Spack

A package manager for HPC systems

Peter Larsson - PDC

Spack - Spack

https://spack.io

Spack

About Blog Events Services Spack CI Status

Spack

A flexible package manager supporting multiple versions, configurations, platforms, and compilers.

GitHub | @spackpm | @spack | Slack | Docs | Discussion

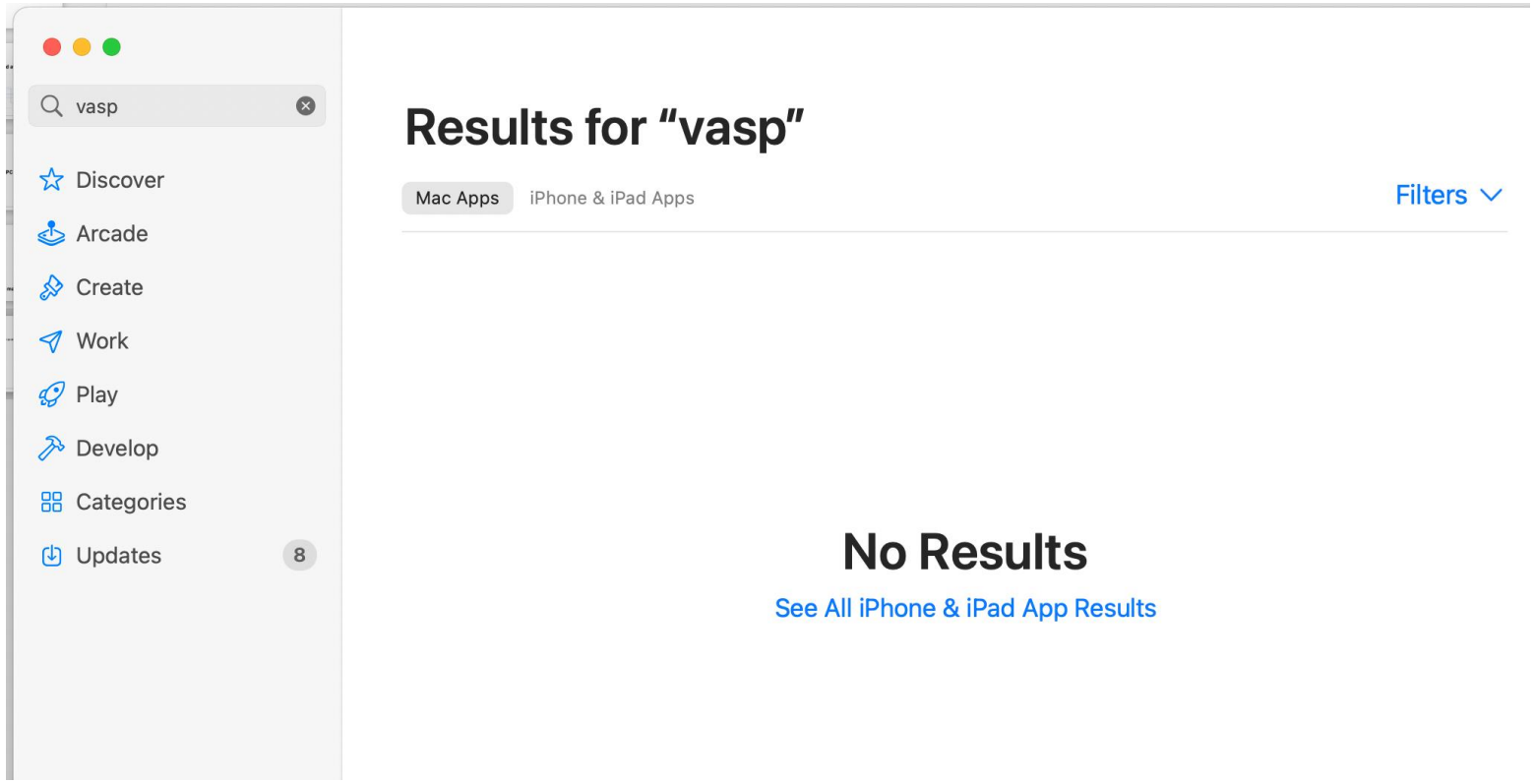
Photo: LLNL Sierra

Welcome to Spack!

