Porting legacy software packages to the Conda Package Manager

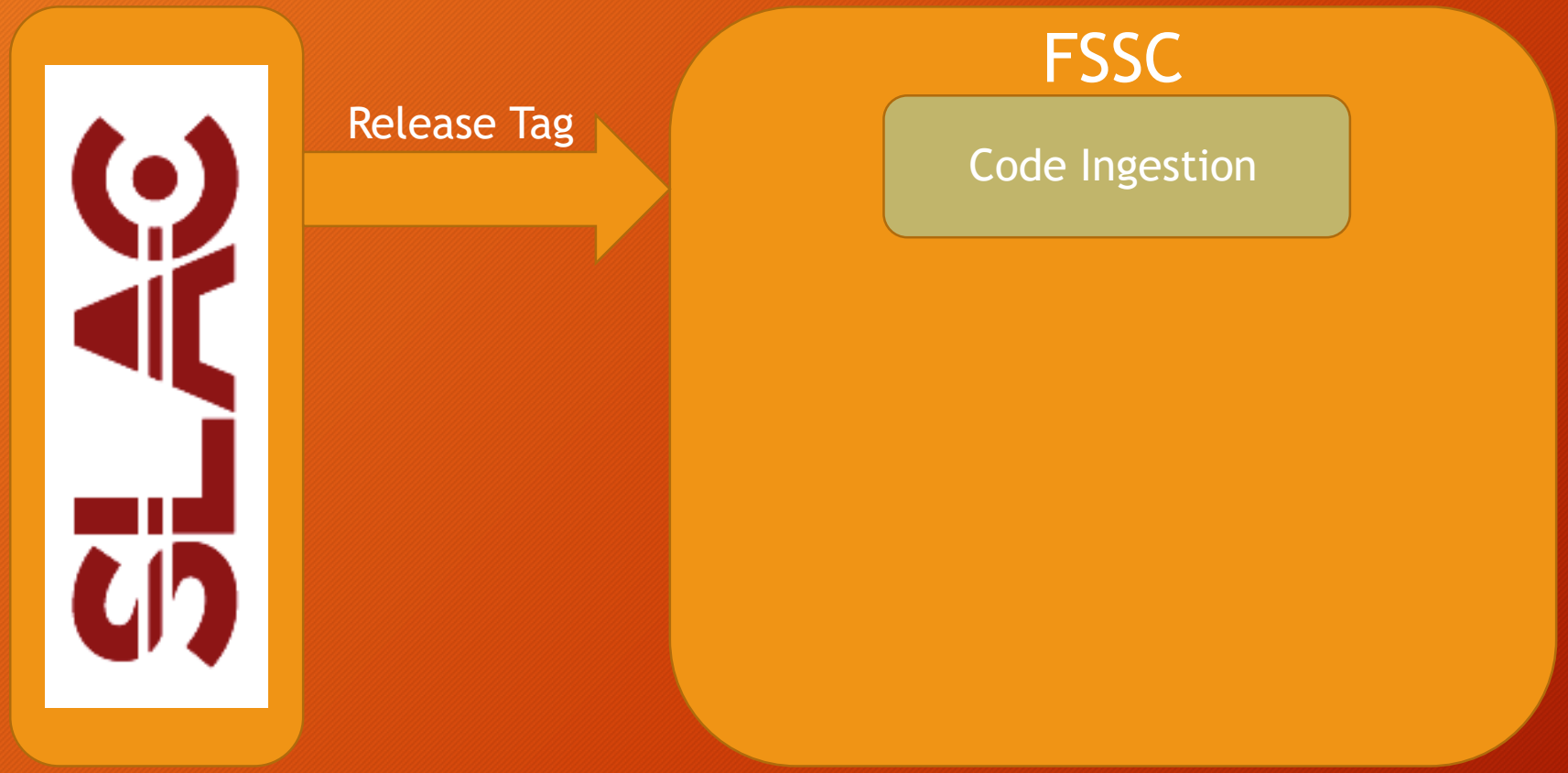
Joe Asercion, Fermi Science Support Center, NASA/GSFC

