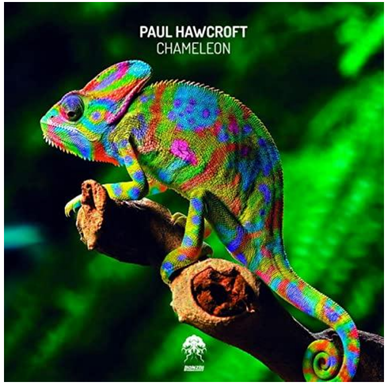
M. Rue M. Pruvost M. Faverge

# Validation and evaluation of the Chameleon Lapack interface

Second year internship

Alycia :)

# What is Chameleon ?




- Dense Linear Algebra Library
  - LAPACK (OpenBlas, Intel MKL) Multi-Thread
  - SCALAPACK (Netlib, Intel MKL) MPI
- Parallel → MPI, PThread, CUDA
- Task based

StarPU Parsec

OpenMP Quark

# Chameleon Algorithms

MM $C = \alpha\, A \times B + \beta\, C$

RK $C = \alpha\, A \times A^T + \beta\, C$

R2K $C = \alpha\, A \times B^T + \alpha\, A^T \times B + \beta\, C$

BLAS3

# Chameleon Algorithms

MM $C = \alpha A \times B + \beta C$

RK $C = \alpha A \times A^T + \beta C$

R2K $C = \alpha A \times B^T + \alpha A^T \times B + \beta C$

BLAS3

TRF

$A = L \times U$

$A = L \times L^T$

$A = U^T \times U$

CHOLESKY /
LU DECOMPOSITION

SV / TRS $A \times X = B$

TRI $A^{-1}$

# Chameleon Algorithms

## BLAS3

MM $C = \alpha\, A \times B + \beta\, C$

RK $C = \alpha\, A \times A^T + \beta\, C$

R2K $C = \alpha\, A \times B^T + \alpha\, A^T \times B + \beta\, C$

## QR / LQ Factorisation

QR

$A = Q \times R$

$A = Q \times R$

LQ

$A = L \times Q$

$A = L \times Q$

## Cholesky / LU Decomposition

TRF

$A = L \times U$

$A = L \times L^T$

$A = U^T \times U$

SV / TRS $A \times X = B$

TRI $A^{-1}$

# Chameleon Algorithms

## BLAS3

MM $C = \alpha\, A \times B + \beta\, C$

RK $C = \alpha\, A \times A^T + \beta\, C$

R2K $C = \alpha\, A \times B^T + \alpha\, A^T \times B + \beta\, C$

## QR / LQ Factorisation

QR

$A = Q \times R$

$A = Q \times R$

LQ

$A = L \times Q$

$A = L \times Q$

## Cholesky / LU Decomposition

TRF

$A = L \times U$

$A = L \times L^T$

$A = U^T \times U$

SV / TRS $A \times X = B$

TRI $A^{-1}$

## Singular Value Decomposition

SVD $A = U \times \Sigma \times V^T$

# Chameleon Algorithms

BLAS3

MM $C = \alpha\, A \times B + \beta\, C$

RK $C = \alpha\, A \times A^T + \beta\, C$

R2K $C = \alpha\, A \times B^T + \alpha\, A^T \times B + \beta\, C$

QR / LQ FACTORISATION

QR

$A = Q \times R$

$A = Q \times R$

LQ

$A = L \times Q$

$A = L \times Q$

CHOLESKY / LU DECOMPOSITION

TRF

$A = L \times U$

$A = L \times L^T$

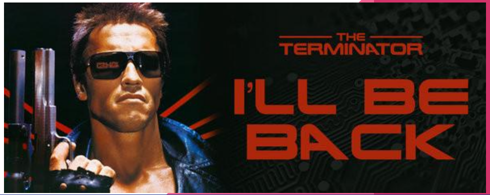
$A = U^T \times U$

SV / TRS $A \times X = B$

TRI $A^{-1}$

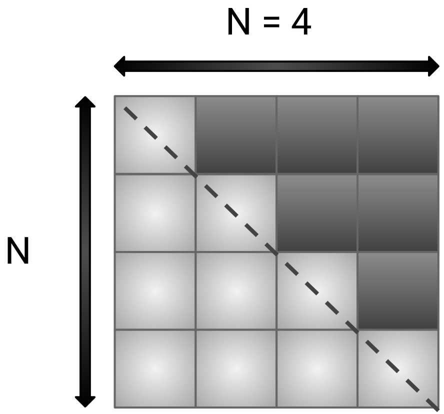
SINGULAR VALUE DECOMPOSITION

SVD



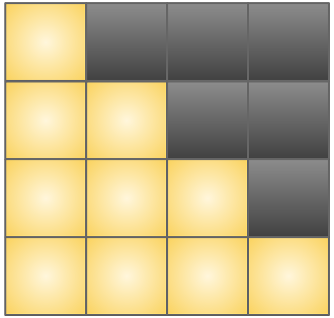
# Task-based Algorithm: POTRF

```
Chameleon_potrf( A )
  for j = 0 to N-1 do
    potrf( A_RW[j][j] )
    for i = j+1 to N-1 do
      trsm( A_RW[i][j], A_R[j][j] )
    done
    for i = j+1 to N-1 do
      syrk( A_RW[i][i], A_R[i][j] )
      for k = j+1 to i-1 do
        gemm( A_RW[i][k], A_R[i][j], A_R[k][j] )
      done
    done
  done
```



