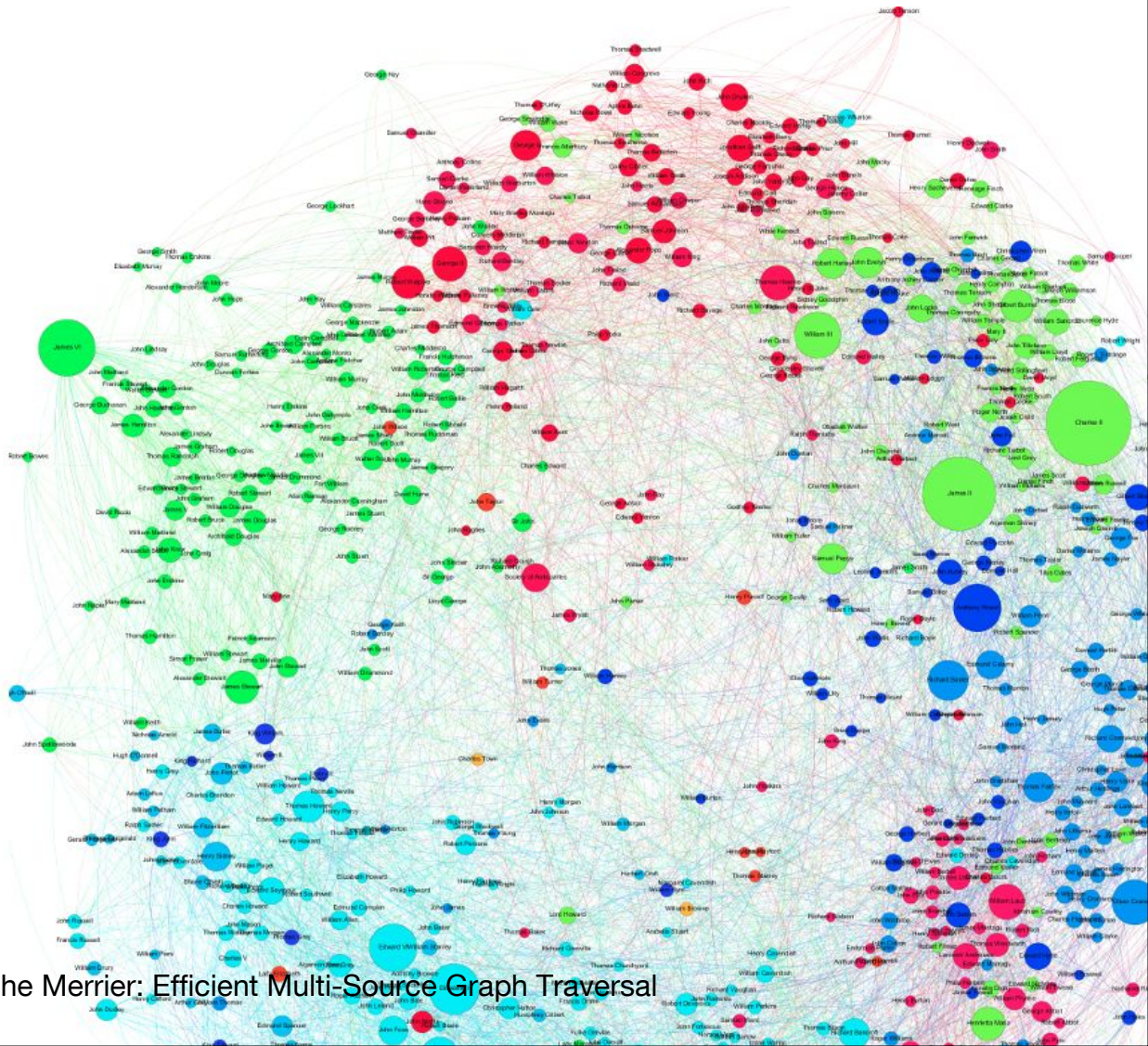
The More the Merrier:
Efficient Multi-Source Graph Traversal

Manuel Then*, Moritz Kaufmann*, Fernando Chirigati†, Tuan-Anh Hoang-Vu†,
Kien Pham†, Huy T. Vo†, Alfons Kemper*, Thomas Neumann*

* Technische Universität München, † New York University

Outline

- Motivation
- Challenges
- Goals
- Multi-Source BFS
- Evaluation
- Summary



Motivation

- Graph traversal vital part of graph analytics
  - BFS, DFS, Neighbor traversals, Random walks, ...

Motivation

- Graph traversal vital part of graph analytics
  - BFS, DFS, Neighbor traversals, Random walks, ...
- Often **multiple BFS traversals** necessary to compute results
  - Closeness centrality, Shortest paths, ...

Motivation

- Graph traversal vital part of graph analytics
  - BFS, DFS, Neighbor traversals, Random walks, ...
- Often **multiple BFS traversals** necessary to compute results
  - Closeness centrality, Shortest paths, ...
- Real-world graphs often are small-world networks
  - Social networks, Web graphs, Communication networks

Motivation

- Graph traversal vital part of graph analytics
  - BFS, DFS, Neighbor traversals, Random walks, ...
- Often **multiple BFS traversals** necessary to compute results
  - Closeness centrality, Shortest paths, ...
- Real-world graphs often are small-world networks
  - Social networks, Web graphs, Communication networks

- Subject of this talk: efficiently run **multiple BFSs on real-world graphs**

Challenges

- **Random data access intrinsic** to graph traversal algorithms
  - bad cache behavior, frequent CPU stalls

Challenges

- **Random data access intrinsic** to graph traversal algorithms
  - bad cache behavior, frequent CPU stalls
- **Single bit accesses** waste memory bandwidth
  - e.g. for BFS seen bitmaps

Challenges

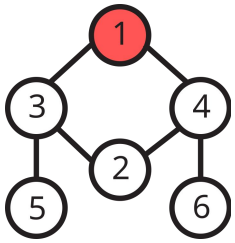
- **Random data access intrinsic** to graph traversal algorithms
  - bad cache behavior, frequent CPU stalls
- **Single bit accesses** waste memory bandwidth
  - e.g. for BFS seen bitmaps
- Independent BFS runs **redundantly visit vertices** multiple times

