





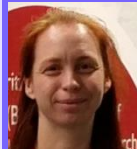



































# Data-awareness with Maestro middleware in Climate Digital Twin workflows

Utz-Uwe Haus, Christopher Haine, [Ali Mohammed](#)  
HPE EMEA Research Lab

21<sup>st</sup> ECMWF HPC workshop in meteorology, Bologna  
2025-09-19

# EMEA Research Lab Team overview

<div><div><b>Utz-Uwe Haus</b> Head of EMEA Research Lab (ZH)</div></div>				
<div><div><b>Harvey Richardson</b> (EPCC)</div></div>	<div><div><b>Tim Dykes</b> (Br)</div></div>	<div><div><b>Jess Jones</b> (Br)</div></div>	<div><div><b>Holly Judge</b> Archer2 (UK)</div></div>	<div><div>NN Isambard AI (UK)</div></div>
<div><div><b>Aniello Esposito</b> (ZH)</div></div>	<div><div><b>Alfio Lazarro</b> (L/Ba)</div></div>	<div><div><b>Alessandro Rigazzi</b> (L/Ba)</div></div>	<div><div><b>Ali Mohammed</b> (Ba)</div></div>	<div><div>NN (3.11.) Isambard AI (UK)</div></div>
<div><div><b>Irene Ferrario</b> Prj. Mngr. (Ba)</div></div>	<div><div><b>Ahmed Eleliemy</b> OpenCUBE (Ba)</div></div>	<div><div><b>Christopher Haine</b> (Ba)</div></div>	<div><div><b>Tiziano Müller</b> (ZH)</div></div>	<div><div><b>Sebastien Cabaniols</b> (Gr)</div></div>
<div><div><b>David Brayford</b> (MUC)</div></div>	<div><div><b>Sergei Shudler</b> BlueLion (1.11.) (MUC)</div></div>	<div><div>NN BlueLion (DE)</div></div>	<div><div>NN Daedalus (GR)</div></div>	



ERL (summer) internship program



# Maestro

<https://gitlab.com/maestro-data/maestro-core>



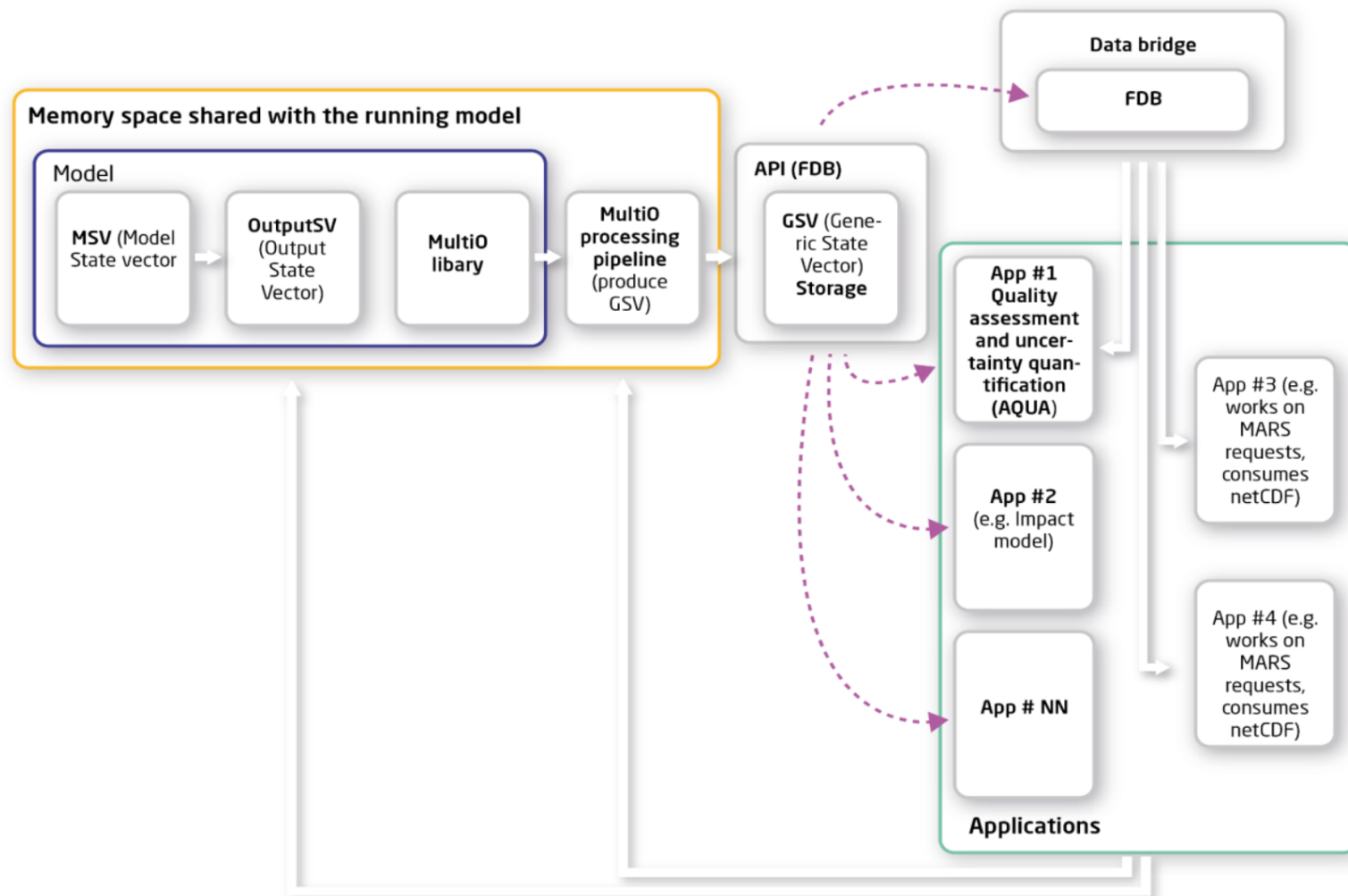
# Destination Earth: Climate DT workflow

## Today's bottleneck

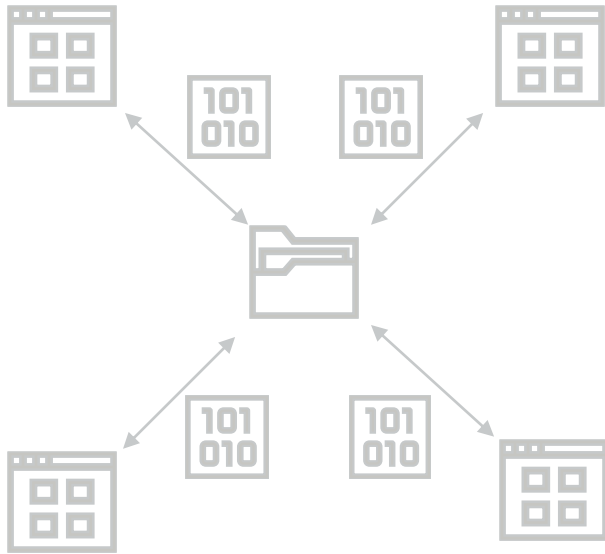
- Data movement between forecast stages and product generation
- Archiving via I/O aggregator nodes into PFS
- Each product generation job is reading from PFS

