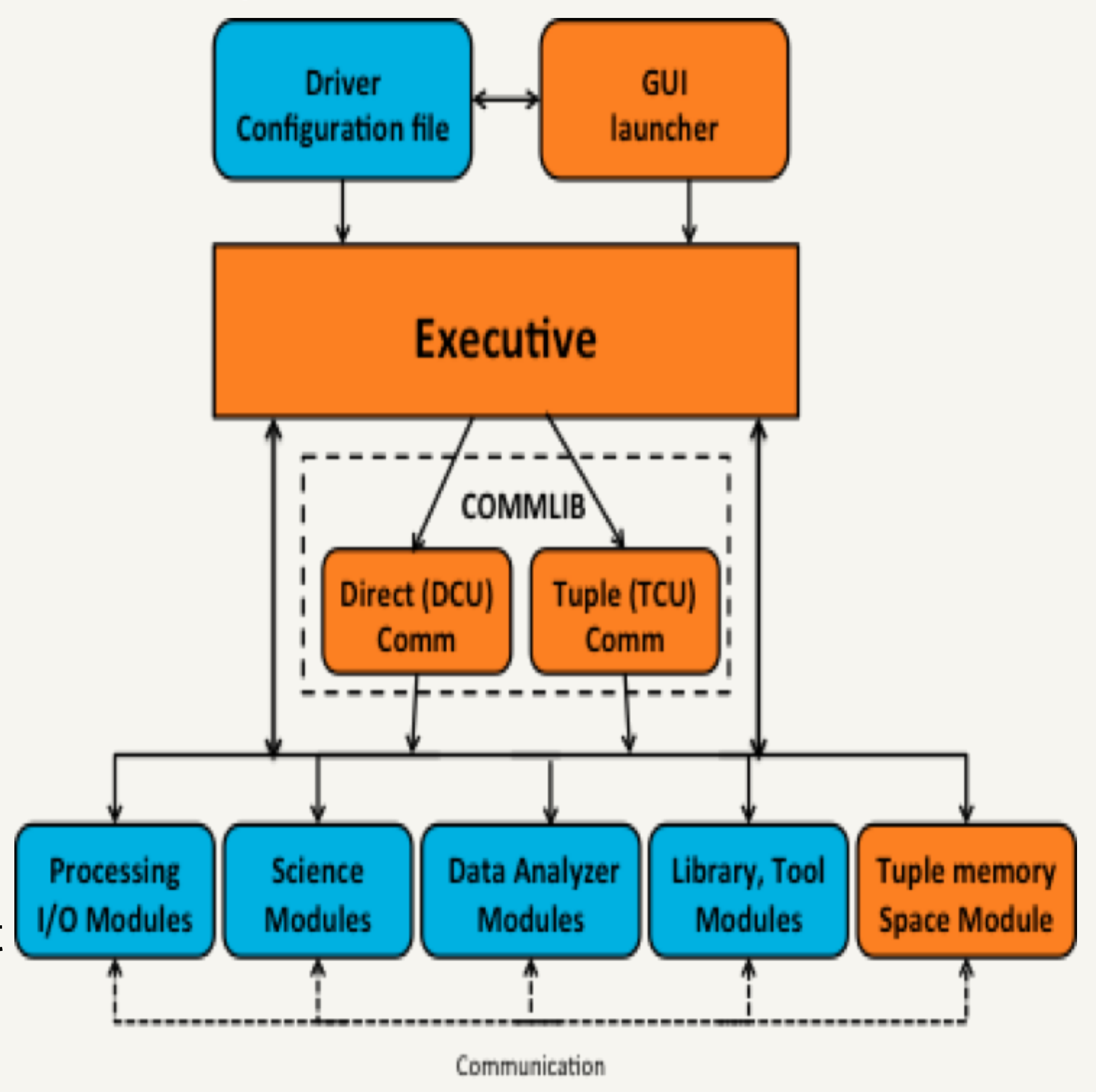
A Parallel Workflow Framework for Data and Compute Intensive Application openDIEL : cfdlab.utk.edu/opendiel

