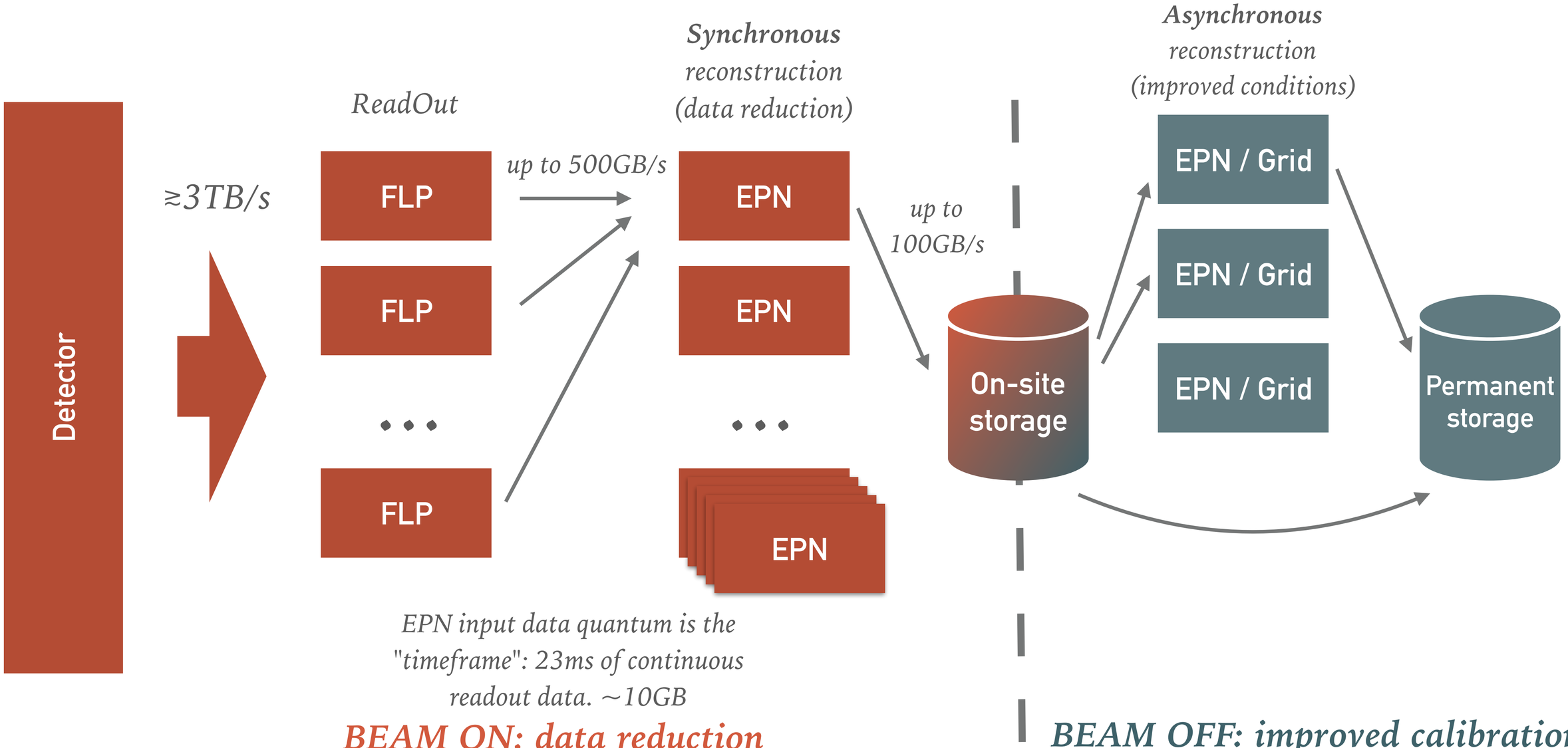
WP4 – FRAMEWORK

Giulio Eulisse - CERN EP/AIP
(Taking credit for the job of many)

BEFORE WE START: DATA PROCESSING LAYER (DPL) RECAP

A REMINDER: ALICE IN RUN 3



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 (devices)** *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 with other tools.*

Transport Layer: ALFA / FairMQ[1]

- **Standalone processes (devices)** *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 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 implicit workflow definition.*
- ***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.*

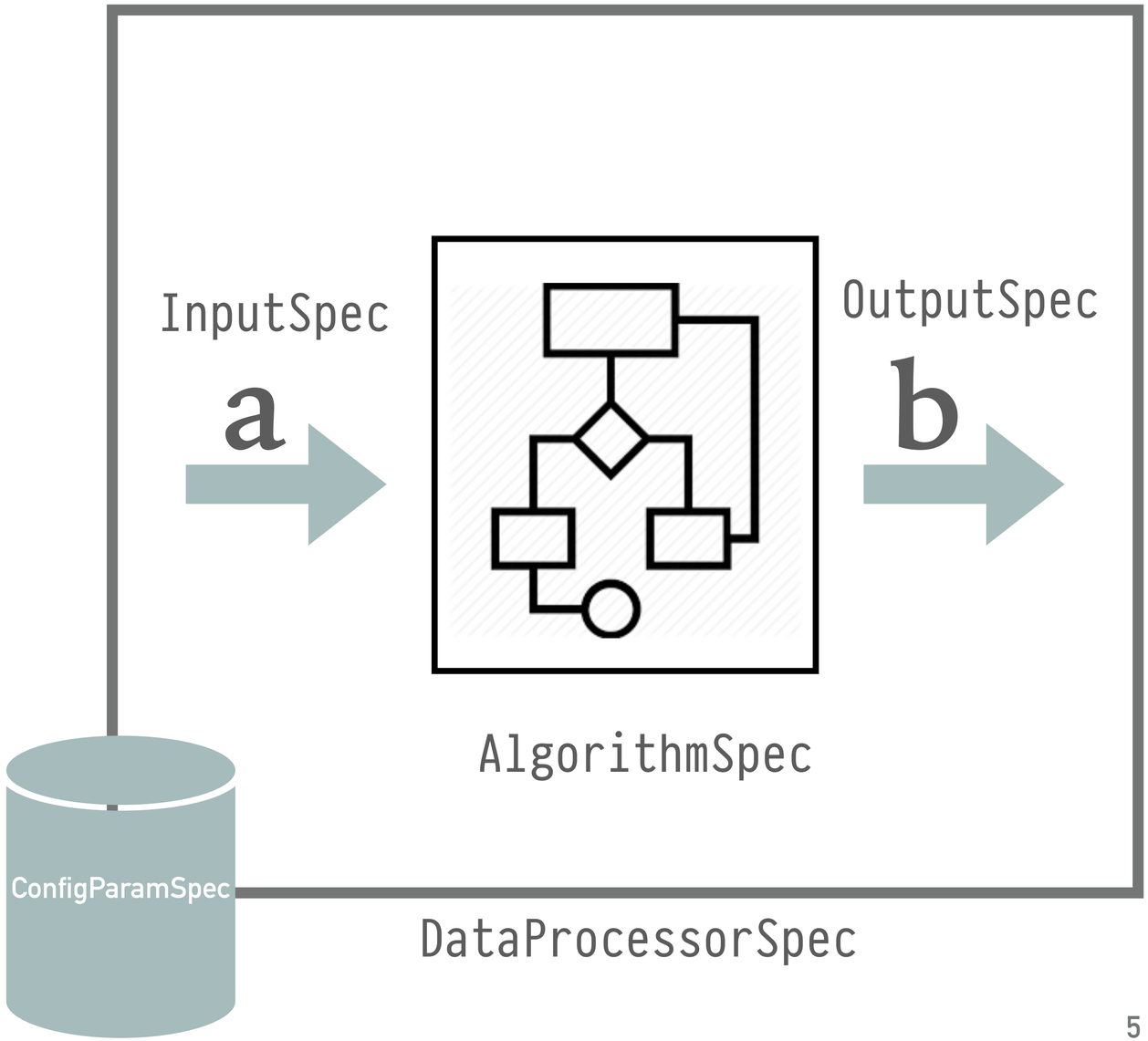
Transport Layer: ALFA / FairMQ[1]

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

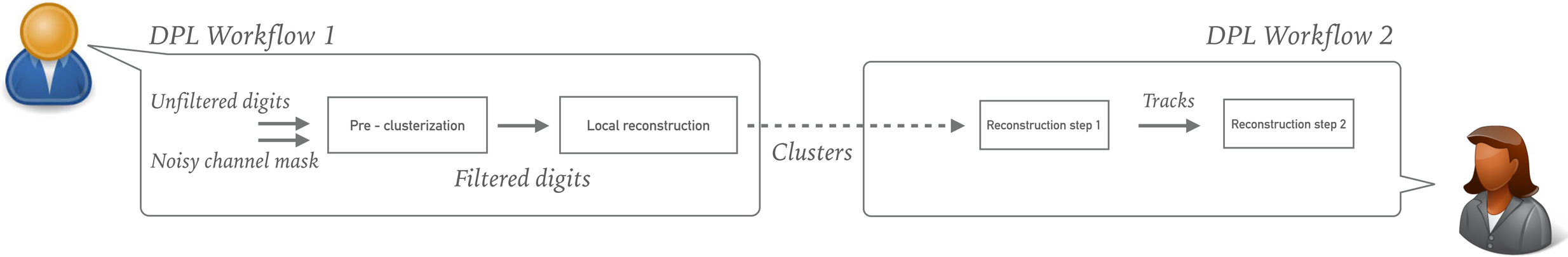
DPL: BUILDING BLOCK

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

- *Specifies **inputs and outputs** in terms of the O2 Data Model descriptors (**physics types, not sockets and ports**).*
- *Provides an implementation of how to act on the inputs to produce the output.*
- *Advanced user can express possible data or time parallelism opportunities.*
- *Integrates configuration options*

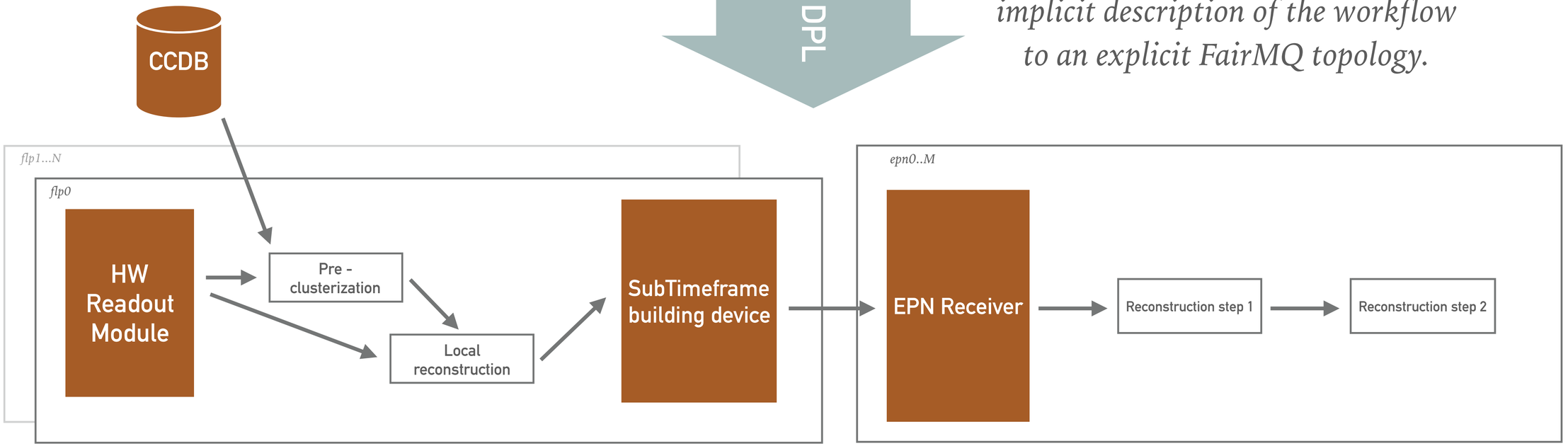


DPL IMPLICIT WORKFLOW DEFINITION



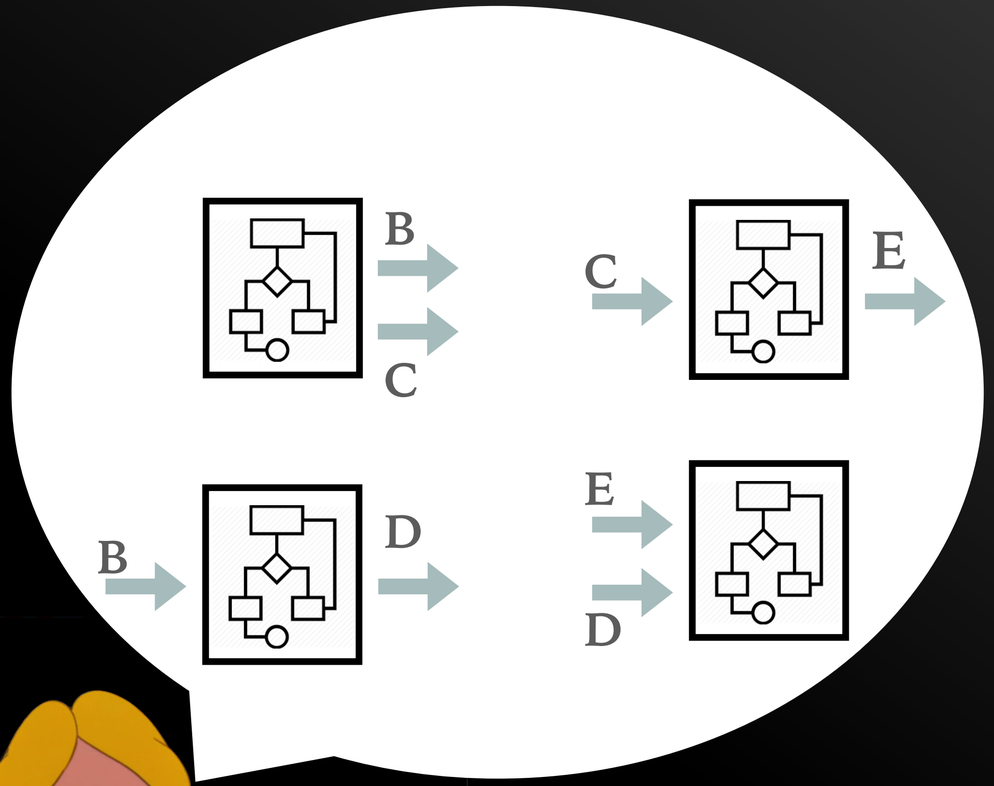
DPL

DPL converts a physics oriented implicit description of the workflow to an explicit FairMQ topology.

