LAWSCHEP workshop,November 20-23 2019, Mexico City

Software development, deployment, validation, verification

Pere Mato, CERN EP-SFT

Challenges

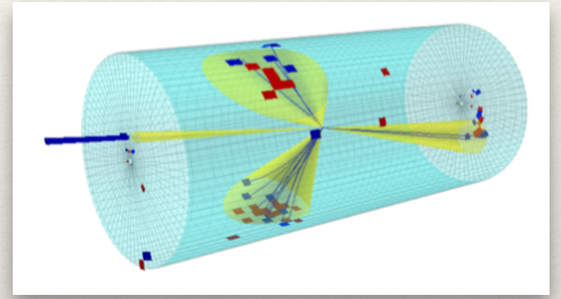
- Distributed development of HEP software implies best practices of code and people management
- Software quality in terms of modularity, interoperability and reusability
- Software sustainability given the long timescale of HEP experiments
- Software testing and validation
- Software licensing and distribution
- Recognition of the contributors

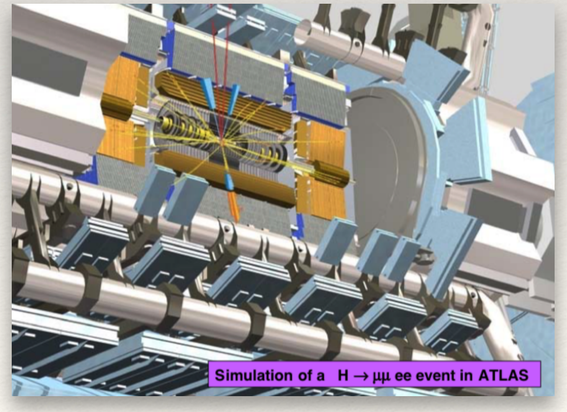
Motivations for HSF

- Much of our HEP software is now old (> 25 years)
  - It needs to be adapted to more modern standards
- Paradigm-shift resulting from the evolution of CPU architectures (accelerators)
  - Our code has to be re-engineered to make use of the full capabilities
- Make use of all resources available to our community
  - HPC facilities, commercial clouds, volunteer resources, etc
- Must attract people with the required advanced skills and experience
- Ensure interoperability with software developed by other scientific communities
- Opportunity for sharing software between different experimental programs

Experiment Software Lifecycle I

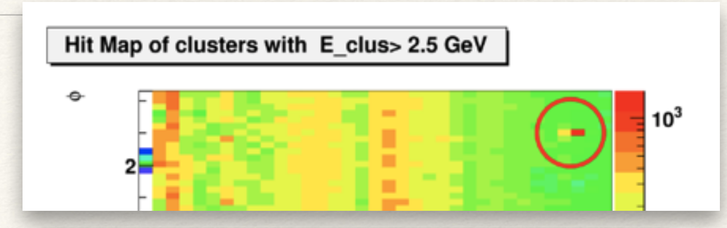
- First ideas and inspiration…
  - Cool idea... would that work?
- Concepts
  - Very fast approximate methods, e.g. Delphes (smeared tracks, etc.)
- Design
  - Still need to be flexible to decide between alternatives
  - Ultimately need to pay a lot of attention to details for accurate performance evaluation
    - Accurate geometry, full simulation, realistic digitisation, ...

