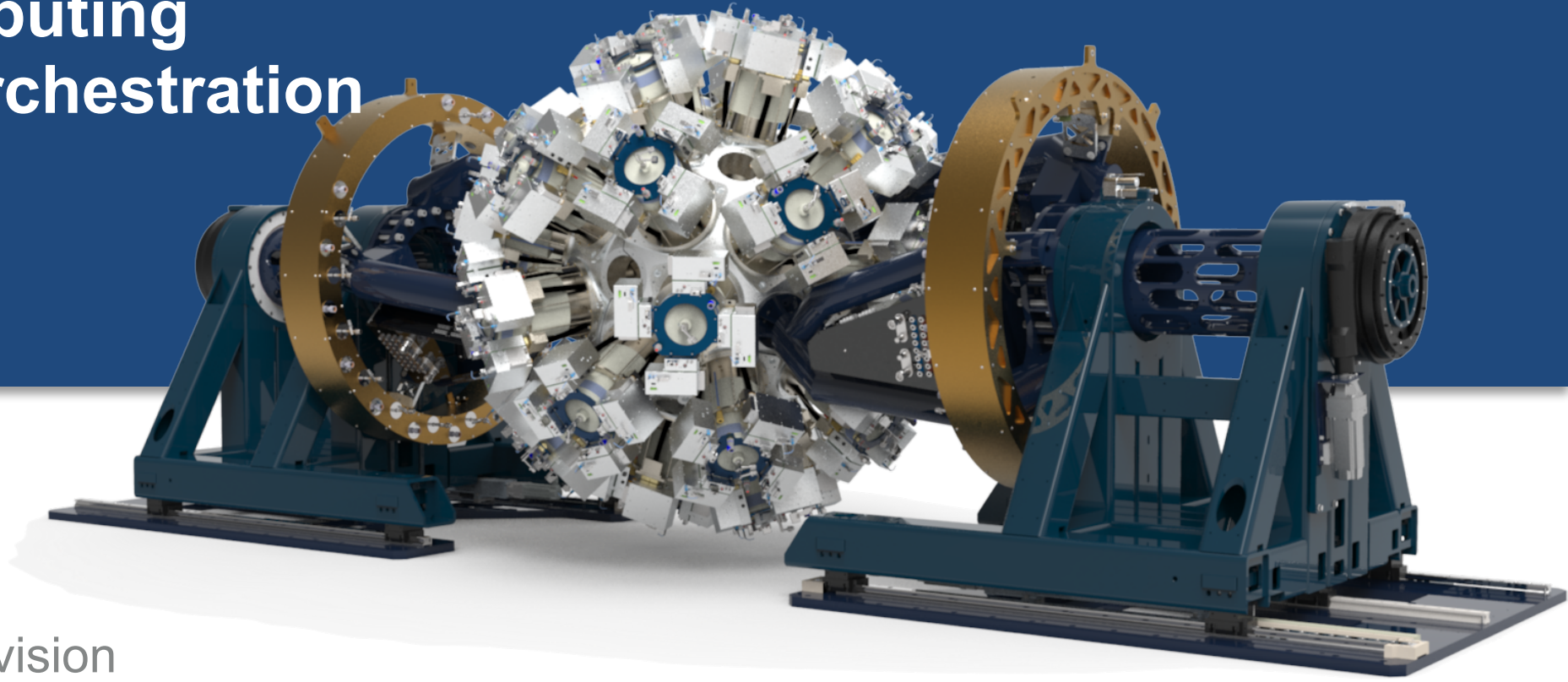


GRETA Computing Systems - Orchestration



Mario Cromaz

Nuclear Science Division

Lawrence Berkeley National Laboratory



BERKELEY LAB

Bringing Science Solutions to the World



Office of
Science

Software-based Data Streaming System



Specs:

4800 100 MHz digitizers

Raw data rate: 7.5 Tb/s

Post FPGA farm: 32 Gb/s

Event rate: 480 kHz

Processing time: 4 ms /
event / core

End-to-end latency:
seconds

**GRETA is a large-scale gamma-ray
spectrometer**

Primary instrument at FRIB

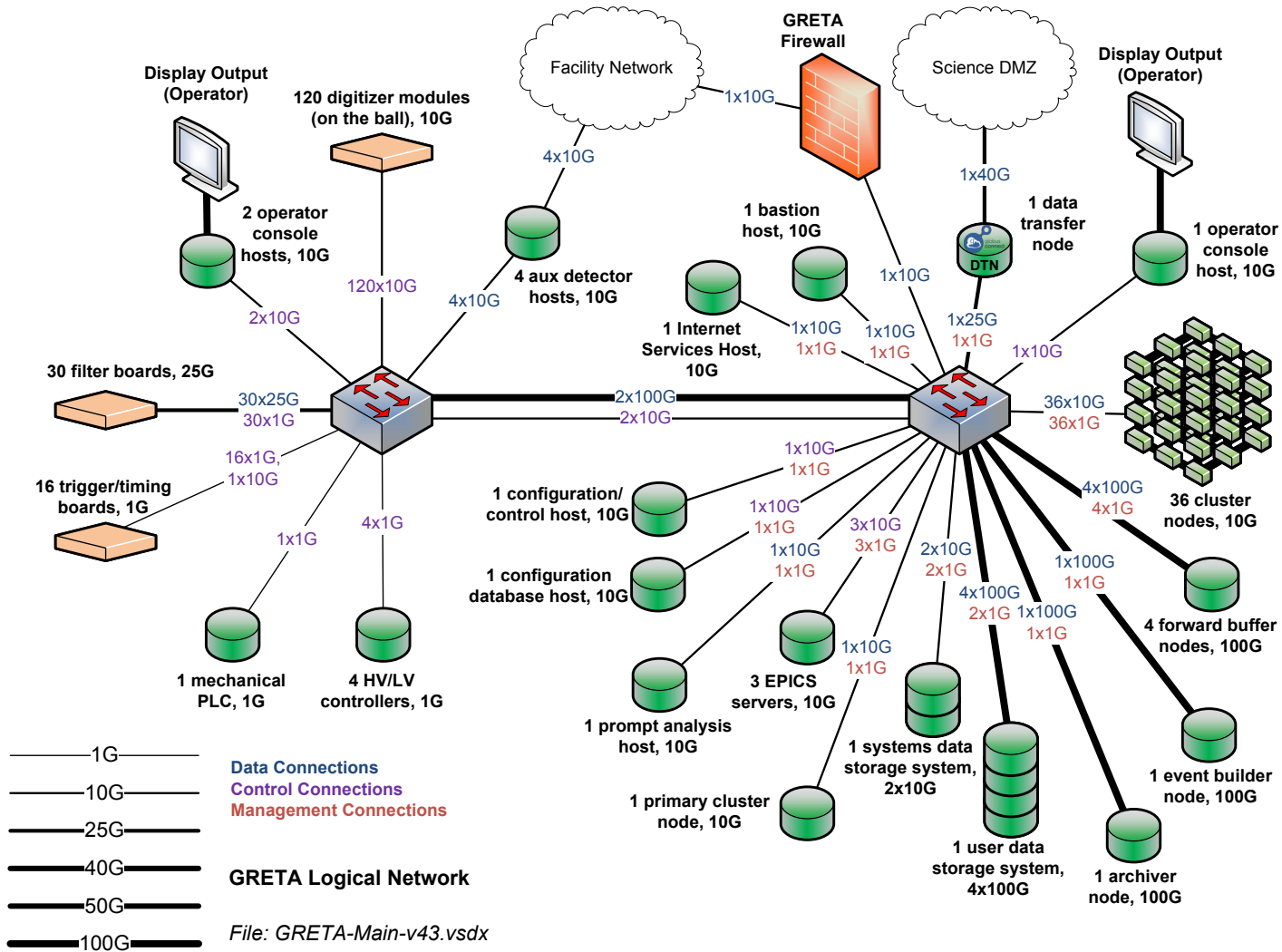
413b project (NP)

Collab with ESnet, ORNL/NCCS

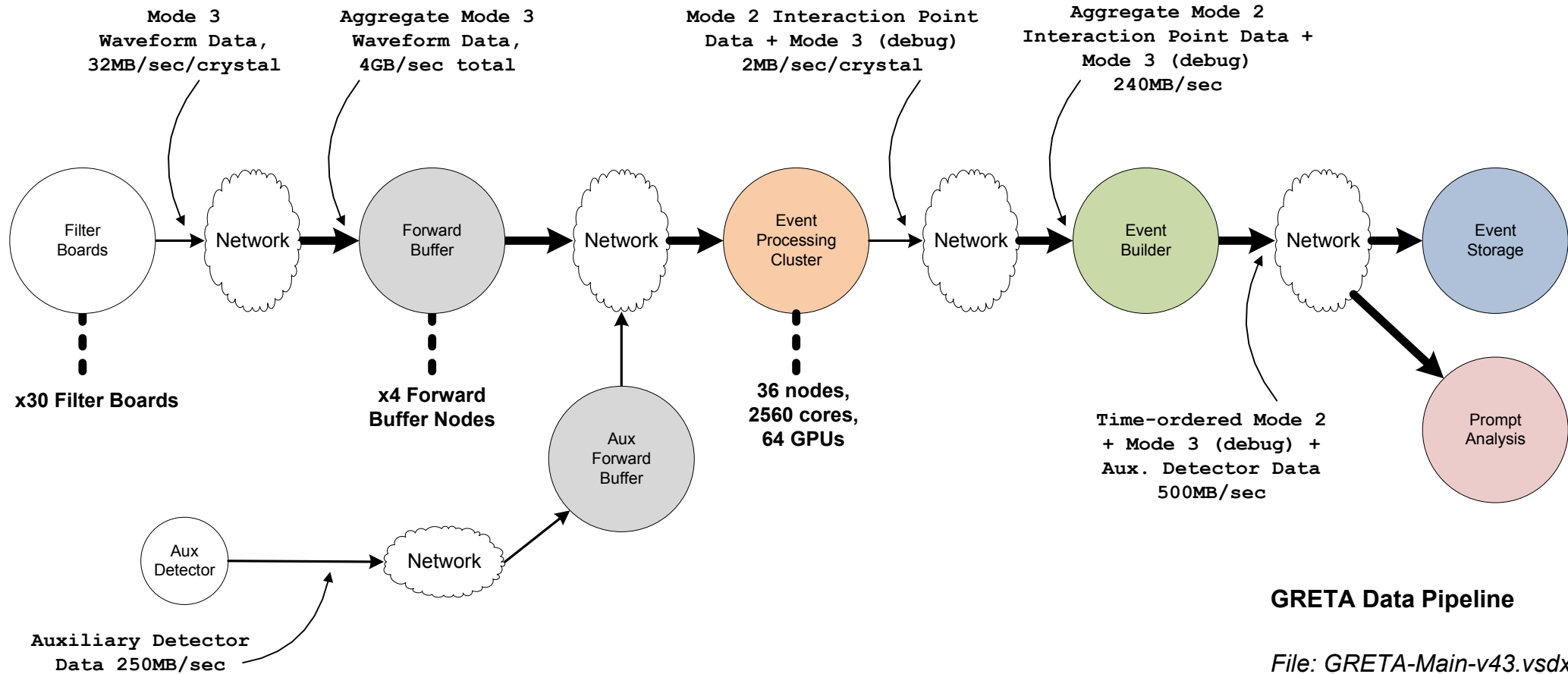
Complete - Ships summer 2025

**‘Greenfield’ design - not based
on a previous (high-level)
software stack**

GRETA Production Computing



GRETA's Data Pipeline



See: *Simple and Scalable Streaming: The GRETA Data Pipeline*, EPJ Web of Conferences 251, 04018 (2021)