Challenge - Redundant visits

Example: BFSs in a simple graph

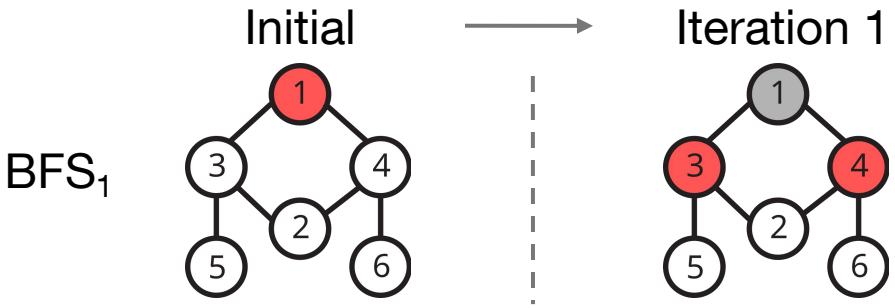
Initial

BFS_1



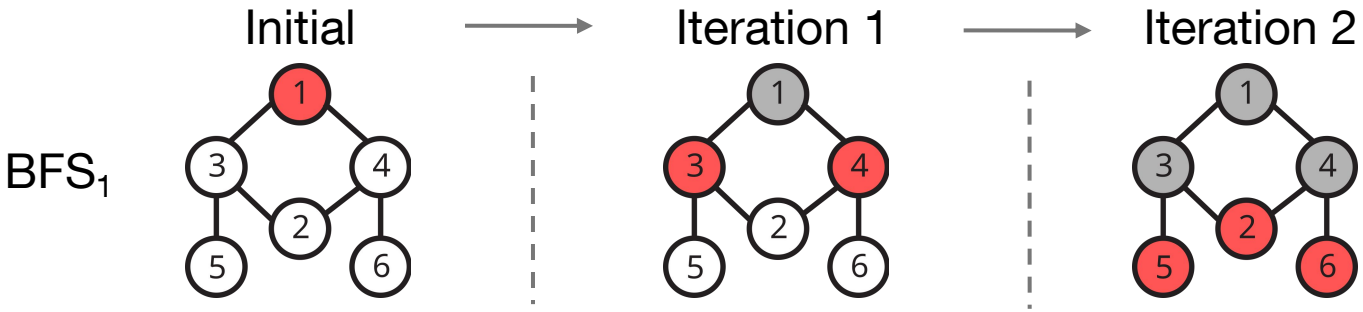
Challenge - Redundant visits

Example: BFSs in a simple graph



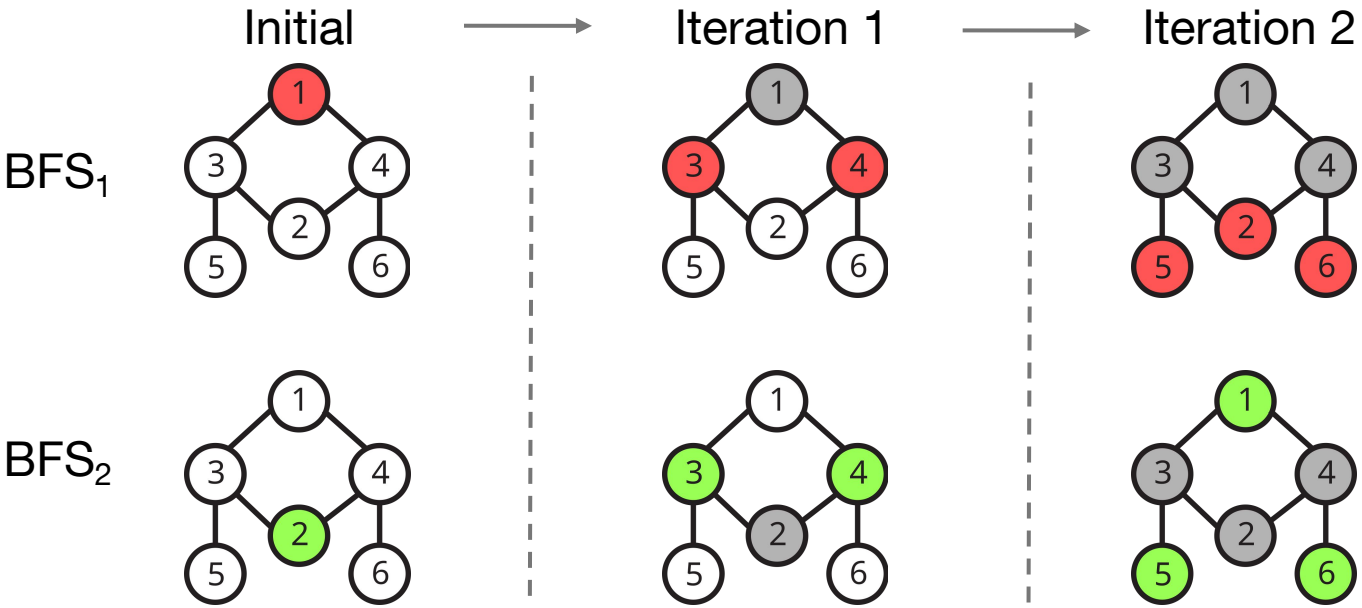
Challenge - Redundant visits

Example: BFSs in a simple graph



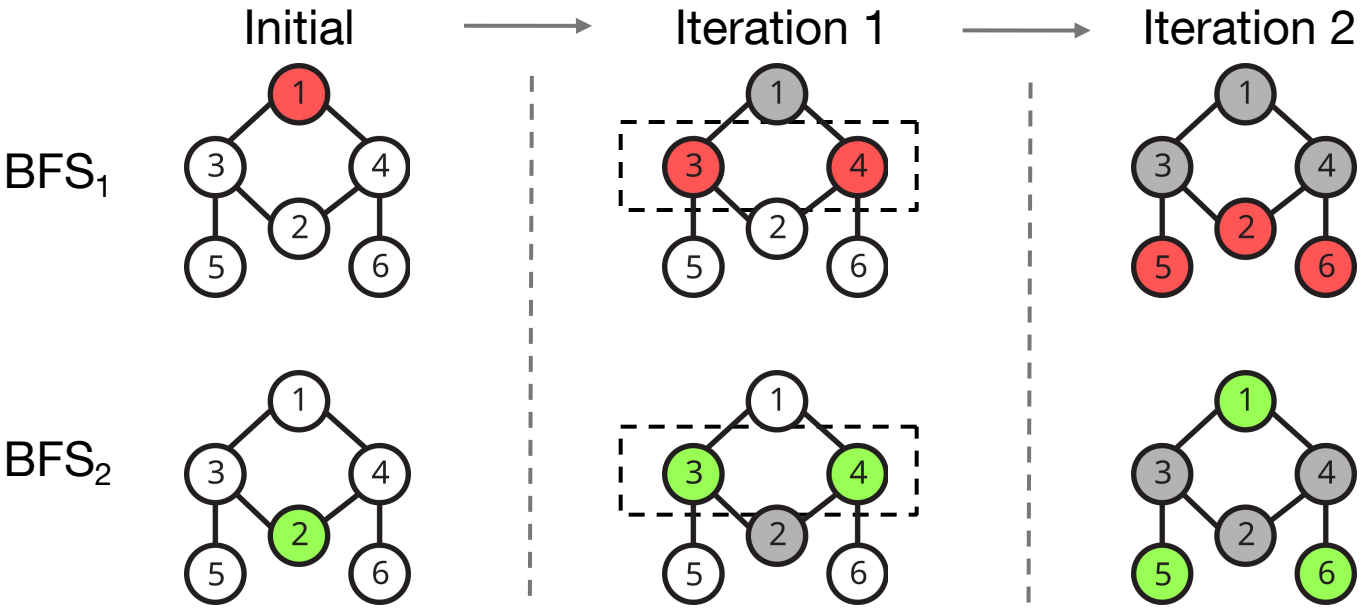
Challenge - Redundant visits

Example: BFSs in a simple graph



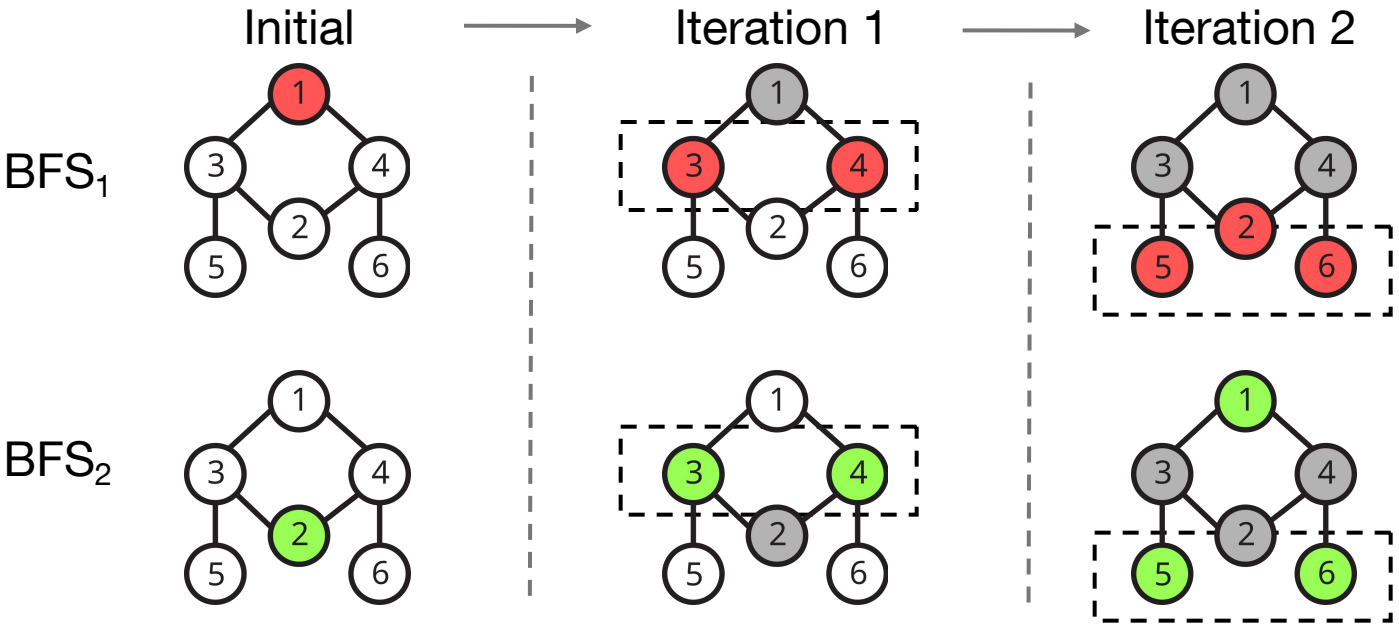
Challenge - Redundant visits

Example: BFSs in a simple graph



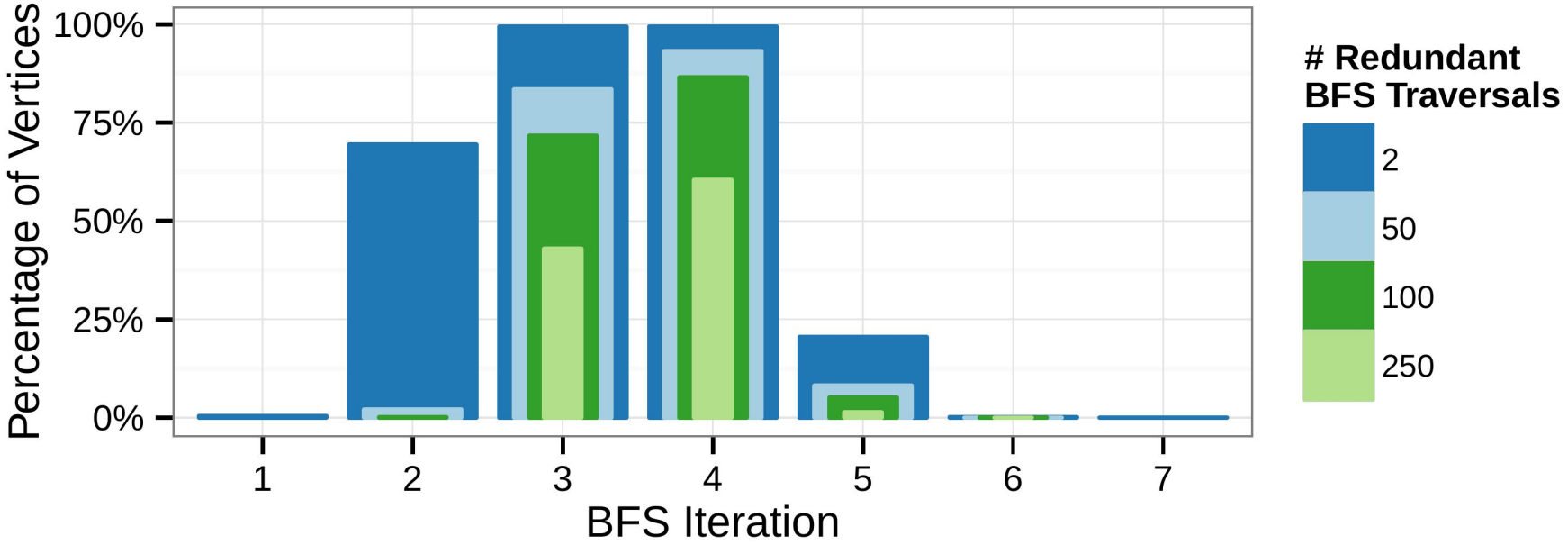
Challenge - Redundant visits

Example: BFSs in a simple graph



Challenge - Redundant visits (cont.)

Redundant vertex visits for 512 BFSs on LDBC 1M social network graph



- After a few iterations, **many redundant visits** in small-world networks

Goals

- Leverage knowledge that **multiple BFS traversal** are run

