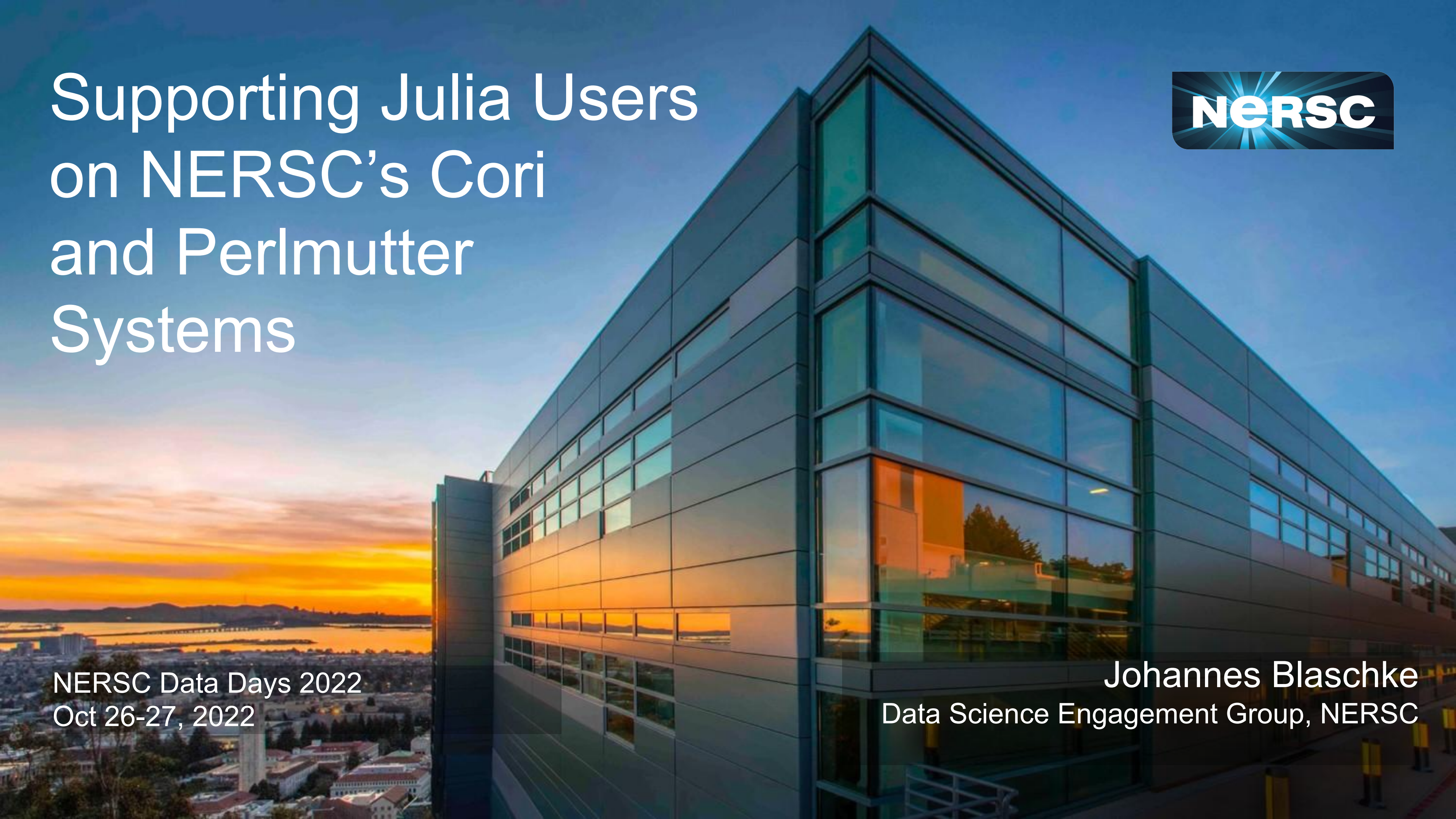


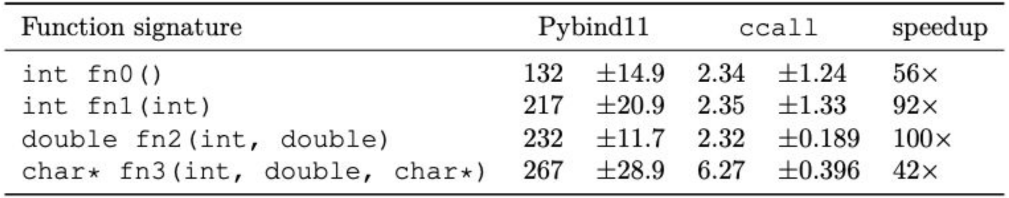
Supporting Julia Users on NERSC’s Cori and Perlmutter Systems

