Finanziato dall'Unione europea NextGenerationEU
Ministero dell'Università e della Ricerca
Italiadomani PIANO NAZIONALE DI RIPRESA E RESILIENZA
terabit ICSC Centro Nazionale di Ricerca in HPC, Big Data and Quantum Computing

AccelerateRL: scaling beam dynamics simulations on INFN Cloud for Reinforcement Learning training

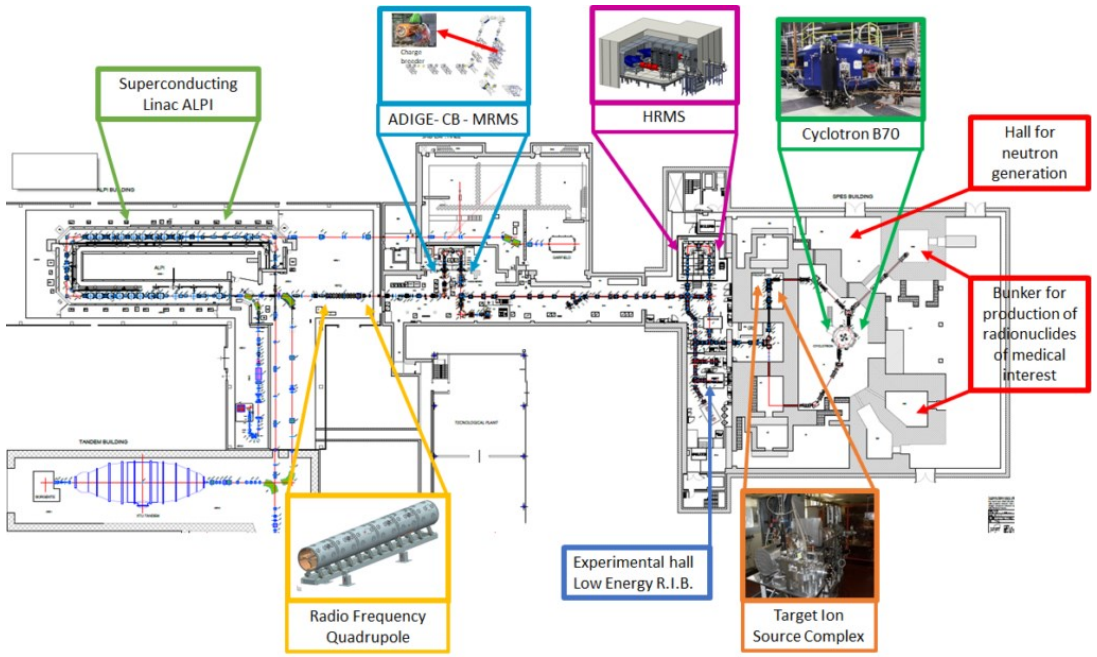
Davide Marcato, Daniel Lupu, Massimo Biasotto, Sergio Fantinel, Michele Gulmini
INFN-LNL

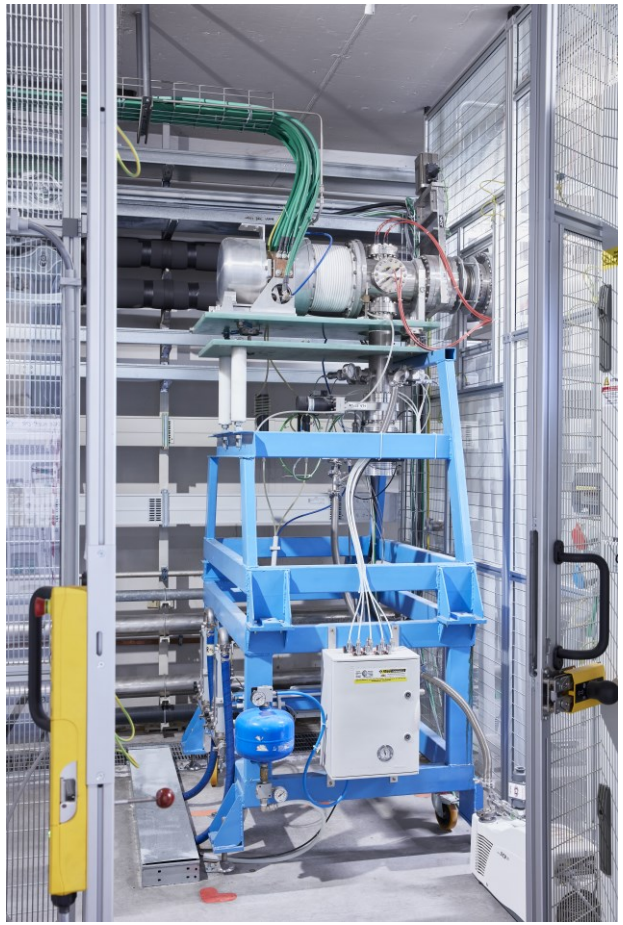
20 - 24 maggio 2024

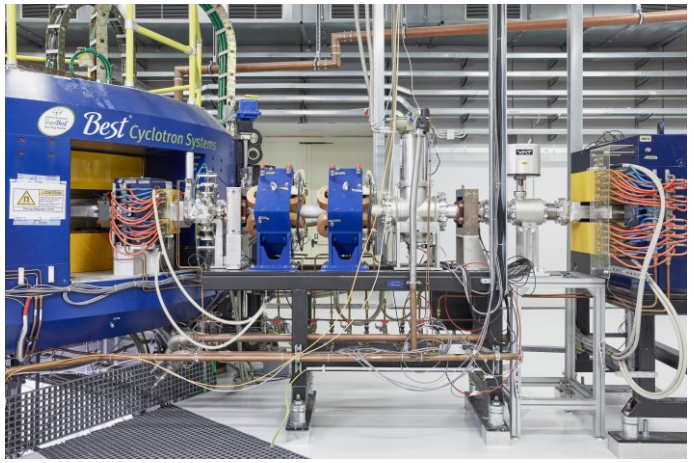
Workshop sul Calcolo nell'I.N.F.N.—Palau (Sassari)

