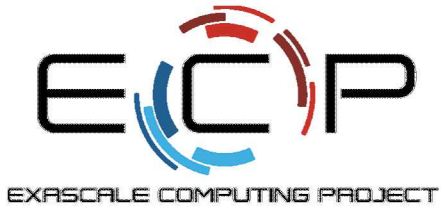
SAND2020-1885C



XSDK: Toward Efficient and Interoperable Scientific Library Collection

Keita Teranishi and xSDK Developers

Sandia National Laboratories, California

Feb 15, 2020

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

Who are we?

- Developers of **high-quality, robust, portable high-performance math libraries**

hypre high performance preconditioners
PETSc/TAO
PHIST
MAGMA
mfem
AMReX
SuperLU
TRILINOS
sundials
PLASMA
PUMI
STRUMPACK
Omega_h
DTK
TASMANIAN
SLEPc
deal.II

Why are we leading this tutorial?

- Explain how the xSDK's approach toward a scientific software ecosystem **improves quality, sustainability, and combined use of independent packages**, as needed for extreme-scale computational science and engineering
- Encourage discussions with package developers: **How you can leverage math libraries for extreme-scale computational science**

xSDK
ECP EXASCALE COMPUTING PROJECT

xSDK collaborators

xSDK Release 0.5.0, Nov 2019

- **xSDK release lead:** Jim Willenbring, SNL
- **xSDK PI**
  - Ulrike Meier Yang (LLNL)
- **Leads for xSDK testing**
  - Satish Balay, ANL: ALCF testing
  - Piotr Luszczek, UTK: OLCF testing
  - Aaron Fischer, LLNL: NERSC testing
  - Cody Balos, LLNL: general testing
  - Keita Teranishi, SNL: general testing
- **Spack liaison:** Todd Gamblin, LLNL

- **Package compatibility with xSDK community policies and software testing:**
  - **AMReX**: Ann Almgren, Michele Rosso (LBNL)
  - **DTK**: Stuart Slattery, Bruno Turcksin (ORNL)
  - **deal.II**: Wolfgang Bangerth (Colorado State University)
  - **hypre**: Ulrike Meier Yang, Sarah Osborn, Rob Falgout (LLNL)
  - **Gunkgo**: Hartwig Anzt, (Karlsruhe Institute of Technology/UTK)
  - **libEnsemble**: Stephen Hudson (ANL)
  - **MAGMA** and **PLASMA**: Piotr Luszczek (UTK)
  - **MFEM**: Aaron Fischer, Tzanio Kolev (LLNL)
  - **Omega_h**: Dan Ibanez (SNL)
  - **PETSc/TAO**: Satish Balay, Alp Denner, Barry Smith (ANL)
  - **PUMI**: Cameron Smith (RPI)
  - **preCICE**: Hans-Joachim Bungartz (TU Munich)
  - **SUNDIALS**: Cody Balos, David Gardner, Carol Woodward (LLNL)
  - **SuperLU** and **STRUMPACK**: Sherry Li and Pieter Ghysels (LBNL)
  - **TASMANIAN**: Miroslav Stoyanov, Damien Lebrun Grandie (ORNL)
  - **Trilinos**: Keita Teranishi, Jim Willenbring, Sam Knight (SNL)
  - **PHIST**: Jonas Thies (DLR, German Aerospace Center)
  - **SLEPc**: José Roman (Universitat Politècnica de València)
  - **Alquimia**: Sergi Mollins (LBNL)
  - **PFLOTRAN**: Glenn Hammond (SNL)

and many more …

Software libraries facilitate progress in computational science and engineering

- **Software library:** a high-quality, encapsulated, documented, tested, and multiuse software collection that provides functionality commonly needed by application developers
  - Organized for the purpose of being reused by independent (sub)programs
  - User needs to know only
    - Library interface (not internal details)
    - When and how to use library functionality appropriately

- **Key advantages** of software libraries
  - Contain complexity
  - Leverage library developer expertise
  - Reduce application coding effort
  - Encourage sharing of code, ease distribution of code

- **References:**
  - https://en.wikipedia.org/wiki/Library_(computing)
  - What are Interoperable Software Libraries? Introducing the xSDK

Why is reusable scientific software important?

User perspective:

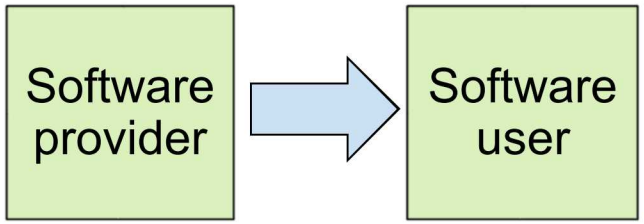
Focus on primary interests

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

Provider perspective:

Share your capabilities

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



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

Software libraries are not enough

- Well-designed libraries provide critical functionality … But alone are not sufficient to address all aspects of next-generation scientific simulation and analysis.

- Applications need to use software packages **in combination** on ever evolving architectures

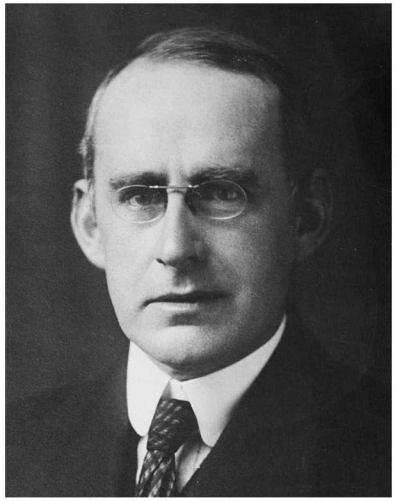
> **"The way you get programmer productivity is by eliminating lines of code you have to write."**
>
> – Steve Jobs, Apple World Wide Developers Conference, Closing Keynote, 1997

