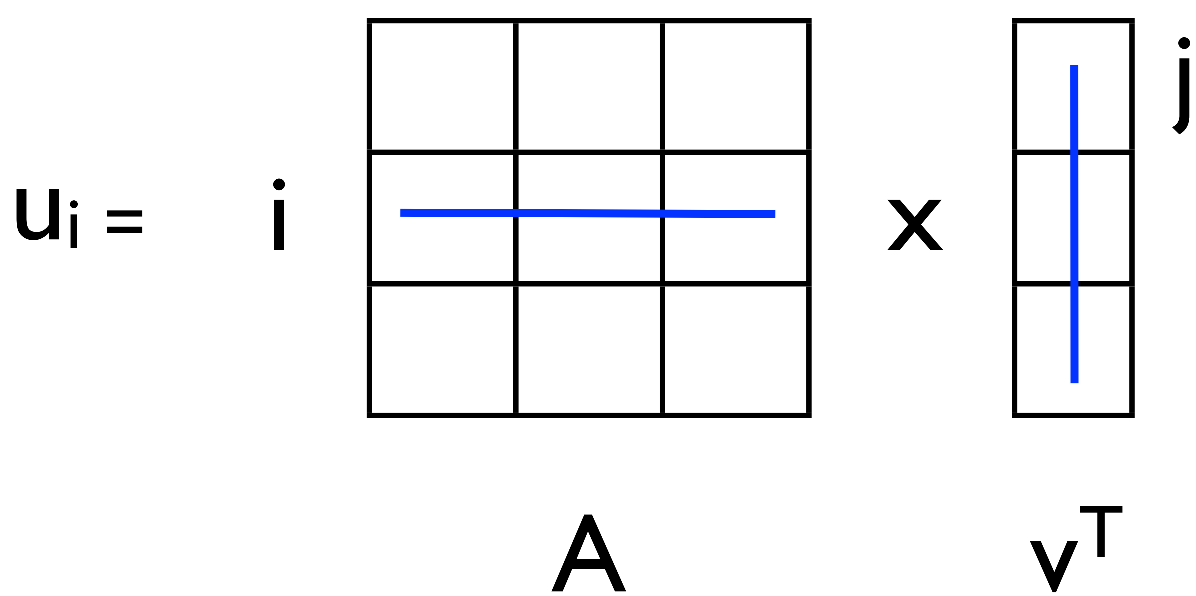
# Parallelizing Algorithms in MapReduce

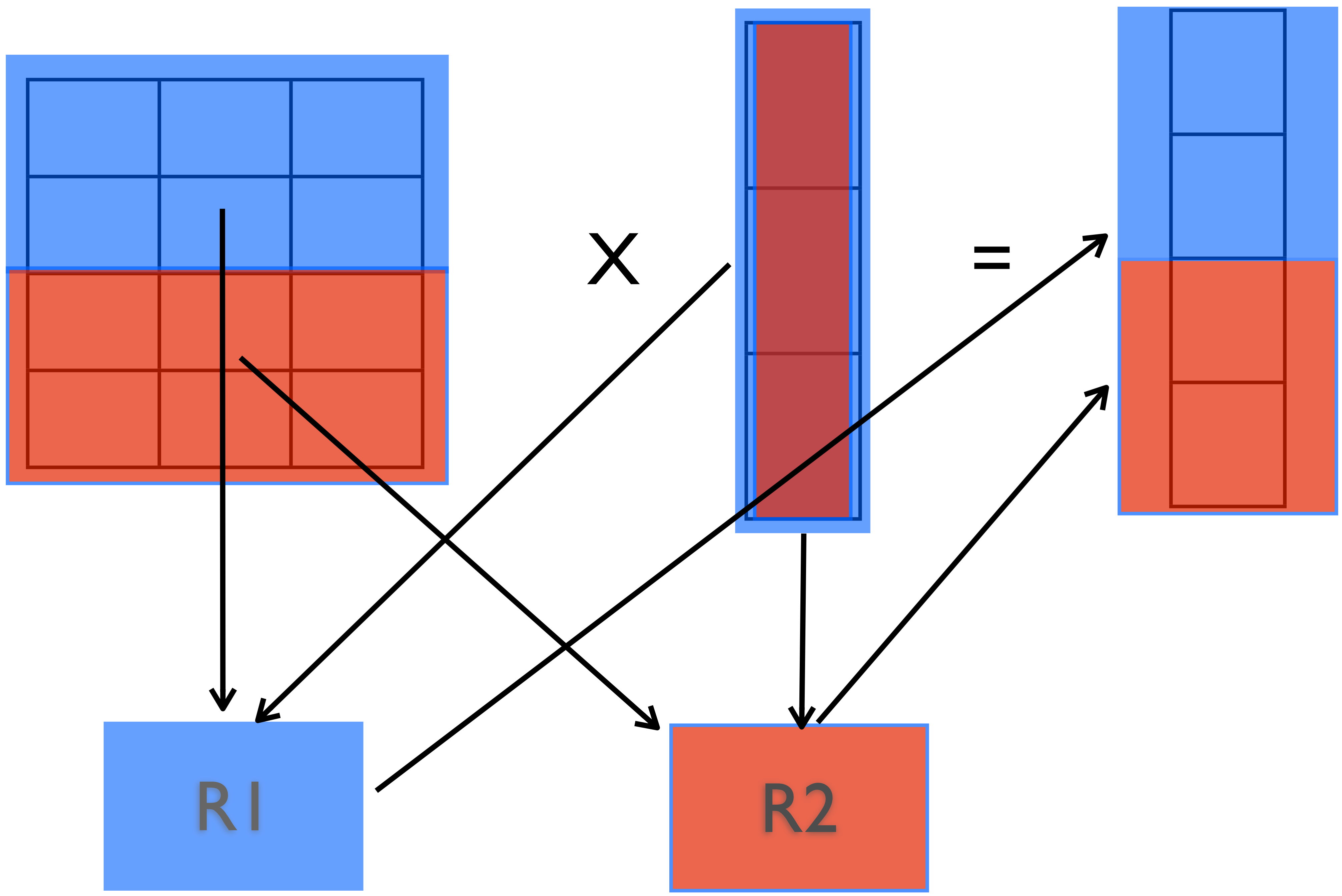
*Mauro Sozio*
*Institut Mines-Telecom*
*sozio@telecom-paristech.fr*

