Introduction to LArSoft CI system

Erica Snider

Fermilab

CI development team:

Eric Church, Mark Dykstra*, Lynn Garren

Patrick Gartung, Igor Mandrichenko

Mark Mengel**, Gianluca Petrillo,

Vladimir Podstavkov**, Erica Snider

* *Summer student (2014)*

** *Lead developers*

LArSoft Architecture and Testing Workshop
June 3, 2015
Fermilab

Outline

- Introduction
  - Testing basics
  - LArSoft testing objectives
- The LArSoft CI system
  - Architecture overview
  - Components
- Configuring and running tests
- Work session
  - Test strategy
  - Developing a tiered testing framework
  - Discussion / work

Introduction

Testing basics

- The goals of testing
  - Examine an application to ensure it
    - fulfills the requirements for which it was designed
      - Does it do what we designed it to do (for all target experiments)
    - meets quality expectations
      - E.g., reliability, performance, (for LArSoft) interoperability, etc.
    - meets customer (= our!) expectations
      - Does it do what we want? Will it produce the physics results we want?
- Two basic types
  - Unit testing
  - integration testing

Unit tests

- Why unit test?
  - Unit testing improves the efficiency of the development process:
    - Finds problems early
      - Should write test as you go (or test *first*, then write)
        - *You will never understand the code better than when you first wrote it*
    - Provides good test coverage
      - Consistent unit testing program => every logical piece gets tested
    - Avoids the waste of disposable tests
      - Unit test results should always be the same, so can be used indefinitely
    - Provides a set of working examples
    - Allows a developer to change code with confidence
      - E.g., code changes to improve computing performance, to extend functionality, etc.
    - Increases the probability that code produces “correct” answers

Unit tests

- A good unit test:
  - tests a single logical concept in the system
  - is fully automated
  - has full control over all pieces running (e.g., uses mocks/stubs for isolation)
  - runs in memory (e.g., no DB or file access)
  - can be run in any order, if together with other tests
  - consistently returns the same result (e.g., no random numbers)
  - runs fast
  - is readable
  - is maintainable
  - is trustworthy (i.e., when the test fails, it means your code is broken!)

Unit tests

- A good unit test:
  - tests a single logical concept in the system
  - is fully automated
  - has full control over all pieces running (e.g., uses mocks/stubs for isolation)
  - runs in memory (e.g., no DB or file access)
  - can be run in any order, if together with other tests
  - consistently returns the same result (e.g., no random numbers)
  - runs fast
  - is readable
  - is maintainable
  - is trustworthy (i.e., when the test fails, it means your code is broken!)

Integration tests

- Logical extension of unit testing
  - Identifies problems when “units” are combined
- Any test that uses “lar -c …”
  - Tests of one or more modules and services
  - Tests of reconstruction or simulation chain
  - Tests that check readability of data
  - etc.

Regression testing

- A testing strategy by which
  - Existing tests are run against modified code
    - Checks whether code changes break anything that worked prior to the change
  - Write new tests only where necessary

This is how the LArSoft testing system is designed to work

LArSoft tests

- Tests within the LArSoft context
  - Any command / script / program that:
    - tests some piece of code
    - exits with 0 when the test passes, or a non-zero value when the test fails
  - Unit tests
    - Utilizes 'make test' during the build procedure
    - Configured via CMakeLIsts.txt files
    - Tests run prior to mrb install phase
  - Integration tests
    - Tests run by LArSoft CI system scripts
    - Configured via text files under the test sub-directory of each repository
    - Test run after mrb install phase

LArSoft testing objectives

- Maintain a capability to:
  - Identify major problems before each individual integration or soon after
    - Support “continuous integration” (CI)
    - “Major” = build failures, detector interoperability, crashes, missing functionality, data file backward compatibility
  - Track changes in computing performance metrics over time
    - E.g, identify unexpected changes in CPU performance or memory usage
  - Ensure that develop branch always builds and runs
  - Ensure that code tagged for release operates as expected prior to release
    - Contributes to release validation to some extent – a much larger topic than CI

LArSoft testing objectives

- Provide a framework, automated tools that make testing easy
  - Simple: tests can be arbitrary scripts or use built-in features of the system
  - Easily configurable: configuration files in source code
  - Flexible: can aggregate tests, create workflows of dependent tests, etc.
  - Conveniently monitored: view results in layers of detail via web GUI
  - Manageable: scripts + http interface for initiating tests
    - Many ways to trigger tests
  - User friendly: e.g., can run everything or parts of the system locally
    - Anywhere LArSoft is installed

The LArSoft CI system

LArSoft CI system

- Five major components
  - The Central Build Service
  - Results database / web server
  - Server-side driver software
  - Client-side driver software
  - Monitoring/reporting software
    - Web GUI interface

See https://cdcvs.fnal.gov/redmine/projects/lar-ci/wiki

LArSoft CI system

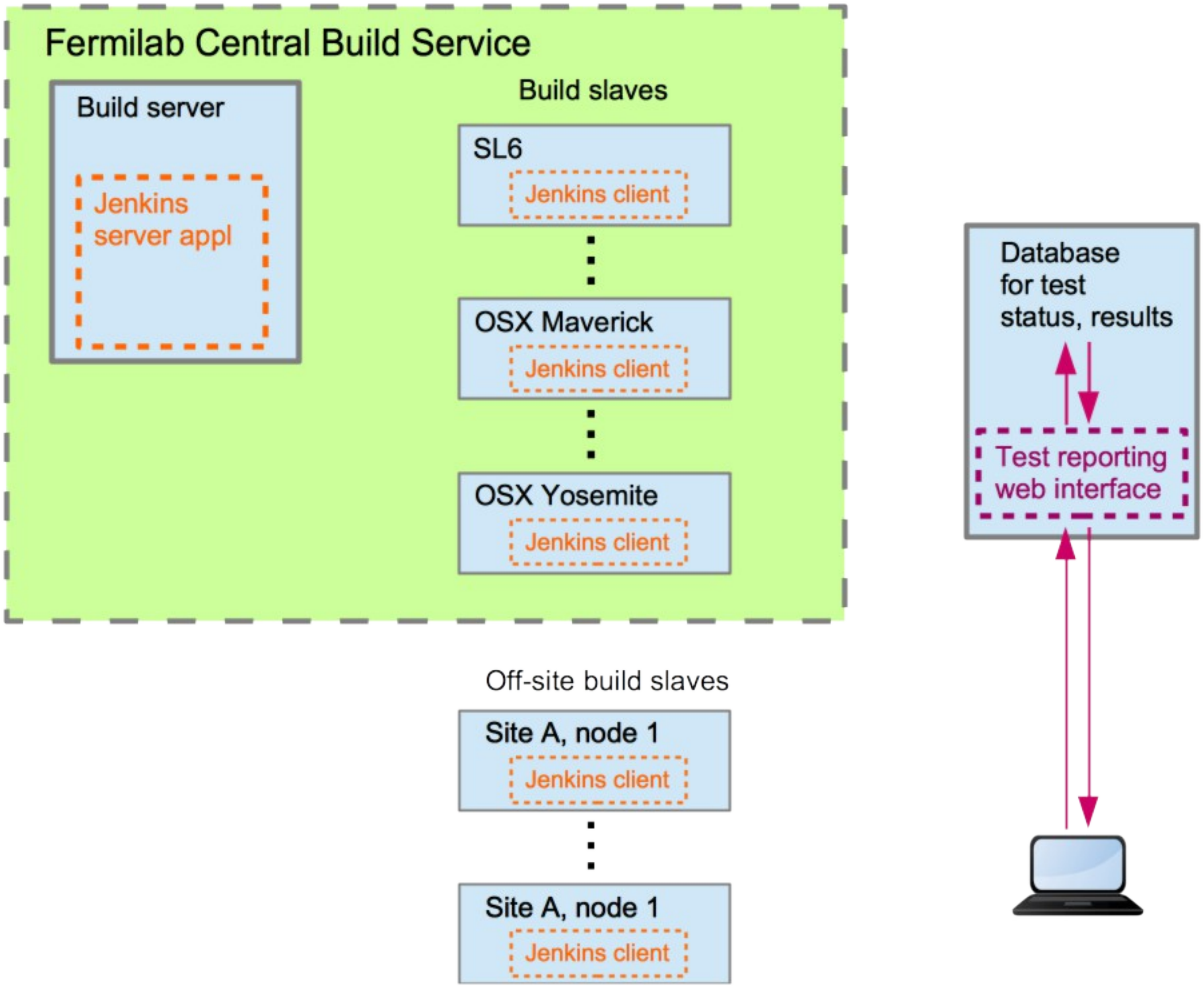
- Five major components
  - The Central Build Service
  - Results database / web server
  - Server-side driver software
  - Client-side driver software
  - Monitoring/reporting software
    - Web GUI interface