## Vision

- Speed up data-movement
- More flexible dependencies
- Improve streaming workflow execution and scheduling

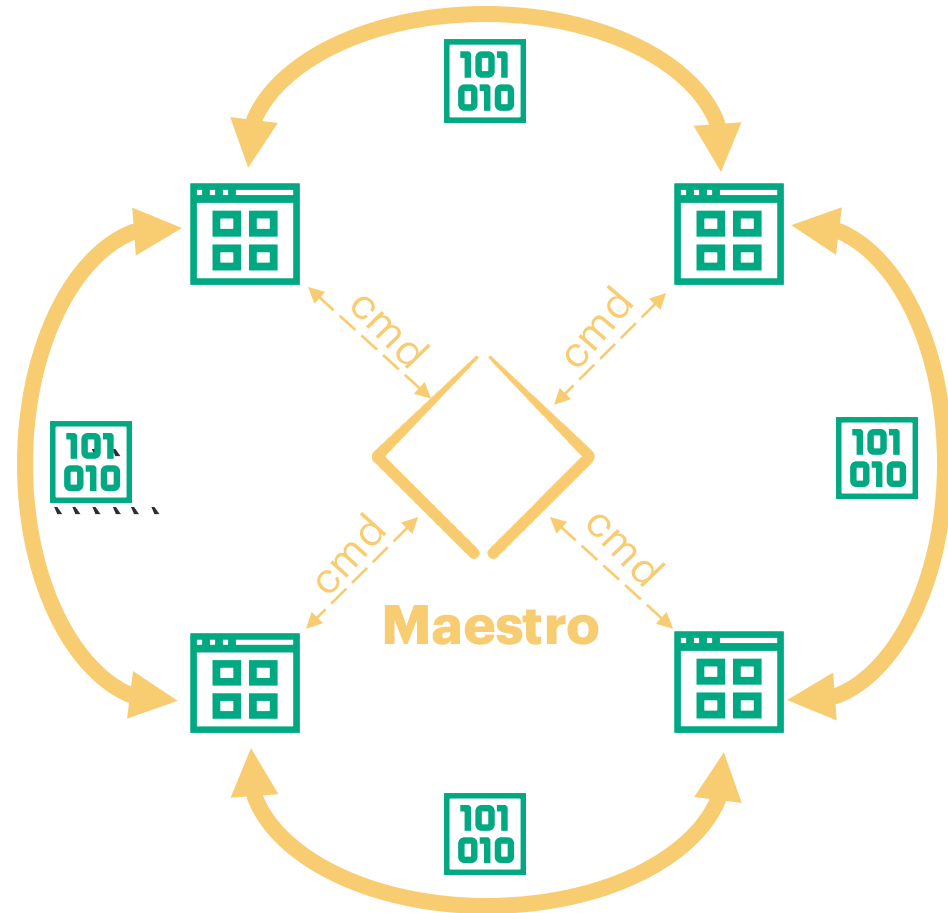


# Application Coupling



## Traditional

Central data repository (PFS, Database) or tightly integrated coupling framework (MPMD or split Comm)



## Maestro-enabled

Data object 'Marketplace', peer-to-peer data transfer, cross-process



# Overview/Architecture: Maestro in a nutshell

CDO

## CDO (Core Data Object)

It is at the heart of Maestro's design and is used to encapsulate data and metadata. Supports dependencies.

## OFFER+WITHDRAW

Applications OFFER CDOs to the management pool. Maestro manages the data, until WITHDRAW occurs.

## REQUIRE+DEMAND

When an application REQUIRES a CDO, Maestro makes data available. At DEMAND is hands over resources containing the data and relinquishes all control it.

Scope  
Object

## SCOPE OBJECT

Captures information about scope, size, access relations and schedules of the data to enable efficient movement and/or transformation

## MAESTRO SYSTEM MODEL

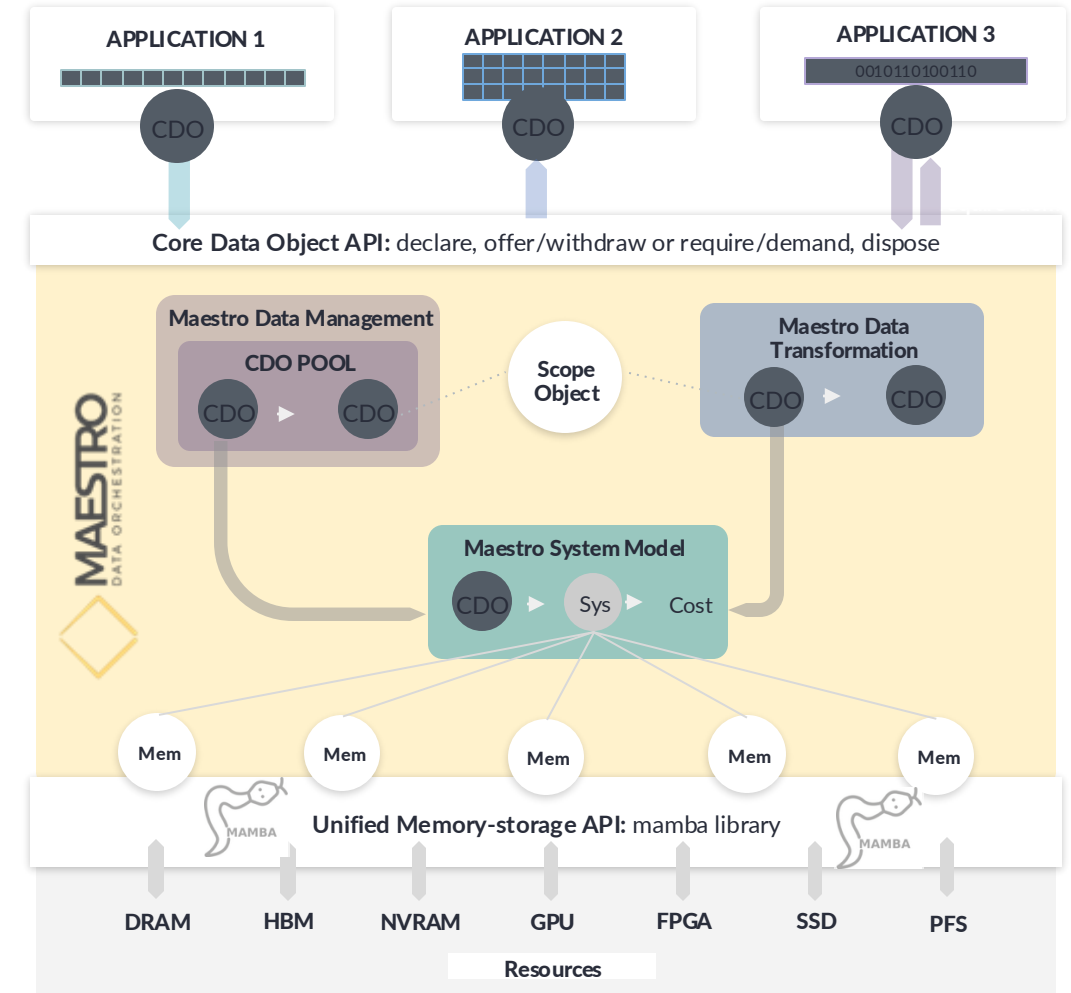
Computes the cost of moving, transforming or copying data a CDO

Sys



## SYS

Interface to every memory level, enabling core functionality of that memory via mamba library.



