Elephants Can Split Graphs, or Very Large Graph Partitioning via PargreSQL

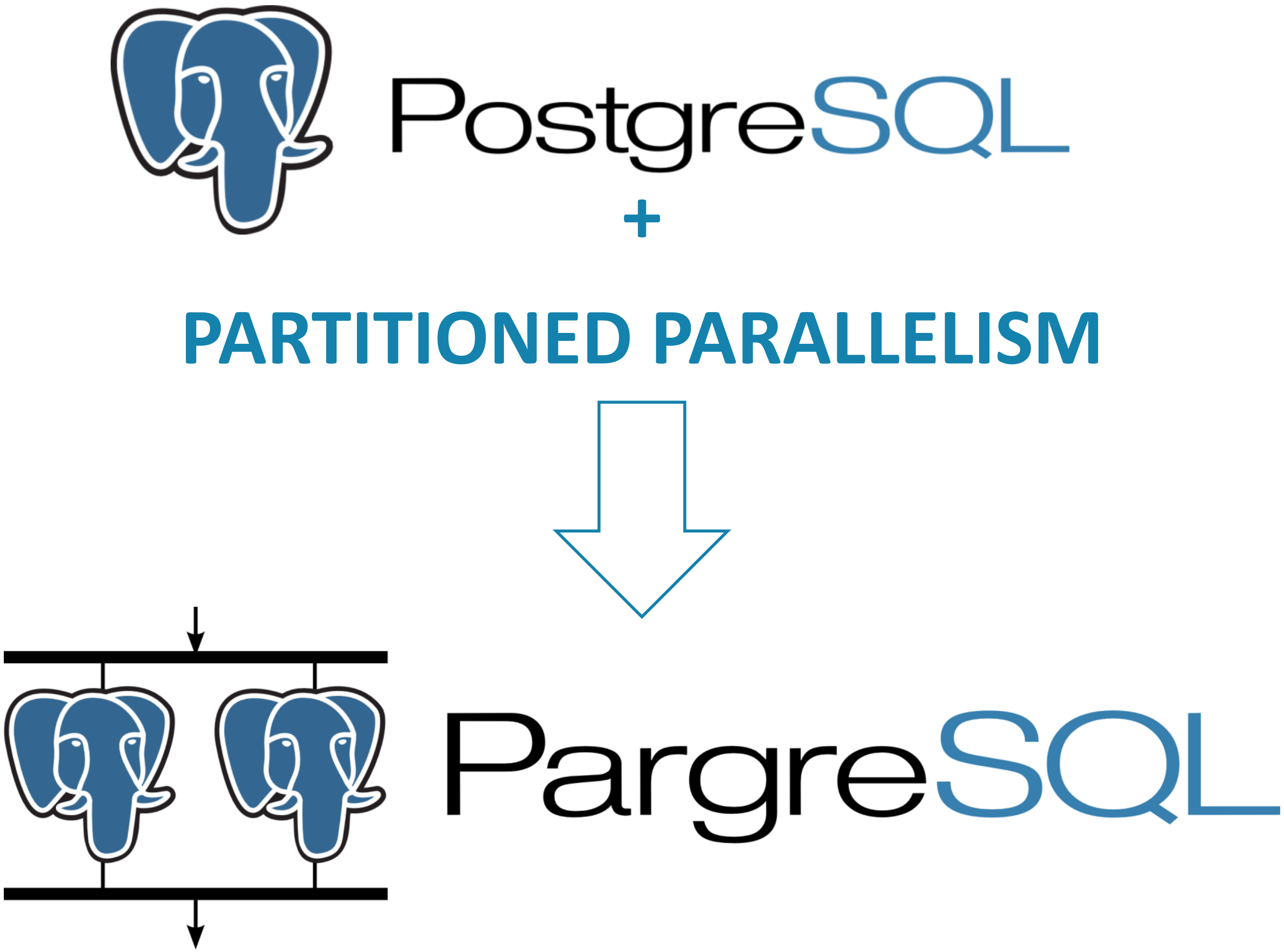
Constantin Pan and Mikhail Zymbler,

South Ural State University,
Chelyabinsk, Russia

Outline

- ParqreSQL DBMS in brief
- Graph partitioning via PargreSQL
- Experimental results

