LSSw Meeting 7

April 21, 2022

LSSw Meeting 8: May 19, 2022

Topic: Reducing the gap: Making open-source leadership software more accessible for industry

Description: Open-source scientific software is successfully used within a variety of industries. At the same time, leadership-class software for high-performance and emerging application areas can be difficult for industrial scientists and engineers to use, for both technical and non-technical reasons. In this panel, we explore opportunities for reducing the impediments for using leadership scientific software in industrial settings.

Panelists to help explore expanding the use of leadership scientific software in industry are:

- TBD

Prompts:

- How do you use leadership scientific software (HPC, AI, and other emerging domains) in your work today?
- What are some successful attributes of the current technical and business environments in your use of this software?
- What are some key technical and non-technical impediments to using this software?
- What are some concrete steps that could improve your ability to leverage this software?

LSSw Meeting 7: April 21, 2022

Topic: Expanding the Scope of What is Reusable: A panel discussion

Description: Application-specific, libraries and tools have also had some success, for example, the Co-Design Centers sponsored by the Exascale Computing Project, but have received less attention and can be more challenging to sustain.

Panelists to help explore expanding the kinds of functionality that can be encapsulated for reuse:

- Ethan Coon, ORNL
- Angela Herring, LANL,
- Slaven Peles, ORNL
- Andrew Salinger, SNL
- Andrew Siegel, ANL

Prompts:

- Do you think there is value in designing, implementing, and delivering application-specific libraries, tools, and environments as reusable components?
- What has worked and not worked well with past efforts in this area?
- What are some near-term opportunities to componentize in your application area?
- How could this kind of software collection be adapted and sustained?

Ethan Coon

- Senior Research Scientist, Oak Ridge National Laboratory (Computational Hydrologist)
- Lead developer and community coordinator of the open-source code Amanzi-ATS, a 2020 R&D100 award winning Integrated Hydro-Terrestrial Model.
- Co-Lead of DOE's Biological & Environmental Research's "Cyberinfrastructure Working Group on Software Engineering"
- Likes writing multiscale, multiphysics code for climate research.

Prompts

From the perspective of the scientific software ecosystem you represent:

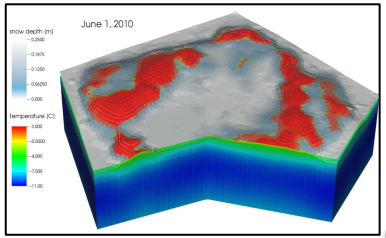
- Do you think there is value in designing, implementing, and delivering application-specific libraries, tools, and environments as reusable components?
- What has worked and not worked well with past efforts in this area?
- What are some near-term opportunities for componentization in your application area?
- How could this kind of software collection be adopted and sustained?

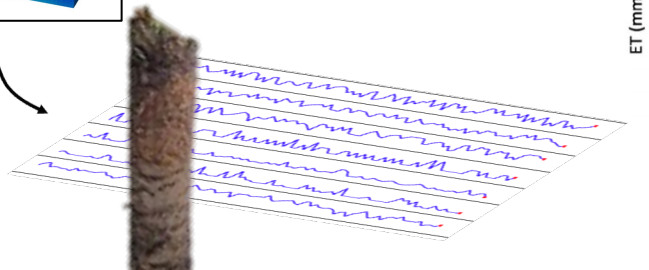
Environmental Systems Modeling across scales

