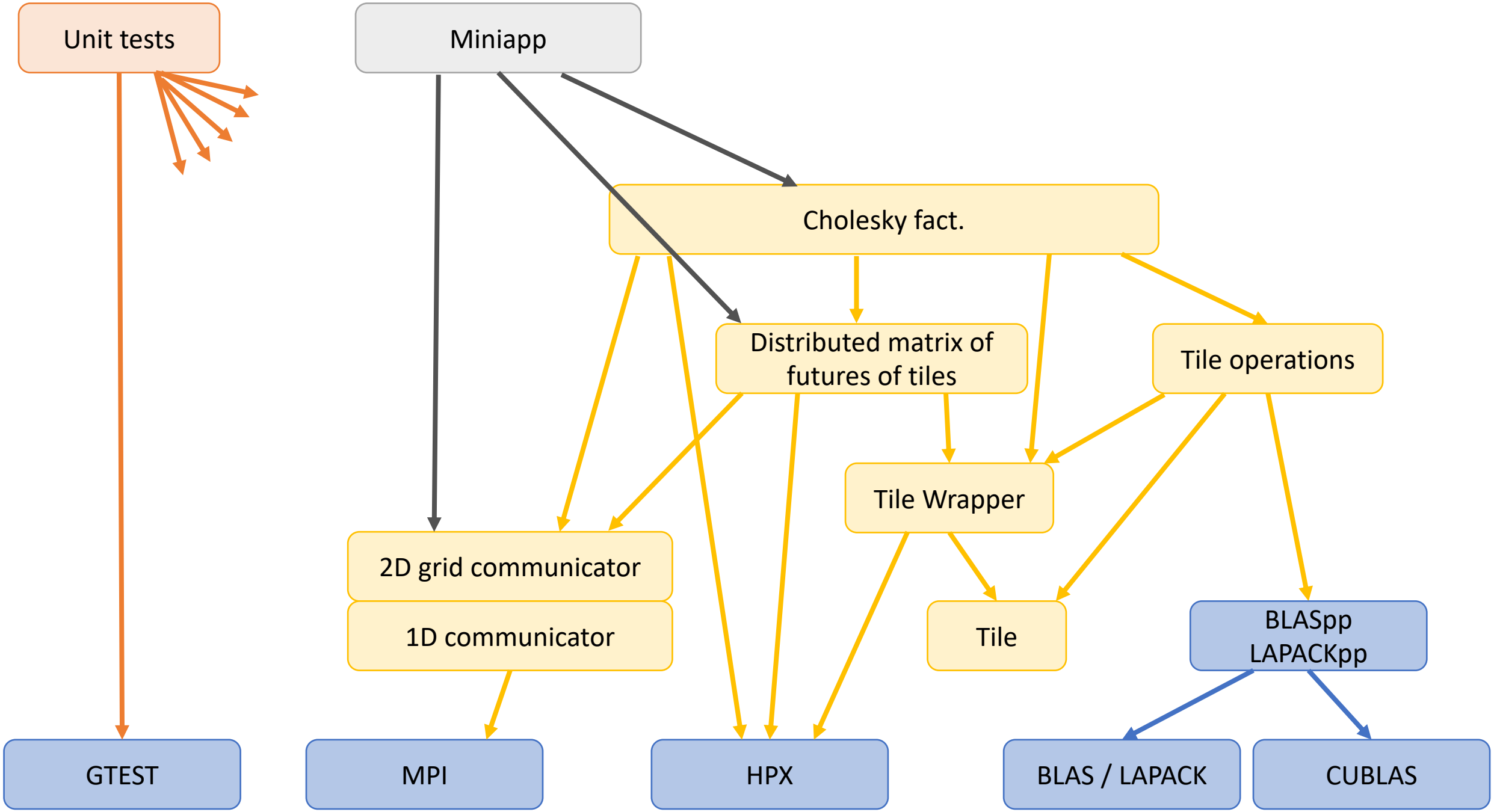
Code structure

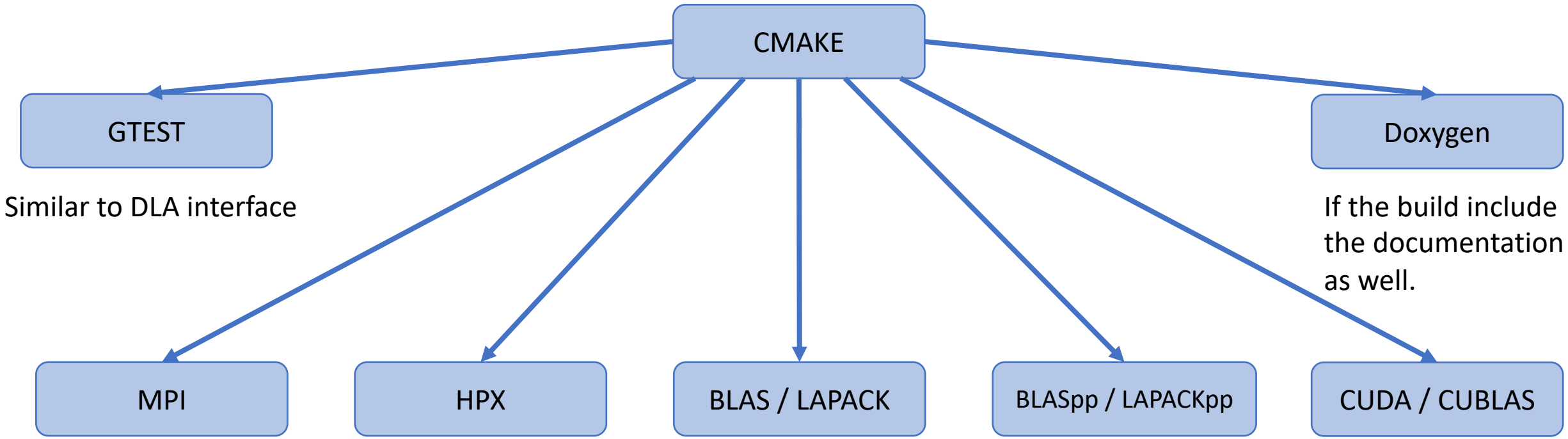
- We start in the “refactoring” branch in the current repository. This branch will be moved to a new repo next month.

- See Coding style (coding.md)
  - Use clang-format (only version 8) with the given .clang-format file

- Class names are indicative, feel free to use better options.


Unit tests
Miniapp
Cholesky fact.
Distributed matrix of futures of tiles
Tile operations
Tile Wrapper
2D grid communicator
1D communicator
Tile
BLASpp
LAPACKpp
GTEST
MPI
HPX
BLAS / LAPACK
CUBLAS

CMAKE structure



Communicators

- Changes w.r.t. prototype:
  - Use MPI_Comm_split instead of MPI grid functions for 2d grids:
    - Compute row and col index w.r.t. the ordering (row/col major)
    - Create row comm. using MPI_Comm_split with color = row index and key = col index,
    - Create col comm. in a similar way.
- Requirements:
  - Construction
  - Construction from existing MPI comms
  - tests

Tile

- Templated on type and device
- Requirements:
  - Construction from (m, n, pointer, leading dimension)
  - Move constructor/ move assignment
  - Copy constructor / copy assignment (only if really needed)
  - Get pointer of (i, j) element
  - Get value of (i, j) element (operator())
  - Get size
  - Get leading dimension
  - tests

Tile operations

- Execute blas operations with the correct pointers, sizes and leading dimensions of the involved tiles.
- E.g. gemm(alpha, tile_a, tile_b, beta, tile_c)
  - tile_a, tile_b are const Tile& and tile_c is Tile&
  - tile_a, tile_b are const Wrapper<Tile>& and tile_c is Wrapper<Tile>&&.
- Operations:
  - gemm
  - trsm
  - Others as needed
  - tests

