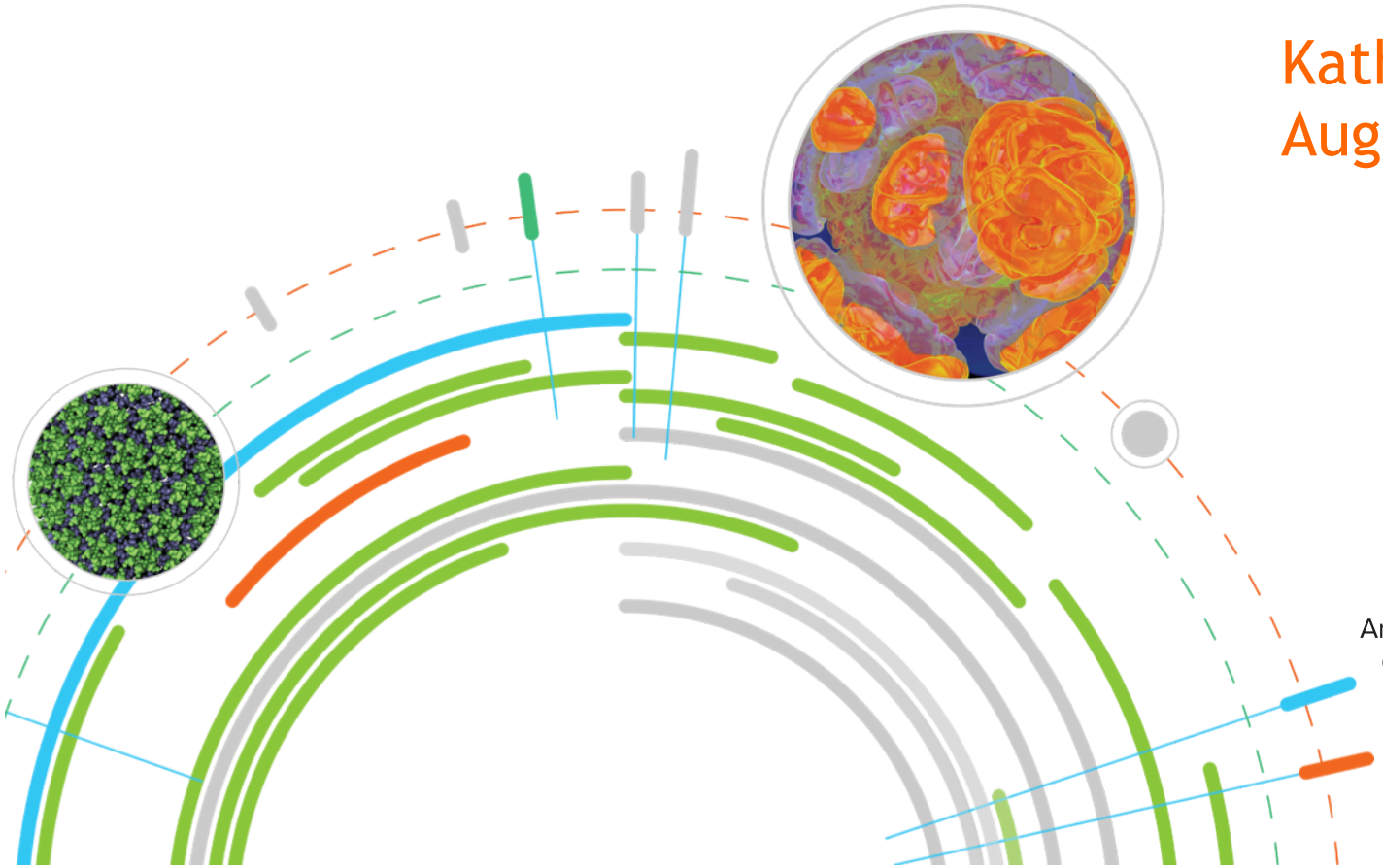
Community Codes and Good Software Techniques