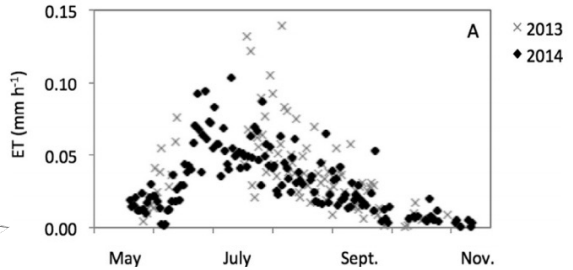
Global

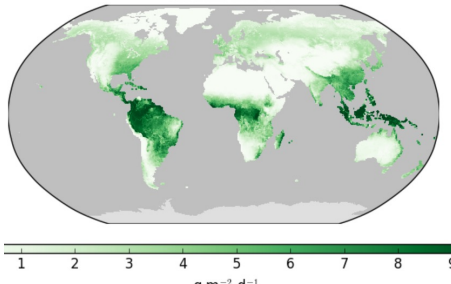
Regional

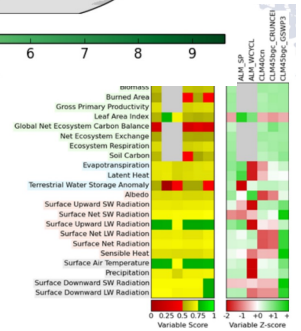
Local



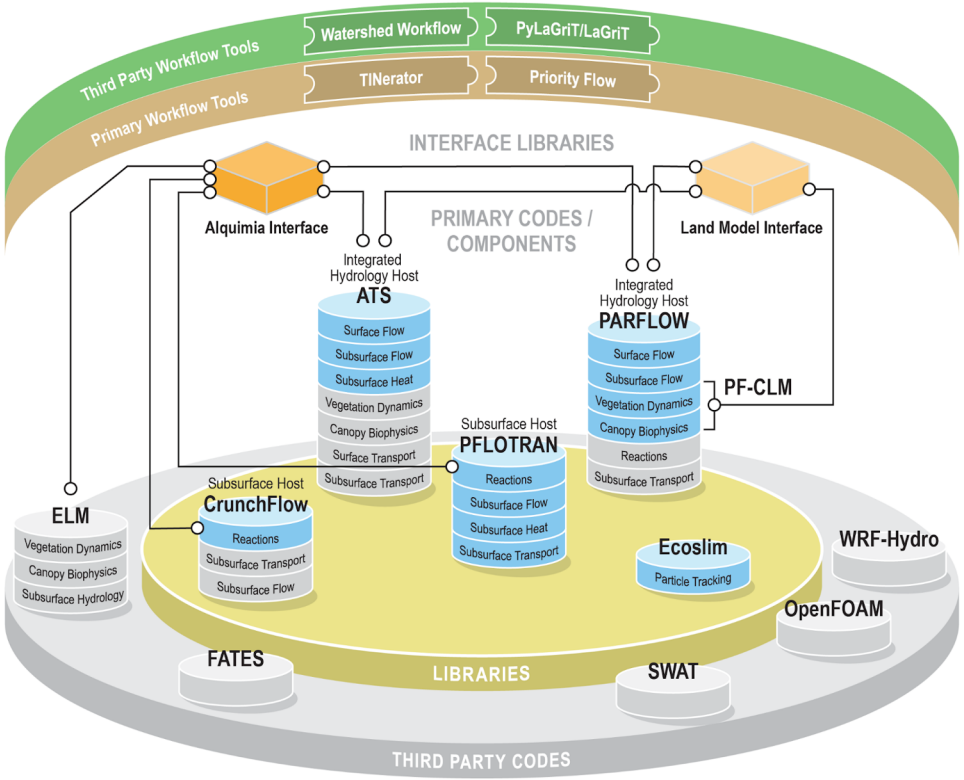








IDEAS-Watersheds Software Ecosystem



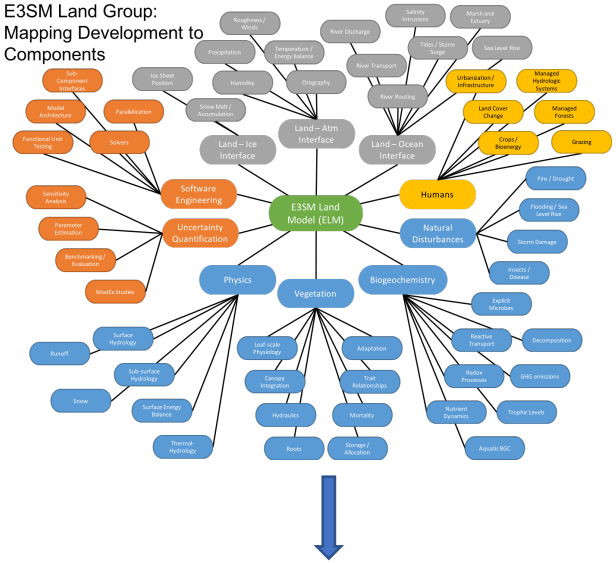
Workflow Tools support data collection and preprocessing, model setup, ensembles of simulations, post processing and analysis.

Codes and Interface Libraries support flexible mechanistic and multi-scale simulations of coupled hydro-biogechemical processes.

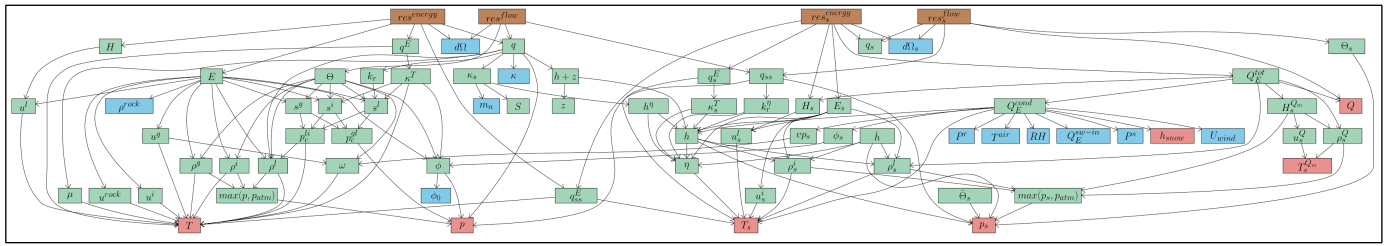
Primary Codes and Workflow Tools: Members of our team contribute to design and development.

Third Party Codes and Workflow Tools: We are users, development is led outside IDEAS-Watersheds.

Componentization



Amanzi-ATS dependency graph, formally stored in the code and generated at runtime.



Advantages of componentization are clear…

- Address process uncertainty
- Opportunity to introduce performance portable solutions
- Increased code reuse across projects
- Improved testing, maintainability, incremental improvement, …

Challenges are also clear and more often social than technical…

- Developers are scientists first, and the business model of code maintenance and refactoring isn't always clear.
- Institutional and community code memory
- Inconsistent granularity: "what is a component"
- Legacy code
- Poor testing practices

Continue to search for the right arguments to motivate progress.

Angela Herring

- Staff Scientist, Los Alamos National Laboratory
