Everything you always wanted to know about multicore graph processing but were afraid to ask

Jasmina Malicevic — EPFL, Switzerland

Baptiste Lepers — EPFL, Switzerland

Willy Zwaenepoel — EPFL, Switzerland

USENIX Annual Technical Conference, Jul 12th -14th 2017 , Santa Clara, CA ,USA

Graphs are everywhere

Social networks

Item recommendation

Search and website ranking

The maze of graph analytics platforms

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

	In-memory	Out-of-core
Single machine	Ligra Polymer Galois	GraphChi X-Stream GridGraph Mosaic
Distributed	Pregel Powergraph PowerLyra Gemini	Chaos

The maze of graph analytics platforms

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

	In-memory	Out-of-core
Single machine	Ligra Polymer Galois	GraphChi X-Stream GridGraph Mosaic
Distributed	Pregel Powergraph PowerLyra Gemini	Chaos

Everything you always wanted to know…

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

What techniques work and why?

Why is our work different?

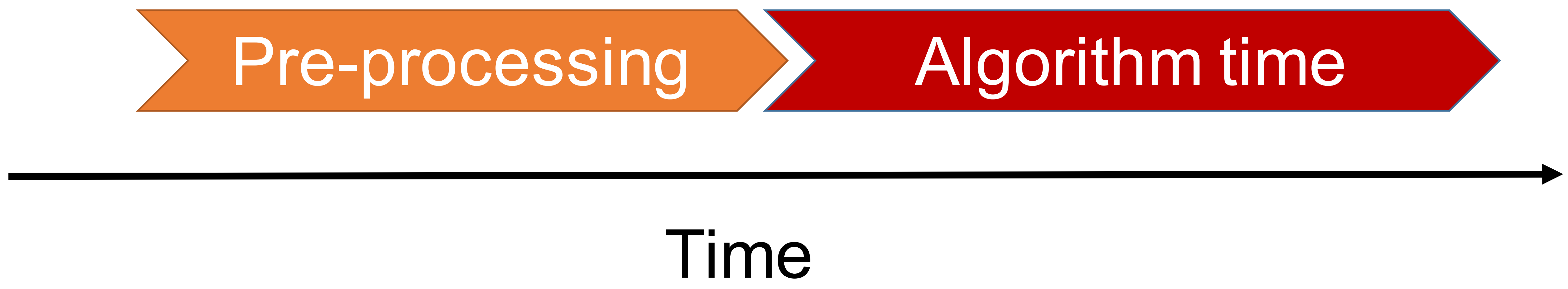
ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

- End-to-end evaluation

- Comparison of techniques, rather than systems

End-to-end evaluation

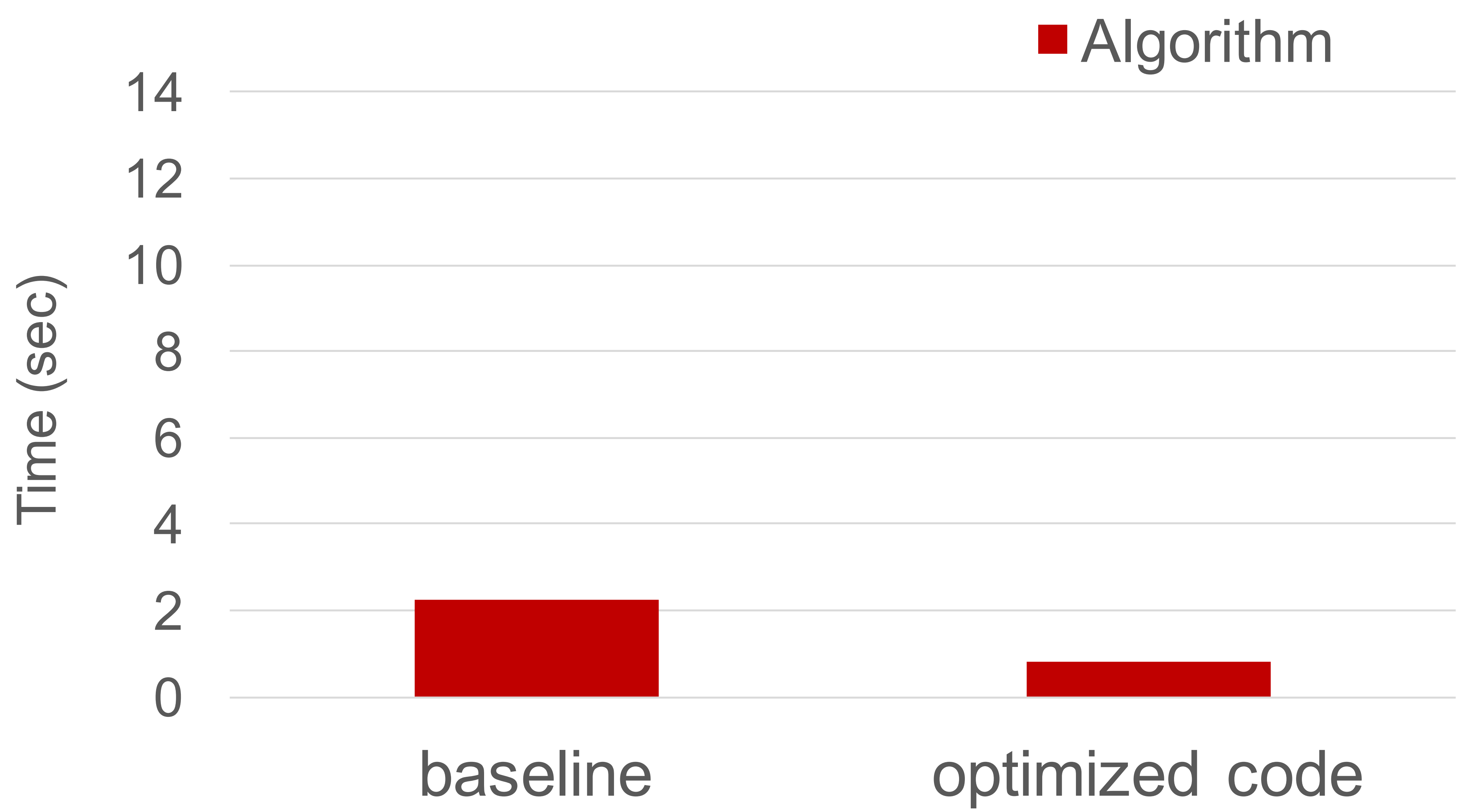
- Executing the algorithm is only one piece of the puzzle



End-to-end time = Pre-processing + Algorithm time

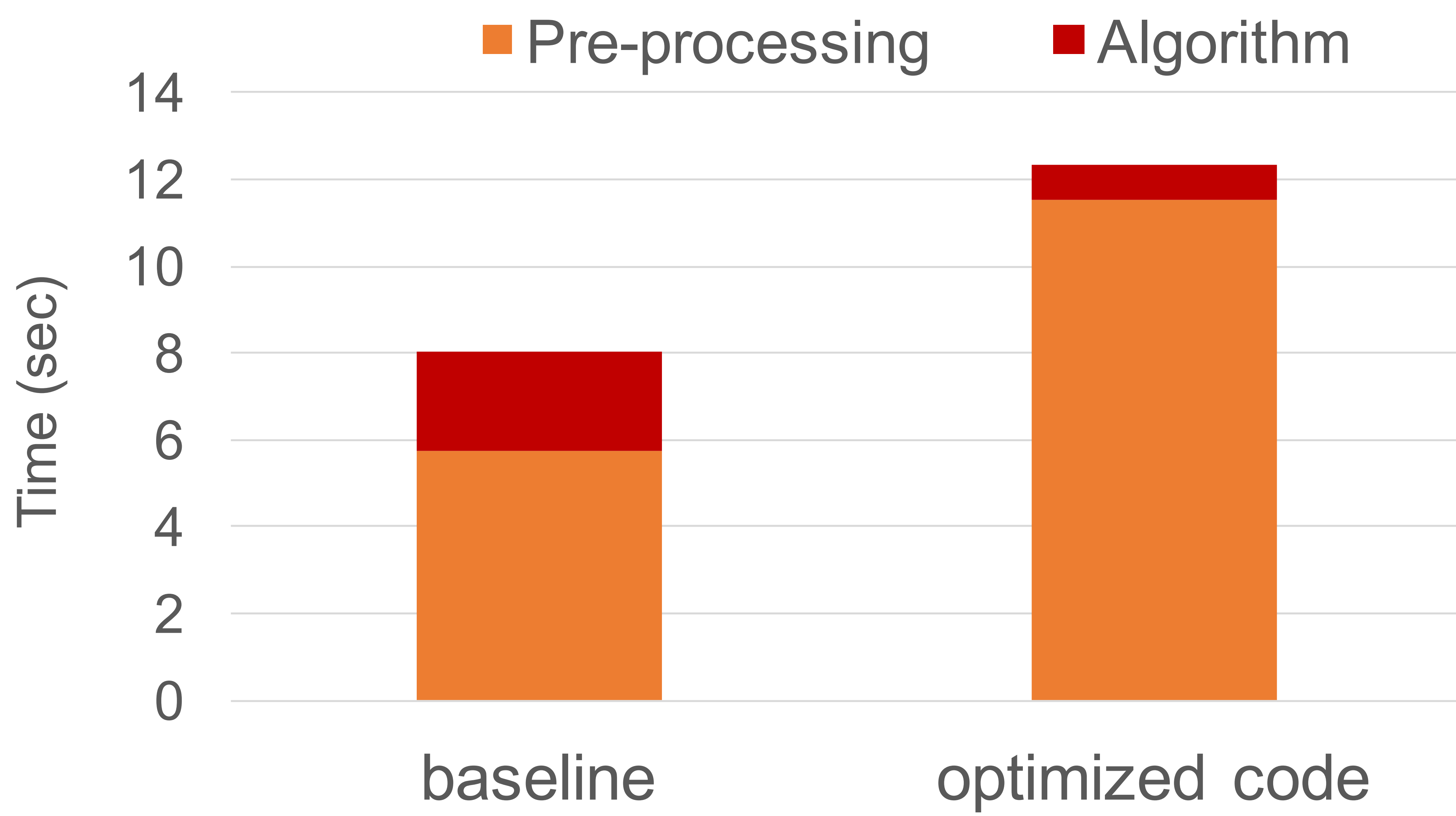
Motivation: Why end-to-end time?

- BFS on Twitter [Ligra]



Motivation: Why end-to-end time?

- BFS on Twitter [Ligra]



Need to understand the trade-off in end-to-end time!

Comparison of techniques not systems

- Implement techniques from different systems within **one system**

- Evaluation of techniques in isolation
  - Not constrained by system defined API

- Implementation is comparable/better than the original system

Questions we want to answer:

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Pre-processing

- How to represent the graph?
- Cost of creating the representation?
- What data layout is best?

Algorithm

- Can we improve cache locality?
- Should we optimize for NUMA?
- Information flow: **push**, **pull** or a **both**?

The answers depend on:

- Algorithm – differ in # of active vertices per iteration
  - Only a subset active: **BFS**
  - Entire graph active: **Pagerank**, **SpMV**…
- Graph shape
  - Social networks (power law) graphs
  - **Synthetic graph; 1B edges 64M vertices**
  - Stored as edge array

Questions we want to answer:

Pre-processing

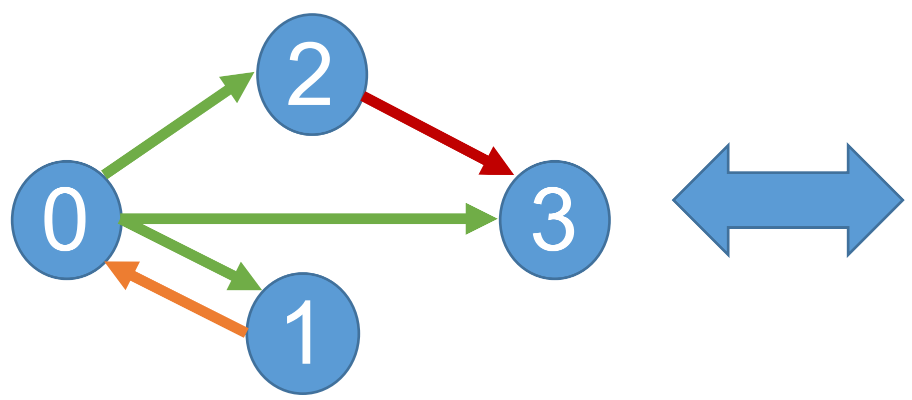
- **How to represent the graph?**
- Cost of creating the representation?
- What data layout is best?

Algorithm

- Can we improve cache locality?
- Should we optimize for NUMA?
- Information flow: **push**, **pull** or a **both**?

Graph representation

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE



Edge array

0 - 2	2 -3	0 - 3	1 - 0	0 - 1

✓ Layout is the same as input – no pre-processing

x To locate edges of a vertex, need to read all edges

Adjacency list: outgoing edges

0 →	2	3	1
1 →	0		
2 →	3		
3			

x Pre-processing to group edges by vertex

✓ Easy to locate edges of a particular vertex

Questions we want to answer:

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

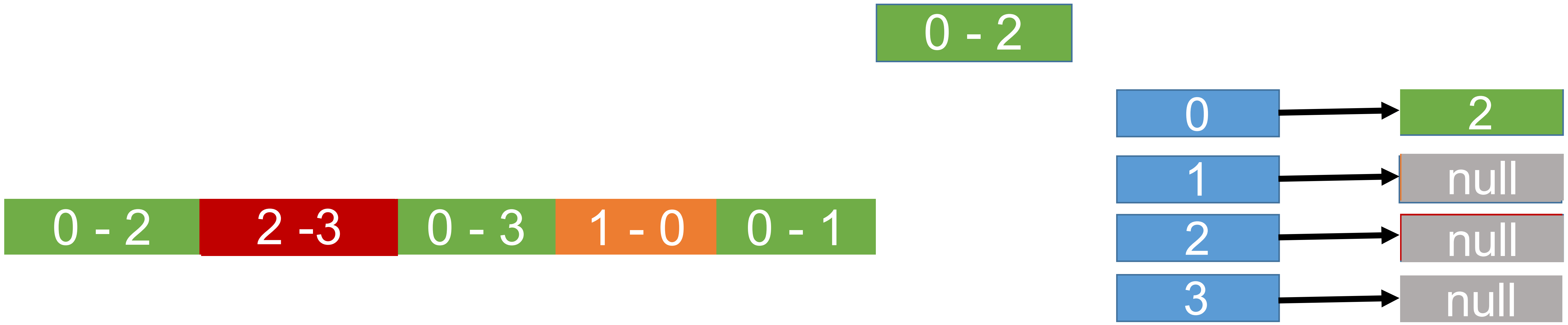
Pre-processing

- ✓ How to represent the graph?
  - Adjacency lists
  - Edge arrays
- **Cost of creating the representation?**
- What data layout is best?

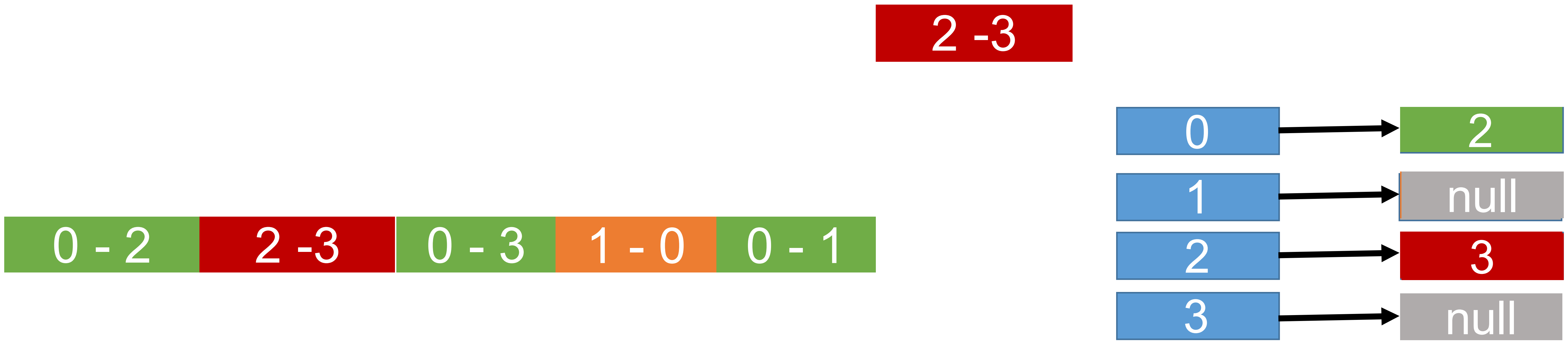
Algorithm

- Can we improve cache locality?
- Should we optimize for hardware?(NUMA)
- Information flow: **push**, **pull** or a **both**?

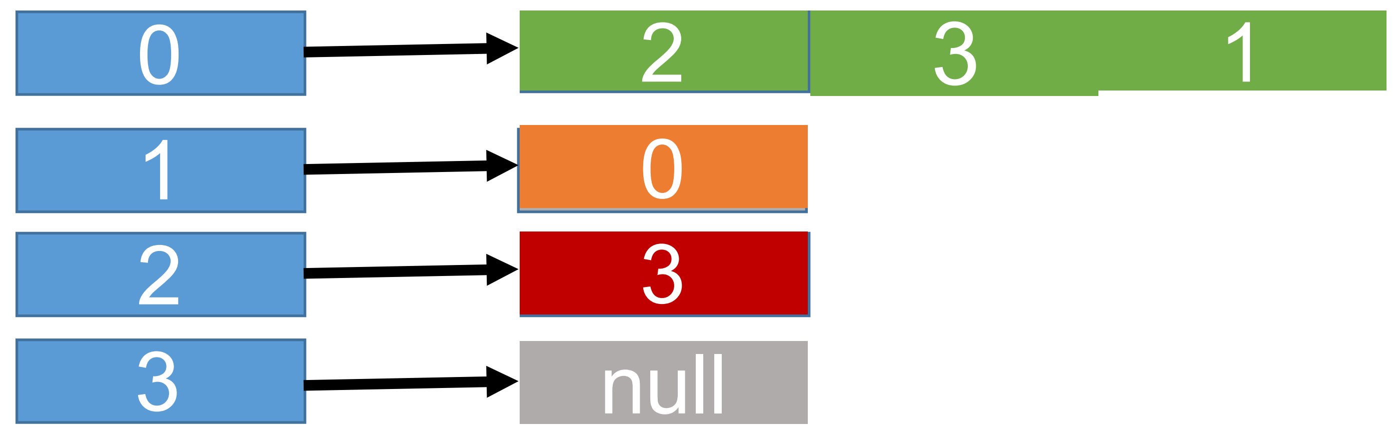
Creating adjacency lists using dynamic allocation



Creating adjacency lists using dynamic allocation



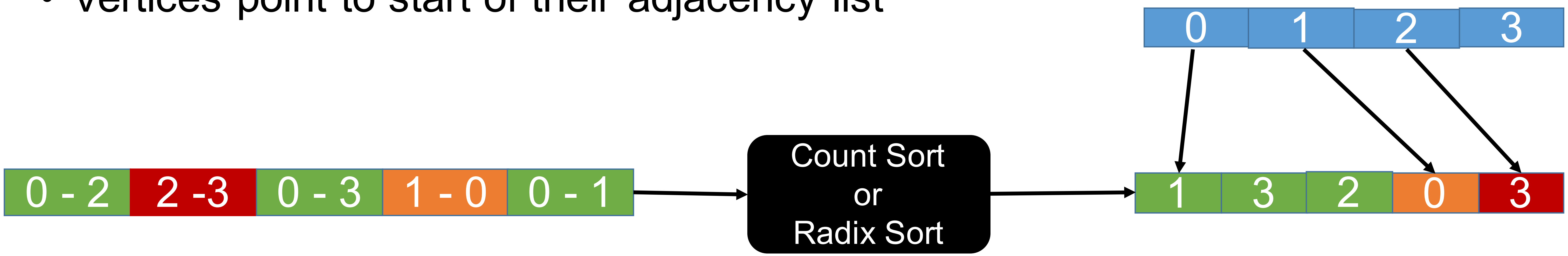
Creating adjacency lists using dynamic allocation



- Frequent reallocations
- Adjacency lists spread out in memory

Creating adjacency lists using sorting

- Load edge array into memory
- Sort by source or destination
- Vertices point to start of their adjacency list

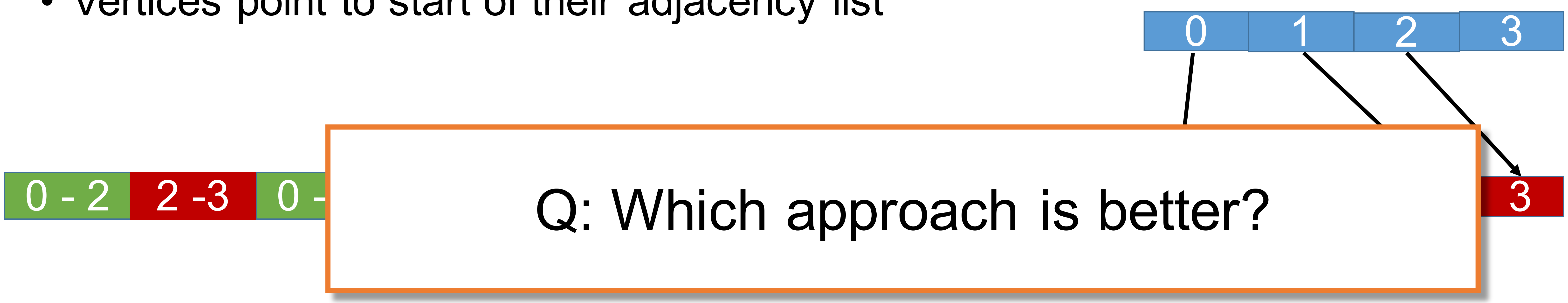


✓ Avoid reallocations

✓ Adjacency lists contiguous in memory

Creating adjacency lists using sorting

- Load edge array into memory
- Sort by source or destination
- Vertices point to start of their adjacency list



Q: Which approach is better?

