Multi-Repository Development and Integration with TriBITS

Roscoe A. Bartlett

Oak Ridge National Laboratories

Trilinos User Group Meeting

October 29, 2014

Motivations and Outline

Outline:

- Overview of CASL VERA Development Efforts
- Multi-Repository Configuration and Building
- Multi-Repository Version Control and Repository Management
- Multi-Repository Integration Models and Processes
- Future TriBITS Development

Overview of CASL VERA Development Efforts

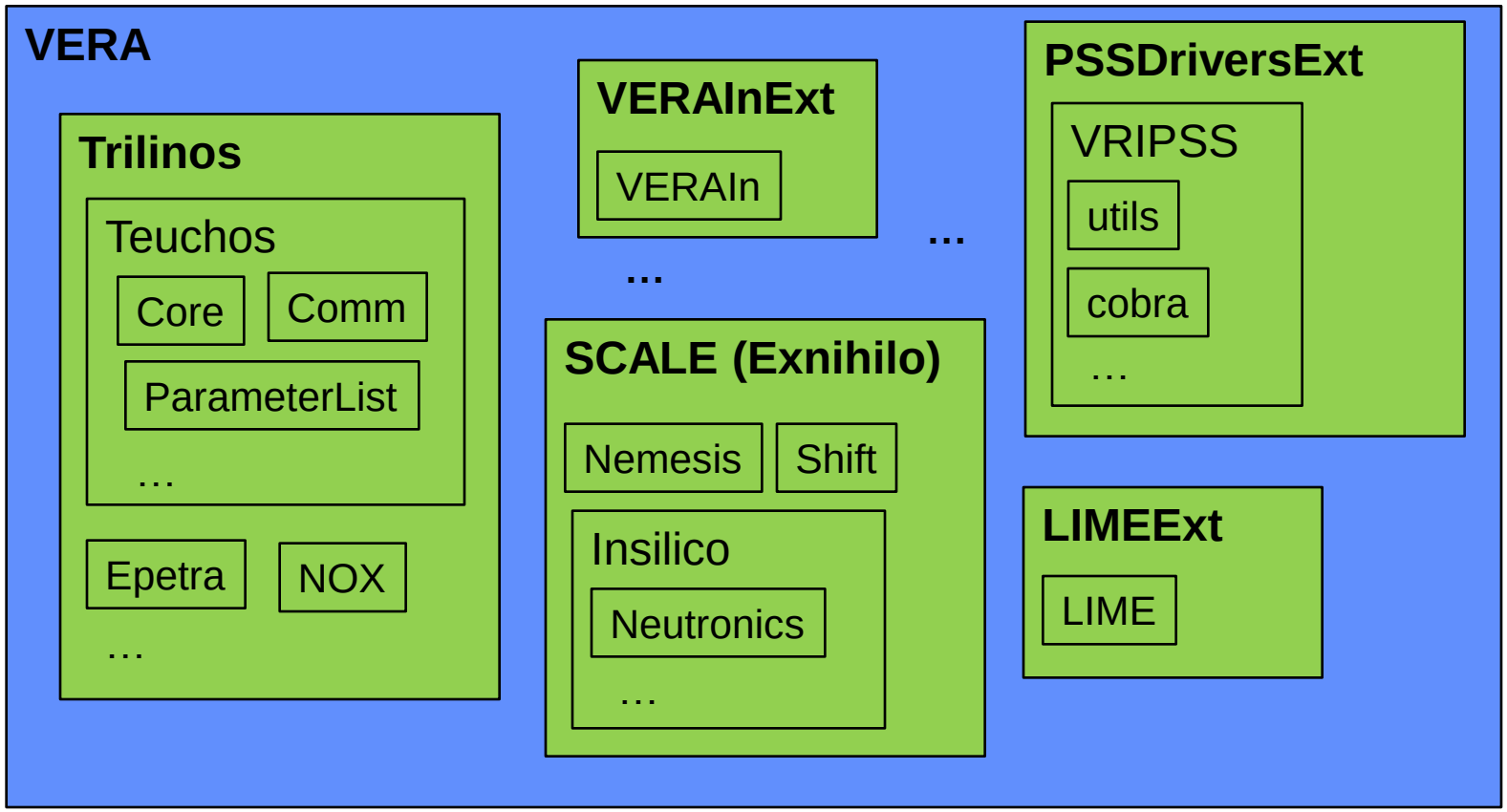
Overview of CASL

- **CASL: C**onsortium for the **A**dvanced **S**imulation of **L**ightwater reactors
- DOE Innovation Hub including DOE labs, universities, and industry partners
- Goals:
  - Advance modeling and simulation of lightwater nuclear reactors
  - Produce a set of simulation tools to model lightwater nuclear reactor cores to provide to the nuclear industry: **VERA: Virtual Environment for Reactor Applications**.
- Phase 1: July 2010 – July 2015
- Phase 2: Likely to be approved?
- Organization and management:
  - ORNL is the hub of the Hub
  - Milestone driven (6 month plan-of-records (PoRs))
  - Focus areas: **Physics Integration (PHI)**, Thermal Hydraulic Methods (THM), Radiation Transport Methods (RTM), Advanced Modeling Applications (AMA), Materials Performance and Optimization (MPO), Validation and Uncertainty Quantification (VUQ)

VERA Development Overview

- VERA Development is complicated in almost every way ☹
- VERA Currently Composed of:
  - 18 different git repositories on casl-dev.ornl.gov (clones of other repos) most with a different access list (NDAs, Export Control, etc.)
  - 14 different TriBITS repositories providing TriBITS packages
  - VERA: 144 SE Packages, 12 TPLs
- TriBITS (Tribal Build, Test, and Integrate System):
  - Based on CMake/CTest/CDash
  - Scalable package dependency system
- Software Development Process:
  - Official definition of VERA is 'master' branch of git repos under gitolite control at git@casl-dev.ornl.gov:<repo-name>.
  - Primary development platform: CASL Fissile/Spy Machines
  - VERA integration maintained by continuous and nightly testing:
    - Pre-push CI testing: checkin-test-vera.sh, cloned VERA git repos, on Fissile machine
    - Post-push CI testing: CTest/CDash, all VERA git repos, shared libs
    - Nightly CI testing: Debug and Release builds
    - 100% passing builds and tests!
  - VERA snapshots and releases are taken off of 'master' branches on casl-dev git repos.

VERA Meta-Project, Repositories, Packages & Subpackages



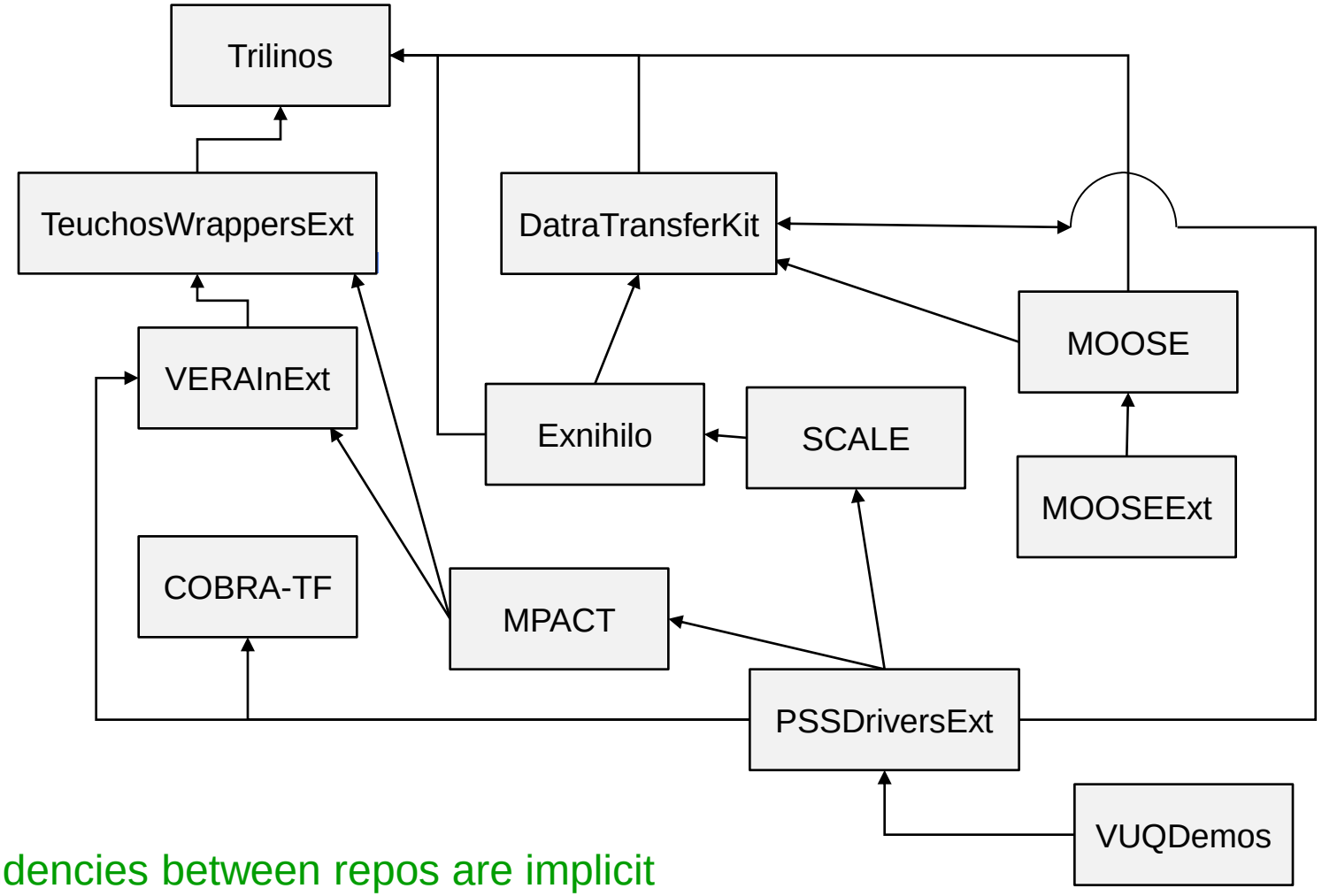
- **VERA**: Git repository and TriBITS meta-project (contains no packages)
- Git repos and TriBITS repos: **Trilinos, VERAInExt, LIMEExt, Exnihilo, …**
- TriBITS packages: **Teuchos, Epetra, VERAIn, Insilico, LIME, VRIPSS, …**
- TriBITS subpackages: **TeuchosCore, InsilicoNeutronics, …**
- TriBITS SE (Software Eng.) packages: **Teuchos, TeuchosCore, VERAIn, Insilico, InsilicNeutronics, …**