Key Pipeline Components

• Forward Buffer:

- Mediates communication between the electronics and computing cluster
- Allows electronics to be free running and cluster nodes to self-schedule

• Event Processing:

- Locate interaction points from waveforms (signal decomposition)
- Each compute node presents a number of job slots which are assigned to a specific detector
- Hybrid CPU/GPU implementation

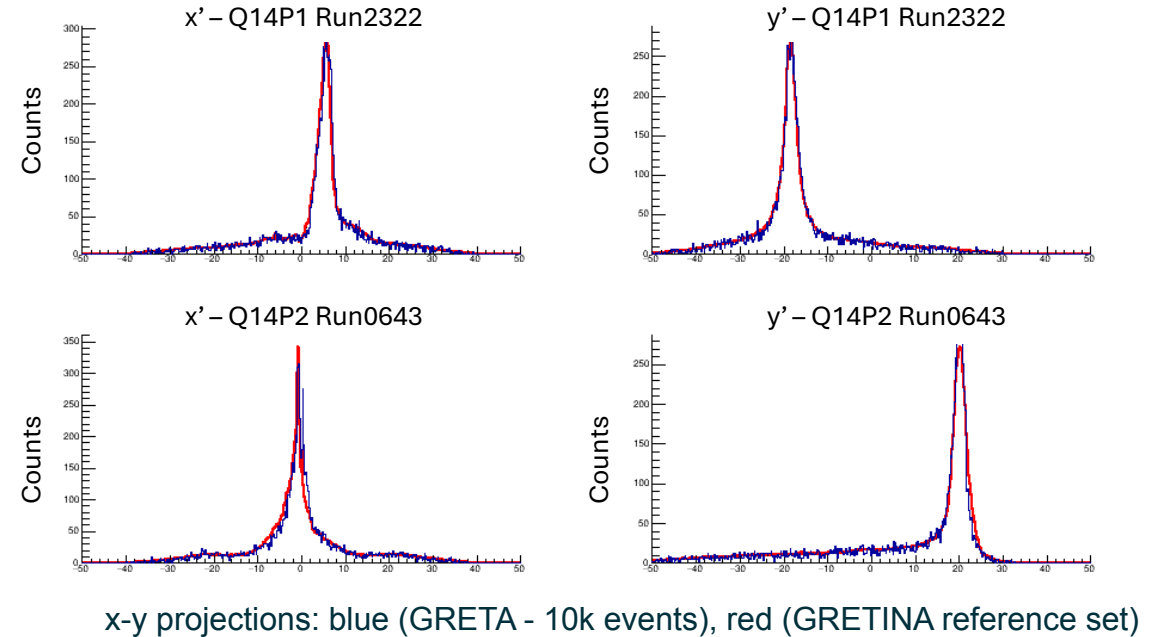
• Event Building:

- Aggregates events according to timestamps into global events
- Will provide a tap for online data monitoring

- Pipeline components transfer data via GAP (Greta Application Protocol) layer
- Based on nanomsg - zero copy, low latency, excellent stability (<https://nanomsg.org>)
- GAP implements standard communication design patterns (fan-in, fan-out)
- Meet performance specifications

Computing KPP Test

- Hour test over which time 1.85×10^9 compressed waveform messages were processed (14 TB ingest, single file of 636 GB of interaction point messages was generated)
- 80 event-processing containers were run across 40 nodes - 2560 workers/cores
- No events lost in signal decomposition or transport through the pipeline:
 - 1857633780 events at forward buffer ingress, 1857633780 decomposed events in file
- Aggregate rate of 510725 signal decompositions / s - meets objective KPP for computing
- 3 s end-to-end latency



Forward Buffer	Start (GMT)	Stop Request (GMT)	Stop (GMT)	Duration (s)	Total Msg In	Total Msg Out	Rate (Msg/s)
n00	23:09:53	0:10:30	0:10:30	3637	463775188	463775188	127516
n01	23:09:53	0:10:30	0:10:30	3637	465339834	465339834	127946
n02	23:09:53	0:10:30	0:10:31	3638	463581217	463581217	127427
n03	23:09:53	0:10:30	0:10:30	3637	464937541	464937541	127835
Aggregate					1857633780	1857633780	510725

Statistics gathered from 4 forward buffers in KPP test

Transport through the Pipeline - Routing Headers

- Each event (8k compressed for HPGe data) is sent as a UDP packet for electronics
- A type and subtype field determine the type of processing the event/message undergoes
- Events/messages are prepended with a routing header
- The payload is opaque except to the analysis components

```
struct routingHdr {  
    uint8_t version;    /* protocol version */  
    uint8_t flags;      /* for future use */  
    uint8_t type;       /* e.g. ge waveform */  
    uint8_t subtype;    /* message sub-type */  
    uint16_t length;    /* length of payload */  
    uint16_t seqnum;    /* sequence number */  
    int64_t timestamp; /* 48-bit, 10 nS clock */  
    int64_t checksum;  /* integrity checksum */  
} hdr;
```

GRETA routing header (version = 1)

Type: detector type (Ge, silicon, PPAC, ...)