# Matrix-Vector Multiplication in MR

- Input: A,v with n x K and K elem. resp.
- Output: vector u=Av$^T$ i.e. $u_i = \sum_{j=1}^{k} A_{ij} v_j$ .



# Matrix-Vector Multiplication



# MapReduce jobs

Input:

- Sparse matrix (list of <$i,j,A_{ij}$>, where $A_{ij}!=0$), a vector v

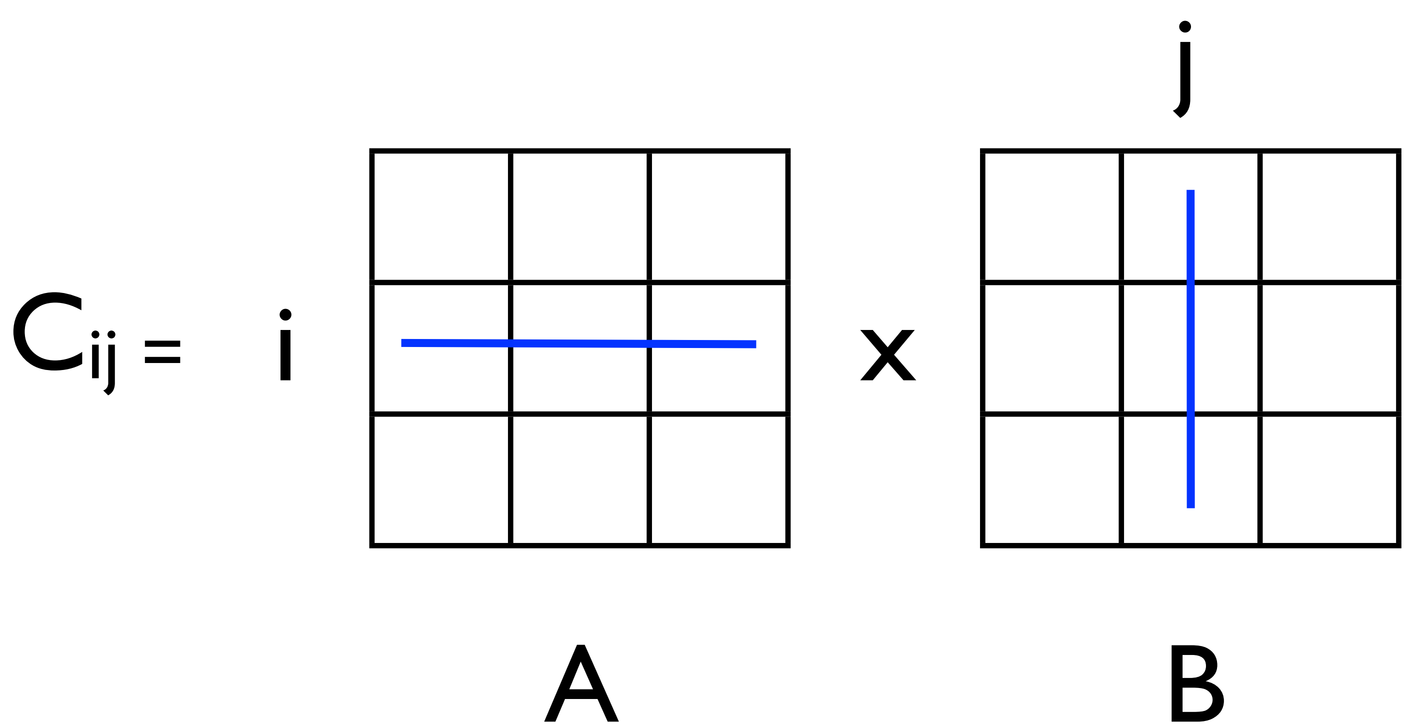
One MapReduce job:

-- Map. Partition A and make copies of v.

-- Reduce. Compute $A_i$ x v and output the results.

# Matrix-Matrix Multiplication in MR

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



# Matrix-Matrix Multiplication in

- 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 to compute products between small blocks in main memory.

# Matrix Partitioning

Partition rows and columns of M (n x K) into k "contiguous" blocks each, so that all k x k submatrices have a same size and each fits into main memory.

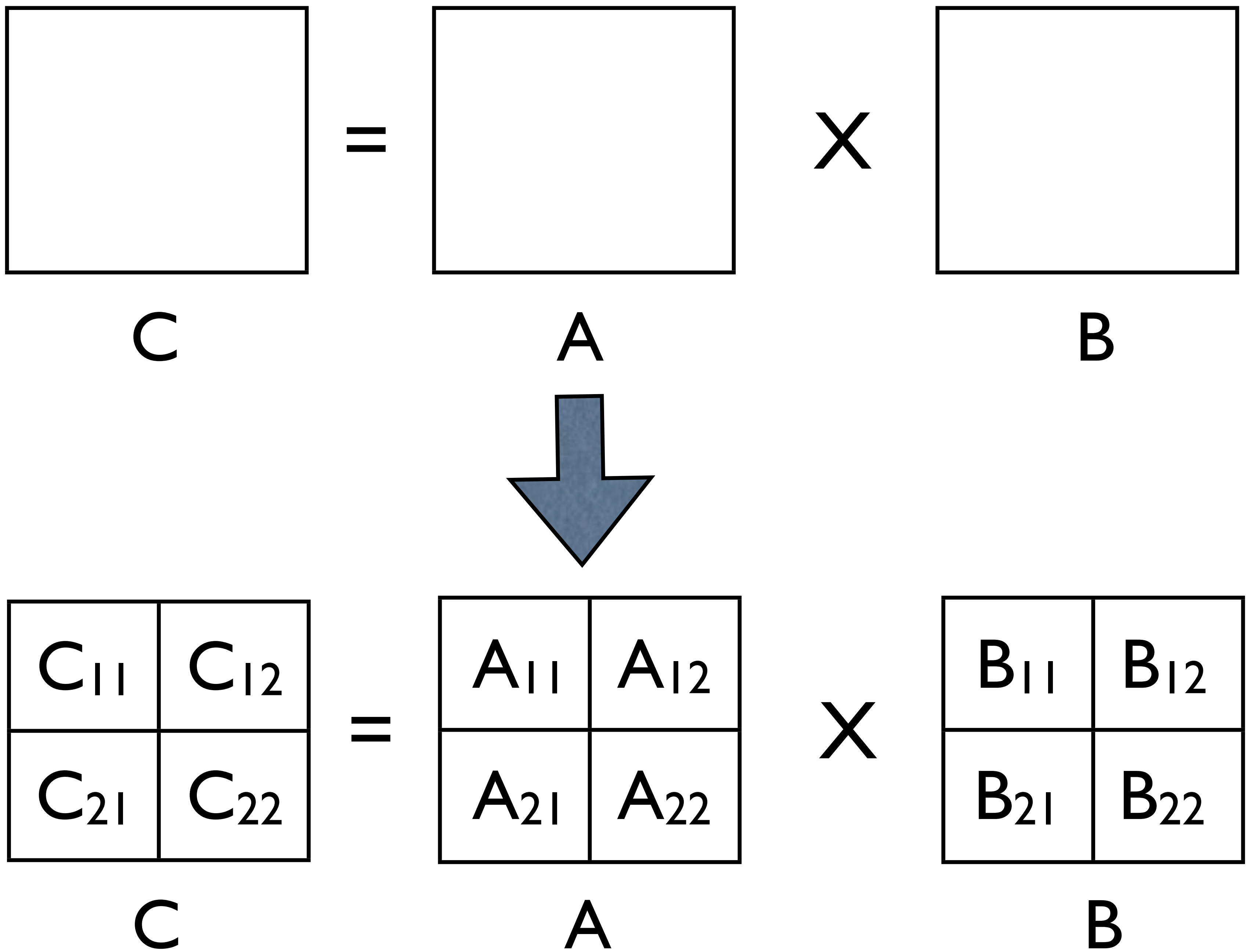
Ex. Partition A into 4 submatrices with a same size (K=2).