- Angela specializes in leading multi-disciplinary, Agile research teams
- Currently, she leads two research software teams:
  - Ristra team efforts:
    - The remap software Portage (www.github.com/laristra/portage)
    - An interface reconstruction library, Tangram (www.github.com/laristra/tangram).
  - Lynx: Focuses on applications of code-to-code linking

Bio source: https://bssw.io/events/strategies-for-working-remotely-making-the-transition-to-virtual-software-teams

Prompts

From the perspective of the scientific software ecosystem you represent:

- Do you think there is value in designing, implementing, and delivering application-specific libraries, tools, and environments as reusable components?
- What has worked and not worked well with past efforts in this area?
- What are some near-term opportunities for componentization in your application area?
- How could this kind of software collection be adopted and sustained?

Slaven Peles

- Computational Scientist, ORNL
- Experience related to scientific software development and use:
  - PI for ECP subproject ExaSGD – Grid Optimization at Exascale (2019-2021).
  - SUNDIALS library developer (2015-2019)
  - Software architect and lead developer for aerospace and energy systems commercial applications at United (Raytheon) Technologies Corp. (2008-2015).
- Scientific software ecosystem:
  - Computational tools for system integration and control design
  - Component-based modeling
  - Real-time simulations and rapid prototyping computations

Do you think there is value in delivering application-specific libraries as reusable components?

