2019 Collegeville Workshop on Sustainable Scientific Software (CW3S19)
July 22-24, Collegeville, MN

From Research Prototype to Production Software: The Lessons of the SLATE Project

Jakub Kurzak <kurzak@icl.utk.edu>
Mark Gates
Ali Charara
Jack Dongarra

Innovative Computing Laboratory
University of Tennessee

ScaLAPACK

Implementation Language

- FORTRAN 77

Fork–Join Execution

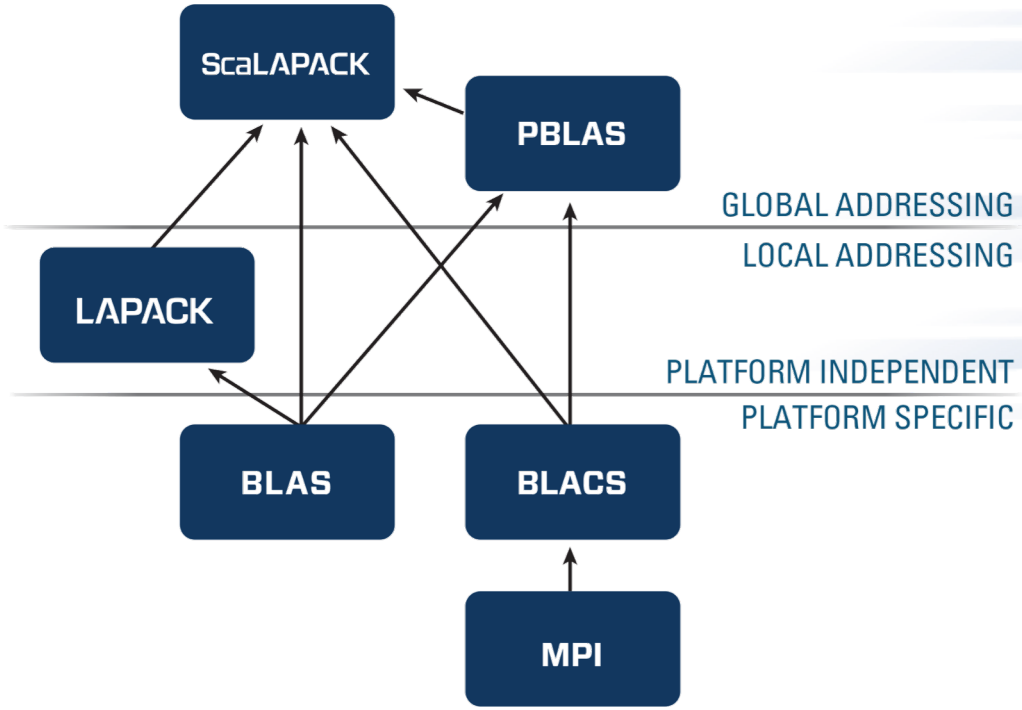
- parallelism in PBLAS

Hardcoded Memory Layout

- 2D bock-cyclic

Extremely Memory-Conservative Yet Wasteful

- requires the user to allocate workspaces
- stores symmetric and triangular matrices as full matrices



Design Challenges

No unified model for programming distributed-memory, multi-GPU systems

- MPI+OpenMP+CUDA/HIP?
- Managed Memory?
- NCCL?

No standardized solution for node-level memory coherency

- OpenMP/OpenACC compiler directives?
- OpenMP/OpenACC runtime API?
- CUDA/HIP runtime API?
- Managed Memory?

No standardized solution for programming GPU kernels

- ~~OpenCL~~
- OpenACC?
- CUDA/HIP?

