



# Overview of Best Practices in HPC Software Development



David E. Bernholdt  
Oak Ridge National Laboratory

Anshu Dubey, Katherine M. Riley  
Argonne National Laboratory

Better Scientific Software Tutorial  
RF SciDAC 2020 Workshop



See slide 2 for  
license details

# License, Citation and Acknowledgements



## License and Citation

- This work is licensed under a [Creative Commons Attribution 4.0 International License](https://creativecommons.org/licenses/by/4.0/) (CC BY 4.0).
- **The requested citation the overall tutorial is: David E. Bernholdt, Better Scientific Software tutorial, in RF SciDAC 2020 Workshop, Knoxville, Tennessee. DOI: [10.6084/m9.figshare.11918397](https://doi.org/10.6084/m9.figshare.11918397)**
- Individual modules may be cited as *Speaker, Module Title*, in Better Scientific Software Tutorial...

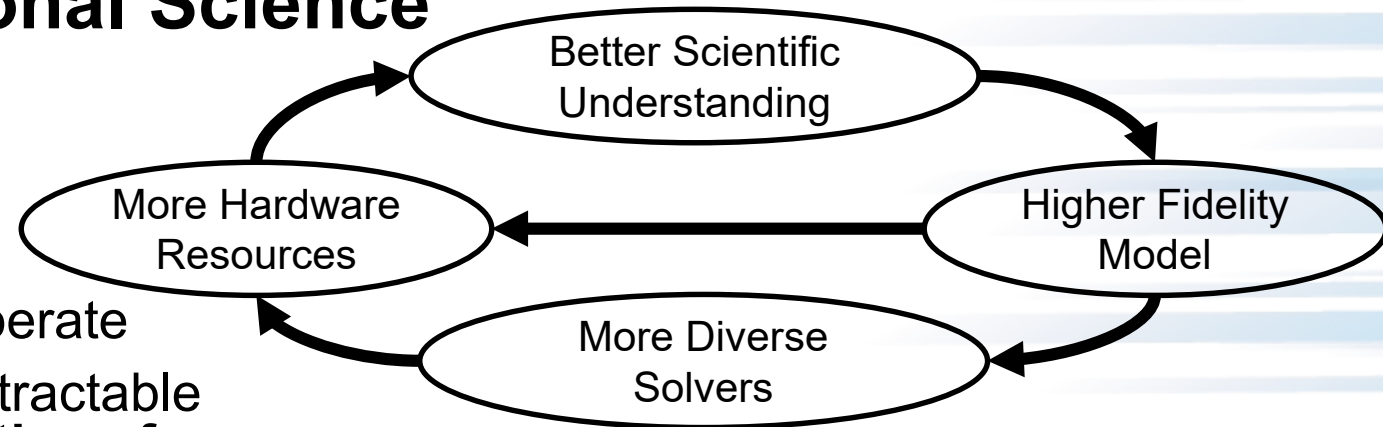
## Acknowledgements

- Additional contributors to this this tutorial include: Anshu Dubey, Mike Heroux, Alicia Klinvex, Jared O'Neal, and Katherine Riley, James M. Willenbring
- 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 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 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. SAND NO SAND2017-5474 PE

# The Success of Computational Science Creates the Challenges of Computational Science

- Positive feedback loop

- More complex codes, simulations and analysis
- More moving parts that need to interoperate
- 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**