Previous Process

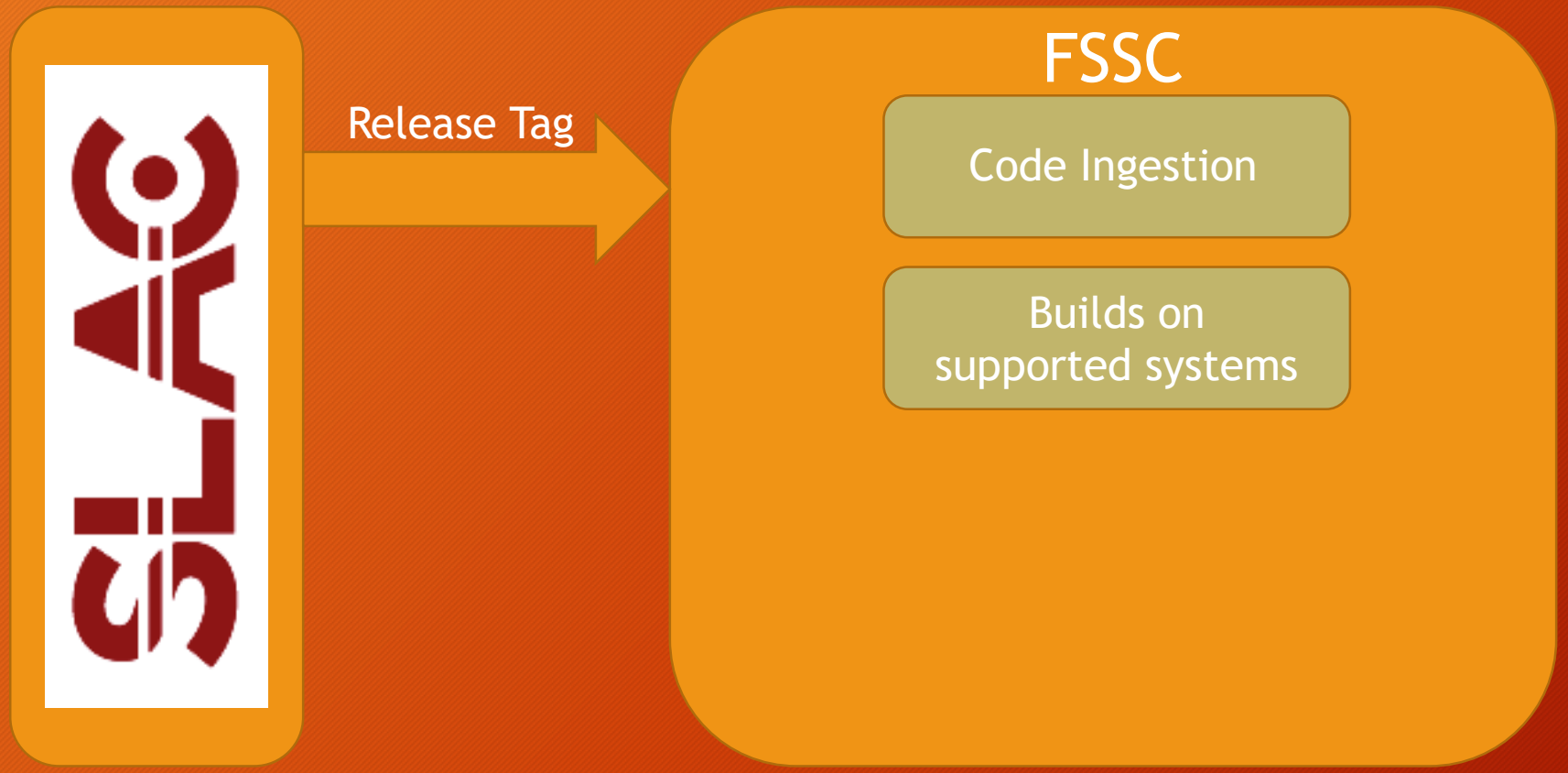
FSSC

Previous Process

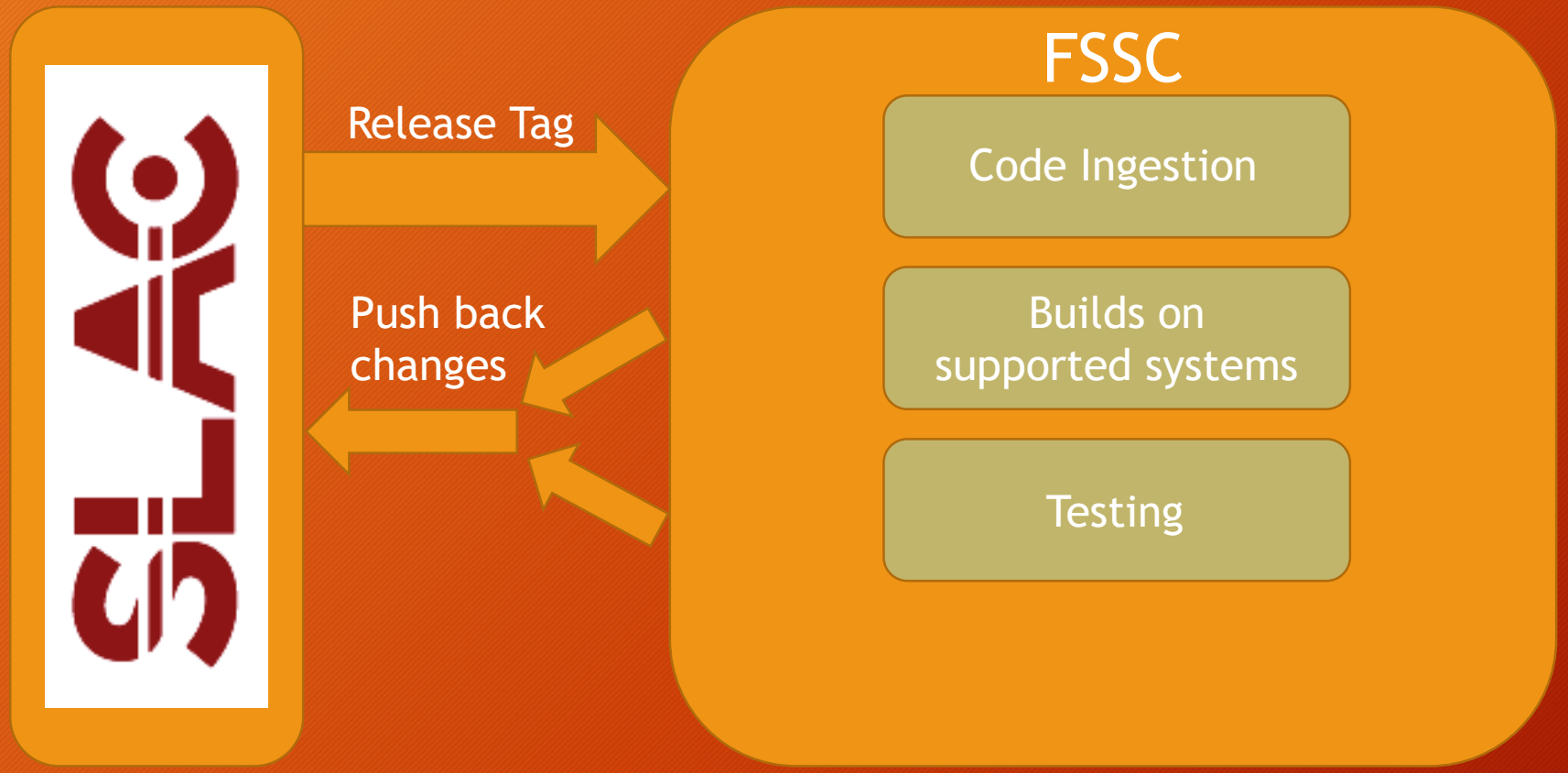
Previous Process



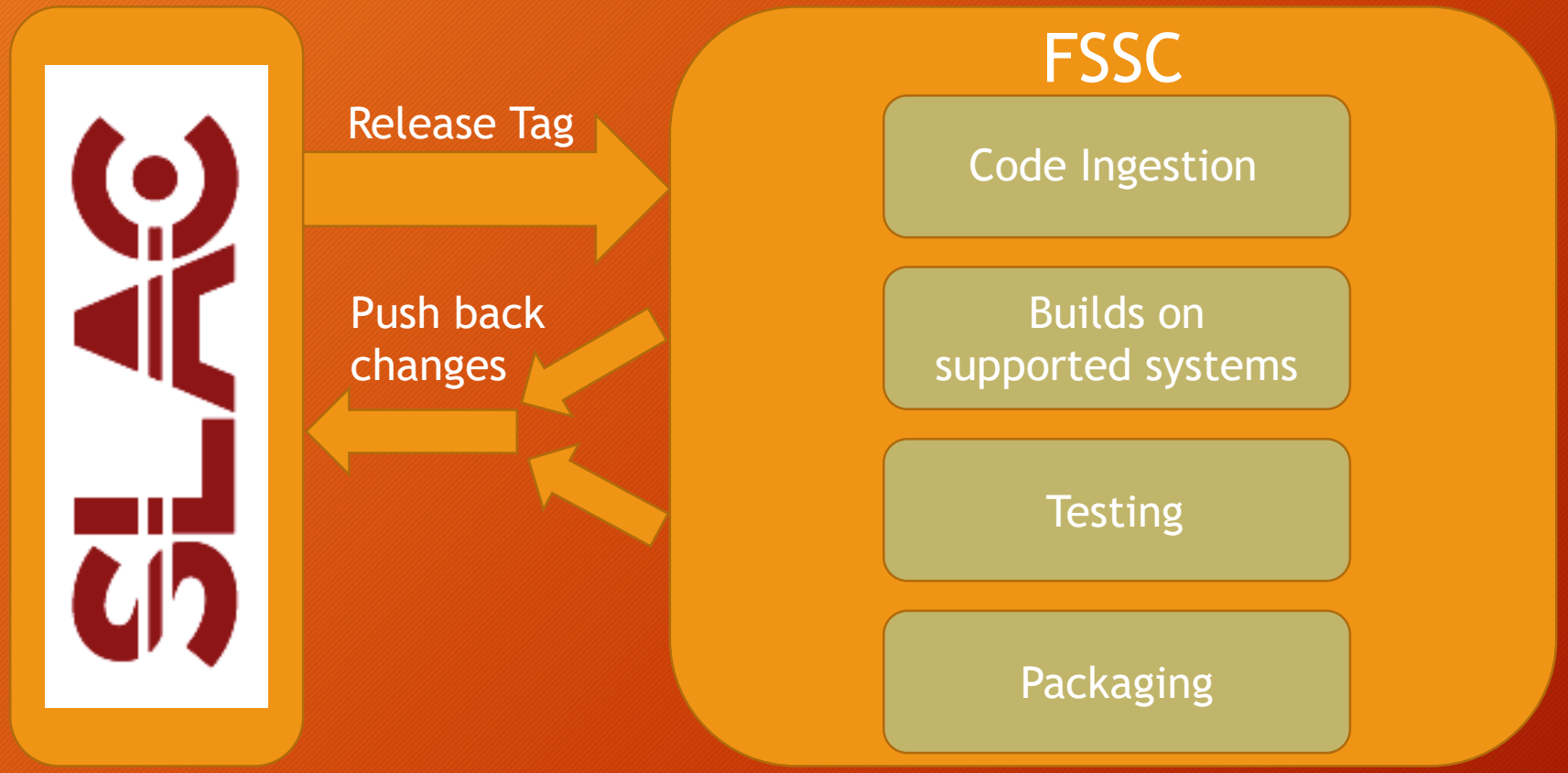
Previous Process



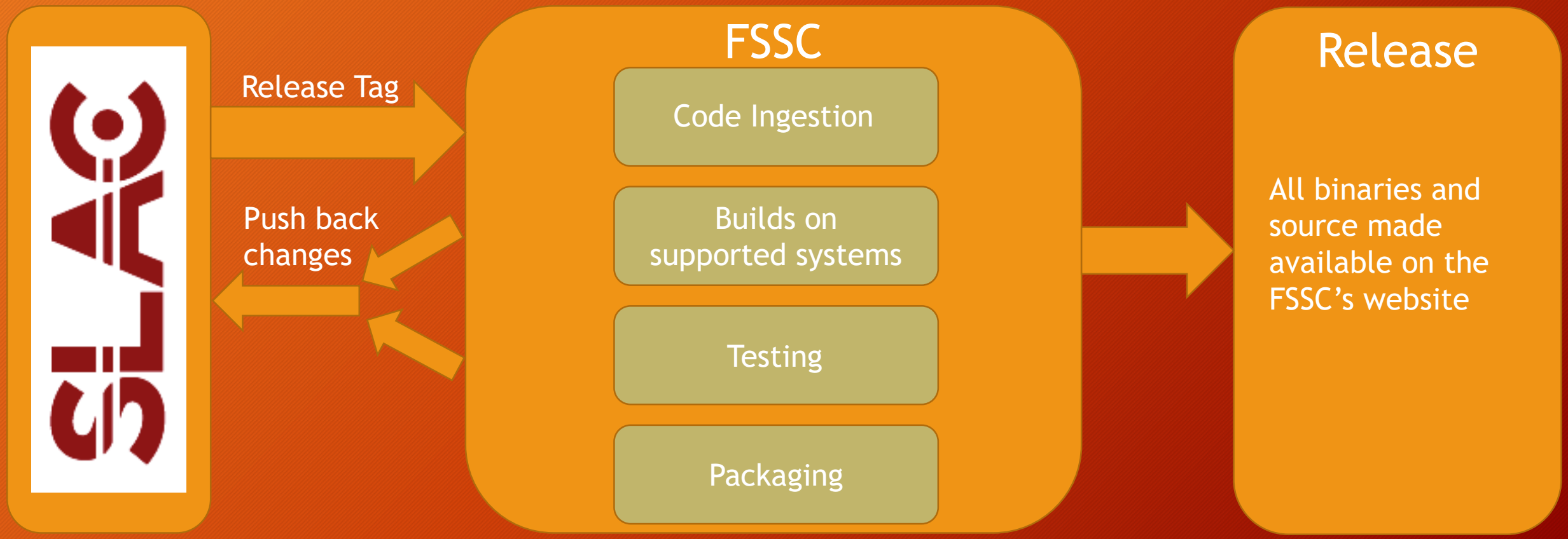
Previous Process



Previous Process



Previous Process



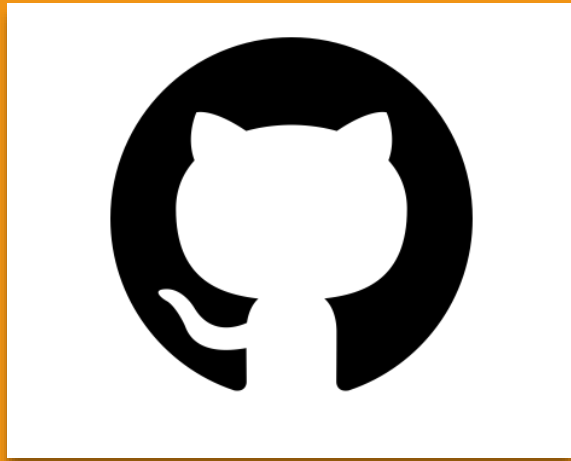
Previous Process

Issues

- Very long development cycle
- Many bottlenecks
- Increase in build instability
- Duplication of effort
- Large download size
- Difficult dependency management
- Frequent library collision errors with user machines
- Large number of individual binaries to support

Goals of Process Overhaul

- Continuous Integration/Release Model
  - Faster report/patch release cycle
  - Increased stability in the long term
- Increased Automation
- Increase process efficiency
- Increased Process Transparency
- Improved user experience
  - Better dependency management



Conda Package Manager

- Languages: Python, Ruby, R, C/C++, Lua, Scala, Java, JavaScript
- Combinable with industry standard CI systems
- Developed and maintained by Anaconda (a.k.a. Continuum Analytics)
- Variety of channels hosting downloadable packages

Packaging with Conda

Conda Build

Meta.yaml

- Contains metadata of the package to be build
- Contains metadata needed for build
  - Dependencies, system requirements, etc
- Allows for staged environment specificity
- Large amount of customization available
- Jinja2 macros

Build.sh/bld.bat

- Executed during the build stage
- Written like standard build script
- Ideally minimalist
- Any customization not handled by meta.yaml can be implemented here.

Meta.yaml

```yaml
{% set name = "fermitools" %}
{% set version = "1.0.2" %}

package:
  name: {{ name|lower }}
