SPI
external software build tool and distribution mechanism

http://spi.cern.ch

Application Area Meeting – 22 June 2005

Presentation Outline

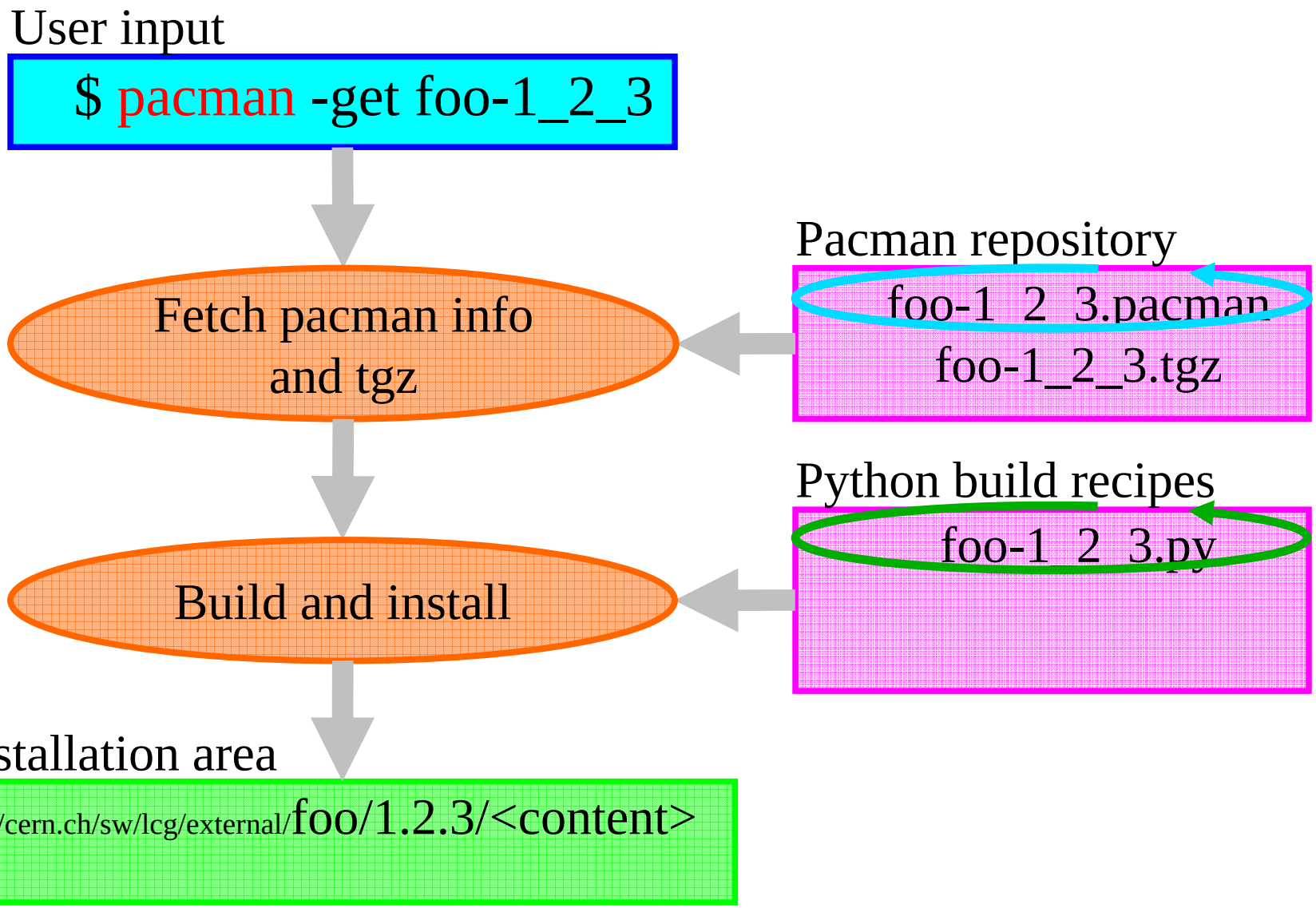
- Installation of external softwares from source
  - **User point of view**
  - **Generation of the pacman and python files**
- Binary packages installation with pacman
  - **User point of view**
  - **Generation of pacman files**
  - **SCRAM and CMT integration**

Requirements and solutions

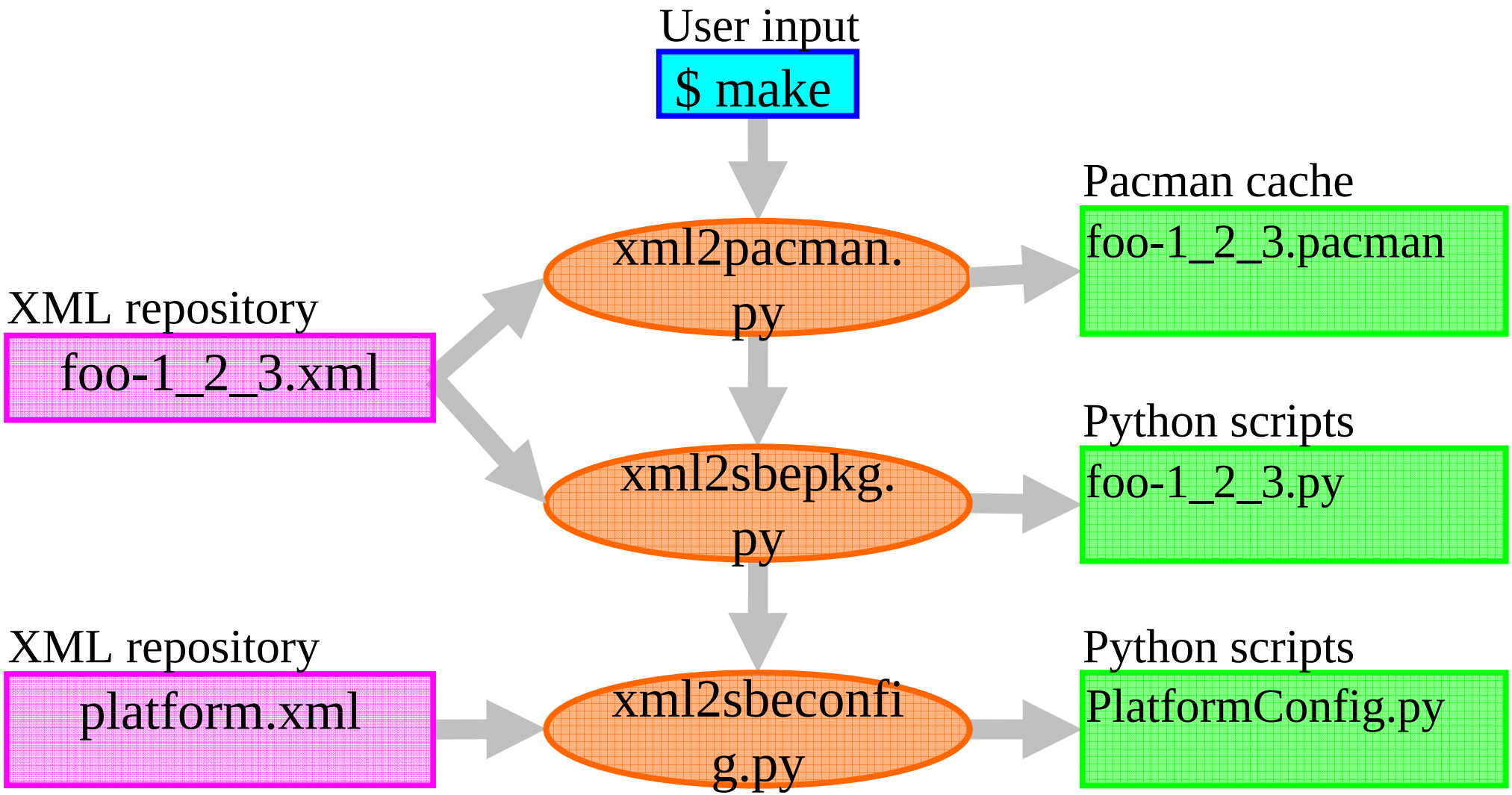
- **Two needs**
  - Building tool for external packages
    - Lots of platforms and packages (several hundreds of combinaisons)
  - Distributing binaries for users
- **Two tools sharing a lot of code and concepts**
  - Based on Pacman
    - Gives a lot of build-in functionalities with minimal intrusiveness.
  - Wrote in python
    - Because it's beautiful.
  - Using XML representation for the package informations
    - Package knowledge is expressed in a tool-independant format.
  - XSLT
    - Natural language to process XML.

