# Testing Complex Software

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

Developing a Testing and Continuous Integration Strategy for your Team tutorial @ Exascale Computing Project Annual Meeting

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: Gregory R. Watson and David M. Rogers, Developing a Testing and Continuous Integration Strategy for your Team tutorial, in Exascale Computing Project Annual Meeting, online, 2022. DOI: 10.6084/m9.figshare.19608927**
- Individual modules may be cited as *Speaker, Module Title*, in Better Scientific Software 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 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 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 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.

# 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

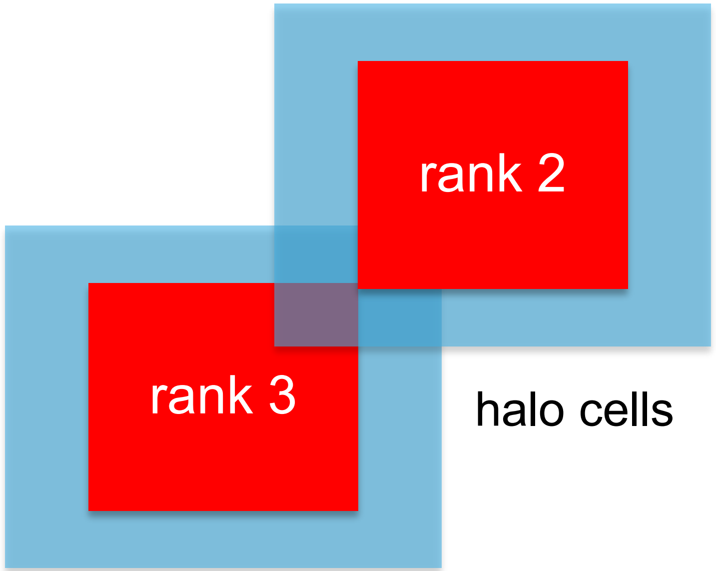
Useful resources https://ideas-productivity.org/resources/howtos/

# 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

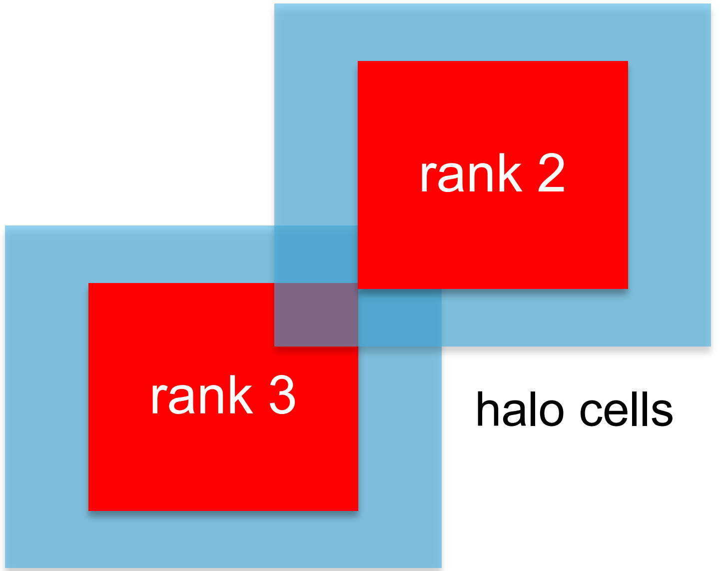
# Why not always use the most stringent testing?