VERA/cmake/ExtraRepositoriesList.cmake

```
SET( VERA_EXTRAREPOS_DIR_REPOTYPE_REPOURL_PACKSTAT_CATEGORY
  TriBITS          ""   GIT  git@casl-dev:TriBITS          ""  Continuous
  Trilinos         ""   GIT  git@casl-dev:Trilinos         ""  Continuous
  Dakota  Trilinos/packages/TriKota/Dakota  GIT
     git@casl-dev:Dakota  NOPACKAGES  Continuous
  TeuchosWrappersExt "" GIT git@casl-dev:TeuchosWrappersExt ""  Continuous
  COBRA-TF         ""   GIT  git@casl-dev:COBRA-TF         ""  Continuous
  VERAInExt        ""   GIT  git@casl-dev:VERAInExt        ""  Continuous
  DataTransferKit  ""   GIT git@casl-dev:DataTransferKit  ""  Continuous
  MOOSEExt         ""   GIT  git@casl-dev:MOOSEExt         ""  Continuous
  MOOSE            MOOSEExt/MOOSE   GIT
      git@casl-dev:MOOSE   NOPACKAGES  Continuous
  SCALE            ""   GIT  git@casl-dev:SCALE           ""  Continuous
  Exnihilo         SCALE/Exnihilo   GIT
      git@casl-dev:Exnihilo                       NOPACKAGES  Continuous
  MPACT            ""   GIT  git@casl-dev:MPACT           ""  Continuous
  LIMEExt          ""   GIT  git@casl-dev:LIMEExt         ""  Continuous
  Mamba            ""   GIT  git@casl-dev:Mamba           ""  Continuous
  hydrath          ""   GIT  git@casl-dev:hydrath         ""  Nightly
  PSSDriversExt    ""   GIT  git@casl-dev:PSSDriversExt   ""  Continuous
  VUQDemos         ""   GIT  git@casl-dev:VUQDemos        ""  Nightly
)
```

- Official version of VERA in on master branch used for CI & Nightly testing
- Partial set of repos can be cloned (protected by different groups)
- Non-git repos are converted into git repos: Dakota, Scale, MOOSE

Dependencies Between Selected VERA Repositories



- Dependencies between repos are implicit
- Real dependencies are between packages in repos

CASL VERA Repository Management: Gitolite

- Gitolite Basics:
  - Special account “git” controls access to repos under /home/git/repositories
  - Users register public ssh keys: public SSH key => <userid>
  - Access to repos using git@casl-dev:<repo-name>
  - Flexible repo access rules based on gitolite groups
  - Repo git@casl-dev:gitolite-admin: SSH keys, group definitions and repo access rules:

```
gitolite-admin/
    keysdir/
    conf/gitolite.conf
```

- Advantages:
  - Provide repo access without providing accounts on the machine
  - Define access groups right in gitolite.conf
  - User can see repos and permissions using ssh git@casl-dev info
  - Flexible access control by repo, by directory, etc.
  - Supports custom git push hooks (e.g. use our existing git custom hooks)
  - Add new repos by adding to them to gitolite.conf and pushing

- Disadvantages:
  - Some initial setup

checkin-test-vera.sh

```
#!/bin/bash -e

…

$VERA_BASE_DIR/VERA/checkin-test.py \
--src-dir=$VERA_BASE_DIR_ABS/VERA \
--extra-repos-file=project \
--extra-repos-type=Continuous \
--ignore-missing-extra-repos \
--default-builds=MPI_DEBUG,SERIAL_RELEASE \
-j16 \
--ctest-timeout=400 \
$EXTRA_ARGS
```

- Very thin bash script wrapper for TriBITS checkin-test.py
- Automatically picks up cloned repos listed in ExtraRepositoriesList.cmake
- Safe pushes requires all affected repos to be cloned and available

Current Adoption and Usage of TriBITS in CASL

- VERA Repositories that are also independent projects using TriBITS:
  - Trilinos: SNL
  - SCALE: ORNL
    - Requires GCC 4.6.1+ and Intel 13.1+
    - Mixed Fortran, C, C++
    - Linux builds
    - Windows builds
  - Exnihilo: ORNL
    - Mostly C++ with some Fortran 90/77, Python, etc.
    - Contains Denovo, Shift, Insilico
  - MPACT: Univ. of Mich.
    - Requires GCC 4.6.1+ and Intel 13.1+
    - Mostly Fortran
    - Windows builds
  - COBRA-TF: Penn. State Univ.
    - Mostly Fortran 77 and 90
- Native TriBITS repos providing packages: TeuchosWrappersExt, VERAInExt, DataTransferKit, LIMEExt
- VERA Repositories/packages not using TriBITS as native build system but have secondary native TriBITS support: MAMBA, Hydra-TH
- VERA Repositories/packages not providing secondary TriBITS build: MOOSE
- Include external CMake project as a TriBITS package: DAKOTA

Multi-Repository Configuration and Building

TriBITS Structural Units

- TriBITS Project:
  - Complete CMake “Project”
  - Overall projects settings
- TriBITS Repository:
  - Collection of Packages and TPLs
  - Unit of distribution and integration
- TriBITS Package:
  - Collection of related software & Tests
  - Lists dependencies on SE Packages & TPLs
  - Unit of testing, namespacing, documentation, and reuse
- TriBITS Subpackage:
  - Partitioning of package software & tests
- TriBITS TPLs (Third Party Libraries):
  - Specification of external dependency (libs)
  - Required or optional dependency
  - Single definition across all packages

VC Repo not always one-to-one with TriBITS Repo!

Packages + Subpackages = Software Engineering (SE) Packages

Tacking on Extra Packages to a TriBITS Project

What if you want to tack on extra packages to an existing TriBITS project?