Goals

- Leverage knowledge that **multiple BFS traversal** are run
- Optimize data access patterns
  - **embrace memory accesses** instead of trying to hide them
  - CPUs always fetch full cache lines - use all of them

Goals

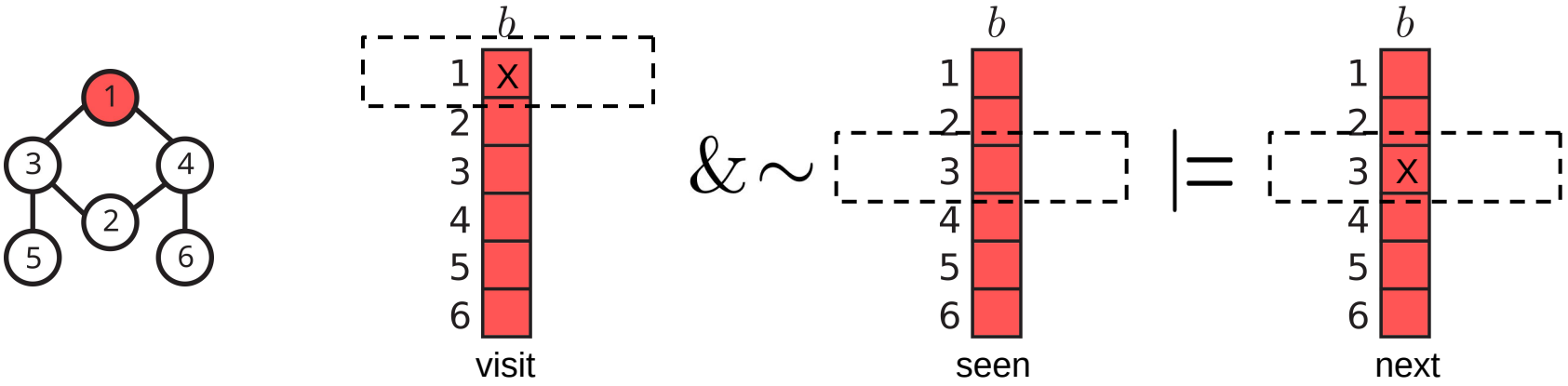
- Leverage knowledge that **multiple BFS traversal** are run
- Optimize data access patterns
  - **embrace memory accesses** instead of trying to hide them
  - CPUs always fetch full cache lines - use all of them
- **Avoid redundant computation** and vertex visits
  - touch vertex information as rarely as possible

Multi-Source BFS

- Concurrently **run many independent BFS traversals** on the same graph
  - 100s of BFSs on a single CPU core

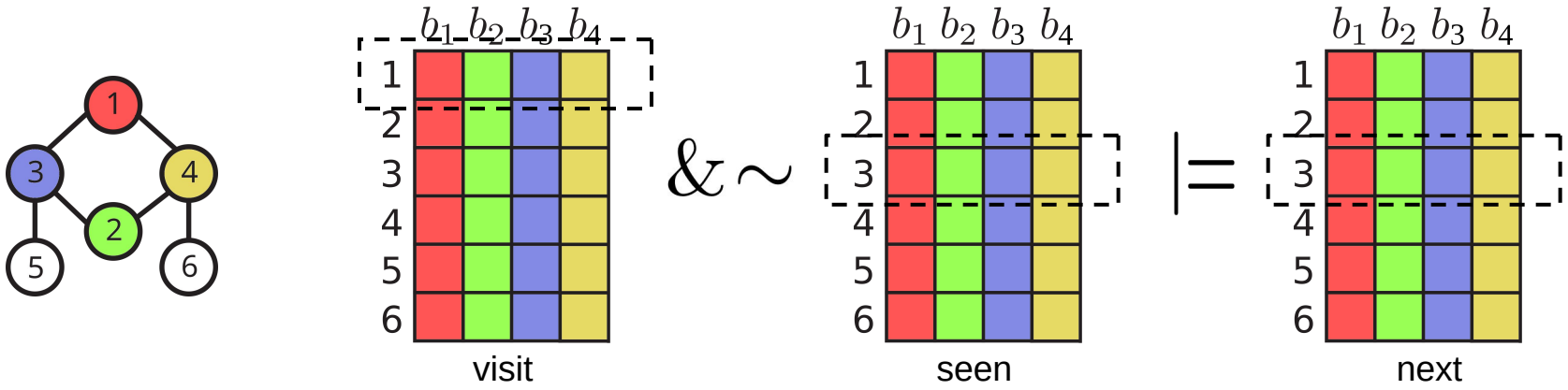
Multi-Source BFS

- Concurrently **run many independent BFS traversals** on the same graph
  - 100s of BFSs on a single CPU core