- I will argue that not only there is value in designing, implementing, and delivering application-specific libraries, tools, and environments as reusable components, but there is no viable alternative to this approach.

Digital thread and digital twin paradigms, as well as V-model product design introduce multi-fidelity computations and analyses within a single workflow. Without reusable components, the product design cost becomes prohibitive.

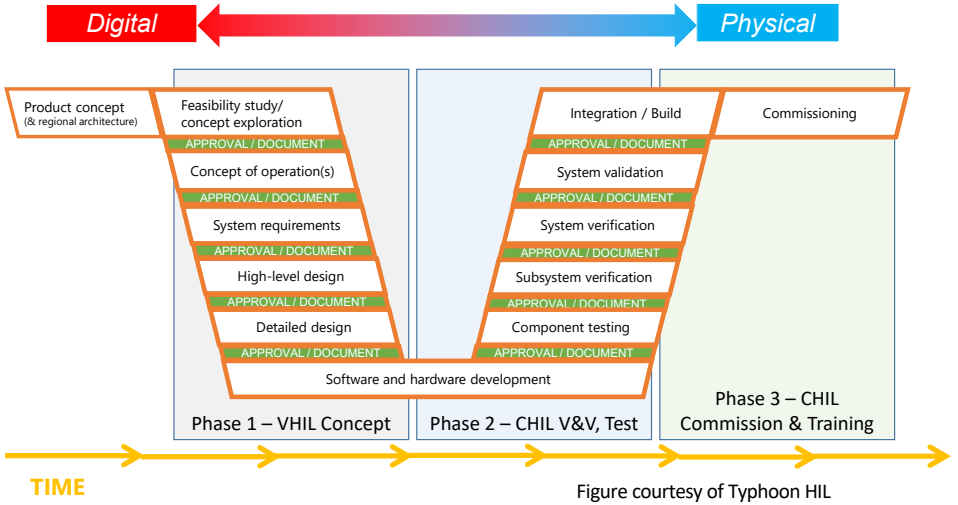
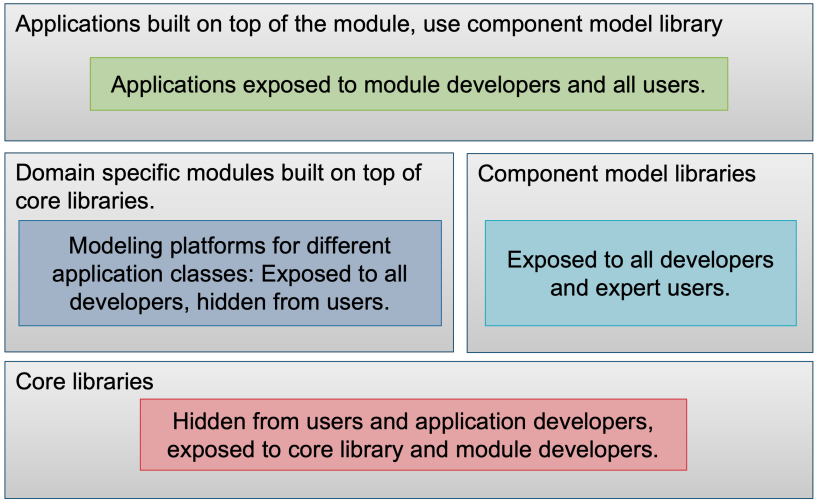


