THE O^2 SOFTWARE FRAMEWORK AND GPU USAGE IN ALICE ON AND OFFLINE RECONSTRUCTION IN RUN3*

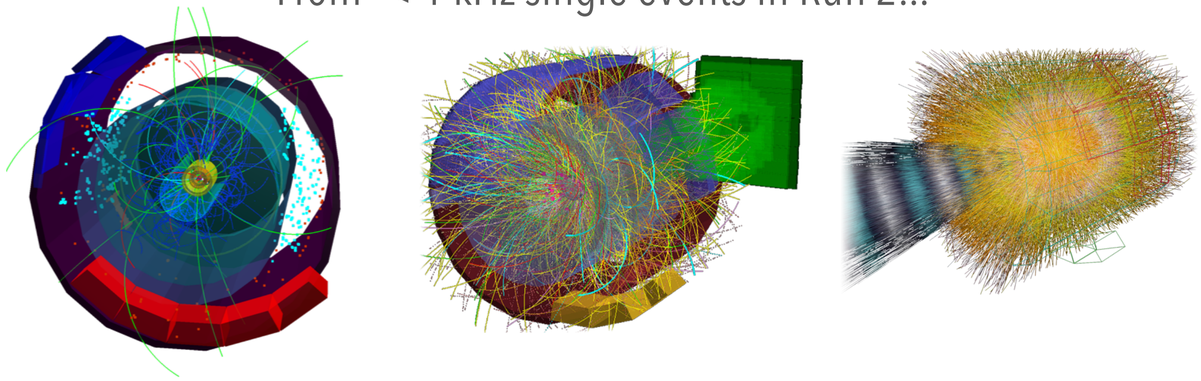
David Rohr, ***Giulio Eulisse*** *for the ALICE Collaboration*

**i.e. Everything you wanted to know about ALICE Software but you were too afraid to ask.*

Challenges for ALICE in Run 3

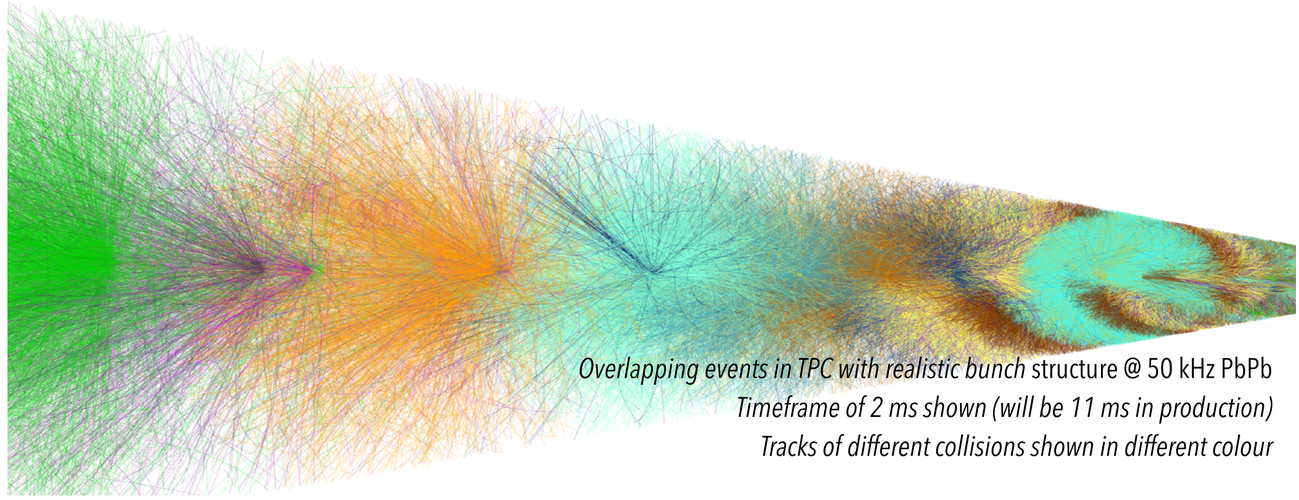
- ***Completely new detector readout*** *and* ***substantial detector upgrades:*** *new ITS, MFT, FIT. New GEM for TPC readout.*
- *Reconstruct TPC data in* ***continuous readout*** *in combination with triggered detectors.*
- ***Reconstruct O(100x) more*** *events online.*
- ***Store O(100x)*** *more events (needs factor 36x for TPC compression). Cannot store all raw data, use* ***GPUs to do compression online****.*
- *WLCG "****flat budget****" scenario (4x more resources over 10 years, for 100x more events).* ***Use online GPU farm offline to speedup processing****.*

From < 1 kHz single events in Run 2...



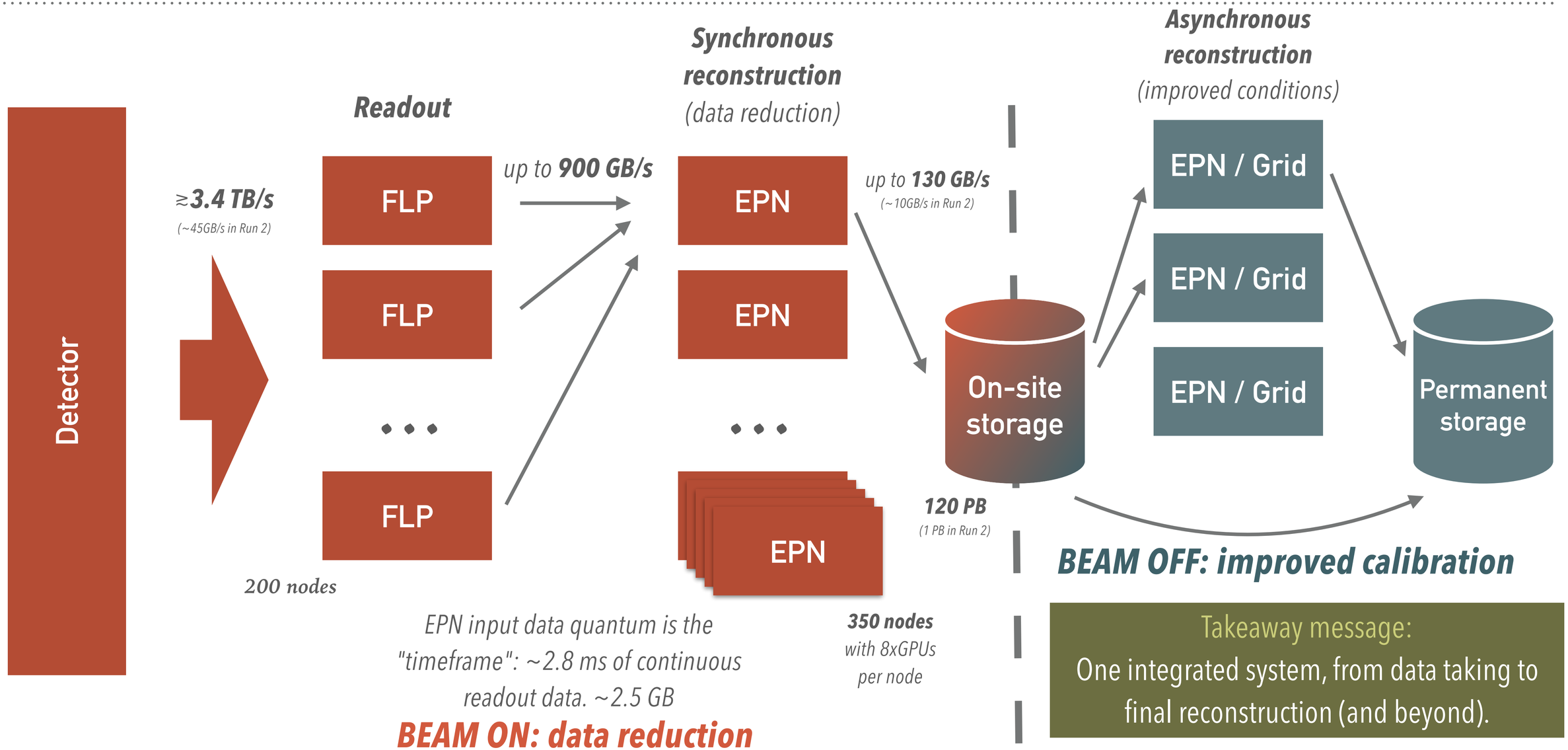
pp *pPb* *PbPb*

...to 50 kHz of continuous readout data in (PbPb) Run 3.



Overlapping events in TPC with realistic bunch structure @ 50 kHz PbPb
Timeframe of 2 ms shown (will be 11 ms in production)
Tracks of different collisions shown in different colour

ALICE IN RUN 3: THE O² PROJECT



Takeaway message:
One integrated system, from data taking to final reconstruction (and beyond).

O^2: Software Framework in One Slide

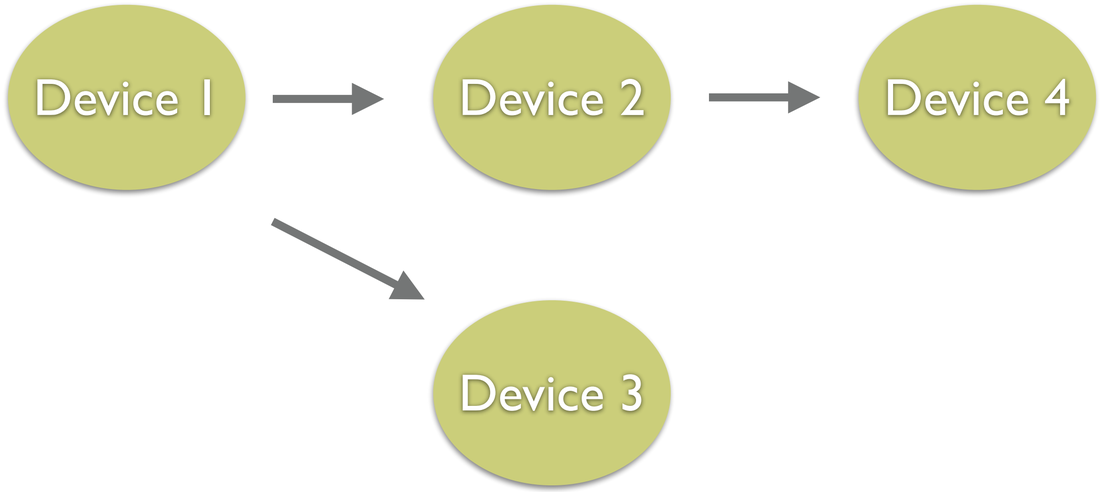
Transport Layer: ALFA / FairMQ

- **Joint collaboration with FAIR and GSI**

ALFA / FairMQ: General Idea

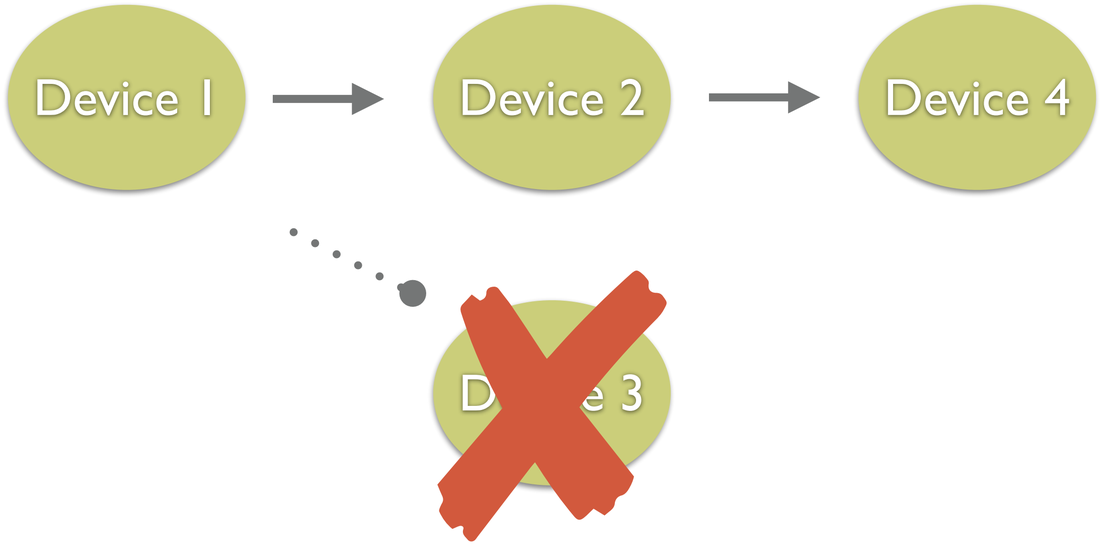
Data processing happens in **separate processes**, called **devices**.

ALFA / FairMQ: General Idea



Multiple devices form a **topology**. Devices exchange **messages** over so called **channels**.

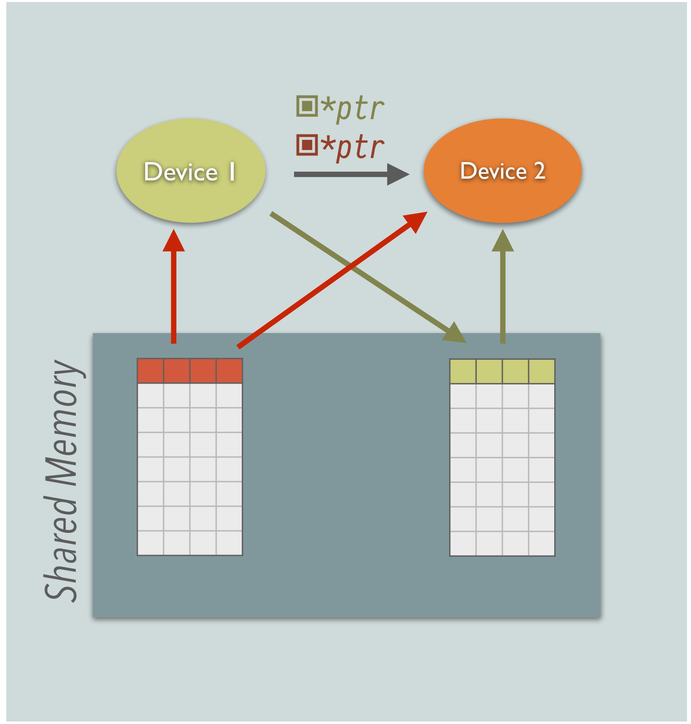
ALFA / FairMQ: General Idea



Certain "**expendable**" devices are allowed to die without killing the processing.

