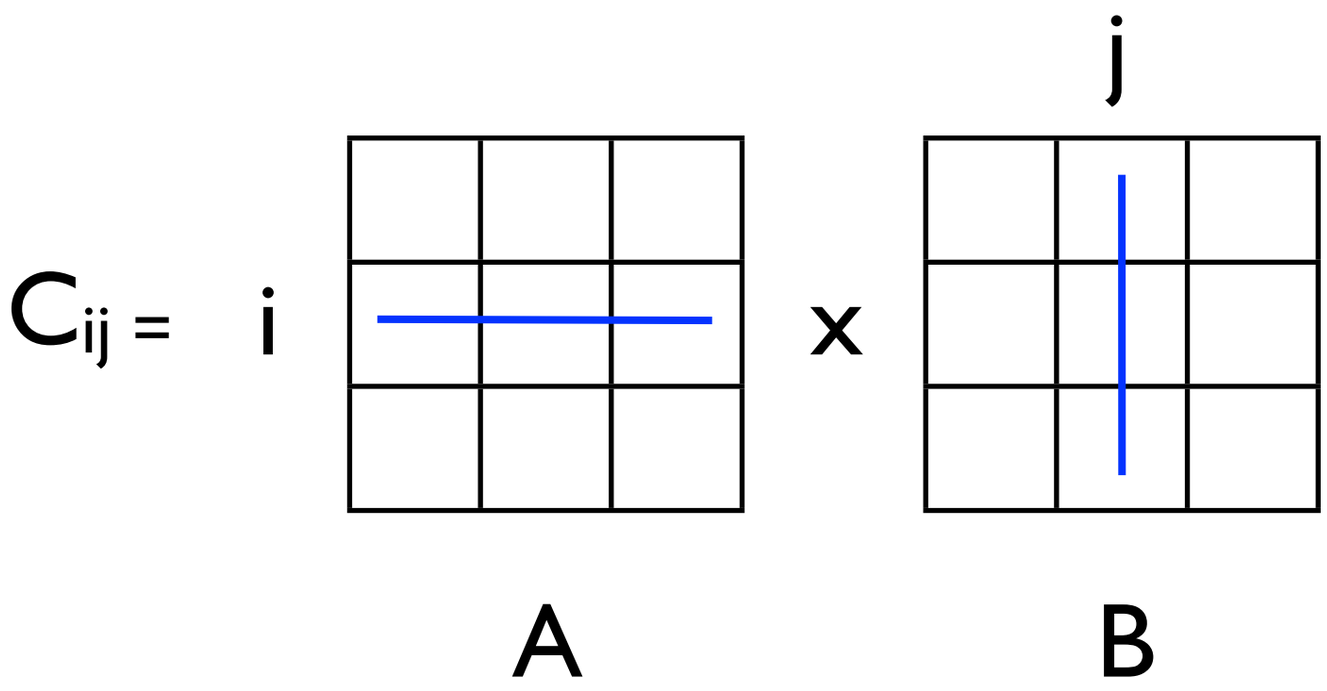
Parallelizing Algorithms in MapReduce

Mauro Sozio
sozio@telecom-paristech.fr

Matrix Multiplication

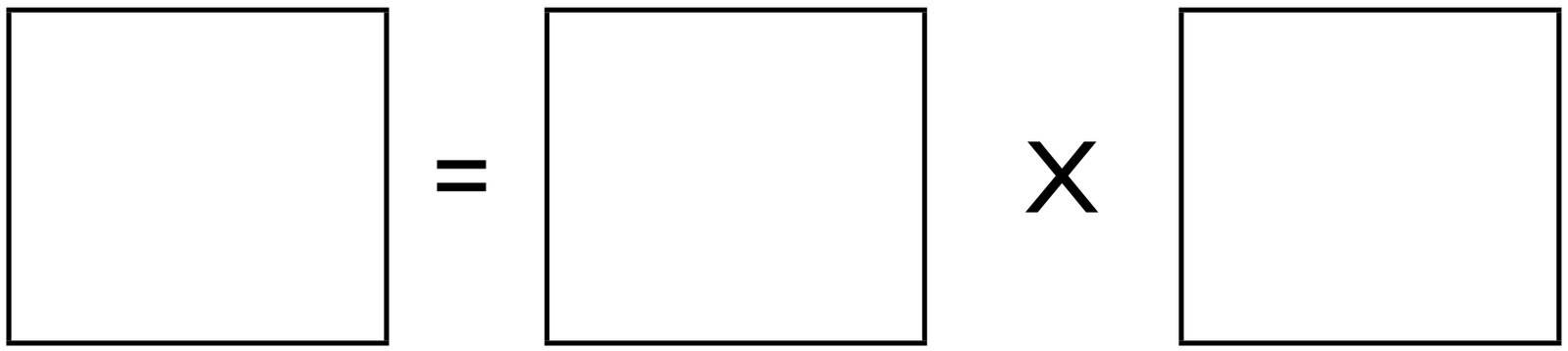
- Input: A,B with n x K and K x m elem. resp.
- Output: C = A x B, where $C_{ij} = \sum_{k=1}^{K} A_{ik} * B_{kj}$.



Matrix Multiplication in MapReduce

