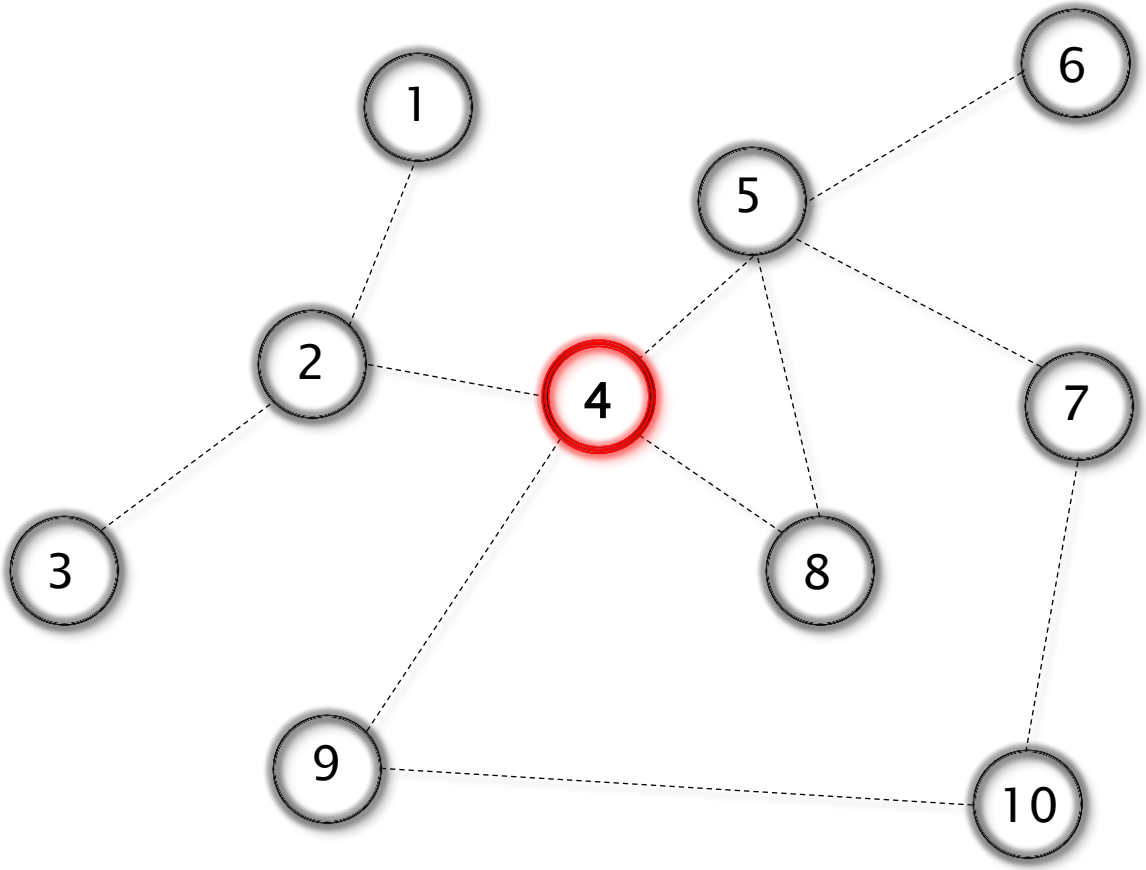
The Implementation of the Breadth First Search Algorithm for Clusters with GPUs.

Kolganov A.S.

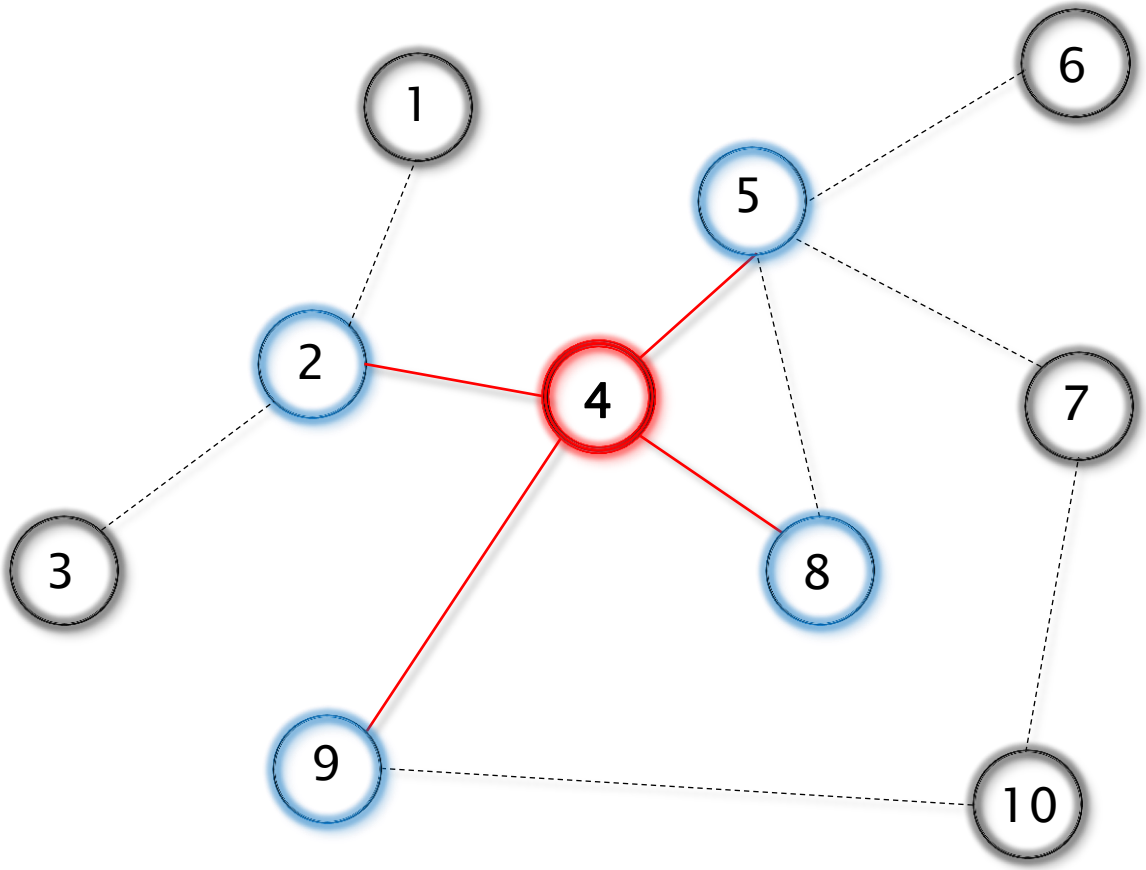
Outline

- Breadth First Search, BFS;
- Implementation of BFS on single GPU;
- Overview of input data format;
- Mapping onto cluster;
- Some optimizations;
- Results.

