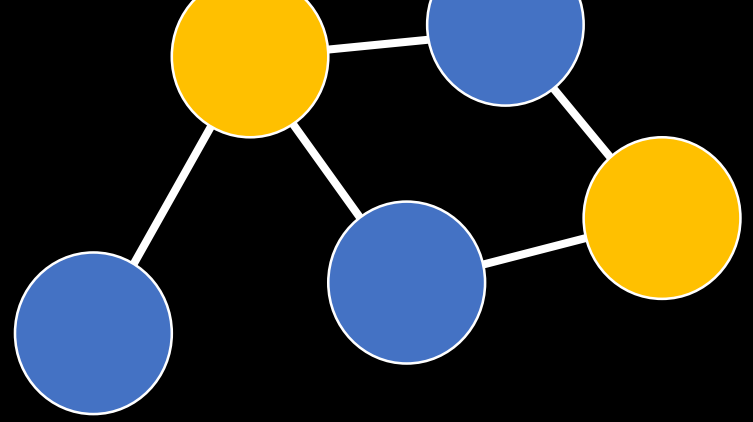


Graph Coloring on the GPU

Muhammad Osama* with Minh Truong*, Carl Yang*†, Aydin Buluç† and John D. Owens*

*University of California, Davis
†Lawrence Berkeley National Laboratory
mosama@ucdavis.edu

UCDAVIS
UNIVERSITY OF CALIFORNIA

GrAPL 2019: Workshop on Graphs, Architectures, Programming, and Learning
2019 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)

Outline

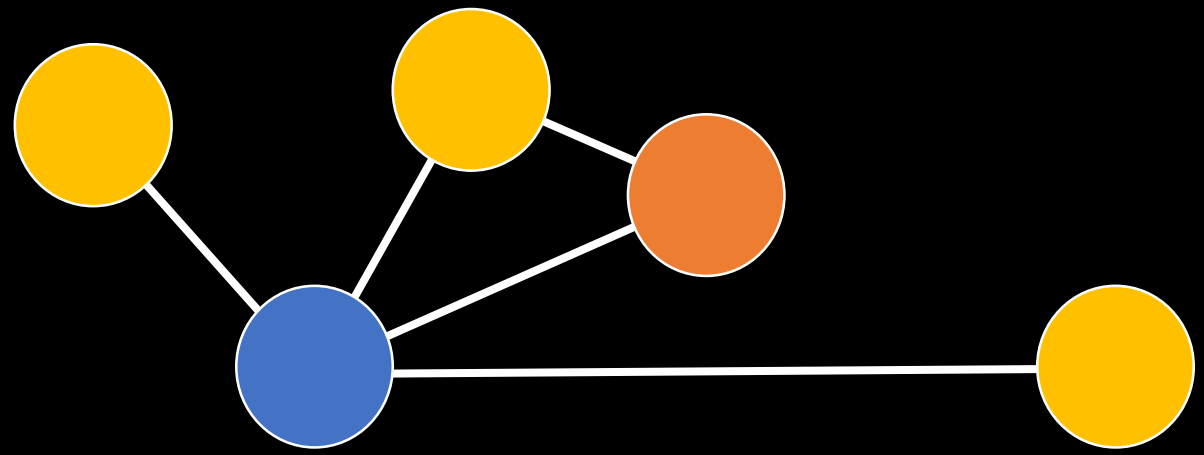
- Contributions
- Graph Coloring Problem
- Graph Analytics Frameworks
- Mapping to Frameworks
- Experiments and Results
- Conclusion

Contributions

- Surveyed parallel graph coloring algorithms on the GPU
- Implemented graph coloring on a **data-centric** and a **linear-algebra-based** graph frameworks
- First to design a parallel graph coloring algorithm that uses linear-algebra-based primitives based on GraphBLAS API

What is Graph Coloring?

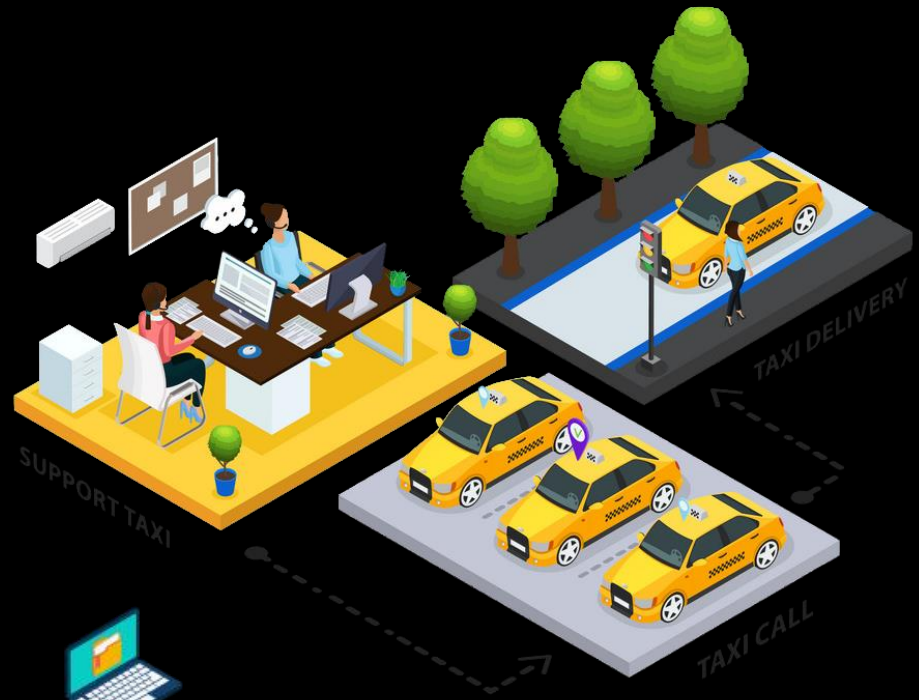
Assign colors to vertices or edges in a graph such that **no two neighboring** vertices or edges have the **same color**.



Motivation

- Scheduling
- Register allocation
- Pattern Matching

and more…



Images sources:
https://www.preferredtechnology.com/maximize-potential-wireless-access-point/
and https://www.spotnrides.com/taxi-dispatch-software

Serial Greedy Algorithm

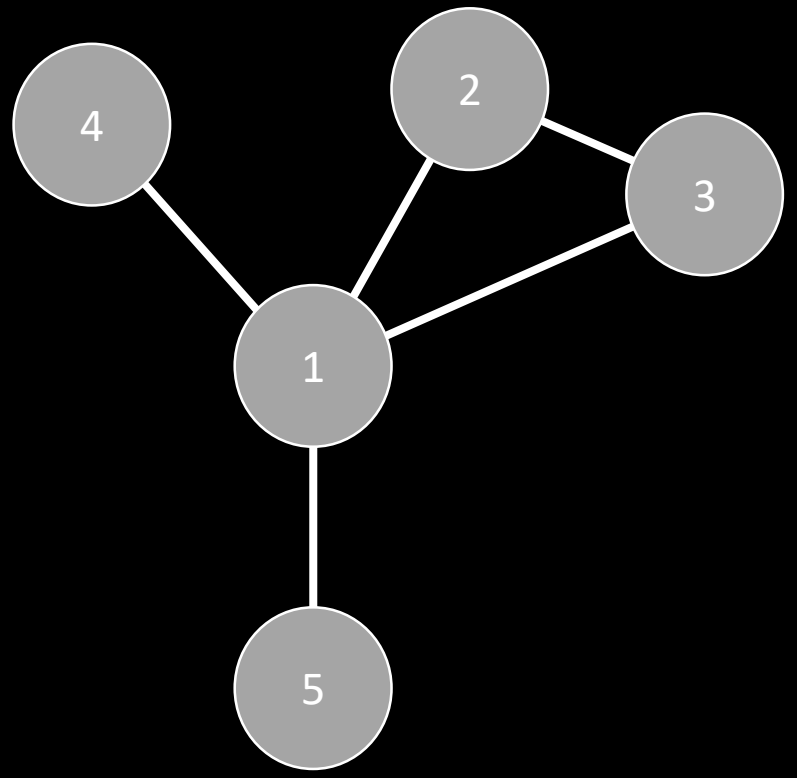
1. Starts with some **ordering of vertices**