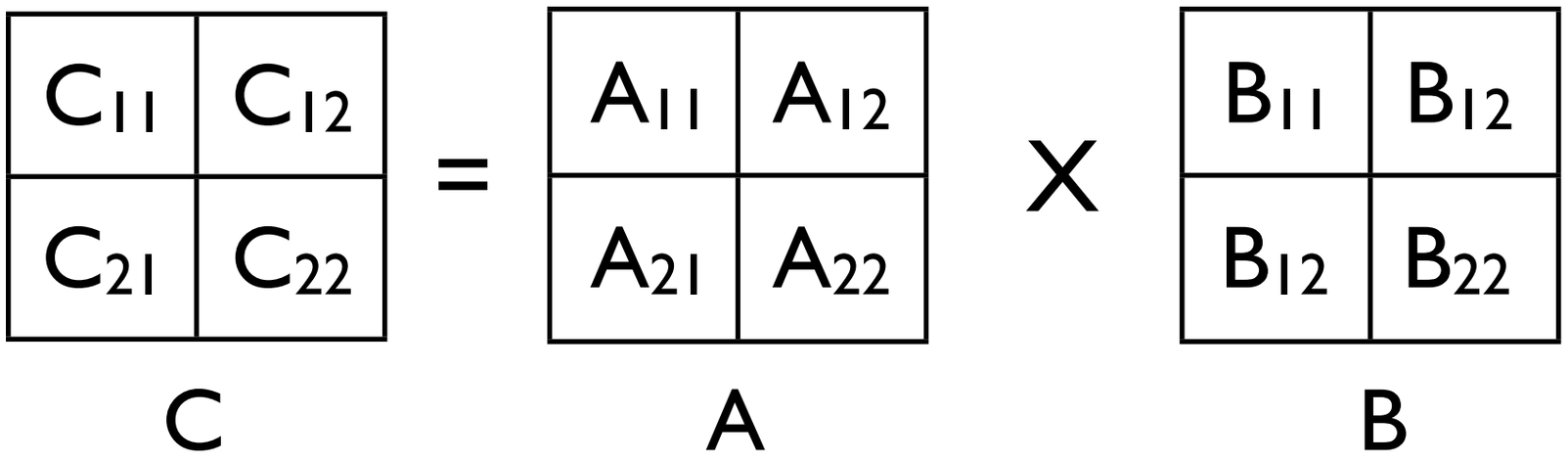
- Main problem: A and/or B might not fit into one single machine's main memory.
- Solution: split A and B into small blocks, so product can be computed in main memory.

Splitting Matrices into Submatrices



$$C_{ij} = \sum_{k=1}^{K} A_{ik} * B_{kj}$$

where A,B,C are submatrices.