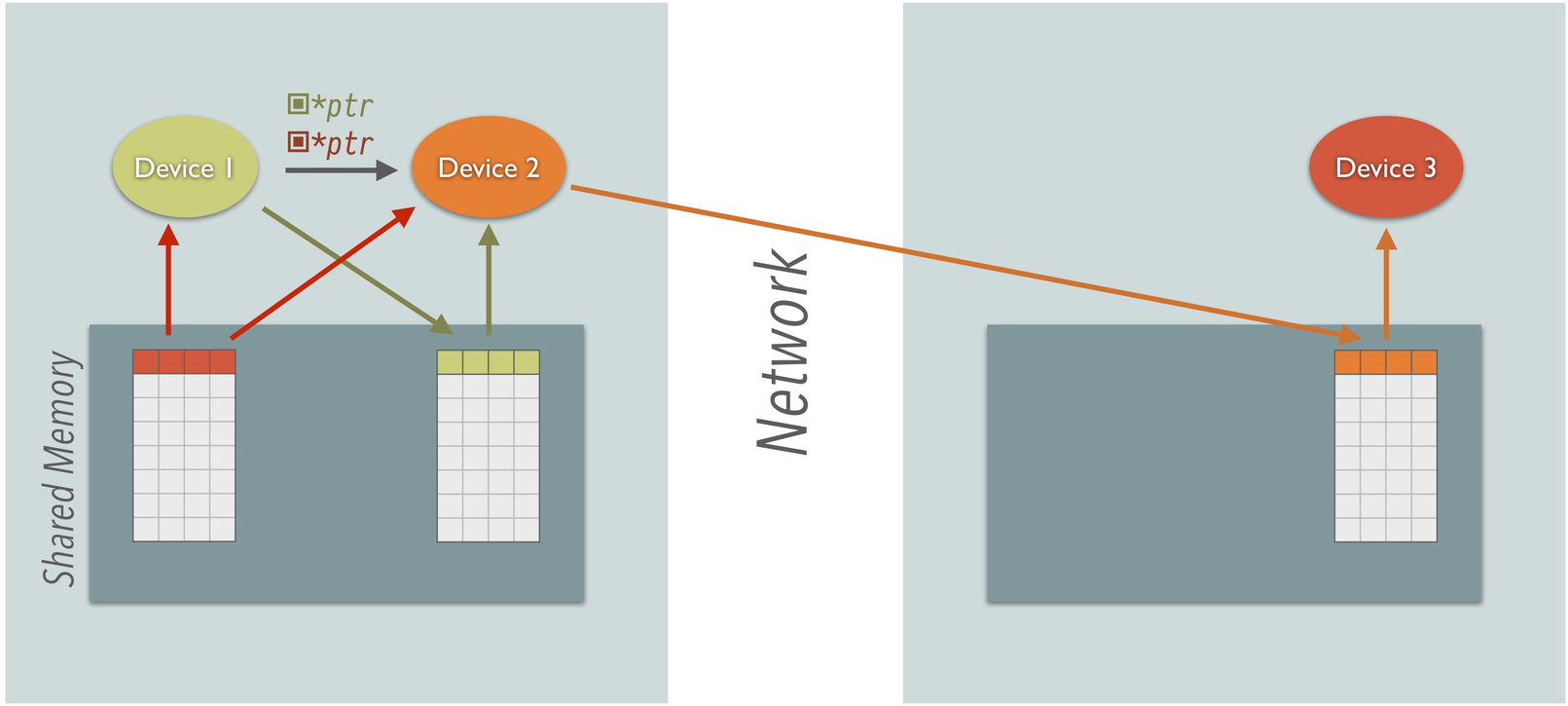
ALFA / FairMQ: General Idea

When running on the same node, message passing is actually optimised via the shared memory backend provided by FairMQ. **Only pointers in shared memory are exchanged.**



ALFA / FairMQ: General Idea

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



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.*
- **ROOT based serialisation.** *Useful for QA and final results.*
- **Apache Arrow based.** *Backend of the analysis data model and for integrating with other tools.*
- *We contributed the* **RDataFrame Arrow backend to ROOT**.

Transport Layer: ALFA / FairMQ[1]

- **Joint collaboration with FAIR and GSI**
- **Standalone processes (devices)** *for deployment flexibility & resilience.*
- **Message passing** *as a parallelism paradigm*
- **Shared memory** *backend for reduced memory usage and improved performance*
- **Seamless remote** communication

O^2: Software Framework in One Slide

Framework &
Data Processing Layer (DPL)

Hides the hiccups of a distributed system, presenting a familiar "Data Flow" system.

- ***Reactive-like design*** *(push data, don't pull)*
- ***Implicit workflow definition*** *via modern C++ API.*
- ***Core common tasks:*** *topological sort of dependencies, deployment of generated topologies, data lifecycle handling, service management, common infrastructure services, plug-in manager.*
- ***Integration*** *with the rest of the production system, e.g. Monitoring, Logging, Control.*

Data Layer: O2 Data Model

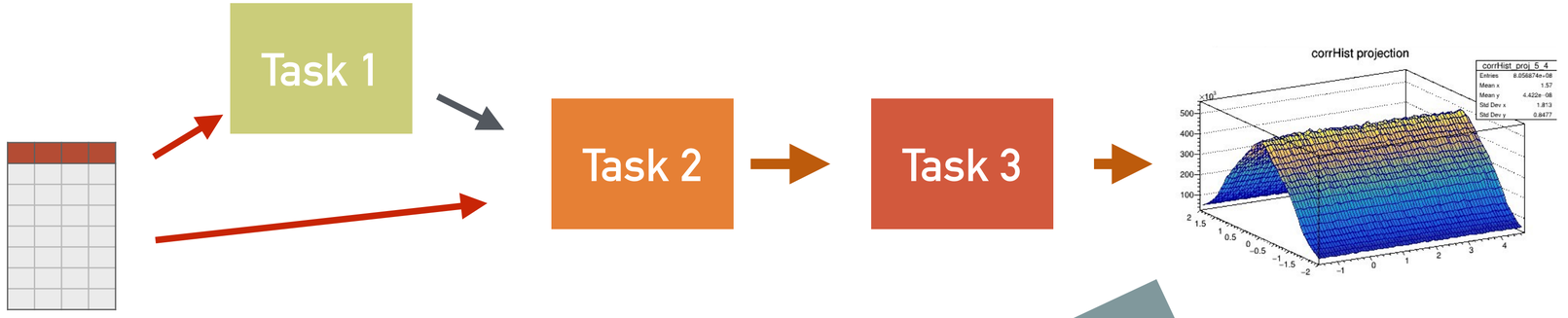
Message passing aware data model. Support for multiple backends:

- **Simplified, zero-copy** *format optimised for performance and direct GPU usage.*
- **ROOT based serialisation.** *Useful for QA and final results.*
- **Apache Arrow based.** *Backend of the analysis data model and for integrating with other tools.*
- *We contributed the* **RDataFrame Arrow backend to ROOT**.

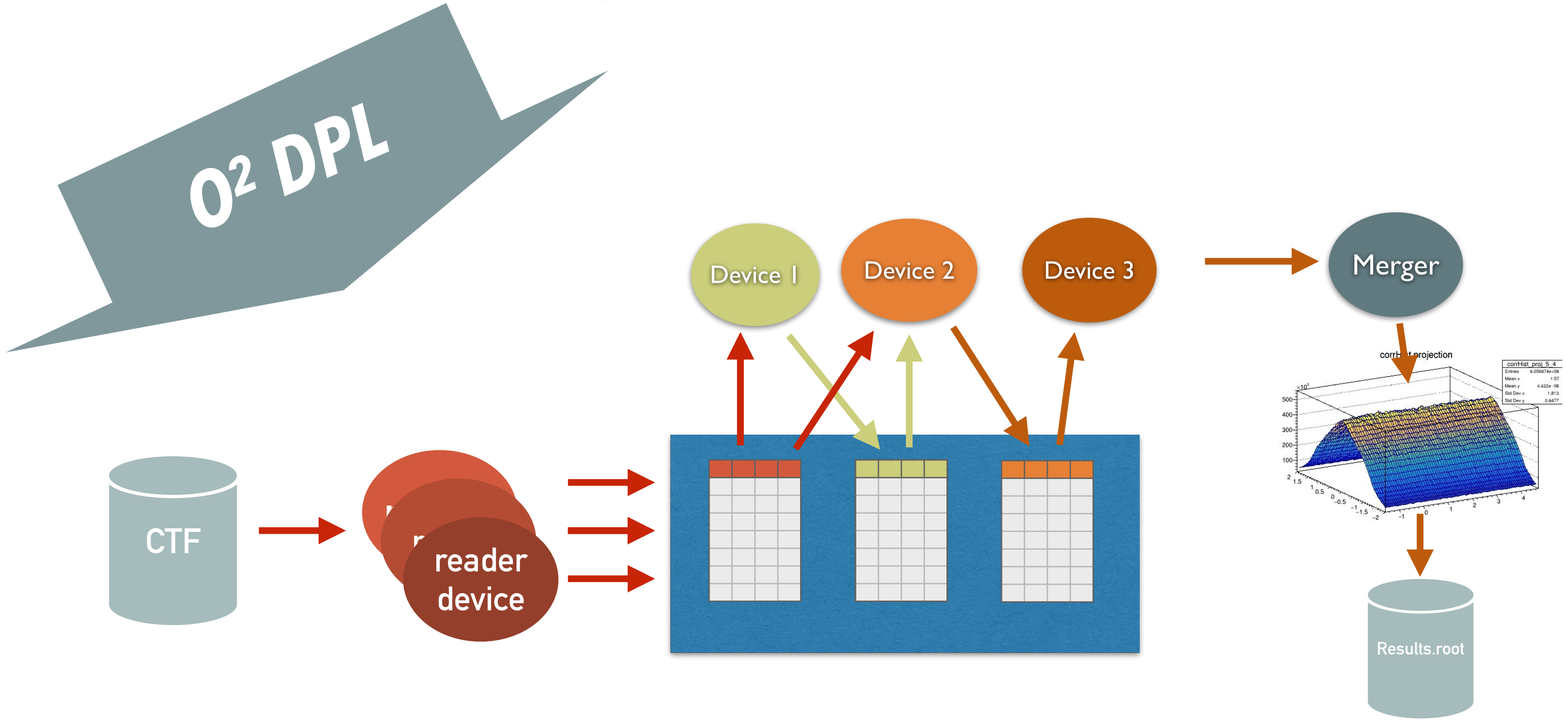
Transport Layer: ALFA / FairMQ[1]

- **Joint collaboration with FAIR and GSI**
- **Standalone processes (devices)** *for deployment flexibility*
- **Message passing** *as a parallelism paradigm*
- **Shared memory** *backend for reduced memory usage and improved performance*
- **Seamless remote** communication

O² Data Processing Layer



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

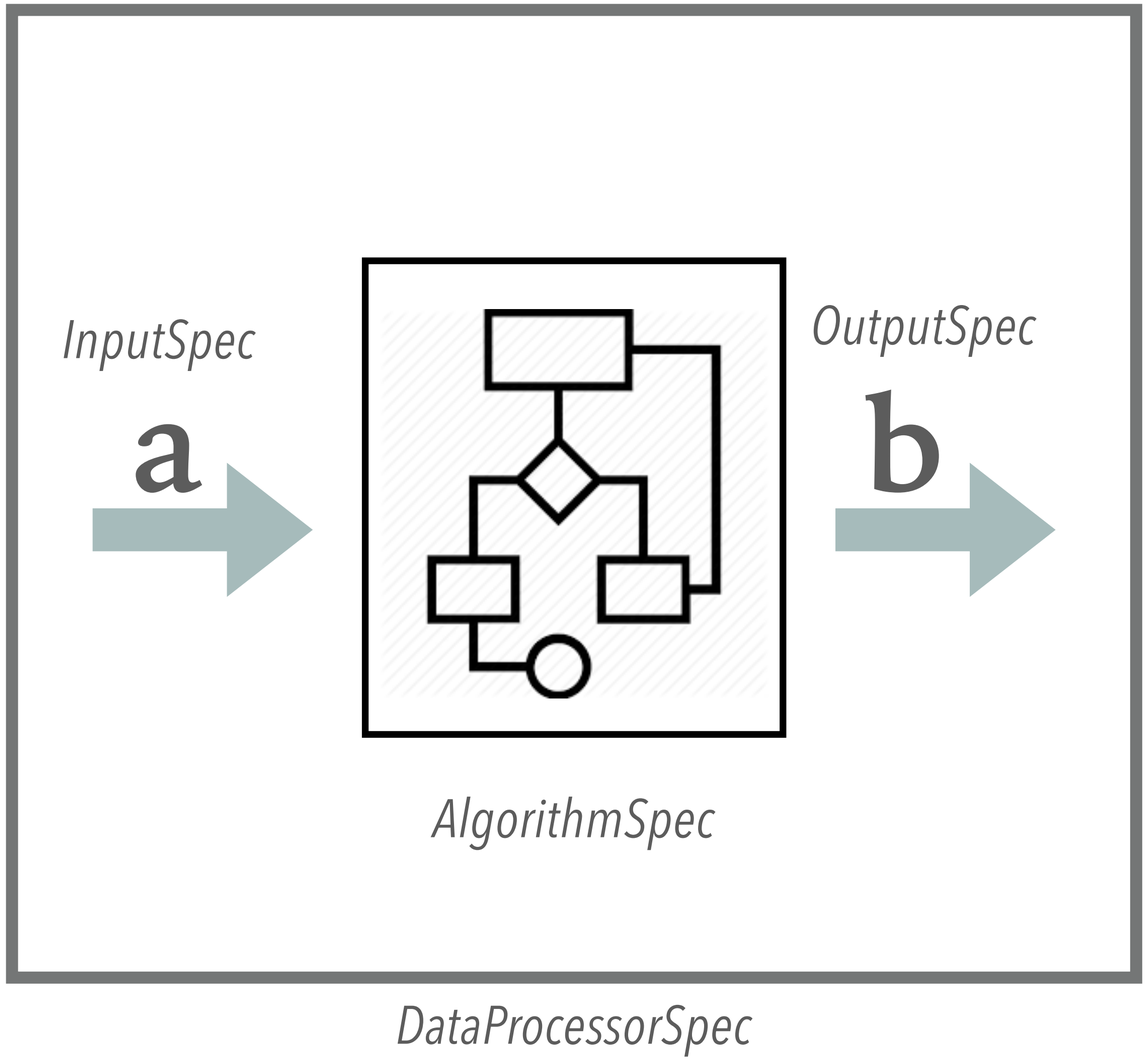


O² Data Processing Layer (DPL) translates the implicit workflow(s) defined by physicists to an actual FairMQ topology of devices, injecting readers and merger devices, completing the topology and taking care of parallelism & rate limiting.