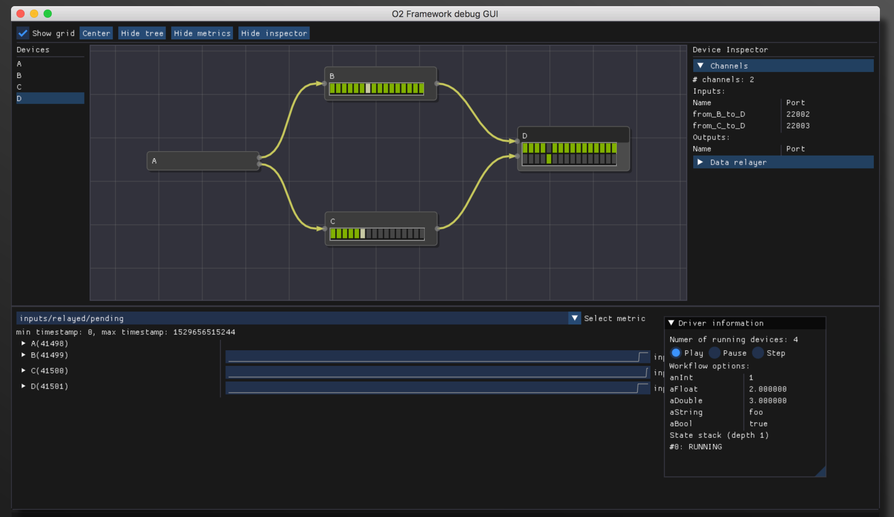


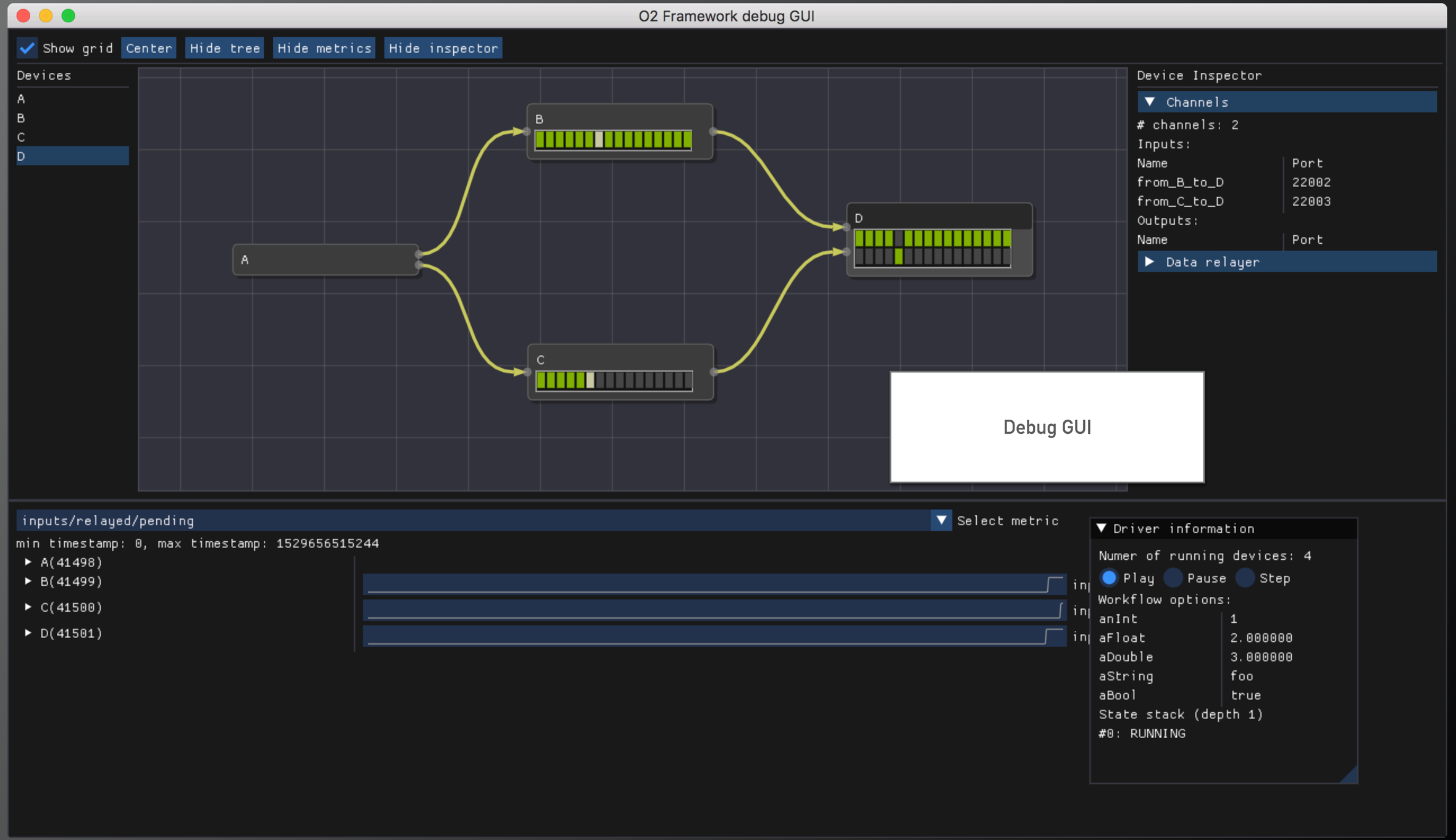
FairMQ topology

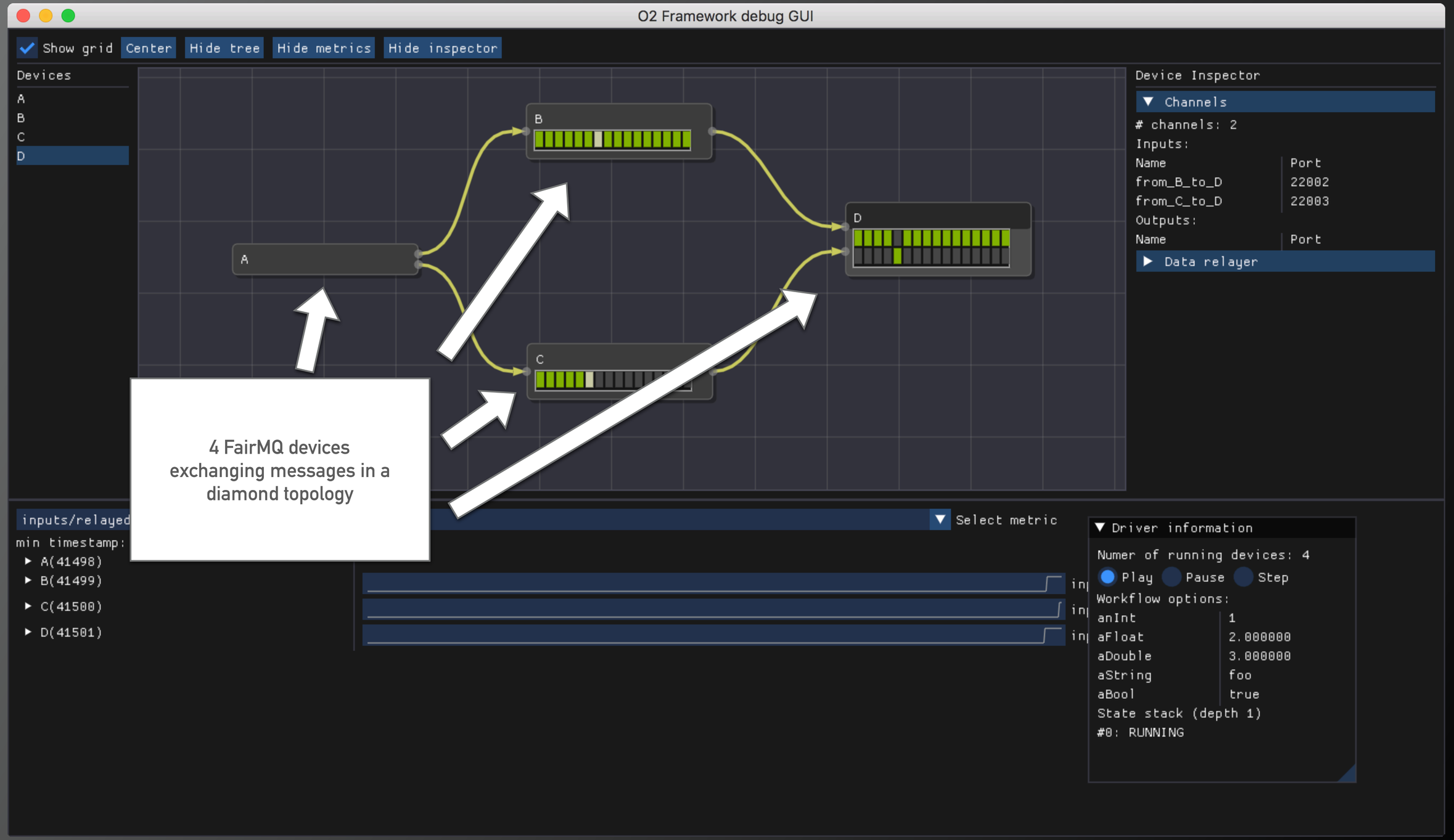
DATA PROCESSING LAYER: IMPLICIT TOPOLOGY

