This paper describes objective technical results and analysis. Any subjective views or opinions that might be expressed in the paper do not necessarily represent the views of the U.S. Department of Energy or the United States Government.

SAND2023-05604C

Vertical Applications for Spack: What, Why, and How

Phil Sakievich (SNL)

psakiev@sandia.gov
@psakievich (slack/github)

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & 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.

06/29/2023

https://github.com/sandialabs/spack-manager

EXASCALE COMPUTING PROJECT

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & 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.

Overview

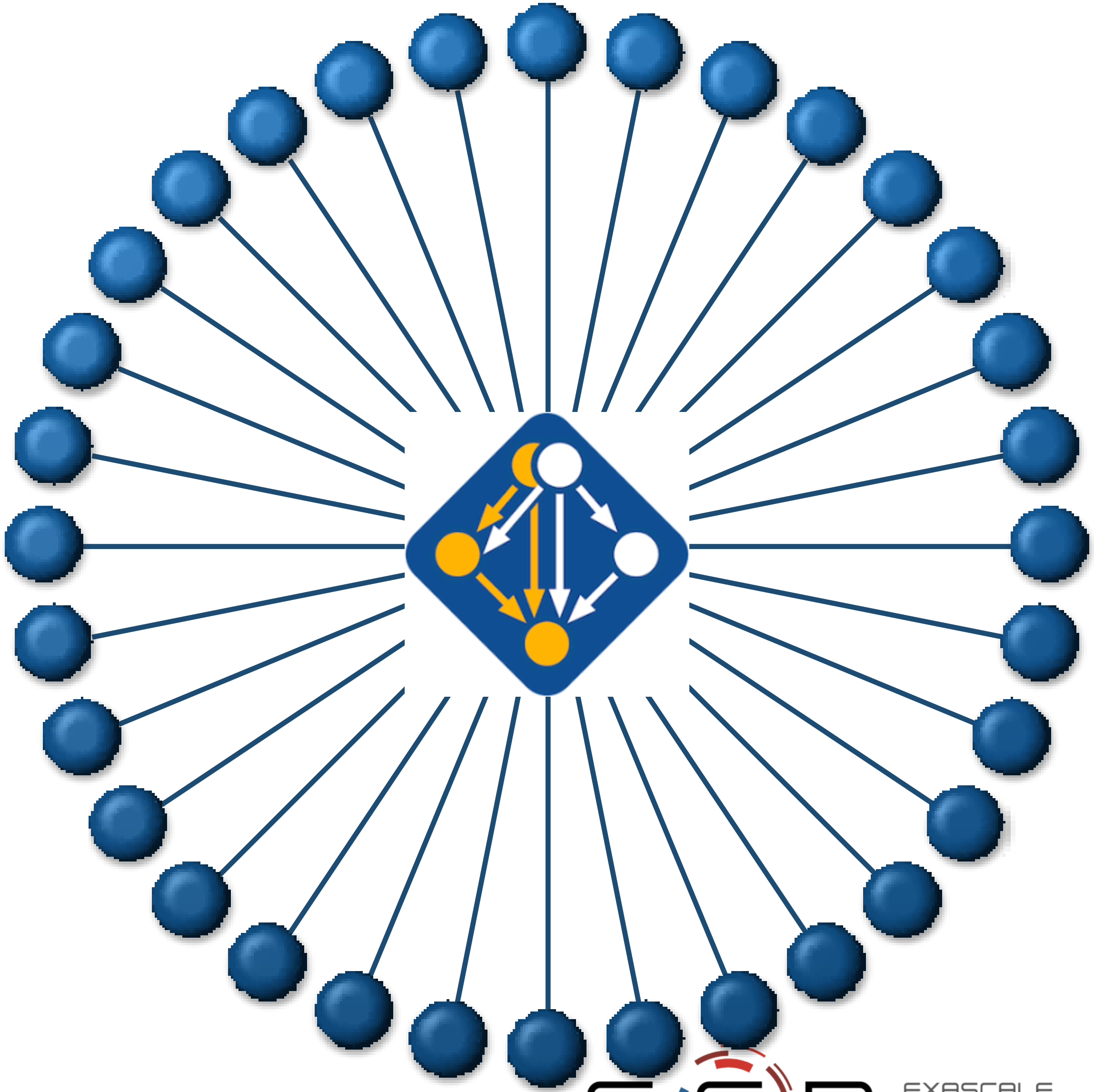
- What is Spack?
- What is a Vertical Application?
- Where does Spack fit in?
- Software Democratization
- What would this look like for Spack
- Spack-Manager: Case study for vertical application
- Summary and Discussion

ECP: Funding Statement

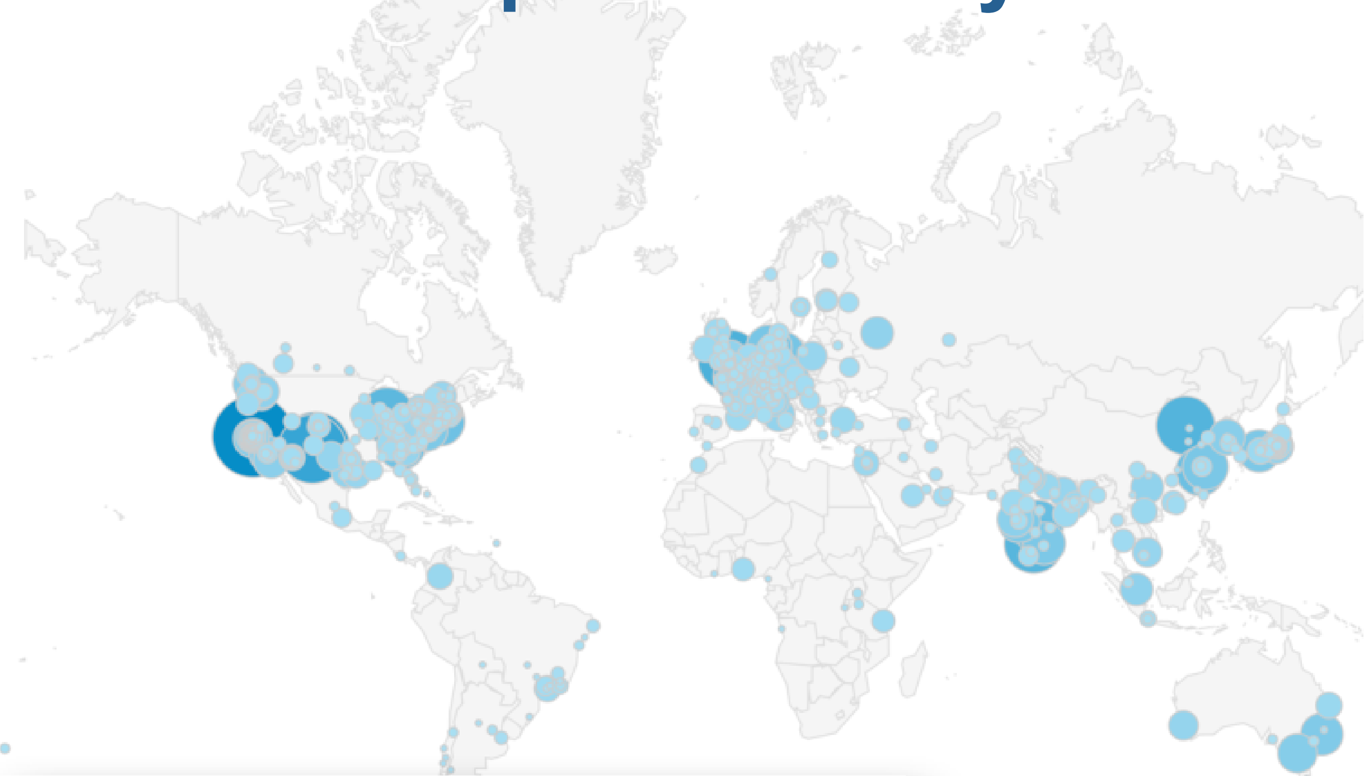
This research was supported by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of two U.S. Department of Energy organizations (Office of Science and the National Nuclear Security Administration) responsible for the planning and preparation of a capable exascale ecosystem, including software, applications, hardware, advanced system engineering, and early testbed platforms, in support of the nation's exascale computing imperative.

Spack: Package Manager++

- Spack at its core is a package manager
- Spack has many attractive features:
  - Simplea, expressive syntax for software configurations
  - Non-destructive installs
  - Embedded tribal HPC knowledge
  - Containerization automation
  - CI pipeline construction
  - Module creation
  - Binary caching
  - Massively parallel build processes
  - A unique, scalable, multicomponent development tool (spack develop)
  - ….