Rocco Febbo, Mentor : Kwai Wong

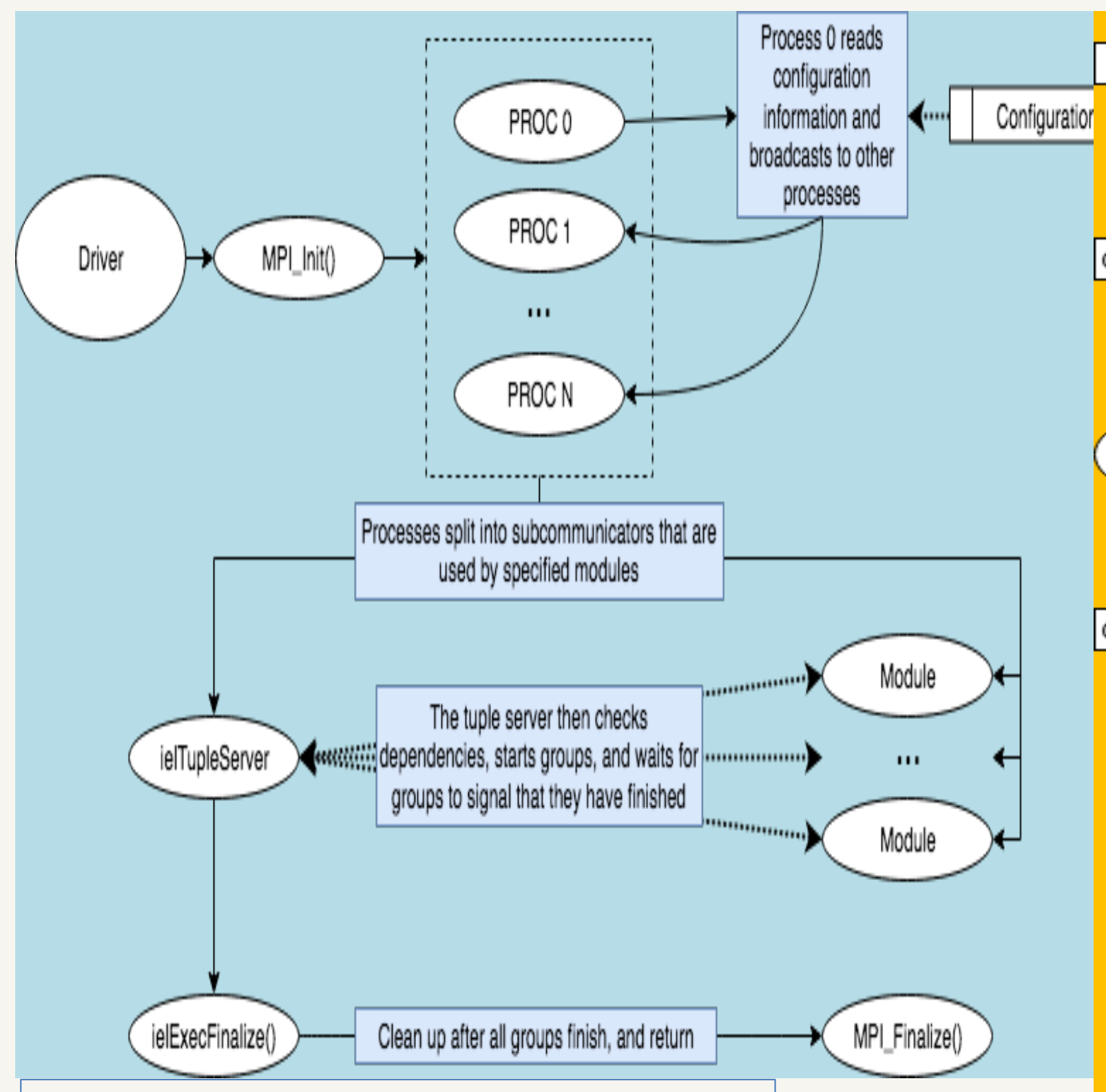
What is OpenDIEL?