| | |
|---|---|
| $A_{11}$ | $A_{12}$ |
| $A_{21}$ | $A_{22}$ |

Formally, $A_{ij}$ contains all elements with:

- rows in $[(i-1)n/k+1, i \times n/k]$, i in $[1,...,k]$,
- cols in $[(j-1)K/k+1, j \times K/k]$, j in $[1,...,k]$.

# Matrix Multiplication




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

where $A_{ik}$,$B_{kj}$,$C_{ij}$ are submat. of A,B,C.

# Matrix Multiplication

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

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:

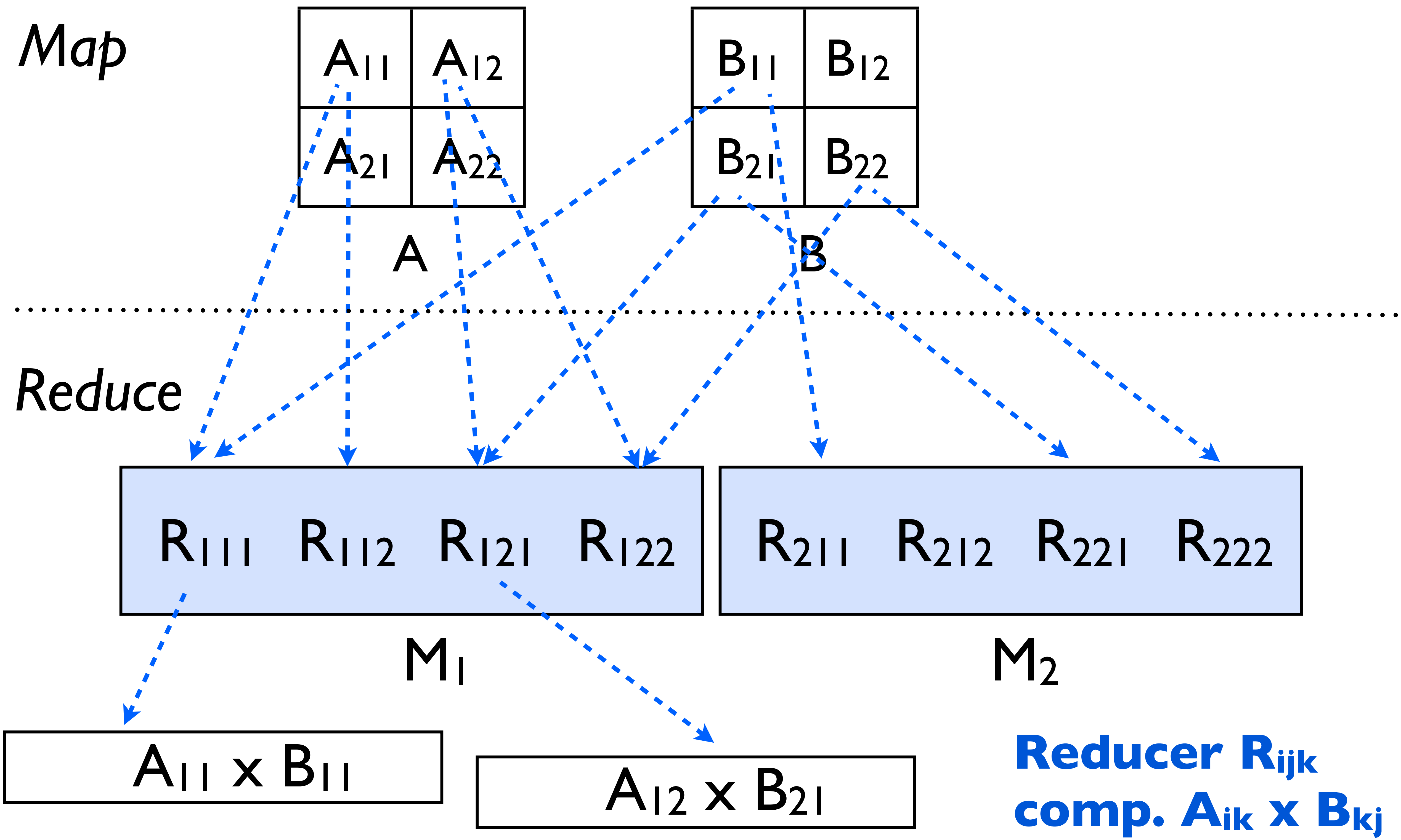
-- *first job*. One reducer $R_{ijk}$ for each i,j,k computing $A_{ik} \times B_{kj}$. Mappers must route a copy of each $A_{ik}$ and a copy of each $B_{kj}$ to all reducers $R_{ijk}$;