Building externals from sources

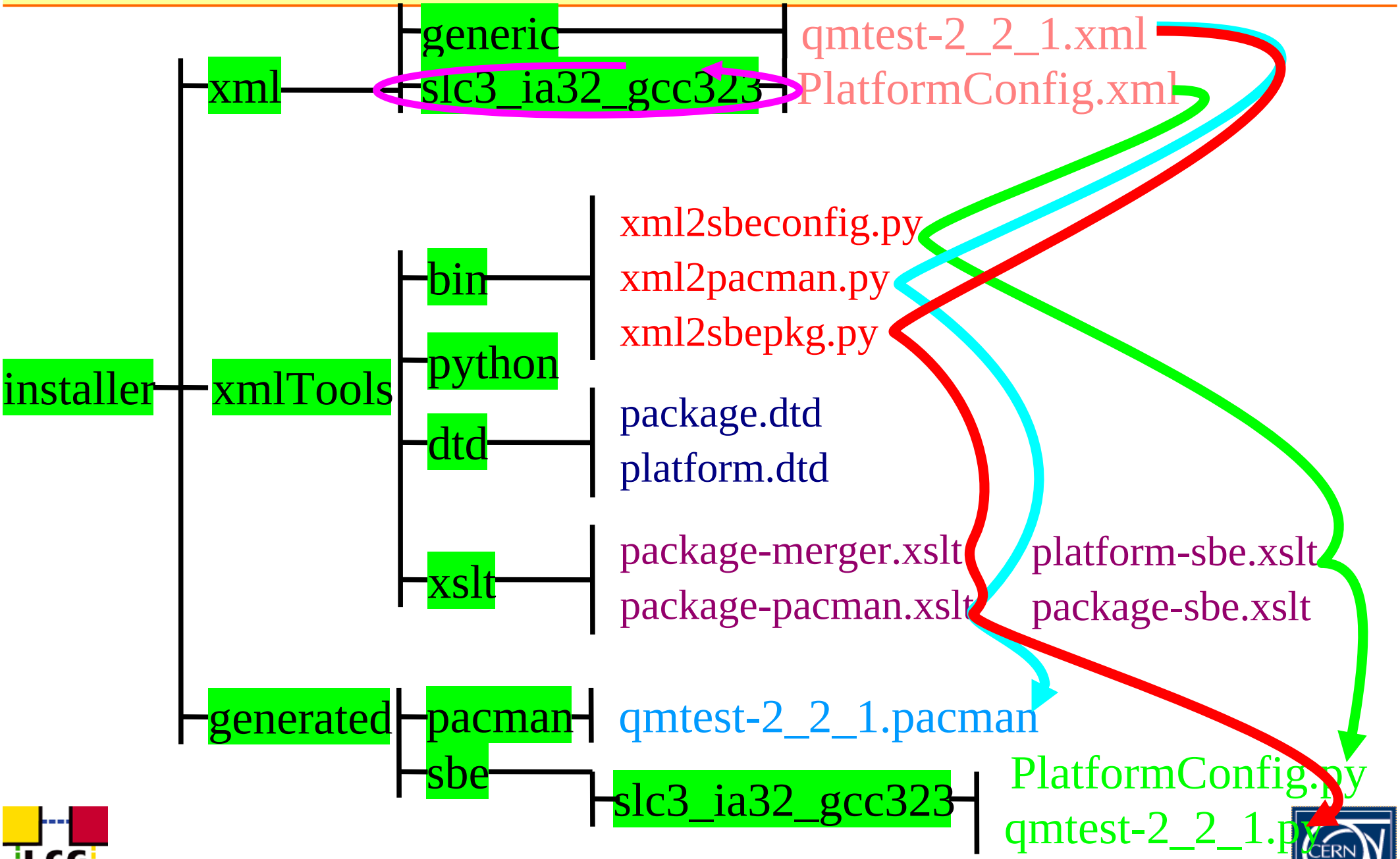
Pacman point of view



Generation of the pacman and python files



Internal Directory structure



XML file format for packages

Package root version 4.04.02, file root-4_04_02.xml

```
<?xml version="1.0"?>
<!DOCTYPE package SYSTEM "dtd/package.dtd" >
<package name="root" version="4.04.02" inherit="../root_4_03_x" >
  <info>
    <description>Package root version 4.04.02 on linux gcc
```

Informations: name, version, description...

```
    <url>http://root.cern.ch </url>
  </info>
  <source>
```

