Testing Strategies – Condensed Version

David M. Rogers (he/him)
Oak Ridge National Laboratory

Better Scientific Software tutorial @ ISC23

Contributors: David E. Bernholdt (ORNL), Anshu Dubey (ANL), Rinku Gupta (ANL), Mark C. Miller (LLNL), Greg Watson (ORNL), David M. Rogers (ORNL)

License, Citation and Acknowledgements

License and Citation

- **The requested citation the overall tutorial is:** Anshu Dubey and David M. Rogers, Better Scientific Software tutorial, in ISC High Performance, Hamburg, Germany and online, 2023. DOI: 10.6084/m9.figshare.22790762.
- Individual modules may be cited as *Speaker, Module Title*, in *Tutorial Title*, …

Acknowledgements

- This work was supported by the U.S. Department of Energy Office of Science, Office of Advanced Scientific Computing Research (ASCR), and by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration.
- This work was performed in part at the Argonne National Laboratory, which is managed by UChicago Argonne, LLC for the U.S. Department of Energy under Contract No. DE-AC02-06CH11357.
- This work was performed in part at the Lawrence Livermore National Laboratory, which is managed by Lawrence Livermore National Security, LLC for the U.S. Department of Energy under Contract No. DE-AC52-07NA27344.
- This work was performed in part at the Los Alamos National Laboratory, which is managed by Triad National Security, LLC for the U.S. Department of Energy under Contract No.89233218CNA000001
- This work was performed in part at the Oak Ridge National Laboratory, which is managed by UT-Battelle, LLC for the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.
- This work was performed in part at Sandia National Laboratories. Sandia National Laboratories is a multi-mission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC., a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

IDEAS productivity

ECP EXASCALE COMPUTING PROJECT

Hypothetical

Matthew Norman and Jeffrey Larkin. A Holistic Algorithmic Approach to Improving Accuracy, Robustness, and Computational Efficiency for Atmospheric Dynamics. SIAM J. SCI. COMPUT. Vol. 42, No. 5, pp. B1302-B1327 (2020). DOI:10.1137/19M128435X

We want to test out a new time-integration method for our atmospheric transport equations. It's still finite-volume, but uses a different way to limit oscillations.

How does it perform on 1D transport?

It looks like it preserves discontinuity shapes better. We'd like to achieve higher accuracy at lower time-to solution. How can we test that?

We'll need to compare the old and new integration outputs, and change the grid resolutions. We'll need some timings too.

What is Testing

Whenever you write a code you are doing it

- When you compile it, you are testing for defects in syntax
- When you run it for the first time you are testing for correctness
- When you add any code and run it again, you are testing it again
- When you break down your development into smaller chunks you test each chunk, then you combine the chunks, and you test again.

What is Testing

Whenever you write a code you are doing it

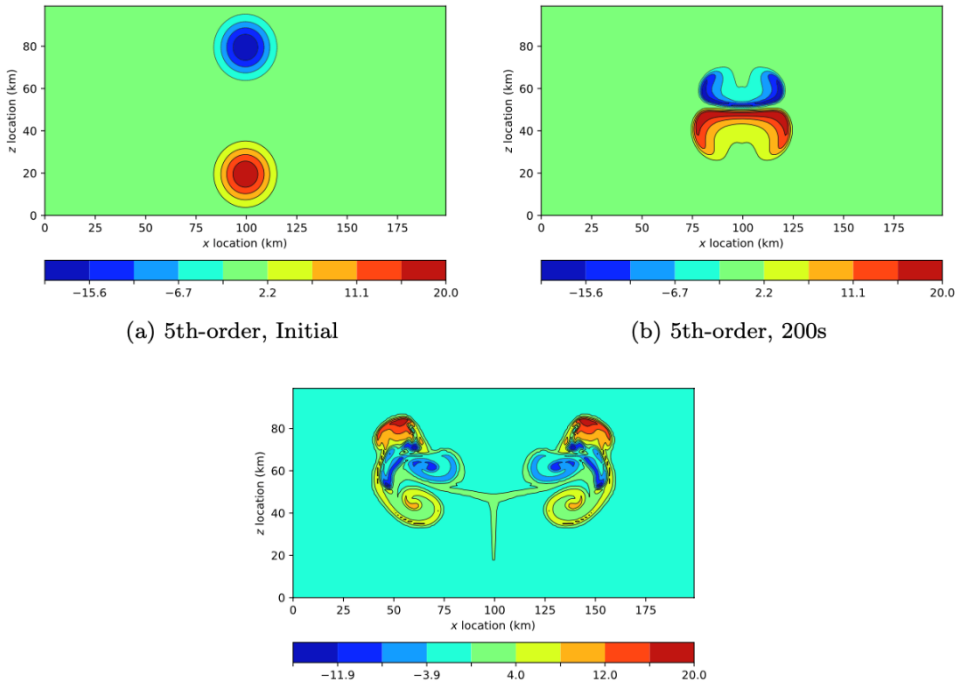
- When you compile it, you are testing for defects in syntax
- When you run it for the first time you are testing for correctness
- When you add any code and run it again, you are testing it again
- When you break down your development into smaller chunks you test each chunk, then you combine the chunks, and you test again.

Testing is an integral part of code development

Hypothetical

Matthew Norman and Jeffrey Larkin. A Holistic Algorithmic Approach to Improving Accuracy, Robustness, and Computational Efficiency for Atmospheric Dynamics. SIAM J. SCI. COMPUT. Vol. 42, No. 5, pp. B1302-B1327 (2020). DOI:10.1137/19M128435X

Here's a 2D result from the new integrator method.



(a) 5th-order, Initial (b) 5th-order, 200s

(c) 5th-order, 400s

FIG. 2. *Contours of potential temperature for colliding warm and cold thermals using ADER and WENO with a CFL value of* 0.9 *using* 200 x 100 *cells.*

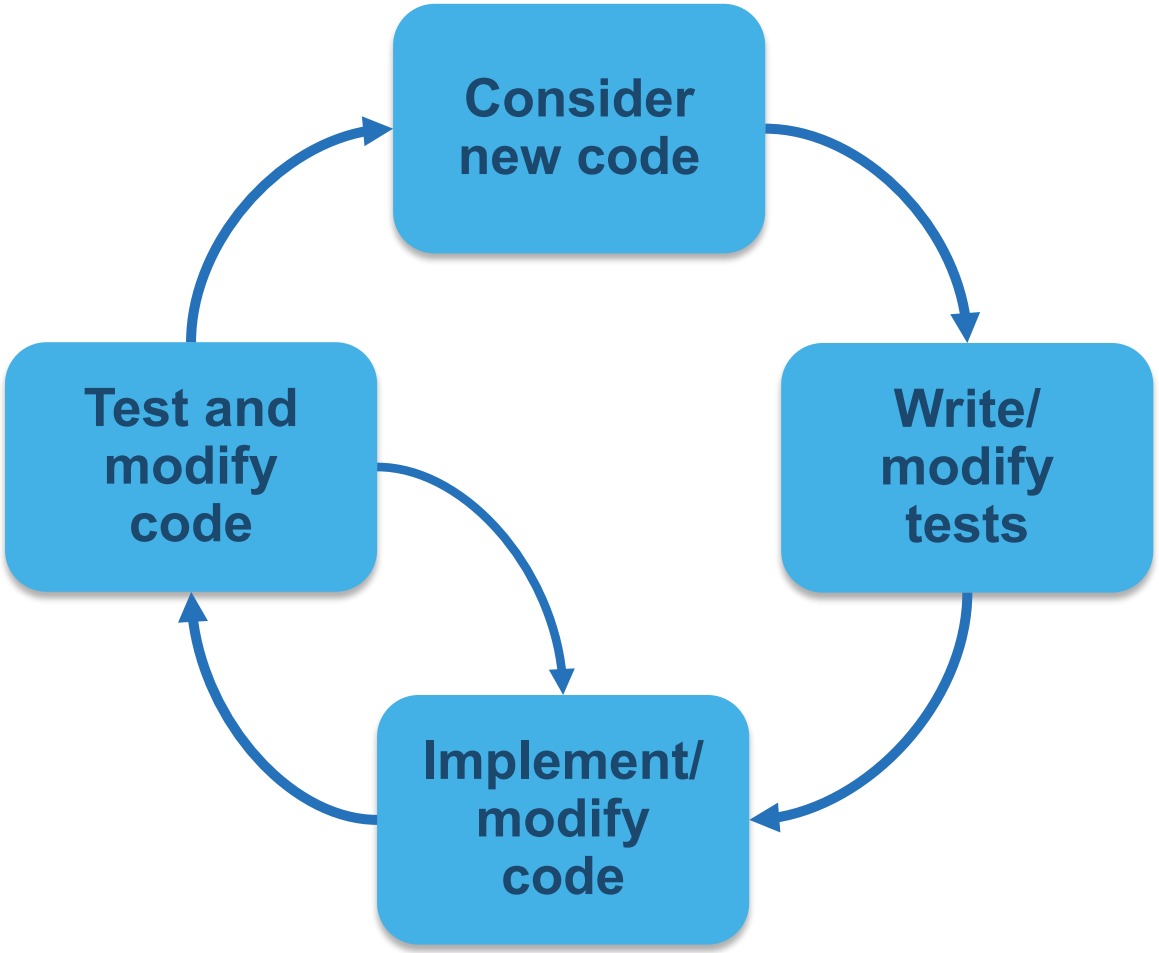
Great, let's get the rest of the code switched over.