Scientific codes are complex
Introduction to the next two days

Katherine Riley
August 10, 2015



Argonne Leadership Computing Facility

Argonne NATIONAL LABORATORY

Over the next two days

Today - Applications	
Architecting Community Codes	Anshu Dubey
The Impact of Community Codes on Astrophysics	Sean Couch
Designing Scalable Scientific Software	Bill Tang
Modern Features of Production Scientific Code	Martin Berzins
HEP – Complex Workflows	Tom LeCompte
HACC – Application Performance Across Diverse Architectures	Salman Habib
Tomorrow – Process (mostly	
Software Engineering Practices	Aron Ahmadia
NAMD	Jim Phillips
Types of Workflows	Tom Uram
Swift as a Workflow Solution	Mike Wilde
Data Provenance	David Koop

Real Application Experience

- Experience is almost everything
- Speakers are PIs with a lot of experience in their domain science and computational science
- Look for lessons that might be relevant – even from far outside your domain

Workflows & Provenance

- You all have a computational workflow
- Capturing that is important
  - Document it!

- Provenance – where does your data come from?
  - Can you track the exact run and the conditions of that run?
  - Critical part of software engineering and scientific process

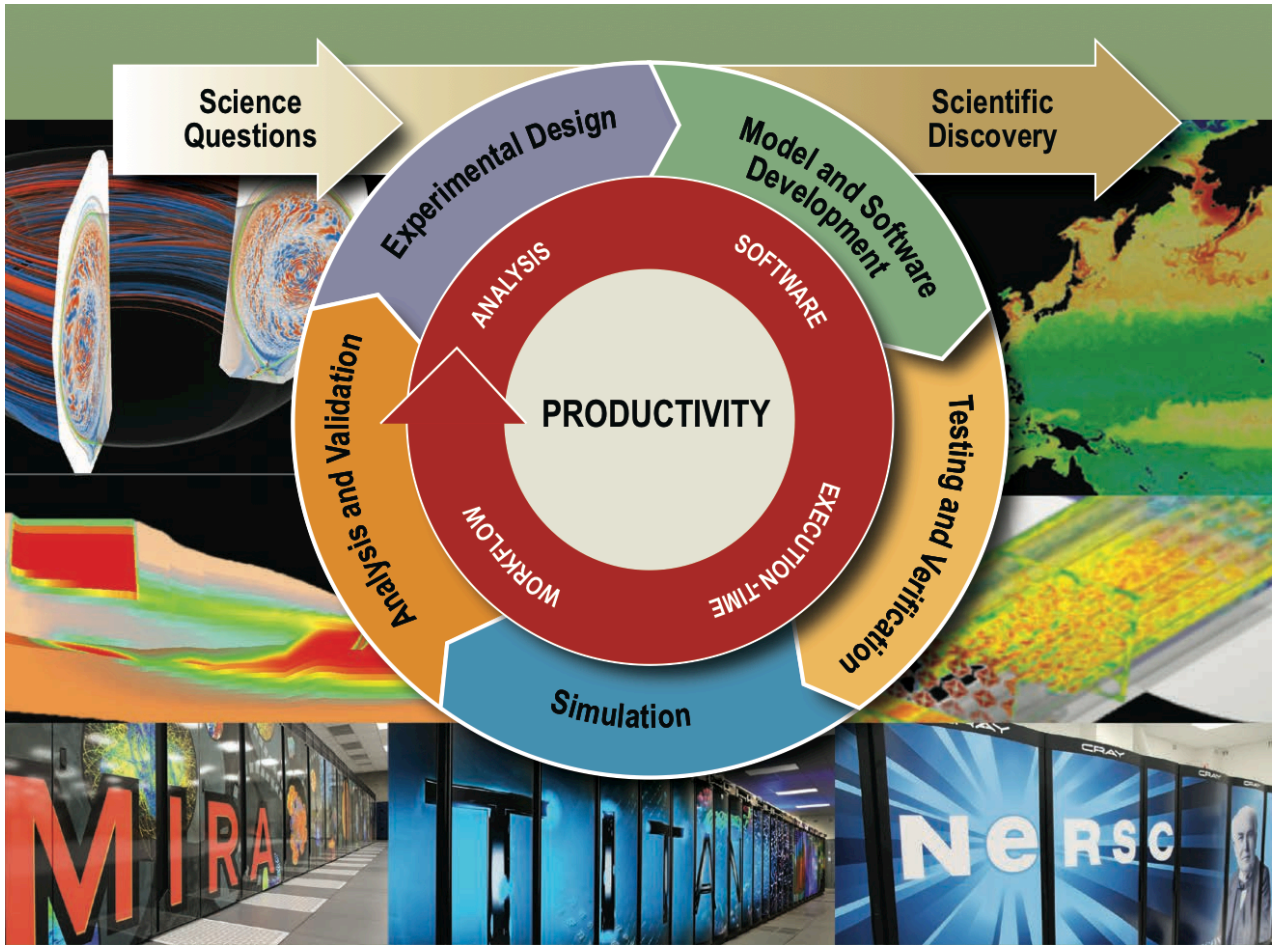
Goals

- Expose some of the processes required for developing scientific codes
- Show you the approach and effectiveness of code cooperation in a variety of domains
- Illustrate some of the challenges of those approaches
  - Sociological & Technical
- Ensure you know the importance and specifics of the software & scientific process

- We are not trying to teach you the applications
- We are passing on experience
- A lot of people have spent a lot of time thinking about maintain codes that use the largest systems in the world

Scientific applications are complex

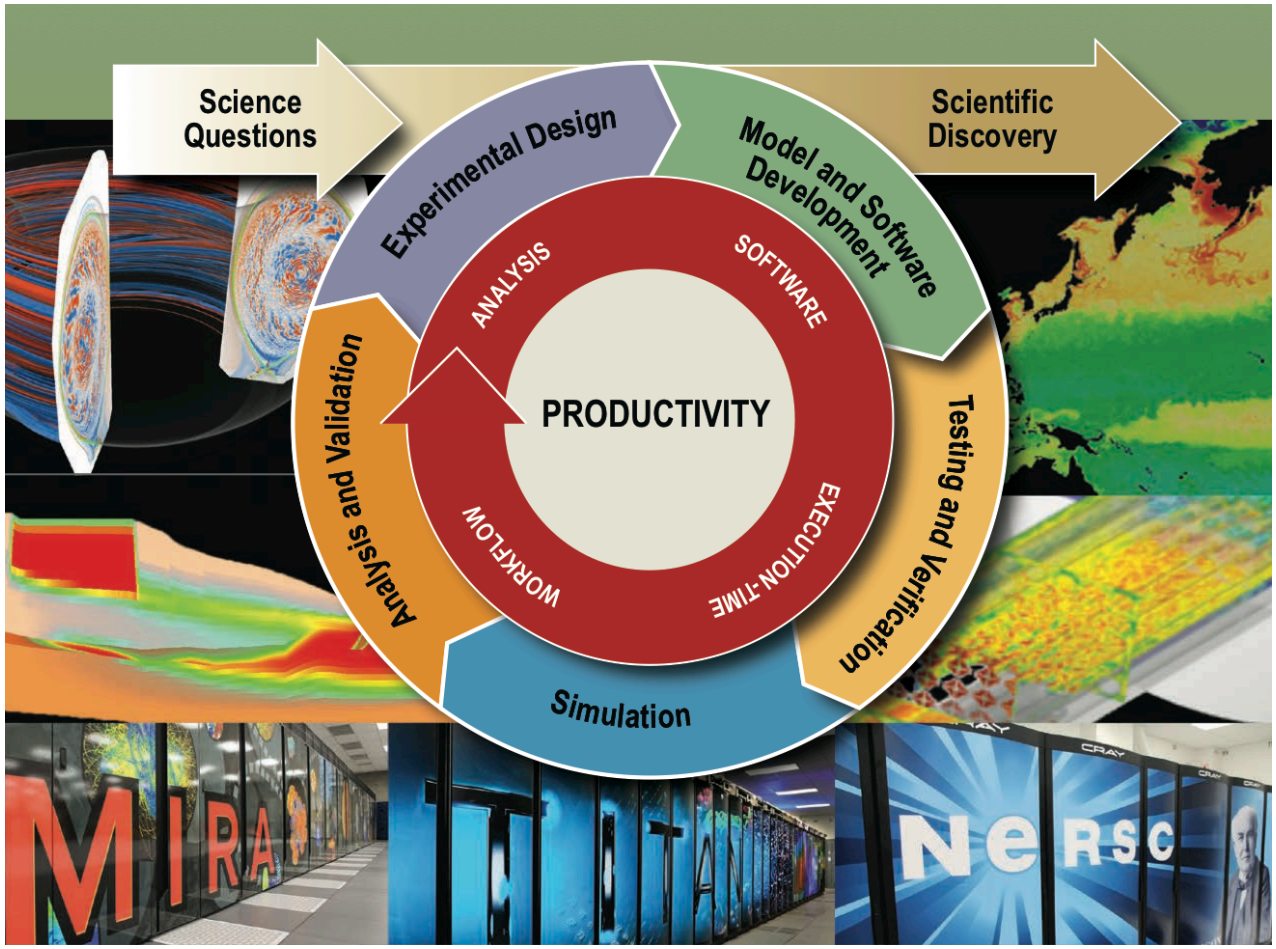
- Physics/Domain Problem
- Applied Mathematics
- Computer Science
- **I/O**
- Verification
- Validation
- Software Architecture & Engineering



