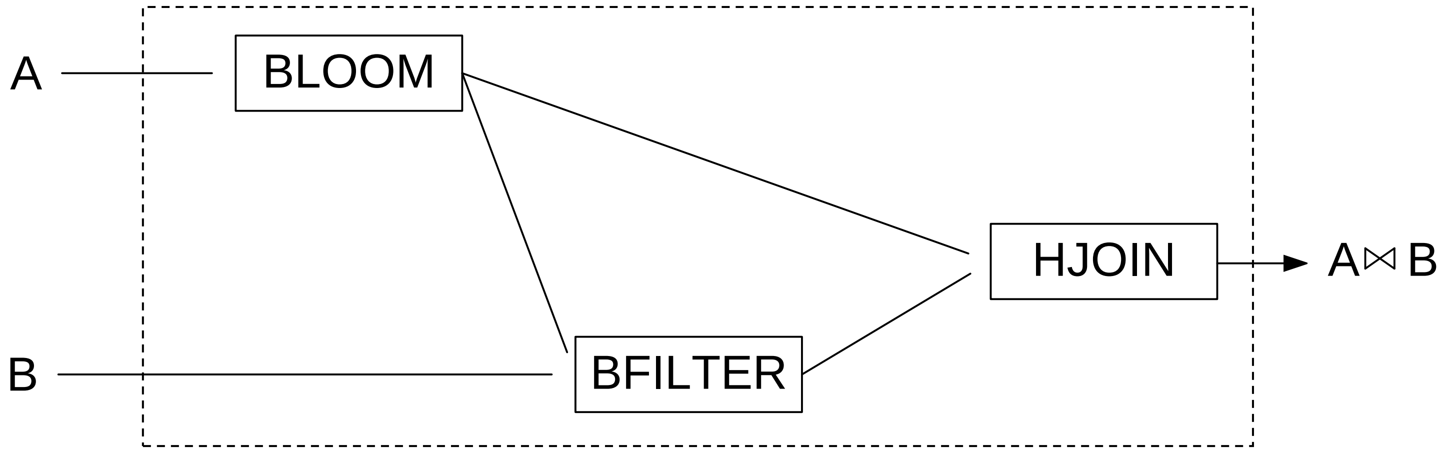
Illustration of join in Gamma. Two Tuples A and B are joined, producing the joined tuples

To be more efficient, the HJOIN can be implemented as follows (next page)



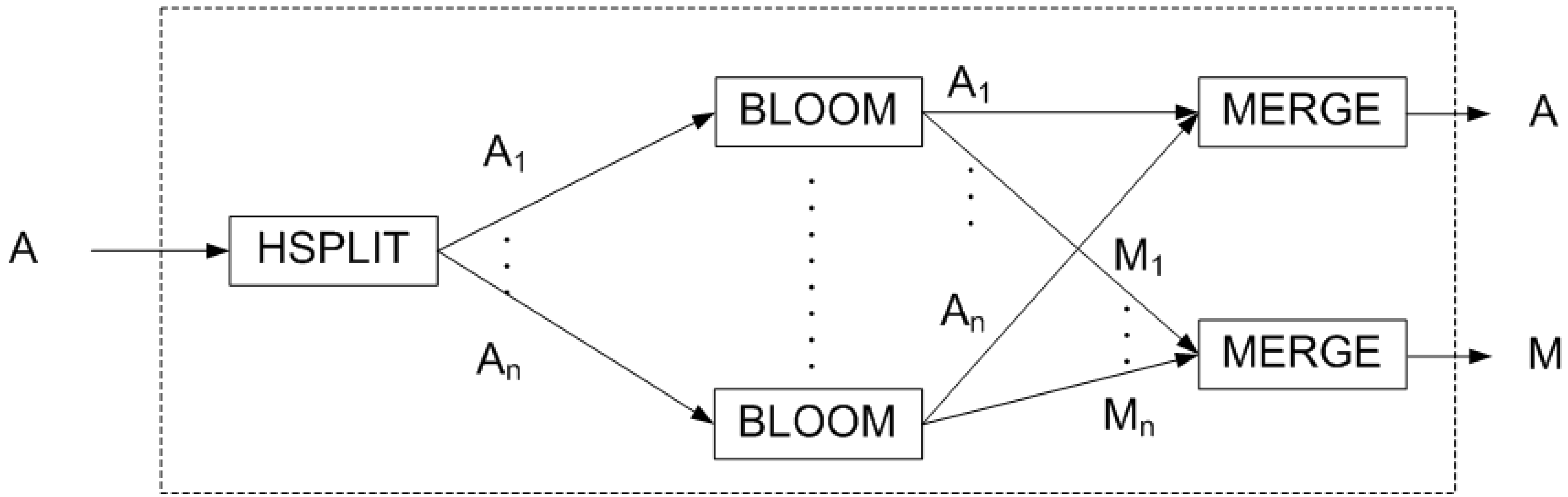
BLOOM:

