Large Scale Algorithms, Clustering and the MPC model

Silvio Lattanzi
Google Zurich

ADGA 2019, Budapest

Outline

- Models, MapReduce and Simple Examples
- Capacitated Metric Clustering at Scale
  How can we cluster the world map efficiently?
- Hierarchical Graph Clustering at Scale
  Can we obtain a hierarchical clustering efficiently?

Incredible amount of online data

3,557,052,467
Google searches today

3,331,015
Blog posts written today

445,065,893
Tweets sent today

4,055,053,771
Videos viewed today on YouTube

45,866,158
Photos uploaded today on Instagram

73,178,241
Tumblr posts today

1,952,375,064
Facebook active users

532,058,388
Google+ active users

308,862,767
Twitter active users

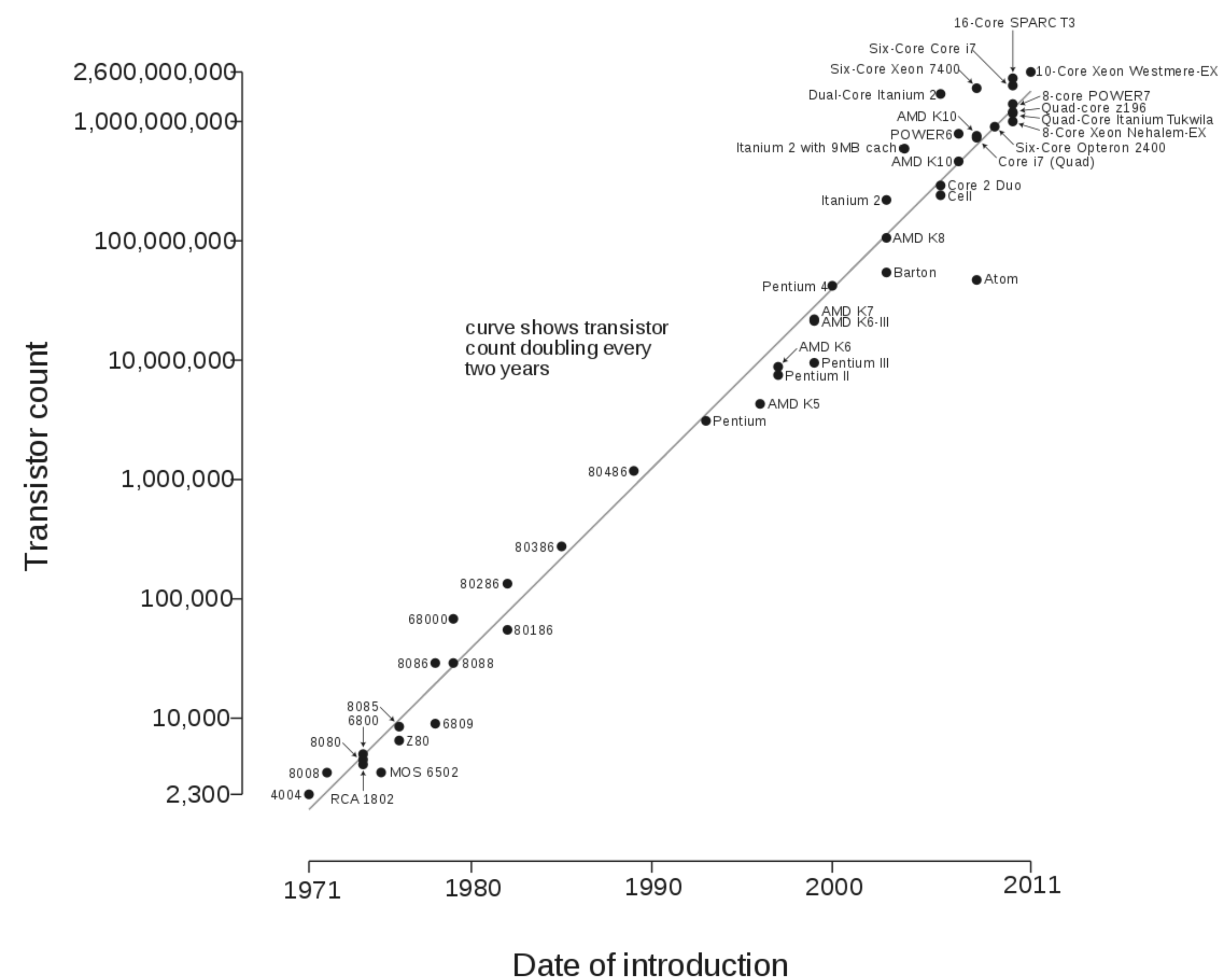
Stats from http://www.internetlivestats.com/.

Moore’s Law

Moore’s Law

Number of transistors double roughly every two years

Microprocessor Transistor Counts 1971-2011 & Moore's Law



Hard Drive evolution

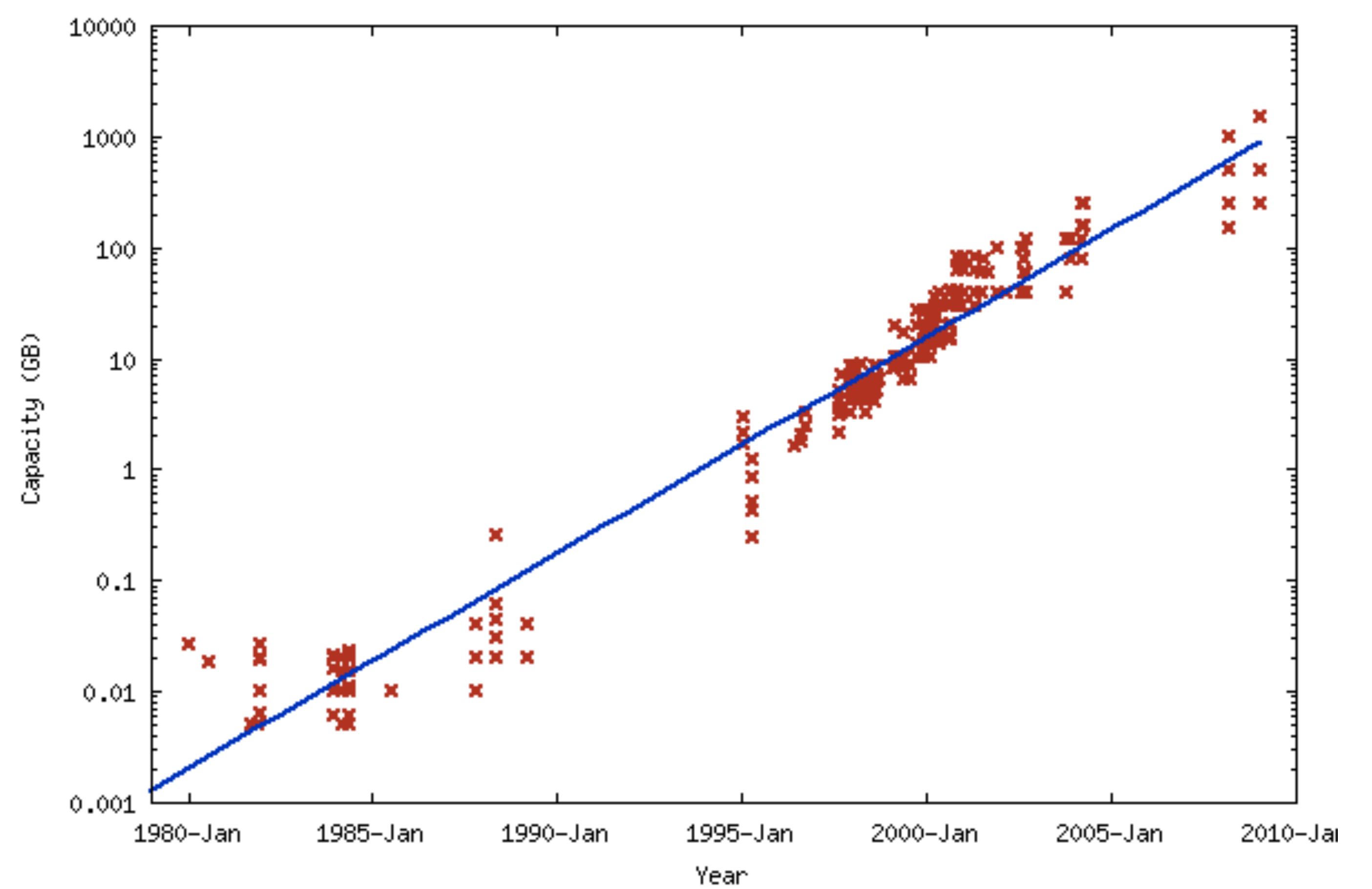
Hard Drive evolution

80s : 10M $\rightarrow$ 100M

90s : 100M $\rightarrow$ 10G

00s : 10G $\rightarrow$ 1T

10s : 1T $\rightarrow$ 100T

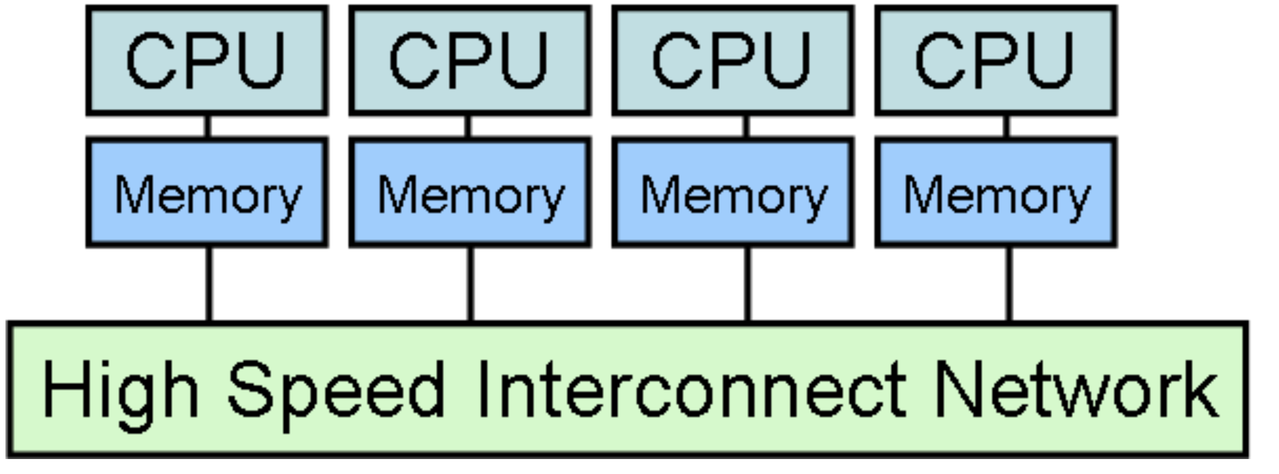


Data >> Hard drive

Models, MapReduce and Simple Examples

Classic Parallel programming

Computers coordinate autonomously



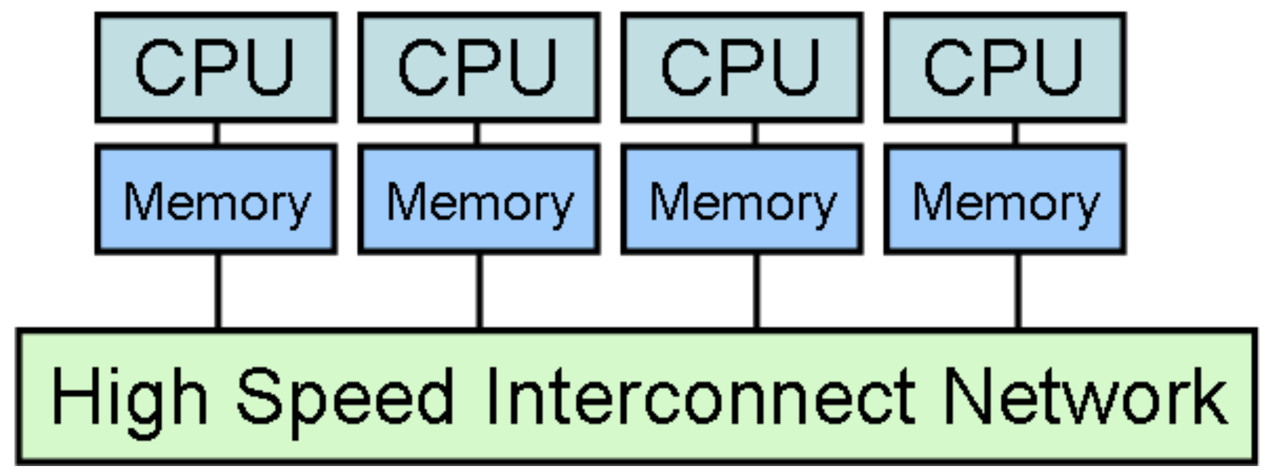
Classic Parallel programming

Computers coordinate autonomously

Hard to read and understand

Hard to write

Hard to debug



A simple model for parallel computing

Main properties:

1. Synchronous vs. Asynchronous

2. Partition Data:
   Adversarial or Random

3. Communication
   A. Topology (complete or not)
   B. Amount (bounded or not)

4. Size of machines

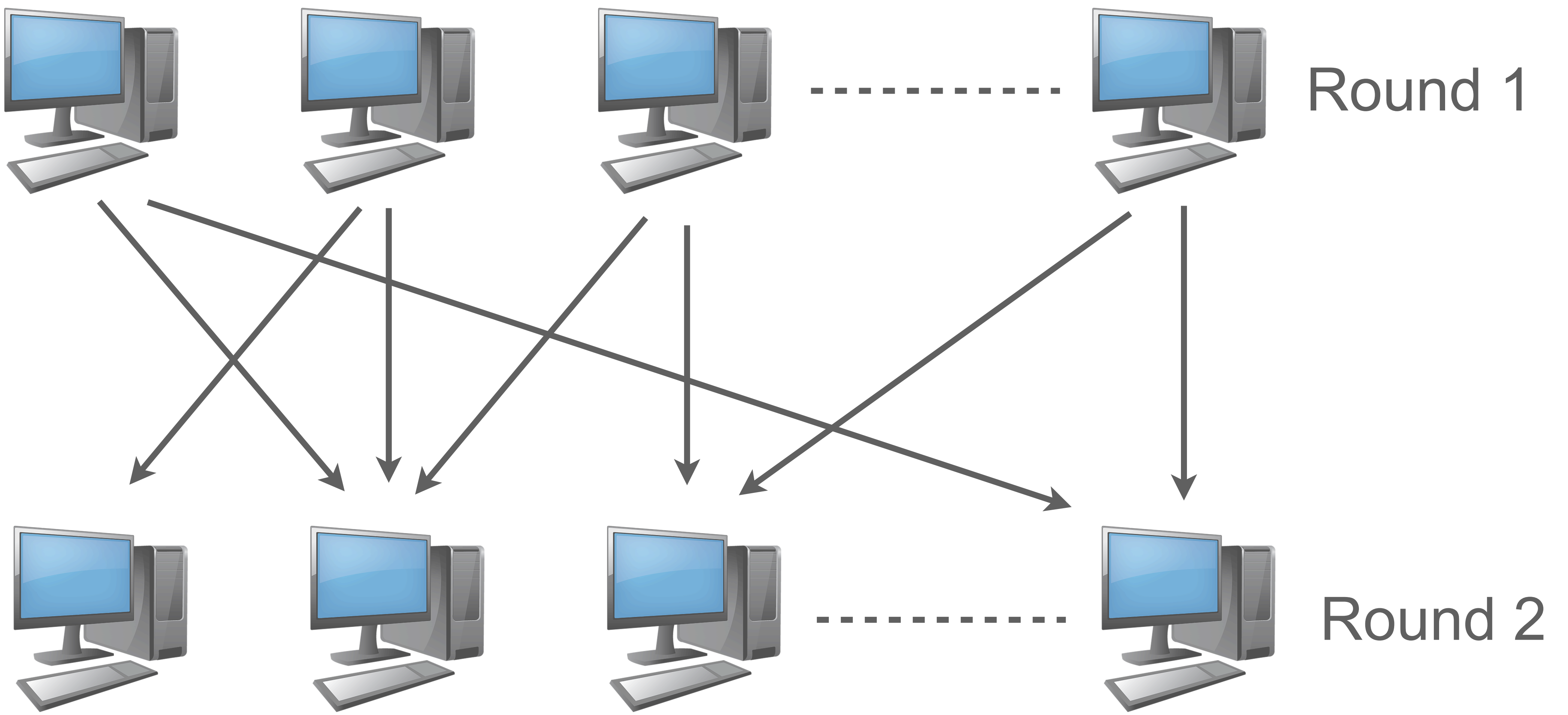
5. Fault-tolerance

A simple model for parallel computing

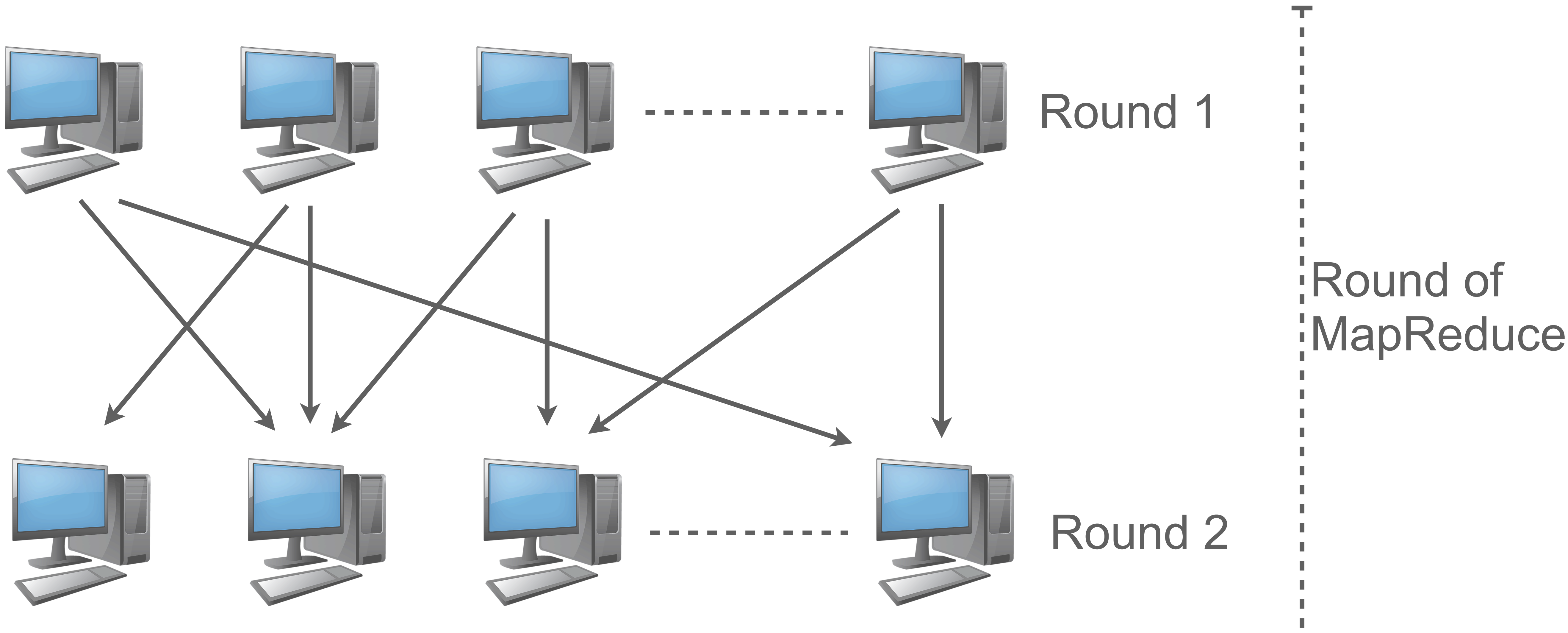
Main properties:

1. **Synchronous** vs. Asynchronous
2. Partition Data:
   Adversarial or Random
3. Communication
   A. Topology (**complete** or not)
   B. Amount (**bounded** or not)
4. Size of machines ~ **largish**
5. Fault-tolerance: **transparent to user**

MapReduce

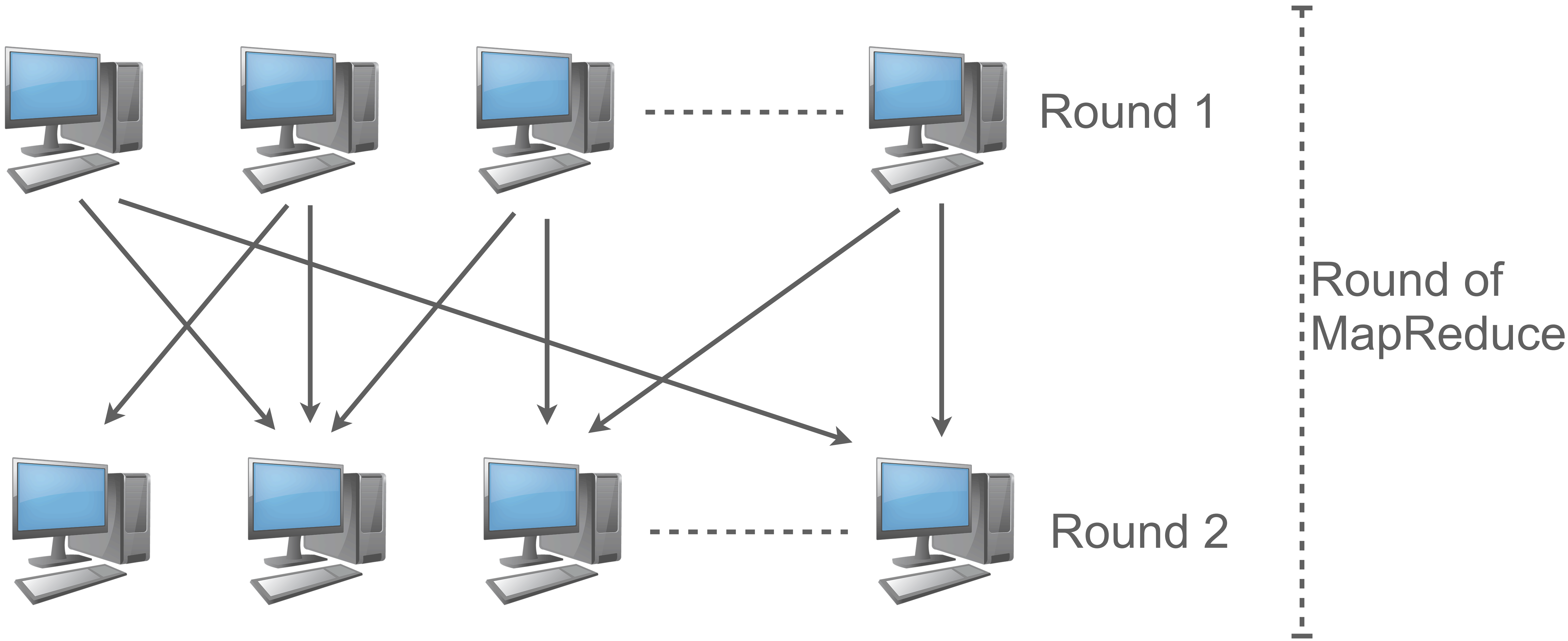


MapReduce

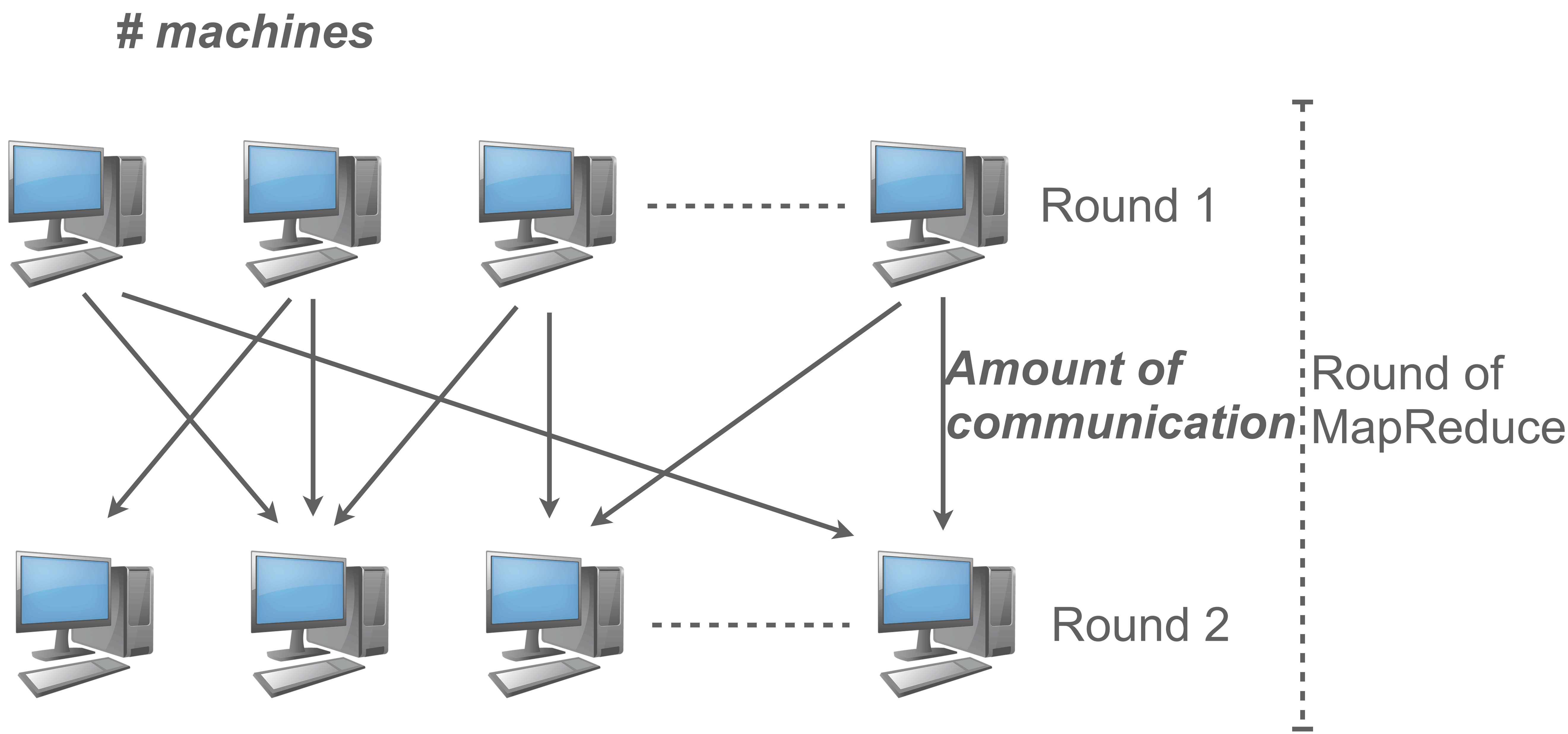


MapReduce

 # machines

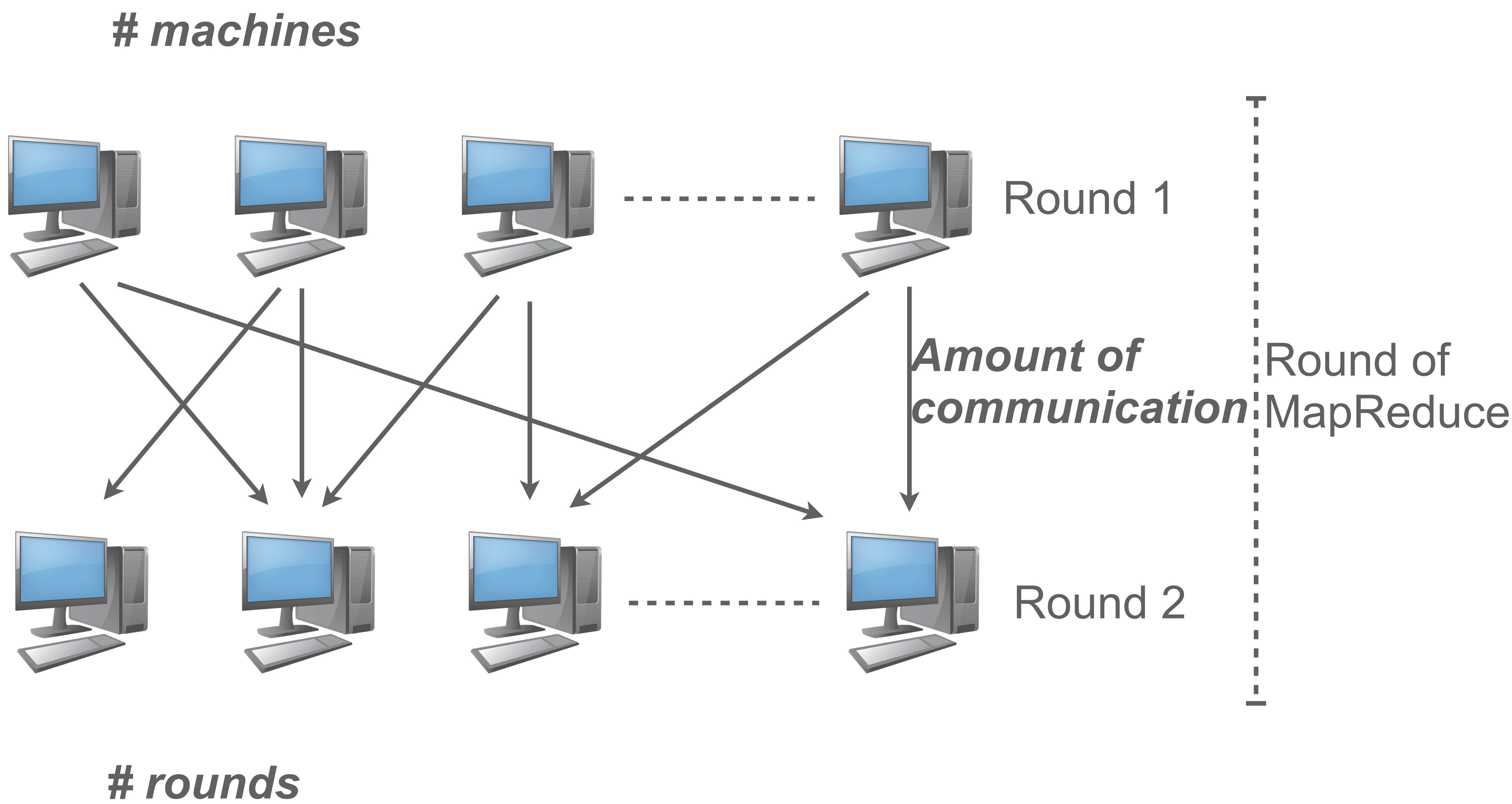


MapReduce



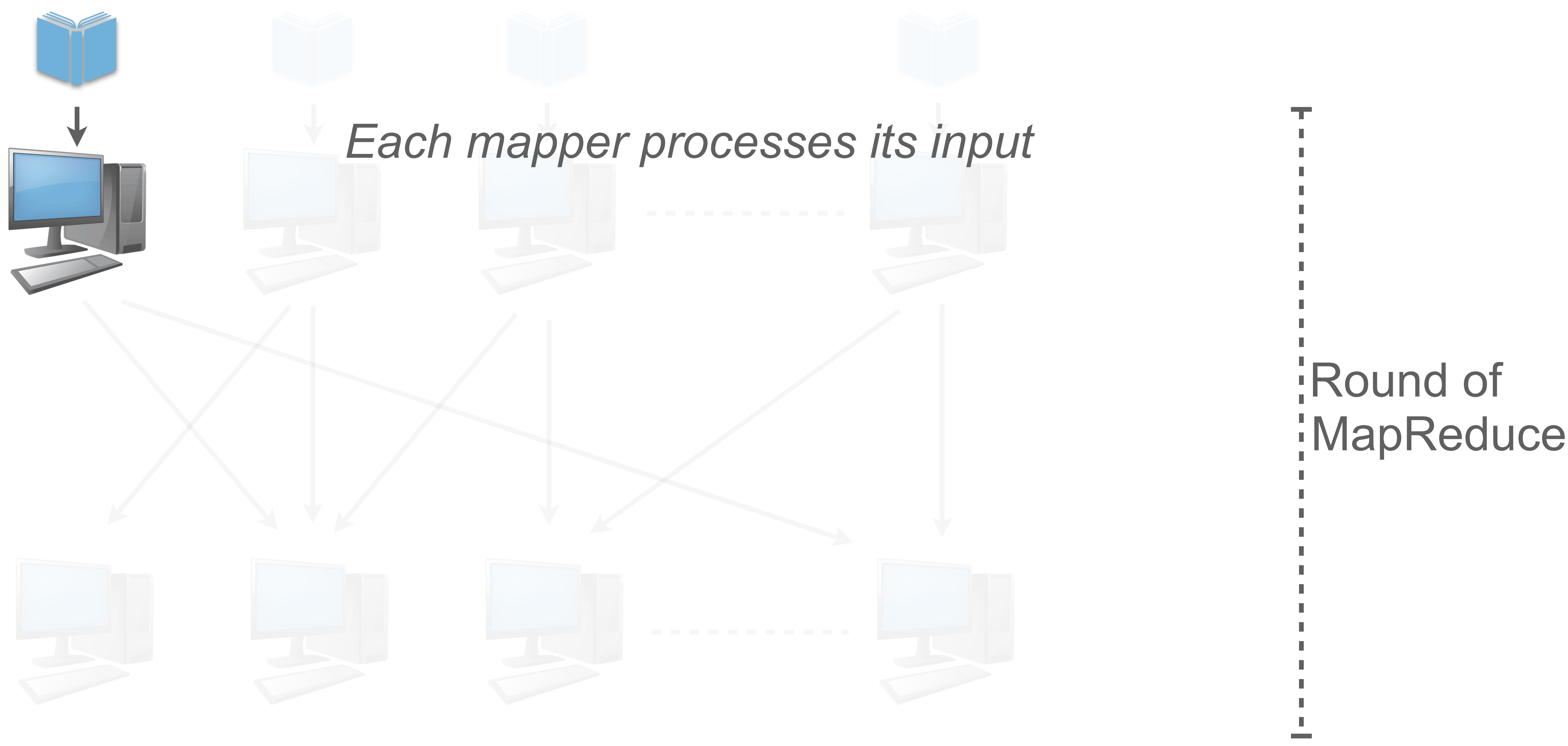
machines
Round 1
Amount of communication
Round of MapReduce
Round 2

