Improved Parallel Algorithms for Density-Based Network Clustering

Mohsen Ghaffari
ETH

Silvio Lattanzi
Google

Slobodan Mitrović
MIT

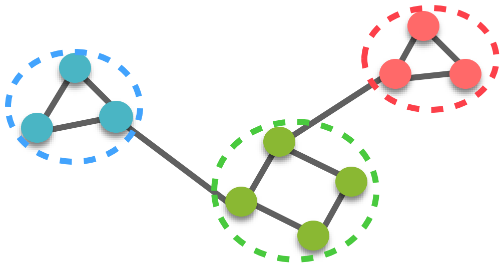
Why density-based network clustering?

A wide range of applications in data mining:

Why density-based network clustering?

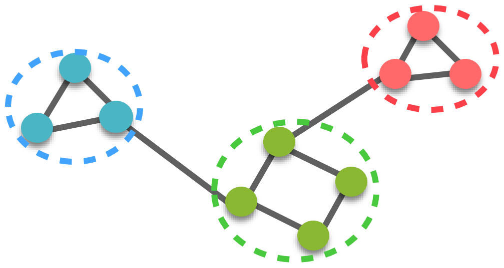
A wide range of applications in data mining:

Community detection
[Leskovec et al. '08; Chen & Saad '12;
Gionis & Tsourakakis '15; Mitzenmacher et al. '15]



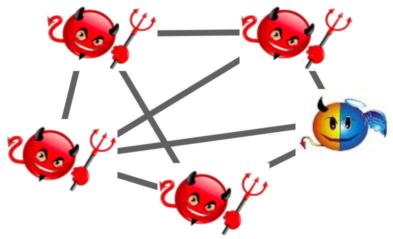
Why density-based network clustering?

A wide range of applications in data mining:



Community detection

[Leskovec et al. ‘08; Chen & Saad ‘12; Gionis & Tsourakakis ’15; Mitzenmacher et al. ‘15]

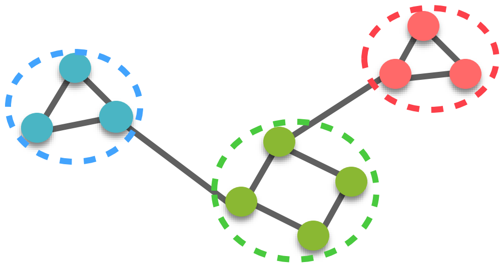


Spam detection

[Gibson et al. ‘05]

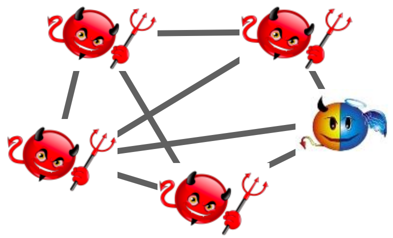
Why density-based network clustering?

A wide range of applications in data mining:



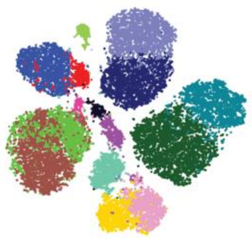
Community detection

[Leskovec et al. '08; Chen & Saad '12;
Gionis & Tsourakakis '15; Mitzenmacher et al. '15]



Spam detection

[Gibson et al. '05]



Computational biology

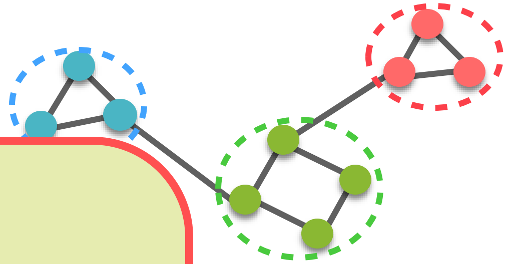
[Altaf-Ul-Amin et al. '06; Fratkin et al. '06;
Saha et al. '10]

...

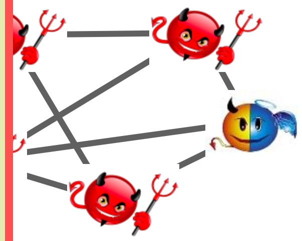
Why density-based network clustering?

A wide range of applications in data mining:

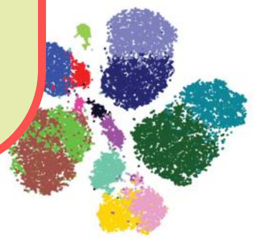
Community detection
[Leskovec et al. '08
Gionis & Tsouraka



Spam detecti
[Gibson et al. '05]



Computation
[Altaf-Ul-Amin et a
Saha et al. '10]



...

We study:
1. Densest subgraph
2. k-core decomposition
3. Graph orientation

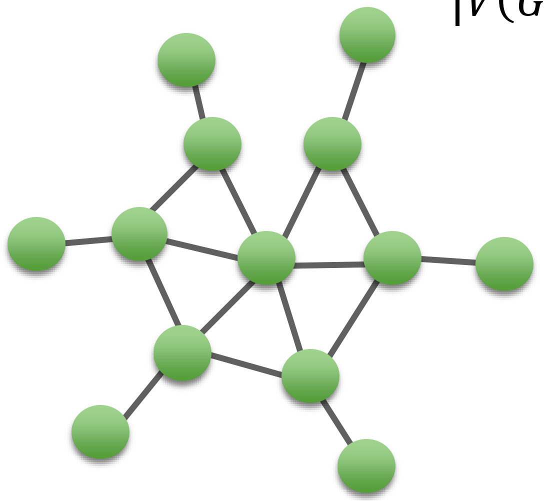
Densest subgraph

Goal: Given a graph G, find a subgraph H such that |E(H)| / |V(H)| is *maximized*.

Densest subgraph

Goal: Given a graph G, find a subgraph H such that |E(H)| / |V(H)| is *maximized*.

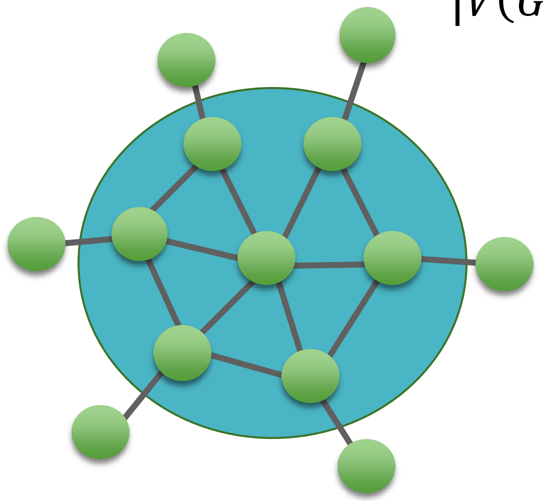
$$\frac{|E(G)|}{|V(G)|} = \frac{17}{13}$$



Densest subgraph

Goal: Given a graph G, find a subgraph H such that |E(H)| / |V(H)| is *maximized*.

$$\frac{|E(G)|}{|V(G)|} = \frac{17}{13}$$



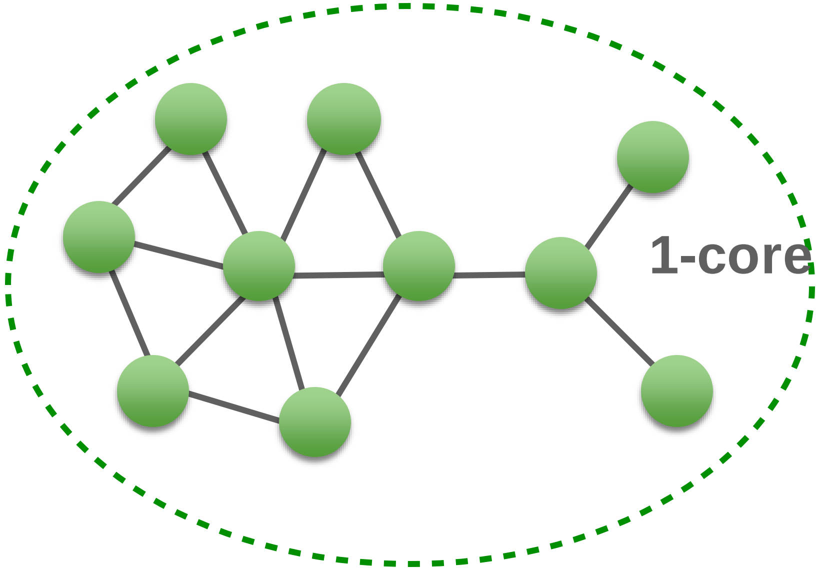
$$\frac{|E(H)|}{|V(H)|} = \frac{11}{7}$$

k-core decomposition

Goal: Given k, find a maximal subgraph of minimum degree at least k. (*k-core*)

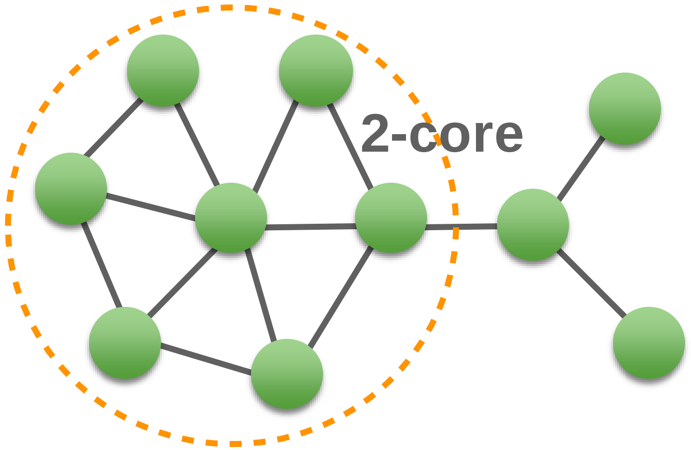
k-core decomposition

Goal: Given k, find a maximal subgraph of minimum degree at least k. (*k-core*)



k-core decomposition

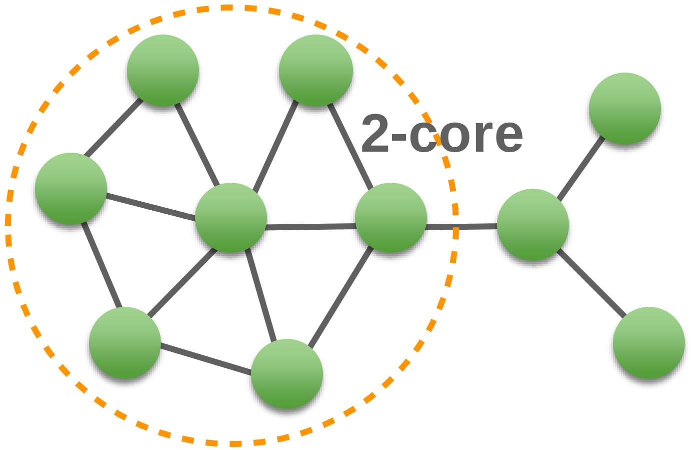
Goal: Given k, find a maximal subgraph of minimum degree at least k. (*k-core*)



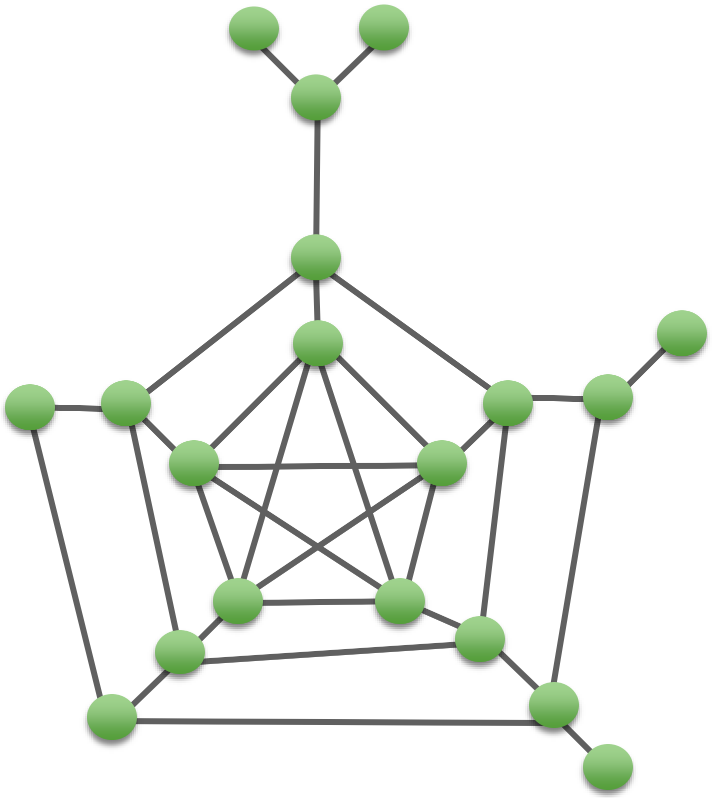
k-core decomposition

Goal: Given k, find a maximal subgraph of minimum degree at least k. (*k-core*)

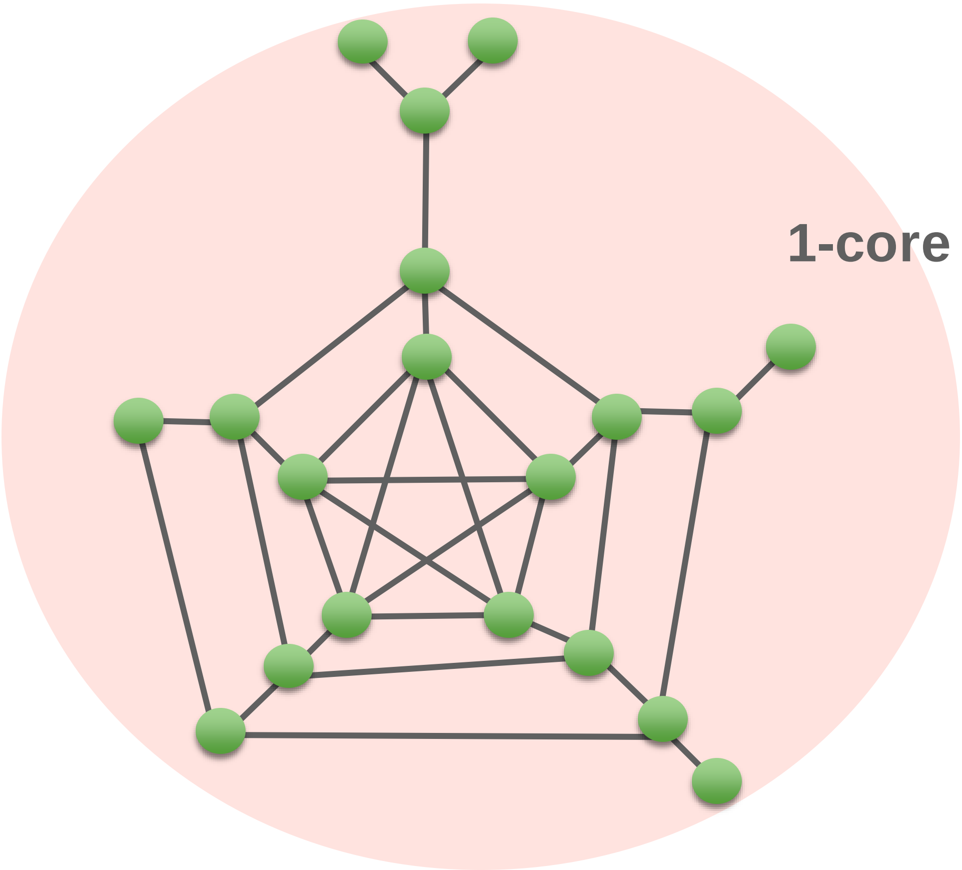
The coreness number of a vertex v is the maximum k for which v is part of the k-core.