This will be faster and safer if we have the right tests in-place.

IDEAS productivity

ECP EXASCALE COMPUTING PROJECT

Test/Doc Driven Development



- Documented specifications and requirements of the code
- Ensures that thought is given to what it means for the program to be correct, rather than just what the program should do
- More efficient development cycle
- Much less debugging
- Requires:
  - Care in writing tests
  - Frequent running of tests
  - Wide adoption by development team

How to build your test suite?

- Two “levels”
  - Automated / scheduled testing
    - May be long running
    - Provide comprehensive coverage
  - Continuous integration
    - Quick diagnosis of error
- A mix of different granularities works well
  - Unit tests for isolating component or sub-component level faults
  - Integration tests with simple to complex configuration and system level
  - Restart tests

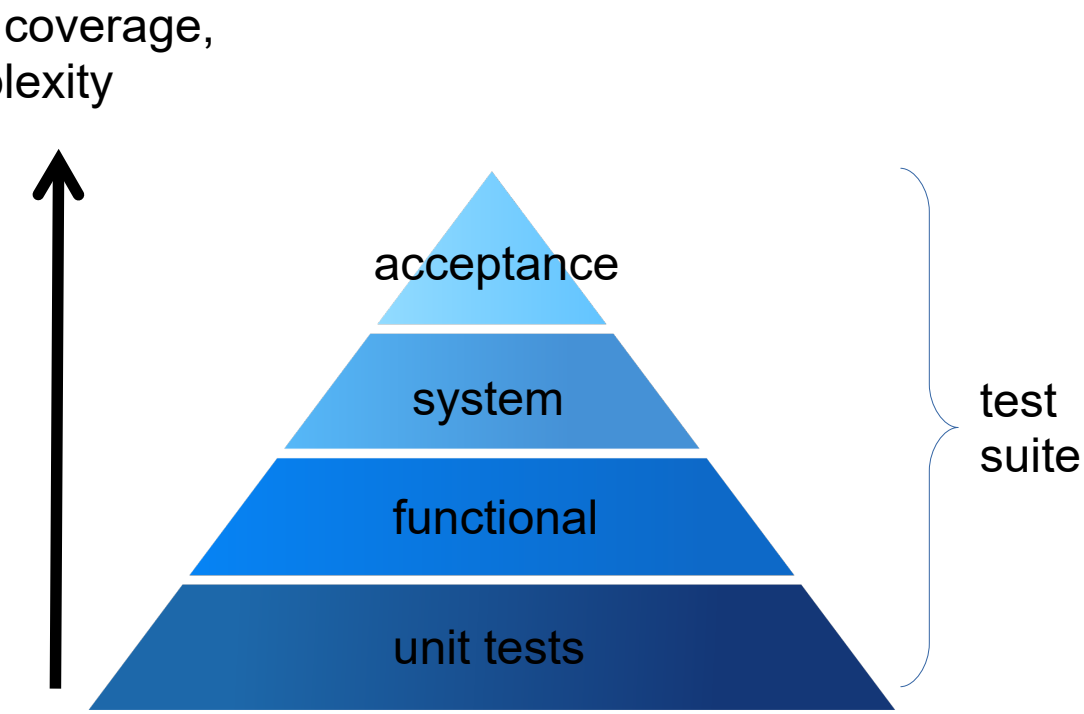
- Rules of thumb
  - Simple
  - Enable quick pin-pointing

Useful resources https://ideas-productivity.org/resources/howtos/

Types of Tests

Well known tests for enterprise software

- Unit tests – verify a single function, extremely quick to run
- Regression tests – verify that there is no degradation in code capabilities
- Integration tests – verify functions working together
- System tests – verify functionality of the entire software
- Acceptance tests – verify that the client needs are met



Types of Tests

Additional types of tests needed for research software

- Composite unit tests – are tests for specific functionalities and/or capabilities
- Granular tests – are integration tests at various granularities verifying correct behavior of interoperating functional units
- Restart tests – verify that a run can restart transparently from a checkpointed state
- Performance tests – apply to high-performance computing codes, verify that there is no performance loss

Classes of Tests

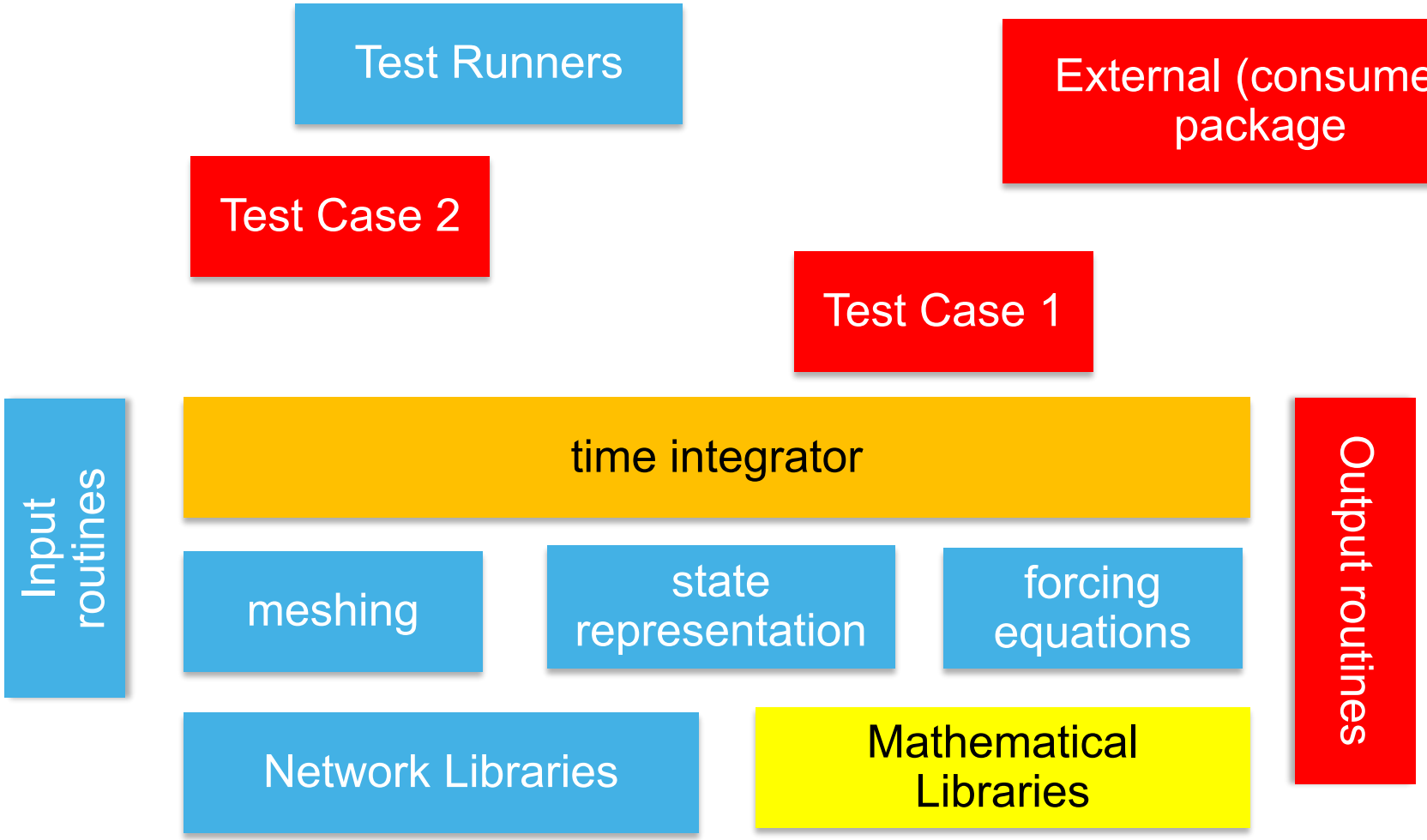
- Open box testing – when you know the internals and can modify the code you are testing
  - Likely to be the code you and your collaborators are developing
  - You can insert assertions
  - You can insert code snippets that make testing easier

