Testing Strategies

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

Software Practices for Better Science: Testing, Reproducibility, and Documentation tutorial @ Exascale Computing Project Tutorial Days

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

License, Citation and Acknowledgements

License and Citation

- **The requested citation the overall tutorial is:** David E. Bernholdt, David M. Rogers, and Gregory R. Watson, Software Practices for Better Science: Testing, Reproducibility, and Documentation tutorial, in Exascale Computing Project Tutorial Days, online, 2023. DOI: 10.6084/m9.figshare.21989507.
- Individual modules may be cited as *Speaker, Module Title*, in Software Practices for Better Science: Testing, Reproducibility, and Documentation tutorial, …

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.

Outline

- Testing
  - Guidelines
  - Examples
  - Bugs / Docs / Issue interactions - Where, Who and How
  - Test Development Examples

- CI
  - Definitions
  - Process outline
  - Examples from Open Projects
    - WarpX
    - Ginkgo
    - JEDI
  - Hints from the front lines
  - Summary: Team experiences with CI

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

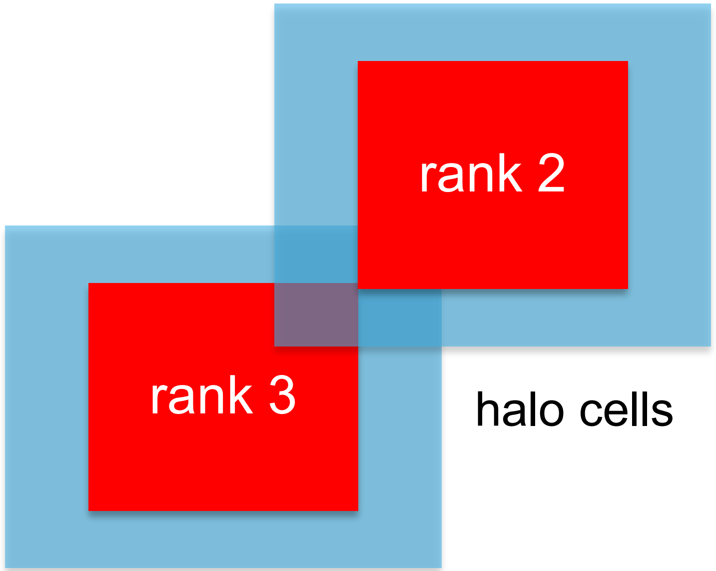
Useful resources https://ideas-productivity.org/resources/howtos/

Why not always use the most stringent testing?

- Effort spent in devising running and maintaining test suite is a tax on team resources
- When the tax is too high…
  - Team cannot meet code-use objectives
- When is the tax is too low…
  - Necessary oversight not provided
  - Defects in code sneak through

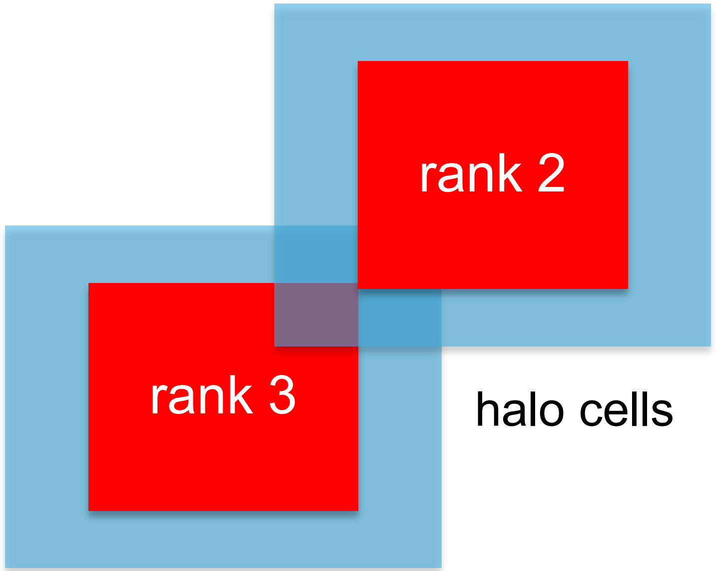
Why not always use the most stringent testing?

- Effort spent in devising running and maintaining test suite is a tax on team resources
- When the tax is too high…
  - Team cannot meet code-use objectives
- When is the tax is too low…
  - Necessary oversight not provided
  - Defects in code sneak through



Why not always use the most stringent testing?

- Effort spent in devising running and maintaining test suite is a tax on team resources
- When the tax is too high…
  - Team cannot meet code-use objectives
- When is the tax is too low…
  - Necessary oversight not provided
  - Defects in code sneak through
- Evaluate project needs:
  - Objectives: expected use of the code
  - Lifecycle stage: new or production or refactoring
  - Team: size and degree of heterogeneity
  - Lifetime: one off or ongoing production
  - Complexity: modules and their interactions



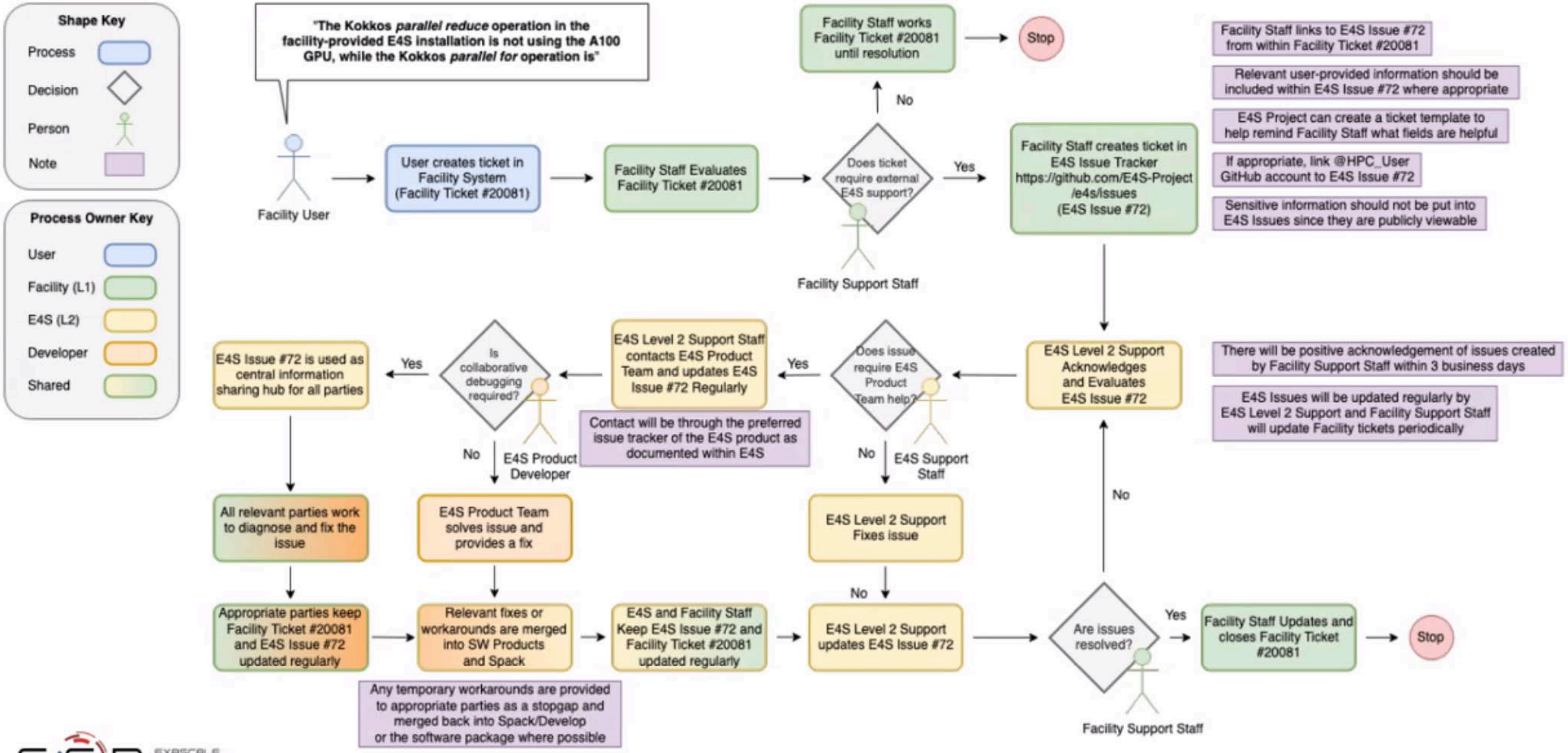
Additional Notes: Good Testing Practices

- 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

What and Where to File Bugs? Issues? Doc. Requests?