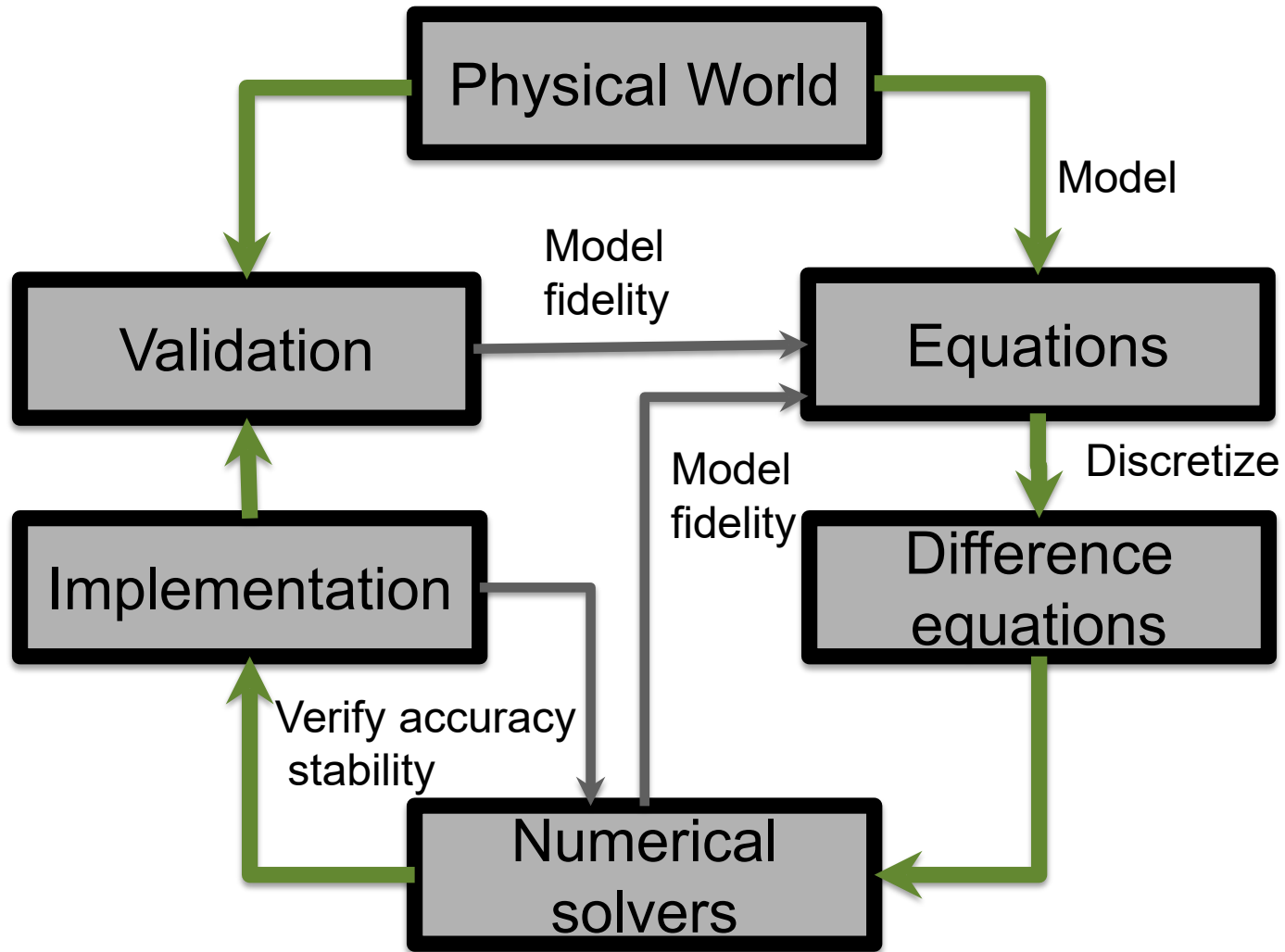


- Onset of higher platform heterogeneity

- Requirements are unfolding, not known *a priori*
- **The only safeguard is investing in flexible design and robust software engineering process**

Supercomputers change fast  
Especially now!

# Lifecycle of a Scientific Application



- Modeling

- Approximations
- Discretizations
- Numerics
  - Convergence
  - Stability

- Implementation

- Verification
  - Expected behavior
- Validation
  - Experiment/observation

# Challenges Developing a Scientific Application

## Technical

- All parts of the cycle can be under research
- Requirements change throughout the lifecycle as knowledge grows
- Verification complicated by floating point representation
- Real world is messy, so is the software

## Sociological

- Competing priorities and incentives
- Limited resources
- Perception of overhead without benefit
- Need for interdisciplinary interactions

