The O2 software framework and GPU usage in ALICE online and offline reconstruction in Run 3

David Rohr, Giulio Eulisse for the ALICE Collaboration

CERN Compute Accelerator Forum

12.7.2023

drohr@cern.ch

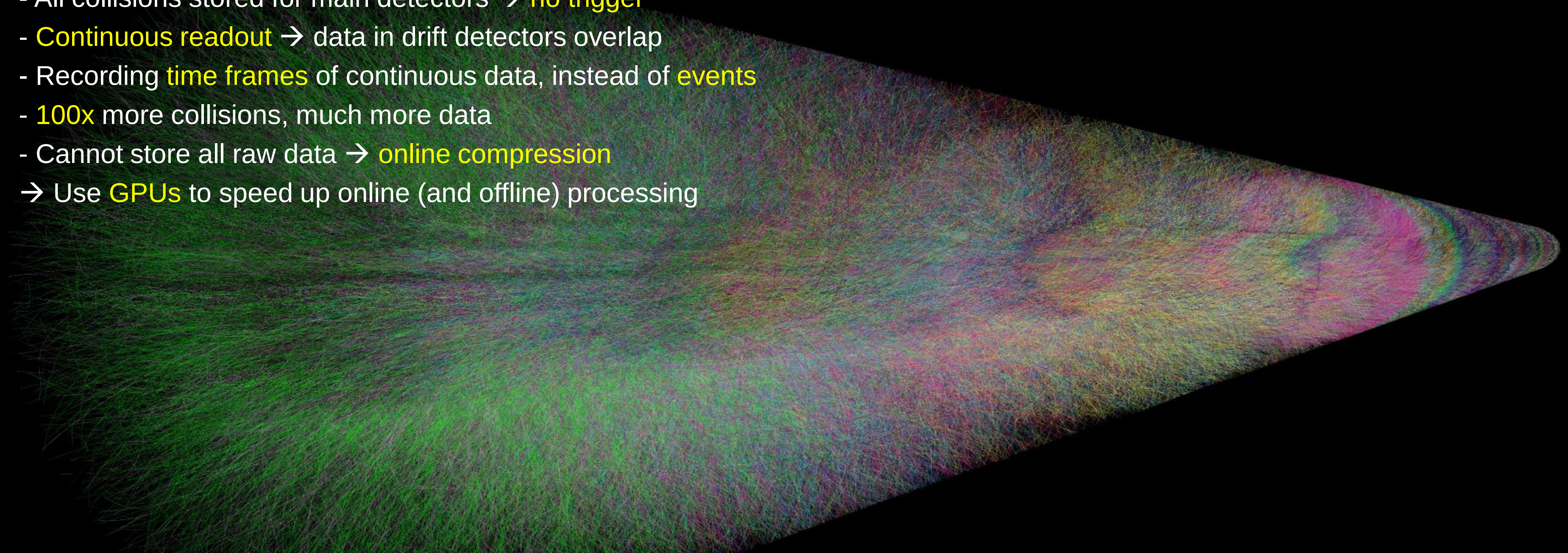
ALICE DATA TAKING / PROCESSING CONCEPT

ALICE in Run 3

Targeting to record large minimum bias sample.

- All collisions stored for main detectors → no trigger
- Continuous readout → data in drift detectors overlap
- Recording time frames of continuous data, instead of events
- 100x more collisions, much more data
- Cannot store all raw data → online compression

→ Use GPUs to speed up online (and offline) processing

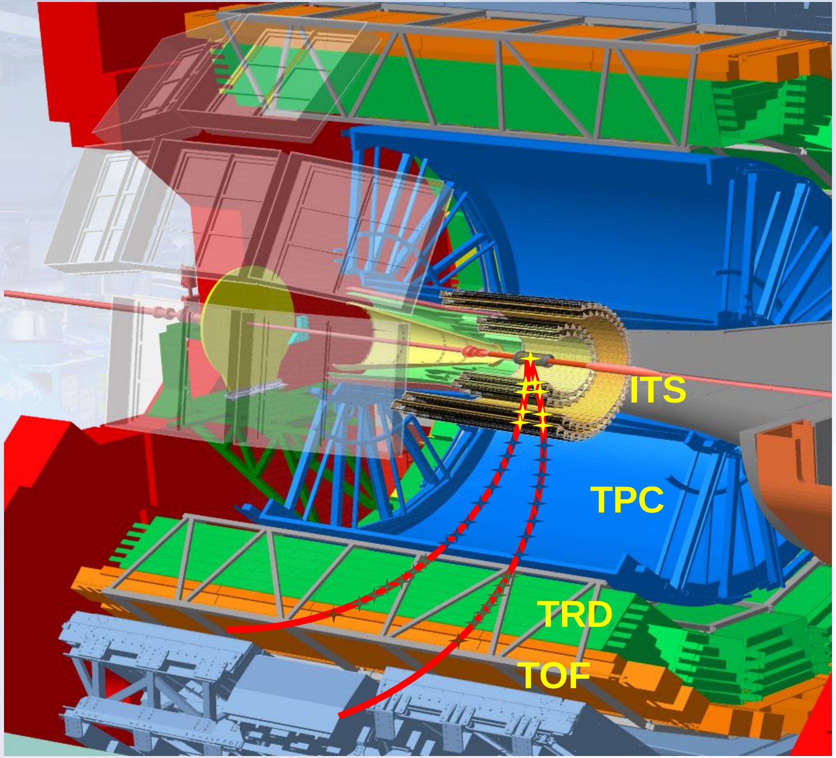


- Overlapping events in TPC with realistic bunch structure @ 50 kHz Pb-Pb.
- Timeframe of 2 ms shown (will be 10 – 20 ms in production).
- Tracks of different collisions shown in different colors.

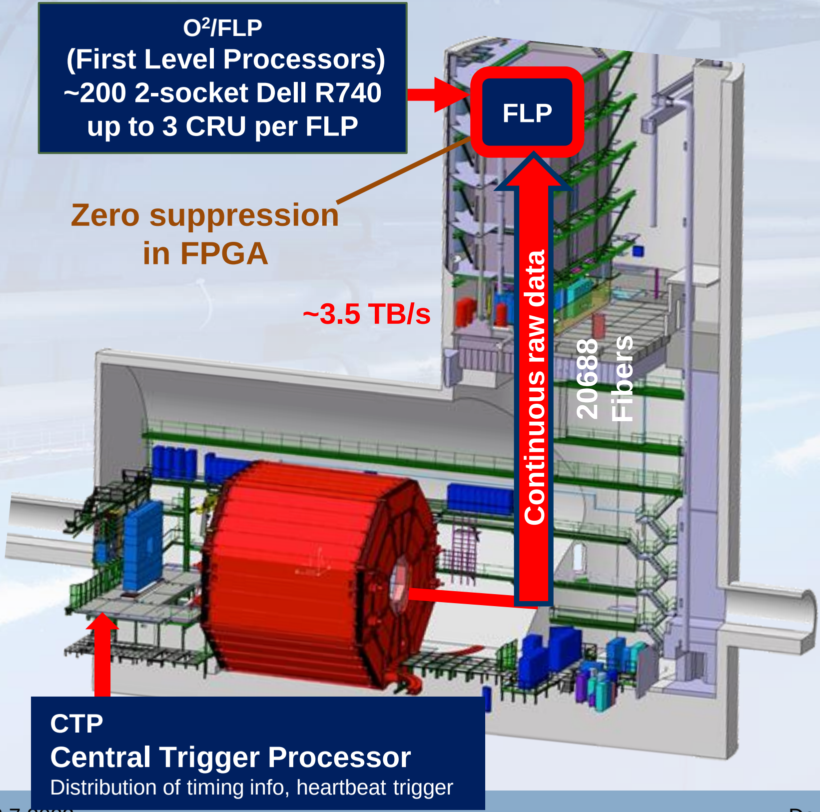
The ALICE detector in Run 3

- **ALICE uses mainly 3 detectors for barrel tracking: ITS, TPC, TRD + (TOF)**
  - **7 layers ITS** (Inner Tracking System – silicon tracker)
  - **152 pad rows TPC** (Time Projection Chamber)
  - **6 layers TRD** (Transition Radiation Detector)
  - **1 layer TOF** (Time Of Flight Detector)

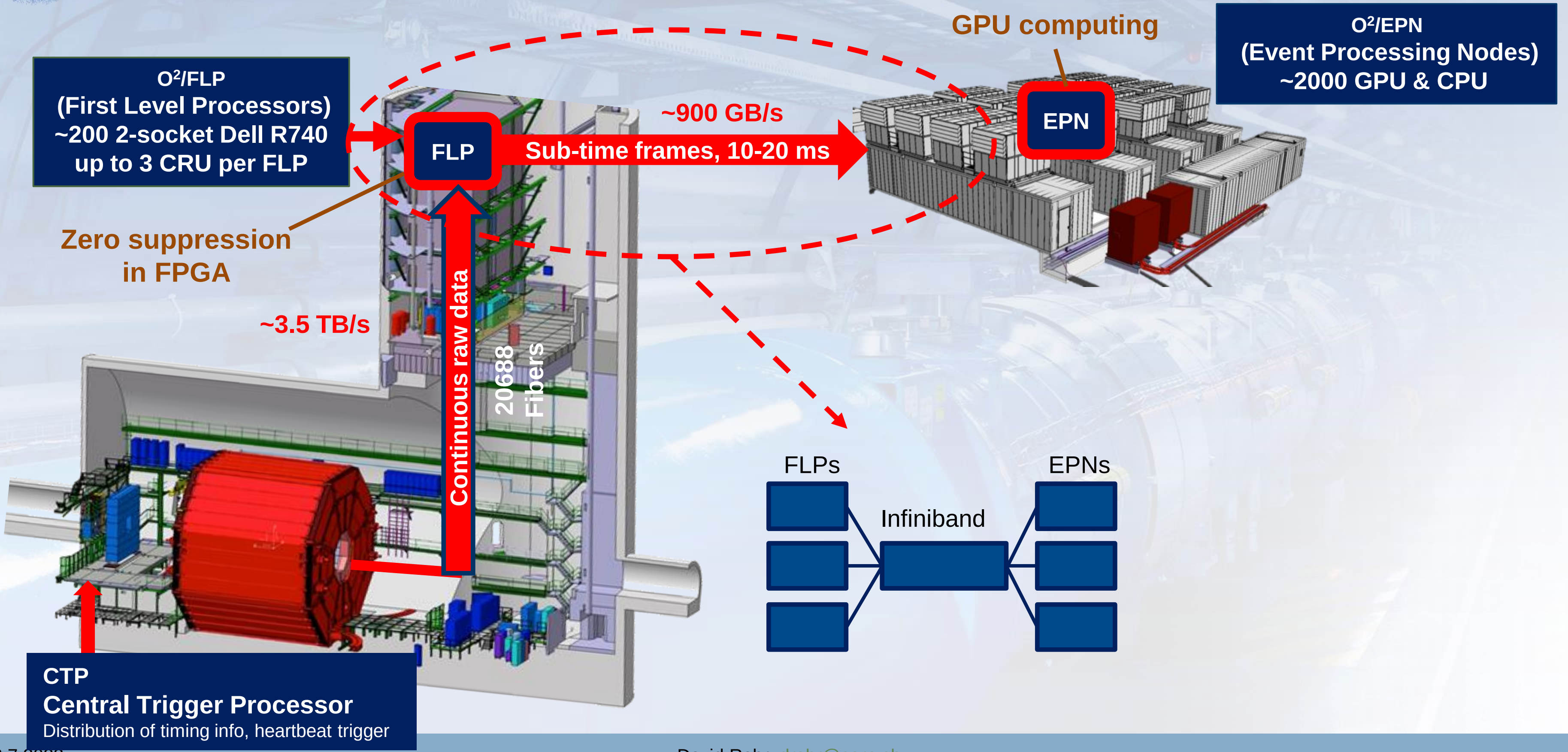
- **ALICE performs continuous readout.**
- **Native data unit is a time frame: all data from a configurable period of data up to 256 LHC orbits.**
  - Default was ~11 ms (128 LHC orbits) before 2023.
  - Current default is **~2.8 ms** (32 LHC orbits)



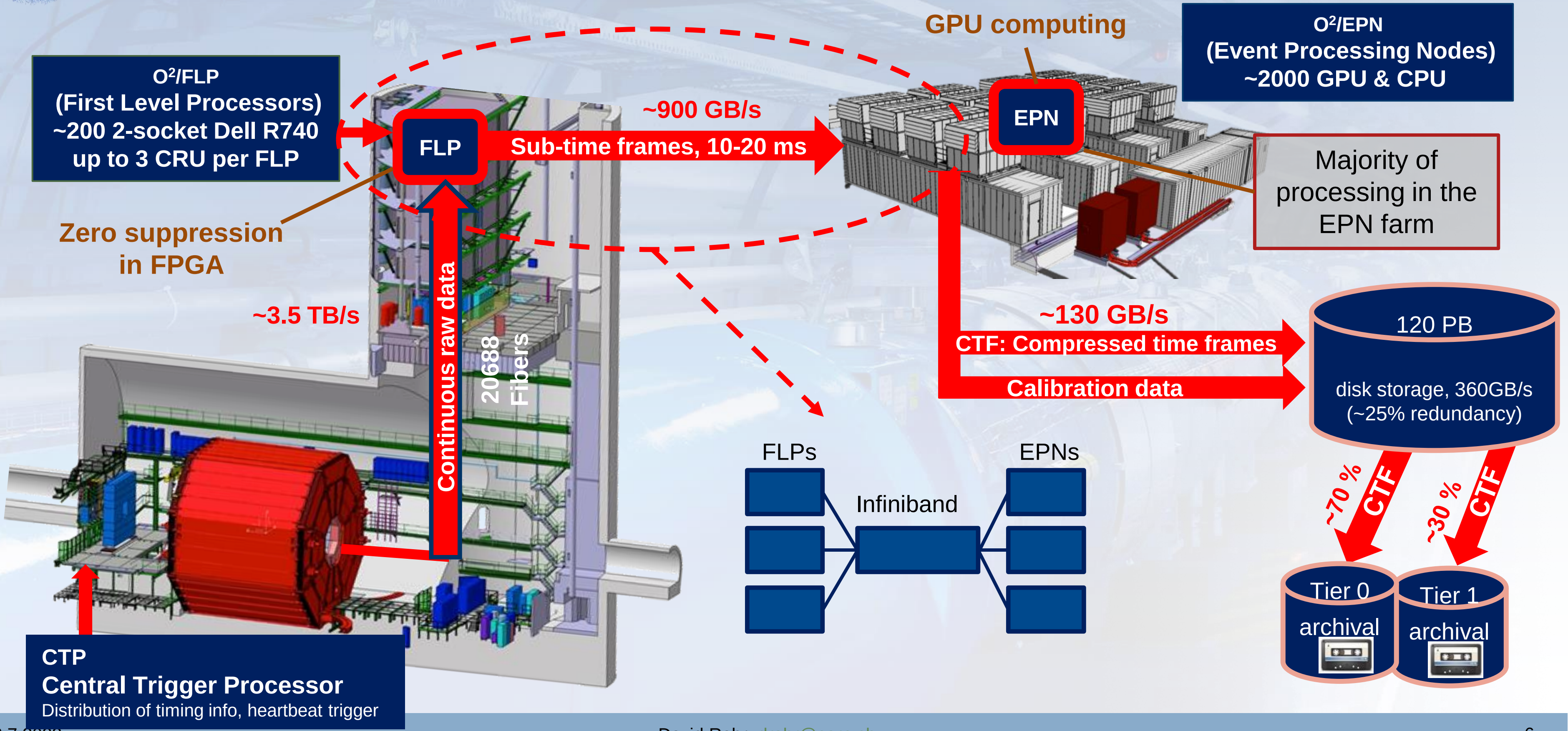
ALICE Raw Data Flow in Run 3



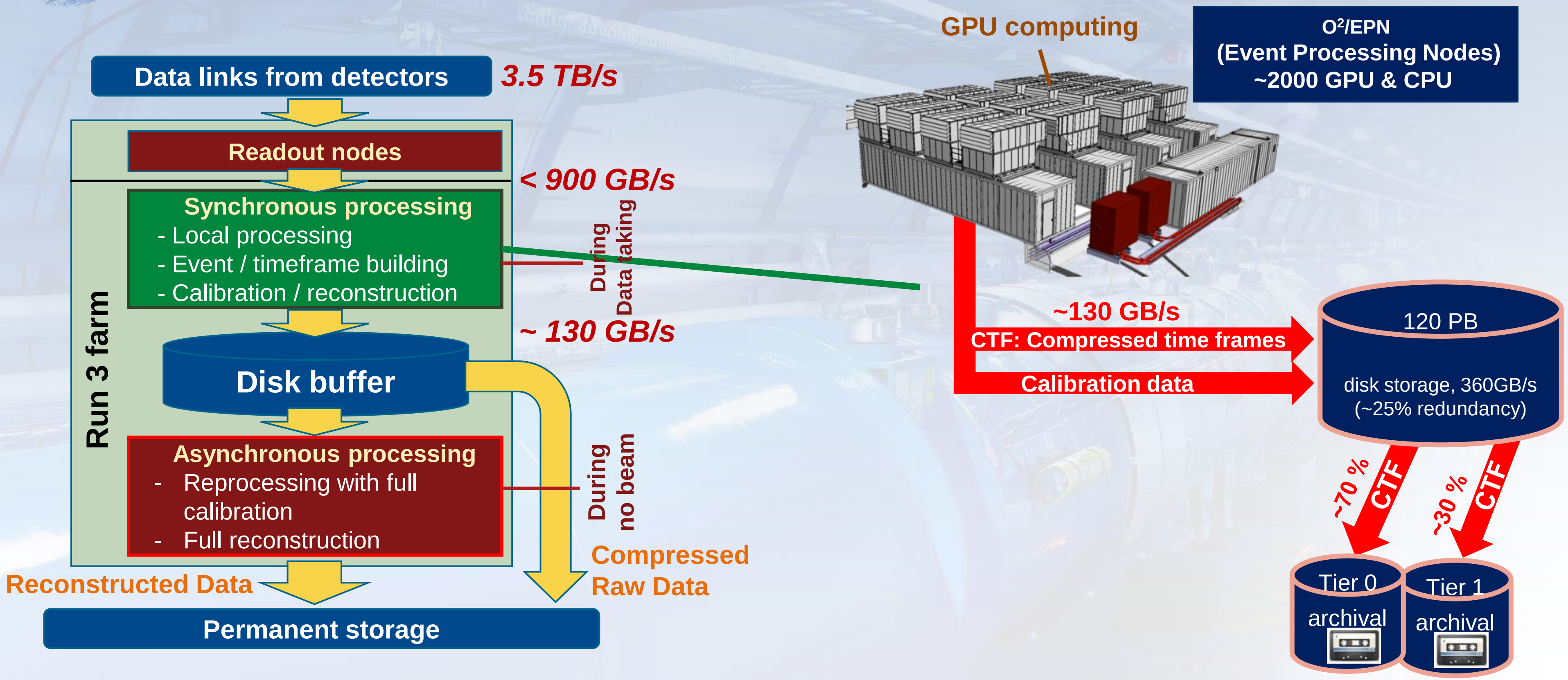
ALICE Raw Data Flow in Run 3



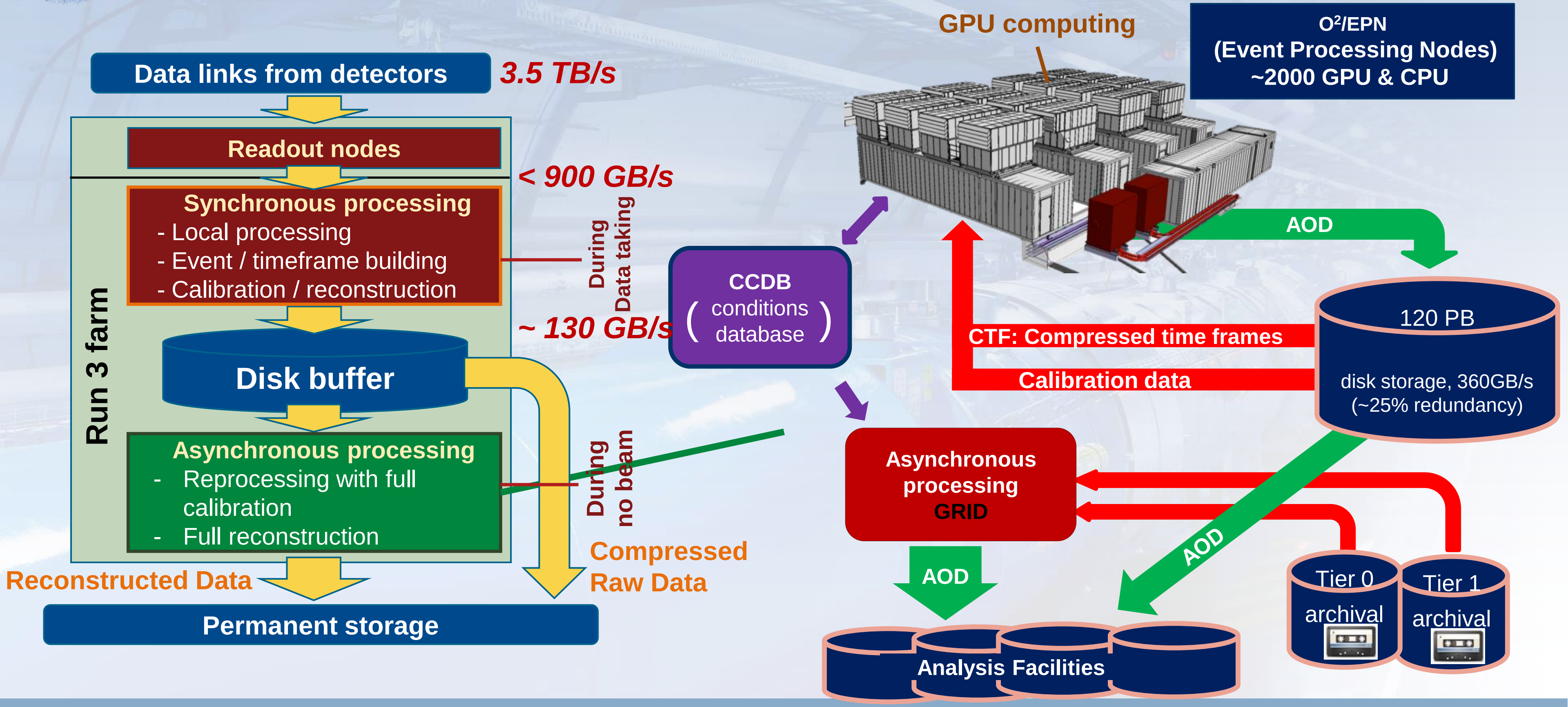
ALICE Raw Data Flow in Run 3



Synchronous and Asynchronous Processing



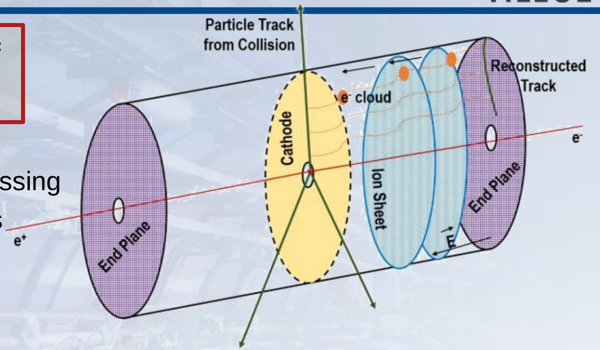
Synchronous and Asynchronous Processing



O^2 Processing steps

- **Synchronous processing (what we called online before):**
  - Extract information for **detector calibration**:
    - Previously performed in 2 offline passes over the data after the data taking
    - Run 3 avoids / reduces extra passes over the data but extracts all information in the sync. processing
    - An intermediate step between sync. and async. processing produces the final calibration objects
    - The most complicated calibration is the correction for the TPC space charge distortions

Needs tracking of 1% of tracks

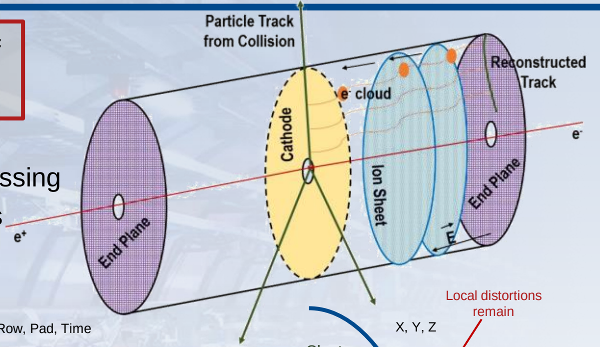


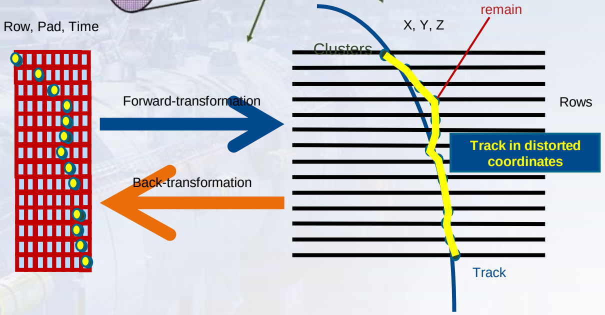
O² Processing steps

- **Synchronous processing (what we called online before):**
  - Extract information for **detector calibration**:
    - Previously performed in 2 offline passes over the data after the data taking
    - Run 3 avoids / reduces extra passes over the data but extracts all information in the sync. processing
    - An intermediate step between sync. and async. processing produces the final calibration objects
    - The most complicated calibration is the correction for the TPC space charge distortions
  - **Data compression**:
    - TPC is the largest contributor of raw data, and we employ sophisticated algorithms like storing space point coordinates as residuals to tracks to reduce the entropy and remove hits not attached to physics tracks
    - We use ANS entropy encoding for all detectors

Needs tracking of 1% of tracks

Needs 100% TPC tracking



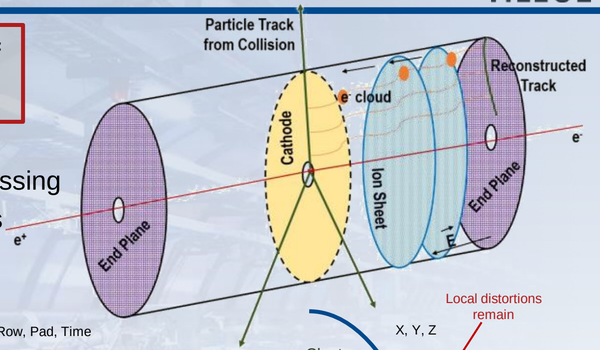


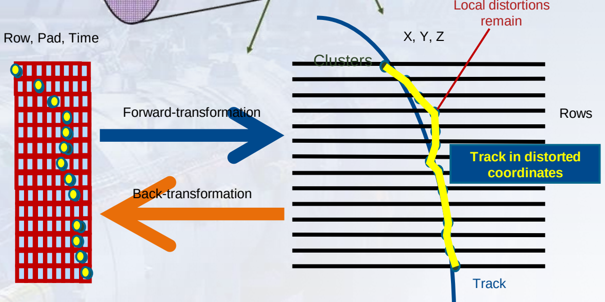
O² Processing steps

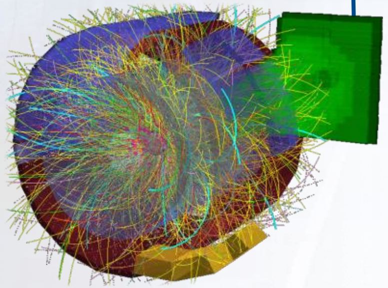
- **Synchronous processing (what we called online before):**
  - Extract information for **detector calibration**:
    - Previously performed in 2 offline passes over the data after the data taking
    - Run 3 avoids / reduces extra passes over the data but extracts all information in the sync. processing
    - An intermediate step between sync. and async. processing produces the final calibration objects
    - The most complicated calibration is the correction for the TPC space charge distortions
  - **Data compression**:
    - TPC is the largest contributor of raw data, and we employ sophisticated algorithms like storing space point coordinates as residuals to tracks to reduce the entropy and remove hits not attached to physics tracks
    - We use ANS entropy encoding for all detectors
  - **Event reconstruction** (tracking, etc.):
    - Required for calibration, compression, and online quality control
    - Need **full TPC tracking** for data compression
    - Need tracking in all detectors for ~1% of the tracks for calibration
    - → **TPC tracking dominant part**, **rest** almost negligible (**< 5%**)