Operated by Fermilab

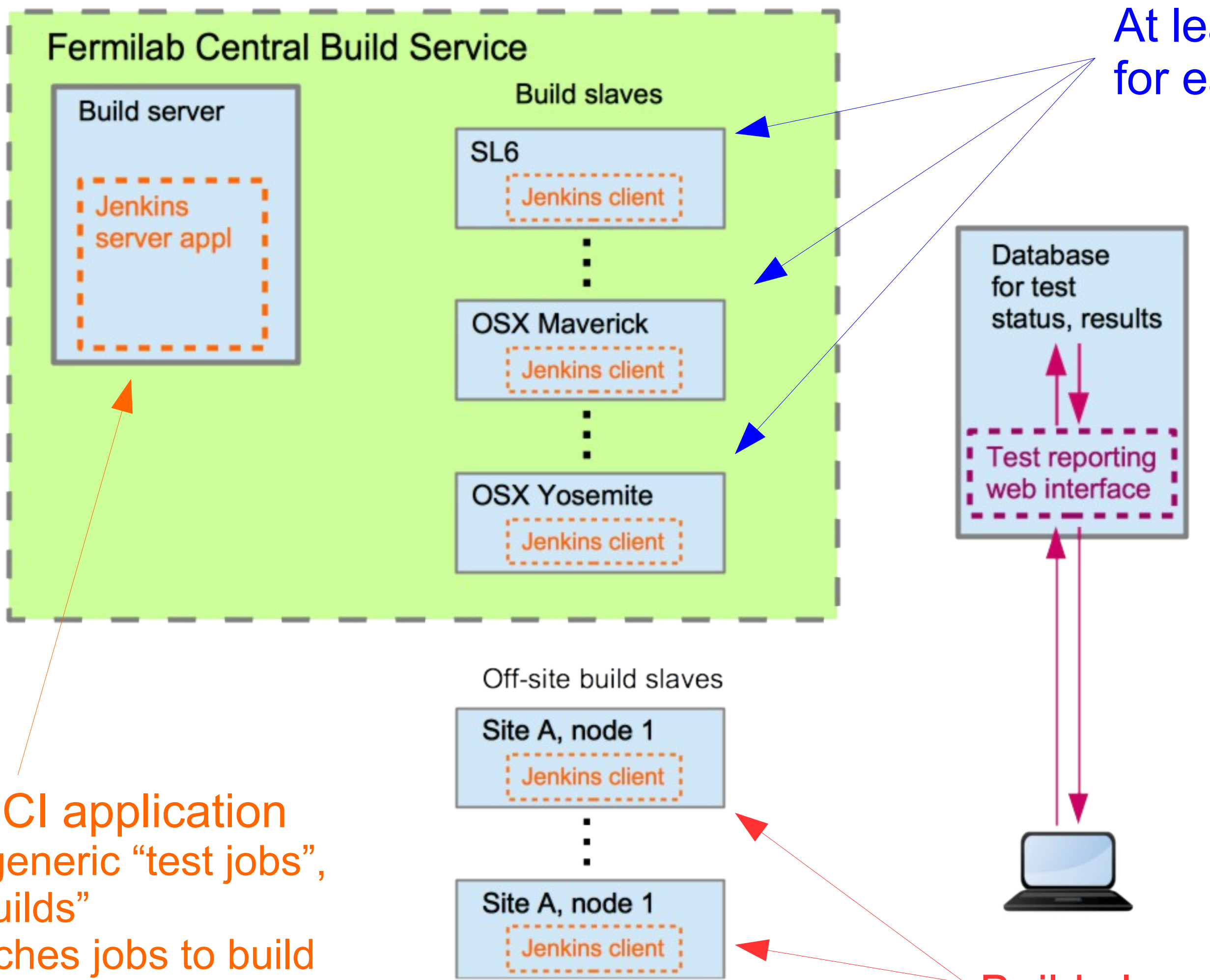
Written, maintained by LArSoft

See https://cdcvs.fnal.gov/redmine/projects/lar-ci/wiki

CI system overview



CI system overview



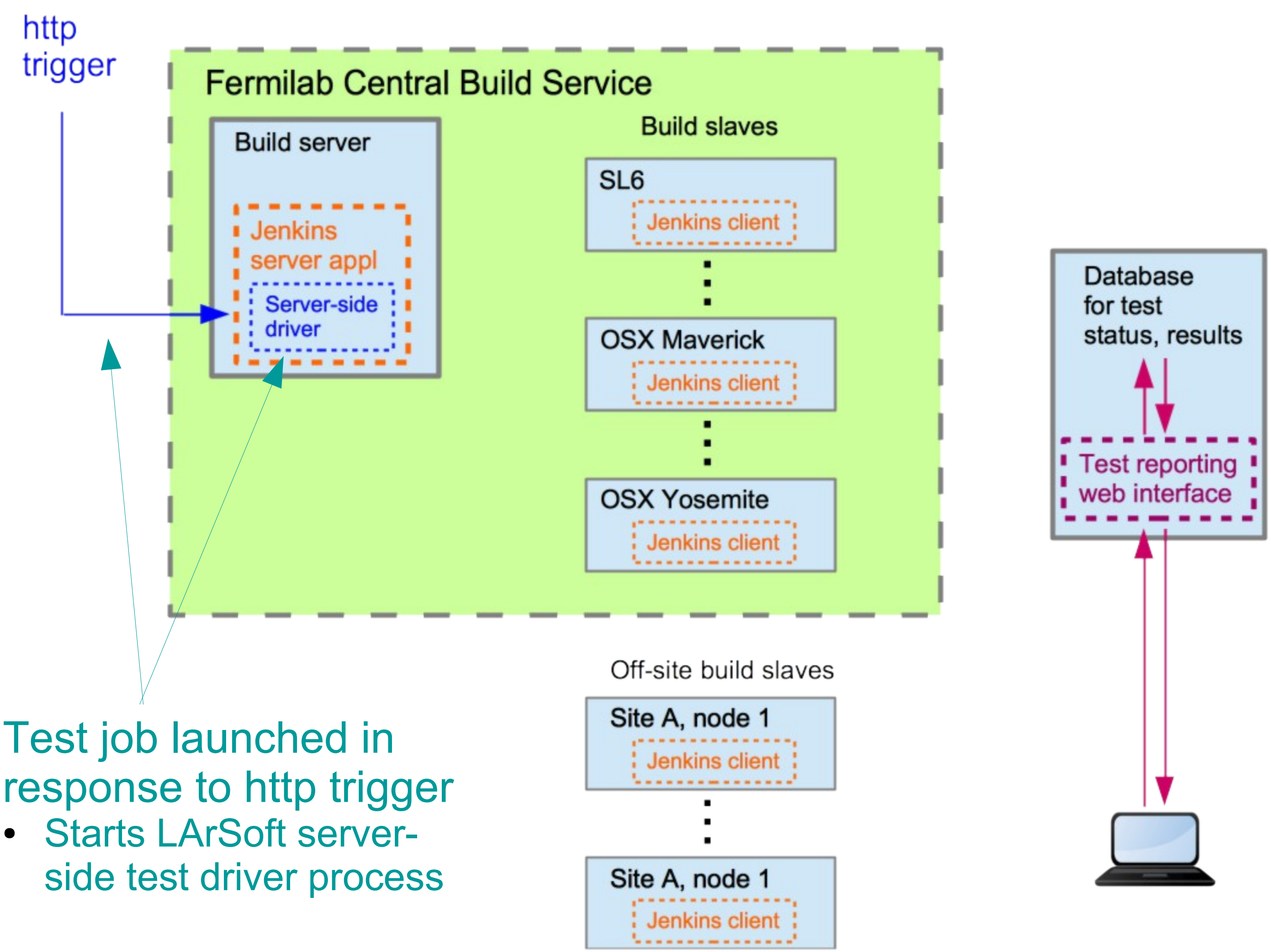
At least on build slave for each supported OS

Jenkins CI application
- Runs generic “test jobs”, aka “builds”
- Dispatches jobs to build slaves

Build slaves can be located anywhere.

