Tree-Based Density Clustering using Graphics Processors

Ben Welton and Evan Samanas
Paradyn Project

CScADS 2012
Snowbird, Utah
June 26-29, 2012

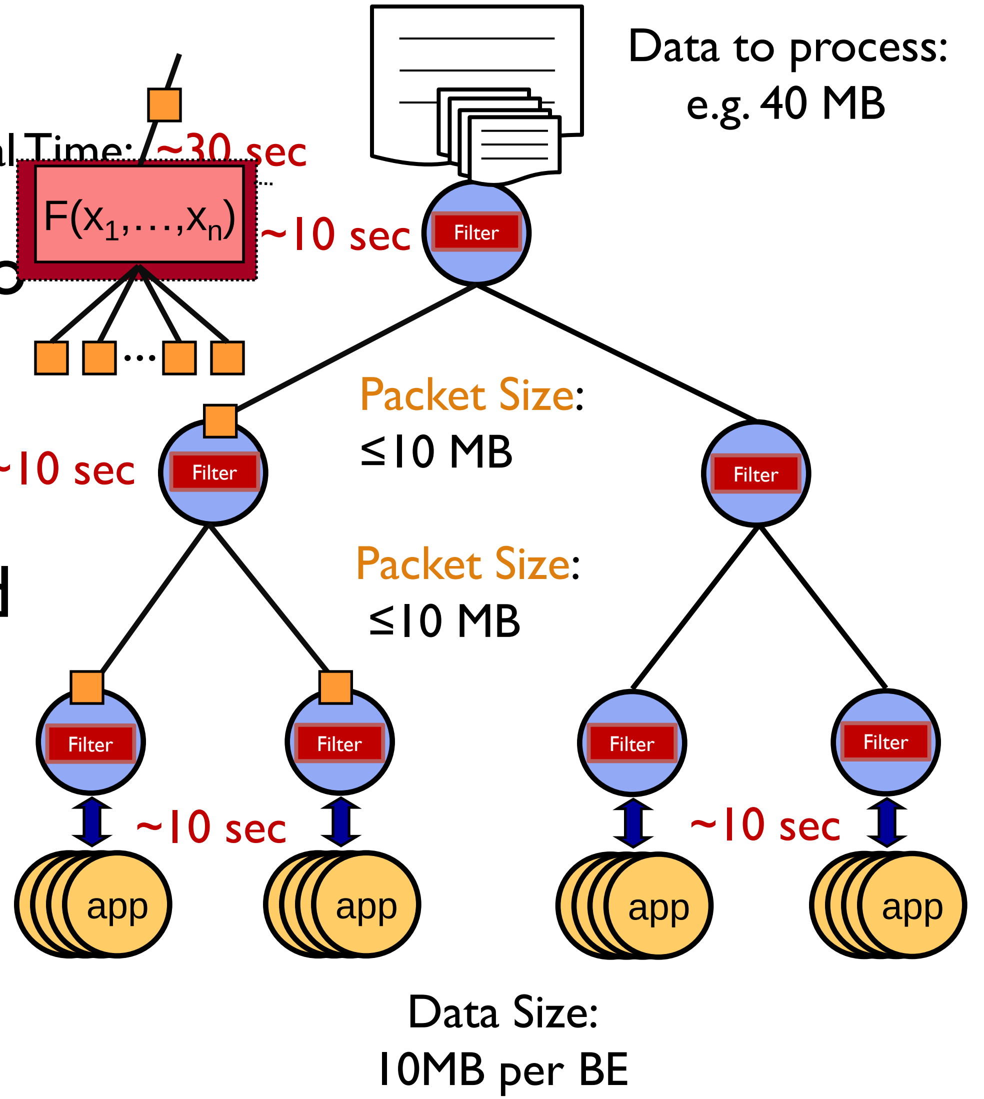
TBON Computation

TBON is a distributed computing model designed to be scalable, efficient, and flexible.

Flexible aggregation provided by user defined functions in filters

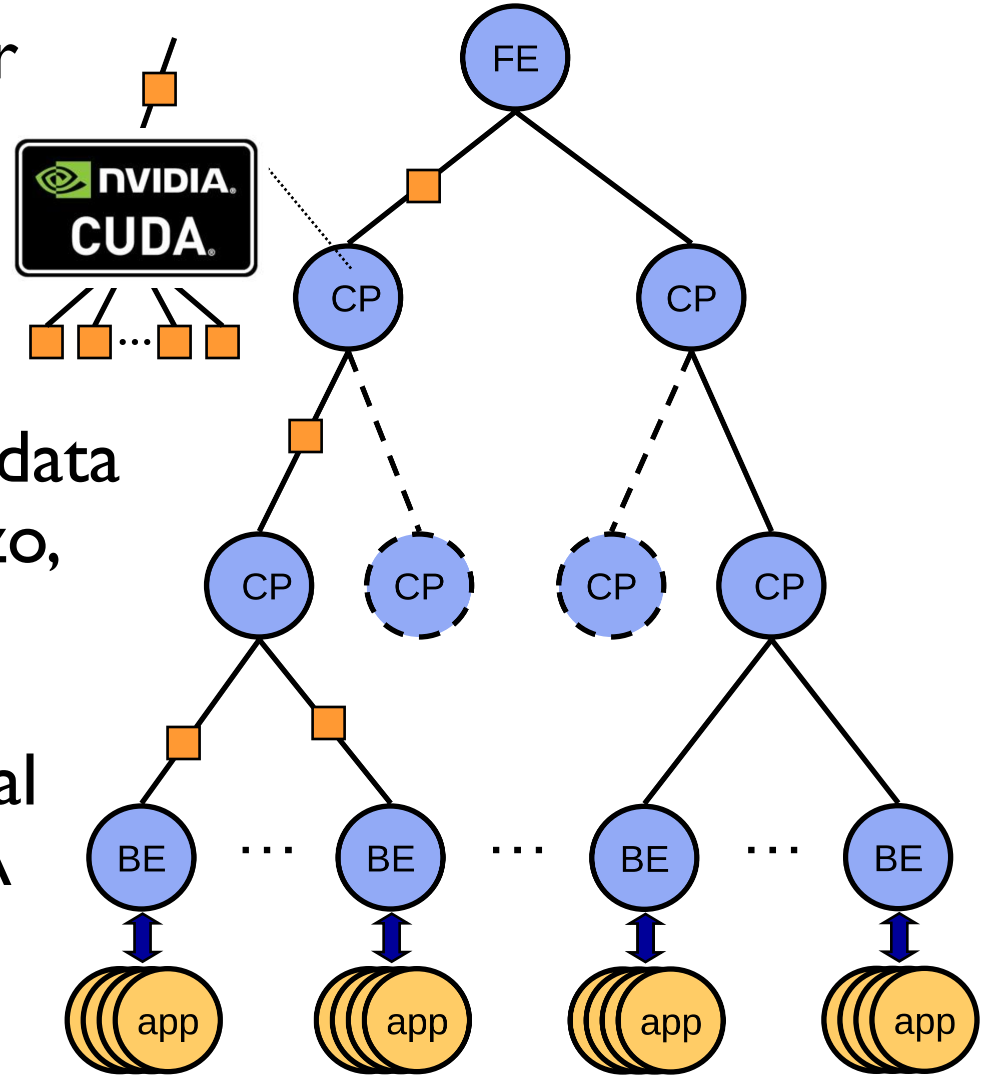
Ideal Characteristics:

- Filter output size constant or decreasing
- Computation rate similar across levels

