Next Generation Applications: Using a Productivity Focus

Michael A. Heroux

2015 OLCF Users Meeting

June 22, 2015

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

Sandia National Laboratories

Outline

- Background.
- SW Engineering and Productivity
- Application Design and Productivity
- Productivity Incentives.
- Modeling & Measuring Productivity.

Trilinos

trilinos.org

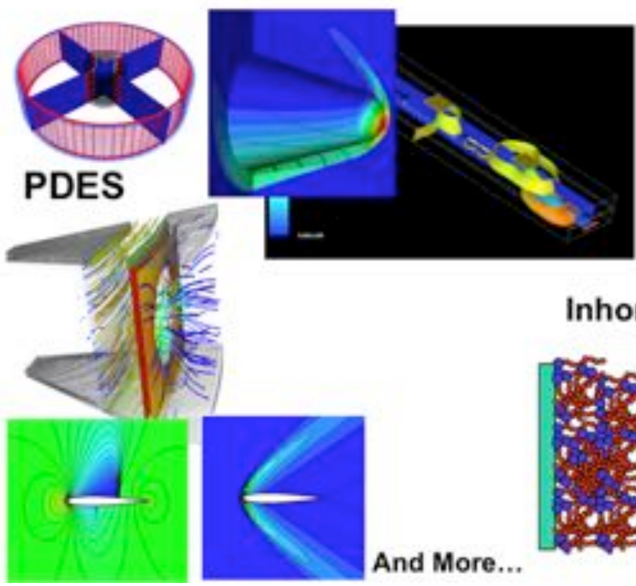
R&D 100

- R&D 100 Winner
- 11,851 Registered Users.
- 41,000 Downloads.
- Open Source.

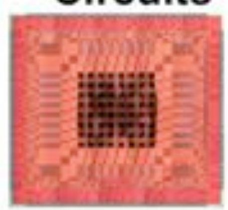
Laptops to Leadership systems

Vertical stacking and integration:
- Geometry, Meshing
- Discretizations, Load Balancing.
- Scalable Linear, Nonlinear, Eigen, Transient, Optimization, UQ solvers.
- Scalable I/O

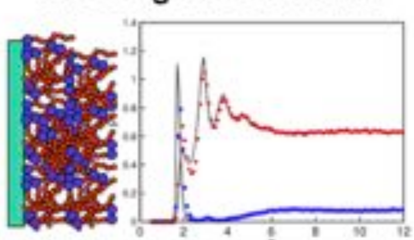
- 60 Packages.
- Binary distributions:
    - Cray LIBSCI
    - Debian, Ubuntu