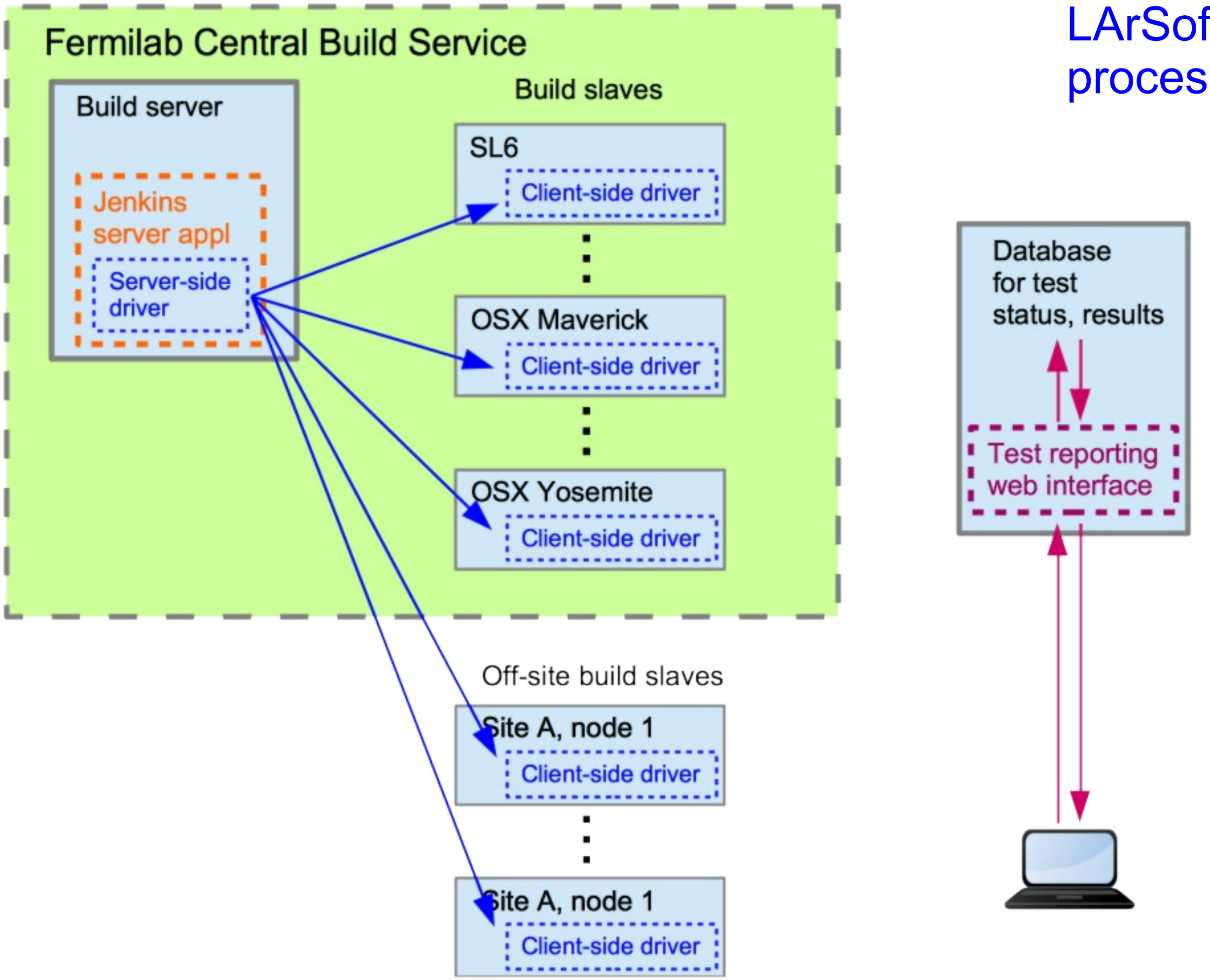
CI system overview

Arguments in trigger specify run-time configuration options
- The release to use
- The branch to build
- The “test suite” to run (more on this later)
- Other options



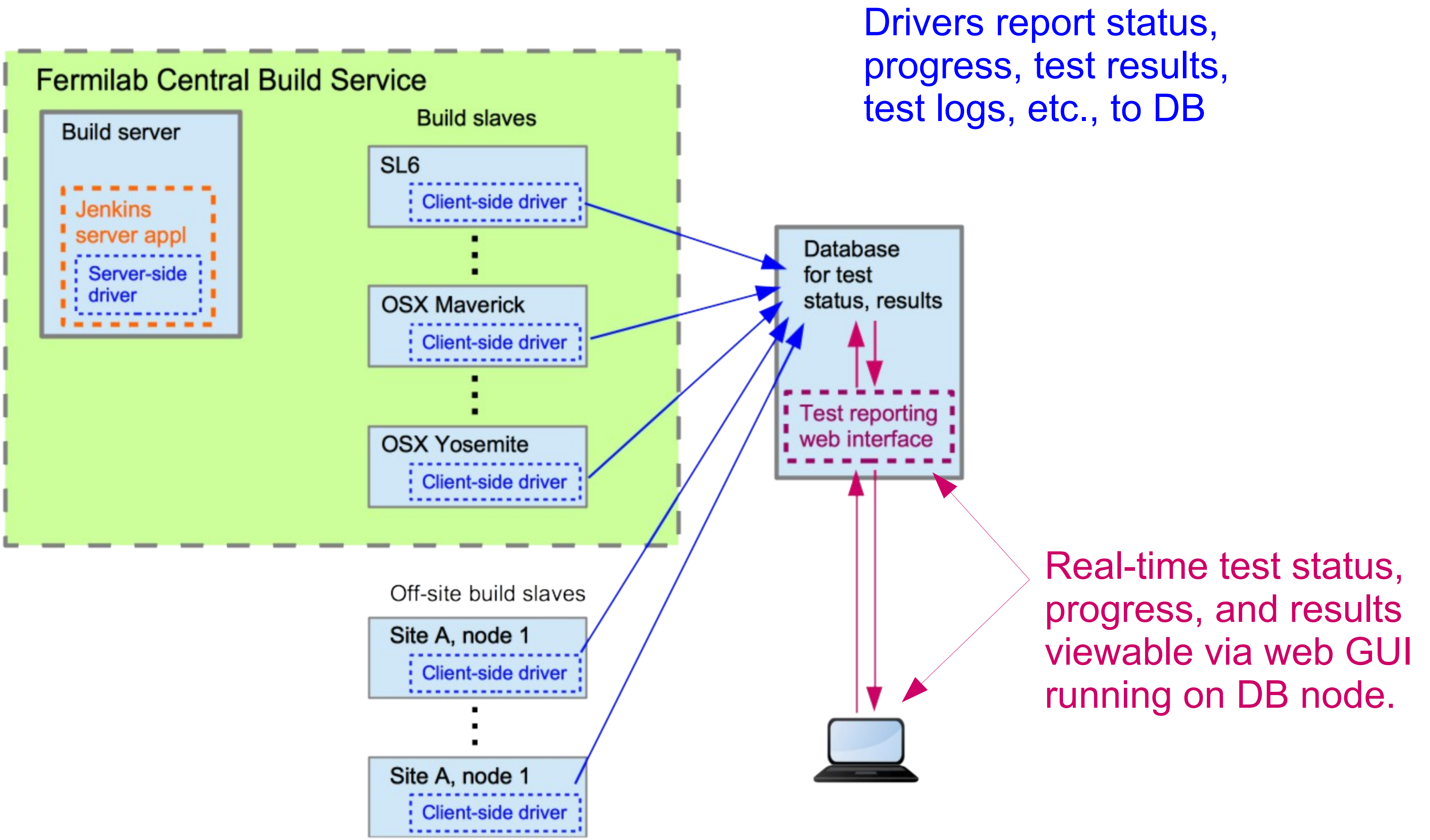
Test job launched in response to http trigger
- Starts LArSoft server-side test driver process