- 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

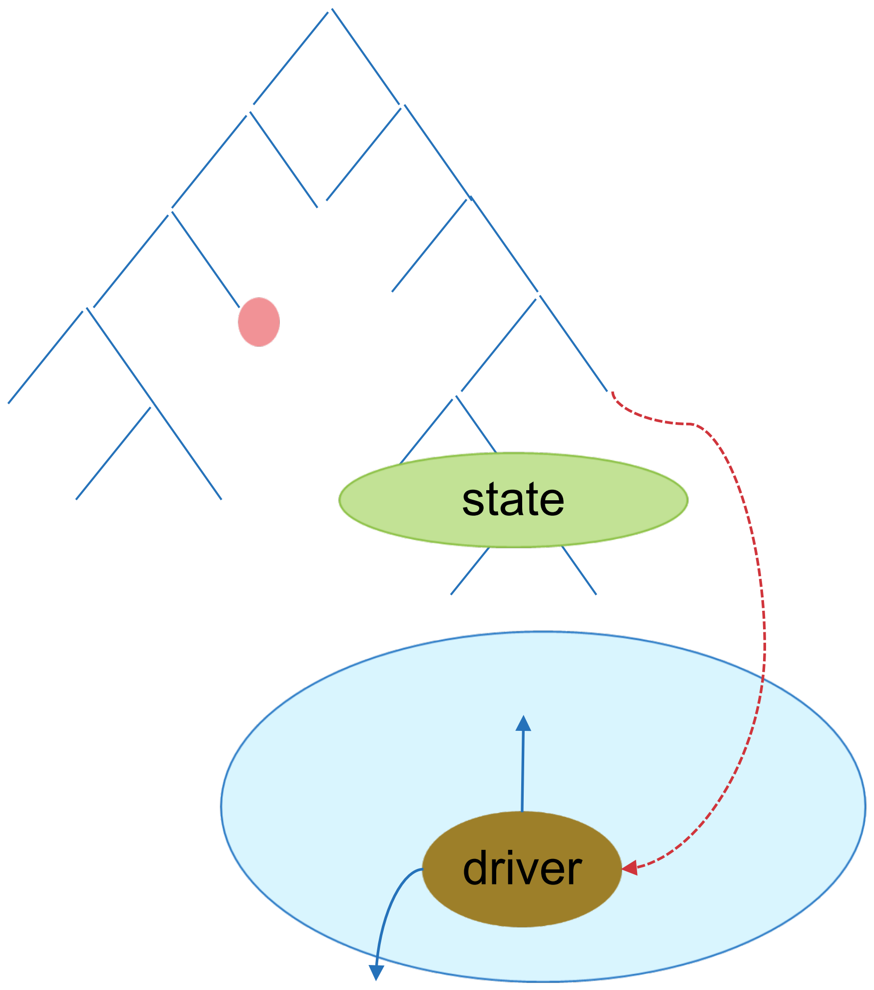
# Example 1: Test Development For a New Code