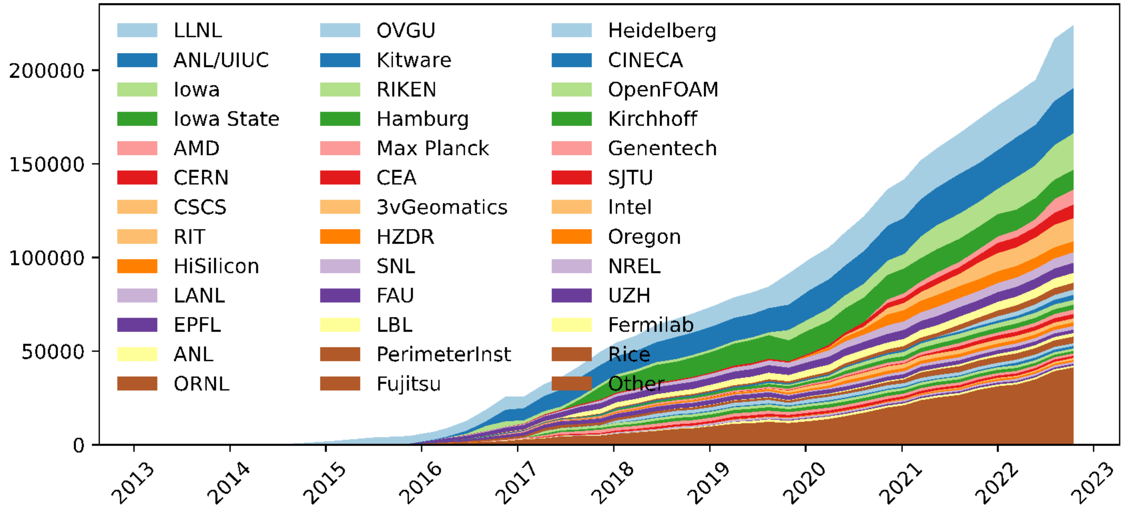
Spack sustains the HPC software ecosystem with the help of its many contributors



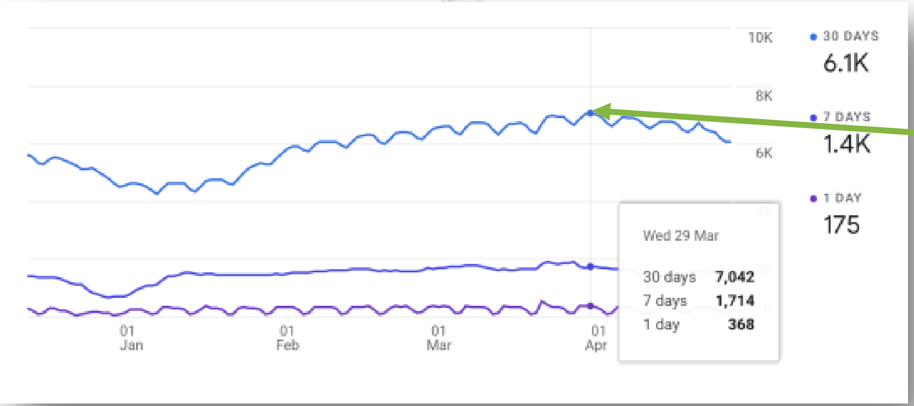
Over 7,100 software packages

Over 1,100 contributors

Contributions (lines of code) over time in packages, by organization



Most package contributions are ***not*** from DOE
But they help sustain the DOE ecosystem!



Over 7,000 monthly active users
(per documentation site)

Slide provided by Todd Gamblin

EXASCALE COMPUTING PROJECT

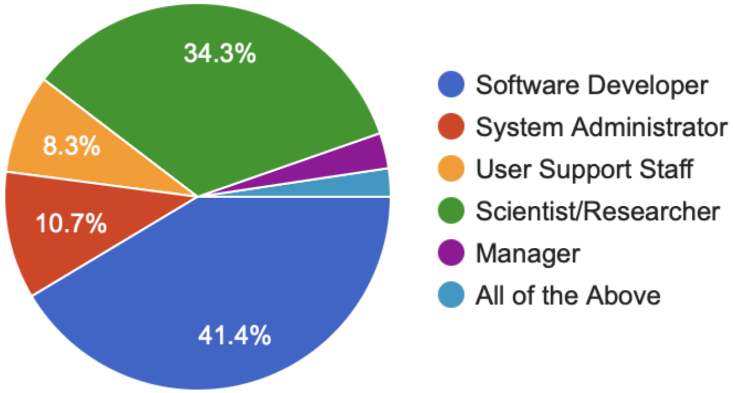
Spack users over the years

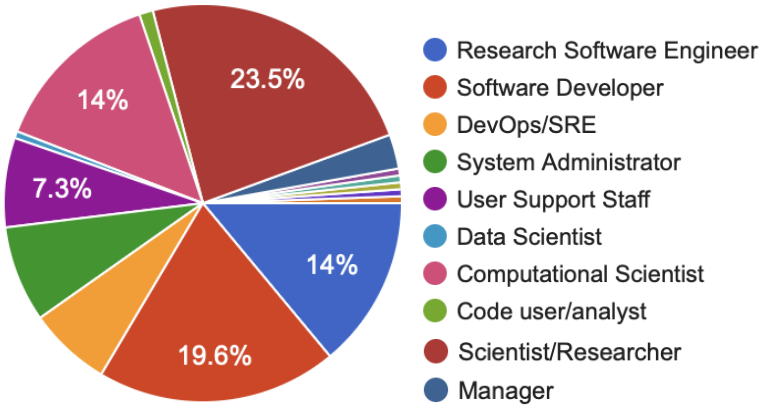
2020

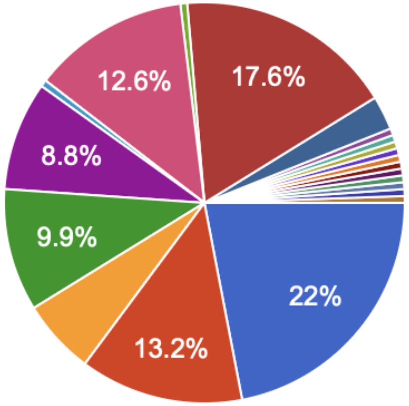
2021

2022

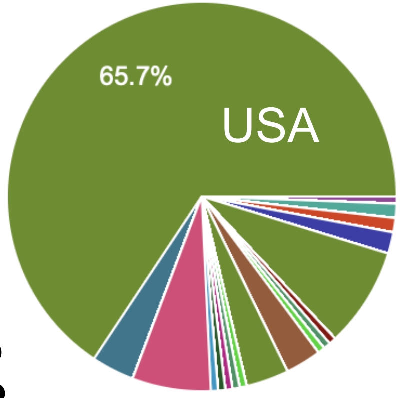
What type of user are you?



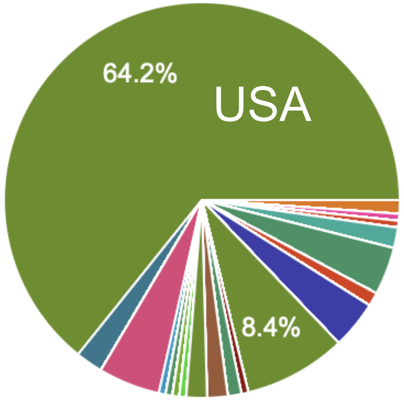




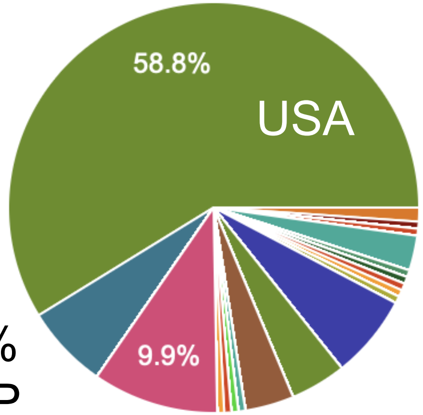
What country are you in?



36% ECP



37% ECP



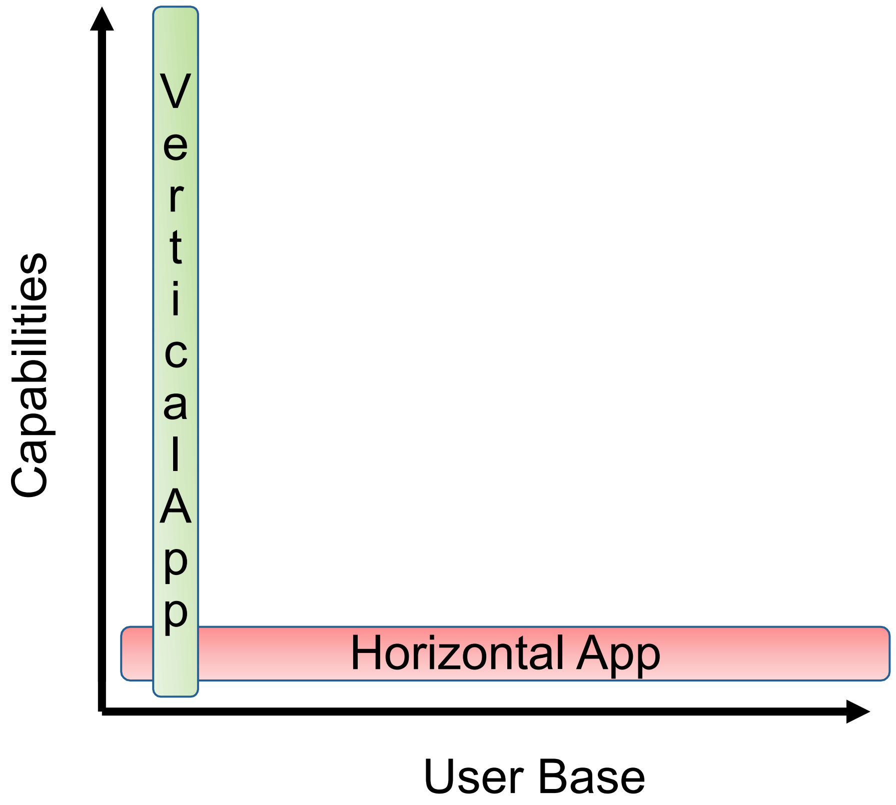
26% ECP

Slide provided by Todd Gamblin

What is a Vertical Application?

