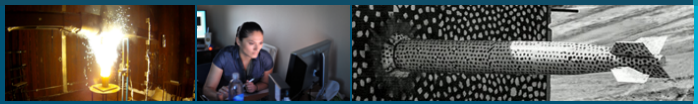
Sandia National Laboratories

# Adoption and Usage of Spack in ALEGRA DevOps and Development



*Presented by:*

Tim Fuller

November 1, 2023

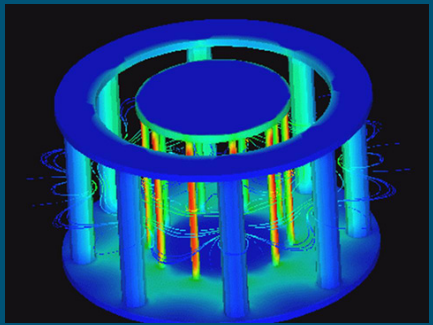
Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia LLC, a wholly owned subsidiary of Honeywell International Inc. for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525. SAND NO. 2023-11914C

# Alegra

## Summary

Alegra is a roughly 34-year old code that provides approximate solutions to multiphysics problems involving

- large-deformation Lagrangian, Eulerian, or ALE solid dynamics/hydrodynamics;
- electrical conductivity, magnetic induction/diffusion, nonlinear ohmic heating, Lorentz forces;
- finite element discretizations;
- material data and equations of state;
- radiation transport, thermonuclear burn; and
- piezo and ferro electric effects.



# Alegra

## Challenges

**Code base**

- roughly 34 year old “legacy code”
- large code base with C++, Fortran, C, and other language components
- extremely complex physics

**Dependencies**

- complex dependencies: roughly 30 TPLs including Dakota, Trilinos, Xyce
- each having its own build system
- some TPLs have proprietary licenses

**Data**

- relies on material data from a variety of sources
- ITAR, UCNI, LANL proprietary, and LLNL proprietary data
- not all customers are authorized to receive data

**Testing**

- most testing done on gifted, and aging, hardware
- thousands of tests with tens of Gb of data
- some tests take longer then 24 hours to execute

**Building**

- maintaining builds on all SNL CEE-LAN and HPC machines
- maintaining builds on select SNL test beds
- providing builds on customer machines for which there are no SNL counterparts

**Running**

- complex user interface
- interactions with many other tools: MPI, exodus, etc.

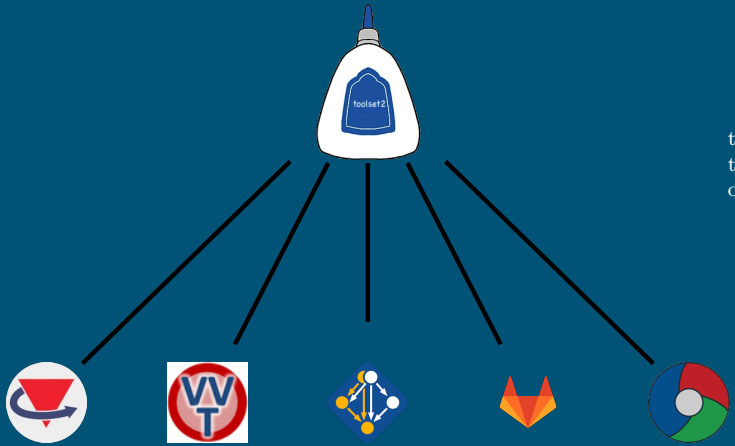
# Alegra

- manage and build TPLs;
- manage and build `alegranevada` source code;
- manage source code testing;
- manage source code releases; and
- define compiler interfaces and compiler flags.




