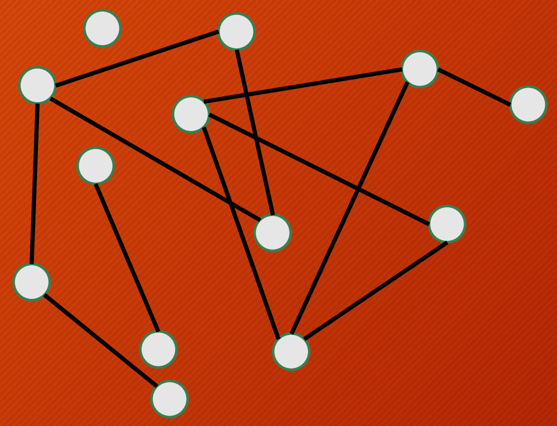
Parallel Connected Components

The Parallelepipeds:
Fabian Meier
Gustavo Segovia
Seraiah Walter

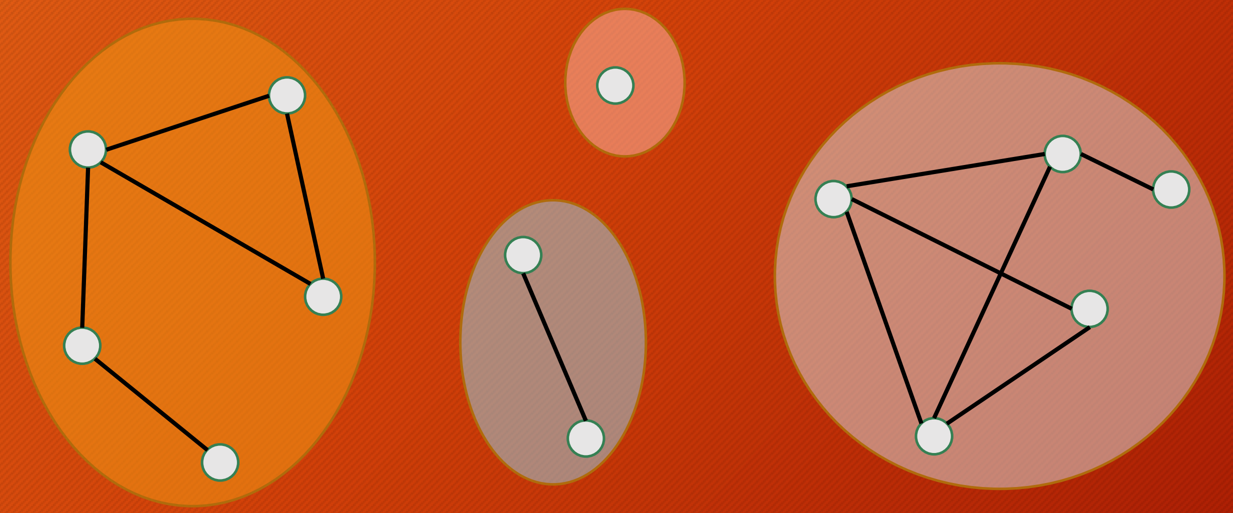
Our goal

- Develop a high performance algorithm for connected components
- Graph fits in memory



Our goal

- Develop a high performance algorithm for connected components
- Graph fits in memory



Algorithms – Related work

- Basic serial
  - Traversal (BFS/DFS) – O(n + m)
  - Union find – O(m log(n))
- Boost
  - Serial
  - Parallel
- Randomized Contraction Parallel Connected Components
  - http://www3.cs.stonybrook.edu/~rezaul/Spring-2012/CSE613/CSE613-lecture-11.pdf

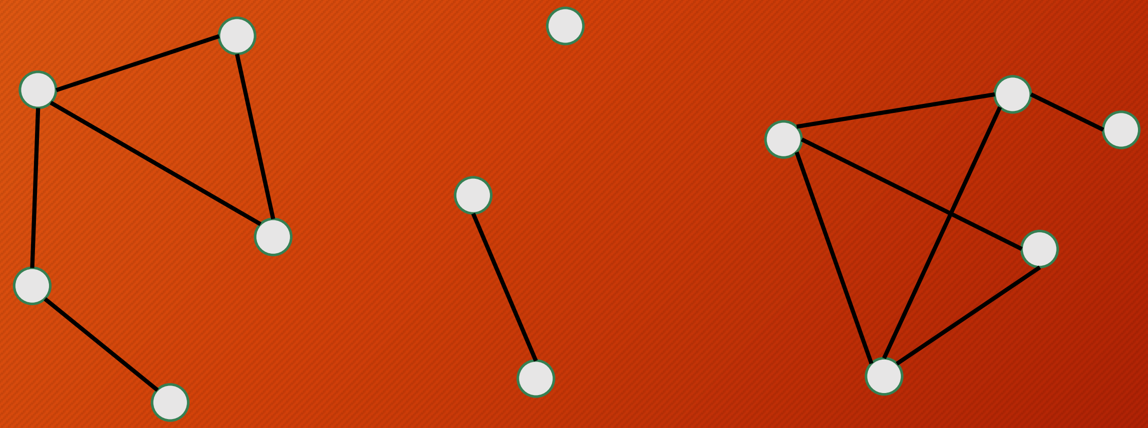
Algorithms – Ours

- Lock free parallel traversal
  - Parallel BFS
  - Parallel BFS with atomics
- Parallel spanning tree – inspired by union find