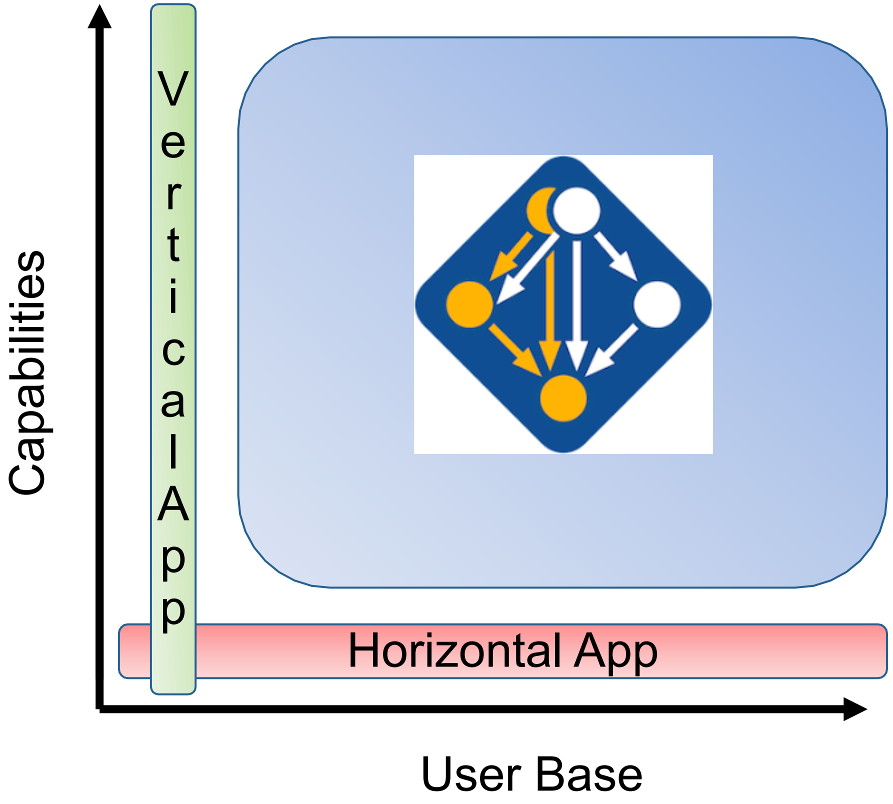
- *A **vertical application** is any software application that supports a specific business process and targets a smaller number of users with specific skill sets and job responsibilities within an organization.*

https://www.webopedia.com/definitions/vertical-application/



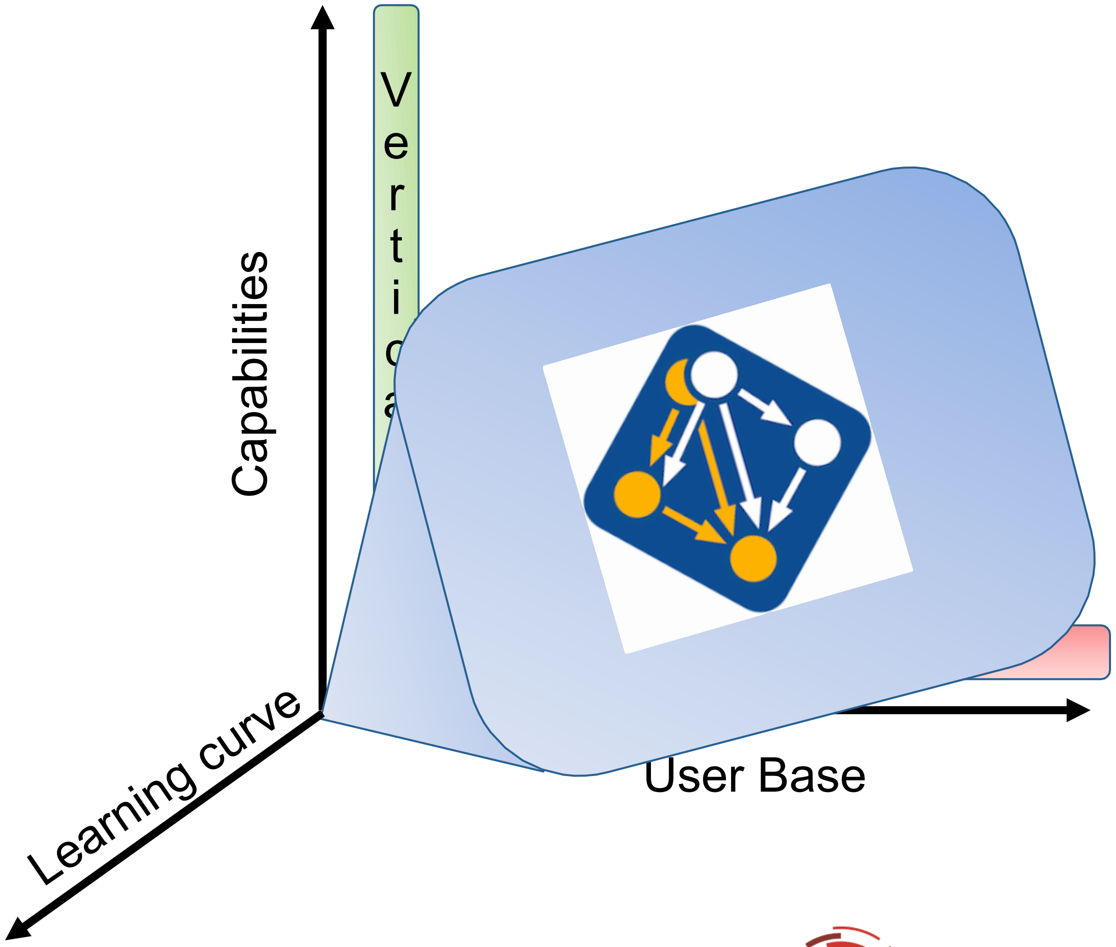
So where does Spack fit in?

- User base is growing, and so are features…



So where does Spack fit in?

- User base is growing, and so are features…
- Natural consequence is the learning curve grows
- Different user types can lead to friction
- Main strategies I've seen
  A. Silo/hide Spack
  B. Tell users to embrace Spack fully



Democratization of Spack

- ***Democratization of technology*** *refers to the process by which access to technology rapidly continues to become more accessible to more people. New technologies and improved user experiences have empowered those outside of the technical industry to access and use technological products and services* – **Wikipedia**

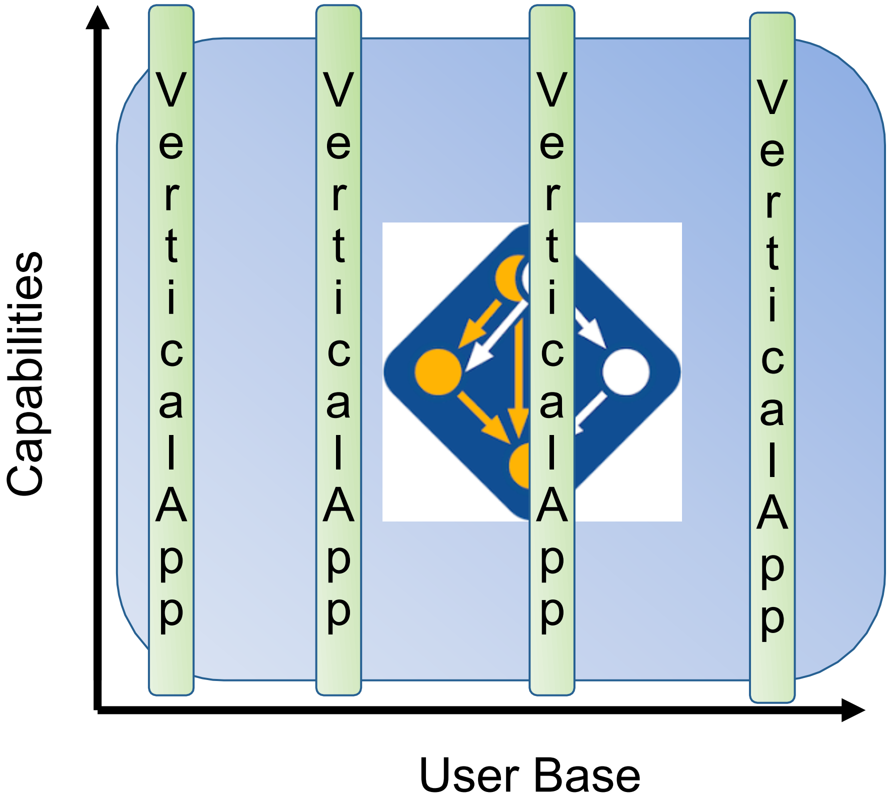
Rest of the talk …

- What would democratization of Spack look like?
- Is that a good idea?