Need software ecosystem perspective

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

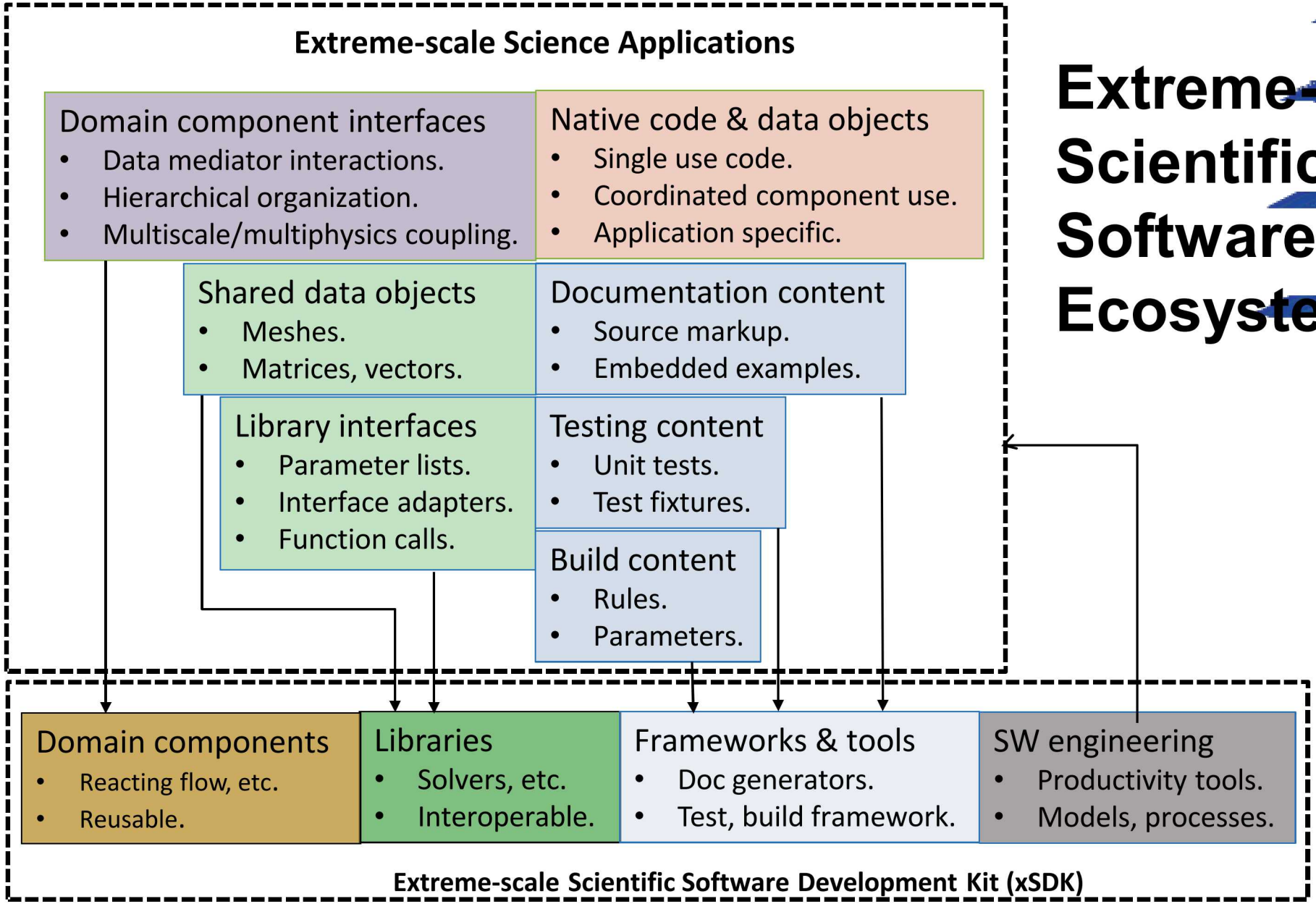
Ecosystems are challenging!

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



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

Extreme-scale Scientific Software Ecosystem



Difficulties in combined use of independently developed software packages

Challenges:

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

Ref: What are Interoperable Software Libraries? Introducing the xSDK

Levels of package interoperability:

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

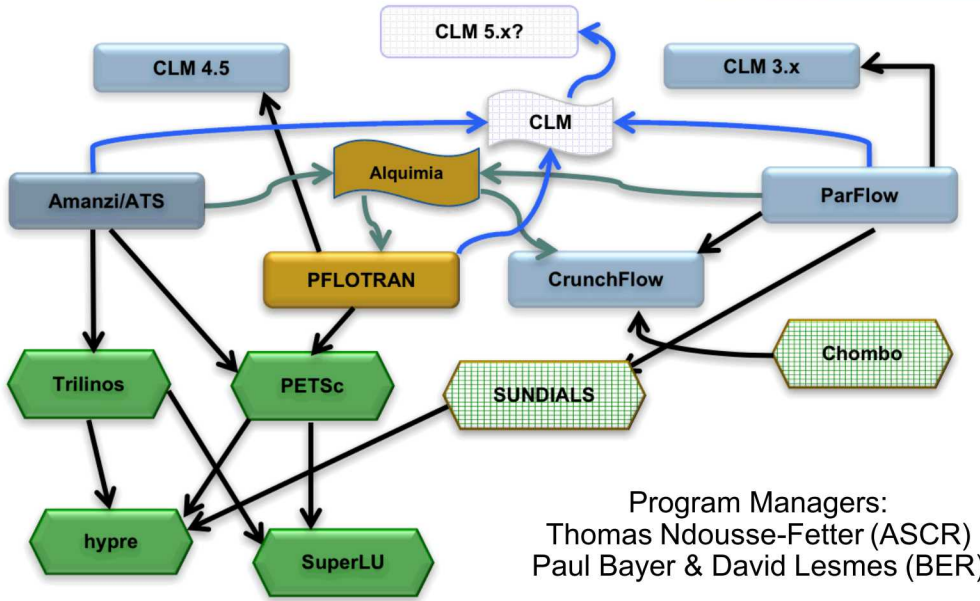
Motivation and history of xSDK

Next-generation scientific simulations require combined use of independent packages

- Installing multiple independent software packages is tedious and error prone
  - Need consistency of compiler (+version, options), 3rd-party packages, etc.
  - Namespace and version conflicts make simultaneous build/link of packages difficult
- Multilayer interoperability among packages requires careful design and sustainable coordination
- **Prior to xSDK effort, could not build required libraries into a single executable due to many incompatibilities**

xSDK history: Work began in ASCR/BER partnership, IDEAS project (Sept 2014)

Needed for BER multiscale, multiphysics integrated surface-subsurface hydrology models



Program Managers:
Thomas Ndousse-Fetter (ASCR)
Paul Bayer & David Lesmes (BER)

xSDK

ECP EXASCALE COMPUTING PROJECT

xSDK community policies

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

https://xsdk.info/policies

xSDK compatible package: Must satisfy mandatory xSDK policies:

M1. Support xSDK community GNU Autoconf or CMake options.

M2. Provide a comprehensive test suite.

M3. Employ user-provided MPI communicator.

M4. Give best effort at portability to key architectures.

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

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

M7. Come with an open source license.

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

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

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

M11. Have no hardwired print or IO statements.

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

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

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

M15. All xSDK compatibility changes should be sustainable.

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

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

R1. Have a public repository.

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

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

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

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

R6. Provide versions of dependencies.

R7. Provide README, SUPPORT, LICENSE and CHANGELOG files or their equivalent.

