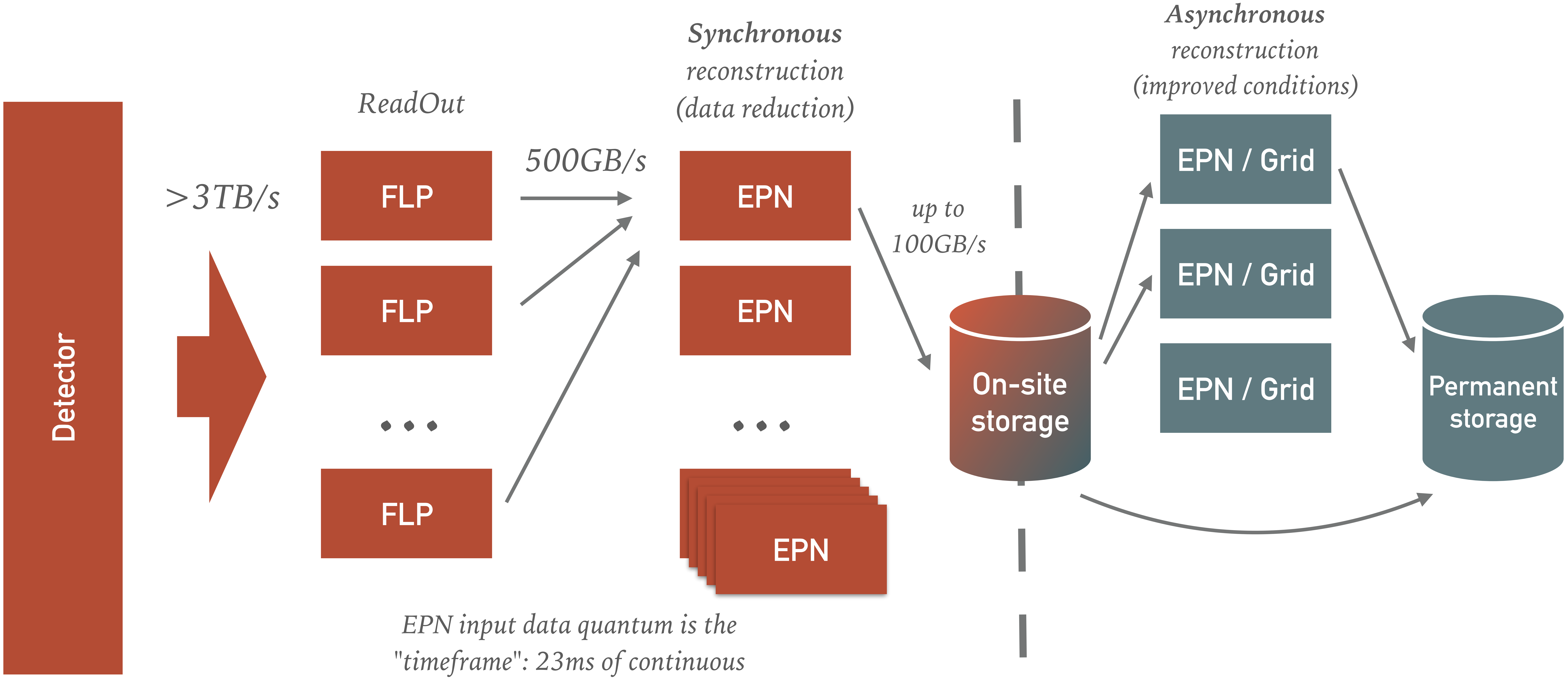
Evolution of the ALICE Software Framework for LHC Run 3

Giulio Eulisse (CERN), for the ALICE Collaboration

ALICE IN RUN 3: POINT2



EPN input data quantum is the "timeframe": 23ms of continuous readout data. ~10GB

BEAM ON: Data reduction

BEAM OFF: improved calibration

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.*

[1] *https://indico.cern.ch/event/587955/contributions/2938082/*

ALICE O2 SOFTWARE FRAMEWORK IN ONE SLIDE

Data Layer: O2 Data Model

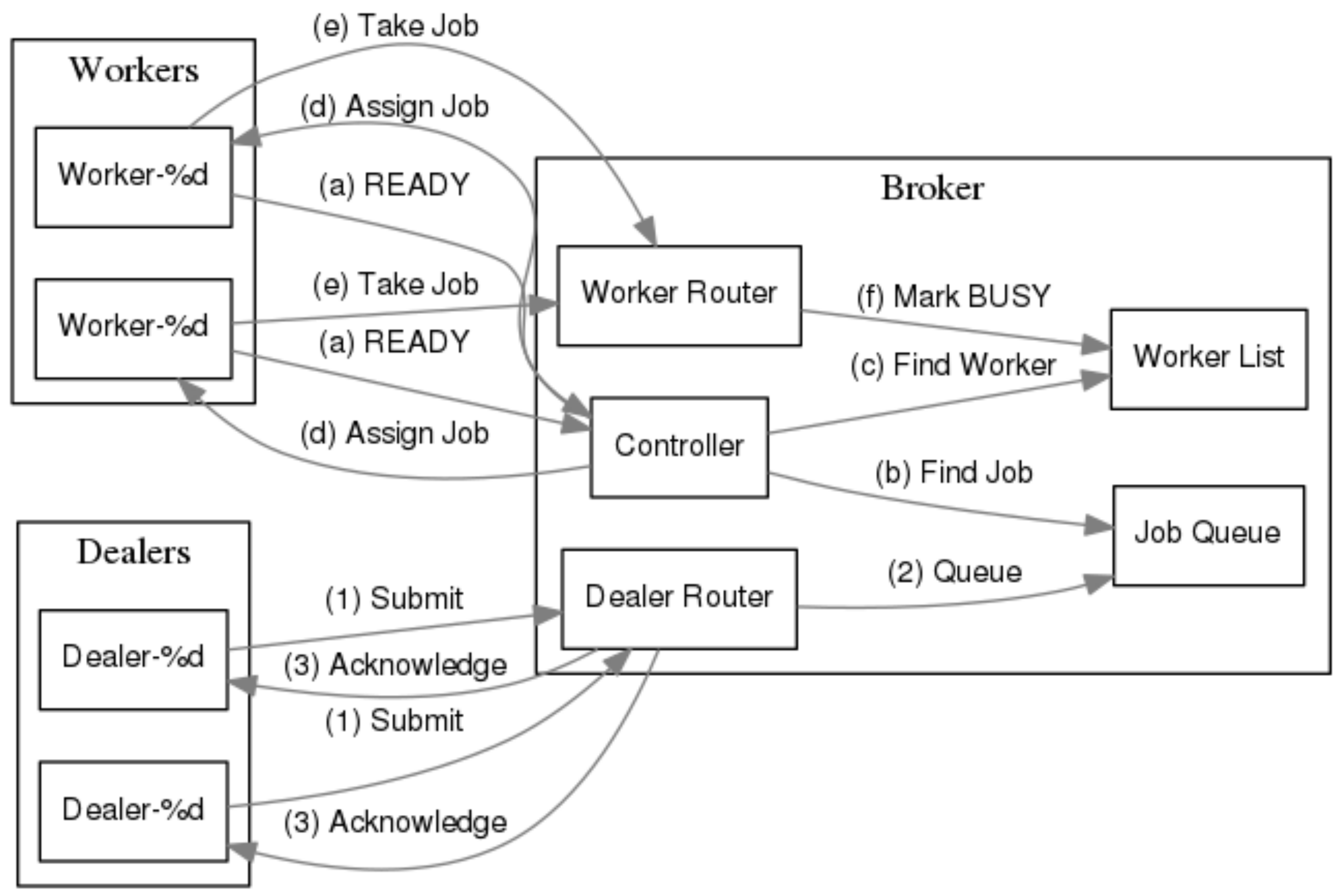
Message passing aware data model. Support for multiple backends:

- **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 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.*

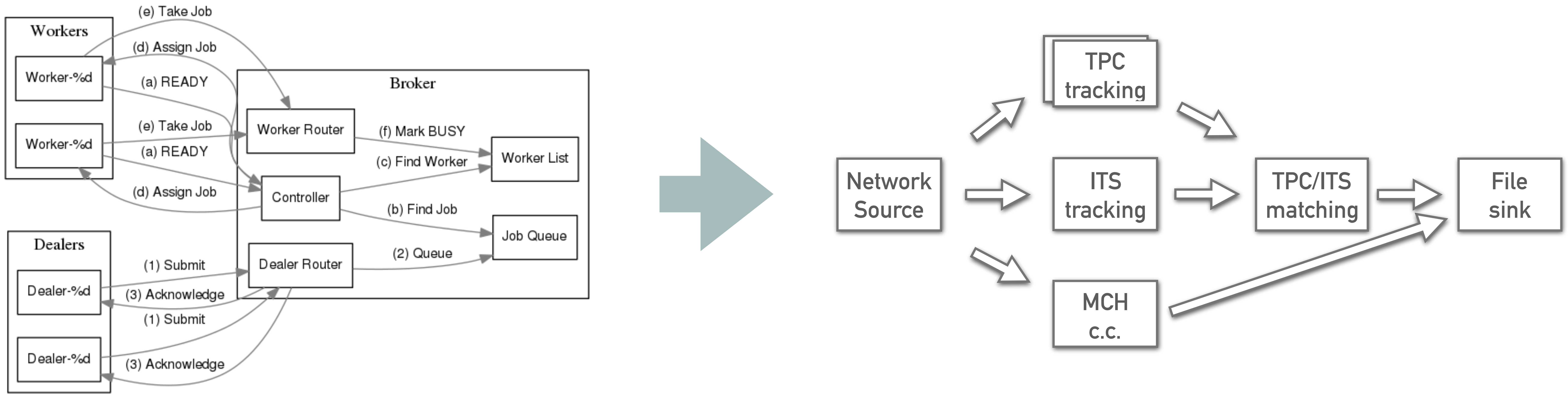
DISTRIBUTED SYSTEMS ARE HARD



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

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 DSL*** *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.*

