Build and Test Automation at Livermore Computing

National Center for Computational Sciences Seminar
Presented at Oak Ridge National Laboratory

Todd Gamblin
Center for Applied Scientific Computing

October 20, 2015

LLNL-PRES-678531
This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344. Lawrence Livermore National Security, LLC

Lawrence Livermore National Laboratory

Modern software developers employ a range of collaborative tools.

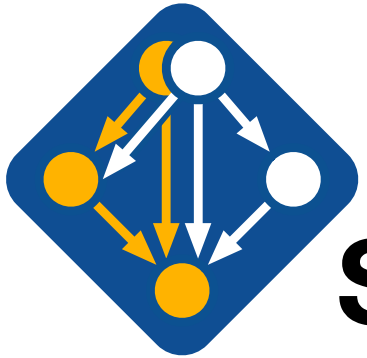
- Collaborative web applications:
  - Wikis (MediaWiki, Trac, Confluence)
  - Issue trackers (RoundUp, Bugzilla, JIRA)
  - Repository systems (Stash, Bitbucket, GitHub, GitLab)
- Automated Testing:
  - Build & Test / Continuous Integration (Jenkins, Bamboo, Travis CI)
  - Also generally involves a web dashboard.
- Package Management / automated build:
  - Homebrew, MacPorts, RPM, apt-get, yum, BSDPorts, Emerge/Gentoo
  - For HPC: Spack, Smithy, Easybuild

For many open source projects, this infrastructure is freely available.
Not widely available inside HPC centers.

This talk gives an overview of two tools currently in use at Livermore Computing

1. **Atlassian Bamboo** Continuous Integration
   - Security challenges for CI
   - How Bamboo was deployed securely at LLNL

2. The **Spack Package Manager**
   - Developed at LLNL
   - Handles combinatorial build explosion
   - Now in use to build/test production codes



Spack

Web-based tools are often difficult to deploy in HPC environments

- Two-factor authentication often hard to integrate with many web applications.

- Security policies require more strict permissions than most open source tools offer.
  - Tools often assume trust among users.
  - Many tools don't have fine-grained access controls.
- Maintenance costs are higher with security
  - With no permissions, need an instance per team
    - Centers can't maintain all the instances.
  - Small teams have no resources to deploy tools.
    - HPC developers (for the most part) lack web skills.
  - Most centers prohibit running arbitrary web services.

LLNL Networks are a challenge for deployment

Organization	Network/ URL	Identity management			Accessible from								SFNs can use	
		Username	*Authentication*	*Groups*	*Internet (no VPN)*	*VPN VPN-C*	*VPN-B*	*Blue*	*Yellow*	*CZ*	*RZ*	*iSRD/ SCF*	*On-site*	*From home*
Institutional	**Yellow / Blue** atlassian.llnl.gov	OUN	AD	Custom, Managed by admins	✗	✓	✓	✓	✓	⚠	⚠	✗	✓	✗
Livermore Computing (LC)	**Collaboration Zone (CZ)** lc.llnl.gov	LC Username	RSA SecurID 	LC Unix Groups	✓	✓	✓	✓	✓	✓	✓	✗	✓	✓
	Restricted Zone (RZ) rzlc.llnl.gov		CryptoCard 		✗	✓	✓	✓	✓	✓	✓	✗	✗	✗
	Classified (SCF) lc.llnl.gov		RSA SecurID 		✗	✗	✗	✗	✗	✗	✗	✓	✗	✗

⚠ U.S. Citizens on the CZ and RZ can access the Institutional instance over https (port 443) and SSH port 7999, but foreign nationals cannot.

- Maximum consolidation is one tool per network
- Can't merge across identity systems or security levels.
  - Exception: LANL & Sandia users have federated login with LC, can use own credentials.

Livermore Computing (LC) has deployed Atlassian Tools for all users

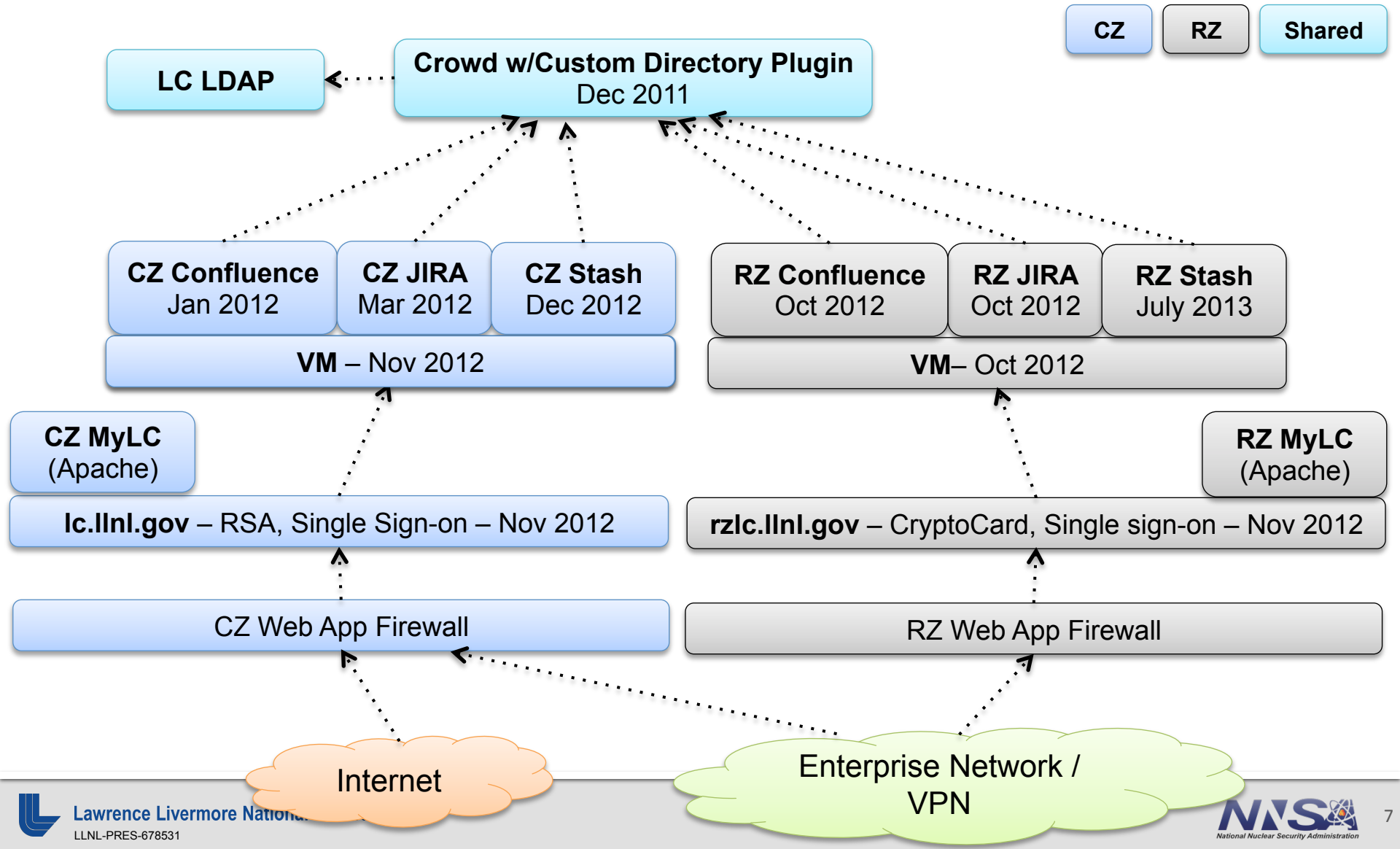
- Wiki software for collaborative projects
- Collaborative meeting notes, design documents, documentation, other projects