Spack is a package manager for supercomputers, Linux, and macOS. It makes installing scientific software easy. Spack isn't tied to a particular language; you can build a software stack in Python or R, link to libraries written in C, C++, or Fortran, and easily swap compilers or target specific microarchitectures. Learn more here.

What if we had an App Store for the HPC cluster?

App Store.app



(VASP is used to simulate materials at the atomic level.
It is probably the most popular HPC application?)

Phone: App store

Consider the modern way of installing software on a new phone:

- You log in to the app store...
- your previously installed software from the other phone can/will be downloaded automatically...
- ...and **optimized for the new hardware automatically** (it might be a completely new CPU/GPU etc)
- All Apps are sandboxed and independent of each other.
- New Apps can be added by searching for their name, and just click install.

HPC cluster: Spack

Spack is the closest thing to an "App store" model for HPC:

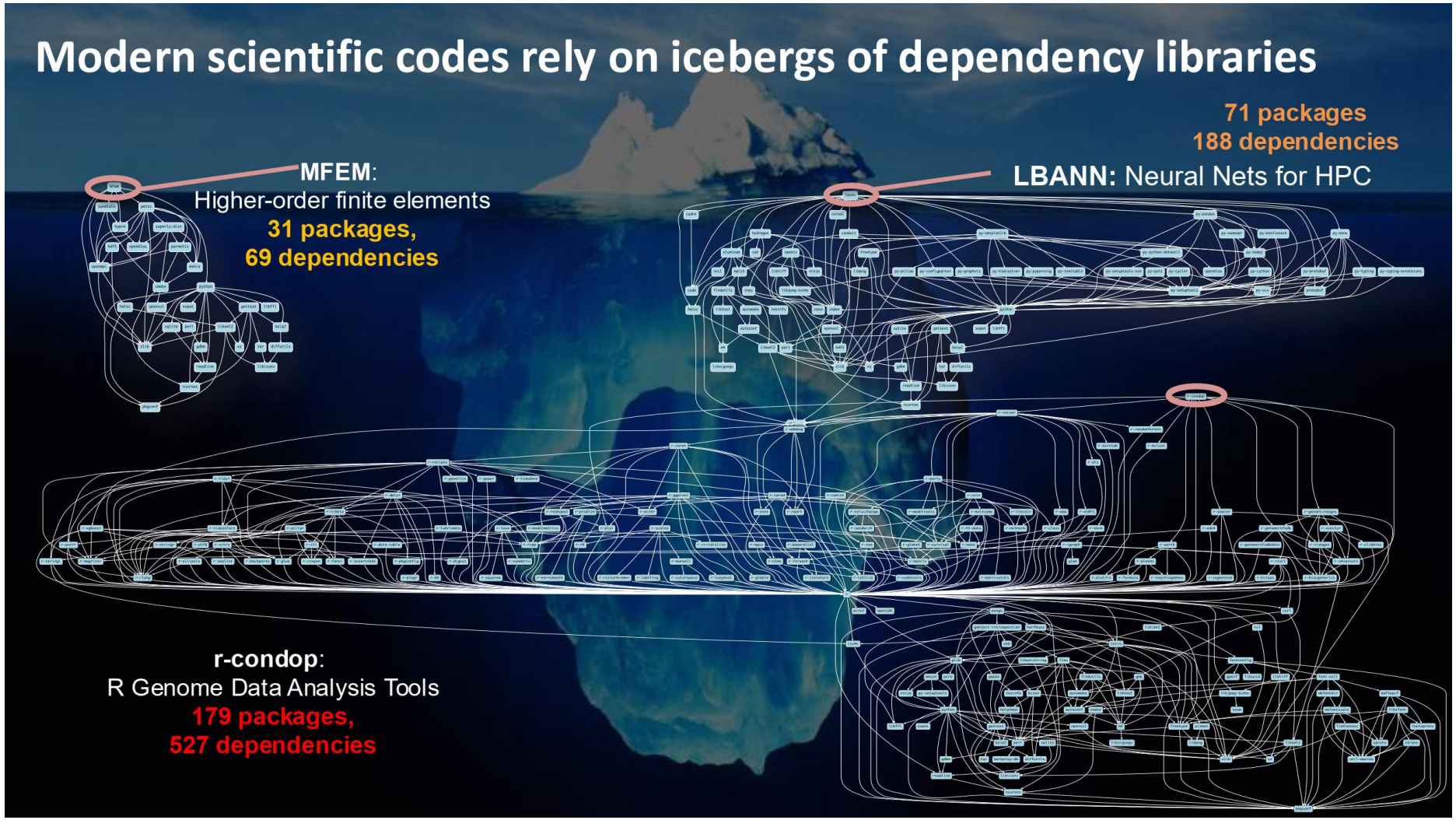
- Install Spack
- Copy your spack.yaml file from your other HPC cluster. It defines your "software environment".
- Recreate the environment on the new machine
  spack env create $NAME spack.yaml
- Tell Spack to recompile the environment and optimize it for the new machine
  spack concretize
  spack install