- Closed box testing – when you do not know the internals of the code being tested, and cannot modify the code
  - Third party software or legacy code
  - The only means of verification available is reasoning about output to be obtained from supplied input

Hypothetical

Matthew Norman and Jeffrey Larkin. A Holistic Algorithmic Approach to Improving Accuracy, Robustness, and Computational Efficiency for Atmospheric Dynamics. SIAM J. SCI. COMPUT. Vol. 42, No. 5, pp. B1302-B1327 (2020). DOI:10.1137/19M128435X

downstreams



upstreams

A partially transformed code state

IDEAS productivity

ECP EXASCALE COMPUTING PROJECT

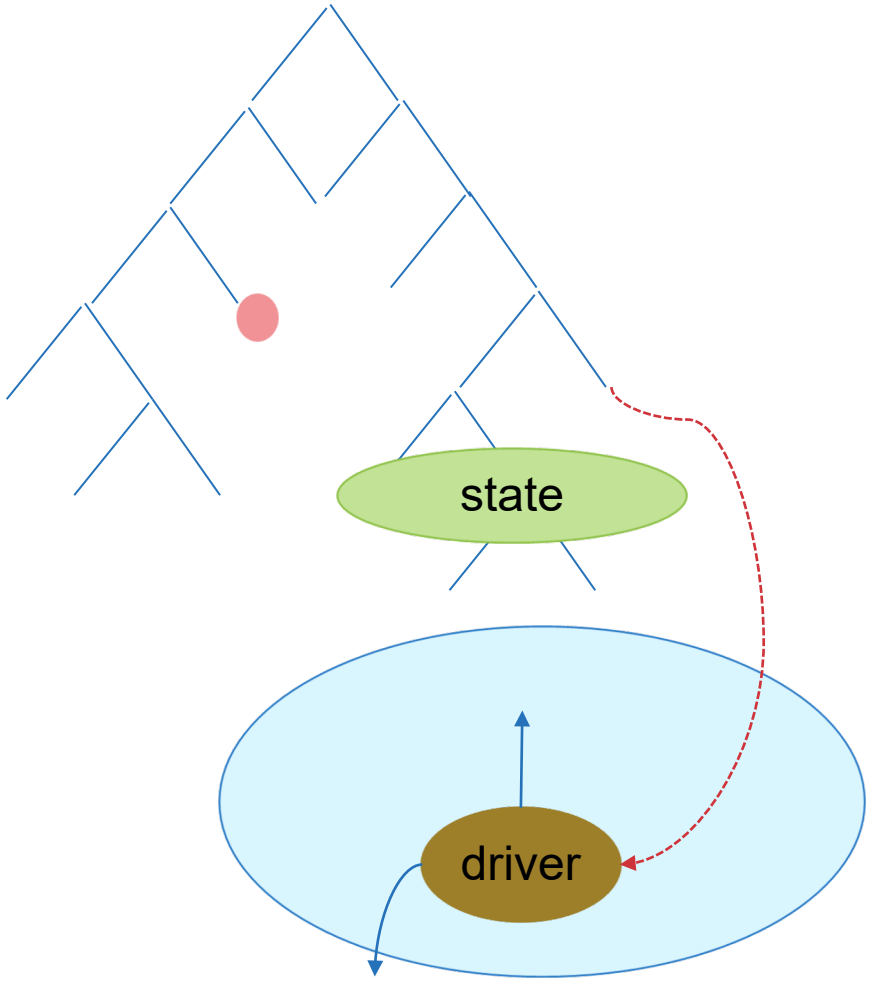
Additional Notes: Good Testing Practices

- Verify Code coverage
- Must have consistent policy on dealing with failed tests
  - Issue tracking
    - How quickly does it need to be fixed?
    - Who is responsible for fixing it?
- Someone should be watching the test suite
- When refactoring or adding new features, run a regression suite before check in
  - Add new regression tests or modify existing ones for the new features
- Code review before releasing test suite is useful
  - Another person may spot issues you didn't
  - Incredibly cost-effective

Mixed Open/Closed Box Testing For a Legacy Code

There may not be existing tests

- Isolate a small area of the code
- Dump a useful state snapshot
- Build a test driver
  - Start with only the files in the area
  - Link in dependencies
    - Copy if any customizations needed
- Read in the state snapshot
- Restart from the saved state
- Verify correctness
  - Always inject errors to verify that the test is working



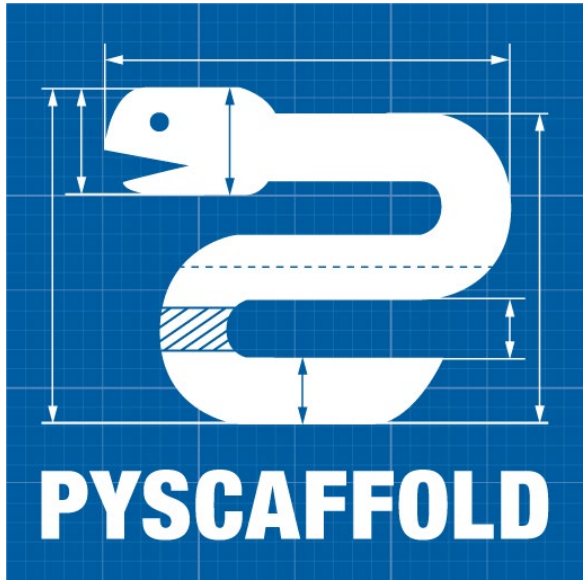
How to build your test suite?

- Start from a good testing framework
- Integrate into your development workflow
- Adding more tests: Does this test add value?
  - Simple
  - Enable quick pin-pointing
  - Helpful for onboarding new developers.
  - Useful for scientific reviewers of this package and its results.

Python Example

```
$ pip install pyscaffold
$ pip install tox
$ putup autoQCT
$ cd autoQCT # tests in tests/ subdir.
$ tox
```

```
default run-test: commands[0] | pytest
========================= test session starts ==========================
platform darwin -- Python 3.9.0, pytest-6.2.2, py-1.10.0, pluggy-0.13.1 -- plugins: cov-2.11.1
collected 2 items

tests/test_skeleton.py::test_fib PASSED                  [ 50%]
tests/test_skeleton.py::test_main PASSED                 [100%]

---------- coverage: platform darwin, python 3.9.0-final-0 -----------
Name                    Stmts   Miss Branch BrPart  Cover   Missing
-------------------------------------------------------------------
src/autoqct/__init__.py       6      0      0      0   100%
src/autoqct/skeleton.py      32      1      2      0    97%   135
-------------------------------------------------------------------
TOTAL                        38      1      2      0    98%

========================== 2 passed in 0.07s ===========================
  default: commands succeeded
  congratulations :)
```



pyscaffold.org

CMake Example

```
$ cat >CMakeLists.txt <<.
cmake_minimum_required(VERSION 3.8)
project( blank )
set(CMAKE_CXX_STANDARD 11)
set(CMAKE_CXX_STANDARD_REQUIRED ON)
include(blt/SetupBLT.cmake)
.
$ git clone https://github.com/LLNL/blt/
$ mkdir build && cd build
$ cmake ..
$ make –j
```

llnl-blt.readthedocs.io

```
...
[100%] Linking CXX executable ../../../tests/blt_gtest_smoke
[100%] Built target blt_gtest_smoke
$ make test
Running tests...
Test project /Users/99r/work/autoQCT/blank_project/build
    Start 1: blt_gtest_smoke
1/1 Test #1: blt_gtest_smoke .....   Passed    0.46 sec

100% tests passed, 0 tests failed out of 1

Total Test time (real) =   0.46 sec
```

