PLASMA

Parallel Linear Algebra Software for Multicore Architectures

Innovative Computing Laboratory
Electrical Engineering and Computer Science
University of Tennessee
Piotr Luszczek (presenter)

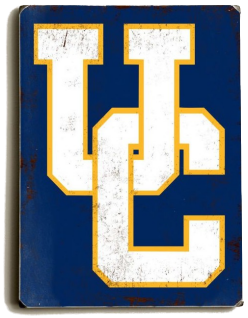
web.eecs.utk.edu/~luszczek/conf/

TOC: Table of Contents

- Functionality
- Performance
- Software Stack
- Documentation
- Installation
- Matrix Layout
- Algorithms
- Classes of routines and interfaces
- Settings

PLASMA Institutions

Knoxville Tennessee



Berkeley California

Denver Colorado

Microsoft

FUJITSU

shaping tomorrow with you

PLASMA People

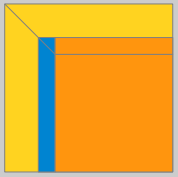
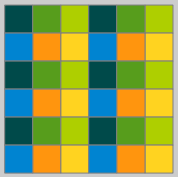
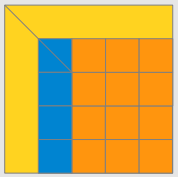
- Current Team
  - Dulceneia Becker
  - Henricus Bouwmeester
  - Jack Dongarra
  - Mathieu Faverge
  - Mark Gates
  - Azzam Haidar
  - Blake Haugen
  - Jakub Kurzak
  - Julien Langou
  - Hatem Ltaief
  - Piotr Łuszczek
  - Ichitaro Yamazaki
  - Asim YarKhan
  - Vijay Joshi

- Past Members
  - Emmanuel Agullo
  - Wesley Alvaro
  - Alfredo Buttari
  - Bilel Hadri

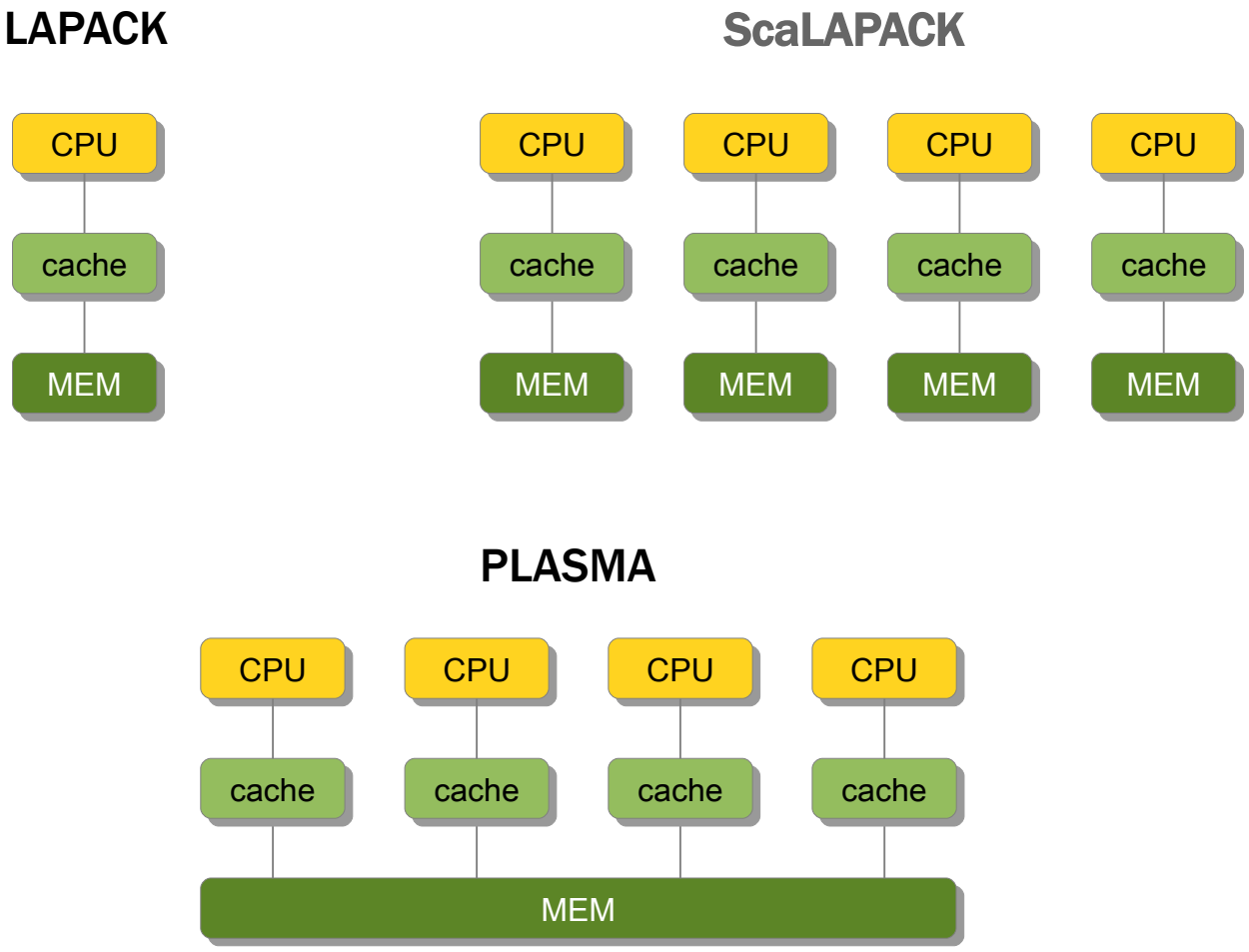
- Outside Contributors
  - Fred Gustavson
  - Lars Karlsson
  - Bo Kågström

names listed alphabetically

Dense Linear Algebra: Software Evolution

	LINPACK (70's) vector operations	• Level 1 BLAS
	LAPACK (80's) block operations	• Level 3 BLAS
	**ScaLAPACK** (90's) block cyclic data distribution	• PBLAS • BLACS (message passing)
	**PLASMA** (00's) tile operations	• tile layout • dataflow scheduling

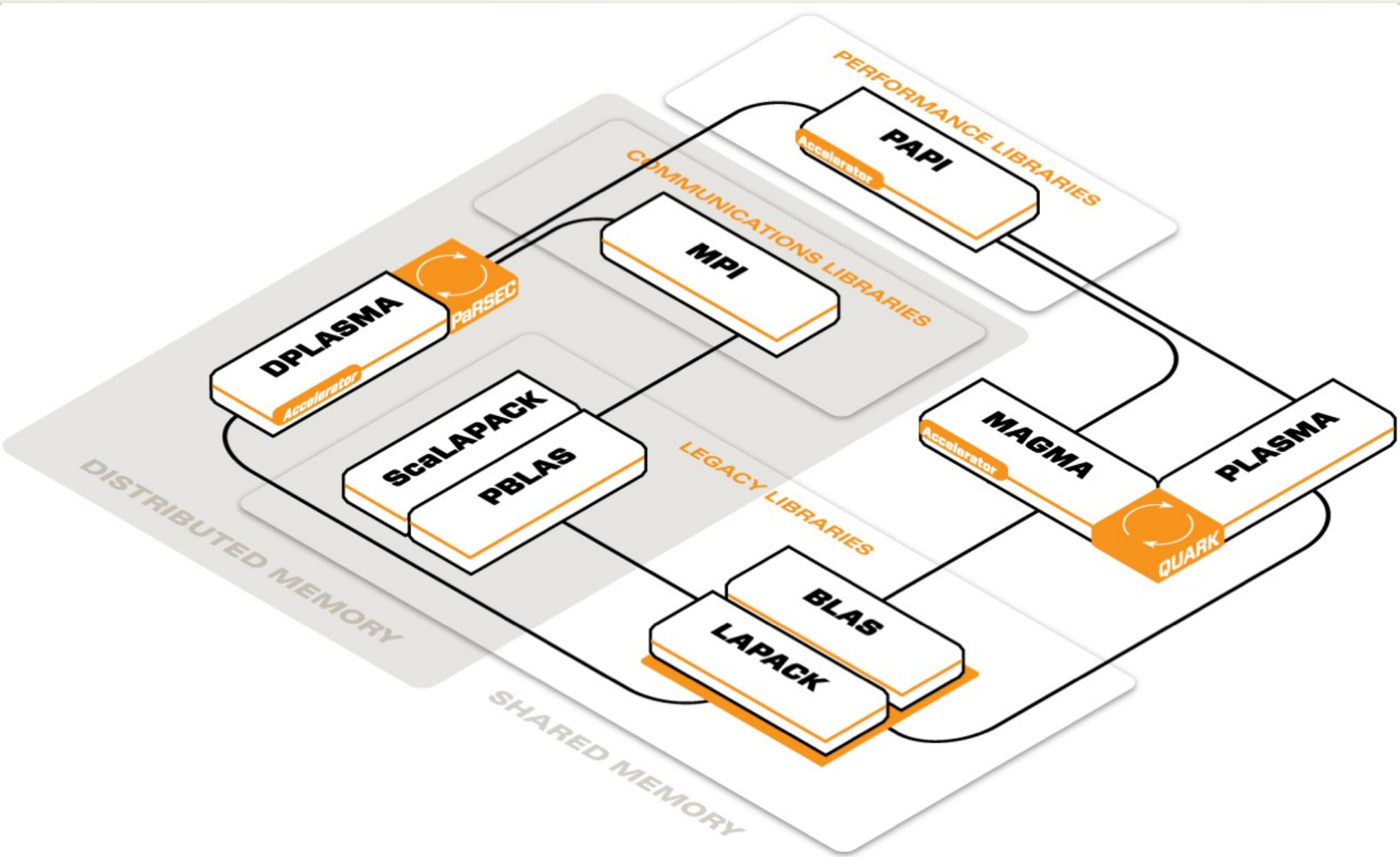
Dense Linear Algebra: Architectural Models



Dense Linear Algebra: Related Developments

- Matrix Algebra for GPUs and Multicore Architectures - MAGMA
  - shared memory
  - GPU acceleration
- Distributed Memory PLASMA – DPLASMA
- Parallel Runtime Scheduler & Execution Control - ParSEC
  - distributed memory
  - compact DAG representation
  - dynamic DAG scheduling
- Parallel Unified Linear Algebra with Systolic Arrays – PULSAR
  - distributed memory
  - Virtual Systolic Array architecture

ICL Project Map



PLASMA In a Nutshell

- Software Library
- Dense Linear Algebra
  - linear systems of equations
  - least square problems
  - singular value problems
  - eigenvalue problems
- Multicore Systems
  - multicore (e.g. 12-core AMD Magny-Cours)
  - multi-socket (e.g. quad-socket Dell PowerEdge)
  - shared memory
  - possibly NUMA

- all precisions (S, D, C, Z)
- Linux, Windows, Mac OS, AIX

hardware:
today – tens of cores
tomorrow – hundreds of cores

Linear Systems: AX = B

- Fully Determined m = n
  - PA = LU Gaussian elimination non-symmetric
  - $A = LL^T$ Cholesky factorization symmetric positive definite
  - $A = LDL^T$ LDL^T decomposition non-symmetric positive definite
- Overdetermined m > n (least squares)
  - A = QR
- Underdetermined m < n (minimum norm)
  - A = LQ

prototypes

Linear Systems: AX = B

- Explicit Matrix Inversion
  - non-symmetric
  - symmetric positive definite
  - explicitly calculating the inverse of a matrix
  - e.g., calculating the variance-covariance matrix in statistics
- Tall and Skinny Factorizations m <<n m>>n
  - fast (highly parallel) algorithms
  - based on tree reductions
- Mixed Precision Iterative Refinement
  - factorization in single precision
  - iterative refinement in double precision

Eigenvalues, SVD: $A=X\Lambda X^T$, $A = U\Sigma V^T$, $AX = \Lambda BX$

- Symmetric Eigenvalue Problem
  - reduction to "block tridiagonal" (band)
  - band reduction to tridiagonal (bulge chasing)
  - LAPACK eigensolver (QR/D&C/MRRR)
- Singular Value Problem
  - reduction to "block bidiagonal" (band)
  - band reduction to bidiagonal (bulge chasing)
  - QR algorithm (LAPACK)
- Generalized Eigenvalue Problem
  - Cholesky factorization of B and application to A
  - reduction to band & band reduction; LAPACK eigensolver (QR/D&C/MRRR)

non-symmetric eigenvalue problem
work in progress

extremely fast eigenvalues / singular values
fast eigenvectors / singular vectors
possibility of calculating a subset

symmetric positive definite matrices only

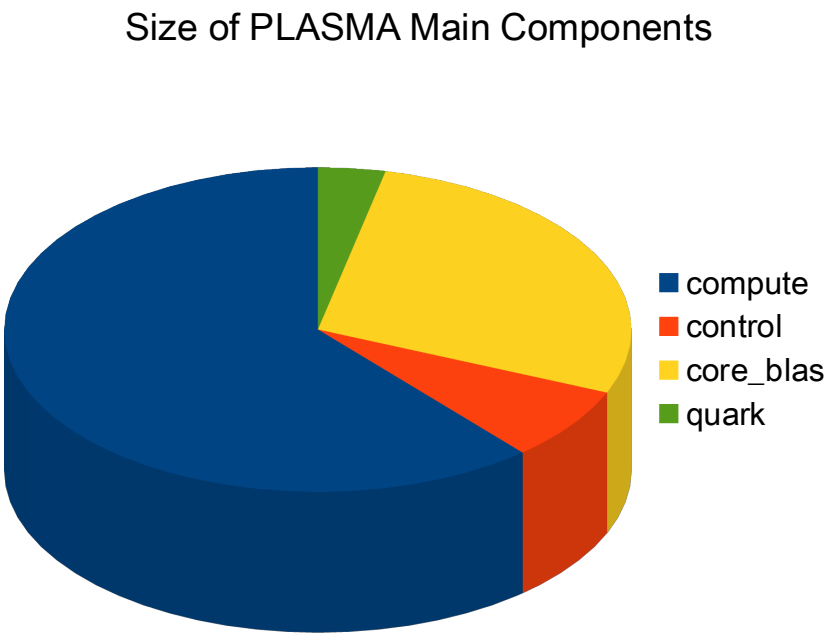
Auxiliary Functionality: Layouts & Tile BLAS 3

- In-Place Matrix Layout Translation
  - CM, RM (column-major / row-major)
  - CCRB, CRRB, RCRB, RRRB (column / row – rectangular block)
  - LAPACK ↔ PLASMA (CM ↔ CCRB)
  - any other combination
  - (fast transposition)
  - cache-efficient
  - parallel
- Tile BLAS 3
  - GEMM, HEMM, HER2K, HERK, SYMM, SYR2K, SYRK, TRMM, TRSM
    - Complete set

Thank you Fred, Lars, Bo!