Needs tracking of 1% of tracks

Needs 100% TPC tracking





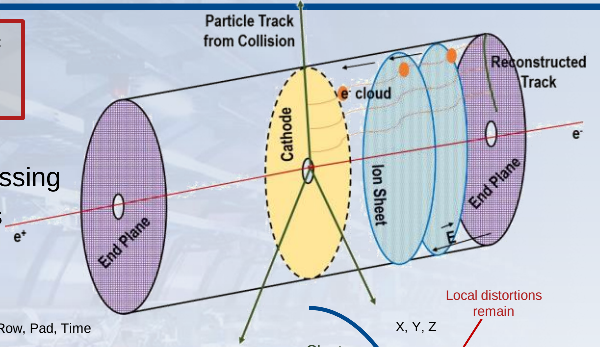


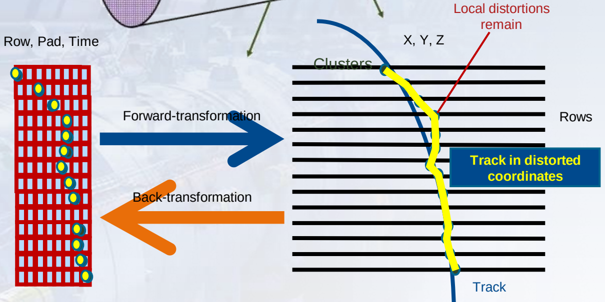
O² Processing steps

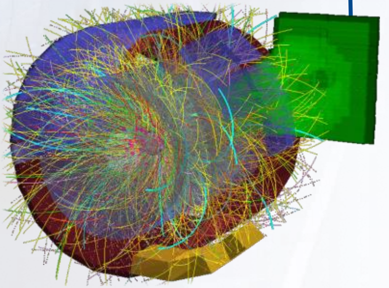
- **Synchronous processing (what we called online before):**
  - Extract information for **detector calibration**:
    - Previously performed in 2 offline passes over the data after the data taking
    - Run 3 avoids / reduces extra passes over the data but extracts all information in the sync. processing
    - An intermediate step between sync. and async. processing produces the final calibration objects
    - The most complicated calibration is the correction for the TPC space charge distortions
  - **Data compression**:
    - TPC is the largest contributor of raw data, and we employ sophisticated algorithms like storing space point coordinates as residuals to tracks to reduce the entropy and remove hits not attached to physics tracks
    - We use ANS entropy encoding for all detectors
  - **Event reconstruction** (tracking, etc.):
    - Required for calibration, compression, and online quality control
    - Need **full TPC tracking** for data compression
    - Need tracking in all detectors for ~1% of the tracks for calibration
    - → **TPC tracking dominant part**, **rest** almost negligible (**< 5%**)

Needs tracking of 1% of tracks

Needs 100% TPC tracking







- **Asynchronous processing (what we called offline before):**
  - **Full reconstruction**, **full calibration**, **all detectors**
  - TPC part faster than in synchronous processing (less hits, no clustering, no compression)
  - → **Different relative importance of GPU / CPU** algorithms compared to synchronous processing

O²: Software Framework in One Slide

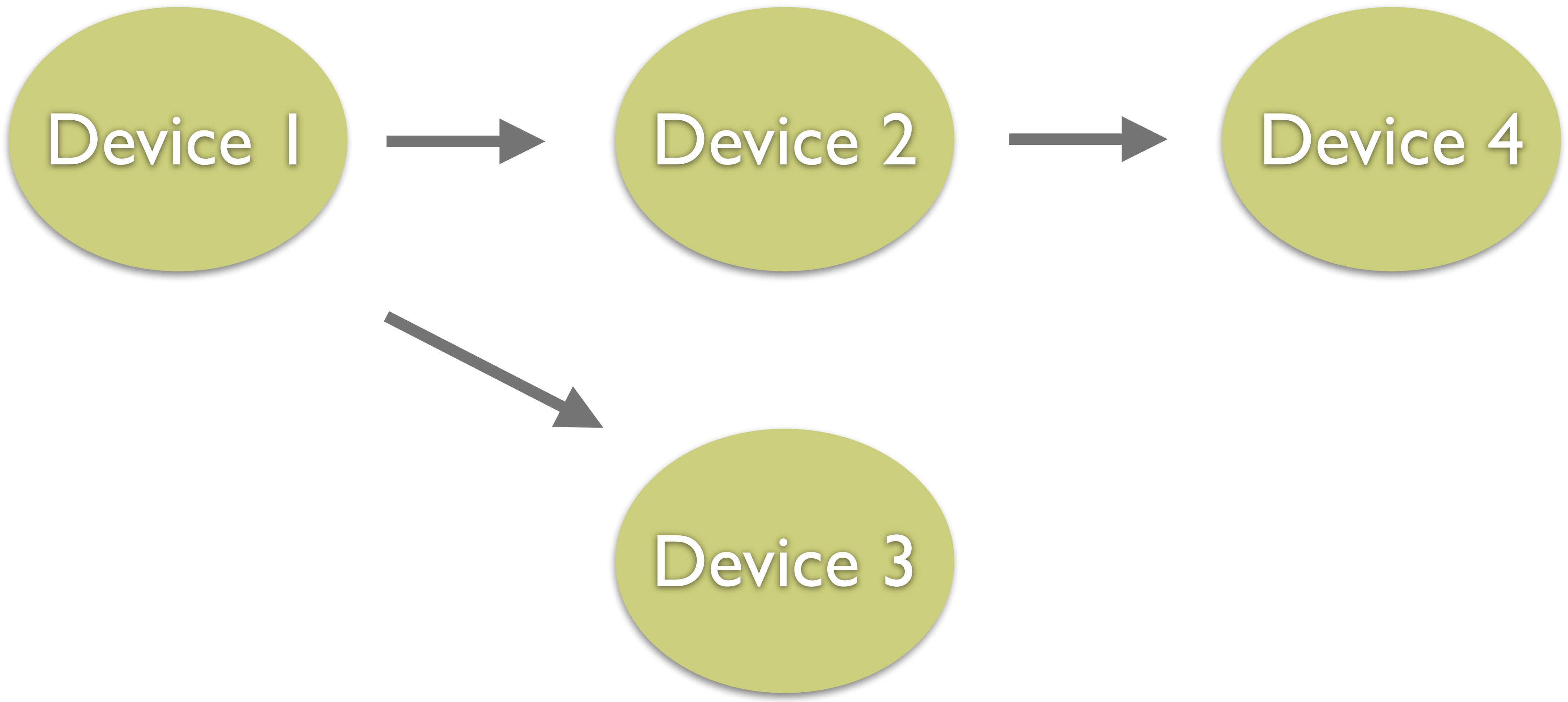
Transport Layer: ALFA / FairMQ

- Joint collaboration with FAIR and GSI

ALFA / FairMQ: General Idea

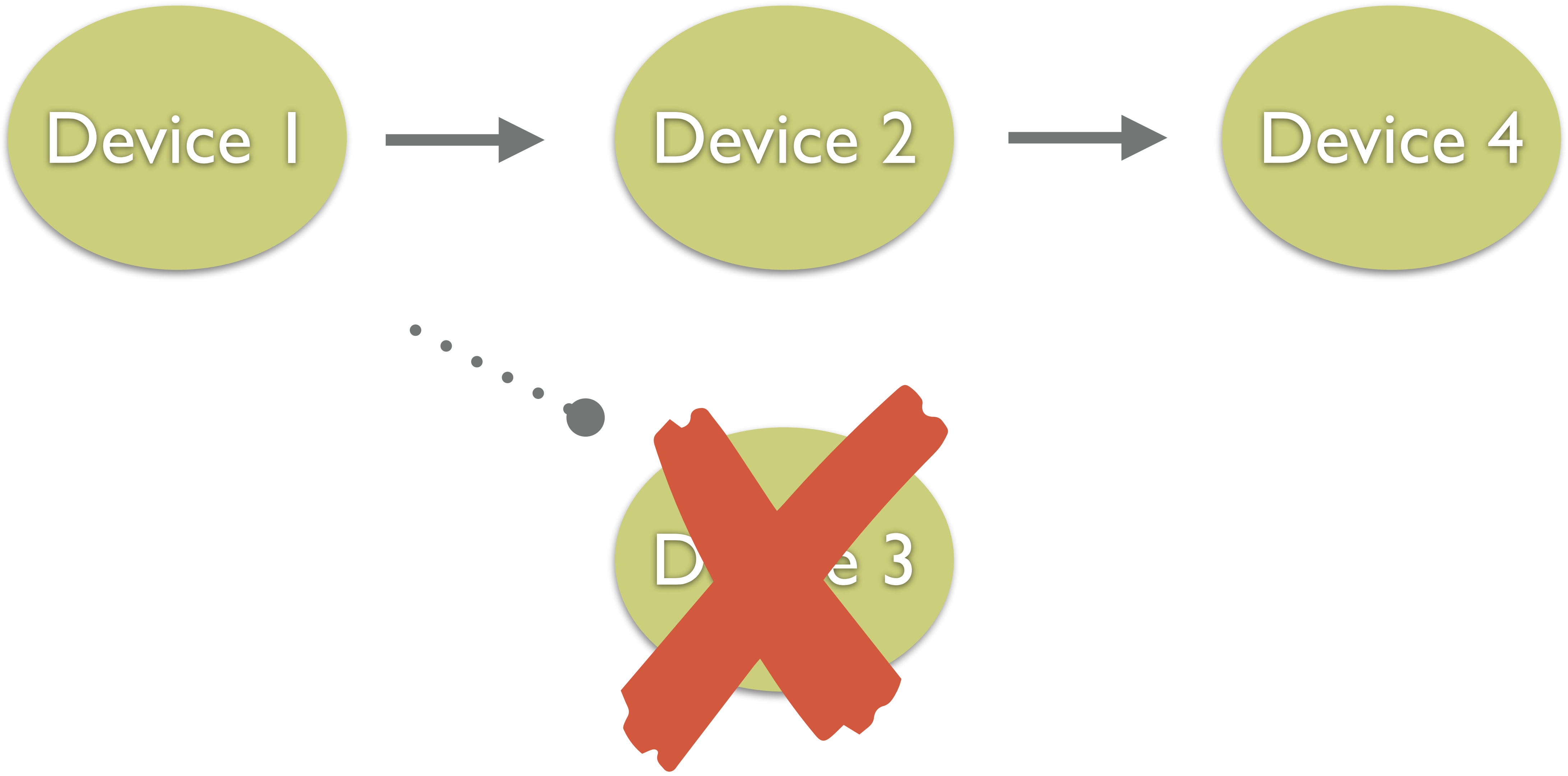
Data processing happens in **separate processes**, called **devices**.

ALFA / FairMQ: General Idea



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

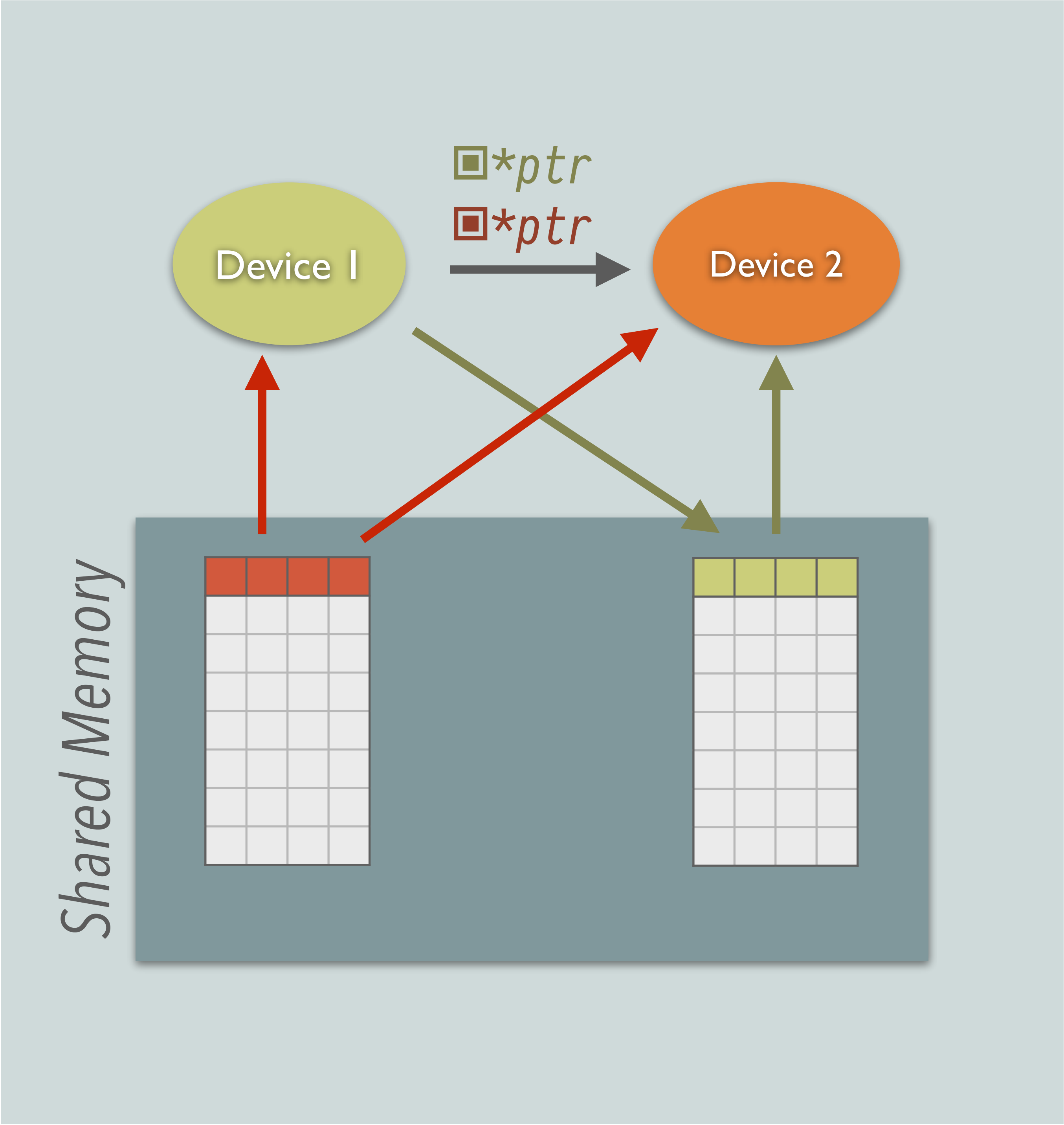
ALFA / FairMQ: General Idea



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

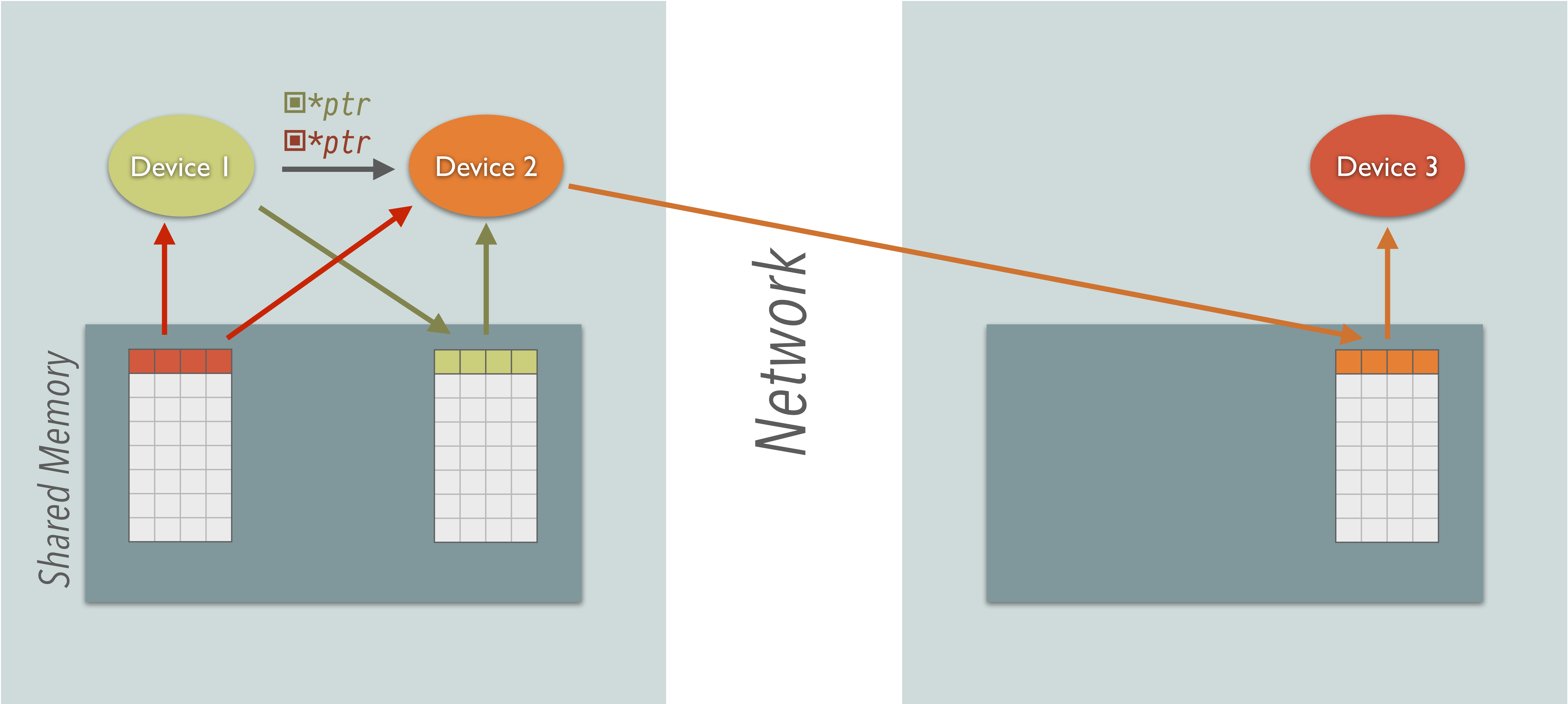
ALFA / FairMQ: General Idea

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

O²: 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

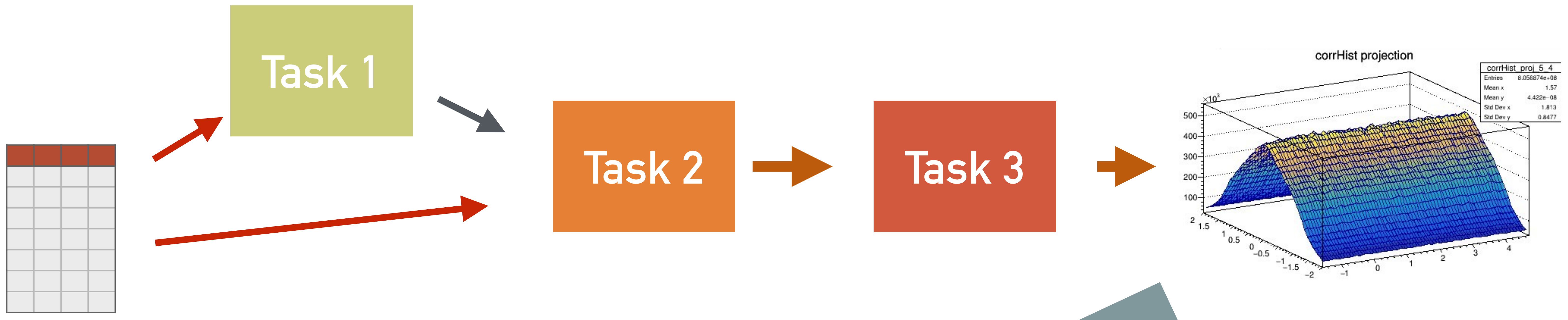
Message passing aware data model. Support for multiple backends:

- **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**.

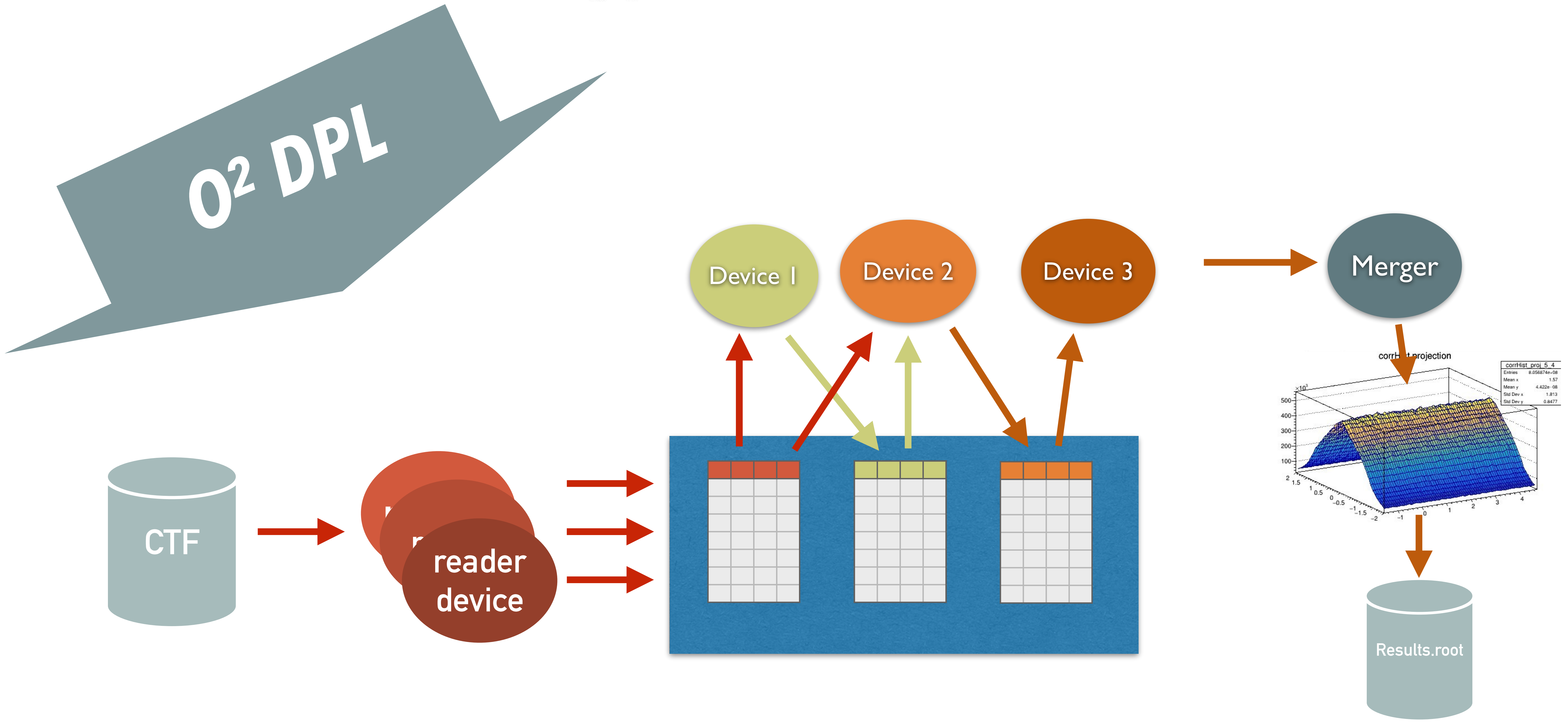
Transport Layer: ALFA / FairMQ[1]

- **Joint collaboration with FAIR and GSI**
- **Standalone processes (devices)** *for deployment flexibility & resilience*
- **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.

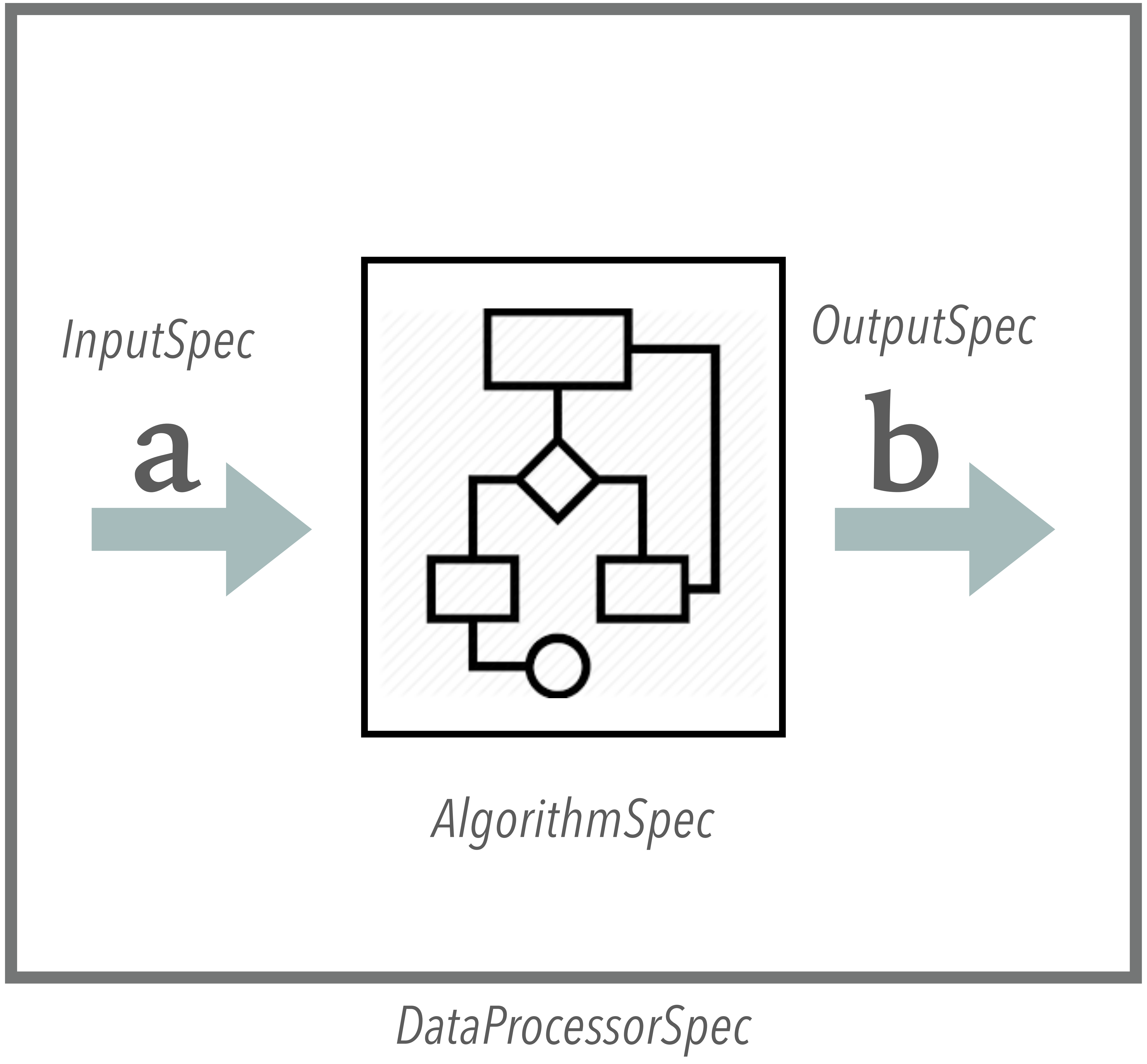


O2 Data Processing Layer (DPL) translates the implicit workflow(s) defined by physicists to an actual FairMQ topology of devices, injecting readers and merger devices, completing the topology and taking care of parallelism & rate limiting.