- Issue tracking for software and other projects
- Agile project management tools

- Git repository hosting;
- Like github or sourceforge
- Cheaper and inside the firewall

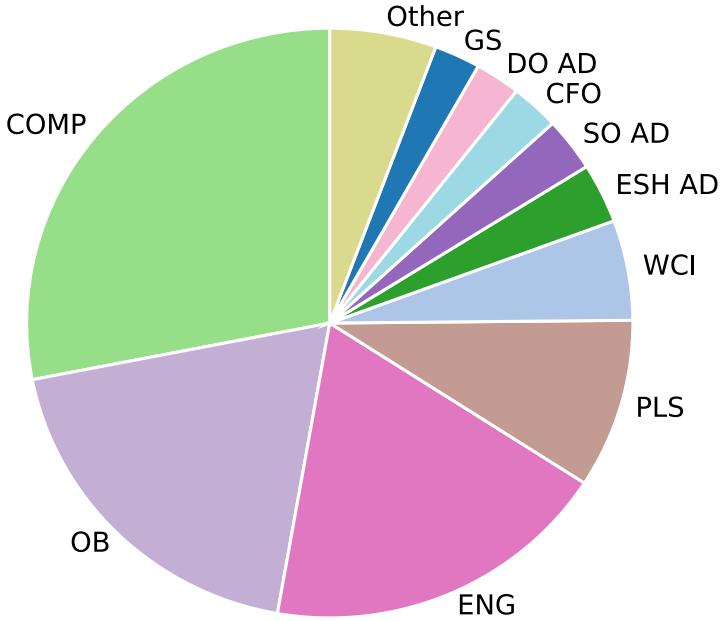
Atlassian deployment has been ongoing since 2011

This shows how our ecosystem evolved.

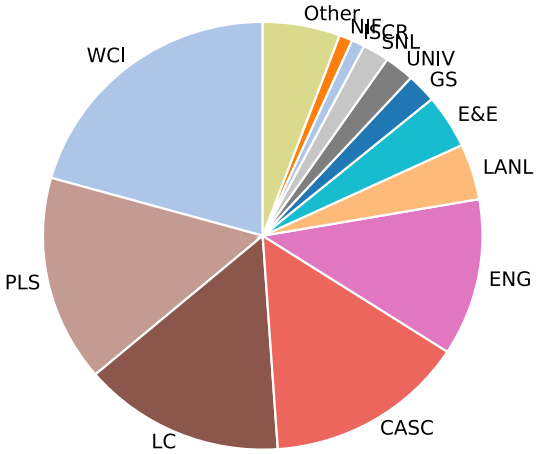


Atlassian tools have become extremely popular at LLNL

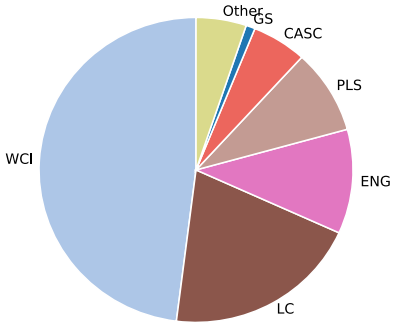
Enterprise Network
(2,479 users)



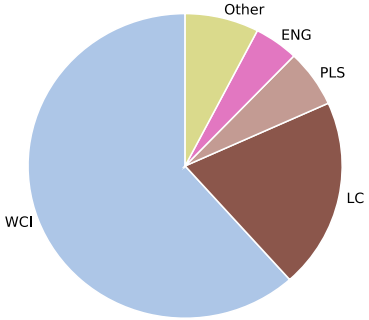
HPC Networks



Collaboration Zone (1,231 users)



Restricted Zone (425)



Secure Computing (263)

These 4 instances consolidated hundreds of disparate web tools across the lab.

Why Atlassian Tools?