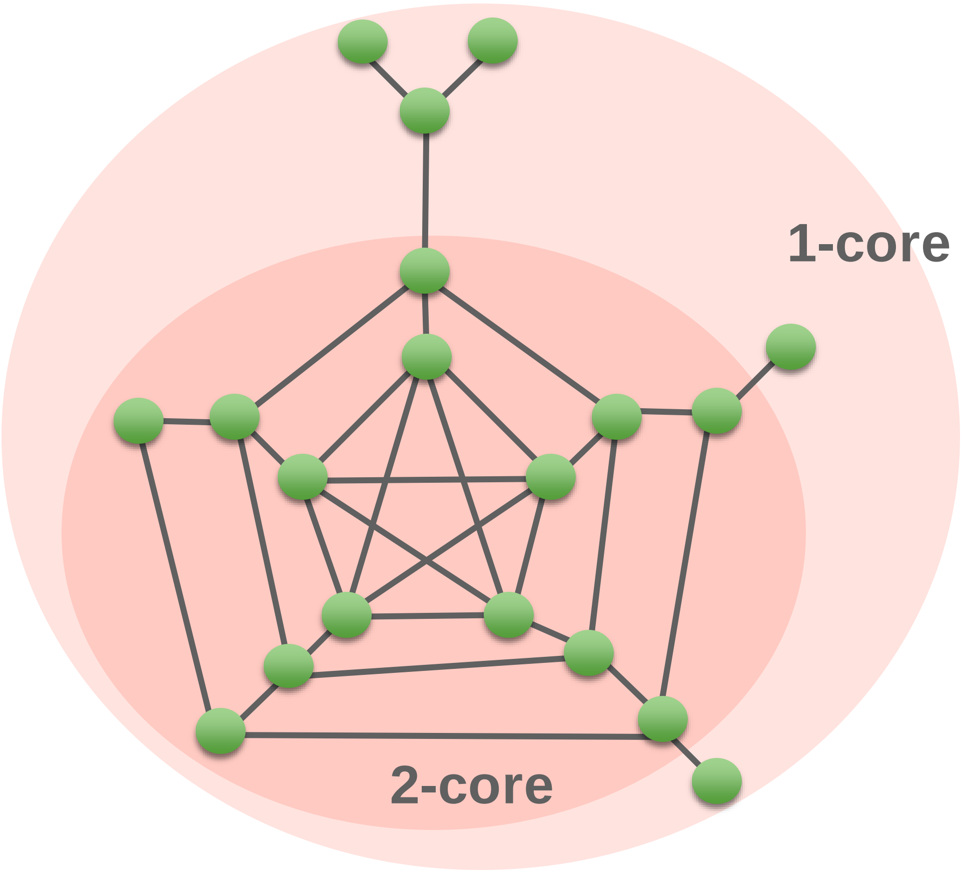
Hierarchical clustering via k-core



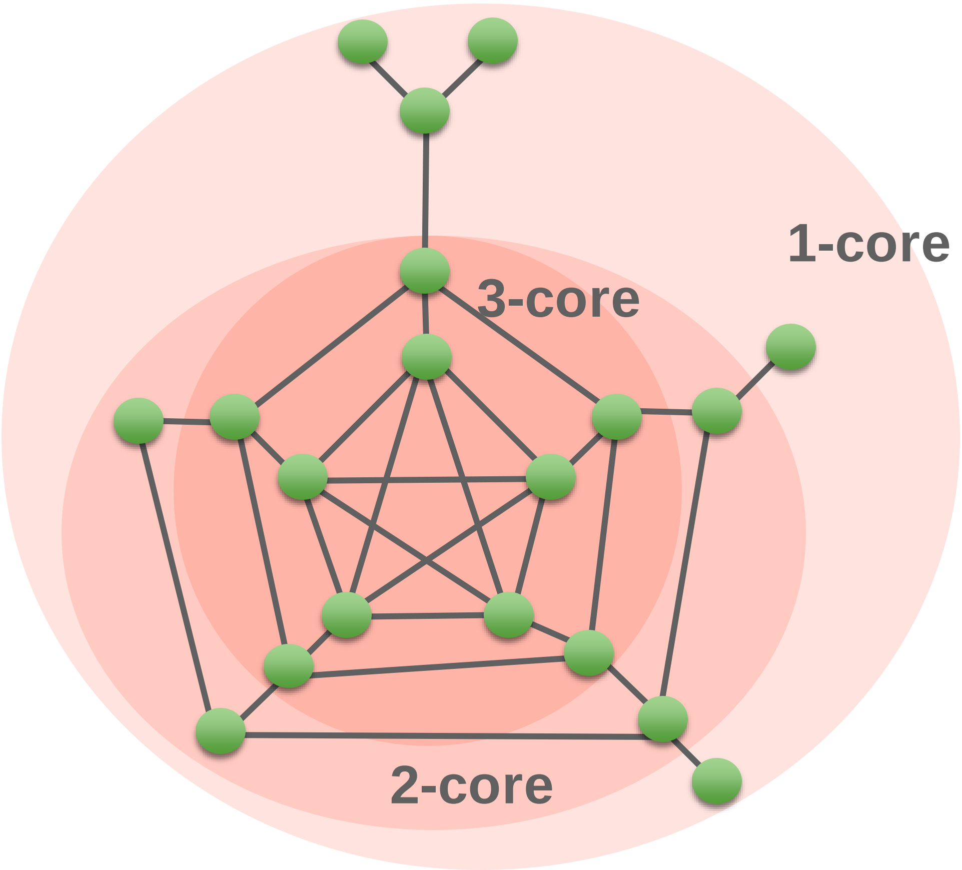
Hierarchical clustering via k-core



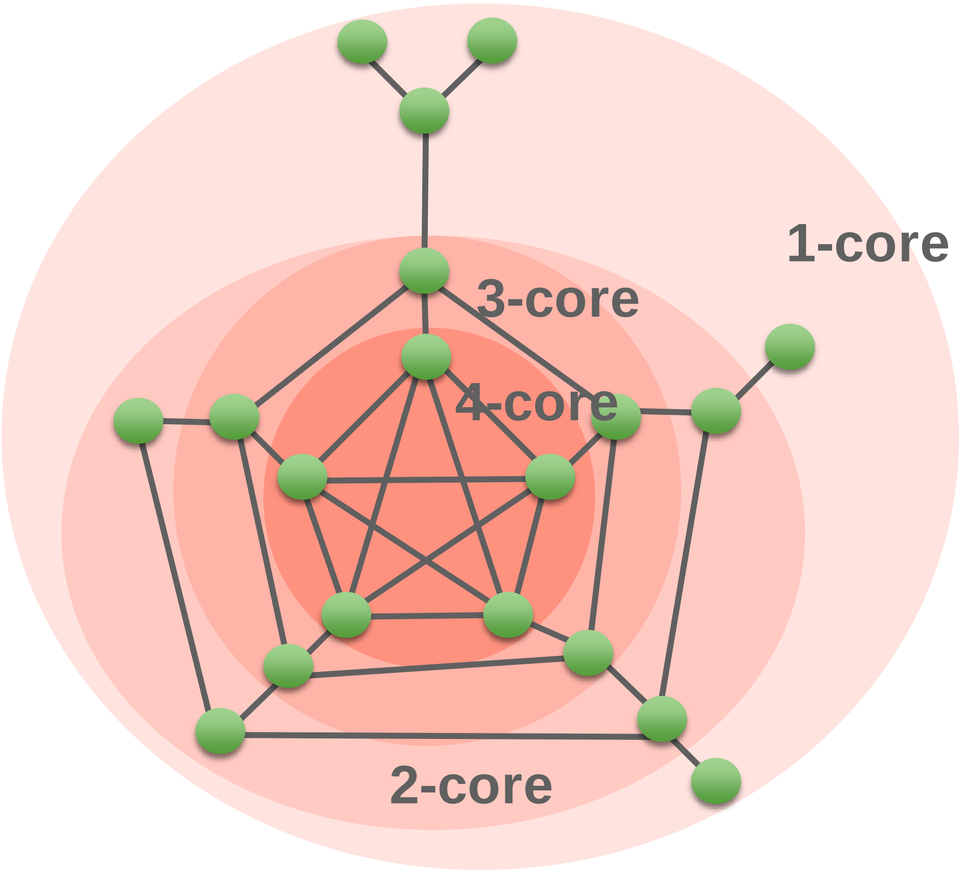
Hierarchical clustering via k-core



Hierarchical clustering via k-core

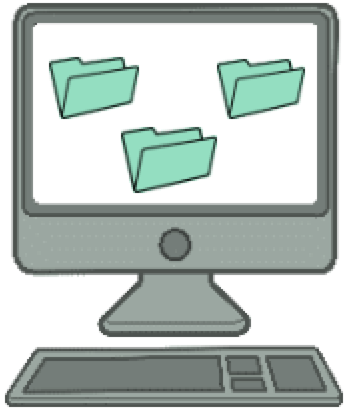


Hierarchical clustering via k-core

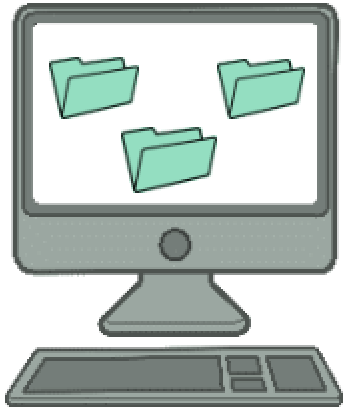


How to compute these clusters?

Traditional

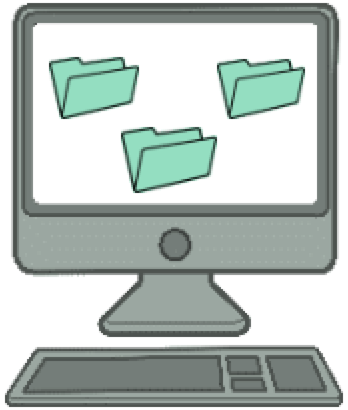


Traditional



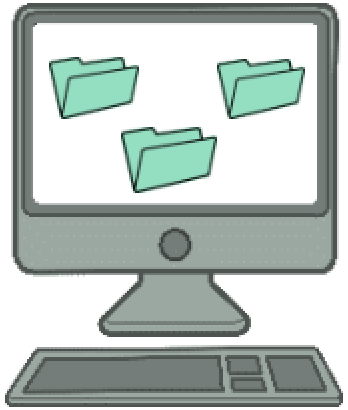
Algorithms performed sequentially.

Traditional

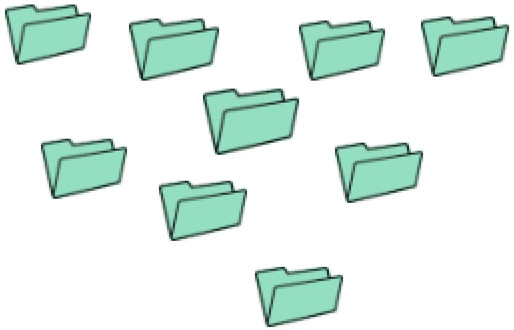


Algorithms performed

Traditional

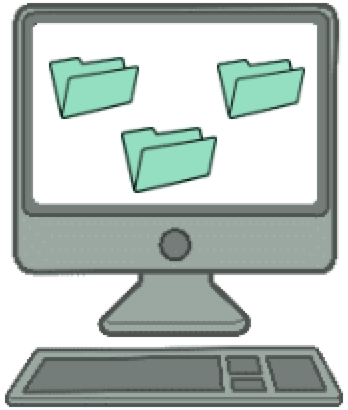


Modern



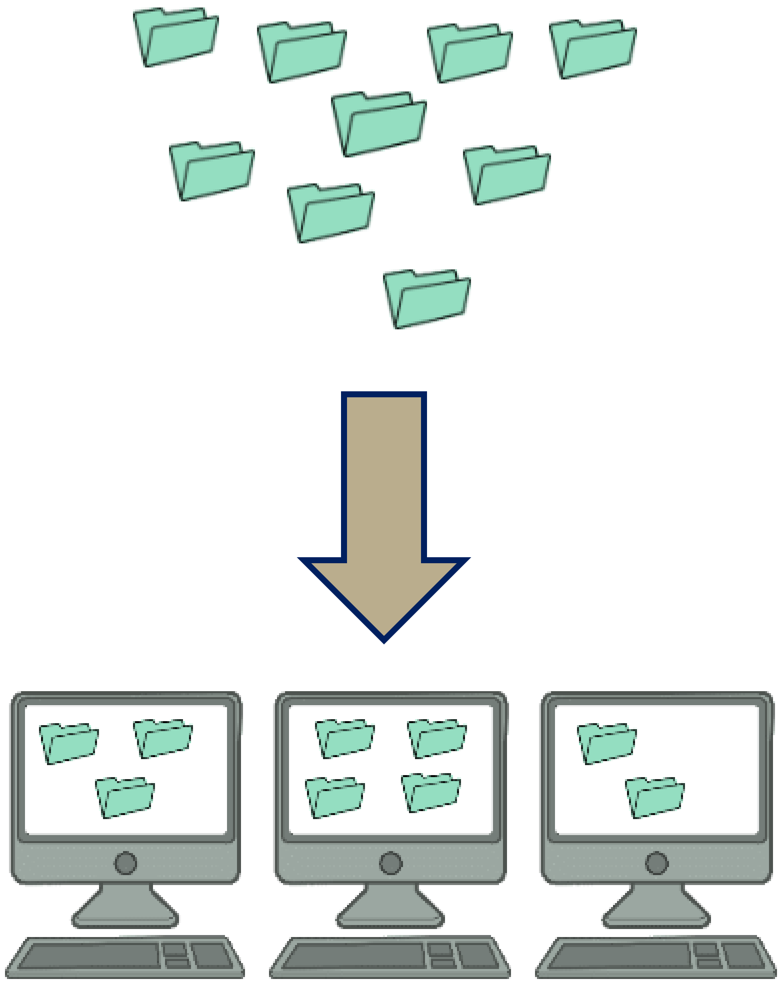
Algorithms performed

Traditional

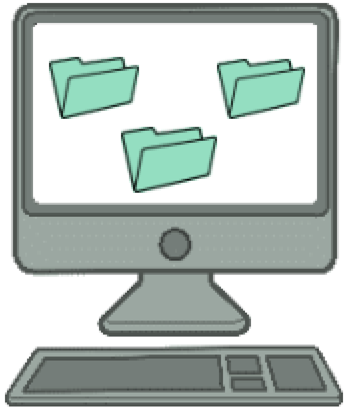


Algorithms performed

Modern

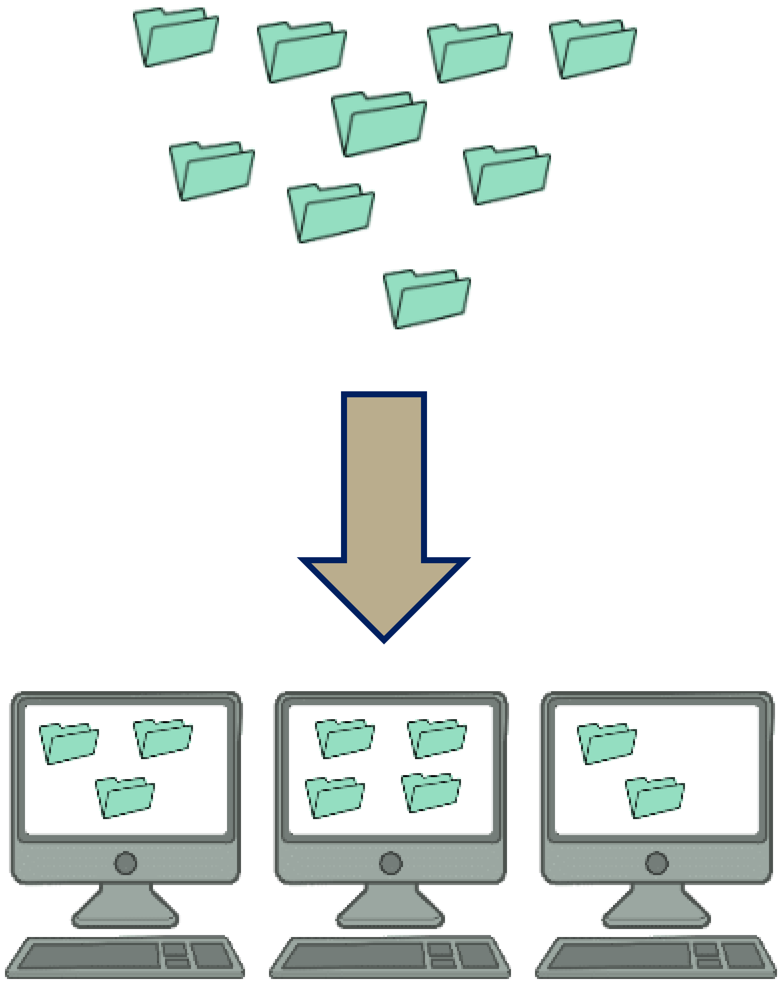


Traditional



Algorithms performed

Modern



Massively Parallel Computation (MPC) model

An approach to handling massive data

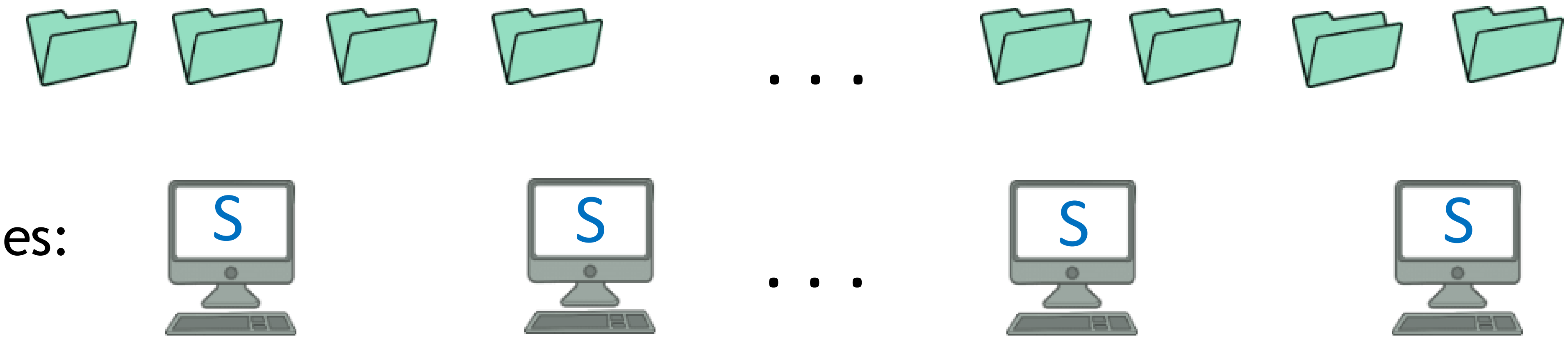
Examples:

- MapReduce [DG, '04, '08]
- Hadoop [W, '12]
- Pregel [Google, '09]
- Dryad [IBYBF, '07]
- Spark [ZCFSS, '10]

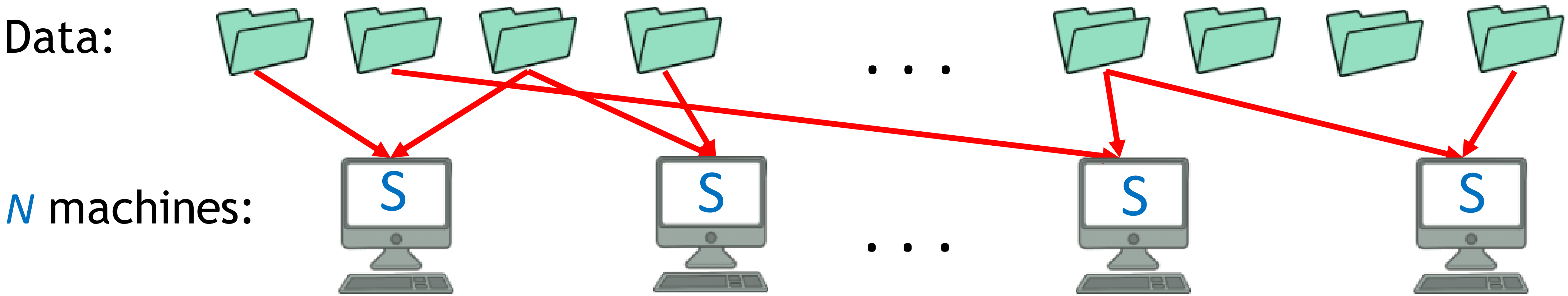
Massively Parallel Computation (MPC) round

Data:

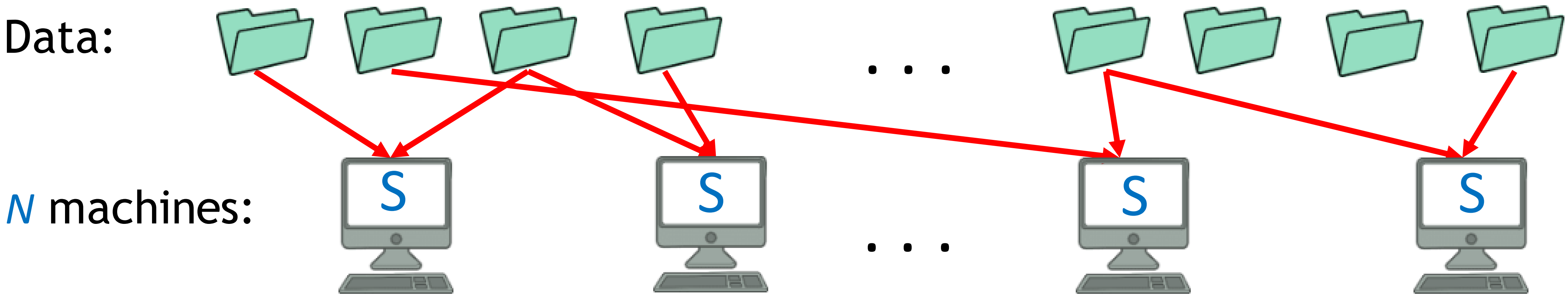
N machines:



Massively Parallel Computation (MPC) round

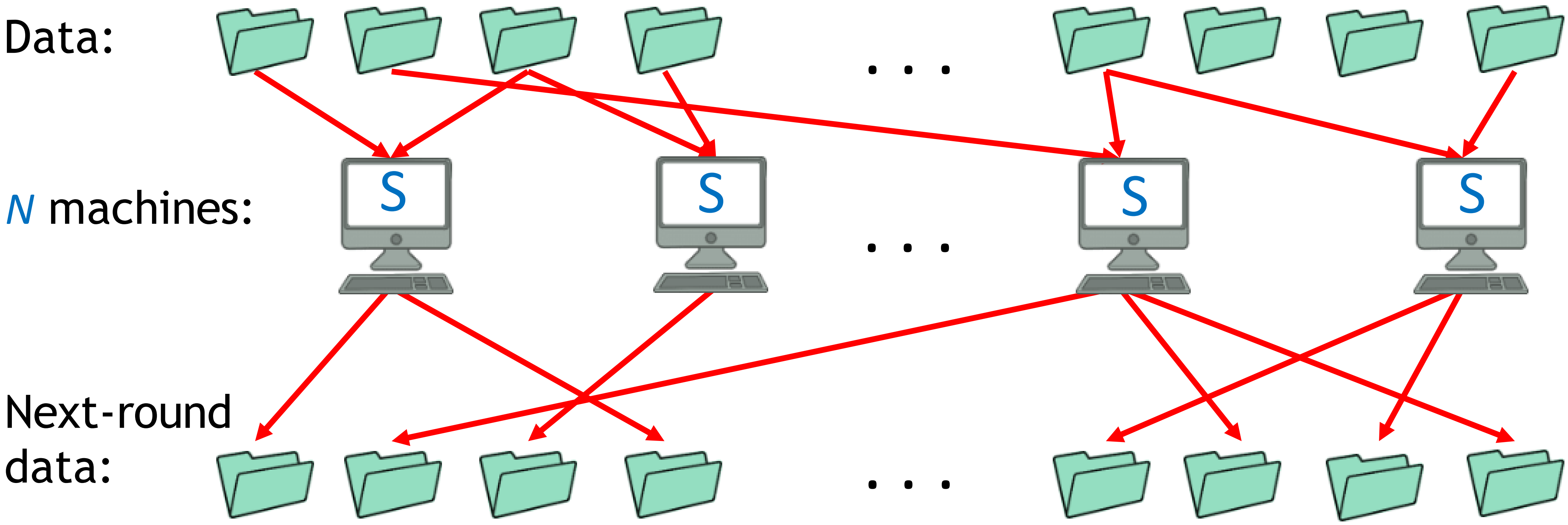


Massively Parallel Computation (MPC) round

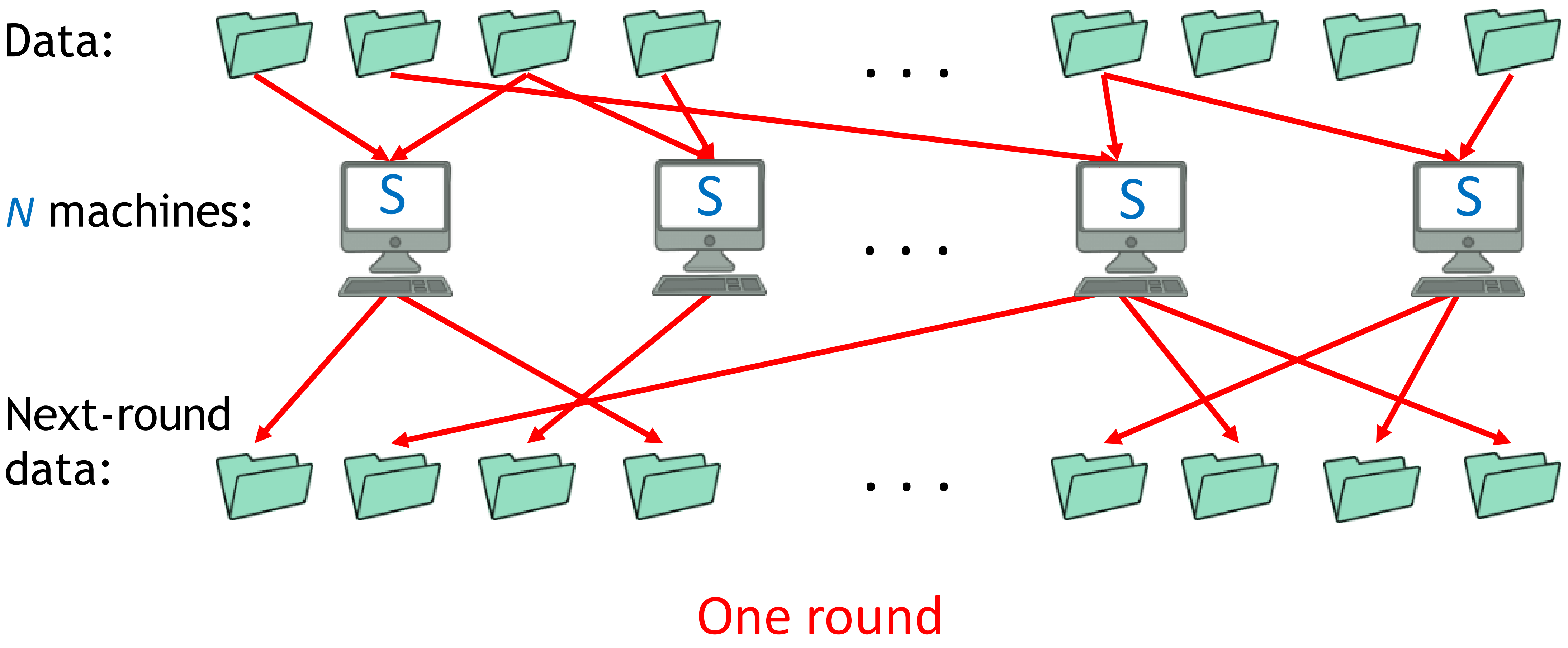


process data locally

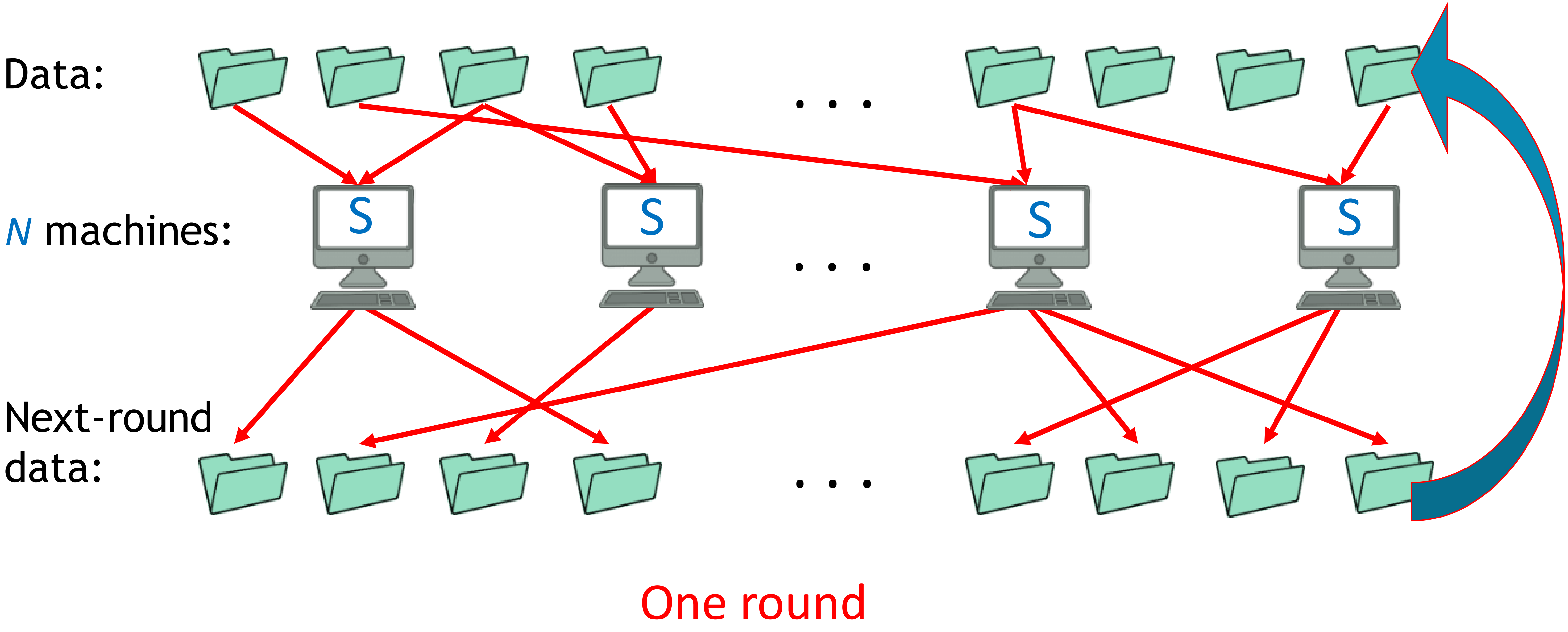
Massively Parallel Computation (MPC) round



Massively Parallel Computation (MPC) round



Massively Parallel Computation (MPC) round



Related work

1. *Densest Subgraph in Streaming and MapReduce*
   Bahmani, Kumar, Vassilvitskii, VLDB 2012.
2. *Space- and Time-Efficient Algorithm for Maintaining Dense Subgraphs on One-Pass Dynamic Streams*
   Bhattacharya, Henzinger, Nanongkai, Tsourakakis, STOC 2015.
3. *Efficient Densest Subgraph Computation in Evolving Graphs*
   Epasto, Lattanzi, Sozio, WWW 2015.
4. *Densest Subgraph in Dynamic Graph Streams*
   McGregor, Tench, Vorotnikova, Vu, MFCS 2015.
5. *Brief Announcement: Applications of Uniform Sampling: Densest Subgraph and Beyond*
   Esfandiari, Hajiaghayi, Woodruff, SPAA 2016.
6. *Efficient primal-dual graph algorithms for MapReduce*
   Bahmani, Goel, Munagala, Workshop on Algorithms and Models for the Web-Graph 2014.
7. *Parallel and streaming algorithms for k-core decomposition*
   Esfandiari, Lattanzi, and Mirrokni, ICML 2018.
8. *Streaming algorithms for k-core decomposition*
   Saríyüce, Gedik, Jacques, Wu, Çatalyürek, VLDB 2013.
9. *Distributed-Core View Materialization and Maintenance for Large Dynamic Graphs*
   Aksu, Canim, Chang, Korpeoglu, Ulusoy, TKDE 2014.

Our results

n = number of vertices

Theorem 1

$(1+\epsilon)$-approximate k-core decomposition can be obtained in $O(\log\log n)$ MPC rounds with $\tilde{O}(n)$ memory per machine.

Theorem 3

$(1+\epsilon)$-approximate densest subgraph can be obtained in $\tilde{O}(\sqrt{\log n})$ MPC rounds with $O(n^{\delta})$ memory per machine and the total memory of $\tilde{O}(\max\{n^{1+\delta}, m\})$.

Theorem 2

$(2+\epsilon)$-approximate k-core decomposition can be obtained in $\tilde{O}(\sqrt{\log n})$ MPC rounds with $O(n^{\delta})$ memory per machine and the total memory of $\tilde{O}(\max\{n^{1+\delta}, m\})$.

Theorem 4

For a graph of arboricity λ, a $(2+\epsilon)\lambda$ orientation can be obtained in $\tilde{O}(\sqrt{\log n})$ MPC rounds with $O(n^{\delta})$ memory per machine and the total memory of $\tilde{O}(\lambda n)$.

Our results

n = number of vertices

Theorem 1

$(1+\epsilon)$-approximate k-core decomposition can be obtained in $O(\log\log n)$ MPC rounds with $\tilde{O}(n)$ memory per machine.

Theorem 3

$(1+\epsilon)$-approximate densest subgraph can be obtained in $\tilde{O}(\sqrt{\log n})$ MPC rounds with $O(n^{\delta})$ memory per machine and the total memory of $\tilde{O}(\max\{n^{1+\delta}, m\})$.

Poster: Wed, Pacific Ballroom #166

Theorem 2

$(2+\epsilon)$-approximate k-core decomposition can be obtained in $\tilde{O}(\sqrt{\log n})$ MPC rounds with $O(n^{\delta})$ memory per machine and the total memory of $\tilde{O}(\max\{n^{1+\delta}, m\})$.

Theorem 4

For a graph of arboricity λ, a $(2+\epsilon)\lambda$ orientation can be obtained in $\tilde{O}(\sqrt{\log n})$ MPC rounds with $O(n^{\delta})$ memory per machine and the total memory of $\tilde{O}(\lambda n)$.

Next

Theorem 1

$(1+\epsilon)$-approximate k-core decomposition can be obtained in $O(\log\log n)$ MPC rounds with $\tilde{O}(n)$ memory per machine.

Next

Theorem 1

$(1+\epsilon)$-approximate k-core decomposition can be obtained in $O(\log \log n)$ MPC rounds with $\tilde{O}(n)$ memory per machine.

High-level idea:
Simulate the sequential algorithm.