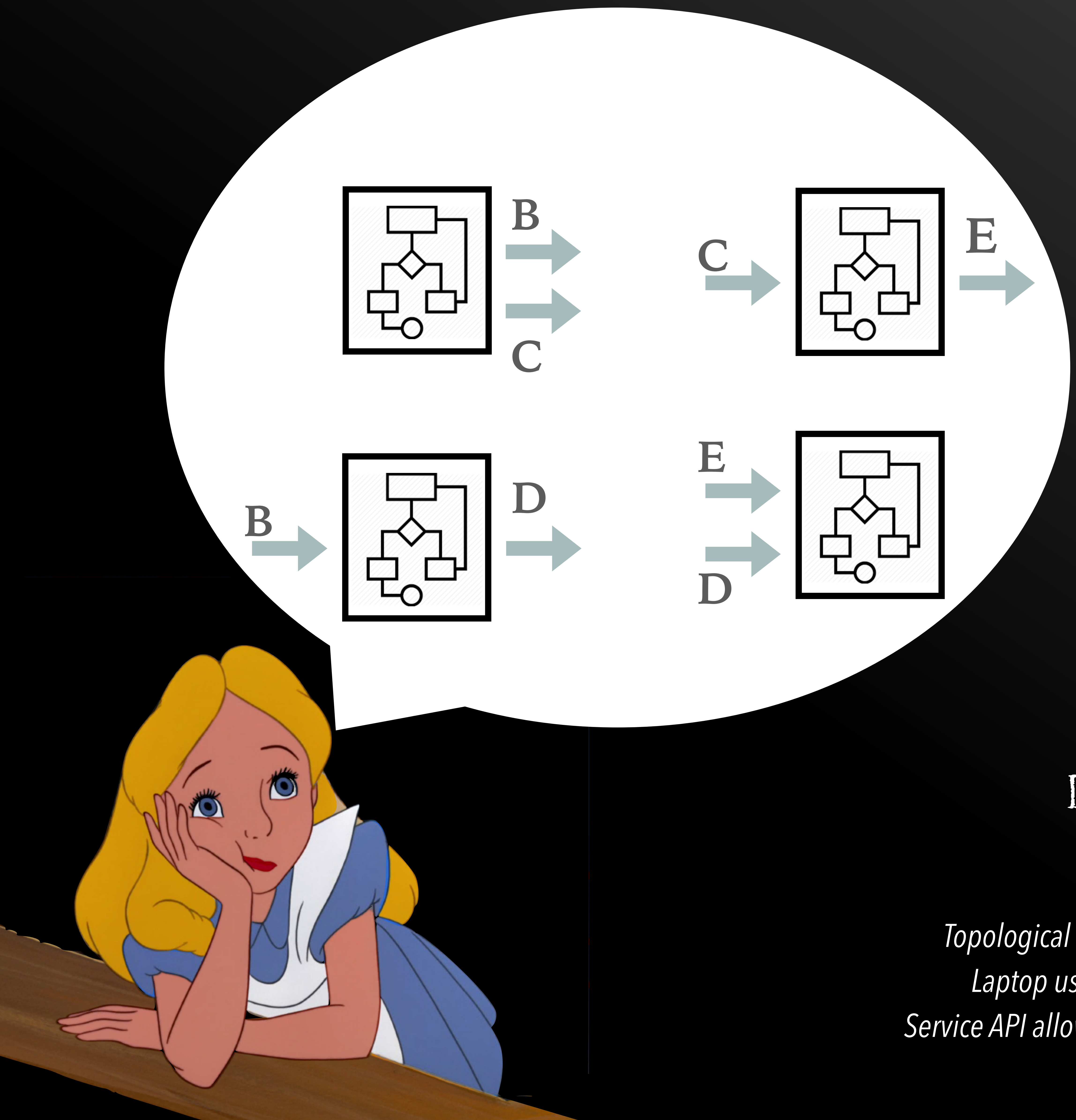
DATA PROCESSING LAYER: BUILDING BLOCK

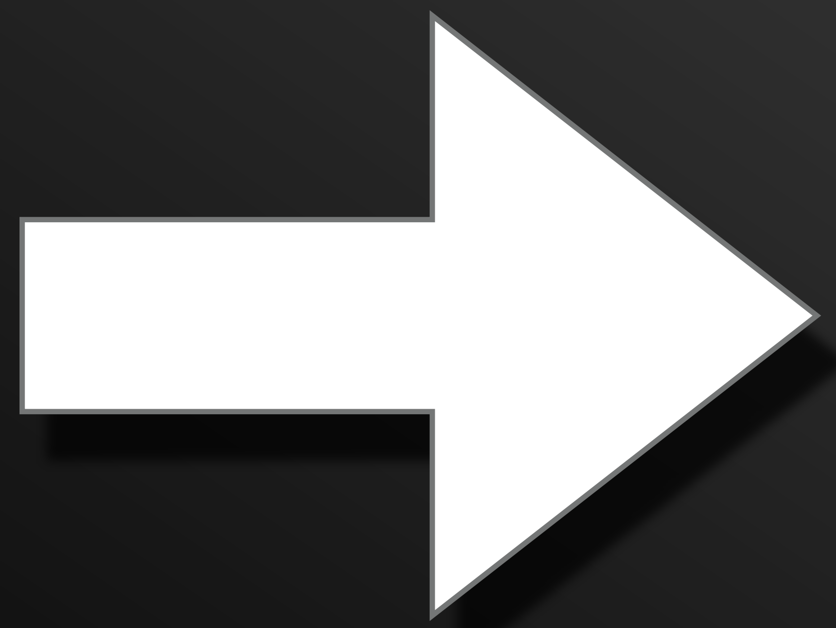
A DataProcessorSpec *defines a pipeline stage as a building block.*

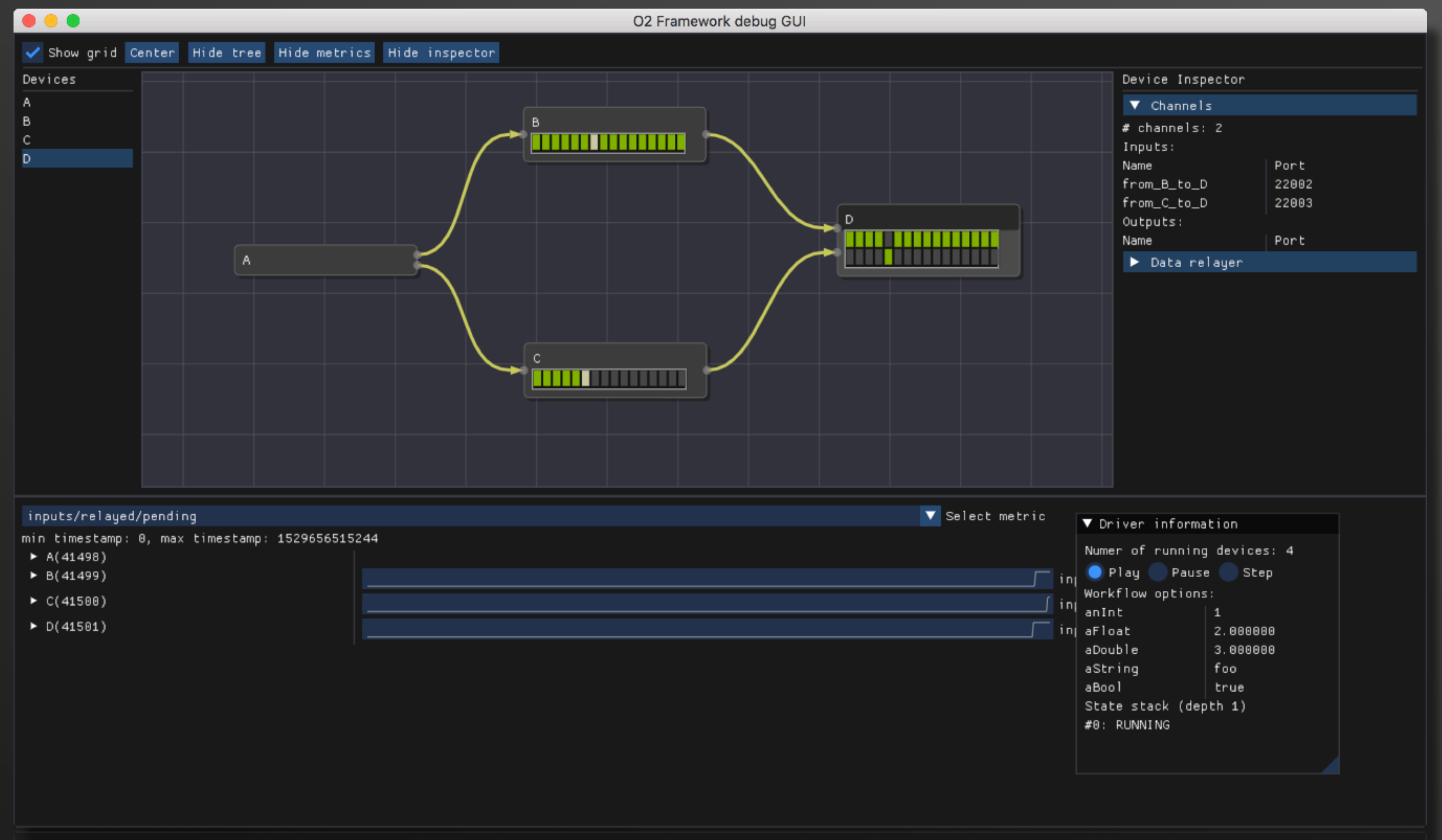
- *Specifies* ***inputs and outputs*** *in terms of the O^2 Data Model descriptors.*
- *Provide an implementation of how to act on the inputs to produce the output.*
- *Advanced user can express possible data or time parallelism opportunities.*



DATA PROCESSING LAYER: IMPLICIT TOPOLOGY







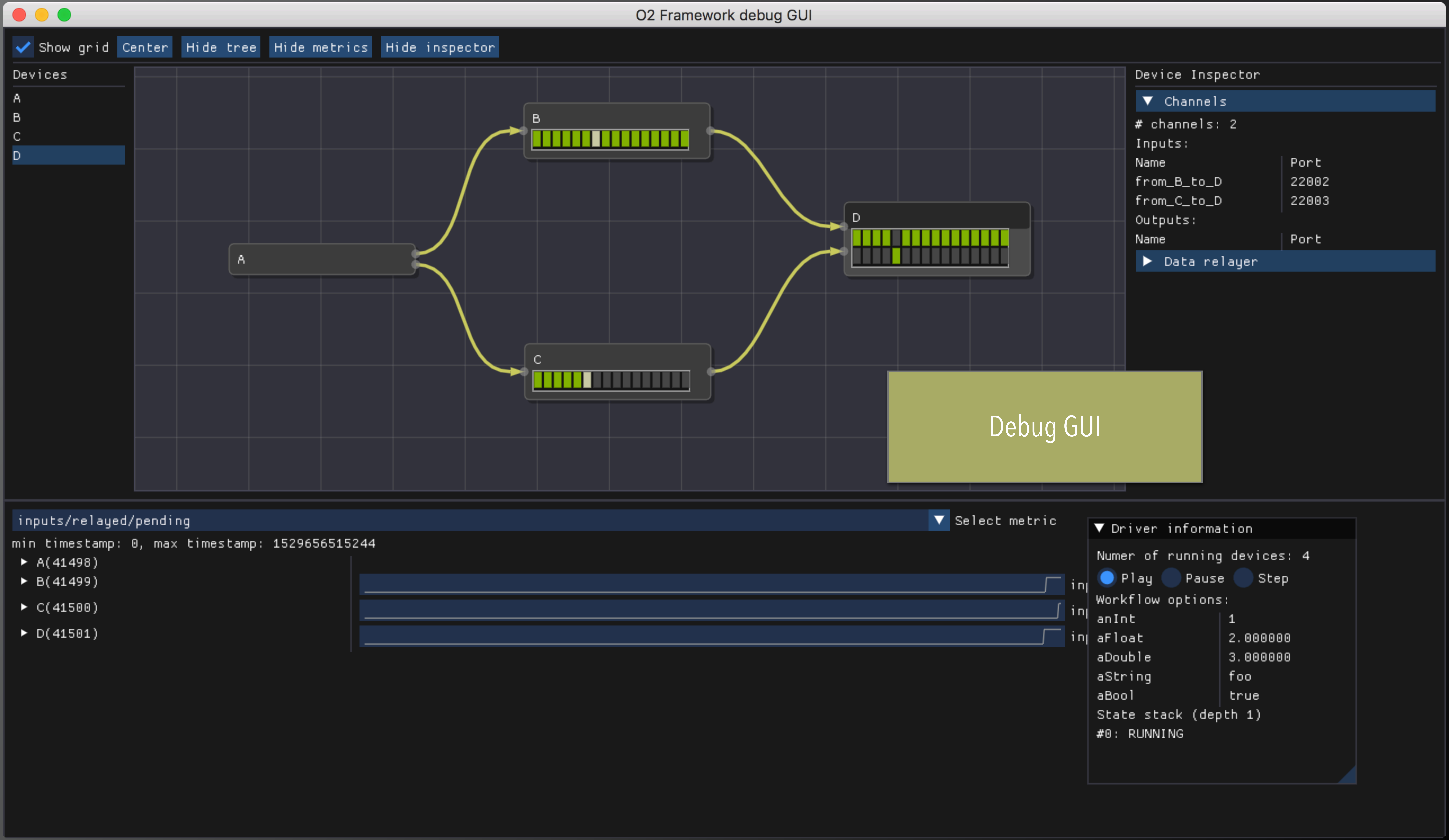
Data Processing Layer

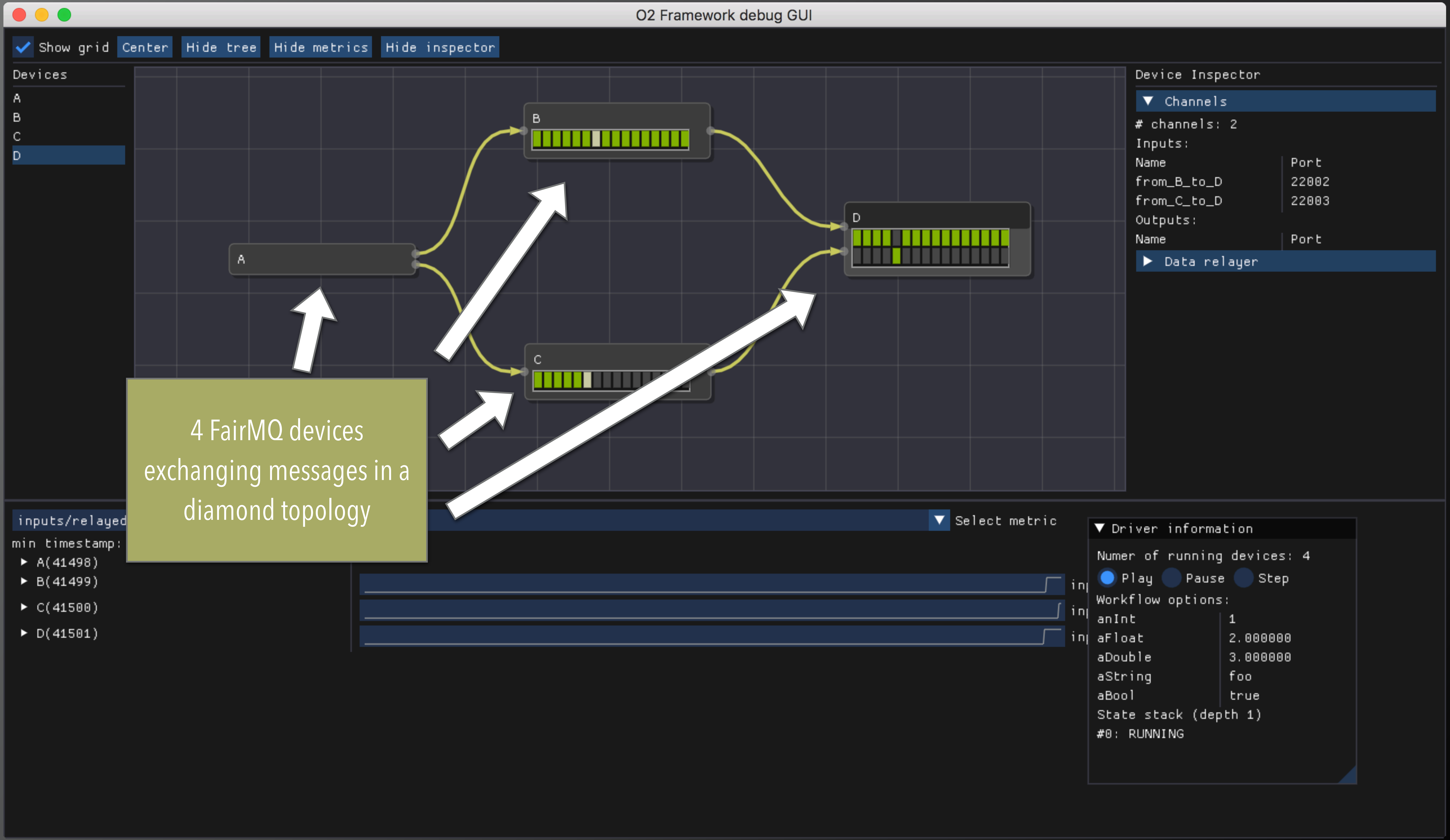
Topology is defined implicitly.

Topological sort ensures a viable dataflow is constructed (no cycles!).

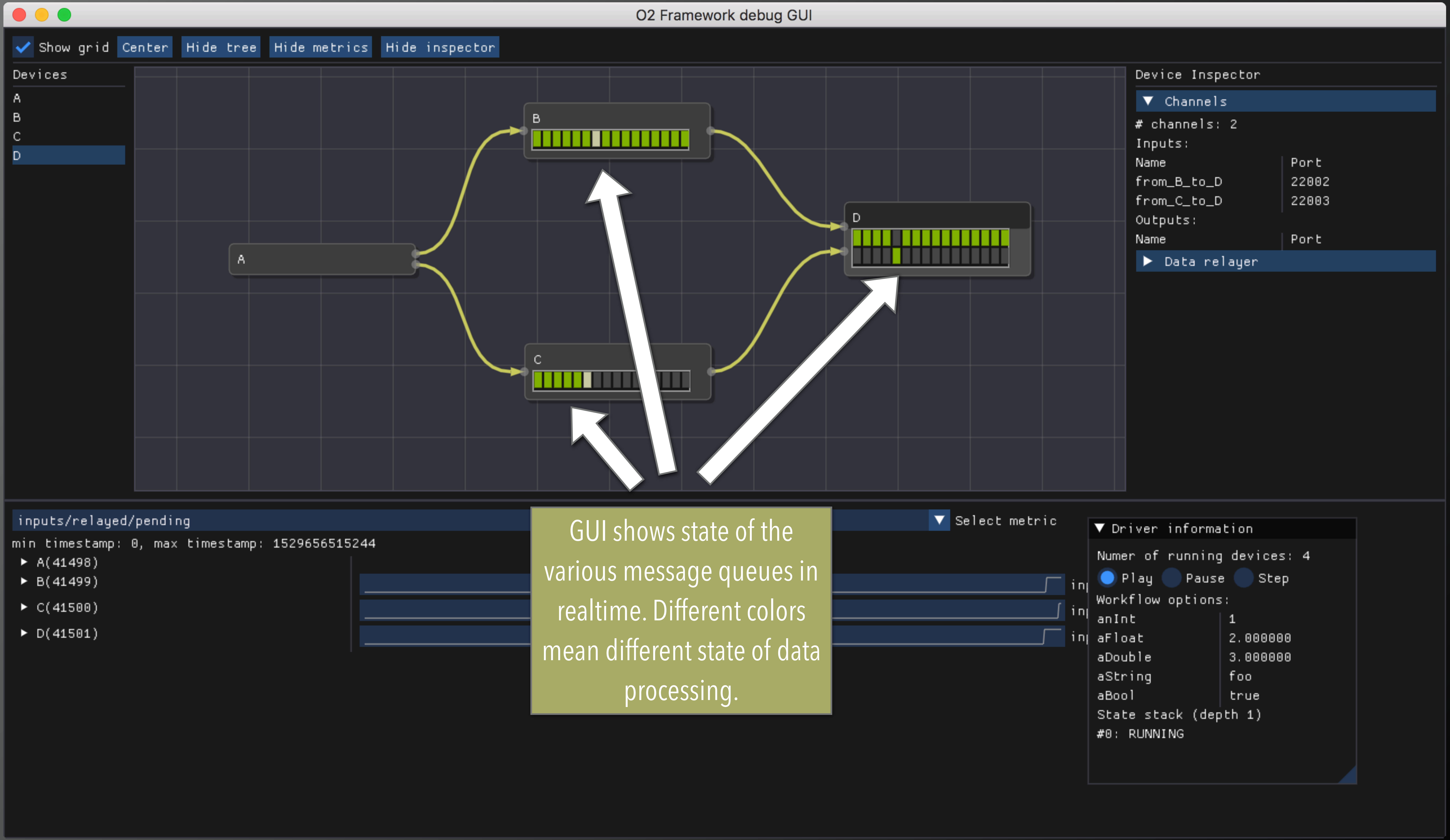
Laptop users gets immediate feedback through the debug GUI.

Service API allows integration with non data flow components (e.g. Control)

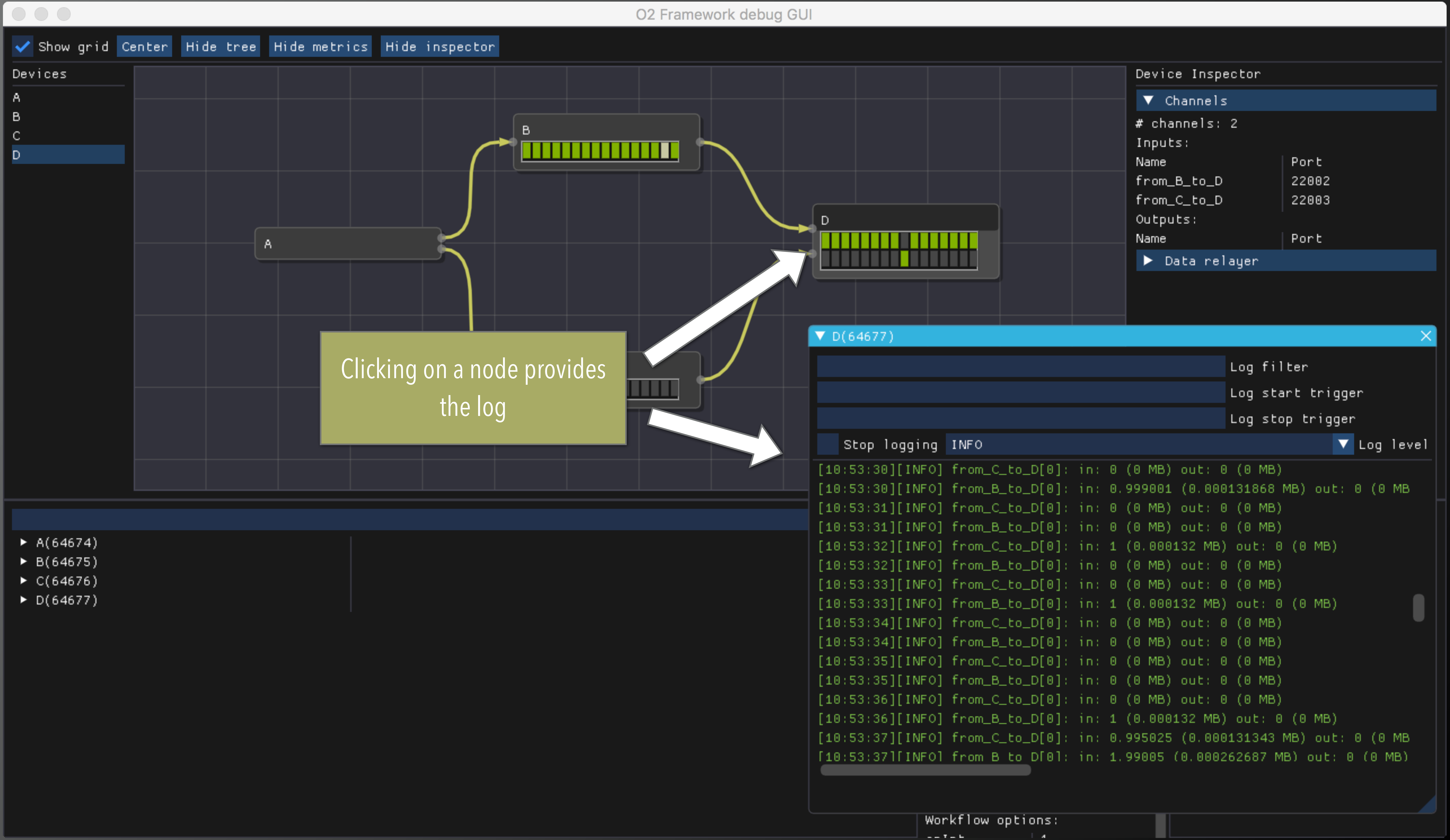

O2 Framework debug GUI
Show grid
Center
Hide tree
Hide metrics
Hide inspector
Devices
A
B
C
D
B
A
D
C
Device Inspector
Channels
channels: 2
Inputs:
Name
Port
from_B_to_D
22002
from_C_to_D
22003
Outputs:
Name
Port
Data relayer
Debug GUI
inputs/relayed/pending
Select metric
min timestamp: 0, max timestamp: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



4 FairMQ devices exchanging messages in a diamond topology



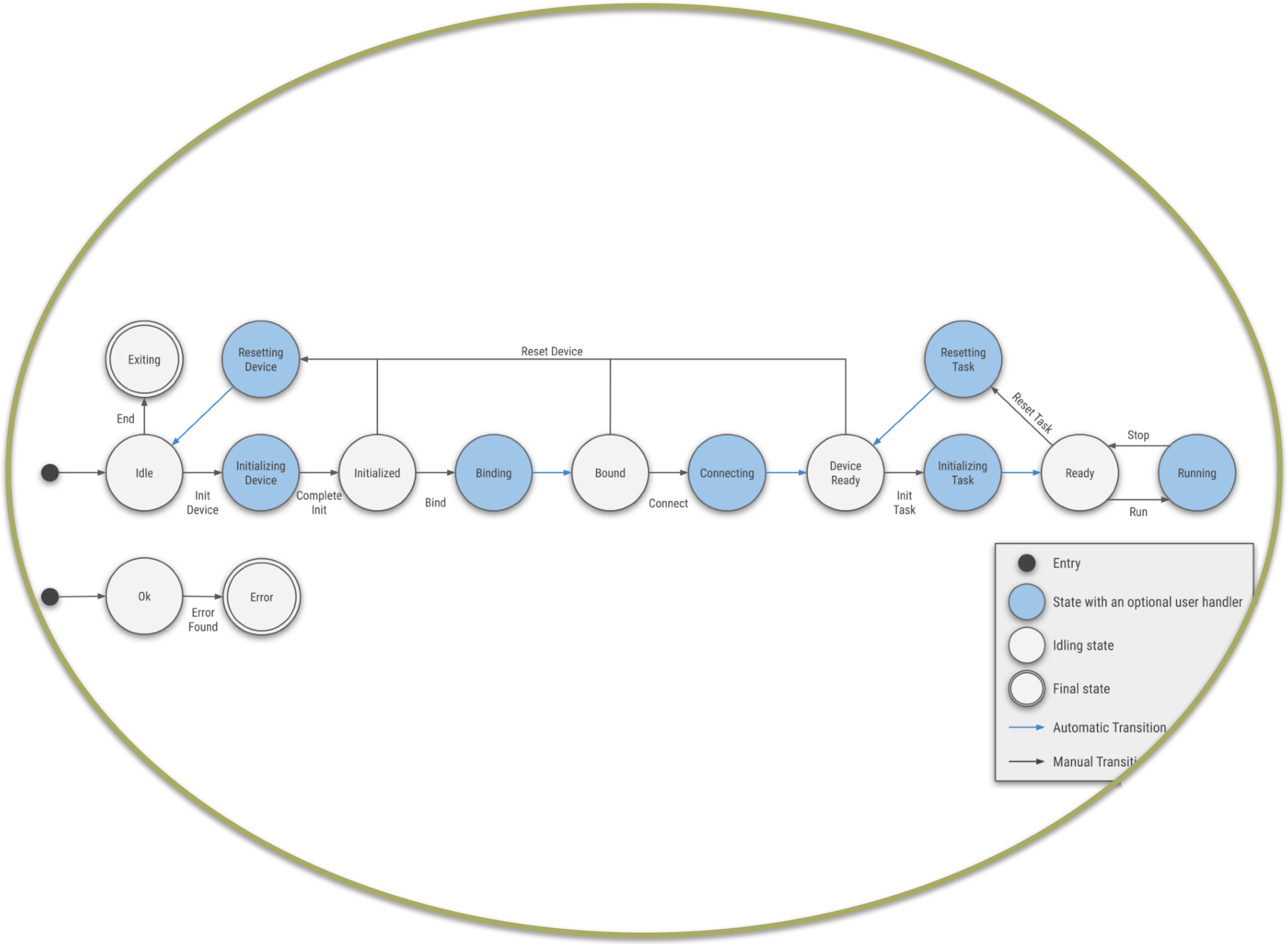
GUI shows state of the various message queues in realtime. Different colors mean different state of data processing.