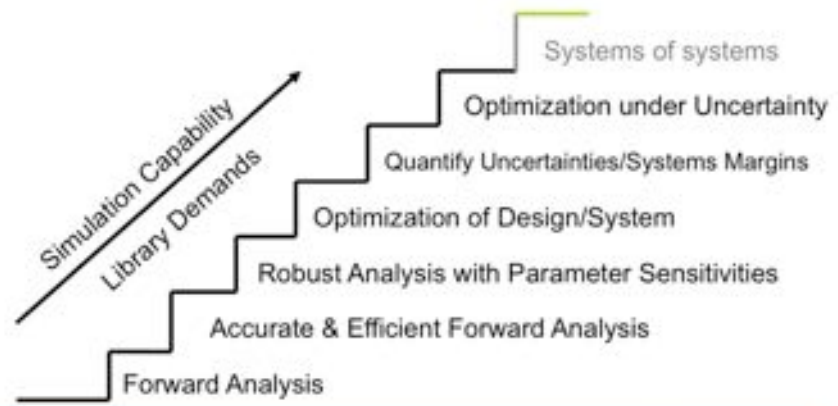
Circuits



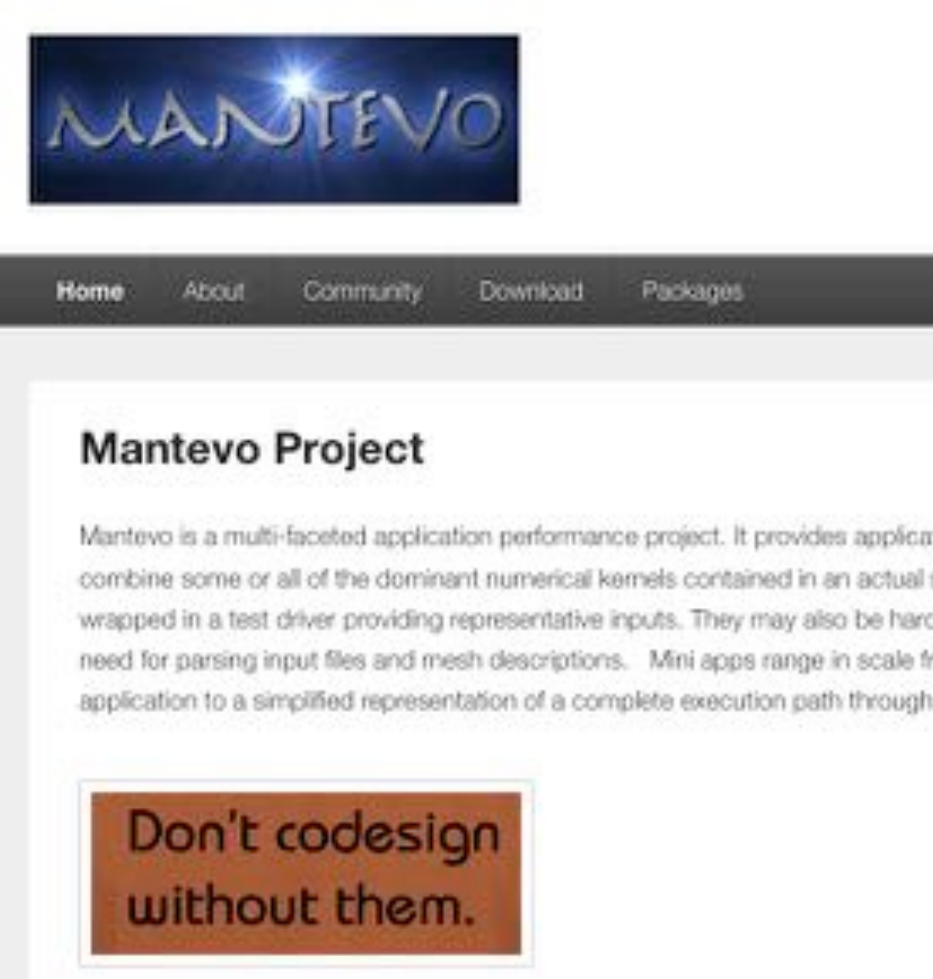
Inhomogeneous Fluids



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.



- mantevo.org
- Annual release prior to SC'XY.
- Open source.
- 200+ citations.
- 2013 R&D 100 winner.
- Collaboration: SNL, LLNL, LANL, AWE

Application Proxies for Co-Design

Release 3.0: At SC'14

Miniapps:

- ***CloverLeaf:** Version 1.1, Reference Version 1.1
- ****CloverLeaf3D**: Version 1.0, Reference Version 1.0
- **CoMD**: Reference Version 1.1
- **HPCCG**: Reference Version 1.0
- ****MiniAero**: Version 1.0
- ****MiniAMR**: Version 1.0, Reference Version 1.0
- ***MiniFE: Version 2.0.1, Reference Version 2.0**
- **MiniGhost**: Version 1.0.1, Reference Version 1.0.1
- ***MiniMD: Version 1.2, Reference Version 2.0**
- ***MiniSMAC2D**: Reference Version 2.0
- **MiniXyce**: Reference Version 1.0
- ****Pathfinder**: Version 1.0.0
- ****TeaLeaf**: Version 1.0, Reference Version 1.0

Minidrivers:

- ***CleverLeaf**: Version 2.0, Reference Version 2.0
- **EpetraBenchmarkTest**: Version 1.0

** New miniapp for Suite Release 3.0.

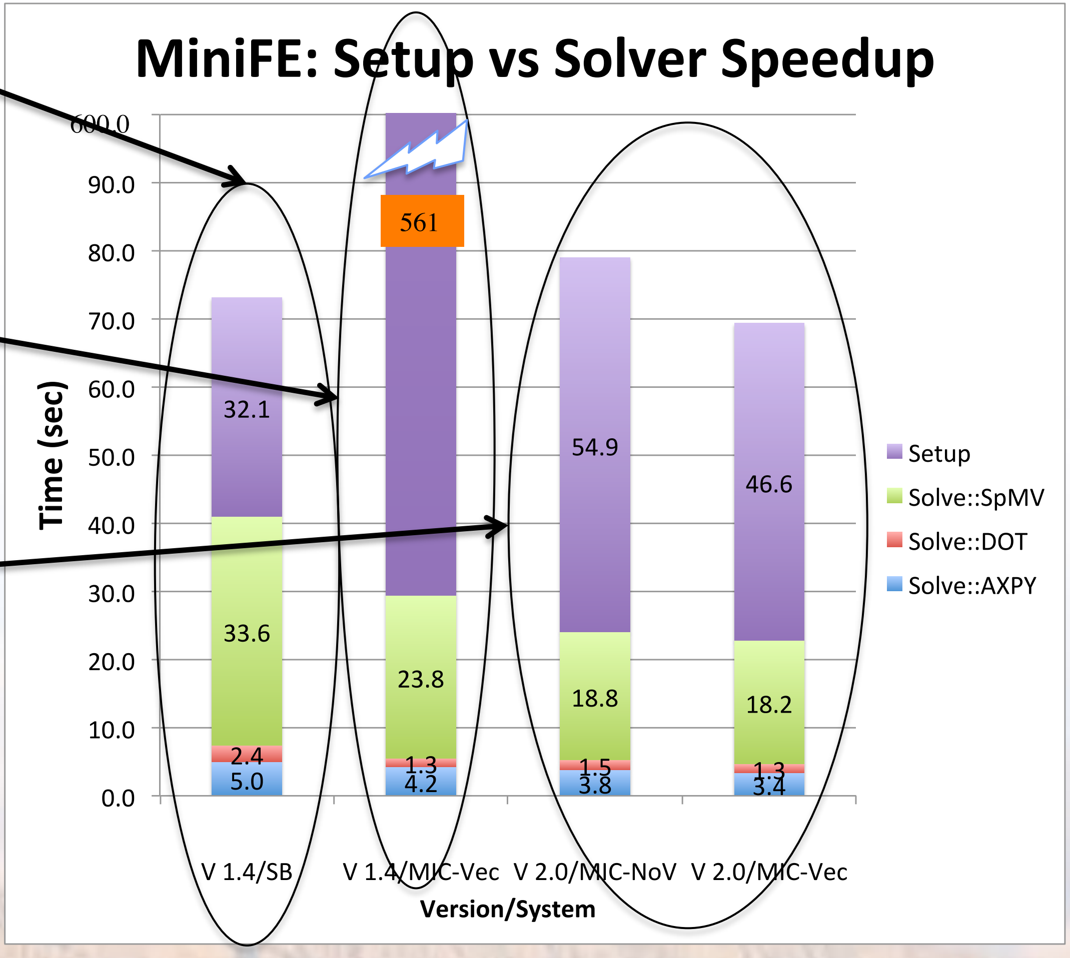
*New version for Suite Release 3.0.

