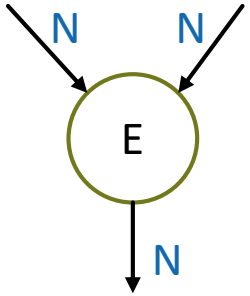
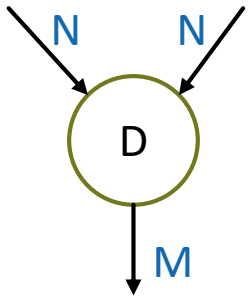
ASA: Scheduling

Type of tasks

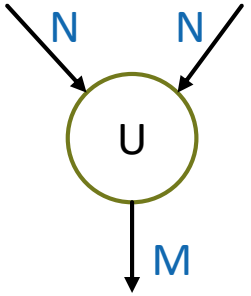
We begin by restricting ourselves to certain types of nodes



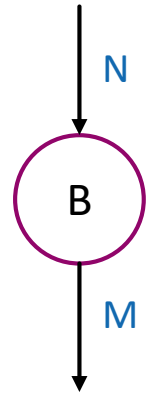
Element-Wise
(e.g., vect add)



Down-sampler
(M <N, e.g. dot product)



Up-sampler
(M>N, e.g., out.product)



Buffer node
(acts as a barrier)

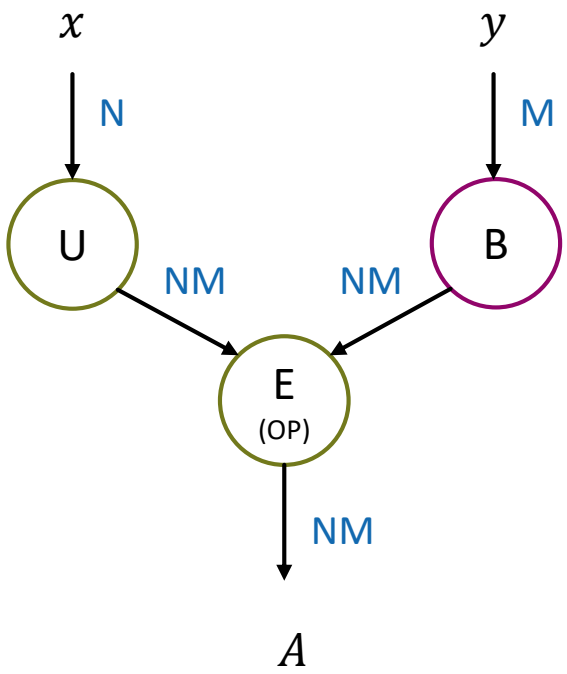
Up-samplers may trigger backpressure effect

What about more generic computations?

Outer product

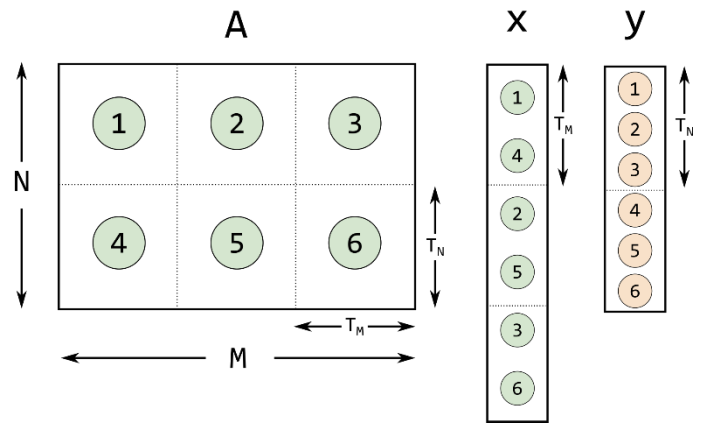
We want to compute $A = xy^T$, where *x* is an N-element vector and *y* is an M-element vector.