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

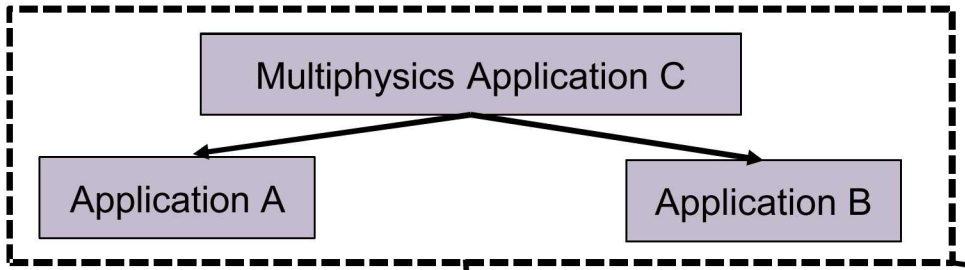
xSDK

ECP EXASCALE COMPUTING PROJECT

xSDK History: Version 0.1.0: April 2016

https://xsdk.info

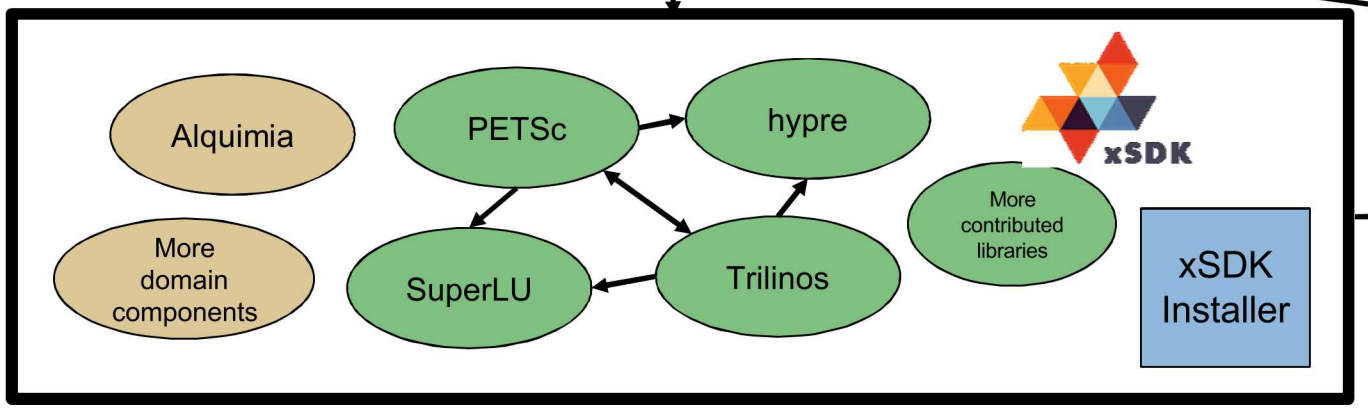
Notation: A → B:
A can use **B** to provide functionality on behalf of **A**

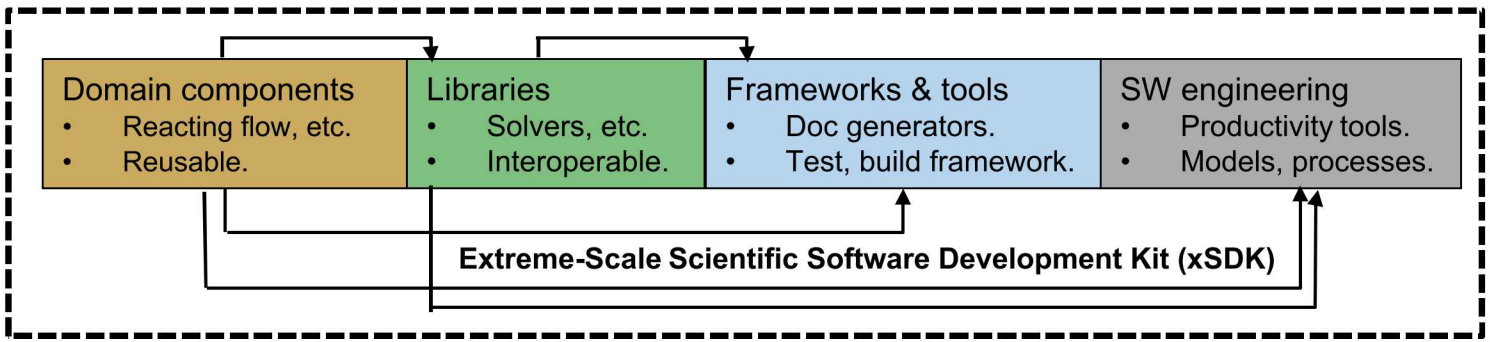


xSDK functionality, April 2016

Tested on key machines at ALCF, NERSC, OLCF, also Linux, Mac OS X

April 2016
- 4 math libraries
- 1 domain component
- PETSc-based xSDK installer
- **14 mandatory (5 rec.) xSDK community policies**

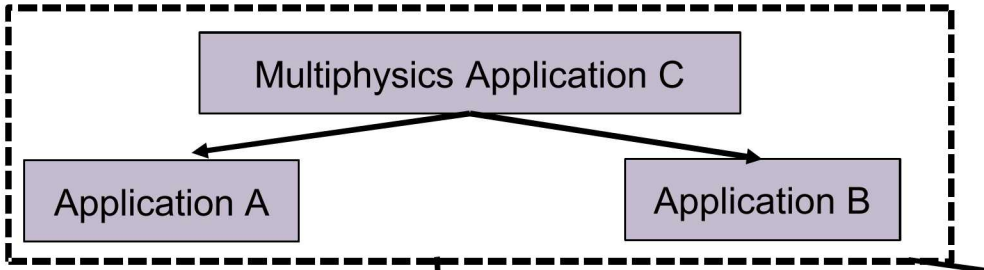




xSDK History: Version 0.5.0: November 2019

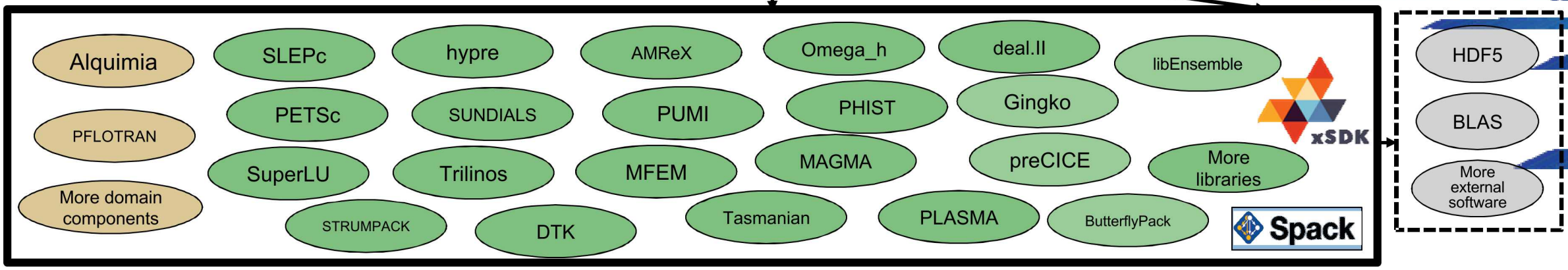
https://xsdk.info

Each xSDK member package uses or can be used with one or more xSDK packages, and the connecting interface is regularly tested for regressions.