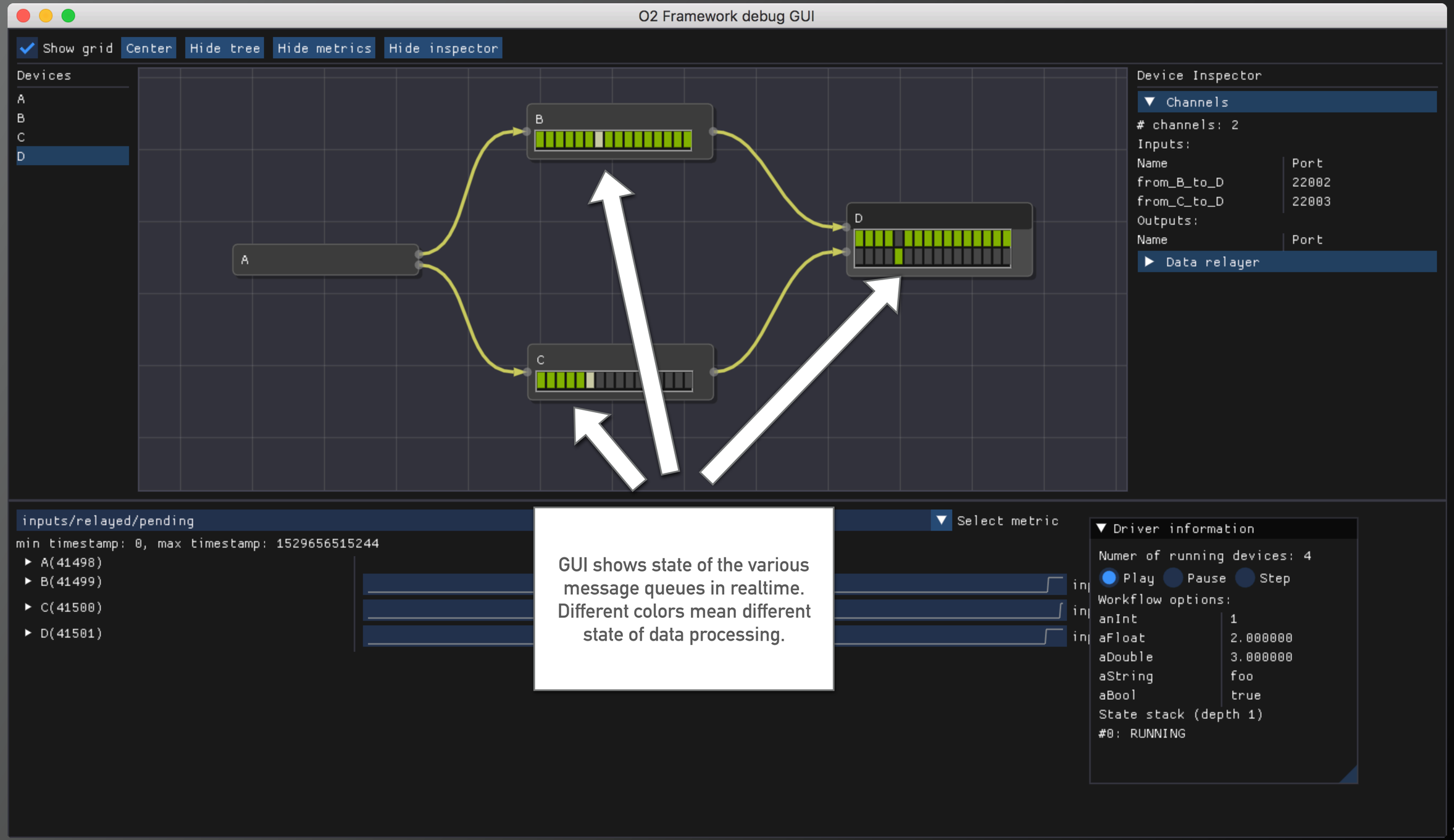


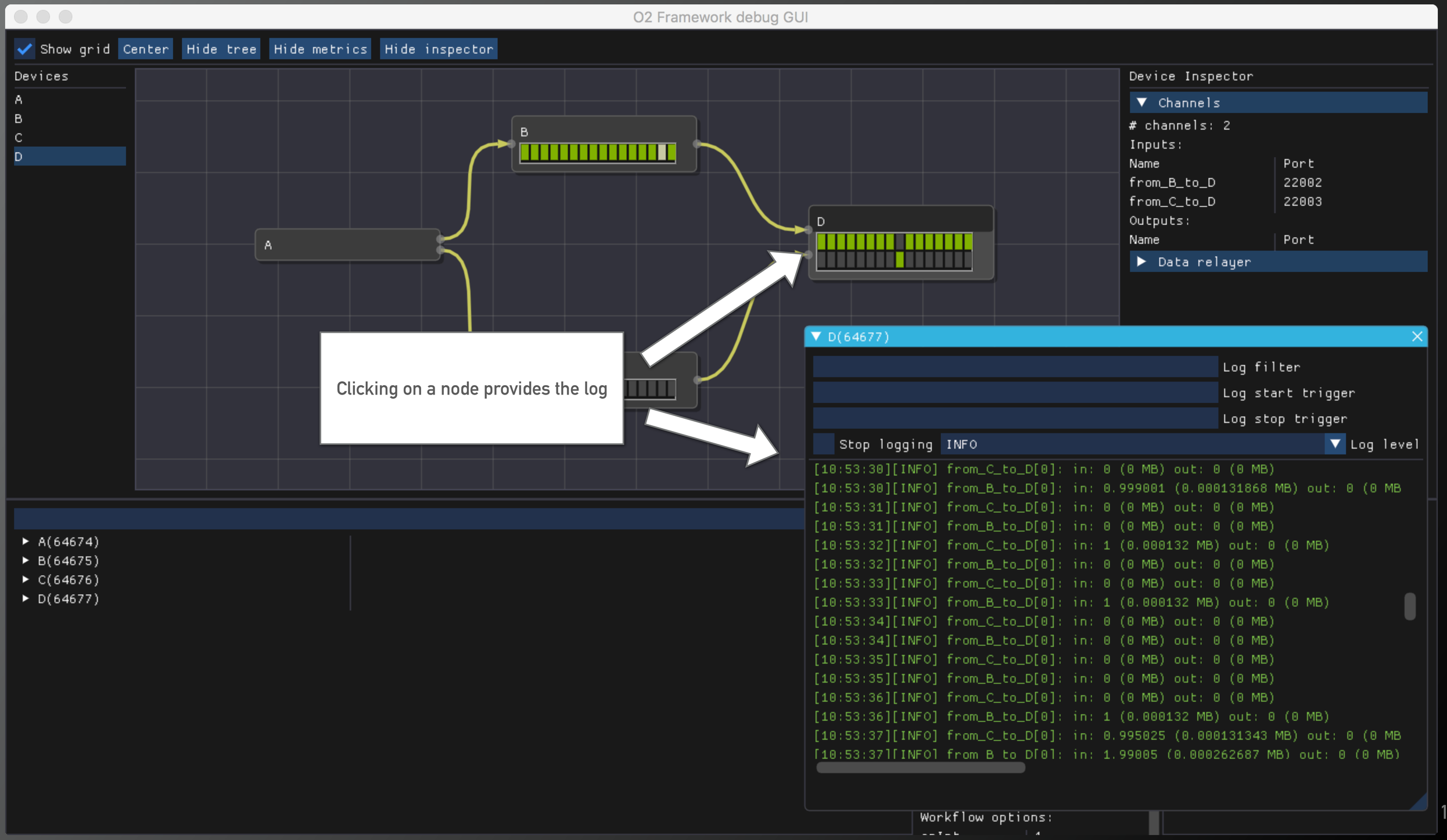
Data Processing Layer

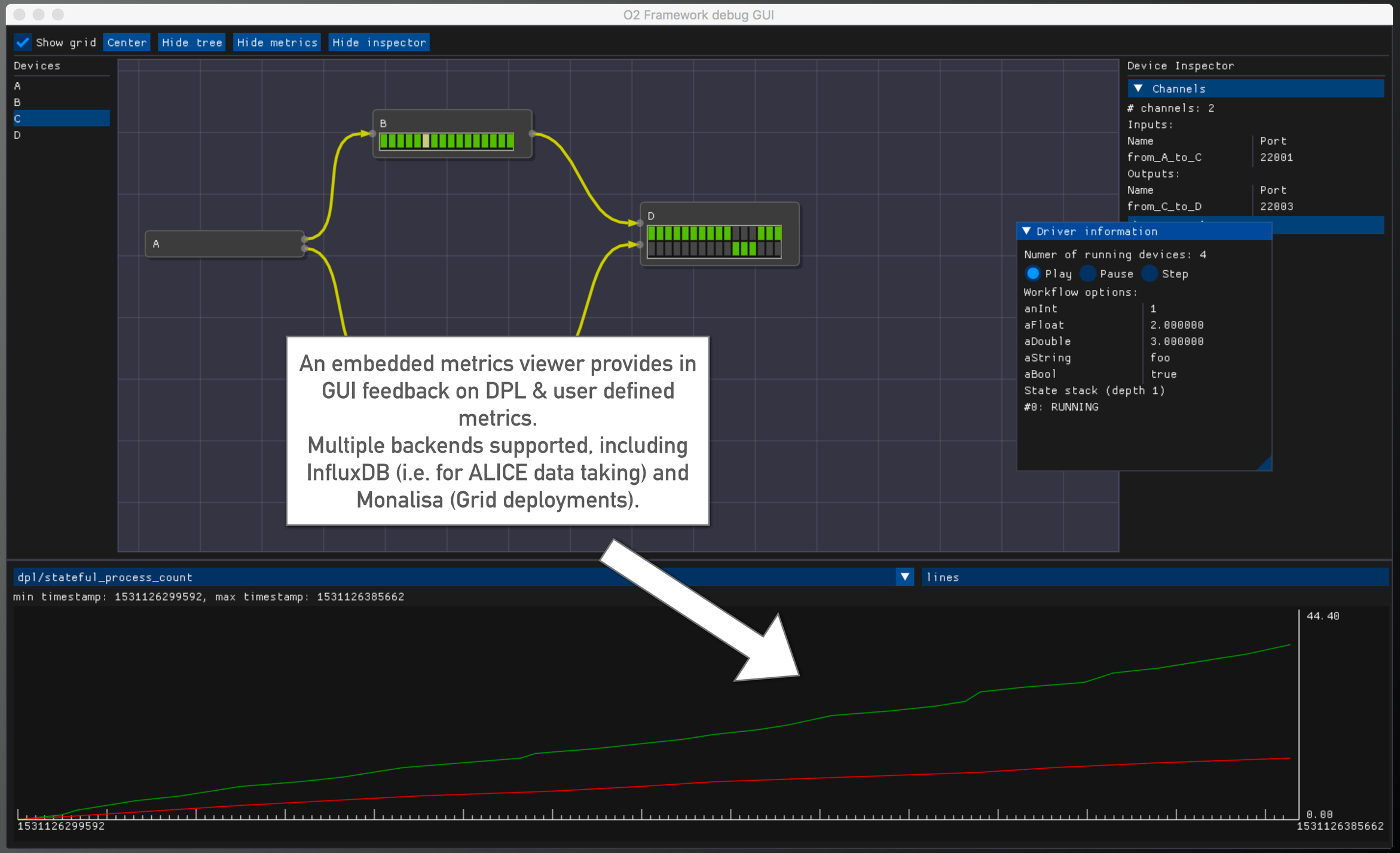



O2 Framework debug GUI
Show grid
Center
Hide tree
Hide metrics
Hide inspector
Devices
A
B
C
D
A
B
C
D
Device Inspector
Channels
channels: 2
Inputs:
Name
Port
from_B_to_D
22002
from_C_to_D
22003
Outputs:
Name
Port
Data relayer
Debug GUI
inputs/relayed/pending
Select metric
min timestamp: 0, max timestamp: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


O2 Framework debug GUI
Show grid
Center
Hide tree
Hide metrics
Hide inspector
Devices
A
B
C
D
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)
GUI shows state of the various message queues in realtime. Different colors mean different state of data processing.
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

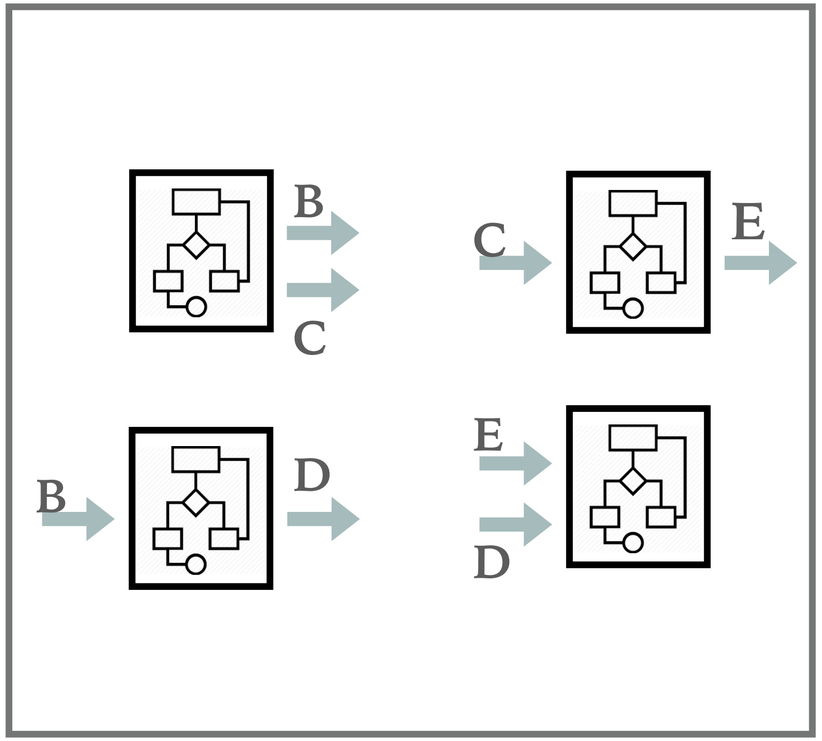

O2 Framework debug GUI
Show grid
Center
Hide tree
Hide metrics
Hide inspector
Devices
A
B
C
D
B
A
D
Clicking on a node provides the log
Device Inspector
Channels
channels: 2
Inputs:
Name
Port
from_B_to_D
22002
from_C_to_D
22003
Outputs:
Name
Port
Data relayer
D(64677)
Log filter
Log start trigger
Log 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:


O2 Framework debug GUI
Show grid
Center
Hide tree
Hide metrics
Hide inspector
Devices
A
B
C
D
B
A
D
Device Inspector
Channels
channels: 2
Inputs:
Name
Port
from_A_to_C
22001
Outputs:
Name
Port
from_C_to_D
22003
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
An embedded metrics viewer provides in GUI feedback on DPL & user defined metrics.
Multiple backends supported, including InfluxDB (i.e. for ALICE data taking) and Monalisa (Grid deployments).
dpl/stateful_process_count
lines
min timestamp: 1531126299592, max timestamp: 1531126385662
44.40
0.00
1531126299592
1531126385662

```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.

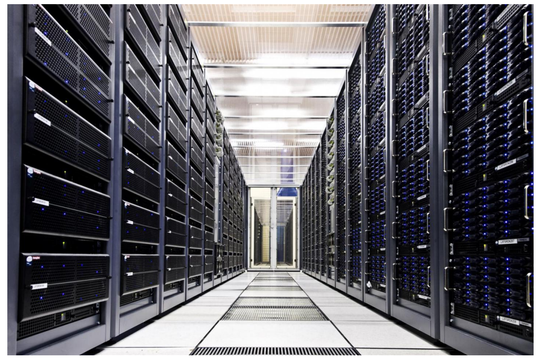
DPL as a workflow definition language: support for multiple deployment strategies.



Compiles into a single executable for the laptop user.

Generates DDS configuration for deployment on a analysis facility farm.

```
<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>
```



..or it generates the configuration to integrate with O2 Control system.