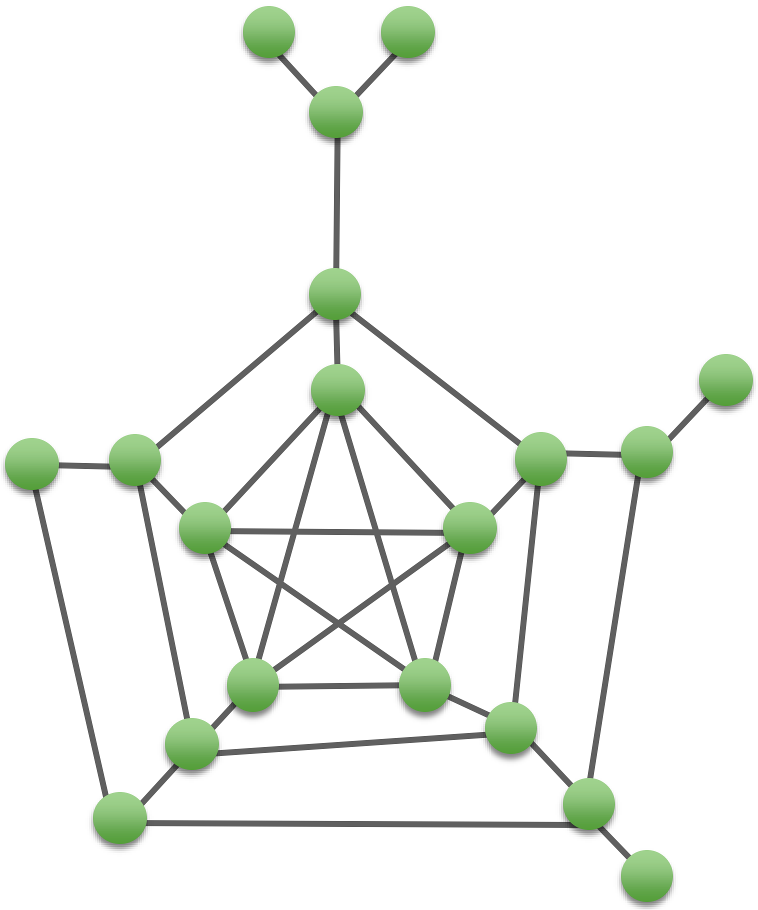
The sequential algorithm

- Given a threshold k, repeatedly remove all the vertices of degree less than k.
- The coreness value of a vertex is the largest k for which it is not removed.

The sequential algorithm

- Given a threshold k, repeatedly remove all the vertices of degree less than k.
- The coreness value of a vertex is the largest k for which it is not removed.

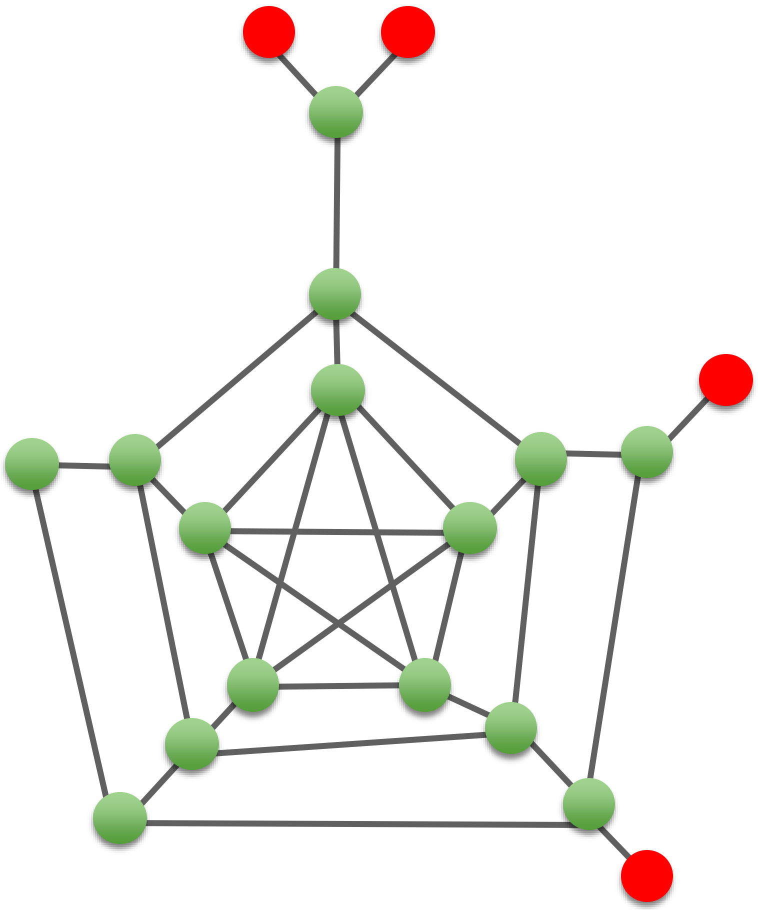
k=2



The sequential algorithm

- Given a threshold k, repeatedly remove all the vertices of degree less than k.
- The coreness value of a vertex is the largest k for which it is not removed.

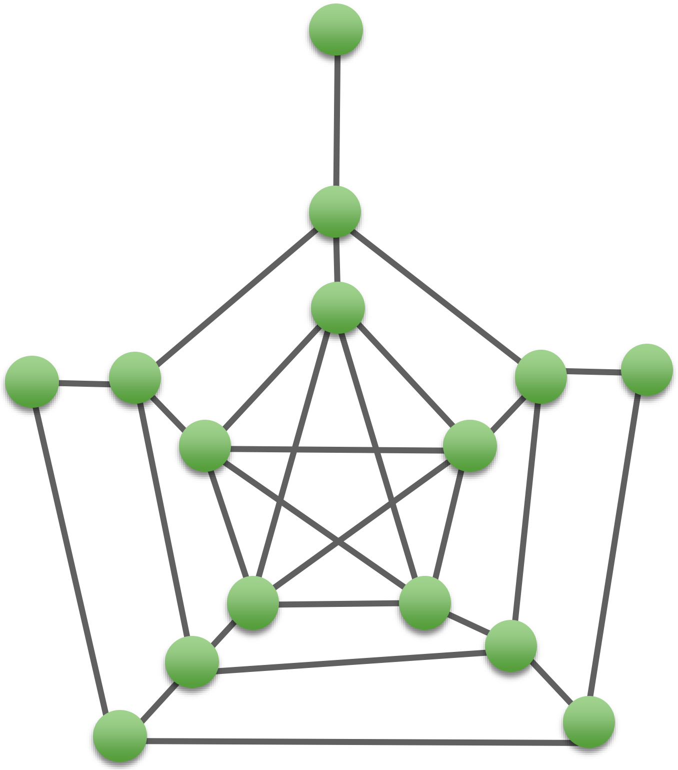
k=2



The sequential algorithm

- Given a threshold k, repeatedly remove all the vertices of degree less than k.
- The coreness value of a vertex is the largest k for which it is not removed.

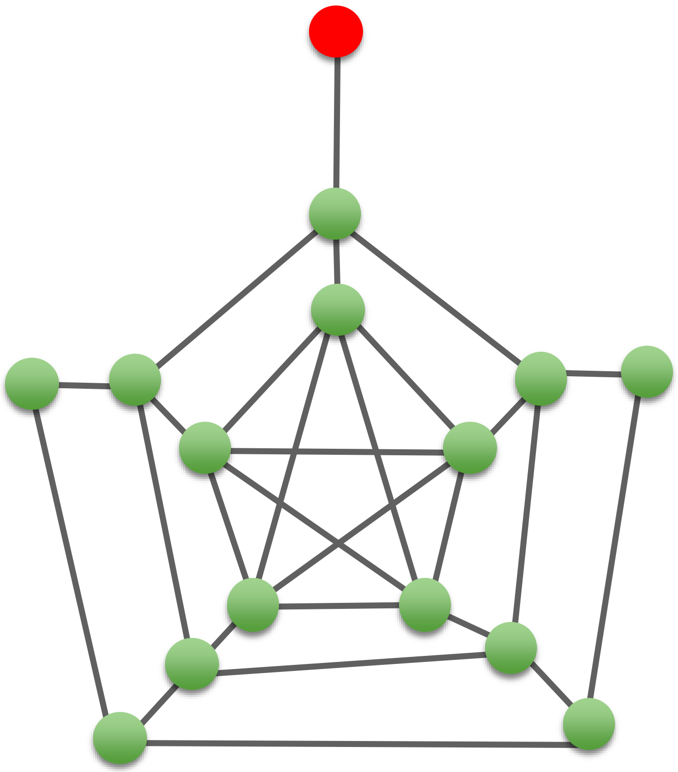
k=2



The sequential algorithm

- Given a threshold k, repeatedly remove all the vertices of degree less than k.
- The coreness value of a vertex is the largest k for which it is not removed.

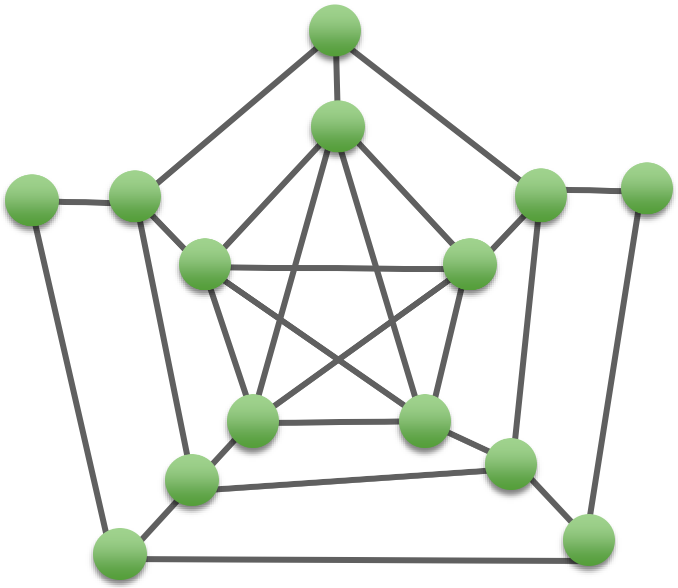
k=2



The sequential algorithm

- Given a threshold k, repeatedly remove all the vertices of degree less than k.
- The coreness value of a vertex is the largest k for which it is not removed.

k=2



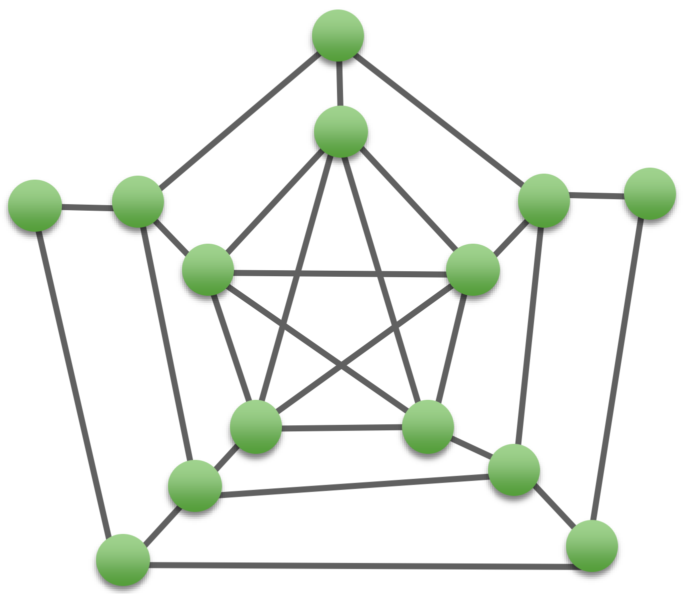
Coreness value of all remaining vertices >= 2.

The sequential algorithm

- Given a threshold k, repeatedly remove all the vertices of degree less than k.
- The coreness value of a vertex is the largest k for which it is not removed.

Implementing this approach directly can take too many rounds.

k=2

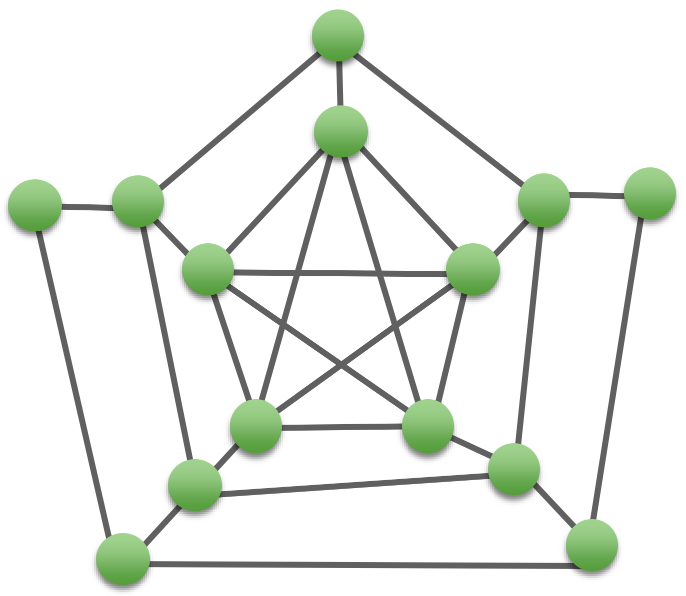


Coreness value of all remaining vertices >= 2.

The sequential algorithm

- Given a threshold k, repeatedly remove all the vertices of degree less than k.
- The coreness value of a vertex is the largest k for which it is not removed.

k=2

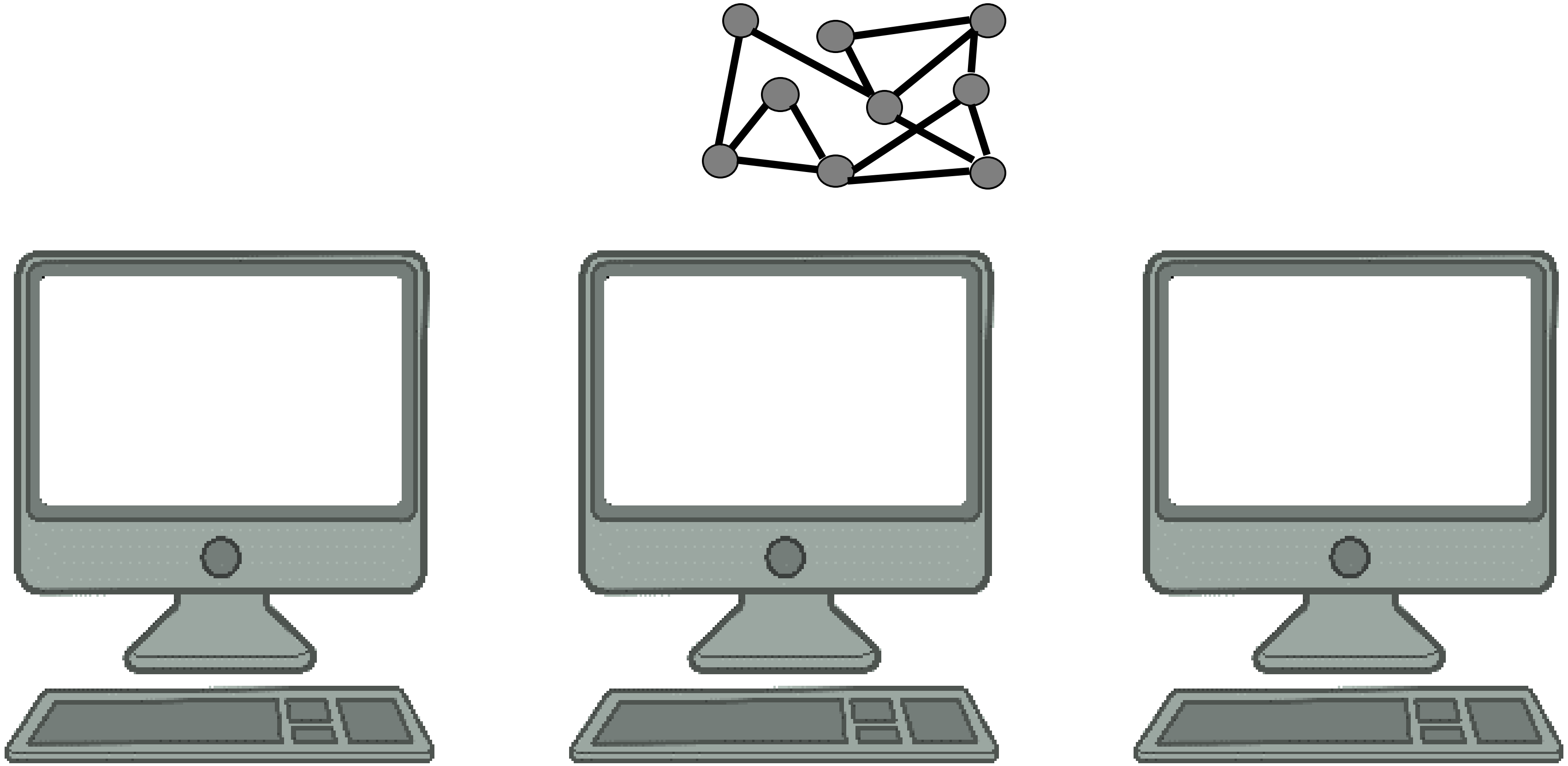


Coreness value of all remaining vertices >= 2.