- clear bit map M
- read each A tuple, hash its join key, and mark corresponding bit in M
- output each tuple A
- after all A tuples read, output M

BFILTER

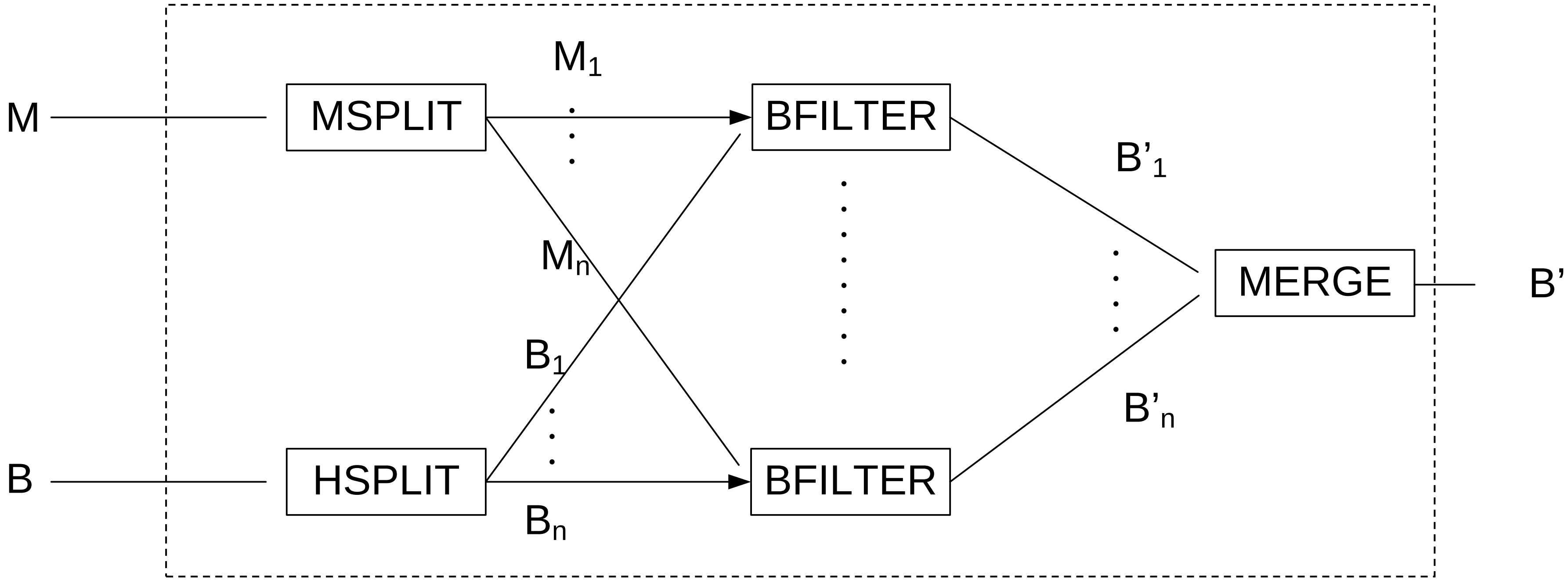
- read bit map M
- read each tuple of B, hash its join key: if corresponding bit in M is not set discard tuple (as it will never join with A tuples)
- else output tuple