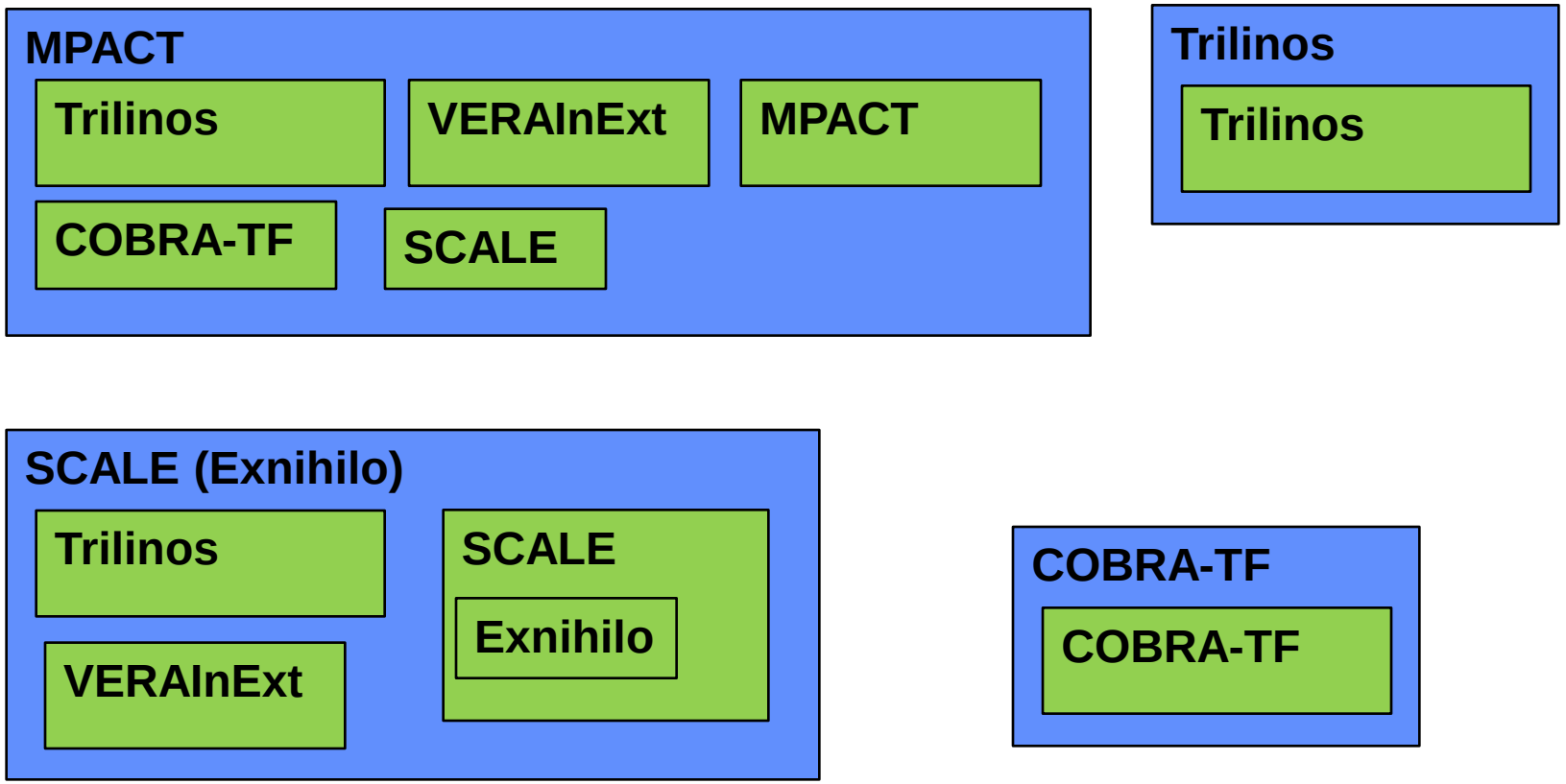
```
<Project>/
    CMakeLists.txt
    PackagesList.cmake
    …
    <repo0>/
        PackagesList.cmake
        …
    <repo1>/
        PackagesList.cmake
        …
```

```
$ ./do-configure -D<Project>_EXTRA_REPOSITORIES=<repo0>,<repo1> …
```

- Add-on packages from <repo0>, <repo1> automatically show up and can be enabled.
- **Every TriBITS project supports extra repositories!**

ToDo: Support pre-extra repositories (packages come before project native packages) … But will be more useful once merging of Packages and TPL support …

Flexibility in TriBITS Projects and Repositories

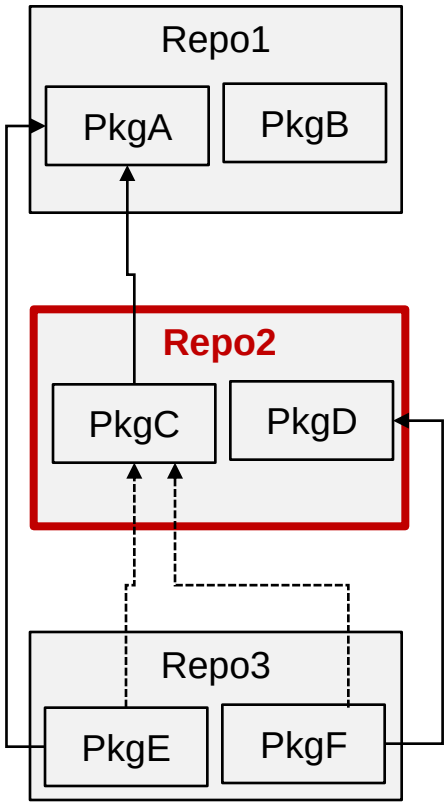


The same TriBITS repositories can be arranged into multiple TriBITS projects

Dealing with Missing Repos giving Missing Packages

What if **Repo2** is missing? Can we still configure and build the remaining packages in Repo3?

```
# Repo3/PkgE/cmake/Dependencies.cmake
   LIB_REQUIRED_PACKAGES PkgA
   LIB_OPTIONAL_PACKAGES PkgC
…
TRIBITS_ALLOW_MISSING_EXTERNAL_PACKAGES(PkgC)

# Repo3/PkgF/cmake/Dependencies.cmake
  LIB_REQUIRED_PACKAGES  PkgD
  LIB_OPTIONAL_PACKAGES  PkgC
…
TRIBITS_ALLOW_MISSING_EXTERNAL_PACKAGES(PkgC  PkgD)
```



Now when configuring the Project with Repo2 missing TriBITS automatically adjusts:

```
WARNING: PkgC is being ignored since its directory is missing and
PkgC_ALLOW_MISSING_EXTERNAL_PACKAGE = TRUE!
…
WARNING: Setting <PROJECT>_ENABLE_PkgF=OFF because PkgD is a
required missing package!
```

Inserting a package into Upstream Repo

What if you want to insert a package into the package's list of an upstream TriBITS Repo?

```
<projectDir>/
    PackagesList.cmake
    …
    DownstreamRepo/ # Not part of base repo!
       inserted_package/
```

<projectDir>/PackagesList.cmake:

```
TRIBITS_REPOSITORY_DEFINE_PACKAGES(
  …
  InsertedPackage   DownstreamRepo/inserted_package   ST
  …
  )
TRIBITS_ALLOW_MISSING_EXTERNAL_PACKAGES(InsertedPackage)
```

- InsertedPackage can depend on upstream packages and other packages listed in PackagesList.cmake.
- TriBITS automatically removes InsertedPackage if DownstreamRepo/inserted_package/ does not exit and disables all downstream dependencies.
- See ExternalPkg in TribitsExampleProject in TriBITS.

Primary Meta-Project Packages (PMPP)

- Some packages are “primary” to the project and are under development by the project. Other packages are just there to satisfy downstream dependencies.

Example: VERA

```
SET(Trilinos_NO_PRIMARY_META_PROJECT_PACKAGES TRUE)
SET(SCALE_NO_PRIMARY_META_PROJECT_PACKAGES TRUE)
SET(SCALE_NO_PRIMARY_META_PROJECT_PACKAGES_EXCEPT Insilico)
SET(LIMEExt_NO_PRIMARY_META_PROJECT_PACKAGES TRUE)
```

- -DVERA_ENABLE_ALL_PACKAGES=ON: Only explicitly enable the PMPPs for VERA and not packages in Trilinos, SCALE (except Insilico), LIME, etc.

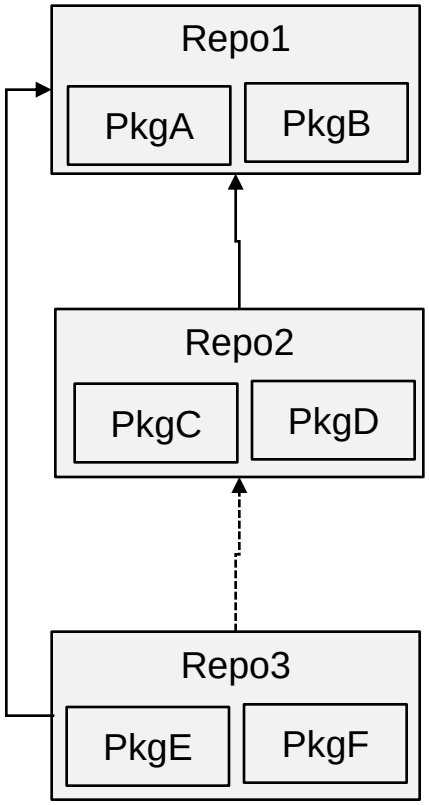
- -DVERA_ENABLE_ALL_TESTS=ON: Only enable tests for PMPPs that are explicitly enabled and not others that might be enabled in Trilinos, SCALE, etc.

