Modularity

NumPEx

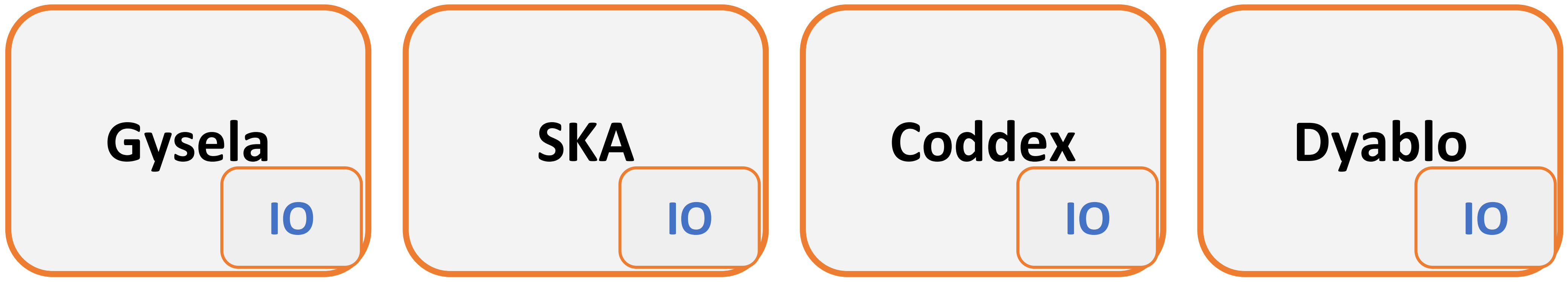
Exa-DoST – Nov 2025 - Bordeaux

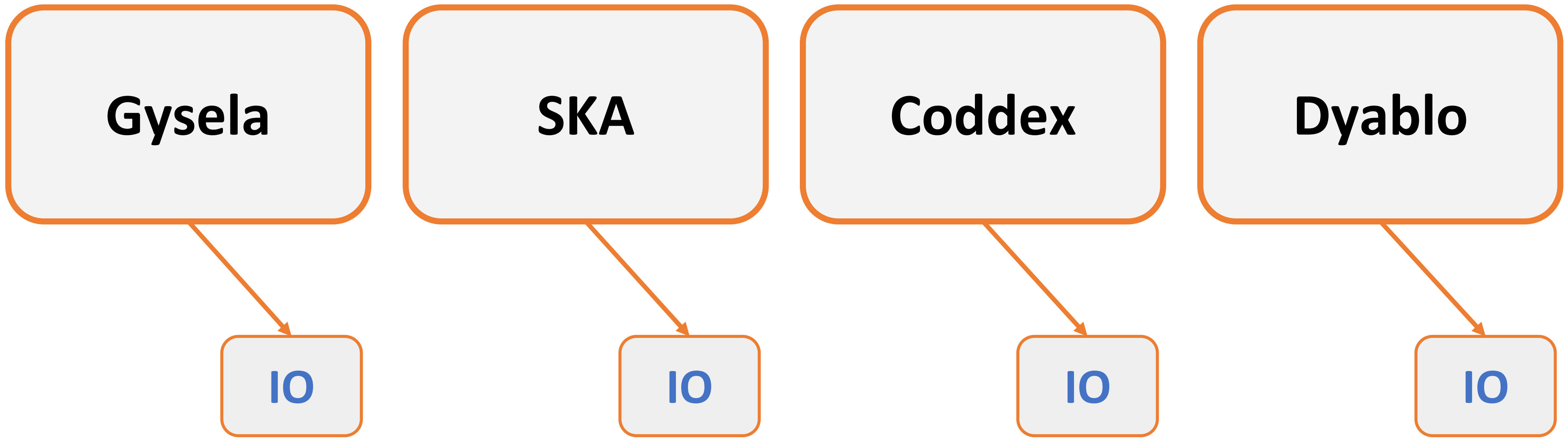
Gysela

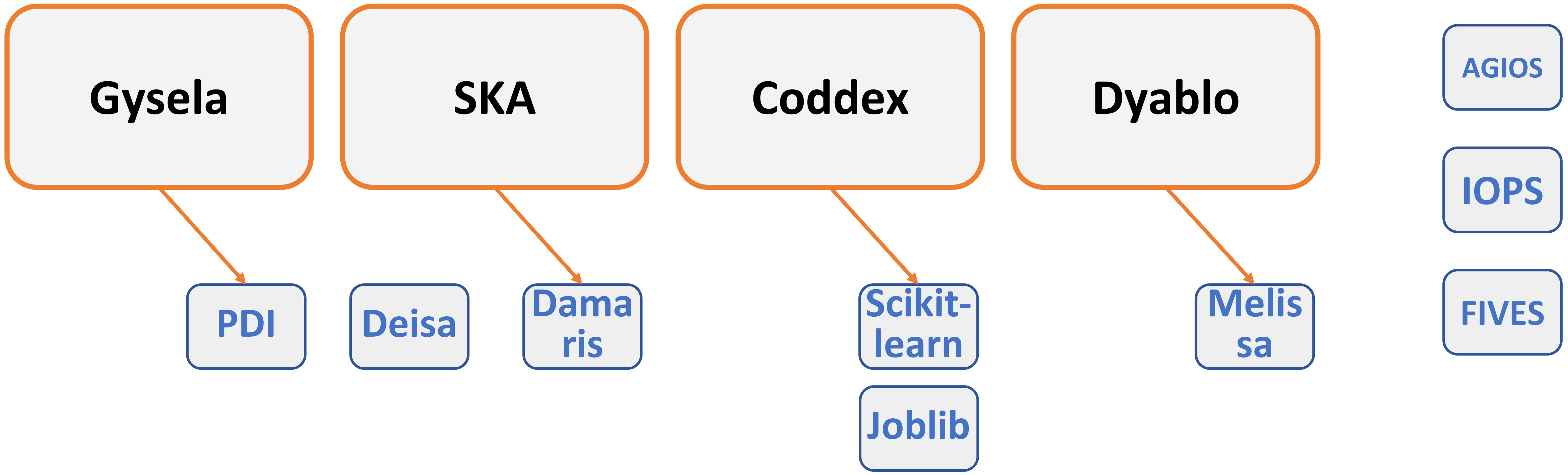
SKA

Coddex

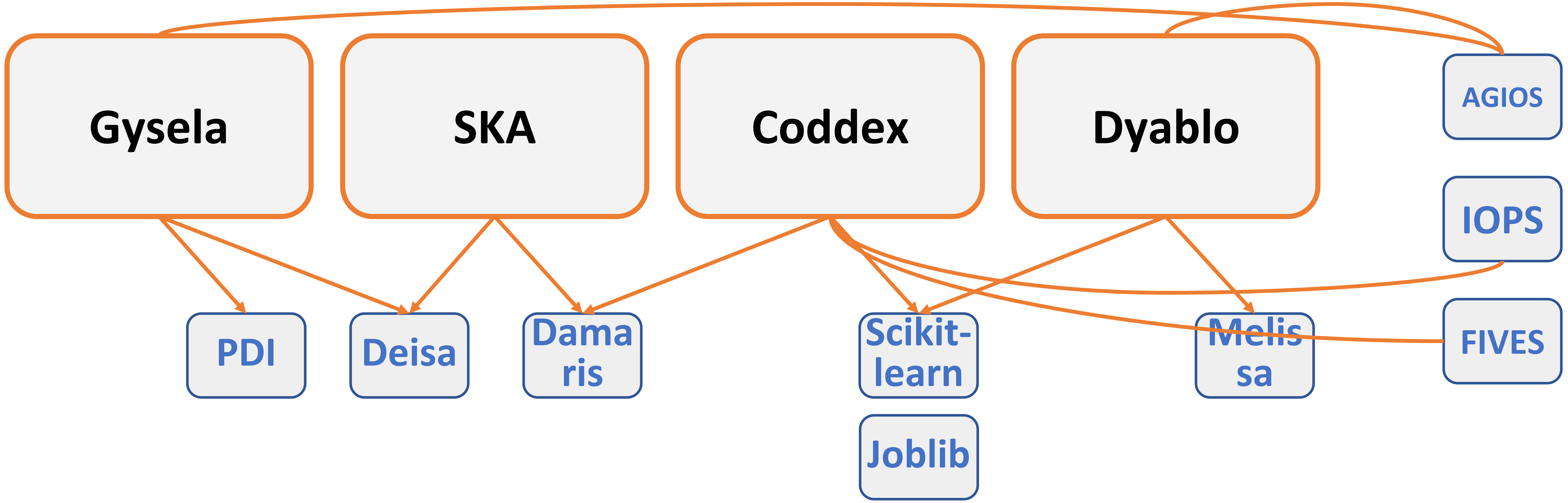
Dyablo


Gysela
IO
SKA
IO
Coddex
IO
Dyablo
IO


Gysela
SKA
Coddex
Dyablo
IO
IO
IO
IO



Application modularity



Tools interoperability

AGIOS

IOPS

PDI

Deisa

Dama
ris

Scikit-
learn

Joblib

Melis
sa

FIVES

PDI

Deisa

Damaris

Scikit-learn

Melissa

FIVES

IOPS

AGIOS

Joblib

PDI

Deisa

Damaris

Scikit-learn

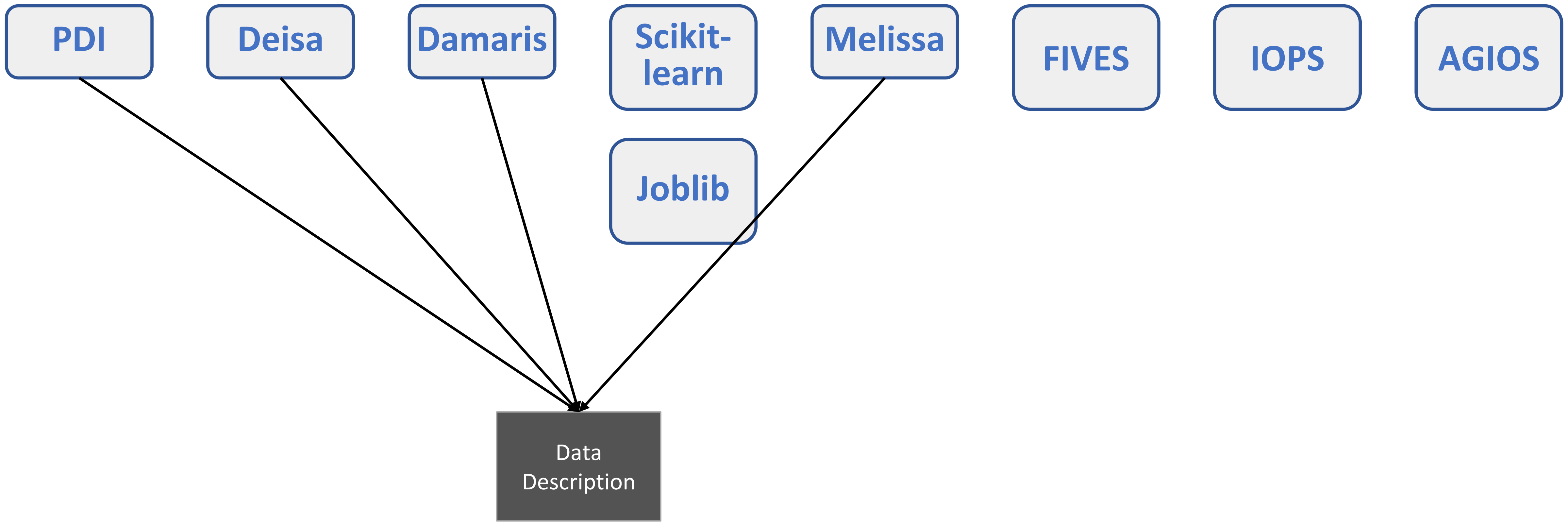
Melissa

FIVES

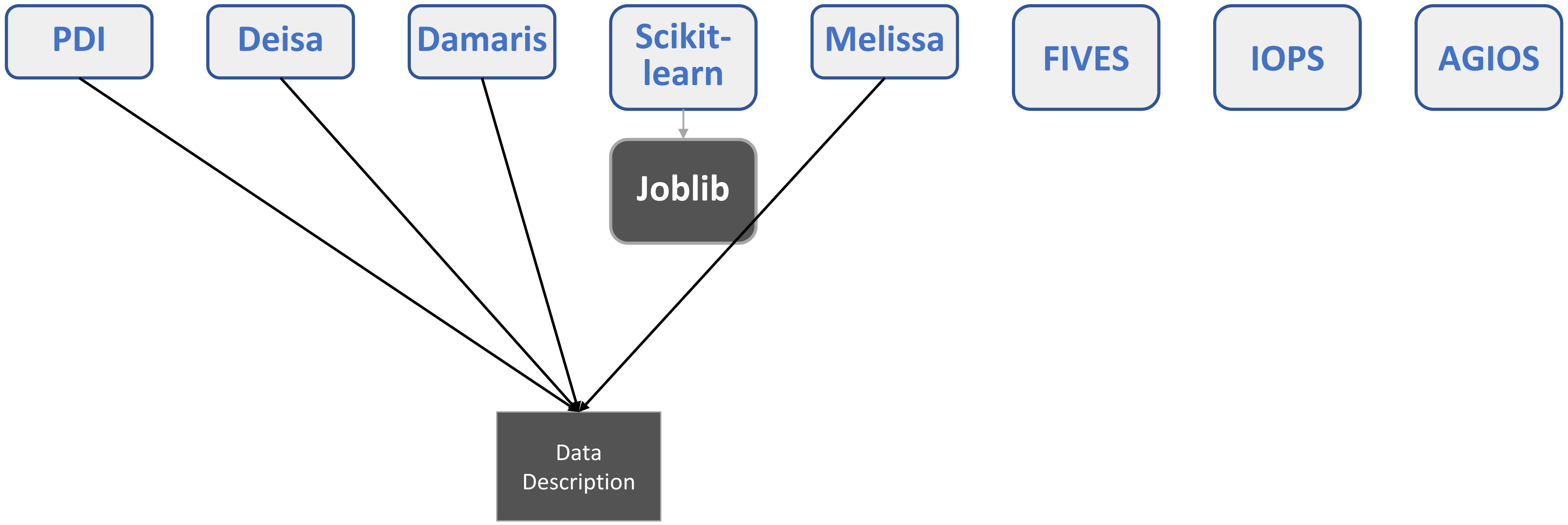
IOPS

AGIOS

Joblib



Tools modularity



Tools modularity

Hey, you, application developer, please stop redeveloping your data handling tools in you application, we develop great reusable tools in Exa-DoST