Multi-Source BFS

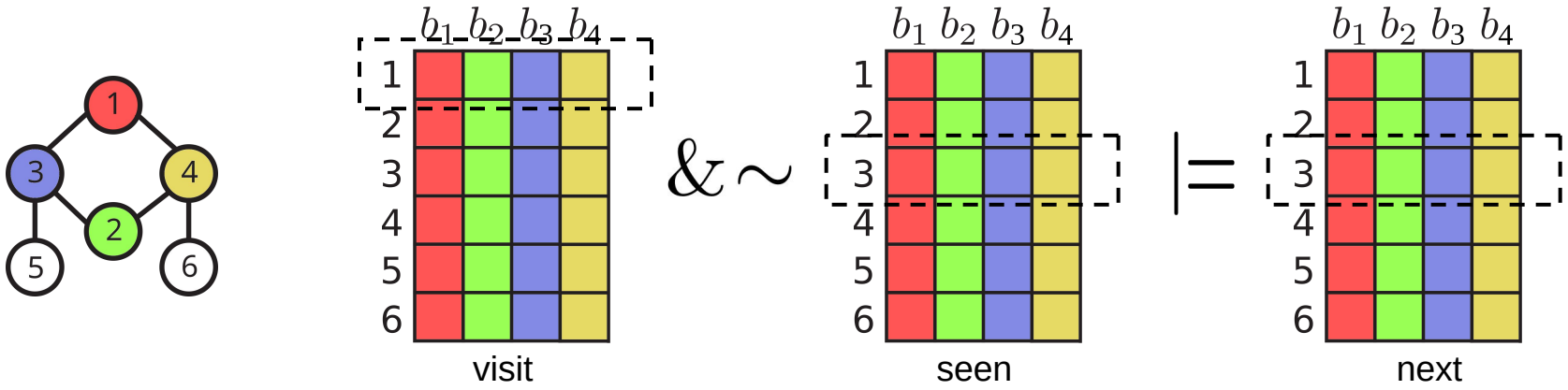
- Concurrently **run many independent BFS traversals** on the same graph
  - 100s of BFSs on a single CPU core
- Store concurrent **BFSs state as 3 bitsets** per vertex



- Represent **BFS traversal as SIMD bit operations** on these bitsets

Multi-Source BFS

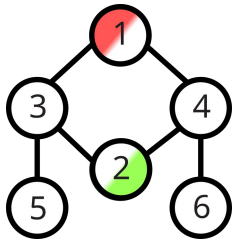
- Concurrently **run many independent BFS traversals** on the same graph
  - 100s of BFSs on a single CPU core
- Store concurrent **BFSs state as 3 bitsets** per vertex

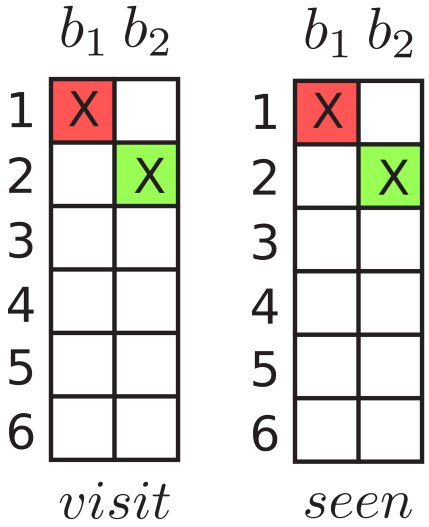


- Represent **BFS traversal as SIMD bit operations** on these bitsets
- Fully utilize cache line-sized memory accesses of modern CPUs
- Efficiently **share traversals** whenever possible
  - neighbors traversed only **once** for all concurrent BFSs

