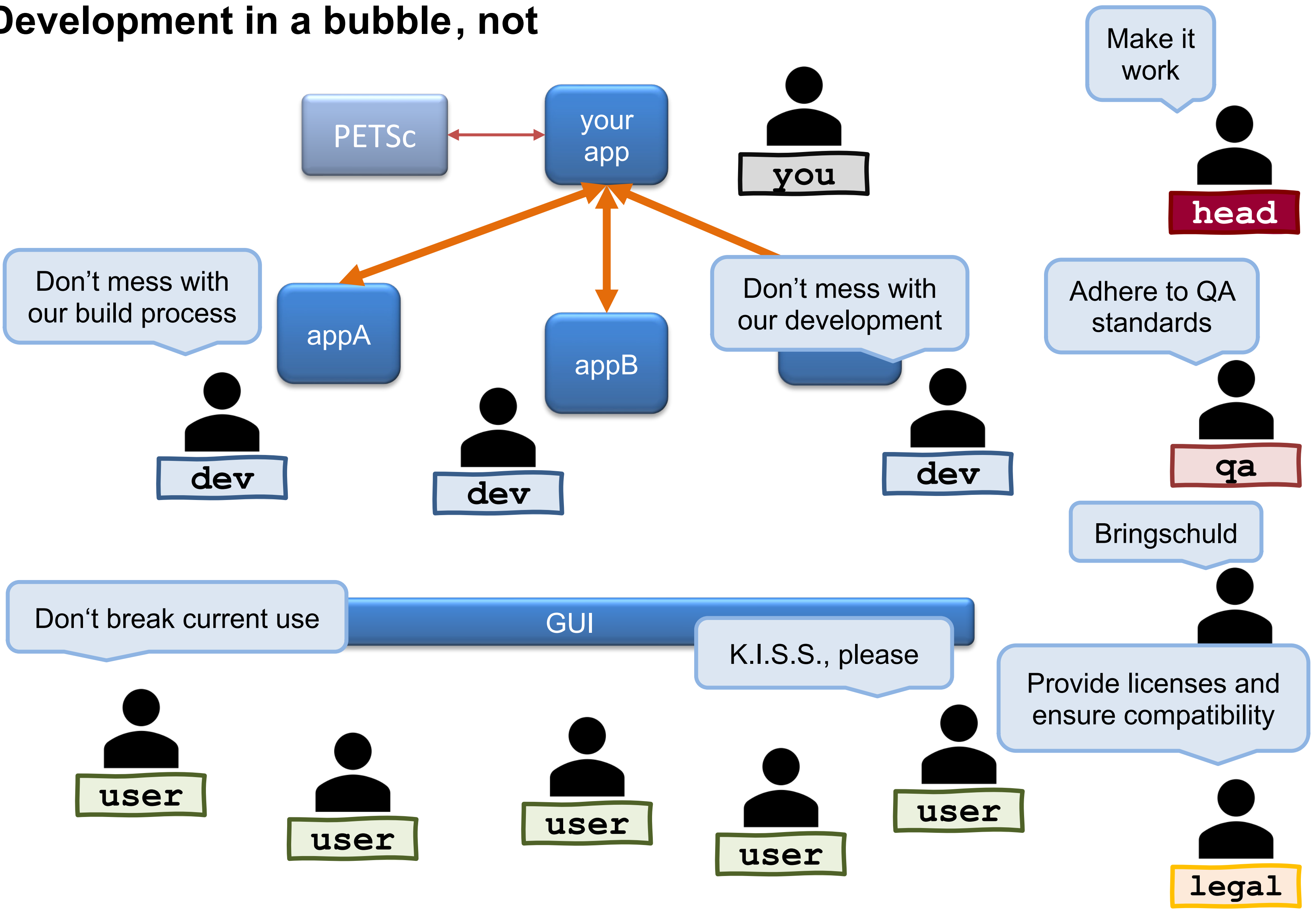
Software Development and Deployment Including PETSc