The legacy toolset implements functionality from many modern tools

# Alegra tooling modernizations




toolset2 is a Python library that glues together the pieces of our CI/CD workflow:

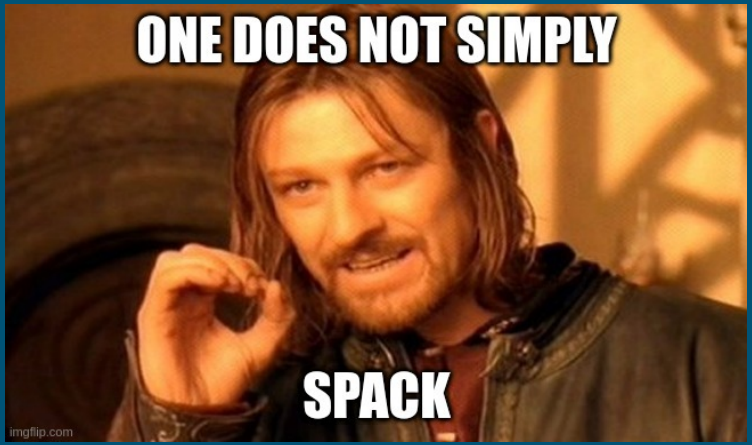
- Spack
- VVTest
- GitLab
- CDash

# It's...


https://medium.com/the-adventures-of-platoboy/it-is-not-turtles-all-the-way-down-4306ea599c6b

# Spack all the way down

# Adoption strategy, part 1



# Adoption strategy, part 1

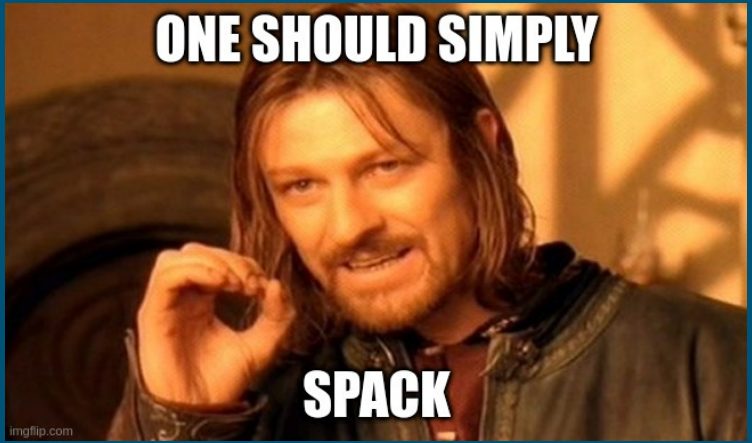
- Fork and wrap spack with our toolset, hide as many Spack details from developers

```
$ # setup environement
$ spacktivate
$ spack add ...
$ spack concretize...
$ spack install
```

```
$ nevada -E ENV install ...
```

- Provide default environments and reference area (`upstreams`)
- Provide spackages for every package in our software stack
- Modified Spack to fit our needs

# Adoption strategy, part 2




*For every application that uses Spack, there is a wrapper to wrap Spack.*

(Chris Siefert)

# Adoption strategy, part 2

- Don’t wrap, adapt
- Don’t wrap, extend
- Don’t wrap, contribute

# Don't wrap, adapt

Expect developers to read Spack documentation and learn basics of Spack

- Spack spec language
- `spack find`
- `spack info`
- `spack develop`
- `spack install`

provide 90% of the functionality we need for using Spack.

# Don't wrap, extend

Spack extensions allow one to extend Spack with custom commands.

- Originally, we provided additional functionality by stitching together different Spack commands with scripts
- These scripts were fragile and often broke when we updated Spack
- Spack provides a better solution in the form of extensions

```
spack:
  config:
    extensions:
    - $toolset2/var/spack/extensions/spack-nevada
```

```
$ ls $toolset2/var/spack/extensions/spack-nevada
nevada/
$ ls $toolset2/var/spack/extensions/spack-nevada/nevada
cmd/
$ ls $toolset2/var/spack/extensions/spack-nevada/nevada/cmd
distribution.py  make.py  multi_develop.py  prune_build_dirs.py  pull.py
```

# Don't wrap, extend: multi-develop

```
$ spack multi-develop -h
usage: spack multi-develop [-h] [-f FILE] ...

add multiple specs to an environment's dev-build information

options:
  -h, --help  show this help message and exit

Input format:
  details     colon separated list of details
  -f FILE     File containing develop specs

'spack multi-develop' is a wrapper around 'spack develop' that allows
adding multiple specs to an environment's dev-build information.
$ cat specs.yaml
develop:
- alegranevada@master
  path: $CWD/alegranevada
- trilinos@develop
  path: $CWD/trilinos
$ spack multi-develop -f specs.yaml
$ spack add alegranevada@master ^trilinos@develop+shared
$ spack concretize -f &> concretize.txt
$ spack install
```