# Low intrusiveness

## Producer side

```
cdo = mstro_cdo_declare("name")
```

- same name = same object

```
mstro_cdo_attribute_add(cdo, key, val)
```

- Important: size, layout, (distribution), data reference
- Optional: user-defined attributes

```
mstro_cdo_offer(cdo)
```

- At this point all other workflow participants can access `cdo`

```
mstro_cdo_withdraw(cdo)
```

- may block, async variant available

```
mstro_cdo_dispose(cdo)
```

## Alternatives

- Batch up CDOs in a CDO Group (for batched OFFERs)

## Consumer side

```
cdo = mstro_cdo_declare("name")
```

- same name = same object

```
mstro_cdo_attribute_add(cdo, key, val)
```

- Important: size, layout, (distribution), data reference
- Optional: user-defined attributes

```
mstro_cdo_require(cdo)
```

- At this point reference to a suitable source for CDO will be established

```
mstro_cdo_demand(cdo)
```

- may block, async variant available

```
mstro_cdo_dispose(cdo)
```

## Alternatives

- Subscribe to pool events (like offer, require, withdraw) and act on them
- Create CDO Group based on attributes (think: SQL SELECT) and iterate on them

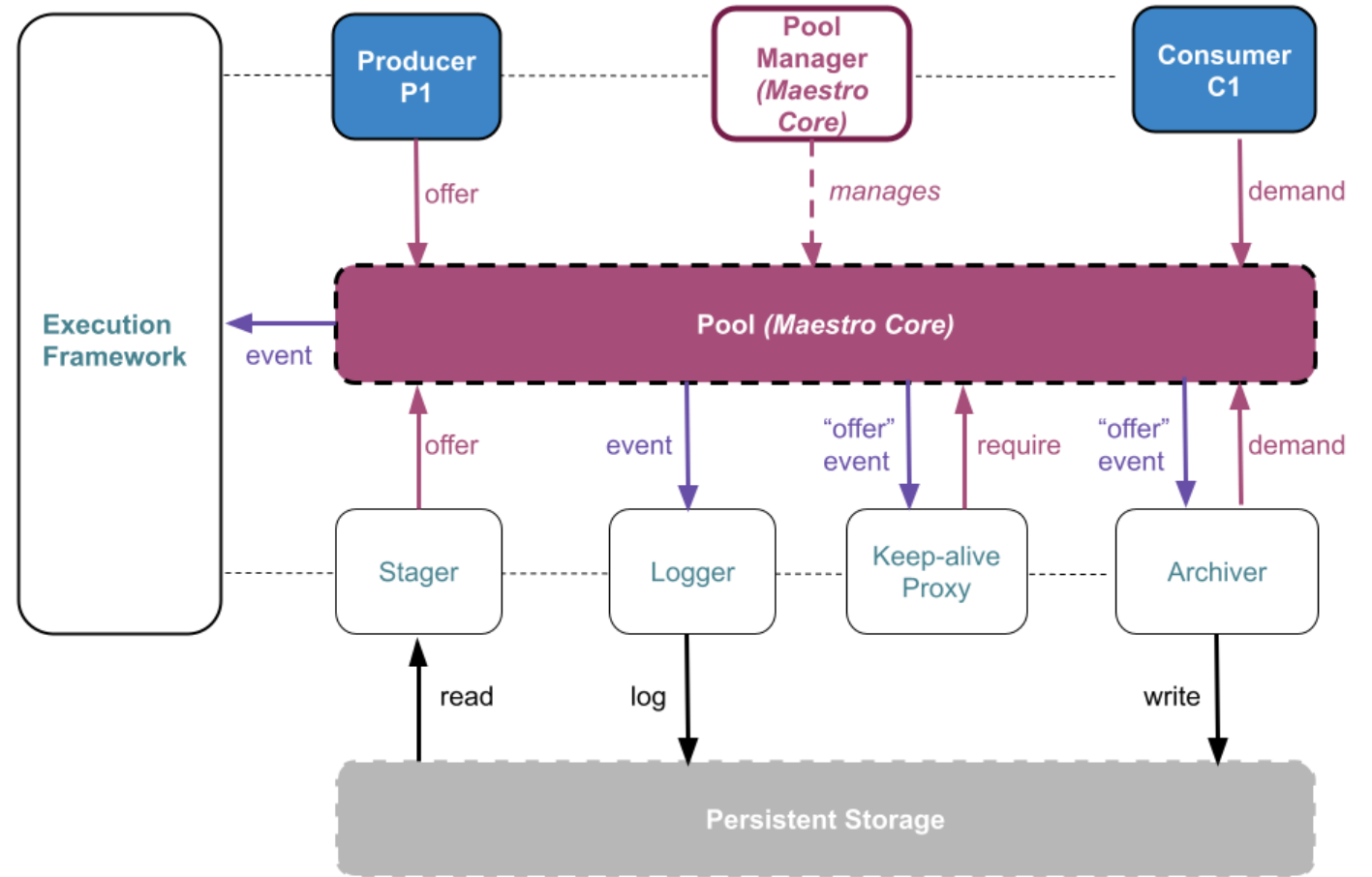
# Data- and Memory-aware workflows with maestro

Applications coupling **bypassing filesystem** intermediary.

Pool events allow the implementation of useful **workflow components**.

