The Extreme-scale Scientific Software Stack (E4S) for Collaborative Open Source Software

Michael A. Heroux, Sandia National Laboratories
Director of Software Technology, US Exascale Computing Project

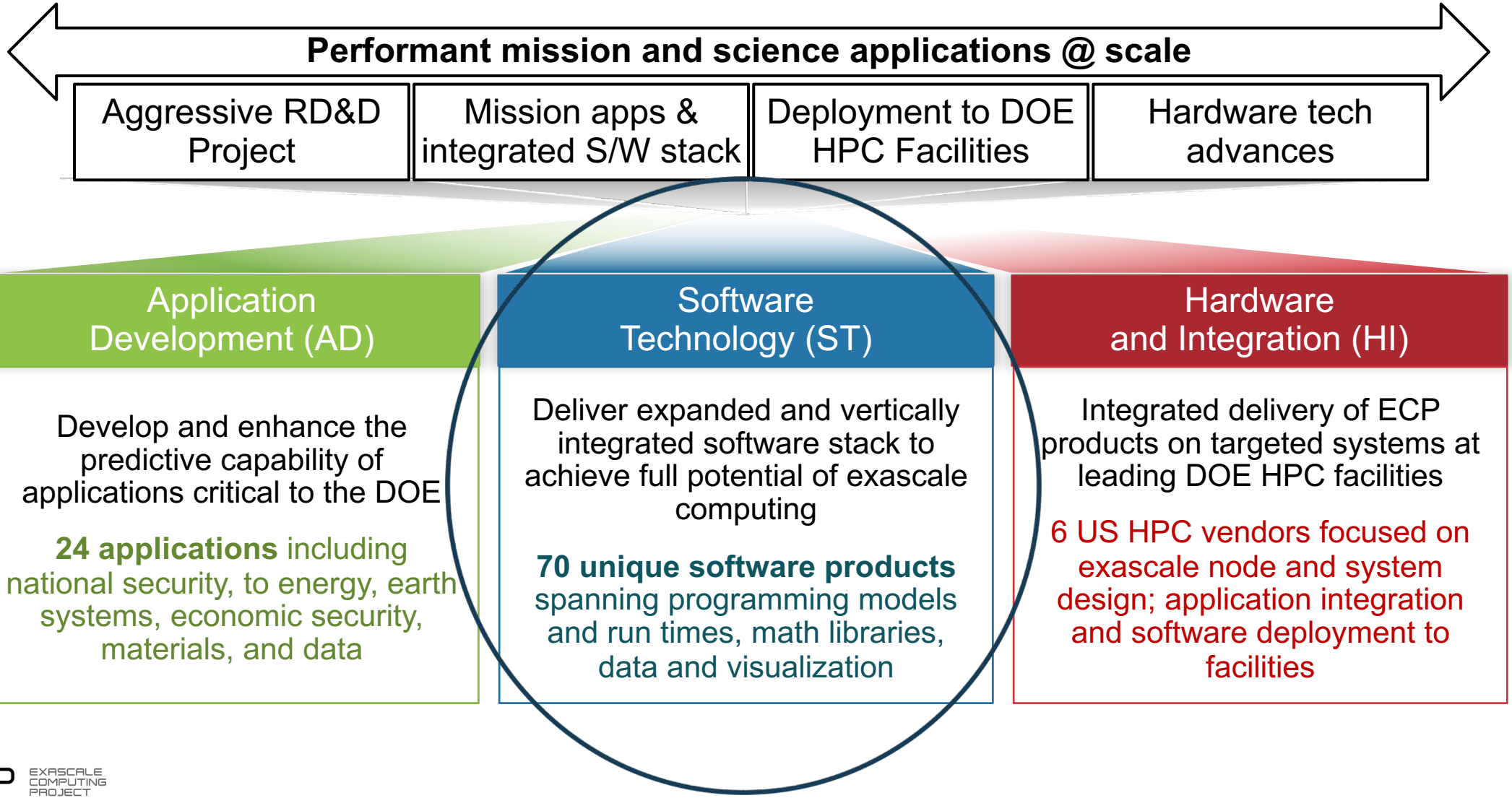
The 2nd R-CCS International Symposium
February 18, 2020

NNSA National Nuclear Security Administration

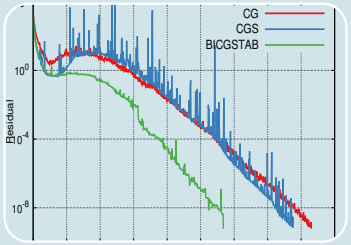
U.S. DEPARTMENT OF ENERGY | Office of Science

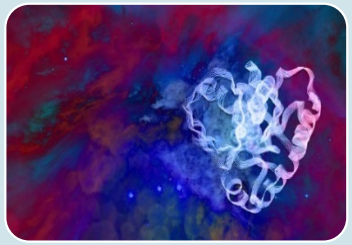
ECP Context for E4S

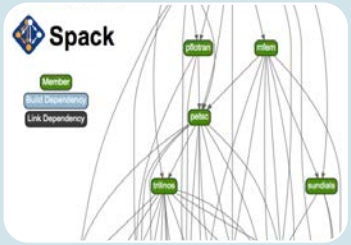
ECP Software Technology (ST) is one of three focus areas



ECP ST has six technical areas







Programming Models & Runtimes

- Enhance and get ready for exascale the widely used MPI and OpenMP programming models (hybrid programming models, deep memory copies)
- Development of performance portability tools (e.g. Kokkos and Raja)
- Support alternate models for potential benefits and risk mitigation: PGAS (UPC++/GASNet) ,task-based models (Legion, PaRSEC)
- Libraries for deep memory hierarchy and power management

Development Tools

- Continued, multifaceted capabilities in portable, open-source LLVM compiler ecosystem to support expected ECP architectures, including support for F18
- Performance analysis tools that accommodate new architectures, programming models, e.g., PAPI, Tau

Math Libraries

- Linear algebra, iterative linear solvers, direct linear solvers, integrators and nonlinear solvers, optimization, FFTs, etc
- Performance on new node architectures; extreme strong scalability
- Advanced algorithms for multi-physics, multiscale simulation and outer-loop analysis
- Increasing quality, interoperability, complementarity of math libraries

Data and Visualization

- I/O via the HDF5 API
- Insightful, memory-efficient in-situ visualization and analysis – Data reduction via scientific data compression
- Checkpoint restart

Software Ecosystem

- Develop features in Spack necessary to support all ST products in E4S, and the AD projects that adopt it
- Development of Spack stacks for reproducible turnkey deployment of large collections of software
- Optimization and interoperability of containers on HPC systems
- Regular E4S releases of the ST software stack and SDKs with regular integration of new ST products

NNSA ST

- Open source NNSA Software projects
- Projects that have both mission role and open science role
- Major technical areas: New programming abstractions, math libraries, data and viz libraries
- Cover most ST technology areas
- Subject to the same planning, reporting and review processes

ECP Software Technology Leadership Team

Mike Heroux, Software Technology Director
Mike has been involved in scientific software R&D for 30 years. His first 10 were at Cray in the LIBSCI and scalable apps groups. At Sandia he started the Trilinos and Mantevo projects, is author of the HPCG benchmark for TOP500, and leads productivity and sustainability efforts for DOE.

Lois Curfman McInnes, Software Technology Deputy Director
Lois is a senior computational scientist in the Mathematics and Computer Science Division of ANL. She has over 20 years of experience in high-performance numerical software, including development of PETSc and leadership of multi-institutional work toward sustainable scientific software ecosystems.

Rajeev Thakur, Programming Models and Runtimes (2.3.1)
Rajeev is a senior computer scientist at ANL and most recently led the ECP Software Technology focus area. His research interests are in parallel programming models, runtime systems, communication libraries, and scalable parallel I/O. He has been involved in the development of open source software for large-scale HPC systems for over 20 years.

Jeff Vetter, Development Tools (2.3.2)
Jeff is a computer scientist at ORNL, where he leads the Future Technologies Group. He has been involved in research and development of architectures and software for emerging technologies, such as heterogeneous computing and nonvolatile memory, for HPC for over 15 years.

Xaioye (Sherry) Li, Math Libraries (2.3.3)
Sherry is a senior scientist at Berkeley Lab. She has over 20 years of experience in high-performance numerical software, including development of SuperLU and related linear algebra algorithms and software.

Jim Ahrens, Data and Visualization (2.3.4)
Jim is a senior research scientist at the Los Alamos National Laboratory (LANL) and an expert in data science at scale. He started and actively contributes to many open-source data science packages including ParaView and Cinema.

Todd Munson, Software Ecosystem and Delivery (2.3.5)
Todd is a computational scientist in the Math and Computer Science Division of ANL. He has nearly 20 years of experience in high-performance numerical software, including development of PETSc/TAO and project management leadership in the ECP CODAR project.

Rob Neely, NNSA ST (2.3.6)
Rob is an Associate Division Leader in the Center for Applied Scientific Computing (CASC) at LLNL, chair of the Weapons Simulation & Computing Research Council, and lead for the Sierra Center of Excellence. His efforts span applications, CS research, platforms, and vendor interactions.

We work on products applications need now and into the future

Key themes:
- Exploration/development of new algorithms/software for emerging HPC capabilities:
- High-concurrency node architectures and advanced memory & storage technologies.
- Enabling access and use via standard APIs.

Software categories:
- The next generation of well-known and widely used HPC products (e.g., MPICH, OpenMPI, PETSc)
- Some lesser used but known products that address key new requirements (e.g., Kokkos, RAJA, Spack)
- New products that enable exploration of emerging HPC requirements (e.g., SICM, zfp, UnifyCR)

Example Products	Engagement
MPI – Backbone of HPC apps	Explore/develop MPICH and OpenMPI new features & standards.
OpenMP/OpenACC –On-node parallelism	Explore/develop new features and standards.
Performance Portability Libraries	Lightweight APIs for compile-time polymorphisms.
LLVM/Vendor compilers	Injecting HPC features, testing/feedback to vendors.
Perf Tools - PAPI, TAU, HPCToolkit	Explore/develop new features.
Math Libraries: BLAS, sparse solvers, etc.	Scalable algorithms and software, critical enabling technologies.
IO: HDF5, MPI-IO, ADIOS	Standard and next-gen IO, leveraging non-volatile storage.
Viz/Data Analysis	ParaView-related product development, node concurrency.