Most existing tools do not support combinatorial versioning

- Traditional binary package managers
  - RPM, yum, APT, yast, etc.
  - Designed to manage a single stack.
  - Install *one* version of each package in a single prefix (/usr).
  - Seamless upgrades to a *stable, well tested* stack

- Port systems
  - BSD Ports, portage, Macports, Homebrew, Gentoo, etc.
  - Minimal support for builds parameterized by compilers, dependency versions.

- Virtual Machines and Linux Containers (Docker)
  - Containers allow users to build environments for different applications.
  - Does not solve the build problem (someone has to build the image)
  - Performance, security, and upgrade issues prevent widespread HPC deployment.



Todd Gamblin, Gregory Becker, Alec Scott. Managing HPC Software Complexity with Spack. HPCIC Tutorials 2024, Livermore, California. July 22, 2024.

https://spack-tutorial.readthedocs.io/en/latest/

Spack is not the only HPC/AI/data science package manager out there

1. **"Functional" Package Managers**
   - Nix — https://nixos.org
   - Guix — https://hpc.guix.info

2. **Build-from-source Package Managers**
   - Homebrew, LinuxBrew — https://brew.sh
   - MacPorts — https://www.macports.org
   - Gentoo — https://gentoo.org

Other tools in the HPC Space:

- **Easybuild** — https://easybuild.io
  - An installation tool for HPC
  - Focused on HPC system administrators – different package model from Spack
  - Relies on a fixed software stack – harder to tweak recipes for experimentation

- **Conda / Mamba / Pixi**
  - Very popular binary package ecosystem for data science — https://conda.io
  - Not targeted at HPC; generally has unoptimized binaries — https://mamba.readthedocs.io, https://prefix.dev

How to learn more about Spack

- Main documentation: https://spack.readthedocs.io/en/latest/
- Spack tutorial
  - https://spack-tutorial.readthedocs.io/en/latest/
  - Video part 1: https://www.youtube.com/watch?v=SShzurXZr4w (3 hours)
  - Video part 2: https://www.youtube.com/watch?v=fhijfzbVCH8 (3 hours)

- Part 1

- Part 2

Spack in 1 slide

I want to build the OpenBLAS linear algebra library with GCC 12.3.0 and OpenMP multithreading enabled.

```
$ source spack/share/spack/setup-env.sh

$ spack spec -I openblas threads=openmp %gcc@12.3.0
[some output removed]

$ spack install openblas threads=openmp %gcc@12.3.0
[some output removed]
==> openblas: Successfully installed openblas-0.3.26-r7xl6gpe2esp3qwpmjj7lqjmo2mwdt62
  Stage: 0.77s.  Edit: 0.00s.  Build: 4m 16.69s.  Install: 1.49s.  Post-install: 0.09s.  Total: 4m 19.13s
[+] /home/pla/spack/opt/spack/linux-ubuntu22.04-zen2/gcc-12.3.0/openblas-0.3.26-r7xl6gpe2esp3qwpmjj7lqjmo2mwdt62

$ ls /home/pla/spack/opt/spack/linux-ubuntu22.04-zen2/gcc-12.3.0/openblas-0.3.26-r7xl6gpe2esp3qwpmjj7lqjmo2mwdt62/lib
cmake  libopenblas.a  libopenblas.so  libopenblas.so.0  libopenblasp-r0.3.26.a  libopenblasp-r0.3.26.so  pkgconfig

$ spack load openblas /r7xl6gp
$ echo $CMAKE_PREFIX_PATH
/home/pla/spack/opt/spack/linux-ubuntu22.04-zen2/gcc-12.3.0/openblas-0.3.26-
r7xl6gpe2esp3qwpmjj7lqjmo2mwdt62:/home/pla/spack/opt/spack/linux-ubuntu22.04-zen2/gcc-12.3.0/gcc-runtime-12.3.0-
25tpakyzumeqrsmxzqcnsd4uifyzvrwf
```