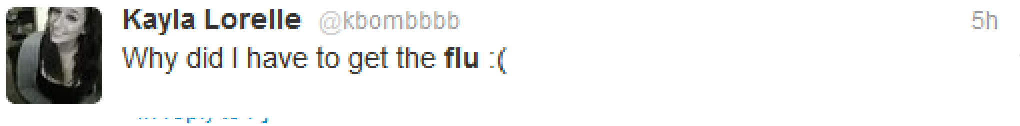


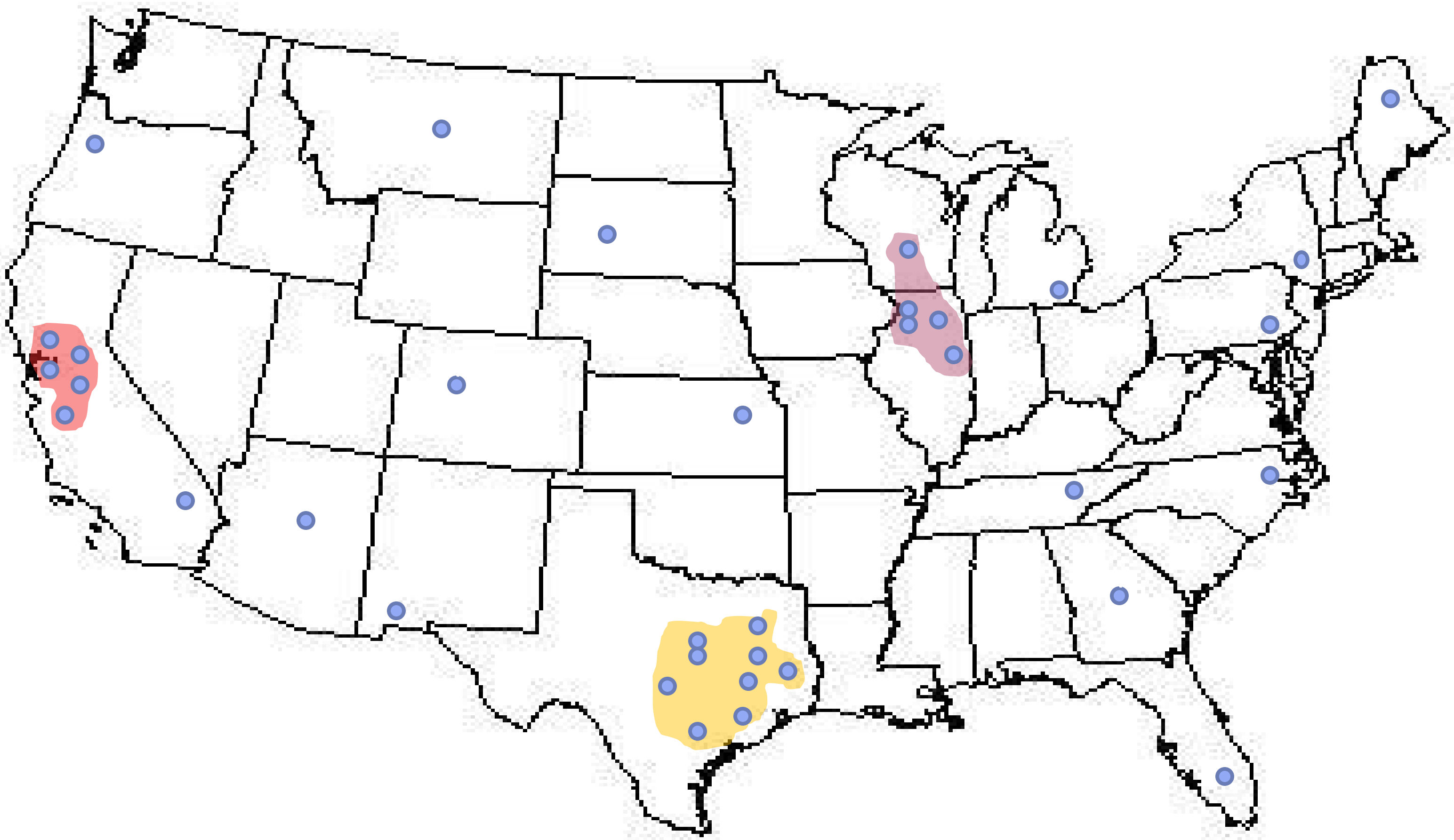
Why GPUs In A TBON?

- Increase compute power
- Trade computation for bandwidth
  - Derived summaries
  - Compute and send Δ data
  - Compressions (bzip, lzo, gzip, …)
- Filter function is a natural encapsulation for CUDA



The Tweet Stream

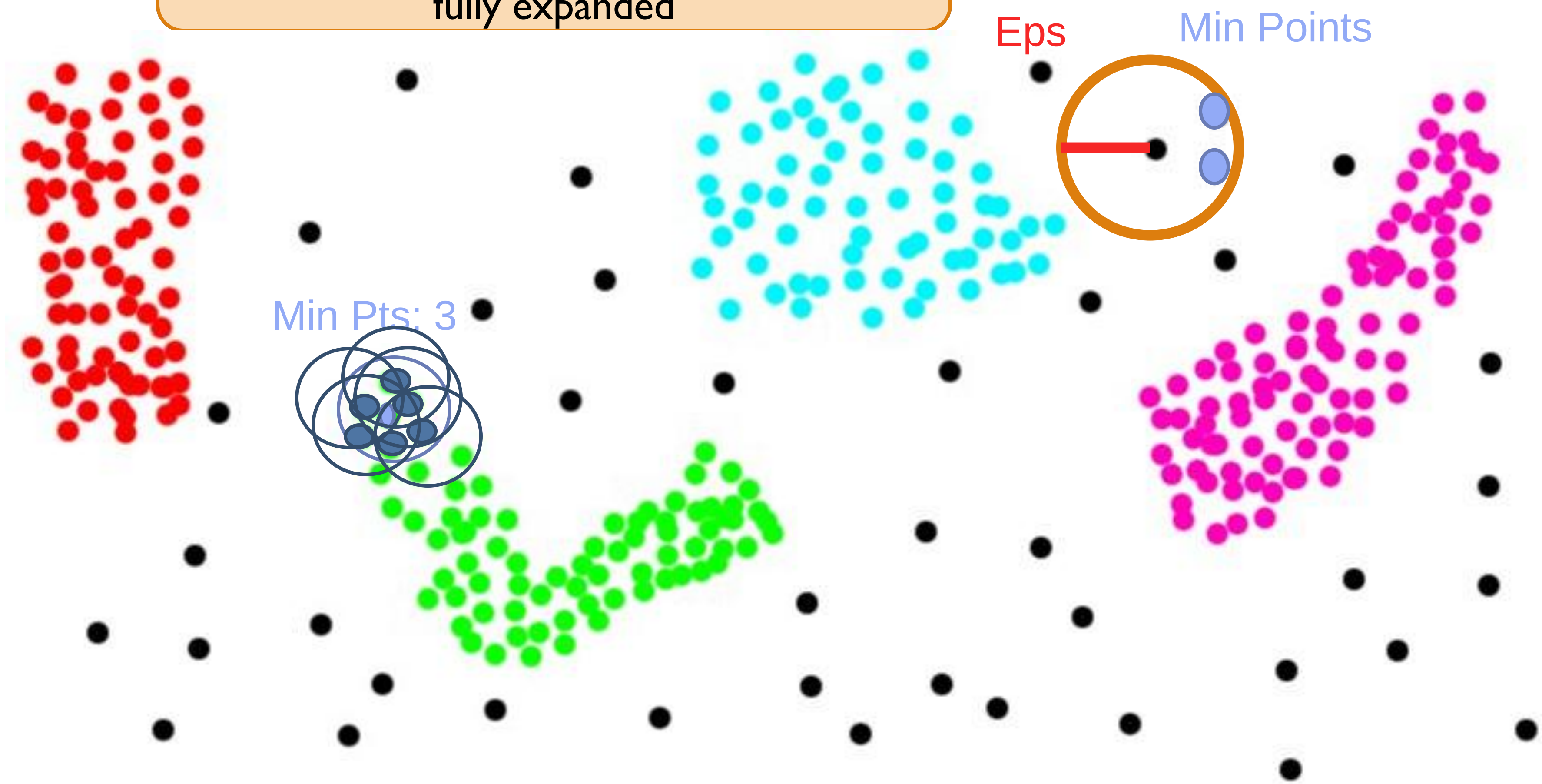




Source: Twitter, Map: About.com

Clustering Example (DBSCAN[1])

For every discovered point, this same calculation is performed until the cluster is fully expanded



