“Compute Canada” site talk

Compute Canada is now The Alliance (National coordinating office, non-profit funded by Government of Canada).
The Federation = The Alliance + 38 partner universities + 5 regional organizations

Bart Oldeman
McGill University, Calcul Québec, Digital Research Alliance of Canada
Research Support National Team Software Installation Coordinator
(with Maxime Boissonneault, Charles Coulombe, Doug Roberts (RSNT), Ryan Taylor (CVMFS))

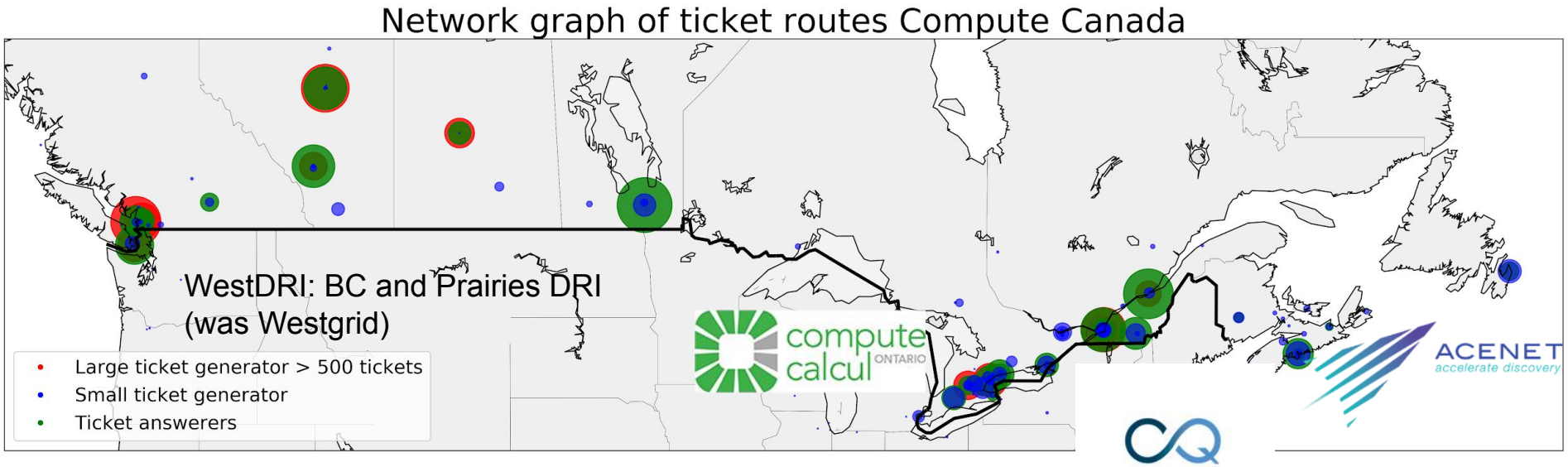
McGill

Calcul Québec

Partenaire régional de l’ Alliance de recherche numérique du Canada

A regional partner of the Digital Research Alliance of Canada

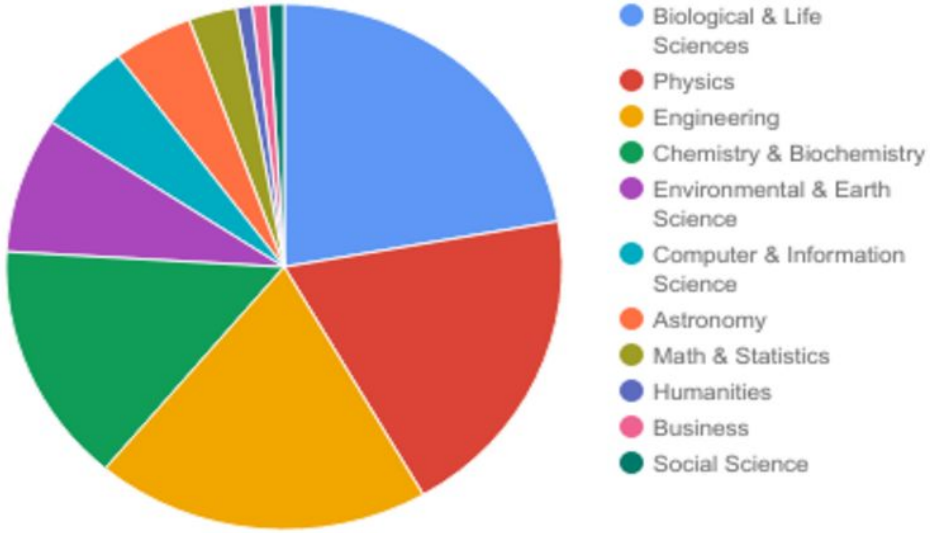
The people



All research disciplines supported

Free access for any researcher at a Canadian institution

- 5 regional consortia
- 38 member institutions
- ~250 technical staff
- ~18,000 user accounts
- 6 clusters, 4 clouds, 300k cores, 2k GPUs, 100s PB storage



The hardware

5 major national systems
300K cores, 30 PF
90 PB disk, 180 PB tape

System	Type	Network	Production
Arbutus	Cloud	10 GbE	2016 H2
Cedar	General	OPA	2017 H1
Graham	General	EDR IB	2017 H1
Niagara	Large MPI	EDR IB	2018 H1
Béluga	General	EDR IB	2019 H1
Narval	General	HDR IB	2021 H2

Starting in 2017, new bigger national systems replaced many smaller local clusters, with common software stack, scheduler (Slurm), and so on, administered by national teams.
Many sites have no physical cluster but still support.

Guiding principle

Users should be presented with an interface that is as **consistent** and as **easy to use** as possible across **all sites**. It should also offer **optimal performance.**

All sites

1. Need a distribution mechanism
    a. CVMFS

Consistency

2. Independent of the OS (Ubuntu, CentOS, Fedora, etc.)
    a. Compatibility layer: was Nix, now Gentoo Prefix