- Doesn't compile / install / run as documented? No documentation?
  - These are vital fixes and the devs will (*should*) thank you.
  - But *first* check HPC site facilities / colleagues.
  - Then complain (politely) to maintainers when something doesn't work.
  - "standard" contribution policy: If it isn't obvious to someone, it should be documented.
- Got it working?
  - Document in your own project (will help onboarding, and you later).
  - Reply to same people anyway. (can increase your project's visibility)
- Submit issues / PRs for docs to upstreams.
  - Great way to make friends & forge collaborations.
- Send self-contained, full examples (reference existing docs).

E4S / Facility Software Support Model



ECP EXASCALE COMPUTING PROJECT

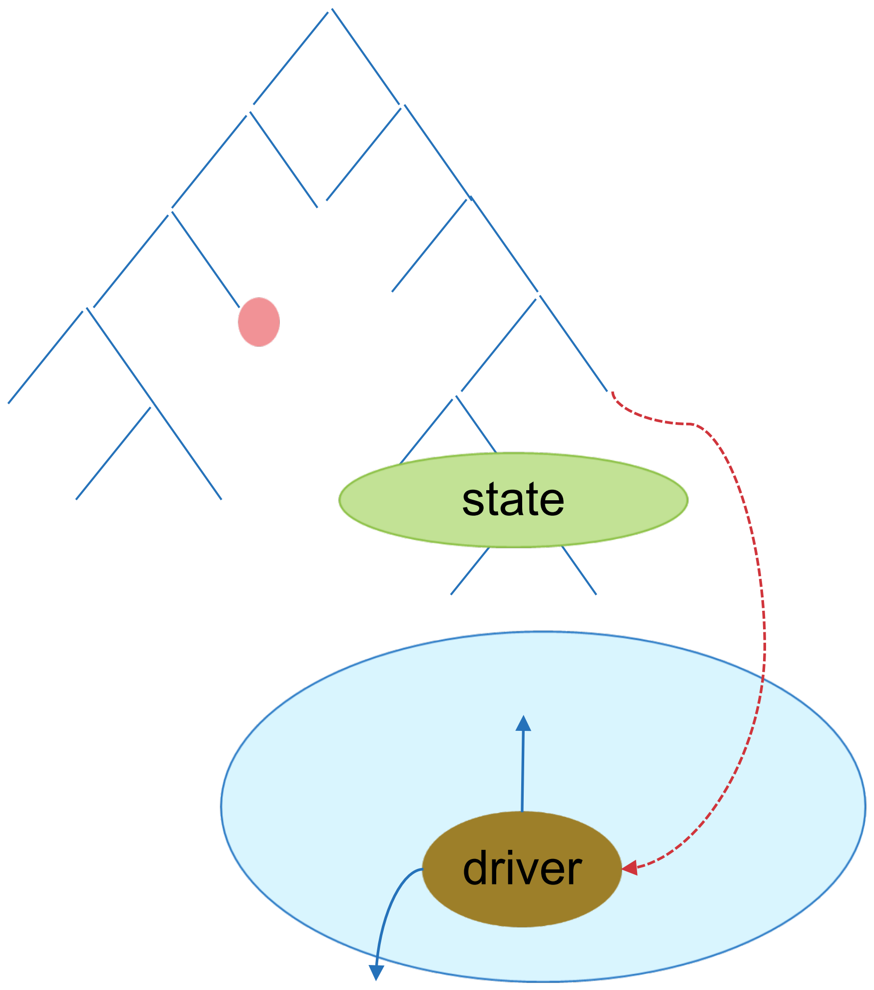
Example 1: Test Development For a New Code

- Development of tests and diagnostics goes hand-in-hand with code development
  - Compare against simpler analytical or semi-analytical solutions
  - Build granularity into testing
  - Use scaffolding ideas to build confidence
  - Always inject errors to verify that the test is working
  - Non-trivial to devise good tests, but extremely important

Example 2:
Test Development For a Legacy Code

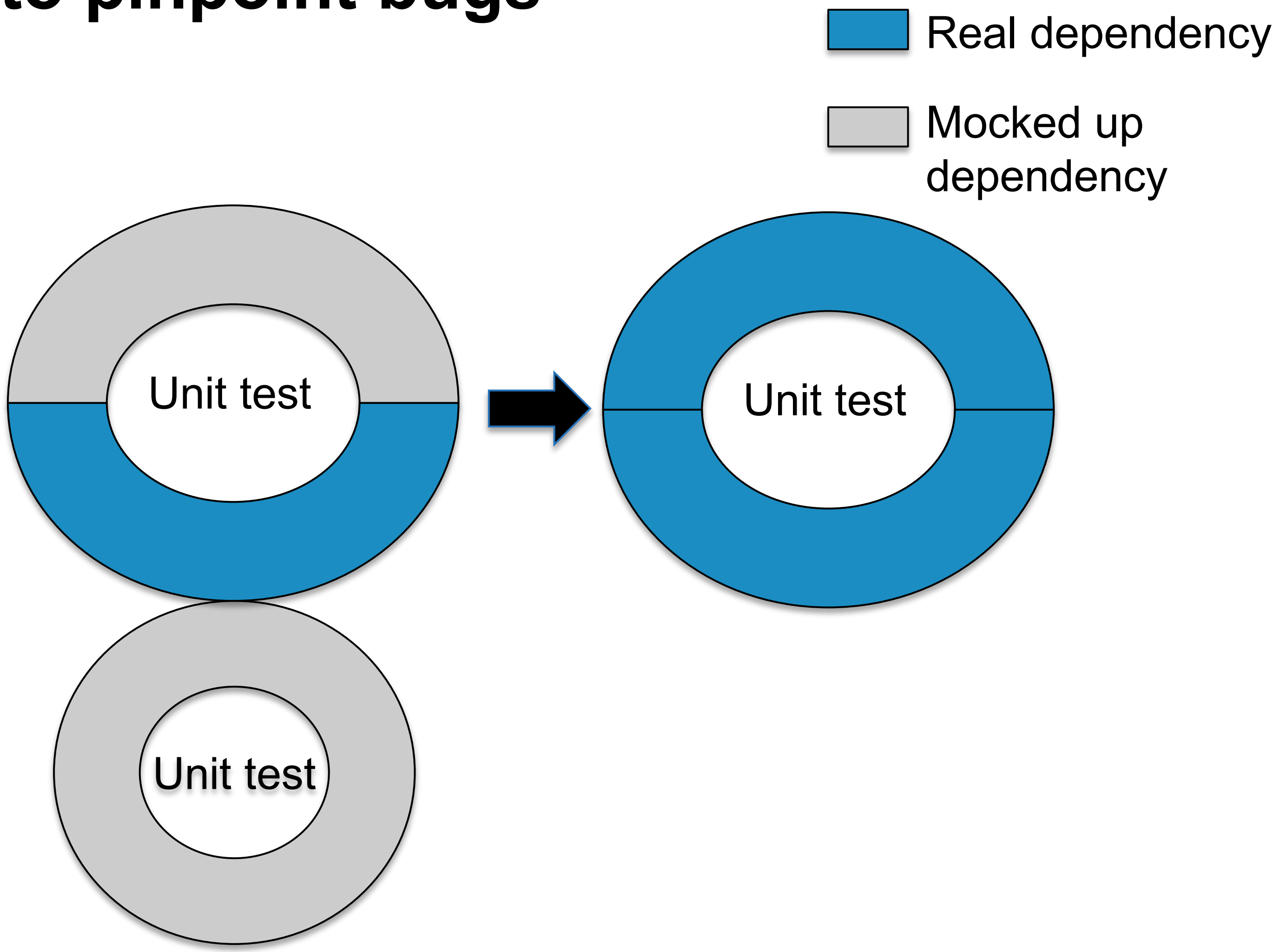
There may not be existing tests

- 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



Example 3: Structuring Tests to pinpoint bugs

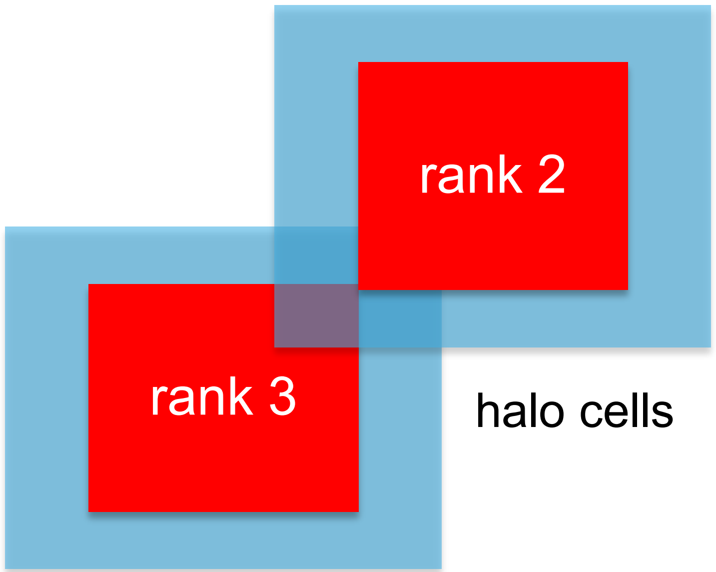
- Bottom-up picture
  - Components can be exercised against known simpler applications
  - Same applies to combination of components
- Build a scaffolding of verification tests to gain confidence