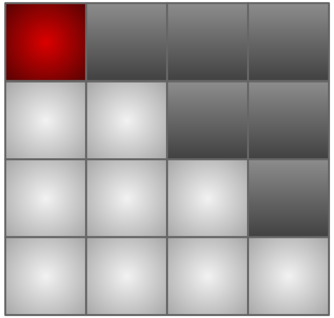
# Task-based Algorithm: POTRF

```
Chameleon_potrf( A )
  for j = 0 to N-1 do
    potrf( A_RW[j][j] )
    for i = j+1 to N-1 do
      trsm( A_RW[i][j], A_R[j][j] )
    done
    for i = j+1 to N-1 do
      syrk( A_RW[i][i], A_R[i][j] )
      for k = j+1 to i-1 do
        gemm( A_RW[i][k], A_R[i][j], A_R[k][j] )
      done
    done
  done
```

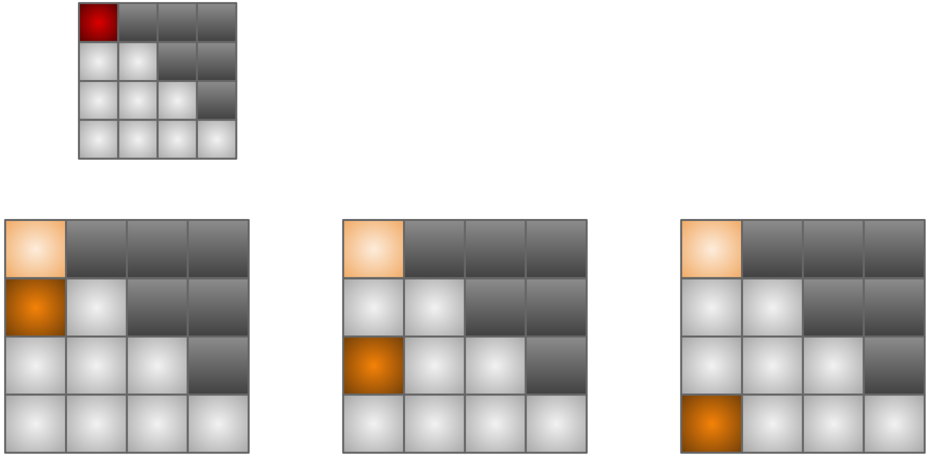


# Task-based Algorithm: POTRF

```
Chameleon_potrf( A )
  for j = 0 to N-1 do
    potrf( A_RW[j][j] )
    for i = j+1 to N-1 do
      trsm( A_RW[i][j], A_R[j][j] )
    done
    for i = j+1 to N-1 do
      syrk( A_RW[i][i], A_R[i][j] )
      for k = j+1 to i-1 do
        gemm( A_RW[i][k], A_R[i][j], A_R[k][j] )
      done
    done
  done
```

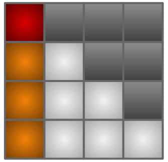


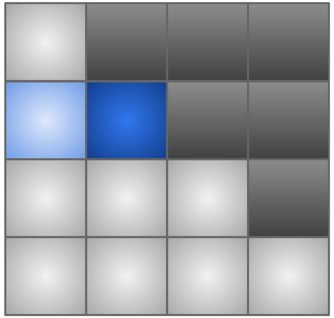
# Task-based Algorithm: POTRF



```
Chameleon_potrf( A )
  for j = 0 to N-1 do
    potrf( A_RW[j][j] )
    for i = j+1 to N-1 do
      trsm( A_RW[i][j], A_R[j][j] )
    done
    for i = j+1 to N-1 do
      syrk( A_RW[i][i], A_R[i][j] )
      for k = j+1 to i-1 do
        gemm( A_RW[i][k], A_R[i][j], A_R[k][j] )
      done
    done
  done
```

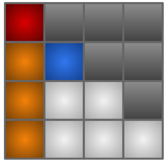
# Task-based Algorithm: POTRF



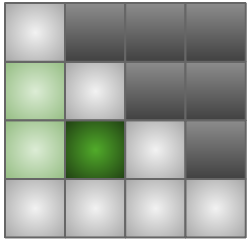


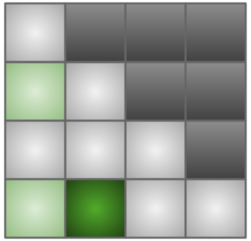
```
Chameleon_potrf( A )
  for j = 0 to N-1 do
    potrf( A_RW[j][j] )
    for i = j+1 to N-1 do
      trsm( A_RW[i][j], A_R[j][j] )
    done
    for i = j+1 to N-1 do
      syrk( A_RW[i][i], A_R[i][j] )
      for k = j+1 to i-1 do
        gemm( A_RW[i][k], A_R[i][j], A_R[k][j] )
      done
    done
  done
```