Sandia National Laboratories

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

5

- 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



A Confluence of Trends

- Fundamental trends:
  - Disruptive HW changes: Requires thorough alg/code refactoring.
  - Demands for coupling: Multiphysics, multiscale.
- Challenges:
  - Need *k* refactorings: 1+ε*k*, not *k*-ε. 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.
  - Better OS/Runtime/HW layers to assist apps.
- Basic strategy: Focus on productivity.

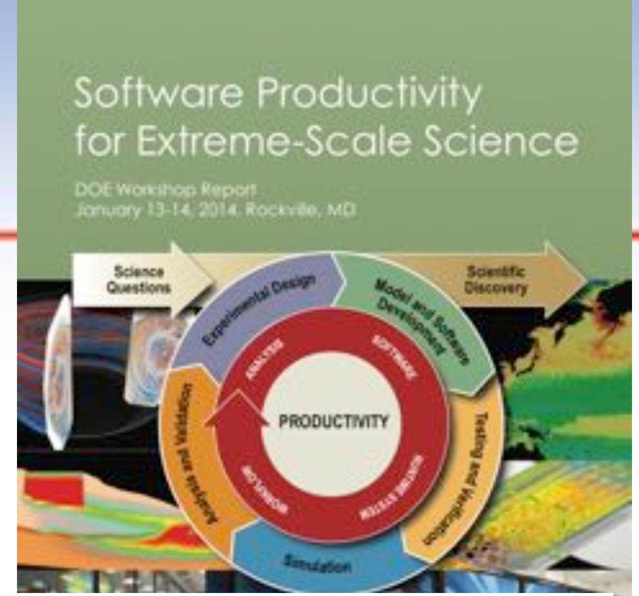
Productivity

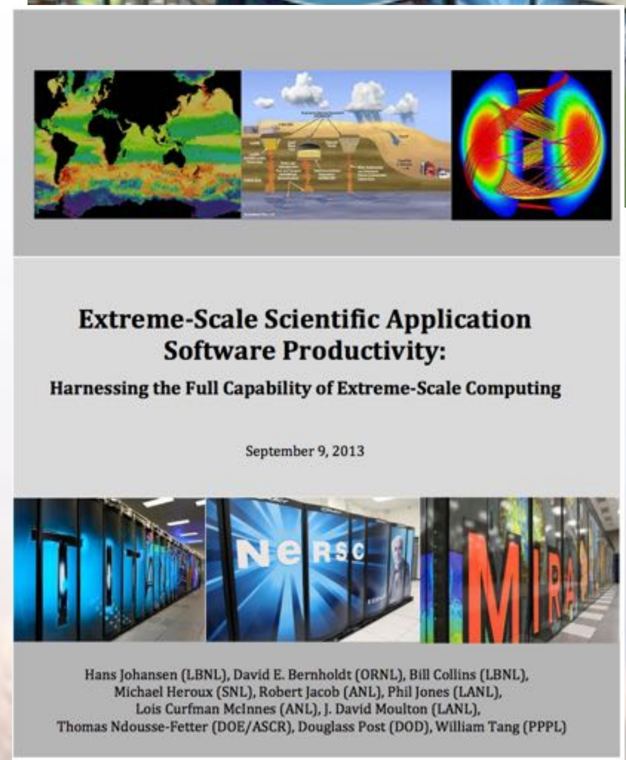
Better, Faster, Cheaper: Pick all three

Sandia National Laboratories

Productivity Emphasis

- *Scientific* Productivity.
- Many design choices ahead.
- Productivity emphasis:
  - ***Simple*** Metrics. Want a ***process*** to define.
  - Design choice ***processes*** (How to).
- Focus on actionable productivity metrics.
- **2 Productivity improvement strategies:**
  - **Local (Optometrist):**
    - Which is better, this or this?
  - **Global (Time bi-section):**
    - Use proxies for “paradigm shifts”.
    - Rapid design space exploration.
    - Co-design, miniapps, etc.





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*.

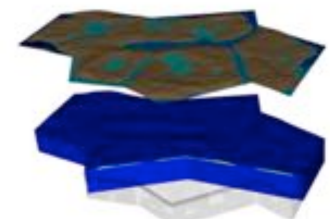
Objectives

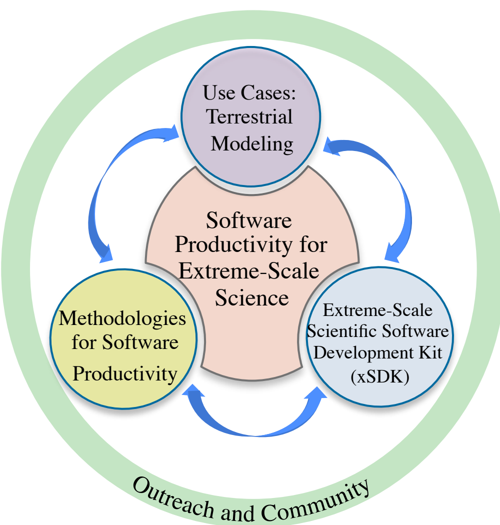
Address confluence of trends in hardware and increasing demands for predictive multiscale, multiphysics simulations.

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).