There also is a fast out-of-place PLASMA ↔ LAPACK translation

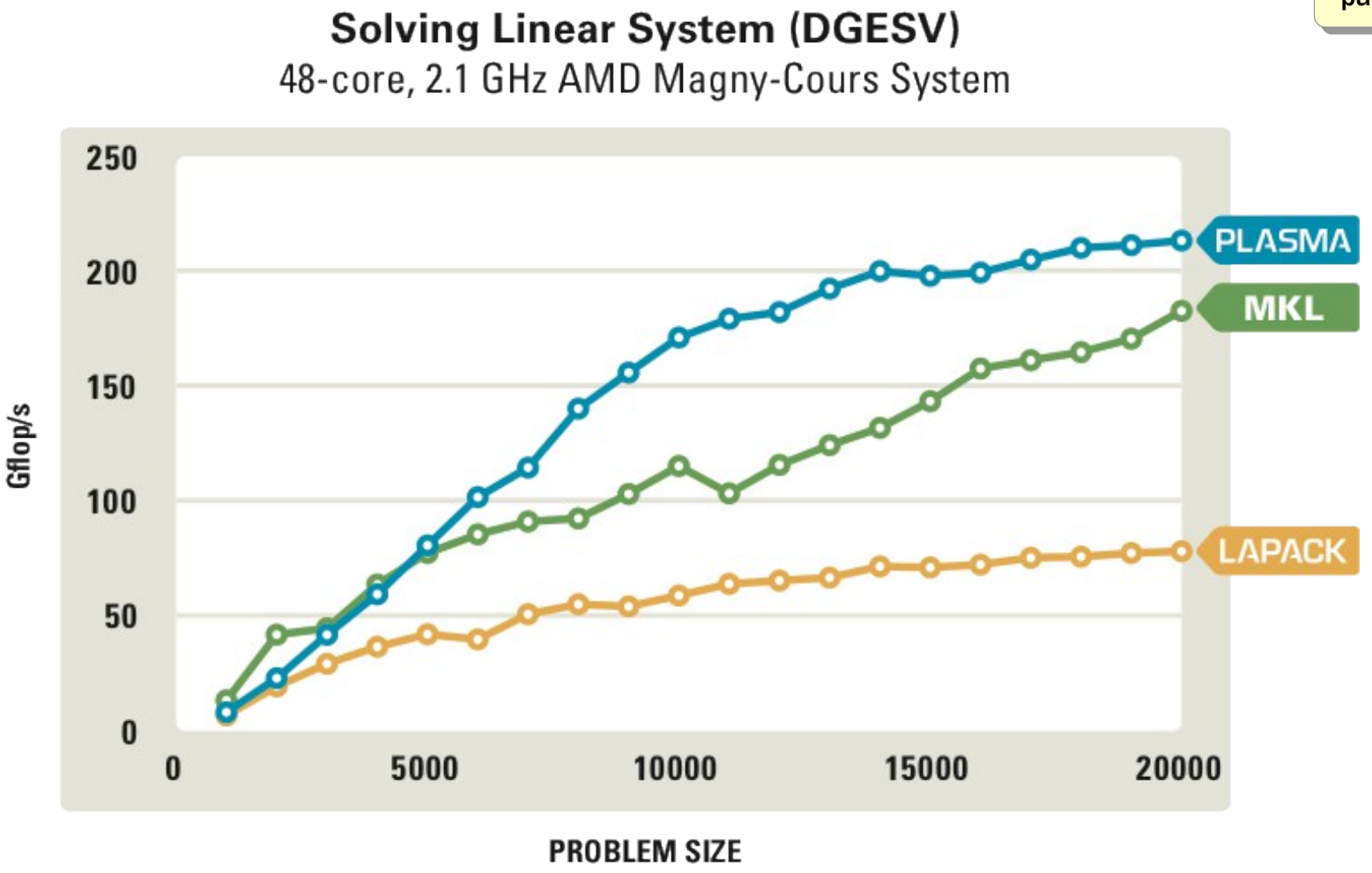
PLASMA Size In Numbers



- ~200,000 lines of code
- ~1,000 source files
- ~600 API functions

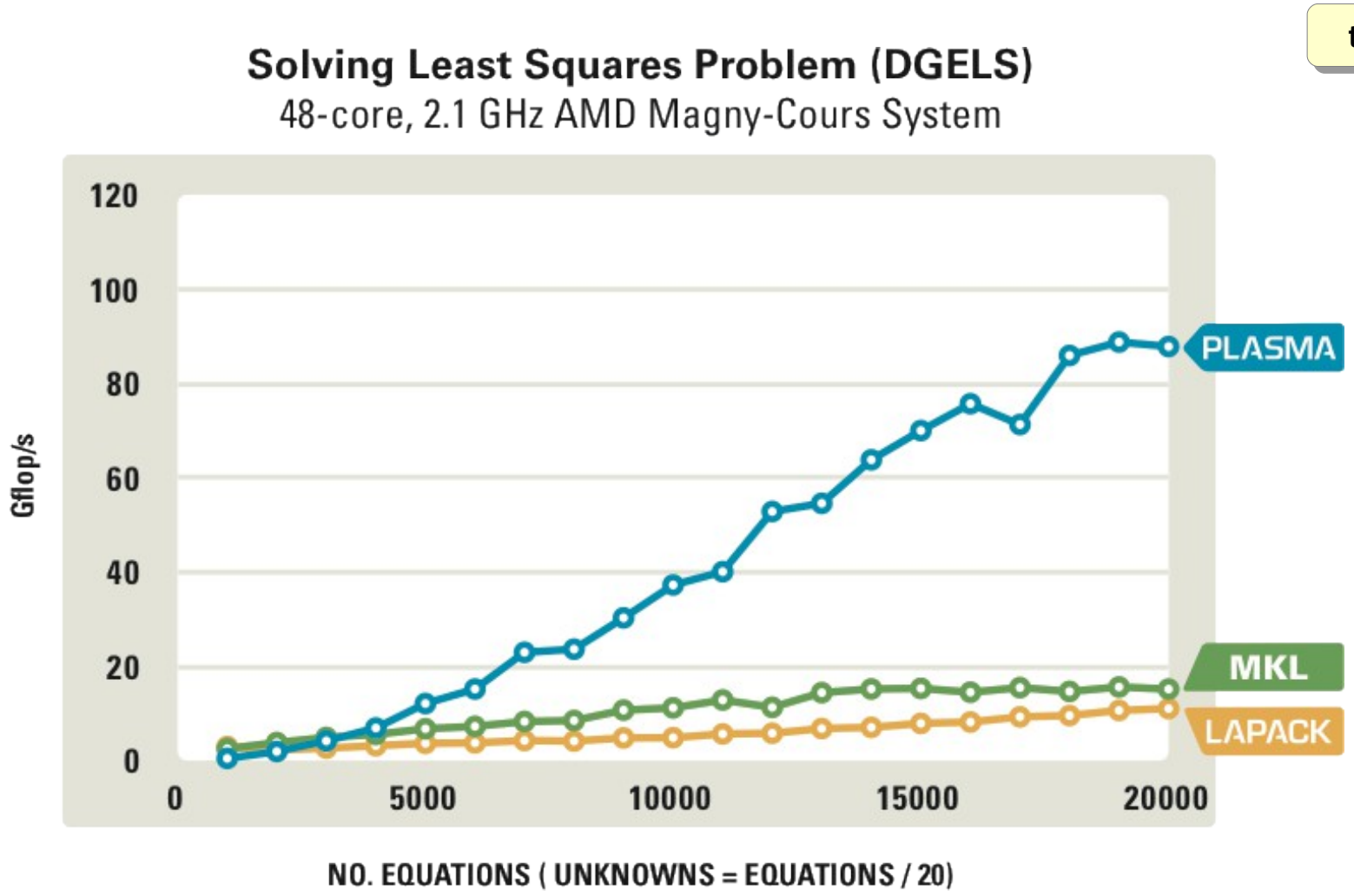
Motivation: Performance for LU Solve

partial pivoting



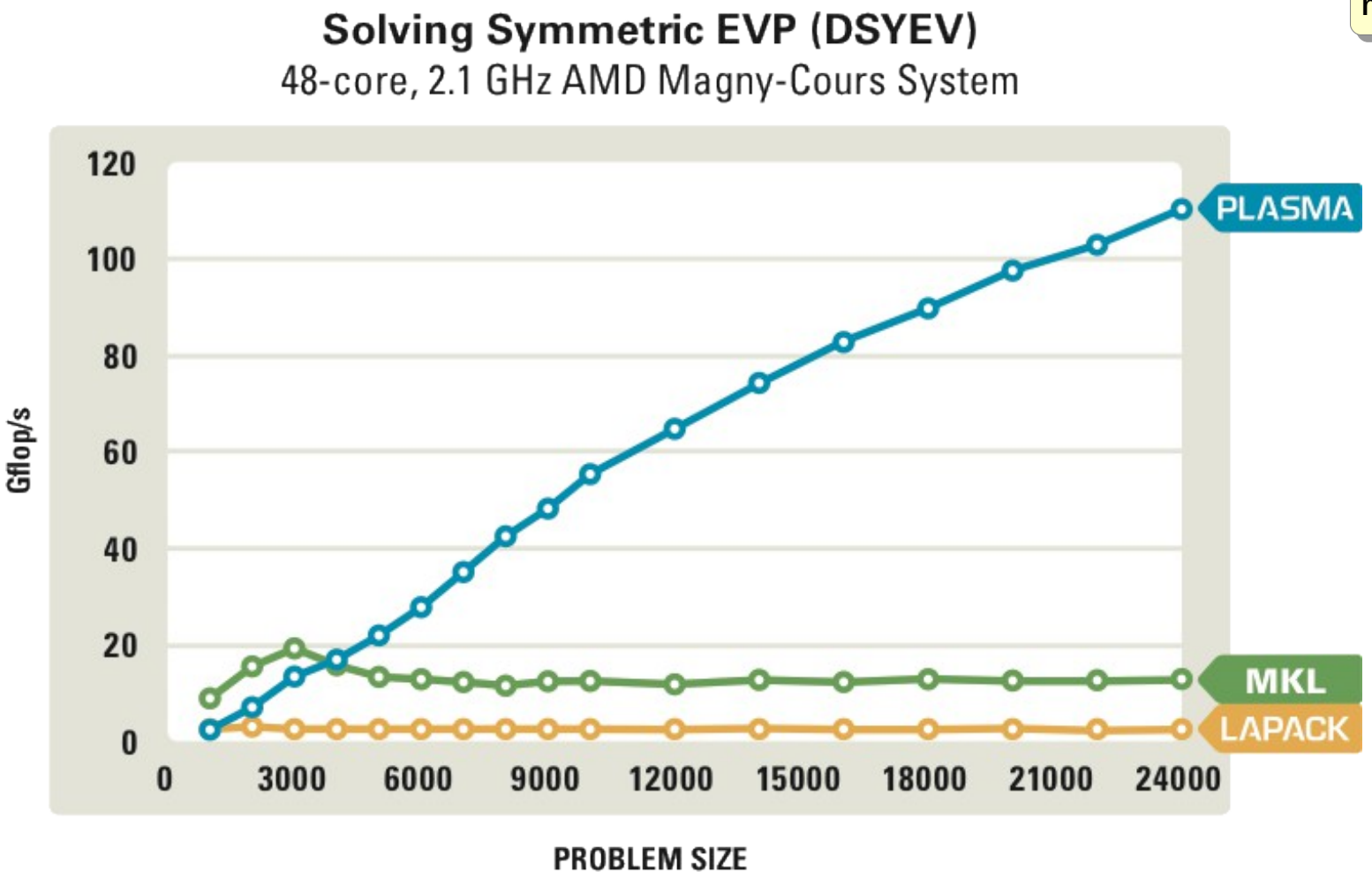
Motivation: Performance for QR Factorization