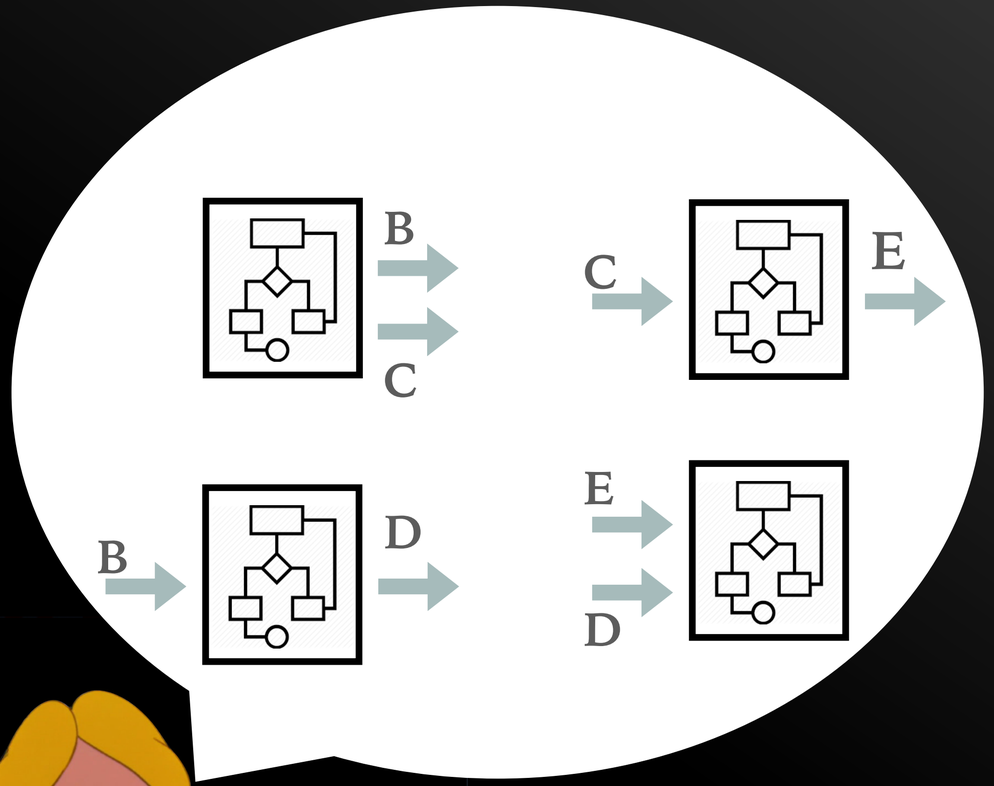
DATA PROCESSING LAYER: BUILDING BLOCK

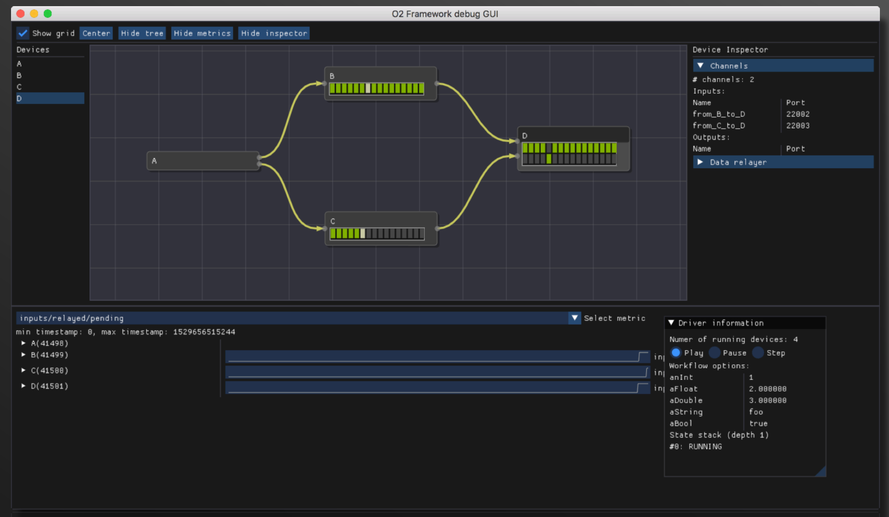
A DataProcessorSpec *defines a pipeline stage as a building block.*

- *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.*



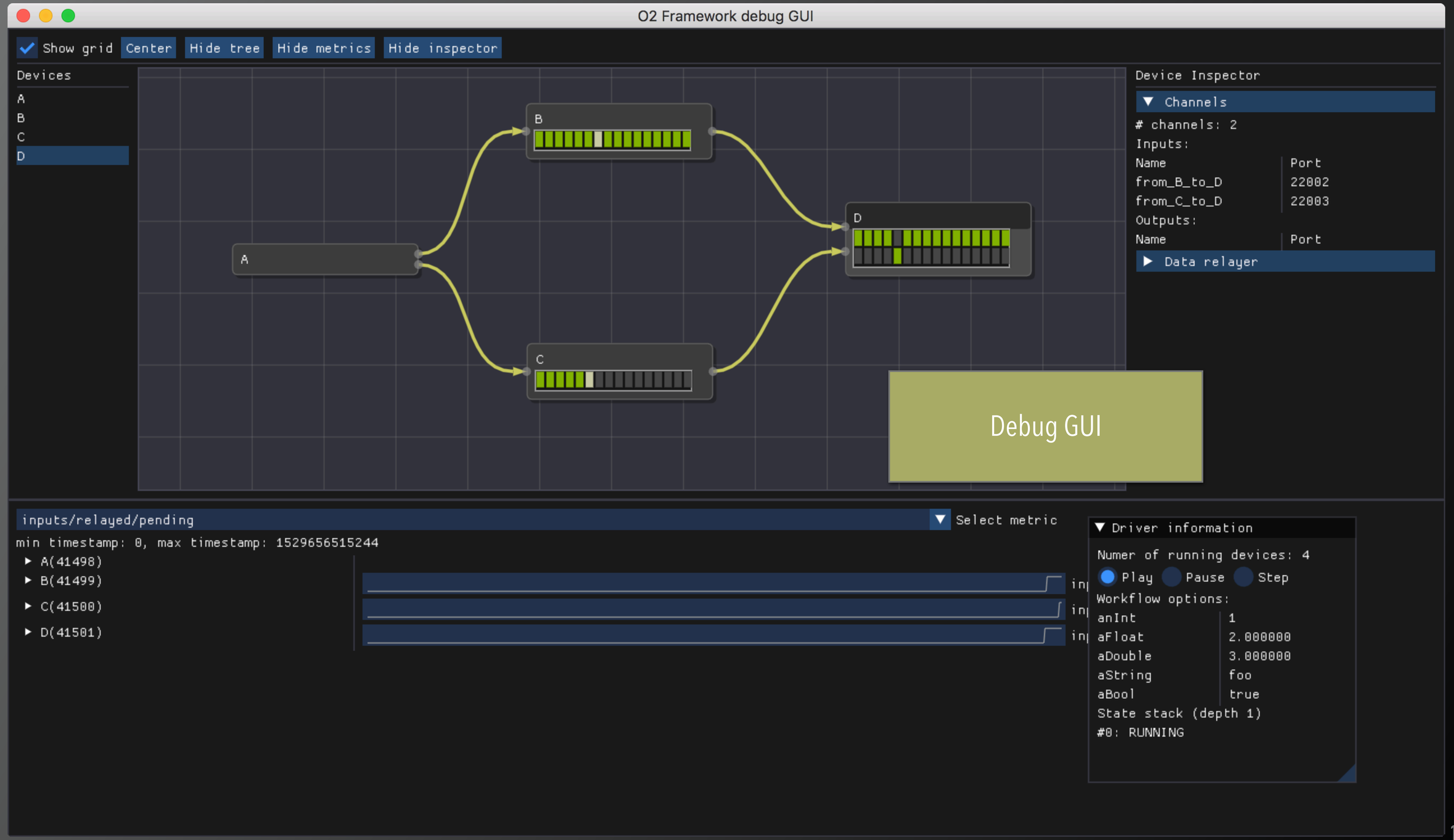
DATA PROCESSING LAYER: IMPLICIT TOPOLOGY