3. Automated installation (humans are not so consistent)
    a. EasyBuild

Easy to use

4. Needs a module interface that scale well
    a. Lmod with a hierarchical structure

Background

Most HPC clusters use enterprise Linux distributions for good reasons (vendor support for network, parallel filesystems, etc)

CentOS/RHEL 7

Linux kernel 3.10, GCC 4.8.5, Glibc 2.17, Python 2.7.5 (+ backports of course)

CentOS/RHEL/Rocky/... 8

Linux kernel 4.18, GCC 8.4, Glibc 2.28, Python 3.9.2 (+ backports of course)

CentOS/RHEL/Rocky/... 9

Linux kernel 5.14, GCC 11.2.1, Glibc 2.34, Python 3.9.10

compare:

Fedora 36

Linux kernel 5.17.5, GCC 12.0.1, Glibc 2.35, Python 3.10.4

Background

But users on those clusters want shiny new things and install them as if it were a local Linux computer (following documentation):

```
$ sudo apt-get install python3.9-dev
We trust you have received the usual lecture from the local
System Administrator. It usually boils down to these three
things:
    #1) Respect the privacy of others.
    #2) Think before you type.
    #3) With great power comes great responsibility.
[sudo] password for jsmith:
sudo: apt-get: command not found
$ sudo yum install python39-devel
[sudo] password for jsmith:
Sorry, try again.
[sudo] password for jsmith:
Sorry, user jsmith is not allowed to execute ...
```

Solution: modules

Create a “modulefile” named “python/2.7.9” somewhere in $MODULEPATH

```
#%Module1.0#############################
proc ModulesHelp { } {
       puts stderr "\tAdds Python 2.7.9 to your environment"
}
module-whatis   "Adds Python 2.7 to your environment"
set  root /software/CentOS-6/tools/python-2.7.9
prepend-path MANPATH         $root/share/man
prepend-path PATH            $root/bin
prepend-path LD_LIBRARY_PATH $root/lib
prepend-path CPATH           $root/include
```