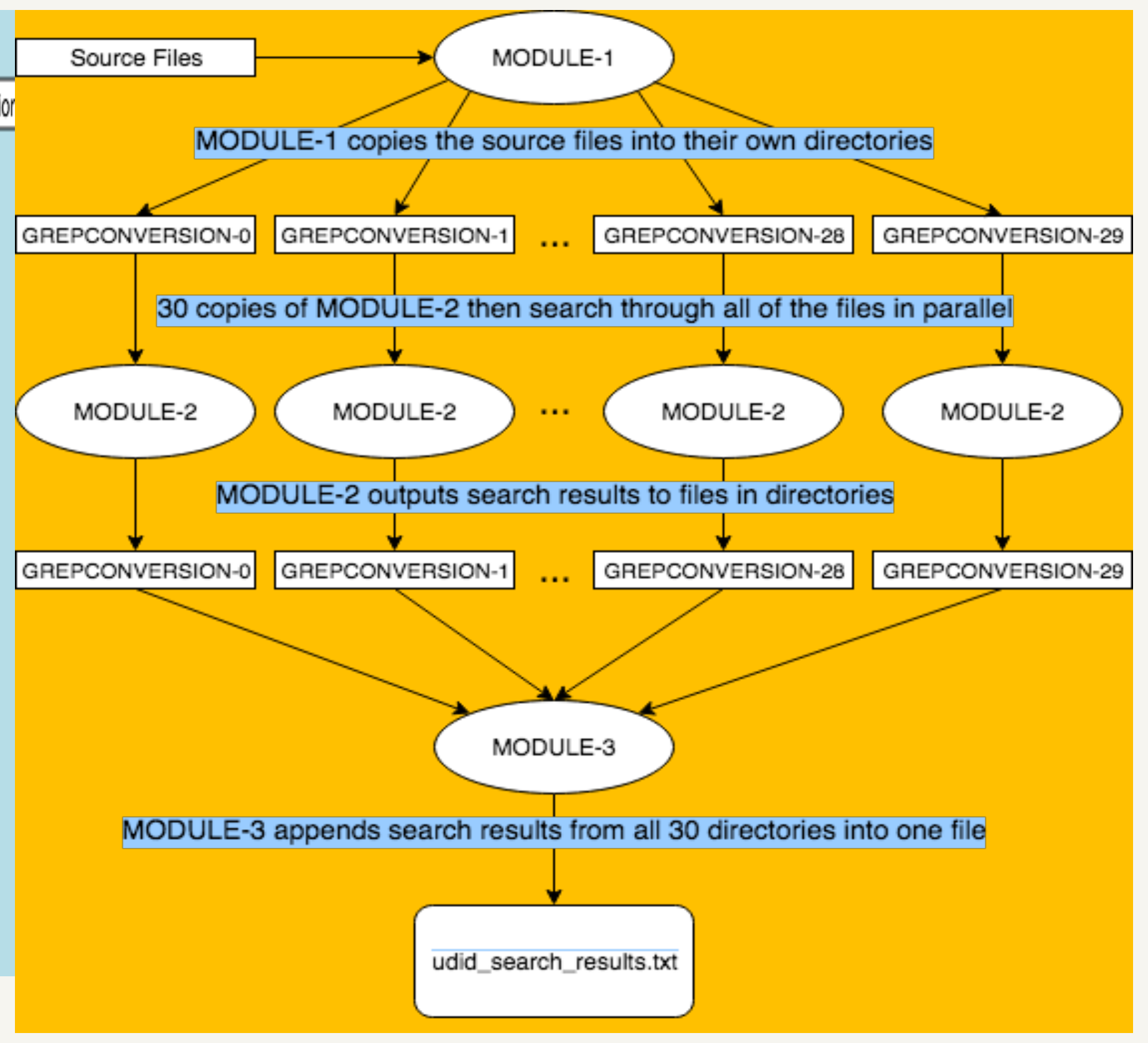
- The open Distributive Interoperable Executive Library (openDIEL) is a parallel workflow framework aims to run a collection of a user's codes (serial and parallel) collectively under a single MPI executable on HPC platforms.
- The user defines the function modules and schedules its workflow in an input file.
- Communication among modules can use the direct or tuple space interfaces
- Incorporate ML framework



Modules and Workflow for Applications

- A multicore single node program requires NO code changes, use normal executable to run : openmp, cuda, scripts, python, java, matlab, …..
- A MPI parallel program will be run as a function, a wrapper is available to convert a MPI programs to a function module.
- Modules attributes : automatic or managed mode, function name and input arguments, I/O directory path, GPU, thread, core, copy, size….
- Workflow arrangement : sets run in parallel, groups run in parallel with dependency, modules within a group run in sequential order


Driver
MPI_Init()
PROC 0
PROC 1
...
PROC N
Process 0 reads configuration information and broadcasts to other processes
Configuration
Processes split into subcommunicators that are used by specified modules
ielTupleServer
The tuple server then checks dependencies, starts groups, and waits for groups to signal that they have finished
Module
...
Module
ielExecFinalize()
Clean up after all groups finish, and return
MPI_Finalize()


Source Files
MODULE-1
MODULE-1 copies the source files into their own directories
GREPCONVERSION-0
GREPCONVERSION-1
...
GREPCONVERSION-28
GREPCONVERSION-29
30 copies of MODULE-2 then search through all of the files in parallel
MODULE-2
MODULE-2
...
MODULE-2
MODULE-2
MODULE-2 outputs search results to files in directories
GREPCONVERSION-0
GREPCONVERSION-1
...
GREPCONVERSION-28
GREPCONVERSION-29
MODULE-3
MODULE-3 appends search results from all 30 directories into one file
udid_search_results.txt

