Support for multiple micro-architectures

Pere Mato

HSF Packaging Working Group, 2 May 2018

Problem Statement

- Need to build **separate** software stacks for specific micro-architectures, in particular supporting **specific instructions sets** (fma, avx, avx2, sse4,…), using appropriate flags
  - e.g. `-mavx2`
- Need to make available all these stacks on a shared file system
  - e.g. CVMFS
- High performance applications should **select the appropriate software stack** performing run-time CPU detection
  - e.g. `cat /proc/cpuinfo | grep flags`
- Not all the packages does make use of the sophisticated instruction sets
  - e.g. they do not deal with floating point operations

Recap LCGCMake

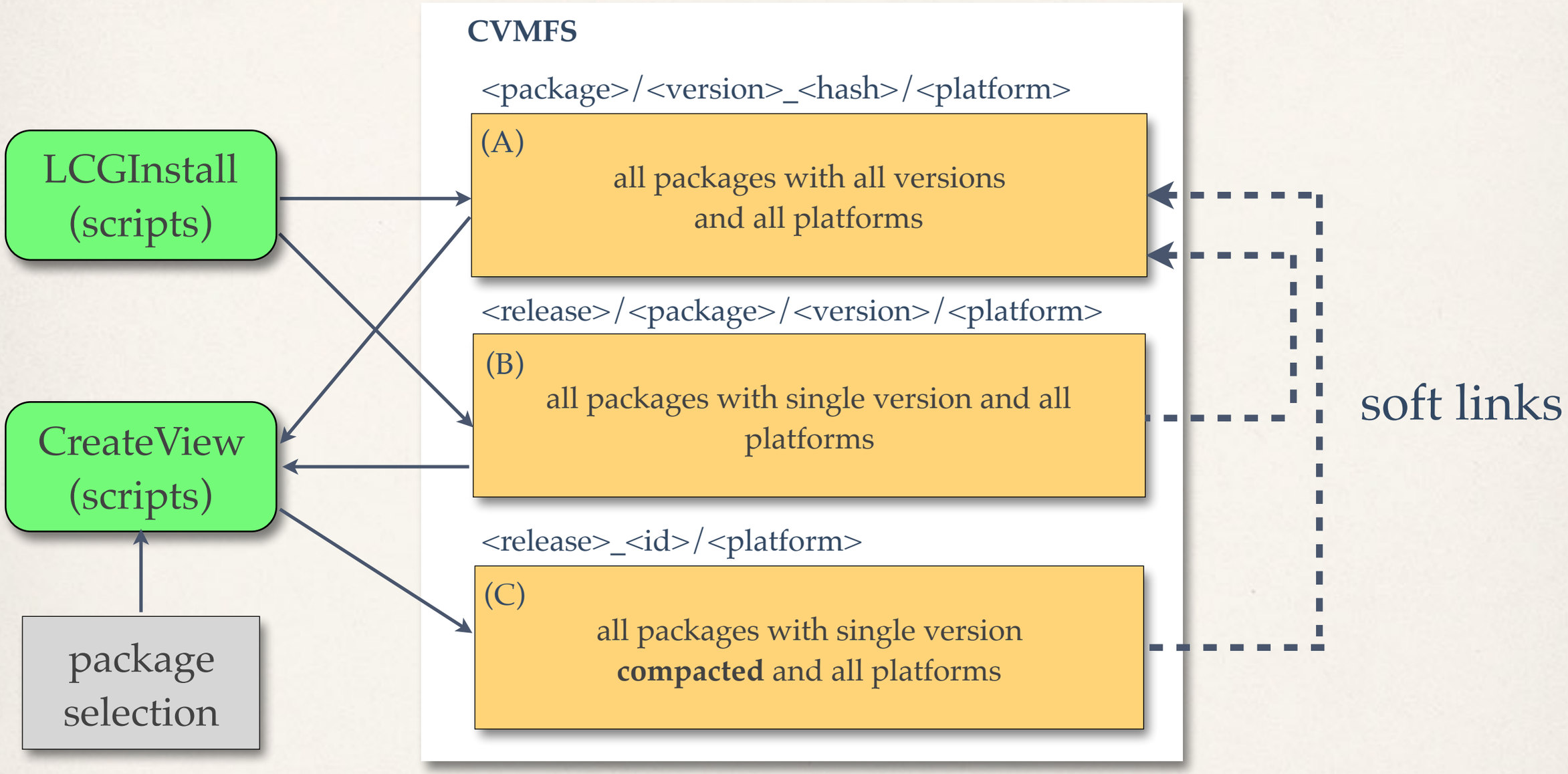
- The LCGCMake is based on the CMake standard module *ExternalProject* that creates custom targets to drive download, update/patch, configure, build, install and test steps of an external package
- A single file lists all the packages and their required versions for a given configuration of the software stack

```
#---agile-----------------------------------------------------------------------

LCGPackage_Add(
  agile
  URL http://www.hepforge.org/archive/agile/AGILe-${agile_native_version}.tar.bz2
  CONFIGURE_COMMAND ./configure --prefix=<INSTALL_DIR>
                                --with-hepmc=${HepMC_home}
                                --with-boost-incpath=${Boost_home_include}
                                --with-lcgtag=${LCG_platform}
                    PYTHON=${Python_home}/bin/python
                    LD_LIBRARY_PATH=${Python_home}/lib:$ENV{LD_LIBRARY_PATH}
                    SWIG=${swig_home}/bin/swig
  BUILD_COMMAND make all LD_LIBRARY_PATH=${Python_home}/lib:$ENV{LD_LIBRARY_PATH}
  INSTALL_COMMAND make install
                  LD_LIBRARY_PATH=${Python_home}/lib:$ENV{LD_LIBRARY_PATH}
  BUILD_IN_SOURCE 1
  DEPENDS HepMC Boost Python swig
)
```

```
# Application Area Projects
LCG_AA_project(COOL  COOL_2_8_17)
LCG_AA_project(CORAL CORAL_2_3_26)
LCG_AA_project(RELAX RELAX_1_3_0k)
LCG_AA_project(ROOT  5.34.05)
LCG_AA_project(LCGCMT LCGCMT_${heptools_version})

# Externals
LCG_external_package(4suite            1.0.2p1                              )
LCG_external_package(AIDA              3.2.1                                )
LCG_external_package(blas              20110419                             )
LCG_external_package(Boost             1.50.0                               )

...

# Generators
LCG_external_package(starlight         r43           MCGenerators/starlight )
LCG_external_package(herwig            6.520         MCGenerators/herwig    )
LCG_external_package(herwig            6.520.2       MCGenerators/herwig    )
LCG_external_package(crmc              v3400         MCGenerators/crmc      )
LCG_external_package(cython            0.19          MCGenerators/cython    )
LCG_external_package(yaml_cpp          0.3.0         MCGenerators/yaml_cpp  )
LCG_external_package(yoda              1.0.0         MCGenerators/yoda      )
```