2. Color each vertex in order using the **minimum color** that does not appear in its neighbors

Serial Greedy Algorithm

1. Order

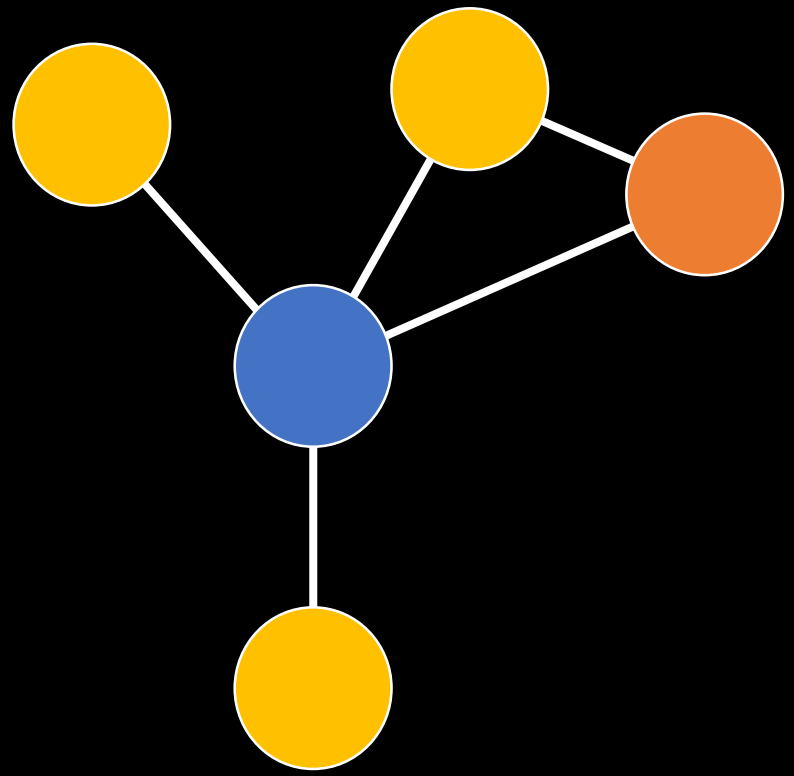
2. Color using minimum color



Serial Greedy Algorithm

Finding the optimal ordering is NP-hard

Not easily Parallelizable as is



Parallel Luby's Algorithm

1. For each $v \in V$, generate a random number $p(v)$

2. Vertex v is added to the independent set I if and only if $p(v) > p(w)$ for all $w \in adj(v)$

3. Color the set I with color c, such that c is the minimum available color.

Parallel Luby's Algorithm + Jones and Plassmann

1. Assign each vertex to a processor
2. Communicate its colors with the neighboring vertices
3. Colors itself using the minimum available color

Parallel Luby's Algorithm + **Jones and Plassmann**

1. Assign ea

All vertices working in Parallel

2. Communicate its colors with the neighbori vertices

Asynchronous execution

3. Colors itself using the minimum available color

Challenges

Difficult to express, implement and optimize
a high-performance hardwired graph
algorithm on the GPU

Goals

GPU Graph Framework allow us to express,
implement and optimize graph algorithms.

We set two goals to test these frameworks:

1	**Flexibility** of GPU graph frameworks
2	**Performance** against state-of-the-art

Graph Analytics Frameworks

Gunrock

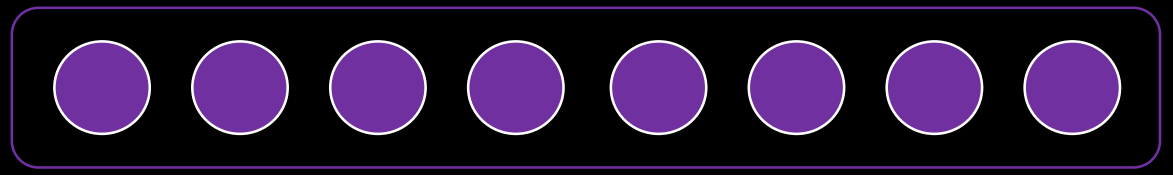
Data-centric abstraction

GraphBLAS

Linear-algebra-based abstraction

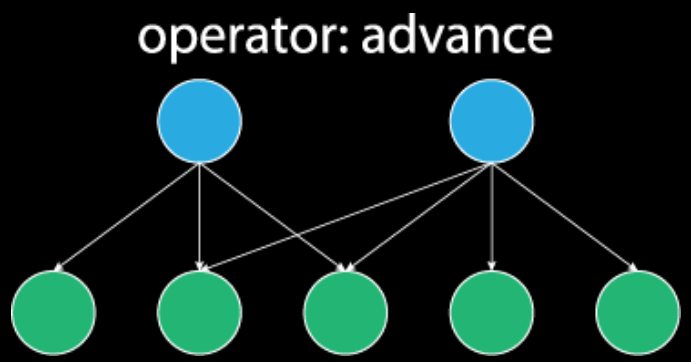
Graph Analytics Framework: Gunrock

- **Data-centric** abstraction – using frontiers
- A **frontier** is a group of vertices or edges

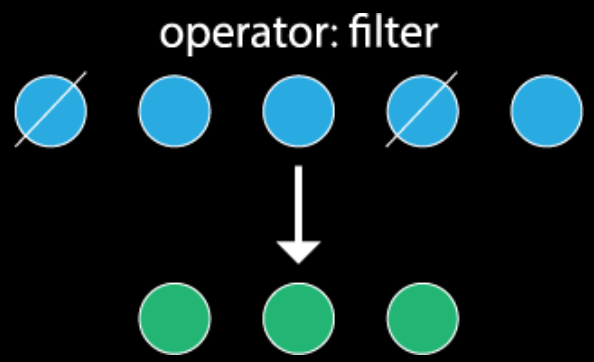
frontier: 

