Scientific Software Ecosystems: Why and How

Blue Waters Webinar Series
https://bluewaters.ncsa.illinois.edu/webinars
November 8, 2017

Lois Curfman McInnes
MCS Division
Argonne National Laboratory

Michael Heroux
Sandia National Laboratories
St. Johns University

New webinar track: Scientific Software Ecosystems

Objectives:

- Promote quality reusable research software for computational and data-enabled discovery
- Promote community efforts to improve research software quality, culture, credit, collaboration, …

While considering issues in scientific software ecosystems

Target participants: Entire CSE/HPC community

- Applications scientists, developers of reusable software (math/CS/apps), stakeholders

Why is reusable scientific software important for you?

User perspective:

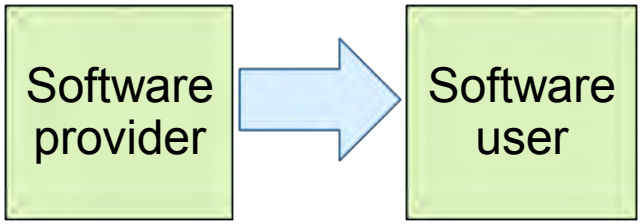
Focus on primary interests

- Reuse algorithms and data structures developed by experts
- Customize and extend to exploit application-specific knowledge
- Cope with complexity and changes over time

Provider perspective:

Share your capabilities

- Broader impact of your work
- Motivate new directions of research



- **More efficient, robust, reliable, sustainable software**
- **Improve developer productivity**
- **Better science**

Outline

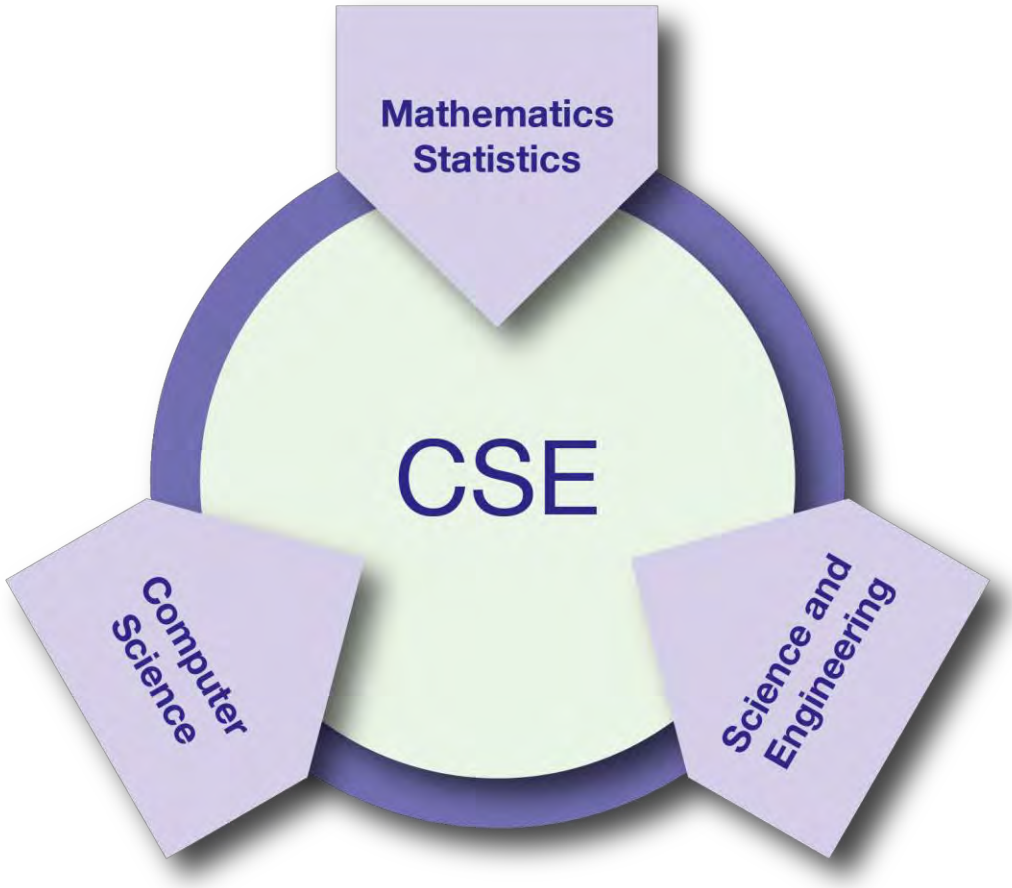
- Motivation
- Libraries: Reusable research software
- Managing risks of external software use
- Toward scientific software ecosystems
  - xSDK
- International community efforts
- Get involved!

Acknowledgments

- **Collaborators**
  - IDEAS Scientific Software Productivity Project
  - Developers and users of PETSc and Trilinos
  - DOE Exascale Computing Project
  - SIAM Activity Group on Computational Science and Engineering
- **U.S. Department of Energy**

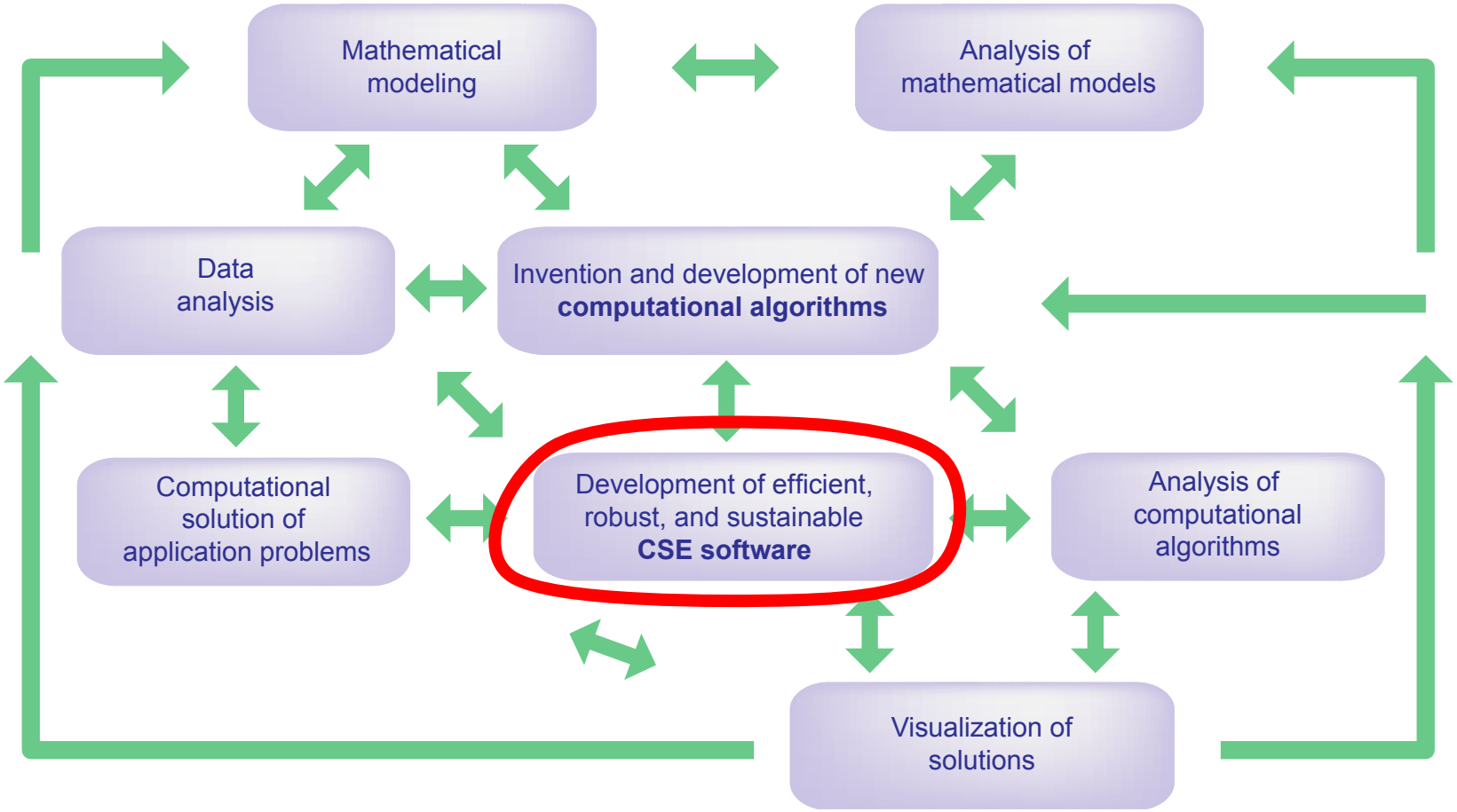
What is CSE?