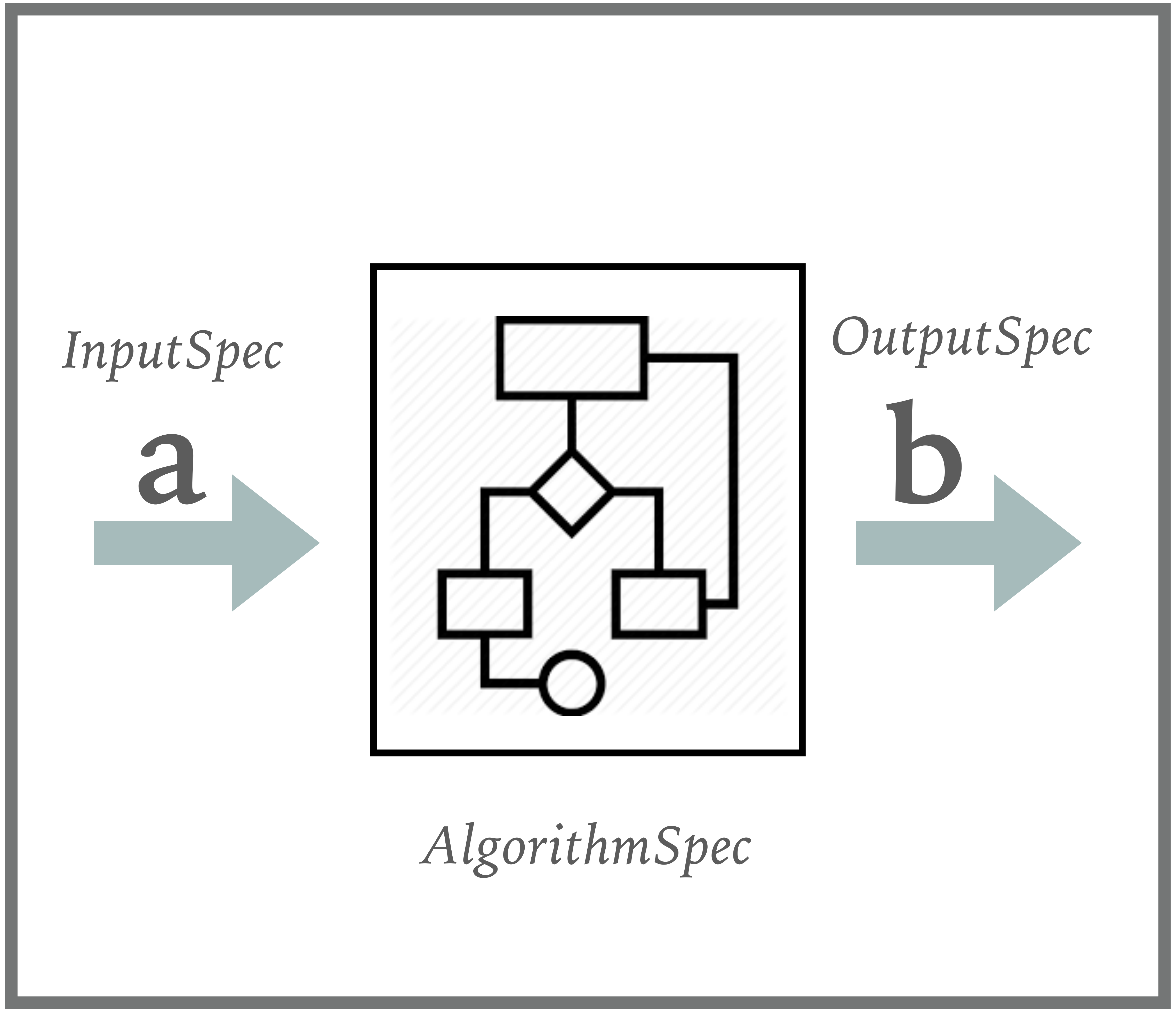
Transport Layer: FairMQ

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

DATA PROCESSING LAYER: BUILDING BLOCK

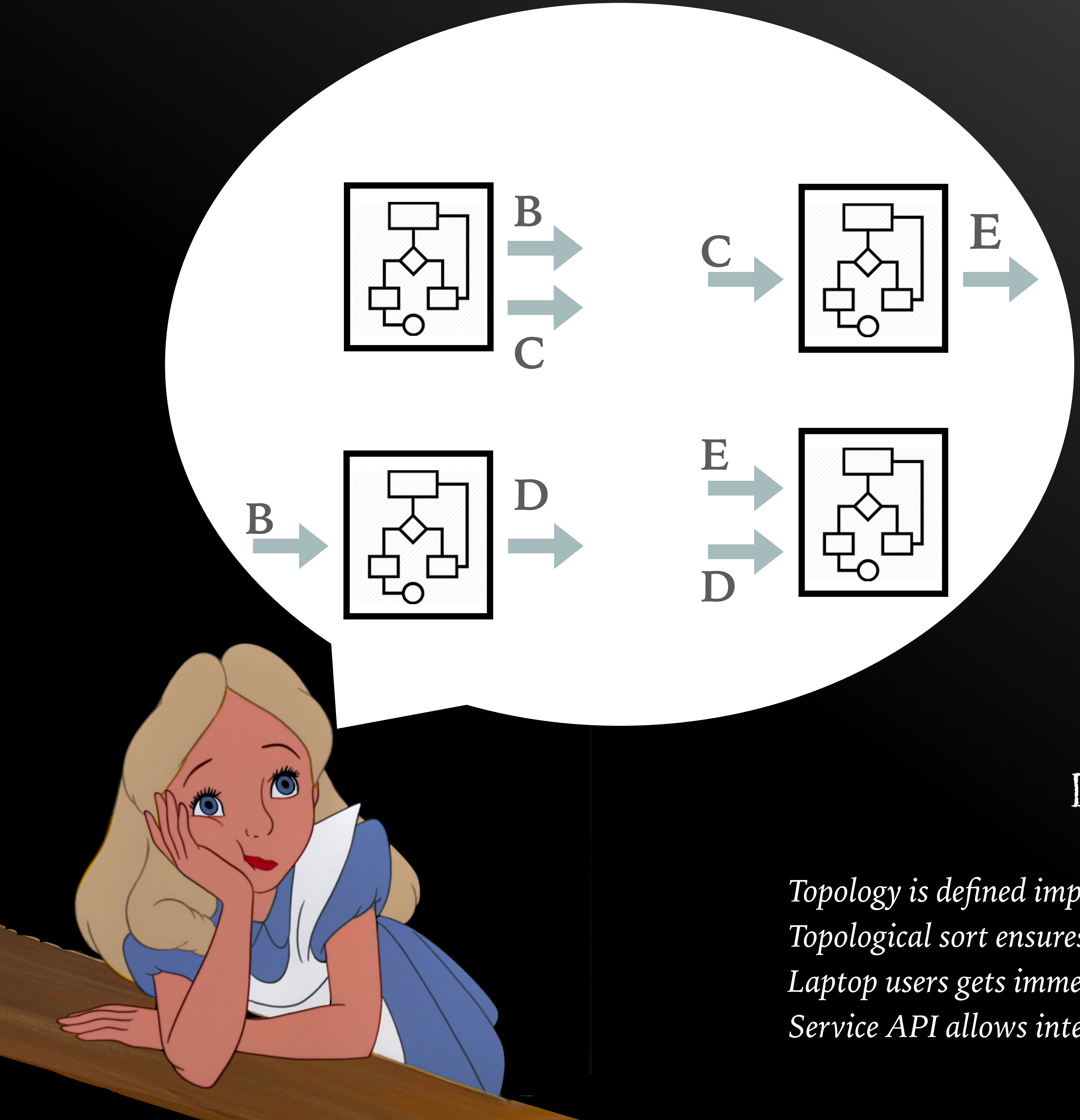
A **DataProcessorSpec** *defines a pipeline stage as a building block.*

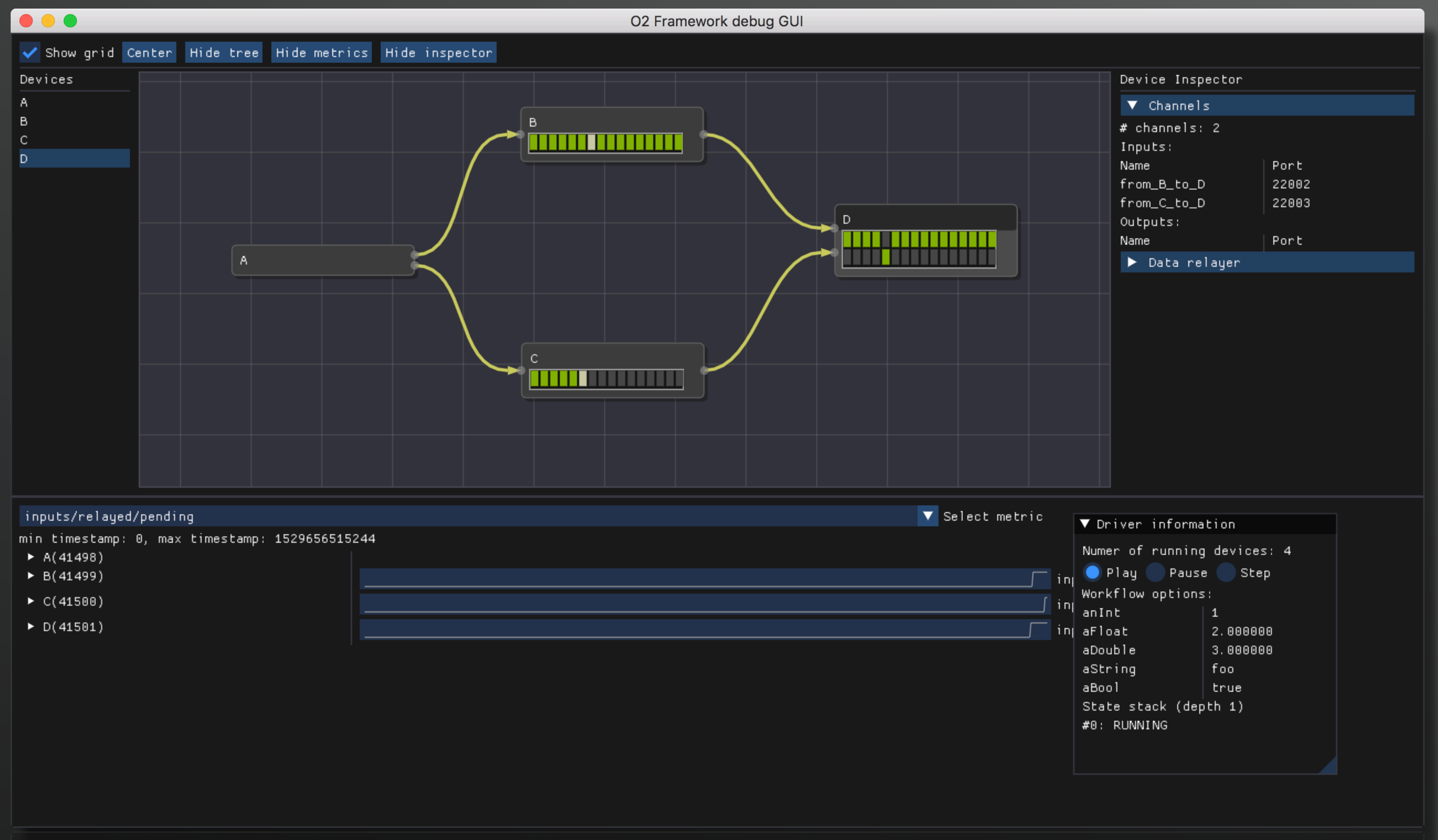
- *Specifies* ***inputs and outputs*** *in terms of the O2 Data Model descriptors.*
- *Provide an implementation of how to act on the inputs to produce the output.*
- *Advanced user can express possible data or time parallelism opportunities.*



DataProcessorSpec

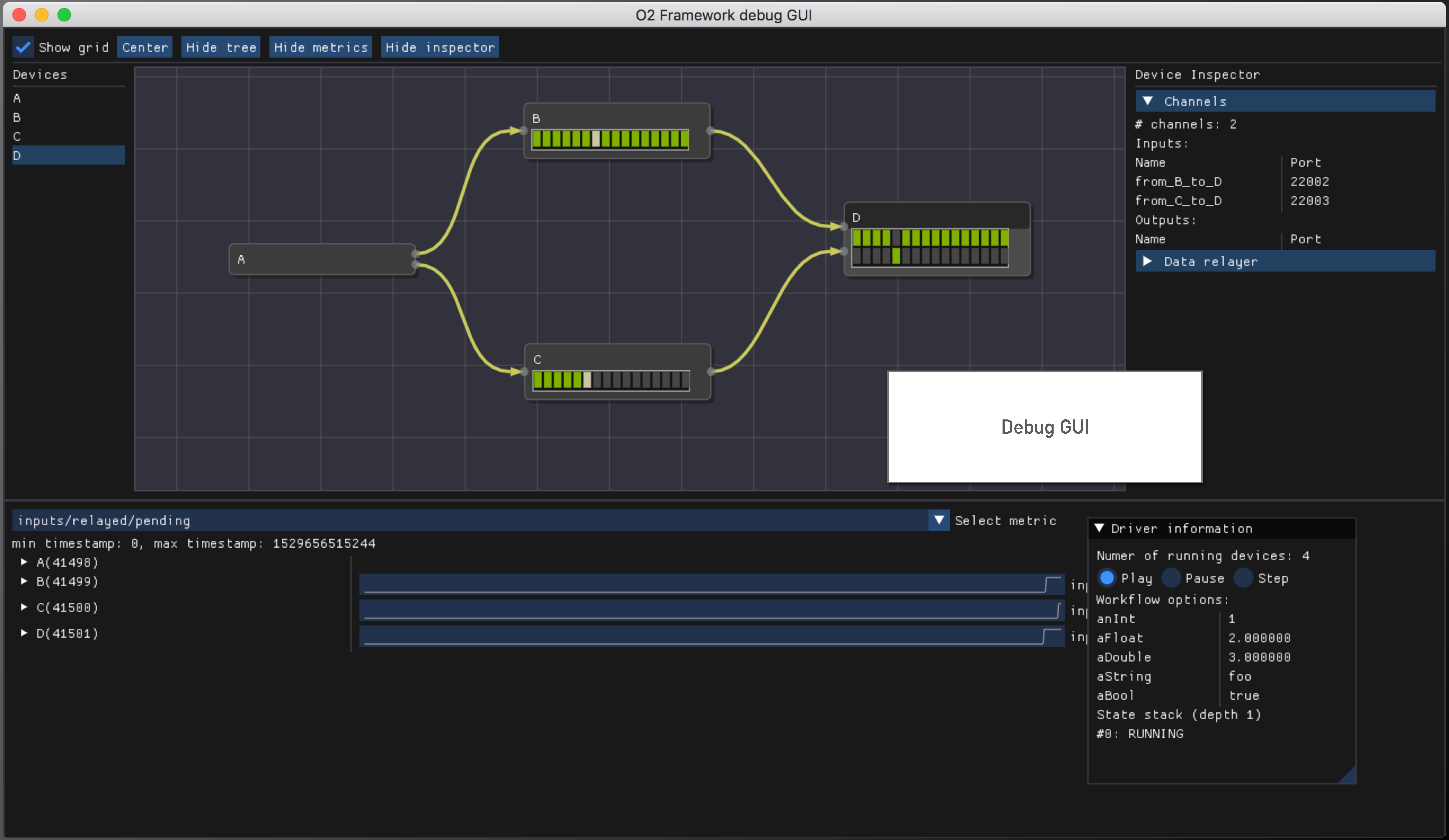
DATA PROCESSING LAYER: IMPLICIT TOPOLOGY