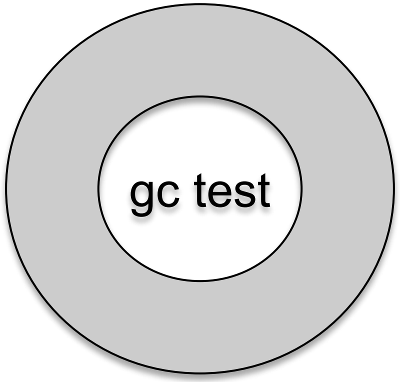


Example 3: Structured Testing

Unit test for Grid halo cell fill

- Verification of guard/ghost/halo cell fill
- Initialize field on interior cells (red)
- Apply guard cell fill
- Check for equivalence with known fill pattern

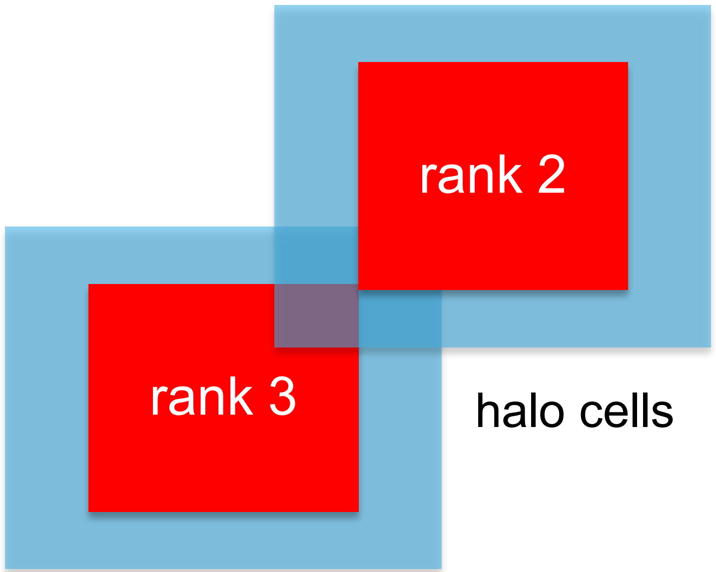


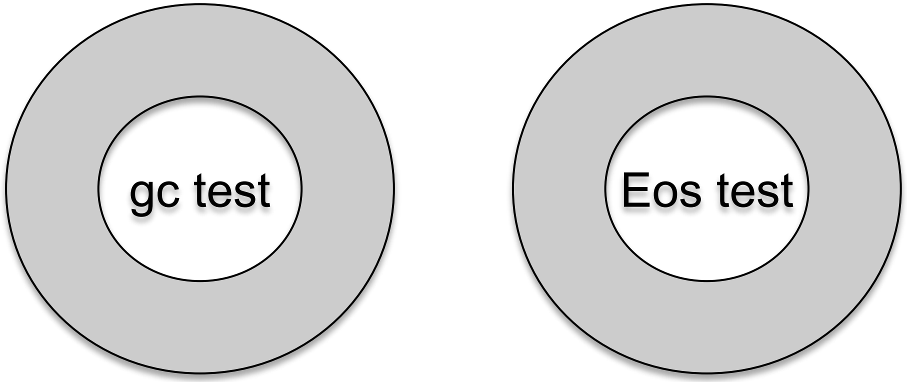


Example 3: Structured Testing

Unit test for Grid halo cell fill

- Verification of guard/ghost/halo cell fill
- Initialize field on interior cells (red)
- Apply guard cell fill
- Check for equivalence with known fill pattern



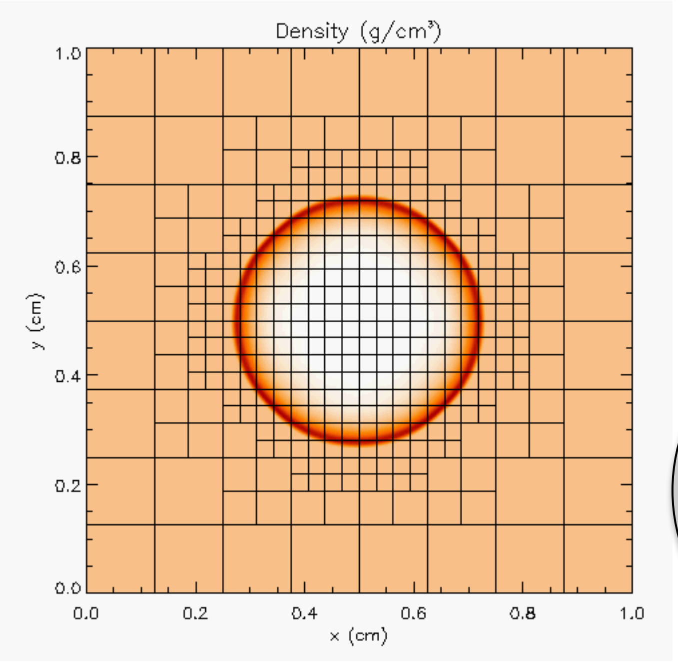


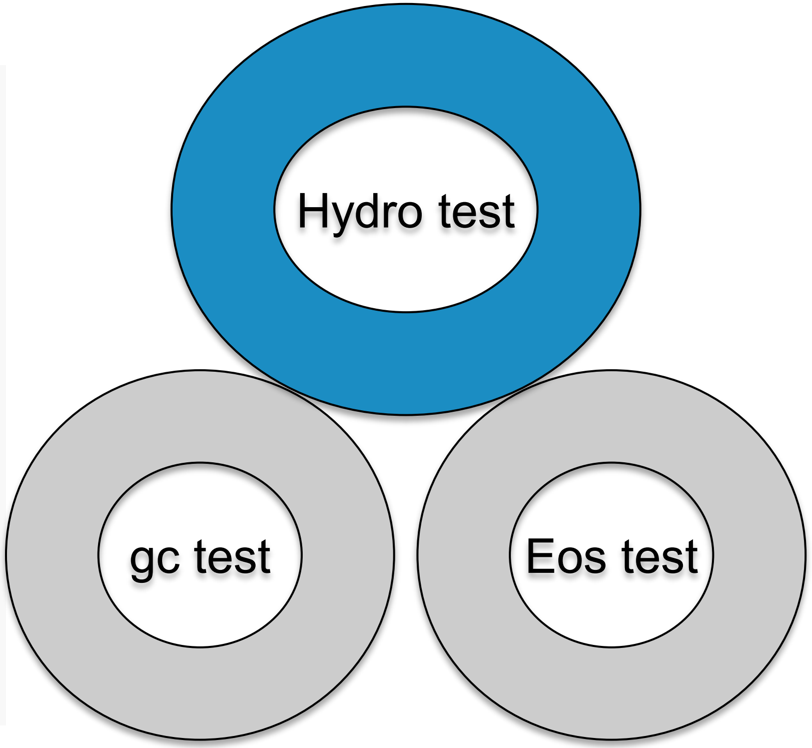
Next, build an EOS Test – is E(V,T) consistent with P(V,T)?

Example 3: Structured Testing

Unit test for Hydrodynamics

- Sedov blast wave
- High pressure at the center
- Shock moves out spherically
- Known analytical solution





Though it exercises mesh, hydro and eos, if mesh and eos are verified first, then this test verifies hydro

More testing needed for Grid using AMR
Flux correction and regridding

Example 3: Structured Testing

For AMR, correct behavior of flux conservation and regridding should also be verified.