Figure courtesy of Typhoon HIL

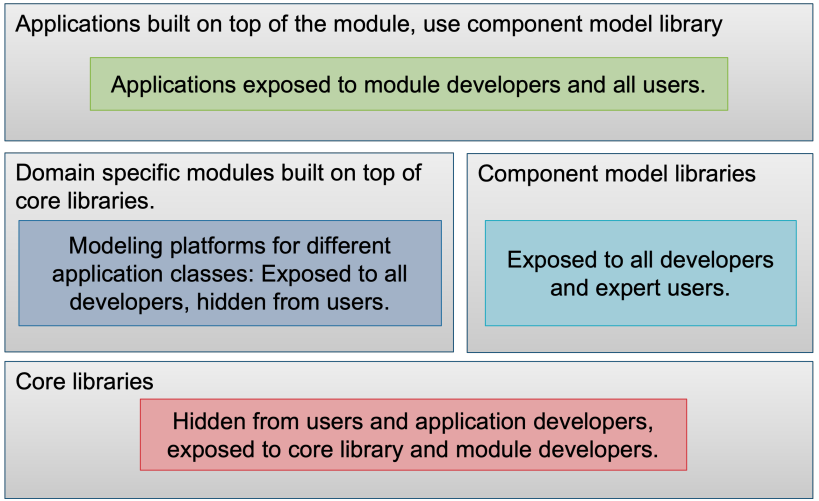
What has worked well with past efforts in this area?

- ✓Layered architecture that provides separation of responsibilities.
- ✓Model interfaces defined in terms of mathematical objects.
- ✓Reusable component models.
- ✓Laterally extensible design.



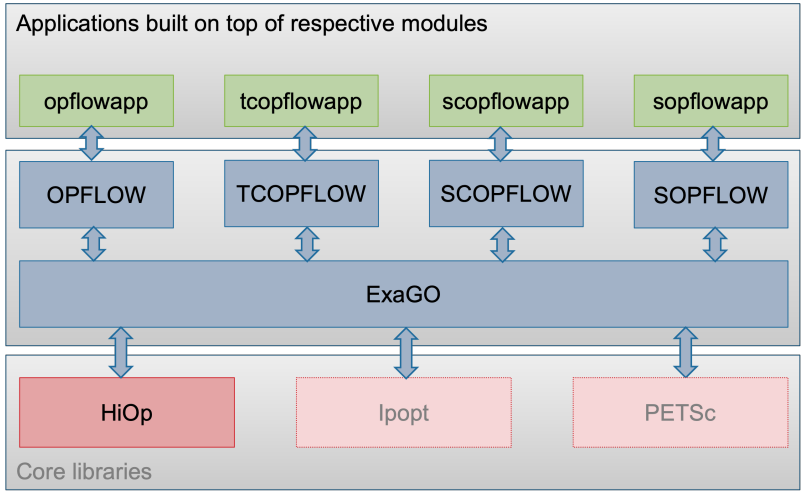
What has **not** worked well with past efforts in this area?

- ✗ Rushing towards implementation without sufficient attention to the software architecture.
- ✗ Model interfaces defined in terms of specific software implementation.
- ✗ Too many PhDs, too few software engineers on the project.



What are some near-term opportunities for componentization in your application area?

- Build upon success at ExaSGD project, in particular development of ExaGO and HiOp libraries
  - ExaGO modeling framework architected to provide responsibility separation and reusable components.
  - TODO: Develop component model libraries independent of analysis modules in ExaGO.
- HiOp and ExaGO are new additions to xSDK.



ExaGO designed by Shri Abhyankar, PNNL
HiOp designed by Cosmin Petra, LLNL

How could this kind of software collection be adopted and sustained?

- Funding
  - Dedicated funding for scientific software research and development.
  - Software quality assurance as a metric of success for research projects.
- Governance
  - Establish advisory (or management?) board to drive future development.
  - Insulate software design decisions from short-term project needs.
- User engagement
  - Bug reporting and feature requests
  - Training
  - Community outreach
  - Example: Xyce in classroom

Use Case: University of Toronto.

Teaching with Typhoon HIL and Xyce.

Transforming the power electronics laboratory from prototypes to detailed simulation models. From simple design tasks to full blown labs and examples.

