GOOD SCIENTIFIC PROCESS
REQUIRES
SOFTWARE ENGINEERING PRACTICES

OBJECTIVES

- To bring knowledge of **useful** software engineering practices to HPC scientific code developers
  - Not to prescribe any set of practices as **must use**
    - Be informative about practices that have worked for some projects
    - Emphasis on adoption of practices that help productivity rather than put unsustainable burden
      - **Make it easier to get trusted science done**
    - Customization as needed – based on information made available
- Community codes are valuable and case studies
  - Even if you are not developing a community code, they are open examples of process
    - Lessons relevant across all domains

SOFTWARE ENGINEERING OF SCIENTIFIC APPLICATIONS

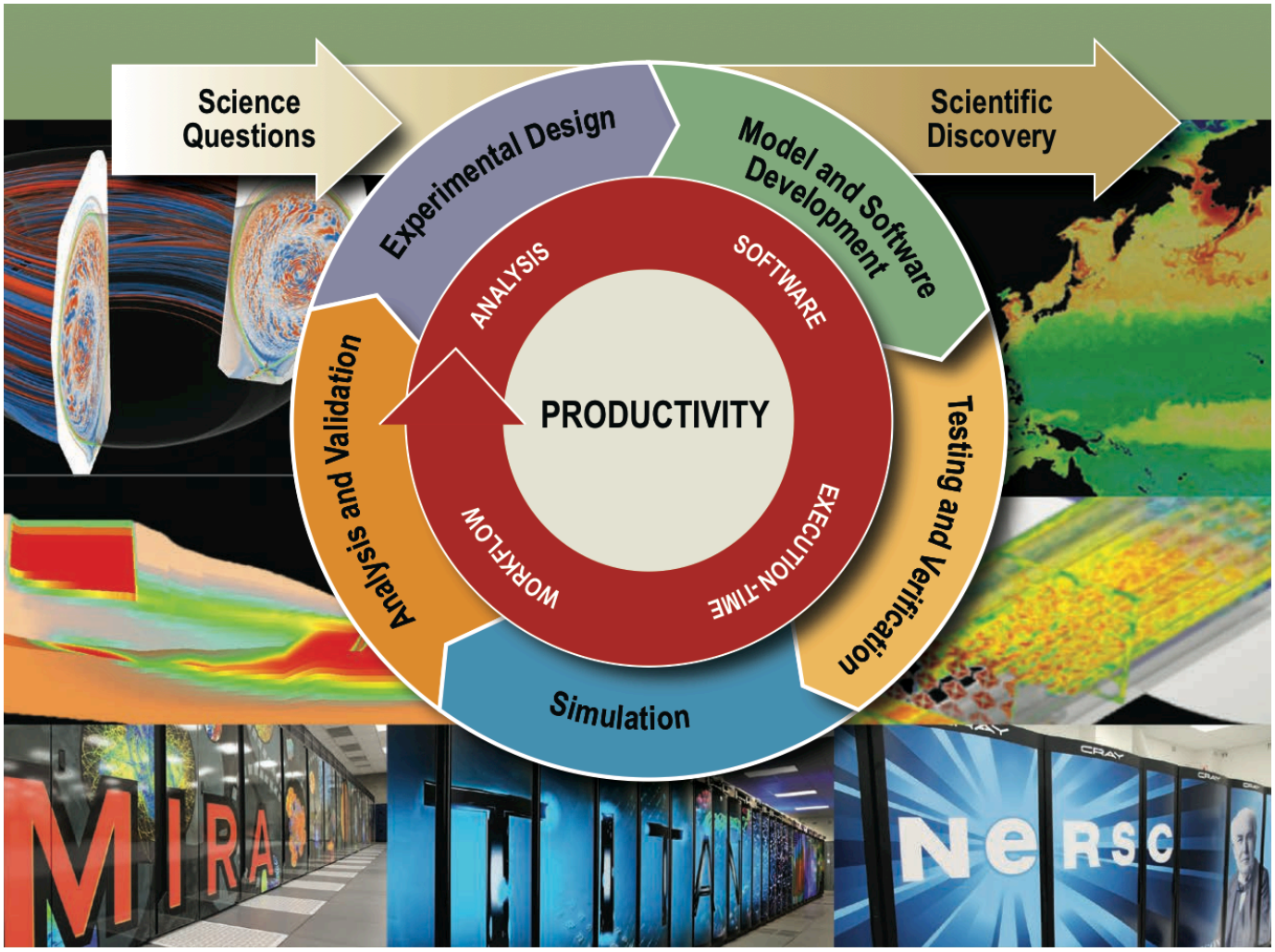
- Systematic approach to the application process
  - Design/Architecture
  - Development
  - Testing/Verification and Validation
  - Maintenance
  - Provenance