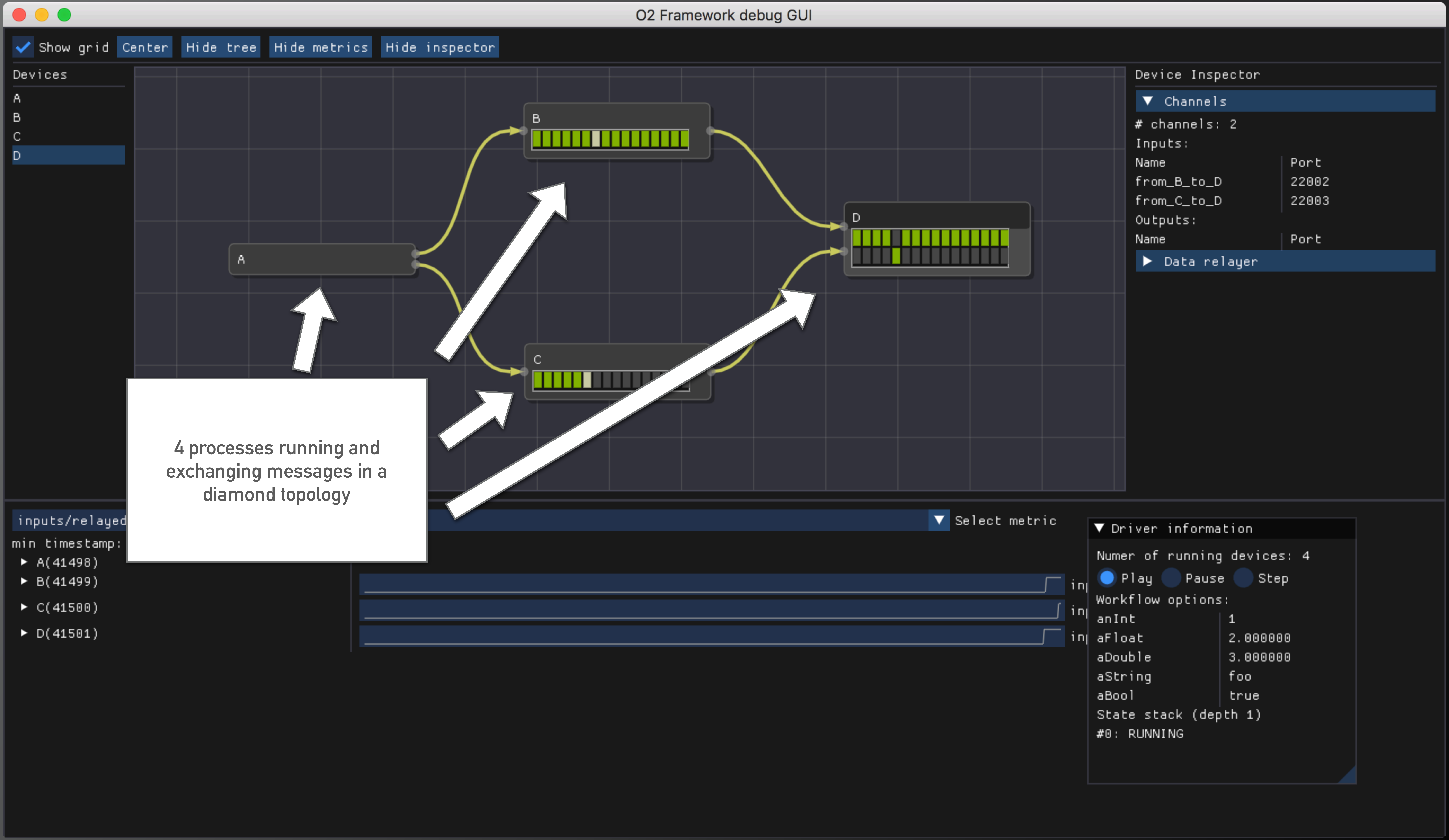




Data Processing Layer

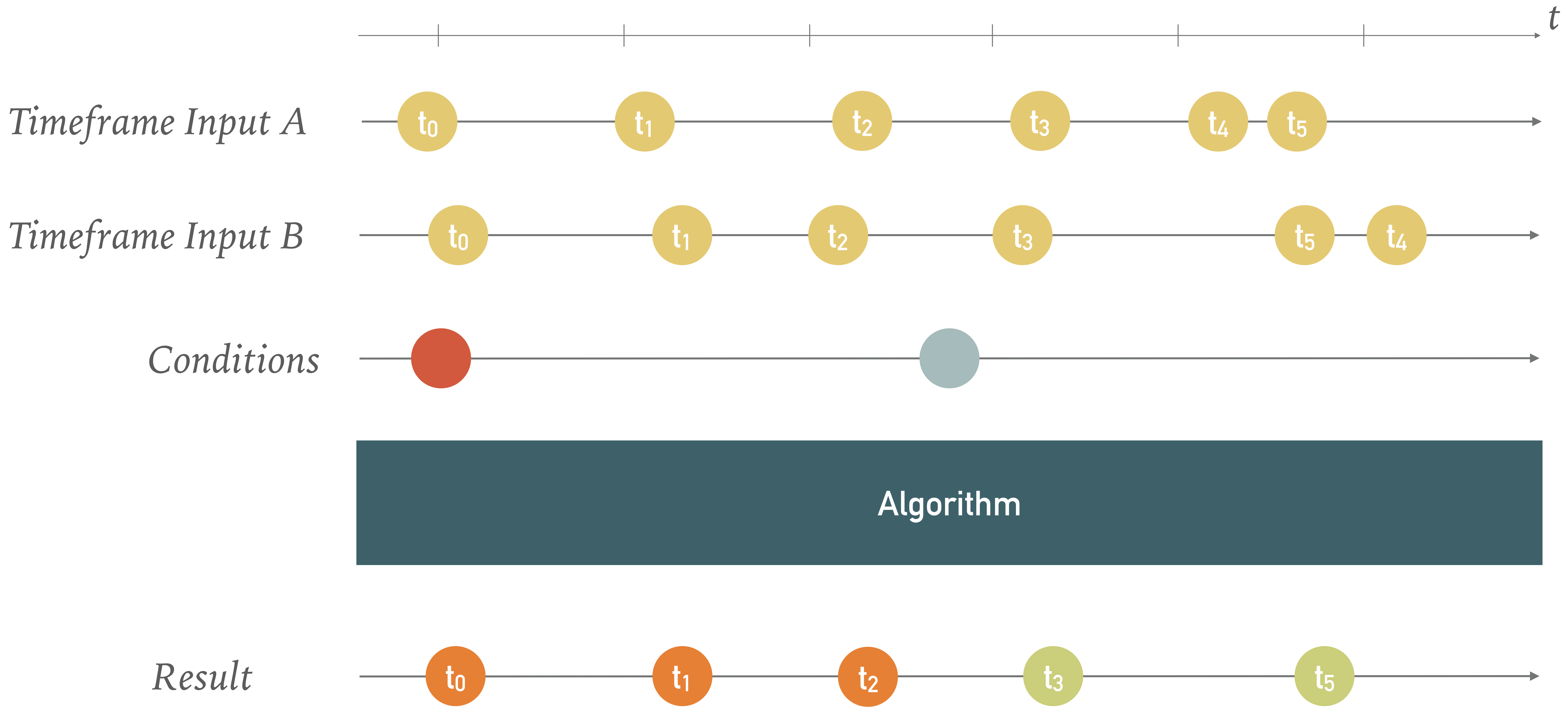
Topology is defined implicitly.
Topological sort ensures a viable dataflow is constructed (no cycles!).
Laptop users gets immediate feedback through the debug GUI.
Service API allows integration with non data flow components (e.g. Control)


O2 Framework debug GUI
Show grid
Center
Hide tree
Hide metrics
Hide inspector
Devices
A
B
C
D
Device Inspector
Channels
channels: 2
Inputs:
Name
Port
from_B_to_D
22002
from_C_to_D
22003
Outputs:
Name
Port
Data relayer
Debug GUI
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

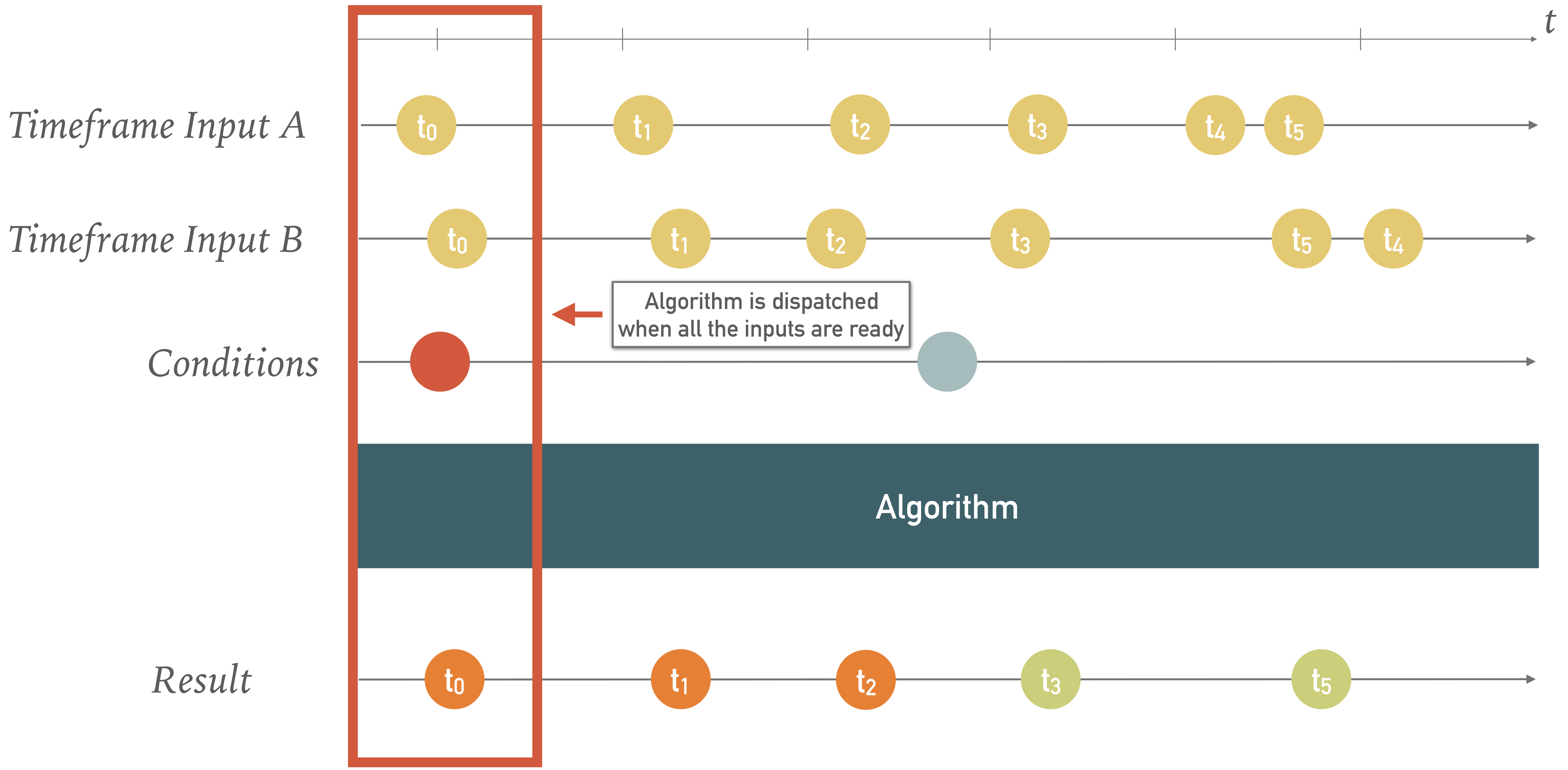

O2 Framework debug GUI
Show grid
Center
Hide tree
Hide metrics
Hide inspector
Devices
A
B
C
D
B
A
D
C
4 processes running and exchanging messages in a diamond topology
Device Inspector
Channels
channels: 2
Inputs:
Name
Port
from_B_to_D
22002
from_C_to_D
22003
Outputs:
Name
Port
Data relayer
inputs/relayed
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