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

Institutional Leads (Pictured) Full Team List

Science Use Cases

J. David Moulton
Tim Scheibe
Carl Steefel
Glenn Hammond
Reed Maxwell
Scott Painter
Ethan Coon
Xiaofan Yang

Project Leads

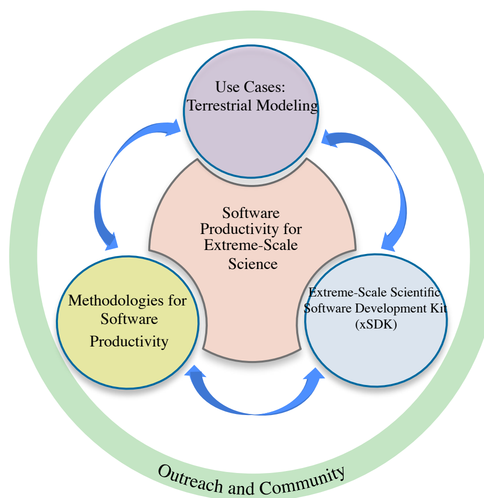
ASCR: M. Heroux and L.C. McInnes
BER: J. D. Moulton

Extreme-Scale Scientific Software Development Kit (xSDK)

Mike Heroux
Ulrike Meier Yang
Jed Brown
Irina Demeshko
Kirsten Kleese van Dam
Sherry Li
Daniel Osei-Kuffuor
Vijay Mahadevan
Barry Smith

Methodologies for Software Productivity

Hans Johansen
Lois Curfman McInnes
Ross Bartlett
Todd Gamblin*
Andy Salinger*
Jason Sarich
Jim Willenbring
Pat McCormick



Outreach

David Bernholdt
Katie Antypas*
Lisa Childers*
Judith Hill*

*Liaison

SW Engineering & Productivity

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	↓	↑	↓	↓	↓	↑	↓	↑

Helps it ↑

Hurts it ↓

Source:
Code Complete
Steve McConnell

TriBITS: One Deliberate Approach to SE4CSE

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

<u>TriBITS Lifecycle Maturity Levels</u>

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.

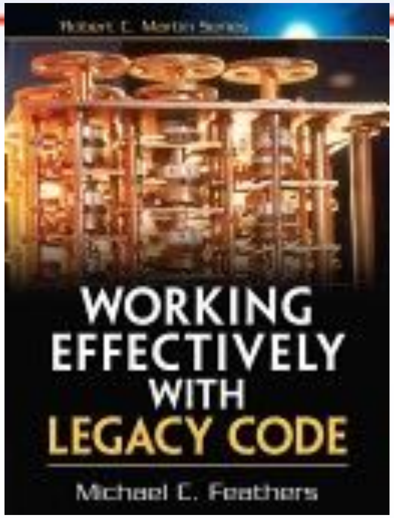
Addressing existing Legacy Software

- 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

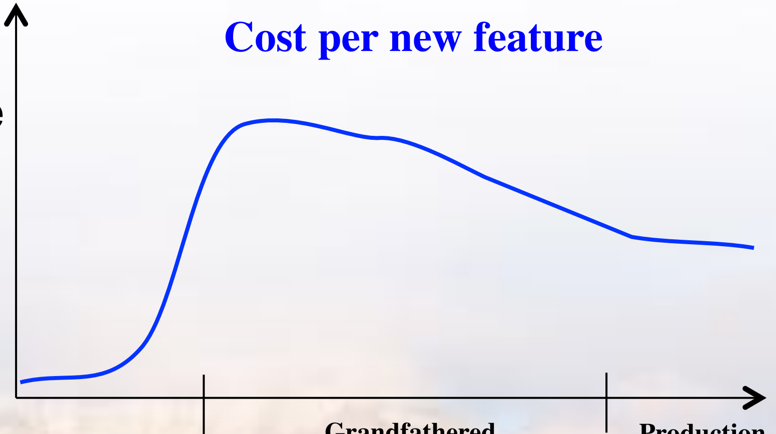


Agile Legacy Software Change Algorithm:

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.

Sandia National Laboratories

Howto and Whatis Documents

What Are Software Testing Practices?

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

IDEAS productivity What is

Motivation: Software requires regular extensi

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

In this document, we introduce some terminolo and general approaches to testing.

Types and granularities of testing: Softwar testing (see Definition and Categorization of T

- **Verification testing**: Tests that verify t
- **No-change (often, perhaps mistaken** code produces the same results (to an Having comprehensive no-change unit code (refactoring) but quickly verify tha

In addition, three granularities of testing are re

- **Unit tests:** Focus on testing individual individual classes.
- **Integration tests:** Focus on testing the the full system level.
- **System-level tests:** Focus on testing t level. For example, a system-level test input files, running the full simulation co solutions.

Managing and reporting on testing: The sin runs one or more executables, saving the outp examine. Once a package becomes too compl satisfactory and requires various enhancemen called **test harnesses**) are introduced to lowe adding new tests. For example, filters can be indicates problems (e.g., here and here), and t color) which build instantiations generated erro requires developers to check a website on a re