- Used in:
  - Automated CTest/CDash testing (only process PMPPs)
  - Pre-push CI testing with checkin-test.py (tests for only PMPPs)
  - Creating source tarball distributions (excludes packages not enabled)

Incorporating Externally Configured/Built Software

- **Motivation**: For some software, it may not be practical or maintainable to create a (secondary) native TriBITS build for a piece of software.
- **Goal:**
  - Use another configure/build tool for external software.
  - Put in CMake/TriBITS hooks to incorporate into TriBITS build with other packages.
- **Easy case:** No upstream TriBITS packages => ***Repo1***
- **Medium case:** No downstream TriBITS packages => ***Repo3***
- **Hard case:** Both upstream and downstream TriBITS packages => ***Repo2***
- **Example:** INL MOOSE developers not willing to support a native TriBITS build of libmesh and MOOSE/Bison for CASL VERA. Libmesh depends on DataTransferKit and therefore Trilinos ☹

  Tpetra <= DataTransferKit <= libmesh <= MOOSE <= VRIPSS
- **Technical challenges in TriBITS:**
  - Generate export makefile for upstream Trilinos packages
  - Create CMake rules to produce libs/executables
  - Add dependencies for changes to upstream code
  - Add dependencies for modified external project files



Incorporating Externally Configured/Built Software: Example

```
# TribitsExampleProject/packages/wrap_external/CMakeLists.txt

TRIBITS_PACKAGE(WrapExternal)

# Blow away build if out of date
TRIBITS_DETERMINE_IF_CURRENT_PACKAGE_NEEDS_REBUILT(
  ${SHOW_MOST_RECENT_FILES_ARG}
  SHOW_OVERALL_MOST_RECENT_FILES
  CURRENT_PACKAGE_OUT_OF_DATE_OUT ${PACKAGE_NAME}_BULID_IS_OUT_OF_DATE
  )
IF (${PACKAGE_NAME}_BULID_IS_OUT_OF_DATE))
  EXECUTE_PROCESS(COMMAND rm -rf "${PACKAGE_BINARY_DIR}/external_func/"  )
ENDIF()

# Write the export makefile that will be used by the external project
TRIBITS_WRITE_FLEXIBLE_PACKAGE_CLIENT_EXPORT_FILES(
  PACKAGE_NAME ${PACKAGE_NAME}
  EXPORT_FILE_VAR_PREFIX TribitsExProj
  WRITE_EXPORT_MAKLEFILE "${EXPORT_MAKKEFILE}" )

# Run external configure
EXECUTE_PROCESS(
  COMMAND ${PYTHON_EXECUTABLE} ${EXTERNAL_FUNC_SOURCE_DIR}/configure.py
    --with-export-makefile=${EXPORT_MAKKEFILE}
    --src-dir=${EXTERNAL_FUNC_SOURCE_DIR}
    --build-dir=${EXTERNAL_FUNC_BINARY_DIR} )

# Define a custom build rule and target to create exteranl_func library
ADD_CUSTOM_COMMAND(
  OUTPUT ${EXTERNAL_FUNC_LIB_FILE}
  DEPENDS ${EXTERNAL_FUNC_SOURCE_DIR}/external_func.hpp
    ${EXTERNAL_FUNC_SOURCE_DIR}/external_func.cpp
  COMMAND make
  WORKING_DIRECTORY ${EXTERNAL_FUNC_BINARY_DIR} )
ADD_CUSTOM_TARGET( build_external_func
  DEPENDS ${EXTERNAL_FUNC_LIB_FILE} )
```

Automatically figure out if reconfigure and rebuild is needed!

Create export file for usage by external software package configure.

Run the external software configure

Create custom command s and targets for building the external software.

Incorporating Externally Configured/Built Software: Example

```
# TribitsExampleProject/packages/wrap_external/CMakeLists.txt

# continued …

# D) Add the imported library with TRIBITS_ADD_LIBRARY( ... IMPORTED ...)
#
# Below, I just manually do what TRIBITS_ADD_LIBRARY() would do automatically.

# D.1) Create an imported library target and set up the depenancies
ADD_LIBRARY(exteranl_func STATIC IMPORTED GLBOAL) # GLOBAL
SET_PROPERTY(TARGET exteranl_func PROPERTY IMPORTED_LOCATION ${EXTERNAL_FUNC_LIB_FILE})
ADD_DEPENDENCIES(build_external_func pws_c)                # Upstream TriBITS libs pws_s
ADD_DEPENDENCIES(exteranl_func build_external_func)
GLOBAL_SET(exteranl_func_IMPORTLIB_TARGET build_external_func)

# D.2) Update the TriBITS varaibles
APPEND_SET(${PACKAGE_NAME}_LIB_TARGETS exteranl_func)
GLOBAL_SET(${PACKAGE_NAME}_LIBRARIES exteranl_func pws_c)     # Upstream TriBITS libs pws_s
GLOBAL_SET(${PACKAGE_NAME}_INCLUDE_DIRS ${EXTERNAL_FUNC_SOURCE_DIR})
GLOBAL_SET(${PACKAGE_NAME}_HAS_NATIVE_LIBRARIES ON)
INCLUDE_DIRECTORIES(${EXTERNAL_FUNC_SOURCE_DIR})

# E) Add an executable and test to show that it works!
TRIBITS_ADD_EXECUTABLE_AND_TEST(run_external_func
  SOURCES run_external_func.cpp
  DEPLIBS exteranl_func
  PASS_REGULAR_EXPRESSION "external_func C B A"
  )

TRIBITS_PACKAGE_POSTPROCESS()
```

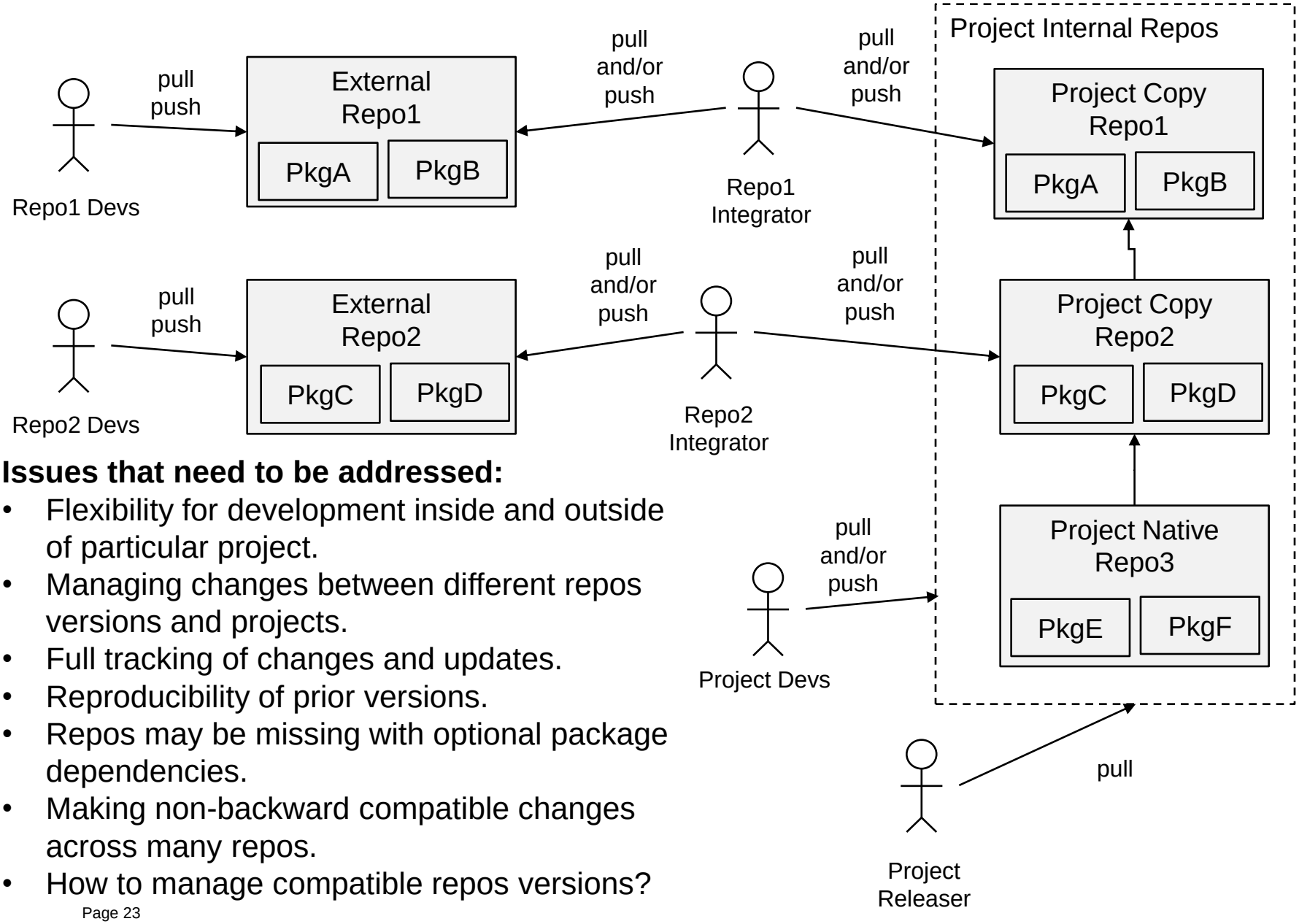
What is missing?

- Nice wrappers in TriBITS to support this better
- Resolve issues with generation of export files.

TriBITS wrapper for MOOSE is MUCH worse!

Multi-Repository Version Control and Repository Management

Managing Compatible Repos and Repo Versions



Issues that need to be addressed:

- Flexibility for development inside and outside of particular project.
- Managing changes between different repos versions and projects.
- Full tracking of changes and updates.
- Reproducibility of prior versions.
- Repos may be missing with optional package dependencies.
- Making non-backward compatible changes across many repos.
- How to manage compatible repos versions?