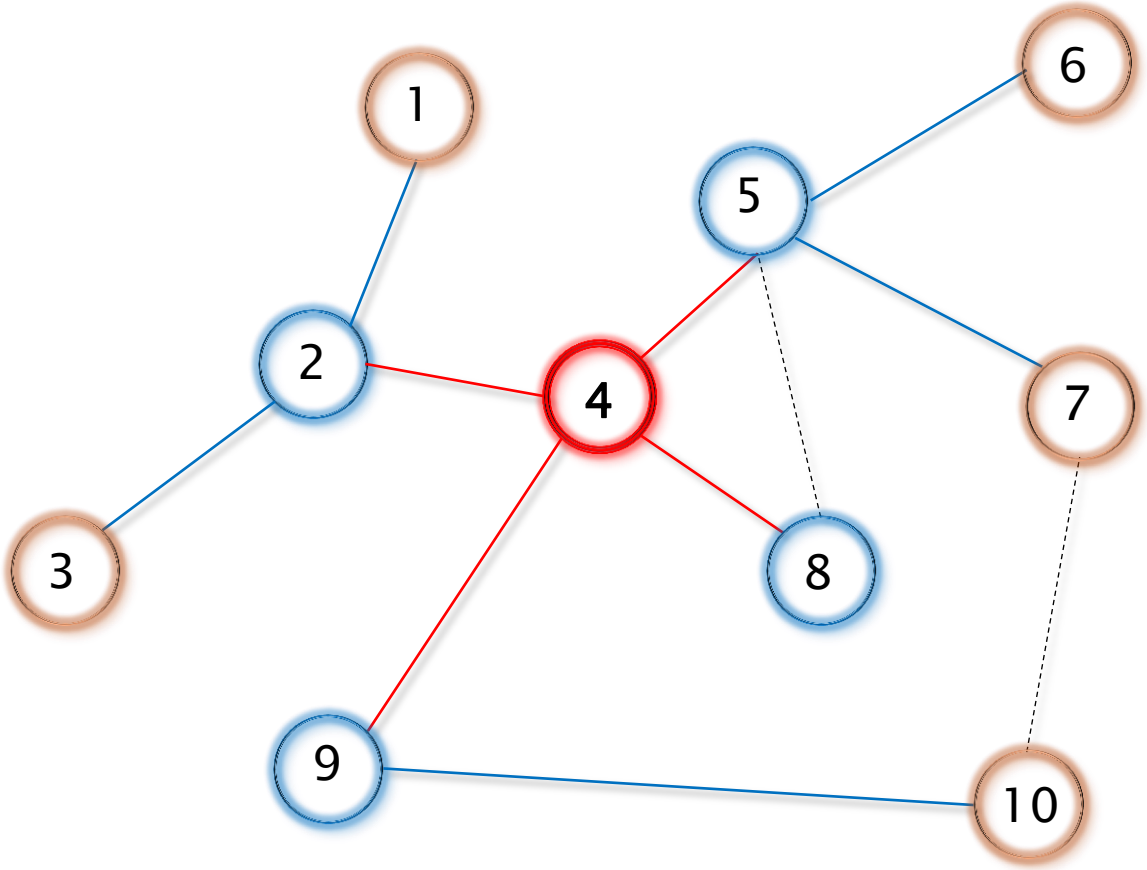
Breadth First Search



Breadth First Search STEP 1



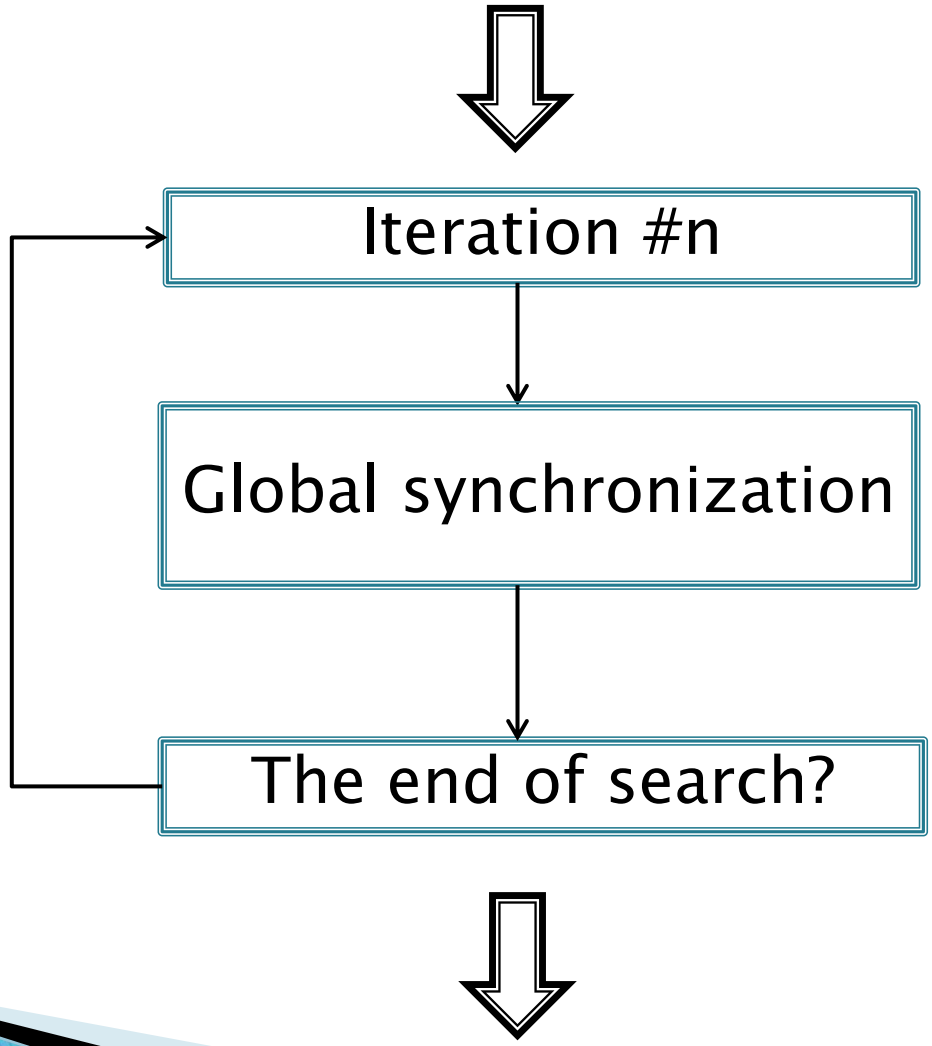
Breadth First Search STEP 2



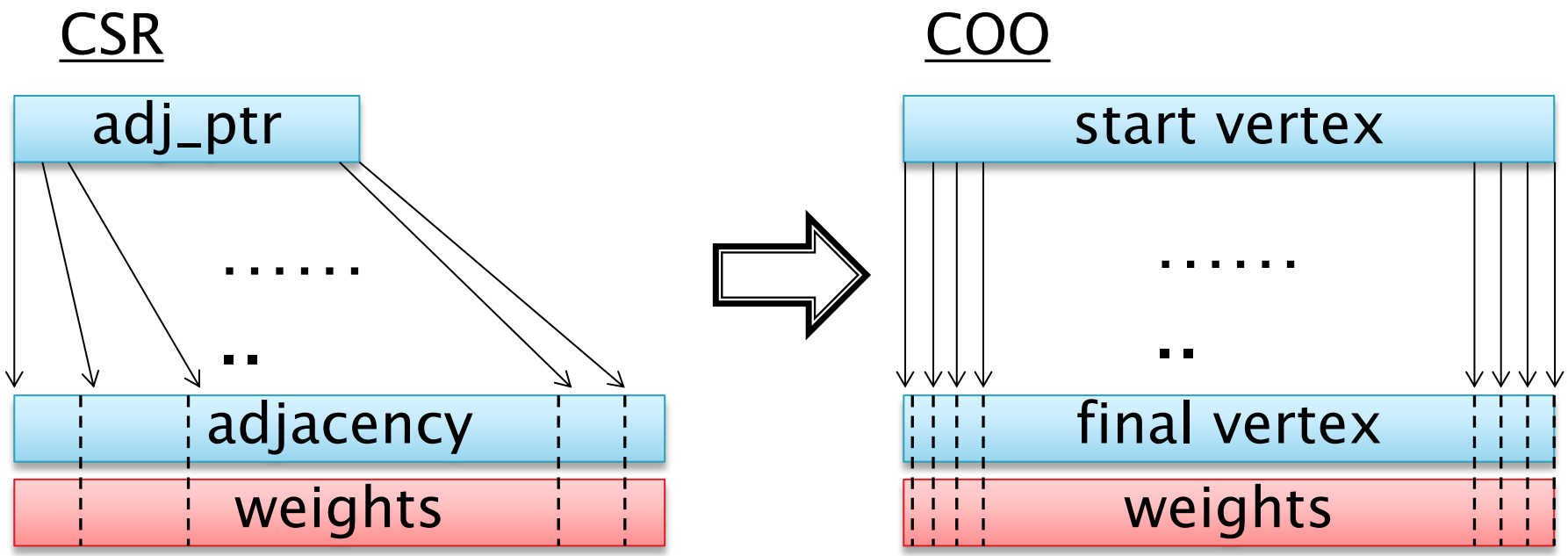
The main ideas of BFS algorithm

- The iterative marking of BFS levels;
- Transformation of input data format;
- Optimizations of GPU memory access;
- Optimizations of interprocessor communications.

Scheme of BFS execution

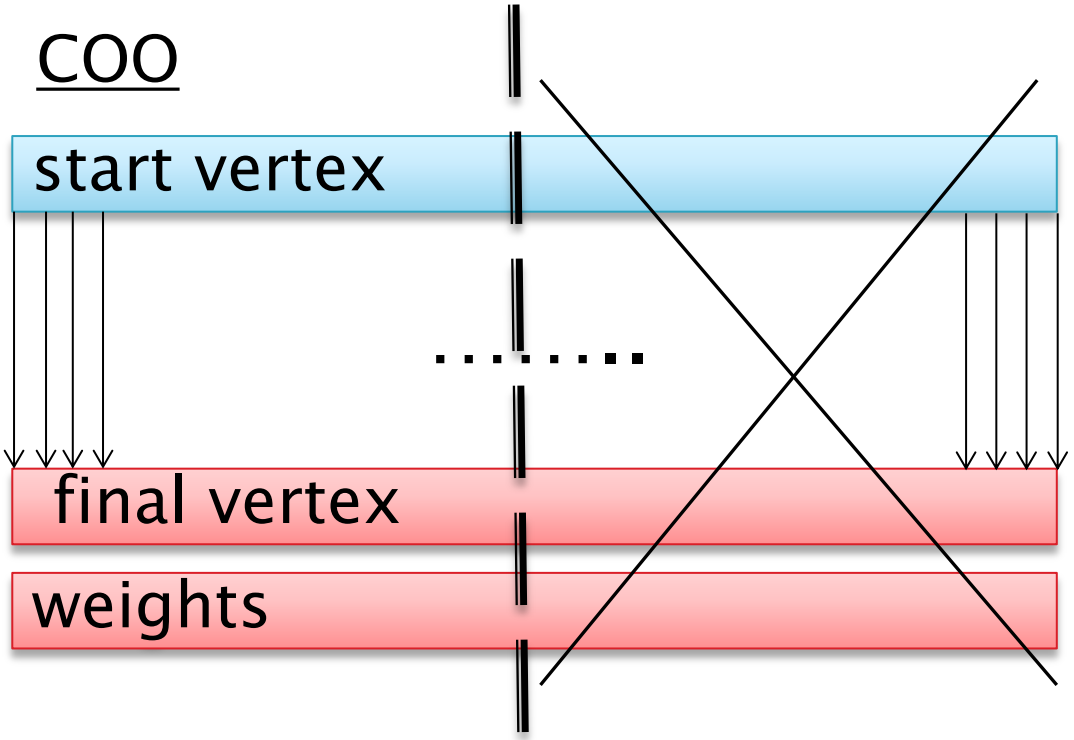


Data representation



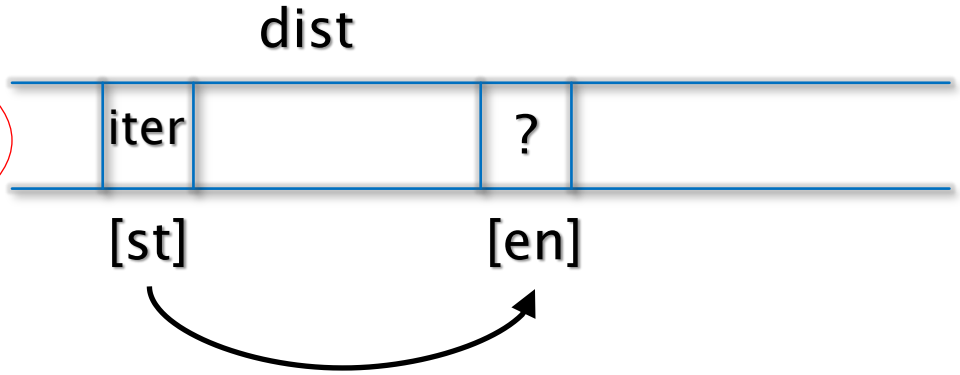
Remove duplicates

- The CSR format includes both edges (S, F) and (F, S), choose one of them – (S, F) or (F, S).



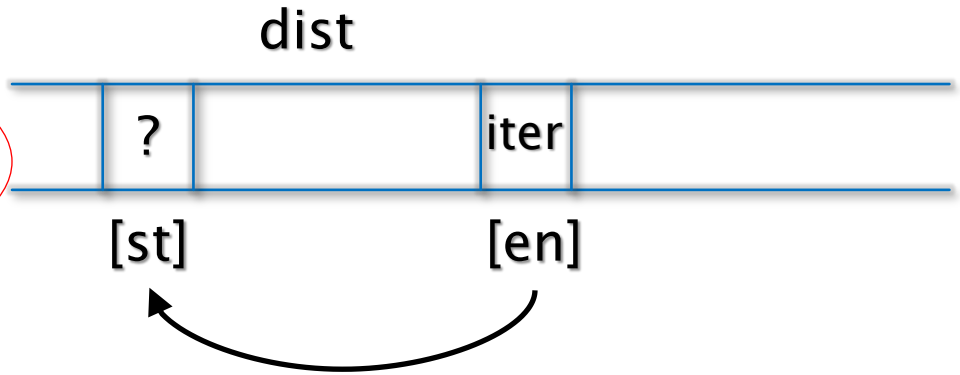
The main iteration

```
if(k < maxV)
{
    unsigned st = startV[k]
    unsigned en = endV[k];

    if(dist[st] == iter)
    {
        if(dist[en] > iter)
            dist[en] = iter + 1;
    }
    else if(dist[en] == iter)
    {
        if(dist[st] > iter)
            dist[st] = iter + 1;
    }
}
```



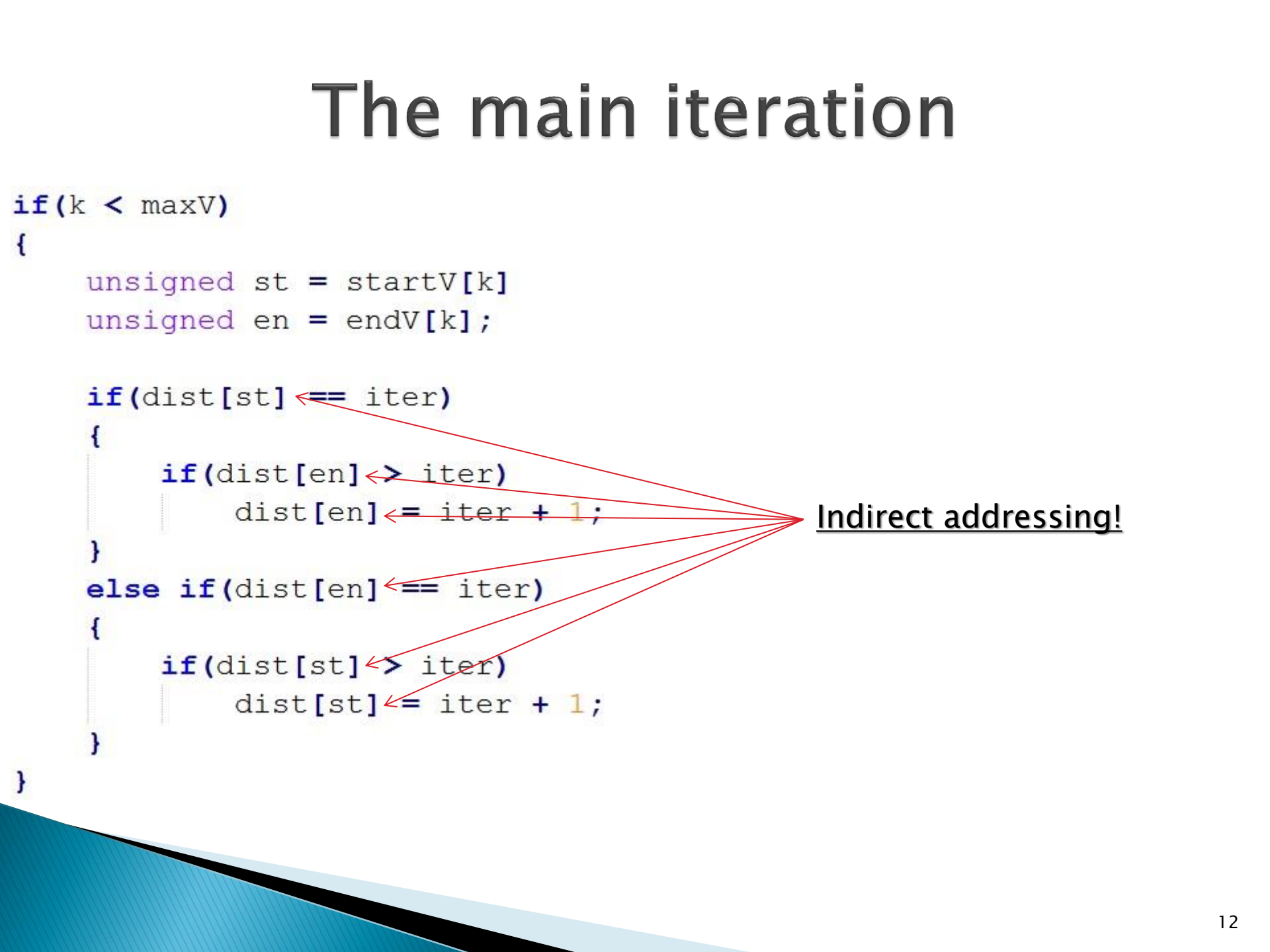
The main iteration

```
if(k < maxV)
{
    unsigned st = startV[k]
    unsigned en = endV[k];

    if(dist[st] == iter)
    {
        if(dist[en] > iter)
            dist[en] = iter + 1;
    }
    else if(dist[en] == iter)
    {
        if(dist[st] > iter)
            dist[st] = iter + 1;
    }
}
```



