SAND2015-10654PE

Strategies for Next Generation HPC Applications and Systems

Michael A. Heroux
Scalable Algorithms Department
Sandia National Laboratories

SNL Collaborators: Erik Boman, Marc Gamell, Carter Edwards, James, Elliot, Mark Hoemmen, Siva Rajamanickam, Keita Teranishi, Christian Trott

IDEAS Project: Lois McInnes, David Bernholdt, David Moulton, Hans Johansen

Sandia National Laboratories is a multi-program laboratory operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin company, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

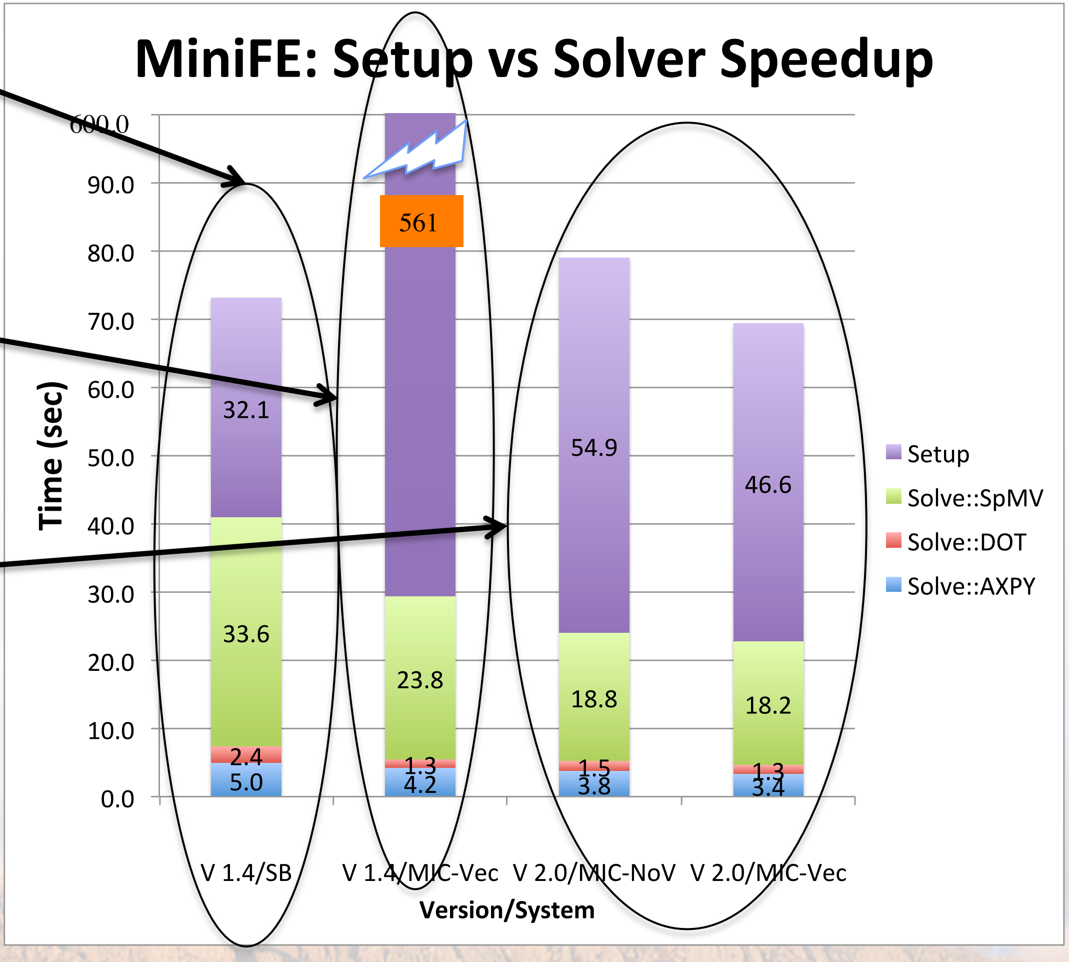
Sandia National Laboratories

Outline

- Background.
- “Easy” and “Hard”.
- SW Engineering and Productivity.
- Application Design and Productivity.
- Productivity Incentives.

The work ahead of us: Threads and vectors MiniFE 1.4 vs 2.0 as Harbingers

- Typical MPI-only run:
  - Balanced setup vs solve
- First MIC run:
  - Thread/vector solver
  - No-thread setup
- V 2.0: Thread/vector
  - Lots of work:
    - Data placement, const /restrict declarations, avoid shared writes, find race conditions, …
  - Unique to each app

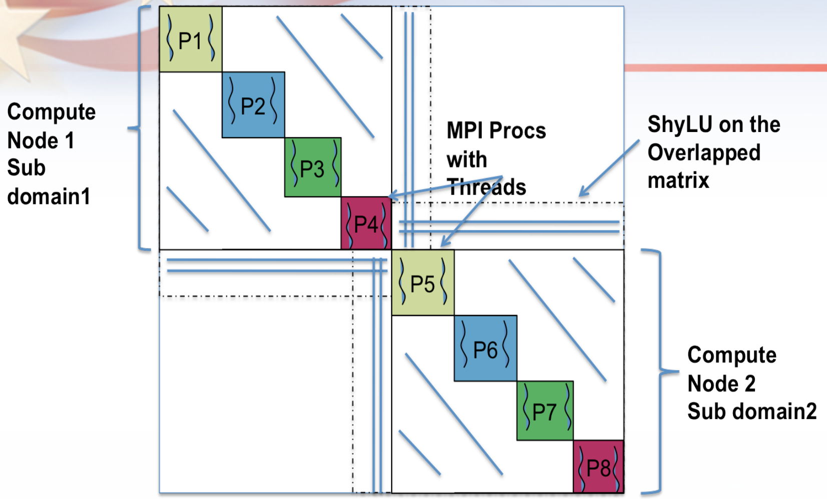


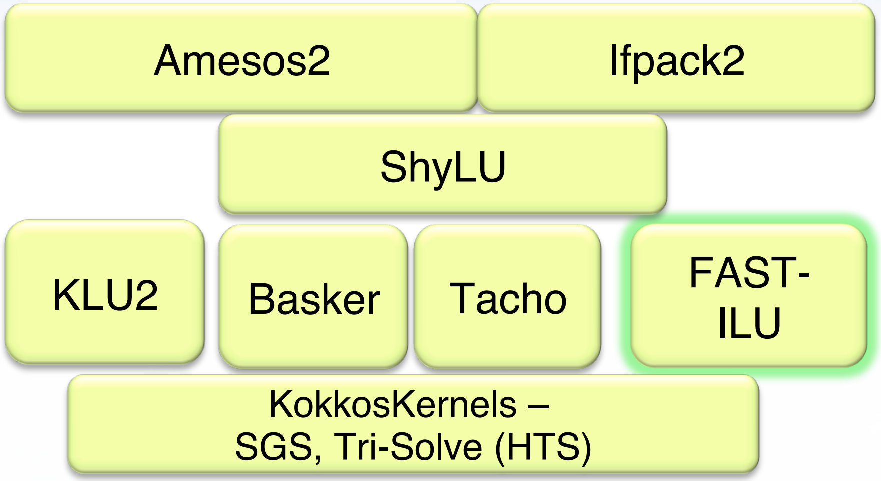
“Easy” Work in Progress

- Thread-scalable algorithms:
  - Turning out to be feasible.
  - Clever ideas: Fast-ILU (Chow, Anzt, Rajamanickam, etc.)
  - Lots to do, but steady progress
  - Much evidence in today’s talks.
- Current Thread Programming Environments:
  - C++, OpenMP, others: Working.
  - Runtimes: Still a lot of work, but progress.
- Lots to do, but community is focused.

Trilinos/ShyLU and Subdomain Solvers : Overview

Led by Siva Rajamanickam, Sandia




Amesos2

Ifpack2

ShyLU

KLU2

Basker

Tacho

FAST-ILU

KokkosKernels – SGS, Tri-Solve (HTS)

- MPI+X based subdomain solvers
  - Decouple the notion of one MPI rank as one subdomain: Subdomains can span multiple MPI ranks each with its own subdomain solver using X or MPI+X
  - Epetra based solver, Tpetra interface still being developed
    - Trilinos Solver Factory a big step forward to get this done (*M. Hoemmen*)
- Subpackages of ShyLU: Multiple Kokkos-based options for on-node parallelism
  - Basker : LU or ILU (t) factorization (J. Booth)
  - Tacho: Incomplete Cholesky - IC (k) (K. Kim)
  - Fast-ILU: Fast-ILU factorization for GPUs (A. Patel)
- KokkosKernels: Coloring based Gauss-Seidel (M. Deveci), Triangular Solves
- **Experimental code base under active development.**

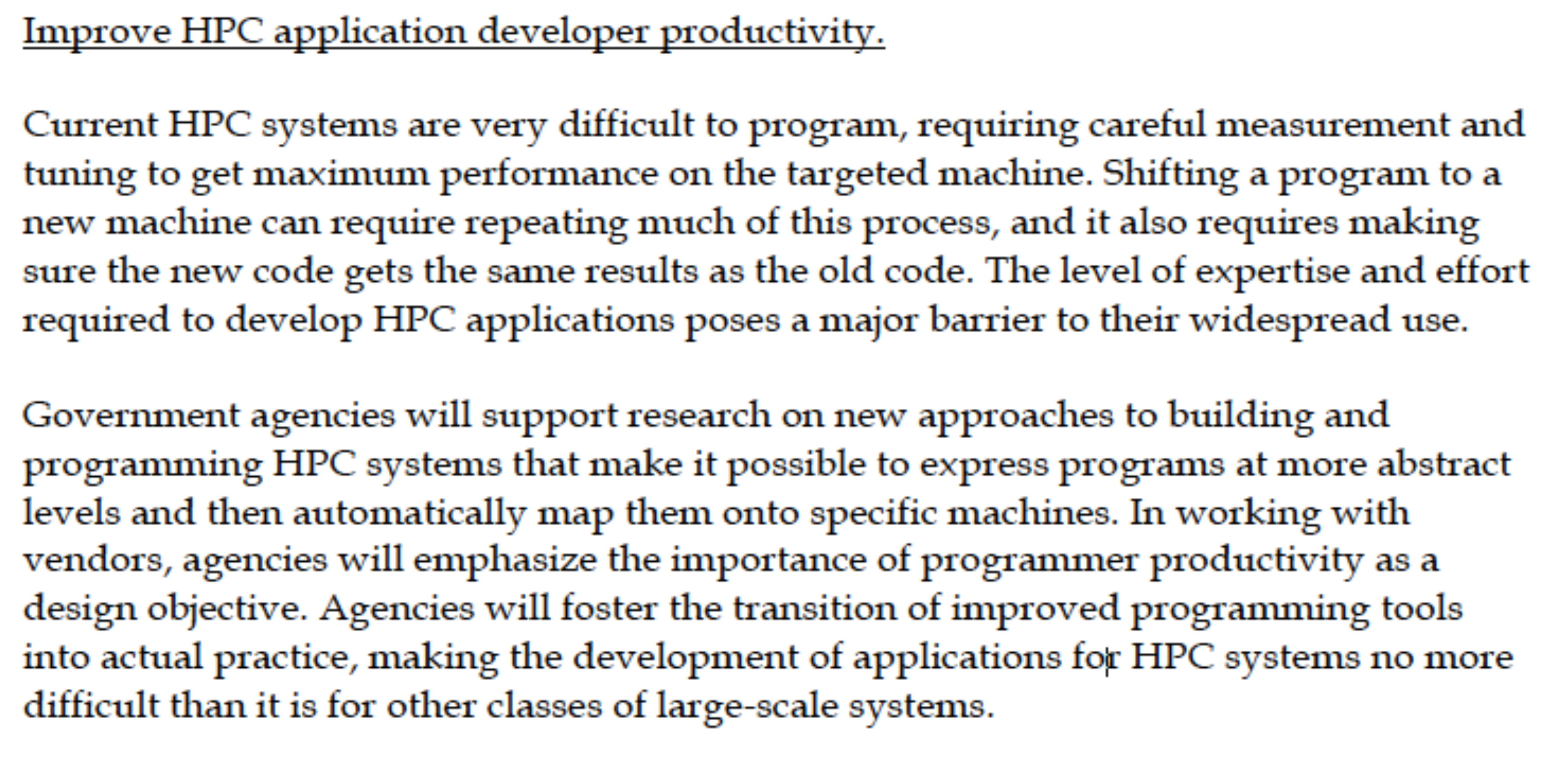
More “Easy” Work in Progress

- Resilience:
  - CPR: Compression, NVRAM, Offloading.
    - Steady progress, long life extension.
  - LFLR: Good progress with ULFM.
    - Example Paper: *Local Recovery And Failure Masking For Stencil-based Applications At Extreme Scales*
      - Marc Gamell, Keita Teranishi, Michael A. Heroux, Jackson Mayo, Hemanth Kolla, Jacqueline Chen, Manish Parashar http://sc15.supercomputing.org/schedule/event_detail?evid=pap682 (Thursday, 1:30)
- System-level error detection/correction.
    - Many unexploited options available. Talk with Al Gara, Intel.
- Conjecture:
  - System developers will not permit reduced reliability until the user community produces more resilient apps.