No programming paradigm or memory management layer enforced, but utilizes and can take advantage of Mamba memory management library

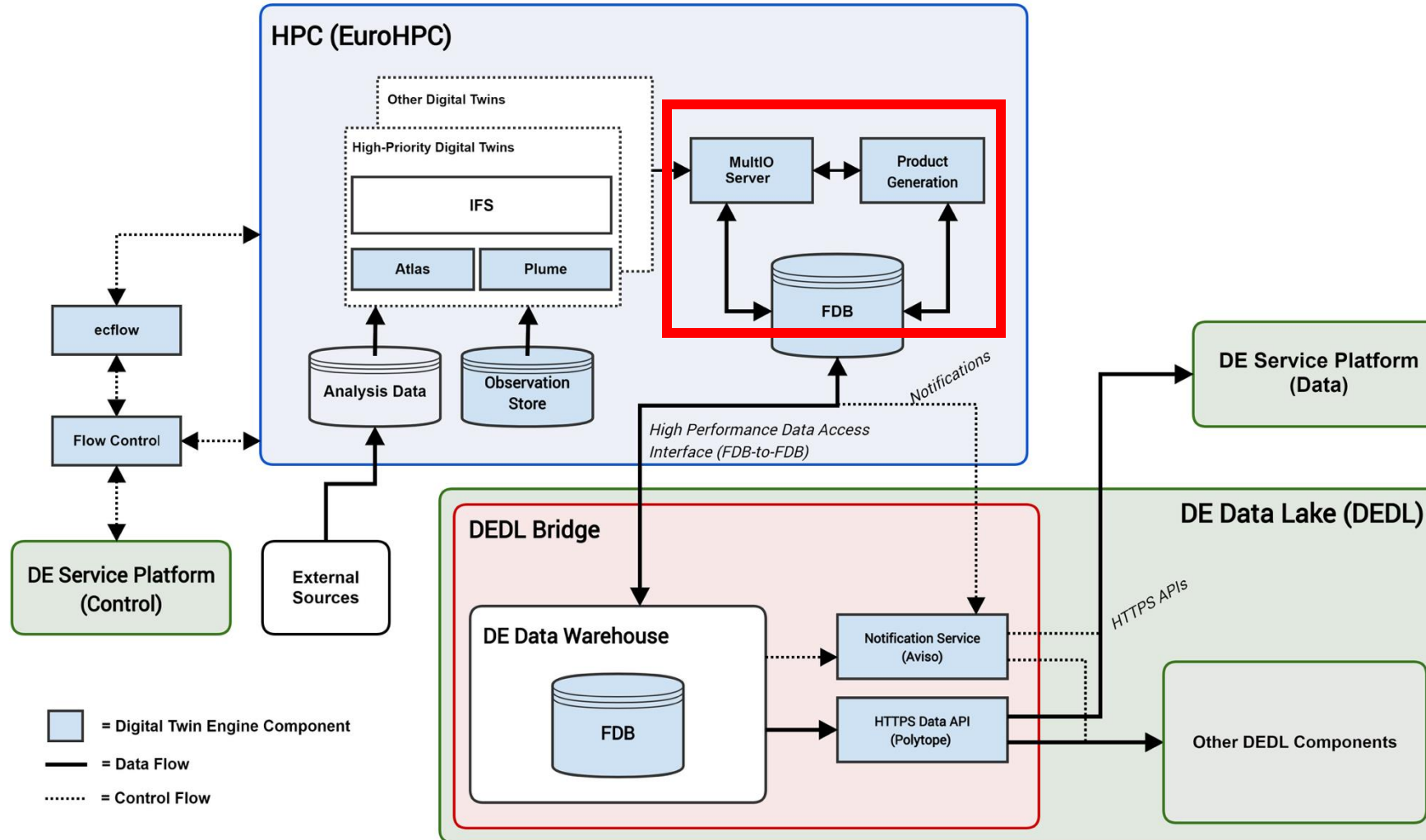
(<https://gitlab.com/cerl/mamba> )