-- *second job*. One reducer $R_{ij}$ for each i,j computing the sum

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

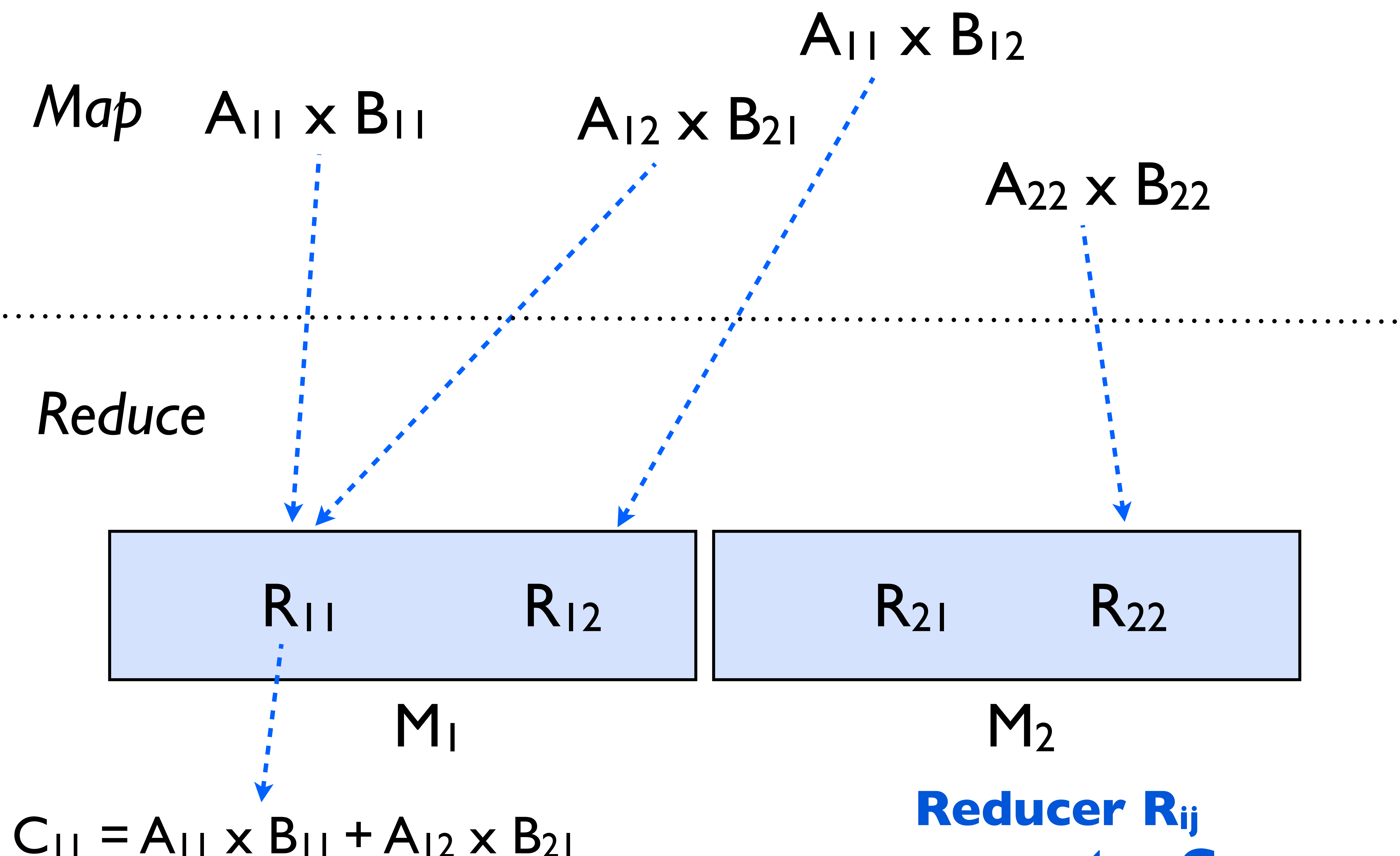
where $A_{ik} \times B_{kj}$ has been computed by $R_{ijk}$.