Reason about correctness for testing Flux correction and regridding

IF Guardcell fill and EOS unit tests passed

- Run Hydro without AMR
  - If failed fault is in Hydro
- Run Hydro with AMR, but no dynamic refinement
  - If failed fault is in flux correction
- Run Hydro with AMR and dynamic refinement
  - If failed fault is in regridding

IDEAS productivity

ECP EXASCALE COMPUTING PROJECT

Example 4: Coverage Matrix (Interoperabilities)

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!**)

Testing Takeaways

- Context: understand testing needs and costs
- Devise tests to enable quick pinpointing of errors through reasoning about their behavior
- test at various granularities – bottom-up (UNIT/verification) through top-down (integration/validation)
- Tests at various complexities – CI vs. regression
- Maintain a holistic validation strategy: think globally, act locally

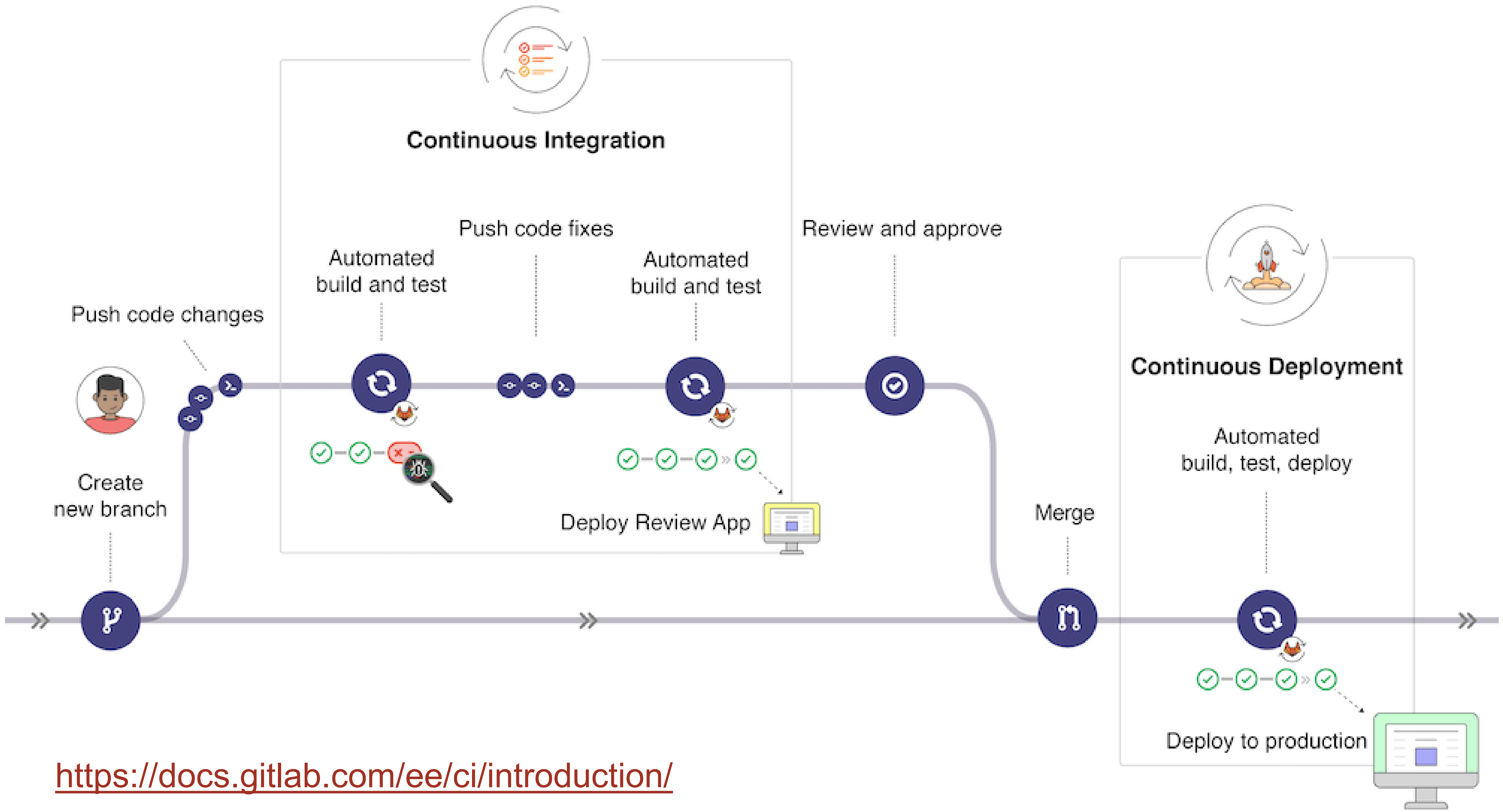
…….Questions ?

Outline

- Testing
  - Guidelines
  - Examples
  - Bugs / Docs / Issue interactions - Where, Who and How
  - Test Development Examples

- CI
  - Definitions
  - Process outline
  - Examples from Open Projects
    - WarpX
    - Ginkgo
    - JEDI
  - Hints from the front lines
  - Summary: Team experiences with CI

What is Continuous Integration (CI)?



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

CI Components

- Testing
  - Focused, critical functionality (infrastructure), fast, 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

Documentation Driven Development vs. Automated Testing vs. CI

- ***Documentation Driven Development:*** A development methodology where documentation is written before the code
  - Forces design of functional units of code before solving implementation details
  - May be combined with test-driven development
- ***Automated Testing***: Software that automatically performs tests on a regular basis and reliably detects and reports anomalous behaviors/outcomes.
  - Examples: Auto-test, CTest/CDash, nightly testing, etc.
  - May live “next to” your development workflow
  - Potential issues: change attribution, timeliness of results, multiple branches of development
- ***Continuous Integration (CI)***: automated testing performed at high frequency and fine granularity
  - Aimed at preventing code changes from breaking key branches of development (e.g. main)
  - Lives “within” your development workflow
  - Potential issues: extreme automation, test granularity, coverage, 3rd-party services/resources

Examples…

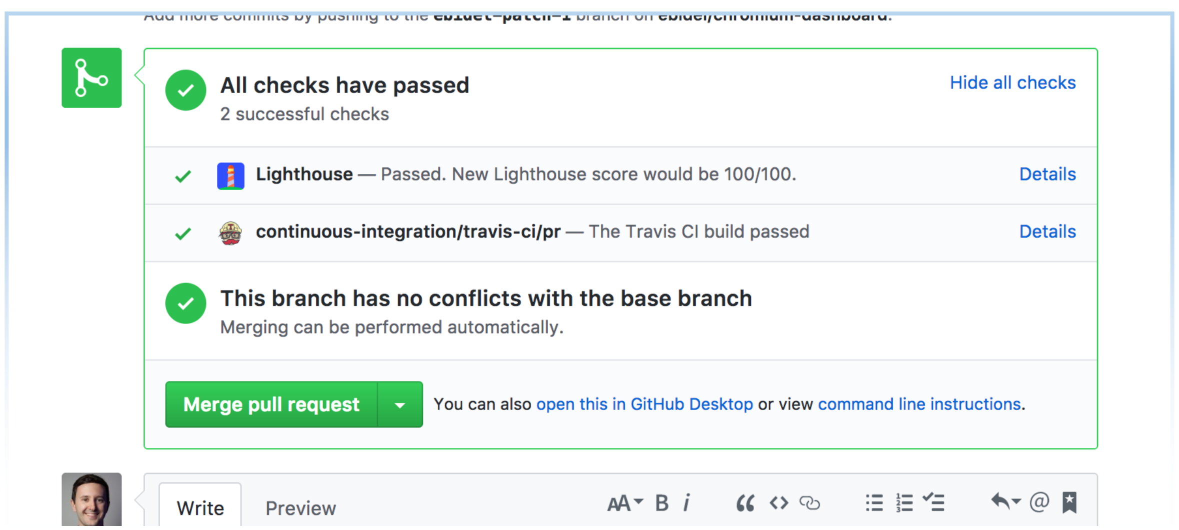
Automated Nightly Testing Dashboard
Lives "next to" your development work

CI Testing
Lives embedded in 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



