SAND2020-13470C

Sandia National Laboratories

CASL VERA Infrastructure - TriBITS

PRESENTED BY

Roscoe A. Bartlett, Ph.D.
Dept. Software Engineering & Research
Center for Computing Research
https://bartlettroscoe.github.io

SAND2018-11082 PE

Background and Motivation

The Challenge => Develop and Deploy Complex Software

- Multiple software repositories and distributed development teams
- Multiple compiled programming languages (C, C++, Fortran) and mixed-language programs
- Multiple development and deployment platforms (Linux, Windows, Super-Computers, etc.)
- Stringent software quality requirements

Solution Approach

=> TriBITS custom CMake build & test framework

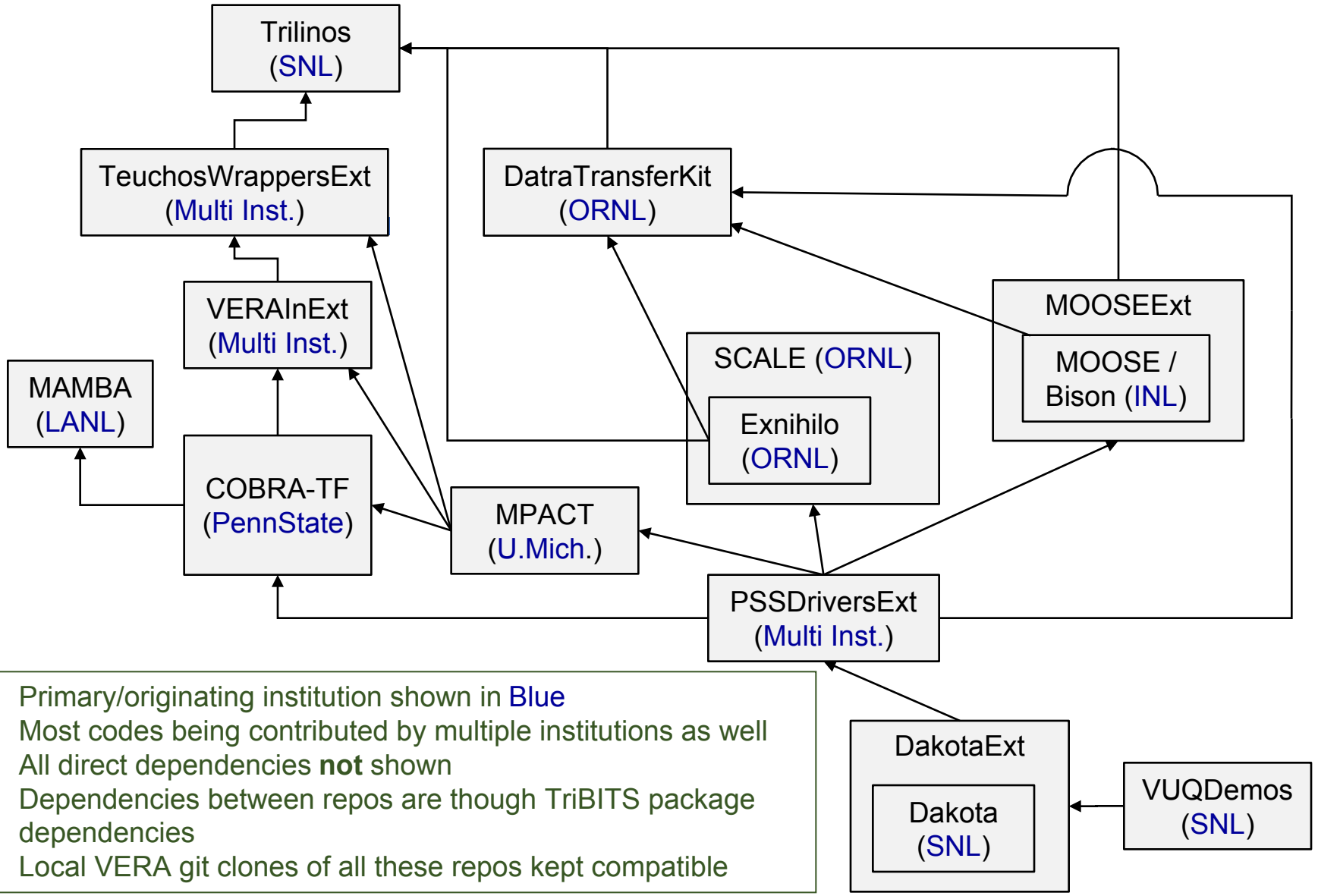
Overview of CASL

- **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 – June 2015
- Phase 2: July 2015 – June 2020
- 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)

CASL VERA Development Overview (2015)

- VERA Development is complicated in almost every way ☹
  - VERA Currently Composed of:
    - 18 different git repositories most with a different access list (NDAs, Export Control, IP, etc.)
  - CMake build system using TriBITS Framework:
    - Over 2700 CMakeLists.txt files!
  - VERA Software Development Process:
    - Official definition of VERA is 'master' branch of git repo mirrors at ORNL.
    - Primary development platform: ORNL CASL Machines (and clones)
    - 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 ORNL git repos.

Dependencies Between Selected VERA Repos (2015)



- Primary/originating institution shown in Blue
- Most codes being contributed by multiple institutions as well
- All direct dependencies **not** shown
- Dependencies between repos are though TriBITS package dependencies
- Local VERA git clones of all these repos kept compatible

Why CMake?

Why TriBITS?

Why CMake?

Open-source tools maintained and used by a large community and supported by a profession software development company (Kitware).

CMake:

- Simplified build system, easier maintenance
- Improved mechanism for extending capabilities (CMake language)
- Support for all major C, C++, and Fortran compilers.
- Automatic full dependency tracking (headers, src, mod, obj, libs, exec)
- Good Fortran support (parallel builds with modules with src => mod => object tracking, C/Fortran interoperability, etc.)
- Shared libraries on all platforms and compilers (support for RPATH)
- Faster configure times (e.g. > 10x faster than autotools)
- Native support for MS Windows (e.g. Visual Studio projects)
- Portable support for cross-compiling

CTest:

- Parallel running and scheduling of tests and test time-outs
- Memory testing (Valgrind)
- Line coverage testing (GCC LCOV)
- Better integration between the test system and the build system

Componentized CMake-based Projects Approaches

CMake, CTest, and CDash are great, but raw usage does not scale very well to large projects and multiple repositories and teams!