https://gitlab.cern.ch/sft/lcgcmake

Recap LCG Views




```
/cvmfs/sft.cern.ch/lcg/views/<release>/<platform>/bin
                                                 lib
                                                 include
                                                 ...
```

Environment Setup

```
source /cvmfs/sft.cern.ch/lcg/views/<release>/<platform>/setup.[c]sh
```

- Sourcing a single and simple file sets the full environment for the complete view. It defines trivially:
  - PATH, LD_LIBRARY_PATH
  - PYTHONPATH
  - CMAKE_PREFIX_PATH
  - ROOTSYS, ROOT_INCLUDE_PATH
- Other variables have been added as needed…

Naming the Platform

- With LCGCMake package binaries are installed in:
  - <prefix>/<package>/<version>_<hash>/<**platform_tag**>/…
    - The <platform_tag> is a combination of processor architecture, os version, compiler version and build type (e.g. x86_64-slc6-gcc48-dbg, aarch64-ubuntu14-gcc49-opt)
    - The <hash> value is calculated taking into account the full list of package dependencies and their versions
- The platform_tag has been extended to include the 'enabled' instruction sets
  - See HSF Note in preparation
  - The `architecture` should be replaced by **`architecture+instructionset1[+instructionset2]`**
  - e.g. `x86_64`**`+avx2+fma`**`-centos7-gcc7-opt`

Selecting HP Packages

- Extended LCGCMake with a new flag for the package recipe
  - BUILD_WITH_INSTRUCTION_SET
- These selected packages will be in general build the the required flags (via a compiler wrapper*) to ensure that these are applied

```
#---Vc---------------------------------------------------
LCGPackage_Add(
  Vc
  URL ${GenURL}/Vc-<Vc_native_version>.tar.gz
  CMAKE_ARGS -DCMAKE_INSTALL_PREFIX=<INSTALL_DIR>
             -DCMAKE_BUILD_TYPE=${CMAKE_BUILD_TYPE}
             -DCMAKE_CXX_STANDARD=${CMAKE_CXX_STANDARD}
             -DCMAKE_C_COMPILER=${CMAKE_C_WRAPPER}
             -DCMAKE_C_FLAGS=${CMAKE_C_FLAGS}
             -DCMAKE_CXX_COMPILER=${CMAKE_CXX_WRAPPER}
             -DCMAKE_CXX_FLAGS=${CMAKE_CXX_FLAGS}
             ${Vc_options}
  BUILD_WITH_INSTRUCTION_SET 1
  REVISION 1
)
```

  - The rest of the packages will be built as usual and placed under the ‘naked’ platform tag
- Packages with static libraries ‘contaminate’ dependent packages and they need to be build for specific instruction set

(*) development done by Patricia, Javier, Rafal

Building

- To build a software stack (full or incremental) for a specific instruction set there is a specific option