What can make CI difficult

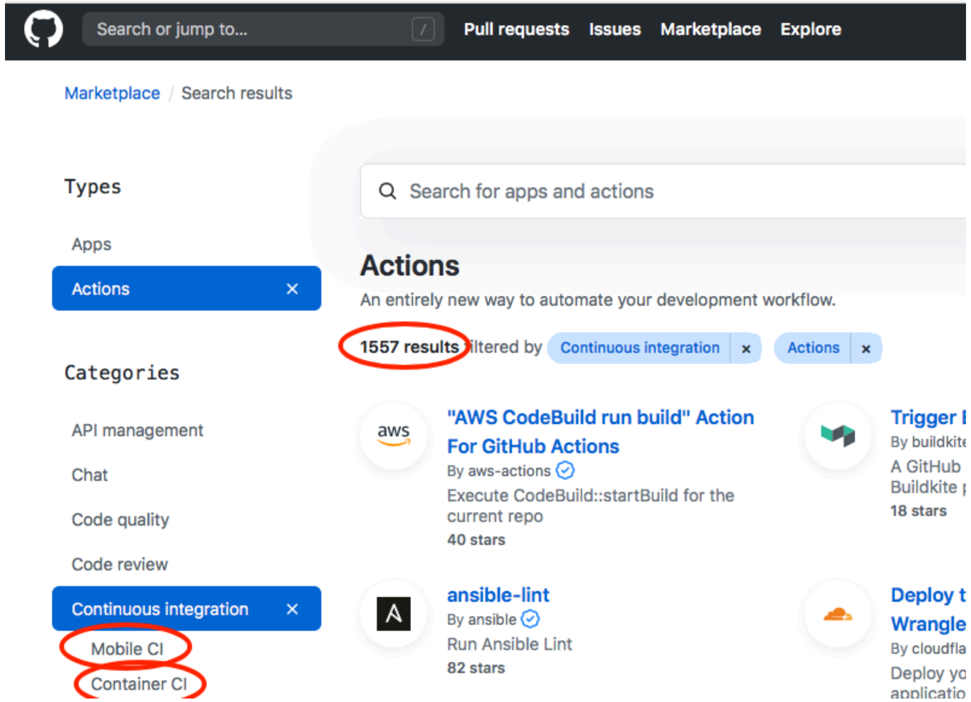
Common situations

- Just getting started
  - Many technologies/choices; often in the ”cloud”
  - Solution: start small, simple, build up
- Developing suitable tests
  - Many project’s existing tests not suitable for CI
  - CI testing is a balance of thoroughness and responsiveness
  - Solution: Simplify/refactor and/or sub-setting test suite
- Ensuring sufficient coverage
  - Some changes to code never get tested – CI can provide a false sense of security
  - Solution: tools to measure it, enforce always increasing

Advanced situations

- Defining failure for *many* configurations / inconsistent failures
  - Bit-for-bit (exact) match vs. fuzzy match
  - Solution: absolute/relative tolerances → AI/ML
- Numerous 3rd party libraries (TPLs)
  - Compiling takes too long
  - Solution: cache pre-built TPLs, containers
- Performance testing
  - Avoid time-, space-, scaling-performance degradation
  - Solution: Performance instrumentation and *scheduled* testing

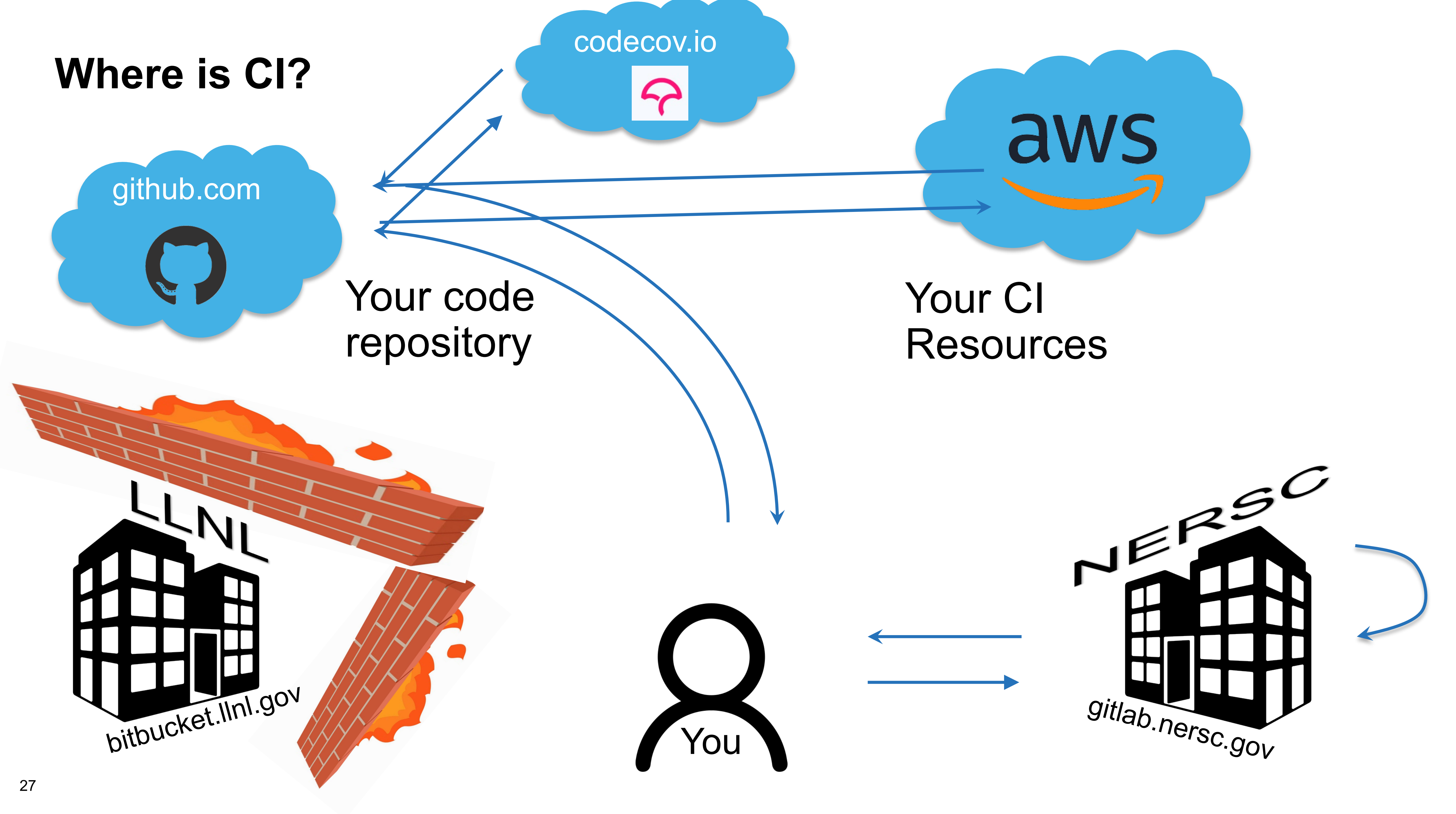
CI Resources (Where do jobs run?)



- Free Resources
  - GitHub, BitBucket, GitLab, etc. provide shared runners
  - AWS, Azure Pipelines have free tiers that can be used
  - All launch a VM (Linux variants, Windows and OSX)
    - Constrained in time/size, hardware (e.g. GPU type/count)
    - Not a complete solution for many HPC/scientific codes, but a useful starting point.
- Site-local Resources
  - Group, department, institution, computing facility
  - Examples: CADES @ ORNL, Bamboo @ LLNL, Jenkins @ ANL, Travis+CDash @ NERSC
  - ECP Program: GitLab-CI @ ANL, LANL, LLNL, NERSC, ORNL, SNL
- Create your own by setting up resources/services

IDEAS productivity EXASCALE COMPUTING PROJECT

Where is CI?


codecov.io
github.com
aws
Your code repository
Your CI Resources
LLNL
bitbucket.llnl.gov
You
NERSC
gitlab.nersc.gov

