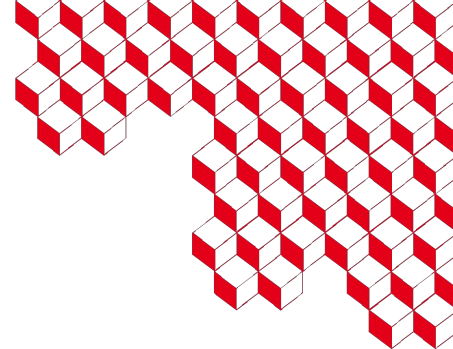
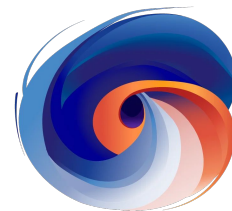




Modern Packaging for NumPEx Applications

Benoît MARTIN - CEA/MdIS



RESEARCH
PROGRAM

DIGITAL
FOR EXASCALE

Workshop on Computing Strategy@CEA - 24/09/2026

Context



RESEARCH
PROGRAM

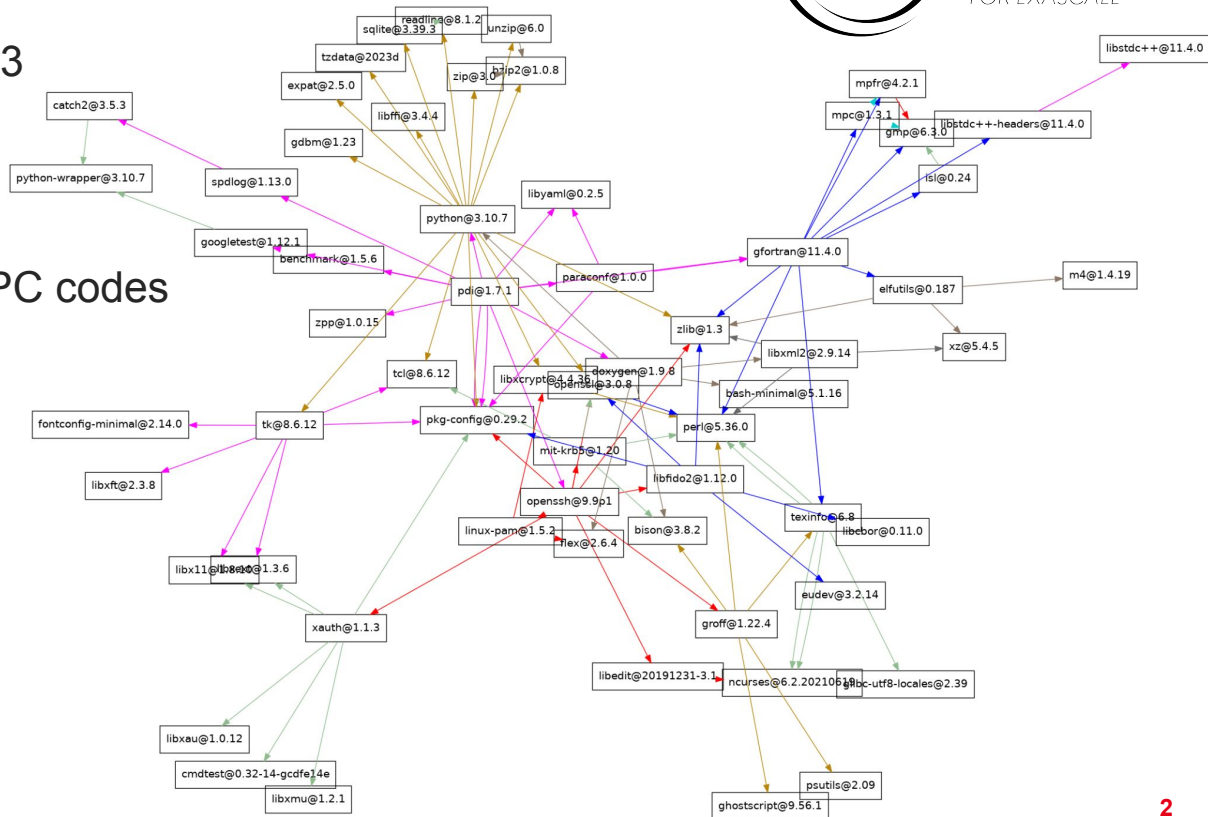
DIGITAL
FOR EXASCALE

PEPR NumPEX Exa-DI (PC5) WP3

Objective: package and deploy HPC codes

Needs:

- Performance
- Portability
- Development environment
- Reproducibility



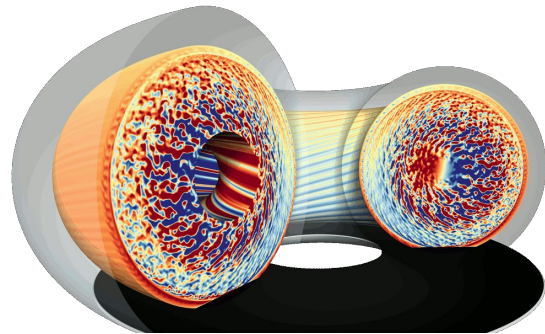
Example - Gysela

Legacy Gysela

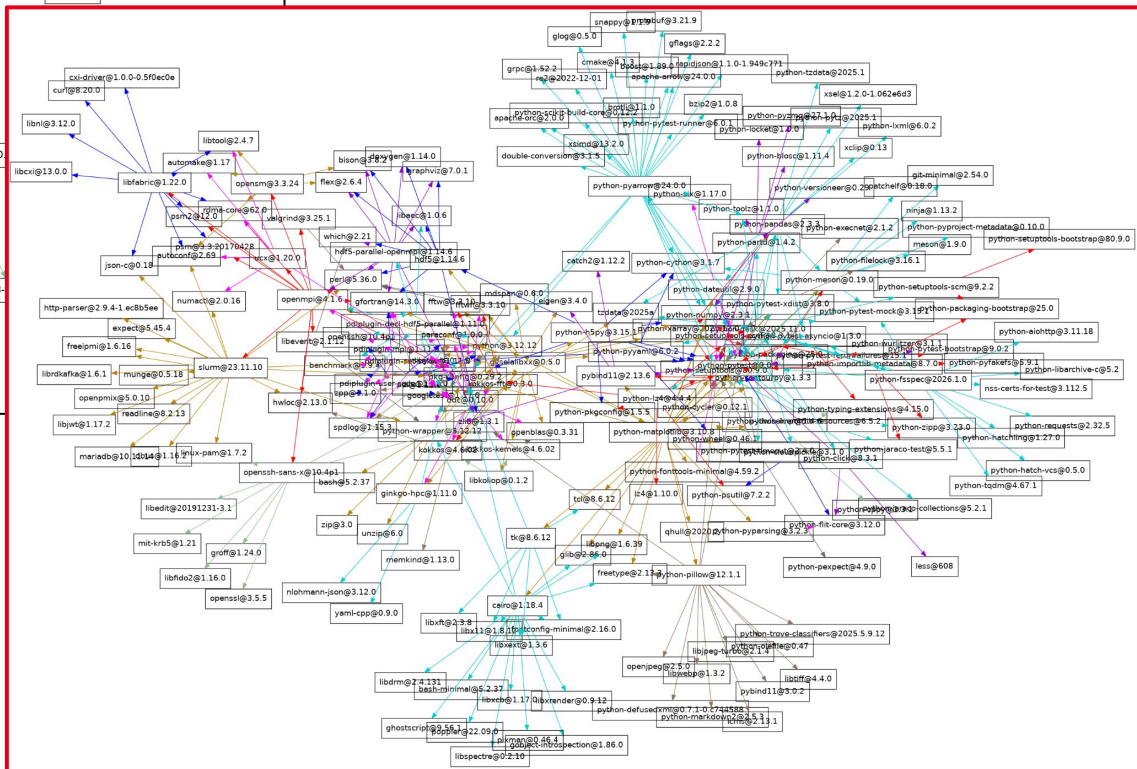
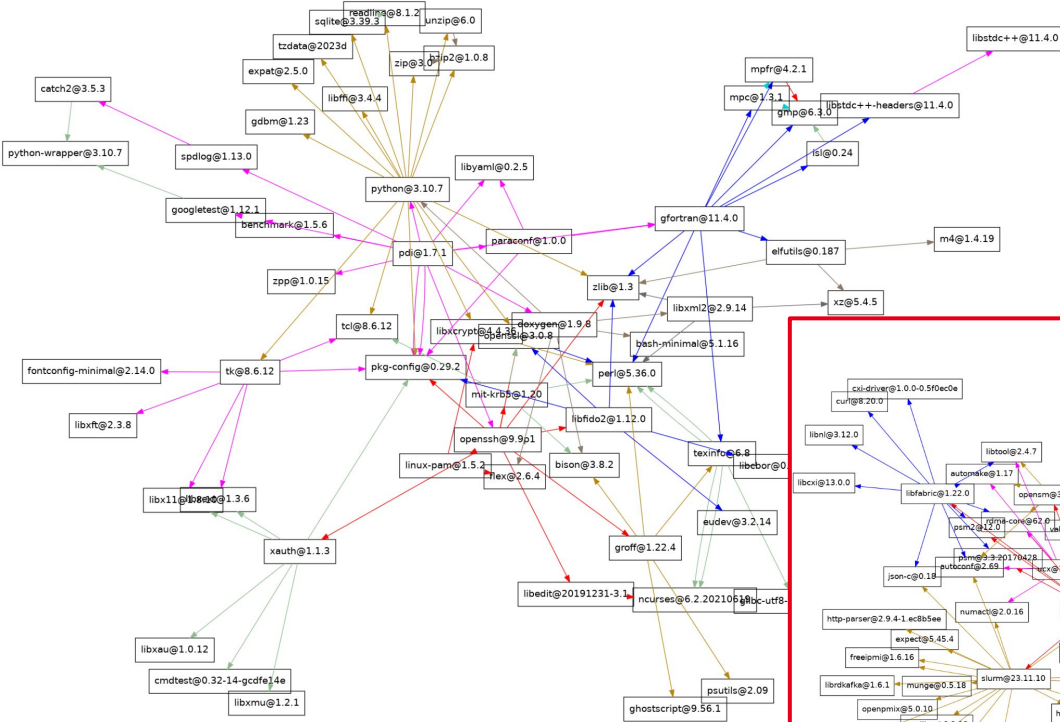
- ❑ Plasma physics for nuclear fusion (DRF/IRFM)
- ❑ Study core turbulence in tokamaks
- ❑ ~30 years old Fortran 90 code (~40 kLOC)
- ❑ “Easy” to install: few dependencies (LAPACK, OpenMP, MPI, HDF5)