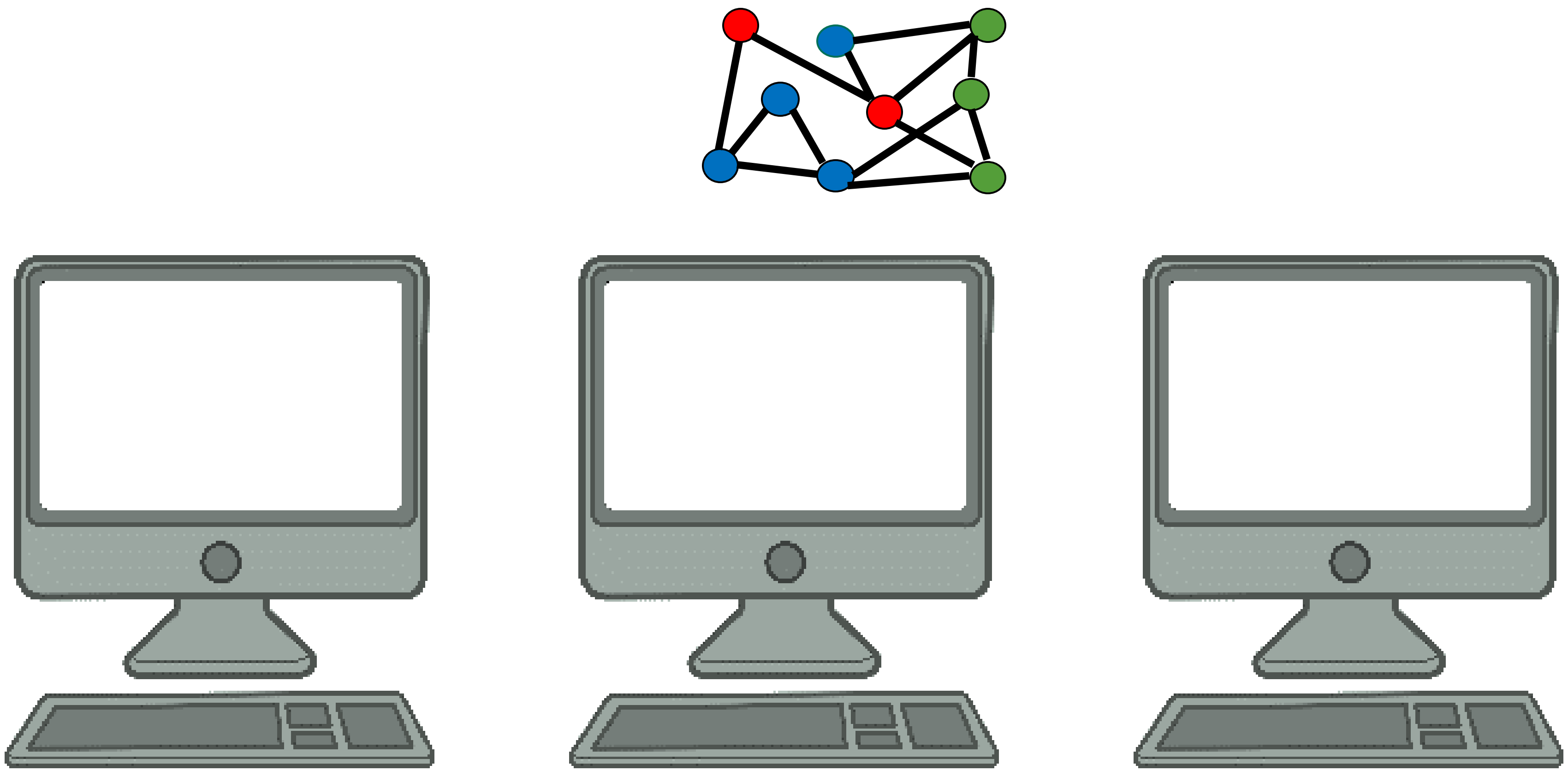
Implementing this approach directly can take too many rounds.

Idea:
Process only large thresholds.

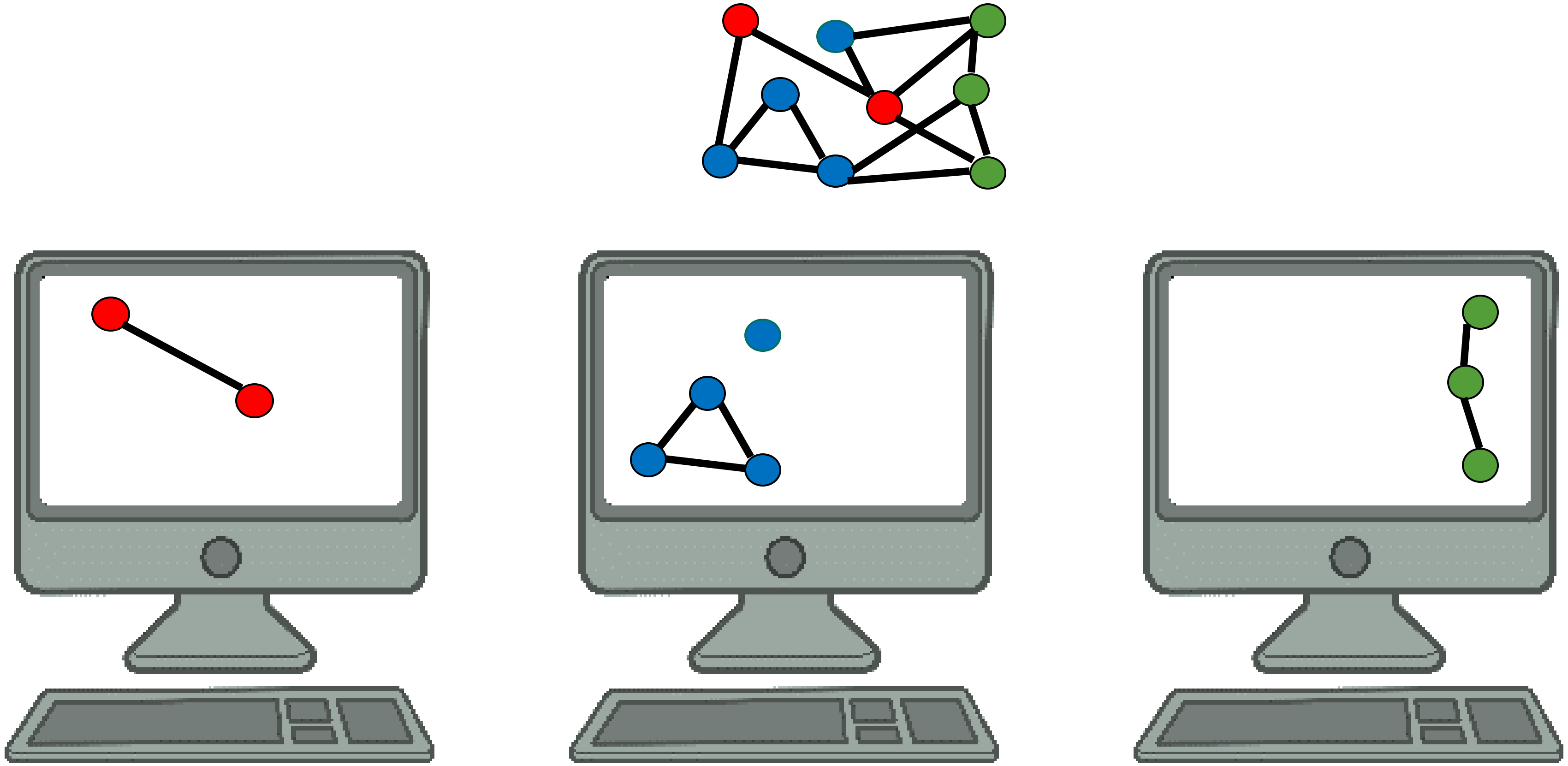
Partition vertices and process induced graphs



Partition vertices and process induced graphs



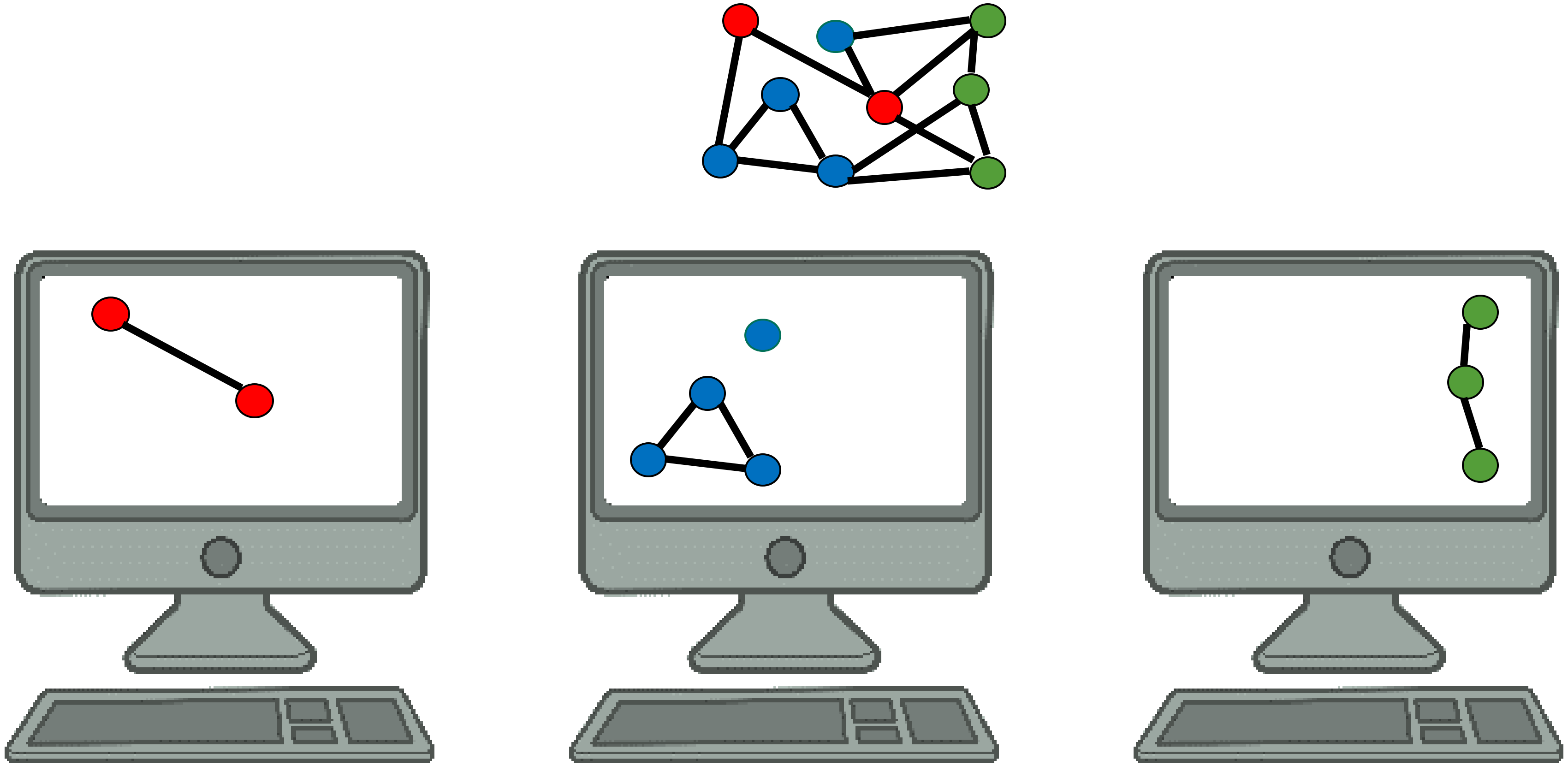
Partition vertices and process induced graphs



Apply the sequential algorithm locally.

Partition vertices and process induced graphs

Partition the graph across $\sqrt{n}$ machines.

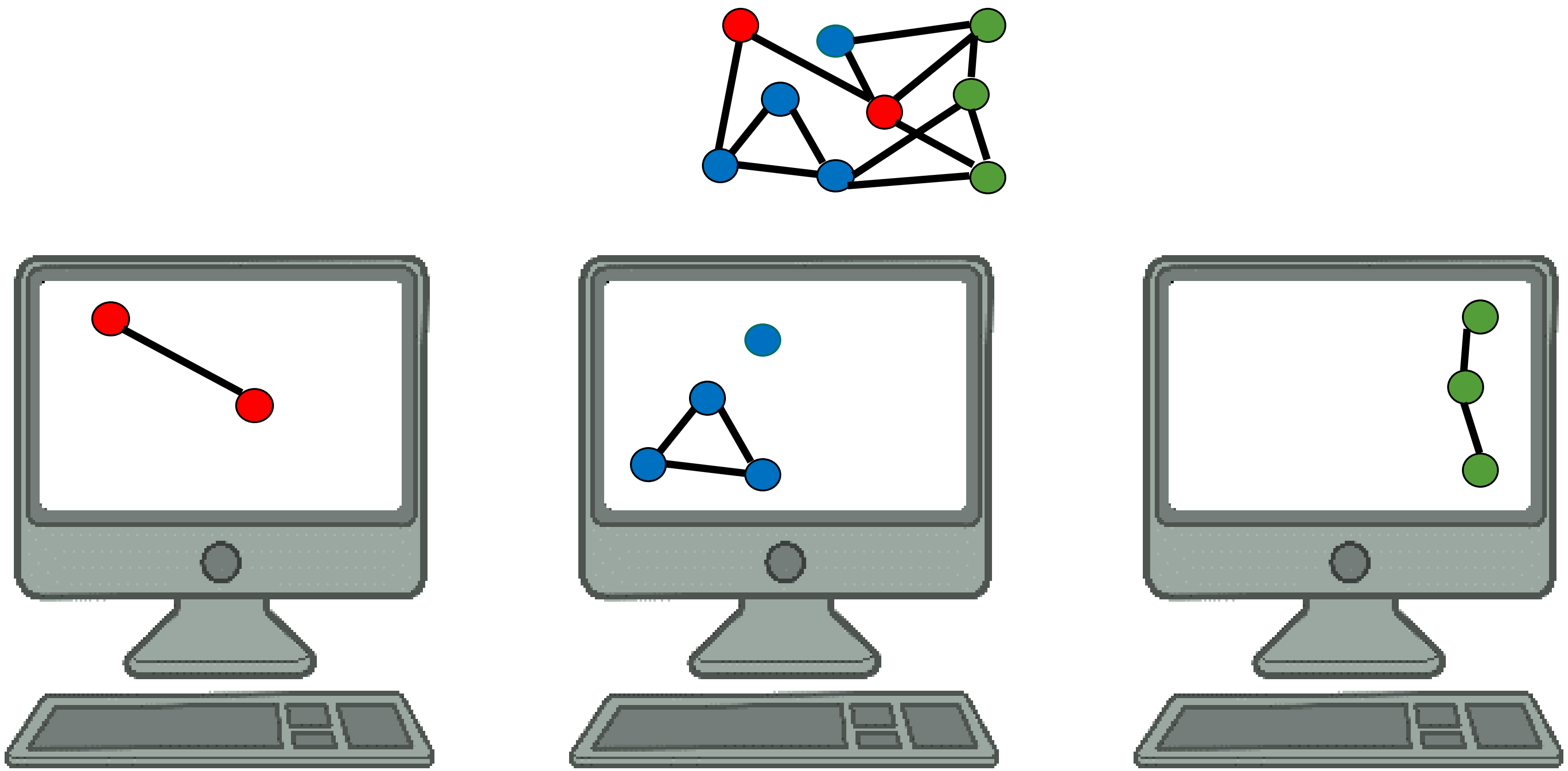


Apply the sequential algorithm locally.

Partition vertices and process induced graphs

Partition the graph across $\sqrt{n}$ machines.