# Task-based Algorithm: POTRF

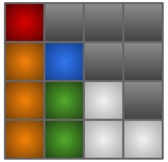


```
Chameleon_potrf( A )
  for j = 0 to N-1 do
    potrf( A_RW[j][j] )
    for i = j+1 to N-1 do
      trsm( A_RW[i][j], A_R[j][j] )
    done
    for i = j+1 to N-1 do
      syrk( A_RW[i][i], A_R[i][j] )
      for k = j+1 to i-1 do
        gemm( A_RW[i][k], A_R[i][j], A_R[k][j] )
      done
    done
  done
```

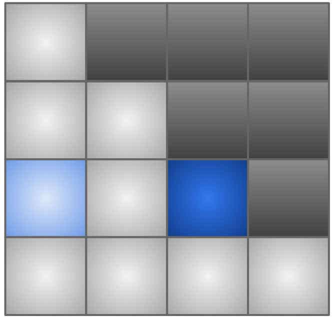




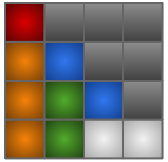
# Task-based Algorithm: POTRF

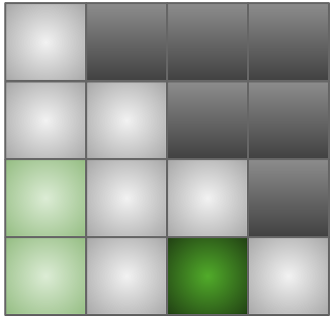


```
Chameleon_potrf( A )
  for j = 0 to N-1 do
    potrf( A_RW[j][j] )
    for i = j+1 to N-1 do
      trsm( A_RW[i][j], A_R[j][j] )
    done
    for i = j+1 to N-1 do
      syrk( A_RW[i][i], A_R[i][j] )
      for k = j+1 to i-1 do
        gemm( A_RW[i][k], A_R[i][j], A_R[k][j] )
      done
    done
  done
```



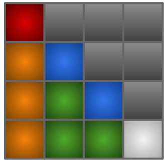
# Task-based Algorithm: POTRF



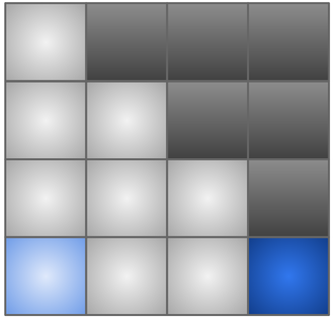


```
Chameleon_potrf( A )
  for j = 0 to N-1 do
    potrf( A_RW[j][j] )
    for i = j+1 to N-1 do
      trsm( A_RW[i][j], A_R[j][j] )
    done
    for i = j+1 to N-1 do
      syrk( A_RW[i][i], A_R[i][j] )
      for k = j+1 to i-1 do
        gemm( A_RW[i][k], A_R[i][j], A_R[k][j] )
      done
    done
  done
```

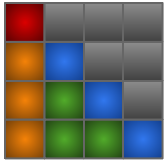
# Task-based Algorithm: POTRF



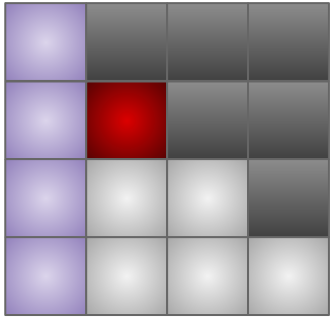
```
Chameleon_potrf( A )
  for j = 0 to N-1 do
    potrf( A_RW[j][j] )
    for i = j+1 to N-1 do
      trsm( A_RW[i][j], A_R[j][j] )
    done
    for i = j+1 to N-1 do
      syrk( A_RW[i][i], A_R[i][j] )
      for k = j+1 to i-1 do
        gemm( A_RW[i][k], A_R[i][j], A_R[k][j] )
      done
    done
  done
```