tree-based



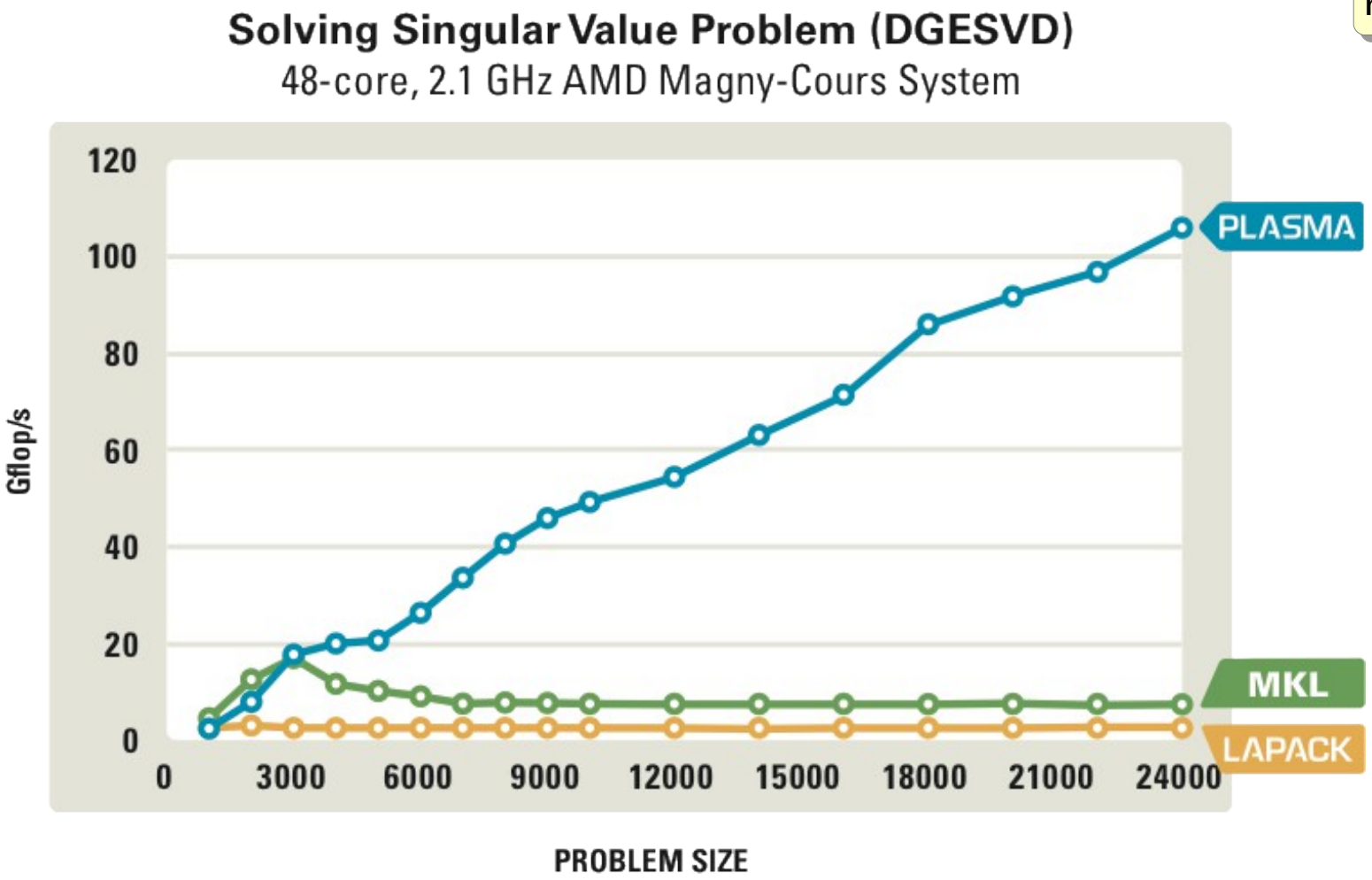
Motivation: Performance for Eigenvalues

no vectors requested

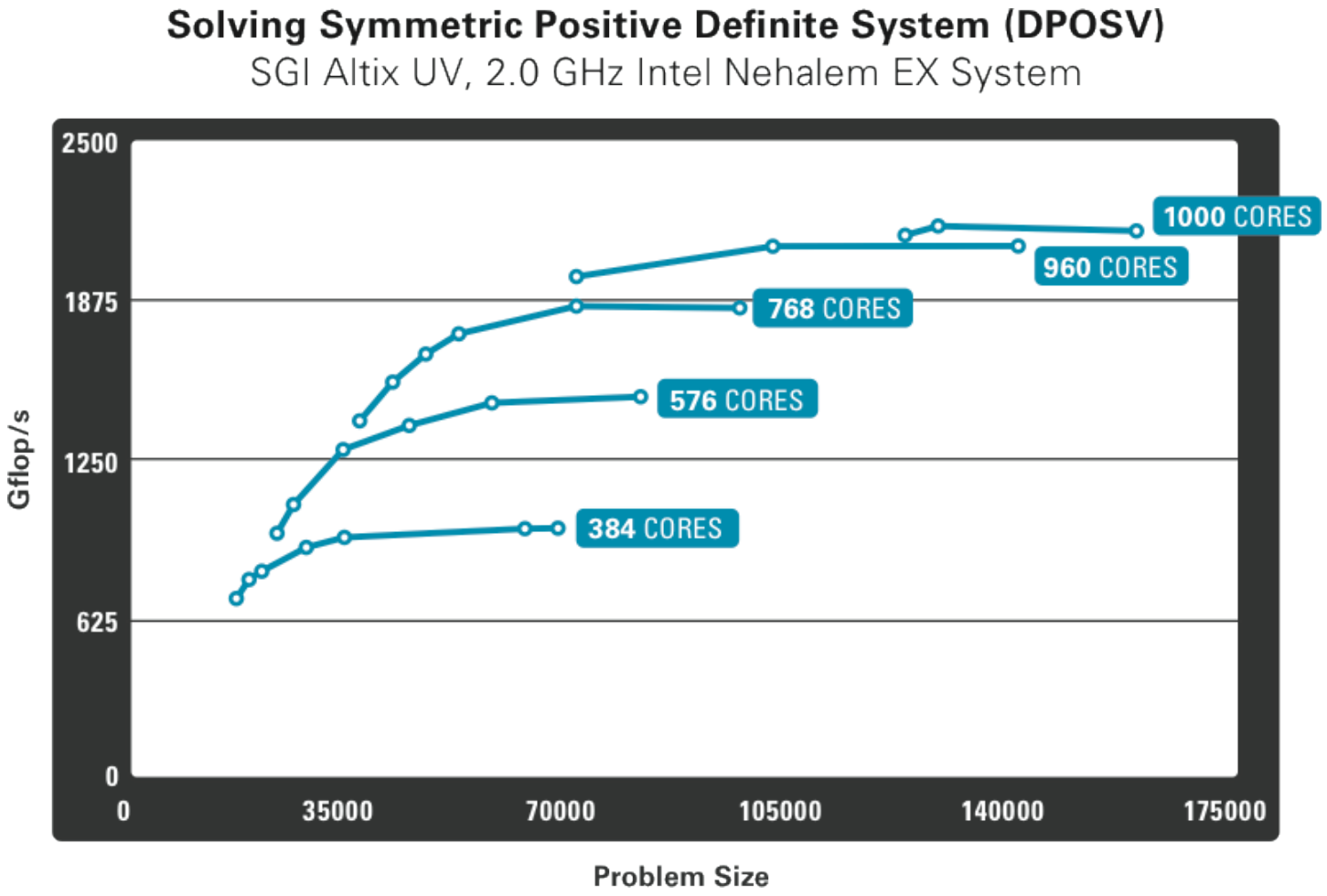


Motivation: Performance for Singular Values

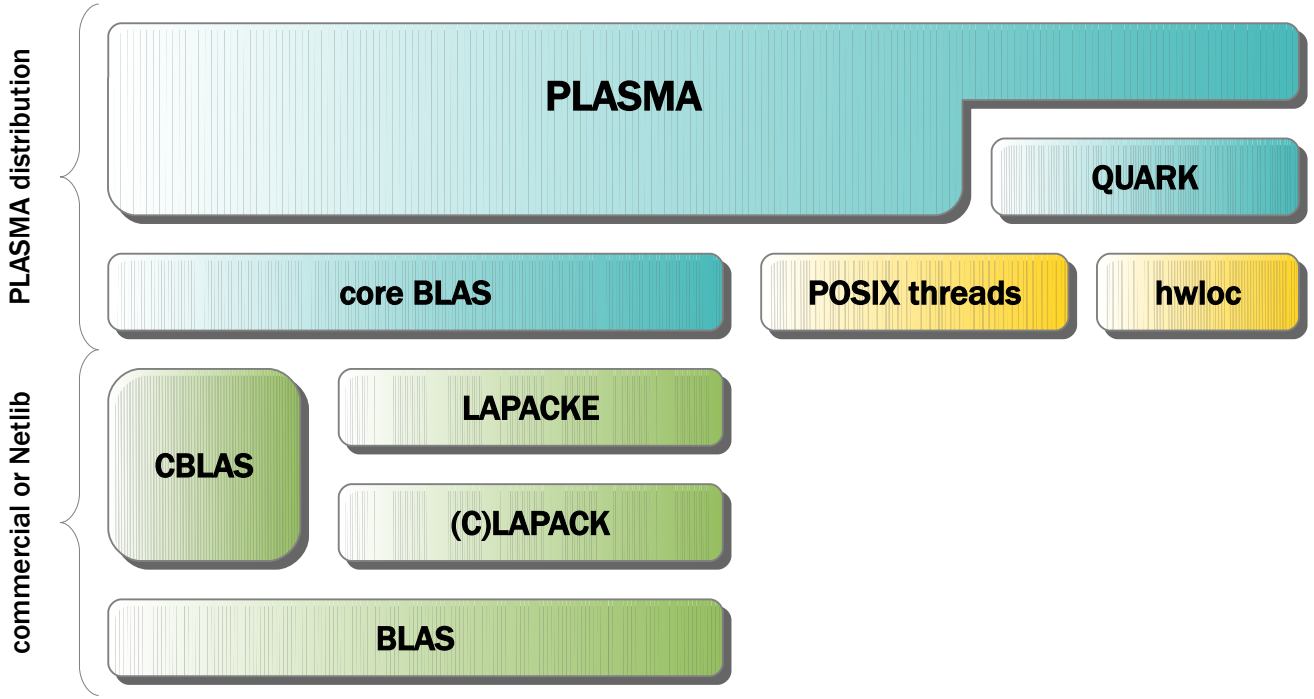
no vectors requested



Motivation: Performance on 1000 Cores



Software Stack of PLASMA



QUARK - **QU**euing **A**nd **R**untime for **K**ernels

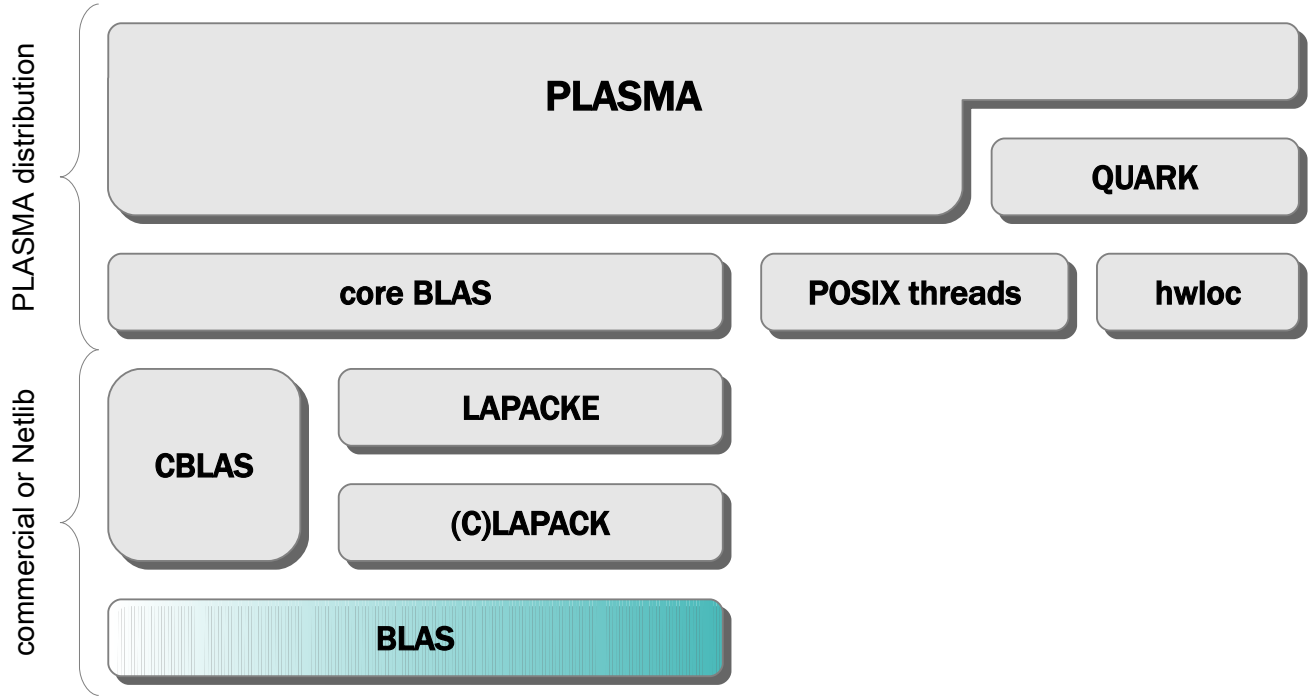
LAPACK - **L**inear **A**lgebra **PACK**age

BLAS - **B**asic **L**inear **A**lgebra **S**ubroutines

hwloc - **h**ardware **loc**ality

LAPACKE is now included with the Netlib LAPACK

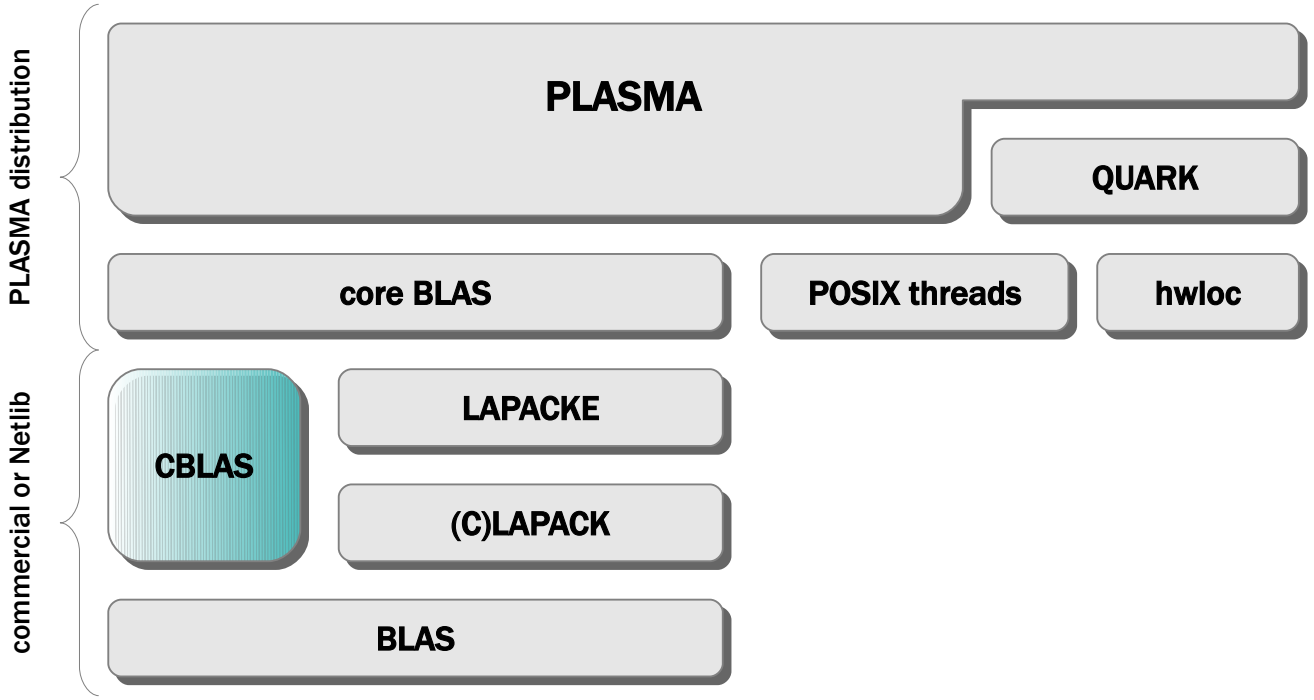
Software Stack: BLAS



- **Basic Linea Algebra Subroutines**
  - critical for performance
  - vendor: Intel MKL, AMD ACML, IBM ESSL, Apple VecLib, Cray LibSci
  - academic: ATLAS, Goto BLAS
  - FORTRAN *definition* available from Netlib

`www.netlib.org/blas/`

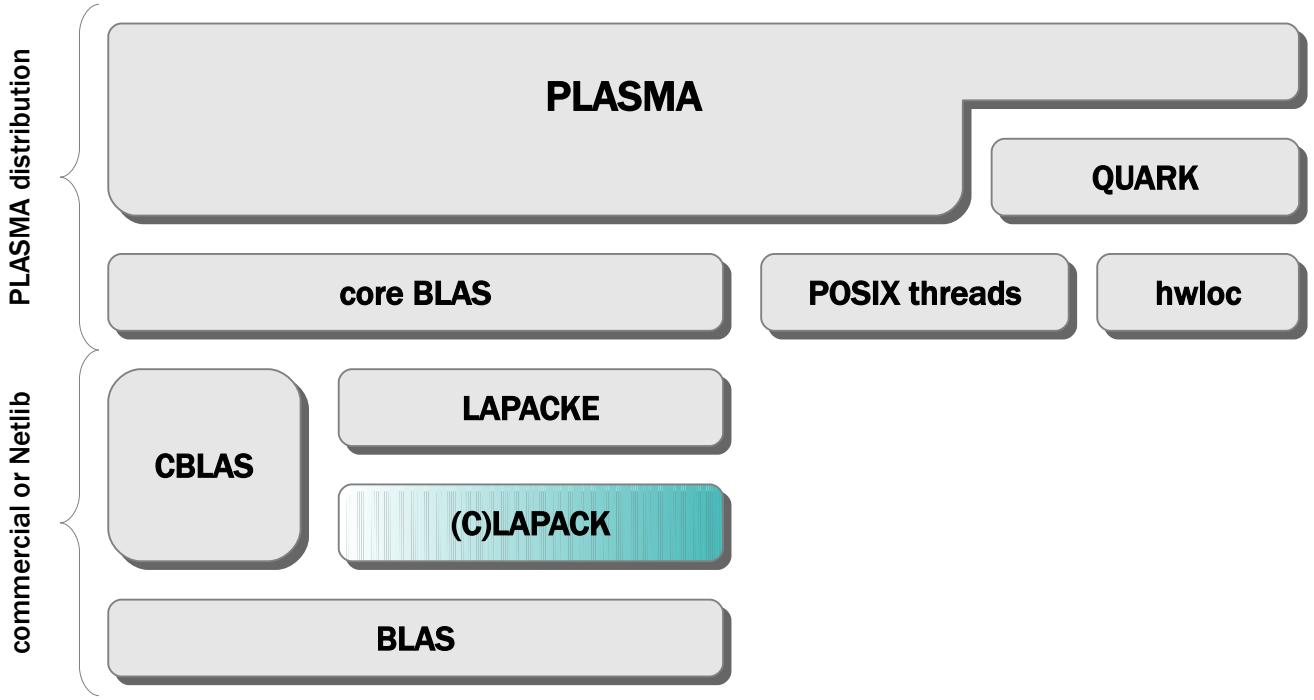
Software Stack: CBLAS



- **C API for BLAS**
  - a set of wrappers
  - not performance critical
  - commonly included in vendor and academic packages
  - available from Netlib

```
http://www.netlib.org/blas/blast-forum/cblas.tgz
```

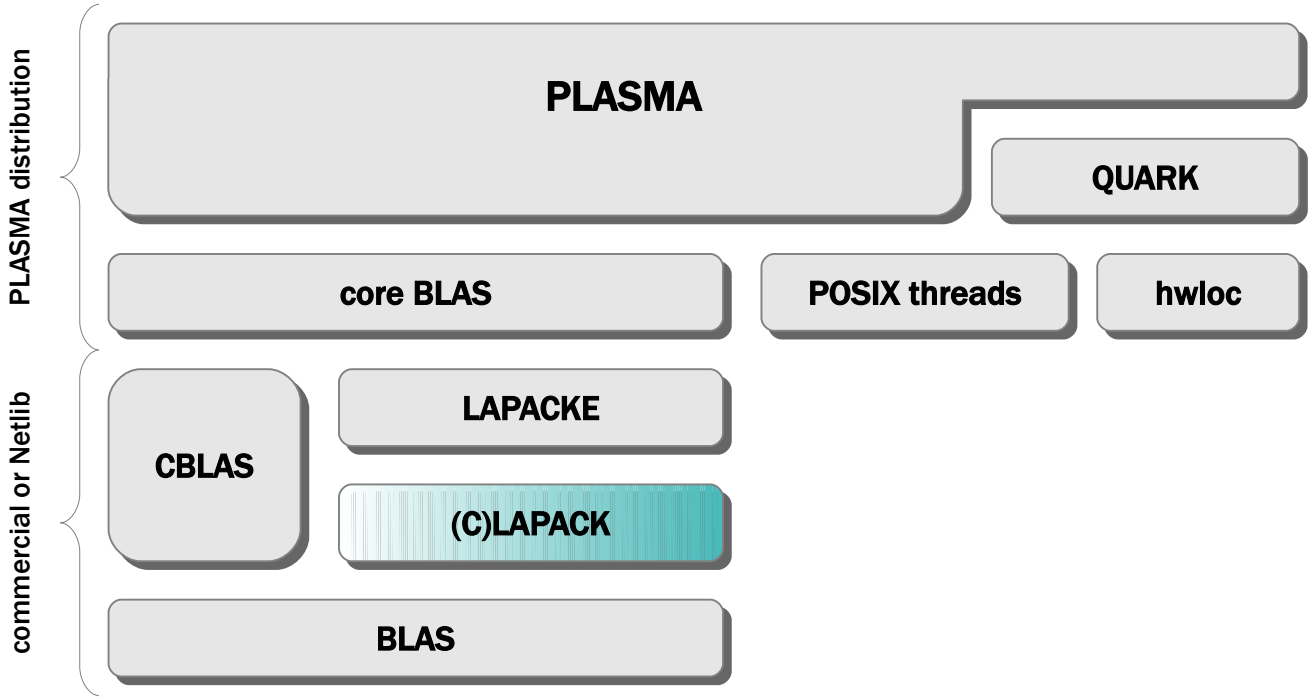
Software Stack: LAPACK



- **Linear Algebra PACKage**
  - dense linear algebra software library
  - FORTRAN implementation
  - no parallel constructs
  - available from Netlib

`www.netlib.org/lapack/`

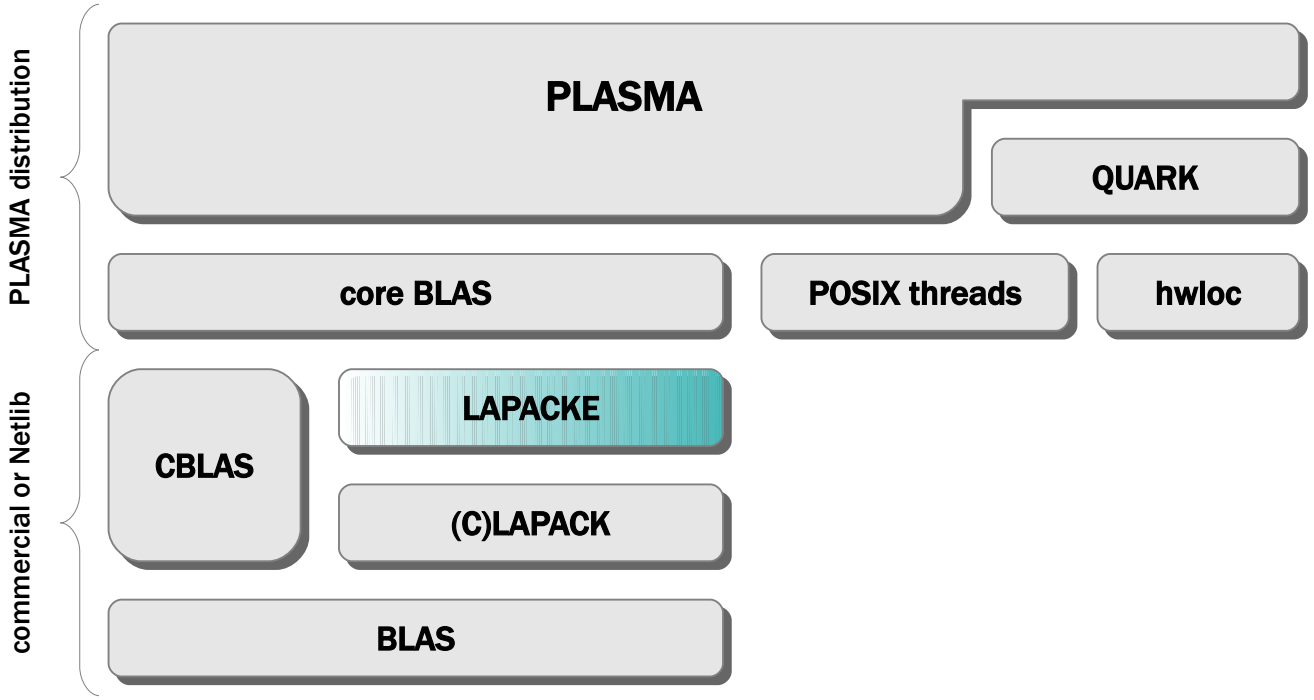
Software Stack: CLAPACK



- **C Implementation of LAPACK**
  - automatically generated
  - C implementation
  - FORTRAN API
  - available from Netlib