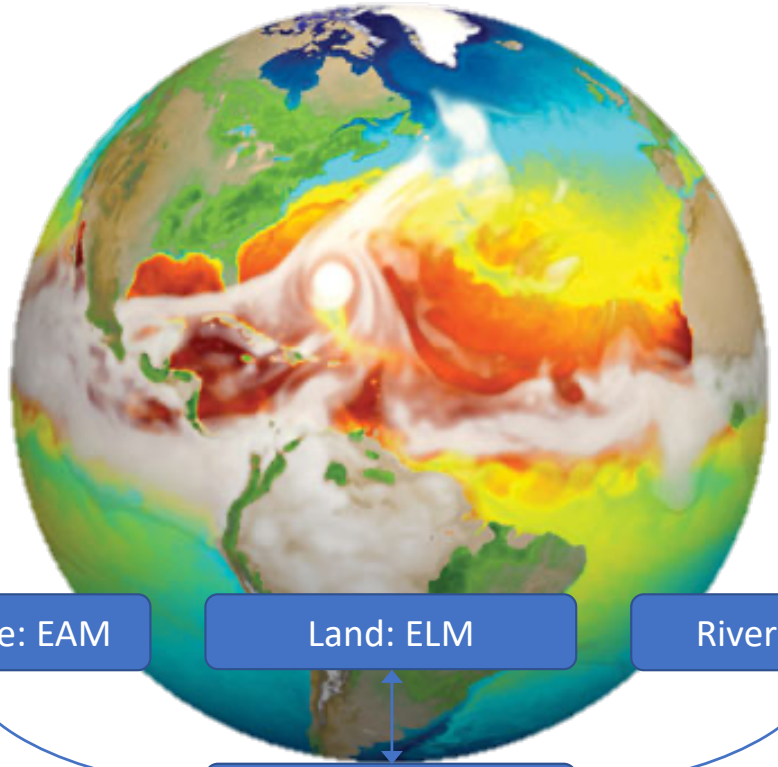
Transform your teaching today using the Typhoon HIL Virtual HIL device with Xyce, contact us to learn more!

Andy Salinger

Acting Senior Manger, Computational Science and Math Group, SNL

- 27 Years as computational scientist: Reacting flows, Bifurcations, HPC
- Last 10 years focused on Climate: E3SM code (DOE SC BER & ASCR)
  - E3SM Sub-Project PI for Next Gen Software and Algorithms (FY19-22)
  - PI of E3SM Software Modernization effort (FY17-19)
  - E3SM Group Lead for Software Engineering (FY15-19)
  - Early developer of MALI: Land Ice code in E3SM (FY12..)
- Biggest E3SM Accomplishments with regard to reusable software
  - New culture: “If there is a feature you care about, protect it with a test.”
  - New programming model: C++/Kokkos-based Atmosphere model
  - Use of reusable math software: MALI code – built on dozens of Trilinos libraries

E3SM
Energy Exascale
Earth System Model



Atmosphere: EAM

Land: ELM

River: MOSART

Flux Coupler: CPL

Ocean: MPAS-O

Sea Ice: MPAS-SI

Land Ice: MALI

1. Do you think there is value in designing, implementing, and delivering application-specific libraries, tools, and environments as reusable components?
   Yes, but I am not representative of the climate code development community.
   Prevailing attitude is: "Collaborate with Math/CS folks until we find a good algorithm, then recode it."
2. What has worked and not worked well with past efforts in this area?
   - **Well**: Climate models are componentized, coarsely: Ocean, Atmosphere, Radiation, …
   - **Not well**: Every new math model corresponds to a new code.
   - **Not Well**: Atmosphere physics sub-components communicate indiscriminately through a data workspace – not really libraries with an interface
3. What are some near-term opportunities to componentize in your application area?
   - EKAT utility library: Common Kokkos environment, typdefs, SIMD-Packs, across all of E3SM
   - SCREAM-AD: Atmosphere Driver: Coupling layer with "Atm process" abstraction. Forces componentization of indivual atmosphere processes
   - Aerosol/Chemistry Interface working group: Discussion across several modeling centers to agree on common interfaces for aerosol processes and common format for chemistry specifications
4. How could this kind of software collection be adapted and sustained?
   - Challenging: Interoperability, flexibility, are not highly valuable
     - Each climate model release has 1 implementation of each component or process.
     - Each climate model has separate funding source and sponsor.