Parallelization of BLOOM Box



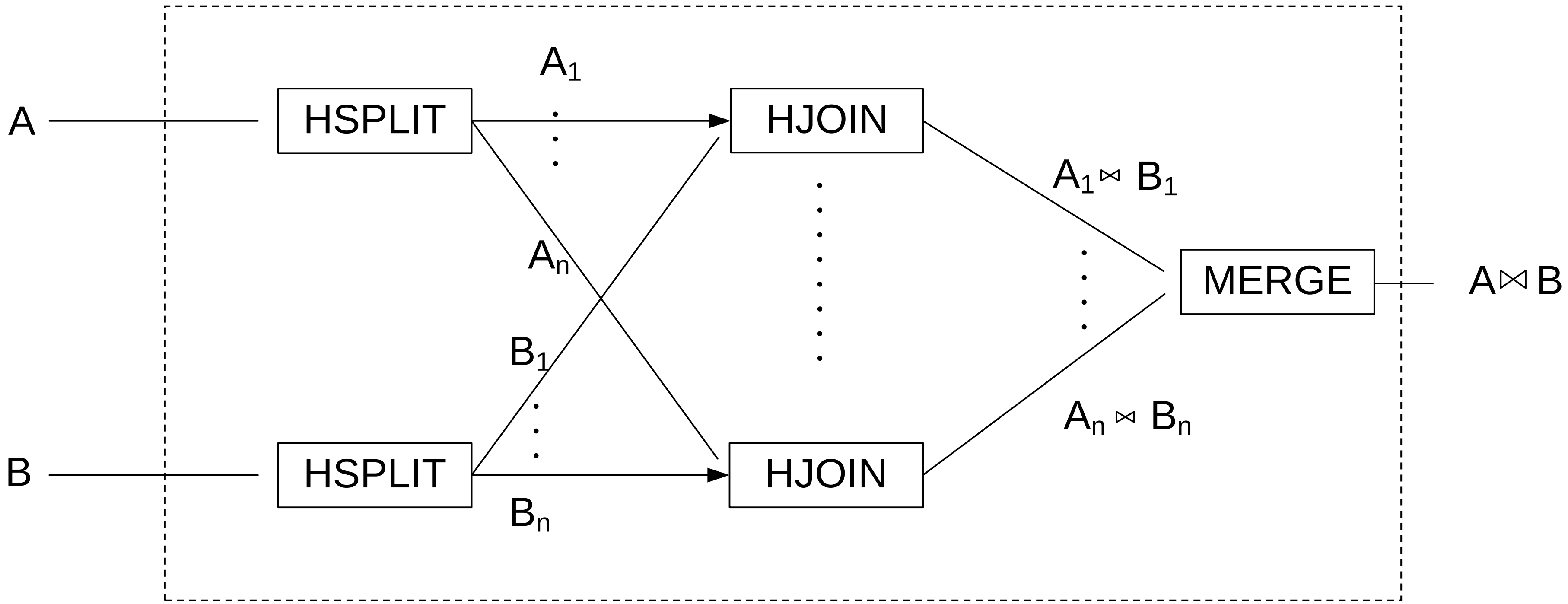
- HSPLIT splits stream A into substreams A_1 to A_n
- passes substreams to BLOOM boxes
- there are as many BLOOM boxes as there are substreams of A
- merge substreams of A and substreams of bit maps to M