```
          TASK ID (20 TASKS)          |          CLASS NAME          |    HOSTNAME    | STATUS |  STATE
+-------------------------------------+------------------------------+----------------+--------+---------+
 952809ca-e8df-11e8-ace4-a08cfdc880fc | source-1                     | 192.168.65.111 | ACTIVE | RUNNING
 9527fa39-e8df-11e8-ace4-a08cfdc880fc | step-1                       | 192.168.65.111 | ACTIVE | RUNNING
 9527ea62-e8df-11e8-ace4-a08cfdc880fc | Dispatcher1                  | 192.168.65.111 | ACTIVE | RUNNING
 9527d1cb-e8df-11e8-ace4-a08cfdc880fc | dataSizeTask1                | 192.168.65.111 | ACTIVE | RUNNING
 9527b950-e8df-11e8-ace4-a08cfdc880fc | source-2                     | 192.168.65.111 | ACTIVE | RUNNING
 9527ac21-e8df-11e8-ace4-a08cfdc880fc | step-2                       | 192.168.65.111 | ACTIVE | RUNNING
 95279e16-e8df-11e8-ace4-a08cfdc880fc | sink-2                       | 192.168.65.111 | ACTIVE | RUNNING
 95278df9-e8df-11e8-ace4-a08cfdc880fc | Dispatcher2                  | 192.168.65.111 | ACTIVE | RUNNING
 95277e12-e8df-11e8-ace4-a08cfdc880fc | dataSizeTask2                | 192.168.65.111 | ACTIVE | RUNNING
 95276cf8-e8df-11e8-ace4-a08cfdc880fc | source-3                     | 192.168.65.111 | ACTIVE | RUNNING
 95275f23-e8df-11e8-ace4-a08cfdc880fc | step-3                       | 192.168.65.111 | ACTIVE | RUNNING
 952750a1-e8df-11e8-ace4-a08cfdc880fc | Dispatcher3                  | 192.168.65.111 | ACTIVE | RUNNING
 952742b4-e8df-11e8-ace4-a08cfdc880fc | dataSizeTask3                | 192.168.65.111 | ACTIVE | RUNNING
 95273026-e8df-11e8-ace4-a08cfdc880fc | dataSizeTask-merger          | 192.168.65.111 | ACTIVE | RUNNING
 952728b3-e8df-11e8-ace4-a08cfdc880fc | dataSizeTask-checker         | 192.168.65.111 | ACTIVE | RUNNING
 95271bc5-e8df-11e8-ace4-a08cfdc880fc | someNumbersTask              | 192.168.65.111 | ACTIVE | RUNNING
 95270902-e8df-11e8-ace4-a08cfdc880fc | someNumbersTask-checker      | 192.168.65.111 | ACTIVE | RUNNING
 9526f670-e8df-11e8-ace4-a08cfdc880fc | sink-1                       | 192.168.65.111 | ACTIVE | RUNNING
 9526e183-e8df-11e8-ace4-a08cfdc880fc | sink-3                       | 192.168.65.111 | ACTIVE | RUNNING
 95267e1e-e8df-11e8-ace4-a08cfdc880fc | dpl-global-binary-file-sink  | 192.168.65.111 | ACTIVE | RUNNING

workflow:
[RUNNING]  qc-advanced-root
├── [RUNNING]  source-1                          ──> task 952809ca-e8df-11e8-ace4-a08cfdc880fc
├── [RUNNING]  step-1                            ──> task 9527fa39-e8df-11e8-ace4-a08cfdc880fc
├── [RUNNING]  Dispatcher1                       ──> task 9527ea62-e8df-11e8-ace4-a08cfdc880fc
├── [RUNNING]  dataSizeTask1                     ──> task 9527d1cb-e8df-11e8-ace4-a08cfdc880fc
├── [RUNNING]  source-2                          ──> task 9527b950-e8df-11e8-ace4-a08cfdc880fc
├── [RUNNING]  step-2                            ──> task 9527ac21-e8df-11e8-ace4-a08cfdc880fc
├── [RUNNING]  sink-2                            ──> task 95279e16-e8df-11e8-ace4-a08cfdc880fc
├── [RUNNING]  Dispatcher2                       ──> task 95278df9-e8df-11e8-ace4-a08cfdc880fc
├── [RUNNING]  dataSizeTask2                     ──> task 95277e12-e8df-11e8-ace4-a08cfdc880fc
├── [RUNNING]  source-3                          ──> task 95276cf8-e8df-11e8-ace4-a08cfdc880fc
├── [RUNNING]  step-3                            ──> task 95275f23-e8df-11e8-ace4-a08cfdc880fc
├── [RUNNING]  Dispatcher3                       ──> task 952750a1-e8df-11e8-ace4-a08cfdc880fc
├── [RUNNING]  dataSizeTask3                     ──> task 952742b4-e8df-11e8-ace4-a08cfdc880fc
├── [RUNNING]  dataSizeTask-merger               ──> task 95273026-e8df-11e8-ace4-a08cfdc880fc
├── [RUNNING]  dataSizeTask-checker              ──> task 952728b3-e8df-11e8-ace4-a08cfdc880fc
├── [RUNNING]  someNumbersTask                   ──> task 95271bc5-e8df-11e8-ace4-a08cfdc880fc
├── [RUNNING]  someNumbersTask-checker           ──> task 95270902-e8df-11e8-ace4-a08cfdc880fc
├── [RUNNING]  sink-1                            ──> task 9526f670-e8df-11e8-ace4-a08cfdc880fc
├── [RUNNING]  sink-3                            ──> task 9526e183-e8df-11e8-ace4-a08cfdc880fc
└── [RUNNING]  dpl-global-binary-file-sink       ──> task 95267e1e-e8df-11e8-ace4-a08cfdc880fc
```

UPDATES SINCE NOVEMBER

DPL: CCDB SUPPORT

Initial CCDB support in DPL

Specify **`Lifetime::Condition`** *in an* **`InputSpec`** *to fetch objects from CCDB via libCURL.*

URL is constructed with **`<prefix>/<origin>/<description>/<timestamp>`**.

- **`<prefix>`**: *specified via* **`--condition-backend`**, *defaults to* `localhost:8080` ⇐ *Deployment dependent*
- **`<origin>/<description>`**: *from the* `InputSpec` ⇐ *User provided*
- **`<timestamp>`**: `DataProcessingHeader` *timestamp* ⇐ *DPL provided*

To come

- *Refactoring of Barth's* `CCDBApi` *class to unify implementations and provide standalone implementation*
- *Standalone proxy device to mediate requests*
- *Caching support (i.e. properly handling E-Tag).*

DPL: ANALYSIS SUPPORT

`RDataFrame`

ROOT's take on modern analysis frameworks. Provides a ***high level interface to describe analysis in a declarative way****.* ***DPL support running `RDataFrame` on incoming inputs thanks to `RArrowDS`****, the RDataSource which allows RDataFrame to ingest Apache Arrow formatted messages.*

`RCombinedDS`

Allows combining two or more ***`RDataSources`****, effectively permitting* ***joins / double / triple loops /event mixing / filtered collections*** *with RDataFrame. In O2 repository for now, integration in upstream ROOT ongoing.*

*Just a building block: "****Analysis Framework****" to provide helpers to reduce boilerplate for common tasks.*

See backup slides for more extensive description.

Examples

- *o2D0Analysis*
- *o2SimpleTracksAnalysis*

Double loop with RDataFrame

Get an RDataFrame iterating on the possible **pairs** of candidates with the **same evtID**

```
auto pairs = o2::analysis::doSelfCombinationsWith(input, "d0", "evtID");
```

Select Good candidates

```
auto filtered = pairs.Filter("(d0_cand_type & 1) && (d0bar_cand_type & 1)");
```

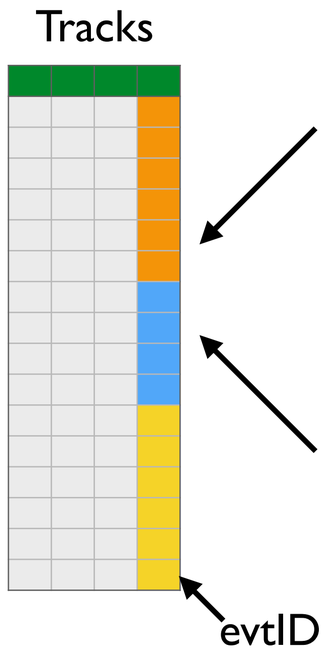
Define extra variables

```
auto deltas = filtered.Define("Dphi", "d0_phi_cand - d0bar_phi_cand")
                      .Define("Deta", "d0_eta_cand - d0bar_eta_cand");
```

Create your histograms

```
auto h2 = deltas.Histo1D("Dphi");
auto h3 = deltas.Histo1D("Deta");
```

RCombinedDS: helper functions



RDataSource

RDataSource

RCombinedDS ← RDataFrame

Iterate on all the track pairs within an event

The user does not see the internals. All the gymnastic is hidden inside a framework provides helper function

```
auto pairs = o2::analysis::doSelfCombinationsWith(input, "d0", "evtID");
```

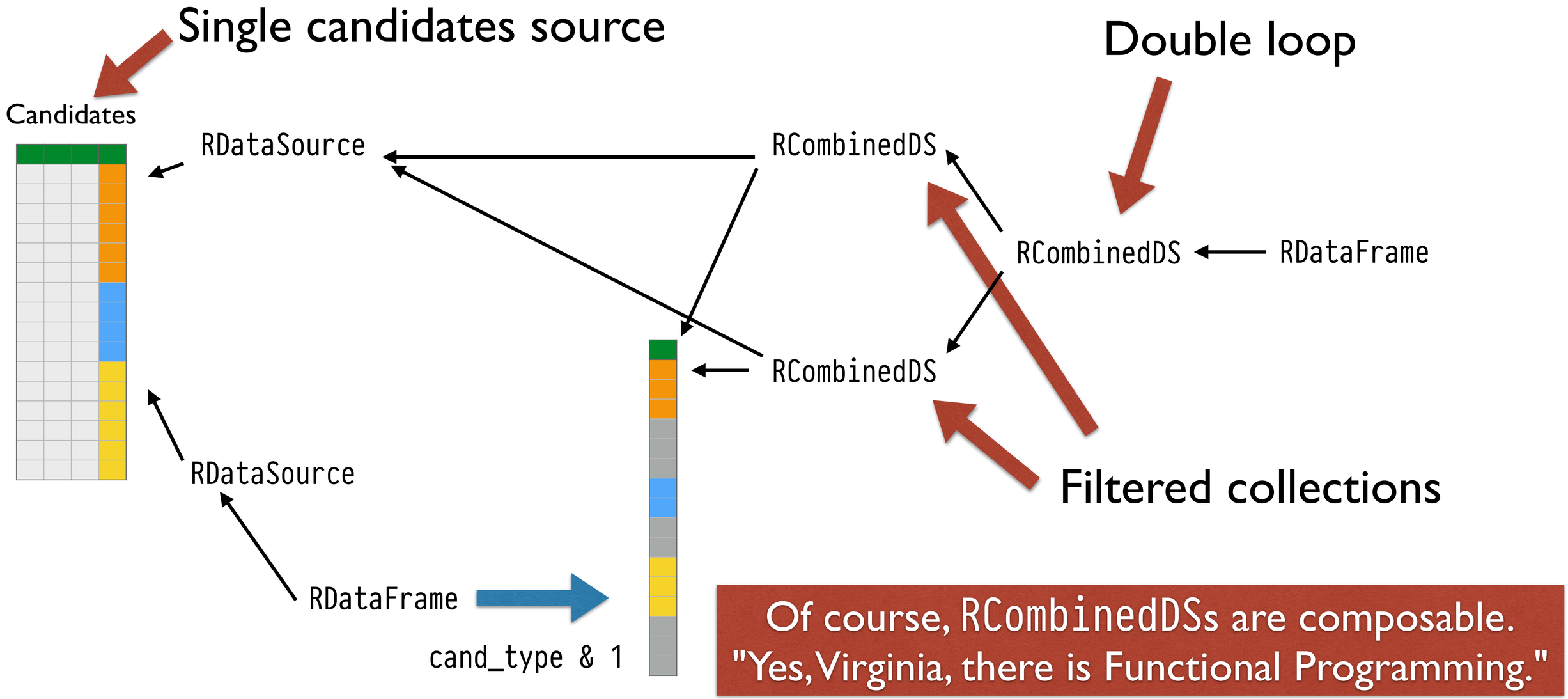
Properly setup RDataFrame

Input subscribed from DPL

mnemonic for the table

Column to use for category

RCombinedDS: putting everything together



Of course, RCombinedDSs are composable.
"Yes, Virginia, there is Functional Programming."

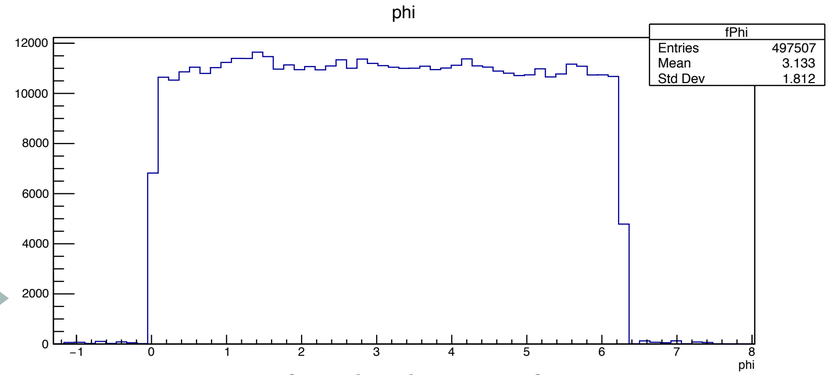
RUN2 ESD TO RUN3 AOD-LIKE CONVERSION

***Standalone converter** from **Run2 ESD to Run3 AOD**:*

https://github.com/AliceO2Group/Run2ESDConverter

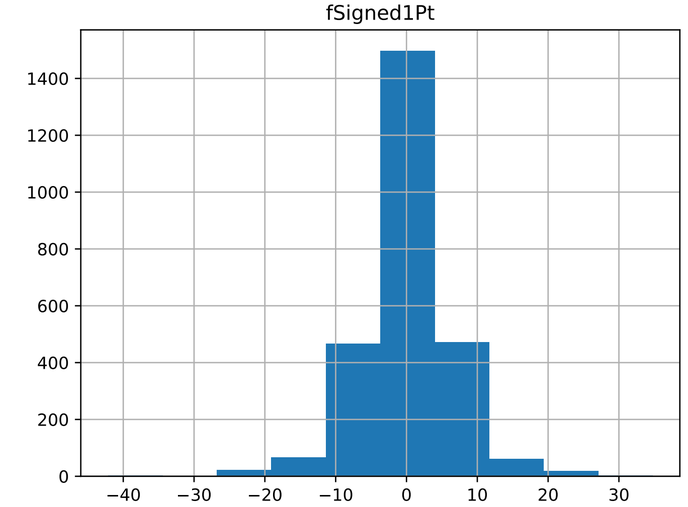
***DPL component** available, can be used with, e.g., `RDataFrame`.*

*Thanks to **Apache Arrow** usage, it's compatible with a number of Python / Java packages, e.g. Pandas.*



Courtesy of Naghmeh, inputs from Run2 ESD file, read with converter + DPL, processing with RDataFrame