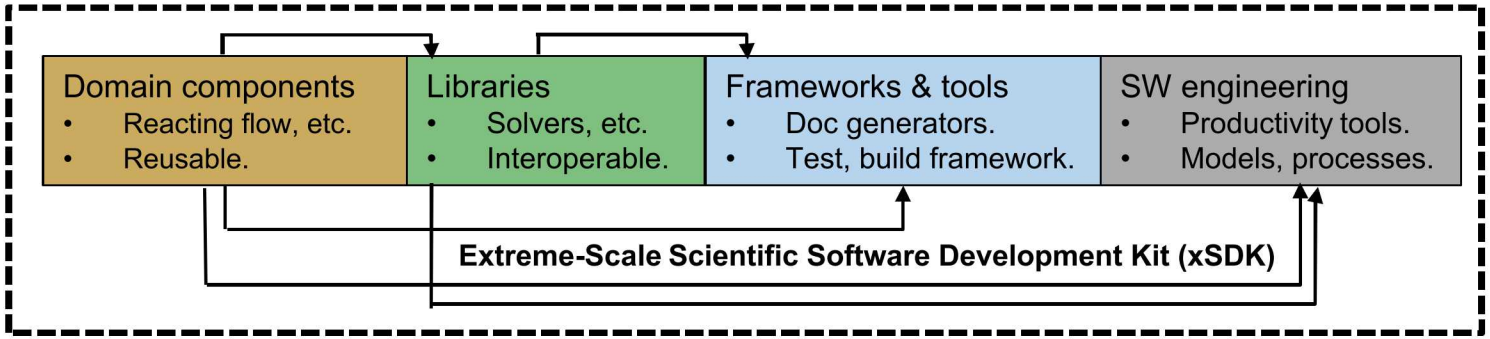
xSDK functionality, Nov 2019

Tested on key machines at ALCF, NERSC, OLCF, also Linux, Mac OS X



November 2019

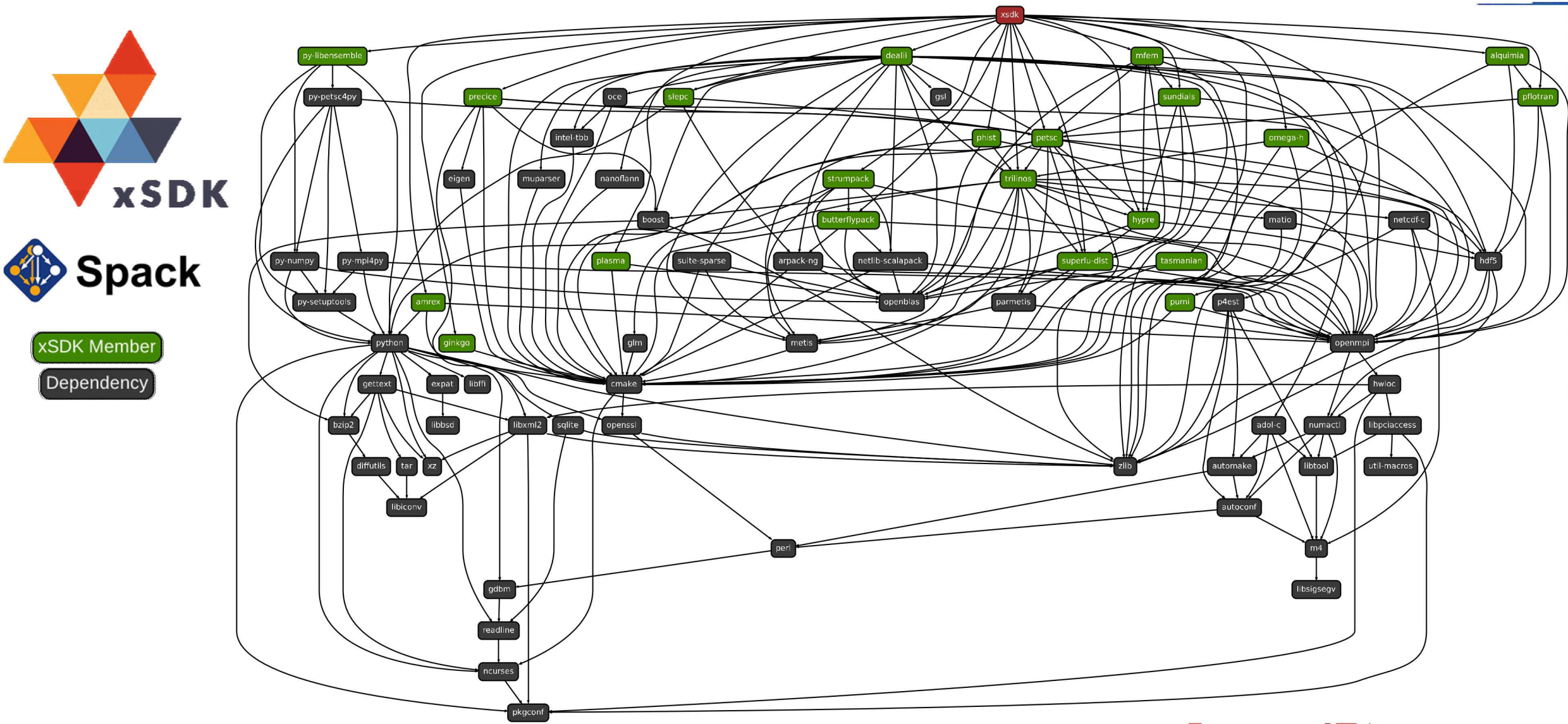
- 21 math libraries
- 2 domain components
- **16 mandatory (7 rec) xSDK community policies**
- Spack xSDK installer



Impact: Improved code quality, usability, access, sustainability

Foundation for work on performance portability, deeper levels of package interoperability

xSDK Dependency Graph



xSDK community policies

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

https://xsdk.info/policies

xSDK compatible package: Must satisfy mandatory xSDK policies:

M1. Support xSDK community GNU Autoconf or CMake options.

M2. Provide a comprehensive test suite.

M3. Employ user-provided MPI communicator.

M4. Give best effort at portability to key architectures.

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

M6. R… packa…

M7. C… …f library

M8. R… softw…

M9. U… name… …NGELOG

M10. …

M11. …

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

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

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

M15. All xSDK compatibility changes should be sustainable.

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

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

R1. Have a public repository.