  version: {{ version }}

source:
  git_url: https://github.com/fermi-lat/ScienceTools.git

build:
  number: {{ environ.get('BUILD_NUMBER', 0)}}
  skip: true  # [win]
  skip: true  # [py3k]

requirements:
  build:
    #- {{ compiler('c') }}
    #- {{ compiler('cxx') }}
    #- {{ compiler('fortran') }}
    - gcc 4.8.5 # [linux]
    #- scons 2.*
    #- fermi-repoman

  host:
    #- {{ compiler('c') }}
    #- {{ compiler('cxx') }}
    #- {{ compiler('fortran') }}
    - ape 2.9
    - aplpy
    - astropy
    - blas 1.1
    - clhep 2.4.1.0
    - cppunit 1.14.0
```

Custom Variables

Package versioning metadata

Source code handling

Build process parameters

Build stage dependencies

Host stage dependencies

```yaml
  run:
    - ape 2.9
    - aplpy
    - astropy
    - blas 1.1
    - clhep 2.4.1.0
    - cppunit 1.14.0
    - f2c 0.0.1
    - fftw 3.3.8
    - fermitools-data >=0.11 # This must be updated with every model release
    - gsl 2.2
    - healpix_cxx 3.31
    - matplotlib
    - ncurses <6.0
    - numpy
    - openblas 0.2.19|0.2.19.*
    - pmw 2.0.1
    - python 2.7
    - pyyaml
    - readline 6.2
    - root5 5.34.38
    - scipy
    - wcslib 5.18
    - xerces-c 3.2.0

test:
  imports:
    - UnbinnedAnalysis
    - pyLikelihood

  commands:
    # These tests inspect linkages and libraries. They are very slow, so comment them out
    # for a quicker turnaround when testing
    #- conda inspect linkages -p $PREFIX {{ name }}  # [not win]
    #- conda inspect objects -p $PREFIX {{ name }}  # [osx]
```

Run stage dependencies

Test handling

Defining Package Metadata

Only mandatory sections: package/name & package version

Package

Descriptive metadata for the package. Name and version string must be included.