Gyselalib++

- ❑ Prepare for exascale
- ❑ Modern, reusable modules
- ❑ Libraries have many compile options
- ❑ Some are available on clusters, but not always built with required options
- ❑ Non-exhaustive list of direct dependencies: DDC, Eigen, Koliop, Kokkos, Kokkos-fft, Kokkos Kernels, Google test, Ginkgo, MPI, PDI, NetCDF, HDF5, Python, Deisa, Dask, Xarray, h5py, matplotlib, ...

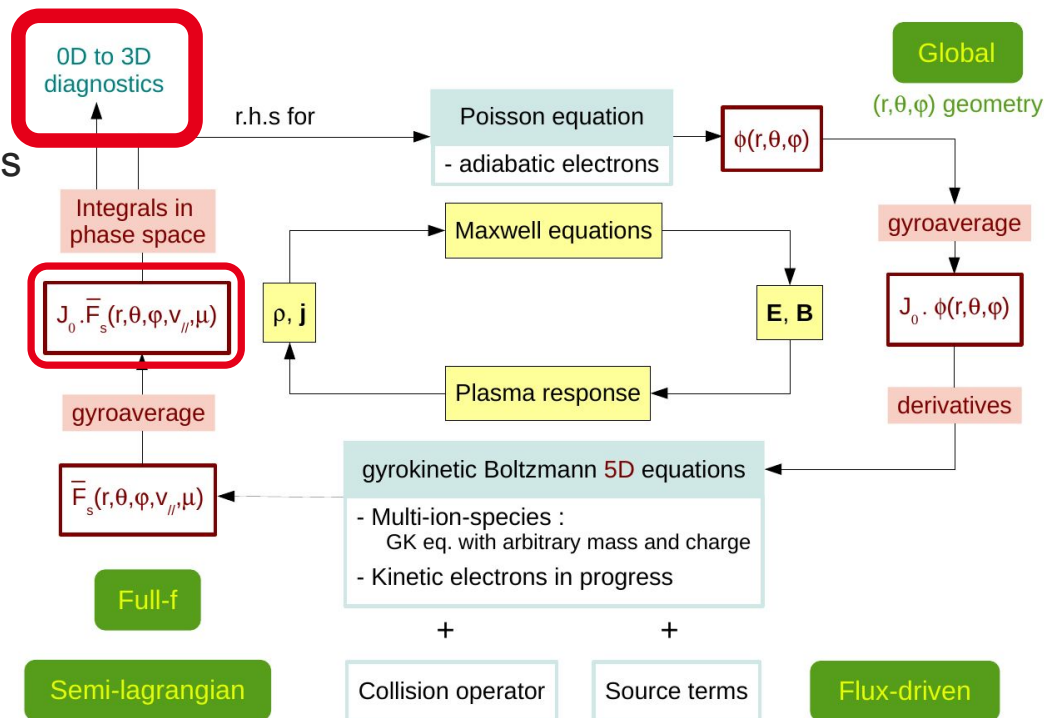


Challenge 1: complex dependency graph

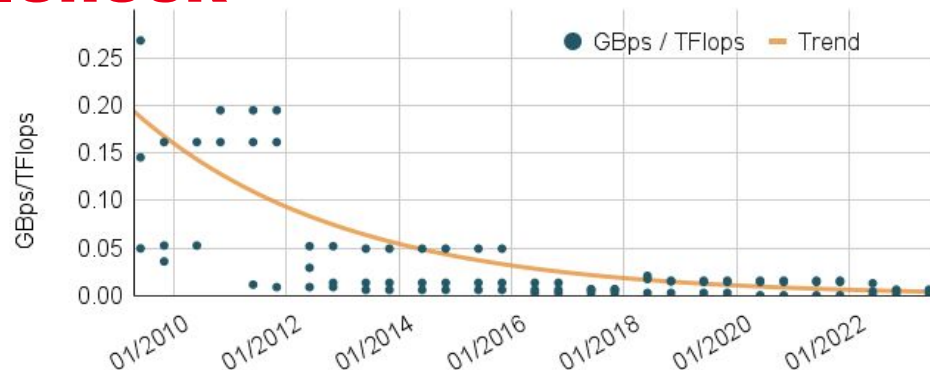


Challenge 2a: very large data

- ❑ 3D means “small”: ~GB or so
- ❑ 5D is where the real space usage is
- ❑ 1 single variable f fills $\frac{1}{4}$ of RAM
 - Of the full cluster
 - That's ~100TB on Joliot-Curie
- ❑ You don't write to disk
 - (or not too often)
 - Diagnostics instead