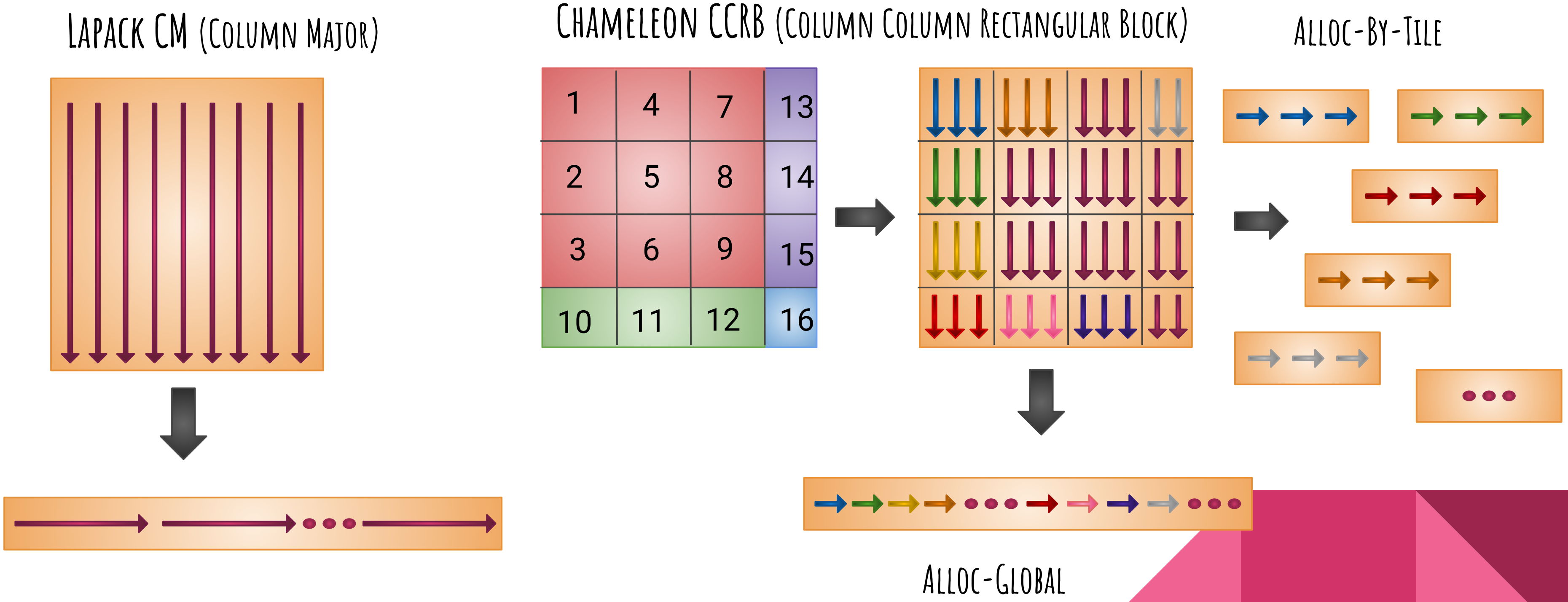
# Task-based Algorithm: POTRF



```
Chameleon_potrf( A )
  for j = 0 to N-1 do
    potrf( A_RW[j][j] )
    for i = j+1 to N-1 do
      trsm( A_RW[i][j], A_R[j][j] )
    done
    for i = j+1 to N-1 do
      syrk( A_RW[i][i], A_R[i][j] )
      for k = j+1 to i-1 do
        gemm( A_RW[i][k], A_R[i][j], A_R[k][j] )
      done
    done
  done
```

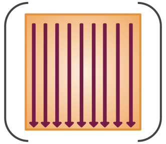


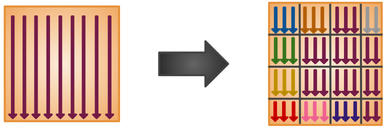
# Chameleon Matrix Descriptor

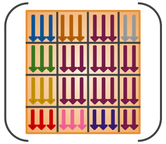


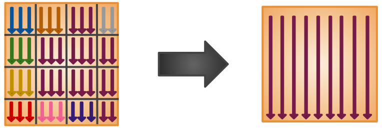
# So what did I do ?

# Chameleon Interfaces

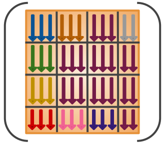
- Chameleon_zalgo 

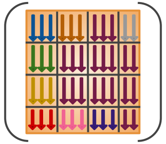


Chameleon_zalgo_Tile_Async 

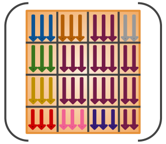


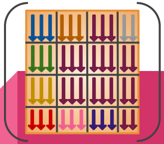
Synchronisation

- Chameleon_zalgo_Tile 

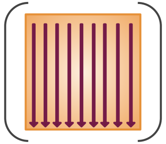
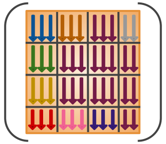
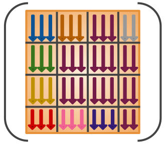
Chameleon_zalgo_Tile_Async 