# First MapReduce Job




**Reducer $R_{ijk}$ comp. $A_{ik} \times B_{kj}$**

# Second MapReduce Job



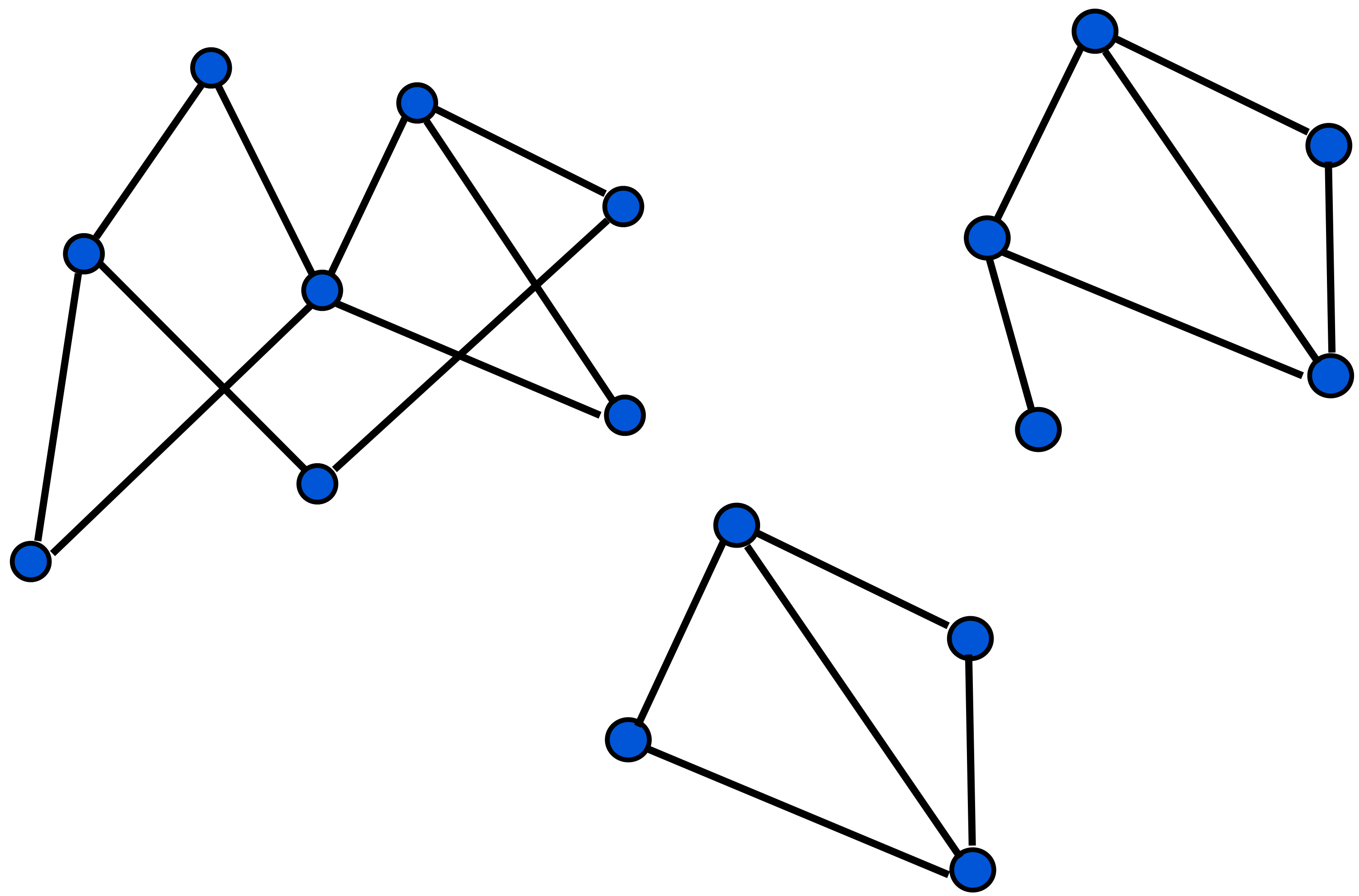

**Reducer $R_{ij}$ computes $C_{ij}$**

# Connected Components

Definition: Given a graph $G=(V,E)$, we say that two nodes $u$ and $v$ are in a same connected component if $u$ and $v$ are *connected* in G.