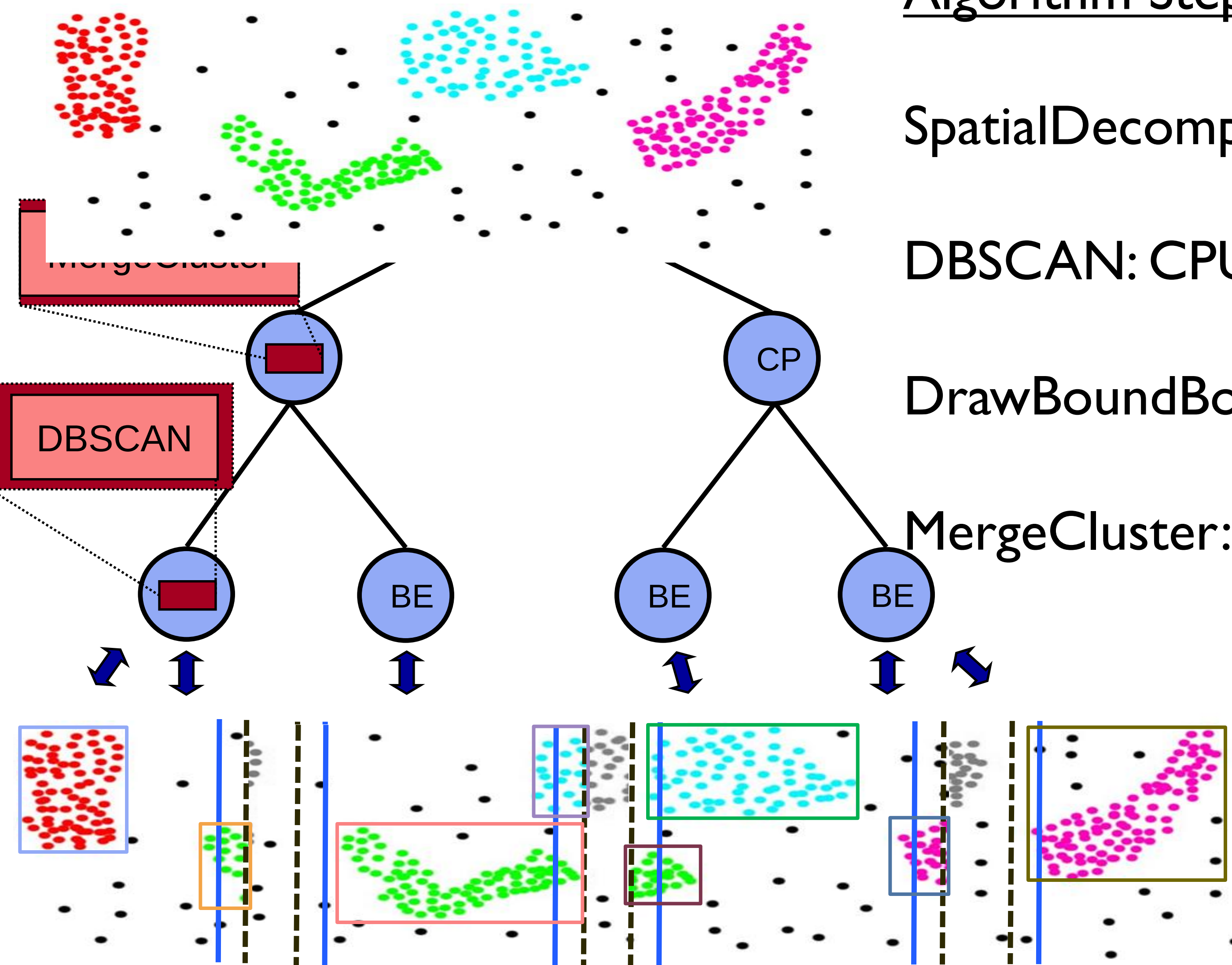
[1] M. Ester et. al., A density-based algorithm for discovering clusters in large spatial databases with noise, (1996)

Previous Work In Scaling DBSCAN

- PDBSCAN[2]
  - Quality equivalent to single DBSCAN
  - Linear speedup up to 8 nodes
- DBDC[3]
  - Sacrifices quality
  - ~30x speedup on 15 nodes
- CUDA-Dclust[4]
  - Quality equivalent to DBSCAN
  - ~15x faster on 1 node

[2] X. Xu et. al., A fast Parallel Clustering Algorithm for Large Spatial Databases (1999)
[3] E. Januzaj et. al., DBDC: Density Based Distributed Clustering (2004)
[4] C. Bohm et al., Density-based clustering using graphics processors (2009)

Tree-Based Clustering: Mr. Scan



<u>Algorithm Steps</u>

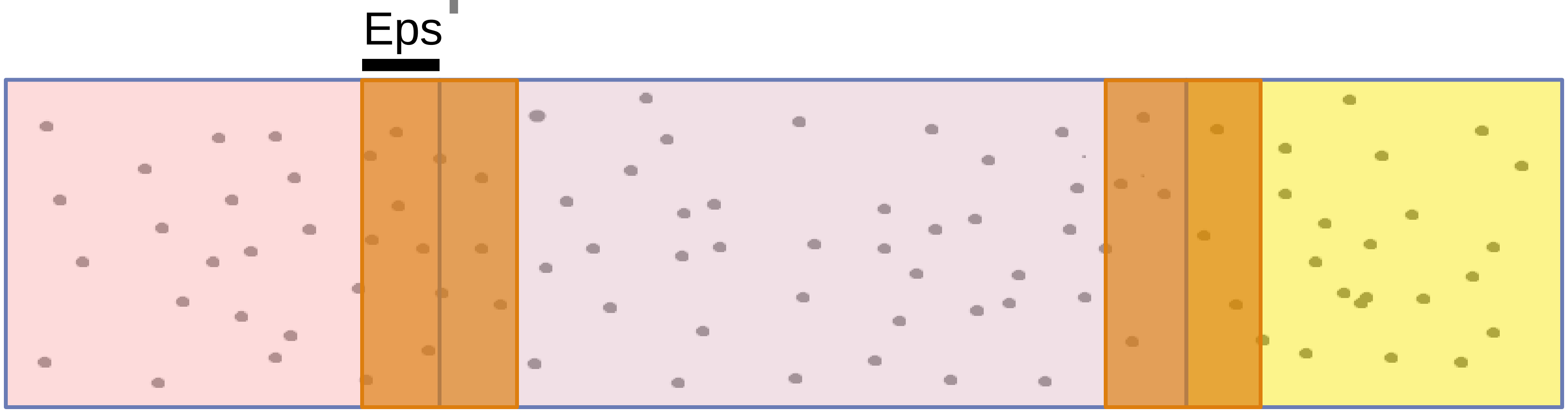
SpatialDecomp: CPU(@ FE)

DBSCAN: CPU or GPU(@ BE)

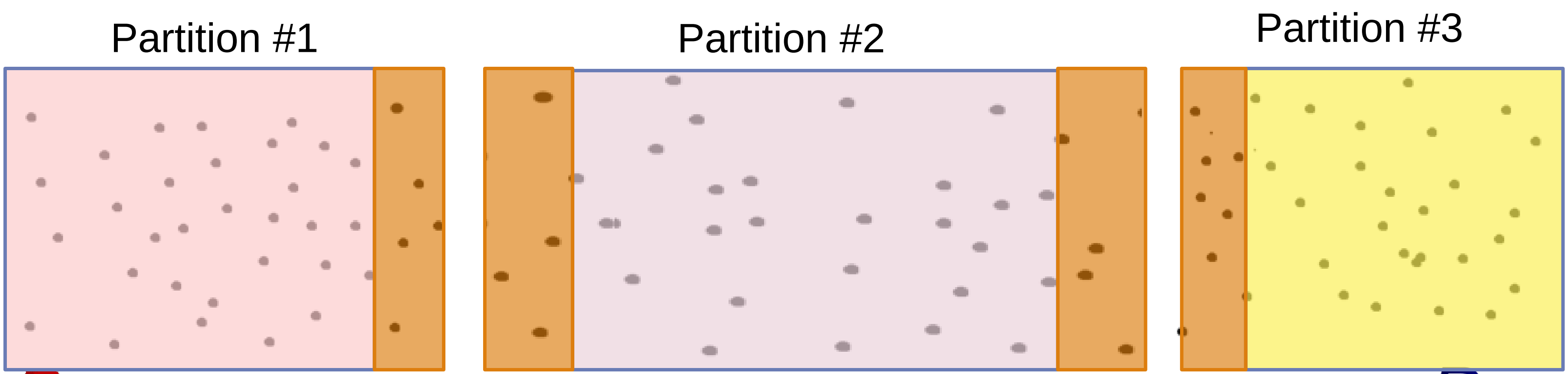
DrawBoundBox: CPU or GPU

MergeCluster: CPU (x #levels)