ECP ST Subprojects

- WBS
- Name
- PIs
- Project Managers (PMs)

ECP ST Stats

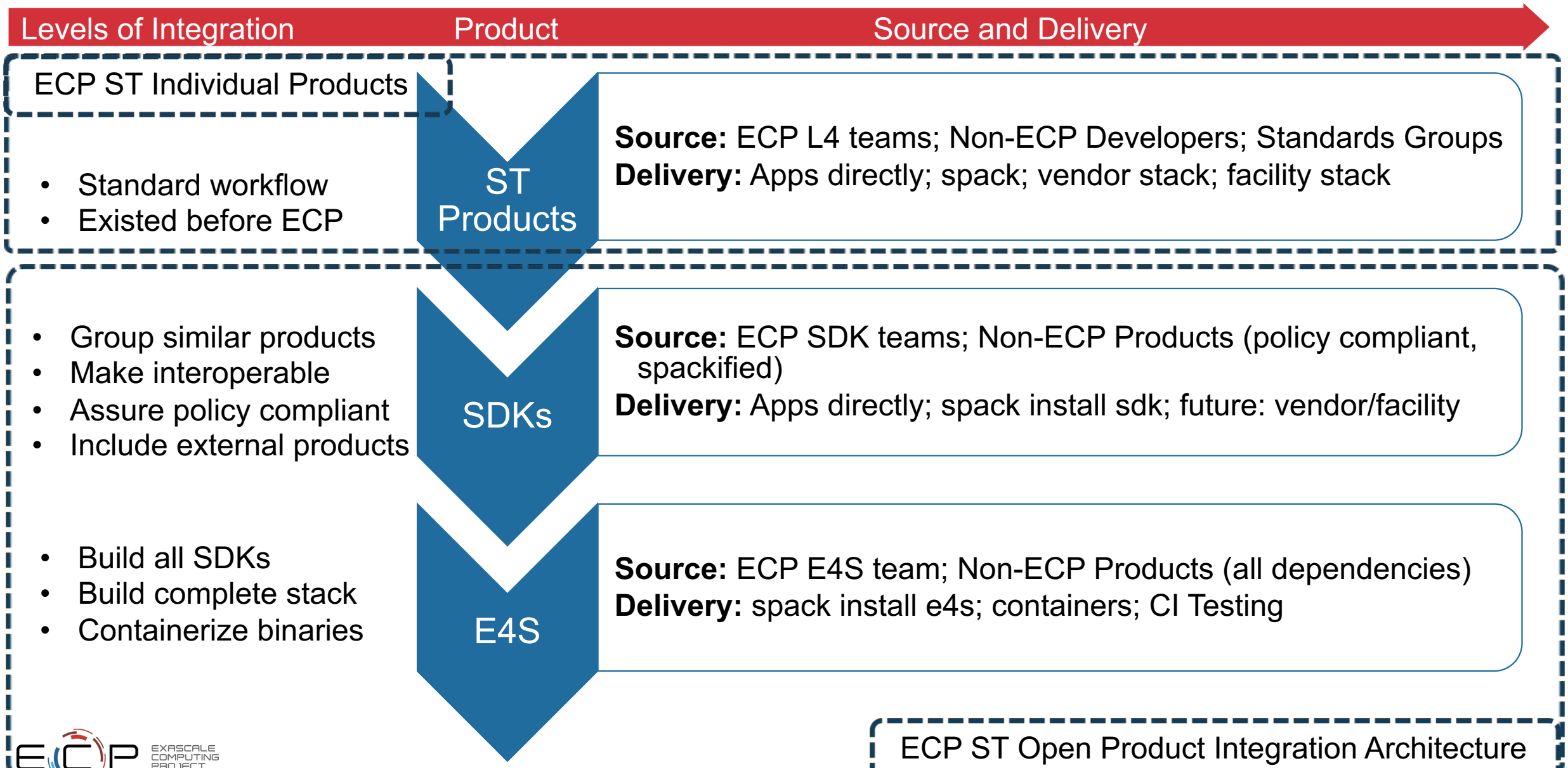
- 33 L4 subprojects
- 10 PI/PC same
- 23 PI/PC different

WBS	WBS Name	CAM/PI	PC
2.3	**Software Technology**	**Heroux, Mike, Carter, J.**	
2.3.1	**Programming Models & Runtimes**	**Thakur, Rajeev**	
2.3.1.01	PMR SDK	Shende, Sameer	Shende, Sameer
2.3.1.07	Exascale MPI (MPICH)	Balaji, Pavan	Guo, Yanfei
2.3.1.08	Legion	McCormick, Pat	McCormick, Pat
2.3.1.09	PaRSEC	Bosilica, George	Carr, Earl
2.3.1.14	Pagoda: UPC++/GASNet for Lightweight Communication and Global Address Space Support	Baden, Scott	Hargrove, Paul (and PI)
2.3.1.16	SICM	Lang, Michael	Vigil, Brittney
2.3.1.17	OMPI-X	Bernholdt, David	Grundhoffer, Alicia
2.3.1.18	RAJA/Kokkos	Trott, Christian Robert	Trujillo, Gabrielle
2.3.1.19	Argo: Low-level resource management for the OS and runtime	Beckman, Pete	Gupta, Rinku
2.3.2	**Development Tools**	**Vetter, Jeff**	
2.3.2.01	Development Tools Software Development Kit	Miller, Barton	Tim Haines
2.3.2.06	Exa-PAPI++: The Exascale Performance Application Programming Interface with Modern C++	Dongarra, Jack	Jagode, Heike
2.3.2.08	Extending HPCToolkit to Measure and Analyze Code Performance on Exascale Platforms	Mellor-Crummey, John	Mellor-Crummey, John
2.3.2.10	PROTEAS-TUNE	Vetter, Jeff	Glassbrook, Dick
2.3.2.11	SOLLVE: Scaling OpenMP with LLVm for Exascale	Chapman, Barbara	Kale, Vivek
2.3.2.12	FLANG	McCormick, Pat	Perry-Holby, Alexis
2.3.3	**Mathematical Libraries**	**McInnes, Lois**	
2.3.3.01	Extreme-scale Scientific xSDK for ECP	Yang, Ulrike	Yang, Ulrike
2.3.3.06	Preparing PETSc/TAO for Exascale	Smith, Barry	Munson, Todd
2.3.3.07	STRUMPACK/SuperLU/FFTX: sparse direct solvers, preconditioners, and FFT libraries	Li, Xiaoye	Li, Xiaoye
2.3.3.12	Enabling Time Integrators for Exascale Through SUNDIALS/ Hypre	Woodward, Carol	Woodward, Carol
2.3.3.13	CLOVER: Computational Libraries Optimized Via Exascale Research	Dongarra, Jack	Carr, Earl
2.3.3.14	ALExa: Accelerated Libraries for Exascale/ForTrilinos	Turner, John	Grundhoffer, Alicia
2.3.4	**Data and Visualization**	**Ahrens, James**	
2.3.4.01	Data and Visualization Software Development Kit	Atkins, Chuck	Bagha, Neelam
2.3.4.09	ADIOS Framework for Scientific Data on Exascale Systems	Klasky, Scott	Grundhoffer, Alicia
2.3.4.10	DataLib: Data Libraries and Services Enabling Exascale Science	Ross, Rob	Ross, Rob
2.3.4.13	ECP/VTK-m	Moreland, Kenneth	Moreland, Kenneth
2.3.4.14	VeloC: Very Low Overhead Transparent Multilevel Checkpoint/Restart/Sz	Cappello, Franck	Ehling, Scott
2.3.4.15	ExaIO - Delivering Efficient Parallel I/O on Exascale Computing Systems with HDF5 and Unify	Byna, Suren	Bagha, Neelam
2.3.4.16	ALPINE: Algorithms and Infrastructure for In Situ Visualization and Analysis/ZFP	Ahrens, James	Turton, Terry
2.3.5	**Software Ecosystem and Delivery**	**Munson, Todd**	
2.3.5.01	Software Ecosystem and Delivery Software Development Kit	Willenbring, James M	Willenbring, James M
2.3.5.09	SW Packaging Technologies	Gamblin, Todd	Gamblin, Todd
2.3.6	**NNSA ST**	**Neely, Rob**	
2.3.6.01	LANL ATDM	Mike Lang	Vandenbusch, Tanya Marie
2.3.6.02	LLNL ATDM	Becky Springmeyer	Gamblin, Todd
2.3.6.03	SNL ATDM	Jim Stewart	Trujillo, Gabrielle

E4S Overview

Delivering an open, hierarchical software ecosystem

Delivering an open, hierarchical software ecosystem



E4S: Extreme-scale Scientific Software Stack

- Curated release of ECP ST products based on Spack [http://spack.io] package manager
- Spack binary build caches for bare-metal installs
  - x86_64, ppc64le (IBM Power 9), and aarch64 (ARM64)
- Container images on DockerHub and E4S website of pre-built binaries of ECP ST products
- Base images and full featured containers (GPU support)
- GitHub recipes for creating custom images from base images
- GitLab integration for building E4S images
- E4S validation test suite on GitHub
- E4S VirtualBox image with support for container runtimes
  - Docker
  - Singularity
  - Shifter
  - Charliecloud
- AWS image to deploy E4S on EC2

https://e4s.io

Extreme-scale Scientific Software Stack (E4S)

- E4S: A Spack-based distribution of ECP ST and related and dependent software tested for interoperability and portability to multiple architectures
- Provides distinction between SDK usability / general quality / community and deployment / testing goals
- Will leverage and enhance SDK interoperability thrust