```
import pyarrow as pa
import matplotlib.pyplot as plt
import sys

reader = pa.ipc.open_stream(pa.PythonFile(sys.stdin))
t = reader.read_next_batch()
df = t.to_pandas(zero_copy_only=True)
h1 = df.hist(column='fSigned1Pt')
plt.savefig('figure.pdf')
```



Single ESD event, 2607 tracks, 10 bins, not for publication. ;-)

DEPLOYMENT FOR SYNCHRONOUS RECONSTRUCTION

AliECS template topology from DPL workflow

Initial iteration successfully completed: AliECS can deploy DPL data processors. Waiting for further WP8 requests to adjust template generation as needed.

Configuration of DPL data processors

Scalar options (int, float, strings, ...) supported since DPL day 0. New request to support nested collections being discussed with WP8. Initial implementation available.

Monitoring / InfoLogger integration

Integration in DPL since a while. Now tuning performance / detail level to support requested message rate.

Support to connect to Readout / STFB

Adaptor to connect to Readout directly now available, demonstrates usage of ***DPL + FairMQ unmanaged shared region****. Work ongoing with WP5 to ensure DPL can consume sub-timeframes. Strategy on how the processing on FLP will look like being defined together with interested parties (WP1 / WP5 / WP6 / WP11).*

TRACING TOOLS

Motivation

Scaling DPL for FLP processing rates requires to drop a bunch of the per message metrics, due to performance reasons. Still, we do want to have a way to introspect per-message quantities when debugging.

DTrace(-like) to the rescue

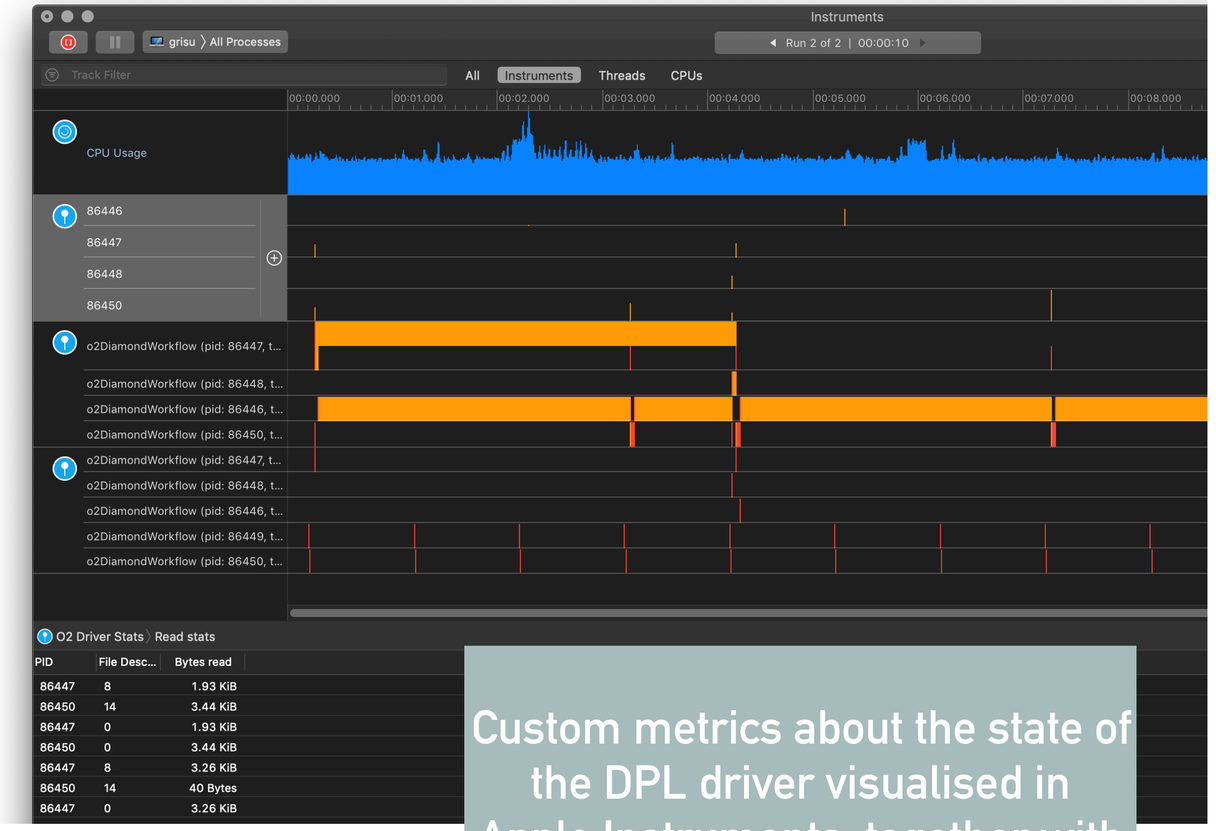
Modern Operating Systems have support for low-footprint instrumentation mechanism, mostly inspired by the venerable DTrace, from (RIP) Solaris.

- **SystemTap** *on Linux*
- **kdebug_signpost** *on macOS*

Support for those is now available in `Framework/Signpost.h` *via the* `O2_SIGNPOST` *API.*

Custom Apple Instruments

For those of you who are using Apple Instruments profiling tool (sorry, I've no idea what is the Linux equivalent), you can take advantage of the custom Alice O2 plugin, ***O2Dissociator****, which has initial support for some of the DPL per message metrics.*



Custom metrics about the state of the DPL driver visualised in Apple Instruments, together with standard CPU profiler.

SUPPORT FOR EVENT DISPLAYS

Event display callback

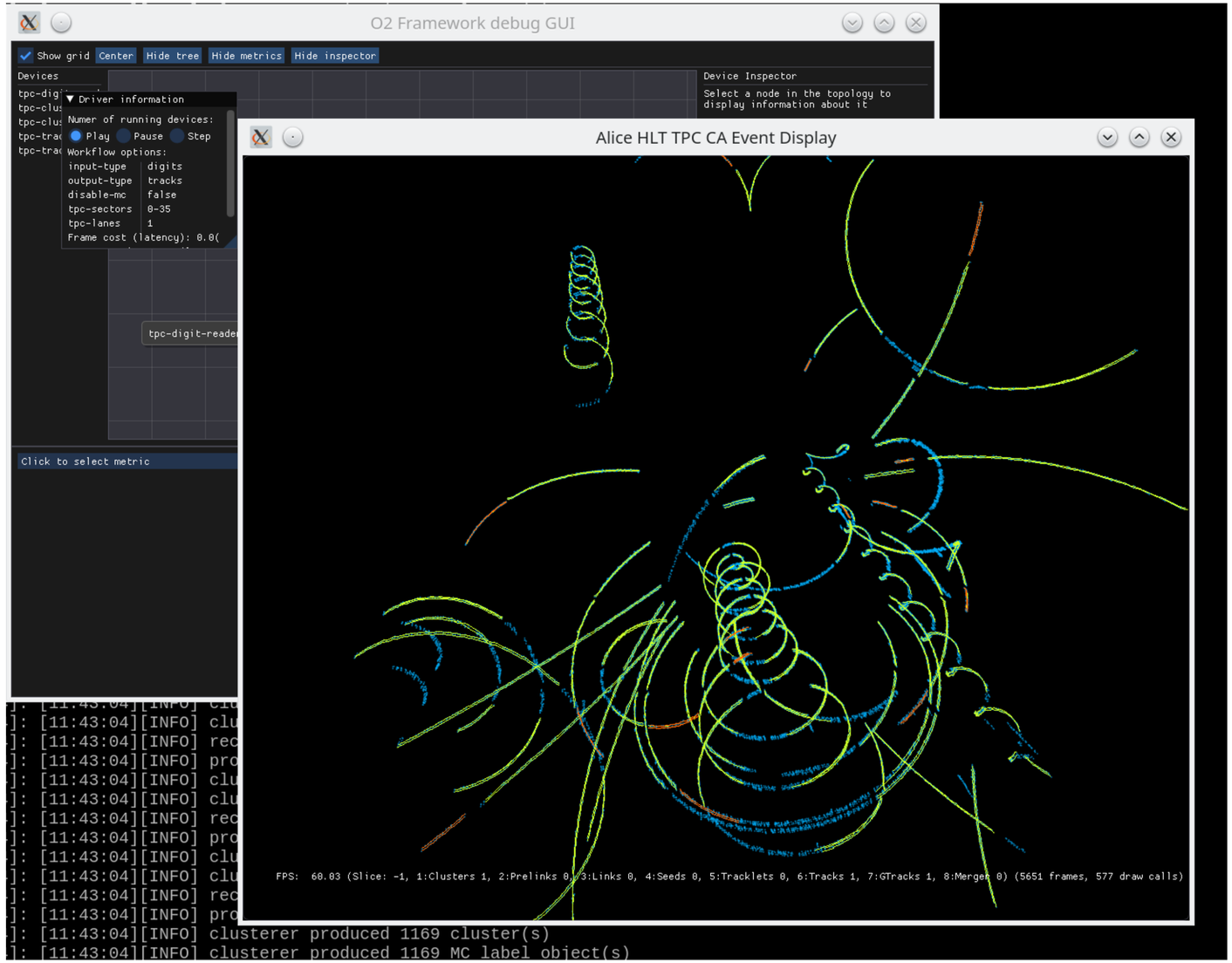
*DPL now support injections of generic "Event Display" callbacks. This is **neither** a 3D engine, **nor** a full featured event display, but it allows integration of one as part of a DPL workflow.*

Two modes

- "Raw" OpenGL
- Sokol 3D Engine*

Example usage

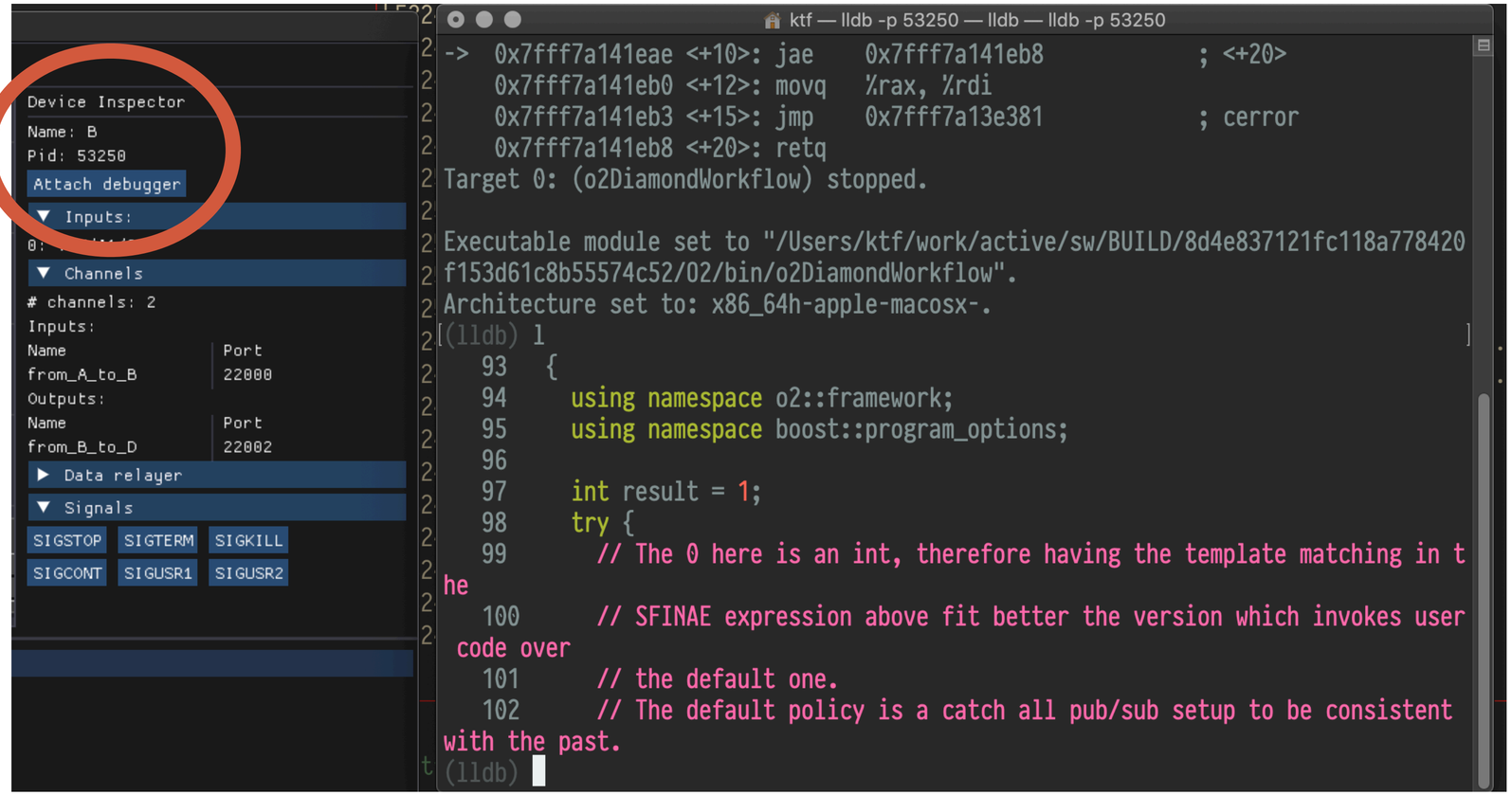
Usage showcased by David Rohr with his TPC Event Display (Linux only).



* https://github.com/floooh/sokol/

MISC UPDATES

- Support for **`Lifetime::Timer`** / **`Lifetime::Enumeration`** now available for InputSpec.
- Improved API to retrieve inputs, outputs, and services.
- Debug GUI now supports attaching a debugger to running devices.

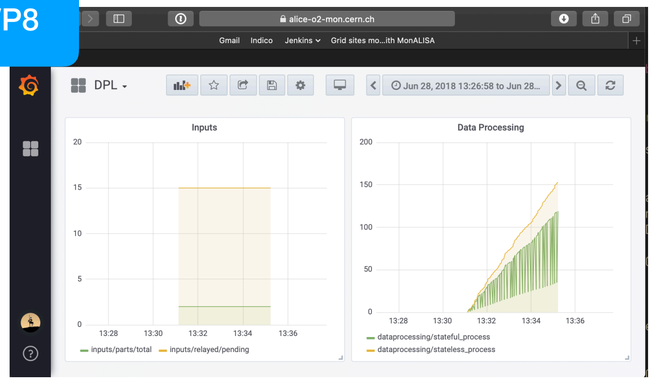


MORE TO COME

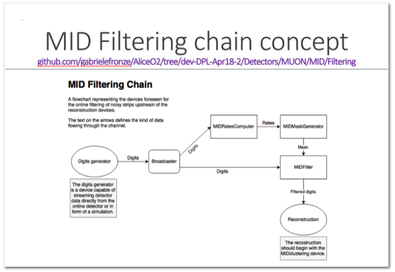
- *Shared memory as default transport for laptops / single node*
- *Proper integration of polymorphic allocators*
- *Ability to merge workflows*
- *Possibility to test algorithms in isolation*
- *Possibility to coalesce multiple (pipelined) data processors on one device*
- *Support for vector of inputs*
- *Non-`RDataFrame` based iteration over AOD collections*

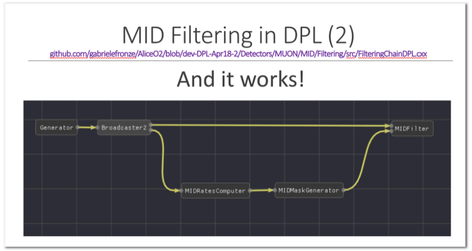
DPL AS AN INTEGRATION PLATFORM FOR O2

WP8

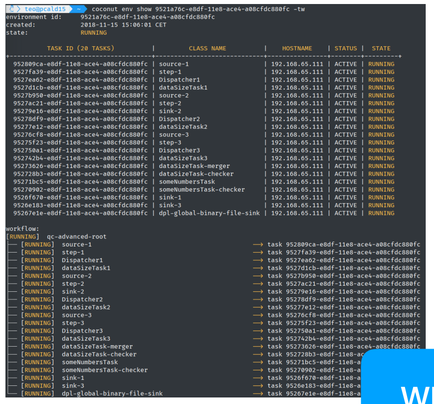


O2 Monitoring and InfoLogger integration



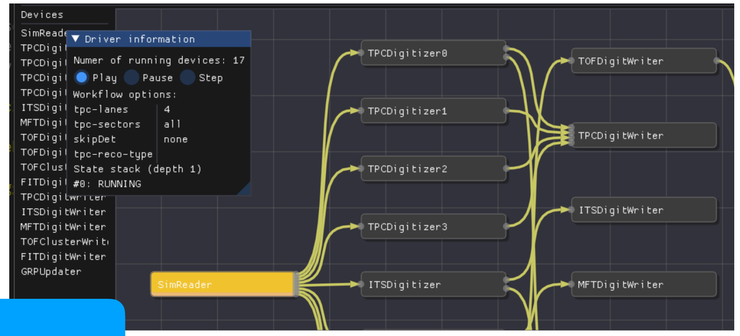


MID Filtering Chain



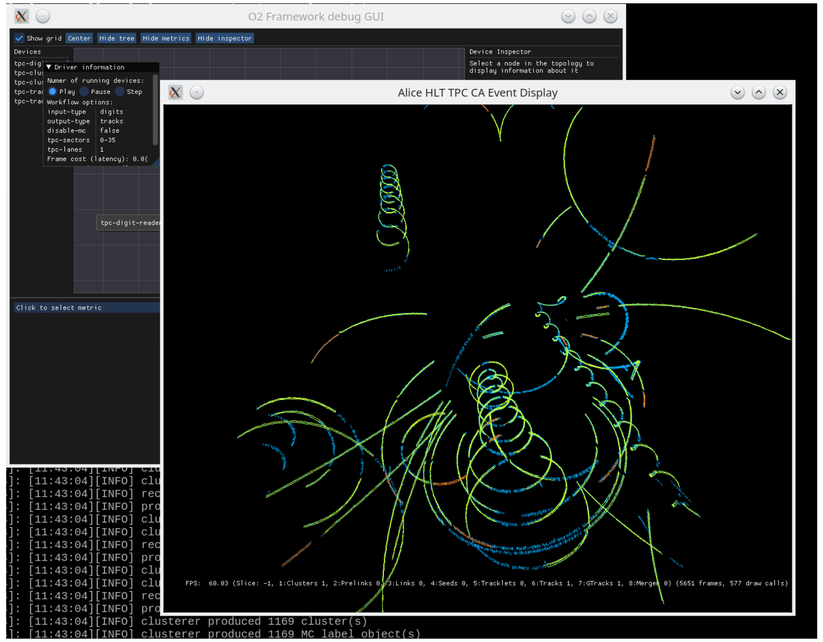
O2 AliECS Integration

WP8

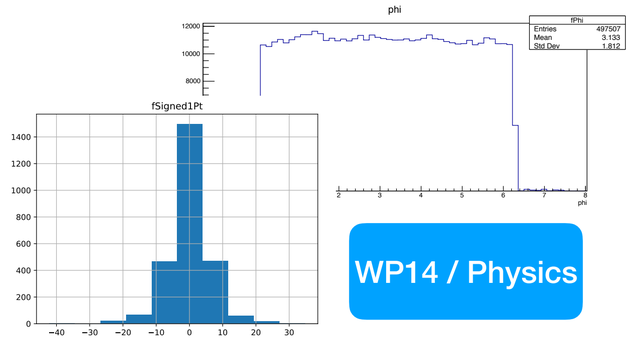


WP12

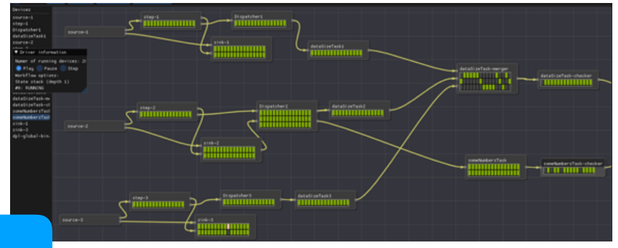
Digitization in DPL



TPC Event Displays



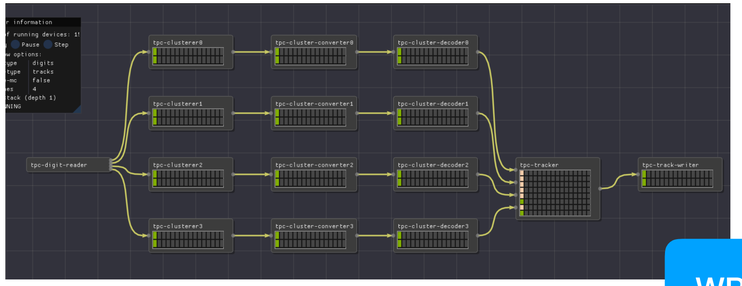
WP14 / Physics



WP7

DataSampling using DPL

Kudos to everyone using DPL



TPC reconstruction prototype

WP13

BACKUP

IMPROVED API TO RETRIEVE INPUTS

From:

```
AlgorithmSpec{
[](ProcessingContext& context) {
    auto &monitoring = context.get<MonitoringService>();
    auto someInput = context.inputs().get<int>("SomeInputRef");
}
}
```

to:

```
AlgorithmSpec{
    adaptStateless(
        [](Monitoring& monitoring, InputRecord& inputs) {
            auto someInput = inputs.get<int>("SomeInputRef");
      })
}
```

Implementation of the toy analysis with RDataFrames in the O2 software package

Full source at: https://github.com/AliceO2Group/AliceO2/blob/dev/Framework/TestWorkflows/src/o2D0Analysis.cxx

Single loop with RDataFrame

Get an RDataFrame iterating on candidates obtained via O2

```
auto candidates = o2::analysis::doSingleLoopOn(input);
```

Select Good candidates

```
auto filtered = candidates.Filter("cand_type & 1");
```

Fill an histogram

```
auto h1 = filtered.Histo1D("inv_mass");
```

Draw it

```
h1->Draw();
```


31

Single loop with RDataFrame

Get an RDataFrame iterating on candidates obtained via O2

```
auto candidates = o2::analysis::doSingleLoopOn(input);
```

Select Good candidates

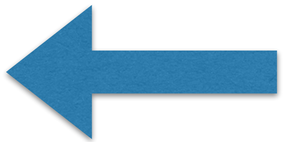
```
auto filtered = candidates.Filter("cand_type & 1");
```

Fill an histogram

```
auto h1 = filtered.Histo1D("inv_mass");
```

Draw it

```
h1->Draw();
```



Event loop actually runs here.

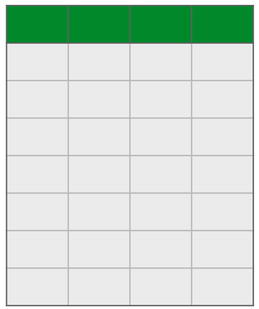
32

RDataFrame internals

Iterate on all the items 1...N

```
table.GetRow(i)
```

```
for i in 1..N:
  datasource.SetEntry(i)
```



RDataSource ← RDataFrame ←

The user does not see the internals.
All the gymnastic is hidden inside a framework provided helper function

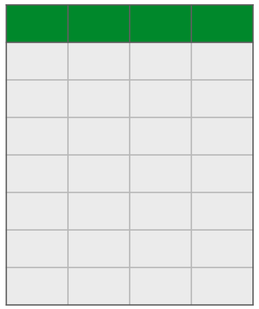
```
auto candidates = o2::analysis::doSingleLoopOn(input);
```

Properly setup RDataFrame

Input subscribed from DPL

33

RDataFrame internals



Actual table stored in memory / disk / created on the fly. The actual backend will depend on the context.

`RDataSource`

RDataSource abstracts from the user the act of fetching row from a table. Different implementations depending on the backend.

`RDataFrame`

RDataFrame abstracts from the user how to iterate on a set of rows and how actions are performed on them.

Role of the analysis framework: provide helpers to construct useful views on the data, using the above building blocks.

Double loop with RDataFrame

Get an RDataFrame iterating on the possible **pairs** of candidates with the **same evtID**

```
auto pairs = o2::analysis::doSelfCombinationsWith(input, "d0", "evtID");
```

Select Good candidates

```
auto filtered = pairs.Filter("(d0_cand_type & 1) && (d0bar_cand_type & 1)");
```

Define extra variables

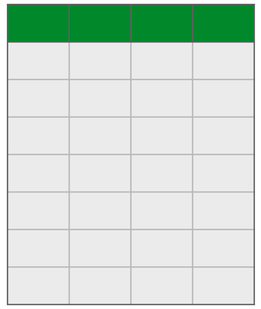
```
auto deltas = filtered.Define("Dphi", "d0_phi_cand - d0bar_phi_cand")
                      .Define("Deta", "d0_eta_cand - d0bar_eta_cand");
```

Create your histograms