Source

Conda can pull from multiple sources natively.

- url
- Local path
- Git
- Hg
- Svn

Conda can handle aggregate builds from multiple source locations

Build

Specifies metadata which describes the build itself including build targets

Build number is used to specify new builds of the same version

Conda Build Environments

Build

- Meant for all low-level build system libraries needed for the compilation
- Anything that provides ‘sysroot’ files
- These packages need to be able to run on the build machine but be capable of outputting builds for a target platform

Host

- Host was added to represent packages “that need to be specific to the target platform when the target…is not necessarily the same as the native build platform.”
- Fermi lists the vast majority of its dependencies here
- ’Host’ and ‘Build’ prefixes are almost always separate except in a few specific cases

Run

- Packages required to run the package
- Installed automatically when the built package is installed
- Stored as metadata in the binary which is distributed via the Anaconda Cloud
- Good practice is to pin required versioning information to the package.
- As of Conda-Build 3 you can augment packages in the build and host sections with the ’run_exports’ header to negate the need for this section. (Weak vs. Strong)

Conda Build Environments

Build

- Meant for all low-level build system libraries needed for the compilation
- Anything that provides ‘sysroot’ files
- These packages need to be able to run on the build machine but be capable of outputting builds for a target platform

Build Machine

Host

- Host was added to represent packages “that need to be specific to the target platform when the target…is not necessarily the same as the native build platform.”
- Fermi lists the vast majority of its dependencies here
- ’Host’ and ‘Build’ prefixes are almost always separate except in a few specific cases

Run

- Packages required to run the package
- Installed automatically when the built package is installed
- Stored as metadata in the binary which is distributed via the Anaconda Cloud
- Good practice is to pin required versioning information to the package.
- As of Conda-Build 3 you can augment packages in the build and host sections with the ’run_exports’ header to negate the need for this section. (Weak vs. Strong)

Conda Build Environments

Build

- Meant for all low-level build system libraries needed for the compilation
- Anything that provides ‘sysroot’ files
- These packages need to be able to run on the build machine but be capable of outputting builds for a target platform

Build Machine

Host

- Host was added to represent packages “that need to be specific to the target platform when the target…is not necessarily the same as the native build platform.”
- Fermi lists the vast majority of its dependencies here
- ’Host’ and ‘Build’ prefixes are almost always separate except in a few specific cases

Target Machine

Run

- Packages required to run the package
- Installed automatically when the built package is installed
- Stored as metadata in the binary which is distributed via the Anaconda Cloud
- Good practice is to pin required versioning information to the package.
- As of Conda-Build 3 you can augment packages in the build and host sections with the ’run_exports’ header to negate the need for this section. (Weak vs. Strong)

Target Machine

Conda Macros

Conda build helpfully provides a number of Jinja2 functions which help automate and generalize the build process.

Common Jinja2 use cases

- Automatic compiler selection/setup
- Pinning expressions
- Templating

Conda Macros

Conda build helpfully provides a number of Jinja2 functions which help automate and generalize the build process.

Common Jinja2 use cases

- Automatic compiler selection/setup
- Pinning expressions
- Templating

```
{% set name = "fermitools" %}
{% set version = "1.0.2" %}

package:
  name: {{ name|lower }}
  version: {{ version }}

source:
  git_url: https://github.com/ferm

build:
  number: {{ environ.get('BUILD_NU
  skip: true  # [win]
  skip: true  # [py3k]

requirements:
  build:
    #- {{ compiler('c') }}
    #- {{ compiler('cxx') }}
    #- {{ compiler('fortran') }}
```

Custom Variable Setting

Variable Calling

Conda Macros

Conda build helpfully provides a number of Jinja2 functions which help automate and generalize the build process.

Common Jinja2 use cases

- Automatic compiler selection/setup
- Pinning expressions
- Templating

```
1  {% set name = "fermitools" %}
2  {% set version = "1.0.2" %}
3
4  package:
5    name: {{ name|lower }}
6    version: {{ version }}
7
8  source:
9    git_url: https://github.com/ferm
10
11 build:
12   number: {{ environ.get('BUILD_NU
13   skip: true  # [win]
14   skip: true  # [py3k]
15
16 requirements:
17   build:
18     #- {{ compiler('c') }}
19     #- {{ compiler('cxx') }}
20     #- {{ compiler('fortran') }}
```

Custom Variable Setting

Variable Calling

Shell Environment reference

Conda Macros

Conda build helpfully provides a number of Jinja2 functions which help automate and generalize the build process.

Common Jinja2 use cases

- Automatic compiler selection/setup
- Pinning expressions
- Templating

```
{% set name = "fermitools" %}
{% set version = "1.0.2" %}

package:
  name: {{ name|lower }}
  version: {{ version }}

source:
  git_url: https://github.com/ferm

build:
  number: {{ environ.get('BUILD_NU
  skip: true  # [win]
  skip: true  # [py3k]

requirements:
  build:
    #- {{ compiler('c') }}
    #- {{ compiler('cxx') }}
    #- {{ compiler('fortran') }}
```

Custom Variable Setting

Variable Calling

Shell Environment reference

Conda Compiler function

Conda Compilers

As of Anaconda 5 Conda provides a set of compilers which are recommended for use in build recipes.

Use of the Anaconda compilers helps make the recipe more agnostic with regards to the host machine.

Linux

gcc_liunx_64
g++_linux_64
gfortran_linux_64

macOS

clang_liunx_64
clangxx_linux_64
gfortran_linux_64

- Explicitly built for cross-compilation
- Customizable

Preprocessing Selectors

Preprocessing selectors can be used to specify different meta-data cases for different build environments. This is most commonly used to specify different platforms/architectures as build/dependency targets.

Almost all of the selector variables are Booleans and allow for logical statements to be passed in preprocessing. However, good practice is to employ comparison operators over specific selector variables.

Selectors always follow a target in the format 'TARGET # [SELECTOR]'

x86	True if the system architecture is x86, both 32-bit and 64-bit, for Intel or AMD chips.
x86_64	True if the system architecture is x86_64, which is 64-bit, for Intel or AMD chips.
linux	True if the platform is Linux.
linux32	True if the platform is Linux and the Python architecture is 32-bit.
linux64	True if the platform is Linux and the Python architecture is 64-bit.
armv6l	True if the platform is Linux and the Python architecture is armv6l.
armv7l	True if the platform is Linux and the Python architecture is armv7l.
ppc64le	True if the platform is Linux and the Python architecture is ppc64le.
osx	True if the platform is macOS.
unix	True if the platform is either macOS or Linux.
win	True if the platform is Windows.
win32	True if the platform is Windows and the Python architecture is 32-bit.
win64	True if the platform is Windows and the Python architecture is 64-bit.
py	The Python version as an int, such as `27` or `36`. See the CONDA_PY environment variable.
py3k	True if the Python major version is 3.
py2k	True if the Python major version is 2.
py27	True if the Python version is 2.7. Use of this selector is discouraged in favor of comparison operators (e.g. py==27).
py34	True if the Python version is 3.4. Use of this selector is discouraged in favor of comparison operators (e.g. py==34).
py35	True if the Python version is 3.5. Use of this selector is discouraged in favor of comparison operators (e.g. py==35).
py36	True if the Python version is 3.6. Use of this selector is discouraged in favor of comparison operators (e.g. py==36).
np	The NumPy version as an integer such as `111`. See the CONDA_NPY environment variable.

