Making Pull-Based Graph Processing Performant

Samuel Grossman[1], Heiner Litz[2], and Christos Kozyrakis[1]
[1]Stanford University [2]University of California, Santa Cruz

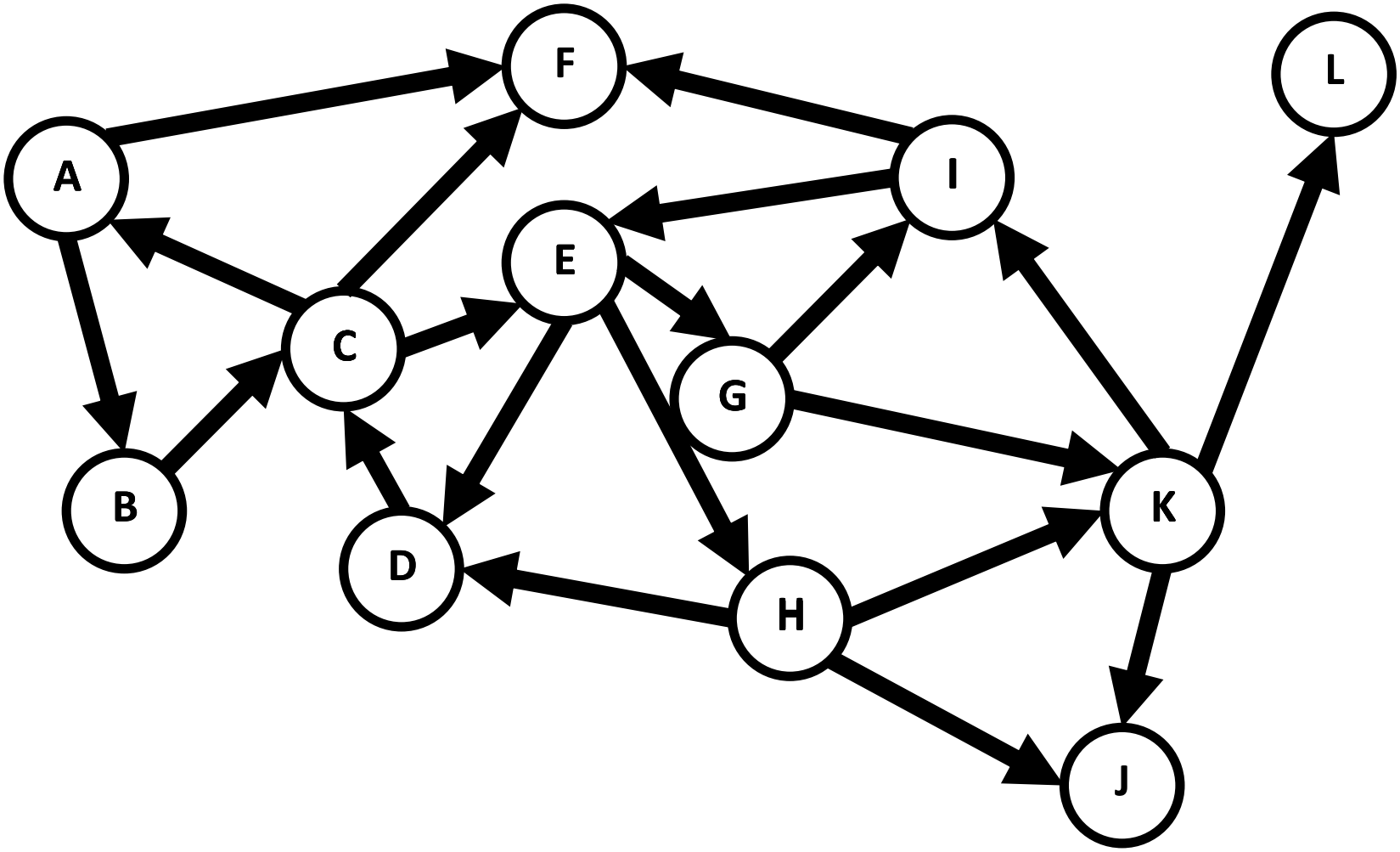
PPoPP 2018 · February 27, 2018

Graph Processing

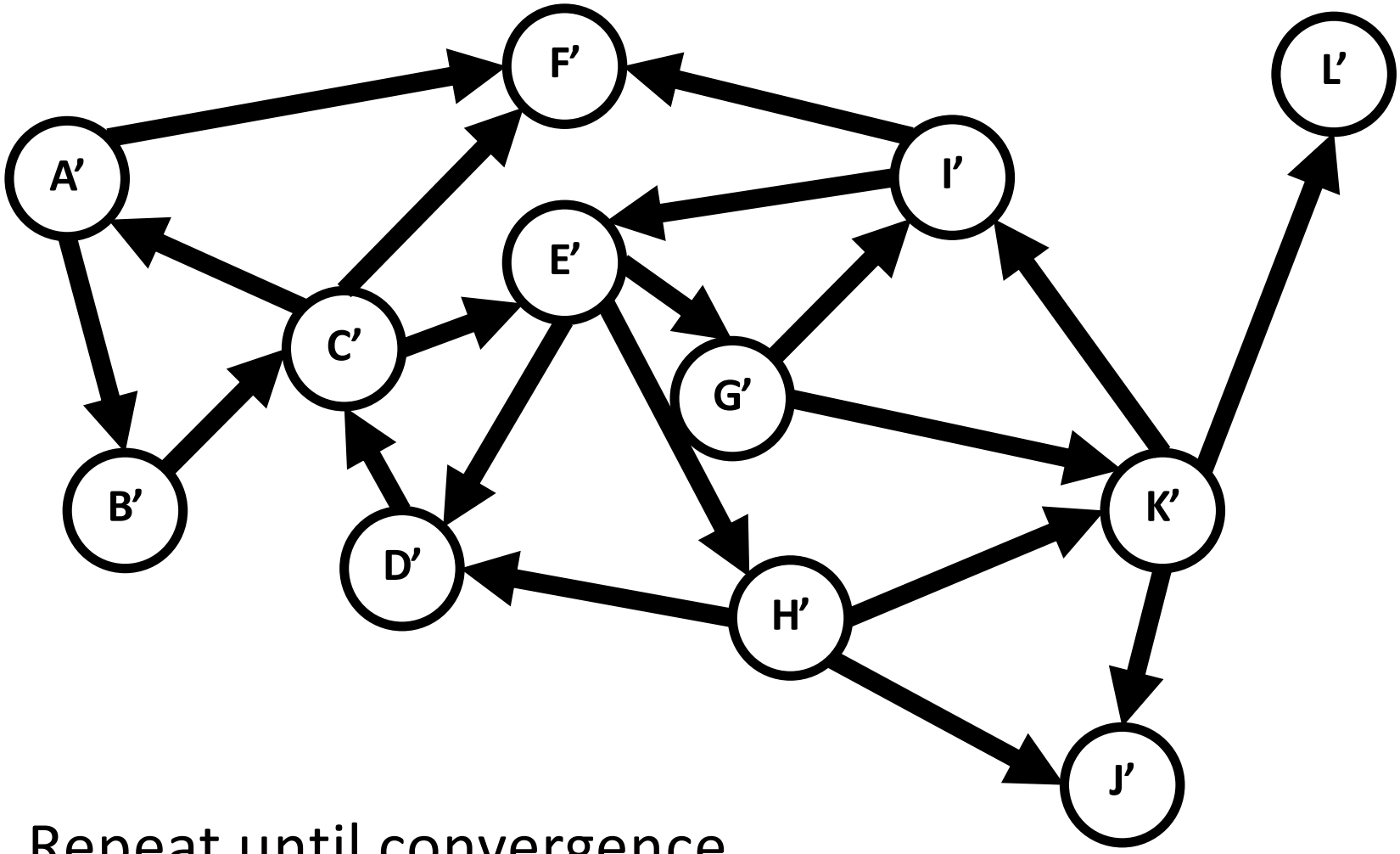
- Problem modelled as **objects** (vertices) and **connections between them** (edges)

- Examples:
  - Internet (pages and hyperlinks)
  - Social network (people and friendships)
  - Roads and intersections
  - Products and ratings