Algorithm details

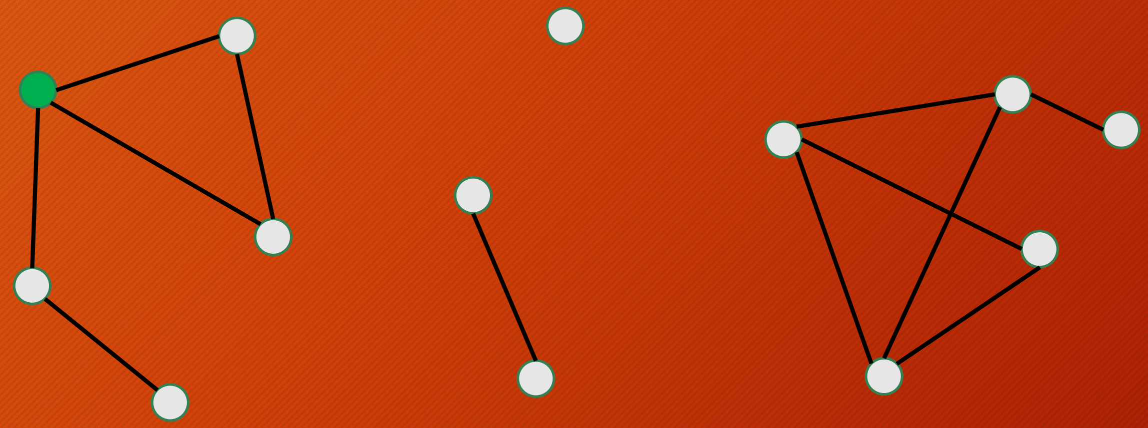
Serial traversal

- Unmarked vertices



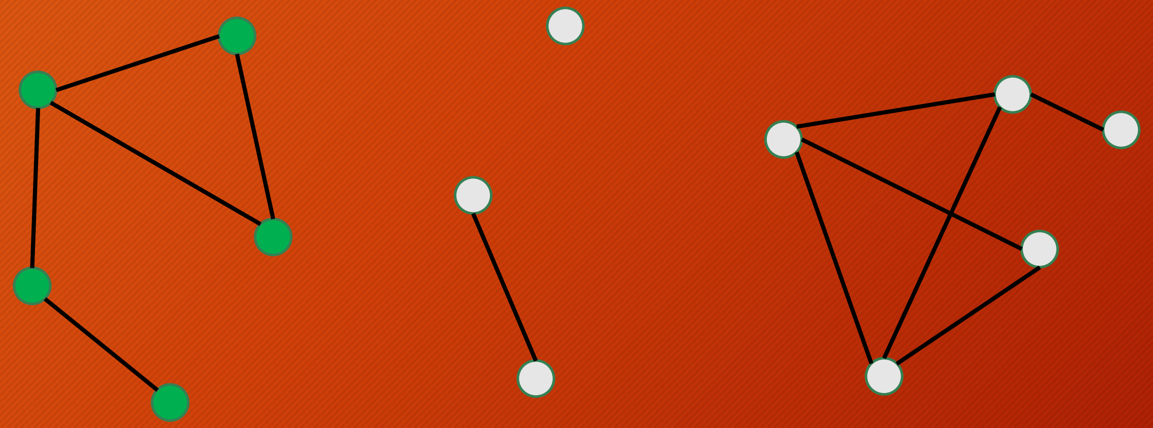
Serial traversal

- Algorithm starts at one vertex



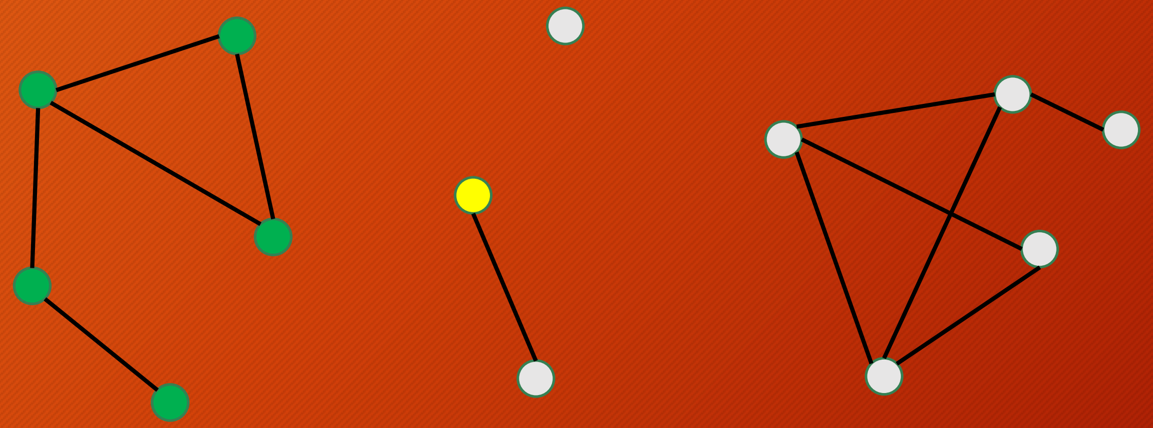
Serial traversal

- And transverses all connected vertices



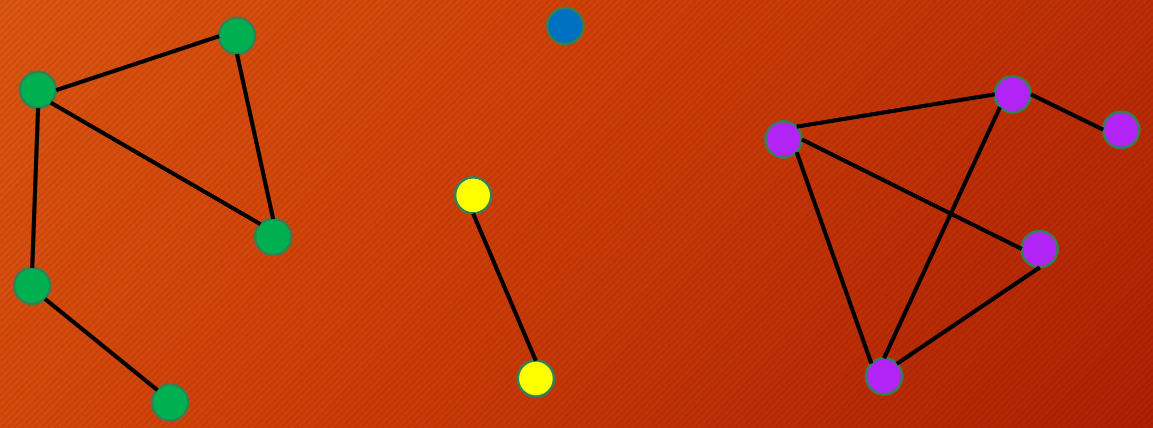
Serial traversal

- Continues to next unmarked vertex

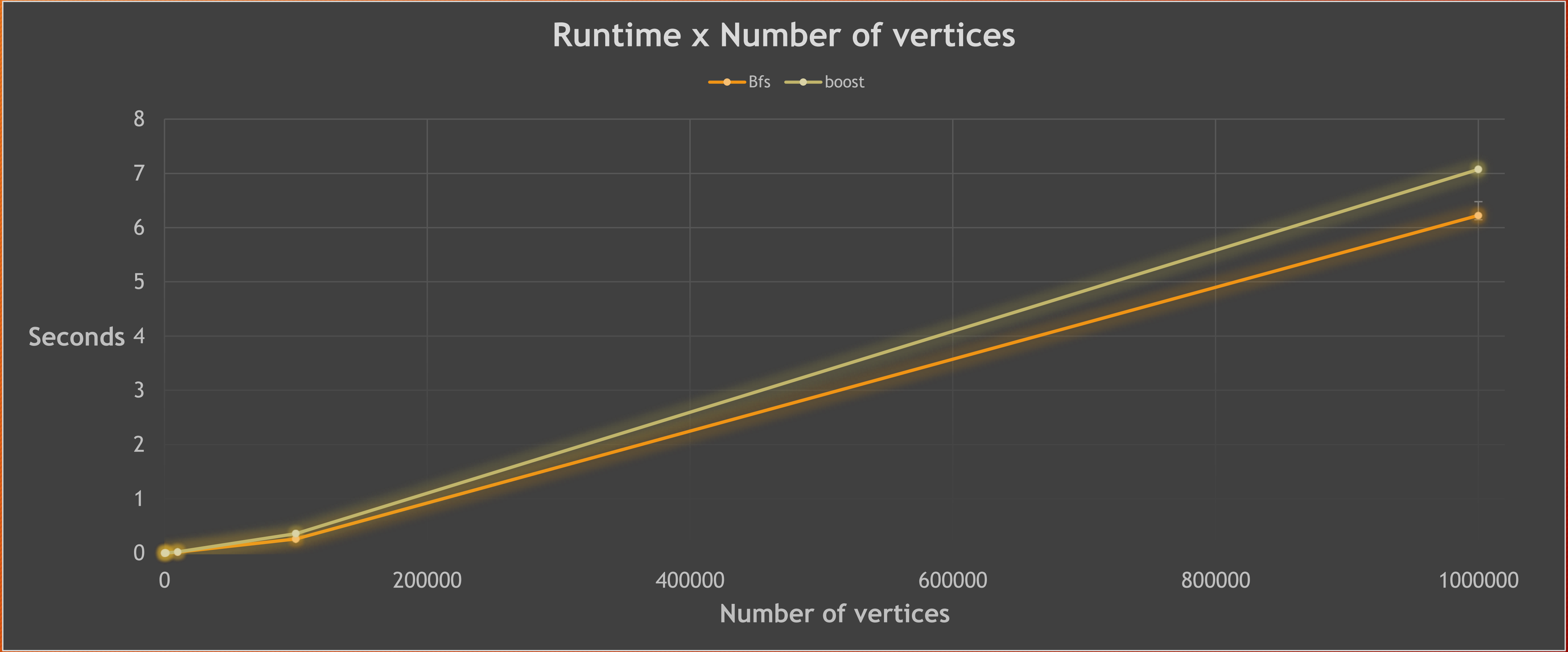


Serial traversal