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

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

How to Add and Improve Testing in Your CSE Software Project

IDEAS productivity How to

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

Sandia National Laboratories

Three Application Design Strategies for Productivity & Sustainability

Strategy 1: Array and Execution Abstraction

Multi-dimensional Dense Arrays

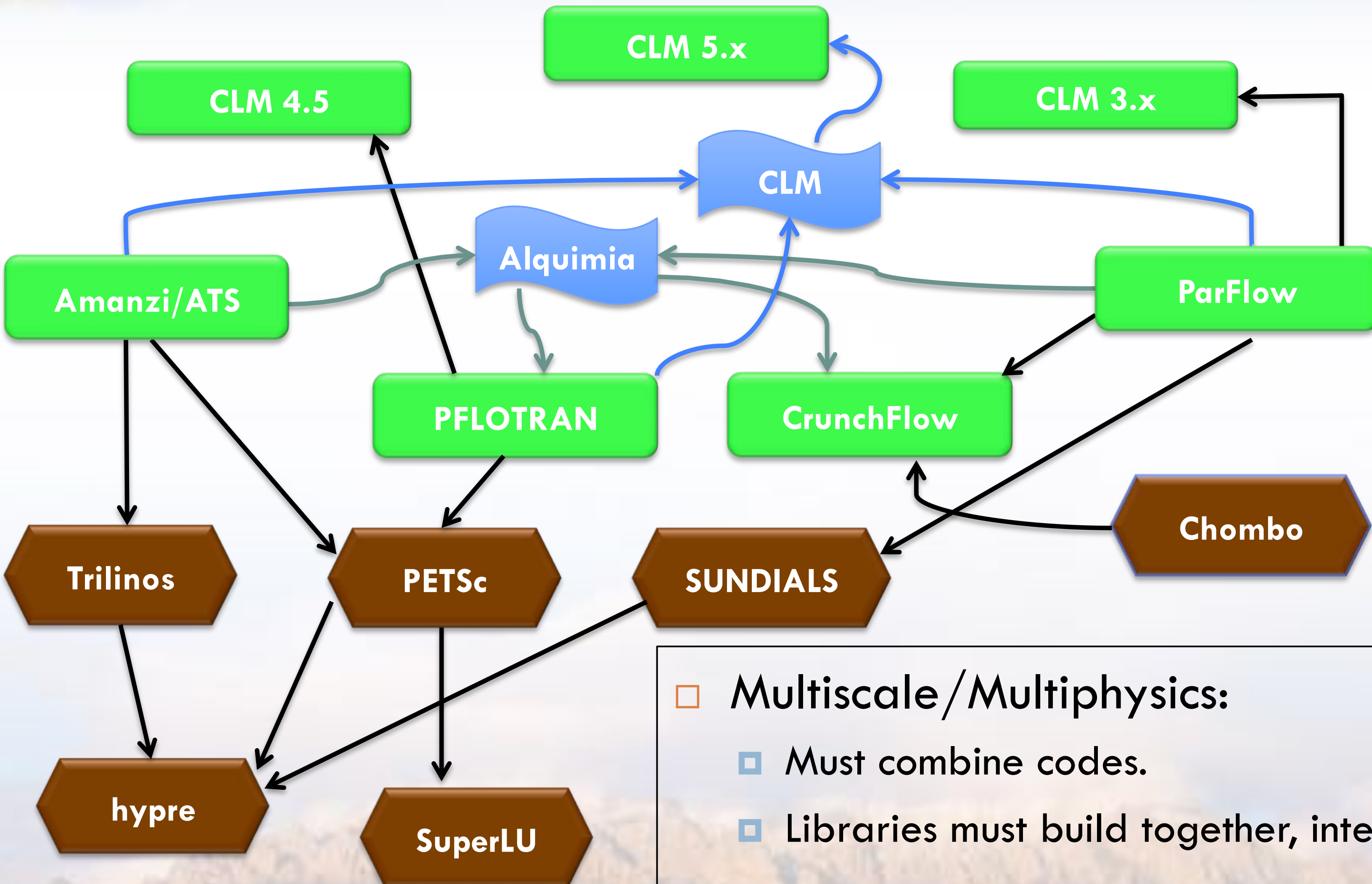
- Many computations work on data stored in multi-dimensional arrays:
  - Finite differences, volumes, elements.
  - Sparse iterative solvers.
- Dimension are (k,l,m,…) where one dimension is long:
  - A(3,1000000)
  - 3 degrees of freedom (DOFs) on 1 million mesh nodes.
- A classic data structure issue 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.**

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.

