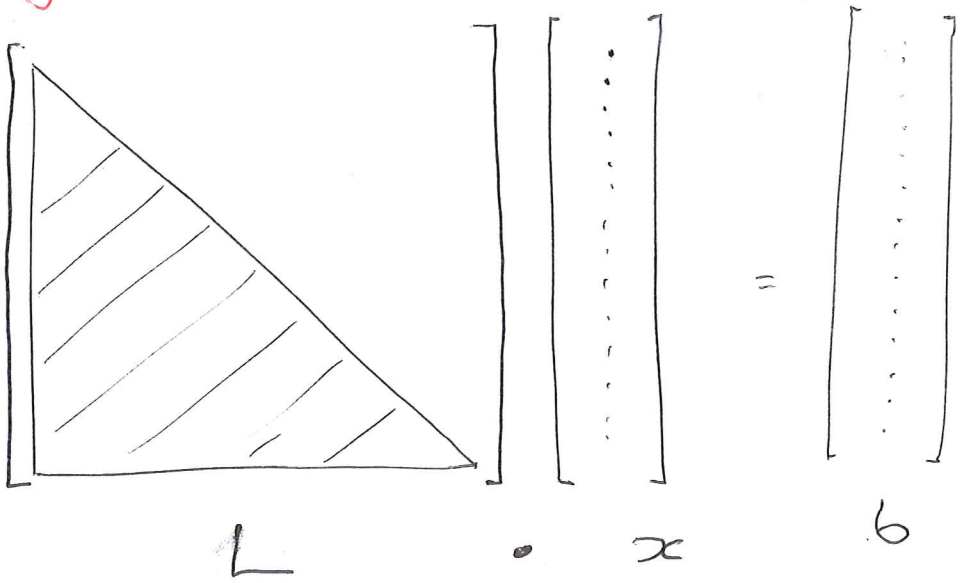
Solving a system with a lower-triangular matrix



Theory describes the algorithm as:

```
for i = 1 ... N                    // Foreach row
    x_i ← b_i                      // Start with right-hand side
(A1)  for j = 1 ... (i-1)
        x_i ← x_i - L_ij x_j       // Subtract all lower-indexed x_j's (move them to RHS)
    x_i ← x_i / L_ii               // Divide by diagonal element
```

→ we will next consider some useful variations of this "baseline" algorithm

"In-Place" Forward Substitution

Applicable when the solution to $Lx = b$ is allowed to "overwrite" the vector b.

Stated another way... let's assume that initially $\underline{\underline{x}}$ contains the right-hand side, and then is overwritten in-place by the solution. This is then written:

```
for i = 1 ... N                    // Assume x contains b at start
(*1)    for j = 1 ... (i-1)
            x_i ← x_i - L_ij x_j
        x_i ← x_i / L_ii
```

Of course, the statement of these algorithms implicitly presumes that L is a dense triangular matrix. How would we express this with CSR matrices?

forward substitution - CSR version

(Note: We will switch to 0-based indexing to match our implementation)

The trick is to replace the loop marked by ($*_1$) with a traversal of the sparse row!

```
for i = 0 ... (N-1)
    x[i] ← b[i]                    (⇐ skip this for in-place)
    for k = rowOffsets[i] ... rowOffsets[i+1] - 2
    [ j ← columnIndices[k]
    [ x[i] ← x[i] - values[k] * x[j]
    x[i] ← x[i] / values[offsets[i+1]-1]
```

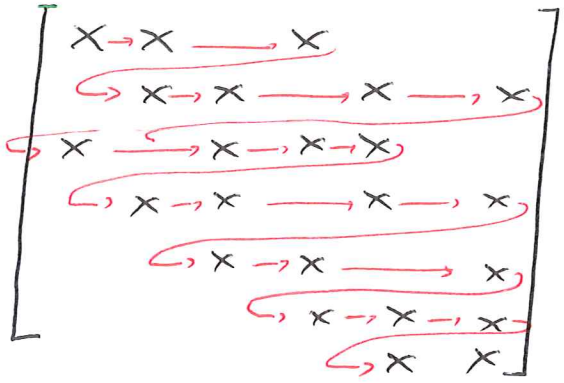
Stop 1-entry before diagonal! This loop might be empty!

Notes:

→ Primary contributor to the cost: reading the matrix L

→ Algorithm "feels" very serial ... more on this later

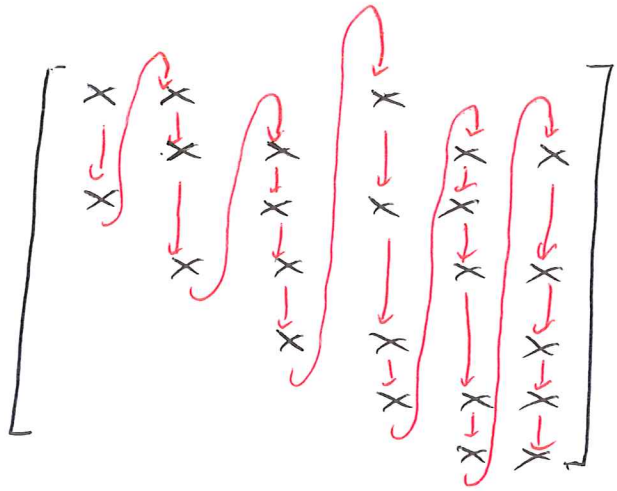
Challenge: What if the matrix was given in the CSC (compressed spare column) format instead?