Subtype: board channel

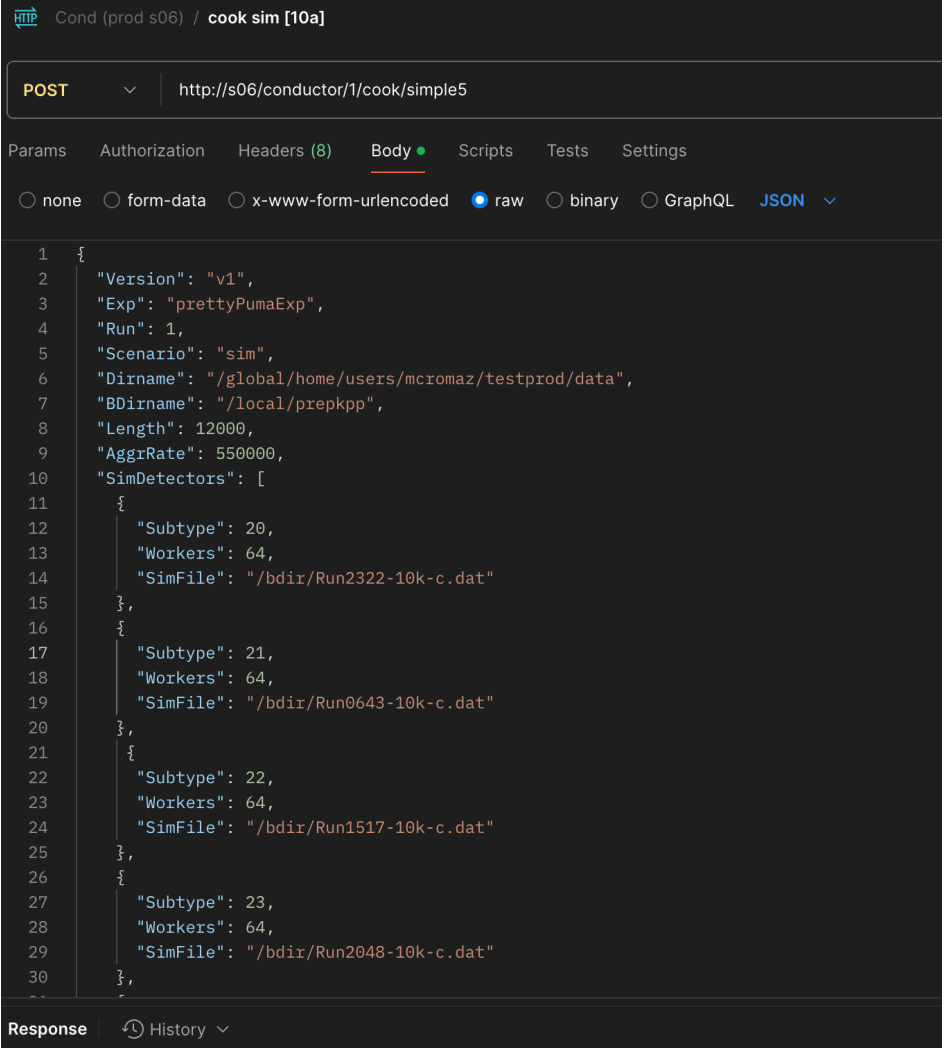
Different types, subtypes have different online analysis procedures or calibrations

What Is Orchestration in the GRETA Context?

- Instantiating of pipeline components needed for a given experimental situation
- Determining the hosts on which the pipeline is being deployed and the resources they are using
- The ‘wiring’ of the pipeline so data of the appropriate type gets routed to the appropriate pipeline components
- The distribution of user-level run control directives
- Directives to the electronics subsystem is the source of the data pipeline

Scenarios

- GRETA defines experimental scenarios
- Describes the experimental problem at hand (electronics test, pipeline test with replayed data, GRETA quads, ...)
 - Filter Board Test:
 - forwardBuffer → msgStore
 - Sim Scenario:
 - sim → forwardBuffer → decomp → eventBuilder → storage
- Represented by a json file - this is produced by a UI element or scripts
- Gives performance expectations (rates, message sizes) and expected types/subtypes



```
1 {
2   "Version": "v1",
3   "Exp": "prettyPumaExp",
4   "Run": 1,
5   "Scenario": "sim",
6   "Dirname": "/global/home/users/mcromaz/testprod/data",
7   "BDirname": "/local/prepkpp",
8   "Length": 12000,
9   "AggrRate": 550000,
10  "SimDetectors": [
11    {
12      "Subtype": 20,
13      "Workers": 64,
14      "SimFile": "/bdir/Run2322-10k-c.dat"
15    },
16    {
17      "Subtype": 21,
18      "Workers": 64,
19      "SimFile": "/bdir/Run0643-10k-c.dat"
20    },
21    {
22      "Subtype": 22,
23      "Workers": 64,
24      "SimFile": "/bdir/Run1517-10k-c.dat"
25    },
26    {
27      "Subtype": 23,
28      "Workers": 64,
29      "SimFile": "/bdir/Run2048-10k-c.dat"
30    },
31  ]
32 }
```

Postman API tool showing `sim` scenario with KPP parameters

Cooking

- Inputs:
 - the scenario
 - 'phosts' - pipeline hosts
- Output:
 - Config for each pipeline component

Cooking is the responsibility of the conductor

```
[  
  { "hostname" : "x02.greta.local", "hosttype" : "sim" },  
  { "hostname" : "n02.greta.local", "hosttype" : "network" },  
  { "hostname" : "c28.greta.local", "hosttype" : "compute" },  
  { "hostname" : "n03.greta.local", "hosttype" : "network" },  
  { "hostname" : "x00.greta.local", "hosttype" : "service" }  
]
```

