Evaluating SPACK for CMS offline Software

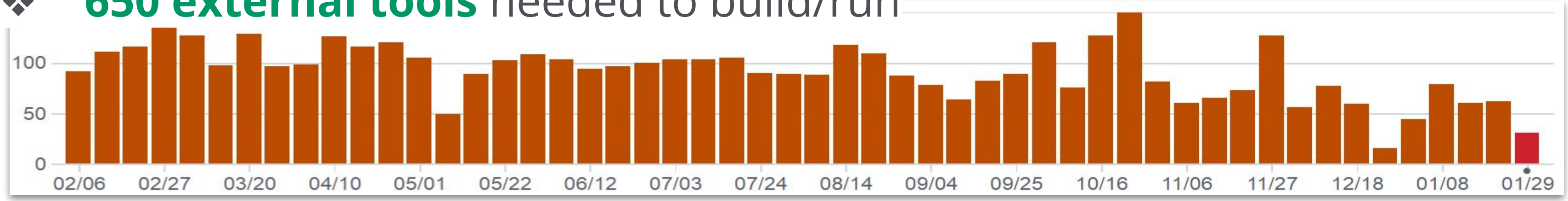
SFT Group Meeting

13th Mar 2023

CMS Software Stack

CMS Offline Software Stack

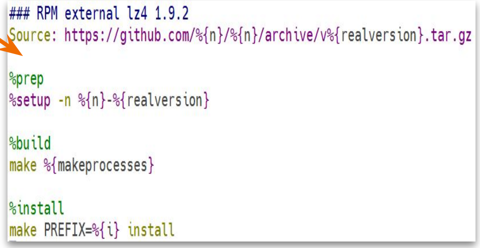
- **CMSSW:** Software and services needed by the simulation, calibration and alignment, and reconstruction
  - **5.5M** code lines:
    - **66% C/C++**, **27% Python**, **5% fortran** and rest are build-rules/data files
  - **17 active release cycles: 5.3, 8.0, 9.4 up to 13.0/13.1**
    - 13.1.X development release cycle has 15 flavors
      - **ROOT, GEANT, CLANG, LTO, Multi-Vectorization, ASAN, UBSAN** etc.
      - **Build for multiple OS/Arch/Compilers (cc7/el8/el9, x86_64/arm/power…)**
  - Build production: **2.5K shared libs/plugins, 1K binaries**
- **650 external tools** needed to build/run



CMS Offline Software Build/Packaging System

- **CMS** Offline Software Build/Packaging system consists of various components
  - **SCRAM**: Software Configuration, Release and Management tool
  - **CMSSW-Config**: Build rules based on gmake
  - **PKGTOOLS**: Packaging system based on Redhat Package Manager (RPM)
  - **CMSPKG**: Software distribution and Installation/Deletion tool
  - **CMSDIST**: The build recipes for building the package
- All of these are though customized for CMS offline SW but can be used by other projects
  - **LCG Projects** were using **SCRAM** in past to build **POOL/CORAL**.
  - CMS still builds copy of **CORAL** using **SCRAM**

More details in backup slides



```
### RPM external lz4 1.9.2
Source: https://github.com/%{n}/%{n}/archive/v%{realversion}.tar.gz

%prep
%setup -n %{n}-%{realversion}

%build
make %{makeprocesses}

%install
make PREFIX=%{i} install
```

CMS Package System: Summary

- **All tools used for building/distribution/installation are independent of CMSSW releases/OS/archs/compilers**
  - **Except for CMSDIST** where though most of build recipes are shared between releases/OS/archs/compilers but we have separate git branches for each release/compiler
  - Build recipes do not depend of packaging/distribution tools so can be easily backported to older release cycles
- **All tools are very stable and require low maintenance**
  - Few commits per year (mostly for additional features)
  - **Over all 40K lines of code to maintain**
    - **20K lines are stable and independent of CMSSW releases/OS/archs/compilers**
    - **20K lines of build recipes gets most of the changes and mostly for development release cycles**

SPACK

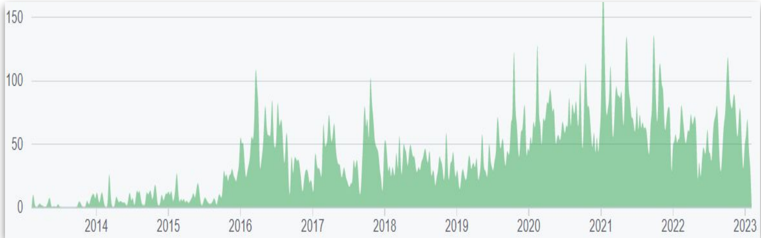
Let’s dive into SPACK world

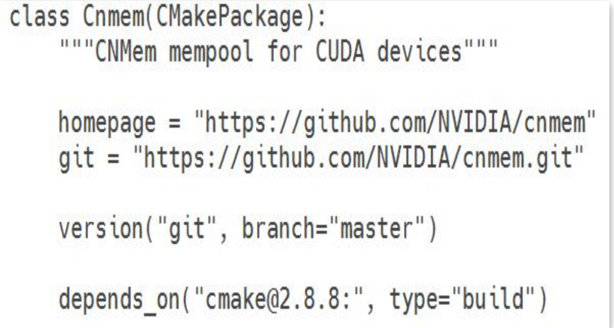
SPACK

- **SPACK flexible package manager for HPC software**
  - A lot of development for/from **HEP** experiments in last 6 years
  - Supports multiple version and configurations of software
  - Single package file to have all supported versions, configurations and variants of the software
- **Just not a package manager but supports software distribution and installation**
  - Binary relocatable packages
- **PKGTOOLS, CMSPKG and CMSDIST all in one**
  - Also covers part of **SCRAM** e.g package env setup (***spack load/unload***)
  - Interferes with build rules too via its compiler wrappers

Why we looked in to SPACK

- Large community specially after 2016 when **HEP** exp start looking in to it
- **HSF recommends SPACK for HEP community**
  - **LCG/SPI** team looking in to it since 2020
- **+6.5K built-in package recipes**
- **Out of the box RPATH/RUNPATH builds**
  - No LD_LIBRARY_PATH required at runtime
- **Python based simple package recipes**
- **Ivan Razumov, joined CMS in May 2021**
  - **1.5y** of experience building LCG packages w/ SPACK
  - A very active contributor of SPACK (**>700 PRs**)

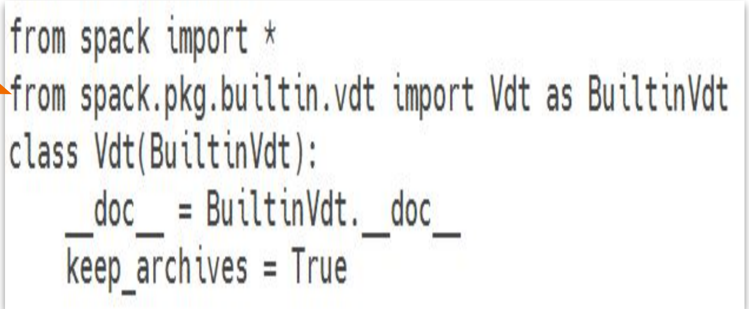




```
class Cnmem(CMakePackage):
    """CNMem mempool for CUDA devices"""

    homepage = "https://github.com/NVIDIA/cnmem"
    git = "https://github.com/NVIDIA/cnmem.git"

    version("git", branch="master")

    depends_on("cmake@2.8.8:", type="build")
```

Building CMS Software stack using SPACK

- After 22 months long roller coaster ride, Ivan finally had managed to build **CMSSW** externals software stack
  - Had to rewrite package recipes multiple time due to fast moving **SPACK** development
    - **v0.16, v0.17, v0.18**
  - Final implementation uses inheritance
    - **SPACK v0.19.0**



```
from spack import *
from spack.pkg.builtin.vdt import Vdt as BuiltinVdt
class Vdt(BuiltinVdt):
    __doc__ = BuiltinVdt.__doc__
    keep_archives = True
```

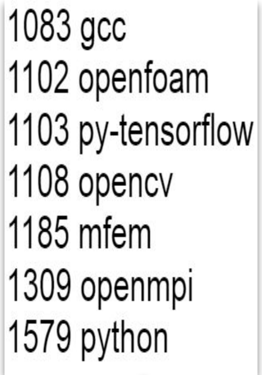
- **CMS SPACK env contains 650 packages**
  - **350** package recipes from upstream **SPACK**
  - **300** in cms-spack repository
    - **100** CMS specific (including +80 cms-data)
    - **140** use built-in package with CMS specific patches/sources/changes
    - **60** recipes are rewritten (mostly copied and changed according to CMS needs)

SPACK

Few major design/scalability issues

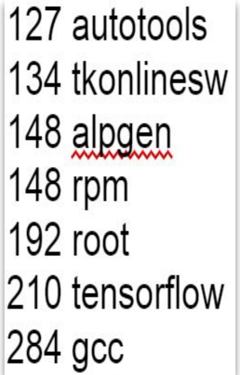
Package recipes: are they really simple … ?