`www.netlib.org/clapack/`

Software Stack: LAPACKE



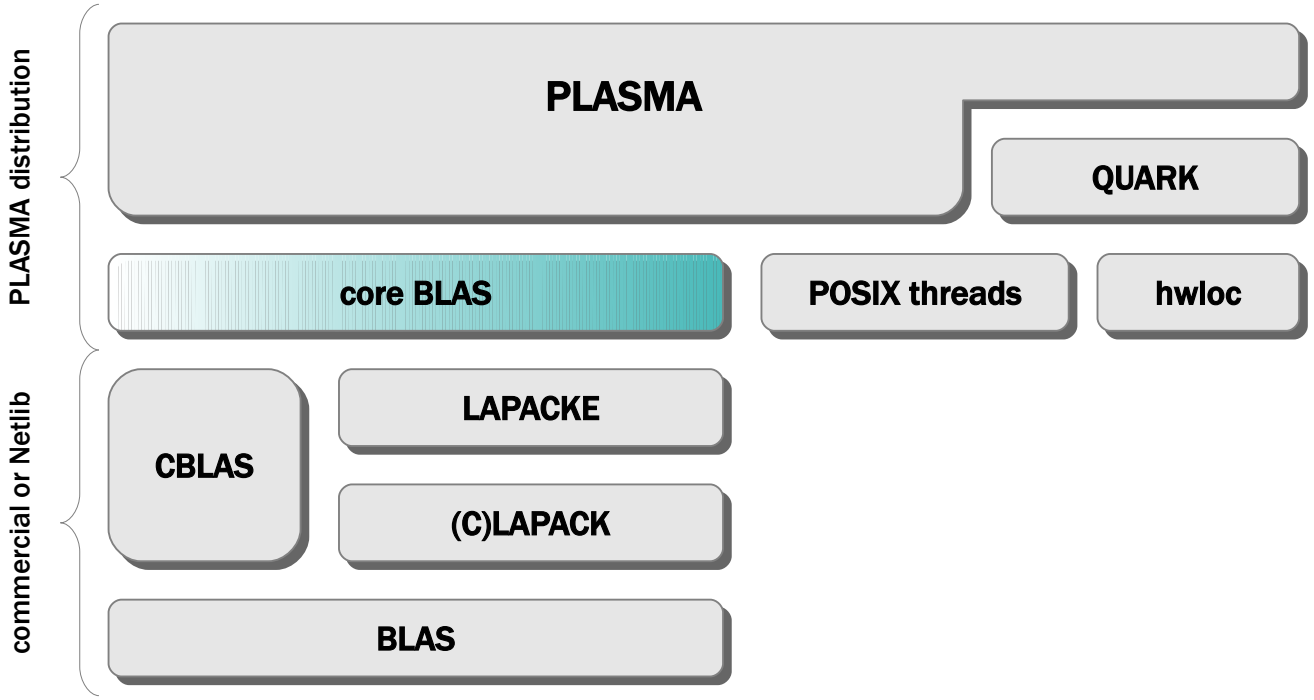
- **C API for LAPACK & CLAPACK**
  - a set of wrappers
  - developed by: LAPACK team & Intel
  - provided by: Netlib & MKL
  - now included in Netlib LAPACK

U Tennessee, UC Berkeley, UC Denver

`www.netlib.org/lapack/lapacke.tgz`

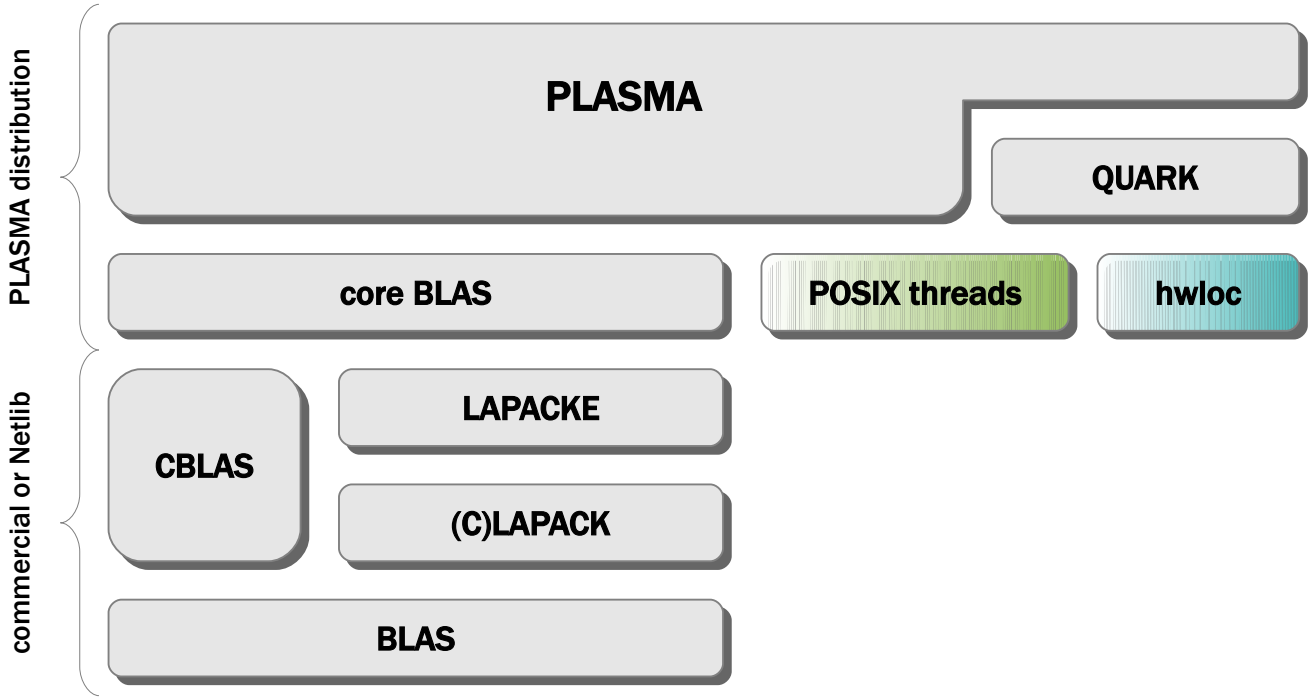
`icl.cs.utk.edu/projectsfiles/plasma/pubs/lapacke-3.3.0.tgz`

Software Stack: core BLAS



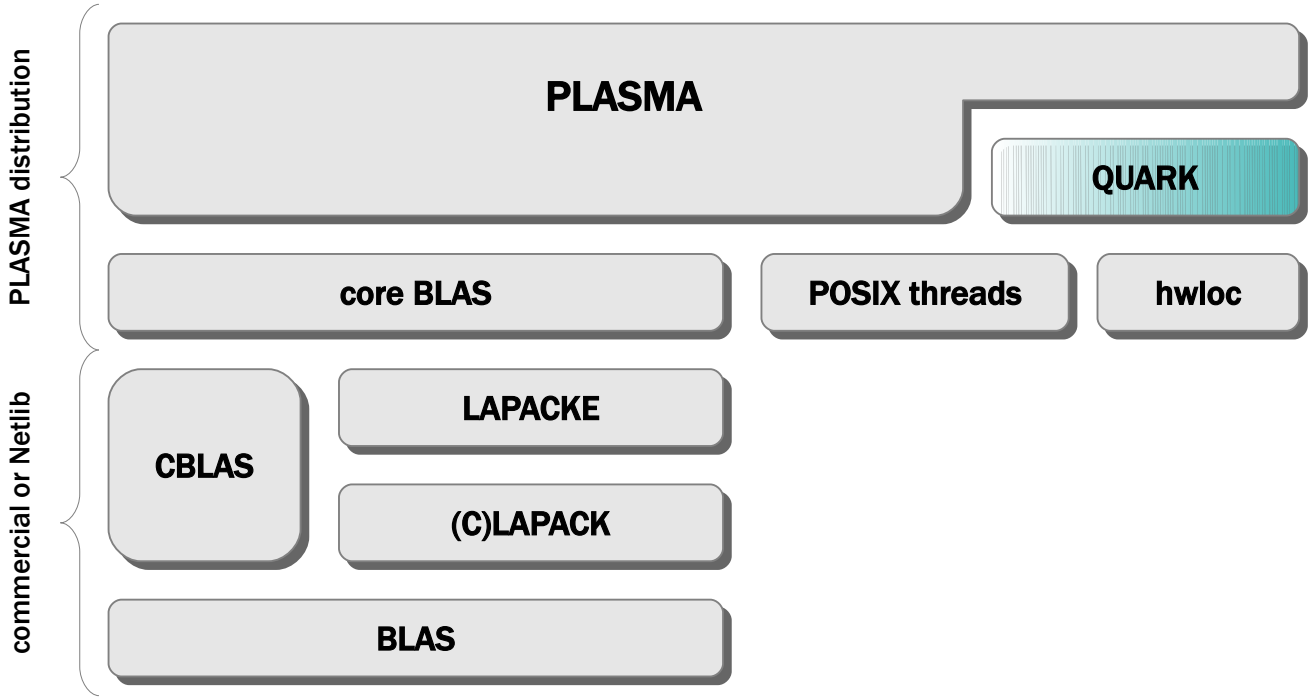
- **core BLAS**
  - serial kernels
  - PLASMA's building blocks
  - ideally monolithic & highly tuned
  - currently implemented with multiple calls to BLAS

Software Stack: pthreads & hwloc



- **hwloc – Portable Hardware Locality**
  - topology abstraction
  - caches / NUMA nodes / sockets / cores / SMT
  - in PLASMA thread affinity control (thread placement)

Software Stack: QUARK



- **Queuing and Runtime for Kernels**
  - API-based
  - superscalar scheduler
  - similar to StarPU and SMPSs
  - suitable for implementing a numerical library

PLASMA Software Engineering: Precision Generation

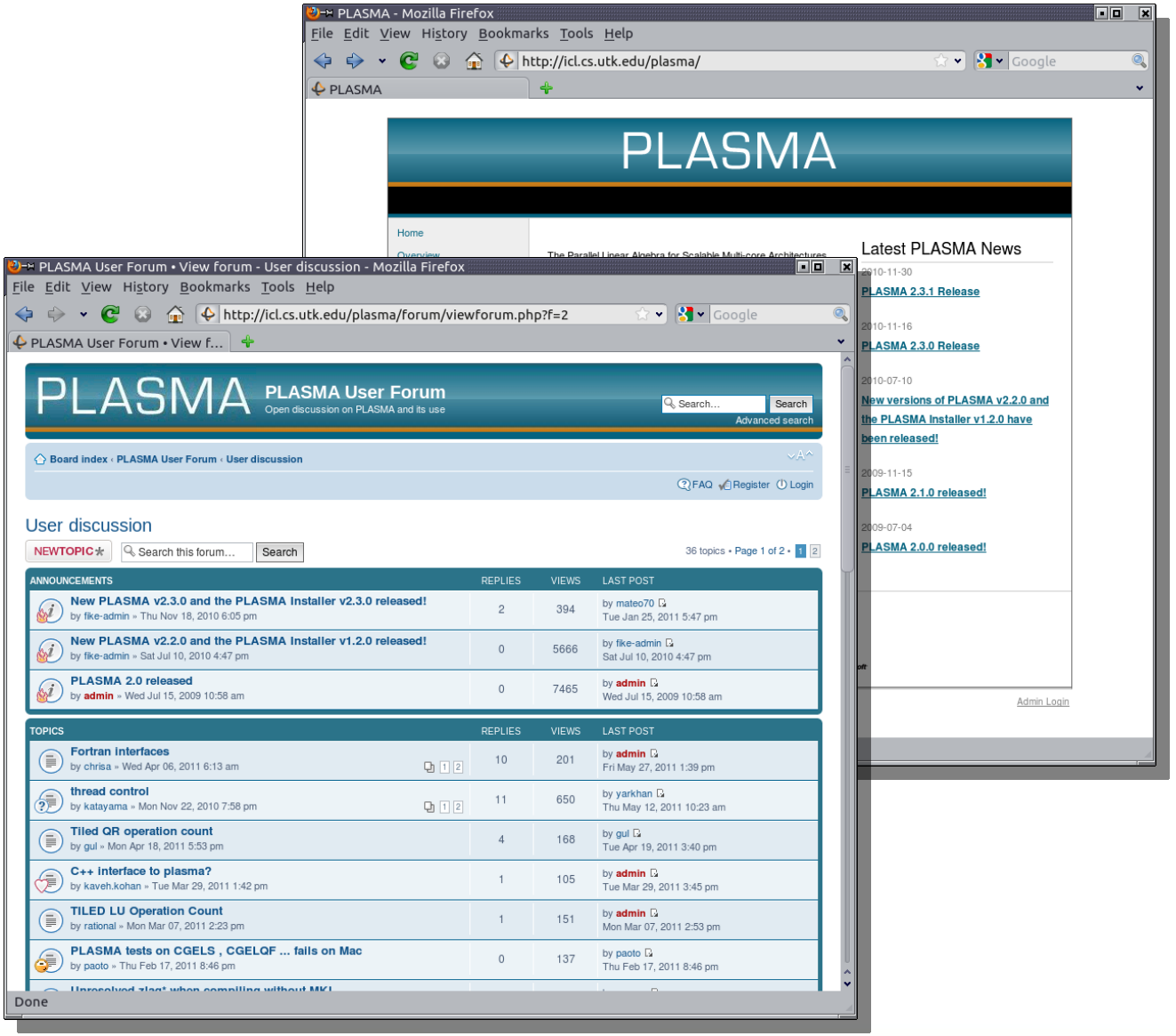
- Python scripts
- All precisions automatically generated
  - Z → D, C, S
  - ZC → DS (mixed precision)
  - one mechanism covers all scenarios
- Easily extendable
- Intended to also cover testing, timing, autotuning

PLASMA Software Engineering: Documentation Generation

- Doxygen comments used for all source code
- LAPACK-like Doxygen comments used for API routines
  - general description
  - parameters
  - return values
  - “see also” section
- Automatic generation of online documentation
  - grouping of routines by precision (Z, C, D, S, ZC, DS)
  - grouping of routines by interface (standard, tile, tile async)

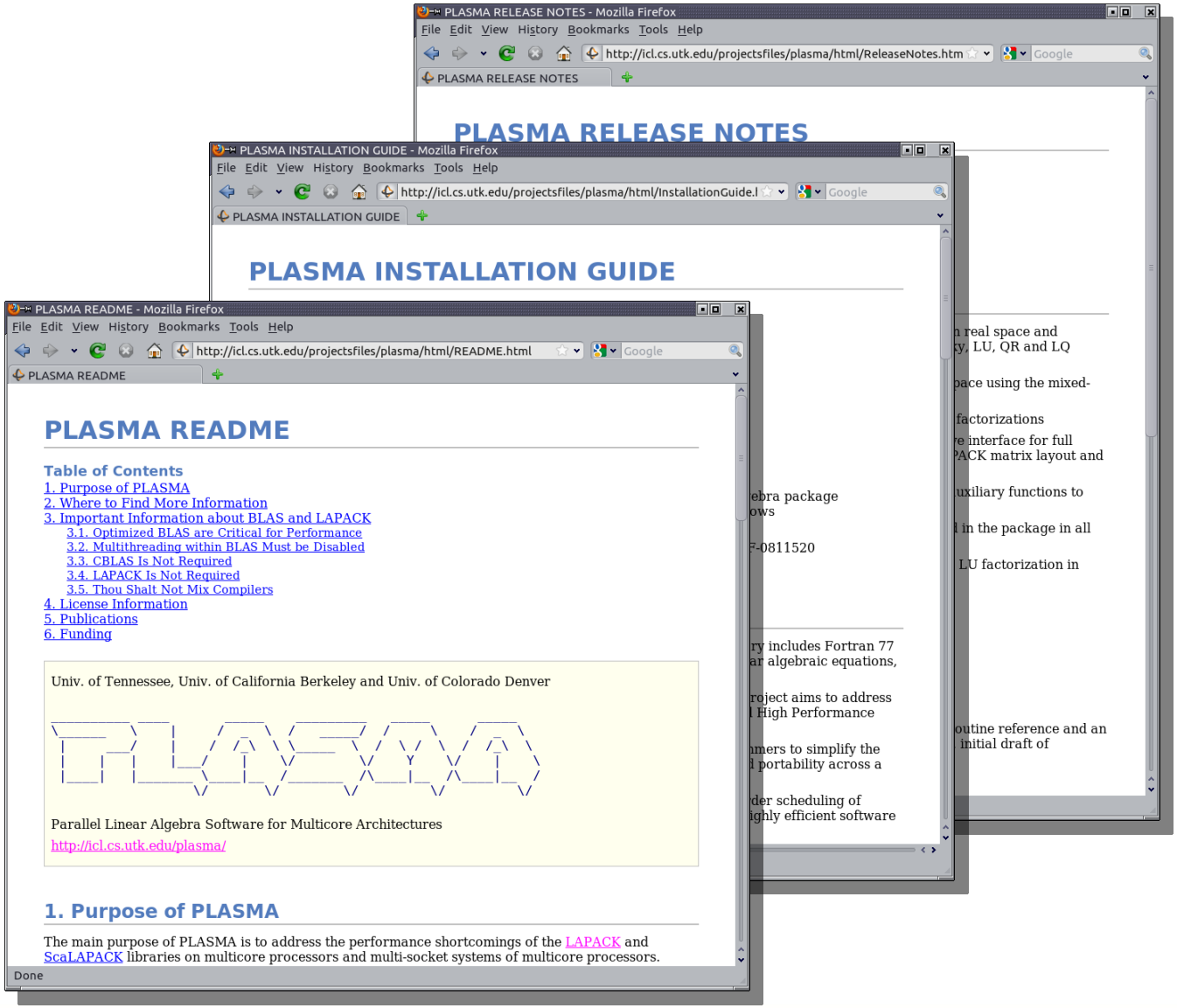
PLASMA Documentation Website

- Home
- Overview
- News
- **Software**
- Publications
- Links
- People
- **Documentation**
- **User Forum**