“Hard” Work

- Billions (yes, billions) SLOC of encoded science & engineering.
- Challenge:
  - Transfer, refactor, rewrite for modern systems.
  - Do so with modest investment bump up.
  - Deliver science at the same time.
  - Make the next disruption easier to address.

From the NSCI Announcement (Fact sheet):



Improve HPC application developer productivity.

Current HPC systems are very difficult to program, requiring careful measurement and tuning to get maximum performance on the targeted machine. Shifting a program to a new machine can require repeating much of this process, and it also requires making sure the new code gets the same results as the old code. The level of expertise and effort required to develop HPC applications poses a major barrier to their widespread use.

Government agencies will support research on new approaches to building and programming HPC systems that make it possible to express programs at more abstract levels and then automatically map them onto specific machines. In working with vendors, agencies will emphasize the importance of programmer productivity as a design objective. Agencies will foster the transition of improved programming tools into actual practice, making the development of applications for HPC systems no more difficult than it is for other classes of large-scale systems.

https://www.whitehouse.gov/sites/default/files/microsites/ostp/nsci_fact_sheet.pdf

Productivity

Better, Faster, Cheaper: Pick all three

Sandia National Laboratories

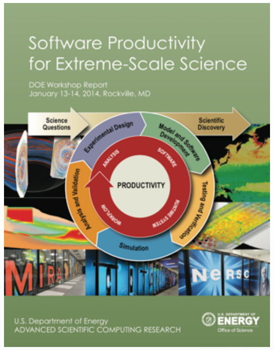
Confluence of trends

- Fundamental trends:
  - Disruptive HW changes: Requires thorough algorithm/code refactoring
  - Demands for coupling: Multiphysics, multiscale
- Challenges:
  - Need refactorings: Really, continuous change
  - Modest app development funding: No monolithic apps
  - Requirements are unfolding, evolving, not fully known *a priori*
- Opportunities:
  - Better design and SW practices & tools are available
  - Better SW architectures: Toolkits, libraries, frameworks
- Basic strategy: Focus on <u>productivity</u>

Interoperable Design of Extreme-scale Application Software (IDEAS)

Motivation

Enable *increased scientific productivity,* realizing the potential of extreme- scale computing, through *a new interdisciplinary and agile approach to the scientific software ecosystem*.



Software Productivity for Extreme-Scale Science

DOE Workshop Report
January 13-14, 2014, Rockville, MD

U.S. Department of Energy
ADVANCED SCIENTIFIC COMPUTING RESEARCH

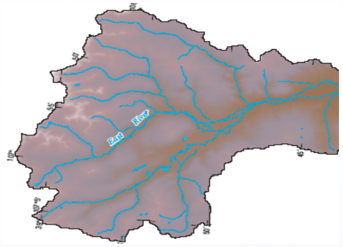
Objectives

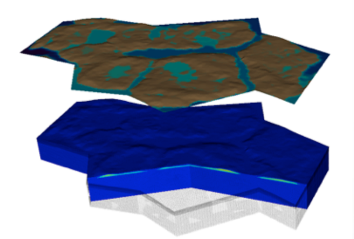
Address confluence of trends in hardware and increasing demands for predictive multiscale, multiphysics simulations.

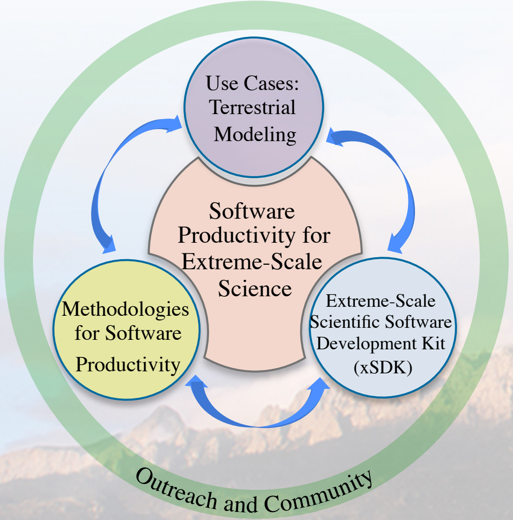
Respond to trend of continuous refactoring with efficient agile software engineering methodologies and improved software design.

Impact on Applications & Programs

Terrestrial ecosystem *use cases tie IDEAS to modeling and simulation goals* in two Science Focus Area (SFA) programs and both Next Generation Ecosystem Experiment (NGEE) programs in DOE Biologic and Environmental Research (BER).







Use Cases: Terrestrial Modeling
Software Productivity for Extreme-Scale Science
Methodologies for Software Productivity
Extreme-Scale Scientific Software Development Kit (xSDK)
Outreach and Community

Approach

ASCR/BER partnership ensures delivery of both crosscutting methodologies and metrics with impact on real application and programs.

Interdisciplinary multi-lab team (ANL, LANL, LBNL, LLNL, ORNL, PNNL, SNL)

ASCR Co-Leads: Mike Heroux (SNL) and Lois Curfman McInnes (ANL)

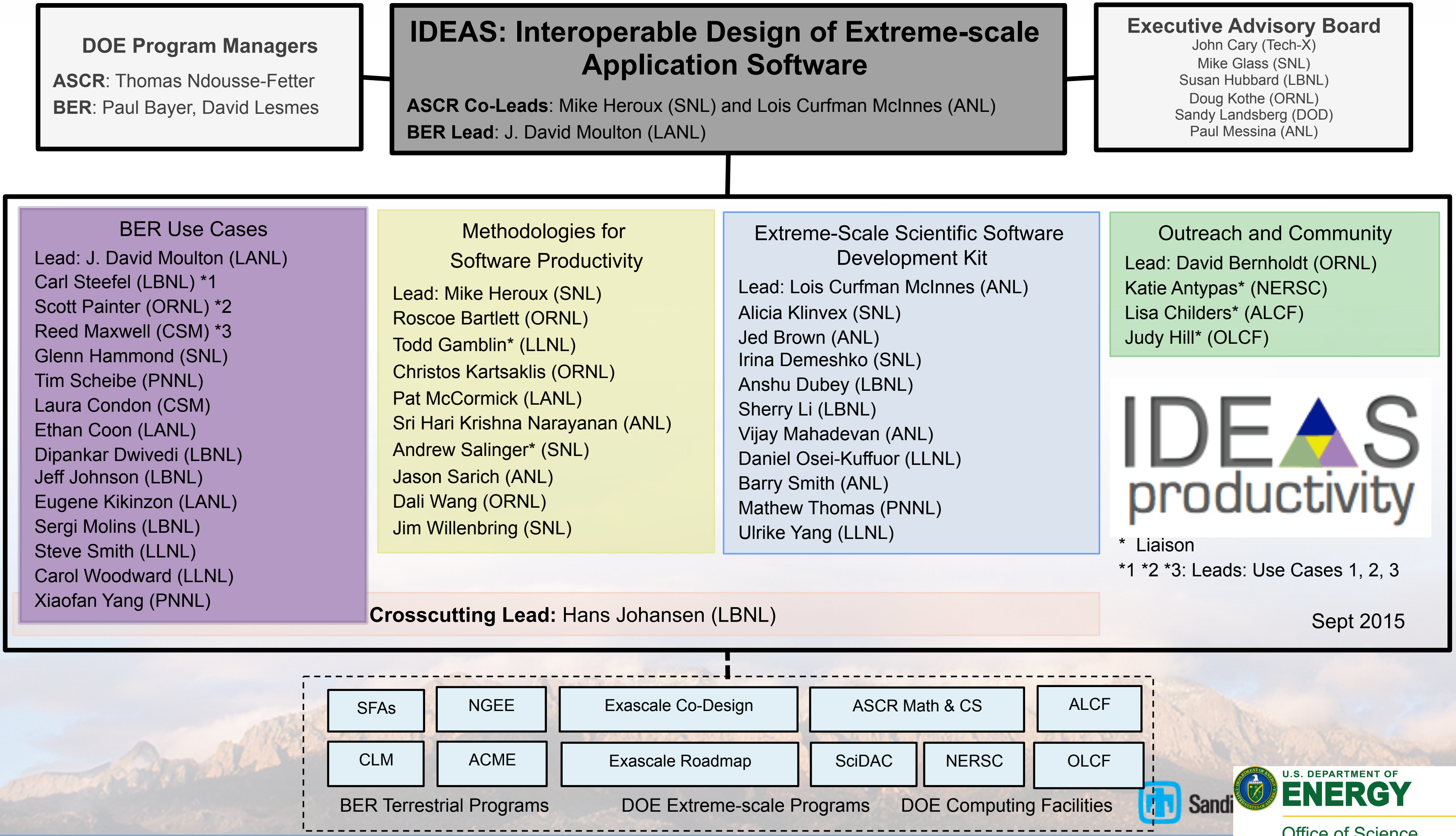
BER Lead: David Moulton (LANL)

Topic Leads: David Bernholdt (ORNL) and Hans Johansen (LBNL)

Integration and synergistic advances in three communities deliver scientific productivity; outreach establishes a new holistic perspective for the broader scientific community.

www.ideas-productivity.org

IDEAS project structure and interactions



DOE Program Managers

ASCR: Thomas Ndousse-Fetter

BER: Paul Bayer, David Lesmes

IDEAS: Interoperable Design of Extreme-scale Application Software

ASCR Co-Leads: Mike Heroux (SNL) and Lois Curfman McInnes (ANL)

BER Lead: J. David Moulton (LANL)

Executive Advisory Board

John Cary (Tech-X)
Mike Glass (SNL)
Susan Hubbard (LBNL)
Doug Kothe (ORNL)
Sandy Landsberg (DOD)
Paul Messina (ANL)

BER Use Cases

Lead: J. David Moulton (LANL)
Carl Steefel (LBNL) *1
Scott Painter (ORNL) *2
Reed Maxwell (CSM) *3
Glenn Hammond (SNL)
Tim Scheibe (PNNL)
Laura Condon (CSM)
Ethan Coon (LANL)
Dipankar Dwivedi (LBNL)
Jeff Johnson (LBNL)
Eugene Kikinzon (LANL)
Sergi Molins (LBNL)
Steve Smith (LLNL)
Carol Woodward (LLNL)
Xiaofan Yang (PNNL)

Methodologies for Software Productivity

Lead: Mike Heroux (SNL)
Roscoe Bartlett (ORNL)
Todd Gamblin* (LLNL)
Christos Kartsaklis (ORNL)
Pat McCormick (LANL)
Sri Hari Krishna Narayanan (ANL)
Andrew Salinger* (SNL)
Jason Sarich (ANL)
Dali Wang (ORNL)
Jim Willenbring (SNL)

Extreme-Scale Scientific Software Development Kit

Lead: Lois Curfman McInnes (ANL)
Alicia Klinvex (SNL)
Jed Brown (ANL)
Irina Demeshko (SNL)
Anshu Dubey (LBNL)
Sherry Li (LBNL)
Vijay Mahadevan (ANL)
Daniel Osei-Kuffuor (LLNL)
Barry Smith (ANL)
Mathew Thomas (PNNL)
Ulrike Yang (LLNL)

Outreach and Community

Lead: David Bernholdt (ORNL)
Katie Antypas* (NERSC)
Lisa Childers* (ALCF)
Judy Hill* (OLCF)