Source files: tgz and patches

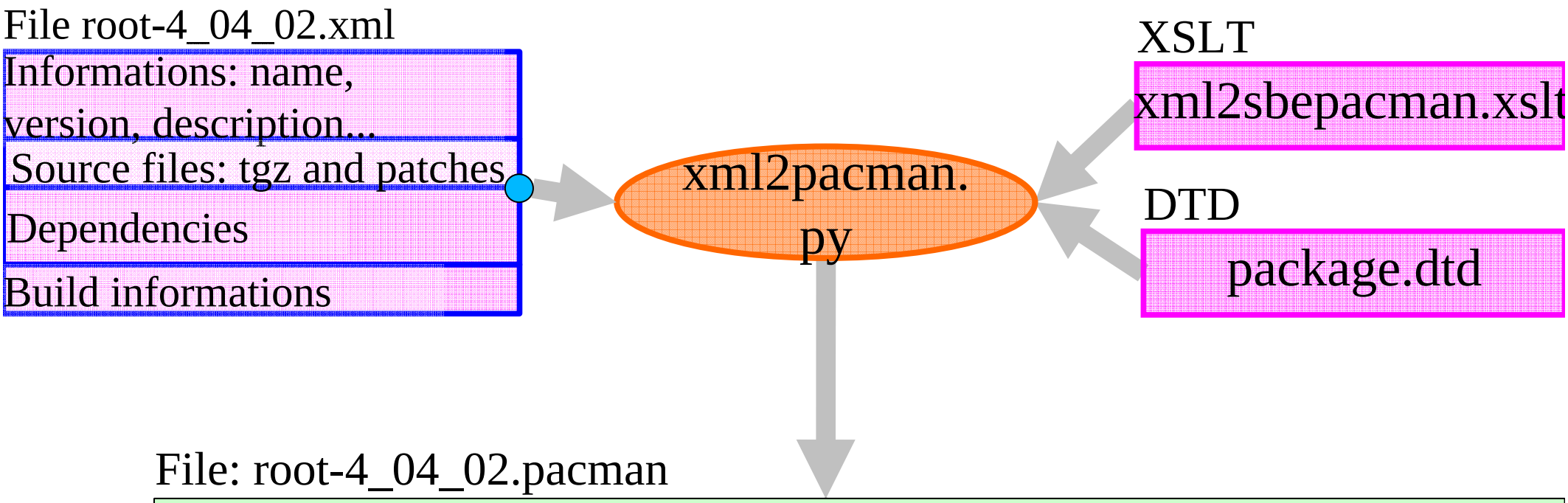
```
  </source>
  <dependency>
    <pkgreference name="dcap"   version="1.2.35" />
    <pkgreference name="mysql"  version="4.0.18" />
    <pkgreference name="python" version="2.3.4"  />
  </dependency>
```

Dependencies

```
  <build>
    <buildinfo>Trivial derivation of root_4_03_x.xml. </buildinfo>
  </build>
</package>
```

Build informations

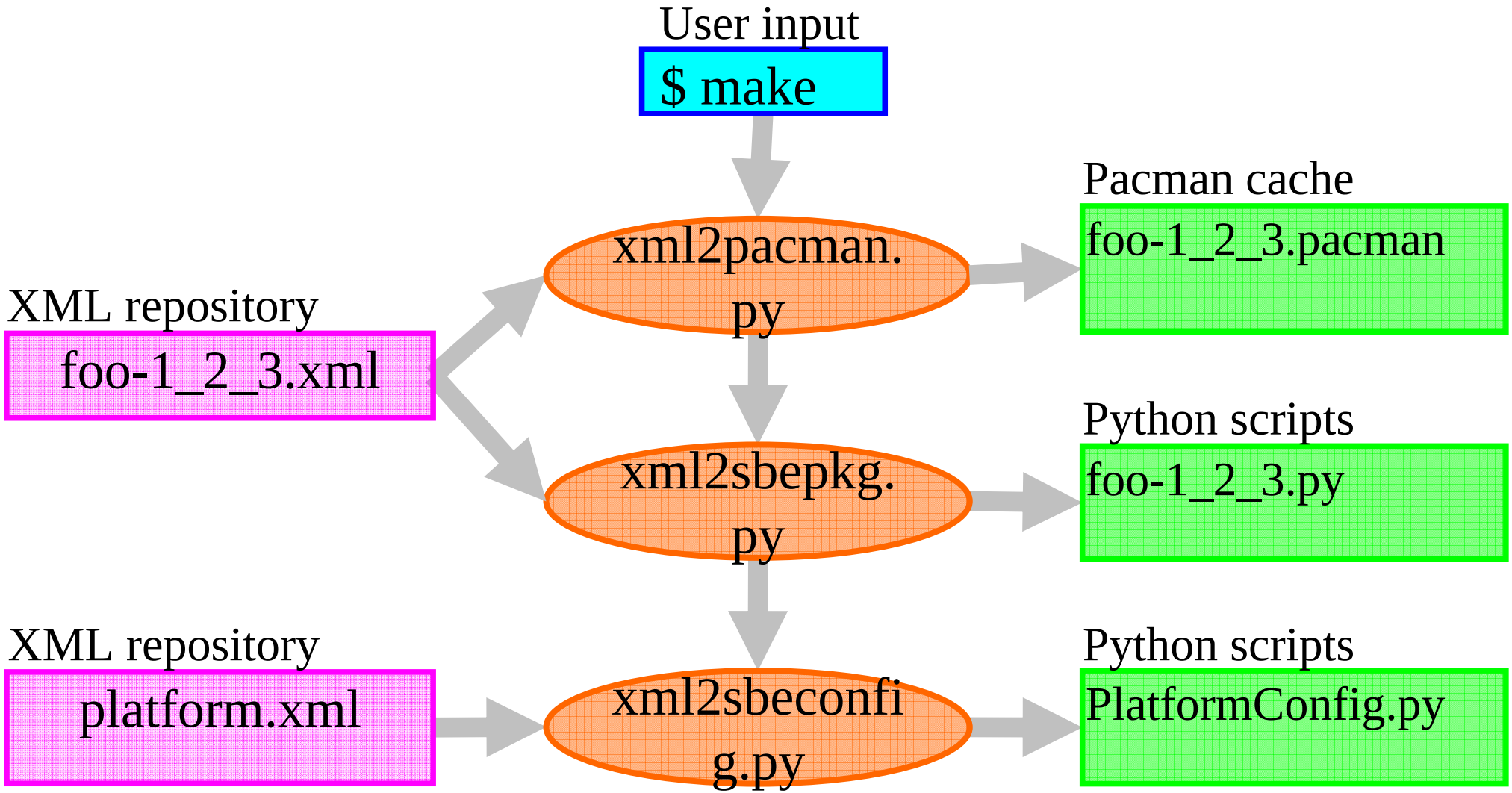
Pacman file generation (for source distribution)