```
auto h2 = deltas.Histo1D("Dphi");
auto h3 = deltas.Histo1D("Deta");
```

RCombinedDS: combining multiple datasources

Table A



N

RDataSource

GetRow(i=p/M)

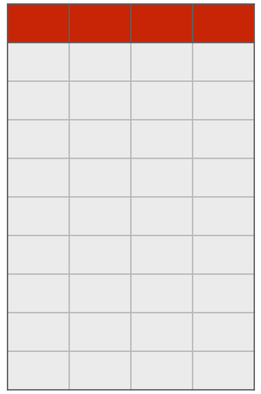
RCombinedDS

RDataFrame

SetEntry(p)

GetRow(j=p%M)

Table B



M

RDataSource

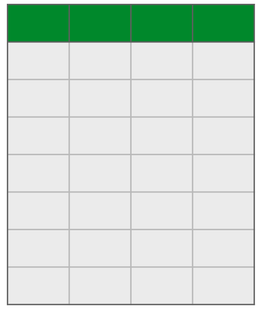
Iterate on all the pairs p=1...NxM

This is effectively doing a double loop on all the possible row pairs of the table A and B.

36

RCombinedDS: generalisation

Table A



N

RDataSource

```
if Index[p] != 0:
  GetRow(getI(Index[p]))
```

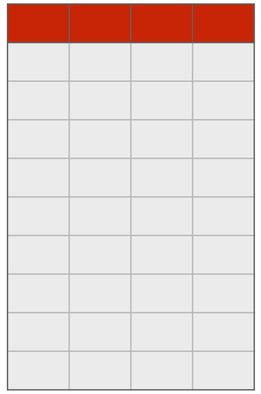
Iterate on all the pairs p=1...NxM

RCombinedDS

RDataFrame

SetEntry(p)

Table B



M

RDataSource

```
if Index[p] != 0:
  GetRow(getJ(Index[p]))
```

We can generalise the mechanism by having an index to select the pairs to yield.

37

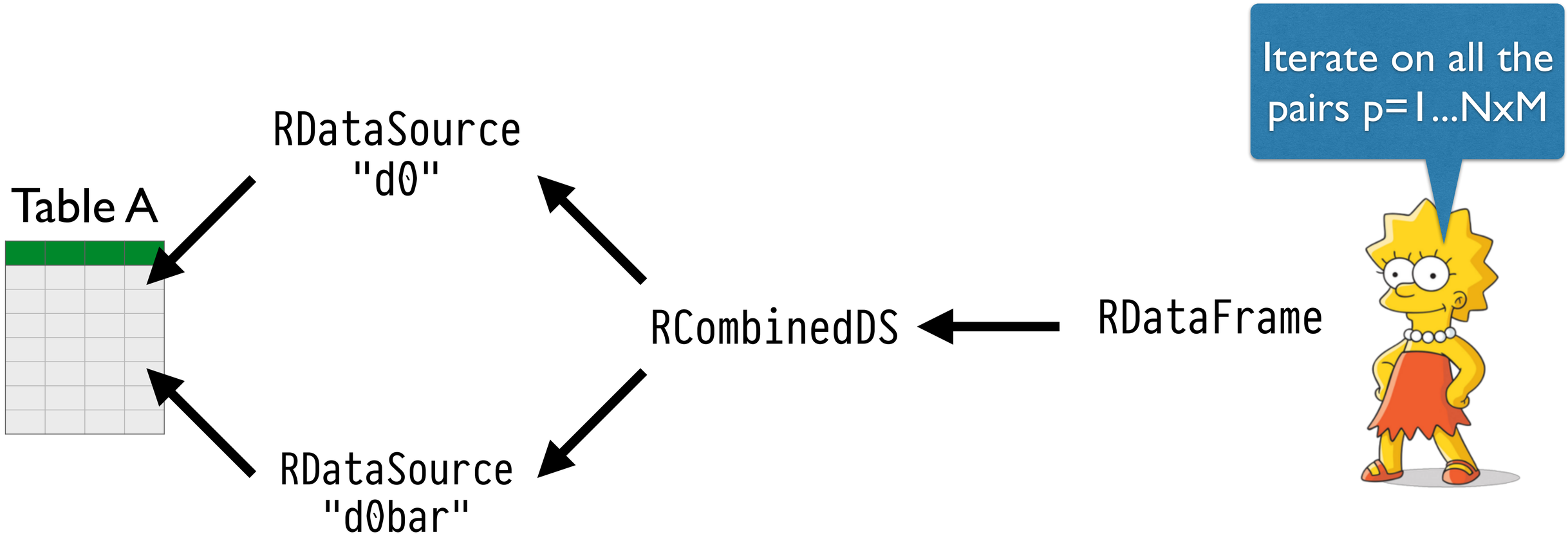
RCombinedDS: full double loop

M

Index = N

1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	…	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	…	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	…	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	…	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	…	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	…	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	…	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	…	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	…	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	…	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	…	1
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	…	1
…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…	…
1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	…	1

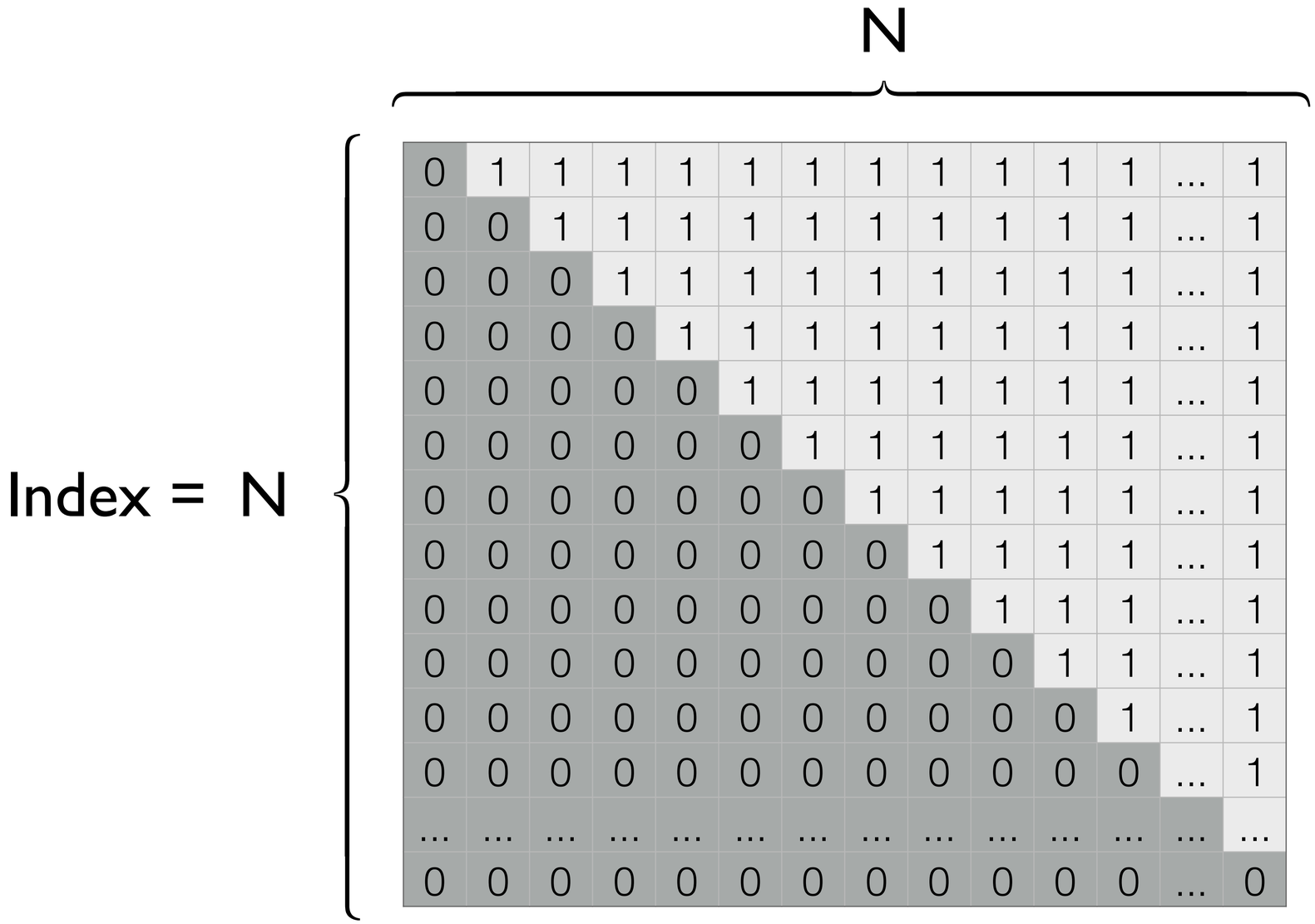
RCombinedDS: same source



Two RDataSource can actually point to the same table.

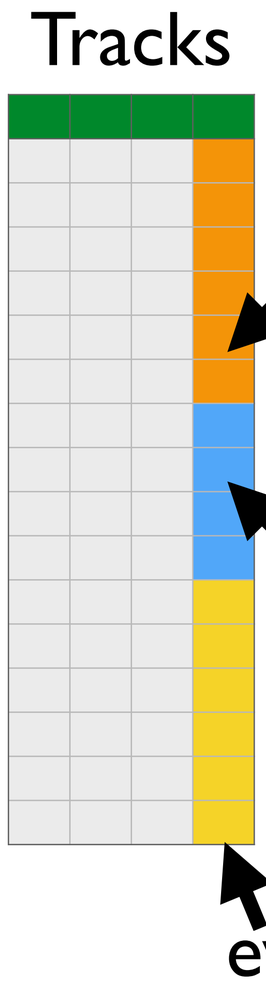
39

RCombinedDS: double loop without repetitions



A strictly upper triangular square matrix index can represent all the possible pairs of rows within the same table, avoiding repetitions.

RCombinedDS: generalisation

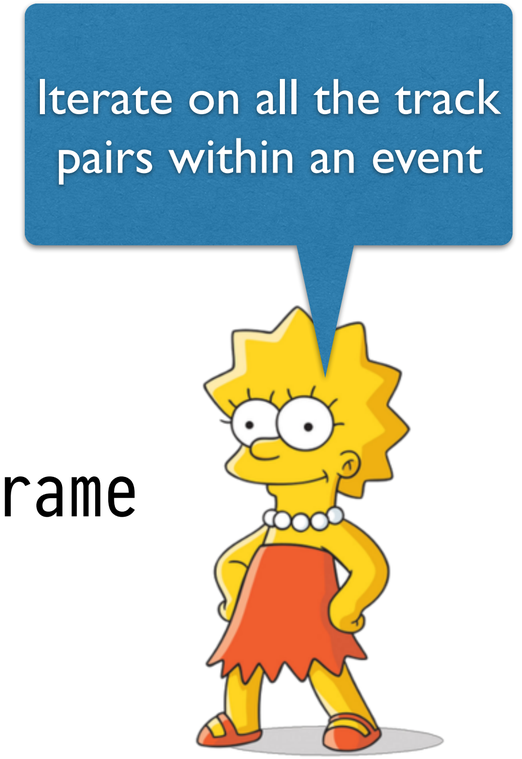


RDataSource

RDataSource

RCombinedDS

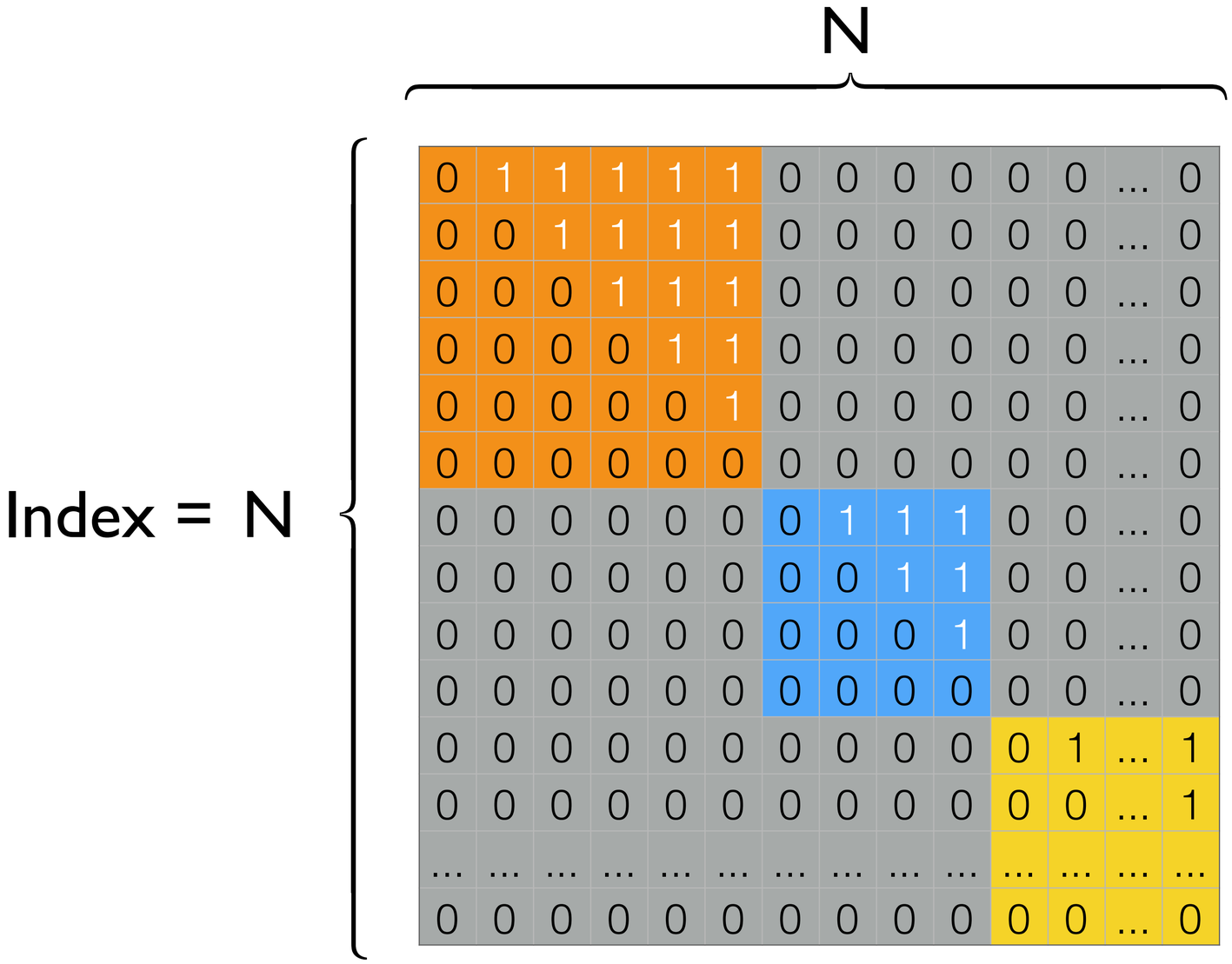
RDataFrame



A column can define categories of rows, e.g. the event id.

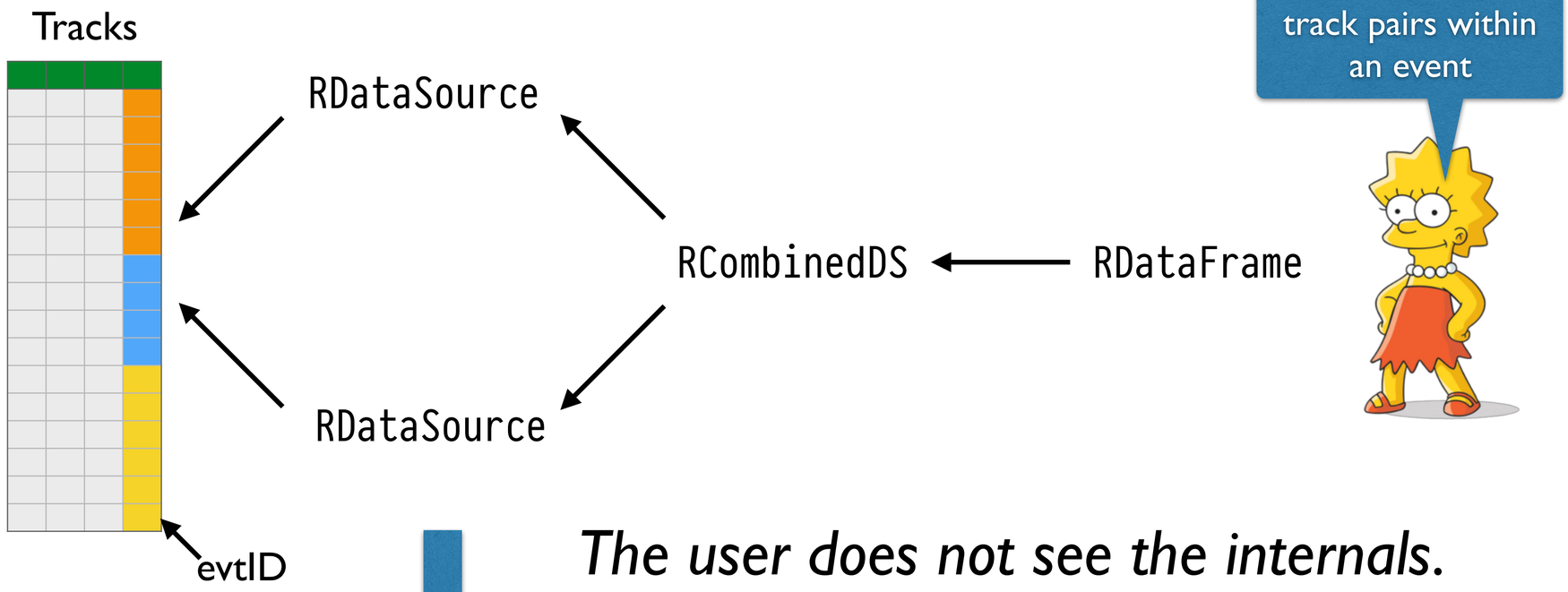
41

RCombinedDS: double with loop within a category



An index (similar to) a block diagonal matrix represents combinations within the same category. Most natural category is the event, but RCombinedDS is not limited to that (e.g. for Event Mixing)

RCombinedDS: helper functions



The user does not see the internals. All the gymnastic is hidden inside a framework provides helper function