Volker Jacht, Tim Steinhoff, GRS

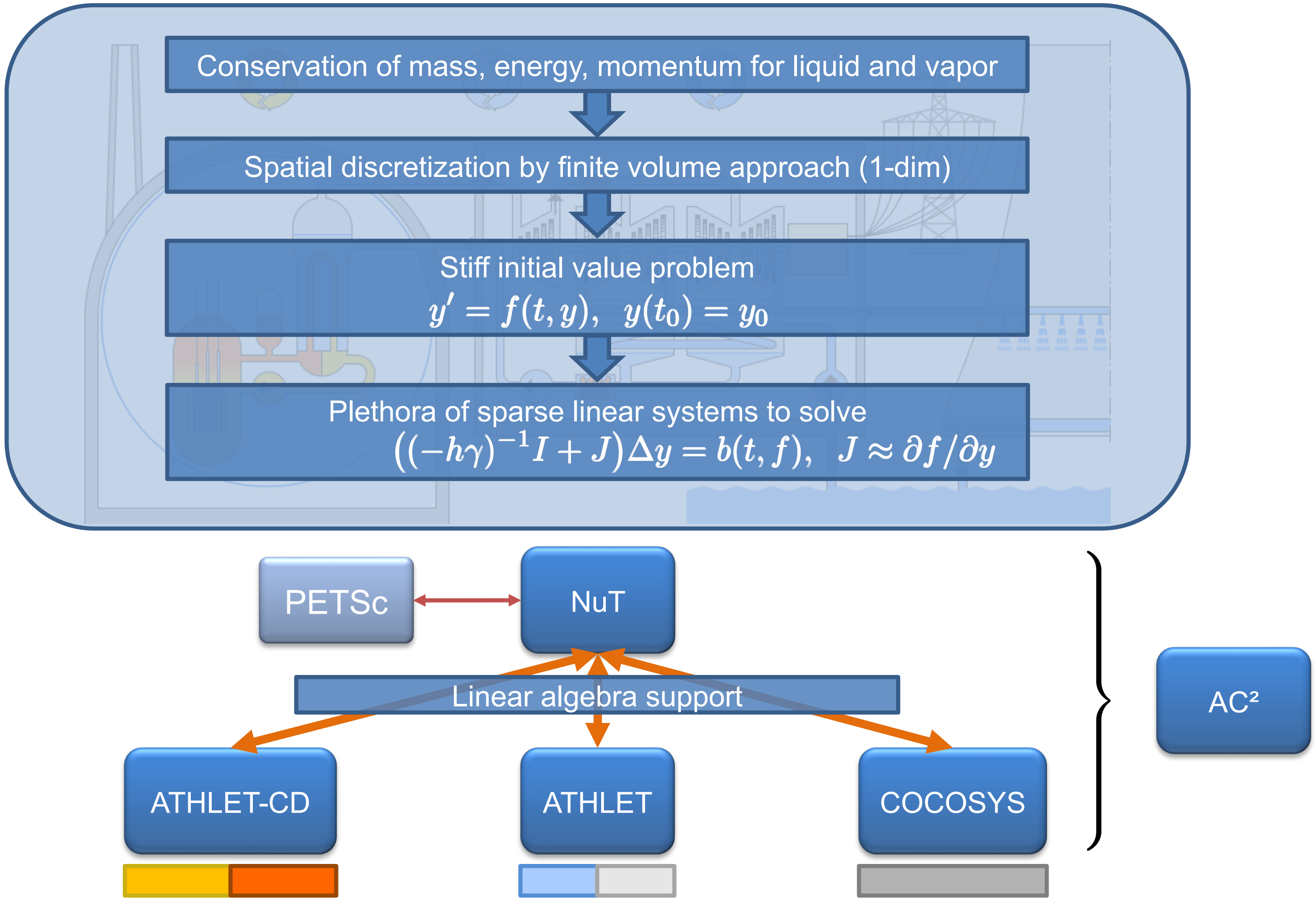
2023-06-05

PETSc Annual Meeting 2023

Development in a bubble, not



Application background



Conservation of mass, energy, momentum for liquid and vapor

Spatial discretization by finite volume approach (1-dim)

