Fermilab U.S. DEPARTMENT OF ENERGY | Office of Science

Spack - HSF

Stephen White, Chris Green, Marc Mengel

April 2021

A Brief History of ~~Time~~ Fermi Package Management

- Between 1986 & 1990 Femi developed a package management system, originally for VMS, called ~~VMS~~ Unix Product Support (UPS).
  - UPS was created for the distribution of binary packages as well as managing application environments.
  - It has been well-used for over 30 years! Not necessarily well-loved.
- UPS Supports
  - Multiple versions, product dependencies.
  - Multiple ‘flavors’ (IRIX, SunOS, *etc.*) .
- Reimplemented about 1992 resulting in a large speed increase
  - Could setup the CDF product stack before your coffee got cold.
  - Used by D0 and CDF while finding the Top quark.
- Overlayable UPS (2011)
  - Allowed overlaid tarball distribution.
- Over time, UPS has become embedded in virtually all product stacks at FermiLab, and has been used by most of the experiments.

Why Replace UPS?

Although still useful

- UPS relocatability allows multi-package tarball distribution.
- Allows overlapping, coherent sets of software in the same package areas.
- Non-root installation.

Downsides are

- Product maintenance is solely the responsibility of Fermi with little to no outside help.
- Fermi-specific packaging not desired by many remote collaborators.
- Encouraged build tools to use non-portable UPS/Fermi-isms.
  - Example: using `-L $FRED_DIR/lib/ -lfred` vs `pkg-config`, *etc.*
  - Builds required ups-like framework.
  - Hard to share software with non-UPS users.
- Authors/developers starting to retire.
- Large software environments lead to huge `$PATH`, `$LD_LIBRARY_PATH`.
- Use of `$LD_LIBRARY_PATH` clashes with macOS System Integrity Protection.
- No consistency in how packages are built/per-organization build scripts.
- Optimized rebuilds for supercomputer centers, *etc.* are problematic.

Requirements For a New Package Manager

It is very difficult to get users to switch to a new system if it does not support concepts of the old system that the users consider critical.

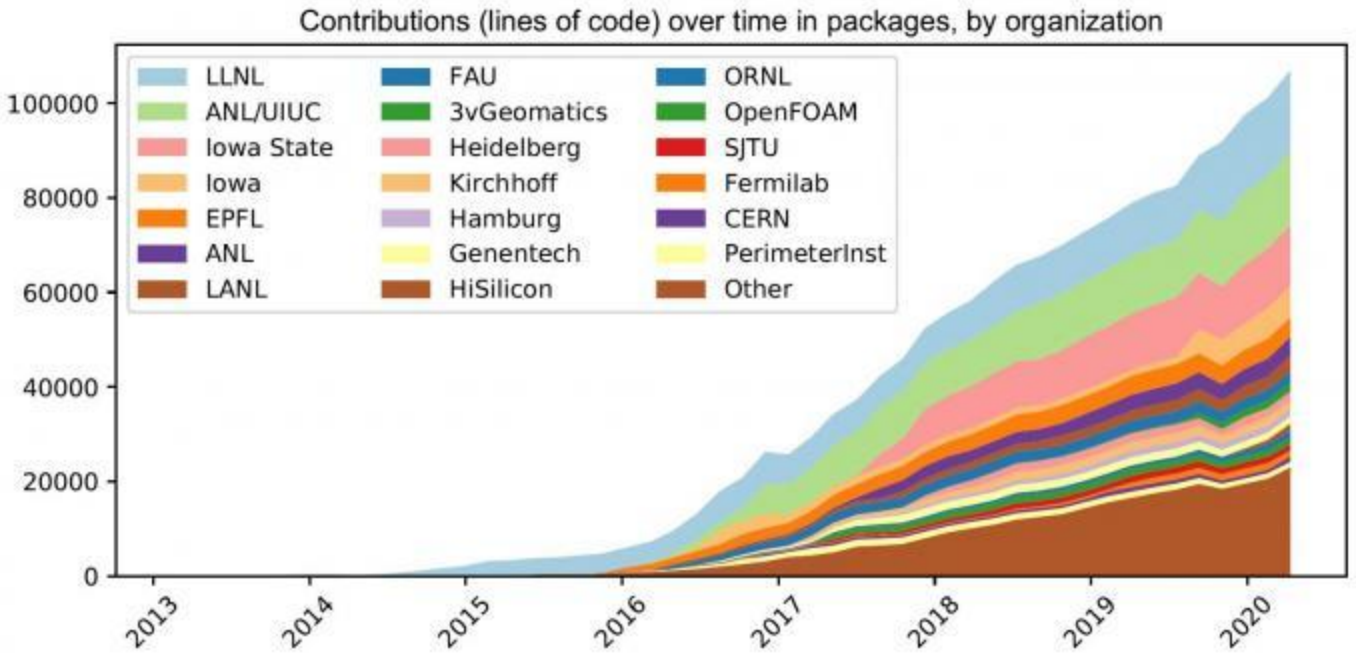
- Requirements learned from UPS:
  - A superuser must not be required.
  - Ability to setup packages area anywhere.
  - Must support multi-package, multi-platform, multi-os variability.
  - Must not be a Fermi-only solution.
  - Packages uploadable and maintainable by outside organizations.
  - Able to distribute frozen/tested binary releases.
  - Suitable for installing packages into CVMFS, allowing wide area software distribution.
- Additional requirements:
  - Additional requirements:
  - Support supercomputer-optimized rebuilds.
  - Support for multi-package simultaneous development against centrally-installed dependencies.