- 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

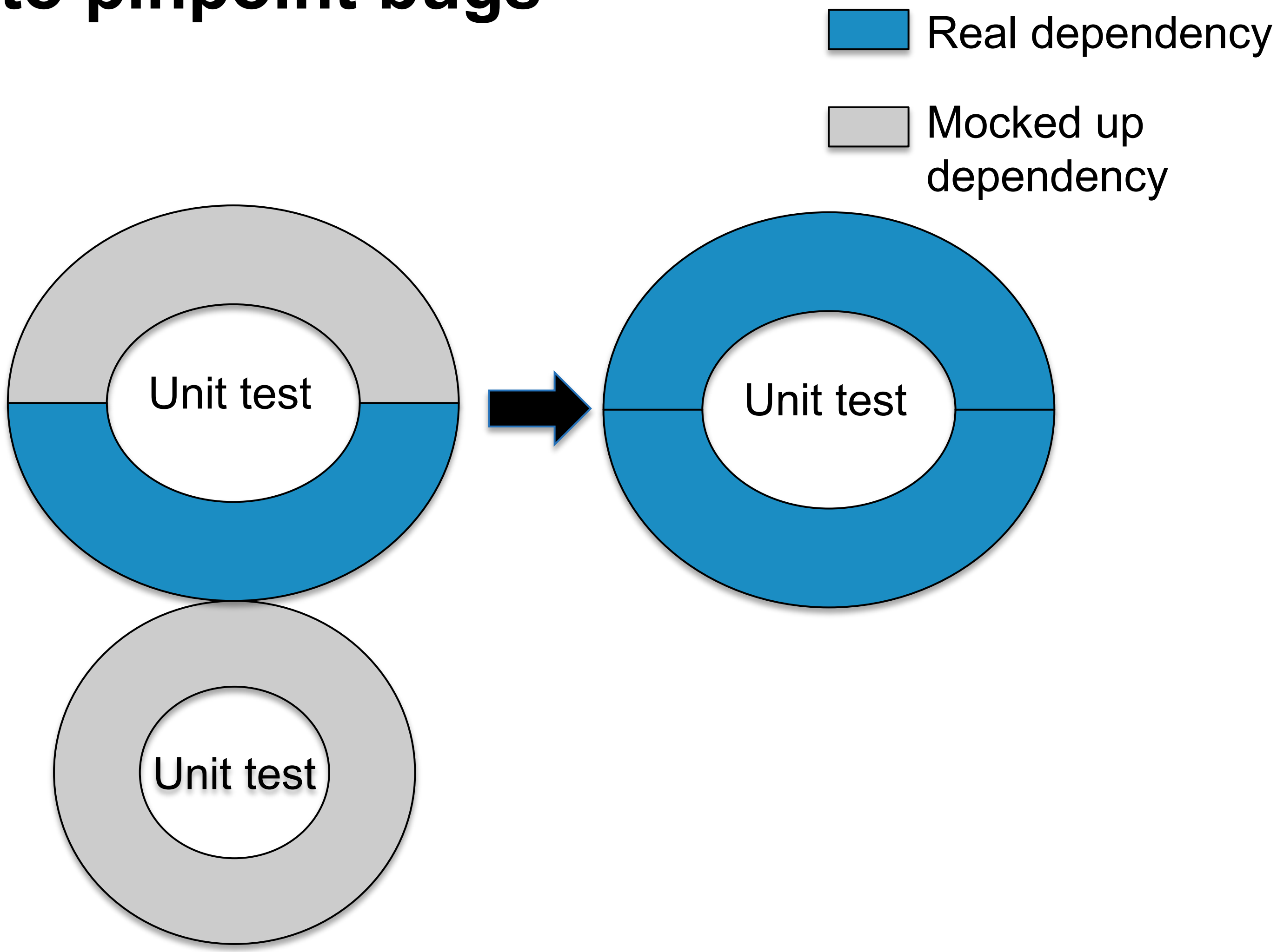
# Example 2: Test Development For a Legacy Code

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

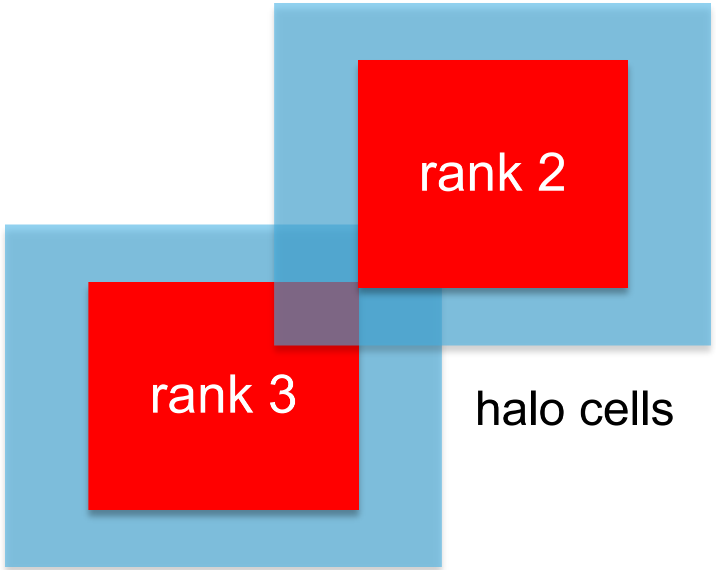


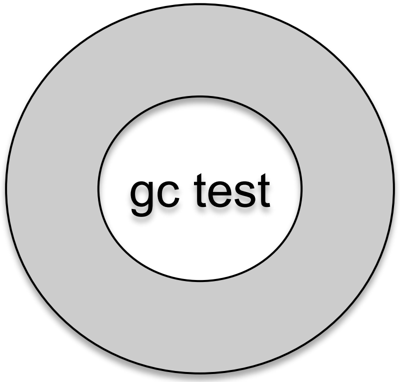

- Bottom-up picture
  - Components can be exercised against known simpler applications
  - Same applies to combination of components
- Build a scaffolding of verification tests to gain confidence