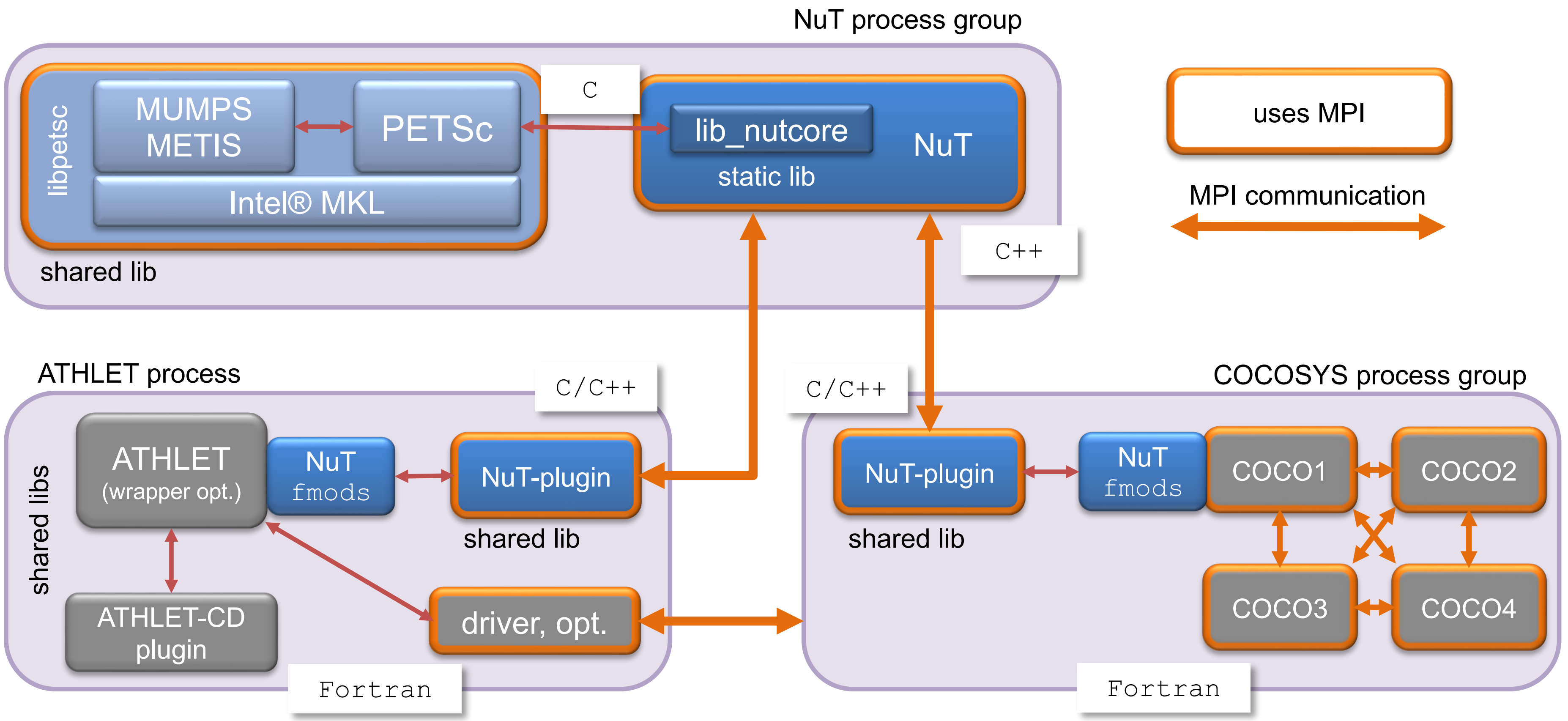
Stiff initial value problem

$$y' = f(t,y), \quad y(t_0) = y_0$$

Plethora of sparse linear systems to solve

$$\left((-h\gamma)^{-1} I + J\right)\Delta y = b(t,f), \quad J \approx \partial f / \partial y$$

Software architecture in AC²



Due to plugin concept NuT (and PETSc) are completeley optional **dev** **user** 👍

FDE — Fortran Development Extensions
plugin, hooks, hashmaps and lots more
gitlab.com/Zorkator/libfde – LGPL v3

MMA — MPI for Multiple Applications
initiate communication
gitlab.com/nordfox/mma – BSD-2

Providing data

All data is made available locally on

- self-managed **GitLab** instance or
- dedicated object storage (**MinIO**)