PargreSQL project



PostgreSQL

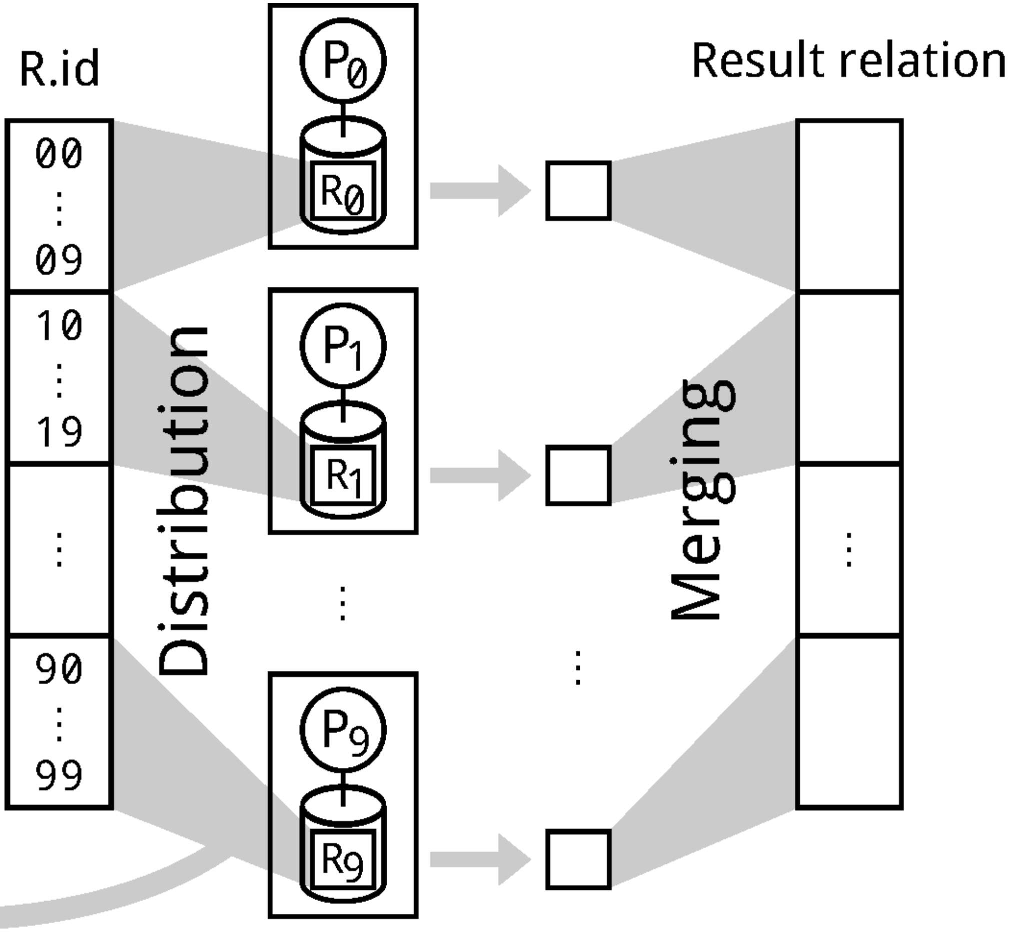
+

PARTITIONED PARALLELISM

PargreSQL

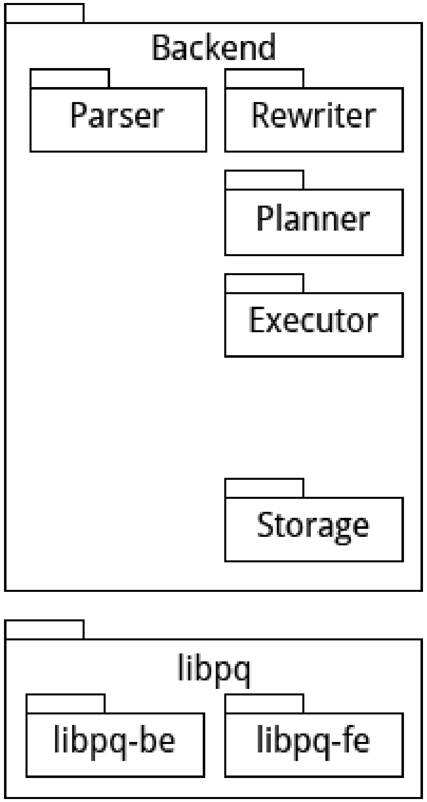
Partitioned parallelism

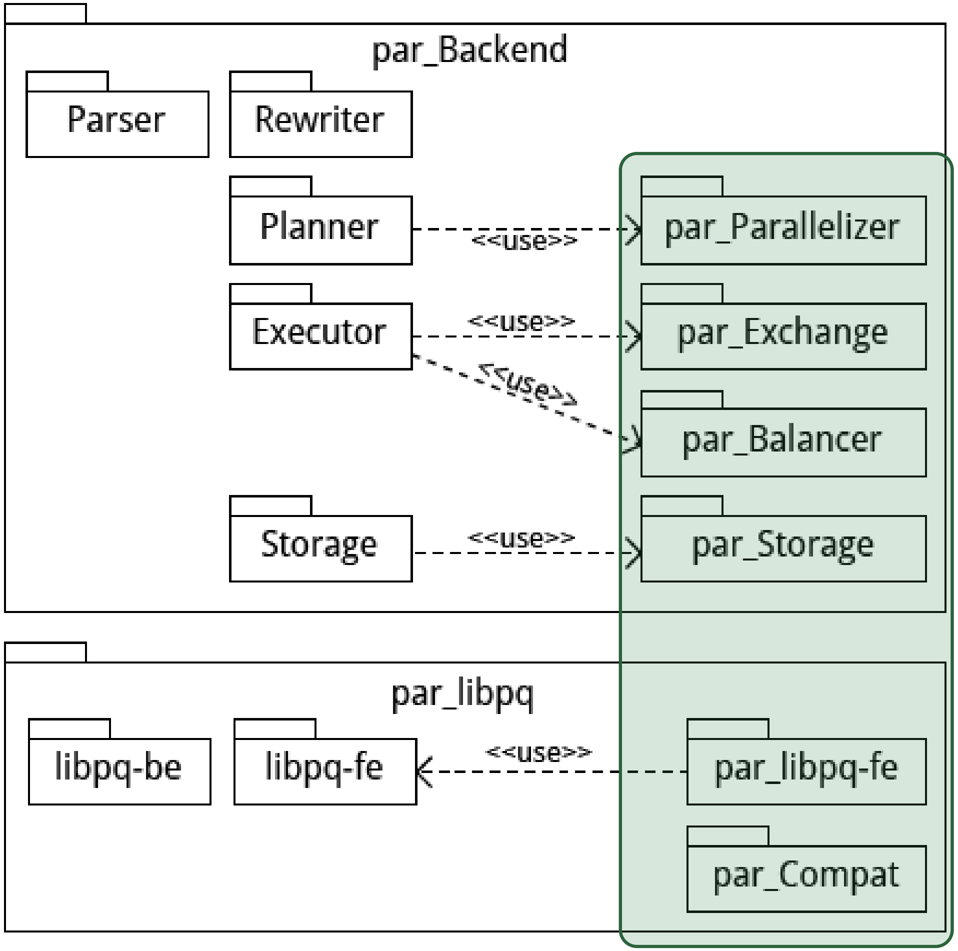
$R_i = \{t \mid t \in R, \phi(t) = i\}$
$i = 0, \dots, 9$



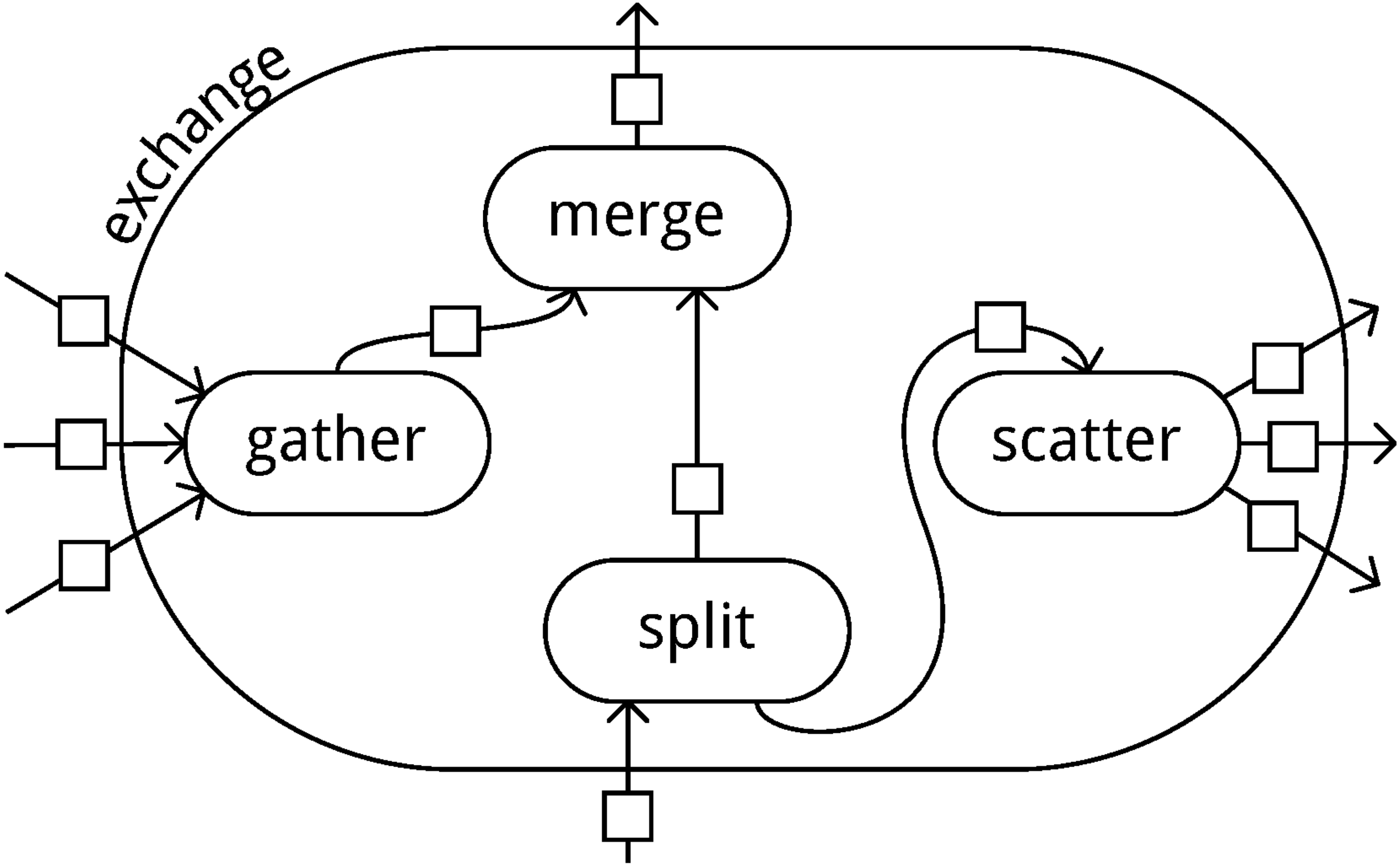
Fragmentation function
$\phi(t) = (t.id \text{ div } 10) \bmod 10$