* Liaison

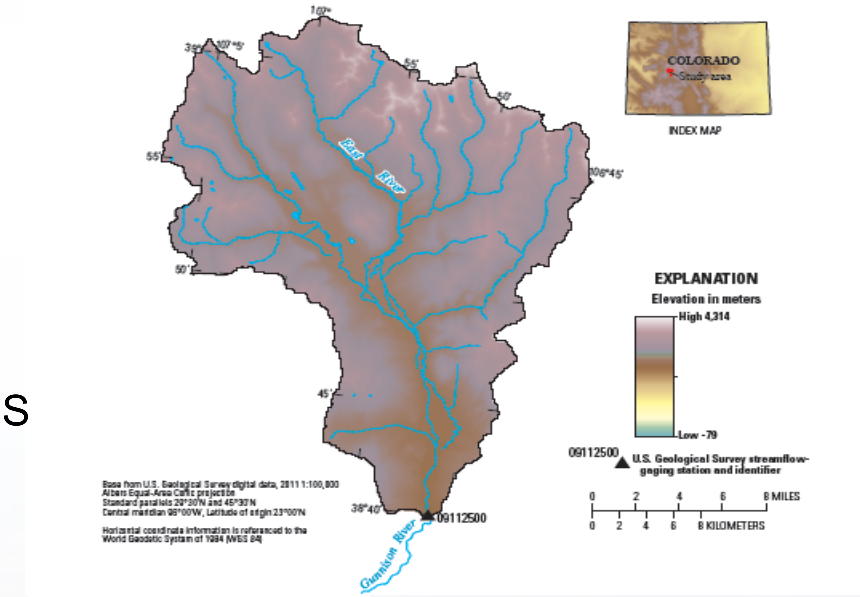
*1 *2 *3: Leads: Use Cases 1, 2, 3

Crosscutting Lead: Hans Johansen (LBNL)

Sept 2015

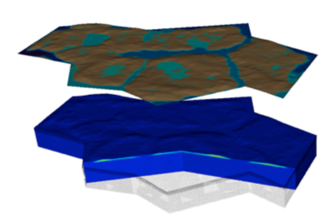
Use cases: Multiscale, multiphysics representation of watershed dynamics

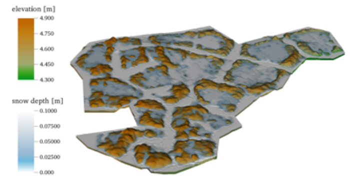
- Use Case 1: Hydrological and biogeochemical cycling in the Colorado River System
- Use Case 2: Thermal hydrology and carbon cycling in tundra at the Barrow Environmental Observatory
- Use Case 3: Hydrologic, land surface, and atmospheric process coupling over the continental United States
- Leverage and complement existing SBR and TES programs:
  - LBNL and PNNL SFAs
  - NGEE Arctic and Tropics
- General approach:
  - Leverage existing open source application codes
  - Improve software development practices
  - Targeted refactoring of interfaces, data structures, and key components to facilitate interoperability
  - Modernize management of multiphysics integration and multiscale coupling

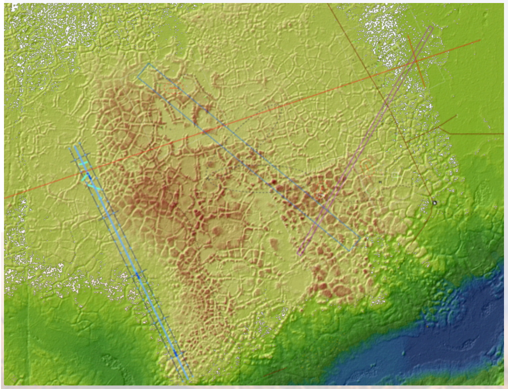


Base from U.S. Geological Survey digital data, 2011 1:100,000
Albers Equal-Area Conic projection
Standard parallels 29°30'N and 45°30'N
Central meridian 96°00'W, Latitude of origin 23°00'N

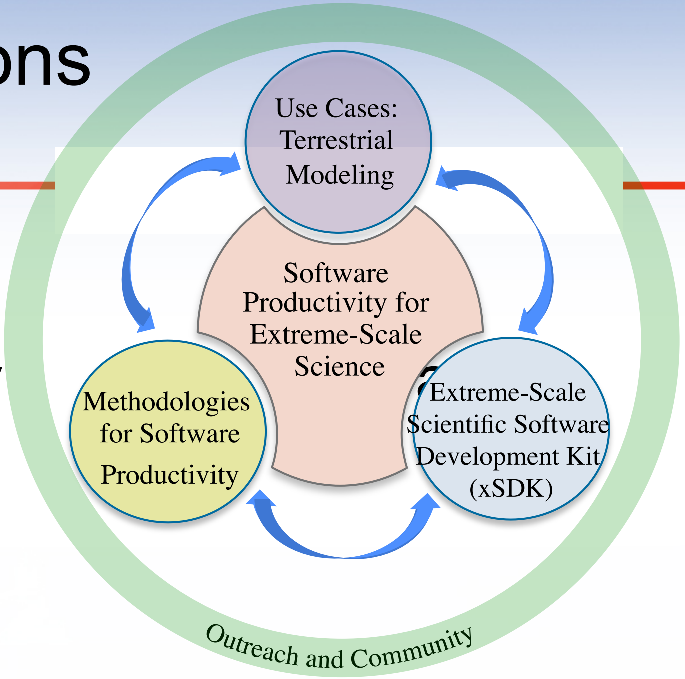
Horizontal coordinate information is referenced to the World Geodetic System of 1984 (WGS 84)







IDEAS interconnections


Use Cases: Terrestrial Modeling
Software Productivity for Extreme-Scale Science
Methodologies for Software Productivity
Extreme-Scale Scientific Software Development Kit (xSDK)
Outreach and Community

- Use cases: Drive efforts. Traceability efforts
  - But generalized for future efforts
- Methodologies (“HowTo”) for SWP:
  - Infrastructure, testing, porting, refactoring, portability, etc.
  - Workflows, lifecycles: Document and formalize. Identify best practices
- xSDK: frameworks + components + libraries
  - Build apps by aggregation and composition
- Outreach: Foster communication, adoption, interaction
- First of a kind: Focus on <u>software productivity</u>

SW Engineering & Productivity

Sandia National Laboratories

Scientific Software Engineering

"A scientist builds in order to learn;
an engineer learns in order to build."

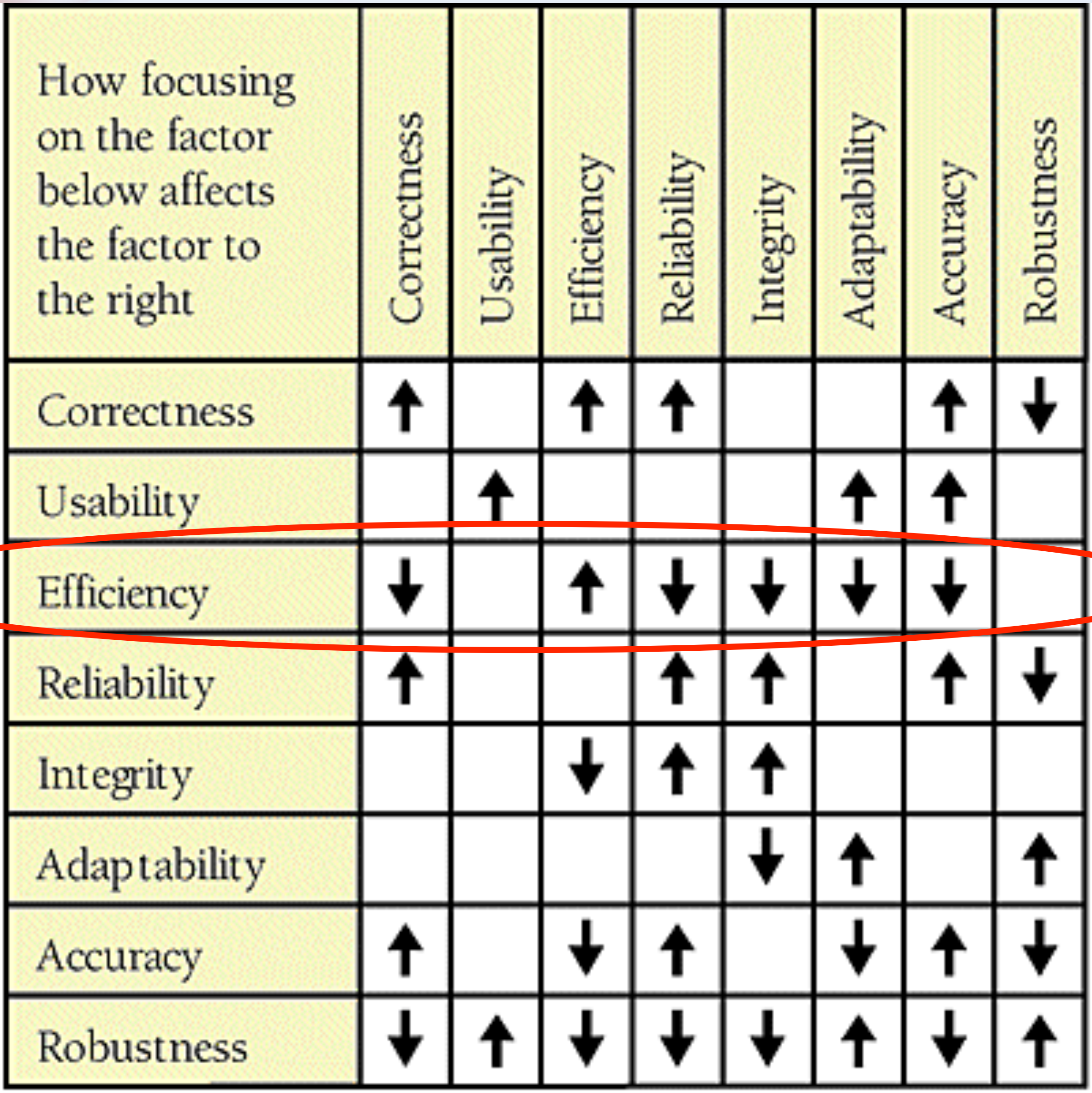
- Fred Brooks

Scientist: Barely-sufficient building.

Engineer: Barely-sufficient learning.

Both: Insufficiency leads to poor SW.

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 ↓

TriBITS: One Deliberate Approach to SE4CSE

Component-oriented SW Approach from Trilinos, CASL Projects, LifeV, …
Goal: “Self-sustaining” software

TriBITS Lifecycle Maturity Levels

0: Exploratory
1: Research Stable
2: Production Growth
3: Production Maintenance
-1: Unspecified Maturity

Goals

- *Allow Exploratory Research to Remain Productive*: Minimal practices for basic research in early phases

- *Enable Reproducible Research*: Minimal software quality aspects needed for producing credible research, researchers will produce better research that will stand a better chance of being published in quality journals that require reproducible research

- *Improve Overall Development Productivity*: Focus on the right SE practices at the right times, and the right priorities for a given phase/maturity level, developers work more productively with acceptable overhead

- *Improve Production Software Quality*: Focus on foundational issues first in early-phase development, higher-quality software will be produced as other elements of software quality are added

- *Better Communicate Maturity Levels with Customers*: Clearly define maturity levels so customers and stakeholders will have the right expectations

End of Life?

Long-term maintenance and end of life issues for Self-Sustaining Software:

- User community can help to maintain it (e.g., LAPACK).
- If the original development team is disbanded, users can take parts they are using and maintain it long term.
- Can stop being built and tested if not being currently used.
- However, if needed again, software can be resurrected, and continue to be maintained.

NOTE: Distributed version control using tools like Git greatly help in reducing risk and sustaining long lifetime.

Addressing existing Legacy Software

- One definition of "Legacy Software": Software that is too far from away from being Self-Sustaining Software, i.e:
  - Open-source
  - Core domain distillation document
  - Exceptionally well testing
  - Clean structure and code
  - Minimal controlled internal and external dependencies
  - Properties apply recursively to upstream software

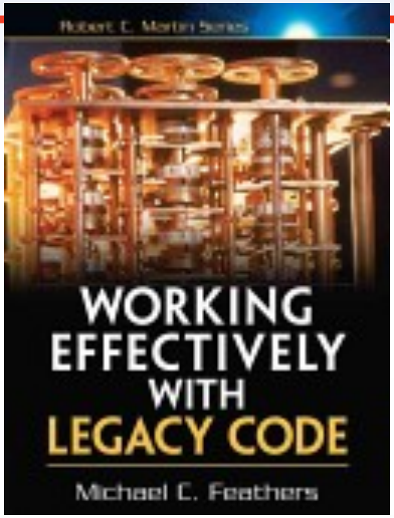
- Question: What about all the existing "Legacy" Software that we have to continue to develop and maintain? How does this lifecycle model apply to such software?

- Answer: Grandfather them into the TriBITS Lifecycle Model by applying the Legacy Software Change Algorithm.