https://en.wikipedia.org/wiki/Democratization_of_technology

Consider Spack as a Platform

- User base is growing, and so are features. We don't really want to stop either…
  - Creating vertical applications that use Spack as a platform
  - Expose the most important parts to further education, impact, and comfort without overwhelming
- *"Everything should be made as simple as possible but not simpler"* – Albert Einstein

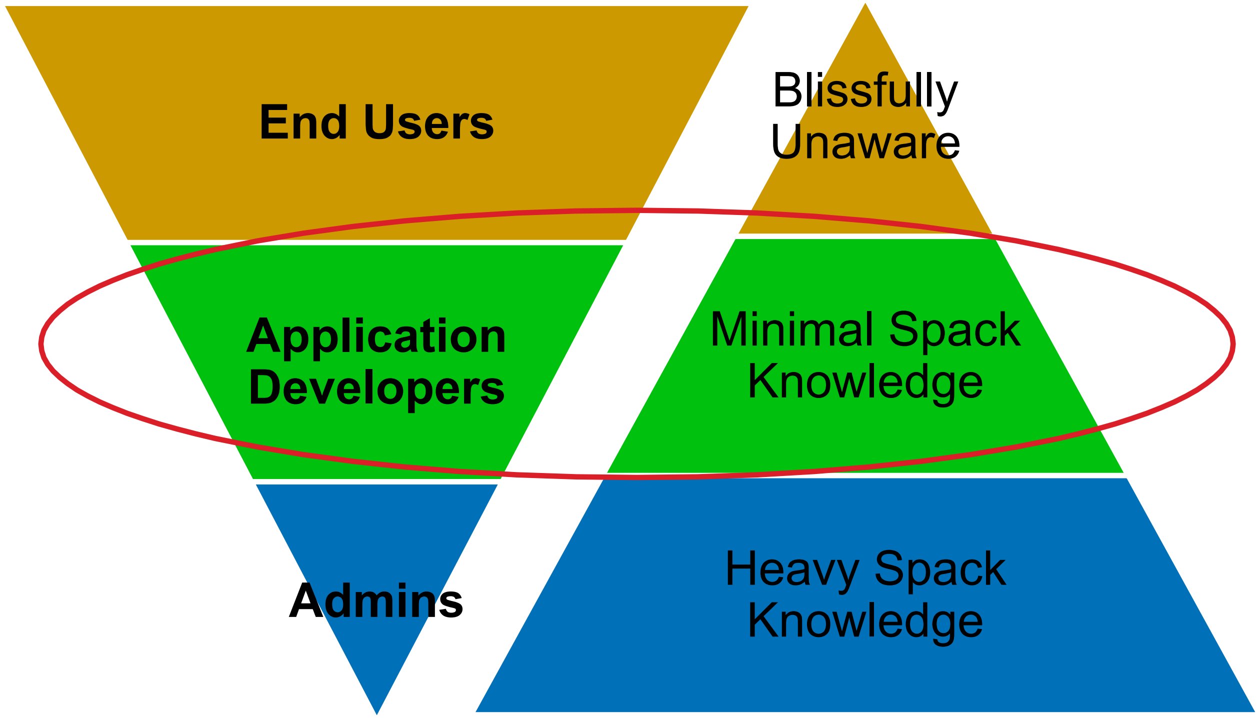


Attributes of an Ideal Vertical Application

- Minmax problem: min(user inputs) max(user impact)
- Intuitive
- Easy to update and maintain
- Provides feedback and features that help steer the direction of Spack
- Gateway to broader Spack usage

Spack-Manager: Example of a Vertical Spack Application

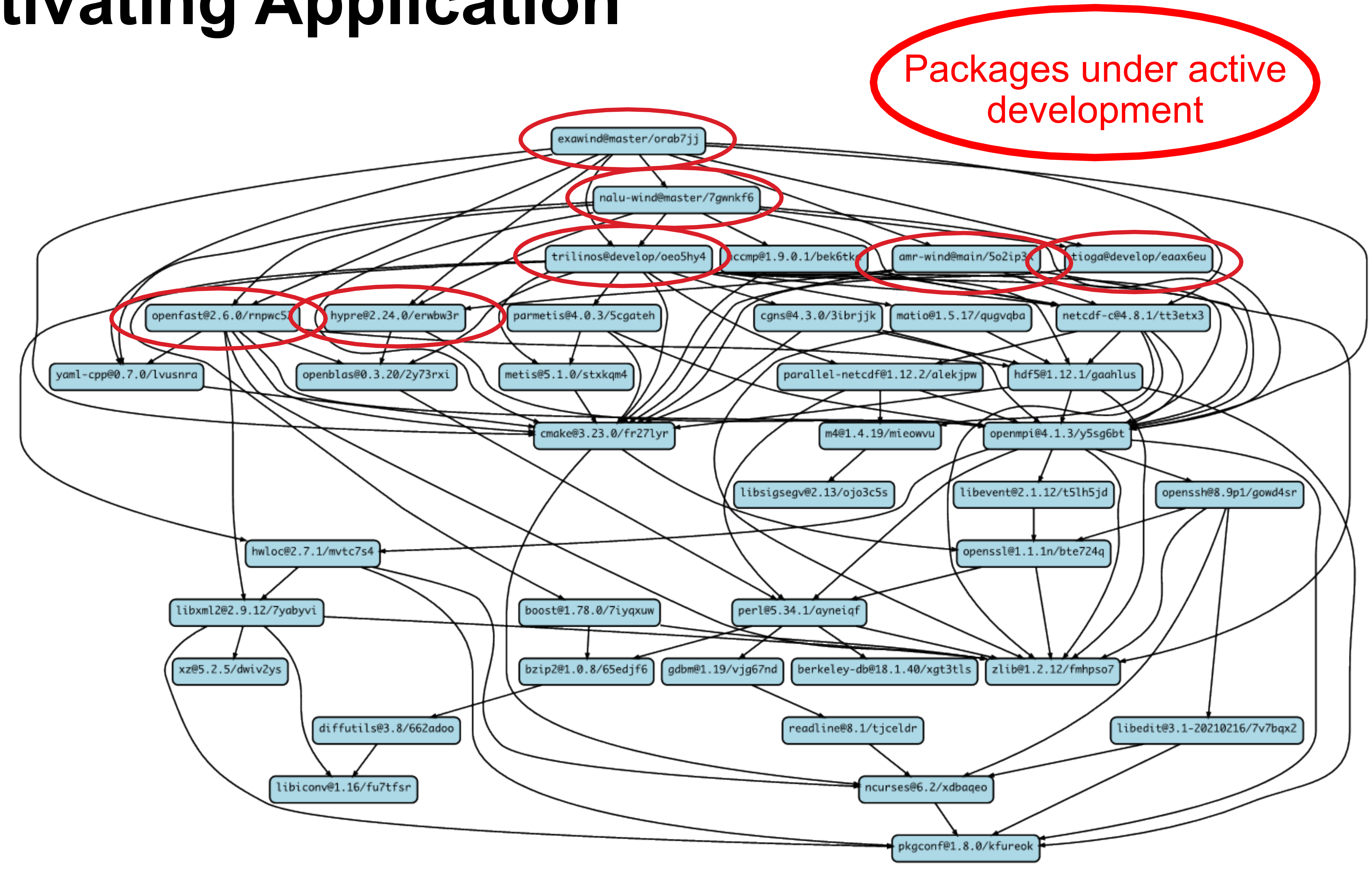
- Spack-Manager is an **extension** to Spack that aims to act as a buffer between Spack and our end application
  - Increases our agility
  - Framework to prototype new Spack features
  - Manage machine specific configurations and create a machine agnostic interface
- This talk will focus on accelerating application developer workflows



Philosophy: Population size of the user profile should have an inverse relationship with required understanding of Spack

ExaWind: The Motivating Application

- ExaWind software stack:
  - Combine two loosely coupled CFD codes with entirely different software stacks (Trilinos and AMReX)
  - Living on the develop branch of multiple dependencies
  - Project is actively supporting development of 7+ software packages in the stack (CPU+GPU)
  - Many, many machines: Laptops, Azure, Eagle, Summit, Crusher/Frontier, Sunspot, etc.
- Developer Configuration Challenges:
  - Building
  - Developing
  - Testing
  - Deploying
  - 😭 😭 😭