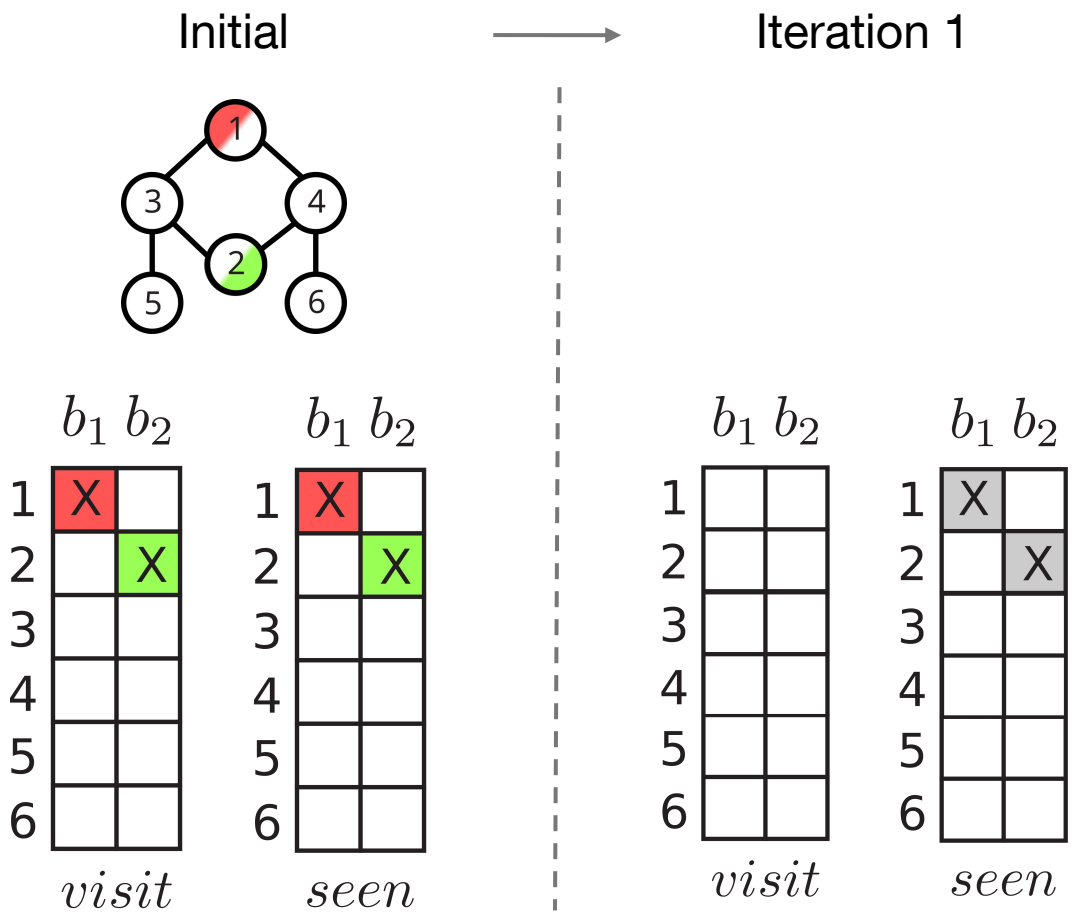
Multi-Source BFS - Example

Initial

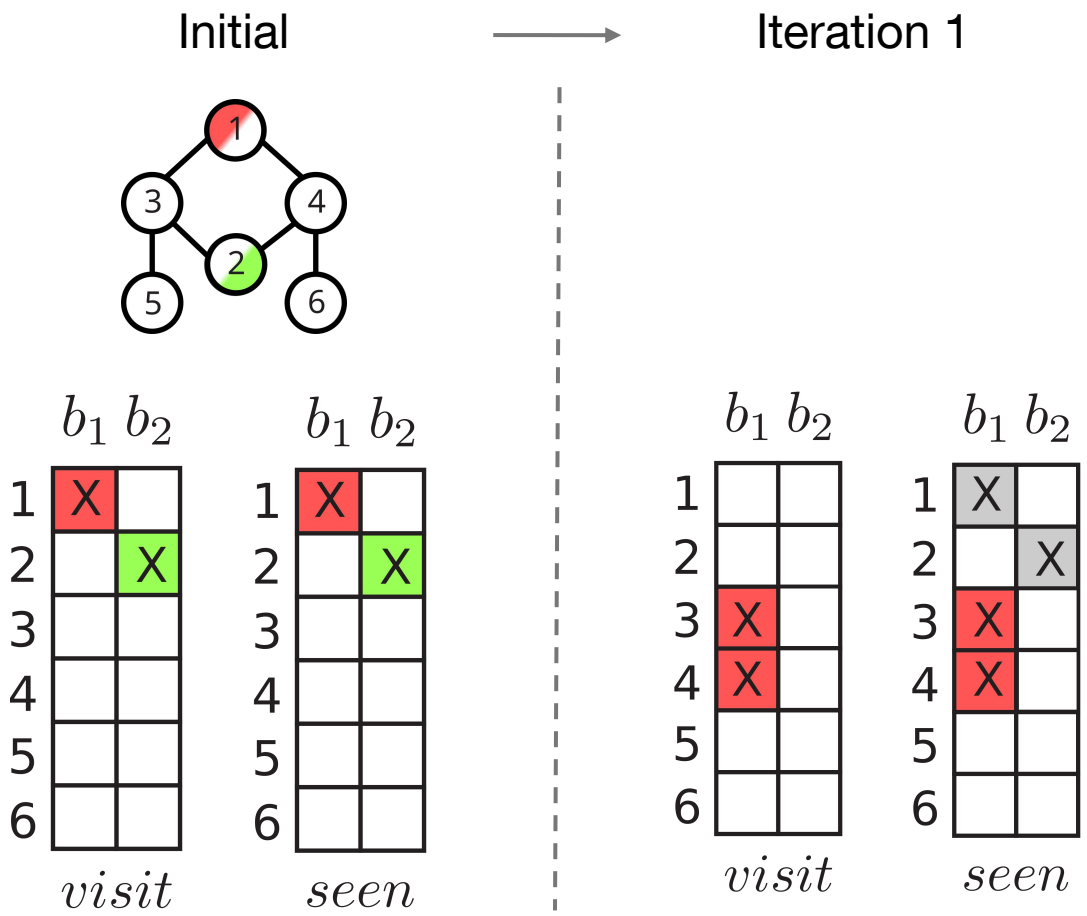




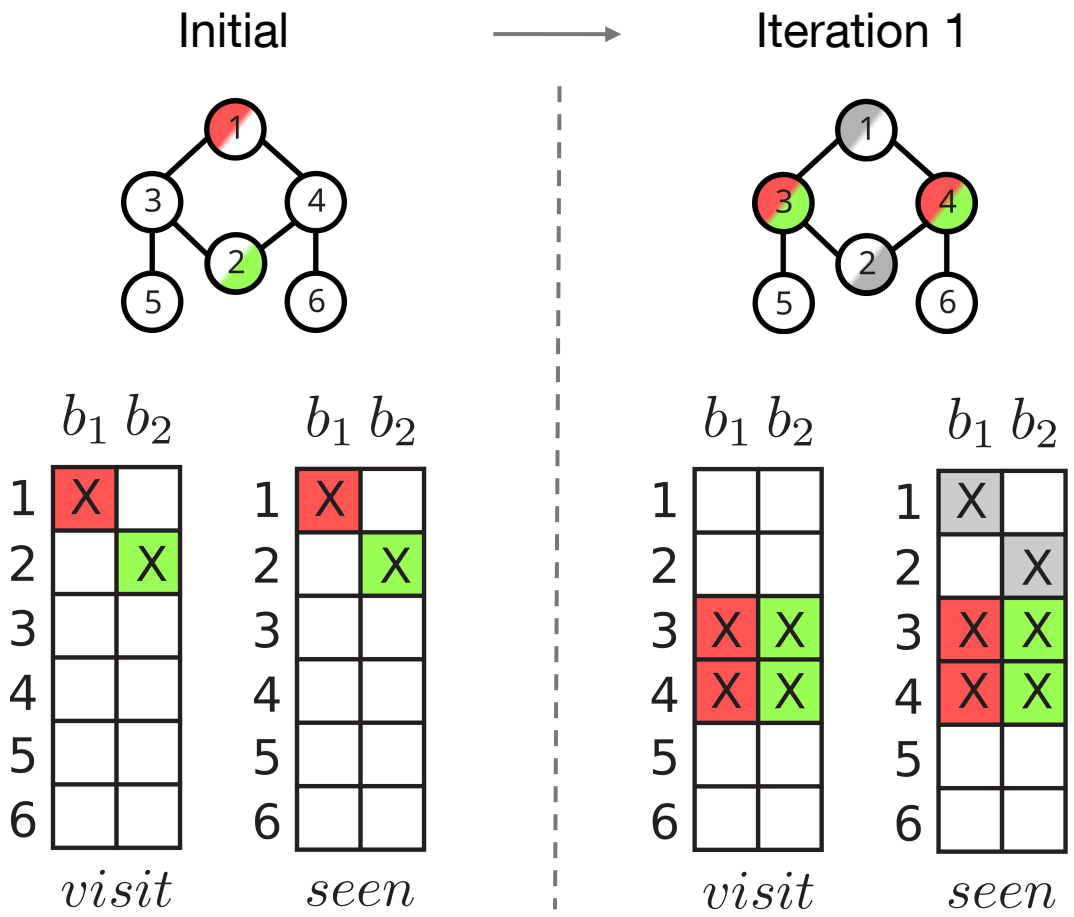
Multi-Source BFS - Example



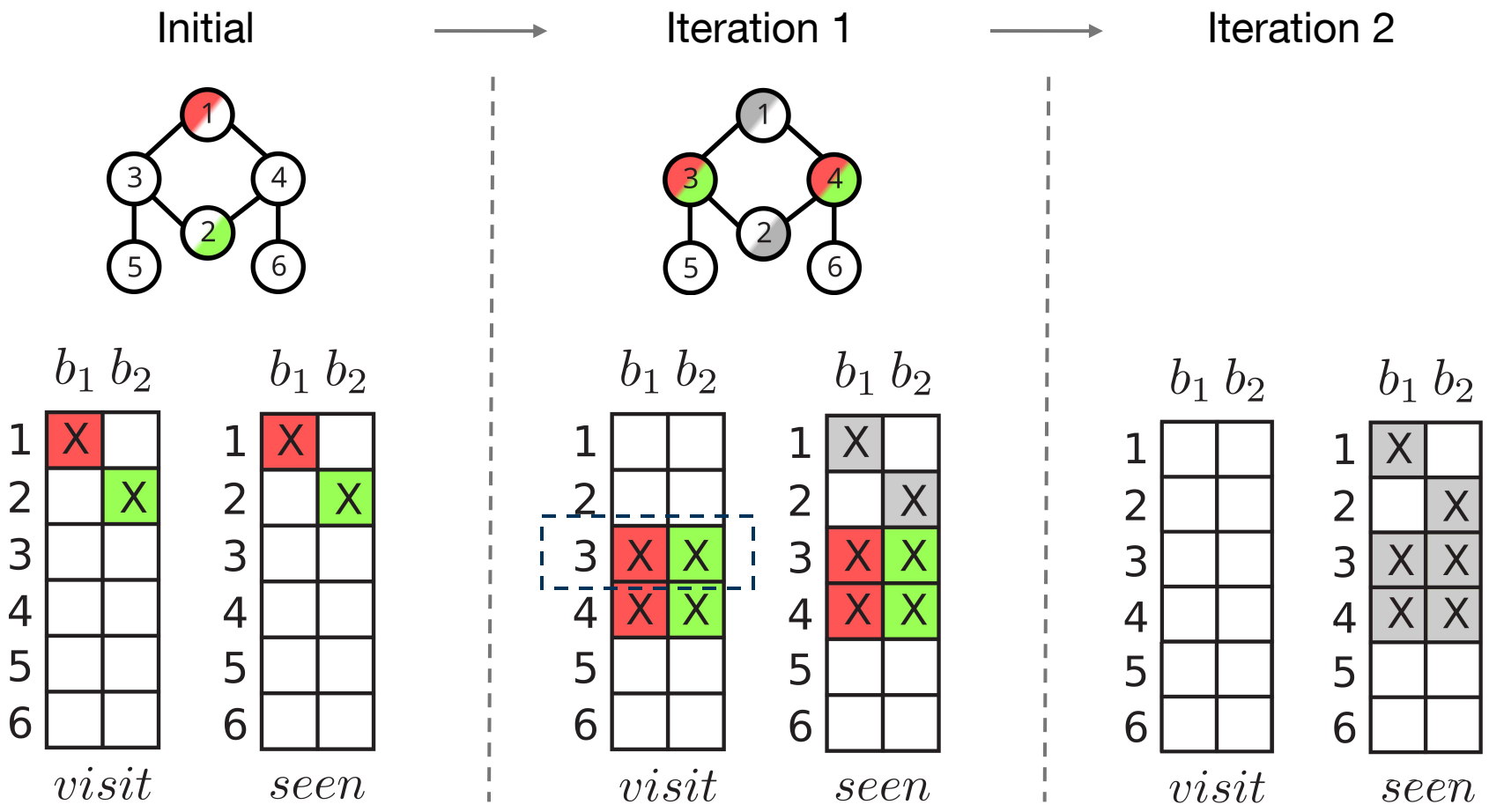