- And marks each component



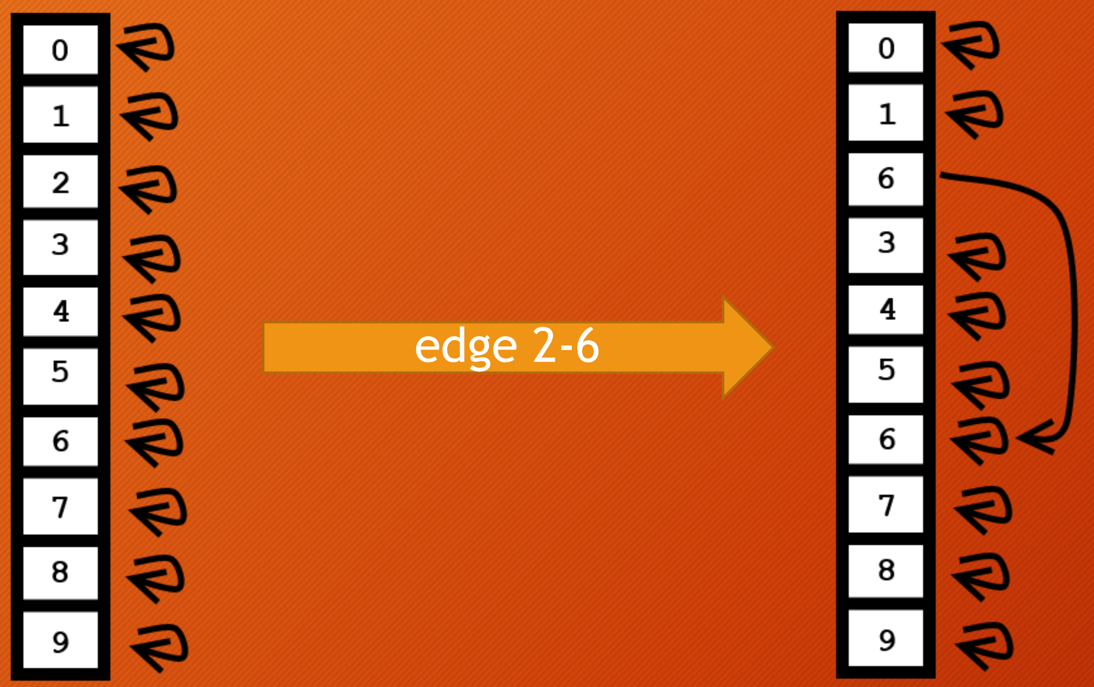
Serial traversal



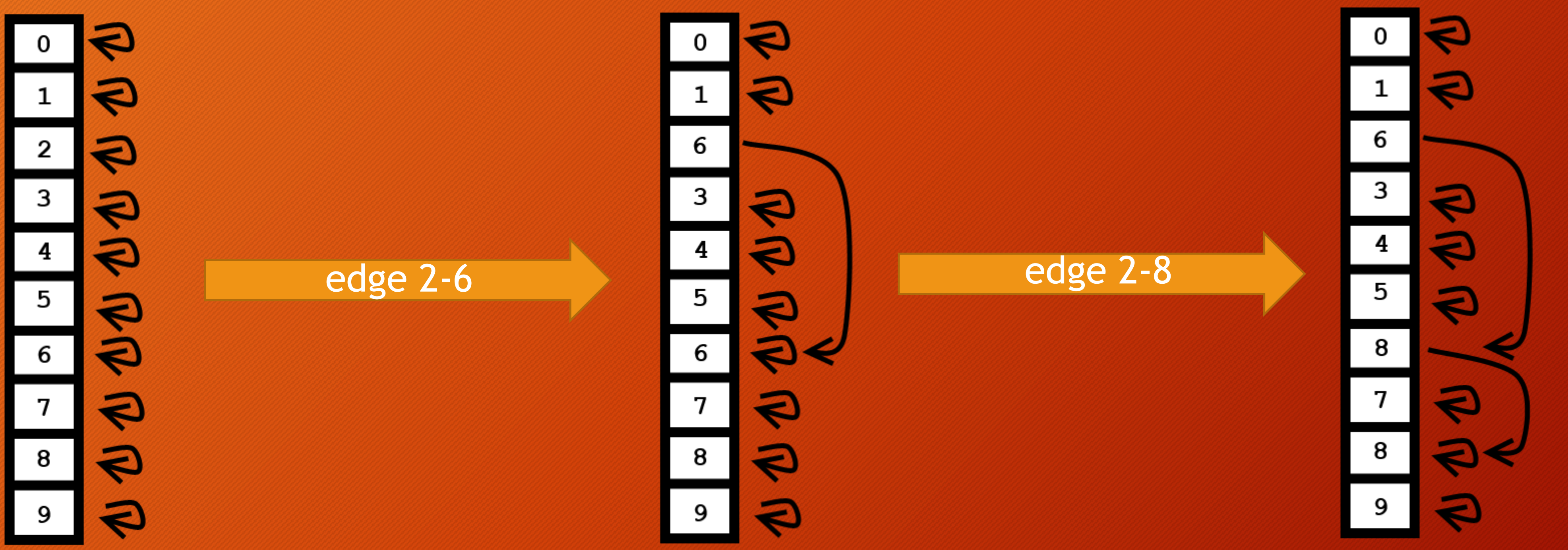
Union find

Union Find

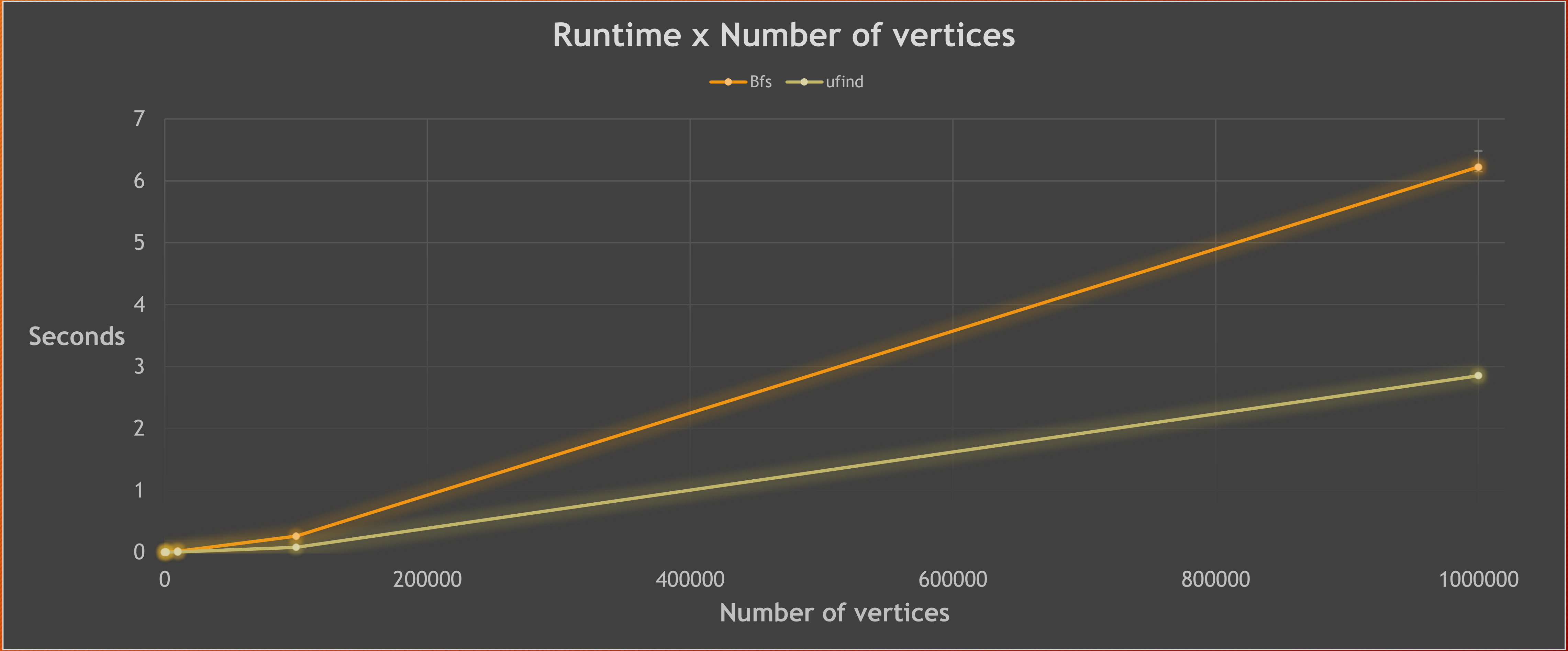
Union Find



Union Find

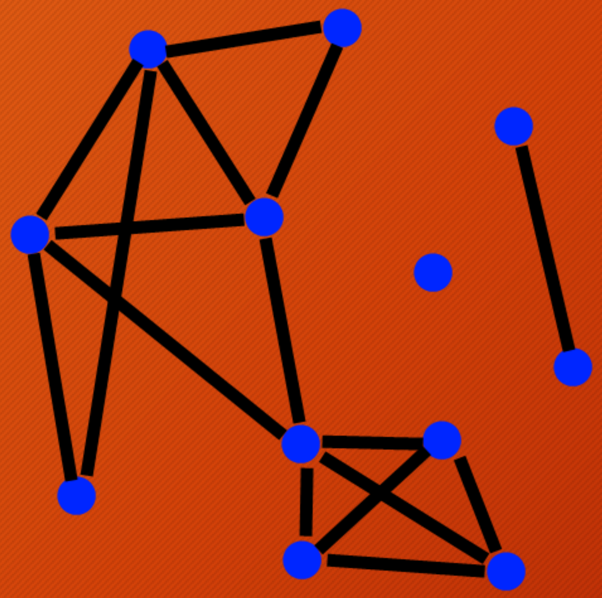


Union find

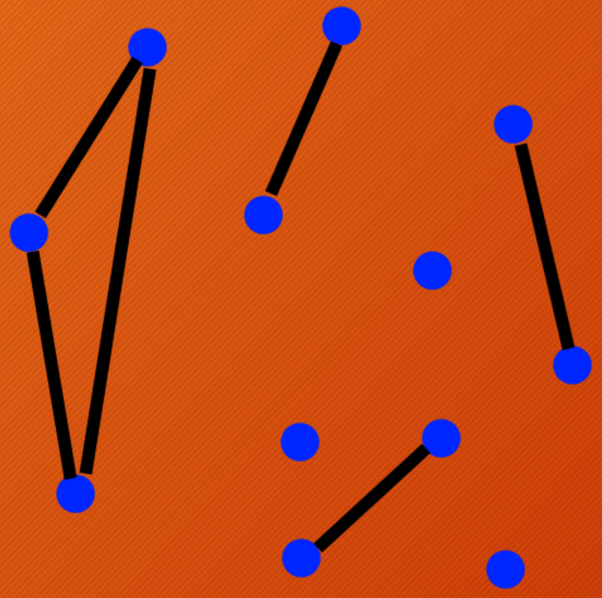

Runtime x Number of vertices
Bfs
ufind
Seconds
7
6
5
4
3
2
1
0
0
200000
400000
600000
800000
1000000
Number of vertices