- versioning
- easy and reliable access

qa

depl

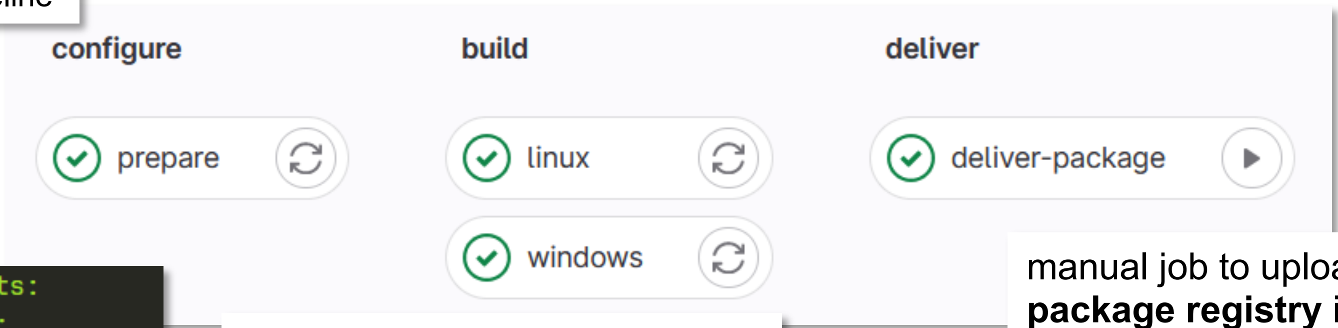
native Git repository	mirror	object storage
NuT + plugin / fmods	PETSc (gitlab/petsc)	Intel® MKL offline installer
ATHLET	MUMPS (bitbucket/petsc/pkg-mumps)	Intel® MPI offline installer
ATHLET-CD	METIS (bitbucket/petsc/pkg-metis)	compilers offline installer
COCOSYS	FDE (gitlab/…/libfde), MMA (gitlab/…/mma)	Docker images (build / test environments)

Making PETSc available – the PETSc Builder project

Pre-build PETSc and create suitable archive

- `devs` don‘t need to bother with details, no dedicated environment required (no Cygwin!) `dev` 👍
- use dedicated branches from mirrors (minor build-fixes, **added licenses**) `legal` 👍
- use **Docker** executor to provide clean and reproducable build environment `qa` 👍
- ensure binary compatibility by using build environment defined by internal coding policy

CI-pipeline



```yaml
artifacts:
  paths:
    - petsc.tgz
    - metis
    - mumps
    - licensing
```

1. install MKL
2. configure and build PETSc
   - plus MUMPS and METIS
3. discard unnecessary files
4. create **archive**
   - lib binaries
   - header files
   - licenses

manual job to upload archive to **package registry** if things are fine

- `PETSC_VERSION_NAME`
- `CI_PROJECT_ID`
- `CI_PIPELINE_ID`
- `IMAGE_TAG`

`qa` 👍

```
petsc_3.17.5-grs1_build. /
467.43599_ /
21.0-windows-1809.zip
```

CMake in breve

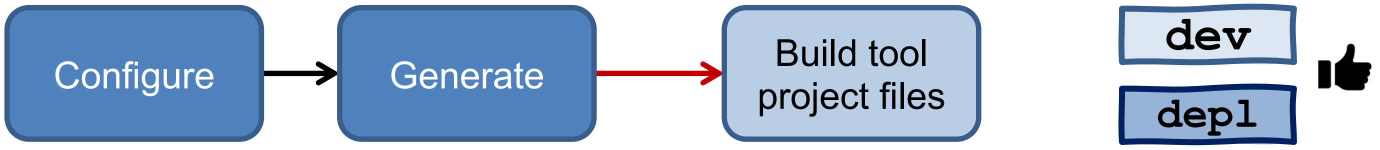
Without a build system, a project is just a collection of files. CMake brings some order to this, starting with a human-readable file called `CMakeLists.txt` that defines

- what should be built and how,
- what tests to run and
- what package(s) to create.

This file is a **platform independent description** of the whole project, which CMake then turns into **platform specific** build tool project files.