- Yes they are simple/readable for first version, single configuration
- **SPACK package recipes contain all versions/configurations/variants/build systems in single file**
  - Over the time these recipes become a nightmare to manage specially when you had to backport changes to few years old software stack
    - **LZ4:** **10** to **50+** lines
    - **ROOT:** **60** to **650+** lines
    - **Python: 25** to **1.5K** lines
    - **Boost:** **60** to **740** lines
  - Package recipe grows only (unless someone does a cleanup)



1083	gcc
1102	openfoam
1103	py-tensorflow
1108	opencv
1185	mfem
1309	openmpi
1579	python

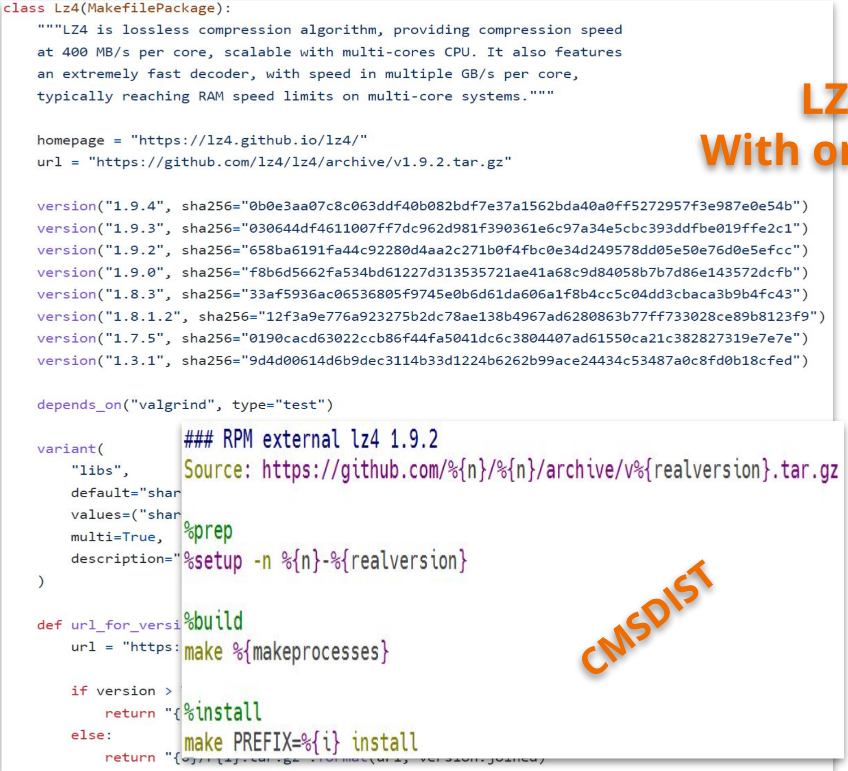
spack



127	autotools
134	tkonlinesw
148	alpgen
148	rpm
192	root
210	tensorflow
284	gcc

cmsdist

Package recipe: LZ4 package

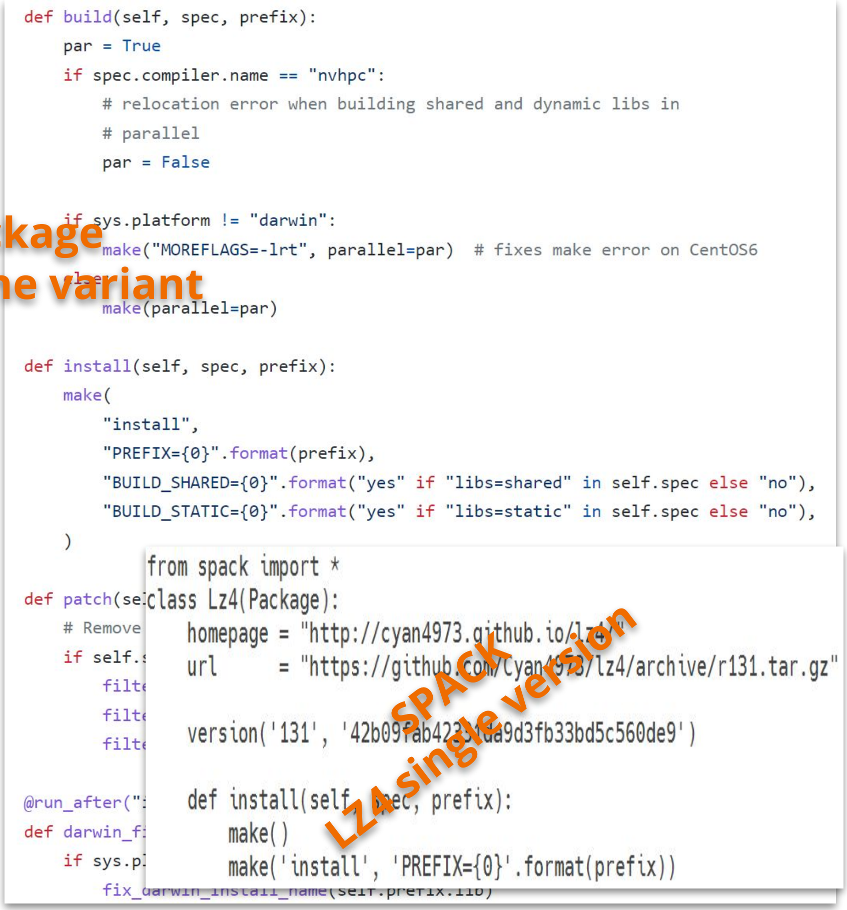


```
class Lz4(MakefilePackage):
    """LZ4 is lossless compression algorithm, providing compression speed
    at 400 MB/s per core, scalable with multi-cores CPU. It also features
    an extremely fast decoder, with speed in multiple GB/s per core,
    typically reaching RAM speed limits on multi-core systems."""

    homepage = "https://lz4.github.io/lz4/"
    url = "https://github.com/lz4/lz4/archive/v1.9.2.tar.gz"

    version("1.9.4", sha256="0b0e3aa07c8c063ddf40b082bdf7e37a1562bda40a0ff5272957f3e987e0e54b")
    version("1.9.3", sha256="030644df4611007ff7dc962d981f390361e6c97a34e5cbc393ddfbe019ffe2c1")
    version("1.9.2", sha256="658ba6191fa44c92280d4aa2c271b0f4fbc0e34d249578dd05e50e76d0e5efcc")
    version("1.9.0", sha256="f8b6d5662fa534bd61227d313535721ae41a68c9d84058b7b7d86e143572dcfb")
    version("1.8.3", sha256="33af5936ac06536805f9745e0b6d61da606a1f8b4cc5c04dd3cbaca3b9b4fc43")
    version("1.8.1.2", sha256="12f3a9e776a923275b2dc78ae138b4967ad6280863b77ff733028ce89b8123f9")
    version("1.7.5", sha256="0190cacd63022ccb86f44fa5041dc6c3804407ad61550ca21c382827319e7e7e")
    version("1.3.1", sha256="9d4d00614d6b9dec3114b33d1224b6262b99ace24434c53487a0c8fd0b18cfed")

    depends_on("valgrind", type="test")

    variant(
        "libs",
        default="shar
        values=("shar
        multi=True,
        description="
    )

    def url_for_versi
        url = "https:

        if version >
            return "{
        else:
            return "{0}/r{1}.tar.gz".format(url, version.joined)
```

```
### RPM external lz4 1.9.2
Source: https://github.com/%{n}/%{n}/archive/v%{realversion}.tar.gz

%prep
%setup -n %{n}-%{realversion}

%build
make %{makeprocesses}

%install
make PREFIX=%{i} install
```