Strategy 2: Application Composition

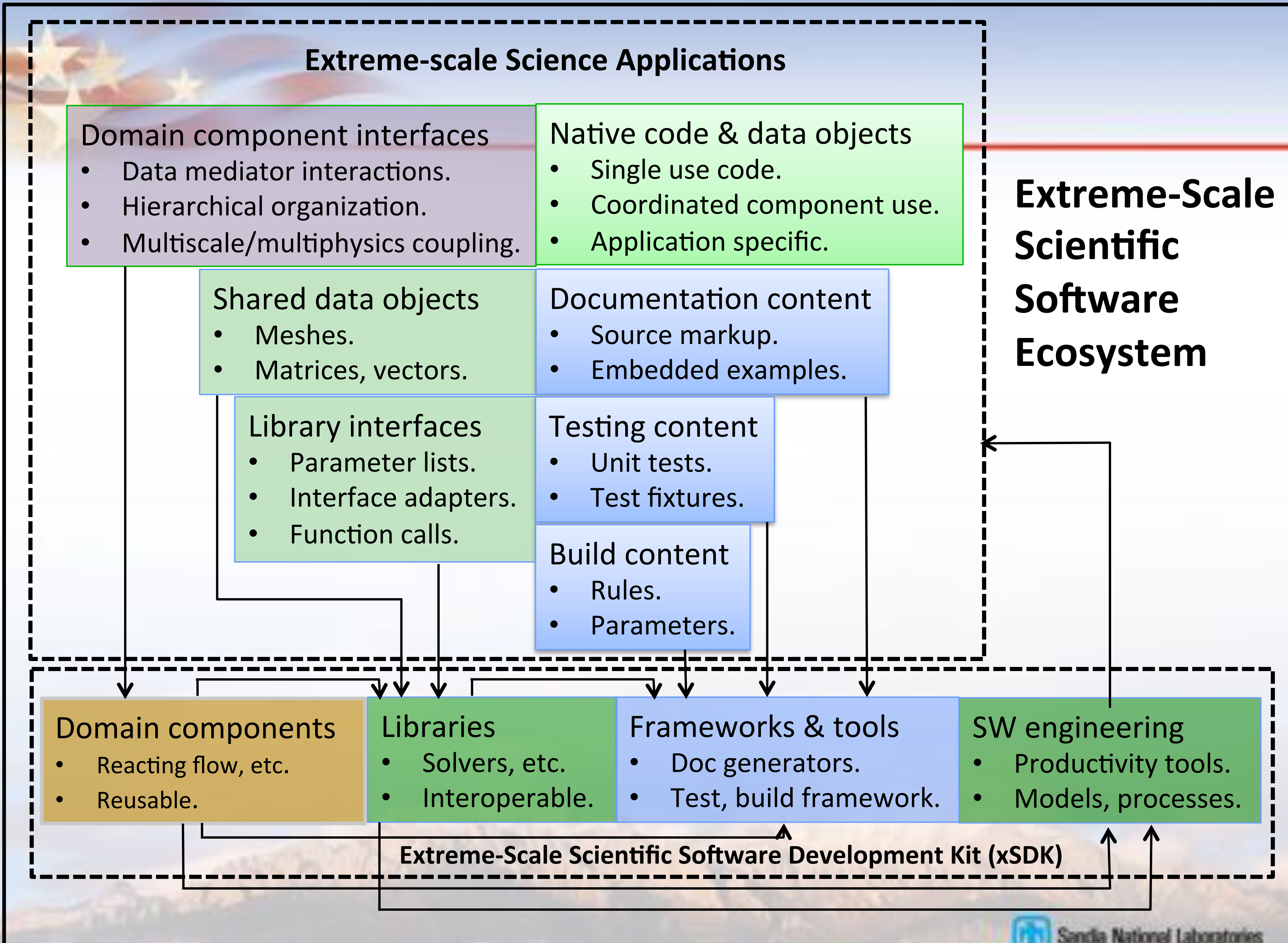
IDEAS Codes and Libraries



- Multiscale/Multiphysics:
  - Must combine codes.
  - Libraries must build together, interact.

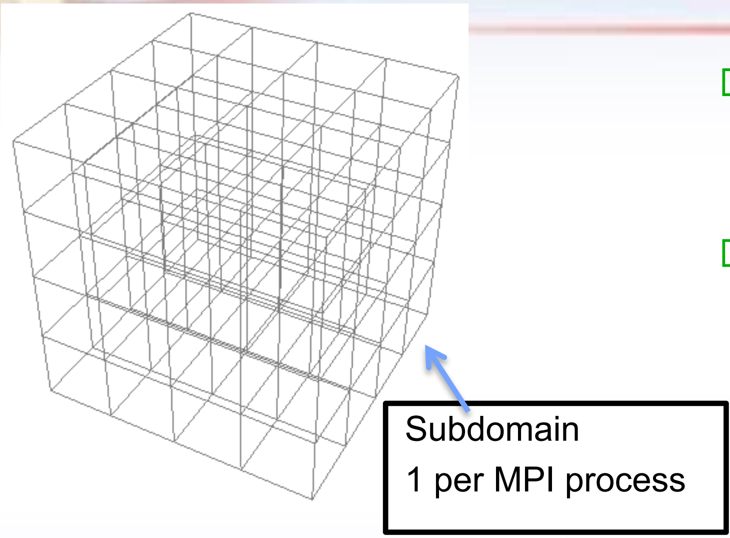
xSDK focus

- Common configure and link capabilities
  - Initial emphasis: Chombo, hypre, PETSc, SuperLU, Trilinos
  - Approach:
    - Determine common definition of configure arguments, eliminate namespace collisions
    - Develop approach that can be adapted by any library development team for standardized configure/link process
    - Develop testing capabilities to assure configure/link processes continue to work indefinitely
- Library interoperability
- Designing for performance portability
- Compositional approach to application design:
  - Build app from components.
  - Tuned algorithms.
  - Performance portability: now and in the future.


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)
Sandia National Laboratories

