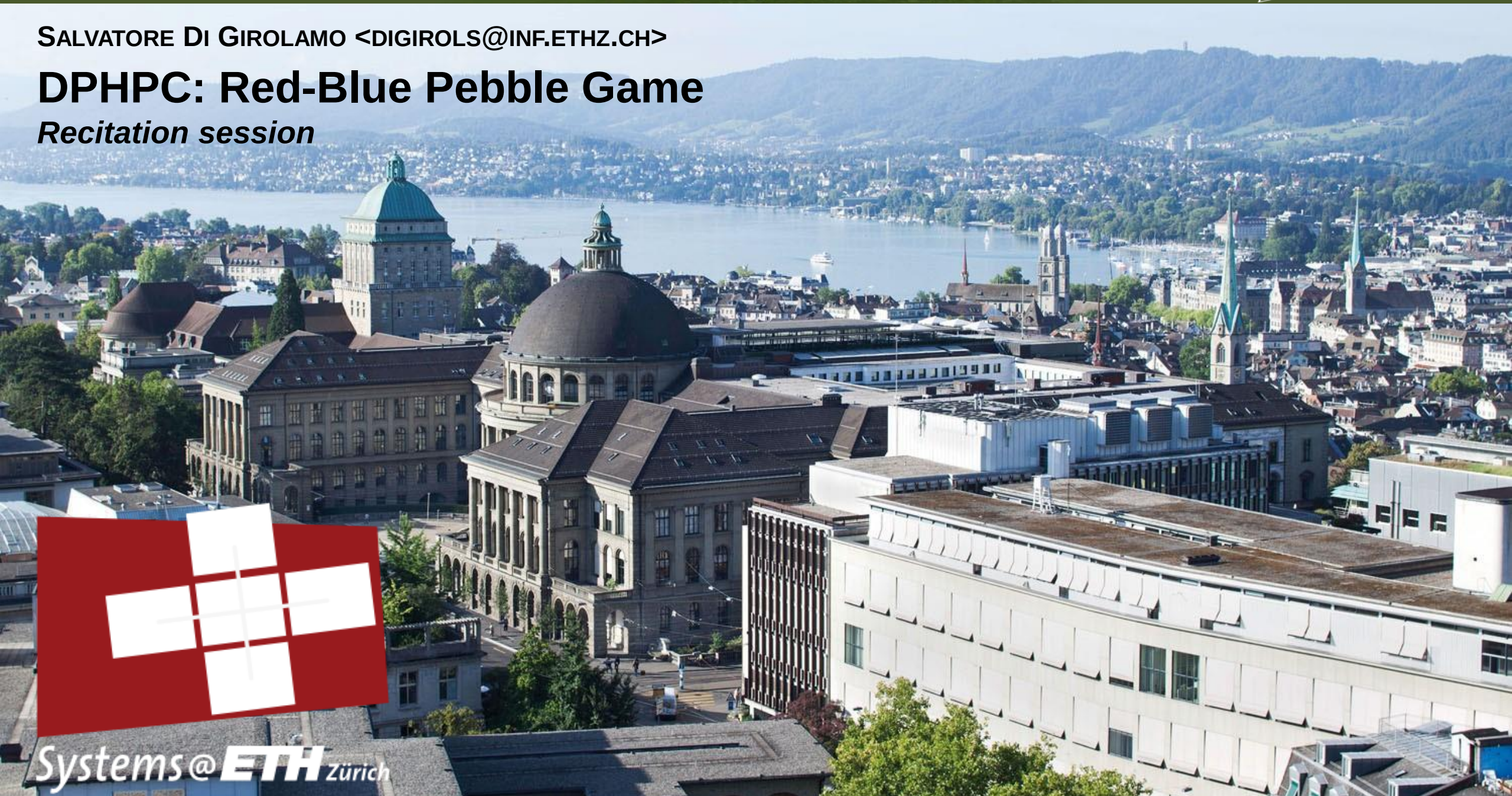
SALVATORE DI GIROLAMO <DIGIROLS@INF.ETHZ.CH>

DPHPC: Red-Blue Pebble Game

Recitation session



Administrativia

- **Final presentations on Monday 17/12**
- **Send slides by Sunday 16/11 11:59pm CH time!**
  - PDF or PowerPoint
- **10min/team (8min talk + 2min Q&A)**

Little's Law

Imagine you want to board a train which leaves in 20 minutes. But before you have to buy the train ticket at a counter. You see that there are about 50 people in line before you. Serving a customer takes 40 seconds on average.

What property has to hold for this system to be stable? Will you miss your train?

Roofline Model

Assume a NUMA architecture with 2 nodes and a peak memory bandwidth of $B_1 = 74.2$ GB/s. Every node has 4 cores and can carry out up to $P = 332.8$ GFLOPs/sec. Each core runs at 2.6 GHz. If the memory accesses are not well balanced, the peak memory bandwidth becomes $B_2 = 31.32$ GB/s. Draw a roofline plot for this processor. If a program and input combination land on the lower left of the plot, what does this tell you about the program?

Will all program executions yield points which lie either on the diagonal or on the "roof" of the roofline plot?

Sparse Matrix Vector Multiplication SpMV

The following code compute a Sparse Matrix Vector Multiplication $\vec{y} = A \cdot \vec{x}$ between a matrix A (sparse) and a vector $\vec{x}$ (dense). The matrix is stored in the Compressed Row Storage format.

```
<fill data structures: blockptr, values, col_idx, row_start>

#pragma omp parallel private(i,j,is,ie,j0,y0,thread,bs,be)

thread = omp_get_thread_num()

//Compute the block boundaries
bstart = blockptr[thread]
bend = blockptr[thread+1]

for (i=bstart; i<bend; i++){
  y0=0
  row_start = row_start[i]
  next_row_start = row_start[i+1]

  for (j=row_start; j<next_row_start; j++){
    j0 = col_idx[j]
    y0 += value[j] * x[j0]
  }

  y[i] = y0
}
```

Assume that $\vec{x}$ and $\vec{y}$ are kept in cache. The CSR format uses 4byte `integers` to store column indexes. Values are stored using 8byte `doubles`. Compute the operational intensity and check if the code is memory- or compute-bound w.r.t. the previously described architecture (consider only the innermost loop).

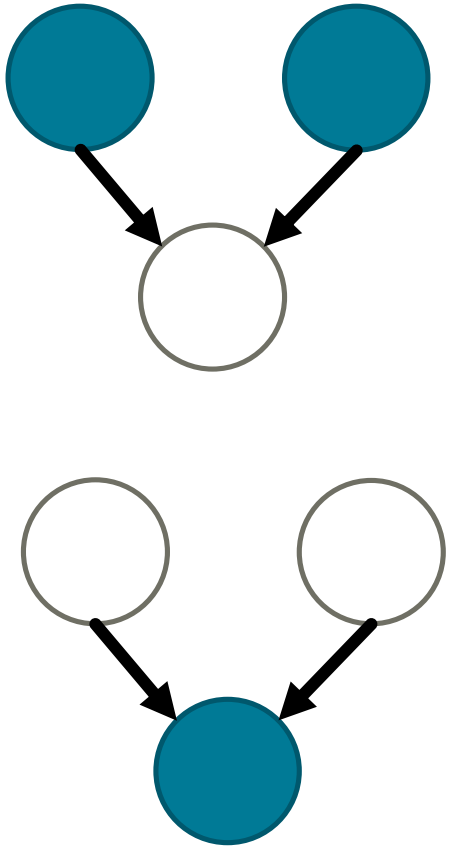
You run this code, observing that it reaches a performance up to 5.22 GFLOPS, and you notice that this is mostly due to how the array `value` is stored. Describe an optmization that you can apply to improve the performance.

Vectorizing the Floyd-Warshall Algorithm

The Floyd Warshall Algorithm is for solving the All Pairs Shortest Path problem. The problem is to find shortest distances between every pair of vertices in a given edge weighted directed graph. Starting from the code provided in `floyd_warshall.tar.gz`, use SSE4 or AVX intrinsics to vectorize it and analyze the obtained speed up.

Red-Blue Pebble Game

- **DAG with:**
  - Input vertices (no predecessors)
  - Output vertices (no successors)
  - **Initial state:** blue pebbles on the input vertices
  - **Final state:** blue pebbles on the output vertices



Red-Blue Pebble Game

- **R1:** (Input) A red pebble may be placed on any vertex that has a blue pebble.
- **R2:** (Output) A blue pebble may be placed on any vertex that has a red pebble.
- **R3:** (Compute) If all the immediate predecessors of a vertex have red pebbles, a red pebble may be placed on that vertex.
- **R4:** (Delete) A pebble (red or blue) may be removed from any vertex.

Transition: ordered pair of configurations (C1, C2). C2 follows from C1 according to one of the above rules.

Calculation: sequence of configurations, each successive pair of which form a transition.

Complete calculation: calculation starting with the *initial* configuration and *ending* with the final one.

Red-Blue Pebble GameJia-Wei, Hong, and Hsiang-Tsung Kung. "I/O complexity: The red-blue pebble game." Proceedings of the thirteenth annual ACM symposium on Theory of computing. ACM, 1981.

Red-Blue Pebble Game: I/O complexity

- **Can model a two-level memory layout**
  - Blue pebbles: slow memory
  - Red pebbles: fast memory

- **Vertices represent operations and their result**
- **An edge means that the result of an operation is the operand of another**
- **An operation can be performed only if all of its operands are in fast memory**

- **R1: get an operand from slow memory to fast memory**
- **R2: store the result from fast memory to slow memory**
- **R3: perform the operation and store the result in fast memory**
- **R4: free memory location in slow or fast memory**

- **Maximum number of blue or red pebbles represents the size of the slow or fast memory, respectively.**

Red-Blue Pebble Game: I/O complexity

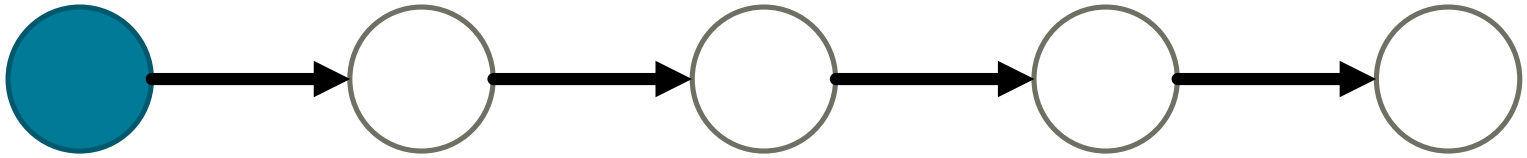
- **Assume:**
  - Slow memory is infinite (number of blue pebbles not limited)
  - Size of the fast memory is S: at most S red pebbles on the graph at any time!

- **I/O time: minimum number of transitions according to R1 or R2 required by any complete calculation.**

- **Finding this is PSPACE-complete!**
  - Can be bounded by partitiongs and covering schemes of the DAG (e.g., S-Partitioning).

Red-Blue Pebble Game

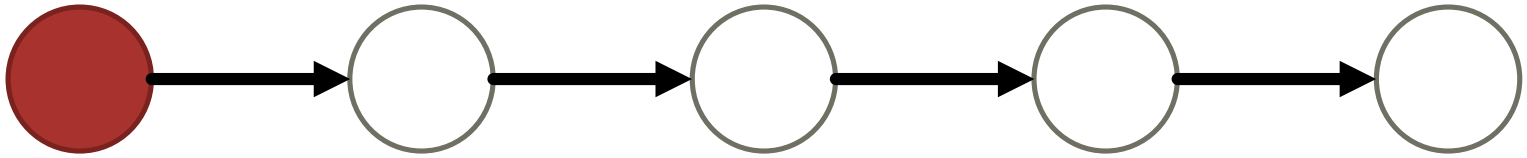
S = 2



I/O transfers: 0

Red-Blue Pebble Game

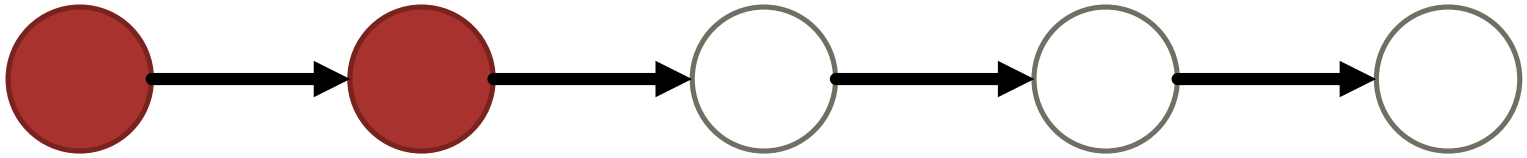
S = 2



I/O transfers: 1

Red-Blue Pebble Game

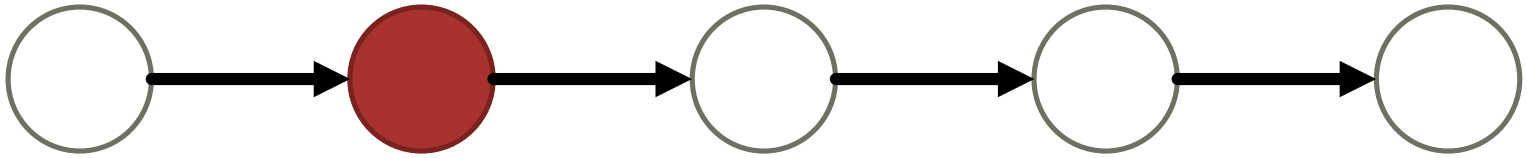
S = 2



I/O transfers: 1

Red-Blue Pebble Game

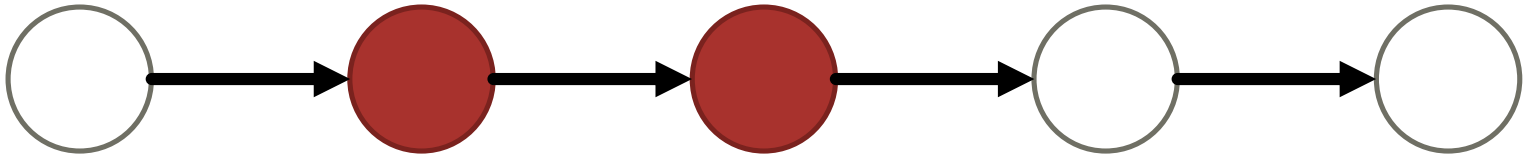
S = 2



I/O transfers: 1

Red-Blue Pebble Game

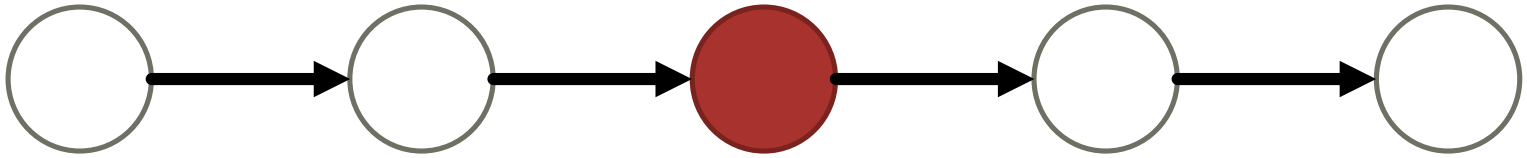
S = 2



I/O transfers: 1

Red-Blue Pebble Game

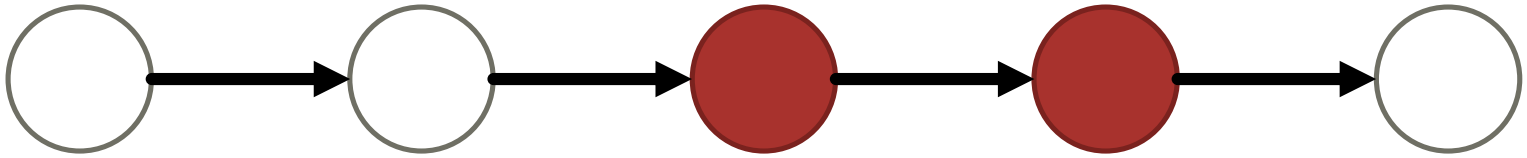
S = 2



I/O transfers: 1

Red-Blue Pebble Game

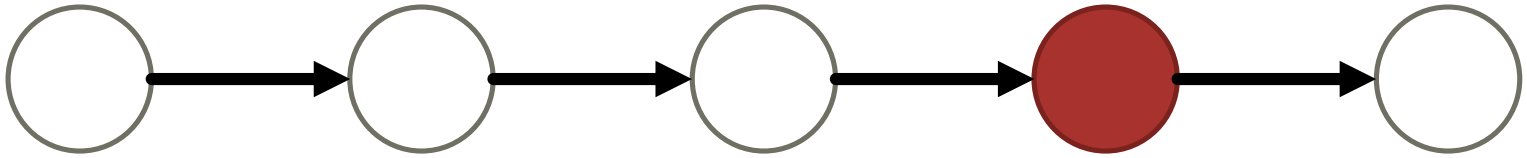
S = 2



I/O transfers: 1

Red-Blue Pebble Game

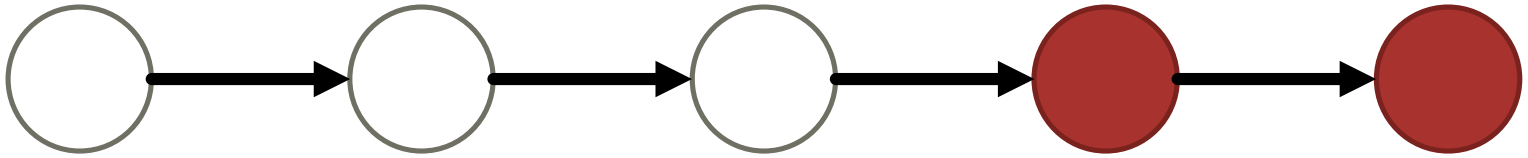
S = 2



I/O transfers: 1

Red-Blue Pebble Game

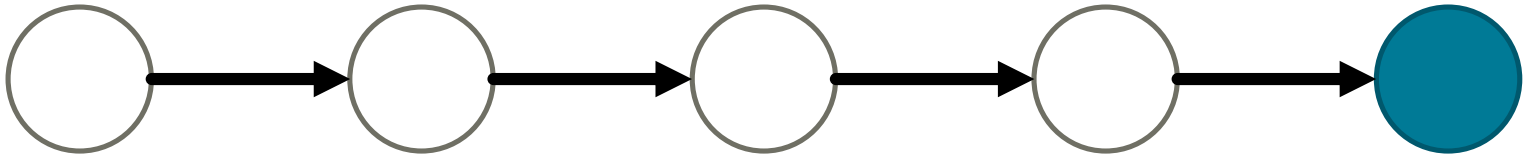
S = 2



I/O transfers: 1

Red-Blue Pebble Game

S = 2



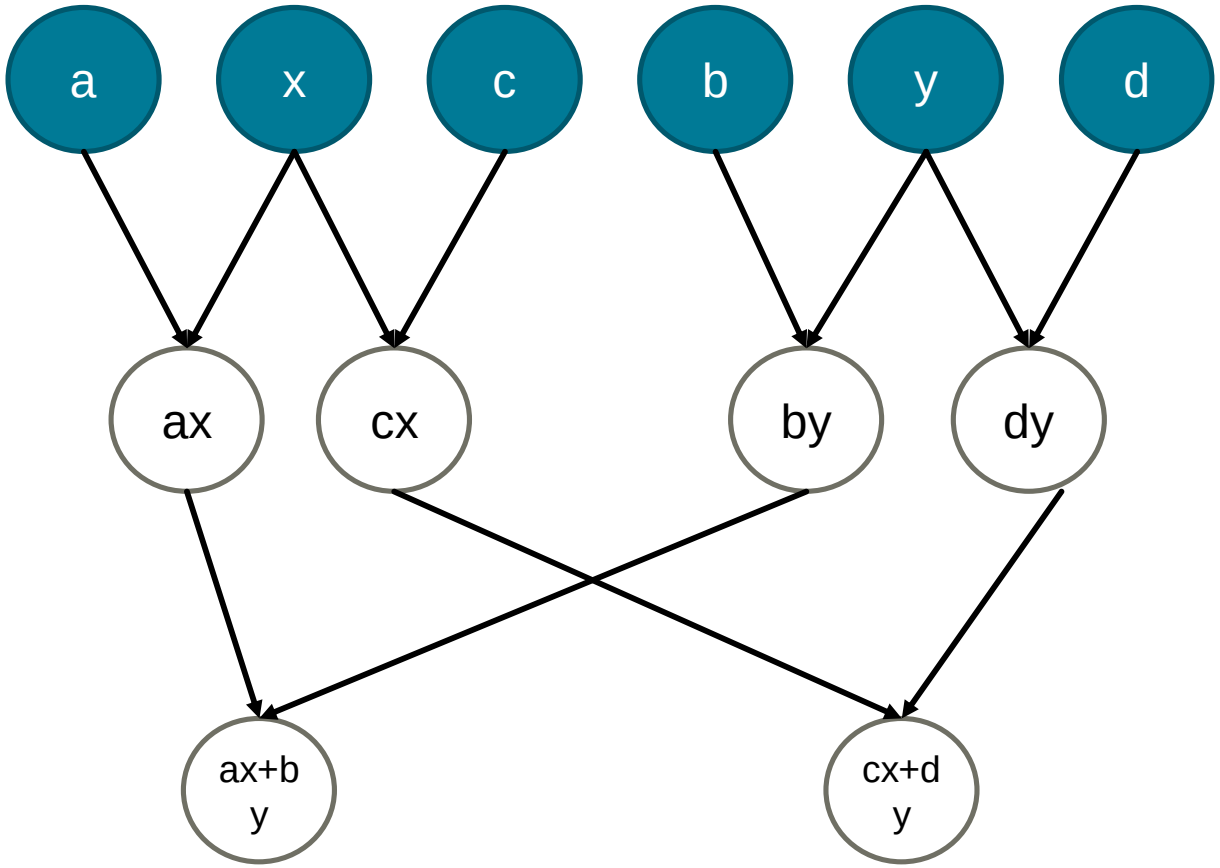
I/O transfers: 2

Exercise

- **Given a matrix-vector multiplication between a NxN matrix and a N vector, define a DAG for the computation when N=2 and S=5. What is the minimum number of I/O transfers you can find?**