✓ Avoid reallocations

✓ Adjacency lists contiguous in memory

Which pre-processing method is better?

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

Pre-processing technique	Time (sec)	LLC misses
Dynamic	15.0	69%
Count sort	13.5	71%
Radix sort	**4.0**	**26%**

Radix sort low LLC miss rate => 3.5X better

Questions we want to answer:

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

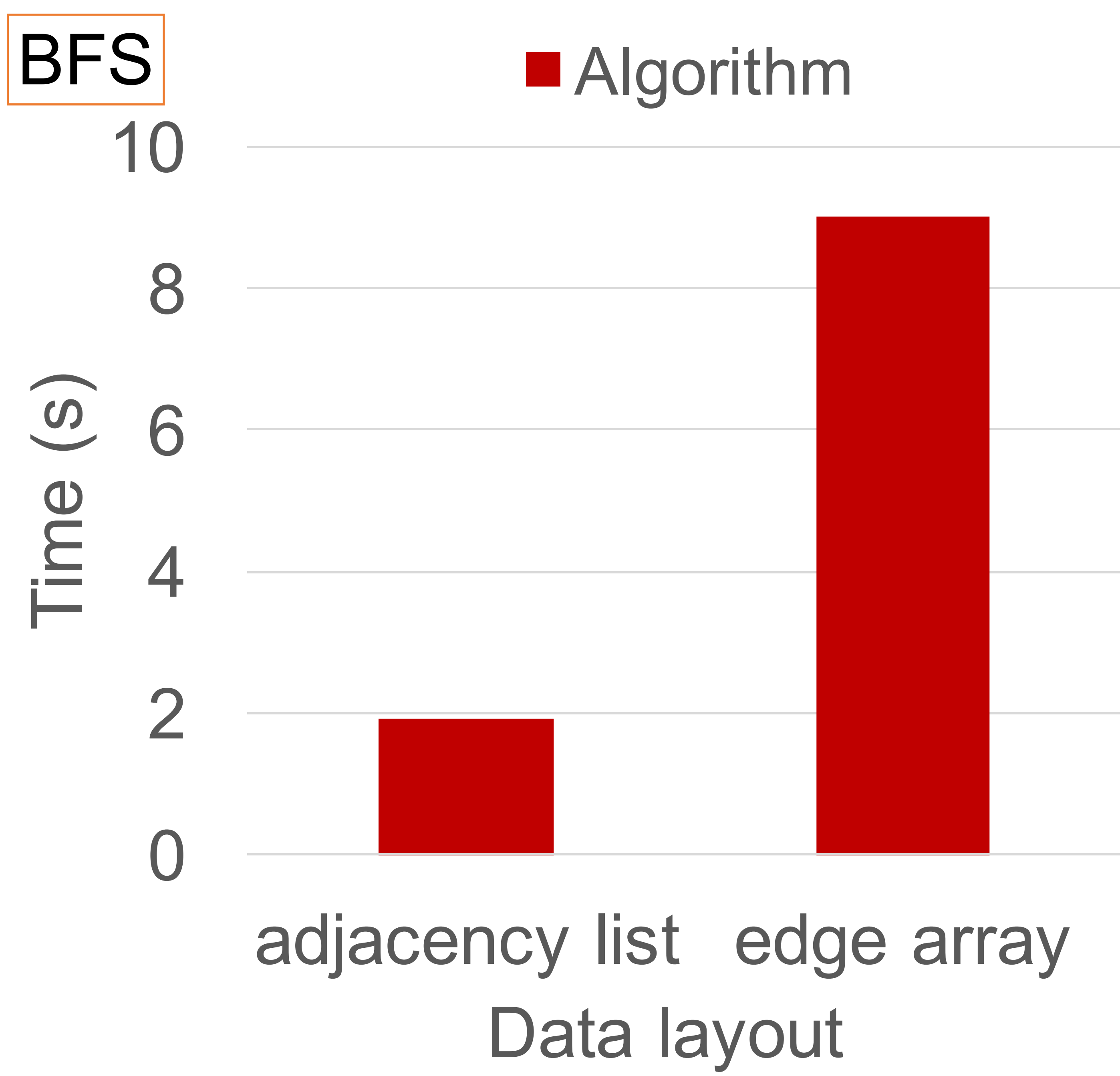
Pre-processing

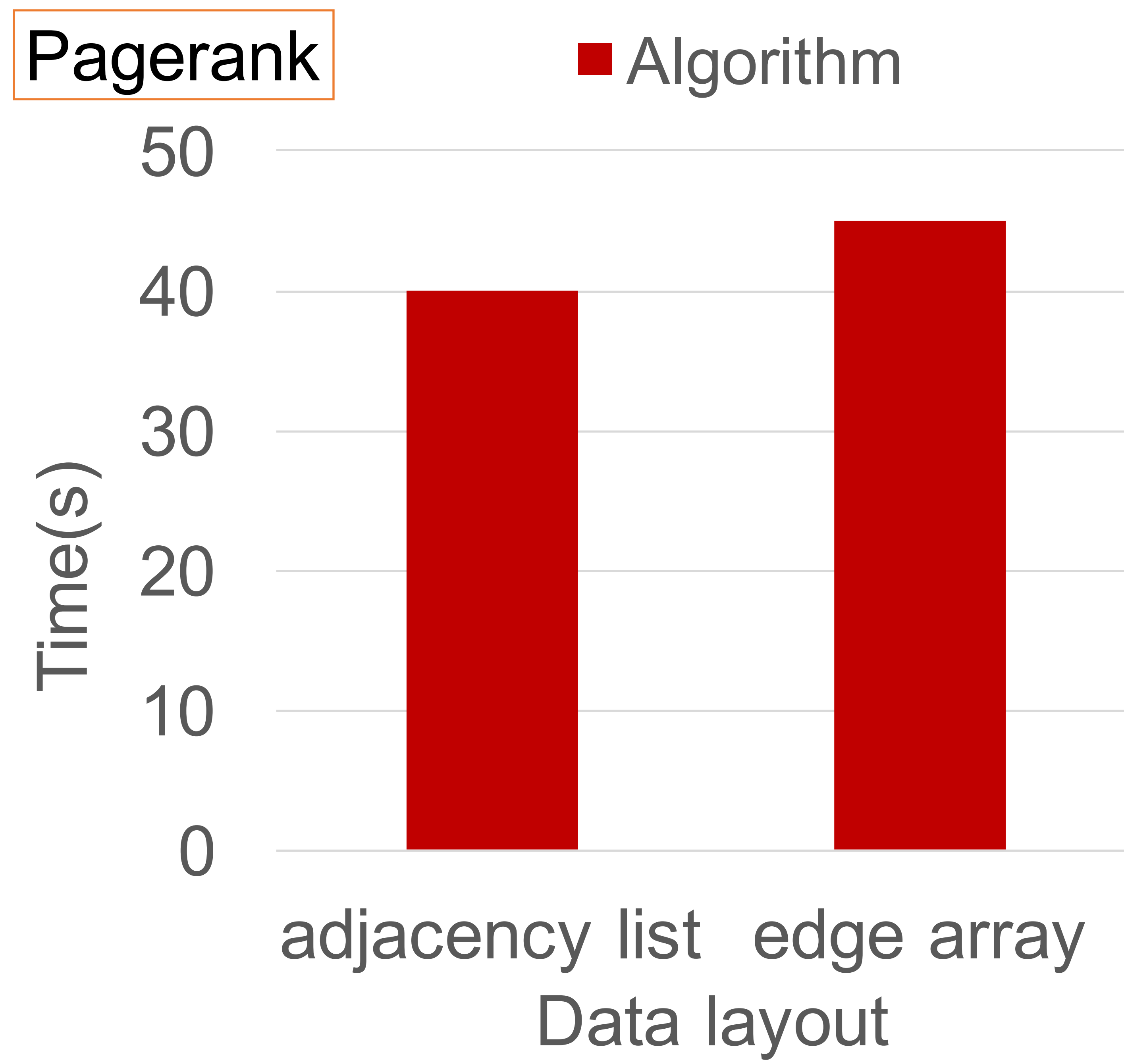
- ✓ How to represent the graph?
  - ○ Adjacency lists ○ Edge arrays
- ✓ Cost of creating the representation?
  - ○ Radix sort wins for adjacency lists
- **What data layout is best?**

Algorithm

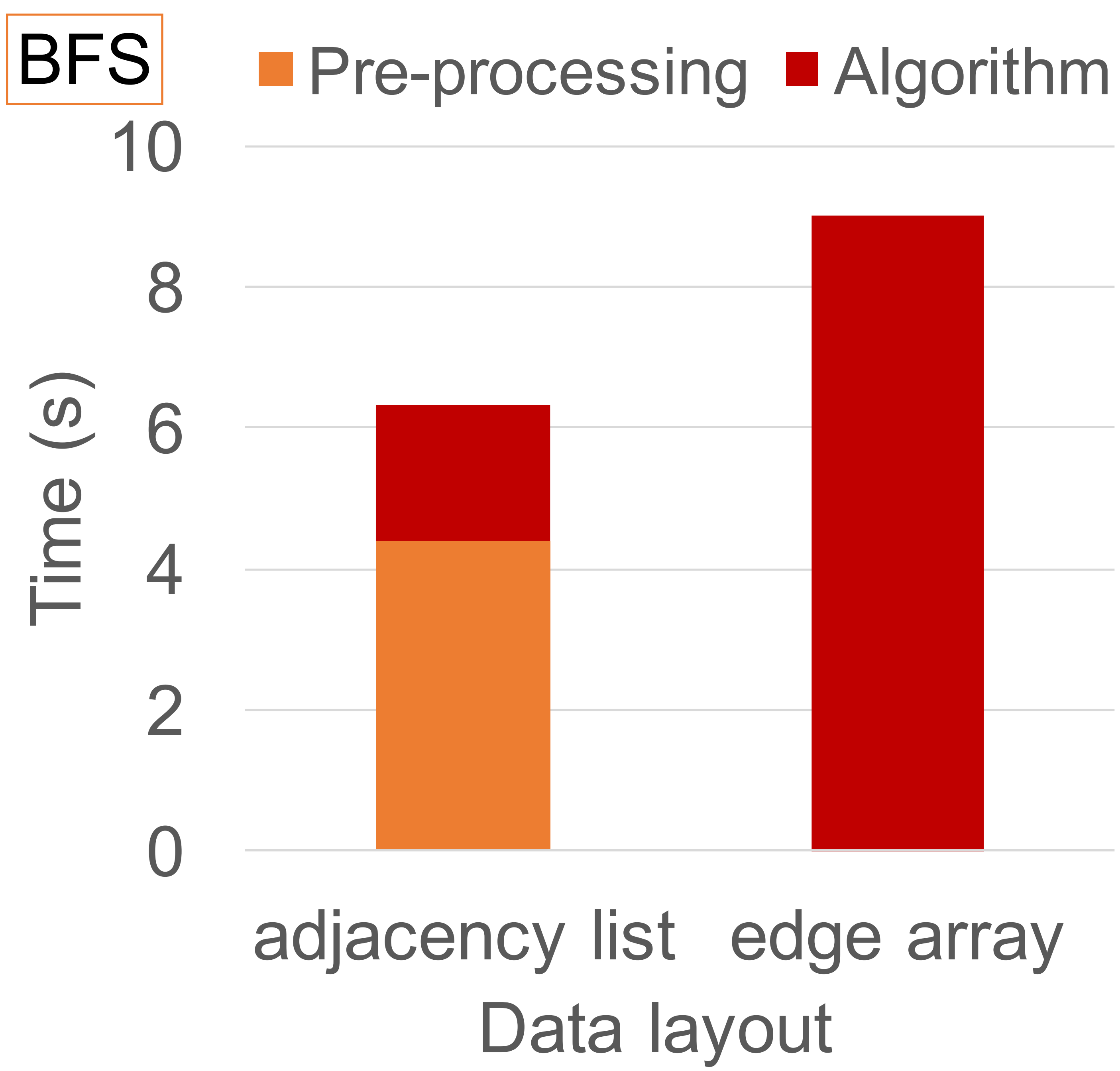
- Can we improve cache locality?
- Should we optimize for NUMA?
- Information flow: **push**, **pull** or a **both**?

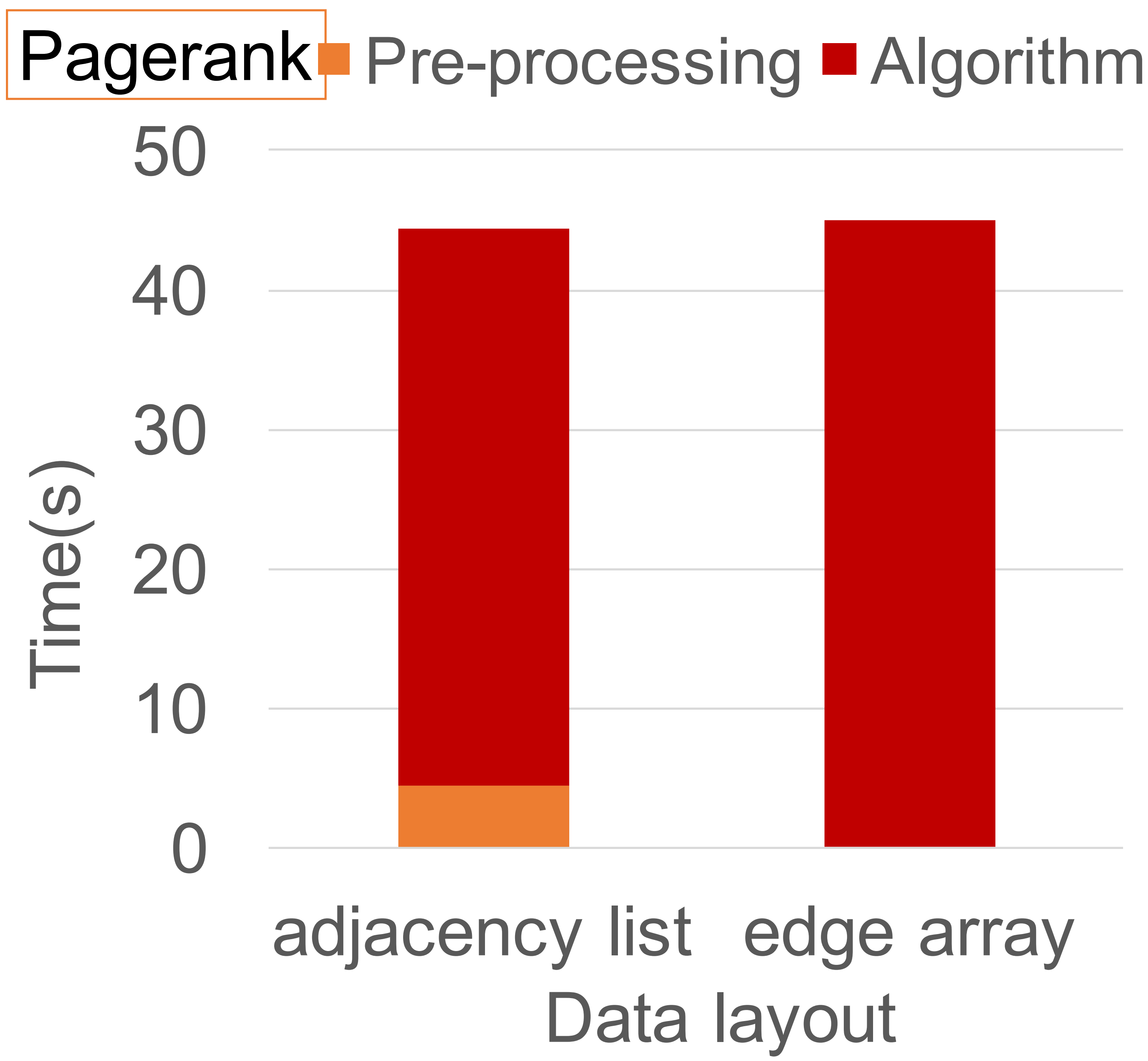
Which is data layout is better?





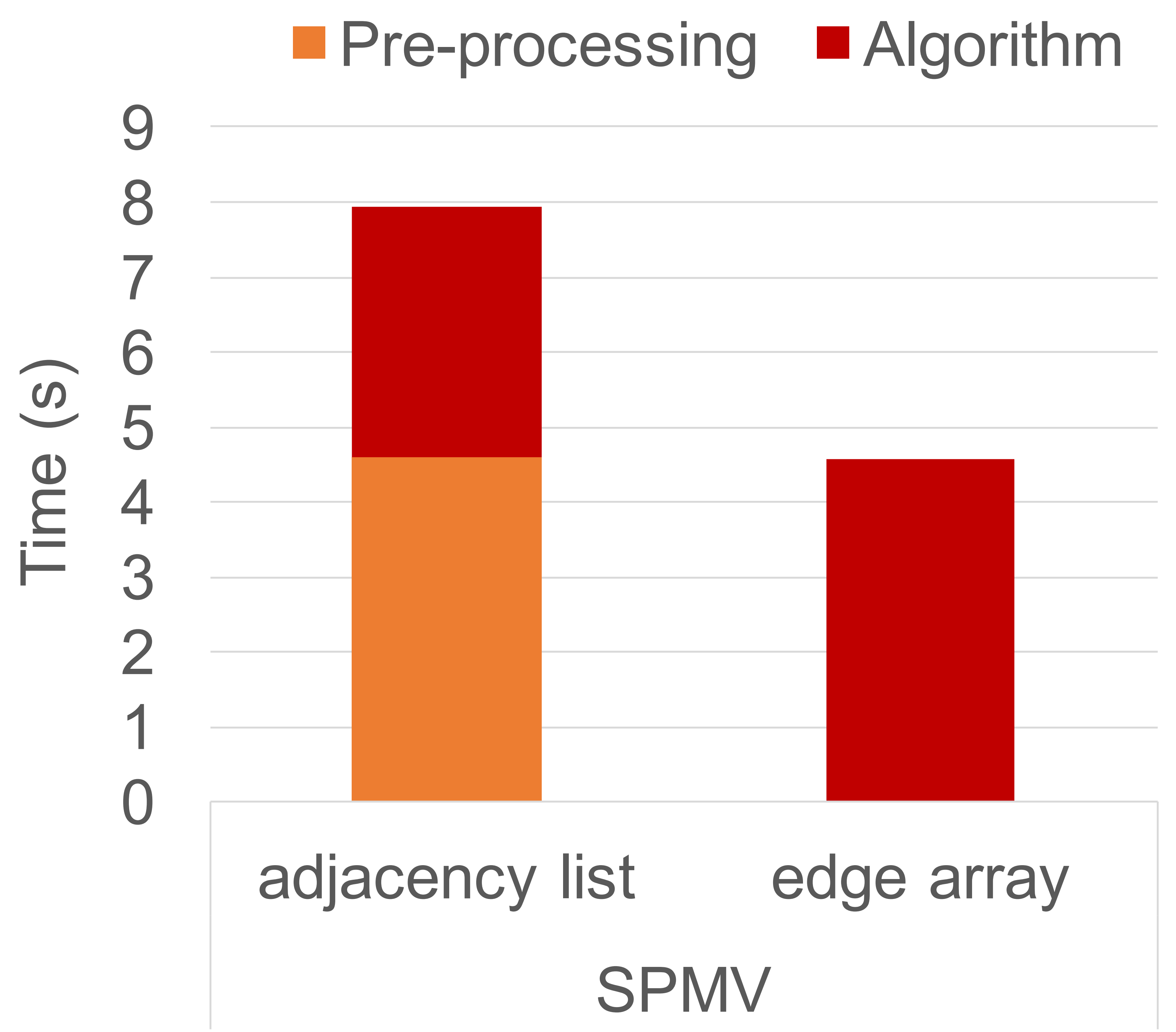
Which is better?