- **Excellent support for fine-grained permissions across tools**
  - Easily conforms to LLNL security plans.
  - Projects can be private or share via unix groups.
- **Scalable to many teams with low maintenance**
  - Users can create their own projects on demand; no admin involved.
  - Nice UI and online documentation makes them easy to adopt
- **Single integration point (Crowd), single sign-on.**
  - Crowd is our interface to LC's (rather esoteric) LDAP directory
  - One custom directory developed for all tools.
  - Offers single sign-on – user types two-factor auth once per day.
- **Consolidation, collaboration, support for small teams.**
  - NOT FREE or OSS, but customers do get access to the source.

Tools have broken down organizational silos and lowered maintenance burden.
Some free tools are starting to catch up (GitLab).

HPC developers need better Build and Test tools.

- **Bleeding-edge machines need the most testing**
  - Newest hardware
  - Least stable OS
  - Newly ported software
  - Constantly changing environment
- **Hard to run build/test suites on these machines**
  - Some centers don't even allow cron jobs.
  - Job submission systems are different across HPC centers
    - Extra porting work
  - Raises barrier to entry for software
- **Many code teams need to run in multiple environments**
  - Manual testing for so many configurations is intractable

HPC Centers must lower barriers to entry for end users!

Continuous Integration (CI) Tools can help

- CI systems monitor repositories for changes, then:
  - Build latest version(s)
  - Run tests
  - Generate reports

Jenkins

Atlassian Bamboo

- Automate common development workflows
  - Provide nice GUIs for constructing test workflows
  - Integrate with bug trackers, repository systems

Travis CI

- Examples:
  - Jenkins: Java server with remote “agents”
  - Bamboo: Java server with remote “agents”
  - Travis CI: Hosts CI builds in the cloud (mostly Ubuntu)

Continuous Integration tools pose a number of security challenges for centers.

1. They launch real jobs on machines.
2. May launch jobs on remote machines.
3. Job launch is automated, without a user present.
4. Typically need to check out code from a repository.
   - May store credentials for that repository.
5. Need access to resource manager to submit parallel jobs.
6. Build agents may share data between builds.
   - Can't allow users' jobs to share data.

LLNL code teams currently run their own CI systems

- Resource-intensive maintenance
  - Each team has many single-node Jenkins instances
  - Each instance has its own web dashboard with a strange URL
    - https://clusternode2134:7777/jenkins/
- URLs are only accessible on the local network
  - Most users launch a browser over X11 to administer (extremely slow)
  - More sophisticated users can access via SSH tunnel
- Teams only use local, single-node agents
  - HPC code teams not have the knowhow to secure SSL remote agents
  - Insecure agents would result in spurious builds.
- Teams are responsible for security of their Jenkins instances
  - Have to

We started by listing our requirements for CI systems.

1. LC users are able to log into the system with a web browser using their LC username and two-factor authentication.
2. LC users can set up suites of build and test jobs to run regularly using a simple UI.
3. LC Users can collaborate on test configurations using LC Unix groups and permissions.
4. LC Unix groups and permissions can be used to restrict access to viewing and editing test configurations.
5. Server can launch build/test jobs on cluster front-end nodes.
6. Build/test jobs must run only as a user authorized by the creator of a test configuration.
7. Network traffic between the server and build agents must be encrypted.

Comparison of Jenkins, Bamboo, & Travis

Product	Hosting Options	User Interface	Plugins	Remote Build	Price	Atlassian integration	Users/ Groups	Permissions	Agent security	Node Isolation
Bamboo	Hosted or internal	Easy, clean	Some	Build agents	$$	JIRA, Stash	Crowd	Per project	SSL	Can dedicate agents to a project.
Jenkins	Internal server	Complex, cluttered. ROSE team wants something simpler (currently use Jenkins).	Lots (1k+)	Build agents	Free	Partial	Crowd	Matrix-based, or Role strategies. Configuring per project seems complex.	SSL/ SSH	Projects can restrict jobs to particular slaves and vice versa.
Travis CI	Hosted only	Simple, but based on yaml + git		Hosted only	Free	N/A	Github	N/A	N/A	N/A

The Bamboo web application can be configured to satisfy LC's security requirements.

1. **Authentication**
   - Use Atlassian Crowd for two-factor (already deployed)

2. **Per-project configuration**
   - Bamboo allows users to set up *Projects* and *Plans*
   - Users only see projects and plans they're added to

3. **Permissions**
   - Bamboo essentially has 3 levels of in-application permissions:
     a. System administrator (manages Bamboo)
     b. Plan creator/administrator (team lead, manages members)
     c. Plan user/developer (team member)

- Many users can share the same web server.
- Users can mostly self-manage on this system, using Unix groups.
- No sys-admin support needed for setting permissions.

Bamboo permissions are configurable per build plan.

Plan details | Stages | Repositories | Triggers | Branches | Dependencies | Permissions | Notifications | Variables | Miscellaneous | Audit log

Permissions

You can edit your plan permissions here. Permissions can be granted to specific users or groups.

Add user | Add group

Users	View	Edit	Build	Clone	Admin
Todd Gamblin	☑	☑	☑	☑	☑
Matthew P. Legendre	☑	☑	☑	☑	☑
Peter J. Scheibel	☑	☑	☑	☑	☑
Groups	**View**	**Edit**	**Build**	**Clone**	**Admin**
deg	☑	☐	☐	☐	☐
Other	**View**	**Edit**	**Build**	**Clone**	**Admin**
Logged in users	☑	☐	☐	☐	☐
Anonymous users	☐				

Save | Cancel

Permission types

View - User can view the plan in Bamboo, including its builds.
Edit - User can view and edit the configuration of the plan and it's jobs. This does not include the ability to change a plan's permissions or its stages.
Build - User can trigger a manual build on the plan, as well as suspending and resuming the plan.
Clone - User can clone the plan.
Admin - User can administrate all components of this plan including the stages and the plan's permissions.