Block Matrices Multiplication

or

$$A = \left[\begin{array}{c|c|c} A_{11} & A_{12} & A_{13} \\ \hline A_{21} & A_{22} & A_{23} \end{array}\right], \qquad B = \left[\begin{array}{c|c} B_{11} & B_{12} \\ \hline B_{21} & B_{22} \\ \hline B_{31} & B_{32} \end{array}\right]$$

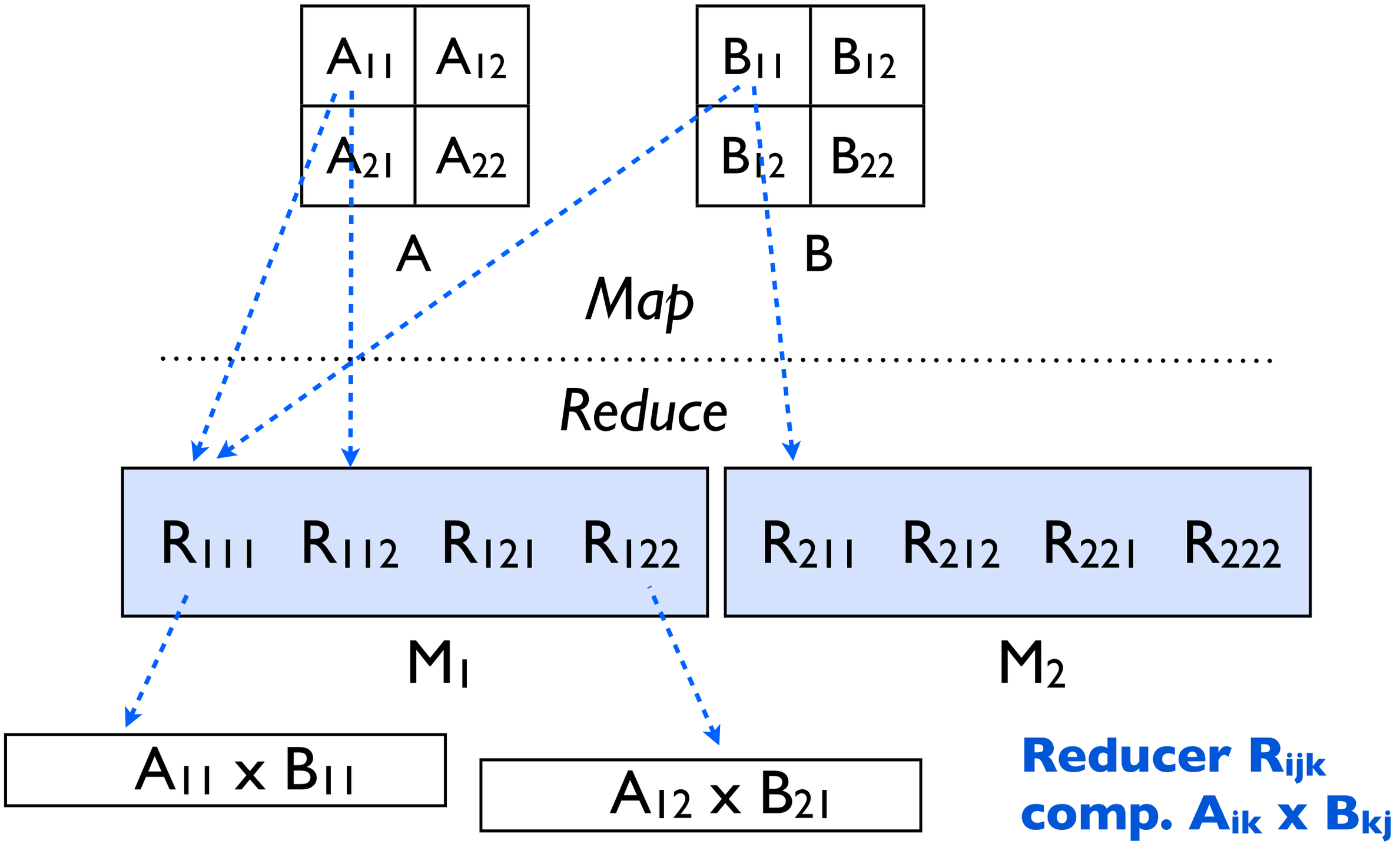
Then the product is given by

$$AB = \begin{bmatrix} A_{11}B_{11} + A_{12}B_{21} + A_{13}B_{31} & A_{11}B_{12} + A_{12}B_{22} + A_{13}B_{32} \\ A_{21}B_{11} + A_{22}B_{21} + A_{23}B_{31} & A_{21}B_{12} + A_{22}B_{22} + A_{23}B_{32} \end{bmatrix} = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = C$$