Spatial Decomposition

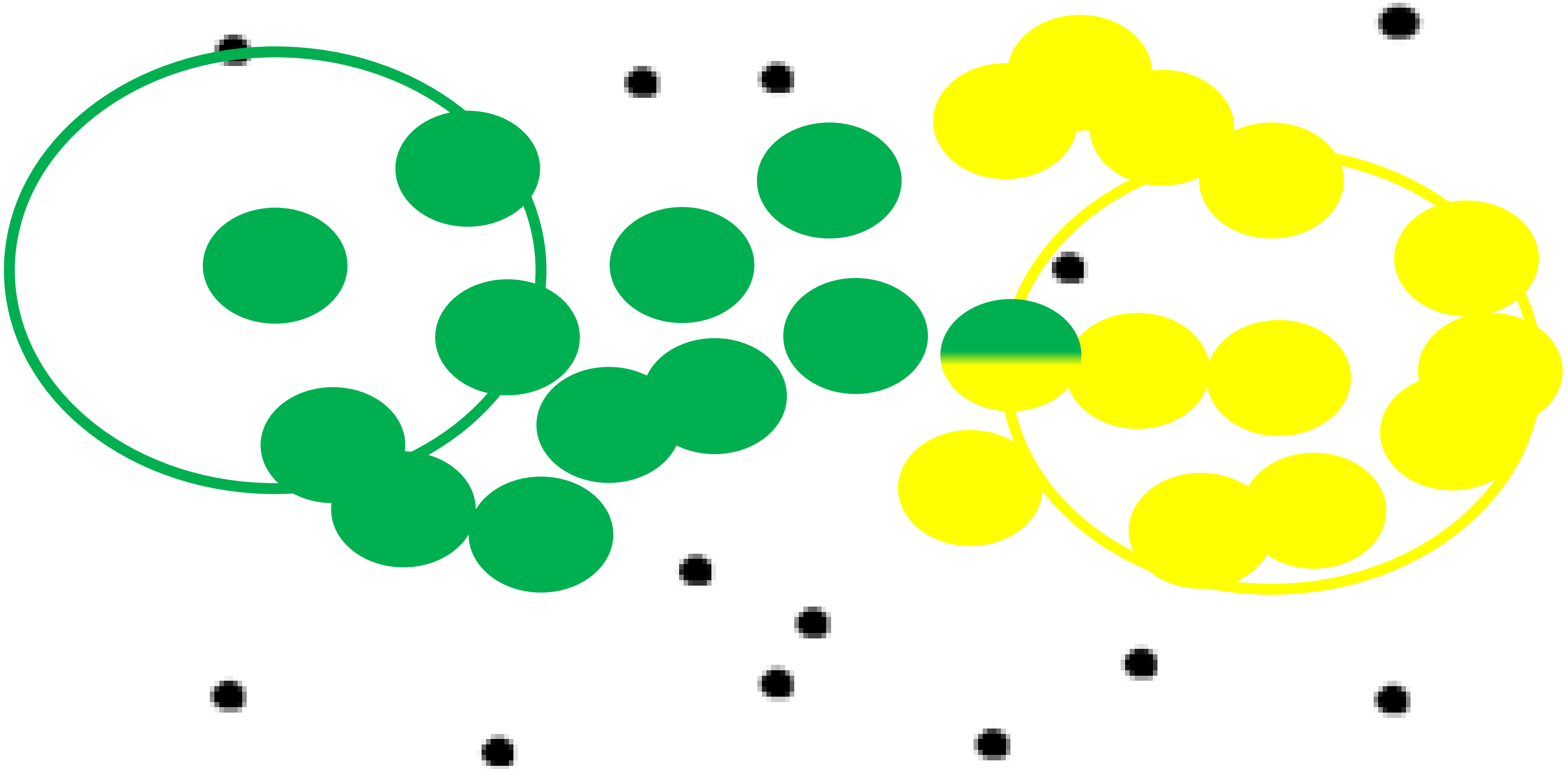


1. Start with an input of Spatially Referenced points
2. Partition the region into equal sized density regions across one dimension
3. Add the shadow region area of one Epsilon to all density regions