Note:

Users with the global 'admin' permission have all of the above permissions for this plan.

Job launch starts to get more complicated.

- **Bamboo can launch jobs locally on the server**
  - Local jobs would run as the web application user, but server is shared.
  - We **disable local agents** and *do not* run local jobs.

- **Bamboo can also run jobs in Amazon EC2 cloud**
  - We've disabled this.
  - Could be interesting for testing open software in the future

- **Bamboo can send jobs to remote agents.**
  - LC *only* uses remote agents on cluster login nodes.
  - Jobs launched by agents run as same user as agent.
  - How to secure remote agents?

Remote Agents:
Plan A: Dedicated agents (currently deployed)

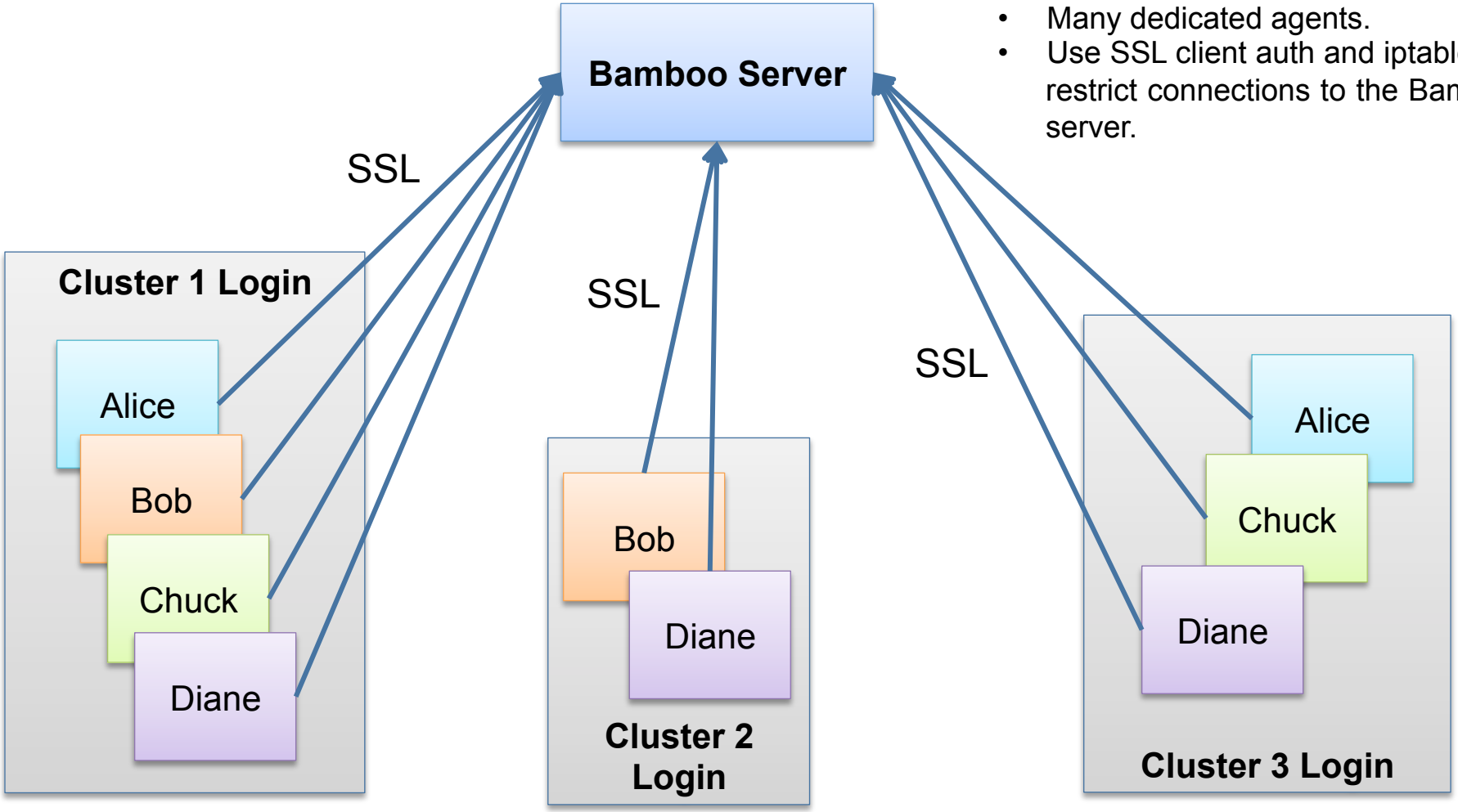
- **Launch:**
  - Agents are launched by LC users
  - Agents run on login nodes as the user who started them

- **Use control:**
  1. **Dedication**
     - Agents must be "dedicated" to specific build plans
     - Dedicated agents will *only* run builds for specific plans.
     - Agents are typically owned by the same user as the build plan.
  2. **Authorization**
     - Can require sysadmins to *authorize* agents before they can do any builds
     - Sysadmin has to first approve each agent and ensure that it's dedicated to the right plan.

- **Transport:**
  - Agents use SSL with client authentication to connect to the server
  - Only agents with LC-signed certificate can connect.

Pros: Requires no special customization of Bamboo.
Pitfalls: Requires one-time administrator effort for each agent.
Requires many per-user agents on the same node. Can get expensive.

Remote Agents:
Plan A: Dedicated agents (currently deployed)



- Many dedicated agents.
- Use SSL client auth and iptables to restrict connections to the Bamboo server.