MapReduce

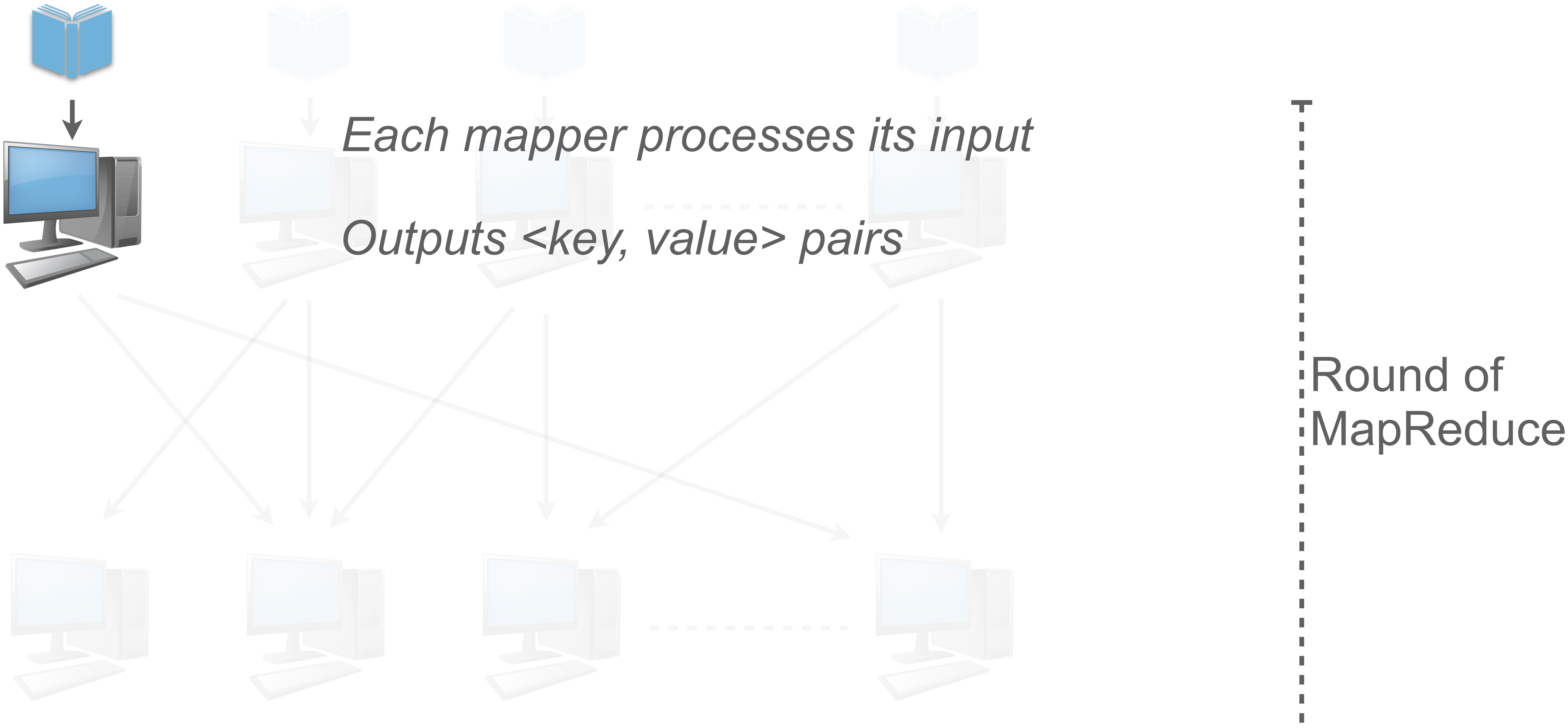


machines
Round 1
Amount of
communication
Round of
MapReduce
Round 2
rounds

MapReduce

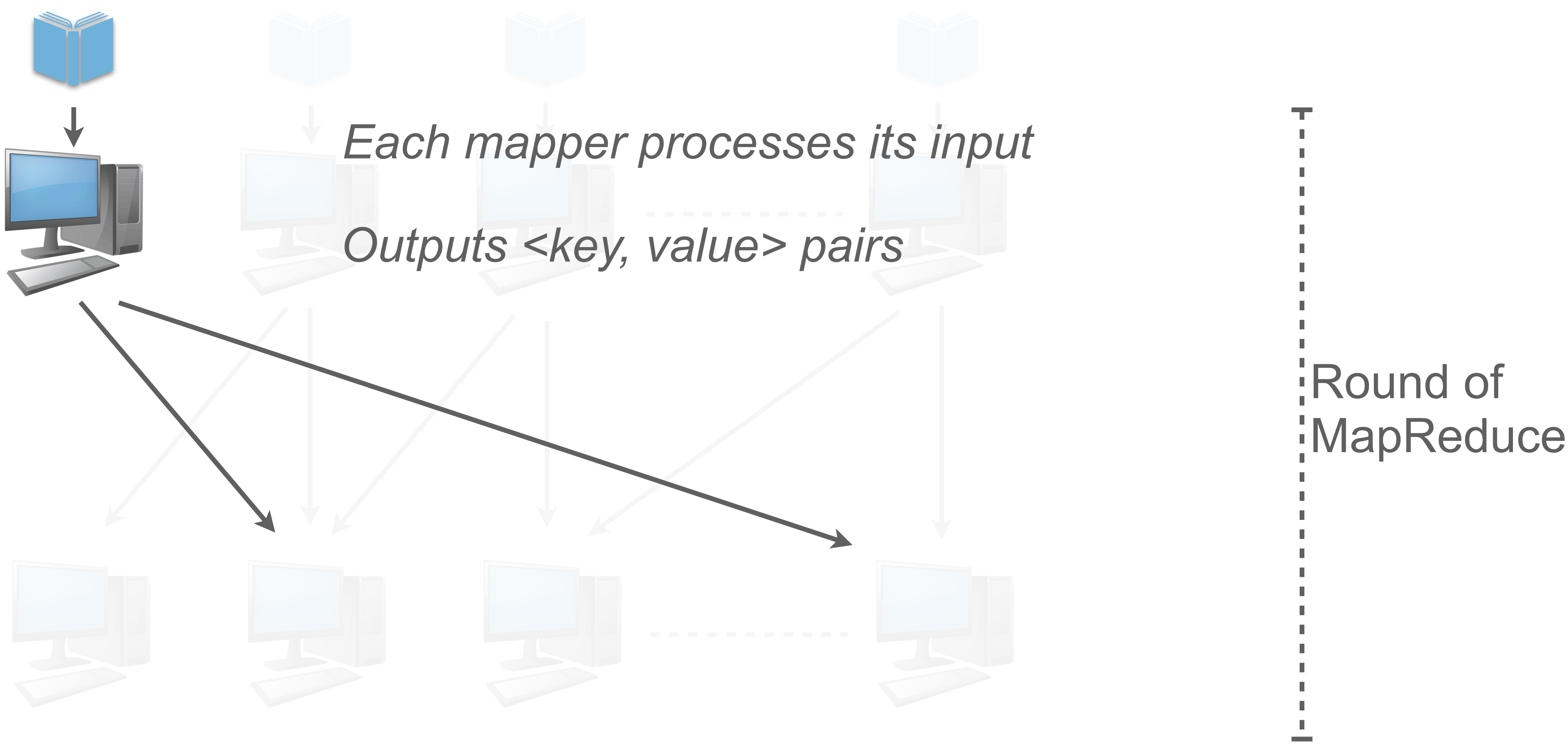


MapReduce



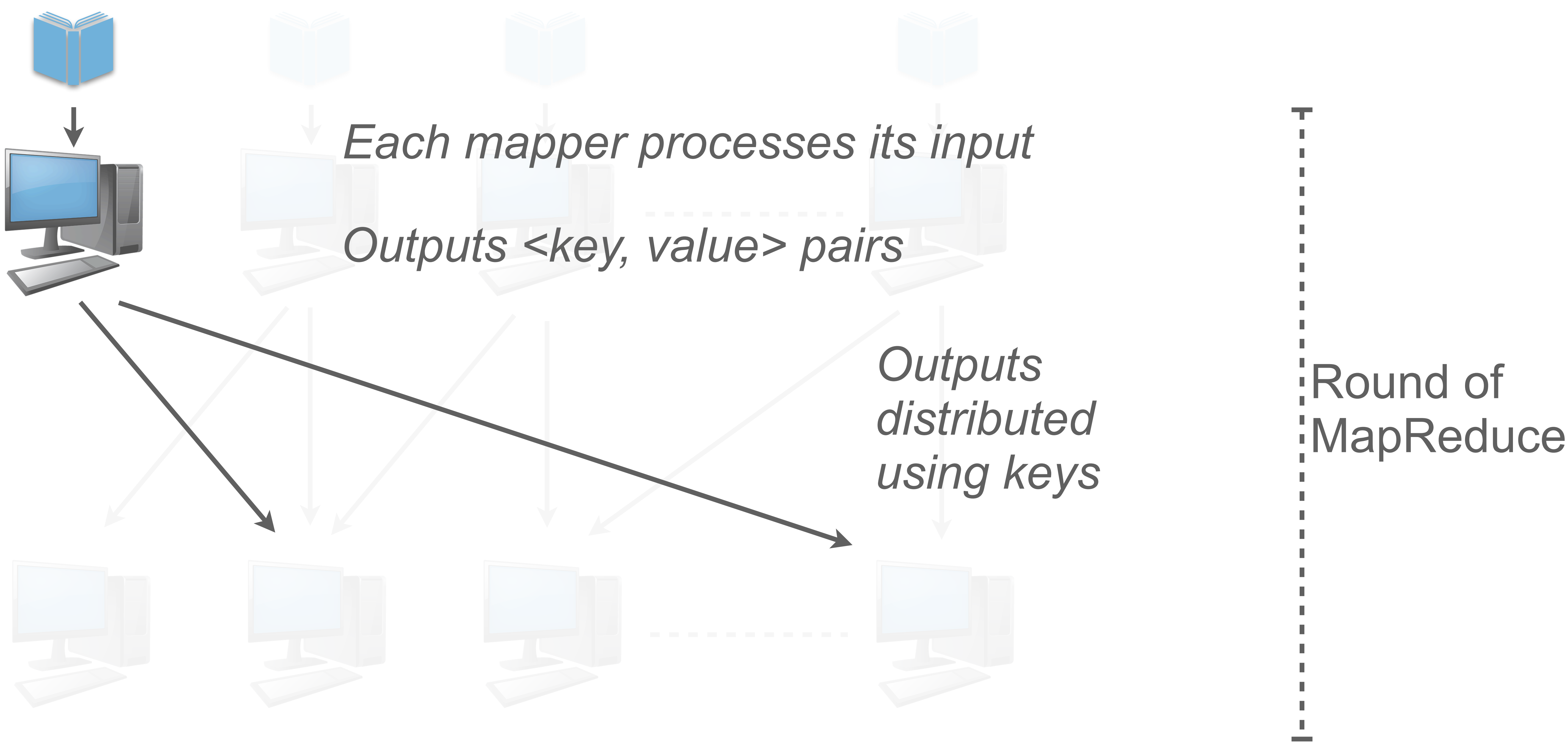
Each mapper processes its input
Outputs <key, value> pairs
Round of MapReduce

MapReduce

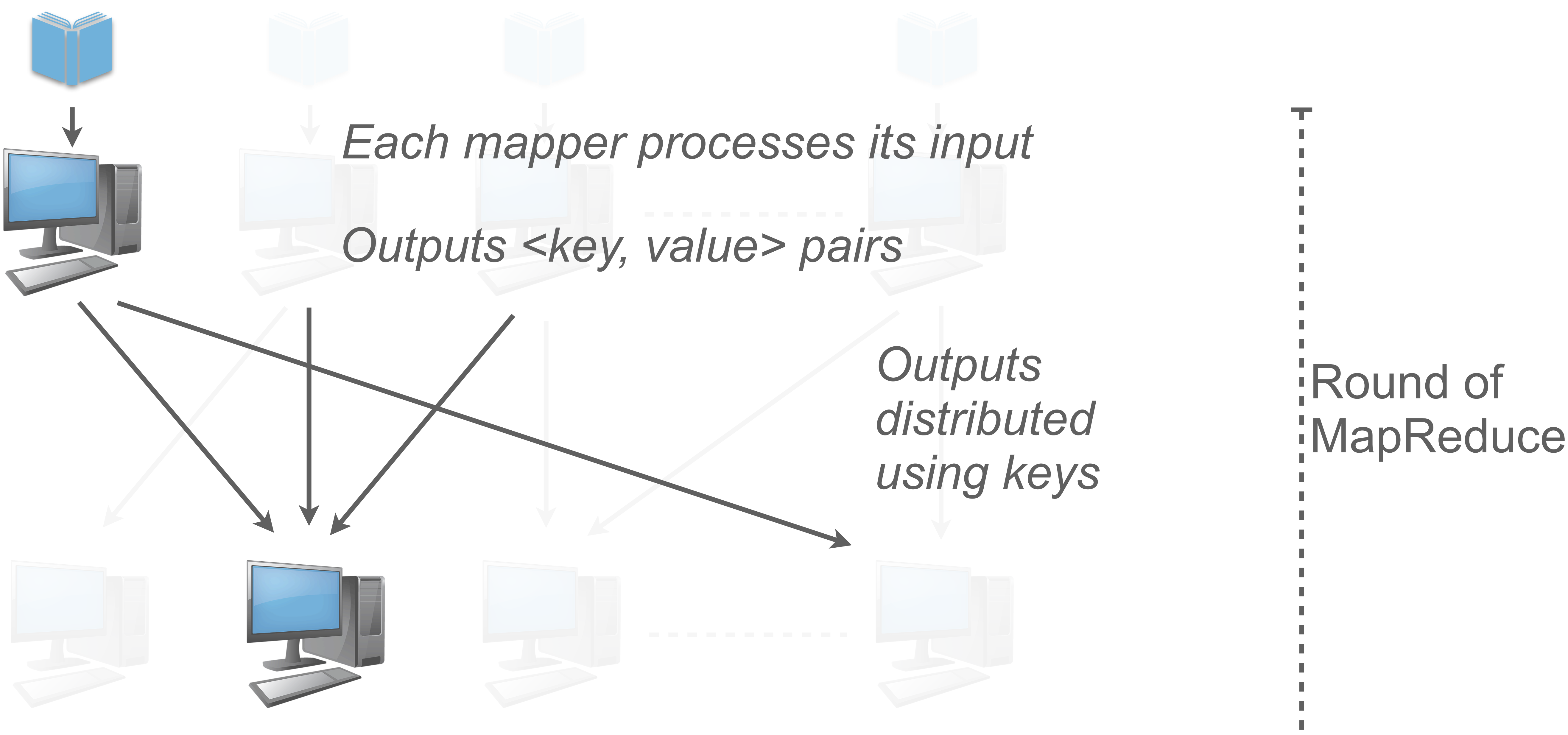


Each mapper processes its input
Outputs <key, value> pairs
Round of MapReduce

MapReduce

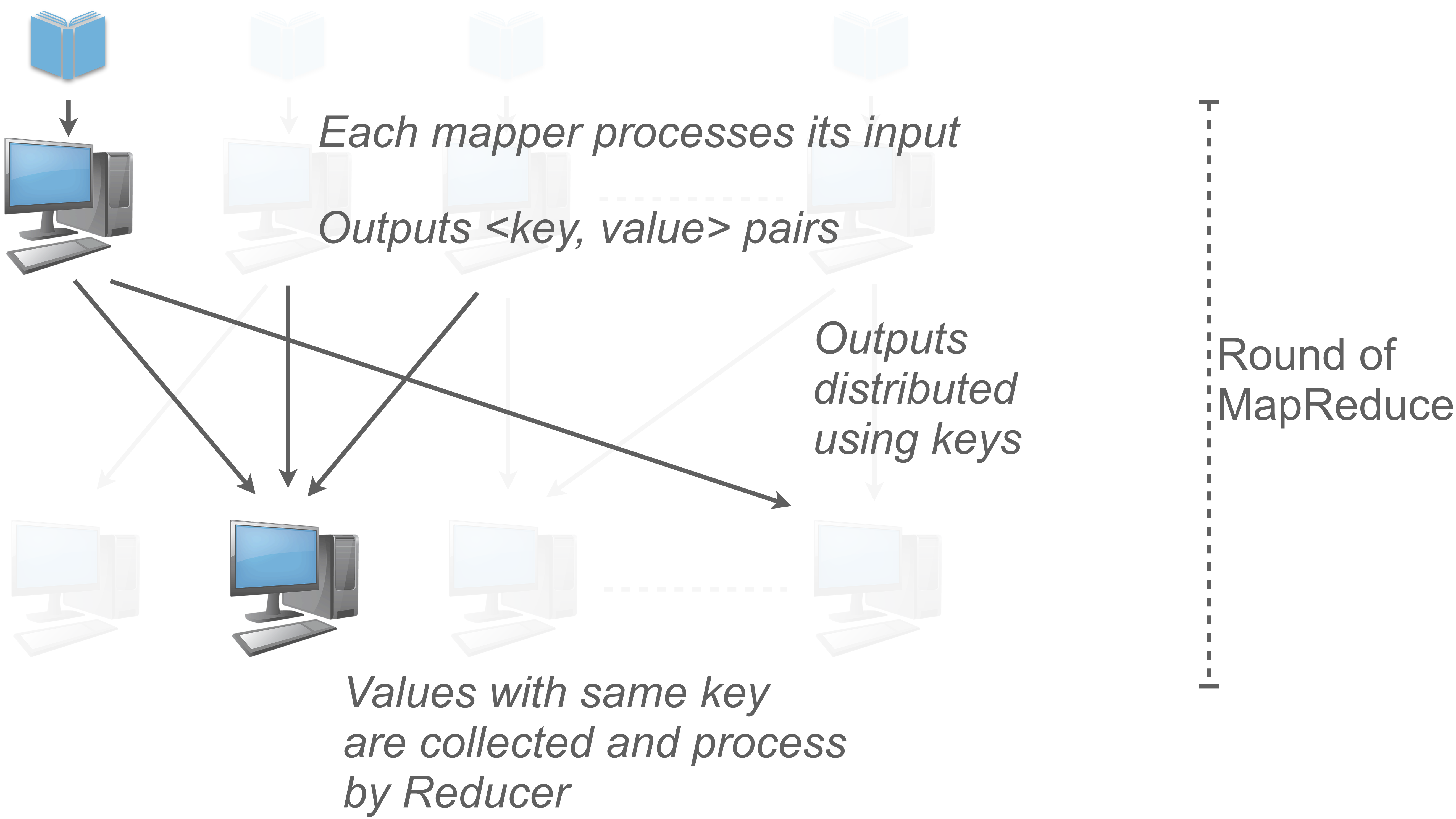

Each mapper processes its input
Outputs <key, value> pairs
Outputs distributed using keys
Round of MapReduce

MapReduce



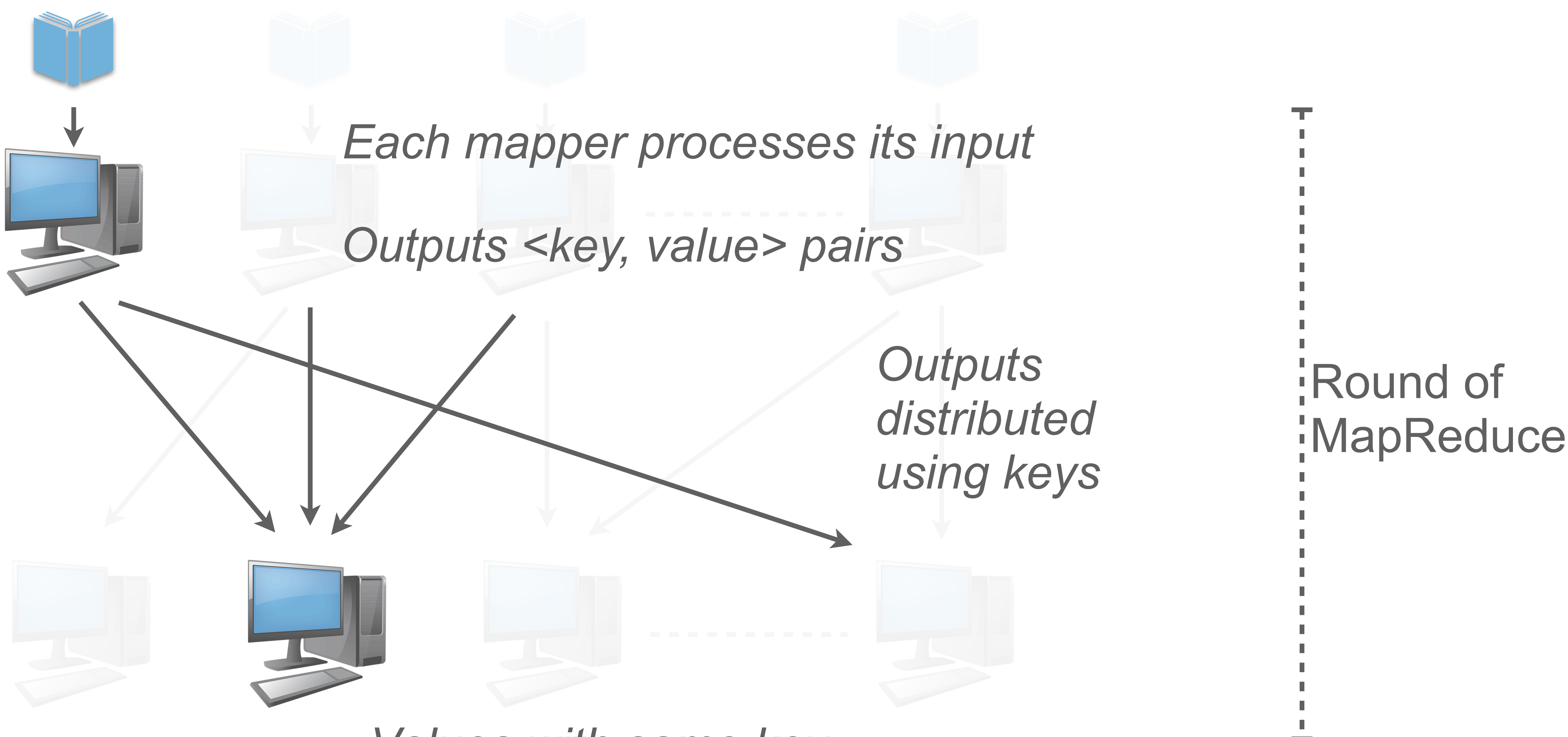
Each mapper processes its input
Outputs <key, value> pairs
Outputs distributed using keys
Round of MapReduce

MapReduce



Each mapper processes its input
Outputs <key, value> pairs
Outputs distributed using keys
Round of MapReduce
Values with same key are collected and process by Reducer

MapReduce



Values with same key are collected and process by Reducer

[MapReduce: simplified data processing on large clusters](#)
J Dean, S Ghemawat
Communications of the ACM 51 (1), 107-113

Simple example: WordCount

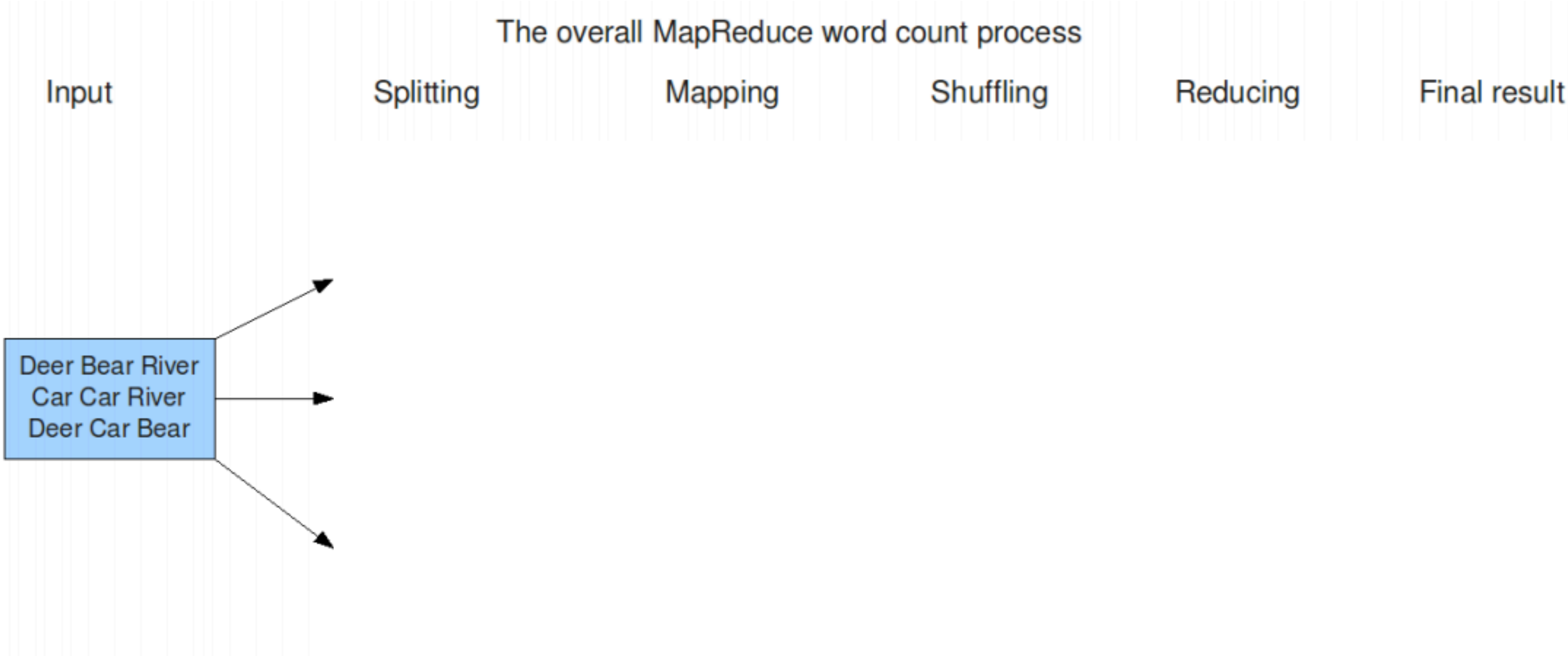


Diagram from http://blog.jteam.nl/wp%E2%80%90content/uploads/2009/08/MapReduceWordCountOverview1.png.

Simple example: WordCount

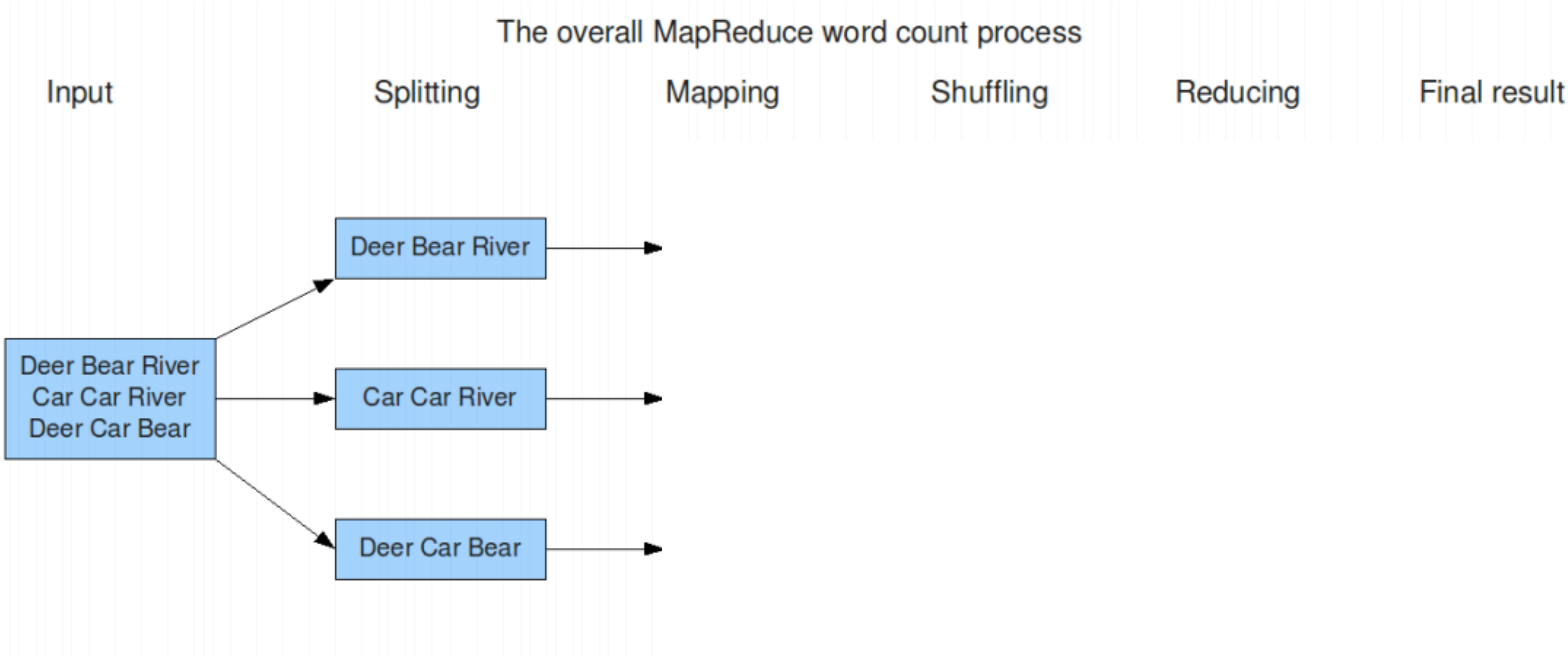


Diagram from http://blog.jteam.nl/wp%E2%80%90content/uploads/2009/08/MapReduceWordCountOverview1.png.

Simple example: WordCount

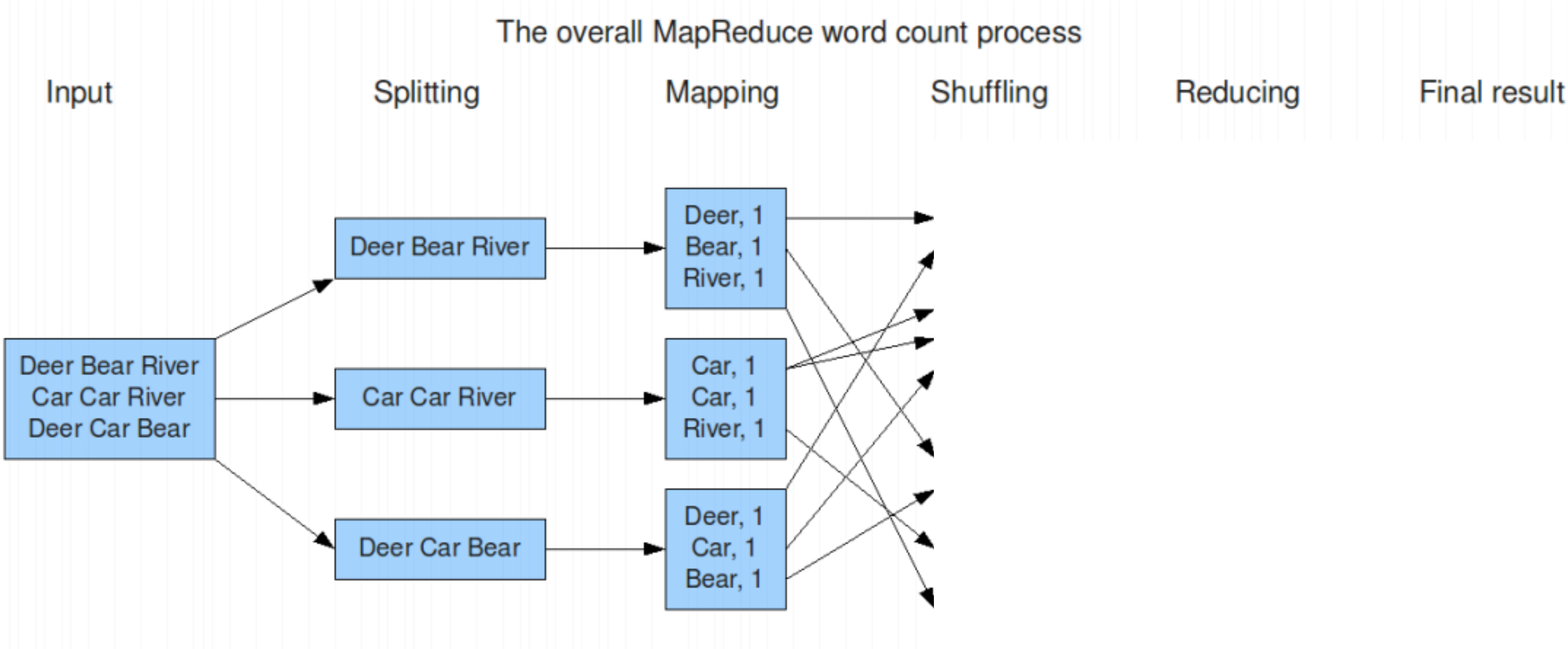


Diagram from http://blog.jteam.nl/wp%E2%80%90content/uploads/2009/08/MapReduceWordCountOverview1.png.