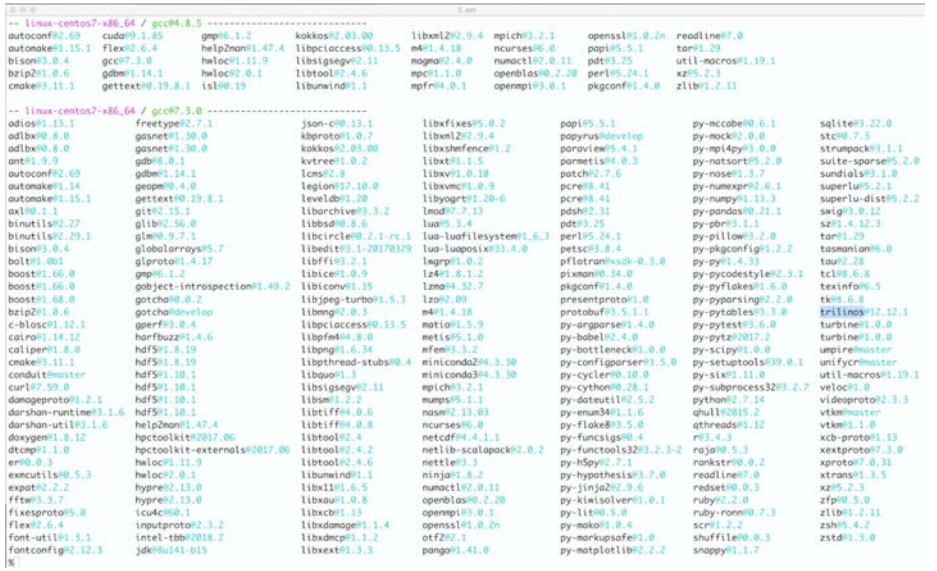
- Oct 2018: E4S 0.1 - 24 full, 24 partial release products
- Jan 2019: E4S 0.2 - 37 full, 10 partial release products
- Nov 2019: E4S 1.0 - 50 full, 5 partial release products
- Feb 2020: E4S 1.1 - 50 full,10 partial release products

e4s.io

Lead: Sameer Shende (U Oregon)

E4S 1.1 Full Release: 50 ECP Packages and all dependencies

- Adios
- AML
- Argobots
- Bolt
- Caliper
- Darshan
- Dyninst
- Faodel
- Flecsi
- Gasnet
- GEOPM
- Gotcha
- HDF5
- HPCToolkit
- Hypre
- Kokkos
- Legion
- Libnrm
- Libquo
- Magma
- Mercury
- MFEM
- MPICH
- MPIFileUtils
- Ninja
- OpenMPI
- PAPI
- Papyrus
- Parallel netCDF
- PDT
- PETSc
- Qthreads
- Raja
- Rempi
- SCR
- Spack
- Strumpack
- Sundials
- SuperLU
- SZ
- Tasmanian
- TAU
- Trilinos
- Turbine



Packages installed using Spack

- Umpire
- UnifyFS
- UPC++ Veloc
- Zfp

All ST products will be released through E4S

Partial (in-progress) Release: ASCENT, Catalyst, Flang, libEnsemble, LLVM, Visit, others

ECP SW Technology Software Architecture – Spack Software Distribution System

Spack enables Software distribution for HPC

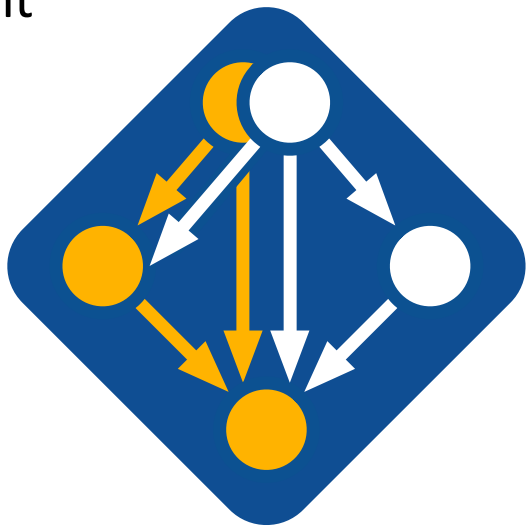
- Spack automates the build and installation of scientific software
- Packages are *templated,* so that users can easily tune for the host environment

No installation required: clone and go

```
$ git clone https://github.com/spack/spack
$ spack install hdf5
```

Simple syntax enables complex installs

```
$ spack install hdf5@1.10.5                    $ spack install hdf5@1.10.5 cppflags="-O3 -g3"
$ spack install hdf5@1.10.5 %clang@6.0         $ spack install hdf5@1.10.5 target=haswell
$ spack install hdf5@1.10.5 +threadssafe       $ spack install hdf5@1.10.5 +mpi ^mpich@3.2
```



github.com/spack/spack

- Ease of use of mainstream tools, with flexibility needed for HPC tuning

- Major victories:
  - ARES porting time on a new platform was reduced from **2 weeks to 3 hours**
  - Deployment time for 1,300-package stack on Summit supercomputer reduced from **2 weeks to a 12-hour overnight build**
  - Used by teams across ECP to **accelerate development**

Spack is being used on many of the top HPC systems

- Official deployment tool for the U.S. Exascale Computing Project
- 7 of the top 10 supercomputers
- High Energy Physics community
  - Fermilab, CERN, collaborators
- Astra (Sandia)
- Fugaku

Fugaku at RIKEN
DOE/MEXT collaboration

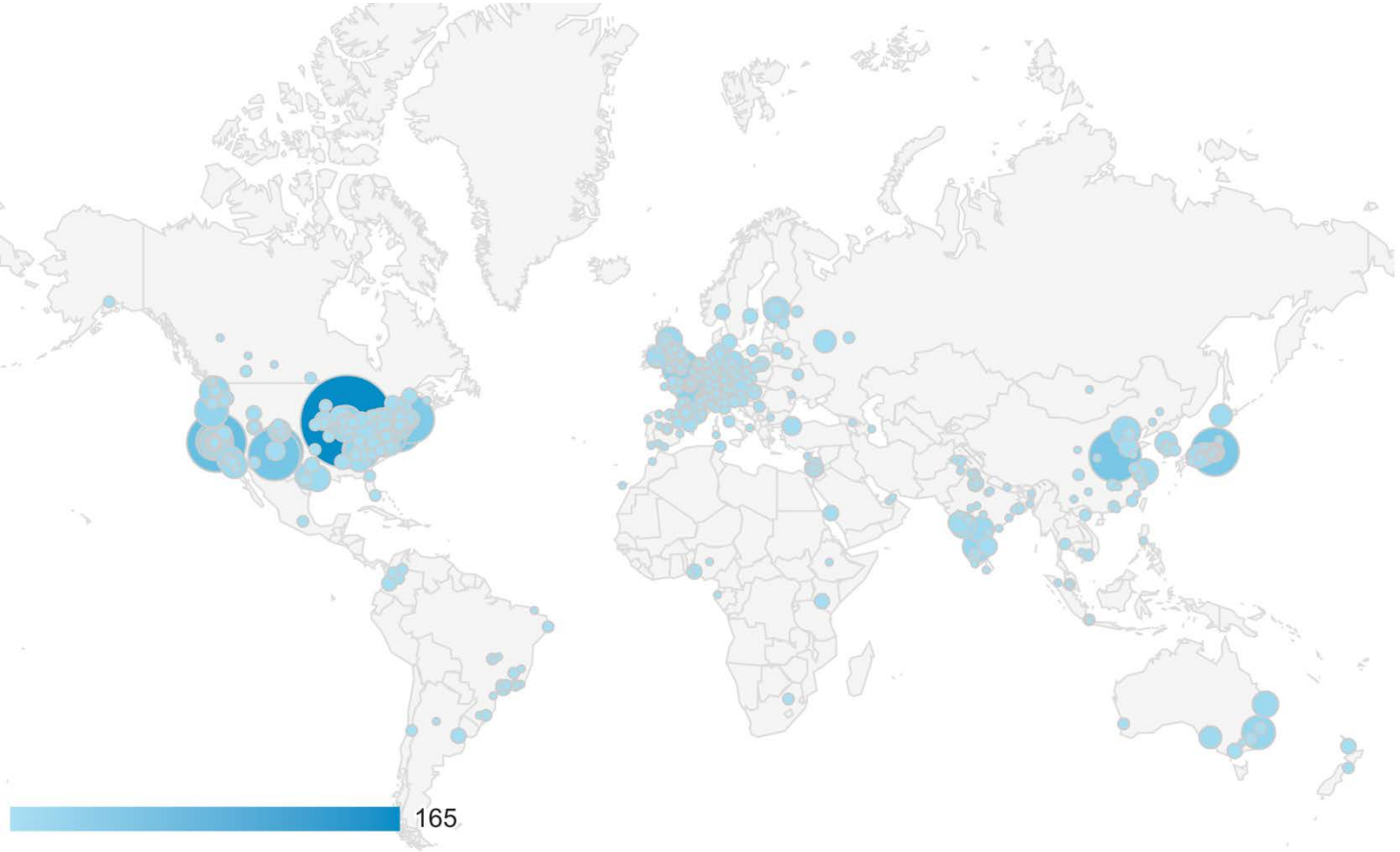
Summit (ORNL), Sierra (LLNL)

SuperMUC-NG (LRZ, Germany)

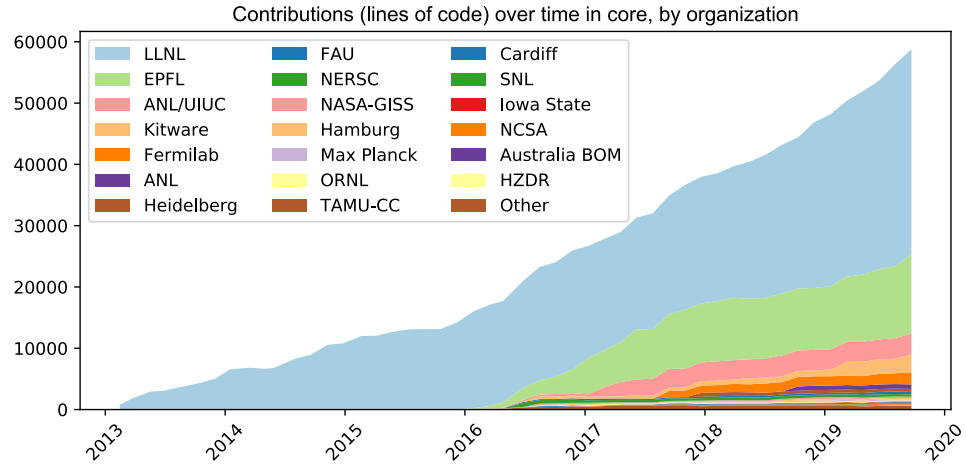
Edison, Cori, Perlmutter (NERSC)