REACTIVE DESIGN

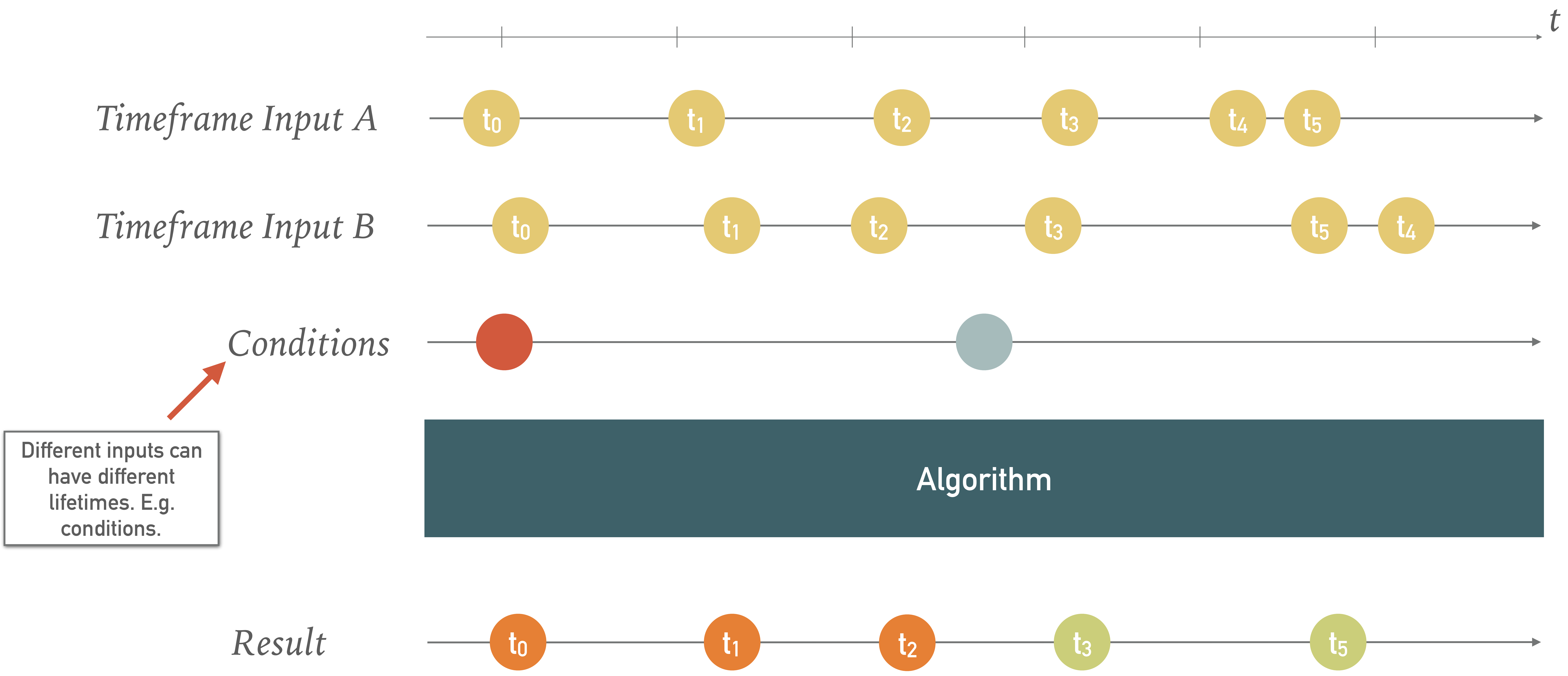
Data is described as pushed through the pipeline.