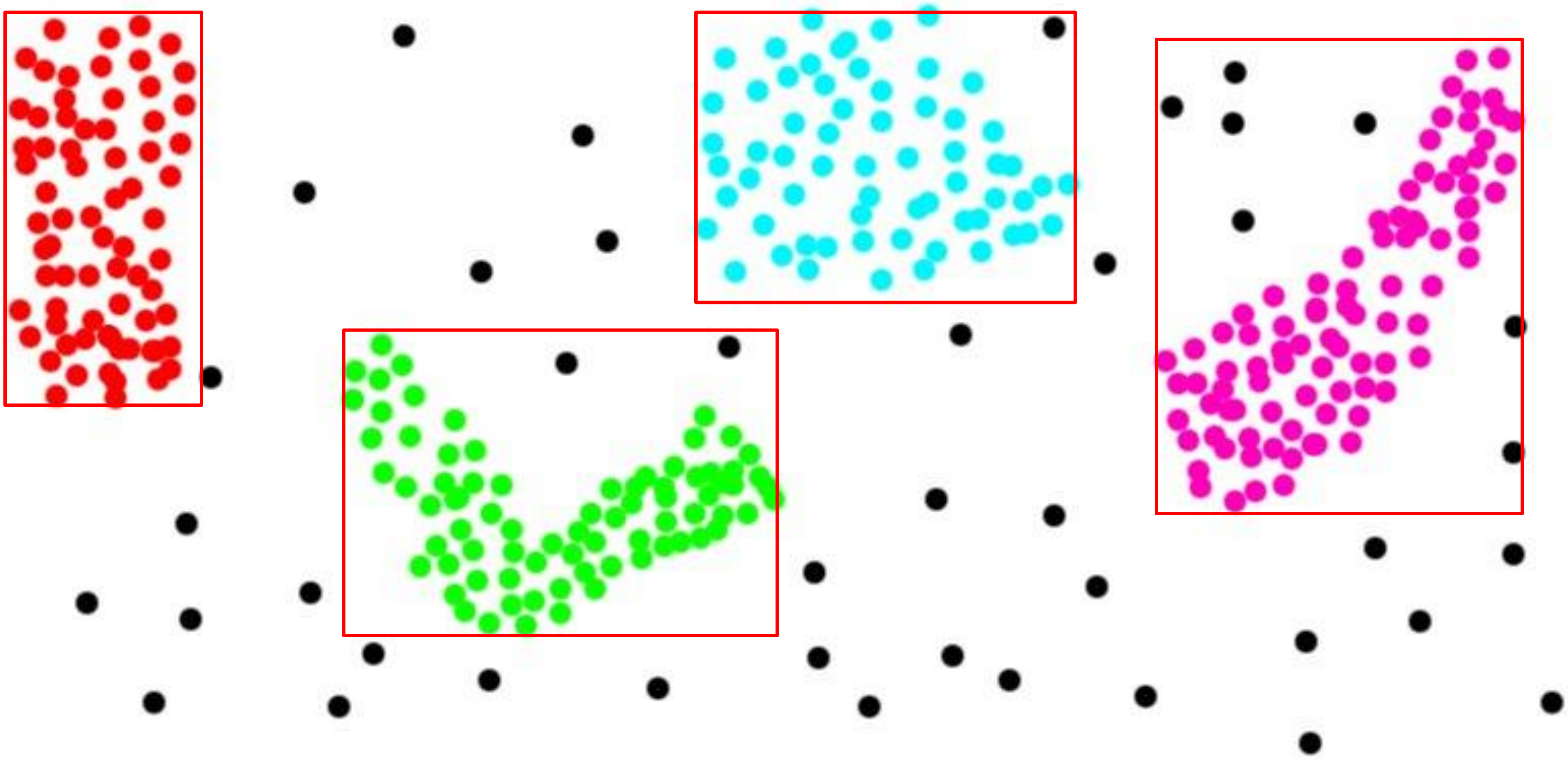
GPU DBSCAN Filter

Multiple clusters are expanded simultaneously



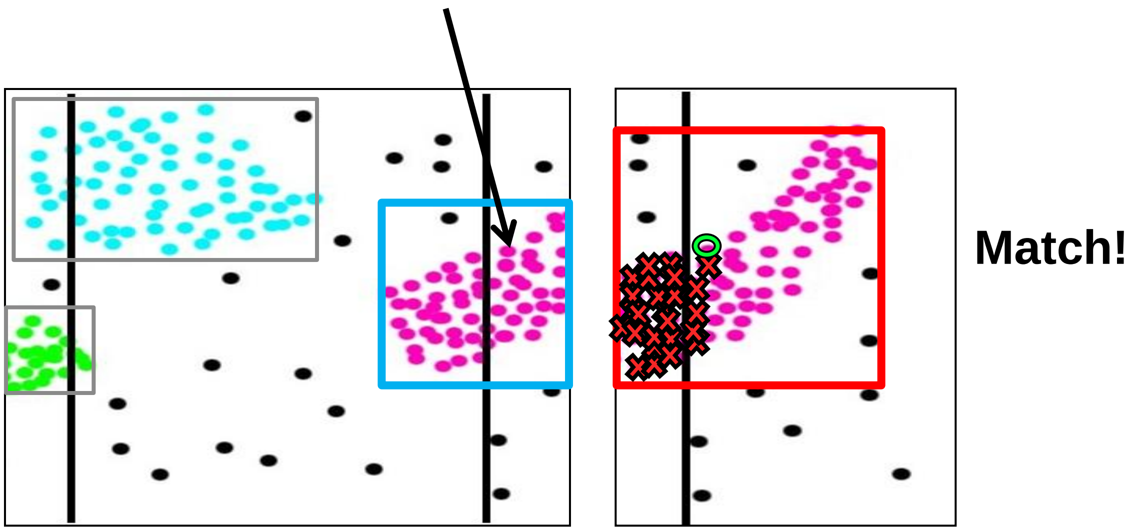
CUDA-DCLUST [09 – Böhm]

DrawBoundBox – CPU or GPU



Merge Step

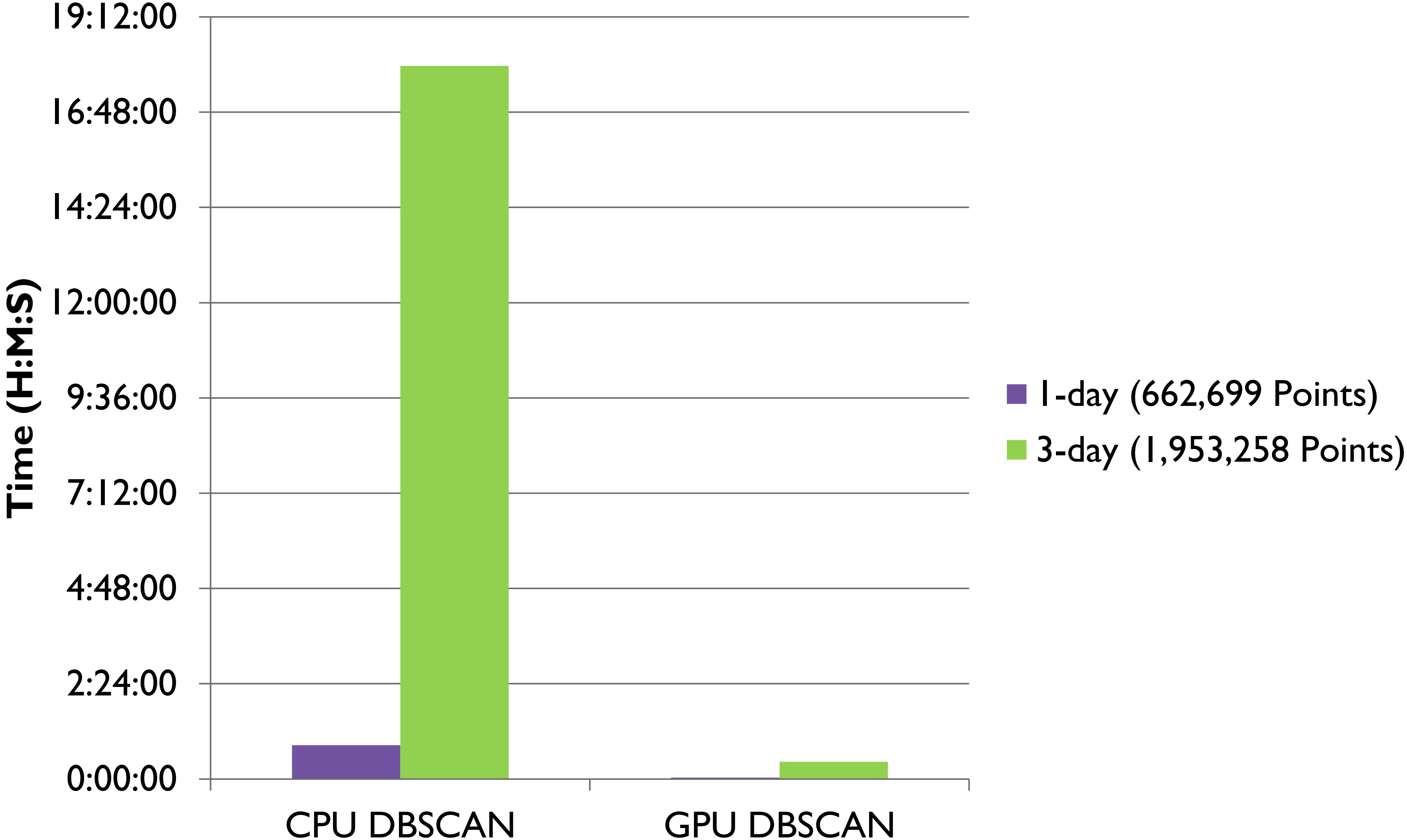
- Checks for merge if box within shadow
- At least one core point MUST be in common
- Iterate through ALL points in right cluster