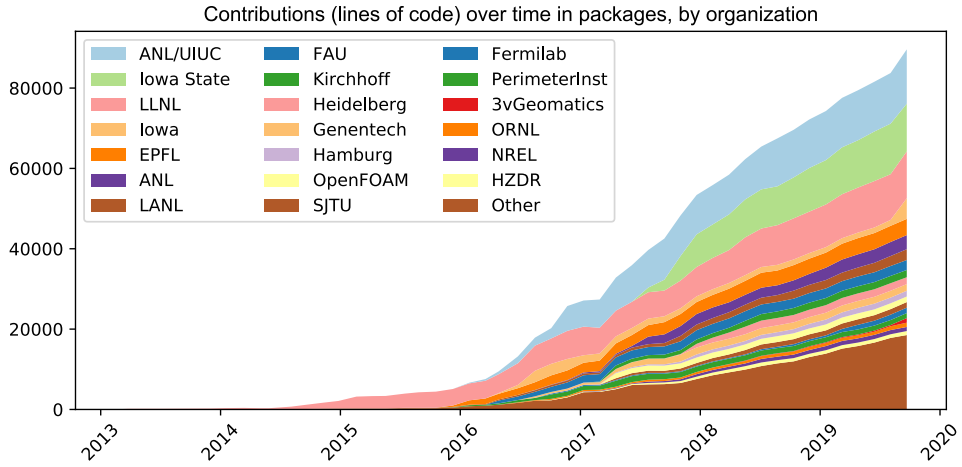
Spack is used worldwide!

Over **3,400** software packages
Over **2,000** monthly active users
Over **400** contributors (and growing)



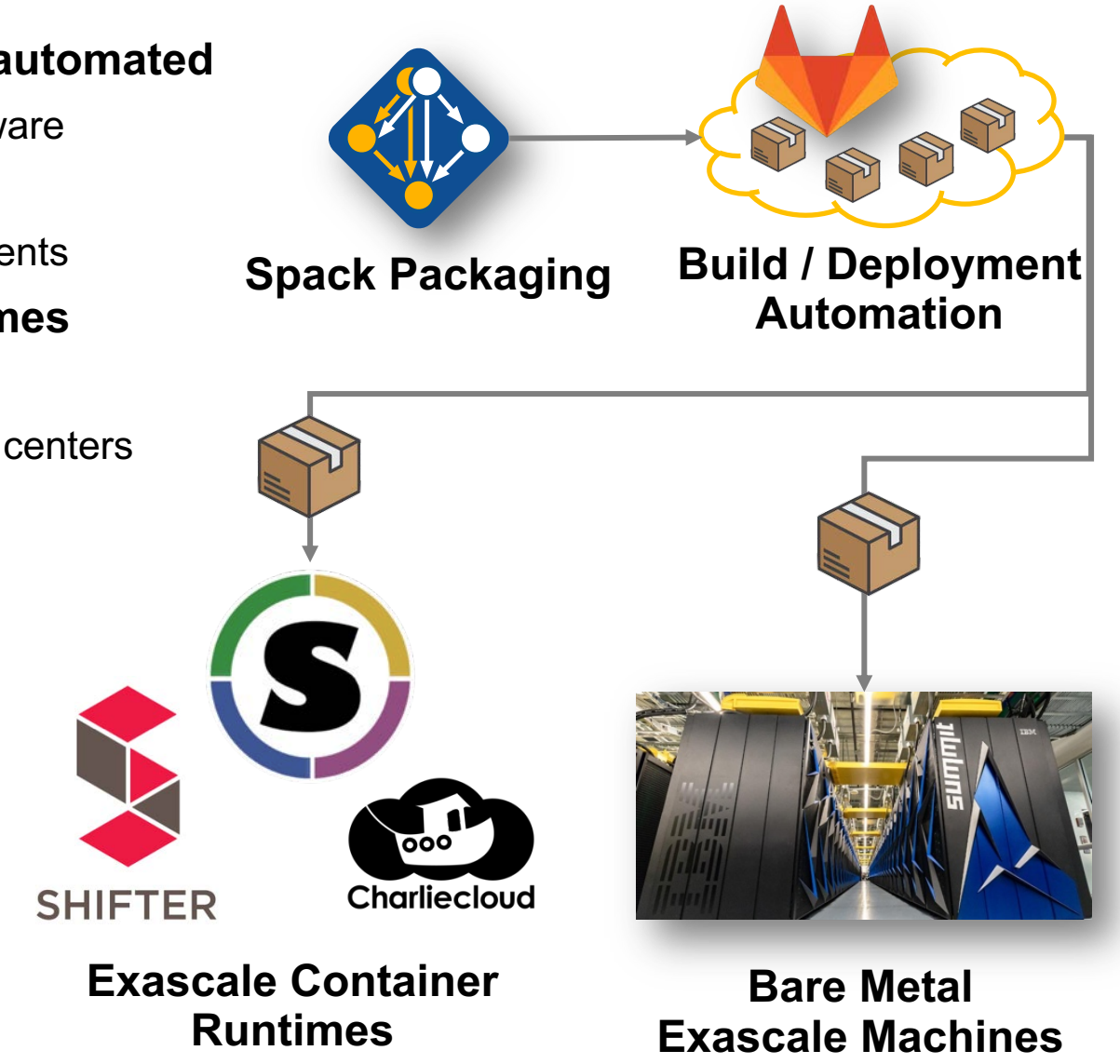
***Active users* of Spack documentation site for one month**
https://spack.readthedocs.io





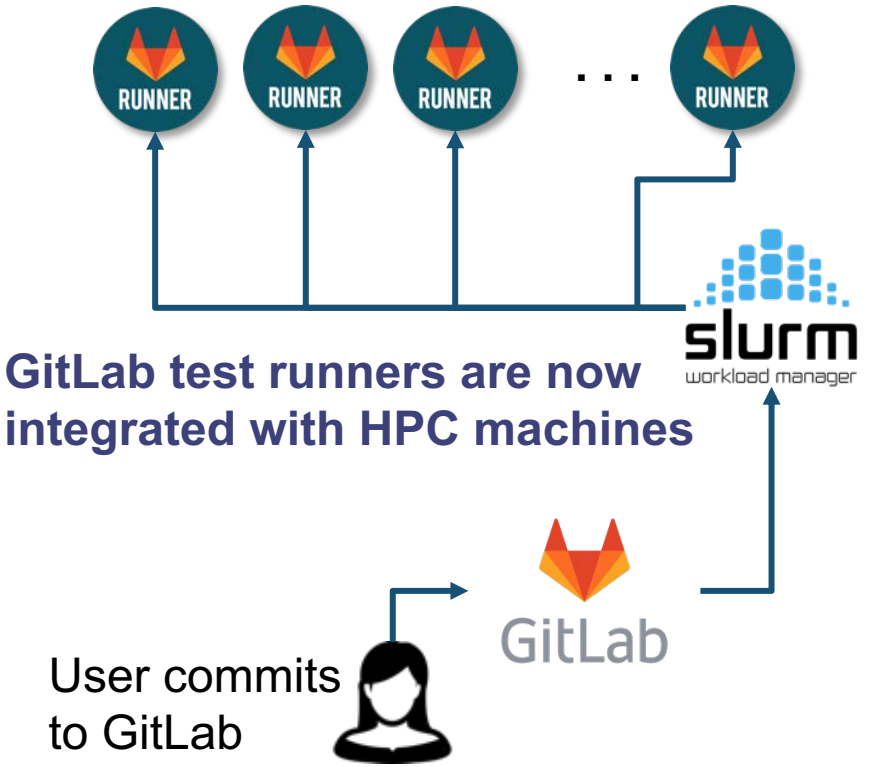
Spack strategy is to enable exascale software distribution on *both* bare metal and containers

1. **New capabilities to make HPC packaging easy and automated**
   - Optimized builds and package binaries that exploit the hardware
   - Workflow automation for facilities, developers, and users
   - Strong integration with containers as well as facility deployments
2. **Develop exascale enhancements to container runtimes**
   - Understand and automate performance/portability tradeoffs
   - Optimized containers & build automation for exascale HPC centers
   - Enable portability from laptops to exascale
3. **Outreach to users**
   - Ongoing working groups, Best practice guides
   - Tutorials, workshops, BOFs for Spack and Containers
4. **Collaboration across ECP**
   - Work with HI and other areas to build service infrastructure
   - Facilitate curation of packages through E4S and facilities
   - Ongoing ECP user support



Spack heavily involved in the ECP CI project.

- We have added security features to the open source GitLab product.
  - Integration with center identity management
  - Integration with schedulers like SLURM, LSF
- We are democratizing testing at Livermore Computing
  - Users can run tests across 30+ machines by editing a file
  - Previously, each team had to administer own servers
- ECP sites are deploying GitLab CI for users
  - All HPC centers can leverage these improvements
  - NNSA labs plan to deploy common high-side CI infrastructure
  - We are developing new security policies to allow external open source code to be tested safely on key machines



ECP SW Technology Software Architecture – SDKs

Software Development Kits (SDKs): Key delivery vehicle for ECP

A collection of related software products (packages) where coordination across package teams improves usability and practices, and foster community growth among teams that develop similar and complementary capabilities

- **Domain scope**
  Collection makes functional sense
- **Interaction model**
  How packages interact; compatible, complementary, interoperable
- **Community policies**
  Value statements; serve as criteria for membership
- **Meta-infrastructure**
  Invokes build of all packages (Spack), shared test suites
- **Coordinated plans**
  Inter-package planning. Augments autonomous package planning
- **Community outreach**
  Coordinated, combined tutorials, documentation, best practices

ECP ST SDKs: Grouping similar products for collaboration & usability

Programming Models & Runtimes Core

Tools & Technologies

Compilers & Support

Math Libraries (xSDK)

Viz Analysis and Reduction

Data mgmt., I/O Services & Checkpoint/ Restart

"Unity in essentials, otherwise diversity"

xSDK community policies

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.