Graph Processing



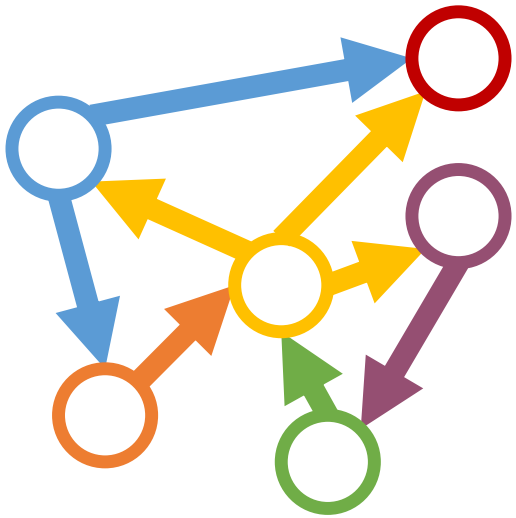
Graph Processing



Repeat until convergence

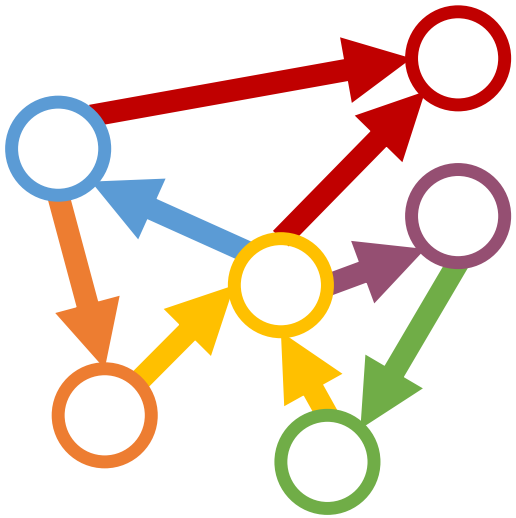
Graph Processing

Push



Group by **source vertex**

Pull



Group by **destination vertex**

Hybrid: dynamically select push or pull for each iteration

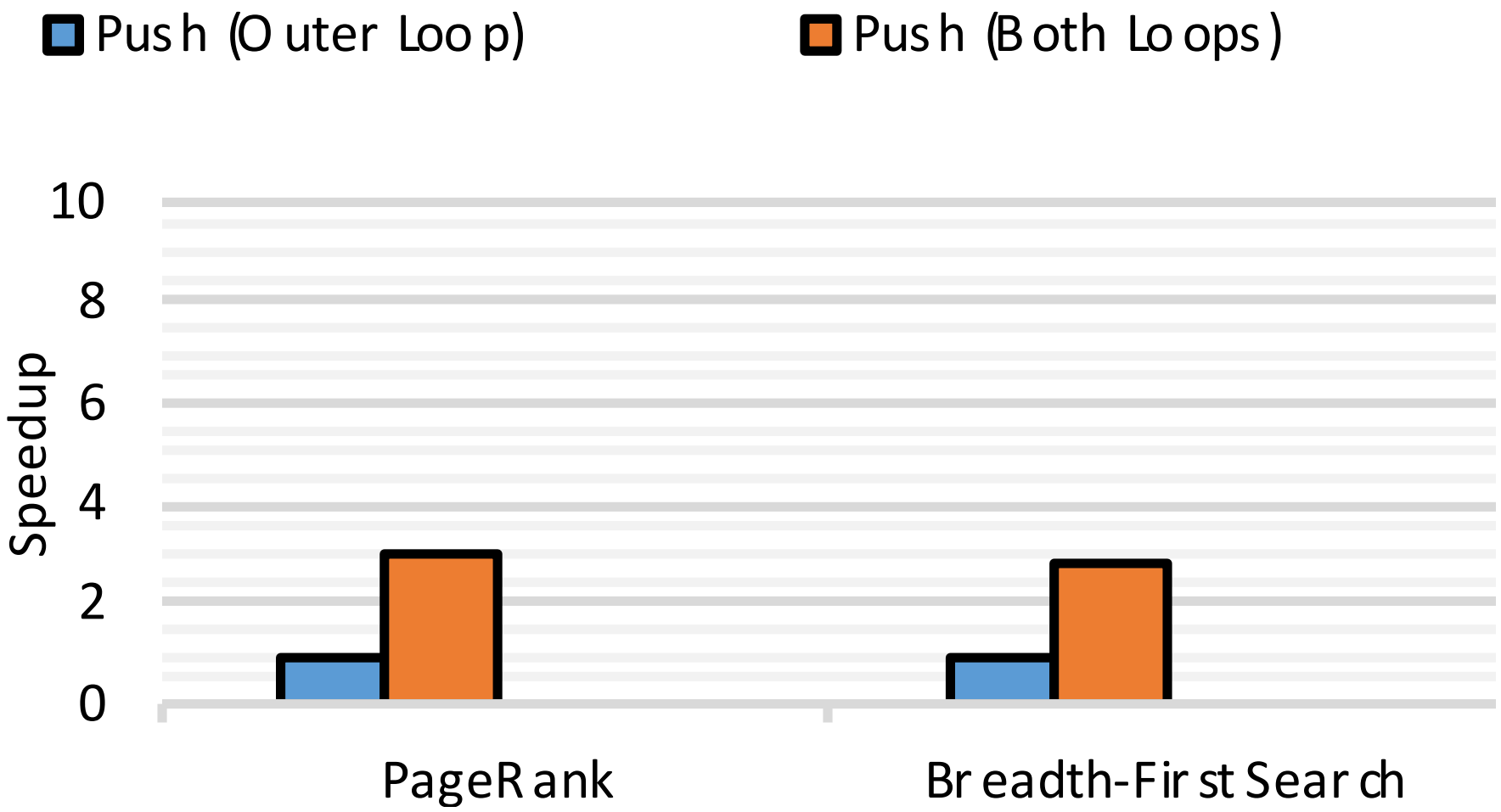
Graph Processing

```
foreach vertex v in graph.vertices
    foreach edge e in v.(in|out)edges
        // process the edge
        ...
```

Parallelizing Graph Processing

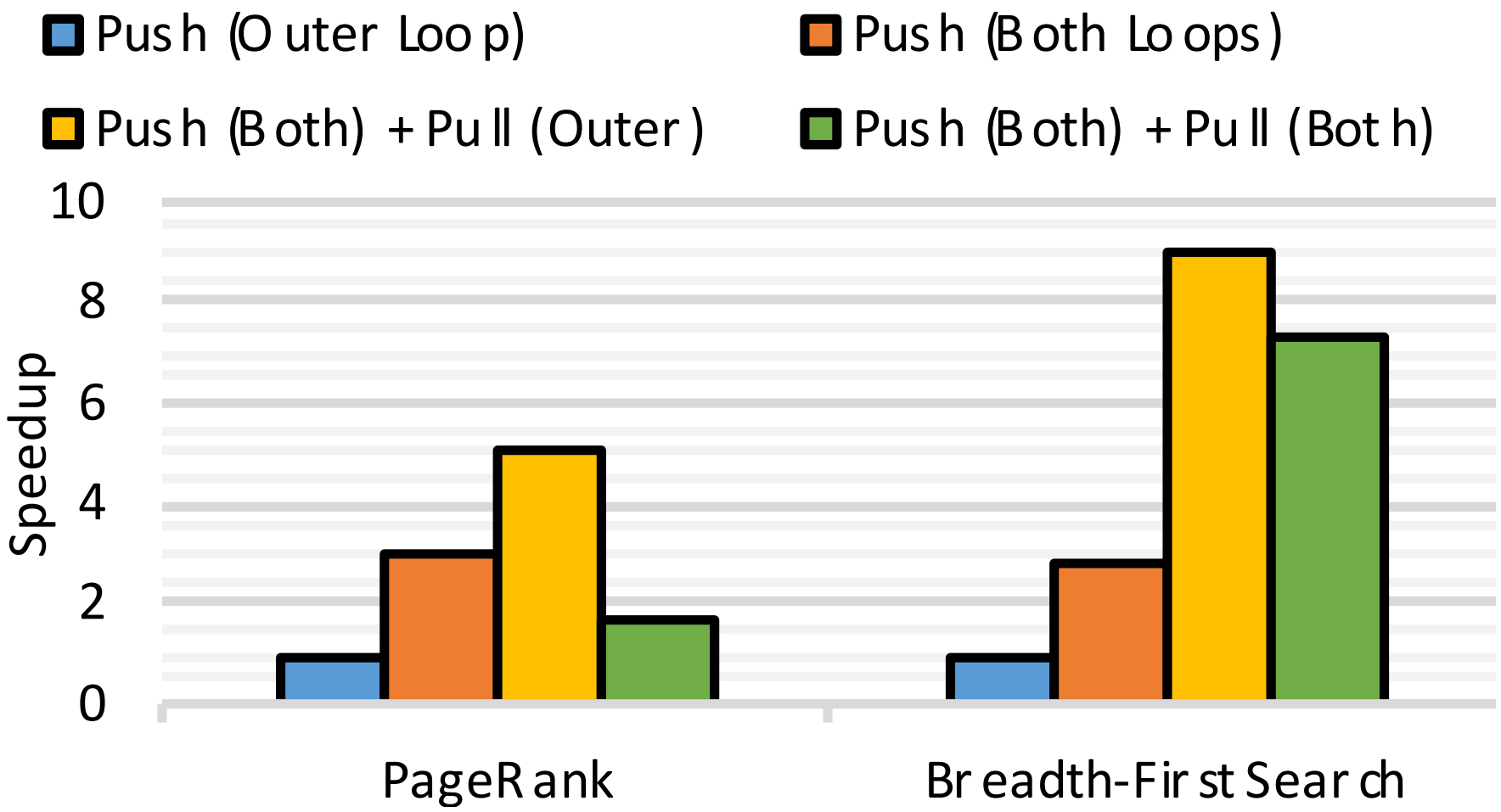
- Outer loop parallelization
  - **Between cores:** assign *entire vertices* to threads
- Inner loop parallelization
  - **Between cores:** subdivide the edges within each vertex
  - **Within one core:** vectorize the loop

Parallelizing Graph Processing



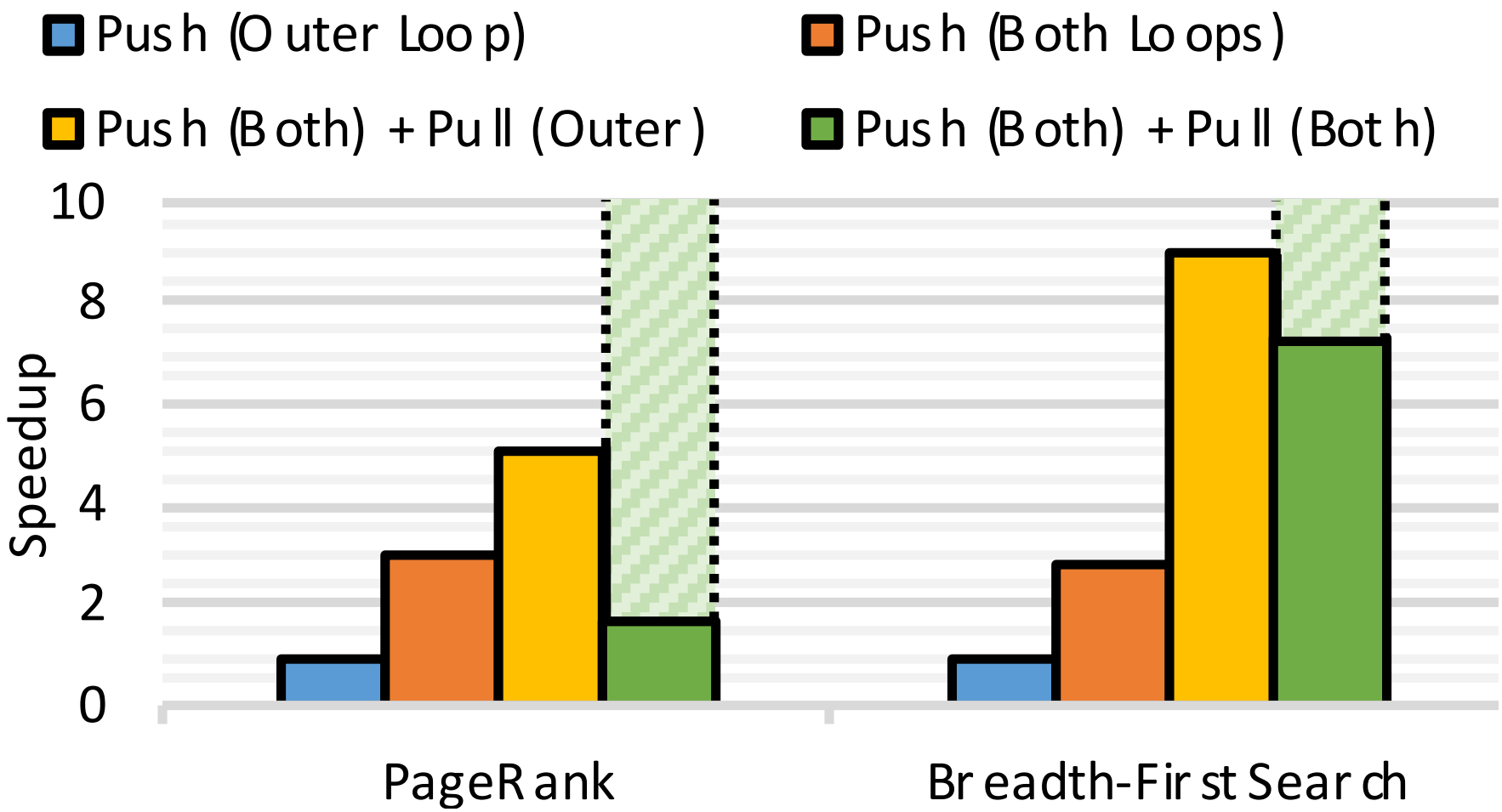
Running Ligra on *twitter-2010* graph

Parallelizing Graph Processing



Running Ligra on *twitter-2010* graph

Parallelizing Graph Processing



Running Ligra on *twitter-2010* graph

Pull's Performance Challenge

Contribution #1: "Scheduler Awareness"

A technique that can be applied to the inner loop of a pull engine to parallelize it without introducing conflicts.