Remote Agents:
Plan B: Setuid agents (in development)

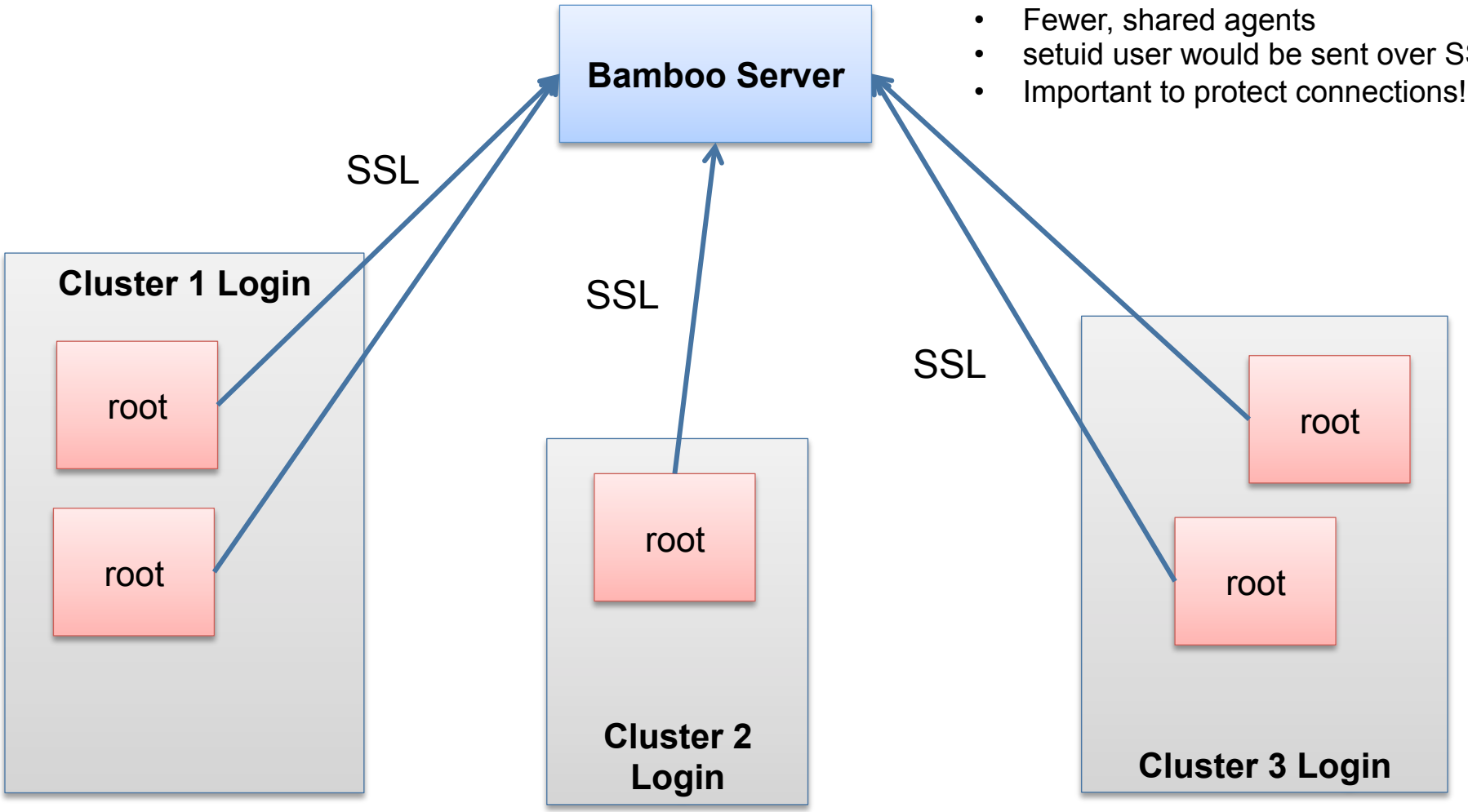
- **Launch:**
  - Agents are launched as system services.
  - Agents run on login nodes as root.

- **Use control:**
  - Agents run as root until they receive a job, then:
    1. Change permissions for working directory to the job's user
    2. setuid to appropriate user and run job
    3. Die and automatically restart.
  - No repeated authorization and dedication required
  - User sees each LC cluster as an agent (or several)

- **Transport:**
  - Agents use SSL with client authentication to connect to the server
  - Only agents with LC-signed certificate can connect.

Pros: One-time agent setup, no user involvement.
Pitfalls: Requires developing plugins for the agents, may require building/maintaining a custom agent (could be brittle).

Remote Agents:
Plan B: Setuid agents (in development)



- Fewer, shared agents
- setuid user would be sent over SSL.
- Important to protect connections!

Current Status of Bamboo deployment

- **Deployed for users in 2 zones: Collaboration Zone and Restricted Zone**
  - Easy web access is appealing: many have started switching from Jenkins.
  - Users can now test on more LC front-end nodes with remote agents.

- **Issues**
  1. Many users have repos in shared FS that can't be polled by server.
     - Server doesn't mount shared FS, REST build triggers with SSL certs.
  2. Some issues with IBM JDK on Blue Gene/Q front-end

- **Future Directions**
  1. Agent setup needs more automation (for signing certs)
  2. **Users want to run LOTS of combinatorial tests.**

Users want to experiment with many different configurations of their code.

- Every machine and app has a different software stack (or several)
- We want to experiment quickly with many architectures, compilers, MPI versions
- All of this is necessary to get the best ***performance***

48 third party packages

x **3 MPI versions**
mvapich mvapich2 OpenMPI

x **3-ish Platforms**
Linux BlueGene Cray

x **Up to 7 compilers**
Intel GCC XLC Clang
PGI Cray Pathscale

x **Oh, and 2-3 versions of each**

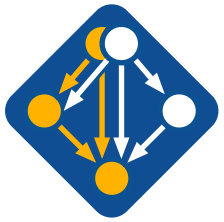
= **~7,500** combinations

- OK, so we don't build **all** of these
  - Many combinations don't make sense
- We want an easy way to quickly sample the space
  - Build a configuration on demand!

How do HPC sites deal with combinatorial builds?