The local degree of each vertex v with $d_v \geq \sqrt{n} \log n$ is sharply concentrated around its expectation. (Chernoff bound)



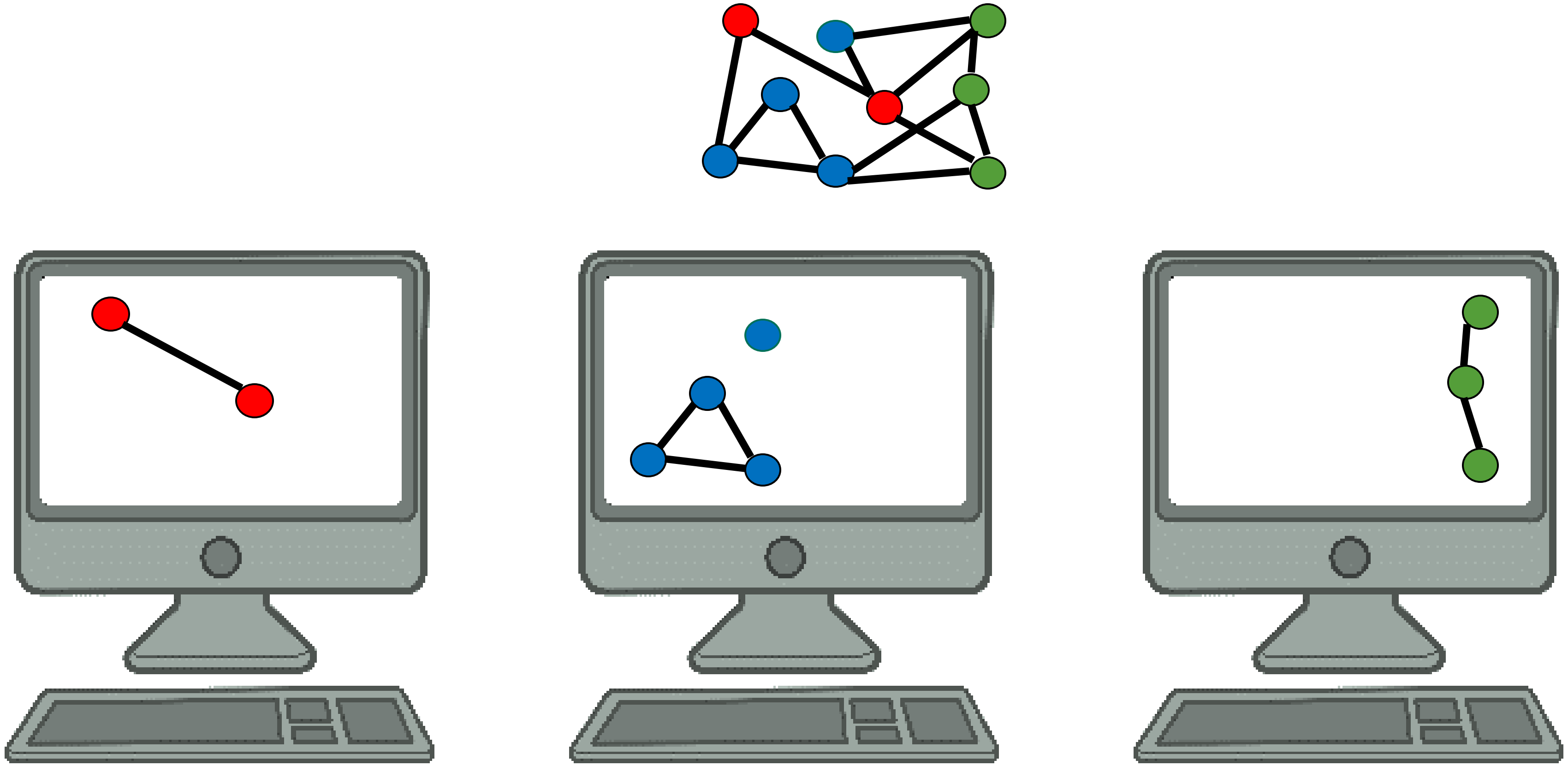
Apply the sequential algorithm locally.

Partition vertices and process induced graphs

Partition the graph across $\sqrt{n}$ machines.

The local degree of each vertex v with $d_v \geq \sqrt{n} \log n$ is sharply concentrated around its expectation. (Chernoff bound)

Run the sequential algorithm locally to find $(1 + \epsilon)$-approximate k-cores for $k \geq \sqrt{n} \log n$.



Apply the sequential algorithm locally.

Detecting all approximate k-cores

Partitioning across $\sqrt{n}$ machines detects the k-cores for $k \geq \sqrt{n} \log n$. How about $k < \sqrt{n} \log n$?

Detecting all approximate k-cores

Partitioning across $\sqrt{n}$ machines detects the k-cores for $k \geq \sqrt{n}\log n$. How about $k < \sqrt{n}\log n$?

Ignore all the edges between vertices of coreness $\geq \sqrt{n}\log n$.

Detecting all approximate k-cores

Partitioning across $\sqrt{n}$ machines detects the k-cores for $k \geq \sqrt{n}\log n$. How about $k < \sqrt{n}\log n$?

→

Ignore all the edges between vertices of coreness $\geq \sqrt{n}\log n$.

→

The number of remaining edges is $\tilde{O}(n\sqrt{n})$.

Detecting all approximate k-cores

Partitioning across $\sqrt{n}$ machines detects the k-cores for $k \geq \sqrt{n}\log n$. How about $k < \sqrt{n}\log n$?

Ignore all the edges between vertices of coreness $\geq \sqrt{n}\log n$.

The number of remaining edges is $\tilde{O}(n\sqrt{n})$.

Partition the vertices across $n^{\frac{1}{4}}$ machines.

Detecting all approximate k-cores

Partitioning across $\sqrt{n}$ machines detects the k-cores for $k \geq \sqrt{n}\log n$. How about $k < \sqrt{n}\log n$?

Ignore all the edges between vertices of coreness $\geq \sqrt{n}\log n$.

The number of remaining edges is $\tilde{O}(n\sqrt{n})$.

Partition the vertices across $n^{\frac{1}{4}}$ machines.

Detect k-cores for $\mathrm{k} \geq n^{\frac{1}{4}}\log n$.

Detecting all approximate k-cores

Partitioning across $\sqrt{n}$ machines detects the k-cores for $k \geq \sqrt{n} \log n$. How about $k < \sqrt{n} \log n$?

Ignore all the edges between vertices of coreness $\geq \sqrt{n} \log n$.

The number of remaining edges is $\tilde{O}(n\sqrt{n})$.

Partition the vertices across $n^{\frac{1}{4}}$ machines.

Detect k-cores for $\mathrm{k} \geq n^{\frac{1}{4}} \log n$.

Repeat.

Detecting all approximate k-cores

Partitioning across $\sqrt{n}$ machines detects the k-cores for $k \geq \sqrt{n}\log n$. How about $k < \sqrt{n}\log n$?

→ Ignore all the edges between vertices of coreness $\geq \sqrt{n}\log n$.

→ The number of remaining edges is $\tilde{O}(n\sqrt{n})$.

→ Partition the vertices across $n^{\frac{1}{4}}$ machines.

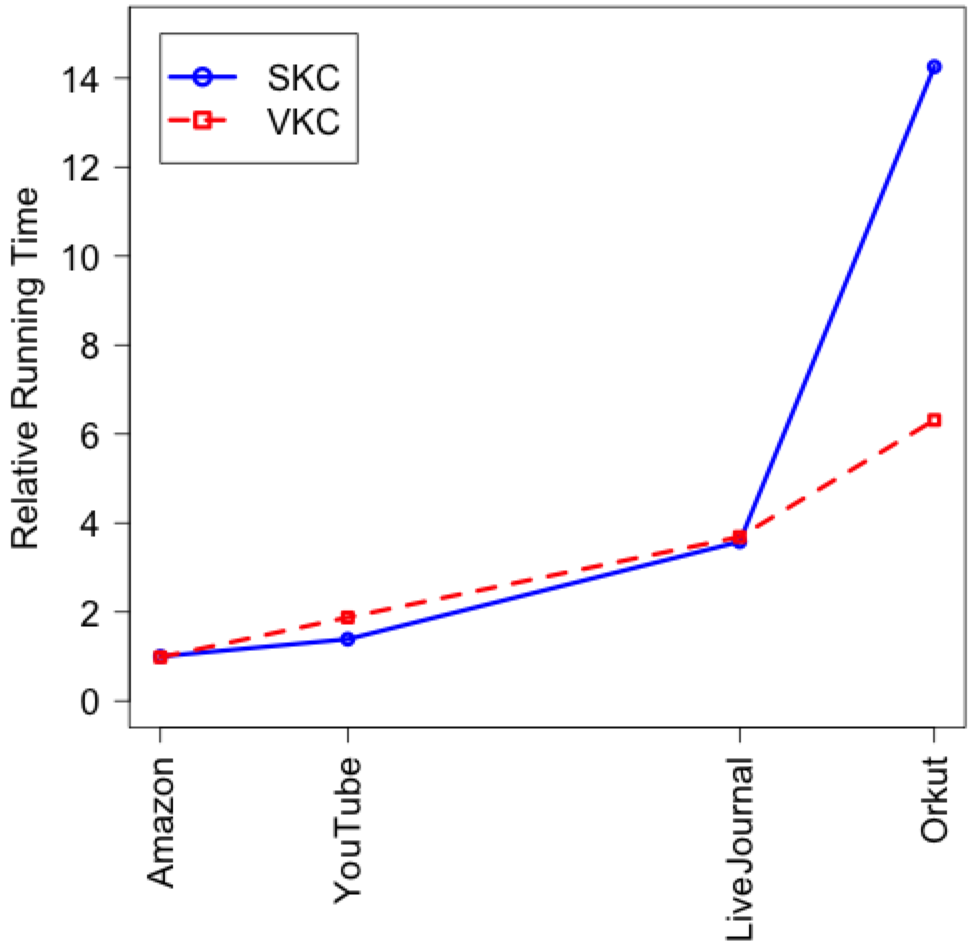
→ Detect k-cores for $k \geq n^{\frac{1}{4}}\log n$.

→ Repeat.

$n \rightarrow n^{1/2} \rightarrow n^{1/4} \rightarrow \ldots \rightarrow n^{1/\log n}$

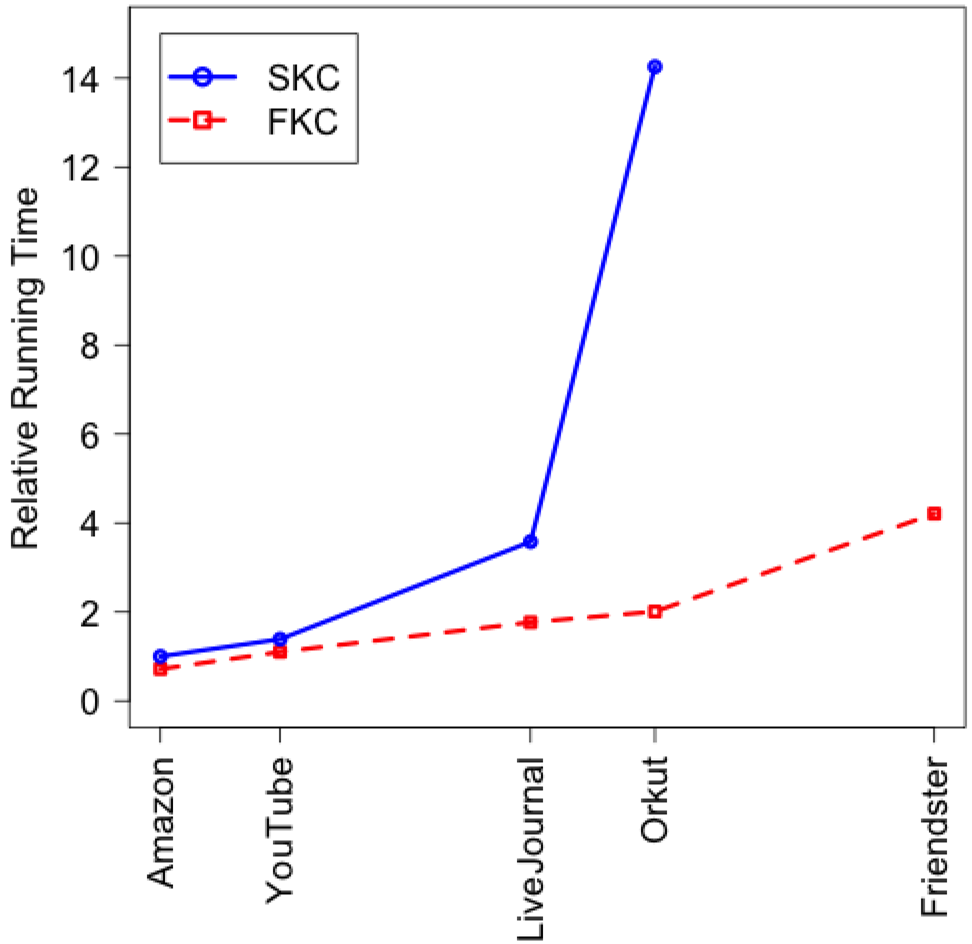
log log n rounds

Experiments



SKC = the algorithm in [Esfandiari et al. 2018]
VKC = Theorem 1

Experiments



SKC = the algorithm in [Esfandiari et al. 2018]
VKC = Theorem 2

Next

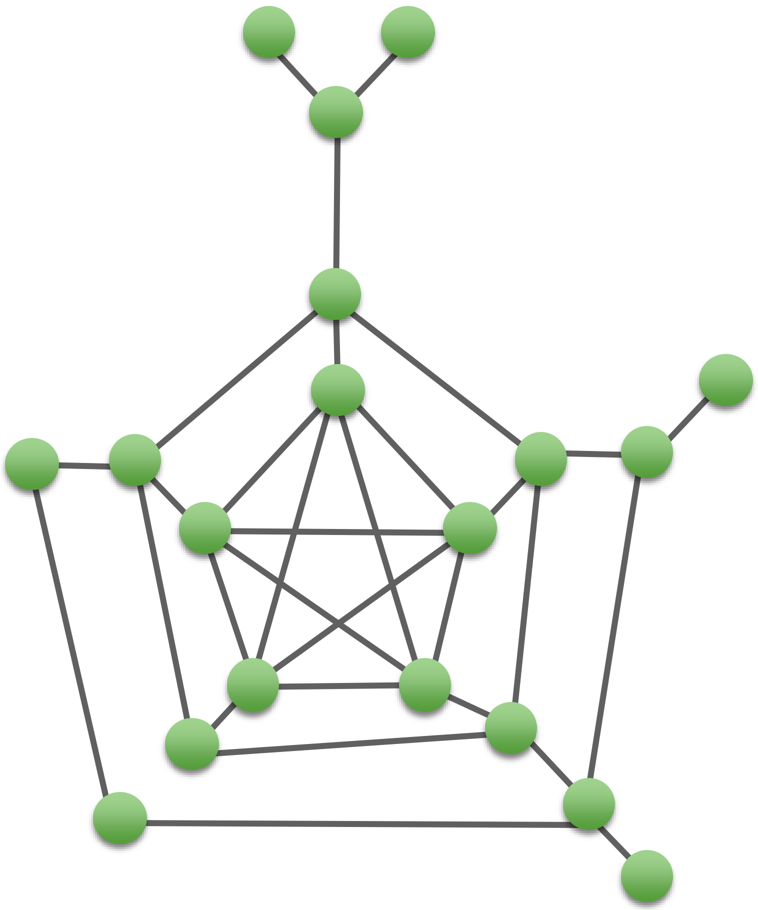
Theorem 2

$(2+\epsilon)$-approximate k-core decomposition can be obtained in $\tilde{O}\left(\sqrt{\log n}\right)$ MPC rounds with $O\left(n^{\delta}\right)$ memory per machine and the total memory of $\tilde{O}\left(\max\left\{n^{1+\delta}, m\right\}\right)$.