- **Multiple CMake projects:**
  - Manual builds and linking through <Package>Config.cmake files (e.g. Albany & Trilinos)
  - CMake ExternalProject: (Also see LLNL Spack)
    + Provided as standard CMake module ExternalProject.cmake
    + Allows non-CMake subprojects
    - Does not do full dependency tracking between component CMake projects
  - Google Catkin (used for the Google Robotics Operating System (ROS) project)
    + Automatic dependency handling logic (implemented in Python)
    + Parallel builds of CMake (sub)projects and posting to CDash
    - Requires Python for basic usage
  - CMakeBasis (Also see LLNL BLT)
    + Standardize CMakeList.txt and creation/usage of <Package>Config.cmake files
    - Core functionality requires Python
- **Single CMake project:**
  - Kitware VTK Modules:
    - Does not support optional dependencies
    - Slow due to need to sort submodules based on dependencies
  - TriBITS:
    + Support multiple repos
    + Core functionality depends only on CMake 3.10+

Why TriBITS?

- Framework for large, distributed multi-repository CMake projects
- Reduce boiler-plate CMake code and enforce consistency across large distributed projects
- Subproject dependencies and namespacing architecture (packages)
- Automatic package dependency handling (directed acyclic graph)
- Additional functionality missing in raw CMake
- Change default CMake behavior when necessary
- Additional tools for agile software development processes (e.g. Continuous Integration (CI))

History of TriBITS:

- 2007: Initially developed as a CMake package architecture for Trilinos
- 2011: Generalized and extended for CASL VERA
- 2014: Source code hosted on GitHub

Raw CMake vs. TriBITS

Example Raw CMakeLists.txt File

```
# Build and install library
set(HEADERS hello_world_lib.hpp)
set(SOURCES hello_world_lib.cpp)
add_library(hello_world_lib ${SOURCES})
install(TARGETS hello_world_lib DESTINATION lib)
install(FILES ${HEADERS} DESTINATION include)

# Build and install user executable
add_executable(hello_world hello_world_main.cpp)
target_link_libraries(hello_world hello_world_lib)
install(TARGETS hello_world DESTINATION bin)

# Test the executable
add_test(hello_world ${CMAKE_CURRENT_BINARY_DIR}/hello_world)
set_tests_properties(hello_world PROPERTIES PASS_REGULAR_EXPRESSION "Hello World")

# Build and run some unit tests
add_executable(unit_tests hello_world_unit_tests.cpp)
target_link_libraries(unit_tests hello_world_lib)
add_test(unit_test ${CMAKE_CURRENT_BINARY_DIR}/unit_tests)
set_tests_properties(unit_test  PROPERTIES PASS_REGULAR_EXPRESSION "All unit tests passed")
```

Executable and test names must be globally unique!

Example TriBITS Package CMakeList.txt File

```
tribits_package(HelloWorld)
tribits_add_library(hello_world_lib  HEADERS  hello_world_lib.hpp SOURCES hello_world_lib.cpp)
tribits_add_executable(hello_world  NOEXEPREFIX  SOURCES  hello_world_main.cpp  INSTALLABLE)
tribits_add_test(hello_world  NOEXEPREFIX  PASS_REGULAR_EXPRESSION  "Hello World")
tribits_add_executable_and_test(unit_tests  SOURCES  hello_world_unit_tests.cpp
   PASS_REGULAR_EXPRESSION  "All unit tests passed")
tribits_package_postprocess()
```

- Less duplication and boiler-plate code
- Fewer commands
- Build command wrappers:
  - Install by default (most common)
  - Optionally Install libraries and headers or just executables?
  - Optional global prefixing of libraries
  - And more …
- CTest command wrappers:
  - Automatic namespacing of tests and test executables
  - Classification of tests (BASIC, CONTINUOUS, NIGHTLY, …)
  - Uniform handling of timeouts (and scaling of timeouts)
  - And more …

Maintain consistency and add/change behavior across different independent repositories and packages and 1000s of CMakeLists.txt files!.

TriBITS Structural Units and Meta-Projects

TriBITS Hierarchical Structural Units

TriBITS Project:
- Complete CMake “Project”
- Overall projects settings

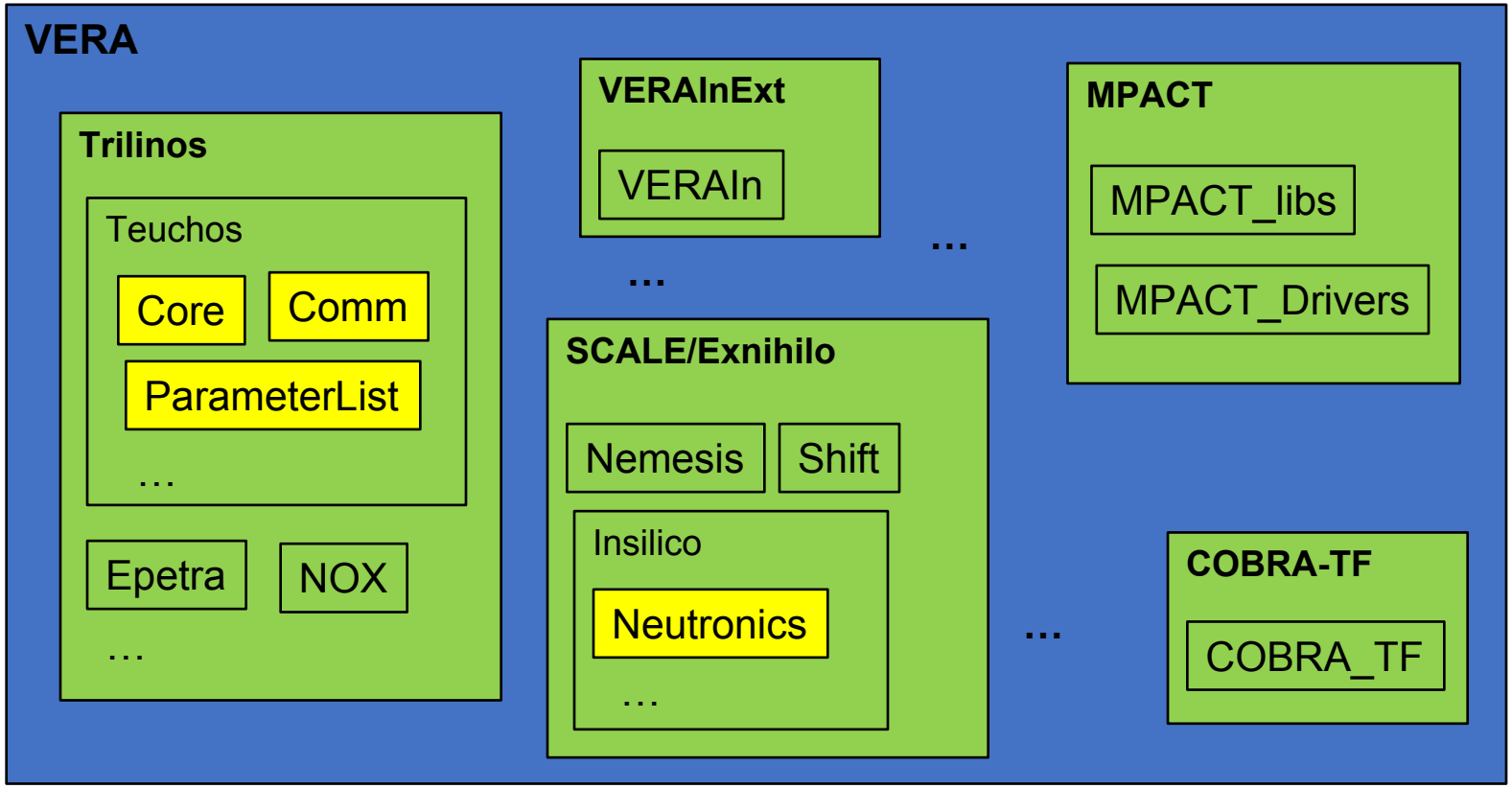
TriBITS Repository:
- Collection of Packages and TPLs
- Unit of distribution and integration
- Typically a version control (git) repository