Initialize Spack

Ask Spack how it would install OpenBLAS this way. “Dry run”

Install!

Load the software into your shell environment (PATH, CMAKE_PREFIX_PATH etc)

A CMake build would now be able to pick up OpenBLAS

Spack provides a *spec* syntax to describe customized package configurations

```
$ spack install mpileaks                          unconstrained
$ spack install mpileaks@3.3                      @ custom version
$ spack install mpileaks@3.3 %gcc@4.7.3           % custom compiler
$ spack install mpileaks@3.3 %gcc@4.7.3 +threads  +/- build option
$ spack install mpileaks@3.3 cppflags="-O3 –g3"   set compiler flags
$ spack install mpileaks@3.3 target=cascadelake   set target microarchitecture
$ spack install mpileaks@3.3 ^mpich@3.2 %gcc@4.9.3  ^ dependency constraints
```

- Each expression is a ***spec*** for a particular configuration
  - Each clause adds a constraint to the spec
  - Constraints are optional – specify only what you need.
  - Customize install on the command line!

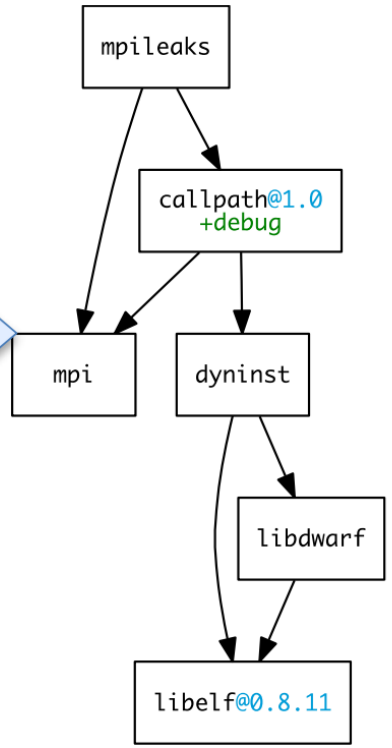
- Spec syntax is recursive
  - Full control over the combinatorial build space

Concretization fills in missing configuration details when the user is not explicit.

mpileaks ^callpath@1.0+debug ^libelf@0.8.11

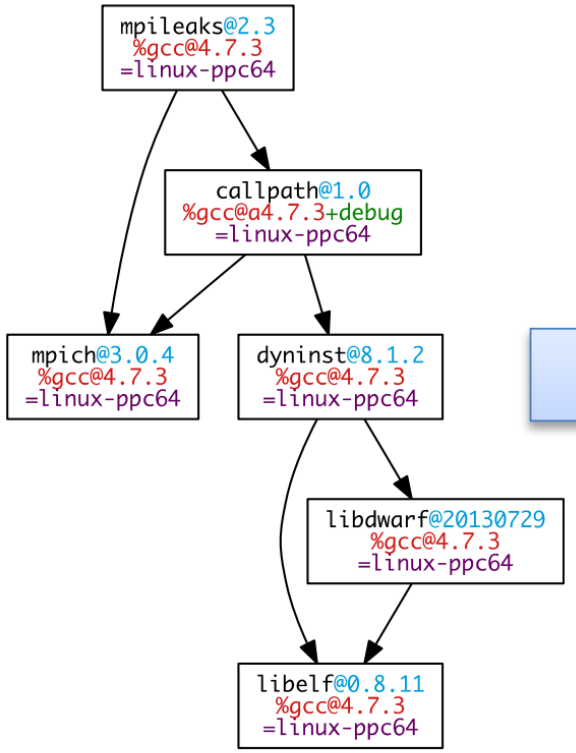
User input: *abstract* spec with some constraints