$(2+\epsilon)$-approximate algorithm in $\log n$ iterations

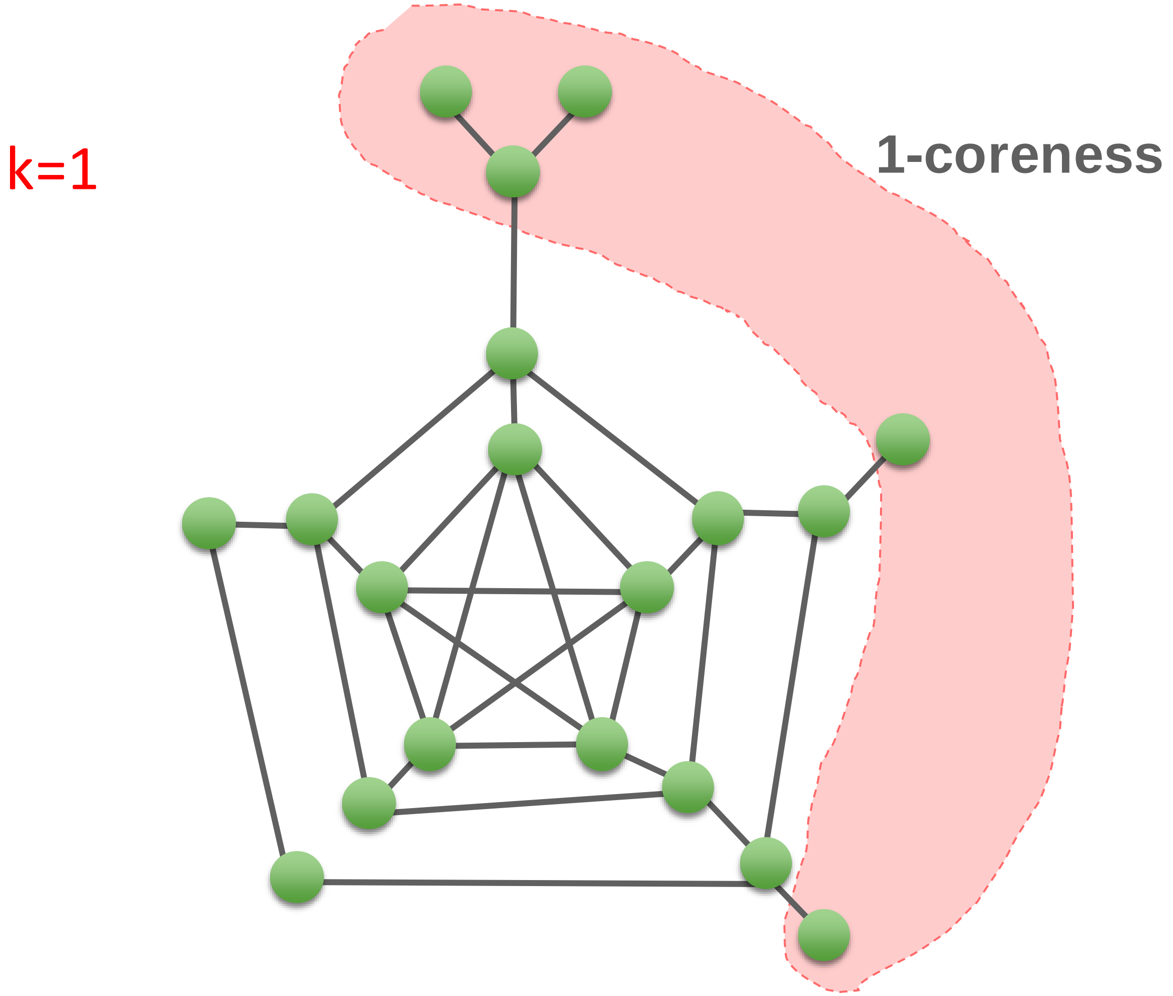
- Given a threshold k, repeatedly remove all the vertices of degree less than $(2+\epsilon)$k.
- The approximate coreness value of a vertex is the largest k for which it is not removed.

k=1



$(2+\epsilon)$-approximate algorithm in $\log n$ iterations

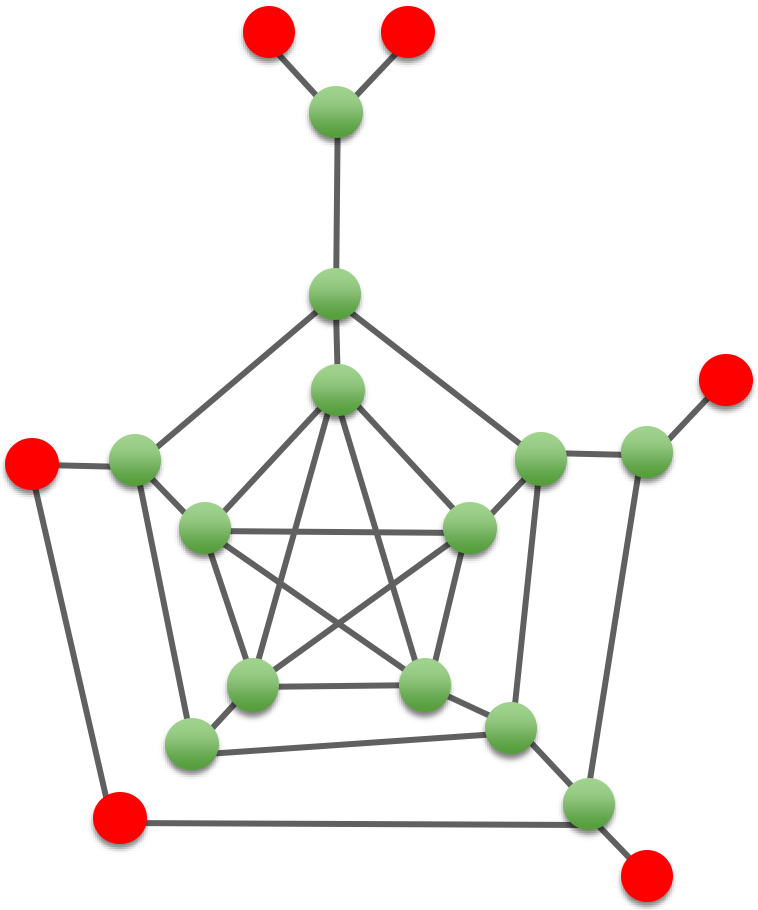
- Given a threshold k, repeatedly remove all the vertices of degree less than $(2+\epsilon)$k.
- The approximate coreness value of a vertex is the largest k for which it is not removed.



$(2+\epsilon)$-approximate algorithm in $\log n$ iterations

- Given a threshold k, repeatedly remove all the vertices of degree less than $(2+\epsilon)$k.
- The approximate coreness value of a vertex is the largest k for which it is not removed.

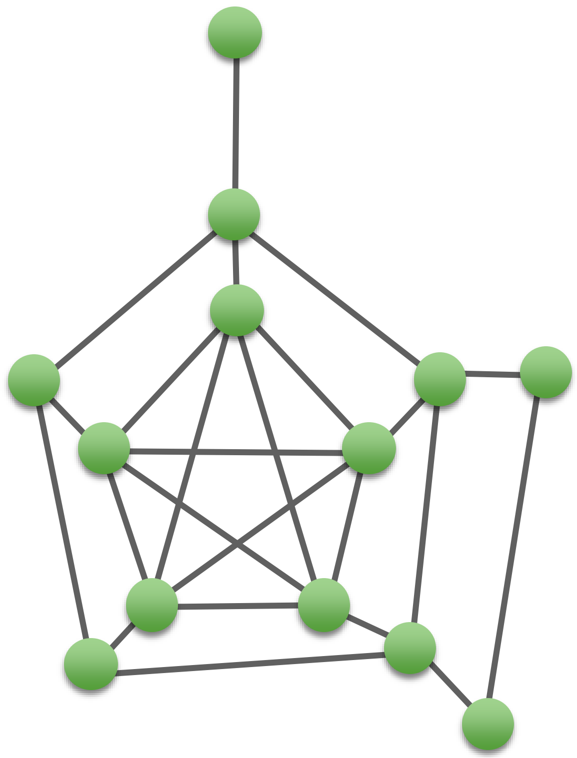
k=1



$(2+\epsilon)$-approximate algorithm in $\log n$ iterations

- Given a threshold k, repeatedly remove all the vertices of degree less than $(2+\epsilon)$k.
- The approximate coreness value of a vertex is the largest k for which it is not removed.

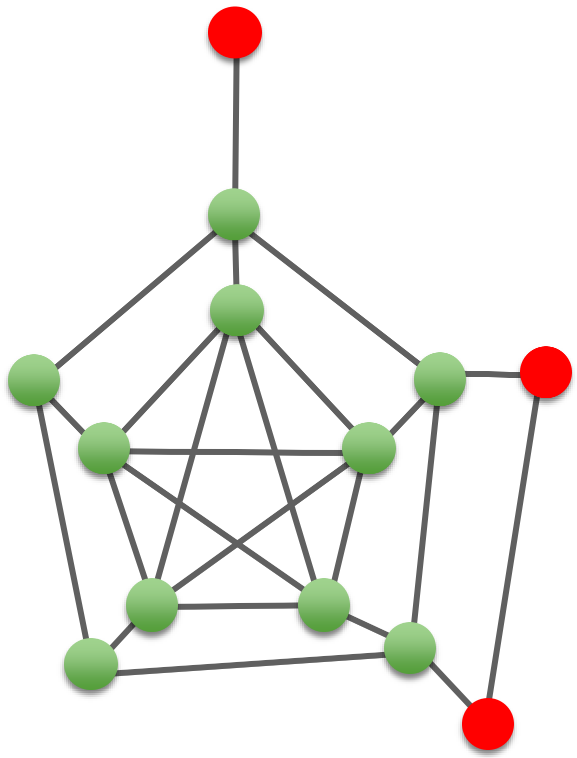
k=1



$(2+\epsilon)$-approximate algorithm in $\log n$ iterations

- Given a threshold k, repeatedly remove all the vertices of degree less than $(2+\epsilon)$k.
- The approximate coreness value of a vertex is the largest k for which it is not removed.

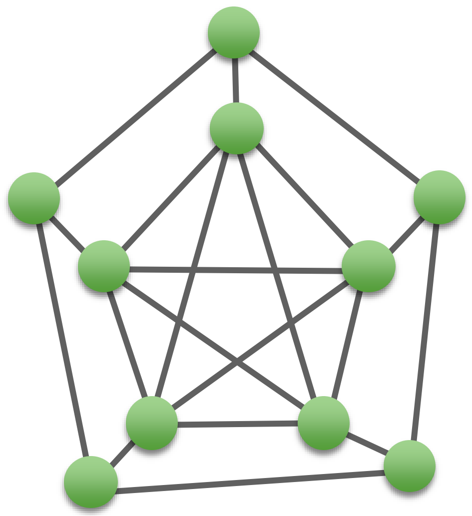
k=1



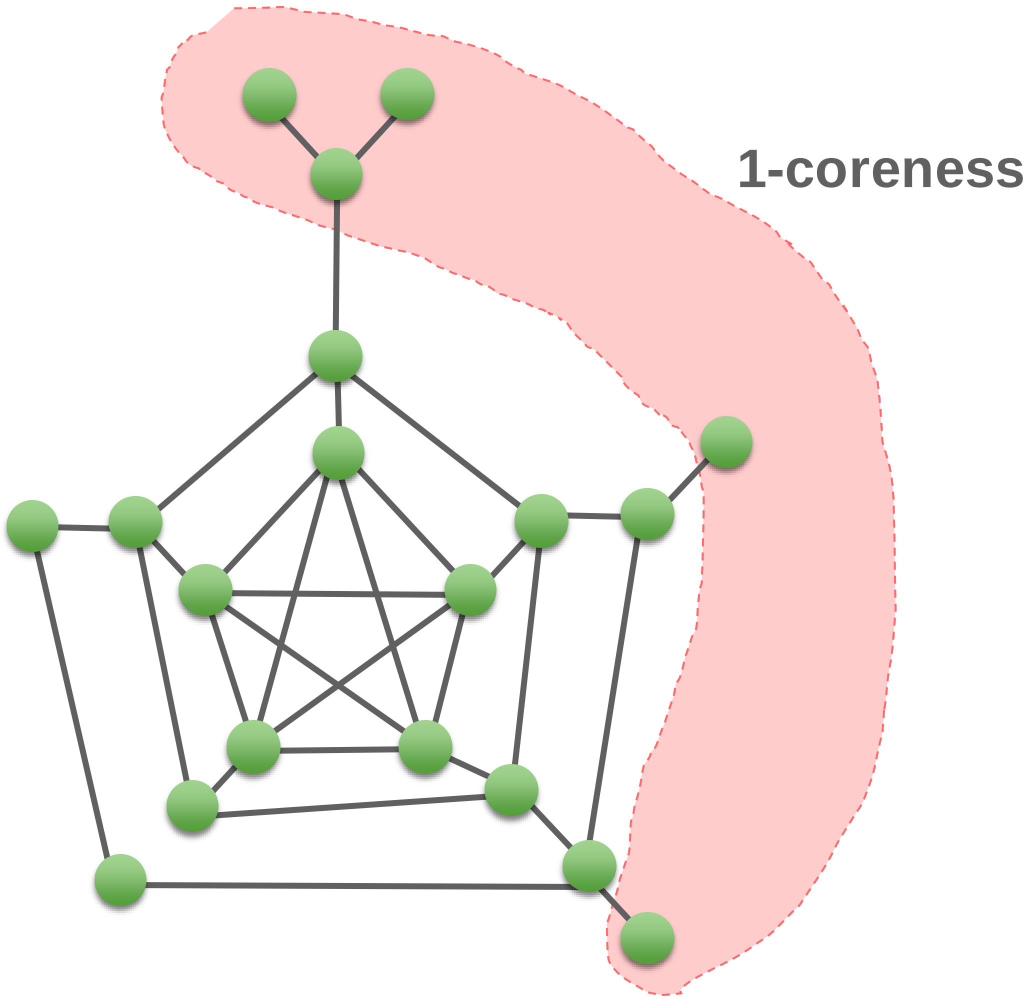
$(2+\epsilon)$-approximate algorithm in $\log n$ iterations

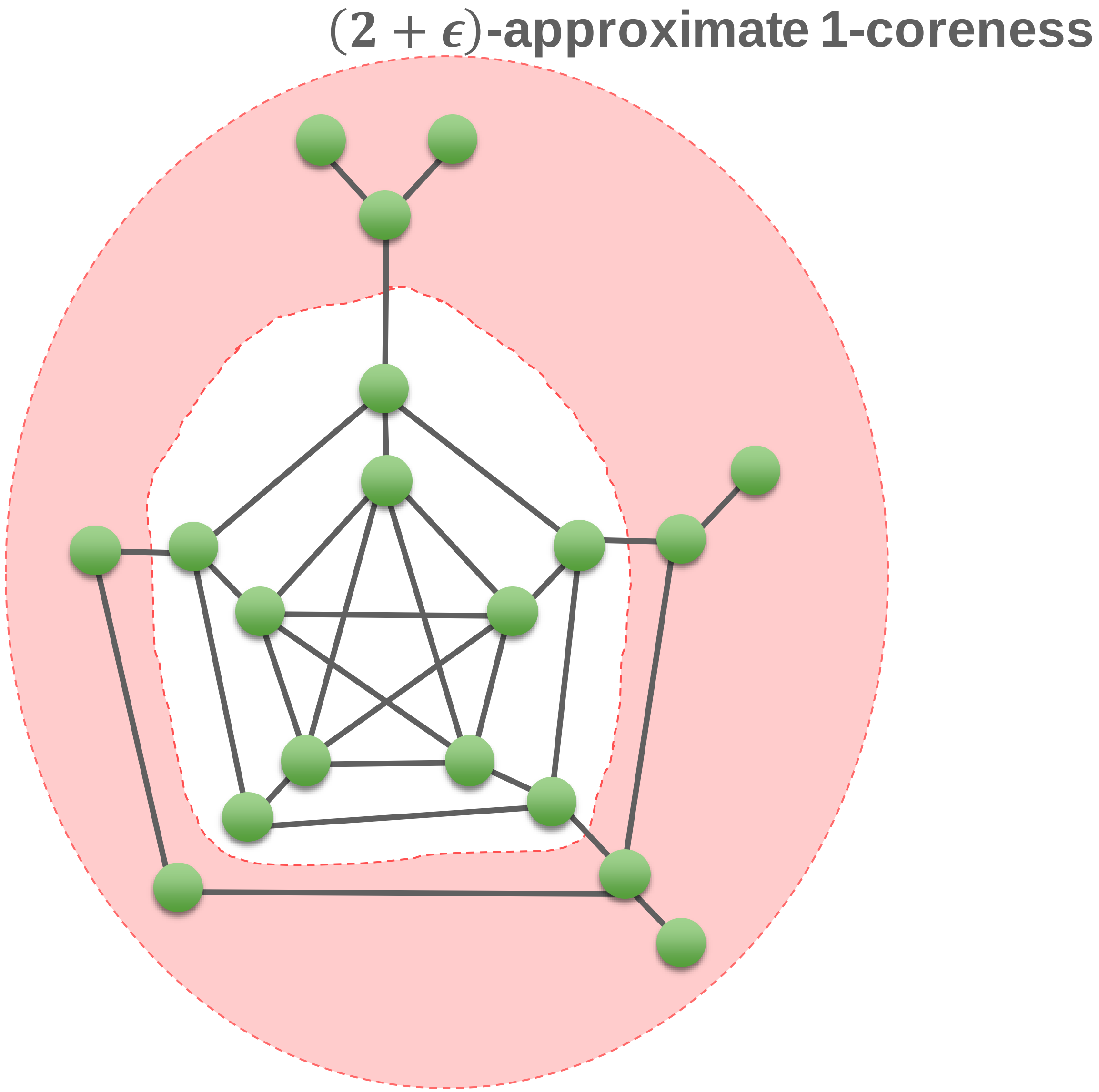
- Given a threshold k, repeatedly remove all the vertices of degree less than $(2+\epsilon)$k.
- The approximate coreness value of a vertex is the largest k for which it is not removed.

k=1



$(2+\epsilon)$-approximate algorithm in $\log n$ iterations

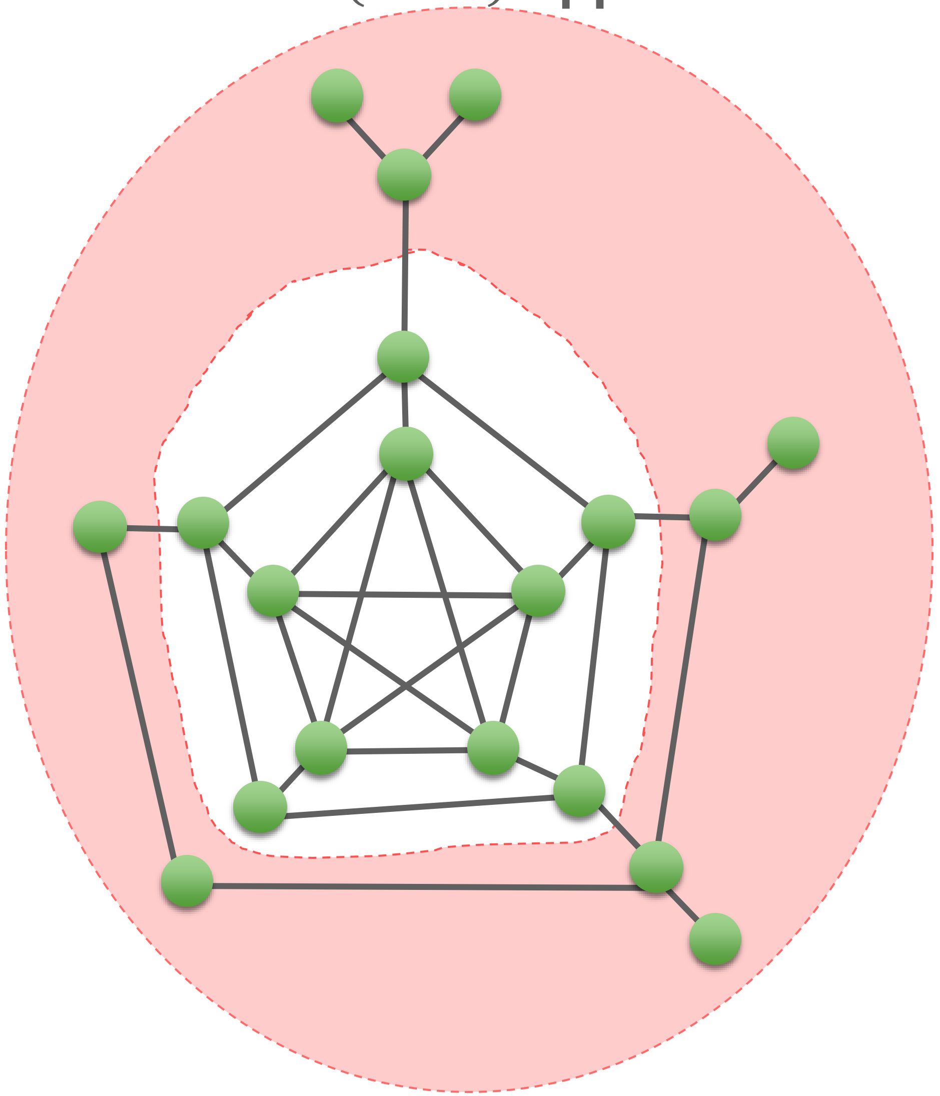




$(2+\epsilon)$-approximate algorithm in $\log n$ iterations

The algorithm terminates in $O(\log n)$ iterations!