Managing Multiple Compatible Repo Versions with git?

- Snapshot all repos into one big repo (e.g. SIERRA/Trilinos style):
  - Advantages:
    - One set of SHA1s, easy to do git bisect
    - One repo to pull and build final version
  - Disadvantages:
    - Harder to coordinate changes back and forth to native repos
    - Does not allow for partitioning based on access control
- Use git modules:
  - Advantages:
    - Built-in git support and documentation
    - Individual repos stay independent (let git do its job).
  - Disadvantages:
    - Extra commands to pull component repos
    - Updating repos versions is complex for non-git savvy developers
    - Does not support co-development well at all
- Clone repos under master repo (**gitdist**) and track sets of compatible repos as files and provide tools for accessing specific versions (**<Project>RepoVersion.txt** files)

CASL VERA => Uses separate repos for native git repos, uses snapshotting for non-native git repos.

gitdist : git for collection of git repos

- **gitdist**: Simple stand-alone Python tool for distributing git commands across multiple git repos. Contained in TriBITS/common_tools/git/gitdist.

```
Usage: gitdist [gitdist options] <raw-git-command> [git options]
```

- .gitdist file in base git repo:

```
TriBITS
Trilinos
Trilinos/packages/TriKota/Dakota
…
```

Example:

```
$ gitdist status
*** Base Git Repo: VERA
(On branch master)
*** Git Repo: Trilinos
(On branch master)
*** Git Repo: Trilinos/packages/TriKota/Dakota
...
```

- Common bulk commands: pull, push, local-stat, log -1
- Works well for < 10-20 repos, not for 100s of repos!

<Project>RepoVersion.txt

- How to keep track of compatible sets of repos?
  => TriBITS support for <Project>RepoVersion.txt file:

```
*** Base Git Repo: SomeBaseRepo
e102e27 [Mon Sep 23 11:34:59 2013 -0400] <author1@someurl.com>
First summary message
*** Git Repo: ExtraRepo1
b894b9c [Fri Aug 30 09:55:07 2013 -0400] <author2@someurl.com>
Second summary message
*** Git Repo: ExtraRepo2
97cf1ac [Thu Dec 1 23:34:06 2011 -0500] <author3@someurl.com>
Third summary message
...
```

- By setting -D<PROJECT>_GENERATE_REPO_VERSION_FILE=ON, the file <Project>RepoVersion.txt gets:
  - generated in the build base directory,
  - echoed in the configure output (therefore archived to CDash),
  - installed in the base install directory,
  - included in the source tarball ('make package_source'),
  - installed in the base install directory from the untarred source.

```
gitdist --dist-repo-file=<Project>RepoVersion.<somedate>.txt [other options]
```

Using <Project>RepoVersion.txt for Snapshot Distributions

- Known “good” versions of the Project code are recorded as <Project>RepoVersion.txt files (e.g. archived on CDash).

- Example: If a given Nightly build of Project passed on all platforms then we can give the associated <Project>RepoVersion.txt file as the “version” for a client to use.

- Send client file <Project>RepoVersion.<newdate>.txt for “good” version to install.

- Client gets updated version:

```
$ cd <SOME-BASE-DIR>/<PROJECT>
$ gitdist fetch
$ gitdist --dist-version-file=~/<PROJECT>RepoVersion.<newdate>.txt \
  checkout _VERSION_
```

- Client can see changes since a previous installs of <Project>:

```
$ gitdist fetch
$ gitdist \
  --dist-version-file=~/<PROJECT>RepoVersion.<newdate>.txt \
  --dist-version-file2=${INSTALL_BASE}/<olddate>/<Project>RepoVersion.txt \
  log-short --name-status _VERSION_ ^_VERSION2_
```

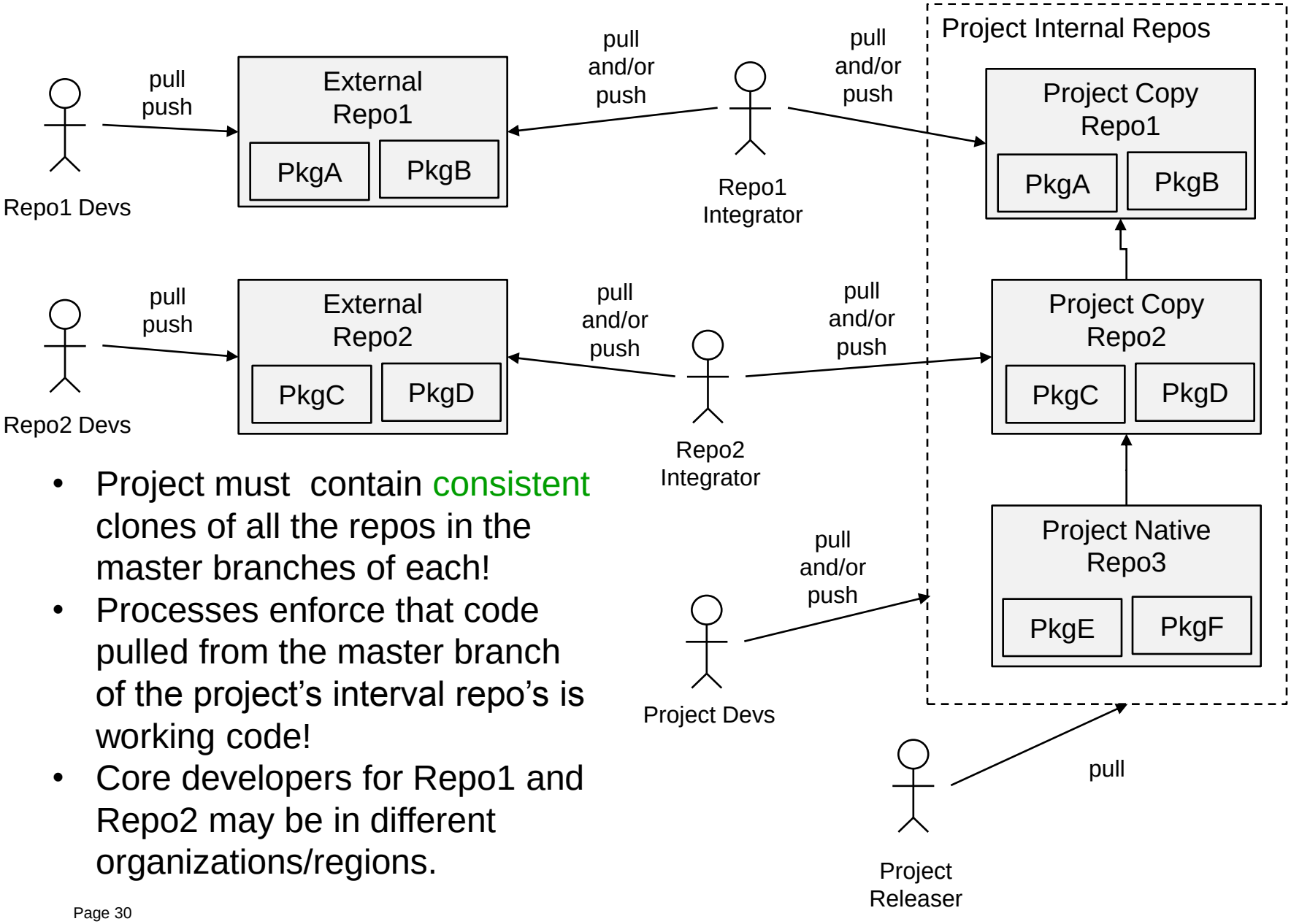
Pre-Push CI Testing and Pushing of Multiple Repos

```
#!/bin/bash -e

…

$VERA_BASE_DIR/VERA/checkin-test.py \
--src-dir=$VERA_BASE_DIR_ABS/VERA \
--extra-repos-file=project \
--extra-repos-type=Continuous \
--ignore-missing-extra-repos \
--default-builds=MPI_DEBUG,SERIAL_RELEASE \
-j16 \
--ctest-timeout=400 \
$EXTRA_ARGS
```

- Very thin bash script wrapper for TriBITS checkin-test.py
- Automatically picks up cloned repos listed in ExtraRepositoriesList.cmake
- Safe pushes requires all affected repos to be cloned and available

Multi-Repository Integration Models and Processes

Integrating Repos into Project: External and Internal



- Project must contain consistent clones of all the repos in the master branches of each!
- Processes enforce that code pulled from the master branch of the project's interval repo's is working code!
- Core developers for Repo1 and Repo2 may be in different organizations/regions.