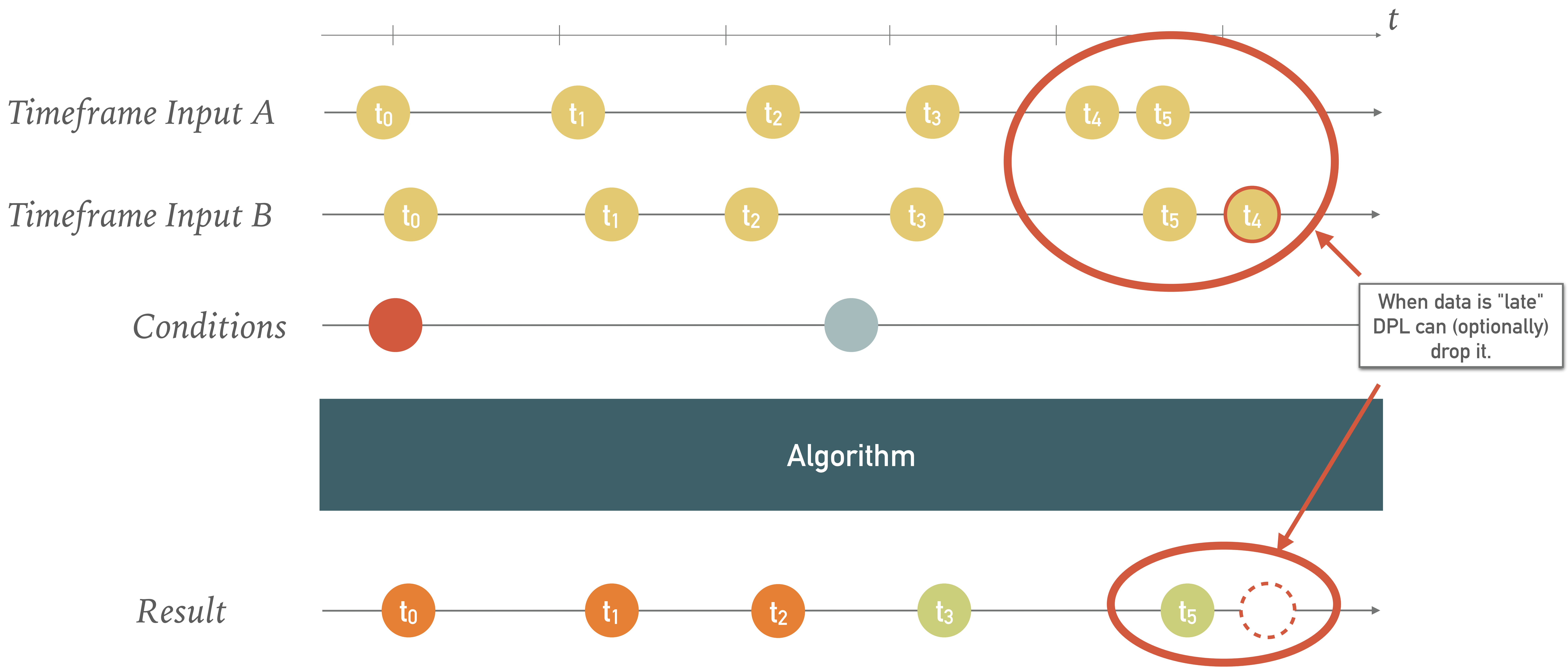
REACTIVE DESIGN

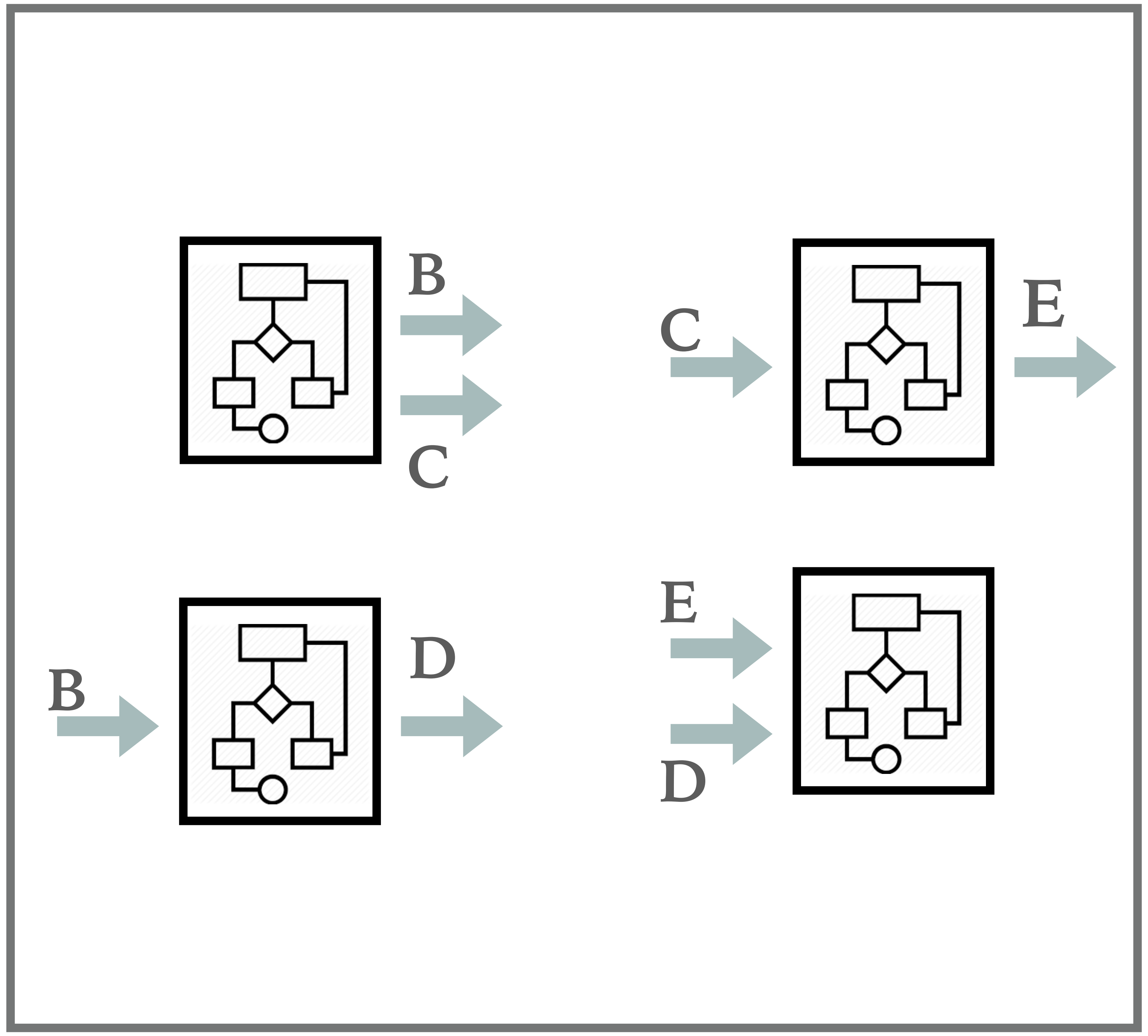


REACTIVE DESIGN



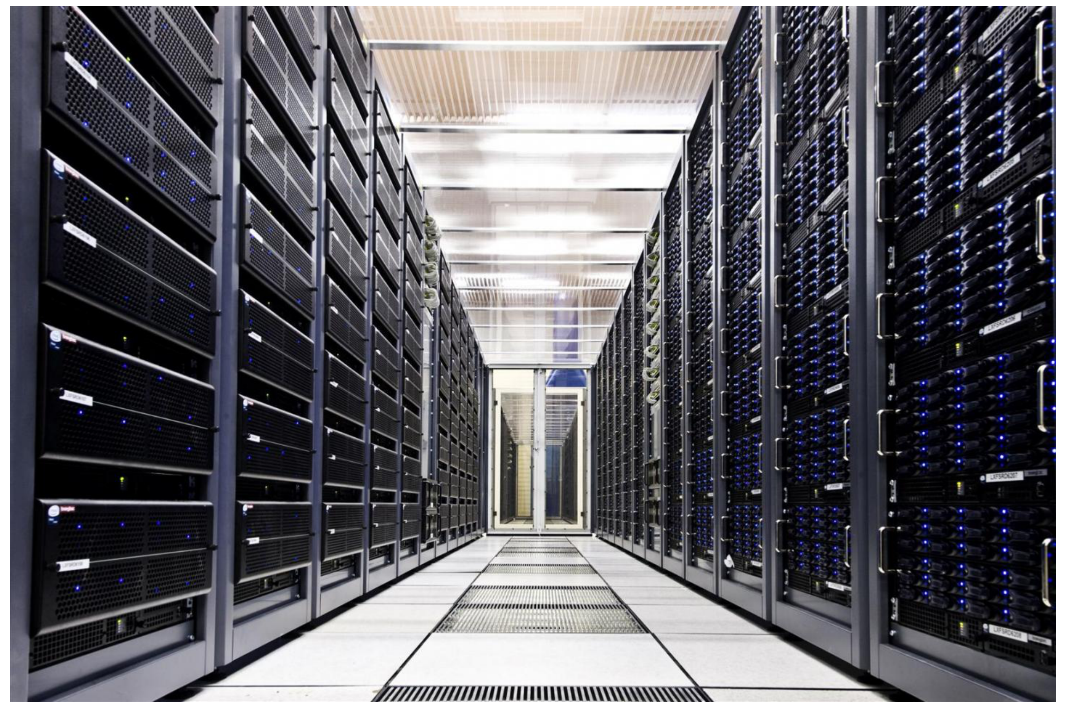
REACTIVE DESIGN





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 ongoing.**

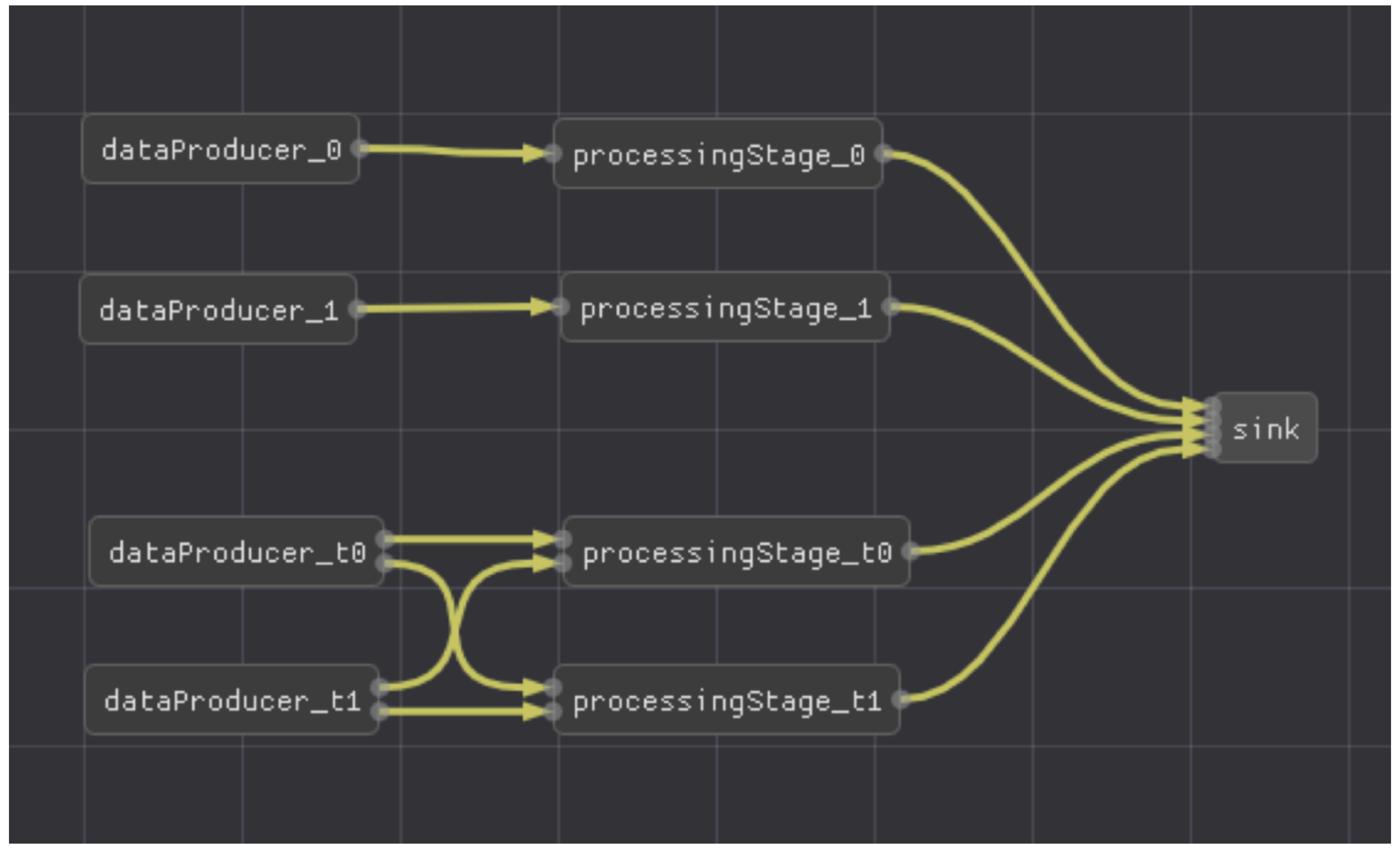
**see Teo's talk: https://indico.cern.ch/event/587955/contributions/2935762/*

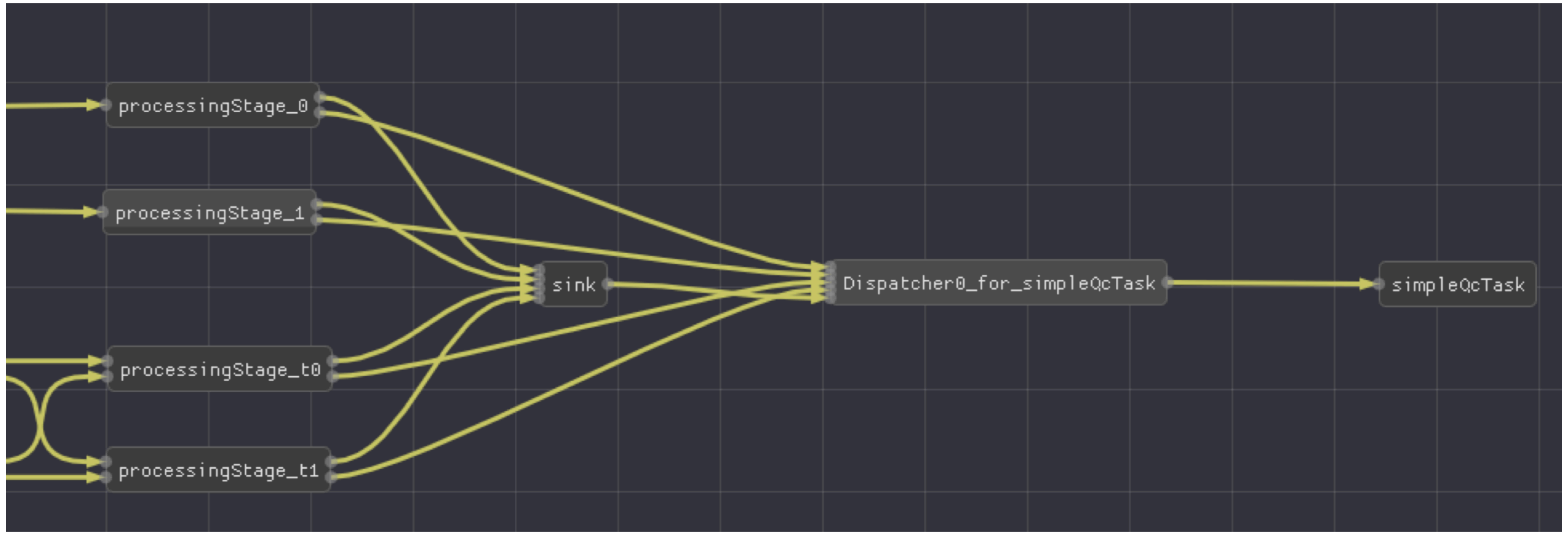
```cpp
#include "Framework/runDataProcessing.h"

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) {}}
  }
  };
}
```

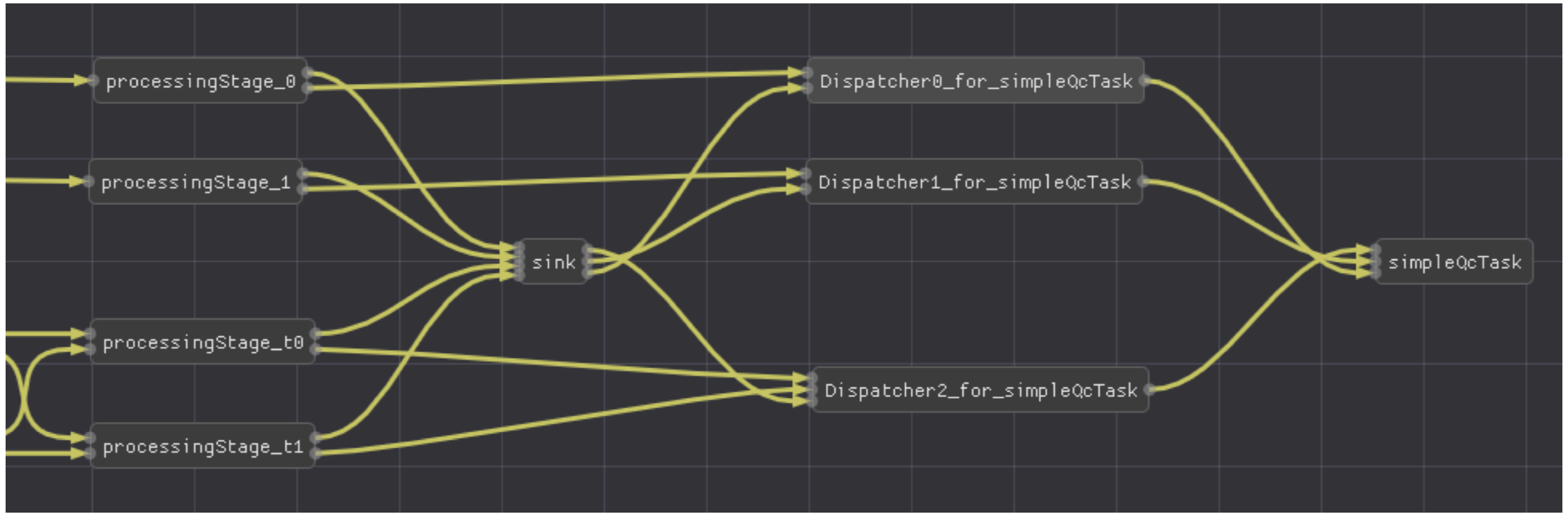
The previous example (GUI included) requires 27 user's SLOC.

A FEW EXAMPLES

COMPOSABLE WORKFLOWS

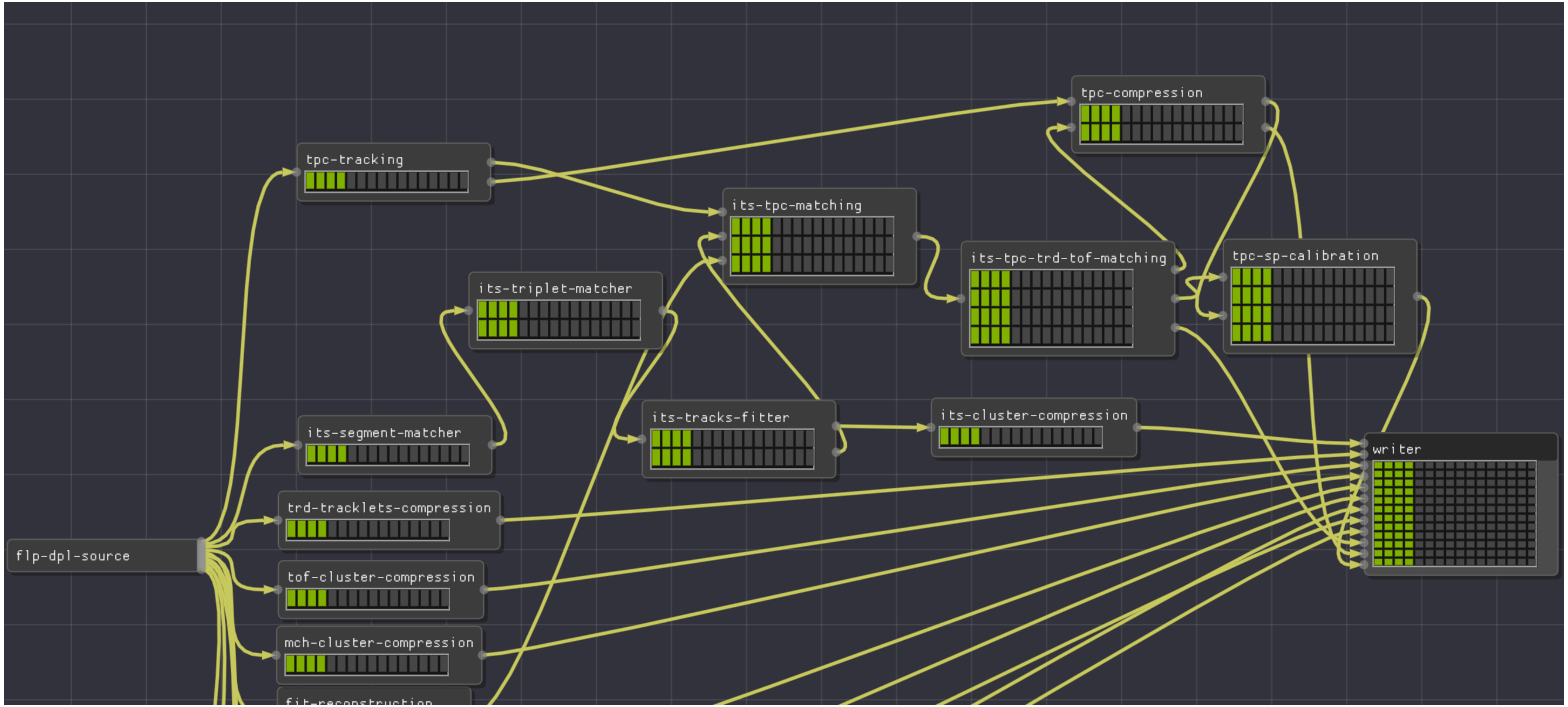






Declarative configuration allows for easy customisation: e.g. adding a (one or more) dispatchers for QA.