TriBITS Package:
- Encapsulated collection of related software & tests
- Unit of testing, namespacing, documentation, and reuse
- Lists dependencies on upstream Packages & TPLs

TriBITS Subpackage:
- Optional partitioning of package software & tests
- Primarily for dependency management (SE principles)

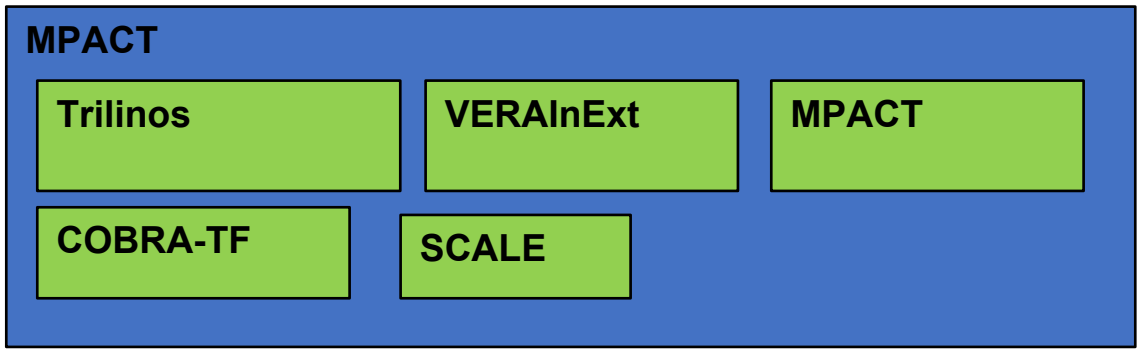
TriBITS TPLs (Third Party Libraries):
- Specification of external dependencies (libs)
- Required or optional dependency
- Single definition across all packages
- Can use standard CMake find_package(<Package>) and native Find<Package>.cmake modules

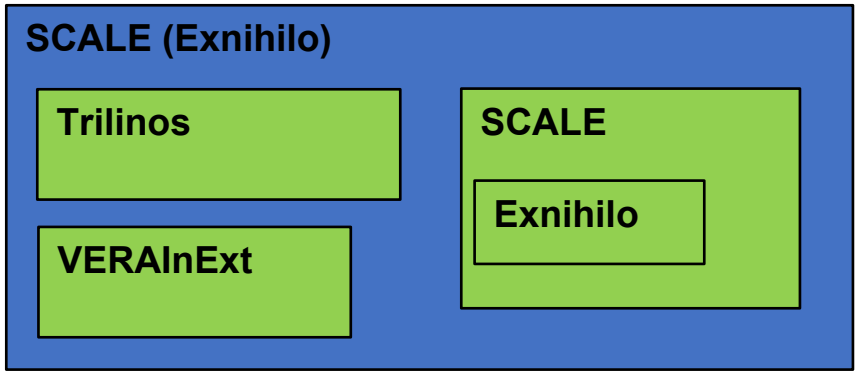
VERA Meta-Project, Repositories, Packages & Subpackages

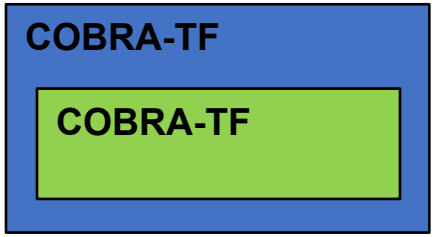


- **VERA:** Git repository and TriBITS meta-project (contains no packages)
- **TriBITS and Git repos:**: Trilinos, VERAInExt, COBRA-TF, MPACT, SCALE, Exnihilo …
- **TriBITS packages**: Teuchos, Epetra, VERAIn, Insilico, COBRA_TF, MPACT_Drivers, …
- **TriBITS subpackages**: TeuchosCore, InsilicoNeutronics …
- **TriBITS SE packages**: Teuchos, TeuchosCore, VERAIn, Insilico, InsilicNeutronics, …