- One thread per vertex
- Updates are thread-private

- Multiple threads per vertex
- Updates will conflict

Pull's Performance Opportunity

- Further gains possible using SIMD vectorization
  - Improve parallelism of the computation
  - Improve memory bandwidth utilization
- Data structure layout issues impede effective vectorization in existing work

Contribution #2: "Vector-Sparse"

A low-level modification to a data structure commonly used to represent graphs, intended to enhance vectorization.

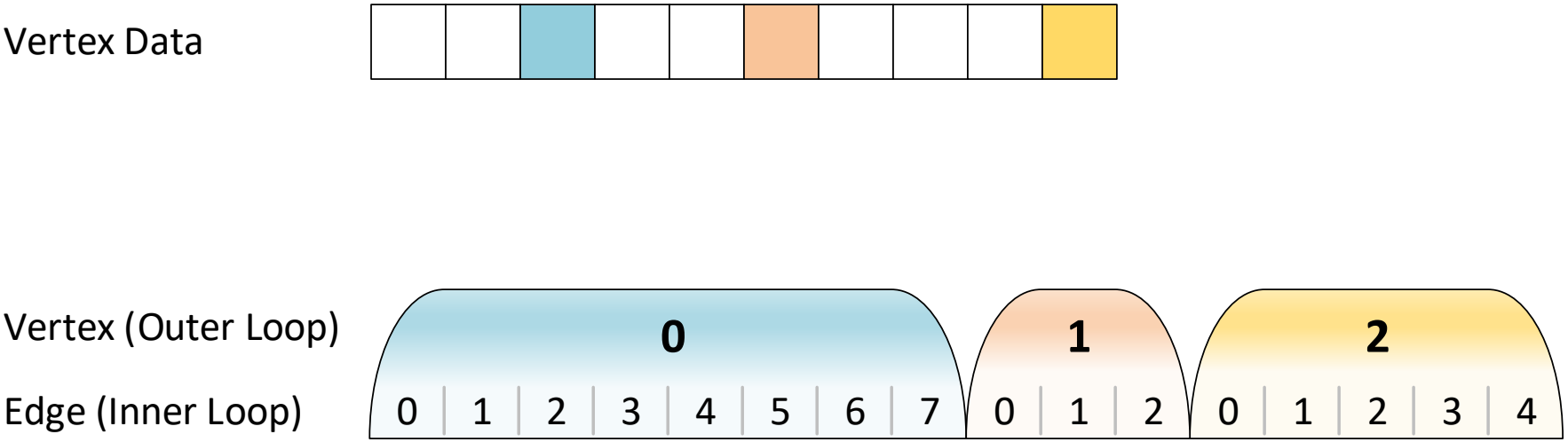
Grazelle

- A hybrid graph processing framework that embodies both of our contributions
- Outperforms the state-of-the-art by over 10× in some cases
- Available for download at https://github.com/stanford-mast/Grazelle-PPoPP18