Packages under active development

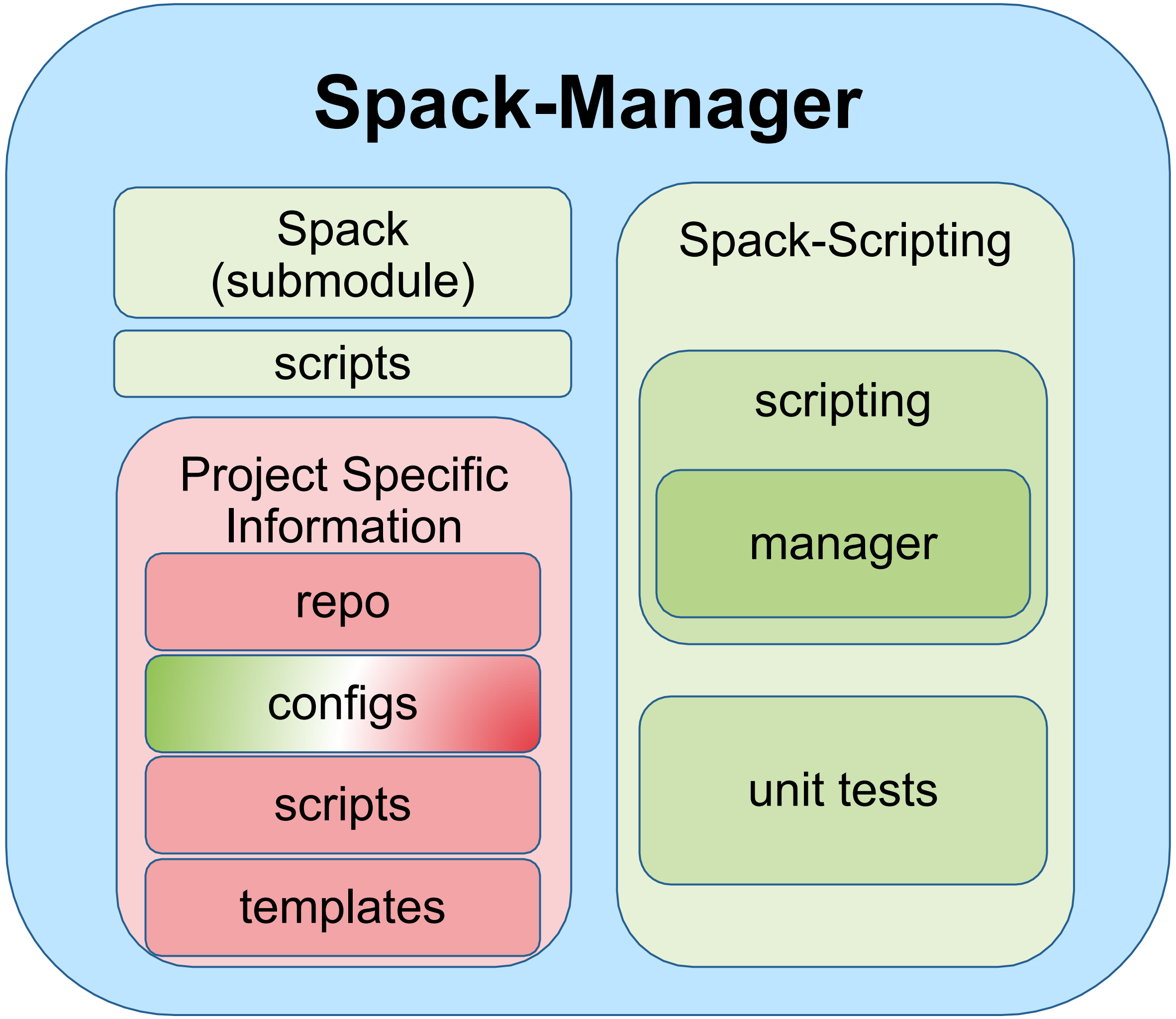
Core Spack Feature: Spack Develop

- In a Spack environment *develop specs* can be added
- Develop specs work as follows
  - If DAG_spec.satisfies(develop_spec)
    - Do a build from the users source code rather than from Spack’s staging procedure
    - Perform incremental builds based on timestamp of files in the source directory
- Allows for arbitrary development of packages in the graph
  - Dependencies will get automatically rebuilt
- Allows for multiple builds from the same source
  - Cuda and Non-Cuda builds from the same source code at the same time
  - Graph level parallelism is available in builds

```
# In this configuration you will get
# 4 develop builds: cuda and non-cuda
# nalu-wind and trilinos coming from
# the same sources
spack:
  specs:
  - nalu-wind +cuda cuda_arch=70
  - nalu-wind ~cuda
  view: false
  concretizer:
    unify: false
  develop:
    nalu-wind:
      spec: nalu-wind@master
    trilinos:
      spec: trilinos@develop
```

Spack-Manager Infrastructure

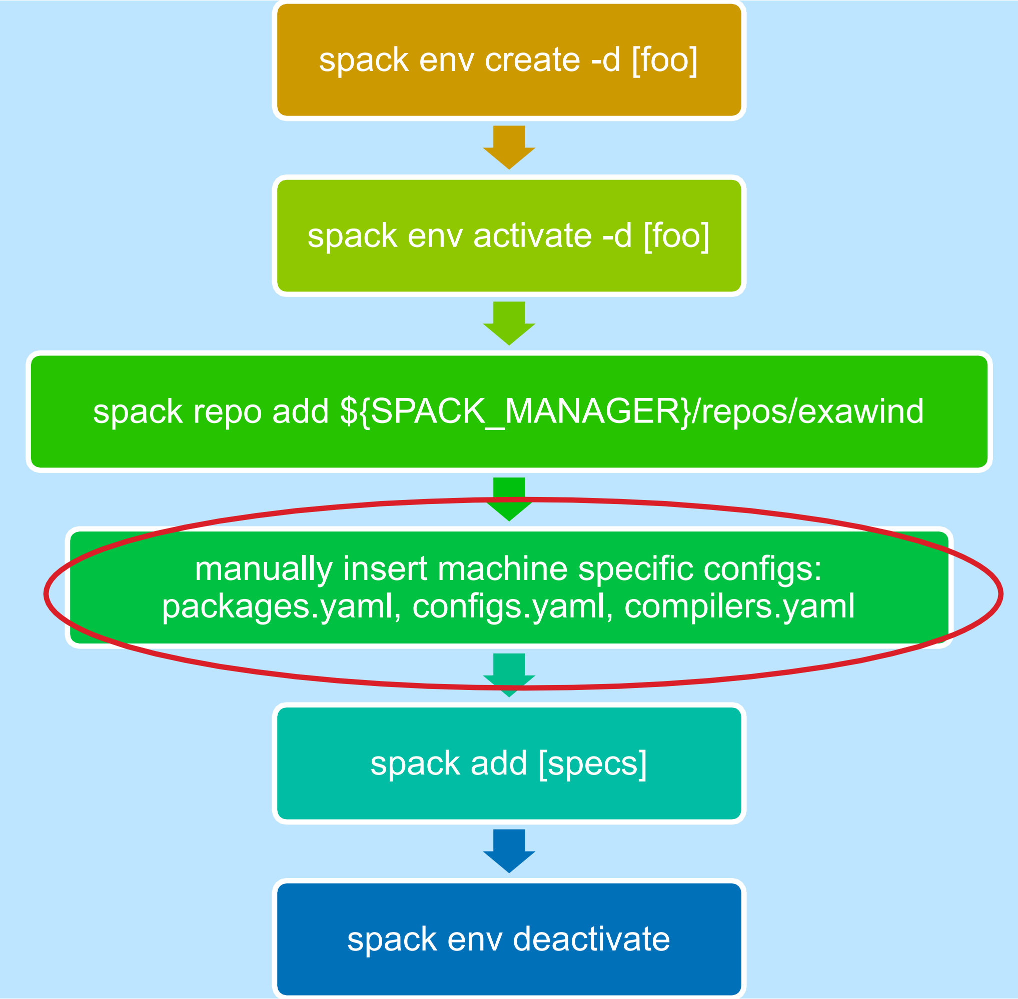
- 1k-2k lines of python code (including prototypes)
- Core features for machine agnostic environment curation is closer to 500 700 lines
- Utilize Spack code through Spack extensions
- Also get the testing and formatting infrastructure
- Relatively simple to maintain



Machine Agnostic API: How do we do this?