# Heroic Programming

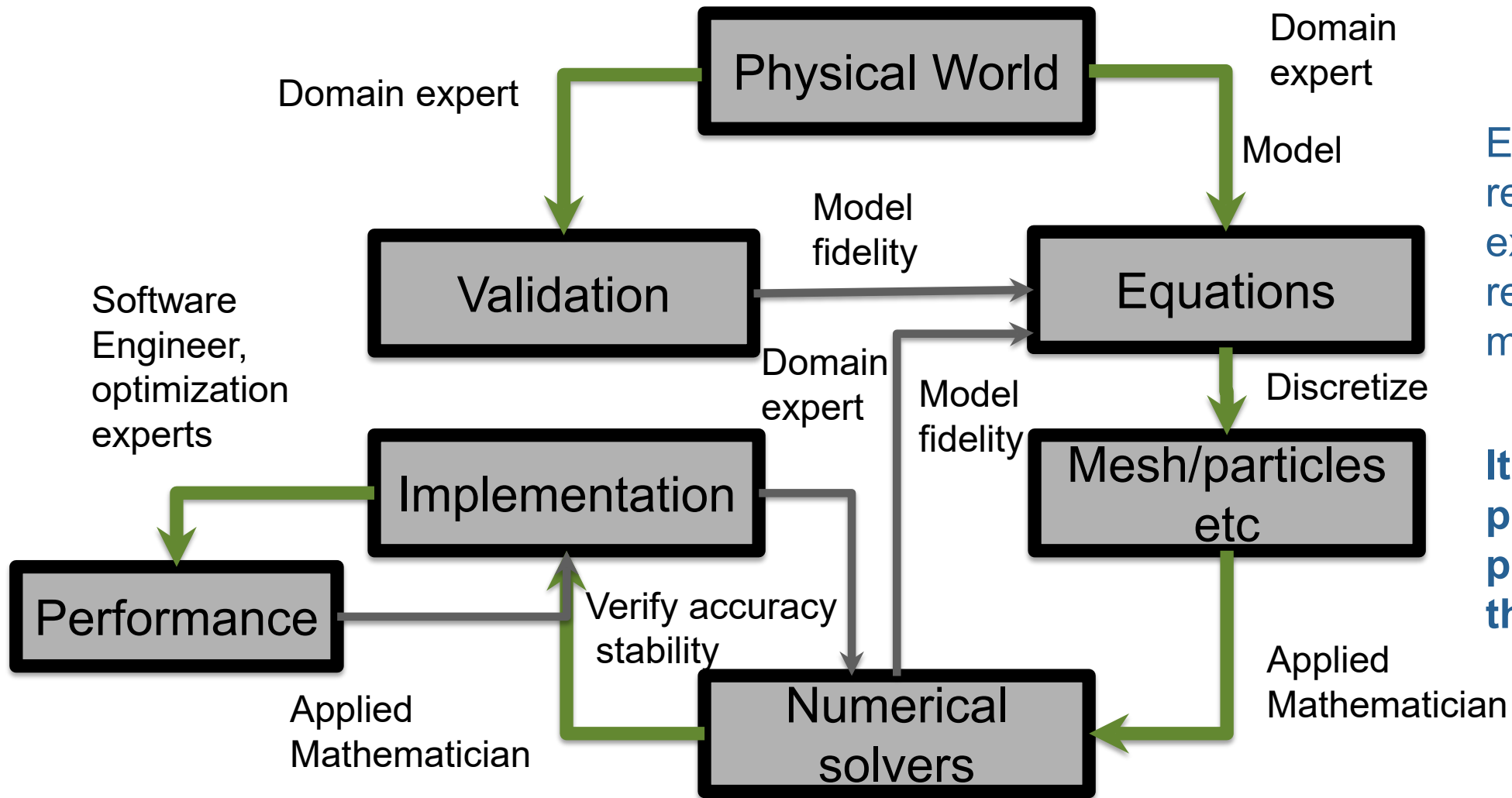
Usually a pejorative term, is used to describe the expenditure of huge amounts of (coding) effort by talented people to overcome shortcomings in process, project management, scheduling, architecture or any other shortfalls in the execution of a software development project in order to complete it. Heroic 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 employ heroic programming**

Many do not see anything wrong with that approach

# Expertise Map



Each of these roles require deeper expertise as scientific requirements grow more complex.