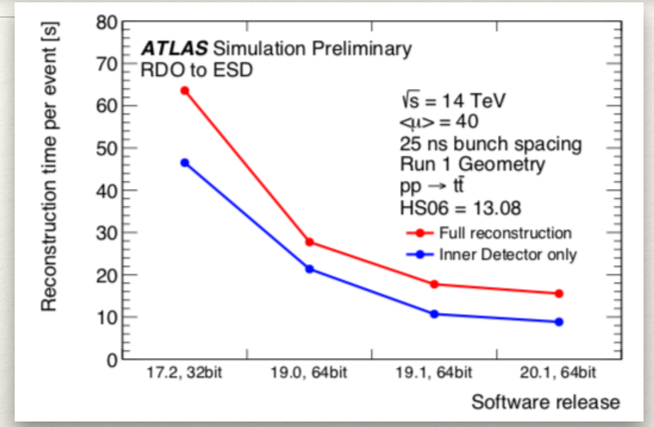


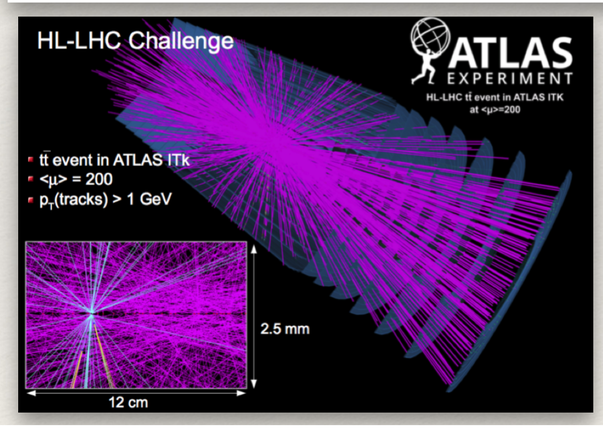


Experiment Software Lifecycle II

- Production
  - Dealing with the real world - calibration, alignments, dead and noisy elements
  - Learn about the detector, need stability but also continual improvements
- Upgrade
  - Design better sub-detectors for the next version
- Preservation
  - How can I make sure we can look at the data in the future?



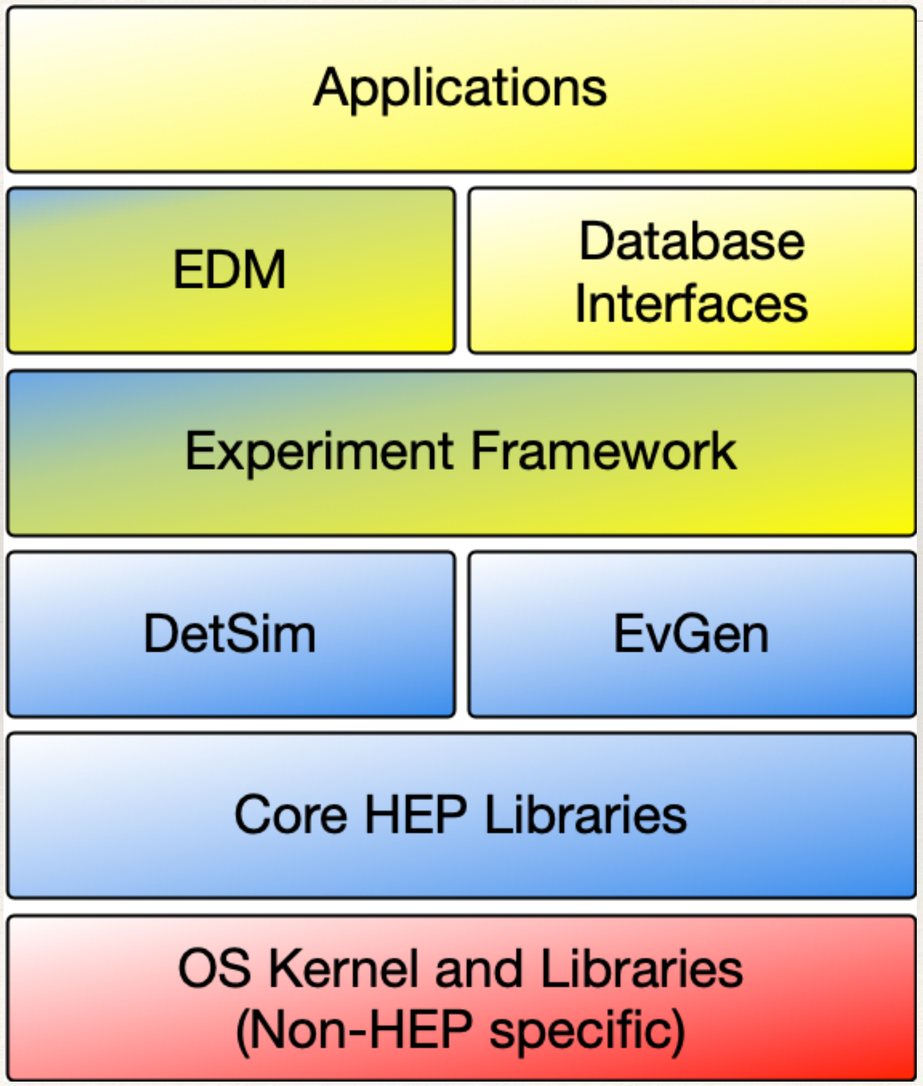





```
TOFGN=TOFG*1.E+9
WRITE(CHMAIL,1000)ITRA,ISTAK,NTMULT,(NAPART(I),I=1,5),TOFGN
CALL GMAIL(0,0)
WRITE(CHMAIL,1100)
CALL GMAIL(0,0)
IEVOLD=IEVENT
NTMOLD=NTMULT
```

Snippet from CERNLIB

HEP Application Software



Application layer of modules/algorithms/processors that perform physics tasks (some generic examples like FastJet, Acts and PandoraPFA)

Usually experiment specific libraries for data representation and access: e.g. xAOD, LCIO; also detector specific conditions data

Experiment core orchestration layer, where everything else plugs in: Gaudi, CMSSW, Marlin

Specific components used by many experiments: Geant4, DELPHES, Pythia, ...