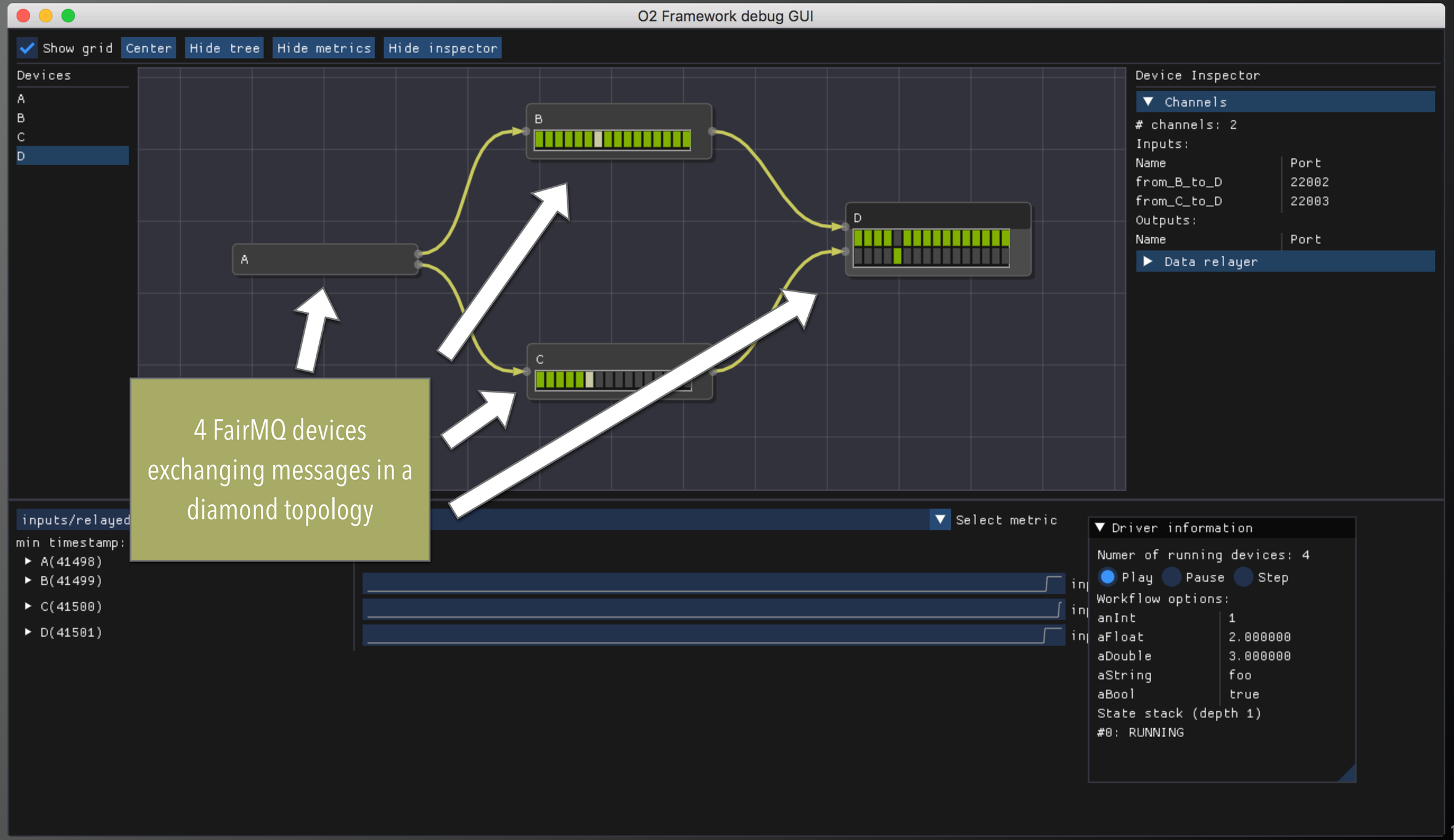


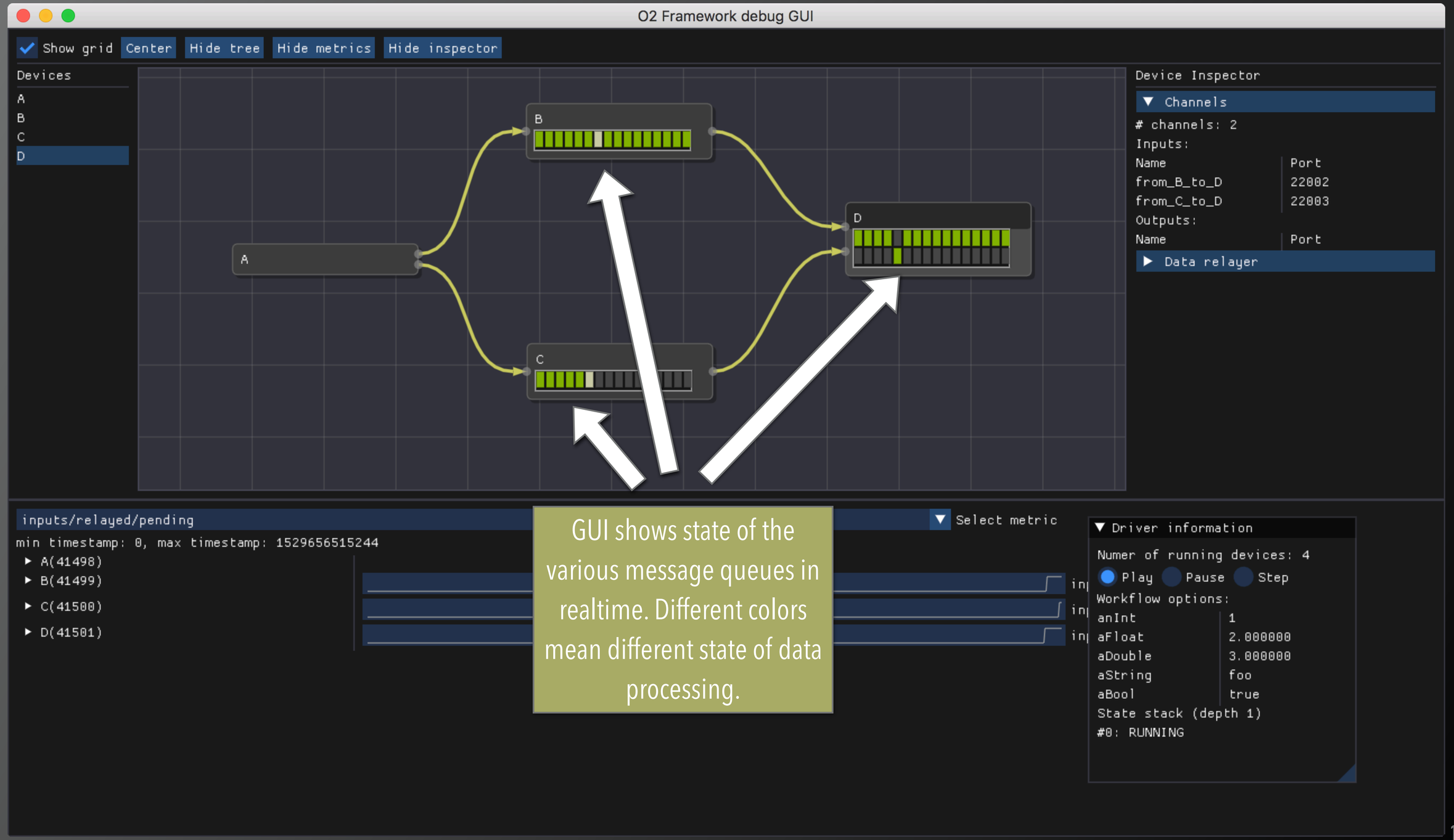


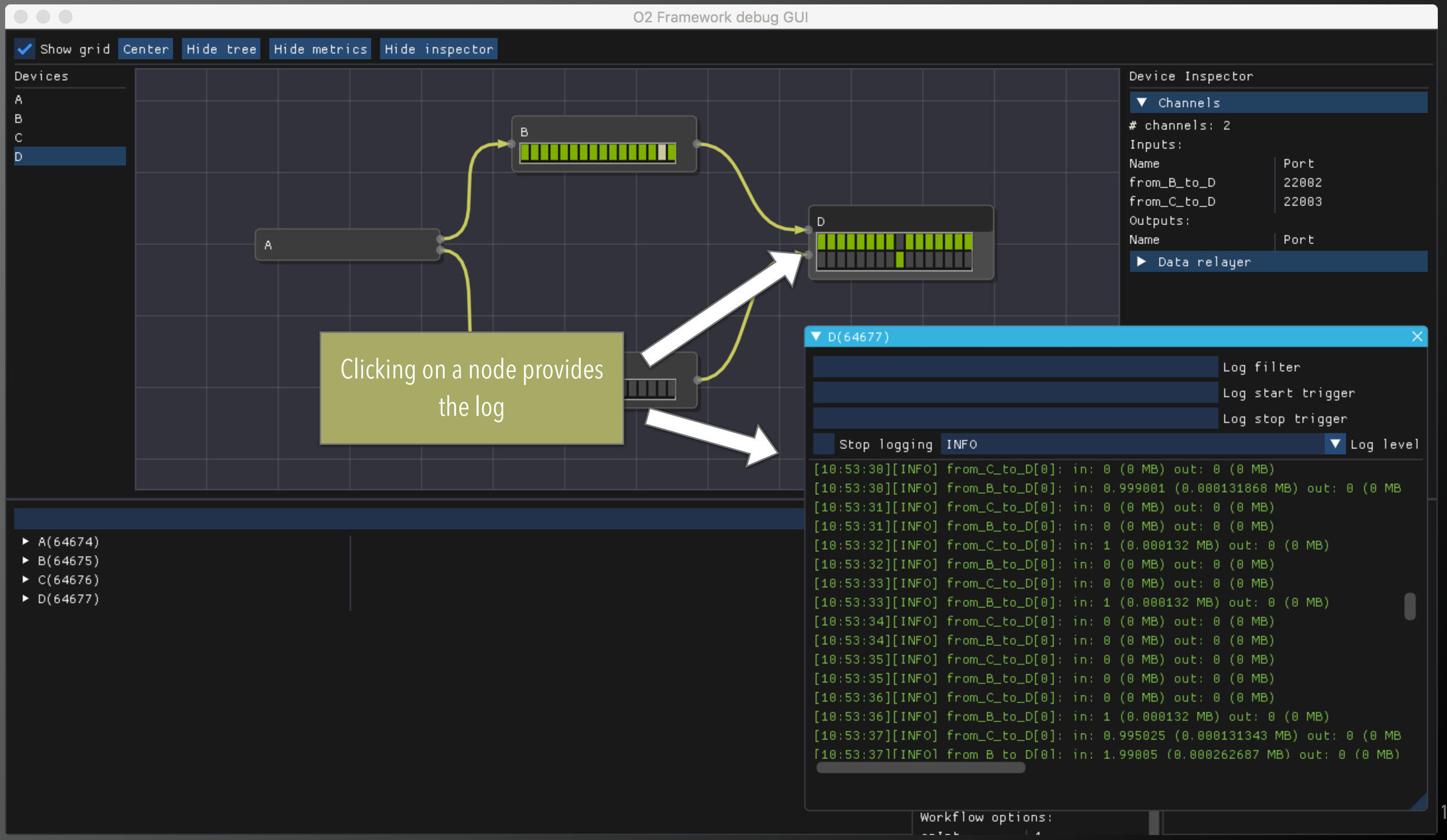
Data Processing Layer

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)

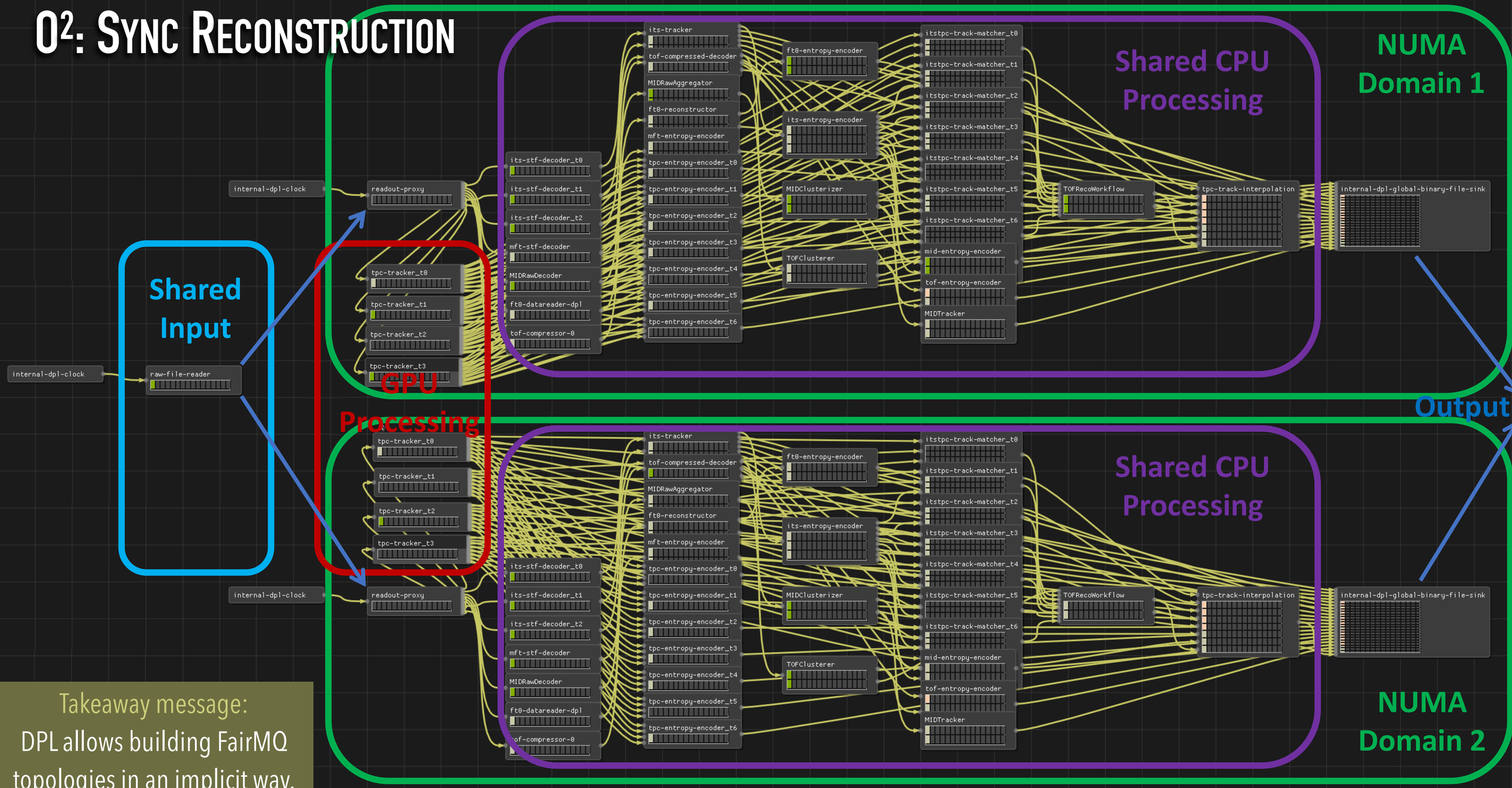

O2 Framework debug GUI
Show grid
Center
Hide tree
Hide metrics
Hide inspector
Devices
A
B
C
D
A
B
C
D
Device Inspector
Channels
channels: 2
Inputs:
Name
Port
from_B_to_D
22002
from_C_to_D
22003
Outputs:
Name
Port
Data relayer
Debug GUI
inputs/relayed/pending
Select metric
min timestamp: 0, max timestamp: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

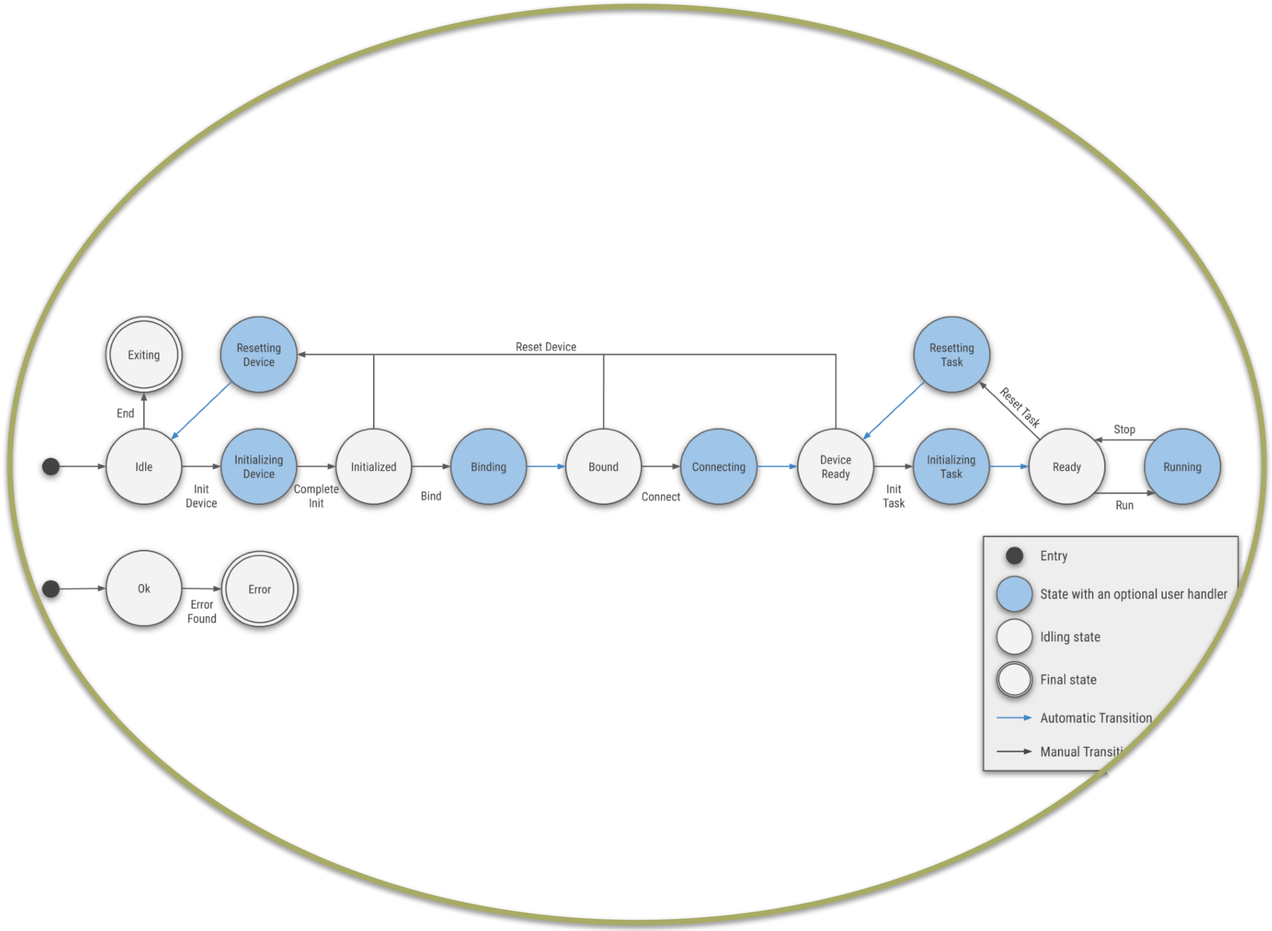
O²: Sync Reconstruction



Takeaway message:

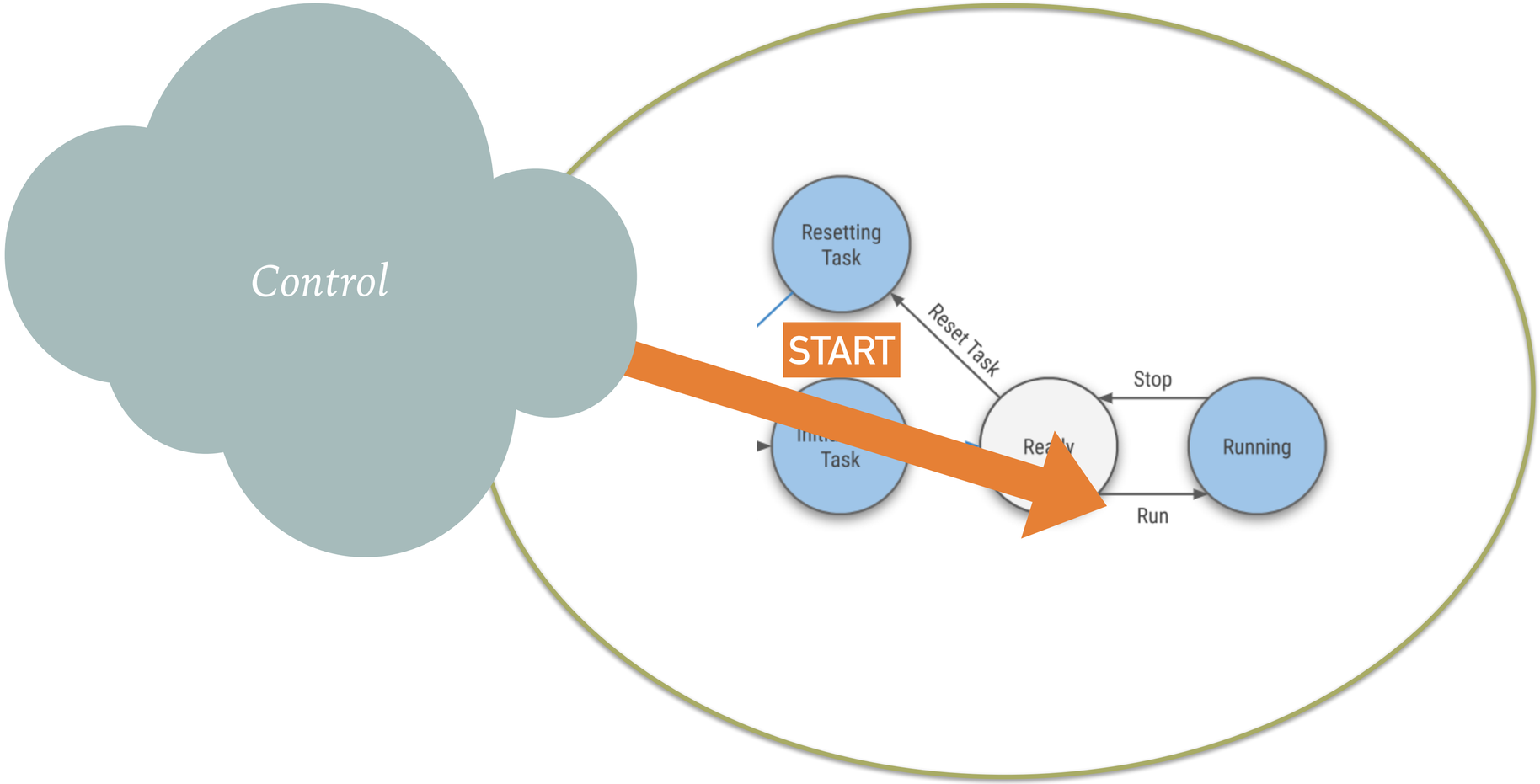
DPL allows building FairMQ topologies in an implicit way.

DATA PROCESSING LAYER: INTEGRATION WITH THE CONTROL SYSTEM



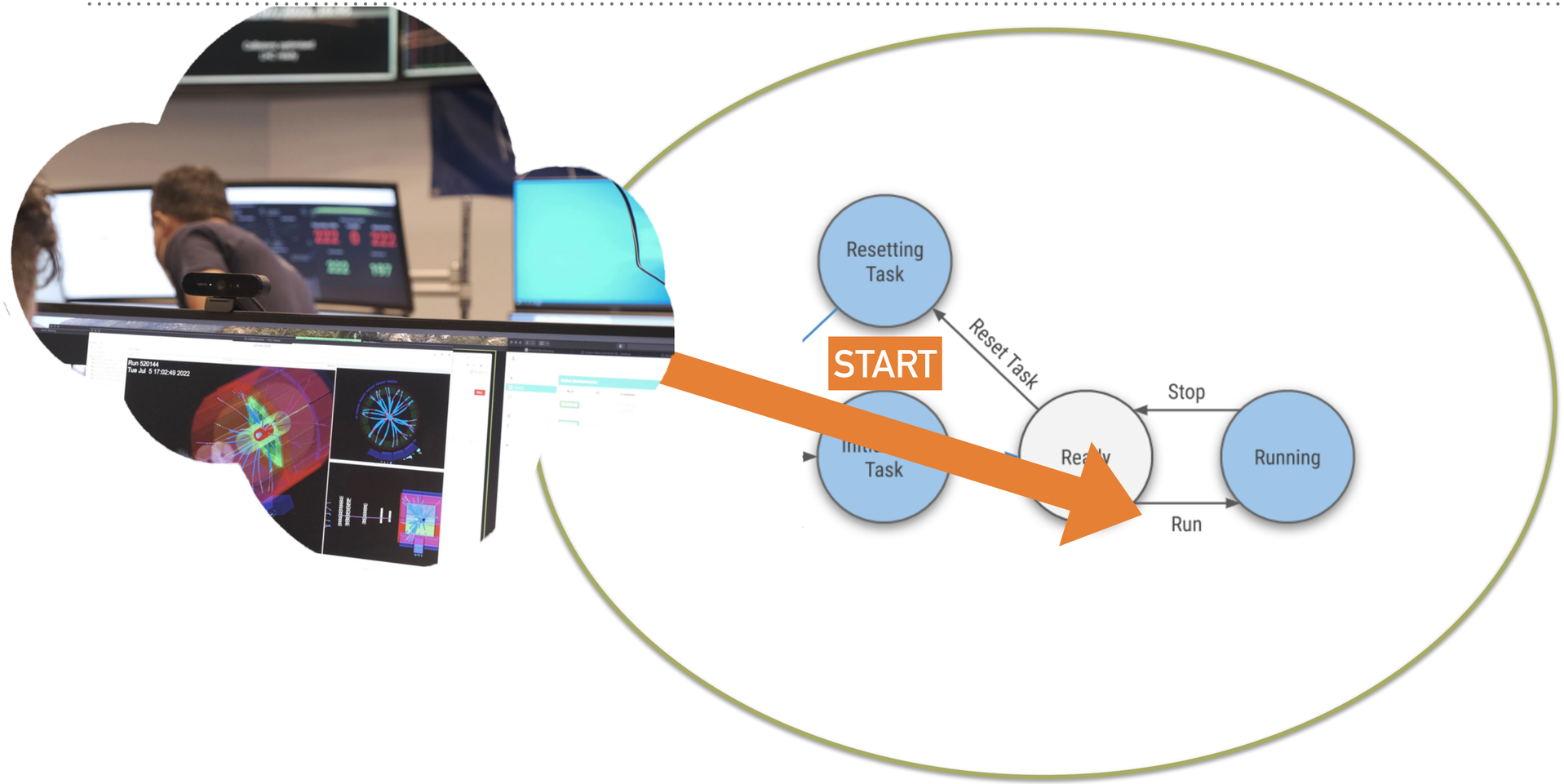
Each device runs a **finite state machine**.

DATA PROCESSING LAYER: INTEGRATION WITH THE CONTROL SYSTEM



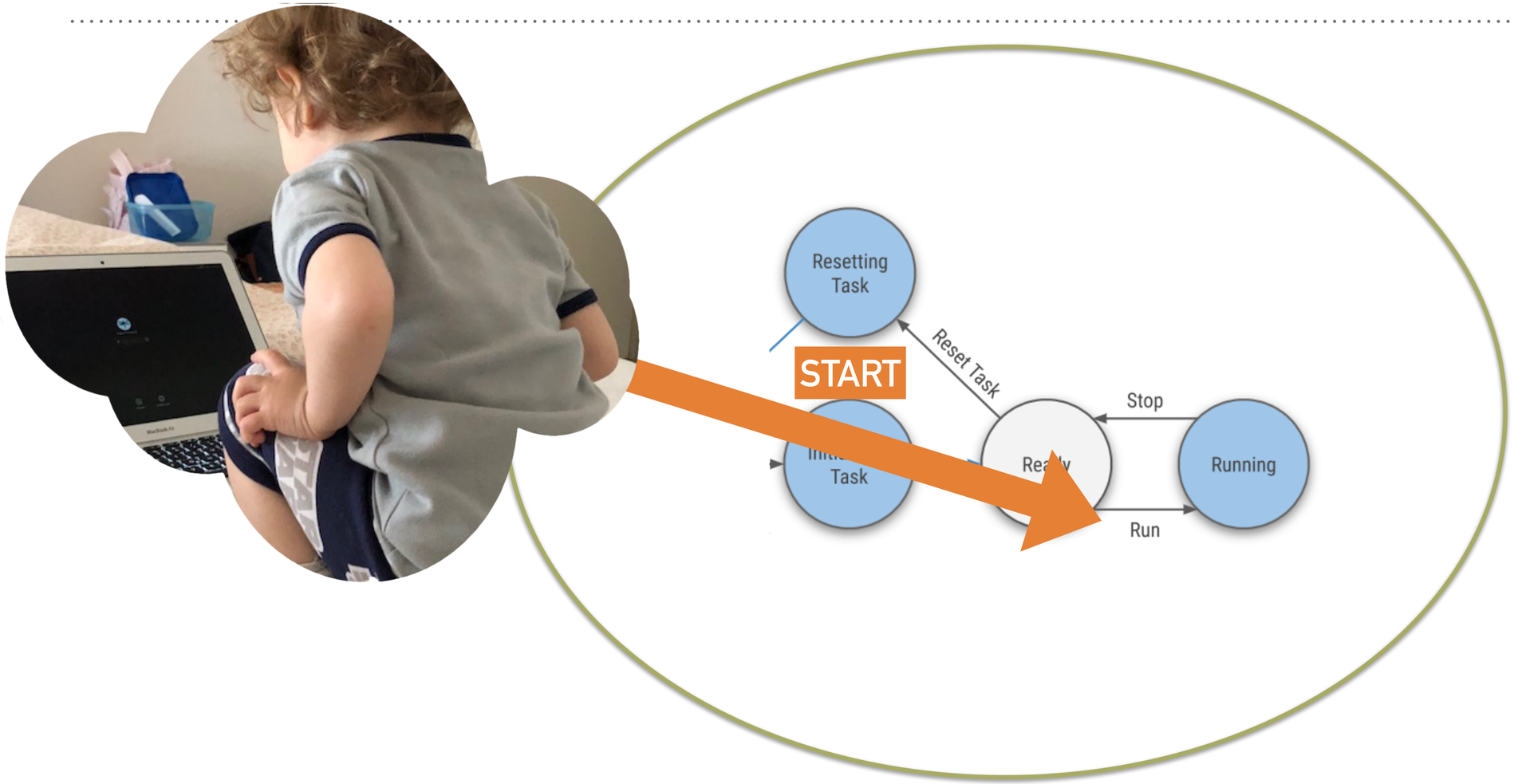
An **external control** is responsible to transition states.

DATA PROCESSING LAYER: INTEGRATION WITH THE CONTROL SYSTEM



An **external control** is responsible to transition states. At P2 this is integrated with the **Experiment Control System**...

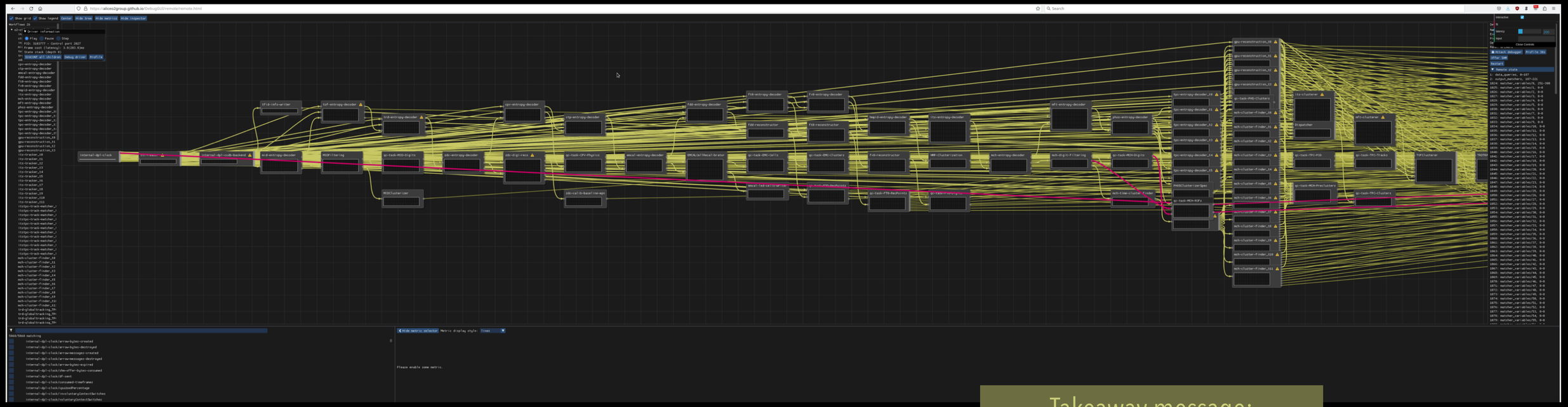
DATA PROCESSING LAYER: INTEGRATION WITH THE CONTROL SYSTEM



Takeaway message:
DPL abstracts away integration with the control system and deployment.

An **external control** is responsible to transition states. At P2 this is integrated with the **Experiment Control System**... while on the user laptop or on the grid we have a **DPL driver process** with such role.