**It is no longer possible for a single person to take on all these roles**

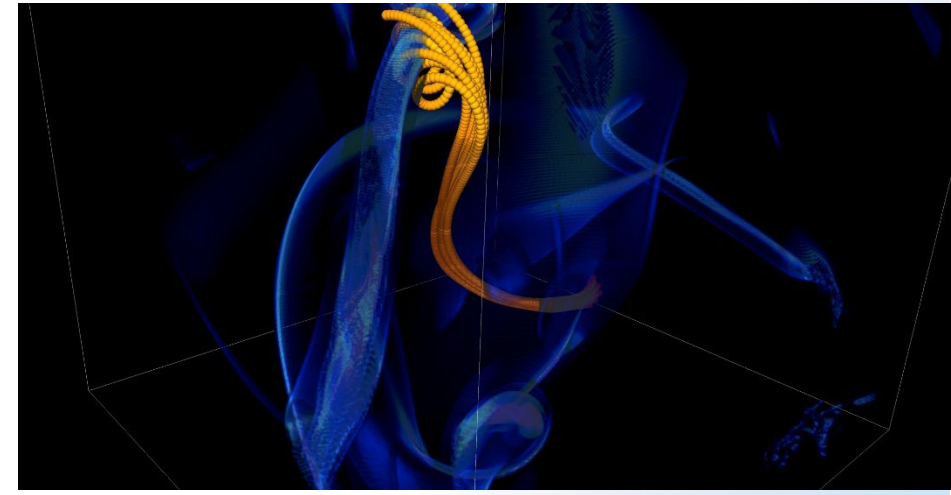
**Good scientific process  
requires  
good software practices**

**Good software practices  
increase  
scientific productivity**



# You Can Mitigate Risk, But It Is Never Zero

- Short notice availability of one of the biggest machines of it's time
  - **< 1month to get ready, run was 1.5 weeks**
- Quick and dirty development of particle capability in code
- Error in tracking particles resulted in duplicated tags from round-off
- Had to develop post-processing tools to correctly identify trajectories
  - **6 months to process results**



FLASH had a software process in place. It was tested regularly. This was one instance when the full process could not be applied because of time constraints.

# Why Be Concerned with Software Engineering?

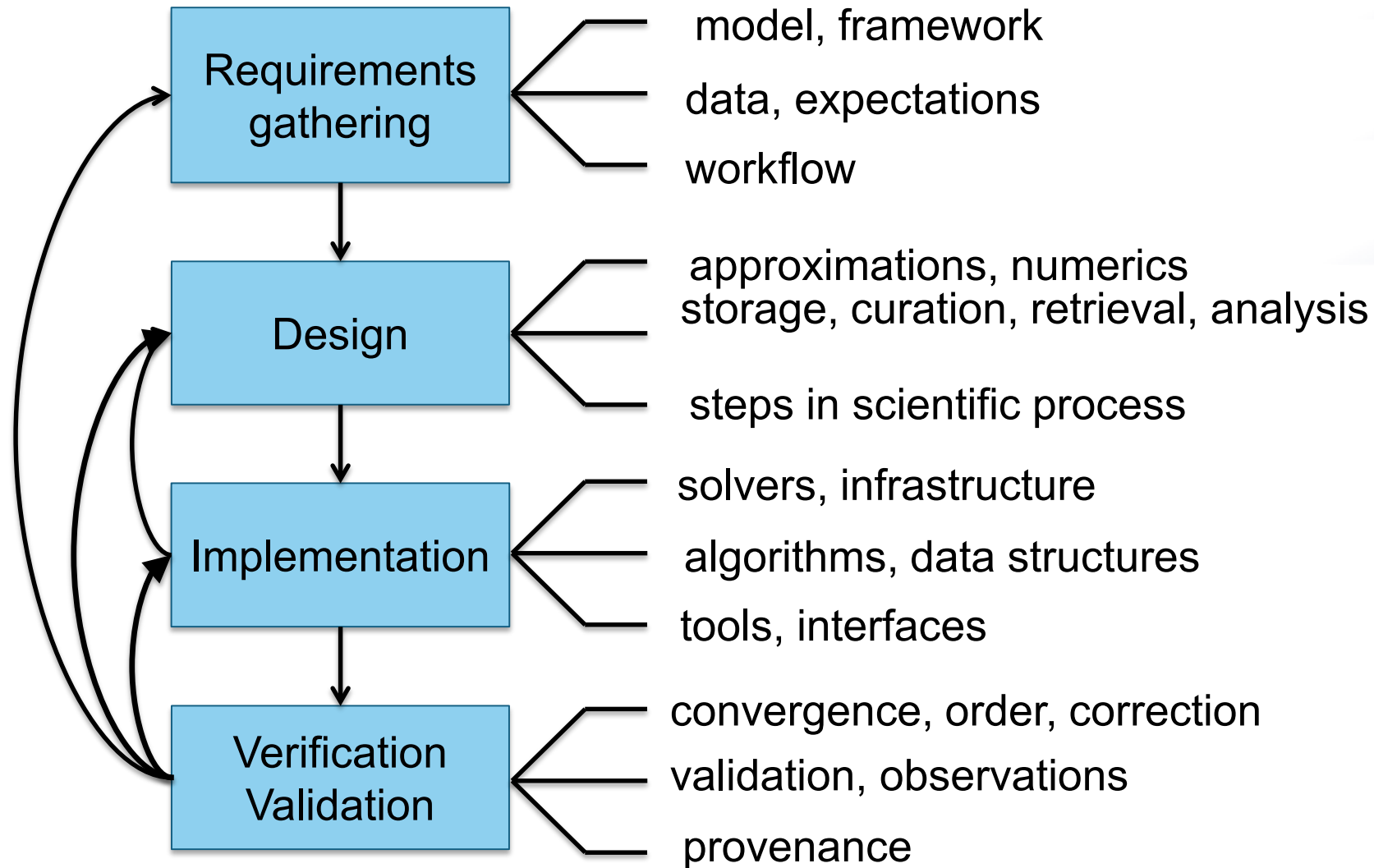
Accretion leads to unmanageable software