# Climate DT use case

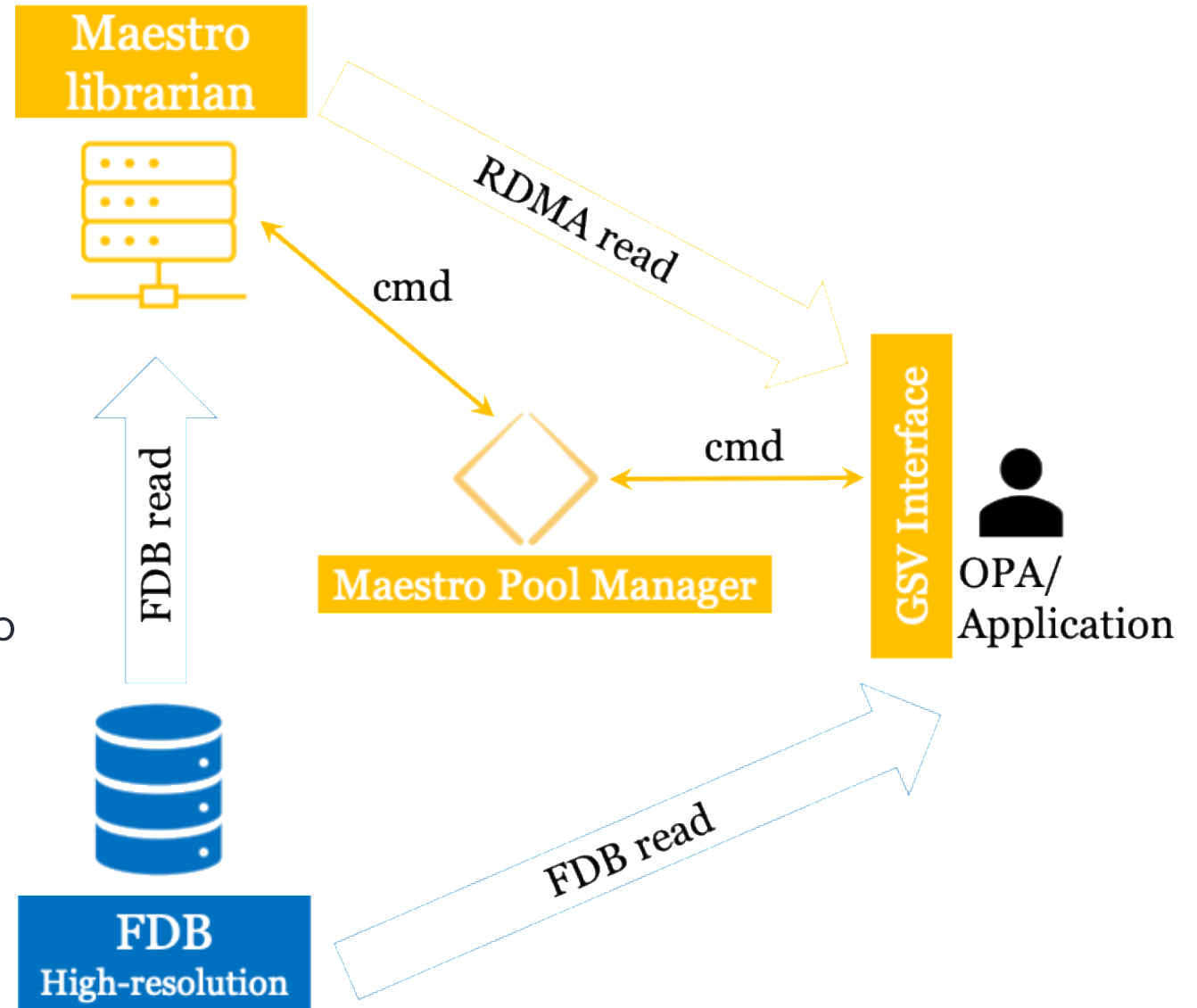
# Digital Twin Engine Dataplane



# Maestro-enabled Workflow

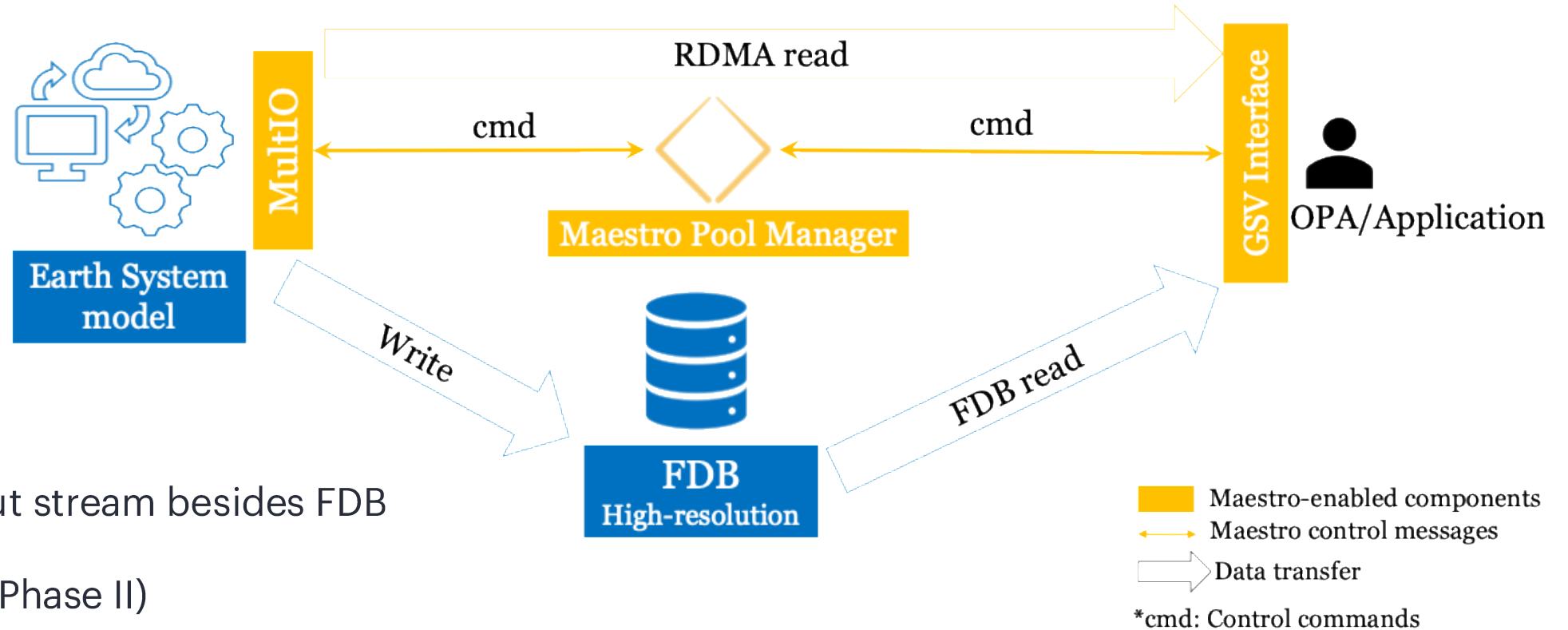
## Application Workflow

- Maestro librarian component
  - Read data from FDB
  - Cache/stage climate DT data
- Python interface
  - Data and meta data management
  - Support for downstream applications
- GSV Schema
  - Supports climate data formats in Maestro
- GSV interface
  - Seamless integration
  - Maestro engine backend besides FDB



# Maestro-enabled Workflows

## End to End Workflow

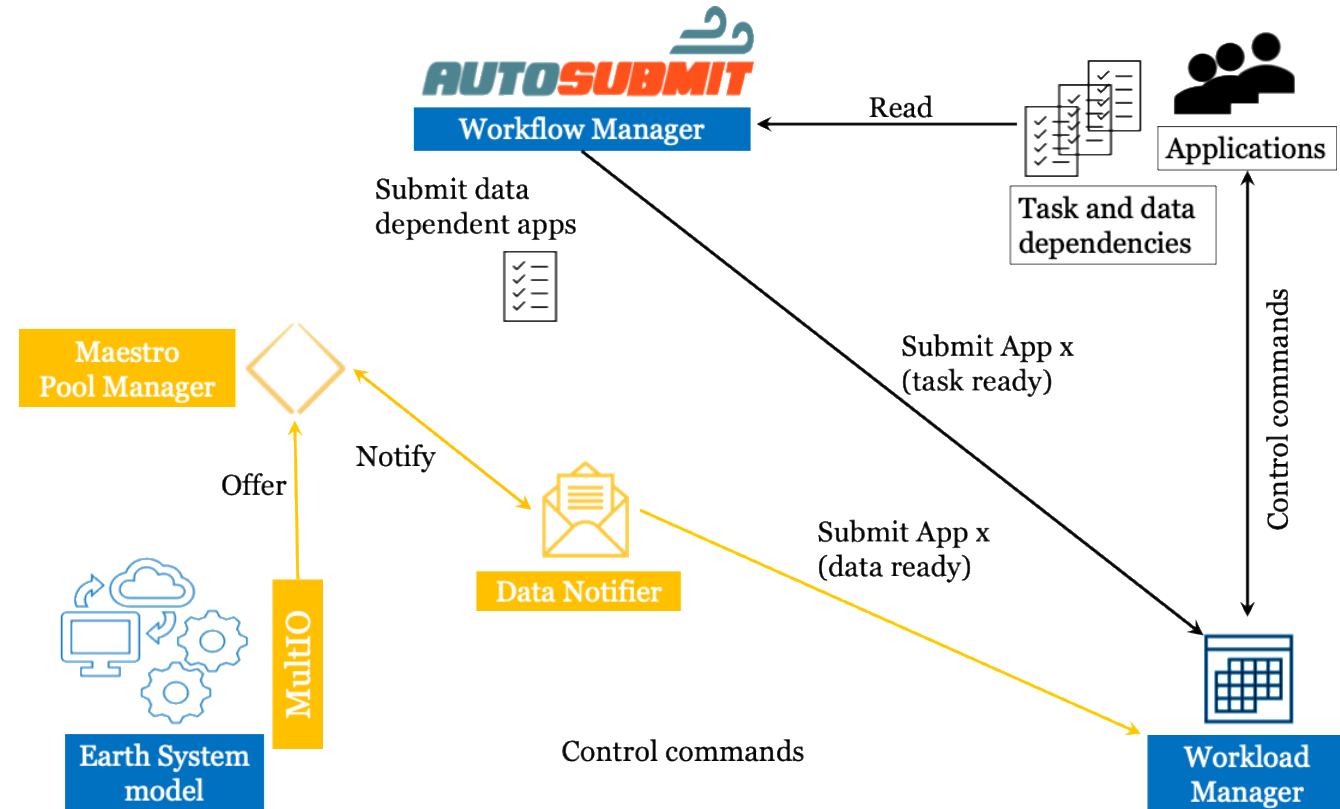


- MultIO integration
  - Maestro output stream besides FDB
- ICON integration (Phase II)