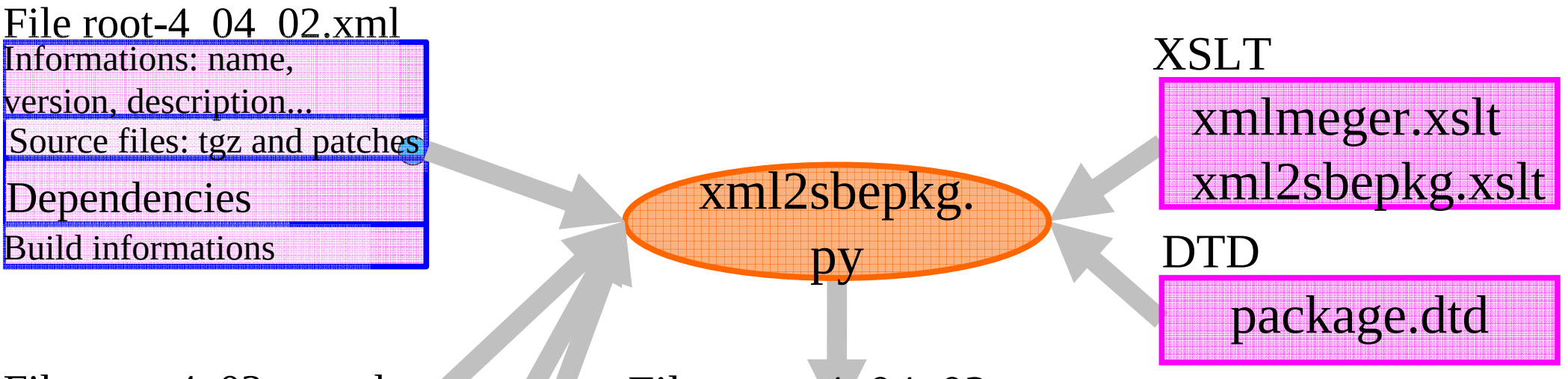
File: root-4_04_02.pacman

```
description = 'Package root version 4.04.02 on linux gcc platform.'
url('root','http://root.cern.ch')
setenvTemp('SPI_PACKAGE','root')
setenvTemp('SPI_VERSION','4.04.02')
mkdir("${SPI_PACKAGE}-${SPI_VERSION}")
cd("${SPI_PACKAGE}-${SPI_VERSION}")
downloadUntarzip('$SPI_CACHE/root_v4.04.02.source.tar.gz','SPI_BUILDDIR')
cd('$SPI_BUILDDIR')
shellDialogue('spi_build_external.py --platform=${SPI_PLATFORM} --
package=${SPI_PACKAGE} --version=${SPI_VERSION} --topdir=${SPI_TOPDIR}')
cd()
```

Generation of the pacman and python files



Pacman file generation (for source distribution)



File: root-4_04_02.py

```
class CustomBuilder(DefaultBuilder.DefaultBuilder):
    def __init__(self,platform,package,version,topdir):
        DefaultBuilder.DefaultBuilder.__init__(
            self,platform,package,version,topdir)
        return

    def Build(self):
        self.init()
        self.init2()
        self.preprocess()
        self.configure()
        self.make()
        self.make1()
        self.makeinstall()
        self.makecheck()
        self.realmakeinstall()
        [...]
```

Build instructions in XML

File: root_4_03_x.xml

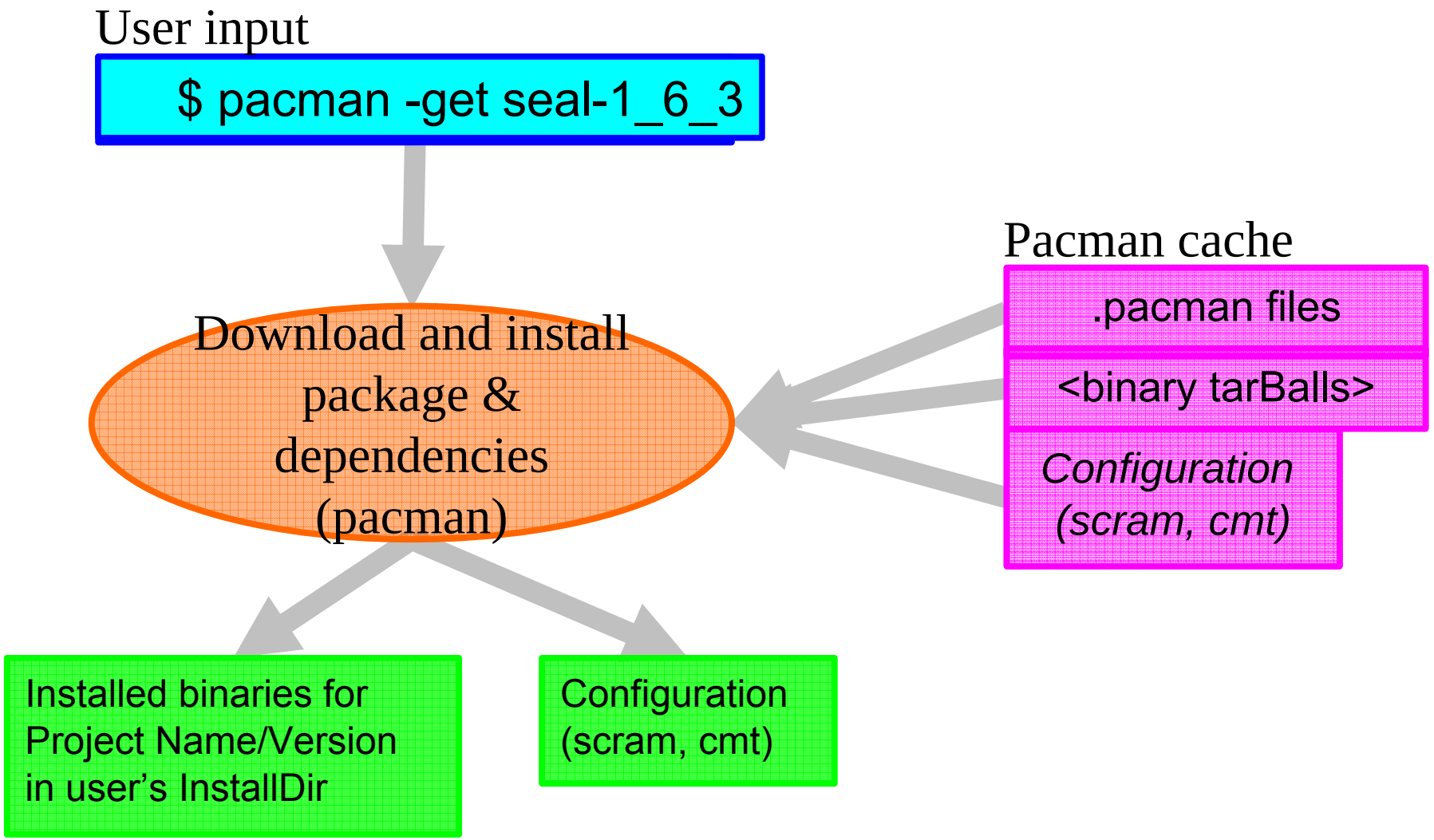
```
<sequence name="realmakeinstall" id="610">
  <buildinfo>Install go through a temporary tar file</buildinfo>
  <action type="command" name="make">
    <option value="dist" />
  </action>
  <!-- Recovering the generated tarfile name -->
  <action type="envvar" name="SPI_UNAME_S" shell="uname -s" />
  <action type="envvar" name="SPI_UNAME_R" shell="uname -r" />
  <action type="envvar" name="SPI_ROOT_TARFILE"
  shell="cd ..; echo root_*${SPI_UNAME_S}.${SPI_UNAME_R}.tar.gz"/>
  <action type="command" name="mkdir">
    <option value="-p" />
    <option value="$SPI_PREFIX" />
  </action>
  <action type="chdir" value="$SPI_PREFIX" />
  <action type="command" name="tar">
    <option value="zxvf" />
    <option value="$SPI_BUILDDIR/../$SPI_ROOT_TARFILE" />
  </action>
  <action type="chdir" value="$SPI_BUILDDIR" />
 </sequence>
</build>
</package>
```