The main iteration

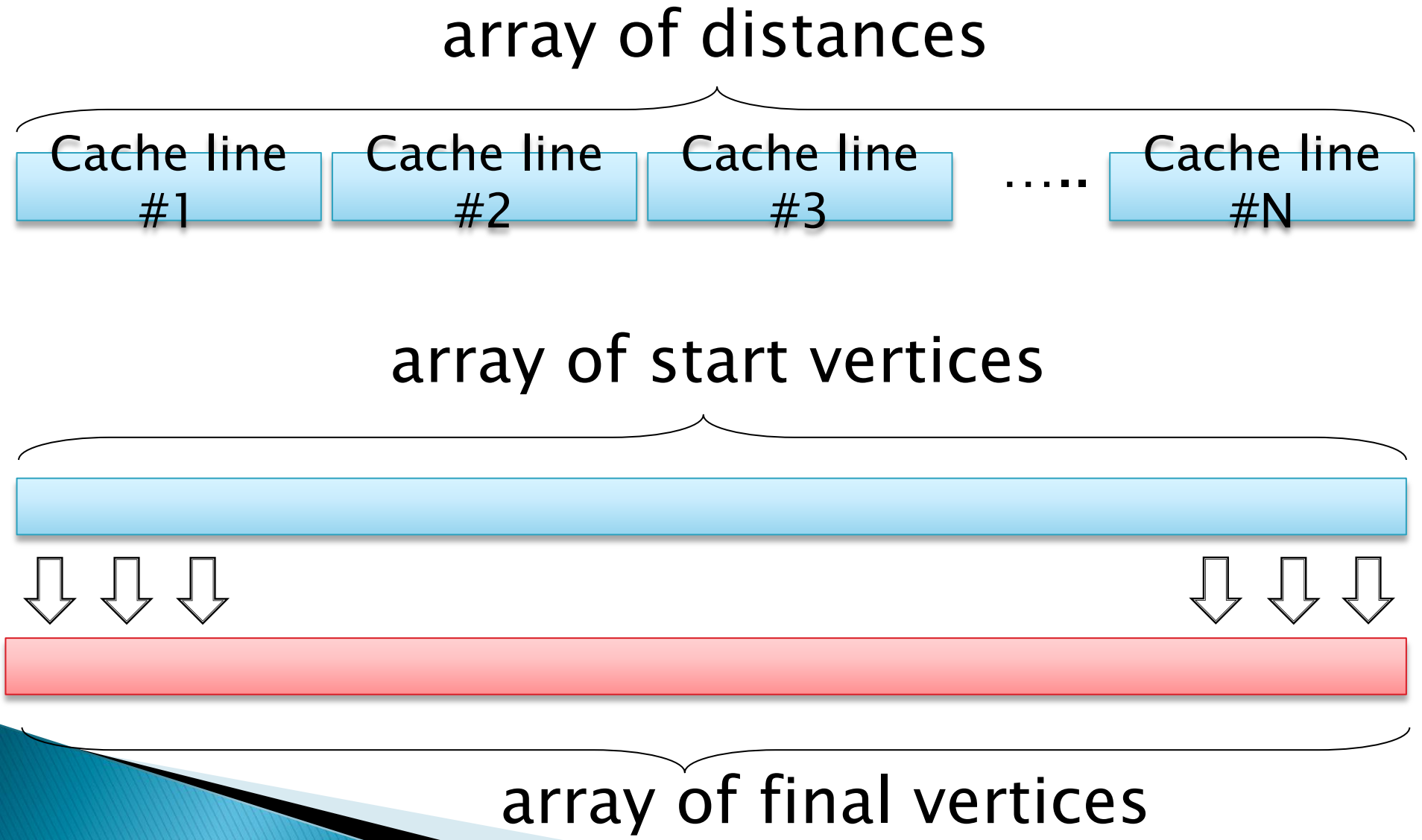
```
if(k < maxV)
{
    unsigned st = startV[k]
    unsigned en = endV[k];

    if(dist[st] == iter)
    {
        if(dist[en] > iter)
            dist[en] = iter + 1;
    }
    else if(dist[en] == iter)
    {
        if(dist[st] > iter)
            dist[st] = iter + 1;
    }
}
```



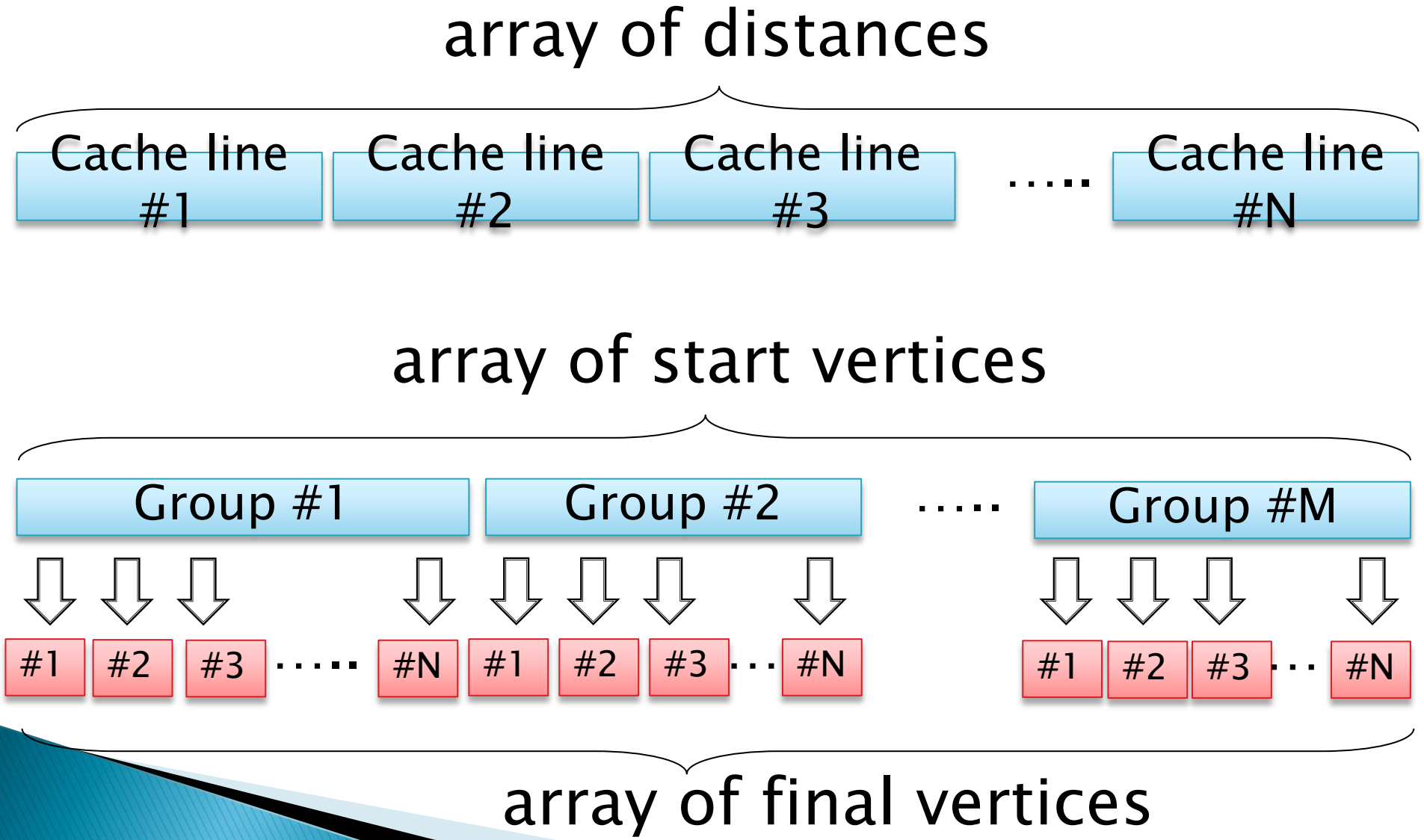
Optimizations

- Data transformation for efficient use of the cache and the GPU memory bandwidth;