# Don't wrap, extend: make

```
$ spack make -h
usage: spack make [-h] ...

make SPEC directly with ‘make‘ or ‘ninja‘

positional arguments:
  SPEC        Spack package to build (must be a develop spec)

options:
  -h, --help  show this help message and exit

Additional arguments can be sent to the build system directly by separating them
from SPEC by ’--’.  Eg, ‘spack make SPEC -- -j16‘
$ spack make trilinos -- -j32
```

# Don't wrap, extend: make

```python
import argparse
import os

import llnl.util.tty as tty
import spack.build_environment as build_environment
import spack.builder
import spack.cmd
import spack.paths

from llnl.util.filesystem import working_dir
from spack.util.executable import Executable

description = "make SPEC directly with `make` or `ninja`"
section = "nevada"
level = "short"

epilog = """\
Additional arguments can be sent to the build system directly by separating them
from SPEC by `--`.  Eg, `spack make SPEC -- -j16`
"""

def setup_parser(parser):
    parser.epilog = epilog
    parser.add_argument(
        "spec",
        metavar="SPEC",
        nargs=argparse.REMAINDER,
        help="Spack package to build (must be a develop spec)",
    )
```

```python
def make(parser, args):
    env = spack.cmd.require_active_env(cmd_name="make")
    try:
        sep_index = args.spec.index("--")
        extra_make_args = args.spec[sep_index + 1 :]
        specs = args.spec[:sep_index]
    except ValueError:
        extra_make_args = []
        specs = args.spec
    specs = spack.cmd.parse_specs(specs)
    if not specs:
        tty.die("You must supply a spec.")
    if len(specs) != 1:
        tty.die("Too many specs.  Supply only one.")
    spec = env.matching_spec(specs[0])
    if spec is None:
        tty.die(f"{specs[0]}: spec not found in environment")
    pkg = spec.package
    builder = spack.builder.create(pkg)
    if hasattr(builder, "build_directory"):
        build_directory = os.path.normpath(
            os.path.join(pkg.stage.path, builder.build_directory)
        )
    else:
        build_directory = pkg.stage.source_path
    build_environment.setup_package(spec.package, False, "build")
    with working_dir(build_directory):
        make_program = "ninja" if os.path.exists("build.ninja") else "make"
        make = Executable(make_program)
        make(*extra_make_args)
```

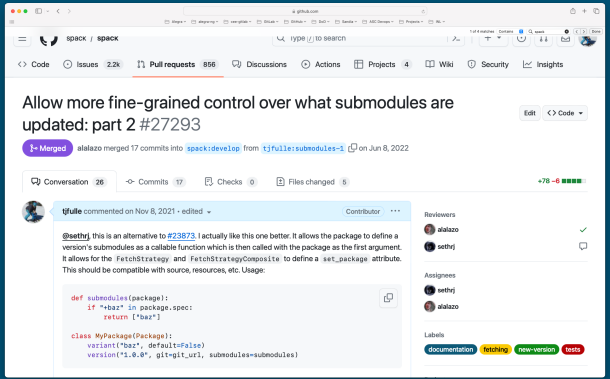
# Don't wrap, extend: distribution

An `alegranevada` distribution is an archive containing ALEGRA examples, tests, documentation, and source code. We use Spack to create and install the distributions:

```
$ spack distribution -h
usage: spack distribution [-h] SUBCOMMAND ...

Create and install alegranevada distributions

positional arguments:
  SUBCOMMAND
    create            Create the AlegraNevada distribution
    install           Install the AlegraNevada distribution
    add-compilers     Find compilers and add them to the AlegraNevada distribution

options:
  -h, --help          show this help message and exit
```

# Don't wrap, contribute

- Contribute changes you require back to Spack
- Spack developers are open to collaborations and helpful in getting modifications incorporated upstream



# Questions?