INFN LNL
Istituto Nazionale di Fisica Nucleare
Laboratori Nazionali di Legnaro

SPES and the accelerator complex at Legnaro National Laboratories



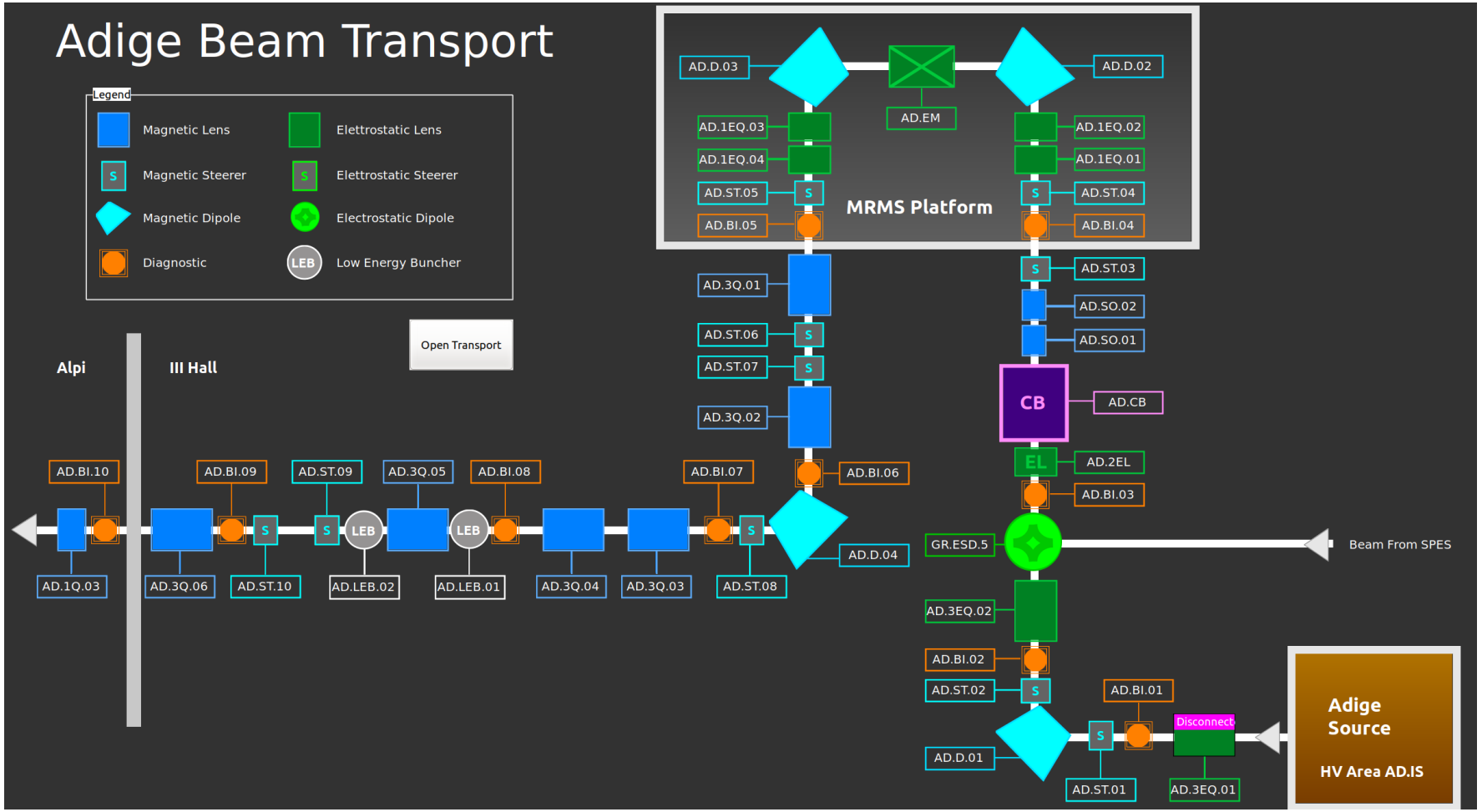




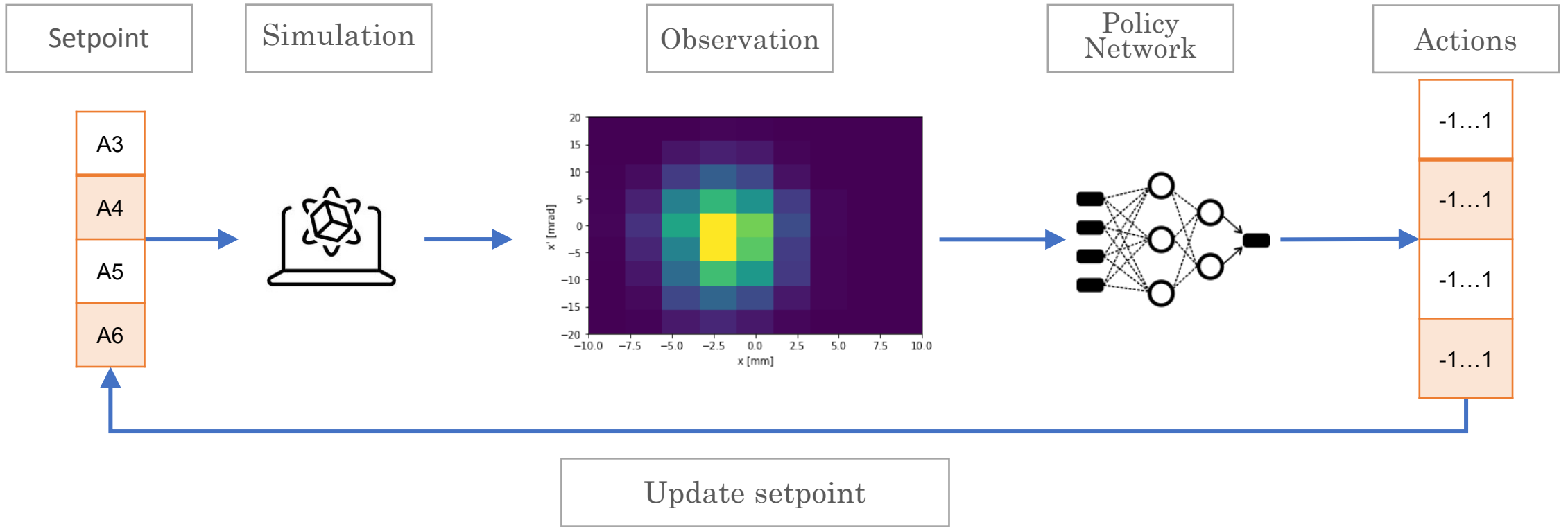
Beam Dynamics Optimization

Find the optimal setpoint of all the control system variables on a beam line

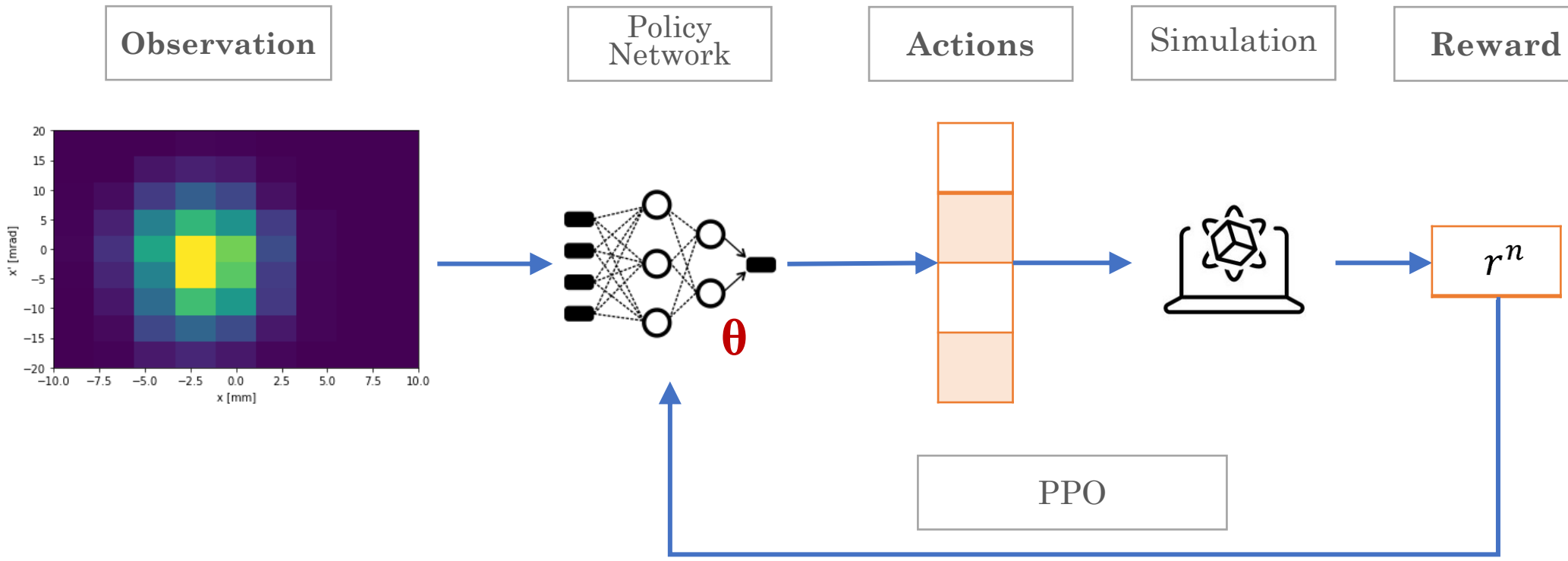
Use case: SPES@LNL



Reinforcement Learning for Beam Dynamics Optmization

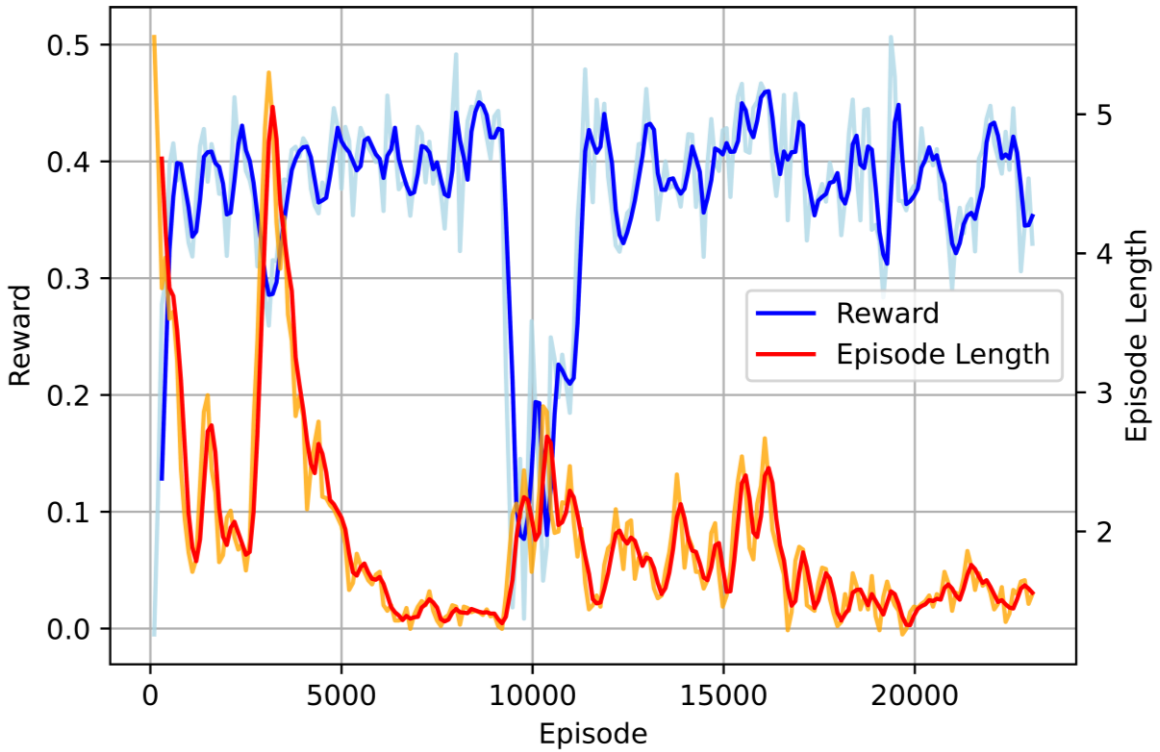


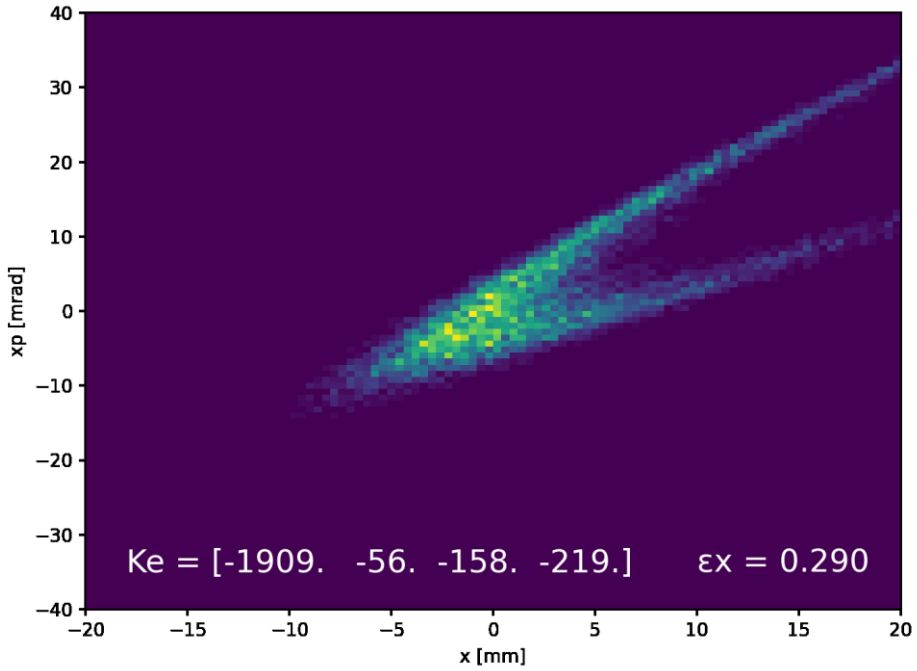
Reinforcement Learning Training



Results

D. Marcato et al., "Demonstration of Beam Emittance Optimization using Reinforcement Learning", in Proc. IPAC'23, Venice, Italy, May 2023, pp. 2881-2884. doi:10.18429/JACoW-IPAC2023-WEPA100





Success rate: 97% Mean number of steps per episode: 2.2

Similar Research Projects

RL4AA Collaboration

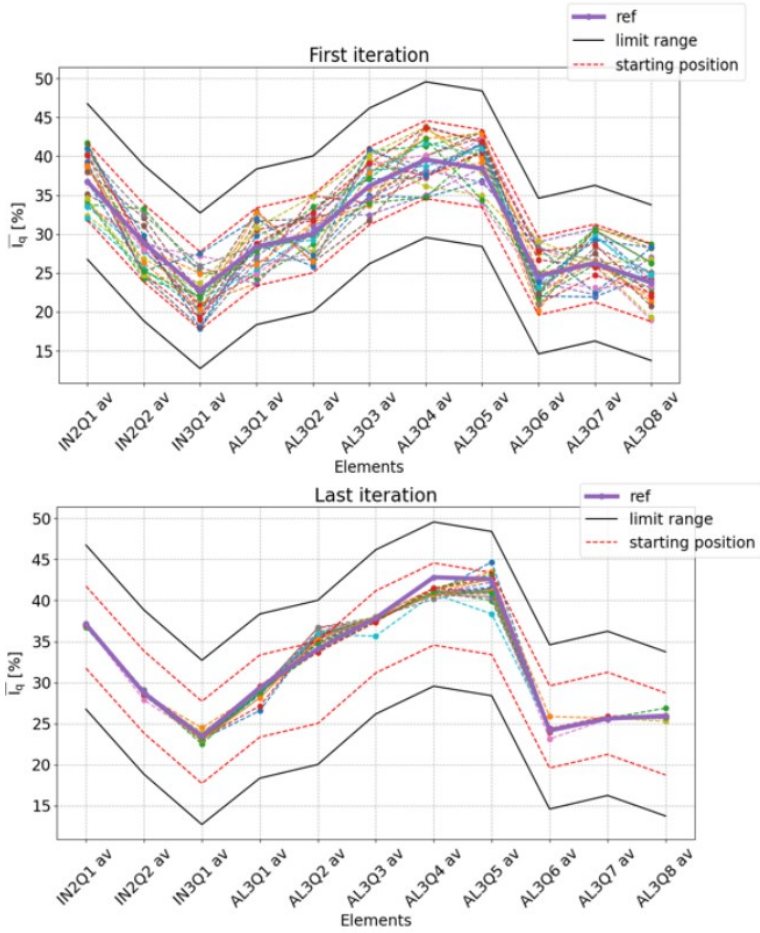