Professional CMake: A Practical Guide – Craig Scott, Ch. 2, add. formatting

- popular build tools like **`make`** and **Visual Studio** supported
- dedicated build-folder
- scopes of settings defined by hierarchical order of `CMakeLists` files
- **easy to run**, some work to set up



Using CMake(It)

ATHLET / COCOSYS root `CMakeLists.txt`

```
cmi_add_git(nut ssh://git@gitlab.grs.de/grs/ac2/nut/nut.git <commit-id>)
...
cmi_add_subdirectory(nut)
```

- `git clone nut <commit-id>` to `_externals/nut`
- run NuT's root `CMakeLists.txt`

NuT

generate target to produce NuT's host-sided interfaces (`.fmod`, `.h`, `.hpp`) based on `.json` input

```
if(BUILD_NUT) # set by user or by AC² deployment
  cmi_add_archive(petsc "${AC2_PACKAGE_REGISTRY}/petsc/<petsc-archive>")
  cmi_add_subdirectory(petsc NO_CMAKE)
```

download archive and extract to `_externals/<petsc-archive>`

set up dependency on libpetsc and create build targets for NuT

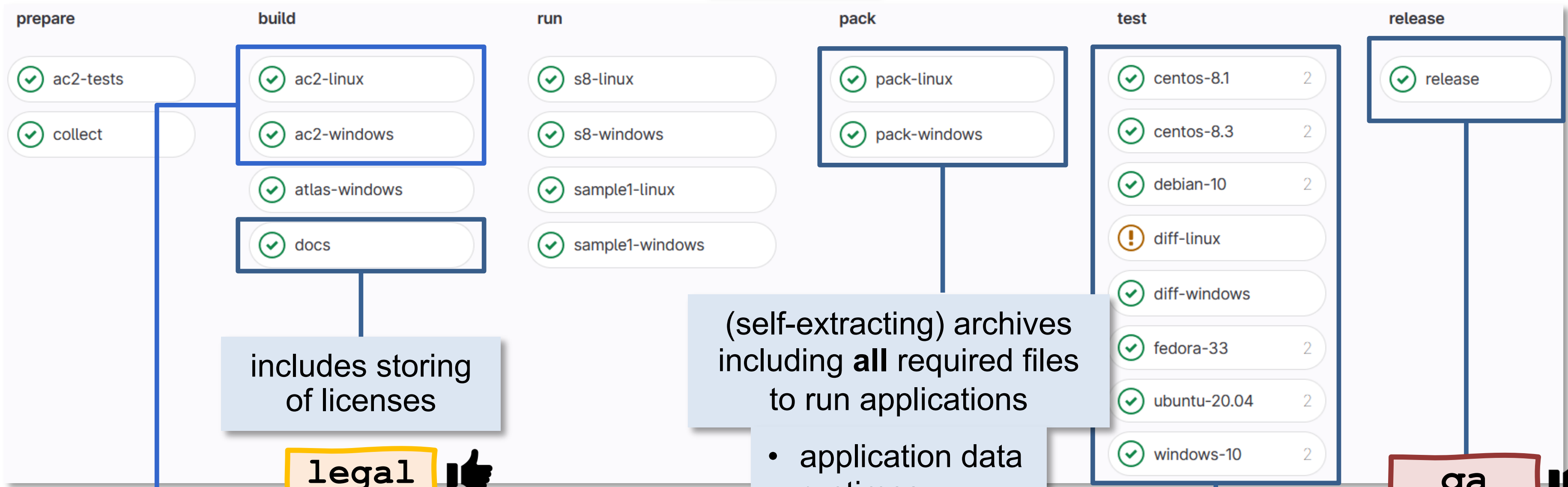
continue creating build targets for **ATHLET / COCOSYS**