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

Scientific applications are complex

- Physics/Domain Problem
- Applied Mathematics
- Computer Science
- I/O
- Verification
- Validation
- Software Architecture & Engineering



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

ATPESC Material Covered so far

ATPESC Topics

- Hardware Architecture
- Programming Models
  - Low Level - MPI, OpenMP, Acceleratos/OpenACC
  - High Level - Chapel, Charm++, UPC, ADLB, etc
- Numerical Algorithms
  - Libraries, toolkits, etc
- Tools & Performance
- Visualizing & Analyzing Data (coming)
- I/O & Data

You should not just bang these together.

Next Two Days

- How do you integrate all the concepts?
- Real examples of applications
- Good process
- Software practices & engineering
- Data provenance and workflows

- There are no perfect answers.
- Ask of everyone

Doing It Right Can Be Hard

Software Practice

Scientific Process

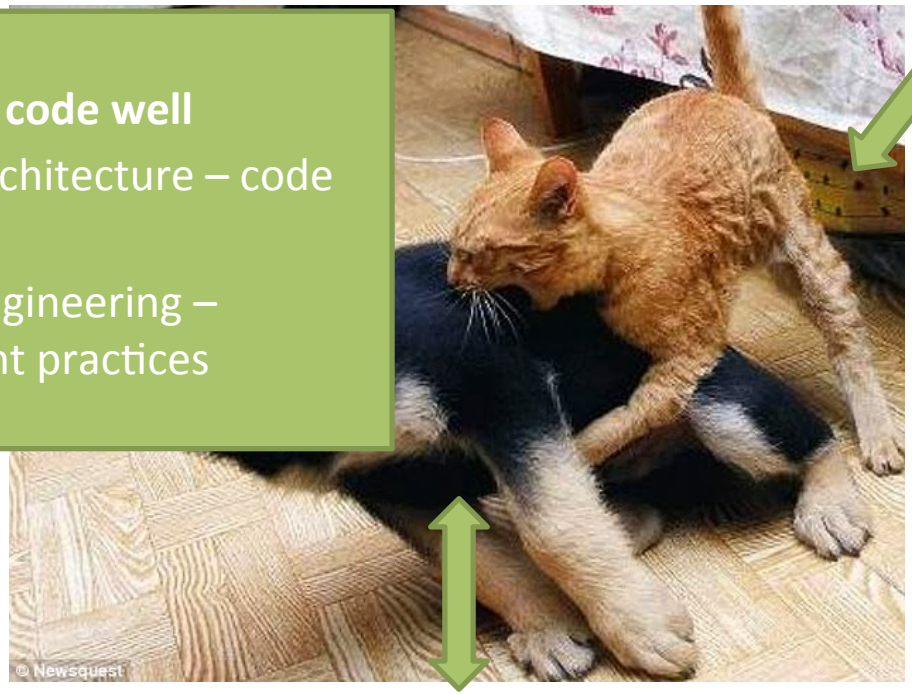
Producing Domain Science

Doing It Right Can Be Hard

Software Practice

Scientific Process

- **Writing the code well**
- Software Architecture – code design
- Software Engineering – development practices



Producing Domain Science

Putting all this to use

Software Practice

Scientific Process

- **Doing Your Science Well**
- Software engineering
- Trade-offs

Producing Domain Science

Argonne Leadership Computing Facility

Putting all this to use

Software Practice

Scientific Process

Why?