- **Computational Science & Engineering (CSE):** development and use of computational methods for scientific discovery
  - all branches of the sciences, engineering and technology
  - support decision-making across a spectrum of societally important apps
- **CSE: essential driver of scientific and technological progress**



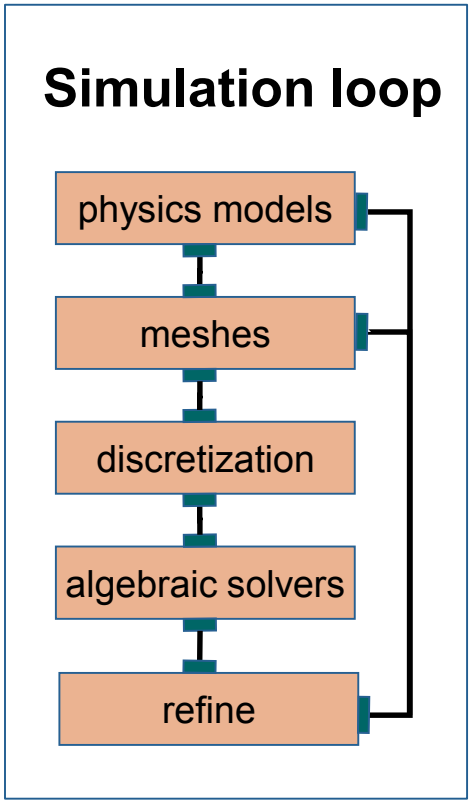
Reference: Research and Education in Computational Science & Engineering, U. Rüde, K. Willcox, L.C. McInnes, H. De Sterck, et al., Oct 2016, **https://arxiv.org/abs/1610.02608**

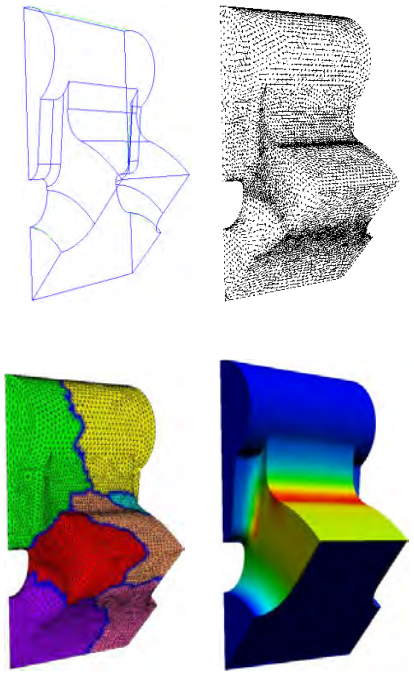
Software: Foundation of sustained CSE collaboration and scientific progress



CSE simulation relies on high-performance numerical algorithms and software

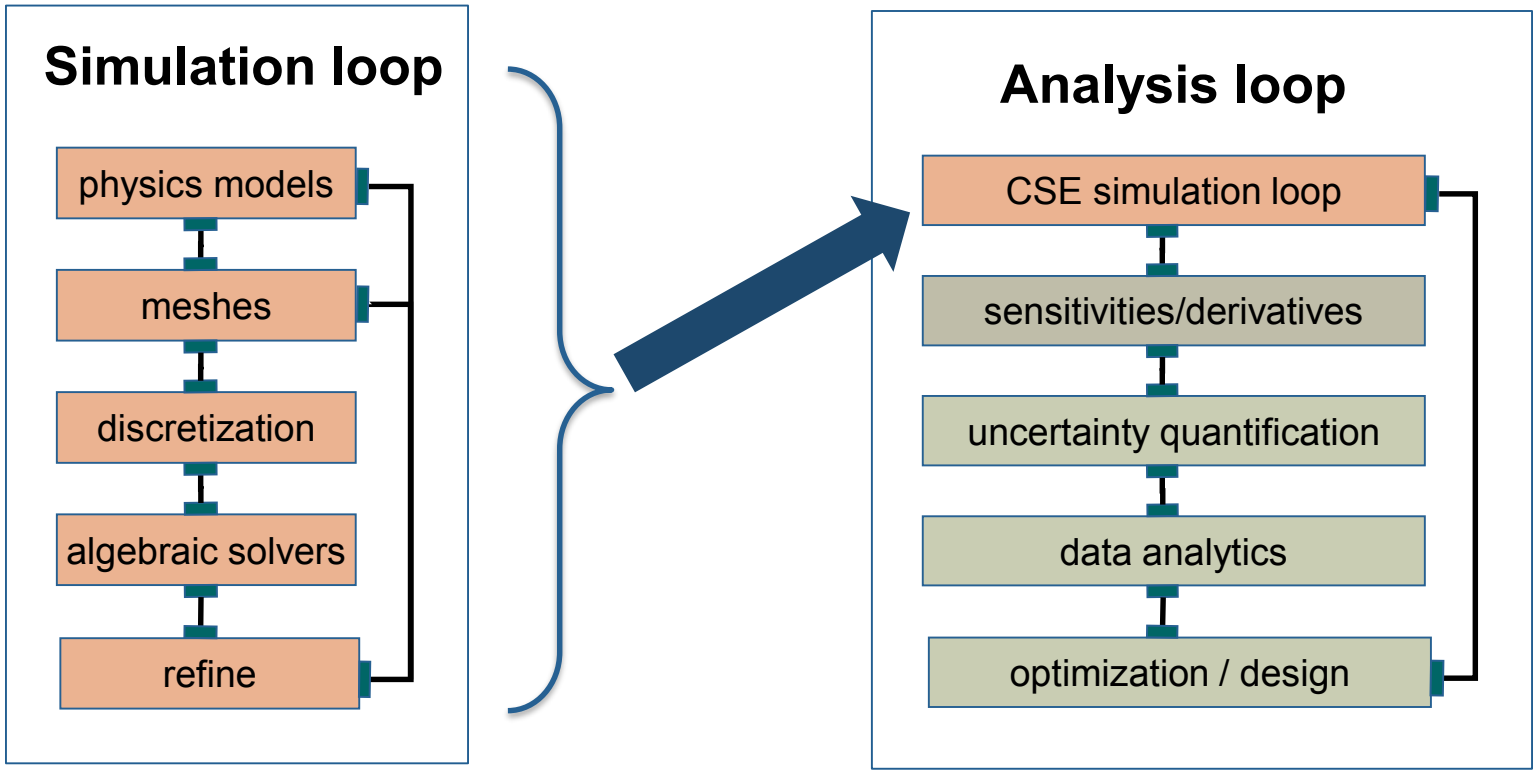
- Develop a mathematical model of the phenomenon of interest
- Approximate the model using a discrete representation
- Solve the discrete representation
- Adapt and refine the mesh or model
- Incorporate different physics, scales





These steps require: mesh generation, partitioning, load balancing, high-order discretization, time integration, linear and nonlinear solvers, eigensolvers, mesh refinement, multiscale/multiphysics coupling methods, etc.

CSE analysis builds on the CSE simulation loop … and relies on even more numerical algorithms and software



These steps require: adjoints, sensitivities, algorithmic differentiation, sampling, ensemble simulations, uncertainty quantification, data analytics, optimization (derivative free and derivative based), inverse problems, etc.

Trends and challenges for CSE software

ALCF early production system

- **Fundamental trends:**
  - Disruptive hardware changes
    - Require algorithm/code refactoring
  - Need coupling, optimization, sensitivities
    - Multiphysics, multiscale, data analytics
- **Challenges:**
  - Need refactoring: Really, continuous change
  - Modest funding for app development: No monolithic apps
  - Requirements are unfolding, evolving, not fully known *a priori*
- **Opportunities:**
  - Better design, software practices, and tools are available
  - Better software architectures: toolkits, libraries, frameworks
  - Open-source software, community collaboration

Outline

- Motivation
- Libraries: Reusable research software
- Managing risks of external software use
- Toward scientific software ecosystems
  - xSDK
- International community efforts
- Get involved!