Adjacency lists always wins?

- SpMV – one pass over the edge array
- Pre-processing not amortized
  - Cost is = pass over edge array



Questions we want to answer:

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

Pre-processing

- ✓ How to represent the graph? → ○ Adjacency lists ○ Edge arrays
- ✓ Cost of creating the representation? → ○ Radix sort wins for adjacency lists
- ✓ What data layout is best? → ○ BFS: Adj. list ○ PR: Adj.list ○ SpMV: Edge array

Algorithm

- **Can we improve cache locality?**
- Should we optimize for NUMA?
- Information flow: **push**, **pull** or a **both**?

Memory accesses – edge arrays

Edge array:

0 - 3	2 -3	0 - 1	1 - 0	0 - 2

Vertex state array:

S(0)
S(1)
S(2)
S(3)

Fetch edge:

Fetch state of source:

Fetch state of destination:

Memory accesses – edge arrays

Edge array:	0 - 3	2 -3	0 - 1	1 - 0	0 - 2

Vertex state array:

S(0)
S(1)
S(2)
S(3)

Fetch edge: 0 - 3

Fetch state of source:

Fetch state of destination:

✓Cache-friendly edge read

Memory accesses – edge arrays

Edge array:

0 - 3	2 -3	0 - 1	1 - 0	0 - 2

Vertex state array:

S(0)
S(1)
S(2)
S(3)

Fetch edge: 0 - 3

Fetch state of source: S(0)

Fetch state of destination:

✓ Cache-friendly edge read

x Potentially random access to source state

Memory accesses – edge arrays

Edge array: 0 - 3 | 2 -3 | 0 - 1 | 1 - 0 | 0 - 2

Vertex state array:

S(0)
S(1)
S(2)
S(3)

Fetch edge: 0 - 3

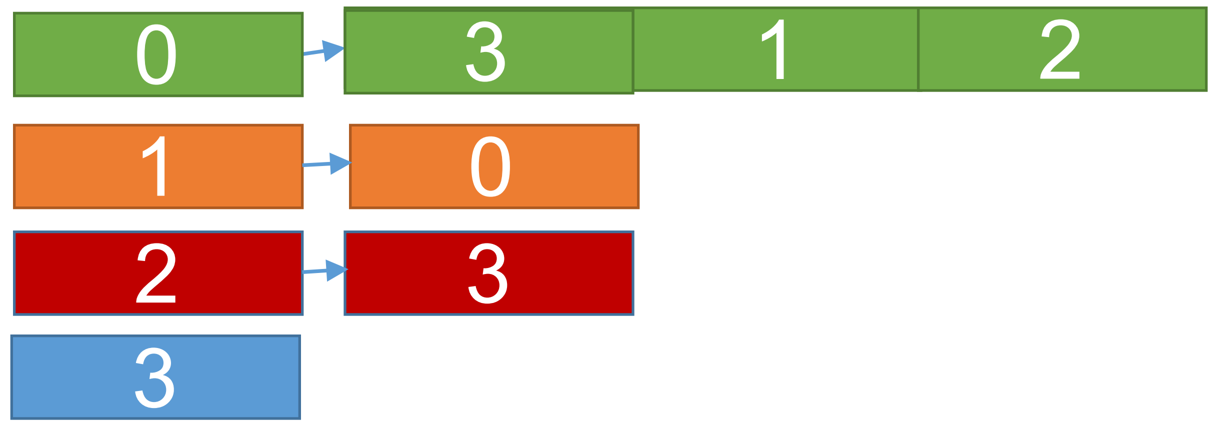
Fetch state of source: S(0)

Fetch state of destination: S(3)

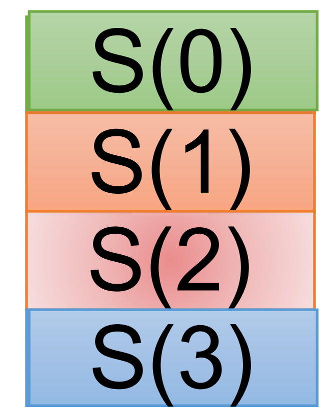
✓ Cache-friendly edge read
x Potentially random access to source state
x Random access to destination state

Memory accesses – adjacency lists

Adjacency list



Vertex state array:



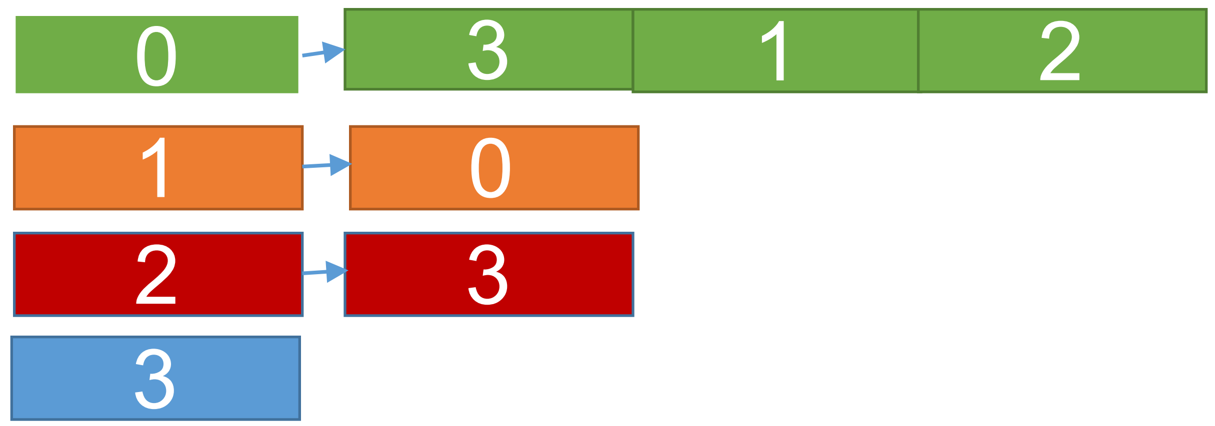
Fetch edge:

Fetch state of source:

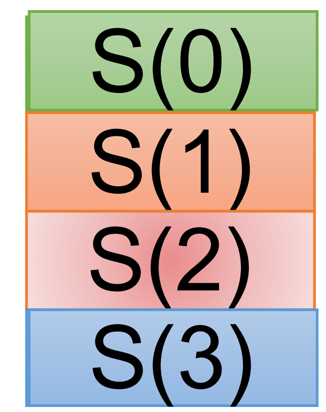
Fetch state of destination:

Memory accesses – adjacency lists

Adjacency list



Vertex state array:



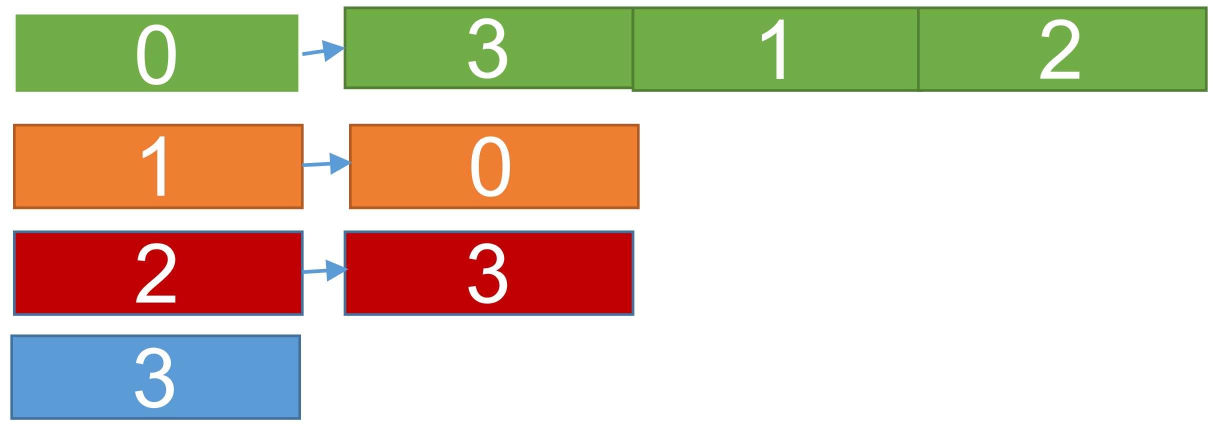
Fetch edge:

Fetch state of source:

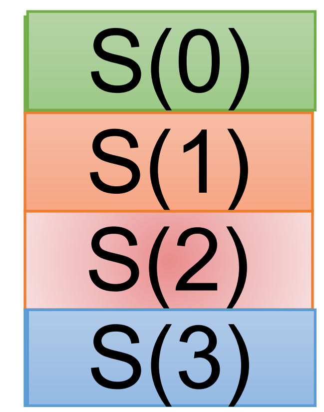
Fetch state of destination:

Memory accesses – adjacency lists

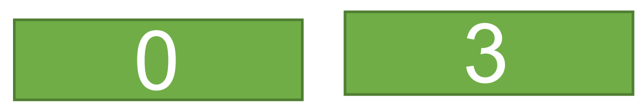
Adjacency list



Vertex state array:



Fetch edge:



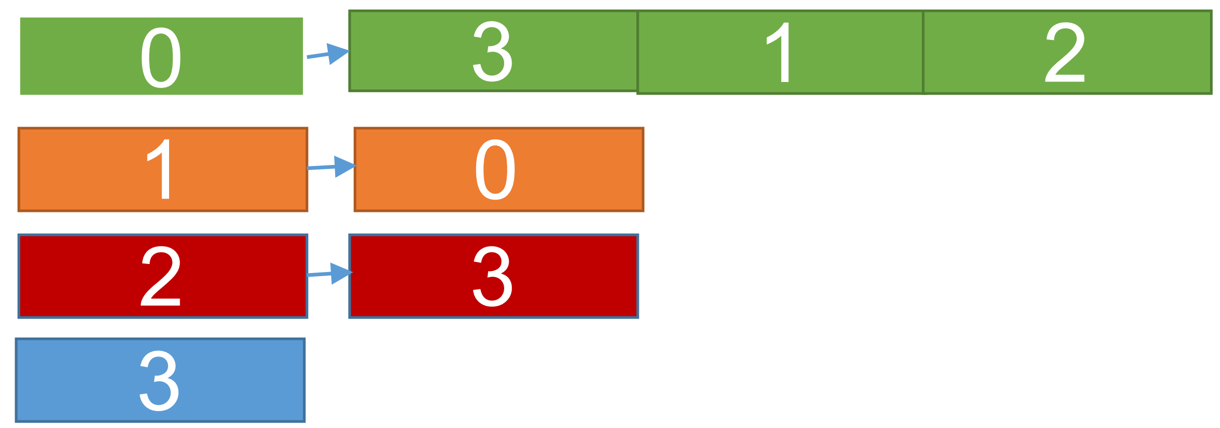
Fetch state of source:

Fetch state of destination:

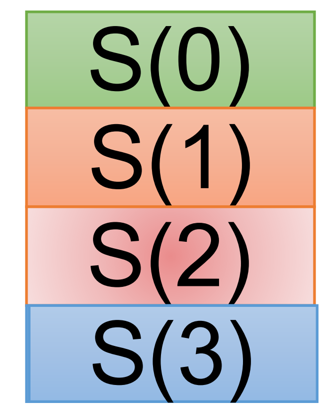
✓Cache-friendly edge read

Memory accesses – adjacency lists

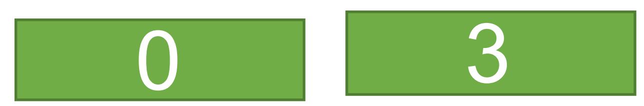
Adjacency list



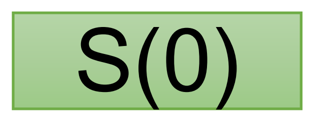
Vertex state array:



Fetch edge:



Fetch state of source:



Fetch state of destination:

✓Cache-friendly edge read
✓Cache-friendly source state read

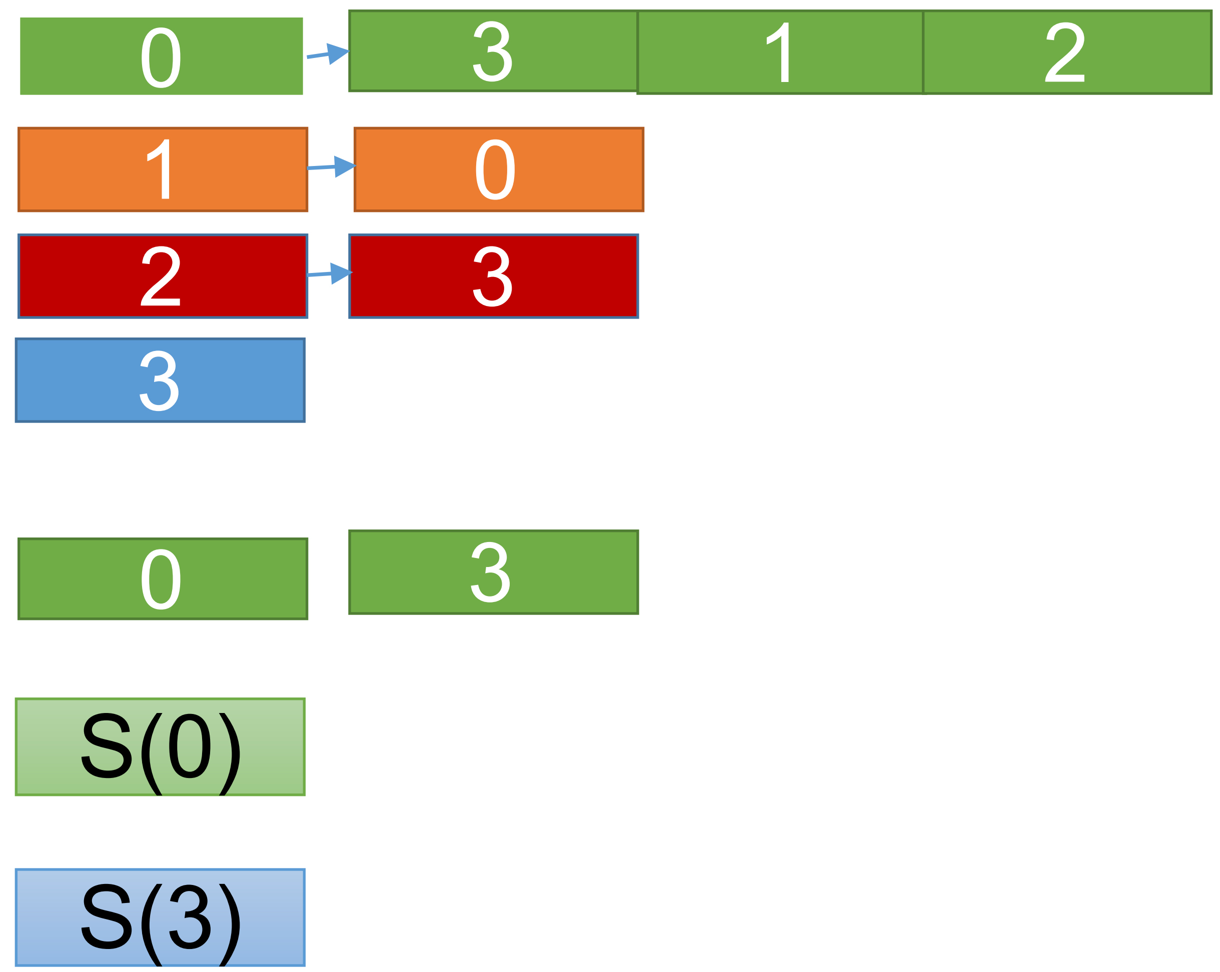
Memory accesses – adjacency lists

Adjacency list

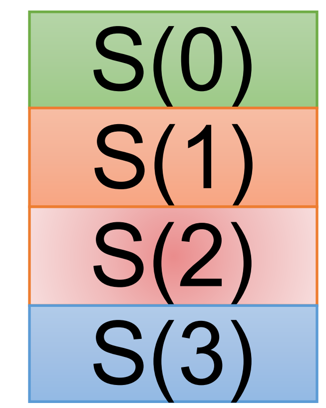
Fetch edge:

Fetch state of source:

Fetch state of destination:



Vertex state array:



✓ Cache-friendly edge read
✓ Cache-friendly source state read
x Random access to destination state

LLC miss rate

Data layout	BFS	PageRank
Edge array	57%	83%
Adjacency list	63%	78%

LLC miss rate

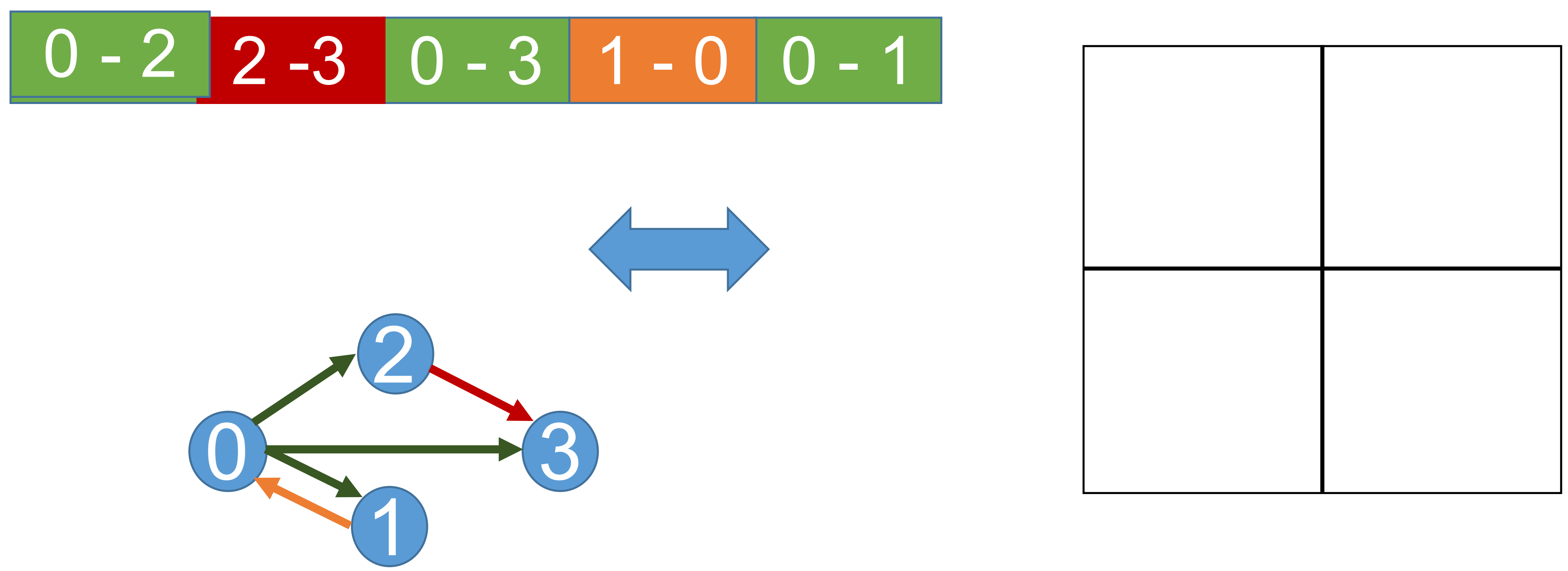
Data layout	BFS	PageRank
Edge array	57%	83%
Adjacency list	63%	78%

Q: Can the miss rate be improved ? At what cost?