CMSDIST

LZ4 package
With only one variant



```
def build(self, spec, prefix):
    par = True
    if spec.compiler.name == "nvhpc":
        # relocation error when building shared and dynamic libs in
        # parallel
        par = False

    if sys.platform != "darwin":
        make("MOREFLAGS=-lrt", parallel=par)  # fixes make error on CentOS6
    else:
        make(parallel=par)

def install(self, spec, prefix):
    make(
        "install",
        "PREFIX={0}".format(prefix),
        "BUILD_SHARED={0}".format("yes" if "libs=shared" in self.spec else "no"),
        "BUILD_STATIC={0}".format("yes" if "libs=static" in self.spec else "no"),
    )

def patch(se
    # Remove
    if self.s
        filte
        filte
        filte

@run_after("
def darwin_f
    if sys.pl
        fix_darwin_install_name(self.prefix.lib)
```

```
from spack import *
class Lz4(Package):
    homepage = "http://cyan4973.github.io/lz4/"
    url      = "https://github.com/Cyan4973/lz4/archive/r131.tar.gz"

    version('131', '42b09fab42331da9d3fb33bd5c560de9')

    def install(self, spec, prefix):
        make()
        make('install', 'PREFIX={0}'.format(prefix))
```

SPACK
LZ4 single version

Compiler Wrappers

- **SPACK** provide compiler wrappers (**800 lines of bash script**) which are added in to **PATH** before the build process
- Compiler wrappers are the **heart and soul of SPACK** and are the magic behind
  - Injecting compilation flags
    - **RPATH/RUNPATH**
    - ***-Idirs*** for dependencies during building phase
    - ***-Ldirs*** for dependencies during link phase
    - Common flags: enabling debug mode, optimization etc.
  - Achieving the simple package recipes
    - Many packages recipes have misused this feature and do not pass the required parameters to build system (autoconf/make/cmake)
- **Disabling include/library directories injection shows that many packages either failed to build or pick up system packages**
  - My test only checked single version/configuration/variant

Compiler Wrappers: Injection of includes paths

- ❖ **Build System generated command**

```
> compiler options  -{I|isystem}{include_dirs|system_dirs} options
```

- ❖ **SPACK's compiler wrapper generated command**

```
> compiler options  -I{include_dirs} -isystem{include_dirs} \
    -I{SPACK_INCLUDE_DIRS}|-isystem{SPACK_INCLUDE_DIRS} \
    -I{system_dirs} -isystem{system_dirs} options
```

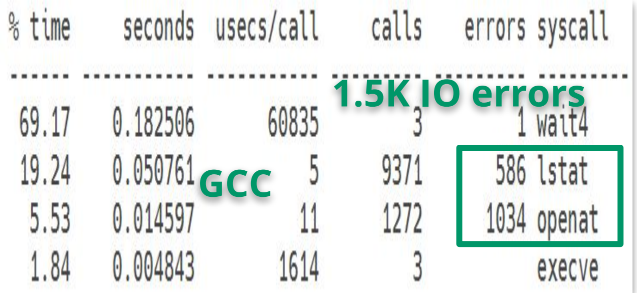
Extra include paths of package dependencies

- ❖ **Overall 25% increase in cmssw build time** (**2h25** vs **3h15**)
  - ➢ **16 Cores VM; building everything on SSD storage**

Compiler Wrappers: Injecting -I/-L…

- ❖ **CMSSW depends on large number of external packages**
  - ➢ **200 of these have include directories**
  - ➢ **325 have lib/lib64 directories**
- ❖ Injecting over 200 *-Idirs* means compiler has to go through all these to find system/compiler headers
  - ➢ A lot of IO operations (specially if you are taking most of the externals from CVMFS or shared file system)

Strace: compiling a source file with just two includes i.e iostream/string

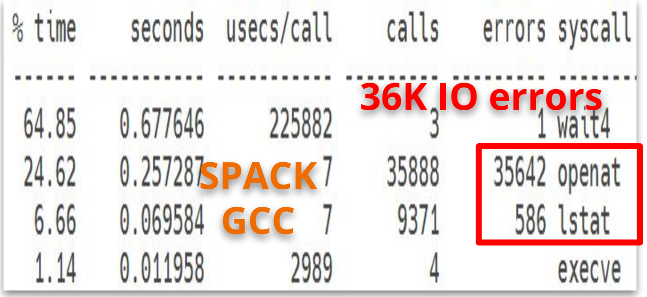


```
% time     seconds  usecs/call     calls    errors syscall
------ ----------- ----------- --------- --------- ---------
 69.17    0.182506       60835         3         1 wait4
 19.24    0.050761           5      9371       586 lstat
  5.53    0.014597          11      1272      1034 openat
  1.84    0.004843        1614         3           execve
```

1.5K IO errors

GCC

For actual CMSSW sources strace reported 100K-230K IO errors (see backup slides)



```
% time     seconds  usecs/call     calls    errors syscall
------ ----------- ----------- --------- --------- ---------
 64.85    0.677646      225882         3         1 wait4
 24.62    0.25728            7     35888     35642 openat
  6.66    0.069584           7      9371       586 lstat
  1.14    0.011958        2989         4           execve
```

36K IO errors

SPACK GCC

Compiler Wrappers: Injecting -Idirs/-Ldirs ...

- **Extra -Idirs can override some system/compiler header**
  - **CMSSW** failed to build as one of external package provide a conflicting system header
    - There could be more externals doing this
  - Multiple packages providing the same header file can cause build/runtime issues
- **Adding -Idirs/-Ldirs does not guarantee that you pick up externals from SPACK build**
  - Many packages now a days bundle/download/build packages internally
    - **Tensorflow** does it for tons of externals
    - **ONNXtime** does the same
    - **Root** also has many internal built-in packages
  - You either need to patch or configure the package to use your external or make sure that external package versions are identical (with same patches applied)

Compiler Wrappers: Debugging long commands

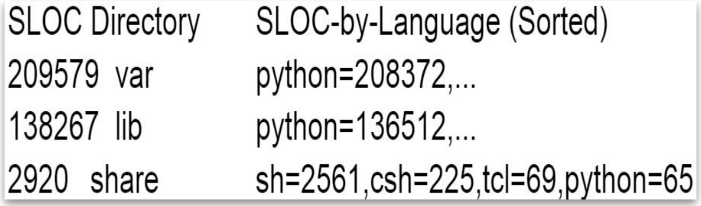
CMSSW compile/link command length comparison

gcc / spack	Compile	Link
Smallest	670 / 41K	2K / 165K
Longest	+40K 12K / 53K	+160K 44K / 207K

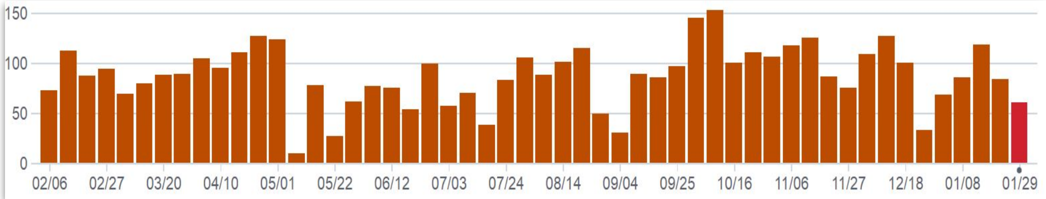
- **Long compile/link commands**
  - Package build logs only contain compile/link commands generated by build system
    - Extra flags/optins added by spack are not visible
- **Really hard for developers of the package to debug the issue**
  - Developers use their build system to build a package.
    - They are only interested to know which configure and build options were used

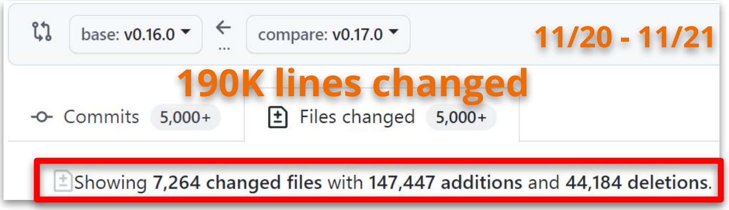
SPACK code base

- ❖ **Very active projects**
  - ➢ Gets many updates/fixes every week (avg **100 commits/week**)
- ❖ **345K** lines of python code
  - ➢ **135K in core spack lib**



SLOC	Directory	SLOC-by-Language (Sorted)
209579	var	python=208372,...
138267	lib	python=136512,...
2920	share	sh=2561,csh=225,tcl=69,python=65

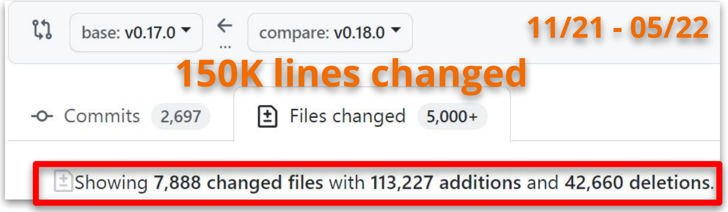




11/20 - 11/21

190K lines changed

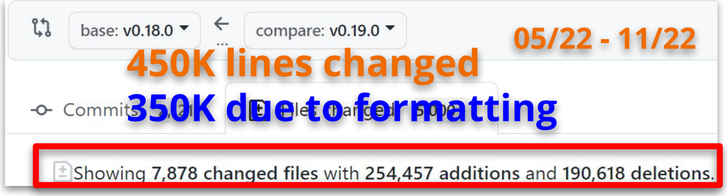
Showing **7,264 changed files** with **147,447 additions** and **44,184 deletions**.