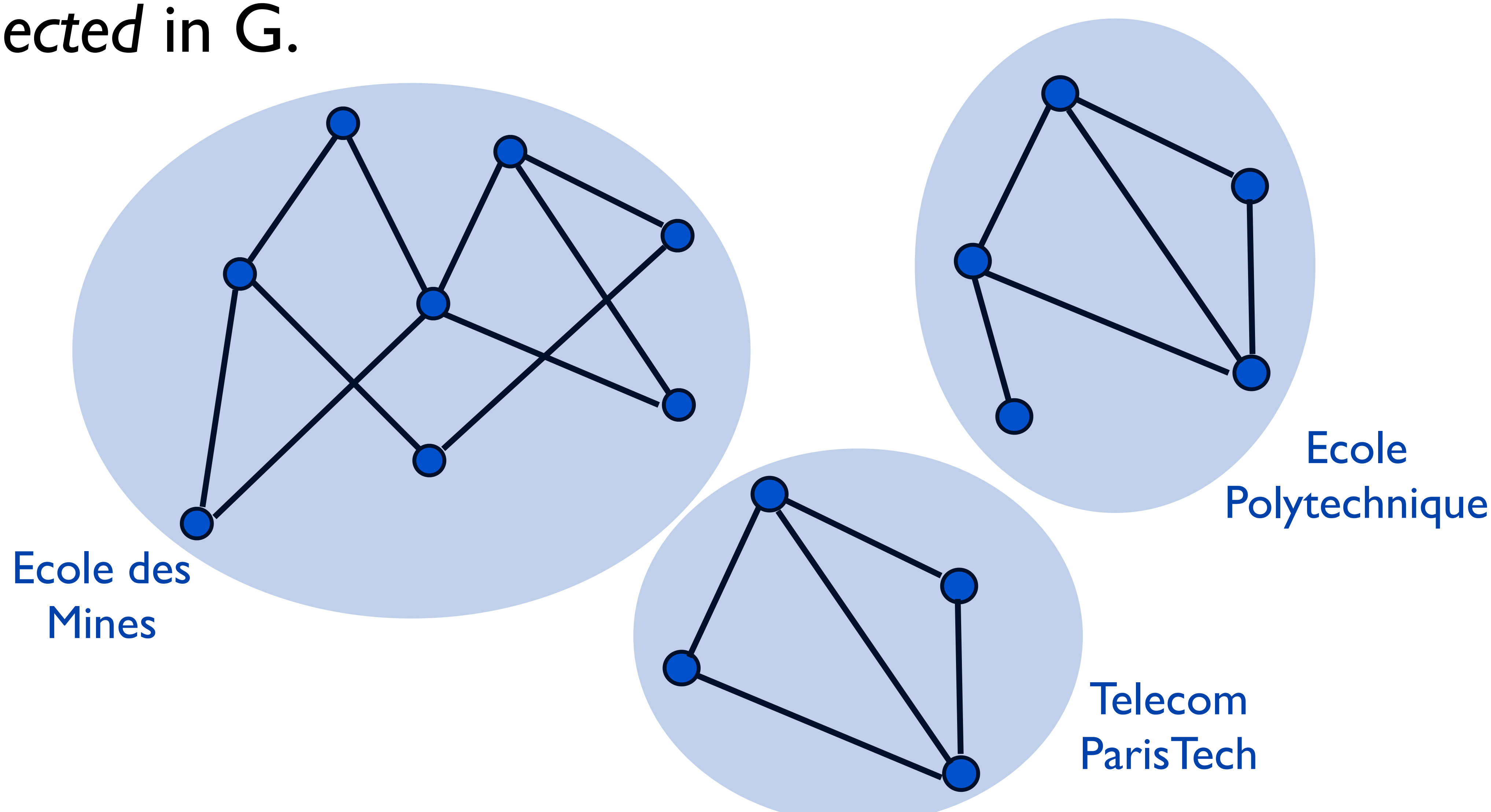
# Connected Components

Definition: Given a graph *G=(V,E)*, we say that two nodes *u* and *v* are in a same connected component if *u* and *v* are *connected* in G.



# Connected Components

Definition: Given a graph *G=(V,E)*, we say that two nodes *u* and *v* are in a same connected component if *u* and *v* are *connected* in G.



# Finding Connected Components in Large Graphs

*Two main issues:*

-- graph might not fit into main memory;
-- graphs like Facebook contain > $3 \times 10^{11}$ friendship links... computation is expensive!

Algorithm in MapReduce?

# Finding Connected Components in MapReduce

Assumption: up to 4 times the number of users (4Gb) can be stored into main memory. The set of links is too large! (several terabytes).

Algorithm in MapReduce?