- Manipulation of frontiers is an **operation**

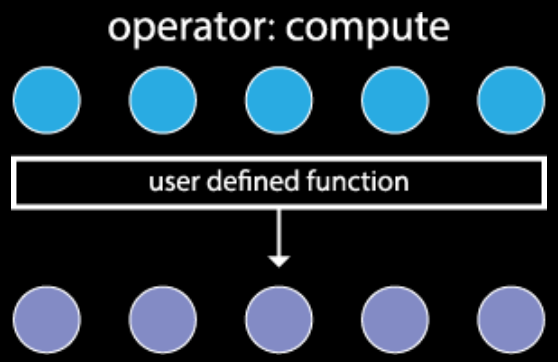
Gunrock: Operators



Generates new frontier by visiting neighbors of the input frontier

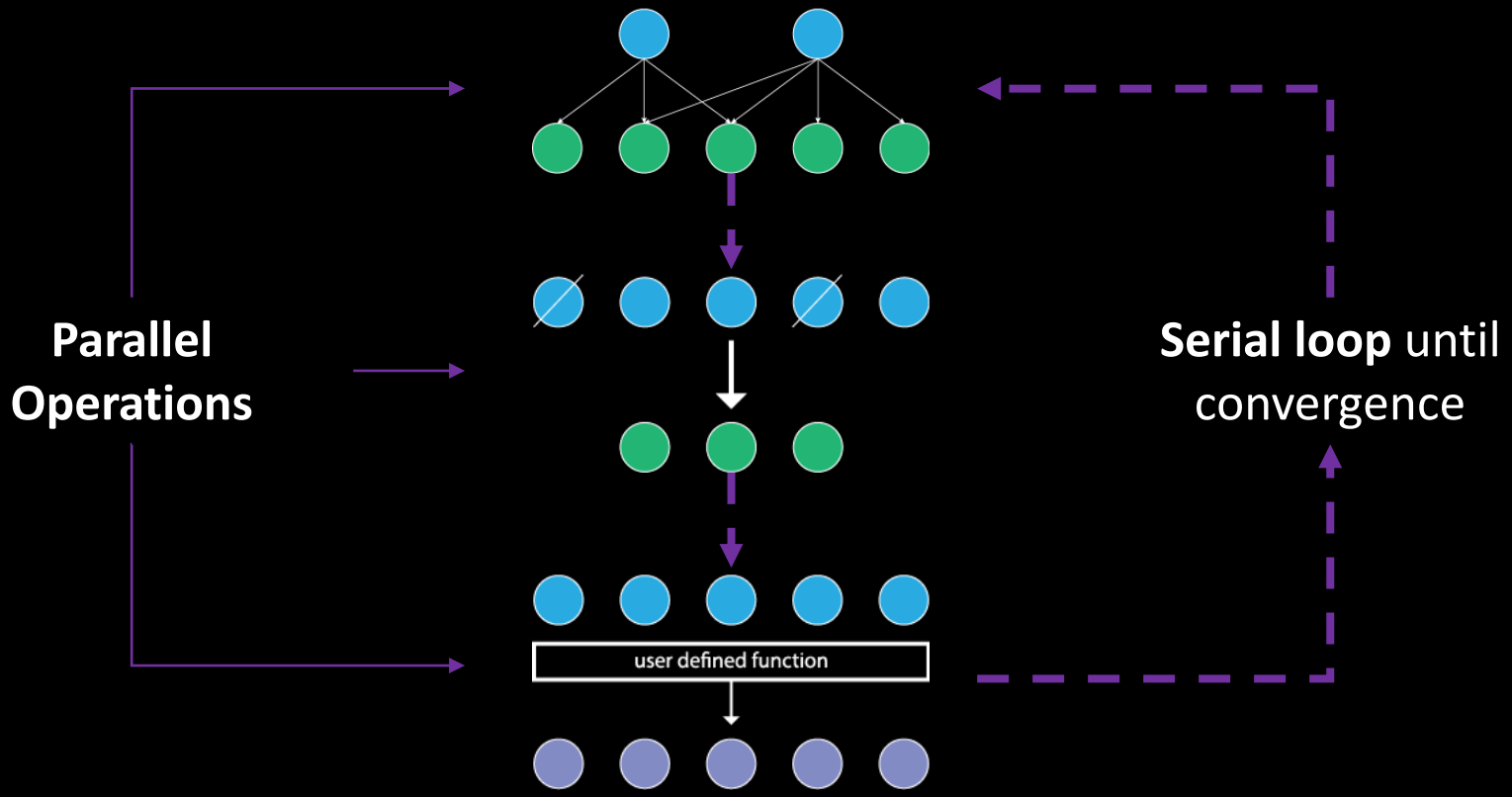


Chooses a subset of current frontier as the new frontier

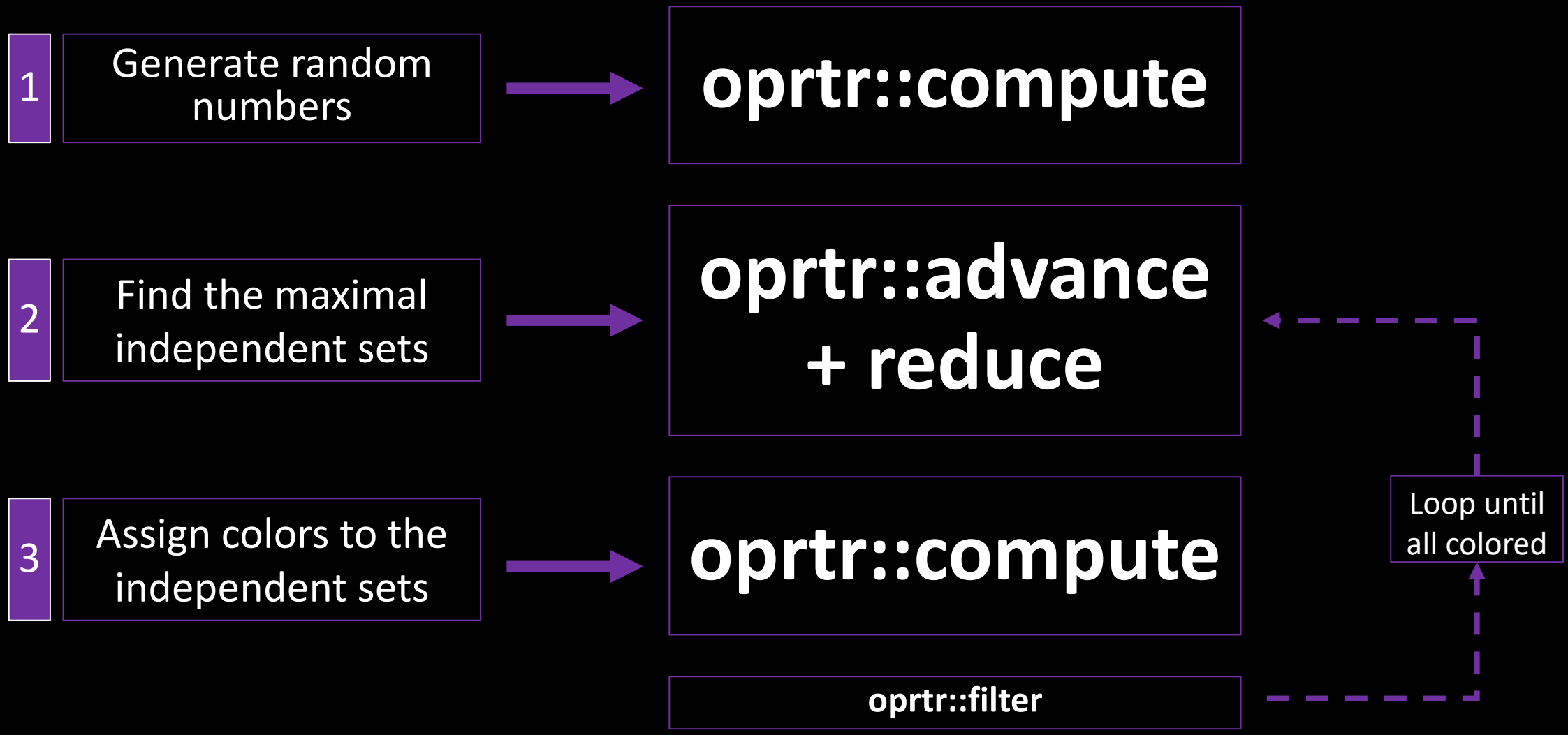


Applies a compute operation on all elements

Gunrock: Bulk-Synchronous Programming



Mapping **Independent Set** Graph Coloring: **Gunrock**

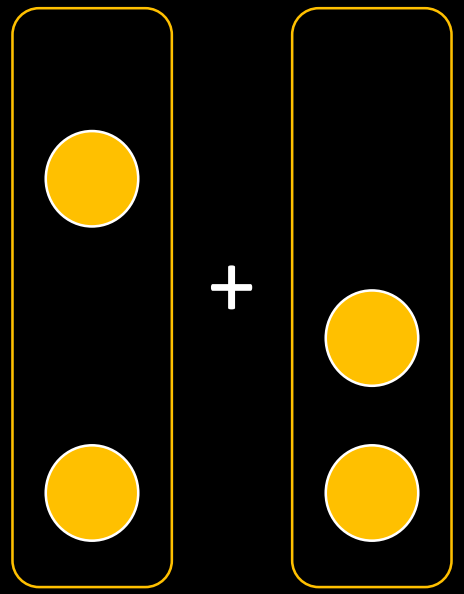


Graph Analytics Framework: GraphBLAS

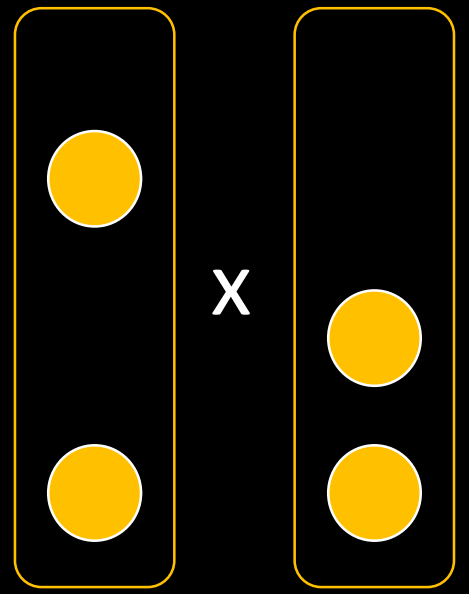
- **Linear-algebra-based** abstraction
- Uses matrix algebra to perform graph operations