Build.sh

- Default: Bash Script (.bat on Windows)
- Executed in a special “Conda Build” environment
- Custom/explicit environmental variables need to be defined via meta.yaml
- Needed compiler flags are specified here
- Can use any installed scriptable build system (make, sCons, etc.)

```bash
export condaname="fermitools"

# REPOMAN! #
# Syntax Help:
# To checkout master instead of the release tag add '--develop' after checkout
# To checkout arbitrary other refs (Tag, Branch, Commit) add them as a space
#   delimited list after 'conda' in the order of priority.
#   e.g. ScienceTools highest_priority_commit middle_priority_ref branch1 branch2 ...
repoman --remote-base https://github.com/fermi-lat checkout --force --develop Science
# repoman --remote-base https://github.com/fermi-lat checkout --force --develop Scie


# condaforge fftw is in a different spot
mkdir -p ${PREFIX}/include/fftw
if [ ! -e ${PREFIX}/include/fftw/fftw3.h ] ; then

    ln -s ${PREFIX}/include/fftw3.* ${PREFIX}/include/fftw

fi

#CXXFLAGS=${CXXFLAGS//c++17/c++11}

# Add optimization
export CFLAGS="-O2 ${CFLAGS}"
export CXXFLAGS="-O2 ${CXXFLAGS}"

# Add rpaths needed for our compilation
export LDFLAGS="${LDFLAGS} -Wl,-rpath,${PREFIX}/lib,-rpath,${PREFIX}/lib/root,-rpath,

if [ "$(uname)" == "Darwin" ]; then

    #std=c++11 required for use with the Mac version of CLHEP in conda-forge
    export CXXFLAGS="-std=c++11 ${CXXFLAGS}"
    export LDFLAGS="${LDFLAGS} -headerpad_max_install_names"
    echo "Compiling without openMP, not supported on Mac"

else

    # This is needed on Linux
    export CXXFLAGS="-std=c++11 ${CXXFLAGS}"
    export LDFLAGS="${LDFLAGS} -fopenmp"

fi

ln -s ${cc} ${PREFIX}/bin/gcc

ln -s ${CXX} ${PREFIX}/bin/g++

scons -C ScienceTools \
      --site-dir=../SConsShared/site_scons \
      --conda=${PREFIX} \
      --use-path \
      -j ${CPU_COUNT} \
      --with-cc="${CC}" \
      --with-cxx="${CXX}" \
      --ccflags="${CFLAGS}" \
      --cxxflags="${CXXFLAGS}" \
      --ldflags="${LDFLAGS}" \
      all

rm -rf ${PREFIX}/bin/gcc

rm -rf ${PREFIX}/bin/g++

# Remove the links to fftw3
rm -rf ${PREFIX}/include/fftw
```

Build Environment Variables

- Specialized environmental variables defined in the Conda Build process.
- Some are inherited and some are defined by Conda

PREFIX – Path to the build directory. Used by all systems

- MacOS/Windows have unique variables

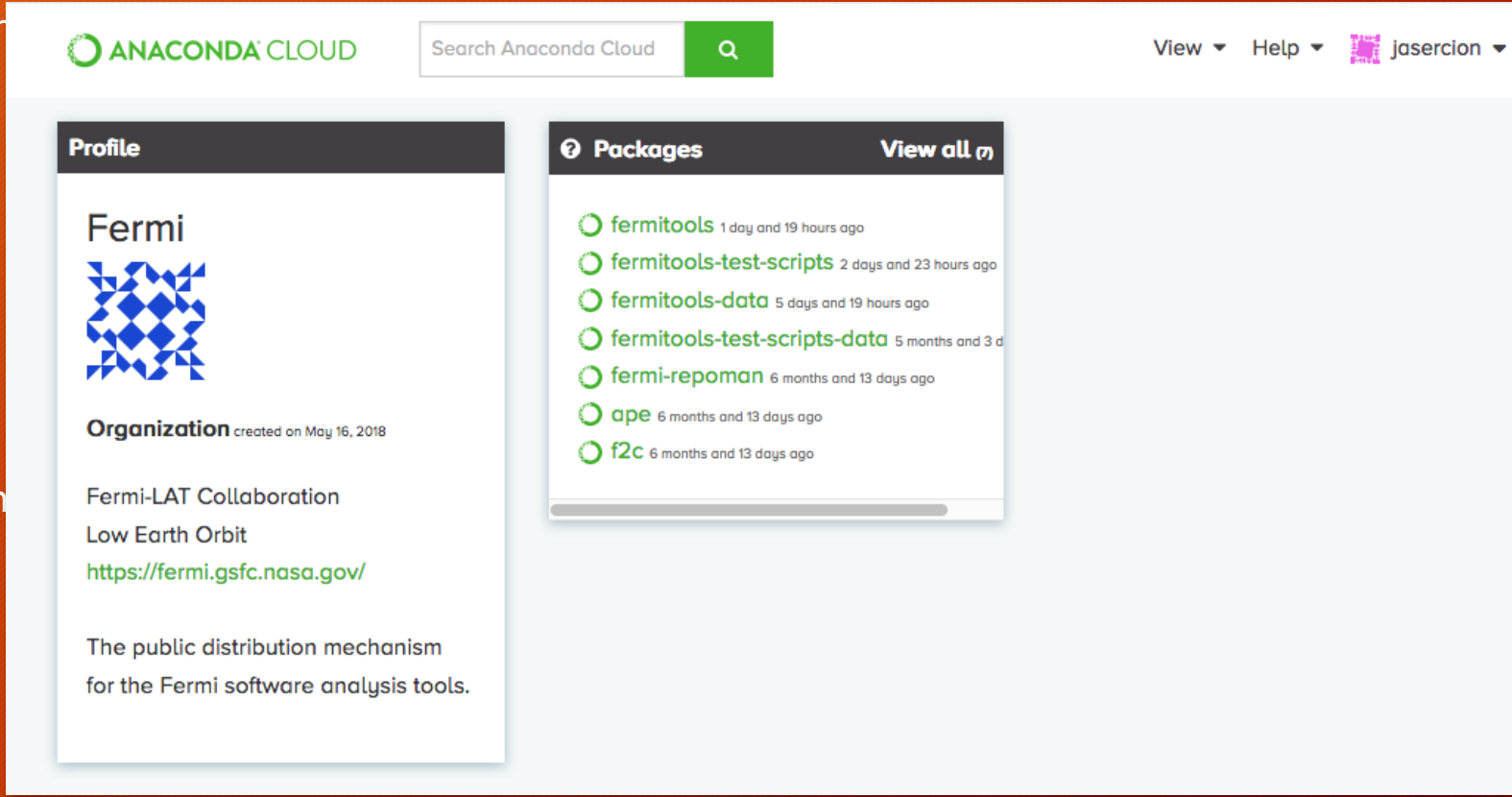
Distribution

Anaconda Cloud is the primary distribution mechanism for the Conda Package manager. Organizations can have dedicated channels to distribute software built and packaged using Conda Build.

Hosting for public projects are free. Private plans are available for a fee.

Fermi has its own organization (the Fermi Channel) which distributes software which is developed and maintained directly by the Fermi Science Support Center

Conda-Forge is another such organization.