High-level idea:
Simulate $O(\log n)$ sequential in $\tilde{O}\left(\sqrt{\log n}\right)$ MPC iterations.



Simulation of the $\log n$-iteration algorithm

Split the $\log n$ iterations into $\sqrt{\log n}$ phase, each phase consisting of $\sqrt{\log n}$ iterations.

Simulation of the $\log n$-iteration algorithm

Split the $\log n$ iterations into $\sqrt{\log n}$ phase, each phase consisting of $\sqrt{\log n}$ iterations.

Simulate each phase for each vertex by gathering its $\sqrt{\log n}$-hop neighborhood.

Simulation of the $\log n$-iteration algorithm

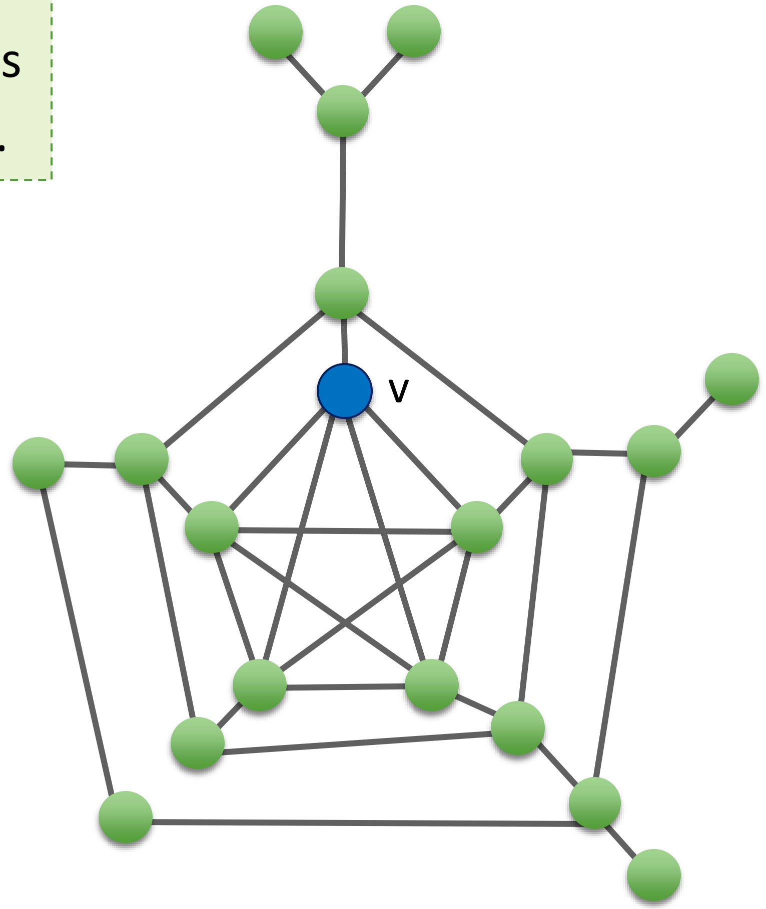
Split the $\log n$ iterations into $\sqrt{\log n}$ phase, each phase consisting of $\sqrt{\log n}$ iterations.

Simulate each phase for each vertex by gathering its $\sqrt{\log n}$-hop neighborhood.

Simulation of the $\log n$-iteration algorithm

Split the $\log n$ iterations into $\sqrt{\log n}$ phase, each phase consisting of $\sqrt{\log n}$ iterations.

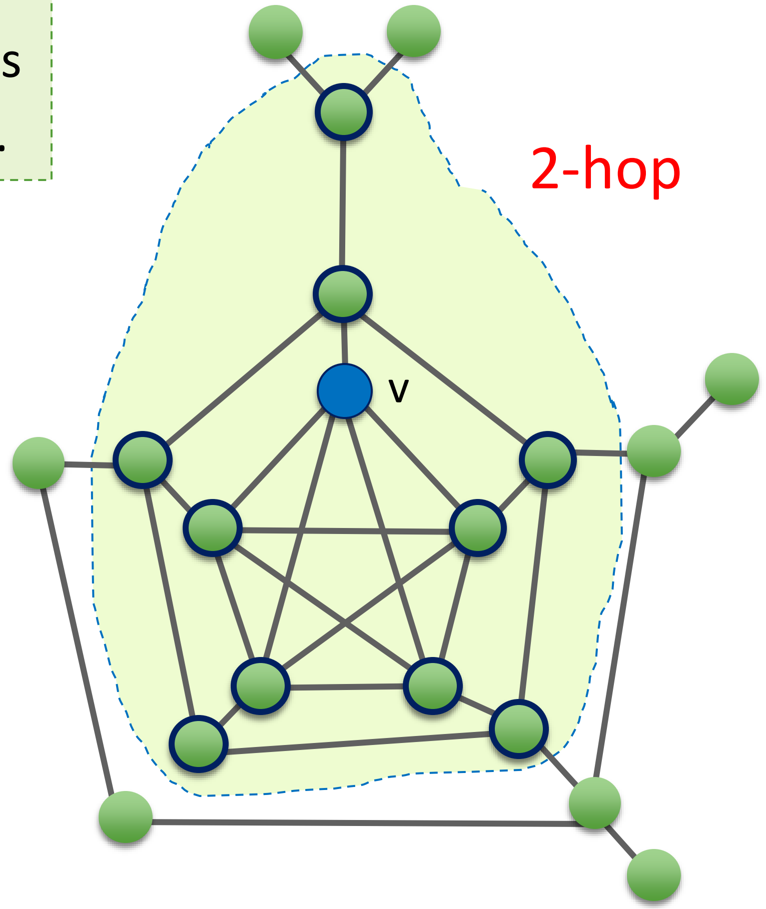
Simulate each phase for each vertex by gathering its $\sqrt{\log n}$-hop neighborhood.



Simulation of the $\log n$-iteration algorithm

Split the $\log n$ iterations into $\sqrt{\log n}$ phase, each phase consisting of $\sqrt{\log n}$ iterations.

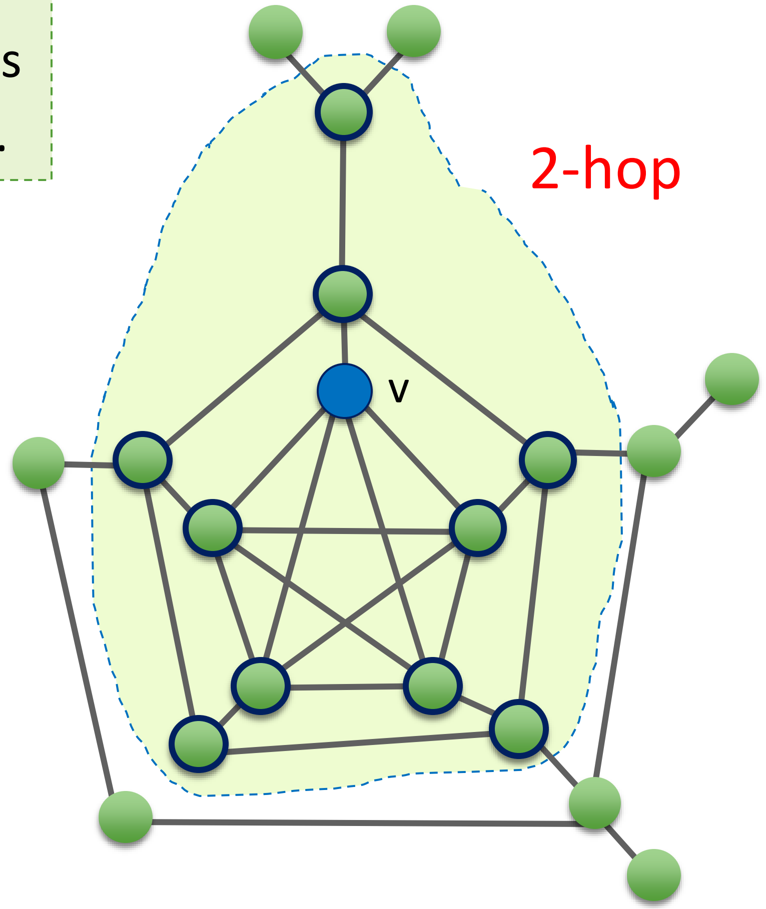
Simulate each phase for each vertex by gathering its $\sqrt{\log n}$-hop neighborhood.



Simulation of the $\log n$-iteration algorithm

Split the $\log n$ iterations into $\sqrt{\log n}$ phase, each phase consisting of $\sqrt{\log n}$ iterations.

Simulate each phase for each vertex by gathering its $\sqrt{\log n}$-hop neighborhood.

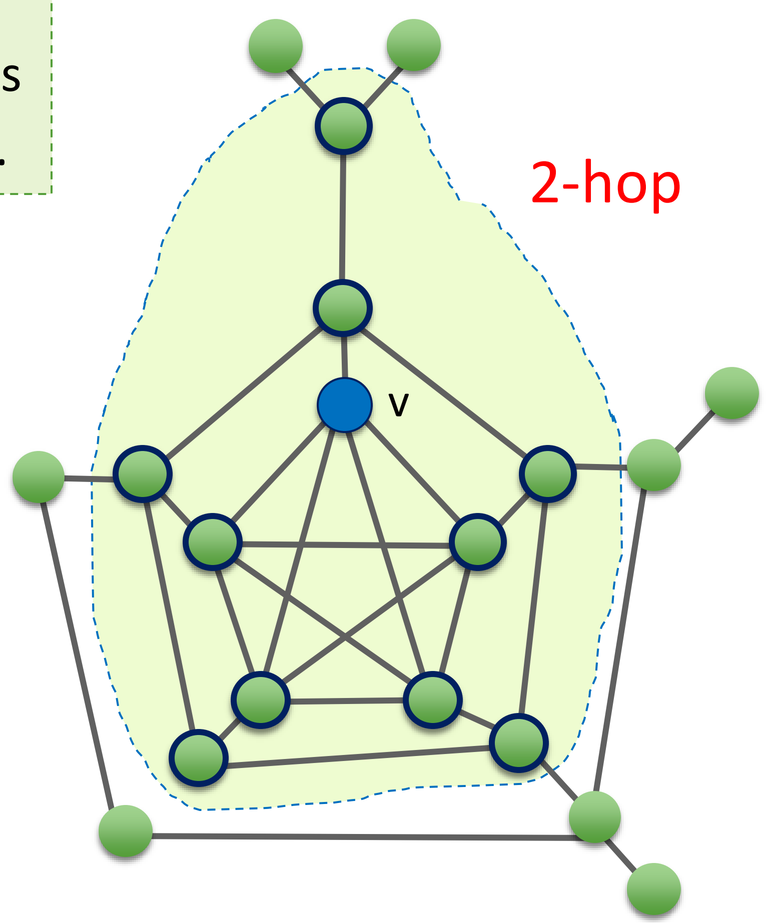


A $\sqrt{\log n}$-hop neighborhood might be too big! E.g., a vertex has degree n.

Simulation of the $\log n$-iteration algorithm

Split the $\log n$ iterations into $\sqrt{\log n}$ phase, each phase consisting of $\sqrt{\log n}$ iterations.

Simulate each phase for each vertex by gathering its $\sqrt{\log n}$-hop neighborhood.



A $\sqrt{\log n}$-hop neighborhood might be too big! E.g., a vertex has degree n.

Idea:
Sparsify the graph.

Sparsification

Given a parameter k, sparsify the graph by keeping each edge with probability $\Theta\left(\frac{\log n}{k}\right)$.

Sparsification

Given a parameter k, sparsify the graph by keeping each edge with probability $\Theta\left(\frac{\log n}{k}\right)$.

→

The approximate k-core is preserved after the sparsification. (Chernoff bound)

Sparsification

Given a parameter k, sparsify the graph by keeping each edge with probability $\Theta\left(\frac{\log n}{k}\right)$.

The approximate k-core is preserved after the sparsification. (Chernoff bound)

Some vertices still might have too large degree. E.g., vertex of degree n for $k=n^{0.1}$.

Sparsification

Given a parameter k, sparsify the graph by keeping each edge with probability $\Theta\left(\frac{\log n}{k}\right)$.

The approximate k-core is preserved after the sparsification. (Chernoff bound)

Some vertices still might have too large degree. E.g., vertex of degree n for $k=n^{0.1}$.

"Freeze" all the vertices of degree more than $2^{\sqrt{\delta \log n}}$ after the sparsification.

Sparsification

Given a parameter k, sparsify the graph by keeping each edge with probability $\Theta\left(\frac{\log n}{k}\right)$.

The approximate k-core is preserved after the sparsification. (Chernoff bound)

Some vertices still might have too large degree. E.g., vertex of degree n for $k=n^{0.1}$.

"Freeze" all the vertices of degree more than $2^{\sqrt{\delta \log n}}$ after the sparsification.

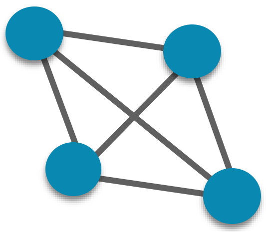
The number of frozen vertices is small and affects the round complexity only by a constant.

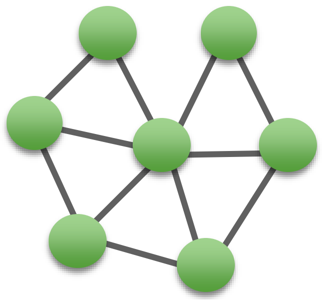
Densest subgraph vs. k-core decomposition

Is the non-empty k-core for the largest k the same as the densest subgraph

Densest subgraph vs. k-core decomposition

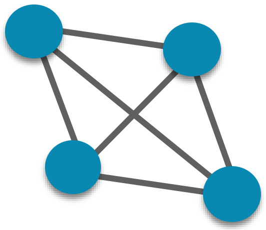
Is the non-empty k-core for the largest k the same as the densest subgraph



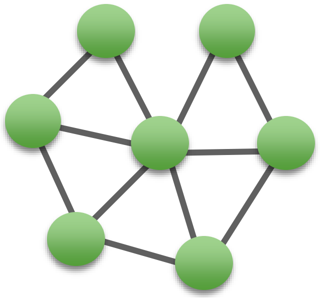


Densest subgraph vs. k-core decomposition

Is the non-empty k-core for the largest k the same as the densest subgraph



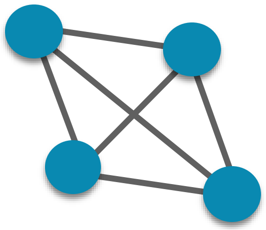
3-core



2-core

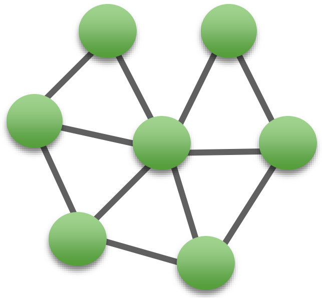
Densest subgraph vs. k-core decomposition

Is the non-empty k-core for the largest k the same as the densest subgraph ?



3-core

density = 3/2



2-core

density = 11/7