Strategy 3: Toward a New Application Architecture

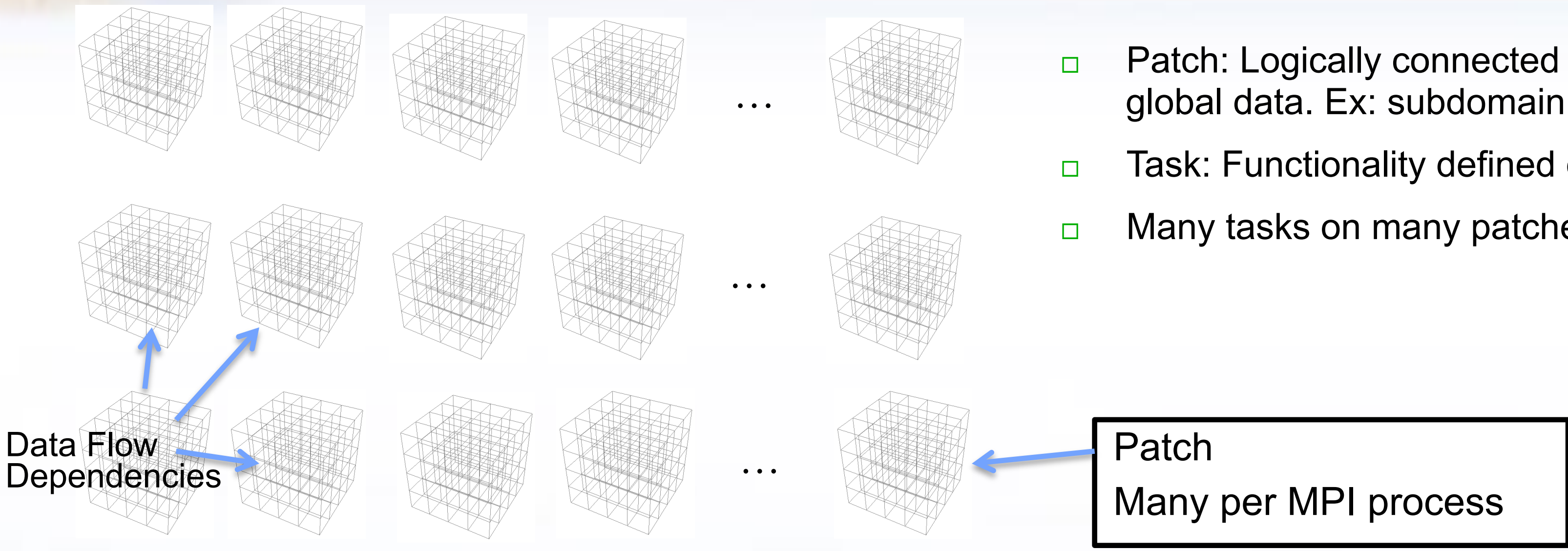
Classic HPC Application Architecture



- 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.

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.

- 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.

Sandia National Laboratories

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?

Sandia National Laboratories

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.

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

Kokkos/Qthread LDRD

- 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

33

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).
  - 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).

Creating Incentives to Improve Productivity

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

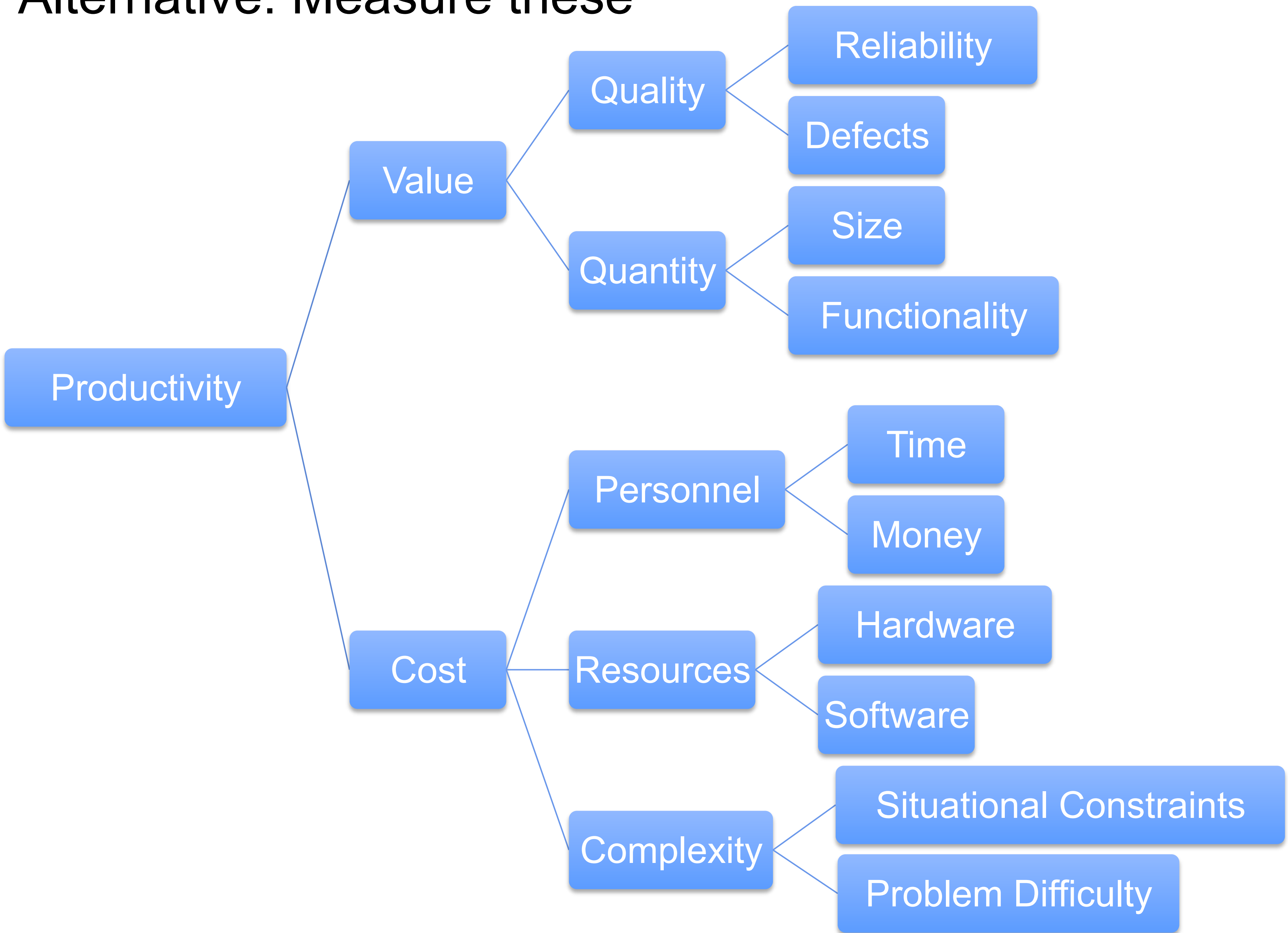
- First RCR paper coming in next TOMS issue
  - Editorial introduction.
  - van Zee & van de Geijn, BLIS paper.
  - Referee report.
- 1 RCR paper per TOMS issue.
  - Hogg & Scott next.