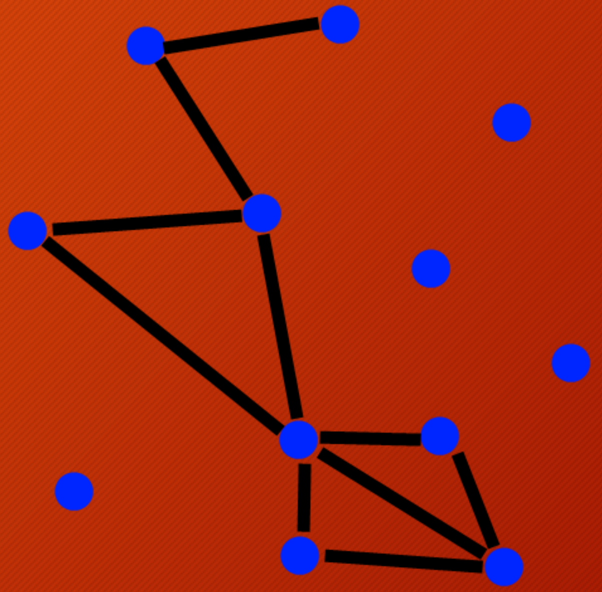
Parallel spanning tree

Parallel spanning tree

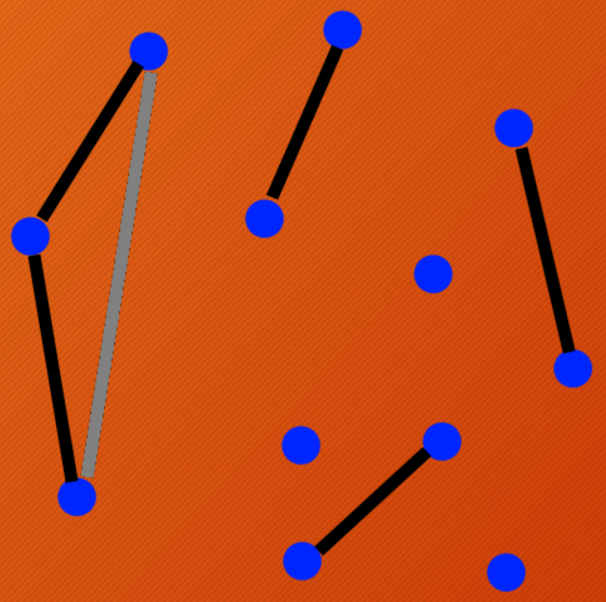


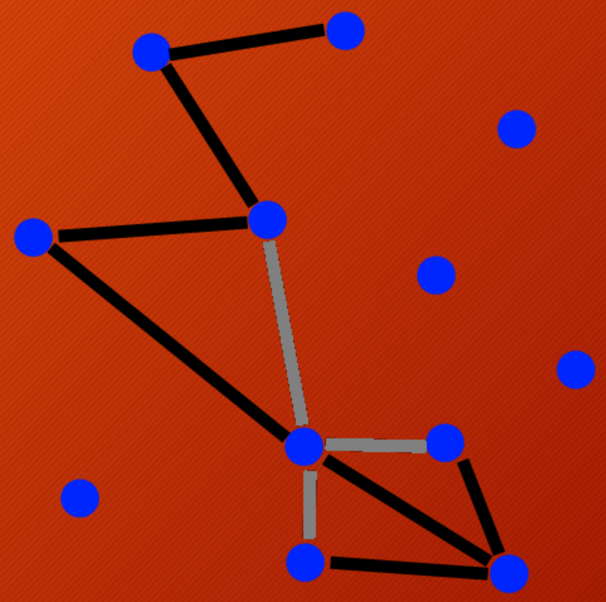
Parallel spanning tree



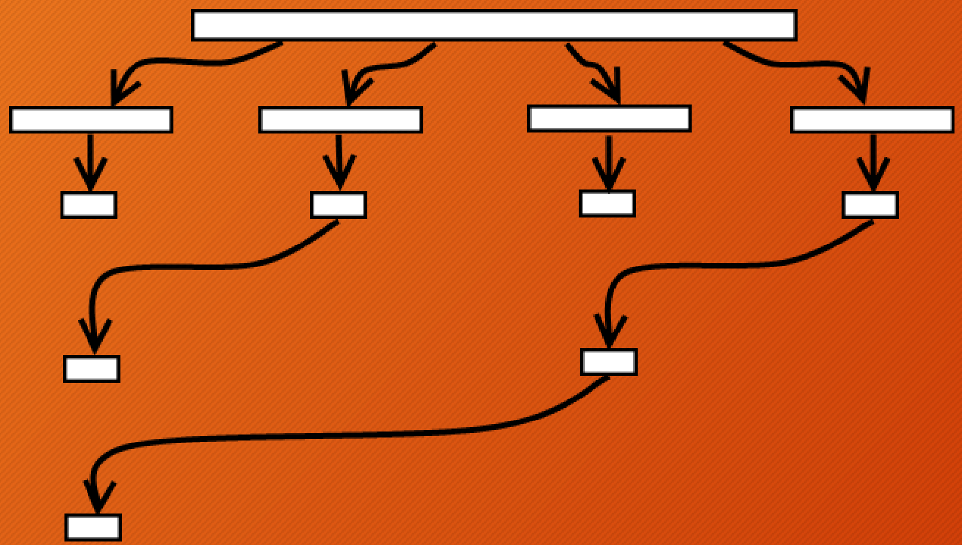


Parallel spanning tree





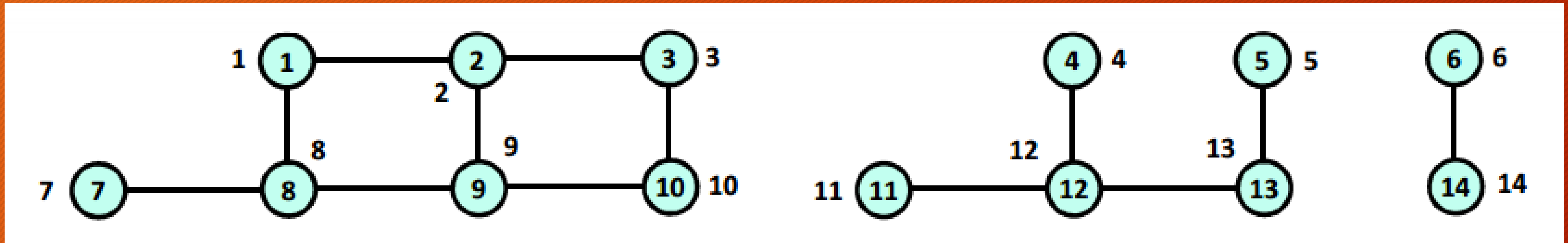
Parallel spanning tree



- Runtime: O(m/p+ n*log(p))
- Strong scaling in number of processors

Randomized contraction

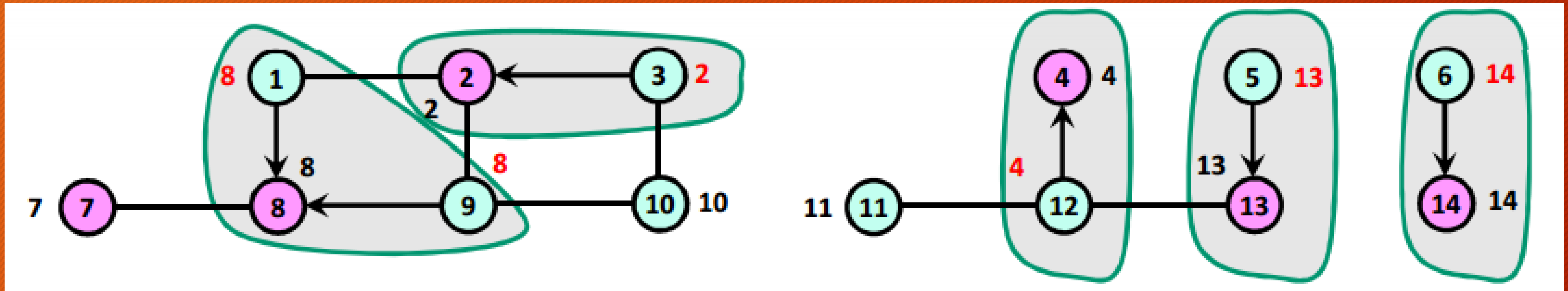
Randomized contraction



source: http://www3.cs.stonybrook.edu/~rezaul/Spring-2012/CSE613/CSE613-lecture-11.pdf

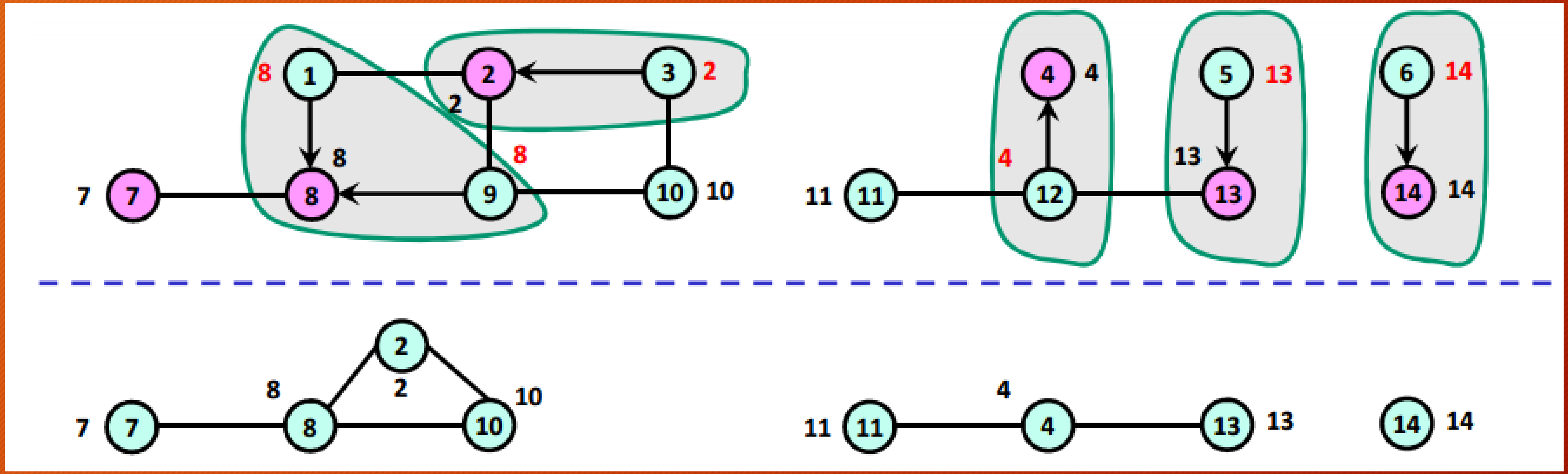
Randomized contraction

- Each vertex is given a random color (pink / light blue)



Randomized contraction

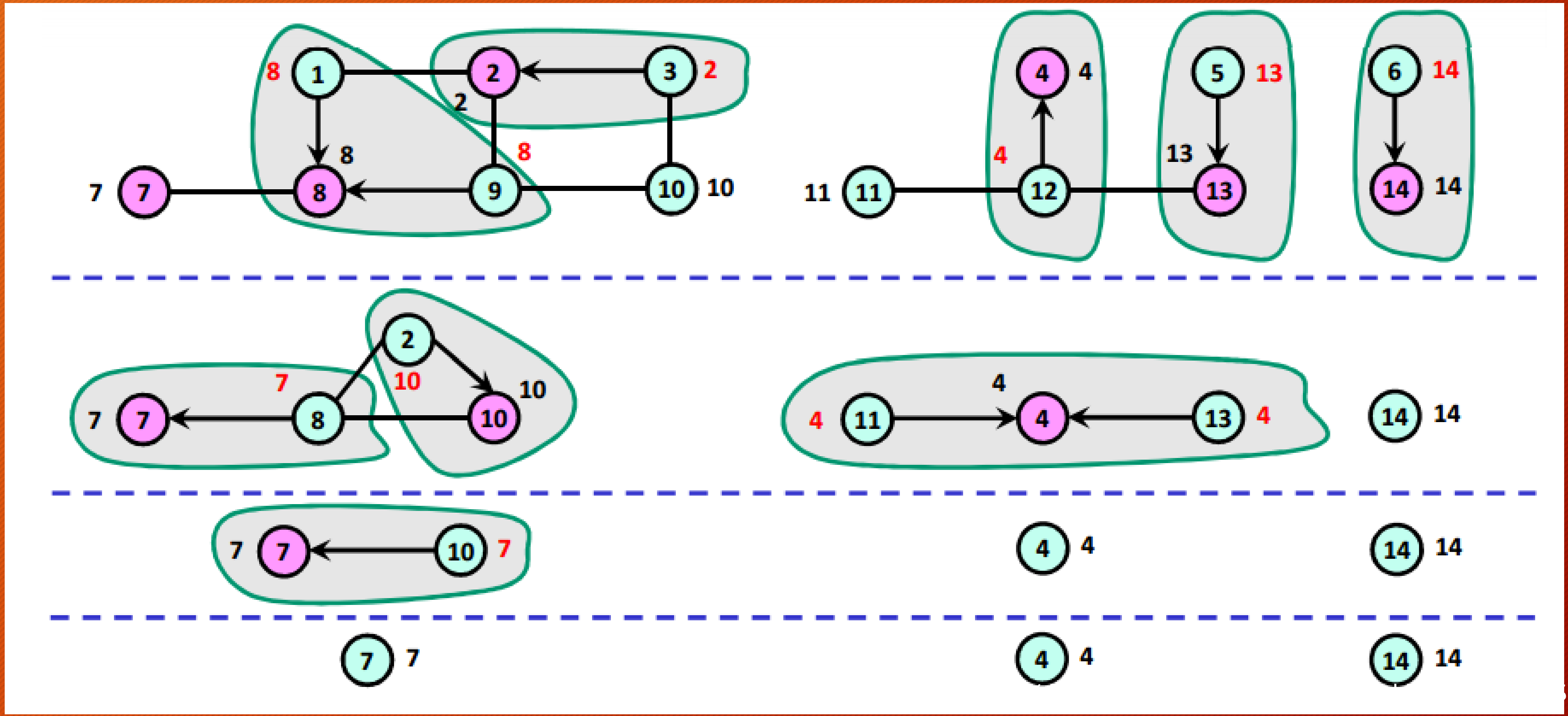
- Each contraction leads to less edges



source: http://www3.cs.stonybrook.edu/~rezaul/Spring-2012/CSE613/CSE613-lecture-11.pdf