- Reinforcement Learning for Autonomous Accelerator Collaboration

Some example papers:

- *Learning to Do or Learning While Doing: Reinforcement Learning and Bayesian Optimisation for Online Continuous Tuning*
- *Ultra fast reinforcement learning demonstrated at CERN AWAKE*
- *Towards automatic setup of 18 MeV electron beamline using machine learning*
- *Orbit Correction Based on Improved Reinforcement Learning Algorithm*

Other approaches

- Swarm optimization algorithms
- Bayesian optimization
- Deep Learning surrogate models of the beam line



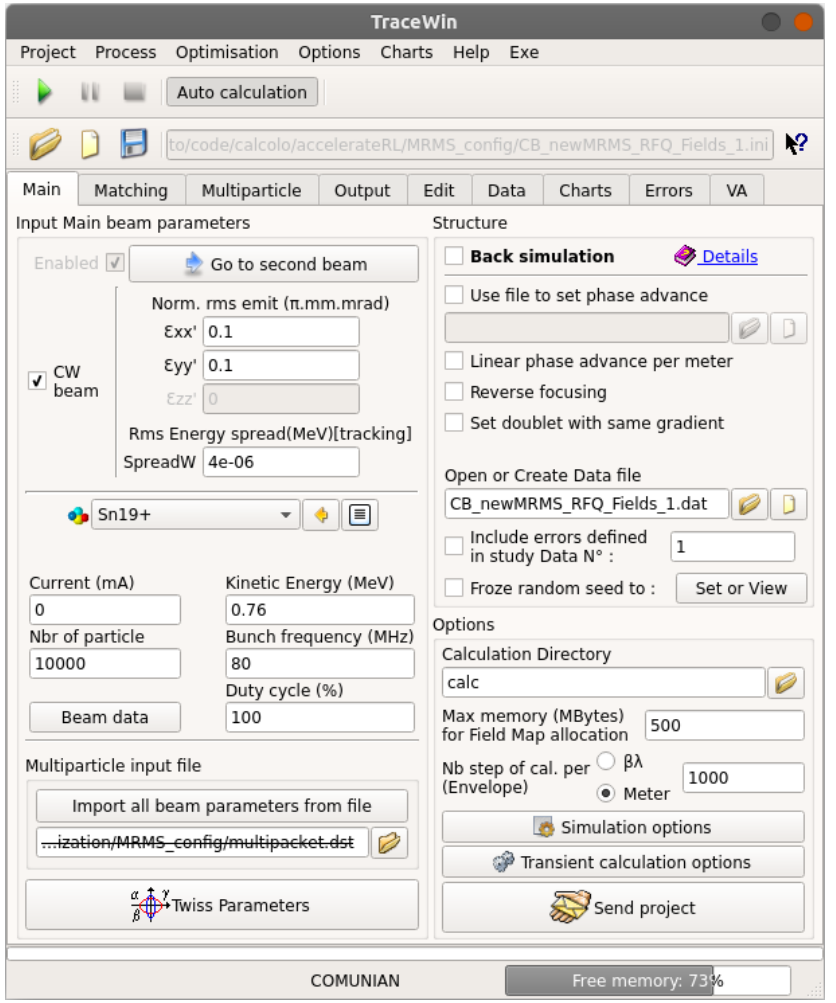
- Luca Bellan et al., New Techniques Method for Improving the Performance of the ALPI Linac, Published in: JACoW HB2023 (2024), FRA1C1 - DOI: 10.18429/JACoW-HB2023-FRA1C1
- Luca Bellan et. al., New techniques for the LNL superconductive linac ALPI beam dynamics simulations and commissioning, Published in: JACoW IPAC2023 (2023), TUODA3 - DOI: 10.18429/JACoW-IPAC2023-TUODA3