Flexibility in TriBITS Projects and Repositories



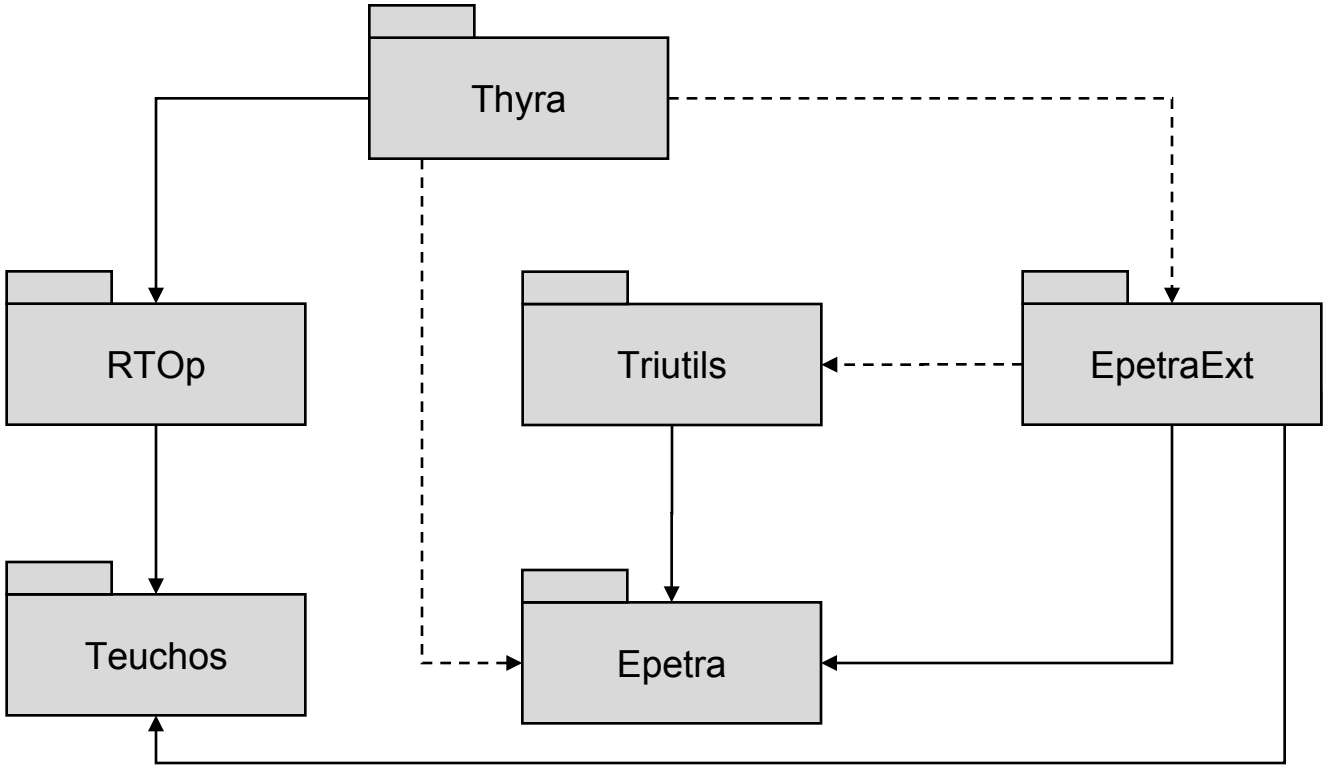




The same TriBITS repositories and packages can be arranged into multiple TriBITS projects.

Automated Package Dependency Handling

Package Dependency Structure (e.g. Trilinos)



Package Dependencies.cmake Files

Teuchos

```
tribits_package_define_dependencies(
  LIB_REQUIRED_TPLS  BLAS  LAPACK
  LIB_OPTIONAL_TPLS Boost  )
```

Epetra

```
tribits_package_define_dependencies(
  LIB_REQUIRED_TPLS  BLAS  LAPACK  )
```

RTOp

```
tribits_package_define_dependencies(
  LIB_REQUIRED_PACKAGES  Teuchos  )
```

Triutils

```
tribits_package_define_dependencies(
  LIB_REQUIRED_PACKAGES  Epetra  )
```

EpetraExt

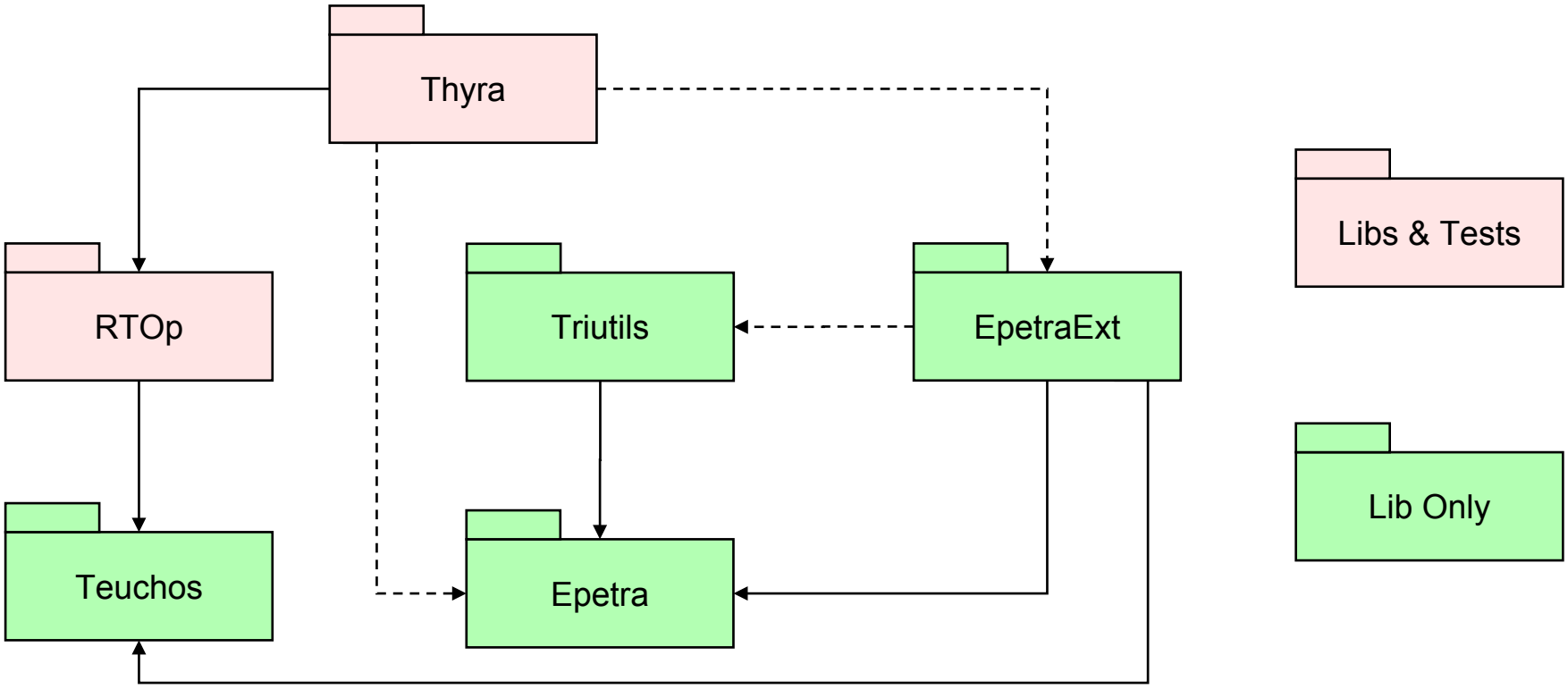
```
tribits_package_define_dependencies(
  LIB_REQUIRED_PACKAGES  Epetra  Teuchos
  LIB_OPTIONAL_PACKAGES  Triutils  )
```

Thyra

```
tribits_package_define_dependencies(
  LIB_REQUIRED_PACKAGES  RTOp  Teuchos
  LIB_OPTIONAL_PACKAGES  EpetraExt  Epera  )
```