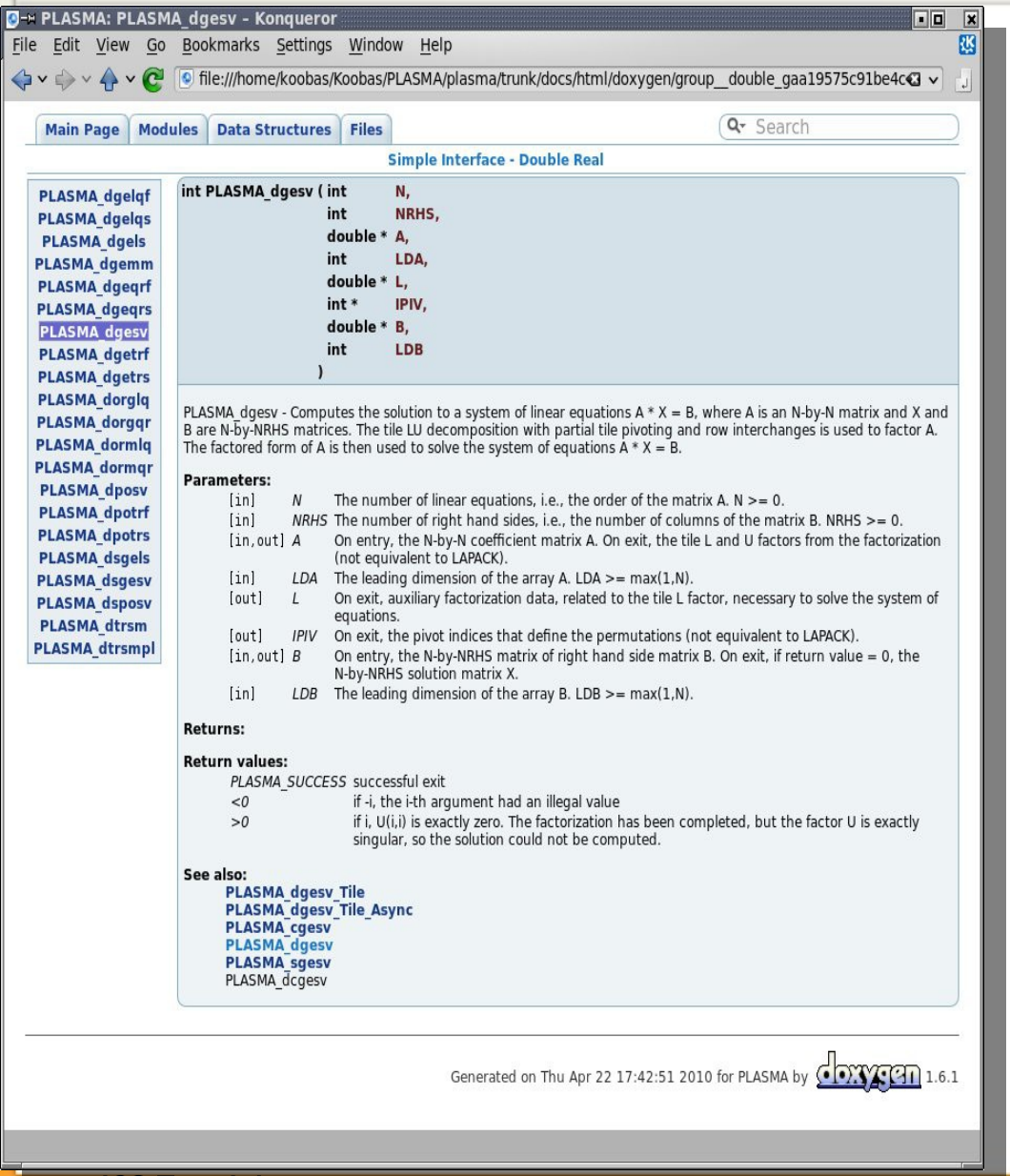


PLASMA Documentation: ASCII

- README
- LICENSE
- Release Notes
- Installation Guide
- Cmake Build Notes

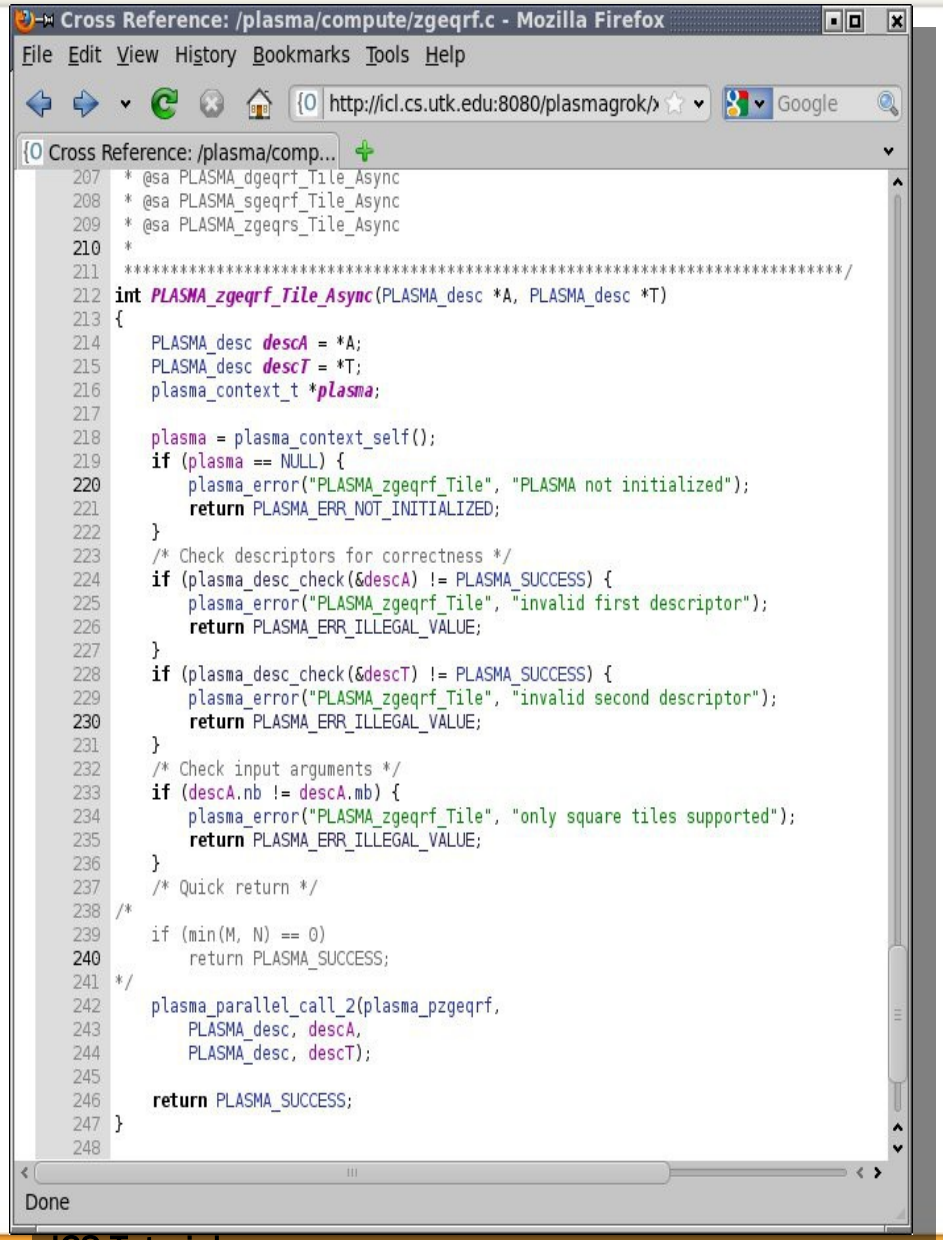


PLASMA Documentatio: HTML Reference



- Grouping by precision
  - Z, C, D, C
  - ZC, DS
- Grouping by interface
  - standard
  - tile
  - tile Async
- “see also” section

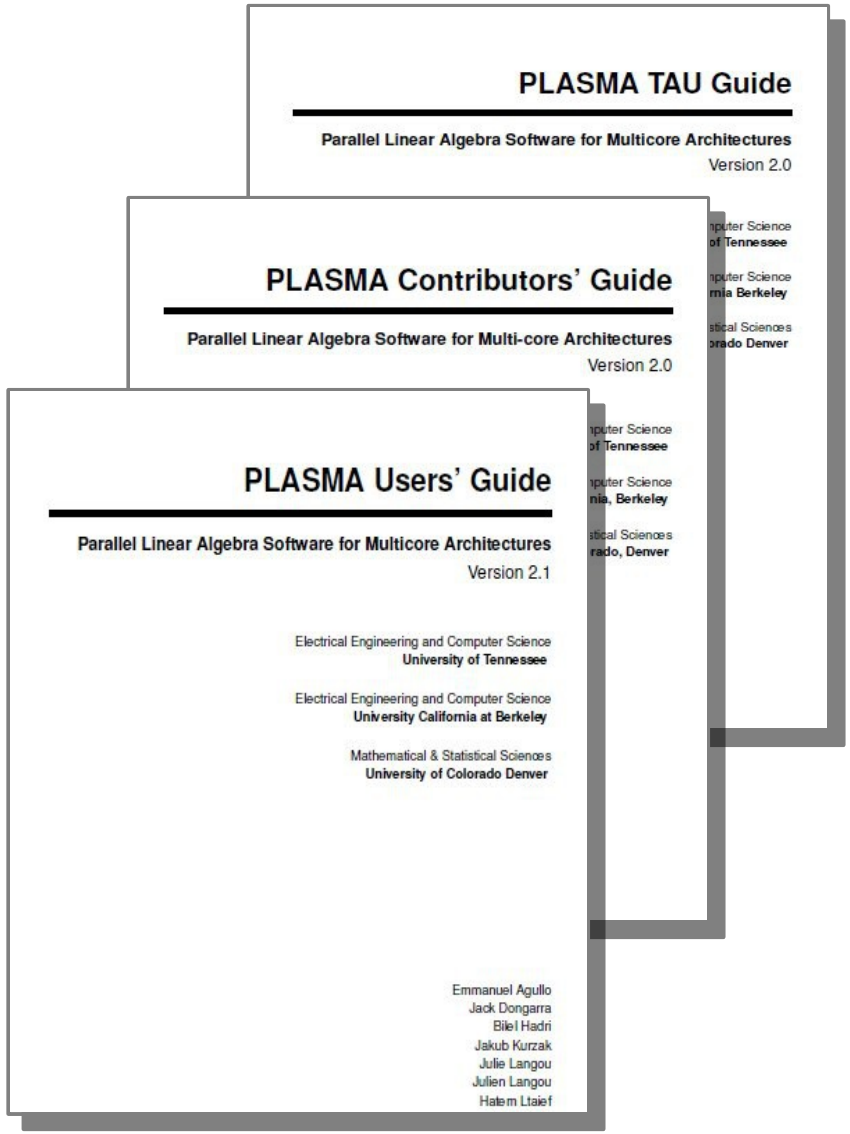
PLASMA Documentation: Src Code Browser



- Syntax highlighting
- Clickable function calls
- All software layers available from Netlib
  - PLASMA
  - QUARK
  - core BLAS
  - LAPACK C wrapper
  - LAPACK
  - Netlib CBLAS
  - Netlib BLAS

PLASMA Documentation: PDF Manuals

- PLASMA Users' Guide
- QUARK Users' Guide
- PLASMA Contributors' Guide
- PLASMA TAU Guide



Installation: Python Installer

- Downloads and installs PLASMA
  - No need to download the PLASMA tarball, just the installer script.
- Downloads and installs all dependencies
  - CBLAS,
  - LAPACK,
  - LAPACKE C API.
- The installer can also download and install Netlib BLAS.
- However,
  - Netlib BLAS is unoptimized (reference implementation), and
  - Will deliver very low performance.

Installation Python Installer

-h or --help

destination directories

compilers

compilation flags

linking flags

build tool

locations of libraries

force download

```
./setup.py  --prefix=""          [DIR]       (./install)
            --build=""           [DIR]       (./build)

            --cc=""              [CMD]       (cc)
            --fc=""              [CMD]       (gfortran)

            --cflags=""          [FLAGS]     (-O2)
            --fflags=""          [FLAGS]     (-O2)

            --ldflags_c=""[FLAGS]
            --ldflags_fc=""      [FLAGS]

            --make=              [CMD]       (make)

            --blaslib=""         [LIB]
            --cblaslib="" [LIB]
            --lapacklib=""[LIB]
            --lapclib=""         [LIB]

            --downblas
            --downcblas
            --downlapack
            --downlapc
            --downall

            --[no]testing                 (testing)
            --nbcores=           [#CORES]    (half)
            --clean
            --src
```

Installation Example

```
./setup.py
```

using default settings

```
./setup.py --prefix="/opt"
           --cc=icc
           --fc=ifort
           --blaslib="-L/intel/mkl/lib/em64t
                      -lmkl_intel_lp64
                      -lmkl_sequential
                      -lmkl_core"
           --downlapack
           --downcblas
```

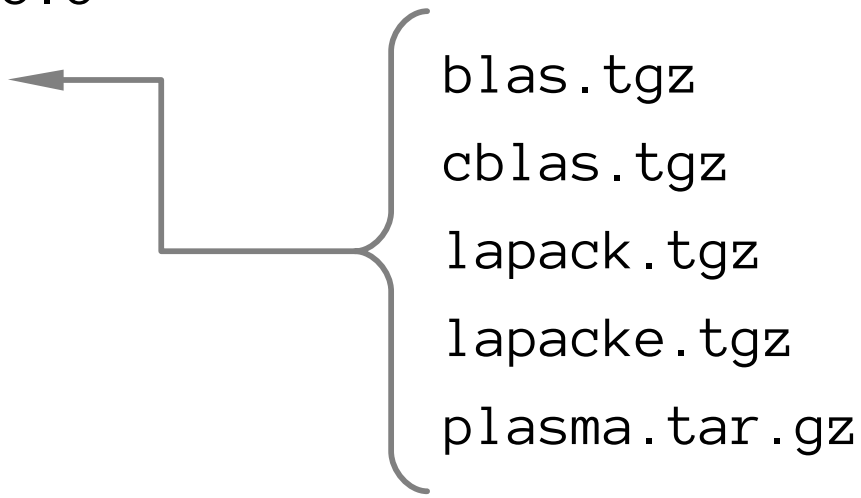
using custom settings

- install libraries in /opt
- use Intel compilers (ICC & IFORT)
- use MKL (EM64T)
- use LAPACK from Netlib
- use CBLAS from Netlib

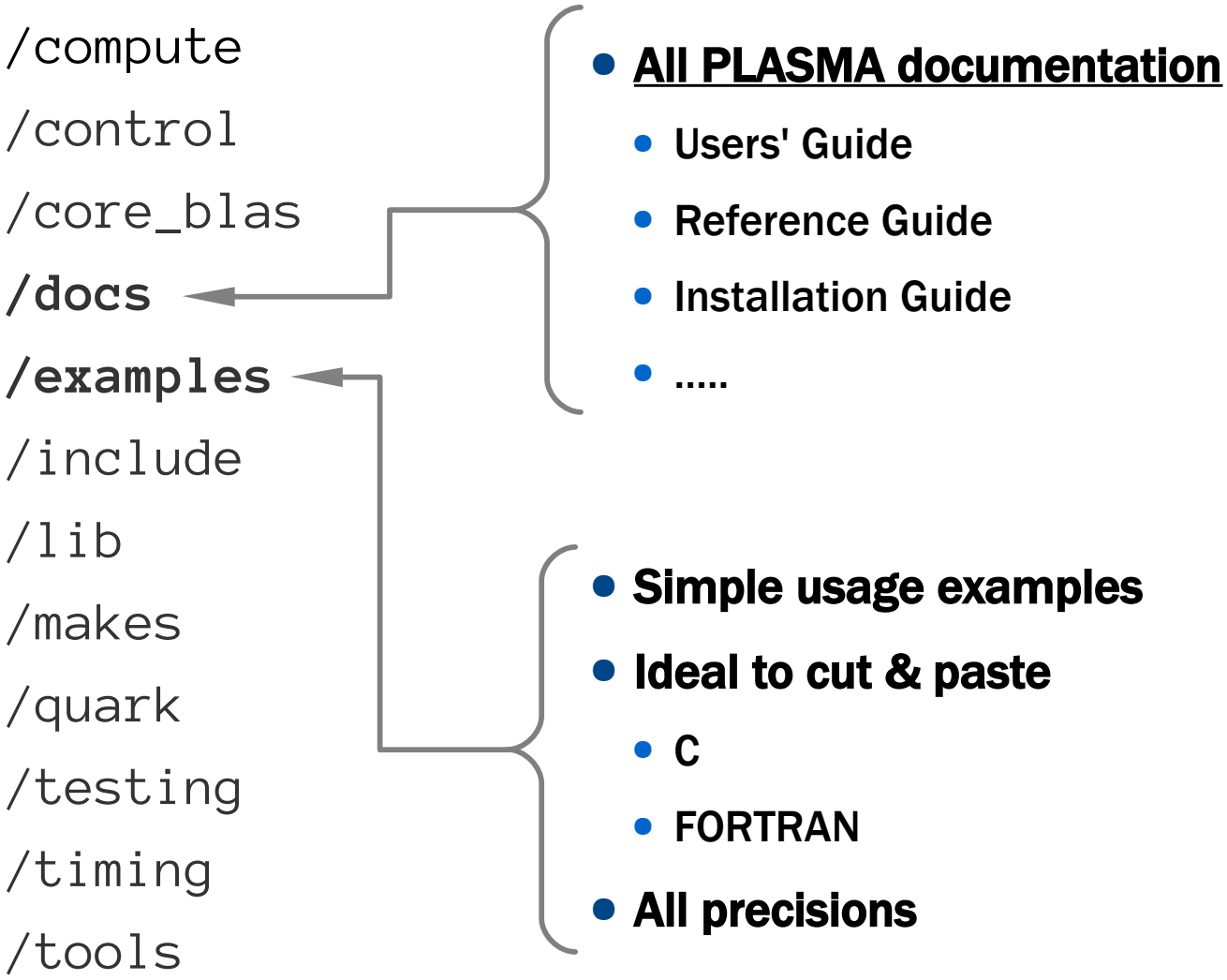
Installation Outcome

```
/build
  /BLAS
  /CBLAS
  /lapack-3.4.0
  /lapacke-3.4.0
  /plasma-2.5.0
  /download
  /log
/install
  /include
  /lib
```