Randomized contraction

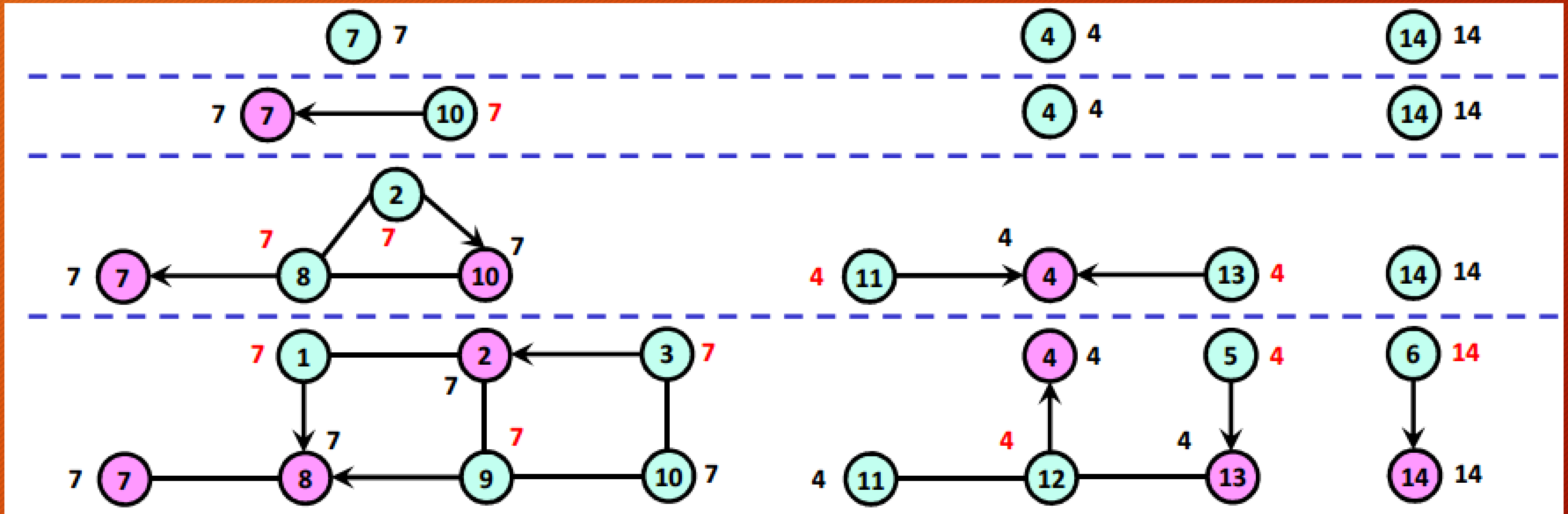
- Contract until no edges are left



E613/CSE613-lecture-11.pdf

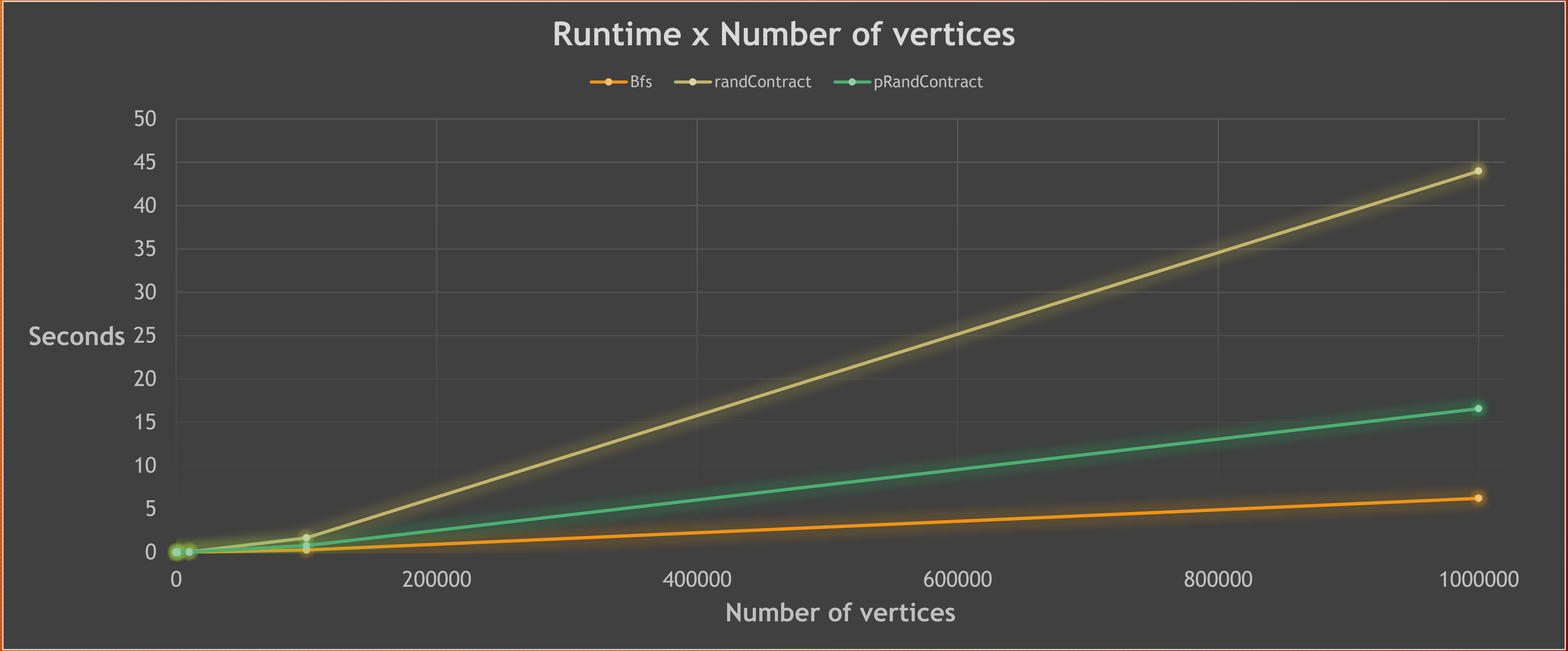
Randomized contraction

- Reverse the path to find the vertices component number



source: http://www3.cs.stonybrook.edu/~rezaul/Spring-2012/CSE613/CSE613-lecture-11.pdf

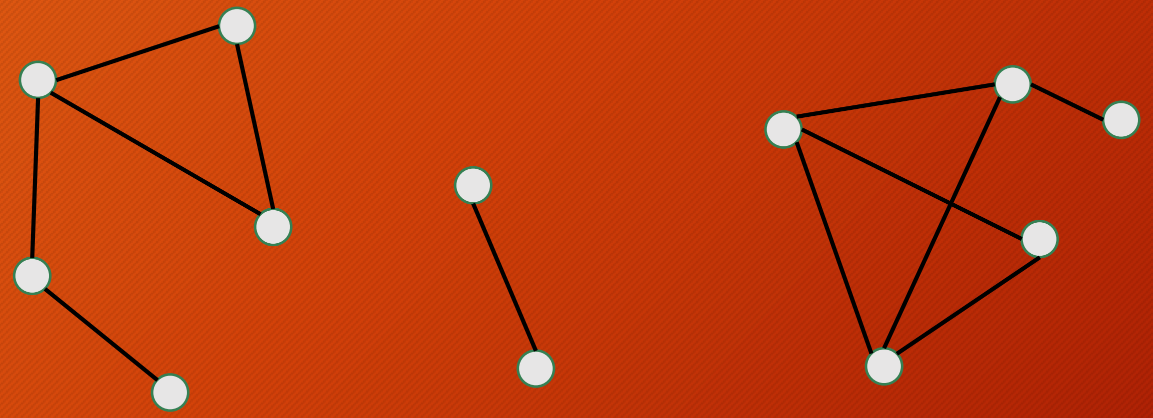
Randomized contraction



Parallel traversal

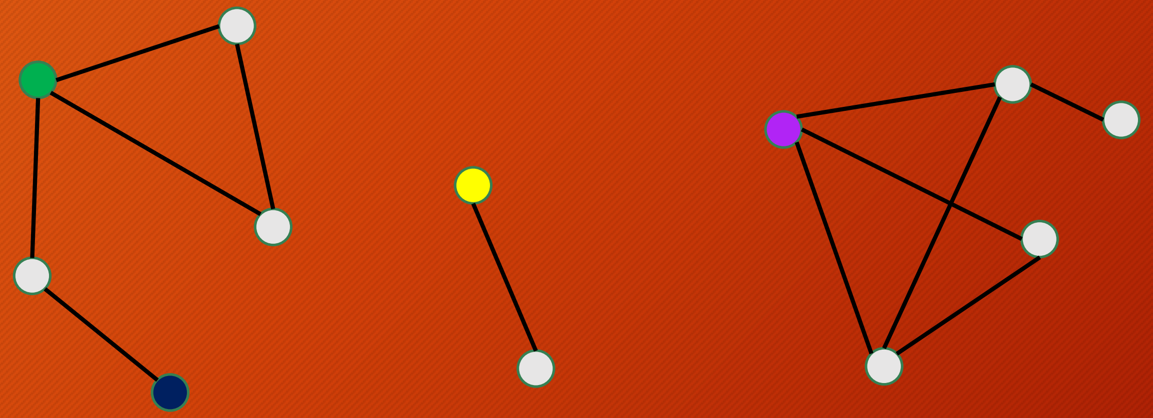
Parallel traversal

- Unmarked vertices



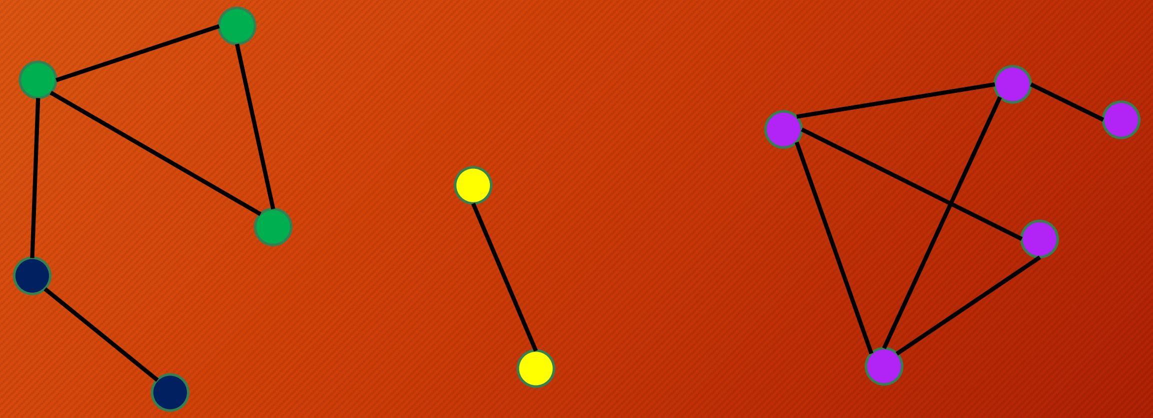
Parallel traversal

- Each thread starts at a vertex



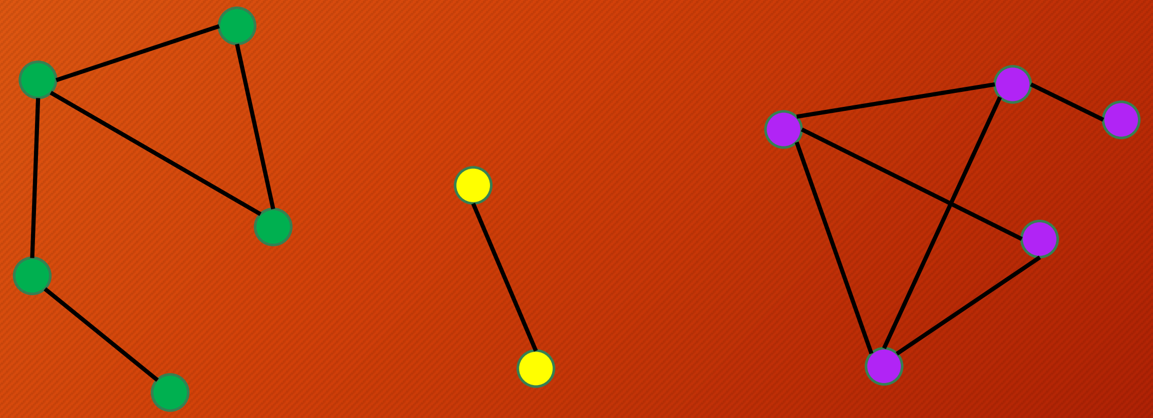
Parallel traversal

- Vertices are marked
- When one thread hits a component marked by another thread, it makes an entry in the merge table



Parallel traversal

- Vertices components are merged



Parallel traversal

- Problem, thread might not notice it is marking a vertex that was already marked by another thread

Parallel traversal

- Problem, thread might not notice it is marking a vertex that was already marked by another thread

Green thread

Step 1

Yellow thread

Step 1

Parallel traversal

- Problem, thread might not notice it is marking a vertex that was already marked by another thread