- Utilize Spack API's to write Spack extensions
  - Environment curation
  - All of our scripts serve to reduce the end user API
  - Can be replicated through core Spack commands and a little manual intervention
- A core example of this is:
  - find-machine + create-env
    - find-machine: a utility that allows custom python scripts to identify the current machine
    - create-env: uses find-machine and stored configs to automate platform specific environments

spack manager create-env –d [foo] –s [specs]



spack env create -d [foo]

spack env activate -d [foo]

spack repo add ${SPACK_MANAGER}/repos/exawind

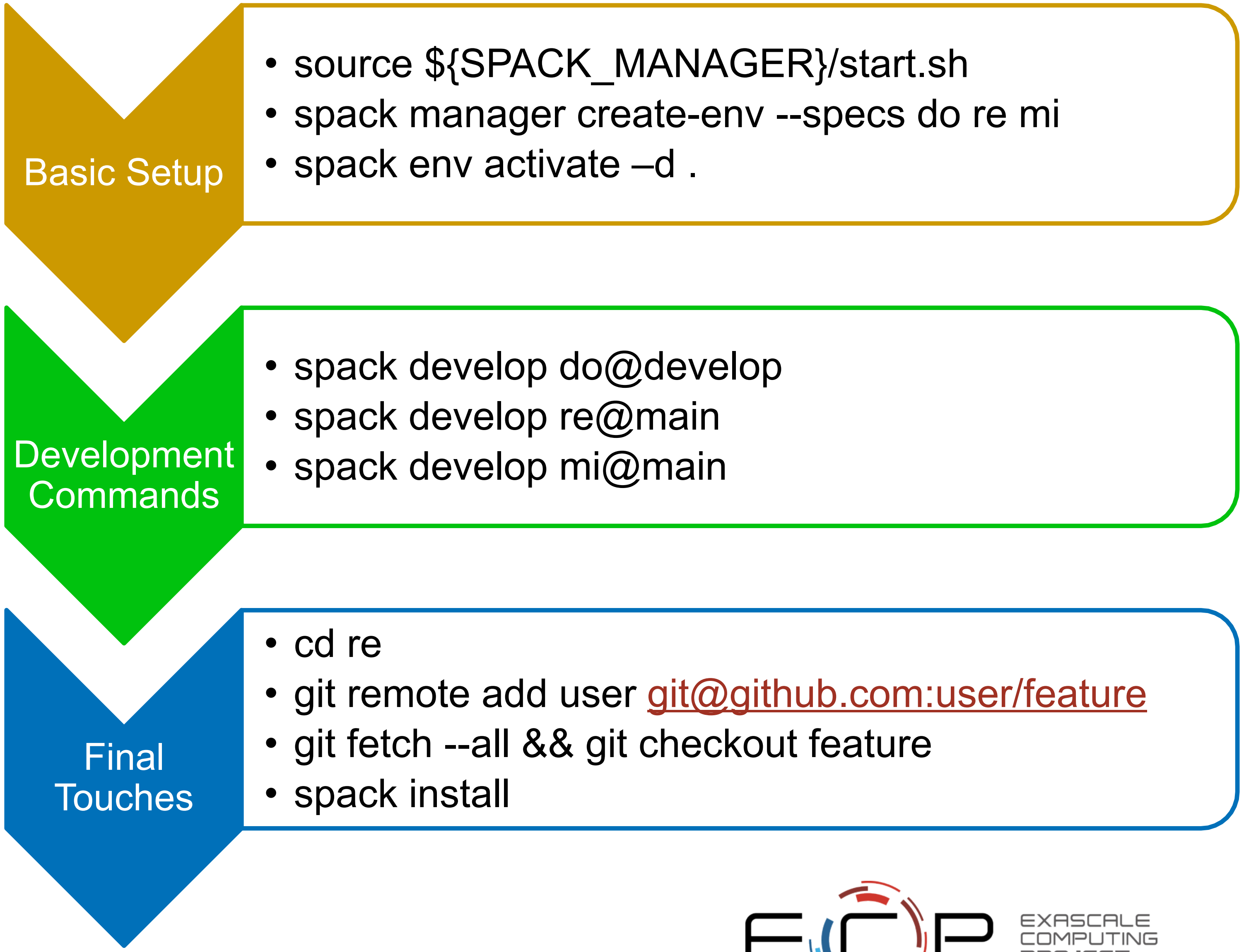
manually insert machine specific configs:
packages.yaml, configs.yaml, compilers.yaml

spack add [specs]

spack env deactivate

Setting Up a Development Environment

- *spack develop* is amazingly powerful but …
- Setting up a development environment can still be a lot of work
- Can start to feel tedious when done often
- Number of commands can be reduced with some basic assumptions



Basic Setup

- source ${SPACK_MANAGER}/start.sh
- spack manager create-env --specs do re mi
- spack env activate –d .

Development Commands

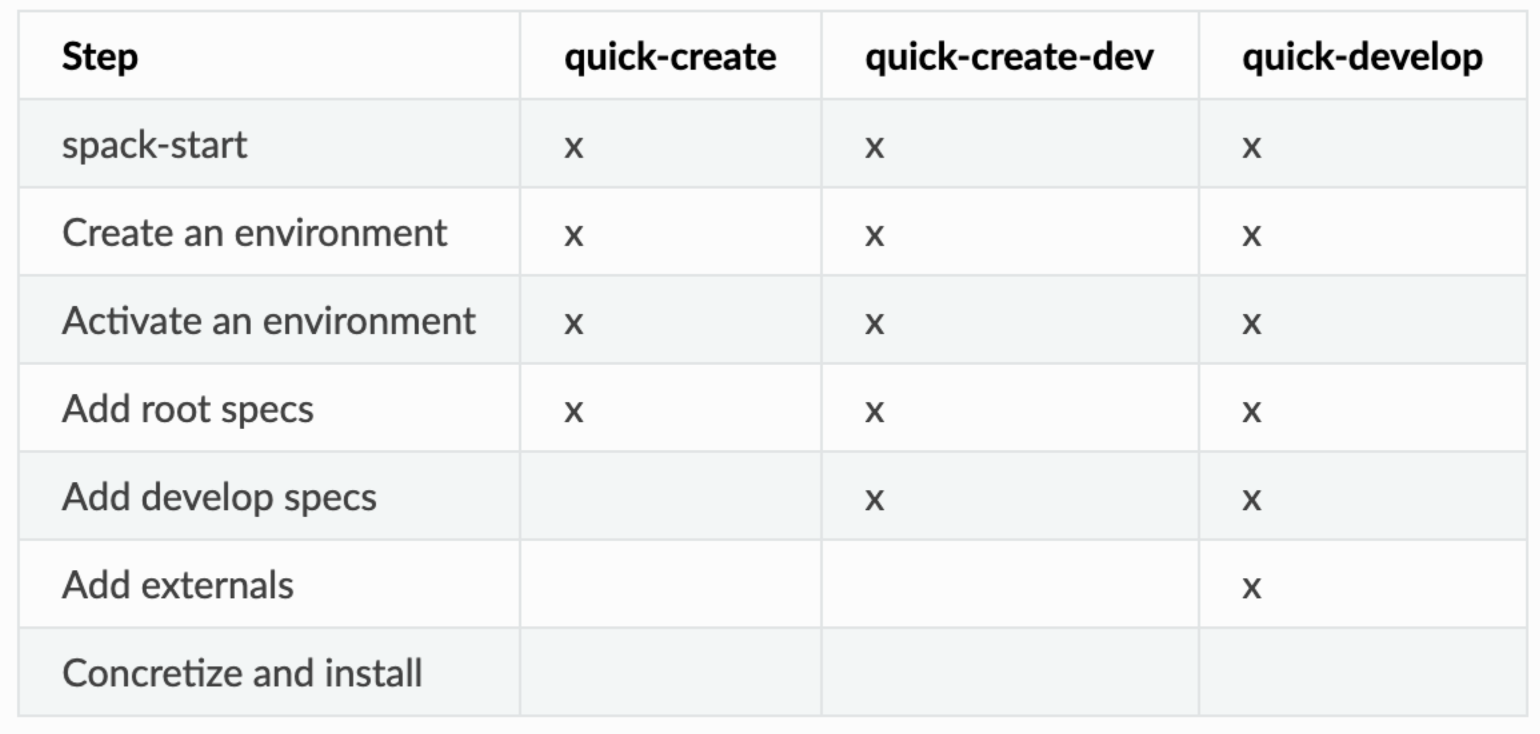
- spack develop do@develop
- spack develop re@main
- spack develop mi@main

Final Touches

- cd re
- git remote add user git@github.com:user/feature
- git fetch --all && git checkout feature
- spack install

Bash ”quick-commands”

- Wrap the functionality of basic setup and development commands together
- Common features:
  - Shell source Spack/Spack-Manager
  - Create an anonymous Spack environment
  - Activate the created environment
- Development specific assumptions:
  - All concrete spec’s are intended as develop specs ([name]@[version])
  - Anything not pre-cloned should be fetched via spack develop