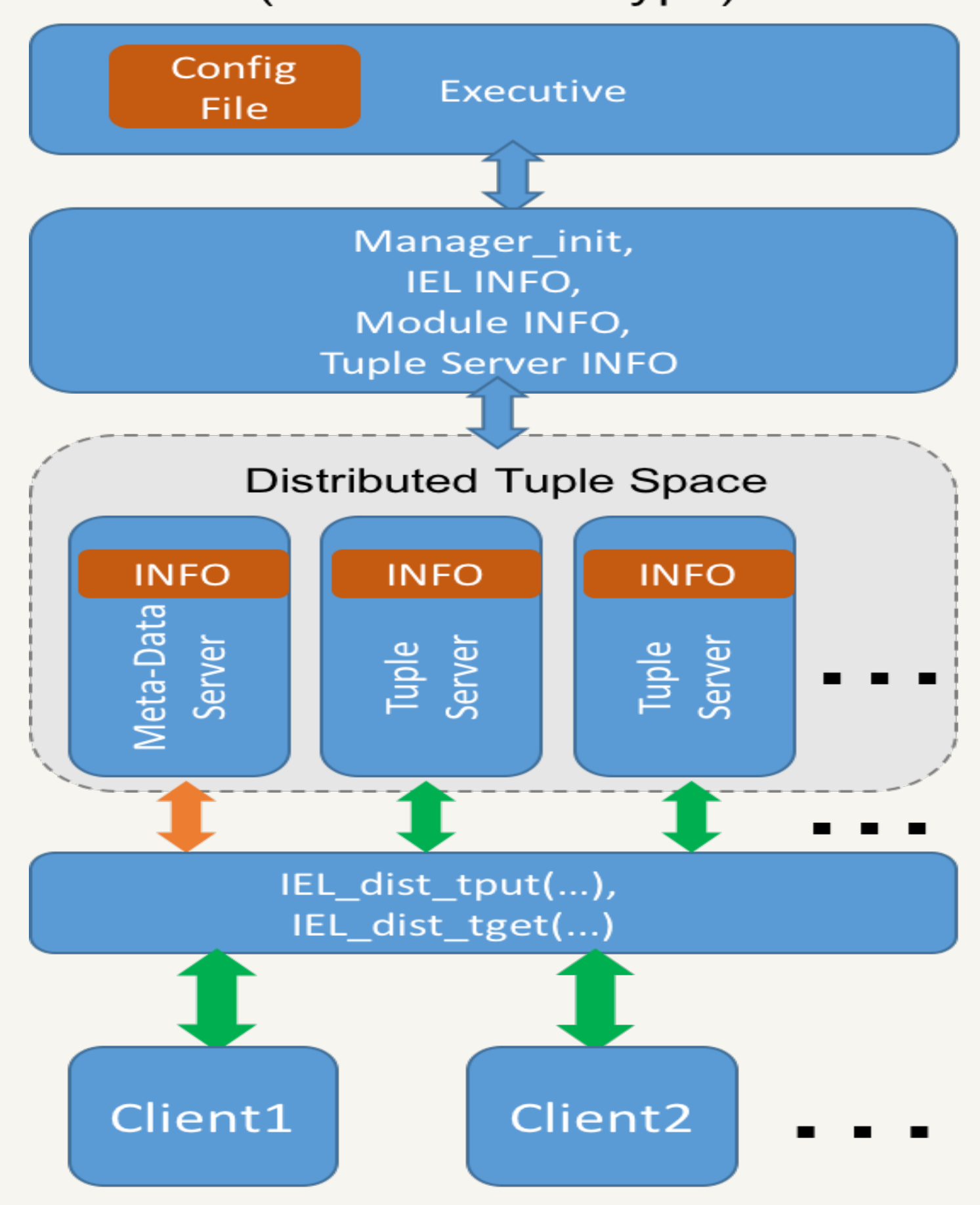
```
modules=(
{.  function="MODULE-1";
    args=("../hellomeexe");
    libtype="static";
    splitdir="HELLOME"
    size=2 },
{.  function="hello"
    args=()
    libtype="static"
    copies=2
    processes_per_copy=3
    size=6
    threads_per_process=4
    cores=24  },
)
```

```
set1:
  {
     num_set_runs=3
     group1:
     { order=("MODULE-1", "MODULE-2","MODULE5")
       iterations=2  },
     group2:
     {  order=("hello")
        iterations=2. },
     group3:
     {. order=("MODULE-4")
        iteration=1
        depends=("group1", "group2") }
  },
```

Distributed Tuple Space

- Modules may use a distributed array of tuple servers to store data in system memory that other modules may access.
- The sender places the data using IEL_dist_tput() and a user-defined data tag as an argument of the function.
- The receiver, using the same tag and the IEL_dist_tget() function will be able to retrieve the data from the distributed array.