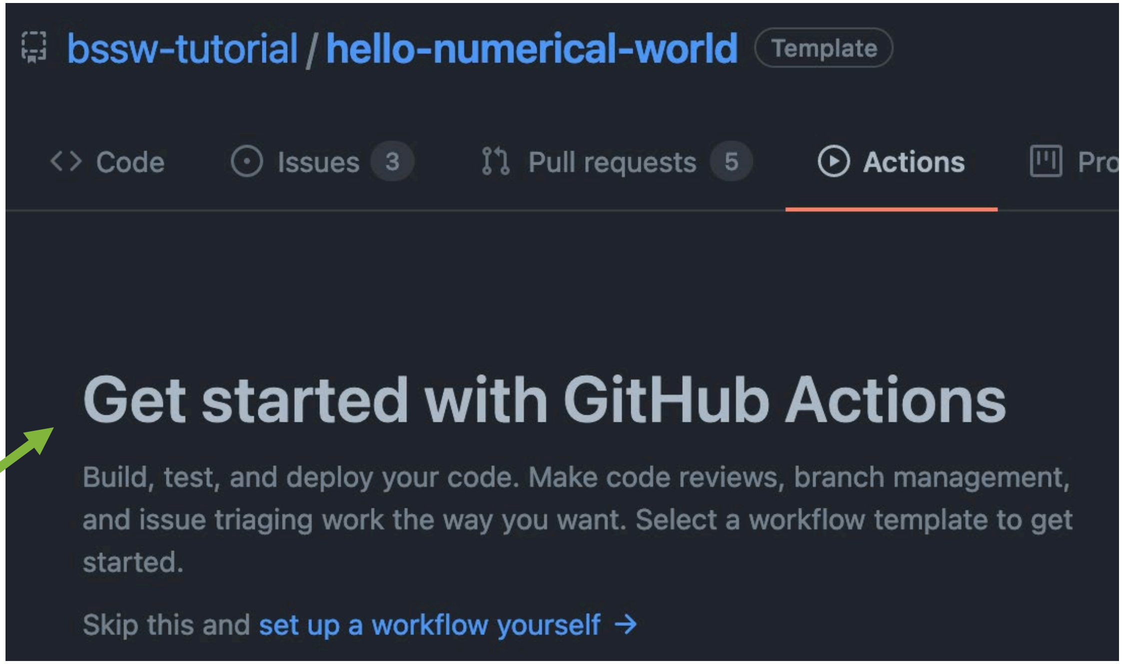
Getting started with CI

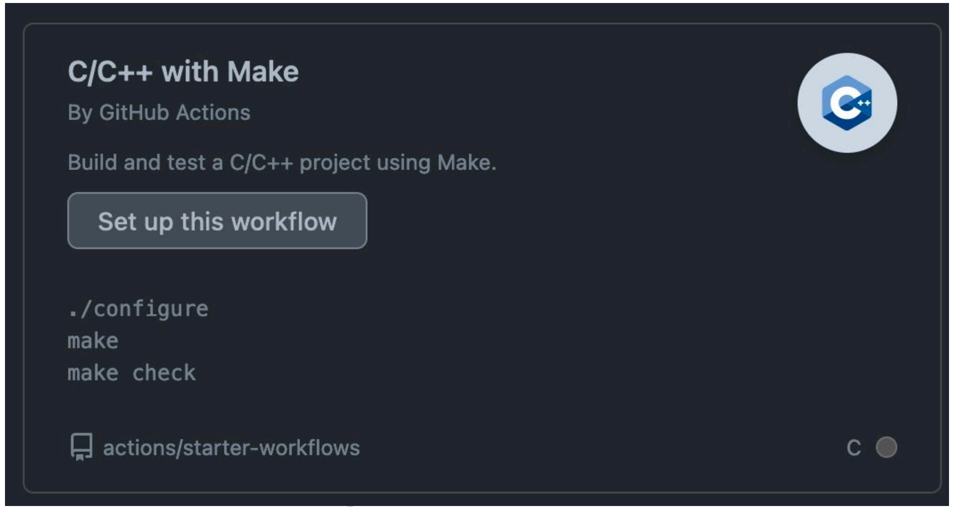
- What *configuration* is most important?
  - Examples: gcc, icc, xlc? MPI-2 or MPI-3? Python 2, 3 or 2 & 3?
- What *functionality* is most important?
  - Examples: vanilla numerical kernels? OpenMP kernels? GPU kernels? All of these?
- Good candidates…
  - A “hello world” example for your project
  - At a minimum, even just building the code can be a place to start!
  - Once you’ve got the basics working, its easy to build up from there

Getting started with CI:

Setting up CI

Service	Interface	
GitHub Actions	Repo YAML file	.github/workflows/<test_name>.yml
GitLab	Web page configurator + repo YAML file [& repo scripts]	/.gitlab-ci.yml in root of repo
Bamboo	Web page configurator + repo scripts	
Travis	repo YAML file [& repo scripts]	/.travis.yml in root of repo





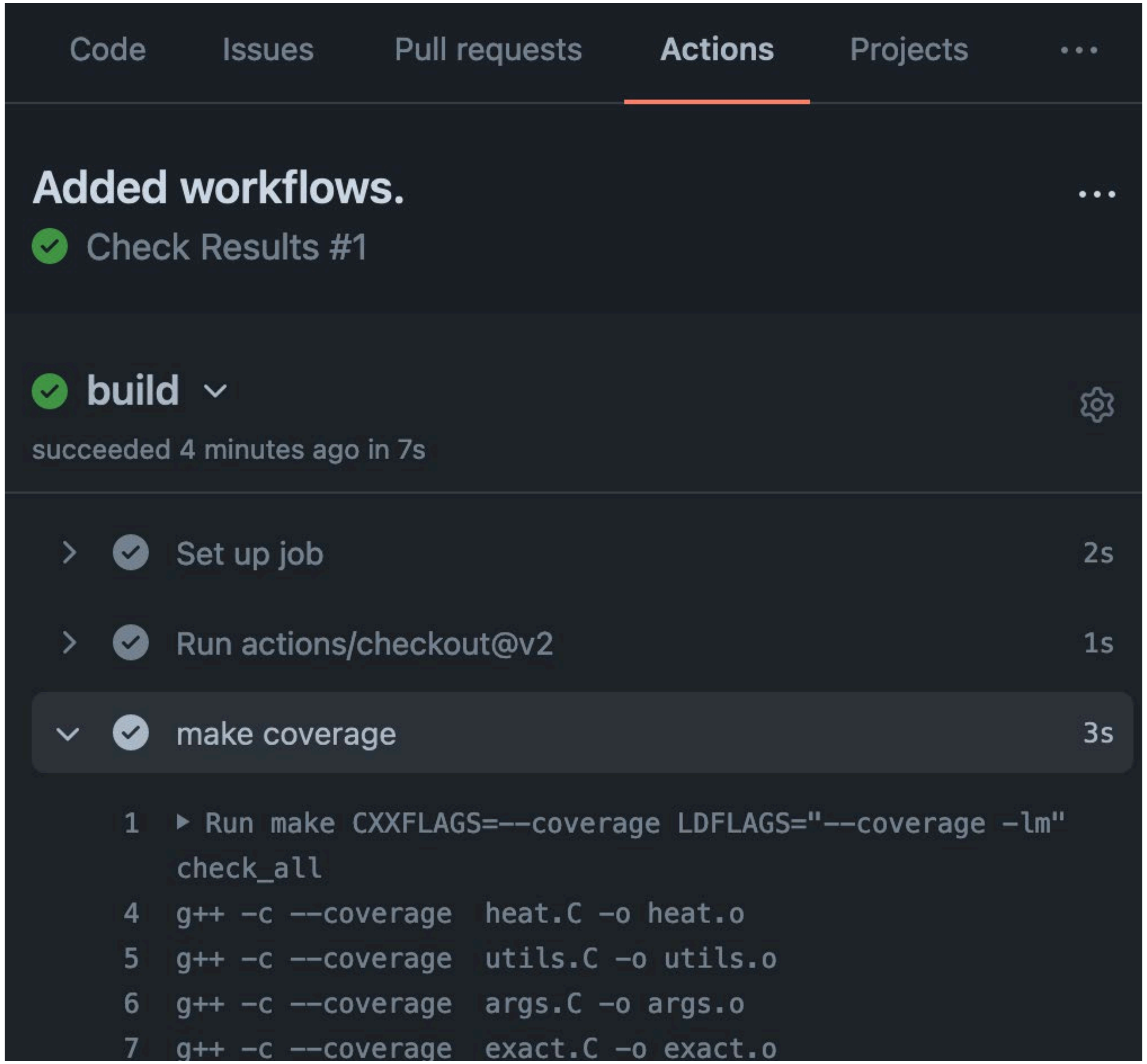
Getting started with GitHub Actions:

19 lines (15 sloc) | 359 Bytes

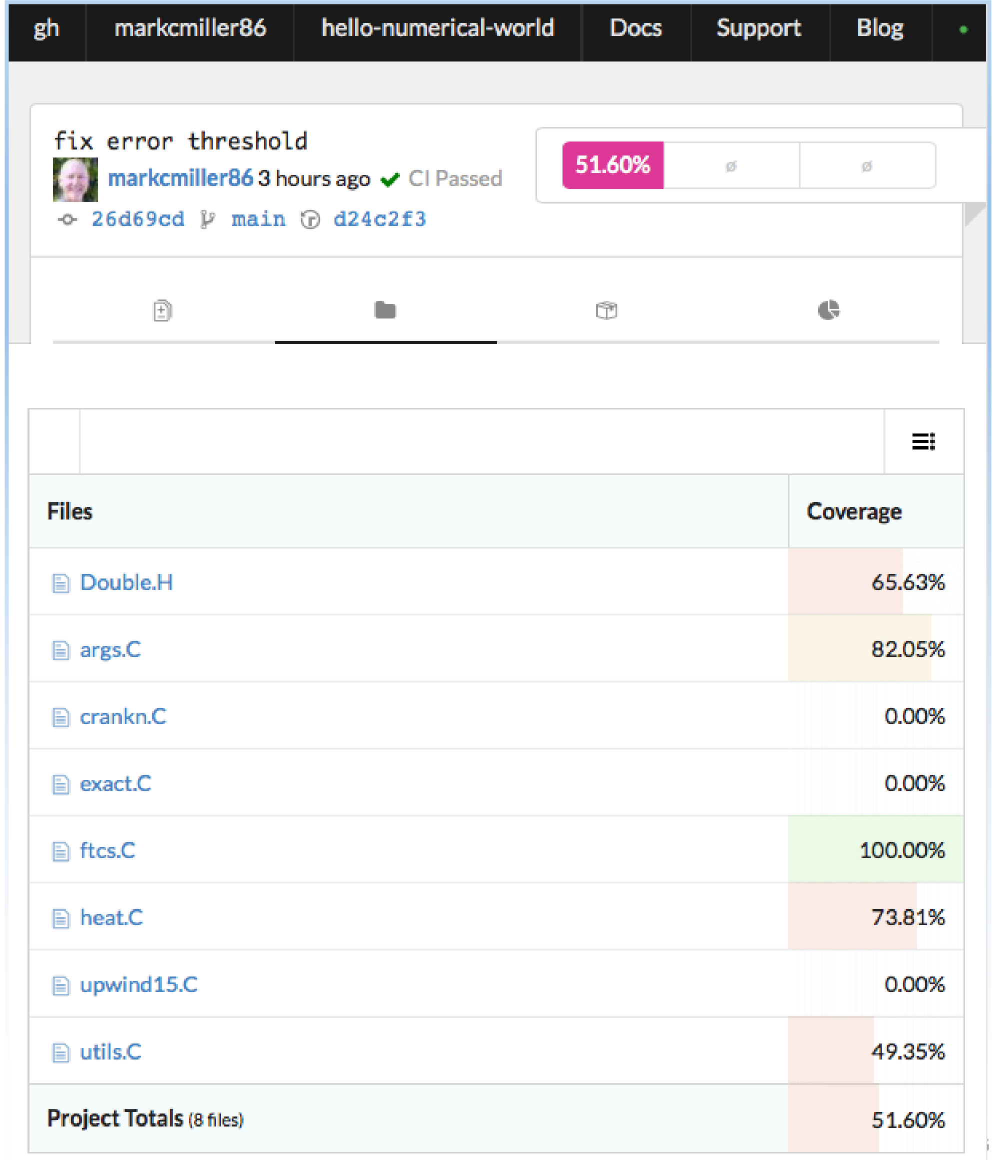
```yaml
name: Check Results

on:
  push:
    branches: [ main ]
  pull_request:
    branches: [ main ]

jobs:
  build:

    runs-on: ubuntu-latest

    steps:
    - uses: actions/checkout@v2
    - name: make coverage
      run: make CXXFLAGS=--coverage LDFLAGS="--coverage -lm" check_all
    - name: upload coverage
      run: bash <(curl -s https://codecov.io/bash)
```

EXASCALE COMPUTING PROJECT

github.com



codecov.io



GitHub Actions – results of workflow test runs

Workflows

- All workflows
- (TEST) Pyomo Windows Tests ...
- (WIP) Pyomo Windows Test (P...
- (WIP) Pyomo Windows Test (P...
- (WIP) Pyomo Windows Tests (...
- (WIP) Windows Pip Cmd Pyom...
- GitHub Branch CI
- GitHub CI
- Pyomo Release Distribution Cr...
- Python package
- Ubuntu Pyomo Single Python ...
- Ubuntu Pyomo Workflow (Slim,...

GitHub CI

Showing runs from all workflows named GitHub CI

event:push workflow:"GitHub CI"

461 workflow run results Event Status Branch Actor

Run	Branch	Time
Merge pull request #1902 from jsiirola/fix-unittest-rc GitHub CI #121: Commit 901b487 pushed by blnicho	main	20 hours ago 1h 3m 55s
Merge pull request #1901 from mrmundt/remove-six GitHub CI #117: Commit a101b6d pushed by mrmundt	main	2 days ago 1h 3m 12s
Merge pull request #1896 from jsiirola/abstract-disa... GitHub CI #112: Commit 1f9dd19 pushed by jsiirola	main	2 days ago 1h 5m 39s
Merge pull request #1898 from mrmundt/py-unittest GitHub CI #109: Commit 1beb848 pushed by michaelbynum	main	3 days ago 1h 3m 33s
Merge pull request #1893 from jsiirola/config-enum GitHub CI #105: Commit 9aa1186 pushed by jsiirola	main	3 days ago 1h 11m 3s

What could possibly go wrong?

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)

Software Supply Chain Stability / Security

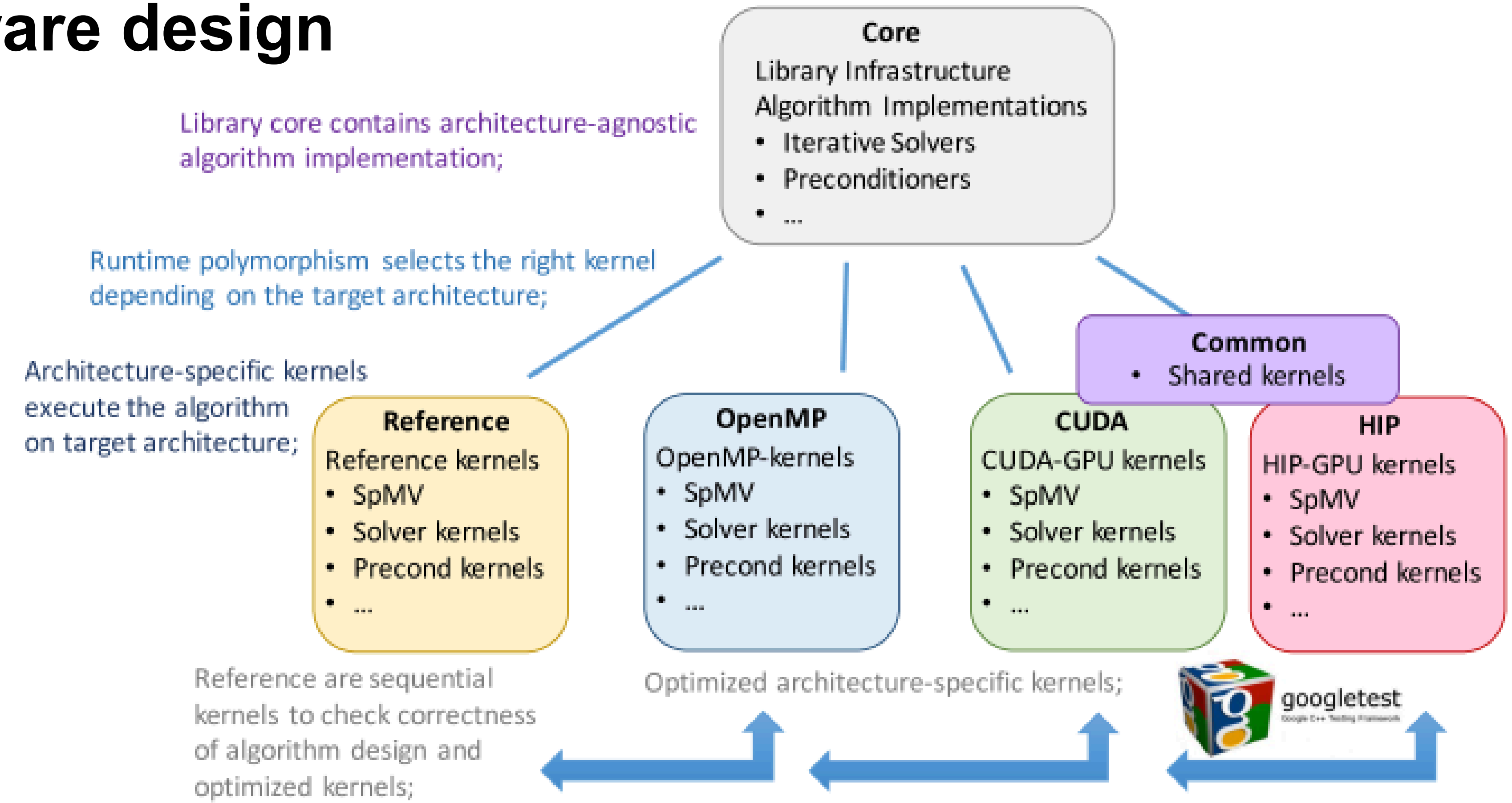
- Testing your dependencies
- GPG-signatures for releases (add confidence that code has not been tampered with)
- How does spack do this? (npm, etc.)

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



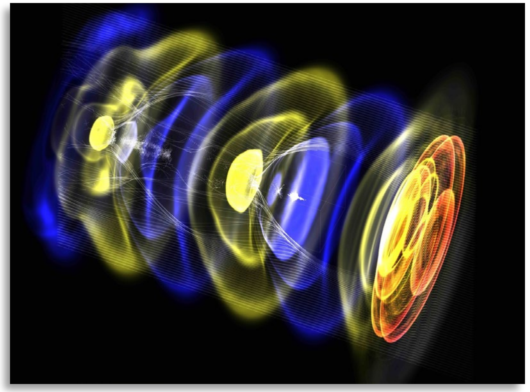
Aside from GINKGO being used as a framework for algorithmic research, **its primary intention is to provide a numerical software ecosystem designed for easy adoption by the scientific computing community**. This requires sophisticated **design guidelines and high quality code**.