CI system overview

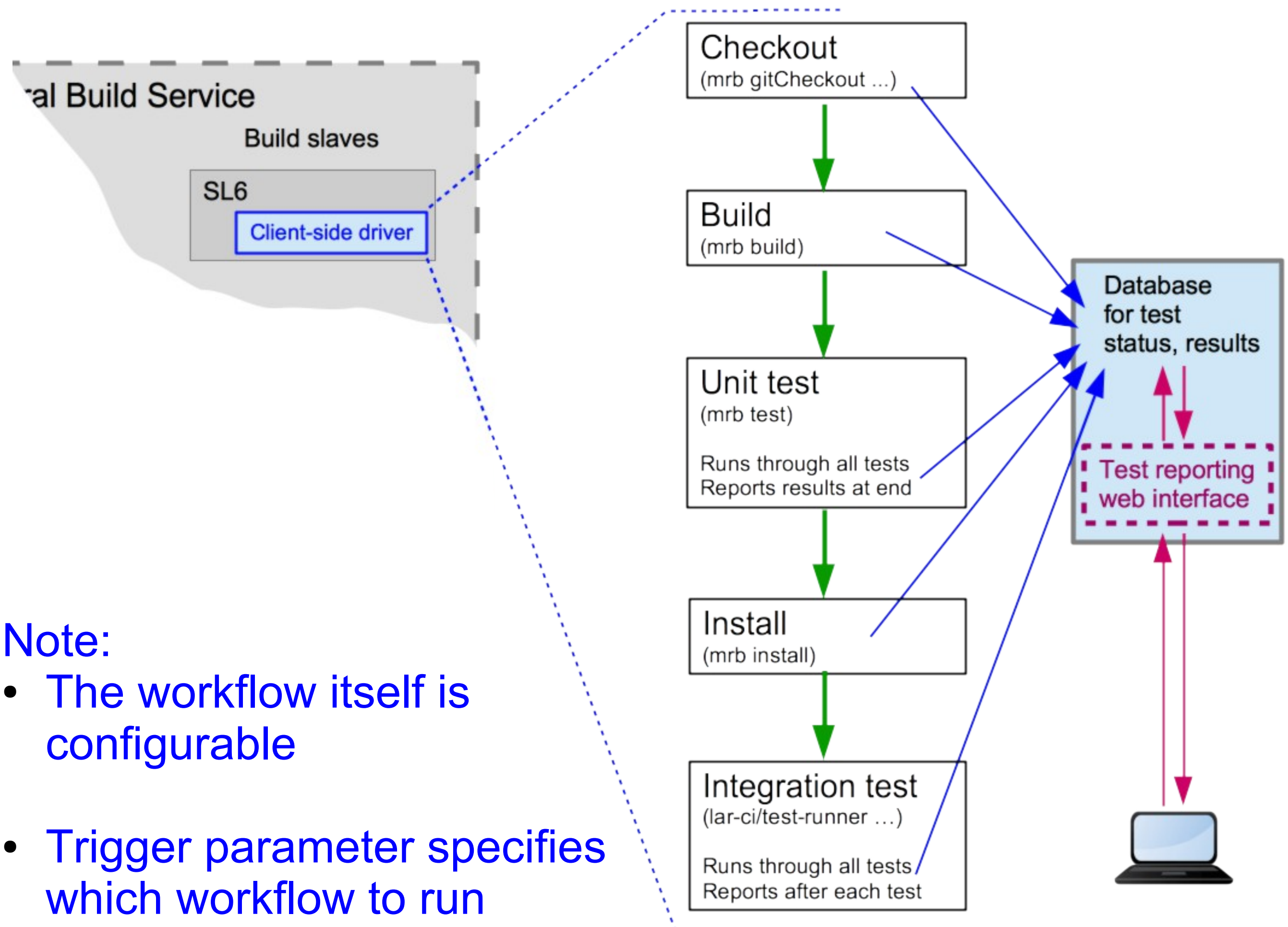


Server dispatches LArSoft client-side driver processes to build slaves

CI system overview



The default test workflow



A LArSoft script runs in each step of the workflow

The workflow terminates if any step exits with a non-zero return value

The reporting system displays the status of each step according to the return value

Note:

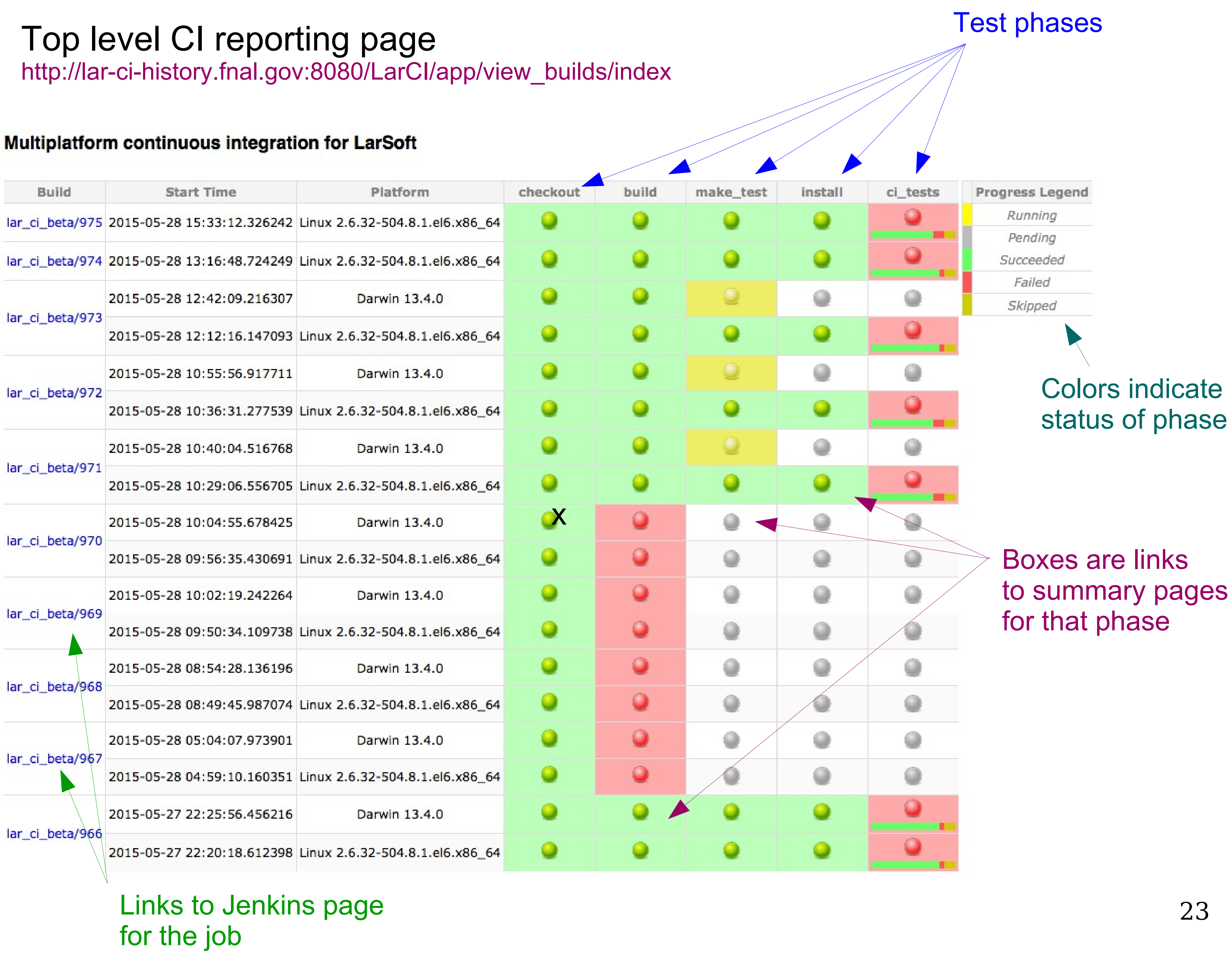
- The workflow itself is configurable
- Trigger parameter specifies which workflow to run