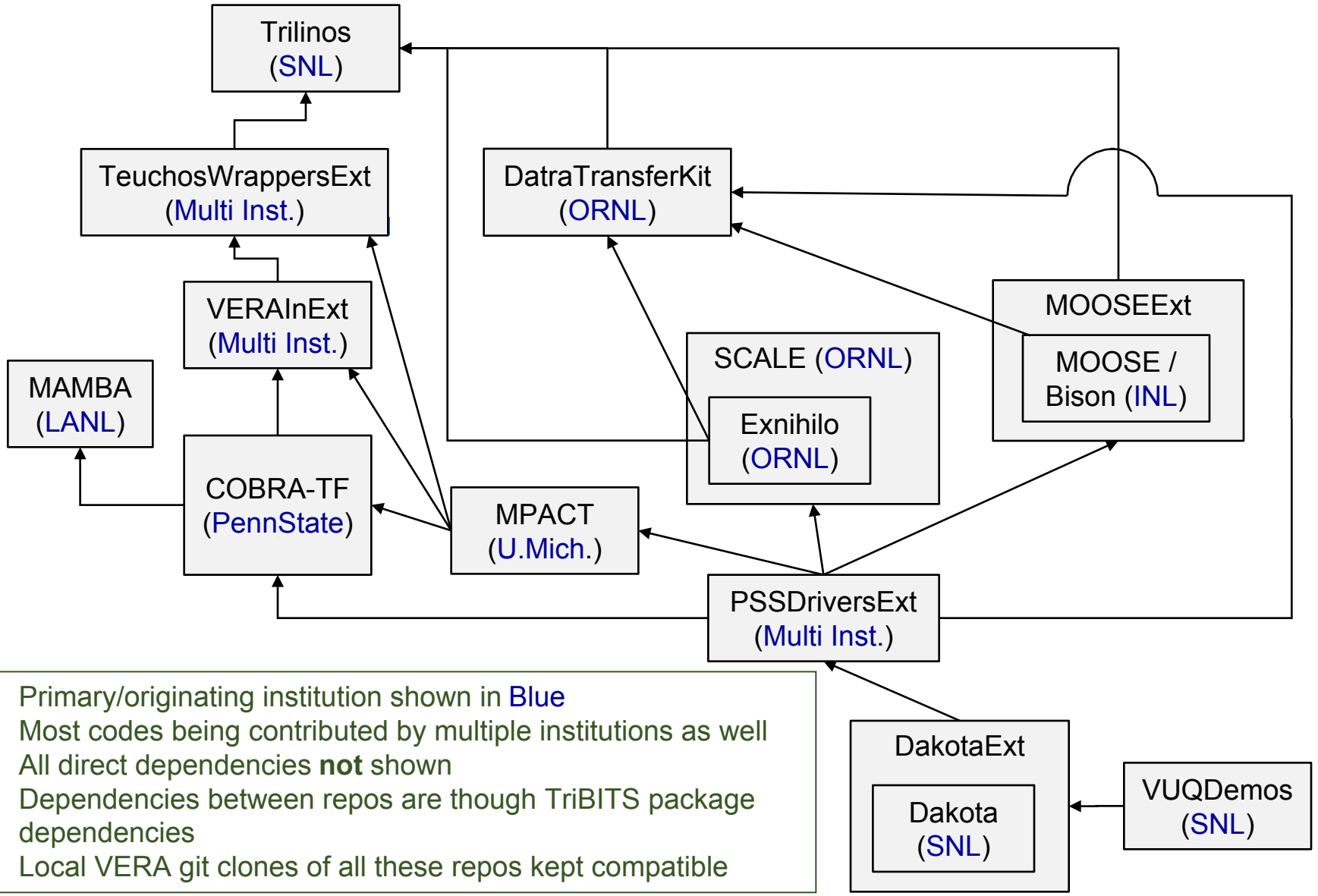
Example: CI Testing: Change RTOp

```
$ ./do-configure \
   -D Trilinos_ENABLE_RTOp=ON \
   -D Trilinos_ENABLE_ALL_FORWARD_DEP_PACKAGES=ON \
   -D Trilinos_ENABLE_TESTS=ON
```



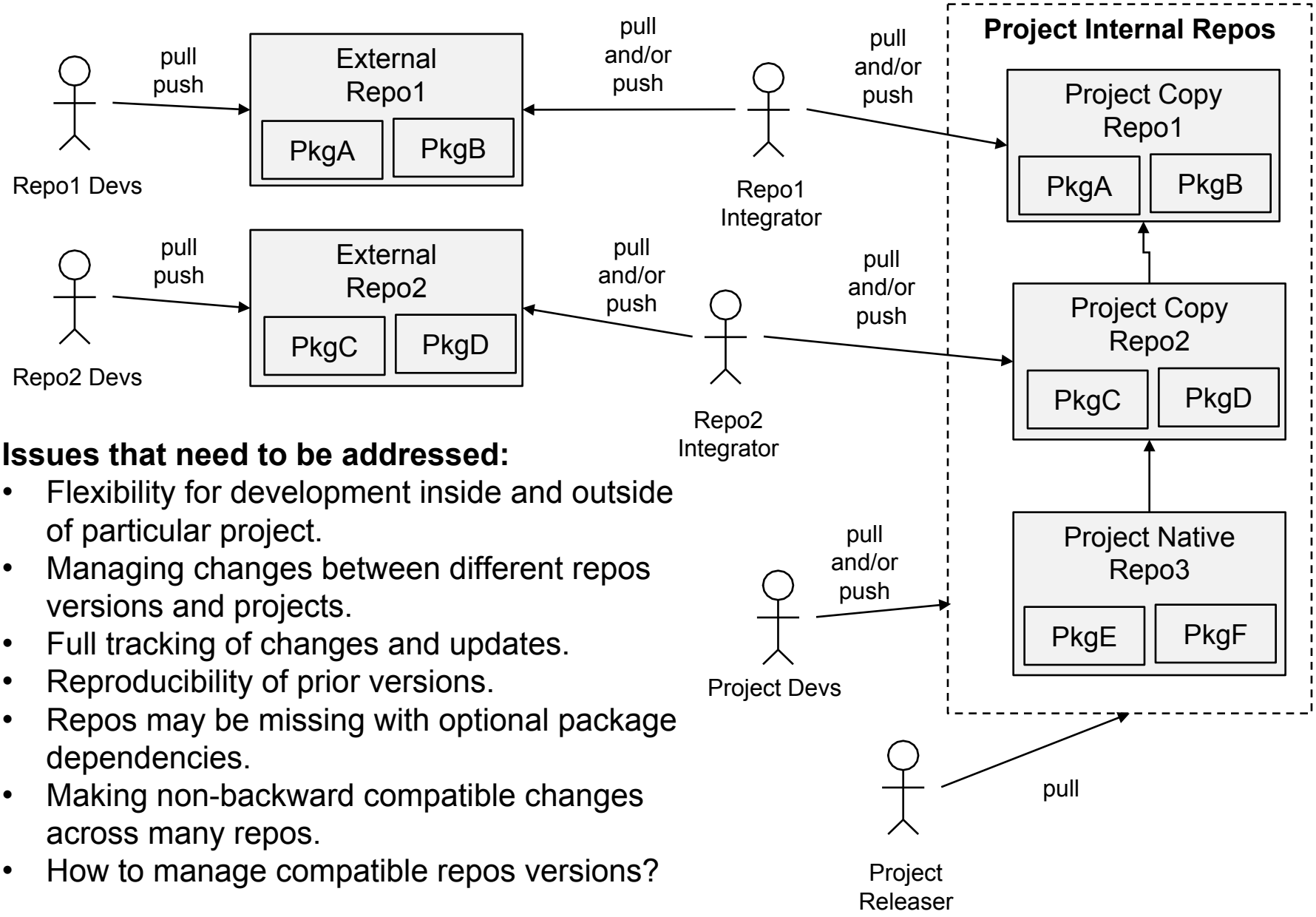
Multi-Repository Support and Integration

Dependencies Between Selected VERA Repos (2015)



- Primary/originating institution shown in Blue
- Most codes being contributed by multiple institutions as well
- All direct dependencies **not** shown
- Dependencies between repos are though TriBITS package dependencies
- Local VERA git clones of all these repos kept compatible

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?