- Time
- People
- Recognition (papers)
- Conflicting Priorities

Producing Domain Science

Doing It Right Can Be Hard

Software Practice

Scientific Process

It is getting friendlier
It Has To

Producing Domain Science

Doing It Right Can Be Hard

Performance

Portability

Readability

Implementations vs. Process

Implementation

- Architecture
- Programming Models
  - MPI, OpenMP, Acceleratos/ OpenACC
  - Chapel, Charm++, UPC, ADLB, etc
- Numerical Algorithms
  - Libraries, toolkits, etc
- Tools & Performance
- Visualizing & Analyzing Data
- I/O

Process - The Life

- Software Practices
- Scientific Process
- Portability
- Extensibility
- Performance
- Provenance
- Resilience
- Reproducibility
- Verification and Validation
- And more…

Your code will live longer than you think it will.

Software Engineering and HPC
Efficiency vs. Other Quality Metrics

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

Source:
Code Complete
Steve McConnell

Helps it ↑
Hurts it ↓

Sandia National Laboratories

Key Practices for Scientific Process

Practice	Description	Actions
Validation	• Compare to experiments • Reproducibility	• Prove it represents the real world • Very science driven
Verification	• All parts of the code keep giving what you expect • Reproducibility	• Unit testing • Regular testing – on scale of development speed
Error	• Numerical Sensitivity • Machine rounding • Reproducibility	• Numerical & Sensitivity analysis of methods chosen and implementation of them
Repro-ducibility	• Exact code used • Documented • Method transparency • Data availability • Coding Standards	• Version tag code used for simulations • Clear documentation on code – even publish it • Data provenance • Data archiving • Understand & document workflow • Agree & Document coding standards

Scientist's Nightmare: Software Problem Leads to Five Retractions
Greg Miller
Science 22 December 2006: **314** (5807), 1856-1857. [DOI:10.1126/science.314.5807.1856]

Collaboration is Hard Without Process

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

Building a Scientific Code

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

Documentation
- Source markup
- Embedded examples

Library Interfaces
- Data transformation
- Parameter config

Testing Content
- Unit Tests
- Glue Testing

Build Content
- Rules
- Parameters

Adapted a slide from Mike Heroux, SNL

Programming Model & Languages

Libraries
- Solvers

Frameworks & tools

SW Engineering
- Productivity tools
- Models, processes

How to start the Software Process

- Science + Architecture + Future
- Decide on crucial data structures
  - Data movement
  - Data flow through functionality
- Architecture of code
  - Functional abstractions
  - Parallel abstractions
  - Data ownership clear
  - Interplay between architecture and performance
  - Coding Standards
- Understand workflow

Software Process Components

For All Codes

- Code Repository
- Build Process
- Code Architecture
- Coding Standards
- Verification Process
- Maintenance (Support) Practices

Publicly Distributed

- Distribution Policies
- Contribution Policies
- Attribution Policies

Obstacles for Reusing Code

- Using externally developed software seen as risk
  - Can be hard to learn
  - May not not be what you need
  - May not be what you *think* you need
  - Upgrades of external software can be risky
    - Backward compatible?
    - Regression in capability?
  - Support model may not be sufficient
  - Long term commitment may be missing
- What can reduce the risk of depending on external software?
  - Use strong software engineering processes and practices
    - high quality, low defects, frequent releases, regulated backward compatibility, …
    - 10-30 year commitment
    - Develop self-sustaining software

Many Choices for Codes and Trade-Offs

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

Assuming best case scenarios

Many Choices for Codes and Trade-Offs

	Speed to science	Speed of Change	Features	Control Methods & Accuracy	Complexity of use	Validation	Verification
Blackbox user	Fast	Slow	??	None	Depend		gh
Alter existing code base	Med	Fast					d/
Use of libraries	De						
Use of framework	Me					Shared	Shared
Development of new code	Slow			High	Depend s	All you	All you