NERSC Data Days 2022
Oct 26-27, 2022

Johannes Blaschke
Data Science Engagement Group, NERSC

The Need for High-Performance Glue Code

- **Objective**: Establish High-Productivity High-Performance Programming Languages
- Common Design Pattern: High-Productivity Language (eg. Python) as Glue Code
    - At NERSC: Julia, Python + C/C++/CUDA
    - Pro: Use appropriate language for algorithms requiring high performance
    - Con: N+1-language problem (code maintainability)
    - Con: Context switching between interpreted and compiled languages



Function signature	Pybind11		ccall		speedup
`int fn0()`	132	±14.9	2.34	±1.24	56×
`int fn1(int)`	217	±20.9	2.35	±1.33	92×
`double fn2(int, double)`	232	±11.7	2.32	±0.189	100×
`char* fn3(int, double, char*)`	267	±28.9	6.27	±0.396	42×

Julia Usage Trends at NERSC

- Growing interest in Julia at NERSC:

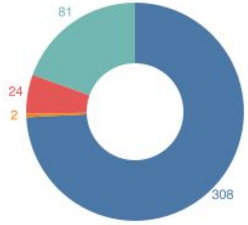
HPCwire

Julia Joins Petaflop Club

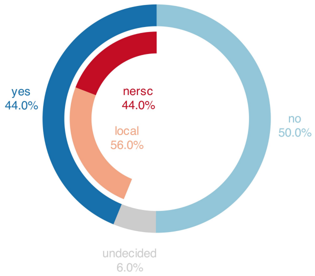
September 12, 2017

BERKELEY, Calif., Sept. 12, 2017 — ,

Do you use Julia locally or at NERSC?	Responses	%
"I do not use Julia (locally or at NERSC)"	308	74
"I use Julia locally but not at NERSC"	81	20
"I use Julia locally and at NERSC"	24	6.8
"I use Julia at NERSC but not locally"	2	0.5

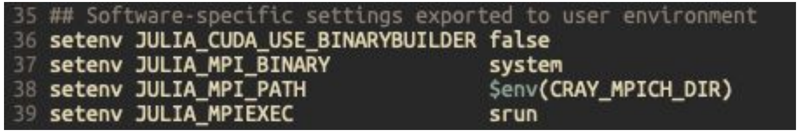


Do you plan to use Julia in future?



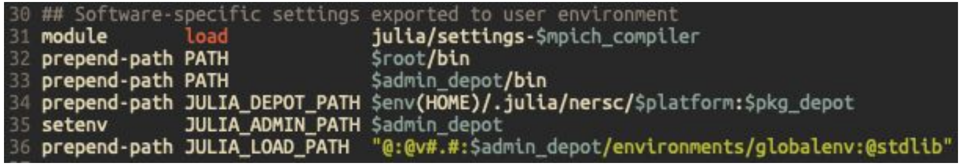
Julia Support at NERSC

- **Objective**: Enable users to “roll their own” Julia install / environment
- Support different “levels” of Julia users:
  a. Provide documentation and use cases
  b. Provide system-wide settings using `Preferences.jl`: user can load a module, and all packages that need vendor libs (MPI, HPF5, etc) gets correctly compiled)



```
## Software-specific settings exported to user environment
setenv JULIA_CUDA_USE_BINARYBUILDER false
setenv JULIA_MPI_BINARY             system
setenv JULIA_MPI_PATH               $env(CRAY_MPICH_DIR)
setenv JULIA_MPIEXEC                srun
```

  c. Provide compatibility interfaces, eg. `MPItrampoline`
  d. Modules include pre-compiled packages in the `JULIA_DEPOT_PATH` and `JULIA_LOAD_PATH`