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:

O²: Sync Reconstruction

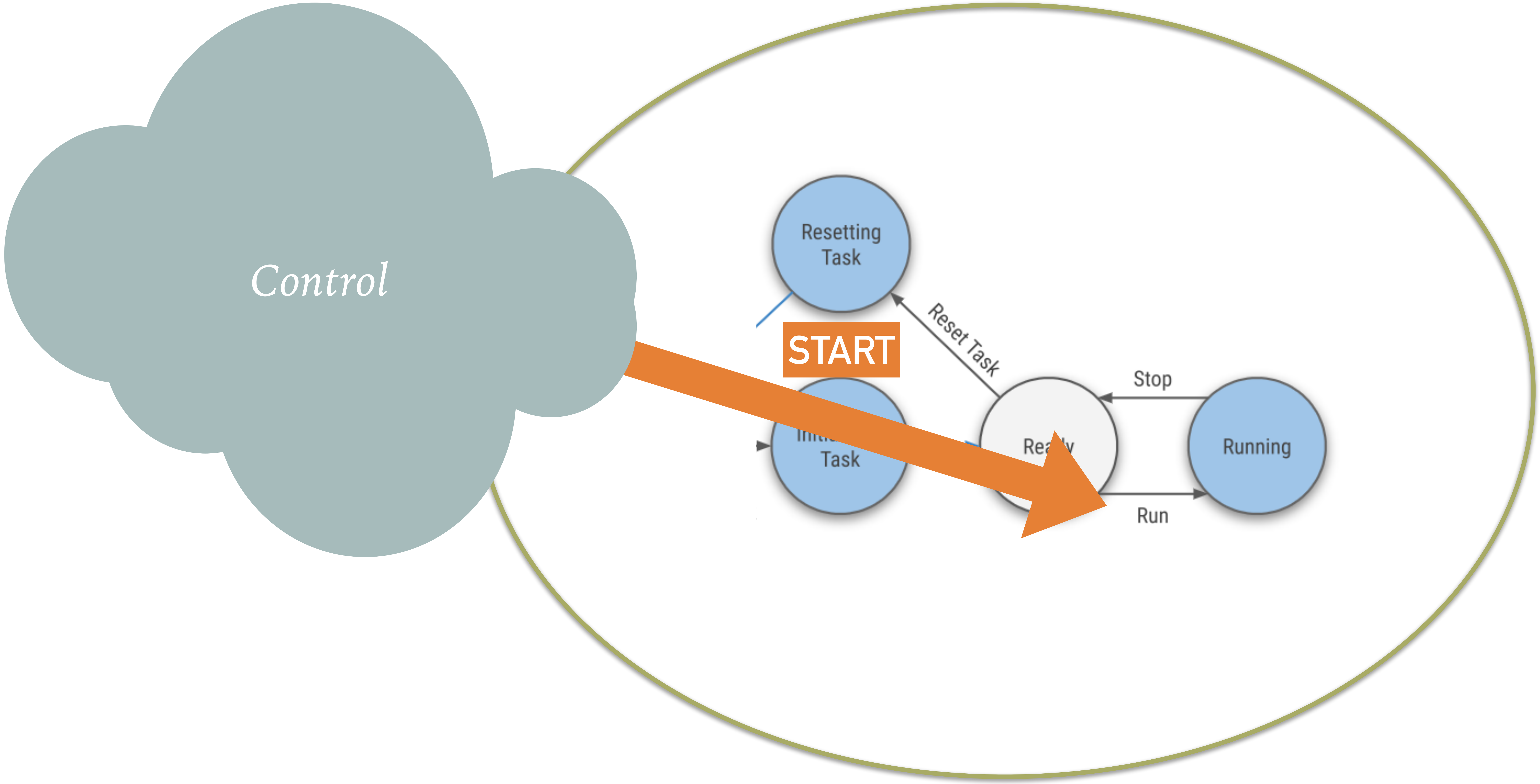
Takeaway message:
DPL allows building FairMQ topologies in an implicit way.

Data Processing Layer: Integration with the Control System



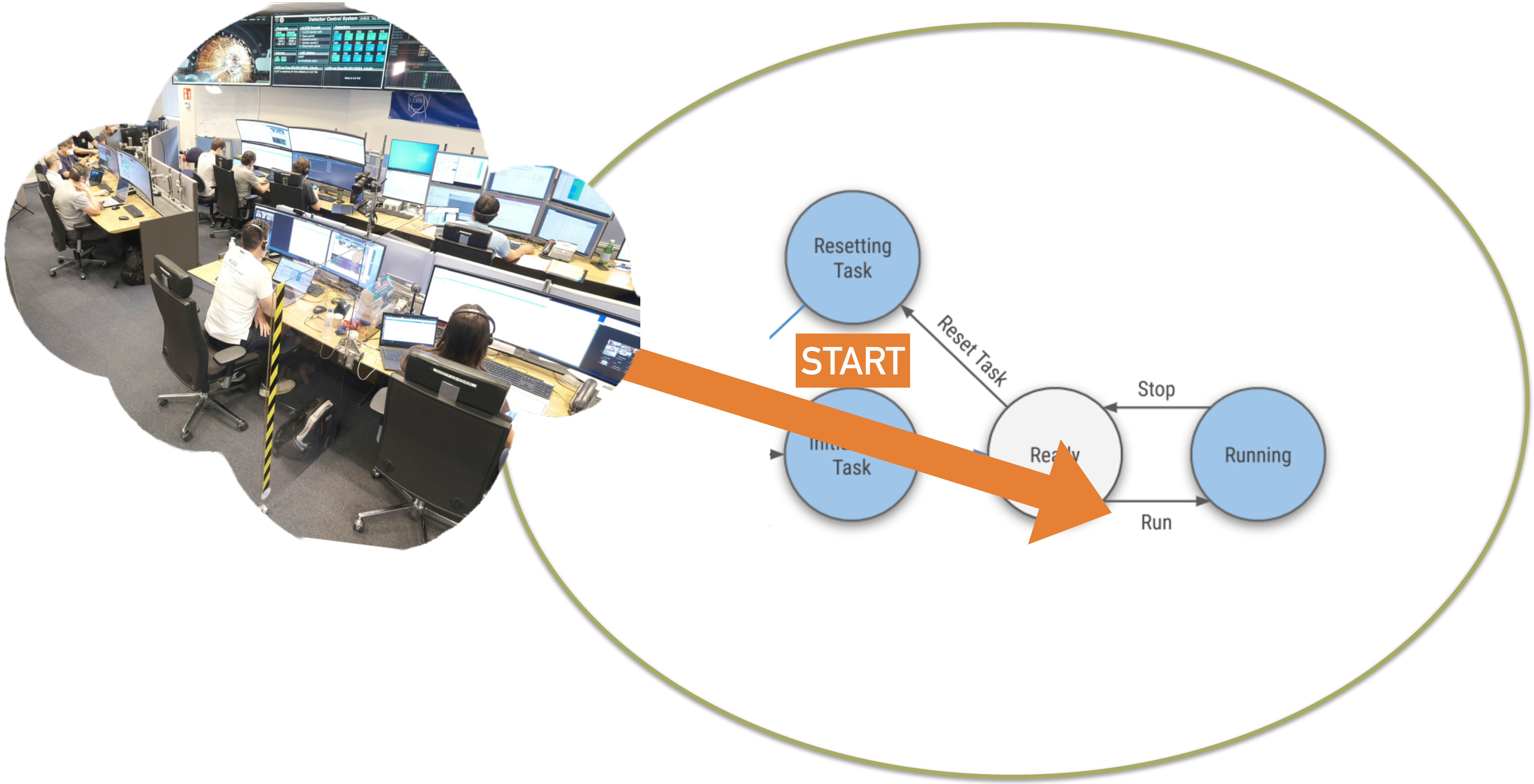
Each device runs a **finite state machine**.

Data Processing Layer: Integration with the Control System



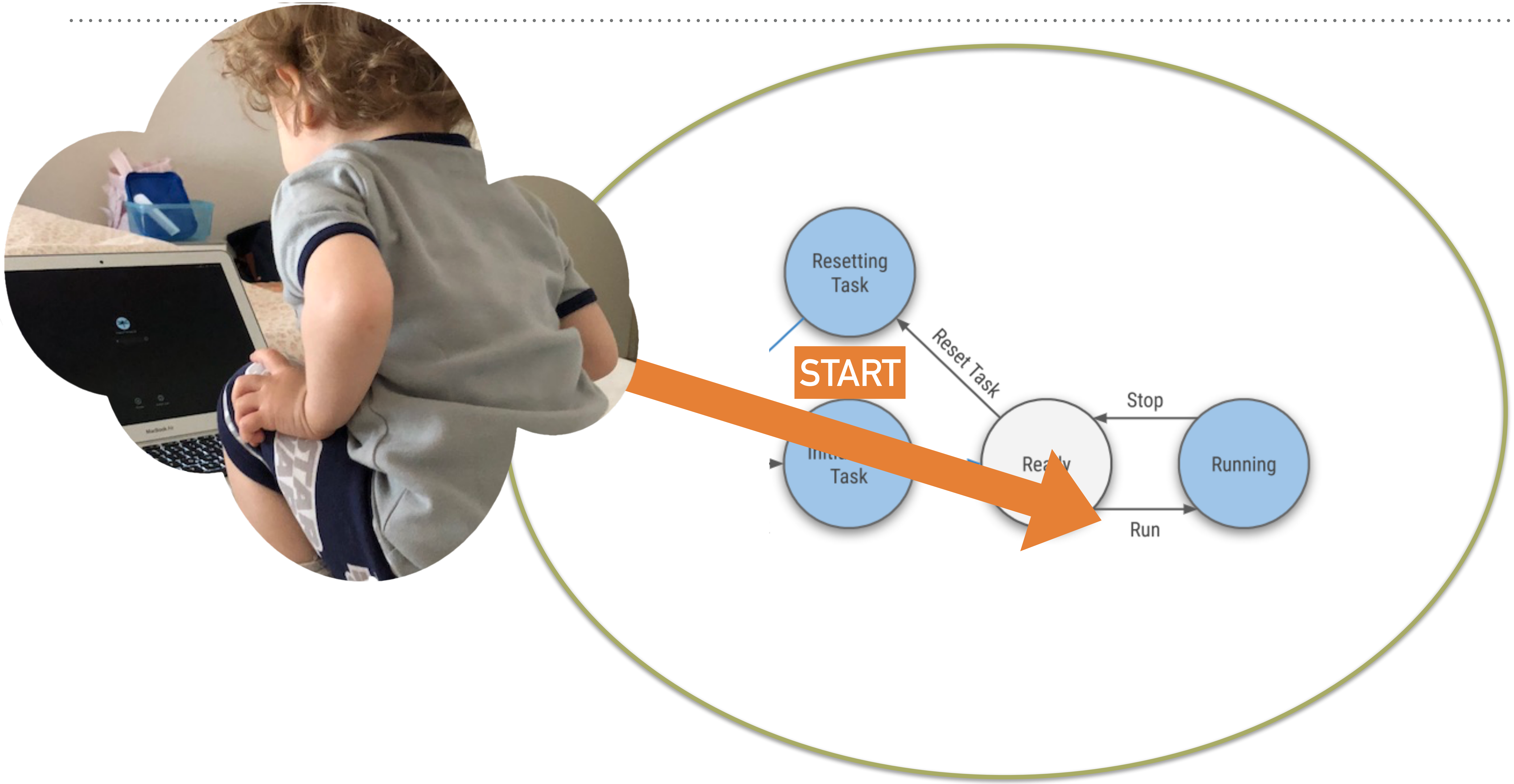
An **external control** is responsible to transition states.

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**...

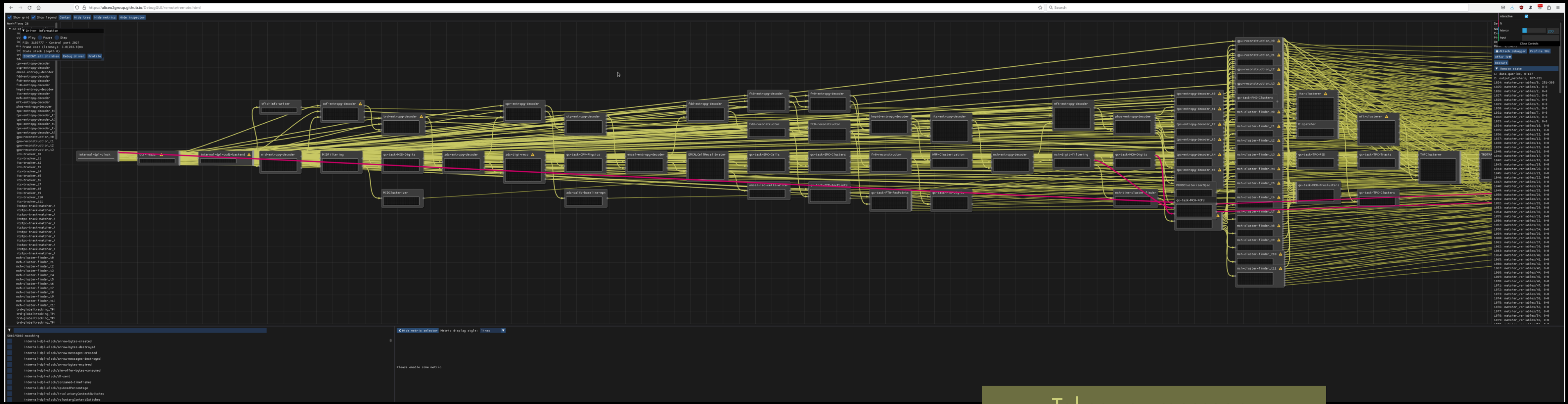
DATA PROCESSING LAYER: INTEGRATION WITH THE CONTROL SYSTEM



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

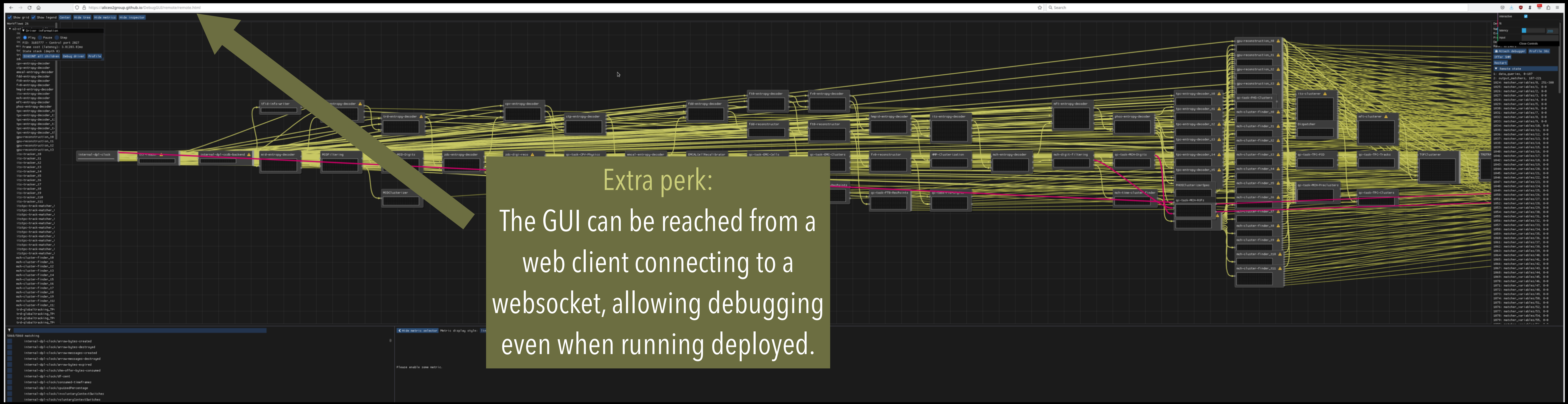
An **external control** is responsible to transition states. At P2 this is integrated with the **Experiment Control System**… while on the user laptop or on the grid we have a **DPL driver process** with such role.

O²: ASYNC RECONSTRUCTION

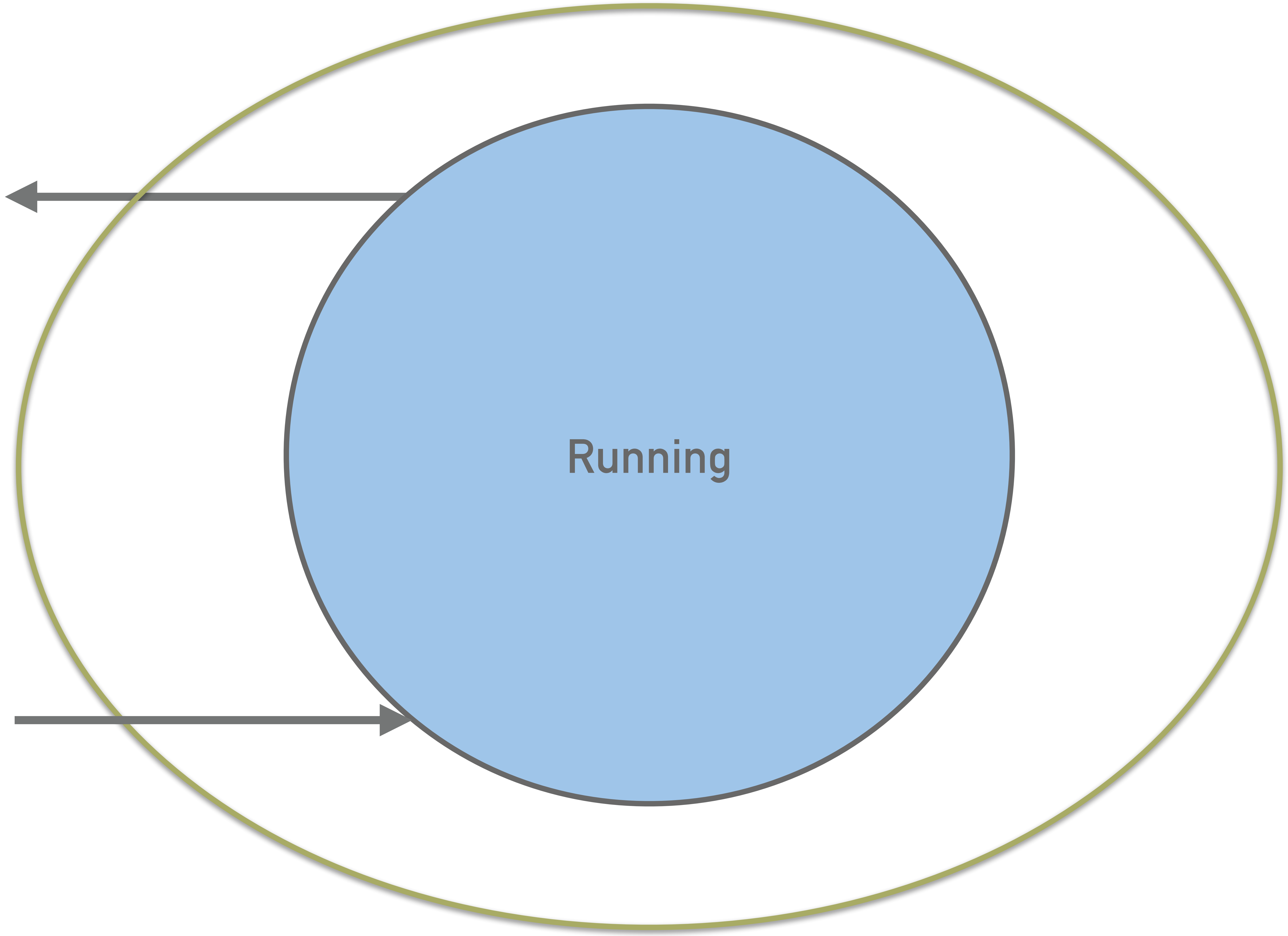


Takeaway message:
One single framework, from sync reconstruction to async and beyond.

O²: ASYNC RECONSTRUCTION

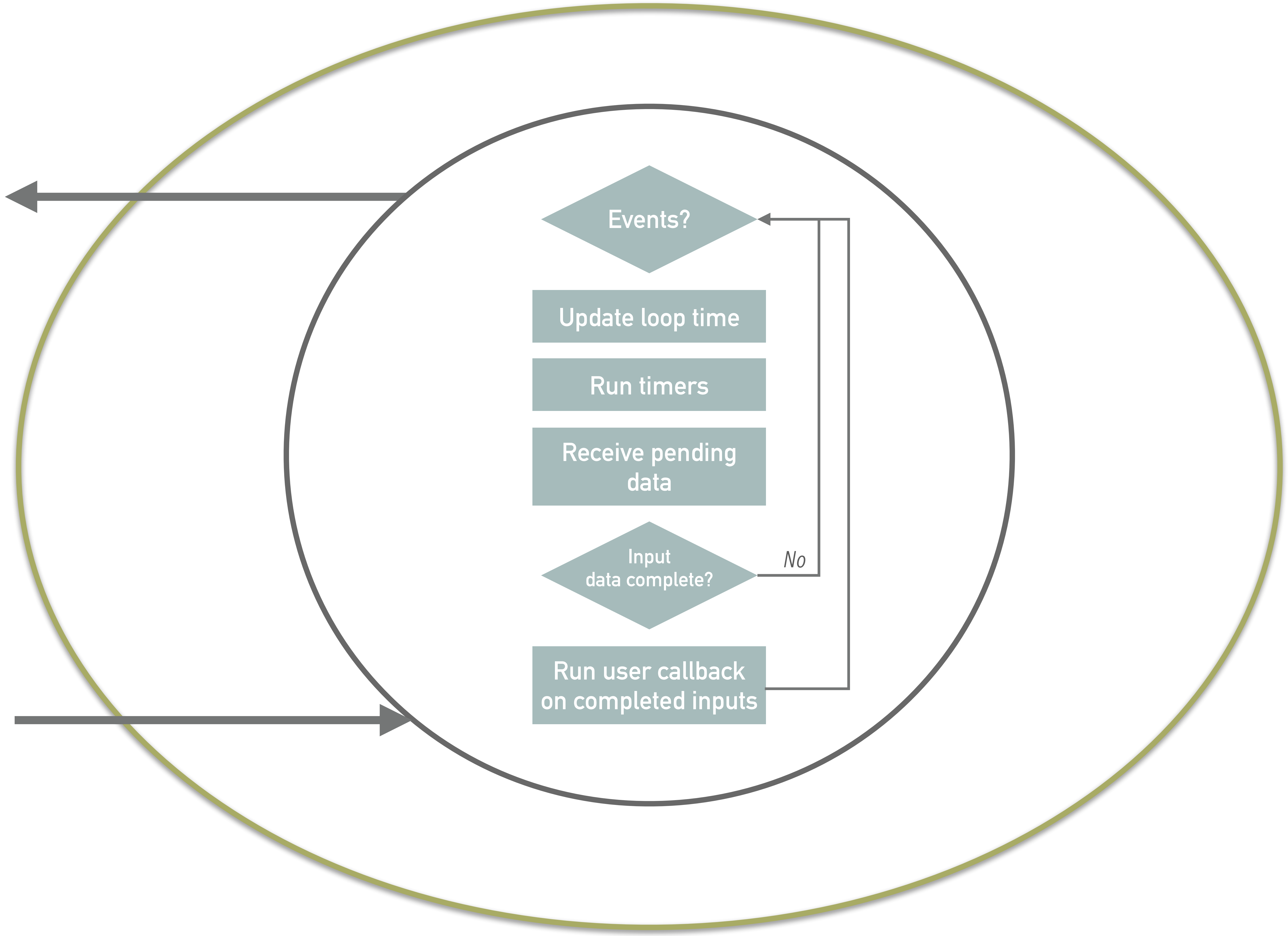


DATA PROCESSING LAYER: EVENT LOOP



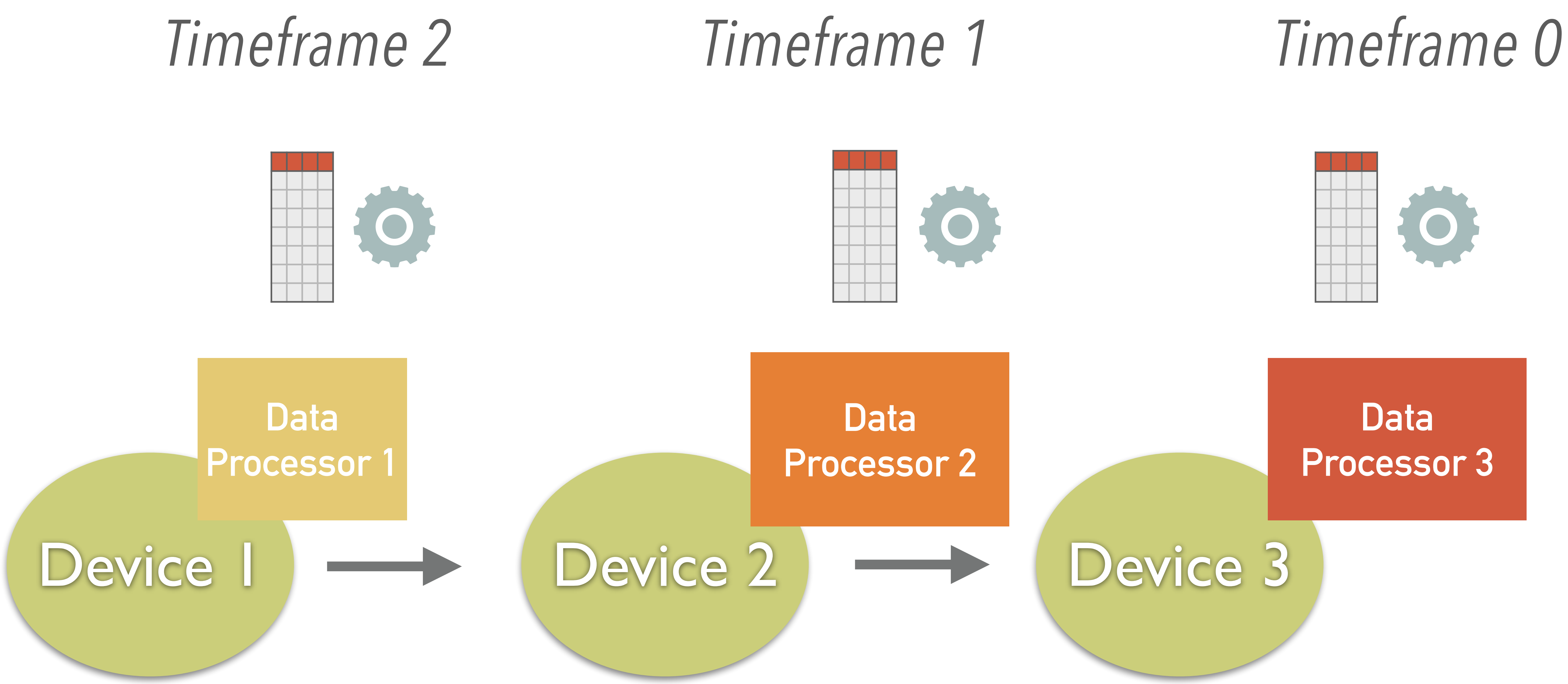
The Data Processing Layer (DPL) actually implements the Running state of a Device.