All these research projects require extensive use of beam dynamics simulations

TraceWin

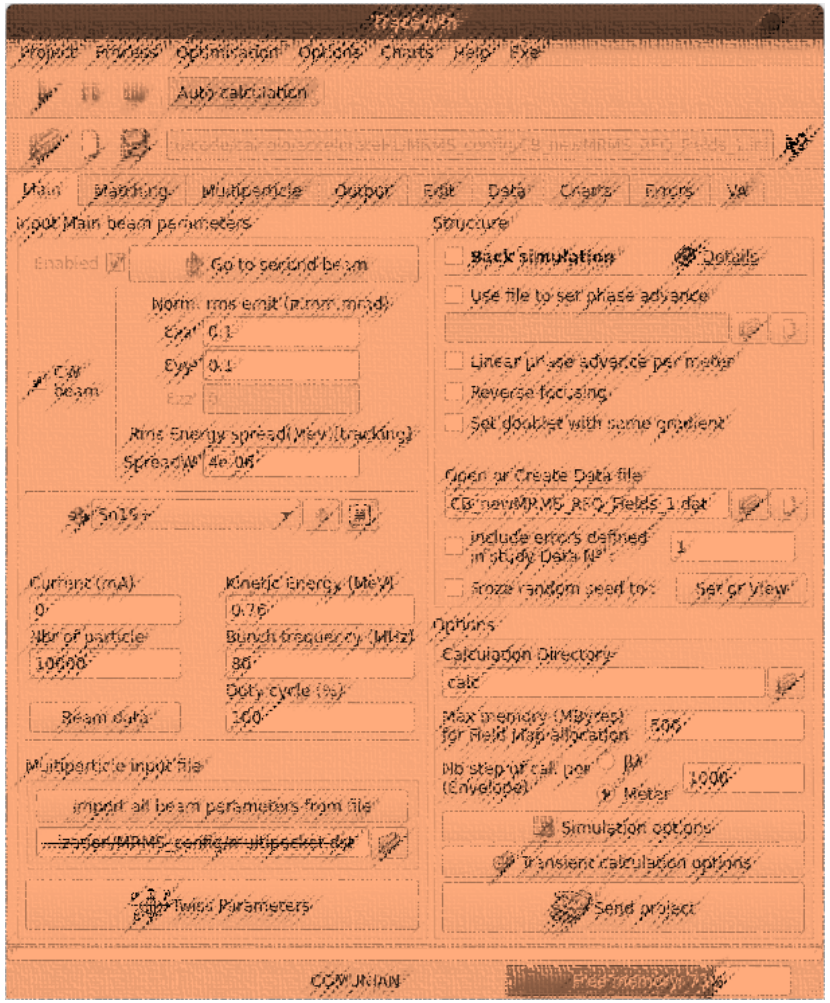
TraceWin Overview

- Simulator of beam dynamics in particle accelerators
- Developed by CEA in France
- Second order momentum or/and by a macro-particle distribution
- Field maps
- GUI-first program (QT-based)
- Windows – Linux – MacOS
- Feature-rich, great physics engine
  - Many graphs available out-of-the-box
  - Some optimization algorithms
  - Error study
- Multicore computing
  - CPU only



TraceWin shortcomings

- GUI-first
  - Command line interface available
  - Some properties only available on the GUI
- Not extensible
  - No API/library for any programming language
  - Input and outputs through files
- Licensed
- Closed source

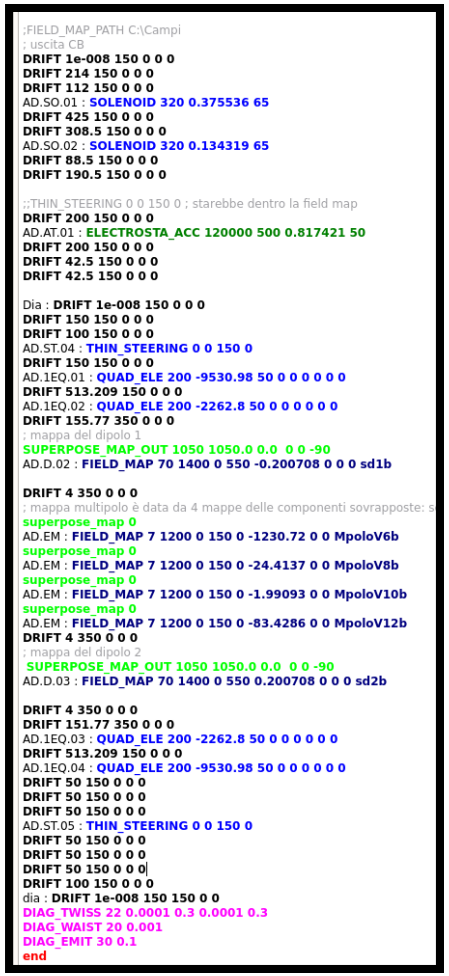


TraceWin Usage

- Configuration file (project_xyz.ini) contains all simulation parameters
  - Including the description of the beam line
  - Initially configured through GUI, then saved/loaded

- Three executables available (for linux)

Executable	GUI	Multicore	Paths
TraceWin	V	V	V
TraceWin_noX11	X	V	V
tracelx64	X	X	X



TraceWin Usage

- Run the simulation
  - Execution time depends on complexity and length of beam line

```
$ TraceWin project.ini path_cal=outdir hide nbr_part1=10000 …
```

- Read the output files from the output folder

Output file	Type	Description
partran1.out	csv	Main beam properties at each beam line element
part_dtl1.dst	binary	Distributions of beam properties at the end of the beam line
dtl1.plt	binary	Distributions of beam properties at each beam line element
Many more…		

Ele[n][v]	Change the v^{th} parameter of for the n^{th} element
hide	Hide the GUI, or cancel console output (no parameter)
tab_file	Save to file the data sheet at the end of calcul (only GUI version)
Synoptic_file	Save the geometric layout at (entance (=1), middle (=2), exit (=3) of elements. (See "Synoptic" tools for file name) .
nbr_thread	Set the max. number of core/thread used
path_cal	Calculation directory
dat_file	Full name of structure file
dst_file1	Full name Input dst of main beam (*)
dst_file2	Full name Input dst of second beam (*)
current1	Input beam current (mA) of main beam
current2	Input beam current (mA) of second beam
nbr_part1	Number of particle of main beam
nbr_part2	Number of particle of second beam
energy1	Input kinetic energy (MeV) of main beam
energy2	Input kinetic energy (MeV) of second beam
etnx1	Input XX' emittance (mm.mrad) of main beam
etnx2	Input XX' emittance (mm.mrad) of second beam
etny1	Input YY' emittance (mm.mrad) of main beam
etny2	Input YY' emittance (mm.mrad) of second beam
eln1	Input ZZ' emittance (mm.mrad) of main beam
eln2	Input ZZ' emittance (mm.mrad) of second beam
freq1	Input beam frequency (MHz) of main beam
freq2	Input beam frequency (MHz) of second beam
duty1	Duty cycle of main beam
duty2	Duty cycle of second beam
mass1	Input beam mass (eV) of main beam
mass2	Input beam mass (eV) of second beam
charge1	Input particle charge state of main beam
charge2	Input particle charge state of second beam