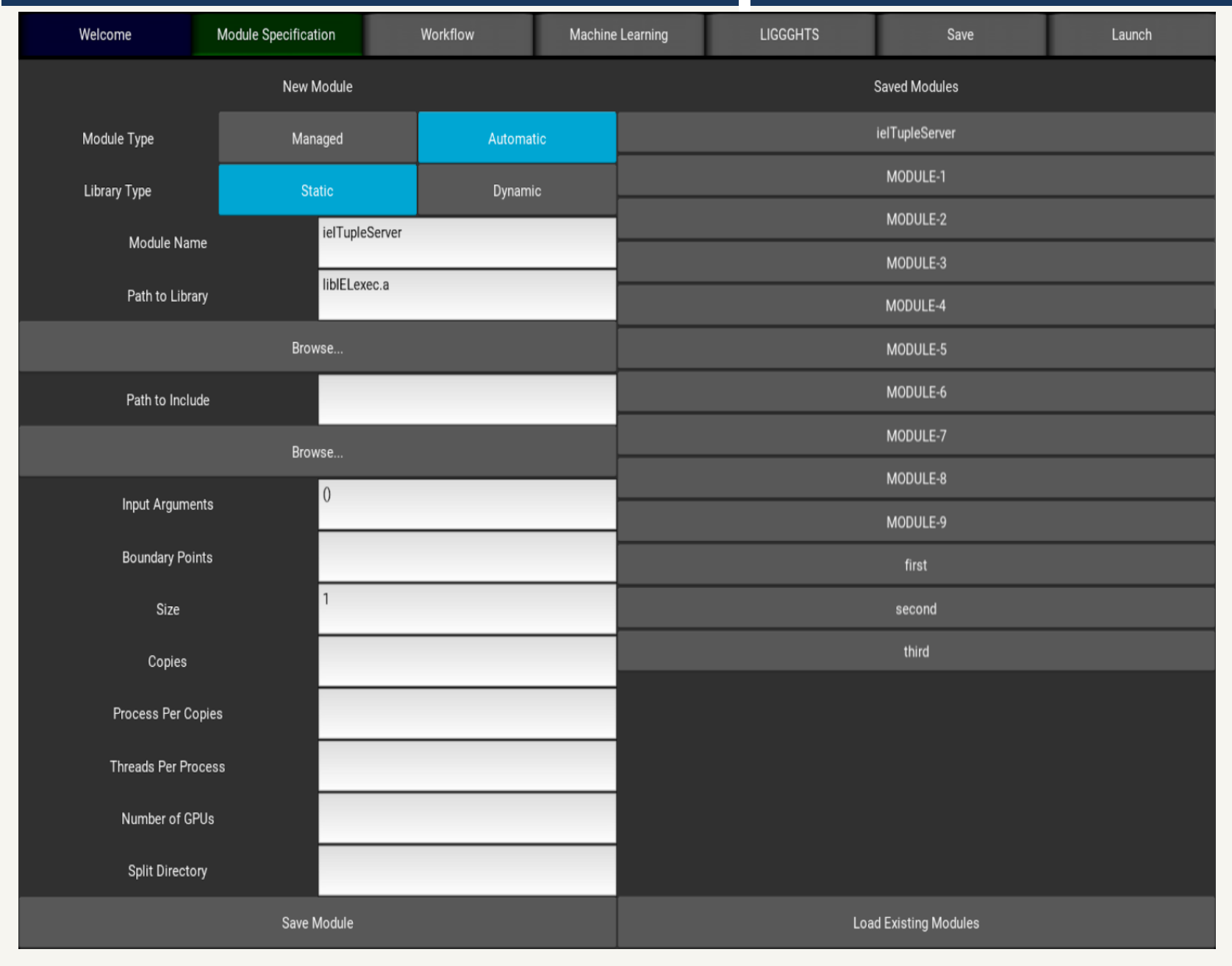
Distributed Tuple Comm (Current Prototype)