Software libraries facilitate CSE progress

- **Software library:** a high-quality, encapsulated, documented, tested, and multiuse software collection that provides functionality commonly needed by application developers
  - Organized for the purpose of being reused by independent (sub)programs
  - User needs to know only
    - Library interface (not internal details)
    - When and how to use library functionality appropriately
- **Key advantages** of software libraries
  - Contain complexity
  - Leverage library developer expertise
  - Reduce application coding effort
  - Encourage sharing of code, ease distribution of code
- References:
  - https://en.wikipedia.org/wiki/Library_(computing)
  - What are Interoperable Software Libraries? Introducing the xSDK

WHY USE LIBRARIES

- A farmer had chickens and pigs. There was a total of 60 heads and 200 feet. How many chickens and how many pigs did the farmer have?
- Let x be the number of chickens, y be the number of pigs.
- Then:

$$\begin{aligned} x + y &= 60 \\ 2x + 4y &= 200 \end{aligned}$$

- From first equation $x = 60 - y$, so replace x in second equation:

$$2(60 - y) + 4y = 200$$

- Solve for y:

$$\begin{aligned} 120 - 2y + 4y &= 200 \\ 2y &= 80 \\ y &= 40 \end{aligned}$$

- Solve for x: $x = 60 - 40 = 20.$
- The farmer has 20 chickens and 40 pigs.

- A restaurant owner purchased one box of frozen chicken and another box of frozen pork for $60. Later the owner purchased 2 boxes of chicken and 4 boxes of pork for $200. What is the cost of a box of frozen chicken and a box of frozen pork?
- Let x be the price of a box of chicken, y the price of a box of pork.
- Then:

$$\begin{aligned} x + y &= 60 \\ 2x + 4y &= 200 \end{aligned}$$

- From first equation $x = 60 - y$, so replace x in second equation:

$$2(60 - y) + 4y = 200$$

- Solve for y:

$$\begin{aligned} 120 - 2y + 4y &= 200 \\ 2y &= 80 \\ y &= 40 \end{aligned}$$

- Solve for x: $x = 60 - 40 = 20.$
- A box of chicken costs $20 and a box of pork costs $40.

Problem Statement

- A restaurant owner purchased one box of frozen chicken and another box of frozen pork for $60. Later the owner purchased 2 boxes of chicken and 4 boxes of pork for $200. What is the cost of a box of frozen chicken and a box of frozen pork?

- Let x be the price of a box of chicken, y the price of a box of pork. Variables

- Then: Problem Setup

$$x + y = 60$$
$$2x + 4y = 200$$

- From first equation $x = 60 - y$, so replace x in second equation: Solution Method

$$2(60 - y) + 4y = 200$$

- Solve for y:

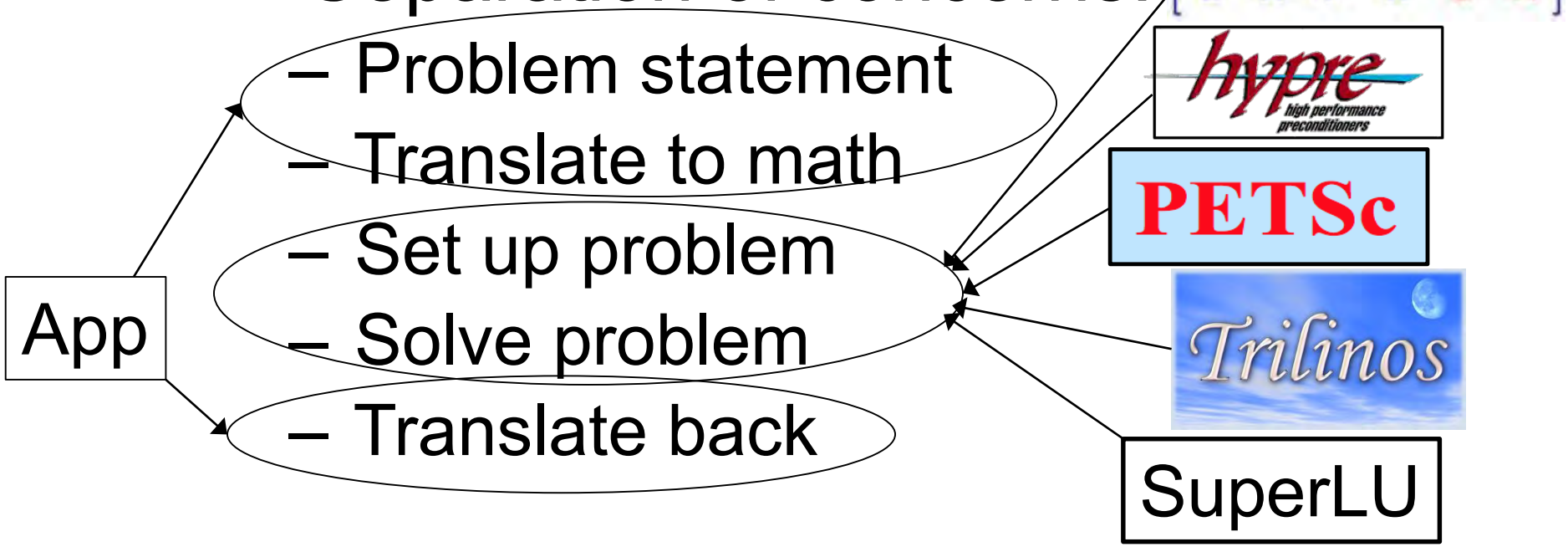
$$120 - 2y + 4y = 200$$
$$2y = 80$$
$$y = 40$$

- Solve for x: $x = 60 - 40 = 20$.

- A box of chicken costs $20. A box of pork costs $40. Translate Back

Why reusable libraries?

- Many types of problems
- Similar algorithms
- Separation of concerns:
  - Problem statement
  - Translate to math
  - Set up problem
  - Solve problem
  - Translate back



Importance of math libraries

- Computer solution of math problems is hard:
  - Floating point arithmetic not exact:
    - $1 + \varepsilon = 1$, for small $\varepsilon > 0$
    - $(a + b) + c$ not always equal to $a + (b + c)$
  - High fidelity leads to large problems: 1M to 10B equations
  - Clusters require coordinated solution across 100 – 1M processors
- Sophisticated solution algorithms and libraries leveraged:
  - Solver expertise highly specialized, expensive
  - Write code once, use in many settings

WHAT YOU SHOULD KNOW THAT WE WON'T DISCUSS

Many important tools and libraries

- Performance Tools: PAPI, HWLOC
- Array Libraries: Global Arrays, Kokkos, RAJA
- I/O: SCR, MPI_IO, ADIOS, ALPINE, VTK
- Many more

- Our focus: Math libraries as part of the ecosystem

BLAS and LAPACK

- **BLAS:** vector-vector, matrix-vector, matrix-matrix functions
  - High-performance (vendor optimized)
  - You are probably using BLAS, even if you don't know it.
- **LAPACK:**
  - Large collection of linear algebra functions
  - Fortran and C APIs
  - Shared memory parallel (mostly through parallel BLAS)
  - Dense, banded, tridiagonal
  - Real and complex, float and double
- **LAPACK linear solvers:** Many dense linear solvers:
  - DGESV: Single function call to solve for x
    - Double GEneral SolVe
    - dgesv(n, nrhs, a, lda, ipiv, b, ldb, info)
    - D = double real, S = single real (float), C = single complex, Z = double complex
  - 2-part call: DGETRF/DGETRS: factorization/solve
  - DSYSV, DHESV, DPOSV: symmetric, Hermitian, positive definite, resp.

Matlab

- Matrix Laboratory:
  - Industrial quality technical computing platform
  - Many toolboxes for important problem domains
  - A very rational productivity option on a single compute node
  - Some distributed parallel support, but more complicated
- Solve $Ax = b$ in Matlab?

```
x = A\b;
```

  - Backslash symbol represents complex decision tree
  - Considerations:
    - Size, sparsity, condition number, …
    - Tim Davis: Backslash guy
- If Matlab works for your problem sizes, use it.
- Makes great prototyping environment in other cases

Problem solving environments

- Many productivity enhancing environments:
  - Numpy, Julia, Jupyter, others
- Python wrappers:
  - SWIG-based and others
  - Wrap high-performance libraries underneath
- Can be the right tool (and compete with Matlab):
  - Especially for exploration, but even for production settings
  - Not generally used on supercomputers, although always discussed

HIGH-PERFORMANCE NUMERICAL LIBRARIES

Fascinating history of early CSE

Early software libraries:

- EISPACK, LINPACK, MINPACK, etc.
- National Energy Software Center, 1972-1991
  - Founded & directed by Margaret Butler
    - Remarkable Career of a Pioneering Computational Scientist, SIAM News, Nov 2013

History of early CSE:

- Hybrid Zone: Computers and Science at Argonne National Laboratory, 1946-1992, Charles Yood, Docent Press, 2013
  - social history of computing: intersection of people, science, and the activity of computing