Multi-Source BFS - Example



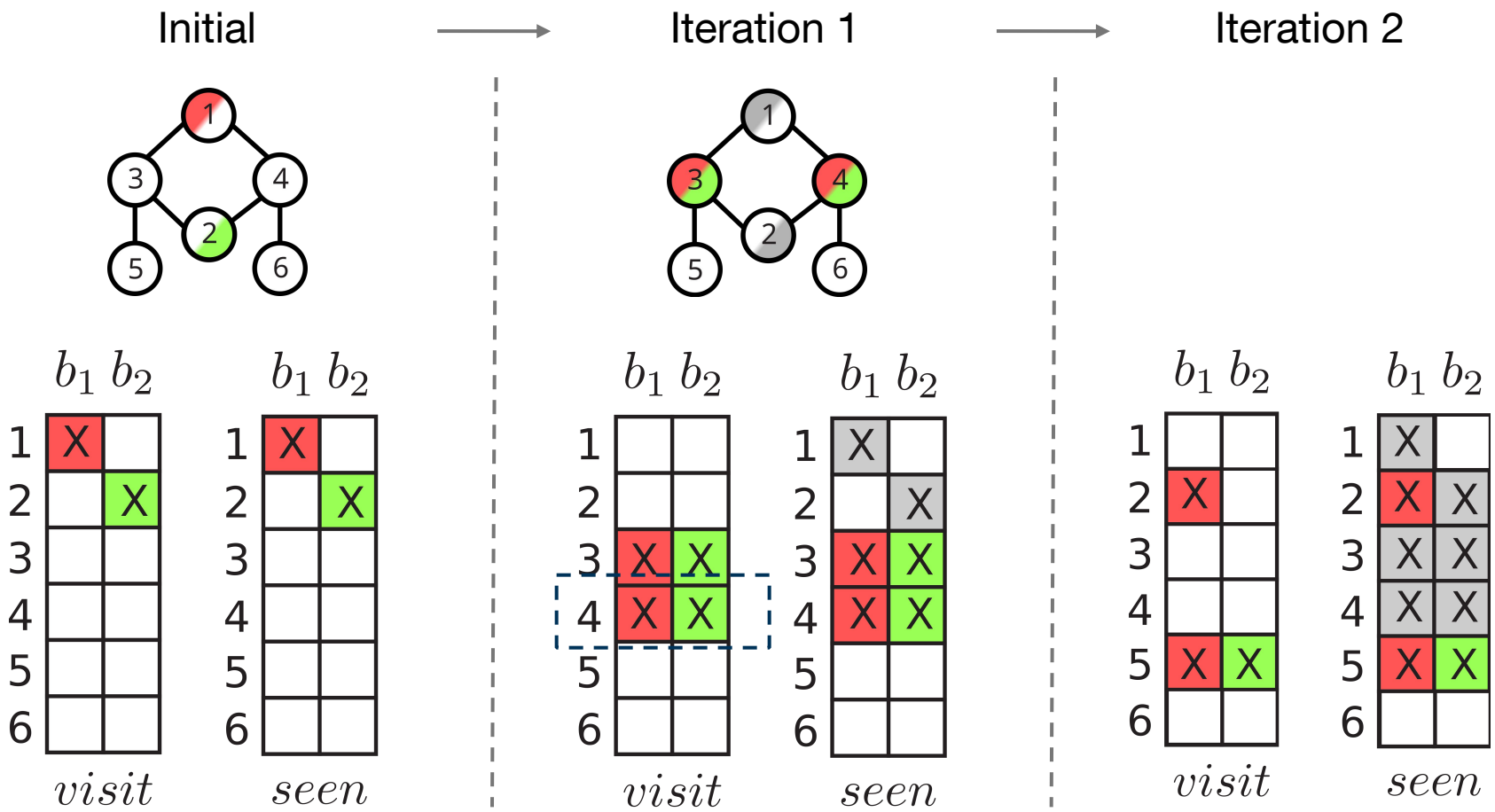
Multi-Source BFS - Example



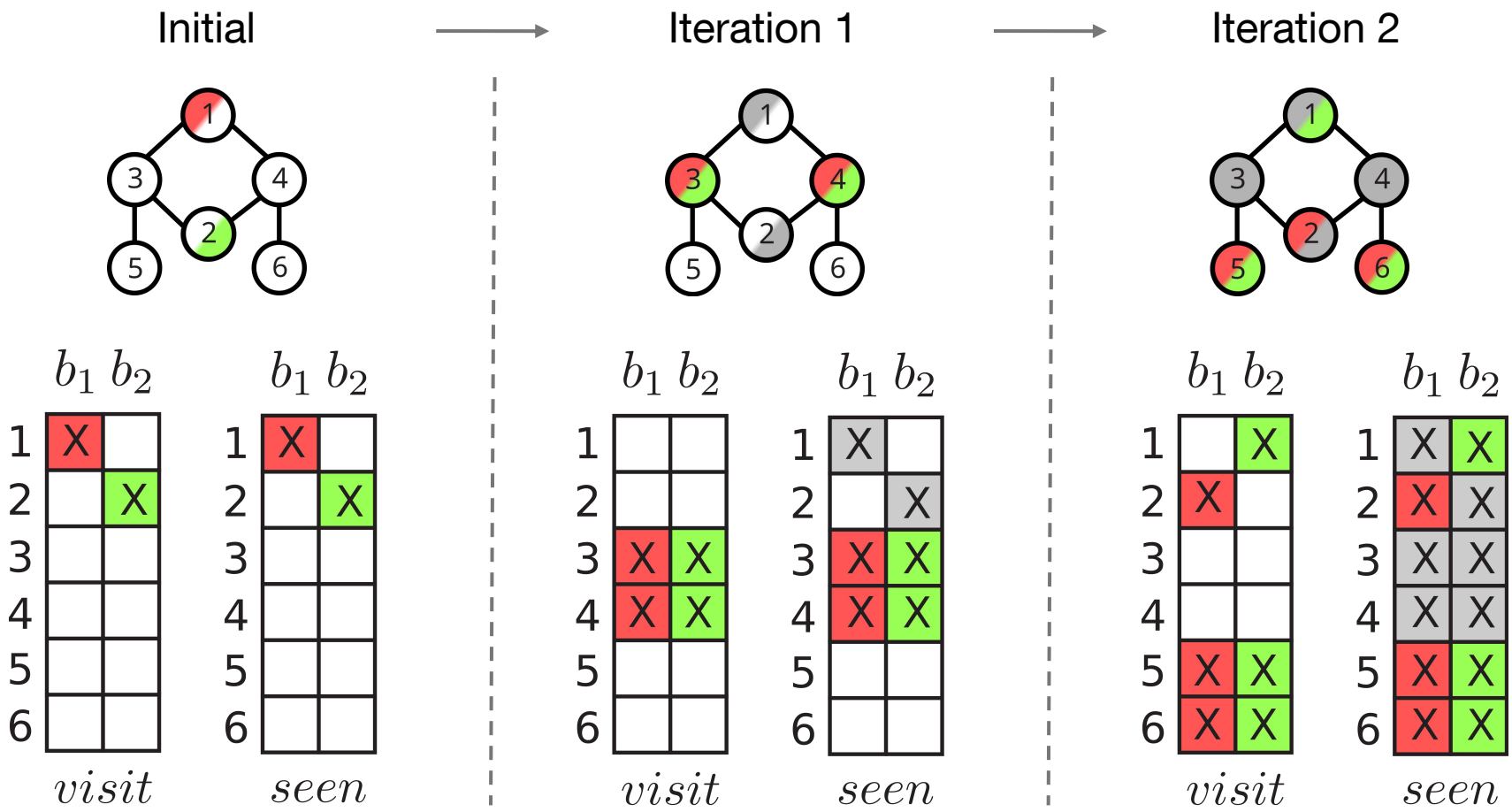
Multi-Source BFS - Example



Multi-Source BFS - Example



Multi-Source BFS - Example