Grandfathering of Existing Packages

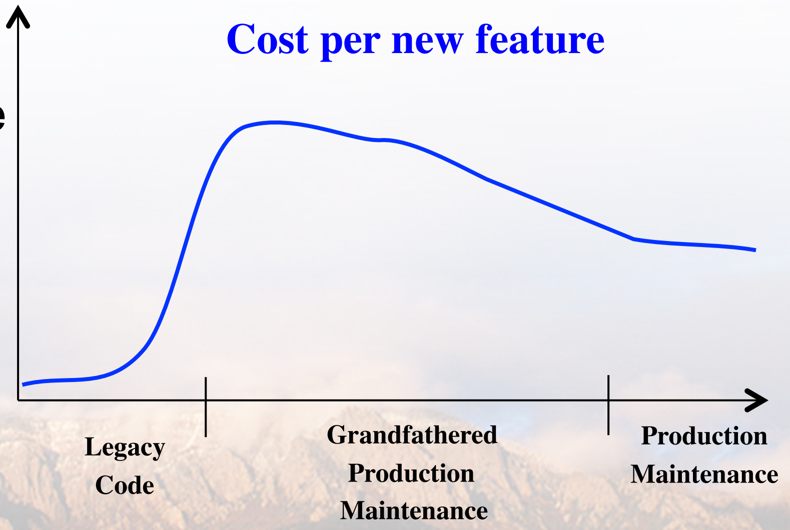
Agile Legacy Software Change Algorithm:

1. Identify Change Points
2. Break Dependencies
3. Cover with Unit Tests
4. Add New Functionality with Test Driven Development (TDD)
5. Refactor to removed duplication, clean up, etc.



Grandfathered Lifecycle Phases:

1. Grandfathered Research Stable (GRS) Code
2. Grandfathered Production Growth (GPG) Code
3. Grandfathered Production Maintenance (GPM) Code



NOTE: After enough iterations of the Legacy Software Change Algorithm the software may approach Self-Sustaining software and be able to remove the "Grandfathered" prefix.

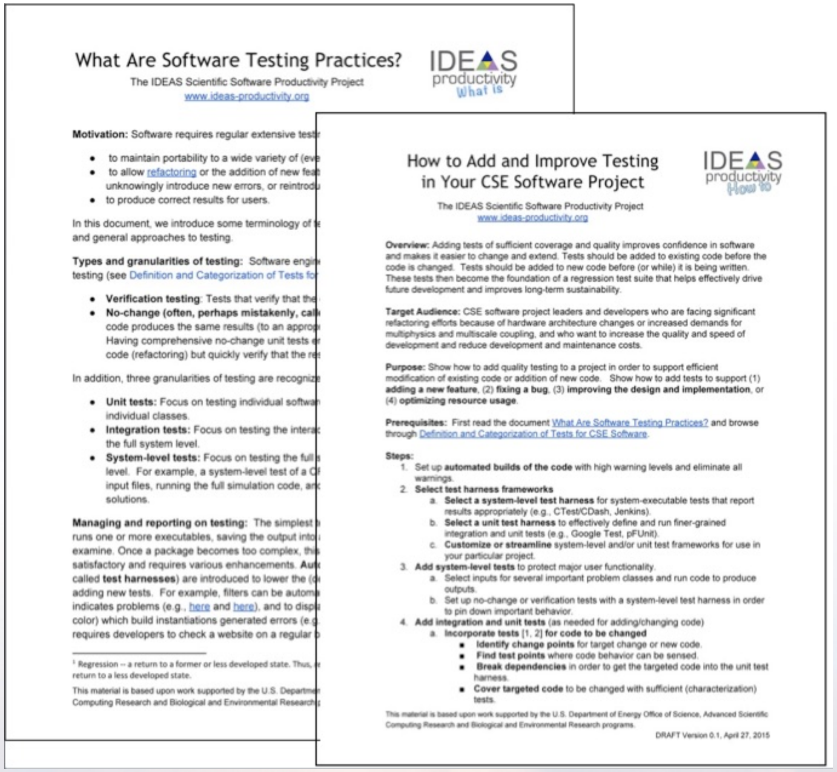
Message to This Audience

Write tests now, while (or before) writing your intended production software.

IDEAS ‘What is’ and ‘How to’ docs

- Motivation: Scientific software teams have a wide range of levels of maturity in software engineering practices
  - Baseline survey of *xSDK* and *BER Use Case* teams
- Approach:
  - ‘What Is’ docs: 2-page characterizations of important software project topics
  - ‘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.
- Initial emphasis:
  - *What is CSE Software Productivity?*
  - *What are Software Testing Practices?*
  - *How to Add and Improve Testing in Your CSE Software Project*
- Topics in progress:
  - Refactoring tools and approaches
  - Best practices for using interoperable libraries
  - Designing for performance portability
  - Etc.

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



What Are Software Testing Practices?

The IDEAS Scientific Software Productivity Project
www.ideas-productivity.org

Motivation: Software requires regular extensive test

- to maintain portability to a wide variety of (eve
- to allow refactoring or the addition of new feat
  unknowingly introduce new errors, or reintrodu
- to produce correct results for users.

In this document, we introduce some terminology of te
and general approaches to testing.

Types and granularities of testing: Software engi
testing (see Definition and Categorization of Tests for

- **Verification testing**: Tests that verify that the
- **No-change (often, perhaps mistakenly, call**
  code produces the same results (to an approp
  Having comprehensive no-change unit tests e
  code (refactoring) but quickly verify that the re

In addition, three granularities of testing are recognize

- **Unit tests:** Focus on testing individual softwar
  individual classes.
- **Integration tests:** Focus on testing the interac
  the full system level.
- **System-level tests:** Focus on testing the full s
  level. For example, a system-level test of a C
  input files, running the full simulation code, an
  solutions.

Managing and reporting on testing: The simplest
runs one or more executables, saving the output into
examine. Once a package becomes too complex, this
satisfactory and requires various enhancements. **Au**
called **test harnesses**) are introduced to lower the (
adding new tests. For example, filters can be autom
indicates problems (e.g., here and here), and to disp
color) which build instantiations generated errors (e.g
requires developers to check a website on a regular

[1] Regression -- a return to a former or less developed state. Thus,
return to a less developed state.

This material is based upon work supported by the U.S. Departme
Computing Research and Biological and Environmental Research

How to Add and Improve Testing in Your CSE Software Project

The IDEAS Scientific Software Productivity Project
www.ideas-productivity.org

Overview: Adding tests of sufficient coverage and quality improves confidence in software and makes it easier to change and extend. Tests should be added to existing code before the code is changed. Tests should be added to new code before (or while) it is being written. These tests then become the foundation of a regression test suite that helps effectively drive future development and improves long-term sustainability.

Target Audience: CSE software project leaders and developers who are facing significant refactoring efforts because of hardware architecture changes or increased demands for multiphysics and multiscale coupling, and who want to increase the quality and speed of development and reduce development and maintenance costs.

Purpose: Show how to add quality testing to a project in order to support efficient modification of existing code or addition of new code. Show how to add tests to support (1) **adding a new feature**, (2) **fixing a bug**, (3) **improving the design and implementation**, or (4) **optimizing resource usage**.

Prerequisites: First read the document What Are Software Testing Practices? and browse through Definition and Categorization of Tests for CSE Software.

Steps:

1. Set up **automated builds of the code** with high warning levels and eliminate all warnings.
2. **Select test harness frameworks**
   a. **Select a system-level test harness** for system-executable tests that report results appropriately (e.g., CTest/CDash, Jenkins).
   b. **Select a unit test harness** to effectively define and run finer-grained integration and unit tests (e.g., Google Test, pFUnit).
   c. **Customize or streamline** system-level and/or unit test frameworks for use in your particular project.
3. **Add system-level tests** to protect major user functionality.
   a. Select inputs for several important problem classes and run code to produce outputs.
   b. Set up no-change or verification tests with a system-level test harness in order to pin down important behavior.
4. **Add integration and unit tests** (as needed for adding/changing code)
   a. **Incorporate tests** [1, 2] **for code to be changed**
      - **Identify change points** for target change or new code.
      - **Find test points** where code behavior can be sensed.
      - **Break dependencies** in order to get the targeted code into the unit test harness.
      - **Cover targeted code** to be changed with sufficient (characterization) tests.

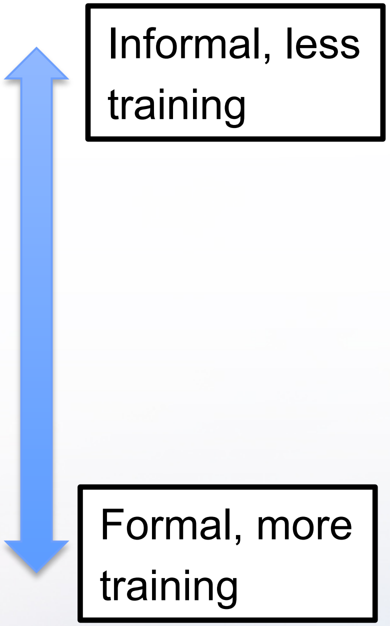
This material is based upon work supported by the U.S. Department of Energy Office of Science, Advanced Scientific Computing Research and Biological and Environmental Research programs.

DRAFT Version 0.1, April 27, 2015

Impact: Provide baseline nomenclature and foundation for next steps in SW productivity and SW engineering for CSE teams

Managing issues:
Fundamental software process

- Issue: Bug report, feature request
- Approaches:
  - Short-term memory, office notepad
  - ToDo.txt on computer desktop (1 person)
  - Issues.txt in repository root (small co-located team)
  - …
  - Web-based tool + Kanban (distributed, larger team)
  - Web-based tool + Scrum (full-time dev team)



Informal, less training

Formal, more training

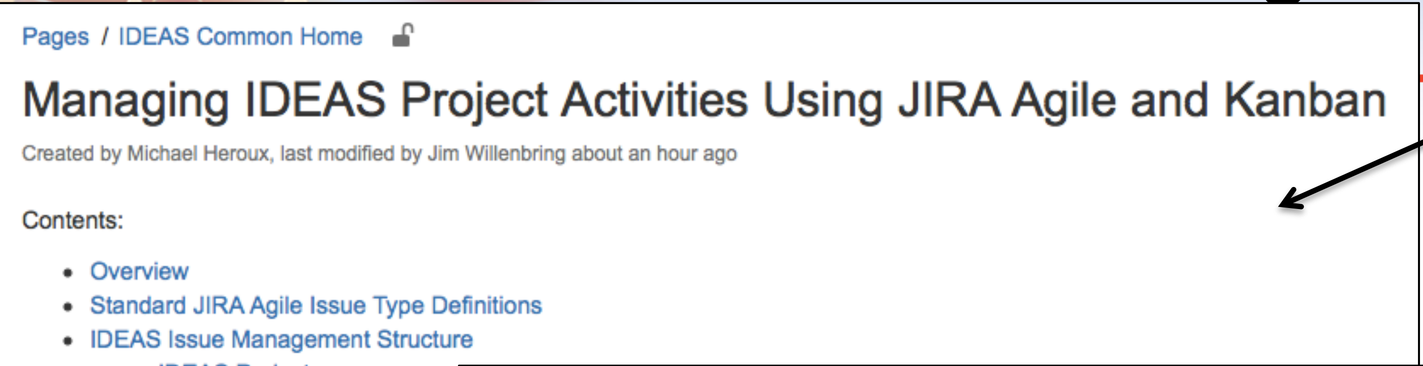
- IDEAS project:
  - Jira Agile + Confluence: Turnkey web platform (ACME too)
  - Kanban: Simplest of widely known Agile SW dev processes

Kanban principles

- Limit number of “In Progress” tasks
- Productivity improvement:
  - Optimize “flexibility vs swap overhead” balance. No overcommitting.
  - Productivity weakness exposed as bottleneck. Team must identify and fix the bottleneck.
  - Effective in R&D setting. Avoids a deadline-based approach. Deadlines are dealt with in a different way.
- Provides a board for viewing and managing issues

Task: Have Eureka moment by Tuesday.