Andrew Siegel

- Senior Scientist, Lead Nuclear Energy Advanced Simulation, ANL
- Relevant experience
  - Past
    - Geophysics (ocean basin modeling): QGMG
    - Astrophysics: Chief architect of FLASH
  - Last 15 years
    - Advanced nuclear energy (reactor core simulation): OpenMC, Nek
    - PI of CESAR co-design Center
    - Fusion tokamak whole device modelling
  - Broader leadership roles
    - National Technical Director NEAMS (2010-2011)
    - Director of Application Development, ECP (2017-)

- Generally, **yes there is value**, with some qualifications
  - What is (1) hard in practice vs. (2) hard
    1. Typically, domain expertise must be blended with math, computer science, and software engineering expertise. No one single specialty covers all aspects of what is needed. physicists might not have proper expertise to implement abstraction in code, etc.
    2. Abstractions are extremely hard to identify. Many code artifacts are better left as one-offs even if they are duplicative. It should not be taken as obvious or trivially true that anything can easily be abstracted.
  - Often better to test/prove ideas within one code, generalize and accrete capabilities vs. premature top down approach
  - In most cases potential for sustainability greatly lowers risk of adoption. (Alternative is to adopt source code directly)
- Strong overlap with ECP motif-based co-design
  - Block structured AMR, unstructured mesh PDEs, particle codes, graph algorithms, etc.
  - Many successes, but how scalable are these models?

- What has worked well
  - Many examples of success prior to ECP: TCE, libxc, constitutive models, equations of state, BLAST, Parmetis, coupling tools (DTK), adaptive mesh refinement, vis, …
  - Within ECP: ECP *Co-design Centers:* embedded ST with exemplar applications
  - Representation from domain experts early leads to community buy in.
  - Source code adoption or longer term investment/vision greatly lowers risk of adoption.
  - Understanding of level of abstraction / potential user community tradeoffs
    - High level of abstraction (close to application) → very useful, many fewer users
    - Low level of abstraction (common low level infrastructure) → less value, many users
  - Layered, flexible APIs that don't require all-in
- What hasn't worked well
  - Most things
  - Capability that is designed too top down, too little early interaction with apps
  - Capability that is unaware of built-in assumptions
  - Most decisions dictated at a strategic or managerial level

- Near-term opportunities
  - Common nuclear data processing code (similar to JNOY)
  - Common particle-geometry API (distance_to_surface, etc.)
  - Common adaptively refined mesh infrastructure (including solvers)
  - Scalable Poisson solver on unstructured mesh (incompressible CFD)
  - Domain-specific feature detection, data compression

- Can generalize to generalized physics components
  - CFD
  - Particle transport on arbitrary geometries
  - Structural mechanics

- Huge range of sustainability requirements
  - Some components endure with low to modest investments for relatively long periods; other require substantial continual investments
  - That said: zero investment is a losing strategy in all almost cases.
  - Biggest barriers to adoption
    - Doesn't do exactly what I need it to, especially as my needs evolve, source code too complex to modify myself, no one to work with.
    - It does what I want but likely won't be around in a few years.

- Sustainability is difficult because it implies picking winners and losers in a research environments.
- Nonetheless this is already done in bits and pieces in an *ad hoc* way.
- Need to formalize process: new ideas funded as research, proven tools (adopted by apps) can graduate to have some predictable, longer term sustainability funds that complement other R&D funding sources.