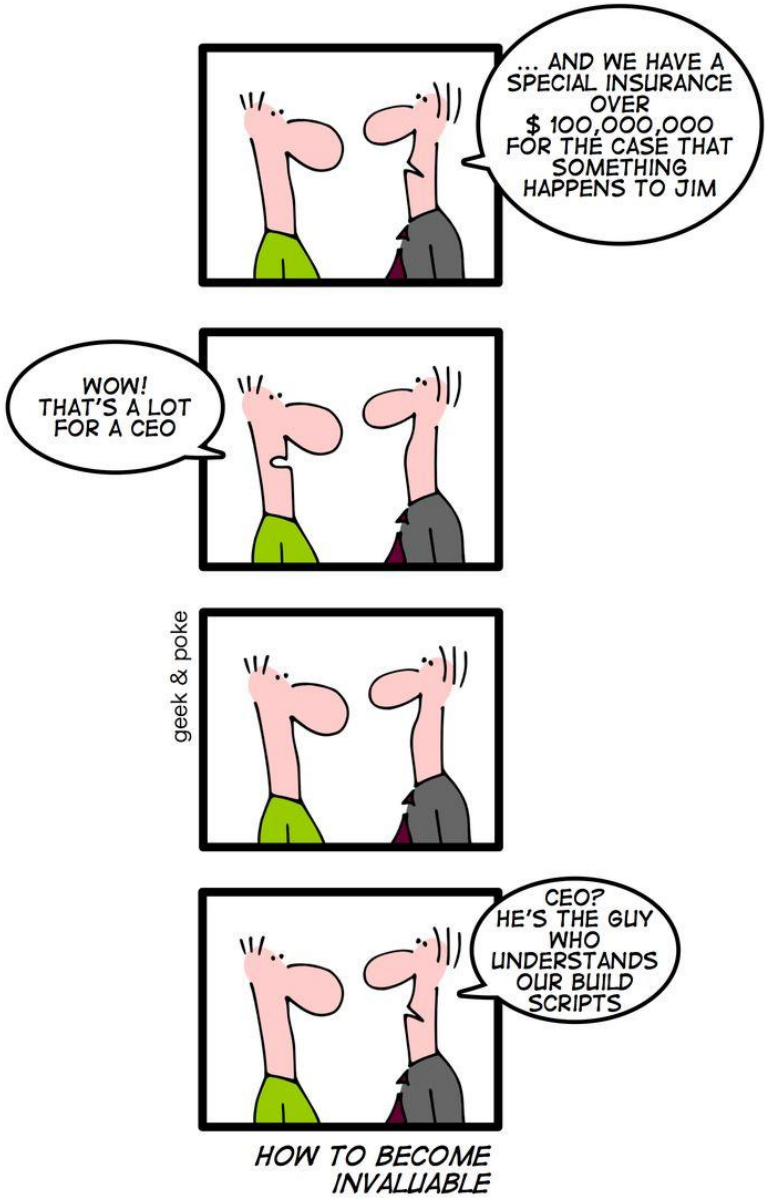
Users do “module load python/2.7.9”, which modifies their environment. “module unload python” restores it then.

Solution: modules

How were modulefiles created: by hand of course, same as how the software was installed.

How to **not** become invaluable:

https://easybuilders.github.io/easybuild/



http://geekandpoke.typepad.com/geekandpoke/2010/05/how-to-become-invaluable.html

Software: design overview

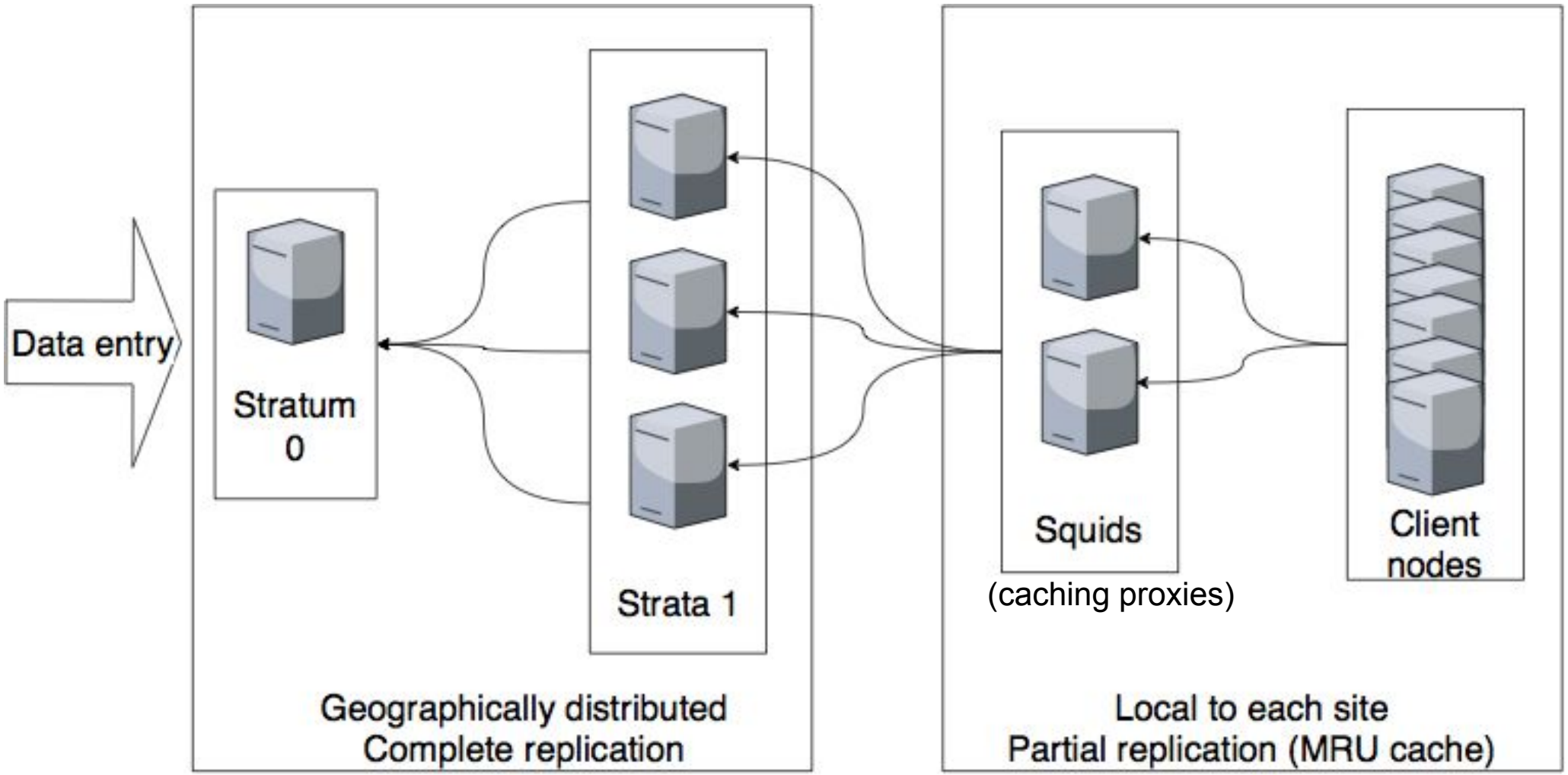
Easybuild layer: modules for Intel, NVHPC, OpenMPI, CUDA, MKL, high-level applications. Multiple architectures (sse3, avx, avx2, avx512)
`/cvmfs/soft.computecanada.ca/easybuild/{modules,software}/~~2017~~2020`

Compatibility: ~~Nix~~ Gentoo Prefix layer: GNU libc, autotools, make, bash, cat, ls, awk, grep, etc.
~~`module nixpkgs/16.09 => $NIXUSER_PROFILE=$EBROOTNIXPKGS=`~~
~~`/cvmfs/soft.computecanada.ca/nix/var/nix/profiles/16.09`~~
`module gentoo/2020 => $EPREFIX=`
`/cvmfs/soft.computecanada.ca/gentoo/2020,  $EBROOTGENTOO=$EPREFIX/usr`

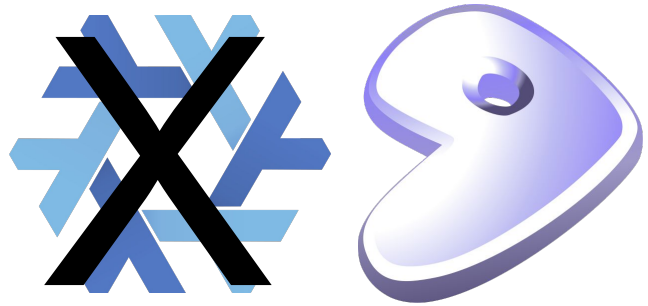
Gray area: ~~Slurm,~~ Lustre client libraries, IB/OmniPath/InfiniPath client libraries (all dependencies of OpenMPI). In Gentoo layer, but can be overridden using PATH & LD_LIBRARY_PATH.