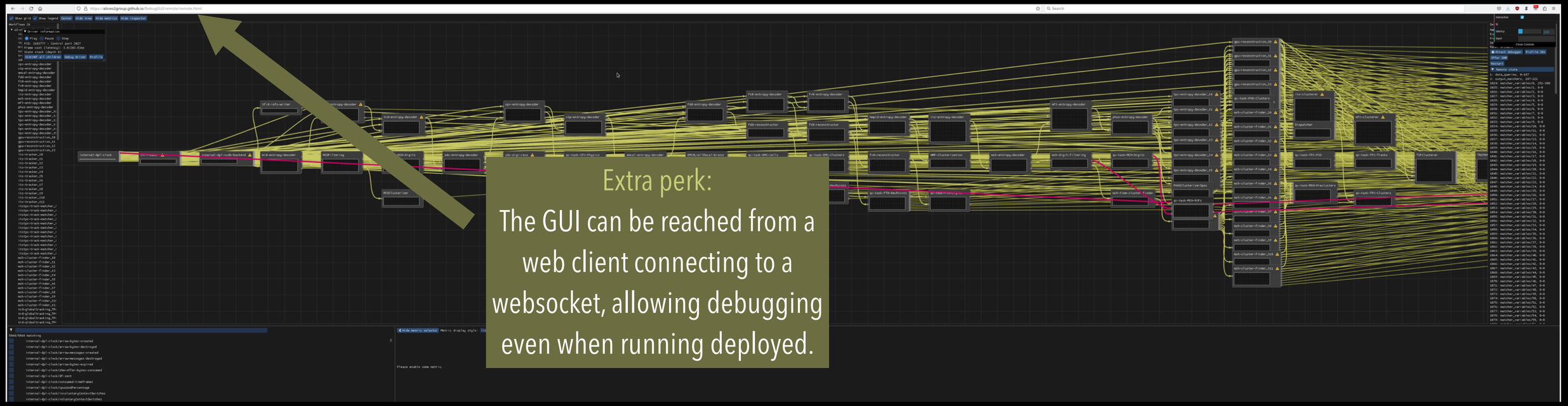
O²: ASYNC RECONSTRUCTION



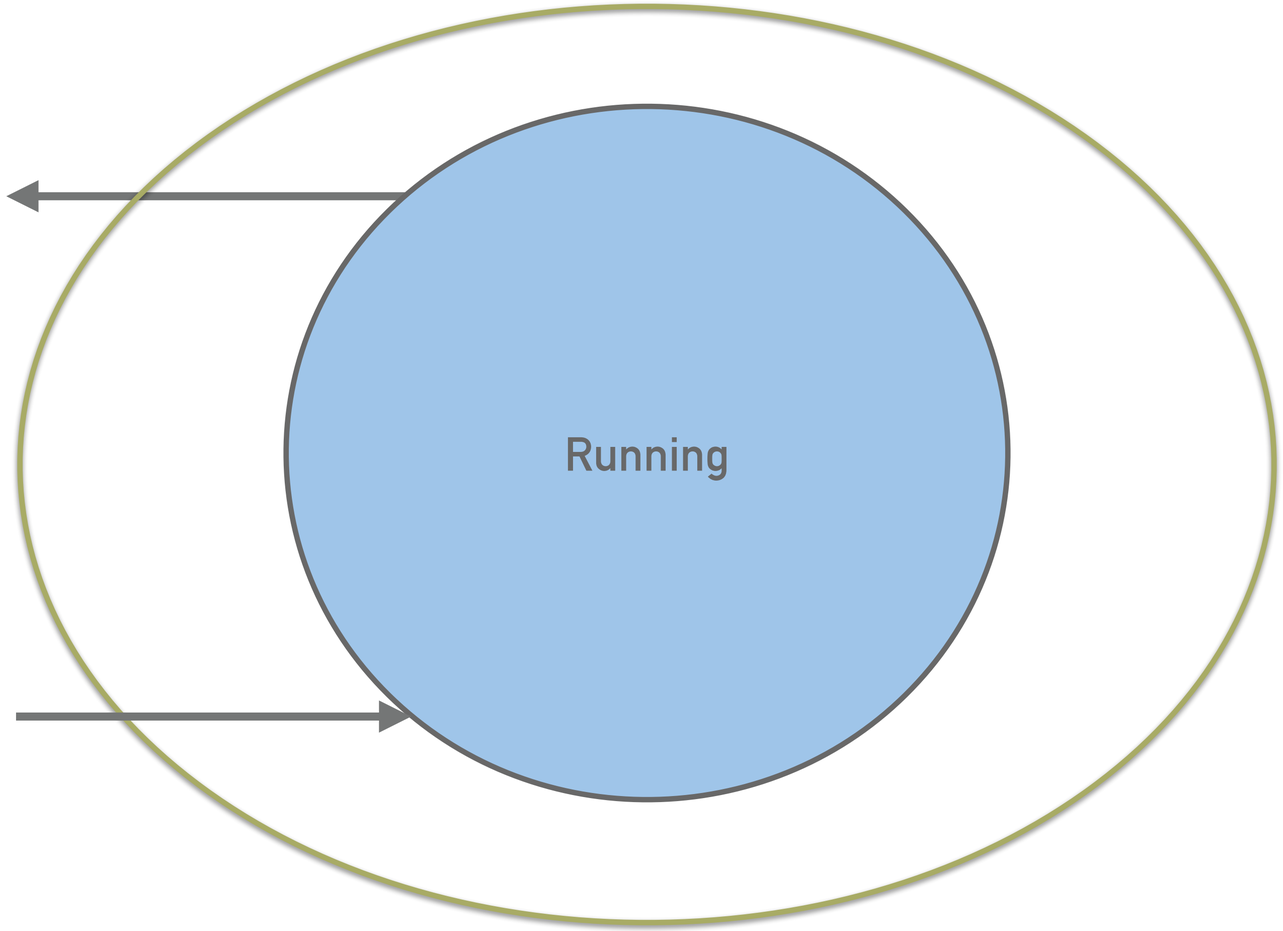
Takeaway message:

One single framework, from sync reconstruction to async and beyond.

O²: ASYNC RECONSTRUCTION

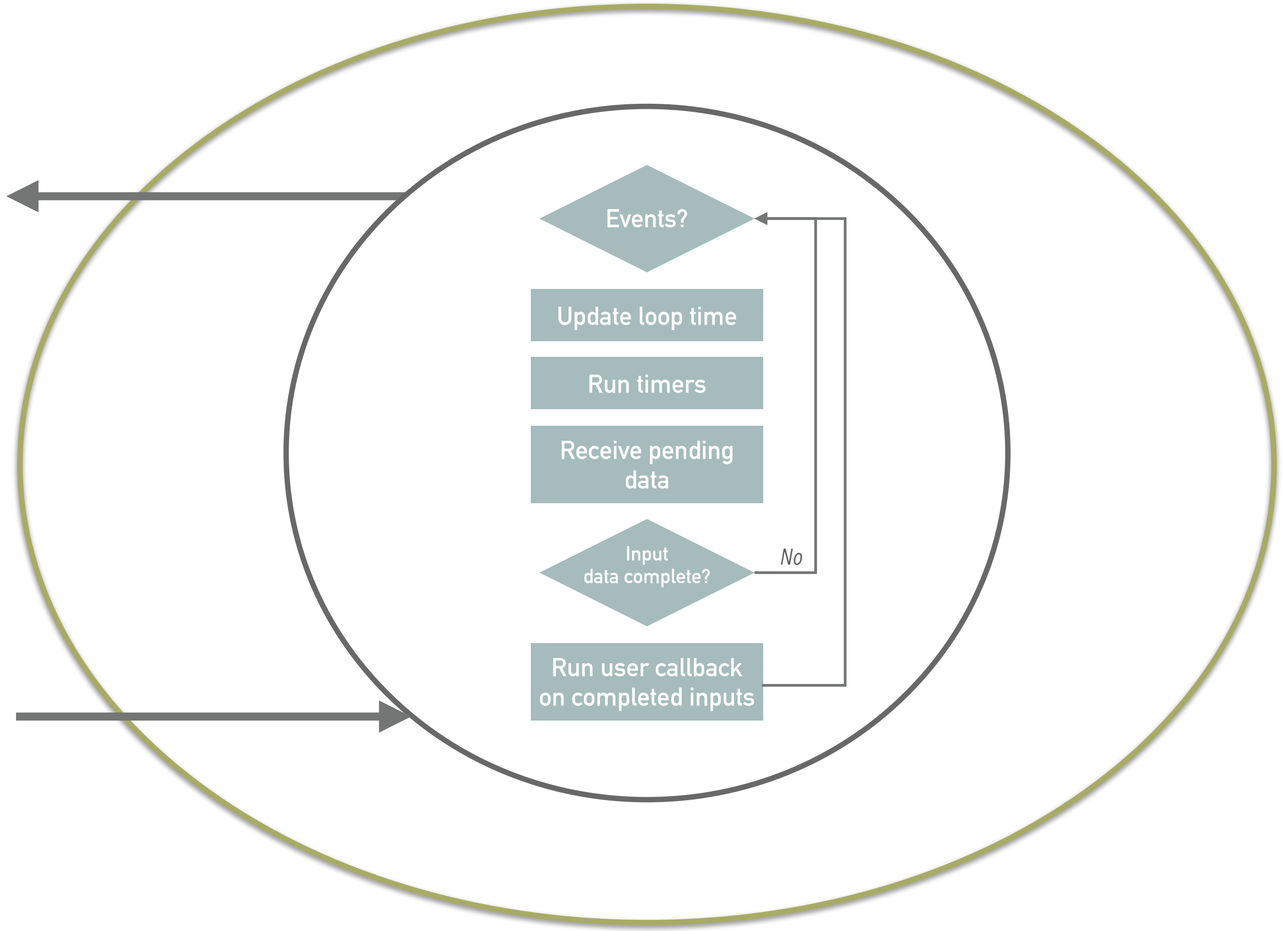


Data Processing Layer: Event Loop



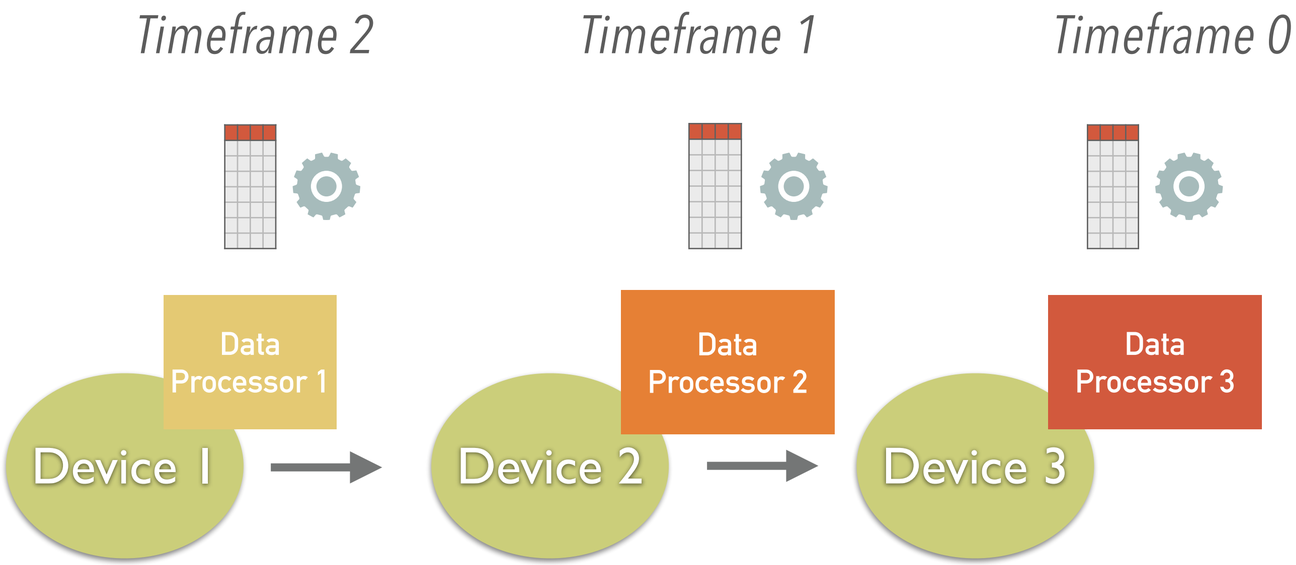
The Data Processing Layer (DPL) actually implements the Running state of a Device.

Data Processing Layer: Event Loop



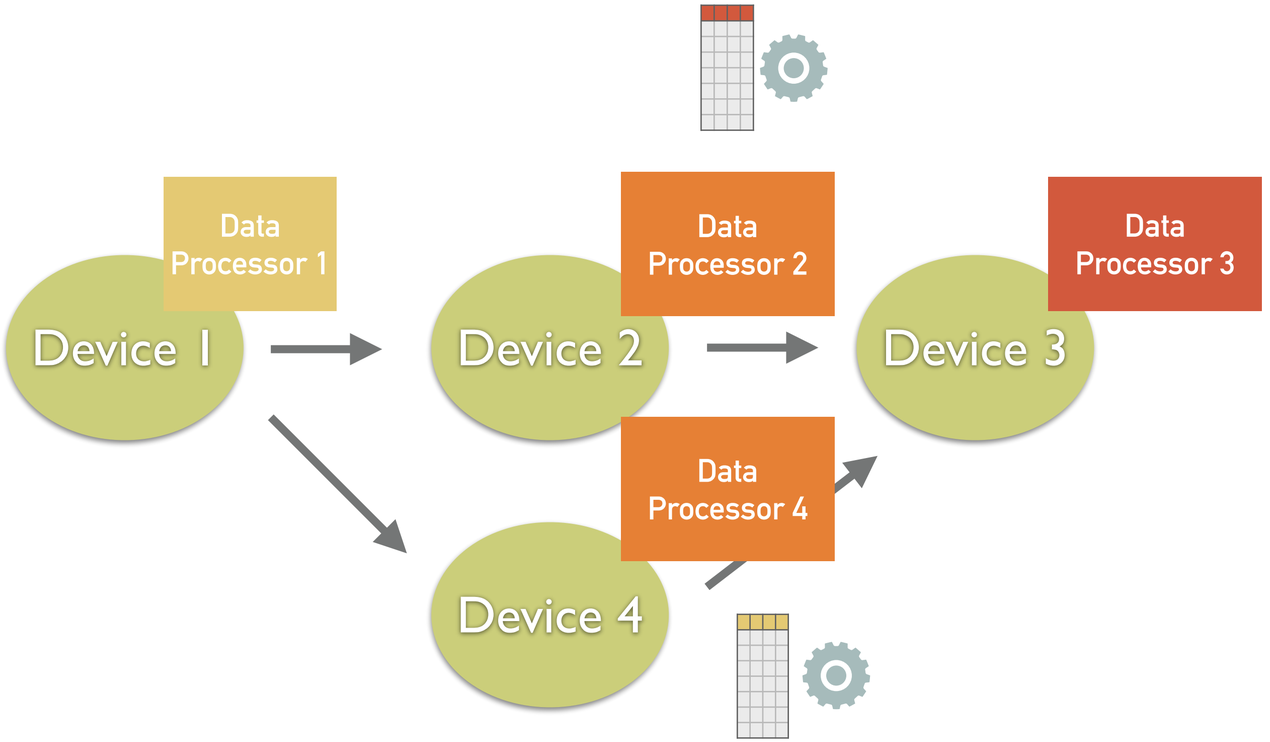
The (epoll / kqueue based) event loop only wakes up the device when there is something to do, e.g. handle incoming data to process using the user provided code.