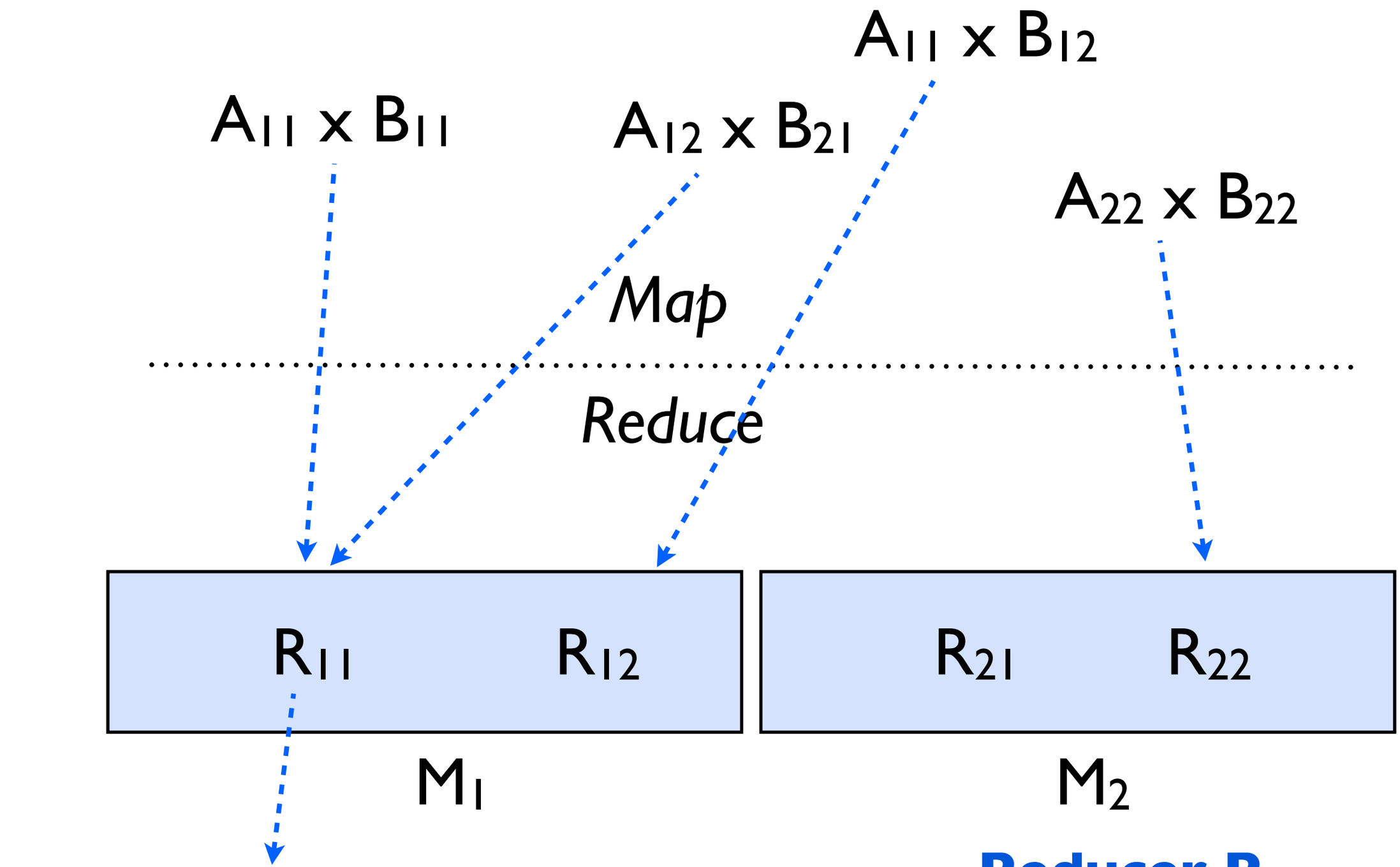
MapReduce Jobs

- Two MapReduce jobs.
  - First job. One reducer R_{ijk} for each i,j,k computing A_{ik} x B_{kj}. Mappers must route a copy of each A_{ik} and a copy of each B_{kj} to all reducers R_{ijk}.
  - Second job computes the sum.