PostgreSQL vs PargreSQL



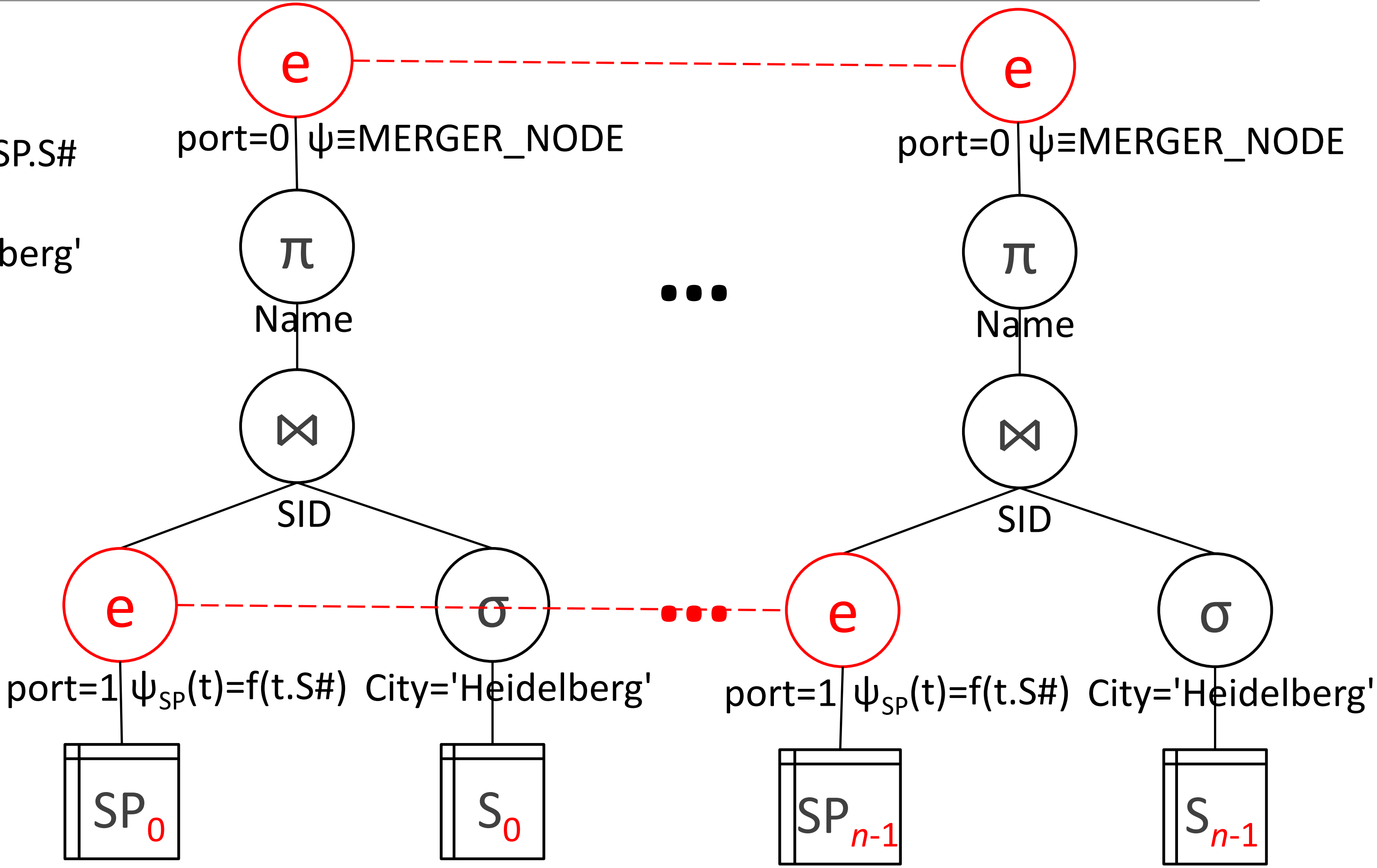


EXCHANGE operator

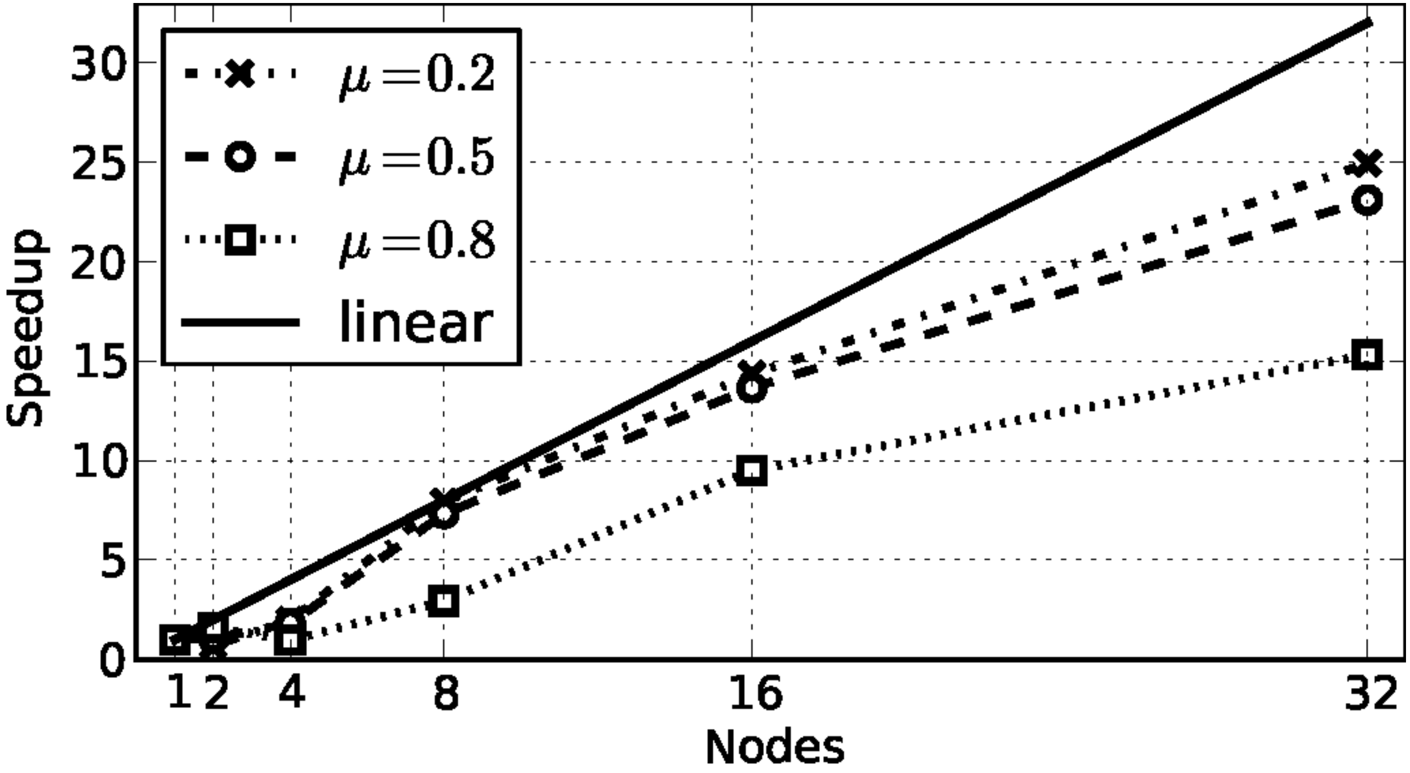


EXCHANGE operator

SELECT Name
FROM S, SP
WHERE S.S#=SP.S#
and
S.City='Heidelberg'