# 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

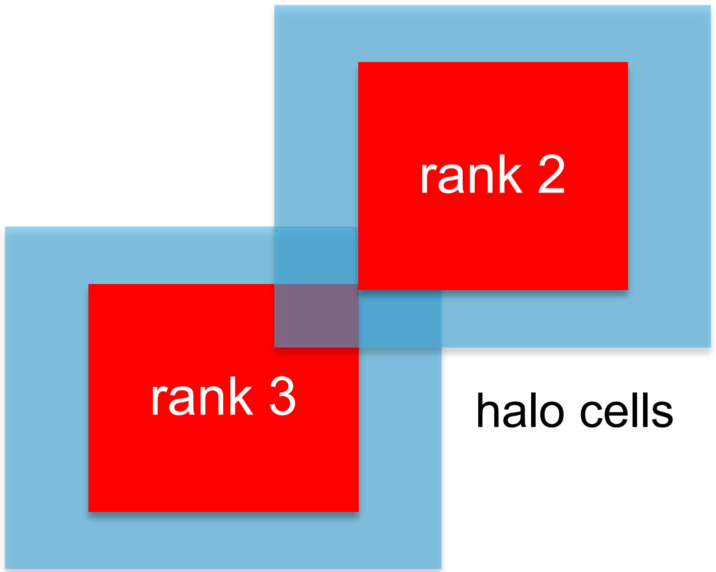


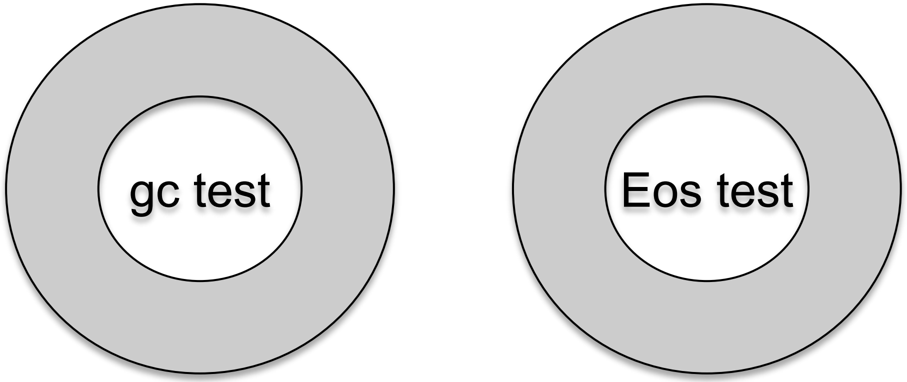



# Example 3: Structured Testing

## 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



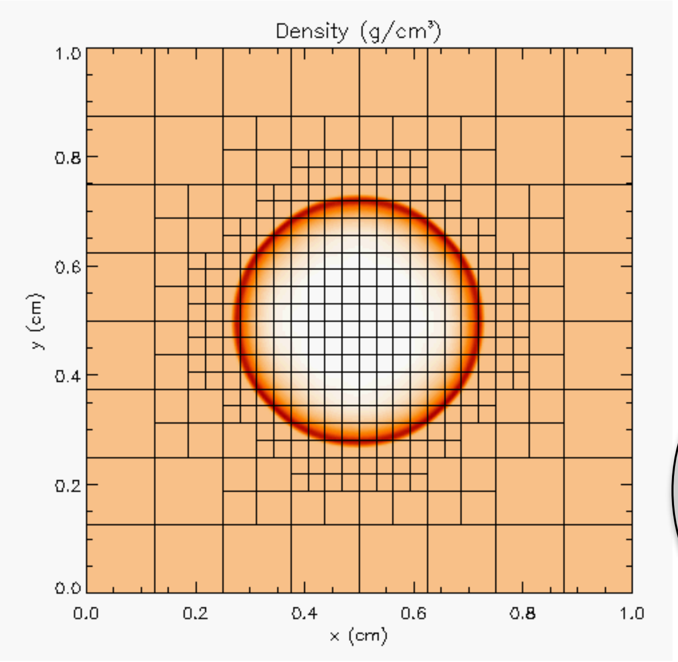



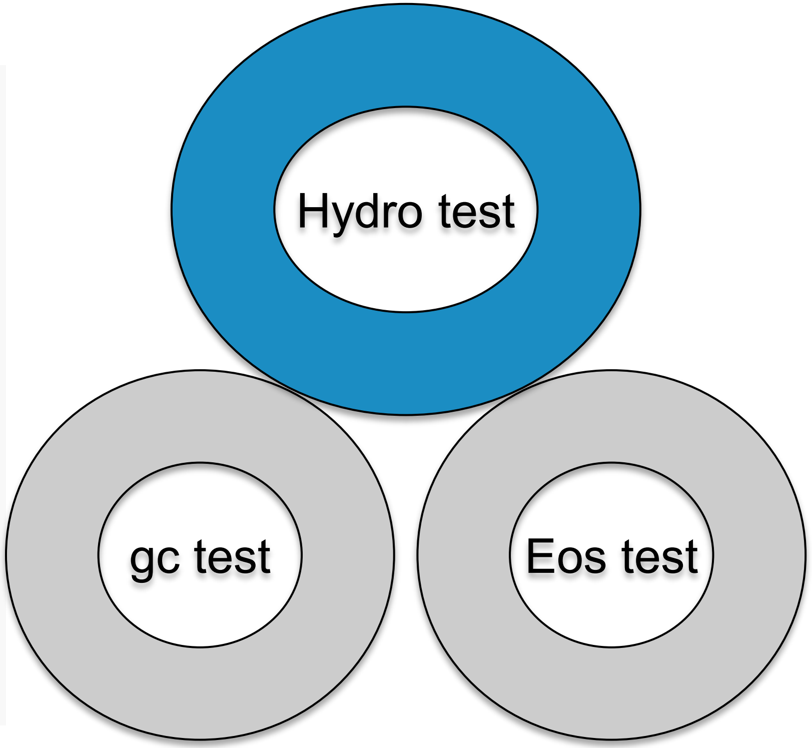

Next, build an EOS Test – is E(V,T) consistent with P(V,T)?

# 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

# 