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

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

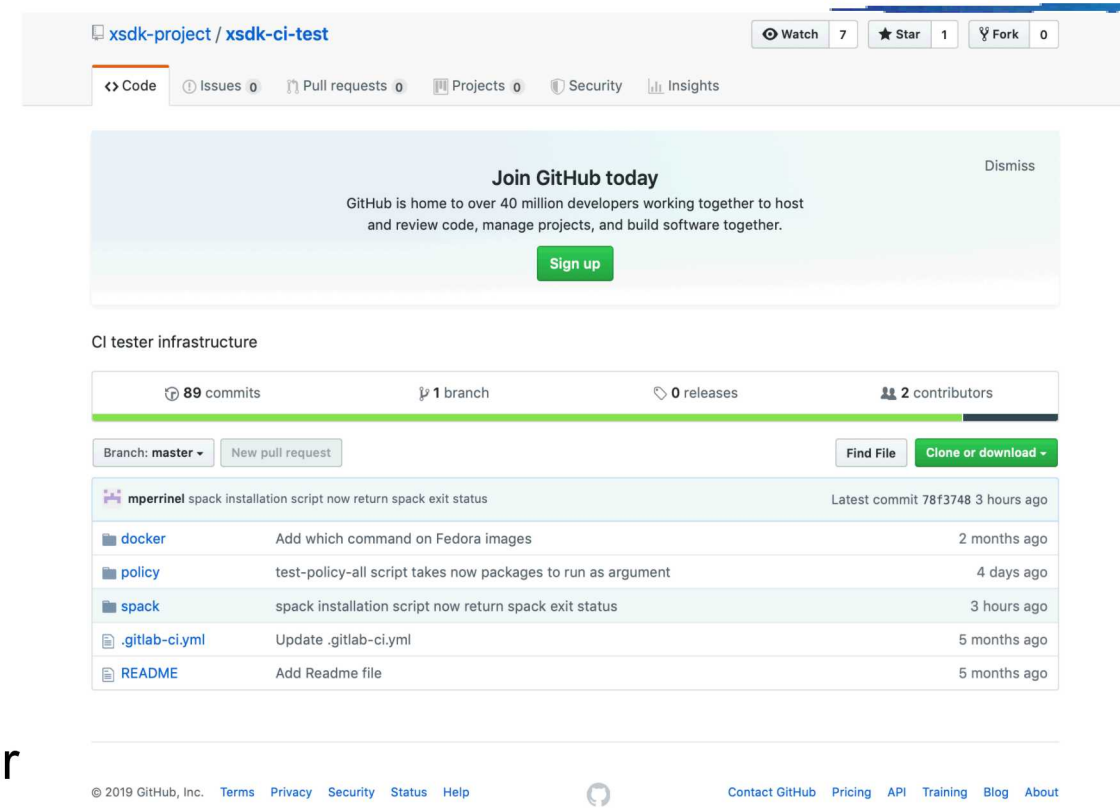
…s soon as

…K-compatible package, and it does or can be used by another package in the xSDK, and the connecting interface is regularly tested for regressions.

How to check the policy compliance?

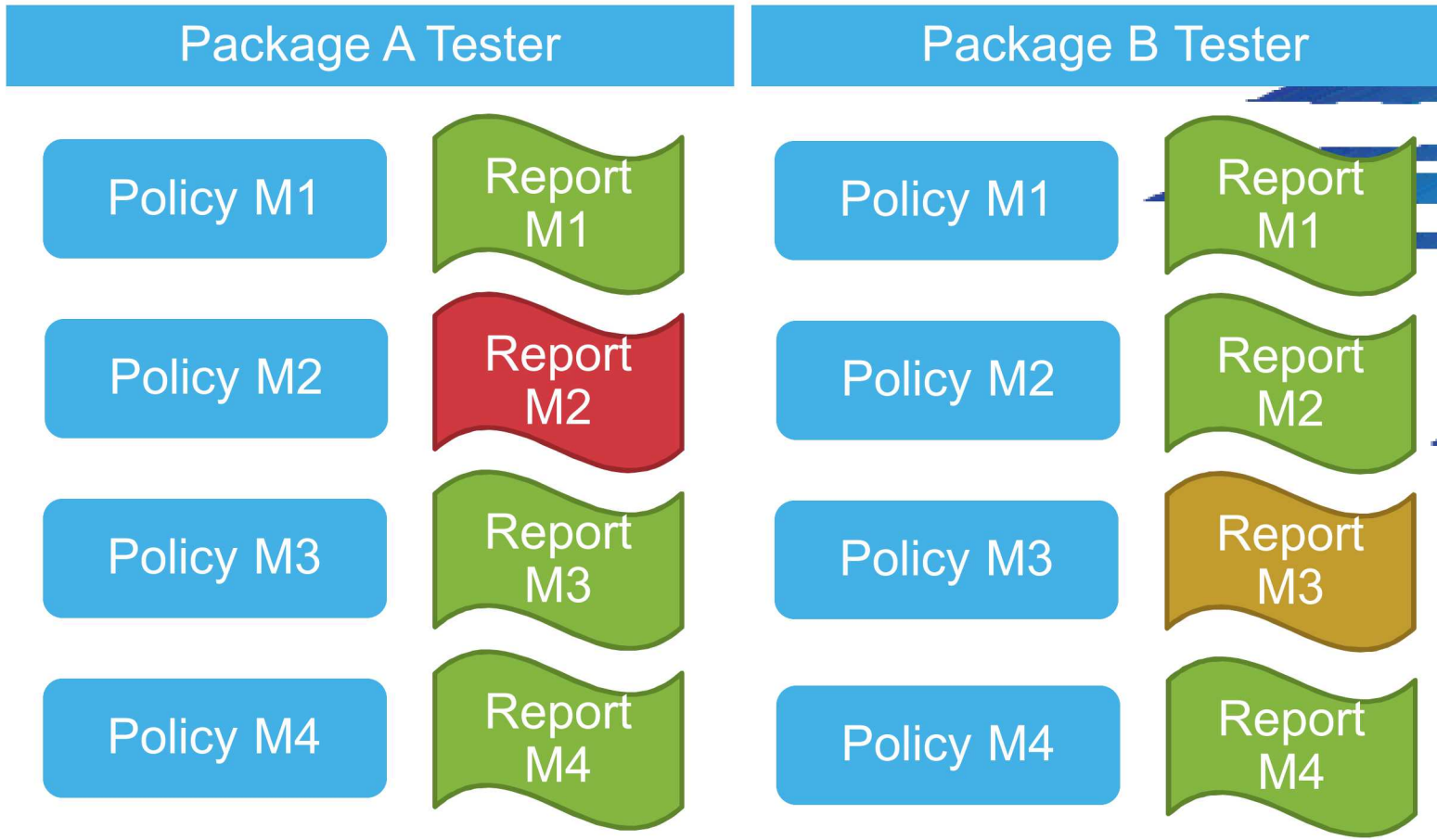
xSDK and Gitlab-CI

- xSDK CI is a Github project that contains:
  - xSDK Spack installation scripts
  - Policy testing scripts
  - Minimal docker images prepared for xSDK testing
- CI pipeline described in gitlab-ci.yml:
  - Pull the xSDK CI project
  - Pull the Spack project
  - Run the xSDK installation scripts
  - Run the policy testing scripts
  - Report all results as a set of artefacts for all xSDK inner packages
  - Repeat for every targeted platform using docker xSDK images
  - (FY20: Built/Run interoperability example programs)