Synchronisation

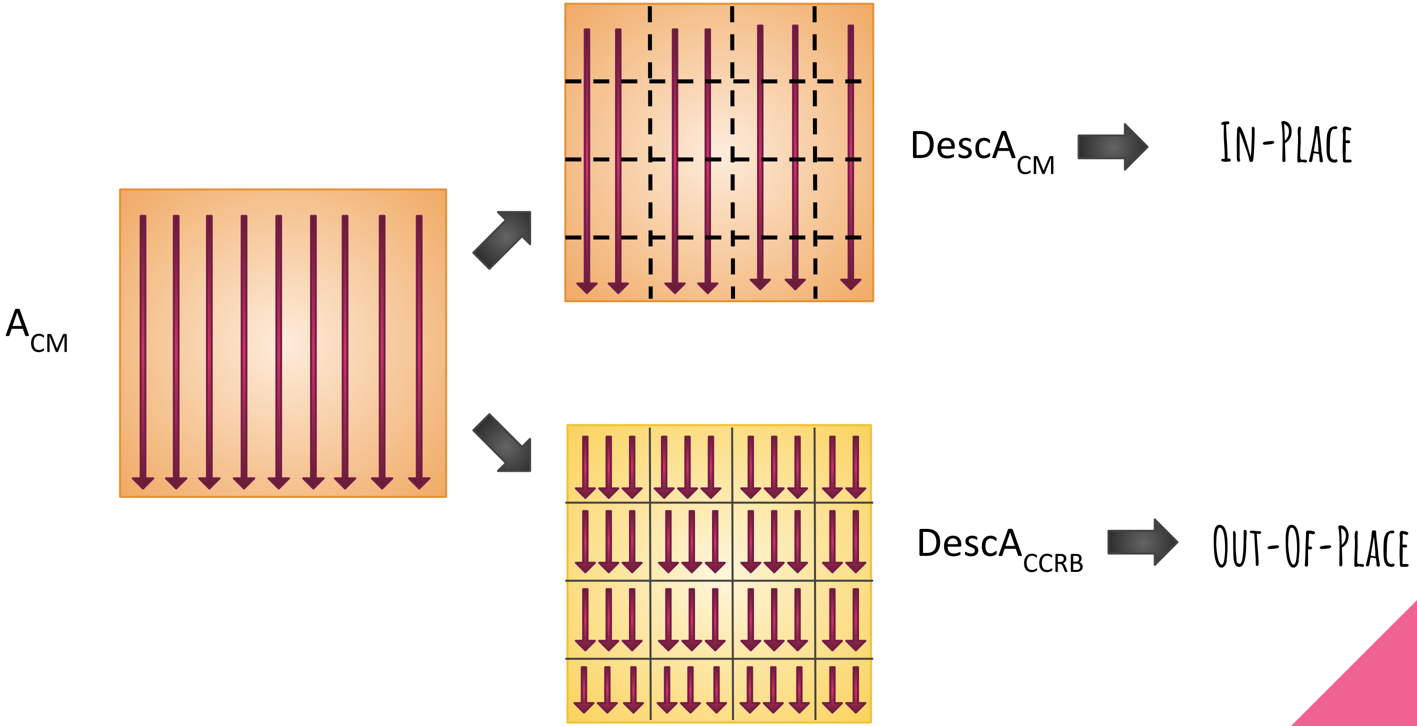
- Chameleon_zalgo_Tile_Async 

chameleon_pzalgo 

# Chameleon Interfaces

- `Chameleon_zalgo`  → No tests or checks made
- `Chameleon_zalgo_Tile`  → Tested and checked since 2020 (commit n. 166) by L. Barros de Assis during his internship
- `Chameleon_zalgo_Tile_Async`  → Tested indirectly since 2020 by Lucas
  Tested directly since 2022
  (commit n. 262)