Spack was chosen as it supports, or could be extended to support, the stated requirements.

Spack

- In 2013 Todd Gamblin created the first prototype of a package manager he named Spack (Supercomputer PACKage manager) at Lawrence Livermore National Lab.
- Spack team at Livermore now includes computer scientists Gregory Becker, Peter Scheibel, Tamara Dahlgren, Gregory Lee, and Matthew LeGendre. The core development team also includes Adam Stewart from UIUC, Massimiliano Culpo from Sylabs, and Scott Wittenburg, Zack Galbreath, and Omar Padron from Kitware.
- Spack is now a flexible, configurable, Open Source, Python-based HPC package manager, automating the installation and fine-tuning of simulations and libraries.
- Spack can install many variants of the same build using different compilers, options, and message passing interface (MPI) implementations.
- WorldWide Use (2019 stats)
  - Over 3000 software packages.
  - Over 150,000 downloads per year.
- Today, three out of four Spack packages are contributed by outside institutions, both domestic and international. For each institution to maintain these packages separately would have been unsustainable; with Spack, each site benefits from the efforts of others.

Spack is Community Developed



Migration Requirements for Fermilab

- Embedding of UPS idioms into experiment stacks and organizational structure makes it impossible to move everything at once.
  - Production and analysis cannot be interrupted.
  - Various experiments at different points in their life-cycle.
  - Existing frozen releases “blessed” for calibration, production, *etc.*
- We must simultaneously support both UPS builds and Spack builds for an extended period of time.
- Avoid duplication of effort building for both UPS and Spack .
- Ultimately we want to use Spack, not UPS, in all but a (very) few grandfathered experiments.

In support of this we have created a suite of migration tools.

Migration Process: Tools

- ups_to_spack script
  - UPS “table file” and version entry → (partial) Spack recipe and install record.
    [*A table file describes setup and teardown actions for the environment and any dependencies required to use the product*]
  - Allows users to do a `spack load` of UPS packages.
  - Allows Spack to use UPS packages as build dependencies.
- UPS module plugins for Spack
  - Allows UPS to `setup` spack packages.
  - Similar to stock lmod / environment-modules support.
- Unified-UPS style layout
  - `.../product/version/details` directory layers.
  - Users who are used to looking in filesystem can still find things.
  - Simple config-file change to Spack.

Spack Additions: buildcache

- Originally, Spack always compiled from source to install.
- Fermilab users prefer “blessed” tested binaries for production.
- We added buildcache subcommand to Spack:

  `buildcache create`

  packs up tarballs of an installed package.

  ...and its dependencies.

  `buildcache install`

  Installs tarballs made by buildcache create.

  Fetches them from a web server.

  Relocates binaries (RPATH, RUNPATH, *etc.*).

- Allows installation on systems without compilers or development libraries.
- Buildcache now officially supported as an integral component of Spack.

Spack, buildcache and CVMFS

- Spack works well with CVMFS.
- You *could* do normal Spack installs:
  - Slows down CVMFS primary node.
  - Primary nodes not configured as build nodes.
  - Would keep transaction open a **long** time.
- buildcache installs:
  - Start a CVMFS transaction.
  - **`spack buildcache install`** packages.
  - End CVMFS transaction.
  - Transaction is open for maybe 15 minutes.

Spack Additions: chains

- UPS allowed multiple installation areas to be active simultaneously, allowing:
  - a developer's local package area ("test release"), built against:
  - multiple central shared or experiment-specific package areas ("frozen release").
- Spack now provides this through "chaining:"
  - Can see packages in your local instance and an upstream.
  - Can build packages that depend on packages in chained instances.
- Implemented originally at Fermilab, picked up and extended by a Spack primary developer.

Spack Additions: SpackDev

- FNAL experiments have a long history of relying on "dual-homed multi-package" development environments:
  - Centrally-installed full releases.
  - Locally checked-out packages (a "test release"), built together against centrally-installed dependencies.
- Previous examples: SoftRelTools / GNU Make (CDF, D0, BaBar, NoVA), MRB / cet(buildtools|modules) / CMake (most FNAL-based current experiments).
- Spack proper aimed at release-building—not well suited to multi-package development.

Spack Additions: SpackDev

- Spack extension allowing simultaneous development of multiple packages.
- Builds local packages as monolithic entities with CMake.
- Uses VCS information from recipes for checkout.
- Intended to not constrain package build system (*cf* all previous multi-package development systems).
- “Piecewise-parallel”: may build independent packages simultaneously, but not independent targets of interdependent packages.

