FCC Software stack building with Spack

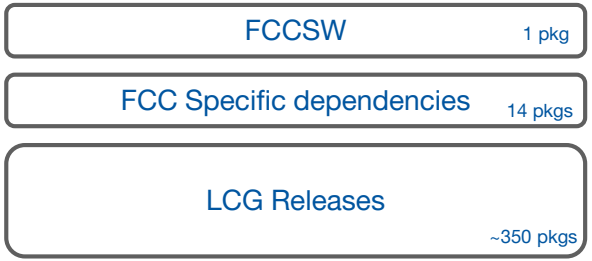
Javier Cervantes Villanueva
EP-SFT

FCC Software Stack

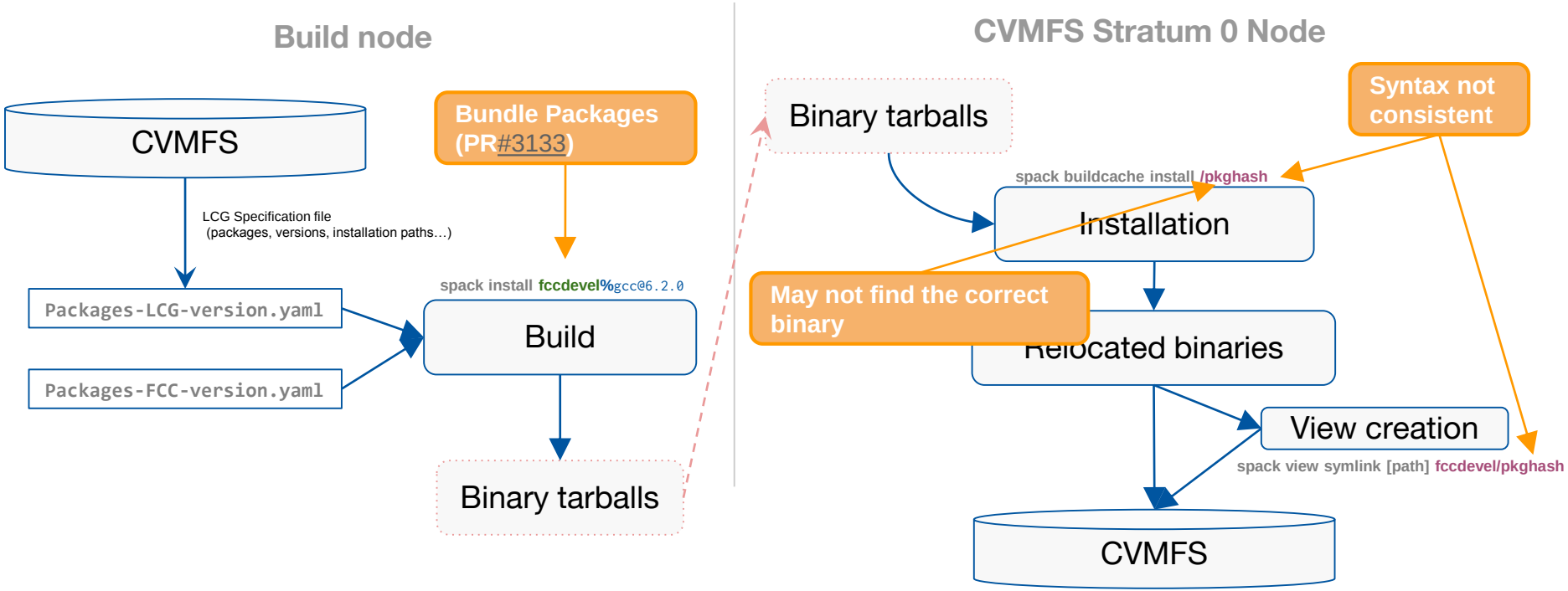
FCC software is built against LCG's CVMFS installation