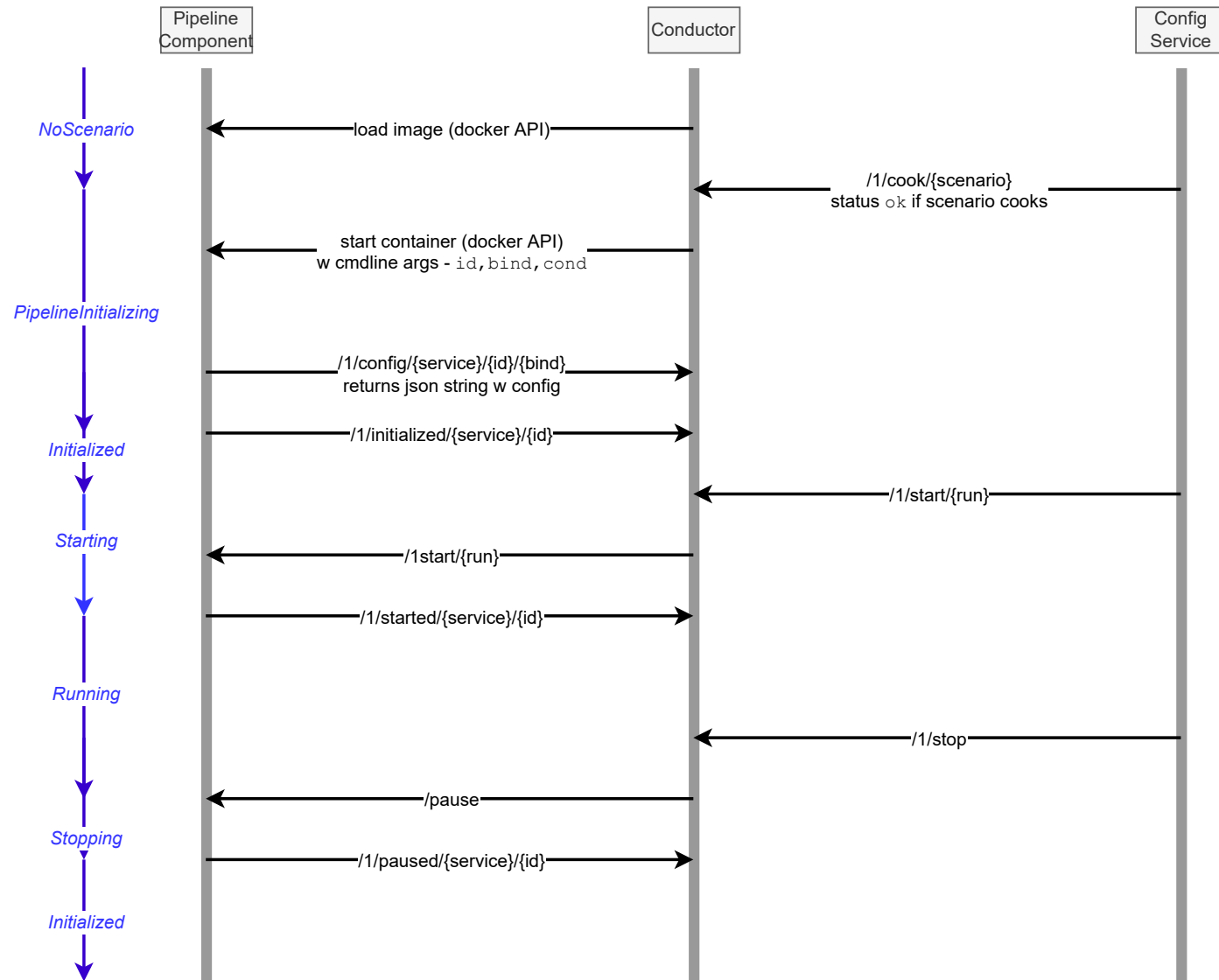
very simple phosts.json

```
[ { "Id": "dec000", "Name": "decsim", "Type": 4, "TypeName": "Decomp", "URL": "s06.greta.local/conductor", "Checksum": false, "PromBind": "", "DataNewProtocol": true, "NoSending": false, "NoEgressProcessing": false, "NoEgressQueueing": false, "NoSharedQueue": true, "SortingOnEgress": false, "LockFreeQueue": true, "QueueCapacity": 1000, "EnergyScalingFactor": 4, "EventBuilderAddr": "n03.greta.local:15000", "MaxBufferSize": 16000, "MaxDelay": 1000000, "Detectors": [ { "TypeId": 4, "Subtype": 20, "MaxMsgLength": 10000, "MaxRate": 400, "ForwardBufferAddr": "n02.greta.local:15003", "MaxResults": 100 } ] }
```

very simple event build config

Conductor

- Main control plane component - maintains system state
 - Instantiates pipeline components on nodes through the Docker engine API
 - Serves config to pipeline components once they come up according to the cook
 - Sends (properly sequenced) run control directives to pipeline components
- 2 API's:
 - 'User facing' ... UI's, scripting
 - 'Pipeline facing' ... config service



Pipeline Control Plane - Interacting Services

- Configuration Service
 - DB + API for configuration of pipeline
- Conductor
 - Generates component specific config for pipeline, orchestrates containers
- UI / Web app
 - User interface to config service, conductor, and run control
- Monitoring Service
 - Records status info from pipeline elements into time series DB (Prometheus)
 - Displays summaries to operator (Grafana)
- Works with GRETA Webapp / scripting concurrently for flexibility