OS kernel, daemons, drivers, libcuda, anything privileged (e.g. the sudo command): always local. Some legally restricted software too (VASP)

CVMFS content delivery



Compatibility layer: ~~Nix~~ Gentoo Prefix



- Package, dependency & environment management system
- Builds using bash-like “ebuilds”.
- Used to provide dependencies for scientific applications, themselves of little scientific interest
  - Glibc, coreutils, awk, grep, Bash, Bison, Flex, GNU Make, ncurses, readline, libxml2, zlib, bzip2, XZ, Autotools, binutils, OpenSSL, libpng, Emacs, vim, X11, texlive, etc., etc.
  - Newer versions of those than found in enterprise distributions, e.g. Bash 5.0, Git 2.31.0, Vim 8.2, Emacs 26.2
- Abstraction layer between the OS and the scientific software stack, using ~~nixpkgs/16.09~~ gentoo/2020 module
- Carries all* the dependencies of scientific software stack

* Exceptions: drivers, kernel modules, etc.

Tools used : EasyBuild

- Automates installation of (mostly) scientifically oriented software and generation of modulefiles.

Tools used : Lmod

- Lua based module system
- Makes it easy to setup a software module hierarchy
  - e.g. modules that depend on MPI implementation X are only visible if you first “modue load X”.
- https://lmod.readthedocs.io/en/latest/

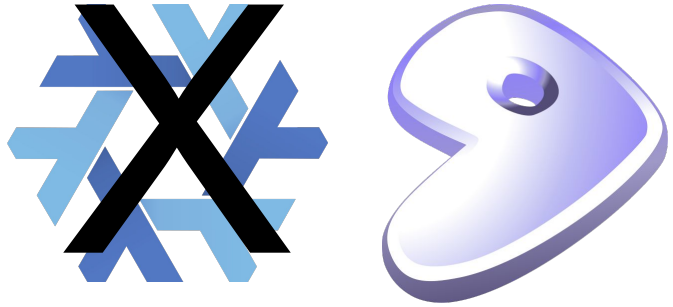
Gentoo/Nix and EasyBuild, conceptually

- Builds are performed through “recipes”
- Recipes are stored on Git. Compute Canada has its own fork of the repos :
  - Nixpkgs / Gentoo Overlay
  - Easybuild:
    - framework (high level Python scripts)
    - easyblocks
      - is it configure; make; make install, cmake, custom? (Python scripts)
    - easyconfigs
      - what are the configure parameters? (configuration files)

Installing software, step by step

1. Figure out if it should be in Gentoo or EasyBuild
    - Is the software performance critical or depends on MPI?
        - Yes => EasyBuild
        - Multiple versions needed via modules ?
            - Yes => EasyBuild
            - No => Gentoo
2. Install on build-node.computecanada.ca with the appropriate package manager, Portage (emerge) or eb: plain `eb` installs in home dir, then with `sudo -iu ebuser`
3. Test on build-node.computecanada.ca
4. Deploy on CVMFS dev repository
5. Test on cvmfs-client-dev.computecanada.ca or with `proot`
6. Deploy on CVMFS production repository
7. Final testing on the production cluster

Why Gentoo instead of Nix?



We used a single read/only Nix environment:

```
.../nix/var/nix/profiles/16.09 ->

.../nix/var/nix/profiles/16.09-523-link ->
.../nix/store/cj3f56cgpms7m9fjnbl9vjkmap5fzgsi-user-environment

.../nix/store/cj3f56cgpms7m9fjnbl9vjkmap5fzgsi-user-environment/bin/ls ->
.../nix/store/cn222k5axppndcfbqlckj57939d9h0h9-coreutils-8.25/bin/ls
```

We wrapped the linker (ld) so only $NIXUSER_PROFILE/lib was used.

Nix components can be upgraded, which changes the store hashes, and allows garbage collect / selective copying.

Sometimes store hashes would “leak” into EasyBuild-compiled software anyway, via cmake, qmake or Python virtualenv, and garbage collect was destructive.

- Nix is better used as a top layer with writable store directories
- But HPC users are familiar with environment modules, not Nix’ tools.

Gentoo Prefix : no symlinks, no store leak, minimal solution

Python wheels

What are wheels?

Wheels are the new standard of Python distribution and are intended to replace eggs. Support is offered in pip >= 1.4 and setuptools >= 0.8.