GraphBLAS: Operations

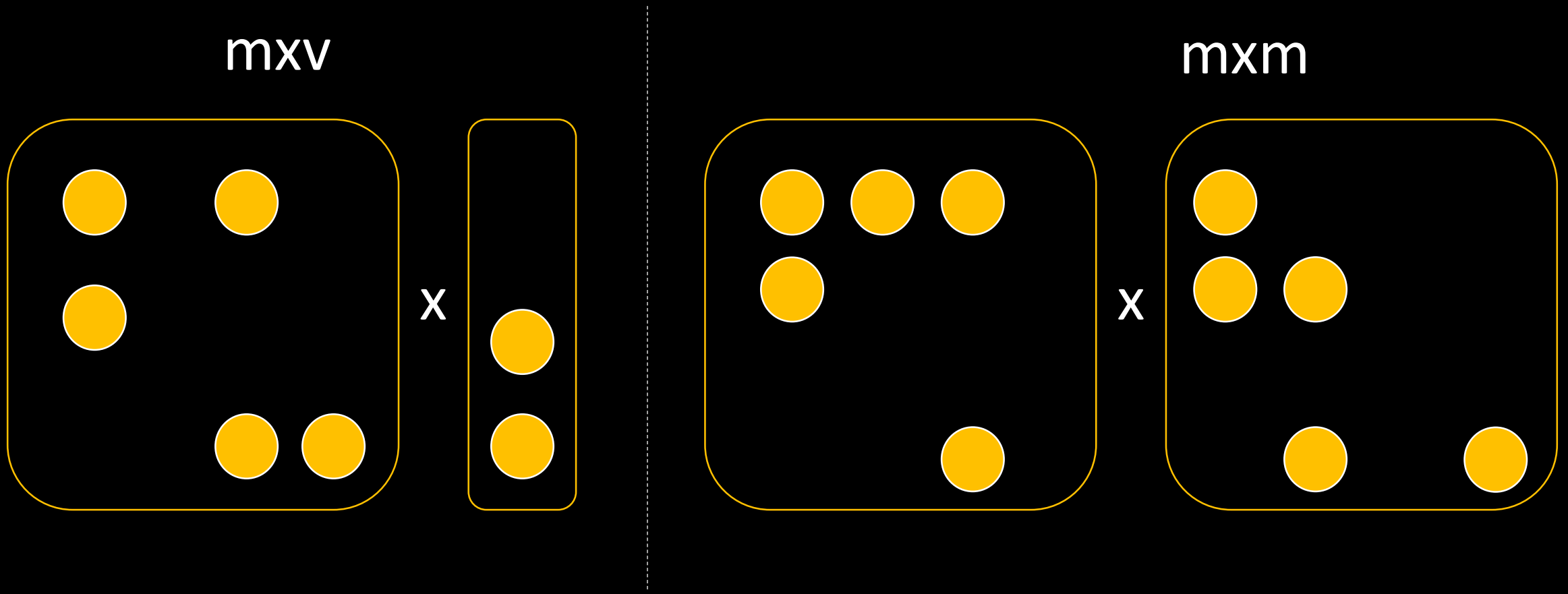
eWiseAdd



eWiseMult



GraphBLAS: Operations

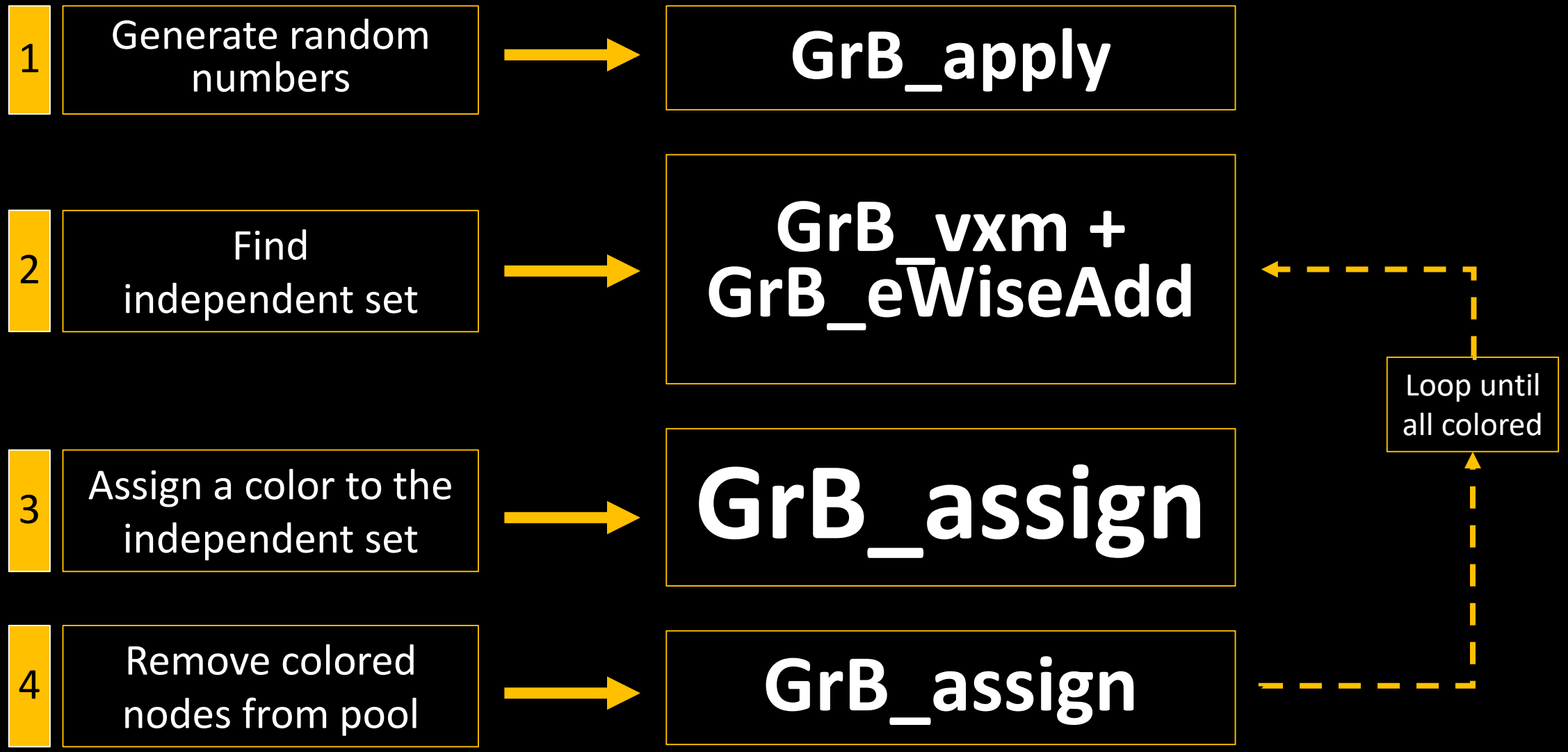


GraphBLAS: Operators

Semiring notation: (Add, Multiply, Domain)

Name	Semiring	Application
Real field	$\{+, \times, \mathbb{R}\}$	Classical numerical linear algebra
Boolean	$\{\|, \&, \{0, 1\}\}$	Graph connectivity
Tropical	$\{\min, +, \mathbb{R} \cup \{\infty\}\}$	Shortest path
Max-plus	$\{\max, +, \mathbb{R}\}$	Graph matching
Min-times	$\{\min, \times, \mathbb{R}\}$	Maximal independent set