Improving cache locality

Idea: Constrain the number of vertices accessed

Solution: Use out-of core technique – 2D Grid [from GridGraph]

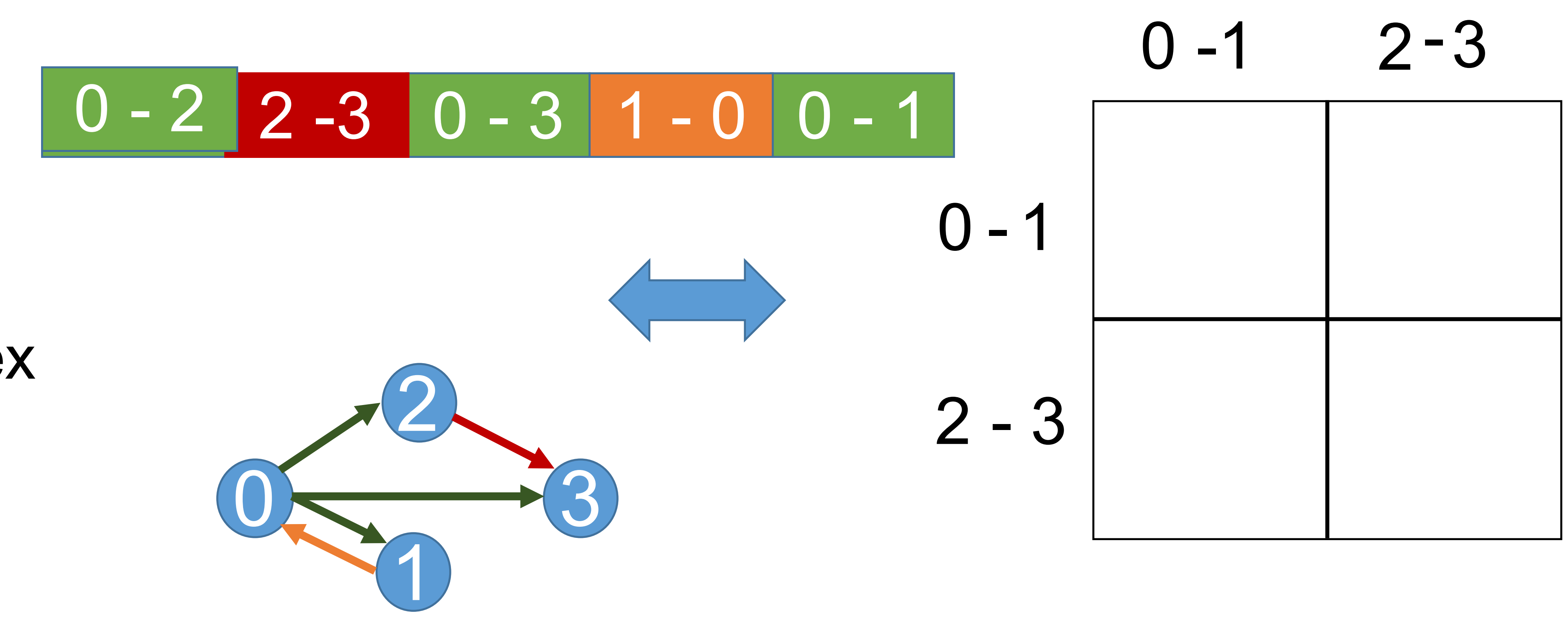


Improving cache locality

Idea: Constrain the number of vertices accessed

Solution: Use out-of core technique – 2D Grid [from GridGraph]

- Vertices divided into ranges
- Edges placed in a cell:
  - Row of source vertex
  - Column of destination vertex

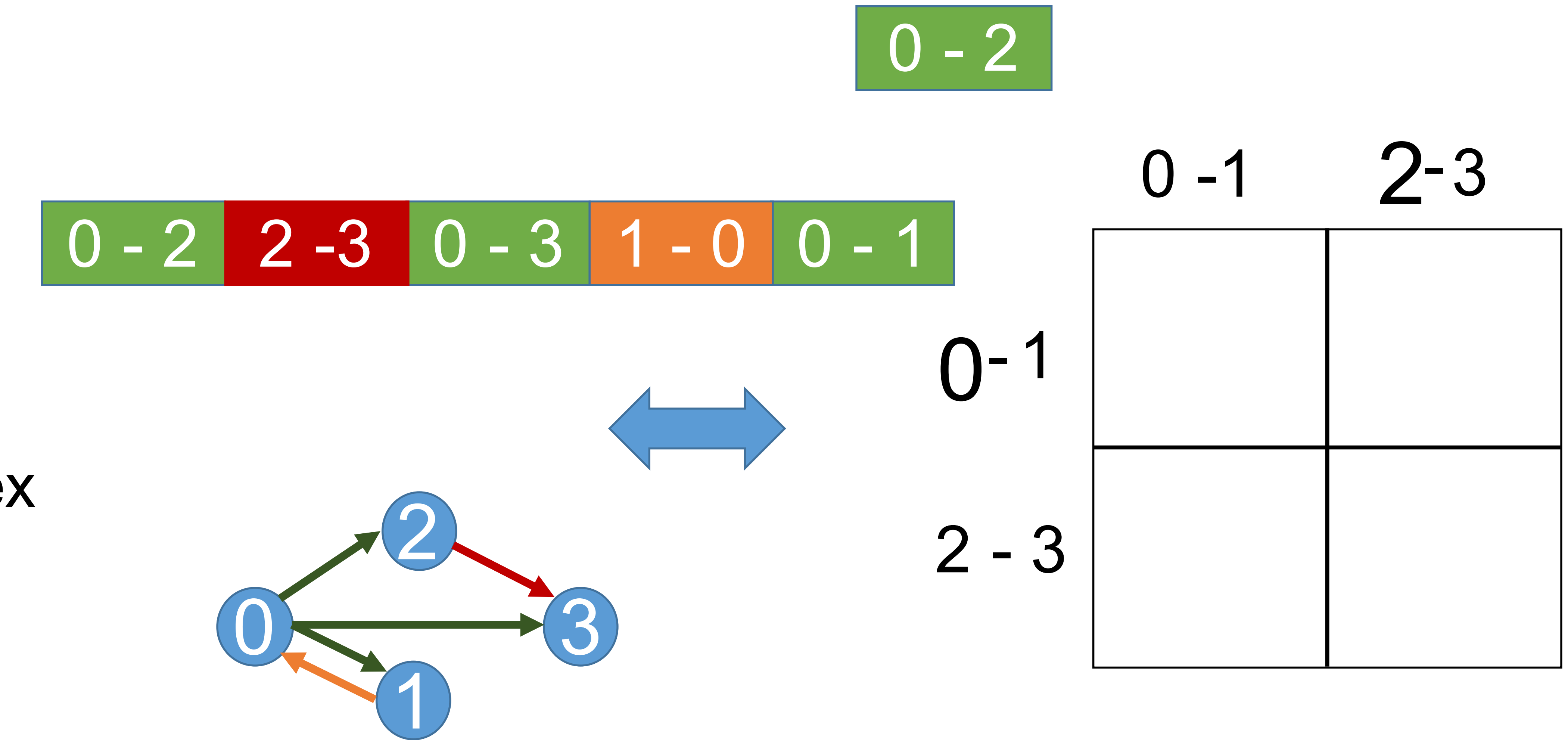


Improving cache locality

Idea: Constrain the number of vertices accessed

Solution: Use out-of core technique – 2D Grid [from GridGraph]

- Vertices divided into ranges
- Edges placed in a cell:
  - Row of source vertex
  - Column of destination vertex

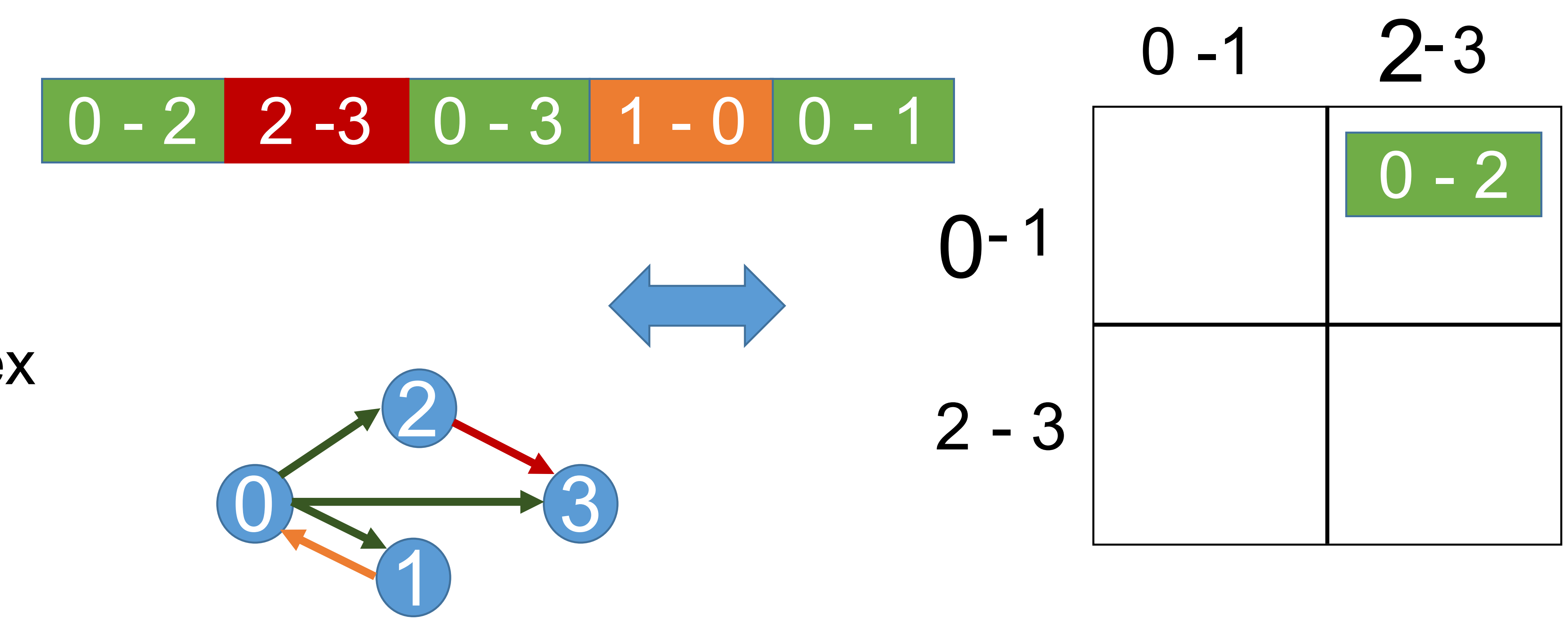


Improving cache locality

Idea: Constrain the number of vertices accessed

Solution: Use out-of core technique – 2D Grid [from GridGraph]

- Vertices divided into ranges
- Edges placed in a cell:
  - Row of source vertex
  - Column of destination vertex

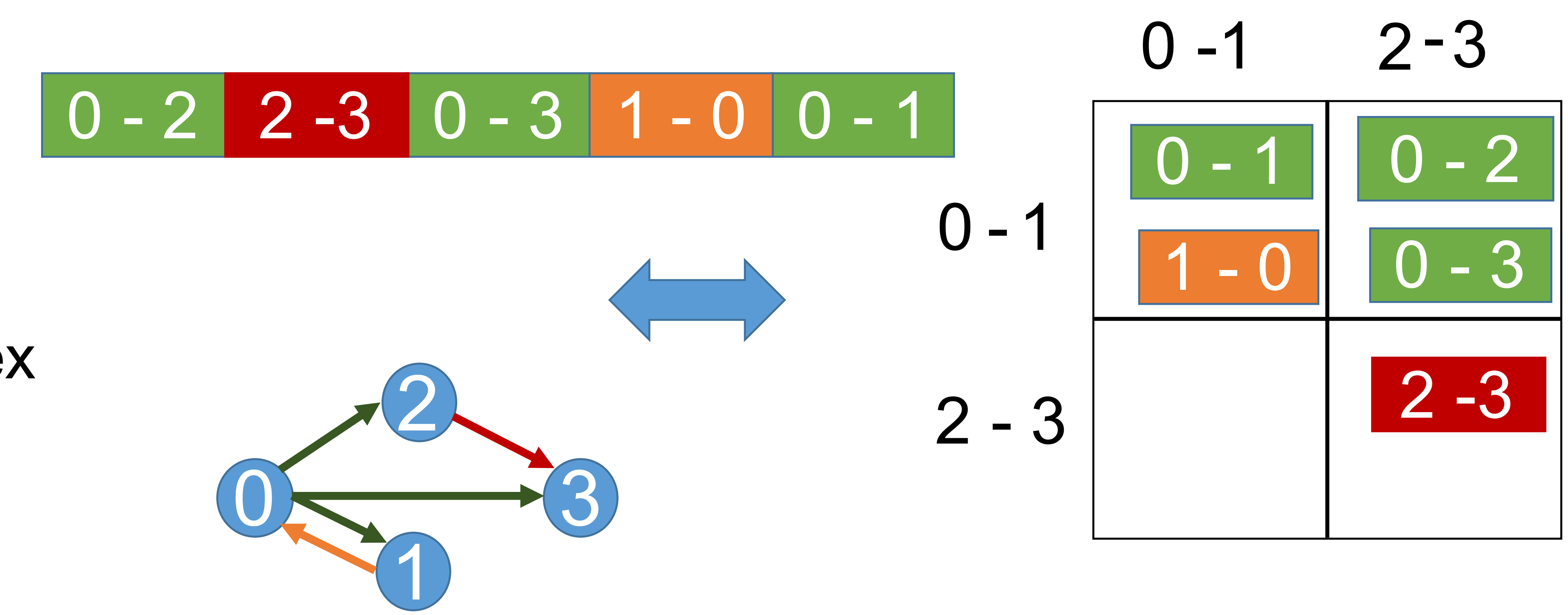


Improving cache locality

Idea: Constrain the number of vertices accessed

Solution: Use out-of core technique – 2D Grid [from GridGraph]

- Vertices divided into ranges
- Edges placed in a cell:
  - Row of source vertex
  - Column of destination vertex

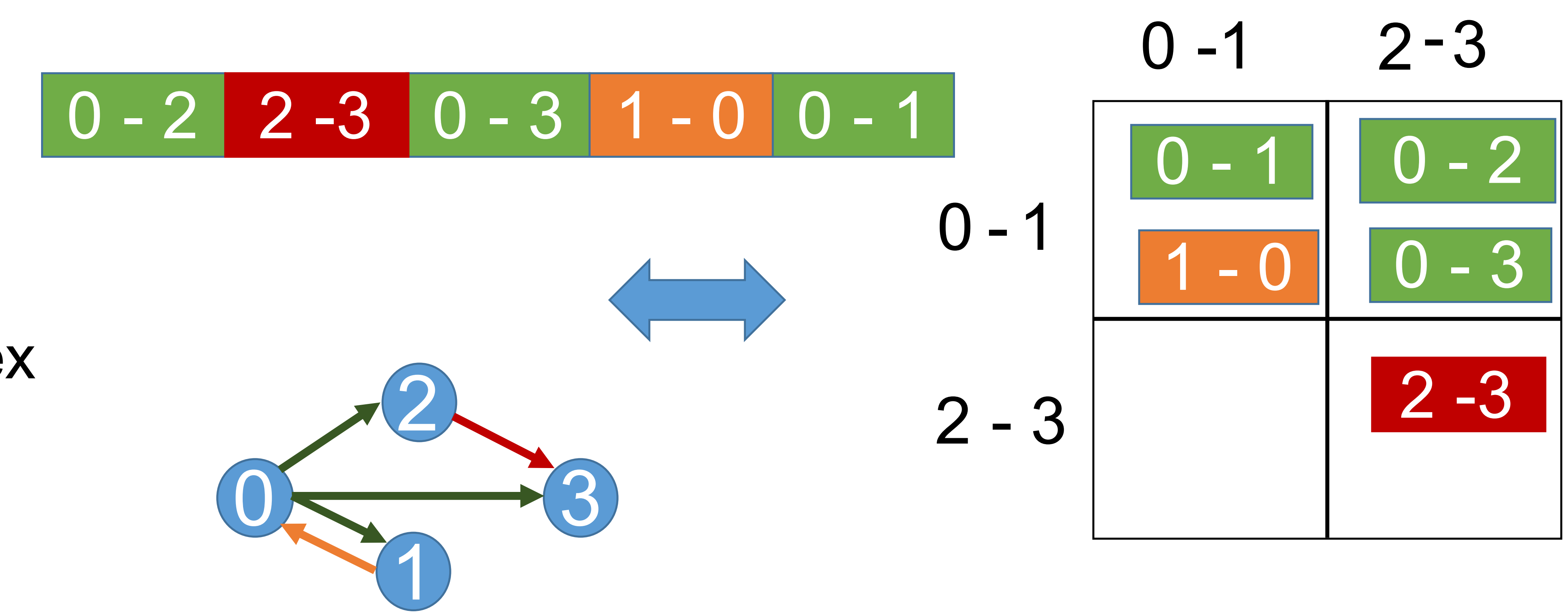


Improving cache locality

Idea: Constrain the number of vertices accessed

Solution: Use out-of core technique – 2D Grid [from GridGraph]

- Vertices divided into ranges
- Edges placed in a cell:
  - Row of source vertex
  - Column of destination vertex