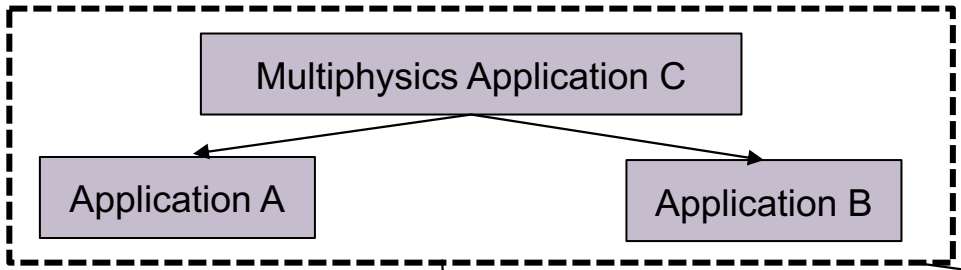
xSDK member package: Must be an xSDK-compatible package, *and* it uses or can be used by another package in the xSDK, and the connecting interface is regularly tested for regressions.

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

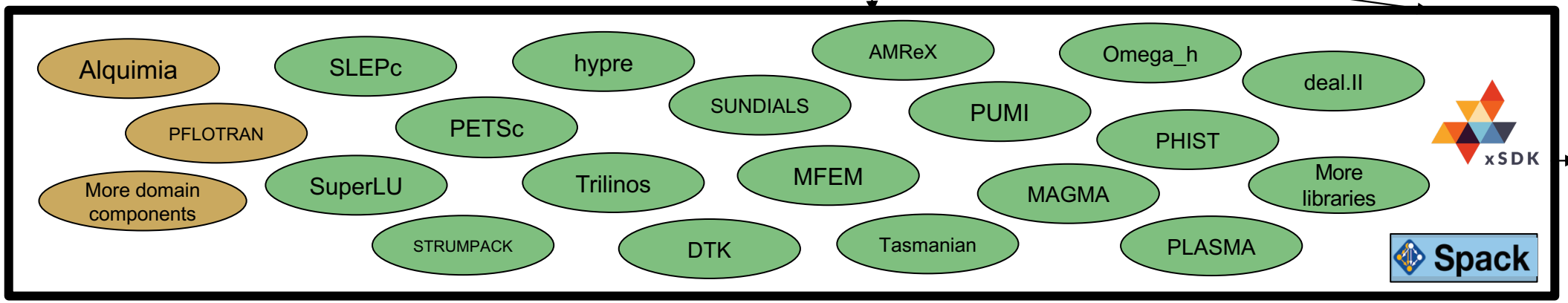
xSDK version 0.4.0: December 2018

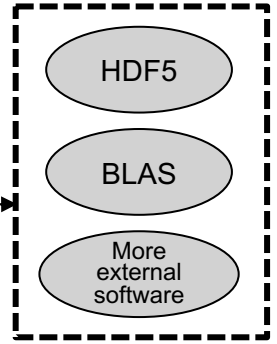
https://xsdk.info

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

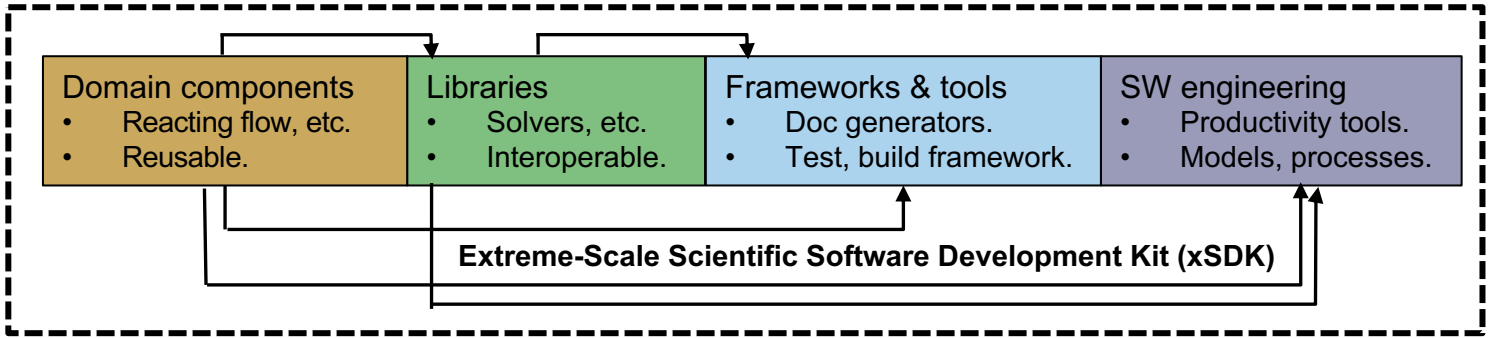


Tested on key machines at ALCF, NERSC, OLCF, also Linux, Mac OS X



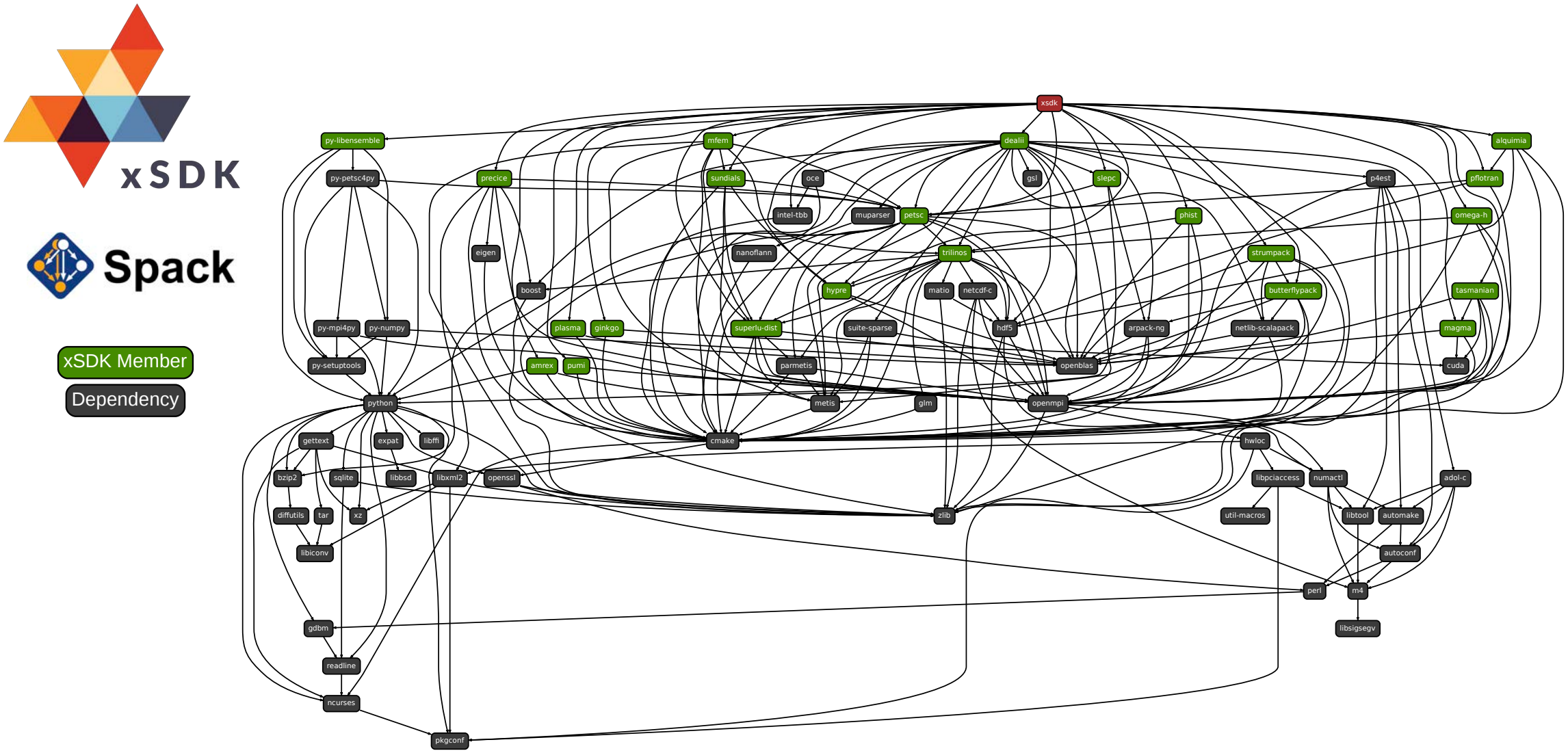


December 2018
- 17 math libraries
- 2 domain components
- 16 mandatory xSDK community policies
- Spack xSDK installer



Impact: Improved code quality, usability, access, sustainability

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


xSDK
Spack
xSDK Member
Dependency
xsdk
py-libensemble
mfem
dealii
alquimia
py-petsc4py
precice
sundials
oce
gsl
slepc
p4est
pflotran
intel-tbb
muparser
petsc
phist
omega-h
eigen
nanoflann
trilinos
strumpack
boost
hypre
matio
netcdf-c
butterflypack
tasmanian
py-mpi4py
py-numpy
plasma
ginkgo
superlu-dist
suite-sparse
hdf5
arpack-ng
netlib-scalapack
magma
py-setuptools
amrex
pumi
parmetis
openblas
cuda
python
metis
glm
openmpi
cmake
gettext
expat
libffi
hwloc
bzip2
sqlite
libbsd
libxml2
openssl
libpciaccess
numactl
adol-c
diffutils
tar
xz
zlib
util-macros
libtool
automake
libiconv
autoconf
perl
m4
gdbm
libsigsegv
readline
ncurses
pkgconf

xSDK version 0.5: November 2019
(21 math libs, 2 domain-specific packages)