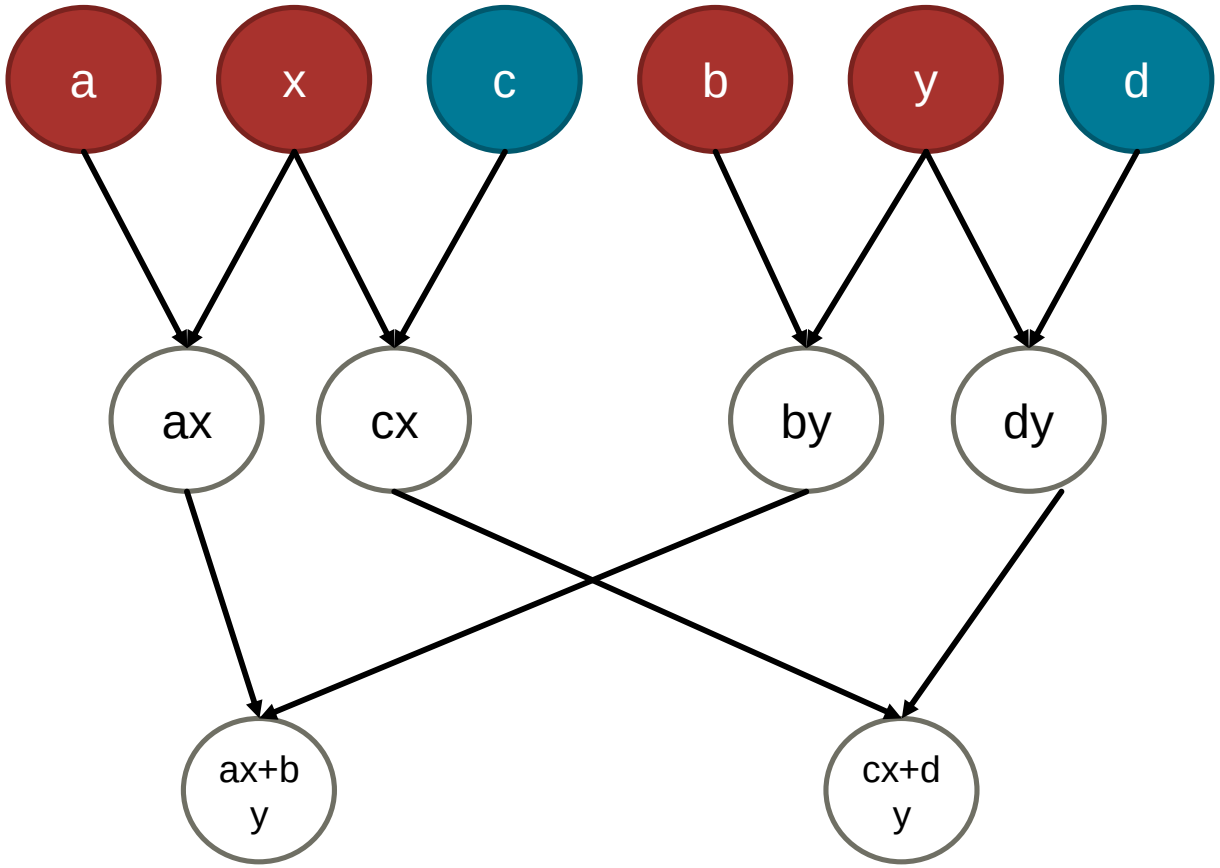
S = 5

I/O transfers: 0



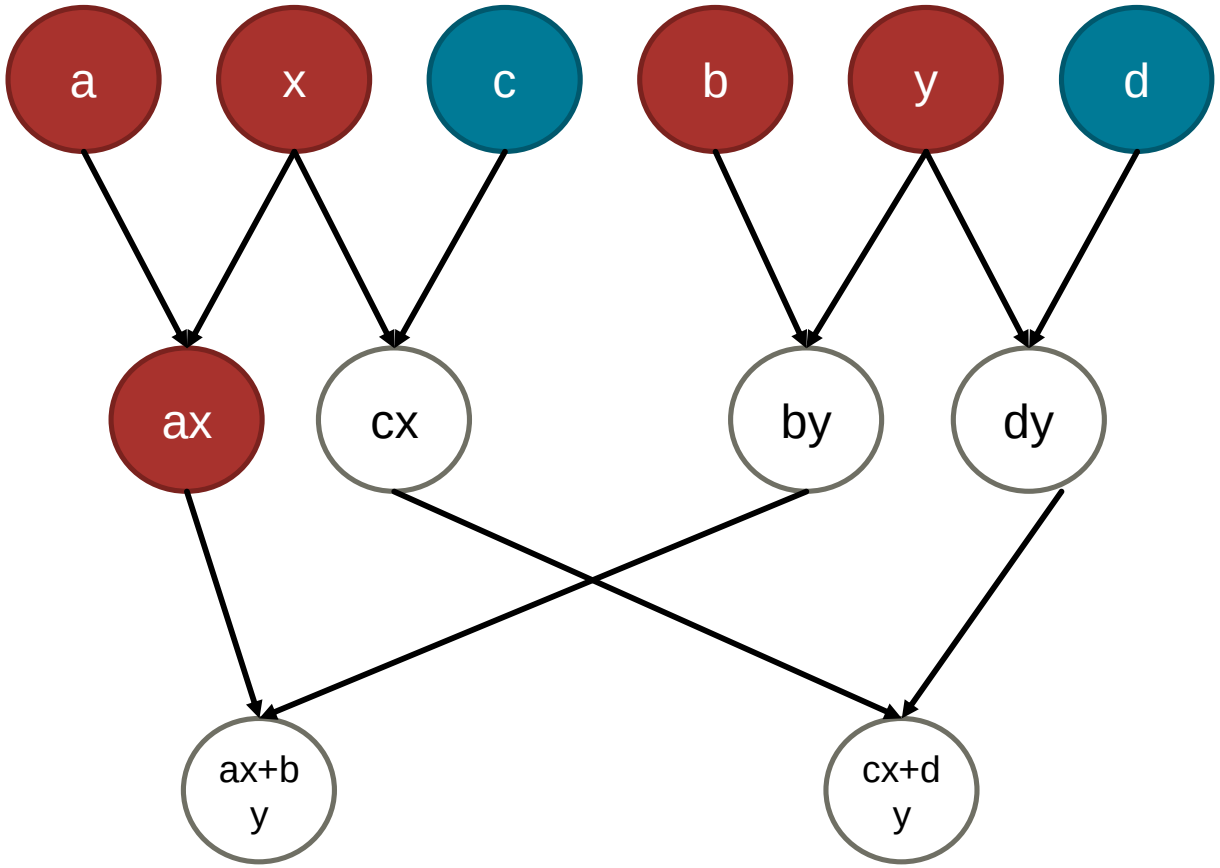
S = 5

I/O transfers: 4



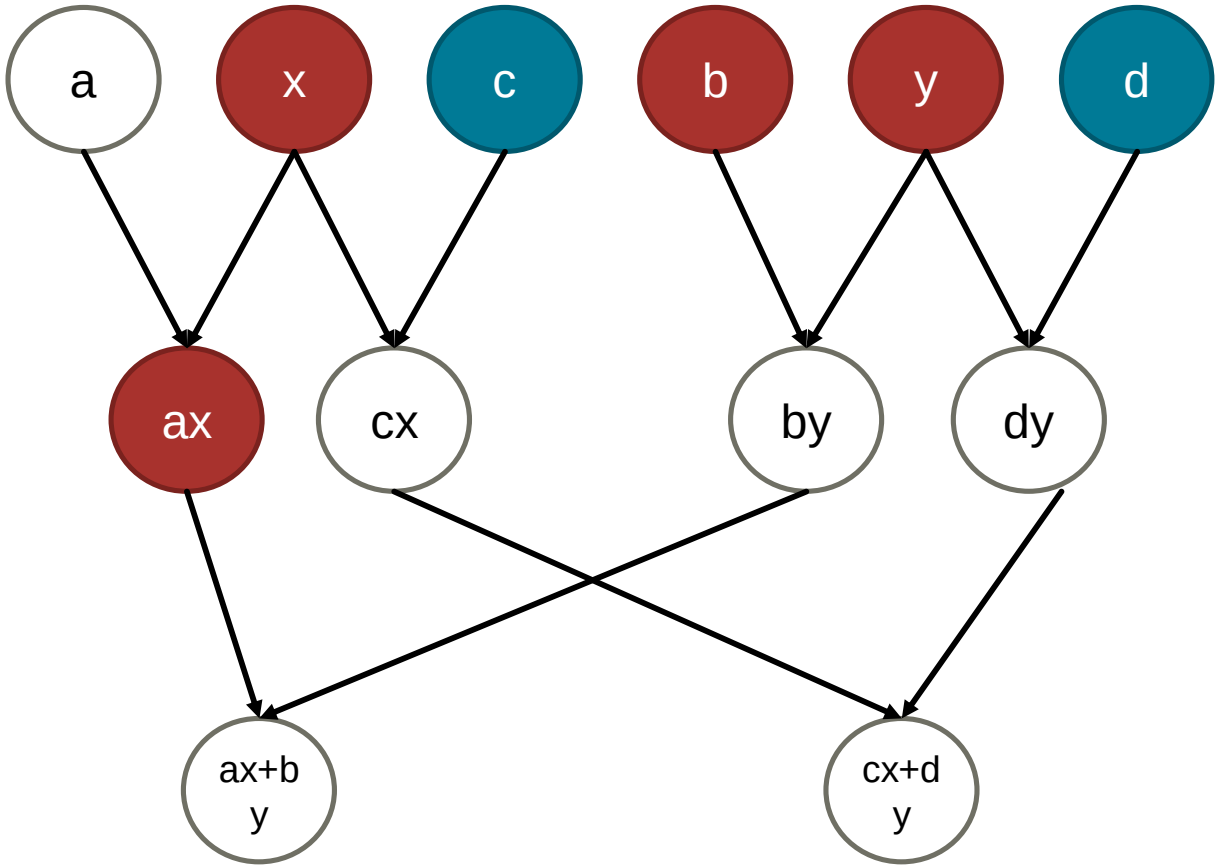
S = 5

I/O transfers: 4



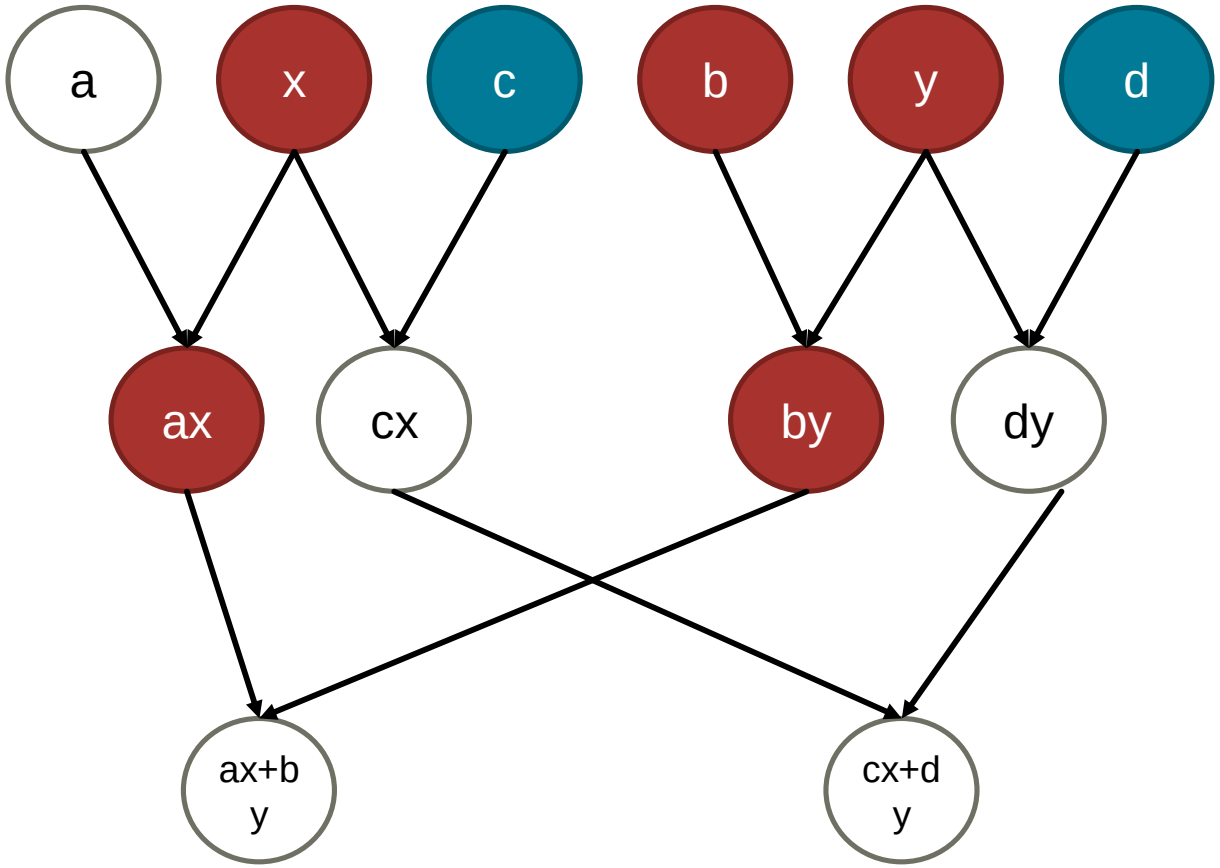
S = 5

I/O transfers: 4



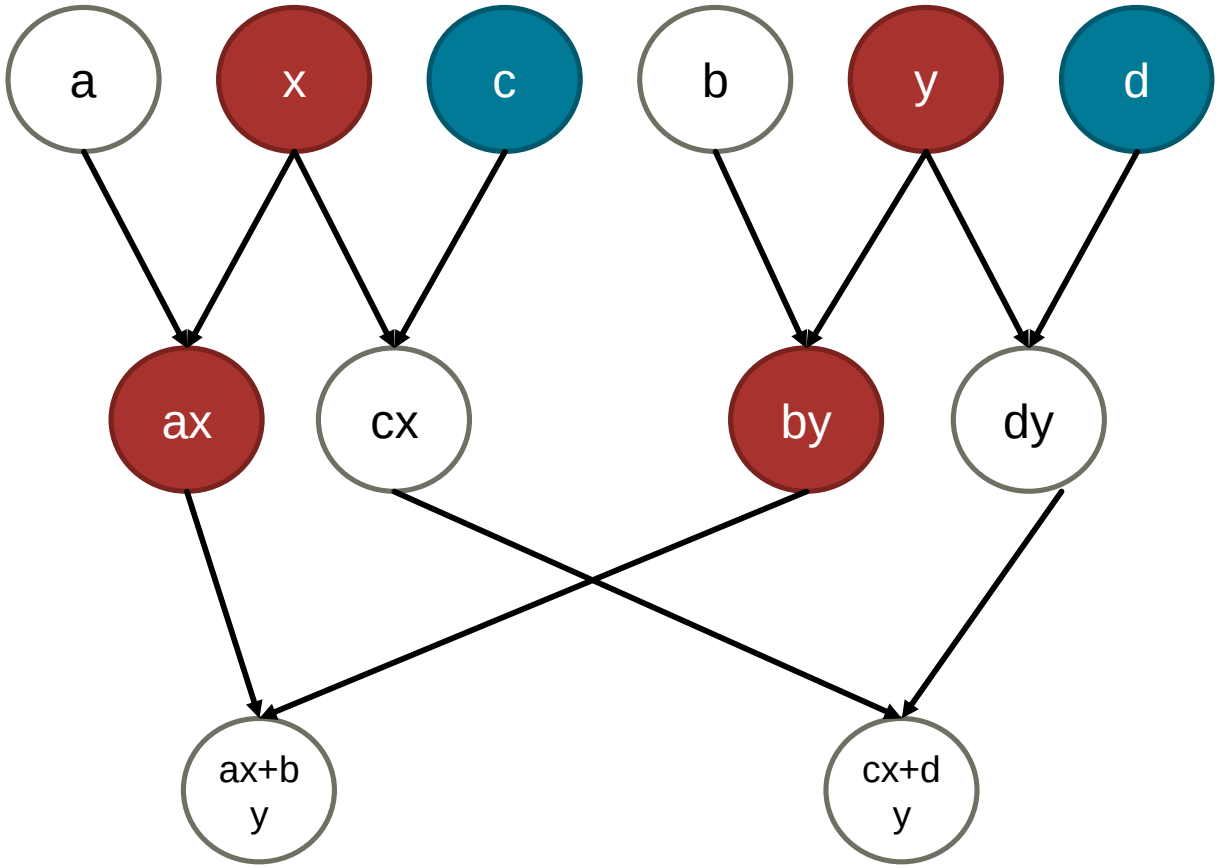
S = 5

I/O transfers: 4



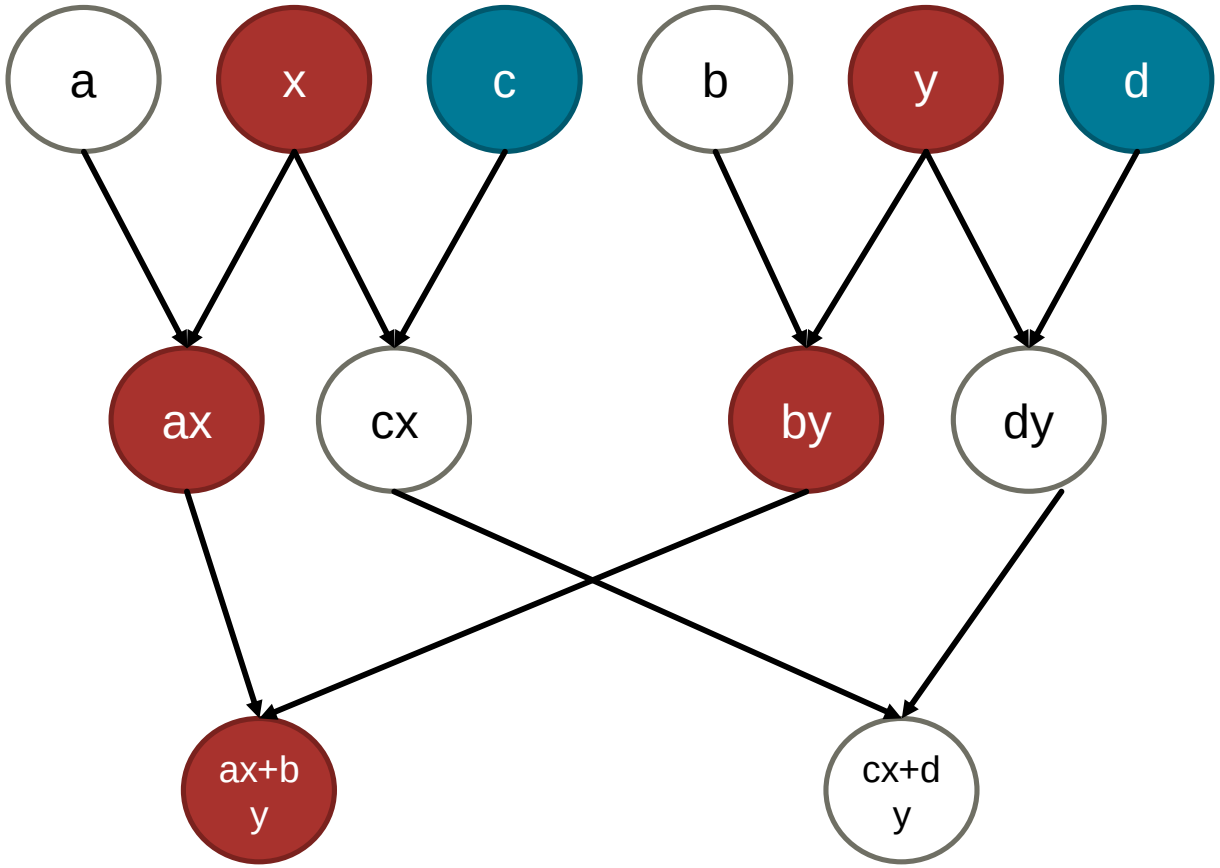
S = 5

I/O transfers: 4



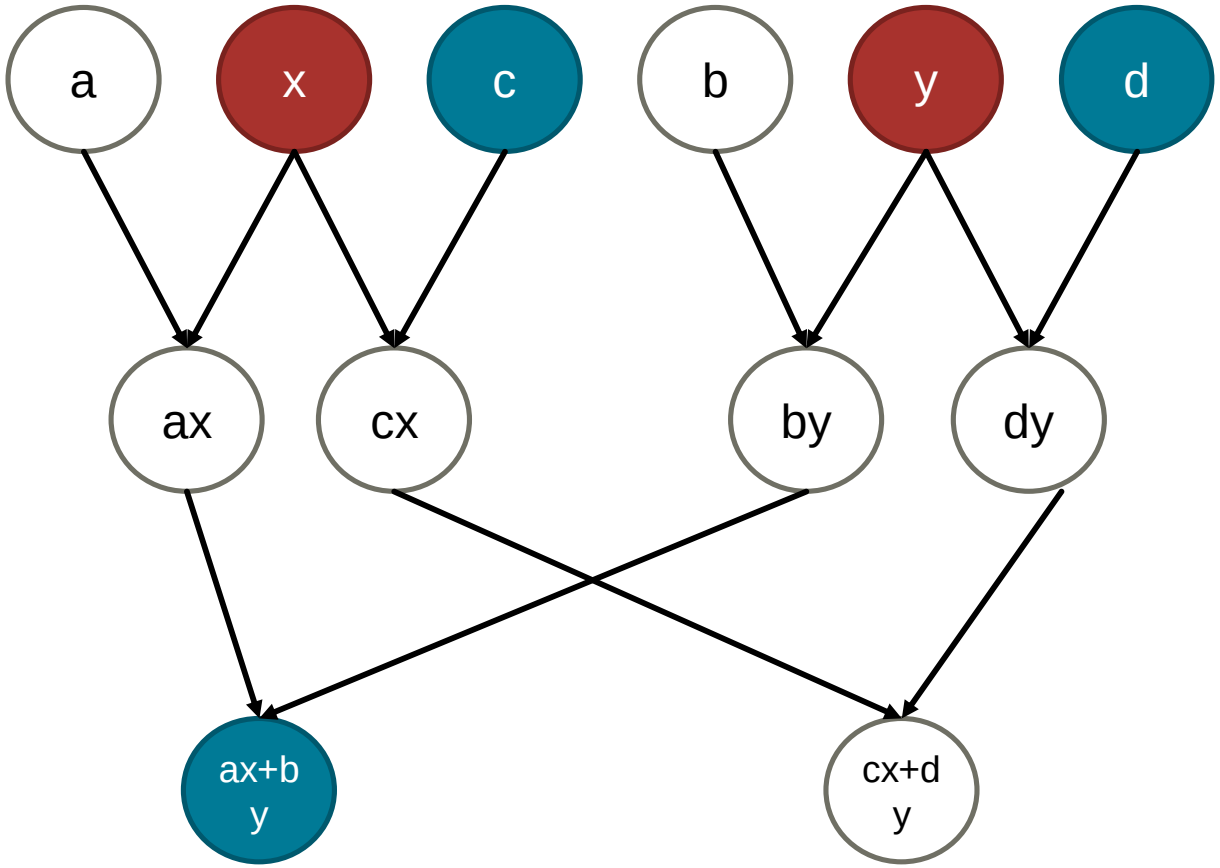
S = 5

I/O transfers: 4