TraceWin manual - CEA/SACLAY - DRF/Irfu/DACM – Didier URIOT

pytracewin

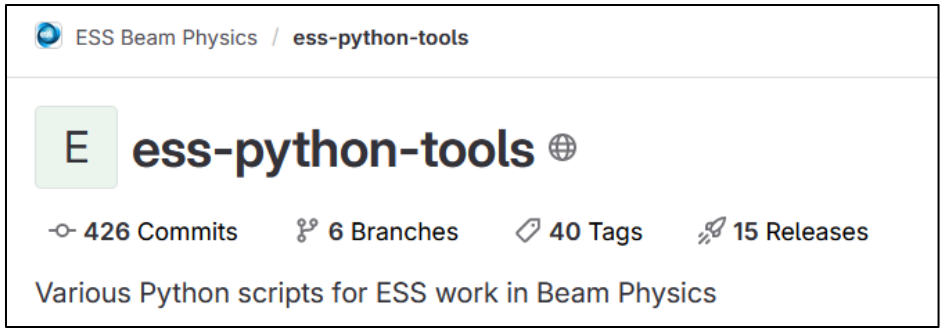
Goal

- Python wrapper for TraceWin executables
- Easy to use
- Run simulations with parameters
- Parsing of the output files
- Combine physics simulations with algorithms or ML libraries
- Do not require GUI by default
    - To be run on remote servers
- Easy to install via pip
    - Do not include proprietary TraceWin executables

Implementation

- **TraceWin** class to interact with the simulator
- Using the **subprocess** package to run the shell command
  - Handle runtime parameters
  - Raise exception when simulation fails
  - Impose timeout to avoid blocking calls
- Using **ess-python-tools** package to decode binary output files
  - From European Spallation Source (ESS)
  - dst, plt
- Using **pandas** to parse the csv result file
- Hide mode
- Debug mode

pandas



Usage

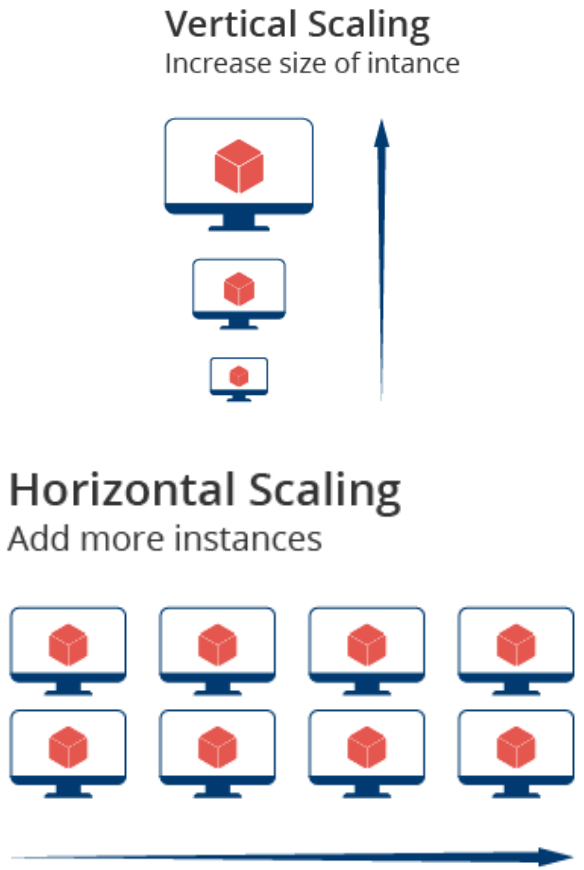
```python
from pytracewin import TraceWin

tracewin = TraceWin("MRMS_config/CB_newMRMS_RFQ_Fields_1.ini",
                    executable="TraceWin",
                    hide=True, debug=False)

tracewin.run(params={"nbr_part1": 10000, "ele[27][6]": -1200},
             timeout=30)

tracewin.dst()
tracewin.plt()
tracewin.results()
```

✓ 11.6s Python

	##	z(m)	gama-1	x0	y0	p0	x'0	y'0	W0	SizeX	...
0	0	-0.000000	0.000006	1.245505e-08	4.161037e-08	-0.017999	5.401000e-09	1.686846e-07	-5.822923e-08	4.475022	...
1	1	0.000000	0.000006	1.245505e-08	4.161037e-08	-0.017999	5.401000e-09	1.686846e-07	-5.822923e-08	4.475022	...
2	2	0.214000	0.000006	1.361086e-08	7.770887e-08	2.677506	5.401000e-09	1.686846e-07	-5.822923e-08	8.968381	...

Scaling to the Cloud

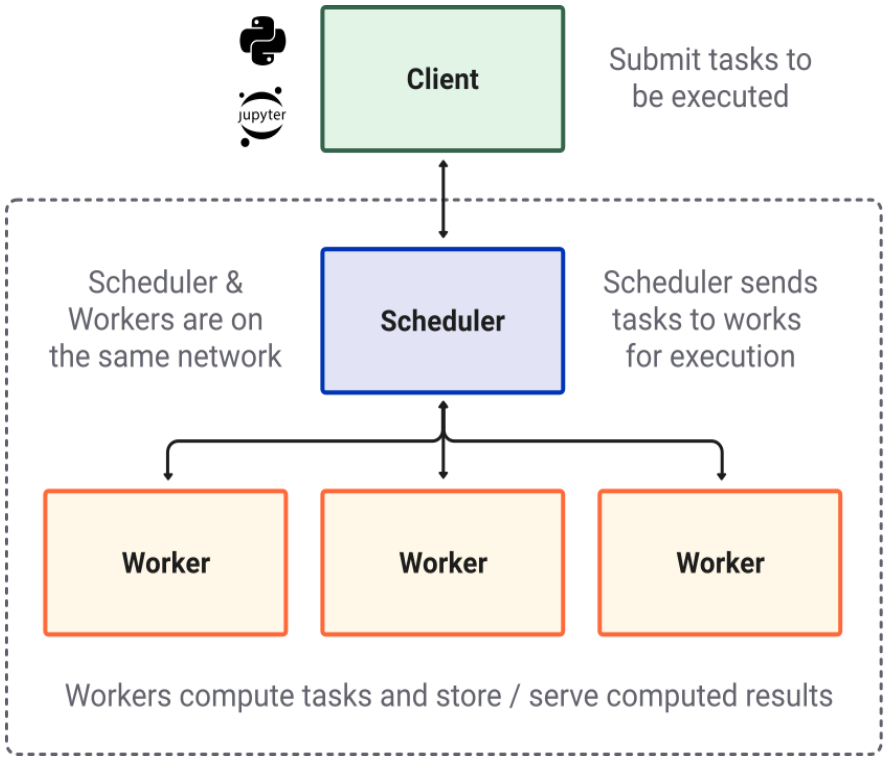
- pytracewin lets you run simulations from python
- This is useful to test optimization algorithms
- The limiting factor becomes the computing power of your local machine
- You can run them on *fat* servers
    - But this doesn't scale much
- Often the simulations can be run independently between one another
    - Embarrassingly parallel workload
- Scale to the cloud
    - You need a coordination mechanism
    - Send jobs and retrieve results
    - Simple user interface
    - Backend-agnostic



Dask

Dask Cluster

- Dask is an open-source Python library for parallel computing
- Can be deployed anywhere:
  - Local, ssh
  - Cloud (AWS, GCP, Azure)
  - SLURM
  - Kubernetes
- Dask is a task scheduler
- Enables embarrassingly parallel computing
- Works on native python data types
- Graph task lazily evaluated
- Diagnostic dashboard



Dask Cluster

Dask Cluster with SSH

SSHCluster: solution for manually managed machines on a local network

- Deploys a Dask Scheduler and Workers on the set of provided machines

We tested it on a couple of local VMs

- Validate the functionality of Dask+pytracewin

Does not scale easily

- All the VMs must be pre-configured with the environment for TraceWin
- Hard to deploy and maintain

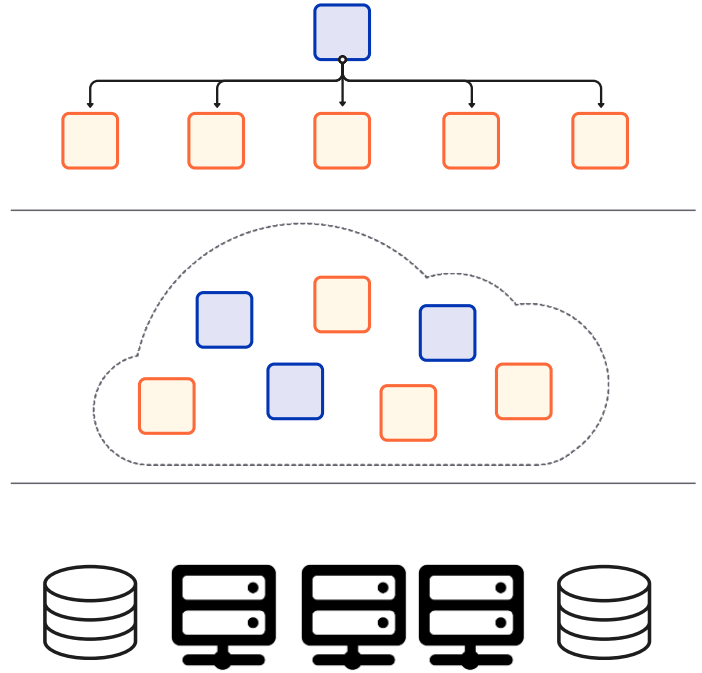
Dask Cluster on Kubernetes

- Automatic setup
  - Service described by YAML files
- **Scale** to a large number of worker nodes
  - Dynamic cluster
- Exploit the available service on INFN-Cloud
- Common interface
  - Do not depend on the VM provider
  - **Portable**
- Docker image with TraceWin environment

dask

kubernetes

openstack.