Mapping **Independent Set** Graph Coloring: **GraphBLAS**

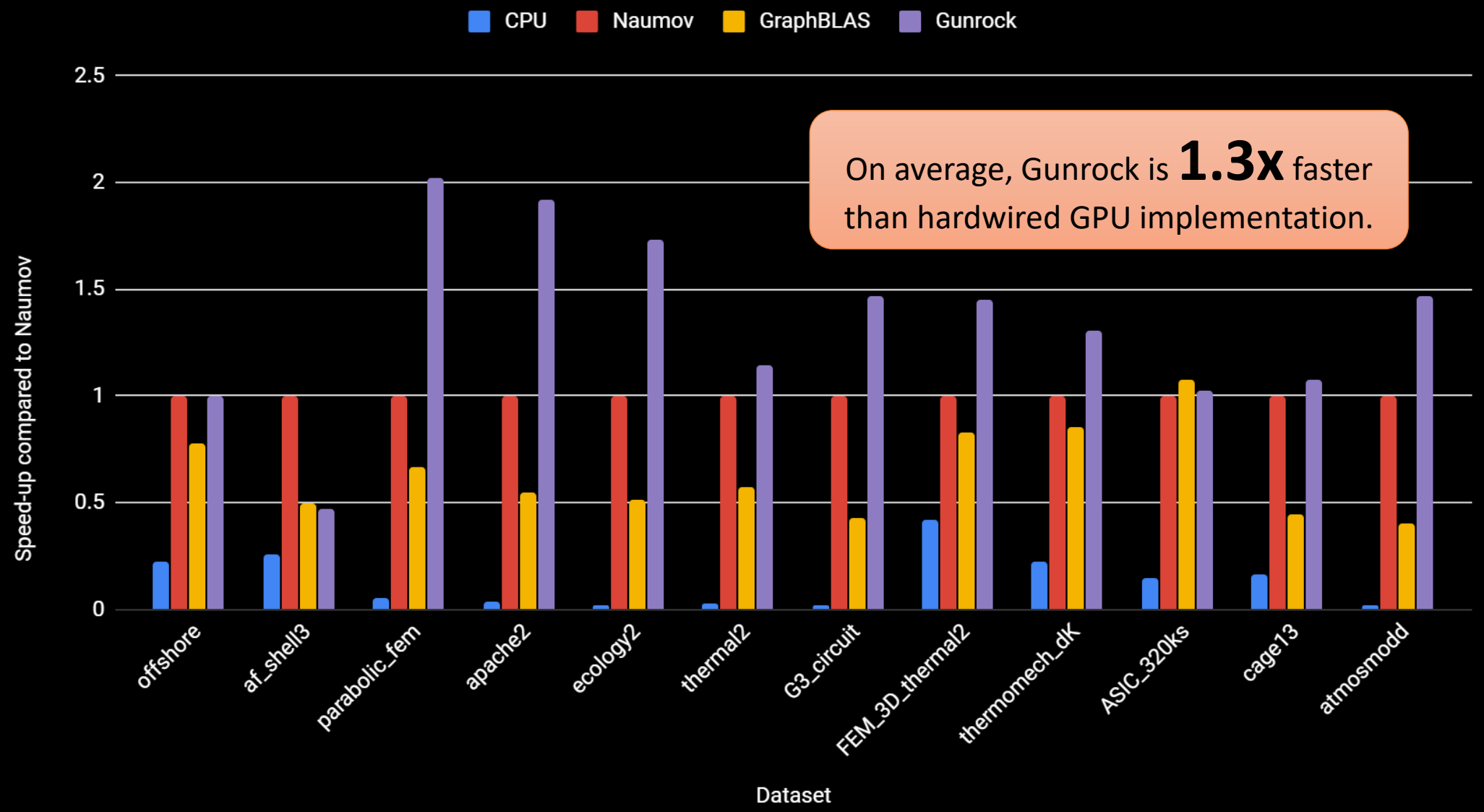


Experimental Setup

- 2x 3.50 GHz Intel 4-core E5-2637 v2 Xeon CPUs
- NVIDIA K40c GPU w/ 12 GB of on-board memory
- 556 GB of Main Memory

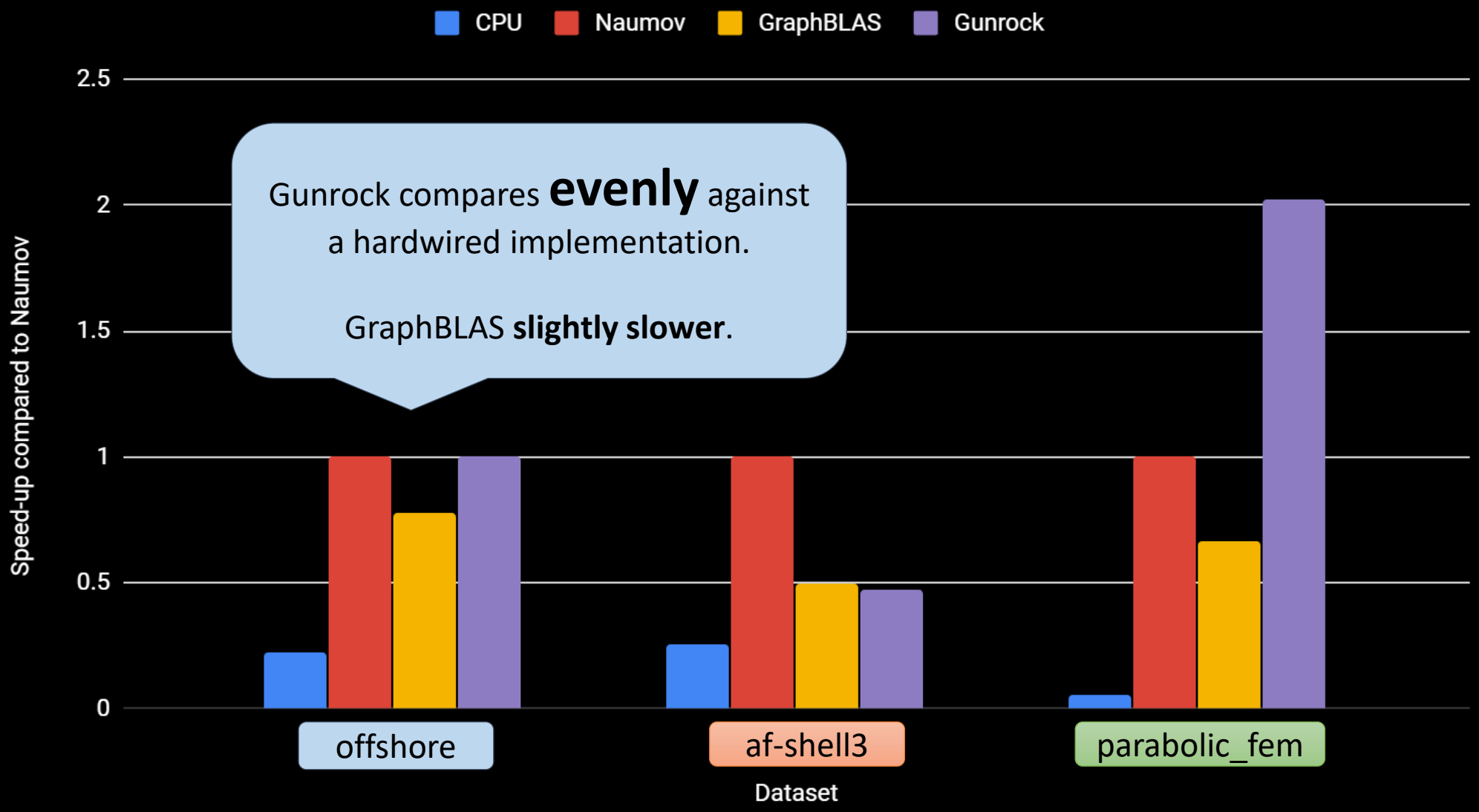
- 22 Dataset
  - 10 randomly generated graphs (rgg)
  - 12 mesh-like or from Finite Element Modeling graphs