Advantages of wheels

1. Faster installation for pure Python and native C extension packages.
2. Avoids arbitrary code execution for installation. (Avoids setup.py)
3. Installation of a C extension does not require a compiler on Linux, Windows or macOS.
4. Allows better caching for testing and continuous integration.
5. Creates .pyc files as part of installation to ensure they match the Python interpreter used.
6. More consistent installs across platforms and machines.
7. **You can compile your own wheels, linking against your compiled libraries**

Our supported wheels

```
$ ls /cvmfs/soft.computecanada.ca/custom/python/wheelhouse/*/* | wc -w
15543
$ avail_wheels tensorflow
name        version    python  arch
----------  ---------  --------  -------
tensorflow  2.9.0      cp39    generic
tensorflow  2.9.0      cp38    generic
tensorflow  2.9.0      cp310   generic
```

- https://docs.computecanada.ca/wiki/Available_wheels

“Compute Canada” Software Stack

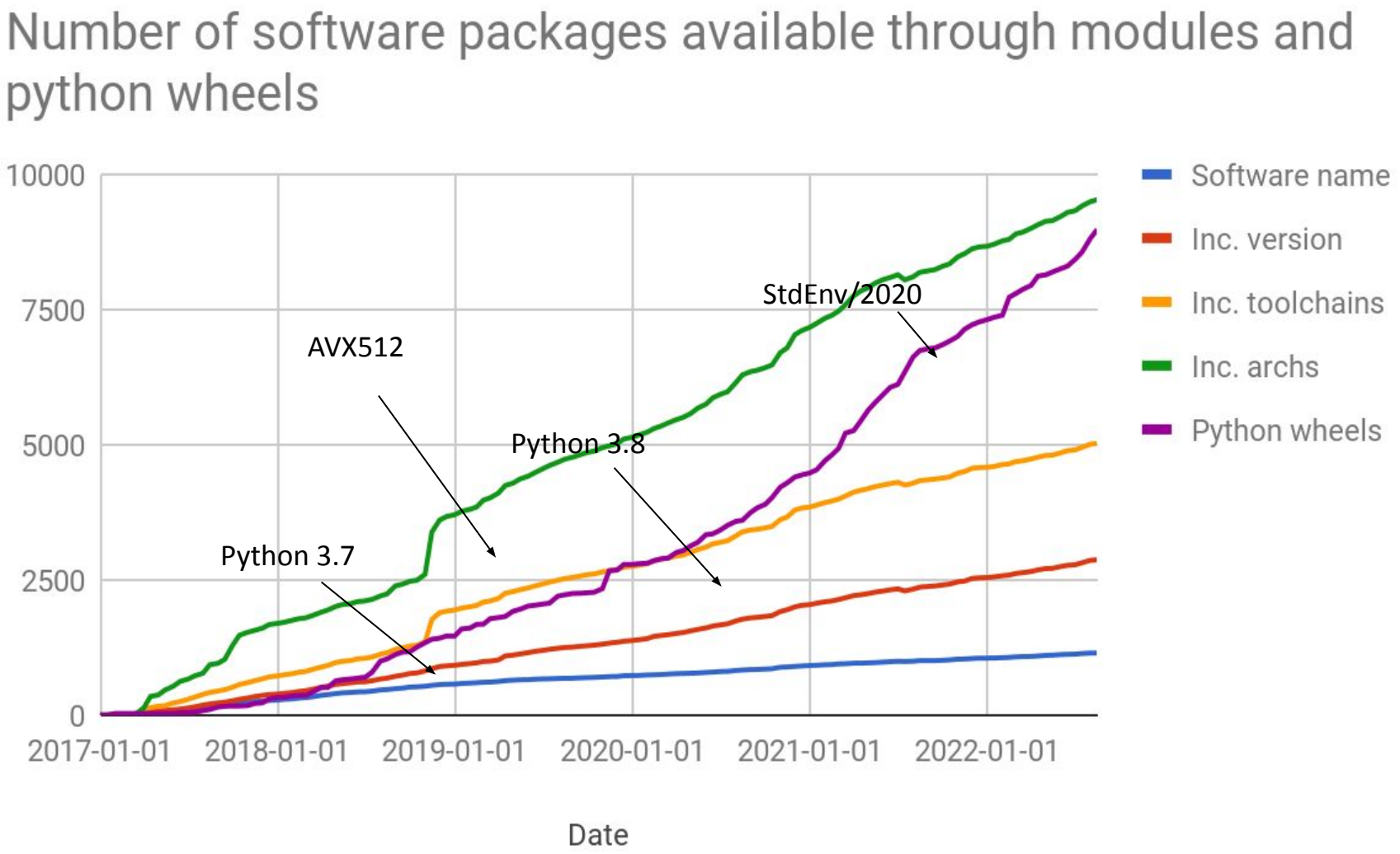
~1000 scientific applications

9500+ permutations of version/CPU/toolchain

Optimized for

- 4 major generations of CPUs (from early 2000s to recent CPUs in 2020)
- 4 major generations of NVidia GPUs
- InfiniBand, OmniPath, Ethernet

15000+ python wheels



War story:

https://sft.its.cern.ch/jira/browse/CVM-2001

May 2021: In-place update of glibc caused widespread corruption.

- Started as fairly innocent looking change to enable `memusage` and `memusagestat` for memory profiling.
- After pushing `$EPREFIX/lib64/libc-2.30.so` into cvmfs, processes already started randomly crashing (they all mmap this file). Newly started programs on clean nodes were fine.
- Reason: “cache poisoning”
- sha256sum would give different results every time you ran it.
- Similar things happen when you “cp” a shared library, but not if you unlink it first and then place a new one on a local file system (two different inodes)