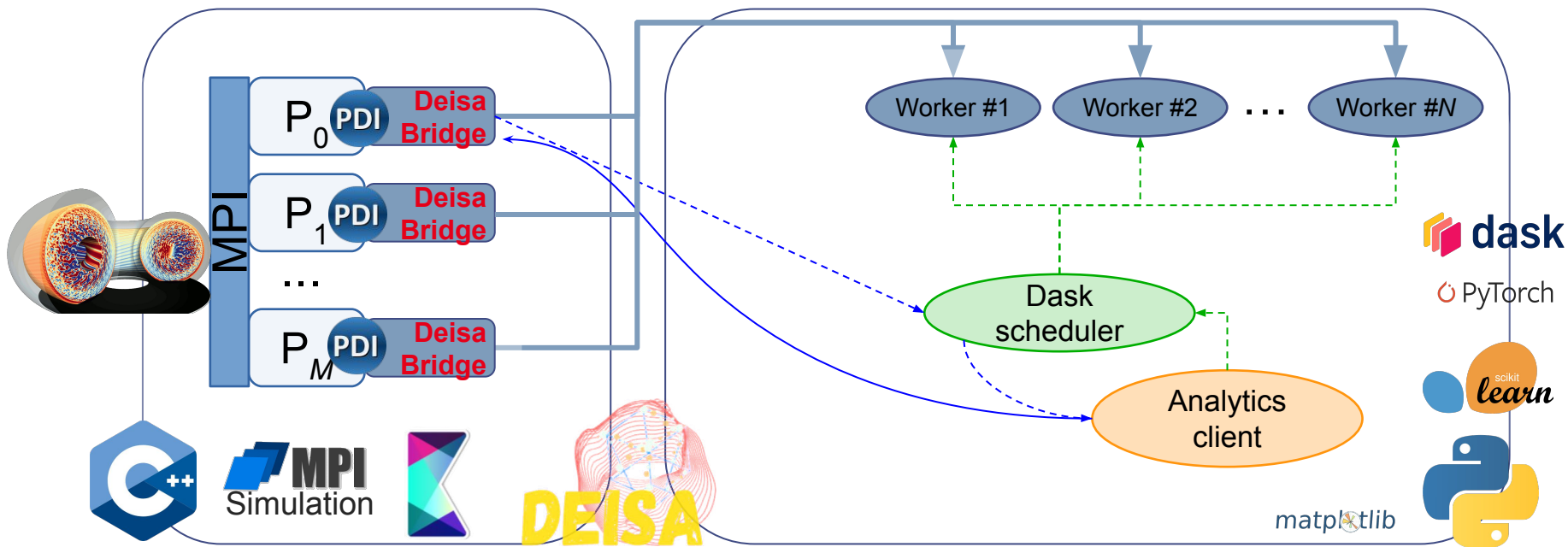


Challenge 2b: I/O bottleneck



- ❑ Exascale
 - Compute bandwidth **exceeds** memory bandwidth: performance bottleneck
 - Post-hoc analysis no longer feasible: need to release tension on the file system
- ❑ Solution: In-situ data processing
 - Stream simulation data to dedicated analytics nodes
 - Only write to disk: statistics, zones of interest, reductions, ...
 - Reduce tension on the file system

Complex dataflow: in-situ analytics using Deisa

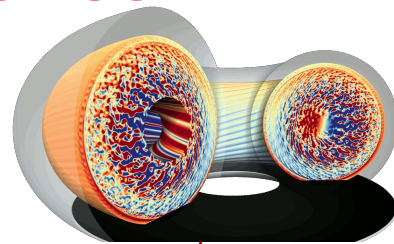


Problems:

- ❑ **Multi language:** C++, Python
- ❑ **Many** libraries

Challenge 3: portability and performance

- ❑ **Portability:** non-heterogeneous machines
- ❑ **Performance:** system libraries/drivers (i.e.: GPU/network)
- ❑ **Development environment:**
 - Source-code for dependencies
 - Easy for non-experts
 - Control over compile options
 - Run on laptop, local cluster
- ❑ **Reproducibility:** re-run experiments



	Jean-Zay	Adastral	Alps	Alice Recoque
CPU	<u>Intel</u>	<u>AMD</u>	<u>AMD</u>	<u>AMD</u>
GPU	<u>Nvidia</u> V100, A100, H100	<u>AMD</u> MI250X	<u>Nvidia</u> GH200, A100, <u>AMD</u> MI250X, MI300A	<u>AMD</u> MI430X
Interconnect	Intel Omni-Path, InfiniBand	Cray SlingShot	Cray SlingShot	BX13

How we deploy software

Manual installation

- ❑ `git clone, cmake, make install`
- ❑ Time-consuming, error-prone, fragile scripts

Environment modules

- ❑ `module load gysela`
- ❑ Cleaner than manual install
- ❑ Specific to each cluster, different between centers
- ❑ Limited to admin-provided versions, not reproducible

Alternatives you may know

- ❑ **conda / pip**
 - great for Python
 - weak for C/C++, MPI and accelerators
 - Not tuned per machine
- ❑ **Easybuild**
 - solid on a single cluster
 - less portable across centers
- ❑ **Docker** is not HPC-native

The landscape is rich. The right choice depends on the goal.
We focus on **Guix** and **Spack** because they are HPC-ready.