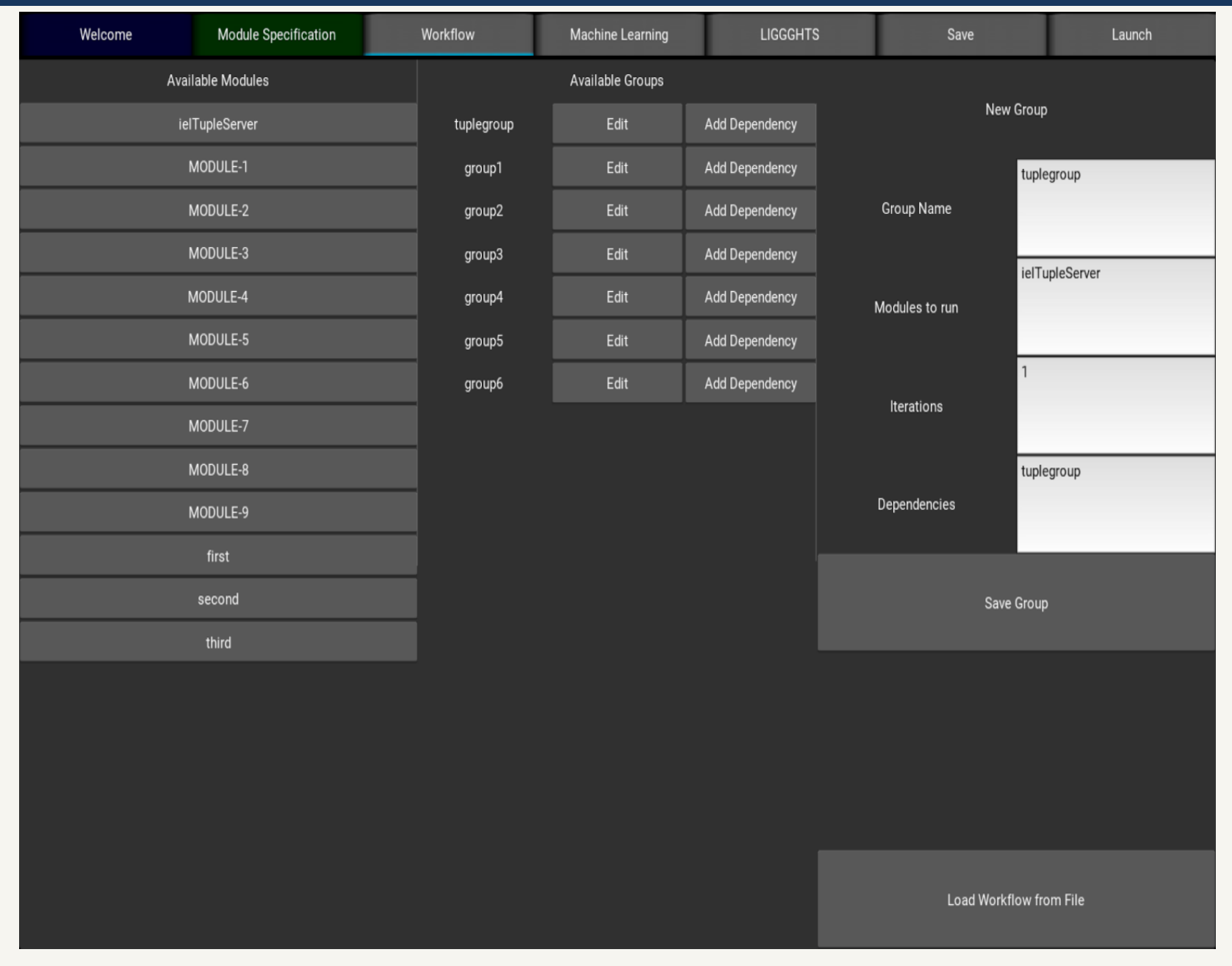


- **Sending data:** A client sends data to the distributed array of tuple servers by calling IEL_dist_tput():
- Distributes data among available tuple servers
- Stores the meta-data on the first tuple server

- **Receiving data:** A client receives data stored on the tuple servers by calling IEL_dist_tget()
- Queries the meta data server for the information using tag
- Uses the meta data to pull the data from the servers
- Reconstructs the data into an array that the client passed to the function

Graphical User Interface





The above pictures display the widgets which enable users to easily create modules or load existing modules to be ran with openDIEL. Once the user has either created or loaded their modules, they can proceed to create the workflow section for the modules. Then with the click of a button the configuration file that openDIEL uses will be created. The number of mpi processes will be calculated behind the scenes and the users example is ready to be launched.

Machine Learning – MagmaDNN

- A machine learning framework built around the Magma BLAS aimed at providing a modularized and efficient tool for training deep nets.
- MagmaDNN makes use of the highly optimized Magma BLAS giving significant speed boosts over other modern frameworks.