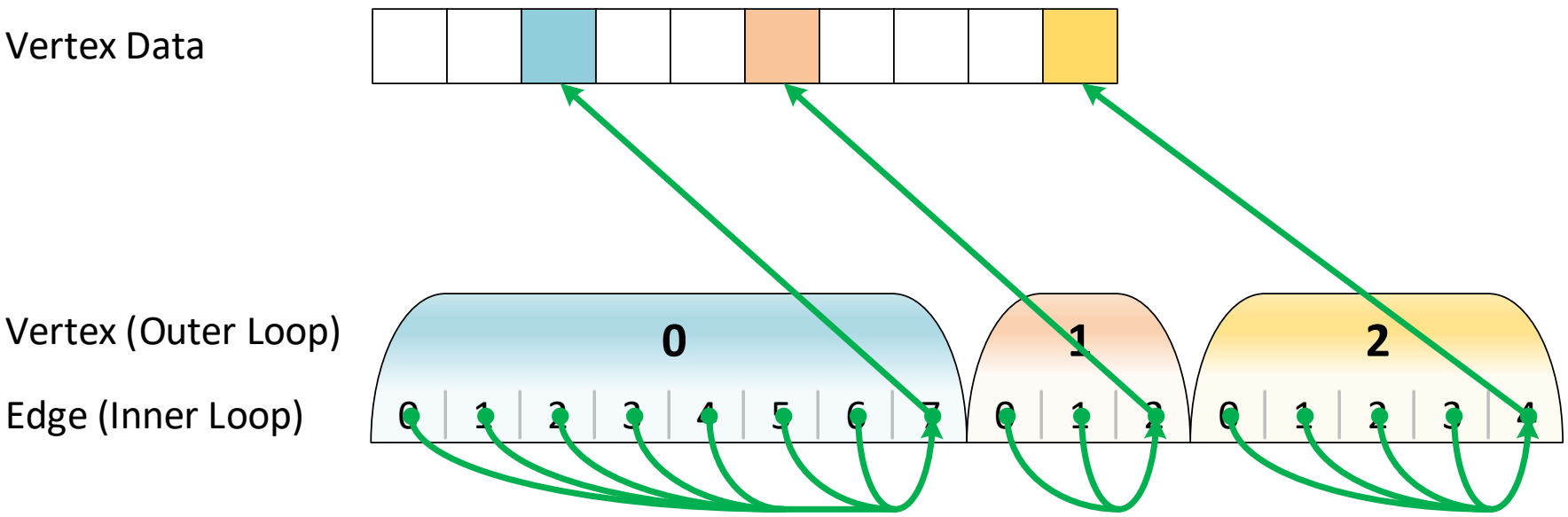
Scheduler Awareness

Contribution #1

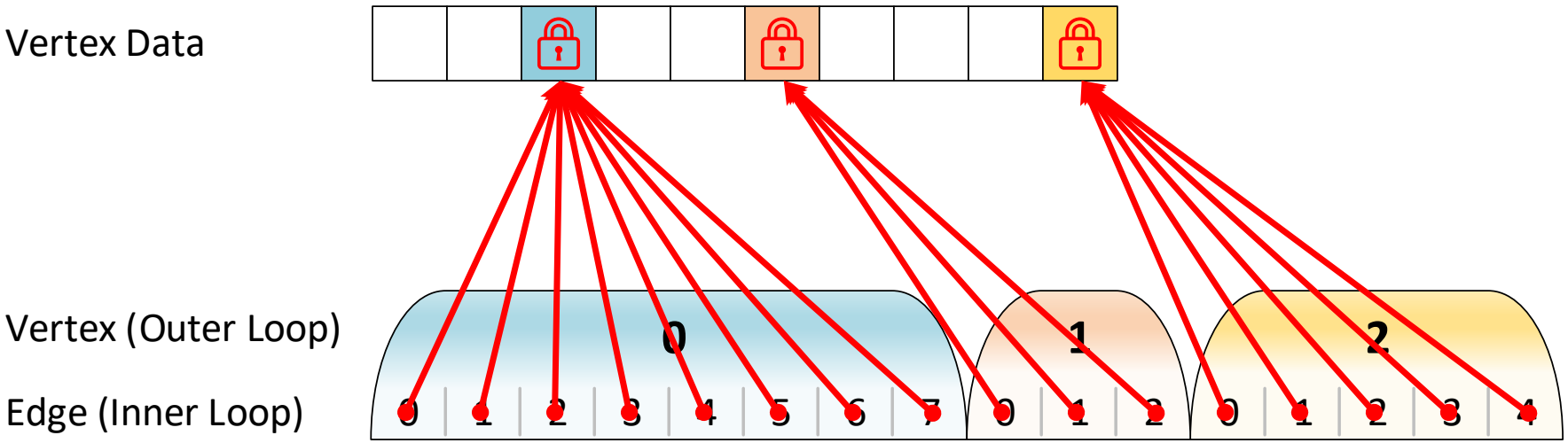
Serial Inner Loop



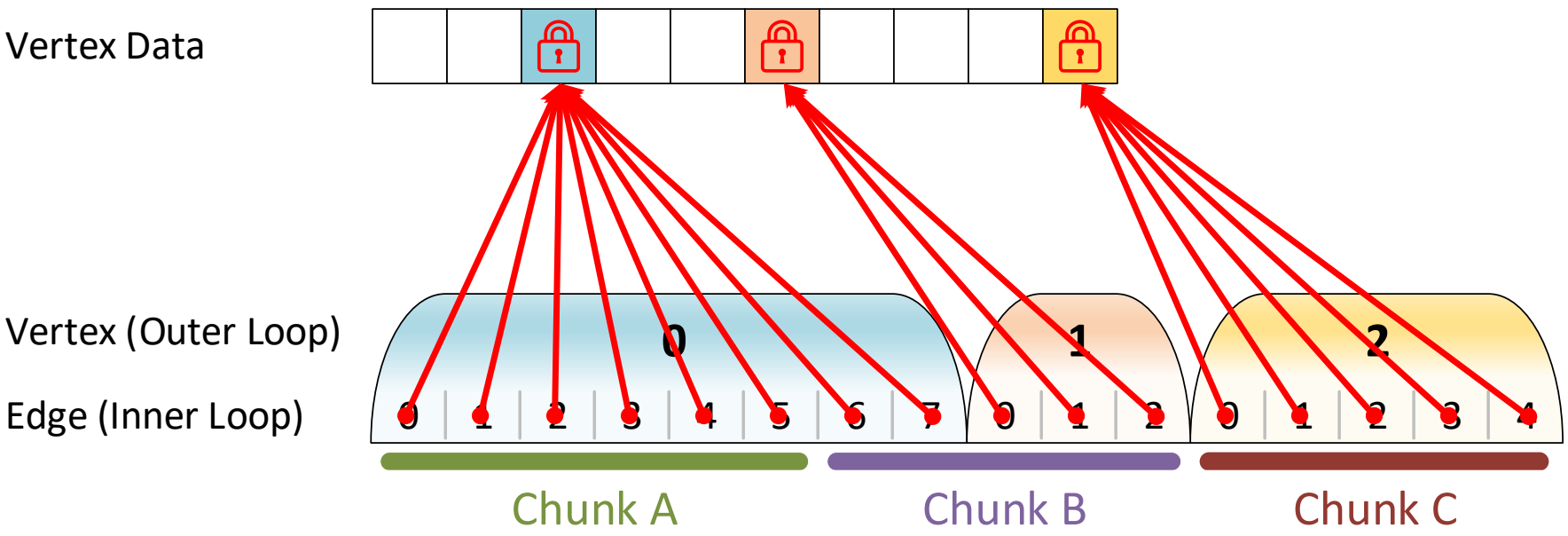
Serial Inner Loop



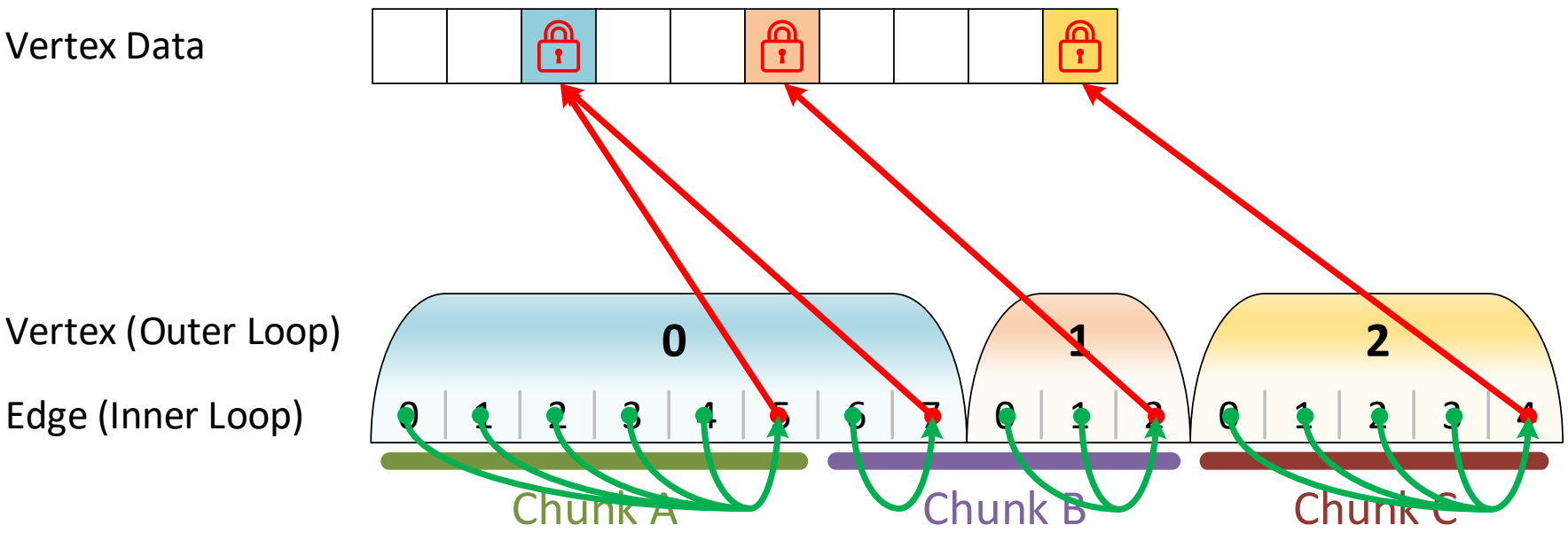
Scheduler Un-Awareness



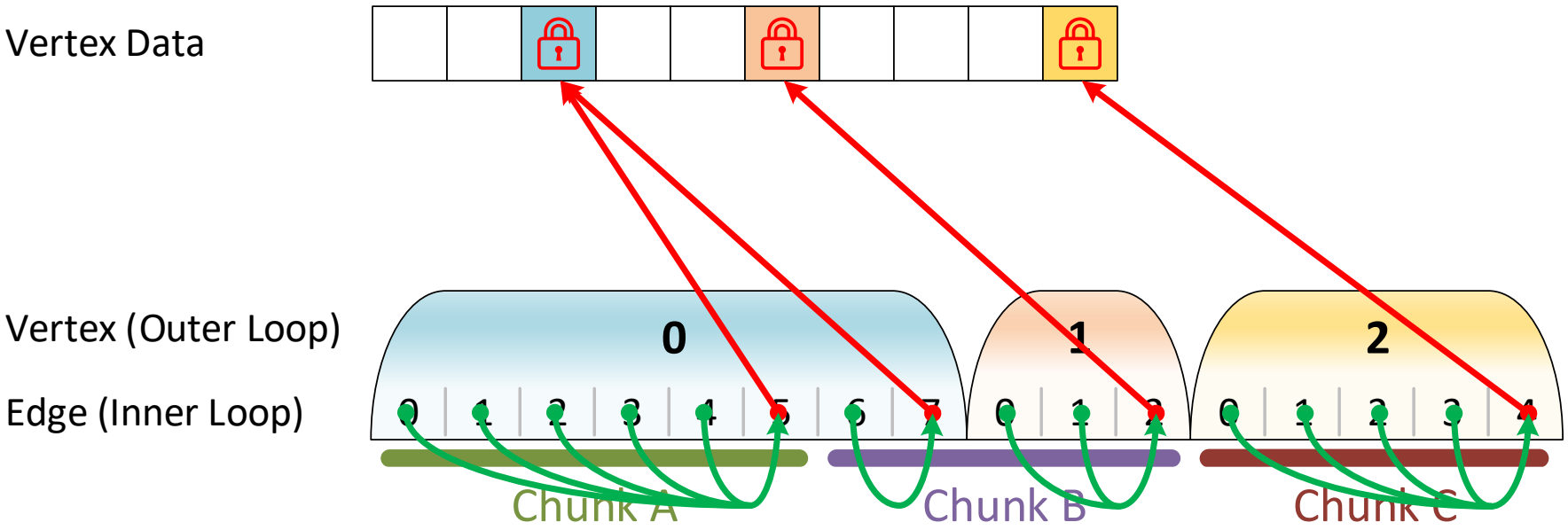
Scheduler Un-Awareness



Scheduler Awareness

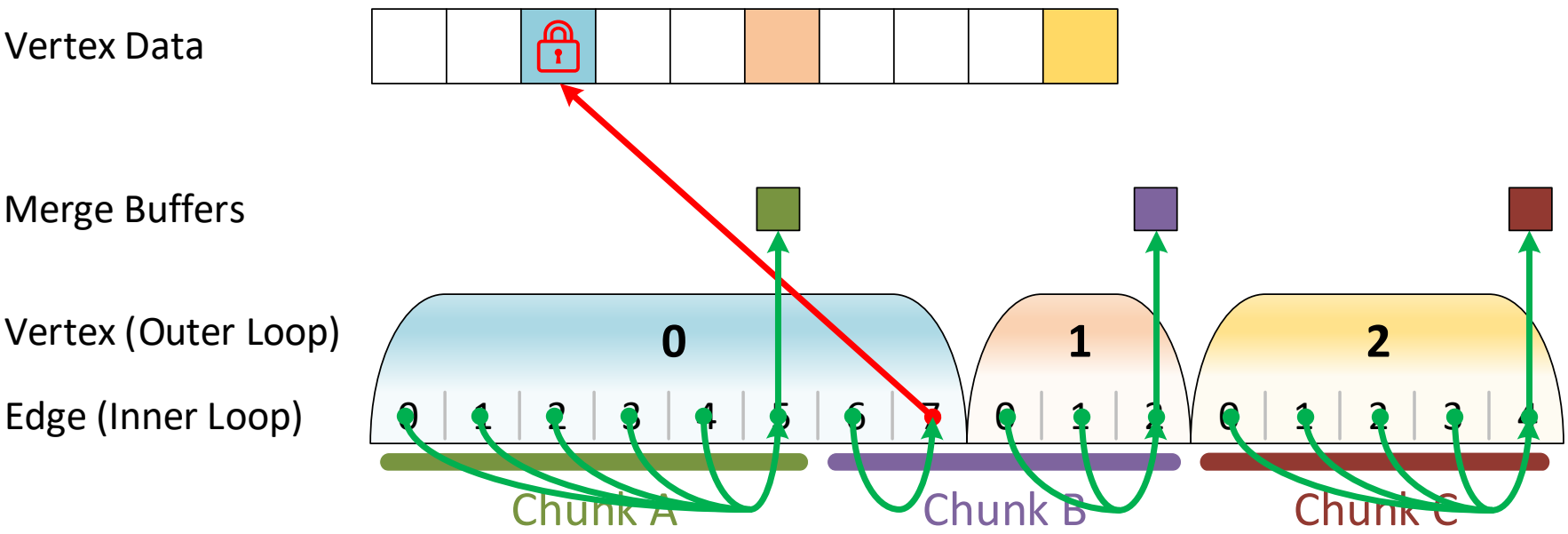


Scheduler Awareness



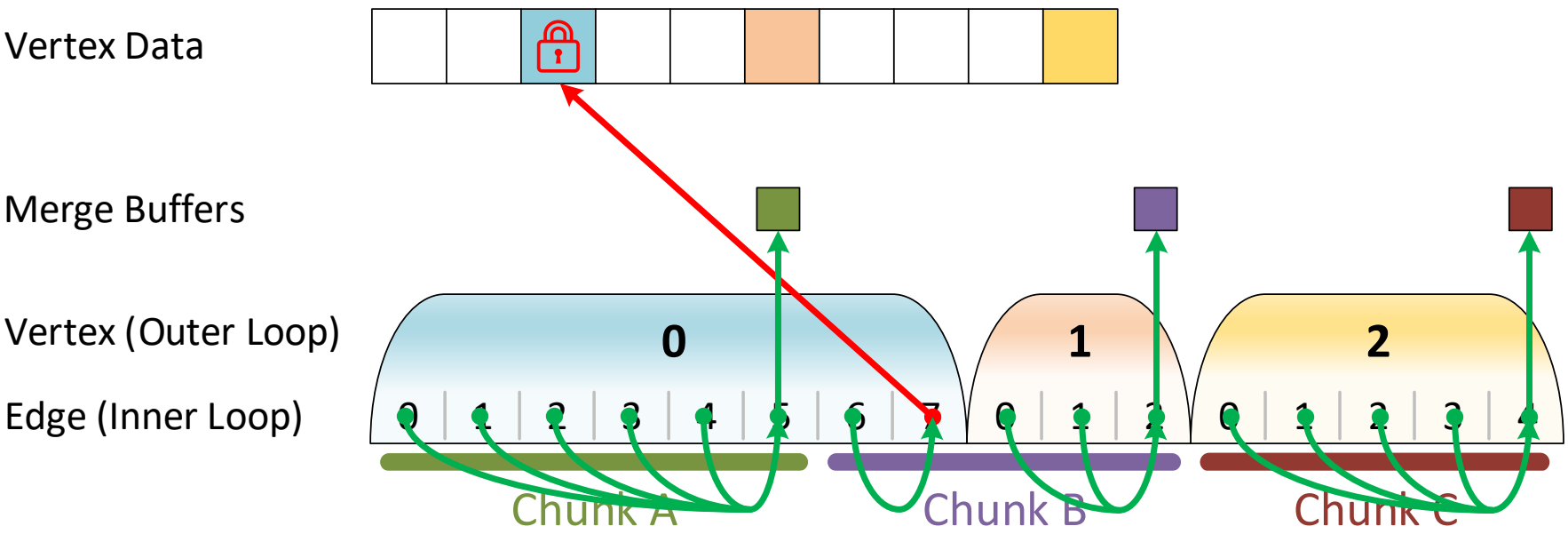
1. Writes **at the end of a chunk** are redirected to a private per-chunk merge buffer.