- **OS distribution does not deal with this**
  - OS typically has one version of each package, installed in a common prefix: `/usr`
- **HPC software typically installed manually in a directory hierarchy.**
  - Hierarchy often doesn't give you all the information you need about a build.
  - Typically run out of unique names for directories quickly.
- **Environment modules allow you to enable/disable packages.**

Site	Naming Convention
LLNL	`/ usr / global / tools / $arch / $package / $version` `/ usr / local / tools / $package-$compiler-$build-$version`
Oak Ridge	`/ $arch / $package / $version / $build`
TACC	`/ $compiler-$comp_version / $mpi / $mpi_version / $package / $version`



Spack handles combinatorial version complexity.

```
spack/opt/
    linux-x86_64/
        gcc-4.7.2/
            mpileaks-1.1-0f54bf/
    bgq/
        gcc-4.5.1/
            libelf-0.8.13-251fqb/
    ...
```

- Each unique DAG is a unique configuration.
- Many configurations can coexist.
- Each package ***configuration*** is installed in a unique directory.
- **Hash** appended to each prefix allows versioning of full dependency DAG.

- **Installed packages will automatically find their dependencies**
  - Binaries are installed with proper RPATHs
  - No need to use modules or customize LD_LIBRARY_PATH
  - Things continue to work *the way you built them*

- **Installation works just as well in $HOME as in shared FS.**

`spack list` shows what's available

```
$ spack list
==> 219 packages.
adept-utils  dyninst          lcms          mesa          pixman          py-pypar             spindle
arpack       extrae           libarchive    metis         pmgr_collective py-pyparsing         sqlite
asciidoc     exuberant-ctags  libcircle     Mitos         postgresql      py-pyqt              stat
atk          flex             libdrm        mpc           ppl             py-pyside            sundials
atlas        flux             libdwarf      mpe2          py-basemap      py-python-daemon     swig
autoconf     fontconfig       libelf        mpfr          py-biopython    py-pytz              task
automaded    freetype         libevent      mpibash       py-cffi         py-rpy2              taskd
automake     gasnet           libffi        mpich         py-cython       py-scientificpython  tau
bear         gcc              libgcrypt     mpileaks      py-dateutil     py-scikit-learn      tcl
bib2xhtml    gdk-pixbuf       libgpg-error  mrnet         py-epydoc       py-scipy             the_silver_searcher
binutils     geos             libjpeg-turbo munge         py-genders      py-setuptools        thrift
bison        git              libjson-c     muster        py-gnuplot      py-shiboken          tk
boost        glib             libmng        mvapich2      py-h5py         py-sip               tmux
boxlib       global           libmonitor    nasm          py-ipython      py-six               tmuxinator
bzip2        gmp              libNBC        ncurses       py-libxml2      py-sympy             uncrustify
cairo        gnutls           libpciaccess  netcdf        py-lockfile     py-virtualenv        util-linux
callpath     gperf            libpng        netgauge      py-mako         py-yapf              vim
cblas        gperftools       libsodium     netlib-blas   py-matplotlib   python               vtk
cgm          graphlib         libtiff       nettle        py-mock         qhull                wget
clang        gtkplus          libtool       ompss         py-mpi4py       qt                   wx
cloog        harfbuzz         libunwind     opari2        py-mx           qthreads             wxpropgrid
cmake        hdf5             libuuid       openmpi       py-nose         R                    xcb-proto
coreutils    hwloc            libxcb        openssl       py-numpy        ravel                xz
cppcheck     hypre            libxml2       otf           py-pandas       readline             yasm
cram         icu              libxshmfence  otf2          py-pexpect      rose                 zeromq
cscope       icu4c            libxslt       pango         py-pil          ruby                 zlib
cube         ImageMagick      llvm          papi          py-pmw          SAMRAI               zsh
czmq         isl              llvm-lld      paraver       py-pychecker    scalasca
dbus         jdk              lua           parmetis      py-pycparser    scorep
docbook-xml  jpeg             lwgrp         parpack       py-pyelftools   scotch
dri2proto    lapack           lwm2          pcre          py-pygments     scr
dtcmp        launchmon        memaxes       petsc         py-pylint       silo
```

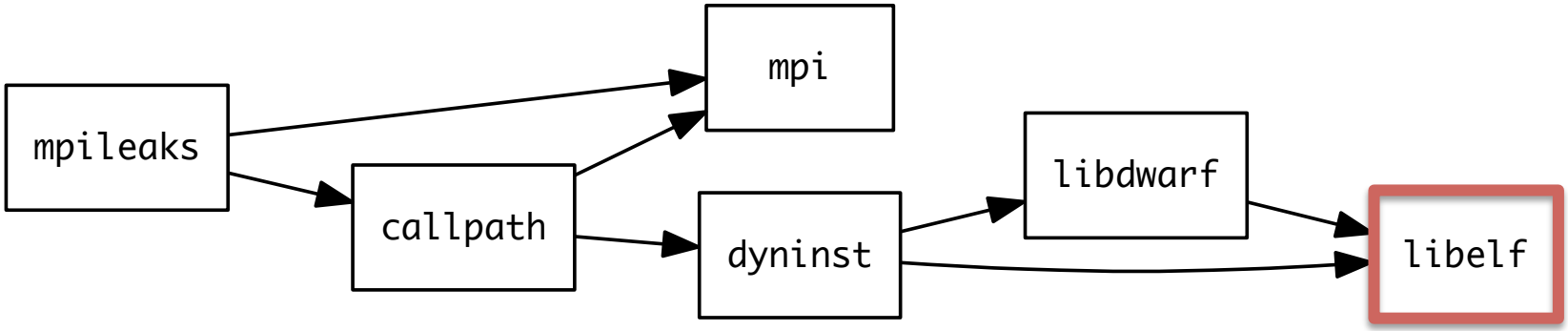
Spack provides a *spec* syntax to describe customized DAG configurations

```
$ spack install tool                          default: unconstrained

$ spack install tool@3.3                            @ custom version

$ spack install tool@3.3 %gcc@4.7.3                 % custom compiler

$ spack install tool@3.3 %gcc@4.7.3 +threads        +/- build option

$ spack install tool@3.3 =bgqos_0                   = cross-compile
```

- 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!