- This is all an issues of control and effort
  - People
  - Expertise
  - Sustainability
  - Requirements

Assuming best case scenarios

Considerations

Some of the technical considerations

- Choosing your tools, codes, etc
  - Libraries
  - Frameworks
  - Open source code
  - Community code
  - Closed or commercial code
- Writing the code
  - Data structures
  - Data structures
  - Data structures from storage to memory to cache and back to storage (locality)
  - Parallelization of work and data
  - Languages

Everything else

- Development
  - Availability where and when you need them
  - Sustained support
  - Feature support
- The future of the code
  - HPC is the land of low level languages
  - HPC is the land of some bleeding
- Flexibility to replace libraries
- Flexibility to adapt to architectures
- ..

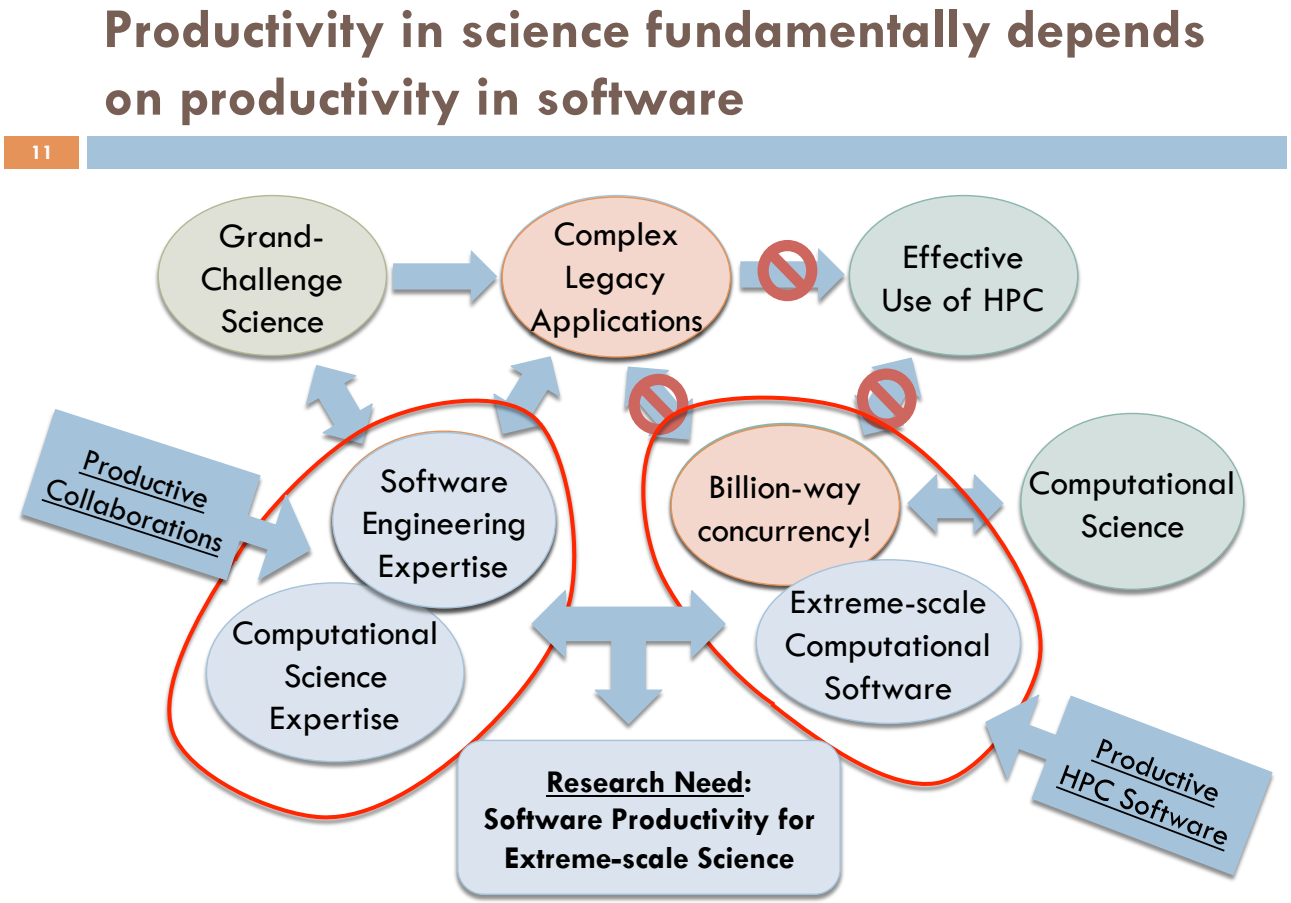
Models for Developing Scientific Codes

- Open source community developed codes
  - Always available, any contribution open source code
  - Central controls of code development
  - Closed non-commercial codes
  - Commercial code
- Speed of change
- Key design ideas
  - Scientific mission – scientists involved
  - Always capable of science
  - Portability – range of platform scale very beneficial
  - Documented
  - Clear design
  - Prove the code
- Solo development
- Small team

Self Sustaining Software

- ***Open-source***: The software has a sufficiently loose open-source license allowing the source code to be arbitrarily modified and used and reused in a variety of contexts (including unrestricted usage in commercial codes).
- ***Core domain distillation document***: The software is accompanied with a short focused high-level document describing the purpose of the software and its core domain model.
- ***Exceptionally well testing***: The current functionality of the software and its behavior is rigorously defined and protected with strong automated unit and verification tests.
- ***Clean structure and code***: The internal code structure and interfaces are clean and consistent.
- ***Minimal controlled internal and external dependencies***: The software has well structured internal dependencies and minimal external upstream software dependencies and those dependencies are carefully managed.