Data Processing Layer: Event Loop



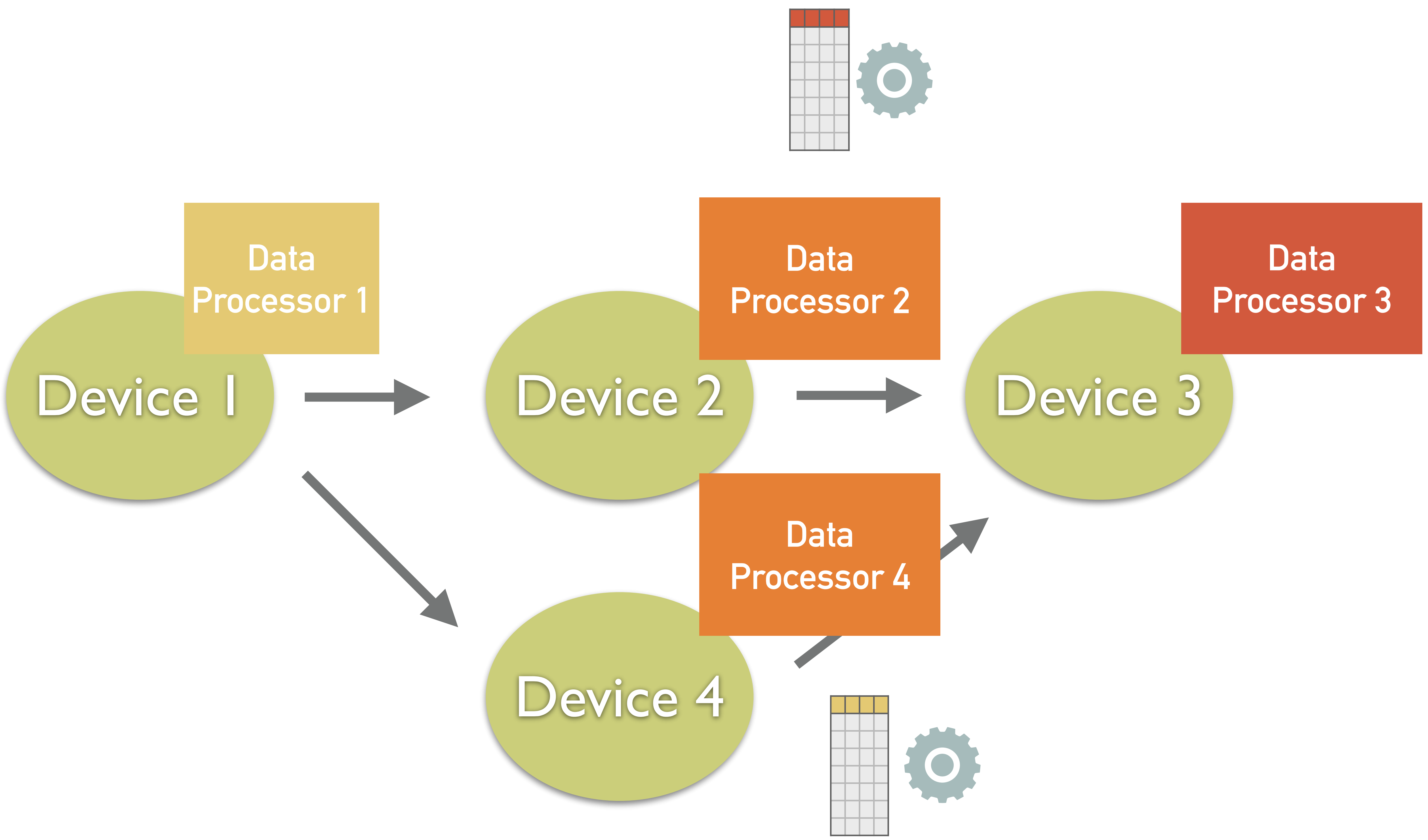
The (epoll / kqueue based) event loop only wakes up the device when there is something to do, e.g. to handle incoming data to process using the user provided code.

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**.

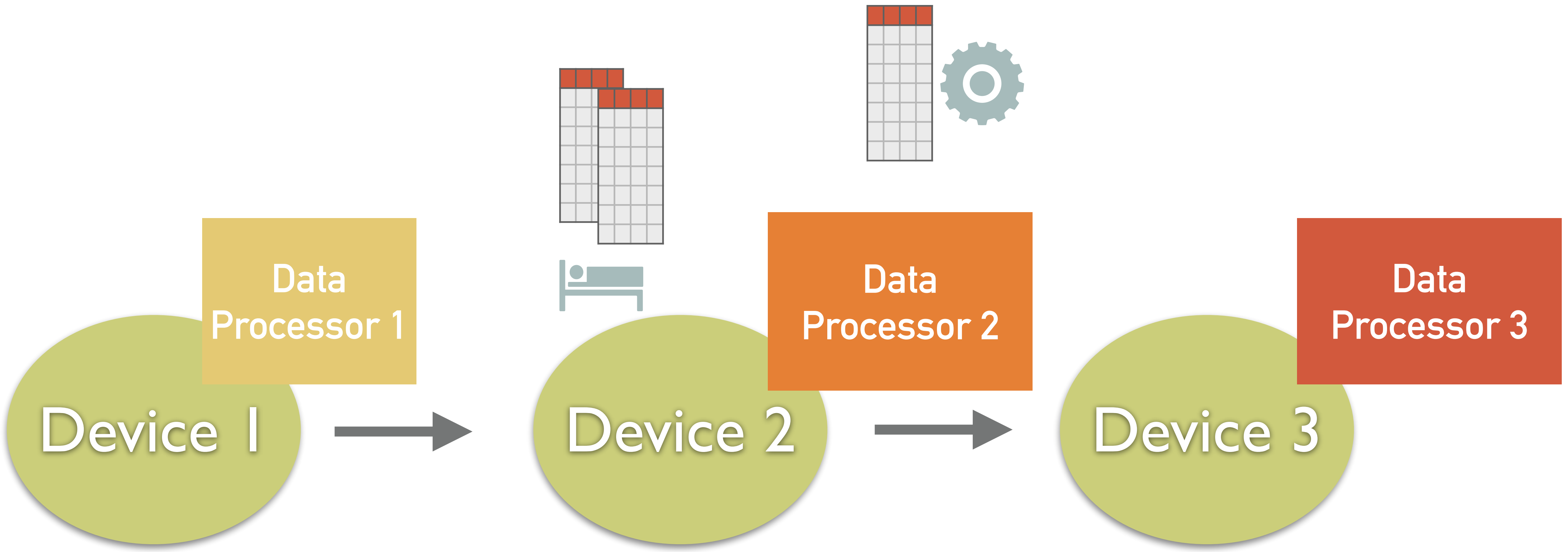
Data Processing Layer: Parallelism Opportunities



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

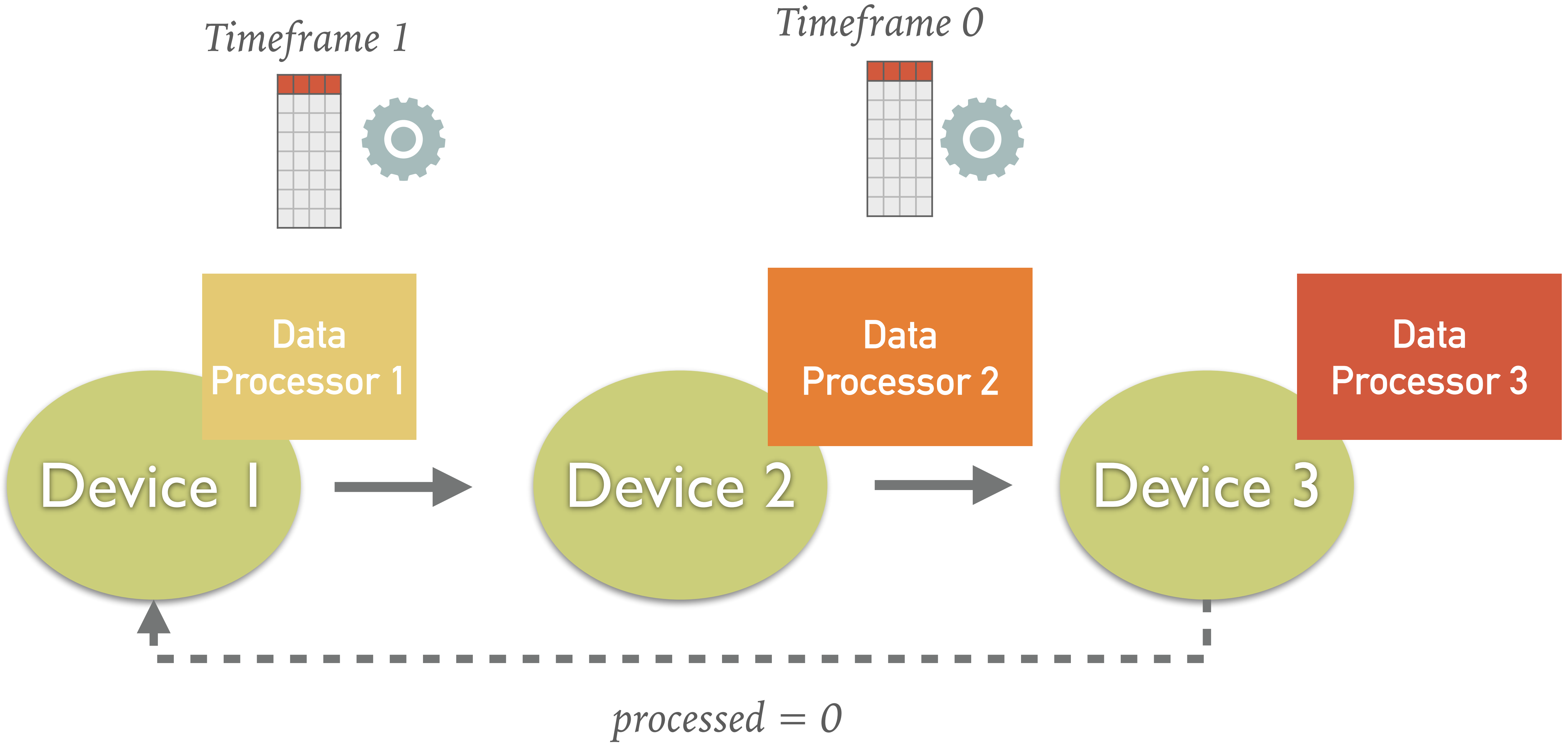
Vertical Parallelism.

Data Processing Layer: Rate Limiting



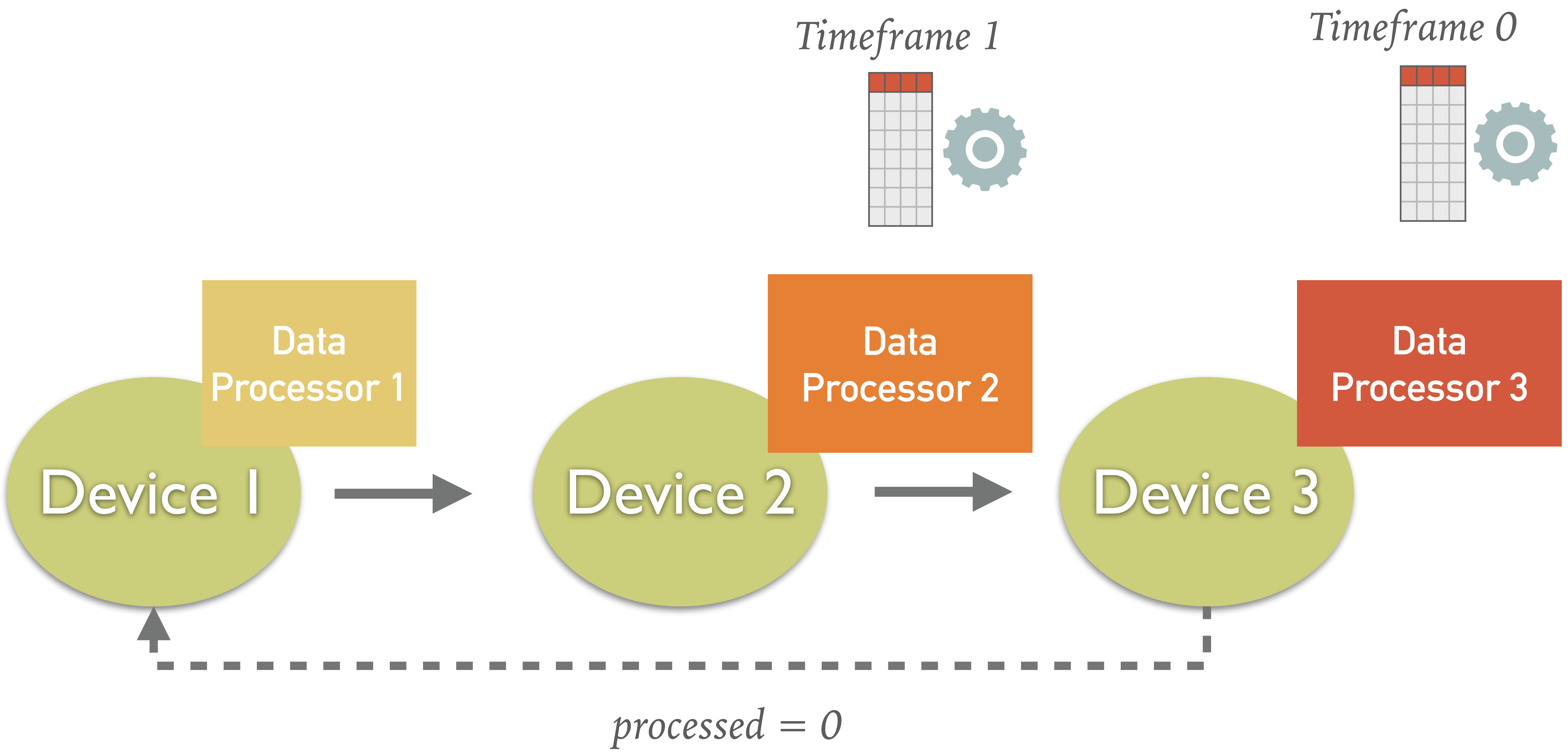
Without precautions, timeframes pile up in the input queue of the slowest device.

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.

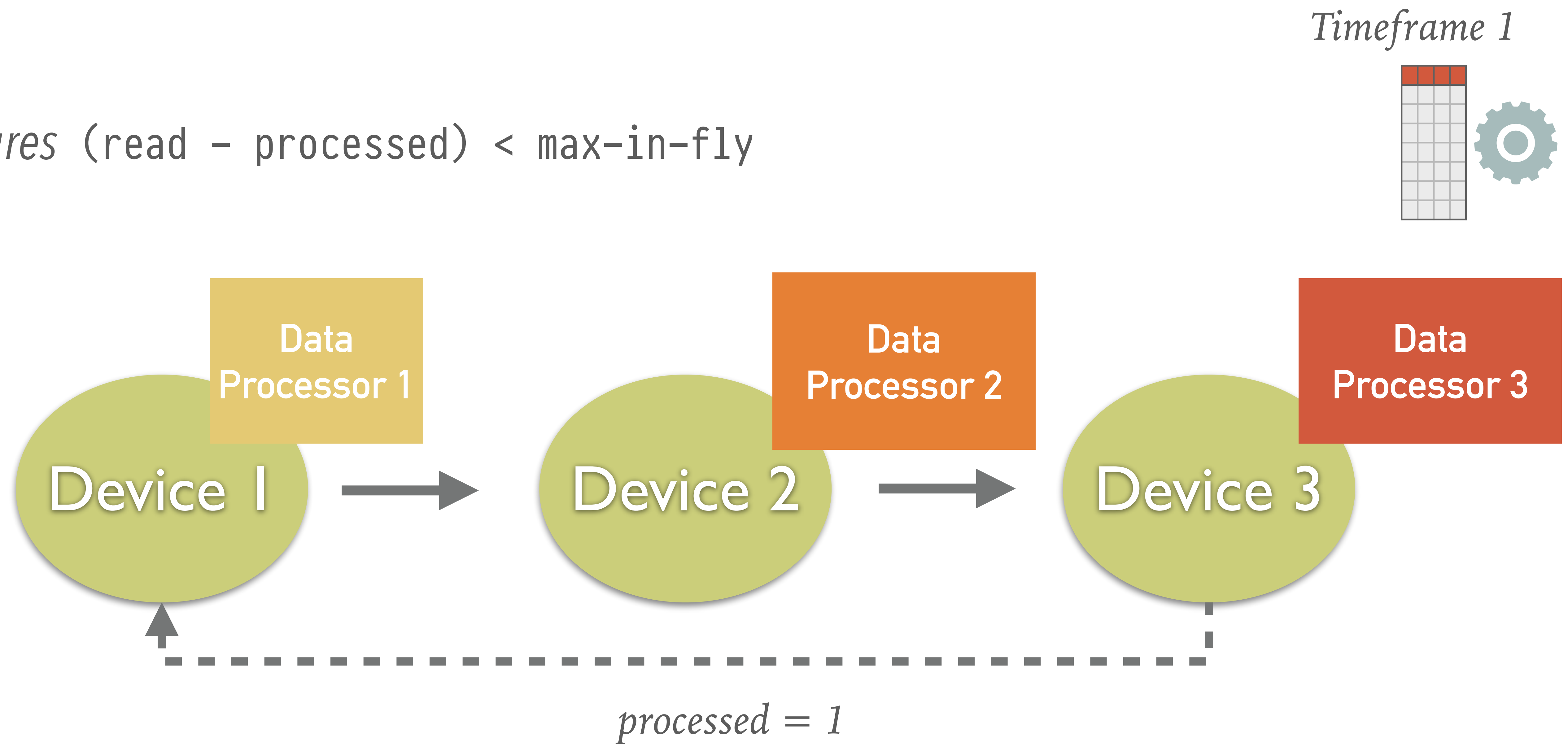
Data Processing Layer: Rate Limiting



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

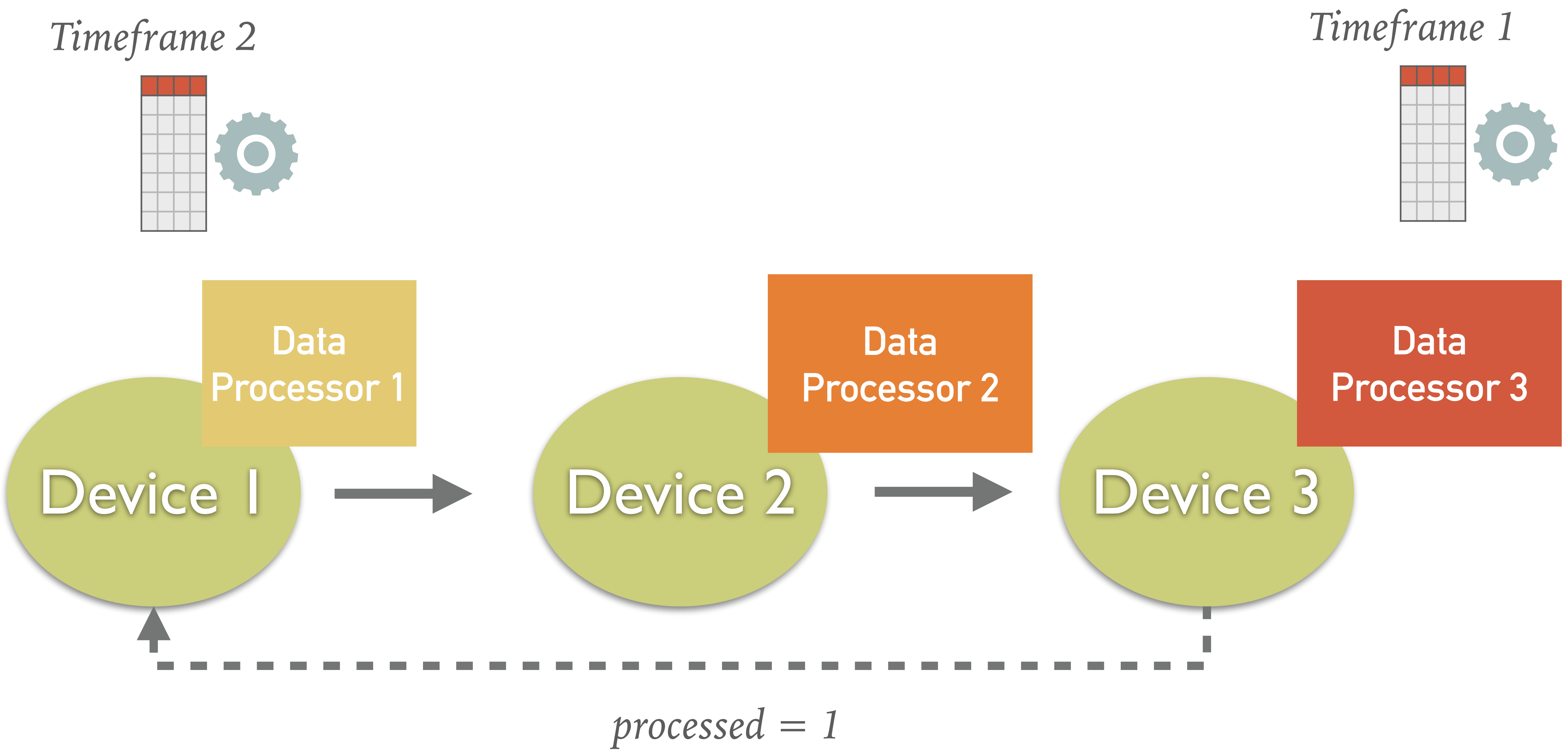
Data Processing Layer: Rate Limiting

First device ensures `(read - processed) < max-in-fly`



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

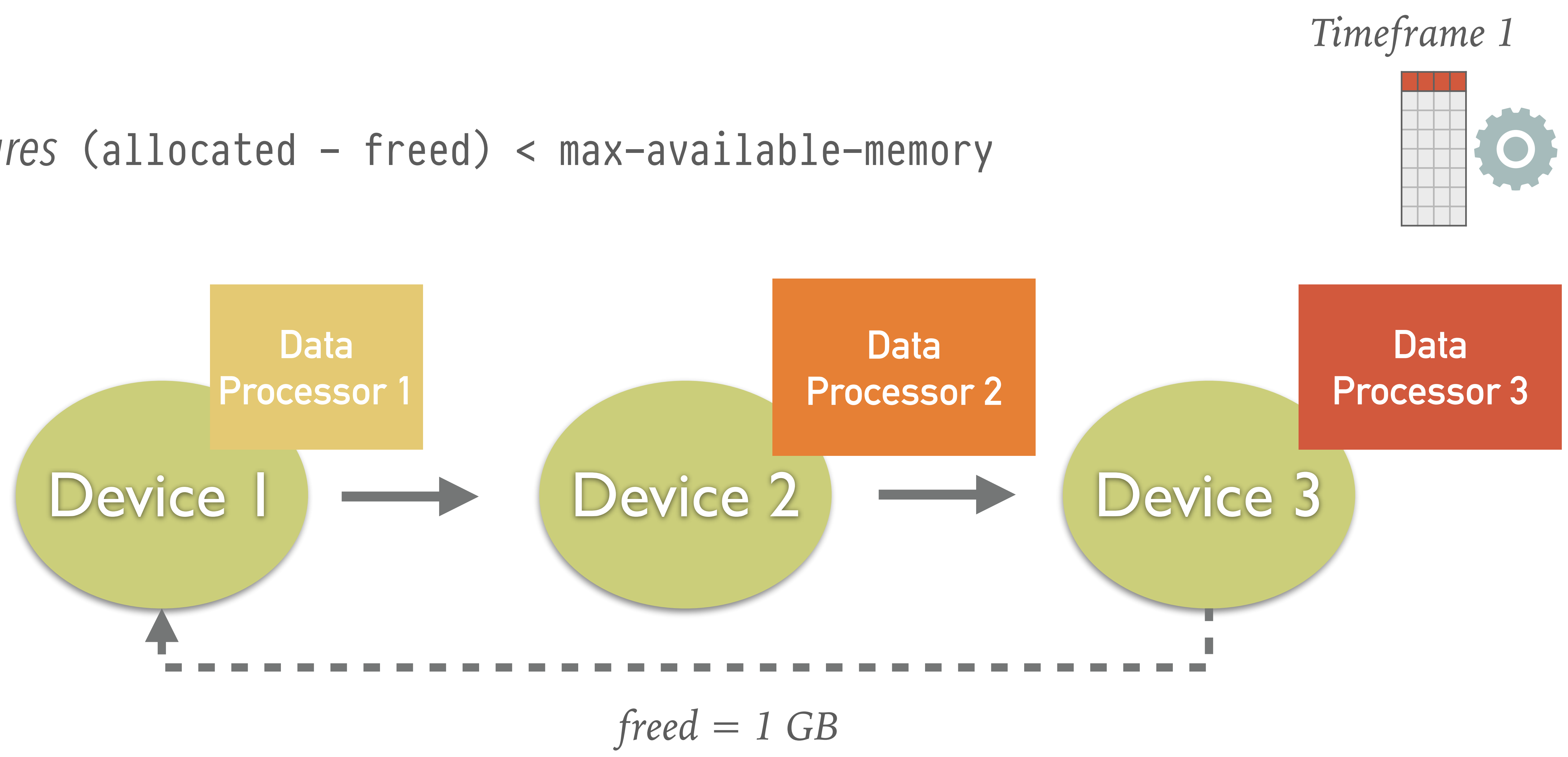
Data Processing Layer: Rate Limiting



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

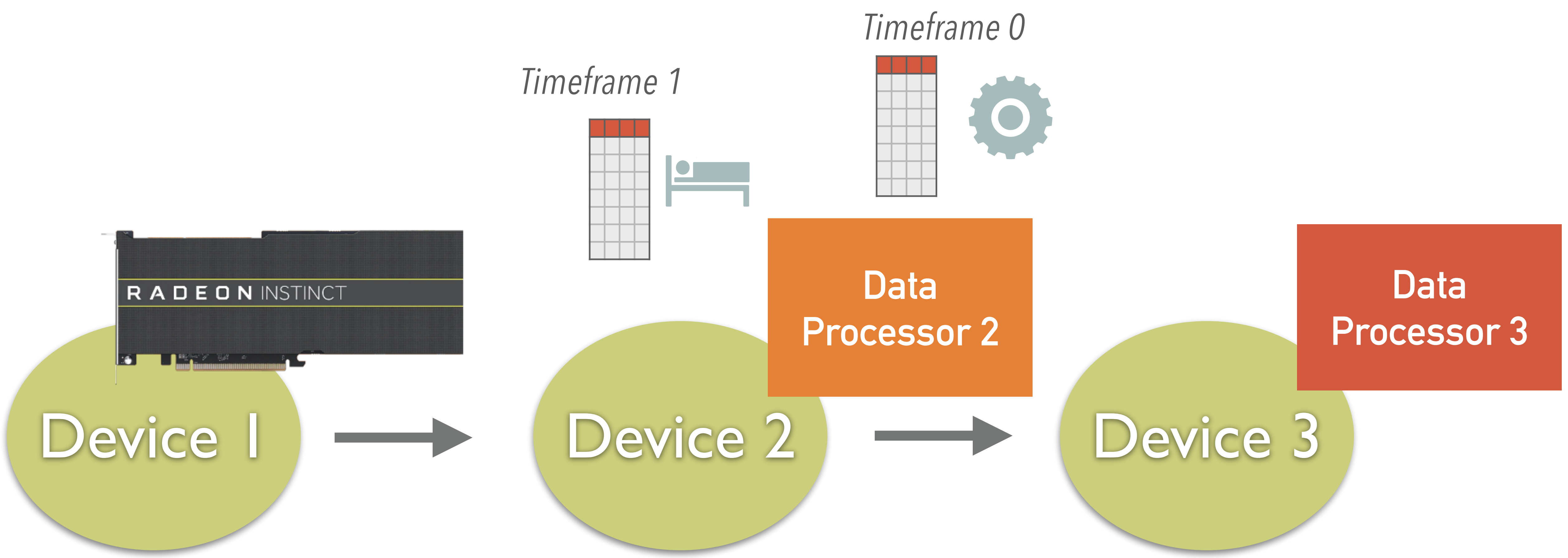
Data Processing Layer: Rate Limiting

First device ensures `(allocated - freed) < max-available-memory`



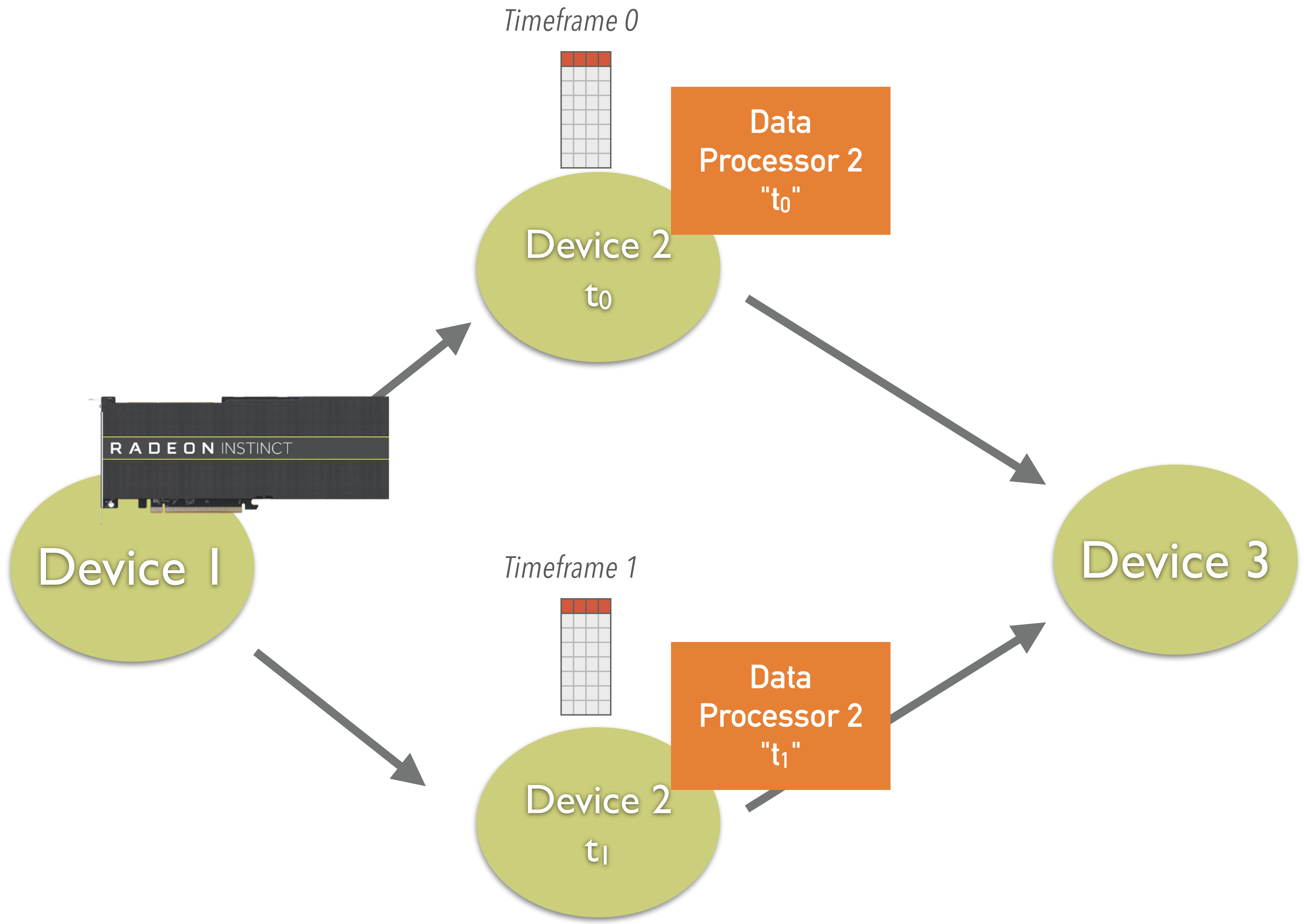
Besides the number of timeframes, we have the possibility to rate limit based on other quantities, e.g. available shared memory.