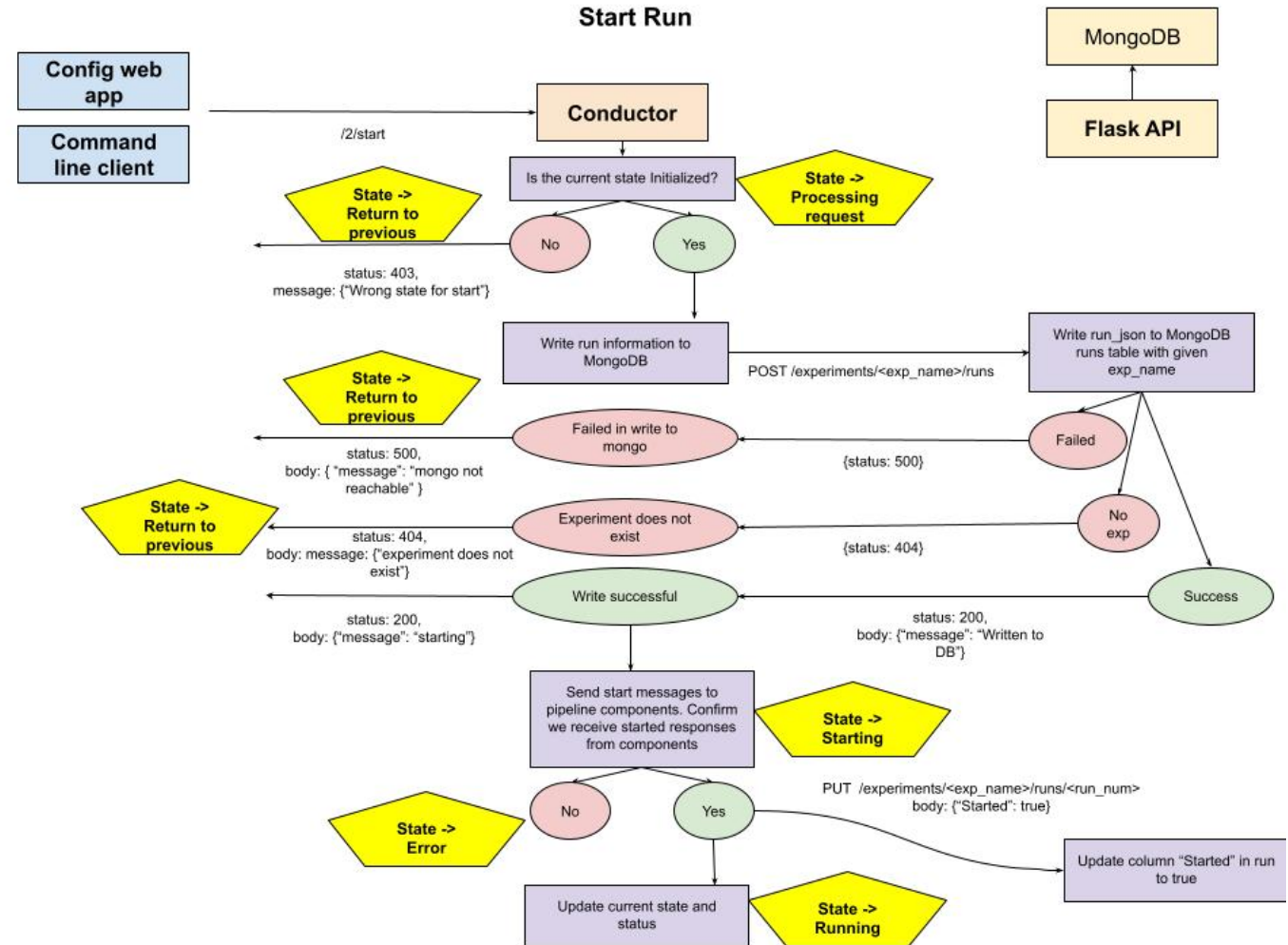
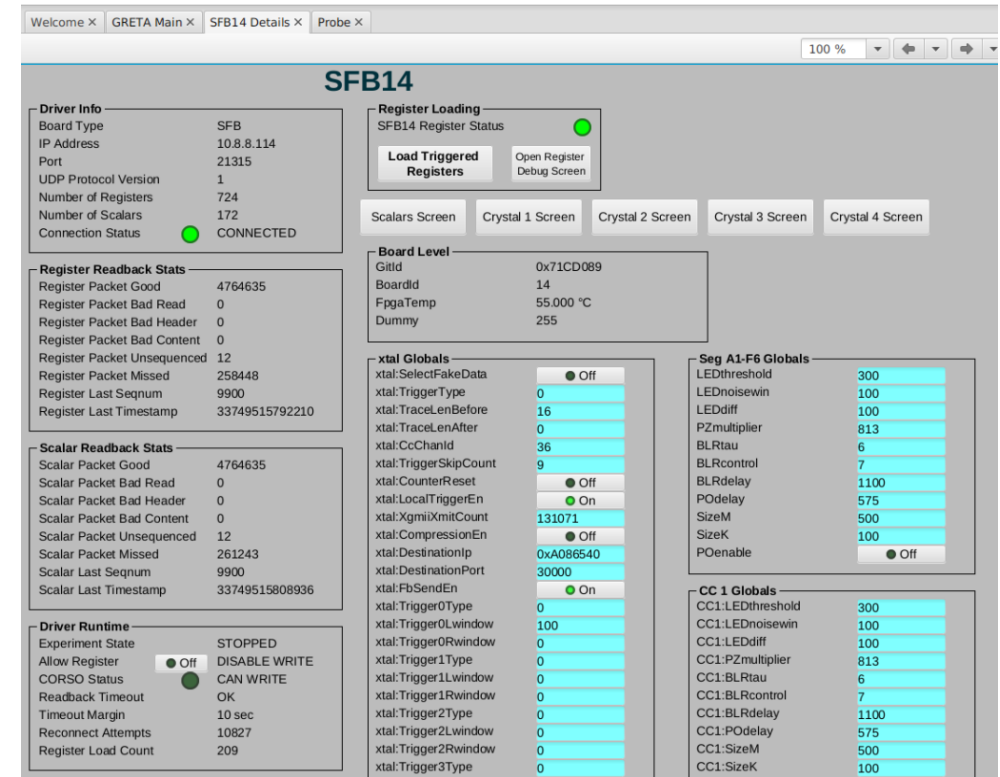


Diagram showing State Transitions / API calls for Run Start

Need to Orchestrate Electronics Too!