- 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**

## Consequences of Choices

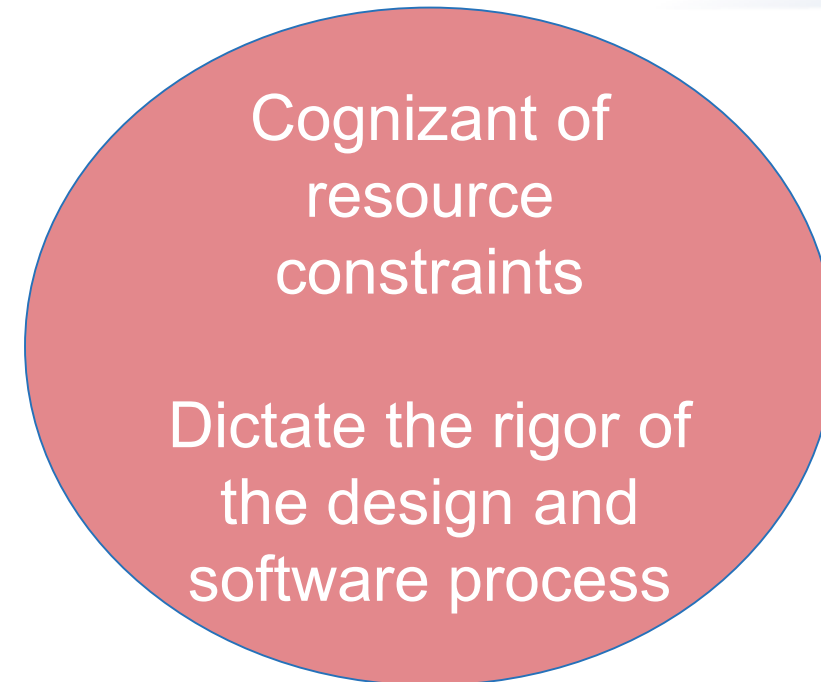
“Quick and dirty” collects technical debt, which means more effort required to add features.

# Lifecycle: Software Engineering View



# Taking Stock of *Your* Situation

- Software architecture and process design is an overhead
  - Value lies in avoiding technical debt (future saving)
  - Worthwhile to understand the trade-off
- The goals of the software
  - Proof-of-concept
  - Verification
  - Exploration of some phenomenon
  - Experiment design
  - Analysis
  - Other ...