Broad range of HPC numerical software

Some packages with general-purpose, reusable algorithmic infrastructure in support of high-performance CSE:

- **AMReX -** https://ccse.lbl.gov/AMReX
- **Chombo** - https://commons.lbl.gov/display/chombo
- **Clawpack** - http://www.clawpack.org
- **Deal.II** - https://www.dealii.org
- **FEniCS** - https://fenicsproject.org
- **hypre** - http://www.llnl.gov/CASC/hypre
- **libMesh** - https://libmesh.github.io
- **MAGMA** - http://icl.cs.utk.edu/magma
- **MOOSE** - http://mooseframework.org
- **PETSc/TAO** – http://www.mcs.anl.gov/petsc
- **SUNDIALS** - http://computation.llnl.gov/casc/sundials
- **SuperLU** - http://crd-legacy.lbl.gov/~xiaoye/SuperLU
- **Trilinos** - https://trilinos.org
- **Uintah** - http://www.uintah.utah.edu
- **waLBerla** - http://www.walberla.net

See info about scope, performance, usage, including

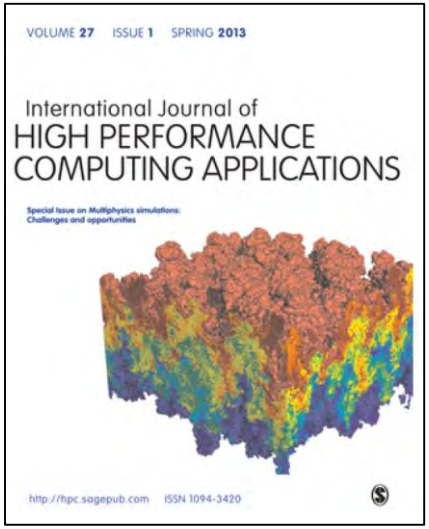
- tutorials
- demos
- examples
- how to contribute

… And many more projects address important aspects of high-performance CSE ... Explore, use, contribute!

Two key aspects of HPC library design

- **Library interfaces that are independent of physical processes and separate from choices of algorithms and data structures**
  - Cannot make assumptions about program startup or the use of state
  - Cannot seize control of 'main' or assume MPI_COMM_WORLD

- **Abstractions for mathematical objects** (e.g., vectors and matrices), enable dealing with composite operators and changes in architecture
  - **Any state data must be explicitly exchanged through an interface to maintain consistency**



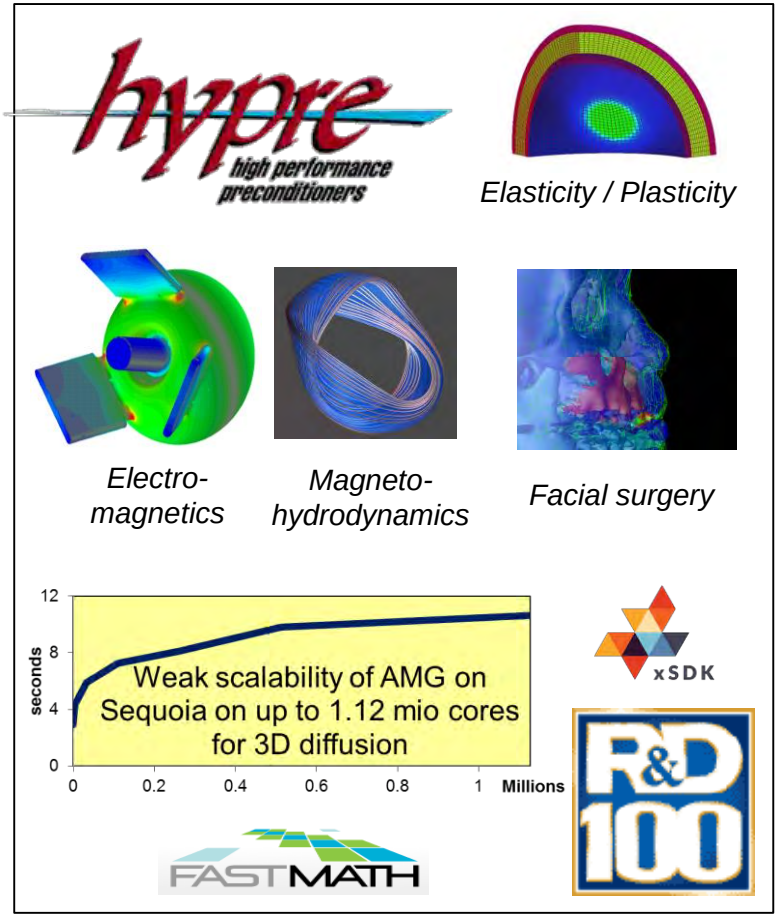
Reference: *Multiphysics simulations: Challenges and opportunities*, D.E. Keyes, L.C. McInnes, C.S. Woodward, et al., IJHPCA, special issue, Feb 2013

hypre

Lawrence Livermore National Laboratory

Highly scalable multilevel solvers and preconditioners.
Unique user-friendly interfaces. Flexible software design.
Used in a variety of applications. Freely available.

- **Conceptual interfaces**
  - Structured, semi-structured, finite elements, linear algebraic interfaces
  - Provide natural "views" of the linear system
  - Provide for more efficient (scalable) linear solvers through more effective data storage schemes and more efficient computational kernels
- **Scalable preconditioners and solvers**
  - Structured and unstructured algebraic multigrid (including constant-coefficient solvers)
  - Maxwell solvers, H-div solvers, and more
  - Matrix-free Krylov solvers
- **Open source software**
  - Used worldwide in a vast range of applications
  - Can be used through PETSc and Trilinos
  - Available on github

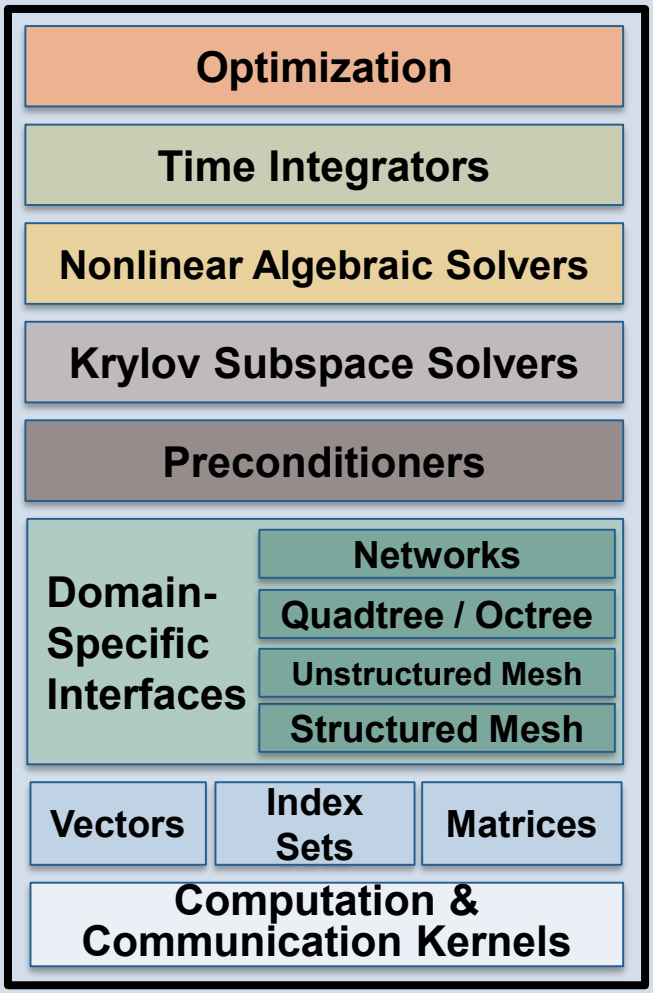


http://www.llnl.gov/CASC/hypre

PETSc/TAO:

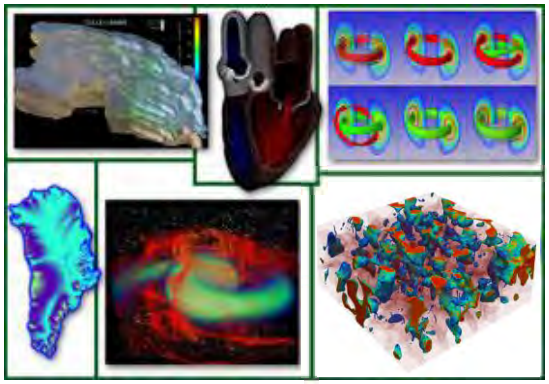
Portable, Extensible Toolkit for Scientific Computation / Toolkit for Advanced Optimization

Scalable algebraic solvers for PDEs. Encapsulate parallelism in high-level objects. Active & suppported user community. Full API from Fortran, C/C++, Python.