11/21 - 05/22

150K lines changed

Showing **7,888 changed files** with **113,227 additions** and **42,660 deletions**.

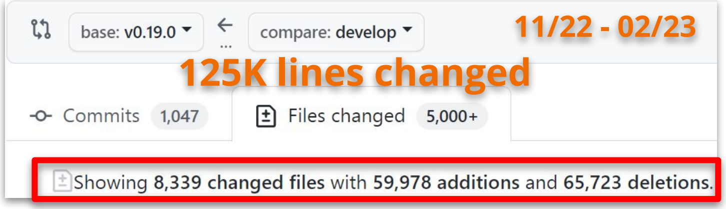


05/22 - 11/22

450K lines changed

350K due to formatting

Showing **7,878 changed files** with **254,457 additions** and **190,618 deletions**.



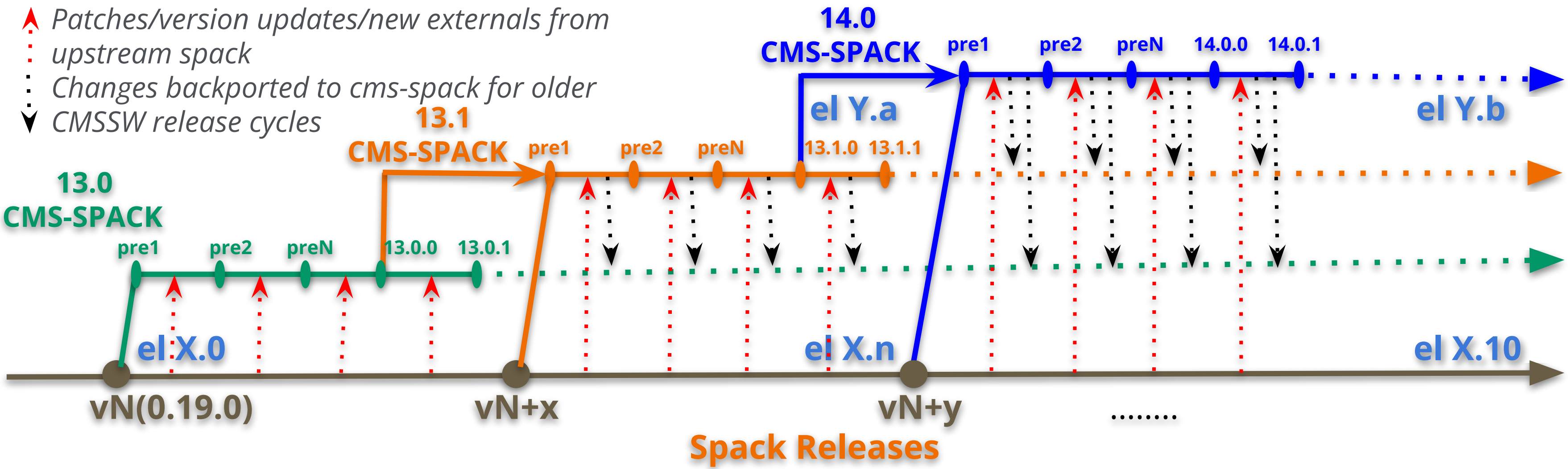
11/22 - 02/23

125K lines changed

Showing **8,339 changed files** with **59,978 additions** and **65,723 deletions**.

SPACK in large scale projects

- SPACK is moving target, so one can not use head of it
  - Projects needs to start with a tagged version and then ….
  - Rest of the life of your release you have to maintain that SPACK code base

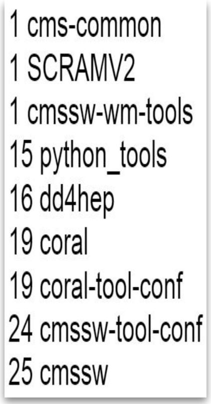


Software stack install time

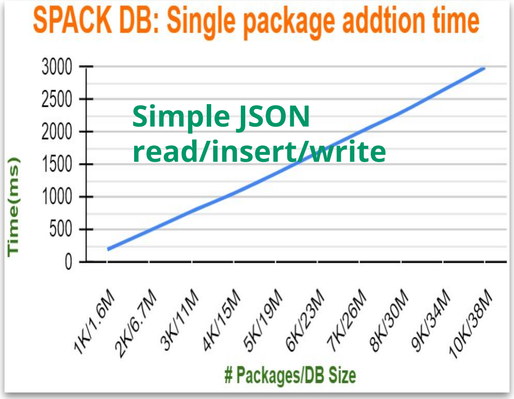
- **cmspkg takes 15mins to deploy full CMSSW software stack**
  - CMSSW + dependencies (including bootstrap, GCC)
- **SPACK takes 2hours to install CMS software stack (without GCC)**
  - Installs one package at a time
  - For simple packages (with nothing to relocate) it takes (on average) 5-6s to download/install
    - For 650 packages it is already 1 hour
  - For complex packages (with a lot of libraries/binaries) the install time is very high
    - **CMSSW with over 4K binary products took 30 mins to install**
  - It can easily add, at least, additional couple of hours for Pull Request testing

SPACK DB

- SPACK maintains a json file as DB to keep track of installed packages
  - Single package insertion at a time
- **DB Insert time linearly increases based on DB size**
- CMS Software for unique combination of OS/arch/Compiler goes under single install path
  - **slc7_amd64_gcc900: 8K packages in 2.5y**
  - **el8_amd64_gcc11: 2.1K packages in 9m**
- **Install time will only increase with time**
- What if SPACK decided to change DB structure
  - Sqlite implementation #34655
  - **We might not be able to use two different SPACK versions to install under same path**



1 cms-common
1 SCRAMV2
1 cmssw-wm-tools
15 python_tools
16 dd4hep
19 coral
19 coral-tool-conf
24 cmssw-tool-conf
25 cmssw