CSR

row offsets
column Indices
values

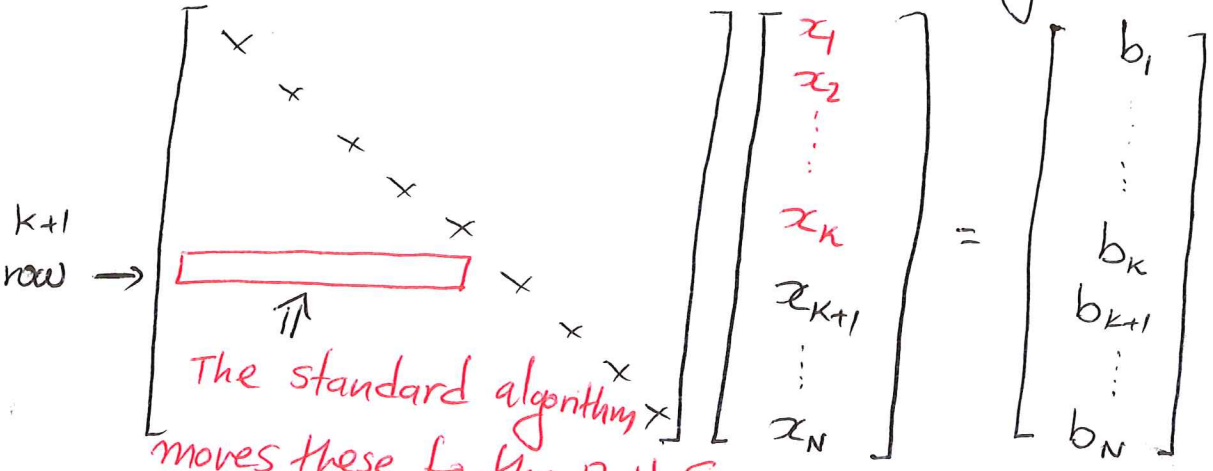
vs



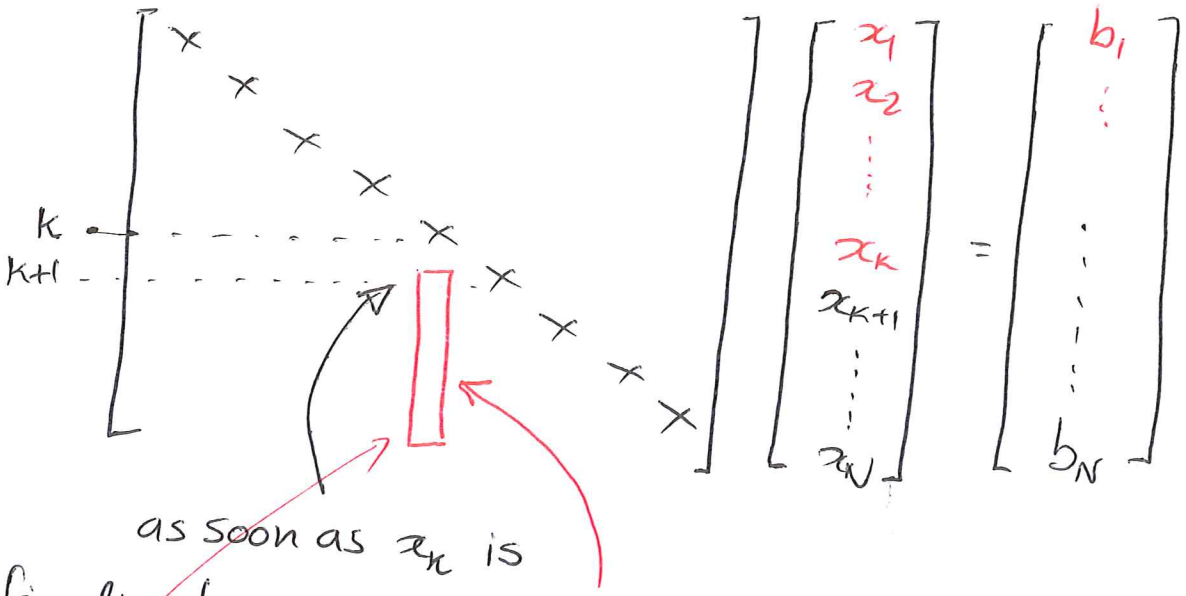
CSC

column offsets
row Indices
values

Key insight for algorithm transformation:
At any stage of forwardSub, we have some finalized x's $(x_1, x_2, \ldots, x_k)$ and some yet-to compute.



An alternative view of the $(k+1)$th step:



as soon as x_k is finalized, move all of these to the R.H.S. instead!

Mathematical Statement (pseudocode)

$x \leftarrow b$ // copy entire $b \rightarrow x$ (unless in-place)

for $j = 1 \ldots N$

$x_j \leftarrow x_j / L_{jj}$ (This yields the final x_j! I can do this because we eliminated all entries to the left of x_j, in the j-th equation!)

for $i = (j+1) \ldots N$

$x_i \leftarrow x_i - L_{ij} x_j$

this eliminates the column shown here

CSC version - Sparse

```
x ← b        // copy entire array - skip for in-place
for j = 0 ... N-1
    x[j] ← x[j] / values[columnOffsets[j+1] - 1]
    for k = columnOffsets[j] + 1 ... columnOffsets[j+1] - 1   ⇐ skip diagonal
        i ← rowIndices[k]
        x[i] ← x[i] - values[k] * x[j]
```