- *IO performance analysis & IO pattern characterization tools*
- *IO libraries & in-situ data analytics tools*
- *Data analytics libraries & tools*

You're not alone with similar problems, trust the experts, share & reuse, reduce your development costs!

Actually, what about us?
Does this also apply to us? To the development of our tools?

NumPEx
Exascale computing

Context:

Modularity: build an application or framework by assembling base components/services

Why modularity is important:

- Reduce development time and costs
- Reduce maintenance cost
- Get access to advanced features others have polished.

Why modularity is difficult:

- Delimiting services with the right API is difficult
- “Often” easier/faster (no need to survey and master others’ code) to code an app specific integrated service (but accumulate technical debt)
- Want to rely on reliable, well documented codes with long term support service

Modularity in NUMPEX: a way to collaborate and leverage each other’s work ?

- **Next January**, we have a **deliverable**
  - *Report on data-oriented software modularization and component mutualization between libraries*
- **Discussions started in 2023, still not much tangible actions**
  - not much feedback… no issue ? Ill posed ? Too hard ? Too distractive ?

Today's we need to make a decision

- Because next January, we also have a **Milestone**
  - *Identification of mutualization potential of data-oriented software components*
- 2 options for our answer:
  - **Yes**, we have identified potential for mutualization and rationalization in the development of our libraries, here they are, described in our deliverable
  - **No**, we looked carefully, there is no potential for sharing, see our deliverable that explains why, we surrender the resources that dedicated to this and will to reallocate them

What’s the decision

YES, WE CAN!

- We’ll now **work on it actively**
- We have identified 1 **representative** per tool/lib
  - Who is technical enough to identify modularity opportunities
  - Who is senior enough to commit to implementing this change
- But we need to **better understand each other**
  - We’ll setup bi-weekly visio meetings for this task force
  - To identify modularization opportunities

We delay the deliverable to April so as to get this task force to build it

We must list and identify the tools & libraries we develop & contribute to

- Which ones are initial prototypes?
- Which ones are more advanced and ready for development rationalization?
- What sub-components can we identify in each of these tools & libraries?
- Are there intersections of similar sub-components that would make sense to share?