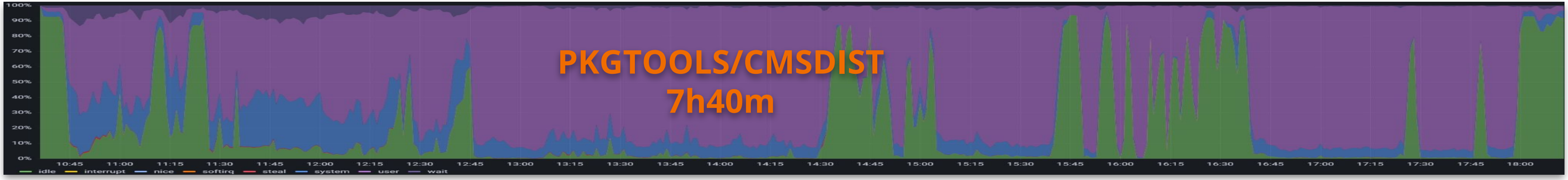


Simple JSON read/insert/write

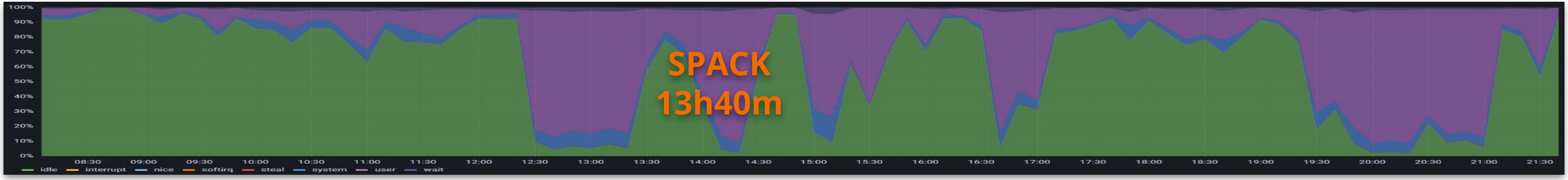
SPACK

Some minor issues which can be fixed in future releases

CMS Software stack: Build time w/o GCC and CMSSW



PKGTOOLS/CMSDIST
7h40m

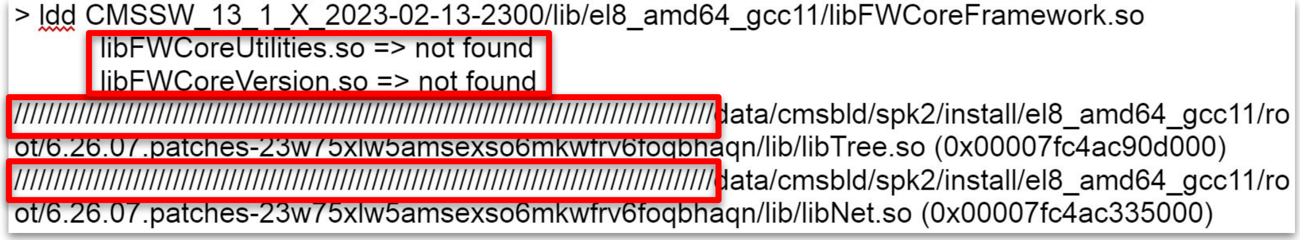


SPACK
13h40m

- Test were done on 16 Cores VM with 120GB memory with SSD
- SPACK builds single package at a time: Larger the machine bigger the waste
  - Parallel build support might improve but I was not able to tests it

Software stack install time: Parallel install

- **SPACK parallel build/install fix applied**
  - Updated SPACK DB lock wait time to 60s (instead of 3s)
- **SPACK install with -j16 failed**
  - For some unknown reason it start rebuilding packages
- **SPACK install with -j2 worked but took more time than serial mode**
  - 2h40m to install cmssw + dependencies (without GCC)
    - 30mins to install CMSSW alone
    - Might take more time if cmssw was built with runpath



```
> ldd CMSSW_13_1_X_2023-02-13-2300/lib/el8_amd64_gcc11/libFWCoreFramework.so
        libFWCoreUtilities.so => not found
        libFWCoreVersion.so => not found
/////////////////////////////////////////////////////////////////////////////////////////data/cmsbld/spk2/install/el8_amd64_gcc11/ro
ot/6.26.07.patches-23w75xlw5amsexso6mkwfrv6foqbhaqn/lib/libTree.so (0x00007fc4ac90d000)
/////////////////////////////////////////////////////////////////////////////////////////data/cmsbld/spk2/install/el8_amd64_gcc11/ro
ot/6.26.07.patches-23w75xlw5amsexso6mkwfrv6foqbhaqn/lib/libNet.so (0x00007fc4ac335000)
```

Environment setup

- **SPACK** provides various ways to use an installed package
  - **spack load/unload**
  - **spack environment**
    - Bundle many related packages and activate all of them at once
  - **spack environment modules**
    - SPACK generates module files which one can use with **module load/purge**
- All of these failed for large software stack
  - For large software stacks, the generated env is so large that it **exceeds the max argument size (2MB)**

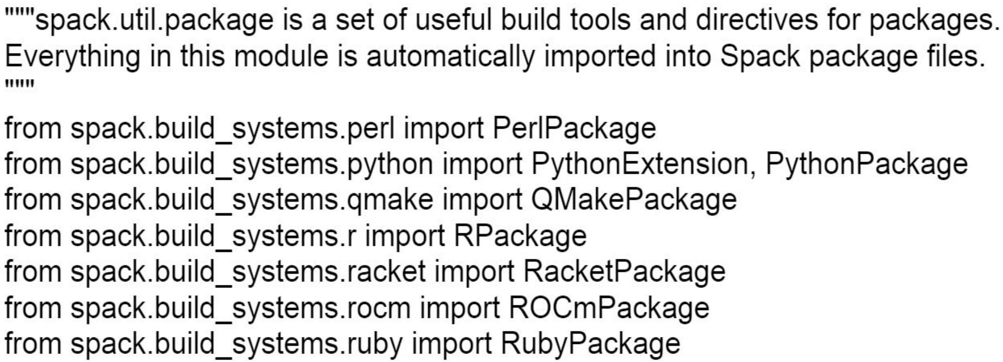
Setting only the Python tools env (330 packages): slow as compared to CMSDIST: <1s vs >40s

```
> which python3
/usr/bin/python3
> spack load python-tools@3.0
> which python3
bash: /usr/bin/which: Argument list too long
```

```
> which python3
/usr/bin/python3
> module load python-tools-3.0-gcc-11.2.1-ovyelip
> which python3
bash: /usr/bin/which: Argument list too long
```

Package dependency on all Build systems

- **Every SPACK package loads all the build system known to SPACK**
  - Why we should care about it?
    - We have to either fix or remove the Build System if it fails due to python version updates within EL distros
      - **EL8 has python 3.6, 3.8 and 3.9**
      - **EL9 has python 3.9 and 3.10**



```
"""spack.util.package is a set of useful build tools and directives for packages.
Everything in this module is automatically imported into Spack package files.
"""
from spack.build_systems.perl import PerlPackage
from spack.build_systems.python import PythonExtension, PythonPackage
from spack.build_systems.qmake import QMakePackage
from spack.build_systems.r import RPackage
from spack.build_systems.racket import RacketPackage
from spack.build_systems.rocm import ROCmPackage
from spack.build_systems.ruby import RubyPackage
```

Missing system dependency checks

- **PKGTOOLS/CMSDIST** uses **RPM** to build CMS Offline packages
  - We maintain our own RPM DB to control what our packages can pick up from system
  - RPM post build checks make sure that packages we build do not accidentally depend on any thing which is not in our RPM DB
  - This helped use control our system level dependencies and allowed CMS Offline SW to work (build/run) with in a major EL release cycle
- https://github.com/spack/spack/pull/28109 provides a partial implementation
  - **Opened since Dec 2021**
  - **Uses ldd and hardcoded list of system libs**
    - This functionality is not any way near to what **rpmdeps** provides
  - **Shows that SPACK package can randomly pick up system libraries**

Some other random things

- SPACK concretize your whole env even if you want to build single package
  - Copies all the patches of all the versions/variants for all packages in your env
  - Might be very annoying for developers
- SPACK compiles/generates byte code for all the packages known to it (6.7K)
  - So you have to maintain all packages even if you do not use those

```
> ls var/spack/repos/builtin/packages/*/__pycache__/package.cpython-36.pyc |wc -l
  6753
```

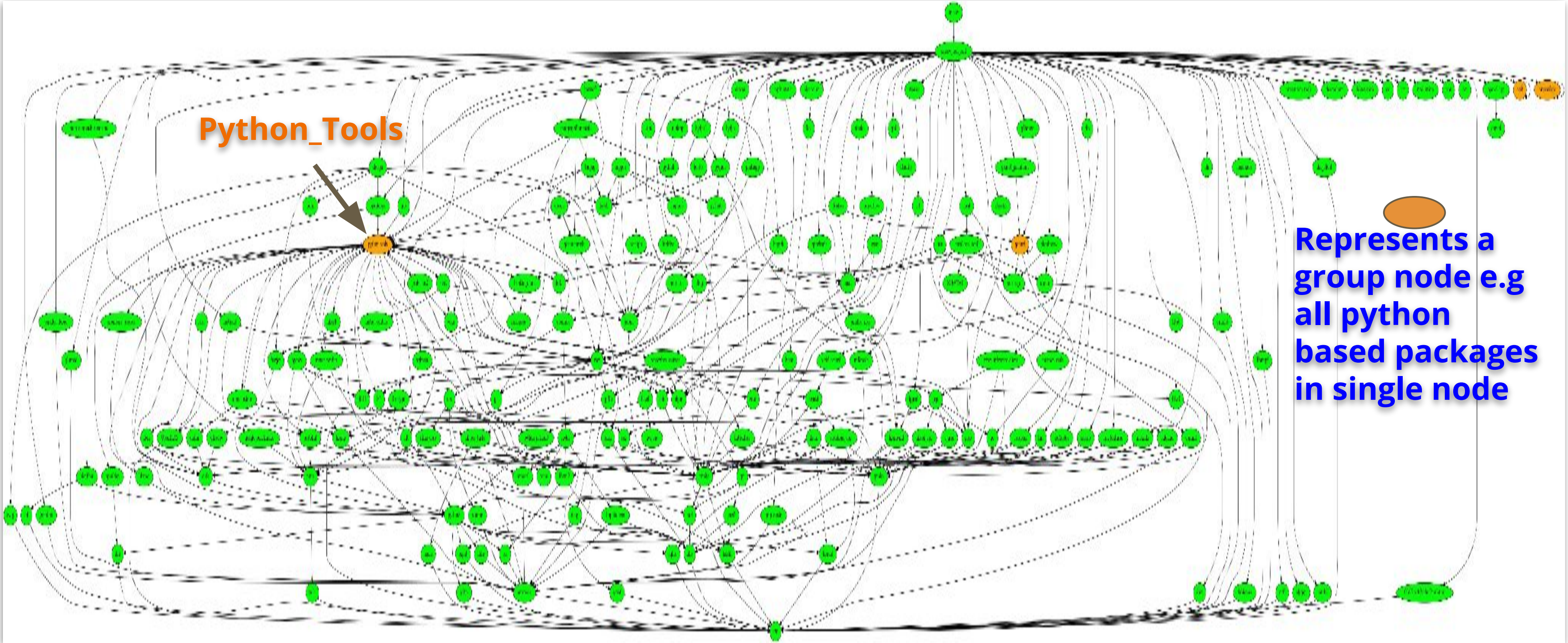
- No easy way to search binary caches for installation
  - We use tricks like copying a pre-concretized lock file (from the build step) to tell SPACK what to install
- We still have no experience with multiple parallel builds building/uploading same package
- Relatively large distribution size: **+15% (45G vs 39G)**