Why Guix and Spack

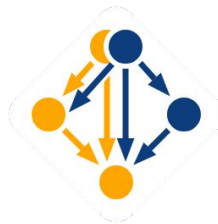
GNU Guix (hpc.guix.info)

- ❑ Pure dependency tree: no reliance on system libraries
- ❑ Excellent reproducibility
- ❑ Very easy container generation
- ❑ Strong HPC community



Spack (spack.io)

- ❑ Can use system libraries: flexible
- ❑ More package versions and variants
- ❑ Widely adopted on HPC clusters in the US
- ❑ Strong HPC community



Guix in a nutshell

A package manager as a pure functional store.

```
# classic, fragile
tar xvf mylibrary.tar.gz && make

# with Guix
guix install mylibrary

# temporary development shell
guix shell -D mylibrary

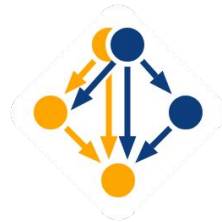
# declarative "requirements.txt"
guix shell cmake -m manifest.scm
```

The store

- ❑ Every build lands in `/gnu/store/<hash>-...`
- ❑ A pure environment does not touch system libraries
- ❑ Packaging definitions are written in Scheme

```
(specifications->manifest
 (list "cmake" "openmpi"
       "gcc-toolchain"))
```

Spack in a nutshell



A package manager that is flexible about the machine.

```
# classic, fragile
tar xvf mylibrary.tar.gz && make
```

```
# with Spack
spack install mylibrary
```

```
# declarative environment
spack env create -d ./myenv
spack env activate ./myenv
spack add cmake openmpi
```

Specs and variants

- ❑ Variants and targets are first-class
(+cuda, target=cascadelake)
- ❑ Packaging definitions written in Python

```
spack:
  specs:
    - cmake
    - openmpi
    - gcc
```

What the two share: no package collisions

Both Guix and Spack use content-addressed stores: each distinct configuration gets its own hash

- ❑ Two builds for two CPUs coexist on the disk, no conflict
- ❑ Old version stay, so old results stay reproducible
- ❑ The hash depends on the full input graph: one change anywhere is captured

```
guix build gyselalibxx  
→ /gnu/store/0001-...-gyselalibxx  
guix build -tune=cascadelake gyselalibxx  
→ /gnu/store/0002-...-gyselalibxx  
(simplified hash)
```

Violating dependencies (“works on my machine”) becomes the exception, not the rule.

Comparing Guix and Spack

Aspect	GNU Guix	Spack
Dependencies	Pure, no system libs	Can use system libs
Reproducibility	Excellent	Good
Flexibility	Lower	Higher
Version / variants	Fewer	More
Packaging language	Scheme	Python
Tuning	-tune=cpu	target=cpu
On cluster	Tier-1, Tier-2, Tier-3	Nearly everywhere
Containers	Guix pack	Build-cache push

The hard reality of supercomputers

Package managers alone are not enough on the biggest machines

Guix

- ❑ Installed by system admin
- ❑ Available on some Tier-1, Tier-2, PC
- ❑ Not available (yet) on Tier-0

Spack

- ❑ Can be installed by user
- ❑ Installs almost everywhere
- ❑ **Inode saturation** on shared filesystems

Containers

- ❑ Available on Tier-0, Tier-1, Tier-2
 - Singularity/Apptainer
- ❑ Laptop and desktops
 - Docker/Podman
- ❑ Can be generated from Guix and Spack
- ❑ Isolation from the system
- ❑ Easy to share

Building a container from Guix or Spack

From Guix

```
guix pack -f squashfs -r image.sif \  
  bash gysela  
  
# ship & run anywhere  
scp image.sif lumi  
srun --pty singularity shell  
image.sif
```

(bash is required by Singularity)