# Maestro-enabled Workflow

## Workflow Data Notification Integration (WIP)

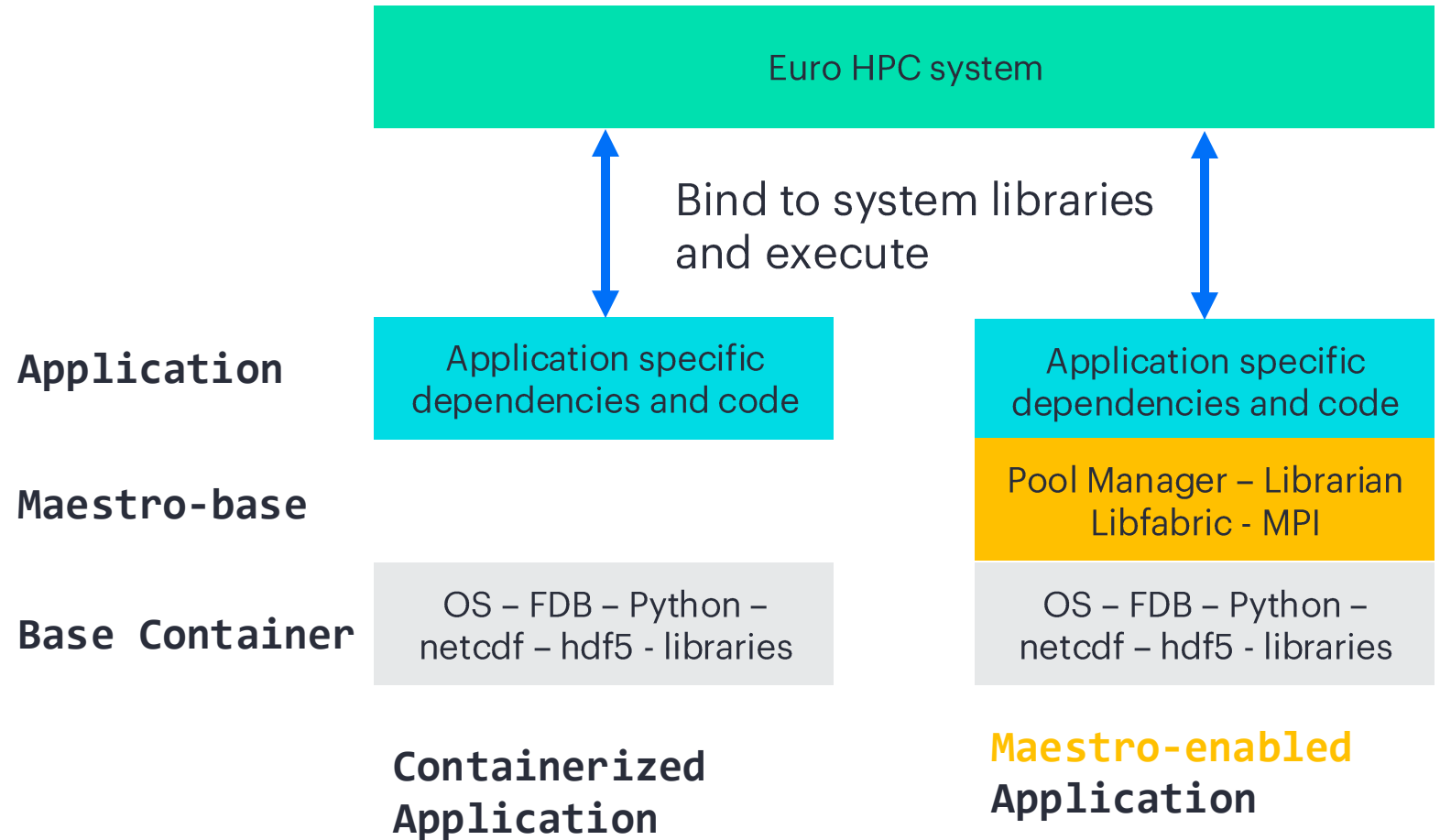
- Concept
  - Separate data from job dependencies
  - Improve scheduling
  - Improve resource utilization
- Integration with Autosubmit
  - Annotate dependencies data/job
  - Maestro “platform”
- Data notifier component
  - Subscribe to data events
  - Launch jobs on data availability



# Maestro-enabled Workflow

## Containers infrastructure

- Hierarchical design
- Maestro container based on ClimateDT base containers
- Provide Maestro libraries and Maestro main components
- Bind and use native HPC fabric interconnect
- Applications can easily build upon Maestro-base and adopt Maestro



# Fault tolerance

# Fault tolerance considerations and design

## Assumptions:

1. Failure of **Maestro pool manager** is **fatal**, similar to the **workflow manager** (requires full restart)
2. Failure is not caused by users/**misuse**.

## Architectural Resiliency :

1. **Loosely** coupled components (clients join/leave during execution)
2. Automatically **retry** certain libfabric calls (e.g., export **MSTRO\_OFI\_NUM\_RETRY=10**)
3. **Redundancy** of transport **methods** (RDMA, GFS, OpenFAM, Object store (MIO/FDB))
4. **Redundancy** of sources for the same **data** (core data objects/CDOs)
5. **Redundancy** of network **endpoints** (alternative links)

## FT Design Architecture:

- Two-Policy System: **Fail-fast** (abort immediately) vs. **Best-effort** (retry with alternatives)
- State-Driven Tickets: state machine tracking transfer progress from creation to completion
- Timeout-Based Detection: PM monitors transfers and triggers failure handling when timeouts expire
- Message-Based Recovery: protobuf **TransferFailed** messages to coordinate failure handling