Checking Code Coverage - Example

https://github.com/bssw-tutorial/hello-numerical-world

- Example of heat equation
  - Add -coverage as shown below to Makefile
  - Run ./heat runame="ftcs_results"
  - Run gcov heat.C
  - Examine heat.C.gcov

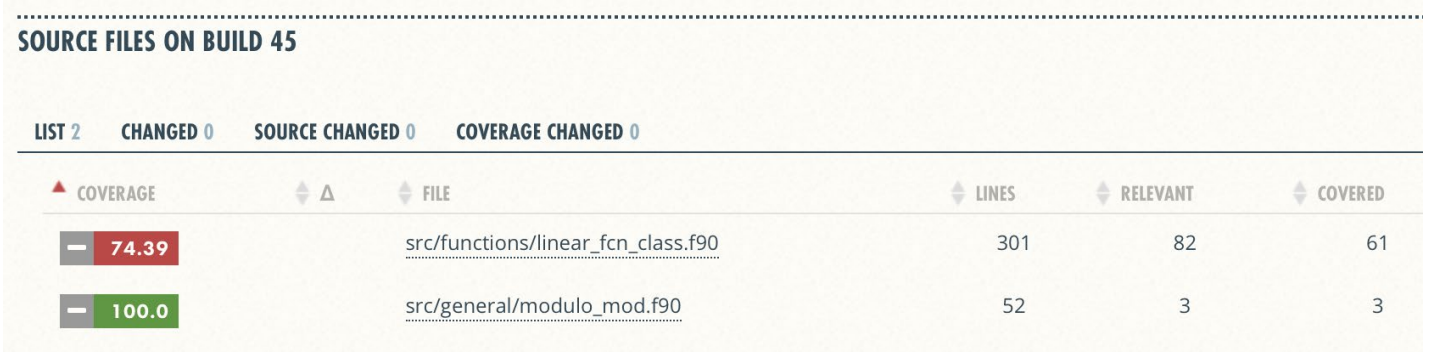
```
HDR = Double.H
SRC = heat.C utils.C args.C exact.C ftcs.C upwind15.C crankn.C
OBJ = $(SRC:.C=.o)
GCOV = $(SRC:.C=.C.gcov) $(SRC:.C=.gcda) $(SRC:.C=.gcno) $(HDR:.
H=.H.gcov)
EXE = heat

# Implicit rule for object files
%.o : %.C
        $(CXX) -c -coverage $(CXXFLAGS) $(CPPFLAGS) $< -o $@

# Linking the final heat app
heat: $(OBJ)
        $(CXX) -coverage -o heat $(OBJ) $(LDFLAGS) -lm
```

- A dash indicates non-executable line
- A number indicated the times the line was called
- ##### indicates line wasn't exercised

```
    -:  143:static bool
  500:  144:update_solution()
    -:  145:{
  500:  146:    if (!strcmp(alg, "ftcs"))
  500:  147:        return update_solution_ftcs(Nx, curr, last, alpha, dx, dt, bc0, bc1);
#####:  148:    else if (!strcmp(alg, "upwind15"))
#####:  149:        return update_solution_upwind15(Nx, curr, last, alpha, dx, dt, bc0, bc1);
#####:  150:    else if (!strcmp(alg, "crankn"))
#####:  151:        return update_solution_crankn(Nx, curr, last, cn_Amat, bc0, bc1);
#####:  152:    return false;
  500:  153:}
    -:  154:
    -:  155:static Double
  500:  156:update_output_files(int ti)
    -:  157:{
  500:  158:    Double change;
    -:  159:
  500:  160:    if (ti>0 && save)
    -:  161:    {
#####:  162:        compute_exact_solution(Nx, exact, dx, ic, alpha, ti*dt, bc0, bc1);
#####:  163:        if (savi && ti%savi==0)
#####:  164:            write_array(ti, Nx, dx, exact);
#####:  165:    }
```

Graphical View of Gcov Output and Tutorials for Code Coverage

Coverage Summary

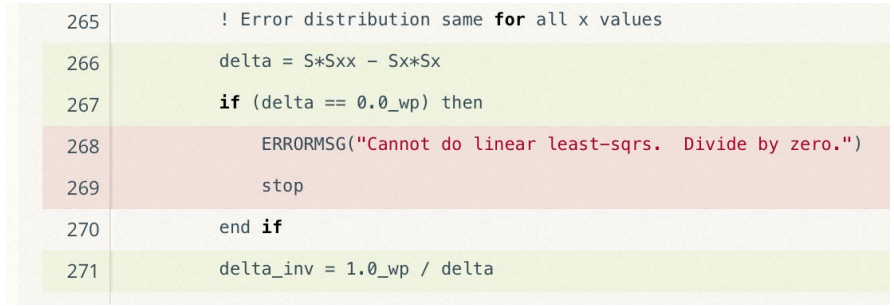


SOURCE FILES ON BUILD 45

LIST 2 CHANGED 0 SOURCE CHANGED 0 COVERAGE CHANGED 0

COVERAGE	Δ	FILE	LINES	RELEVANT	COVERED
74.39		src/functions/linear_fcn_class.f90	301	82	61
100.0		src/general/modulo_mod.f90	52	3	3

Line-by-line details



```
265         ! Error distribution same for all x values
266         delta = S*Sxx - Sx*Sx
267         if (delta == 0.0_wp) then
268             ERRORMSG("Cannot do linear least-sqrs.  Divide by zero.")
269             stop
270         end if
271         delta_inv = 1.0_wp / delta
```

Online tutorial - https://github.com/amklinv/morpheus

Other example - https://github.com/jrdoneal/infrastructure

Going Further

- C, C++, Fortran
  - Running and Reporting Tests: ctest / cdash
  - Code Coverage: gcov / lcov/ hosted codecov.io/coveralls (C, C++, Fortran, ...)
  - Static Analysis: clang-tidy (only C, C++)
  - LLVM UndefinedBehaviorSanitizer, profile-guided optimizations, and more
- Python
  - Running and Reporting Tests: pytest / unittest / nose
  - Code Coverage: pytest-cov
  - Static Source Code Analysis: mypy, pylint / flake8

Building Test-suite

First line of defense – code coverage tools

- Code coverage tools necessary but not sufficient
- Do not give any information about interoperability

	Hydro	EOS	Gravity	Burn	Particles
AMR	CL	CL		CL	CL
UG	SV	SV			SV
Multigrid	WD	WD	WD	WD	
FFT			PT		

- Map your tests and examples – what do they do?
- Follow the order
  - All unit tests – including full module tests (e.g., CL)
  - Tests sensitive to perturbations (e.g., SV)
  - Most stringent tests for solvers (e.g., WD, PT)
  - Least complex test to cover remaining spots (**Aha!**)

Summary: Rules of Thumb

- Test your tests!
  - Make sure tests fail when they're supposed to!
- Add "regression tests"
  - Ensure that old bugs aren't reappearing
- Test regularly
  - Critical when teams are adding code regularly
  - To identify and document where changes to the underlying platform change code behavior/results
- Automate regular testing
  - Inculcate the discipline of monitoring the outcome of regular testing
- Test your assumptions
  - Exercise third-party dependencies, prove they work for your case
  - Add tests when joining a new code
  - Stash tests used to diagnose issues
- Physics/math-based strategies
  - Conserved quantities, symmetries, synthetic operators
  - Use "specs" to eliminate dependence on bitwise reproducibility