PargreSQL speedup



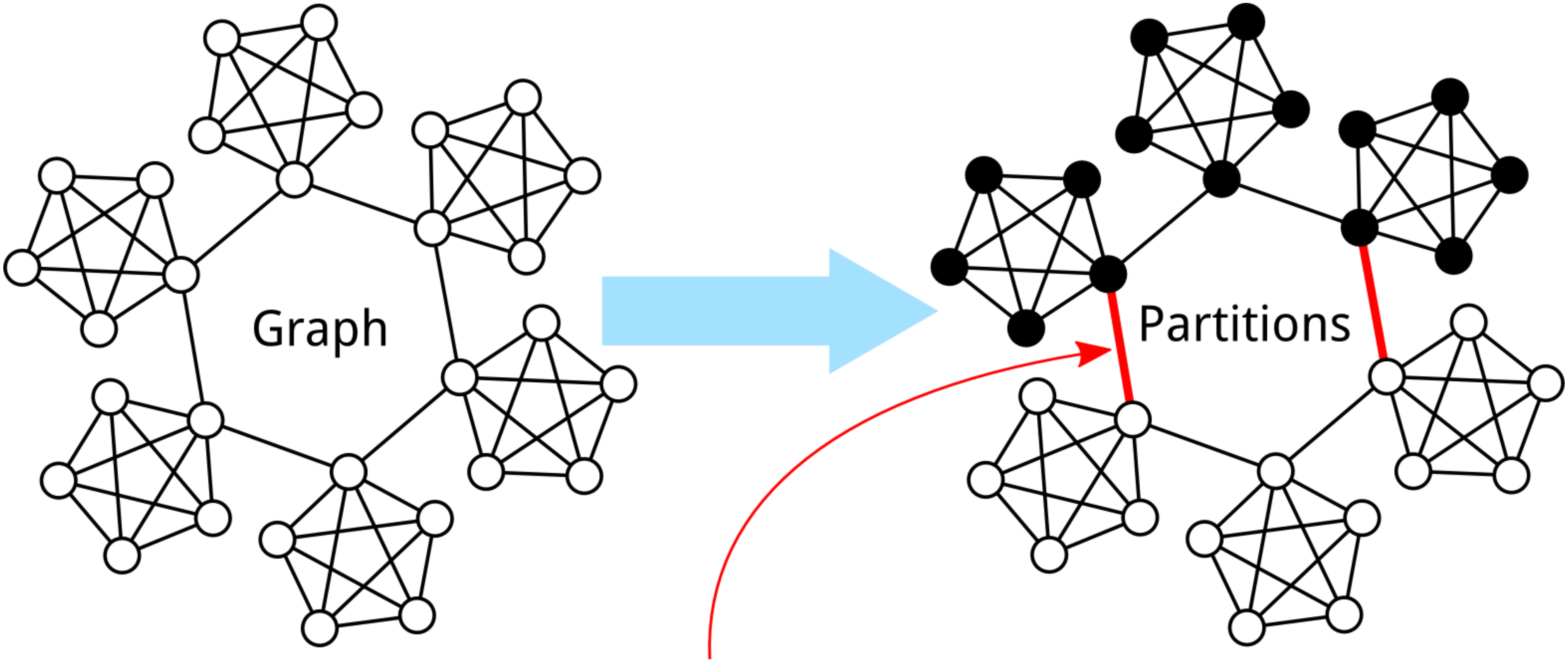
R* ⋈ *S

$|R| = 6 \cdot 10^7$

$|S| = 1.5 \cdot 10^6$

μ is a portion of tuples at every partition of the table to be sent to other nodes