- First step: create `packages.yaml` file describing LCG specs
  - Using LCG compiler and packages description files
- Allows to build with Spack against LCG stack
- Specific package versions might replace LCG ones
- Package definitions on github:
  - Separate `HEP-FCC/fcc-spack` for fcc-specific packages
  - Using `HEP-SF/hep-spack` as a basis
  - Spack `builtin`

Emulating incremental build



Workflow


Build node
CVMFS
Bundle Packages (PR#3133)
LCG Specification file (packages, versions, installation paths…)
Packages-LCG-version.yaml
Packages-FCC-version.yaml
spack install fccdevel%gcc@6.2.0
Build
Binary tarballs
CVMFS Stratum 0 Node
Binary tarballs
Syntax not consistent
spack buildcache install /pkghash
Installation
May not find the correct binary
Relocated binaries
View creation
spack view symlink [path] fccdevel/pkghash
CVMFS

Next goal: Speed up builds

Reduce redundant work repeated every day

Incremental way to not build what it's already installed in CVMFS

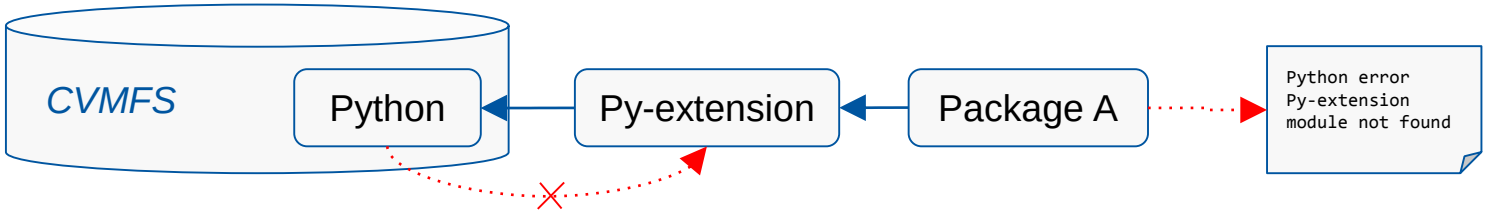
- Options
  - Custom `packages.yaml` file (LCG Releases approach)
    - Manually scan through different CVMFS paths (not scalable)
    - Does not consider different hashes
  - Binary repository with all possible binaries/combinations installed so far
    - Still requires download and installation of binaries
  - Read from remote `opt/spack/.spack-db/index.json`
    - Not writable from CVMFS
    - Not easy to synchronize

- Desirable
  - Given an external path, automatically find out and consider matching hashes
  - Discovered packages might be linked as they were specified in the `packages.yaml`

Main limitations

- Taking packages from CVMFS get in conflict with the concept of Python extensions
  - `spack active py-numpy`
  - Creates a link inside `opt/linux/python` (problem for read-only systems)

- Running tests at installation process requires a **view** to prepare the environment, so it needs to be done as a post-installation step.



Various setups needed

Prepared infrastructure to provide setup:

- Tailored python scripts to create `packages.yaml` from external sources
- `packages.yaml`: To define versions and external packages
- `compilers.yaml`: To define custom compiler locations
- `config.yaml`: To define installation path (in CVFMS)
- `mirrors.yaml`: To define `buildcache` repo

Conclusions

- FCC software infrastructure is currently built and deployed using Spack
  - Spack fully covers our basic requirements
    - Build on top of the LCG releases
    - Build and install in different nodes
    - Manage different stack of versions
  - Additional scripts needed to complete the workflow

- Most of the limitations can be worked around, although not always with the best solution

- Further optimizations during the build process are needed since they are crucial to maintain large sets of packages in an efficient way

Open discussion

- Is it worth to have a higher and more generic layer of software to configure Spack in order to build and maintain a large stack of packages?
  - How do I build all the packages in *debug* mode?
- How do you manage the Spack configuration when building/installing a full software stack?
  - Do you use any incremental approach?
- How do you distribute software using Spack?
  - *CVMFS? Build and install in the same node?*

Further examples of processes and workflows using Spack are essential to see if the list of desired features is completed.