Summary

- SPACK is nice tool for small env with relatively small deps and life cycles
  - CMS Software stack life cycle is even longer than EL distros
- Along with all the slowness/issues (as compare to existing CMS Packaging system), it also comes with high maintenance overhead
- Package recipes are not written with much care and will break any large software stack sooner or later
  - Specially for SW stack with multiple ML tools which maintain/build their own deps
- I am sure I might have missed many nice features of SPACK but things which matters most for CMS software stack do not look good
- I am afraid it is not the right tool for packaging large software stacks

Backup

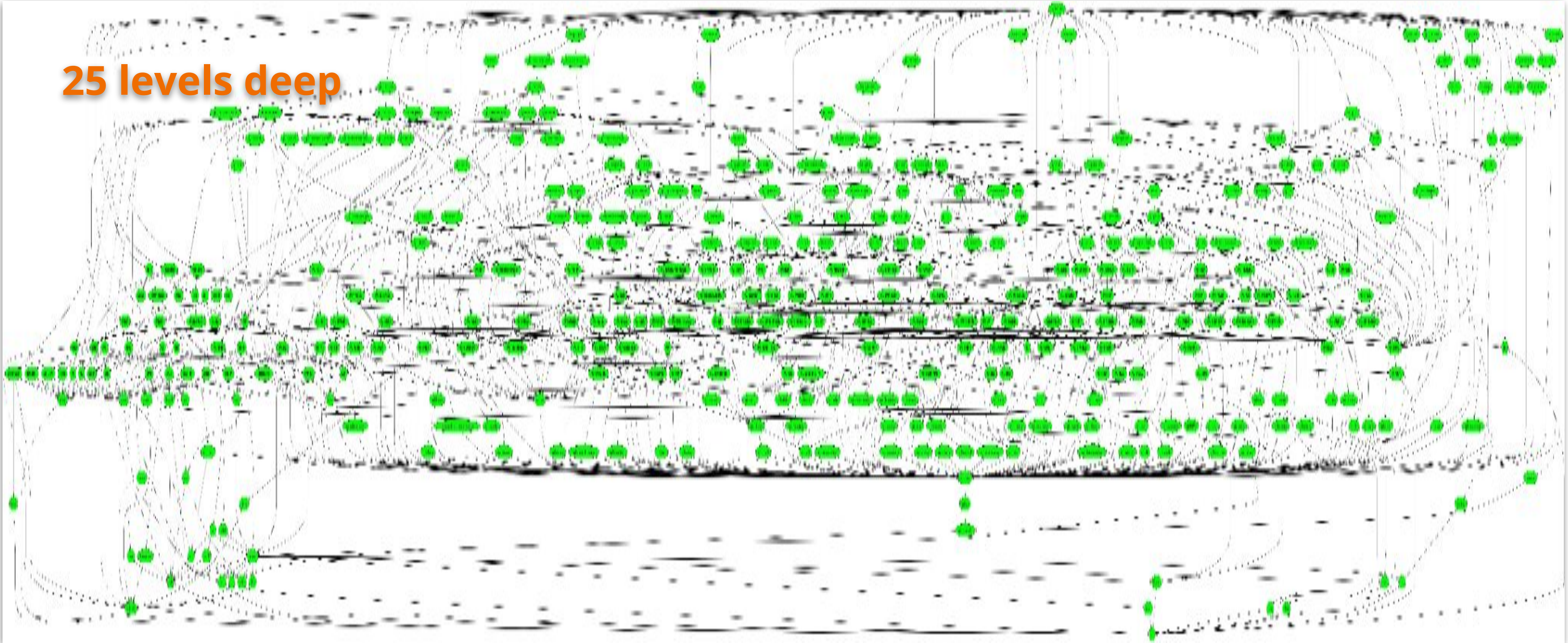
CMSSW: External Package Dependency



Python_Tools

Represents a group node e.g all python based packages in single node

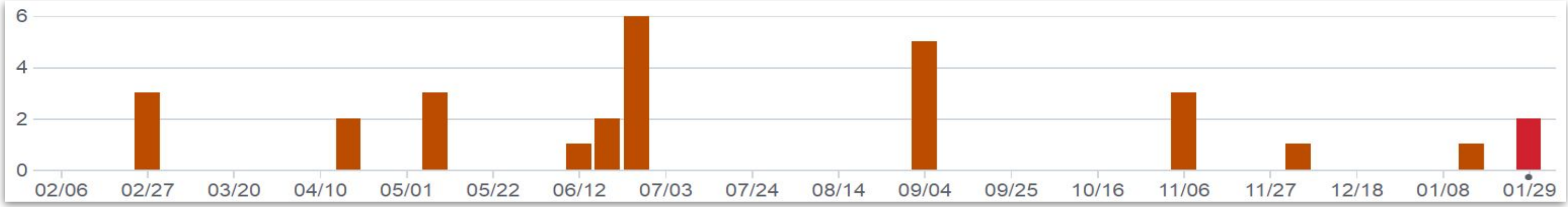
CMS Software stack: Python Packages Dependency



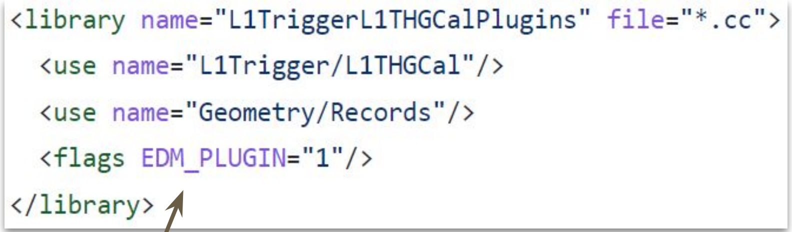
25 levels deep

SCRAM

- **CMS Software Configuration, Release and Management tool**
  - Build System
  - Software Development and runtime environment setup
- Since 1998, this is the only tool CMS has used to build offline software
  - Initially written in **PERL** but in 2020/21 it was rewritten in **PYTHON**
    - Reduced code base: **V2: 8.5K Perl**, **V3: 3.5K Python**
- **Backward compatibility within a SCRAM major version**
  - SCRAM V2 latest version can still build and set env for 12 year old **CMSSW_3_9**
- It has been very stable and has not shown any sign of aging

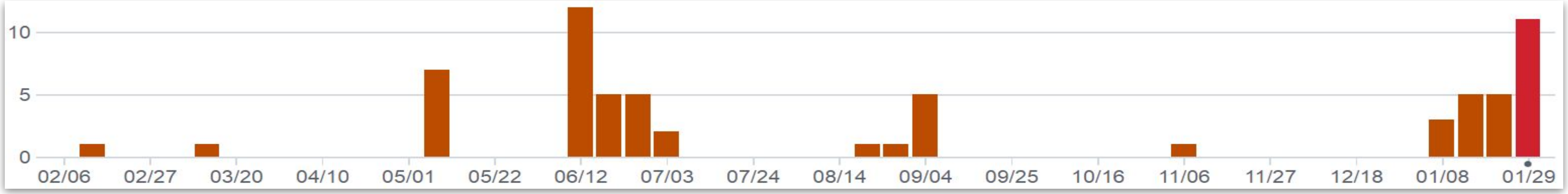


CMSSW-Config



```
<library name="L1TriggerL1THGCalPlugins" file="*.cc">
  <use name="L1Trigger/L1THGCal"/>
  <use name="Geometry/Records"/>
  <flags EDM_PLUGIN="1"/>
</library>
```