- **Easy customization and composability of solvers <u>at runtime</u>**
  - Enables optimality via flexible combinations of physics, algorithmics, architectures
  - Try new algorithms by composing new/existing algorithms (multilevel, domain decomposition, splitting, etc.)
- **Portability & performance**
  - Largest DOE machines, also clusters, laptops
  - Thousands of users worldwide

Argonne NATIONAL LABORATORY



PETSc provides the backbone of diverse scientific applications.

clockwise from upper left: hydrology, cardiology, fusion, multiphase steel, relativistic matter, ice sheet modeling

xSDK

FASTMATH

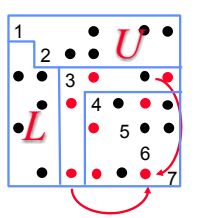
R&D 100

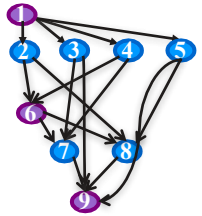
https://www.mcs.anl.gov/petsc

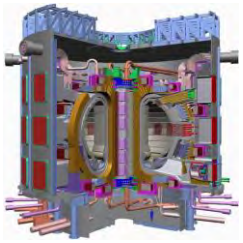
SuperLU

Supernodal Sparse LU Direct Solver. Unique user-friendly interfaces. Flexible software design. Used in a variety of applications. Freely available.

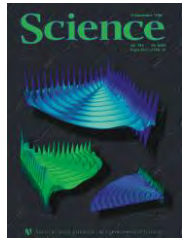
- **Capabilities**
  - Serial (thread-safe), shared-memory (SuperLU_MT, OpenMP or Pthreads), distributed-memory (SuperLU_DIST, hybrid MPI+ OpenM + CUDA).
    - Implemented in C, with Fortran interface
  - Sparse LU decomposition, triangular solution with multiple right-hand sides
  - Incomplete LU (ILU) preconditioner in serial SuperLU
  - Sparsity-preserving ordering:
    - Minimum degree ordering applied to A^TA or A^T+A
    - Nested dissection ordering applied to A^TA or A^T+A [(Par)METIS, (PT)-Scotch]
  - User-controllable pivoting: partial pivoting, threshold pivoting, static pivoting
  - Condition number estimation, iterative refinement.
  - Componentwise error bounds
- **Open source software**
  - Used worldwide in a vast range of applications
  - Can be used through PETSc and Trilinos
  - Available on github







ITER tokamak



quantum mechanics

Widely used in commercial software, including AMD (circuit simulation), Boeing (aircraft design), Chevron, ExxonMobile (geology), Cray's LibSci, FEMLAB, HP's MathLib, IMSL, NAG, SciPy, OptimaNumerics, Walt Disney Animation.

FASTMATH xSDK

http://crd-legacy.lbl.gov/~xiaoye/SuperLU

Trilinos

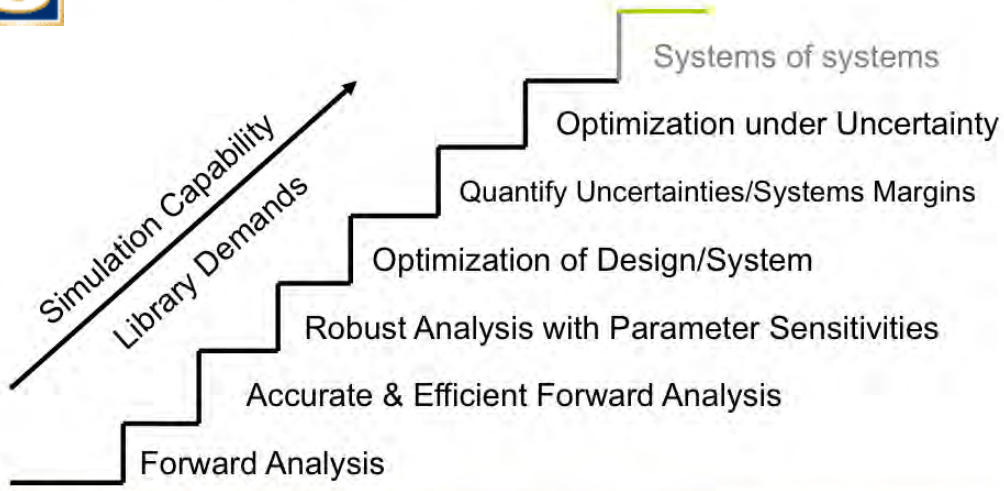
Sandia National Laboratories

Optimal kernels to optimal solutions. Over 60 packages. Laptops to leadership systems. Next-gen systems, multiscale/multiphysics, large-scale graph analysis.

- **Optimal kernels to optimal solutions**
  - Geometry, meshing
  - Discretization, load balancing
  - Scalable linear, nonlinear, eigen, transient, optimization, UQ solvers
  - Scalable I/O, GPU, manycore
- **60+ packages**
  - Other distributions: Cray LIBSCI, Github repo
  - Thousands of users, worldwide distribution
  - Laptops to leadership systems

R&D 100

Transforming Computational Analysis To Support High Consequence Decisions



Each stage requires *greater performance* and *error control* of prior stages:
Always will need: more accurate and scalable methods.
more sophisticated tools.

https://trilinos.org

xSDK FASTMATH Trilinos

Outline

- Motivation
- Libraries: Reusable research software
- Managing risks of external software use
- Toward scientific software ecosystems
  - xSDK
- International community efforts
- Get involved!

Risks in using external software

- Risk of not being in control of the software that is being leveraged, or its long-term availability
  - Concern about long-term maintenance of the 3rd-party software
  - Ability to influence the development of the 3rd-party software to enhance or maintain its relevance to the application being developed
  - Concern about the "return on investment" of time spent in learning the 3rd-party software
- Software portability
- Software quality
- Software sustainability

Managing risks in using external software

- **Strategy for an existing code**
  1. Identify functionality in your code that could be replaced by a 3rd party
  2. Create or isolate an interface (function call) that calls your code as though it is 3rd party
  3. Integrate 3rd party code as an option to use instead of yours
  4. Future: Quickly integrate other 3rd party codes via the same interface

- **Strategy for a new code**
  1. Study available 3rd party options that provide functionality
  2. Create an interface that supports more than one similar 3rd party code
  3. Write adapter interfaces that support two or more 3rd party options

Outline

- Motivation
- Libraries: Reusable research software
- Managing risks of external software use
- Toward scientific software ecosystems
  - xSDK
- International community efforts
- Get involved!