Data Processing Layer: Parallelism Opportunities



By default, **we process inputs asynchronously**, where we can have more than one timeframe in fly at the same time. **Horizontal parallelism**.

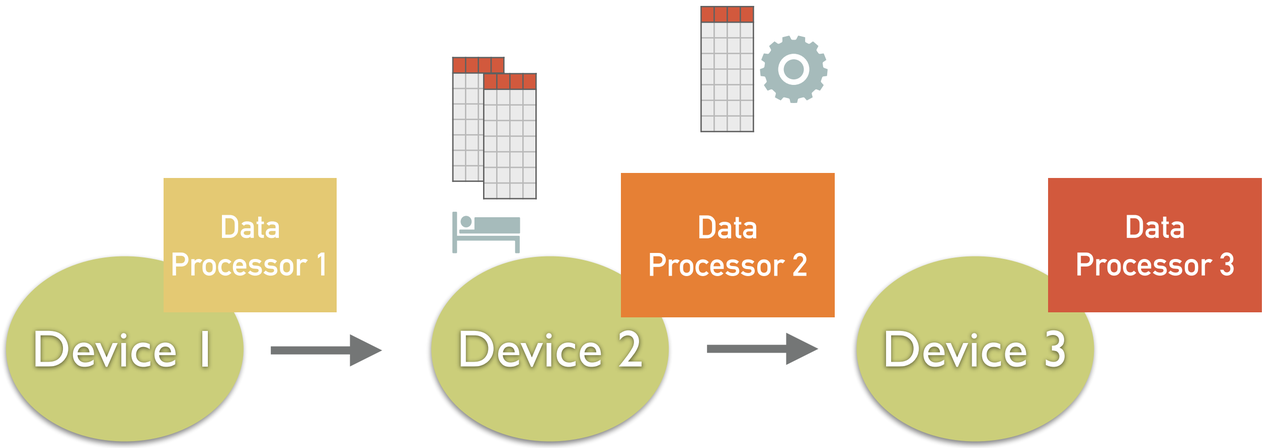
Data Processing Layer: Parallelism Opportunities



Different parts of a given timeframe can be processed in parallel.

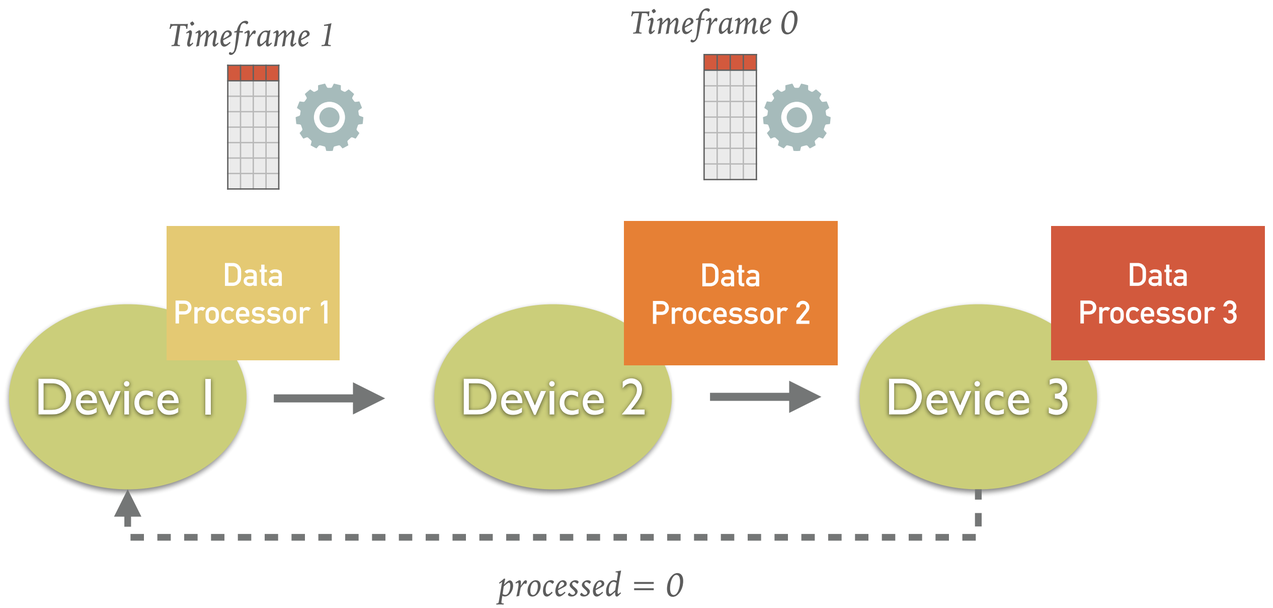
Vertical Parallelism.

Data Processing Layer: rate limiting



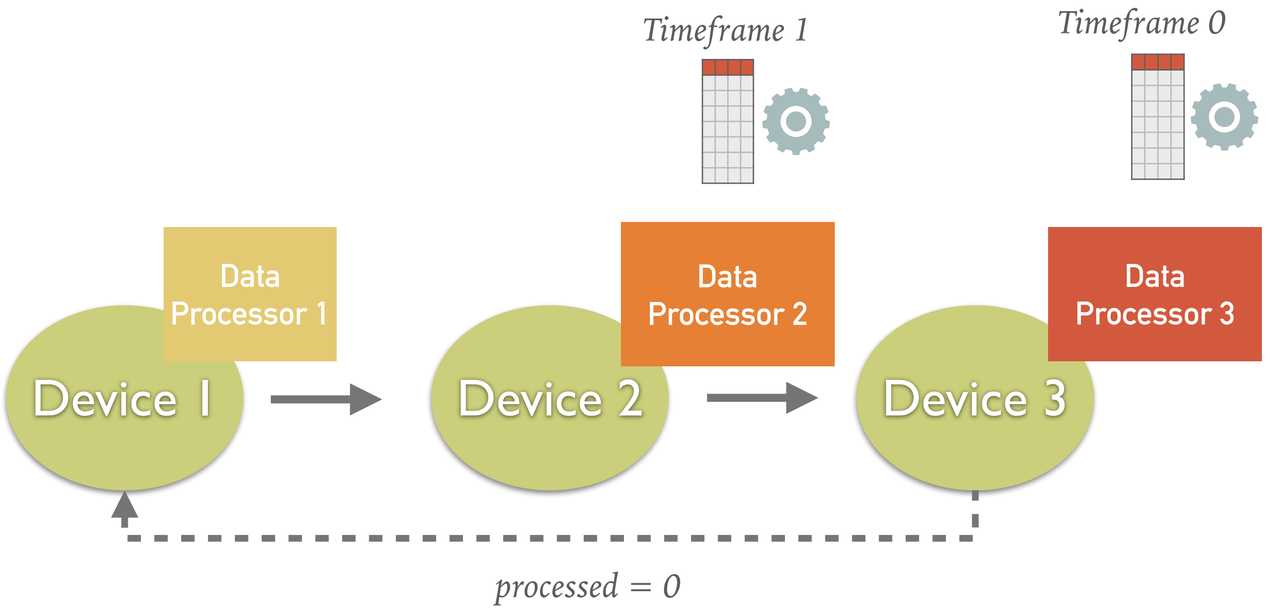
Without precautions, timeframes pile up in the input queue of the slowest device.

Data Processing Layer: rate limiting



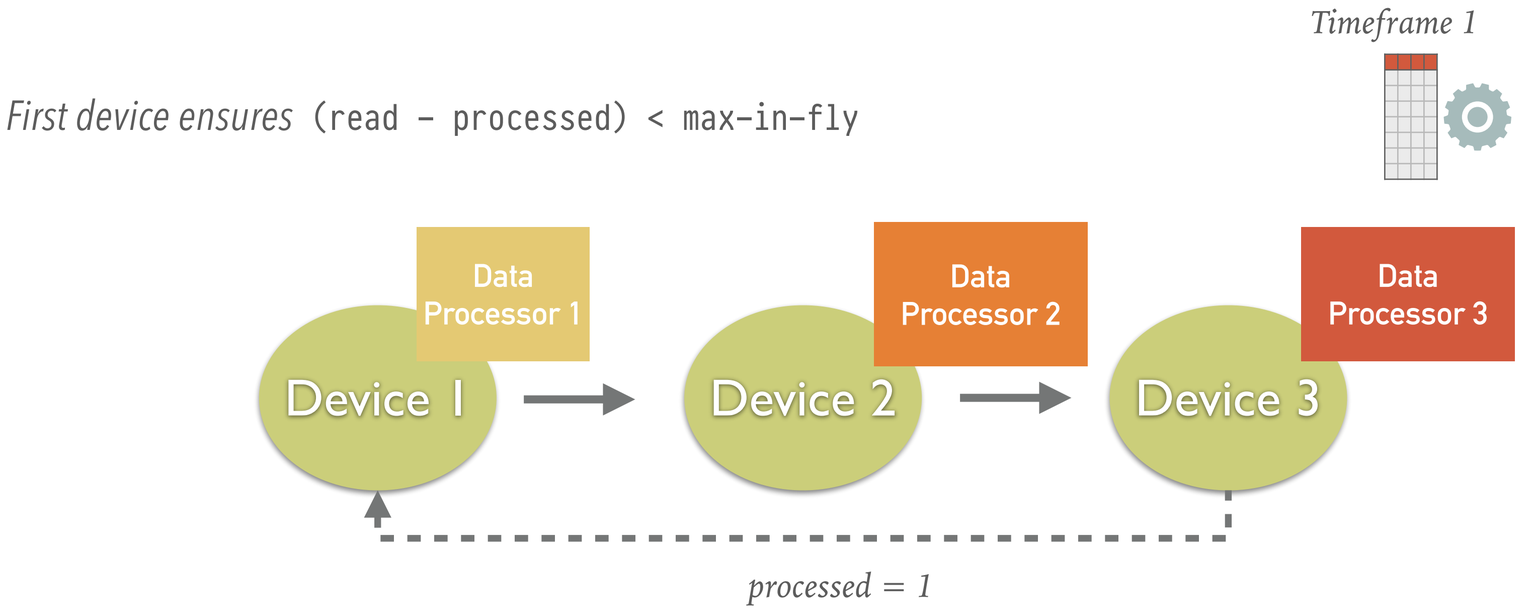
A back-channel reporting how many timeframes were processed to the source device is used to limit the number of in-fly timeframes.

Data Processing Layer: rate limiting



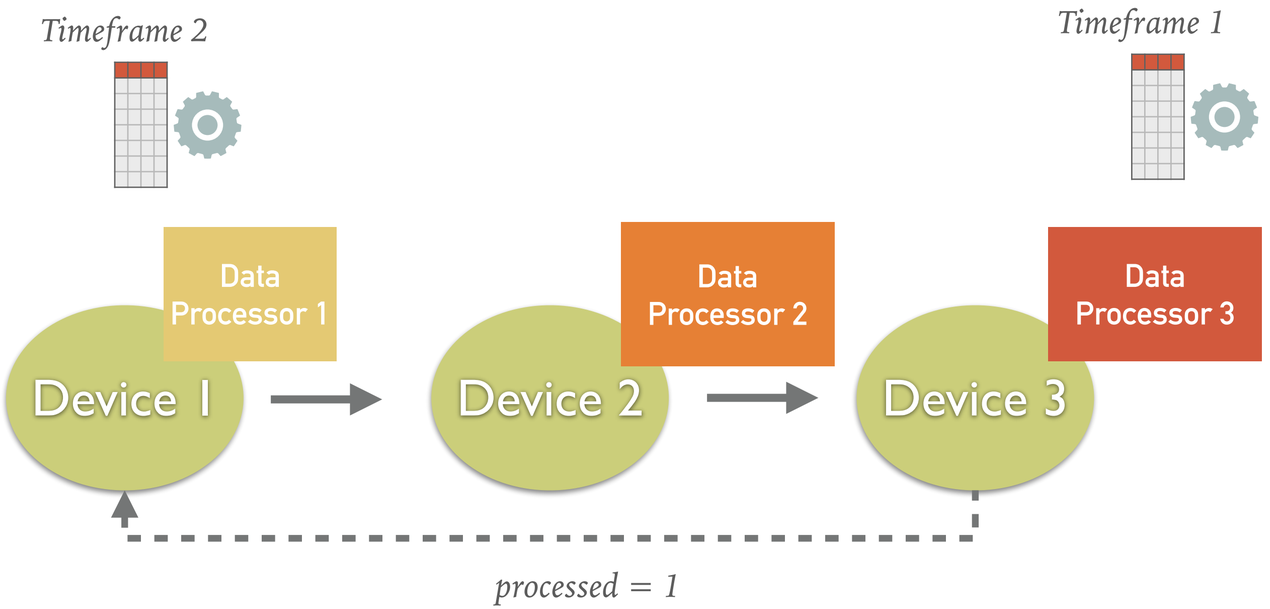
A back-channel reporting how many timeframes were processed to the source device is used to limit the number of in-fly timeframes.