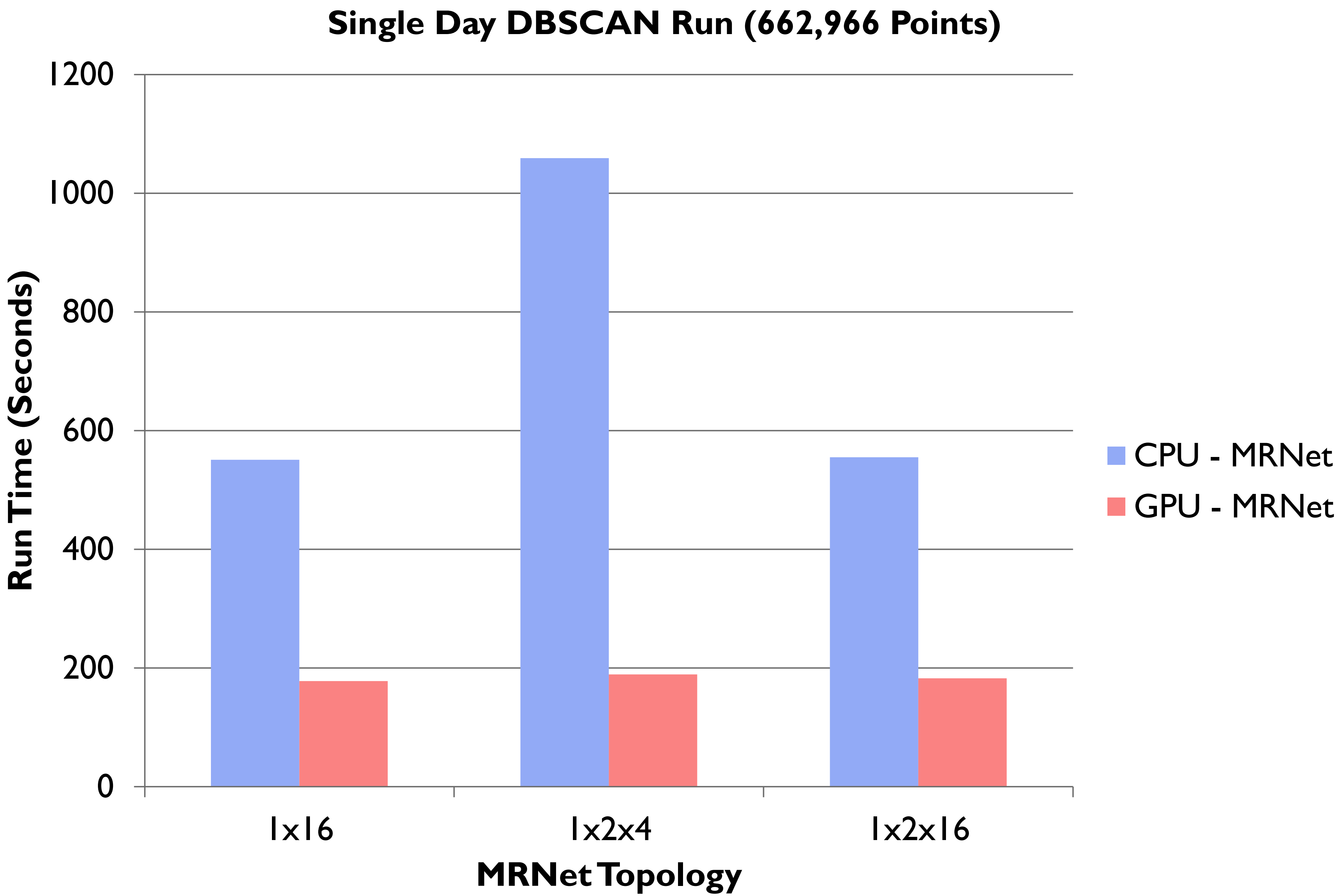


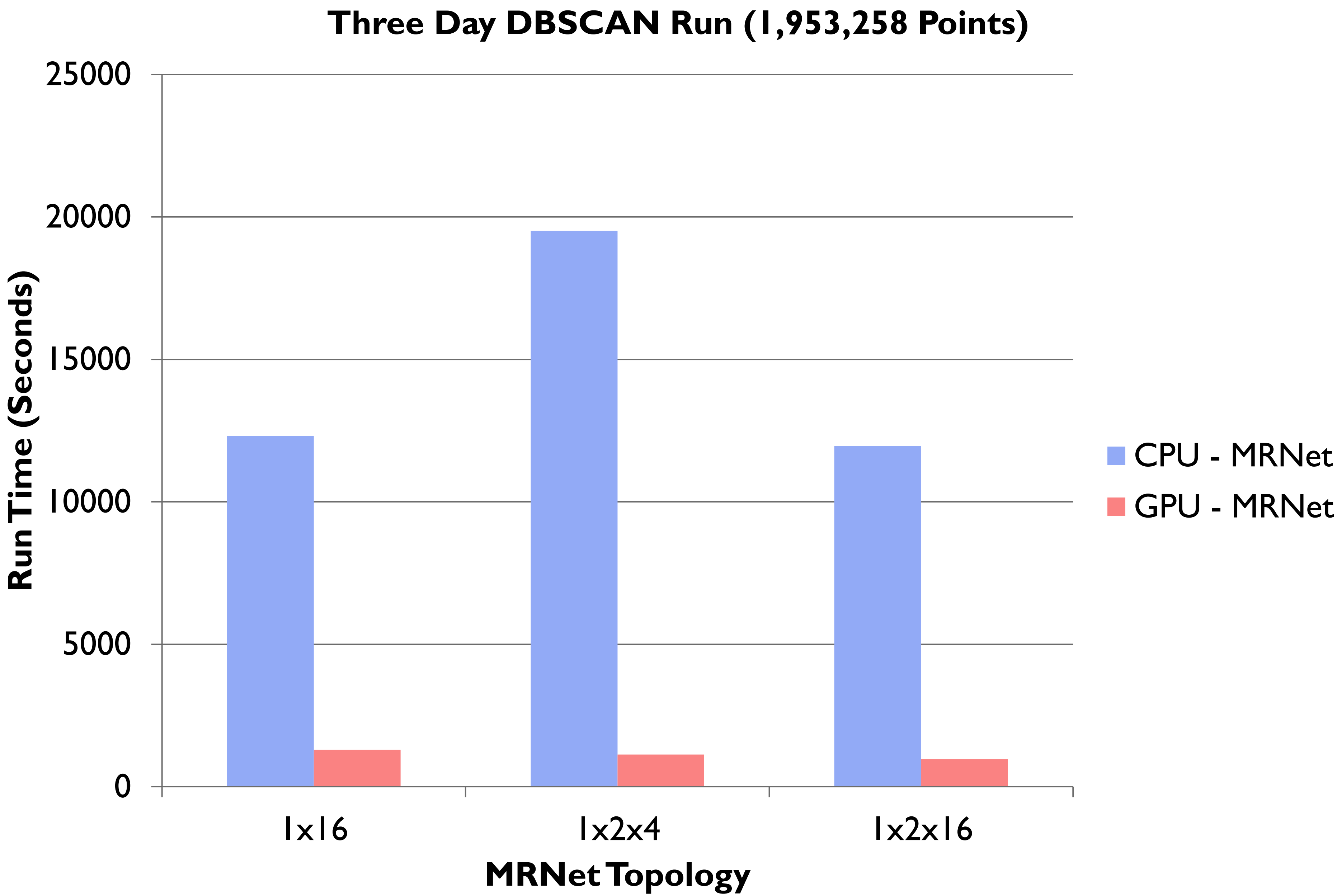
Preliminary Evaluation

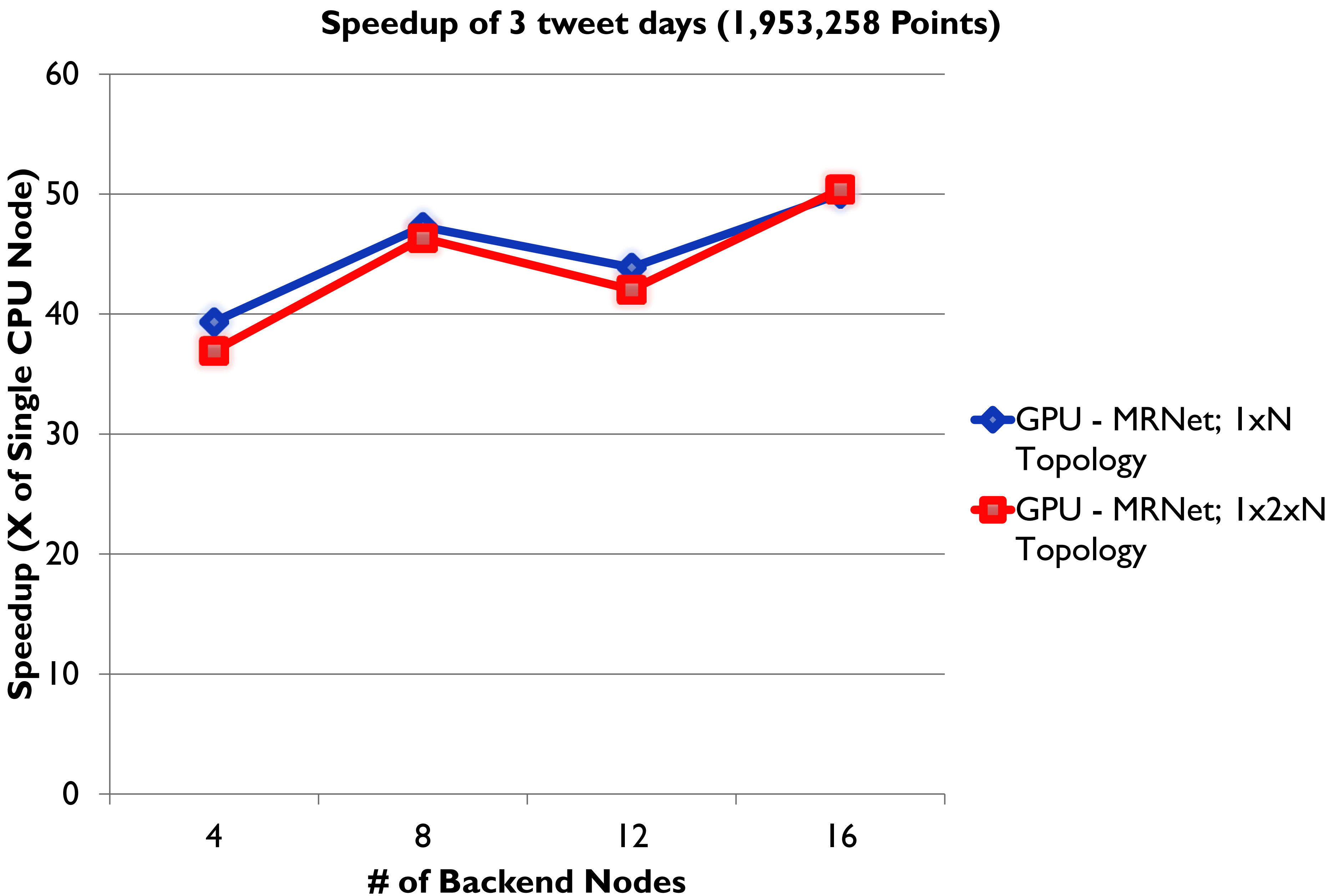
- Dataset: 1-3 "Tweet Days"
- Measuring:
  - Time to completion
- Algorithms:
  - Single-Threaded DBSCAN
  - MRNet w/DBSCAN filter
  - MRNet w/DBSCAN GPU filter