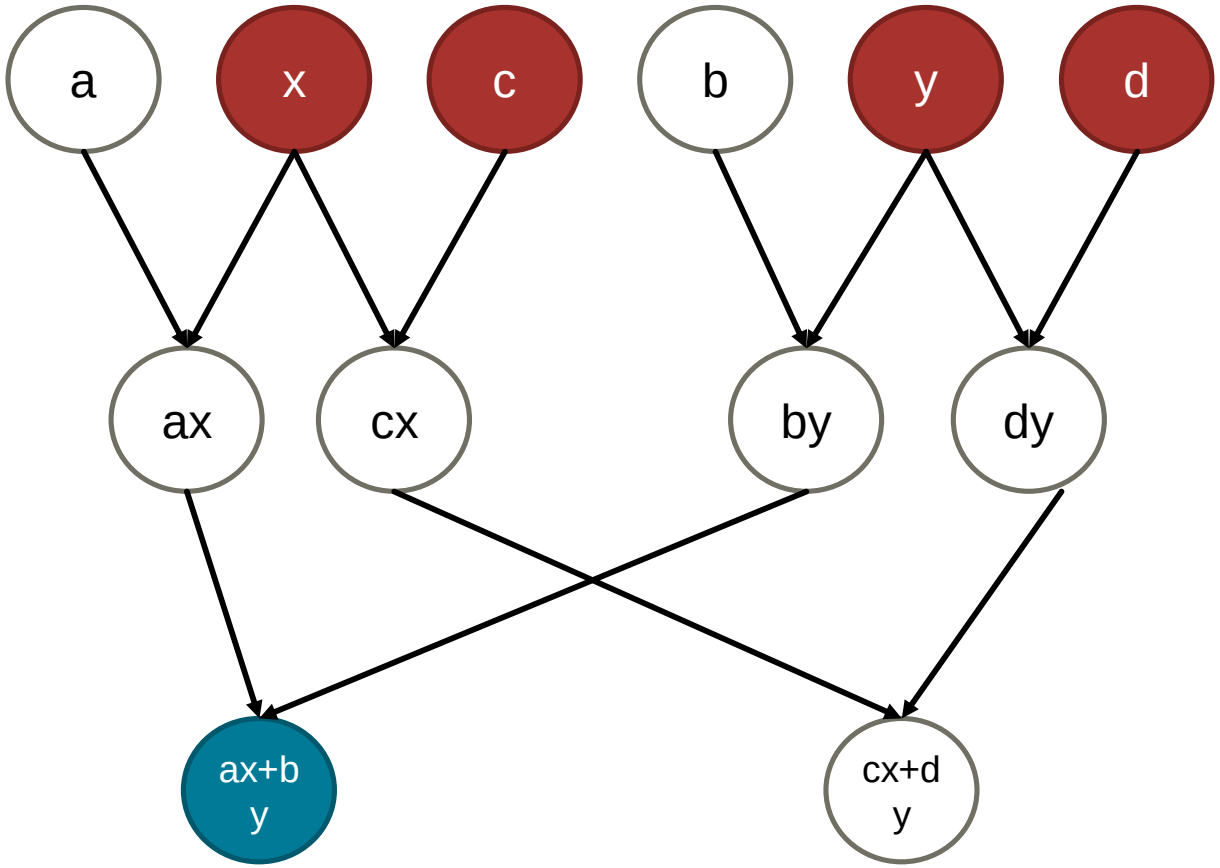
S = 5

I/O transfers: 5



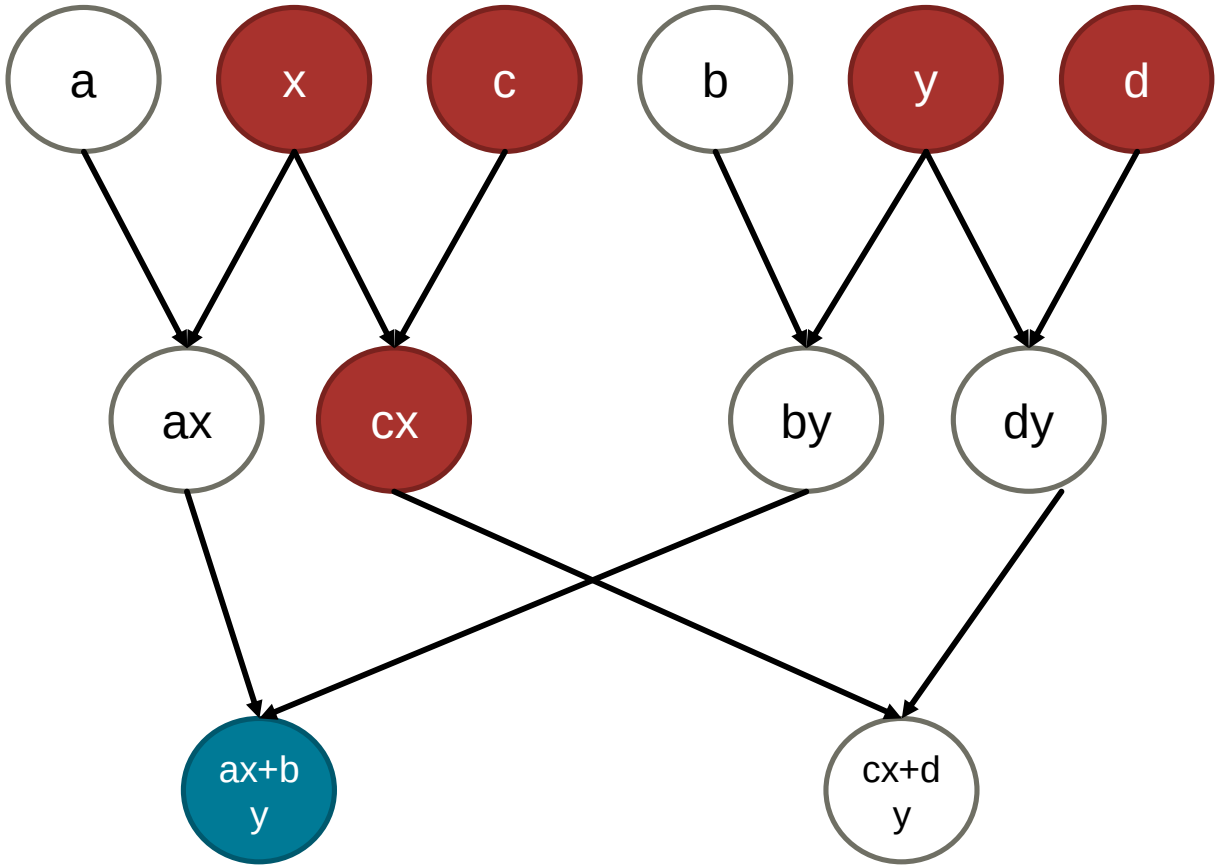
S = 5

I/O transfers: 7



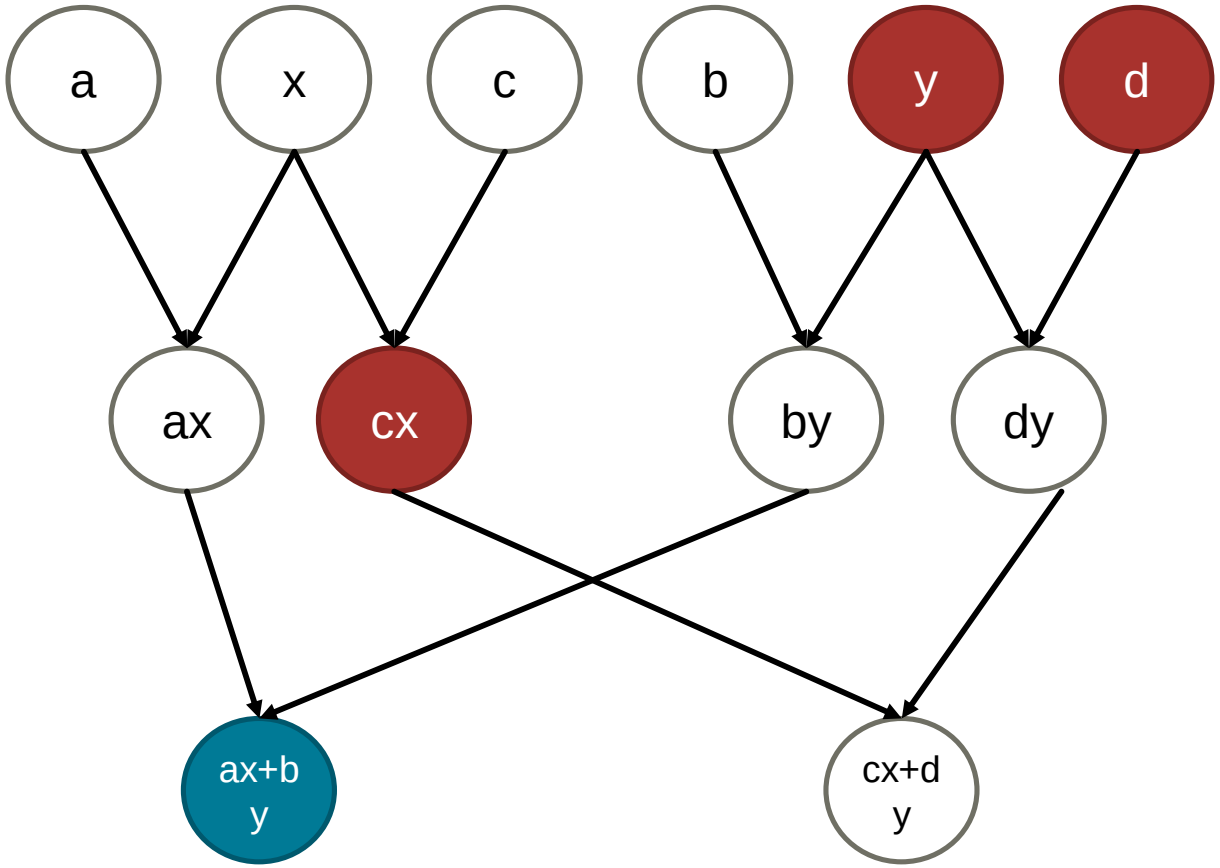
S = 5

I/O transfers: 7



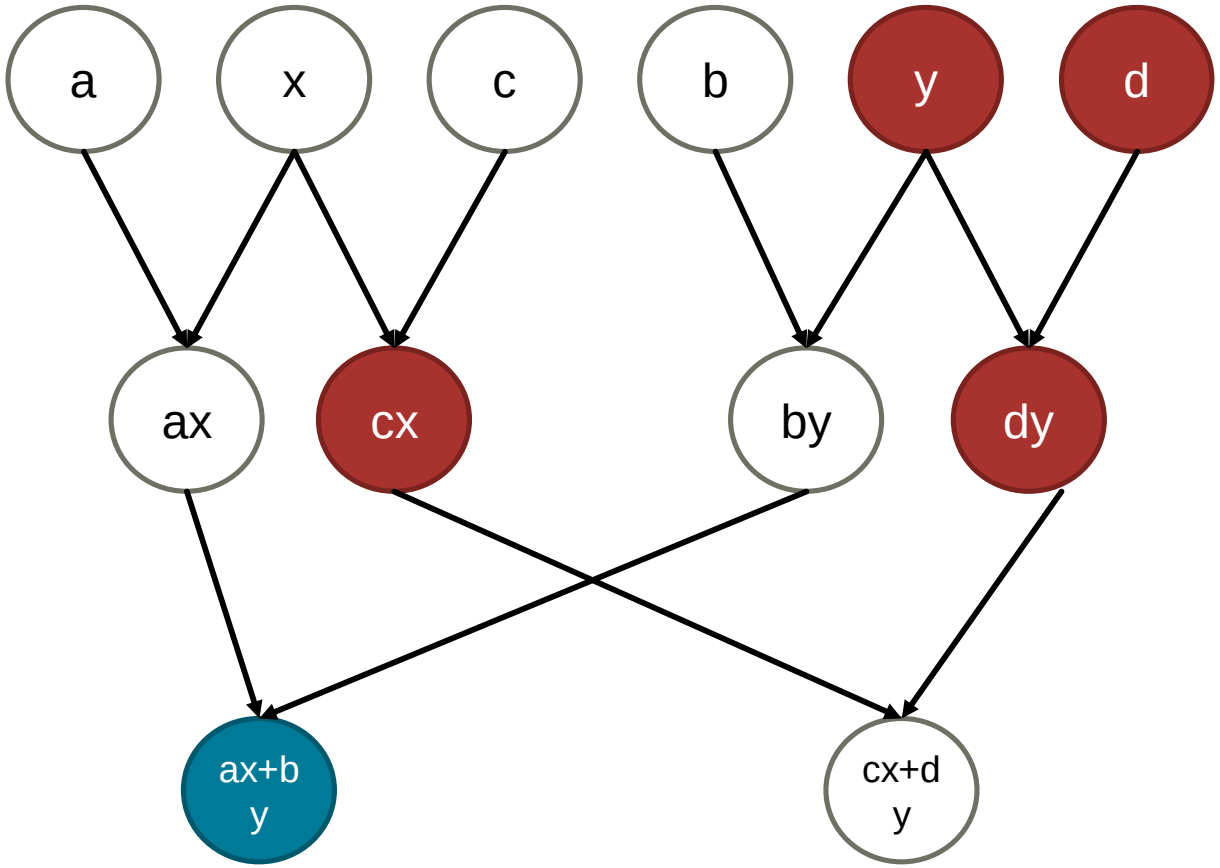
S = 5

I/O transfers: 7



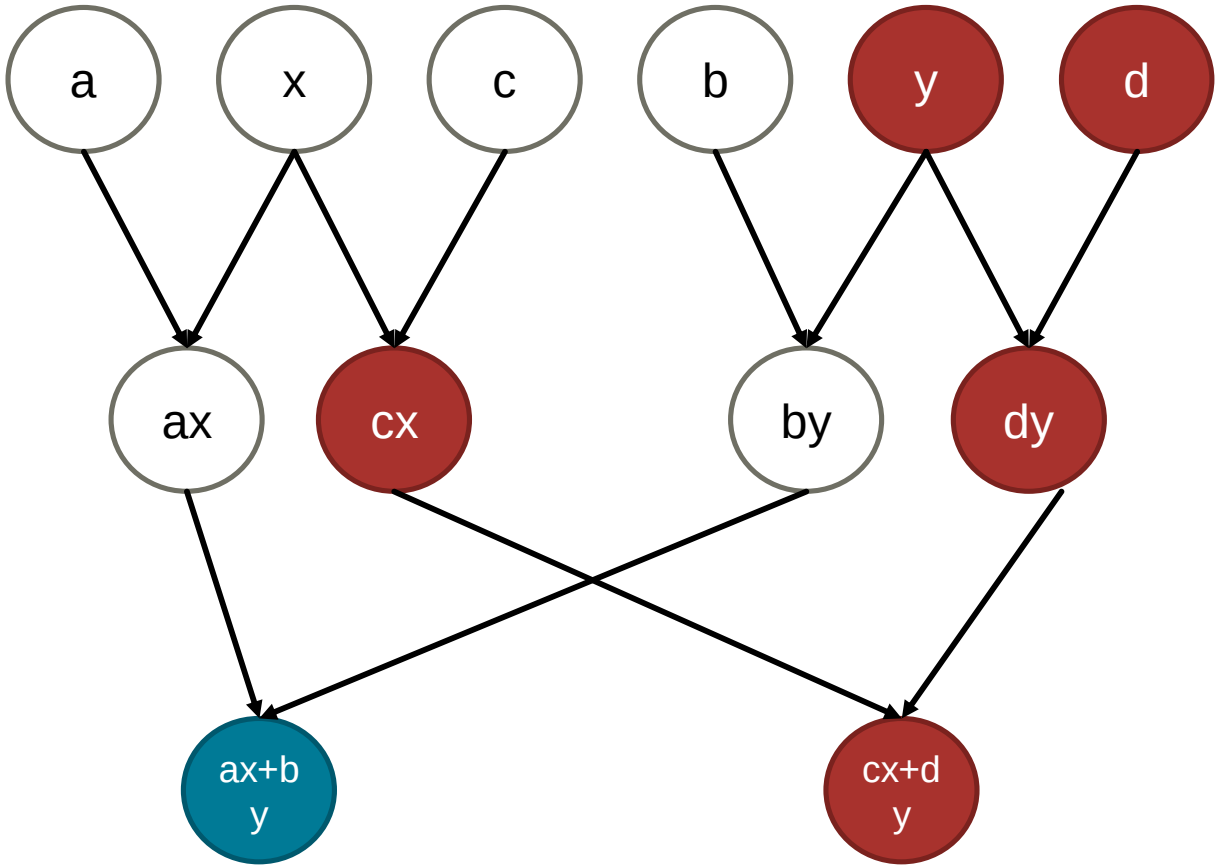
S = 5

I/O transfers: 7



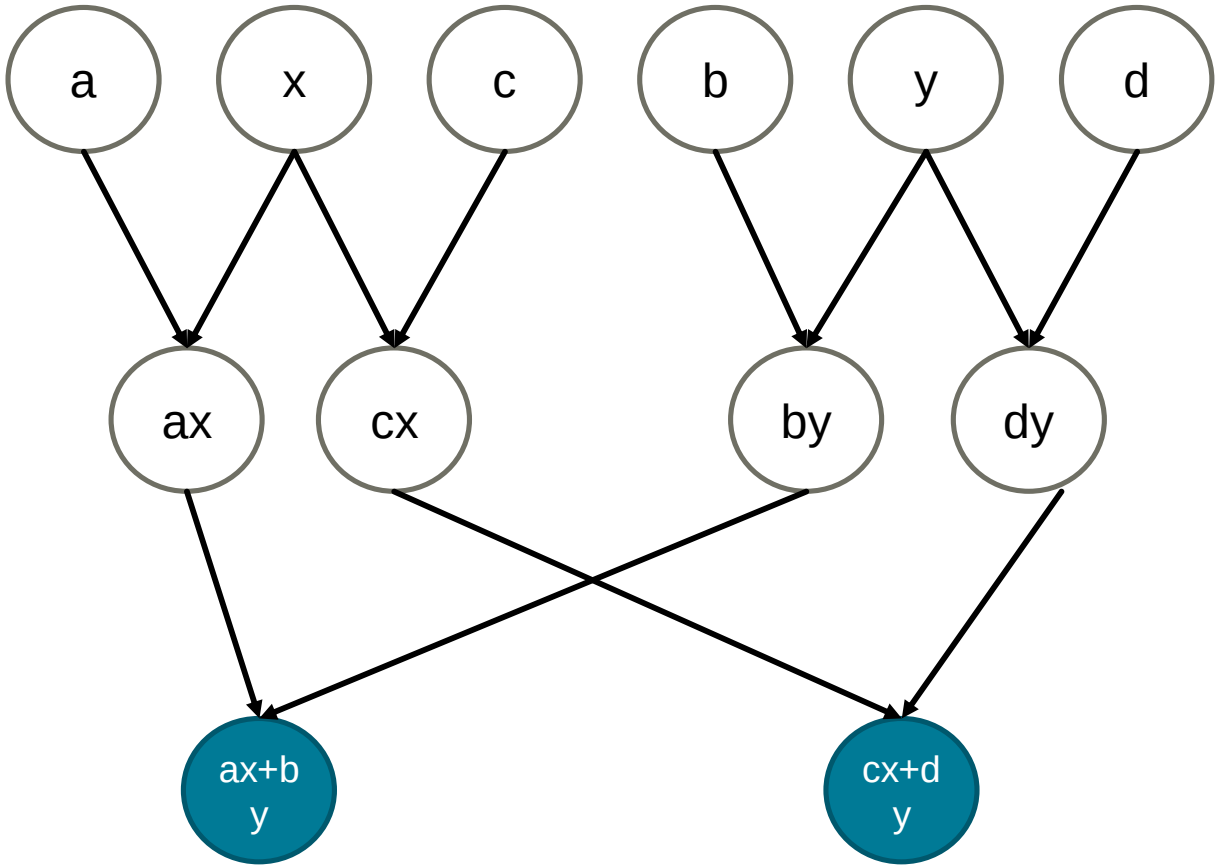
S = 5

I/O transfers: 7



S = 5

I/O transfers: 8

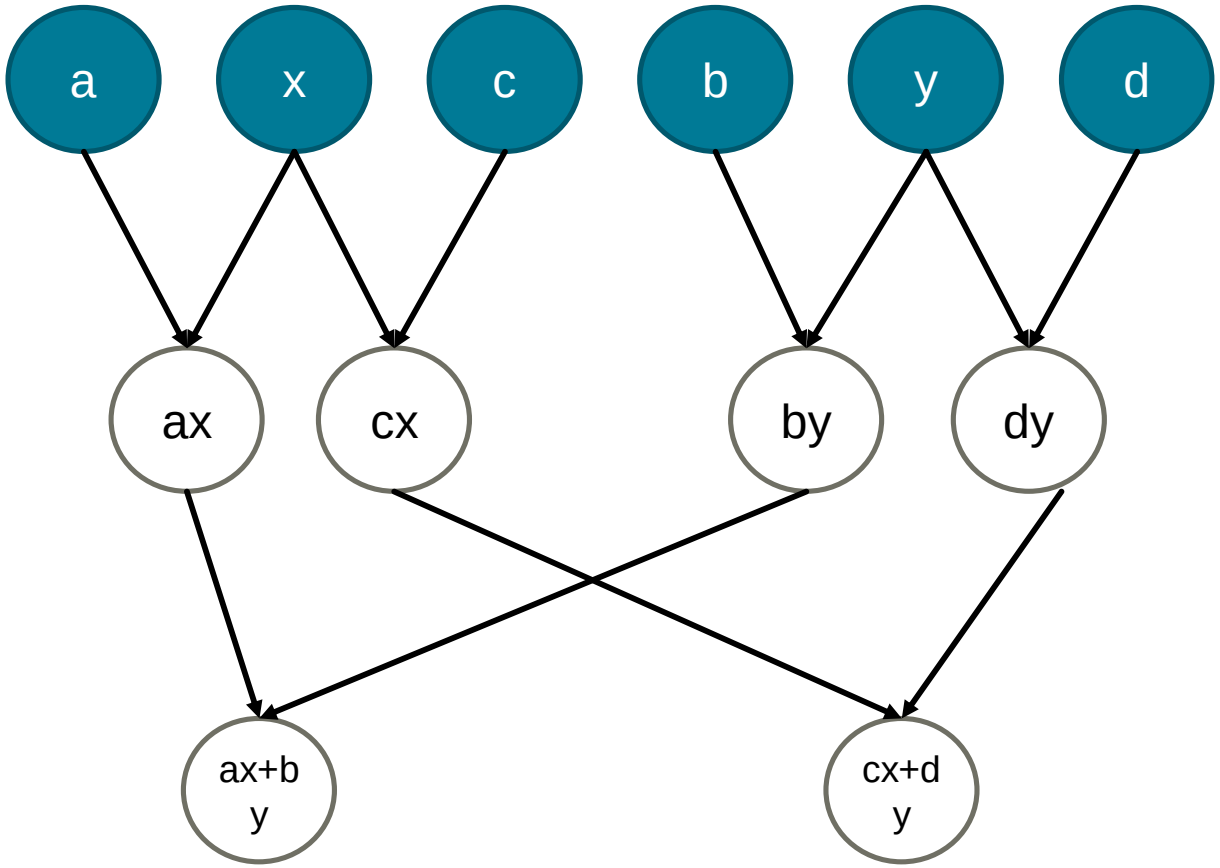


Exercise