Data Processing Layer: rate limiting



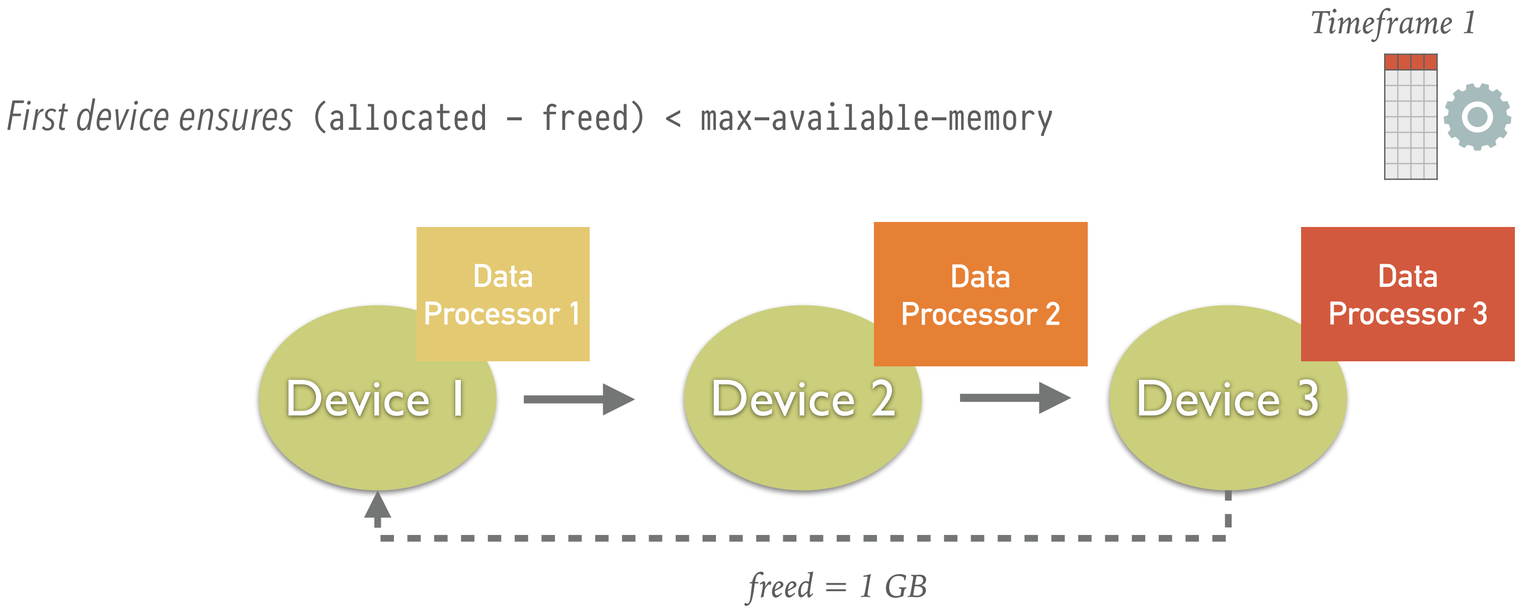
A back-channel reporting how many timeframes were processed to the source device is used to limit the number of in-fly timeframes.

Data Processing Layer: rate limiting



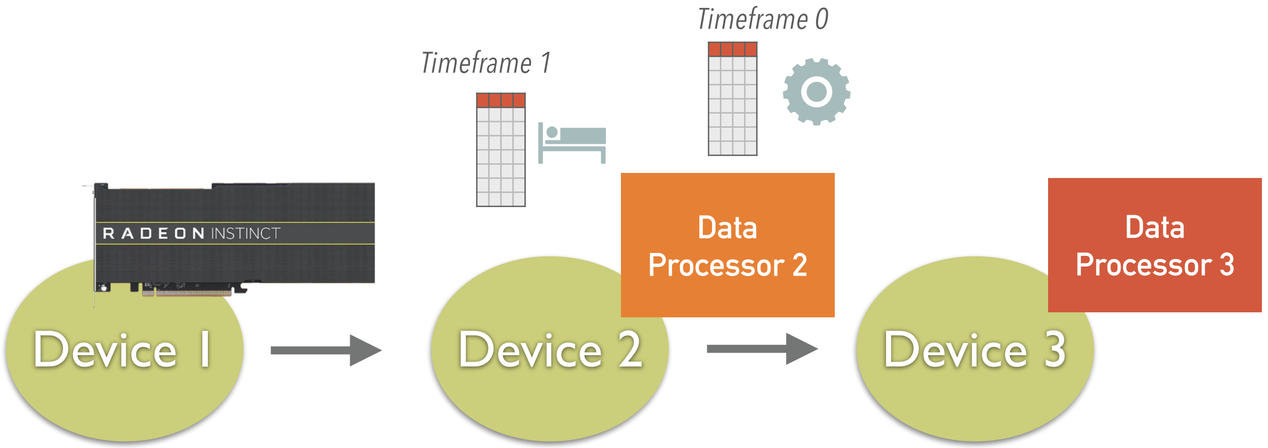
A back-channel reporting how many timeframes were processed to the source device is used to limit the number of in-fly timeframes.

Data Processing Layer: rate limiting



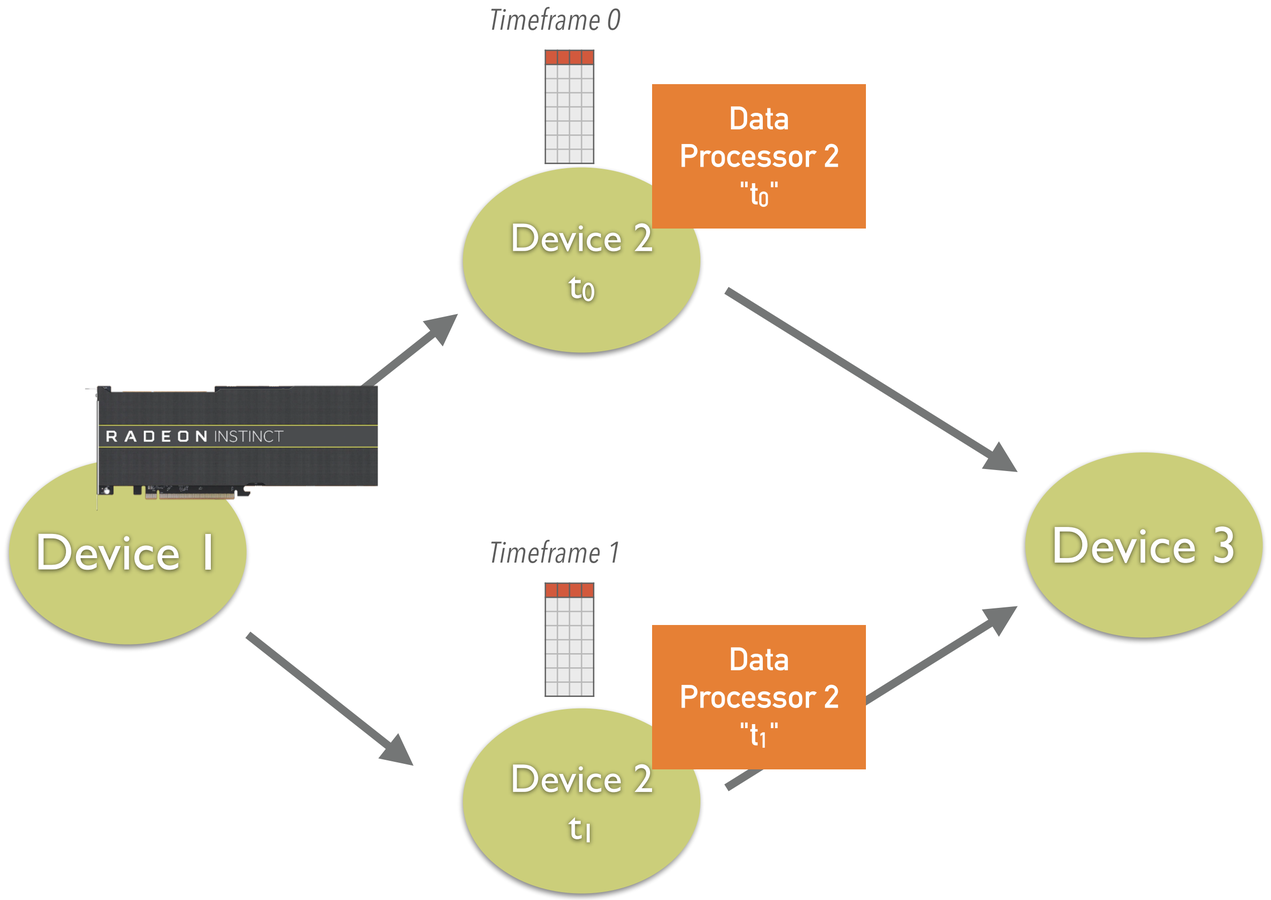
Besides the number of timeframes, we have the possibility to rate limit based on other quantities, e.g. available shared memory.

Data Processing Layer: Pipelining



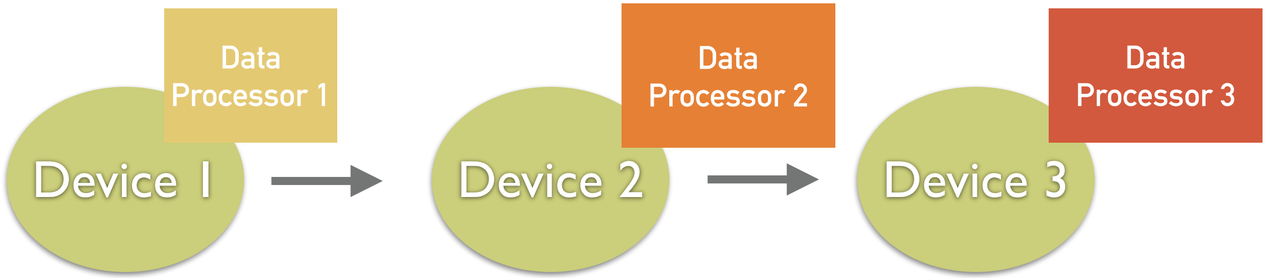
Parts of the chain can be faster due to offloading to GPUs. We can easily increase the number of downstream devices to increase throughput (at the cost of memory).

Data Processing Layer: Pipelining



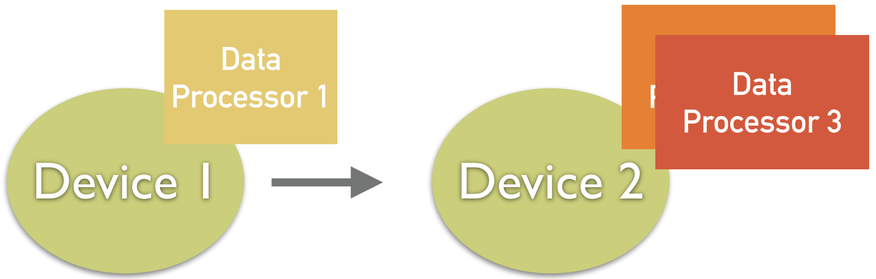
DPL allows to specify pipelining for a given DataProcessors, providing easy parallelisation of processing.

Data Processing Layer: Multiplexing



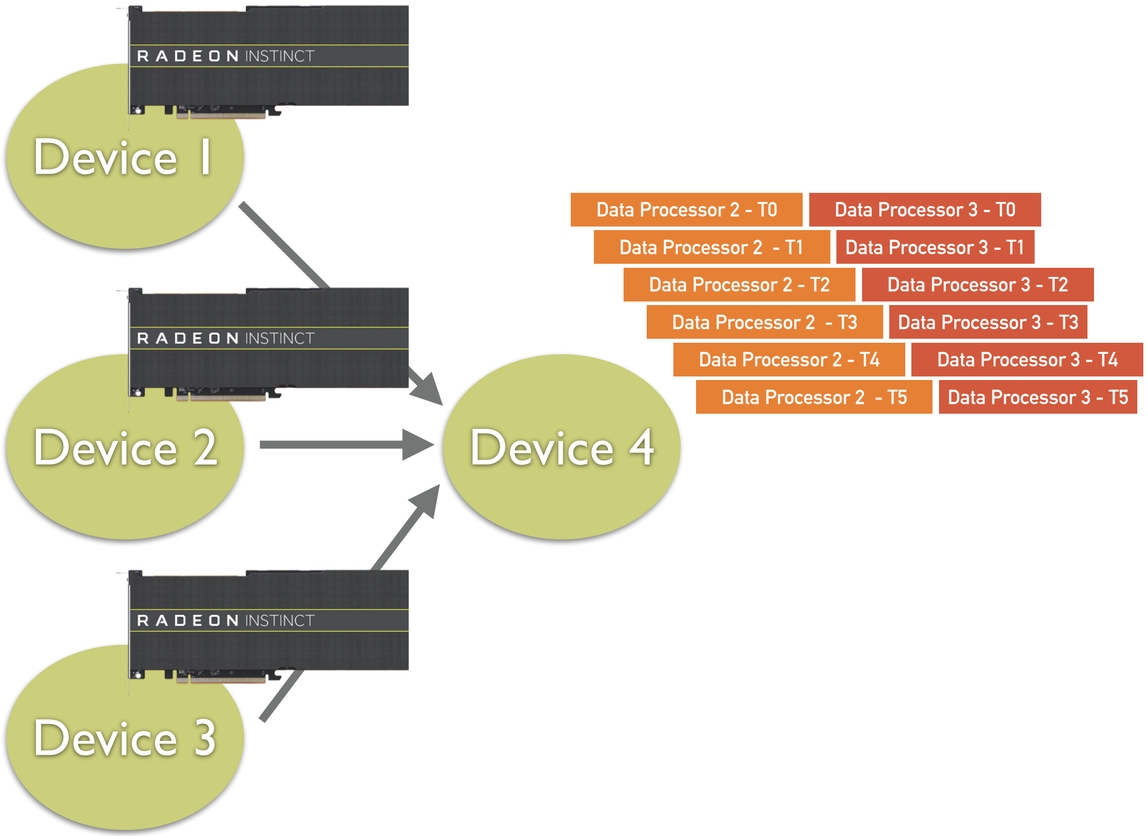
1-to-1 mapping between Devices and DataProcessors not mandatory!

Data Processing Layer: Multiplexing



We allow **multiple DataProcessors to run cooperatively** on the same device. This is **currently ad-hoc**, e.g. for digitisation. We are working to have it available in a generic way for the cases where the extra protections of multiprocessing are not needed.

Data Processing Layer: Future



We are working to **integrate multiplexing and pipelining** features to allow multithreaded execution of (thread safe) data processors.