Measuring and Modeling Productivity

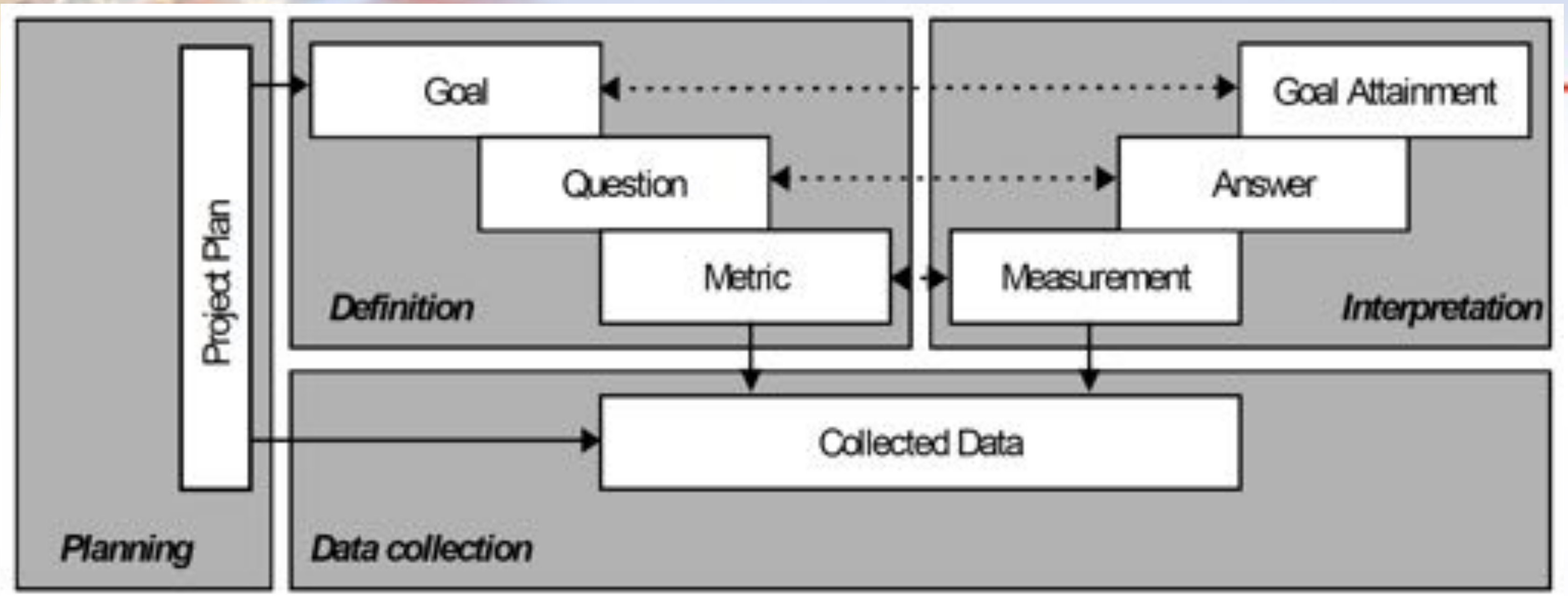
Task: Measure Productivity

Productivity

Alternative: Measure these



Measuring Productivity: A Practical Approach



- Define *processes* to define metrics.
  - Starting point: Goals, questions, metrics (GQM).
    - Define goals, ID questions to answer, define progress metrics.
- GQM Example:
  - Goal: xSDK Interoperability.
  - Question: Can IDEAS xSDK components & libs link?
  - Metric: Number of namespace collisions.

Source: *The GQM Method: A Practical Guide for Quality Improvement of SW Development*, Solingen and Berghout.

Sandia National Laboratories

Toward Effective Models

- Hobby: Audible *Great Courses* on Economics, Human Development.
  - Models developed and used extensively.
  - Never right, but useful.
  - Catalysts for innovation and insight.
- Idea: Models for HPC lifecycles, productivity.
  - Focus: Requirement that you *have* one.
  - But: No specifications.
- Path: Use data management plan approach.
  - Required element for NSF, DOE proposals
  - Very non-specific: Forces innovation, creativity.

Productivity-related Meetings

- 3rd Workshop on Sustainable Software for Science Practice and Experiences (WSSSPE3)
  - September 28-29, 2015, Boulder, CO
  - http://wssspe.researchcomputing.org.uk/wssspe3
  - (Co-located 10th Gateway Community Environments (GCE15) Wkshp)
- CSE Software Sustainability and Productivity (CSESSP) Challenges Workshop
  - October 15 – 16, 2015, Washington, DC
  - Multi-agency (DOE, DOD, NSF, NIST, NIH) event.
- SEHPCCSE'15: The Third International Workshop on Software Engineering for High Performance Computing in Computational Science and Engineering
  - Friday, Nov. 20 - in Conjunction with SC15 – Austin, TX
  - http://SE4Science.org/workshops/se4hpccse15

Summary

- **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 and iOS 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.
  - Proposals:
    - We expect data management plans.
    - Can we start expecting a SW lifecycle, productivity model?
- **Productivity models:**
  - P = V/C, but start at the leaves, more intuitive.
- **Productivity metrics:**
  - Need data.
  - Consider GQM.

Sandia National Laboratories

Summary

- **An explicit focus on productivity is compelling.**
- **Simple productivity definitions often sufficient:**
  - Majority of productivity initiatives use “eye doctor” approach.
  - But this approach is not enough: Global changes are needed.
- **Need models: Lifecycle, Productivity**
  - SC papers: Require explicit performance model, not unreasonable.
  - Programmatically: Establish requirements, not specifications.
  - Need instrumentation: Inputs: What? How much? Outputs: ditto.
- **Effective models enable “bold” behavior:**
  - Choose approaches with better overall productivity.
  - Defend these choices.

Sandia National Laboratories