- **Same as before but with S=4.**

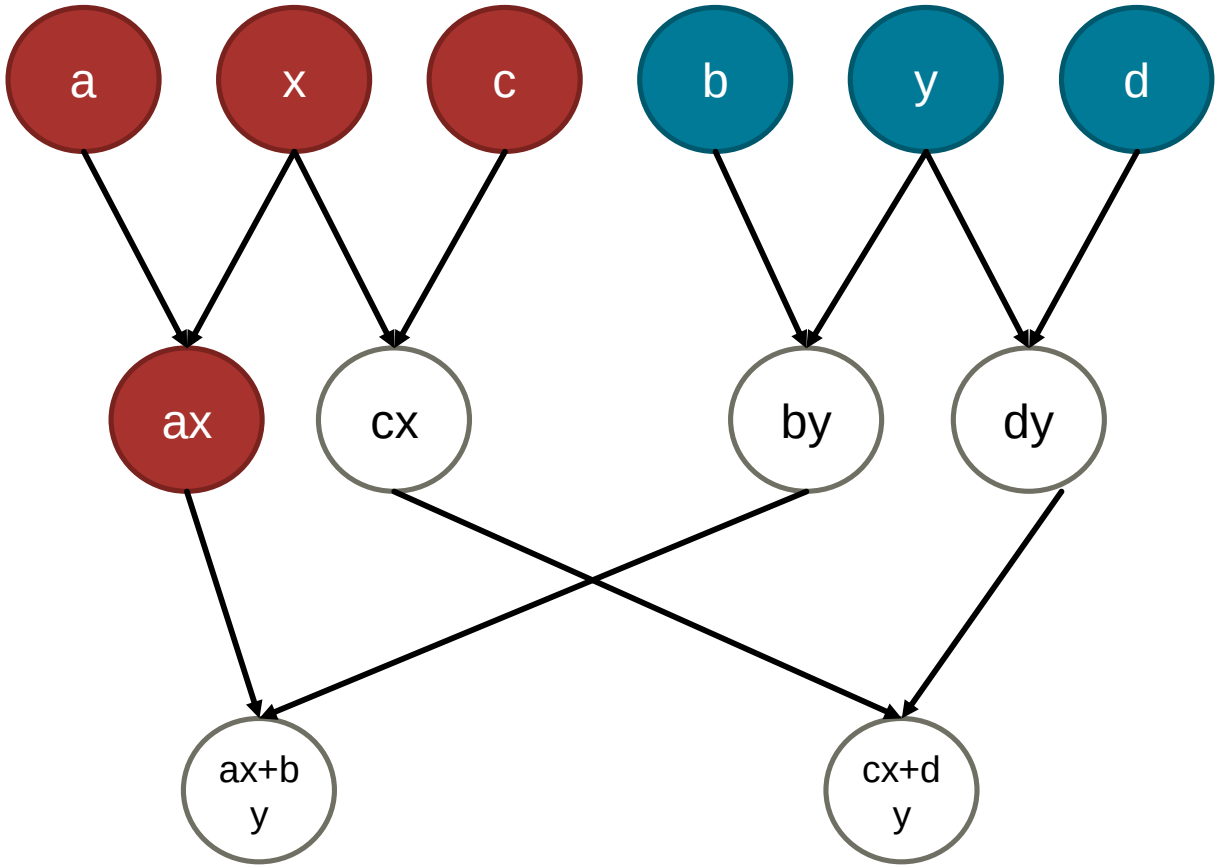
S = 4

I/O transfers: 0



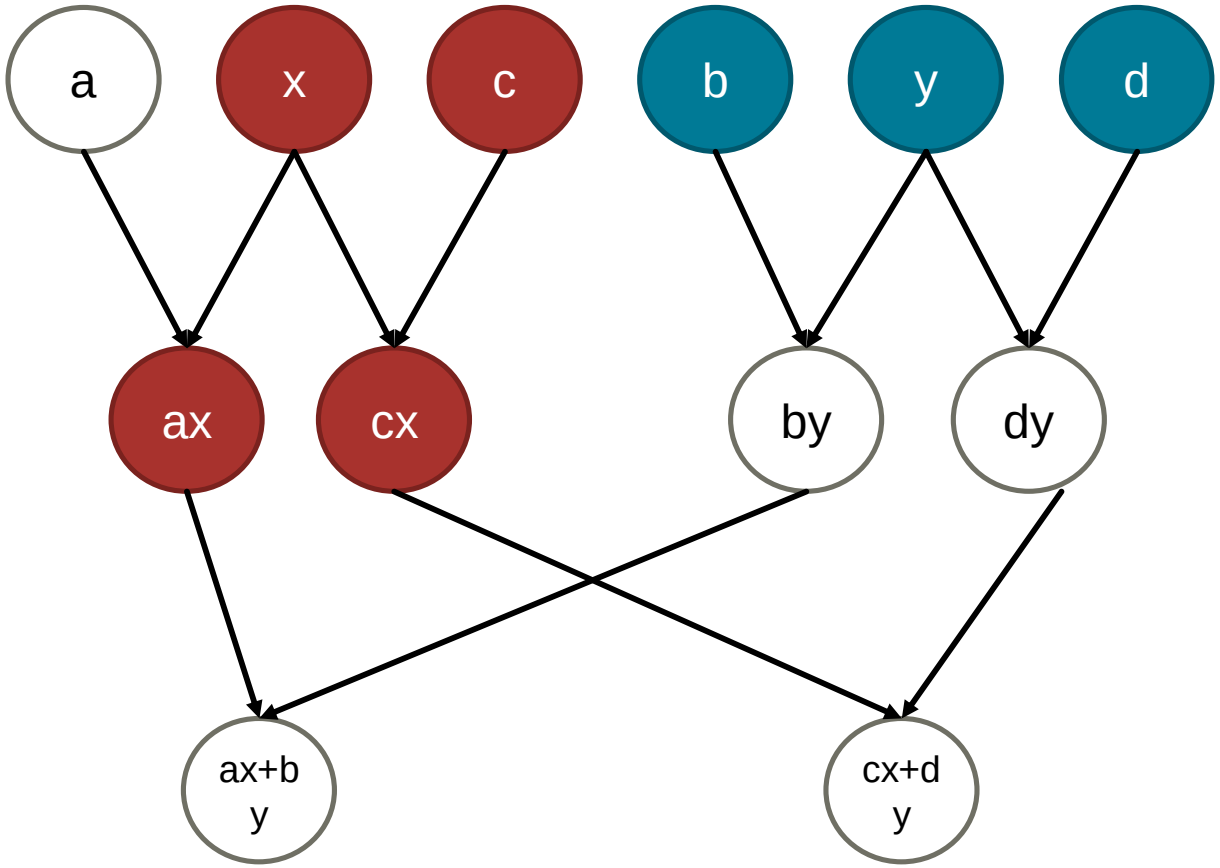
S = 4

I/O transfers: 3



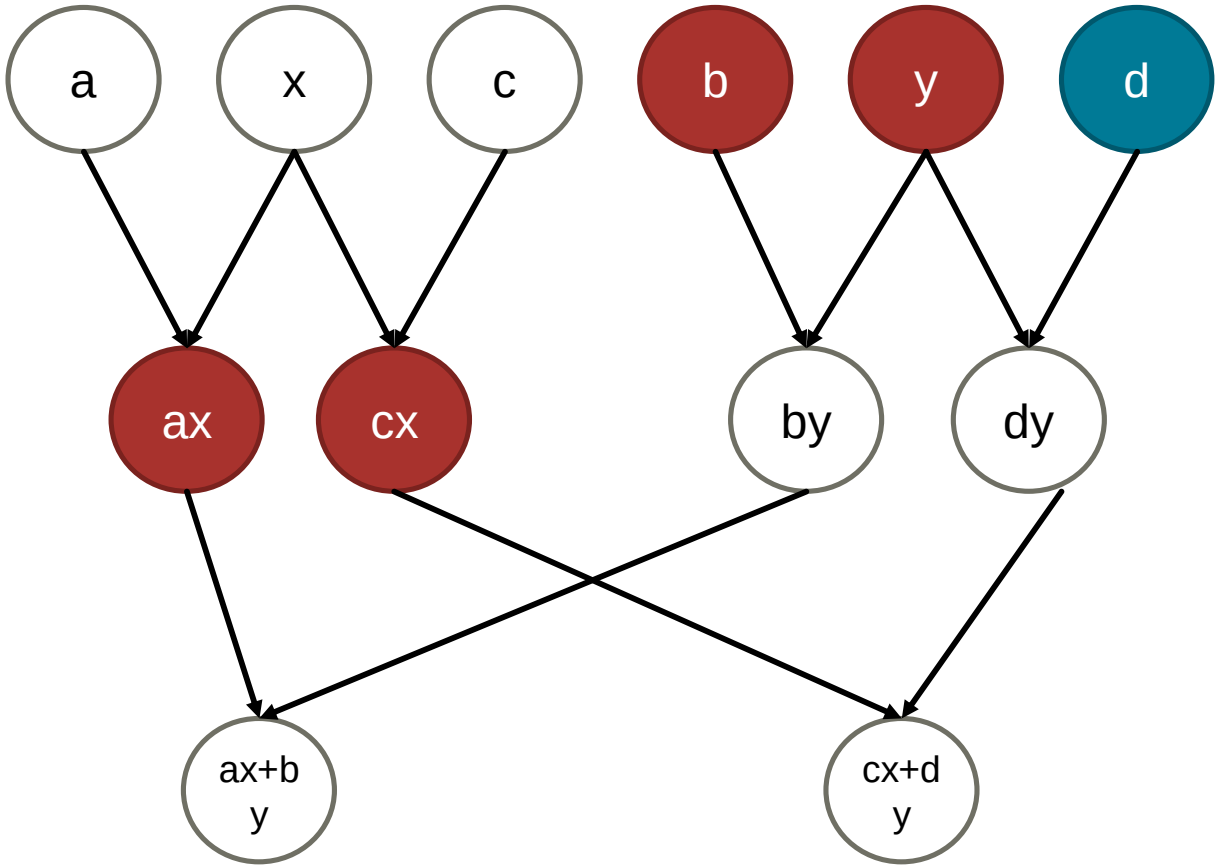
S = 4

I/O transfers: 3



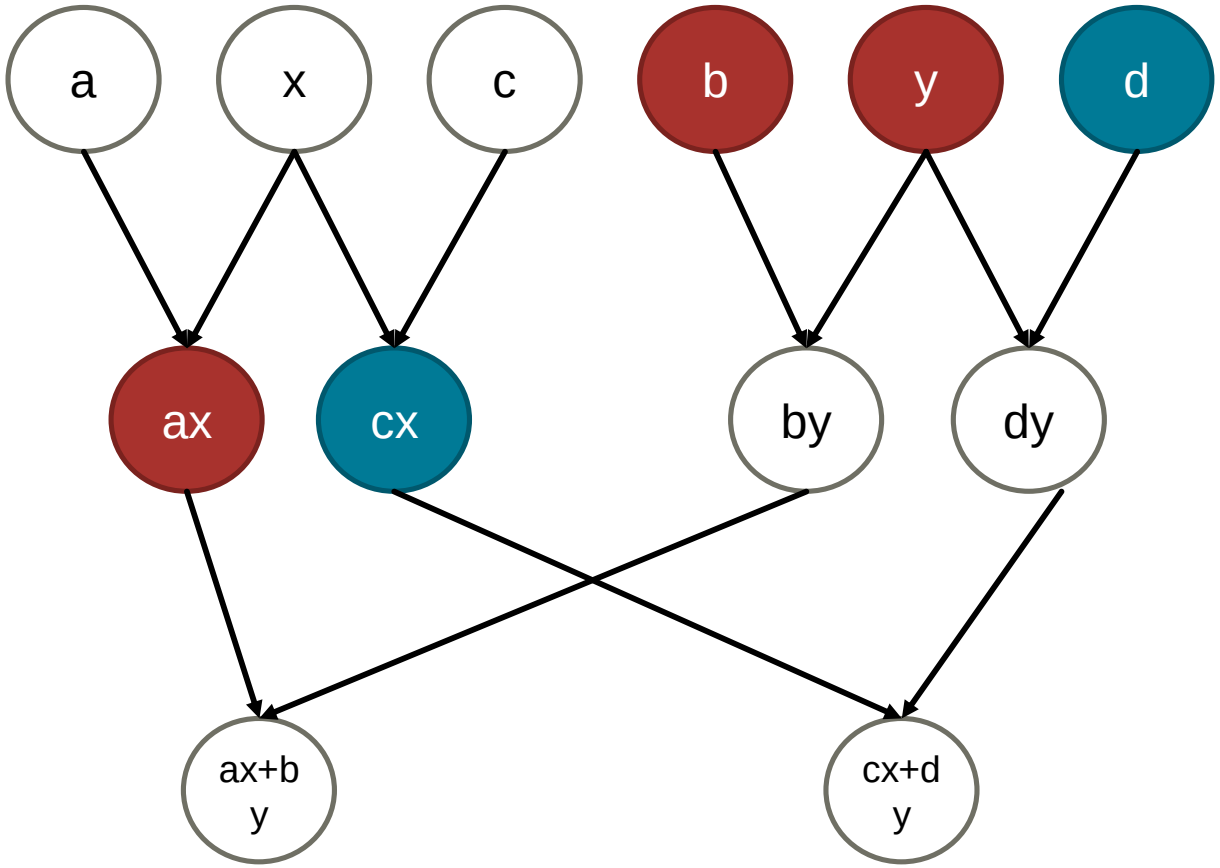
S = 4

I/O transfers: 5



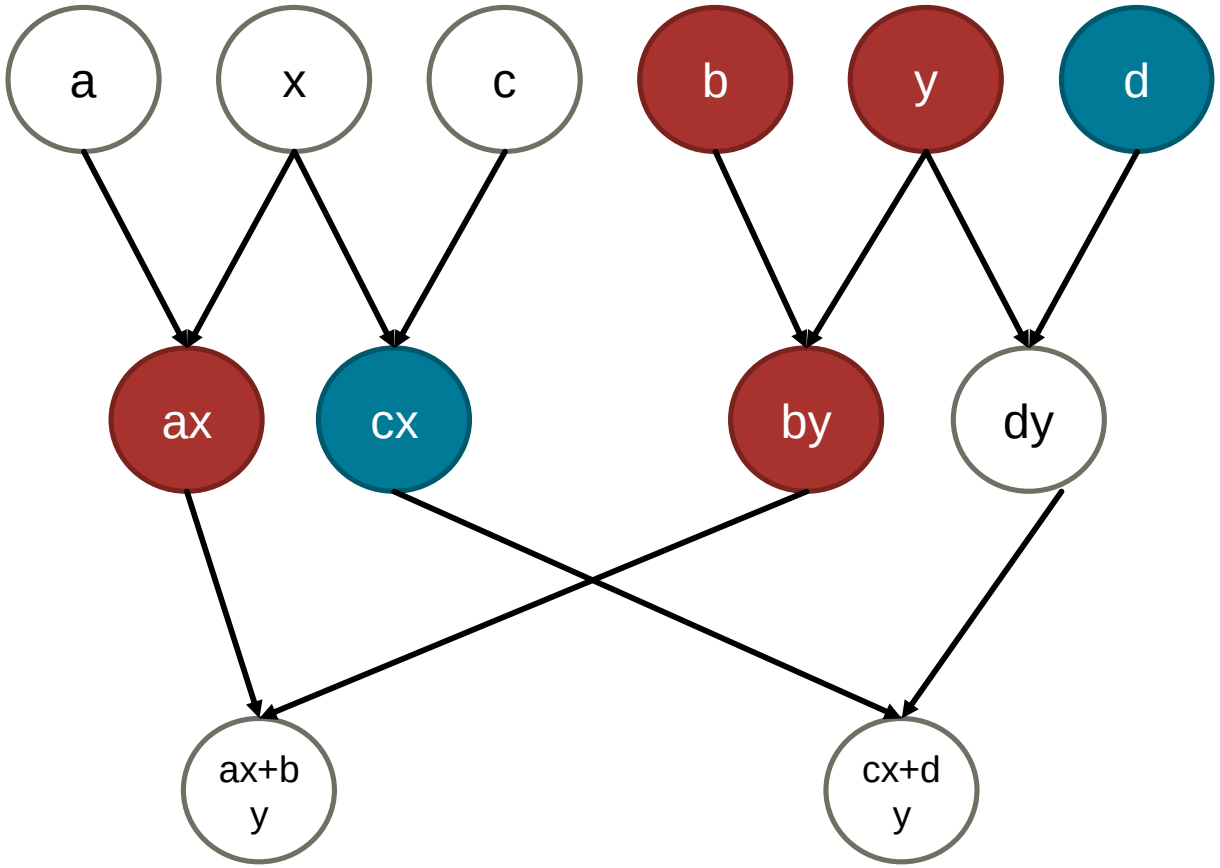
S = 4

I/O transfers: 6



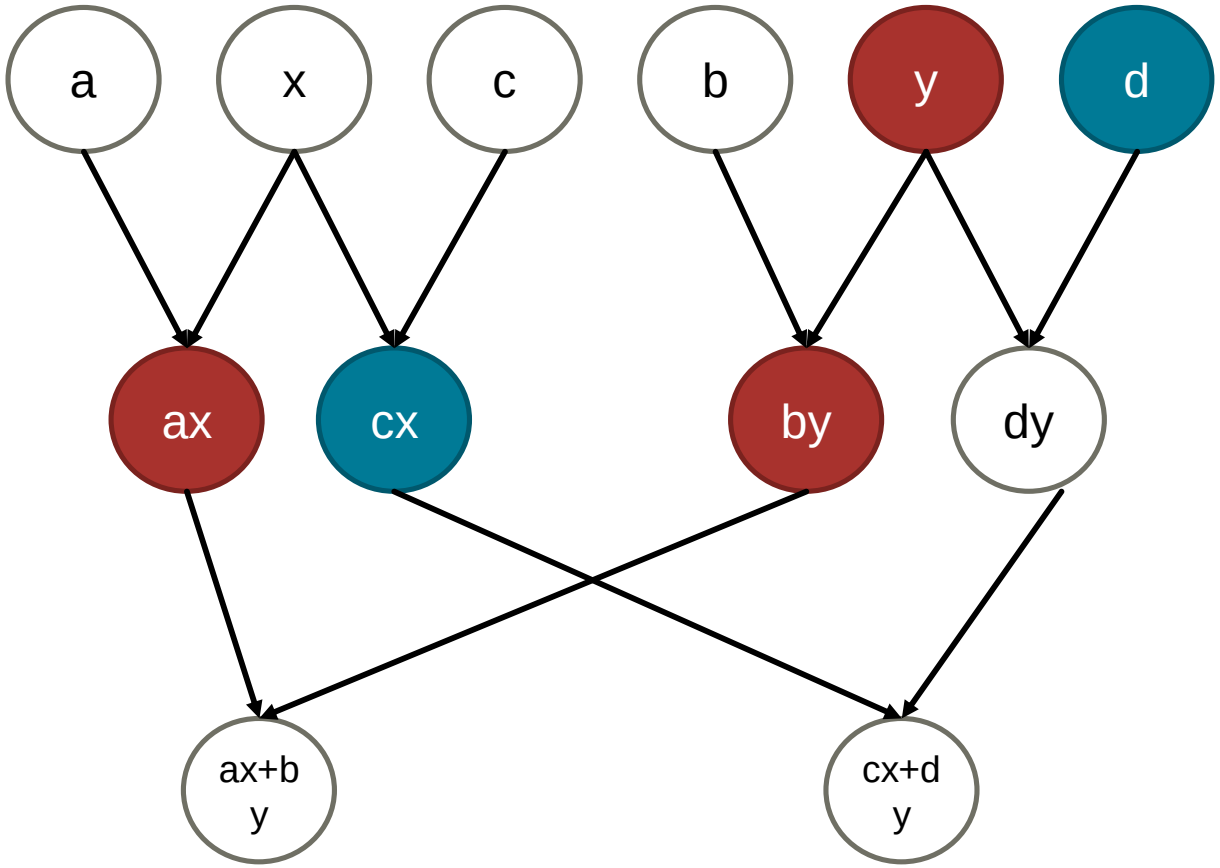
S = 4

I/O transfers: 6



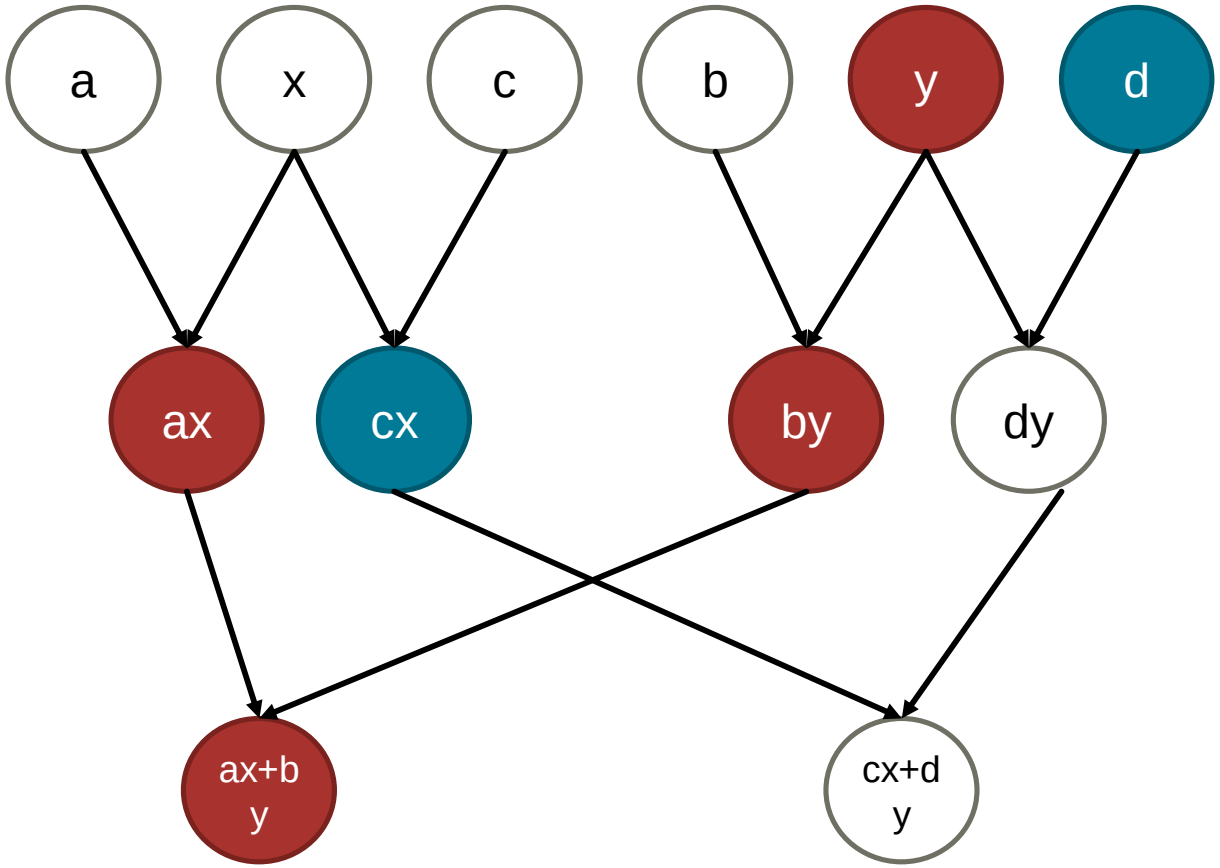
S = 4