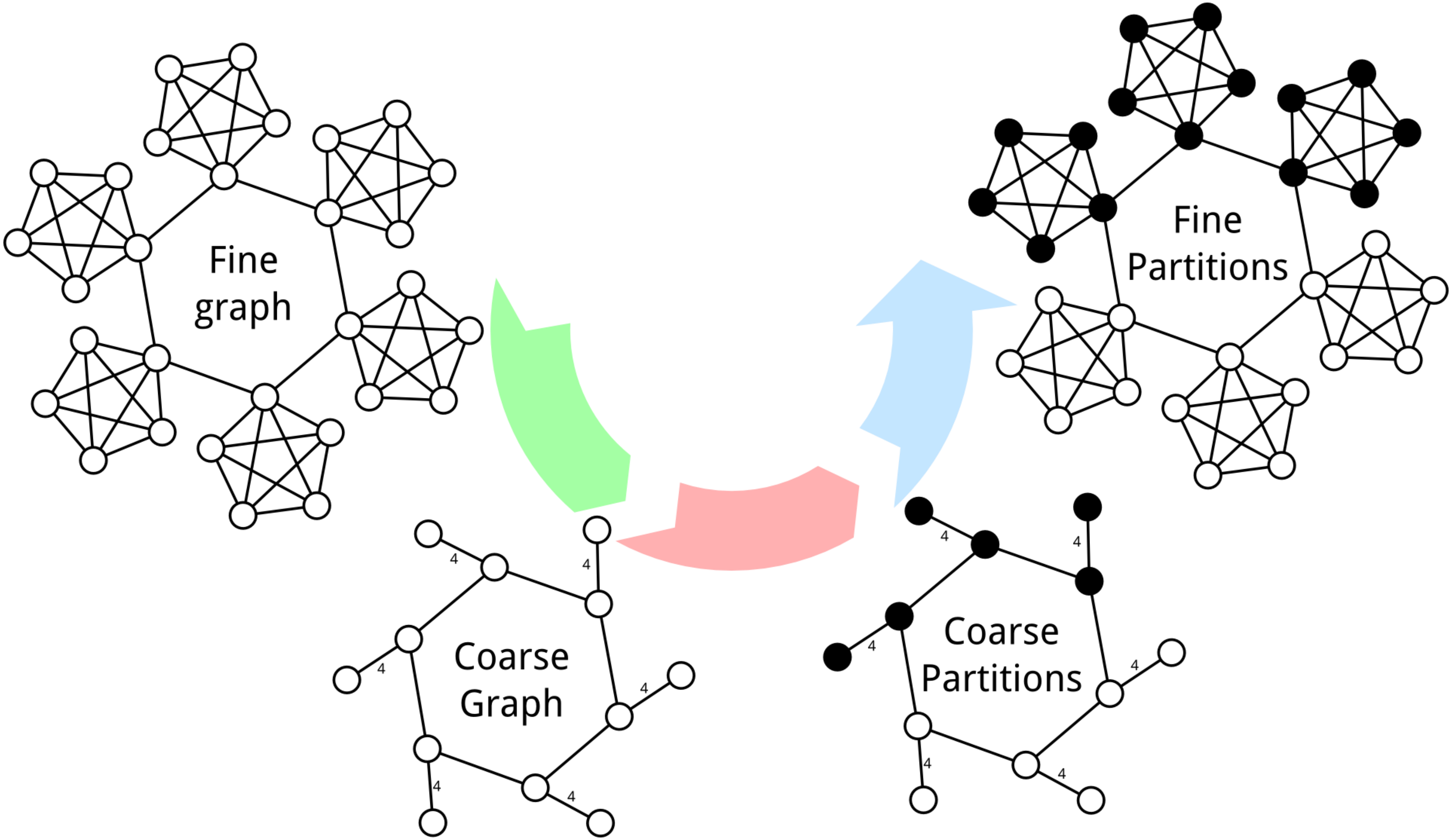
Graph partitioning



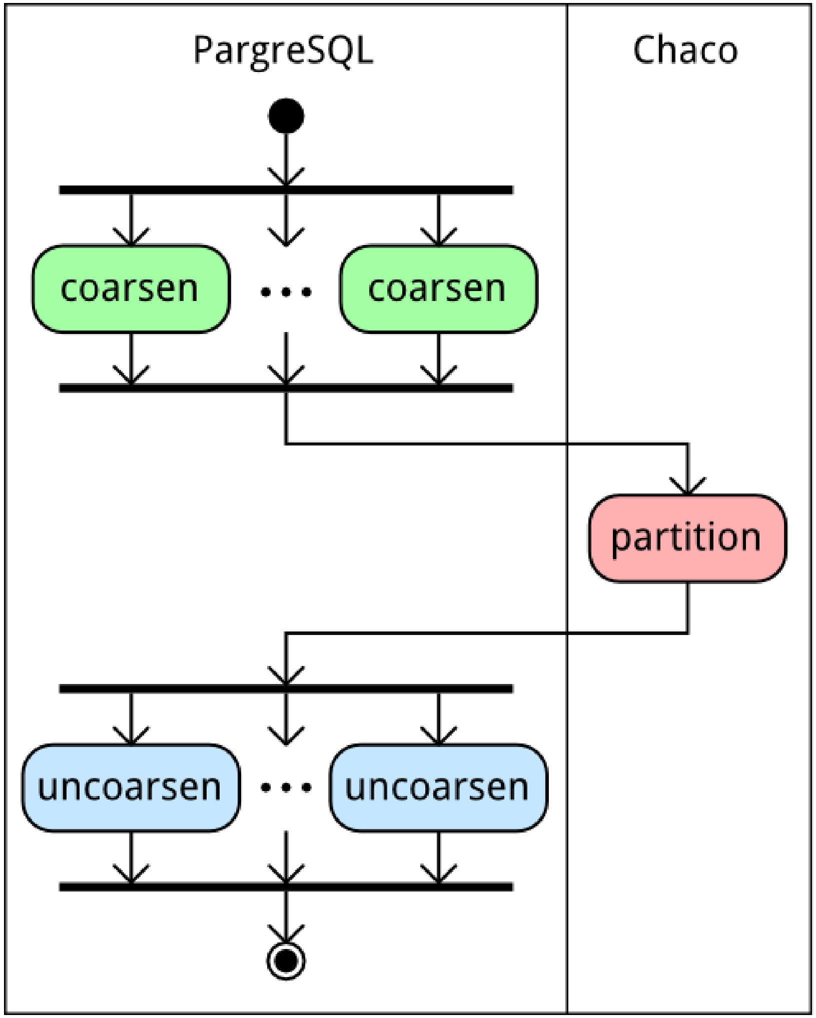
cut size → min

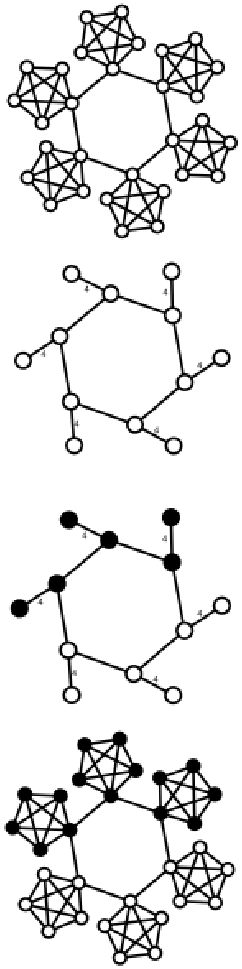
○ partition size ≈ ● partition size

Multilevel partitioning

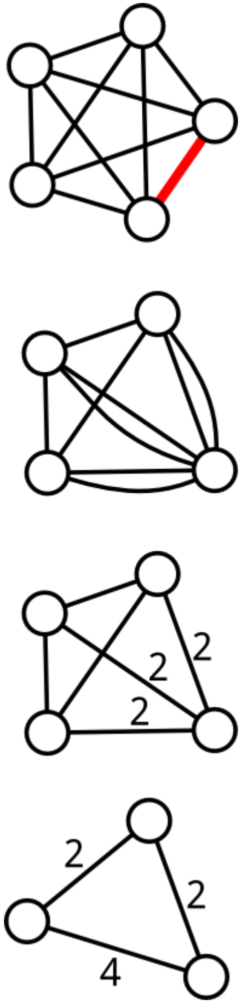


Using PargreSQL



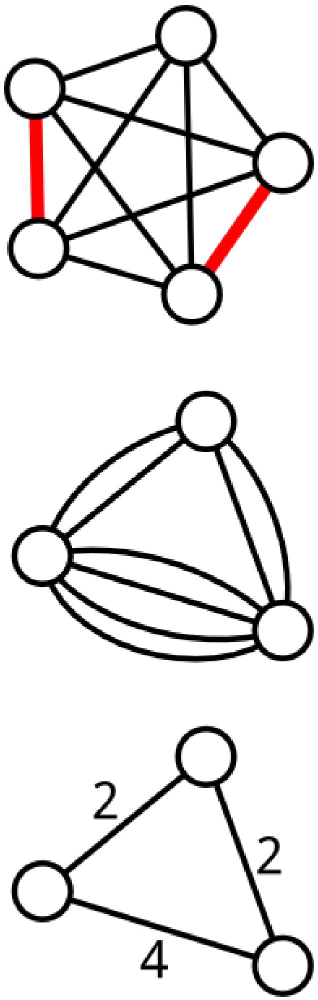


Coarsening in memory



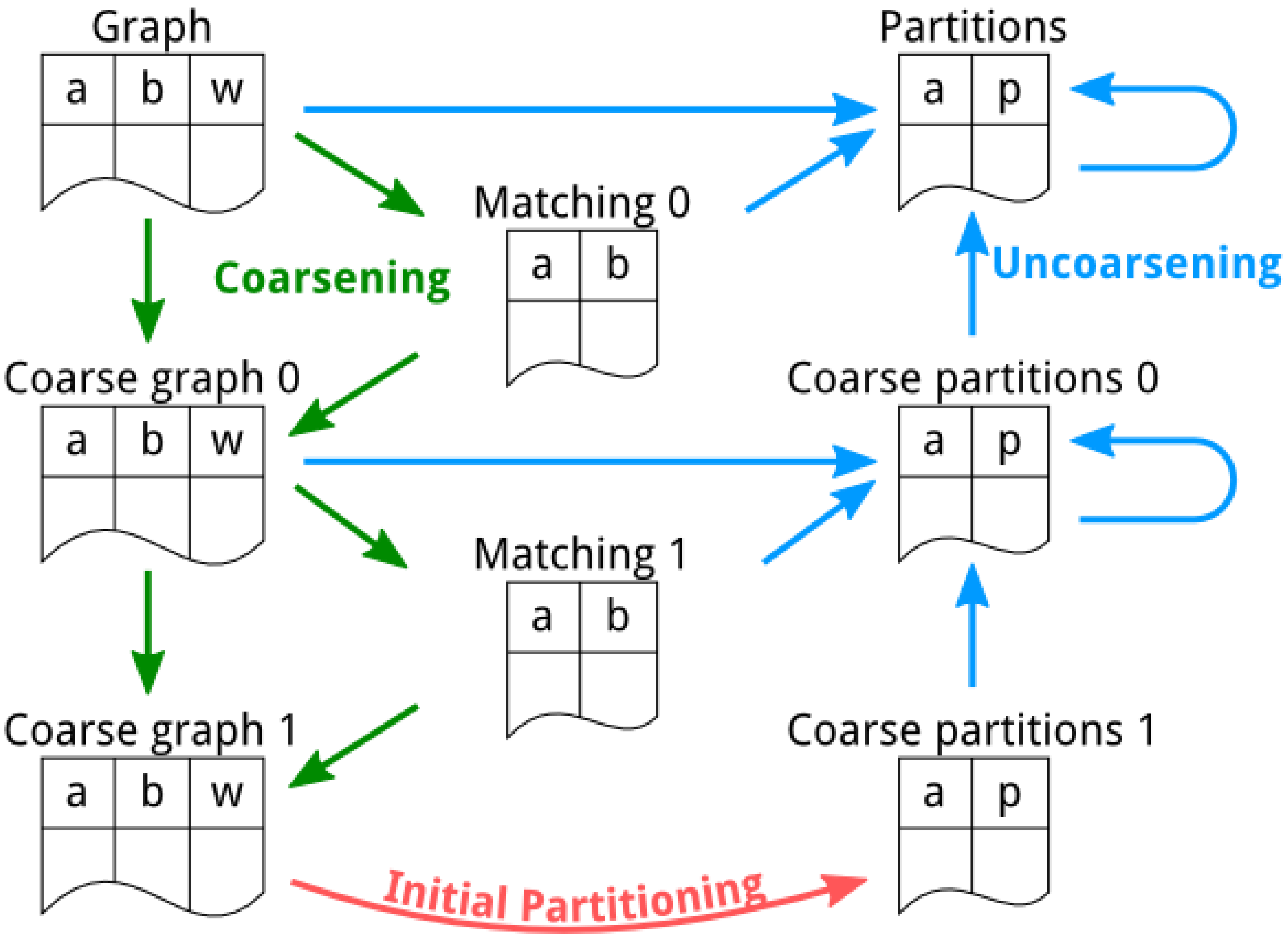
1. Find the heaviest (or a random) edge.
2. Collapse the edge into a vertex.
3. Merge the duplicates and remove the loops.
4. Repeat, avoiding the vertices generated this way, until nothing is left.