From Spack

```
# generate a Dockerfile  
spack containerize > Dockerfile  
  
# or push a build cache  
spack buildcache push \  
  --base-image ubuntu:24.04 \  
  myregistry gysela  
  
singularity shell \  
  docker://myregistry/.../gysela.spack
```

Performance on the real machine

- ❑ **MPI:** Two deployment styles
 - Bring-your-own MPI: ship MPI inside the container
 - Load MPI from the host: requires ABI matching
- ❑ **CPU tuning**
 - Pass the machine's instruction set down to every package
 - `-tune=cascadelake / target=...`
- ❑ **GPU**
 - CUDA and ROCm packages exist in Guix and Spack
 - Container run with the right flag (`-nv` for Singularity)

The open question we carry

How to load system libraries and drivers (GPU, network) inside a container, Guix or Spack stack, without losing the native path ?

What works well

- ❑ **Package managers remove the pain** of assembling a stack manually or depending on the cluster's own modules.
- ❑ **Containers avoid inode limits:** a squashfs image is a single large file instead of many smaller files.
- ❑ **The declarative file is the asset:** a `manifest.scm` or `spack.yaml` committed to the repo rebuilds the whole environment anywhere.
- ❑ **HPC catalog:** dependencies like OpenMPI, MPICH, CUDA, ROCm, Kokkos are already packaged, maintenance is community driven (relevant for Gyselalib++).
- ❑ **Before/after experience is stark**
“`guix install mylibrary`” versus days of make assembly. Users feel the difference immediately.

What did not go as smoothly

- ❑ **Build matrix hell:** CPU/GPU architecture and libc version combinations balloon quickly.
- ❑ **Spack spec concretization:** powerful but hard to master. Surprises when a spec resolves unexpectedly.
- ❑ **System libraries and drivers:** GPU/network stacks are per-center. Inside the stack or the container ?
- ❑ **Guix not deployed on many clusters.** We could not rely on it where it matters most. Containers became mandatory.
- ❑ **Spack inode saturation** on shared filesystems.
- ❑ **Center-specific constraints** for containers: safe directory (Jean-Zay), admin validation (Adastra), PCCOC (TGCC).
- ❑ **Naming friction** cuda vs cuda-toolkit, variants as +cuda vs a -hip package.

What we are doing now

- ❑ **Working with computing centers** to adopt a native and modern package manager
- ❑ **Writing Guix and Spack recipes** with the community
- ❑ **Building a binary-cache** to speed compilation/deployment
- ❑ **Performance benchmarks** to check that the built binaries are well compiled
- ❑ **Automation** in CI/CD:
 - <https://github.com/numpex/spack-stack>
 - <https://guix.bordeaux.inria.fr>
- ❑ **Training:** <https://numpex-pc5.gitlabpages.inria.fr/tutorials>

The team

Inria Bordeaux

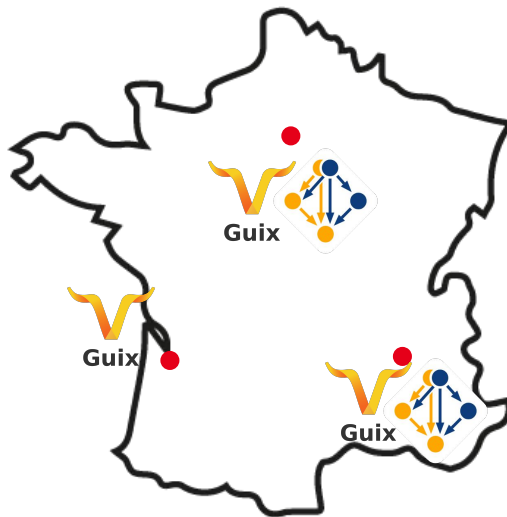
- ❑ Ludovic COURTES
- ❑ Vivien KRAUS
- ❑ Sergio PASTORE PEREZ
- ~~❑ Romain GARBAGE~~

CEA Saclay

- ❑ Benoit MARTIN
- ❑ Thomas BOUVIER

Inria Grenoble

- ❑ Bruno RAFFIN
- ❑ (Pierre NEYRON)
- ~~❑ Fernando AYATS~~



RESEARCH
PROGRAM

DIGITAL
FOR EXASCALE