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

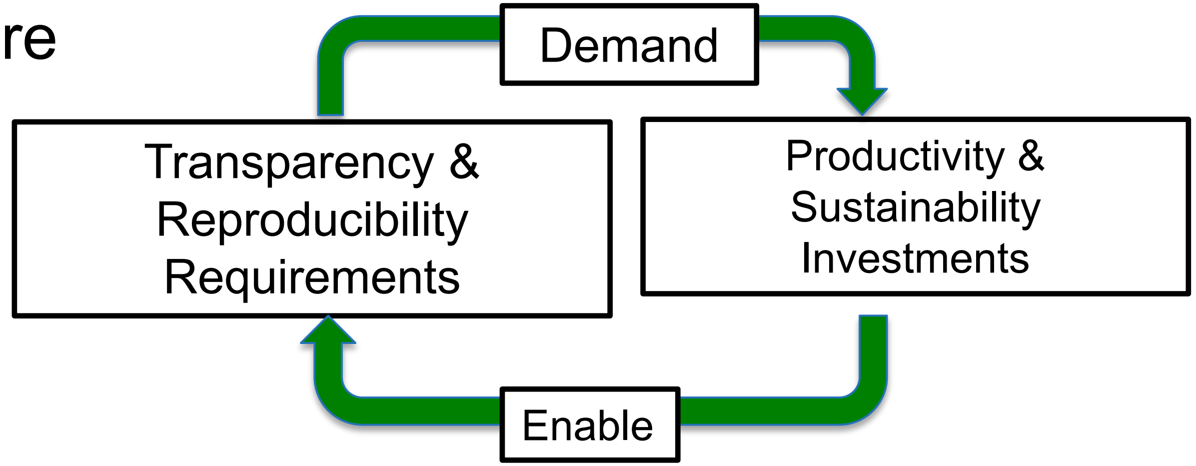
| | Hydro | EOS | Gravity | Burn | Particles |
|---|---|---|---|---|---|
| AMR | CL | CL | | CL | CL |
| UG | SV | SV | | | SV |
| Multigrid | WD | WD | WD | WD | |
| FFT | | | PT | | |

- 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!**)

# Incentives for Paying Attention to Reproducibility

Common statement: “I would love to do a better job on my software, but I need to..."