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

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)

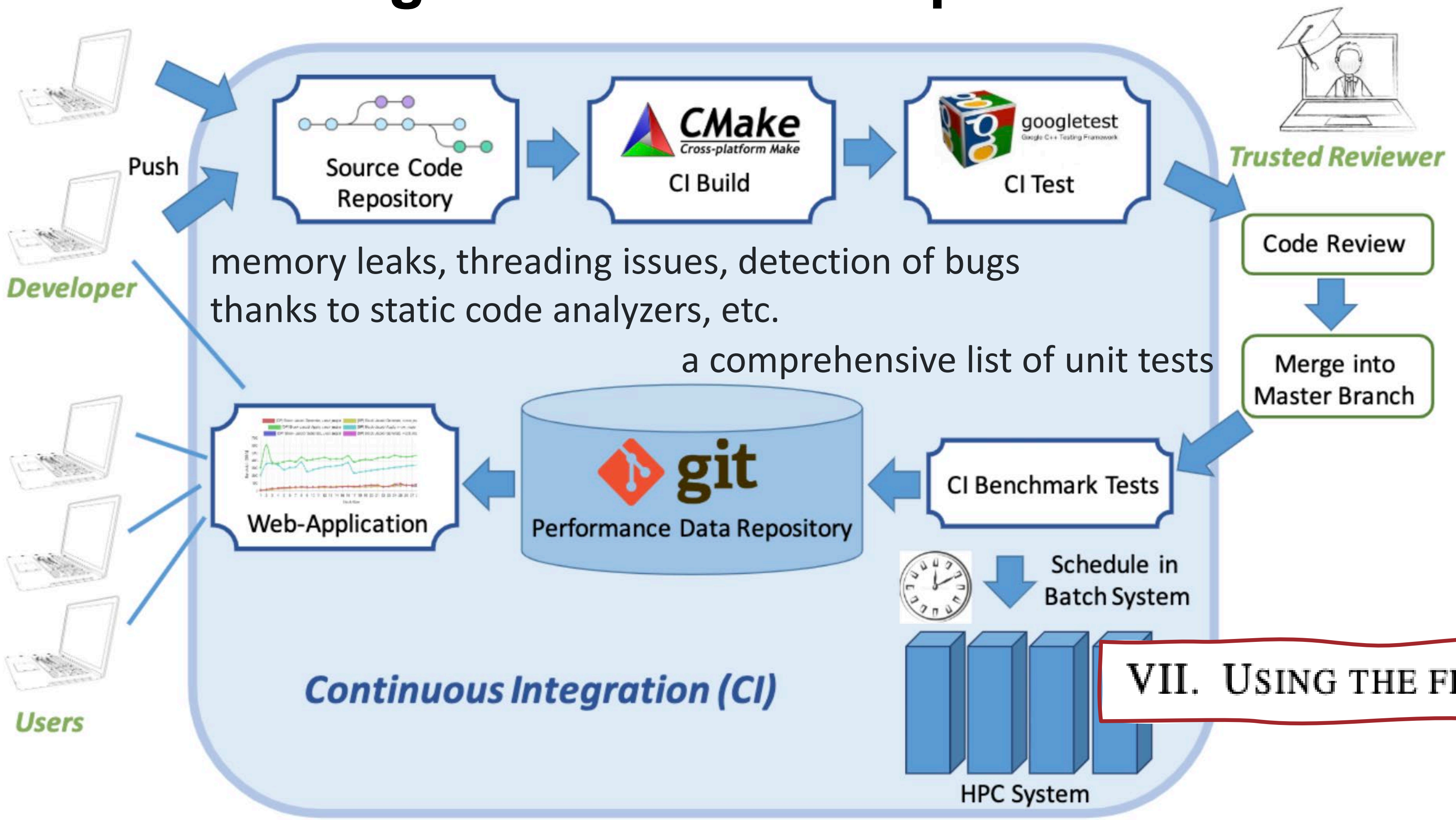
TravisCI tests on every commit on GitHub

- 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/

Ginkgo Contribution Pipeline



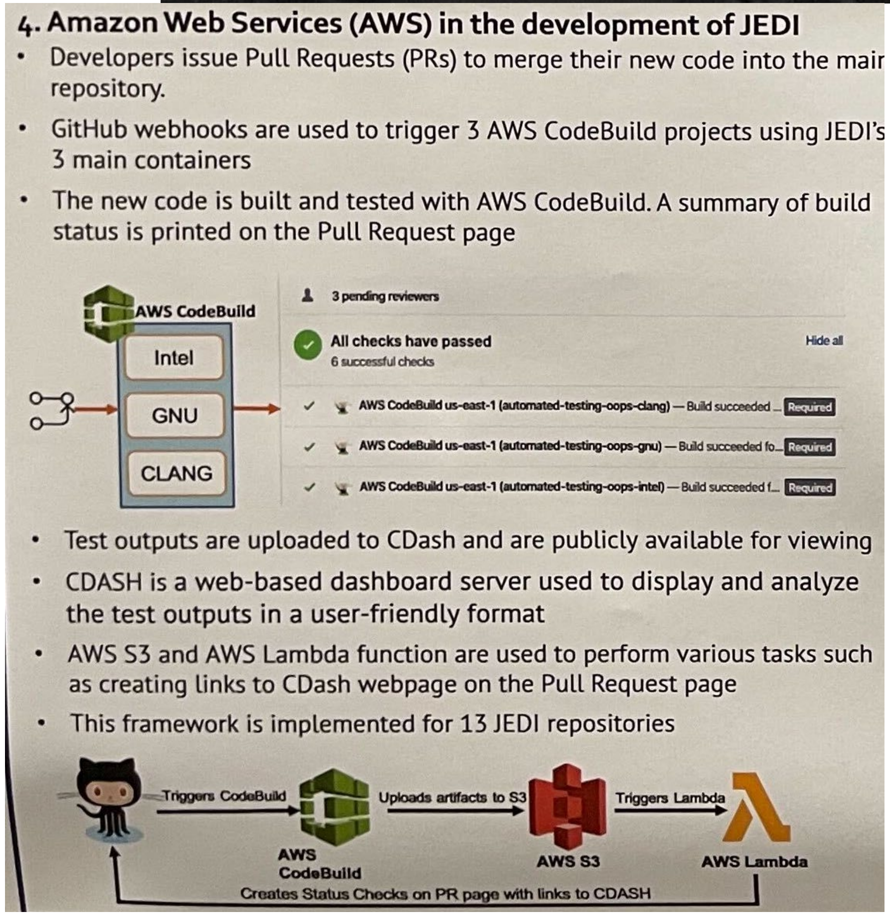
VII. USING THE FRAMEWORK IN OTHER PROJECTS

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

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).

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

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**/](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
```

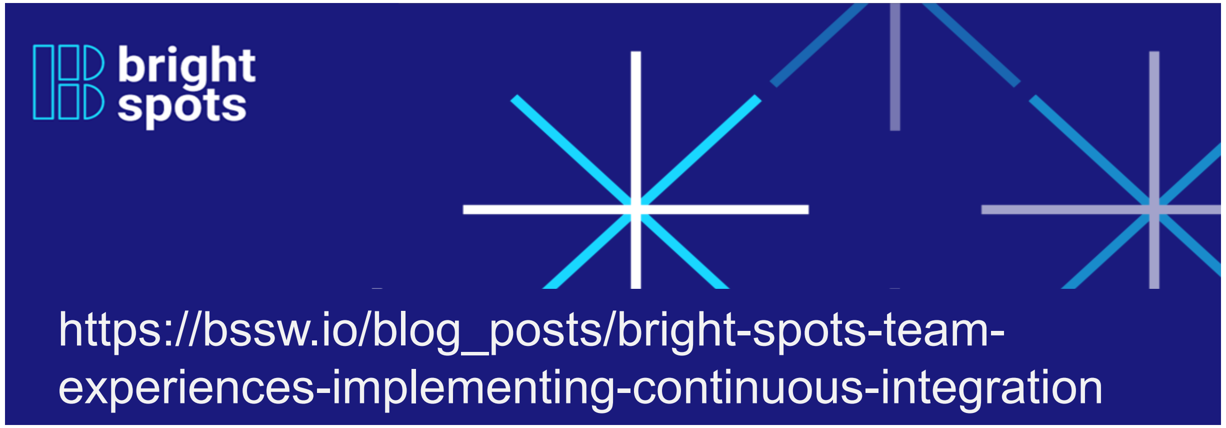
Hints from the front lines

Configure in Settings / Github Pages

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

```yaml
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 }
```

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