Coarsening with PargreSQL

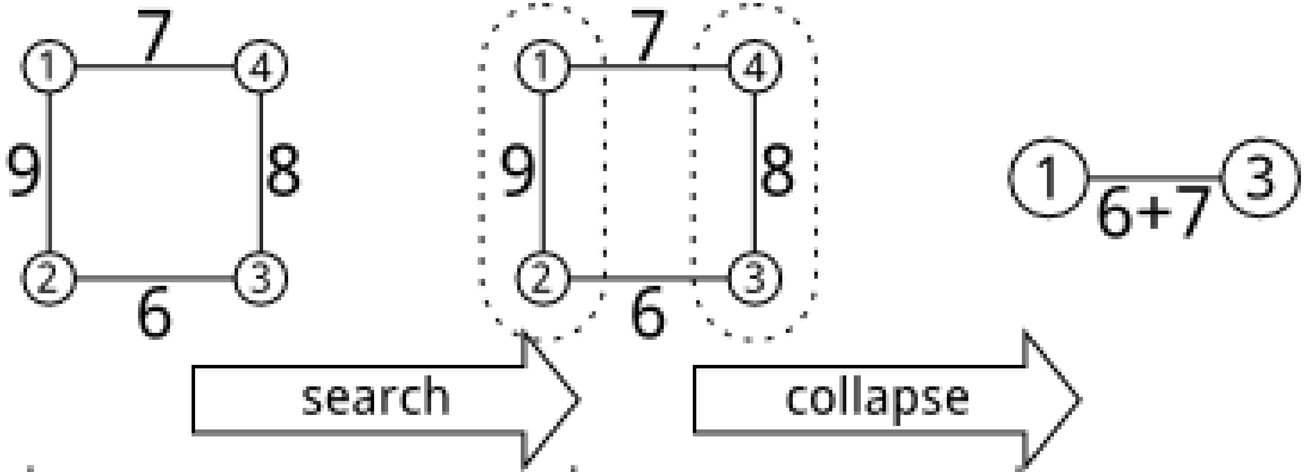


1. Find the heaviest matching.
2. Collapse the edges of the matching into vertices.
3. Merge the duplicates and remove the loops.

Data flow


Graph
a
b
w
Partitions
a
p
Matching 0
a
b
Coarsening
Uncoarsening
Coarse graph 0
a
b
w
Coarse partitions 0
a
p
Matching 1
a
b
Coarse graph 1
a
b
w
Coarse partitions 1
a
p
Initial Partitioning

Coarsening implementation



GRAPH

A	B	W
1	2	9
2	3	6
3	4	8
4	1	7

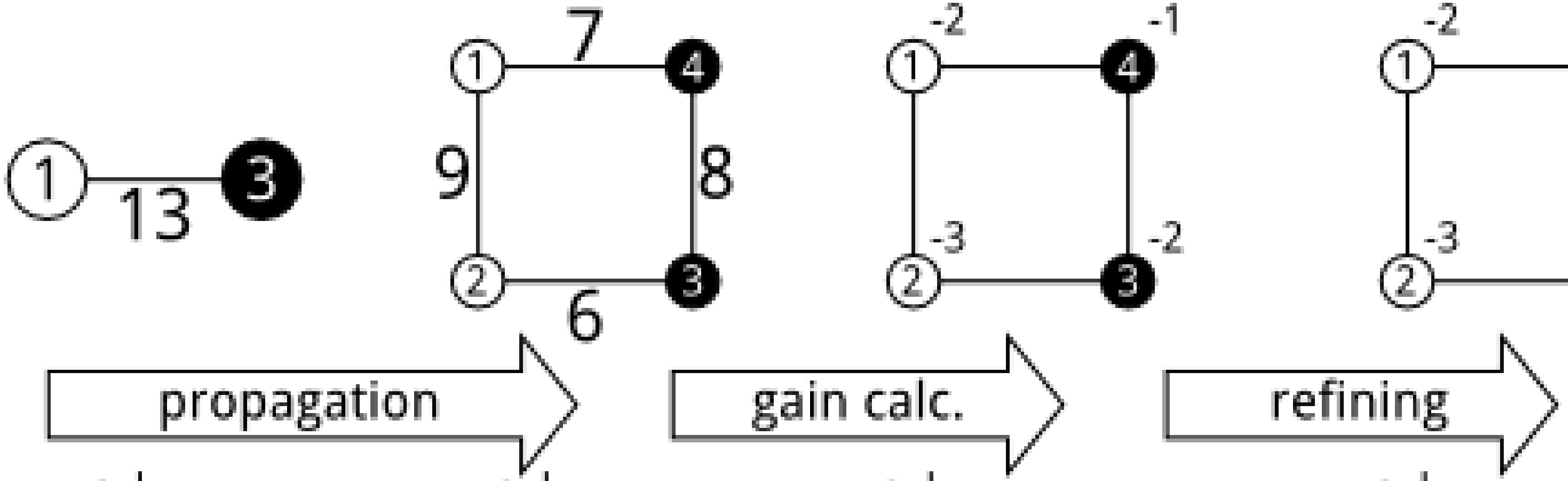
MATCH

A	B
1	2
3	4

COARSE GRAPH

A	B	W
1	3	13

Uncoarsening implementation



COARSE PARTITIONS

A	P
1	1
3	0

PARTITIONS

A	P
1	1
2	1
3	0
4	0

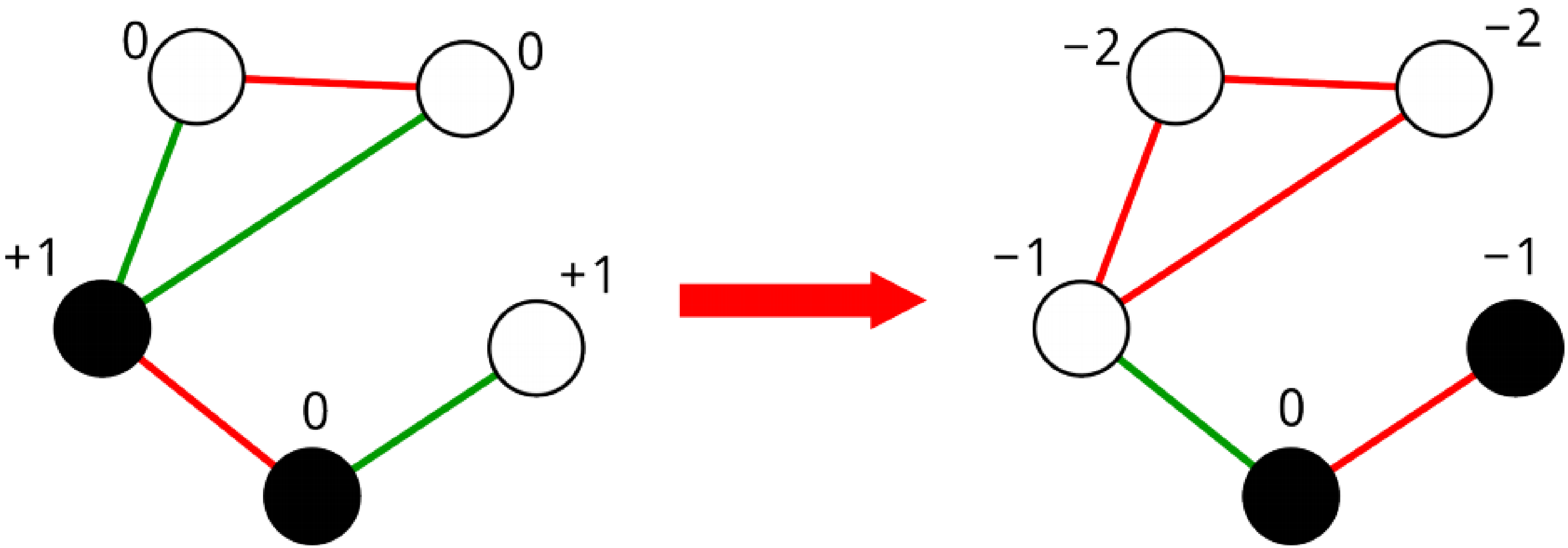
PARTITIONS

A	P	G
1	1	-2
2	1	-3
3	0	-2
4	0	-1

PARTITIONS

A	P	G
1	1	-2
2	1	-3
3	0	-2
4	0	-1

Partitioning quality (by Kernighan and Lin)