I/O transfers: 6



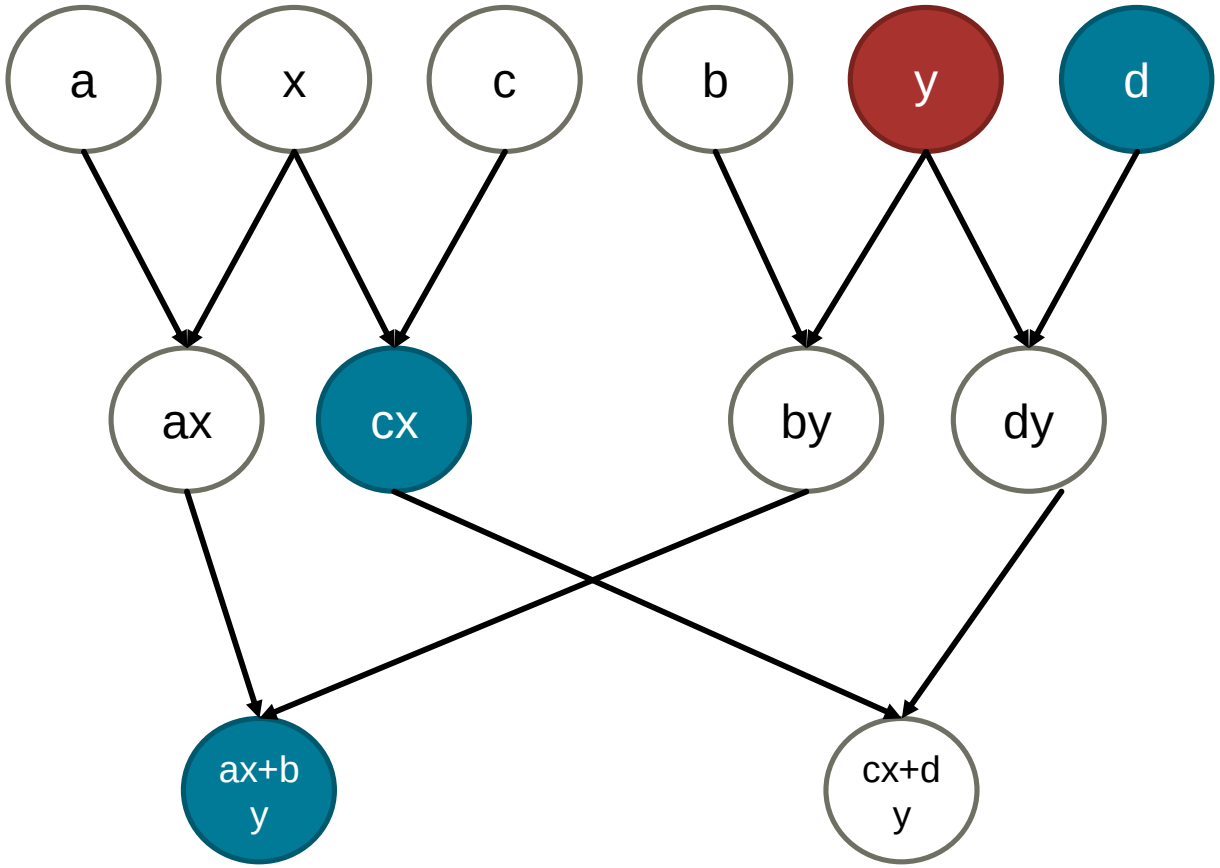
S = 4

I/O transfers: 6



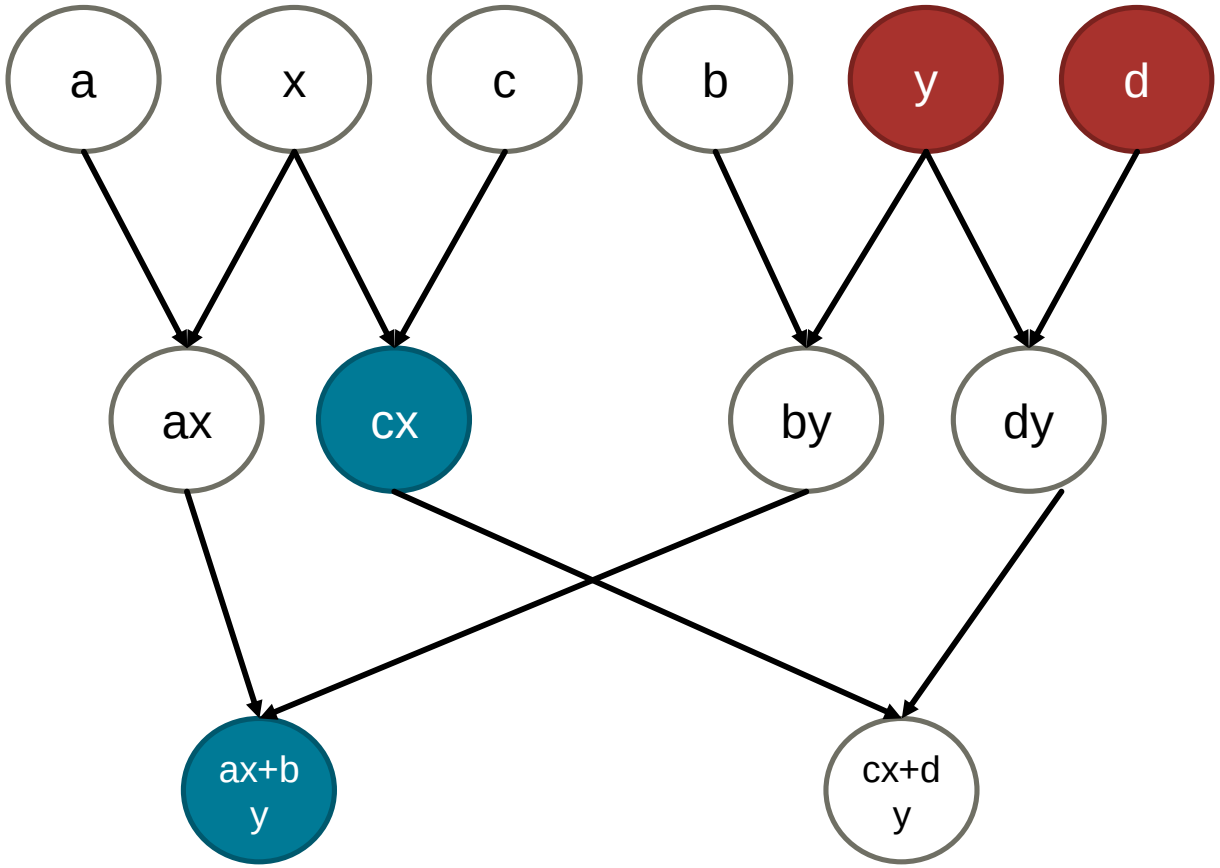
S = 4

I/O transfers: 7



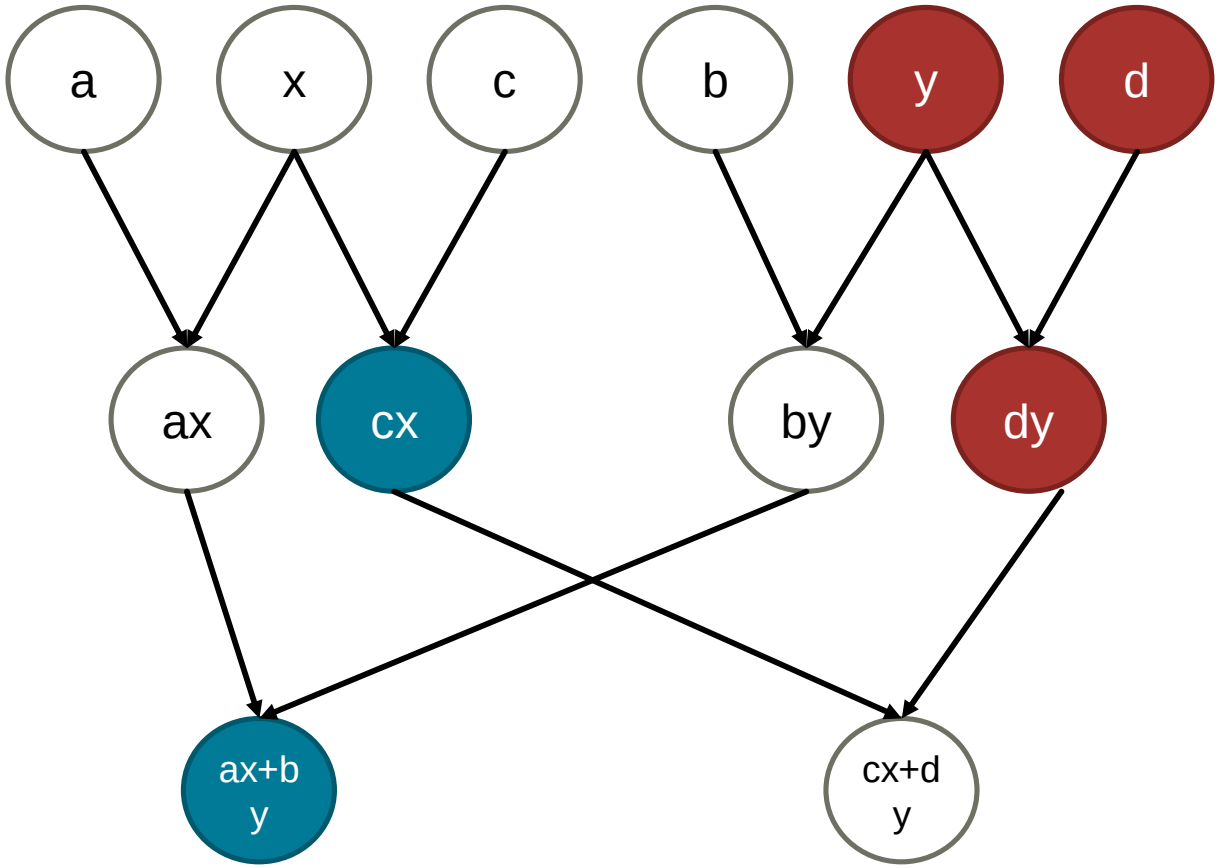
S = 4

I/O transfers: 8



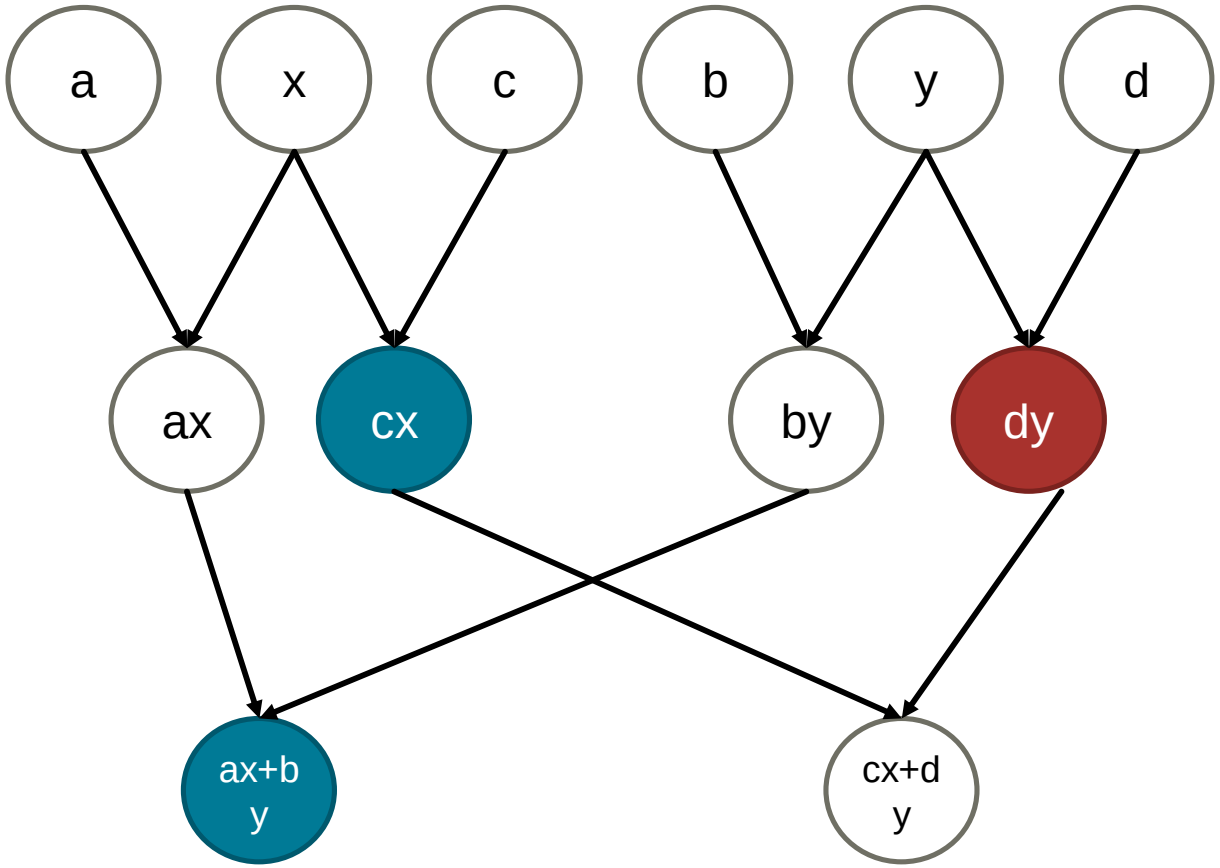
S = 4

I/O transfers: 8



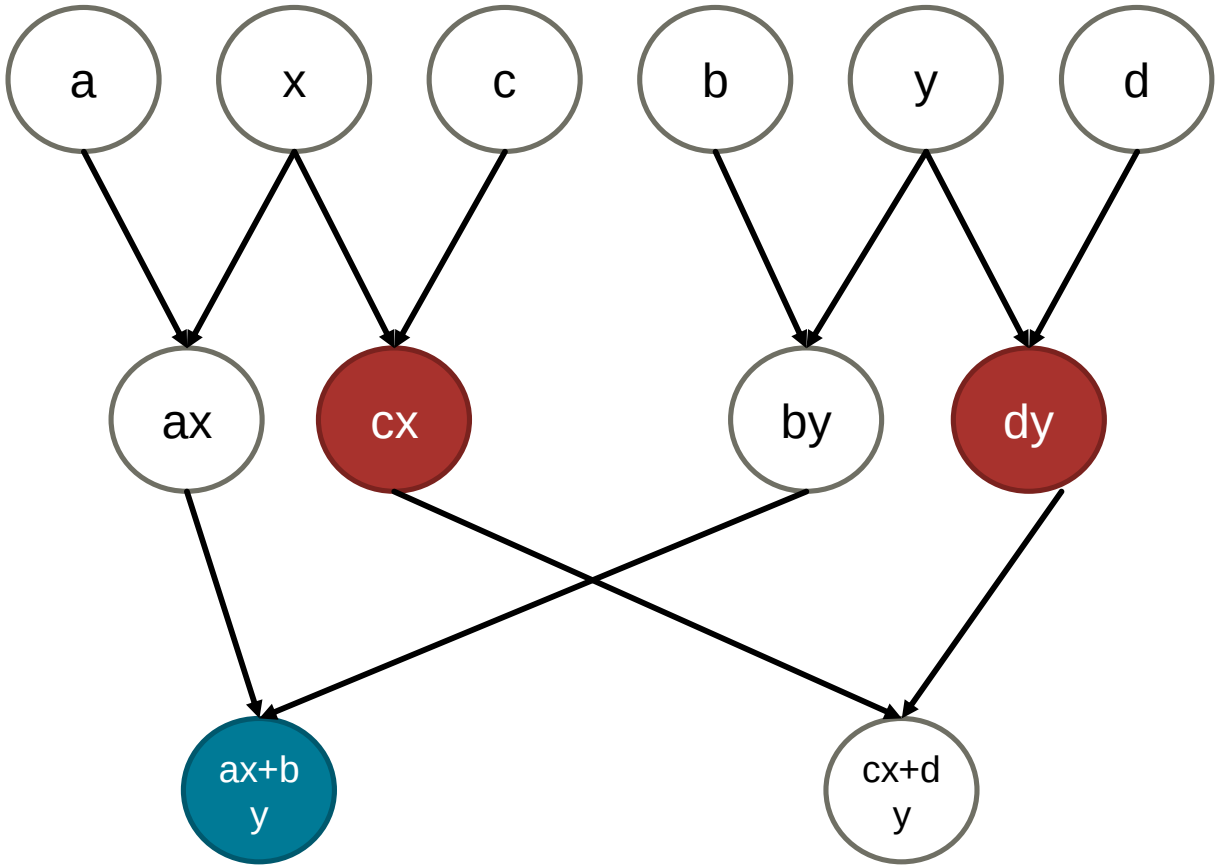
S = 4

I/O transfers: 8



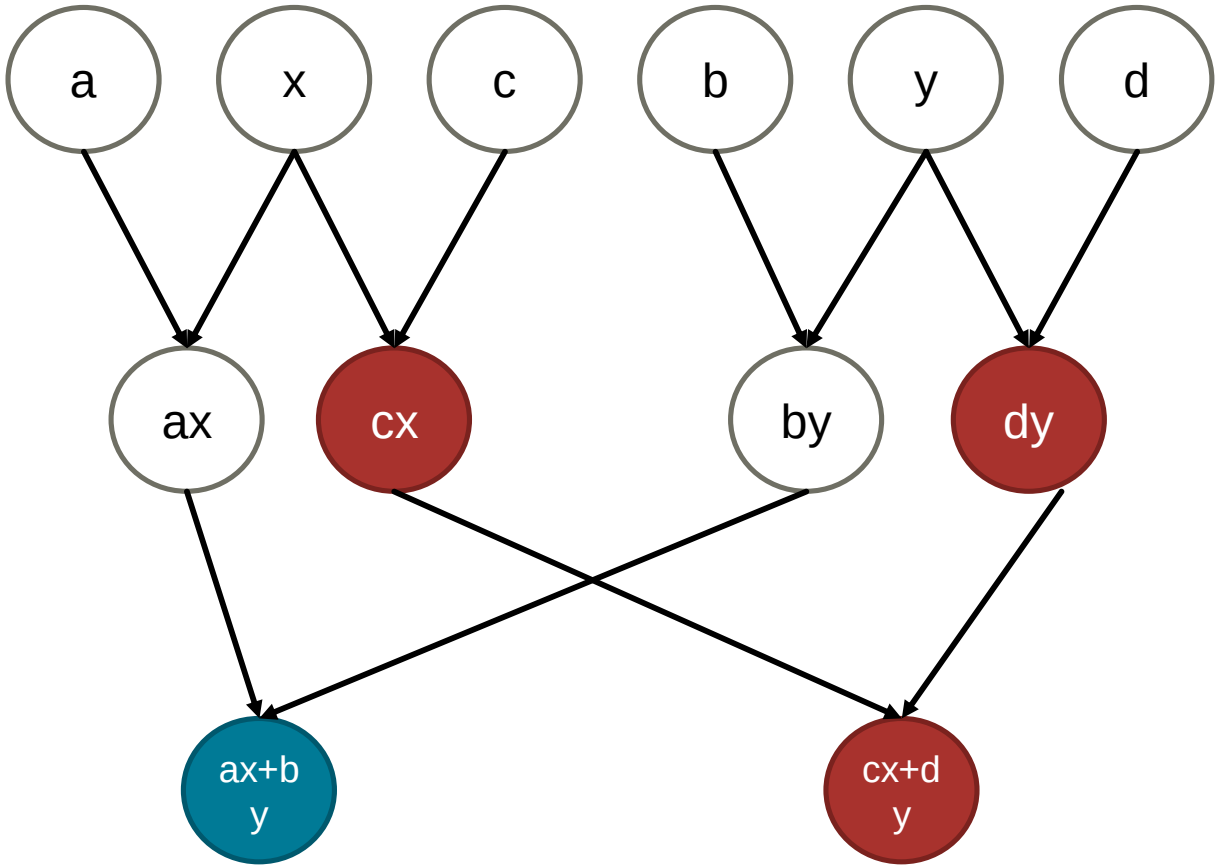
S = 4

I/O transfers: 9



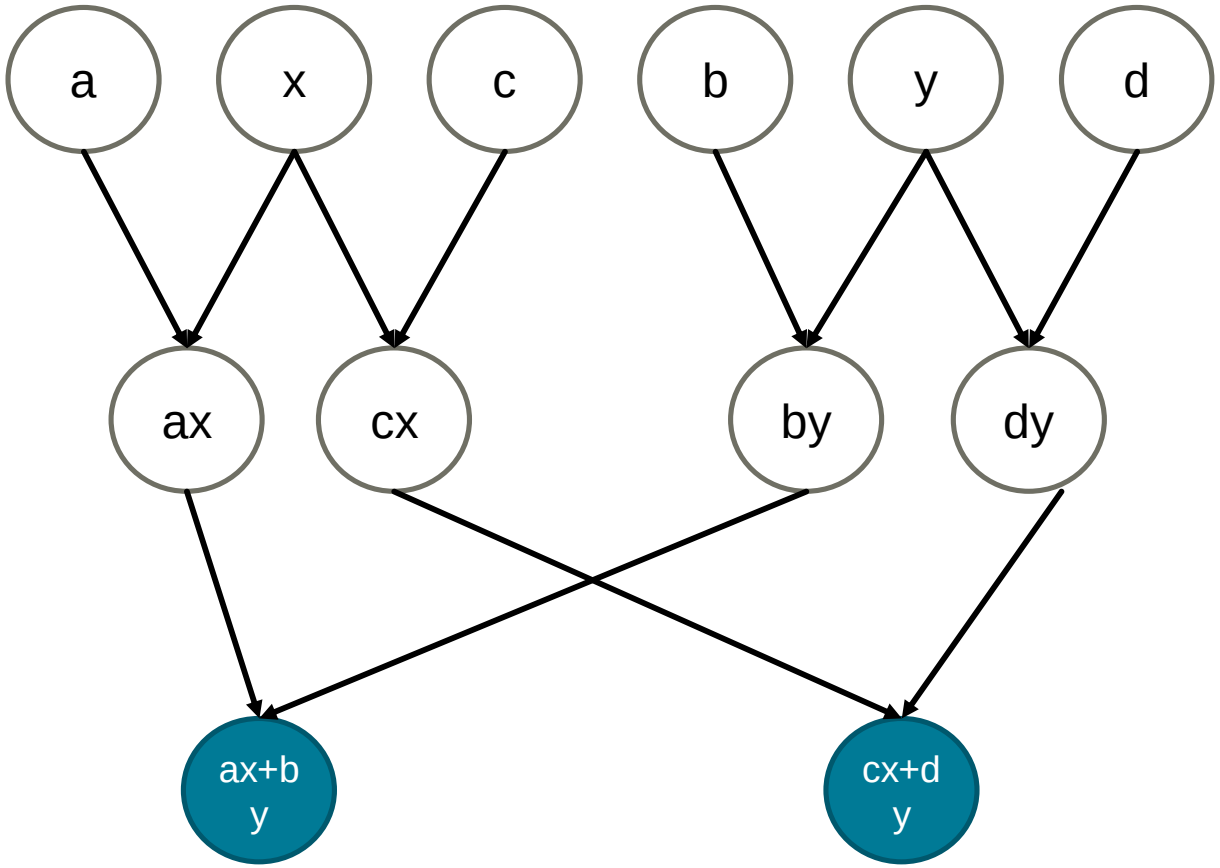
S = 4

I/O transfers: 9