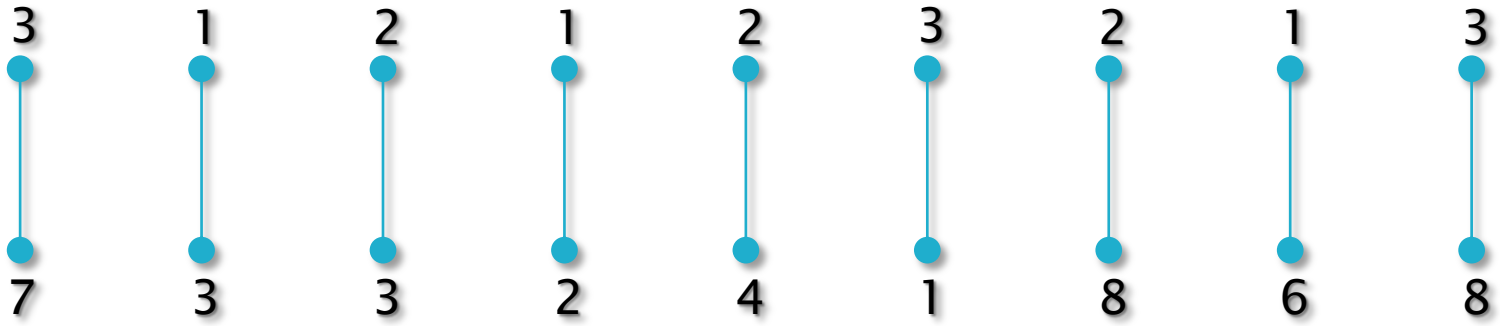
Data transformation



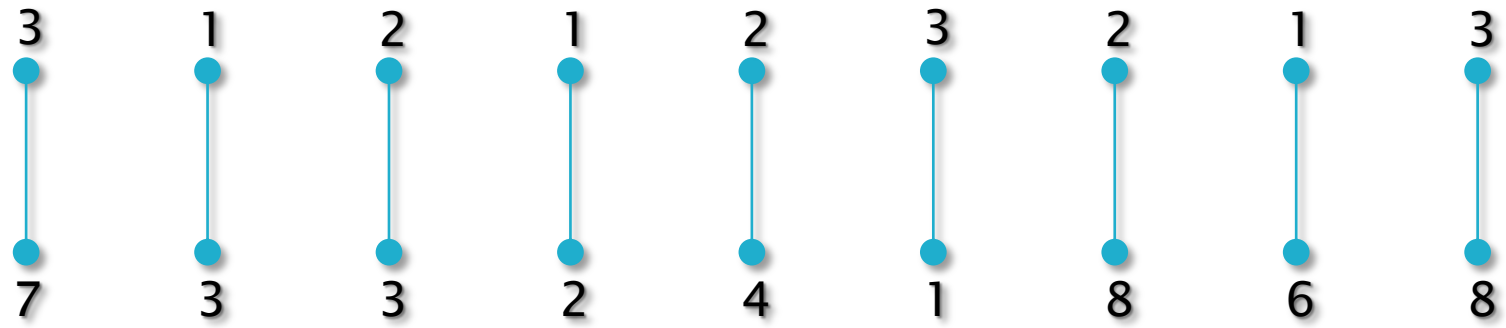
Data transformation



Example of transform



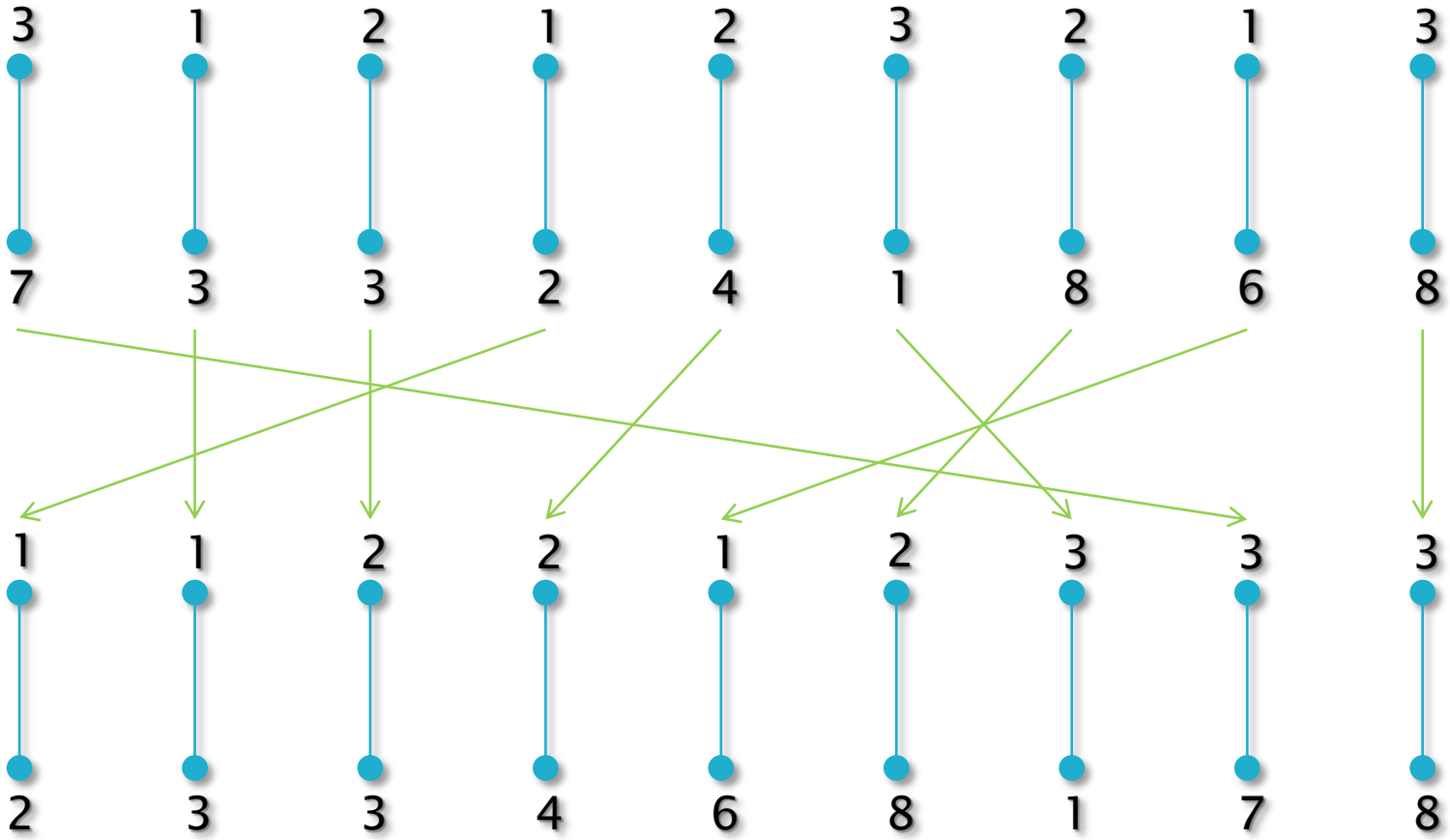
Example of transform



Cache Line = 2

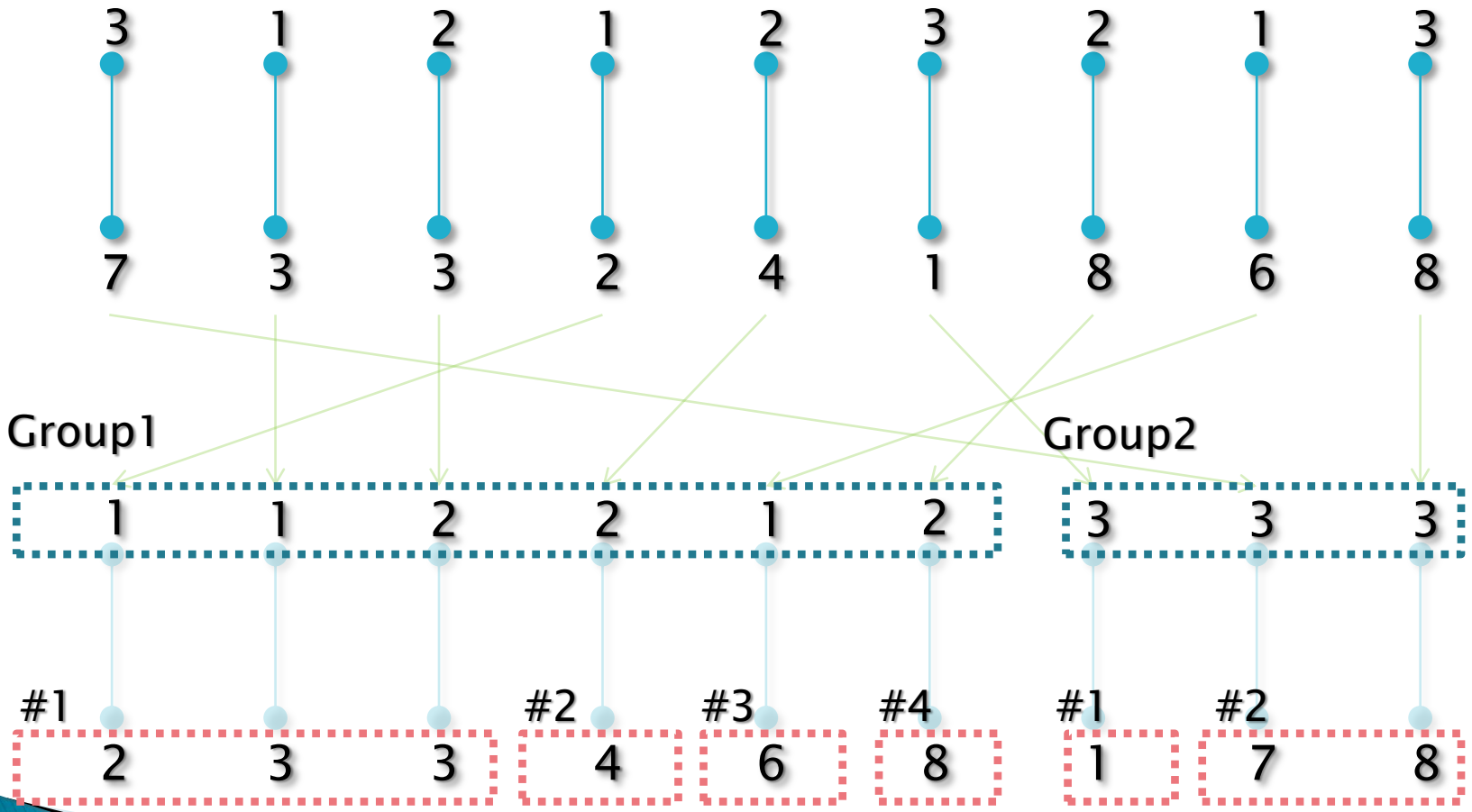
Example of transform

Cache Line = 2

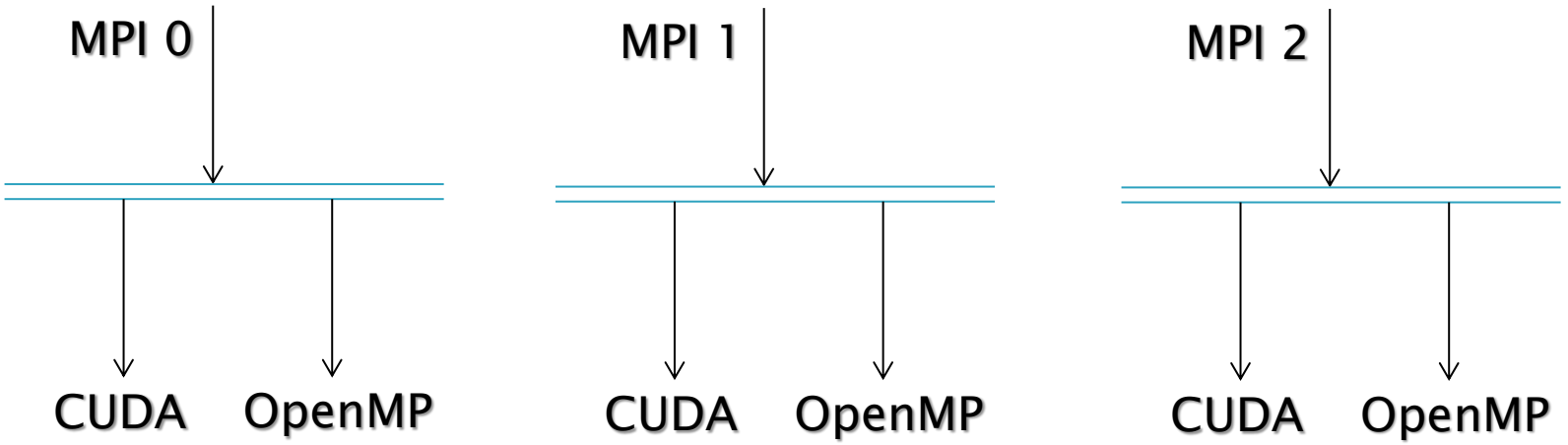


Example of transform

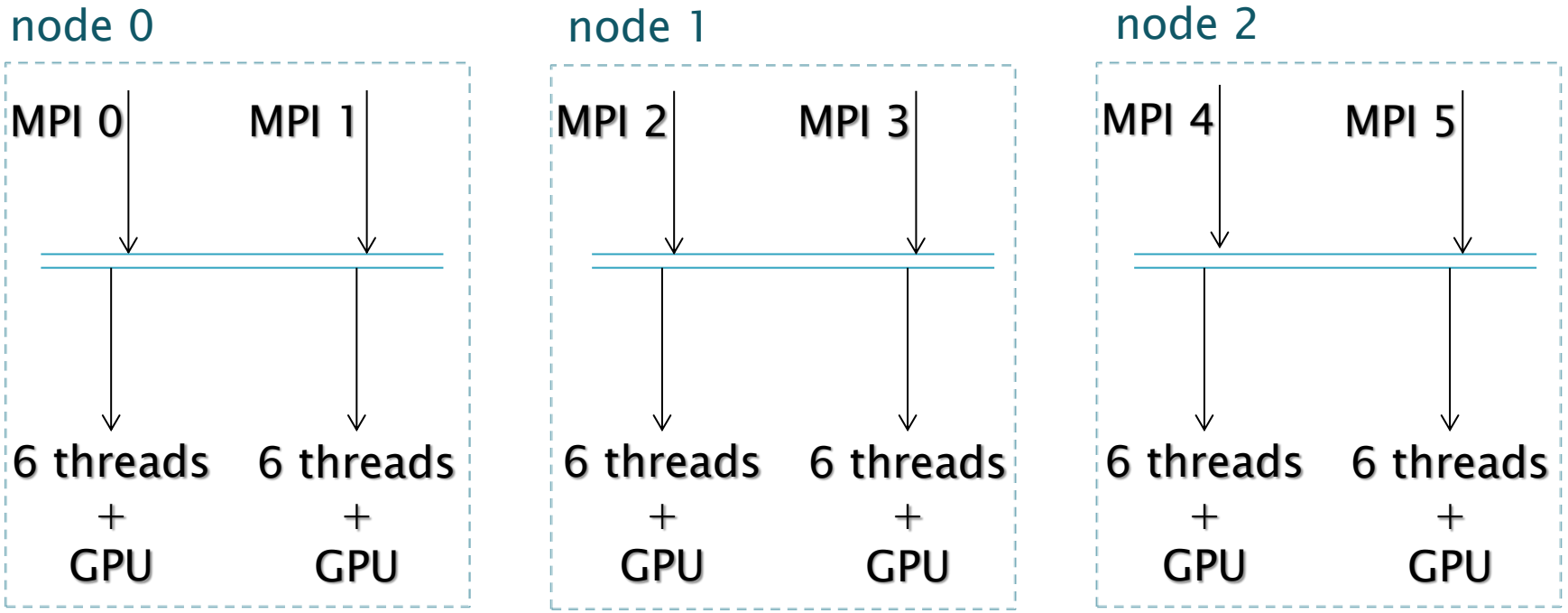
Cache Line = 2



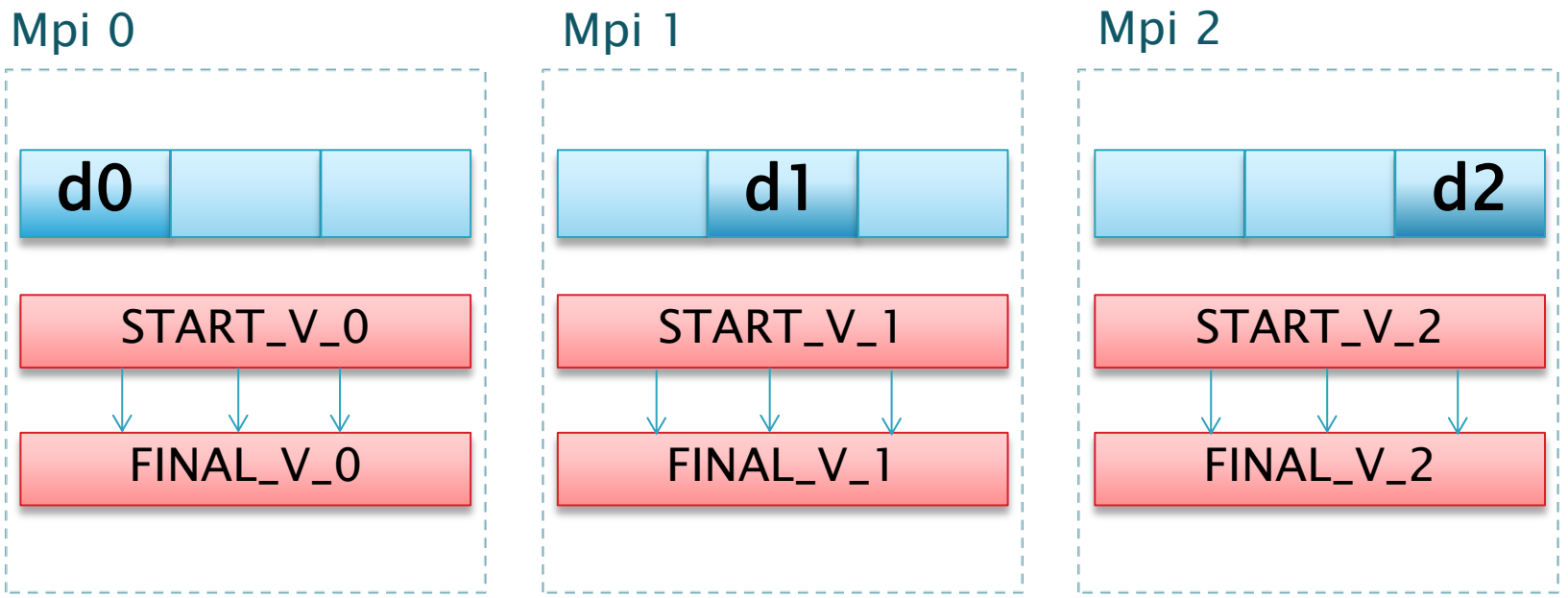