Simple example: WordCount

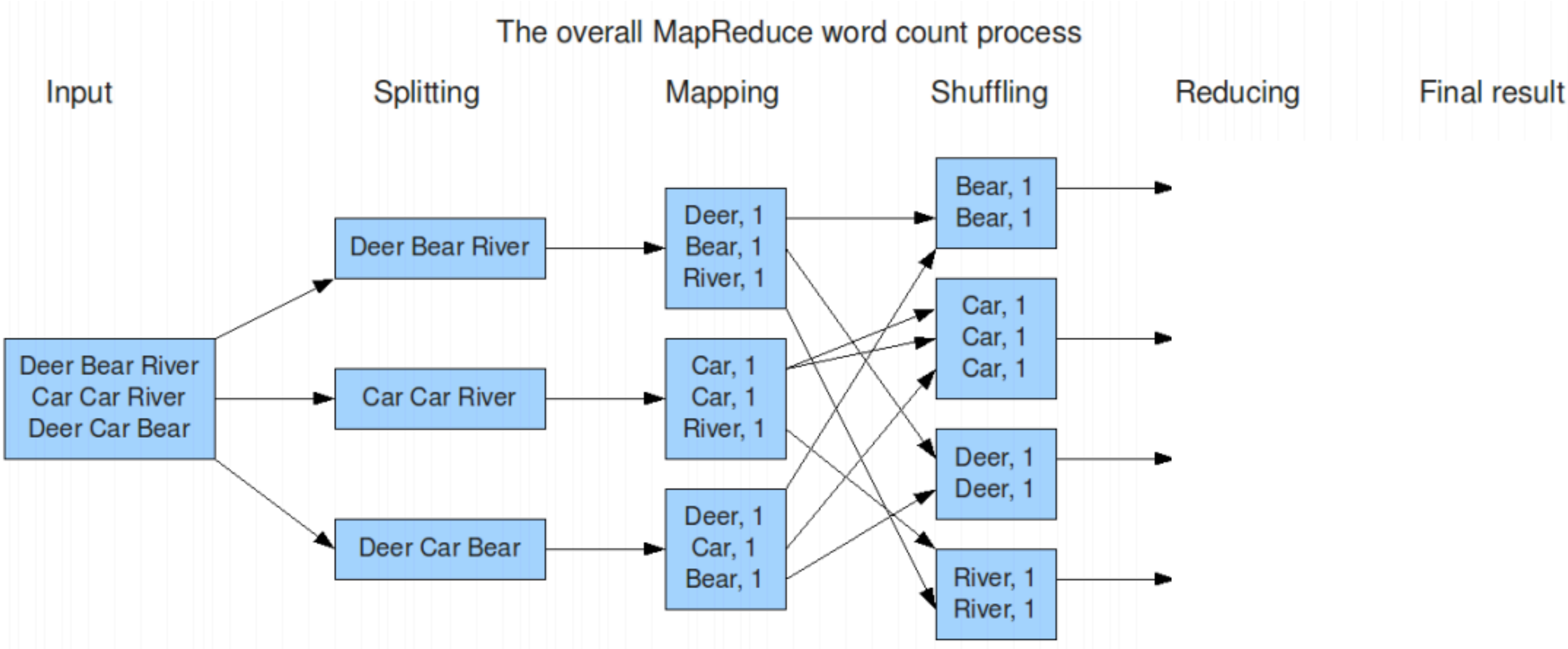


Diagram from http://blog.jteam.nl/wp%E2%80%90content/uploads/2009/08/MapReduceWordCountOverview1.png.

Simple example: WordCount

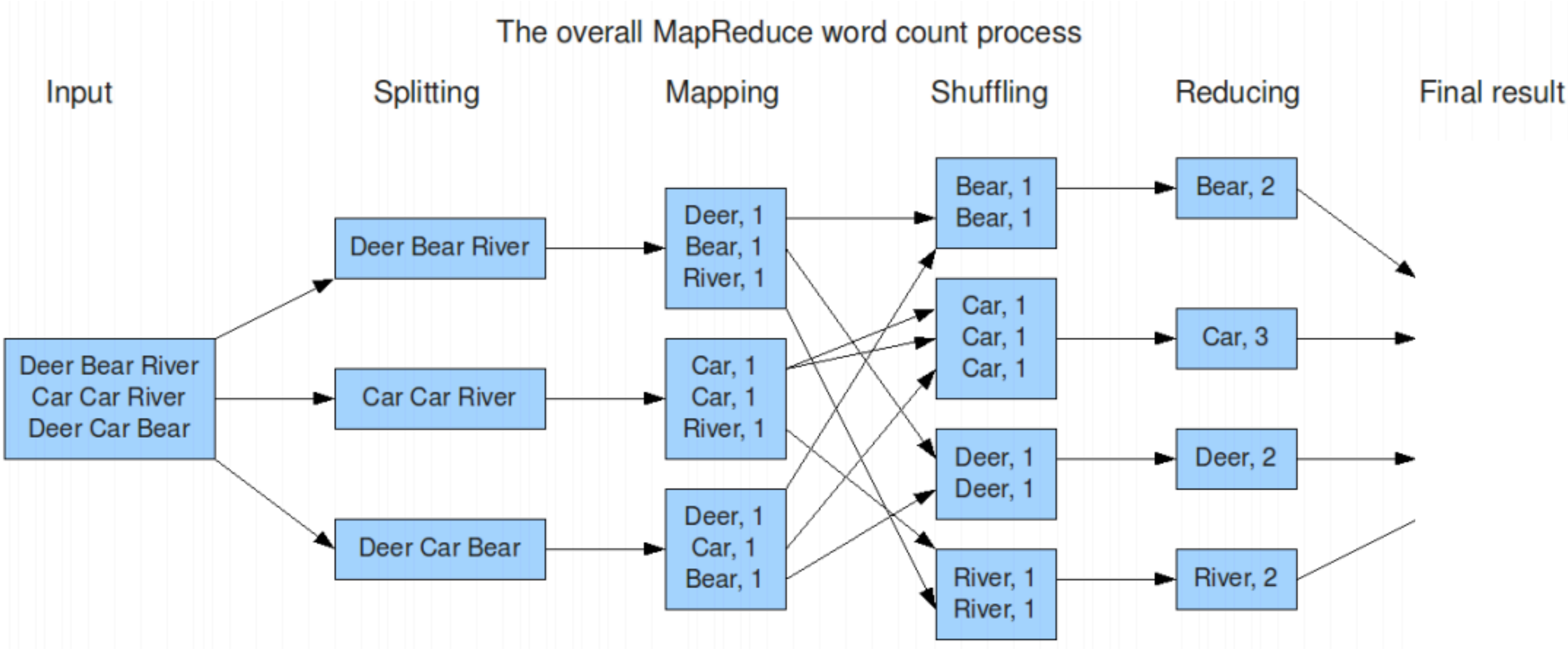


Diagram from http://blog.jteam.nl/wp%E2%80%90content/uploads/2009/08/MapReduceWordCountOverview1.png.

Simple example: WordCount

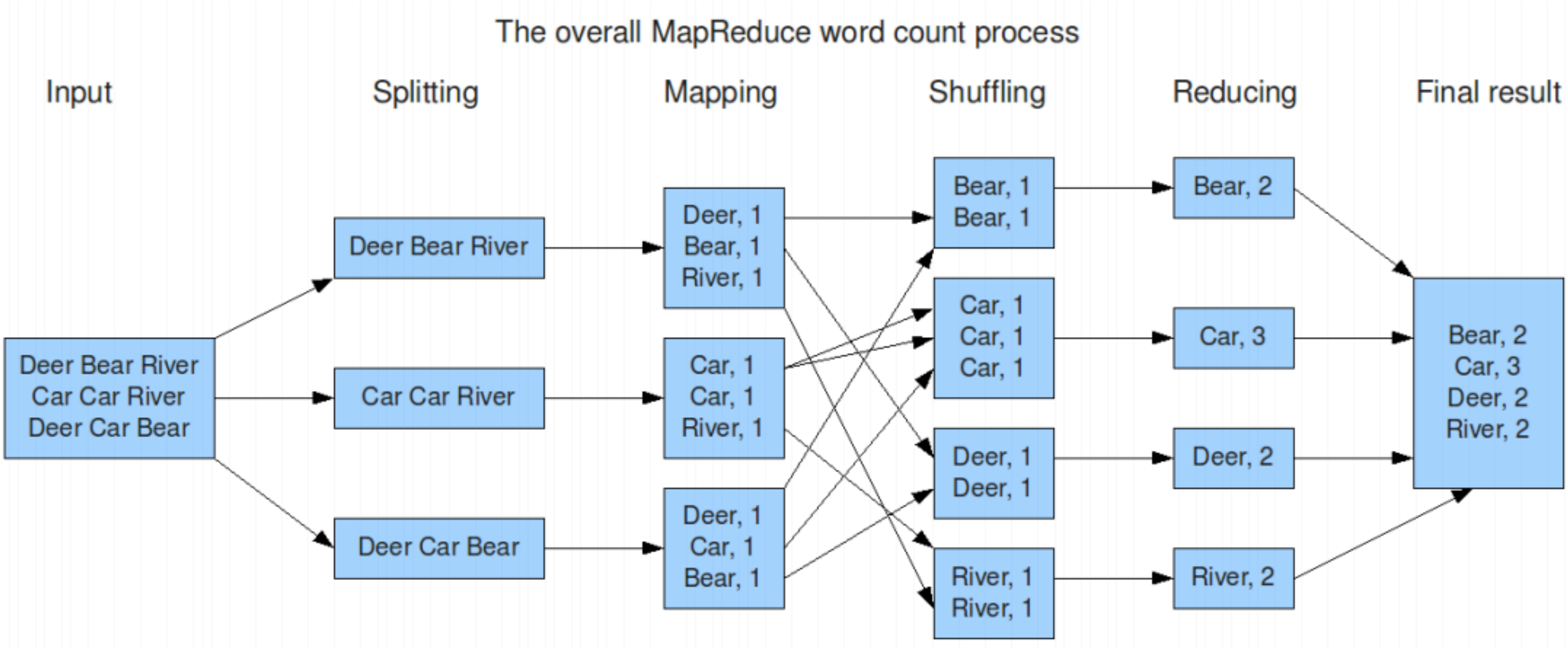
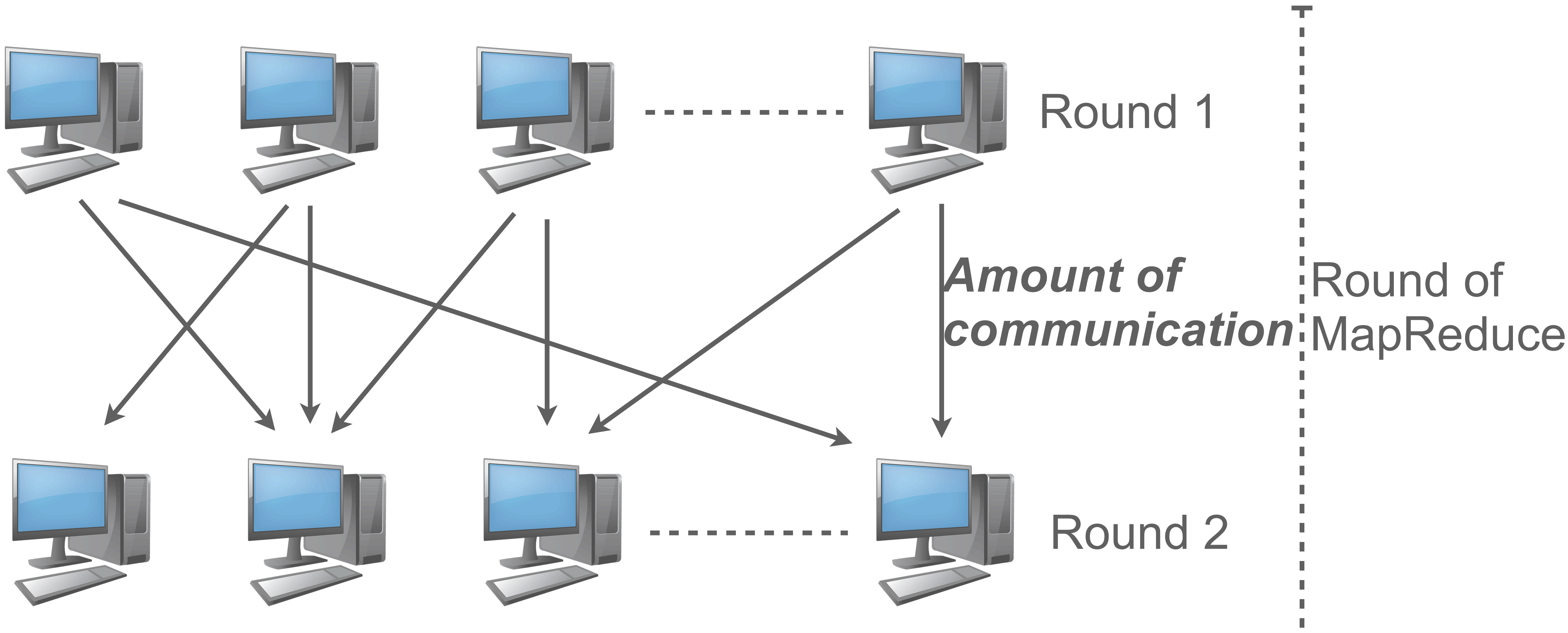


Diagram from http://blog.jteam.nl/wp%E2%80%90content/uploads/2009/08/MapReduceWordCountOverview1.png.

MapReduce

 # machines



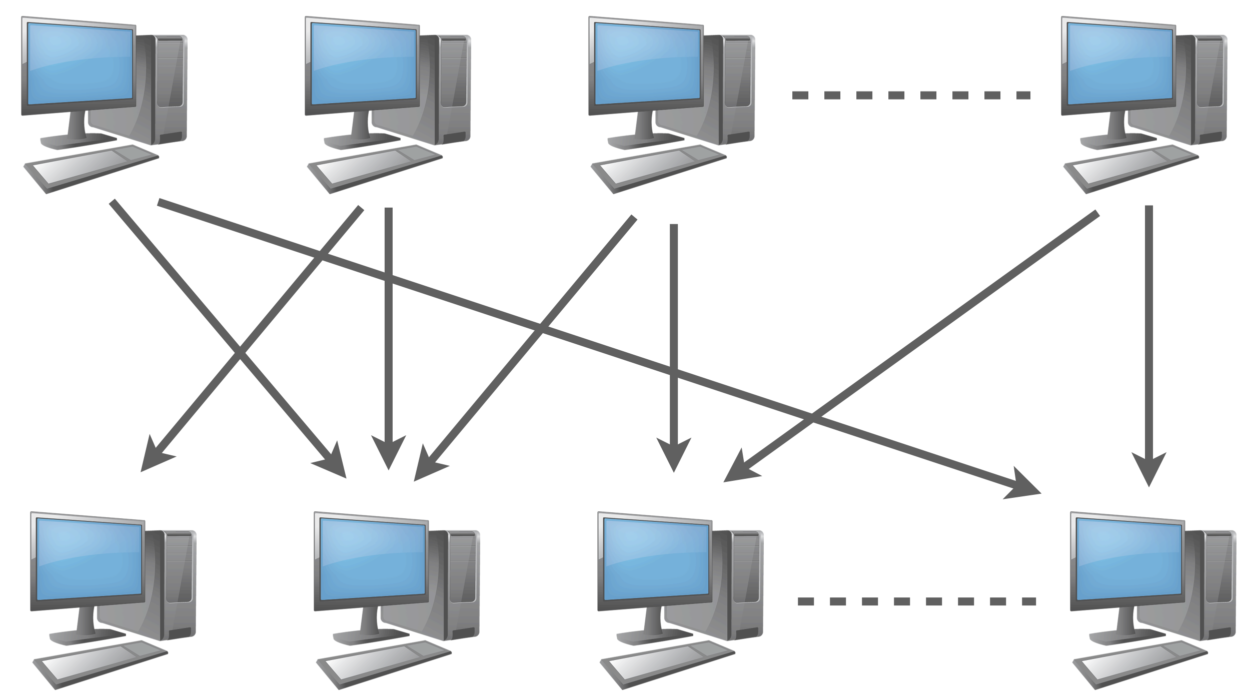
 # rounds

MapReduce model

 # machines (M)

Input size to machines(S)

 # rounds (R)



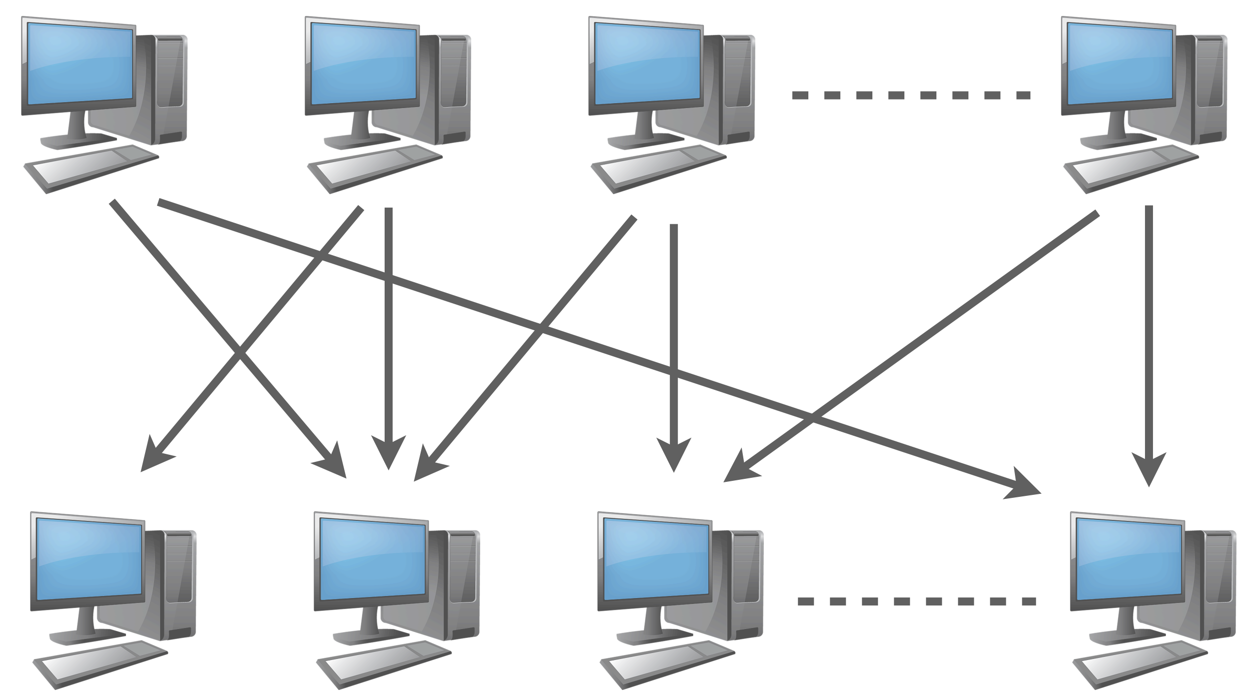
MapReduce model

Input size N

 # machines (M)

Input size to machines(S)

 # rounds (R)



MapReduce model

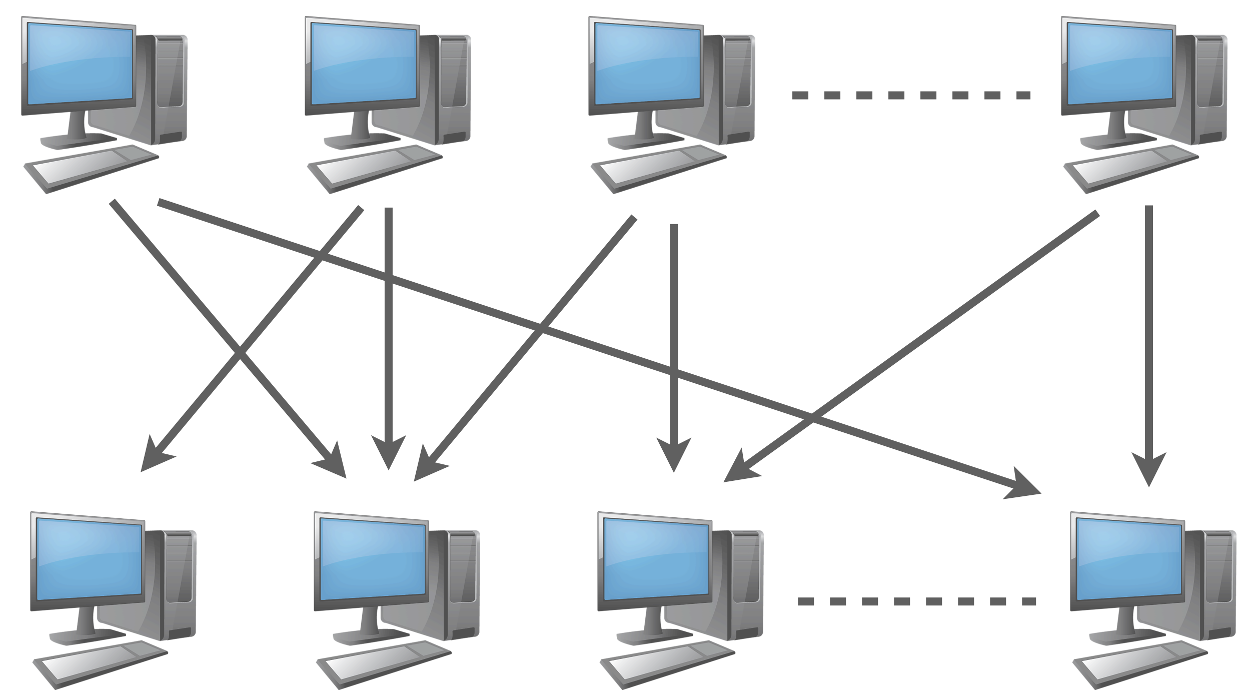
Input size N

 # machines (M)

$O\left(N^{1-\epsilon}\right)$

for constant $\epsilon > 0$

Input size to machines(S)



 # rounds (R)

MapReduce model

Input size N

 # machines (M)

$O\left(N^{1-\epsilon}\right)$

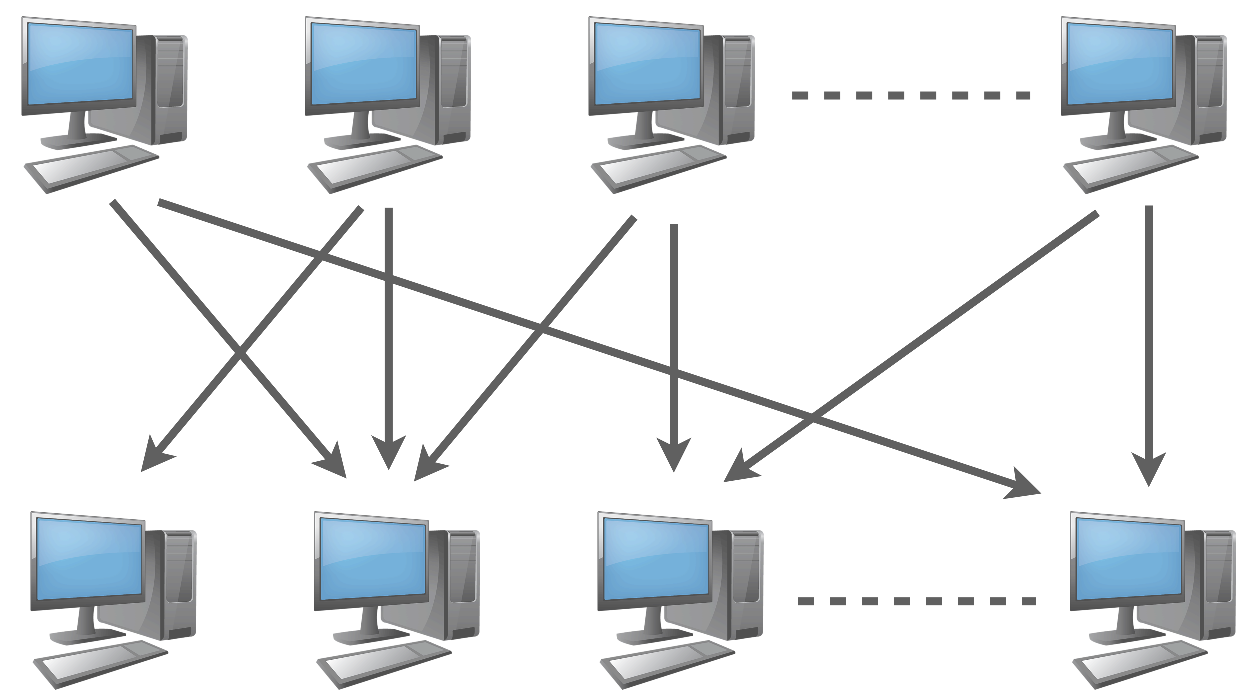
for constant $\epsilon > 0$

Input size to machines(S)

$O\left(N^{1-\epsilon}\right)$

for constant $\epsilon > 0$

 # rounds (R)



MapReduce model

Input size N

 # machines (M)

$O\left(N^{1-\epsilon}\right)$

for constant $\epsilon > 0$

Input size to machines(S)

$O\left(N^{1-\epsilon}\right)$

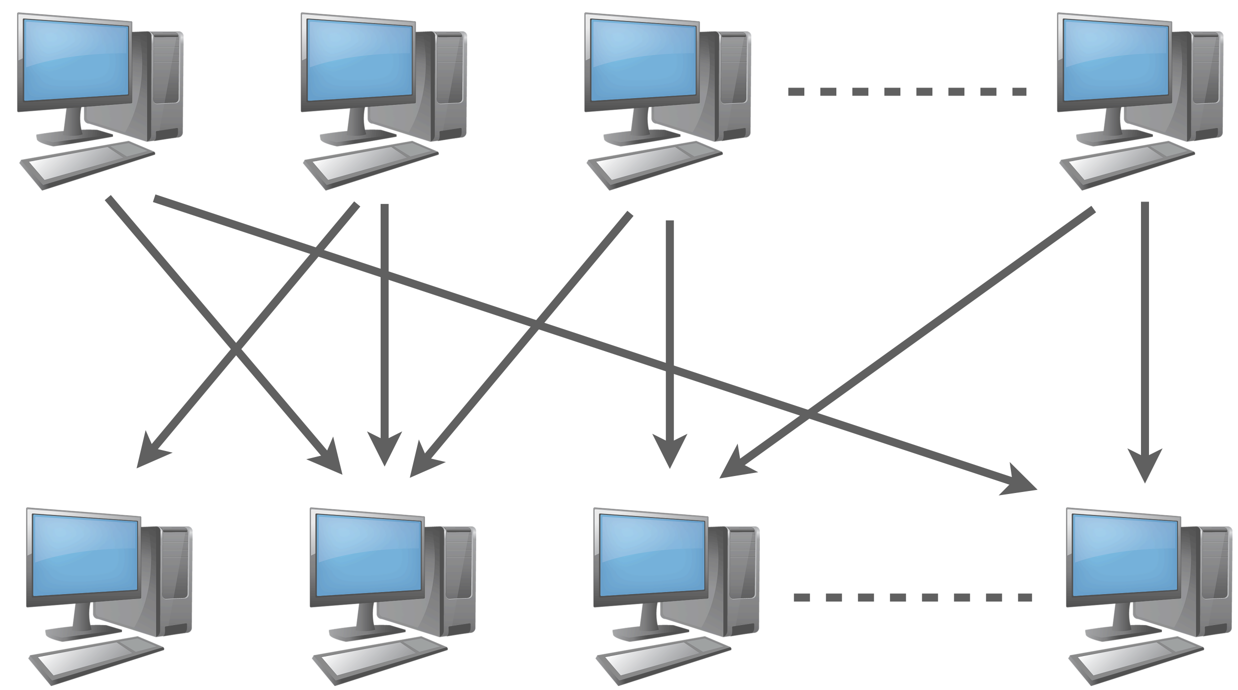
for constant $\epsilon > 0$

 # rounds (R)

$O(1)$ ideal

$O(\log N)$ happy

$O(polylog N)$ content



MapReduce model

Input size N

 # machines (M)

$O\left(N^{1-\epsilon}\right)$

for constant $\epsilon > 0$

Input size to machines(S)

$O\left(N^{1-\epsilon}\right)$

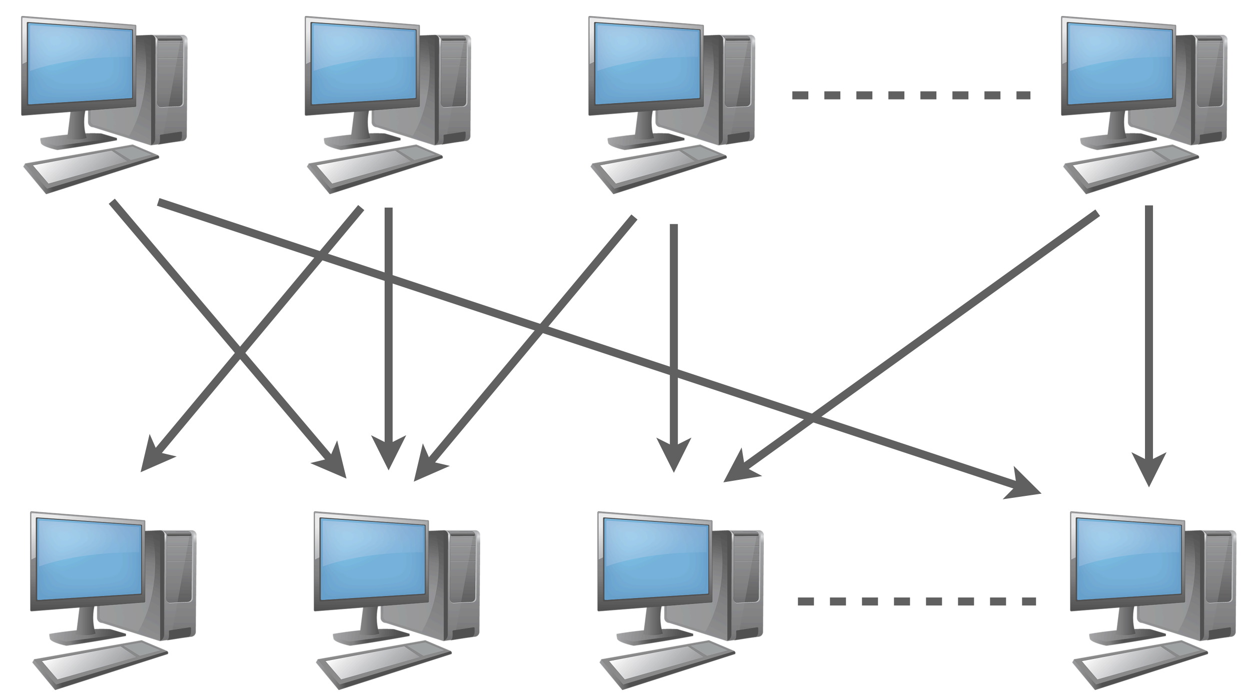
for constant $\epsilon > 0$

 # rounds (R)

$O(1)$ ideal

$O(\log N)$ happy

$O(polylog N)$ content



A model of computation for MapReduce
H Karloff, S Suri, S Vassilvitskii
Proceedings of the twenty-first annual ACM-SIAM symposium on Discrete Algorithms