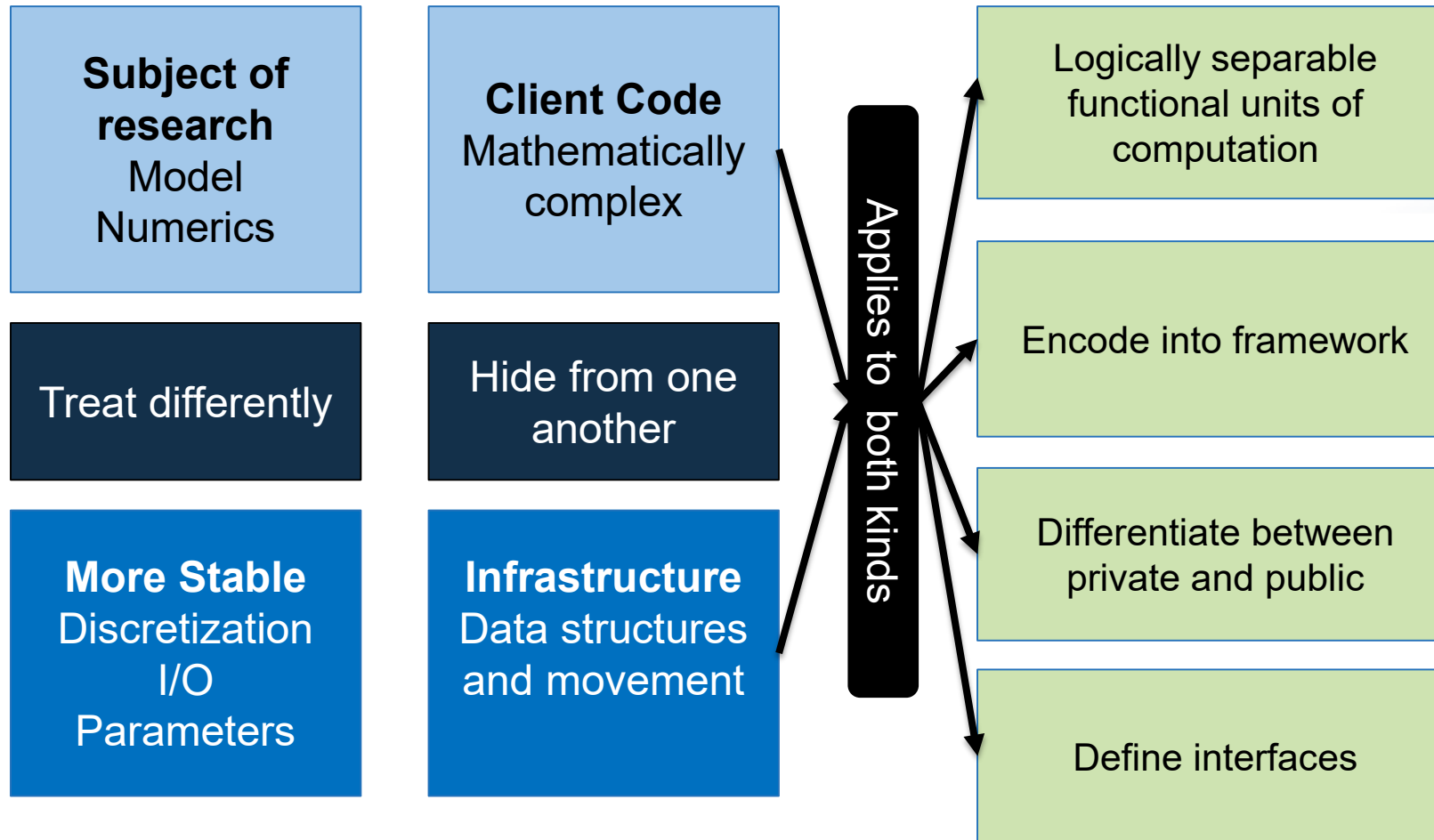


# Reconcile Conflicting Requirements

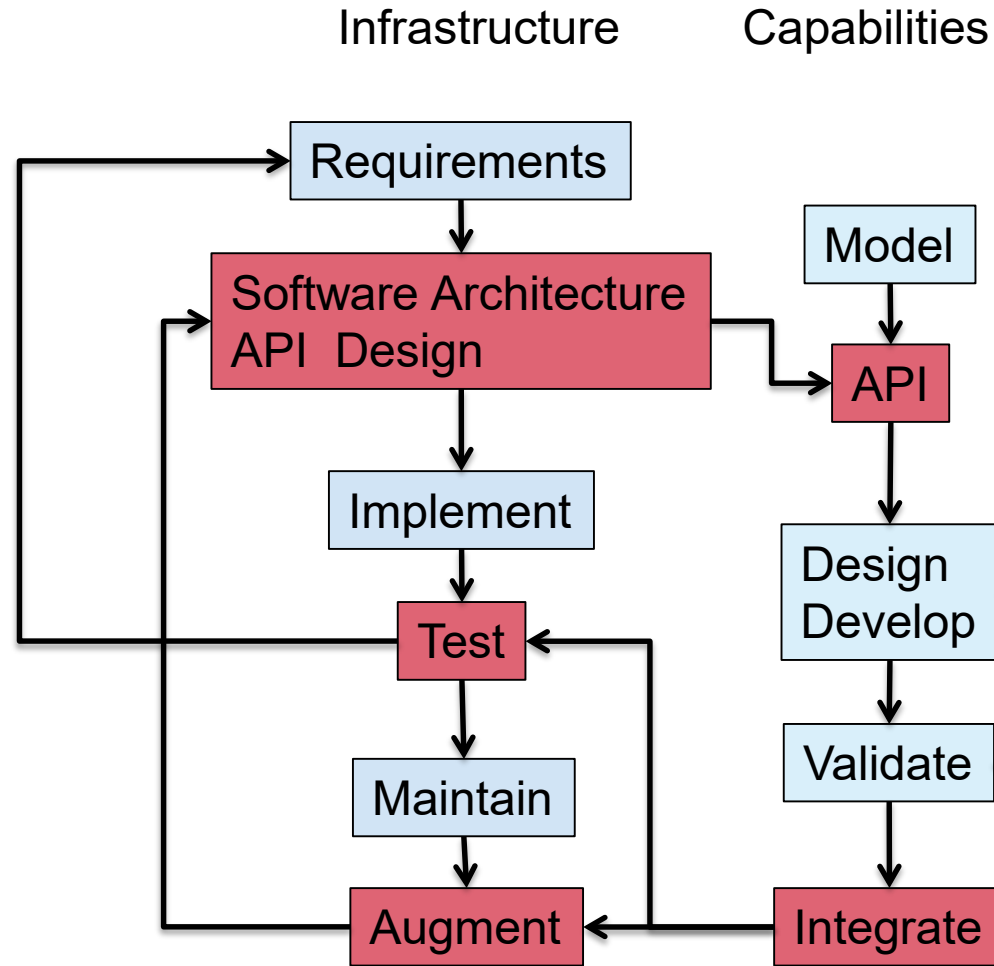
- Separation of concerns
  - Encapsulation of functionalities where possible
  - Abstractions for encapsulations
    - Offload complexity where possible