Custom Docker image

- Derived from *dask:latest*
  - Includes python, dask, conda
- Adding **USER**
  - Required by the software license
- Adding headless **tracelx64** executable
- Installing **pytracewin**
- 350MB

```
1   FROM ghcr.io/dask/dask:latest
2
3   ARG USER
4
5   RUN groupadd -g 1000 ${USER} && \
6       useradd -ms /bin/bash -g ${USER} -u 1000 ${USER}
7
8   COPY TraceWin/tracelx64 /usr/local/bin/tracelx64
9
10  USER ${USER}
11  WORKDIR /home/${USER}
12
13  COPY --chown=${USER}:${USER} pytracewin accelerateRL/pytracewin
14  COPY --chown=${USER}:${USER} pyproject.toml accelerateRL/pyproject.toml
15
16  RUN pip install --no-cache-dir --upgrade pip && \
17      pip install --no-cache-dir ./accelerateRL
```

```
$ docker build -t baltig.infn.it:4567/marcato/accelerated-rl/dask_tracewin:1.1.0
--build-arg USER=marcato
```

```
$ docker push baltig.infn.it:4567/marcato/accelerated-rl/dask_tracewin:1.1.0
```

Dask Operator

A set of custom resources and a controller that runs on your Kubernetes cluster and allows you to create and manage your Dask clusters as Kubernetes resources

```
$ helm install --repo https://helm.dask.org --create-namespace -n dask-operator
--generate-name dask-kubernetes-operator
```

Dask Cluster definition

dask_cluster.yaml

- Number of workers
- Custom image to be used
  - Credentials to download the custom image
- CPU/memory requests and limits
  - Horizontal scaling
- nthreads=1
  - Force a single TraceWin execution per container
  - Avoid conflicts on I/O files

```yaml
# dask_cluster.yaml
apiVersion: kubernetes.dask.org/v1
kind: DaskCluster
metadata:
  name: accelerate-rl
spec:
  worker:
    replicas: 40
    spec:
      containers:
      - name: worker
        image: "baltig.infn.it:4567/marcato/accelerated-rl/dask_tracewin
        imagePullPolicy: "IfNotPresent"
        args:
          - dask-worker
          - --name
          - $(DASK_WORKER_NAME)
          - --nthreads
          - "1"
          - --dashboard
          - --dashboard-address
          - "8788"
        ports:
          - name: http-dashboard
            containerPort: 8788
            protocol: TCP
        resources:
          limits:
            cpu: "1"
            memory: "2Gi"
          requests:
            cpu: "0.8"
            memory: "1Gi"
      imagePullSecrets:
          - name: regcred
```

Dask Cluster definition

dask_cluster.yaml

- Dask Scheduler
- Same docker image as the workers
- Scheduler and Dashboard port 8786, 8787
- *Fat* scheduler pod
- Service NodePort for connectivity to the scheduler

```
$ kubectl apply -f dask_cluster.yaml
$ kubectl port-forward svc/accelerate-rl-scheduler 8786:8786
$ kubectl port-forward svc/accelerate-rl-scheduler 8787:8787
```

```
6   spec:
36    scheduler:
37      spec:
38        containers:
39        - name: scheduler
40          image: "baltig.infn.it:4567/marcato/accelerated-rl/dask_tracewin
41          imagePullPolicy: "IfNotPresent"
42          args:
43            - dask-scheduler
44          ports:
45            - name: tcp-comm
46              containerPort: 8786
47              protocol: TCP
48            - name: http-dashboard
49              containerPort: 8787
50              protocol: TCP
51          readinessProbe:
52            httpGet:
53              port: http-dashboard
54              path: /health
55            initialDelaySeconds: 5
56            periodSeconds: 10
57          livenessProbe:
58            httpGet:
59              port: http-dashboard
60              path: /health
61            initialDelaySeconds: 15
62            periodSeconds: 20
63          resources:
64            limits:
65              cpu: "8"
66              memory: "16Gi"
67            requests:
68              cpu: "7.5"
69              memory: "14Gi"
70        imagePullSecrets:
71          - name: regcred
72      service:
73        type: NodePort
74        selector:
75          dask.org/cluster-name: accelerate-rl
76          dask.org/component: scheduler
77        ports:
78        - name: tcp-comm
79          protocol: TCP
80          port: 8786
81          targetPort: "tcp-comm"
82        - name: http-dashboard
83          protocol: TCP
84          port: 8787
85          targetPort: "http-dashboard"
```

Basic Usage

```
client = Client("tcp://localhost:8786")

# Upload project file and license key
client.register_plugin(UploadDirectory("MRMS_config"))

def simulate(*args):
    tracewin = TraceWin("/tmp/dask-scratch-space/MRMS_config/CB_newMRMS_RFQ_Fields_1.ini")
    completed_proc = tracewin.run(*args)
    results = tracewin.results()
    return [completed_proc, results]
```

[ ]

```
# Run the simulation
client.submit(simulate, {"ele[26][6]": -1400}).result()
```