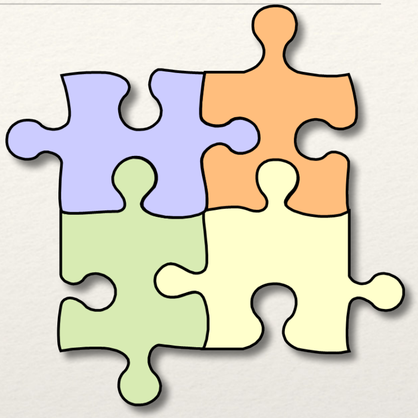
Provide core functionality widely used: ROOT, HepMC, HepPDT, DD4hep, ...

Many widely used non-HEP libraries: Boost, Python, Zlib, CMake, …

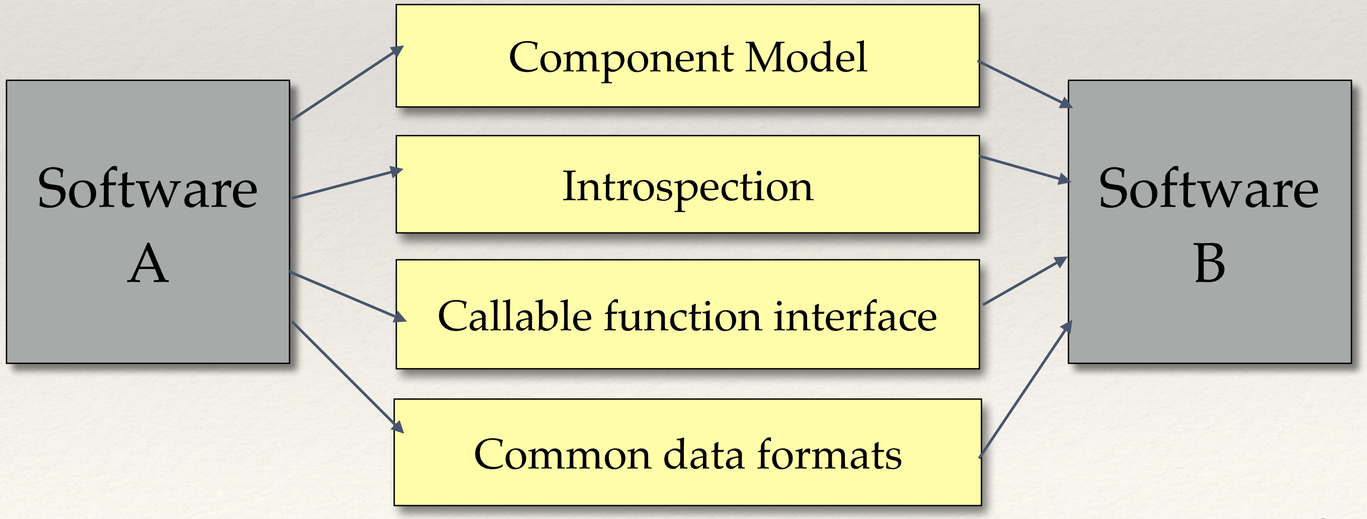
Software Components

- Foundation Libraries
  - Basic types
  - Utility libraries
  - System isolation libraries
- Mathematical Libraries
  - Special functions
  - Minimization, Random Numbers
- Data Organization
  - Event Data
  - Event Metadata (Event collections)
  - Detector Conditions Data
- Data Management Tools
  - Object Persistency
  - Data Distribution and Replication
- Simulation Toolkits
  - Event generators
  - Detector simulation
- Statistical Analysis Tools
  - Histograms, N-tuples
  - Fitting
- Interactivity and User Interfaces
  - GUI
  - Scripting
  - Interactive analysis
- Data Visualization and Graphics
  - Event and Geometry displays
- Distributed Applications
  - Parallel processing
  - Grid computing

Interoperability



- Capability of different software components to exchange data via a common set of exchange formats or interfaces
- There are several levels of interoperability
  - Level 0 - Common Data Formats
  - Level 1 - Callable Interface
  - Level 2 - Introspection Capabilities
  - Level 3 - Component Model