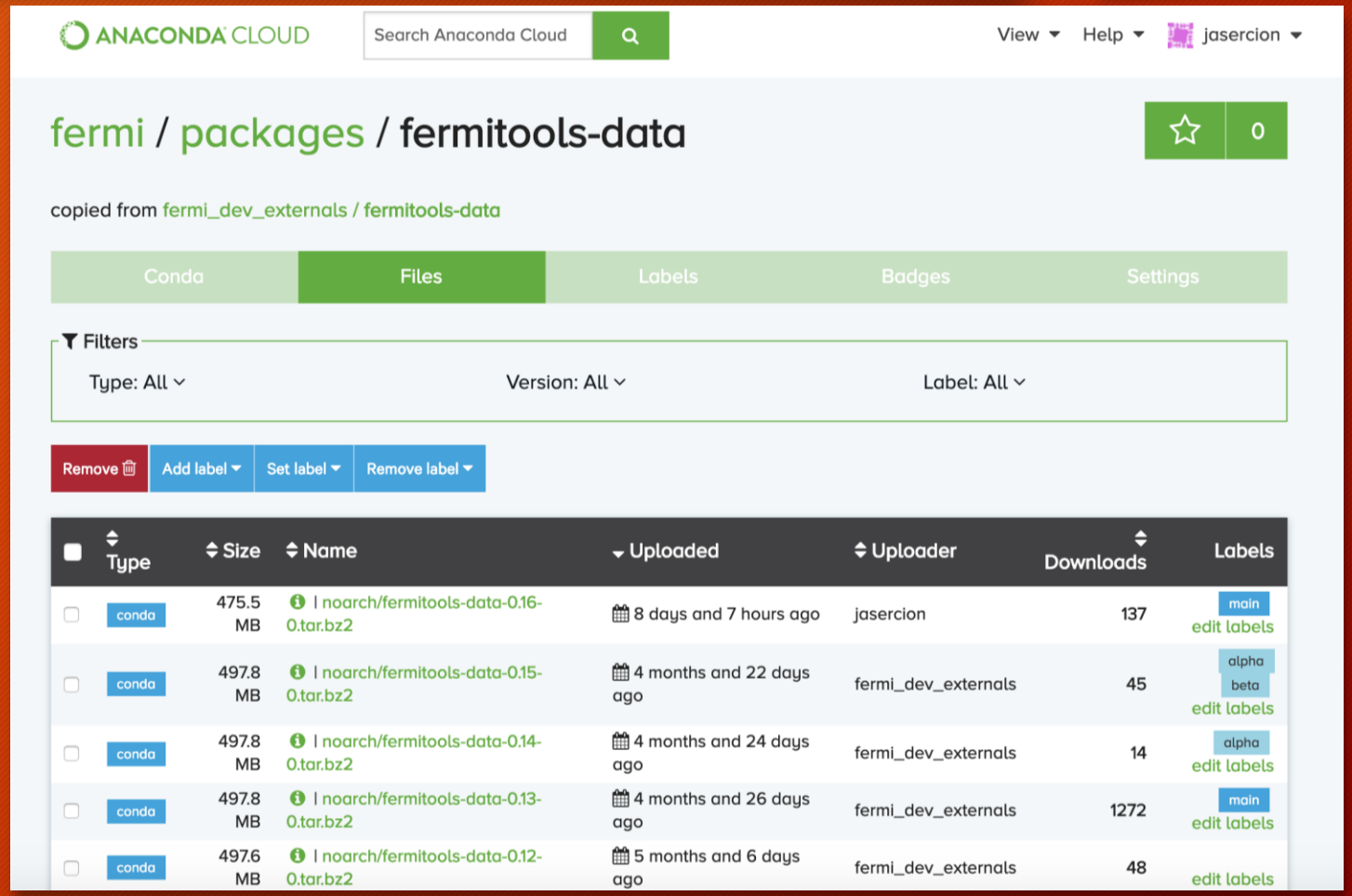


Anaconda Channels

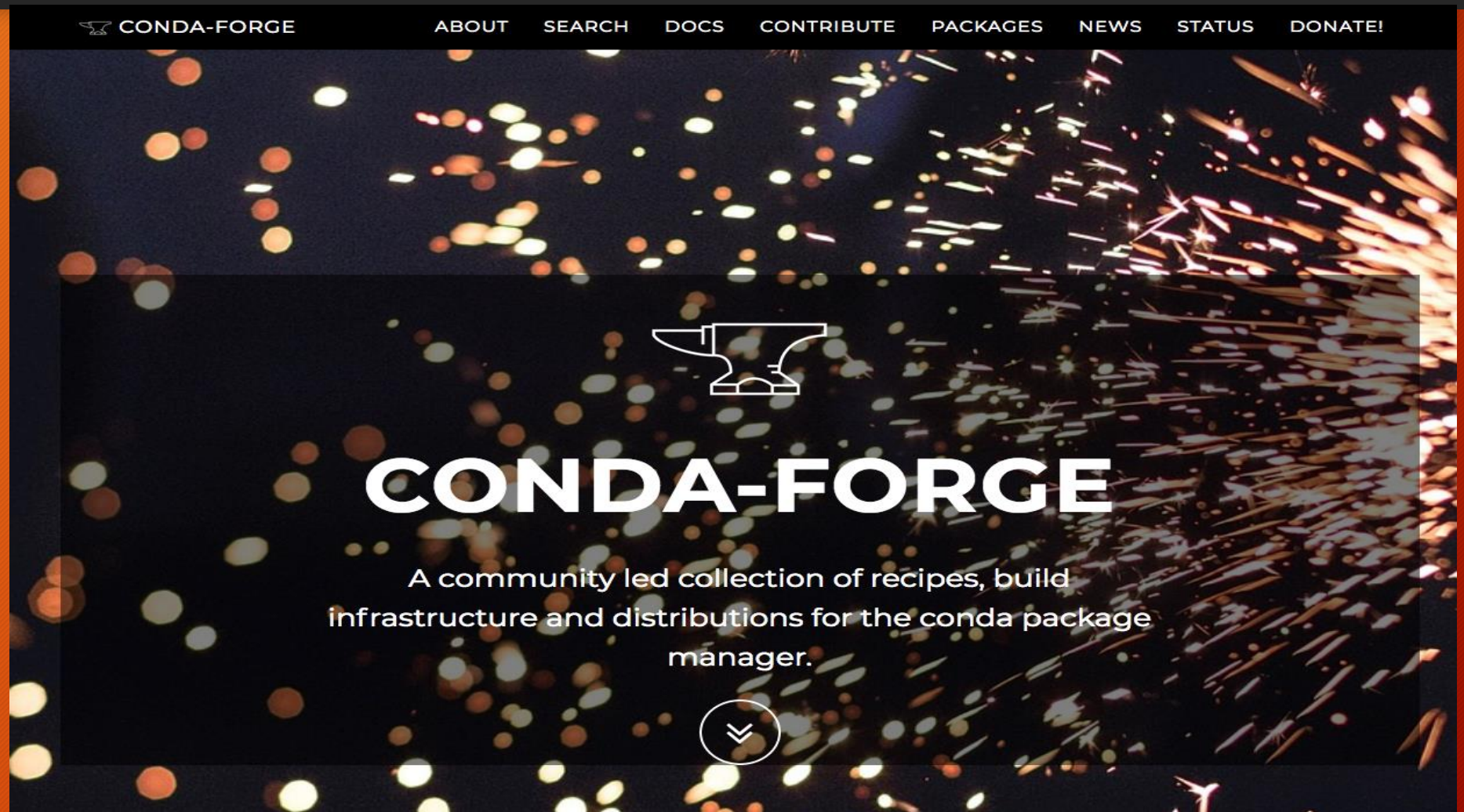
Channels organize packages by the user or group of users (Organization) that uploaded them.

Label help differentiate different different packages hosted in a channel, effectively creating ‘sub-channels’.

Label checking in Conda is strict. Including a label tag in a conda install or update command will search for matching packages with that label alone.



Conda Forge


CONDA-FORGE
ABOUT
SEARCH
DOCS
CONTRIBUTE
PACKAGES
NEWS
STATUS
DONATE!
CONDA-FORGE
A community led collection of recipes, build infrastructure and distributions for the conda package manager.