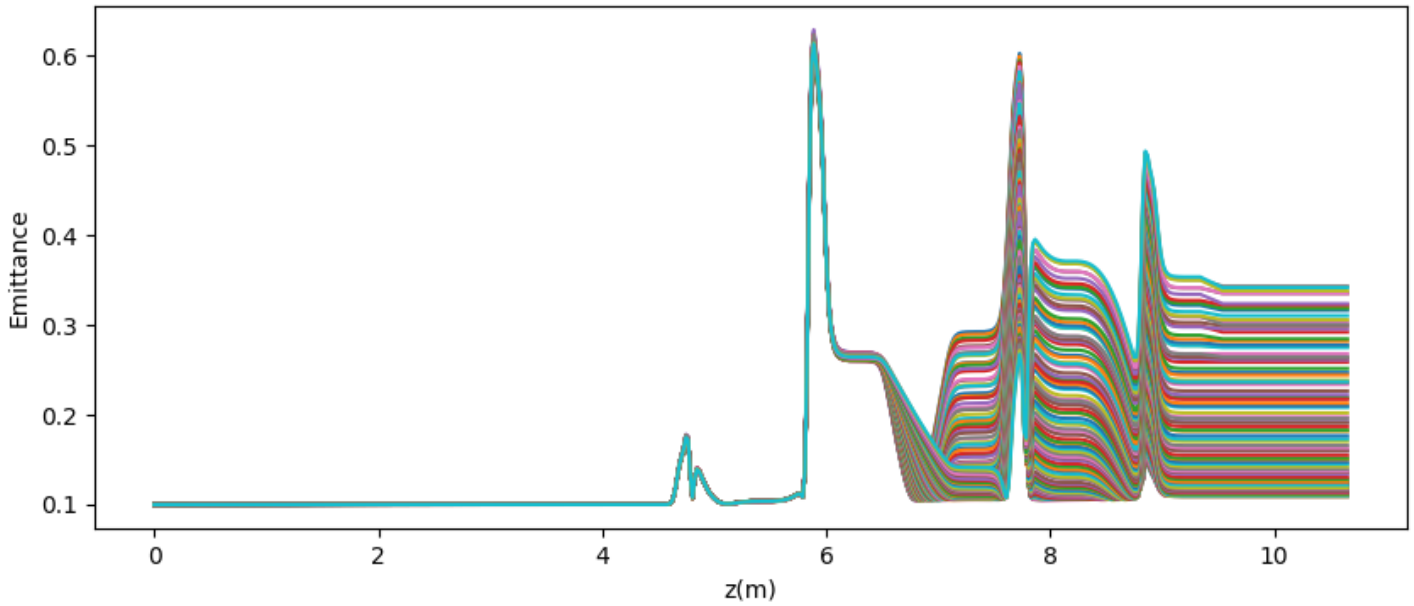
[4] ✓ 11.9s

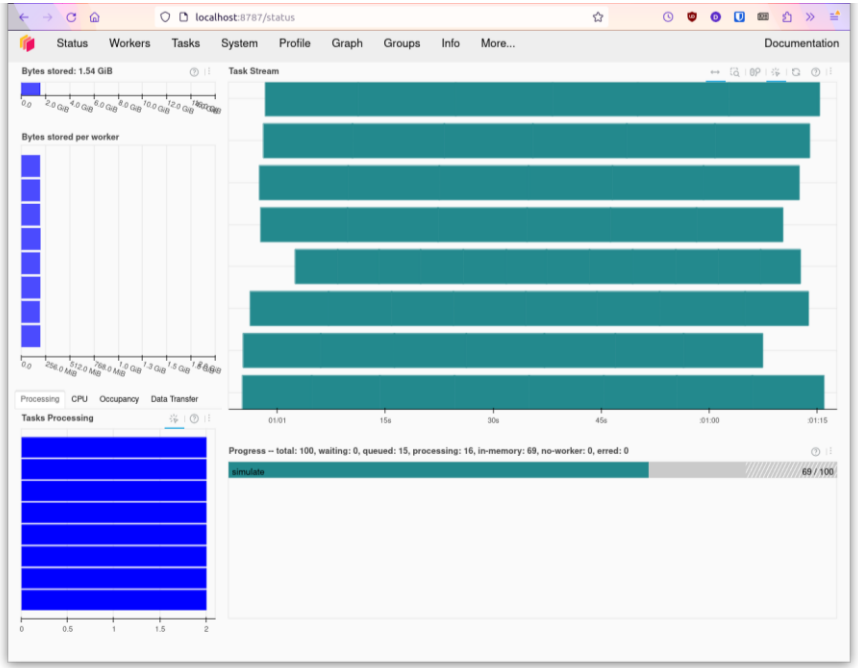
```
[CompletedProcess(args=['tracelx64', 'CB_newMRMS_RFQ_Fields_1.ini', 'hide', 'ele[26][6]=-1400'], returncode=0),
      ##       z(m)    gama-1            x0            y0         p0  \
 0     0  -0.000000  0.000006  2.896655e-09 -2.276210e-08  -0.017999
 1     1   0.000000  0.000006  2.896655e-09 -2.276210e-08  -0.017999
 2     2   0.214000  0.000006  1.181234e-08 -5.706434e-08   2.677344
 3     3   0.326000  0.000006  1.647850e-08 -7.501692e-08   4.087991
 4     4   0.646000  0.000006 -7.434411e-08  1.198164e-07   4.088095
```

Parallel job submission

```
runs = []
for i in range(100):
    runs.append(client.submit(simulate, {"ele[26][6]": -1400+i*10}))

data = client.gather(runs, errors="skip")
```

[15] ✓ 54.9s





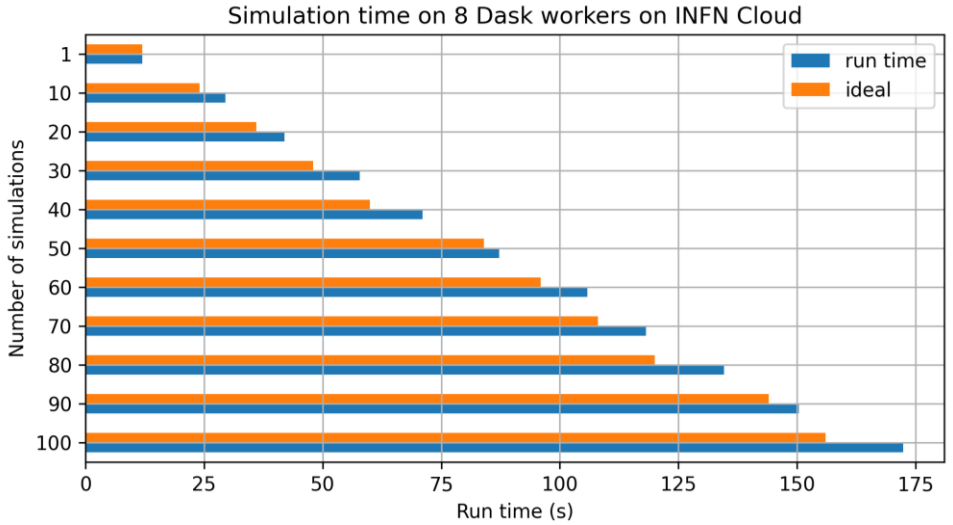
More advanced Dask API can be used

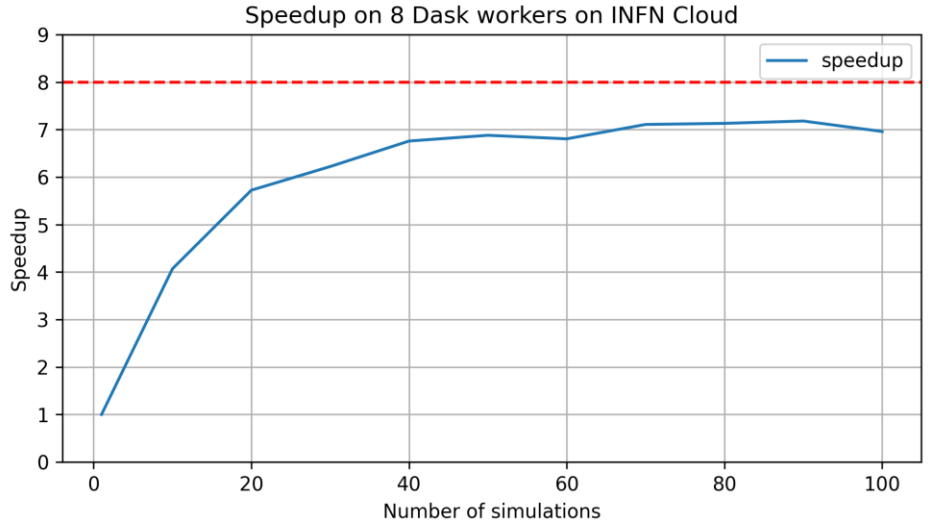
`dask.DataFrame`

Testing:
INFN Cloud
CloudVeneto

INFN Cloud

- Using Kubernetes-as-a-Service
  - Deployed on RECAS-BARI
  - Resources from the admin/catchall group
  - 1 k8s Master (4CPU, 8GB)
  - 4 k8s Nodes (2CPU, 4GB)
  - Open ports 8786, 8787
- Dask Cluster
  - 1 Scheduler (4CPU, 8GB)
  - **8 Workers** (1CPU, 1GB)
- Simulating ADIGE beam line @ LNL
  - From charge breeder to emittance meter
  - Including MRMS





CloudVeneto CaaS

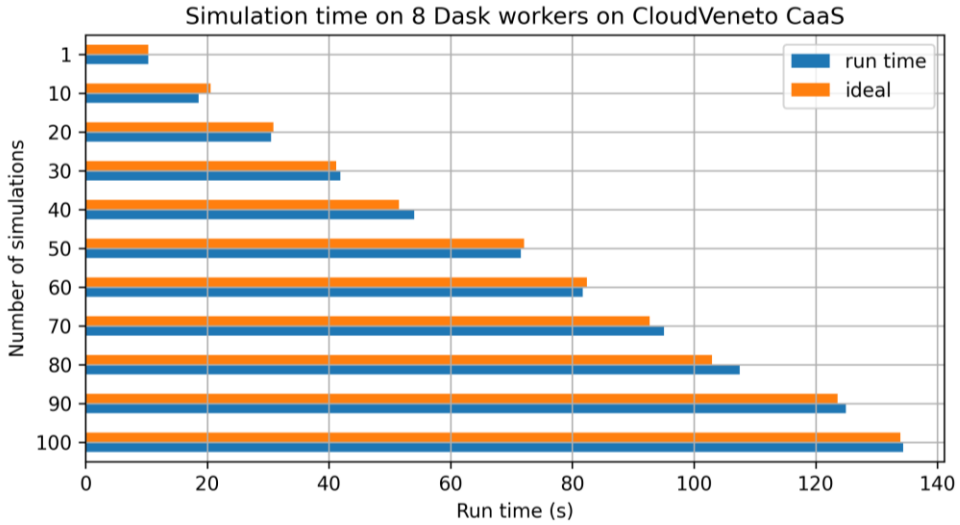
- Managed k8s cluster
  - Avoid instantiation/maintenance
- Large pool of resources available to us via SPES project
- Thanks to Lisa Zangrando for the support
  - Helm operator installation
- Using OSNodes on openstack
  - VMs using resources from own tenant quota