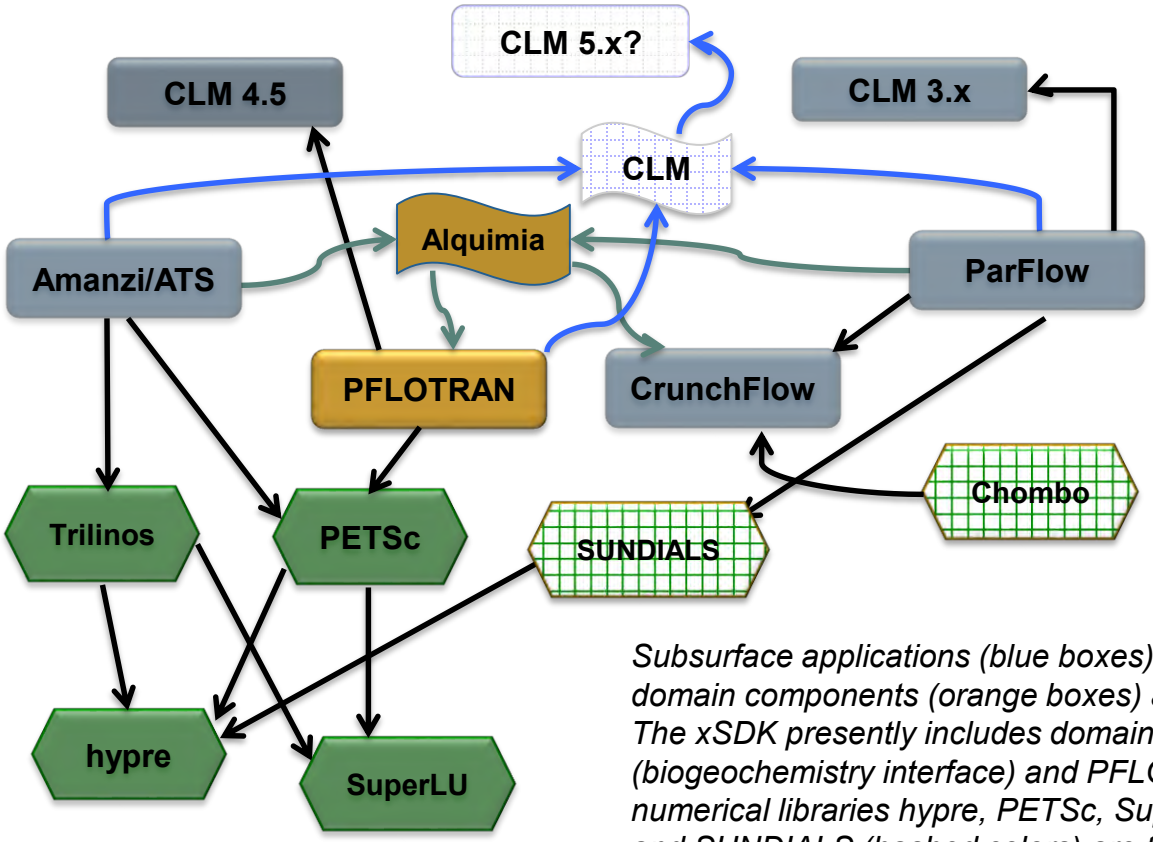
Software libraries are not enough

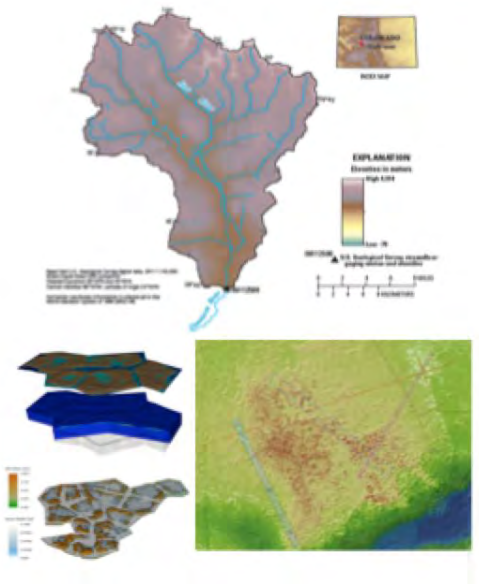
- Well-designed libraries provide critical functionality … But alone are not sufficient to address all aspects of next-generation scientific simulation and analysis.
- Applications need to use software packages **in combination** on ever evolving architectures

"The way you get programmer productivity is by eliminating lines of code you have to write."

– Steve Jobs, Apple World Wide Developers Conference, Closing Keynote, 1997

Example: Multiscale, multiphysics modeling of watershed dynamics requires combined use of independent packages





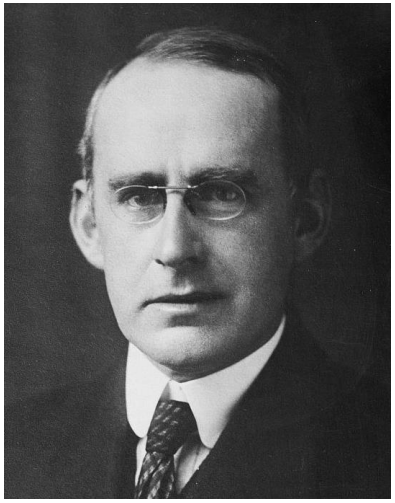
Subsurface applications (blue boxes) and their present usage of SDK domain components (orange boxes) and numerical libraries (green boxes). The xSDK presently includes domain components Alquimia (biogeochemistry interface) and PFLOTRAN (subsurface flow) and the numerical libraries hypre, PETSc, SuperLU, and Trilinos. CLM, Chombo, and SUNDIALS (hashed colors) are targeted for later inclusion.

Need software ecosystem perspective

Ecosystem: A group of independent but interrelated elements comprising a unified whole

Ecosystems are challenging!

"We often think that when we have completed our study of one we know all about two, because 'two' is 'one and one.' We forget that we still have to make a study of 'and.' "



− Sir Arthur Stanley Eddington (1892−1944), British astrophysicist

Difficulties in combined use of independently developed software packages

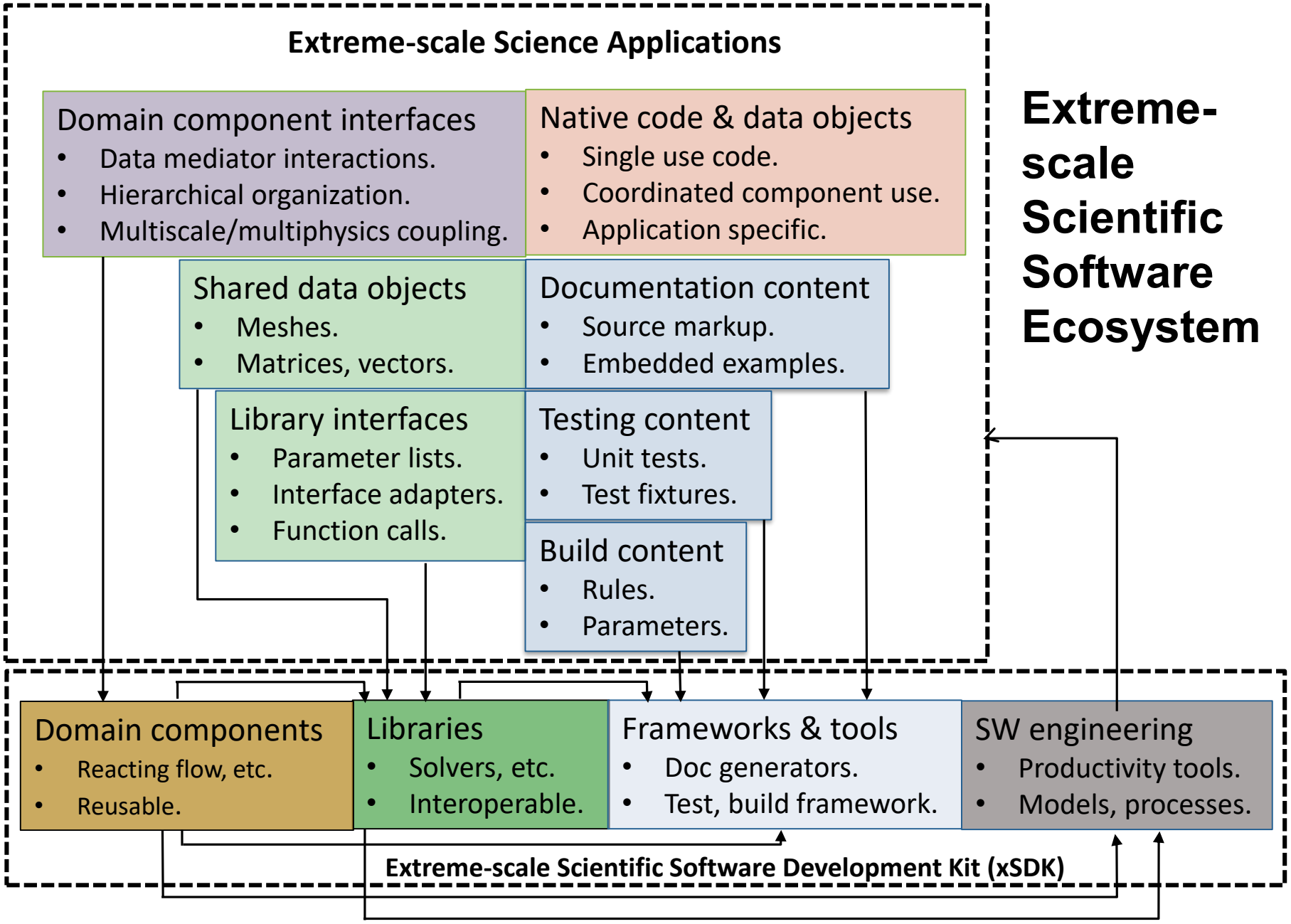
Levels of package interoperability:

- **Interoperability level 1**
  - both packages can be used (side by side) in an application
- **Interoperability level 2**
  - the libraries can exchange date (or control data) with each other
- **Interoperability level 3**
  - each library can call the other library to perform unique computations

Challenges:

- Obtaining, configuring, and installing multiple independent software packages is tedious and error prone.
  - Need consistency of compiler (+version, options), 3rd-party packages, etc.
- Namespace conflicts
- Incompatible versioning
- And even more challenges for deeper levels of interoperability