Status of the source build system

- **Source build system has been used in production for now more than a year in three successive versions.**
- **All packages in external (except for a very few pathological cases) have been deployed using this tool since.**
- **The presented version is the newest one, that is still in a test phase (but already been used to deploy some packages).**
- **Evolution between the latest version and the previous is quite small:**
  - Directory reorganization (to avoid mixing generated files with original ones)
  - XML rewriting, merging some informations that were previously scattered in several files (like dependencies) and having a better DTD
- **This works**

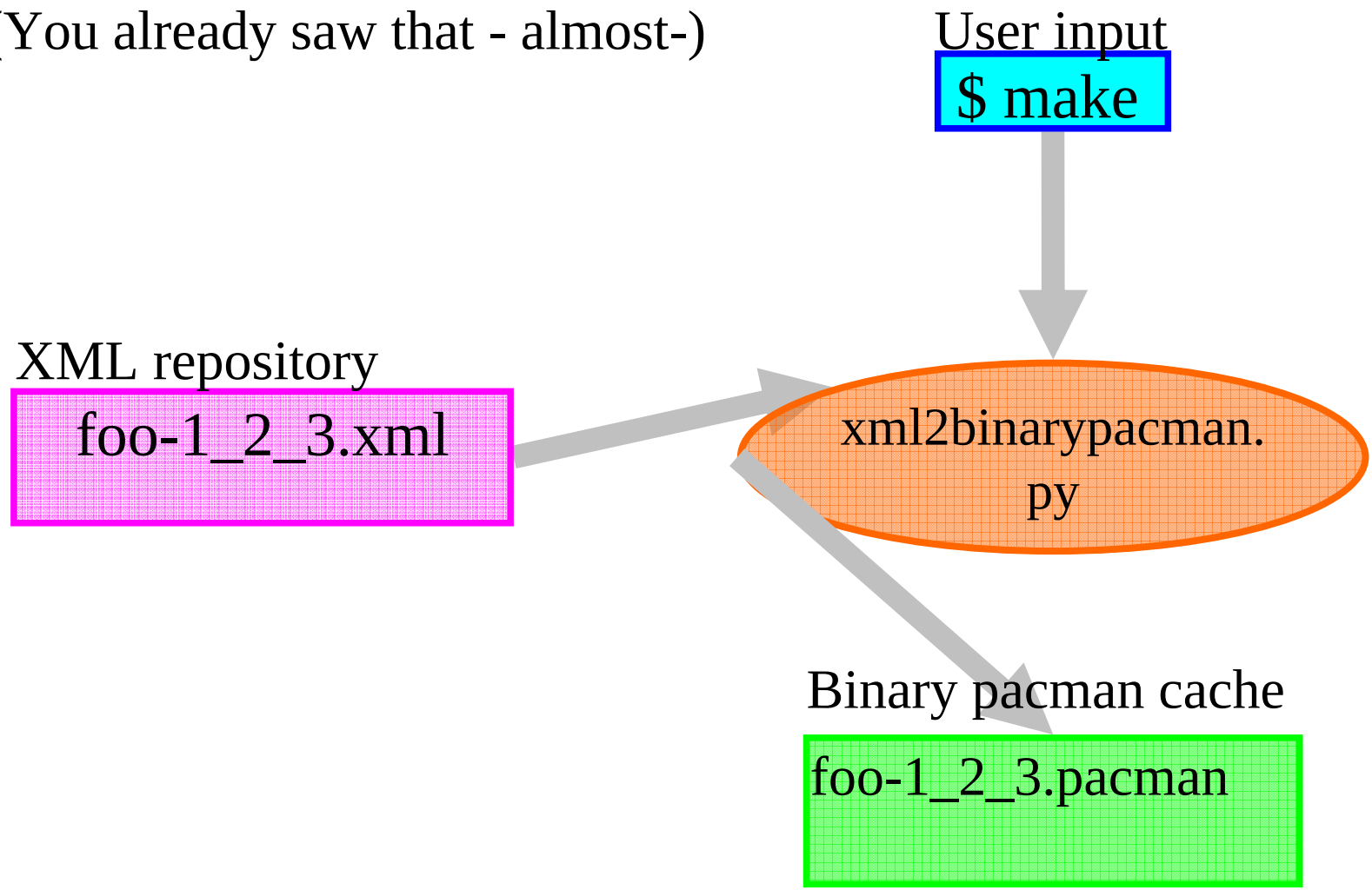
Installation of binaries with pacman

Installation of binaries: User point of view



Generation of the pacman files

(You already saw that - almost-)



Generated pacman file

root-4_04_02.pacman

```
description = 'Package root version 4.04.02 on linux gcc platform.'
url('http://root.cern.ch ')

downloadUntarzip
('root_4.04.02__LCG_slc3_ia32_gcc323.tar.gz','SPI_BUILDDIR')

  package('dcap-1_2_35')
  package('mysql-4_0_18')
  package('python-2_3_4')
```

CMT integration in pacman

- **CMT package can be installed from pacman**
- **Limitations:**
  - Explicit versions
    - Pacman doesn't yet know which version of lcgcmt to download.
    - The actual version has to be choose by the user:
    - **$ pacman -get lcgcmt-35**
  - No environment setup
    - The user have to setup it's environment.
- **Future**
  - A meta-package installing any needed CMT versions:
    - $ pacman -get lcgcmt
  - Some environment scripts ?