Other models: parallel

PRAMS

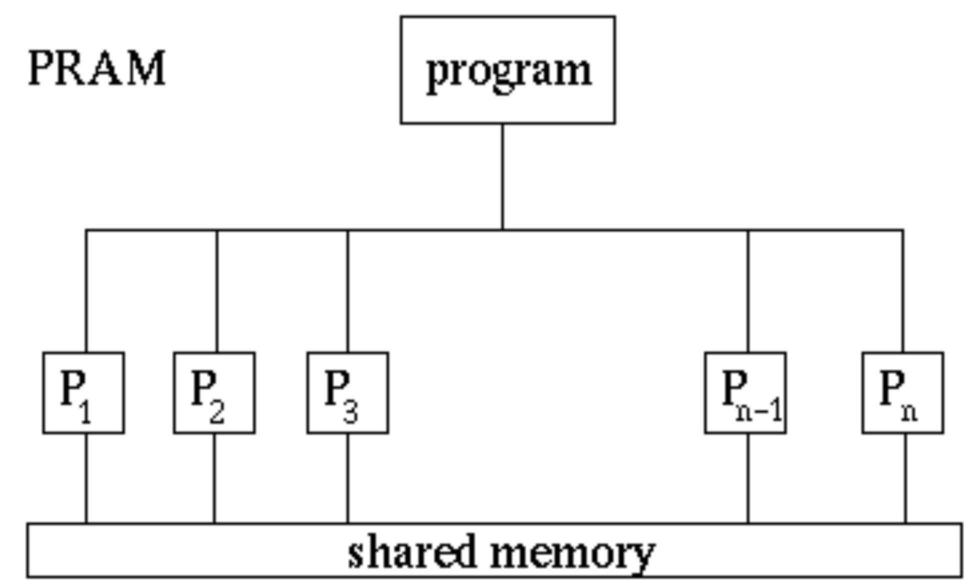
 # machines (M)

$$O(N^c)$$

for constant c > 0

Input size to machines(S)

$$O(1)$$



Other models: parallel

PRAMS

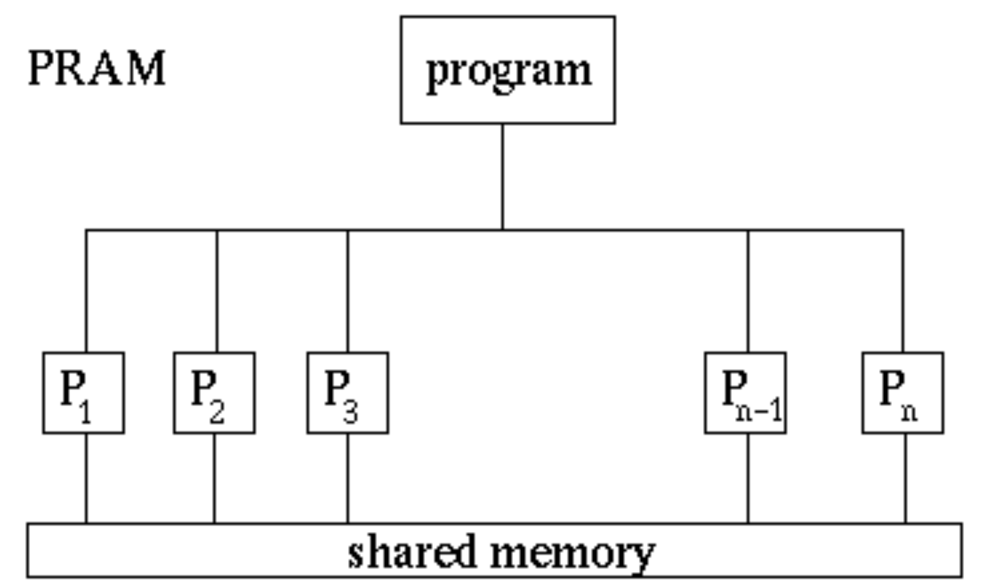
 # machines (M)

$O(N^c)$

for constant c > 0

Input size to machines(S)

$O(1)$

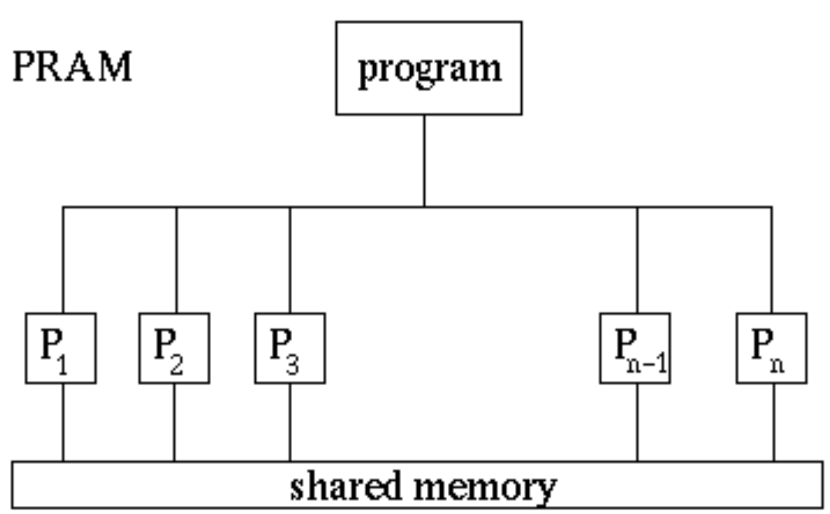


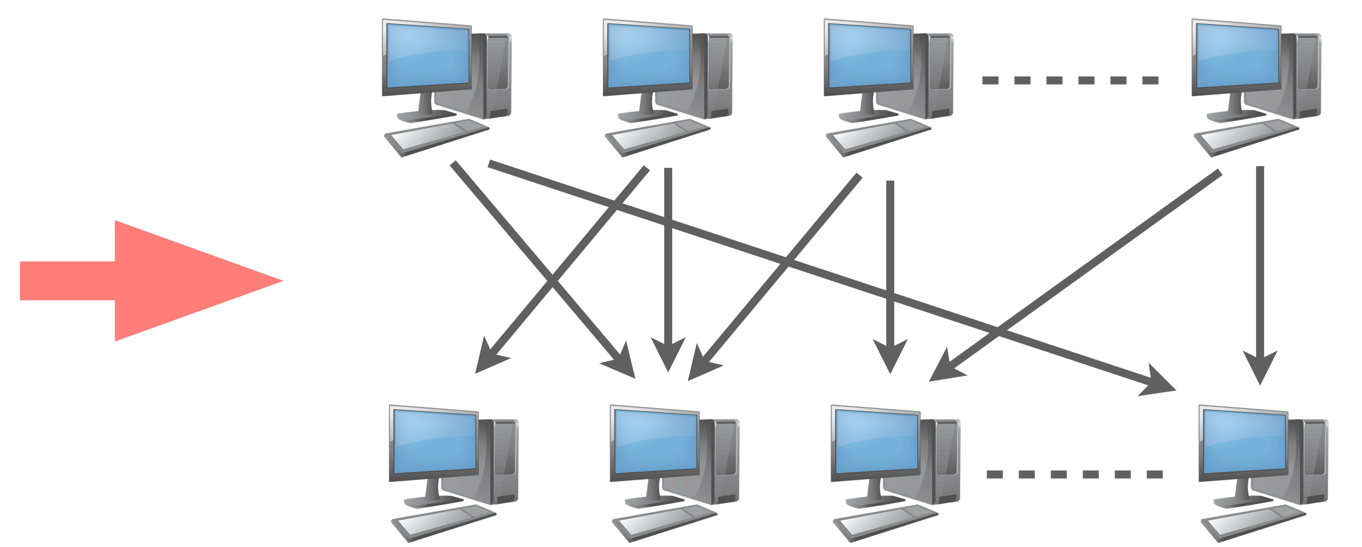
Reduction between EREW PRAMs and MapReduce algorithms

EREW PRAM vs. MapReduce

Theorem

Let $\mathcal{A}$ be a EREW PRAM algorithm for problem $\mathcal{P}$ using $O\left(N^{2-2\epsilon}\right)$ memory and R rounds. Then there exists a MapReduce algorithm using the same number of rounds.



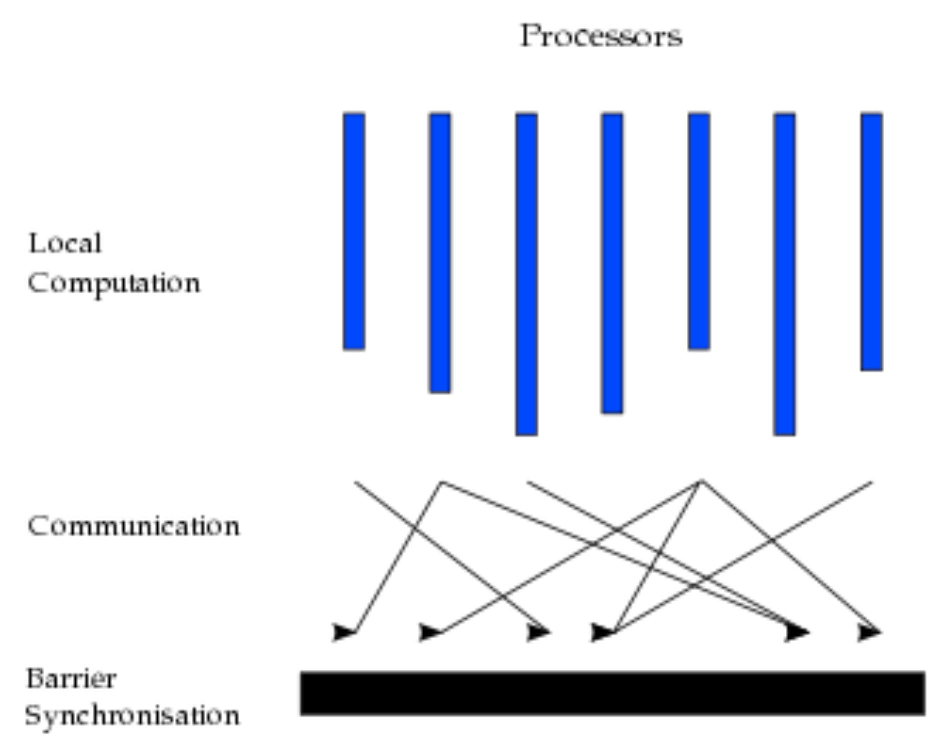


Other models: parallel

BSP

Not assume synchronization

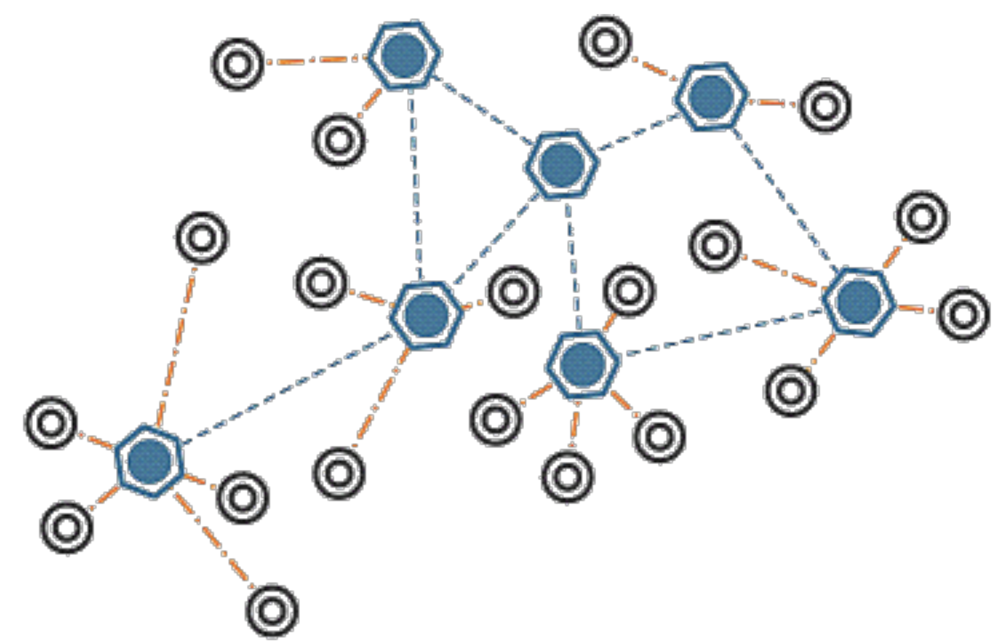
Not assume fault-tolerance



Other models: distributed

LOCAL

Restricted topology



CONGEST

Limited bandwidth

Many connections between the two models, both for upper and lower bounds.

Is it just PRAM?

Is it just PRAM?

Prefix sum problem

Let $\mathbf{v}$ be a vector. Compute $, \forall j, \; PS(j) = \sum_{i \leq j} \mathrm{v}[i]$

Is it just PRAM?

Prefix sum problem

Let $\mathbf{v}$ be a vector. Compute, $\forall j,\ PS(j) = \sum_{i \leq j} \mathrm{v}[i]$

Using EREW PRAM it is not clear how to design an algorithm using $o(\log n)$ rounds.

Using MapReduce it is possible to solve the problem in $O(1)$ rounds

Is it just PRAM?

Prefix sum problem

Let $\mathbf{v}$ be a vector. Compute $, \forall j, \; PS(j) = \sum_{i \leq j} \mathrm{v}[i]$

Using MapReduce:

Is it just PRAM?

Prefix sum problem

Let $\mathbf{v}$ be a vector. Compute $, \forall j, \; PS(j) = \sum_{i \leq j} \mathrm{v}[i]$

Using MapReduce:

Suppose the input is sorted,

$\mathrm{v}[0]$ --- $\mathrm{v}[s-1]$	$\mathrm{v}[s]$ --- $\mathrm{v}[2s-1]$	--------------	$\mathrm{v}[(m-2)s]$--- $\mathrm{v}[(m-1)s-1]$

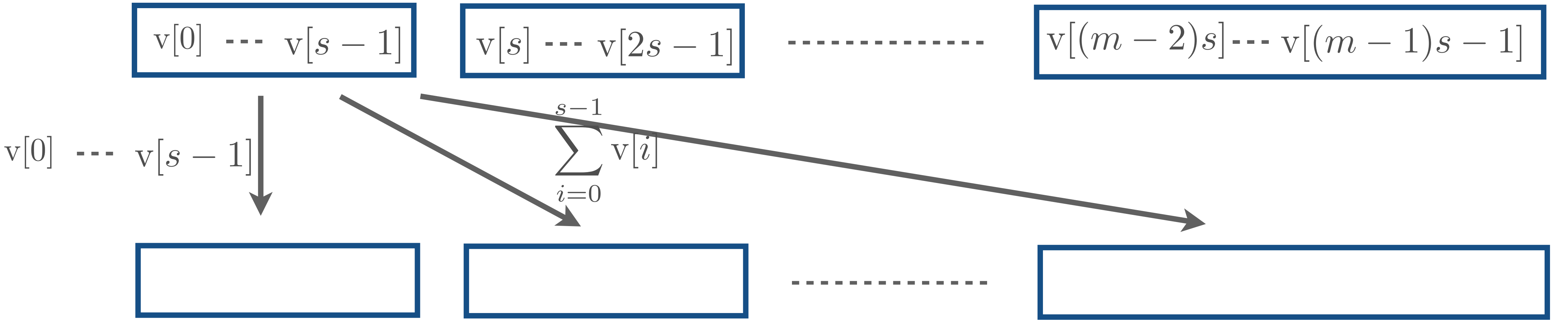
Is it just PRAM?

Prefix sum problem

Let $\mathbf{v}$ be a vector. Compute , $\forall j,\ PS(j) = \sum_{i \le j} \mathrm{v}[i]$

Using MapReduce:

Suppose the input is sorted,



Is it just PRAM?

Prefix sum problem

Let $\mathbf{v}$ be a vector. Compute , $\forall j,\ PS(j) = \sum_{i \leq j} \mathrm{v}[i]$

Using MapReduce:

Sorting also takes $O(1)$

$\sqrt{N}$ machines, each containing $\sqrt{N}$ elements

Is it just PRAM?

Prefix sum problem

Let $\mathbf{v}$ be a vector. Compute $, \forall j,\ PS(j) = \sum_{i \leq j} \mathrm{v}[i]$

Using MapReduce:

Sorting also takes $O(1)$

$\sqrt{N}$ machines, each containing $\sqrt{N}$ elements

Sort in each machine.

Is it just PRAM?

Prefix sum problem

Let $\mathbf{v}$ be a vector. Compute, $\forall j,\ PS(j) = \sum_{i \leq j} \mathrm{v}[i]$

Using MapReduce:

Sorting also takes $O(1)$

$\sqrt{N}$ machines, each containing $\sqrt{N}$ elements

Sort in each machine.

Compute the $N^{1/2-\epsilon}$-quantiles.

Is it just PRAM?

Prefix sum problem

Let $\mathbf{v}$ be a vector. Compute , $\forall j,\ PS(j) = \sum_{i \leq j} \mathrm{v}[i]$

Using MapReduce:

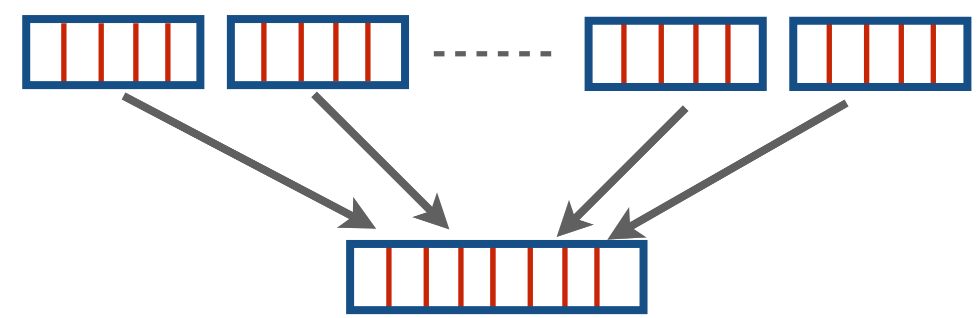
Sorting also takes $O(1)$

$\sqrt{N}$ machines, each containing $\sqrt{N}$ elements

Sort in each machine.

Compute the $N^{1/2-\epsilon}$-quantiles.

Sort quantiles, and use them to partition data in N machines and sort again.



Is it just PRAM?

Prefix sum problem

Let $\mathbf{v}$ be a vector. Compute, $\forall j,\ PS(j) = \sum_{i \leq j} \mathrm{v}[i]$

Using MapReduce:

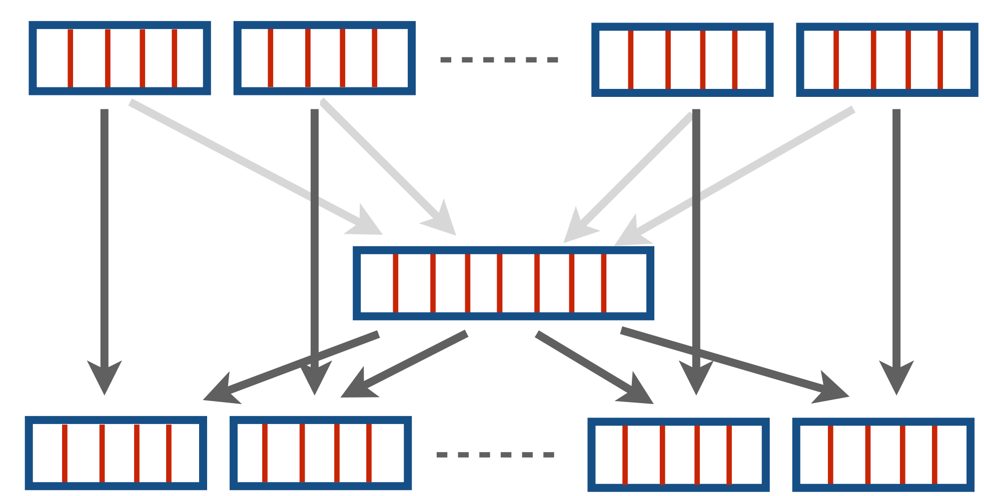
Sorting also takes $O(1)$

$\sqrt{N}$ machines, each containing $\sqrt{N}$ elements

Sort in each machine.

Compute the $N^{1/2-\epsilon}$-quantiles.

Sort quantiles, and use them to partition data in N machines and sort again.



Is it just PRAM?

Prefix sum problem

Let $\mathbf{v}$ be a vector. Compute , $\forall j,\ PS(j) = \sum_{i \leq j} \mathrm{v}[i]$

Using MapReduce:

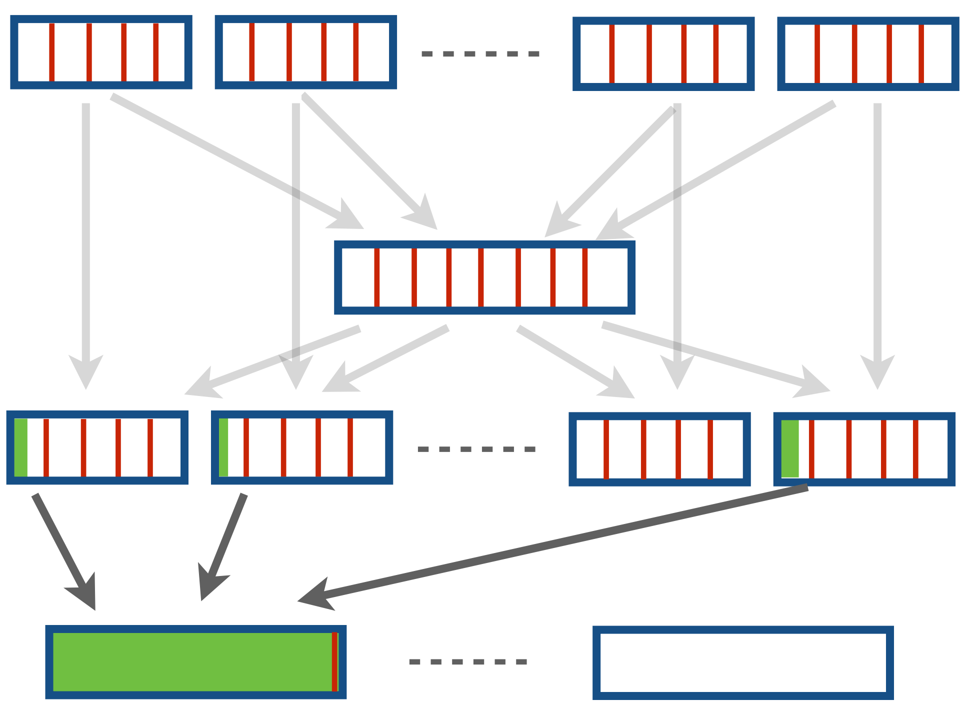
Sorting also takes $O(1)$

$\sqrt{N}$ machines, each containing $\sqrt{N}$ elements

Sort in each machine.

Compute the $N^{1/2-\epsilon}$-quantiles.

Sort quantiles, and use them to partition data in N machines and sort again.



Is it just PRAM?

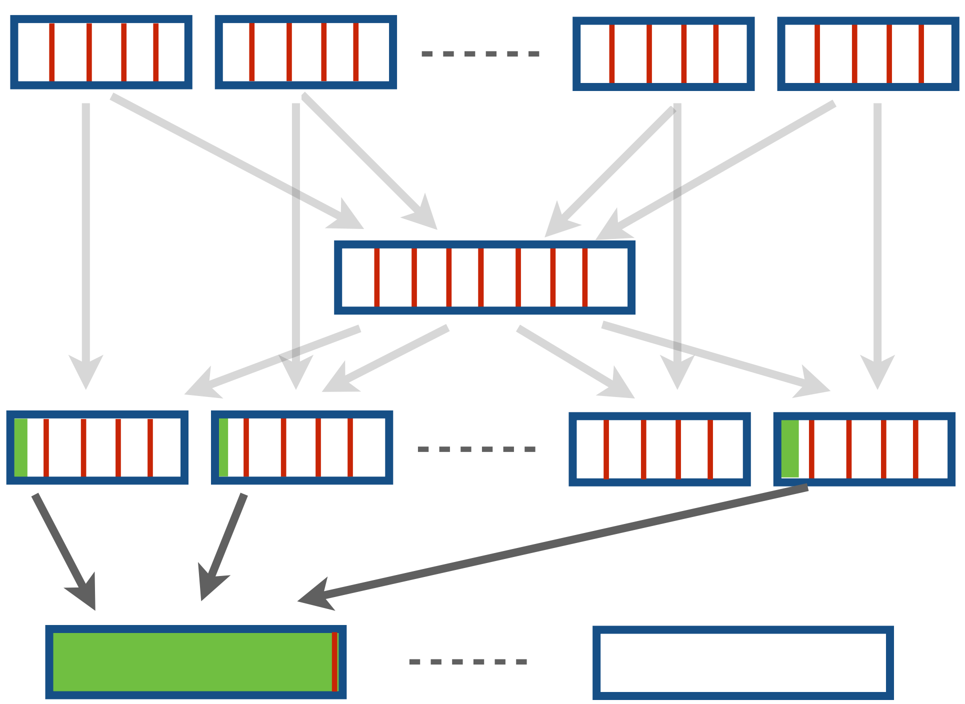
Prefix sum problem

Let $\mathbf{v}$ be a vector. Compute, $\forall j,\ PS(j) = \sum_{i \leq j} \mathrm{v}[i]$

Using MapReduce:

Sorting also takes $O(1)$

3 rounds



Is it just PRAM?

Prefix sum problem

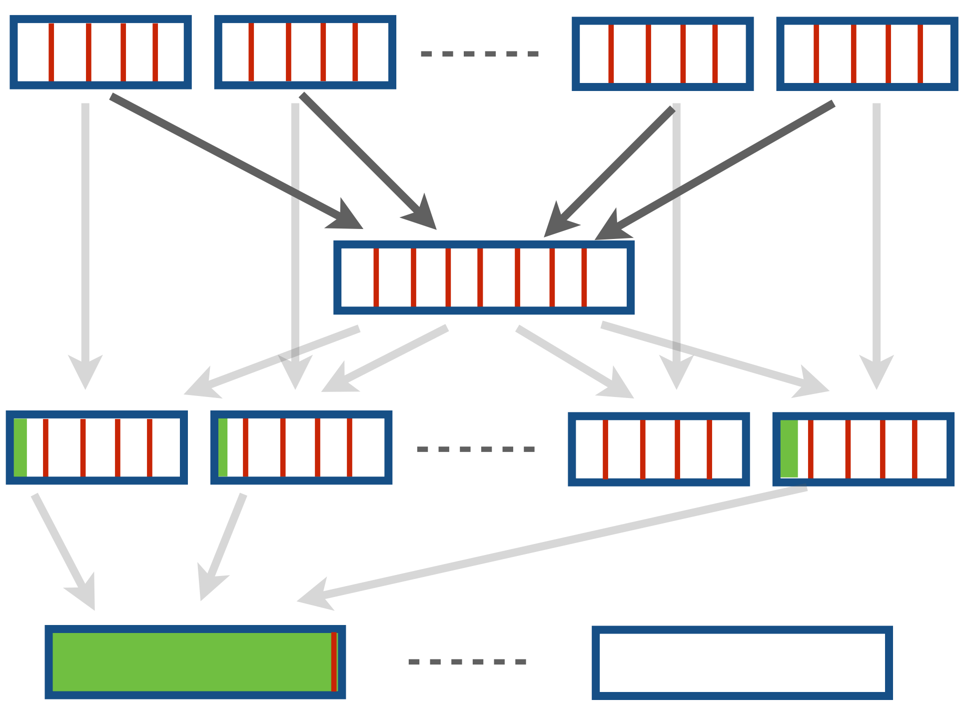
Let $\mathbf{v}$ be a vector. Compute, $\forall j,\ PS(j) = \sum_{i \leq j} \mathrm{v}[i]$

Using MapReduce:

Sorting also takes $O(1)$

1st round

$\sqrt{N}$ machines, each sending $N^{1/2-\epsilon}$ elements



Is it just PRAM?

Prefix sum problem

Let $\mathbf{v}$ be a vector. Compute , $\forall j,\ PS(j) = \sum_{i \leq j} \mathrm{v}[i]$

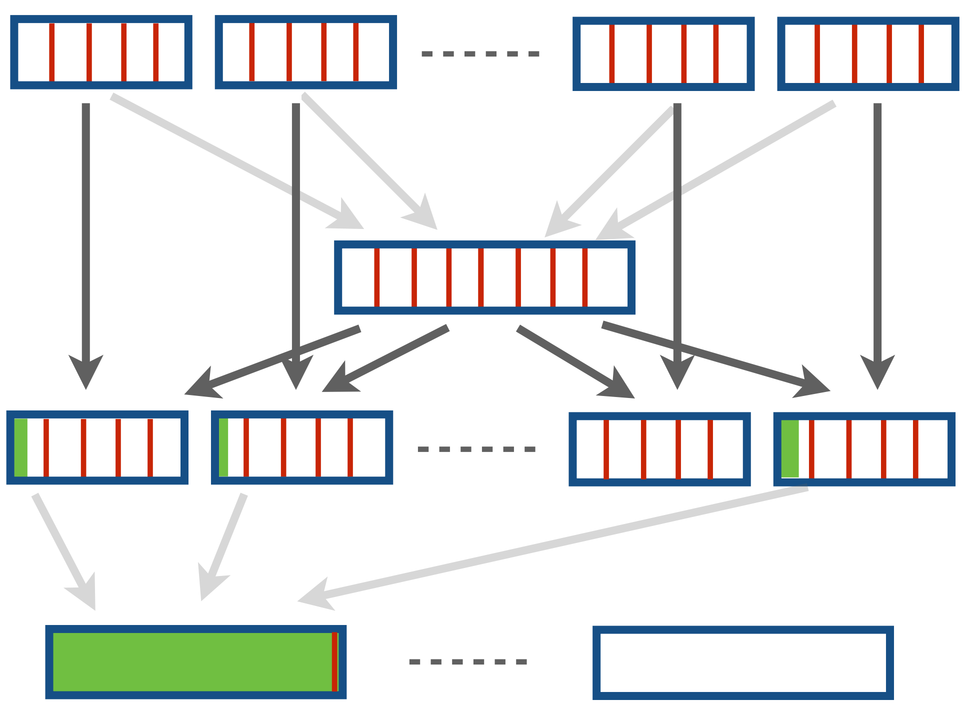
Using MapReduce:

Sorting also takes $O(1)$

2nd round

$\sqrt{N}$ machines, each sending $\sqrt{N}$ elements to different machines

1 machine sending $N^{1-\epsilon}$ elements to all machines



Is it just PRAM?

Prefix sum problem

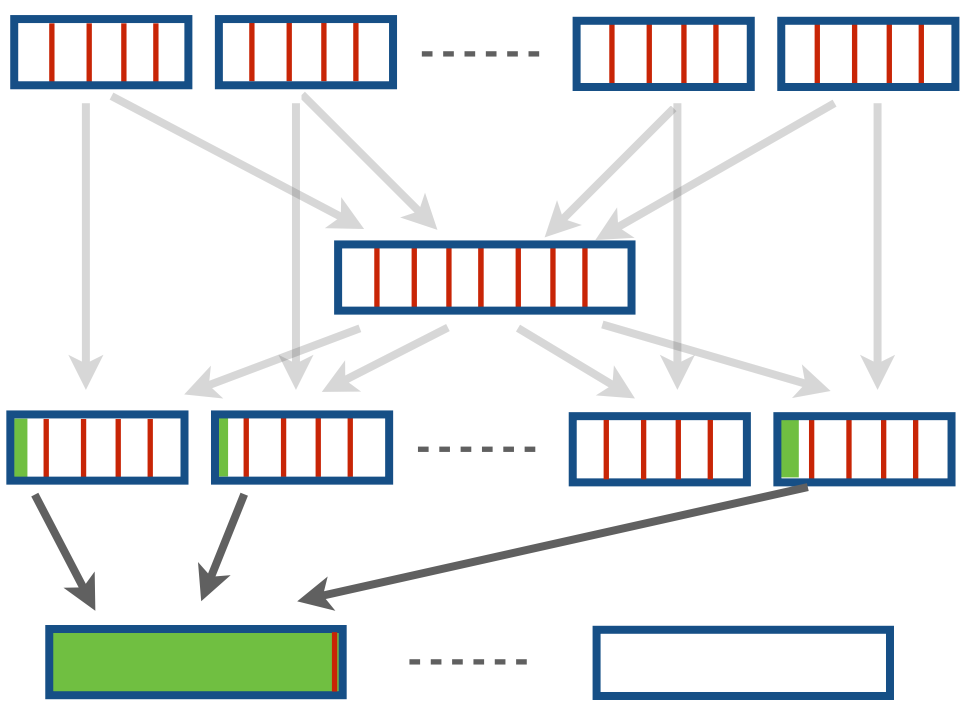
Let $\mathbf{v}$ be a vector. Compute, $\forall j,\ PS(j) = \sum_{i \leq j} \mathrm{v}[i]$

Using MapReduce:

Sorting also takes $O(1)$

3rd round

$\sqrt{N}$ machines, each sending at most N^{ϵ} elements to different machines



Is it just PRAM?

Prefix sum problem

Let $\mathbf{v}$ be a vector. Compute $, \forall j, \;\; PS(j) = \sum_{i \leq j} \mathrm{v}[i]$

Using EREW PRAM it is not clear how to design an algorithm using $o(\log n)$ rounds.