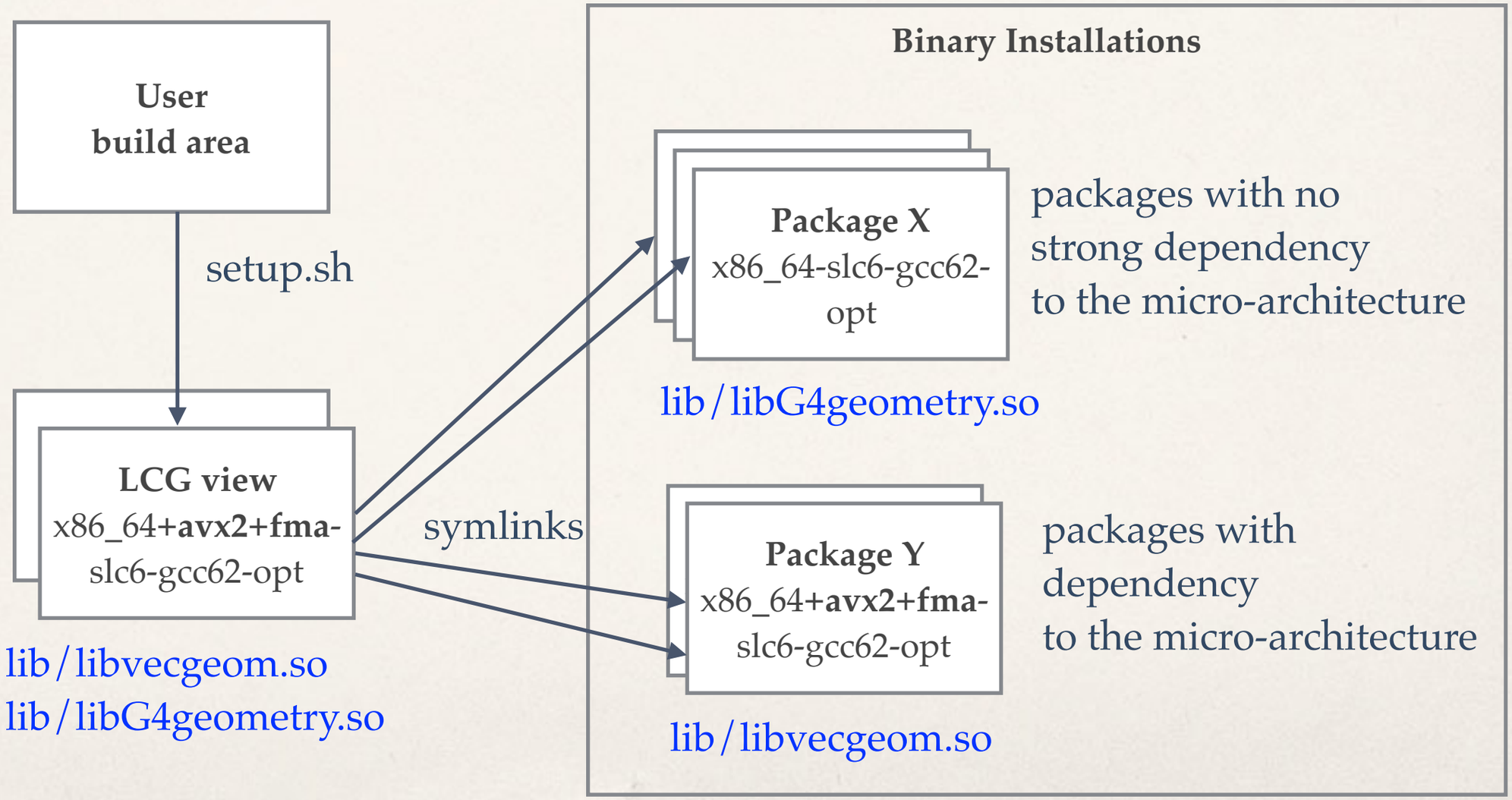
```
$ cmake -DLCG_INSTALL_PREFIX=<prefix>
        -DLCG_VERSION=<version>
        -DLCG_INSTRUCTIONSET=avx2+fma
        ...
        <lcgcmake>

$ make -j<N>
```

- Binaries will be installed in <prefix> with a mix of ‘full platform’ tags and ‘naked’ ones
- To generate the view

```
$ make view
```

Setting the Environment

- The proper runtime environment is set using the `setup.sh` of the view
  - a complete 'view' is made using a mix of packages from different platform tags



Conclusions

- The current solution is minimalistic and builds only a sub-set of packages
  - requires knowledge of the packages themselves: how to build them, what happen with the dependent packages, etc.
- The system is working and being used for the GeantV nightlies to test against different micro-architectures
- Up to us to define what instructions sets are included in the ‘naked’ architecture
- Need still to develop a tool for runtime discovery and selection of the ‘best’ match

Project GeantV-nightly

Configuration Matrix			gcc52			gcc62			gcc7		
Configuration Matrix			NONE	NUMA	TBB	NONE	NUMA	TBB	NONE	NUMA	TBB
centos7	Debug	sse3	●	●	●	●	●	●	●	●	●
		avx2+fma	●	●	●	●	●	●	●	●	●
	Release	sse3	●	●	●	●	●	●	●	●	●
		avx2+fma	●	●	●	●	●	●	●	●	●
slc6	Debug	sse3	●	●	●	●	●	●	●	●	●
		avx2+fma	●	●	●	●	●	●	●	●	●
	Release	sse3	●	●	●	●	●	●	●	●	●
		avx2+fma	●	●	●	●	●	●	●	●	●