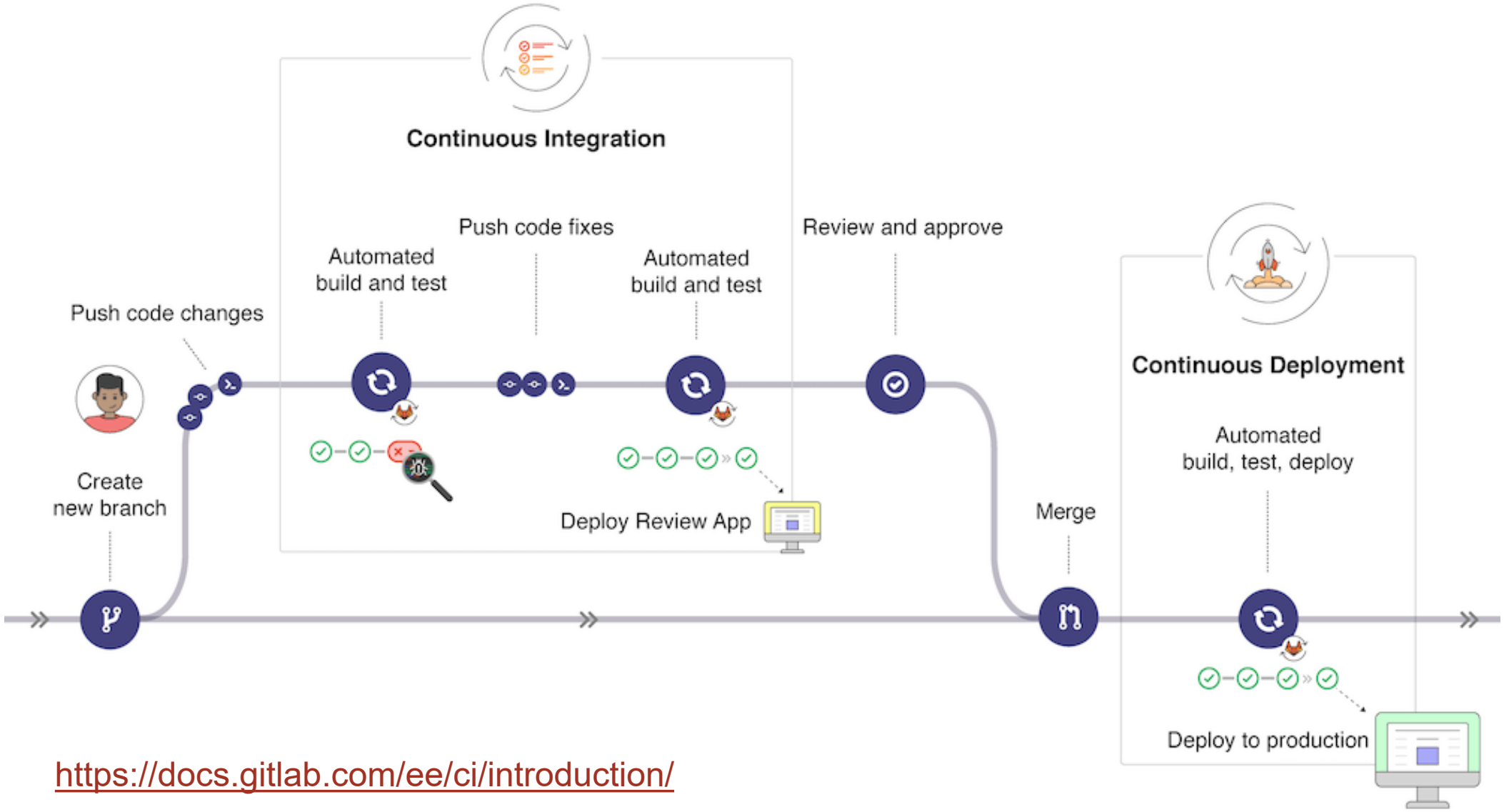
Testing Takeaways

- A testing strategy is essential for producing reliable trustworthy software
  - Invest the time needed to thoroughly test your software at all levels
  - Use automation whenever possible
- Different challenges are associated with exploratory, legacy, and composable codes
  - Adapt your strategy to fit your situation.
  - Eventually you will want to be able to verify all components in a code release.
- Don’t get distracted by all the technologies out there – focus on exercising your code.
  - Scaffolding projects can help with mechanics.

Questions?

- Testing Types
  - scheduled vs. continuous
  - unit vs. module vs. integration
  - restart, performance, regression
  - system and acceptance testing
- Testing Classes
  - open/closed box: ability to modify codebase
  - inferring vs. checking behavior (bottom-up vs. top-down)
- Testing Practices
  - coverage, resolution policies, release checklists
- Testing areas
  - proving: assumptions, invariants, external behavior, interfaces, types and specifications
- Definition of Done
  - coverage and test matrices
  - level of confidence vs. concept / code maturity

What is Continuous Integration (CI)?



https://docs.gitlab.com/ee/ci/introduction/

CI Components

- Testing
  - Focused, critical functionality (infrastructure), independent, orthogonal, complete, …
  - Existing test suites often require re-design/refactoring for CI
- Integration
  - Changes across key branches merged & tested to ensure the “whole” still works
    - Integration can take place at multiple levels
      - Individual project
      - Spack
      - E4S
  - Develop, develop, develop, merge, merge, merge, test, test, test…NO!
  - Develop, test, merge, develop, test, merge, develop, test, merge…YES!
- Continuous
  - Changes tested every commit and/or pull-request (like auto-correct)

- CI generally implies a lot of automation

Examples…

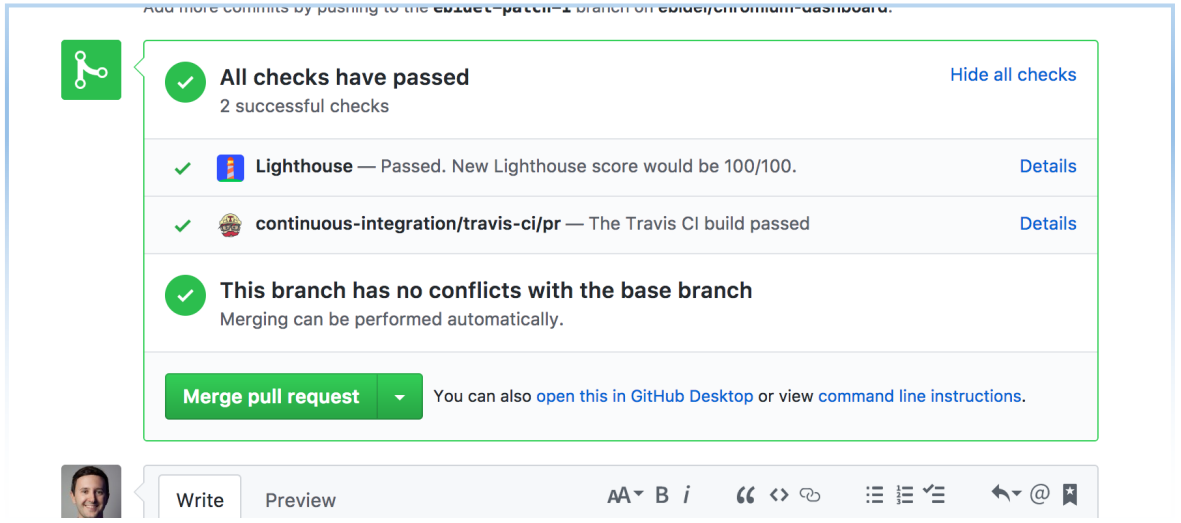
Automated Nightly Testing Dashboard
Lives "next to" your development work

Results of VisIt Regression Test (pascal,trunk,serial)

Test suite run started at 2020:07:09:22:49:46.
(Click on table header to sort)