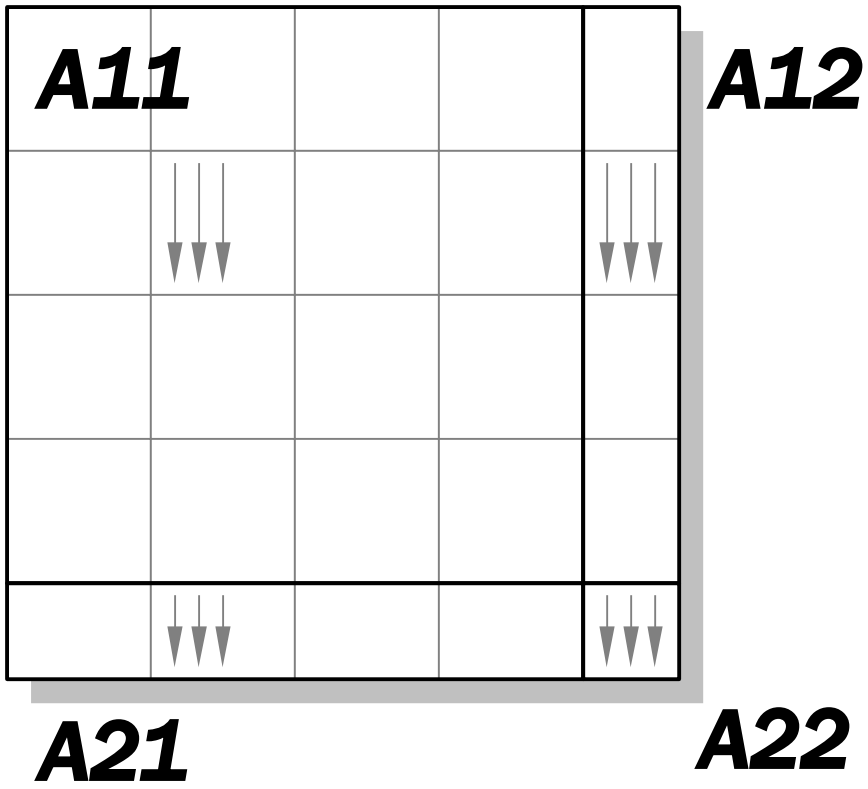
```
./setup.py --downall
```



PLASMA Directory Structure

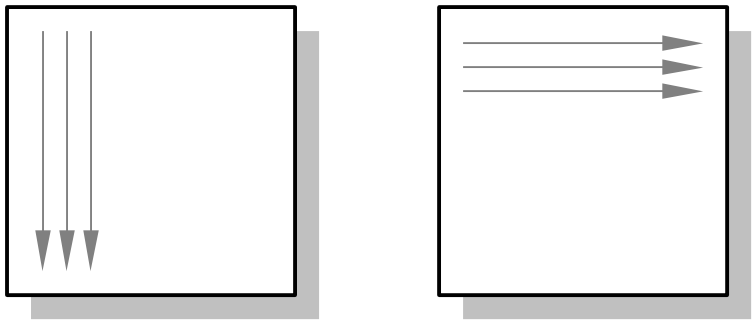


Matrix Layout: PLASMA Native Layout



- One continuous chunk of memory containing (in order): A11, A21, A12, A22
- Each tile is continuous (column-major order)
- Tiles are stored in column-major order (A11)

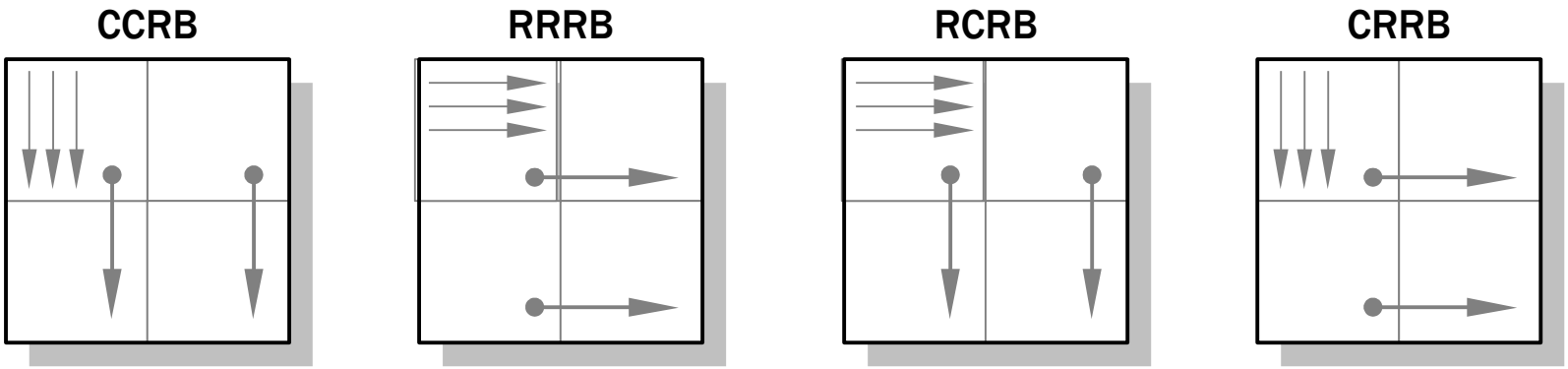
Matrix Layout: In-Place Translations



Column Major / Row Major

```
PLASMA_sgecfi()
PLASMA_cgecfi()
PLASMA_dgecfi()
PLASMA_zgecfi()
```

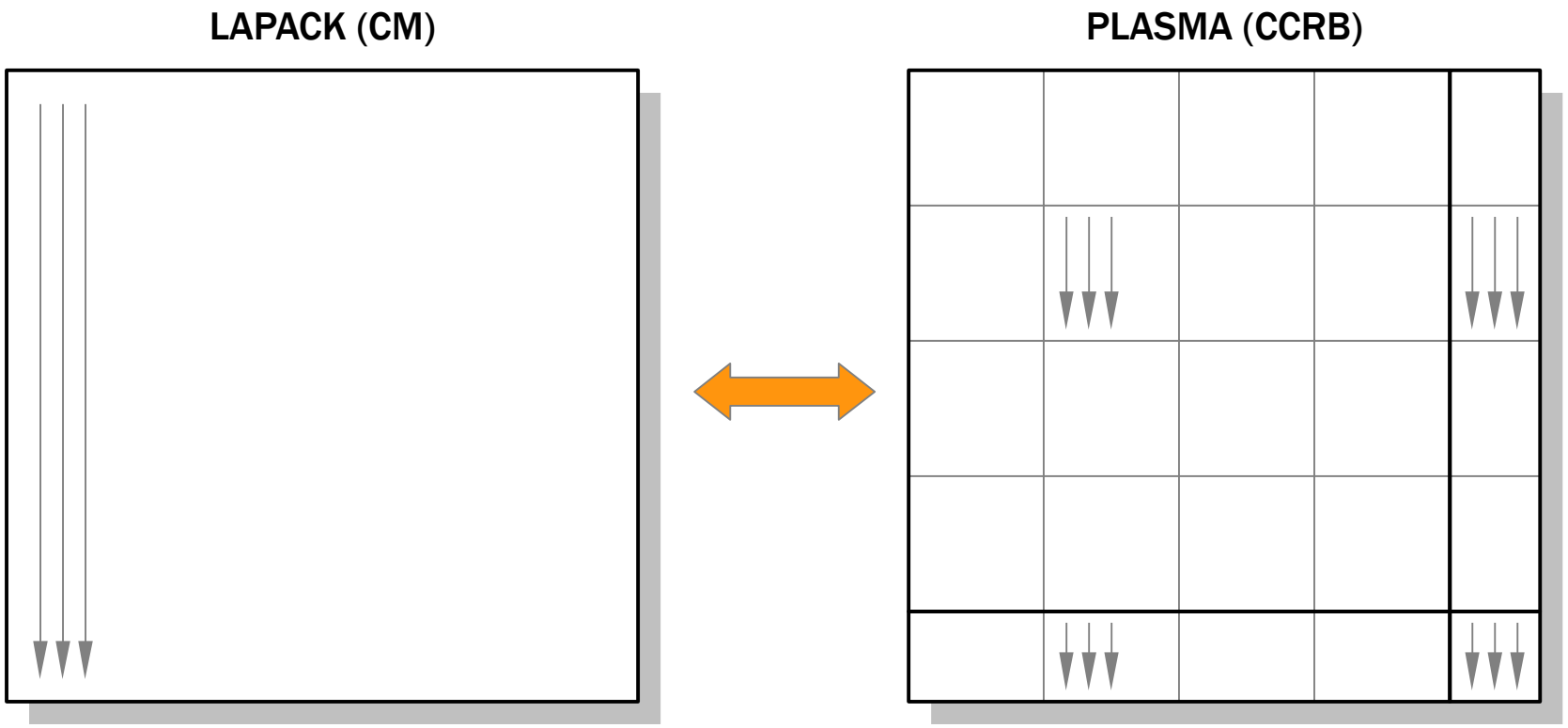
C|R C|R Rectangular Block



- PLASMA provides translation between each two layouts
- PLASMA only accepts input matrices in the CM and CCRB layouts

  (square tiles only)

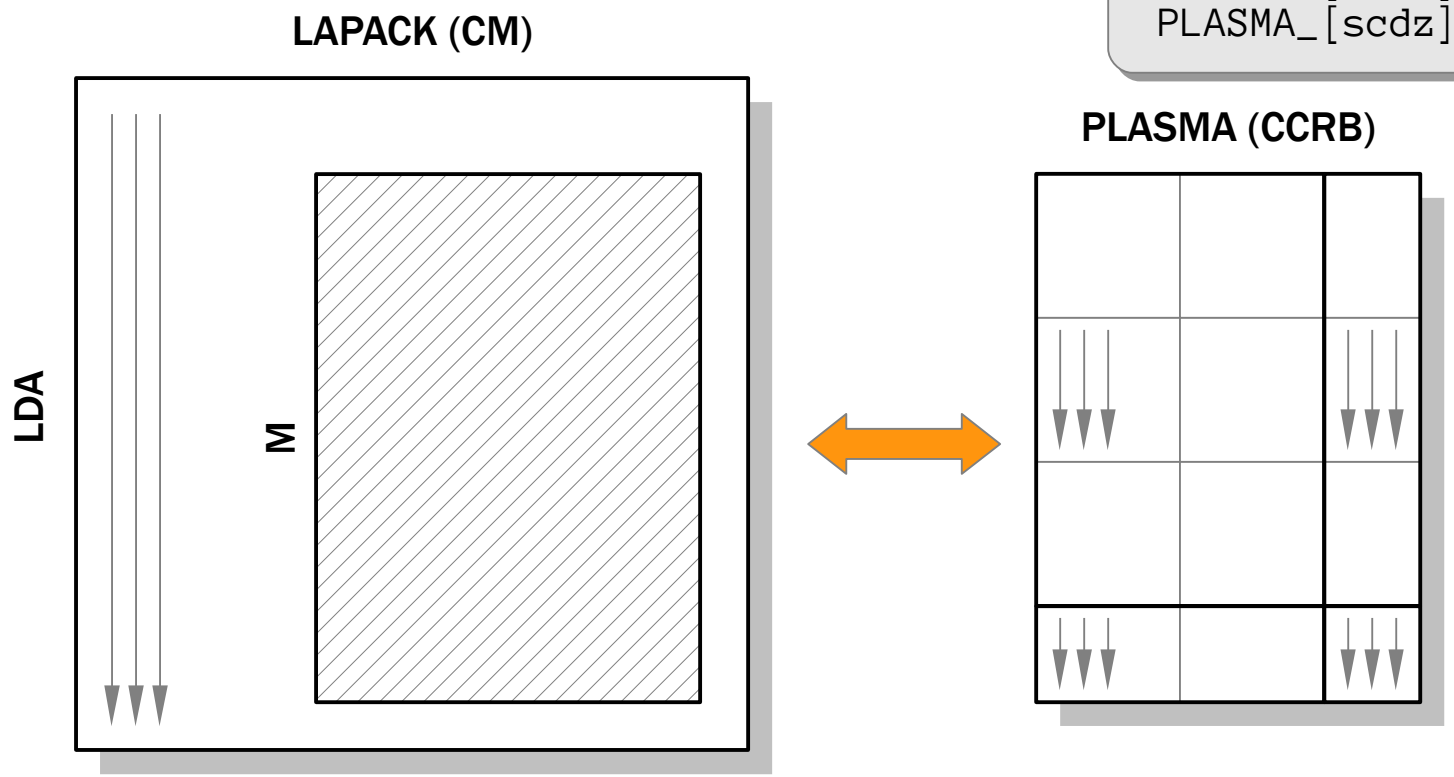
Layout Translation: LAPACK ↔ PLASMA



- **Tile layout & LAPACK layout accepted**
- **LAPACK layout always converted to tile layout (CCRB / square tiles)**
  - in place translation
  - out of place translation

Layout Translation: Out-of-Place Translation

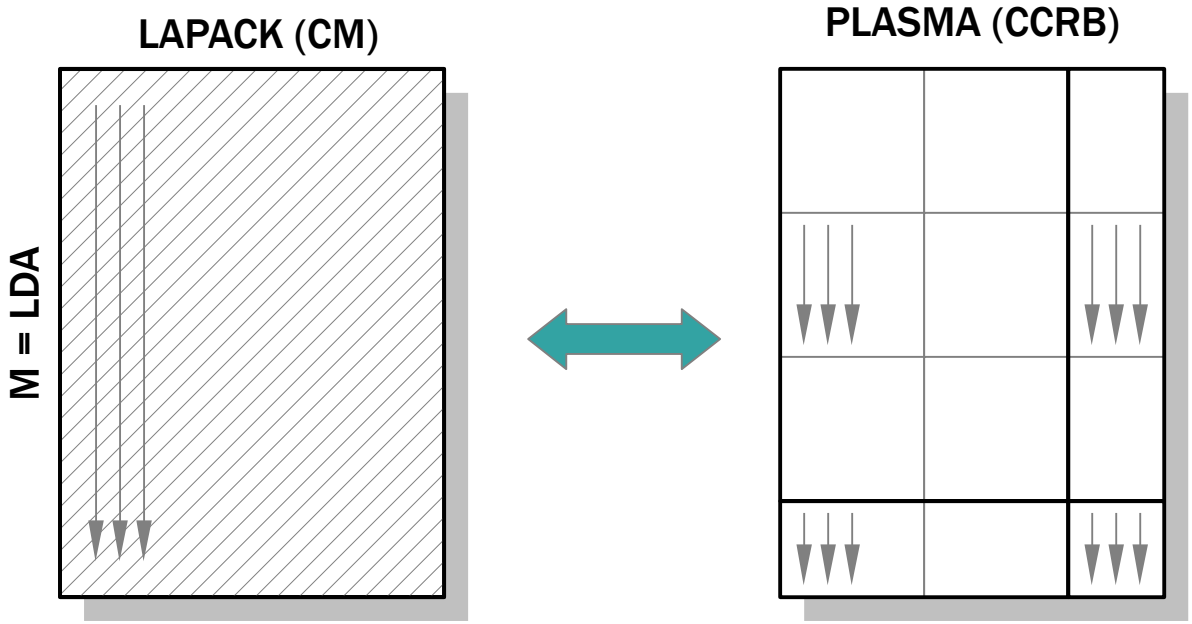
```
PLASMA_[scdz]Lapack_to_Tile()
PLASMA_[scdz]Tile_to_Lapack()
```



- **Out-of-Place Translation**
  - M <= LDA
  - CM ↔ CCRB only
  - storage overhead
  - fast (parallel & cache efficient)