```
auto pairs = o2::analysis::doSelfCombinationsWith(input, "d0", "evtID");
```

Properly setup RDataFrame

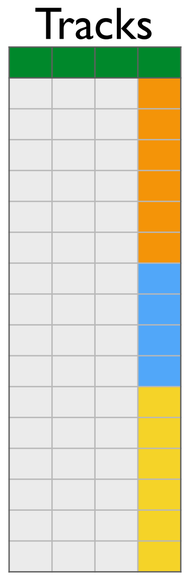
Input subscribed from DPL

mnemonic for the table

Column to use for category

43

RCombinedDS: filtered collections



RDataSource

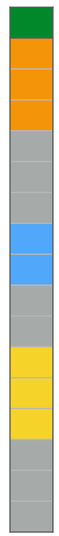
RDataSource

RCombinedDS

RDataFrame

RDataFrame

pt > 10

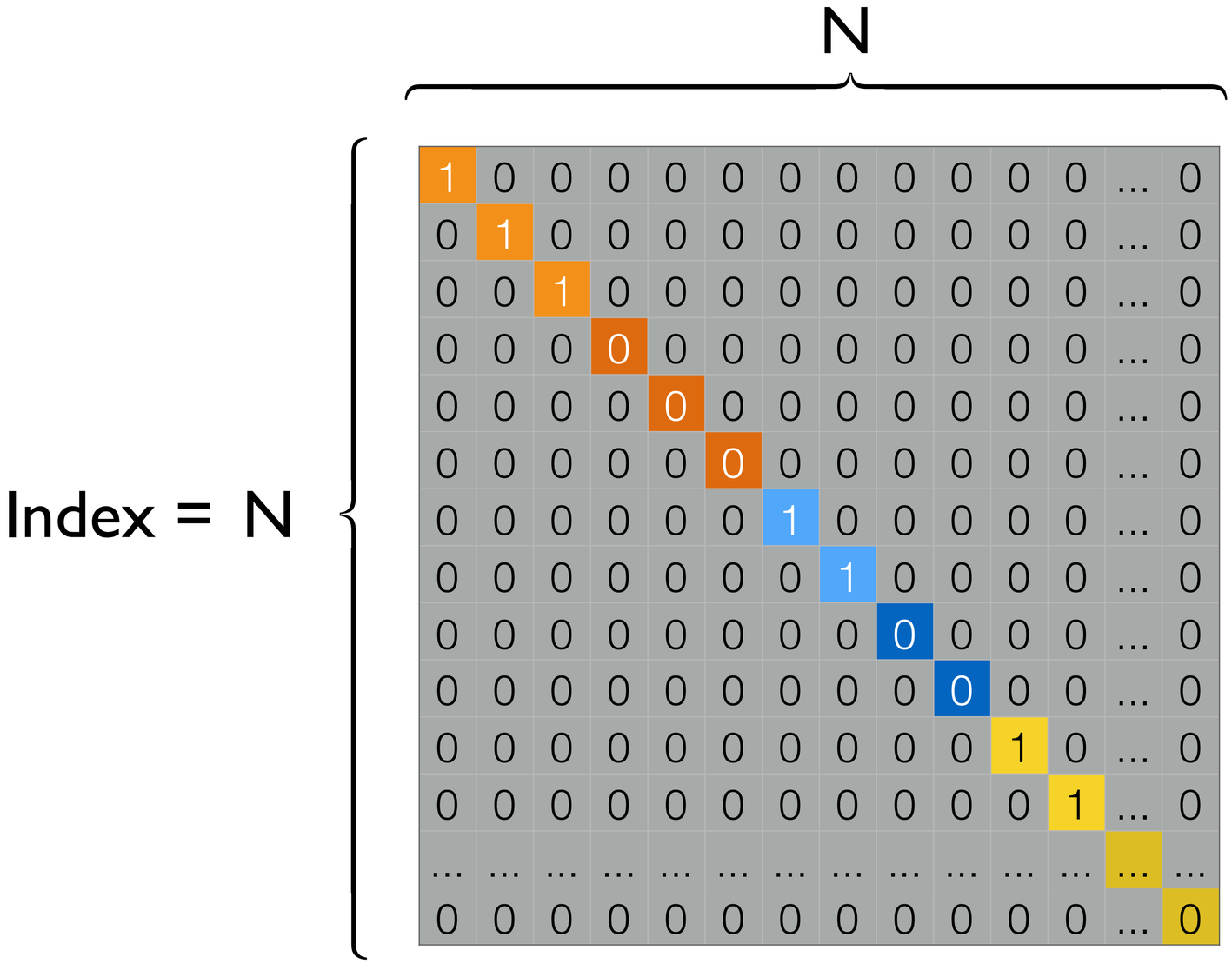


Iterate on all the track pairs within an event with pt > 10

RDataFrame can define new columns, including those defining filtered results.

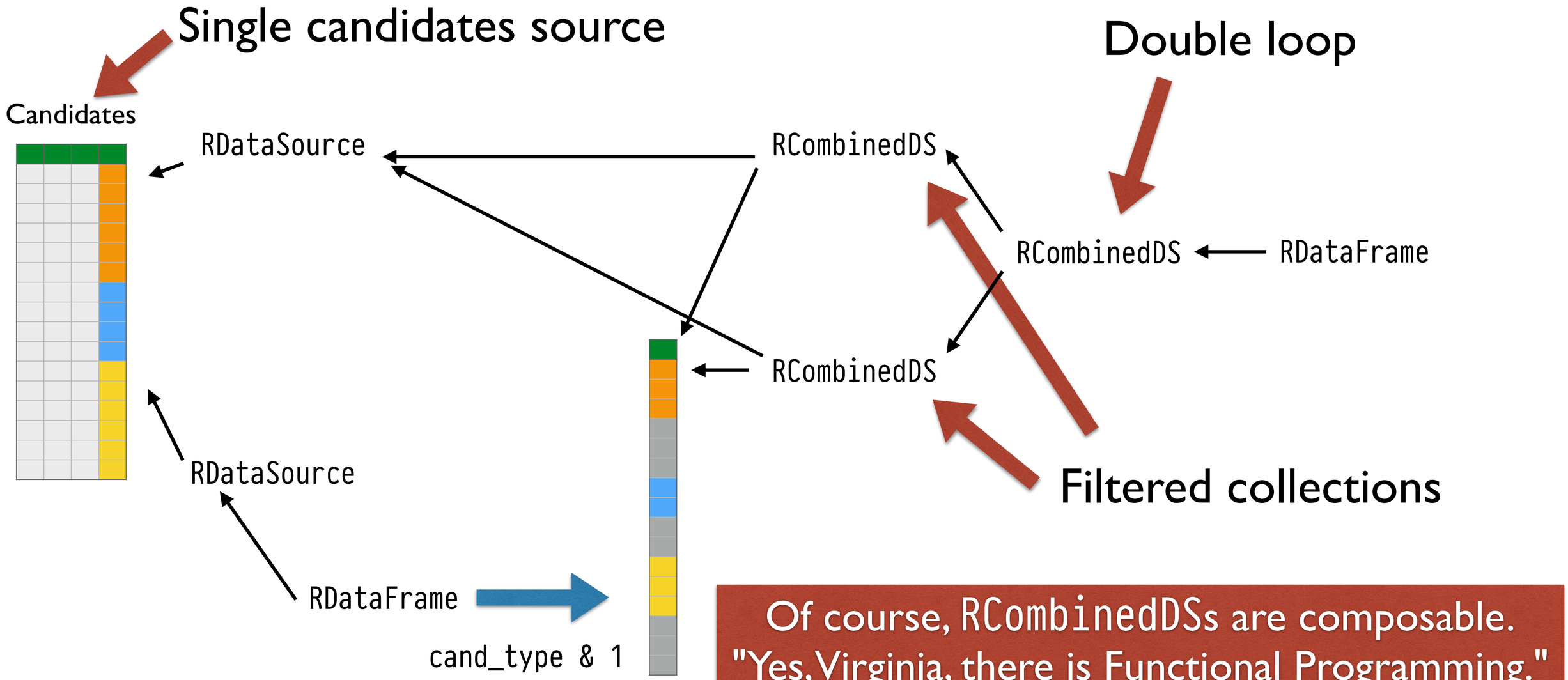
44

RCombinedDS: filtered collections



A diagonal matrix can represent a filtered collection (obviously the actual code does not really use one!)

RCombinedDS: putting everything together



Of course, RCombinedDSs are composable.
"Yes, Virginia, there is Functional Programming."

46