Design Challenges

OpenMP tasking does not mix well with MPI

- MPI_THREAD_MULTIPLE is costly
- MPI_TASK_MULTIPLE is needed

OpenMP tasking does not mix well with GPU APIs

- task are asynchronous by definition, kernels are not
- stream management becomes a nuisance

Development of batched BLAS is lagging

- We need batched kernels for everything
- NVIDIA has little, AMD has nothing

C++11 and up

Standard Library

- containers
  - `std::map`
  - `std::list`
  - `std::set`
  - `std::tuple`
  - `std::vector`
- numerical utilities
  - `std::min()`
  - `std::max()`
  - `std::copysign()`
  - `numeric_limits<T>::min()`
- parallel utilities
  - `std::atomic<T>`

Templates

- precisions
  - `float`
  - `double`
  - `std::complex<float>`
  - `std::complex<double>`
- targets
  - `Target::Host`
  - `Target::HostTask`
  - `Target::HostNest`
  - `Target::HostBatch`
  - `Target::Devices`

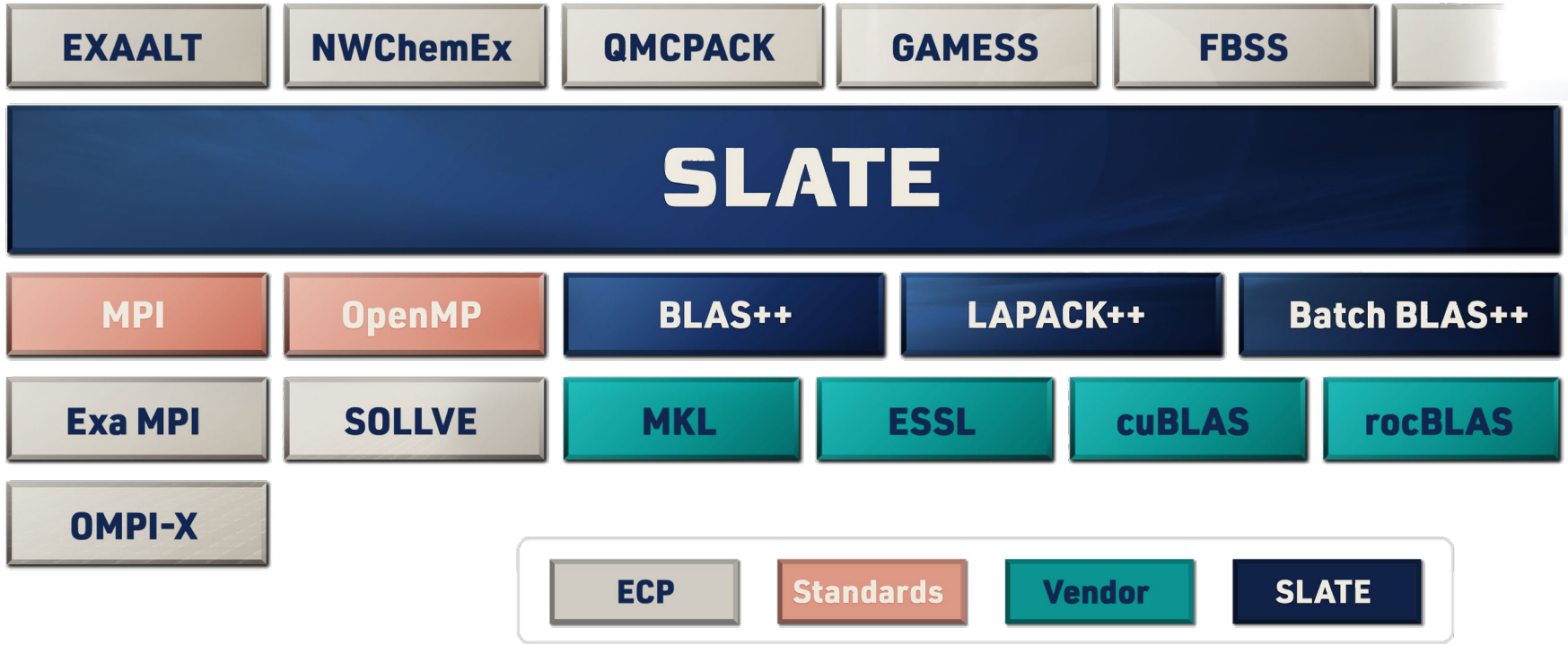