Layout Translation: In-Place Translation

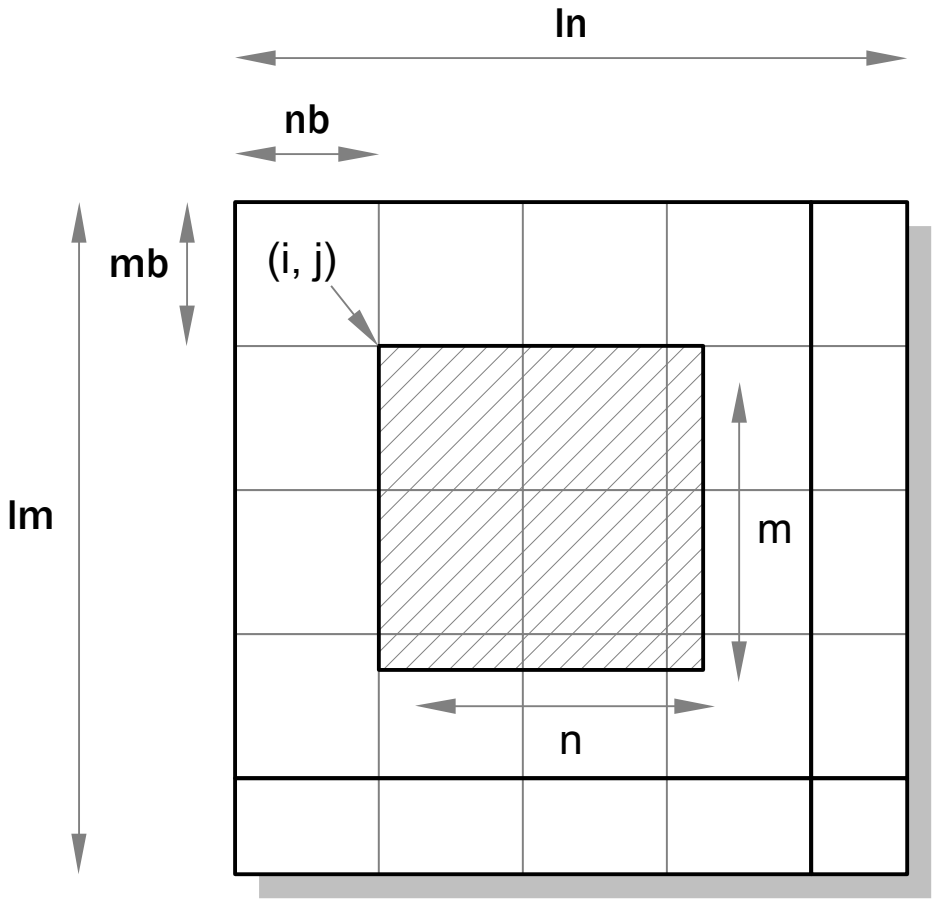
```
PLASMA_[scdz]gecfi()
```



- **In-Place Translation**
  - M = LDA
  - any layout
  - no storage overhead
  - fast (parallel & cache efficient)

Tile Layout: Matrix Descriptor

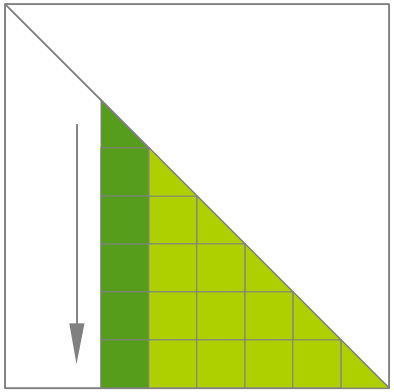
```
PLASMA_Desc_Create()
PLASMA_Desc_Destroy()
```



- **Tile Interface**
  - tiles have to be square (mb = nb)
  - submatrix has to be tile-aligned

    (i%mb = 0, j%nb = 0)

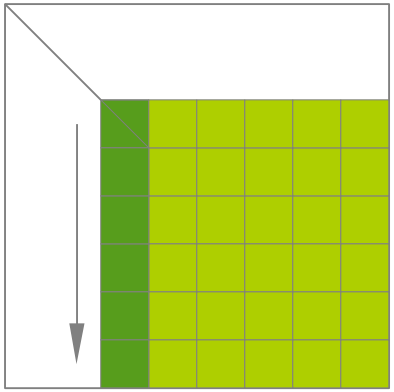
Algorithms: Tile Cholesky



```
PLASMA_[scdz]potrf[_Tile][_Async]()
```

- **Algorithm**
  - equivalent to LAPACK
- **Numerics**
  - same as LAPACK
- **Performance**
  - comparable to vendor on few cores
  - much better than vendor with many cores

Algorithms: LU with Partial Pivoting

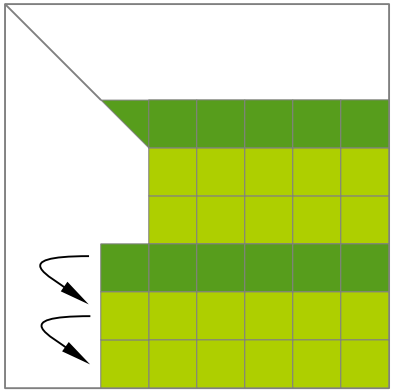


```
PLASMA_[scdz]getrf[_Tile][_Async]()
```

- **Algorithm**
  - equivalent to LAPACK
  - same pivot vector
  - same L and U factors
  - same forward substitution procedure
  - parallel recursive panel factorization
- **Numerics**
  - same as LAPACK
- **Performance**
  - comparable to vendor on few cores
  - much better than vendor with many cores

Algorithms:QR & “Domino” Panel Reduction

```
PLASMA_[scdz]geqrt[_Tile][_Async]()
```

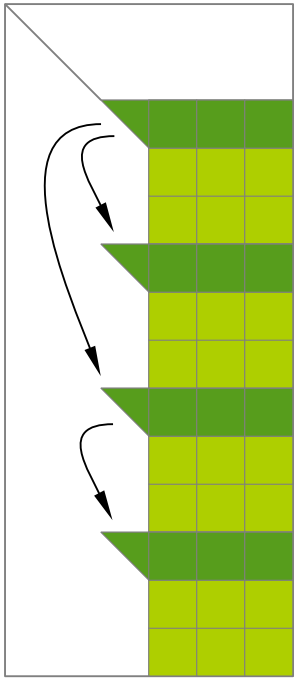


- **Algorithm**
  - the same R factor as LAPACK (absolute values)
  - different set of Householder reflectors
  - different Q matrix
  - different Q generation / application procedure
- **Numerics**
  - same as LAPACK
- **Performance**
  - comparable to vendor on few cores
  - much better than vendor with many cores

Algorithms: QR with "Tree" Panel Reduction

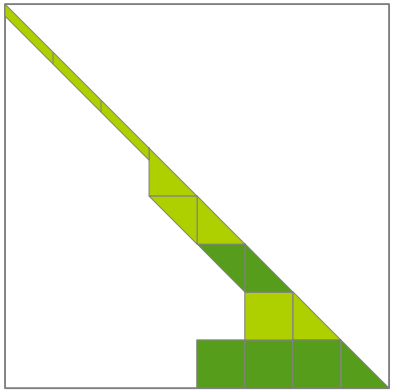
```
PLASMA_[scdz]geqrt[_Tile][_Async]()
```

```
PLASMA_Set(
    PLASMA_HOUSEHOLDER_MODE,
    PLASMA_TREE_HOUSEHOLDER);
```



- **Algorithm**
  - the same R factor as LAPACK (absolute values)
  - different set of Householder reflectors
  - different Q matrix
  - different Q generation / application procedure
- **Numerics**
  - same as LAPACK
- **Performance**
  - absolutely superior for tall matrices

Algorithms: Symmetric Eigenvalue Problem



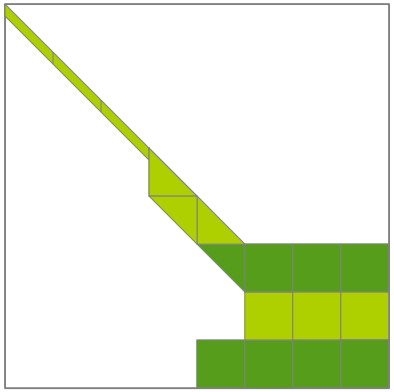
```
PLASMA_[scdz]syev[_Tile][_Async]()
```

- **Algorithm**
  - two-stage tridiagonal reduction + QR iteration
  - fast eigenvalues, slower eigenvectors

    (possibility to calculate a subset)
- **Numerics**
  - same as LAPACK
- **Performance**
  - comparable to MKL for very small problems
  - absolutely superior for larger problems

Algorithms: Singular Value Decomposition

```
PLASMA_[scdz]gesvd[_Tile][_Async]()
```



- **Algorithm**
  - two-stage bidiagonal reduction + QR iteration
  - fast singular values, slower singular vectors

    (possibility of calculating a subset)
- **Numerics**
  - same as LAPACK
- **Performance**
  - comparable with MKL for very small problems
  - absolutely superior for larger problems

Basic Routines: Initialization & Configuration

`PLASMA_Init()`	– initialize PLASMA
`PLASMA_Finalize()`	– finalize PLASMA
`PLASMA_Set()`	– set a parameter
`PLASMA_Get()`	– get the value of a parameter
`PLASMA_Enable()`	– enable a feature
`PLASMA_Disable()`	– disable a feature
`PLASMA_Version()`	– get PLASMA version number

Computational Routines: Linear Systems

`PLASMA_dgesv()`	– solve a linear system using the LU fact.
`PLASMA_dgetrf()`	– LU factorization
`PLASMA_dgetrs()`	– forward and backward substitution
`PLASMA_dtrsm()`	– triangular solve
`PLASMA_dposv()`	– solve a linear system using the Cholesky fact.
`PLASMA_dpotrf()`	– Cholesky factorization
`PLASMA_dpotrs()`	– forward and backward substitution
`PLASMA_dtrsm()`	– triangular solve
`PLASMA_dtrtri()`	– compute an inverse of an SPD matrix

Computational Routines: Over- (Least Squares), Underdetermined (Min. Norm)

`PLASMA_dgels()` – solve the problem using the QR or LQ fact.

`PLASMA_dgeqrf()` – QR factorization

`PLASMA_dgelqf()` – LQ factorization

`PLASMA_dormqr()` – multiply by the Q matrix (QR)

`PLASMA_dormlq()` – multiply by the Q matrix (LQ)

`PLASMA_dorgqr()` – generate the Q matrix (QR)

`PLASMA_dorglq()` – generate the Q matrix (LQ)

`PLASMA_dtrsm()` – triangular solve

Computational Routines: Symmetric Eigenvalues, Singular Values

`PLASMA_dgesvd()` – find singular values

`PLASMA_dsyev()` – find eigenvalues (symmetric system)

`PLASMA_dsygv()` – find generalized eigenvalues (positive definite)

Computational Routines: Tile BLAS 3 – Real

`PLASMA_dgemm()` – matrix multiply

`PLASMA_dsymm()` – symmetric matrix multiply

`PLASMA_dsyr2k()` – symmetric rank 2k update

`PLASMA_dsyrk()` – symmetric rank k update

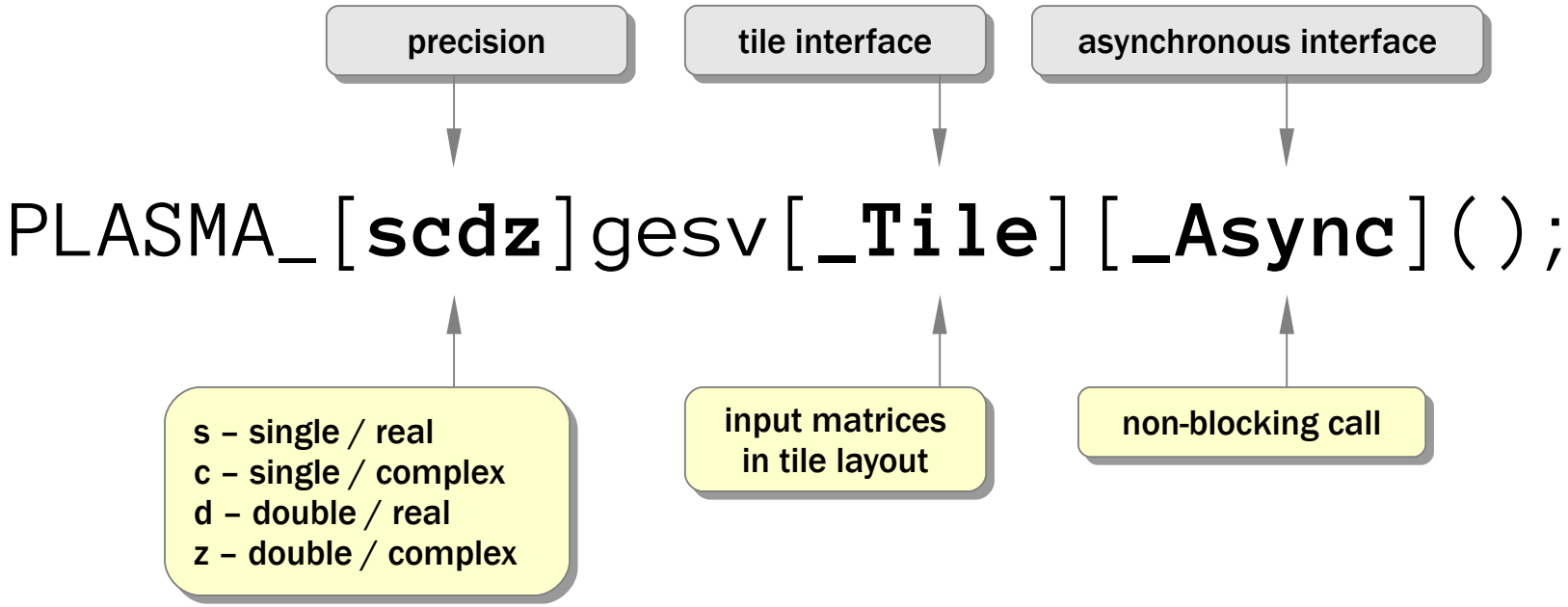
`PLASMA_dtrmm()` – triangular matrix multiply

`PLASMA_dtrsm()` – triangular solve

Computational Routines:Tile BLAS3 Complex

`PLASMA_zgemm()`	– matrix multiply
`PLASMA_zsymm()`	– symmetric matrix multiply
`PLASMA_zhemm()`	– Hermitian matrix multiply
`PLASMA_zsyr2k()`	– symmetric rank 2k update
`PLASMA_zher2k()`	– Hermitian rank 2k update
`PLASMA_zsyrk()`	– symmetric rank k update
`PLASMA_zherk()`	– Hermitian rank k update
`PLASMA_ztrmm()`	– triangular matrix multiply
`PLASMA_ztrsm()`	– triangular solve

Computational Routines: Precisions & Interfaces



Computational Routines: Precisions / Interfaces

```
PLASMA_sgesv();
PLASMA_cgesv();
PLASMA_dgesv();
PLASMA_zgesv();
```

LAPACK Interface

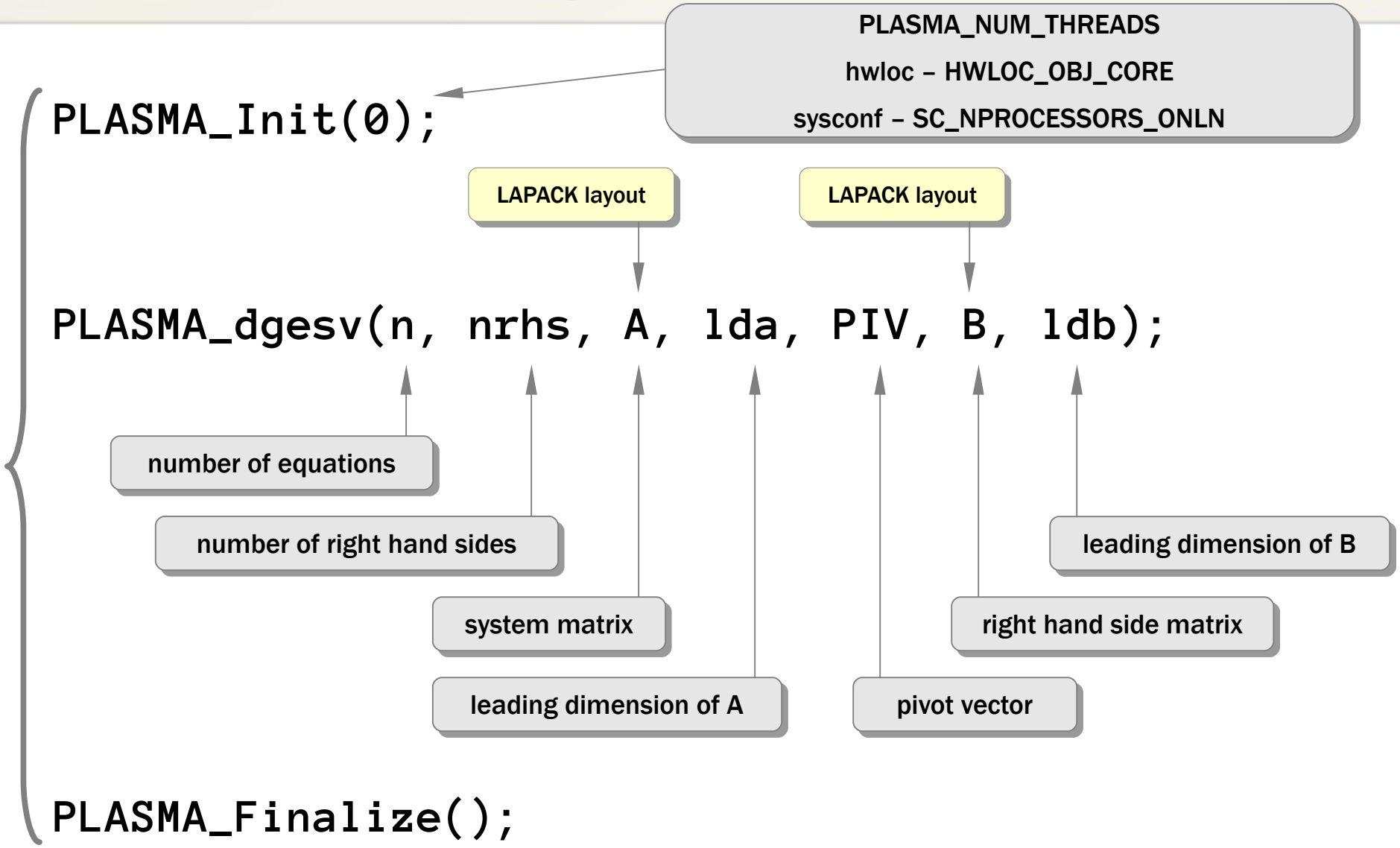
```
PLASMA_sgesv_Tile();
PLASMA_cgesv_Tile();
PLASMA_dgesv_Tile();
PLASMA_zgesv_Tile();
```