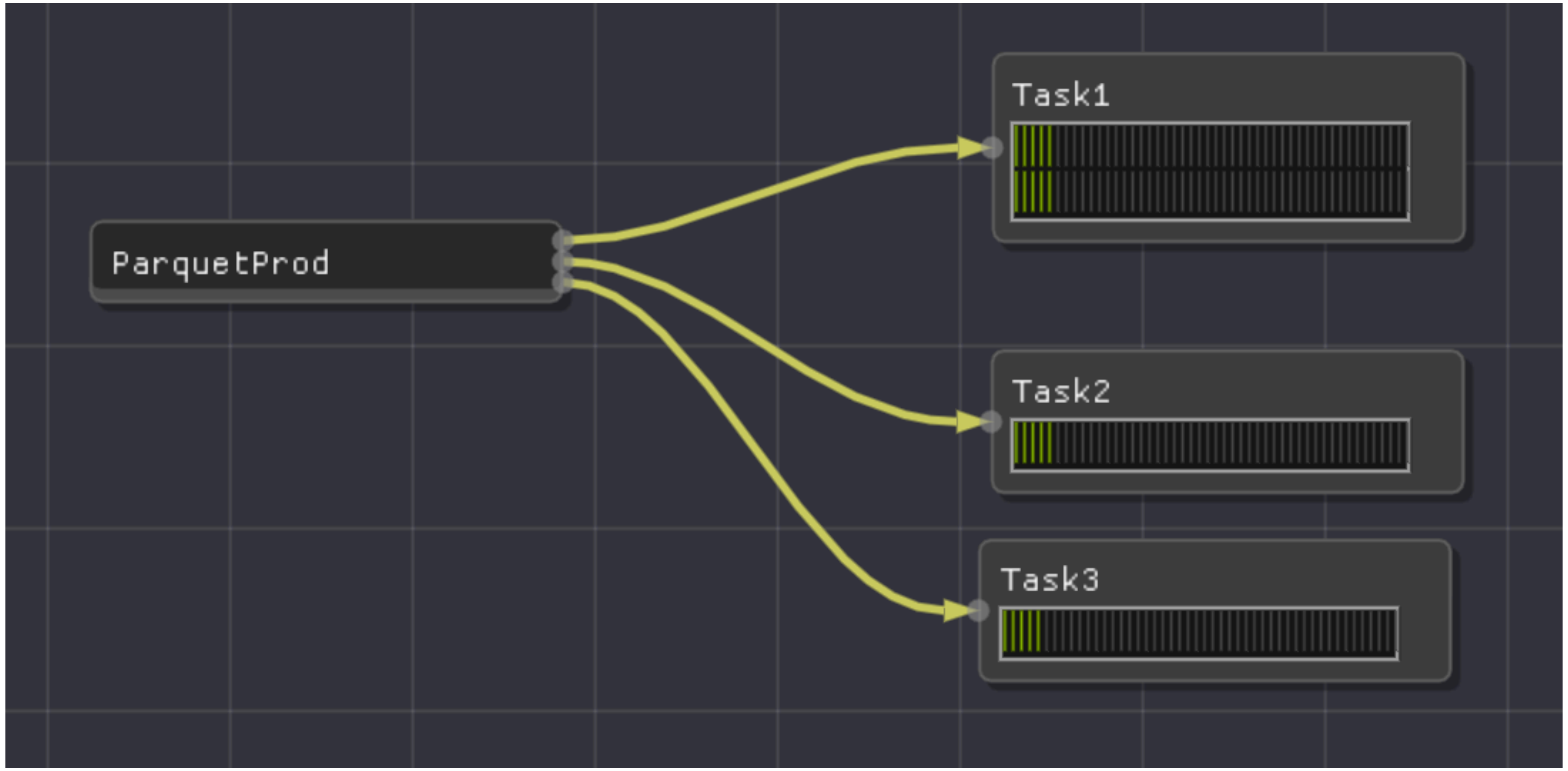
RECONSTRUCTION & GENERAL DATAFLOW



See talk from Matthias https://indico.cern.ch/event/587955/contributions/2935788/

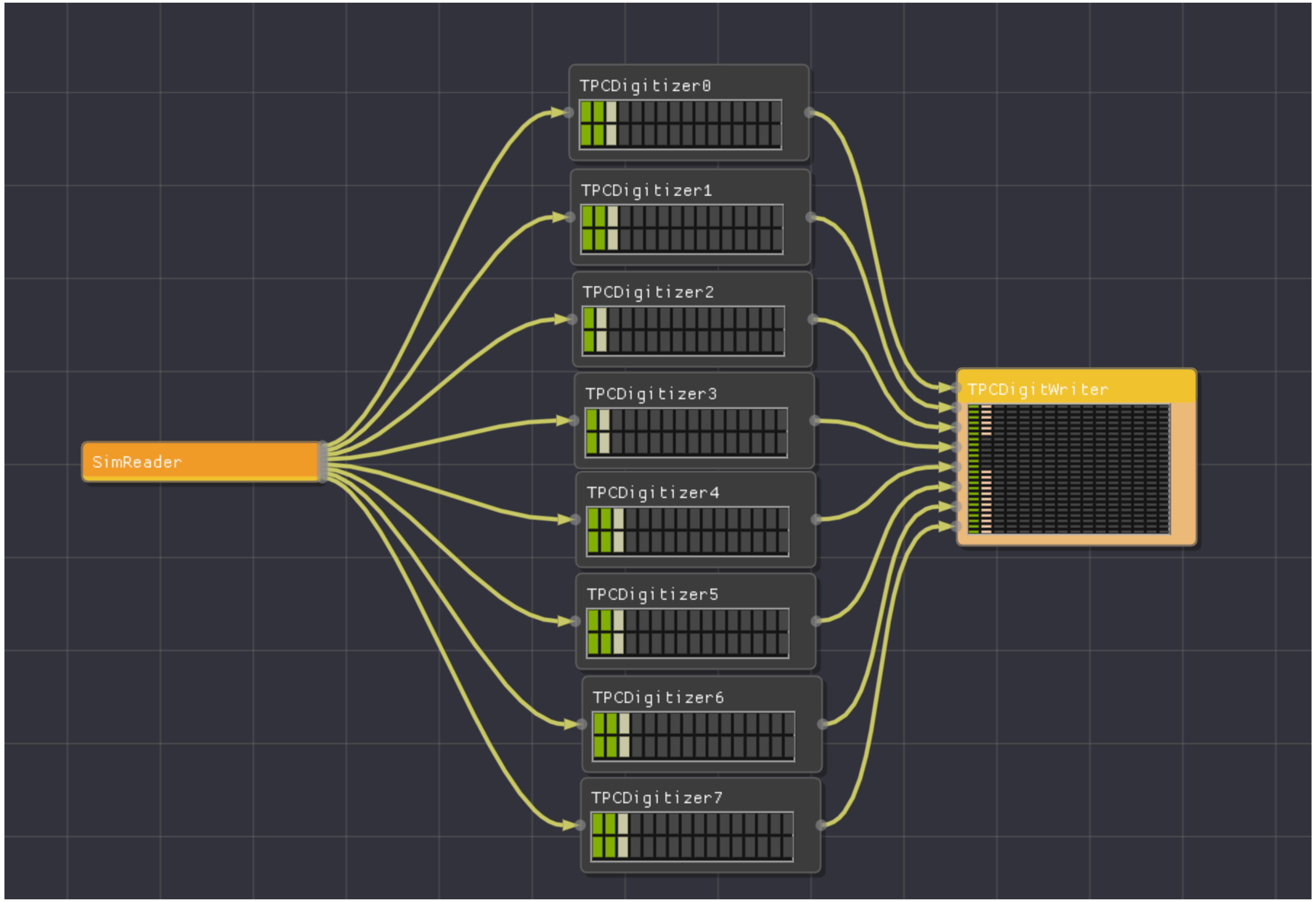
DPL AND ANALYSIS

We are investigating about using the Data Processing Layer also for Analysis.



See talk by Dario https://indico.cern.ch/event/587955/contributions/2938126/

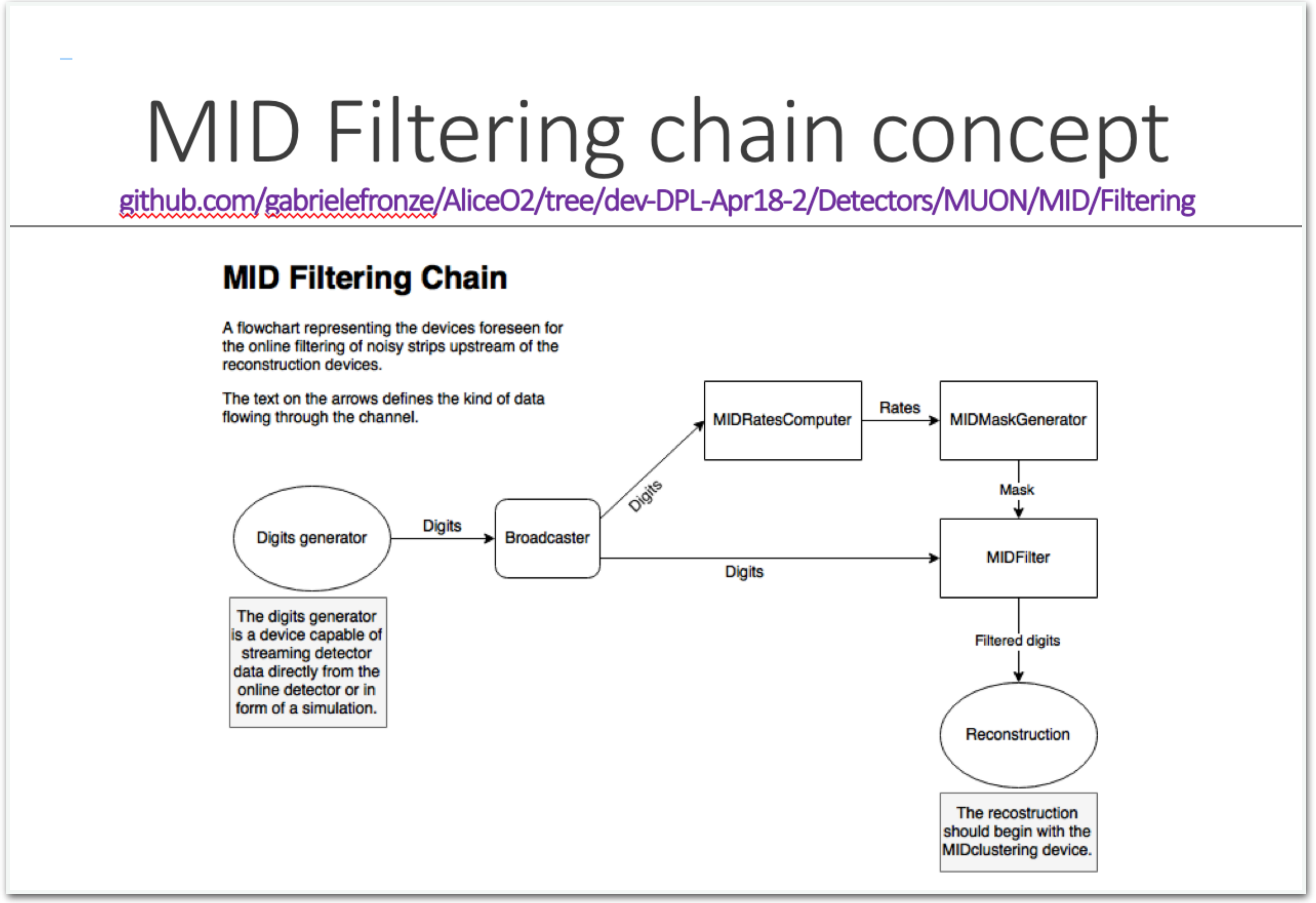
PARALLEL DIGITIZATION

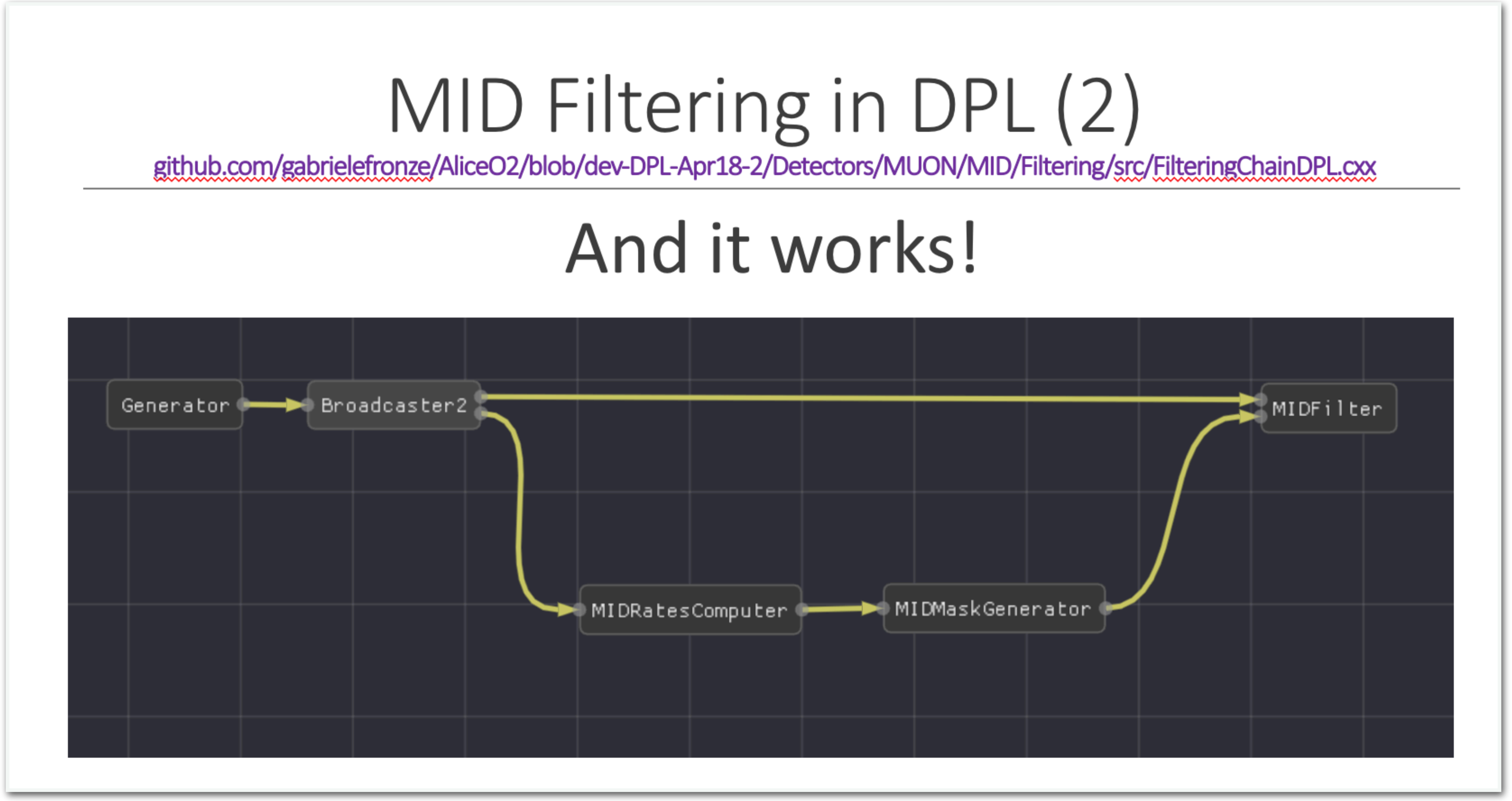


See talk from Sandro: "A scalable and asynchronous detector simulation system based on ALFA" (https://indico.cern.ch/event/587955/contributions/2937621/)