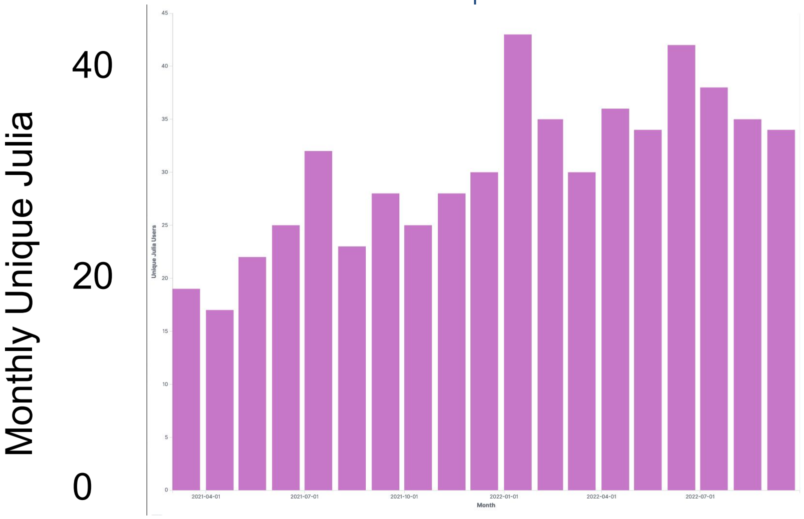


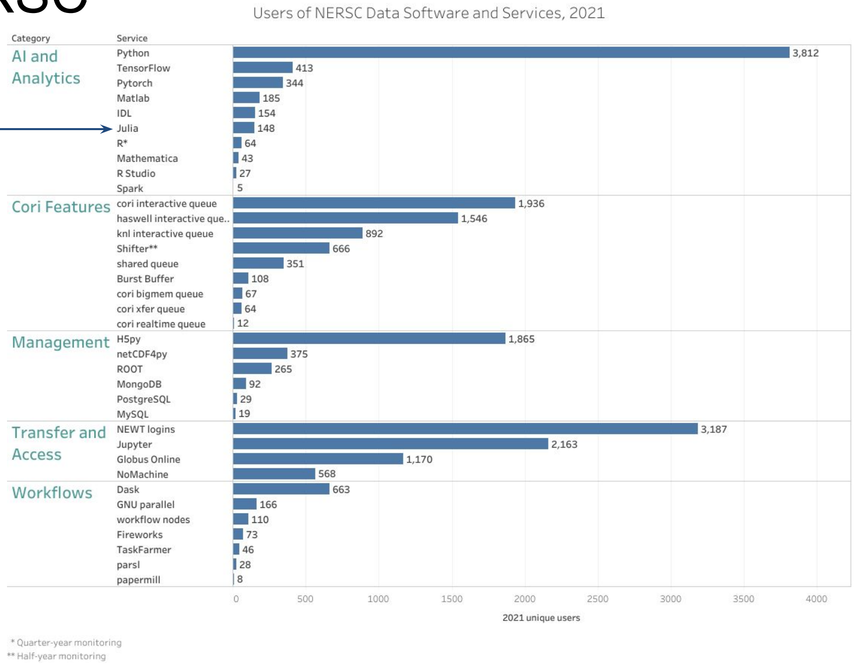
```
## Software-specific settings exported to user environment
module       load              julia/settings-$mpich_compiler
prepend-path PATH              $root/bin
prepend-path PATH              $admin_depot/bin
prepend-path JULIA_DEPOT_PATH $env(HOME)/.julia/nersc/$platform:$pkg_depot
setenv       JULIA_ADMIN_PATH $admin_depot
prepend-path JULIA_LOAD_PATH  "@:@v#.#:$admin_depot/environments/globalenv:@stdlib"
```

Julia Usage Trends at NERSC

- Growing use of Julia modules at NERSC

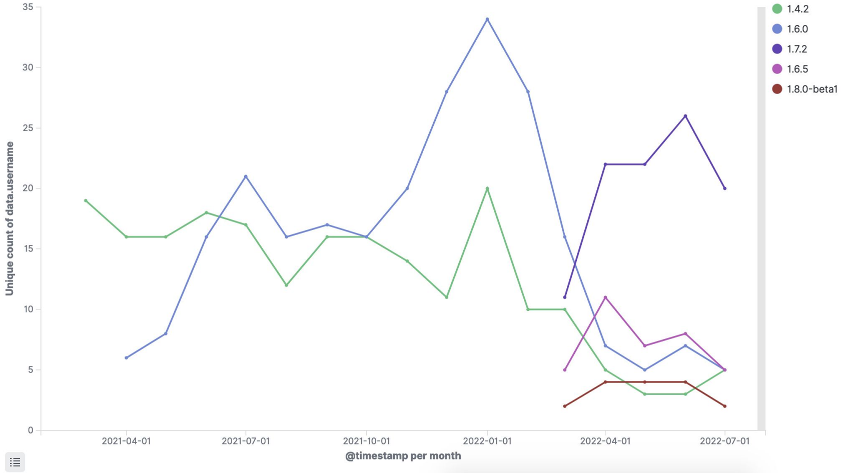
229 module users
(04/21-07/22)