Scheduler Awareness



1. Writes **at the end of a chunk** are redirected to a private per-chunk merge buffer.

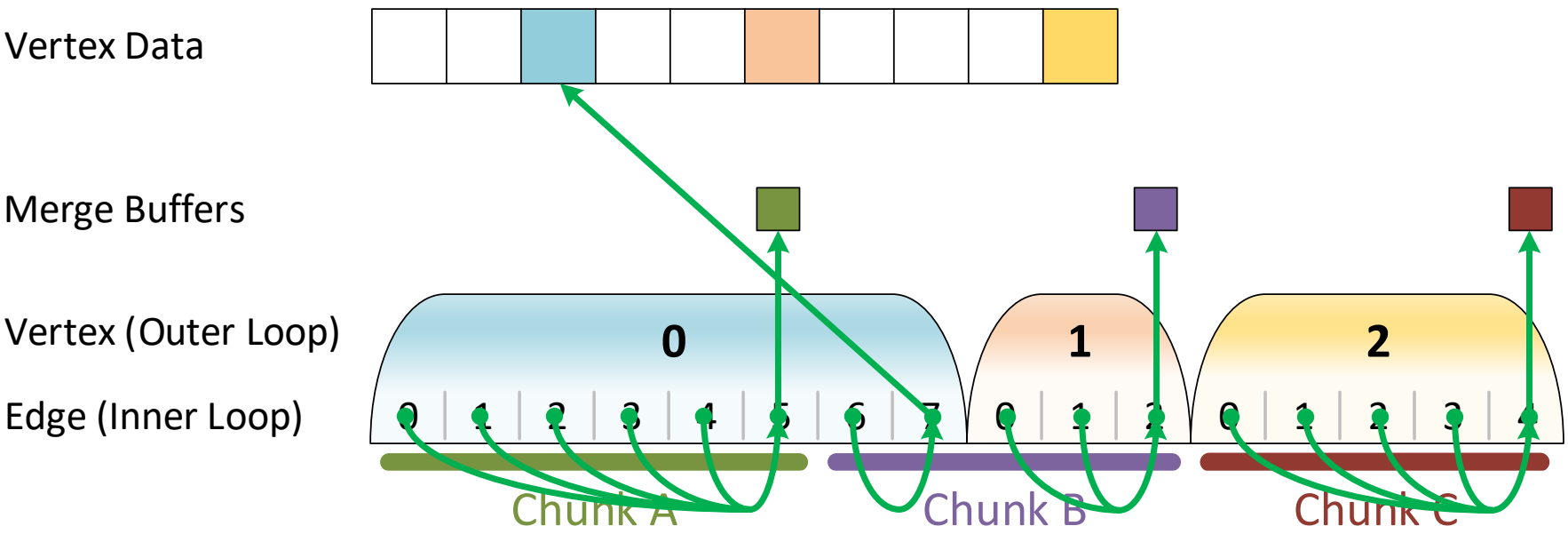
Scheduler Awareness



1. Writes **at the end of a chunk** are redirected to a private per-chunk merge buffer.

2. Writes **in the middle of a chunk** can be committed to shared state without synchronization.

Scheduler Awareness



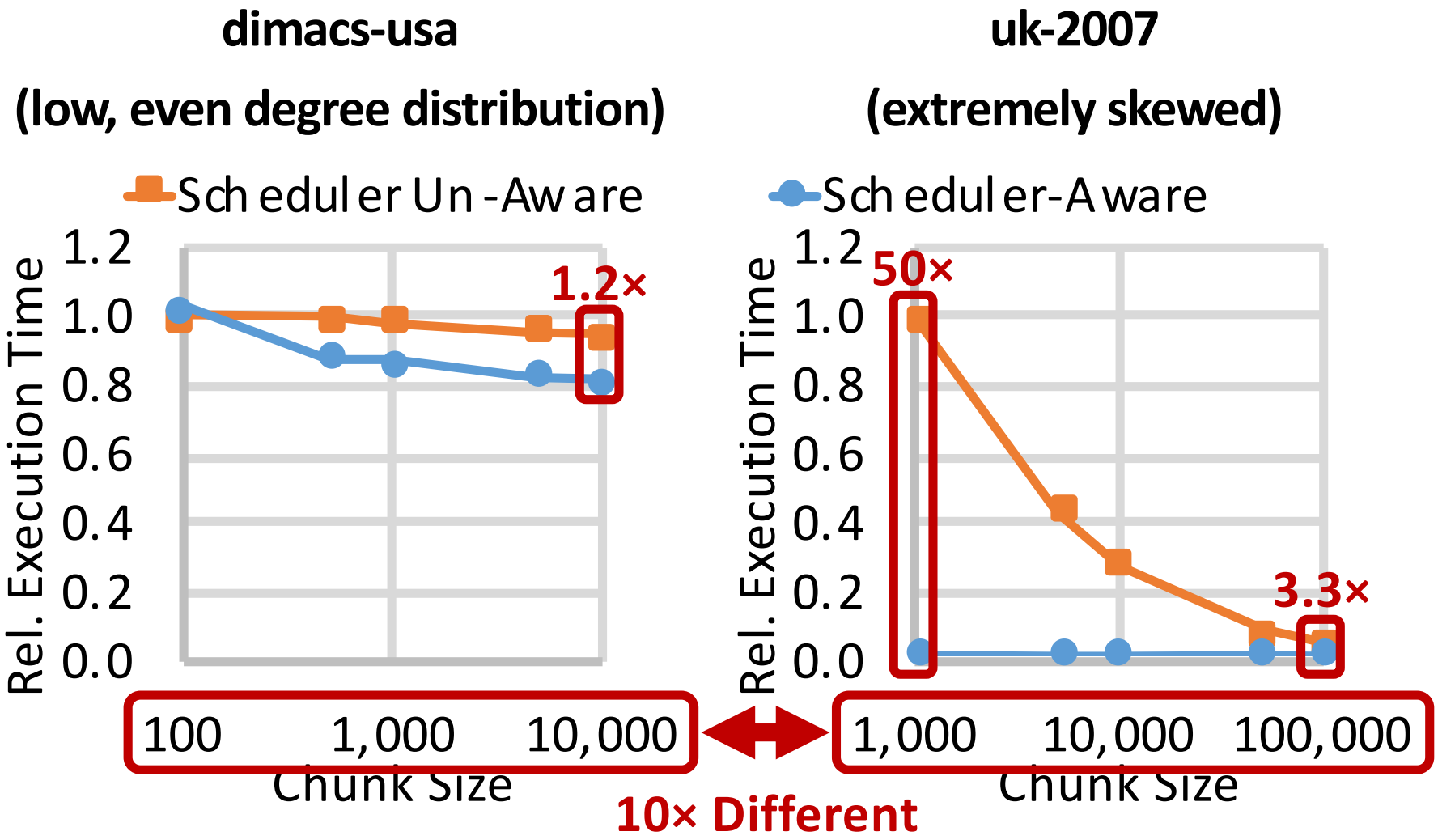
1. Writes **at the end of a chunk** are redirected to a private per-chunk merge buffer.

2. Writes **in the middle of a chunk** can be committed to shared state without synchronization.

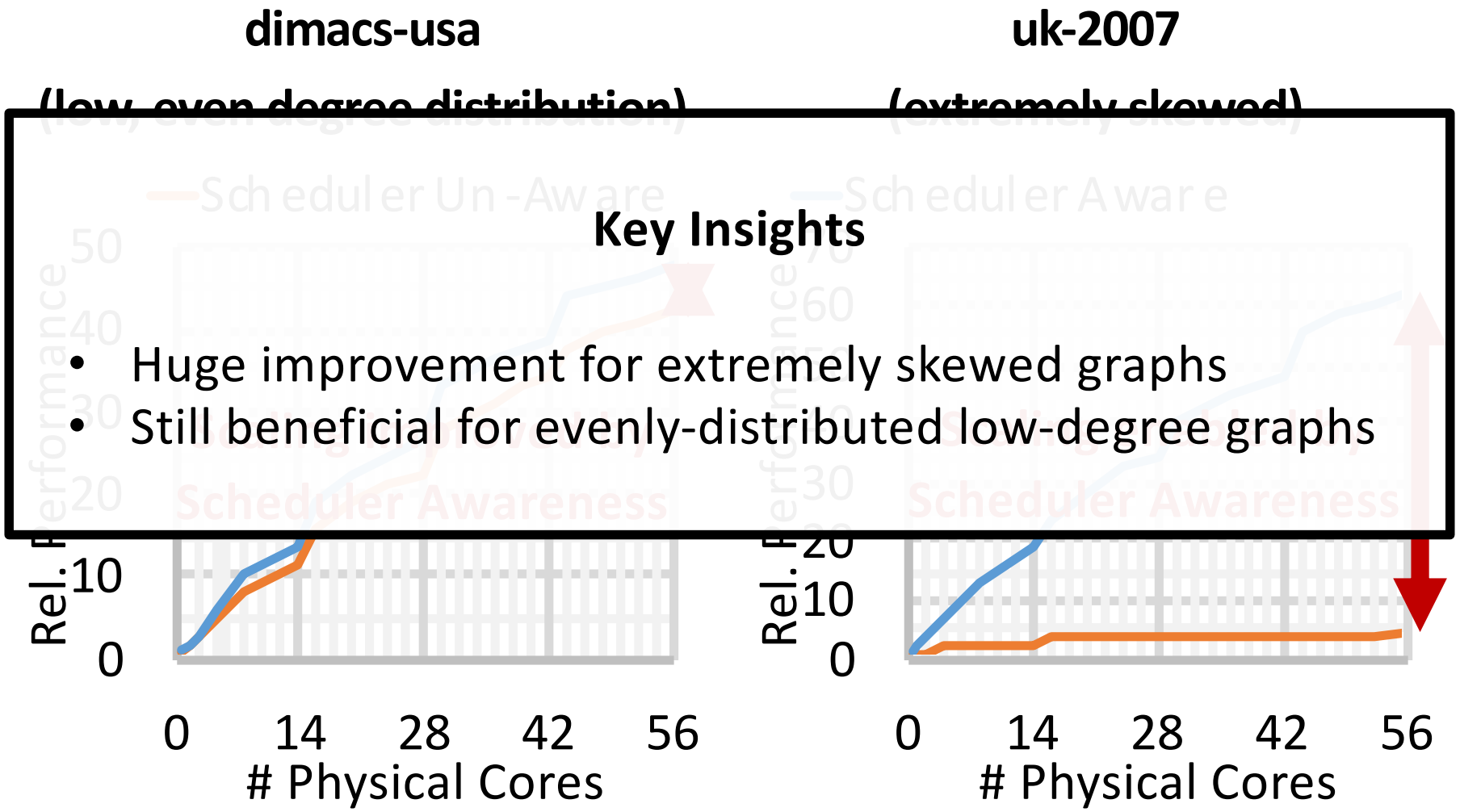
Analyzing Scheduler Awareness

- Performance impact depends primarily on the **scheduling granularity**
- **Scheduler Un-Awareness:** trade-off between load balance and probability of write conflicts
- **Scheduler Awareness:** finer granularity leads to increased merge operation overhead