Data Processing Layer: Pipelining



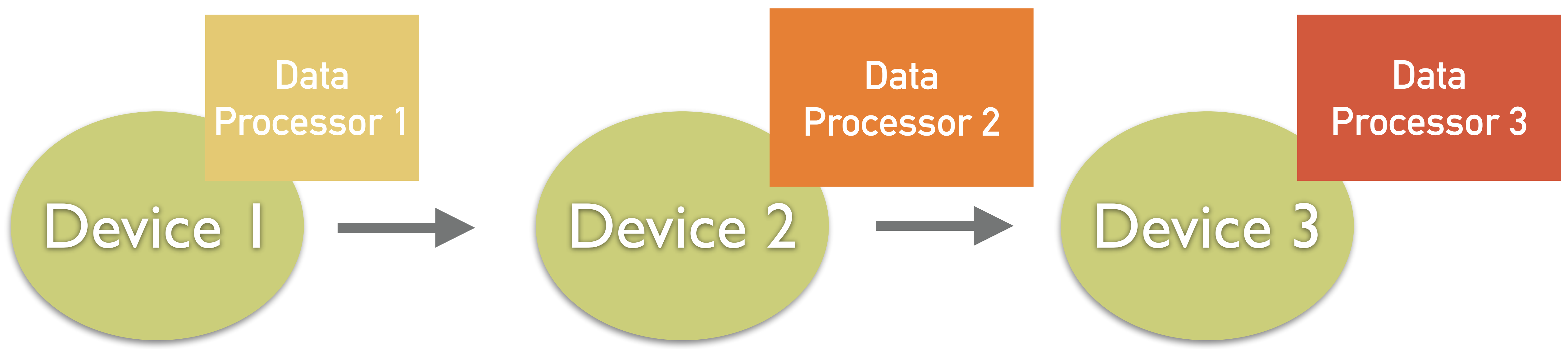
Parts of the chain can be faster due to offloading to GPUs. We can easily increase the number of downstream devices to increase throughput (at the cost of memory).

DATA PROCESSING LAYER: PIPELINING



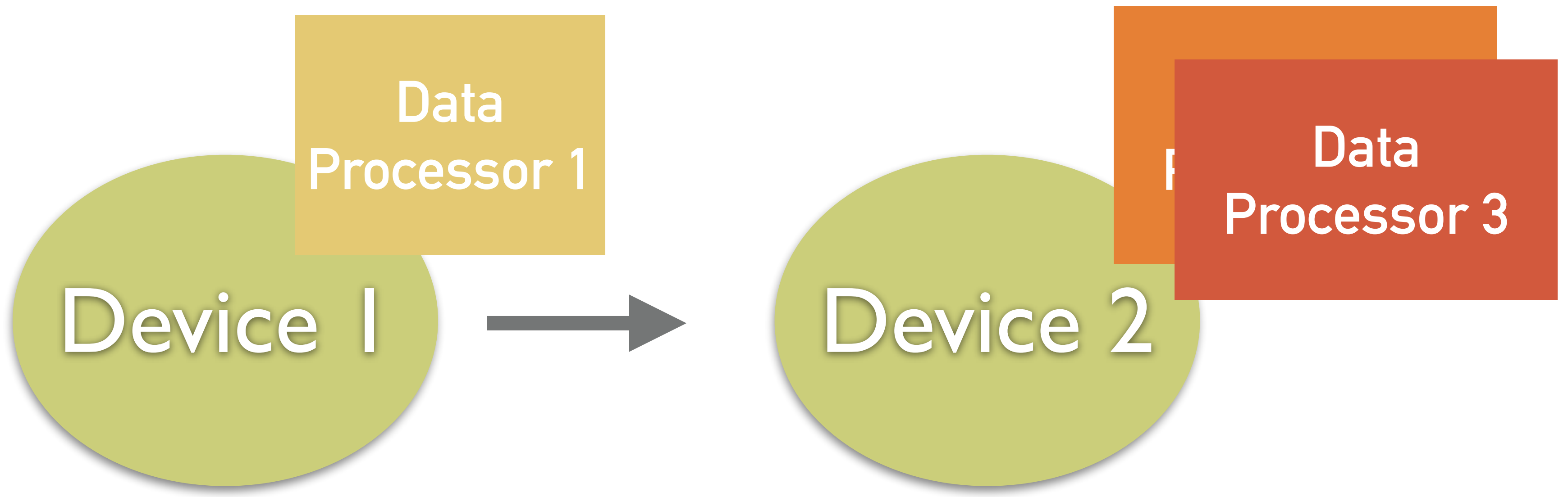
DPL allows to specify pipelining for a given DataProcessors, providing easy parallelisation of processing.

Data Processing Layer: Multiplexing



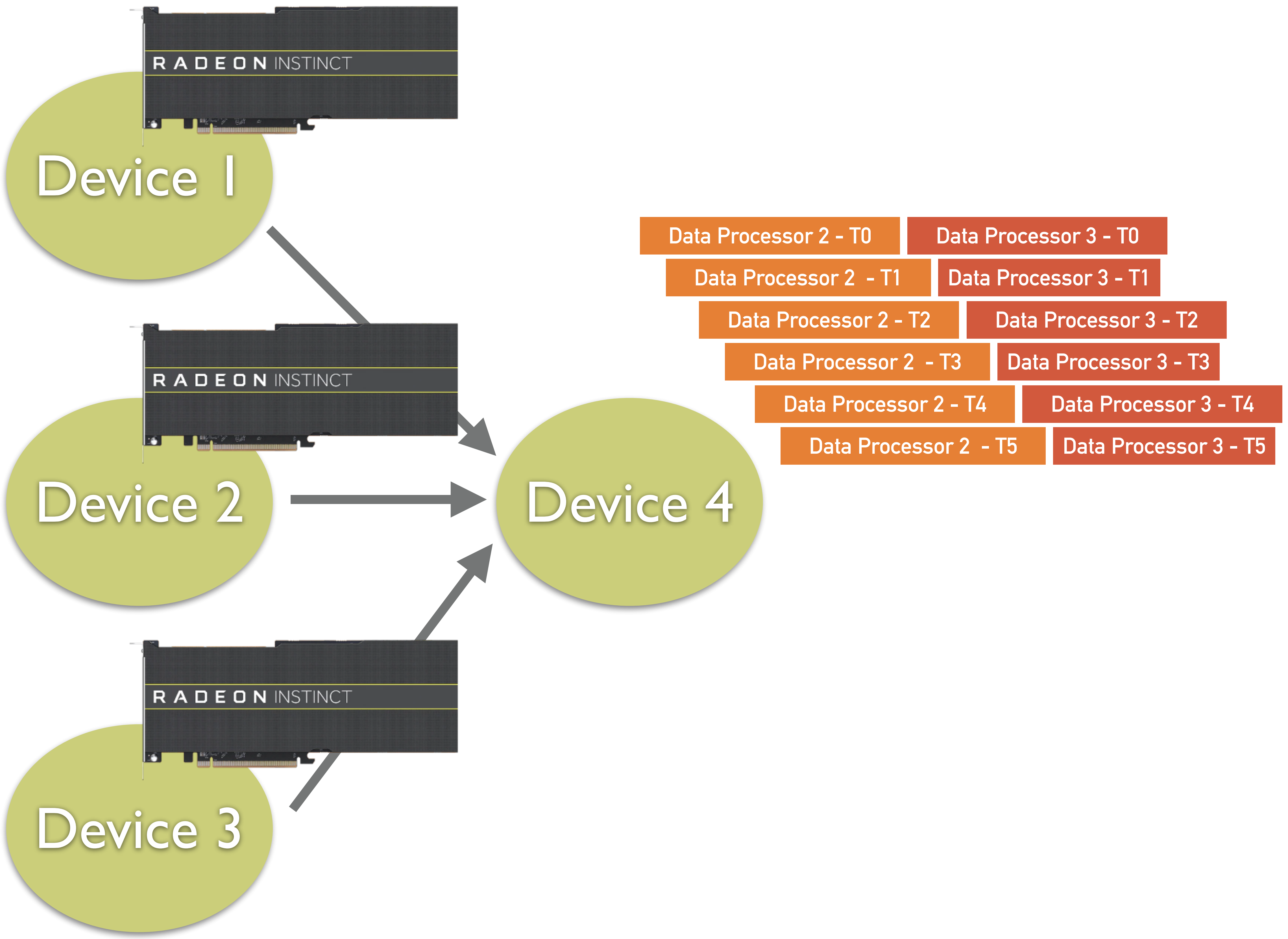
1-to-1 mapping between Devices and DataProcessors not mandatory!

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.

ALICE GPU USAGE STRATEGY

GPU usage in ALICE in the past

- **ALICE has a long history of GPU usage in the online systems, and since 2023 also for offline:**

2010
64 * NVIDIA GTX 480 **in Run 1**
Online TPC tracking

2015
180 * AMD S9000 in **Run 2**
Online TPC tracking

Today
>2000 * AMD MI50 in **Run 3**
Online and Offline barrel tracking

Overview of compute time of reconstruction steps

- **The table below shows the relative compute time (linux cpu time) of the processing steps running on the processor.**

Synchronous processing
(50 kHz Pb-Pb, MC data)

Processing step	% of time
TPC Processing (Tracking, Clustering, Compression)	99.37 %
EMCAL Processing	0.20 %
ITS Processing (Clustering + Tracking)	0.10 %
TPC Entropy Encoder	0.10 %
ITS-TPC Matching	0.09 %
MFT Processing	0.02 %
TOF Processing	0.01 %
TOF Global Matching	0.01 %
PHOS / CPV Entropy Coder	0.01 %
ITS Entropy Coder	0.01 %
Rest	0.08 %

Only data processing steps
Quality control, calibration, event building excluded!

Asynchronous processing
(650 kHz pp, real data, calorimeters not in run)

Processing step	% of time
TPC Processing (Tracking)	61.41 %
ITS TPC Matching	6.13 %
MCH Clusterization	6.13 %
TPC Entropy Decoder	4.65 %
ITS Tracking	4.16 %
TOF Matching	4.12 %
TRD Tracking	3.95 %
MCH Tracking	2.02 %
AOD Production	0.88 %
Quality Control	4.00 %
Rest	2.32 %

Overview of compute time of reconstruction steps

- The table below shows the relative compute time (linux cpu time) of the processing steps running on the processor.

Totally dominated by TPC: >99%

Synchronous processing
(50 kHz Pb-Pb, MC data)

Processing step	% of time
TPC Processing (Tracking, Clustering, Compression)	**99.37 %**
EMCAL Processing	0.20 %
ITS Processing (Clustering + Tracking)	0.10 %
TPC Entropy Encoder	0.10 %
ITS-TPC Matching	0.09 %
MFT Processing	0.02 %
TOF Processing	0.01 %
TOF Global Matching	0.01 %
PHOS / CPV Entropy Coder	0.01 %
ITS Entropy Coder	0.01 %
Rest	0.08 %

Only data processing steps
Quality control, calibration, event building excluded!

Asynchronous processing
(650 kHz pp, real data, calorimeters not in run)

Processing step	% of time
TPC Processing (Tracking)	61.41 %
ITS TPC Matching	6.13 %
MCH Clusterization	6.13 %
TPC Entropy Decoder	4.65 %
ITS Tracking	4.16 %
TOF Matching	4.12 %
TRD Tracking	3.95 %
MCH Tracking	2.02 %
AOD Production	0.88 %
Quality Control	4.00 %
Rest	2.32 %

Overview of compute time of reconstruction steps

ALICE

Synchronous processing
(50 kHz Pb-Pb, MC data)

Processing step	% of time
TPC Processing (Tracking, Clustering, Compression)	**99.37 %**
EMCAL Processing	0.20 %
ITS Processing (Clustering + Tracking)	0.10 %
TPC Entropy Encoder	0.10 %
ITS-TPC Matching	0.09 %
MFT Processing	0.02 %
TOF Processing	0.01 %
TOF Global Matching	0.01 %
PHOS / CPV Entropy Coder	0.01 %
ITS Entropy Coder	0.01 %
Rest	0.08 %

Only data processing steps
Quality control, calibration, event building excluded!

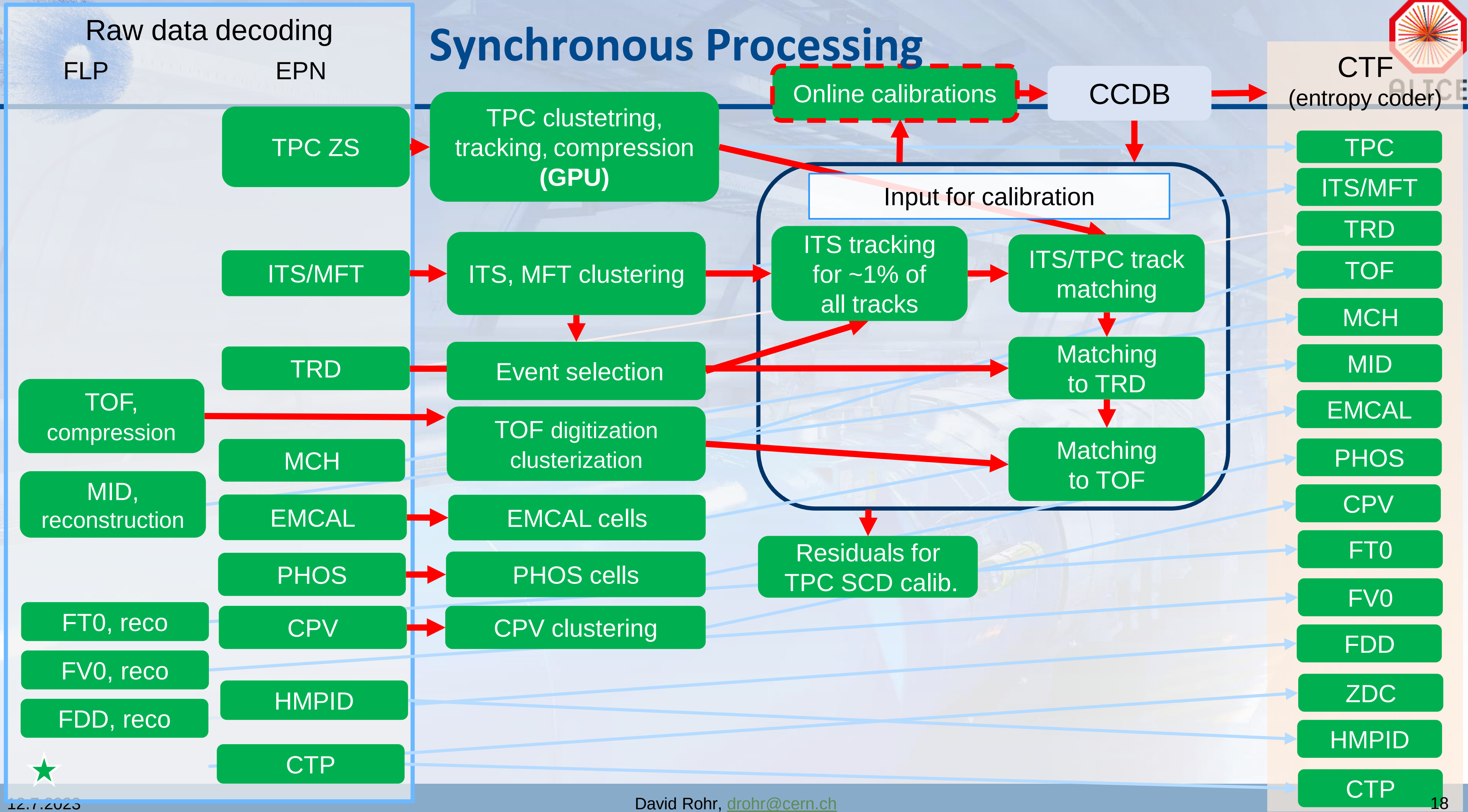
- **Synchronous processing** :
  - **99%** of compute time spent for **TPC**.
  - **EPN farm build for synchronous processing!**
- **Asynchronous reprocessing :**
  - More detectors with significant computing contribution.
  - To be kept in mind, as EPNS also run async. Reco.

- **GPUs** well suited for **TPC** reco (from Run 1 and 2 experience).
- **GPUs** provide the **required compute power**.
  - Time frame concepts yields large enough GPU data chunks.
- Following up **2 scenarios** for EPN GPU processing:

Baseline solution (available today):
- Mandatory for synchronous processing TPC sync. reco on GPU

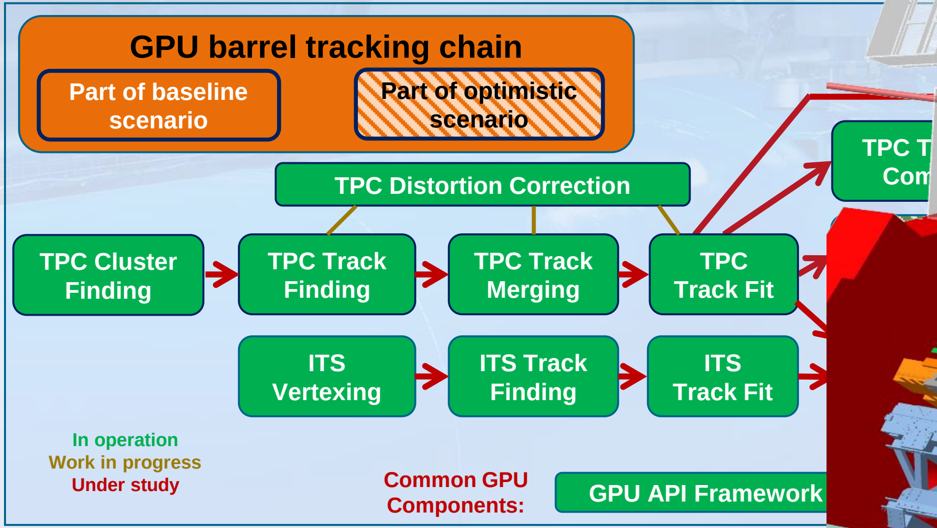
Optimistic solution (under development):
- Achieve best GPU usage in async phase
- Run most of tracking + X on GPU

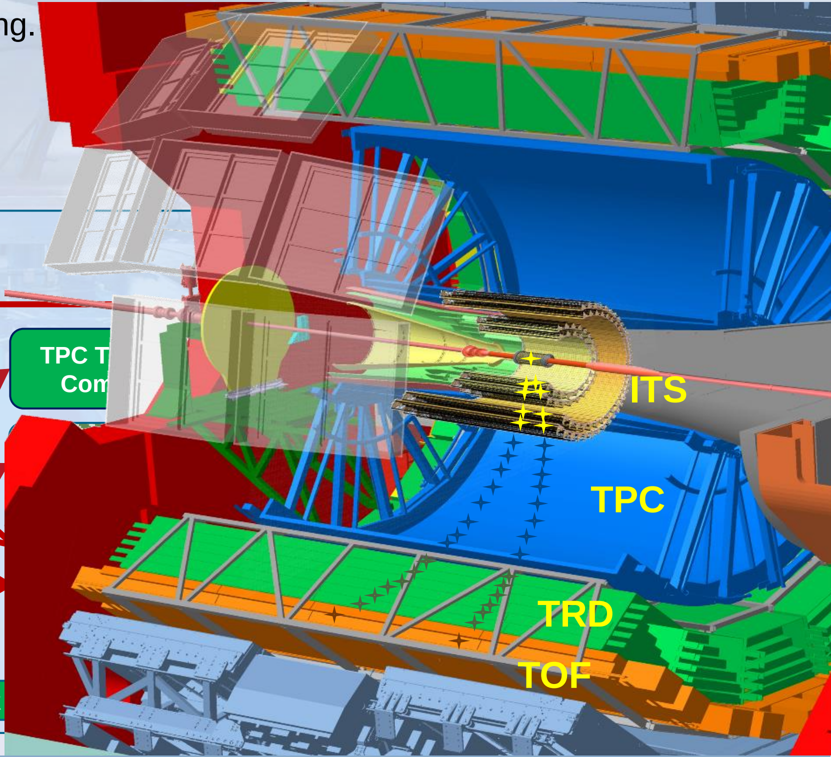
Synchronous Processing



Central barrel global tracking chain

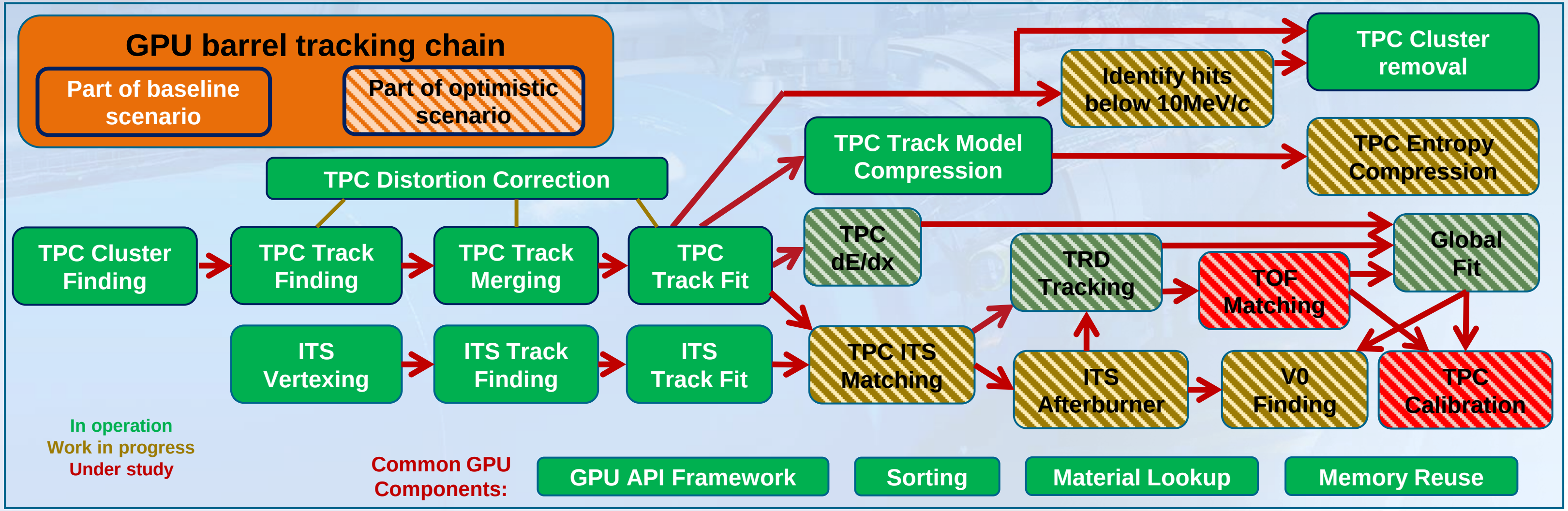
- **Central barrel tracking chosen as best candidate for optimistic scenario for asynchronous reco:**
  - Mandatory **baseline scenario** includes everything that must run on the GPU during synchronous reconstruction.
  - **Optimistic scenario** includes everything related to the barrel tracking.