Single Node Performance



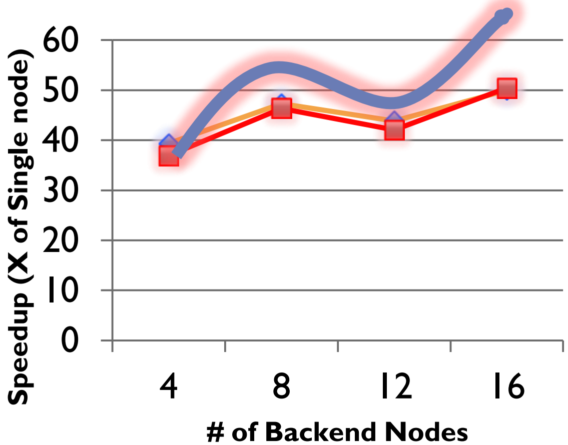

Single Day DBSCAN Run (662,966 Points)
1200
1000
800
600
400
200
0
Run Time (Seconds)
1x16
1x2x4
1x2x16
MRNet Topology
CPU - MRNet
GPU - MRNet

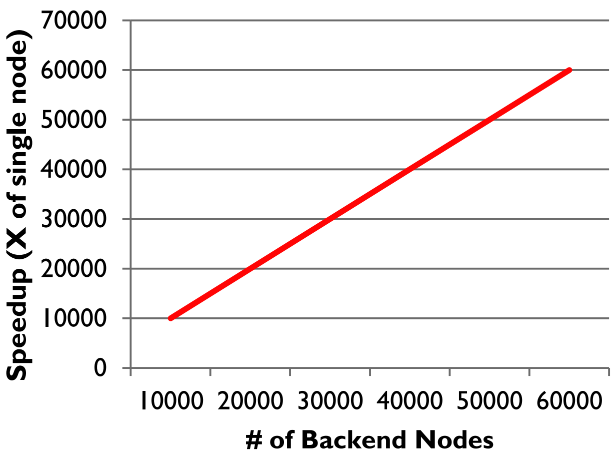

Three Day DBSCAN Run (1,953,258 Points)
25000
20000
15000
10000
5000
0
Run Time (Seconds)
1x16
1x2x4
1x2x16
MRNet Topology
CPU - MRNet
GPU - MRNet


Speedup of 3 tweet days (1,953,258 Points)
Speedup (X of Single CPU Node)
60
50
40
30
20
10
0
4
8
12
16
# of Backend Nodes
GPU - MRNet; 1xN Topology
GPU - MRNet; 1x2xN Topology

Discussion

Preliminary evaluation raises some important questions

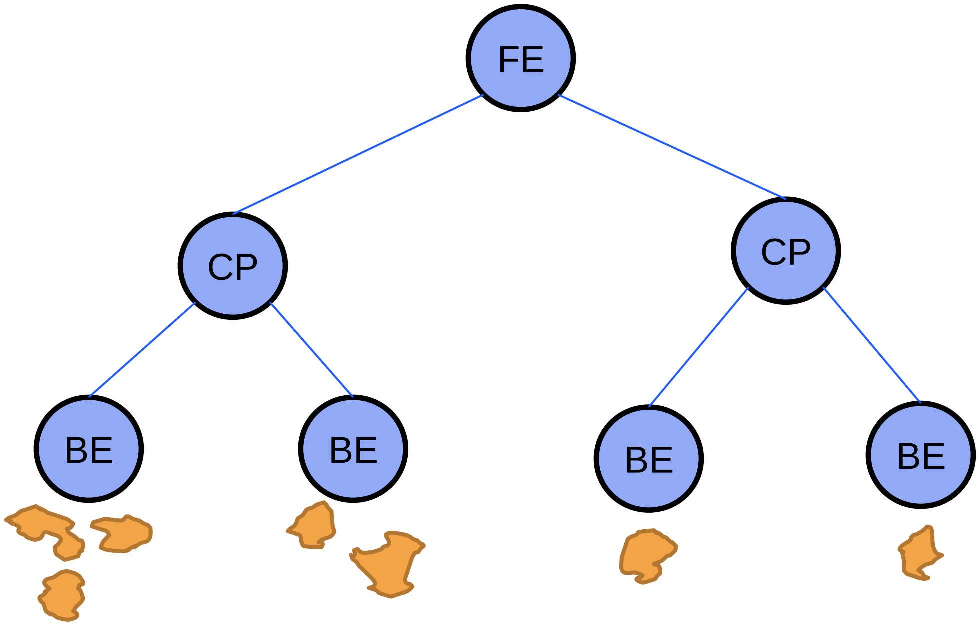




- What is causing DBSCAN to scale poorly?
- Why is GPU scaling somewhat erratic?
- How can we get to really large node counts?

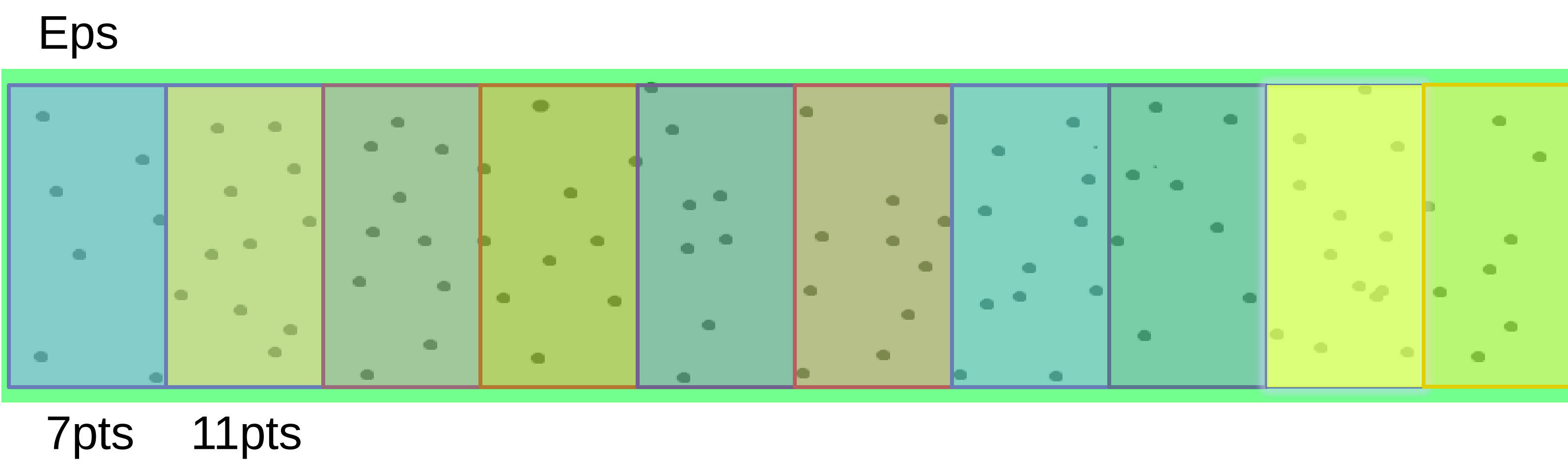
Causes Of Poor Scaling

- Merging Algorithm
  - Slow algorithm for detecting collisions between clusters. Worst case – $O(N^2)$
  - Internal node load imbalance due to partitioning.