- Systematic
  - Just a fixed plan, system, method, etc
  - This plan can be of any scale

Engineering practices will be required for funding

SCIENTIFIC APPLICATIONS ARE COMPLEX

- Domain Problem
- Applied Mathematics
- Computer Science
- I/O
- Data
- Verification
- Validation
- Provenance
- Software Architecture & Engineering
- Performance
- Hardware awareness



Using the largest computer systems pushes the boundaries of all of these

HEROIC PROGRAMMING

Usually a pejorative term, is used to describe the expenditure of huge amounts of (coding) effort by talented people to overcome shortcomings in process, project management, scheduling, architecture or any other shortfalls in the execution of a software development project in order to complete it. Heroic Programming is often the only course of action left when poor planning, insufficient funds, and impractical schedules leave a project stranded and unlikely to complete successfully.

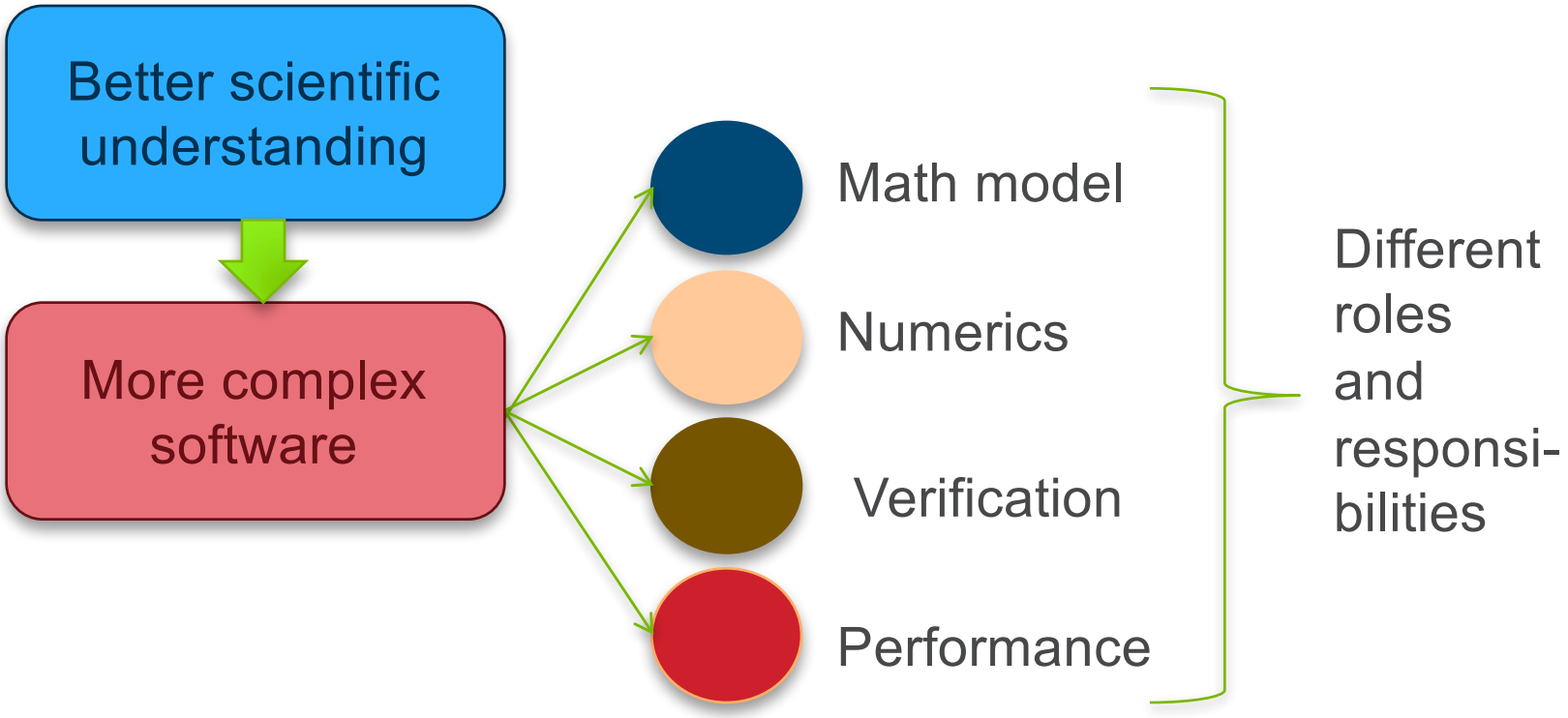
From http://c2.com/cgi/wiki?HeroicProgramming