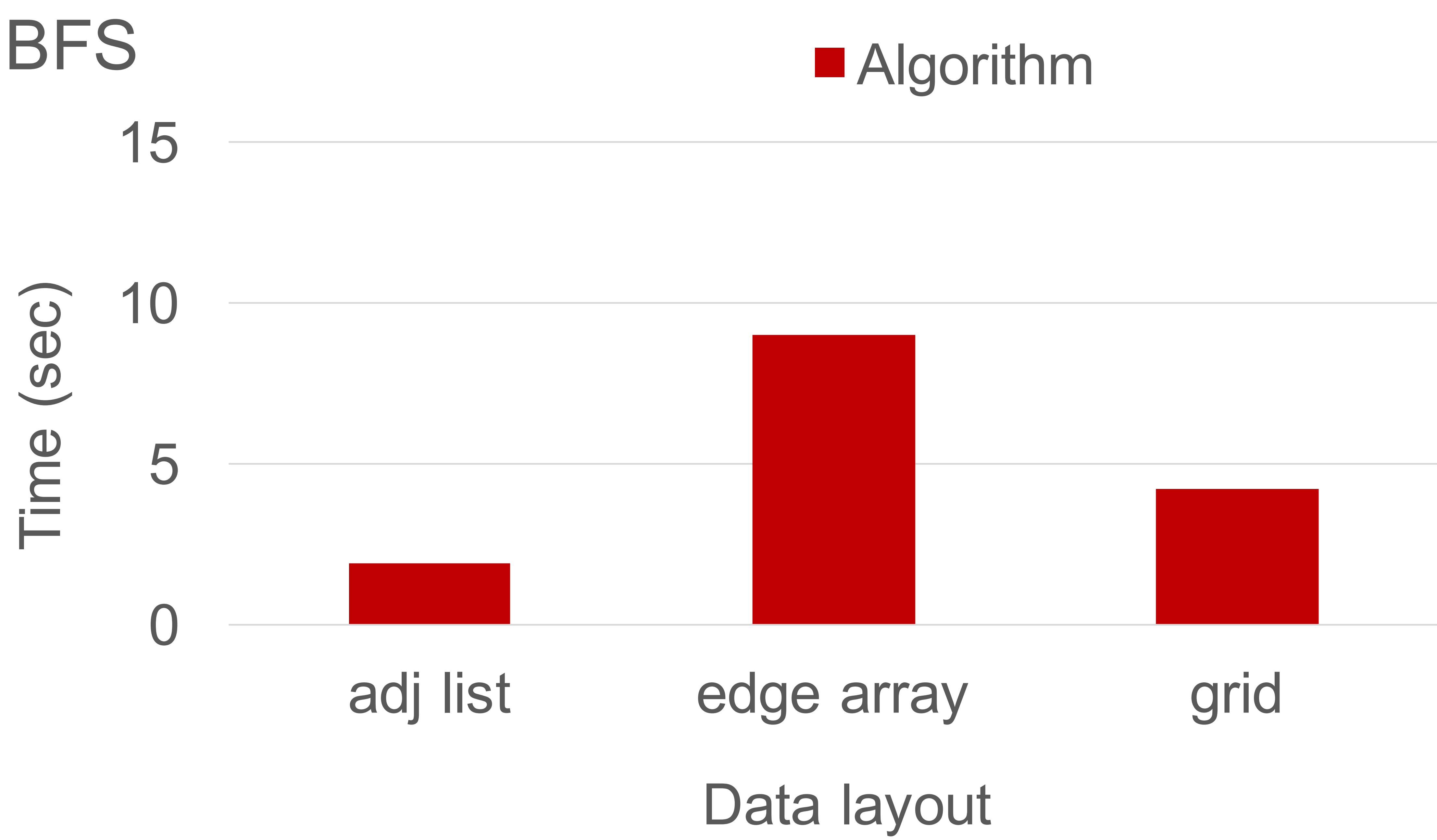
- Compute over cells of row or column

Cache-miss rate: Grid

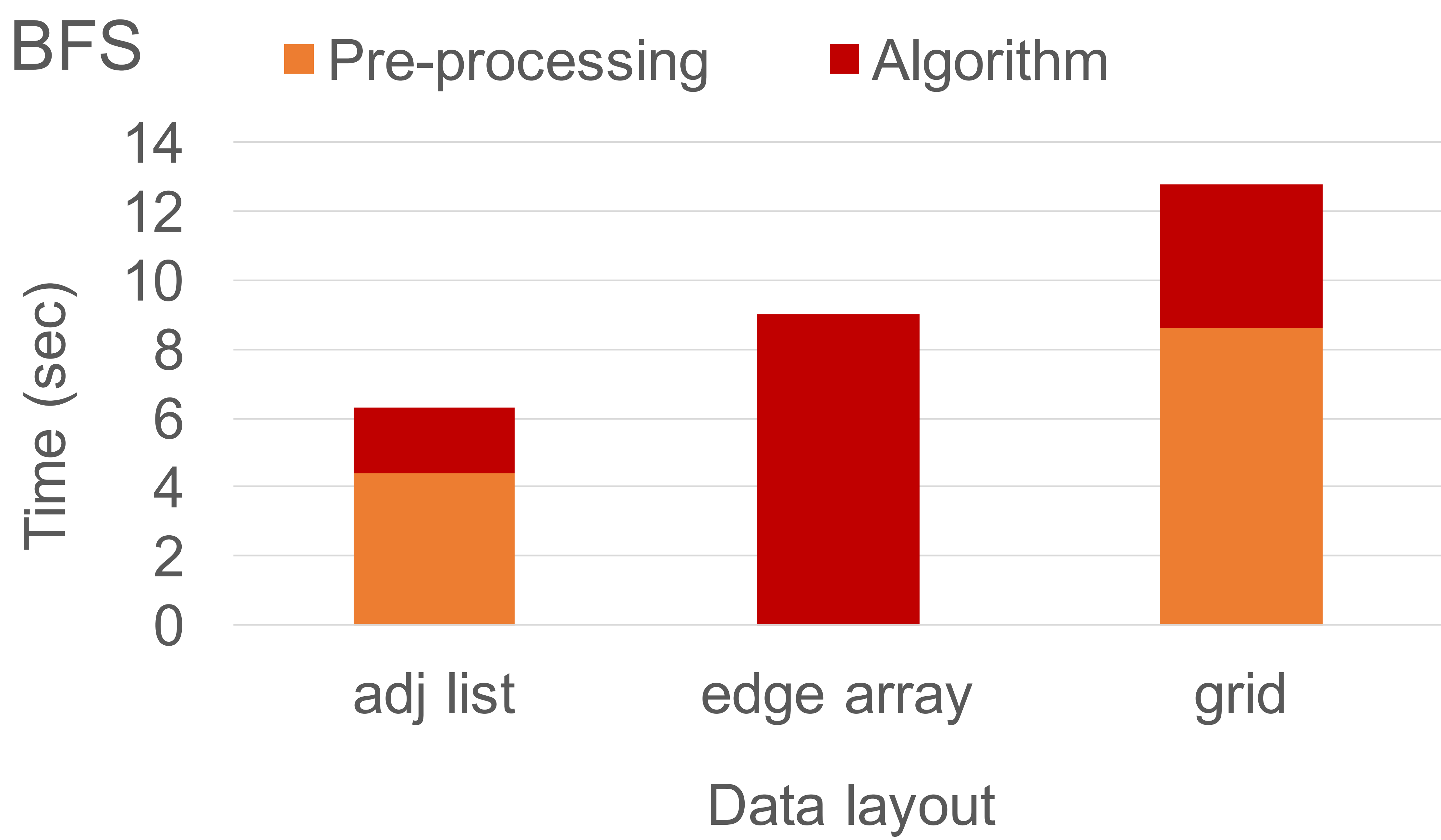
ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Data layout	BFS	PageRank
Edge array	57%	83%
Adjacency list	63%	78%
2D Grid	**23%**	**35%**

Evaluation: cache-optimization (BFS)

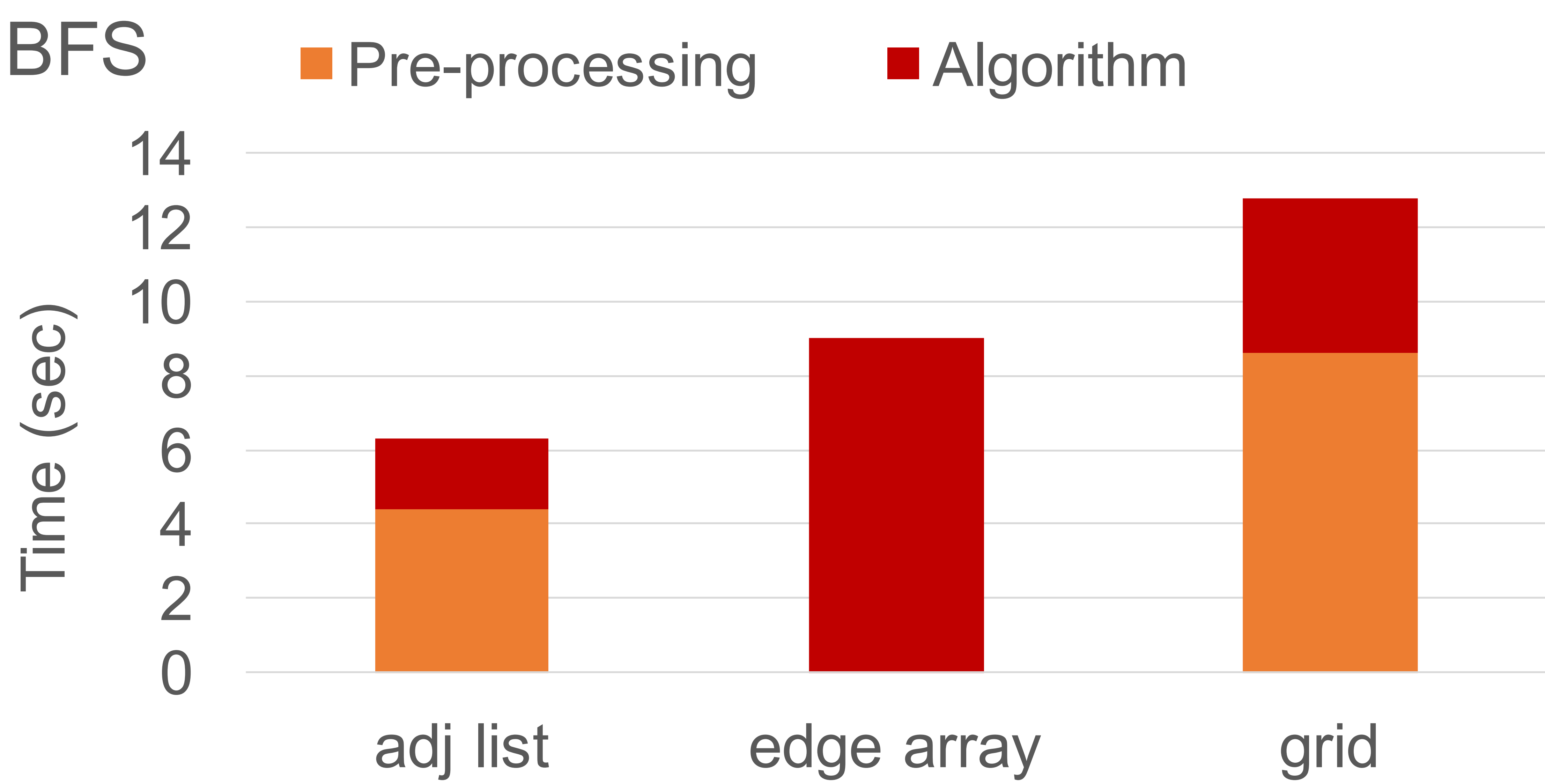


Evaluation: cache-optimization (BFS)



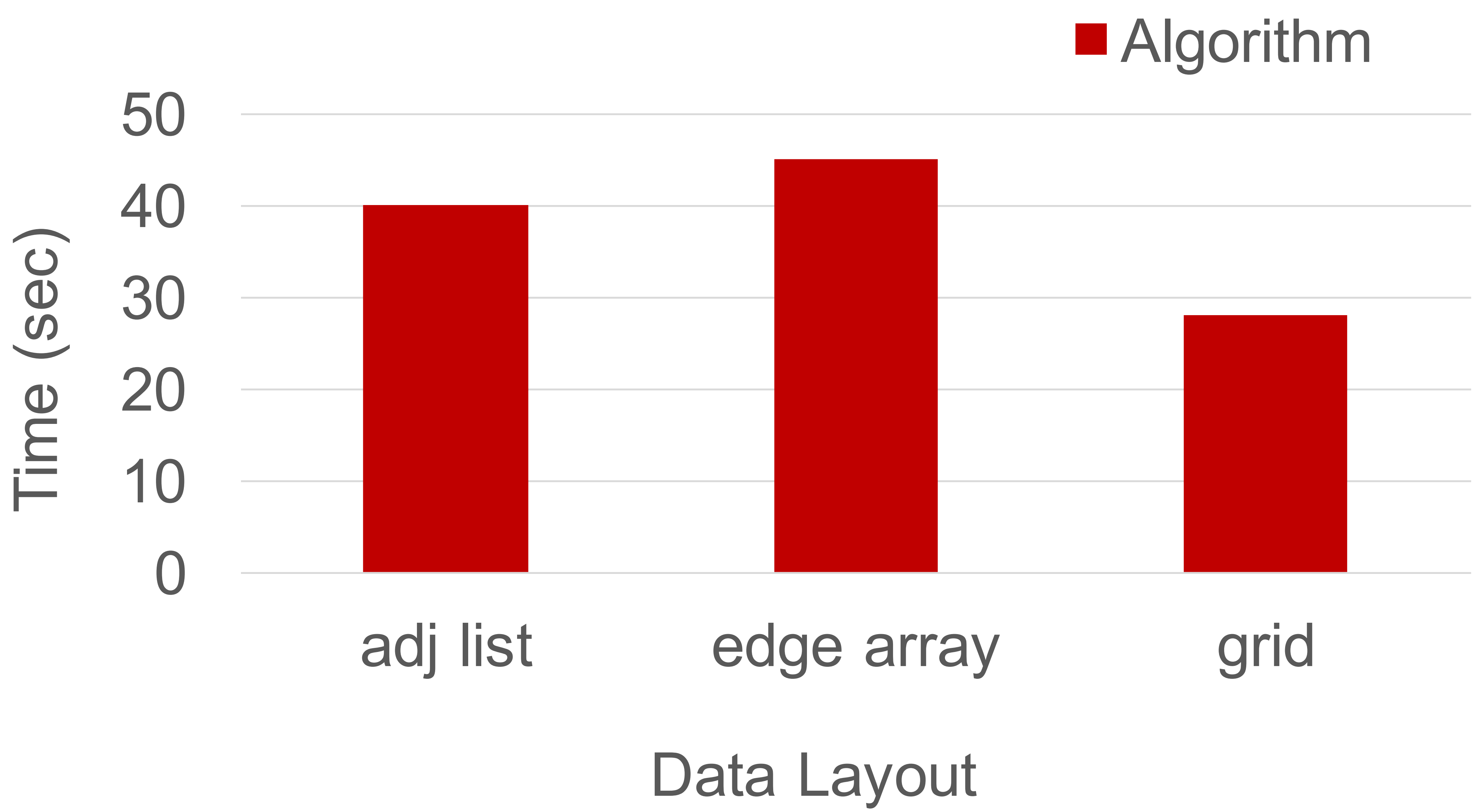
Evaluation: cache-optimization (BFS)

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

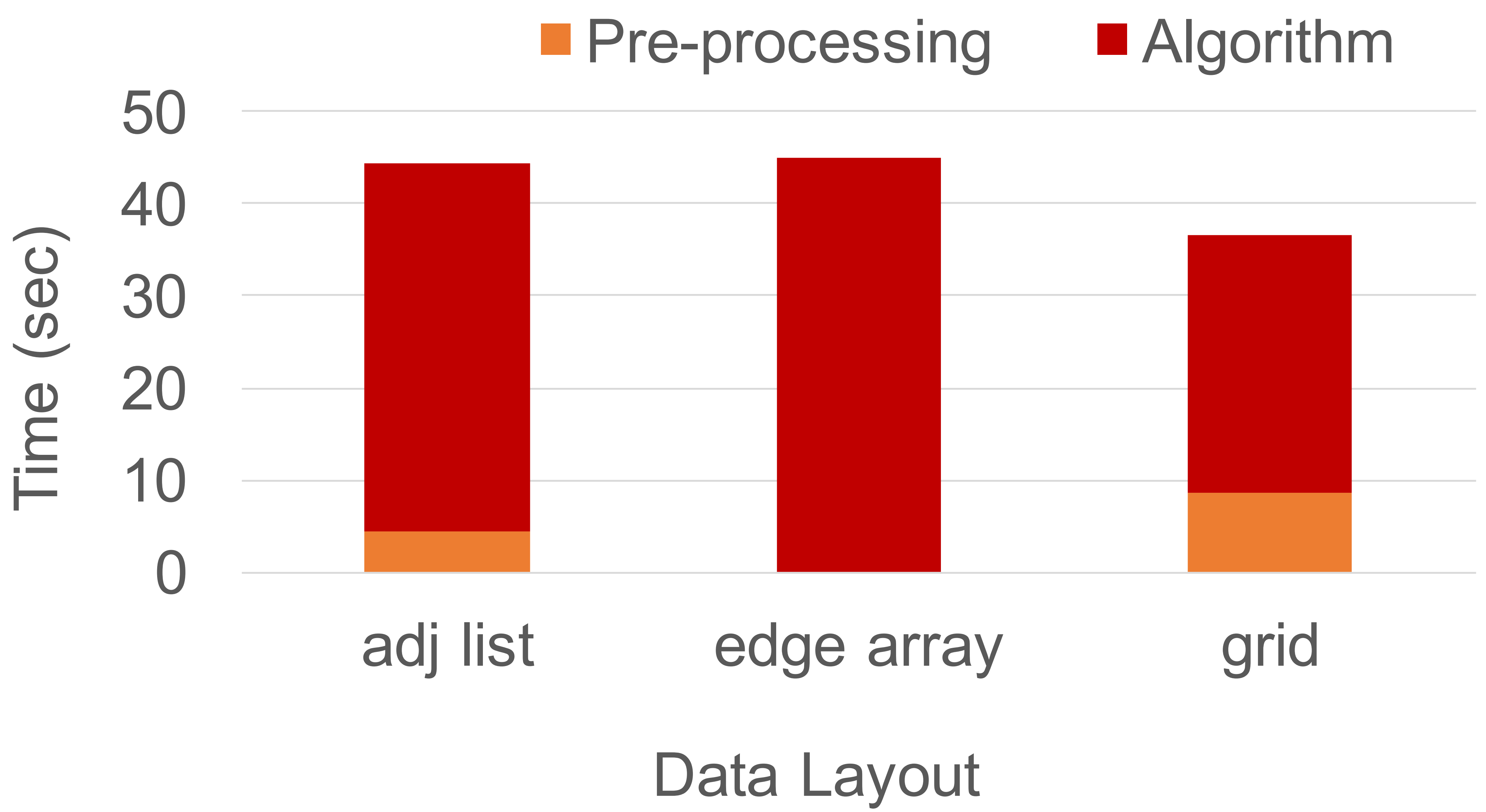


Adjacency lists have the best performance on BFS.

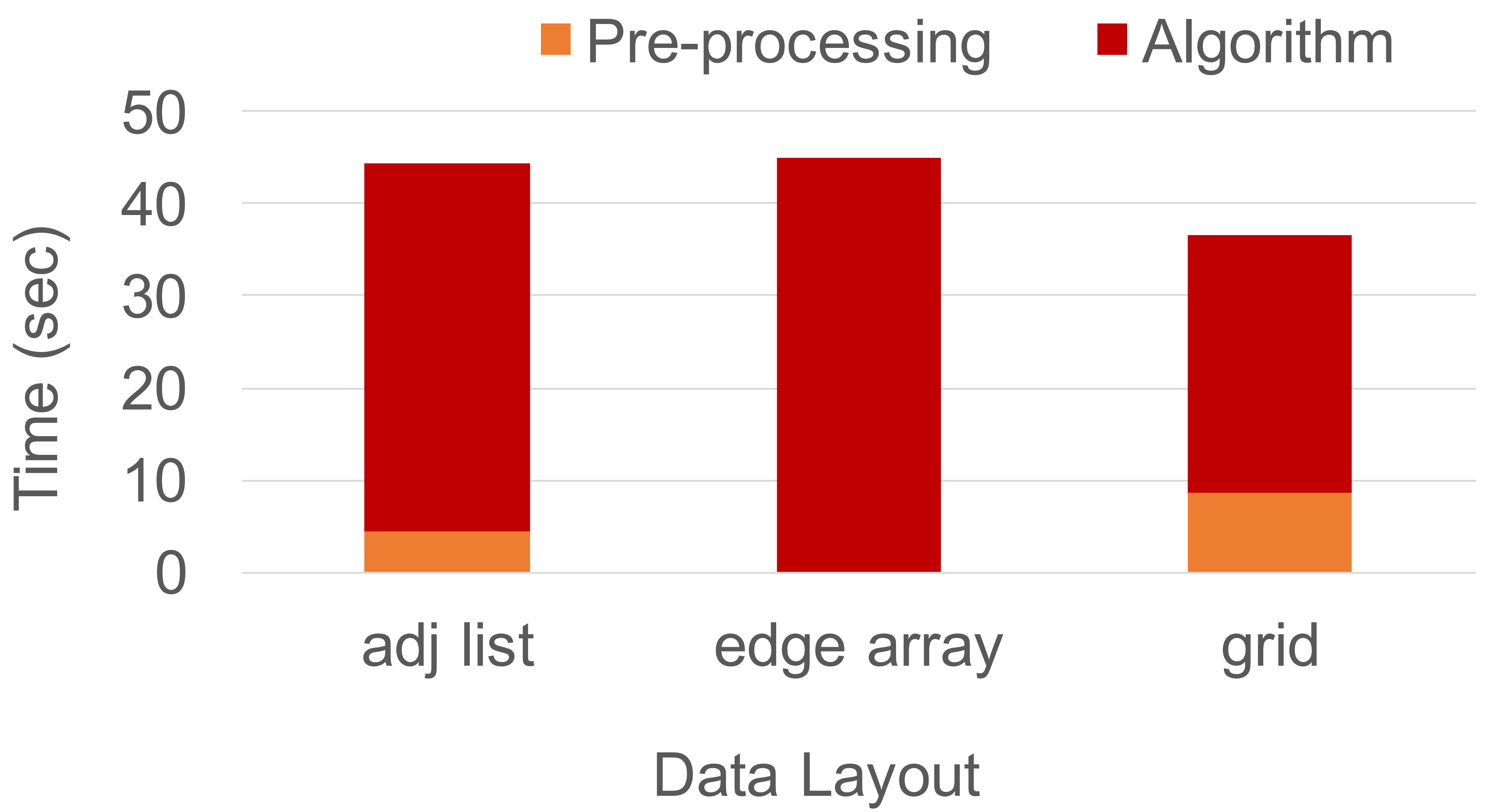
Evaluation: cache-optimization (PageRank)



Evaluation: cache-optimization (PageRank)



Evaluation: cache-optimization (PageRank)



For Pagerank, the grid is the winning approach.