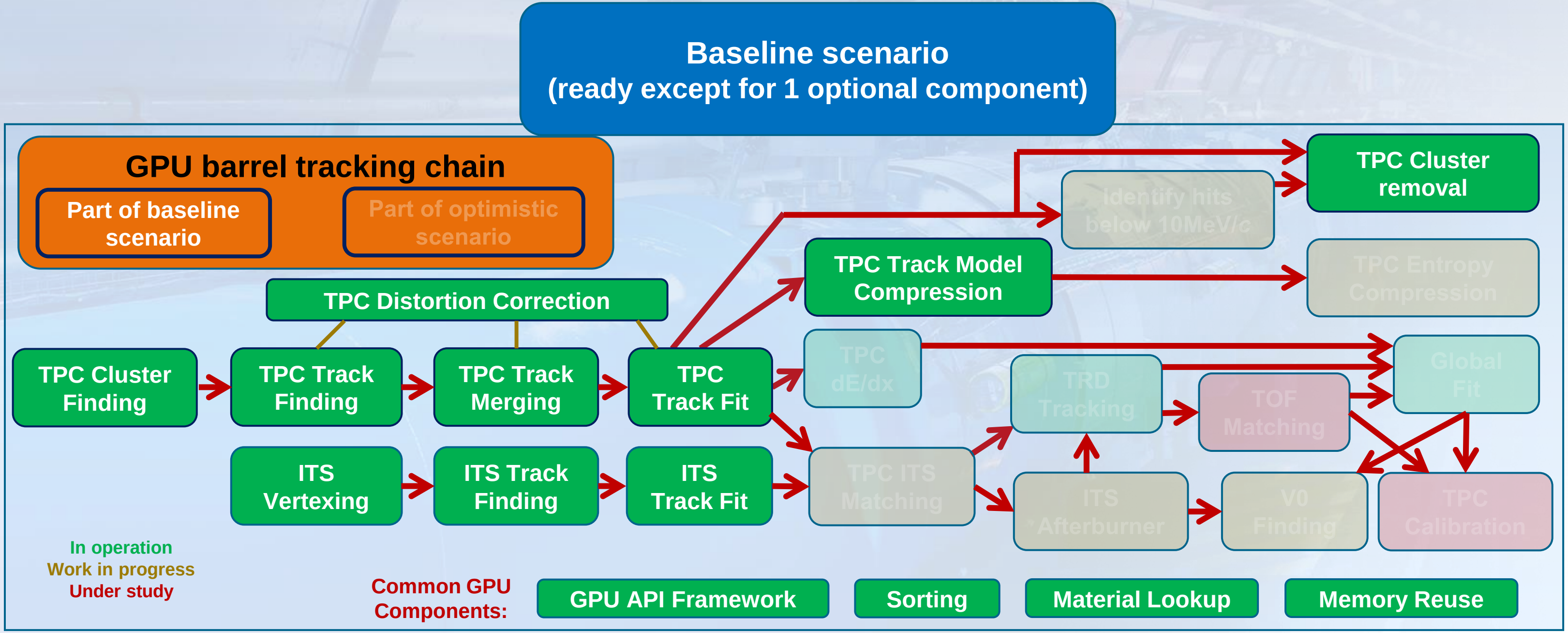
Central barrel global tracking chain

- **Central barrel tracking chosen as best candidate for optimistic scenario for asynchronous reco:**
  - Mandatory **baseline scenario** includes everything that must run on the GPU during synchronous reconstruction.
  - **Optimistic scenario** includes everything related to the barrel tracking.



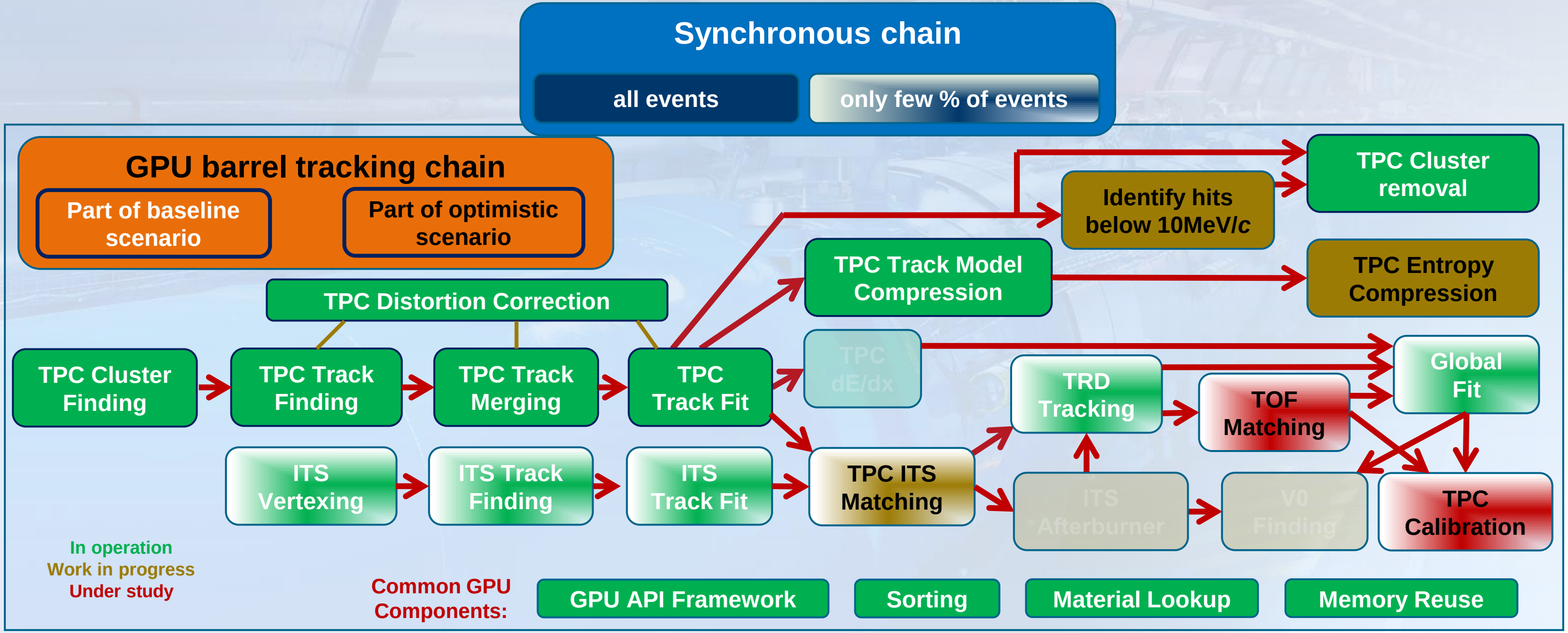
Central barrel global tracking chain

- **Baseline scenario fully implemented.**
  - Not mandatory to speed up the synchronous GPU code further.



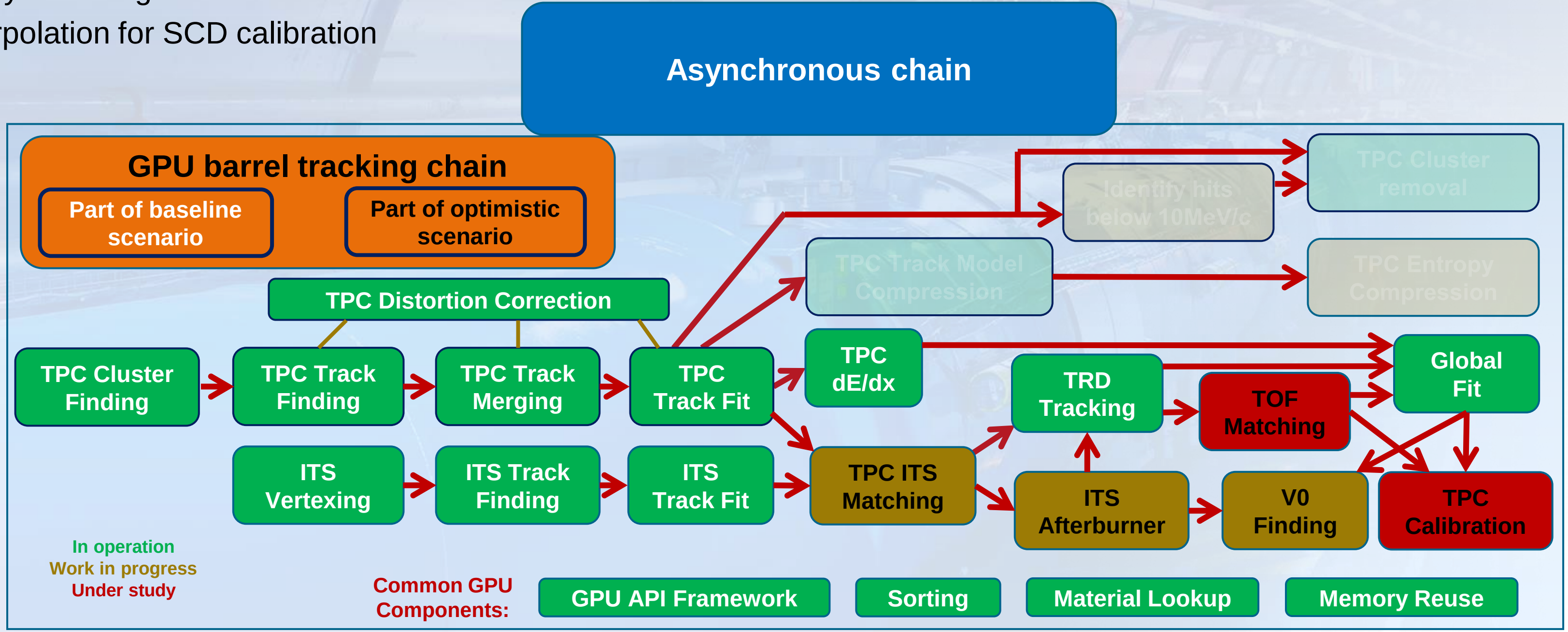
Central barrel global tracking chain

- **TPC synchronous processing almost fully on the GPU.**
  - 2 optional parts still being investigated for sync. reco on GPU: TPC entropy encoding / Looper identification < 10 MeV.



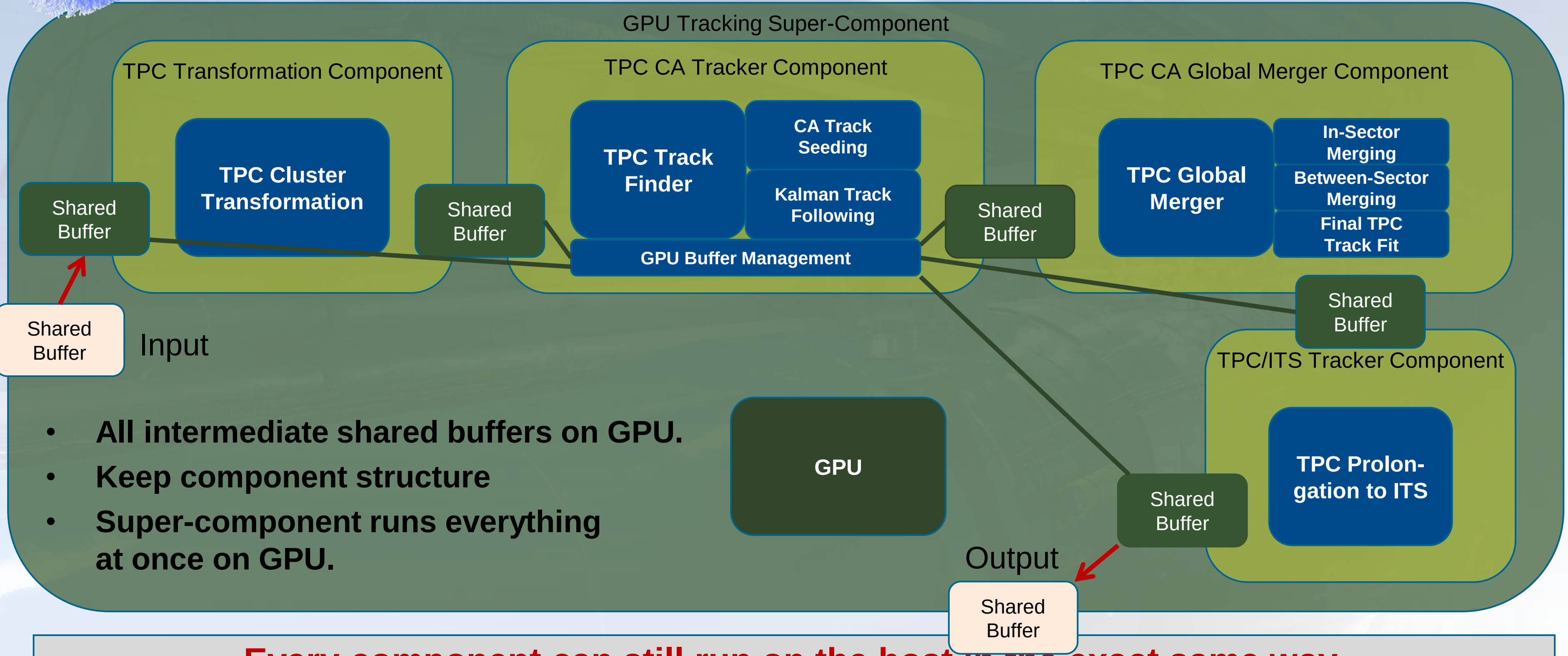
Central barrel global tracking chain

- **Several steps missing in asynchronous reconstruction:**
  - Matching to ITS
  - Matching to TOF
  - Secondary vertexing
  - TPC interpolation for SCD calibration



IMPLEMENTATION

Modular GPU code



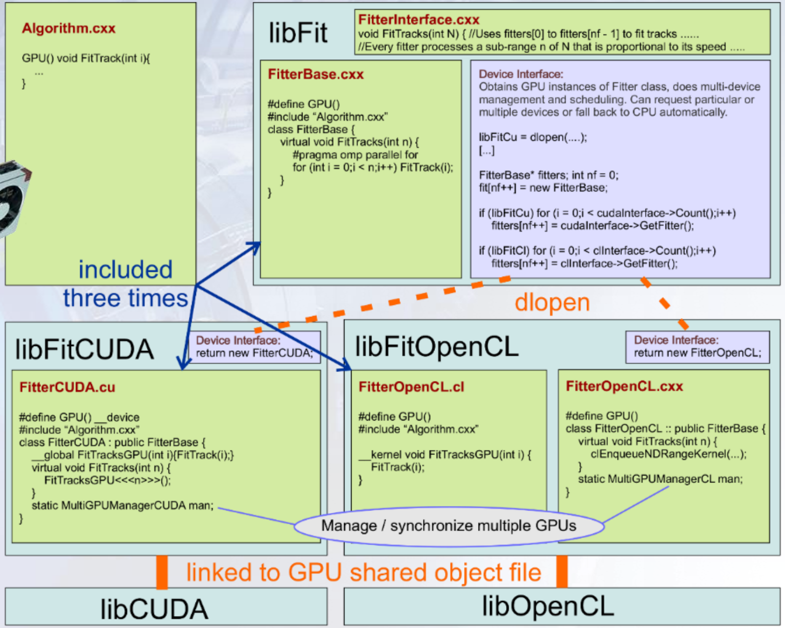
- **All intermediate shared buffers on GPU.**
- **Keep component structure**
- **Super-component runs everything at once on GPU.**

Every component can still run on the host in the exact same way.
Shared buffers either in host memory or in GPU memory.

Plugin system for multiple APIs with common source code

- **Generic common C++ Code compatible to CUDA, OpenCL, HIP, and CPU (with pure C++, OpenMP, or OpenCL).**
  - OpenCL needs clang compiler (ARM or AMD ROCm) or AMD extensions (TPC track finding only on Run 2 GPUs and CPU for testing)
  - Certain worthwhile algorithms have a vectorized code branch for CPU using the Vc library
  - All GPU code swapped out in dedicated libraries, same software binaries run on GPU-enabled and CPU servers

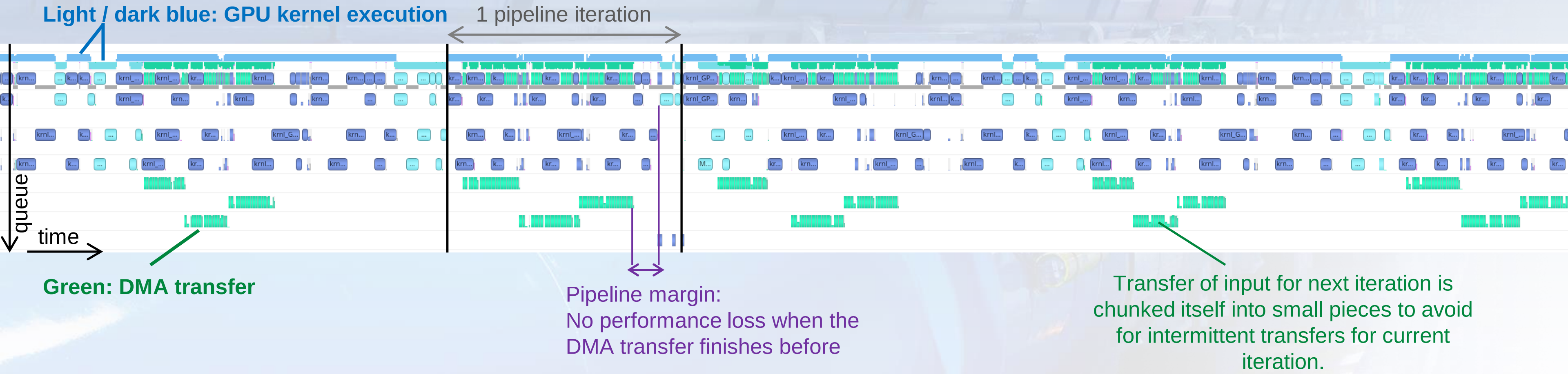
- **Screening different platforms for best price / performance.**
  (including some non-competitive platforms for cross-checks and validation.)
  - **CPUs** (**AMD Zen**, **Intel Skylake**)
    C++ backend with **OpenMP**, AMD **OCL**
  - **AMD GPUs**
    (**S9000** with **OpenCL 1.2**, **MI50** / **Radeon 7** / **Navi** with **HIP** / **OCL 2.x**)
  - **NVIDIA GPUs**
    (**RTX 2080** / **RTX 2080 Ti** / **Tesla T4** with **CUDA**)
  - **ARM Mali GPU** with **OCL 2.x**
    (Tested on dev-board with Mali G52)



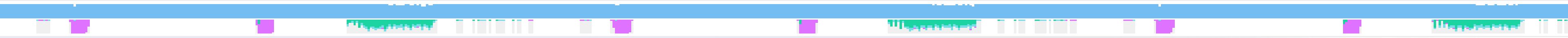
Memory allocation / Pipelined processing

- **Custom allocator: grabs all GPU memory, gives out chunks manually, memory will be reused when possible.**
  - Classically: reuse memory between events.
    - Single events too small for GPU → Process time frames.
  - ALICE reuses memory between different algorithms in a TF, possibly between chunks of collisions in a TF.

- **Zoomed-in plot of TPC Clusterization stage** (part with **largest DMA transfers** → most difficult to hide in pipeline).



- **Full profile** of 3 time frames: **100% GPU utilization** with kernel execution, **No performance loss from data transfer**!



Implementation principles

1. **GPU code should be modular, such that individual parts can run independently.**
   - Multiple consecutive components on the GPU should operate with as little host interaction as possible.
2. **GPU code should be generic C++ and not depend on one particular vendor or API. (O2 supports CUDA, HIP, OpenCL)**
   - No usage of special features that are not portable.
3. **GPU usage should be optional and transparent: running O2 should not require any vendor libraries installed.**
   - All GPU code is contained in plugins, with a common interface.
   - Even multiple plugins (GPU backends) can run on the same node.
4. **Minimize time spent for memory management.**
   - We allocate one large memory segment, and then distribute memory chunks internally.
5. **Processing on GPU and data transfer should overlap, such that the GPU does not idle while waiting for data.**
   - This is implemented via a pipelined processing within time frames, and we also overlap consecutive time frames.
6. **Data chunks processed by the GPU must be large enough to exploit the full parallelism.**
   - Fulfilled by design with TFs containing > 100 collisions.
7. **GPU and CPU output should be as close as possible.**
   - But small differences due to concurrency or non-associative floating point arithmetic cannot be avoided.

Implementation details

- **Multiple GPUs in a server minimize the cost.**
  - Less servers, less network.
  - **Synergies** of using the **same CPU components** for multiple GPUs, same for memory.

- **Splitting the node into 2 NUMA domains minimizes inter-socket communication**
  → **2 virtual EPNs**.
  - Still only **1 HCA** for the input → writing to shared memory segment in **interleaved memory**.

- **GPUs are processing individual time frames → no inter-GPU communication.**
  - Host processes can drive 1 GPU each, or run CPU only tasks.

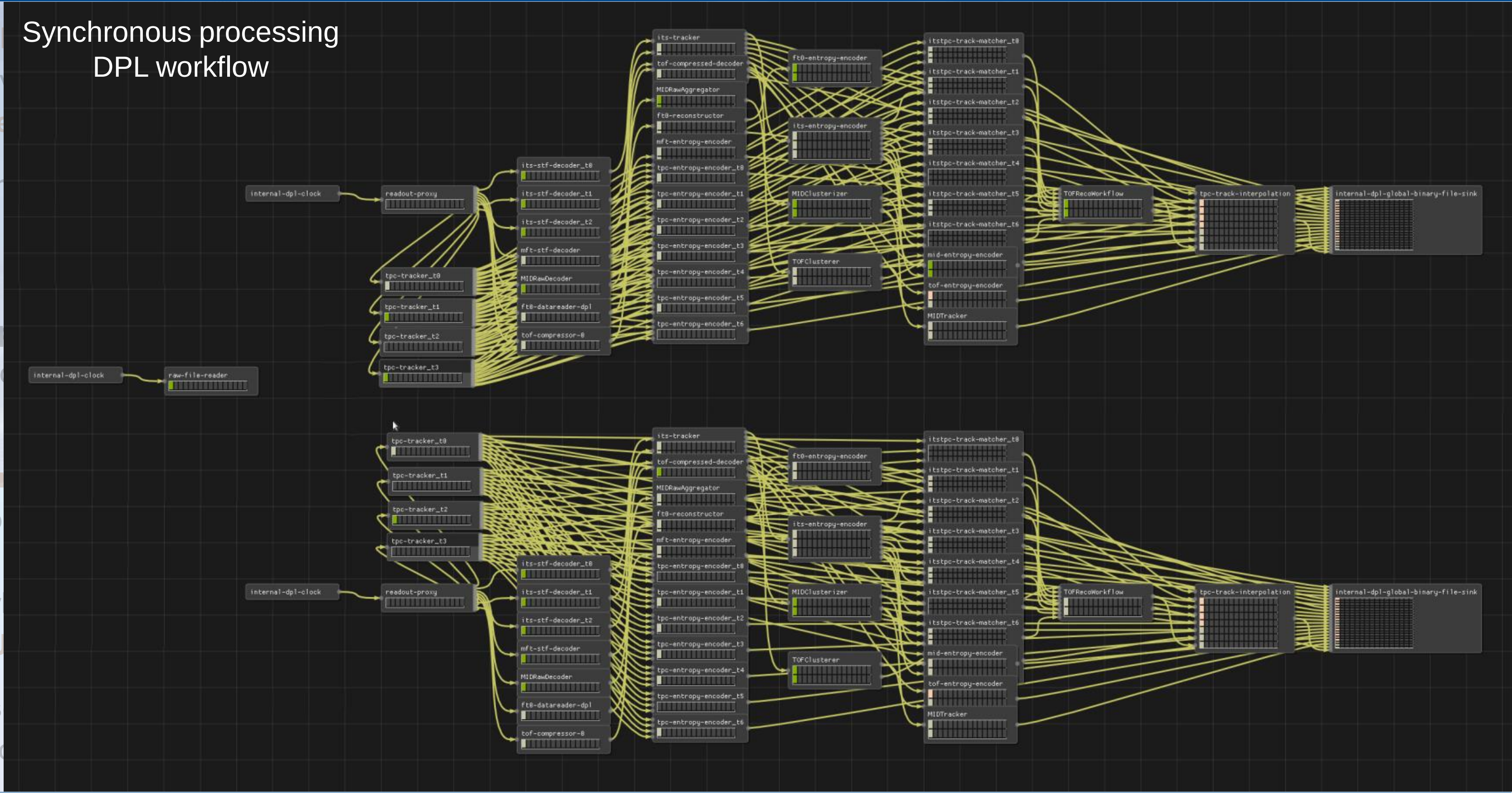
- **GPUs can be shared between algorithms.**
  - With **memory reuse** if within the same process.
  - With separate memory in case of multiple processes (Not done at the moment).

Implementation details

- **Multiple GPUs in a server minimize the cost.**
  - Less servers, less network.
  - **Synergies** of using the **same CPU components** for multiple GPUs, same for memory.
- **Splitting the node into 2 NUMA domains minimizes inter-socket communication**
  → **2 virtual EPNs**.
  - Still only **1 HCA** for the input → writing to shared memory segment in **interleaved memory**.
- **GPUs are processing individual time frames → no inter-GPU communication.**
  - Host processes can drive 1 GPU, or run CPU only tasks.
- **GPUs can be shared between algorithms.**
  - With **memory reuse** if within the same process.
  - With separate memory in case of multiple processes (Not done at the moment).
- **Benchmarked with MC data: For 100% utilization of 8 GPUs** (AMD MI50)**, we need:**
  - ~**50 CPU cores**, ~**400 GB** of memory, **30 GB/s** network input speed, GPU PCIe negligible.
- **Selected server:**
  - Supermicro AS-4124GS-TNR, **8 * MI50** GPU, **2 * 32 core** AMD Rome 7452 CPU (2.35 GHz), **512 GB RAM** (16 * 32GB)
  - Infiniband HDR / HDR100 network.