- In-place updates work fine if you use symbolic links (files or directories) instead.
- Fixed in CVMFS git 5 September 2022 (https://github.com/cvmfs/cvmfs/pull/3043)

Software challenges caused by custom prefix

Using `/cvmfs/soft.computecanada.ca/gentoo/2020/usr` instead of `/usr`

- `Uses Gentoo Prefix` custom loader (ELF interpreter in `$EPREFIX/lib/ld-linux-x86-64.so.2`)
- Custom `setrpaths.sh` script patches (using `patchelf --set-interpreter`) downloaded binaries so they can work with this prefix
- Some users set `LD_LIBRARY_PATH` to `/usr/lib64`, mostly by accident in old `.bashrc` files, which breaks most tools
- Some commercial packages use wrapper scripts which needed to patched
- Anaconda & Julia
  - provide binaries that are not always compatible
  - we provide pip-installable Python wheels and actively discourage Anaconda
    - Anaconda users can even end up with custom, non-optimal, installations of Open MPI or R.
- If all else fails, use `module --force purge`, or Singularity/Apptainer.

Challenge: host OS/compatibility layer boundary

- Various host OS daemons write to files under /var such as /var/run/utmp, but compatibility layer utilities (`last`, `w`, `who`, etc.) read from $EPREFIX/var.
- But (for example) Gentoo Prefix' `$EPREFIX/usr/include/paths.h` sets `_PATH_UTMP` to `$EPREFIX/var/run/utmp`. who then reads from that file. Two solutions:
  - strategic symlinks, e.g. `$EPREFIX/var/run -> /var/run` (CC)
  - change `paths.h`
- Rest works remarkably well:
  - Gentoo provides elogind as lightweight alternative to systemd
  - PAM libraries work fine
  - `libnss_ldap/sss` libraries can be compiled from source, also read from locations under `/var` (symlinks for libraries cause issues if host OS is newer than compat)
  - Whole Mate desktop can be compiled and works with VNC (includes udev but udisks (storage drive monitor, e.g. USB keys) stripped out.
  - Even some selinux support (for filesystem labels).