Normalize



Abstract, normalized spec with some dependencies

Concretize



Concrete spec is fully constrained and can be passed to install

Store

spec.json

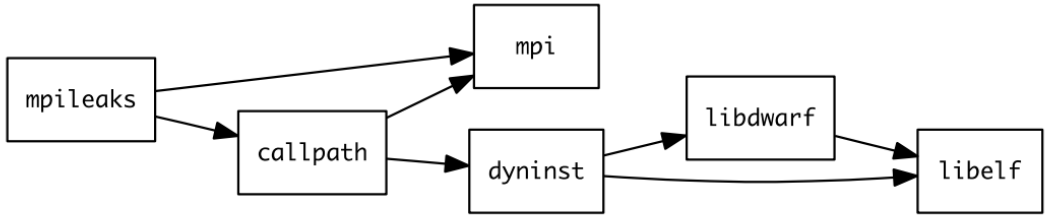

```
{
  "spec": {
    "_meta": {
      "version": 4
    },
    "nodes": [
      {
        "name": "mpileaks",
        "version": "1.0",
        "arch": {
          "platform": "linux",
          "platform_os": "rhel8",
          "target": {
            "name": "cascadelake"
          }
        },
        "compiler": {
          "name": "gcc",
          "version": "10.3.1"
        },
        "namespace": "builtin",
        "parameters": {
          "build_system": "autotools",
          "stackstart": "0",
        },
        "hash": "o7awlh5q6wccrraon4yd2mfmkdtvvnxe"

        "dependencies": [
          {
            "name": "adept-utils",
            "hash": "r3s7ywbhvtixgc3bknpqovl3dn2adce2",
            "parameters": {
              "deptypes": [
                "build",
                "link"
              ],
              "virtuals": []
            }
          },
          {
```


Detailed provenance stored with installed package

Spack handles combinatorial software complexity

Dependency DAG



Hash

Installation Layout

```
opt
└── spack
    ├── linux-rhel7-skylake
    │   └── gcc-8.3.0
    │       ├── mpileaks-1.0-hc4sm4vuzpm4znmvrfzri4ow2mkphe2e
    │       ├── callpath-1.0.4-daqqpssxb6qbfrztsezkmhus3xoflbsy
    │       ├── openmpi-4.1.4-u64v26igxvxyn23hysmklfums6tgjv5r
    │       ├── dyninst-12.1.0-u64v26igxvxyn23hysmklfums6tgjv5r
    │       ├── libdwarf-20180129-u5eawkvaoc7vonabe6nndkcfwuv233cj
    │       └── libelf-0.8.13-x46q4wm46ay4pltriijbgizxjrhbaka6
```

- Each unique dependency graph is a unique ***configuration***.
- Each configuration in a unique directory.
  - Multiple configurations of the same package can coexist.
- **Hash** of entire directed acyclic graph (DAG) is appended to each prefix.
- Installed packages automatically find dependencies
  - Spack embeds RPATHs in binaries.
  - No need to use modules or set LD_LIBRARY_PATH
  - Things work *the way you built them*