- Hard-nosed trade-offs
  - Flexibility and composability vs raw performance
  - Extensibility and developer productivity

# Architecting Scientific Codes

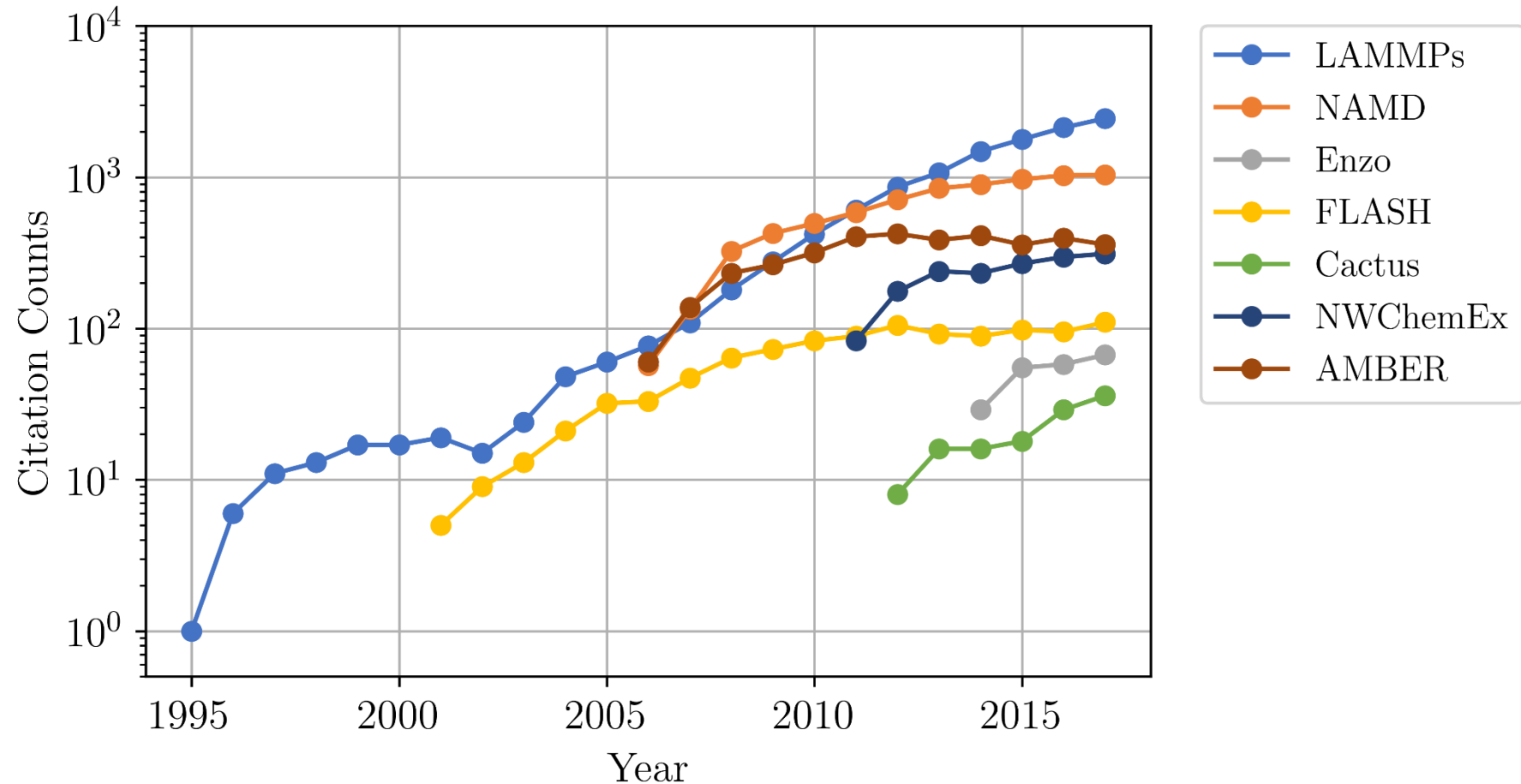
## Taming the Complexity: Separation of Concerns