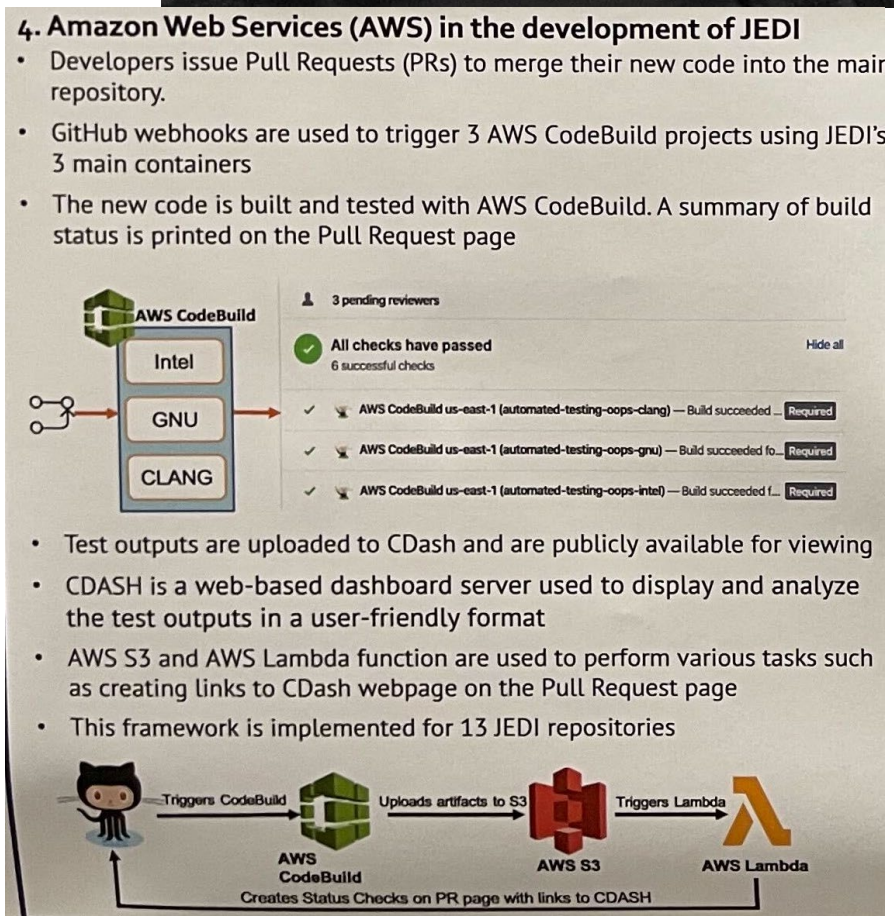
Index	Category	Test File	Status	Runtime (sec)
243	rendering	ospray.py	Unacceptable	5.0
273	simulation	batch.py	Unacceptable	38.0
24	databases	chgcar.py	Succeeded With Skips	11.0
32	databases	exodus.py	Succeeded With Skips	14.0
66	databases	silo.py	Succeeded With Skips	50.0
67	databases	silo_altdriver.py	Succeeded With Skips	87.0
75	databases	xdmf.py	Succeeded With Skips	14.0
109	hybrid	merge_tree.py	Succeeded With Skips	11.0
136	meshtype	emptydomains.py	Succeeded With Skips	7.0
256	rendering	view.py	Succeeded With Skips	17.0
275	simulation	curve.py	Succeeded With Skips	8.0
281	simulation	life.py	Succeeded With Skips	8.0
296	simulation	zerocopy.py	Succeeded With Skips	32.0
0	databases	ANALYZE.py	Succeeded	10.0
1	databases	ANSYS.py	Succeeded	9.0
2	databases	CGNS.py	Succeeded	11.0
3	databases	Cale.py	Succeeded	6.0
4	databases	Chombo.py	Succeeded	7.0
5	databases	EnSight.py	Succeeded	9.0
6	databases	FITS.py	Succeeded	8.0
7	databases	Fluent.py	Succeeded	7.0
8	databases	GDAL.py	Succeeded	20.0
9	databases	NASTRAN.py	Succeeded	15.0

CI Testing
Lives embedded in your development work



Examples... Joint Center for Satellite Data Assimilation (JEDI)

Posted to slack/spack#appreciation from the AGU Fall Meeting, Chicago by Evan Bollig



The JEDI Singularity and CharlieCloud containers are better supported and provide a more familiar working environment for most users and developers. The recommended practice is therefore to first establish a linux environment on your laptop or PC using a virtual machine provider like Vagrant and then to run the JEDI Singularity or Charliecloud container there.

See "Inside JEDI" section for lots of useful recommendations.

https://jointcenterforsatellitedataassimilation-jedi-docs.readthedocs-hosted.com/

Hints from the front lines

github.com/CompFUSE/DCA – be nice to contributors (who create forks)

```
jobs:
  sulfur-cpu:
    if: |
      github.repository_owner == 'CompFUSE' &&
      github.event.issue.pull_request &&
      startsWith(github.event.comment.body, 'Test this please')
```

Build inside a container:

https://docs.docker.com/build/ci/

1. Build inside a container locally
2. Publish your container to docker
3. Reference from a job, e.g.
   "container: node:14.16"

Help with step 1:

$ spack containerize > Dockerfile

https://spack.readthedocs.io/en/latest/containers.html

https://supercontainers.github.io/sc20-tutorial/07.spack/index.html

https://docs.github.com/en/actions/using-jobs/running-jobs-in-a-container

Words of warning from github

Warning: When creating workflows and actions, you should always consider whether your code might execute untrusted input from possible attackers. Certain contexts should be treated as untrusted input, as an attacker could insert their own malicious content. For more information, see "Understanding the risk of script injections."

(github)

Hints from the front lines

github.com/ECP-WarpX/WarpX – combine shell patterns in a function

```
curl -L -o /usr/local/bin/cmake-easyinstall https://git.io/JvLxY
chmod a+x /usr/local/bin/cmake-easyinstall
cmake-easyinstall --prefix=/usr/local                  \
   git+https://github.com/openPMD/openPMD-api.git@0.14.3 \
        -DCMAKE_...
        ...
```

Cristian Adam, 2020:
https://cristianadam.eu/20200113/speeding-up-c-plus-plus-github-actions-using-**ccache**/

```
- name: ccache cache files
   uses: actions/cache@v1.1.0
   path: $HOME/.ccache

- name: build source
  run: |
   sudo apt install -y ccache
   ccache --set-config=max_size=10.0G [WarpX]
   cmake ... -DCMAKE_CXX_COMPILER_LAUNCHER=ccache
```

Hints from the front lines

Configure in Settings / Github Pages

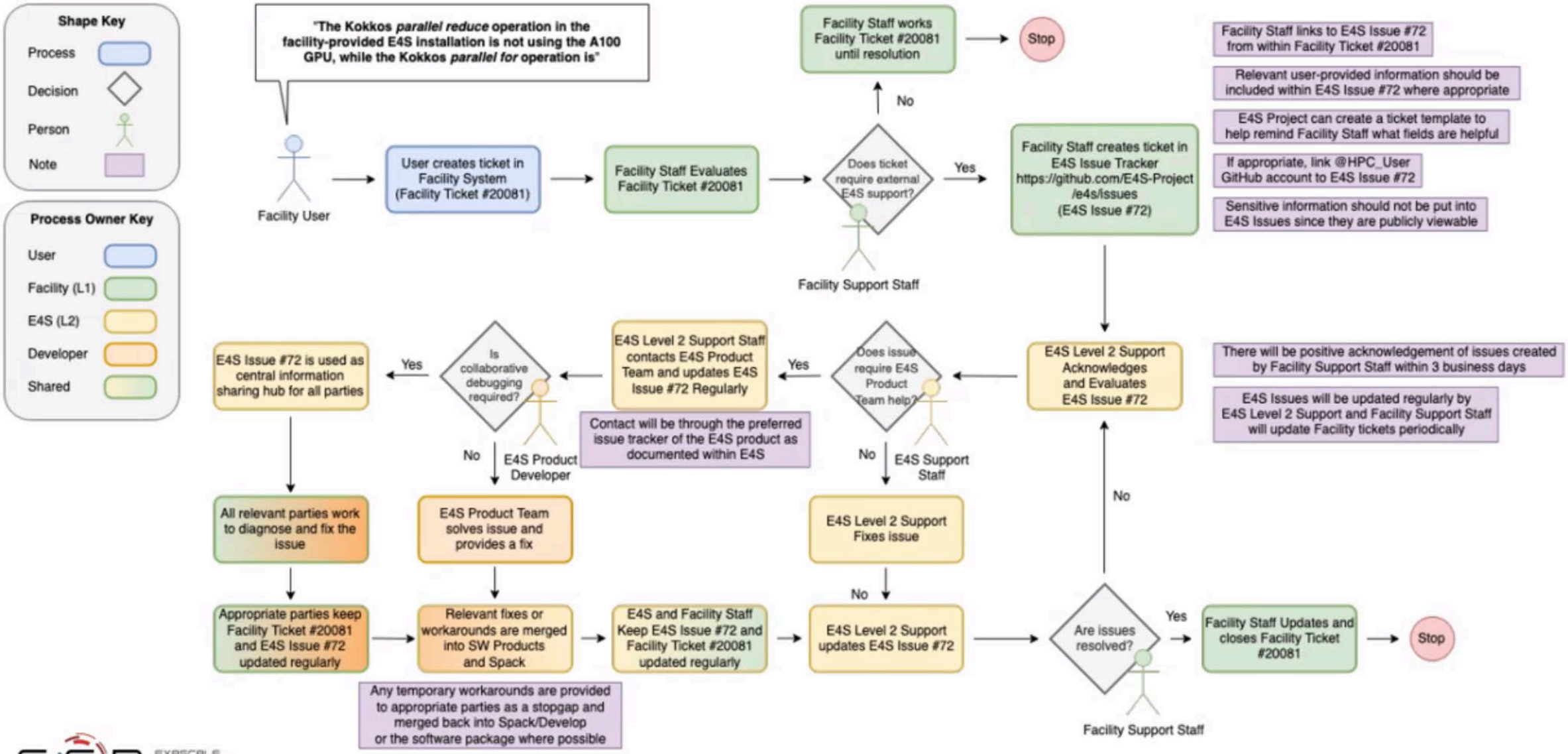
https://docs.gitlab.com/ee/user/project/pages/
https://tomasfarias.dev/posts/sphinx-docs-with-poetry-and-github-pages/

```
jobs:
  build-docs:
    steps:

    ...
    - name: Build documentation
      run: |
        mkdir gh-pages
        touch gh-pages/.nojekyll
        cd docs/
        poetry run sphinx-build -b html . _build
        cp -r _build/* ../gh-pages/

    - name: Deploy documentation
      if: ${{ github.event_name == 'push' }}
      uses: JamesIves/github-pages-deploy-action@4.1.4
      with: { branch: gh-pages folder: gh-pages }
```