## Timeout Configuration:

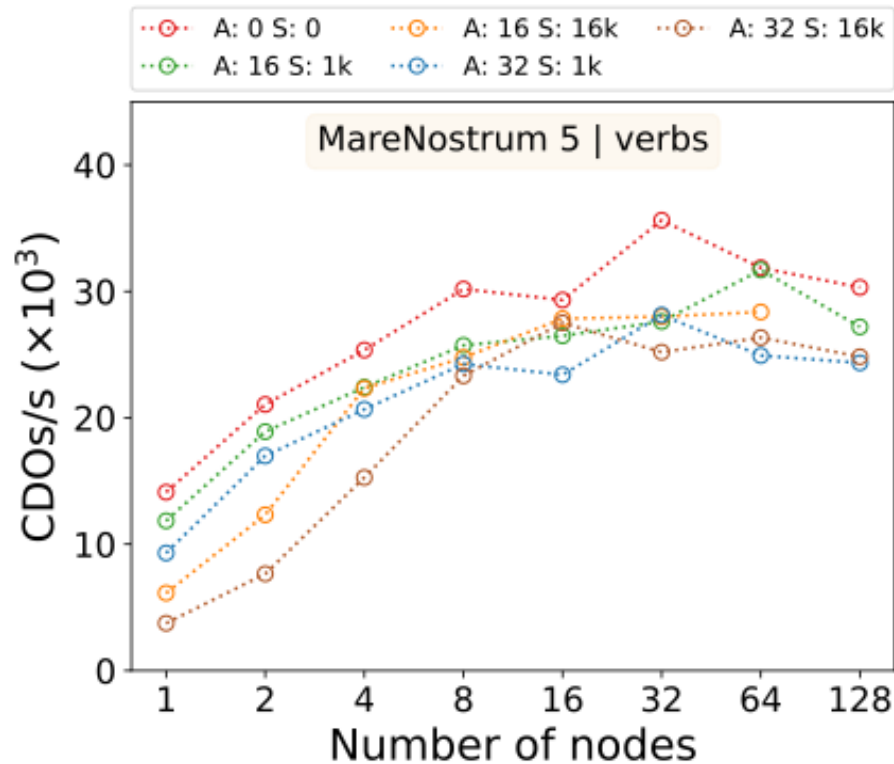
- **MSTRO\_FT\_TIMEOUT**: general timeout for operations
- **MSTRO\_FT\_ACK\_TIMEOUT**: for event acknowledgments
- **MSTRO\_FT\_CDO\_TIMEOUT**: for CDO transfer operations



# Raw performance

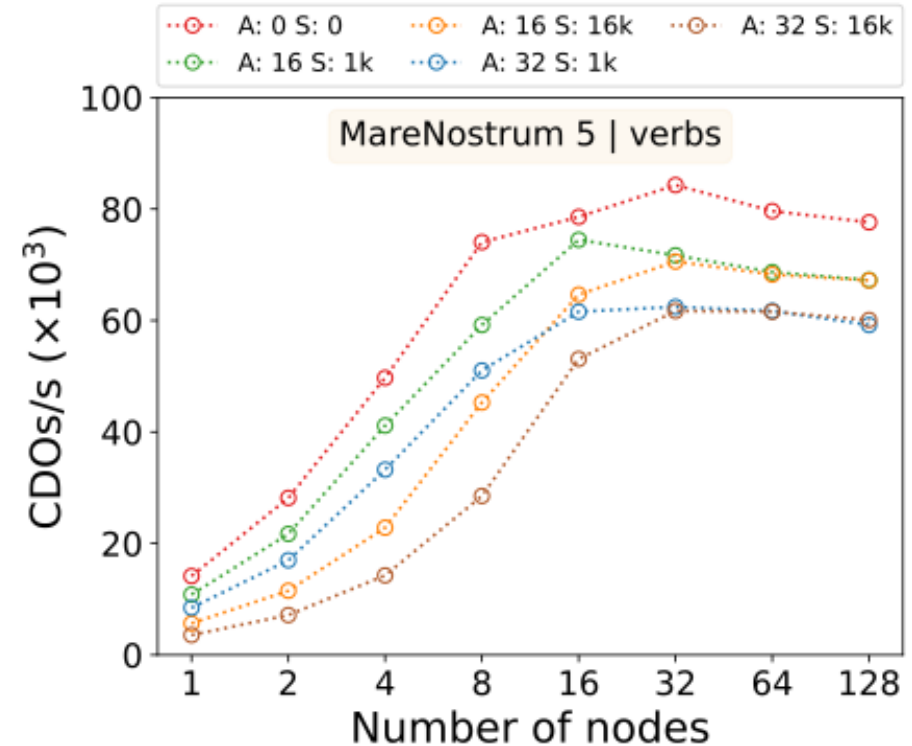
# Synthetic Performance benchmarks

## Handling CDOs (declare/offer)



Single OFI, single OP thread

Almost 85k CDO-ops/s with 4 threads operation threads of two Maestro pool manager.



Two OFI, four OP thread

A: Number of attributes  
S: Size of attributes in bytes  
#nodes: Clients talking to the PM

