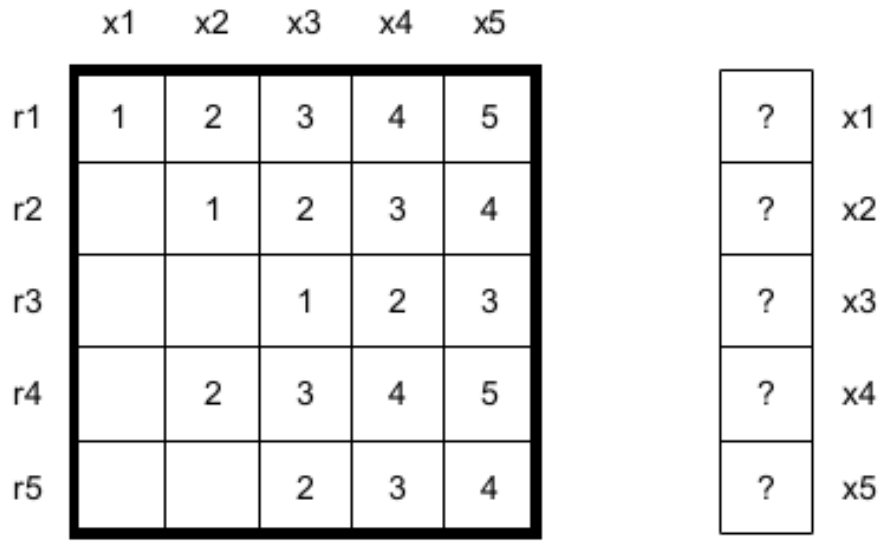
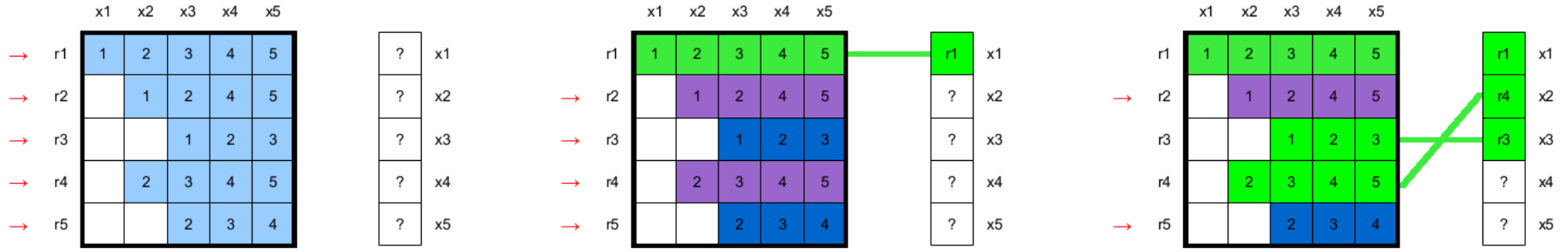
General information



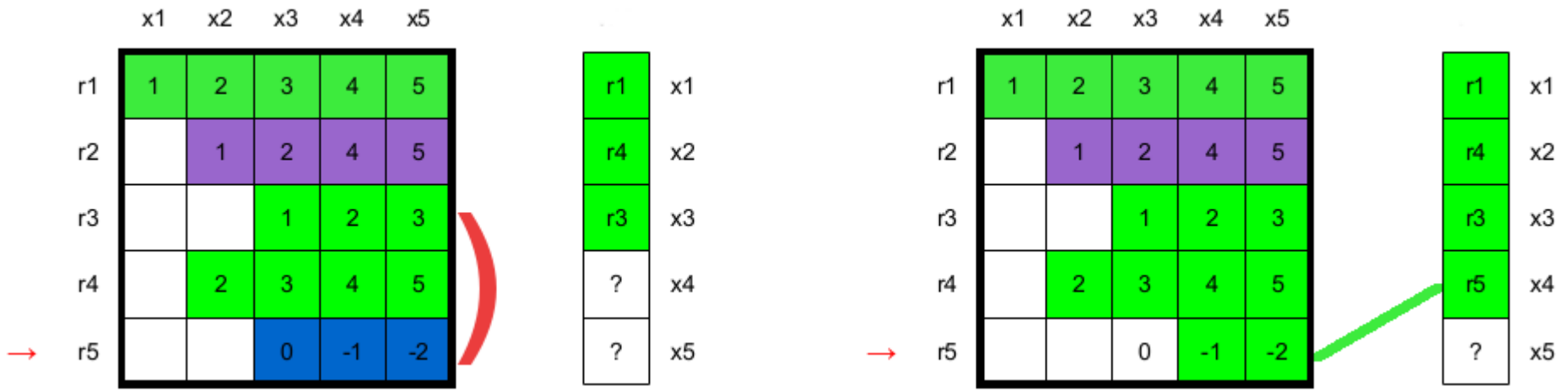
- The solver works as a black box – it only expects a square matrix and a RHS vector; it does not require information about the problem which is represented in the matrix.
- It's only output is a vector containing the solution or – in the case of unsolvable systems – an exception.