S = 4

I/O transfers: 10

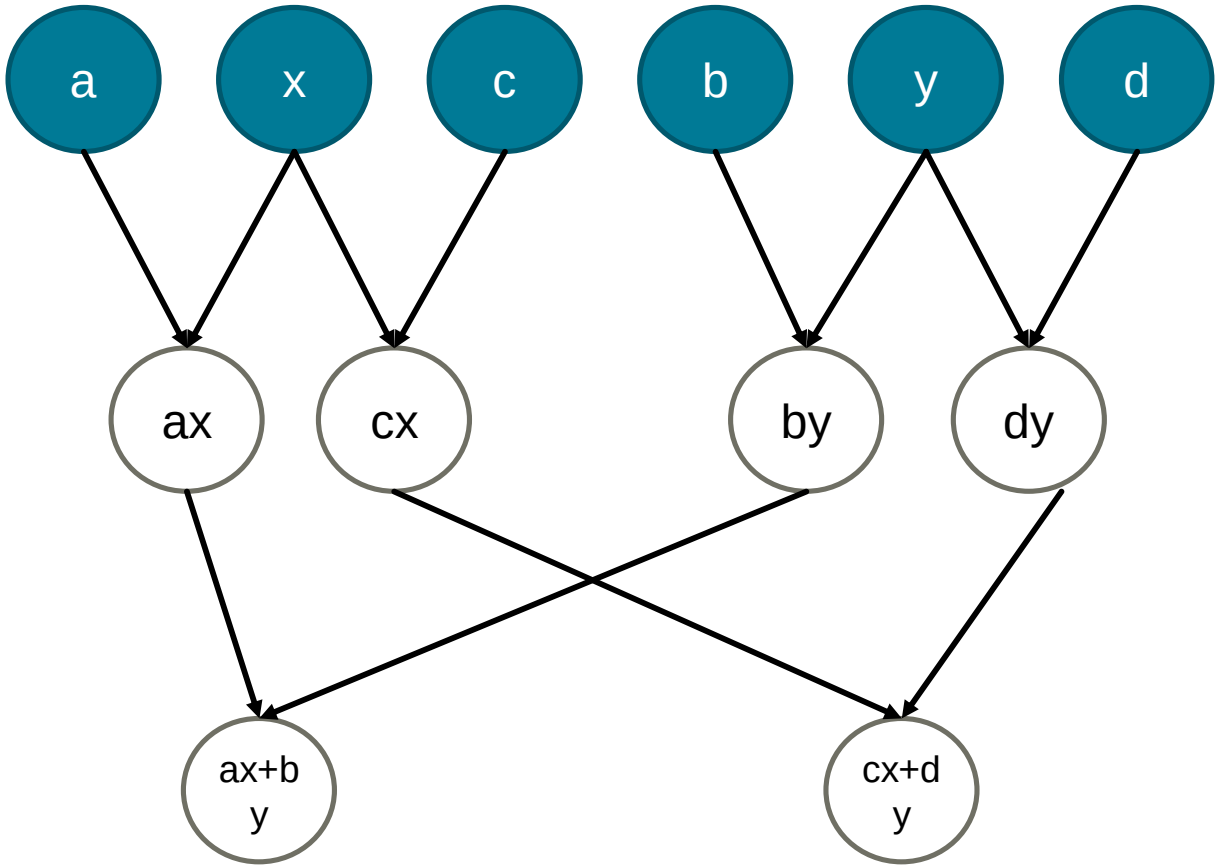


Exercise

- **Same as before but with S=3.**

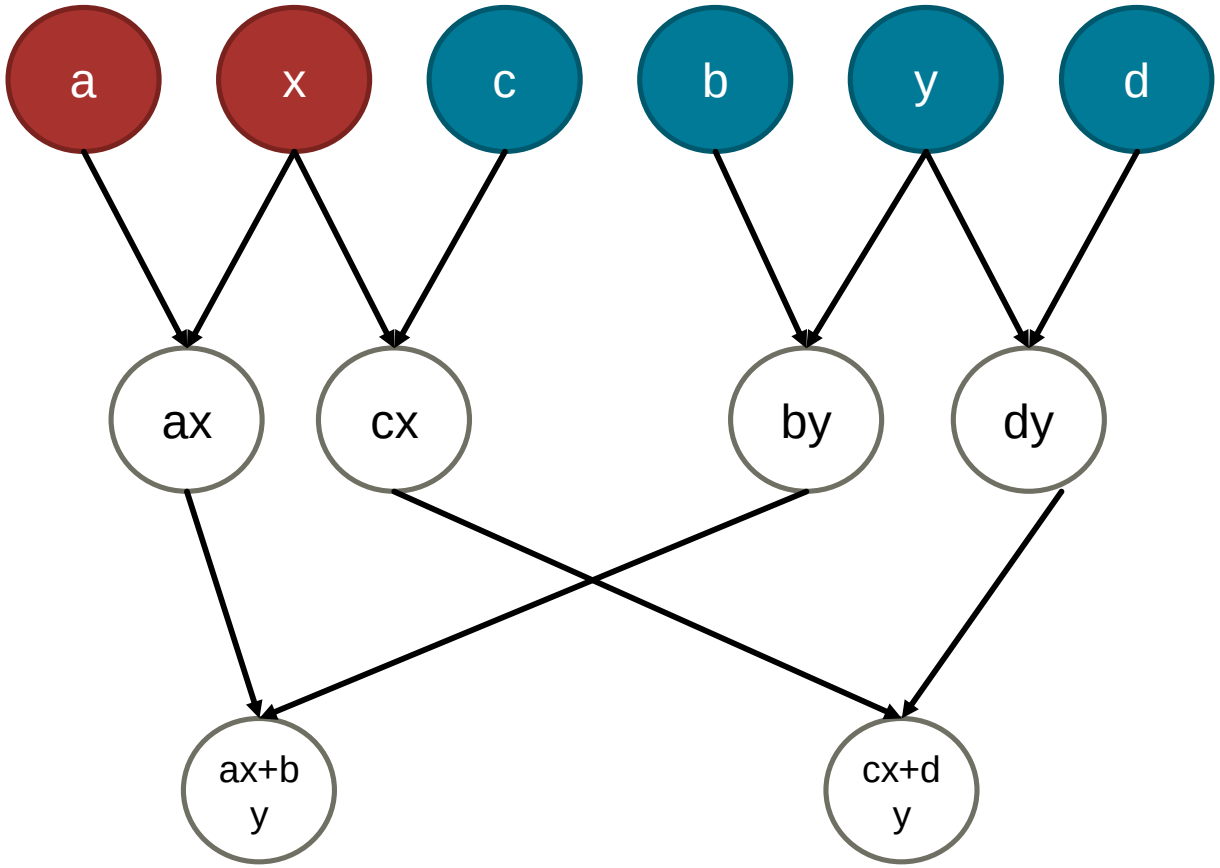
S = 3

I/O transfers: 0



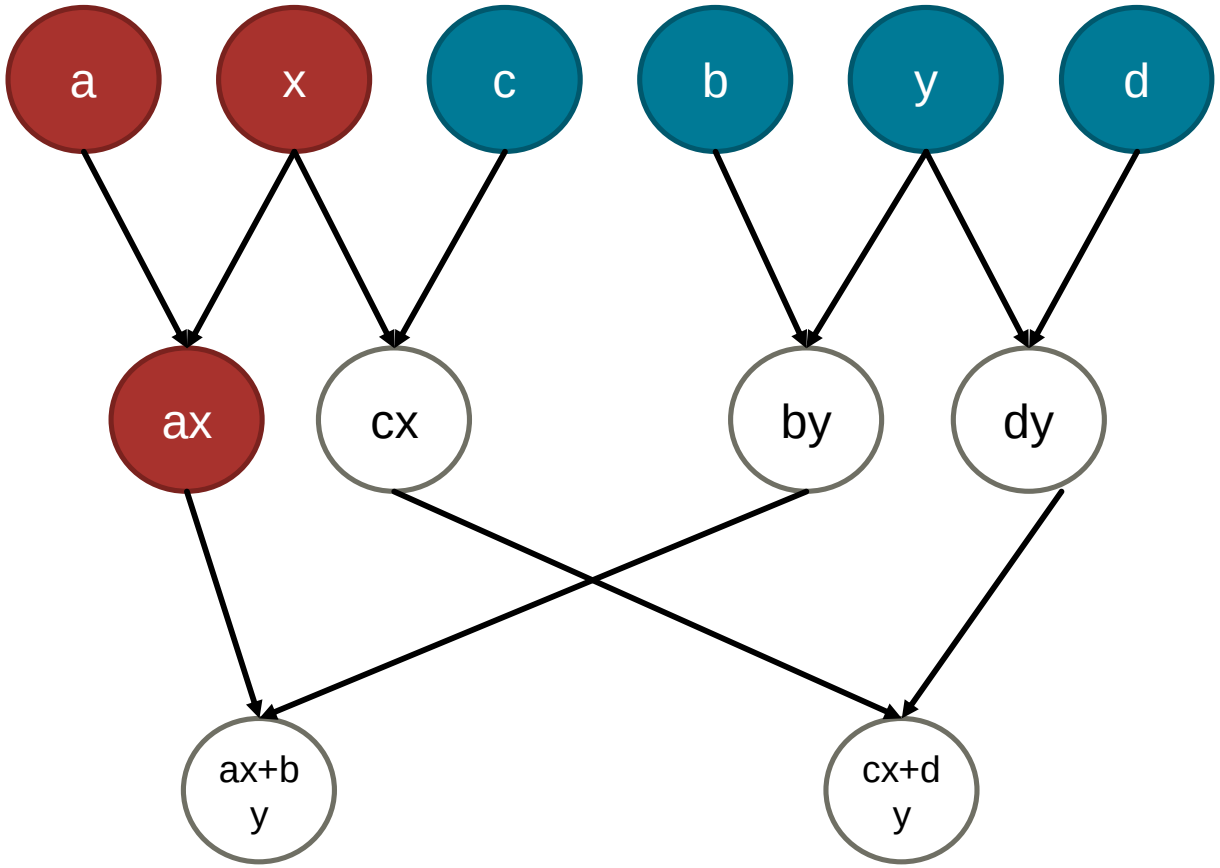
S = 3

I/O transfers: 2



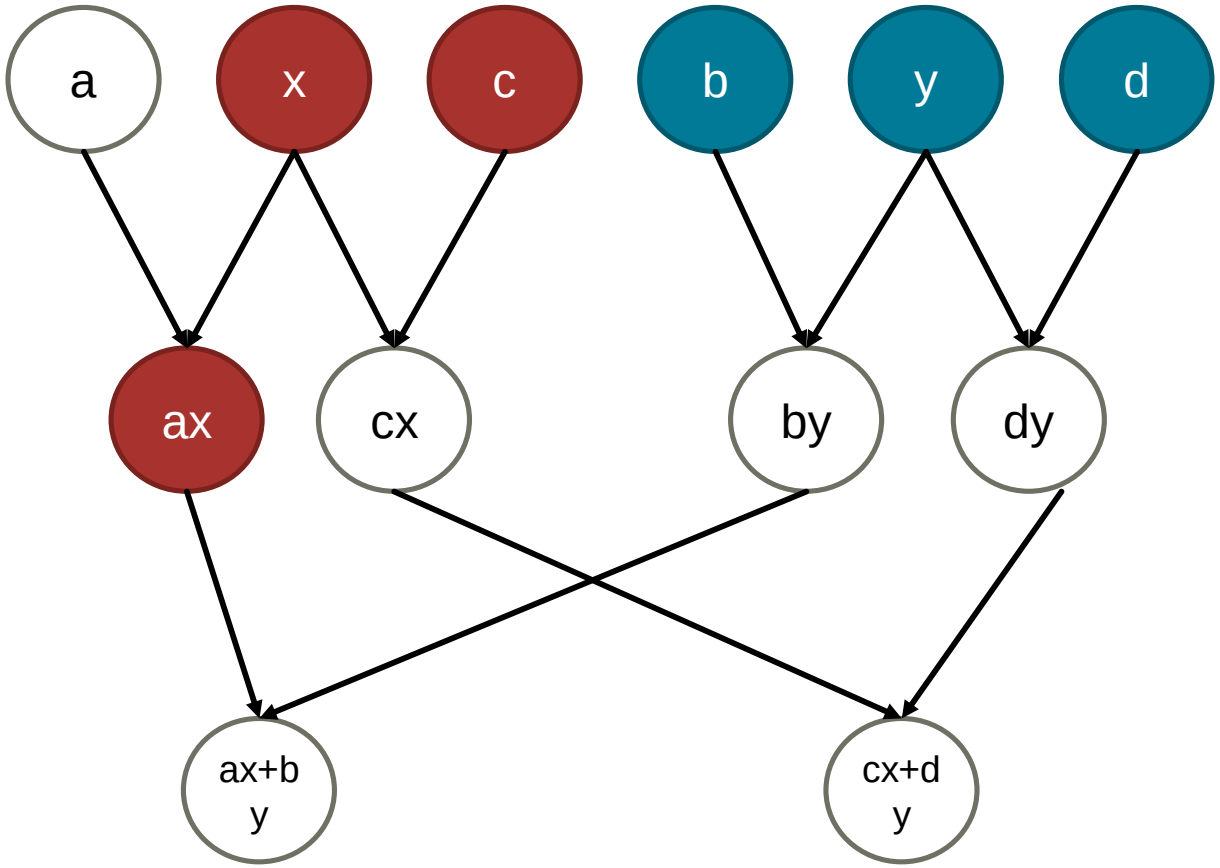
S = 3

I/O transfers: 2



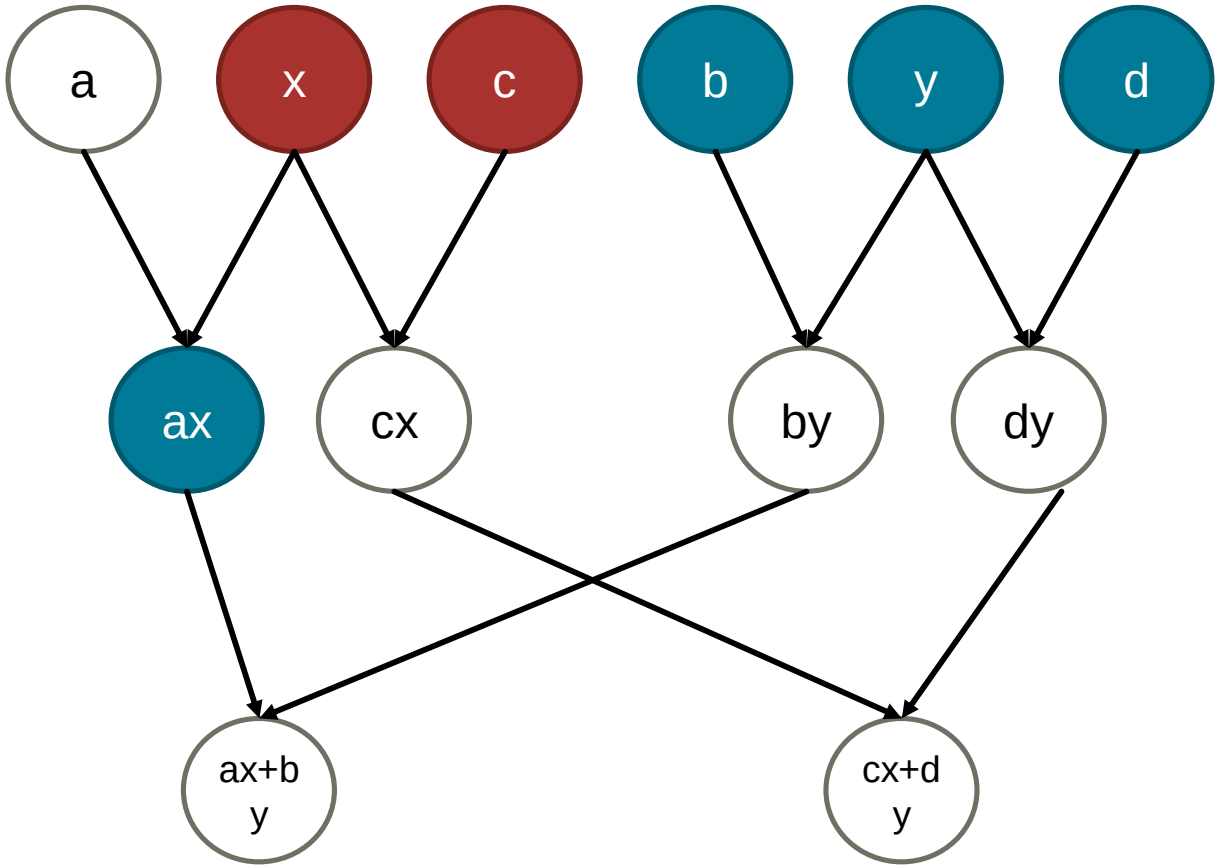
S = 3

I/O transfers: 3



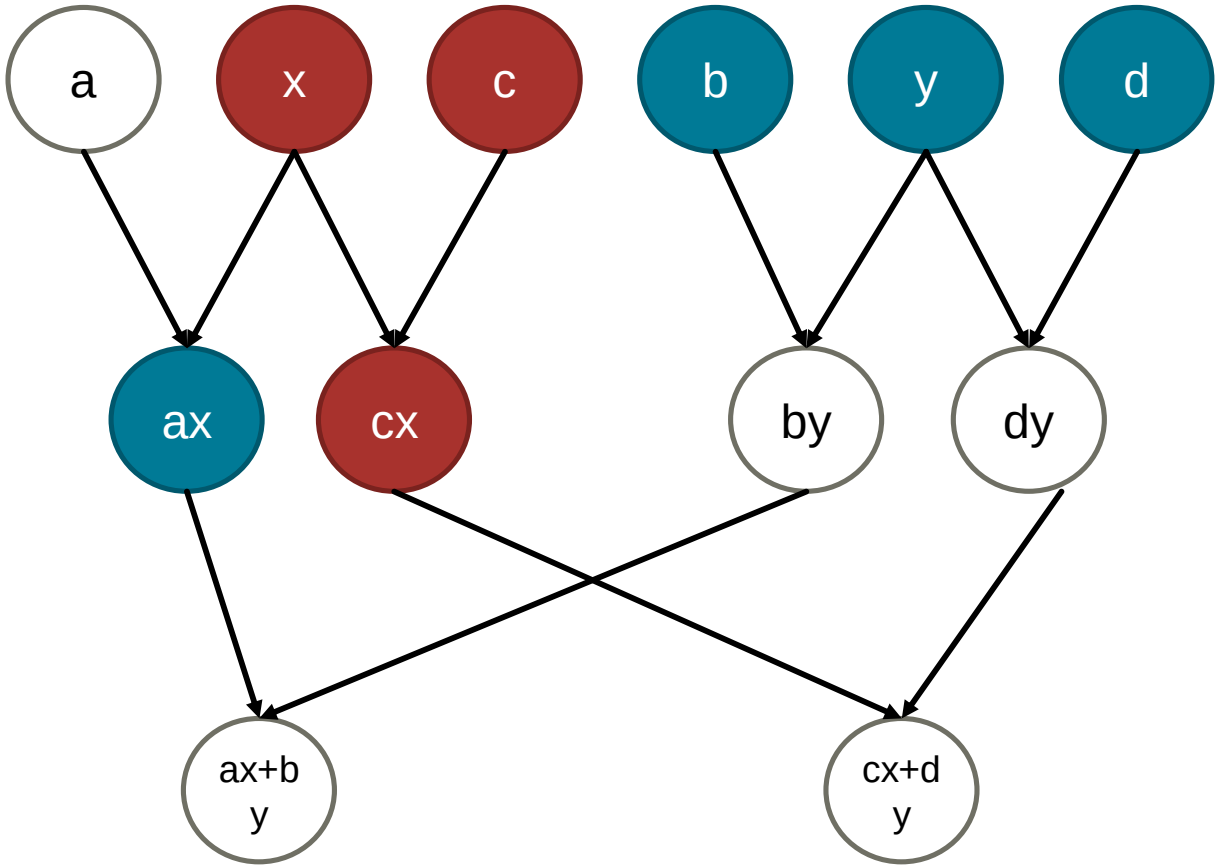
S = 3

I/O transfers: 4



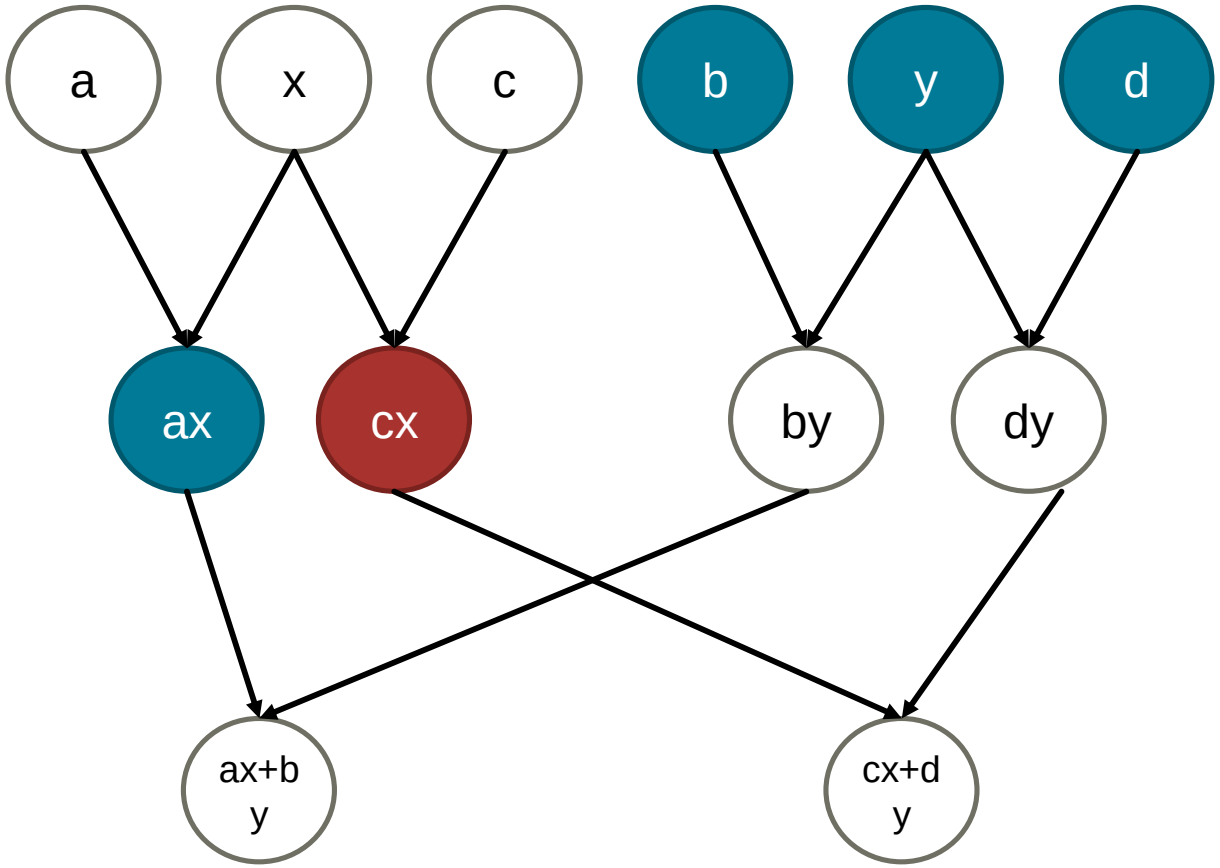