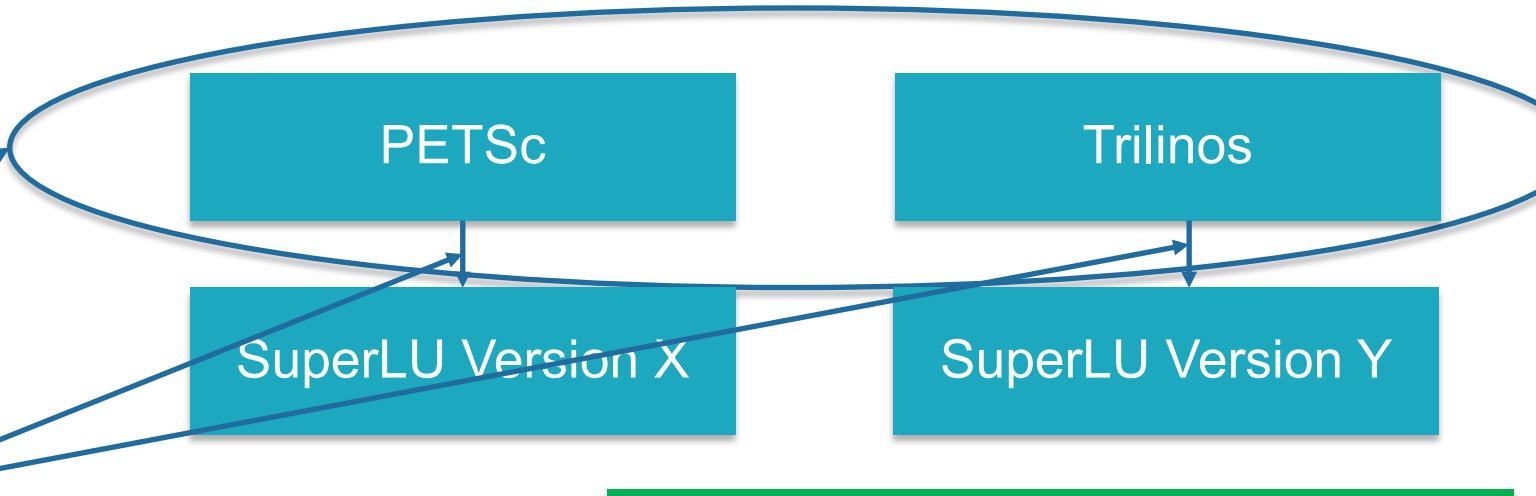
- AMReX
- ButterflyPACK
- DTK
- deal.ii
- Ginkgo
- hypre
- libEnsemble
- MAGMA
- MFEM
- Omega_h
- PETSc/TAO
- PHIST
- PLASMA
- preCICE
- PUMI
- SLEPc
- STRUMPACK
- SUNDIALS
- SuperLU
- Tasmanian
- Trilinos
- Pflotran
- Alquimia

Notes:

- **Growth:**
  - **5 in release 0.1.**
  - **7 in 0.2**
  - **9 in 0.3**
  - **19 in 0.4**
  - **23 in 0.5**

- You do not need to build all packages.
- We build and test all packages.
- Any subset is guaranteed to build if using the same build parameters, platforms.
- Similar builds should work or require less effort for success.

SDK “Horizontal” Grouping: Key Quality Improvement Driver



Horizonal (vs Vertical) Coupling

- Common substrate
- Similar function and purpose
  - e.g., compiler frameworks, math libraries
- Potential benefit from common Community Policies
  - Best practices in software design and development and customer support
- Used together, but not in the long vertical dependency chain sense
- Support for (and design of) common interfaces
  - Commonly an aspiration, not yet reality

Horizontal grouping:
- Assures X=Y.
- Protects against regressions.
- Transforms code coupling from heroic effort to turnkey.

Development of xSDK Community Policies

- Community policies now available on Github for revision control and to preserve history: https://github.com/xsdk-project/xsdk-community-policies
- Established process on how to change or add policies
- Newest release: 0.5.0

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

https://xsdk.info/policies

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. Have README, SUPPORT, LICENSE, and CHANGELOG file in top directory.

Changes in 0.5.0:

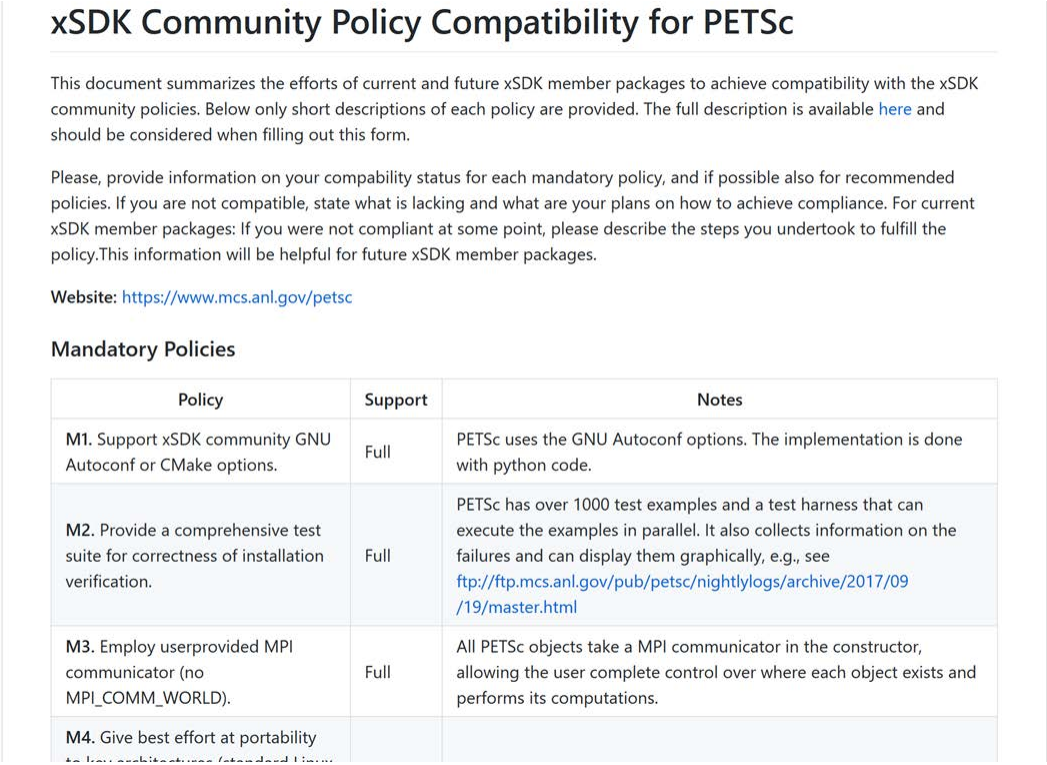
- **New recommended policy R7**
- **Dropped the requirement to detect MPI 2 features in M3**
- **Made various editorial changes in M5, M13, M15, and R2 for clarification or to fix typos.**

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.

Integration of Community Policies

- Potential new xSDK members will fill out compatibility form: https://github.com/xsdk-project/xsdk-policy-compatibility
- xSDK team will check for compatibility and approve if compatible
- Designing and implementing xSDK policy checker to automate the policy compliance checking.
  - Implemented checkers for a few policy rules. Examples:
    - M3: Employ user-provided MPI Communicator: Policy checker lists the file names that contain MPI_COMM_WORLD.
    - M7: Come with an open source license. Policy checker search license files at the top directly of the source and scan the headers of source code files.



xSDK Community Policy Compatibility for PETSc

This document summarizes the efforts of current and future xSDK member packages to achieve compatibility with the xSDK community policies. Below only short descriptions of each policy are provided. The full description is available here and should be considered when filling out this form.

Please, provide information on your compability status for each mandatory policy, and if possible also for recommended policies. If you are not compatible, state what is lacking and what are your plans on how to achieve compliance. For current xSDK member packages: If you were not compliant at some point, please describe the steps you undertook to fulfill the policy.This information will be helpful for future xSDK member packages.

Website: https://www.mcs.anl.gov/petsc

Mandatory Policies

Policy	Support	Notes
M1. Support xSDK community GNU Autoconf or CMake options.	Full	PETSc uses the GNU Autoconf options. The implementation is done with python code.
M2. Provide a comprehensive test suite for correctness of installation verification.	Full	PETSc has over 1000 test examples and a test harness that can execute the examples in parallel. It also collects information on the failures and can display them graphically, e.g., see ftp://ftp.mcs.anl.gov/pub/petsc/nightlylogs/archive/2017/09/19/master.html
M3. Employ userprovided MPI communicator (no MPI_COMM_WORLD).	Full	All PETSc objects take a MPI communicator in the constructor, allowing the user complete control over where each object exists and performs its computations.
M4. Give best effort at portability		