Using MapReduce it is possible to solve the problem in $O(1)$ rounds

MapReduce has been deprecated

Urs Hölzle
@uhoelzle

Follow

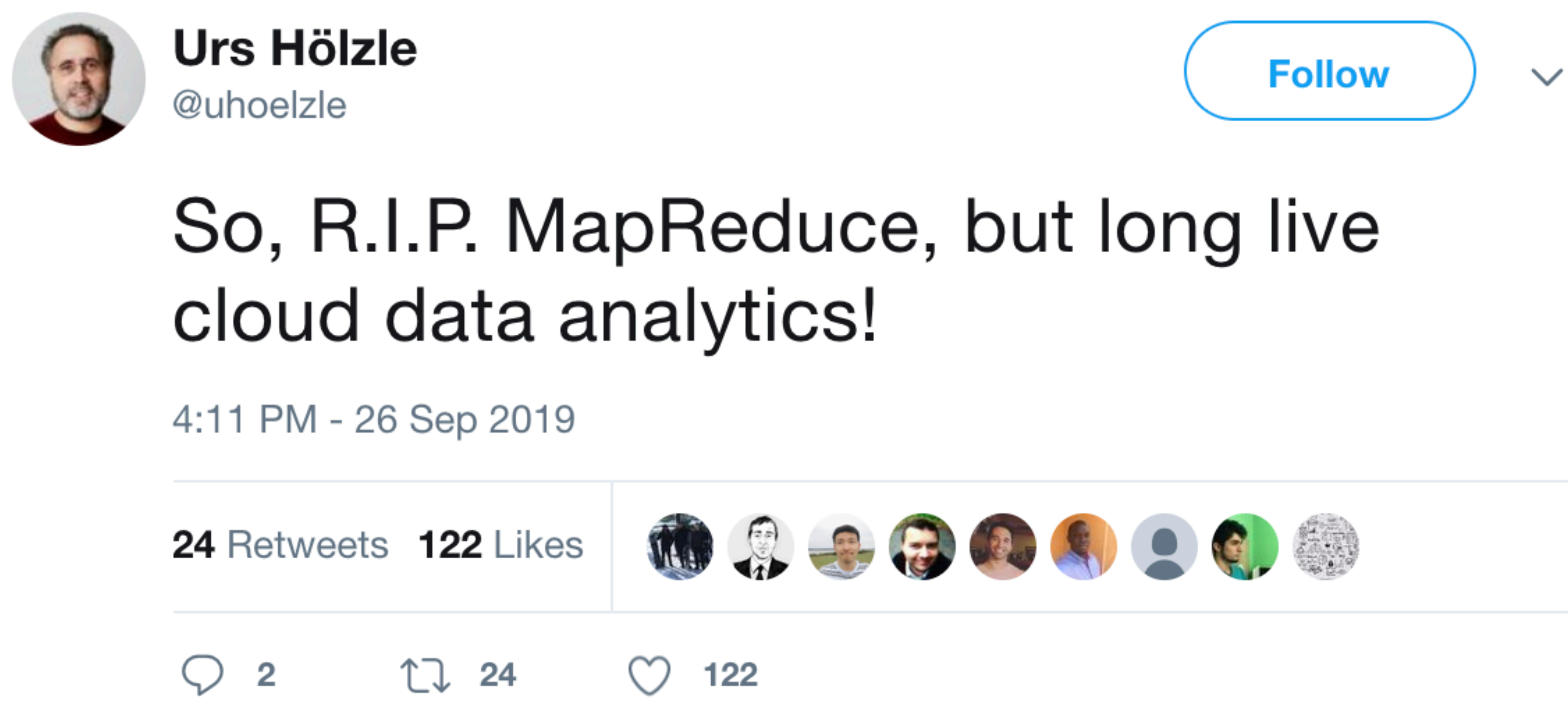
So, R.I.P. MapReduce, but long live cloud data analytics!

4:11 PM - 26 Sep 2019

24 Retweets **122** Likes

2 24 122

MapReduce has been deprecated



Why is this still interesting?

Beyond MapReduce

Different systems but same theoretical abstraction works.

MPC model

Input size N

 # machines (M)

$O\left(N^{1-\epsilon}\right)$ for constant $\epsilon > 0$

Possibly smaller

Input size to machines(S)

$O\left(N^{1-\epsilon}\right)$ for constant $\epsilon > 0$

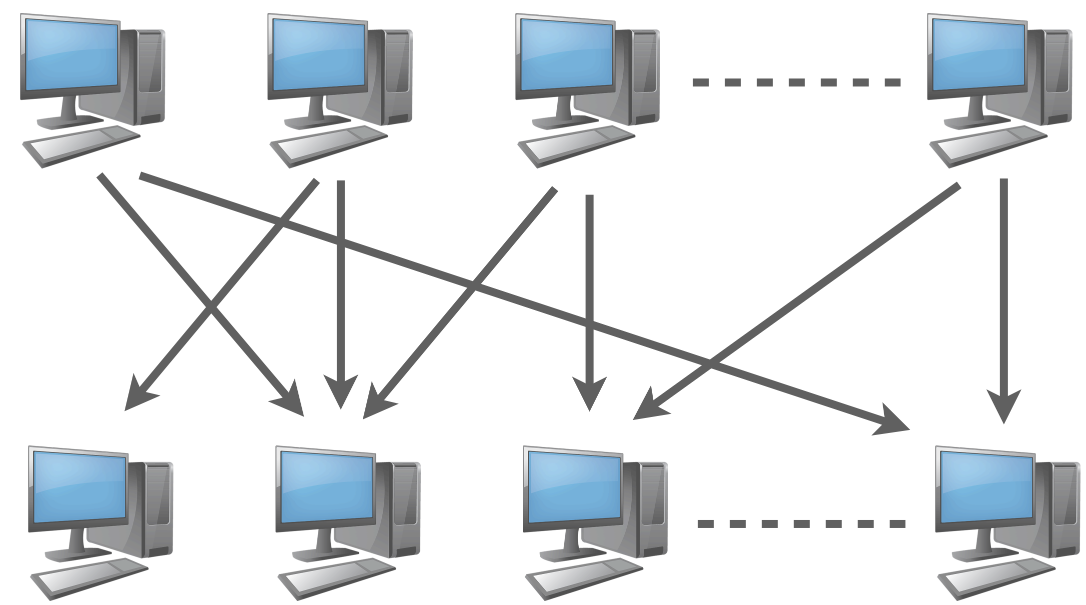
Possibly smaller

 # rounds (R)

$O(1)$ ideal

$O(\log N)$ happy

$O(polylog N)$ content



A model of computation for MapReduce
H Karloff, S Suri, S Vassilvitskii
Proceedings of the twenty-first annual ACM-SIAM symposium on Discrete Algorithms

Some commonalities with EREW PRAM and distributed but different algorithmic power

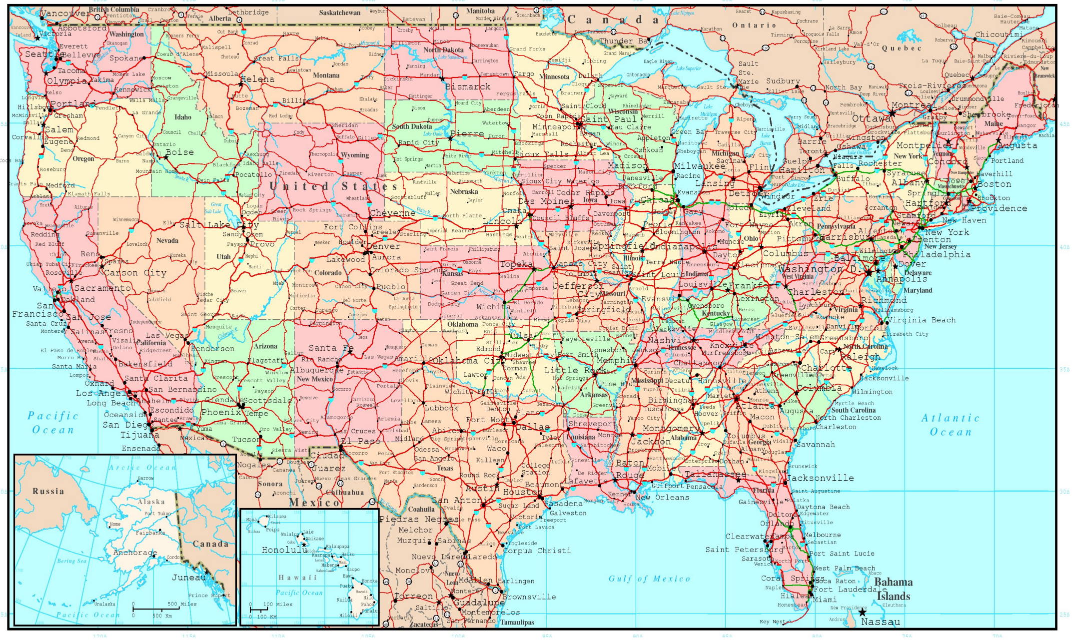
Very active area of research

- Connectivity *[KSV10], [LMSV11], [ASW18], [ASS+18], [BDE+19]…*
- Matching *[LMSV11],[ABB+17],[CML+18],[GGK+18],[GU19],…*
- Metric clustering *[EIM11],[BEL13],[BBLM14],[BW18],…*
- Submodular optimization *[KMVV13],[MZ15],[BENW16],[BEM18]…*
- …

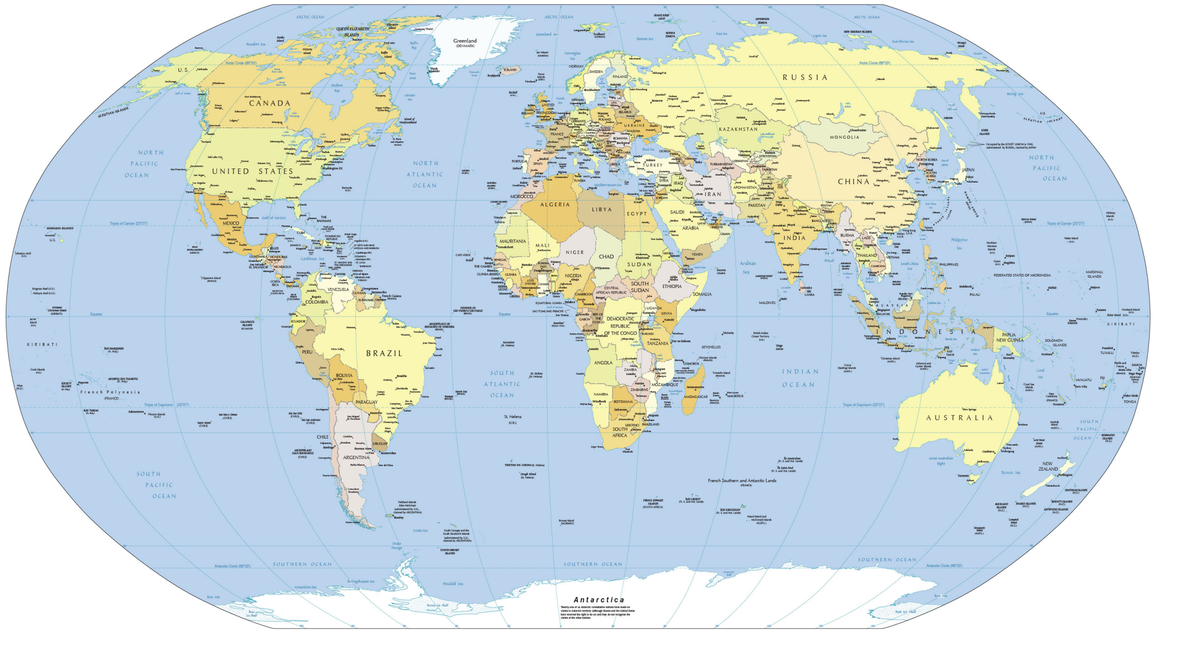
Capacitated Metric clustering At Scale

Why is it important?

How can we cluster these graphs?



US graph: N = x0 Millions
distances: geodesic

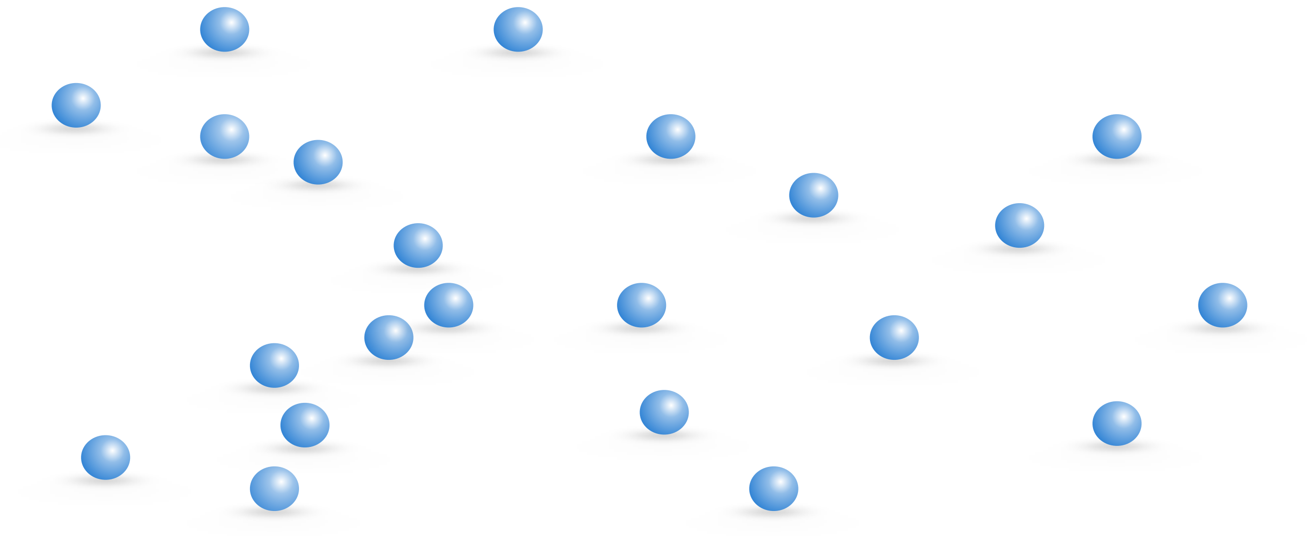


World graph: N = x00 Millions
distances: geodesic

How can we solve such problem?

We use to main ingredients:

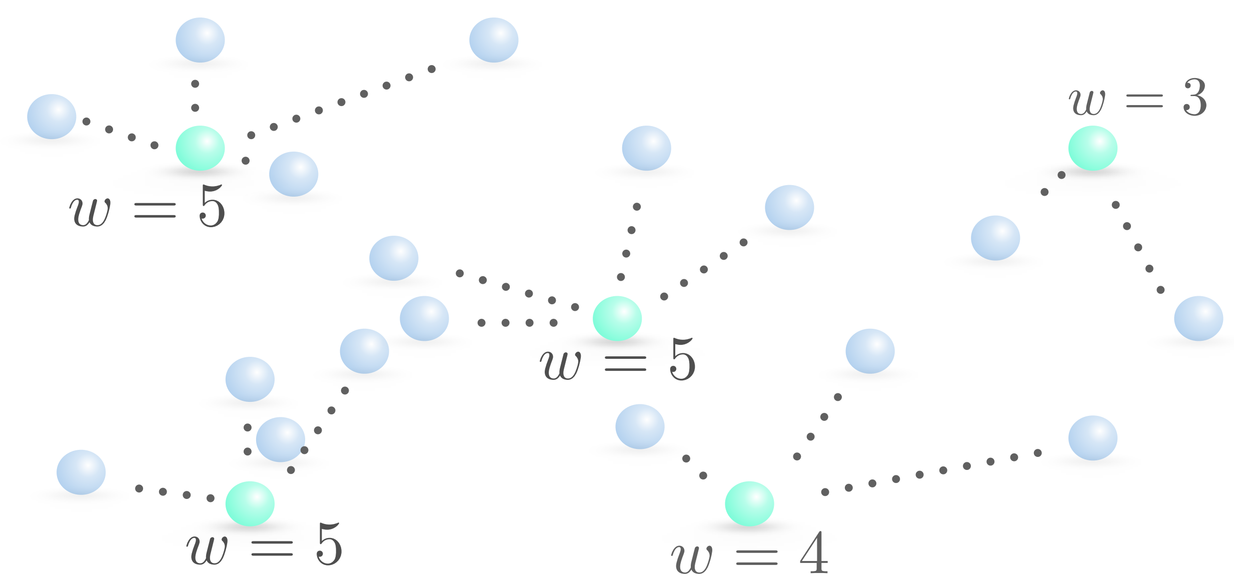
- Composable Core-set



How can we solve such problem?

We use to main ingredients:

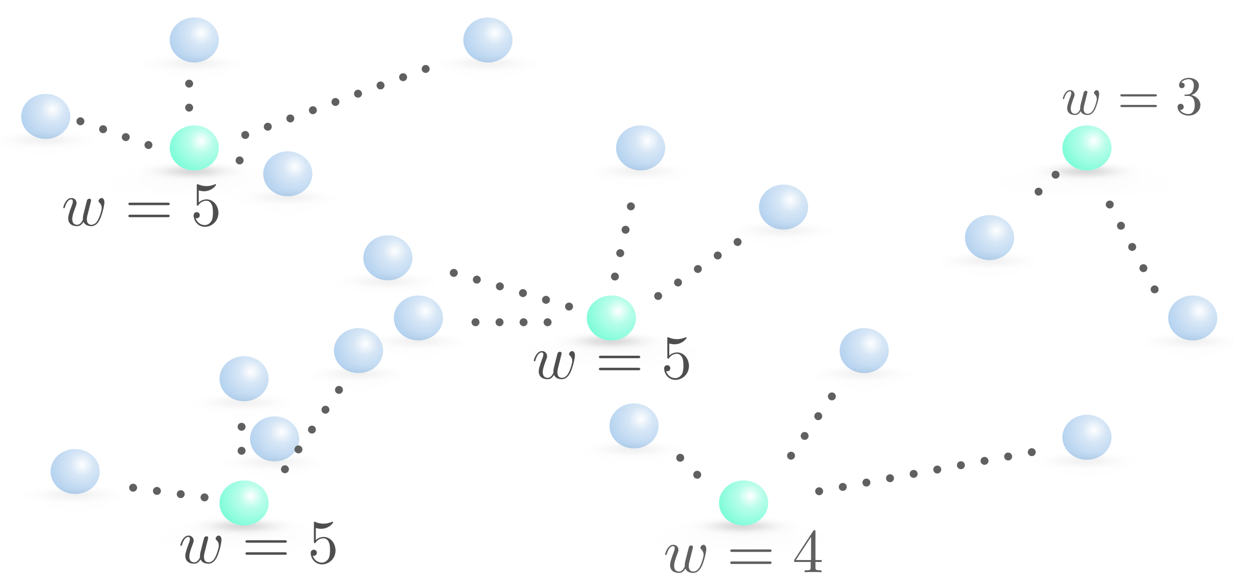
- Composable Core-set



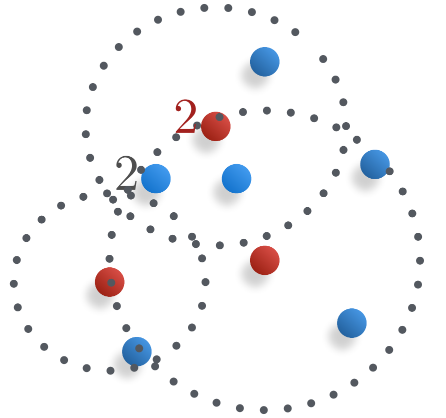
How can we solve such problem?

We use to main ingredients:

- Composable Core-set



- Transform an unbalanced solution in a capacitated one



Composable core-set

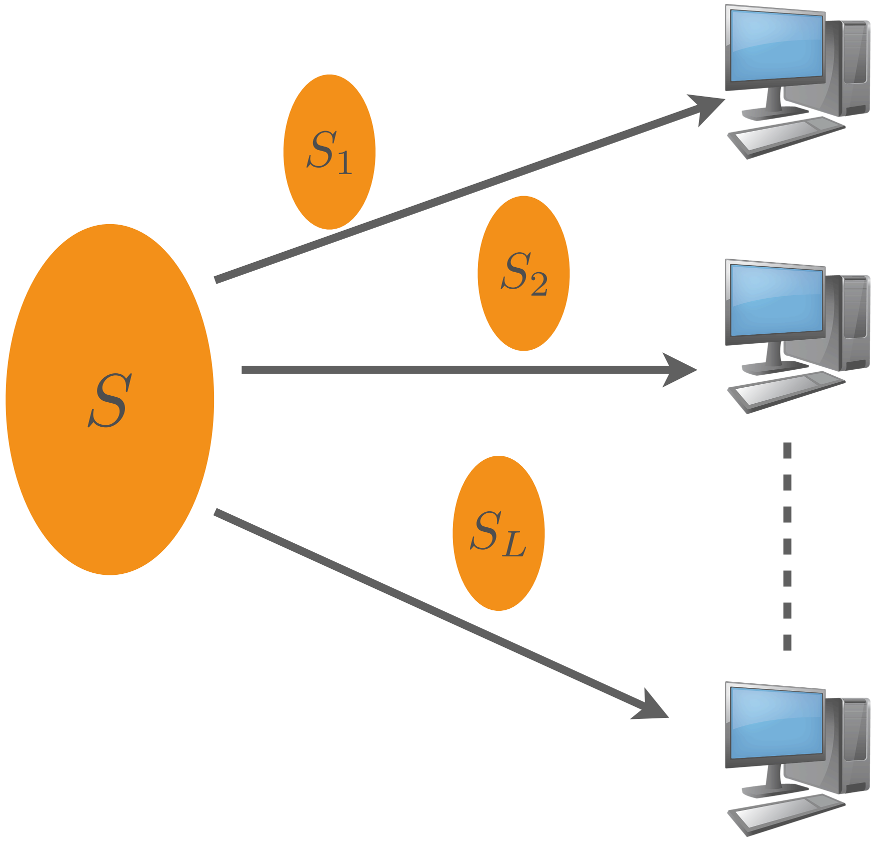
Let f be a function defined for a subset of Δ. A function $c(\Delta)$ is an approximate composable core set if

$$f(c(S_1) \cup \cdots \cup c(S_L)) \approx f(S_1 \cup \cdots \cup S_L)$$

Composable core-set

Let f be a function defined for a subset of Δ. A function $c(\Delta)$ is an approximate composable core set if

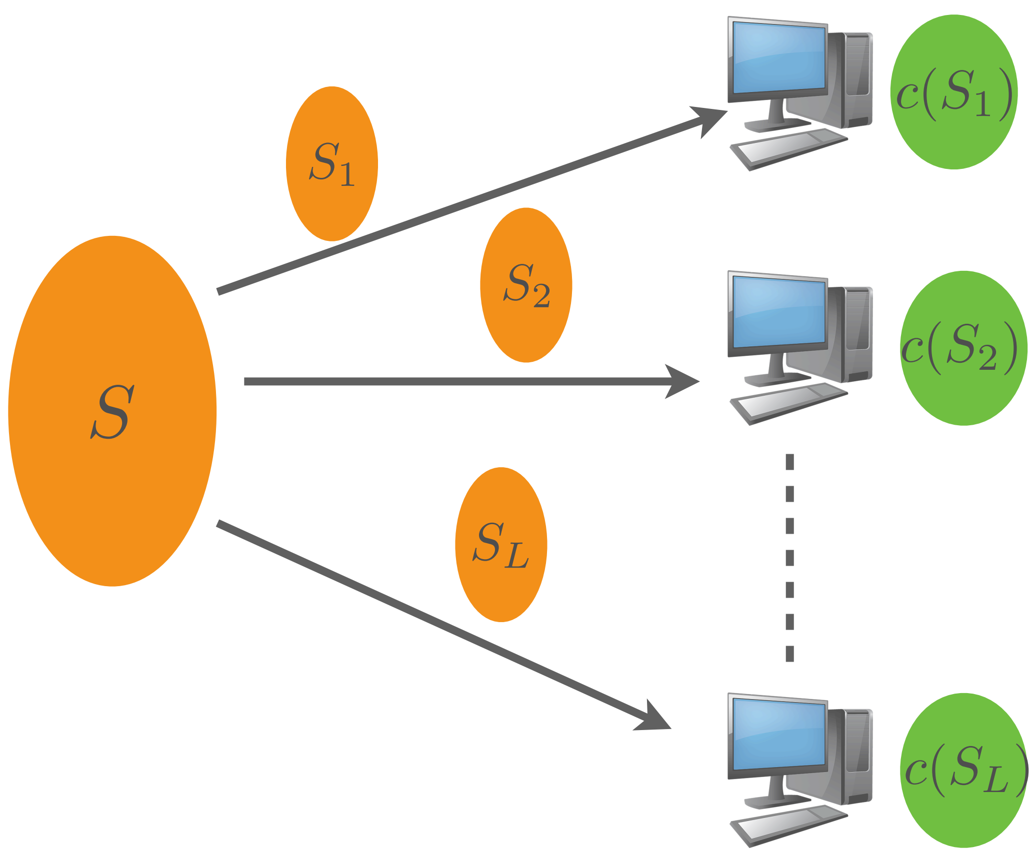
$$f(c(S_1) \cup \cdots \cup c(S_L)) \approx f(S_1 \cup \cdots \cup S_L)$$



Composable core-set

Let f be a function defined for a subset of Δ. A function $c(\Delta)$ is an approximate composable core set if

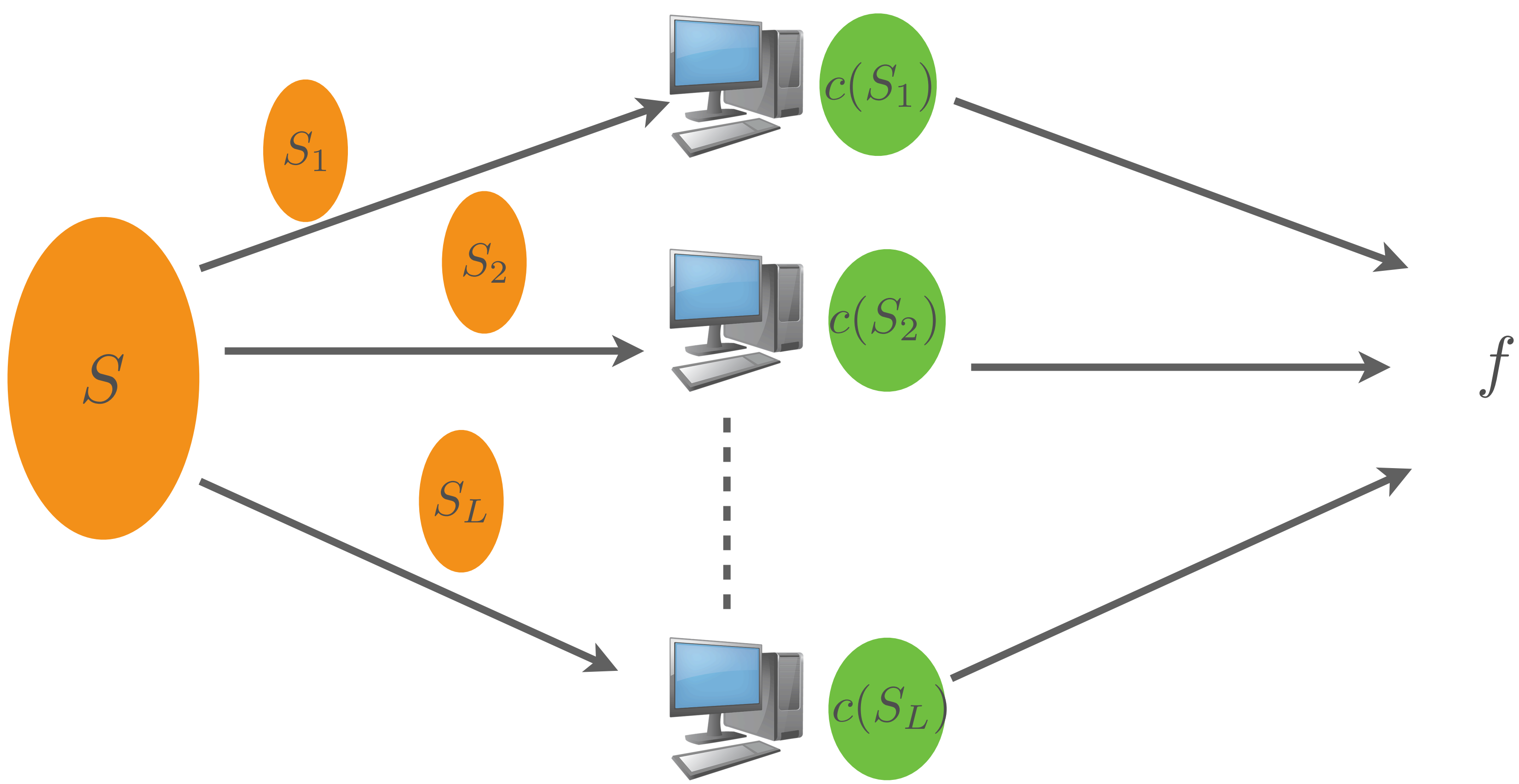
$$f(c(S_1) \cup \cdots \cup c(S_L)) \approx f(S_1 \cup \cdots \cup S_L)$$



Composable core-set

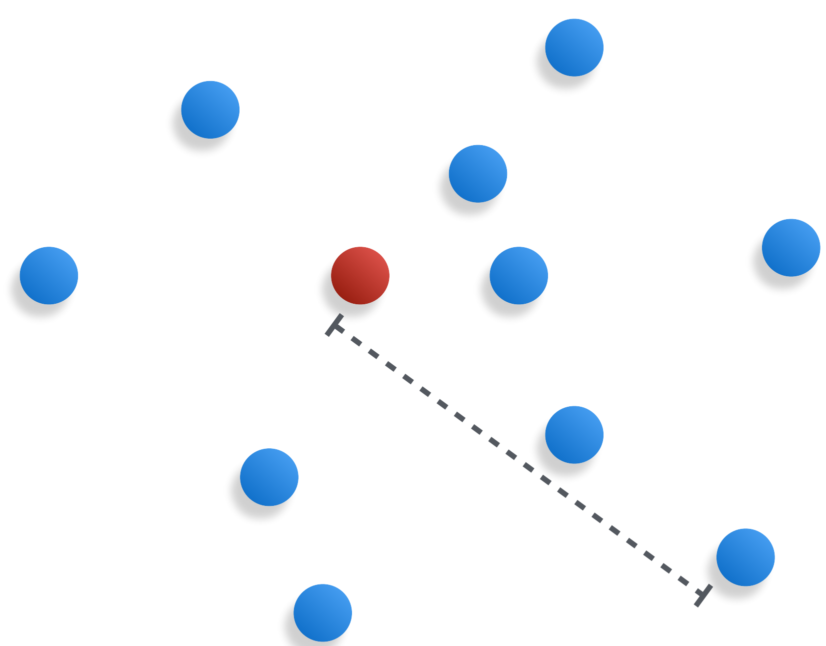
Let f be a function defined for a subset of Δ . A function $c(\Delta)$ is an approximate composable core set if

$$f(c(S_1) \cup \cdots \cup c(S_L)) \approx f(S_1 \cup \cdots \cup S_L)$$



Composable core-set for k-clustering

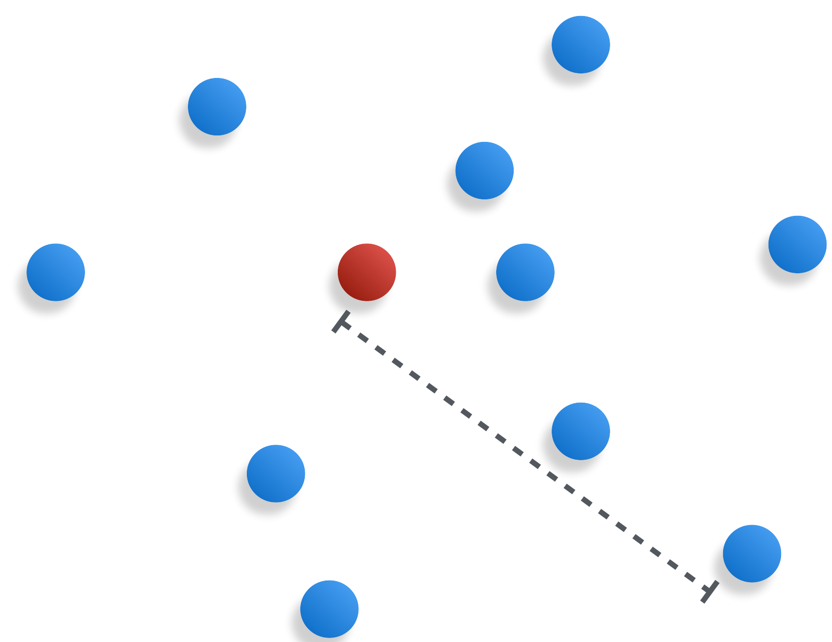
Many ϵ-coresets for clustering problems are composable coresets.