Parallelization of BFILTER Box

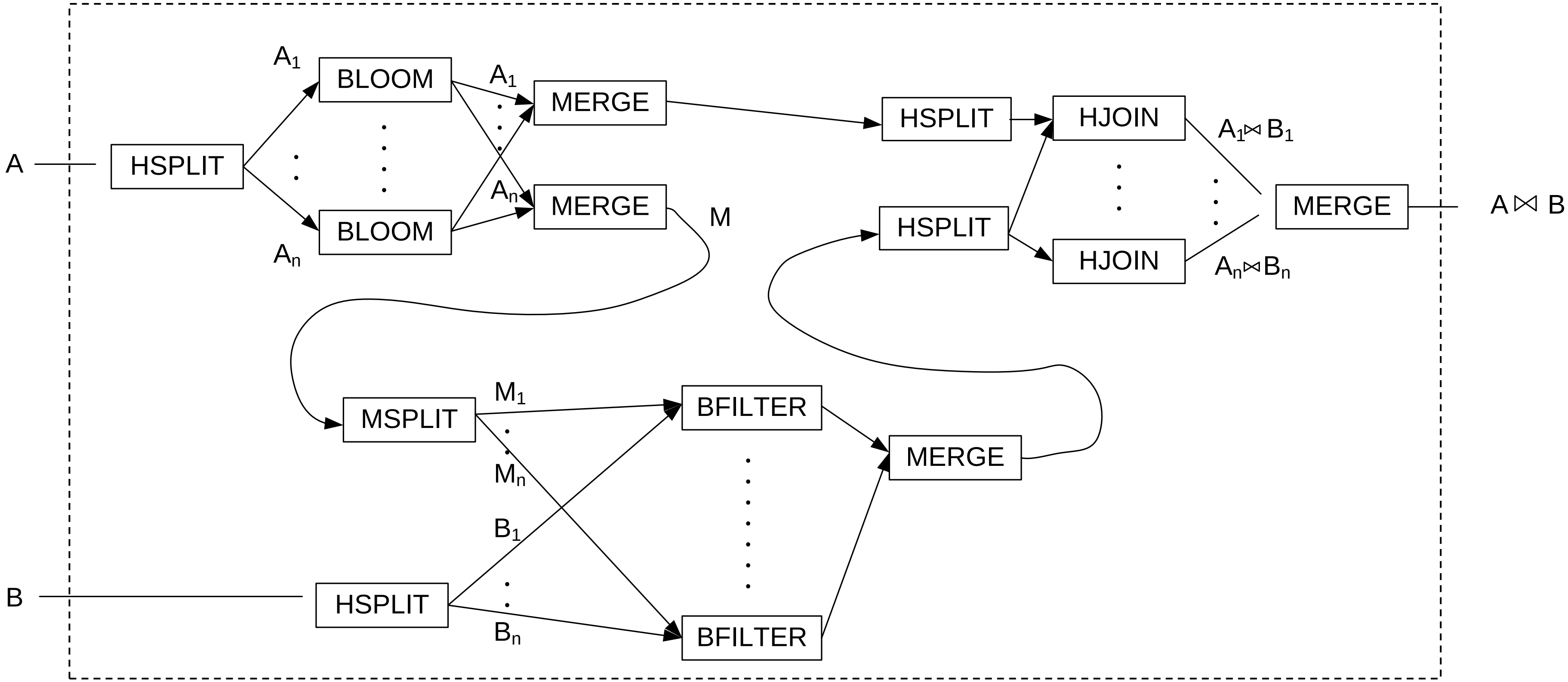


- split M into $M_1 \ldots M_n$
- hash split stream B into $B_1 \ldots B_n$
- filter B_i substreams
- there are as many BFILTER boxes as there are substreams of B and M
- reconstitute stream B'