SCRAM integration

- **SCRAM informations can be installed from pacman**
- **Features**
  - Automatically update <AA project>/<version>/.SCRAM/<platform> information to accurate path for compiler and packages.
  - Install SCRAM and run it's configuration script.
- **Limitations:**
  - Explicit versions
    - Pacman doesn't yet know which package to post-configure.
    - The actual version has to be chosen by the user:
    - **$ pacman -get seal-1_6_0-scram-config-0_1_0**
- **Future**
  - A meta-package installing any needed SCRAM Config versions:
    - $ pacman -get scram-config

Status of Binary Distribution through Pacman

- Still a beta release.
- Not fully tested (need help).
- Support for CMT and SCRAM could be enhanced.
- Dependency granularity is coarse.
- Only some release of some packages on some platforms are available (outside of seal-1_6_0 on slc3_ia32, for now, you're on your own).
- The goal is to provide a fully maintained cache of every supported packages versions on every supported platform with a simple user interface.

Questions or comments ?

Generation of pacman file

```
<package name="qmtest" version="2.2.1" inherit="../default">
  <info>
    <description>Package QMtest version 2.2.1.</description>
    <url>
    http://www.codesourcery.com/public/qmtest/qm-2.2/qm-2.2.tar.gz</url>
    <directory name="QMtest" version="2.2.1" />
  </info>
  <source>
    <file>qm-2.2-command.py.patch</file>
    <file>testqmtest-2.2.tar.gz</file>
    <!-- Order matters: Keep this one as the last file to download. -->
    <file>qm-2.2.tar.gz</file>
```

```
description = 'Package Qmtest version 2.2.1.'
url('QMtest','
    http://www.codesourcery.com/public/qmtest/qm-2.2/qm-2.2.tar.gz)
setenvTemp('SPI_PACKAGE','QMtest')
setenvTemp('SPI_VERSION','2.2.1')
mkdir("${SPI_PACKAGE}-${SPI_VERSION}")
cd("${SPI_PACKAGE}-${SPI_VERSION}")
downloadUntarzip('$SPI_CACHE/qm-2.2-command.py.patch','SPI_BUILDDIR')
downloadUntarzip('$SPI_CACHE/testqmtest-2.2.tar.gz','SPI_BUILDDIR')
downloadUntarzip('$SPI_CACHE/qm-2.2.tar.gz','SPI_BUILDDIR')
```

package-pacman.xslt

```
<xsl:stylesheet version = '1.0'
     xmlns:xsl='http://www.w3.org/1999/XSL/Transform' >
<xsl:output method="text"/>
<xsl:template match="package">
description = '<xsl:value-of select="info/description"/>'
url('<xsl:value-of select="@name"/>','<xsl:value-of select="info/url"/>')
setenvTemp('SPI_PACKAGE','<xsl:value-of select="@name"/>')
setenvTemp('SPI_VERSION','<xsl:value-of select="@version"/>')
mkdir("${SPI_PACKAGE}-${SPI_VERSION}")
cd("${SPI_PACKAGE}-${SPI_VERSION}")
<xsl:apply-templates select="//file"/>
cd('$SPI_BUILDDIR')
shellDialogue('spi_build_external.py --platform=${SPI_PLATFORM} --package=$
{SPI_PACKAGE} --version=${SPI_VERSION} --topdir=${SPI_TOPDIR}')
cd()
</xsl:template>
<xsl:template match="//file">
downloadUntarzip('$SPI_CACHE/<xsl:value-of select="."/>','SPI_BUILDDIR')
</xsl:template>
</xsl:stylesheet>
```

Package DTD

```
<!ELEMENT package (info+,source?,dependency?,build?)>

<!ATTLIST package name     CDATA  #REQUIRED
                  version  CDATA  #REQUIRED
                  virtual  (true) #IMPLIED
                  inherit  CDATA  #IMPLIED>

<!ELEMENT info (description+,url?,directory*,partition*)>
<!ELEMENT description (#PCDATA)>
<!ELEMENT url         (#PCDATA)>
<!ELEMENT directory           EMPTY>
<!ATTLIST directory name        CDATA #REQUIRED
                   version     CDATA #REQUIRED
                   binversion  CDATA #IMPLIED
                   platform    CDATA #IMPLIED>
<!ELEMENT partition           EMPTY>
<!ATTLIST partition name     CDATA #REQUIRED
                   tgtDir   CDATA #IMPLIED
                   tag      CDATA #REQUIRED>
<!ELEMENT source (file+)>
<!ELEMENT file   (#PCDATA)>
<!ELEMENT dependency (pkgreference+)>
<!ELEMENT pkgreference EMPTY>
<!ATTLIST pkgreference name      CDATA          #REQUIRED
                       version   CDATA          #REQUIRED
                       platform  CDATA          #IMPLIED
                       type      (none|build)   #IMPLIED >
<!ELEMENT build (buildinfo*,sequence*)>
<!ELEMENT buildinfo (#PCDATA)>
<!ELEMENT sequence (buildinfo*,action*)>
```

Build instructions

```
  </info>
  <source>
    <file>bzip2-1.0.2.tar.gz</file>
  </source>
  <build>
    <buildinfo>No configure step. make and make install
    customized.</buildinfo>
    <sequence name="init2"  id="110">
      <action type="envvar" name="SPI_MAKEINSTALL_OPTIONS"
      value="PREFIX=$SPI_PREFIX" />
    </sequence>
    <sequence name="configure" id="300"></sequence>
    <sequence name="make2"     id="410">
      <action type="command" name="rm">
        <option value="-f *.o" />
      </action>
      <action type="command" name="make">
        <option value="-f Makefile-libbz2_so" />
      </action>
    </sequence>
  </build>
</package>
```

Build instructions: minimal.xml

```
<?xml version="1.0"?>
<!DOCTYPE package SYSTEM "dtd/package.dtd">
<package name="minimal" version="0.0.1">
  <info>
    <description>Minimal script when no build has to be done.</description>
  </info>
  <build>
    <sequence name="init" id="100">
      <action type="envvar" name="SPI_PREFIX"
      value="${SPI_TOPDIR}/${SPI_PACKAGE}/${SPI_VERSION}/${SPI_PLATFORM}" />
      <action type="envvar" name="SPI_BUILDDIR" shell="pwd" />
    </sequence>
    <sequence name="preprocess" id="200">
      <action type="command" name="rm">
        <option value="-rf" />
        <option value="${SPI_PREFIX}" />
      </action>
      <action type="command" name="mkdir">
        <option value="-p" />
        <option value="${SPI_PREFIX}" />
      </action>
    </sequence>
  </build>
</package>
```

Build instructions: default.xml

```
<?xml version="1.0"?>
<!DOCTYPE package SYSTEM "dtd/package.dtd">
<package name="default" version="0.0.3" inherit="../minimal">
  <info>
    <description>Default build procedure, GNU-like package. </description>
  </info>
  <build>
    <sequence name="configure" id="300">
      <action type="command" name="./configure">
        <option value="--prefix=$SPI_PREFIX" />
        <option value="$SPI_CONFIG_OPTIONS" />
      </action>
    </sequence>
    <sequence name="make" id="400">
      <action type="command" name="make">
        <option value="$SPI_MAKEOPTIONS" />
      </action>
    </sequence>
    <sequence name="makeinstall" id="500">
      <action type="command" name="make">
        <option value="install" />
        <option value="$SPI_MAKEINSTALL_OPTIONS" />
      </action>
    </sequence>
    <sequence name="makecheck" id="600">
      <action type="command" name="make">
```