- ***Properties apply recursively to upstream software***: All of the dependent external upstream software are also themselves self-sustaining software.
- ***All properties are preserved under maintenance***: All maintenance of the software preserves all of these properties of self-sustaining software (by applying Agile/Emergent Design and Continuous Refactoring and other good Lean/Agile software development practices).

Side note on legacy codes



- People disagree
- Scientific applications are huge investment
- Applications last for decades

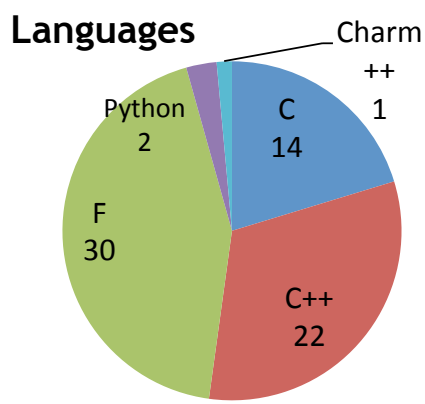
From a set of DOE workshops on HPC productivity

Consider the HPC ecosystem

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

- In HPC, the trade-off with design and performance is omnipresent
- Have reached a complexity point that code reuse & design is very important
- All your lessons from software engineering do ***not*** apply

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



Code Availability	
Open	52%
Closed	26%
Fuzzy	22%

Thoughts Going In

- Use your science goals to guide software process choices
- Explore how to incorporate these practices

- Process is important
- Every application is unique
- We cannot give you the perfect answer

- Architectures evolving; think forward

Over the next two days

Architecting Community Codes	Anshu Dubey
The Impact of Community Codes on Astrophysics	Sean Couch
Desinging Scalable Scientific Software	Bill Tang
Modern Features of Production Scientific Code	Martin Berzins
HEP – Complex Workflows	Tom LeCompte
HACC – Application Performance Across Diverse Architectures	Salman Habib
Software Engineering Practices	Aron Ahmadia
Types of Workflows	Tom Uram
Swift as a Workflow Solution	Mike Wilde
Data Provenance	David Koop

Goals

- Expose some of the processes around developing large, production scientific codes
- Show you the approach and effectiveness of code cooperation in a variety of domains
- Illustrate some of the challenges of those approaches
  - Sociological & Technical
- Ensure you know the importance and specifics of the scientific process

- We are not trying to teach you these codes
- We are passing on experience
- A lot of people have spent a lot of time thinking about maintain codes that use the largest systems in the world

Questions