Reference: What are Interoperable Software Libraries? Introducing the xSDK


Extreme-scale Science Applications
Domain component interfaces
• Data mediator interactions.
• Hierarchical organization.
• Multiscale/multiphysics coupling.
Native code & data objects
• Single use code.
• Coordinated component use.
• Application specific.
Shared data objects
• Meshes.
• Matrices, vectors.
Documentation content
• Source markup.
• Embedded examples.
Library interfaces
• Parameter lists.
• Interface adapters.
• Function calls.
Testing content
• Unit tests.
• Test fixtures.
Build content
• Rules.
• Parameters.
Extreme-scale Scientific Software Ecosystem
Domain components
• Reacting flow, etc.
• Reusable.
Libraries
• Solvers, etc.
• Interoperable.
Frameworks & tools
• Doc generators.
• Test, build framework.
SW engineering
• Productivity tools.
• Models, processes.
Extreme-scale Scientific Software Development Kit (xSDK)

Classic vs Ecosystem Approaches

- **Classic application development approach:**
  - App developers write most code
  - Source code considered private
  - Make occasional use of libraries, but only those “baked into” the OS
  - Portability challenges, unmanaged disruptions: Low risk. But…
- **Ecosystem-based application development approach:**
  - App developers use composition, write glue code & unique functionality
  - Source code includes substantial 3rd party packages
  - Risks (if 3rd party code is poor):
    - Dependent on portability of 3rd party code
    - Upgrades of 3rd party package can be disruptive (interface changes, regressions)
  - Opportunities (if 3rd party code is good):
    - 3rd party improvements are yours (for free!)
    - Portability to new architectures is seamless
- **Ecosystem imperative: High quality is essential, more affordable**

xSDK: https://xsdk.info
Building the foundation of an extreme-scale scientific software ecosystem

xSDK community policies: Help address challenges in interoperability and sustainability of software developed by diverse groups at different institutions

xSDK compatible package: must satisfy the mandatory xSDK policies

Topics include: configuring, installing, testing, MPI usage, portability, contact and version information, open source licensing, namespacing, and repository access

Also specify **recommended policies**, which currently are encouraged but not required

Topics include: public repository access, error handling, freeing system resources, and library dependencies

xSDK member package:

(1) Must be an xSDK-compatible package, *and*
(2) it uses or can be used by another package in the xSDK, and the connecting interface is regularly tested for regressions.

xSDK policies 0.3.0: Nov 2017

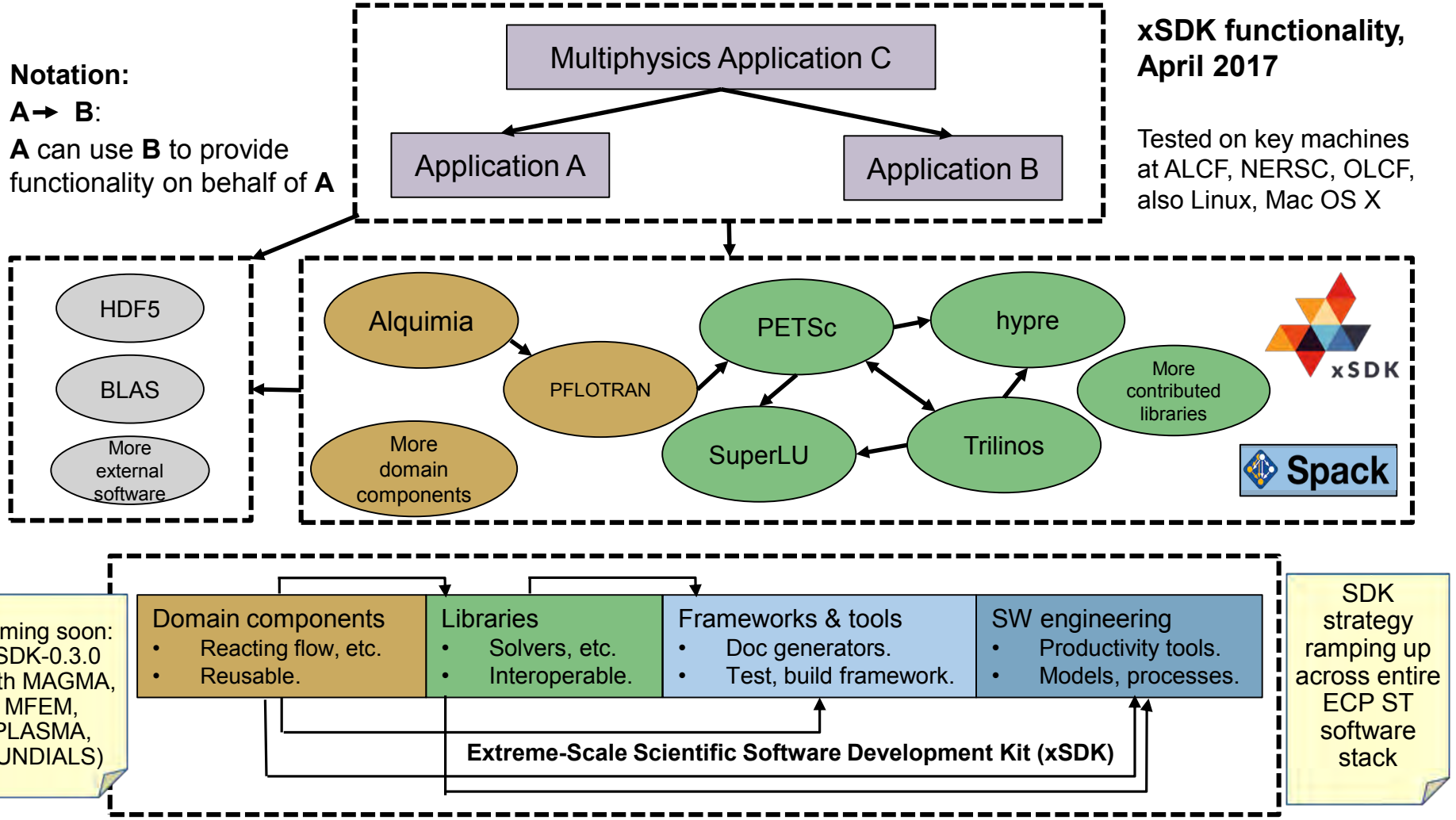
- Combined use of independently developed packages

Impact:

- Improved code quality, usability, access, sustainability
- Foundation for work on performance portability and deeper interoperability

We encourage feedback and contributions!

xSDK release 0.2.0: Packages can be readily used in combination by multiphysics, multiscale apps



xSDK community policies

Version 0.3.0, Nov 2017

xSDK compatible package: Must satisfy mandatory xSDK policies:

M1. Support xSDK community GNU Autoconf or CMake options.

M2. Provide a comprehensive test suite.

M3. Employ user-provided MPI communicator.

M4. Give best effort at portability to key architectures.

M5. Provide a documented, reliable way to contact the development team.

M6. Respect system resources and settings made by other previously called packages.

M7. Come with an open source license.

M8. Provide a runtime API to return the current version number of the software.

M9. Use a limited and well-defined symbol, macro, library, and include file name space.

M10. Provide an accessible repository (not necessarily publicly available).

M11. Have no hardwired print or IO statements.

M12. Allow installing, building, and linking against an outside copy of external software.

M13. Install headers and libraries under <prefix>/include/ and <prefix>/lib/.

M14. Be buildable using 64 bit pointers. 32 bit is optional.

M15. All xSDK compatibility changes should be sustainable.

M16. The package must support production-quality installation compatible with the xSDK install tool and xSDK metapackage.

Also specify **recommended policies**, which currently are encouraged but not required:

R1. Have a public repository.

R2. Possible to run test suite under valgrind in order to test for memory corruption issues.

R3. Adopt and document consistent system for error conditions/exceptions.

R4. Free all system resources it has acquired as soon as they are no longer needed.

R5. Provide a mechanism to export ordered list of library dependencies.

xSDK member package: Must be an xSDK-compatible package, *and* it uses or can be used by another package in the xSDK, and the connecting interface is regularly tested for regressions.

We welcome feedback. What policies make sense for your software?