Multi-Repository Integration Models

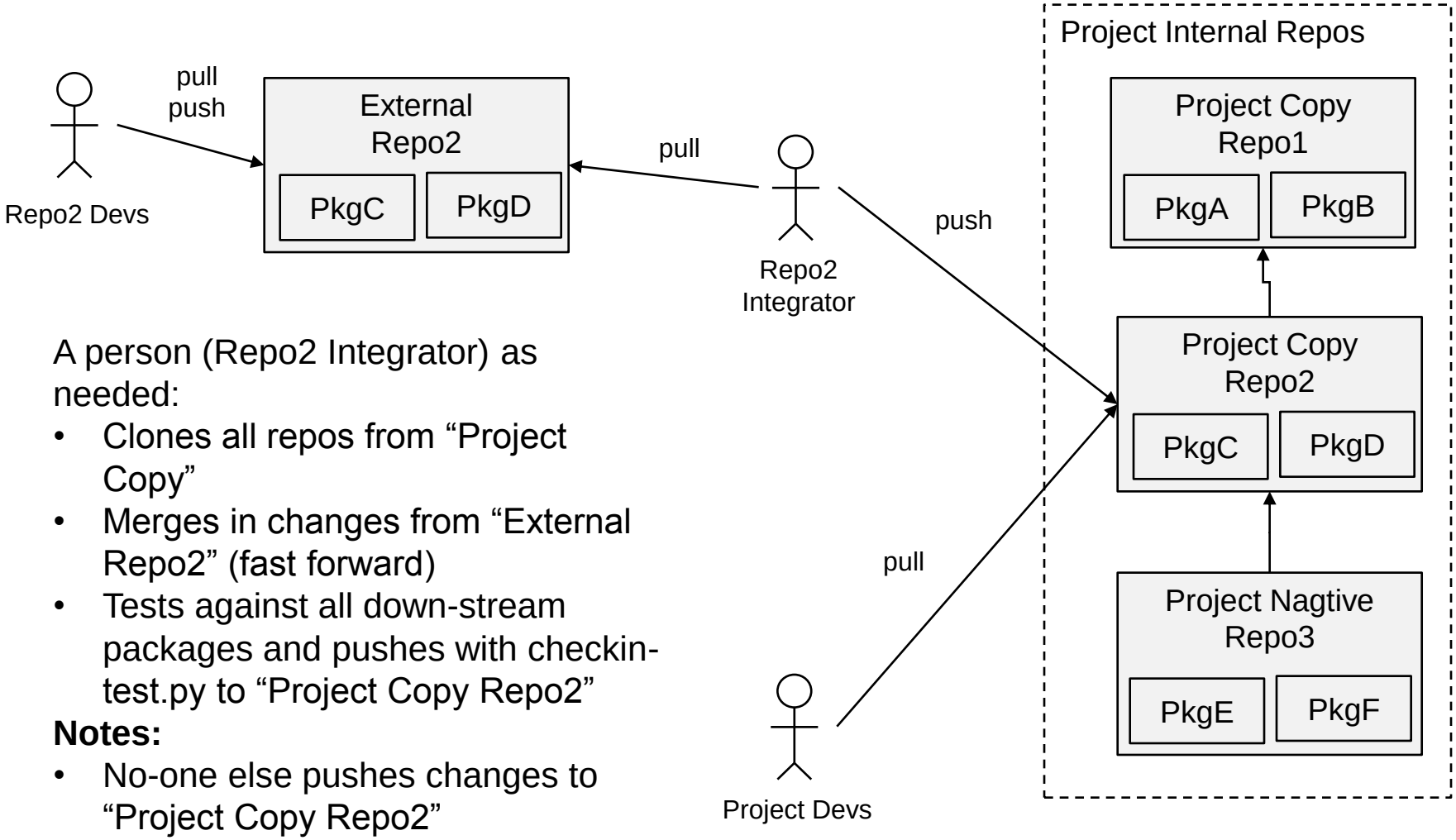
- Range of development and sync models (external dev to internal dev)
  - External repo is manually synced into project/master as needed.
  - External repo is synced automatically using sync server into project/master using the checkin-test.py script.
  - Both external and internal repos pushed to by different development groups with sync servers running one way or both ways.
  - Internally managed repo is synced to an external repo on some schedule to make available to other developers and users and changes from external repo may or may not be synced back into internal repo.
  - Internally managed repo

- A given repo may shift between different integration models at different periods of time (e.g. Trilinos, COBRA-TF)

- Integration of different repos should be done independently if possible (e.g. errors in MPACT should not stop pushes of SCALE/Exnihilo and visa versa).

- Non-backward compatible changes to upstream repos require coordinated development and combined pushing to project/master

External Repo is manually synced



A person (Repo2 Integrator) as needed:

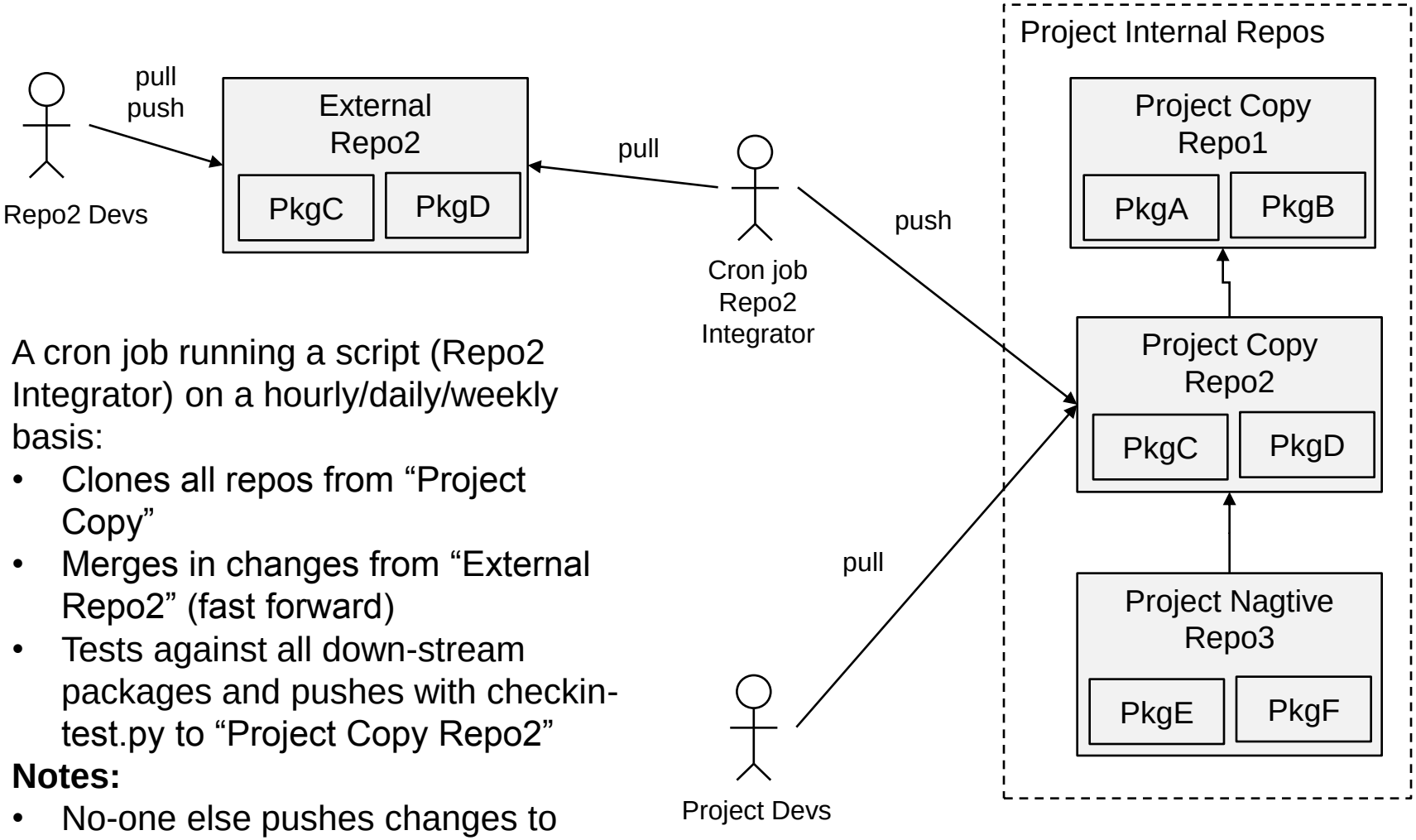
- Clones all repos from “Project Copy”
- Merges in changes from “External Repo2” (fast forward)
- Tests against all down-stream packages and pushes with checkin-test.py to “Project Copy Repo2”

Notes:

- No-one else pushes changes to “Project Copy Repo2”
- Good when changes are **not** urgent for Project or when “External Repo2” is unstable

VERA Examples: DataTransferKit, MOOSE

External repo is synced automatically using sync server



A cron job running a script (Repo2 Integrator) on a hourly/daily/weekly basis:

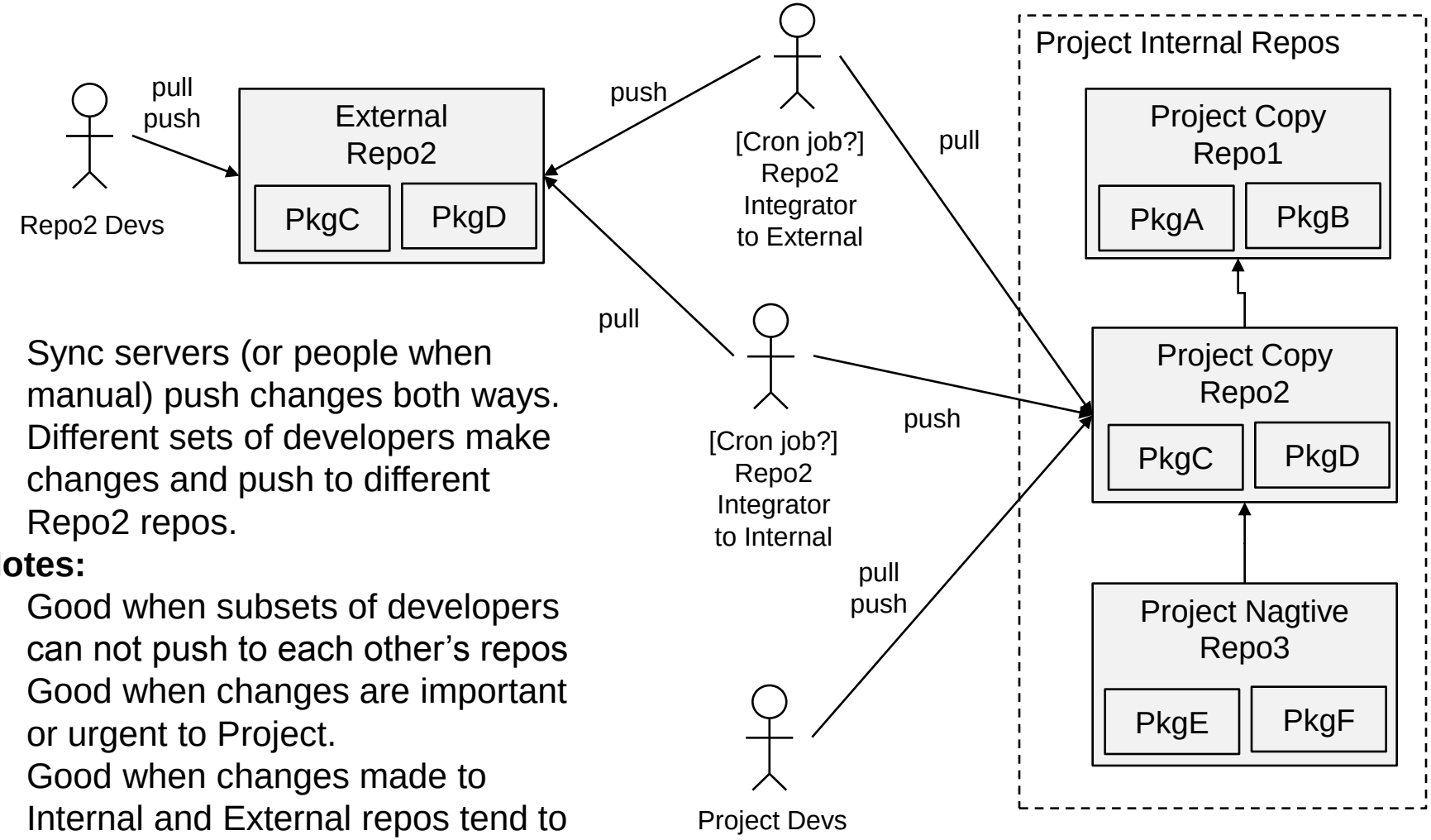
- Clones all repos from “Project Copy”
- Merges in changes from “External Repo2” (fast forward)
- Tests against all down-stream packages and pushes with checkin-test.py to “Project Copy Repo2”

Notes:

- No-one else pushes changes to “Project Copy Repo2”.
- Good when changes are important or urgent to Project and “External Repo2” is fairly stable.

VERA Examples: SCALE/Exnihilo, MPACT, Hydra-TH

Both external and internal repos pushed to



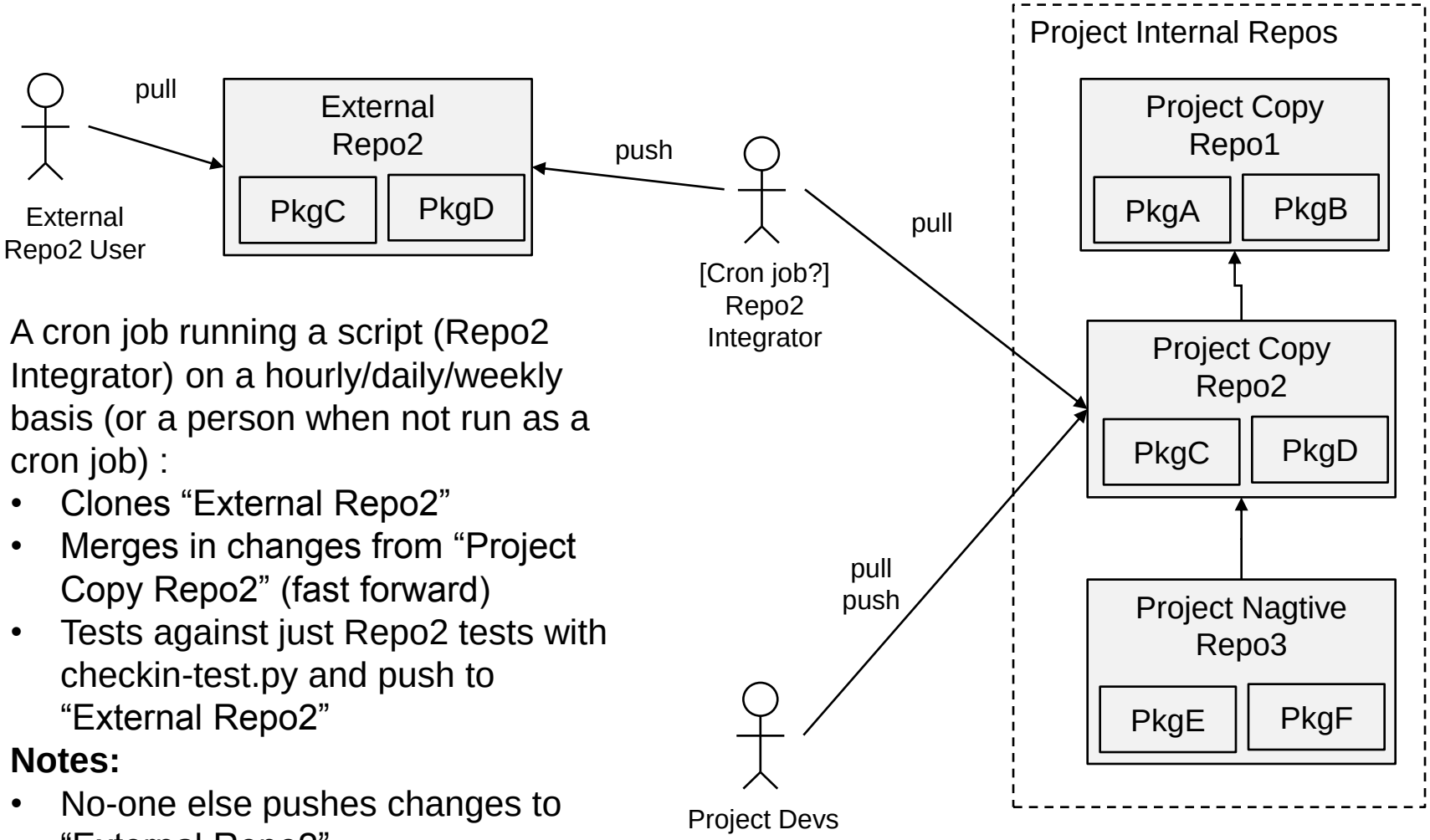
- Sync servers (or people when manual) push changes both ways.
- Different sets of developers make changes and push to different Repo2 repos.

Notes:

- Good when subsets of developers can not push to each other's repos
- Good when changes are important or urgent to Project.
- Good when changes made to Internal and External repos tend to be independent.
- Most complex and danger of merge conflicts that someone has to resolve!

VERA Examples: TriBITS, Trilinos, COBRA-TF (future)

Internally managed repo is synced to an external repo



A cron job running a script (Repo2 Integrator) on a hourly/daily/weekly basis (or a person when not run as a cron job) :

- Clones “External Repo2”
- Merges in changes from “Project Copy Repo2” (fast forward)
- Tests against just Repo2 tests with checkin-test.py and push to “External Repo2”

Notes:

- No-one else pushes changes to “External Repo2”
- Good when you just want to make changes available to external users on a continuous basis.