S = 3

I/O transfers: 4



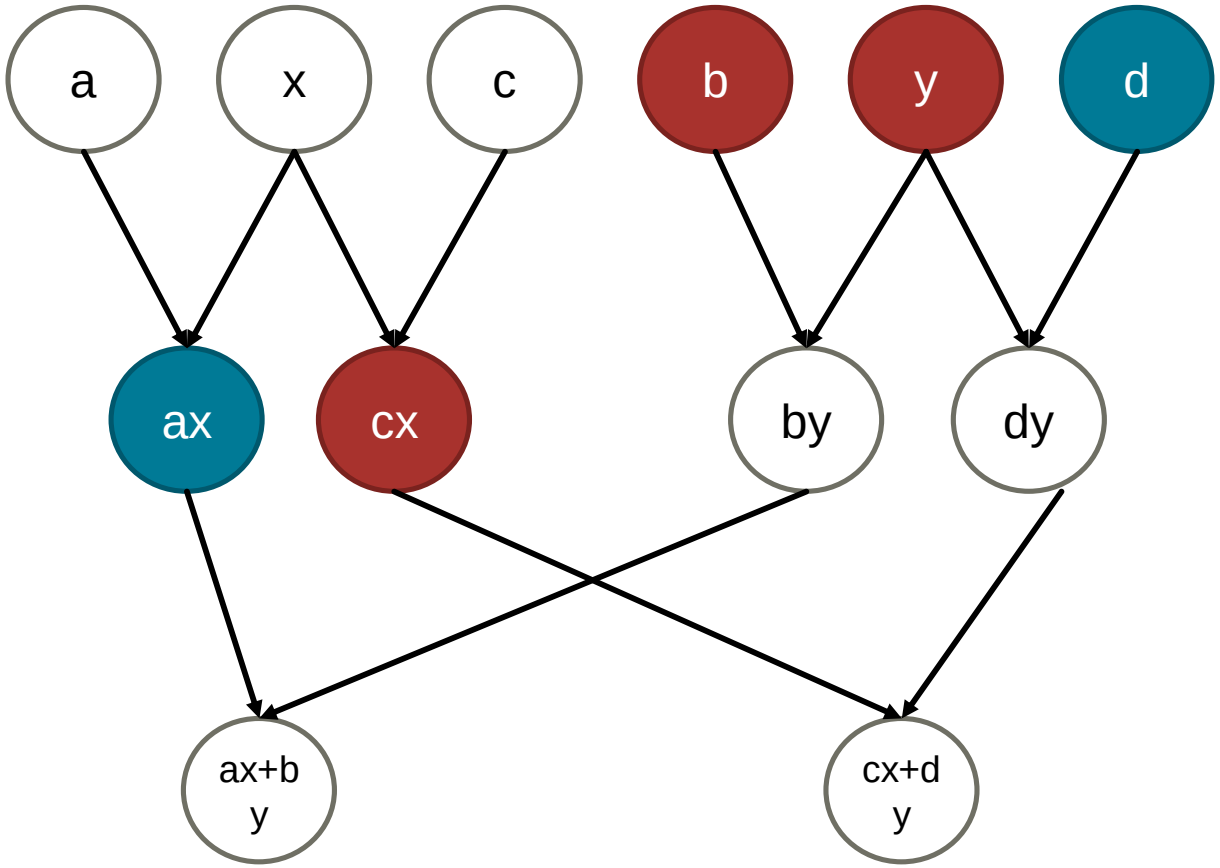
S = 3

I/O transfers: 4



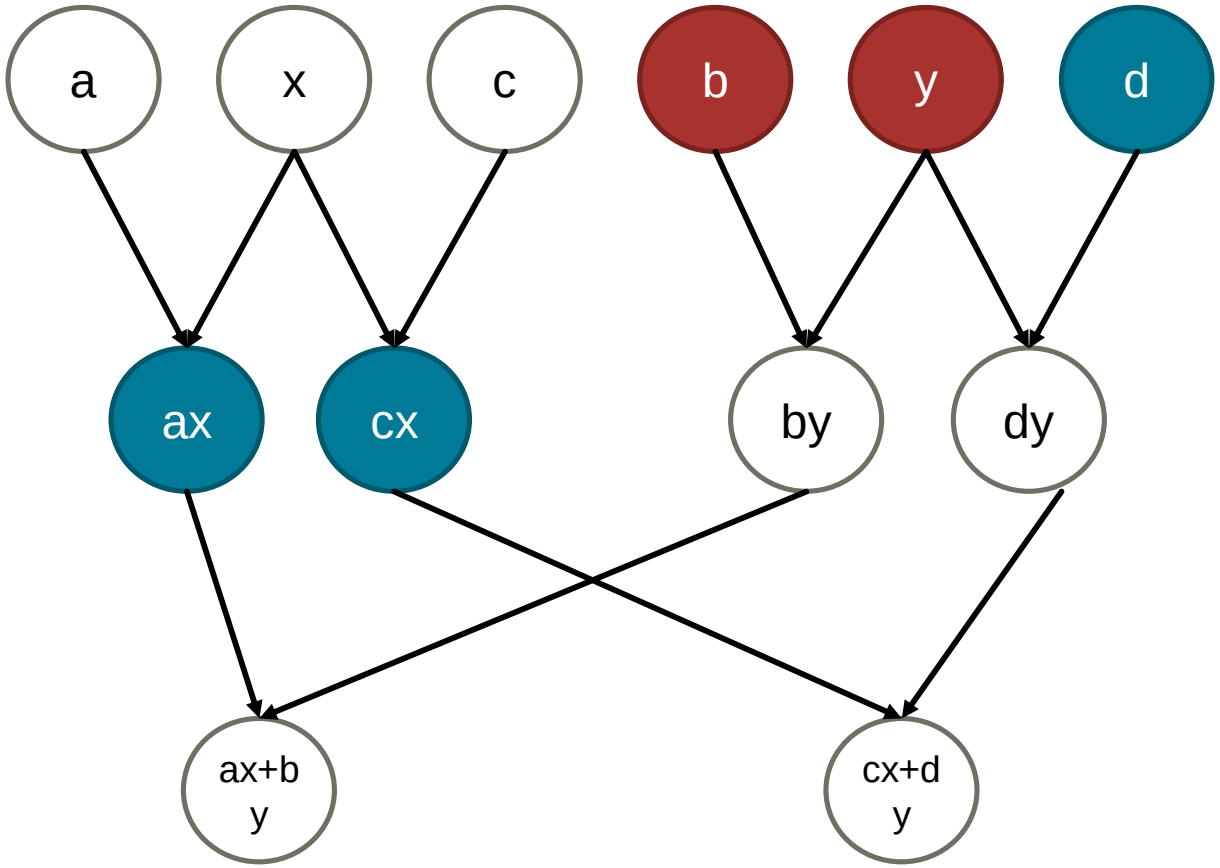
S = 3

I/O transfers: 6



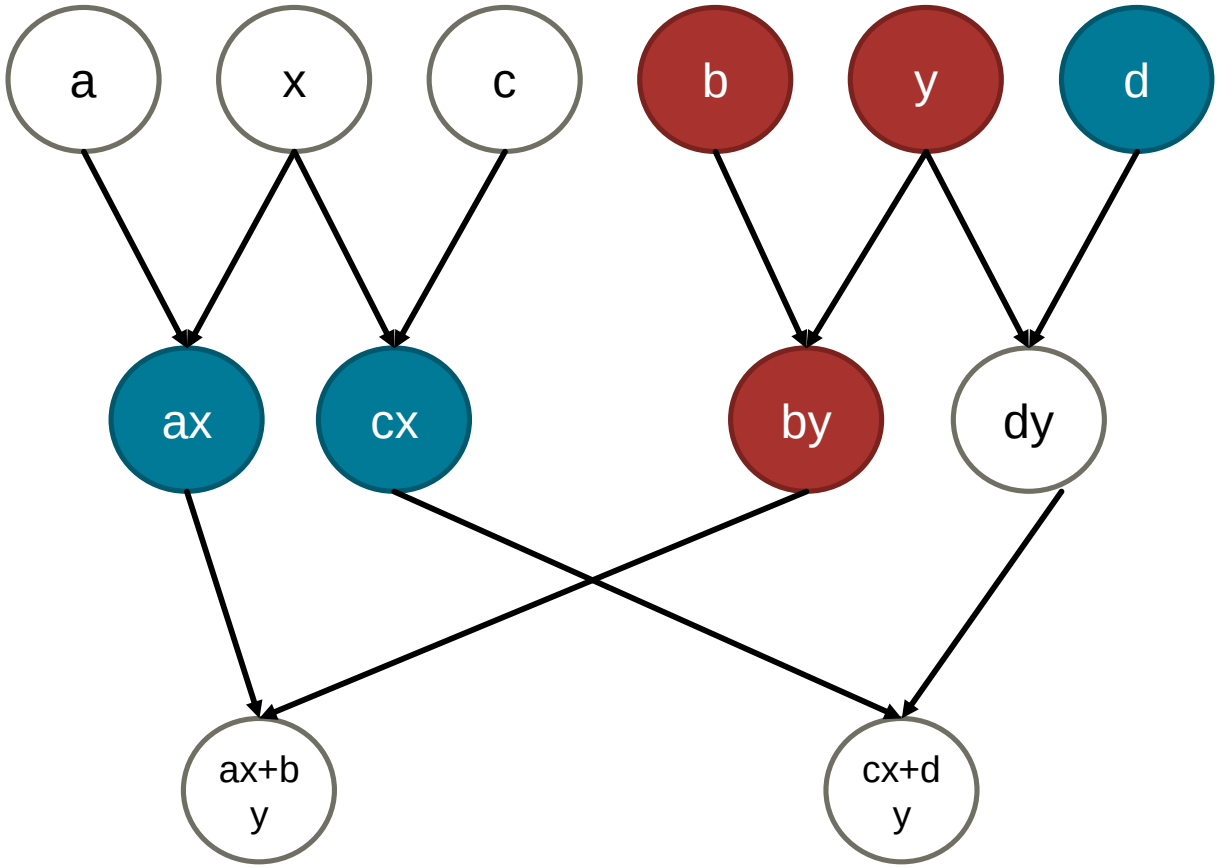
S = 3

I/O transfers: 7



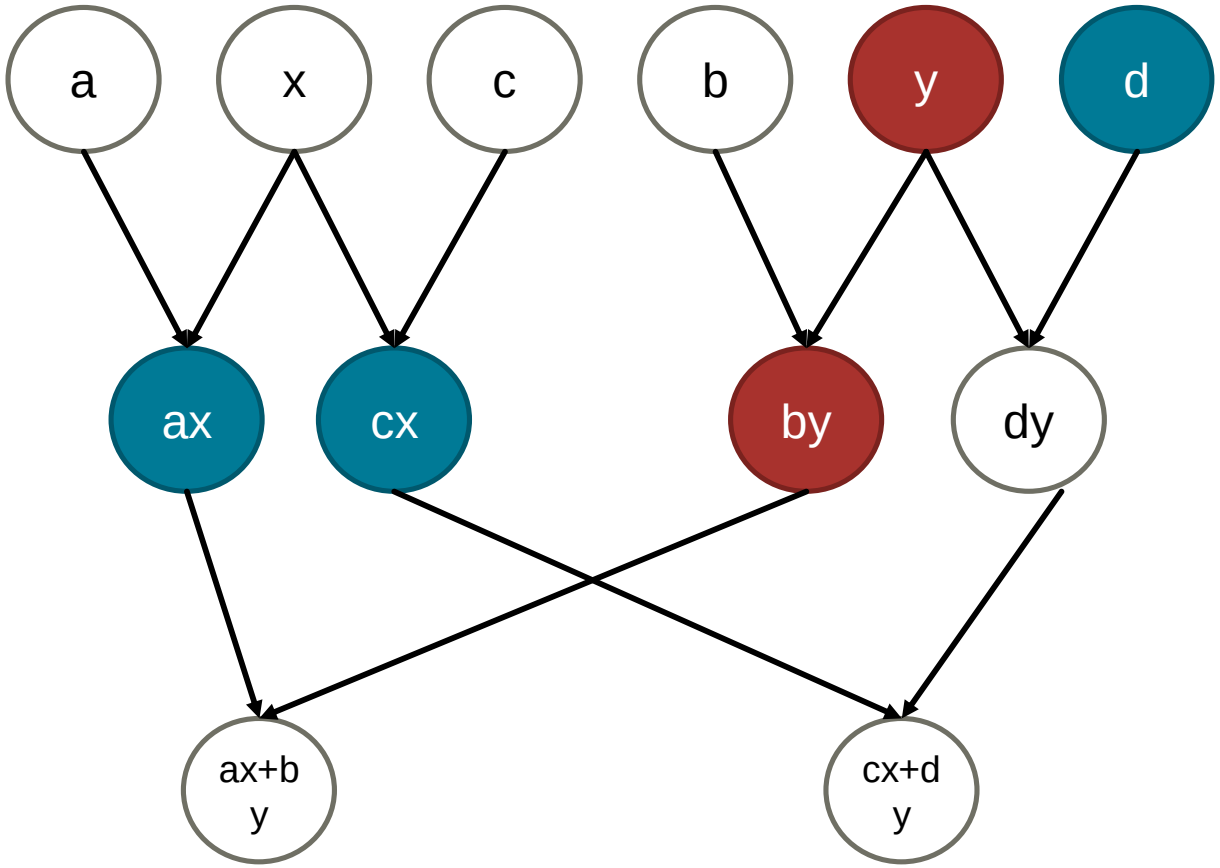
S = 3

I/O transfers: 7



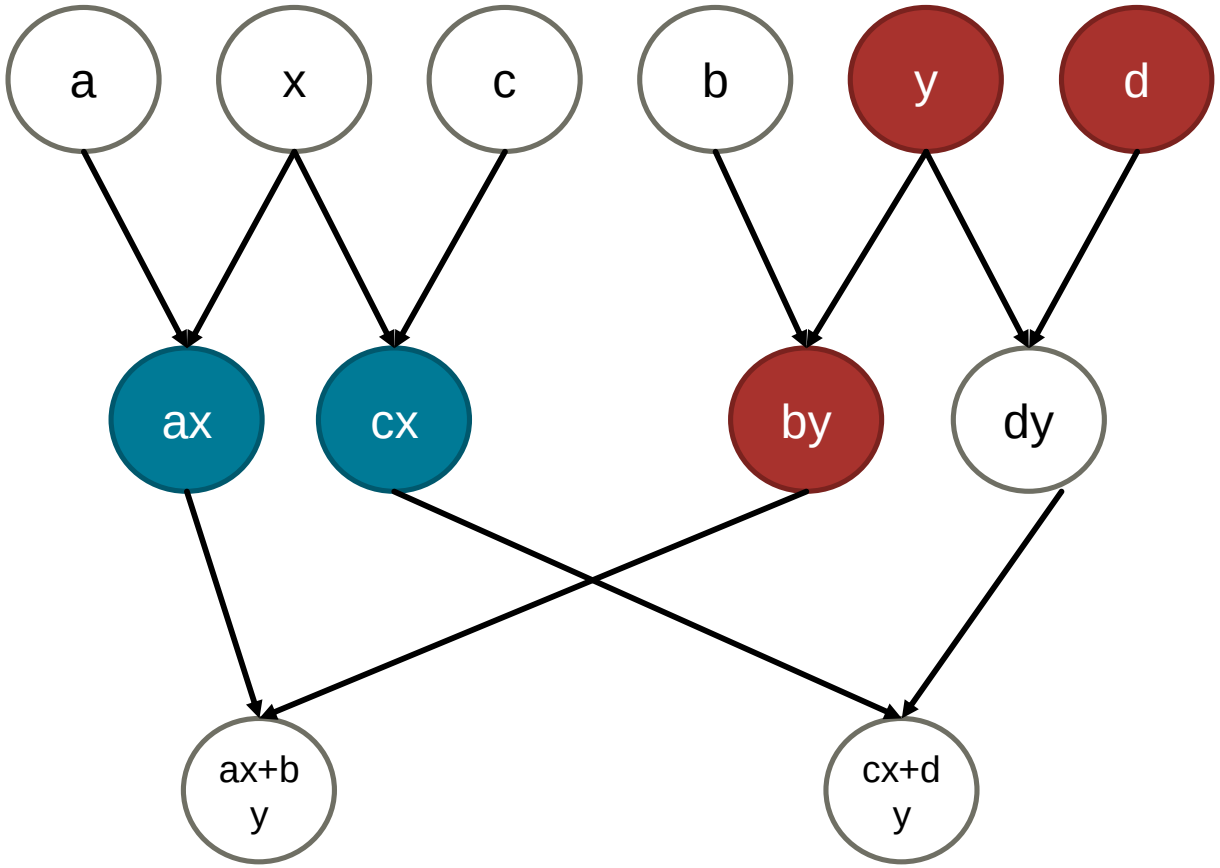
S = 3

I/O transfers: 7



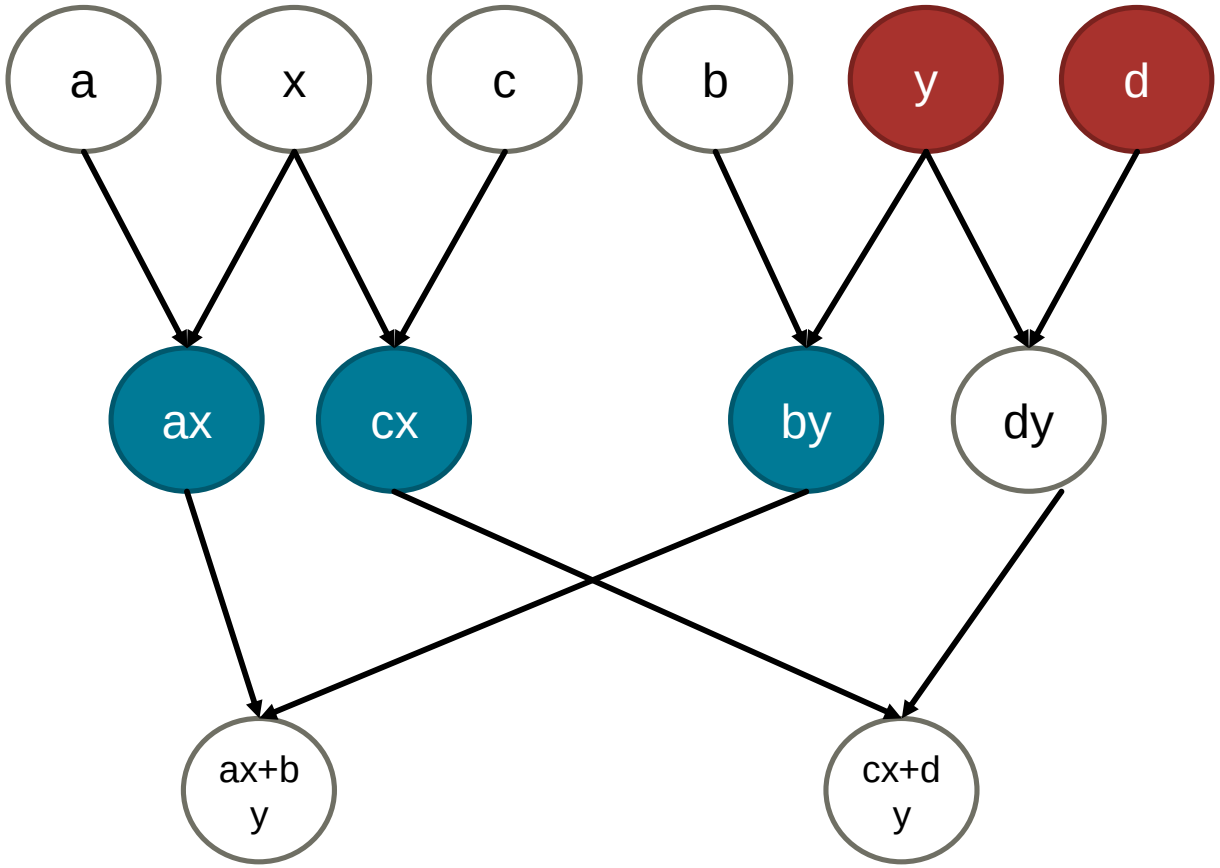
S = 3

I/O transfers: 8



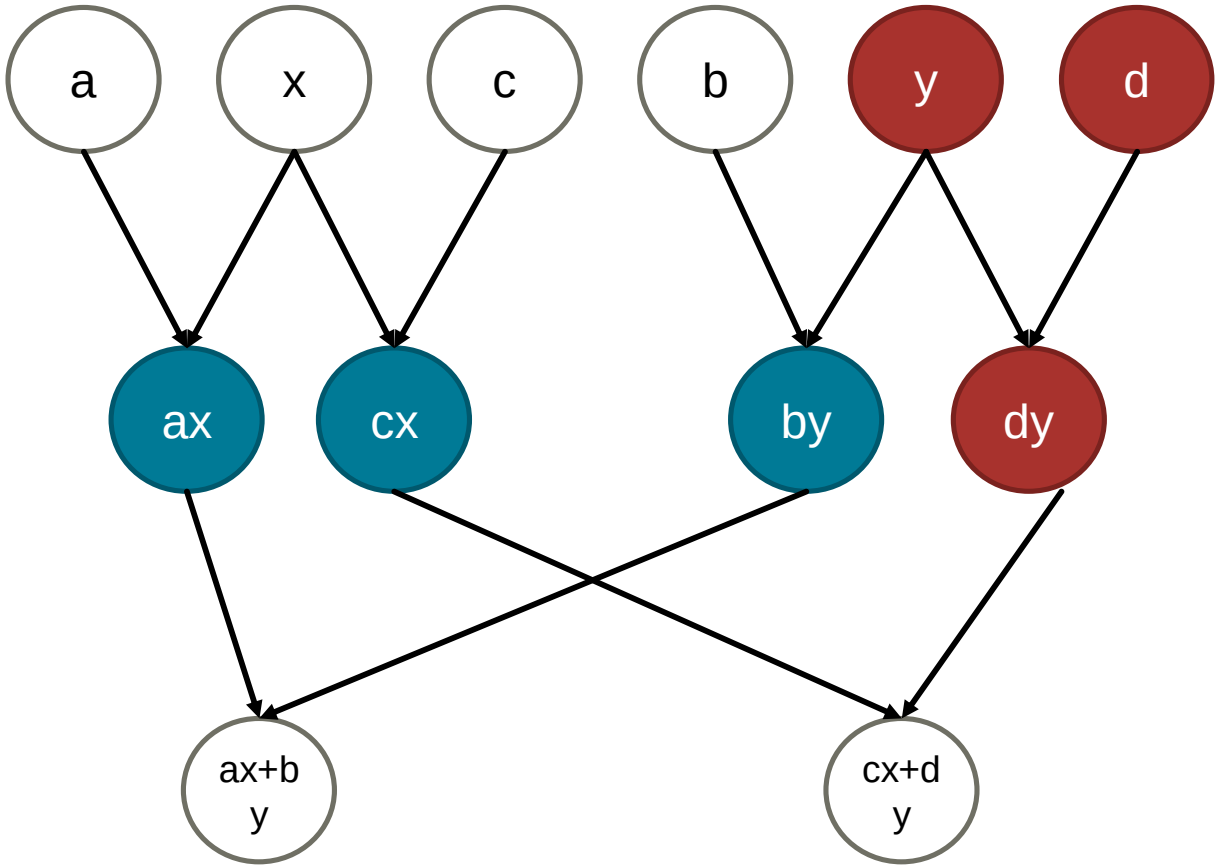