Mapping onto heterogeneous cluster



Mapping onto heterogeneous cluster K100

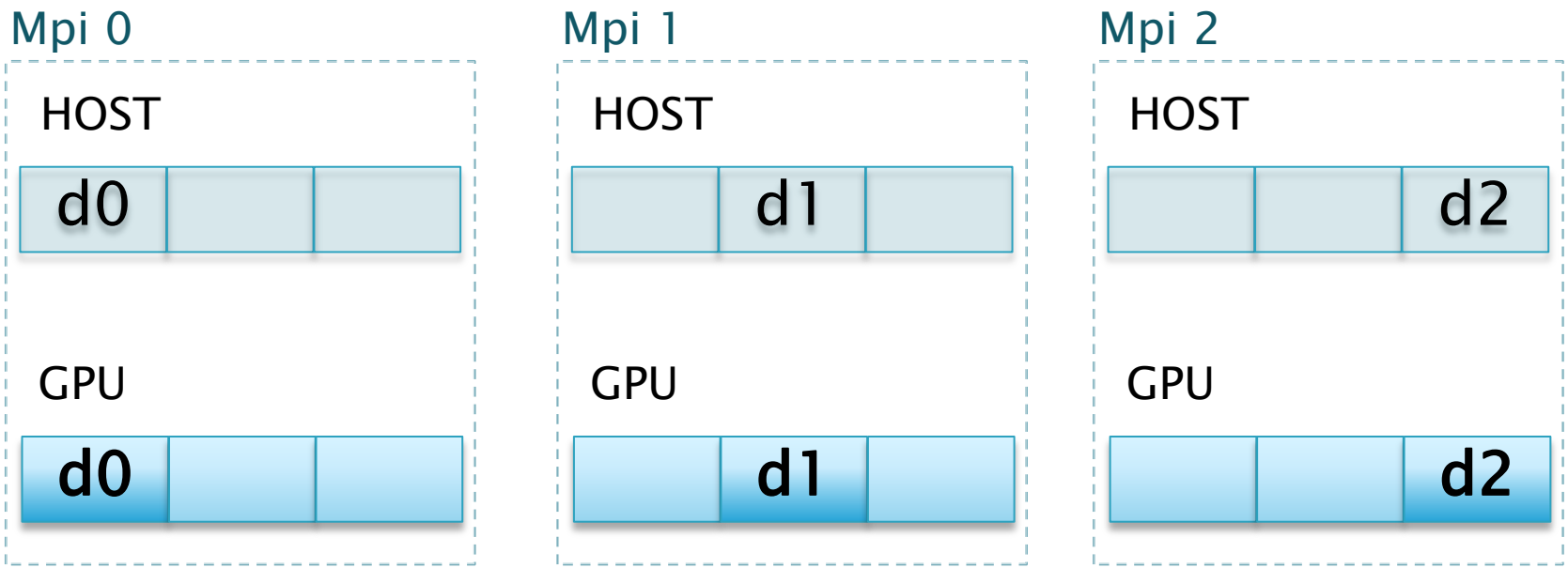


Distribute data between mpi processes



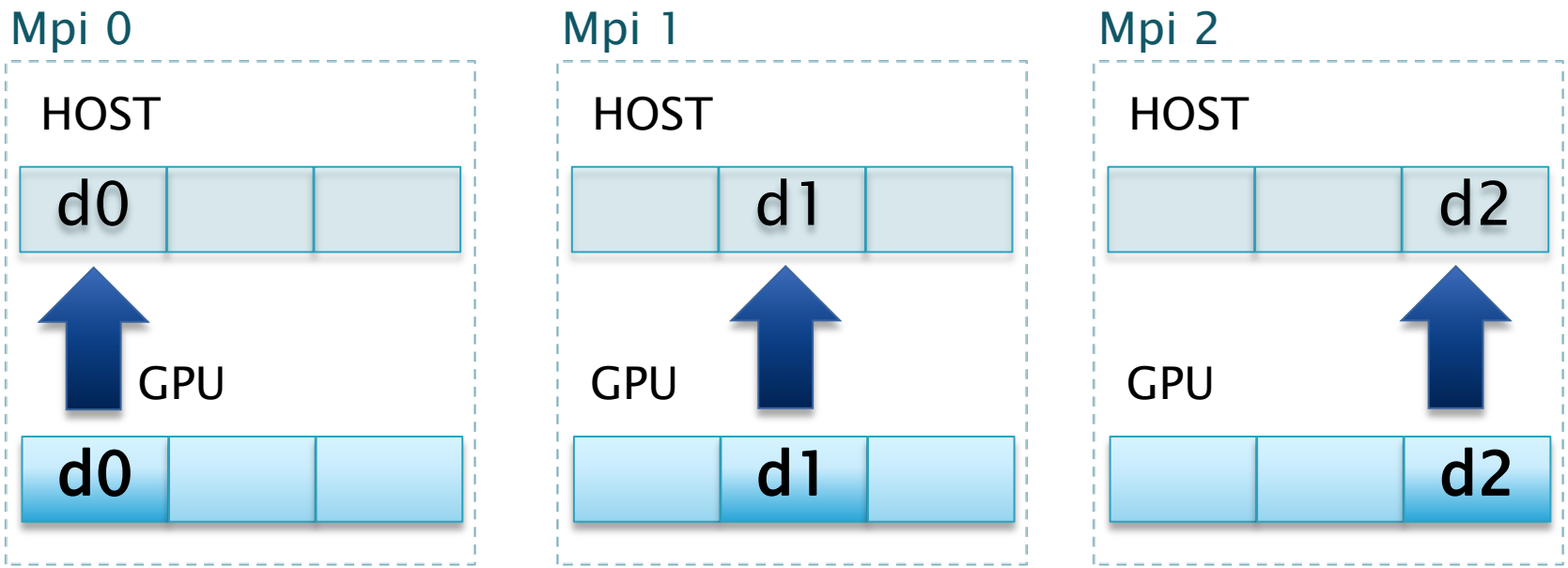
Iteration of algorithm

- Use «top-down» approach for marking the part of distance array in each GPU(d_i);



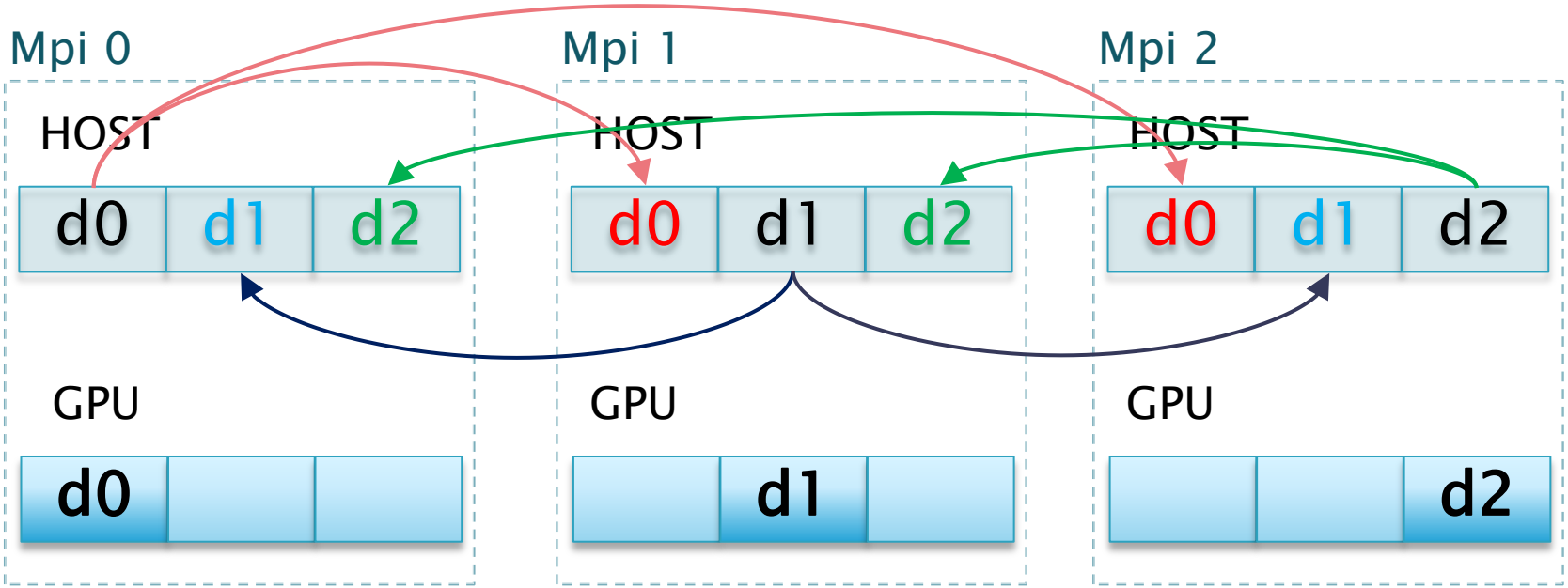
Iteration of algorithm

- Copy the each part of distance array from GPU to CPU;