IDEAS Confluence, Jira Agile, Kanban



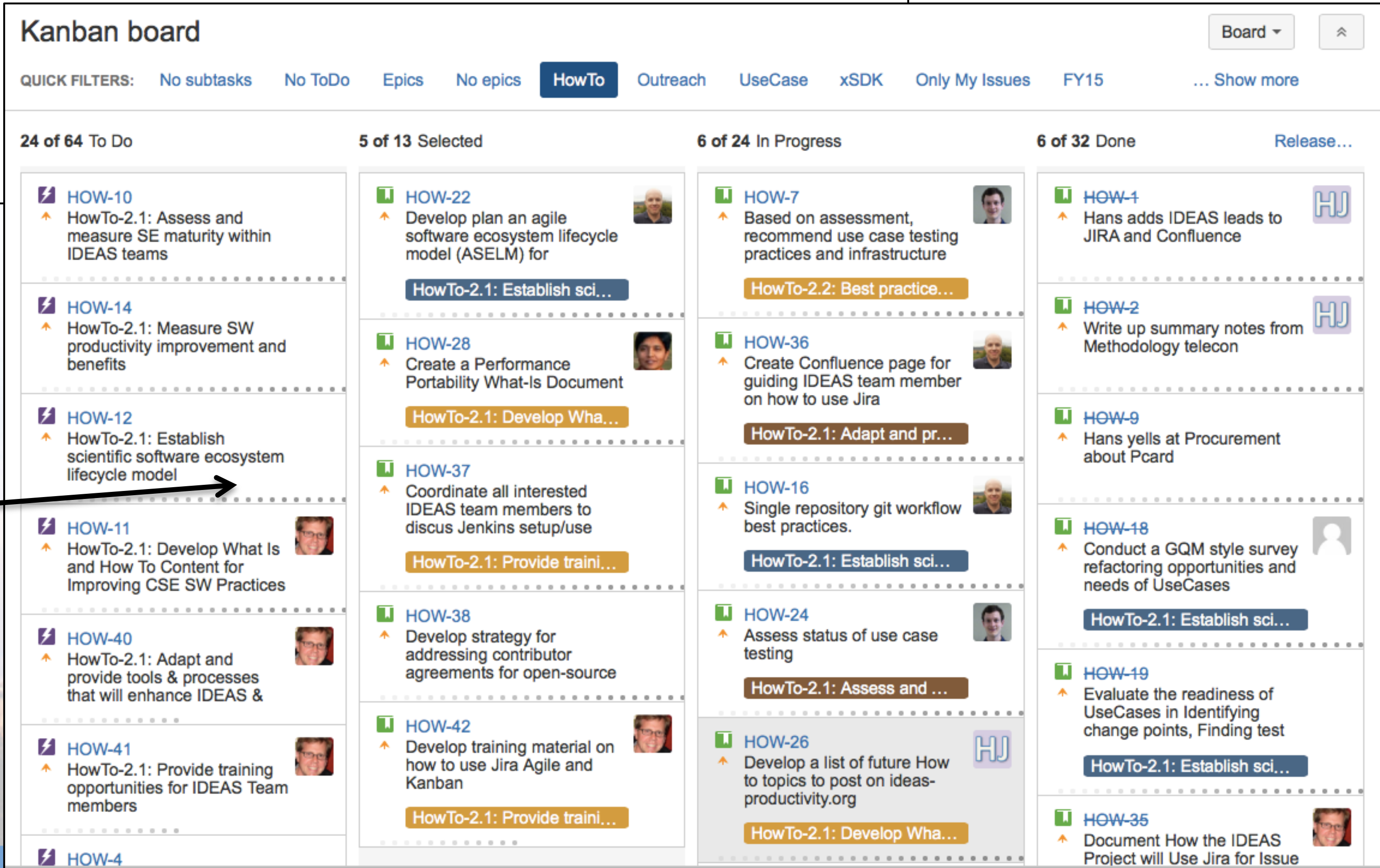
Pages / IDEAS Common Home

Managing IDEAS Project Activities Using JIRA Agile and Kanban

Created by Michael Heroux, last modified by Jim Willenbring about an hour ago

Contents:

Developer Guide, on Confluence site



Kanban board

Board
QUICK FILTERS: No subtasks · No ToDo · Epics · No epics · HowTo · Outreach · UseCase · xSDK · Only My Issues · FY15 · ... Show more
Release...

24 of 64 To Do	5 of 13 Selected	6 of 24 In Progress	6 of 32 Done
HOW-10 HowTo-2.1: Assess and measure SE maturity within IDEAS teams	HOW-22 Develop plan an agile software ecosystem lifecycle model (ASELM) for HowTo-2.1: Establish sci...	HOW-7 Based on assessment, recommend use case testing practices and infrastructure HowTo-2.2: Best practice...	~~HOW-1~~ Hans adds IDEAS leads to JIRA and Confluence
HOW-14 HowTo-2.1: Measure SW productivity improvement and benefits	HOW-28 Create a Performance Portability What-Is Document HowTo-2.1: Develop Wha...	HOW-36 Create Confluence page for guiding IDEAS team member on how to use Jira HowTo-2.1: Adapt and pr...	~~HOW-2~~ Write up summary notes from Methodology telecon
HOW-12 HowTo-2.1: Establish scientific software ecosystem lifecycle model	HOW-37 Coordinate all interested IDEAS team members to discus Jenkins setup/use HowTo-2.1: Provide traini...	HOW-16 Single repository git workflow best practices. HowTo-2.1: Establish sci...	~~HOW-9~~ Hans yells at Procurement about Pcard
HOW-11 HowTo-2.1: Develop What Is and How To Content for Improving CSE SW Practices	HOW-38 Develop strategy for addressing contributor agreements for open-source	HOW-24 Assess status of use case testing HowTo-2.1: Assess and ...	~~HOW-18~~ Conduct a GQM style survey refactoring opportunities and needs of UseCases HowTo-2.1: Establish sci...
HOW-40 HowTo-2.1: Adapt and provide tools & processes that will enhance IDEAS &	HOW-42 Develop training material on how to use Jira Agile and Kanban HowTo-2.1: Provide traini...	HOW-26 Develop a list of future How to topics to post on ideas-productivity.org HowTo-2.1: Develop Wha...	~~HOW-19~~ Evaluate the readiness of UseCases in Identifying change points, Finding test HowTo-2.1: Establish sci...
HOW-41 HowTo-2.1: Provide training opportunities for IDEAS Team members			~~HOW-35~~ Document How the IDEAS Project will Use Jira for Issue
HOW-4			

Kanban Board, on Jira site.

Four columns:

- To Do
- Selected
- In Progress
- Done

Message to This Audience

Improve your issue tracking habits:

- *Nothing -> Desktop/todo.txt*
- *Desktop/todo.txt -> clone/todo.txt*
- *clone/todo.txt -> Git Issues*
- *Git Issues -> Git Issues + Kanban*
  or Jira + Kanban

Three Application Design Strategies for Productivity & Sustainability

Strategy 1: Array and Execution Abstraction

Sandia National Laboratories

Multi-dimensional Dense Arrays

- Many computations work on data stored in multi-dimensional arrays:
  - – Finite differences, volumes, elements.
  - – Sparse iterative solvers.
- Dimension are (k,l,m,…) where one dimension is long:
  - – A(3,1000000)
  - – 3 degrees of freedom (DOFs) on 1 million mesh nodes.
- A classic data structure issue is:
  - – Order by DOF: A(1,1), A(2,1), A(3,1); A(1,2) … or
  - – By node: A(1,1), A(1,2), …
- Adherence to raw language arrays forces a choice.
- **Physics i,j,k should not dictate storage i,j,k.**

Sandia National Laboratories

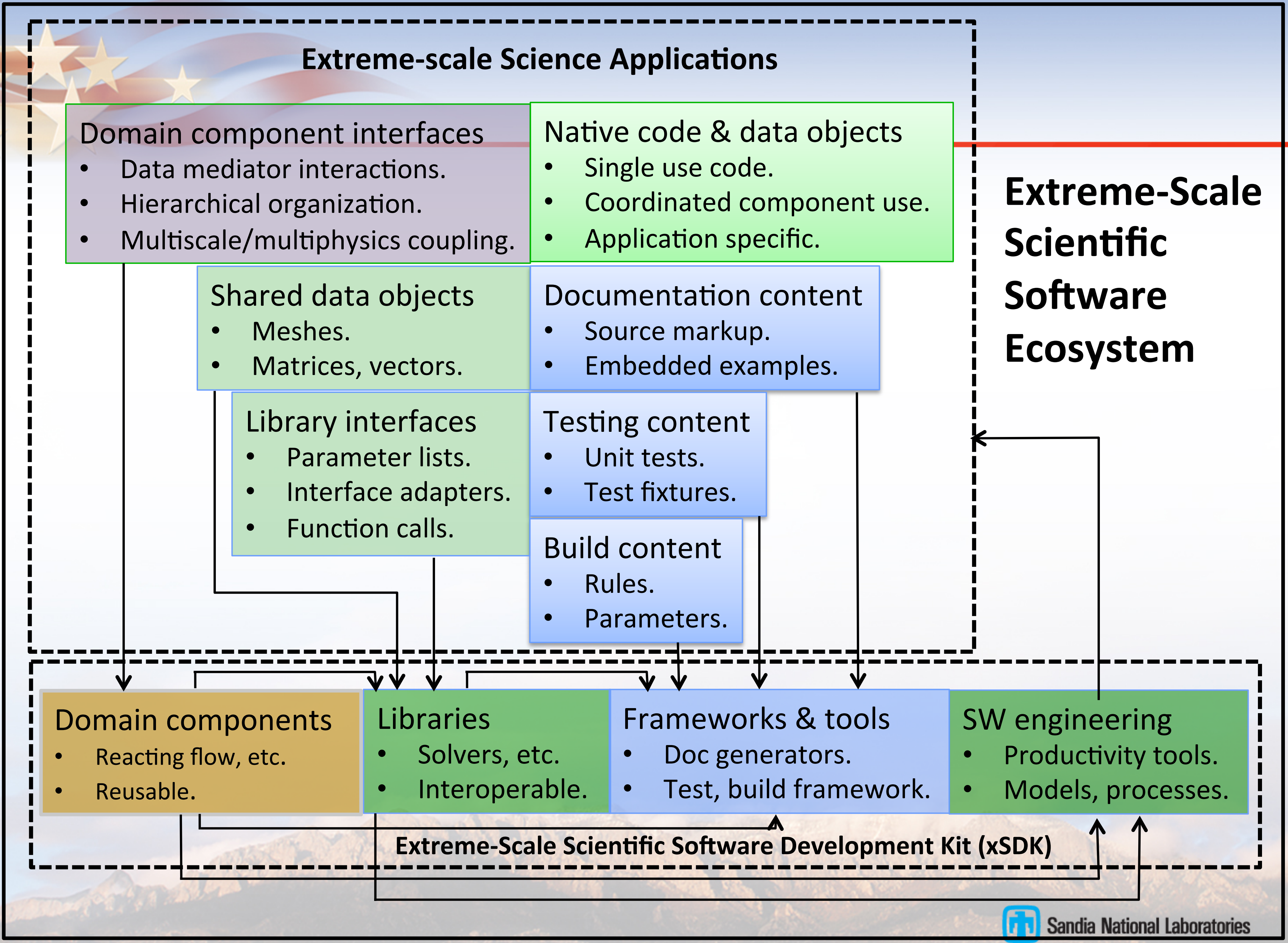
Kokkos: Execution and memory space abstractions

- What is Kokkos:
  - C++ (C++11) template meta-programming library, part of (and not) Trilinos.
  - Compile-time polymorphic multi-dimensional array classes.
  - Parallel execution patterns: For, Reduce, Scan.
  - Loop body code: Functors, lambdas.
  - Tasks: Asynchronous launch, Futures.
- Available independently (outside of Trilinos):
  - https://github.com/kokkos/
- Getting started:
  - GTC 2015 Content:
    - http://on-demand.gputechconf.com/gtc/2015/video/S5166.html
    - http://on-demand.gputechconf.com/gtc/2015/presentation/S5166-H-Carter-Edwards.pdf
  - Programming guide doc/Kokkos_PG.pdf.

Message to This Audience

Consider an array/patterns library, e.g., Kokkos.

Strategy 2: Application Composition



Extreme-Scale Scientific Software Ecosystem

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 Development Kit (xSDK)

Domain components

- Reacting flow, etc.
- Reusable.

Libraries

- Solvers, etc.
- Interoperable.

Frameworks & tools

- Doc generators.
- Test, build framework.

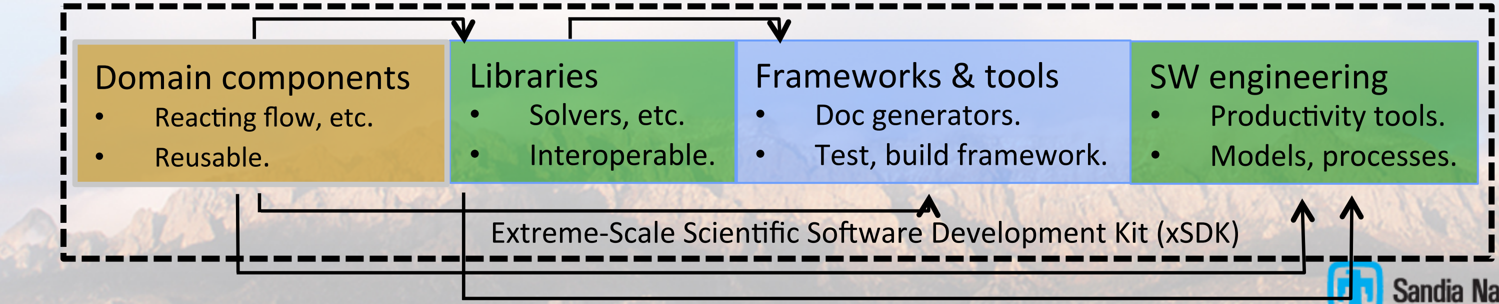
SW engineering

- Productivity tools.
- Models, processes.