# Conversion In-Place vs Out-Of-Place

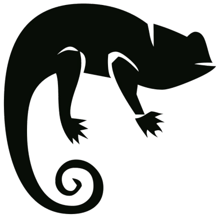


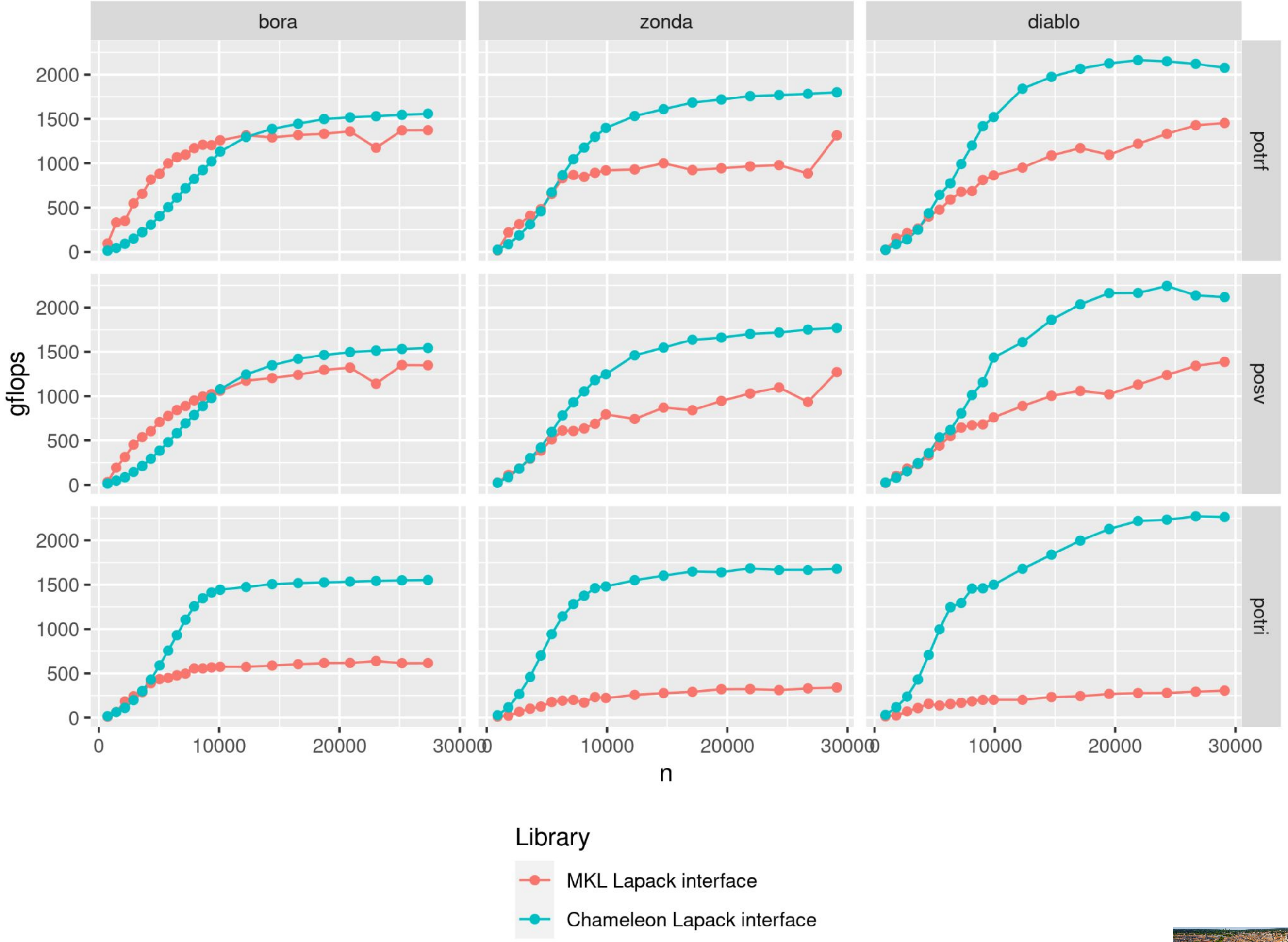
# Performances: machines used

| | Number of Cores | Type of Core |
|---|---|---|
| Bora | 2x 18 | Intel CascadeLake |
| Zonda | 2x 32 | AMD Zen2 |
| Diablo | 2x 64 | AMD Zen3 |