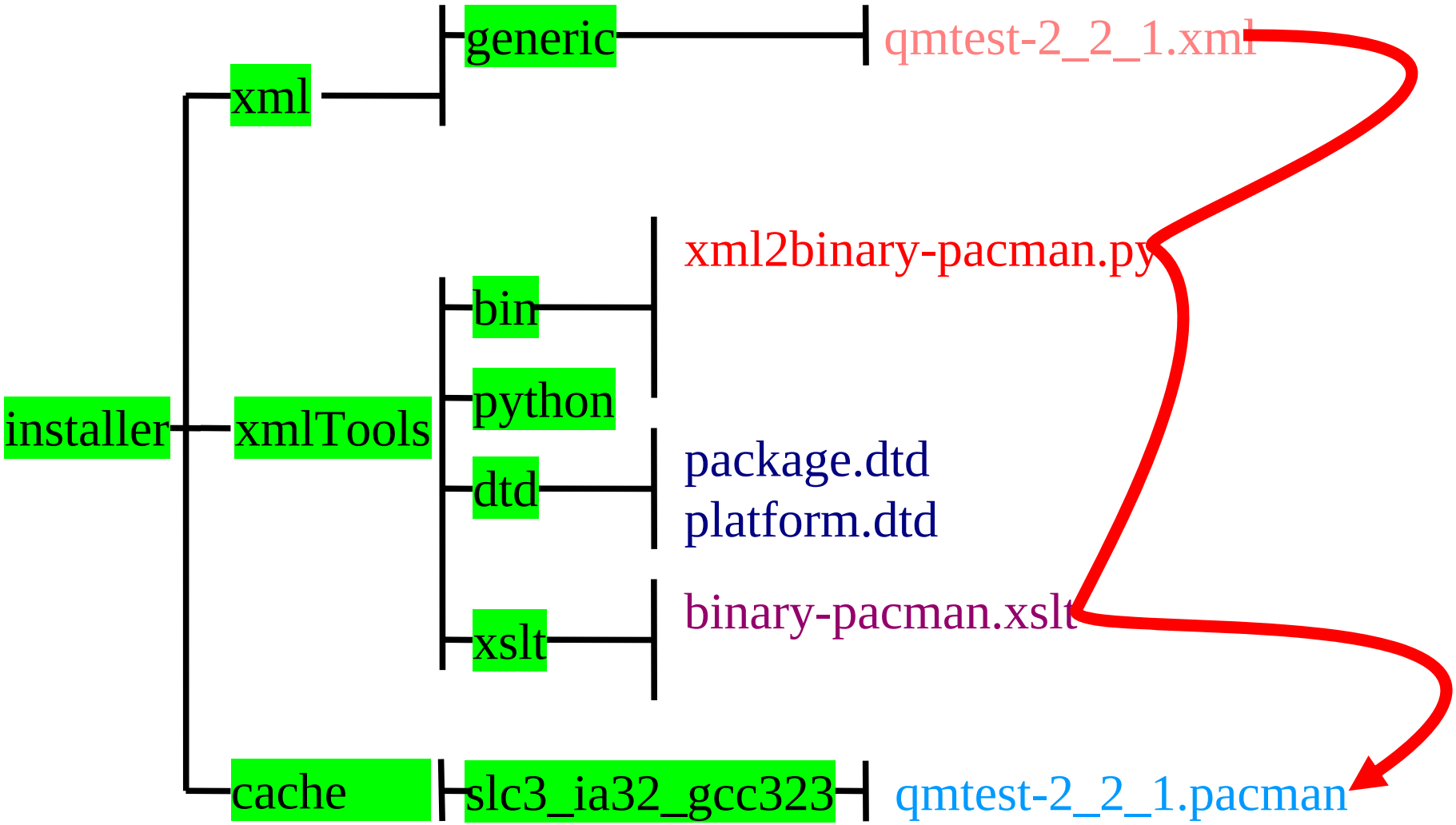
Build instructions in python: DefaultBuilder.py

```
class DefaultBuilder:
#---------------------
    def __init__(self,platform,package,version,topdir):
        Logger().commentlog("---- Build start ----")
        [...]

    def PostInstall(self):
        [...]
        ShellInterface().Execute('mkdir _SPI')
        ShellInterface().Execute('mv install.sh  _SPI')
        [...]

    def Patch(self,patch,depth=1):
        pcommand='patch -p'+str(depth)
        [...]

    def OSXLibCleanup(self,shelllibpath):
        [...]
        for lib in lst:
            [...]
                    ShellInterface().Execute(cmdline)
```

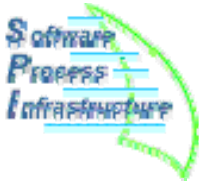
Build instructions in the final python file.

```
class CustomBuilder(DefaultBuilder.DefaultBuilder):
    def __init__(self,platform,package,version,topdir):
        DefaultBuilder.DefaultBuilder.__init__(
            self,platform,package,version,topdir)

    def Build(self):
        self.init(); self.init2(); self.preprocess(); self.configure()
        self.make()
        self.makeinstall()
        self.makecheck()
        self.PostInstall()

    def init(self):
        EnvVariableMng().Set ("SPI_PREFIX",
                              "${SPI_TOPDIR}/${SPI_PACKAGE}/${SPI_VERSION}/
${SPI_PLATFORM}")
        EnvVariableMng().Shell("SPI_BUILDDIR","pwd")

    def init2(self):
        EnvVariableMng().UsePython("2.3.4")
        EnvVariableMng().Set ("SPI_CONFIG_OPTIONS",
                              "--with-python=$SPI_PYTHON_ROOT/bin/python ")
        self.Patch("qm-2.2-command.py.patch ")

    def preprocess(self):
        ShellInterface().Execute(""" rm -rf ${SPI_PREFIX}  """)
```

Binary package distribution



Software configuration and dependencies



- **XML description to have a simple to parse format**
  - ***Build information***
  - ***Dependency description***
- **Build and install for external packages and LCG software**
- **Aim: Generate the configurations for the build and release systems needed by the users**
  - SCRAM support
  - LCGCMT support
  - Config/make based solution
- **We don't want to make one more build and configuration system but generate for the existing ones**

Actions performed to fulfill the recommendation
Build System and Infrastructure (II)

- XML description allows to support several build systems
- Generate and keep up to date configuration files for CMT
- Config/make solution is going to be achieved via the XML description files that he had to implement in order to support several tools for build and distribution