First MapReduce Job



Second MapReduce Job



MapReduce Jobs

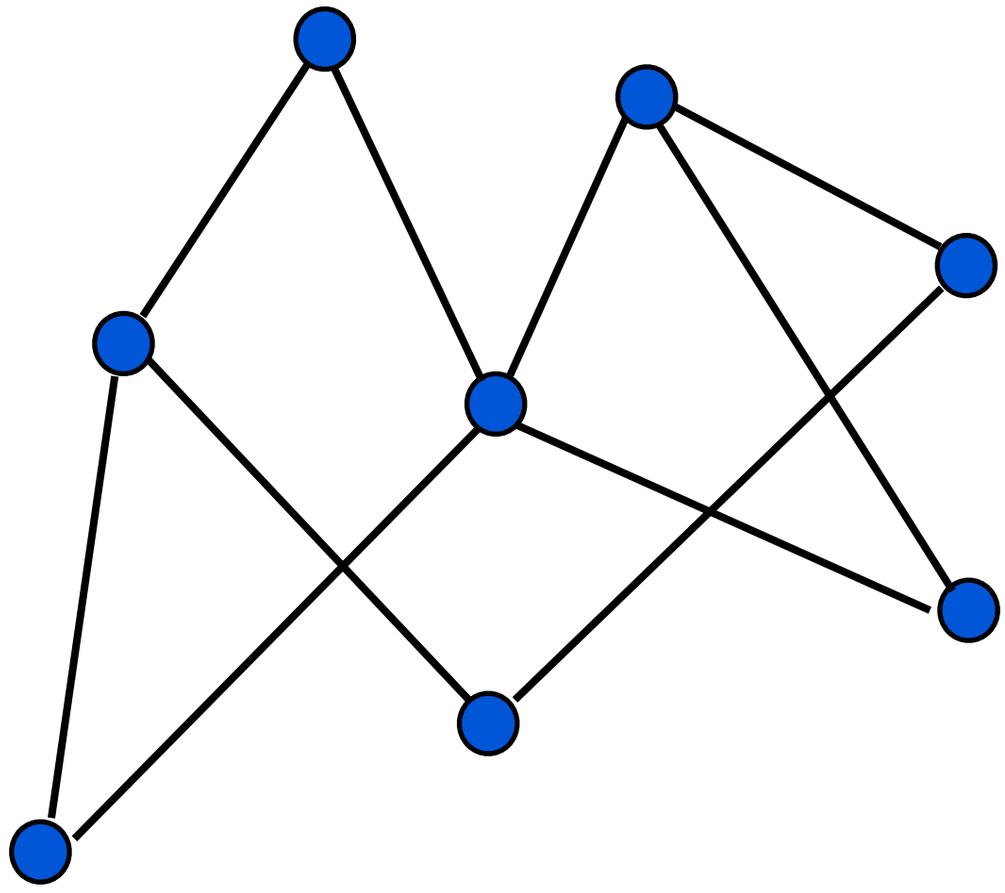
- Two MapReduce jobs.
  - First job. One reducer R_{ijk} computing A_{ik} x B_{kj}, for each i,j,k. Mappers must route a copy of each A_{ik} and a copy of each B_{kj} to all reducers R_{ijk}.
  - Second job computes the sum.

- + There are many reducers -> good parallelization.
- - It requires a large amount of network traffic.