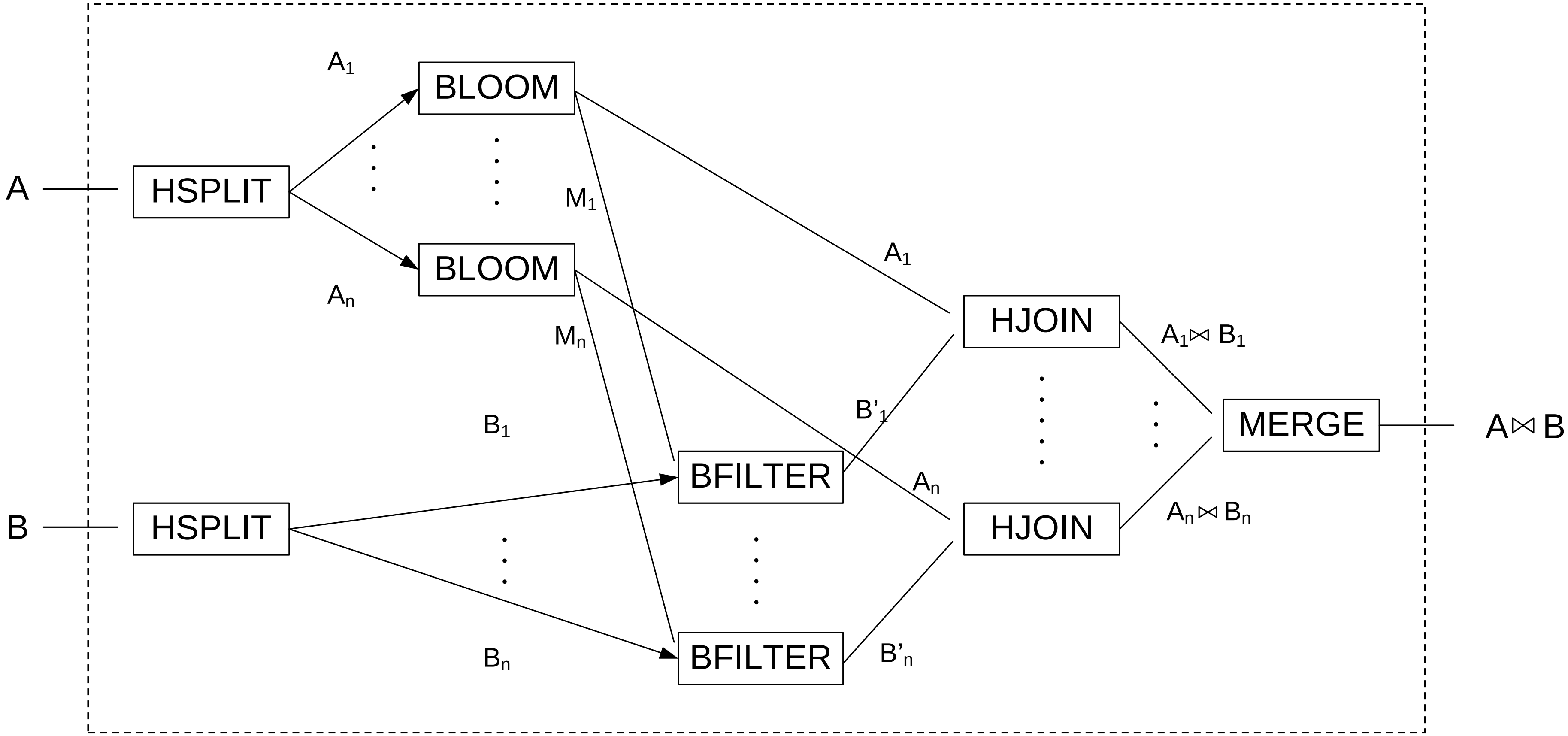
Parallelization of HJOIN Box



- split both streams using same hash function
- join each substream of A and B
- there are as many HJOIN boxes as there are substreams of A and B
- merge result of each HJOIN box



- Substitute parallel implementations for each box
- There are three optimizations possible: Substreams A_1 to A_n are merged and then split, as is bit map M and stream B'
- We can omit these steps and pass substreams A_1 to A_n directly to each HJOIN, pass bit maps M_1 to M_n to each BFILTER, and substreams B'_1 to B'_n to each HJOIN (see next page)



Final architecture of HJOIN in Gamma