PageRank: Performance vs. Scheduling Granularity



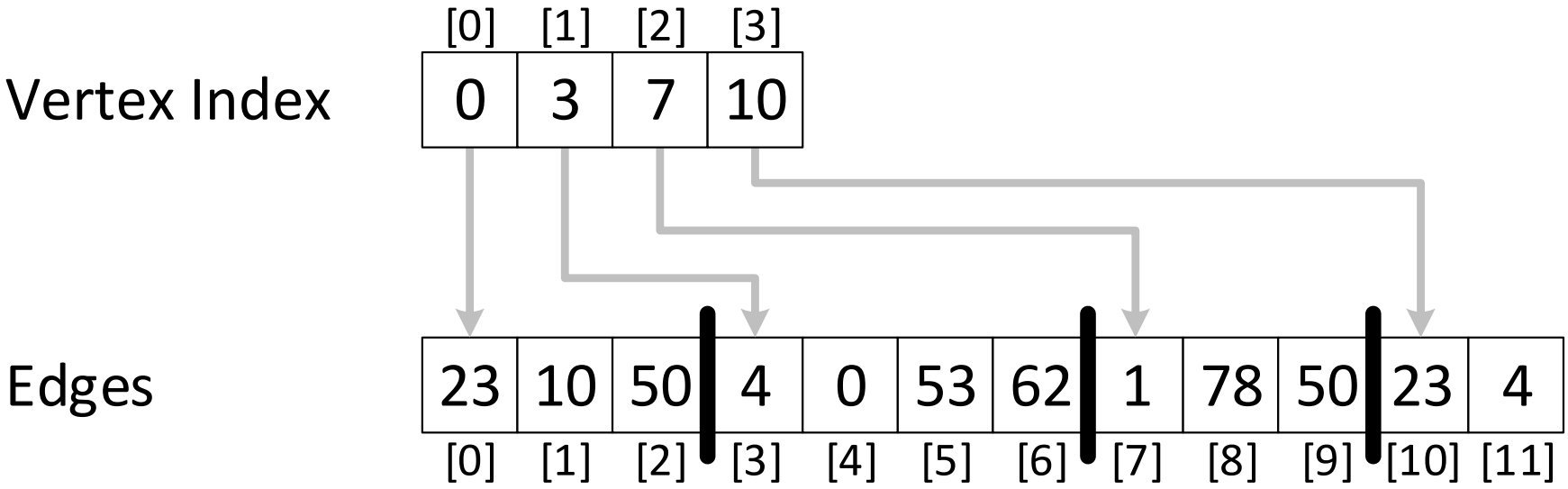
PageRank: Performance vs. Number of Cores