- Build rules for CMS offline software
  - gmake based rules
    - **3.5K lines of GMAKE**
    - **3.3K of Python** : To convert XML based rules in to gmake targets
  - SCRAM uses these rules to build CMSSW/FWLite/CORAL
- Provides hooks for SCRAM to dynamically set build/run time environment
  - CUDA Runtime
  - Micro-architectures

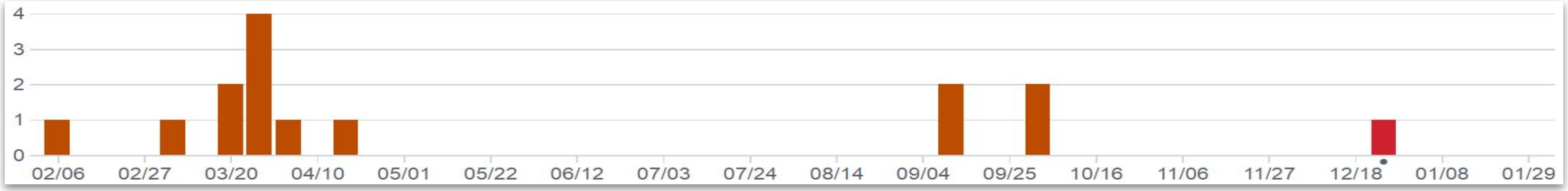


CMSSW-Config...

- **CMSSW** build rules are result of many years of work
- Build rules has been profiled multiple time to maximize the utilization of build and development resources
  - We avoid long **PATH, LD_LIBRARY_PATH, PYTHONPATH, CMSSW_SEARCH_PATH**
    - libs, bins, data and python modules under single directory
      - Just like **view concept of LCG/Spack** but CMS has been using it since 2008
  - Running of all executables with full paths to avoid searching those in **PATH**
  - Avoid duplication of *-Iincdir, -Llibdir* to improve the compile/link time
  - Proper (transitive) dependency handling to avoid passing of extra *-Idir*
  - Compact **GMake** rules in few files instead of many small ones
    - Smaller files (e.g. one per binary/library) takes a lot of time to load

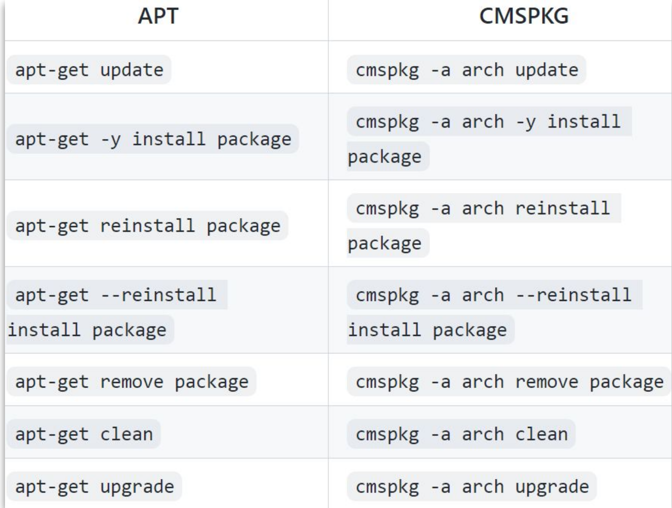
PKGTOOLS

- CMS Offline Packaging tool
  - Build packages and their dependencies from sources
  - Written in **PYTHON** and uses **Redhat Package Manager (RPM)** as backend for building and installing packages
  - Nearly **200K binary packages** in the repository for various OS, archs and compilers
    - **6.7K CMSSW releases**
- Very stable and requires low maintenance
  - **5K lines of PYTHON code**
  - Single version to build full software stack of all **CMSSW** release cycles

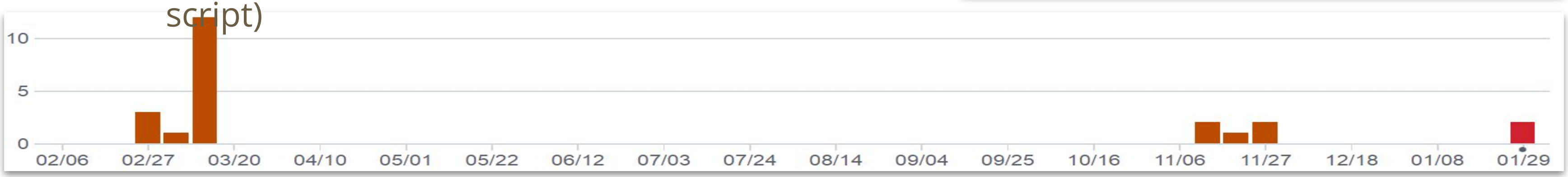


CMSPKG

- Package distribution and installation tool. **PKGTOOLS** uses it to
  - download and install prebuilt packages
  - Upload newly build packages
- It was developed to replace apt/yum usage
- Uses **RPM** dependency information instead of maintaining its own dependency DB
- **<15m** to install full pre-build CMS-SW stack
- One version to install any package
  - Used by all CMSSW release cycles/archs
  - **2K Python, 2K bash** lines of code (bootstrap script)

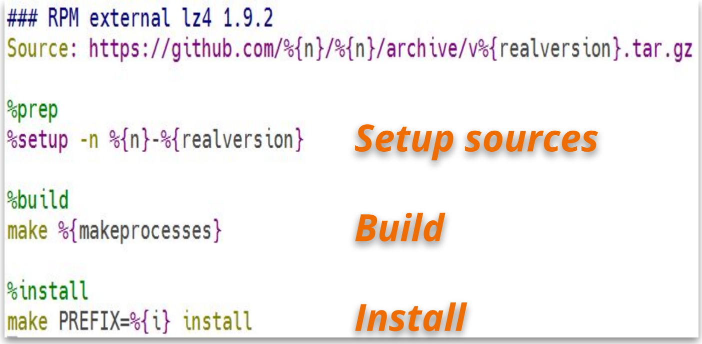


APT	CMSPKG
`apt-get update`	`cmspkg -a arch update`
`apt-get -y install package`	`cmspkg -a arch -y install package`
`apt-get reinstall package`	`cmspkg -a arch reinstall package`
`apt-get --reinstall install package`	`cmspkg -a arch --reinstall install package`
`apt-get remove package`	`cmspkg -a arch remove package`
`apt-get clean`	`cmspkg -a arch clean`
`apt-get upgrade`	`cmspkg -a arch upgrade`



CMSDIST

- ❖ Collection of build recipes
  - ➢ **RPM Spec** files to provide the actual build/install instructions and packages requirements
    - ■ Uses **BASH** for build recipes
- ❖ **650 package recipes**
  - ➢ **320 Python** packages
  - ➢ **90 data** packages
- ❖ One version/configuration per package
  - ➢ Multiple version supported via separate SPECs
- ❖ **20K lines** of build recipes

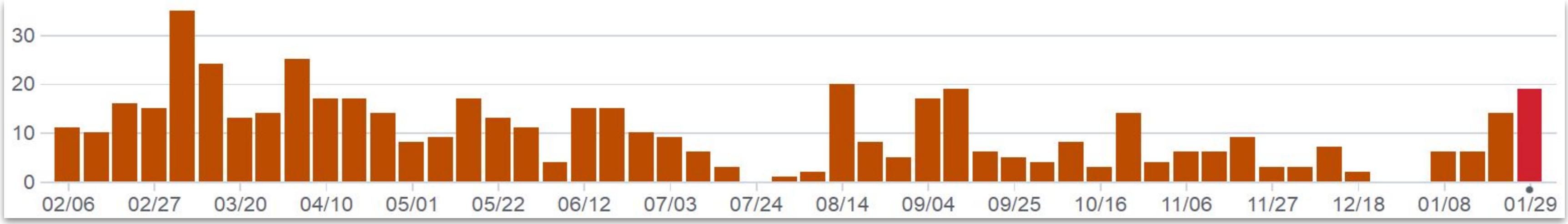


```
### RPM external lz4 1.9.2
Source: https://github.com/%{n}/%{n}/archive/v%{realversion}.tar.gz

%prep
%setup -n %{n}-%{realversion}

%build
make %{makeprocesses}

%install
make PREFIX=%{i} install
```

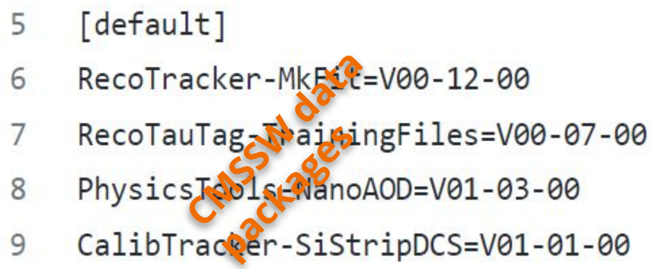
Setup sources

Build

Install



CMSDIST...