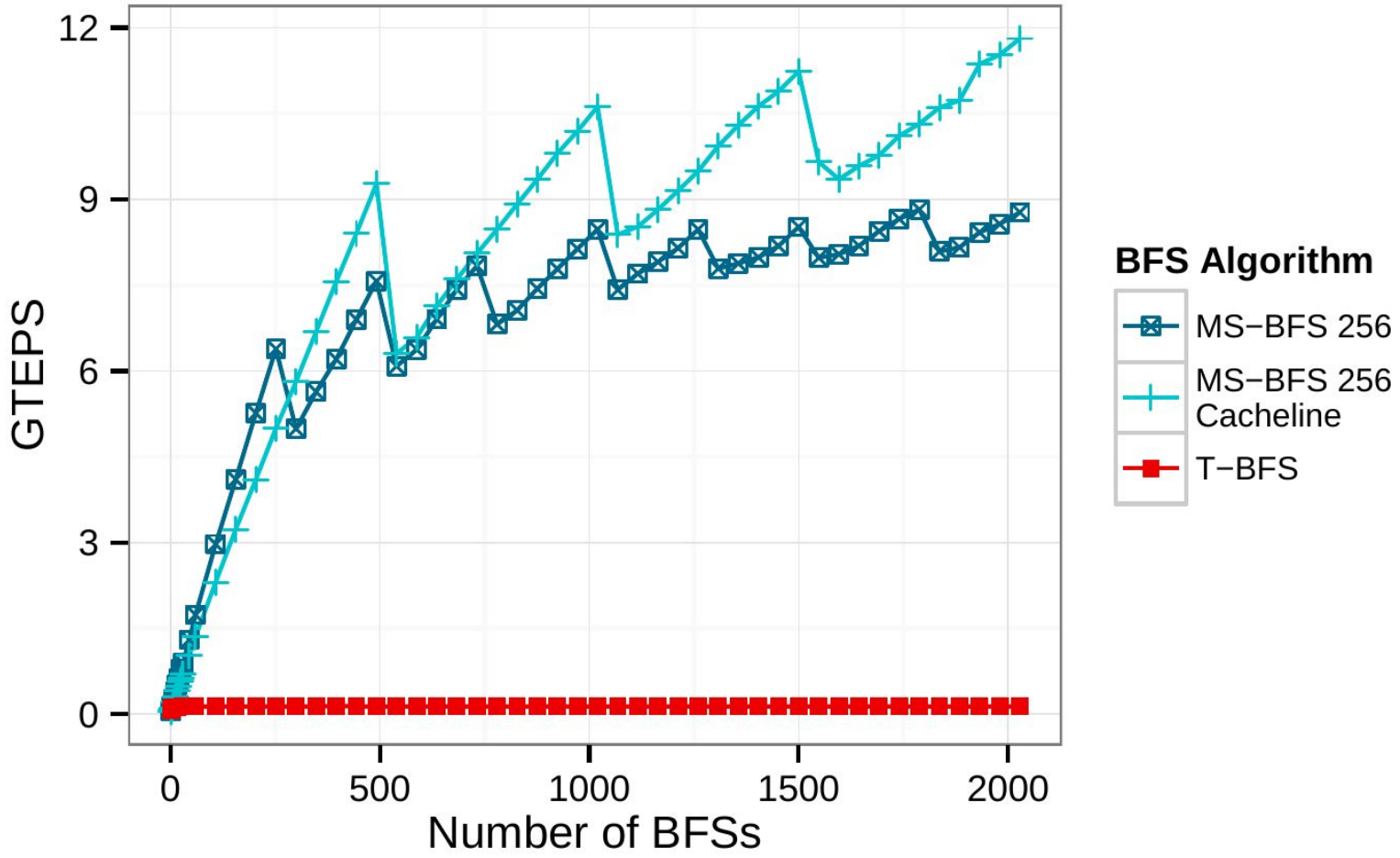


Multi-Source BFS - Further Improvements

- Aggregated neighbor processing
  - reduce number of random writes
- Batching heuristics for maximum sharing
- Direction-optimizing
- Prefetching