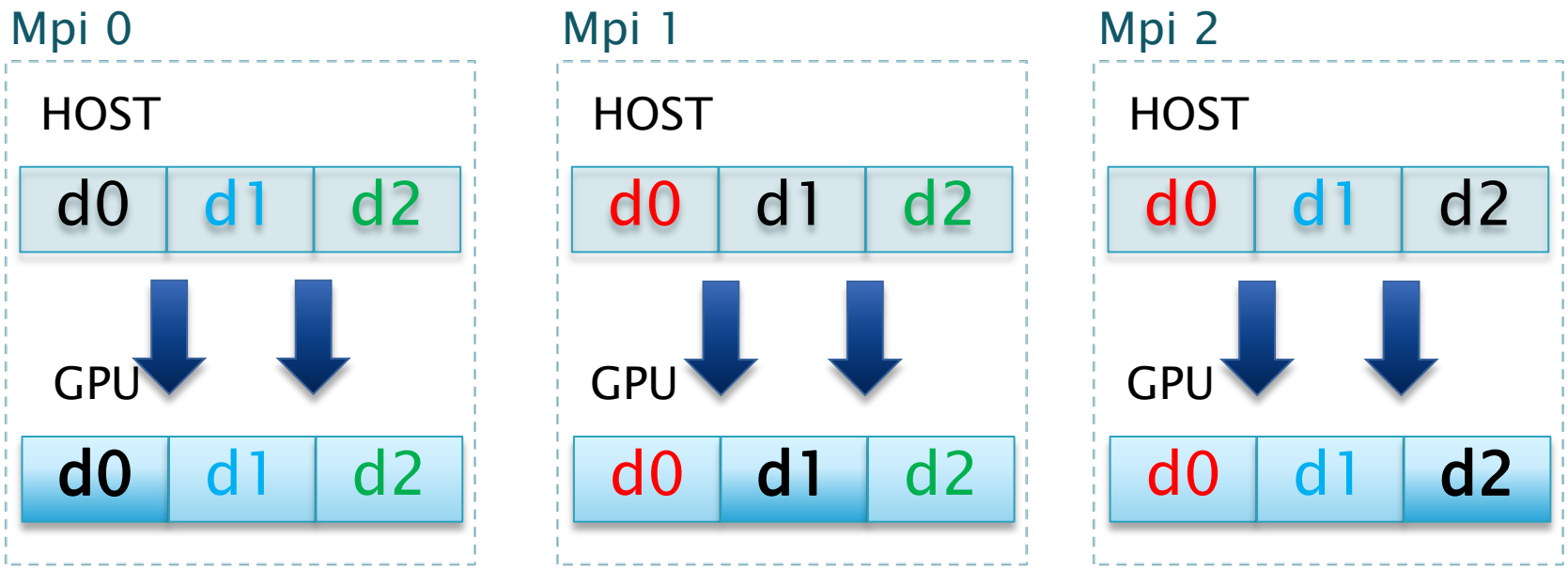
Iteration of algorithm

- Execute the all gather operation between all mpi processes to identify changes from other nodes;

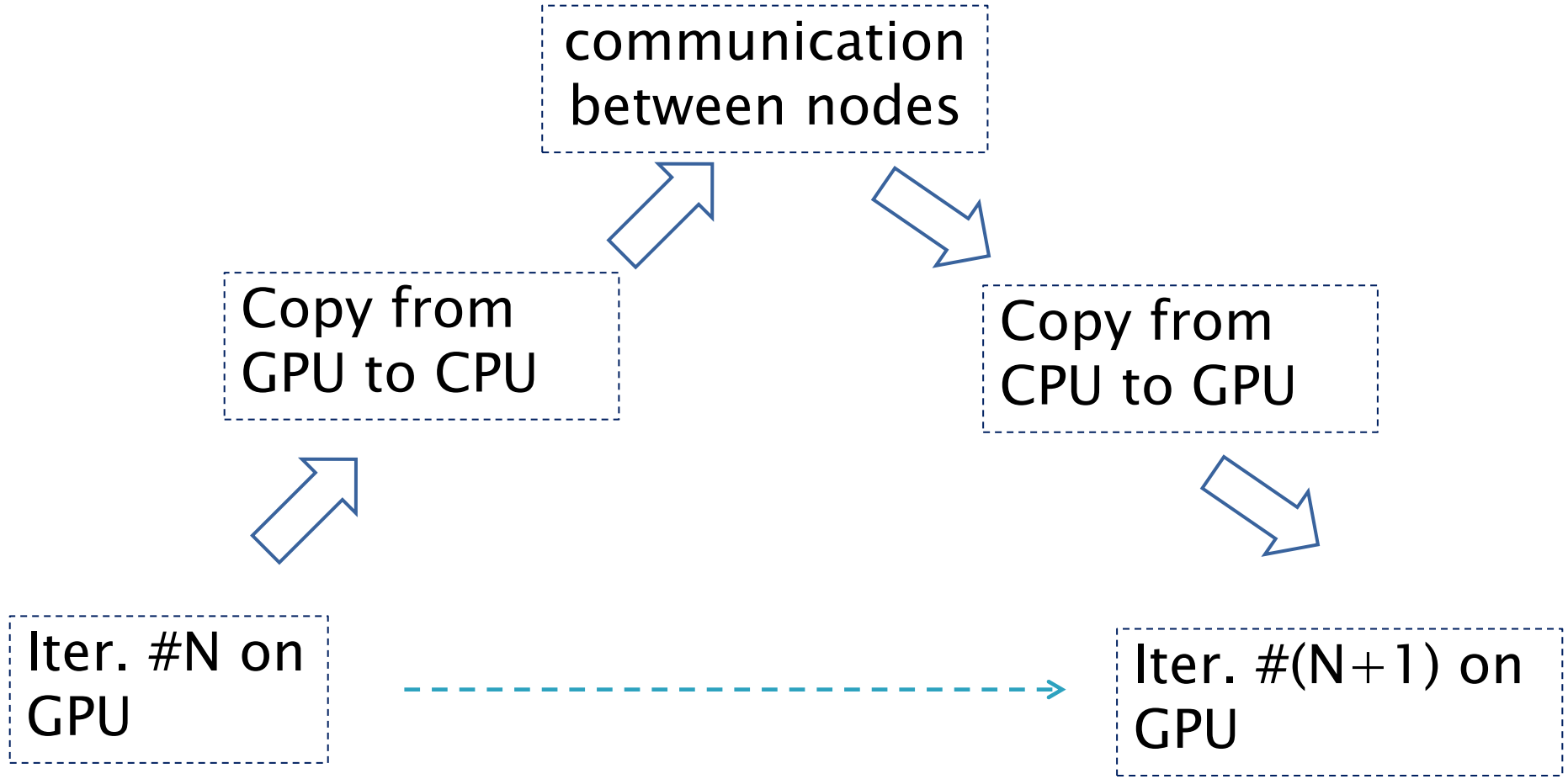


Iteration of algorithm

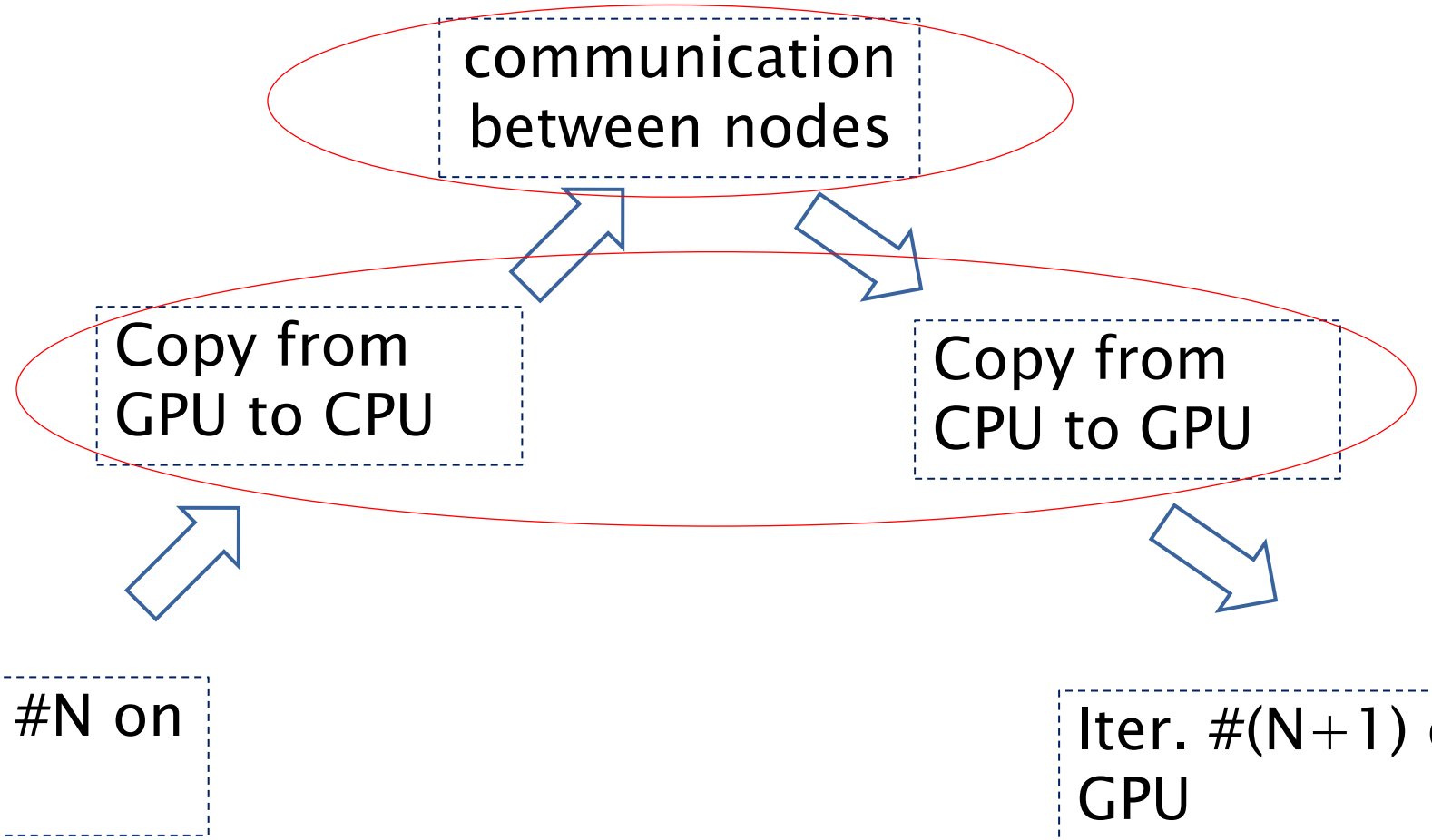
- Copy the given results of distance array from CPU to GPU, go to the next iteration;



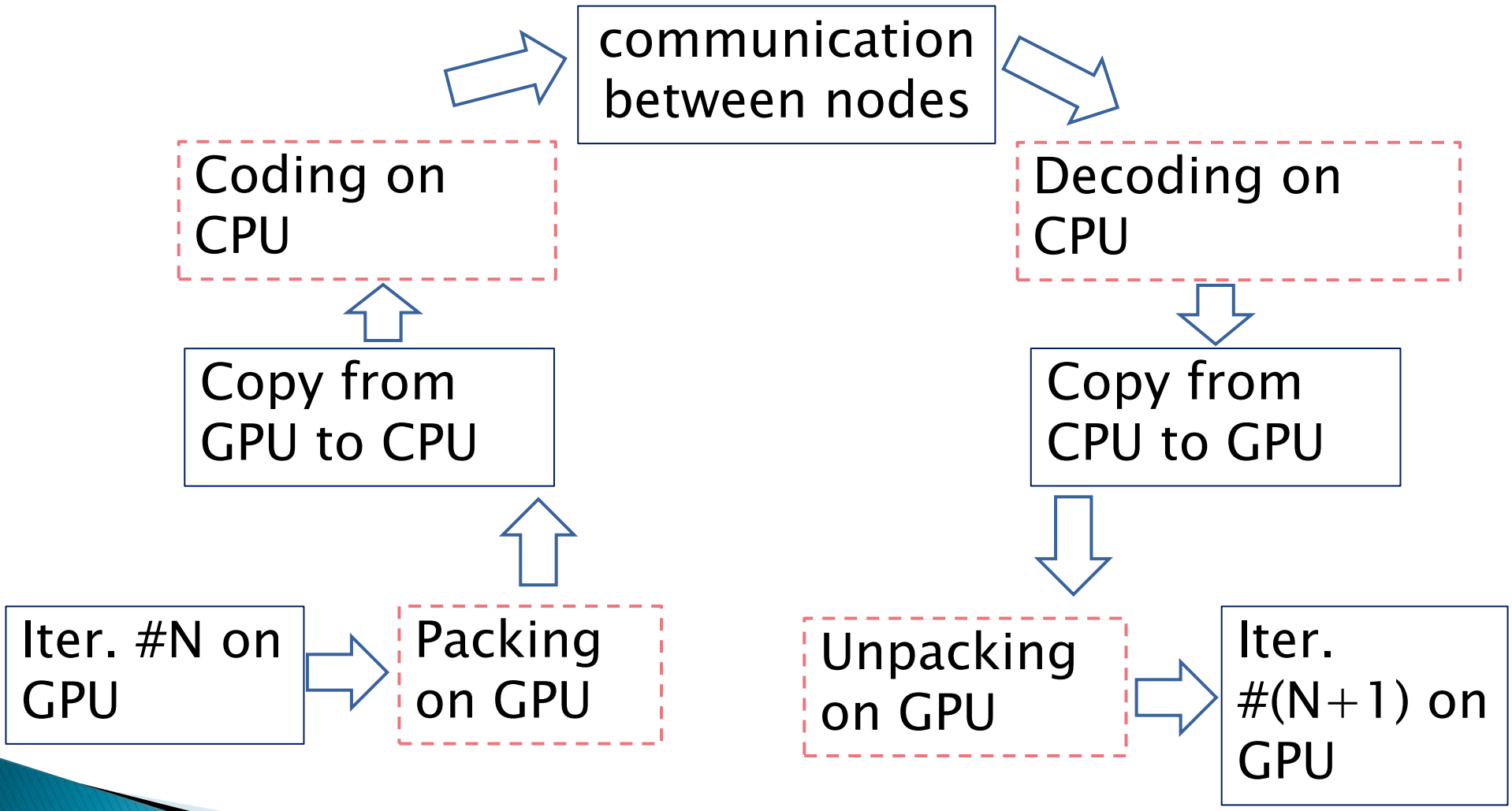
Scheme of one iteration



Overhead


communication between nodes
Copy from GPU to CPU
Copy from CPU to GPU
Iter. #N on GPU
Iter. #(N+1) on GPU

Solution: compression



Packing data on GPU

- It is necessary to send information of the marked vertices only at the current iteration;
- To store this information, one bit is sufficient (instead of 8 bits);
- Allow to compress data 8 times (using the *unsigned char* type).