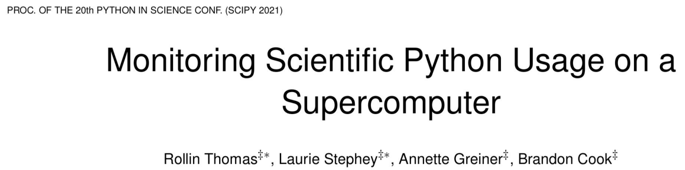
Julia Usage Trends at NERSC

- Julia Users like new versions
- Difficult for center software release cycle to keep up with latest Julia version
    - ○ Use CI/CD to keep up to date
    - ○ Enable users to be productive with their own Julia versions



Ongoing and Future Work

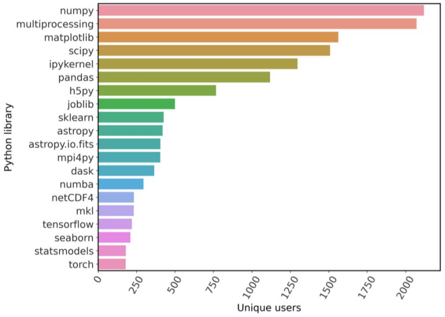
- **Detailed Usage Monitoring:** Use `startup.jl` to register `atexit` hook which monitors loaded packages



PROC. OF THE 20th PYTHON IN SCIENCE CONF. (SCIPY 2021)

Monitoring Scientific Python Usage on a Supercomputer

Rollin Thomas[‡*], Laurie Stephey[‡*], Annette Greiner[‡], Brandon Cook[‡]



- **Production-Level Support:** Optimize Julia performance (eg. GPUs) on NERSC systems and integrate support into center operations
- **Advanced Workflow Control:** Explore how workflow managers interact with center resource scheduler (eg. Slurm) *in situ* using API (eg. PMI2)
- **Documentation, Use Cases, and Training**

Noteworthy Julia Packages (for HPC)

- **JuliaIO**: https://github.com/JuliaIO
  JuliaData: https://github.com/JuliaData
  Collects many Julia packages around I/O and Data
- **JuliaParallel**: https://github.com/JuliaParallel
  Collects many Julia packages around distributed and parallel computing
- **JuliaGPU**: https://github.com/JuliaGPU
  Collects many Julia packages used for GPU computing

Noteworthy I/O Packages

- **Pidfile.jl**: Provides the linux/unix pidfile mechanism to hold mutex'es – useful for locking files

- **HDF5.jl**: HDF5-file support
- **Zarr.jl**: Julia Zarr (N-D array compressed data) support
- **JLD.jl / JLD2.jl**: Julia-native serialization support
- **Tables.jl / DataFrames.jl / CSV.jl**: Tabular data support

- **JuliaDB.jl**: A distributed database for tables (implemented in pure Julia)

Noteworthy REST and Web Frameworks

- **HTTP.jl**: Send and receive HTTP requests
- **Mux.jl / Oxygen.jl**: Routing middleware for HTTP requests – Oxygen is newer and makes multithreading easier (considered an all-Julia replacement for FastAPI)
- **Genie.jl**: Fully-fledged web development framework (Julia’s answer to Flask)

Noteworthy HPC Packages

“Traditional” HPC support: (https://github.com/JuliaParallel)

- **MPI.jl**: no explanation needed (it is CUDA/ROCM-aware)
- **ClusterManagers.jl**: manager HPC resources on the fly (also note **SlurmClusterManager.jl** and **MPIClusterManagers.jl** for HPC clusters)
- **ImplicitGlobalGrid.jl / MPIArrays.jl**: implement a global address space (using the Array interface) built on MPI.jl

Noteworthy HPC Packages

Tasking (producer-consumer) style HPC support: (https://github.com/JuliaParallel)

- **Distributed.jl** / **Dagger.jl**: task-based parallelism (like Dask and Ray)
- **DTables.jl** / **DistributedArrays.jl**: arrays and tables build on distributed

ML support: **Flux.jl** (like pytorch, but different)

Noteworthy HPC Packages

GPU Support:
(https://github.com/JuliaGPU)

- **CUDA.jl** / **AMDGPU.jl** / **oneAPI.jl**: low-level GPU support (expose GPU Array interface + helper functions to manage GPU resources)
- **KernelAbstractions.jl**: lets you write portable code by writing portable kernels (a bit “like” Kokkos)
- + Many Many more

Demo Time!

(sources: https://jblaschke.github.io/HPC-Julia/)