Standard Features

- inheritance
- overloading
- …

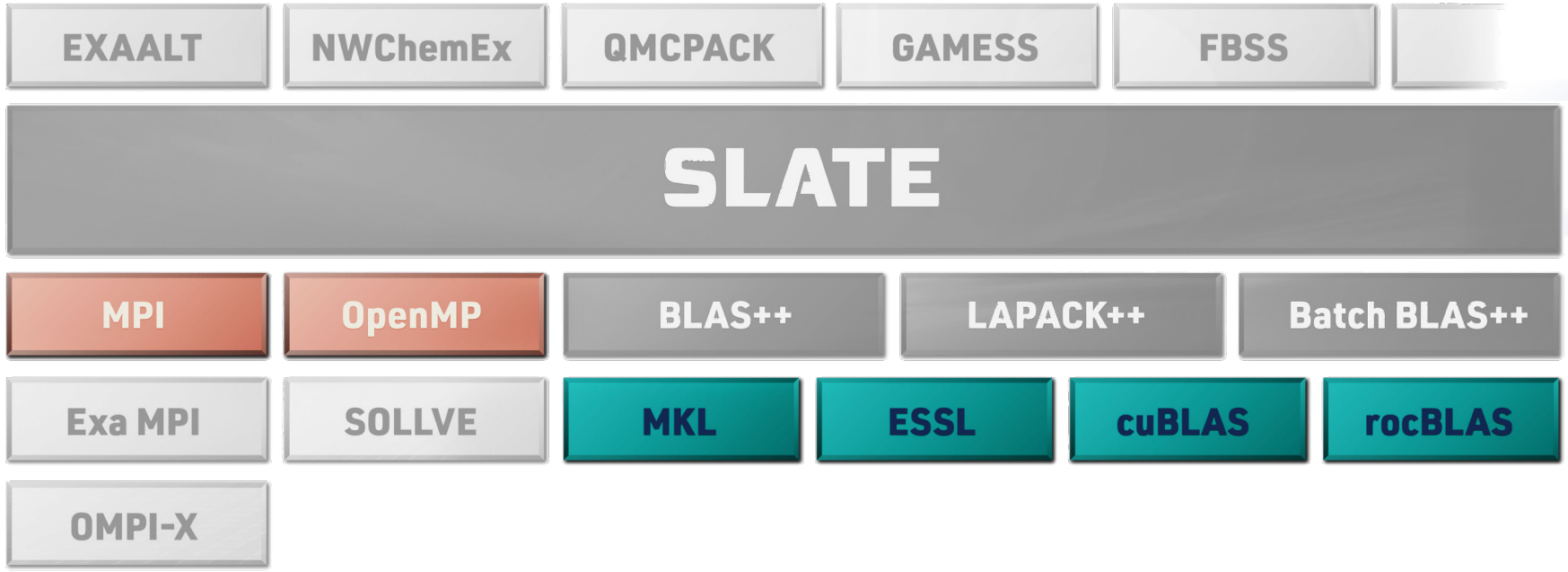
Modern Features

- rvalue references
- smart pointers
- …

SLATE



Standards



Waterfall

- design once
- implement once
- minimal maintenance required

Spiral

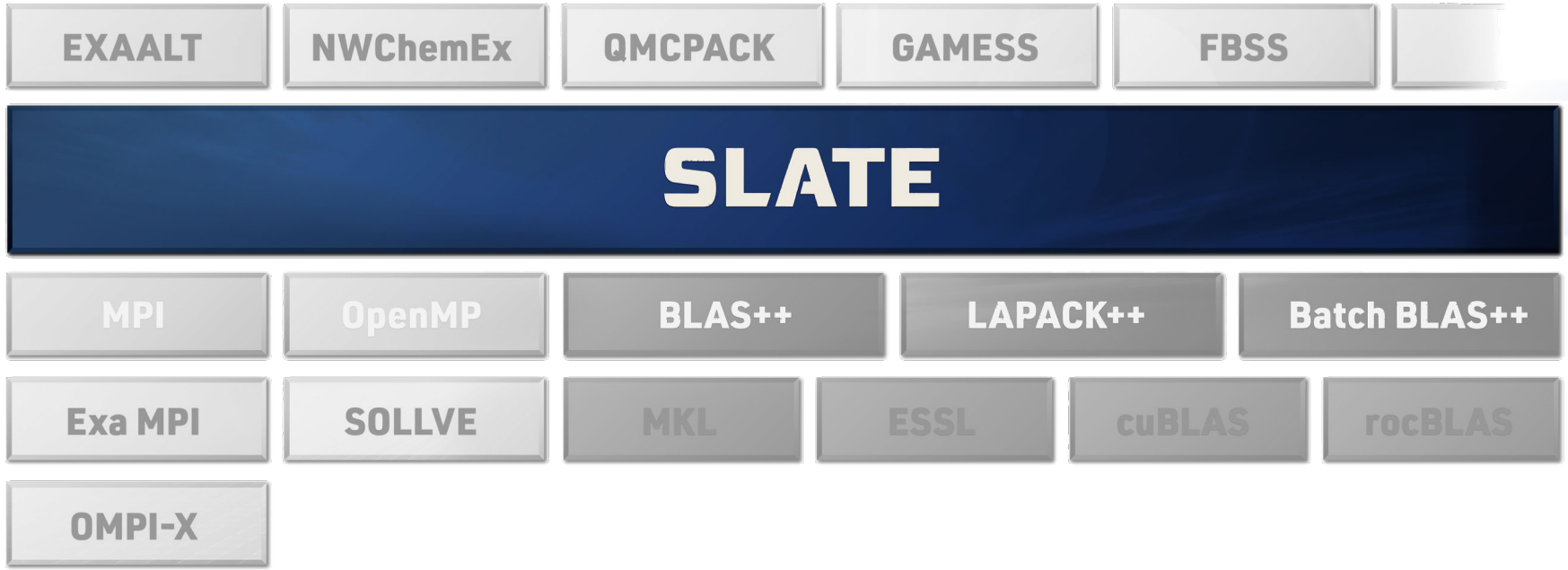
Gradually absorbs complexity from the higher level.

- design
- implement
- refine / add value
- implement

repeat

Agile/XP

E.g., MOSI memory consistency protocol.



- OO design (encapsulation)
- maximum compactness (minimum LOC)
- minimum functionality (delayed generalization)

- all technical risk goes here
- underlaying technologies in flux
- mixing paradigm – MPI+OpenMP+CUDA/HIP

Compactness

Templating precisions

- `float`
- `double`
- `std::complex<float>`
- `std::complex<double>`

Handling of `side, uplo, trans`

- ScaLAPACK
  - 9 cases of `gemm` (36 blocks of code total)
  - 8 cases of `trmm` (32 blocks of code total)
- SLATE
  - 1 case of `gemm` (1 bock of code)
  - 2 cases of `trmm` (2 blocks of code)

- storing `uplo` and `trans` in the matrix object
- implementing inexpensive shallow copy transposition
- swapping tiles' indices and setting trans appropriately for the underlying tile operations

Compactness

Templating targets

- `Target::Host`
- `Target::HostTask`
- `Target::HostNest`
- `Target::HostBatch`
- `Target::Devices`

- MAGMA
  - `magma_sgetrf`
  - `magma_sgetrf_m`
  - `magma_sgetrf_gpu`
  - `magma_sgetrf_mgpu`
  - `magma_dgetrf`
  - `magma_dgetrf_m`
  - `magma_dgetrf_gpu`
  - `magma_dgetrf_mgpu`
  - ...
- SLATE
  - `slate::getrf`

- **Get simple layers out of the way**
  - use Waterfall
  - use Spiral
- **Keep the complex layers compact**
  - use OO design
  - use templating
  - use Agile/XP development
- **Push complexity out of the complex layers – absorb in simple layers**