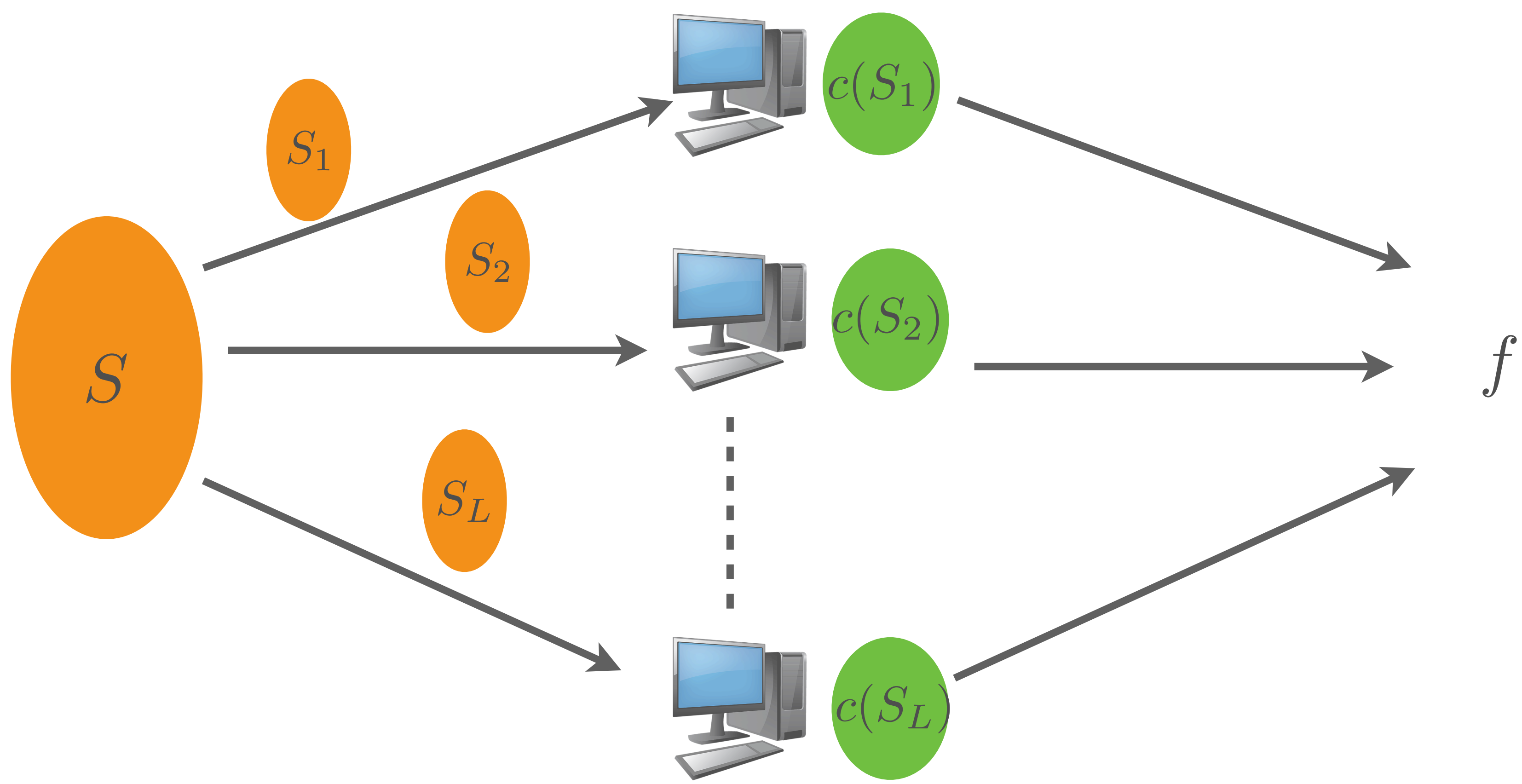
k - center

k- center

$$\phi(X, C) = \max_{x \in X} \min_{c \in C} d(x, c)$$



k-center

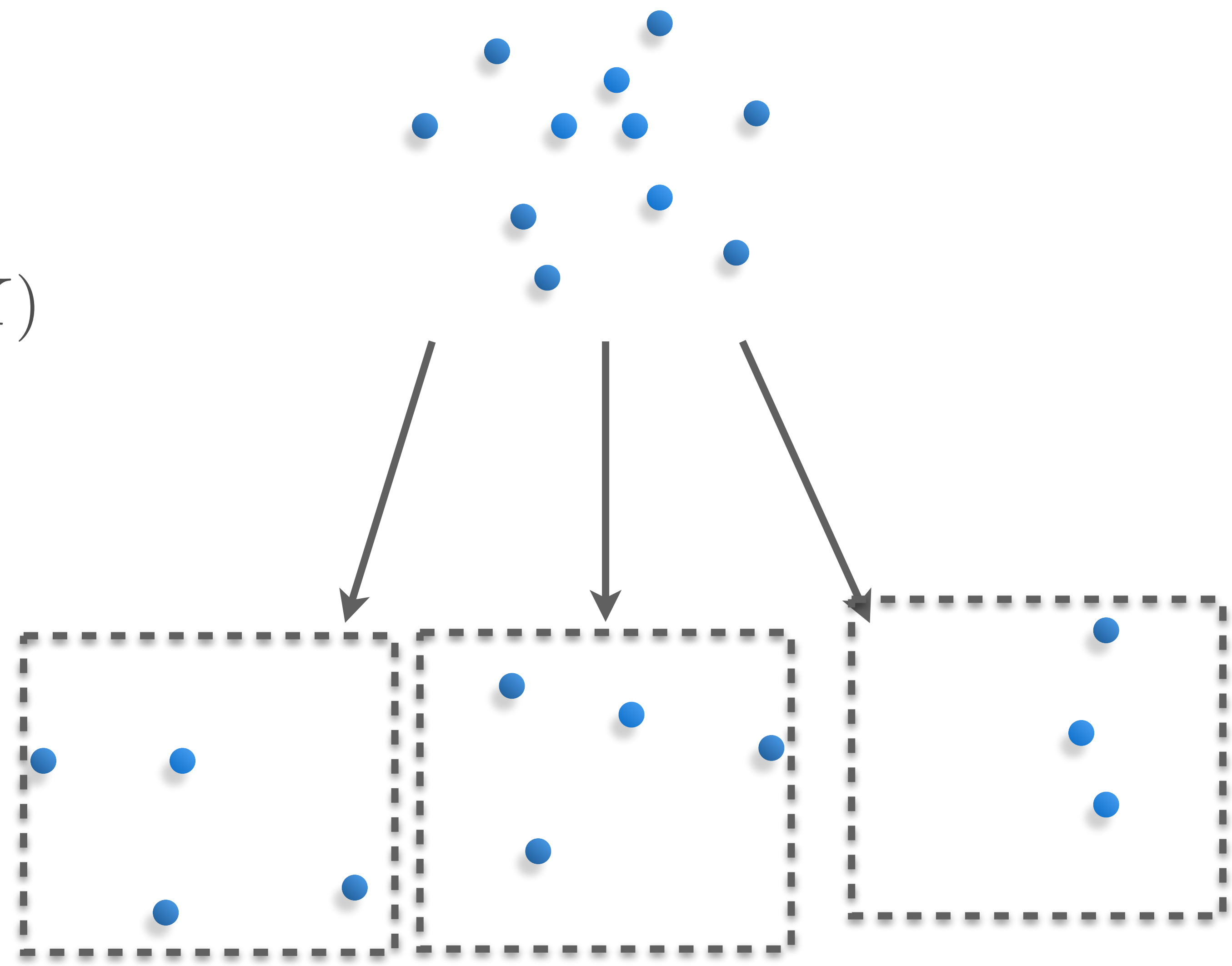


$$\phi(c(X_1) \cup \cdots \cup c(X_L)) \leq 4\phi(X)$$

k-center

Lemma

$$\phi(c(X_1) \cup \cdots \cup c(X_L)) \leq 4\phi(X)$$



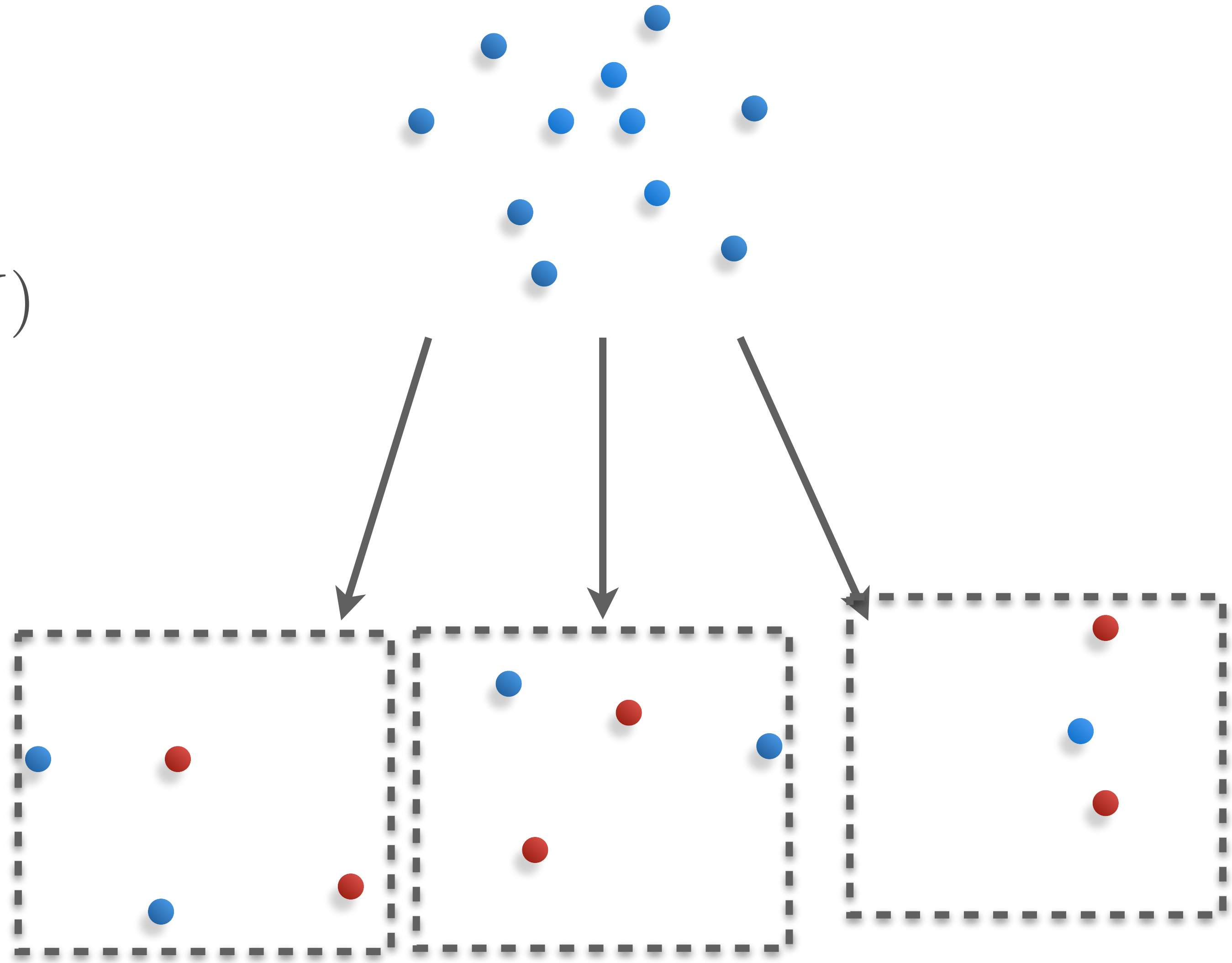
k-center

Lemma

$$\phi(c(X_1) \cup \cdots \cup c(X_L)) \leq 4\phi(X)$$

Proof

Solve k-center independently



k-center

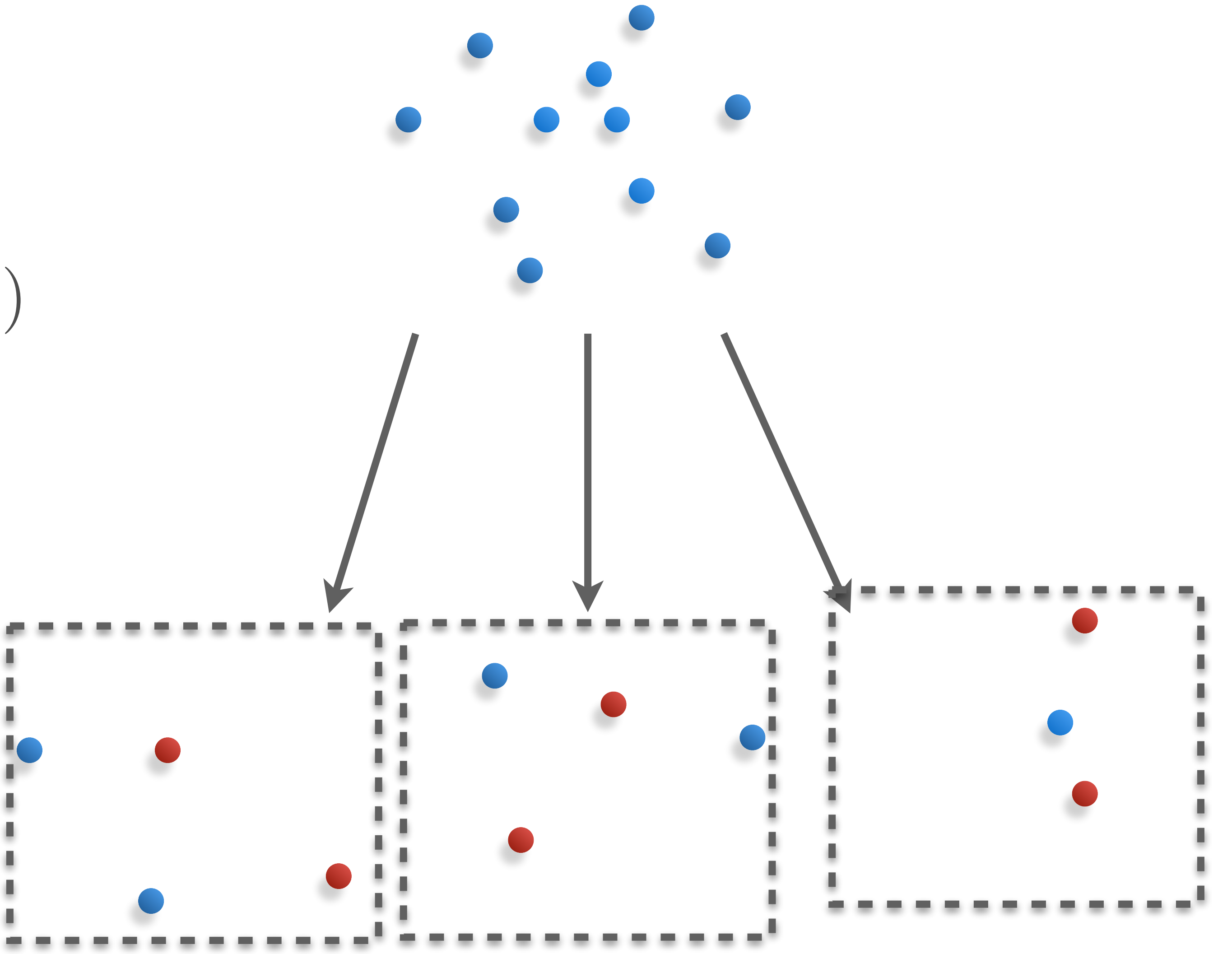
Lemma

$$\phi(c(X_1) \cup \cdots \cup c(X_L)) \leq 4\phi(X)$$

Proof

Solve k-center independently

Cost of each X_i instance is smaller than twice global cost



k-center

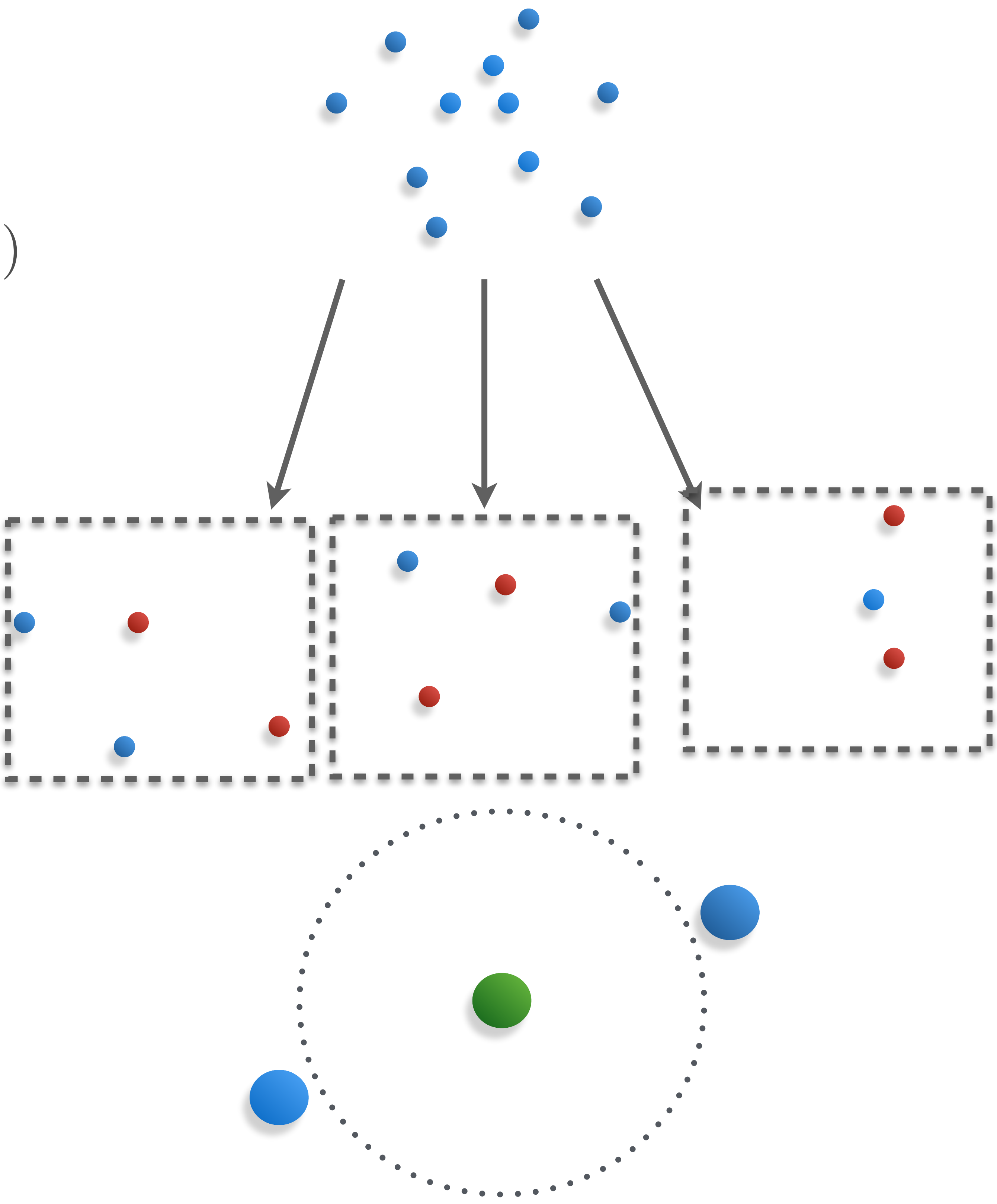
Lemma

$$\phi(c(X_1) \cup \cdots \cup c(X_L)) \leq 4\phi(X)$$

Proof

Solve k-center independently

Cost of each X_i instance is smaller than twice global cost



k-center

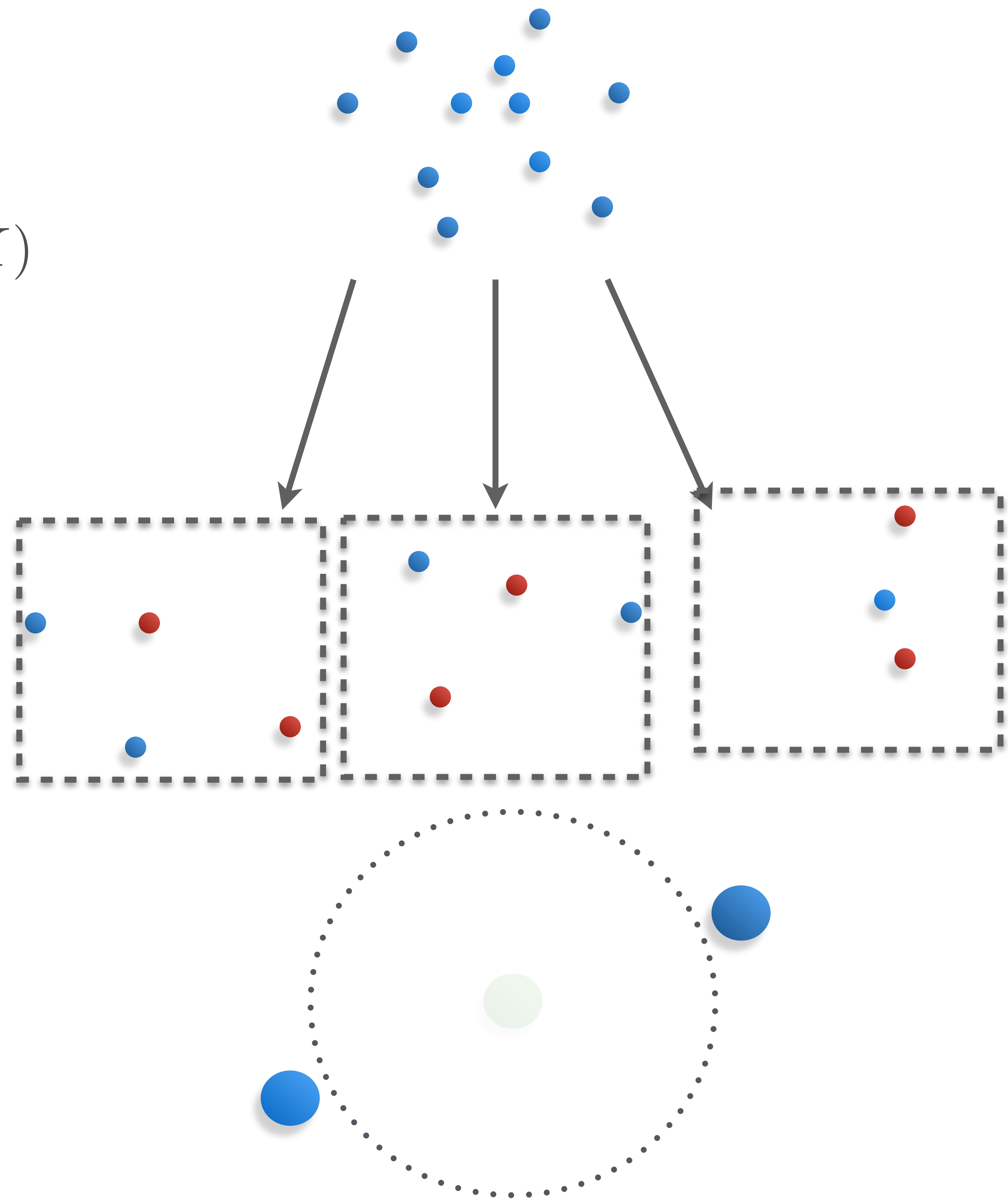
Lemma

$$\phi(c(X_1) \cup \cdots \cup c(X_L)) \leq 4\phi(X)$$

Proof

Solve k-center independently

Cost of each X_i instance is smaller than twice global cost



k-center

Lemma

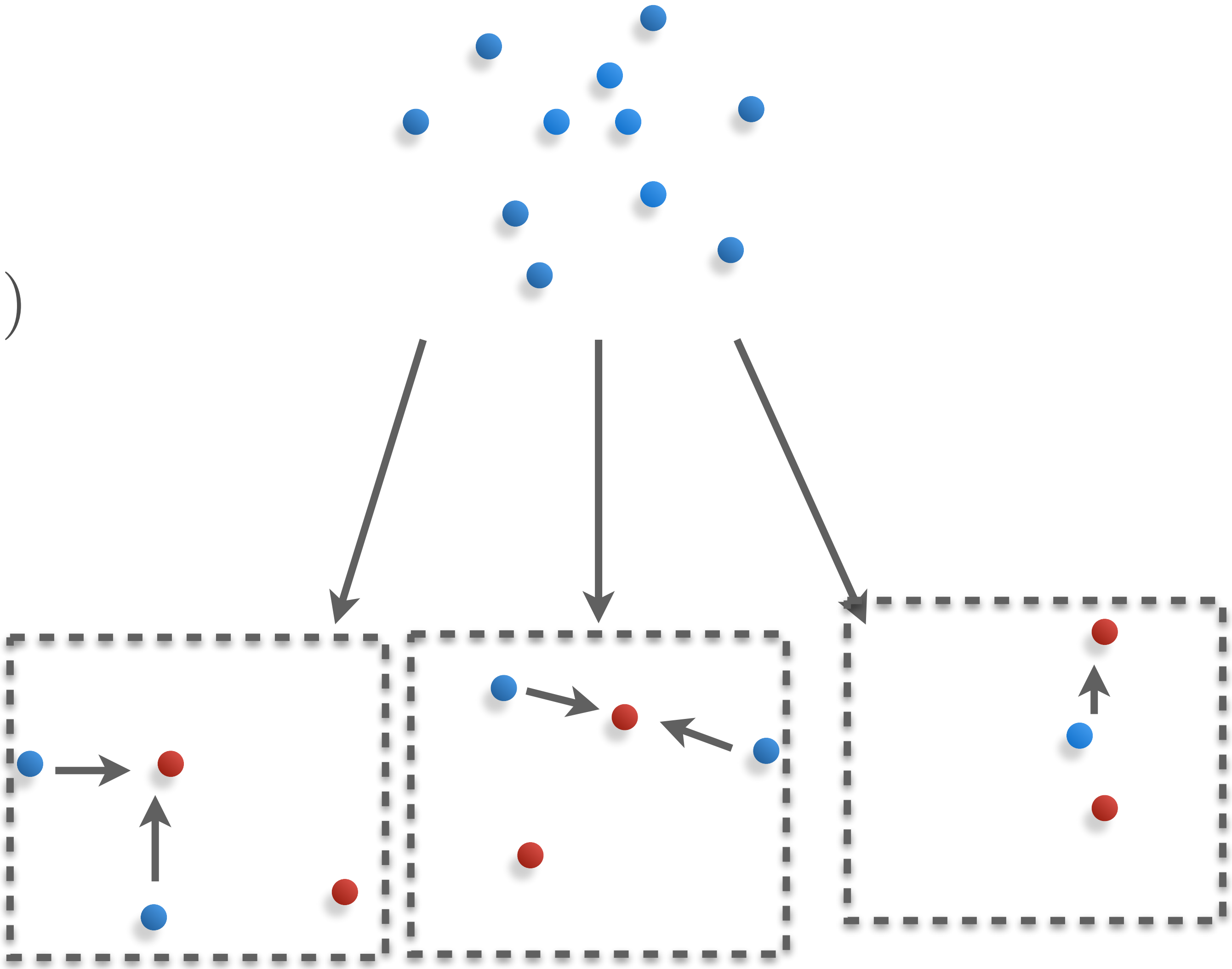
$$\phi(c(X_1) \cup \cdots \cup c(X_L)) \leq 4\phi(X)$$

Proof

Solve k-center independently

Cost of each X_i instance is smaller than twice global cost

Map points to center at cost $2\phi(X)$

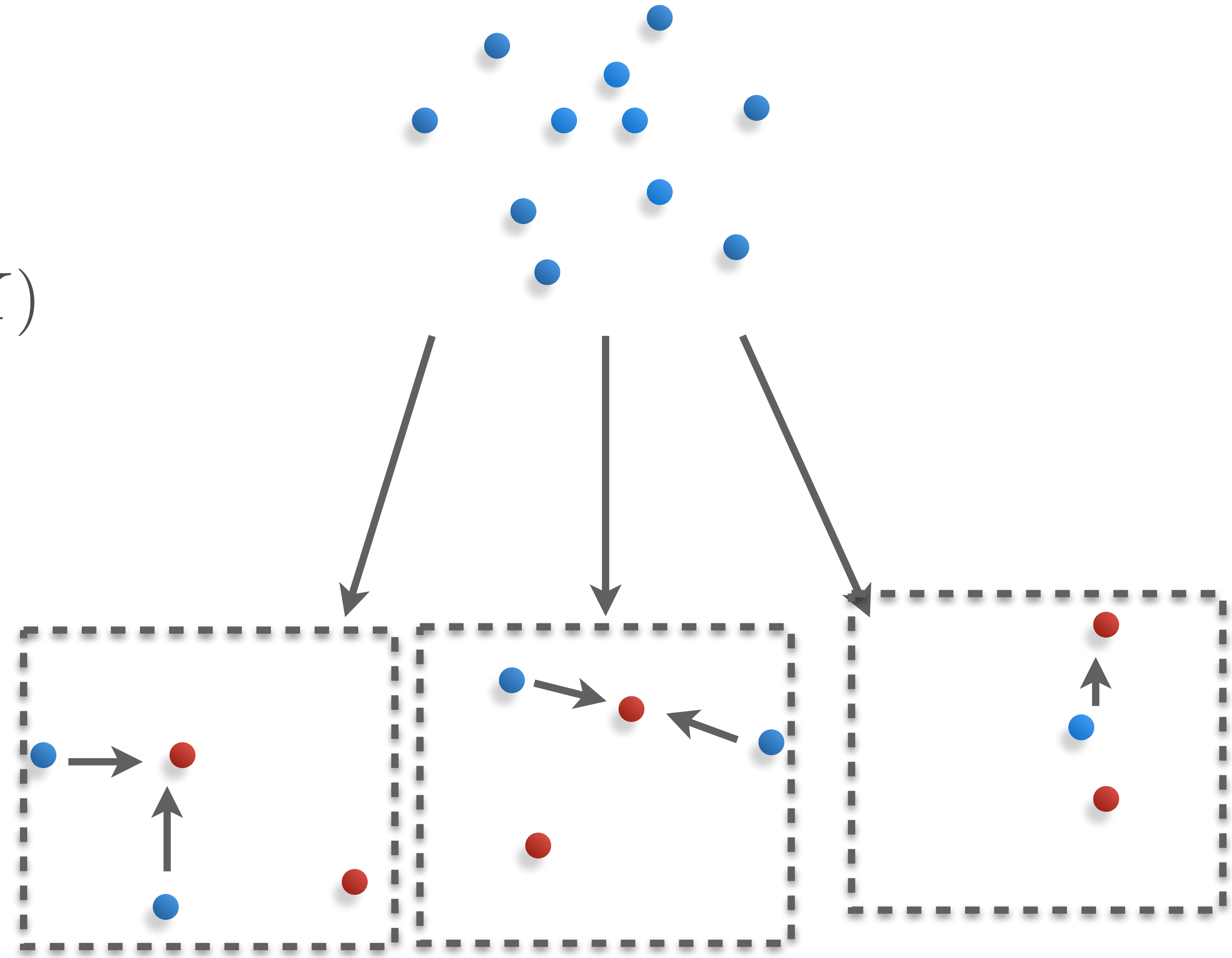


k-center

Lemma

$$\phi(c(X_1) \cup \cdots \cup c(X_L)) \leq 4\phi(X)$$

Proof



Solve k-center independently

Cost of each X_i instance is smaller than twice global cost

Map points to center at cost $2\phi(X)$

Cost of k-center on center is at most $2\,\phi(X)$

k-center

Lemma

$$\phi(c(X_1) \cup \cdots \cup c(X_L)) \leq 4\phi(X)$$

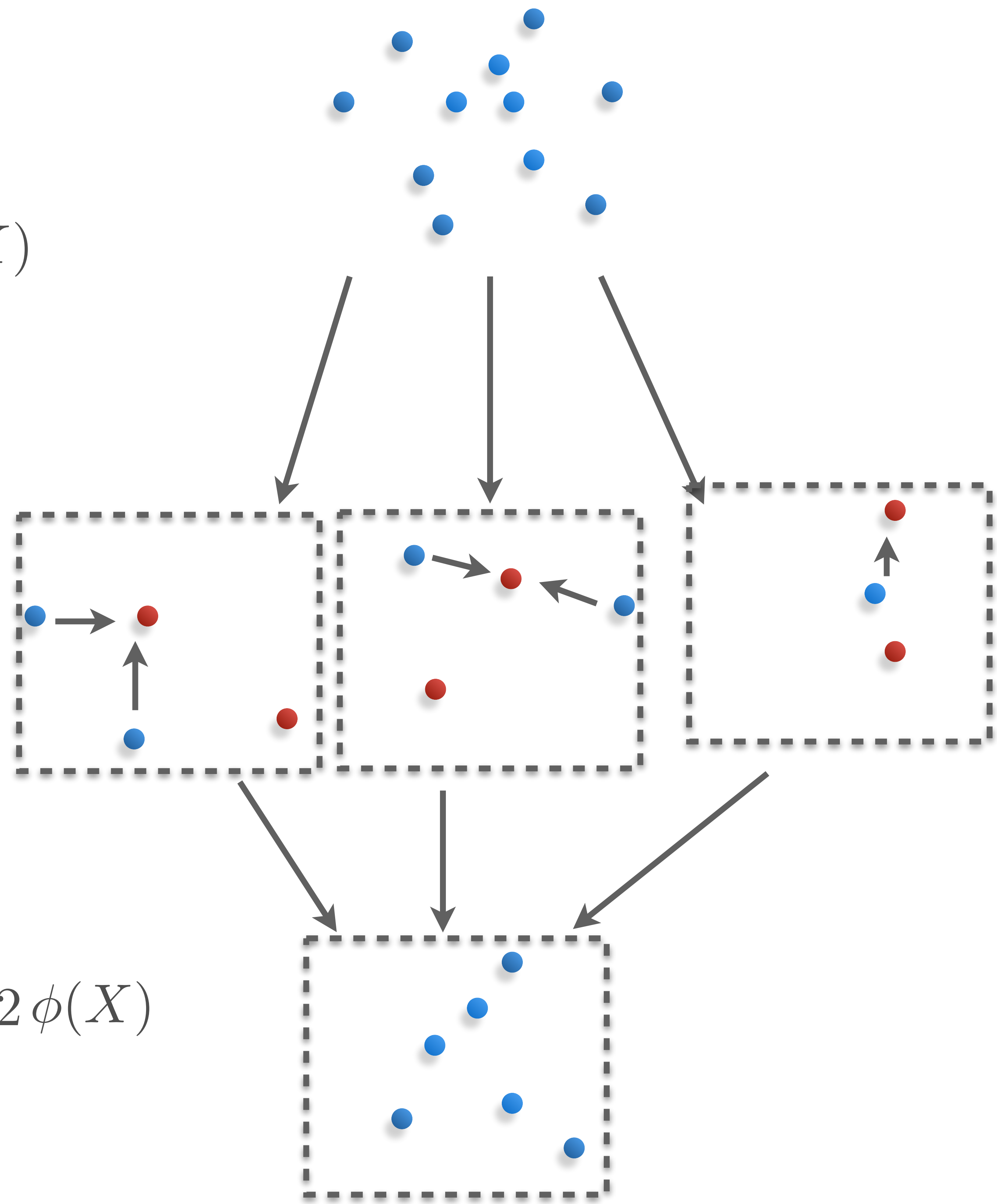
Proof

Solve k-center independently

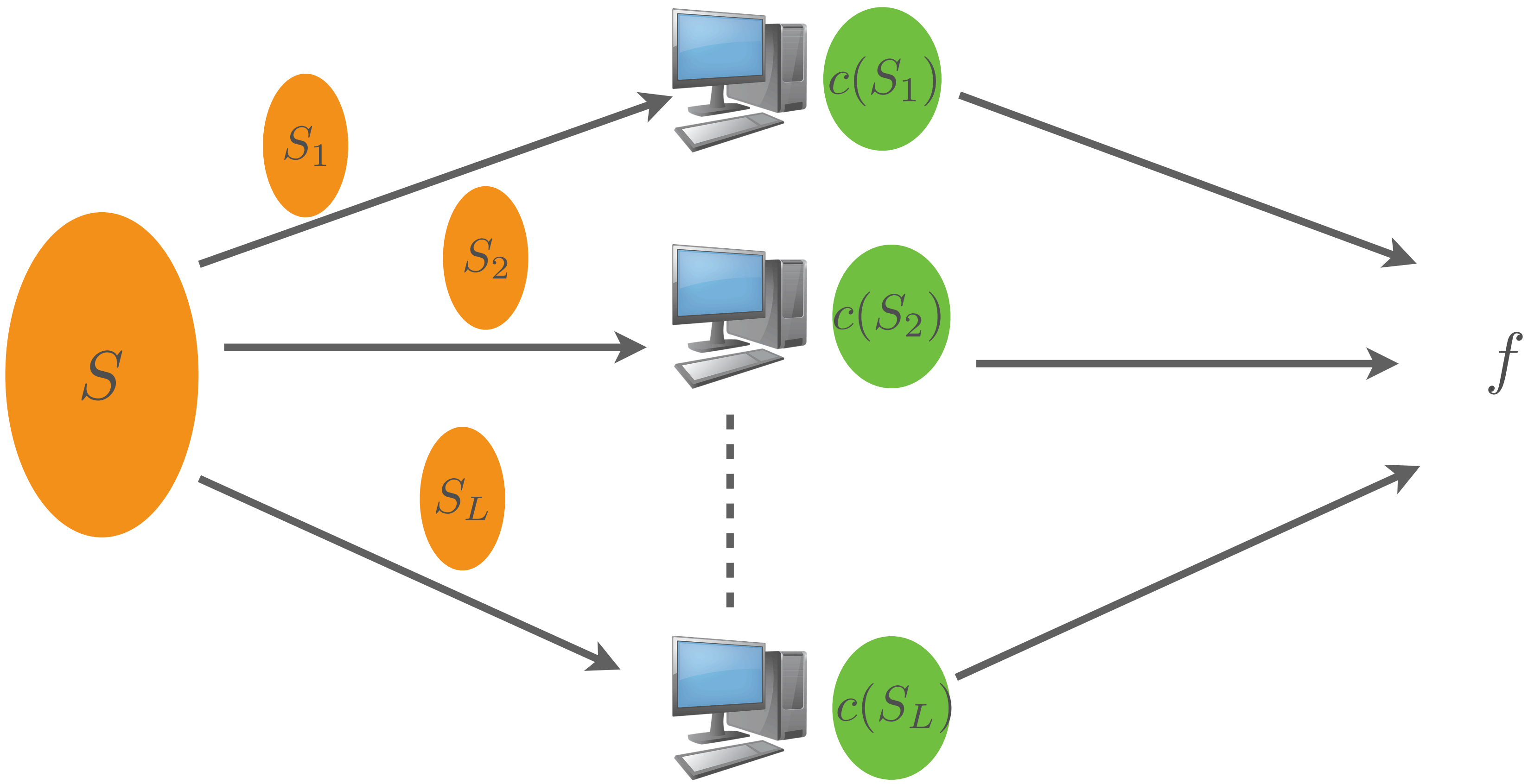
Cost of each X_i instance is smaller than twice global cost