# MapReduce Algorithm

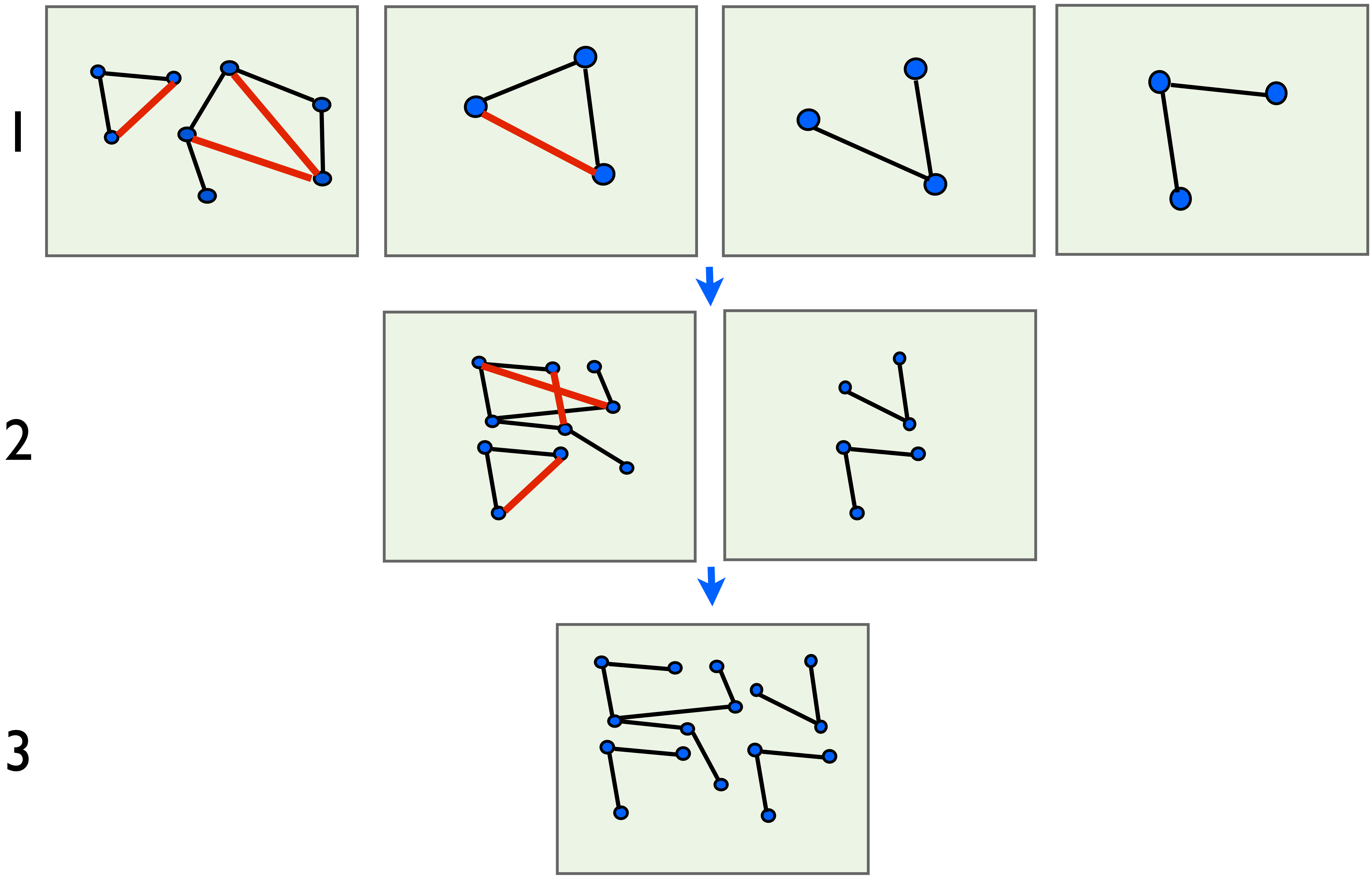
At each iteration:

- Partition the links of the input graph into several “chunks”, randomly, so that each chunk fits into machine main memory.

- Each machine in parallel
  - computes the set of connected components (in its chunk);
  - removes edges that do not contribute to connectivity.

- “Turn off” half of the available machines.

# Connected Components in MapReduce



# MR Algorithm correct?

Questions:

- Correct?
- Graphs always fit into main memory?
- With m machines, how many iterations?

# MR Algorithm correct?

Questions:

- Correct? (yes we remove only "superfluous" edges)
- Graphs always fit into main memory? (at step k after removing "superfluous" edges the number of links is at most $m_k$*n where $m_k$ is the current number of machines and n is the number of nodes. At step k+1 each of the $m_k/2$ machines gets at most 2*n edges.)
- With m machines, how many iterations? (at most $\log_2 m$)

Exercise: compute min. spanning tree in MapReduce.