DPL USAGE: MID FILTERING CHAIN

Nice demonstrator by Gabriele Fronzè for MID filtering.





SUMMARY

- The challenges posed by Run 3 imposed to rethink ALICE Computing Architecture, **blending the traditional Online and Offline roles**.
- The **message passing ALFA Framework** is the foundation of ALICE O2 Software Framework.
- We built a message passing / shared memory friendly data model which minimises copy and (de-)serialisation.
- Taking advantage of the O2 Data Model we build a **data flow engine on top of ALFA** to reduce user code and abstract away common hiccups of distributed systems.

BACKUP

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 readout. Vast majority in TPC clusters.*
- *Compressed to ~2GB after asynchronous reconstruction, mostly thanks to track-model-compression.*
- *100x the number of collisions of RUN2.*
- *All MinBias. We need to (lossly) compress information, not filter it.*

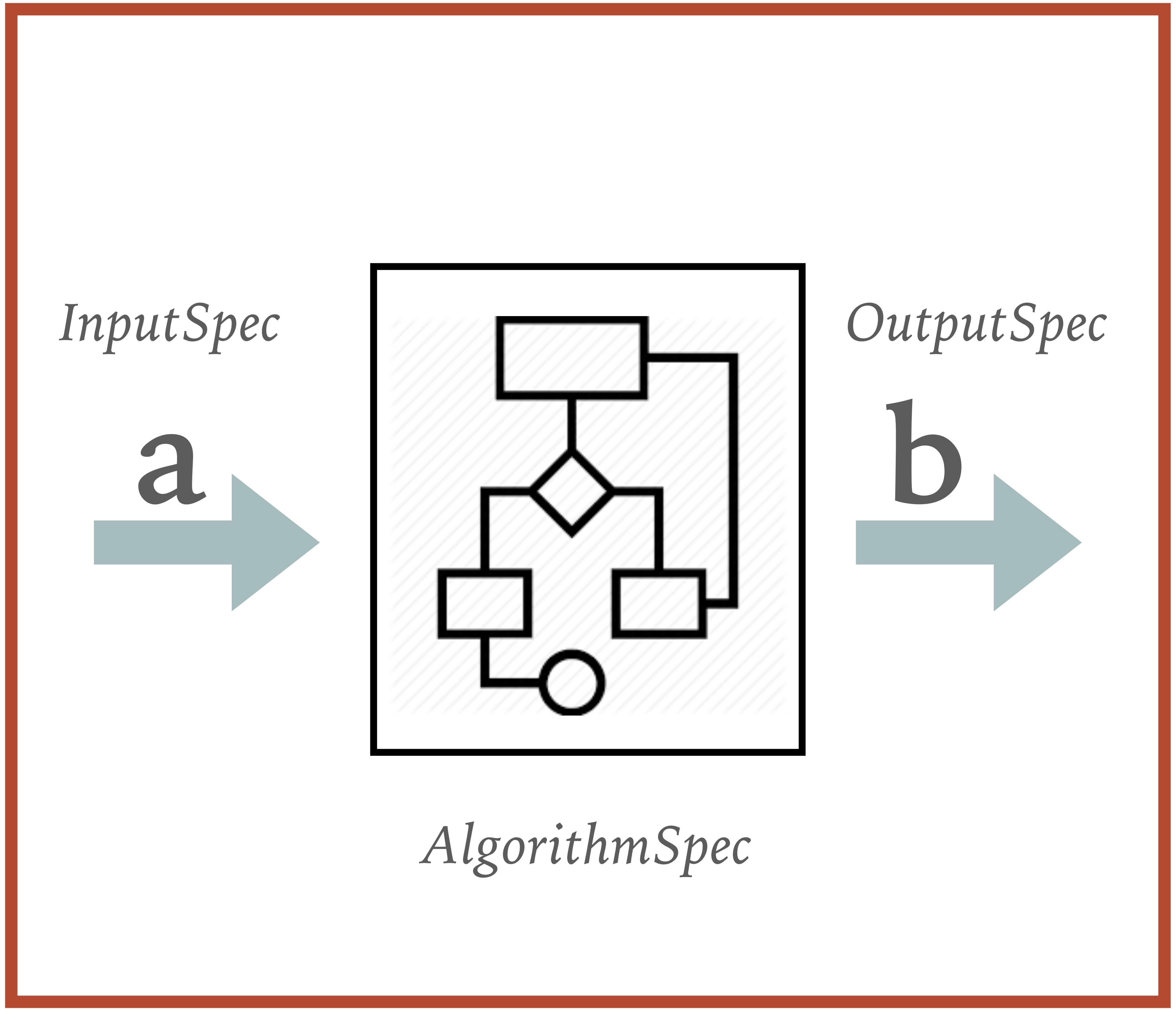
O2 DATA MODEL

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

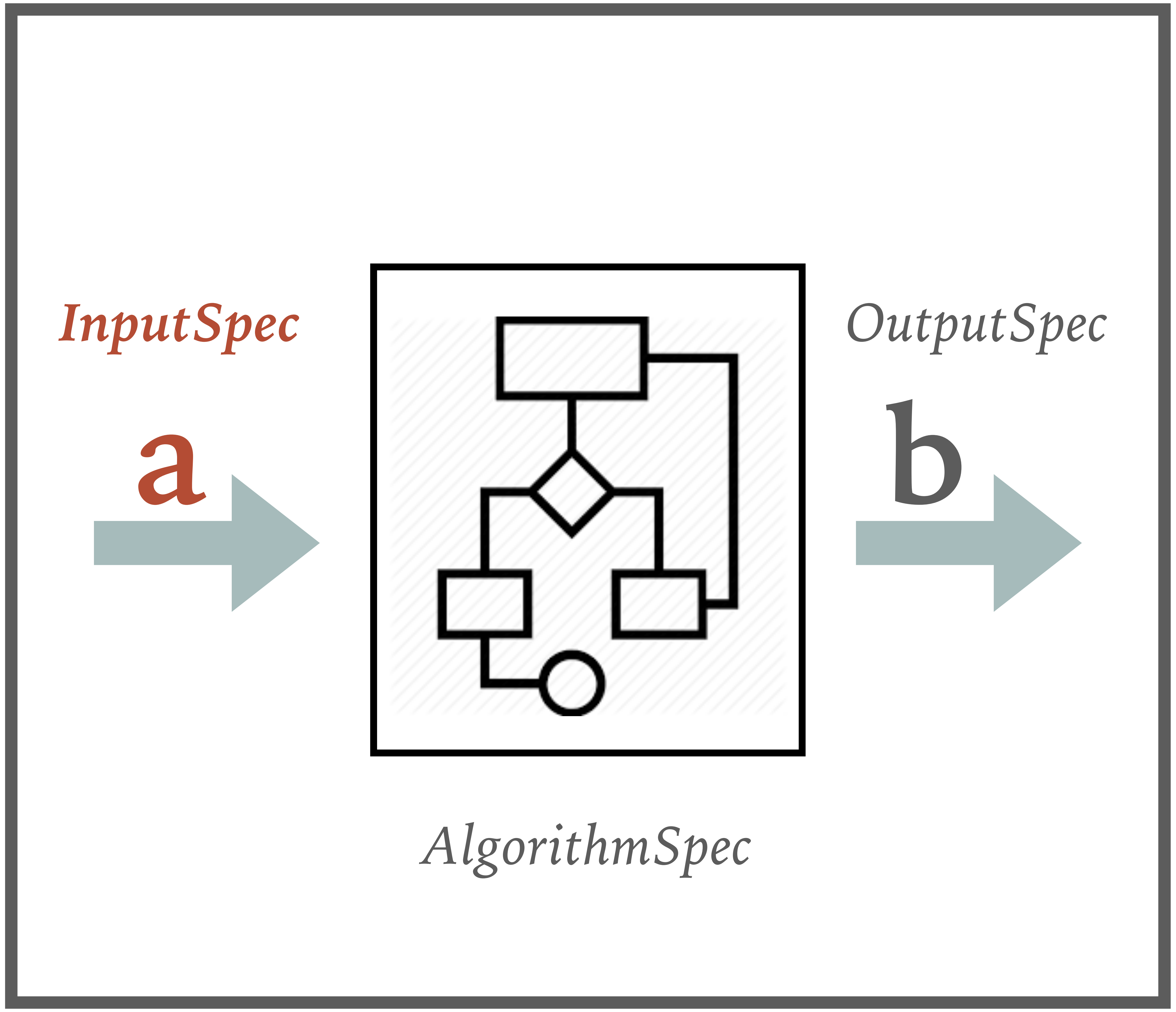
DATA PROCESSING LAYER: HOW



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);
    }
  }
}
```

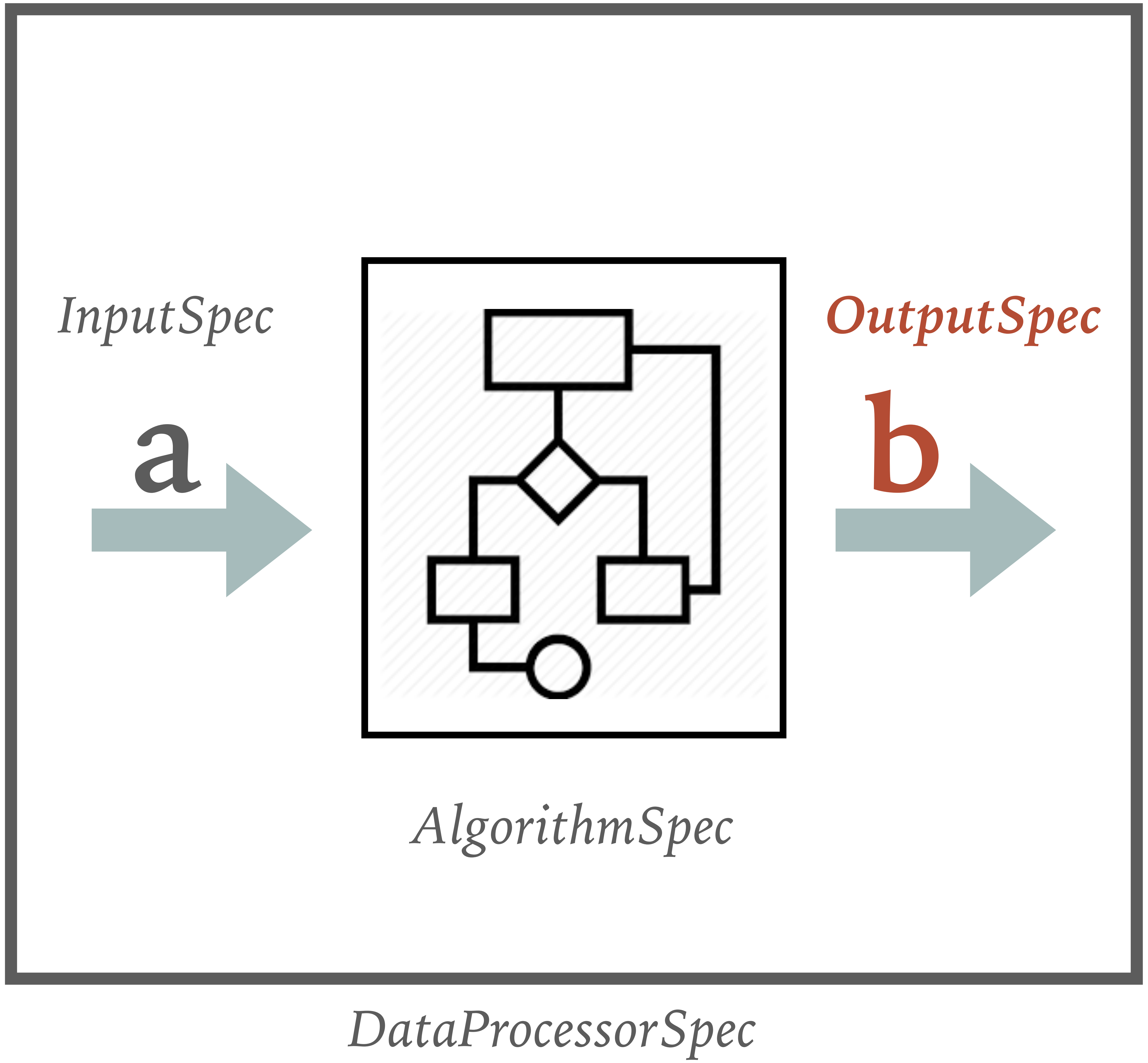
DATA PROCESSING LAYER: HOW



DataProcessorSpec

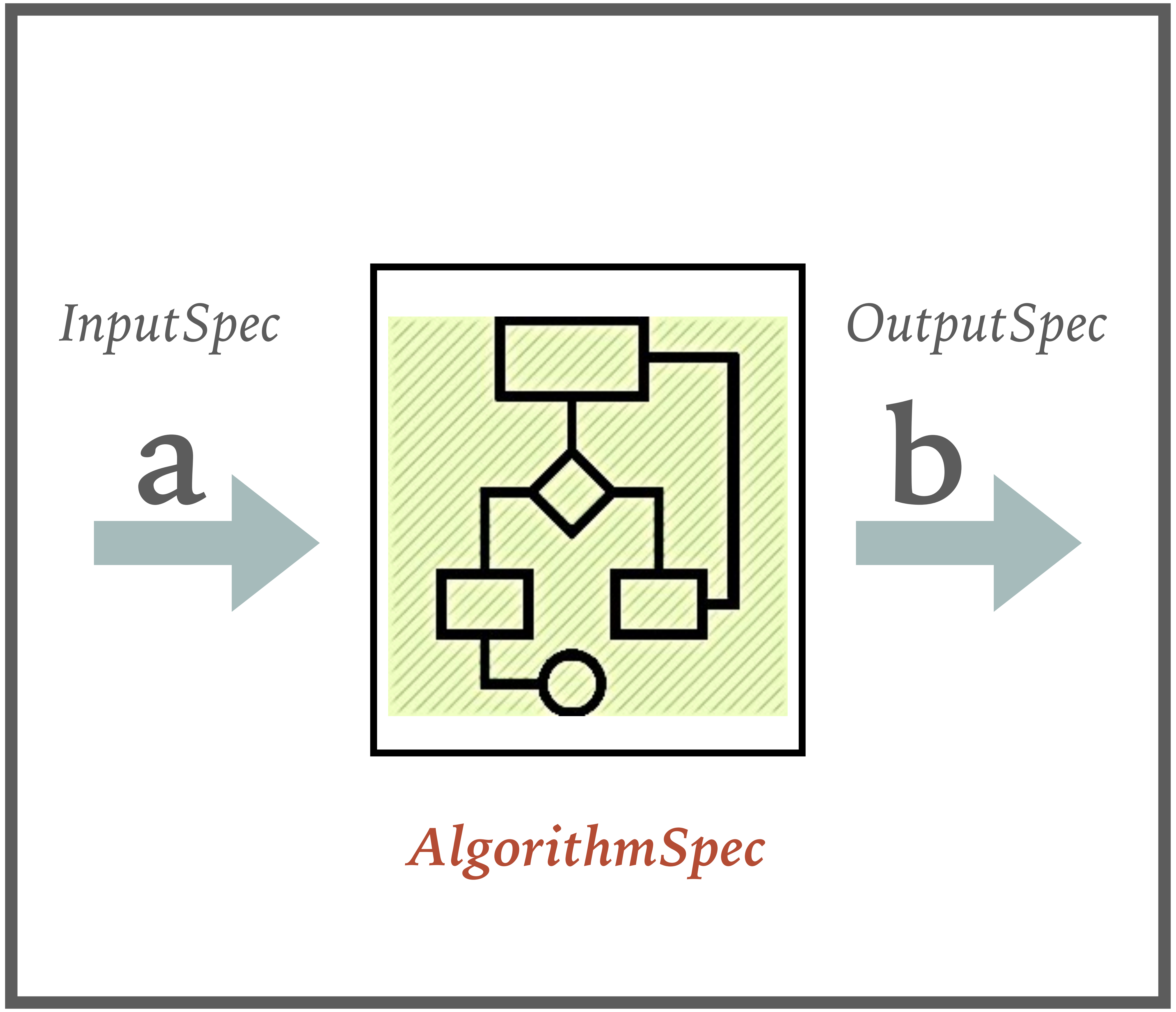
```
DataProcessorSpec{
  "A",
  Inputs{
   InputSpec{"a", "TPC", "CLUSTERS"}
  },
  Outputs{
   OutputSpec{{"b"}, "TPC", "TRACKS"}
  },
  AlgorithmSpec{
    [](ProcessingContext &ctx) {
       auto track = ctx.outputs().make<Track>(OutputRef{ "b" }, 1);
    }
  }
}
```

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);
    }
  }
}
```

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);
    }
  }
}
```

HOW DO YOU LIMIT CONTEXT SWITCH COSTS?

We will have a number of **running** processes which is ≲ the number of cores.

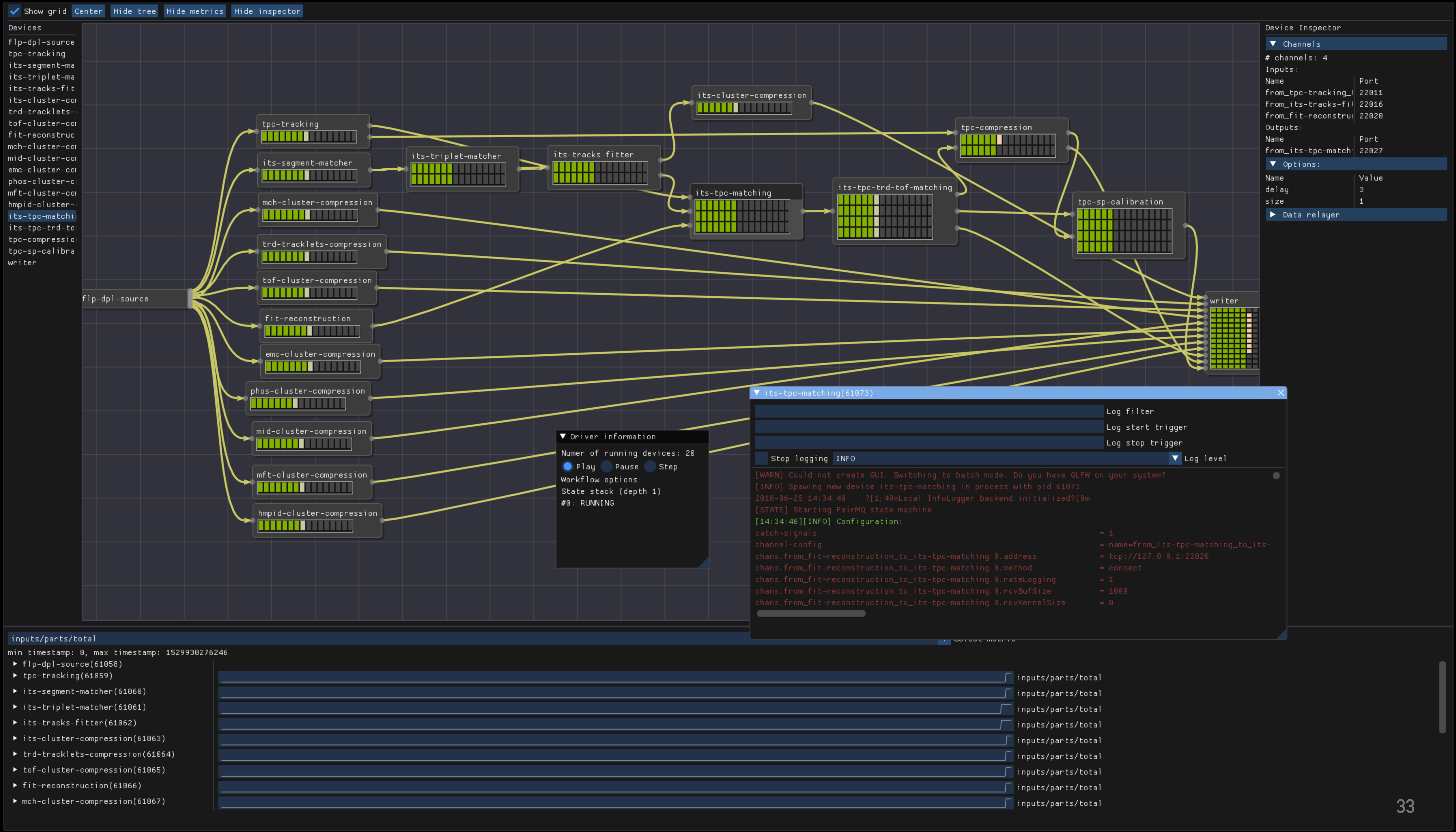
Our tasks **take long on a CPU scale** (seconds) thanks to the fact we treat one timeframe at the time (~1000 collisions). User code runs lock free.

By describing our computation in terms of **composable pipeline stages** we keep door open for (eventually dynamic) NxM mapping between data processors and actual processes.

We are **willing to pay an extra** price for the sake of:

- *Ease of deployment (microservices!)*
- *Crash resilience (data taking!)*
- *Ability to distribute over multiple nodes (HPC!)*