Some tentative ideas (bring your owns):

- API for access to IO libraries (to easily switch from one to the other)
- Library to share dedicated cores in a simulation node for multiple purposes
- Metadata for parallel data structures (serialization/deserialization)
- Launching MPI from a pre-existing set of processes (without mpirun/srun)
- NxM collective communication lib (senders different from receivers)
- Virtual cluster in containers on single node for CI
- Python + C/C++ interoperability
- CPU/GPU memory transfers

When exchanging data between different entities (basically how to serialize/deserialize data and how to interpret it), the consumer needs to understand the structure of the received data. Most of the time this is implicit (the developer just knows what was sent and then decode accordingly at the reception side, but this is very error prone on the long run, especially when the data transmitted gains in complexity. Tools like PDI, Damaris, Adios, have their own metadata system (based on JSON,XML,Yaml), but this is bind to the tool (so note modular). One project, **Conduit** (https://llnl-conduit.readthedocs.io/en/latest/) proposes tof focus only on that aspect. Conduit associates to the data a Json schema and come with many utility fonctions to manipulate

the data and metadata, like serialization/deserialization functions. Conduit is multilangage. Abhishek started experimenting within Melissa. On-going conclusion is that it's easy to install, works, but the doc is not good and the c interface doc totally missing. Conduit has been adopted by the in situ vis. tool Catalyst2 combined with Adios2. We are also looking at alternative, but they are not many. Found this **Apache/fory** (https://fory.apache.org) project that serializes/deserialize directly objects and is multilangage, but no C support (not enough semantics in c) and C++ support is just starting. There is also the LowFive project (https://github.com/diatomic/LowFive) from ANL that takes **HDF5** as its interface and behind the hood seralize/deseralize the data into messages transported ported over MPI. Still not clear to us what is the right way to go. Any need for something similar ? Any experience ?

MPI works great for high performance networks (no need to elaborate on that), but MPI has one big limitation: you cannot start

it without a specific launcher (mpirun or srun). This is a significant limitation to its modularity. In particular it prevents to integrate MPI into runtimes that manage process creation their own way. It's not possible to have a preexisting group of processes (not started with mpirun or srun) and have them become part of a MPI communicator to perform collective communications for instance. This is a need that is discussed for Dask, Ray or JobLib. Take the case of Ray that can create groups of actors. They enable collective communications between these actors (mostly for neural network training) using either Gloo from Meta for inter CPU coms, NCCL from Nvidia for inter GPU coms, but not MPI. The only reason is the launching issue. In comparison NCCL or Gloo do not impose a specific launcher making them more flexible. But there is nonetheless progress on the MPI side with PMIx and MPI_Sessions that seems to make it doable. It's quite technical and related to the OpenMPI ecosystem (and I am far from mastering it fully). OpenMPI pushed in an external lib, called PMIx, everything that is related to process management. PMIx became a standard on its own. You probably use it without knowing when you use mpirun or srun. PMIx enables in particular to create "psets" of processes that identify themself as belonging to the same group. Then these processes can become part of a MPI communicator using the MPI_SESSION API rather than the classical MPI_INIT. As of today I never tried these mechanisms and I am not 100% sure they totally answer the problem mentioned above. So coming back to the modularity, the question is: are you concerned with this MPI limitations ? Is this a blocking issue for your runtime or app ? To stress that we are not indeed the only one asking the question here is a long interesting thread of discussion from the Dask community: https://github.com/dask/dask-mpi/issues/25

Some extra pointers:

- Discussion of the Dask community about having mpi into dask without using mpirun: https://github.com/dask/dask-mpi/issues/25
- Gloo Rendezvous: https://github.com/pytorch/gloo/blob/main/docs/rendezvous.md
- NCCL ncclGetUniqueId(): https://docs.nvidia.com/deeplearning/nccl/user-guide/docs/usage/communicators.html#creating-a-communicator
- MPI session model: https://www.mpi-forum.org/docs/mpi-4.1/mpi41-report/node271.htm
- PMIX and PRRTE in OpenMPI: https://docs.open-mpi.org/en/v5.0.x/launching-apps/pmix-and-prrte.html