Parallel traversal

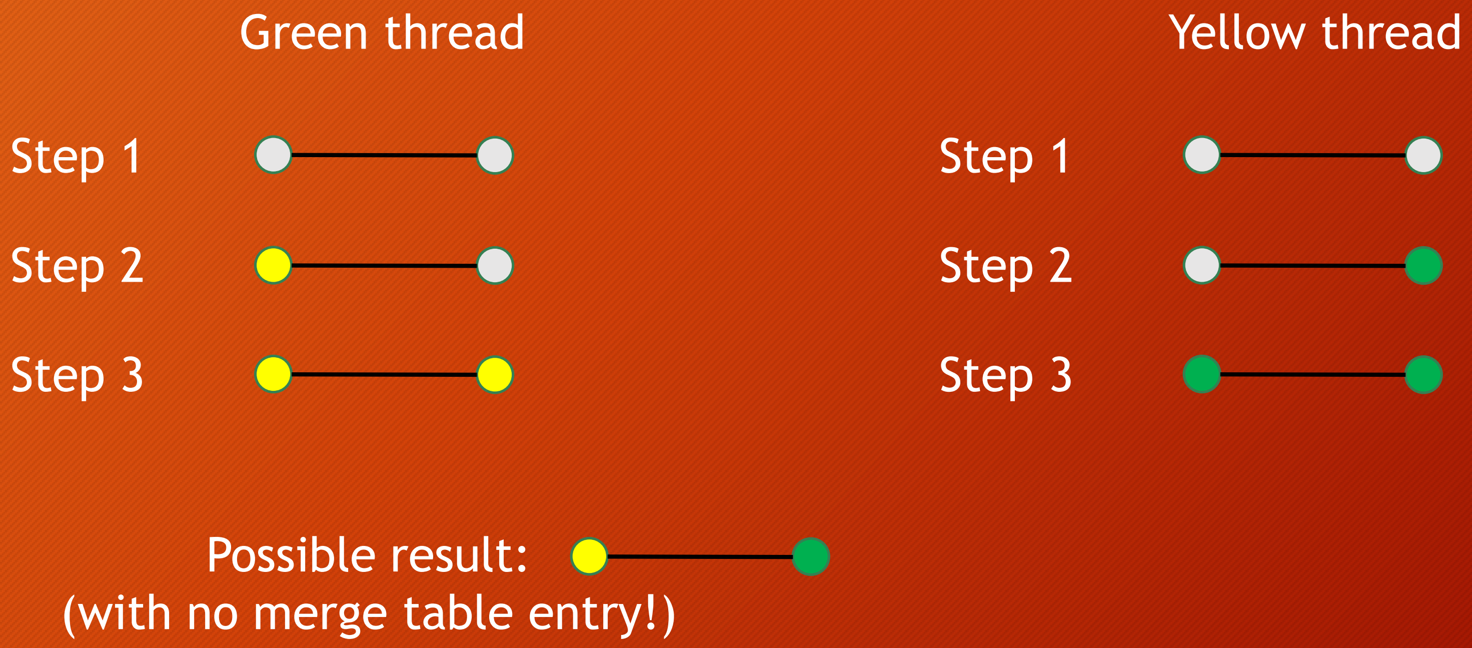
- Problem, thread might not notice it is marking a vertex that was already marked by another thread

Parallel traversal

- Problem, thread might not notice it is marking a vertex that was already marked by another thread

Parallel traversal

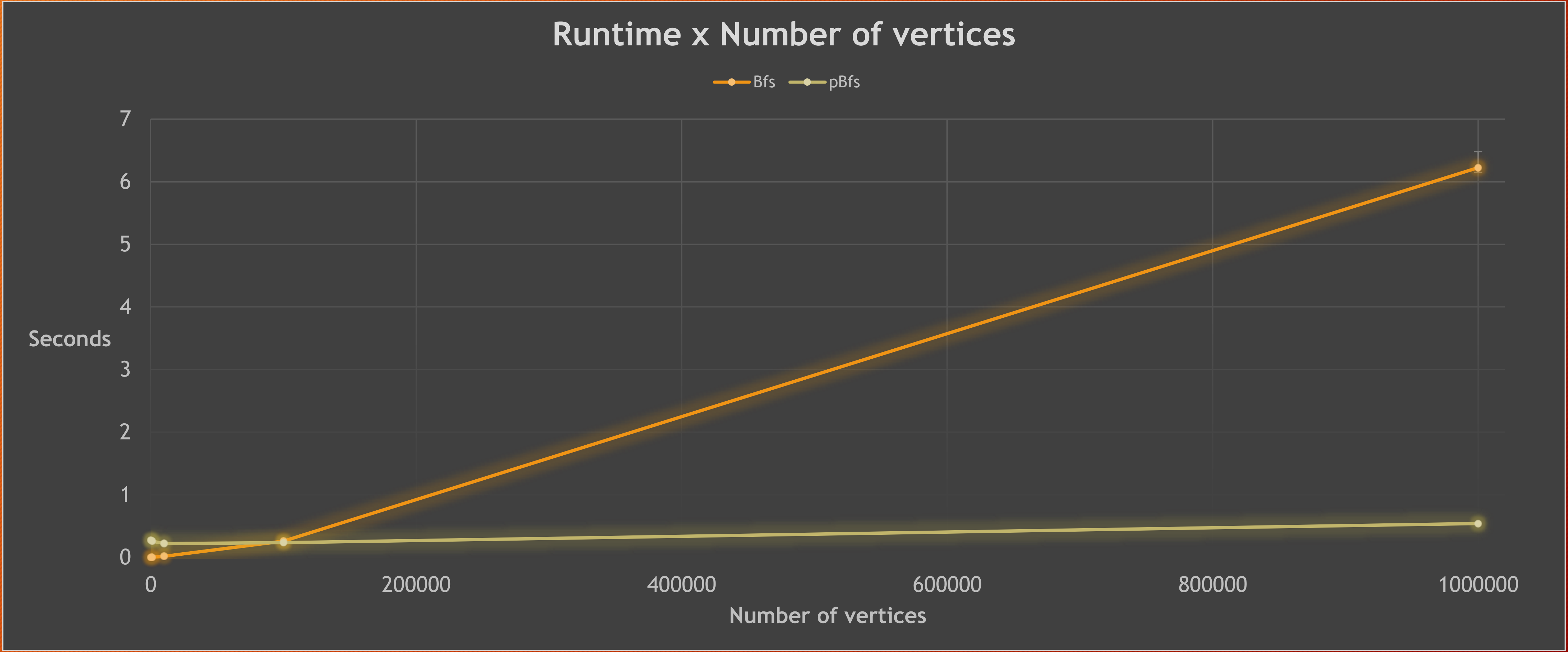
- Problem, thread might not notice it is marking a vertex that was already marked by another thread



Parallel traversal

- Solution 1: check if algorithm output is correct. If not, run again with 1 thread
- Fast to check
- Problem is very unlikely. Slow execution is amortized