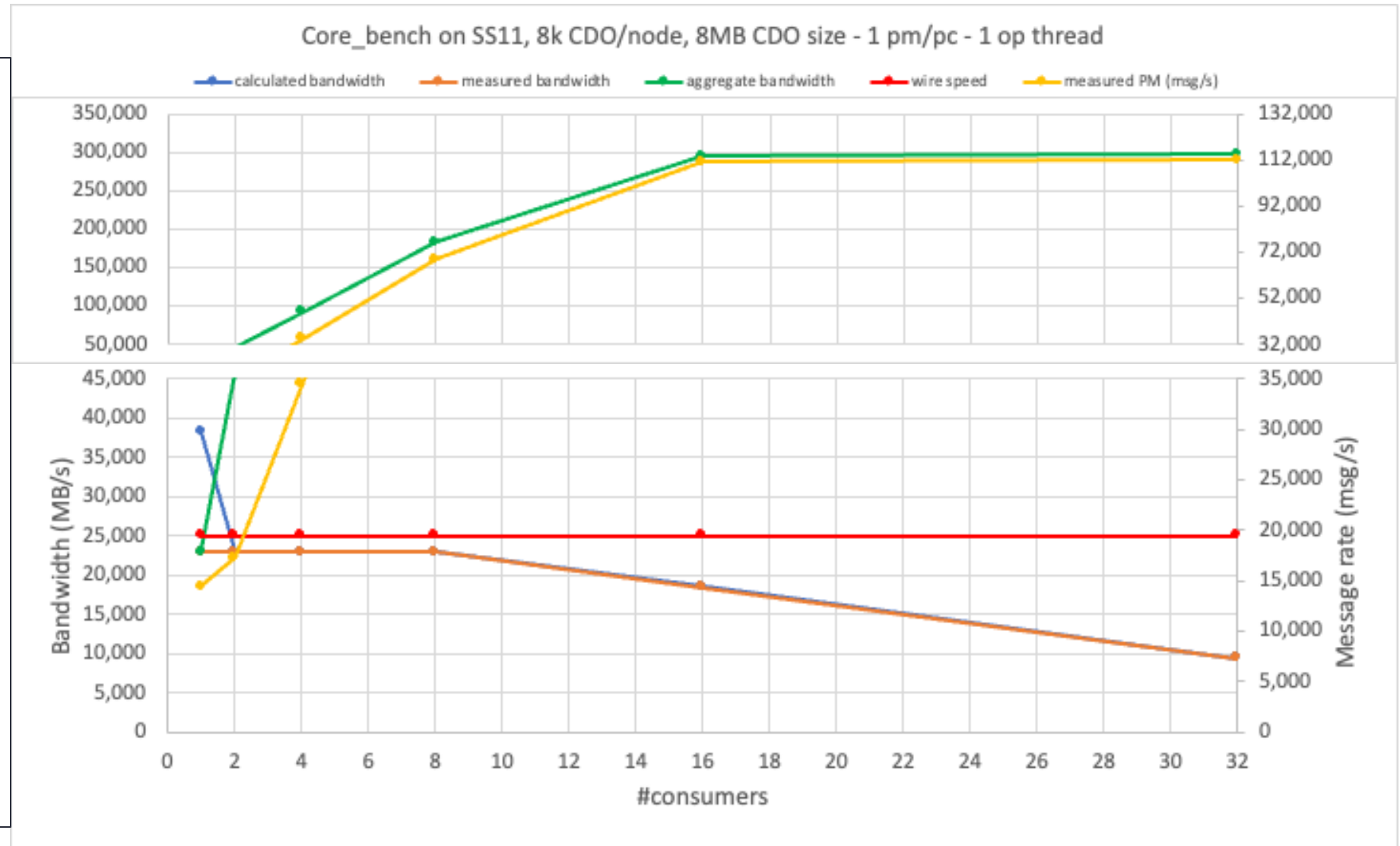
# Synthetic Performance benchmarks

## Transport Bandwidth

Aggregated bandwidth:  
300 GB/s at 110k msg/s  
processed by a single  
thread of the pool  
manager.

Wire speed up to 8  
consumer nodes.

Model bandwidth  
based on #msg/s on the  
PM:  
 $(3 \text{ msg/CDO}) \times \text{CDO\_size}$



# Thank You

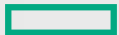
ali.mohammed@hpe.com



<https://gitlab.com/maestro-data/maestro-core>



# backup



# Maestro components:

## Software Container base

- = Climate DT base + Maestro library (C/C++/Python) + Maestro Pool Manager and Librarian components

## Tests on platforms

- LUMI: App workflow (Librarian) and Model workflow (IFS-Nemo)
- MN5: App workflow (Librarian), Model workflow (IFS-Nemo), Maestro/Autosubmit Data Notifier prototype

## Phase 1 deliverables outcome

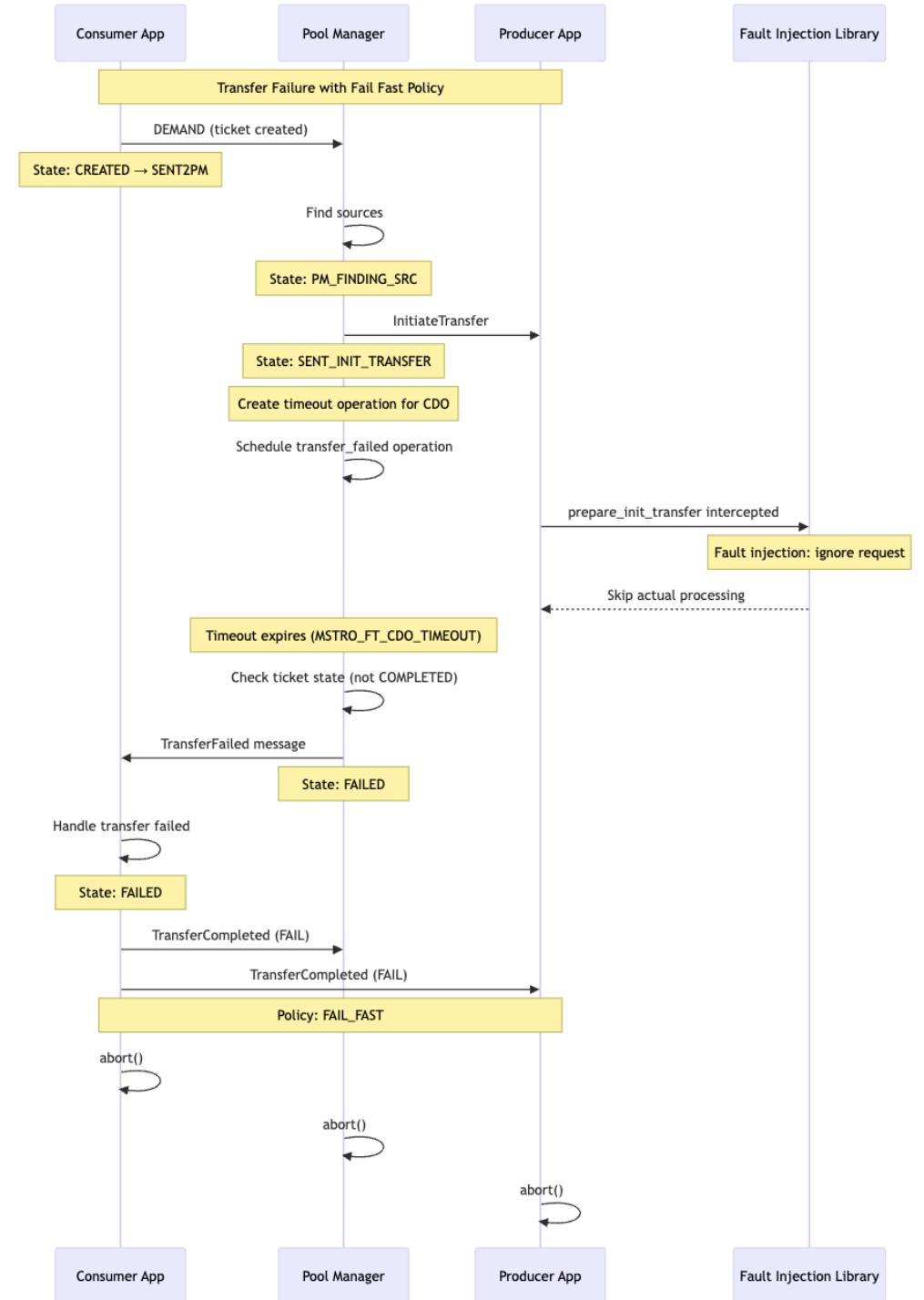
- Maestro Librarian/FDB, Python; Integration in GSV Interface, GSV schema; Climate DT workflow integration: app, model;

## Phase 2 deliverables goals and status

- Fault Tolerance (test done), Maestro Data Notifier (prototype done), OPA integration (prototyping), LLM-based system (on track)
- 3 models benchmarking (currently focusing on IFS-Nemo)
- Application integration (early candidates: energy indicators (prototyping), AQUA (under discussion))

# Fail-fast mode (first results)

- Fault injection by **LD\_PRELOAD** an injection library
- Intercept **init\_transfer** ticket
  - Simulate failed producer, network issues, etc ...
- The state of each transfer is monitored by the PM, Consumer, Producer, transitioning through a state machine
- All components check the state of the ticket before processing
- Based on the state of the ticket, certain cleanup maybe performed on failure.



# Fencing mode (controlled failure mode)