# Performances

Chameleon



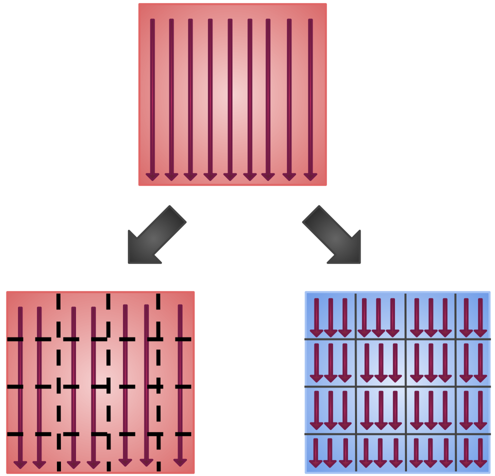



L A P A C K
L -A P -A C -K
L A P A -C -K
L -A P -A -C K
L A -P -A C K
L -A -P A C -K

Lapack

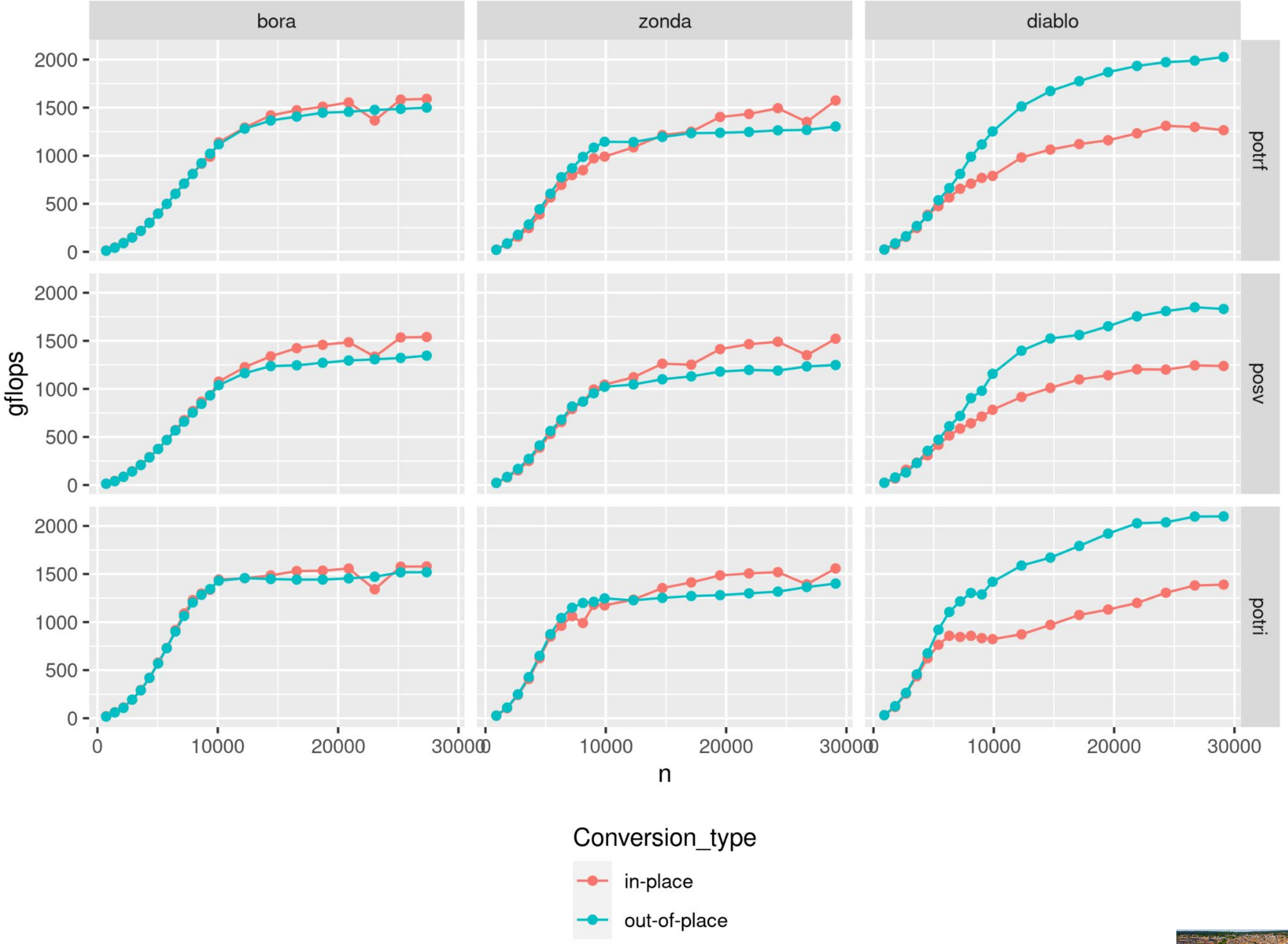
# Performances

Lapack-Layout



Conversion
In-Place

Conversion
Out-Of-Place

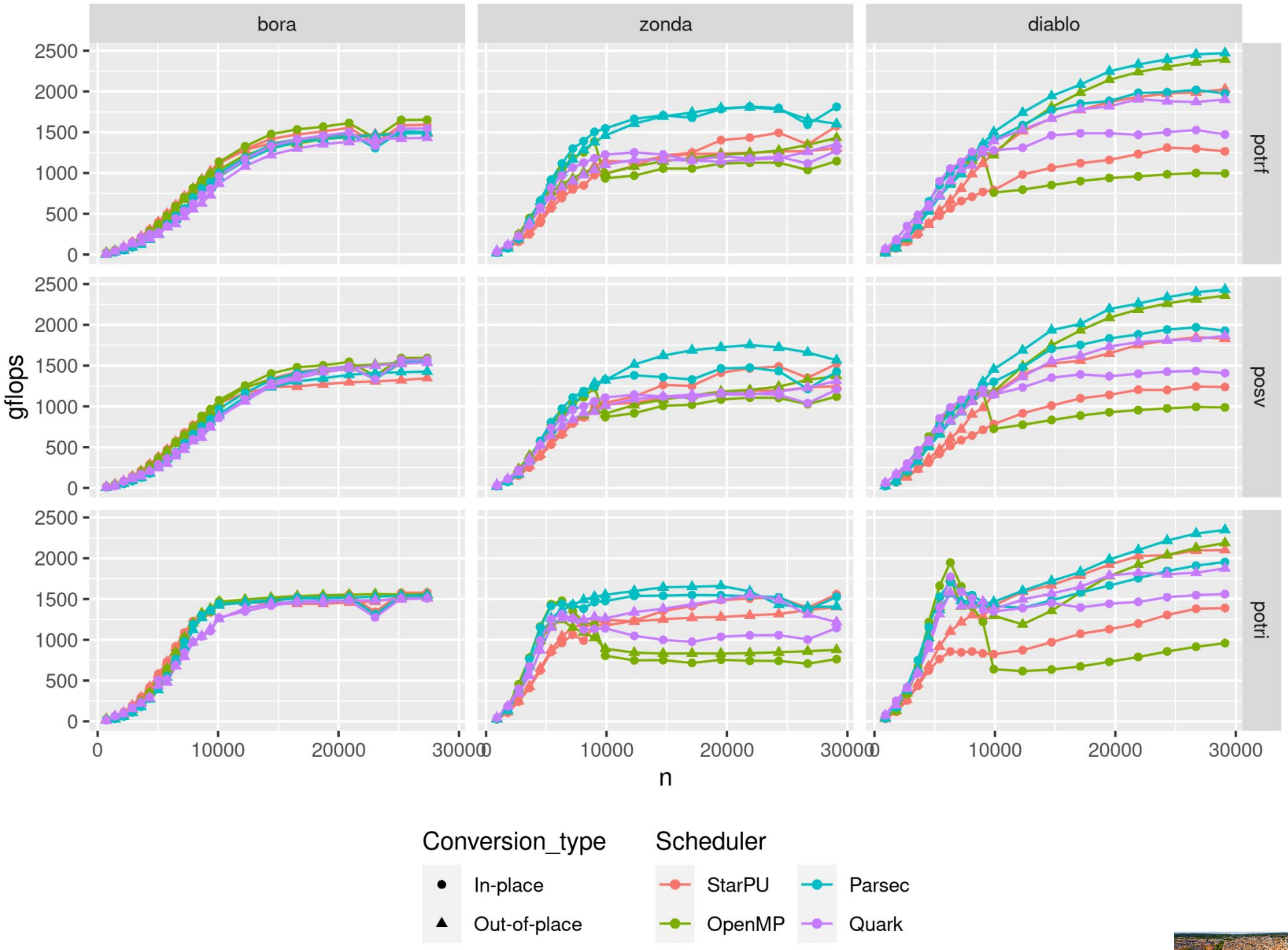


# Performances

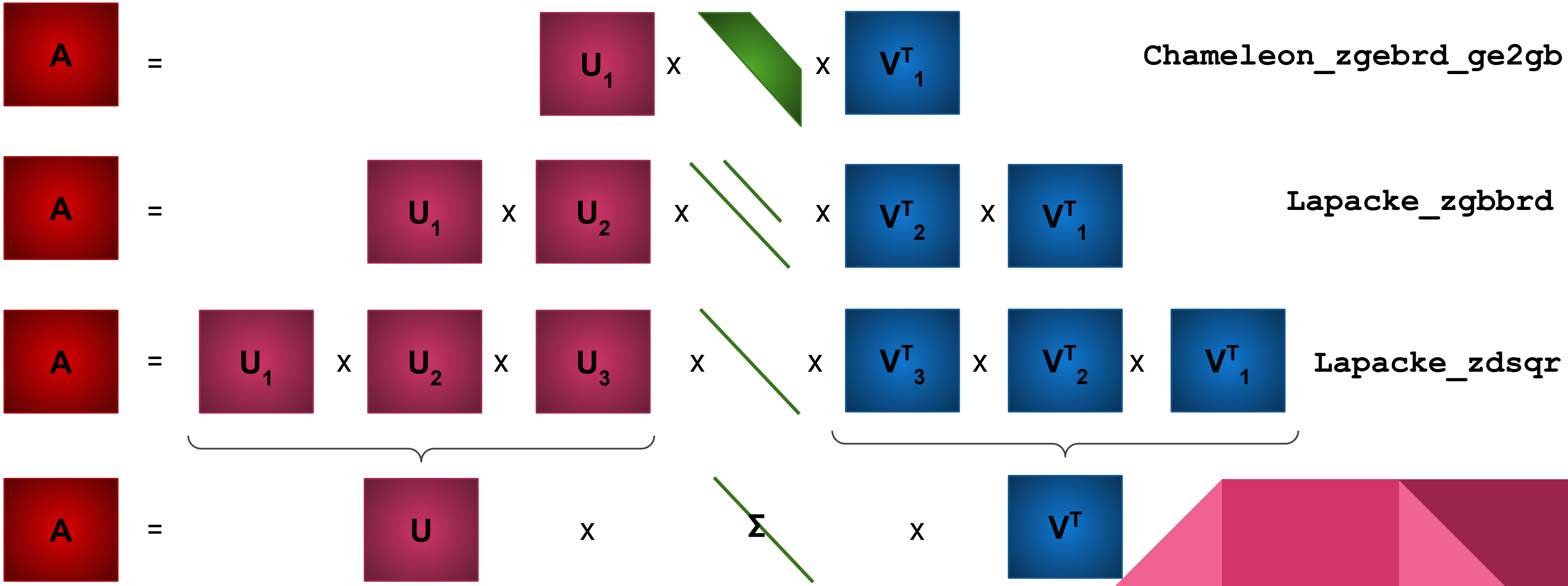
Same behaviour with diablo

Opposite behaviour with zonda

Same behaviour with bora



# Singular Value Decomposition



# Singular Value Decomposition

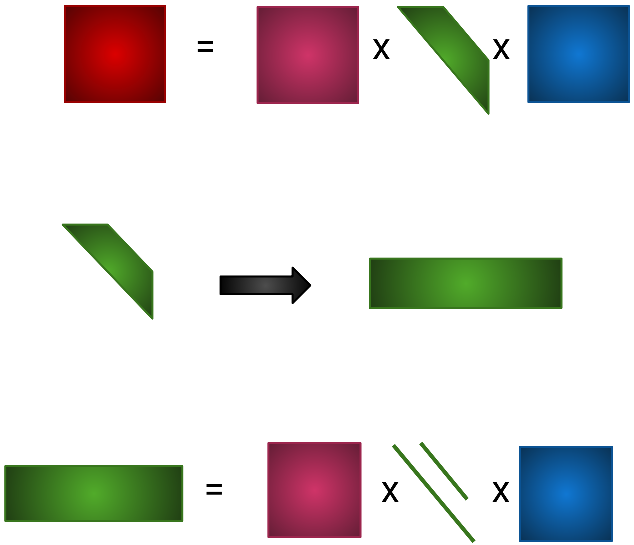
**`Chameleon_zgebrd_ge2gb`** → Had minor errors
Needs to have the tree version

↓

**`Chameleon_pztile2band`** → Was incorrect

↓

**`Lapacke_zgbbrd`**



# Conclusion

Validation of the Standard API:
- Numerical ✓
- On CPU ✓
- On GPU ✓

Benchmark of the Lapack interface ✓

Singular Value Decomposition:
- Testing ✓
- Numerical validation of the singular values ✓
- Numerical validation of the singular vectors ● ● ●
- Improvement of the algorithm using trees ● ● ●


# Françooiiiis, j’ai une questiooonnn