- Electronics controls are implemented using EPICS soft-IOCs running on Linux host(s)
- Developed specialized UDP driver reduce latency for operators given GRETA's (very large number of control points ~100000 registers)
- Conductor talks to electronics via web sockets interface:
 - Destination IP/Port of forward buffers in cluster
 - Flow control directives to trigger (if forward buffers alert)
 - Run management (reset board level scalars, ...)



Engineering screen showing operation of the UDP-based EPICS driver with a SFB.

Everything is a Container

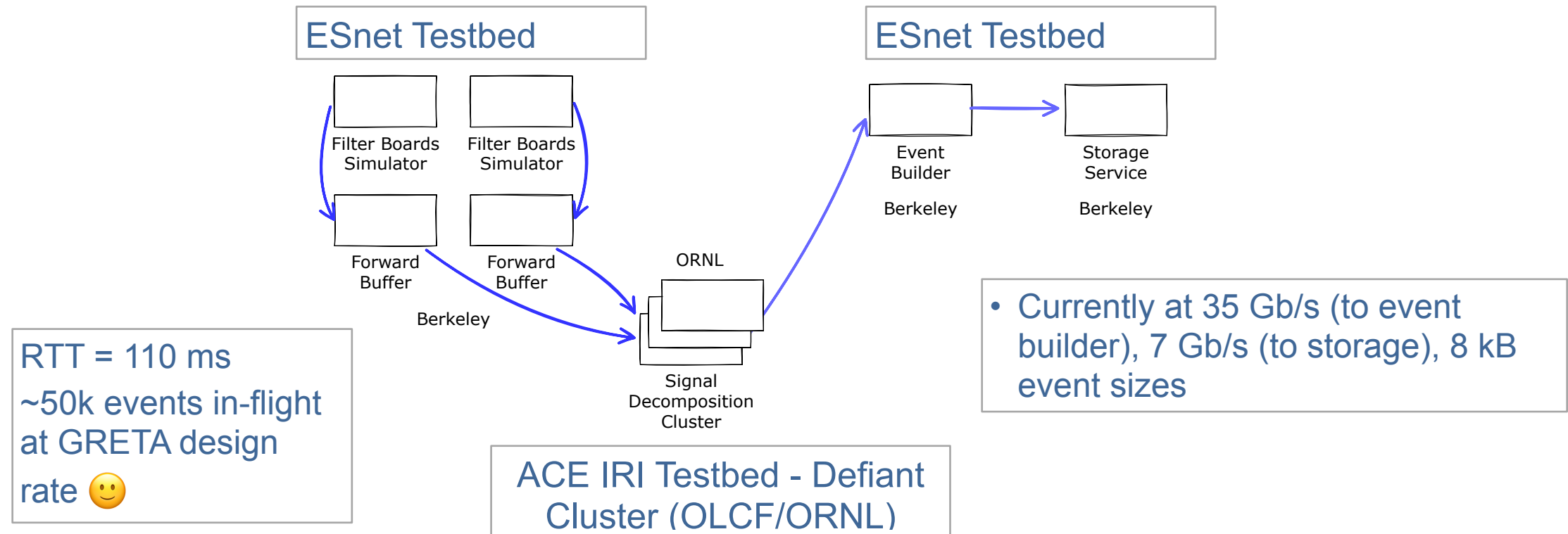
- Pipeline Components (forward-buffer, event processing, event builder, storage, ...)
- Control Plane (conductor, config service, monitoring service)
- Electronics Controls (soft IOC's, HLC's)

- GRETA build system oriented towards containers
- Local docker repo (allows for system isolation)
- Local CI system (Gitlab)