Interoperability I

- Level 0 - Common Data Formats
  - Allows interoperability between different programs, even running on different hardware
  - E.g., HepMC event records, LCIO, GDML, ALFA messages
- Level 1 - Callable Interfaces
  - Basic calling interfaces defined by the programming language
    - Cross language calls are, of course, possible
  - Can be dependent on the compiler and language version (C++ in particular)
  - Details are important
    - how to handle errors and exceptions, is it thread safe, are objects const, dependent libraries and runtime setup
  - E.g., FastJet, Eigen, Boost

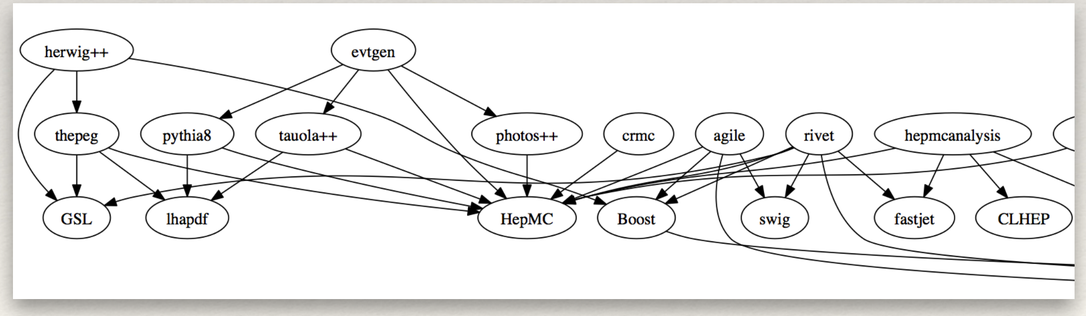
Interoperability II

- Level 2 - Introspection Capabilities
  - Software elements to facilitate the interaction of objects in a generic manner such as Dictionaries and Scripting interfaces
  - Example: PyROOT, which is a Python extension module that allows the user to interact with any ROOT (C++) class from the Python interpreter
- Level 3 - Component Model
  - Software components of a “common framework” offers maximum re-use
  - ‘standard’ way to configure its parameters, to log and report errors, manage object lifetime and ownership rules, ‘standard’ plug-in management, etc.
  - Unfortunately, no single Framework has been generally adopted

The right interoperability point between packages varies, but fixing it correctly eases life a lot for other developers and users

Package Dependencies

- Very few packages are truly standalone (not having any dependency)
  - Very often packages depend of other packages
- Package dependencies are difficult to manage
  - Complicates the configuration, the build process, the distribution and the deployment
- **Avoiding dependencies is not a good solution in general**
  - Adds code duplication
  - Reduces code re-use
- **Managing dependencies is essential**
  - ‘Standards’ and tools are required



Fragment of the dependency graph of some MC generator packages

Lowering the Barriers for Re-use

- Cataloging all packages and tools to avoid the re-invent syndrome
  - If people know what exists, perhaps they will re-invent less
- Users should get what they need very easily and no more
  - ROOT is a very good example of easy installation, however the model has been “take all or nothing”
  - Similarly with Geant4, many specialised physics processes are built and installed by default and they are most of the time not needed
- We can do much better
  - For example in R, additional modules are downloaded, build and installed at runtime when required
- Good moment to **re-think modularity** requirements

HSF Project Template

- To enable interoperability and long term maintainability
  - Build system
    - Share build results, but also knowledge of how to build
    - HSF Packaging working group studying various options
  - Testing
  - Licensing and Copyright
  - Documentation
    - If it is not documented, does it exist?
- Ensure as much uniformity as reasonable
  - https://github.com/HSF/tools

Common HSF Tools

This is a collection of tools used within the context of the HEP Software Foundation (HSF).

create_project

The tool create_project creates a template CMake project. The created project contains the standard use patterns for small CMake projects, plus support for Doxygen and CPack. Further documentation is provided within the created package itself inside the README.md.

```
# - Define the minimum CMake version
# HSF recommends 3.3 to support C/C++ compile features for C/C++11 across all
# platforms
cmake_minimum_required(VERSION 3.3)
# - Call project() to setup system
# From CMake 3, we can set the project version easily in one go
project(prmon VERSION 1.1.1)

#--- Define basic build settings ----------------------------------------
# - Use GNU-style hierarchy for installing build products
include(GNUInstallDirs)

# Add some build option variables, for static builds and profiling
if(NOT BUILD_STATIC)
  set(BUILD_STATIC OFF)
endif()
set(BUILD_STATIC "${BUILD_STATIC}"
  CACHE BOOL "Build a static version of the prmon binary" FORCE)

if(NOT PROFILE_GPROF)
  set(PROFILE_GPROF OFF)
endif()
```