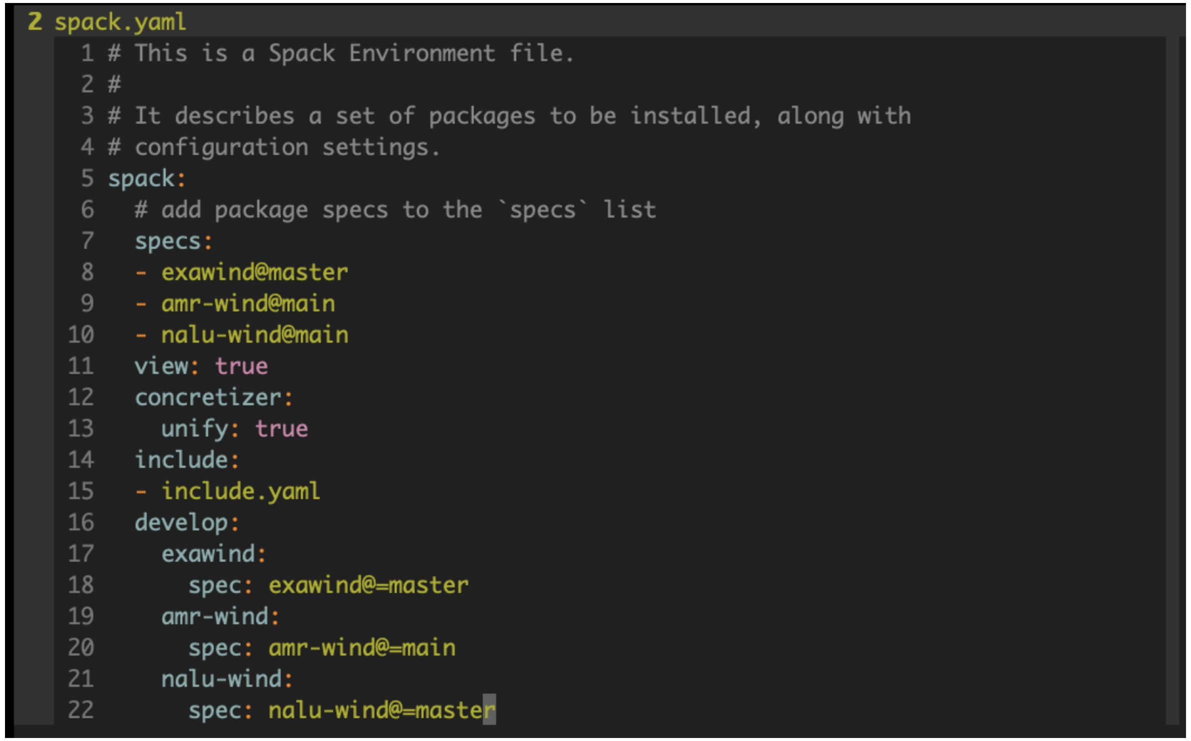


Step	quick-create	quick-create-dev	quick-develop
spack-start	x	x	x
Create an environment	x	x	x
Activate an environment	x	x	x
Add root specs	x	x	x
Add develop specs		x	x
Add externals			x
Concretize and install			

- quick-create-dev --spec do@develop re@main mi@main

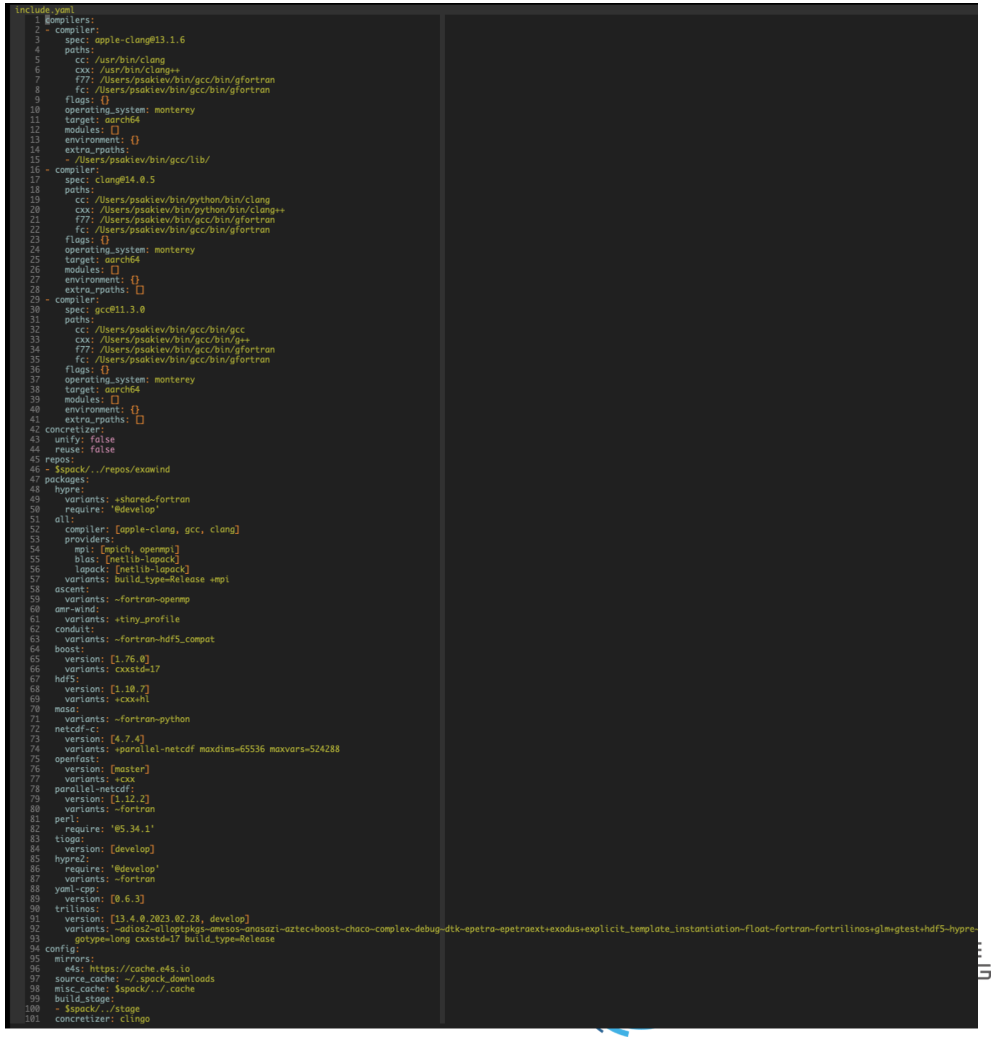
What does it look like?

quick-create-dev --spec exawind@master amr-wind@main nalu-wind@master



```yaml
# This is a Spack Environment file.
#
# It describes a set of packages to be installed, along with
# configuration settings.
spack:
  # add package specs to the `specs` list
  specs:
  - exawind@master
  - amr-wind@main
  - nalu-wind@main
  view: true
  concretizer:
    unify: true
  include:
  - include.yaml
  develop:
    exawind:
      spec: exawind@=master
    amr-wind:
      spec: amr-wind@=main
    nalu-wind:
      spec: nalu-wind@=master
```