Tile Layout

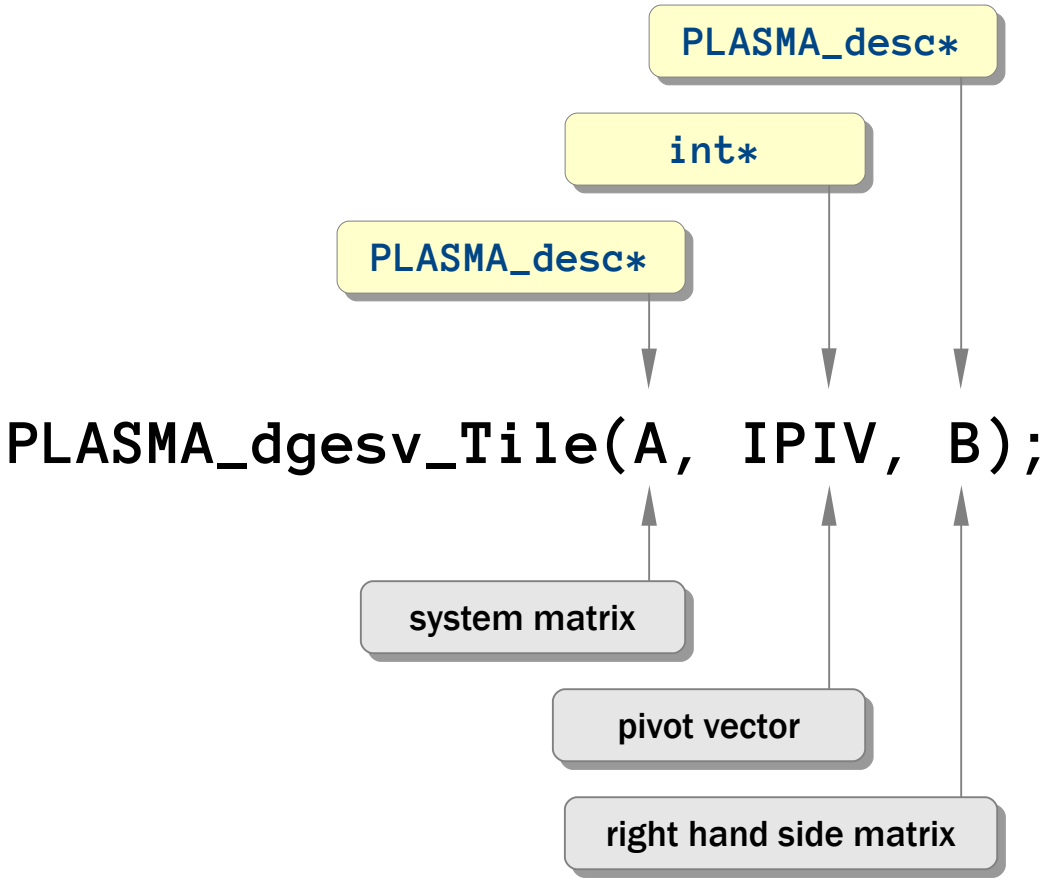
```
PLASMA_sgesv_Tile_Async();
PLASMA_cgesv_Tile_Async();
PLASMA_dgesv_Tile_Async();
PLASMA_zgesv_Tile_Async();
```

Tile, No blocking

Hello World; LAPACK Interface



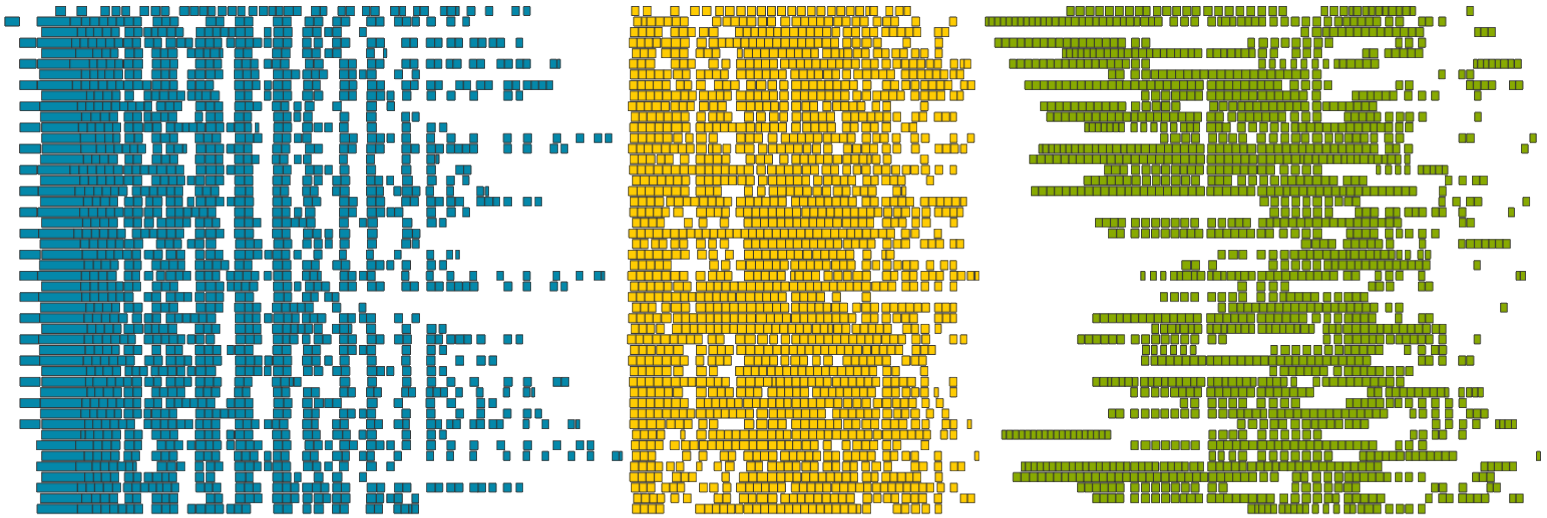
Tile Interface: Linear System



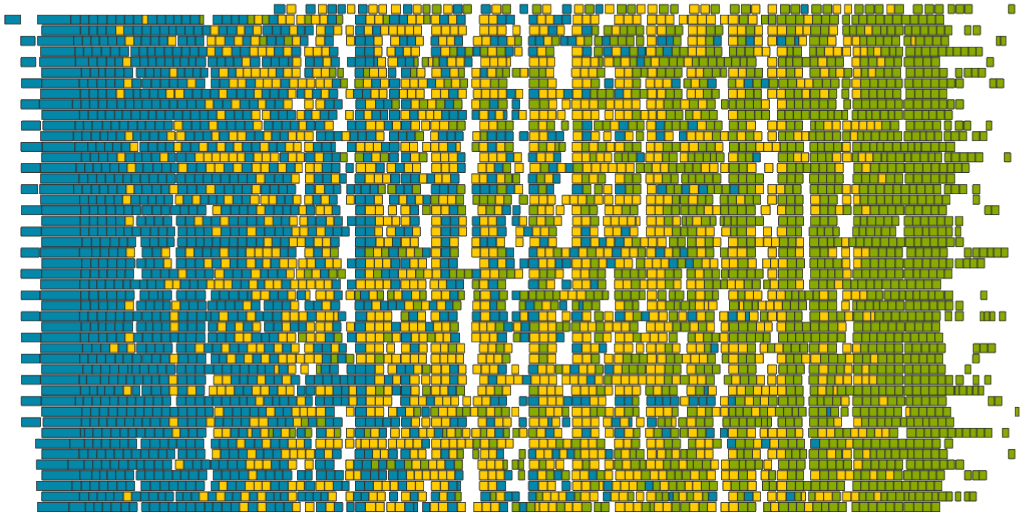
Tile Interface: Linear System

```
PLASMA_Init(0);

PLASMA_Desc_Create(&descA, ...);
PLASMA_Desc_Create(&descB, ...);

PLASMA_dLapack_to_Tile(A, lda, descA);
PLASMA_dLapack_to_Tile(B, ldb, descB);

PLASMA_dgesv_Tile(descA, IPIV, descB);

PLASMA_Desc_Destroy(&descA);
PLASMA_Desc_Destroy(&descB);

PLASMA_Finalize();
```

Tile Asynch Interface: Motivation



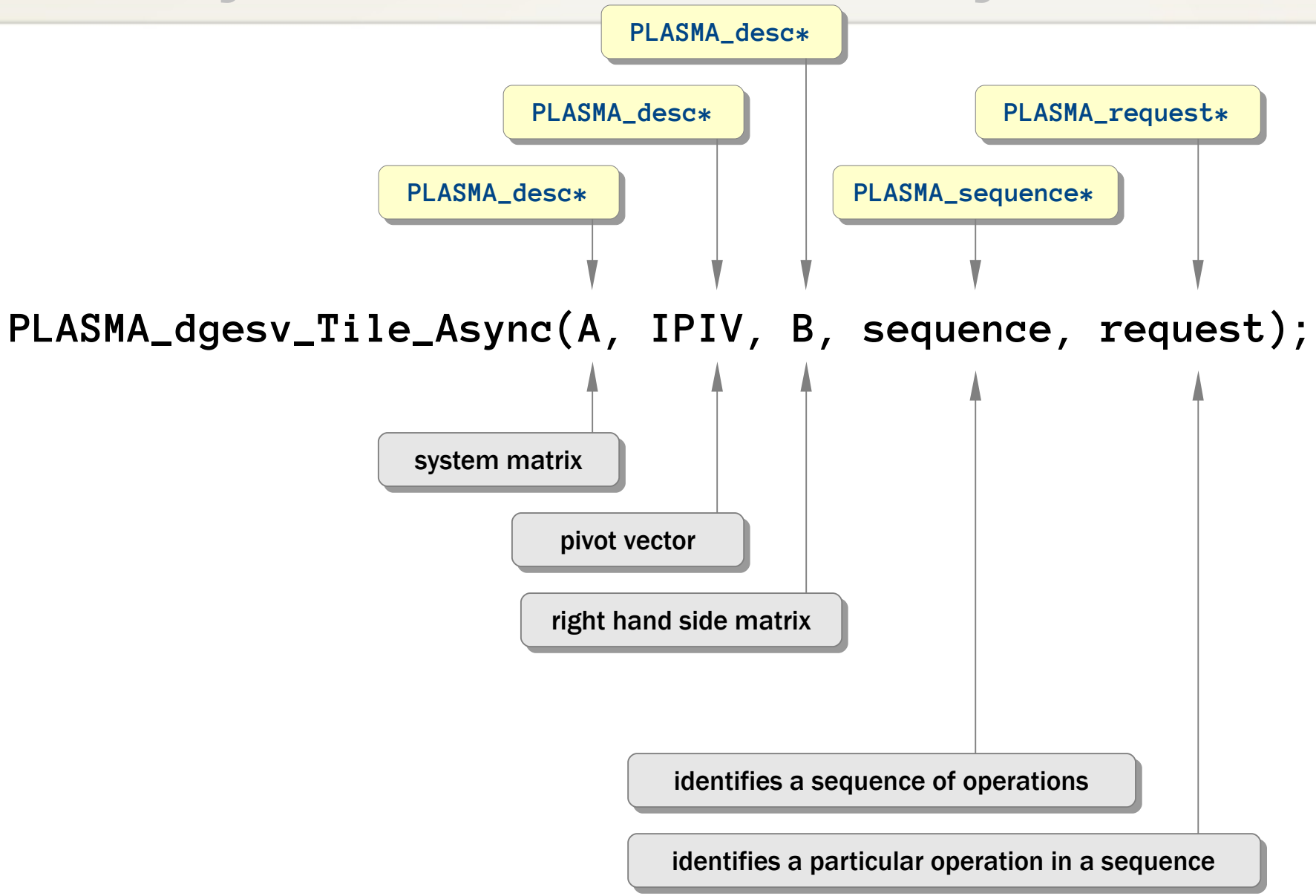
synchronous



asynchronous

POTRI:
POTRF
TRTRI
LAUUM

Tile Async Interface: Linear System



Tile Async Interface: Linear System

```
PLASMA_Init(0);

PLASMA_sequence *sequence = NULL;
PLASMA_request request1 = PLASMA_REQUEST_INITIALIZER;
PLASMA_request request2 = PLASMA_REQUEST_INITIALIZER;

PLASMA_Sequence_Create(&sequence);

PLASMA_dgetrf_Tile_Async(..., sequence, &request1);
PLASMA_dgetrs_Tile_Async(..., sequence, &request2);
PLASMA_Sequence_Wait(sequence);

PLASMA_Sequence_Destroy(sequence);

PLASMA_Finalize();
```

Tile Async Interface: Error Handling

- IF NO Errors
  - sequence->status == PLASMA_SUCCESS
- IF Error
  - Entire sequence is canceled,
  - including tasks queued but not yet executed
  - as well as tasks not yet queued
  - sequence->status returns the error code
  - sequence->requestreturns the failed request

Workspace: Allocation

for auxiliary data that need to be passed from one routine to another

```
int M;
int N;
PLASMA_desc *descT;

PLASMA_Alloc_Workspace_dgels (M, N, &descT);
PLASMA_Alloc_Workspace_dgeqrf(M, N, &descT);
PLASMA_Alloc_Workspace_dgelqf(M, N, &descT);

PLASMA_Alloc_Workspace_dgels_Tile (M, N, &descT);
PLASMA_Alloc_Workspace_dgeqrf_Tile(M, N, &descT);
PLASMA_Alloc_Workspace_dgelqf_Tile(M, N, &descT);
```

Workspaces: Deallocation

```
PLASMA_Init(0);

PLASMA_Alloc_Workspace_dgels(m, n, &descT);

PLASMA_dgeqrf(m, n, A, lda, descT);
PLASMA_dgeqrs(m, n, nrhs, A, lda, descT, B, ldb);

PLASMA_Dealloc_Handle(&descT);

PLASMA_Finalize();
```

LAPACK interface

```
PLASMA_Init(0);

PLASMA_Alloc_Workspace_dgeqrf_Tile(m, n, &descT);

PLASMA_dgeqrf_Tile(descA, descT);
PLASMA_dgeqrs_Tile(descA, descT, descB);

PLASMA_Dealloc_Handle_Tile(&descT);

PLASMA_Finalize();
```

tile interface

Settings: Enable / Disable

- `PLASMA_Enable(int flag)`
- `PLASMA_Disable(int flag)`
  - `PLASMA_WARNINGS`
  - `PLASMA_ERRORS`
  - `PLASMA_AUTOTUNING` ← not a true autotuning (using hardcoded settings)

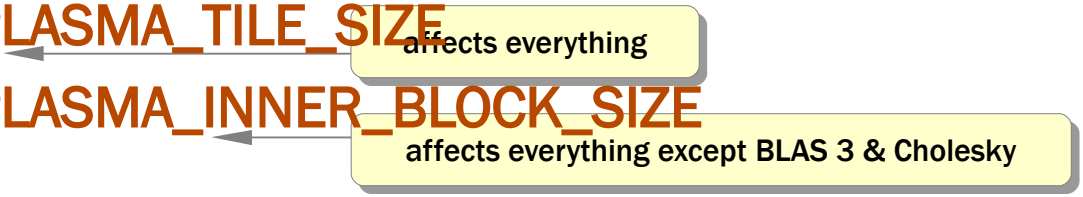
Call

```
PLASMA_Disable(PLASMA_AUTOTUNING);
```

to do manual tuning

Settings: Set / Get Blocking

- PLASMA_Set(int param, int value);
- PLASMA_Get(int param, int *value);
- param:
  - PLASMA_TILE_SIZE
  - PLASMA_INNER_BLOCK_SIZE



Settings: Set / Get Scheduling

- `PLASMA_Set(PLASMA_SCHEDULING_MODE, int mode);`
- `PLASMA_Get(PLASMA_SCHEDULING_MODE, int *mode);`
- `mode:`
  - `PLASMA_STATIC_SCHEDULING`
  - `PLASMA_DYNAMIC_SCHEDULING`

use static scheduling where static implementation exists

use dynamic scheduling where dynamic implementation exists

some routines only have static implementations

some routines only have dynamic implementations

some routines have both

PLASMA switches between the two modes at runtime

Settings: Set / Get Reductions

- `PLASMA_Set(PLASMA_HOUSEHOLDER_MODE, int mode)`
- `PLASMA_Get(PLASMA_HOUSEHOLDER_MODE, int *mode)`
- `mode:`
  - PLASMA_FLAT_HOUSEHOLDER ← use “domino” reduction (serial panel reduction)
  - PLASMA_TREE_HOUSEHOLDER ← use “tree” reduction (parallel panel reduction)

- `PLASMA_Set(PLASMA_HOUSEHOLDER_SIZE, int size)`
- `PLASMA_Get(PLASMA_HOUSEHOLDER_SIZE, int *size)`

size of the serial pieces of the panel in the tree reduction

Settings: Set / Get Translation

- `PLASMA_Set(PLASMA_TRANSLATION_MODE, int mode);`
- `PLASMA_Get(PLASMA_TRANSLATION_MODE, int *mode);`
- **mode:**
  - PLASMA_INPLACE
  - PLASMA_OUTOFPLACE