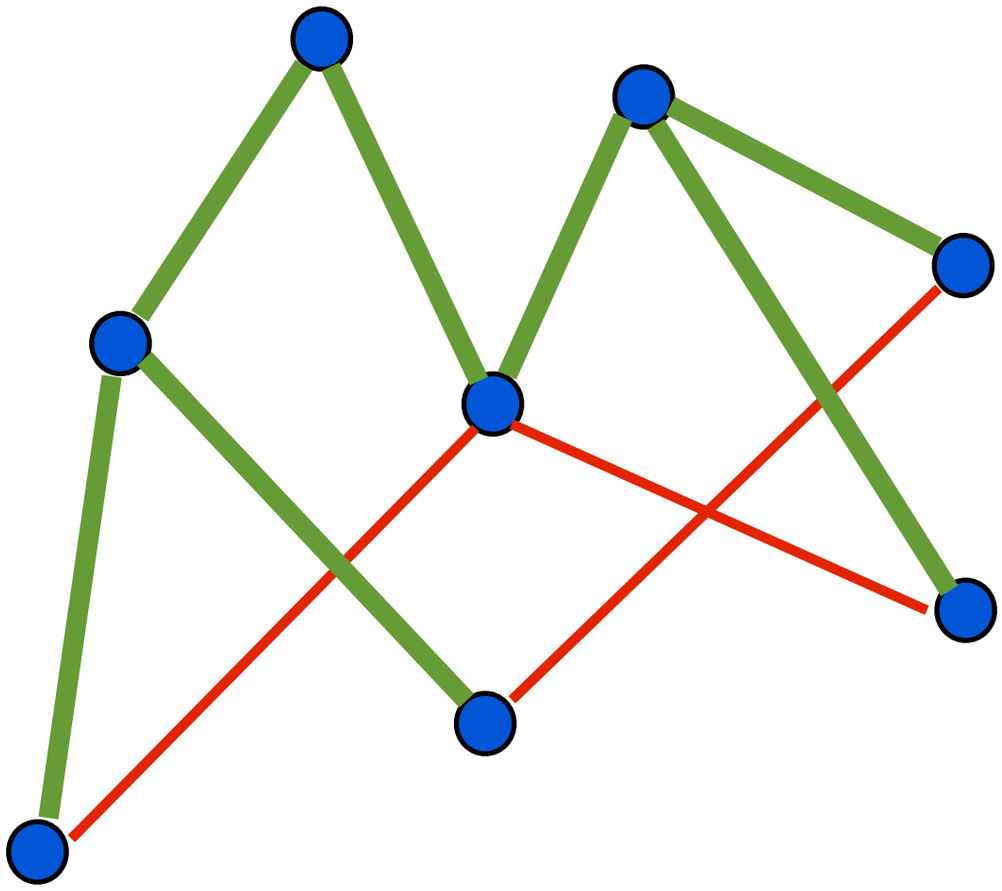
Minimum Spanning Tree

- Definition: Given a graph G=(V,E) and a cost function c : E -> R^+, we seek to find a subset T of E such that:
  - T is connected.
  - T does not contain cycles.
  - T has minimum total cost.
- There is a simple algorithm (greedy) computing the optimum.

Minimum Spanning Tree



Minimum Spanning Tree



Greedy algo for MST

- Order the edges $e_1,..,e_m$ by non-decreasing cost.
- At each step t include e_t, if there is no cycle.
- Iterate until all edges have been processed.

- Can we parallelize it efficiently in MapReduce?

Greedy2

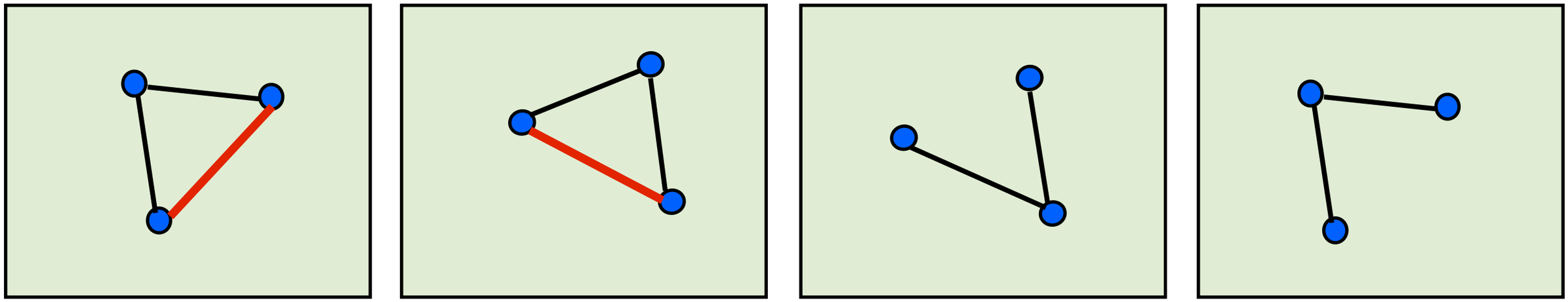
- At each step, find a cycle C and remove an edge with maximum cost in C.
- Iterate until there is no cycle.
- Much easier to parallelize.