Let's assume that the outer product is computed as follows: every single element of *x* is multiplied by the entire *y* to compute a single row of *A*

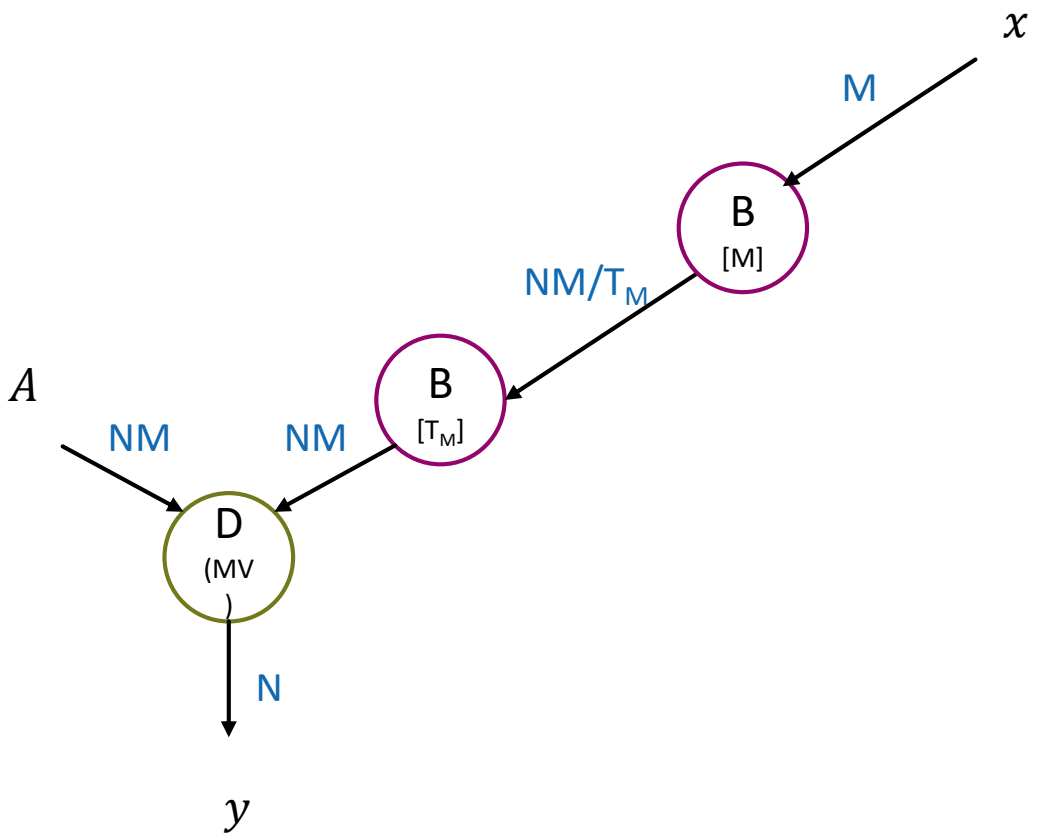


Tiled operations

Let's consider the case of tiled Matrix-Vector multiplication:



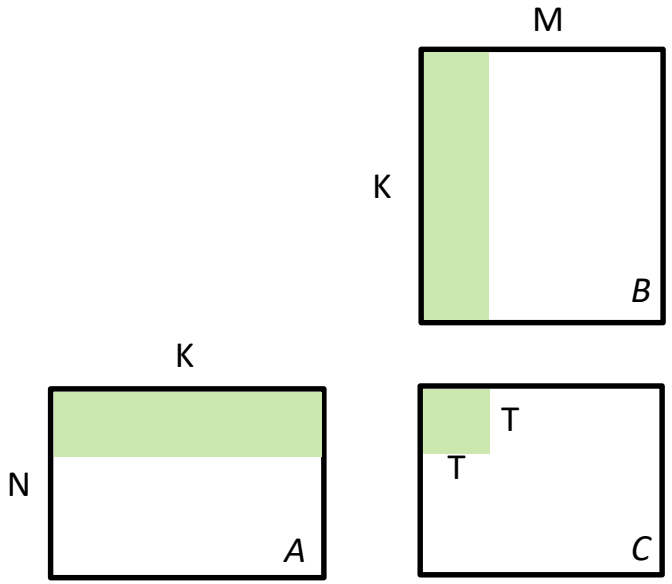
In this case we can use different buffer nodes to represent different buffering opportunities