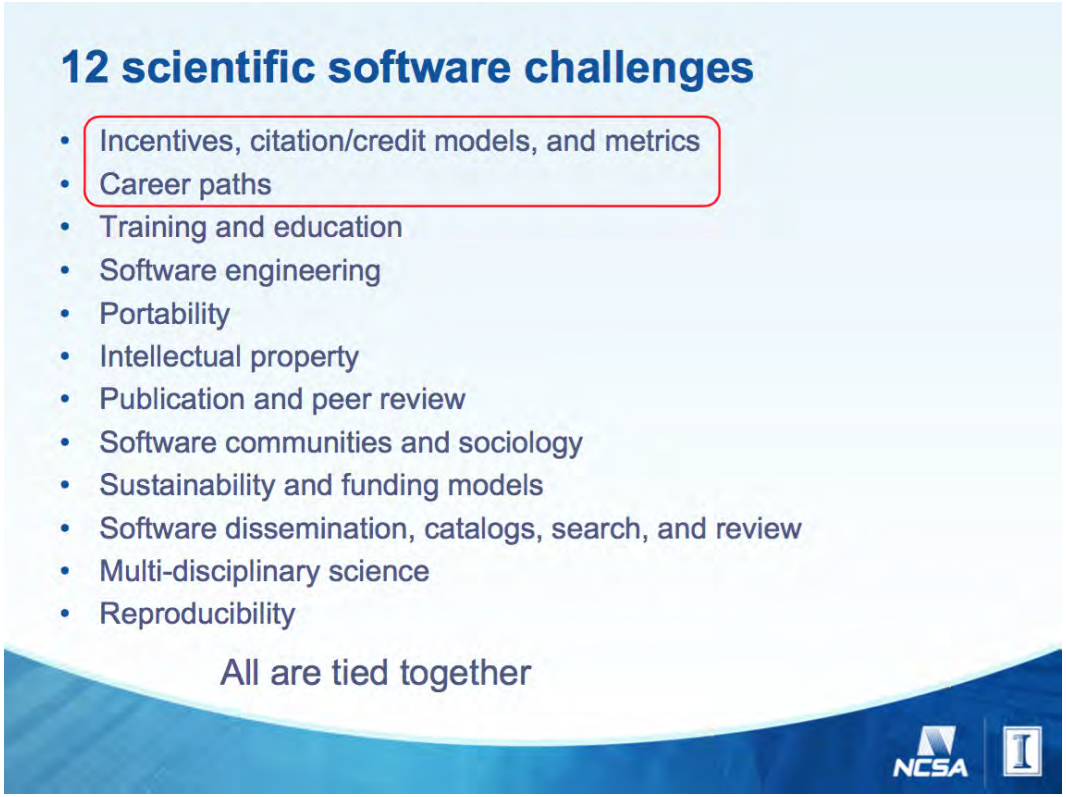
https://xsdk.info/policies

Outline

- Motivation
- Libraries: Reusable research software
- Managing risks of external software use
- Toward scientific software ecosystems
  - xSDK
- International community efforts
- Get involved!

Community software ecosystems require high quality software

- **Complex, intertwined challenges**
- **Need community efforts to**
  - Improve software quality
  - Change research culture
  - Promote collaboration
  - Etc.
- **Get involved!**



Reference: Daniel S. Katz, *Software in Research: Underappreciated and Underrewarded*, 2017 eResearch Australasia conference, Oct 20, 2017, https://doi.org/10.6084/m9.figshare.5518933

Synergistic efforts to enable quality reusable software are essential for next-generation CSE

- Computational Science Stack Exchange
- NUMFocus
- Software Carpentry, Data Carpentry
- Software Sustainability Institute
- WSSSPE
- IEEE THPC Software Engineering Practices Initiative
- Better Scientific Software
- Projects such as
  - Molecular Sciences Software Institute
  - IDEAS Software Productivity project
- And many more …

Resources … and opportunities to get involved

- **Computational Science Stack Exchange: SciComp.StackExchange.com**
  - Question and answer site for scientists using computers to solve scientific problems

- **NUMFocus: https://www.numfocus.org**
  - Umbrella nonprofit that supports and promotes open source scientific computing

- **Software Carpentry: http://software-carpentry.org**
  - Volunteer non-profit organization dedicated to teaching basic computing skills to researchers.
  - Lessons: https://software-carpentry.org/lessons/

Resources … and opportunities to get involved

- **Software Sustainability Institute: http://www.software.ac.uk**
  - Institute to support UK's research software community: cultivating better, more sustainable, research software to enable world-class research
  - Guides: https://www.software.ac.uk/resources/guides-everything
- **WSSSPE: http://wssspe.researchcomputing.org.uk**
  - International community-driven organization that promotes sustainable research software
- **IEEE TCHPC Software Engineering Practices Initiative**: **http://tc.computer.org/tchpc/home-page/software-engineering-practices/**
  - Leading a conversation on impactful software engineering practices for HPC

IDEAS productivity
www.ideas-productivity.org

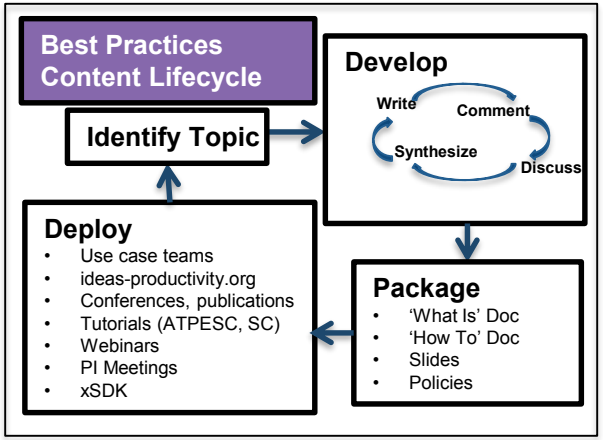
Resources for software productivity & sustainability—key element of overall scientific productivity

Approach: Collaborate with the community to curate, create, & disseminate software methodologies, processes, and tools that lead to improved scientific software

Webinars: Best Practices for HPC Software Developers

- *Managing Defects in HPC Software*
- *Barely Sufficient Project Management: A few techniques for improving your scientific software efforts*
- *Using the Roofline Model and Intel Advisor*
- *Intermediate Git*
- *Python in HPC*

Content available via https://ideas-productivity.org/
Transitioning soon to: better scientific software



Tutorials

- *Introduction to Git*
- *Better (Small) Scientific Software Teams*
- *Improving Reproducibility through Better Software Practices*
- *Testing and Verification*
- *Code Coverage and Continuous Integration*
- *An Introduction to Software Licensing*

What Is & HowTo docs: Brief sketches of best practices

- *What Is Software Configuration?*
- *How to Configure Software*
- *What Is Performance Portability?*
- *How to Enable Performance Portability*
- *What Are Software Testing Practices?*
- *How to Add and Improve Testing in a CSE Software Project*
- *What Is Good Documentation?*
- *How to Write Good Documentation*
- *What Is Version Control?*
- *How to Do Version Control with Git*

[And more]

Under development: New web-based hub for scientific software improvement exchange

Launching at SC17

BSSw Software Platform

Component Technology	Backend		Frontend
	Google Docs	**GitHub**	**Ruby on Rails**
Location	Google Drive	betterscientificsoftware GitHub organization	betterscientificsoftware.io
Purpose	• Rapid collaborative content development. • Multi-user typing, suggest edits, comments.	• Content creation, refinement, management (from Google Drive). • Content packaging for use with BSSw.io	• User-facing portal • Polished backend content • Blogs, forums • Mailing lists.
Contributors	Community subject matter experts	Community subject matter experts, BSSw staff	BSSw staff. Web development experts.
Consumers	BSSw GitHub Backend	BSSw Frontend	CSE community
Content Notes	Content migrates to GitHub after it stabilizes	Content managed in git repos, markdown	Content from Backend

Contribute! Share your insights on CSE software practices and processes:

- https://github.com/betterscientificsoftware/betterscientificsoftware.github.io/blob/master/README.md
- Or search "github betterscientificsoftware"

Collaborative community software ecosystems help improve CSE productivity and sustainability

- **What makes sense for your software?**
  - **Consider resources for improving software quality**
  - **What community software ecosystems do you want to use and be a part of?**
- **Get involved!**
  - Various community efforts
  - Upcoming Blue Waters Webinars
    - *Scientific Software Ecosystems* track
      - Suggestions welcome: Contact Scott Lathrop lathrop@illinois.edu
    - See other tracks too, in particular new track on *Software Engineering*
- **Coming soon**
  - **SC17 BOF**: Software Engineering & Reuse in CSE (November 14, 2017)
  - **Special Issue, IEEE CiSE** (Stay tuned … CFP to be posted soon)

Impact

- **Better:** Science, portability, robustness, composability
- **Faster:** Execution, development, dissemination
- **Cheaper:** Fewer staff hours and lines of code