What user needs to care about



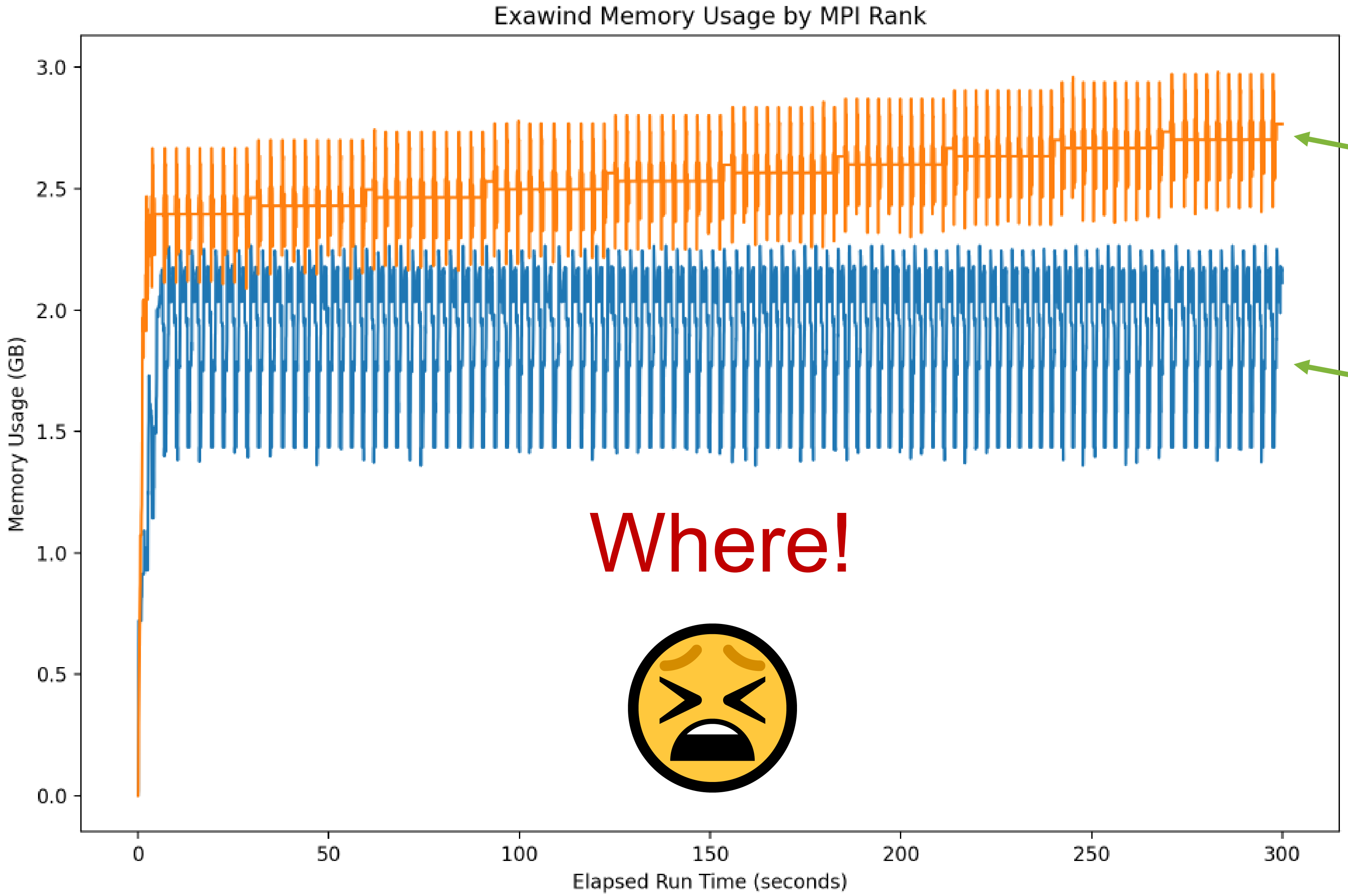
```yaml
compilers:
- compiler:
    spec: apple-clang@13.1.6
    paths:
      cc: /usr/bin/clang
      cxx: /usr/bin/clang++
      f77: /Users/psakiev/bin/gcc/bin/gfortran
      fc: /Users/psakiev/bin/gcc/bin/gfortran
    flags: {}
    operating_system: monterey
    target: aarch64
    modules: []
    environment: {}
    extra_rpaths:
    - /Users/psakiev/bin/gcc/lib/
- compiler:
    spec: clang@14.0.5
    paths:
      cc: /Users/psakiev/bin/python/bin/clang
      cxx: /Users/psakiev/bin/python/bin/clang++
      f77: /Users/psakiev/bin/gcc/bin/gfortran
      fc: /Users/psakiev/bin/gcc/bin/gfortran
    flags: {}
    operating_system: monterey
    target: aarch64
    modules: []
    environment: {}
    extra_rpaths: []
- compiler:
    spec: gcc@11.3.0
    paths:
      cc: /Users/psakiev/bin/gcc/bin/gcc
      cxx: /Users/psakiev/bin/gcc/bin/g++
      f77: /Users/psakiev/bin/gcc/bin/gfortran
      fc: /Users/psakiev/bin/gcc/bin/gfortran
    flags: {}
    operating_system: monterey
    target: aarch64
    modules: []
    environment: {}
    extra_rpaths: []
concretizer:
  unify: false
  reuse: false
repos:
- $spack/../repos/exawind
packages:
  hypre:
    variants: +shared~fortran
    require: '@develop'
  all:
    compiler: [apple-clang, gcc, clang]
    providers:
      mpi: [mpich, openmpi]
      blas: [netlib-lapack]
      lapack: [netlib-lapack]
    variants: build_type=Release +mpi
  ascent:
    variants: ~fortran~openmp
  amr-wind:
    variants: +tiny_profile
  conduit:
    variants: ~fortran~hdf5_compat
  boost:
    version: [1.76.0]
    variants: cxxstd=17
  hdf5:
    version: [1.10.7]
    variants: +cxx+hl
  masa:
    variants: ~fortran~python
  netcdf-c:
    version: [4.7.4]
    variants: +parallel-netcdf maxdims=65536 maxvars=524288
  openfast:
    version: [master]
    variants: +cxx
  parallel-netcdf:
    version: [1.12.2]
    variants: ~fortran
  perl:
    require: '@5.34.1'
  tioga:
    version: [develop]
  hypre2:
    require: '@develop'
    variants: ~fortran
  yaml-cpp:
    version: [0.6.3]
  trilinos:
    version: [13.4.0.2023.02.28, develop]
    variants: ~adios2~alloptpkgs~amesos~anasazi~aztec+boost~chaco~complex~debug~dtk~epetra~epetraext+exodus+explicit_template_instantiation~float~fortran~fortrilinos+glm+gtest+hdf5~hypre~
      gotype=long cxxstd=17 build_type=Release
config:
  mirrors:
    e4s: https://cache.e4s.io
  source_cache: ~/.spack_downloads
  misc_cache: $spack/../.cache
  build_stage:
  - $spack/../stage
  concretizer: clingo
```

Machine/project specific boiler plate abstracted away

Build and Test Process: Debug Issue from Trilinos Update

```
1)  quick-create-dev -s nalu-wind@master+cuda build_type=Debug trilinos@develop build_type=Debug
2)  spack install
3)  spack cd -b nalu-wind
4)  # run test to confirm behavior
5)  # modify trillinos source code
6)  spack install
7)  # run test to see if behavior is fixed
8)  # iterate further
```

Example: Uh-Oh a Memory Leak!



Where!

Single Nalu-Wind rank

Single AMR-Wind rank

- Long-standing memory leak not many developers had time to investigate
- Lots of overhead to set up tools for such a large stack

(Plot generated merely by sampling ps)

Spack-Manager: Debugging a Magnitude Faster

```
quick-develop -s exawind+asan build_type=Debug \
  ^amr-wind+asan build_type=Debug \
  ^nalu-wind+asan build_type=Debug \
  ^tioga+asan build_type=Debug \
  ^trilinos+asan build_type=Debug %clang@10.0.0
spack install
spack cd -b exawind
spack build-env exawind ctest -j $(nproc) --output-on-failure
```

Set up a development environment and turn address sanitizer on for *everything*

Run regression tests

```
SUMMARY: AddressSanitizer: 791 byte(s) leaked in 17 allocation(s).
```

Wait…got some output!

```
spack-manager/environments/exawind/tioga/src/tioga_nogpu.h:24
spack-manager/environments/exawind/tioga/src/tioga_nogpu.h:36
spack-manager/environments/exawind/tioga/src/CartGrid.C:218
spack-manager/environments/exawind/tioga/src/CartGrid.C:239
spack-manager/environments/exawind/tioga/src/CartGrid.C:130
spack-manager/environments/exawind/tioga/src/tioga.C:1081
spack-manager/environments/exawind/exawind/src/OversetSimulation.cpp:90
spack-manager/environments/exawind/exawind/src/OversetSimulation.cpp:69
spack-manager/environments/exawind/exawind/app/exawind.cpp:173
```

Look at the ASAN backtrace

It's in TIOGA (which only executes in the Nalu-Wind ranks)

```
vim spack-manager/environments/exawind/tioga/src/tioga_nogpu.h
spack install
spack cd -b exawind
spack build-env exawind ctest -j $(nproc) --output-on-failure
```

Edit TIOGA code (or any code in the stack) and *easily* get updated results

Builds tioga, relinks nalu-wind, and relinks exawind

Onboarding Developers

- Ask developers to learn 3 things about Spack:
  - How to query the API for help, i.e. --help and spack info
  - How to read and write a Spack spec
  - What the major steps in the Spack build process are
- Learn to speak the basics of the language

For ExaWind development, onboarding new developers and development using Spack-Manager in general has made us an order of magnitude more efficient.
- Jon Rood (NREL)

[…]Spack-manager allows us to easily manage and control a complex build chain for a number of different hardware targets. To me, ExaWind would be much more difficult to configure and build for emerging platforms without this critical build package.
- Paul Mullowney (AMD)

I have to type a whole 12 characters to compile just 2 different codes with a zillion dependencies to debug my code
- Ganesh Vijayakumar (NREL)

Spack Manager and Spack have saved me an incredible amount of time and headache[…].
- Nate deVelder (SNL)

Spack-Manager Summary

- Overall considered a universal win on ExaWind
- Large work reduction in environment curation, after that it is boilerplate Spack
  - Allows developers to work on arbitrarily complex software stacks through the simple syntax of Spack specs
  - Mixed results WRT developers digging further into Spack
- The principles and base code are project agnostic, ExaWind specifics are in the configuration settings
- Extending to more projects is possible with some adjustments
- Needs of Spack-Manager have influenced mainline Spack and features have been upstreamed

Discussions to be Had

- Adding vertical applications creates a buffer between users day to day needs and Spack feature development
  - “Spack is great but it is missing x, y, z”
  - “Spack needs to stabilize more before we can use it"
- Should something like this exist for each app or should we rally around a few more common implementations?
  - Absolve all ideas into into Spack vs adding infrastructure to make it easier to spin up new vertical applications quickly
- Adding vertical applications can turn Spack into a plugin ecosystem.
  - Is that good or bad?
  - Does Spack want to go there?
  - Do users want to go there?
- What users and use cases would benefit most from vertical applications?

Feel free to reach out: psakiev@sandia.gov
@psakievich (github/slack)