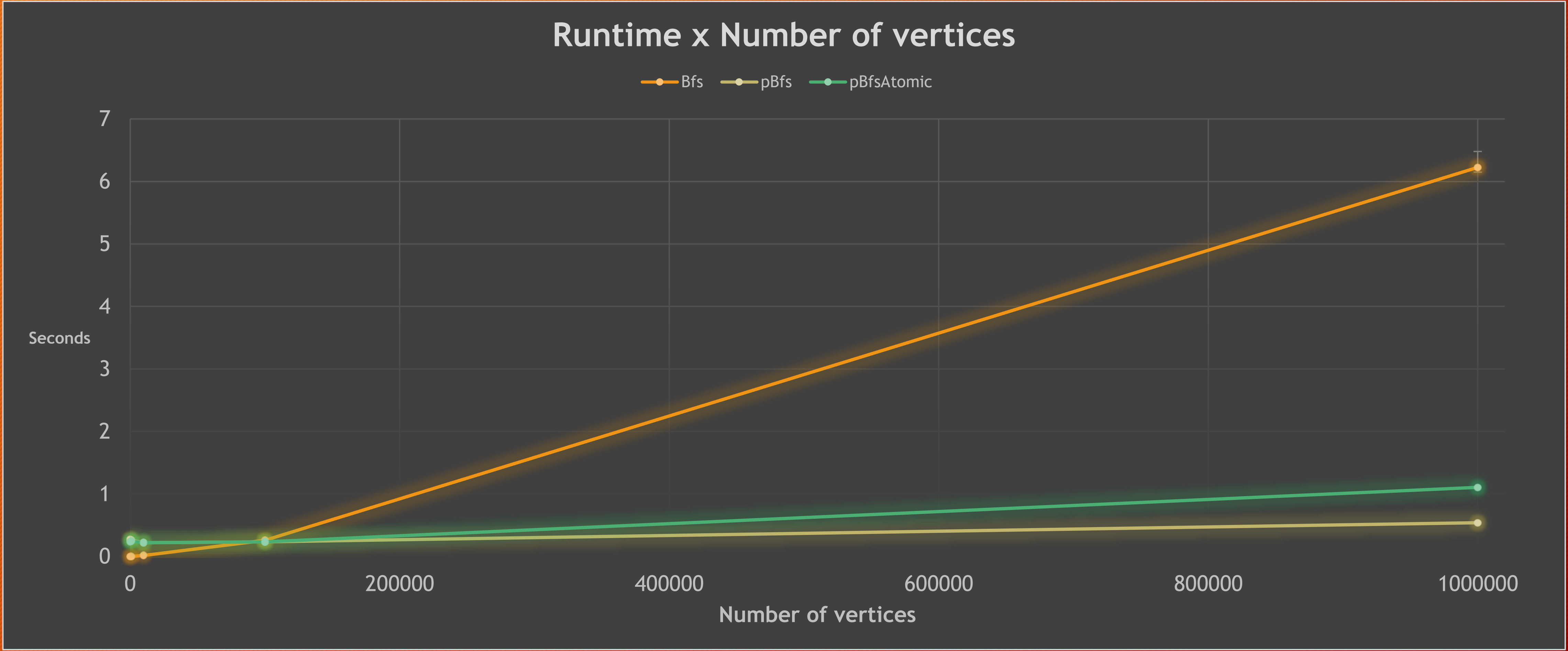
Parallel traversal



Parallel traversal with atomics

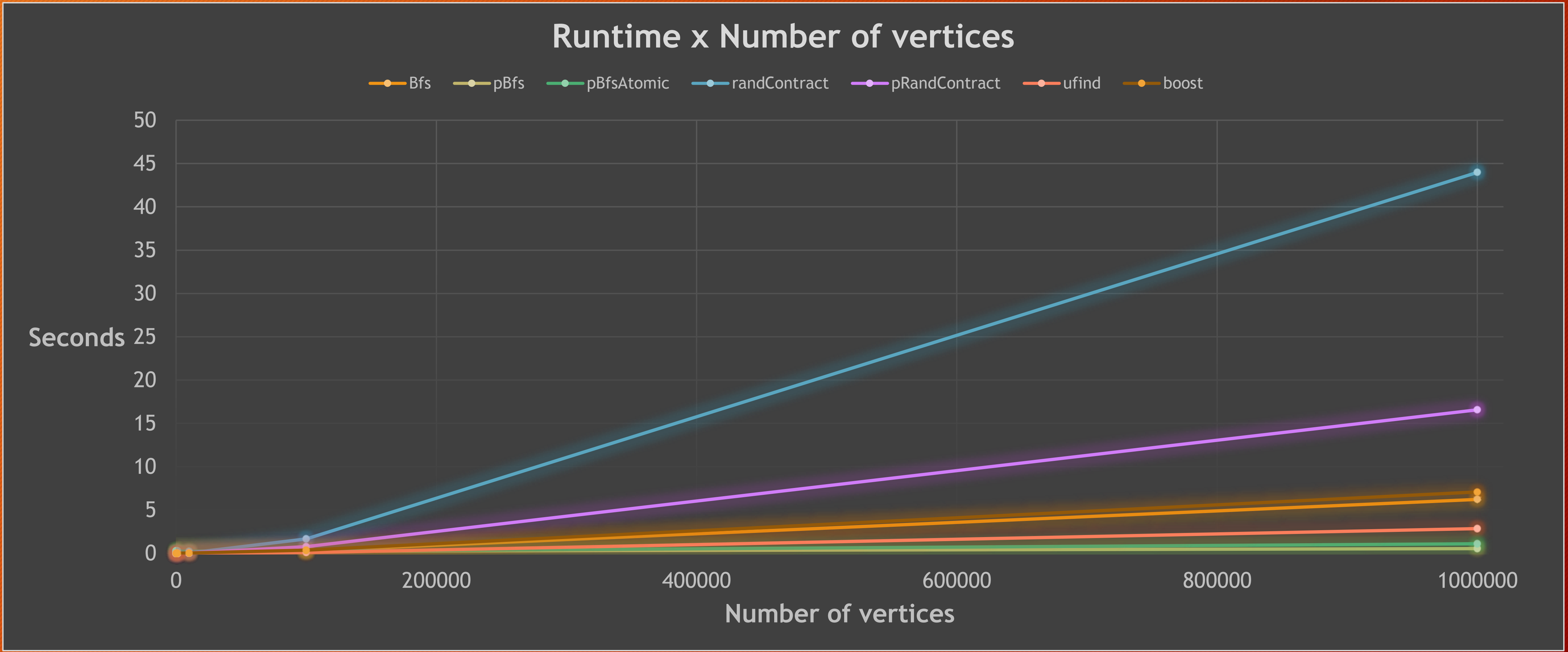
- Solution 2: Use test and set (atomic_flag)
- Every time a thread marks a vertex, it makes sure no other thread marked it before with test and set
- Each vertex has is own atomic_flag

Parallel traversal with atomics

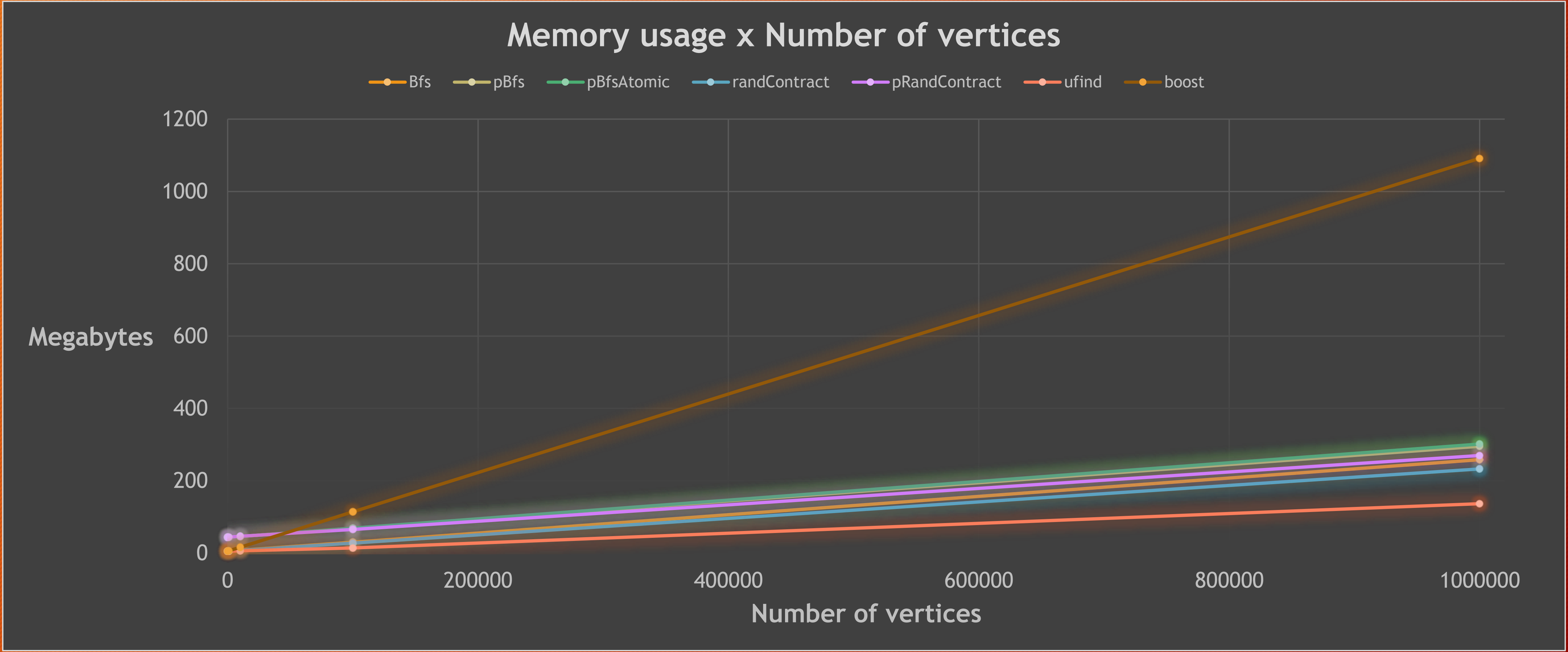


Algorithm results

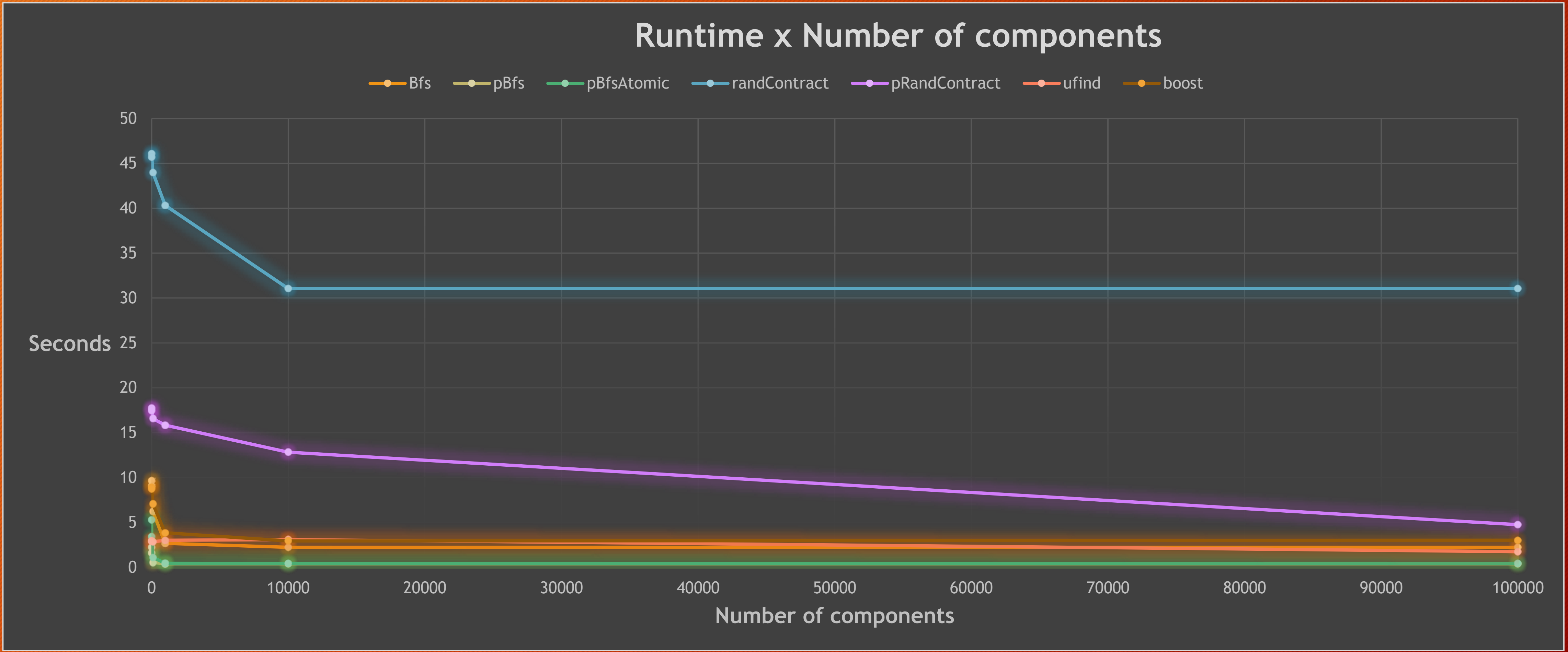
Runtime varying the number of vertices



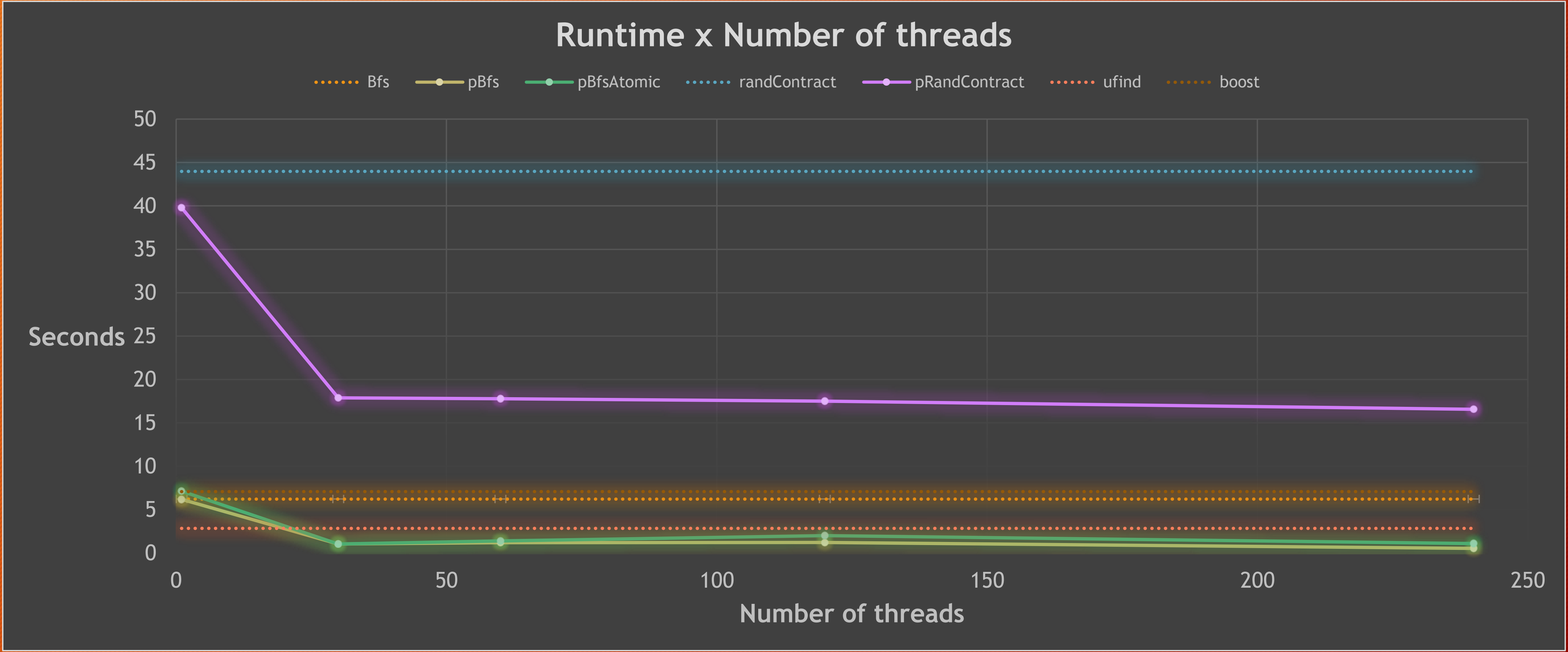
Memory usage varying the number of vertices