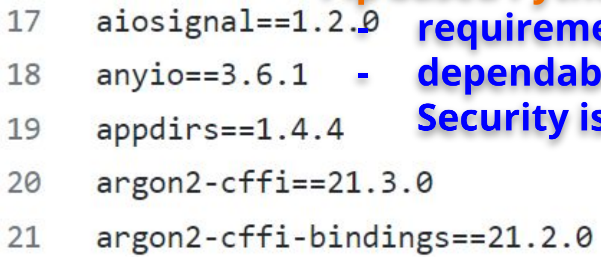


```
[default]
RecoTracker-MkFit=V00-12-00
RecoTauTag-TrainingFiles=V00-07-00
PhysicsTools-NanoAOD=V01-03-00
CalibTracker-SiStripDCS=V01-01-00
```

CMSSW data packages

- ❖ Supports automatic generation of packages
  - ➢ Pip based packages
  - ➢ CMSSW Data packages
  - ➢ Multi-Vectorization packages zlib, zlib_haswell, zlib_skylake-avx512 etc.
- ❖ Non-root installation by creating a separate RPM DB
  - ➢ Well controlled system package dependency
  - ➢ RPM post build dependency checks: To make sure we do not use anything outside our control

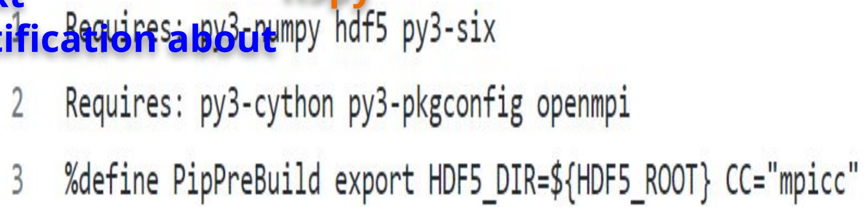
Pip based Python packages



```
aiosignal==1.2.0
anyio==3.6.1
appdirs==1.4.4
argon2-cffi==21.3.0
argon2-cffi-bindings==21.2.0
```

- requirements.txt
- dependabot notification about Security issues

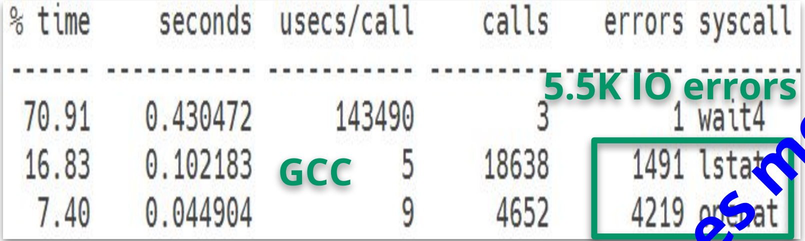
h5py



```
Requires: py3-numpy hdf5 py3-six
Requires: py3-cython py3-pkgconfig openmpi
%define PipPreBuild export HDF5_DIR=${HDF5_ROOT} CC="mpicc"
```

Compiler Wrappers: Injecting -I/-L CMSSW sources

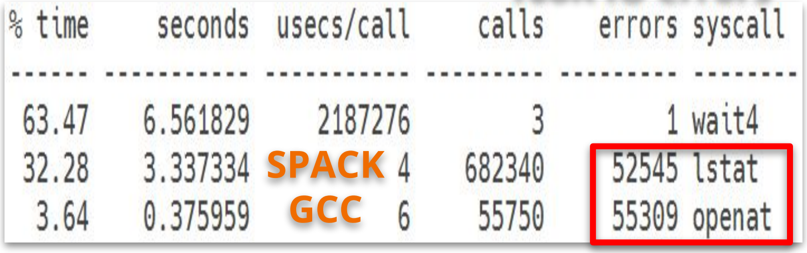
- **Alignment/CocoaDaq/src/CocoaDaqRootEvent.cc**:1 include (indirectly include few root headers)



5.5K IO errors

```
% time     seconds  usecs/call     calls    errors syscall
------ ----------- ----------- --------- --------- --------
 70.91    0.430472      143490         3         1 wait4
 16.83    0.102183           5     18638      1491 lstat
  7.40    0.044904           9      4652      4219 openat
```

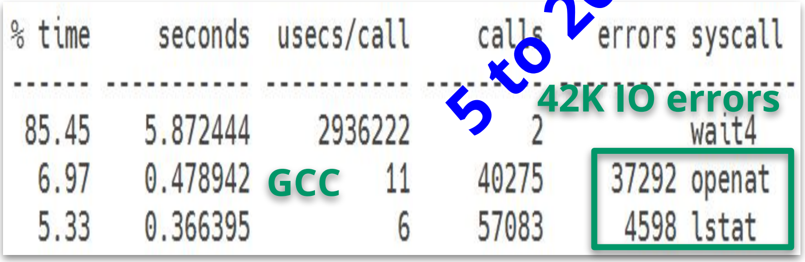
108K IO errors



```
% time     seconds  usecs/call     calls    errors syscall
------ ----------- ----------- --------- --------- --------
 63.47    6.561829     2187276         3         1 wait4
 32.28    3.337334           4    682340     52545 lstat
  3.64    0.375959           6     55750     55309 openat
```

5 to 20 times more IO errors

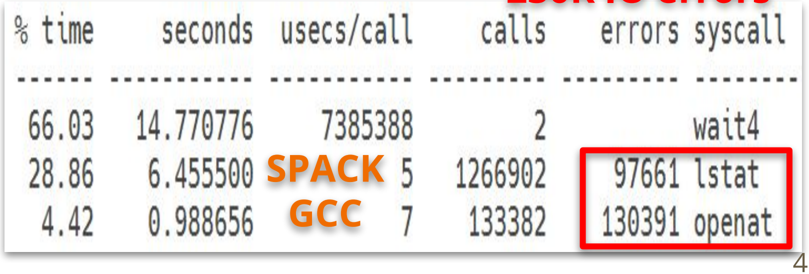
- ❖ **DQM/Physics/src/plugins.cc**: **1055 includes**



42K IO errors

```
% time     seconds  usecs/call     calls    errors syscall
------ ----------- ----------- --------- --------- --------
 85.45    5.872444     2936222         2           wait4
  6.97    0.478942          11     40275     37292 openat
  5.33    0.366395           6     57083      4598 lstat
```

230K IO errors



```
% time     seconds  usecs/call     calls    errors syscall
------ ----------- ----------- --------- --------- --------
 66.03   14.770776     7385388         2           wait4
 28.86    6.455500           5   1266902     97661 lstat
  4.42    0.988656           7    133382    130391 openat
```

rpath/runpath Issues

- **RPATH/RUNPATH are great and works for most of the cases**
  - No need to set global **LD_LIBRARY_PATH** which can break other tools
    - **CMSSW env breaks CMS Computing env**
- **Does not work for packages with stubs libraries** which are suppose to be loaded from correct path at runtime
  - cuda
  - dpm
  - tkonlinesw
- **Can not use RPATH if your software stack has packages with stubs libs**
  - **RUNPATH** works but then you need to set **LD_LIBRARY_PATH** to load the correct library at runtime

Life without LD_LIBRARY_PATH

- **No CMSSW Patch release**
  - CMSSW libraries loaded from full release will load all shared libraries from full release even if those are available in patch release
- **No CMSSW developer area**
  - Same issue as patch release
- **No usage of packages with stubs libraries**
  - cuda
- **No multi micro-architectures builds**
  - Which needs to load different vectorization library at runtime
- **Lose the ability to build an external and test in developer area**
  - Developers of externals e.g. root/geant4 etc. use their build system to build and use it directly in cmssw by updating the tool file
- **ROOT dictionaries loading does not work**