Good ideas and idioms from across developer spaces

- Golang interfaces
- C++ object implementation "rule of 3"
  - https://en.cppreference.com/w/cpp/language/rule_of_three
- Python, pytest
  - use assertions and automate calling tests
- C++, googletest / catch2
  - write your tests in a uniform way
- Javascript, https://jestjs.io/
  - use a well-featured test package – error reporting, parallel, cached, etc.
- Ruby, rails, https://guides.rubyonrails.org/testing.html
  - define common verbs/types of questions

E4S / Facility Software Support Model

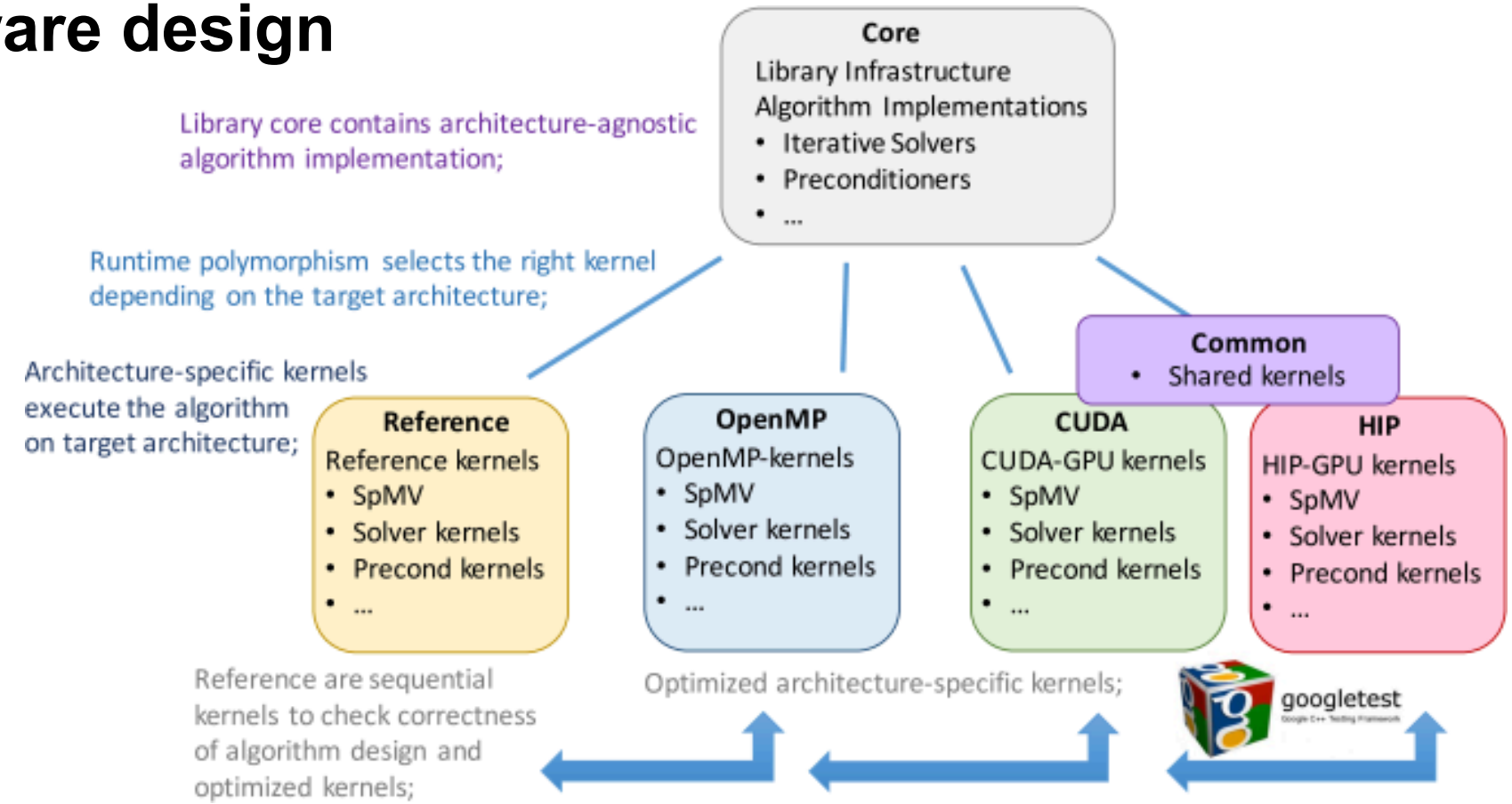


The case for software productivity

In software-driven research, **scientific productivity is strongly coupled to software productivity**. Hence, the scientific output of a research group can be held by challenges such as new computer architectures, advanced algorithms or changing teams of developers. To prevent this productivity collapse, **software development needs to be sustainable and scalable by producing comprehensible, maintainable, and extensible code**. At the same time, **it is essential to release changes ... rapidly to the users**, an ability that usually falls under the term continuous delivery. Last but not least, the scientific standard demands **correctness, credibility and reproducibility of numerical results in published work**. To fulfill these requirements in a challenging environment formed by complex algorithms, performance sensitive codes, the diversity of architectures, and the multidisciplinary of teams, the DCA++ project employs well-proven tools and successful techniques of the software industry [16]. While adopting these methods can require an effort, **we believe that [these methods] represent a substantial factor for a research code to become a long-lived software project**.

DCA++,Hähner, Alvarez, Maier, Solcà, Staar, Summers, Schulthess, Comput. Phys. Commun. 246, 106709 (2020). DOI:10.1016/j.cpc.2019.01.006

The case for software design



Aside from GINKGO being used as a framework for algorithmic research, **its primary intention is to provide a numerical software ecosystem designed for easy adoption by the scientific computing community**. This requires sophisticated **design guidelines and high quality code**.