Compliance with xSDK policies

- Using scripted tests to verify compliance
  - Every policy has been analyzed to establish a list of tests that can be automatically verified
  - For every package, the tests produce reports that can be viewed in the CI



xSDK Policy Testing Design

- Every testable mandatory policy has a dedicated testing script which outputs a report.
  - e.g., the first mandatory policy (M1) has test (m1.sh) with corresponding report (report_m1.log)
- All tests are run daily; they can also be manually triggered on the Gitlab-CI
- xSDK Spack installation is done by running *spack install –keep-stage –source xsdk* that keep the source and the build directories of every inner package.
- Source and build directories are used as an input parameter for the policy tests
  - e.g. the m1.sh run for every inner package takes the corresponding source directory as input.

xSDK Policy Testing Design, Examples

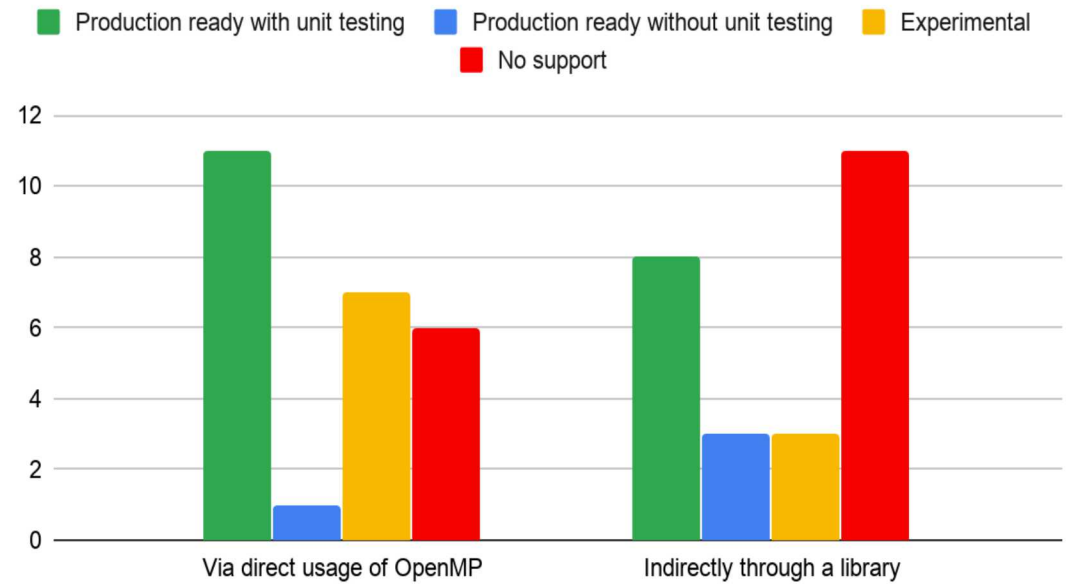
M1. Support xSDK community GNU Autoconf or CMake options.	1) Check the CMakeLists.txt or Configure file existence. 2) Interoperability between packages can be tested here with packages inclusion cmake option -DENABLE_<package>. 3) An option —dis/enable-xsdk-defaults must exist.
M3 Employ user-provided MPI communicator.	1) Scan the code and detect if MPI_COMM_WORLD is called which could making this requirement failed. 2) Check for the option of the MPI error-handling prevention
M5. Provide a documented, reliable way to contact the development team.	1) Read the document and parse some information : - Website of the package - Email address of the package owner 2) Ping the website of the package

Runtime Support of xSDK

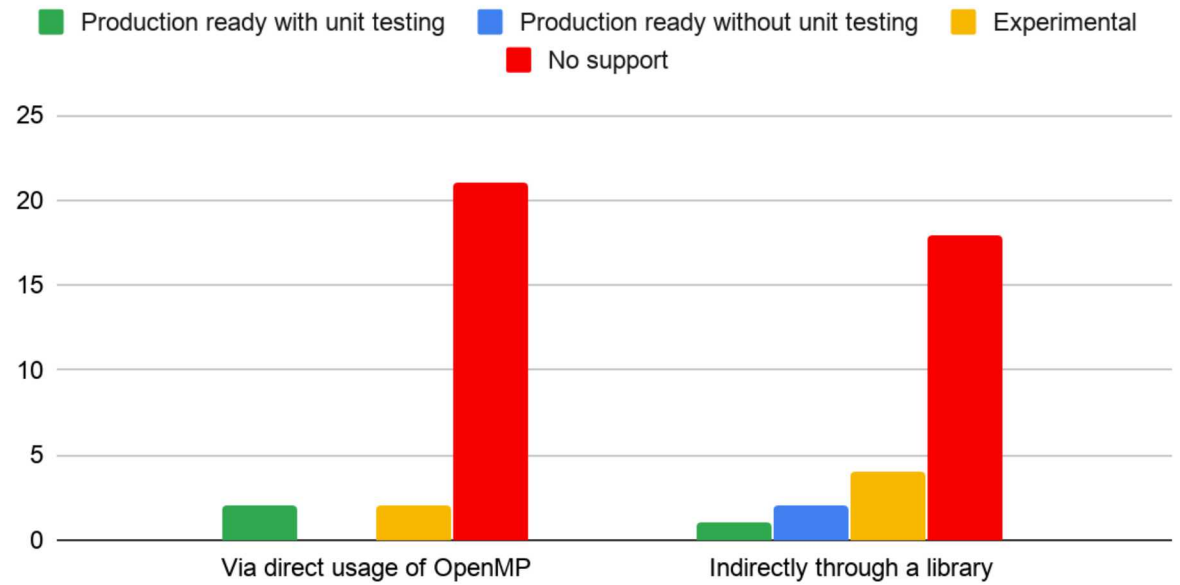
Survey on xSDK and FFTX, heFFTe and SLATE packages

OpenMP Support

Does your library have OpenMP CPU multi-threading capabilities? What is the status?



Does your library have OpenMP target offloading capabilities? What is the status?