Map points to center at cost $2\phi(X)$

Cost of k-center on center is at most $2\,\phi(X)$



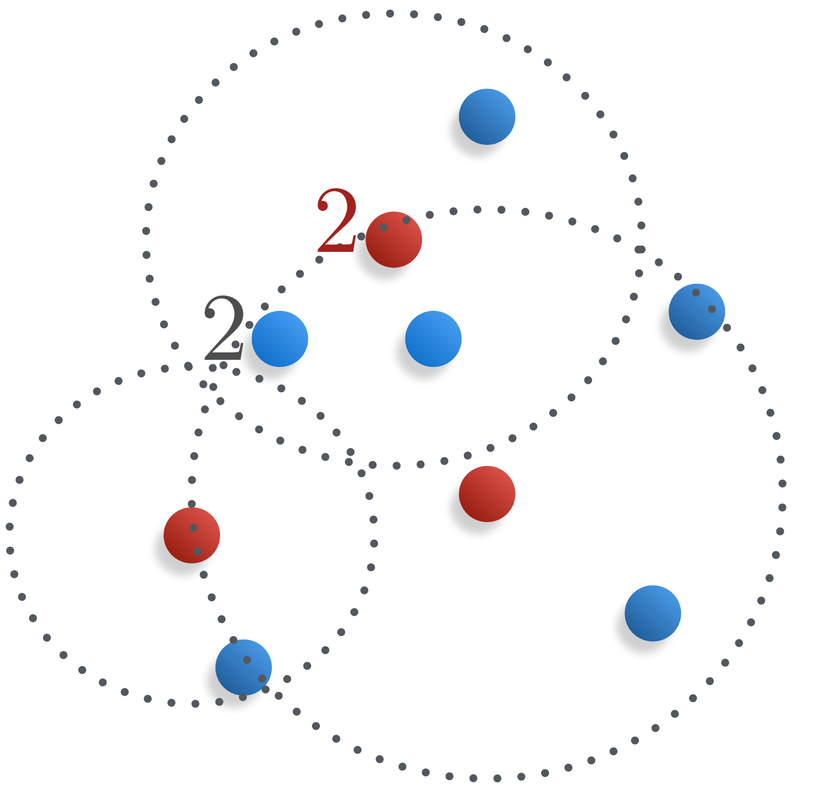
k-center



$$\phi(c(X_1) \cup \cdots \cup c(X_L)) \leq 4\phi(X)$$

Capacitated k - Clustering

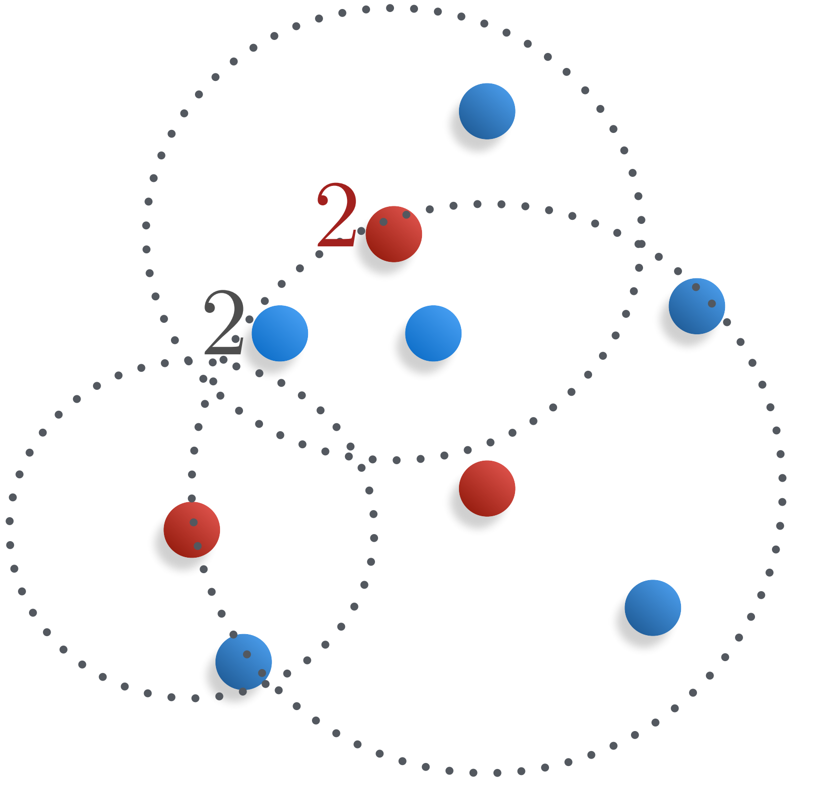
The algorithm has to run using space proportional to the compressed instance.



Capacitated k - Clustering

The algorithm has to run using space proportional to the compressed instance.

Use sequential algorithm in unconstrained setting to get a $O(1)$-approximation

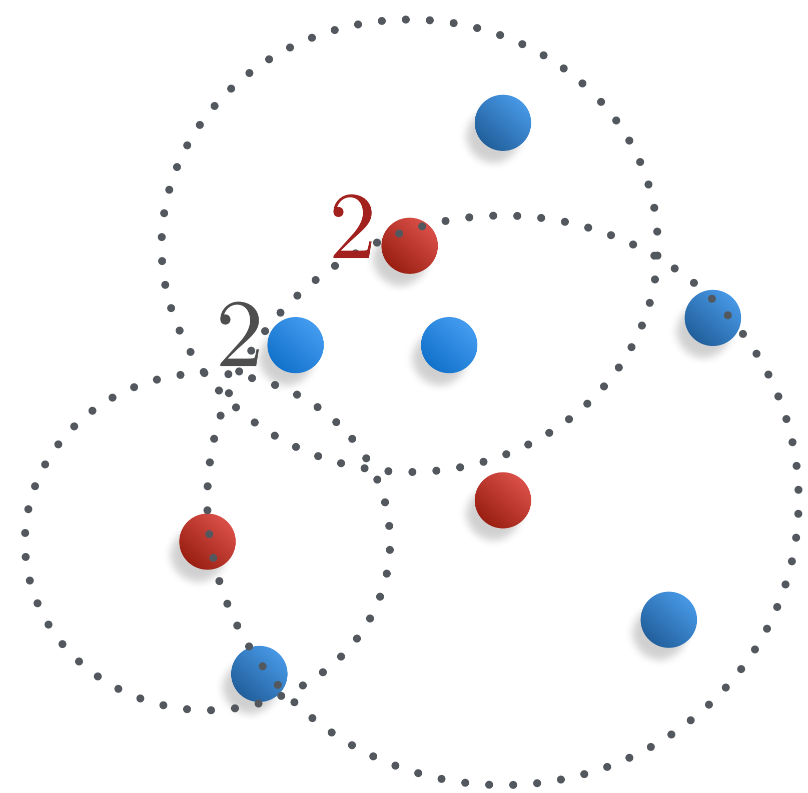


Capacitated k - Clustering

The algorithm has to run using space proportional to the compressed instance.

Use sequential algorithm in unconstrained setting to get a $O(1)$-approximation

If a cluster is too large use as additional centers the closest nodes to the center



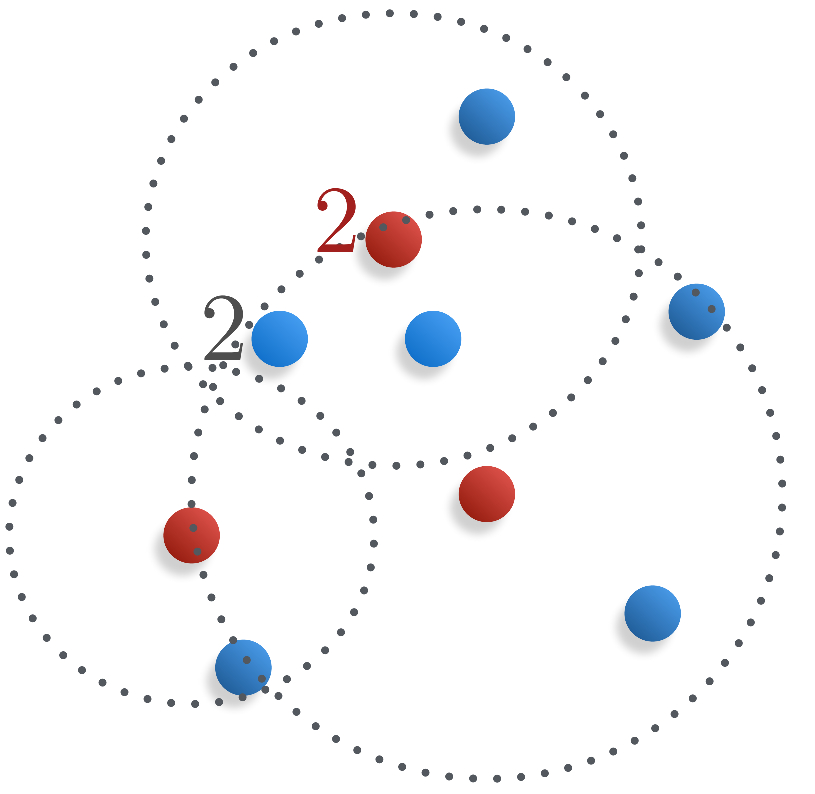
Capacitated k - Clustering

The algorithm has to run using space proportional to the compressed instance.

Use sequential algorithm in unconstrained setting to get a $O(1)$-approximation

If a cluster is too large use as additional centers the closest nodes to the center

Cost of the clustering at most doubles



Capacitated k - Clustering

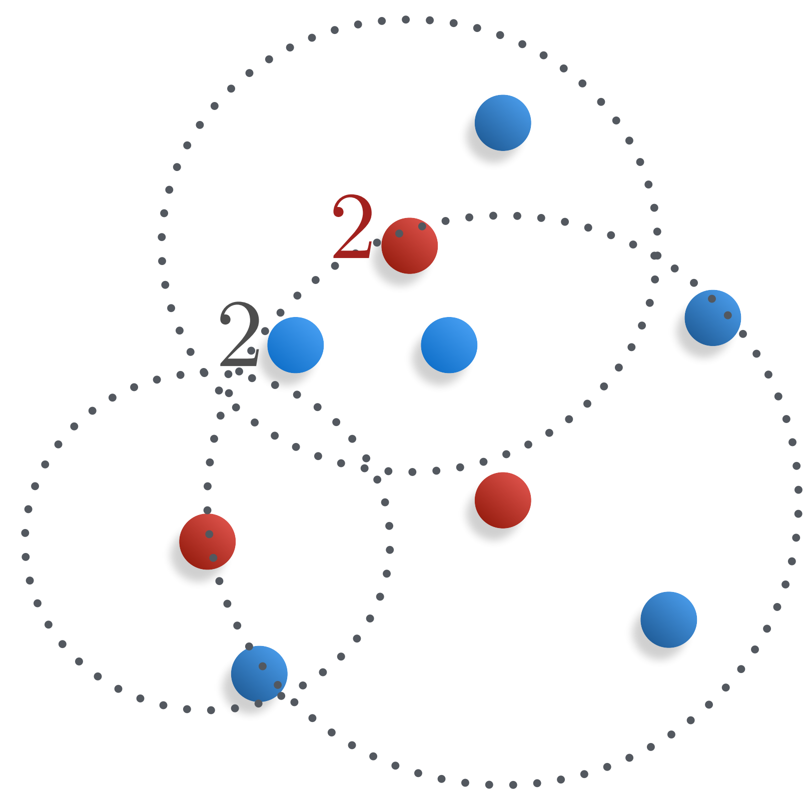
The algorithm has to run using space proportional to the compressed instance.

Use sequential algorithm in unconstrained setting to get a $O(1)$-approximation

If a cluster is too large use as additional centers the closest nodes to the center

Cost of the clustering at most doubles

Number of additional clusters: $\sum_C \frac{n_C}{L} = \frac{n}{L} = k$



Capacitated k - Clustering

The algorithm has to run using space proportional to the compressed instance.

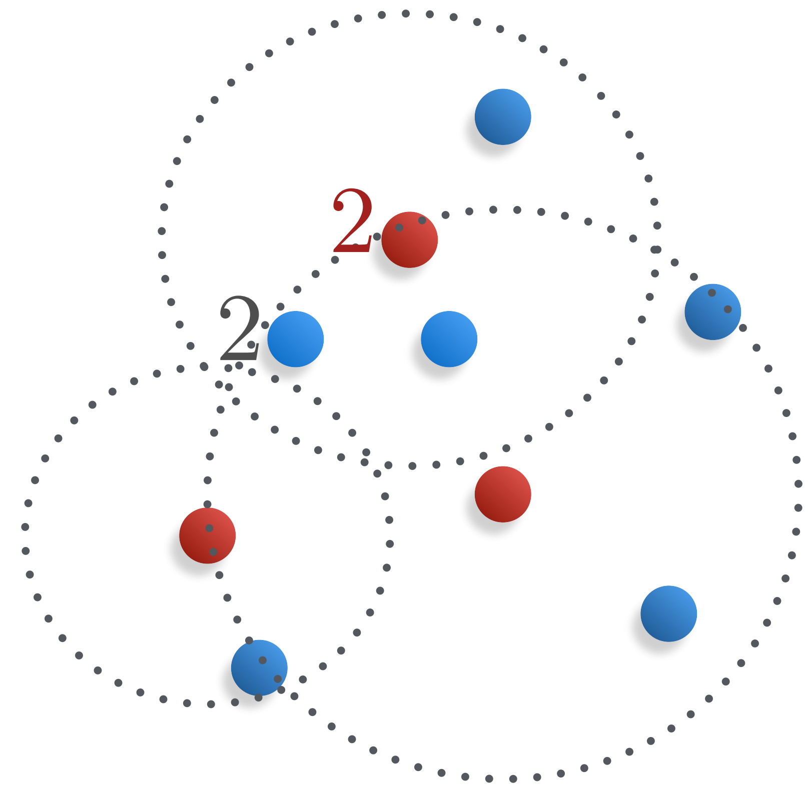
Use sequential algorithm in unconstrained setting to get a $O(1)$-approximation

If a cluster is too large use as additional centers the closest nodes to the center

Cost of the clustering at most doubles

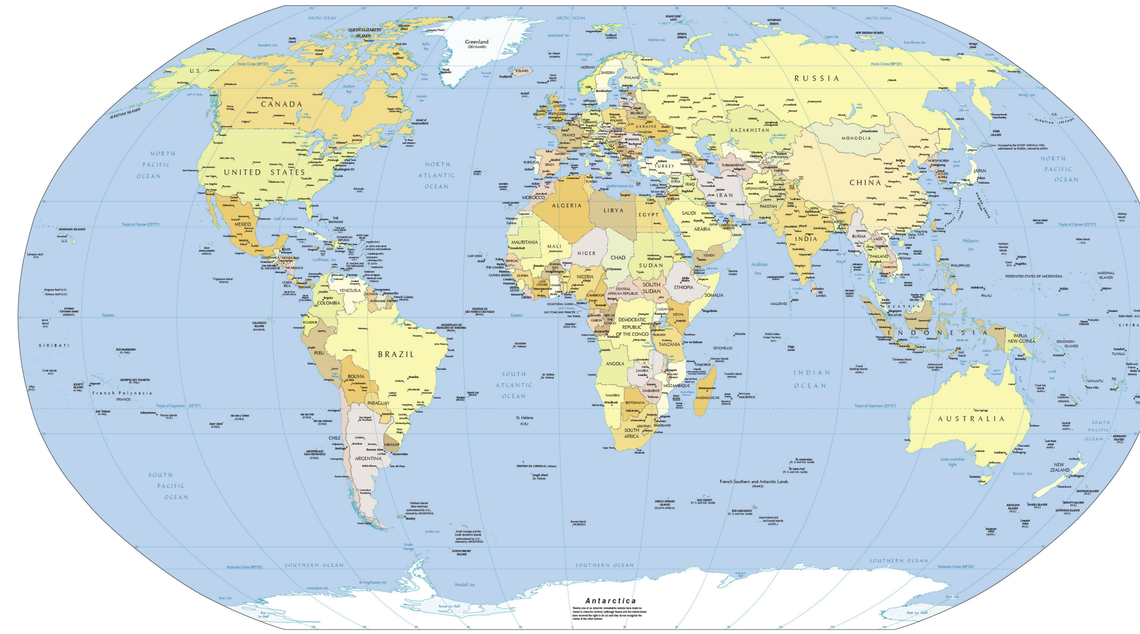
Number of additional clusters: $\sum_C \frac{n_C}{L} = \frac{n}{L} = k$

Bicriteria $(O(1), 2)$ algorithm



Experiments

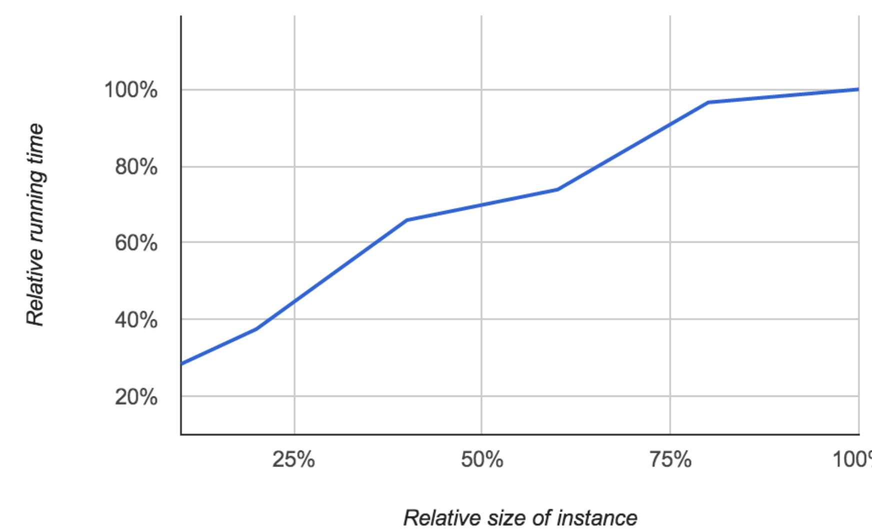
[Distributed Balanced Clustering via Mapping Coresets.](#)
MohammadHossein Bateni, Aditya Bhaskara, Silvio Lattanzi, Vahab S. Mirrokni
NIPS 2014: 2591-2599



US graph: N = x0 Millions

World graph: N = x00 Millions

	size of seq. inst.	increase in OPT
US	1/300	**1.52**
World	1/1000	**1.58**

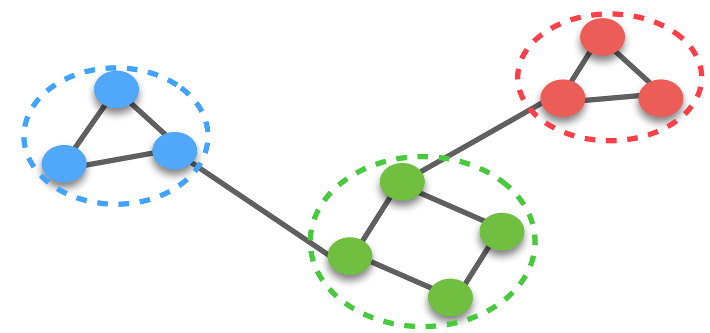


Hierarchical Graph clustering at scale

Density based clustering

Detecting dense structure in the graph is a well-studied problem with many practical applications

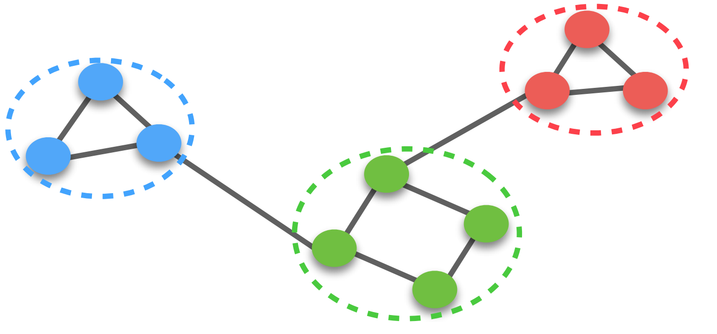
Community detection



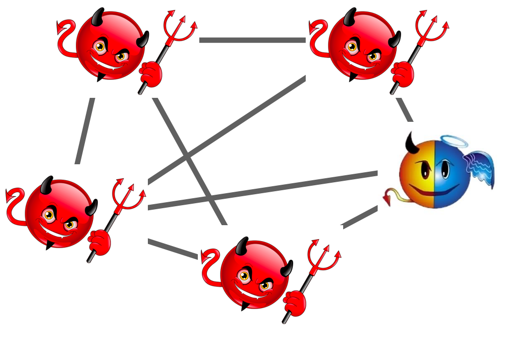
Density based clustering

Detecting dense structure in the graph is a well-studied problem with many practical applications

Community detection



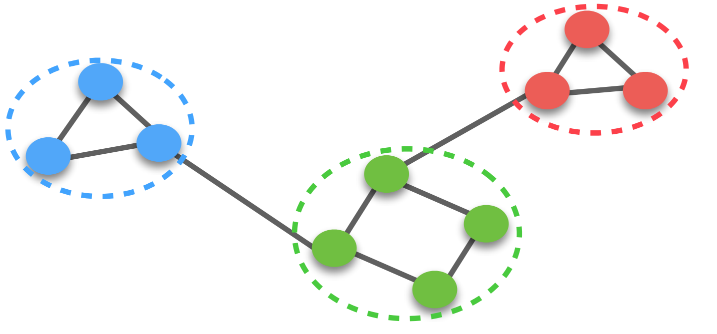
Spam detection



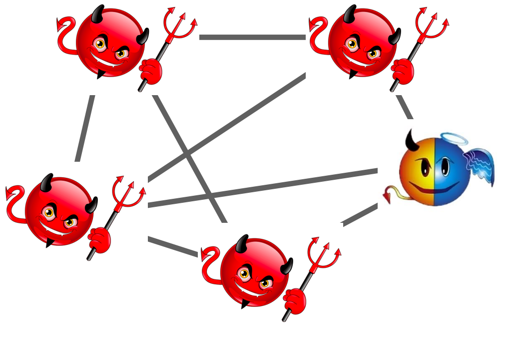
Density based clustering

Detecting dense structure in the graph is a well-studied problem with many practical applications

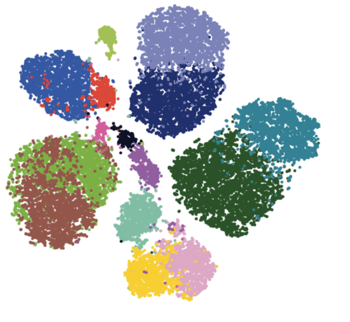
Community detection



Spam detection



Computational biology



…

Minimum versus average degree

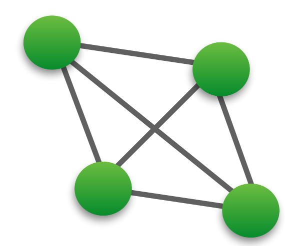
What should we look for?

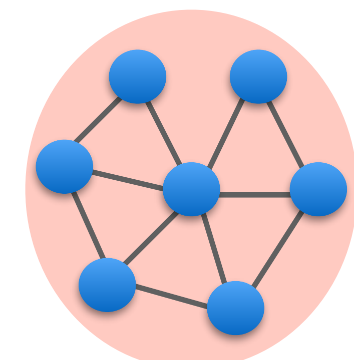
Minimum versus average degree

What should we look for?

A subgraph with high average degree

$$\frac{|E|}{|N|}$$



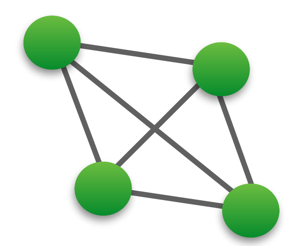


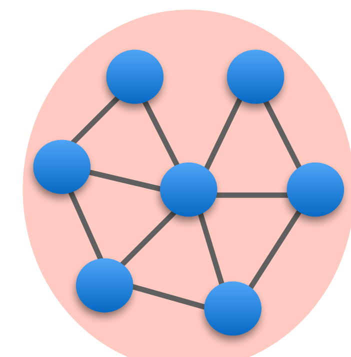
Minimum versus average degree

What should we look for?

A subgraph with high average degree

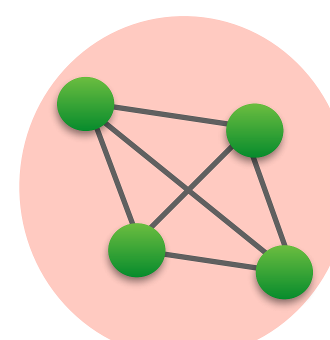
$$\frac{|E|}{|N|}$$

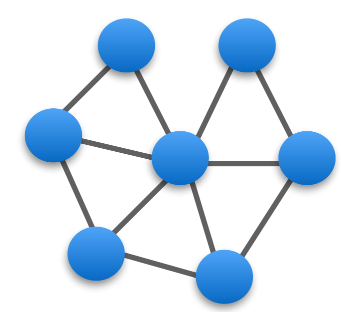




A subgraph with high minimum degree

$$\min d_v$$

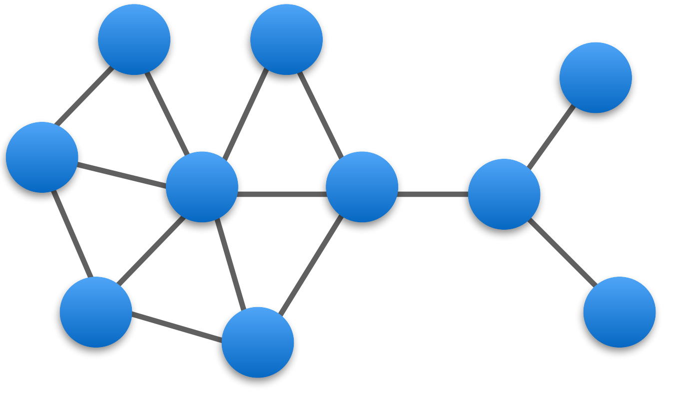




K-core definition

A K-core is a maximal subgraph of minimum degree K

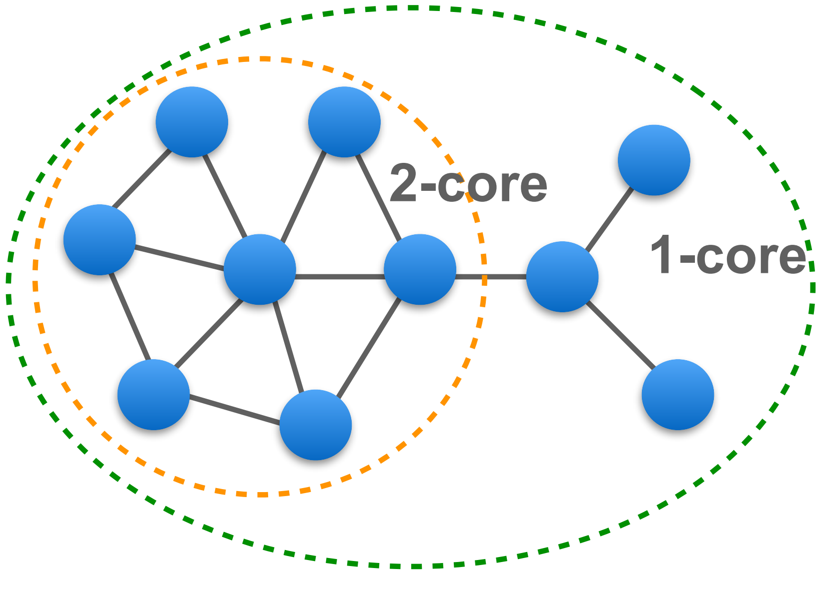
The coreness number of vertex v is maximum K for which v is part of the K-core