Sandia National Laboratories

xSDK focus

- Common configure and link capabilities
  - xSDK users need full and consistent access to all xSDK capabilities
  - Namespace and version conflicts make simultaneous build/ link of xSDK difficult
  - Determining an approach that can be adopted by any library or components development team for standardized configure/link processes
- Library interoperability
- Designing for performance portability



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)

Standard xSDK package installation interface

Motivation: Obtaining, configuring, and installing multiple independent software packages is tedious and error prone.

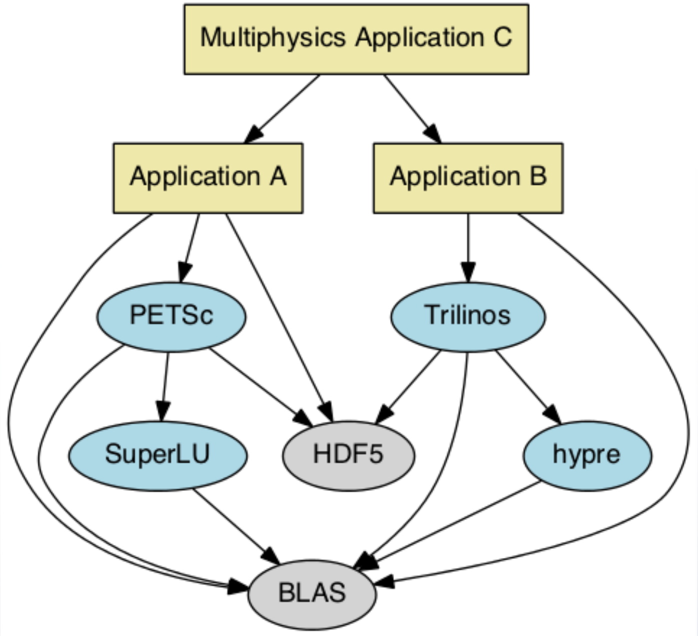
- Need consistency of compiler (+version, options), 3rd-party packages, etc.

Approach: Define a *standard xSDK package installation interface* to which all xSDK packages will subscribe and be tested

Accomplishments:

- Work on implementations of the standard by the hypre, PETSc, SuperLU, and Trilinos developers
- PETSc can now use the “scriptable” feature of the installers to simultaneously install hypre, PETSc, SuperLU, Trilinos with consistent compilers and ‘helper’ libraries.

xSDK Build Example

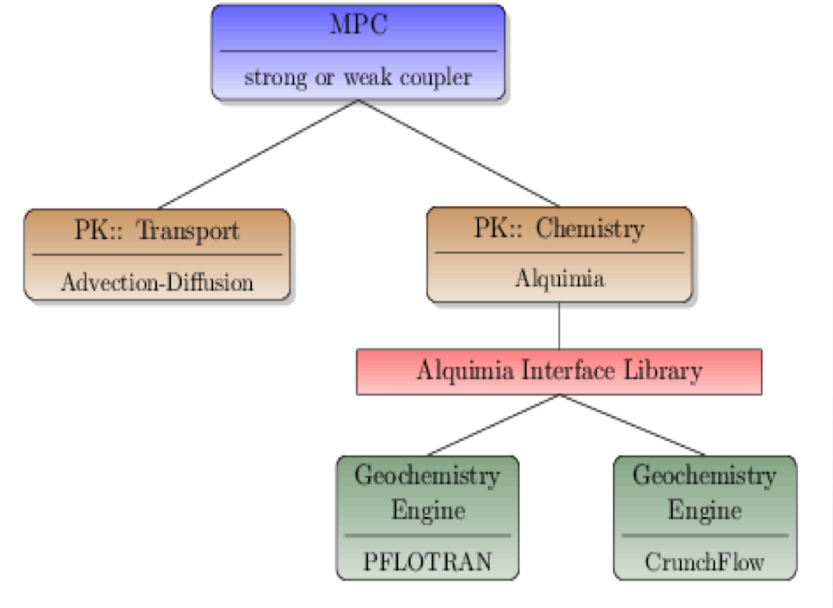


Impact: Foundational step toward seamless combined use of xSDK libraries, as needed by BER use cases and other multiphysics apps

Enabling Interoperable Biogeochemistry with Alquimia

Several geochemistry libraries are well established in the community making geochemistry ideal to explore componentization and interface design. *Alquimia* is an interface library, and does not perform any reaction calculations.

- □ Alquimia currently assumes reactive transport uses operator-splitting.
- □ Fully-implicit reactive transport support is being developed in collaboration with IDEAS.
- □ Assists in enforcing geochemical conditions (speciation) for transport boundary conditions
- □ Alquimia can facilitate benchmarking of geochemical capabilities in existing codes.
- □ Geochemistry libraries, such as PFLOTRAN and CrunchFlow, have implemented interfaces to Alquimia.



Schematic of the Alquimia interface library providing uniform access to PFLOTRAN and CrunchFlow geochemistry in Amanzi

Alquimia is open source, https://bitbucket.org/berkeleylab/alquimia

TRILINOS COMMUNITY 2.0

Trilinos Community 2.0

- GitHub, Atlassian:
  - Open source SW development, tools platforms.
  - Workflows for high-quality community SW product development.
- Trilinos value proposition:
  - Included these same things, but are re-evaluating.
  - Moving to GitHub.
  - Supporting dual-mode package model.
- New types of Trilinos packages:
  - Internal: Available only with Trilinos (traditional definition).
  - Exported: Developed in Trilinos repository, available externally.
  - Imported: Developed outside of Trilinos, available internally.

Trilinos Community 2.0

- Case studies:
  - TriBITS: Was an internal package, now external.
  - DTK: Has always been external.
  - Kokks: Was internal. Is now developed externally, available internal.
  - Move to GitHub: Several packages splitting off.
- Issues to Resolve:
  - Package inclusion policies: Define for each package type.
  - Quality criteria: Contract between Trilinos and packages.
  - Workflows: Development, testing, documentation, etc.
  - Trilinos on GitHub: Almost there.
  - Trilinos Value Proposition: Re-articulate Trilinos Strategic Goals implications.

Common Look-and-feel Expectations

- Consistent data management practices.
- Consistent API styles.
- Testing and other quality metric thresholds, e.g., coverity.
- What else?

xSDK Minimum Compliance Requirements:

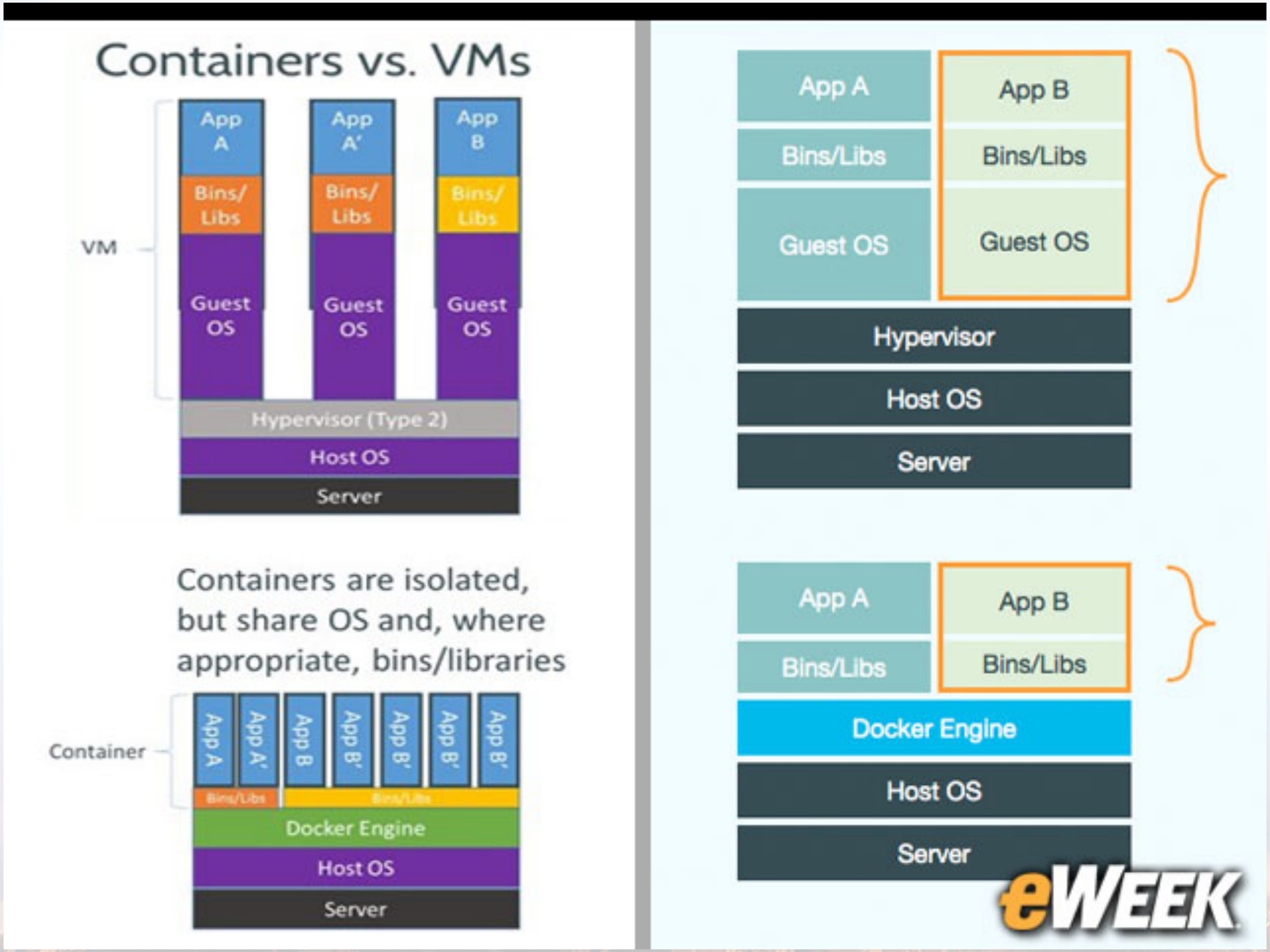
- M1. Each xSDK compliant package must support the the standard xSDK cmake/configure options.
- M2. Each xSDK package must provide a comprehensive test suite that can be run by users and does not require the purchase of commercial software
- M3. Each xSDK compliant package that utilizes MPI must restrict its MPI operations to MPI communicators that are provided to it and not use directly MPI_COMM_WORLD.
- M4. Each package team must do a 'best effort' at portability to key architectures, including standard Linux distributions, GNU, Clang, vendor compilers, and target machines at ALCF, NERSC, OLCF. Apple Mac OS and Microsoft Windows support are recommended.
- M5. Each package team must provide a documented, reliable way to contact the development team; this may be by email or a website. The package teams should not require users to join a generic mailing list (and hence receive irrelevant email they must wade through) in order to report bugs or request assistance.
- M6 – 11…

https://ideas-productivity.org/resources/xsdk-docs: Open for public comment.

xSDK Recommended Compliance Requirements:

- R1. It is recommended that each package have a public repository, for example at github or bitbucket, where the development version of the package is available. Support for taking pull requests is also recommended.
- R2. It is recommend that all libraries be tested with valgrind for memory corruption issues while the test suite is run.
- R3. It is recommended that each package adopt and document a consistent system for propagating/returning error conditions/exceptions and provide an API for changing the behavior.
- R4. It is recommended that each package free all system resources it has acquired as soon as they are no longer needed.