Forward substitution phase



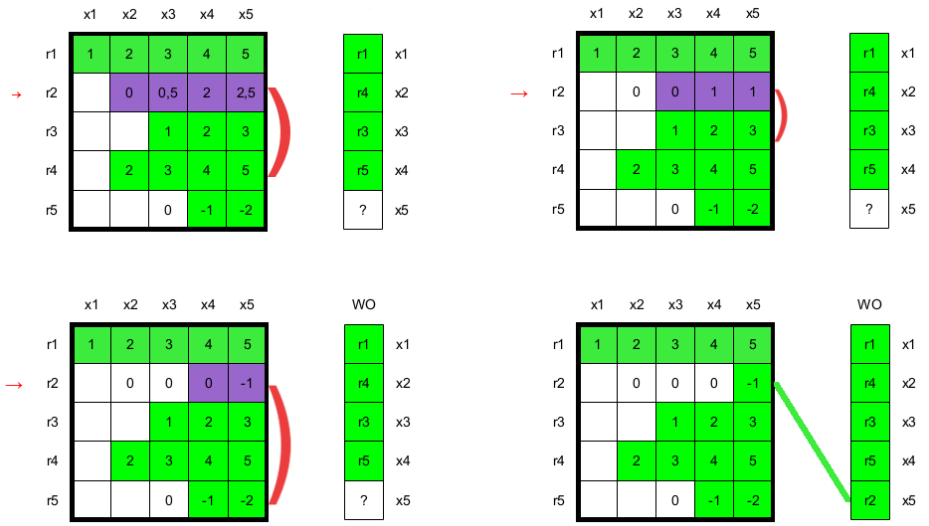
- Ideally, a separate thread would be used for every row in the matrix; in the sequential prototype, this behavior is mimicked by a `for` loop.
- For every row, it is established which of the functions (x_n) it describes
  - In the above example, row 1 describes x_1, row 3 describes x_3 and row 4 describes x_2.
  - The other functions are left as „not yet found” for the time being.
- These information are stored in a separate vector, which serves as a map of described function to row ID.

Forward substitution phase



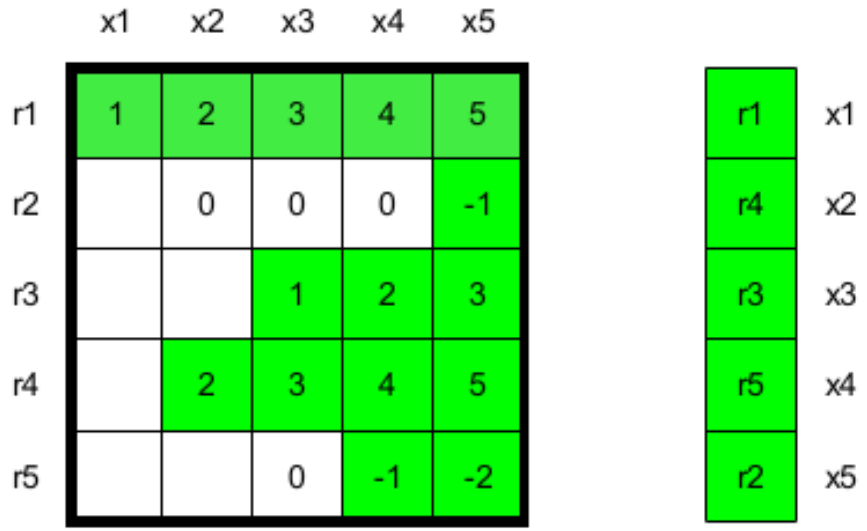
- As in the typical Gauss approach, two rows cannot describe the same function.
- If a thread checks the function to row map and the function in the row is already mapped to a row, a reduction will be performed on the „conflicting” (not found in the map) row – by adding the original row to the conflicting row times an appropriate multiplier.
- In the above example row 5 is reduced using the data in row 3 (since row 3 was the first to „claim” ownership of function x_3).

Forward substitution phase



- In this example, for row 5 a single round of reduction was required.
- However reducing row 2 from x_2 to x_5 requires three rounds of reduction.
- In the prototype this is achieved by stepping the `for` loop back by one upon reduction, in a thread this could be achieved with a `while` loop.
- While every thread will have to perform at most *N-1* reductions in an *N x N* matrix, I still consider this a bottleneck, possibly solvable with preemptive matrix reordering.

Backward substitution phase



- Typical approach to Gauss elimination method produces a matrix in echelon form.
- This method produces a so-called „mapped echelon form”, where backward substitution cannot be performed by walking the matrix in bottom-up order, but it must be performed by walking the function to row map bottom-up.
- The backward substitution phase cannot be easily paralellized, so it is performed in sequence.

Paralellization for a GPU platform

- From the GPU paralellism standpoint, the requirement of locking parts of the function to row map is a major bottleneck (since not only hitting the device memory is very time ineffective, but also has the threads sleeping for significant amounts of time).
- The solution I'm currently exploring involves reordering the matrix, then slicing it into independent parts (workgroups) that will be processed by different streaming multiprocessors.
- This way, locking, waiting and reduction only happens in a particular multiprocessor's memory, which is much cheaper than hitting (and locking on) the global device's memory.
- The steps of reordering and performing parts of the solution in slices will be performed as many times as necessary; this will work well (in a few steps of repetition) for banded matrices, slightly worse for block and dense matrices.
- Reordering the matrix directly on the GPU is time-costly, but allows avoiding repeatedly downloading and reuploading data over PCI-Express connections, which is one of the most costly steps in the overall solution.