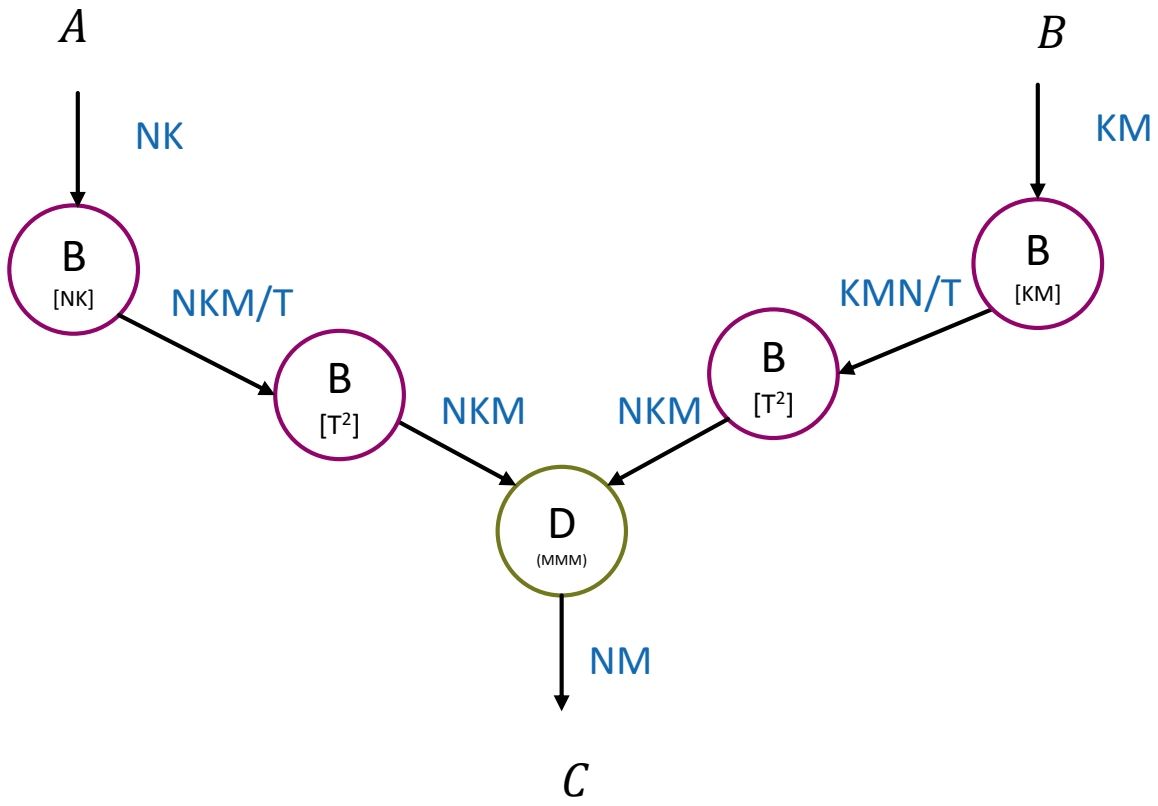


- The rightmost barrier node is used to indicate the entire vector x (that must be read N/T_N times)
- The second node models the buffering of a single tile of *x* (whose elements are replicated T_M times)

Tiled operations

What about a tiled MMM? Let's assume squared tiles of size TxT

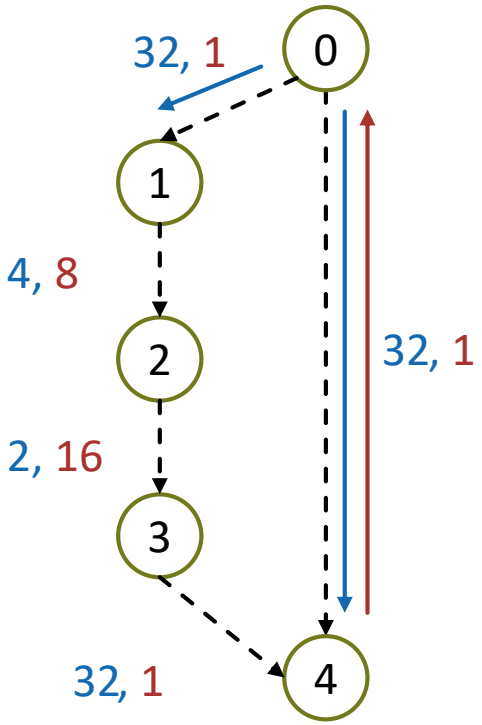




Buffer space

Despite being a DAG, we can still experience deadlocks when we stream.