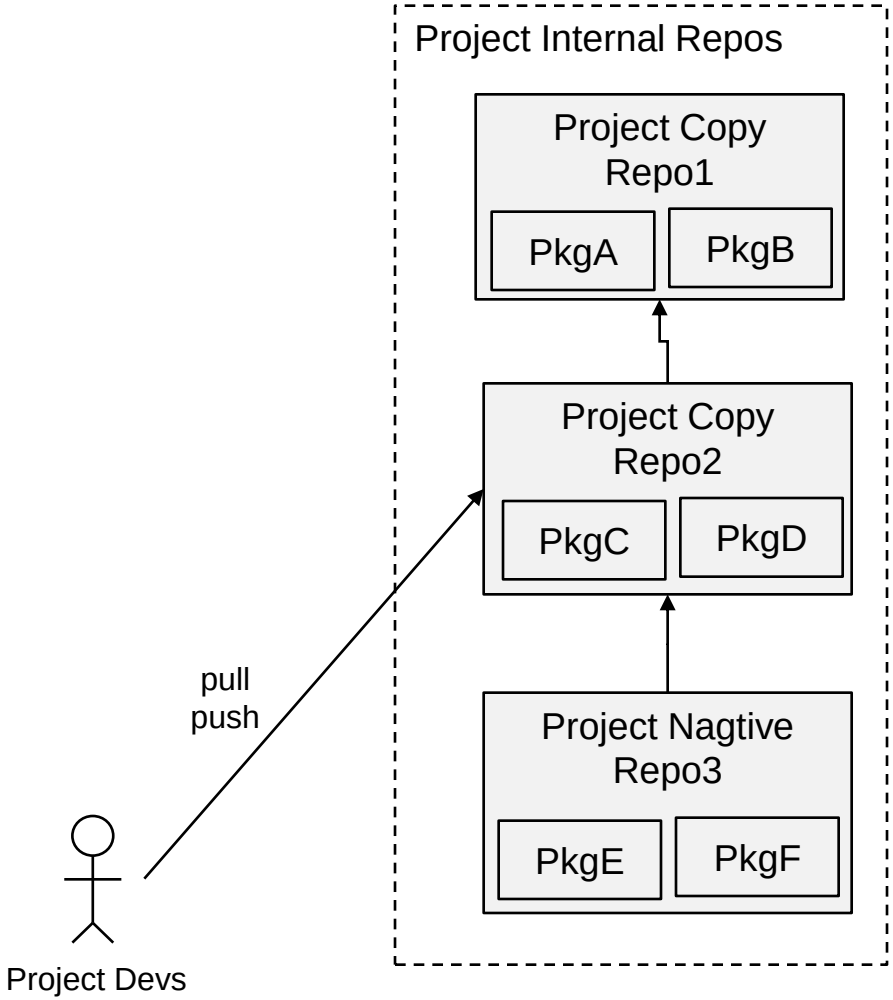
VERA Examples: COBRA-TF (current), TeuchosWrapersExt

Internally managed repo

- Simple single-repo development model

Notes:

- Good when you don't need to coordinate with developers outside of Project



VERA Examples: VERAInExt, PSSDriversExt

Recent and Future TriBITS Development

Trilinos Software Engineering Technologies and Integration

Progress in Last Year:

- TriBITS Hosted on Github
  - URL: https://github.com/TriBITSPub/TriBITS
  - Github issues, pull requests, etc.
  - Snapshotted into Trilinos (keep integrated).
    - http://trac.trilinos.org/wiki/TriBITSTrilinosDev
- TriBITS Documentation: Developers guide 170+ pages, build/test reference 30+ pages, overview document in progress …

Plans for Next Year:

- TriBITS System (IDEAS Project)
  - Partition TriBITS into lighter-weight framework(s), better support wrapping external software as a TriBITS package, etc. => Broader incremental adoption.
  - Merge TriBITS concepts of Packages and TPLs => Construct larger meta-projects, build/install/test meta-projects in pieces, extract and build/install individual packages, (optionally) build Trilinos with TPLs, use export XXXConfig.cmake files as glue.
  - Standard installations of TriBITS => Build individual TriBITS packages stand-alone.
  - Overview and tutorials
  - Implementation of TriBITS Lifecycle Model in TriBITS system => Targeted metrics and testing of backward compatibility, valgrind, coverage, etc.

Summary

- CASL VERA development has pushed multi-repo support in TriBITS
- TriBITS offers a lot of flexibility in assembling TriBITS Repositories and Packages into different TriBITS (complete CMake) projects.
- Major remaining issues yet to be resolved in TriBITS:
  - Combining concepts of packages and TPLs for large meta-projects
  - Finish support for wrapping externally configured/build software as TriBITS packagse.
  - High-level and tutorial documentation
- But once these are done => TriBITS will be a good candidate for a universal meta-build and installation system for a large amount of CSE software.

THE END

Miscellaneous Issues and Topics for TriBITS and VERA

Git Snapshot Repos and Local Project Changes

Basic Concepts:

- The external native repos does not use git (e.g. SCALE uses Mercurial)
- A git copy of the native repo is created and “snapshot” commits are created.
- Each git snapshot commit must contain info about the version of the repo from the native repo to track versions
- Local changes can be made on a local git branch and merged with snapshot branch.

Example: SCALE

```
$ git log

commit 8423c41af901082a0b05a31cbf2b15363fc09773 (master~1)
Author: Kevin Clarno <clarnokt@ornl.gov>
Date:   Wed Oct 30 10:29:46 2013 -0400

    changeset:   9909:ba36380b3a92
    tag:         tip
    user:        Ugur Mertyurek <u2m@ornl.gov>
    date:        Wed Oct 30 10:13:11 2013 -0400
    files:       src/bonami/CMakeLists.txt src/bonamiM/BonamiData.cpp
    description:
    case:3312 Fixed DBC syntax error on bonamiData commented header
information in bonami cmakelist to preven possible install error
```

Git Snapshot Repos and Local Changes : MOOSE and SVN

Branches in casl-dev/MOOSE.git repo:

- inl_clean_svn: Direct snaphot commits for MOOSE SVN repo
- master: Local changes and merges from inl_clean_svn

Updating snapshot:

```
$ cd MOOSE
$ git fetch origin
$ git checkout -b inl_clean_svn origin/inl_clean_svn  # tracking branch
$ ./create_snapshot_commit    # update from current SVN repo
$ git push                    # push to origin/inl_clean_svn
$ git checkout master
$ git pull
$ git merge inl_clean_svn     # hope for no merge conflicts!
$ git push                    # push to origin/master (TEST FIRST!)
```

Example:

```
commit bacba1ed219d9fba4a50132a3173efcf3f469d18 (master~2^2~1^2)
Author: moosetest <moosetest@dd8cd9ef-2931-0410-98ca-75ad22d19dd1>
Date:   Tue May 28 15:03:57 2013 +0000

r18996 | permcj | 2013-05-28 08:46:46 -0600 (Tue, 28 May 2013) | 1 line
    Holy Warnings Batman! refs-#1777
```

Changing Integration Models for TriBITS and Trilinos

- Integration Models with TriBITS and Trilinos have changed in VERA

- Integration phases for Trilinos in VERA phases:
  1. Push all changes to Trilinos (including TriBITS) directly to ssg/master:
     1. Run VERA CI and Nightly testing directly against Trilinos on ssg/master
     2. Run a sync server to test Trilinos against Denovo and push to casl-dev/master if all tests pass, run on loops of 10 minutes between iter.
     3. " … ", adding more packages, run on loops of 3 hours
  2. Push changes to TriBITS under Trilinos directly to casl-dev/master
     1. Turn off sync server from from ssg/master to casl-dev/master. Manually push changes back and forth as needed.
     2. Run sync server to push TriBITS changes from casl-dev/master to ssg/master on a daily basis.
     3. Run sync server to push general Trilinos changes form ssg/master to ssg/master on a weekly basis (but tested every day flagging regressions).
  3. Pull TriBITS out of Trilinos into own repo and manage independently.
     1. Push changes for TriBITS directly to casl-dev TriBITS repo
     2. Push small changes to Trilinos for CASL directly to casl-dev TriBITS repo
     3. Manually push changes for TriBITS from casl-dev repo to github repo
     4. Manually snapshot TriBITS updates from github into Trilinos (see http://trac.trilinos.org/wiki/TriBITSTrilinosDev)
     5. Manually evaluate Trilinos on ssg and test against VERA before pushing to casl-dev (see http://trac.trilinos.org/wiki/VERAIntegrationTriBITSTrilinos).

VERA Development Environment Installation

- VERA Test Stands and RSICC release require a specific build environment:

```
common_tools/
  autoconf-2.69/
  cmake-2.8.5/
  gitdist
gcc-4.6.1/
  toolset/
    gcc-4.6.1/
    openmpi-1.4.3/
  tpls/
    opt/
      common/
        lapack-3.3.1/
        boost-1.49.0/
        zlib-1.2.5-patched/
        moab-4.5.0/
        hypre-2.8.0b/
        petsc-3.3-p4/
      vera_cs/
        hdf5-1.8.7/
        silo-4.8/
        qt-4.8.2/
      ...
```

VERAInstallationGuide.[rst,html,pdf] describe:

- Install scripts in tribits/python download tarballs for common_tools, GCC, and OpenMPI and build/install from source.
- Scripts and source in casl_tpls.svn repo configure/build/install all of the TPLs.
- Standard configurations of VERA set up to automatically work with the Standard VERA Dev Env.