Processes for xSDK release and delivery

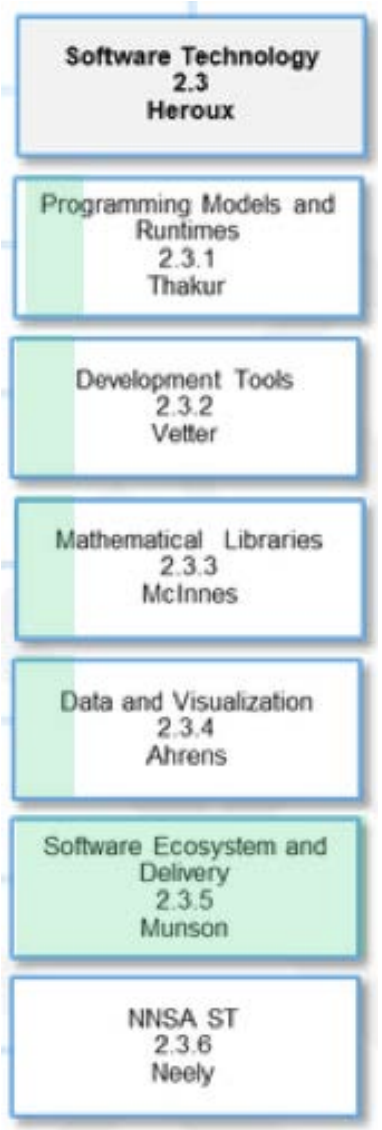
- **2-level release process**
  - **xSDK**
    - Ensure and test compatibility of mostly independent package releases
  - **xSDK member packages**
    - Achieve compatibility with xSDK community policies prior to release
      - https://github.com/xsdk-project/xsdk-policy-compatibility
    - Have a Spack package
    - Port to target platforms
    - Provide user support
- **Obtaining the latest release: https://xsdk.info/releases**
- **Draft xSDK package release process checklist:**
  - https://docs.google.com/document/d/16y2bL1RZg8wke0vY8c97ssvhRYNez34Q4QGg4LolEUk/edit?usp=sharing

xSDK delivery process

- Regular releases of software and documentation, primarily through member package release processes
- Anytime open access to production software from GitHub, BitBucket and related community platforms

There are 5 L4 projects to define and/or enhance SDKs

- Each L3 area has an L4 project devoted to SDK definition and coordination across the portfolio
- Software ecosystem L4 project focuses on packaging
  - Spack
  - Containers
- Strong coordination with HI Software Deployment projects
- Drives milestone-based planning

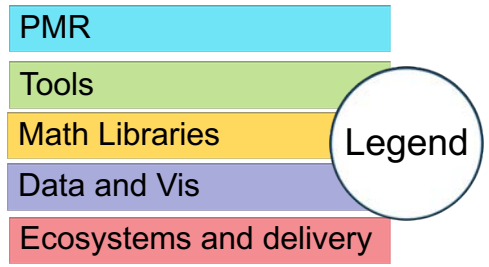


ECP ST SDKs will span all technology areas

Motivation: Properly chosen cross-team interactions will build relationships that support interoperability, usability, sustainability, quality, and productivity within ECP ST.

Action Plan: Identify product groupings where coordination across development teams will improve usability and practices, and foster community growth among teams that develop similar and complementary capabilities.

PMR Core (17)	Compilers and Support (7)	Tools and Technology (11)	xSDK (16)	Visualization Analysis and Reduction (9)	Data mgmt, I/O Services, Checkpoint restart (12)	Ecosystem/E4S at-large (12)
QUO	openarc	TAU	hypre	ParaView	SCR	mpiFileUtils
Papyrus	Kitsune	HPCToolkit	FleSCI	Catalyst	FAODEL	TriBITS
SICM	LLVM	Dyninst Binary Tools	MFEM	VTK-m	ROMIO	MarFS
Legion	CHiLL autotuning comp	Gotcha	Kokkoskernels	SZ	Mercury (Mochi suite)	GUFI
Kokkos (support)	LLVM openMP comp	Caliper	Trilinos	zfp	HDF5	Intel GEOPM
RAJA	OpenMP V & V	PAPI	SUNDIALS	VisIt	Parallel netCDF	BEE
CHAI	Flang/LLVM Fortran comp	Program Database Toolkit	PETSc/TAO	ASCENT	ADIOS	FSEFI
PaRSEC*		Search (random forests)	libEnsemble	Cinema	Darshan	Kitten Lightweight Kernel
DARMA		Siboka	STRUMPACK	ROVER	UnifyCR	COOLR
GASNet-EX		C2C	SuperLU		VeloC	NRM
Qthreads		Sonar	ForTrilinos		IOSS	ArgoContainers
BOLT			SLATE		HXHIM	Spack
UPC++			MAGMA			
MPICH			DTK			
Open MPI			Tasmanian			
Umpire			TuckerMPI			
AML						



Data SDK: HDF5 API Initiative

- Adding The HDF Group (THG) to the Data & Vis SDK project to assist in various aspects of improving the use of the HDF5 API
  - Support and training
    - Direct support and training will be provided to ECP applications and ST projects to assist in the performant use of HDF5
  - Compatible APIs with other I/O libraries
    - DataLib (pnetcdf) and ADIOS will develop interfaces compatible with the HDF5 API, possibly through the HDF5 Virtual Object Layer
    - THG will provide support to the I/O projects to assist in achieving this compatibility

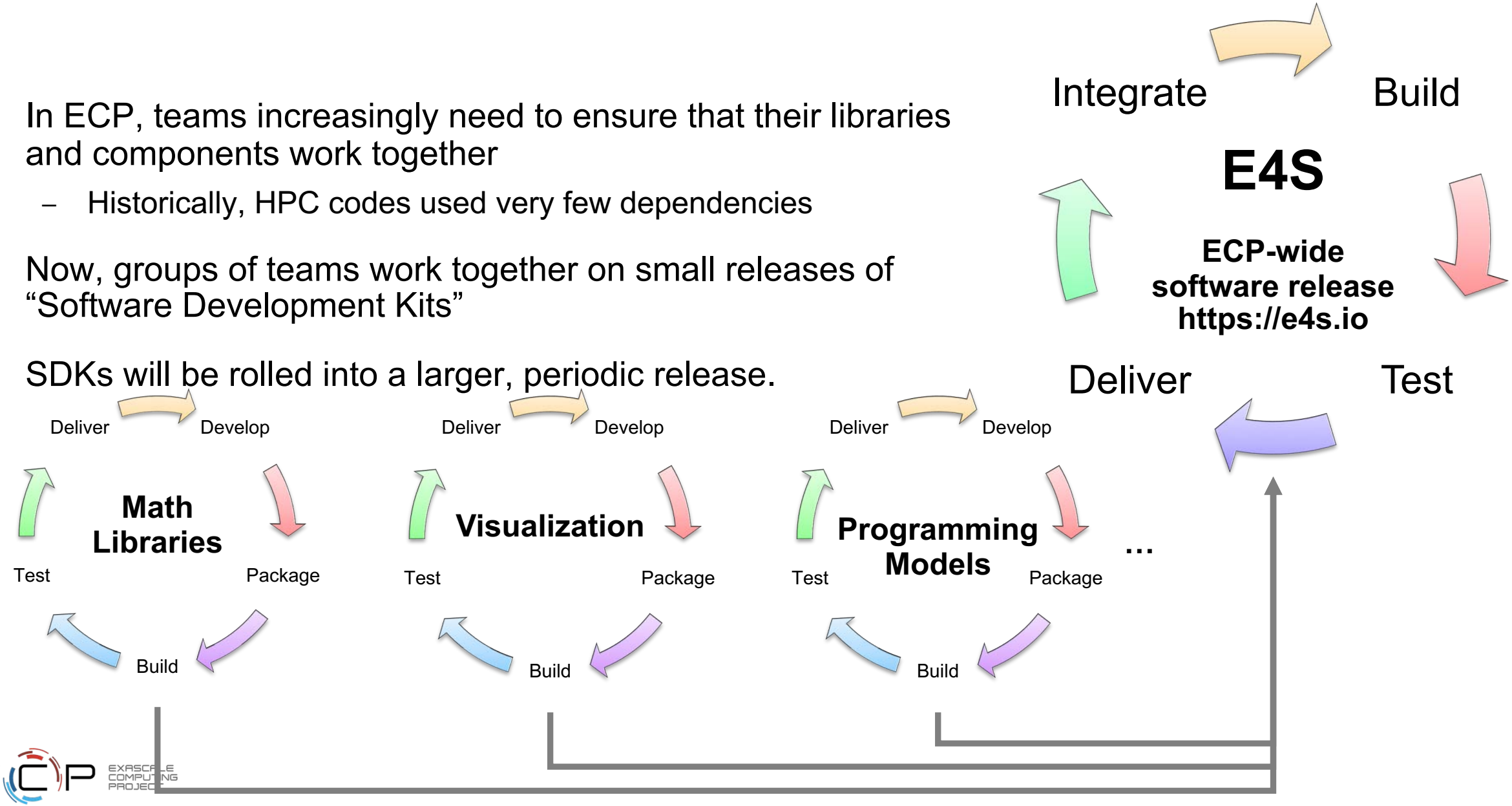
- Both Kitware and THG will identify the I/O usage patterns w.r.t. metadata operations, bulk data operations, access patterns.
  - This will help identify usage patterns that can be improved on by applications and ST projects
  - Also will help identify usage patterns that HDF5 can work to optimize for.

SDK Summary

- SDKs will help reduce complexity of delivery:
  - Hierarchical build targets.
  - Distribution of software integration responsibilities.
- New Effort: Started in April 2018, fully established in August 2018.
- Extending the SDK approach to all ECP ST domains.
  - SDKs create a horizontal coupling of software products, teams.
  - Create opportunities for better, faster, cheaper – pick all three.
- First concrete effort: Spack target to build all packages in an SDK.
  - Decide on good groupings.
  - Not necessarily trivial: Version compatibility issues, Coordination of common dependencies.
- Longer term:
  - Establish community policies, enhance best practices sharing.
  - Provide a mechanism for shared infrastructure, testing, training, etc.
  - Enable community expansion beyond ECP.

Putting it all together: Products + SDKs + E4S

- In ECP, teams increasingly need to ensure that their libraries and components work together
  - Historically, HPC codes used very few dependencies
- Now, groups of teams work together on small releases of "Software Development Kits"
- SDKs will be rolled into a larger, periodic release.