VERA/cmake/ExtraRepositoriesList.cmake (2015)

```
tribits_project_define_extra_repositories(
  TriBITS          ""   GIT  git@casl-dev:TriBITS          ""  Continuous
  Trilinos         ""   GIT  git@casl-dev:Trilinos         ""  Continuous
  TeuchosWrappersExt "" GIT git@casl-dev:TeuchosWrappersExt ""  Continuous
  MAMBA            ""   GIT  git@casl-dev:MAMBA            ""  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
  hydrath          ""   GIT  git@casl-dev:hydrath         ""  Nightly
  PSSDriversExt    ""   GIT  git@casl-dev:PSSDriversExt   ""  Continuous
  DakotaExt       ""   GIT  git@casl-dev:DakotaExt""  Continuous
  Dakota  DakotaExt/Dakota  GIT  git@casl-dev:Dakota  NOPACKAGES  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

clone_extra_repos.py

```
$ ./clone_extra_repos.py

…

```

ID	Repo Name	Repo Dir	VC	Repo URL	Category
1	TriBITS	TriBITS	GIT	git@casl-dev:TriBITS	Continuous
2	Trilinos	Trilinos	GIT	git@casl-dev:Trilinos	Continuous
3	TeuchosWrappersExt	TeuchosWrappersExt	GIT	git@casl-dev:TeuchosWrappersExt	Continuous
4	MAMBA	MAMBA	GIT	git@casl-dev:MAMBA	Continuous
5	COBRA-TF	COBRA-TF	GIT	git@casl-dev:COBRA-TF	Continuous
6	VERAInExt	VERAInExt	GIT	git@casl-dev:VERAInExt	Continuous
7	DataTransferKit	DataTransferKit	GIT	git@casl-dev:DataTransferKit	Continuous
8	MOOSEExt	MOOSEExt	GIT	git@casl-dev:MOOSEExt	Continuous
9	MOOSE	MOOSEExt/MOOSE	GIT	git@casl-dev:MOOSE	Continuous
10	SCALE	SCALE	GIT	git@casl-dev:SCALE	Continuous
11	Exnihilo	SCALE/Exnihilo	GIT	git@casl-dev:Exnihilo	Continuous
12	MPACT	MPACT	GIT	git@casl-dev:MPACT	Continuous
13	LIMEExt	LIMEExt	GIT	git@casl-dev:LIMEExt	Continuous
14	hydrath	hydrath	GIT	git@casl-dev:hydrath	Nightly
15	PSSDriversExt	PSSDriversExt	GIT	git@casl-dev:PSSDriversExt	Continuous
16	DakotaExt	DakotaExt	GIT	git@casl-dev:DakotaExt	Continuous
17	Dakota	DakotaExt/Dakota	GIT	git@casl-dev:Dakota	Continuous
18	VUQDemos	VUQDemos	GIT	git@casl-dev:VUQDemos	Nightly