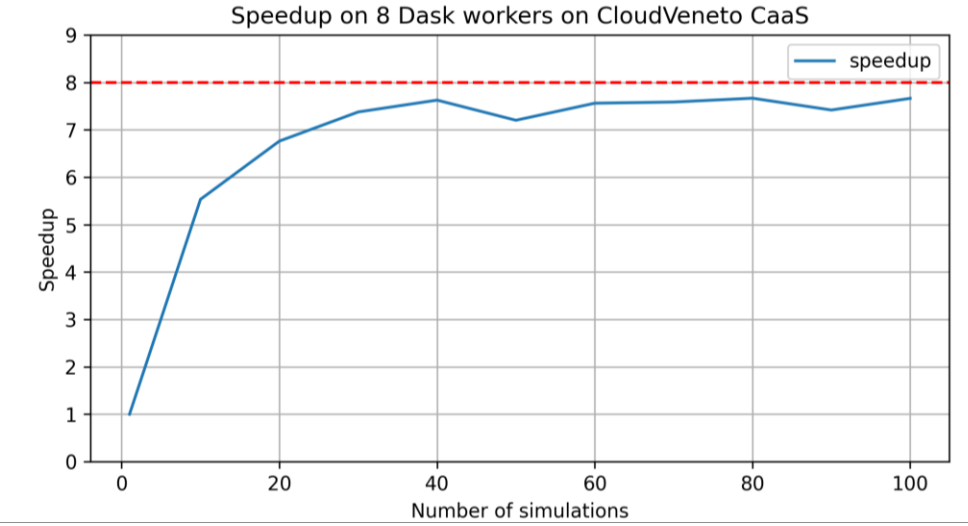
```
kubectl apply –f osnodes.yaml
```

```yaml
---
apiVersion: osnode.infn.it/v1
kind: OpenStackNode
metadata:
  name: k8s-osn-00
spec:
  flavor: cloudveneto.xlarge
  keyPair: pc1815
  policy: shared
---
apiVersion: osnode.infn.it/v1
kind: OpenStackNode
metadata:
  name: k8s-osn-01
spec:
  flavor: cloudveneto.medium
  keyPair: pc1815
  policy: shared
```

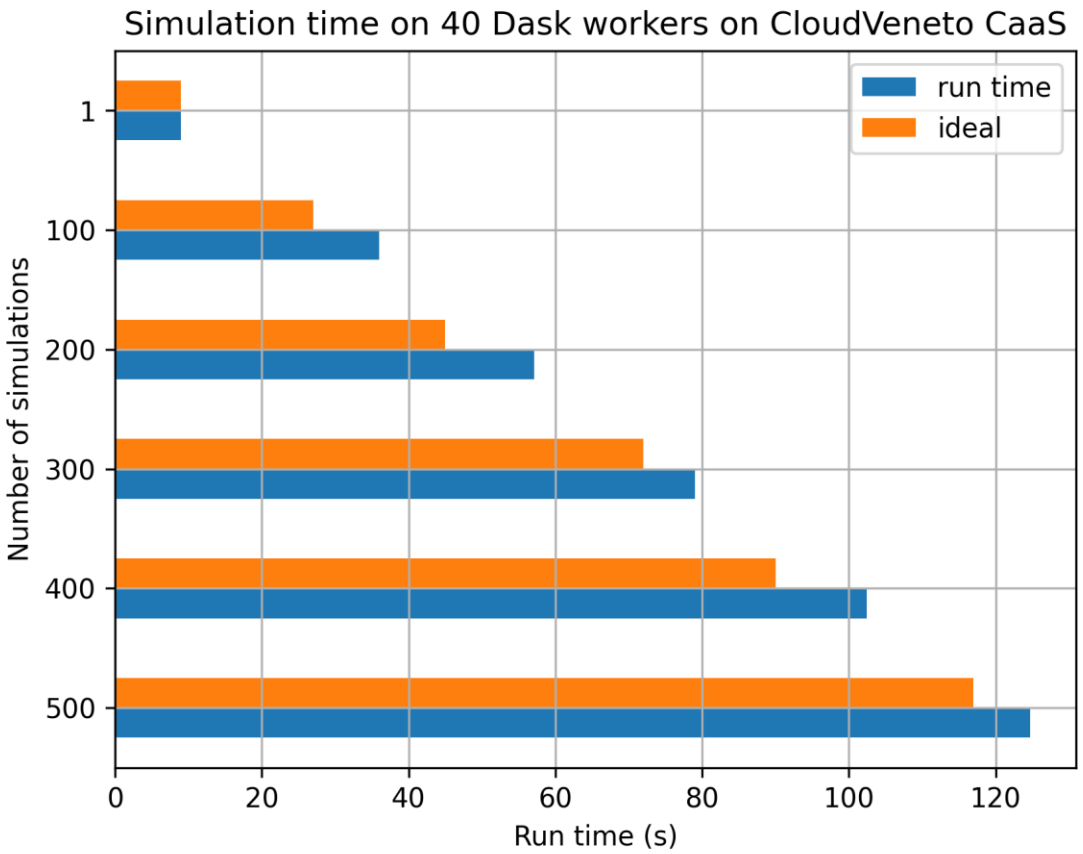
CloudVeneto

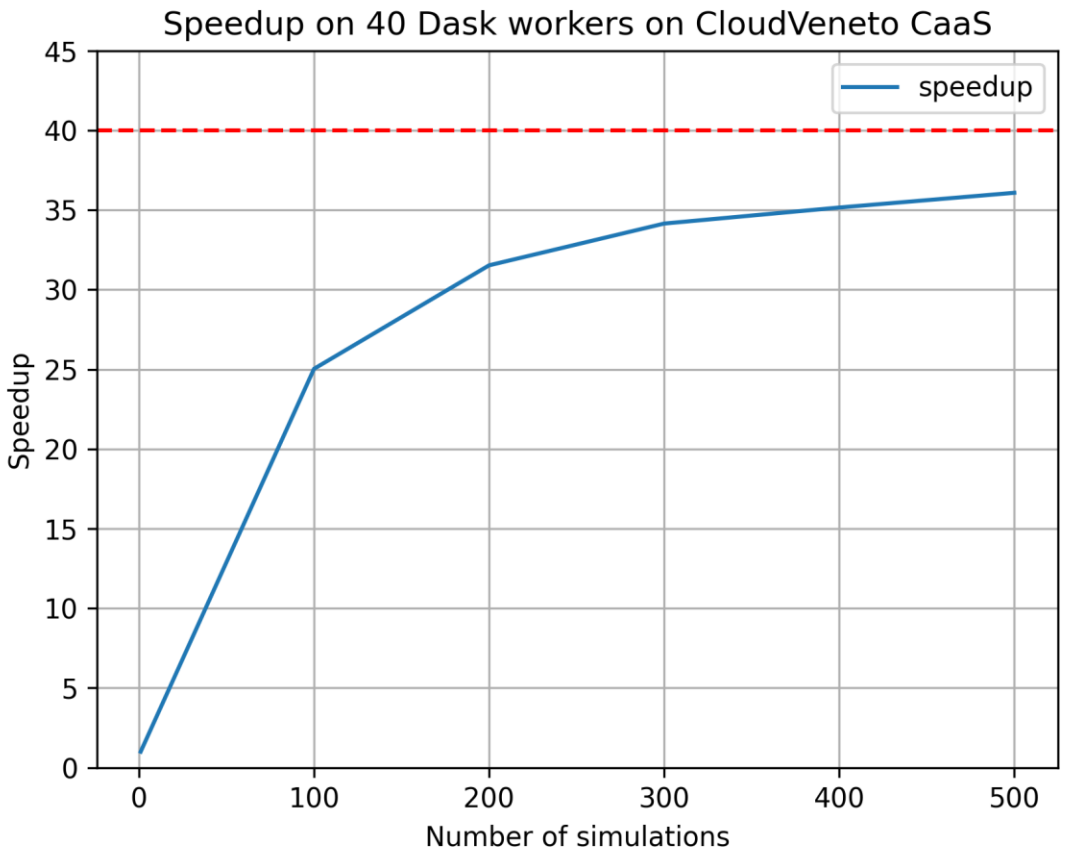
- CaaS cluster
  - 1 OSN xlarge (8CPU, 16GB)
  - 30 OSN medium (2CPU, 4GB)
- Dask Cluster
  - 1 Scheduler (8CPU, 16GB)
  - **Up to 40 Workers** (1CPU, 1GB)
- Limited by the memory on the scheduler
  - Used to distribute the input files





Scaling to 40 workers





Conclusion

What's next

- Manage distribution of I/O files
  - Kubernetes volumes
  - In-memory volumes to get fast I/O
- Improve accessibility for end users
  - Dask Cluster as a service?
  - Authentication?
- Better debug messagges,
  - e.g. failed computation for TraceWin
- Being used in combination with off-policy RL algorithms!

Thank You

davide.marcato@lnl.infn.it
daniel.lupu@lnl.infn.it