Questions we want to answer:

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

Pre-processing

- ✓ How to represent the graph?
  - Adjacency lists
  - Edge arrays
- ✓ Cost of creating the representation?
  - Radix sort wins for adjacency lists
- ✓ What data layout is best?
  - BFS: Adj. list
  - PR: Grid
  - SpMV: Edge array

Algorithm

- ✓ Can we improve cache locality?
  - Yes. By laying out the edges in a grid format
  - BFS: Adj. list
  - PR: Grid
  - SpMV: Edge array
- **Should we optimize for NUMA?**
- Information flow: **push**, **pull** or a **both**?

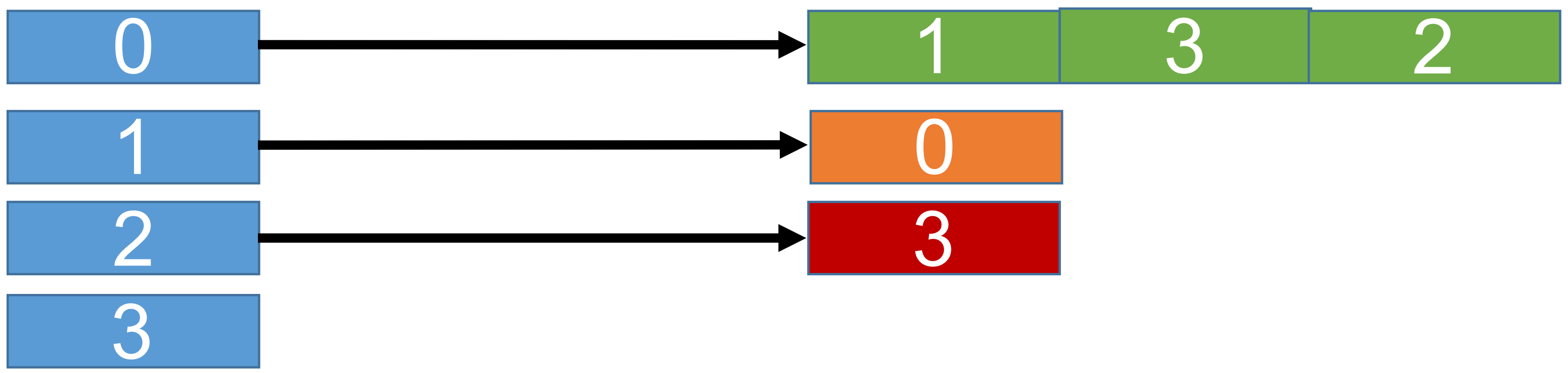
NUMA-Aware optimizations

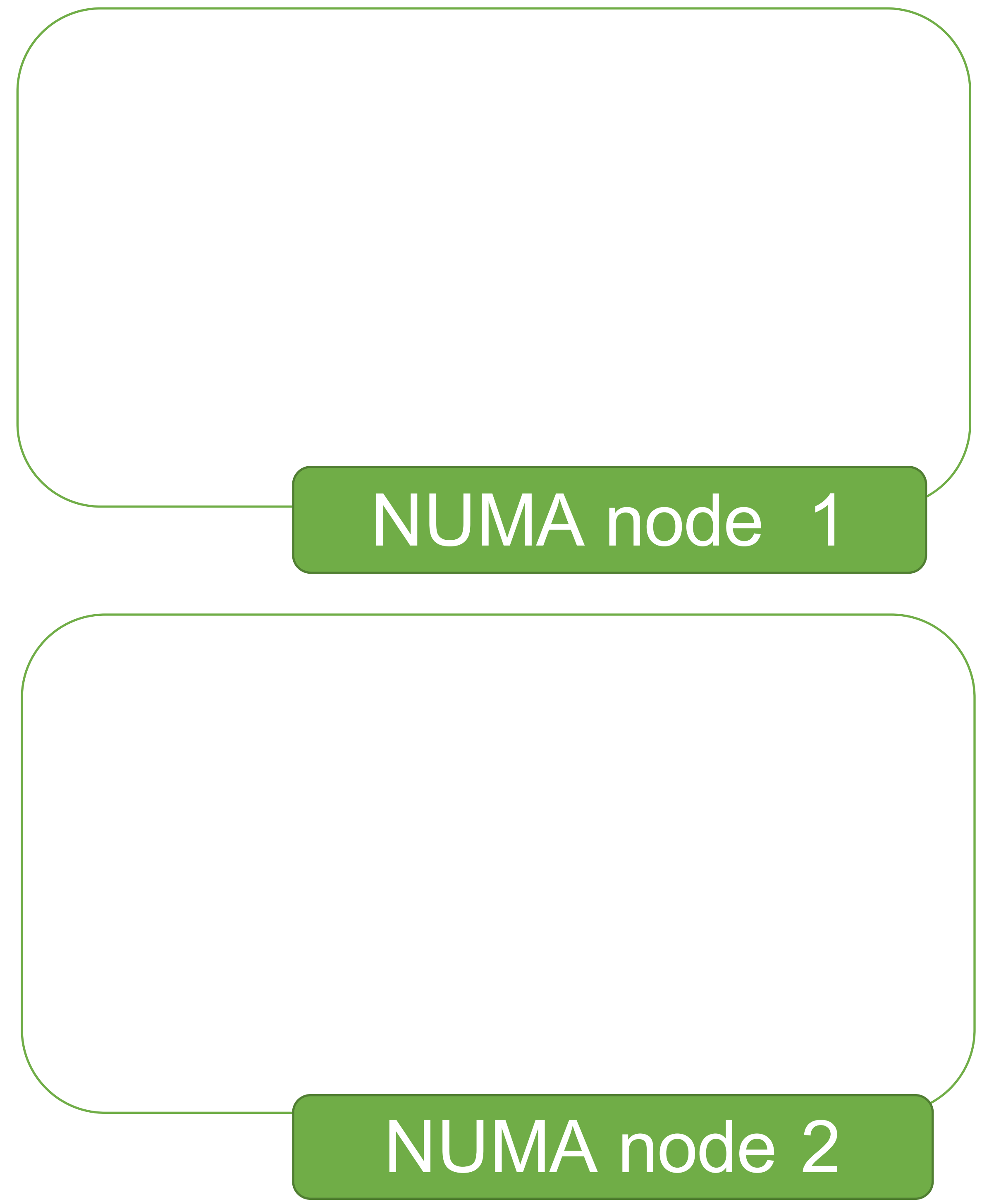
- NUMA-Aware data placement
  - Additional partitioning step in the pre-processing phase

- NUMA-Aware computation
  - Threads compute on local data

- Evaluation environment
  - Machine A: 2 NUMA nodes, 128GB DRAM, 16 Cores
  - Machine B: 4 NUMA nodes, 256GB DRAM, 32 Cores

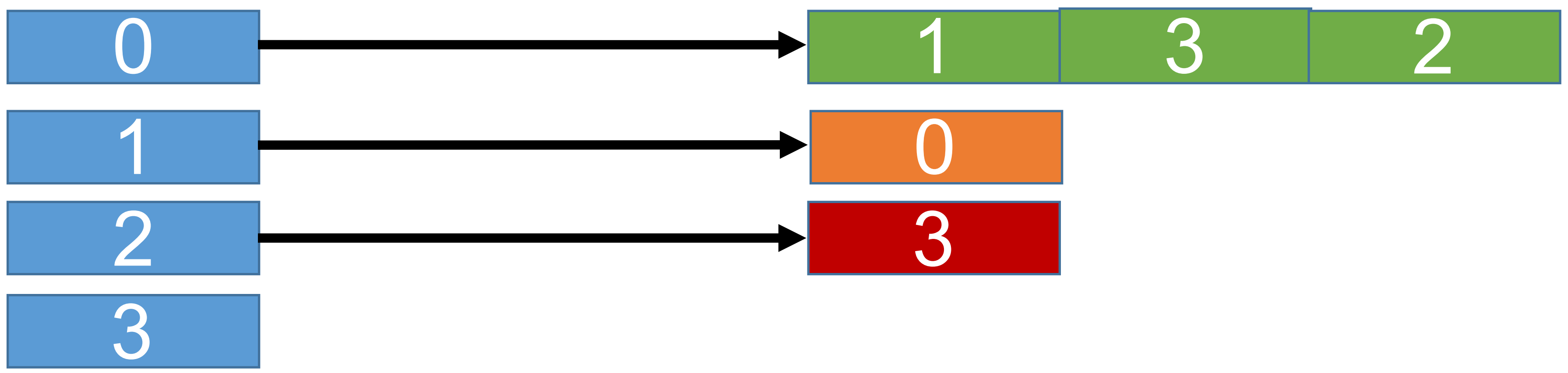
NUMA-Aware data placement

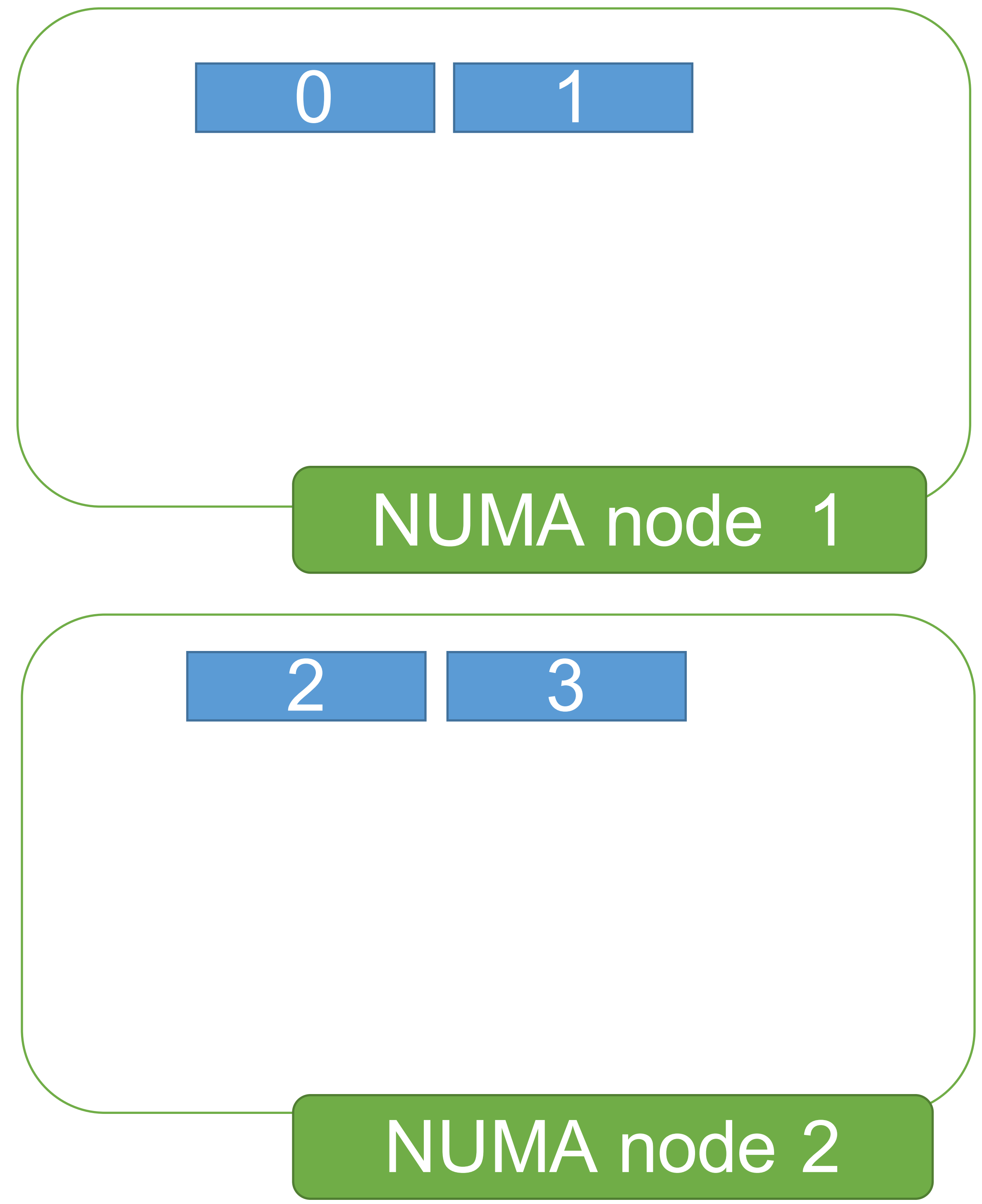




- Vertices spread across NUMA nodes
- Edges collocated with their destination vertex

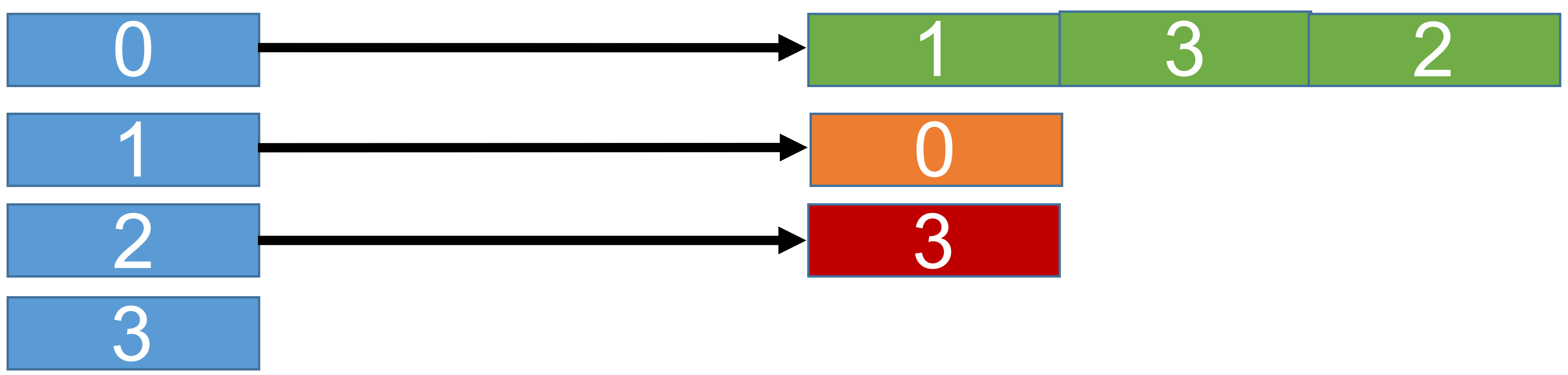
NUMA-Aware data placement

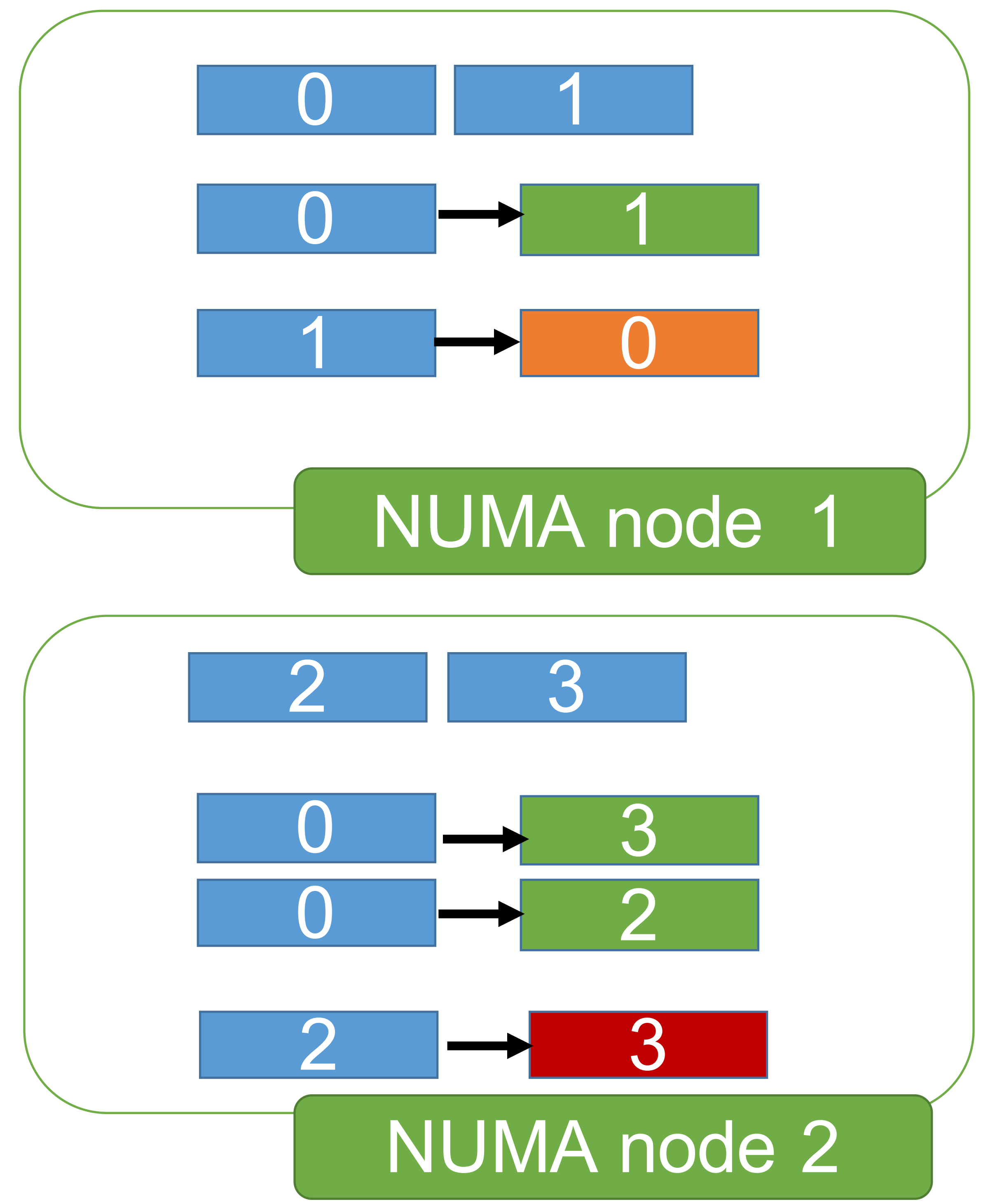




- Vertices spread across NUMA nodes
- Edges collocated with their destination vertex