Docker (about which I know little)



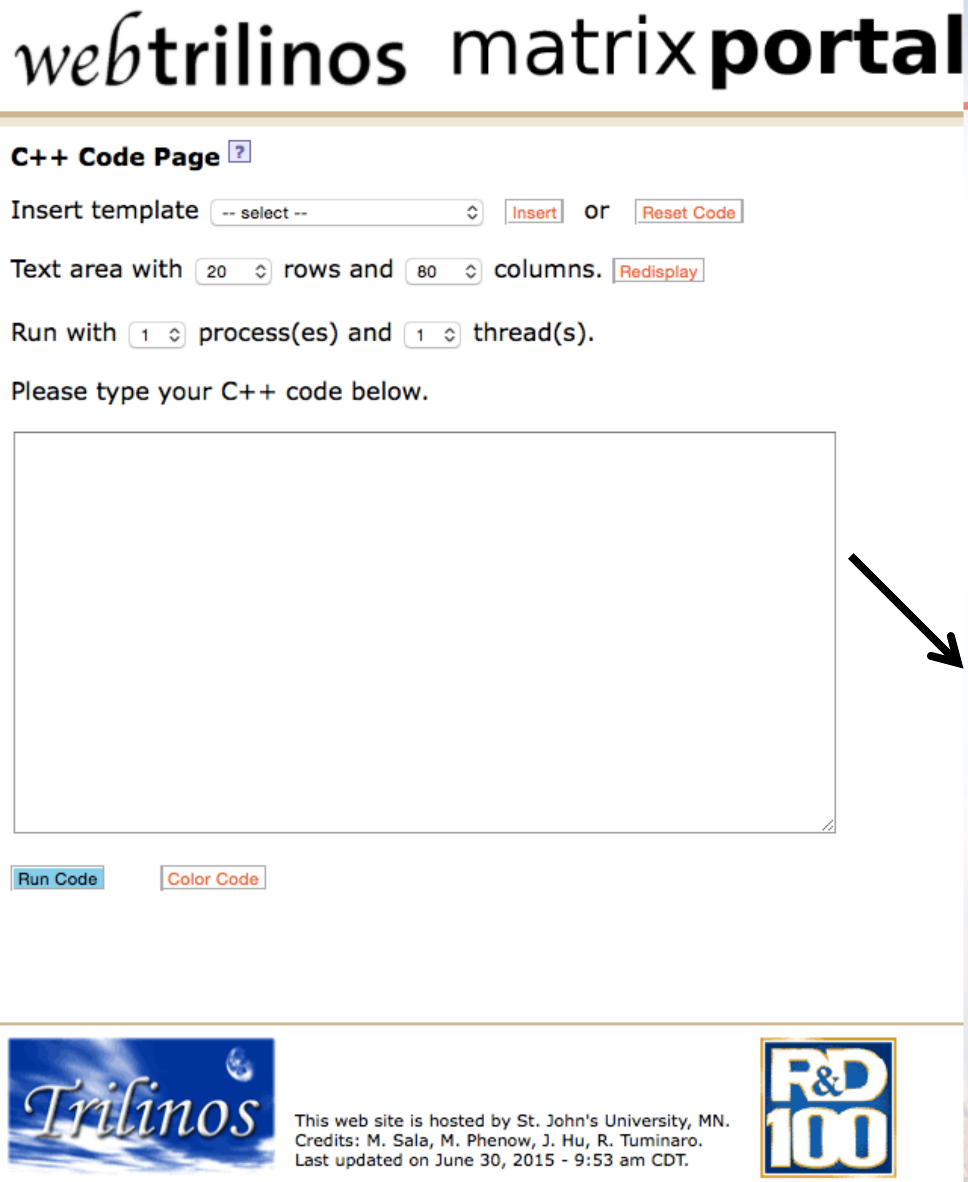
Containers vs. VMs

Containers are isolated, but share OS and, where appropriate, bins/libraries

Typical Trilinos Cmake Script (edison)

```
cmake \
-D MPI_CXX_COMPILER="CC" \
-D MPI_C_COMPILER="cc" \
-D MPI_Fortran_COMPILER="ftn" \
-D Teuchos_ENABLE_STACKTRACE:BOOL=OFF \
-D Teuchos_ENABLE_LONG_LONG_INT:BOOL=ON \
-D Trilinos_ENABLE_Tpetra:BOOL=ON \
-D Tpetra_ENABLE_TESTS:BOOL=ON \
-D Tpetra_ENABLE_EXAMPLES:BOOL=ON \
-D Tpetra_ENABLE_EXPLICIT_INSTANTIATION:BOOL=ON \
-D Teuchos_ENABLE_EXPLICIT_INSTANTIATION:BOOL=ON \
-D TPL_ENABLE_MPI:BOOL=ON \
-D CMAKE_INSTALL_PREFIX:PATH="$HOME/opt/Trilinos/tpetraEval" \
-D BLAS_LIBRARY_DIRS="$LIBSCI_BASE_DIR/gnu/lib" \
-D BLAS_LIBRARY_NAMES="sci" \
-D LAPACK_LIBRARY_DIRS="$LIBSCI_BASE_DIR/gnu/lib" \
-D LAPACK_LIBRARY_NAMES="sci" \
-D CMAKE_CXX_FLAGS="-O3 -ffast-math -funroll-loops" \
\
..
```

WebTrilinos



webtrilinos matrix portal

C++ Code Page ?

Insert template [-- select --] Insert or Reset Code

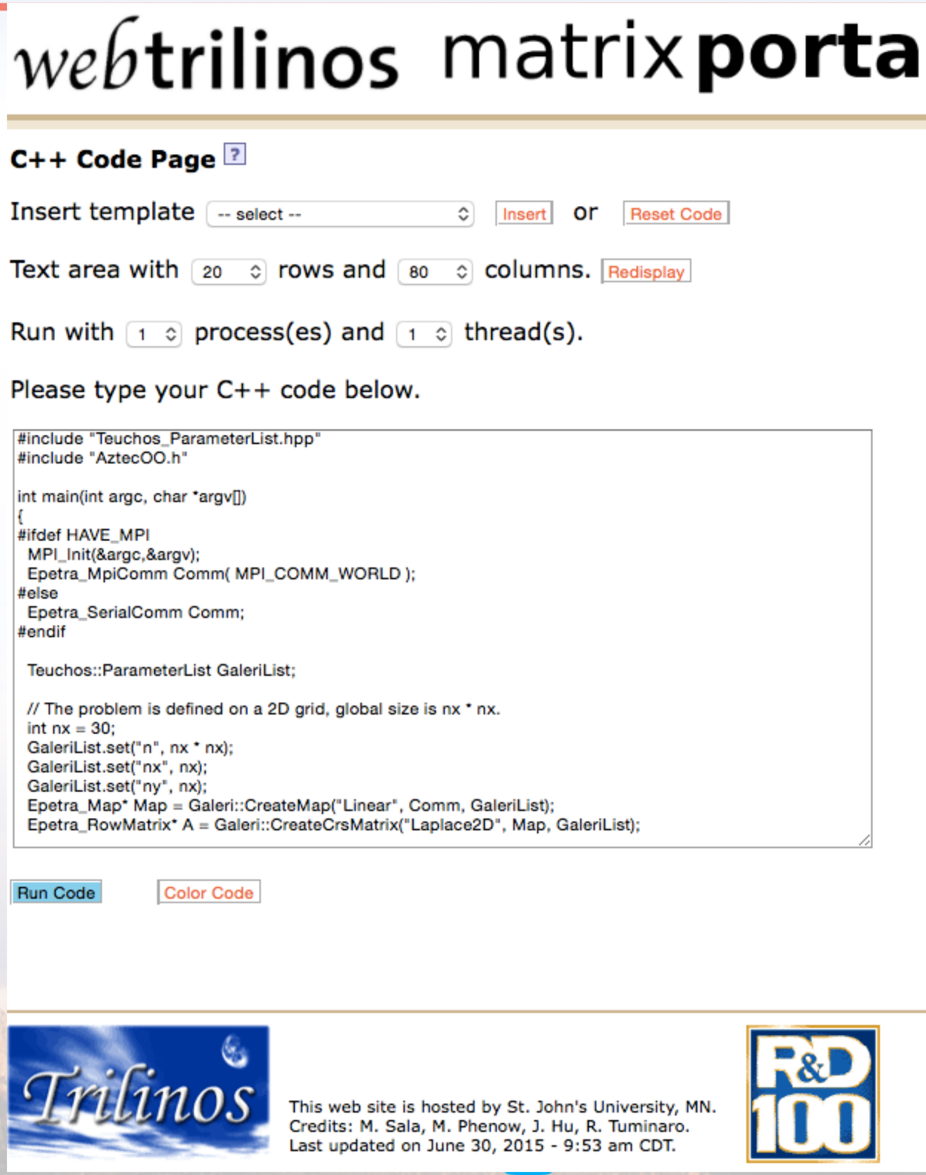
Text area with [20] rows and [80] columns. Redisplay

Run with [1] process(es) and [1] thread(s).

Please type your C++ code below.

Run Code Color Code

This web site is hosted by St. John's University, MN.
Credits: M. Sala, M. Phenow, J. Hu, R. Tuminaro.
Last updated on June 30, 2015 - 9:53 am CDT.



webtrilinos matrix porta

C++ Code Page ?

Insert template [-- select --] Insert or Reset Code

Text area with [20] rows and [80] columns. Redisplay

Run with [1] process(es) and [1] thread(s).

Please type your C++ code below.

```
#include "Teuchos_ParameterList.hpp"
#include "AztecOO.h"

int main(int argc, char *argv[])
{
#ifdef HAVE_MPI
  MPI_Init(&argc,&argv);
  Epetra_MpiComm Comm( MPI_COMM_WORLD );
#else
  Epetra_SerialComm Comm;
#endif

  Teuchos::ParameterList GaleriList;

  // The problem is defined on a 2D grid, global size is nx * nx.
  int nx = 30;
  GaleriList.set("n", nx * nx);
  GaleriList.set("nx", nx);
  GaleriList.set("ny", nx);
  Epetra_Map* Map = Galeri::CreateMap("Linear", Comm, GaleriList);
  Epetra_RowMatrix* A = Galeri::CreateCrsMatrix("Laplace2D", Map, GaleriList);
```

Run Code Color Code

This web site is hosted by St. John's University, MN.
Credits: M. Sala, M. Phenow, J. Hu, R. Tuminaro.
Last updated on June 30, 2015 - 9:53 am CDT.

Trilinos usage via Docker

- WebTrilinos Tutorial
  - https://hub.docker.com/r/sjdeal/webtrilinos
- http://johntfoster.github.io/posts/peridigm-without-building-via-Docker.html
  - docker pull johntfoster/trilinos
  - docker pull johntfoster/peridigm
  - docker run --name peridigm0 -d -v `pwd`:/output johntfoster/peridigm \

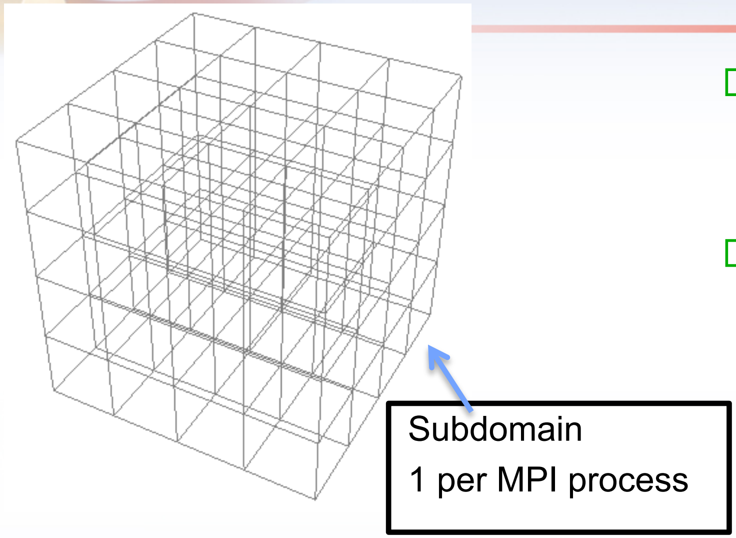
    ```
            Peridigm fragmenting_cylinder.peridigm
    ```
  - Etc…

Message to This Audience

Consider what software ecosystem(s) you want your software to be part of and use.

Strategy 3: Toward a New Application Architecture

Classic HPC Application Architecture


Subdomain
1 per MPI process

- Logically Bulk-Synchronous, SPMD
- Basic Attributes:
  - Halo exchange.
  - Local compute.
  - Global collective.