Project Independency

- Each development team should keeps its autonomy
  - HSF does not enforce any particular software process, project management or methodology
  - Each team may use its own repository, bug tracker, web project site, user forum, etc.
  - But HSF may provide them if needed
- ‘Ownership’ of the package resides with its developers
  - A clear way of recognition and proper credits
  - At the same time the developers need to ensure support and maintenance

Software Process and Tools

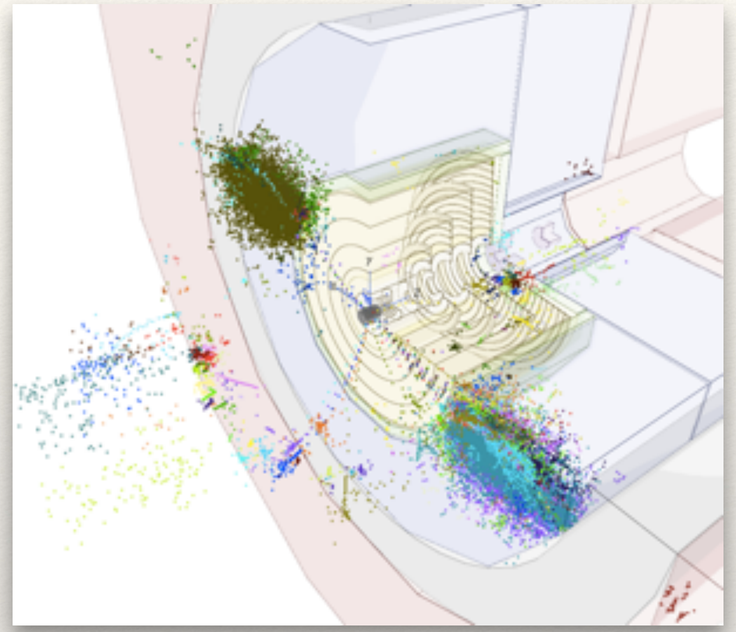
- HSF does not enforce a given software process but it encourages a common set of ‘best practices’ and tools
  - Creating a kind of consensus would certainly facilitate integration, testing, distribution and deployment
- Build Systems
  - For example the HEP community is using more and more CMake
- Continuous Integration
  - GitLab/GitHub CIs, Jenkins
- Testing
  - Defining and running tests should be strait-forward
  - CMake companion tool, CTest could be a good proposal
- Documentation
  - Reference documentation from comments in code (i.e. Doxygen)
  - User guides

Turn-Key Systems

- In addition to individual software packages providing a specific functionality there is the need to provide complete turn-key systems
- Small experiments or experimental programs cannot afford to build custom solutions (from pieces) for their data processing applications
  - For example the experiment studies of FCC
- We need to provide complete solutions including
  - Data processing framework, event data model, detector description
  - Built-in generators and detector simulation
  - Basic reconstruction and analysis tools
  - etc.

Key4hep

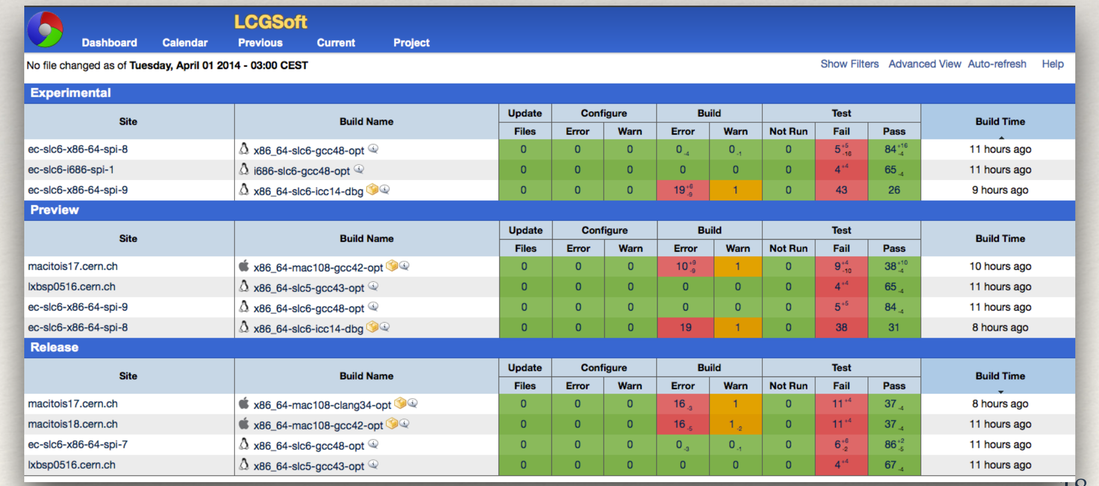
- Software challenges are faced by detector community at future facilities
- Likely to be a Higgs-factory, but several different projects are possible:
  - CEPC, CLIC, FCC-ee, ILC
- Need for software which is robust, mature, yet sufficiently flexible to try new ideas