S = 3

I/O transfers: 9



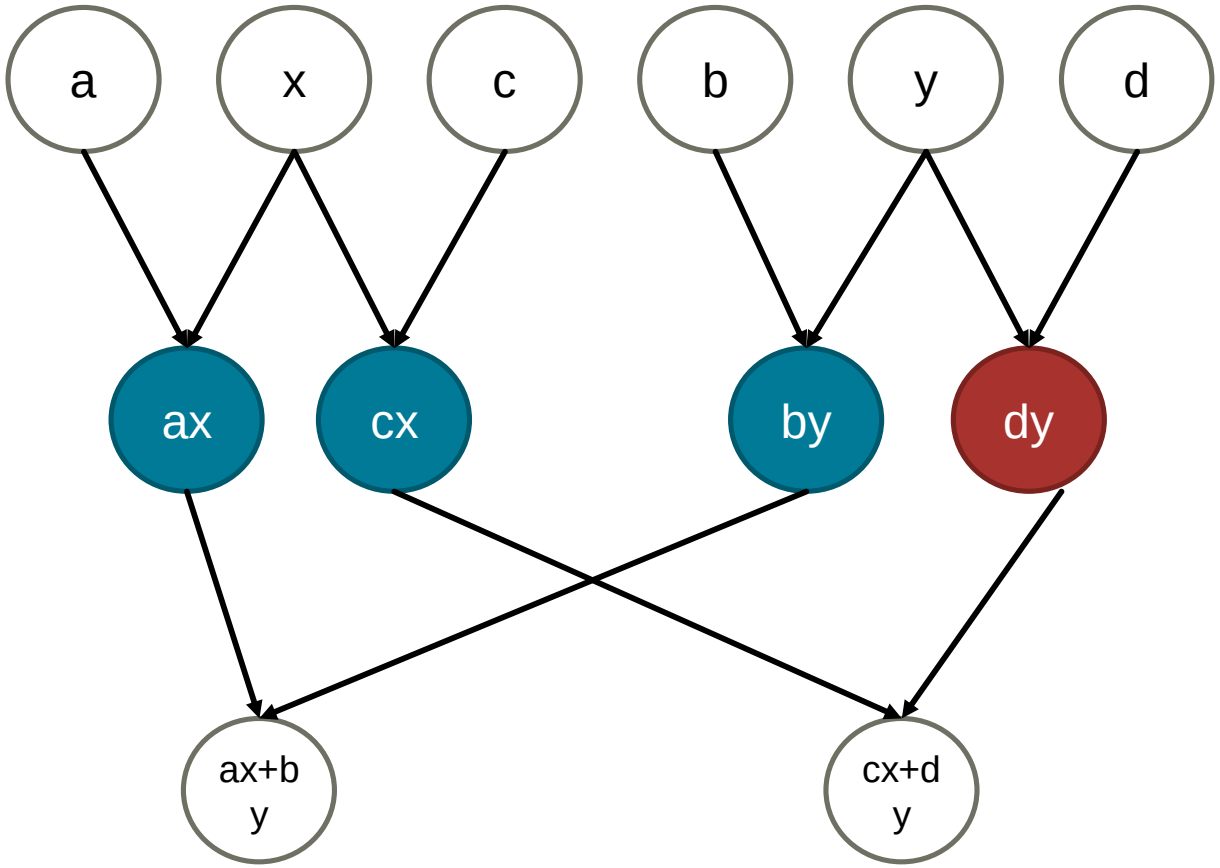
S = 3

I/O transfers: 9



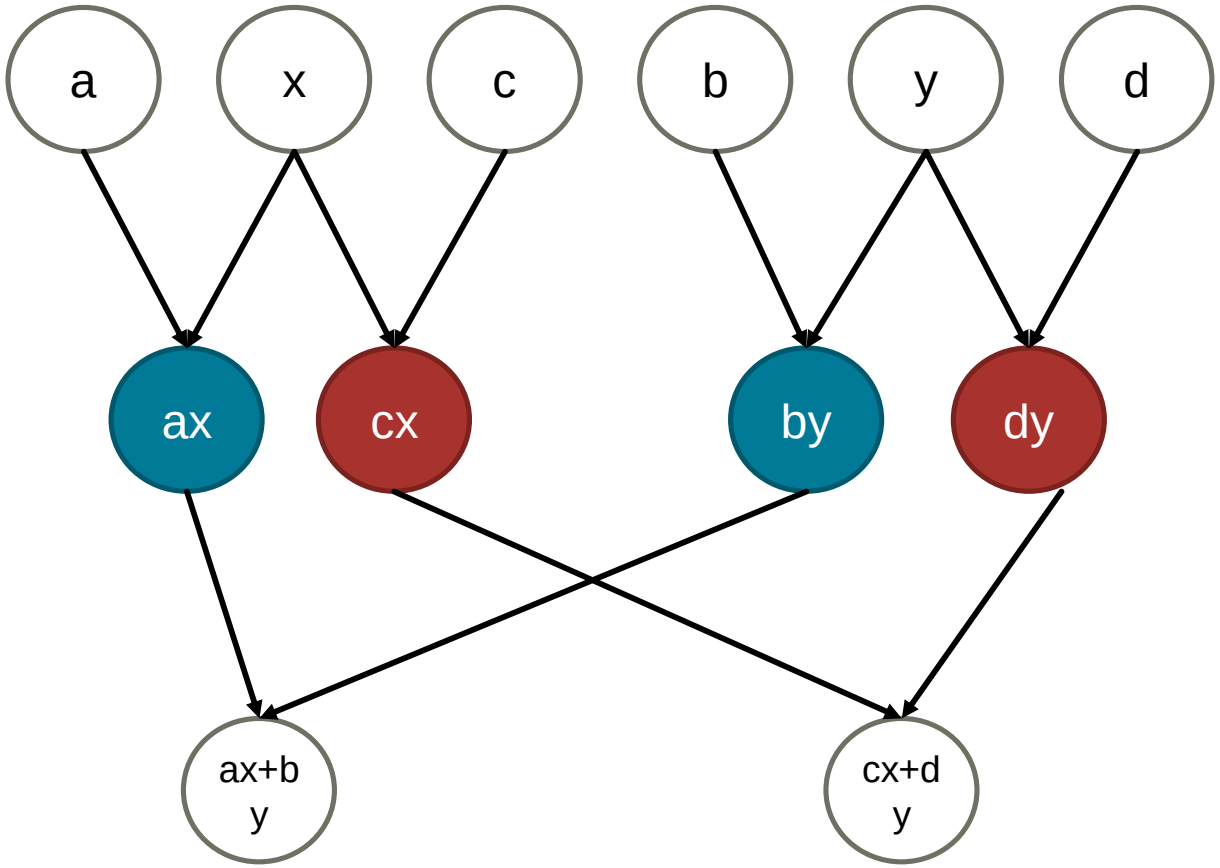
S = 3

I/O transfers: 9



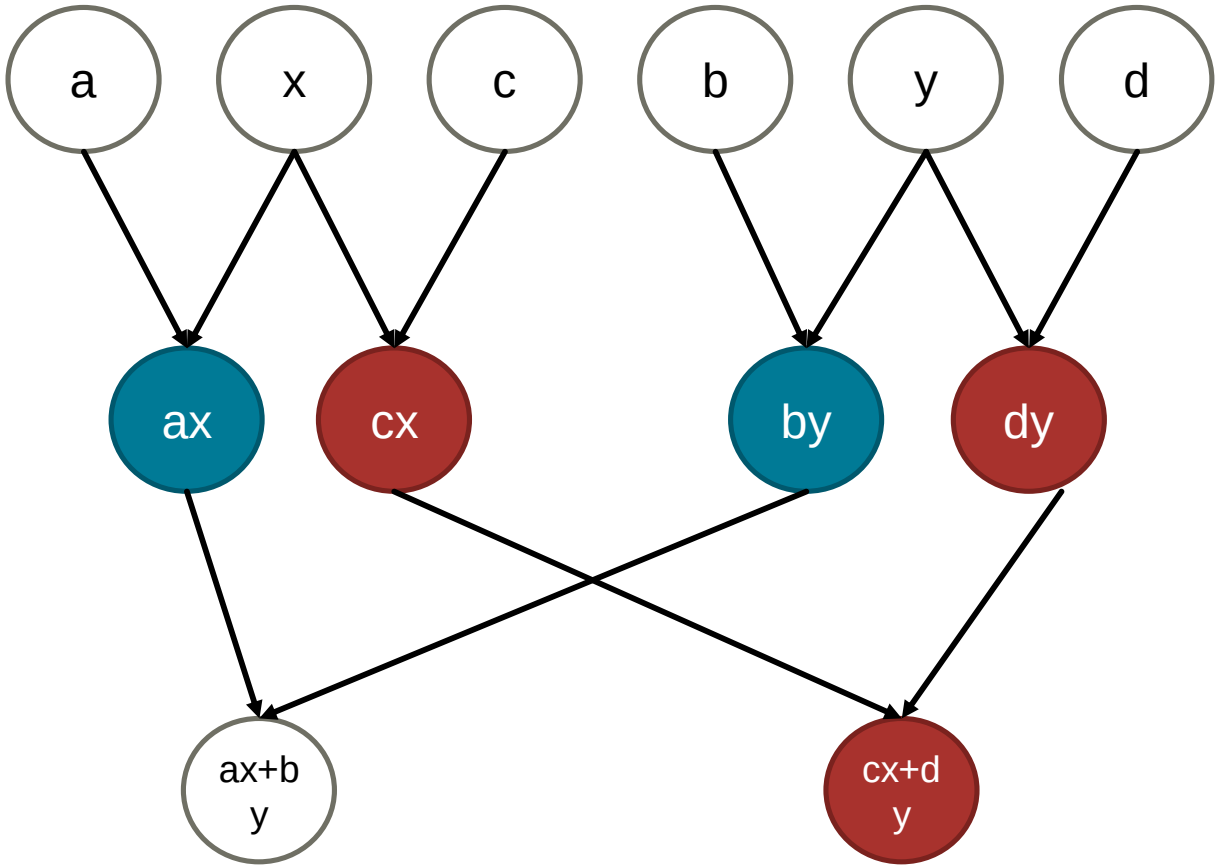
S = 3

I/O transfers: 10



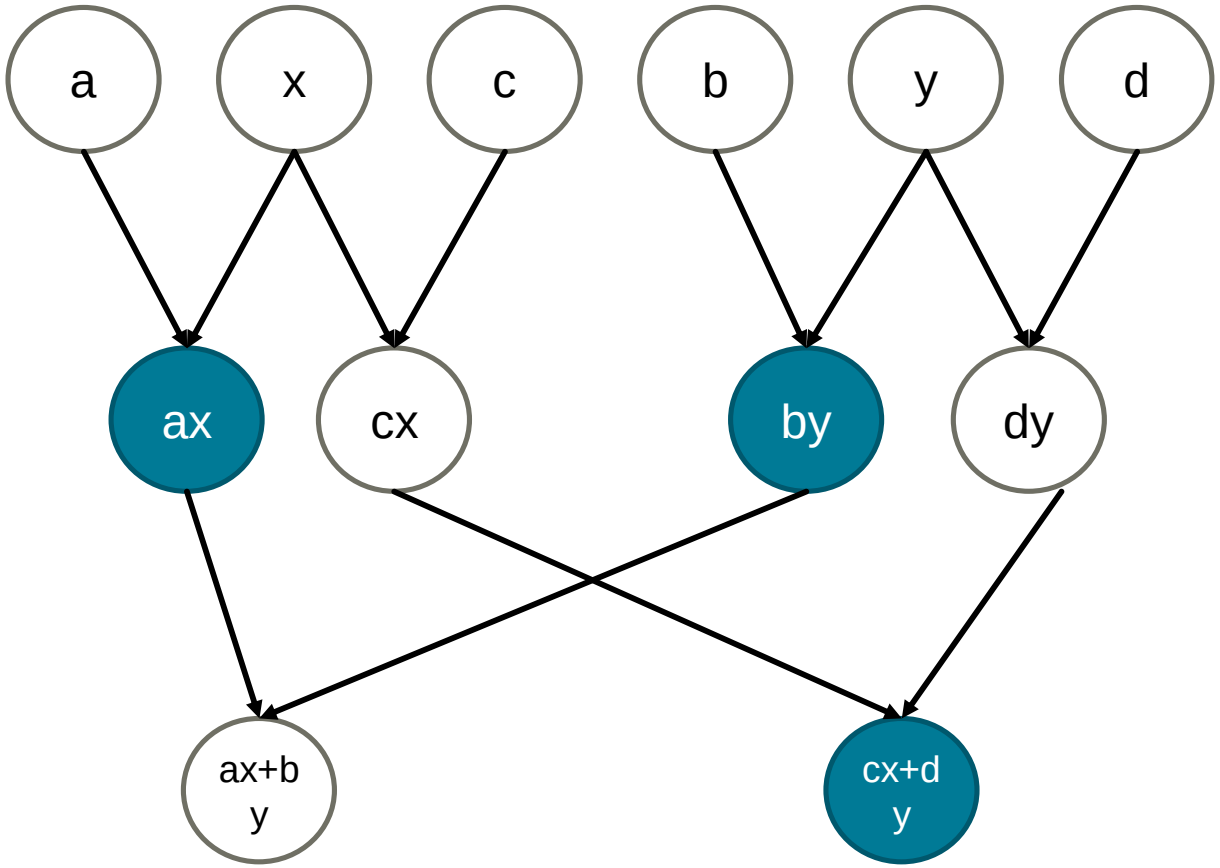
S = 3

I/O transfers: 10



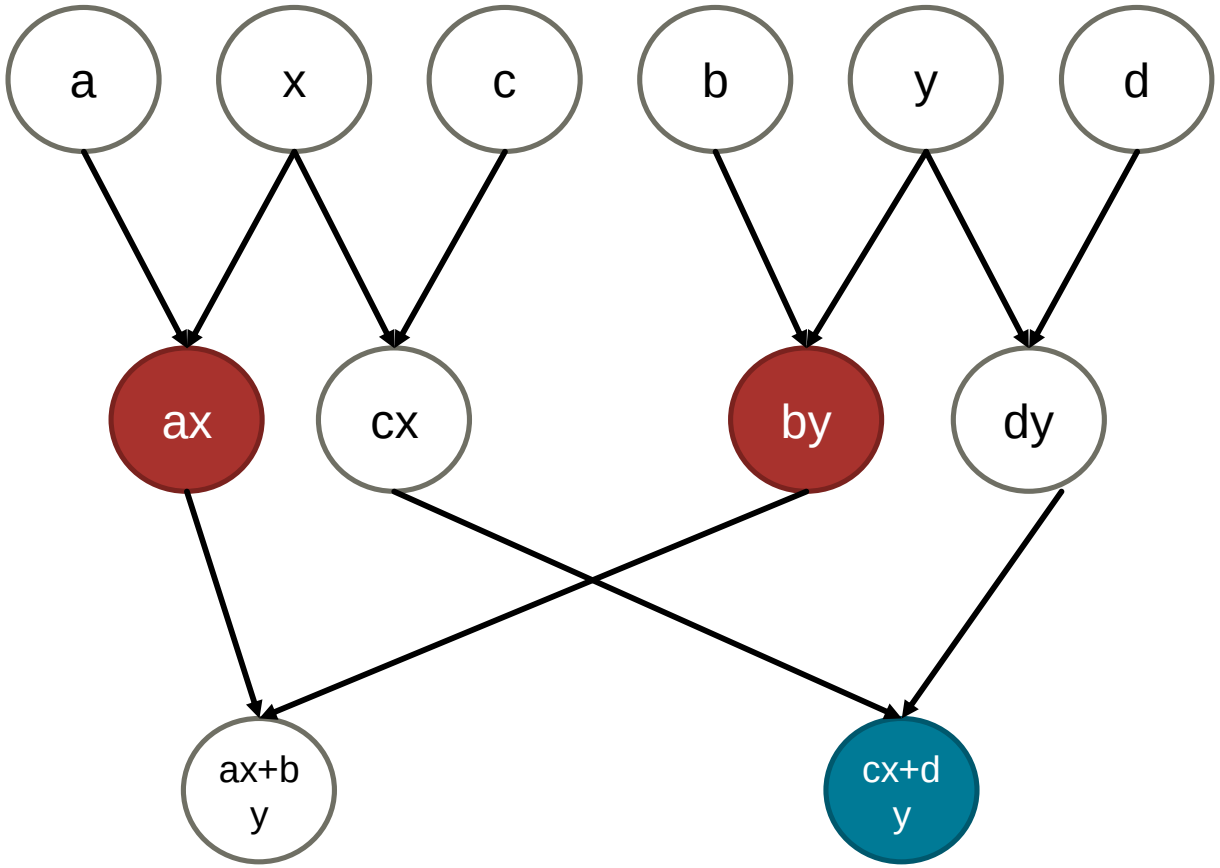
S = 3

I/O transfers: 11



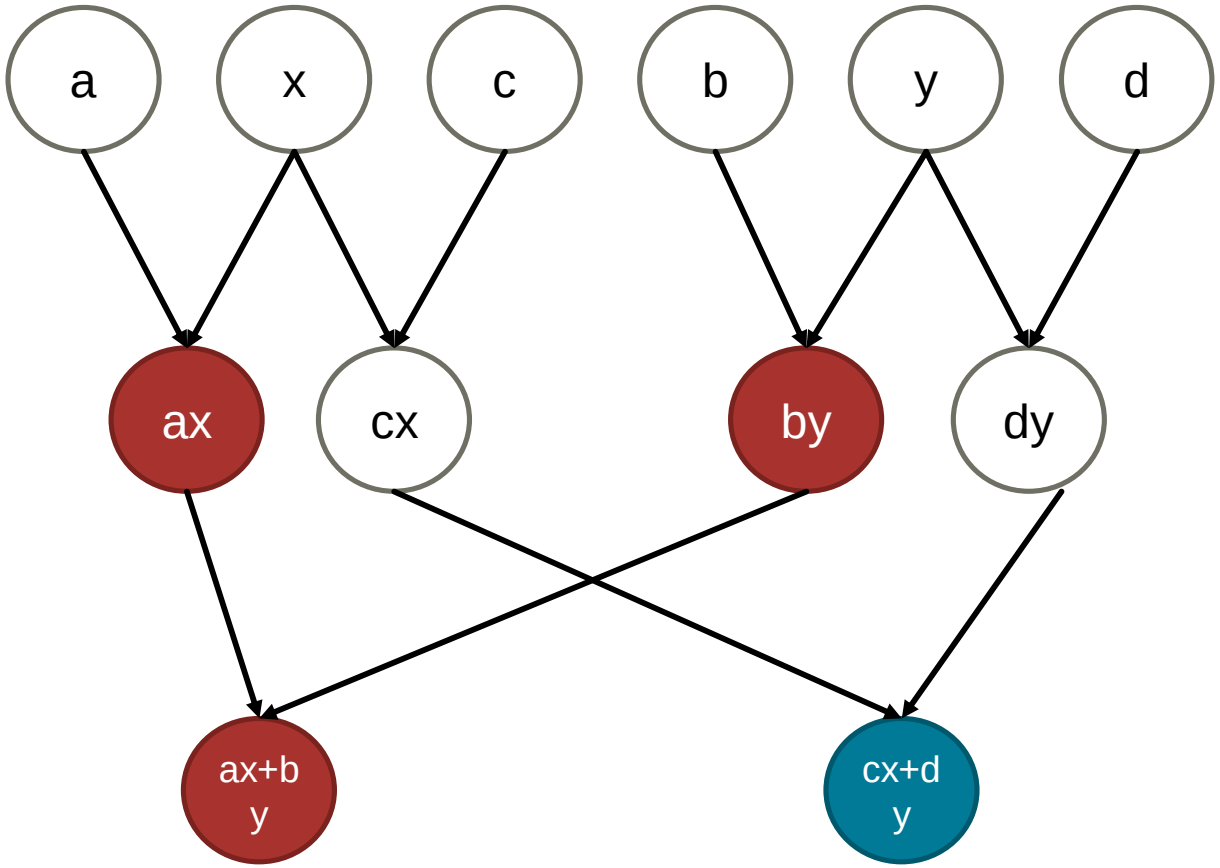
S = 3

I/O transfers: 13



S = 3

I/O transfers: 13



S = 3

I/O transfers: 14