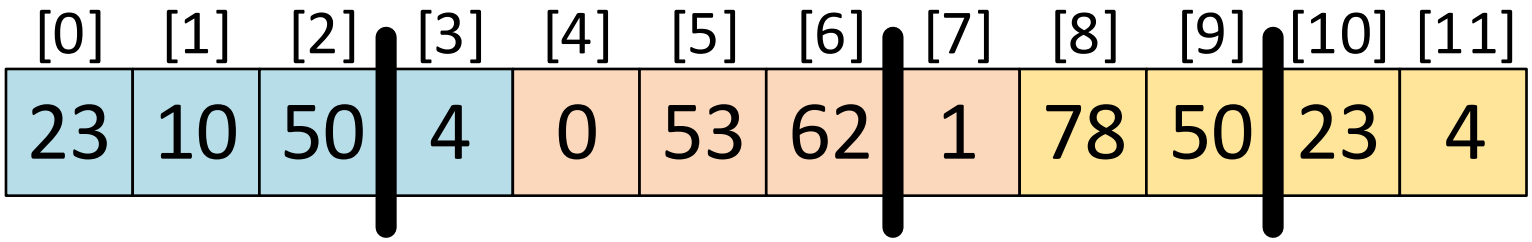
Vector-Sparse

Contribution #2

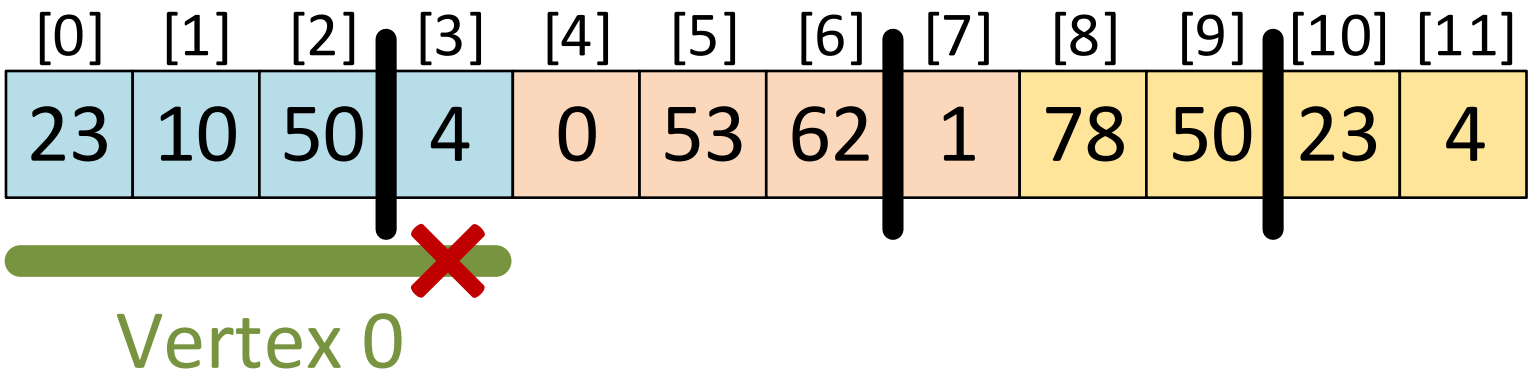
Compressed-Sparse



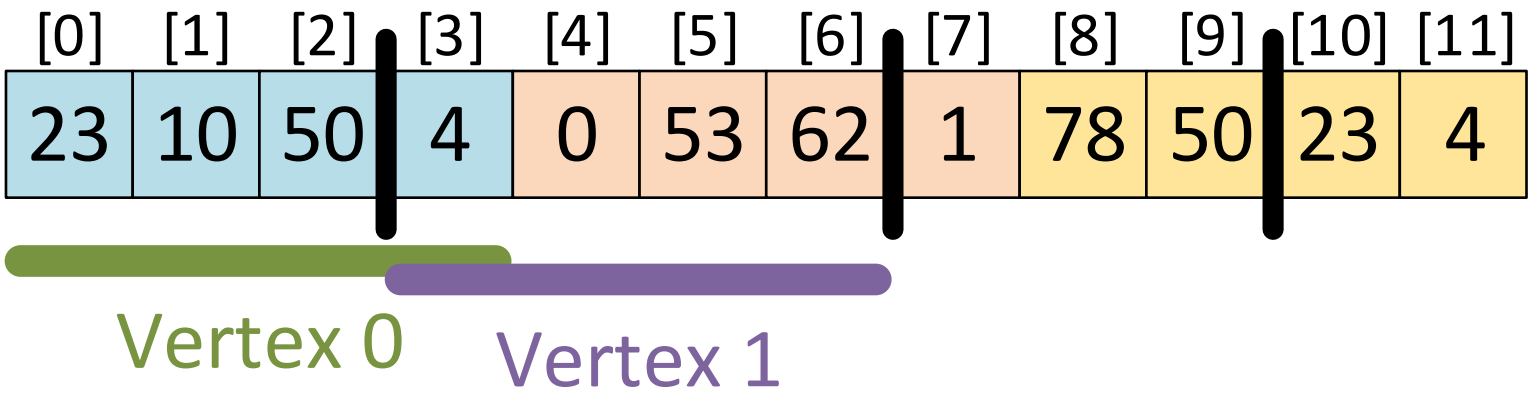
Vectorizing Compressed-Sparse



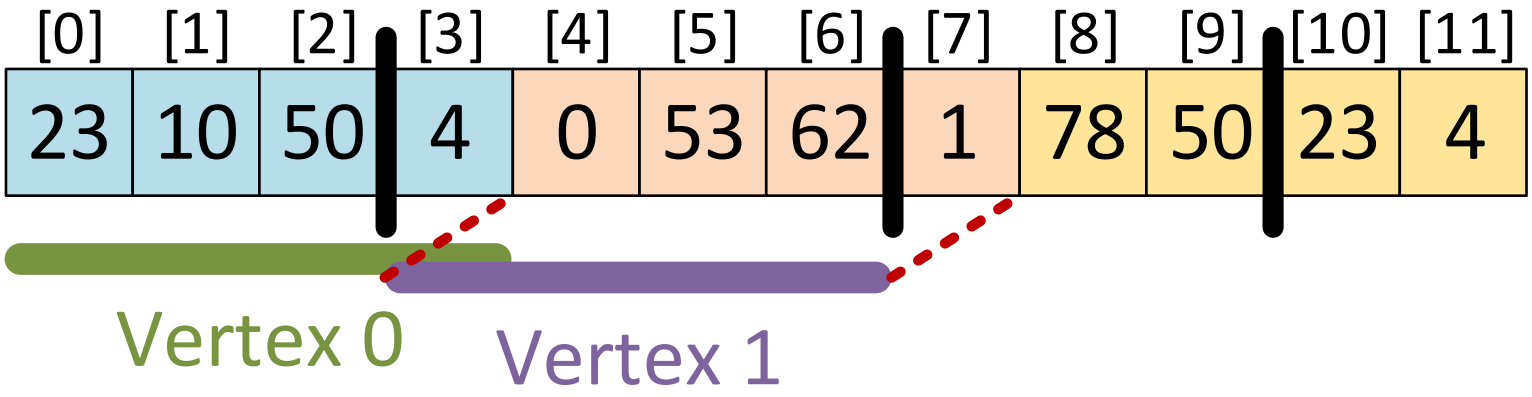
Vectorizing Compressed-Sparse



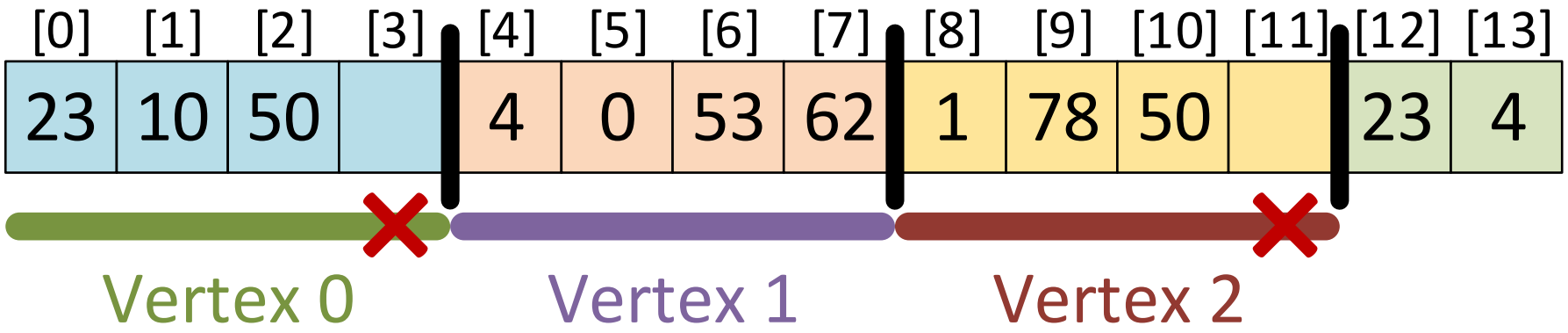
Vectorizing Compressed-Sparse



Vectorizing Compressed-Sparse

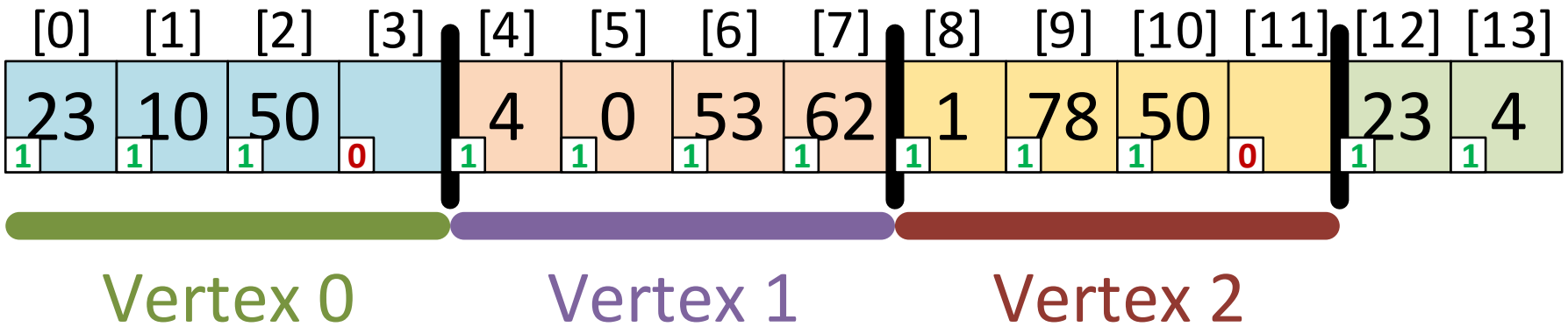


Vector-Sparse



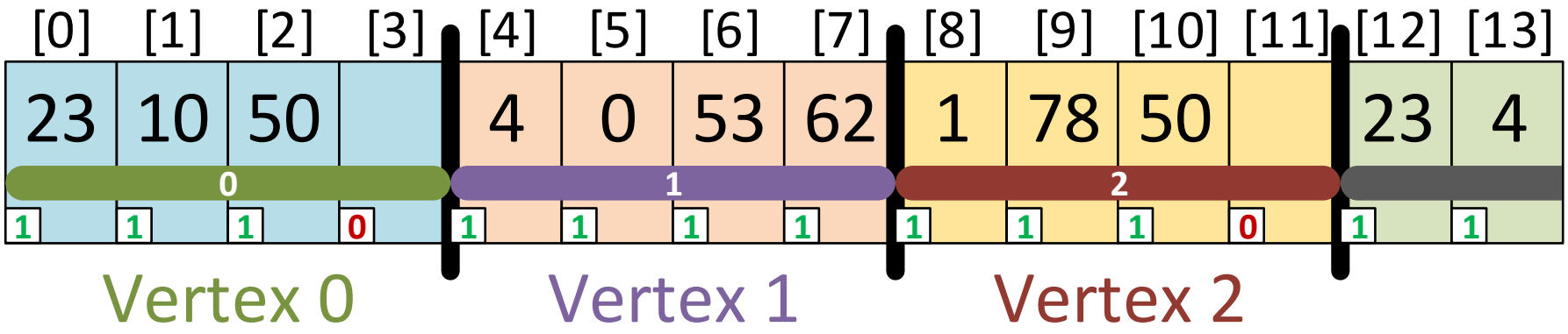
Padding

Vector-Sparse



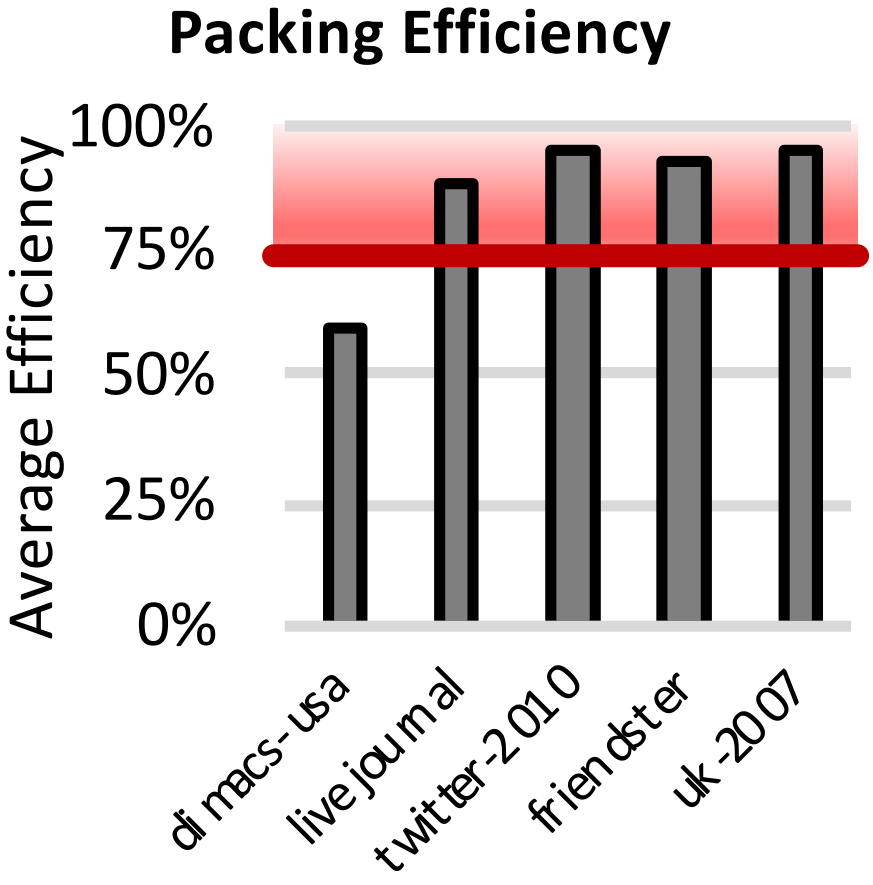
Padding + “valid” bits

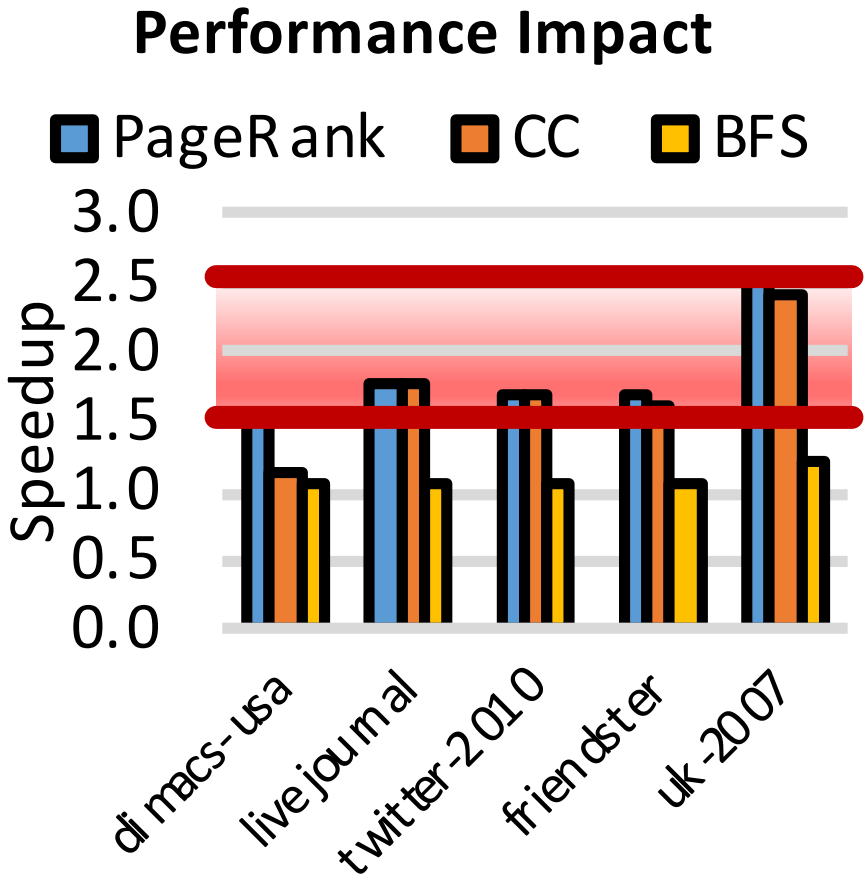
Vector-Sparse



Padding + “valid” bits + top-level vertex spread-encoding

Analyzing Vector-Sparse





Generally ≥ 75%

1.5× to 2.5×

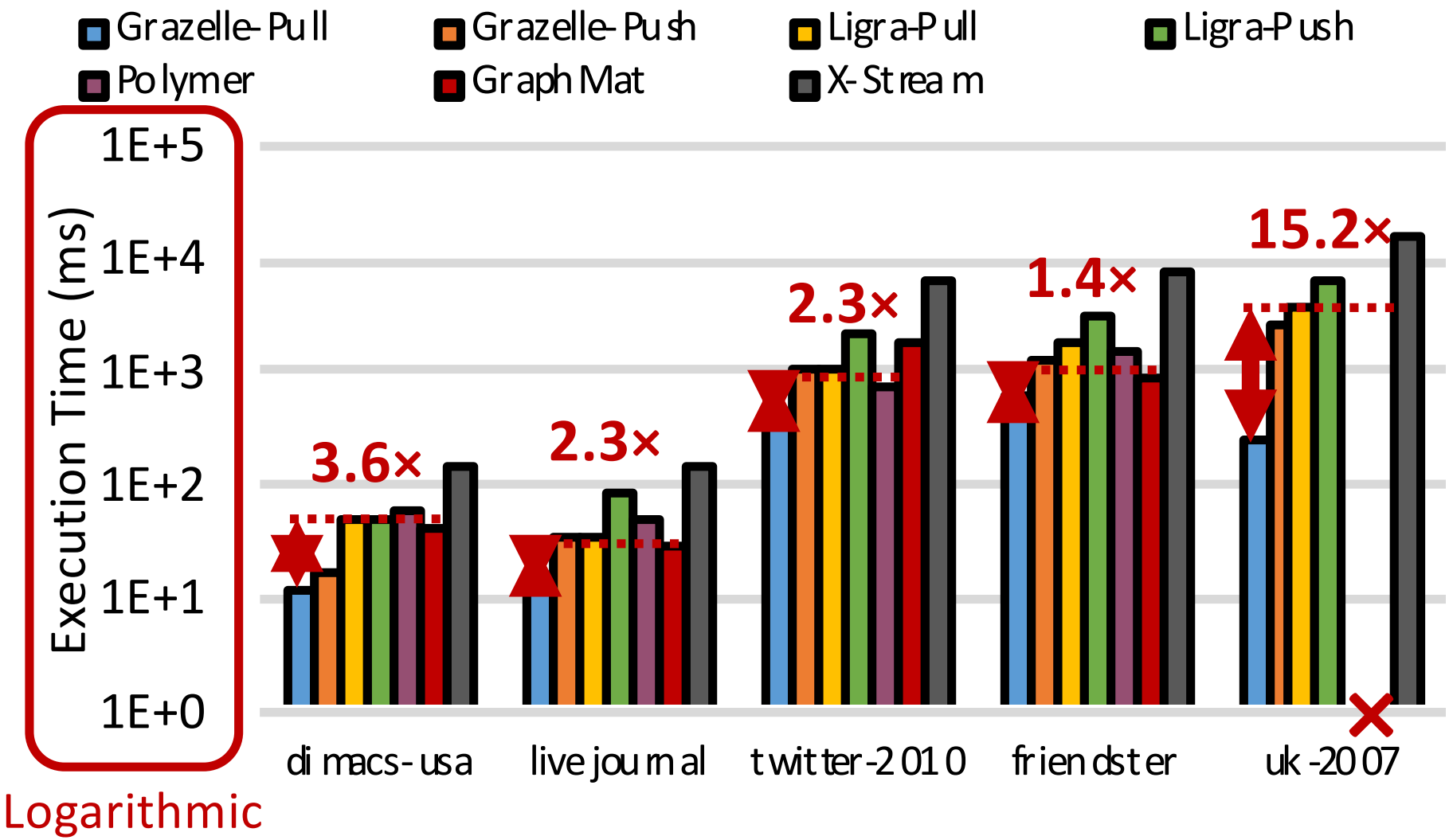
Performance Comparison

Putting it all together

Evaluation Scope

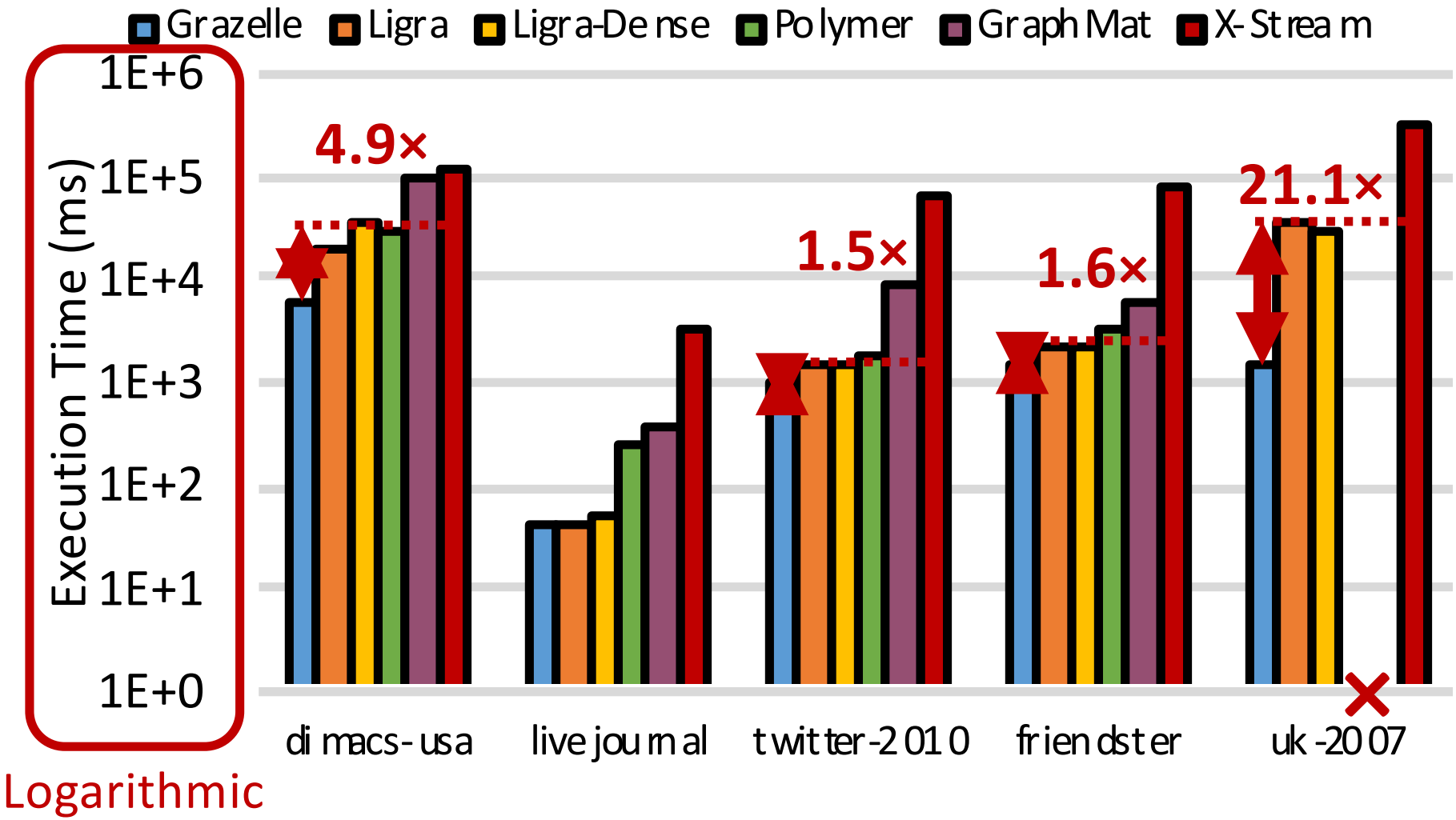