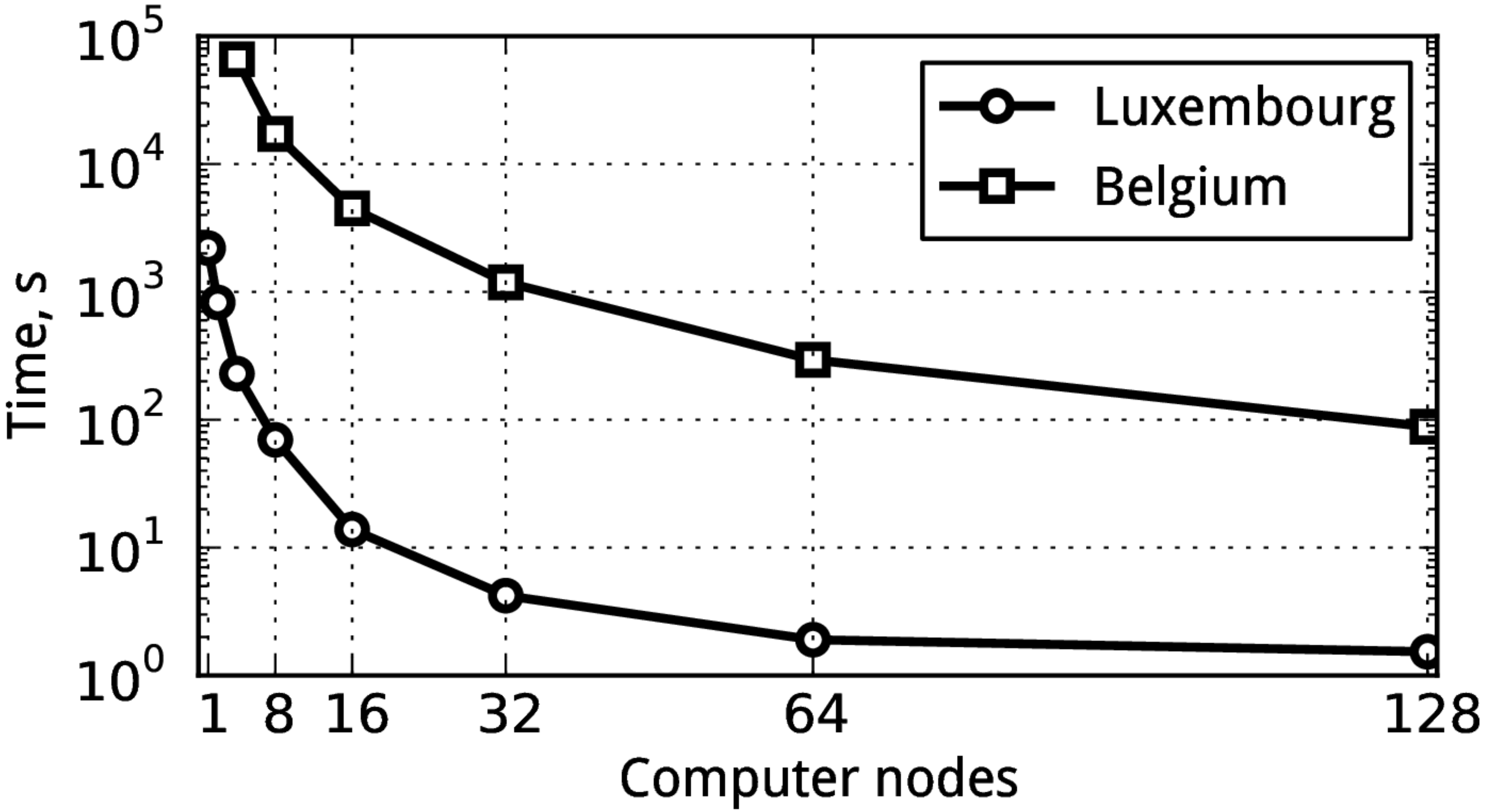


$$\mathrm{gain}(v) = \sum_{(v,u)\in E, P(v)\neq P(u)} w(v,u) - \sum_{(v,u)\in E, P(v)=P(u)} w(v,u)$$

Experiments

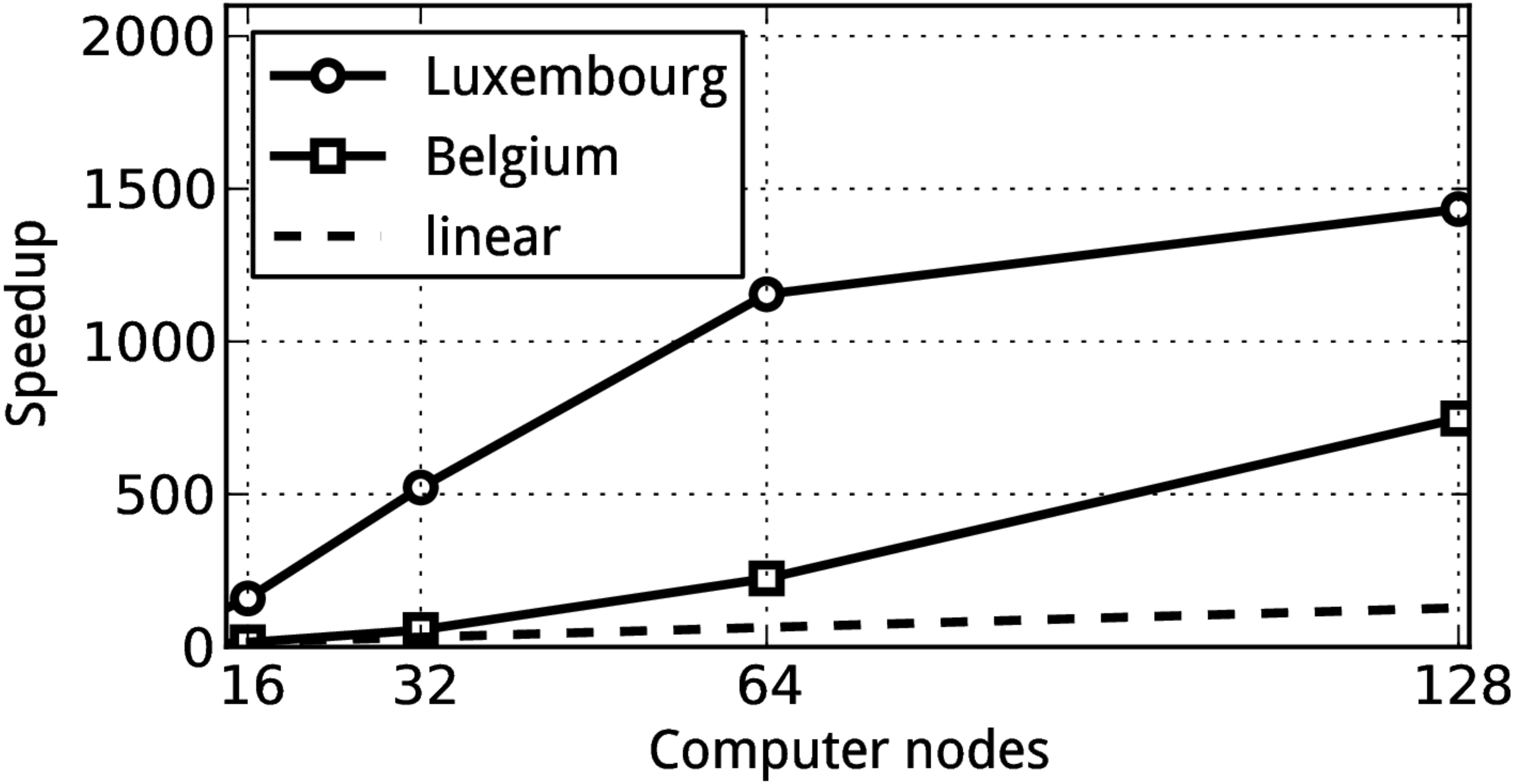
- Computer
  - 128 nodes of Tornado cluster in South Ural State University (471st in top500)
- Data
  - Luxembourg road map from OpenStreetMap (10^5 vertices, 1 iteration)
  - Belgium road map from OpenStreetMap (10^6 vertices, 5 iterations)
  - distributed over the cluster nodes by function $\phi(e) = e.A * |V|/|E|$