Science teams often resemble heroic programming

Many do not see anything wrong with that approach

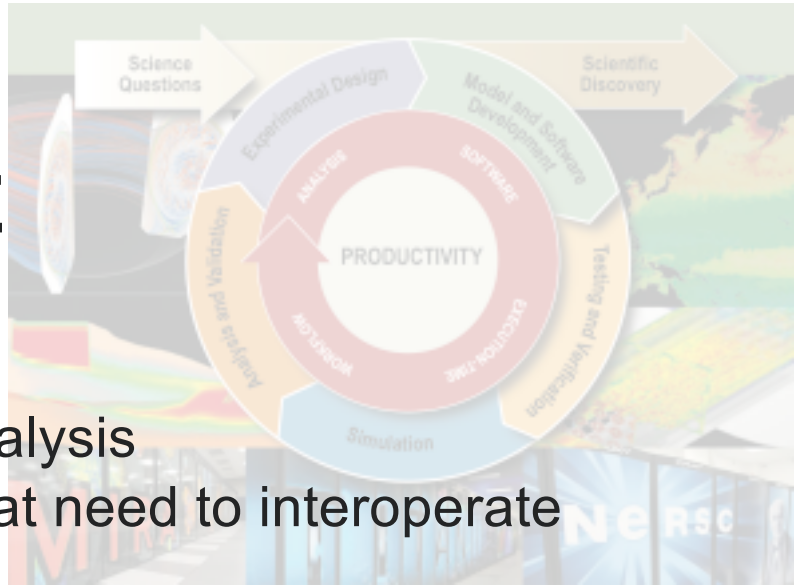
WHAT IS WRONG WITH HEROIC PROGRAMMING

Scientific results that could be obtained with heroic programming have run their course, because:



It is not possible for a single person to take on all these roles

IN EXTREME-SCALE SCIENCE



- Codes aiming for higher fidelity modeling
  - More complex codes, simulations and analysis
  - Numerous models, more moving parts that need to interoperate
  - Larger cost of the work
  - Variety of expertise needed – the only tractable development model is through separation of concerns
  - **It is more difficult to work on the same software in different roles without a software engineering process**

- Onset of higher platform heterogeneity
  - Requirements are unfolding, not known apriori
  - **The only safeguard is investing in flexible design and robust software engineering process**

OTHER REASONS

Accretion leads to unmanageable software

- Increases cost of maintenance
- Parts of software may become unusable over time
- Inadequately verified software produces questionable results
- Increases ramp-on time for new developers
- Reduces software and science productivity due to technical debt

Consequence of Choices

Quick and dirty incurs technical debt, collects interest which means more effort required to add features.

TECHNICAL DEBT

The long term consequences of process

- “By deferring issues such as code readability and maintainability, a debt is created that someone in the future might have to pay, in the extra effort needed to re-run or modify the code. Debt is not bad *per se*. After all, we frequently incur debt to obtain something of immediate value, for example, using a mortgage to buy a house. The point is that such debts have to be managed carefully, to prevent them spiraling out of control.”

http://dx.doi.org/10.1038/ngeo2283

“Open code for open science?”

Steve M. Easterbrook

Nature Geoscience

COLLABORATION IS HARD WITHOUT PROCESS

- Modern scientific computing is no longer a solo effort
  - Should not be a solo effort
  - Most interesting modeling questions that could be simulated by the heroic individual programming scientist have already been investigated
  - “Productivity languages” have not delivered yet
  - Coding is complicated and requires division of roles and responsibilities.
- Working on a common code is difficult unless there is a software process
- Even if solo
  - Code will live longer than expected
  - You need to trust results