Conda Forge

Community Driven

- Est 2016
- Conda Forge seeks to expand upon the default packages uploaded by Anaconda Inc.
- Packages are maintained by their uploaders in accordance with Conda Forge build standards
- Core Conda Forge developers work to maintain stability of legacy code

Standardized Build Process

- Conda Forge uses a Github integrated CI system
- Linux -> Circle CI/Azure
- MacOS -> Travis CI/Azure

Package Centralization

- All feedstocks hosted on github in the conda-forge organization
- Compiled packages hosted on the Conda-Forge Anaconda cloud channel
- 6500+ packages currently availible

Conda Forge

Community Driven

- Est 2016
- Conda Forge seeks to expand upon the default packages uploaded by Anaconda Inc.
- Packages are maintained by their uploaders in accordance with Conda Forge build standards
- Core Conda Forge developers work to maintain stability of legacy code

Standardized Build Process

- Conda Forge uses a Github integrated CI system
- Linux -> Circle CI/Azure
- MacOS -> Travis CI/Azure

Package Centralization

- All feedstocks hosted on github in the conda-forge organization
- Compiled packages hosted on the Conda-Forge Anaconda cloud channel
- 6500+ packages currently available

This is becoming a problem!

Managed Dependencies

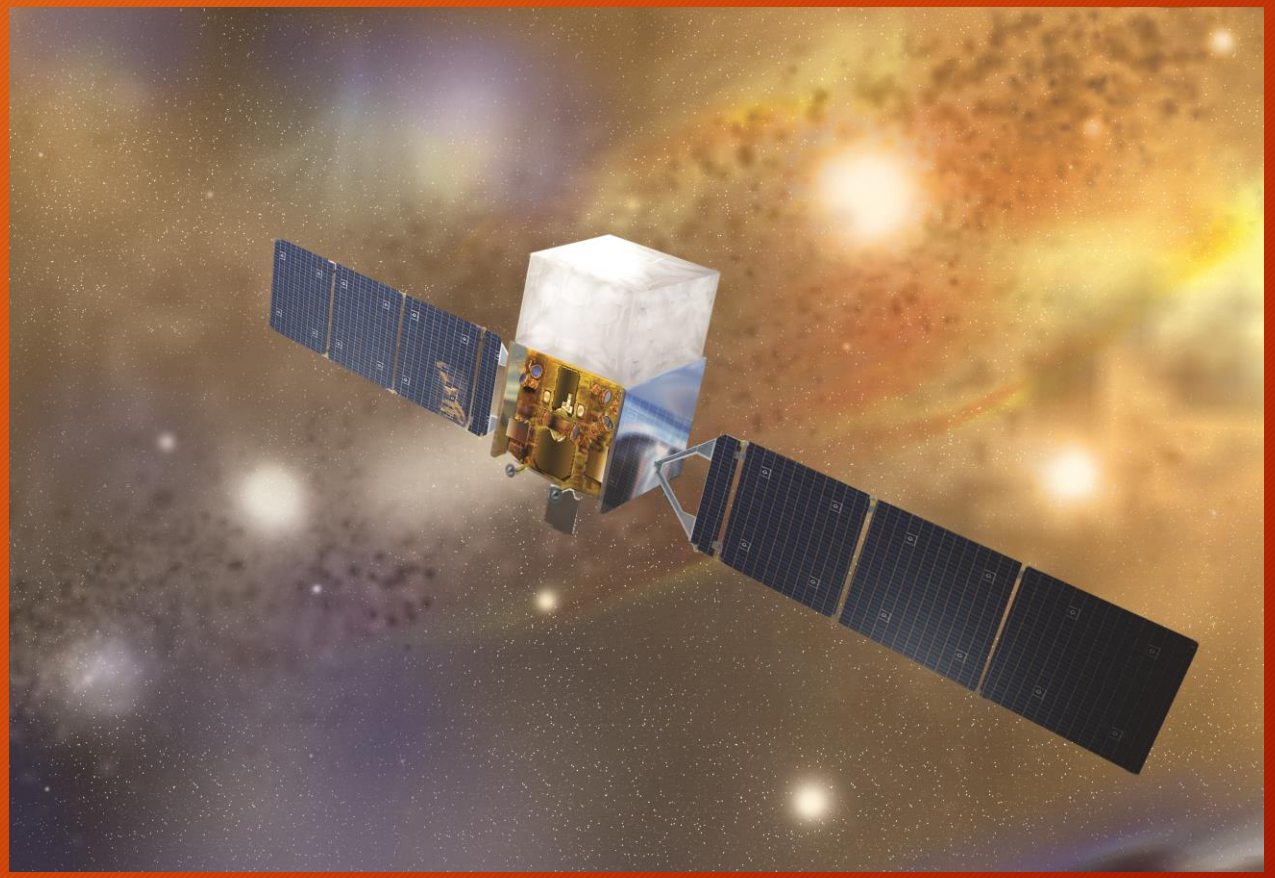
Fermi Channel

- Ape
- Fermitools Data
- Support Packages
  - Fermi Repoman
  - Fermitools Test Scripts
  - Fermitools Test Scripts Data