Time



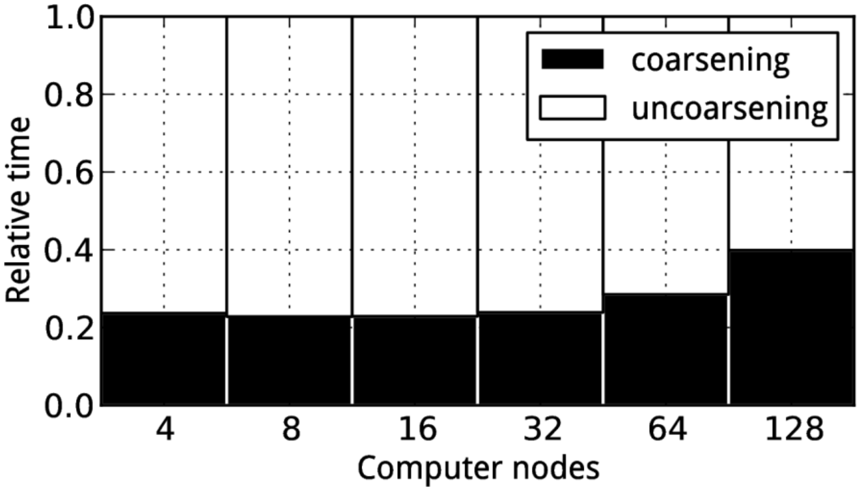
Luxembourg
Belgium
Time, s
10^5
10^4
10^3
10^2
10^1
10^0
1
8
16
32
64
128
Computer nodes

Speedup

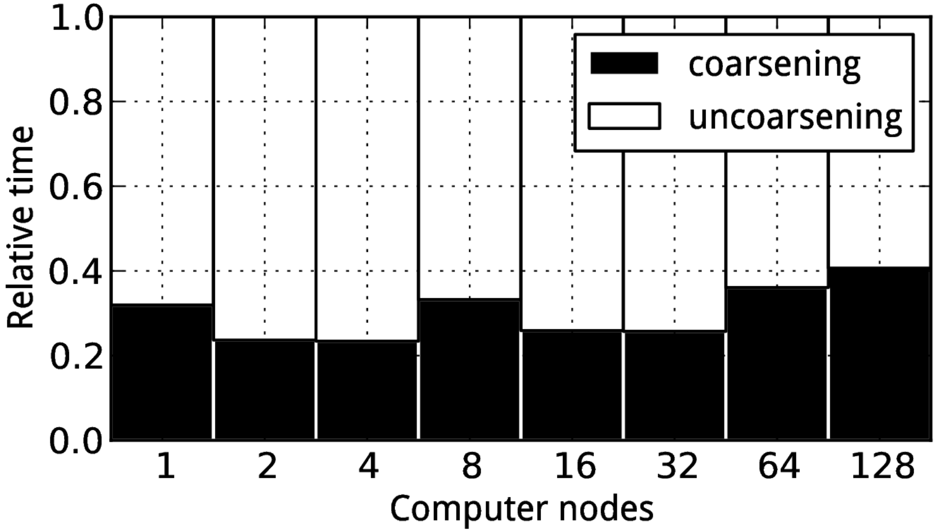

Luxembourg
Belgium
linear
Speedup
2000
1500
1000
500
0
16
32
64
128
Computer nodes

Coarsening and uncoarsening relative time

Belgium

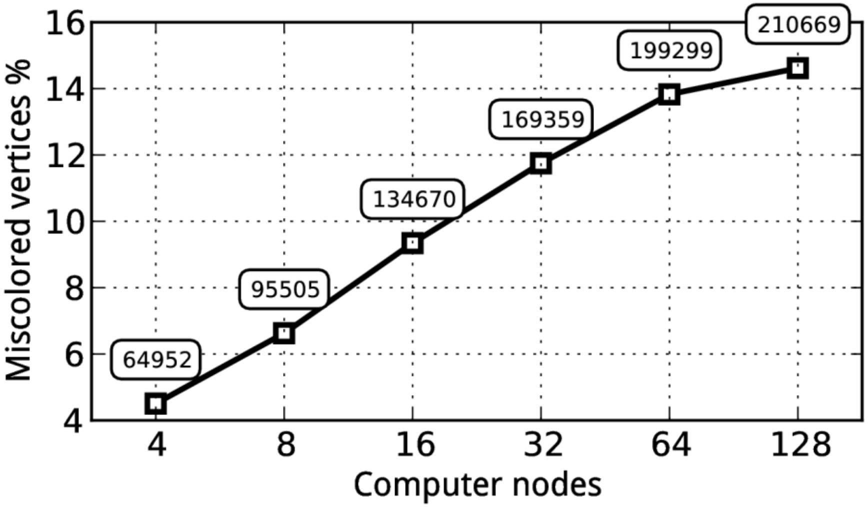


Luxembourg

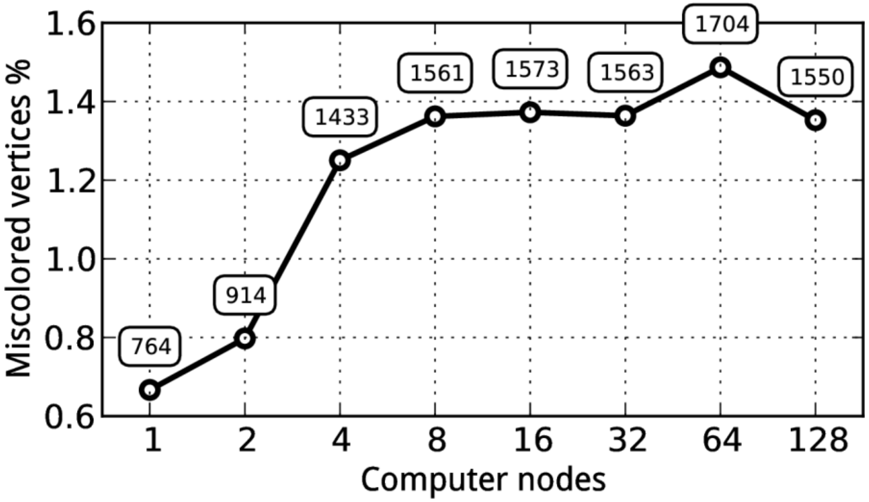


Quality

Belgium



Luxembourg



Random partitioning gives
30 % miscolored vertices

Conclusion

- An approach to partition very large graphs by means of a relational parallel DBMS, that was implemented on the basis of PostgreSQL.
- Good speedup at an acceptable quality loss.
- Try different partitioning schemes and other very large graph problems in future.
- Papers describing this research were published in LNCS (DEXA 2013 and ADBIS 2013 proceedings).

Thanks for attention!

- Questions?
  - Constantin Pan kvapen@gmail.com
  - Mikhail Zymbler zymbler@gmail.com