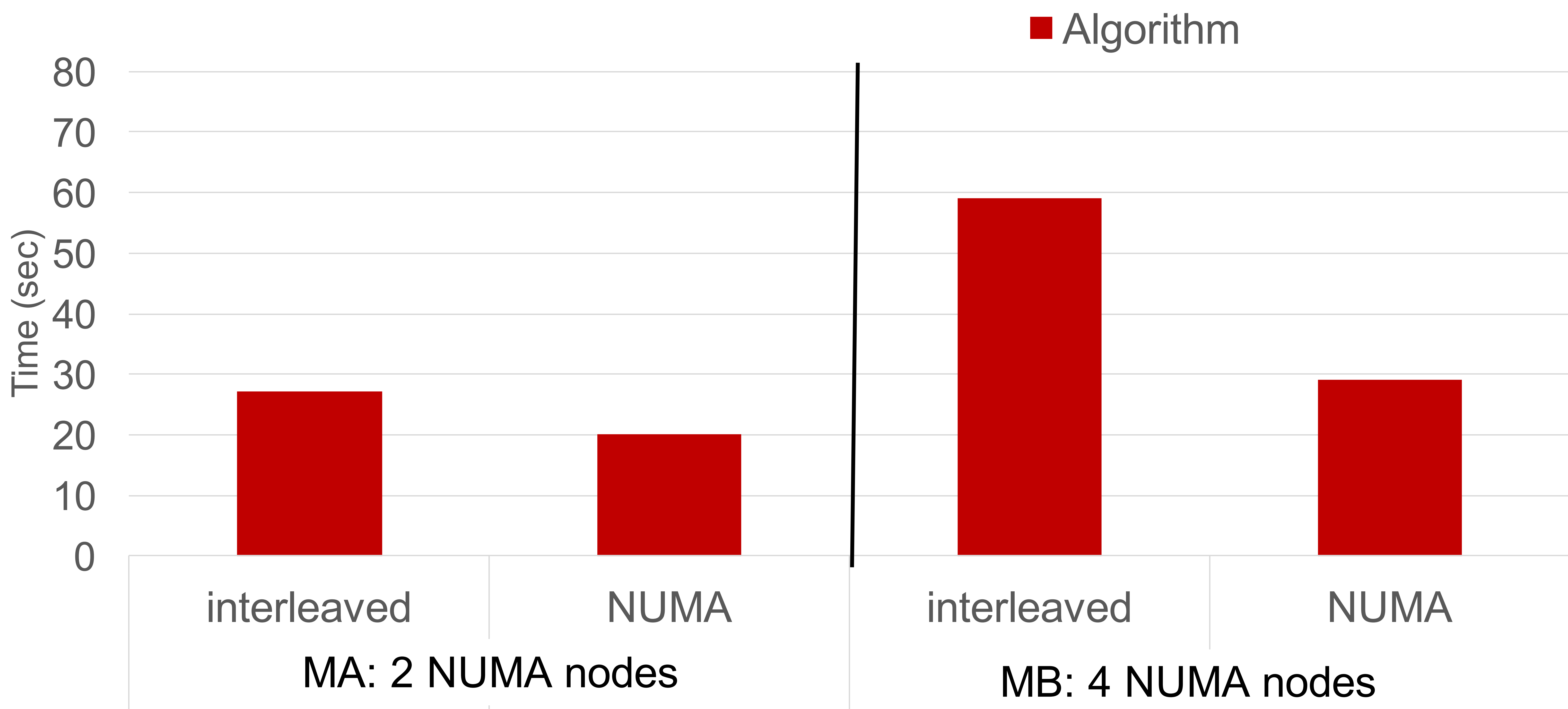
NUMA-Aware data placement



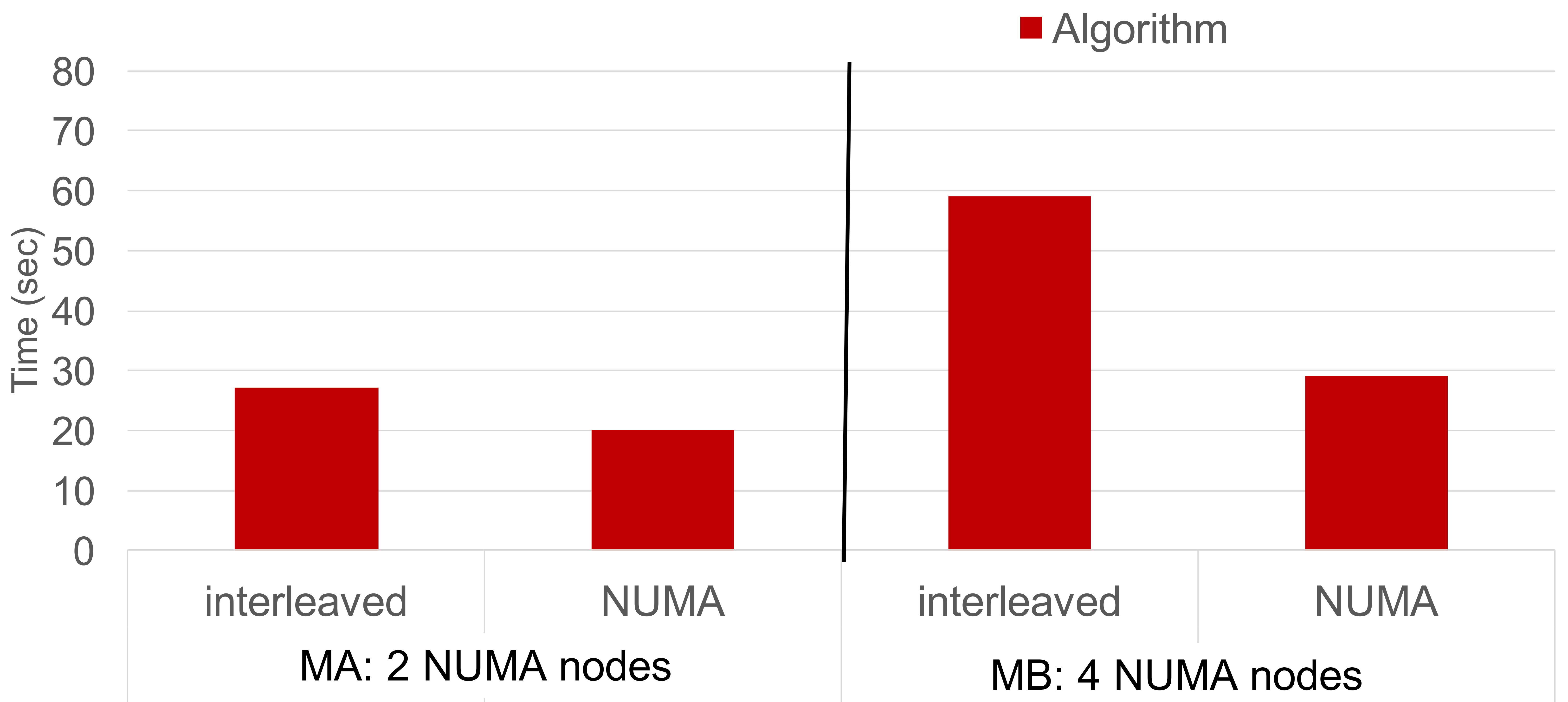


- Vertices spread across NUMA nodes
- Edges collocated with their destination vertex

PageRank

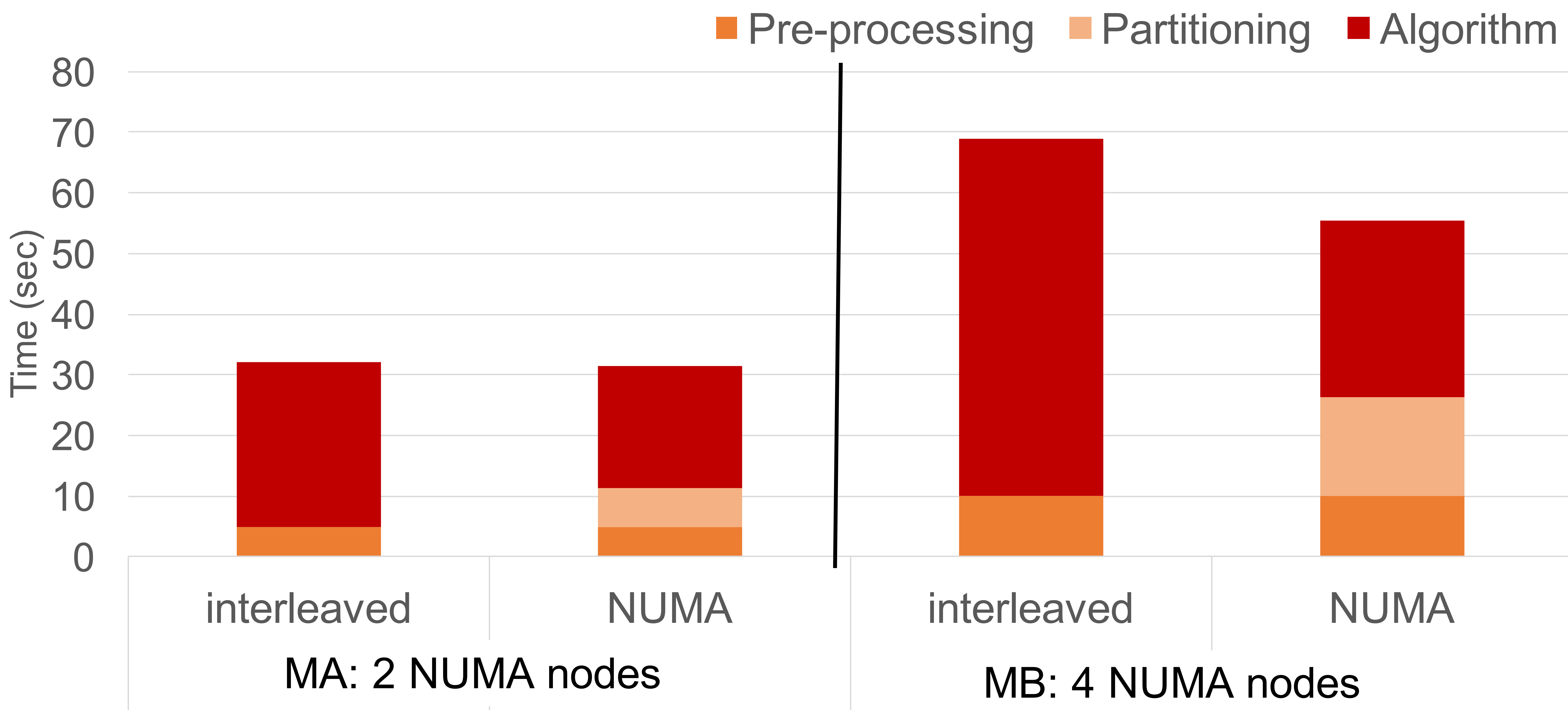

Algorithm
Time (sec)
80
70
60
50
40
30
20
10
0
interleaved
NUMA
interleaved
NUMA
MA: 2 NUMA nodes
MB: 4 NUMA nodes

PageRank



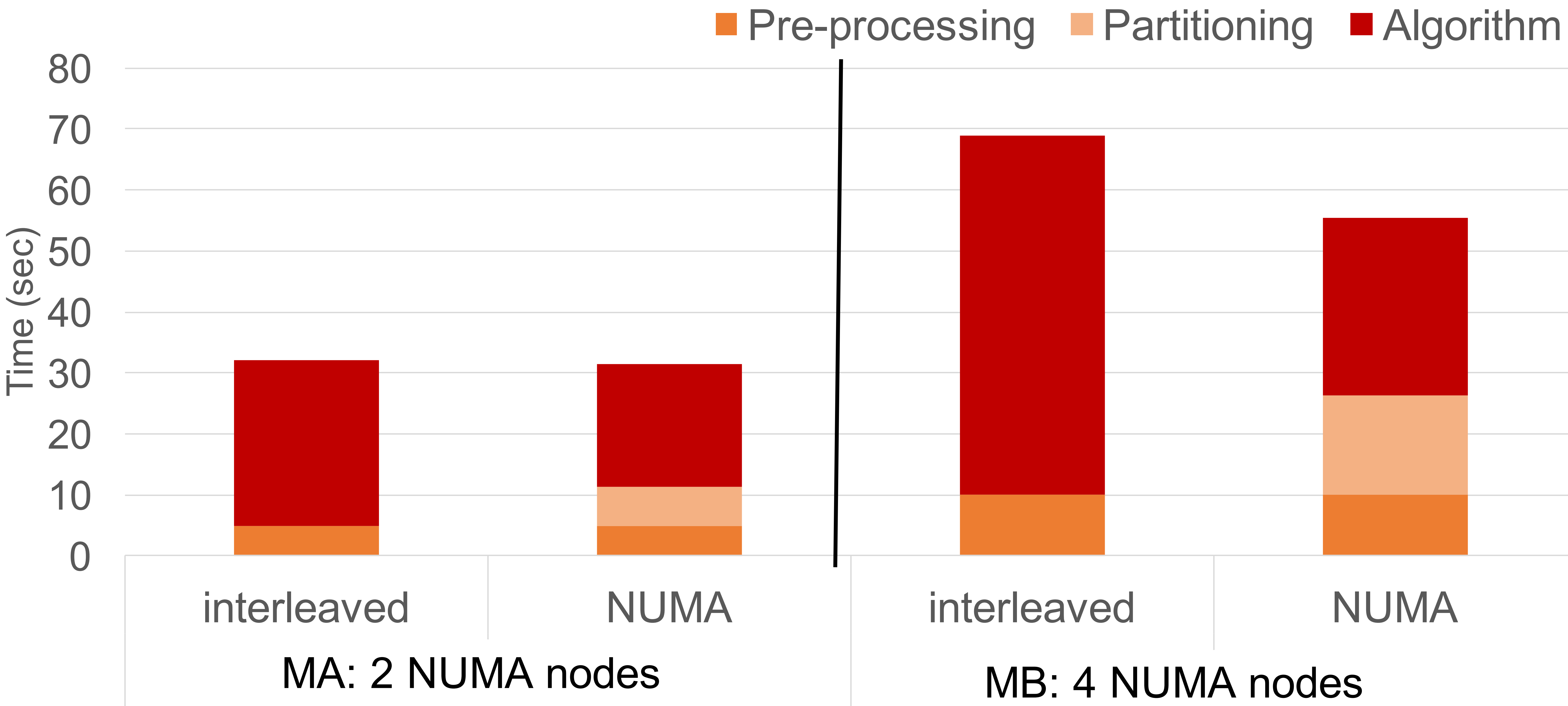
27% - 50% Improved compute time

PageRank



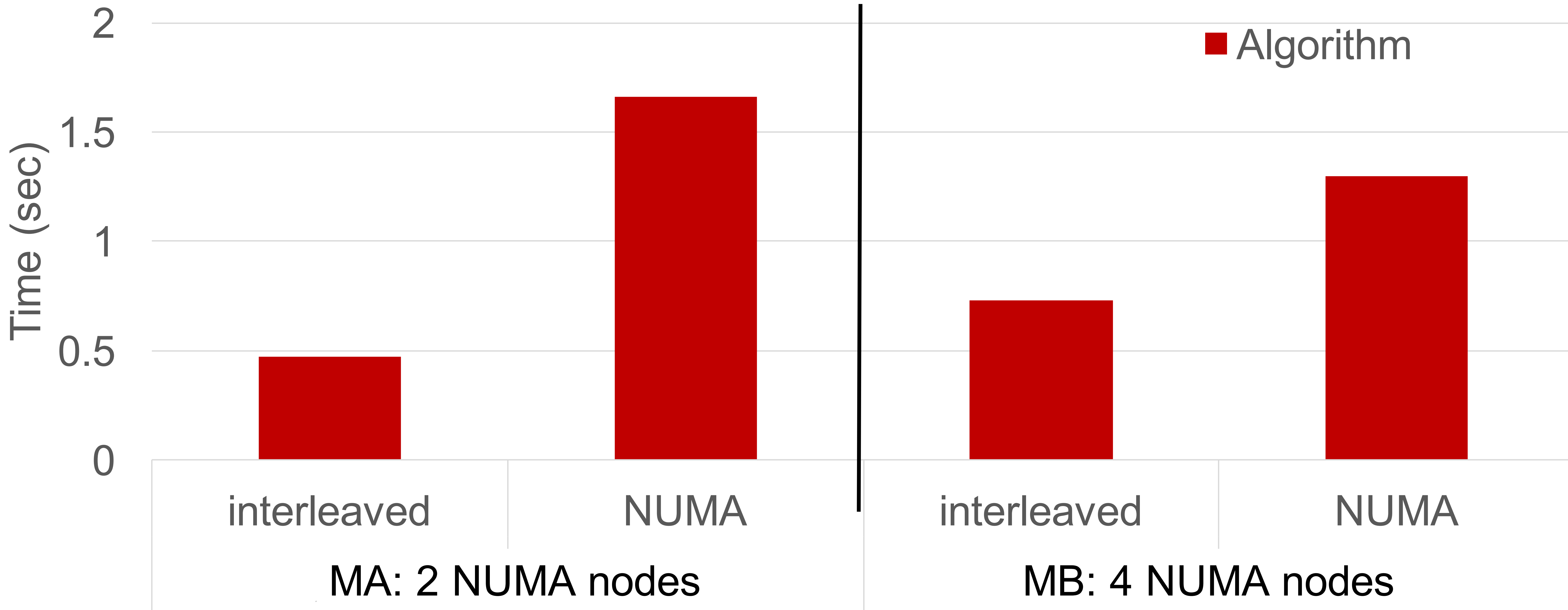
Pre-processing
Partitioning
Algorithm
Time (sec)
0
10
20
30
40
50
60
70
80
interleaved
NUMA
interleaved
NUMA
MA: 2 NUMA nodes
MB: 4 NUMA nodes

PageRank



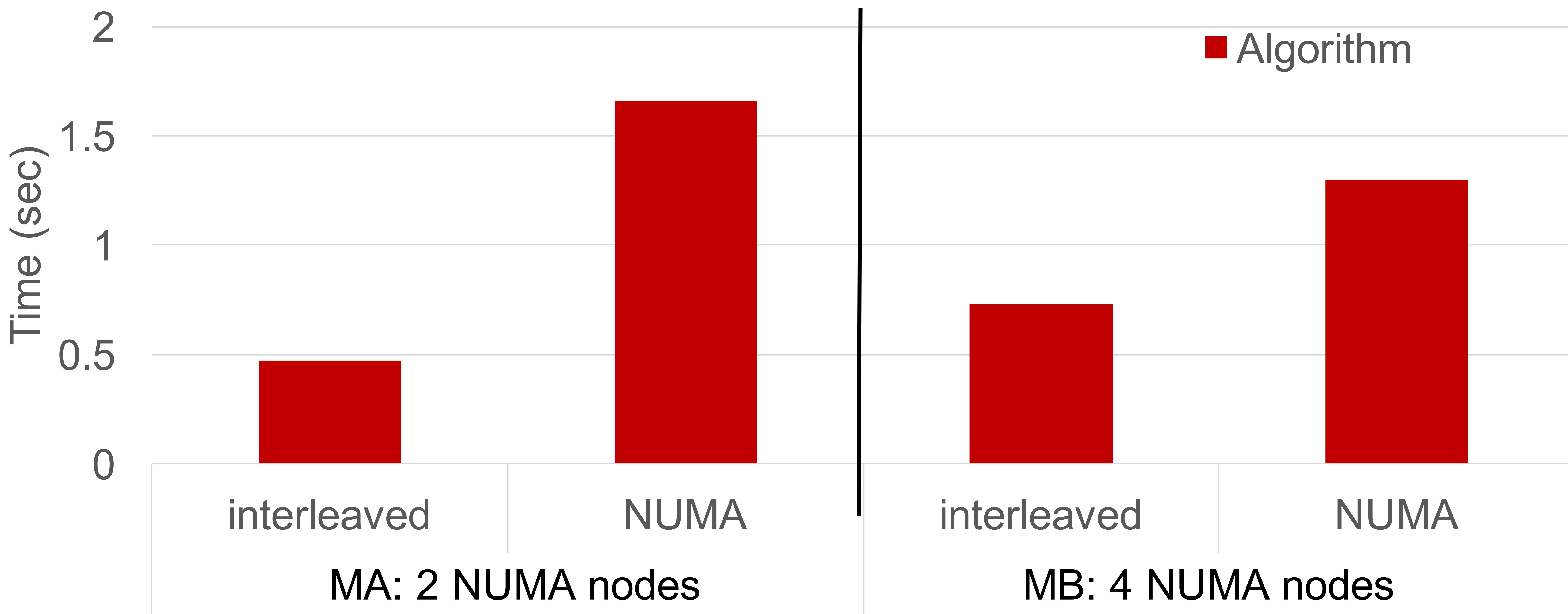
Pre-processing amortized only on Machine B

BFS



Algorithm
Time (sec)
2
1.5
1
0.5
0
interleaved
NUMA
interleaved
NUMA
MA: 2 NUMA nodes
MB: 4 NUMA nodes

BFS



No gain in algorithm time, contention on memory bus

Questions we want to answer:

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

Pre-processing

- ✓ How to represent the graph?
  - ○ Adjacency lists ○ Edge arrays
- ✓ Cost of creating the representation?
  - ○ Radix sort wins for adjacency lists
- ✓ What data layout is best?
  - ○ BFS: Adj. list ○ PR: Grid ○ SpMV: Edge array

Algorithm

- ✓ Can we improve cache locality?
  - ○ Yes. By laying out the edges in a grid format
  - ○ BFS: Adj. list ○ PR: Grid ○ SpMV: Edge array
- ✓ Should we optimize for NUMA?
  - ○ Can pay off only on big machines
  - ○ BFS & SpMV: No gain ○ PR: NUMA-optimize
- **Information flow: push, pull or a both?**

Information flow

- Push
  - You **push** information to your neighbors
  - You need **outgoing edges**

- Pull
  - You **pull** information from your neighbors
  - You need **incoming** edges

Which one is better?