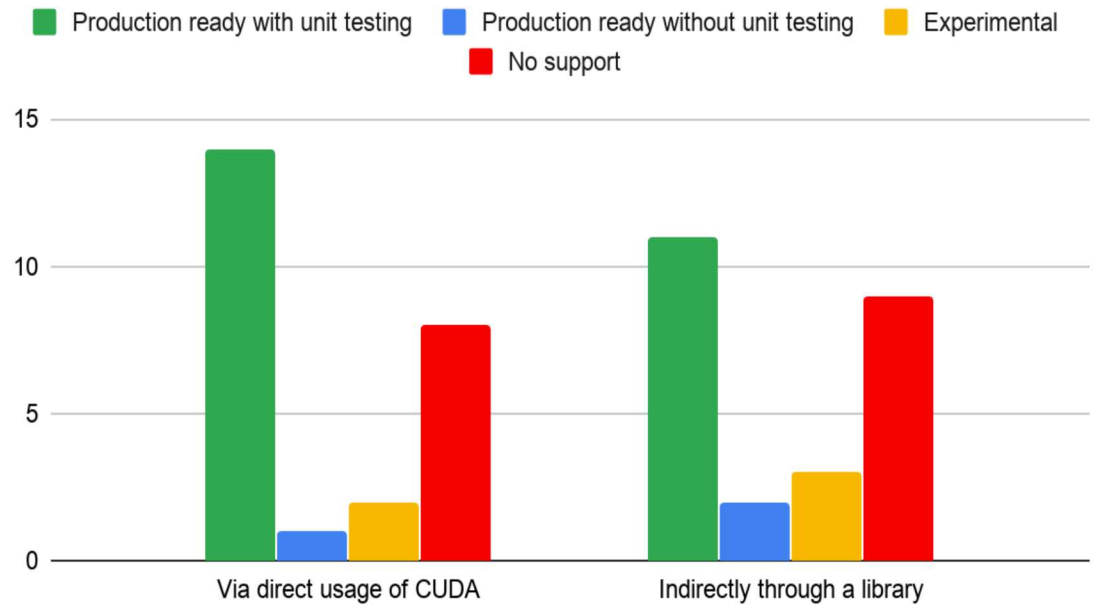


xSDK

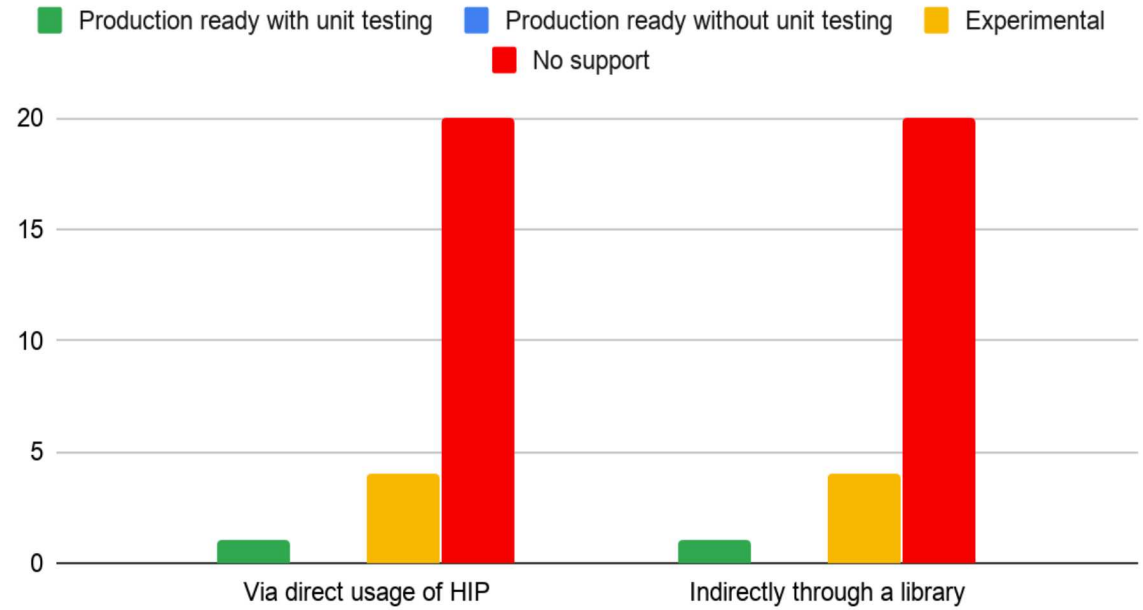
ECP EXASCALE COMPUTING PROJECT

CUDA and HIPS Support

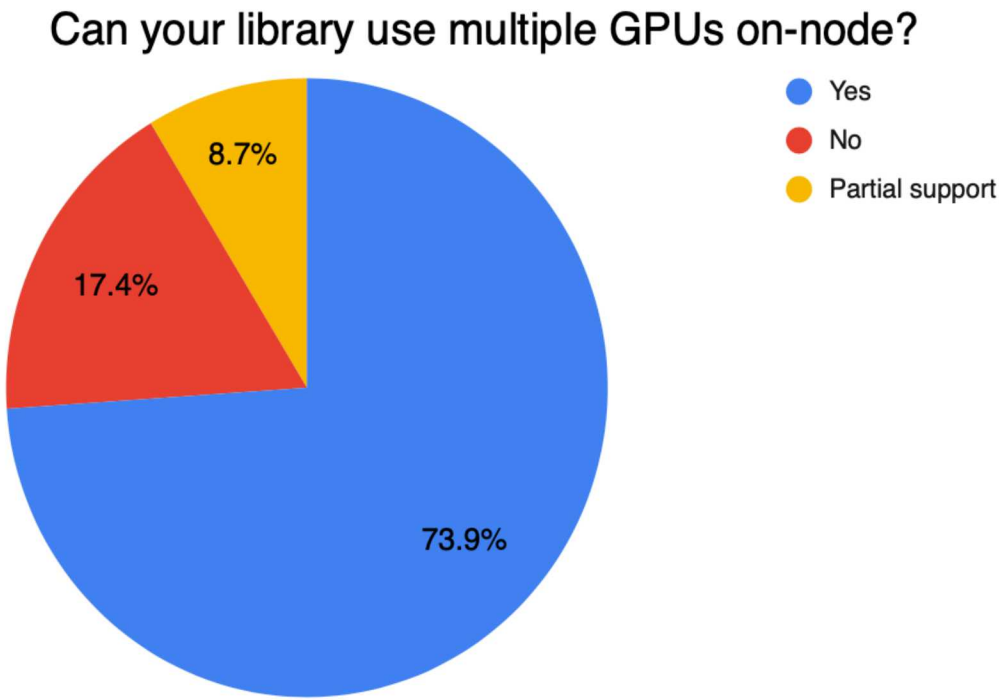
Does your library have CUDA capabilities? What is the status?