CMakeIt (`cmi`): gitlab.com/nordfox/cmakeit – BSD-2

`dev` `depl`

AC² deployment

CI-pipeline



includes storing of licenses

legal 👍

(self-extracting) archives including **all** required files to run applications

- application data
- runtimes
- libpetsc
- mpiexec

user 👍

building of applications including NuT

BUILD_NUT = 1

PETSc as pre-built library included

depl 👍

clean test environments by **Docker** images

dev 👍

qa 👍

stored in **Releases**

Usability

default settings

```
mpiexec -n 1 nut_host_ex01 : -n 1 nut_worker
```

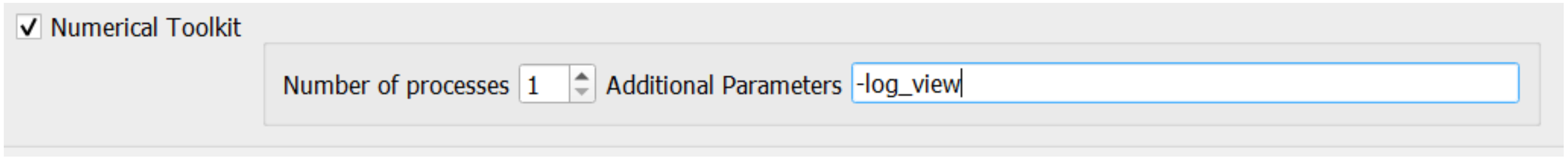
specific solver preset

```
mpiexec -n 1 nut_host_ex01 : -n 1 nut_worker -solver ilu_2-gmres
```

unlock arbitrary petsc command line options

```
mpiexec -n 1 nut_host_ex01 : -n 2 nut_worker -<dev_flag> -ksp_view
```

NuT via GUI



✓ Numerical Toolkit

Number of processes: 1 Additional Parameters: -log_view

user 👍

Summary

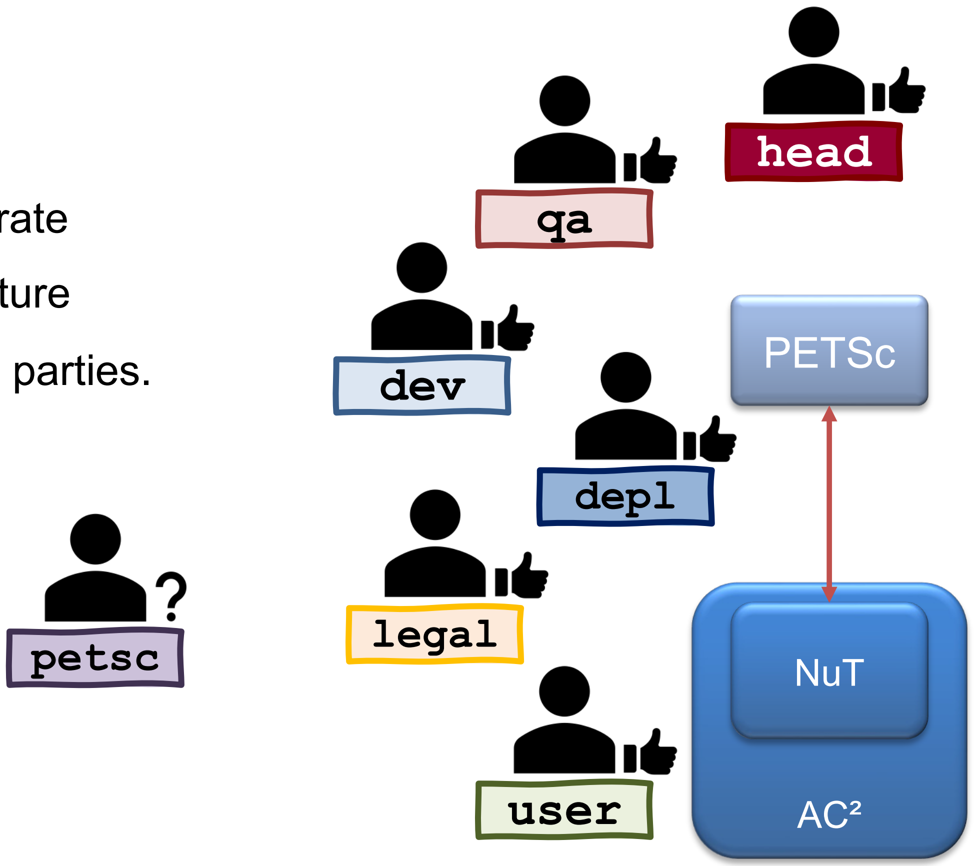
We presented an approach how to incorporate PETSc into a multiple applications architecture complying with the requirements of several parties.

Tools:

- GitLab and its CI feature
- CMake
- Docker

Additional free software:

- CMakeIt: gitlab.com/nordfox/cmakeit
- MMA: gitlab.com/nordfox/mma
- FDE: gitlab.com/Zorkator/libfde



Thank you for your attention!

Supported by:

Federal Ministry for the Environment, Nature Conservation, Nuclear Safety and Consumer Protection

based on a decision of the German Bundestag