Test monitoring and reports

- Status of running test jobs and results of completed test jobs
  - http://lar-ci-history.fnal.gov:8080/LarCI/app/view_builds/index

Top level CI reporting page

http://lar-ci-history.fnal.gov:8080/LarCI/app/view_builds/index

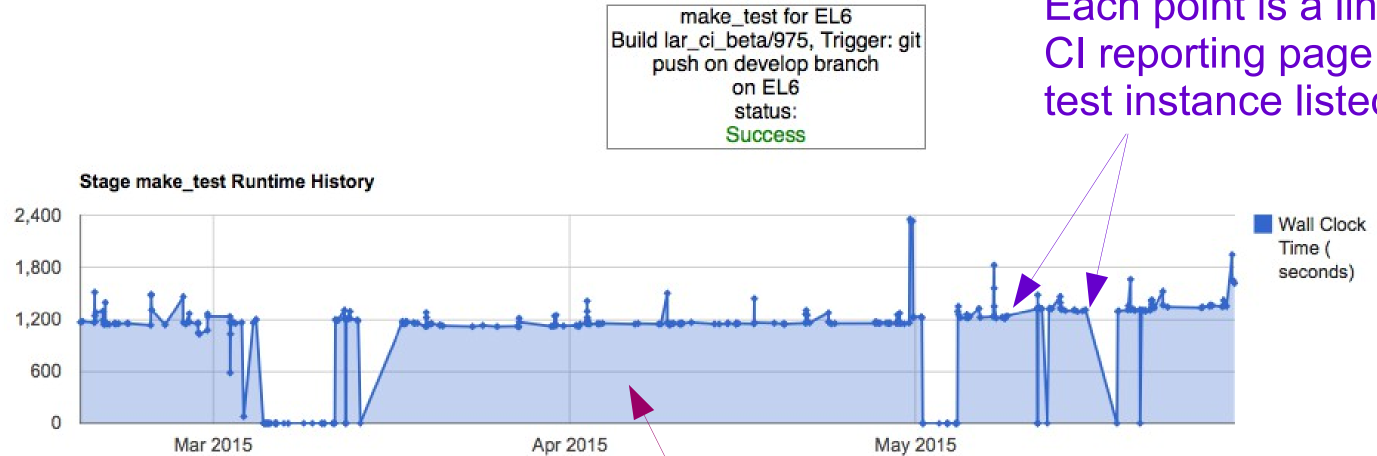


Multiplatform continuous integration for LarSoft

Build	Start Time	Platform	checkout	build	make_test	install	ci_tests
lar_ci_beta/975	2015-05-28 15:33:12.326242	Linux 2.6.32-504.8.1.el6.x86_64					
lar_ci_beta/974	2015-05-28 13:16:48.724249	Linux 2.6.32-504.8.1.el6.x86_64					
lar_ci_beta/973	2015-05-28 12:42:09.216307	Darwin 13.4.0					
	2015-05-28 12:12:16.147093	Linux 2.6.32-504.8.1.el6.x86_64					
lar_ci_beta/972	2015-05-28 10:55:56.917711	Darwin 13.4.0					
	2015-05-28 10:36:31.277539	Linux 2.6.32-504.8.1.el6.x86_64					
lar_ci_beta/971	2015-05-28 10:40:04.516768	Darwin 13.4.0					
	2015-05-28 10:29:06.556705	Linux 2.6.32-504.8.1.el6.x86_64					
lar_ci_beta/970	2015-05-28 10:04:55.678425	Darwin 13.4.0					
	2015-05-28 09:56:35.430691	Linux 2.6.32-504.8.1.el6.x86_64					
lar_ci_beta/969	2015-05-28 10:02:19.242264	Darwin 13.4.0					
	2015-05-28 09:50:34.109738	Linux 2.6.32-504.8.1.el6.x86_64					
lar_ci_beta/968	2015-05-28 08:54:28.136196	Darwin 13.4.0					
	2015-05-28 08:49:45.987074	Linux 2.6.32-504.8.1.el6.x86_64					
lar_ci_beta/967	2015-05-28 05:04:07.973901	Darwin 13.4.0					
	2015-05-28 04:59:10.160351	Linux 2.6.32-504.8.1.el6.x86_64					
lar_ci_beta/966	2015-05-27 22:25:56.456216	Darwin 13.4.0					
	2015-05-27 22:20:18.612398	Linux 2.6.32-504.8.1.el6.x86_64					

Unit test (“make_test”) summary page

Each point is a link to the top-level CI reporting page with the selected test instance listed at the top



Total execution time (wall clock) for unit test phase vs. test instance

Phase: make_test

[make_test](#)
Started 2015-05-28 15:41:14.356933

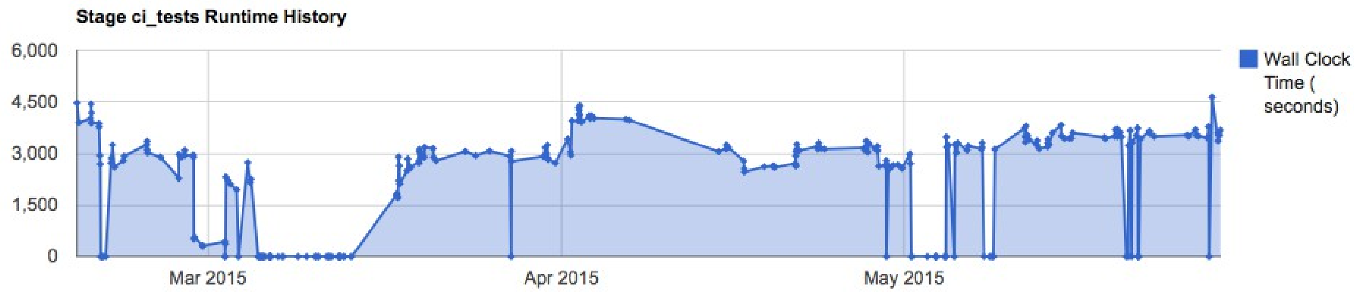
- geometry_iterator_test
- Wire_test
- donothing_lbne35t
- donothing_simul_lbne35t
- build_oplib_lbne35t
- geometry_microboone
- optical_digi_lbne35t
- SurfYZTest
- gensingle
- timingreference_test
- KalmanFilterTest
- testPhysicalConstants
- BulkAllocator_test
- SurfXYZTest
- geometry_iterator_uboone_test
- LATest

Links to result summary page for specific unit tests

Integration test ("ci_test") summary page

Same basic information and layout

ci_tests for EL6
Build lar_ci_beta/975, Trigger: git push on develop branch on EL6
status:
Failed