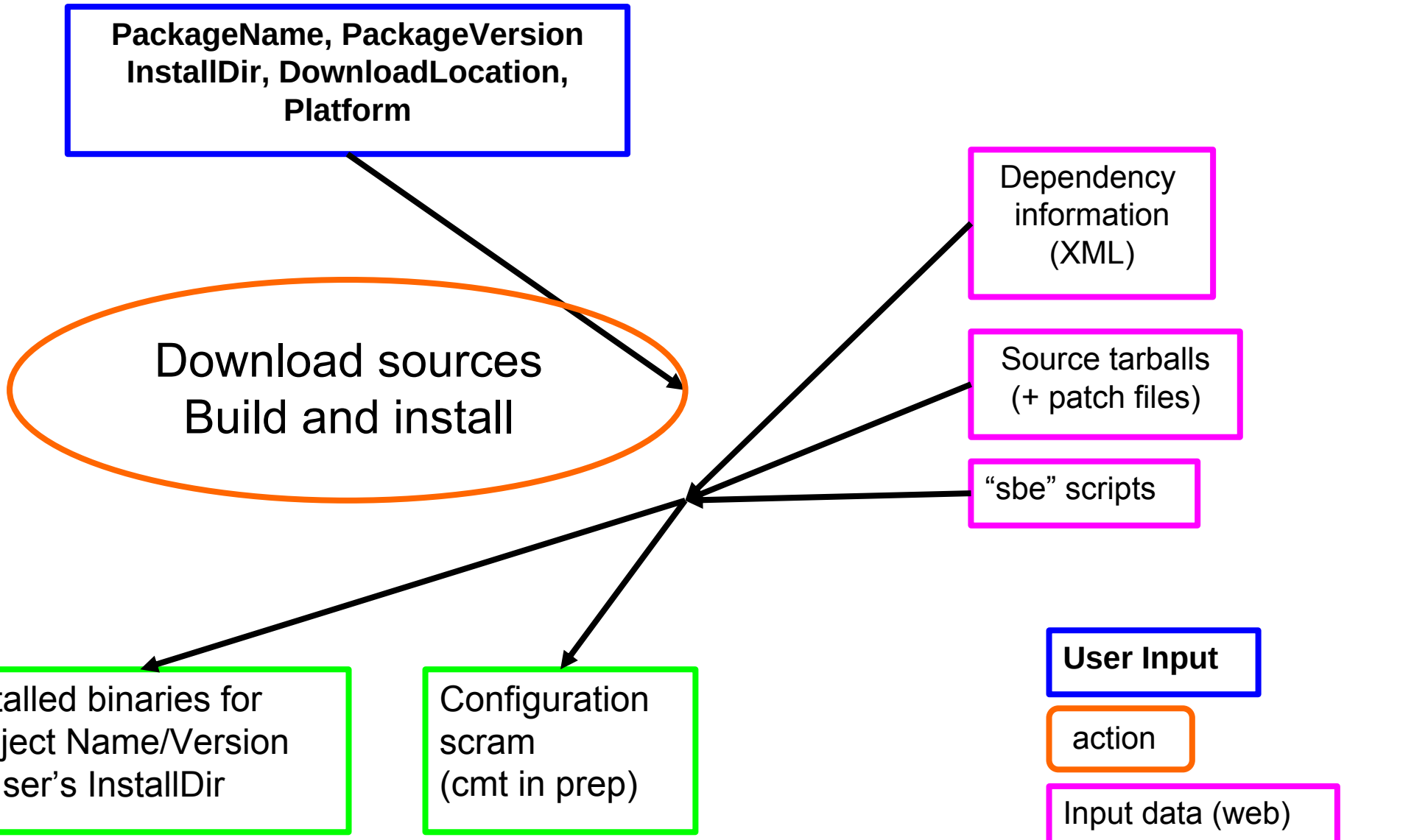
Software Distribution from Sources

- **Some packages have complex settings and/or dependencies**
  - Mysql, root, python
- **Need for consistent treatment of configure and compiler options across all external packages in use for project**
  - On project side this is done through scram
- **Need to accommodate several install/build systems**
  - Pacman, python scripts in use today
  - Rpm, apt, for the future ?
- **Decide to use independent XML format to describe the build process**
  - Use of XSLT (+ few python scripts) to transform this information into python scripts (“SPI build externals”, “sbe”) (Yannick Patois)
    - typically one per package/version

Software Distribution from Sources (II)

- **External treatment of dependency information (XML files)**
  - one per package/version, plus “meta packages”
    - e.g. boost-1.31.0.xml, external-lcg33.xml
  - Decouples build information from dependecy treatment
    - More flexible when dealing with installers
  - Python scripts resolves dependency hierarchy and provides build order
    - SPI configurator
- **Simple script to build all (“seti.py”)**
  - Resolve dependencies and get build order
    - SPI configurator
  - Call “sbe” for each external package neede
  - Build project(s) using scram “as usual”
  - First version ready
    - Documented in wiki: https://uimon.cern.ch/twiki/bin/view/SPI/SetiHowTo

Installation from sources



Actions Summary
Build System and Infrastructure (I)

- **Role of a central librarian**
  - Started in Summer 2004
- **Common build settings, release procedures**
  - Scripts for Linux, Mac
  - Different set of scripts for Windows (Pere)
- **Share of common build tools not developed within the projects**
  - Developed scripts to build and install project releases at CERN
    - Main difference from “outside”: several platforms
- **Build centrally the software and free developers in projects**
  - Done since late summer
- **Have prereleases available to the users**
  - Done. Users need to tell us when they don’t need the prerelease any more (to limit support requests to the projects)

Actions Summary
Build System and Infrastructure (II)

- **Investigate a config/make solution**
  - Postponed due to other (higher) priority issues
  - Do it inside the projects (with partitioning)?
- **Clarify long term strategy**
  - To be done (this review ?)
- **Support other build system used by other experiments (CMT)**
  - LCGCMT is distributed and maintained by SPI
- **Software Distribution**
  - More interaction with LCG deployment and check existing distribution tools (e.g. pacman)
    - Contacts with LCG deployment established but no common solutions adopted yet
    - Pacman has been investigated
      - Cache prepared for users
      - Problems using it internally
  - Customize distribution by platform and component
    - Ongoing work

Possible future improvements

- **Complete automation of procedures**
  - With documentation
  - Needed in “porting activities”
    - Gcc 3.4.3, 64 bit, Tiger, …
- **Provide working pacman caches**
  - Source and binaries
  - Feedback from expts/users needed
- ***Generate* configuration for scram and CMT**
  - from XML information
- **Provide other deployment formats used in expts ?**
  - Rpm, apt, … ?
  - Binary only or also from source ?