Coding data on CPU

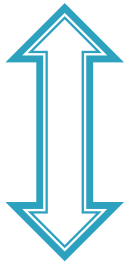
- Use RLE algorithm (*Run-length encoding)*;
- Reduces the amount of data transmitted in the initial and final iterations;
- Allow to restore all the compressed data.

Example of RLE

- Code with RLE (*Run-length encoding)*;

The original array of bits:

|00000|1|0000000000000000|1.....



Compressed array:

|101|0|1|1|1000|0|1|1|.....

Example of RLE

- Code with RLE (*Run-length encoding)*;

|00000|1|0000000000000000|1.....

|101|0|1|1|1000|0|1|1|.....

Example of RLE

- Code with RLE (*Run-length encoding)*;

|00000|1|0000000000000000|1.....

|101|0|1|1|1000|0|1|1|.....

Example of RLE

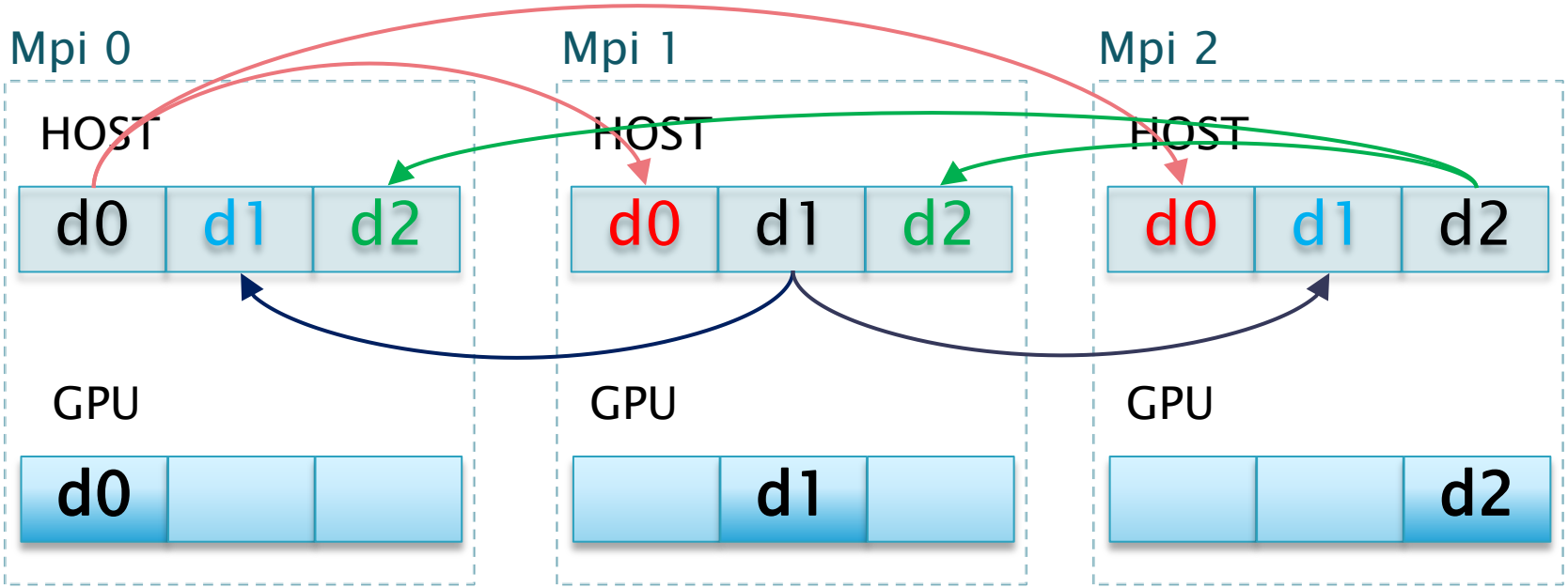
- Code with RLE (*Run-length encoding)*;

|00000|1|0000000000000000|1.....

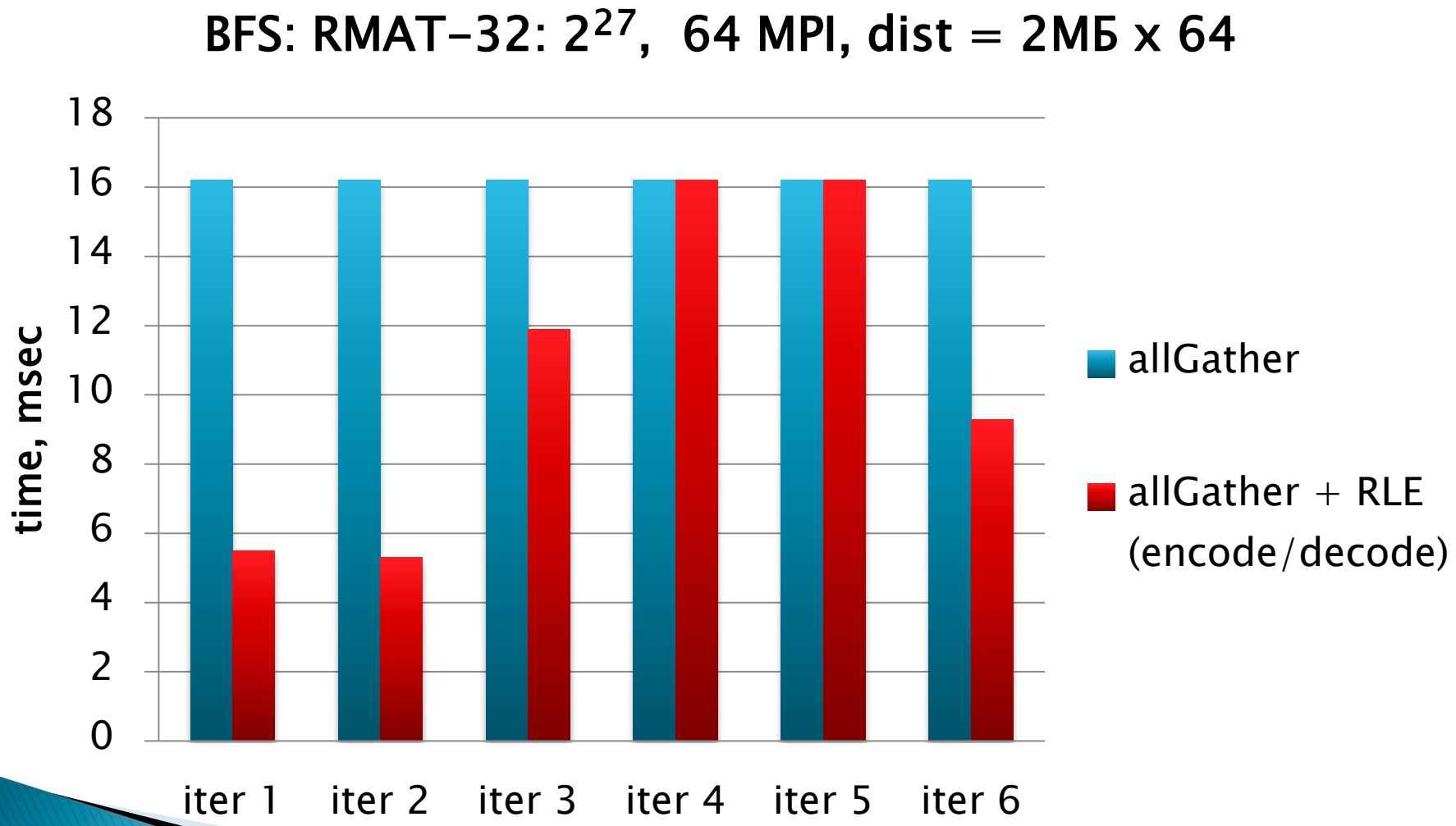
|101|0|1|1|1000|0|1|1|.....

RLE encoding on CPU

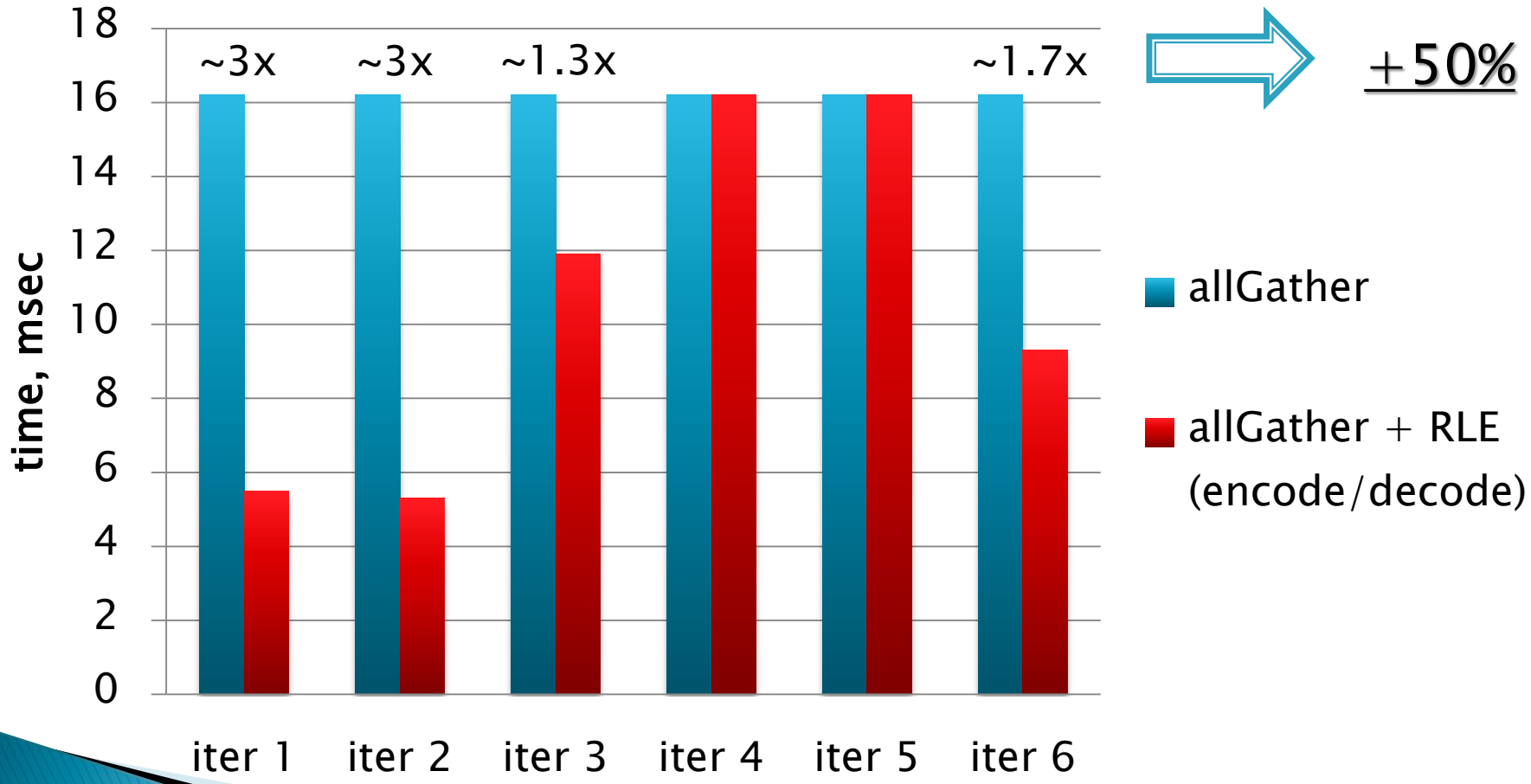
- Code each part of dist array with one thread;
- Decode received parts with many threads;