Conda Forge

- Root5
- F2c
- Clhep
- Healpix_cxx

The Fermi Development Pipeline



Development Process Overview

Development Cycle

Development

Commit

Push to Branch

Pull Request

Continuous Integration & Validation

Jenkins Webhooks

Docker Container Builds

Unit Testing

Alpha Packaging and Upload

Development Process Overview

Development Cycle

Development

Commit

Push to Branch

Pull Request

Continuous Integration & Validation

Jenkins Webhooks

Docker Container Builds

Unit Testing

Alpha Packaging and Upload

Pre-release Beta Generation

Linux Alpha Promotion

MacOS Beta Build/Testing

MacOS Beta Packaging/ Upload

Development Process Overview

Development Cycle

Development

Commit

Push to Branch

Pull Request

Continuous Integration & Validation

Jenkins Webhooks

Docker Container Builds

Unit Testing

Alpha Packaging and Upload

Pre-release Beta Generation

Linux Alpha Promotion

MacOS Beta Build/Testing

MacOS Beta Packaging/ Upload

Unit/Thread /HITL Validation

Scripted Unit Testing

HITL Testing

Thread Testing

Development Process Overview

Development Cycle

Development

Commit

Push to Branch

Pull Request

Continuous Integration & Validation

Jenkins Webhooks

Docker Container Builds

Unit Testing

Alpha Packaging and Upload

Pre-release Beta Generation

Linux Alpha Promotion

MacOS Beta Build/Testing

MacOS Beta Packaging/ Upload

Unit/Thread /HITL Validation

Scripted Unit Testing

HITL Testing

Thread Testing

Software Release

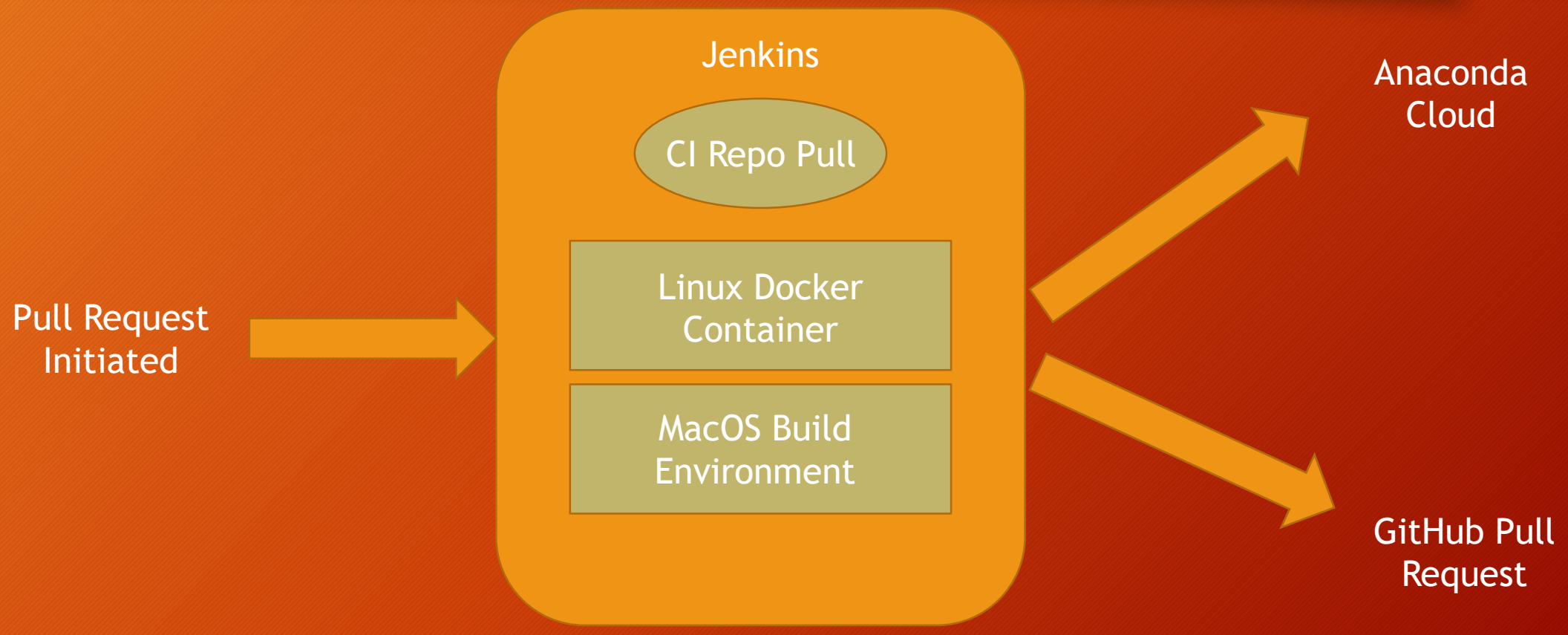
Documentation Update

Update open Github issues

Promote Beta to Main

Announce Release

CI System


Jenkins
CI Repo Pull
Linux Docker Container
MacOS Build Environment
Pull Request Initiated
Anaconda Cloud
GitHub Pull Request

CI System

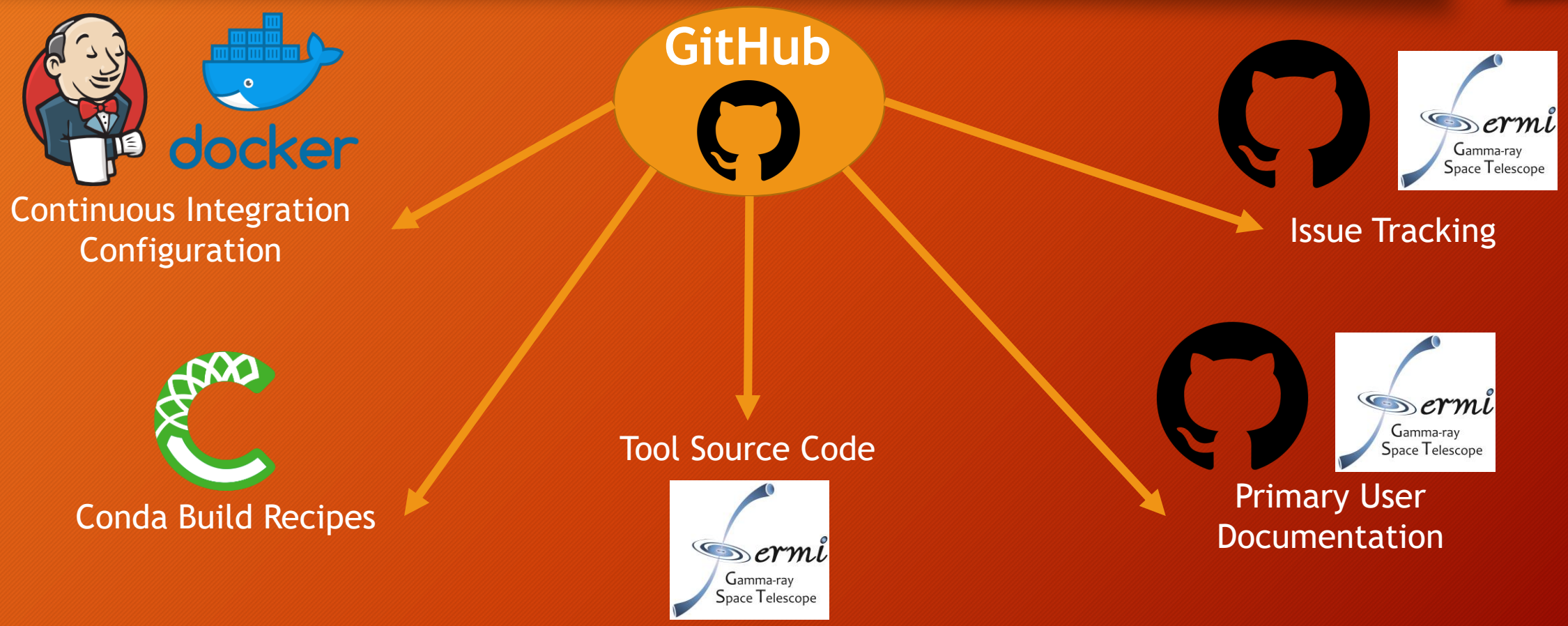
Linux Docker Container

- Customized Conda-Forge Docker container
- Based on Centos 6
- Pulled, configured, and launched on the fly during the CI cycle

MacOS Build System

- Currently physical High Sierra system at Goddard
- Exploring Mac VM/Azure Pipeline for incorporation into Jenkins CI pipeline

GitHub Integration



Meta Packages

Fermitools-Conda

- Contains all necessary Conda recipe files
- Hosts Fermitools wiki
- Primary Github documentation landing page

CI

- Contains all Jenkins CI scripts
- Webhooks script

Sciencetools (meta)

- Contains list of constituent packages for the overarching 'ScienceTools' (Fermitools) checkout
- Test list

Organization Package Management

Fermi Repoman

Custom build of the repoman collection of utilities. Allows checkouts of the entirety of the Fermitools source code. Also has limited tagging capabilities.

Alternative: Git Submodules

Fermi Model Handling

- Packaged via Conda Build for ‘NoArch’ target
- Diffuse models and other outRef files backed up on github
- Galactic Diffuse Model stored in Git LFS
  - Not an ideal solution
  - Easier to deal with copy on local storage

Large File Storage

Git LFS

- Free Tier Available
- 1 GB storage
- 1 GB Bandwidth/Month

Anaconda Cloud

- Free (Public)
- 3 GB storage space
- Files need to packaged by conda

Versioning and Tagging

- Follows standard xx.xx.xx tagging format
- <Version>.<Revision>.<Patch>
- On release all packages are tagged with a release tag in the format ‘Fermitools-xx.xx.xx’
- Allows release traceability via GitHub

Next Steps

- Incorporate Azure MacOS build environment into Jenkins pipeline
- Further modularize the Fermitools subpackages
- Increase test efficiency
- Python3/C++11 updates
- Dependency Pinning review

Thanks!