Acknowledgements

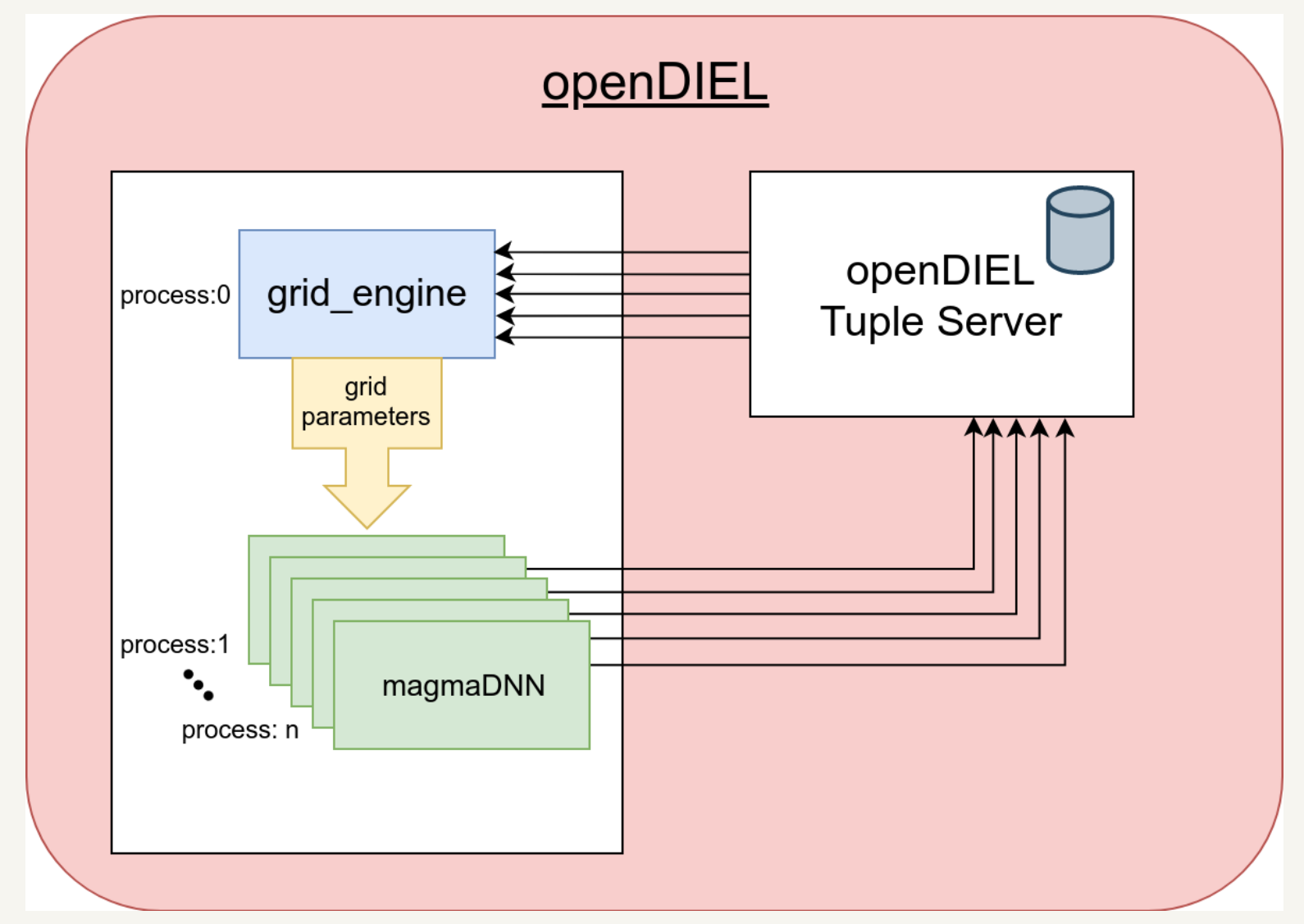
This project is made possible by funding provided by the NSF, computing resources provided by the University of Tennessee, and XSEDE. In addition, the computing work was also performed on technical workstations donated by the BP High Performance Computing Team.

Future Work

- Finish preparation of OpenDIEL for open source release
  - Extend testing suite
  - Create documentation and tutorials on use of OpenDIEL
- Add the ability to train across custom parameters in the grid engine
- Add new search methods such as PBT to the grid engine
- Add new trainee types such as TensorFlow to the grid engine

OpenDIEL Grid Engine

Since hyperparameter tuning is so computationally intensive it is desirable to have a distributed system which manages the process. Thankfully, the process is inherently parallelizable due to the small amount of data required to do a very large amount of work. OpenDIEL is well suited for this task due to its ability to handle intensive data and compute workloads