Let's consider the case of a disjoint path between *u* and *v*



We can exploit the information about the *streaming intervals:*

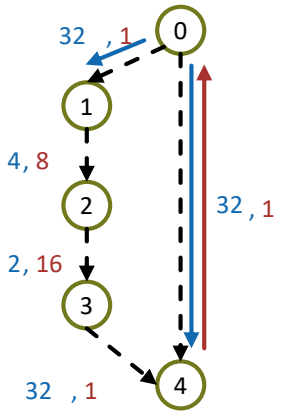
- they tell us how frequently an element flows on a given edge
- For each path (in the example), the maximum str interval will tell us how much data is needed at the input to produce some data at the output
- We look at all paths, take the ratio of the maximum and this will tell us the minimum buffer space needed to avoid deadlocks

In this example it is 16.

But we may be also interested in looking for the *optimal* buffer space, that is the buffer space that will give us max performance (i.e., no *bubbles*).

In this example it is 18

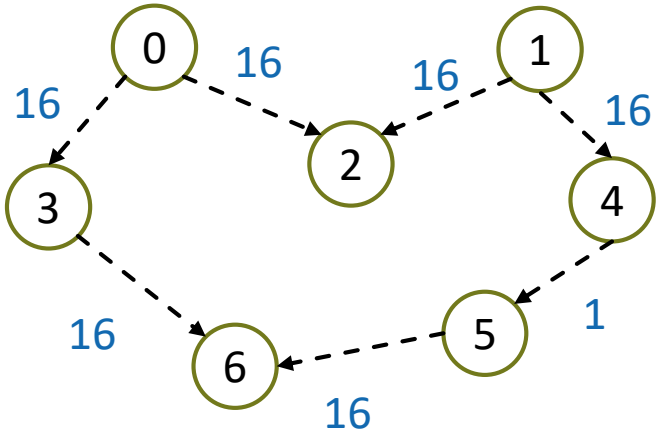
Buffer space



Idea (WIP): look at the delay that it takes from the source to the destination

- Again, we exploit streaming intervals
- But we need also to take into account when each node produce the first output element
- Where do we place the buffer space?

We need to analyze all the undirected cycle



Since the graph is partitioned in Streaming blocks, we have to look at the optimal buffer space only within a Streaming Block. Amongst SBs there is no chance of deadlock.

Finding the optimal buffer can be part of the heuristic to partition the DAG in streaming blocks (WIP)

Consolidate theory (WIP)

Working towards a theory for streaming applications:

- Understand what happens with infinite Pes
- Then use this info to schedule over P Pes (e.g., understand how to partition in Streaming Blocks)

Streaming intervals: how to compute them?

Given a node with a production rate R(v), we define its streaming interval accordingly

$$S^{+}(v) = S^{-}(\text{v})/\text{R(v)}$$

Let $u \downarrow$ be the descendants of a node. If the DAG has a single source, then at steady state its streaming interval:

$$S^{+}(v) = \frac{\max_{u \in v^{\downarrow}} K^{-}(u)}{K^{+}(v)}$$

Where $K^{-}(u)$ is the number of elements read by a node u, while $\mathrm{K}^{+}(u)$ is the number of element being produced

Consolidate theory (WIP)

Now we can compute the streaming intervals for all the edges: let's say that there is a path from source v_i to node v_j (that does not contain any buffer node). Then we can find its streaming interval as :

$$S^-(v_j) = \frac{K^+(v_i)}{K^-(v_j)} S^+(v_i)$$

We have a simple and direct way of computing the streaming interval for a node (TBD: needs to be extended to multiple sources case). Proof omitted for brevity

Consolidate theory (WIP)

Can we derive bounds for scheduling? This would help in deriving good heuristics and assess their quality

Let's discuss the case of a DAG composed by only element-wise nodes.

Let T_1 be the execution time on 1 PE, and T_∞^s the execution time with infinite PEs and with all streaming communications (whenever possible)

Consider the following way of scheduling the DAG over p PEs: we order the tasks by their level in the DAG (break ties arbitrarily). Then, in this order, subdivide the tasks into so-called **streaming blocks** of p tasks. Then the streaming blocks are scheduled one after the other (e.g., by using gang-scheduling)

We can prove that the execution time over p processors is T_p

$$T_\infty^s \leq T_p \leq \frac{T_1}{p} + T_\infty^s$$

Q: can we do the same with more generic DAG (e.g., composed by downsampler, upsampler, …)