E4S Collaborations

NSF Collaborations

1st Workshop on NSF and DOE High Performance Computing Tools

Dates: July 10-11, 2019
Location: 312 Lillis, University of Oregon, Eugene, OR 97403

Workshop Agenda

Wednesday, July 10, 2019: 312 Lillis, University of Oregon

8:00am - 8:45am:	Registration
8:45am - 9:15am:	Welcome, introductions, David Conover (VPRI, UO), Vipin Chaudhary (Program Director, CISE/OAC, NSF), Jonathan Carter (Deputy Director, Software Technology, ECP, and Deputy for Science, LBL)[slides] [video]
9:15am - 10:30am:	Spack tutorial - Todd Gamblin and Greg Becker (LLNL) [slides] [video]
10:30am - 11:00am:	Break
11:00am - 12:30pm:	Spack tutorial - Todd Gamblin (LLNL) [video]
12:30pm - 2:00pm:	Lunch break: 440 Lillis
2:00pm - 3:30pm:	Spack tutorial - Todd Gamblin and Greg Becker (LLNL) [video]
3:30 pm - 4:00pm:	Break
4:00pm - 5:30pm:	Spack tutorial - Todd Gamblin and Greg Becker (LLNL) [video 1] [video 2] [video 3] [video 4] [video 5]
6:30pm - 9:30pm:	Working dinner at the Jordan Schnitzer Museum of Art, UO

https://oaciss.uoregon.edu/NSFDOE19/agenda.html

Thursday, July 11, 2019: 312 Lillis, University of Oregon

9:00am - 9:30am:	Unifying Software Distribution in ECP - Todd Gamblin (LLNL) [slides] [video]
9:00am - 9:30am:	Containers for HPC - Andrew Younge (Sandia National Laboratories, NM) [slides] [video]
9:30am - 10:00am:	Software deployment at DOE facilities - David Montoya (Los Alamos National Laboratory, NM) [slides] [video]
10:00am - 10:30am:	E4S - Sameer Shende (University of Oregon) [slides] [video]
10:30am - 11:00am:	Break
11:00am - 11:30am:	Overview of software architecture - Todd Gamblin, LLNL
11:30am - 12:30pm:	Hands-on, applying Spack to applications.
12:30pm - 2:00pm:	Lunch: 440 Lillis
2:00pm - 3:30pm:	Hands-on, applying Spack to applications.
3:30pm - 4:00pm:	Break
4:00pm - 5:00pm:	Hands-on, Spack and E4S.
5:00pm - 6:15pm:	Closing remarks, planning for the next workshop - Jonathan Carter (Lawrence Berkeley National Laboratory)
6:30pm:	Dinner at Excelsior Inn

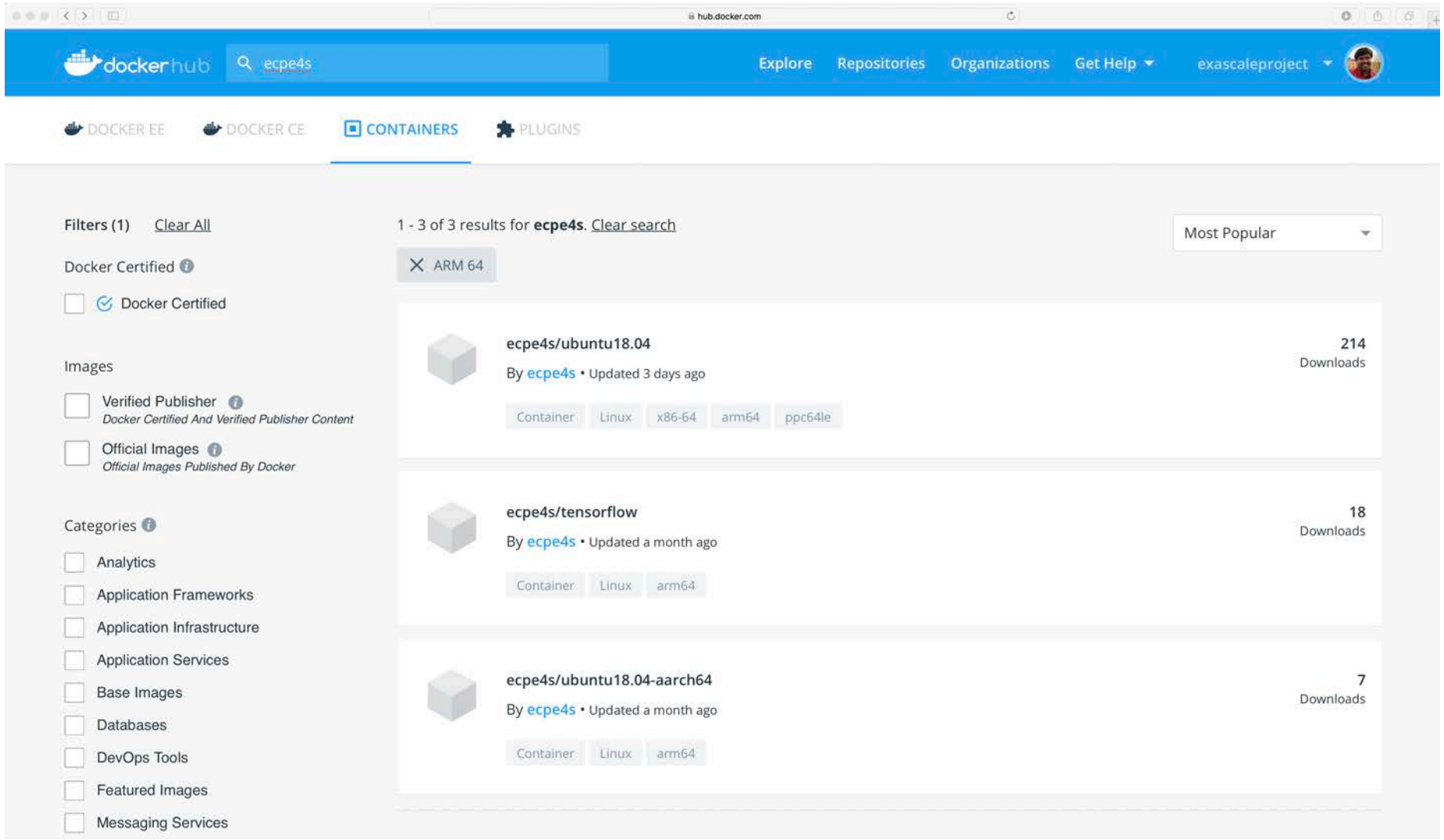
Extending Collaborations

- E4S/SDK collaborations make sense with:
  - HPC open source development projects:
    - deal.II (NSF math library),
    - PHIST (DLR Germany library).
  - Commercial open source packagers/distributors:
    - OpenHPC.
    - HPC systems vendors.
  - HPC systems facilities:
    - SDSC, TACC, others.
  - Other organizations in search of collaborative open source software foundation:
    - NSF science teams.
    - NOAA, others.
    - International collaborators.

E4S: Building on top of previous efforts

- E4S did not emerge from nothing.
- Leveraging the work of many others.
- HPC Linux: Work done at U of Oregon, and at ParaTools.
- IDEAS-Classic: xSDK – the original SDK continuing under ECP.
- Spack – Pre-dates E4S.

E4S: ARM64 Base Container Images



- Dockerhub
- Support for NVDIA GPUs

E4S Wrap Up

Delivering an open, hierarchical software ecosystem

Levels of Integration | Product | Source and Delivery

ECP ST Individual Products

ST Products

- Standard workflow
- Existed before ECP

Source: ECP L4 teams; Non-ECP Developers; Standards Groups
Delivery: Apps directly; spack; vendor stack; facility stack

SDKs

- Group similar products
- Make interoperable
- Assure policy compliant
- Include testing

Source: ECP SDK teams; Non-ECP Products (policy compliant, spackified)
Delivery: Apps directly; spack install sdk; future: vendor/facility

E4S

- Build all SDKs
- Build complete stack
- Containerize binaries

Source: ECP E4S team; Non-ECP Products (all dependencies)
Delivery: spack install e4s; containers; CI Testing

- **ECP Legacy: An open, hierarchical and extensible HPC software ecosystem**
  - SDK communities for enhanced developer and user experience
  - Easy deployment via source and by container technologies
  - In collaboration with DOE facilities, vendors, other US agencies and strategic collaborators
  - A model for post-ECP coordinated software development and delivery.

ECP ST Open Product Integration Architecture

E4S Summary

What E4S is not	What E4S is
• A closed system taking contributions only from DOE software development teams.	• Extensible, open architecture software ecosystem accepting contributions from US and international teams. • Framework for collaborative open-source product integration.
• A monolithic, take-it-or-leave-it software behemoth.	• A full collection if compatible software capabilities **and** • A manifest of a la carte selectable software capabilities.
• A commercial product.	• Vehicle for delivering high-quality reusable software products in collaboration with others.
• A simple packaging of existing software.	• The conduit for future leading edge HPC software targeting scalable next-generation computing platforms. • A hierarchical software framework to enhance (via SDKs) software interoperability and quality expectations.

Final Remarks

- Software Development Kits (SDKs) provide a new software coordination mechanism:
  - Do not significantly increase the number of software products in the HPC ecosystem.
  - Provide intermediate build, install and test targets for E4S (reducing complexity).
  - Enable development teams to improve and standardize look-and-feel, best practices, policies.
  - Improve interoperability, interchangeability of similar products.
  - Fosters communities of developers in a cooperative/competitive environment.
  - **Provides integration point for SDK-compatible, non-ECP products.**
- The Extreme-scale Scientific Software Stack (E4S) provides a complete HPC software stack:
  - Does not significantly increase the number of software products in the HPC ecosystem.
  - Provides a coordinated approach to building, installing and testing HPC software.
  - Tests some products with a subset of configurations on a subset of platforms.
  - Improves stability of the ST stack so that any subset is more stable.
  - **Provides a complete software stack, including non-ECP products.**
- **The SDK/E4S architecture is open:**
  - Enables light-weight coordinated collaboration among open source software teams.
  - ECP seeks collaboration: libraries/tools and SDKs, facilities and E4S.

ECP Software Technology Capability Assessment Report (CAR) Version 2.0

- Comprehensive document about ECP ST structure, progress and planning.
- Version 2.0 – extensive revision:
  - FY20 – 23 Organization.
  - Planning, Execution, Tracking & Assessment processes.
  - E4S/SDK details.
  - 2-page writeups for each product development effort.
  - Released February 1, 2020.

https://exascaleproject.org/wp-content/uploads/2020/02/ECP-ST-CAR-V20-1.pdf