- Grazelle is compared with Ligra, Polymer, GraphMat, and X-Stream
- Three applications: PageRank, Connected Components, Breadth-First Search
- Running on a machine equipped with four Intel Xeon E7-4850 v3 processors
  - 14 physical cores / 28 logical cores per socket

PageRank: Peak Processing Throughput

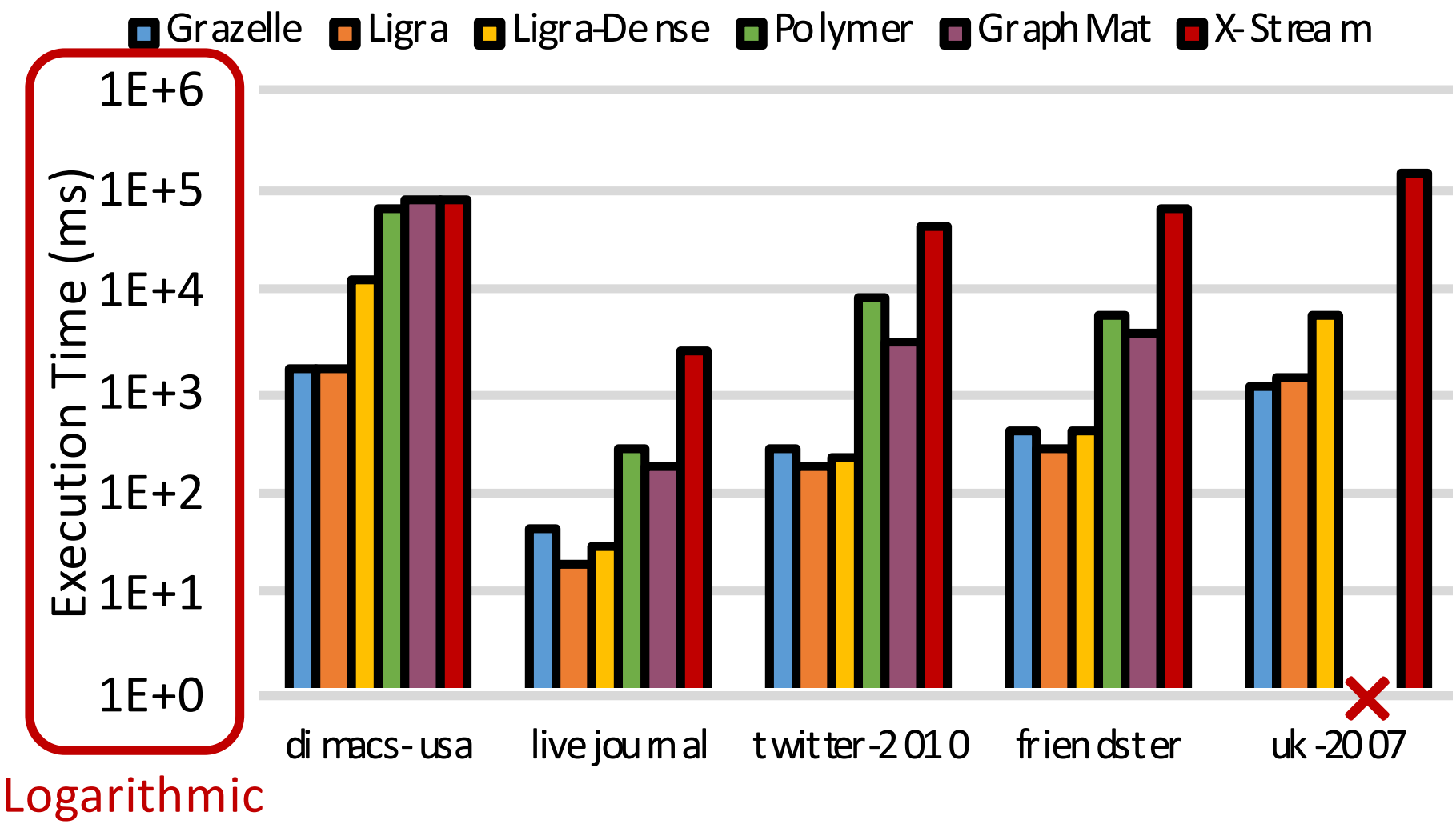


Logarithmic

Connected Components: Dynamic Control Flow



Breadth-First Search: Compatibility of Optimizations



Logarithmic

Conclusion

- Two contributions to improve inner loop parallelization for pull-based graph processing
  - **Scheduler Awareness:** eliminate write conflicts
  - **Vector-Sparse:** enable SIMD vectorization

- Grazelle significantly out-performs state-of-the-art, in some cases by over 10×

- Grazelle is available for download at https://github.com/stanford-mast/Grazelle-PPoPP18

Thank You

Questions?