Runtime Comparison

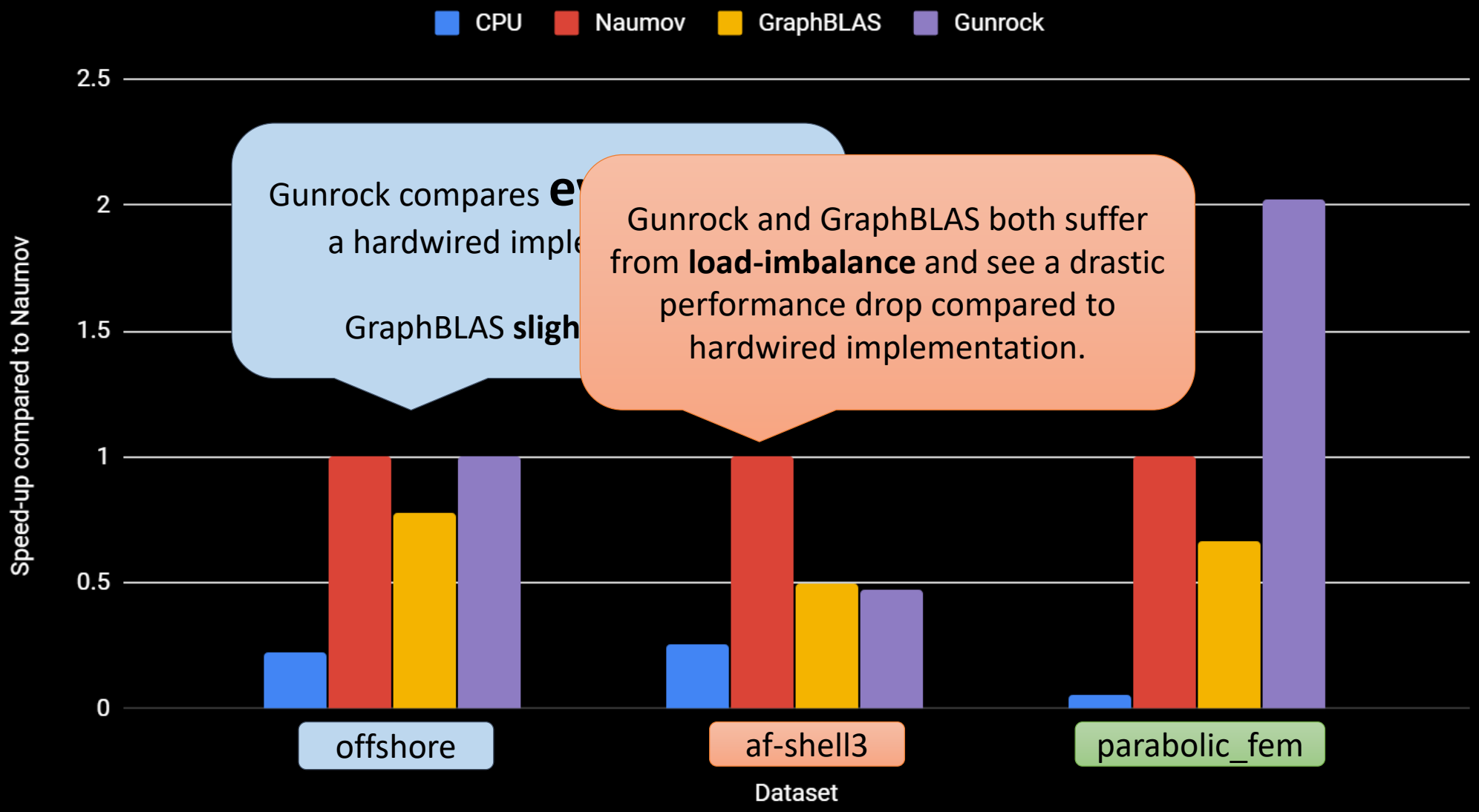


On average, Gunrock is **1.3x** faster than hardwired GPU implementation.

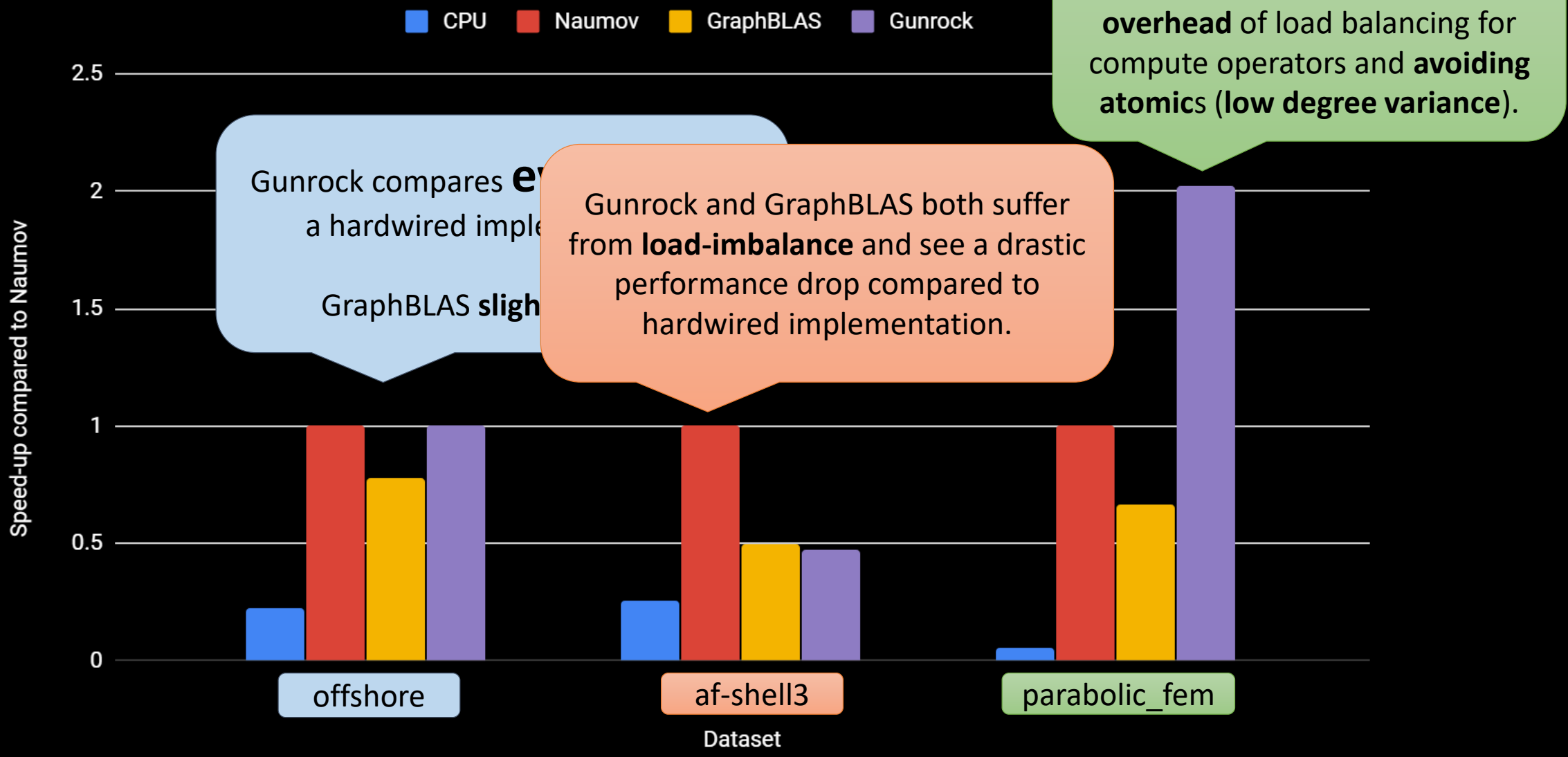
Runtime Comparison (**closer look**)



Runtime Comparison (**closer look**)



Runtime Comparison (**closer look**)



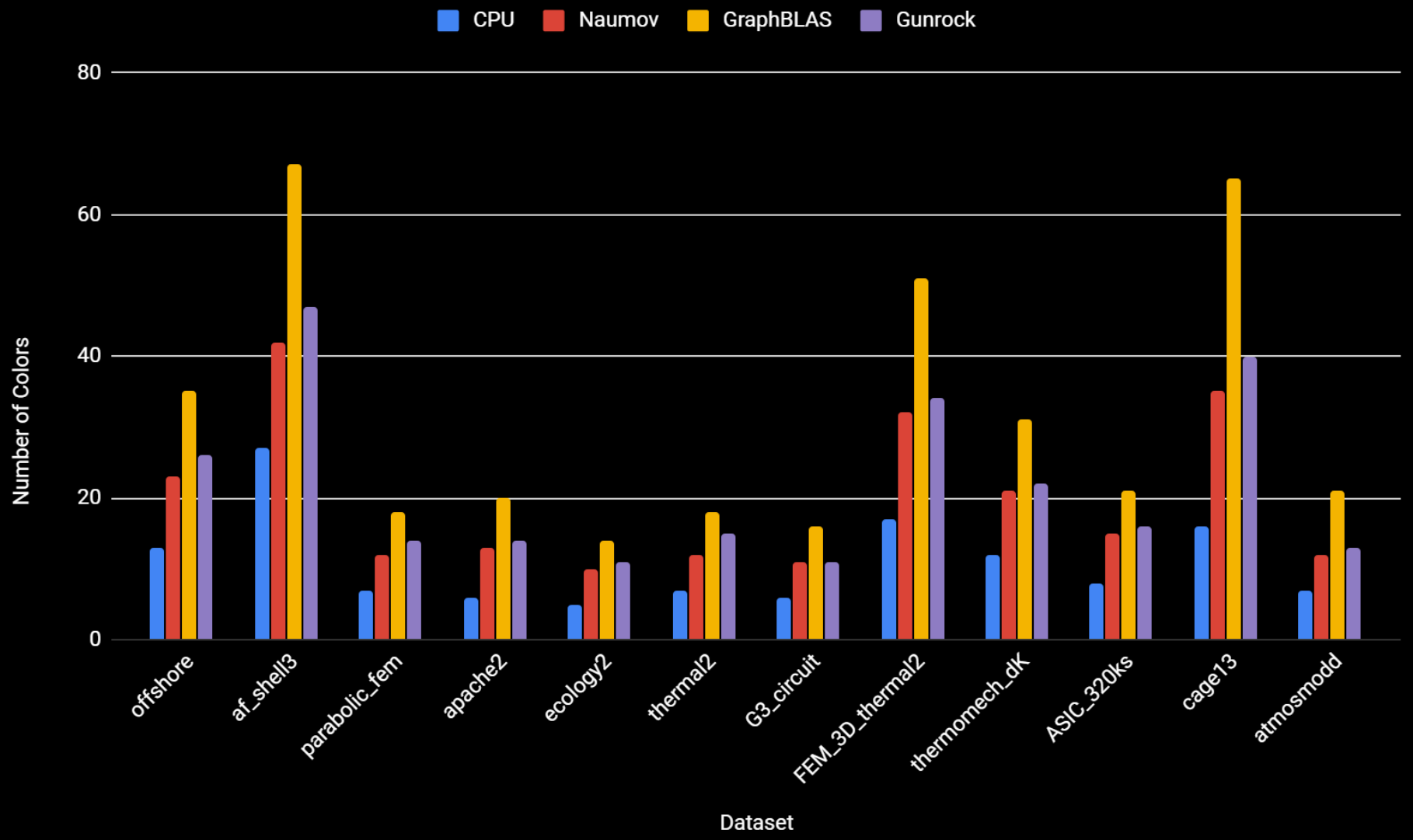
Gunrock compares **ev**
a hardwired imple