GINKGO, Anzt, Cojean, Flegar, Göbl, Grützmcher, Nayak, Ribizel, Tsai, Quintana-Ortí, ACM Trans. Math. Software 48, 1-33 (2022).
DOI:10.1145/3480935

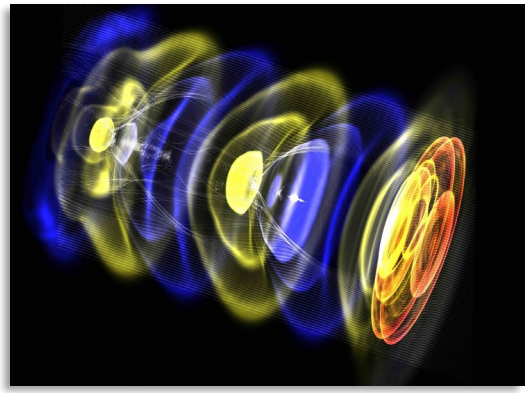
IDEAS productivity

ECP EXASCALE COMPUTING PROJECT

Warp-X

PI: Jean-Luc Vay (LBNL) – DoE Exascale Computing Project (ECP)

A. Almgren, L. D. Amorim, J. Bell, L. Fedeli, L. Ge, K. Gott, D. P. Grote, A. Huebl, R. Jambunathan, R. Lehe, A. Myers, M. Rowan, O. Shapoval, M. Thévenet, J.-L. Vay, H. Vincenti, E. Yang, N. Zaïm, W. Zhang, Y. Zhao, E. Zoni



Developer Training, Maxence Thévenet (LBNL) - 03/05/2020

`WarpX/Regression/` WarpX-tests.ini

`WarpX/Examples/`
- Input files
- Analysis script

`prepare_file_travis.py`
* reformat

Nightly builds on CRD clusters Battra (CPU) and Garuda (GPU)

- Every night
- See https://github.com/ECP-WarpX/regression_testing
- Compare with ref to machine precision
- Published at https://ccse.lbl.gov/pub/RegressionTesting/WarpX/

* Catch everything (and more)

TravisCI tests on every commit on GitHub

- Every time you push on a branch with open PR
- GitHub tells you when they fail
- Jobs are submitted by batch (see .travis.yml)
- Only tests compilation, run and analysis!!

* Only catch what we ask for

https://warpx.readthedocs.io/

IDEAS productivity

ECP EXASCALE COMPUTING PROJECT

Ginkgo Contribution Pipeline

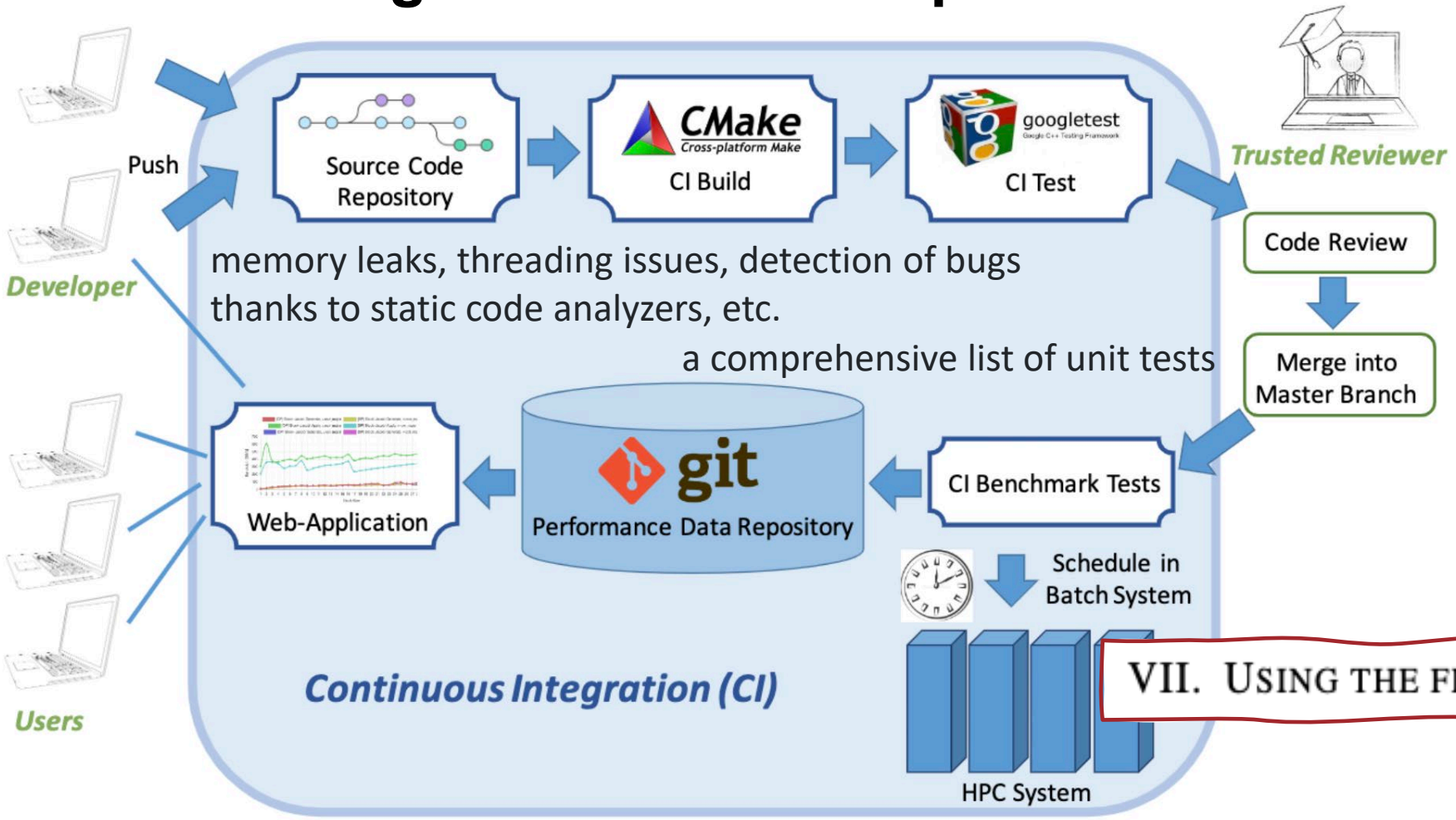
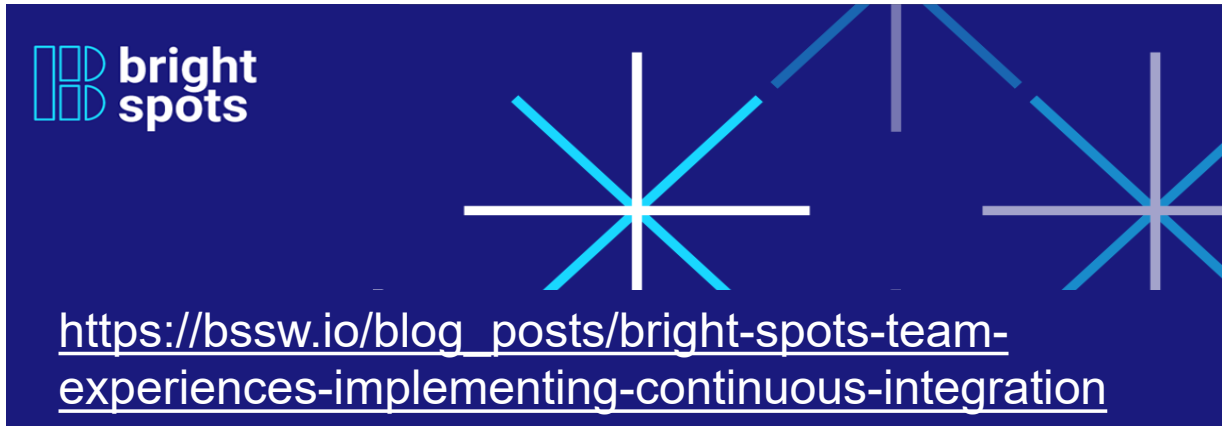


Fig. 1. The software development ecosystem of the GINKGO library.

VII. USING THE FRAMEWORK IN OTHER PROJECTS

An Automated Performance Evaluation Framework for the GINKGO Software Ecosystem Anzt, Cojean, Flegar, Grützmacher, Nayak, Ribizel, 90th Int'l Meeting of Int'l Assoc. Appl. Math. And Mech. (2019).

Team Experiences with CI



- Commonalities:
  - comparing with “golden results” from full-program run
  - unique mindset needed to develop and maintain unit tests
  - "adding armor".
- Most cited benefits:
  - identifying potential bugs early,
  - increasing the project’s ability to receive contributions
- Most cited drawbacks:
  - Effort maintaining tests
  - trial-and-error running tools
  - long-running tests are annoying
  - infrequent random failures (due to network, and other sources)
- Implementation didn't disrupt process
- After initial adoption hurdle, teams start to insist on it

IDEAS productivity

ECP EXASCALE COMPUTING PROJECT