Runtime varying the number of components

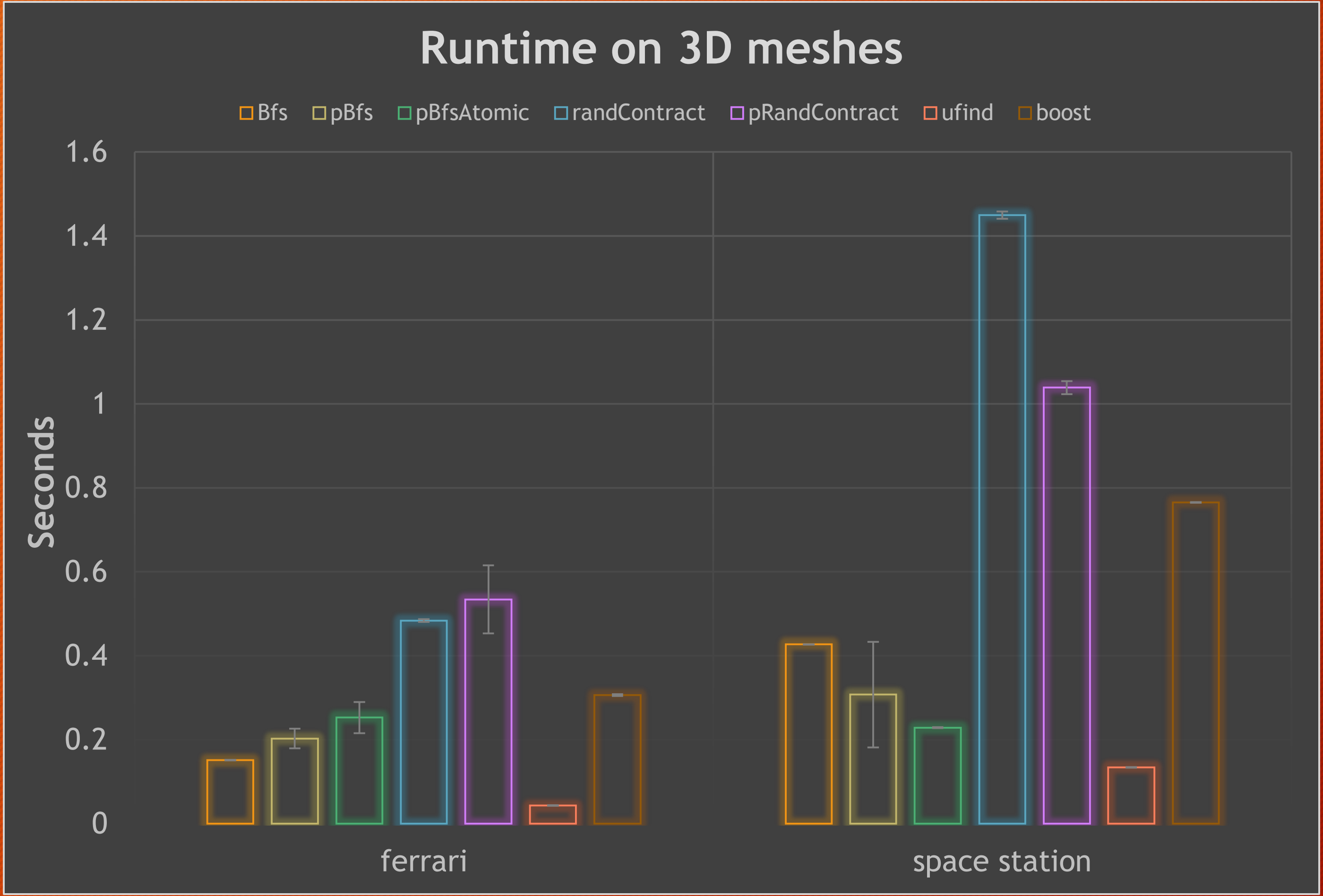


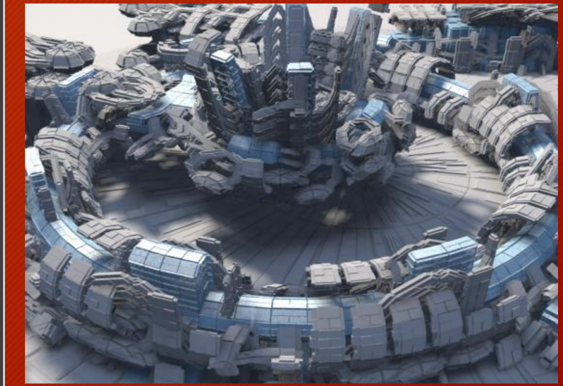
Runtime varying the number of threads



Runtime on real world graphs

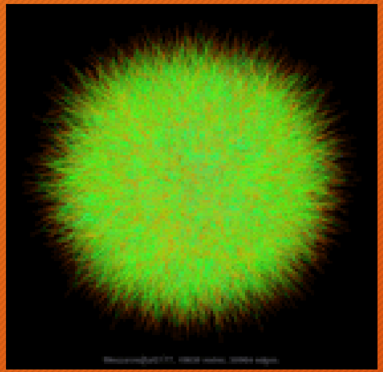
~350 thousand vertices
~700 thousand edges
753 components



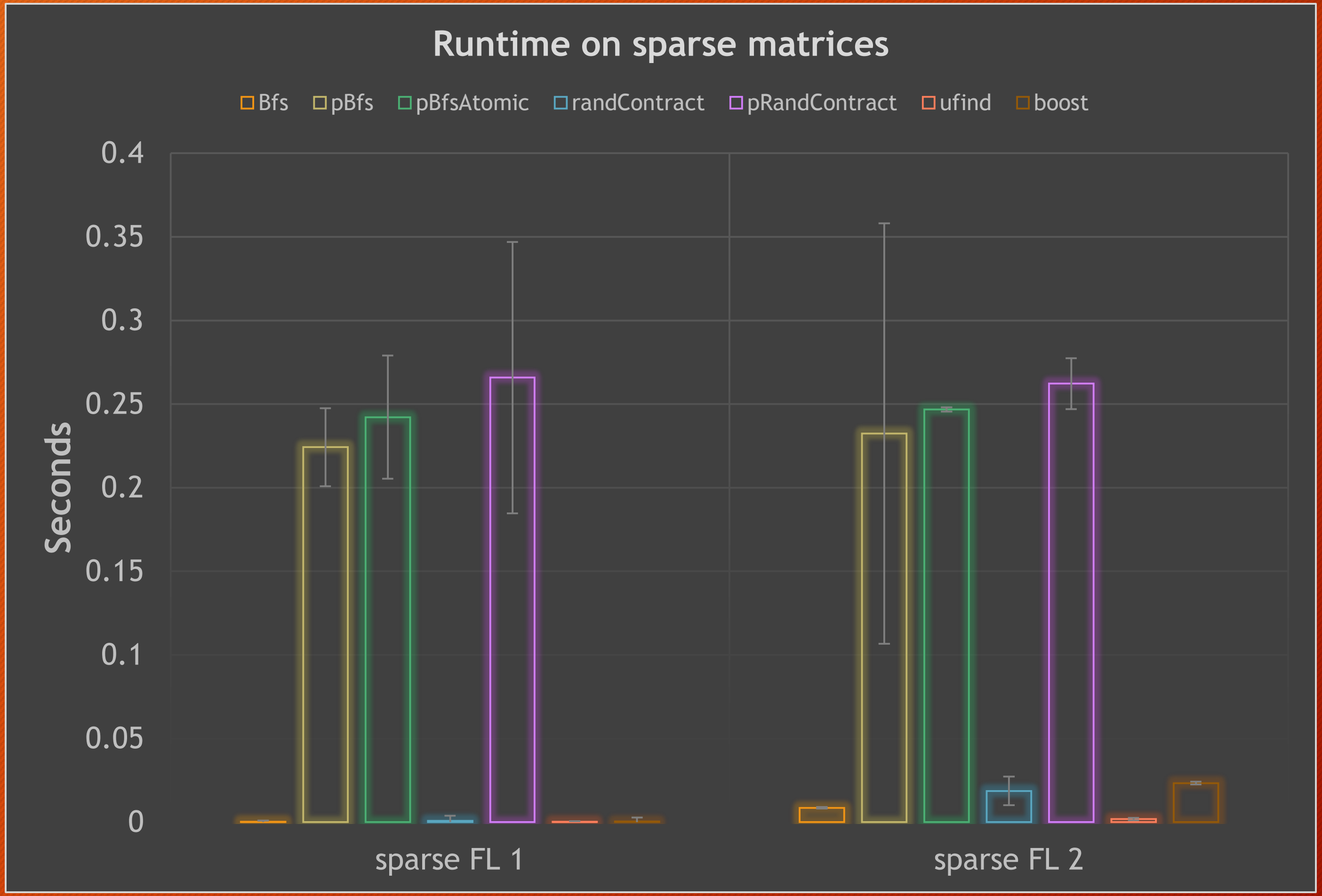


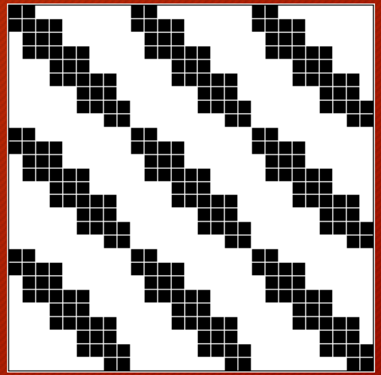
~1.2 million vertices
~2.4 million edges
6617 components

Runtime on real world graphs



608 vertices
1216 edges
1 component





~10 thousand vertices
~31 thousand edges
135 components

Conclusions and lessons learned

- Different algorithms are right for different situations
  - Best overall: union find (even though traversal has better complexity)
  - Best for very large graphs and plenty of cores: parallel traversal with atomics
- Should use algorithm that matches graph datastructure (not worth the time switching)
- Extra threads can help runtime up to a limit
- Boost is very bloated in terms of memory and not difficult to beat in runtime
- It is not a good idea to take measurements on a shared Xeon phi on the weekend before final presentation (5 hour queue)

Thank you