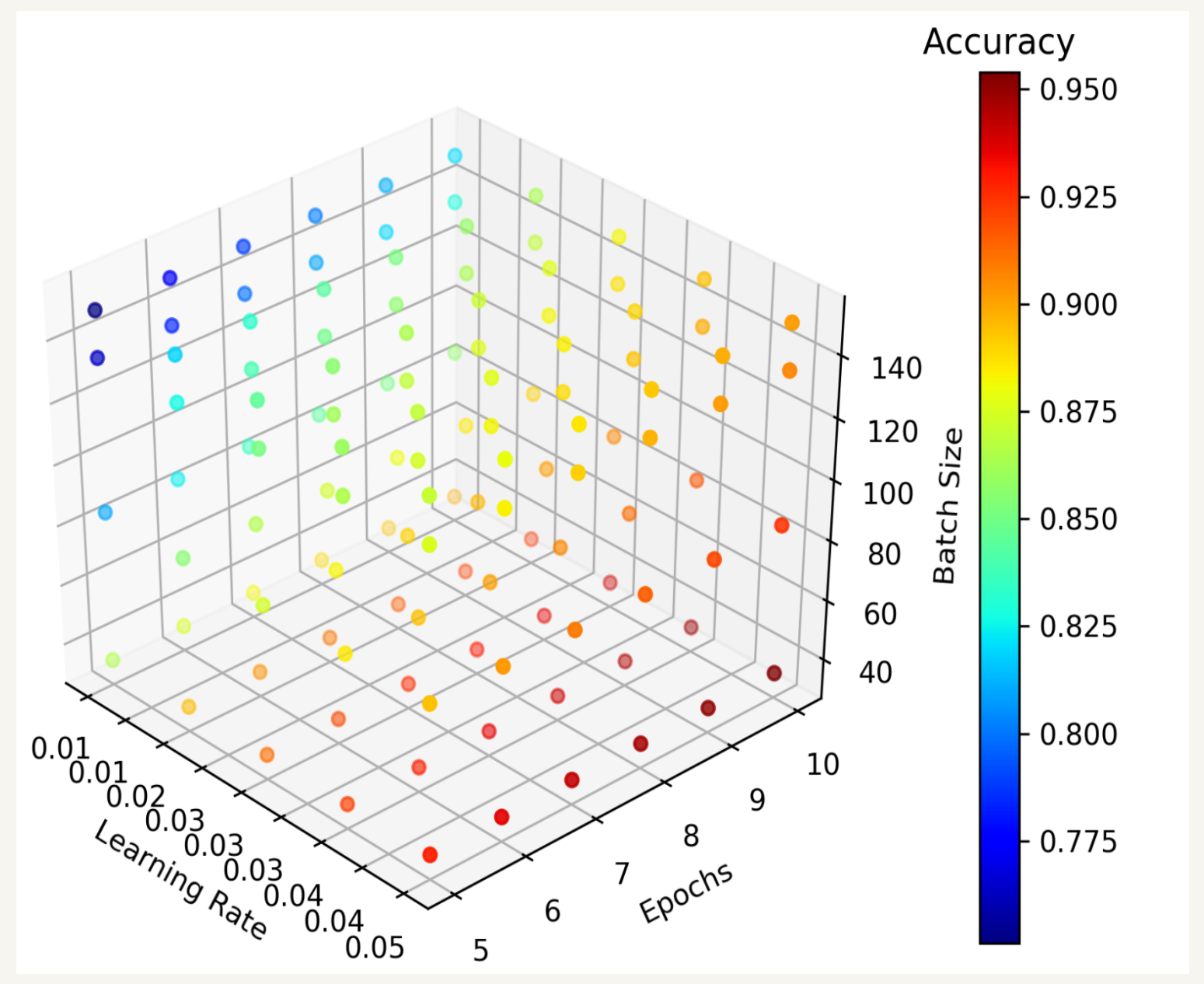


How Does the Tuple Server Work?

The Tuple Server is contained in it's own process. It acts like a storage container. Every piece of data is added to the Tuple Server along with a unique `tag` represented by an integer. The `tag` is how that data is then later accessed by other processes.

How Does the Grid Engine Work?

The Grid Engine manages a trainer and a set of trainees, each one its own process. The number of trainees depends on how big the OpenDIEL module size is and how many MPI process are allocated. The trainee sends hyperparameters to an OpenDIEL Tuple Server and the trainees receive that data, train, then report their accuracies to the Tuple Server. The trainer receives the accuracies and saves them to a file. It is designed to work with different search methods and different trainees. It has currently been tested using a grid search method and a MagmaDNN trainee. On the right is the result after training over a 3D grid space.



How Do You Interface With the Grid Engine?

To train across a grid you only need to provide a parameter configuration file as seen below. However, the OpenDIEL Grid Engine supports the ability to add different search methods and trainee types. It also supports training across different hyperparameters such as network structure. Currently implemented are the grid search method and a MagmaDNN trainee. Below is an example of how the trainer can communicate with the trainees.

An Example Configuration File for a Grid Search Across 1000 Parameters

```
parameters =
(
    # For learning rate, this ends up starting at 0.01, ends at
    # .05, and will increment in .01 steps. So, this will end up
    # trying the values 0.01, 0.02,...,0.05
    {
        name = "learning_rate";
    # can specify either continuous or discrete parameter
        type = "continuous";
    # start is the starting value in the search for this paramter
        start = 0.01;
    # end is the ending value in the search for this parameter
        end = 0.10;
    # step_size is the increment to change the paramter by in
    # each part of the search
        step_size = 0.01;
    },
    {
        name = "epochs";
        type = "discrete";
        start = 1.0;
        end = 10.0;
        step_size = 1.0;
    },
    {
        name = "batch_size";
        type = "discrete";
        start = 10.0;
        end = 100.0;
        step_size = 10.0;
    }
);
```

A Trainer Communicating with an arbitrary number of trainees

```
void grid_search_method::trainer_loop()
{
  vector< grid_layer > layers;
  vector< grid_param > parameters;
  /*
   * do some modification of the hyperparameters
   */
  for (size_t i = 0; i < n_trainees; i++) {
    send_hyperparameters(i, &parameters, &layers);
  }
  /*get the metrics */
  for (size_t i = 0; i < n_trainees; i++) {
    trainee_metric t;
    recv_metrics(i, &t);
  }
  /*send the trainees the done signal */
  for (size_t i = 0; i < n_trainees; i++) {
    send_hyperparameters(i, NULL, NULL);
  }
}
```

```
void grid_search_method::trainee_loop()
{
  t->recv_hyperparameters();
  t->train();
  t->send_metrics();
}
```