- *Flexibility (run GEANT3 + GEANT4 + FLUKA)*

Limiting factor is in any case the GPU for TPC tracking (at least for the synchronous phase).


Show grid
Center
Hide tree
Hide metrics
Hide inspector
Devices
flp-dpl-source
tpc-tracking
its-segment-ma
its-triplet-ma
its-tracks-fit
its-cluster-cor
trd-tracklets-
tof-cluster-cor
fit-reconstruc
mch-cluster-cor
mid-cluster-cor
emc-cluster-cor
phos-cluster-c
mft-cluster-cor
hmpid-cluster-
its-tpc-matchi
its-tpc-trd-to
tpc-compressio
tpc-sp-calibra
writer
flp-dpl-source
tpc-tracking
its-segment-matcher
mch-cluster-compression
trd-tracklets-compression
tof-cluster-compression
fit-reconstruction
emc-cluster-compression
phos-cluster-compression
mid-cluster-compression
mft-cluster-compression
hmpid-cluster-compression
its-triplet-matcher
its-tracks-fitter
its-cluster-compression
its-tpc-matching
its-tpc-trd-tof-matching
tpc-compression
tpc-sp-calibration
writer
Device Inspector
Channels
channels: 4
Inputs:
Name
Port
from_tpc-tracking_
22011
from_its-tracks-fi
22016
from_fit-reconstruc
22020
Outputs:
Name
Port
from_its-tpc-match
22027
Options:
Name
Value
delay
3
size
1
Data relayer
Driver information
Numer of running devices: 20
Play
Pause
Step
Workflow options:
State stack (depth 1)
#0: RUNNING
its-tpc-matching(61073)
Log filter
Log start trigger
Log stop trigger
Stop logging
INFO
Log level
[WARN] Could not create GUI. Switching to batch mode. Do you have GLFW on your system?
[INFO] Spawing new device its-tpc-matching in process with pid 61073
2018-06-25 14:34:40 ?[1;49mLocal InfoLogger backend initialized?[0m
[STATE] Starting FairMQ state machine
[14:34:40][INFO] Configuration:
catch-signals = 1
channel-config = name=from_its-tpc-matching_to_its-
chans.from_fit-reconstruction_to_its-tpc-matching.0.address = tcp://127.0.0.1:22020
chans.from_fit-reconstruction_to_its-tpc-matching.0.method = connect
chans.from_fit-reconstruction_to_its-tpc-matching.0.rateLogging = 1
chans.from_fit-reconstruction_to_its-tpc-matching.0.rcvBufSize = 1000
chans.from_fit-reconstruction_to_its-tpc-matching.0.rcvKernelSize = 0
inputs/parts/total
min timestamp: 0, max timestamp: 1529930276246
flp-dpl-source(61058)
tpc-tracking(61059)
its-segment-matcher(61060)
its-triplet-matcher(61061)
its-tracks-fitter(61062)
its-cluster-compression(61063)
trd-tracklets-compression(61064)
tof-cluster-compression(61065)
fit-reconstruction(61066)
mch-cluster-compression(61067)
inputs/parts/total