Interlude: Previous Experience—Spack

From MVP1 & MVP1a (ref to previous talks):

- Concretization (constraint resolution) *very* slow for deep and wide dependency trees.
- Very difficult to produce a reproducible “release” with known and consistent versions and variants.
- Innocuous recipe changes cause rebuilds even when a perfectly acceptable binary package is available on buildcache due to checksum changes.
- Tight coupling between Spack code and recipes (esp. colocation) causes issues re-using packages unchanged between releases (resolution expected for Spack 1.0).
- Unnecessary rebuilds caused by unwanted rigidity re non-compiled or C-compiled packages.
- Not flexible enough re compatible microarchitectures (x86-64 from buildcache is fine for most packages, occasionally want native-compiled for *eg* Geant4, libtorch, tensorflow, *etc.*).

Interlude: Previous Spack Experience—SpackDev

- SpackDev not yet where we need it:
  - Extra constraint resolution (identify and check out intermediate packages) very slow.
  - No **`spack chains`** integration (central vs test release).
  - Relies on pulling configure / build / install arguments from concretized specs:
    - Must customize for each package build system (ick).
    - May not work for “quirky” recipes.
  - Difficult to define and maintain a consistent “release.”

cetbuildtools Versus cetmodules

- cetbuildtools: CMake-based build system used by experiments to build their own code.
  - Assumes the use of UPS, heavily reliant on environment variables, packaging quirks, *etc.*—replacement/upgrade must be part of any upgrade plan.
- cetmodules v1.0:
  - First attempt to allow compilation of cetbuildtools-using code with Spack, using and producing Spack recipes and binary packages.
  - Required significant, non-script-able changes in CMake files.
  - Dozens of forked packages; conversion must be repeated for each release to be compiled with Spack.
  - Too-close coupling of the build system to the package manager resulted in the need for extensive changes to client code. This impacted significantly our agility and ability to stage the project and expose it to users early.

cetmodules 2

- Complete overhaul with the aim of removing the need for intrusive manual changes to cetbuildtools-using packages.
- Capable of impersonating cetbuildtools *without changes to experiment CMake files.*
- Changes to support building with Spack / without UPS can be made progressively *without compromising the ability to use and build for UPS*—no forking required.
- Supports “modern” (>3.0) CMake paradigms (targets, interface libraries, *etc.*).
- Has a migration script (currently covers most changes necessary for compilation with Spack).

cetmodules 2 Status

- Functionally complete re UPS compatibility, most “new” CMake functionality (no support for building packages with `COMPONENTS` yet).
- With MRB 5, develop cetmodules / cetbuildtools packages together without friction.
- art suite 3.08 migrated to cetmodules, successfully built with Spack & UPS.
- Tested back to LArSoft version used by MicroBooNE production (based on art suite 2.13).
- Documentation still incomplete (working on it!)

Remaining Work

- Finish cetmodules documentation—in progress.
- Make major software stacks buildable both ways (Art Suite, LARSoft, *etc.*)—in progress.
- Investigate and incorporate recent developments to Spack:
  - Much-improved Spack Environments (frozen releases).
  - New concretizer based on a formal Python / C-based constraint solver—still not default, teething troubles, may not (yet) satisfy all our requirements. Will be default in next release (about 2 months).
  - Incorporate Spack chains.
  - Investigate recent Spack-native development features: may be able to incorporate missing features from SpackDev.
- Actually get experiments to change how they develop, build and distribute their releases.
- Retire UPS / Spack migration tools.

Fermilab Projects Converting to Spack

Umbrella of work in progress with experiments and projects to meet their needs while we are progressing in development efforts.

- CosmoSIS
  - Converted their existing code stack to Spack. The package stack was a hybrid UPS system, but their product build system was not dependent on it.
- Icarus for Theta (supercomputer at ALCF)
  - Has built their software stack under Spack on a non-FNAL system.
  - Currently working on a Spack build for Theta.
- Scientific Computing Division
  - Many UPS products now have Spack native builds. (POMS, Redmine, ifdh suite, *etc.*)
- G-2
  - Conversion of UPS is starting. They are not a user of MRB but have their own similar product.
- LArSoft / art
  - art is using native cetmodules 2 and now builds under Spack. The art suite is out and in use by a non-LArSoft experiment (Mu2e).
  - LArSoft conversion is in progress.
- DAQ
  - Waiting for LArSoft & art to stabilize with Spack.
- DUNE
  - Focused efforts enable some HPC tasks to use Spack build. The experiment has decided to wait for a completed Spack build of LArSoft.

Lessons Learned

- Software packages need to be buildable under multiple packaging systems.
  - Tying a package's build system to the packaging system has consequences.
  - Use standard tools for package location ...
    - **`pkg-config`**,
    - **`cmake`**,
    - **`autoconf`**, *etc.*
  - … not packaging system:
    - **`$FRED_FQ_DIR`**
    - **`spack find --paths <package>`**…

Lessons Learned

- Migration efforts are difficult to estimate:
  - Experiments have their own priorities.
  - Requirements are discovered late.
  - Multiplying estimates by π is not sufficient.
- Relying on an external system (Spack) means either waiting on their development efforts or massive effort on our part—but still better than roll-your-own.