- Package authors can use same syntax within package files
  - Makes it easy to parameterize build by version, compiler, arch, etc.

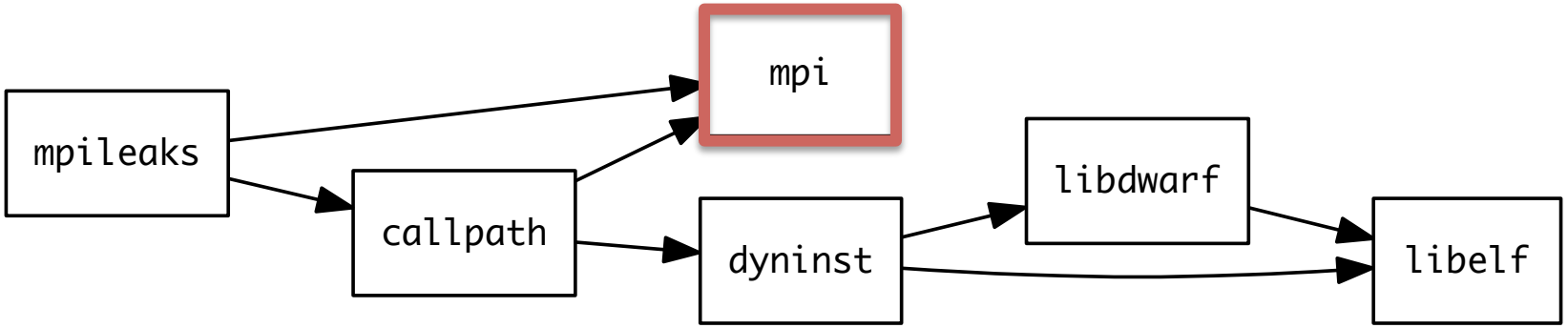
Specs can constrain dependency versions




```
$ spack install mpileaks %intel@12.1 ^libelf@0.8.12
```

- Spack ensures that all packages in the same install are built with the same version of libraries, like **`libelf`**.
- Spack can ensure that builds use the same compiler
  - Can also mix compilers but it's not default

Spack handles ABI-incompatible, versioned interfaces like MPI



Ask specifically for mvapich 1.9

```
$ spack install mpileaks ^mvapich@1.9
```

Ask for openmpi 1.4 or higher

```
$ spack install mpileaks ^openmpi@1.4:
```

These install separately, in unique directories

Ask for an MPI that supports MPI-2 interface

```
$ spack install mpileaks ^mpi@2
```

Spack chooses an MPI version that satisfies constraint

Spack packages are simple Python

```
from spack import *

class Dyninst(Package):
    """API for dynamic binary instrumentation.  Modify programs while they
       are executing without recompiling, re-linking, or re-executing."""

    homepage = "https://paradyn.org"

    version('8.2.1', 'abf60b7faabe7a2e', url="http://www.paradyn.org/release8.2/DyninstAPI-8.2.1.tgz")
    version('8.1.2', 'bf03b33375afa66f', url="http://www.paradyn.org/release8.1.2/DyninstAPI-8.1.2.tgz")
    version('8.1.1', 'd1a04e995b7aa709', url="http://www.paradyn.org/release8.1/DyninstAPI-8.1.1.tgz")

    depends_on("libelf")
    depends_on("libdwarf")
    depends_on("boost@1.42:")


    # new version uses cmake
    def install(self, spec, prefix):
        libelf = spec['libelf'].prefix
        libdwarf = spec['libdwarf'].prefix

        with working_dir('spack-build', create=True):
            cmake('..',
                  '-DBoost_INCLUDE_DIR=%s' % spec['boost'].prefix.include,
                  '-DBoost_LIBRARY_DIR=%s' % spec['boost'].prefix.lib,
                  '-DBoost_NO_SYSTEM_PATHS=TRUE',
                  '-DLIBELF_INCLUDE_DIR=%s'   % join_path(libelf.include, 'libelf'),
                  '-DLIBELF_LIBRARIES=%s'     % join_path(libelf.lib, 'libelf.so'),
                  '-DLIBDWARF_INCLUDE_DIR=%s' % libdwarf.include,
                  '-DLIBDWARF_LIBRARIES=%s'   % join_path(libdwarf.lib, 'libdwarf.so'),
                  *std_cmake_args)
            make()
            make("install")

    # Old version uses configure
    @when('@:8.1')
    def install(self, spec, prefix):
        configure("--prefix=" + prefix)
        make()
        make("install")
```

Metadata

Version/URLs

Dependencies

Patches (not shown)

Commands for install

Access build config through *spec.*

- Package files live in repositories.
- ‘spack create’ command generates boilerplate package given a URL.