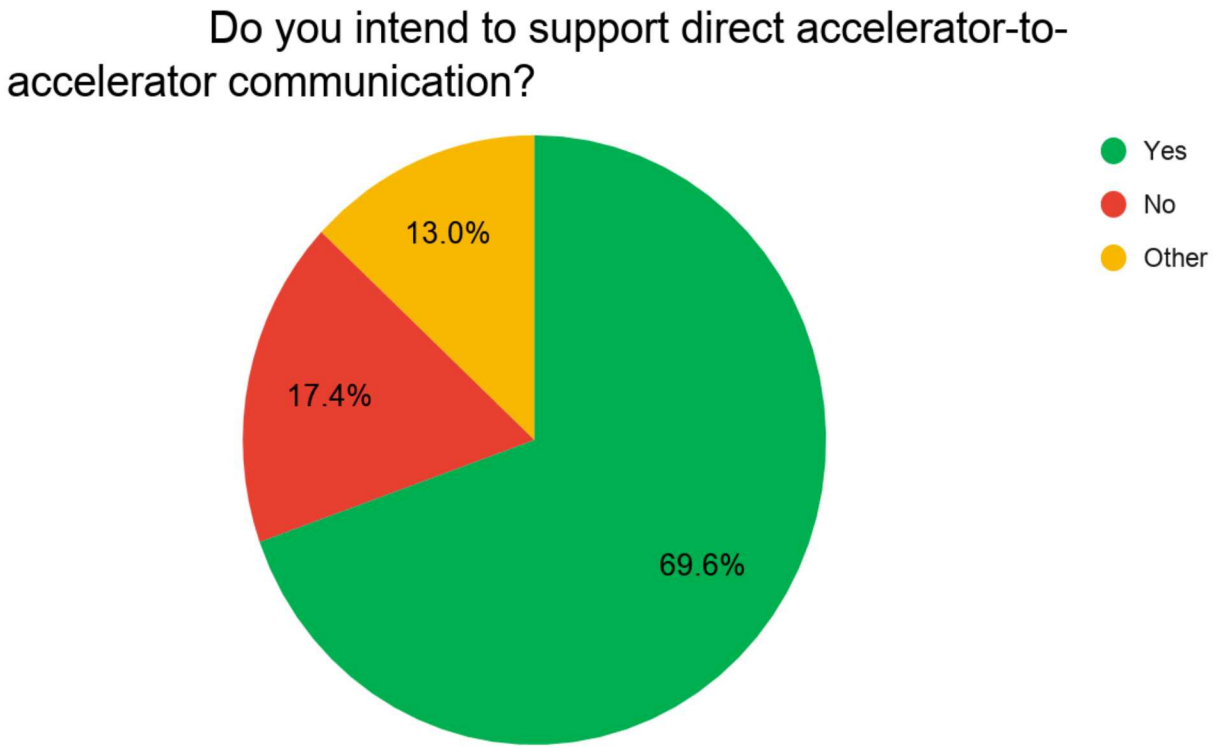


Does your library have HIP capabilities? What is the status?



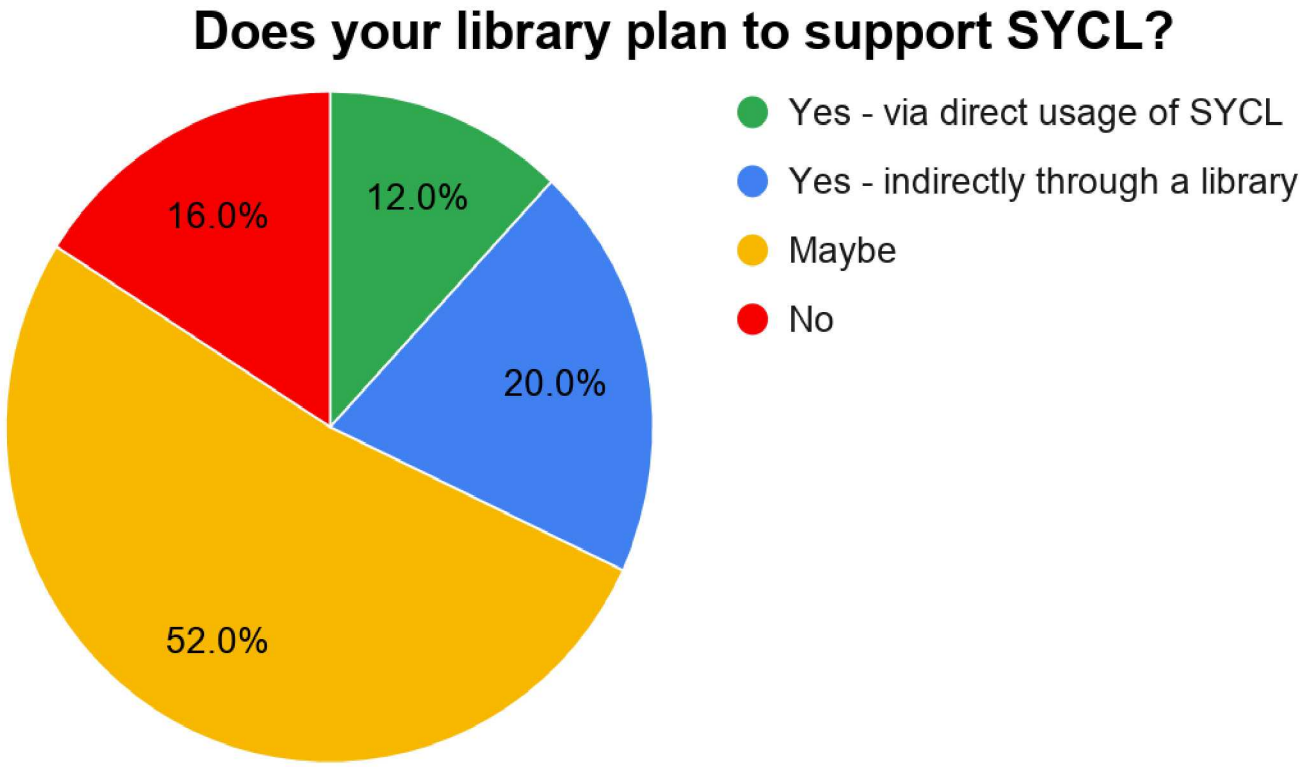
Accelerator Support





SYCL Support

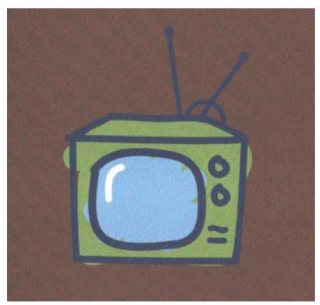
AMReX	
ButterflyPACK	
DTK	
deal.II	
FFTX*	
Ginkgo	
heFFTe*	
hypre	
libEnsemble	
MAGMA	
MFEM	
Omega_h	
PETSc/TAO	
phist	
PLASMA	
preCICE	
PUMI	
SLATE*	
SLEPc	
STRUMPACK	
SUNDIALS	
SuperLU	
Tasmanian	



Impact of xSDK Software Policies

- Improved code quality, and usability of individual libraries (or application codes)
- Addresses challenges in interoperability and sustainability of software developed by diverse groups at different institutions
- Enables common build of libraries
- Foundation for work on deeper levels of interoperability and performance portability
- Base for new sets of software policies
- Engages community

It takes all kinds.



Think outside the box.

Some useful links

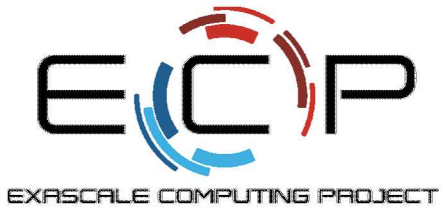
- http://xsdk.info/
- http://xsdk.info/policies/
- https://github.com/xsdk-project/xsdk-community-policies
- http://ideas-productivity.org/resources/howtos/

Acknowledgments

- This work was supported by the U.S. Department of Energy Office of Science, Office of Advanced Scientific Computing Research (ASCR), and by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration.

xSDK

ECP EXASCALE COMPUTING PROJECT



Questions?