Causes Of Poor Scaling

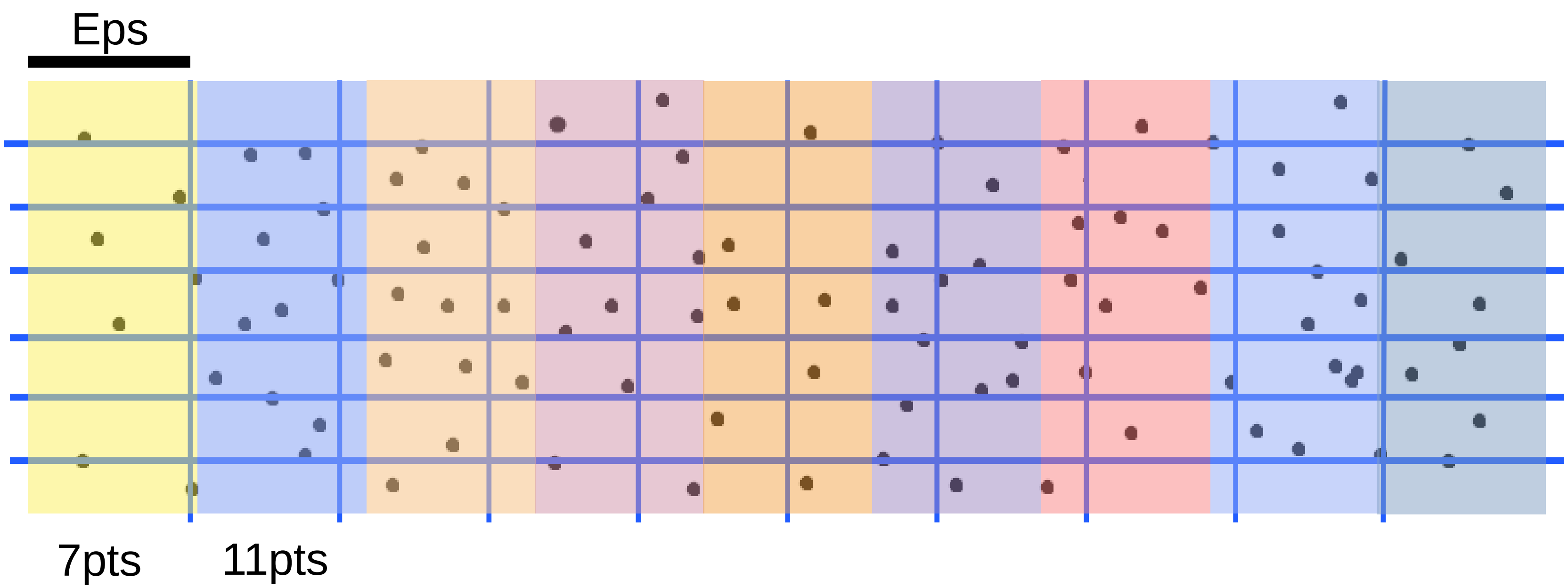
- Decomposition
  - Requiring a full survey of the data on a single node prior to performing the decomposition limits the maximum input data set size
  - Single dimensional decomposition limits the ability to evenly distribute workload.



Current Work

- Addressing Scaling Issues
  - Spatial Decomposition
  - Merging Algorithm

Spatial Decomposition

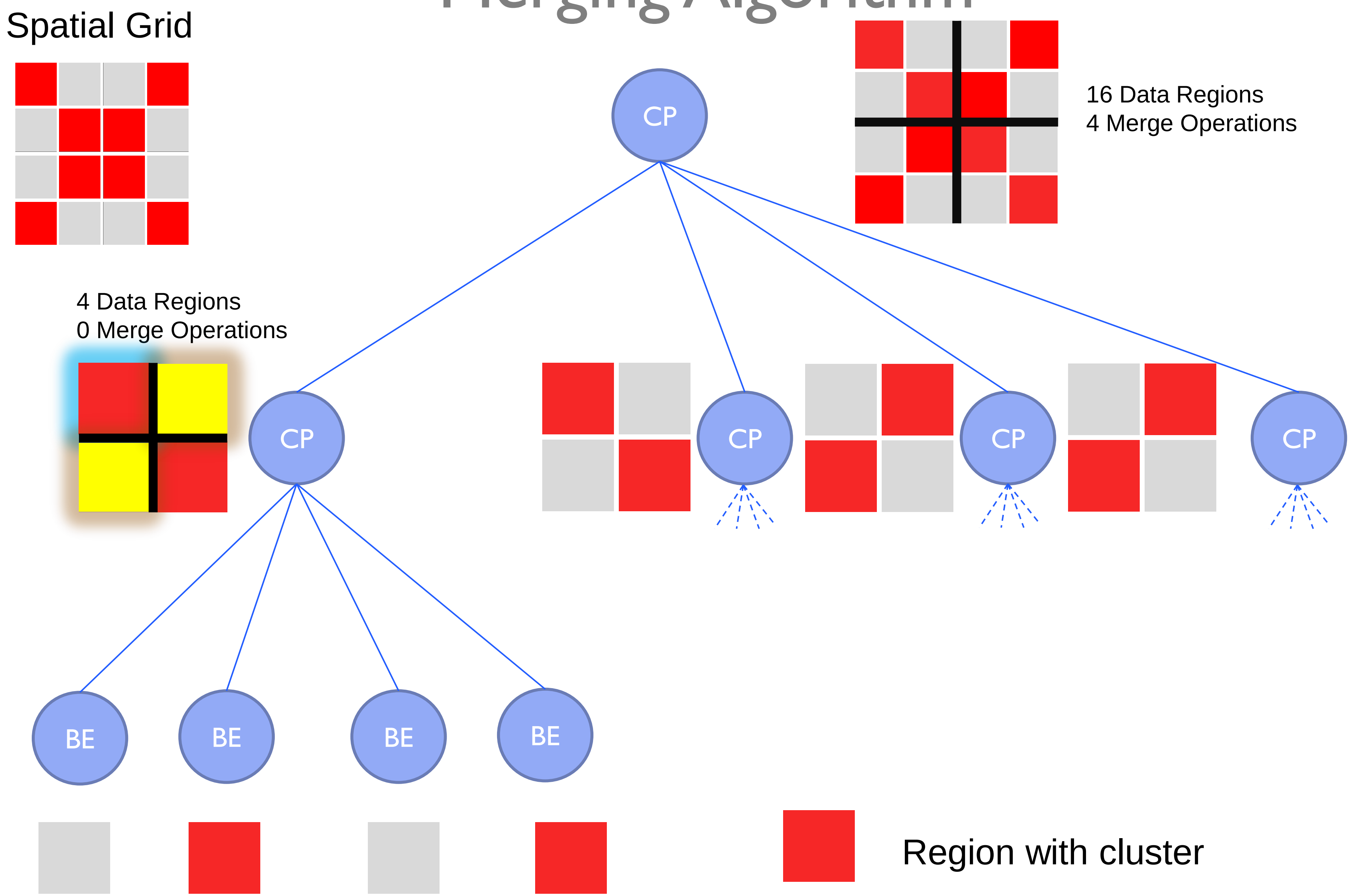


- 1D spatial decomposition has some severe limitations
  - Partitions can have wildly differing point counts
  - Number of partitions are limited by Epsilon
- 2D spatial decomposition allows for a finer grain breakdown of the regions.

Merging Algorithm

- Two major scalability challenges
  - Reducing the total number of required merges as data moves up the tree
  - Computational complexity of the merges

Merging Algorithm

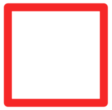


Region with cluster

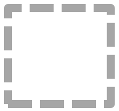
Merging Algorithm

Merge detection is currently to slow.

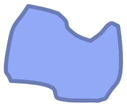
Can we improve our average case running time to avoid $O(N^2)$?



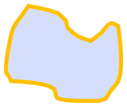
1-Eps Region



1-Eps Shadow Region

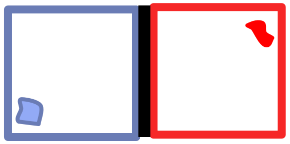


Region of cluster core points

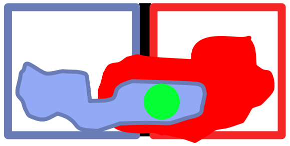


Region of cluster Non-Core points

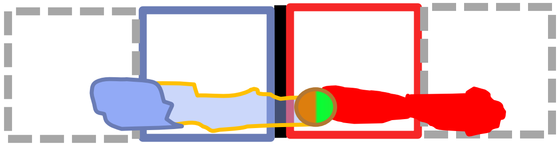
1. No points in common (no merge) – O(1)



2. Core points overlap – O(1)



3. Core/Non-Core point overlap – $O(N^2)$



Wrap Up

- Promising GPU results
- Lots of work left at the tree level
- We have delusions of grandeur

Questions?