Spack supports optional dependencies

- Based on user-enabled variants:

```
variant("python", default=False, “Build with python support”)
depends_on("python", when="+python")
```

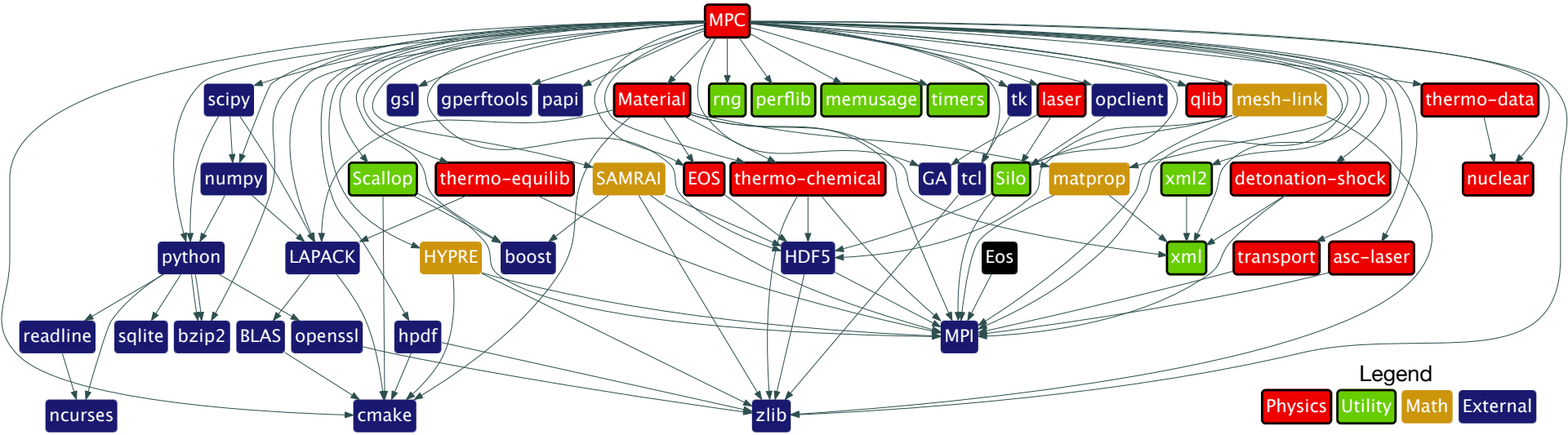
```
spack install vim +python
```

- And according to other spec conditions
  e.g., gcc dependency on mpc from 4.5 on:

```
depends_on("mpc", when="@4.5:")
```

- DAG is not always complete before concretization!

Example: Spack has recently been adopted by an LLNL production code.



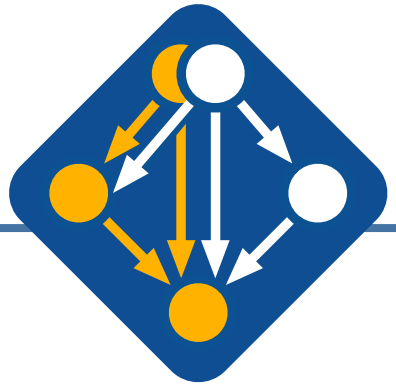
- **Dependencies shown above**
  - 47 component packages
  - ~50% physics + math
  - ~50% open source
- **Spack automates this build of and those of its dependencies**
  - Also being used to automate post-build testing.

Our production code team uses Spack to test 36 different configurations

	Linux			*BG/Q*	*Cray XE6*
	MVAPICH	*MVAPICH2*	*OpenMPI*	*BG/Q MPI*	*Cray MPI*
GCC	C P L D			C P L D	
Intel 14	C P L D				
Intel 15	C P L D	D			
PGI		D	C P L D		C L D
Clang	C P L D			C L D	
XL				C P L D	

- **Above are nightly builds on machines at LLNL and LANL**
  - **Zin, Sequoia, Cielo**
- **4 code versions:**
  - **(C)**urrent Production **(L)**ite
  - **(P)**revious Production **(D)**evelopment
- **Team is currently porting to the new Trinity machine**

Spack has a growing community.



- **Spack is starting to be used in production at LLNL**
  - Used for tool installation at Livermore Computing (LC)
  - Used by code teams at LLNL, starting to be adopted by more.
  - Will enable a common deployment environment for LC and codes.
- **Spack has a growing external community**
  - NERSC working with us to use Spack on Cori XC40
  - Kitware is working with us to package ParaView and develop core features.
  - NNSA: Participation in Spack calls by Sandia, LANL
  - Argonne, IIT, INRIA, Krell Institute, Stowers Medical Research Center
- **Sites can easily leverage efforts by sharing builds.**

- **Get Spack!**

  Github: **http://bit.ly/spack-git**

  Mailing List: **http://groups.google.com/d/spack**


Lawrence Livermore
National Laboratory

Future directions for Spack

- **Dependencies on Compiler features**
  - Profusion of new compiler features frequently causes build confusion:
    - C++11 feature support
    - OpenMP language levels
    - CUDA compute capabilities

```
require('cxx11-lambda')
require('openmp@4:')
```

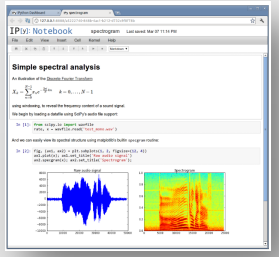
- **Automatic ABI Checking**
  - Ensure that Spack-built packages are ABI-compatible with system versions.
  - Working with RedHat to use **`libabigail`** for this
  - Have used Spack + libabigail to find ABI bugs in clang

- **Compiler wrappers for tools**
  - Spack will allow our largest codes to be used with source instrumentation
  - LLVM tools like thread sanitizers can be applied to more than toy problems.

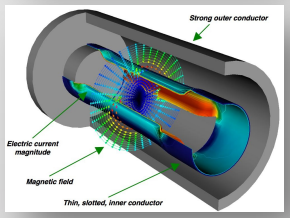
We are looking at providing secure web interfaces for other user services.

MyLC Web Portal

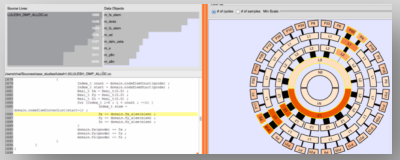
My Running Services



IPython Notebook



In-situ Physics Visualization



Performance/ debug data

- Services run securely as the owning user
- User discovers services through Lorenz.
- No need to set up servers/ connect to obscure ports

We are prototyping a system to securely connect users with running services on clusters