- Push
  - You **push** information to your neighbors - write to state of others
- Pull
  - You **pull** information from your neighbors – write to own state

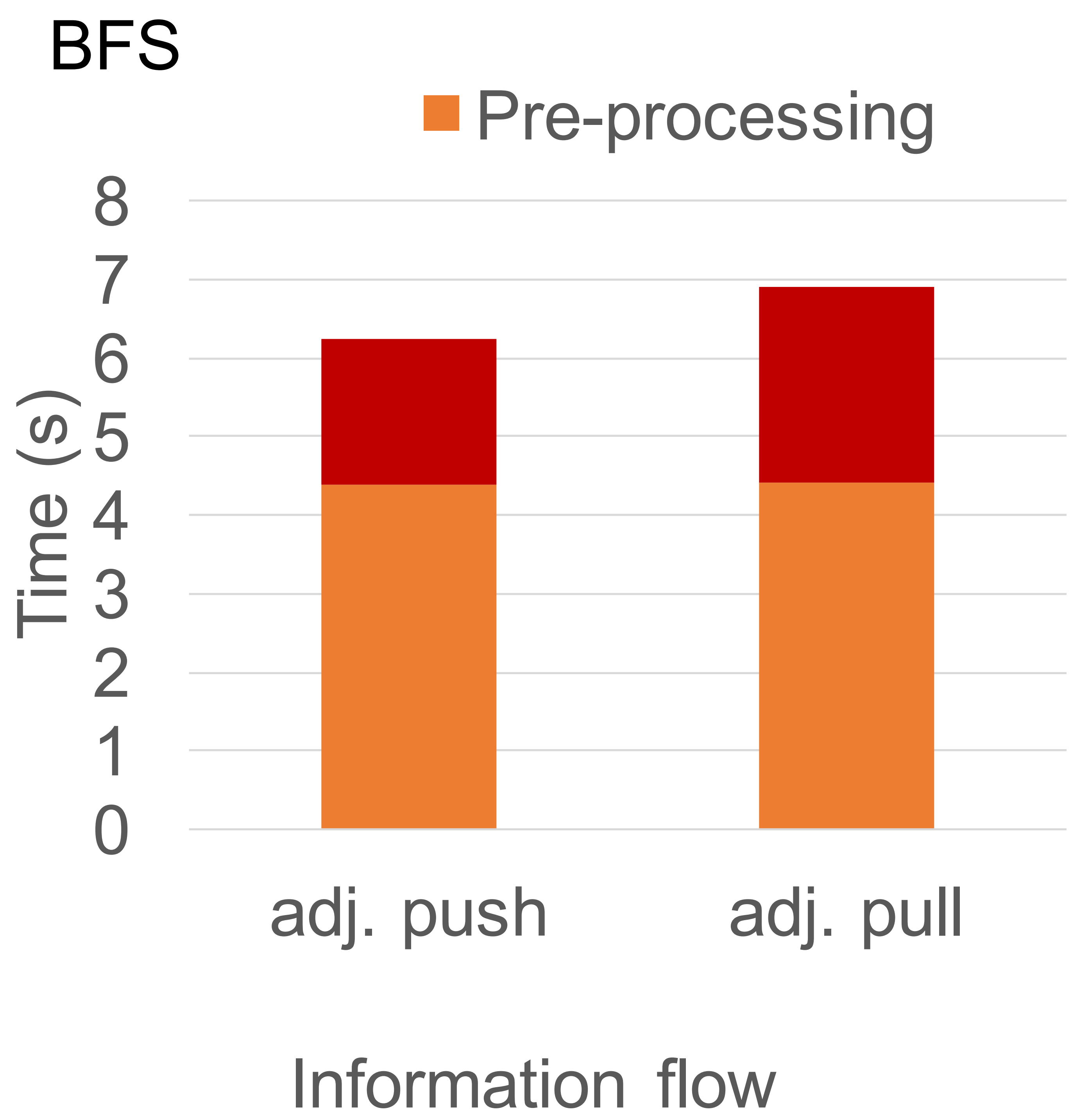
Which one is better?

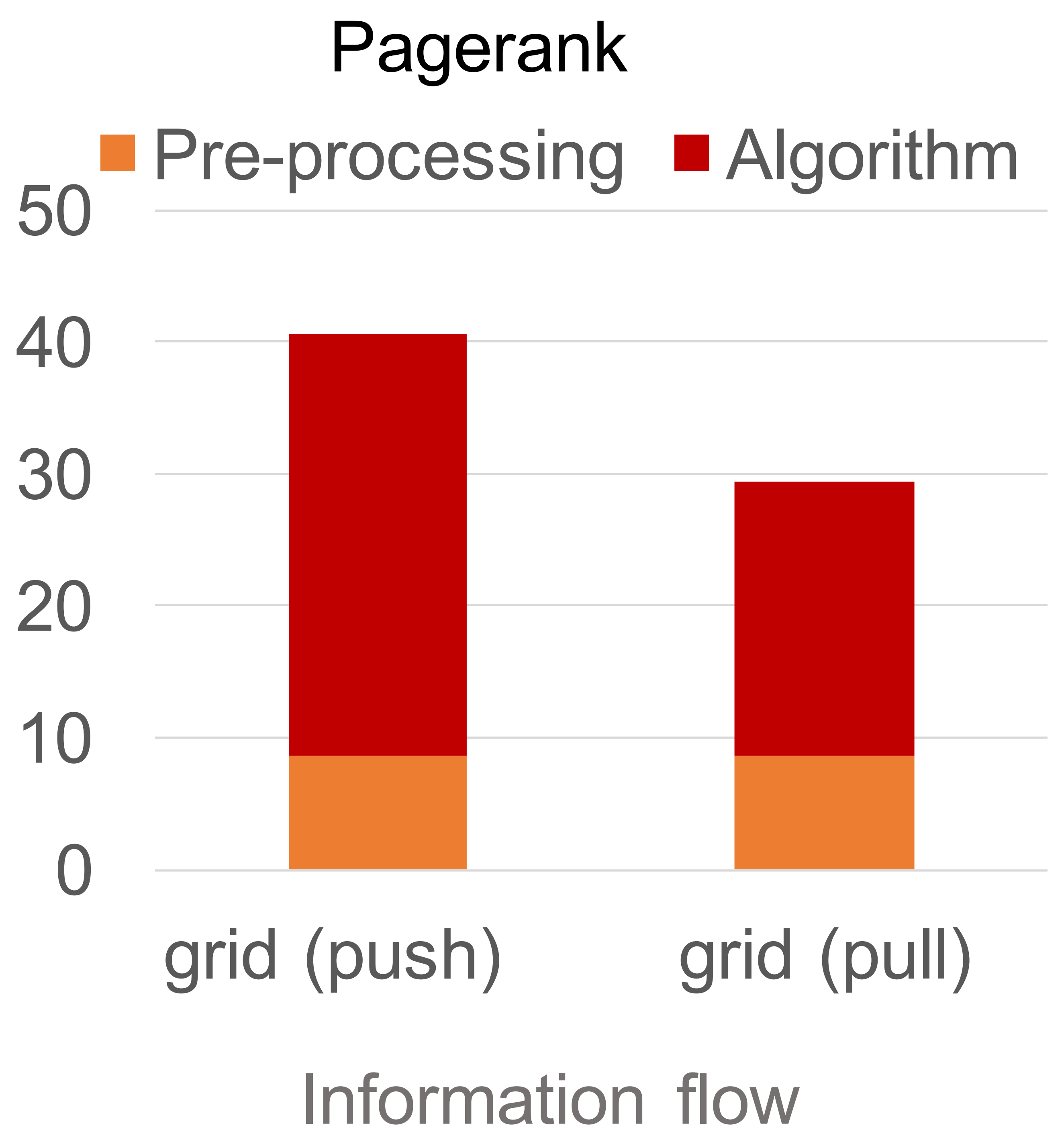
ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

- Push
  - You **push** information to your neighbors - write to state of others
  - ✓Good when few vertices are active
  - x Needs locks

- Pull
  - You **pull** information from your neighbors – write to own state
  - ✓Good when many vertices are active
  - ✓Locks can be avoided

PUSH vs. PULL – BFS & PR





Questions we want to answer:

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

Pre-processing

- ✓ How to represent the graph?
  - Adjacency lists
  - Edge arrays
- ✓ Cost of creating the representation?
  - Radix sort wins for adjacency lists
- ✓ What data layout is best?
  - BFS: Adj. list
  - PR: Grid
  - SpMV: Edge array

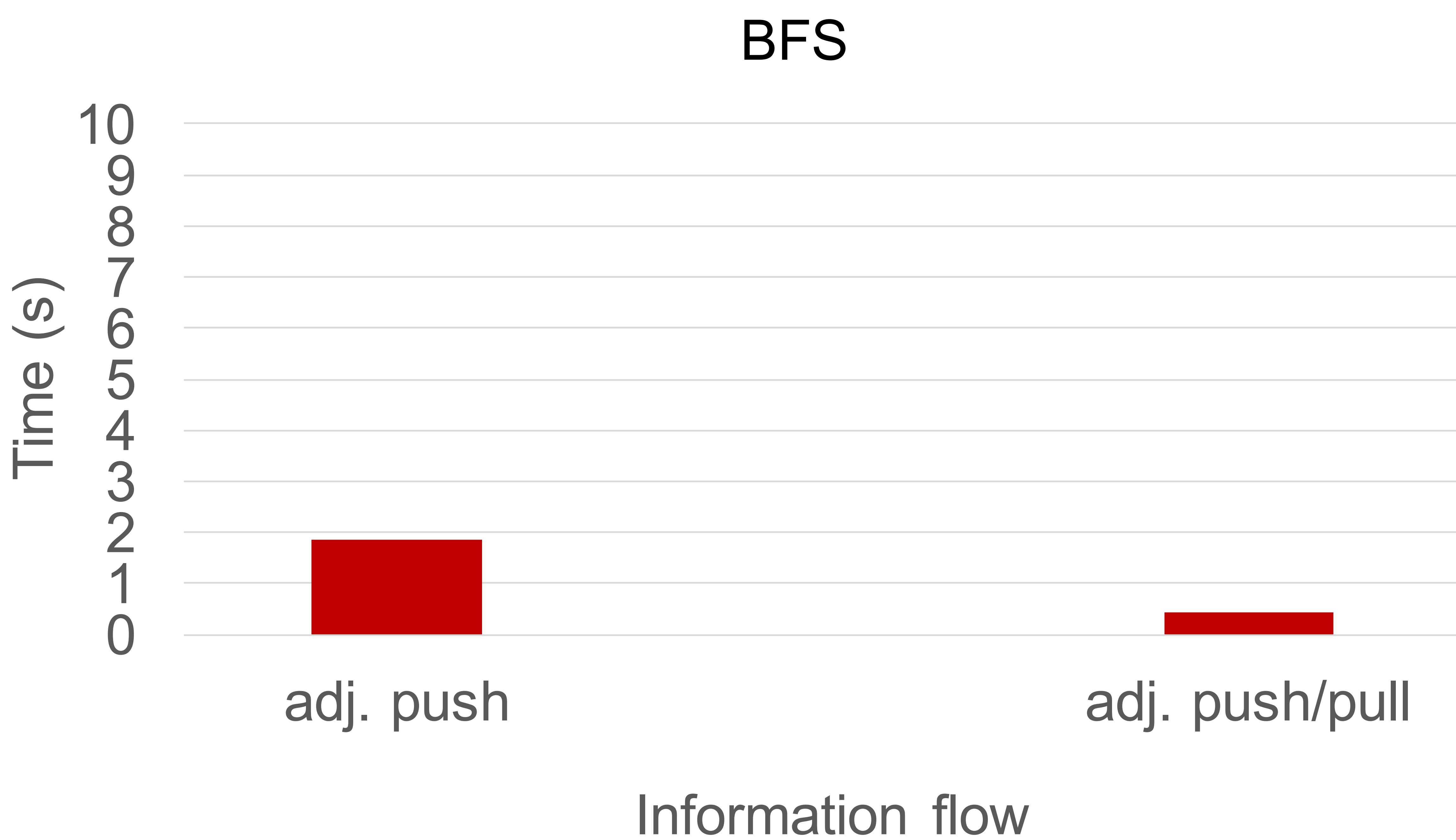
Algorithm

- ✓ Can we improve cache locality?
  - Yes. By laying out the edges in a grid format
  - BFS: Adj. list
  - PR: Grid
  - SpMV: Edge array
- ✓ Should we optimize for NUMA?
  - Can pay off only on big machines
  - BFS & SpMV: No gain
  - PR: NUMA-optimize
- Information flow: **push**, **pull** or a **both**?
  - Less synchronization not always a win
  - BFS: Push (locks)
  - PR: Pull (no locks)

Push & Pull both win in different situations

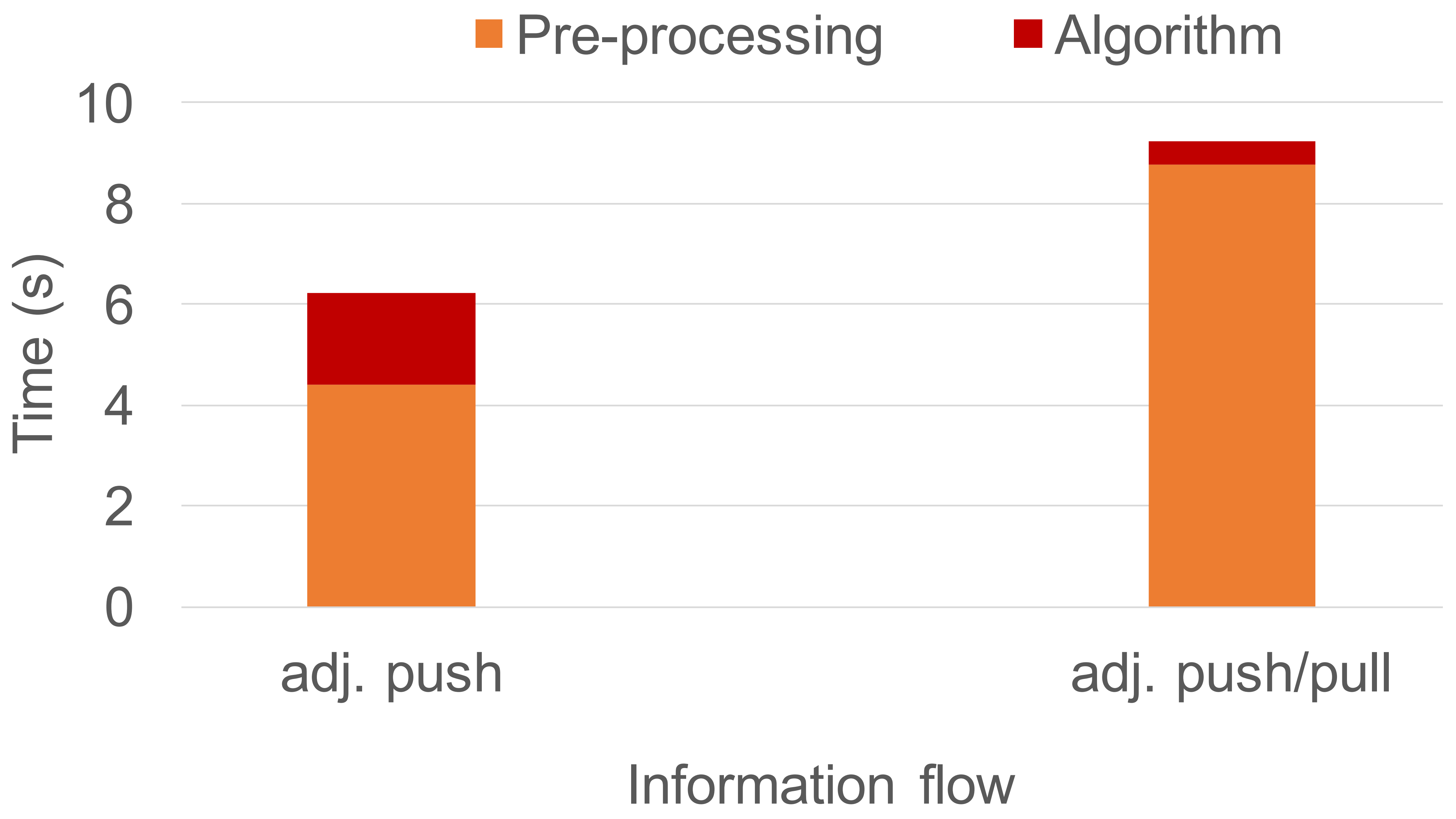
- Combine them
  - Use push when it is efficient
  - Use pull when it is efficient
  - Cost: You need **both**, incoming **and** outgoing edges

Benefit of Push/Pull



Benefit of Push/Pull

BFS



Questions we want to answer:

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

Pre-processing

- ✓ How to represent the graph?
  - Adjacency lists
  - Edge arrays
- ✓ Cost of creating the representation?
  - Radix sort wins for adjacency lists
- ✓ What data layout is best?
  - BFS: Adj. list
  - PR: Grid
  - SpMV: Edge array

Algorithms

- ✓ Can we improve cache locality?
  - Yes. By laying out the edges in a grid format
  - BFS: Adj. list
  - PR: Grid
  - SpMV: Edge array
- ✓ Should we optimize for NUMA?
  - Can pay off only on big machines
  - BFS & SpMV: No gain
  - PR: NUMA-optimize
- ✓ Information flow: **push**, **pull** or a **both**?
  - Less synchronization not always a win
  - BFS: Push (locks)
  - PR: Pull (no locks)
  - Push/Pull no win in end-to-end (directed graphs)

Additional results in the paper

- Scalability of pre-processing approaches
- Relation between pre-processing and loading from HDD and SSD
- Results on other algorithms
- Results for different graph types

Systems that motivated the paper

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

System	Data Layout	Iteration Model	Push or Pull	NUMA-Aware
Ligra [PPoPP '13]	Adj. List	Vertex-centric	Push & Pull	-
Polymer [PPoPP '15]	Adj. List	Vertex-centric	Push & Pull	✔
Gemini [OSDI'16]	Adj. List	Vertex-centric	Push & Pull	✔
X-Stream [SOSP'13]	Edge Array	Edge-centric	Push	-
GridGraph [ATC '15]	Grid	Grid-cell	Push	-

Summary

Pre-processing

- Edge arrays
- Adjacency lists
- Sorting techniques

Algorithm time

- Cache-optimizations
- Push vs. Pull
- Synchronization
- NUMA-aware computation

Conclusion

- **Improvement in computation is not free**
- **Trade-off between added pre-processing time and algorithm time**

Whether optimization cost in pre-processing is amortized, depends on algorithm:

- SpMV: Short algorithm and does not benefit from additional optimizations
- BFS: Building adjacency lists
- Pagerank: Optimizing for cache locality (grid) & NUMA-Awareness

Fork us on GitHub: https://github.com/epfl-labos/EverythingGraph.git