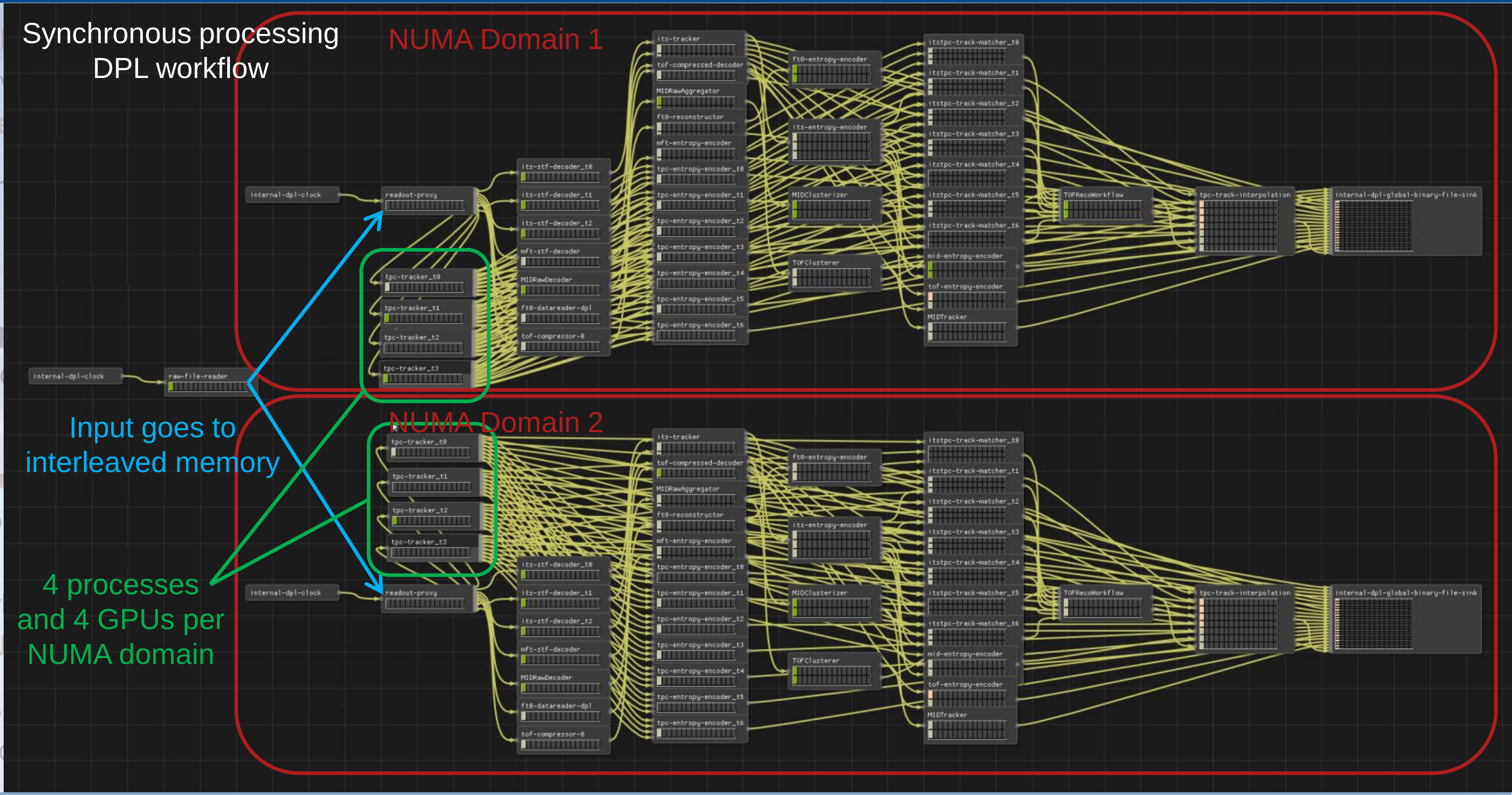
Implementation details

- Multiple GI
  - Less ser
  - Synergie
- Splitting th
  → 2 virtual
  - Still only
- GPUs are
  - Host proc
- GPUs can
  - With me
  - With sep
- Benchmar
  - ~50 CPU
- Selected s
  - Supermic



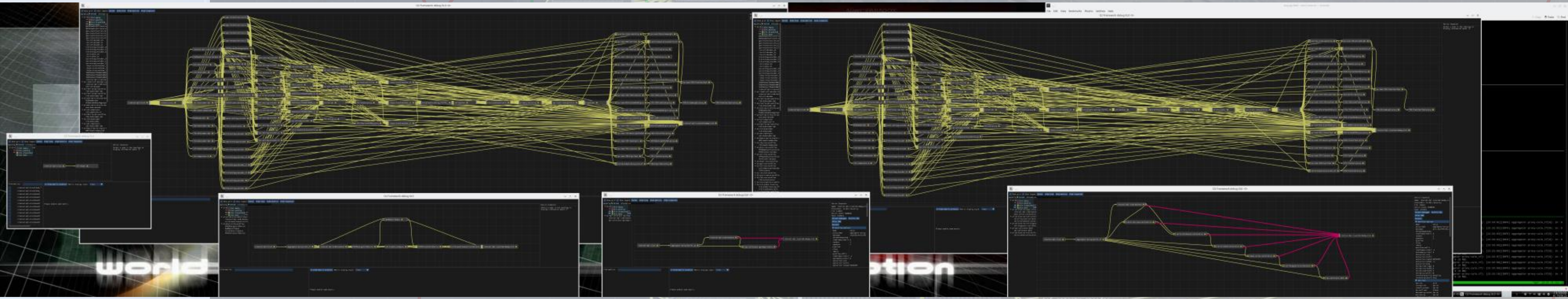
Implementation details

- Multiple G
  - Less ser
  - Synergie
- Splitting th
  → 2 virtual
  - Still only
- GPUs are
  - Host pro
- GPUs can
  - With me
  - With sep
- Benchmar
  - ~50 CPU
- Selected s
  - Supermic



Implementation details

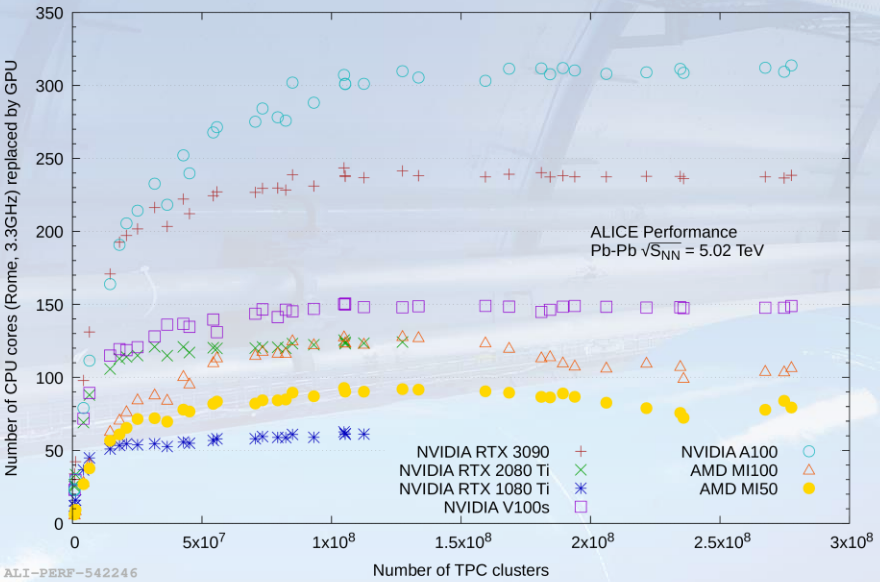
To illustrate the complexity:
Full synchronous workflow including
Quality Control and Calibration

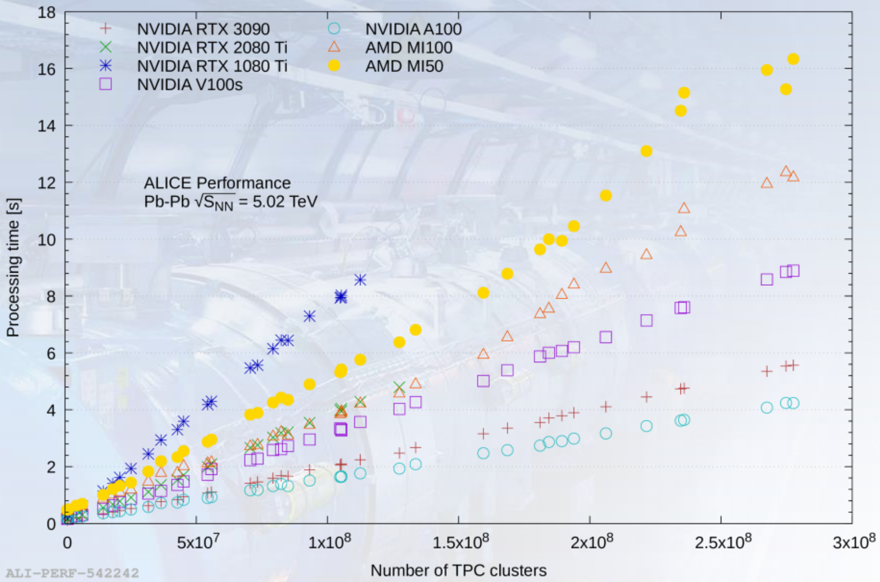


PERFORMANCE

Synchronous processing performance

- Performance of Alice O2 software on different GPU models and compared to CPU.



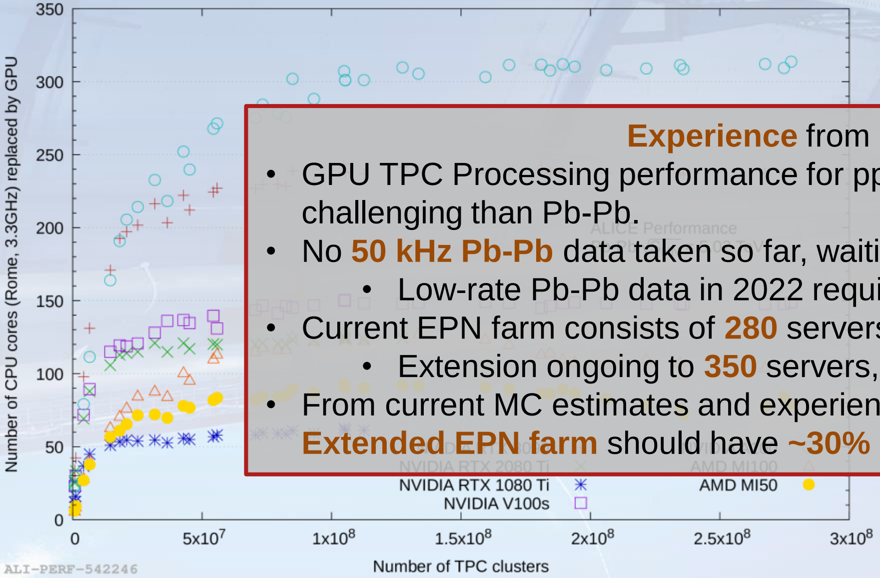


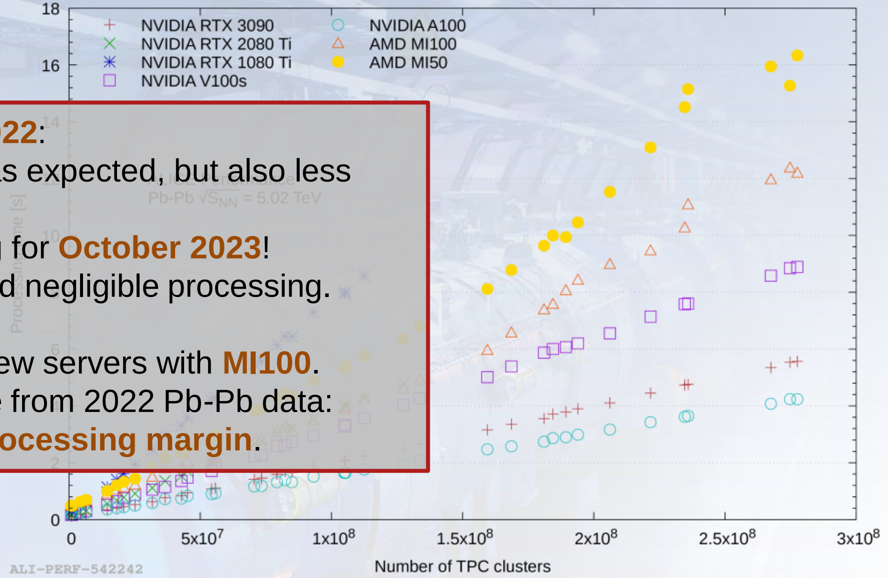
- **MI50 GPU replaces ~80 AMD Rome CPU cores in synchronous reconstruction.**
  - Includes **TPC clusterization**, which is **not optimized** for the CPU!
  - **~55 CPU cores** in **asynchronous** reconstruction (more realistic comparison).
- **Validated software with MI100 GPU, ca 35% faster.**

Without GPUs, more than 2000 64-core servers would be needed for online processing!

Synchronous processing performance

- **Performance of Alice O2 software on different GPU models and compared to CPU.**





Experience from **2022**:

- GPU TPC Processing performance for pp as expected, but also less challenging than Pb-Pb.
- No **50 kHz Pb-Pb** data taken so far, waiting for **October 2023**!
  - Low-rate Pb-Pb data in 2022 required negligible processing.
- Current EPN farm consists of **280** servers.
  - Extension ongoing to **350** servers, new servers with **MI100**.
- From current MC estimates and experience from 2022 Pb-Pb data: **Extended EPN farm** should have **~30% processing margin**.

- **MI50 GPU replaces ~80 AMD Rome CPU cores in synchronous reconstruction.**
  - Includes **TPC clusterization**, which is **not optimized** for the CPU!
  - **~55 CPU cores** in **asynchronous** reconstruction (more realistic comparison).
- **Validated software with MI100 GPU, ca 35% faster.**

Without GPUs, more than 2000 64-core servers would be needed for online processing!

Overview of compute time of reconstruction steps

- The table below shows the relative compute time (linux cpu time) of the processing steps running on the processor.

Synchronous processing
(50 kHz Pb-Pb, MC data, processing only)

Processing step	% of time
TPC Processing (Tracking, Clustering, Compression)	99.37 %
EMCAL Processing	0.20 %
ITS Processing (Clustering + Tracking)	0.10 %
TPC Entropy Encoder	0.10 %
ITS-TPC Matching	0.09 %
MFT Processing	0.02 %
TOF Processing	0.01 %
TOF Global Matching	0.01 %
PHOS / CPV Entropy Coder	0.01 %
ITS Entropy Coder	0.01 %
Rest	0.08 %

Asynchronous processing
(650 kHz pp, real data, calorimeters not in run)

Processing step	% of time
TPC Processing (Tracking)	61.41 %
ITS TPC Matching	6.13 %
MCH Clusterization	6.13 %
TPC Entropy Decoder	4.65 %
ITS Tracking	4.16 %
TOF Matching	4.12 %
TRD Tracking	3.95 %
MCH Tracking	2.02 %
AOD Production	0.88 %
Quality Control	4.00 %
Rest	2.32 %

Overview of compute time of reconstruction steps

- **The table below shows the relative compute time (linux cpu time) of the processing steps running on the processor.**
  - Synchronous reconstruction fully dominated by the TPC (99%), no reason to offload anything else to the GPU.
  - In async reco, currently the 61.4% TPC are on the GPU, with the full optimistic scenario (full barrel tracking) it will be 79.77%.

Synchronous processing
(50 kHz Pb-Pb, MC data, processing only)

Processing step	% of time
TPC Processing (Tracking, Clustering, Compression)	99.37 %
EMCAL Processing	0.20 %
ITS Processing (Clustering + Tracking)	0.10 %
TPC Entropy Encoder	0.10 %
ITS-TPC Matching	0.09 %
MFT Processing	0.02 %
TOF Processing	0.01 %
TOF Global Matching	0.01 %
PHOS / CPV Entropy Coder	0.01 %
ITS Entropy Coder	0.01 %
Rest	0.08 %

Asynchronous processing
(650 kHz pp, real data, calorimeters not in run)

Processing step	% of time
TPC Processing (Tracking)	61.41 %
ITS TPC Matching	6.13 %
MCH Clusterization	6.13 %
TPC Entropy Decoder	4.65 %
ITS Tracking	4.16 %
TOF Matching	4.12 %
TRD Tracking	3.95 %
MCH Tracking	2.02 %
AOD Production	0.88 %
Quality Control	4.00 %
Rest	2.32 %

Running on GPU in baseline scenario **Running on GPU in optimistic scenario**

Overview of compute time of reconstruction steps

- **Async reco GPU speedup on the EPN:**
  - The **speed of light** is **~6.5x** speedup, since **85%** of the **compute power** is in the **GPU** (reduce the CPU time by 85%, more becomes GPU-bound).
    - Only in case everything scales as well as TPC processing.
    - Even then cannot be reached since GPU processing needs CPU resources.
  - **Today**, offloading the ~**60%** of the async to the GPU should yield a **speedup** around **2.5x**.
    - We remove 60% of the CPU time, while we are still CPU-bound, but we have some overhead CPU resources for driving the 8 GPUs.
  - In the **optimistic scenario**, by offloading **80%** we might get close to **5x**.
    - Still a bit away from the speed of light.

Asynchronous processing
(650 kHz pp, real data, calorimeters not in run)

Processing step	% of time
TPC Processing (Tracking)	61.41 %
ITS TPC Matching	6.13 %
MCH Clusterization	6.13 %
TPC Entropy Decoder	4.65 %
ITS Tracking	4.16 %
TOF Matching	4.12 %
TRD Tracking	3.95 %
MCH Tracking	2.02 %
AOD Production	0.88 %
Quality Control	4.00 %
Rest	2.32 %

Running on GPU in baseline scenario

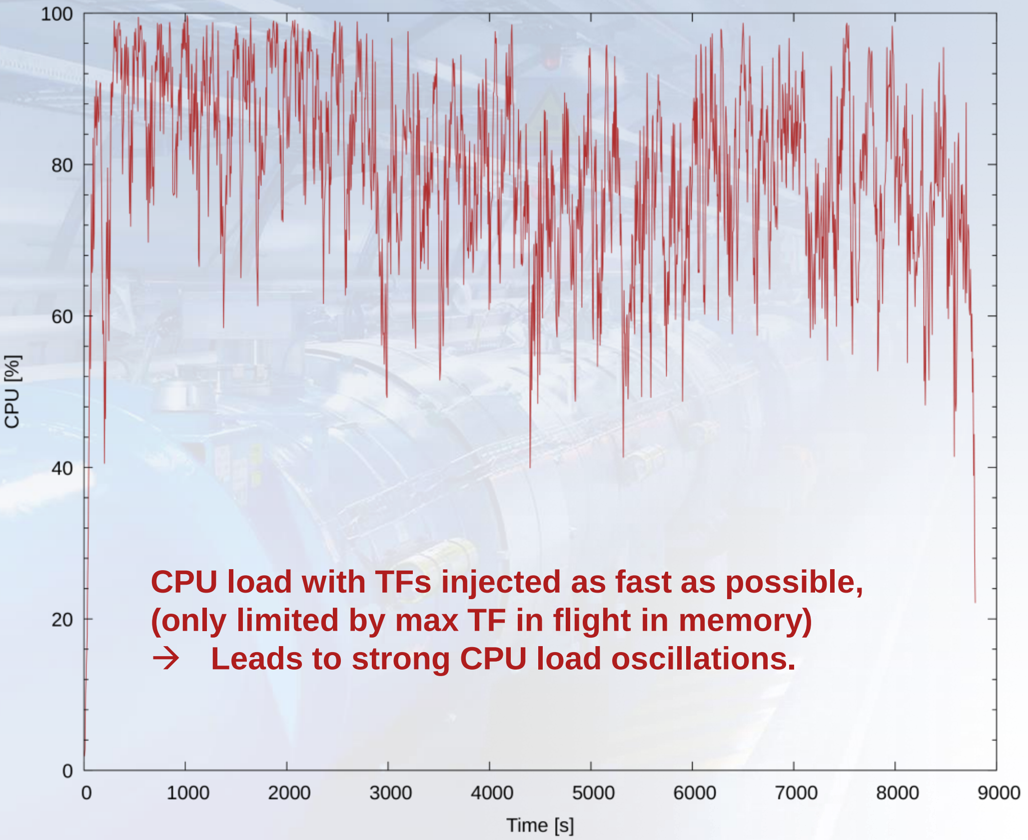
Running on GPU in optimistic scenario

Time frame scheduling sync vs. async

- **Synchronous processing:** rate defined from data taking: **351 TFs per second.**
  - EPNs must handle that rate, and have some margin.
- **Asynchronous processing:** process TFs **as fast as possible**, ideally reach **100% CPU load**.

- Need many TFs in flight, to use all CPU cores via DPL pipelines.
- Available memory limits the maximum number of TFs in flight.

- Constant TF publishing rate ideal to spread the load horizontally and vertically in the processing graph.
- Injecting TFs into the chain with unstable rate leads to oscillations in the processing.



Real speedup in asynchronous reconstruction

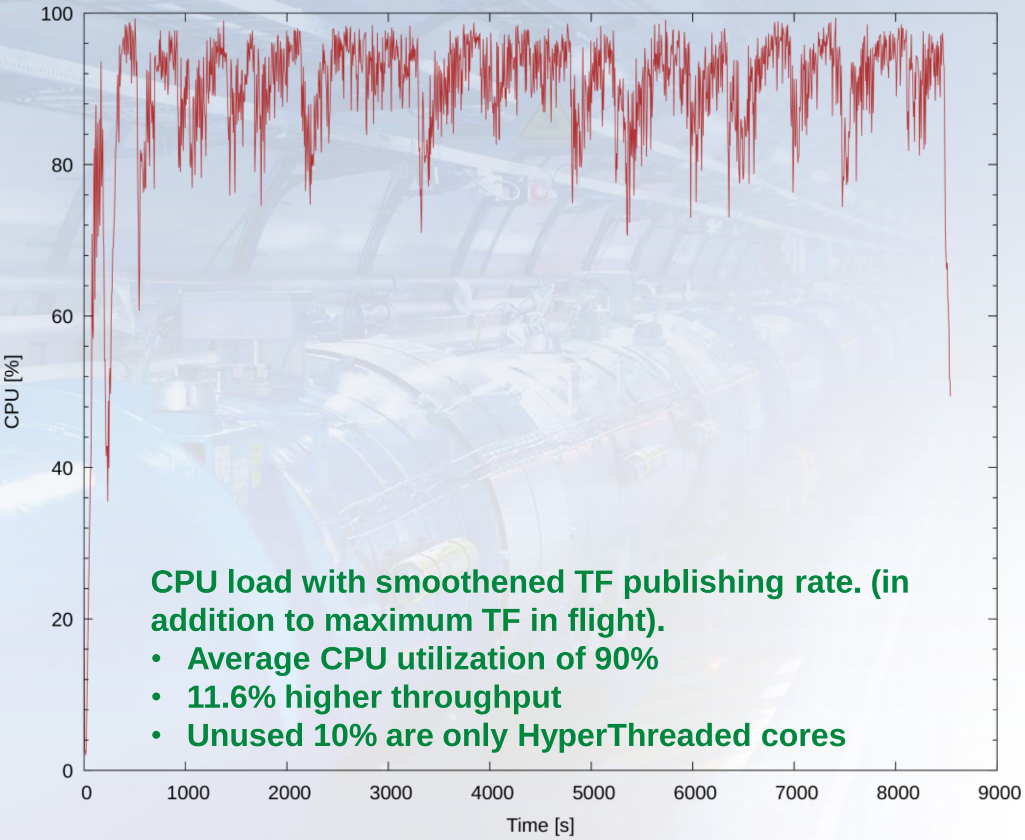
- **Synchronous processing:** rate defined from data taking: **351 TFs per second.**
  - EPNs must handle that rate, and have some margin.
- **Asynchronous processing:** process TFs **as fast as possible**, ideally reach **100% CPU load**.

- Need many TFs in flight, to use all CPU cores via DPL pipelines.
- Available memory limits the maximum number of TFs in flight.

- Constant TF publishing rate ideal to spread the load horizontally and vertically in the processing graph.
- Injecting TFs into the chain with unstable rate leads to oscillations in the processing.

→ **Heuristic to smoothen TF publishing rate solves the problem.**

→ **Will use 2.8 ms TFs from 2023 to reduce memory usage in GRID sites.**



CPU load with smoothened TF publishing rate. (in addition to maximum TF in flight).
- **Average CPU utilization of 90%**
- **11.6% higher throughput**
- **Unused 10% are only HyperThreaded cores**

Real speedup in asynchronous reconstruction

- **For asynchronous reconstruction, EPN nodes are used as GRID nodes.**
  - **Identical workflow** as on other **GRID** sites, only different configuration using GPU, more memory, more CPU cores.
  - EPN farm split in **2 scheduling pools**: synchronous and asynchronous.
    - Unused nodes in the synchronous pool are moved to the asynchronous pool.
    - As needed for data-taking, nodes are moved to the synchronous pool with lead time to let the current jobs finished.
      - If needed immediately, GRID jobs are killed and nodes moved immediately.

Real speedup in asynchronous reconstruction

- **For asynchronous reconstruction, EPN nodes are used as GRID nodes.**
  - **Identical workflow** as on other **GRID** sites, only different configuration using GPU, more memory, more CPU cores.
  - EPN farm split in **2 scheduling pools**: synchronous and asynchronous.
    - Unused nodes in the synchronous pool are moved to the asynchronous pool.
    - As needed for data-taking, nodes are moved to the synchronous pool with lead time to let the current jobs finished.
      - If needed immediately, GRID jobs are killed and nodes moved immediately.
- **Performance benchmarks cover multiple cases:**
  - EPN split into 16 * **8 cores**, or into 8 * **16 cores**, ignoring the GPU : to compare CPUs and GPUs.
  - EPN split into 8 or 2 identical fractions: **1 NUMA** domain (4 GPUs) or **1 GPU**.
- **Processing time per time-frame while the GRID job is running (neglecting overhead at begin / end).**
  - In all cases server **fully loaded** with **identical jobs**, to avoid effects from HyperThreading, memory, etc.

Configuration (2022 pp, 650 kHz)	Time per TF (11ms, 1 instance)	Time per TF (11ms, full server)
CPU 8 core	76.91s	4.81s
CPU 16 core	34.18s	**4.27s**
1 GPU + 16 CPU cores	14.60s	1.83s
1 NUMA domain (4 GPUs + 64 cores)	3.5s	**1.70s**

Factor 2.51
Matches expected factor 2.5

Real speedup in asynchronous reconstruction

- **For asynchronous reconstruction, EPN nodes are used as GRID nodes.**
  - **Identical workflow** as on other **GRID** sites, only different configuration using GPU, more memory, more CPU cores.
  - EPN farm split in **2 scheduling pools**: synchronous and asynchronous.
    - Unused nodes in the synchronous pool are moved to the asynchronous pool.
    - As needed for data-taking, nodes are moved to the synchronous pool with lead time to let the current jobs finished.
      - If needed immediately, GRID jobs are killed and nodes moved immediately.
- **Performance benchmarks cover multiple cases:**
  - EPN split into 16 * **8 cores**, or into 8 * **16 cores**, ignoring the GPU : to compare CPUs and GPUs.
  - EPN split into 8 or 2 identical fractions: **1 NUMA** domain (4 GPUs) or **1 GPU**.
- **Processing time per time-frame while the GRID job is running (neglecting overhead at begin / end).**
  - In all cases server **fully loaded** with **identical jobs**, to avoid effects from HyperThreading, memory, etc.

Configuration (2022 pp, 650 kHz)	Time per TF (11ms, 1 instance)	Time per TF (11ms, full server)
CPU 8 core	76.91s	4.81s
CPU 16 core	34.18s	**4.27s**
1 GPU + 16 CPU cores	14.60s	1.83s
1 NUMA domain (4 GPUs + 64 cores)	3.5s	**1.70s**

Configuration used for async processing (Also resembles most the synchronous processing configuration)

Factor 2.51

Matches expected factor 2.5

Real speedup in asynchronous reconstruction

- **Overhead at begining / end of job:**
  - Constant overhead at start / stop of processing: **149 s (1.8%)**
  - → Negligible compared to job runtime (benchmark job was 8491 **s**, could be extended to >10h)
  - Additional time needed for AOD checking / merging: **238s** (**2.8%**, CPU only Postprocessing to speed up analysis)
  - Time lost at processing dip at the beginning during condition fetching / initialization: **32s** (**0.4%**)
- **Some interesting performance comparisons:**
  - 1 GPU workflow, running **isolated** on a node v.s. running **8 times** in parallel on a node: ??% faster (HyperThreading).
  - 1 NUMA workflow, with **rate smoothing** v.s. **without rate smoothing**: **11.6%** faster.
- **Benefits of 2 * 1 NUMA domain workflow over 8 * 1 GPU workflow:**
  - Not all CPU processes duplicated → fewer processes, and significantly less memory consumption (~ 100 GB difference).
  - Share the CPU processes in DPL workflow → more CPU capacity compensates load fluctuations, less context switches.

Configuration (2022 pp, 650 kHz)	Time per TF (11ms, 1 instance)	Time per TF (11ms, full server)
CPU 8 core	76.91s	4.81s
CPU 16 core	34.18s	**4.27s**
1 GPU + 16 CPU cores	14.60s	1.83s
1 NUMA domain (4 GPUs + 64 cores)	3.5s	**1.70s**

Factor 2.51
Matches expected factor 2.5

Lessons learned

- **GPUs can speed up the processing significantly.**
  - Not necessarily all workload needs to run on GPU, but the hot spot.
- **Inexperienced users can contribute improvements to algorithms, for implementing full new reconstruction steps on GPU more expert knowledge is needed.**
- **(Remote) Debug GUI to inspect topology (remotely) is very useful.**
- **Scheduling for synchronous and asynchronous processing is different.**
- **Should also optimize for memory perhaps sacrificing a bit of performance.**
  - 11ms v.s. 2.8ms TFs.
  - **Memory** is more **limited** on GRID sites than on your online farm.
- **A common software framework for multiple GPU types allows for changing the vendor and simplifies debugging.**
- **Default build should contain all GPU backends, to be enabled transparently and optionally (e.g. via plugins).**
- **Having the full reconstruction in a single monolithic process is failure-prone and difficult to debug (Run 3), too many individual processes can have huge memory demand → good compromise needed.**

Conclusions

- **ALICE employs GPUs heavily to speed up online and offline processing.**
  - **99%** of **synchronous reconstruction** on the **GPU** (no reason at all to port the rest).
  - Today ~**60%** of full **asynchronous processing** (for 650 kHz pp) on **GPU** (if offline jobs on the EPN farm).
    - Will increase to **80%** with full barrel tracking (**optimistic scenario**).
- **Synchronous processing successful in 2021 - 2023.**
  - **pp** data taking and **low-IR Pb-Pb** went **smooth** and as expected, but not causing full compute load.
  - **Full rate** will come with Pb-Pb in **October 2023**.
    - **50 kHz Pb-Pb** processing **validated** with data replay of **MC** data (~ **30% margin**).
- **Asynchronous reconstruction has started, processing the TPC reconstruction on the GPUs in the EPN farm, and in CPU-only style on the CERN GRID site.**
  - **EPN** nodes are **2.51x** faster when using **GPUs**.