Results of using RLE algorithm



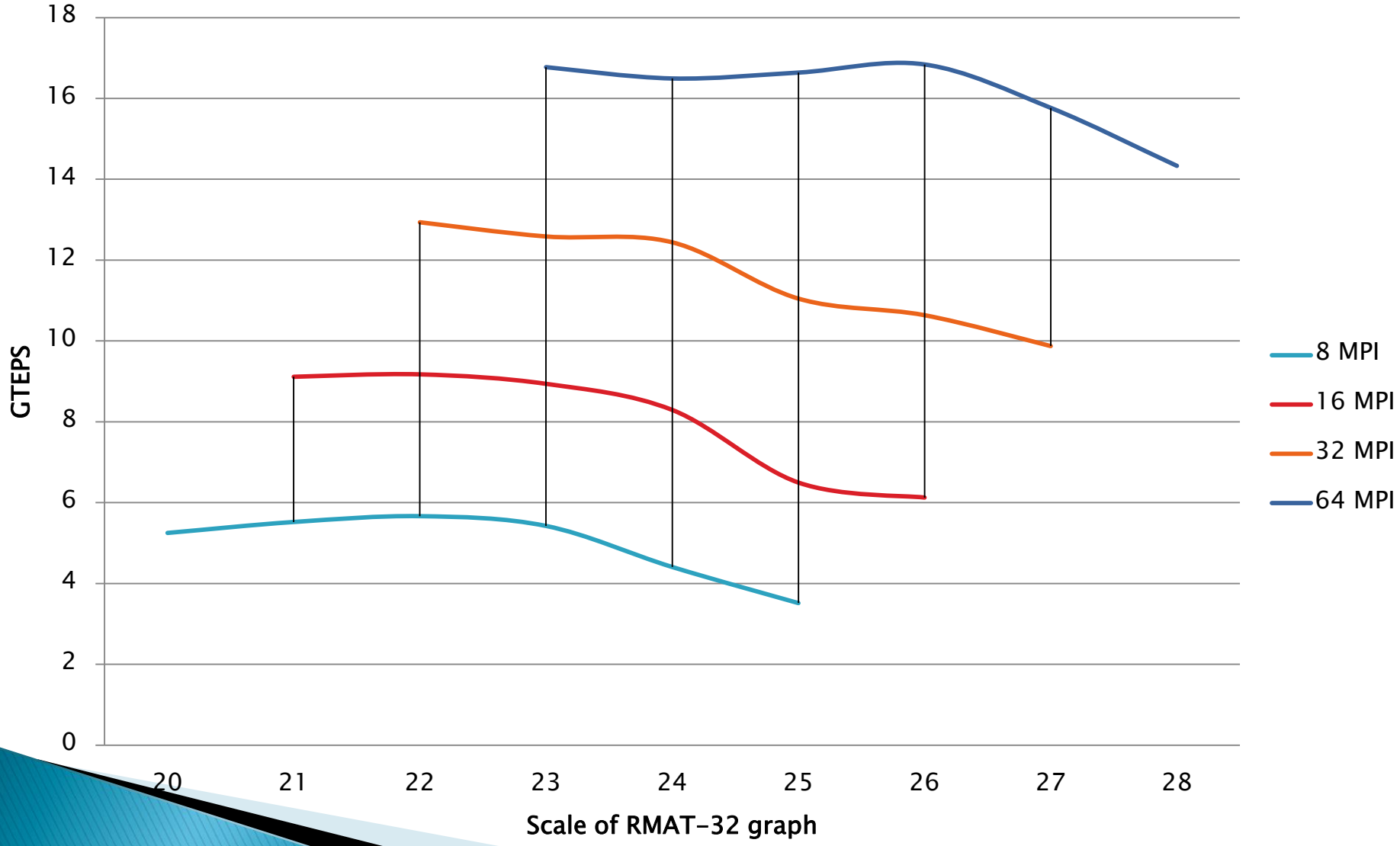
Results of using RLE algorithm: reduce communications



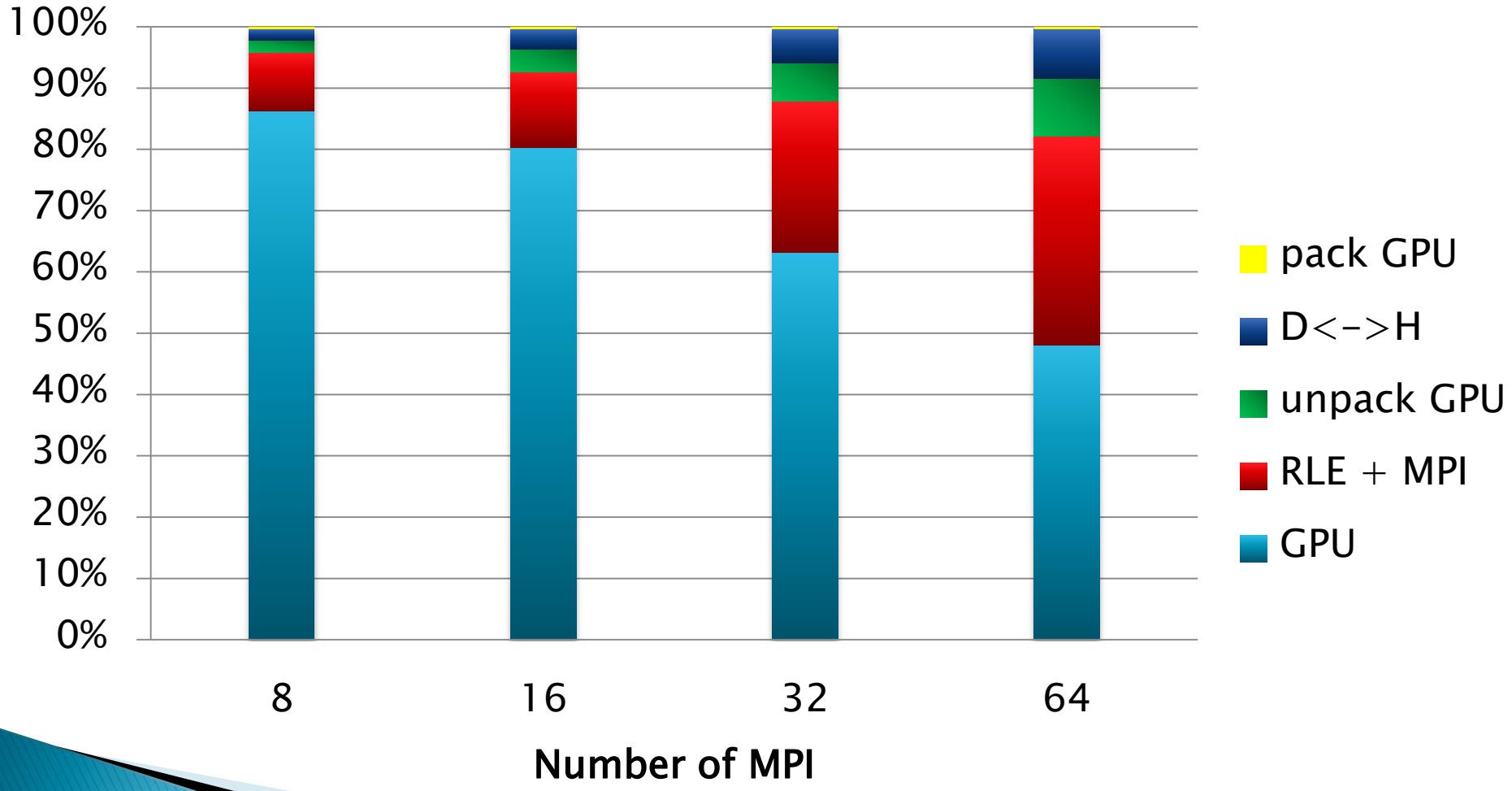
Summary of testing

- Cluster k100:
  node: 2x Xeon 5670 + 3 GPU Tesla c2050;
- RMAT-32 graphs;
- Scales 20-28 (392 MB – 100 352 MB);
- Use up to 64 MPI processes.

Performance

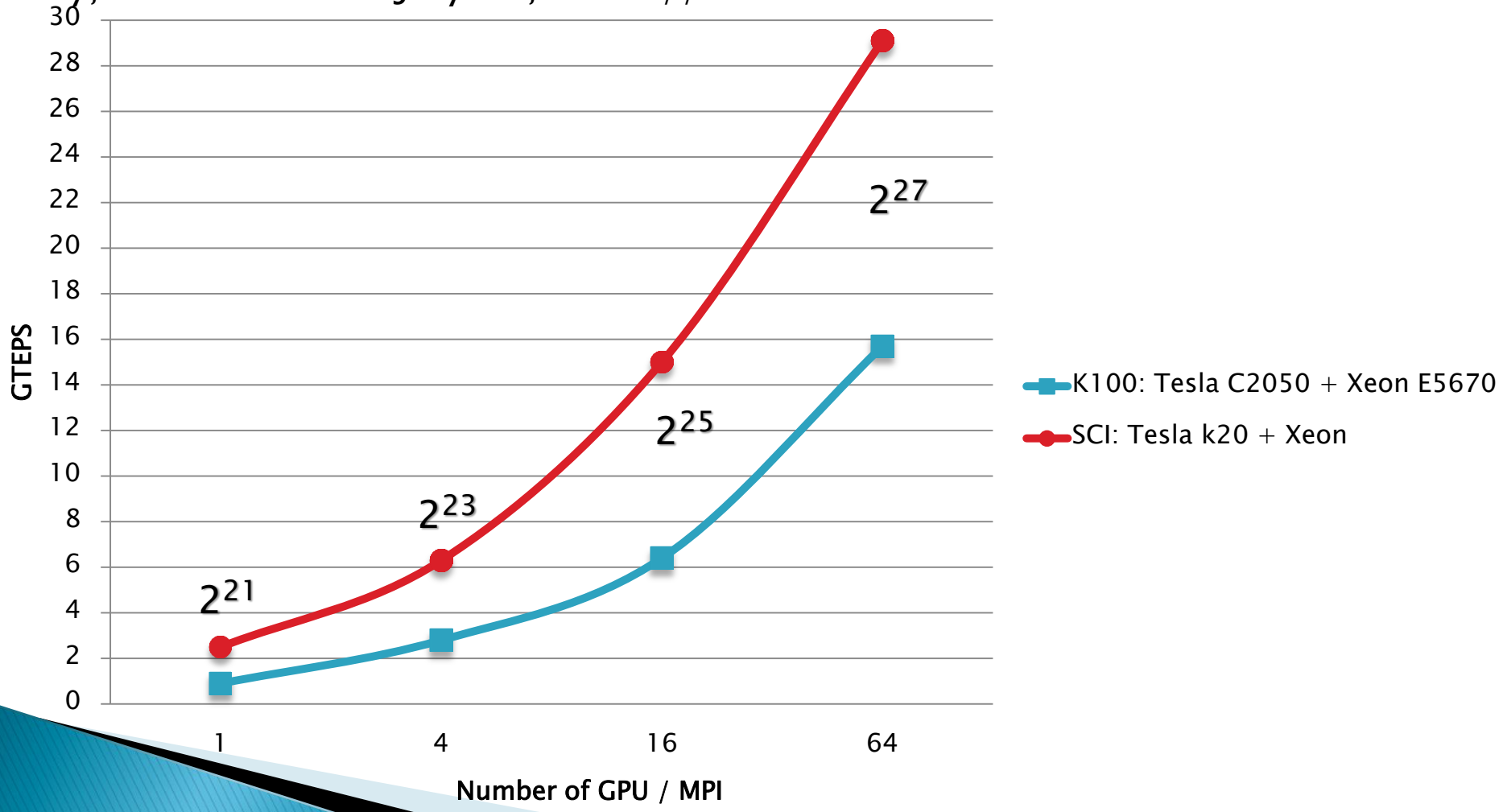


BFS: RMAT-32 2^{25}



Comparison results

“Parallel Breadth First Search on GPU Clusters”, Zhisong Fu, Harish Kumar Dasari, Martin Berzins and Bryan Thompson
// Scientific Computing and Imaging Institute University of Utah Salt Lake City, UT 84112 USA July 29, 2014 //



Thank you for your attention!

Kolganov A.S.: alexander.k.s@mail.ru