Jet tagging capabilities with 5TeV b-jets in FCC-hh, but using the CLIC software and the FCC vertex tracker, combined in the CLIC detector model
André Sailer, CLIC

CI and Testing

- In general the package must portable and not restricted to a particular compiler or operating system
  - Testing of the portability should be ensured by building and running tests on many platforms
- Interoperability between packages should also be validated
  - Complete builds of all packages for a number of configurations should be done
  - E.g. LCG software stacks
- A continuous build and testing system should be setup with clear reports to developers



Software Stacks and Deployment

- HEP software stacks, in common with many software projects, **need to maintain multiple versions**
  - These versions generally evolve their external dependencies as well
  - Unlike other projects these versions usually have to be maintained for many years
- Build system must be able to support and patch stack versions years after their original deployment
  - External dependency issues can occasionally be the issue requiring patching
  - Significant trouble can arise when an underlying OS distribution dependency goes out of support
- Deployment is a closely coupled problem to the actual build
  - Our lives have been hugely eased by the widespread adoption of CVMFS and container technology

HSF Packaging Working Group

HEP Software Foundation

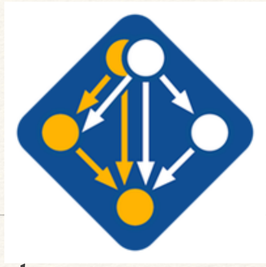
- Packaging and deploying a software stack is a problem faced right across HEP and the wider scientific community
  - Every experiment and software group has to put effort into doing this
  - Naively it seems an easy problem, but it quickly gets complicated and seemingly obvious solutions don't meet requirements
- Motivated formation of WG in 2015 as a forum for working together to improve
  - Knowledge sharing on tools and workflows in and outside HEP
- We looked at many tools - general FOSS, scientific community, HEP specific
  - We extracted use cases and provided bootstrap instructions to try out a number of tools
  - Focus now moved to implementation of stack using the most promising candidates…
    - Group continues to meet regularly for progress reports and to exchange information

Packaging Desiderata

HEP Software Foundation

- Support NxMxC complexity
  - Software versions, architectures (and micro architectures or instruction sets), with build options, compiler versions
- Reproducibility
  - Capture all dependencies reliably
- Relocatable build products
  - Should not be tied to one install path at build time
  - CVMFS, container, local install, binary build products, ...
- Runtime environment setup
  - Production and developer use cases differ slightly, both must be supported