```
…

Running: git clone git@casl-dev:TriBITS TriBITS

Running: git clone git@casl-dev:Trilinos Trilinos

…
```

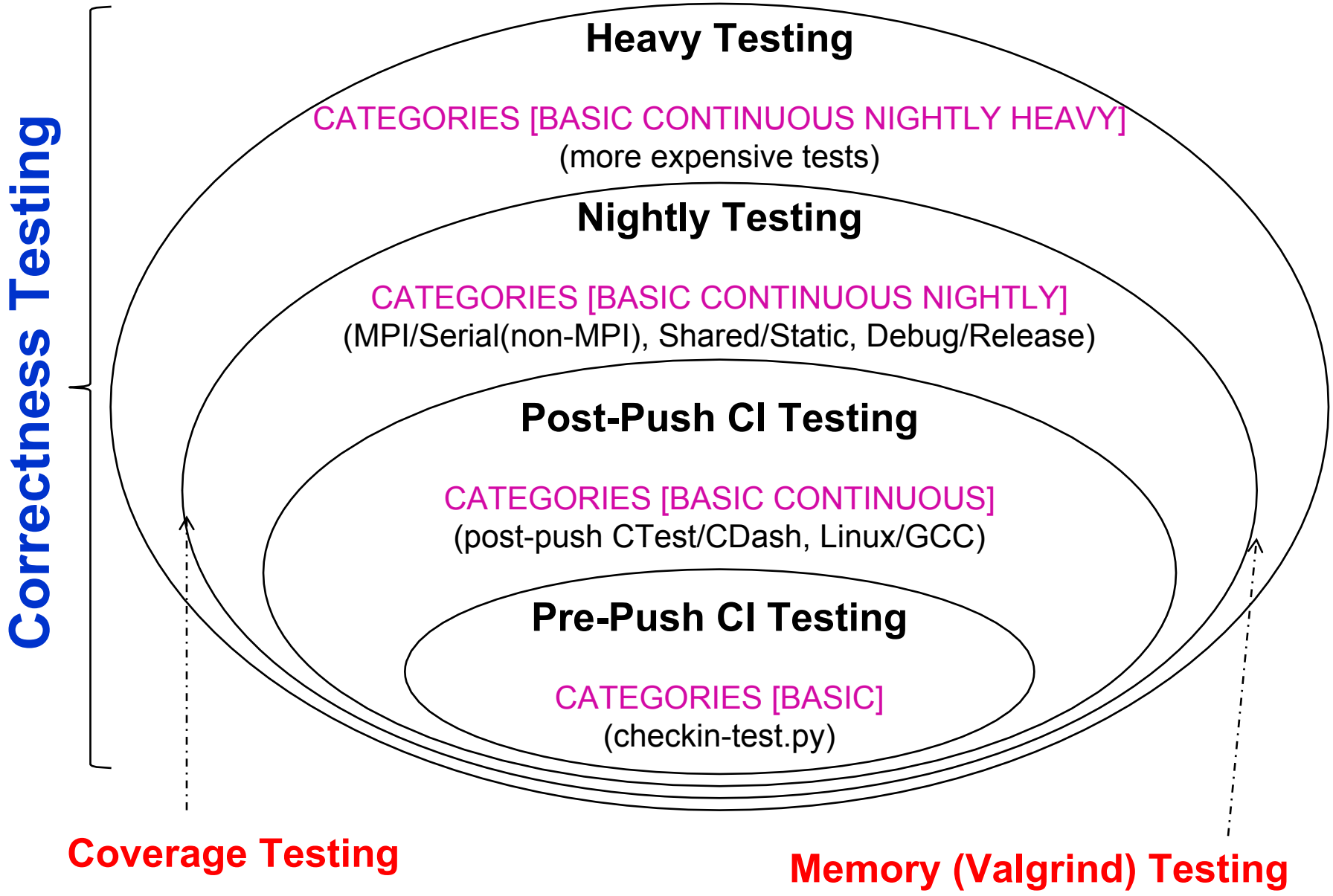
Only clones the repos that the user/developer has access to clone!

gitdist

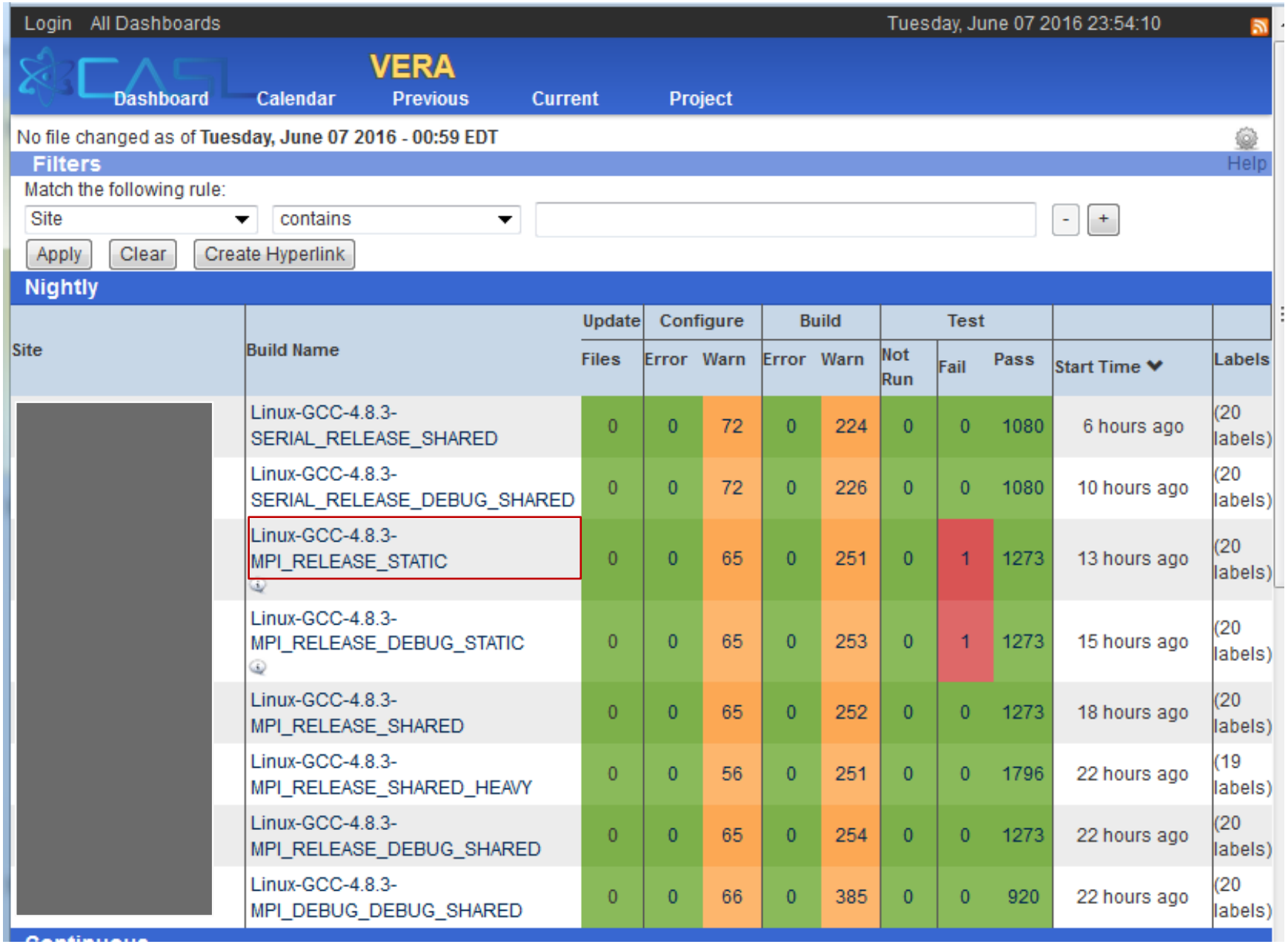
```
$ gitdist dist-repo-status

-------------------------------------------------------------------
| ID | Repo Dir            | Branch | Tracking Branch | C | M | ? |
|----|---------------------|--------|-----------------|---|---|---|
|  0 | VERA (Base)         | master | origin/master   | 2 | 1 |   |
|  1 | TriBITS             | master | origin/master   |   |   |   |
|  2 | Trilinos            | master | origin/master   |   |   |   |
|  3 | TeuchosWrappersExt  | master | origin/master   |   |   | 2 |
|  4 | MAMBA               | master | origin/master   |   |   |   |
|  5 | COBRA-TF            | master | origin/master   |   |   |   |
|  6 | VERAInExt           | master | origin/master   |   |   | 3 |
|  7 | DataTransferKit     | master | origin/master   |   |   |   |
|  8 | MOOSEExt            | master | origin/master   |   |   |   |
|  9 | MOOSEExt/MOOSE      | master | origin/master   |   |   |   |
| 10 | SCALE               | master | origin/master   |   |   |   |
| 11 | SCALE/Exnihilo      | master | origin/master   |   |   |   |
| 12 | MPACT               | master | origin/master   | 2 |   |   |
| 13 | LIMEExt             | master | origin/master   |   |   |   |
| 14 | hydrath             | master | origin/master   |   |   |   |
| 15 | PSSDriversExt       | master | origin/master   |   | 4 |   |
| 16 | DakotaExt           | master | origin/master   |   |   |   |
| 17 | DakotaExt/Dakota    | master | origin/master   |   |   |   |
| 18 | VUQDemos            | master | origin/master   |   |   |   |
-------------------------------------------------------------------