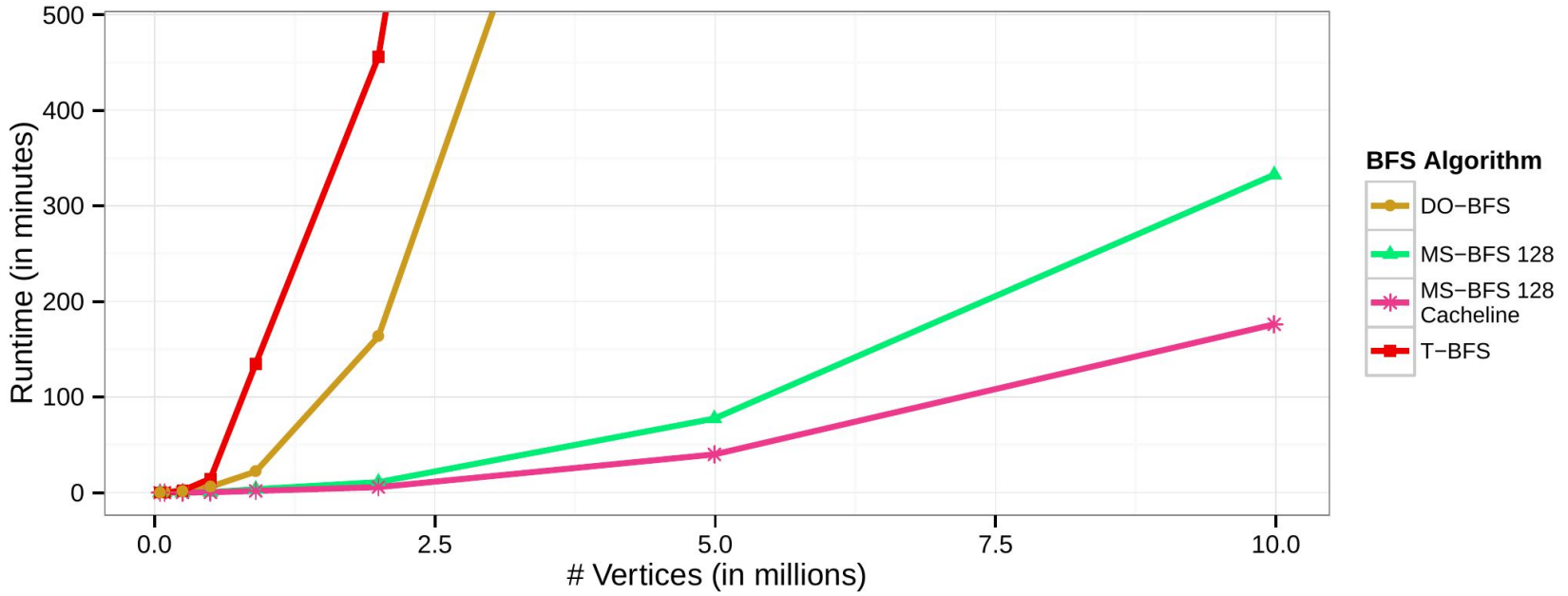
... see paper

Evaluation - The More the Merrier



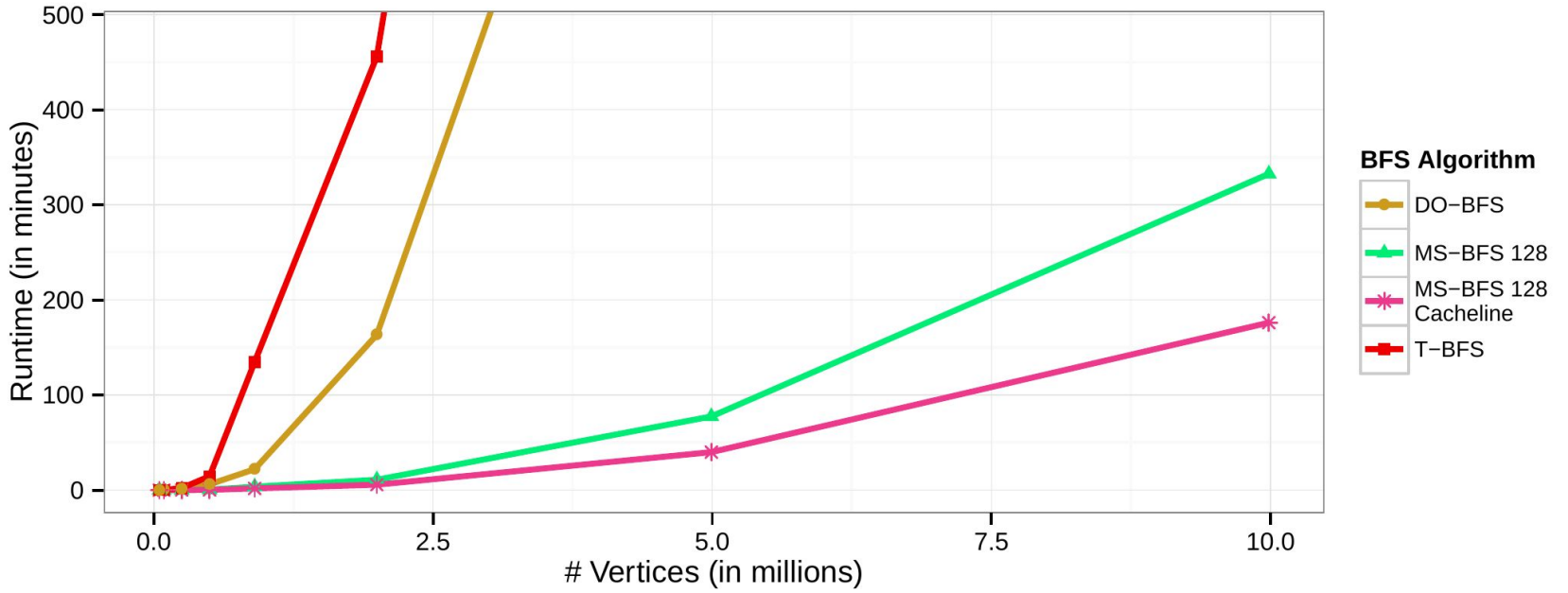
Evaluation

- MS-BFS-based closeness centrality. 4x Intel Xeon E7-4870v2, 1TB



Evaluation

- MS-BFS-based closeness centrality. 4x Intel Xeon E7-4870v2, 1TB



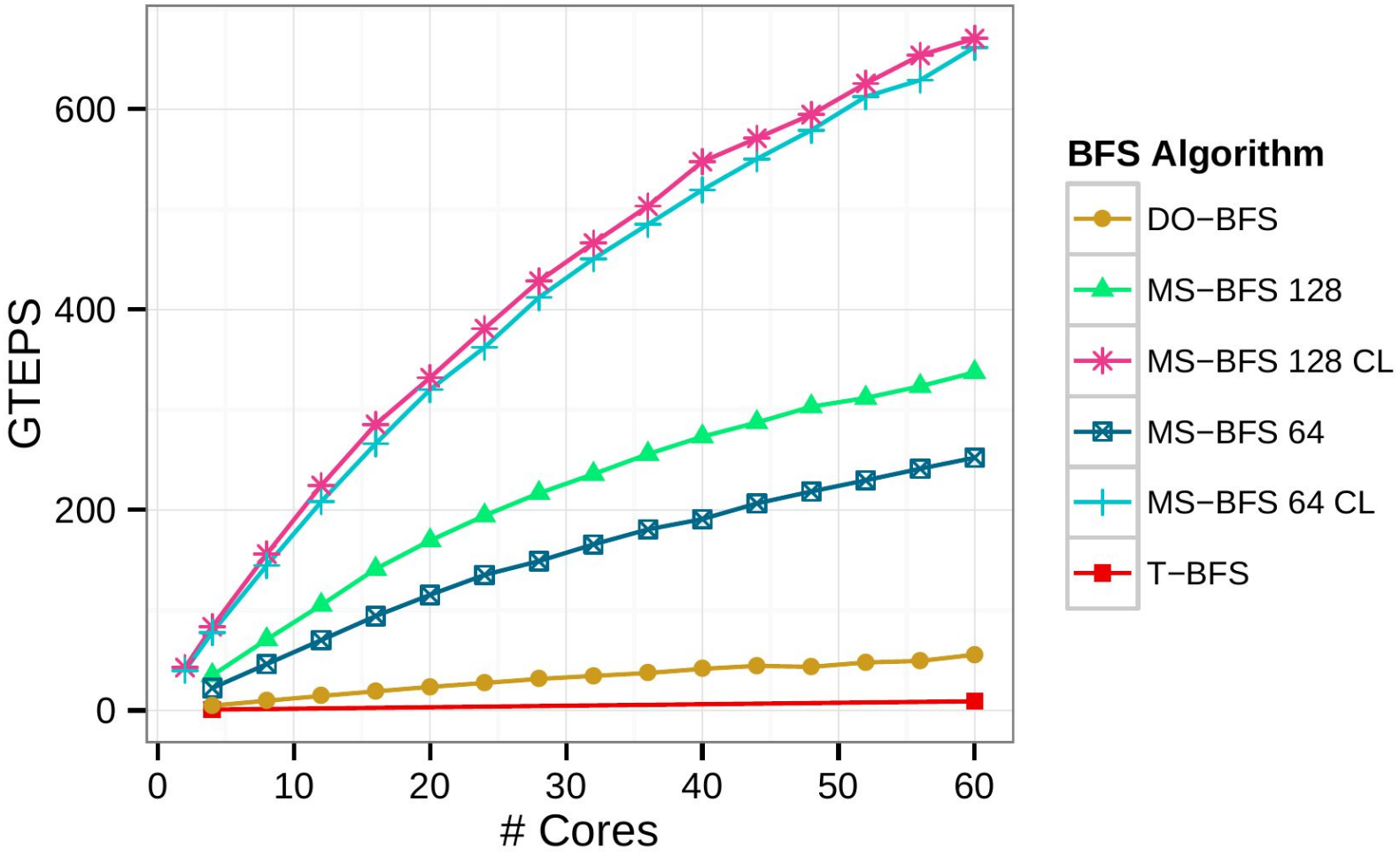
Graph	MS-BFS	Speedup over T-BFS	Speedup over DO-BFS
LDBC 1M	0:02h	73.8x	12.1x
LDBC 10M	2:56h	88.5x	28.7x
Wikipedia	0:26h	75.4x	29.5x
Twitter (1M)	2:52h	54.6x	12.7x

Summary

- Making graph traversals aware of each other can lead to substantial performance increase
- Multi-Source BFS (MS-BFS) runs **multiple independent BFSs** ...
  - ... on the same graph ...
  - ... concurrently on a single CPU ...
  - ... and shares their traversals.
- MS-BFS shows **10-100x speedup** over existing single-source BFSs

Source available at
https://github.com/mtodat/ms-bfs

Backup 1


GTEPS
600
400
200
0
0
10
20
30
40
50
60
Cores
BFS Algorithm
DO−BFS
MS−BFS 128
MS−BFS 128 CL
MS−BFS 64
MS−BFS 64 CL
T−BFS

Backup 2

Table 2: Memory consumption of MS-BFS for N vertices, ω-sized bit fields, and P parallel runs.

N	ω	P	Concurrent BFSs	Memory
1,000,000	64	1	64	22.8 MB
1,000,000	64	16	1,024	366.2 MB
1,000,000	64	64	4,096	1.4 GB
1,000,000	512	1	512	183.1 MB
1,000,000	512	16	8,192	2.9 GB
1,000,000	512	64	32,768	11.4 GB
50,000,000	64	64	4,096	71.5 GB
50,000,000	512	64	32,768	572.2 GB