Phase: ci_tests

ci_tests
Started 2015-05-28 16:11:19.996761

This test failed

- lar_ci_openold_detsim_lbnecode
- lar_ci_openold_detsim_uboonecode
- lar_ci_hitana_g4_uboonecode
- lar_ci_prodsingle_lbnecode
- lar_ci_hitana_tinyana_new_uboonecode
- lar_ci_hitana_detsim_uboonecode
- lar_ci_prodgenie_uboonecode
- lar_ci_hitana_tinyana_canonical_uboonecode
- lar_ci_hitana_prod_uboonecode
- lar_ci_histocomp_uboonecode
- lar_ci_hitana_reco2D_uboonecode
- lar_ci_prodsingle_uboonecode
- lar_ci_openold_detsim3d_uboonecode (Skip reason: lar_ci_openold_detsim3d_uboonecode -- prereq lar_ci_openold_detsim2d_uboonecode failed)
- lar_ci_openold_detsim2d_uboonecode (Skip reason: lar_ci_openold_detsim2d_uboonecode -- prereq lar_ci_openold_detsim_uboonecode failed a)
- lar_ci_openold_reco_lbnecode

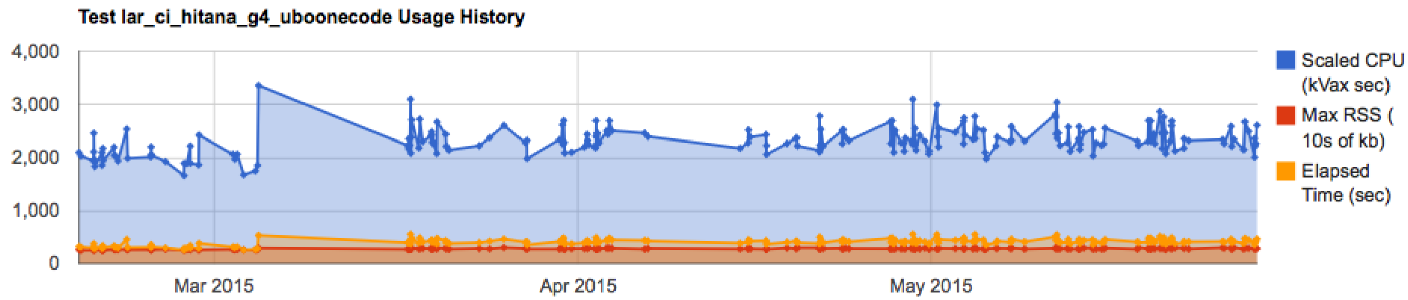
These tests were skipped because a pre-requisite test (above) failed.

The reason is listed here

Finished 2015-05-28 17:12:38.099697
exit code: 4

Summary of test lar_ci_hitana_g4_uboonecode

lar_ci_hitana_g4_uboonecode on EL6
Build lar_ci_beta/975
Trigger: git push on develop branch
Failed
Success



Test: lar_ci_hitana_g4_uboonecode

stdout
stderr

Links to output created by the test

Started 2015-05-28 16:24:07.244806	
exitcode	0
rusage user cpu	465.610000
rusage scaled user cpu	2607.048168
rusage system cpu	0.770000
rusage scaled system cpu	4.311392
rusage elapsed	466.810000
rusage %cpu	99.000000
rusage avgtext	0.000000
rusage avgdata	0.000000
rusage maxrss	2914704.000000
rusage inputs	880.000000
rusage outputs	189632.000000
rusage major faults	0.000000
rusage minor faults	162258.000000
rusage swaps	0.000000
max moduletimesrns	0.00554585
min moduletimesrns	0.000252962
avg moduletimesrns	0.00047065615
max moduletimeslargeant	25.6057
min moduletimeslargeant	3.87427
avg moduletimeslargeant	10.1806505
max moduletimesmcreco	2.19556

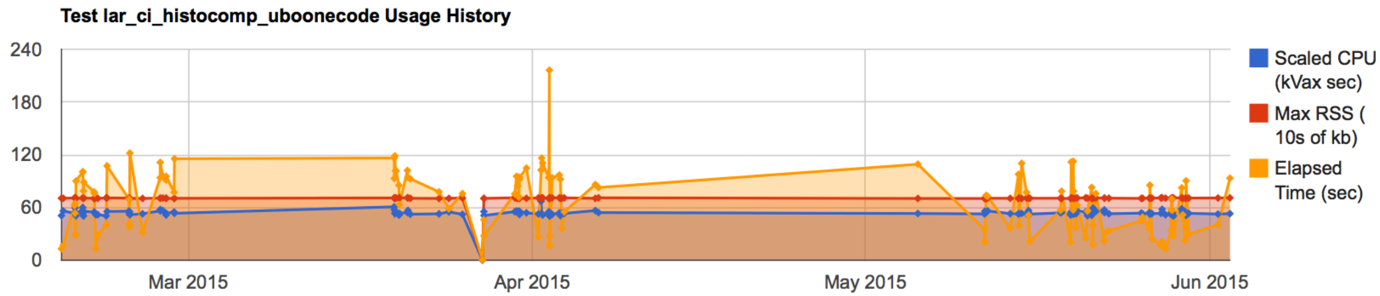
v2.0 display for this test

v2.1.2 will have all times for a given module on a single line.

\+ Plot will show times for each module

Summary of test lar_ci_histcomp_uboonecode

lar_ci_histocomp_uboonecode on EL6
Build lar_ci_beta/980
Trigger: git push on develop branch
Failed
Failed



Plot image(s) created by the test job

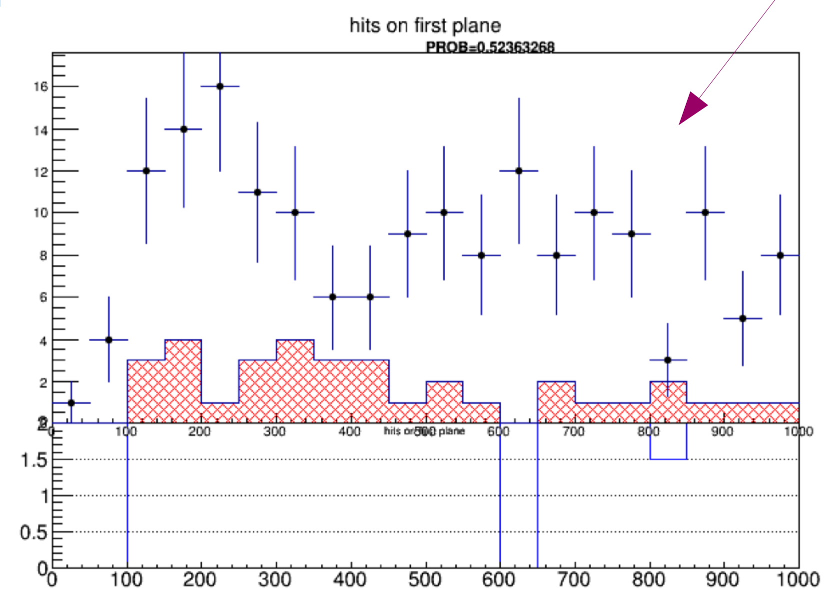
Test: lar_ci_histocomp_uboonecode

stdout
stderr
histcomp

Started 2015-05-30 02:11:37.214453	
min_histo_compare_prob	0.04147194
exitcode	256
rusage_user_cpu	9.550000
rusage_scaled_user_cpu	53.472456
rusage_system_cpu	2.190000
rusage_scaled_system_cpu	12.262270
rusage_elapsed	29.620000
rusage_%cpu	39.000000
rusage_avgtext	0.000000
rusage_avgdata	0.000000
rusage_maxrss	724656.000000
rusage_inputs	0.000000
rusage_outputs	15912.000000
rusage_major_faults	0.000000
rusage_minor_faults	287837.000000
rusage_swaps	0.000000
Finished 2015-05-30 02:14:12.010359 exit code: 1.0	

Comment:

hits1



Comment:

Configuring and running tests

Integration test configuration

- Defined in INI-formatted configuration files
  - Python config library rules
  - Live in the source repositories: <repo>/test/ci/ci_test.cfg file
- Two types of sections within the file
  - Test definition (keyword = test)
    - Specifies the command and arguments to run for the test
    - Files to copy in prior to the test, out after the test command
      - e.g., input data for test + reference data for comparison of result
    - Tests that must be run first
    - Various checks to perform
  - Test suite definition (keyword = suite)
    - Specifies a collection of tests to run as a unit in a single test job
      - The test suite is specified as a trigger argument
    - Dependencies taken into account, run in the correct order

Integration test configuration file

More information at https://cdcvs.fnal.gov/redmine/projects/lar-ci/wiki/Test_Runner_Introduction

- Basic layout for ci_tests.cfg file

Test definition blocks

Test suite definition
(can be more than one)

```
# Simple ci_tests.cfg file

# Definition for a named 'testA' that uses the output from 'testB'
[test testA]
script=$<PRODUCT>_DIR/test/testA.sh
args= -a qualA -b qualB
requires= testB

# Definition for 'testB'
[test testB]
script=$<PRODUCT>_DIR/test/testB.sh
...

# Definition for 'testC”
[test testC]
script = $PRODUCT_DIR/test/testC.sh
...

# Definition of test suite 'test_suiteA'
[suite test_suiteA]
testlist = testA testC
```

Integration test configuration file

More information at https://cdcvs.fnal.gov/redmine/projects/lar-ci/wiki/Test_Runner_Introduction

- Basic layout for ci_tests.cfg file

```
# Simple ci_tests.cfg file

# Definition for a named 'testA' that uses the output from 'testB'
[test testA]
script=$<PRODUCT>_DIR/test/testA.sh
args= -a qualA -b qualB
requires= testB
```

Creates dependency between tests

Pre-requisites will be run first

```
# Definition for 'testB'
[test testB]
script=$<PRODUCT>_DIR/test/testB.sh
...

# Definition for 'testC”
[test testC]
script = $PRODUCT_DIR/test/testC.sh
...
```

In this case, 'testB' will be run as part of the suite due to declared dependency above

```
# Definition of test suite 'test_suiteA'
[suite test_suiteA]
testlist = testA testC
```

Integration test configuration file

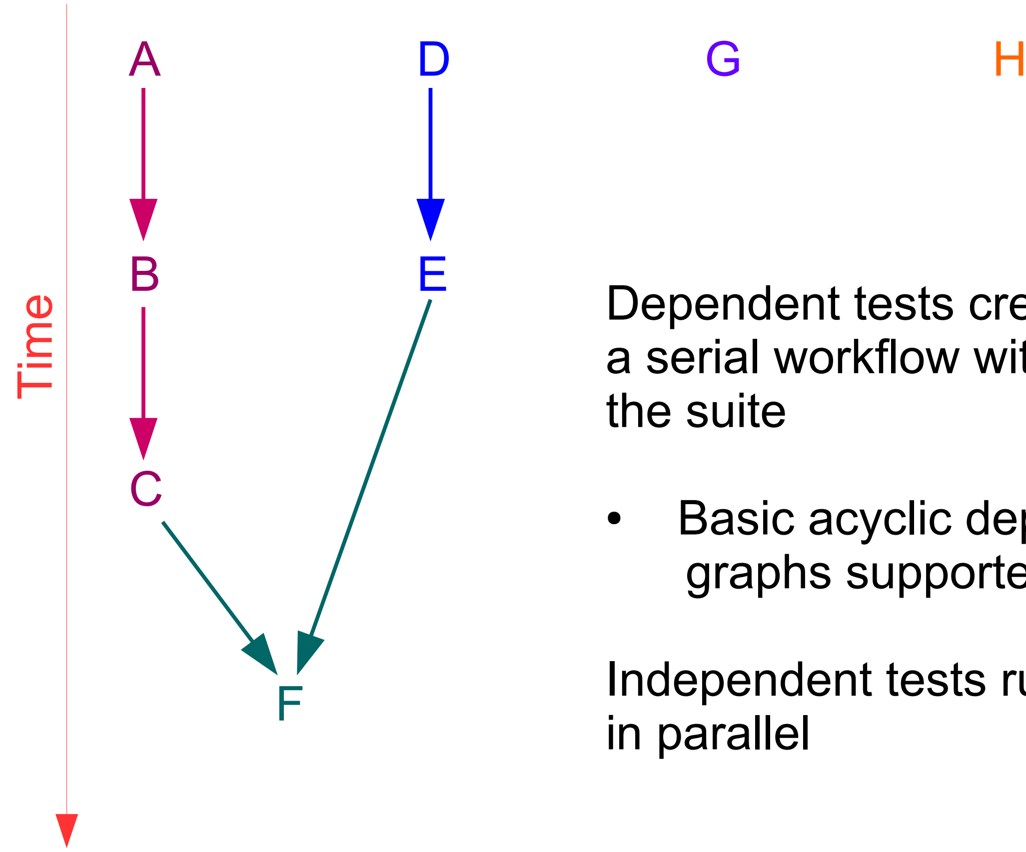
More information at https://cdcvs.fnal.gov/redmine/projects/lar-ci/wiki/Test_Runner_Introduction

- Test section keywords
  - script = <command name>   [required!!]
    - The command to run. Fully qualified path or relative to directory with
  - args = <argument list>
  - requires = <test name 1> [<test name 2> [...]]
    - Pre-requisite tests will be run before dependent tests
  - cpu_usage_range = <low value>:<high value>
    - Test fails if CPU time exceeds the range on either end
  - mem_usage_range = <low value>:<high value>
    - Test fails if memory usage exceeds the range on either end
    - NOTE: currently limited to the maximum for any test run, not a specific test...
  - outputN = <filename N>
    - File sanity checks for N=[1...9]: exists, more than 5 bytes, *.root files start with 'root'
  - check_histograms = <root hist file A> <root hist file B> <min K-S prob value>
    - Runs K-S test on histograms with same name. Test fails if any K-S prob < specified min
    - Produces web page with histogram overlays and K-S probability values
  - parse_art_output = True
    - Reads art log files, parses various usage and statistics messages

Test scheduling

```
# Sample
#
[test A]

[test B]
requires=A

[test C]
requires=B

[test D]

[test E]
requires=D

[test F]
requires=C E

[test G]

[test H]

[suite my_suite]
testlist=A B C D E F G H
```

How the tests will be run:



Dependent tests create a serial workflow within the suite

- Basic acyclic dependency graphs supported

Independent tests run in parallel

How to add a new integration test

- The basic steps
  - Write a command, program, or a fcl file for 'lar' that returns 0 when passed
    - Any non-zero value if the test fails for any reason
  - Add a new test section to the appropriate ci_tests.cfg file
  - Add the newly added test to a test suite

Will discuss running the test in a bit...

Unit test configuration

- Defined in CMakeLists.txt files
  - Can be defined in any CMakeLists.txt file
  - By convention, prefer <repo>/test/<package name>/CMakeLists.txt
    - Tests apply to code in <repo>/<package name>

- Configuring tests
  - cet_test macro in cetbuildtools/Modules/CetTest.cmake
  - Basic usage: cet_test(target [<options>] [<args>] [<data-files])
    - "Target" = test name reported to the system
      - Does not need to be the command executed
  - Options, arguments well documented in the source code

    https://cdcvs.fnal.gov/redmine/projects/cetbuildtools/repository/revisions/master/entry/Modules/CetTest.cmake
  -

Unit test configuration

- Examples from LArSoft and experiment repositories
  - From larcore/test/SimpleTypesAndConstants/CMakeLists.txt

```
cet_test( testPhysicalConstants )
```

    - Builds larcore/test/SimpleTypesAndConstants/testPhysicalConstants, then runs it
  - From uboonecode/test/Geometry/CMakeLists.txt

```
cet_test( geometry_microboone HANDBUILT
        DATAFILES test_geometry_interators.fcl
        TEST_EXEC lar
        TEST_ARGS –rethrow-all –config ./test_geometry_iterators_uboone.fcl
)
```

    - The “HANDBUILT” option prevents cet_test from attempting to build “geoemtry_microboone” (just a name)
    - Runs “lar --rethrow-all --config ./test_geometry_uboone.fcl”
      - Most existing “unit tests” are of this form. Are these really “unit tests”?