Challenge: compatibility/software layer boundary

- Grey area, since Gentoo provides a lot of software already
- In general: needs multiple versions / MPI / scientifically oriented -> EasyBuild
- Many basic EasyBuild dependencies go to filter-deps:
  - EESSI: 21 easyconfigs:
    `Autoconf,Automake,Autotools,binutils,bzip2,cURL,DBus,flex,gettext,gperf,help2man,intltool,libreadline,libtool,Lua,M4,makeinfo,ncurses,util-linux,XZ,zlib`
  - CC: 96 easyconfigs (+ more tools e.g. `mate,texlive,gdb,strace,mc,tmux,pandoc`)
    `ASSIMP,Autoconf,Automake,Autotools,binutils,Bison,bzip2,cairo,Check,CMake=:3.16.5[,cURL,DBus,Doxygen,expat=:2.2.5[,FFmpeg,file,FLANN,flex,FLTK,fontconfig,FreeImage,freetype,FriBidi,gettext,Ghostscript,giflib,git,GL2PS,GLib,GMP,gperf,GraphicsMagick,GST-plugins-base,GTK+,gzip,hwloc=:2.4.0[,ImageMagick,LAME,libarchive,libdrm,libedit,libevent,libGLU,libiconv,libjpeg-turbo,libmatheval,libpciaccess,libpng,libreadline,libsndfile,LibTIFF,libtirpc,libtool,libunwind,libwebp,libX11,libXext,libXft,libxml2,libXpm,libXt,Lua,M4,makedepend,Mesa,MPFR,NASM,ncurses,numactl,OpenSSL,Pango,PCRE,pkg-config,pkgconf,SDL2,SQLite=:3.36[,SWIG=:3.0.12],Szip,Tcl,tcsh,texinfo,time,Tk,Tkinter,UnZip,util-linux,X11,x264,x265,XZ,Zip,zlib,zstd`

Other differences

- Some differences in Gentoo configuration: EESSI uses sets, CC a custom profile
- EESSI: aarch, x86_64, ppc64, risc-v in progress, CC: x86_64 subarch only (avx512, avx2, avx, sse3).
- EESSI: flat module naming scheme, mixed case, CC: hierarchical, lower case
- EESSI: more automated (Ansible)
  CC: `sudo -iu ebuser eb xxx`, `sudo -iu gentoouser emerge xxx` done manually.
- CC uses central libstdc++ (+libgfortran etc) at runtime from compat layer, presently:
  `/cvmfs/soft.computecanada.ca/gentoo/2020/usr/lib/gcc/x86_64-pc-linux-gnu/11.3.0/libstdc++.so.6`
  - Adding newer GCCcore installation via EasyBuild necessitates adding newer GCC in Gentoo as well (multiple can be installed in parallel)
  - Advantages
    - we can collapse GCCcore EasyBuilds for different versions to “SYSTEM”
      - `foo-1.0-GCCcore-11.3.0.eb` -> `foo-1.0.eb` in `ebfiles_repo` via hook.
      - Large repository of compatible modules at Core level.
    - Python wheels compiled with g++11 compatible with compiled-with-g++9

RPATH difference

- EESSI: uses EasyBuild's RPATH functionality, CC: uses ld (linker) wrapper
  - neither set LD_LIBRARY_PATH in modules
    - EESSI doesn't work for regular user `gcc -lxxx foo.c`

```
[EESSI pilot 2021.12] $ module load libffi
[EESSI pilot 2021.12] $ echo "int main(){return 0;}" > foo.c
[EESSI pilot 2021.12] $ gcc -lffi foo.c
[EESSI pilot 2021.12] $ ./a.out
./a.out: error while loading shared libraries: libffi.so.7: cannot open shared object file: No such file or directory
```

  - Possible alternative solution… next slide

Promising RPATH alternative

- Harmen Stoppels: https://stoppels.ch/2022/08/04/stop-searching-for-shared-libraries.html
- Idea: put absolute path in soname in library, regular linking inherits it in `needed` entry.

```
$ gcc -shared -o libf.so -x c -Wl,-soname,$PWD/libf.so - <<EOF
#include <stdio.h>
void f() { puts("hello world"); }
EOF

$ gcc -o main -x c - -L. -lf <<EOF
void f();
int main() { f(); }
EOF

$ ./main
hello world

$ patchelf --print-needed main
/tmp/hello/libf.so
libc.so.6
```

- Removes need for RPATH, wrapper, just adjust soname in shared libraries.
- https://github.com/spack/spack/pull/31948

Opportunities, collaboration

- Public part of the stack is available *everywhere*
  - https://docs.alliancecan.ca/wiki/Accessing_CVMFS
  - `source /cvmfs/soft.computecanada.ca/config/profile/bash.sh`
  - Proprietary packages, e.g. Intel compilers, MATLAB, in restricted repositories
- Could share Compatibility Layer (Gentoo Prefix Ansible-based bootstrap) with EESSI, just needs a different larger set + some USE flag configuration.