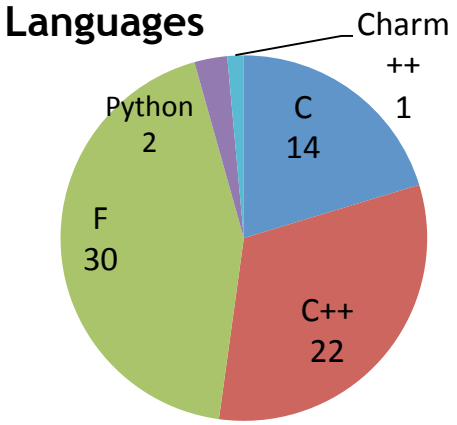
CONSIDER THE HPC ECOSYSTEM

SCIENTIFIC SOFTWARE IS DIFFERENT

- Developing code exclusively for a small cluster is not the same as developing code for HPC
- You ***can*** develop HPC code that will work well on your cluster and your laptop

- In HPC, the trade-off with design and performance is omnipresent
- Have reached a complexity point that code reuse & design is very important

Quick glimpse of some stats on Mira applications.
100% MPI, 65% threaded



Code Availability	
Open	52%
Closed	26%
Fuzzy	22%

MANY CHOICES FOR CODES AND TRADE-OFFS

	Speed to science	Speed of Change	Features	Control Methods & Accuracy	Complexity of use	Validation	Verification
Blackbox user	Fast	Slow	??	None	Depends	High	High
Alter existing code base	Med	Fast	Depends	Partial	Med to High	Med/ Low	Med/ Low
Use of libraries	Dep.	Med	High	Partial to Low	Med	Med	Med
Use of framework	Med	Med	High	Partial to High	High	Shared	Shared
Development of new code	Slow	Slow	Low	High	Depends	All you	All you

Assuming best case scenarios

Software Engineering and HPC
Efficiency vs. Other Quality Metrics

How focusing on the factor below affects the factor to the right	Correctness	Usability	Efficiency	Reliability	Integrity	Adaptability	Accuracy	Robustness
Correctness	↑		↑	↑			↑	↓
Usability		↑				↑	↑	
Efficiency	↓		↑	↓	↓	↓	↓	
Reliability	↑			↑	↑		↑	↓
Integrity			↓	↑	↑			
Adaptability					↓	↑		↑
Accuracy	↑		↓	↑		↓	↑	↓
Robustness	↓	↑	↓	↓	↓	↑	↓	↑

Source:
Code Complete
Steve McConnell

Helps it ↑
Hurts it ↓

Sandia National Laboratories

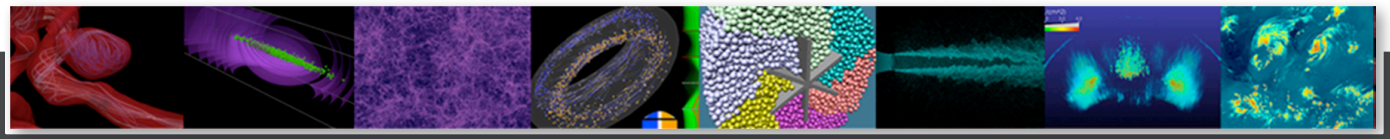
GOOD SCIENTIFIC PROCESS
REQUIRES
GOOD SOFTWARE ENGINEERING

Community Codes and Software Engineering – Track 4

MONDAY, August 8

Time	Title of presentation	Lecturer
9:30 am	**Introduction to the Session – Software Engineering**	Anshu Dubey, ANL
9:35 am	**Overview and Objectives**	Katherine Riley and Anshu Dubey, ANL
10:30 am	*Break*	
11:00 am	**Repo, Continuous Integration**	Jeff Johnson, LBNL
12:00 pm	*Lunch and Hands-on Exercises*	
1:00 pm	**IDE/Config/Build/Deploy**	Barry Smith, ANL
2:00 pm	**Documentation**	Alicia Klinvex, SNL
2:30 pm	*Break*	
3:00 pm	**Testing**	Alicia Klinvex, SNL
3:45 pm	**Software Refactoring**	Anshu Dubey, ANL
4:30 pm	**Putting it All Together: Example**	Glenn Hammond, SNL

+ Hands-on



OBJECTIVES

- To bring knowledge of **useful** software engineering practices to HPC scientific code developers
  - Not to prescribe any set of practices as **must use**
    - Be informative about practices that have worked for some projects
    - Emphasis on adoption of practices that help productivity rather than put unsustainable burden
      - **Make it easier to get trusted science done**
    - Customization as needed – based on information made available
- Community codes are valuable and case studies
  - Even if you are not developing a community code, they are open examples of process
    - Lessons relevant across all domains

STRIVE FOR A BETTER PROCESS