MRGreedy

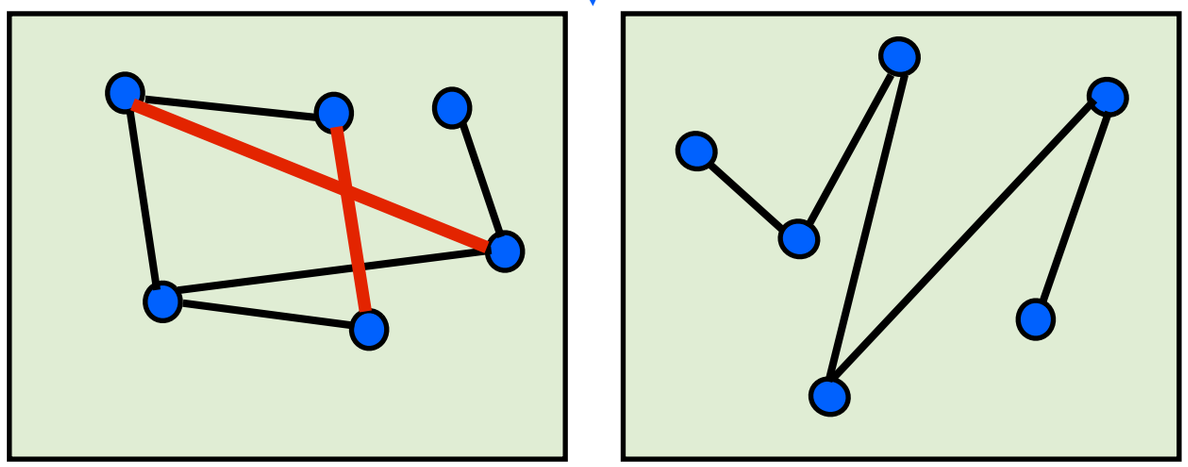
- Assumption: main memory is at least 4n bytes (n= number of nodes).
- Partition edges randomly.
- Each reducer executes Greedy2 on "its" set of edges.
- At each iteration, we halve the number of machines and iterate, until only one reducer is left.

MST in MapReduce

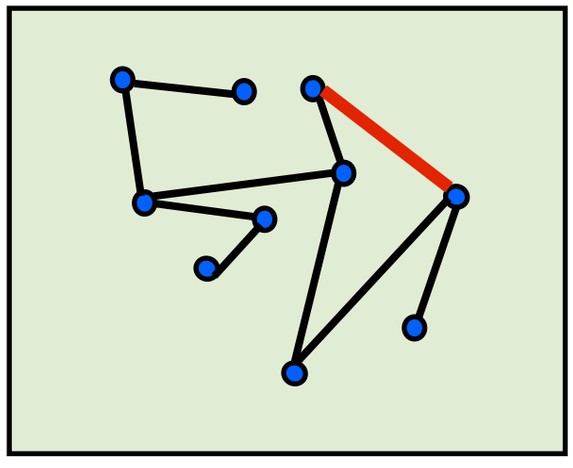
1



2



3



MRGreedy

- Assumption: main memory is at least 8n bytes (n= number of nodes).
- Partition edges randomly.
- Each reducer executes Greedy2 on "its" set of edges.
- At each iteration, we halve the number of reducers and iterate, until only one reducer is left.
- Questions:
  - do we compute a minimum spanning tree?
  - Is the total main memory sufficient?
  - how many iterations do we need?