# A Successful Model



# Community Impact of Well Done Software





# Software Process Best Practices

## Baseline

- Invest in extensible code design
- Use version control and automated testing
- Institute a rigorous verification and validation regime
- Define coding and testing standards
- Clear and well defined policies for
  - Auditing and maintenance
  - Distribution and contribution
  - Documentation

## Desirable

- Provenance and reproducibility
- Lifecycle management
- Open development and frequent releases

# A Useful Resource

<https://ideas-productivity.org/resources/howtos/>

- **‘What Is’ docs:** 2-page characterizations of important topics for SW projects in computational science & engineering (CSE)
- **‘How To’ docs:** brief sketch of best practices
  - Emphasis on “bite-sized” topics enables CSE software teams to consider improvements at a small but impactful scale
- We welcome feedback from the community to help make these documents more useful

# Other Resources

<http://www.software.ac.uk/>

<http://software-carpentry.org/>

<http://flash.uchicago.edu/cc2012/>

<http://journals.plos.org/plosbiology/article?id=10.1371/journal.pbio.1001745>

<http://ieeexplore.ieee.org/xpls/icp.jsp?arnumber=4375255>

<http://www.orau.gov/swproductivity2014/SoftwareProductivityWorkshopReport2014.pdf>

<http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6171147>

# Summary

- Good software practices are needed for scientific productivity
- Science at extreme-scales is complex and requires multiple expertise
- Software process does need to address reality
- Open codes, community contribution, are a powerful tool

Science through computing is  
*at best*  
as credible as the software that produces it

# Agenda

Time	Module	Topic	Speaker
1:00pm-1:05pm	00	Introduction	David E. Bernholdt, ORNL
1:05pm-1:30pm	01	Overview of Best Practices in HPC Software Development	David E. Bernholdt, ORNL
1:30pm-2:00pm	02	Agile Methodologies and Useful GitHub Tools	David E. Bernholdt, ORNL
2:00pm-2:30pm	03	Improving Reproducibility through Better Software Practices	David E. Bernholdt, ORNL
2:30pm-2:45pm		Q&A	All
<i>2:45pm-3:30pm</i>		<i>Break</i>	
3:30pm-4:15pm	04	Software Design and Testing	David E. Bernholdt, ORNL
4:14pm-4:45pm	05	Continuous Integration	David E. Bernholdt, ORNL
4:45pm-5:00pm		Q&A	All