Spack

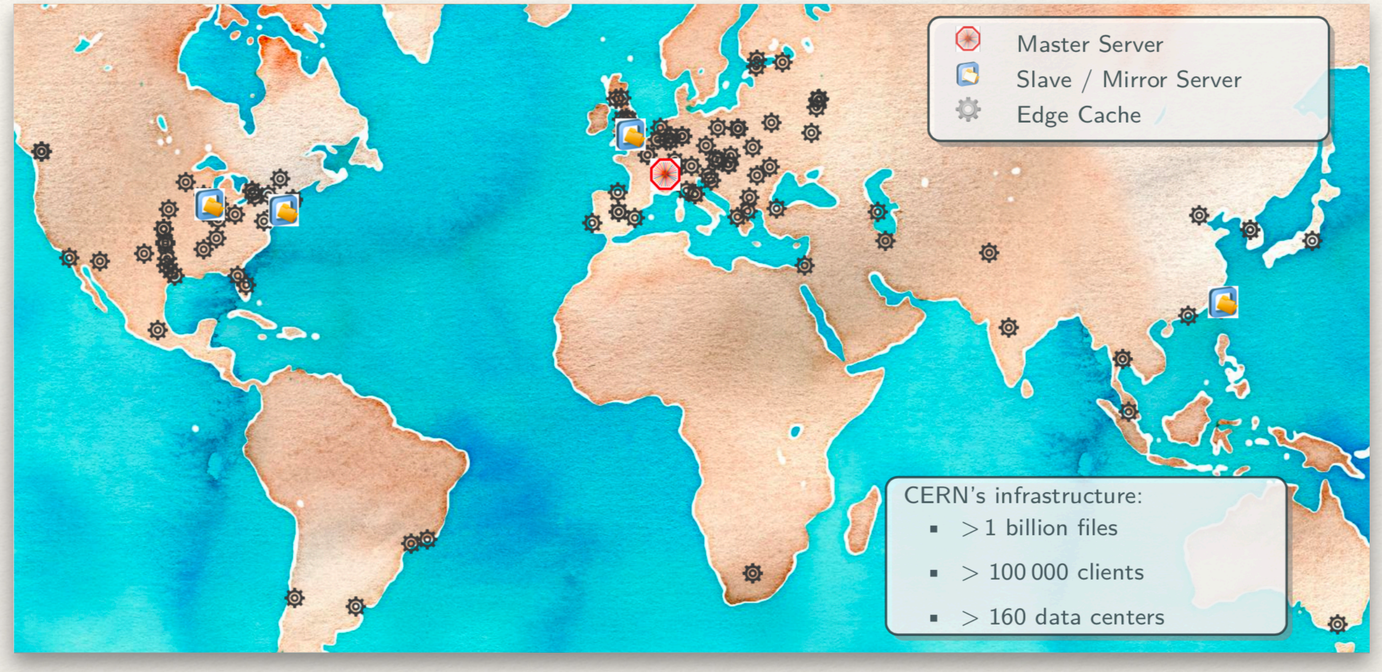


spack.io

- Package manager and build orchestrator developed at Lawrence Livermore National Laboratory
- Originally developed for installing software to HPC systems with strong emphasis on scientific software
- Supports multiple versions of software concurrently
  - Appends build hashes to install locations, RPATH used to resolve the correct dependencies
  - Common dependencies are shared
  - Support for different compiler toolchains as a core concept
- Dependencies are found and installed automatically
  - Full specification of all build options for dependencies supported
  - Will rebuild or install from existing binary build products
- Configuration on command line or from YAML files and package descriptions written in Python
- Large community of contributors, supporting 3.5k packages

CernVM-FS

- File-system optimised for large scale software distribution
  - readonly, heavily cached, install once use everywhere
  - **any client, anywhere, anytime**
- Mission critical for LHC experiments
  - all software distributed to the >160 data centres in a coherent manner
- >100 experiments from HEP and other sciences, including industry users



LCG Releases (an example)

- Configuring, building and deploying external libraries (~300) and MC Generators (~80) for all the supported (~12) platforms (3 os, 8 compilers) used by LHC experiments
  - Releasing full configurations. Content, versions and platforms discussed/agreed with experiments (LIM+AF)
  - The EP-SFT groups have been providing this service to the experiments successfully for the last 10 years
  - Recently we have been adding many packages for data analysis (Python, R, Machine Learning)
- Implementation
  - Builds are done with a home-made tool based on CMake (LCGCMake). Evaluating Spack.
  - The output are RPM/Tarfiles that are installed in AFS and CVMFS and available to experiments
  - Generation of 'Views'