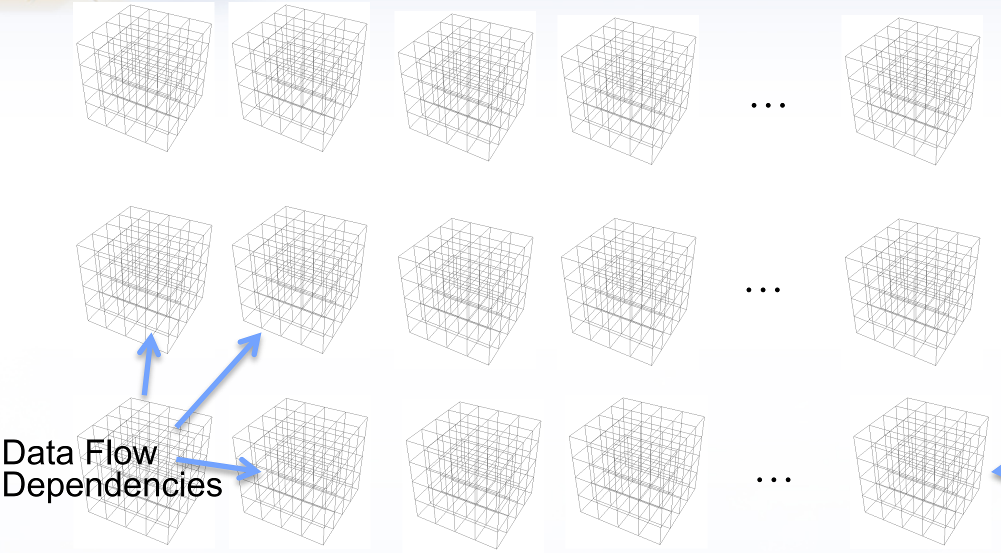
- Strengths:
  - Portable to many specific system architectures.
  - Separation of parallel model (SPMD) from implementation (e.g., message passing).
  - Domain scientists write sequential code within a parallel SPMD framework.
  - Supports traditional languages (Fortran, C).
  - Many more, well known.

- Weaknesses:
  - Not well suited (as-is) to emerging manycore systems.
  - Unable to exploit functional on-chip parallelism.
  - Difficult to tolerate dynamic latencies.
  - Difficult to support task/compute heterogeneity.

Sandia National Laboratories

Task-centric/Dataflow Application Architecture

- Patch: Logically connected portion of global data. Ex: subdomain, subgraph.
- Task: Functionality defined on a patch.
- Many tasks on many patches.



Data Flow Dependencies

Patch
Many per MPI process

- Strengths:
  - Portable to many specific system architectures.
  - Separation of parallel model from implementation.
  - Domain scientists write sequential code within a parallel framework.
  - Supports traditional languages (Fortran, C).
  - Similar to SPMD in many ways.

- More strengths:
  - Well suited to emerging manycore systems.
  - Can exploit functional on-chip parallelism.
  - Can tolerate dynamic latencies.
  - Can support task/compute heterogeneity.

Task on a Patch

- Patch: Small subdomain or subgraph.
  - Big enough to run efficiently once its starts execution.
    - CPU core: Need ~1 millisecond for today’s best runtimes (e.g. Legion).
    - GPU: Give it big patches. GPU runtime does manytasking very well on its own.
- Task code (Domain scientist writes most of this code):
  - Standard Fortran, C, C++ code.
  - E.g. FEM stiffness matrix setup on a “workset” of elements.
  - Should vectorize (CPUs) or SIMT (GPUs).
  - Should have small thread-count parallel (OpenMP)
    - Take advantage of shared cache/DRAM for UMA cores.
  - Source line count of task code should be tunable.
    - Too coarse grain task:
      - GPU: Too much register state, register spills.
      - CPU: Poor temporal locality. Not enough tasks for latency hiding.
    - Too fine grain:
      - Too much overhead or
      - Patches too big to keep task execution at 1 millisec.

Portable Task Coding Environment

- Task code must run on many types of cores:
  - Standard multicore (e.g., Haswell).
  - Manycore (Intel PHI, KNC, KNL).
  - GPU (Nvidia).
- Desire:
  - Write single source.
  - Compile phase adapts for target core type.
  - Sounds like what?
- Kokkos (and others: OCCA, RAJA, …):
  - Enable meta programming for multiple target core architectures.
- Future: Fortran/C/C++ with OpenMP 4:
  - Limited execution patterns, but very usable.
  - Like programming MPI codes today: Déjà vu for domain scientists.
- Other future: C++ with Kokkos/OCCA/RAJA derivative in std namespace.
  - Broader execution pattern selection, more complicated.

Task Management Layer

- New layer in application and runtime:
  - Enables (async) task launch: latency hiding, load balancing.
  - Provides technique for declaring inter-task dependencies:
    - Data read/write (Legion).
      - Task A writes to variable x, B depends on x. A must complete before B starts.
    - Futures:
      - Explicit encapsulation of dependency. Task B depends on A's future.
    - Alternative: Explicit DAG management.
  - Aware of temporal locality:
    - Better to run B on the same core as A to exploit cache locality.
  - Awareness of data staging requirements:
    - Task should not be scheduled until its data are ready:
      - If B depends on remote data (retrieved by A).
  - Manage heterogeneous execution: A on Haswell, B on PHI.
  - Resilience: If task A launched task B, A can relaunch B if B fails or times out.
- What are the app vs. runtime responsibilities?
- How can each assist the other?

Open Questions for Task-Centric/Dataflow Strategies

- Functional vs. Data decomposition.
  - Over-decomposition of spatial domain:
    - Clearly useful, challenging to implement.
  - Functional decomposition:
    - Easier to implement. Challenging to execute efficiently (temporal locality).
- Dependency specification mechanism.
  - How do apps specify inter-task dependencies?
  - Futures (e.g., C++, HPX), data addresses (Legion), explicit (Uintah).
- Roles & Responsibilities: App vs Libs vs Runtime vs OS.
- Interfaces between layers.
- Huge area of R&D for many years.

Data challenges:

- Read/write functions:
  - Must be task compatible.
  - Thread-safe, non-blocking, etc.
- Versioning:
  - Computation may be executing across multiple logically distinct phases (e.g. timesteps)
  - Example: Data must exist at each grid point and for all active timesteps.
- Global operations:
  - Coordination across task events.
  - Example: Completion of all writes at a time step.

Sandia National Laboratories

Execution Policy for Task Parallelism

- TaskManager< ExecSpace > execution policy
  - Policy object shared by potentially concurrent tasks

```
TaskManager<...> tm( exec_space , ... );
Future<> fa = spawn( tm , task_functor_a ); // single-thread task
Future<> fb = spawn( tm , task_functor_b );
```

  - Tasks may be data parallel

```
Future<> fc = spawn_for( tm.range(0..N) , functor_c );
Future<value_type> fd = spawn_reduce( tm.team(N,M) , functor_d );
wait( tm ); // wait for all tasks to complete
```

  - Destruction of task manager object waits for concurrent tasks to complete
- Task Managers
  - Define a scope for a collection of potentially concurrent tasks
  - Have configuration options for task management and scheduling
  - Manage resources for scheduling queue

Kokkos/Qthread LDRD

Manytasking: A Productive Application Architecture

- Atomic Unit: Task
  - Domain scientist writes code for a task.
  - Task execution requirements:
    - Tunable work size: Enough to efficiently use a core once scheduled.
    - Vector/SIMT capabilities.
- Utility of Task-based Approach:
  - Oversubscription: Latency hiding, load balancing.
  - Dataflow: Task-DAG or futures.
  - Resilience: Re-dispatch task from parent.
  - Déjà vu for apps developers: Feels a lot like MPI programming.
  - Universal portability: Works within node, across nodes.

Manytasking Implications

- Parallel Programming:
  - Task is small thread, vector/SIMT parallel only. (Fortran can do this, including the new Open Source LLVM-based Fortran!).
  - Parallel Task management is external concern.
- Task scheduling:
  - Runtime: Many tasks per node. Many tasks in-flight.
  - Parallelism across node components: Really important.
  - Issue: How to manage creation/completion rates.
- Resilience:
  - How to coordinate task protection (parent), re-dispatch (child).

Sandia National Laboratories

Message to This Audience

- *Where does your software fit in a manytasking application framework?*
- *How will data be pass to/from your software?*

Four Resilient Programming Models

- Relaxed Bulk Synchronous (rBSP)
- Skeptical Programming. (SP)
- Local-Failure, Local-Recovery (LFLR)
- Selective (Un)reliability (SU/R)

Toward Resilient Algorithms and Applications
Michael A. Heroux arXiv:1402.3809v2 [cs.MS]

Resilience & Task-centric/Dataflow

- Relaxed Bulk Synchronous (rBSP)
  - Async tasking: Addresses same issues.
  - “Porous barriers”:
    - Tasks contribute portion to global collective, move on.
    - Come back later to collect global result.
- Skeptical Programming. (SP)
  - Skepticism applied at task level.
  - Parent task can apply cheap validation test up child’s return.
- Local-Failure, Local-Recovery (LFLR)
  - Applied at task level.
  - SSD storage available for task-level persistent store.
- Selective (Un)reliability (SU/R)
  - Parent task (at some level in the task graph) executes reliably.
  - Children are fast, unreliable.
  - Parent corrects or regenerates child task if it times out or SDC detected.

Creating Incentives to Improve Productivity

Sandia National Laboratories

Reproducibility & Independent Verification Requirement

- In order to publish a paper: *Someone other than the authors must be able to reproduce the computational results.*
- Latitude in “reproduce”:
  - Exactly the same numerical results?
  - Exactly the same runtime?
  - Close, in the opinion of an expert reviewer?
- What about:
  - Access to the same computing environment?
  - High end systems?
- Lots of challenges.
- But just the *expectation [threat]* can drive efforts…

Fruits of the Threat

- **Source management tools:** In order to guarantee that results can be reproduced, the software must be preserved so that the exact version used to produce results is available at a later date.
- **Use of other standard tools and platforms:** In order to reduce the complexity of an environment, standard software libraries and computing environments will be helpful.
- **Documentation:** Independent verification requires that someone else understand how to use your software.
- **Source code standards:** Improves the ability of others to read your source code.
- **Testing:** Investment in greater testing makes sense because the software will be used by others.
- **High-quality software engineering environment:** If a research team is serious about producing high-quality, reproducible and verifiable results, it will want to invest in a high-quality SE environment to improve team efficiency.

Evidence: Cover letter excerpt from RCR candidate paper

Thank you for taking the time to consider our paper for your journal.

XXX has agreed to undergo the RCR process should the paper proceed far enough in the review process to qualify. ***To make this easier we have preserved the exact copy of the code used for the results (including additional code for generating detailed statistics that is not in the library version of the code).***

ACM Transactions on
Mathematical Software

ACM TOMS

- TOMS RCR Initiative: Referee Data.
- Why TOMS? Tradition of real software that others use.
- Two categories: Algorithms, Research.
- TOMS Algorithms Category:
  - Software Submitted with manuscript.
  - Both are thoroughly reviewed.
- TOMS Research Category:
  - Stronger: Previous implicit “real software” requirement is explicit.
  - New: Special designation for replicated results.

ACM TOMS Reproducible Computational Results (RCR) Process

- Submission: Optional (for now) RCR option.
- Standard reviewer assignment: Nothing changes.
- RCR reviewer assignment:
  - Concurrent with the first round of standard reviews
  - Known to and works with the authors during the RCR process.
- RCR process:
  - Multi-faceted approach.
- Publication:
  - Replicated Computational Results Designation.
  - The RCR referee acknowledged.
  - Review report appears with published manuscript.

RCR Process

- Independent replication:
  - Transfer of or pointer to software given to RCR reviewer.
  - Guest account, access to software on author’s system.
  - Detailed observation of the authors replicating the results.
- Review of computational results artifacts:
  - Results may be from a system that is no longer available.
  - Leadership class computing system.
  - In this situation:
    - Careful documentation of the process.
    - Software should have its own substantial verification process.

Status

- First RCR paper available:
  - Editorial introduction.
  - van Zee & van de Geijn, BLIS paper.
  - Referee report.
- 1 RCR paper per TOMS issue.
  - Hogg & Scott next.

Message to This Audience

Be prepared to have someone else replicate your results.

Summary

- Thread-scalable algorithms making steady progress: “easy”.
- Resilience strategies too, and reliability will persist until we are ready: “easy”.
- Big task: Transforming application base to new systems and beyond.
- SW engineering focus is important for HPC:
  - Pursuing efficiency negatively impacts many other quality metrics.
- Productive application designs will require disruptive changes:
  - Array and execution abstractions needed for portability.
  - Reuse via composition is attractive (think Android/iOS, Docker environments).
  - A Task-centric/dataflow app architecture is very attractive for performance portability.
- Journal, funding agency policies can provide productivity incentives:
  - Replicability expectations: Better SW practices are a natural reaction.
  - Funding Proposals:
    - We expect data management plans.
    - Can we start expecting a SW quality management plan?

Final Thought: Commitment to Quality

Canadian engineers' oath (taken from Rudyard Kipling):

My Time I will not refuse;
my Thought I will not grudge;
my Care I will not deny
toward the honour, use,
stability and perfection of
any works to which I may be
called to set my hand.

http://commons.bcit.ca/update/2010/11/bcit-engineering-graduates-earn-their-iron-rings