(tip: to see a legend, pass in --dist-legend.)
```

Testing Support

Standard TriBITS Test Categories and Layers



CDash: VERA (Collapsed) Builds



Login All Dashboards Tuesday, June 07 2016 23:54:10

VERA

Dashboard Calendar Previous Current Project

No file changed as of **Tuesday, June 07 2016 - 00:59 EDT**

Filters Help

Match the following rule:

Site | contains | - | +

Apply | Clear | Create Hyperlink

Nightly

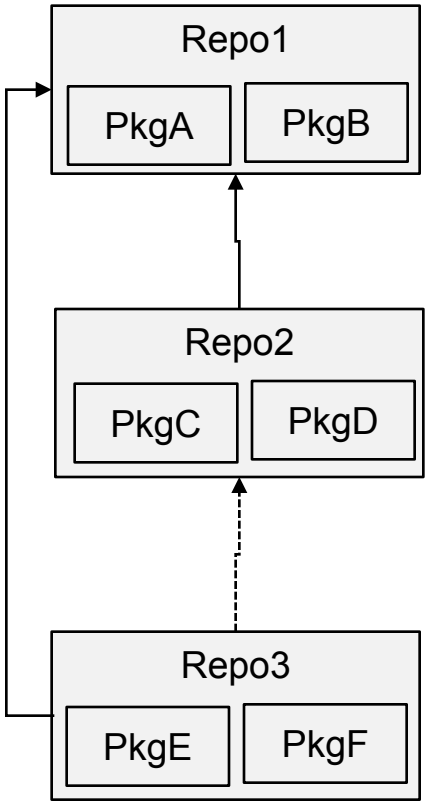
Site	Build Name	Update	Configure		Build		Test			Start Time	Labels
		Files	Error	Warn	Error	Warn	Not Run	Fail	Pass		
[redacted]	Linux-GCC-4.8.3-SERIAL_RELEASE_SHARED	0	0	72	0	224	0	0	1080	6 hours ago	(20 labels)
	Linux-GCC-4.8.3-SERIAL_RELEASE_DEBUG_SHARED	0	0	72	0	226	0	0	1080	10 hours ago	(20 labels)
	Linux-GCC-4.8.3-MPI_RELEASE_STATIC	0	0	65	0	251	0	1	1273	13 hours ago	(20 labels)
	Linux-GCC-4.8.3-MPI_RELEASE_DEBUG_STATIC	0	0	65	0	253	0	1	1273	15 hours ago	(20 labels)
	Linux-GCC-4.8.3-MPI_RELEASE_SHARED	0	0	65	0	252	0	0	1273	18 hours ago	(20 labels)
	Linux-GCC-4.8.3-MPI_RELEASE_SHARED_HEAVY	0	0	56	0	251	0	0	1796	22 hours ago	(19 labels)
	Linux-GCC-4.8.3-MPI_RELEASE_DEBUG_SHARED	0	0	65	0	254	0	0	1273	22 hours ago	(20 labels)
	Linux-GCC-4.8.3-MPI_DEBUG_DEBUG_SHARED	0	0	66	0	385	0	0	920	22 hours ago	(20 labels)

Continuous

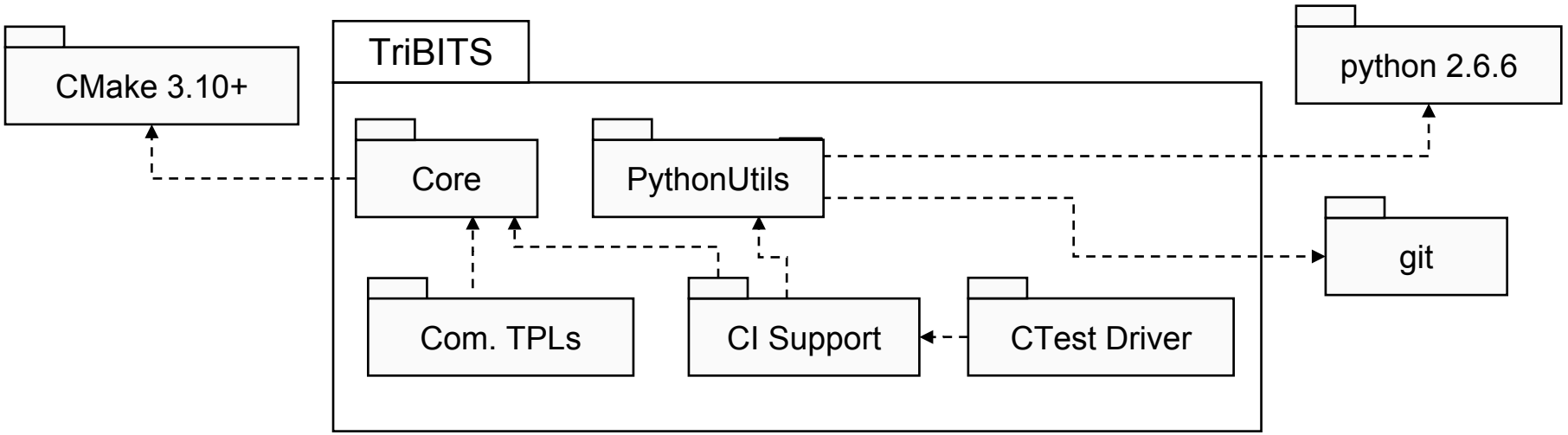
Incorporating non-TriBITS Packages

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*** (e.g. Dakota)
- **Hard case:** Upstream & downstream TriBITS packages => ***Repo2***
- **Example:** No native TriBITS build of libmesh and MOOSE/Bison for CASL VERA. Libmesh depends on DataTransferKit and therefore Trilinos.
  Tpetra <= DataTransferKit <= libmesh <= MOOSE/Bison <= Tiamat
- **Technical challenges addressed 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



TriBITS Misc. Facts and Upcoming Work



- TriBITS Partitioning and Dependencies:
  - **TriBITS Core** (tribits/core/): Core TriBITS package-based architecture for CMake projects includes configure, build, test, install, deploy (tarballs) for multi-repo projects. (1M size)
  - **TriBITS Python Utils** (tribits/python_utils/): Some basic Python utilities that are not specific to TriBITS (e.g. gitdist, snapshot_dir.py).
  - **TriBITS CI Support** (tribits/ci_support/): Support code for pre-push continuous integration testing (e.g. checkin-test.py).
  - **TriBITS CTest Driver** (tribits/ctest_driver/): Support for package-by-package testing driven by CTest submitting to CDash (e.g. TribitsCTestDriverCore.cmake).
- Upcoming Work:
  - Collect requirements for TriBITS refactoring and upgrade
  - Upgrade to use modern target-based CMake implementation and features
  - More flexibility to pre-build/install packages and linking in as external packages (TPLs)

THE END

- **Contact:** rabartl@sandia.gov
- **Sponsors:**
  - CASL: Consortium for the Advanced Simulation of Lightwater reactors

EXTRA SLIDES

TriBITS: Keeping track of compatible sets of repositories

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: ExtraRepo1/ExtraRepo2
97cf1ac [Thu Dec 1 23:34:06 2011 -0500] <author3@someurl.com>
Third summary message
...
```

Setting -D<PROJECT>_GENERATE_REPO_VERSION_FILE=ON results in <Project>RepoVersion.txt getting:

- 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.

Use with gitdist to access compatible versions:

```
$ gitdist --dist-version-file=<Project>RepoVersion.<somedate>.txt \
    [git command and arguments]
```

Using <Project>RepoVersion.txt for Dev Distributions

- Known “good” versions of the Project code are recorded as <Project>RepoVersion.txt files (e.g. archived on CDash, committed directly to base repo, etc.).
- **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 VERA/
$ 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_
```

- NOTE: A variation of the <Project>RepoVersion.txt file could be committed to the base repo (at well-defined points) to allow using git bisect to search for introduction of defects and other changes.