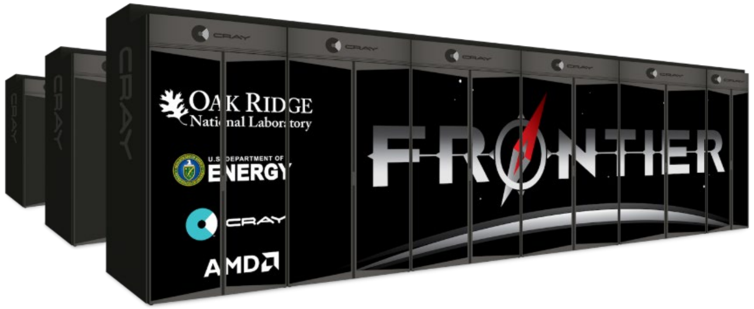
- Get this paper submitted
- Complete this project task
- Do something my employer values more




**Goal**: Change incentives to include valuing of better software, better science

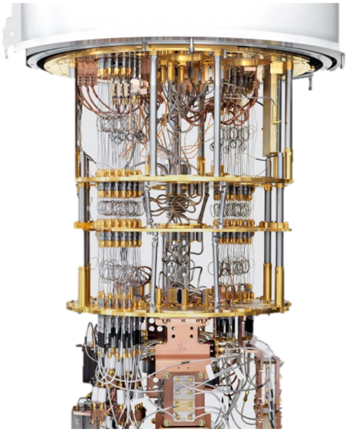
This is a long-term goal, requiring a culture change in (computational) science which in the early stages

# Additional Motivation – Testing Practices




- Supercomputer Cycles are Scarce Resources
  - Goal = capture QA details during science runs


- Many people need to have confidence in your results:
  - You
  - Your project lead or boss
  - Your sponsor
  - Your reviewers or referees
  - Your readers



- Testing helps build credibility *without* repeating runs.

# Strategies for Improving Reproducibility

- **Solid versioning practices are fundamental to reproducibility**
  - Source code, dependencies, copy through to outputs
- **Build in quality from the start**
  - Expectation for documentation and level of testing (write in-sync with code).
  - Increasing expectations as code becomes more "public"
  - Peer review / team productivity tracking meetings
- Watch Out for Numerical Artifacts
  - Integer overflows, floating point underflows, numerical algorithm stability
- Plan Ahead for Experimental Campaign Progression
  - How do scaling tests & intermediate outputs relate to code features?
  - What in-run correctness checks exist?
  - What if a patch needs to be applied during a campaign?

# Testing Strategies for Improving Reproducibility

- **Testing, testing, and more testing!**
- Add “regression tests”
  - If you fix a bug, add a test to make sure that bug doesn’t creep back in
- Add more tests
  - Be creative
  - Think about common cases, then corner cases
  - Think about misuse (unintentional or intentional)
  - Think about synthetic tests with synthetic data
  - Think about low-cost tests that can be “always on” (even if they’re not so stringent)
  - Can you detect silent data corruption?
- Physics / Math Based Strategies
  - Conserved quantities, symmetries, synthetic operators
- Design by Contract
  - Input / Output specifications, program invariants
- Test your third-party dependencies
  - Are your tools doing what you think they’re doing?
  - What if you’re using a new version?
  - How do you decide if it is okay to upgrade to a new version?
- Test your tests!
  - Make sure tests fail when they’re supposed to!
- Thoroughly verify the code
  - Does the code do what you intended it to do?
  - On all relevant platforms (compilers, hardware, etc.)
- Test regularly
  - To identify and document where changes to the underlying
    platform change code behavior/results

# 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 ?