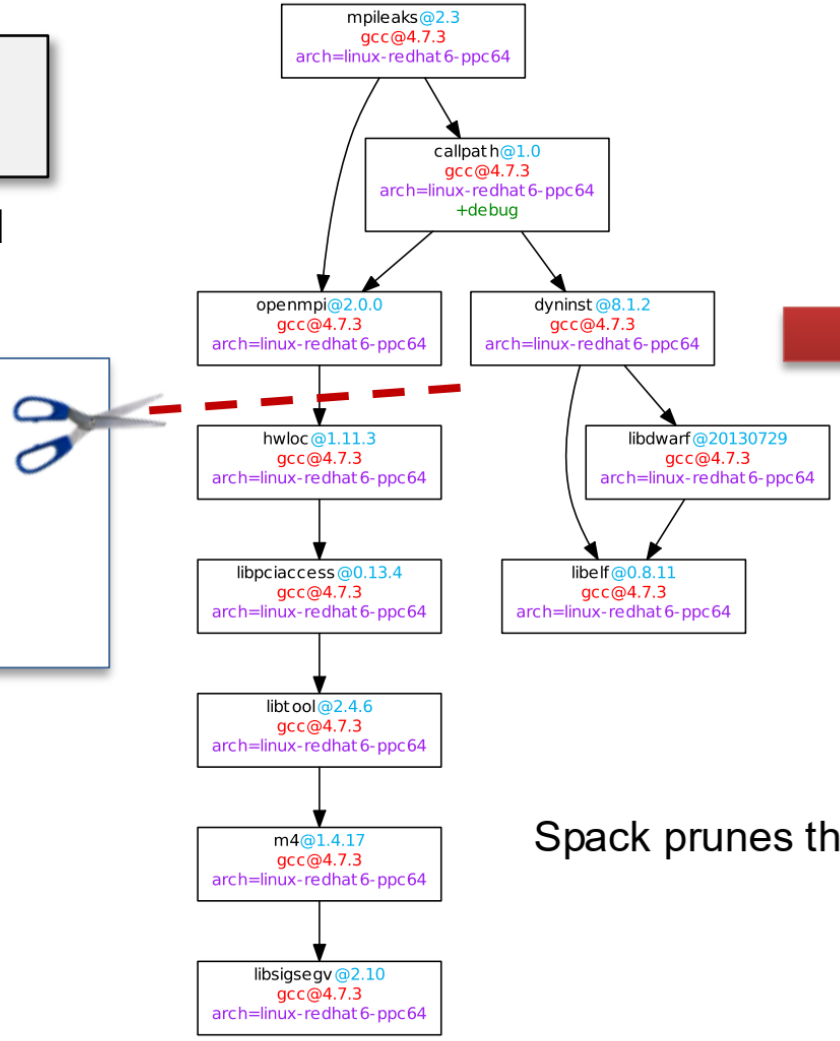
We can configure Spack to build with external software

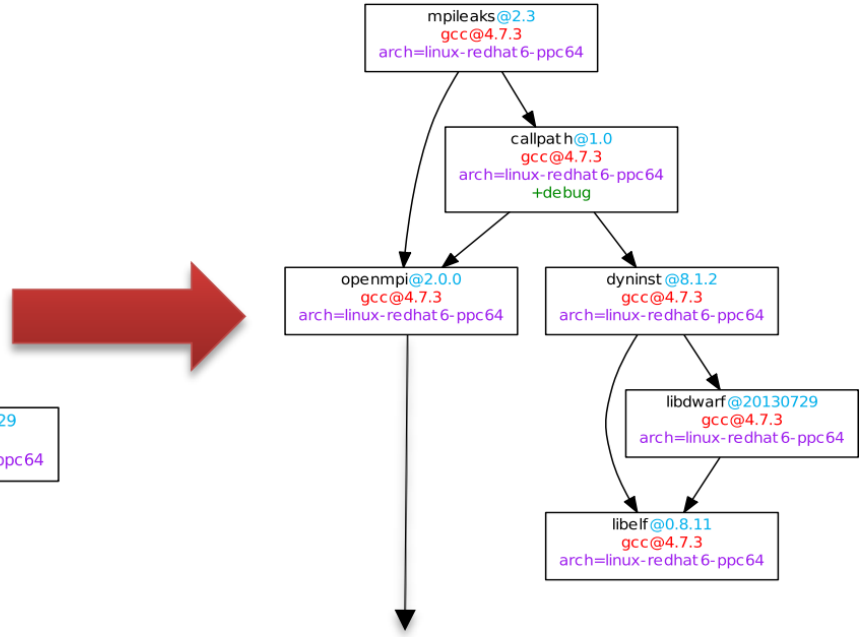
```
mpileaks ^callpath@1.0+debug
  ^openmpi ^libelf@0.8.11
```

Build using the system's OpenMPI

packages.yaml

```yaml
packages:
  mpi:
    buildable: False
    paths:
      openmpi@2.0.0 %gcc@4.7.3 arch=linux-rhel6-ppc64:
        /path/to/external/gcc/openmpi-2.0.0
      openmpi@1.10.3 %gcc@4.7.3 arch=linux-rhel6-ppc64:
        /path/to/external/gcc/openmpi-1.10.3
      ...
```





/path/to/external/gcc/openmpi-2.0.0

Users register external packages in a configuration file (more on these later).

Spack prunes the DAG when adding external packages.