Unit test configuration

- Some important keywords
  - Options
    - HANDBUILT: do not build the target (which is typically just a name anyway)
    - NO_AUTO: do not add to “auto test” list
  - Arguments
    - TEST_EXEC
      - The executable to run if different from the “target” name. (Must also specify HANDBUILT)
    - TEST_ARGS
      - Arguments passed to the test to be run
    - REF
      - Standard output captured and compared to the specified reference file
    - OPTIONAL_GROUPS
      - Assigns the test to the listed “test groups”
      - “Test groups” are equivalent to, but distinct from “test suites”
      - Can be run by setting -DCET_TEST_GROUPS <group name> on cmake command line

Running tests

- Two methods to run integration tests
  - Trigger test jobs on the central build service
    - All code must be pushed to central repository first
    - By default, “git push origin develop” triggers a build
    - Can trigger manually with a script that allows code on non-develop branches to be tested
      - Can specify which branch to use repository-by-repository
    - View results on reporting web page:
      - http://lar-ci-history.fnal.gov:8080/LarCI/app/view_builds/index
  - Running tests locally
    See https://cdcvs.fnal.gov/redmine/projects/lar-ci/wiki/Test_runner_introduction for details
    - In a working directory

```
setup <software version to be tested>
setup lar_ci
test_runner <test1> <test2> ...
```

    - Summary of results printed to stdout

Work session

LArSoft test strategy

- Questions about current test suite
  - Are the tests fast enough?
    - Takes about 2 hours to run the suite on Linux build slaves
    - Too slow for most people to pay attention to for pre or post commit testing
  - Is the rate of false positives manageable?
    - For example, almost all recent test jobs have “failed” status
    - Typically the problem is that the test breaks, not the code being tested
  - Test coverage?
    - Are we adequately checking all important code, stages?
      - Even if algorithms run, do we always check that the appropriate output is there?
    - Are we testing each of 35T, uBooNE, LArIAT, SBND, ArgoNeuT sufficiently well?
  - Do we have tests well matched to the questions at hand?
    - Every-commit questions are not the same ones we need to check new releases
    - But we have only one test suite

LArSoft test strategy

- Propose a tiered testing strategy (for unit and integration tests)
  - On each individual integration (i.e on each push to central repository)
    - Completeness less important than duration.
      - No more than 10-15 minutes?
    - Focus on identifying “major” problems only.
      - Build failures, detector interoperability, crashes, missing functionality, can't read old data, can't read new data
    - Highly managed suite of tests
  - Once daily / nightly tests
    - Focus on finding more subtle problems
    - Needs to be more complete, but time still matters.
  - Before / after every release (for at least production releases)
    - Completeness is more important that speed
    - Possibly physics validation-like tests?
    - Less managed suite of tests
- Low rate of false positives is critical in all cases

What we have now

Current unit tests

- geometry_iterator_test
- Wire_test
- donothing_lbne35t
- donothing_simul_lbne35t
- build_oplib_lbne35t
- geometry_microboone
- optical_digi_lbne35t
- SurfYZTest
- gensingle
- timingreference_test
- KalmanFilterTest
- testPhysicalConstants
- BulkAllocator_test
- SurfXYZTest
- geometry_iterator_uboone_test
- LATest
- geometry_test
- sparse_vector_test
- geometry_iterator_dunefd_test
- optical_sim_lbne35t
- geometry_lbne35t
- PropTest
- SimpleFits_test
- geometry_iterator_loop_uboone_test
- optical_reco_lbne35t
- geometry_dune35t_test
- prodsingle_uboone_max2
- geometry_iterator_loop_dunefd_test
- GausFitCache_test
- geometry_lbnefd
- geometry_iterator_loop_test
- donothing_simul_lbnefd
- donothing_lbnefd
- TrackTest
- geometry_uboone_test
- RawDigit_test
- NestedIterator_test
- CountersMap_test
- geometry_iterator_loop_dune35t_test
- StatCollector_test
- OpFlashAlg_test
- geometry_iterator
- Cluster_test
- geo_types_test
- geometry_iterator_dune35t_test
- geometry
- HitAnaAlg_test
- GeneratedEventTimestamp_test2
- AlgoThreshold_test
- geometry_dunefd_test
- donothing
- FastMatrixMath_test
- Hit_test
- Dereference_test
- GeneratedEventTimestamp_test1_1
- raw_test
- test_fcl.sh
- GeneratedEventTimestamp_test1_2

Total time: almost 30 min!!
About 60 tests
Vast majority are 'lar' jobs

What we have now

Current integration tests

Total time: about 1 hour
(excludes the two tests that did not run)
15 tests

- lar_ci_openold_detsim_lbnecode
- lar_ci_openold_detsim_uboonecode
- lar_ci_hitana_g4_uboonecode
- lar_ci_prodsingle_lbnecode
- lar_ci_hitana_tinyana_new_uboonecode
- lar_ci_hitana_detsim_uboonecode
- lar_ci_prodgenie_uboonecode
- lar_ci_hitana_tinyana_canonical_uboonecode
- lar_ci_hitana_prod_uboonecode
- lar_ci_histocomp_uboonecode
- lar_ci_hitana_reco2D_uboonecode
- lar_ci_prodsingle_uboonecode
- lar_ci_openold_detsim3d_uboonecode
- lar_ci_openold_detsim2d_uboonecode
- lar_ci_openold_reco_lbnecode

The task at hand

- Define test use cases, create a suite for each
- Set target run times for each use case / test suite
- Define specific tests to be run in each
  - What needs to be tested?
  - How it should be checked in a way that avoids / minimize false positives?
  - Does it run fast enough for the use case?

Backup

Continuous integration

- What is continuous integration (CI)?
  - A software development practice in which team members integrate their work into the main development branch frequently, usually at least daily.
    Source: an amalgam of quotes from a search on “continuous integration definition”
  - Each integration is tested by an automated build and test system designed to detect integration errors as quickly as possible
  -
  - At odds with our “managed integration” approach of using Coordination Meetings to decide what gets merged and when?
    - No... explain
- Benefits
  - Low-cost method that catches problems quickly
  - Maintain more stable main-line development branch
  - Can create a release with known properties quickly
  - Low cost

CI system components

- Central Build Service
  https://buildmaster.fnal.gov/
  - Server plus distributed build slave nodes
    - One or more slave nodes per OS to be supported
    - Slave nodes can be off-site
  - Jenkins CI application
    - Server-side application
      - The application runs "test jobs", aka "builds", in response to http-based triggers
      - Arguments in trigger specify run-time configuration options
        - *e.g., the release, the test suite to run, the branches to use, etc*
        - *More on triggers later*
- Results database / reporting web server
  - Used to store test job status, results of individual tests.
  - Web server hosts monitoring and reporting GUI

CI system components

- Client-side driver software

  (The lar-ci product)

  - Runs on the build slaves
  - Processes a workflow with a number of steps, or "phases"
    - The number of phases and the actions in each is configurable by CI admins
    - The specific workflow to run is specified as a trigger argument
  - The default workflow
    - 5 phases: checkout, build, unit test, install, integration test
    - Non-zero exit status at any step terminates workflow.

Two test phases
in the default workflow

CI system components

- Test monitoring and results viewer
  (The lar-ci-reporting product)
  - Extracts and presents test status, results from the database
    - http://lar-ci-history.fnal.gov:8080/LarCI/app/view_builds/index
  - Increasing levels of detail exposed via drill-down links on each page
    - Top level summary, which leads to...
    - Test phase summary, which leads to...
    - Single test result page, which leads to...
    - Low-level output and results
  - Plots of test times at the top of some pages
    - Test phase summary: elapsed time required for the phase over time
    - Single test result page: elapsed time required for that test over time, maximum memory usage for the test over time