Tile wrapper, distributed matrix of futures, MatrixRead and MatrixRW

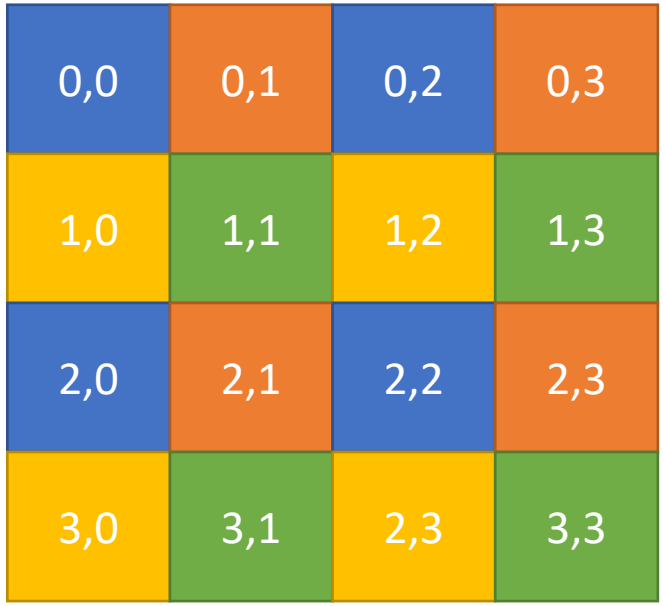
- See m.h for a 1D local example with single element tiles.
- Dist. Matrix of futures requirements:
  - Constructor (allocates memory and created ready futures of tiles)
  - Constructor (for existing memory)
  - Move constructor / assignment
  - Operator(tile_i, tile_j) (tile_i and tile_j are global tile indices)
  - Read(tile_i, tile_j)
  - tests

Cholesky

- Argument should be MatrixRW&&
- Using read and operator() of MatrixRW the code should be simpler.

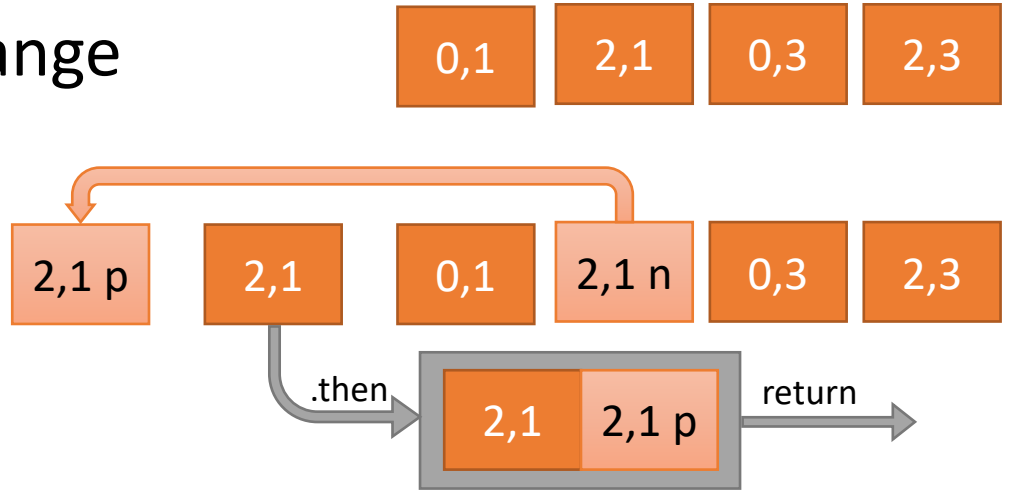
How the matrix of future works

- Matrix is 2D block cyclic distributed.
- Each local tile is represented by a future and a pointer to a shared future (used when the task only reads the tile)



Example operator() orange rank:

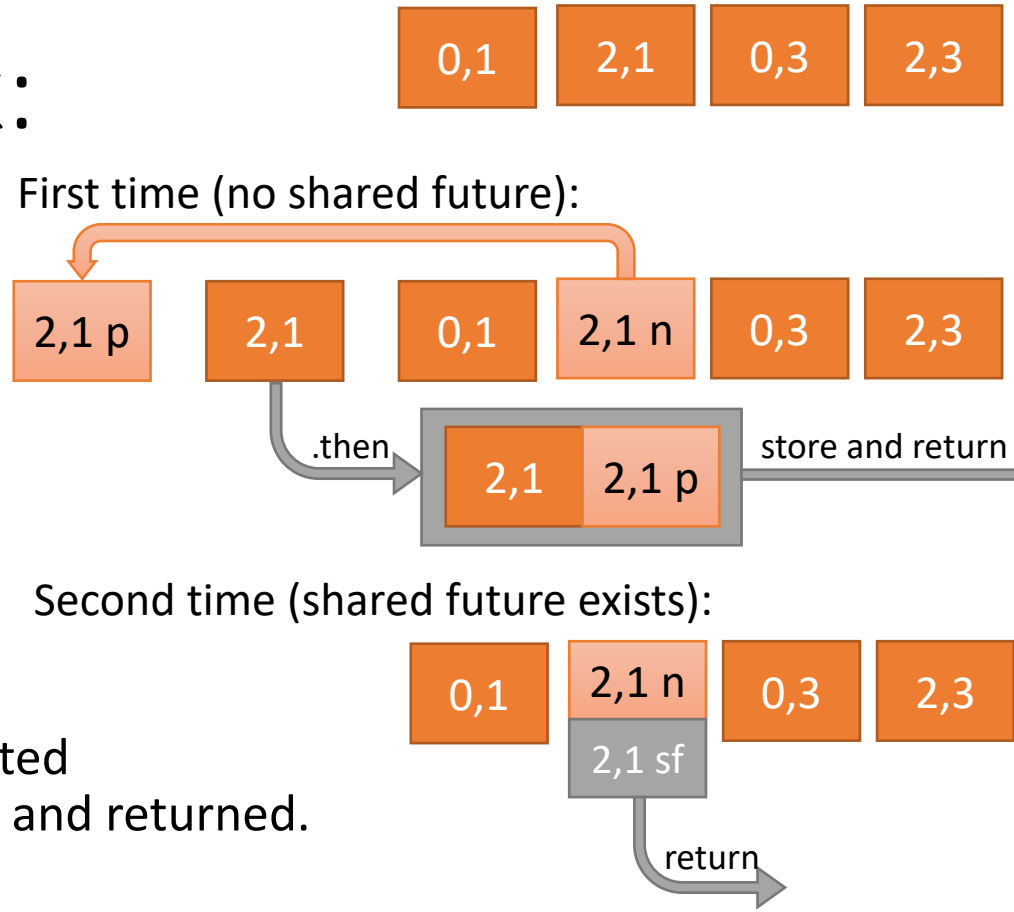
- When operator()(2,1) is called on the orange rank:
  - a promise<Tile> “2,1 p” is created
  - the future<Tile> is replaced by the future of the promise “2,1 n”



  - a future<Wrapper<Tile>> is created from the “old” future<Tile> and returned.
    (a synchronous .then is used, therefore the task is executed immediately if the future<Tile> is ready, otherwise the task is executed immediately after the “old” future<Tile> gets ready.

  - After the task that modify the Tile 2,1 is finished, the destructor of Wrapper<Tile> is called. It sets the value of “2,1 p” moving the modified Tile.

Example read() orange rank:



- When read(2,1) is called on the orange rank:
  - If the shared future of 2,1 doesn't exist:
    - a promise<Tile> "2,1 p" is created
    - the future<Tile> is replaced by the future of the promise "2,1 n"
    - a shared_future<Wrapper<Tile>> "2,1 sf" is created from the "old" future<Tile> stored in the pointer and returned.
  - Otherwise the shared future is returned
    - The copy of the shared_future stored in the Matrix is destroyed when operator() is called for the same tile.
  - After all the task that read the Tile 2,1 are finished, the destructor of Wrapper<Tile> is called. It sets the value of "2,1 p" moving the modified Tile.

MatrixRead

- MatrixRead and MatrixRW allow a routine to schedule the computation tasks in a different task.
- MatrixRead creates all the shared futures and store a copy of them.
- When the MatrixRead object is destructed (or done() is called) the local copy of the shared future is destroyed (Note: the Wrapper is destroyed only when all shared_future instances are destroyed).
- Therefore:
  - Reads of the original Matrix and of MatrixRead can be concurrent
  - Writes of the original Matrix needs to wait the end of all read operation. (when no more shared future of the given Tile exists)

MatrixRW

- MatrixRW moves all the futures and shared future of the original Matrix and replace them with promise futures which are stored by MatrixRW.
- When the MatrixRW object is destructed (or done() is called) the local copy of the shared future is destroyed and the future of the original Matrix will be set with the value of MatrixRW current future.
- Therefore:
  - Reads of the original Matrix and reads of MatrixRW after doneWrite() is called can be concurrent.
  - Writes of the original Matrix needs to wait the end of all operation scheduled using MatrixRW.