GraphBLAS **sligh**

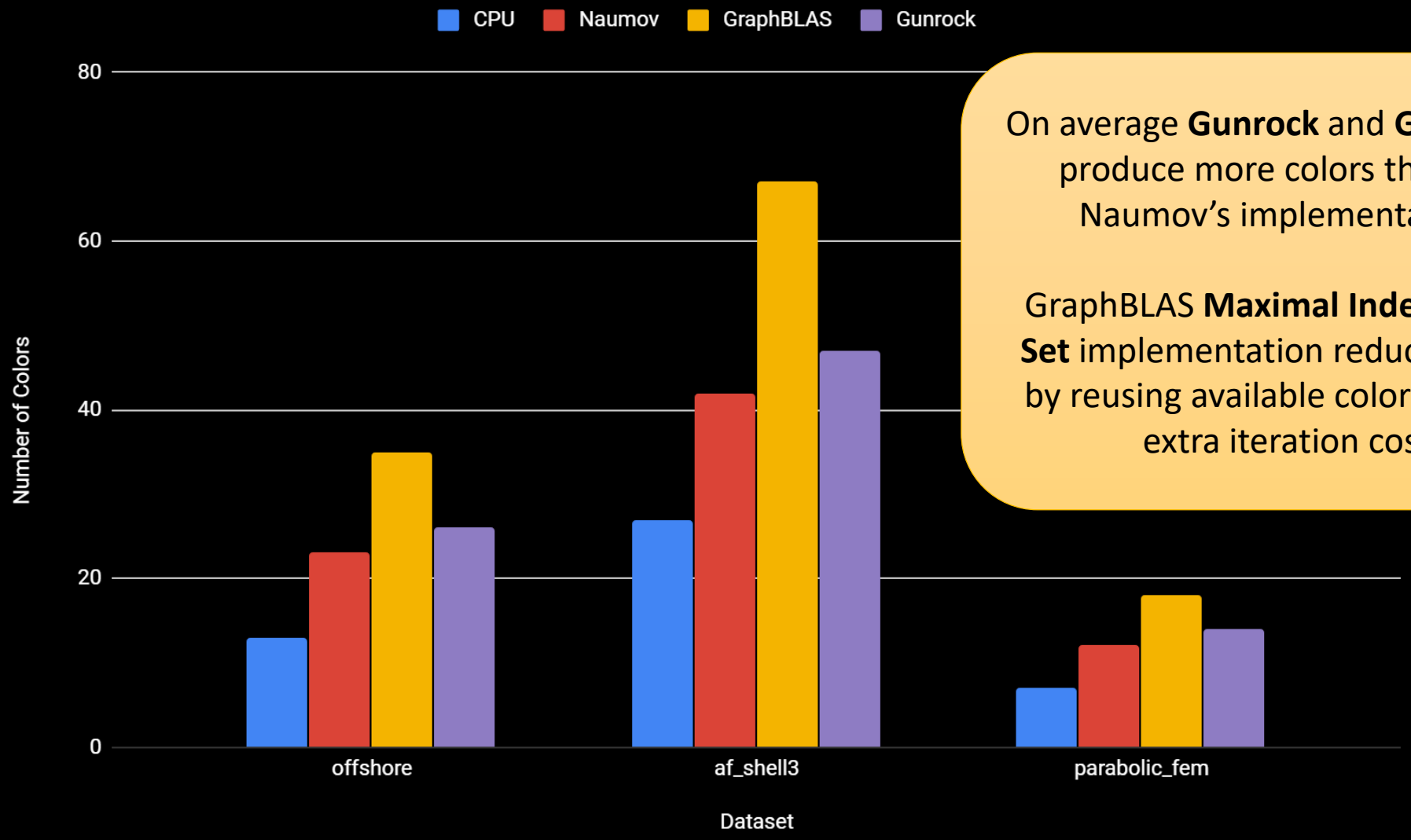
Gunrock and GraphBLAS both suffer from **load-imbalance** and see a drastic performance drop compared to hardwired implementation.

Gunrock does extremely well for a structured dataset due to **low overhead** of load balancing for compute operators and **avoiding atomics** (**low degree variance**).

Number of Colors Comparison



Number of Colors Comparison (closer look)



On average **Gunrock** and **GraphBLAS** produce more colors than the Naumov's implementation.

GraphBLAS **Maximal Independent Set** implementation reduces colors by reusing available colors with an extra iteration cost

Conclusion

1. **Flexibility** of GPU graph frameworks
2. **Performance** against state-of-the-art

Conclusion

1

Implemented 3 different graph coloring **algorithms** with **optimizations** on both **GraphBLAS** and **Gunrock**

2

compared to **hard-wired** state-of-the art implementations Naumov et al.

Gunrock peak **speed-up of 2x**,
a geomean **speed-up of 1.3x** with 1.6x more colors
GraphBLAS **1.9x fewer colors** at a cost of **3x extra run-time**

How to do better…

- **Load-Balance** for scale-free graphs
- **Kernel Fusion** to fuse advance and neighborhood reduction – avoiding cost of two kernels
- Exploring more graph coloring algorithms.

Acknowledgements

- Maxim Naumov for explaining technical details about his implementation

We appreciate the funding support from:

- The Defense Advanced Research Projects Agency (Awards # FA8650-18-2-7835 and HR0011-18-3-0007)
- The National Science Foundation (Awards # OAC-1740333 and CCF-1629657)
- The DOE Office of Advanced Scientific Computing Research under Contract No. DE-AC02-05CH11231
- The Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration

Code and other stuff…

Gunrock

github.com/gunrock/gunrock

(released soon in v1.0.0)