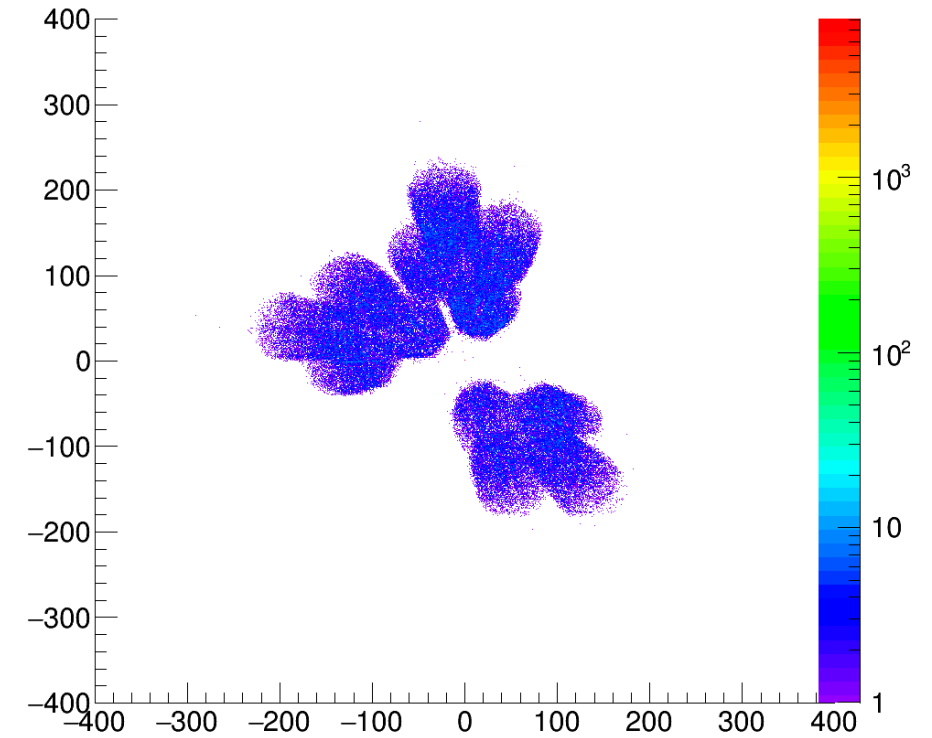
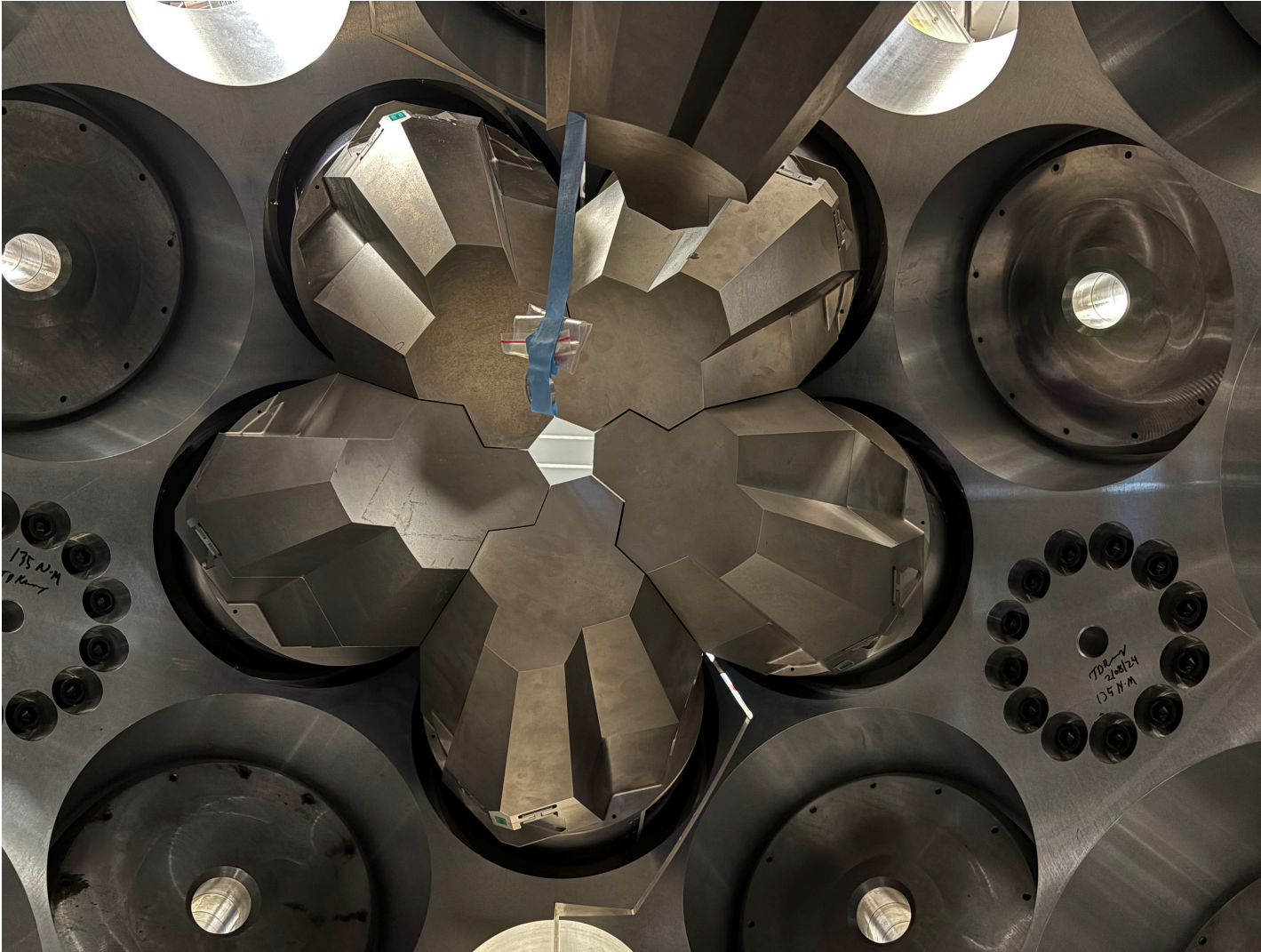
- Orchestrator responsible for container config (bind mounts, pinning containers to cores, ...)

Deleria - The GRETA Pipeline over Wide Area Networks

- Topic of area-wide LDRD at LBNL in collaborations with OLCF/ORNL (Deleria project)
- Bandwidth testing with passthrough container - achieved very high transfer rates



3 Quad Test (12 detectors)



Orchestrating Across Facilities

- Much more complicated (and I don't think we understand it yet)
- Facility API's used to instantiate remote containers:
 - cond → facility API → slurm → container start
- 'Wiring'
 - Flat address space (now, maybe future)
 - Proxies (future)
 - Working with Argonne 'SciStream' group

Administrative > Technical

Lessons Learned

- We underestimated the complexity of orchestration (we took this as a low risk activity - did not formally prototype it)
 - It was the only major refactor we needed to do in the project (to support simultaneous support of scripting with the UI)
- Delays in control plane deployment led to delays in system-level testing

People

- System Architect **[Eli Dart - ESnet, LBNL]**
- High-performance data plane components (forward buffer, global event builder) **[Eric Pouyoul - ESnet, LBNL]**
- Scientific programming (signal decomposition) **[Gustav Jansen - NCCS / Physics, ORNL]**
- Cluster deployment, Development environment **[Tin Ho - Scientific IT group, LBNL]**
- Control system engineering **[Tynan Ford - Engineering division, LBNL]**
- CAM, Scientific contact **[Mario Cromaz - NSD, LBNL]**