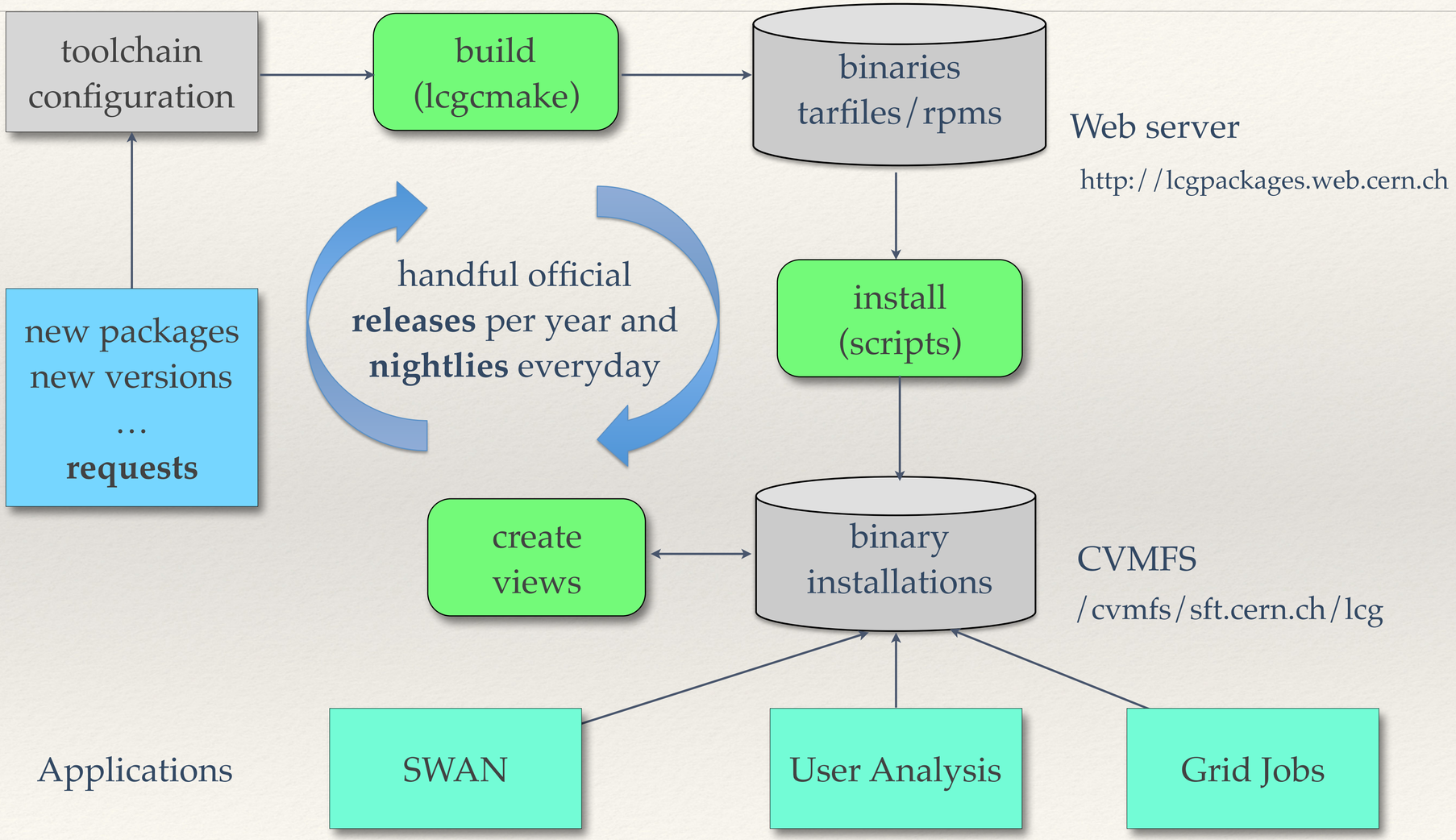
LCG Configurations

- An LCG configuration is a **named** and **immutable** list of (package, version) tuples
  - Examples:
    - LCG_84, LCG_85, LCG_85a
- We also provide every night a **mutable** release candidate for different purposes: **nightlies**
  - For the experiments to test new versions, for users to play, etc.
  - Understanding compatibility and overall integration

pyqt	4.11.4
pyqt5	5.5.1
pytest	2.2.4
pytz	2015.4
Python	2.7.10
PythonFWK	2.7.10
python_dateutil	2.4.0
pytools	1.9
PyYAML	3.11
pyxml	0.8.4p1
pyzmq	14.5.0
QMtest	2.4.1
Qt	4.8.7
Qt5	5.6.0
qwt	6.0.1
R	3.2.0
scikitlearn	0.16.1
scipy	0.15.1

http://lcginfo.cern.ch

Building and Installing



Software Licensing

HEP Software Foundation

- HSF encourages all its members and partners to make available the software they develop as **Open Source**
  - No need to create a new licence. Unusual licence may hinder the redistribution by third parties
  - The exact licence chosen may depend on several factors: redistributable, reusable, whether copyleft (e.g. GPLv3) or permissive (e.g. MIT, Apache 2), etc.
- Need to make sure that we have a **consistent and compatible license situation** within the community
- Software is owned by its copyright holders
  - If copyright can be assigned to one single organisation then the practicalities become far easier
  - This is the approach adopted by ATLAS and CMS, CERN holds copyright ‘for the benefit’

https://hepsoftwarefoundation.org/notes/HSF-TN-2016-01.pdf

Final Remarks

- The community needs to **develop interoperable and sustainable software** made of independent packages with the best development practices
  - Reengineering packages to be adapted to new architectures
  - Ensure a minimal set of standards to facilitate interoperability and compatible licenses
- Building, packaging and deploying software is a shared problem across HEP
  - **HSF Packaging Working Group is an active open forum** for discussion and cooperation
  - Packing tools such as Spack has been successfully tested as a build orchestrator for modern HEP software stacks
- The turnkey stack connects and extends the individual packages to enable a complete data processing ecosystem
  - Converting a set of disconnected packages into a ‘turnkey’ system by providing the necessary ‘glue'
- **Plenty of room for collaboration** in the software development endeavour for HEP