GraphBLAS

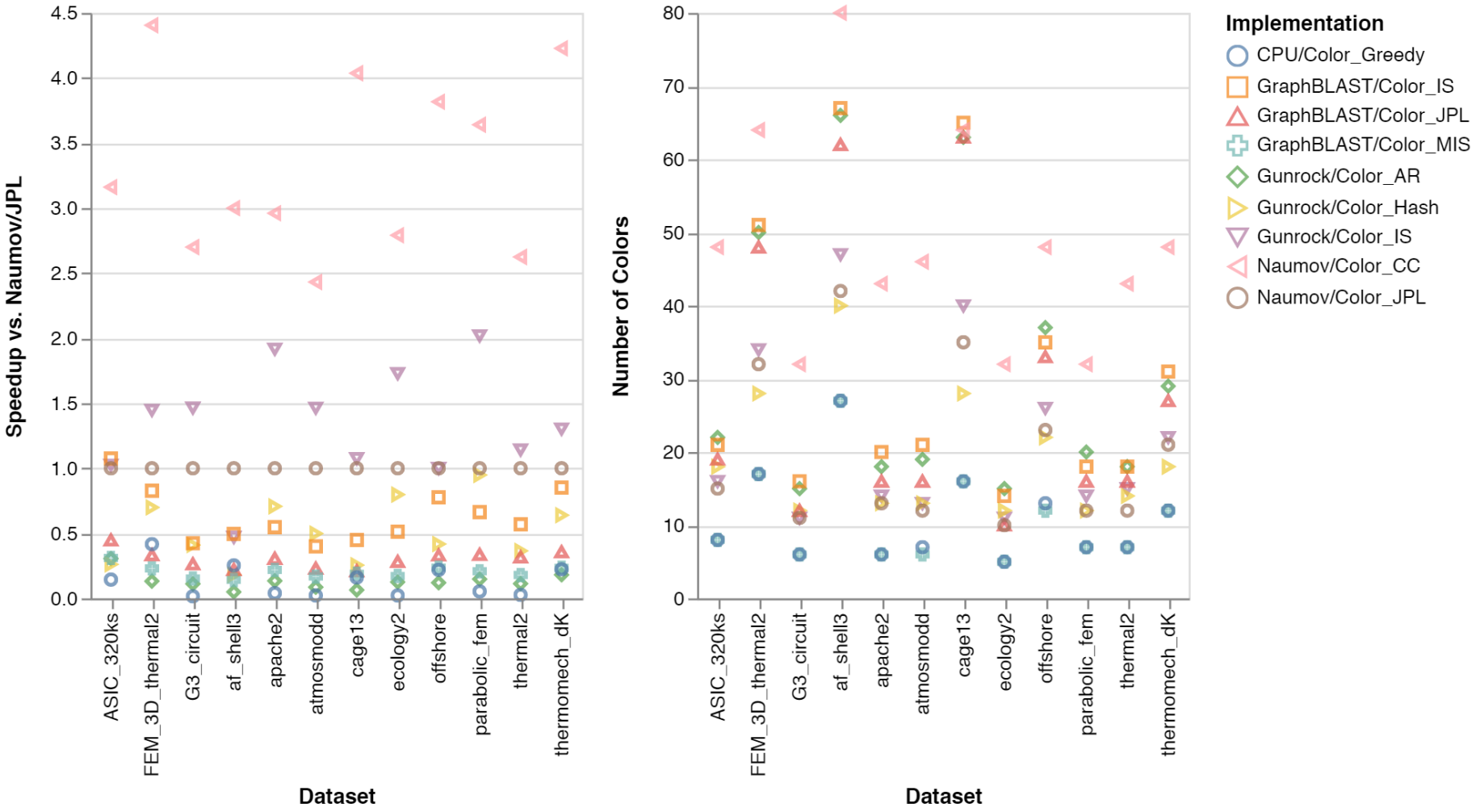
github.com/gunrock/graphblast

mosama@ucdavis.edu

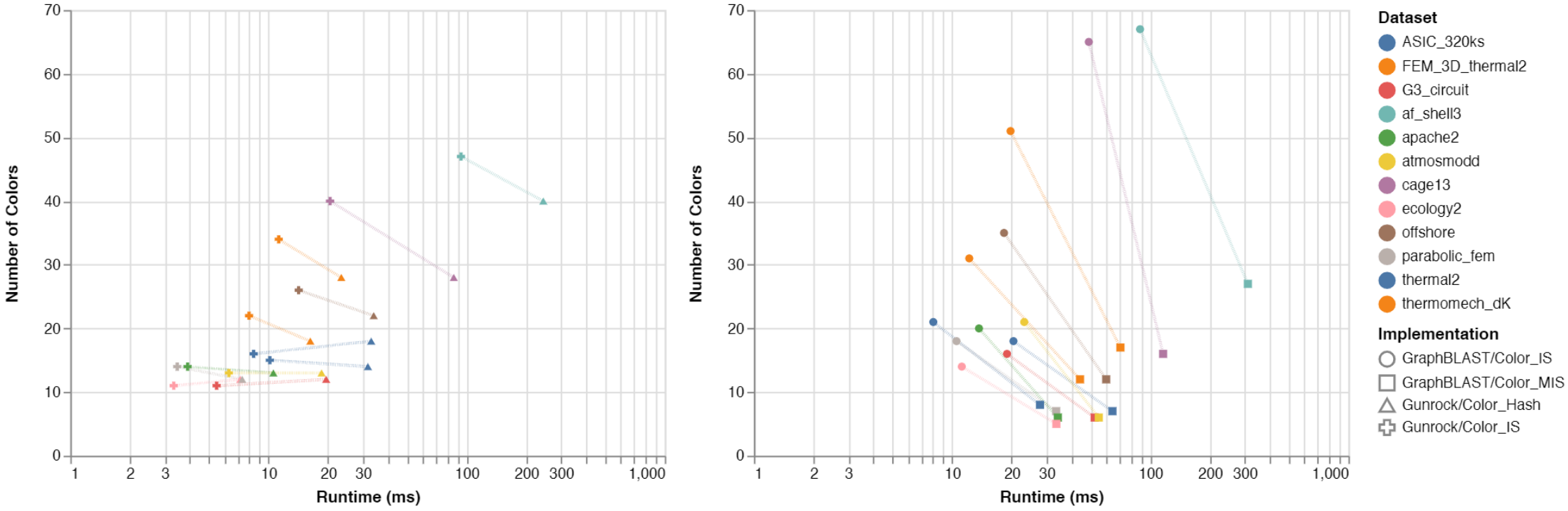
Thank you!

Back-up Slides

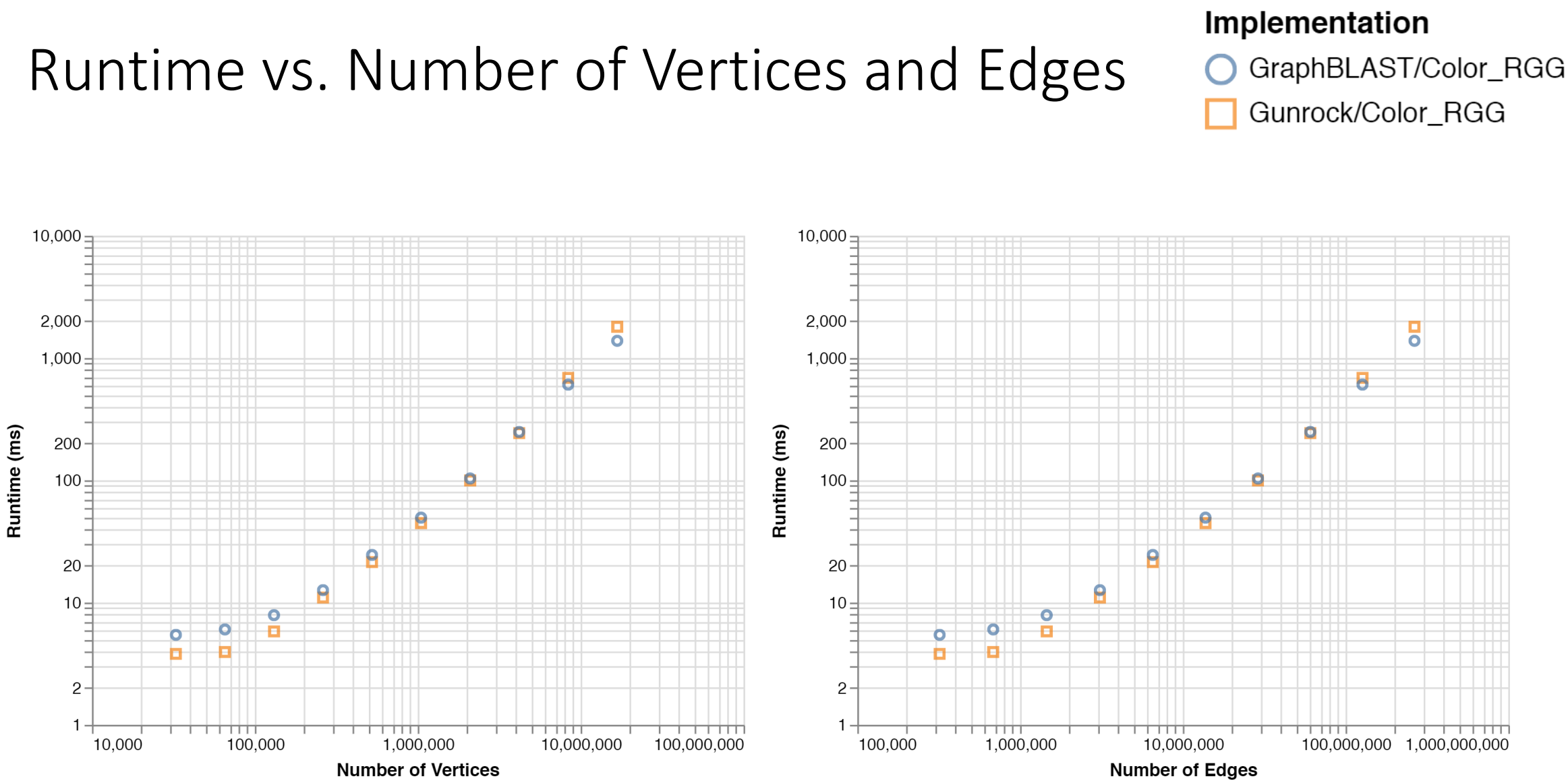
Runtime and Number of Colors Comparison



Tradeoffs Between Different Implementations



Runtime vs. Number of Vertices and Edges



Number of Colors vs. Number of Vertices and Edges