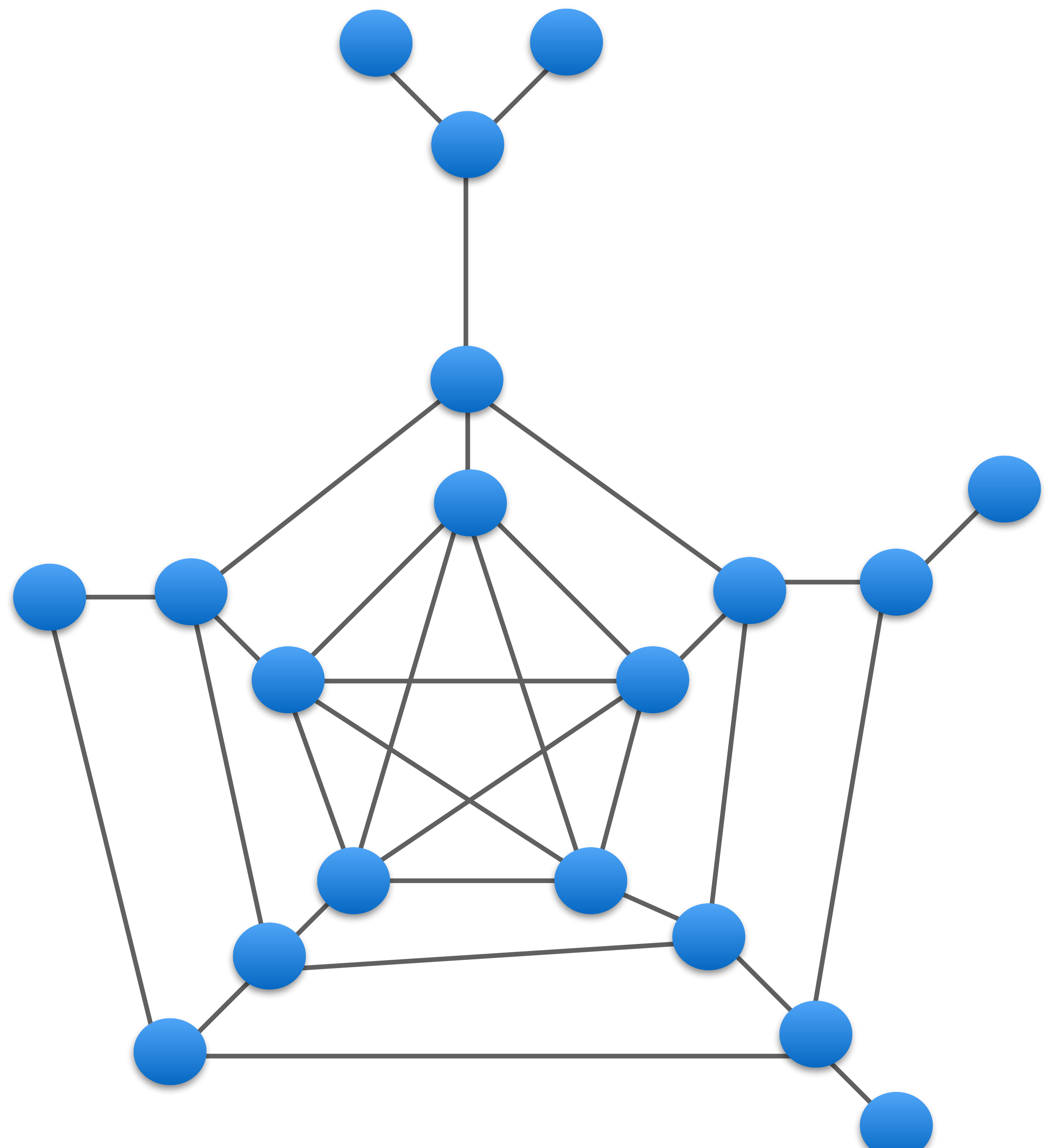
K-core definition

A K-core is a maximal subgraph of minimum degree K

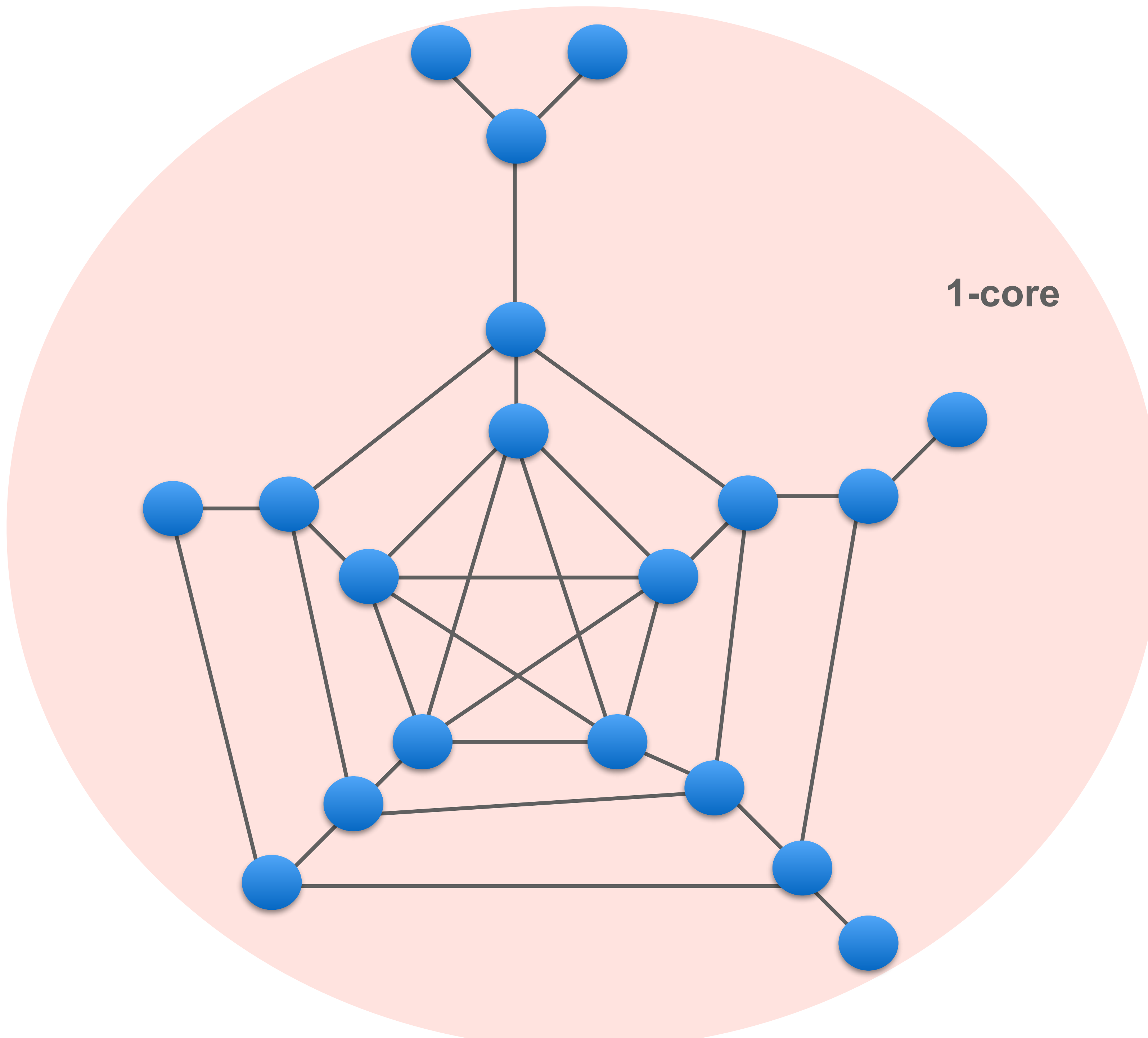
The coreness number of vertex v is maximum K for which v is part of the K-core



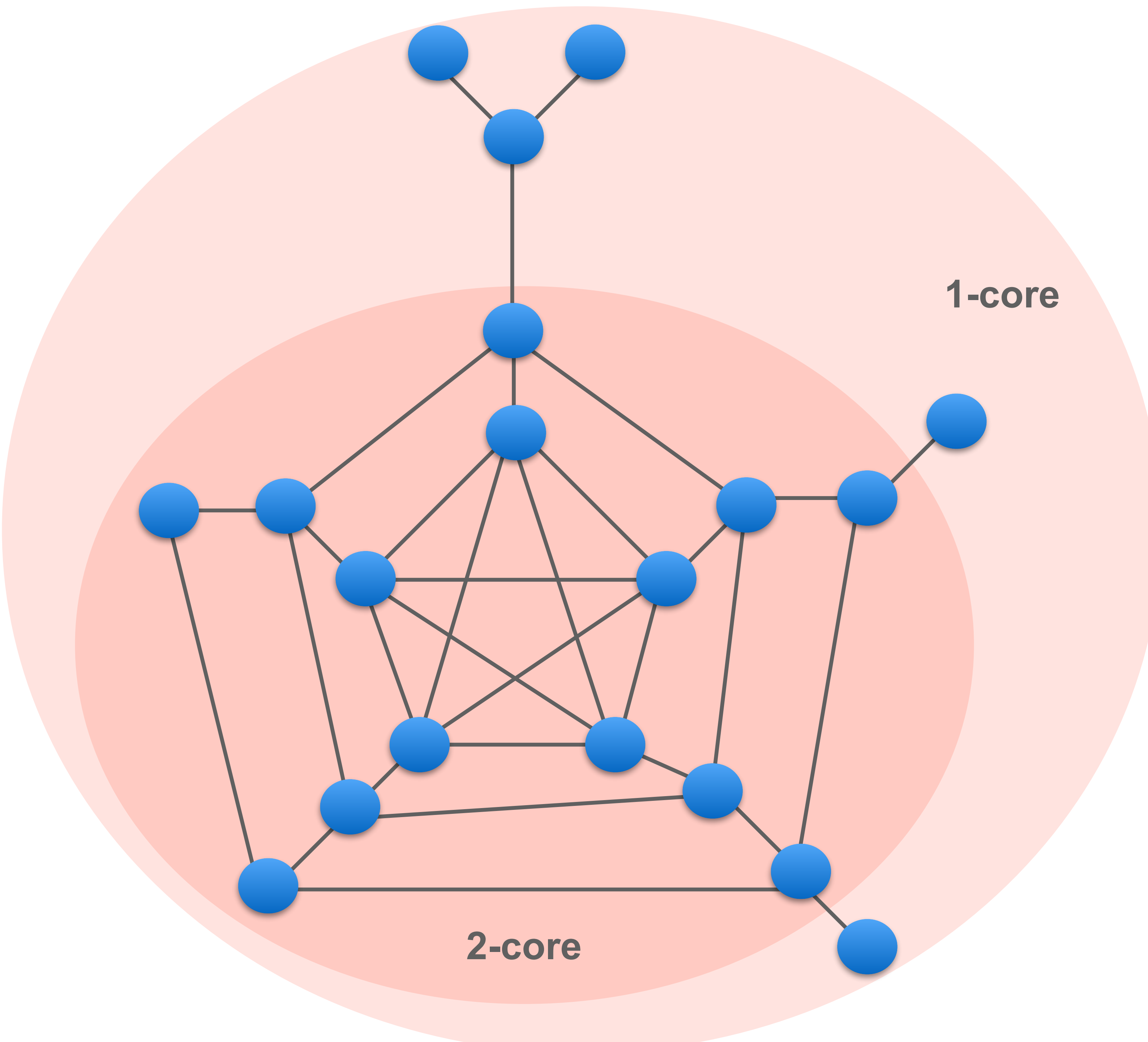
Hierarchical clustering via K-core



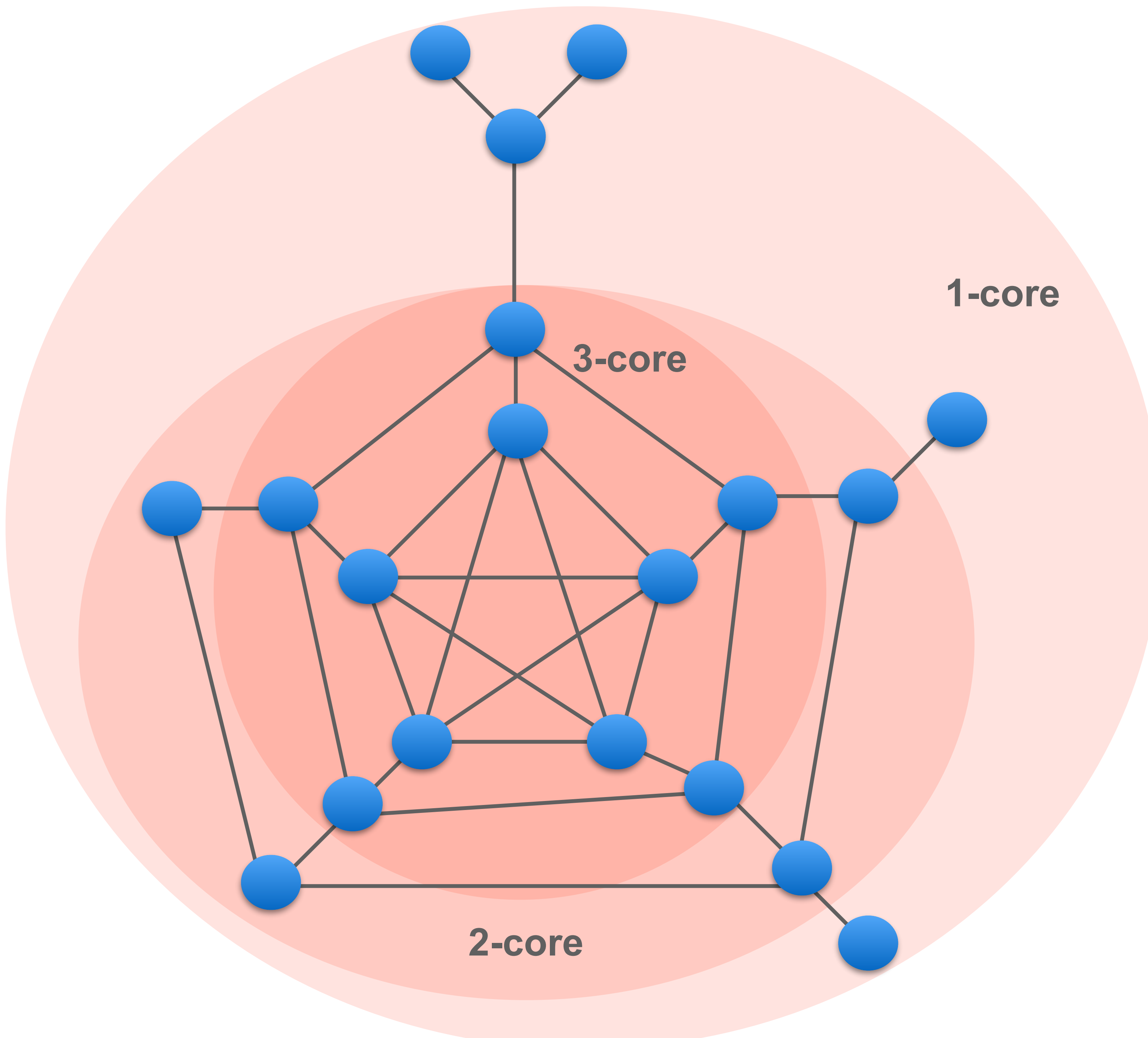
Hierarchical clustering via K-core



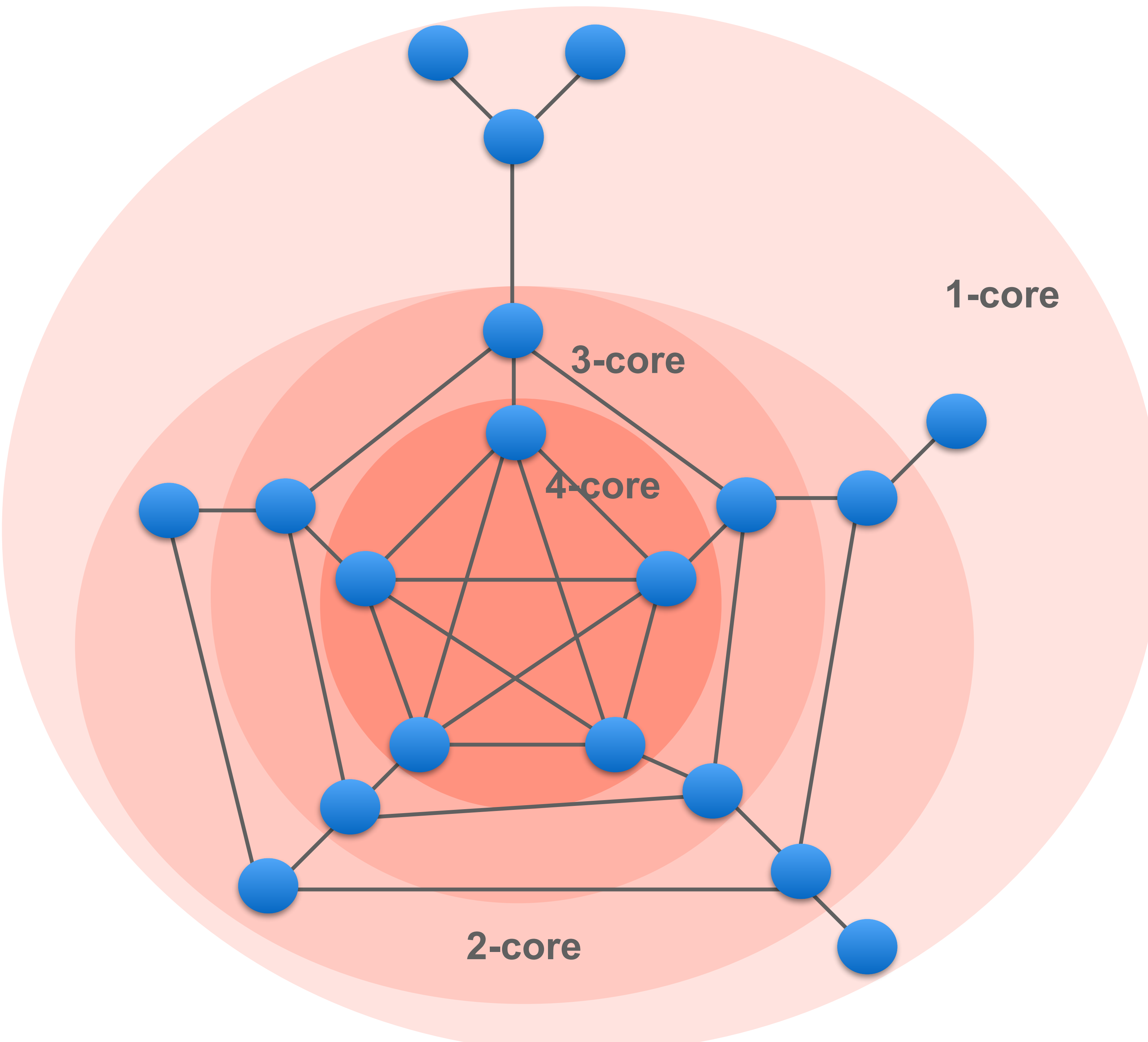
Hierarchical clustering via K-core



Hierarchical clustering via K-core

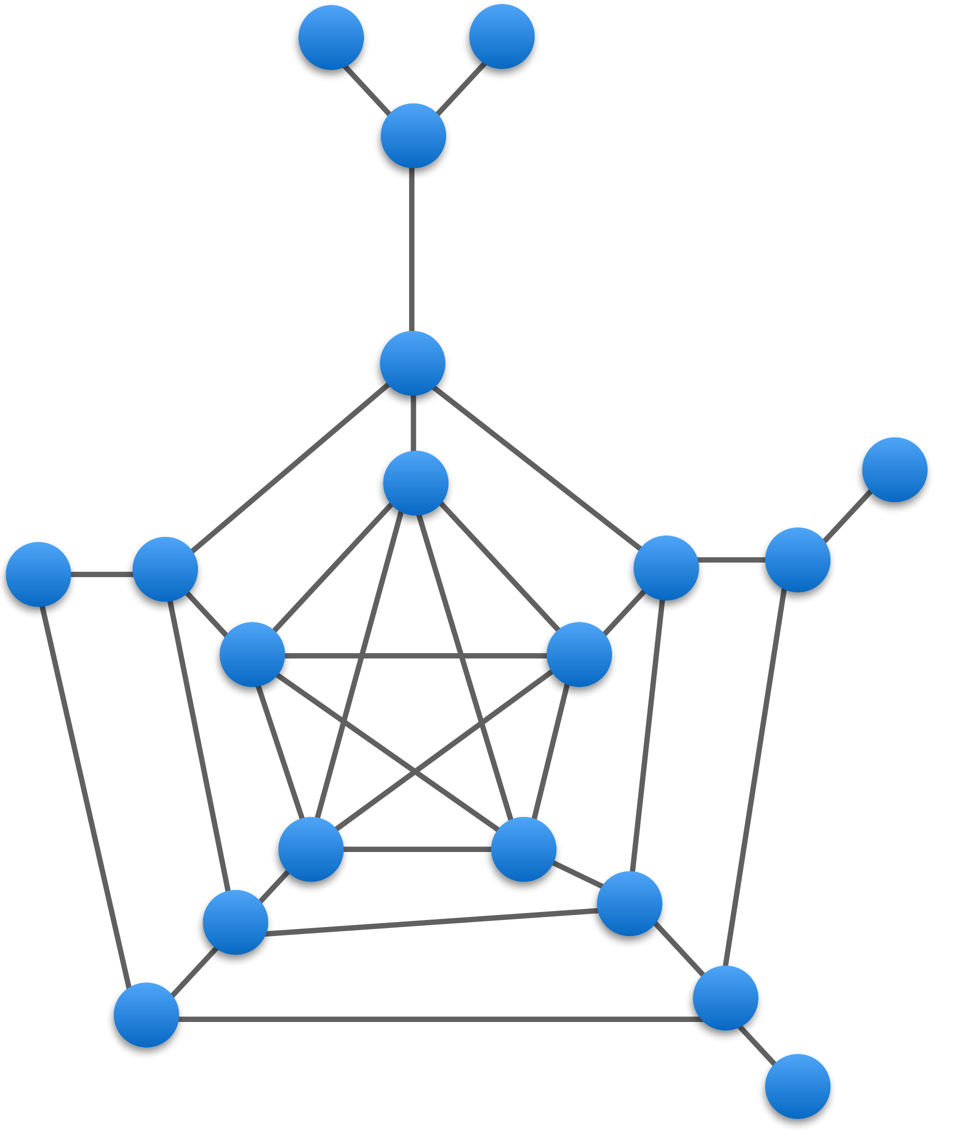


Hierarchical clustering via K-core



Sequential algorithm

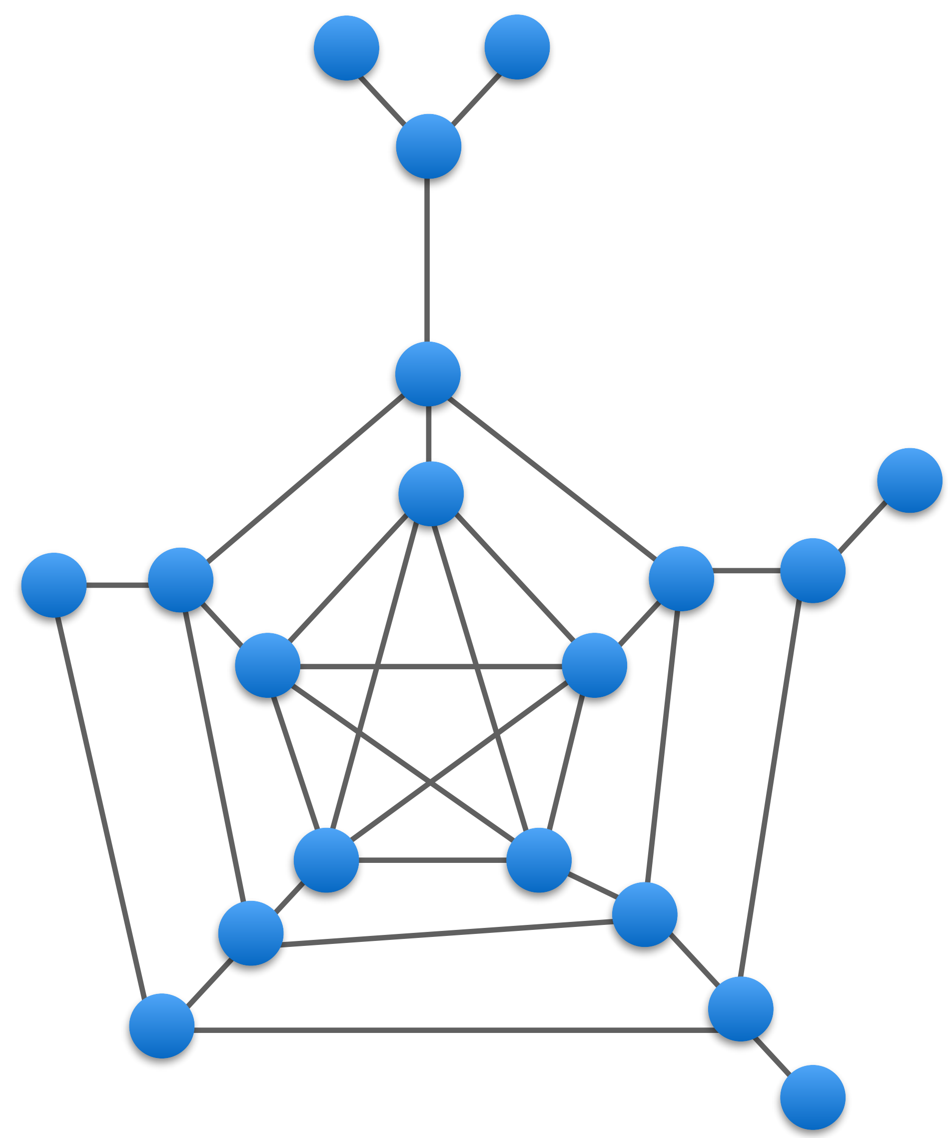
There is a simple algorithm to compute the coreness number of every node



Sequential algorithm

There is a simple algorithm to compute the coreness number of every node

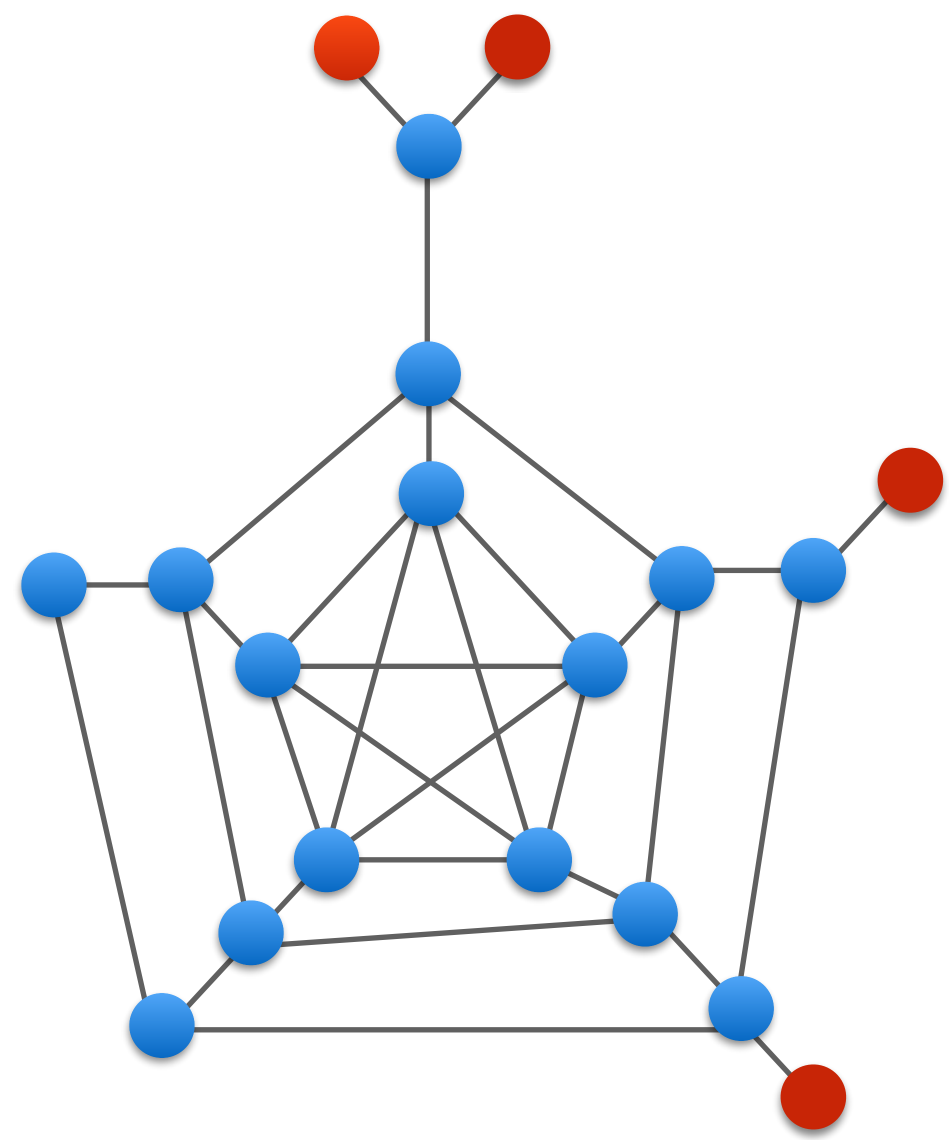
Remove all nodes of minimum degree from the graph and assign their current degree as their coreness number



Sequential algorithm

There is a simple algorithm to compute the coreness number of every node

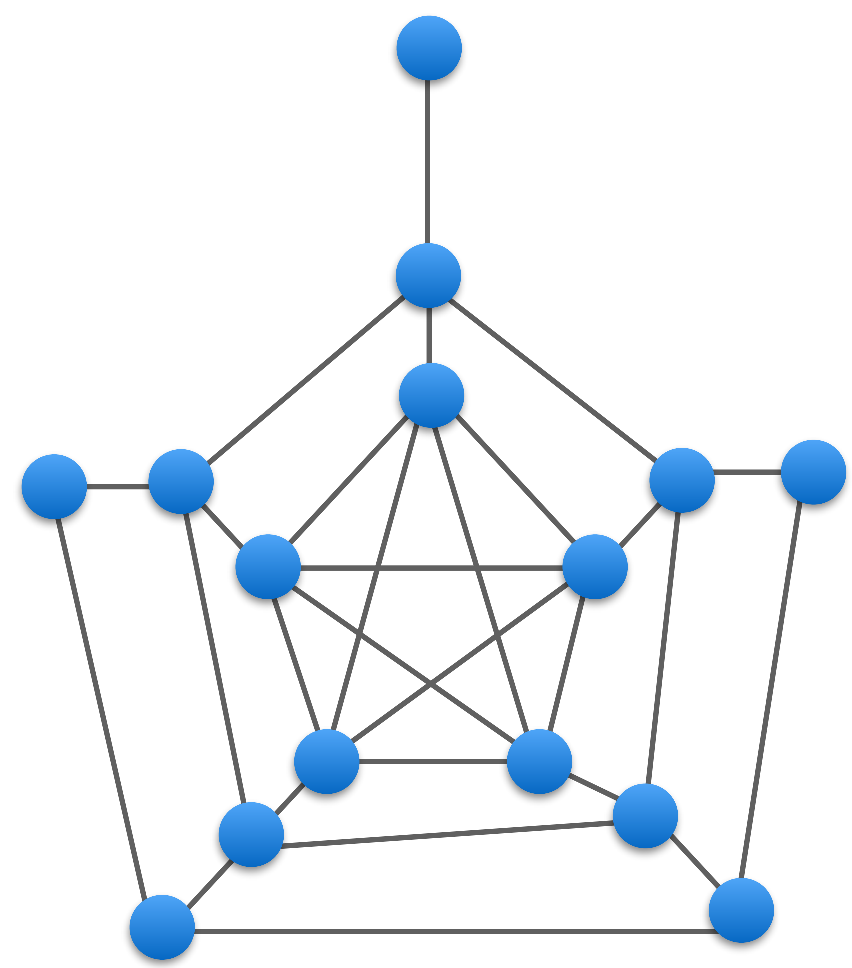
Remove all nodes of minimum degree from the graph and assign their current degree as their coreness number



Sequential algorithm

There is a simple algorithm to compute the coreness number of every node

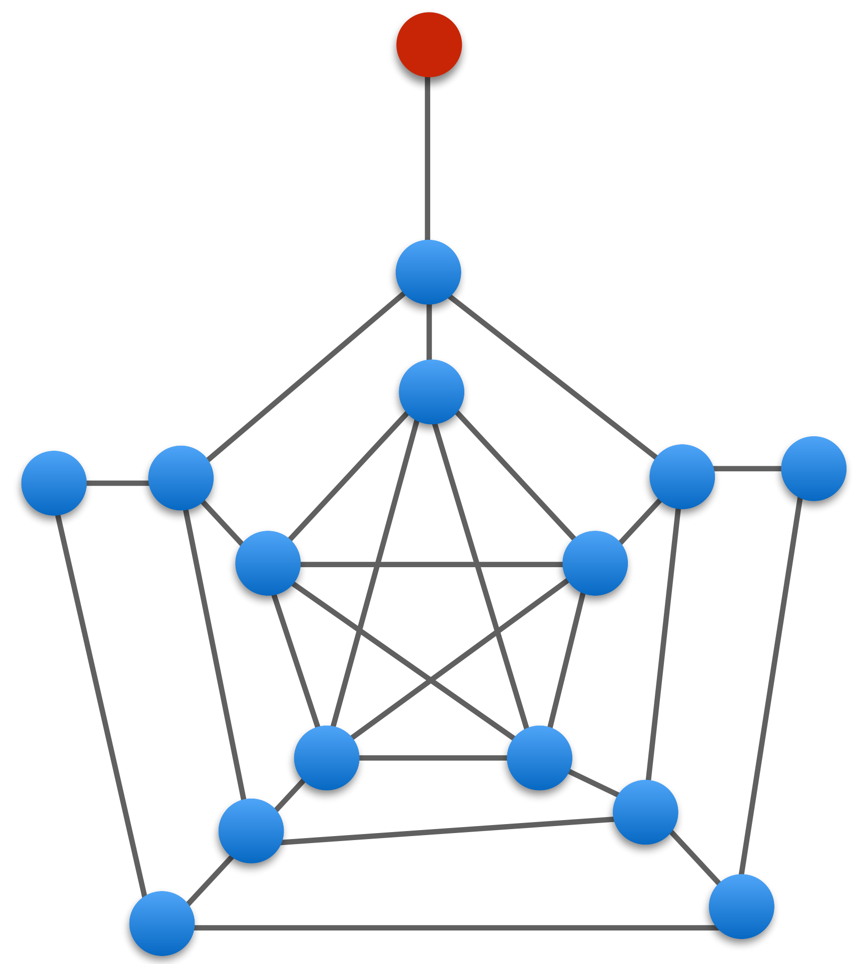
Remove all nodes of minimum degree from the graph and assign their current degree as their coreness number



Sequential algorithm

There is a simple algorithm to compute the coreness number of every node

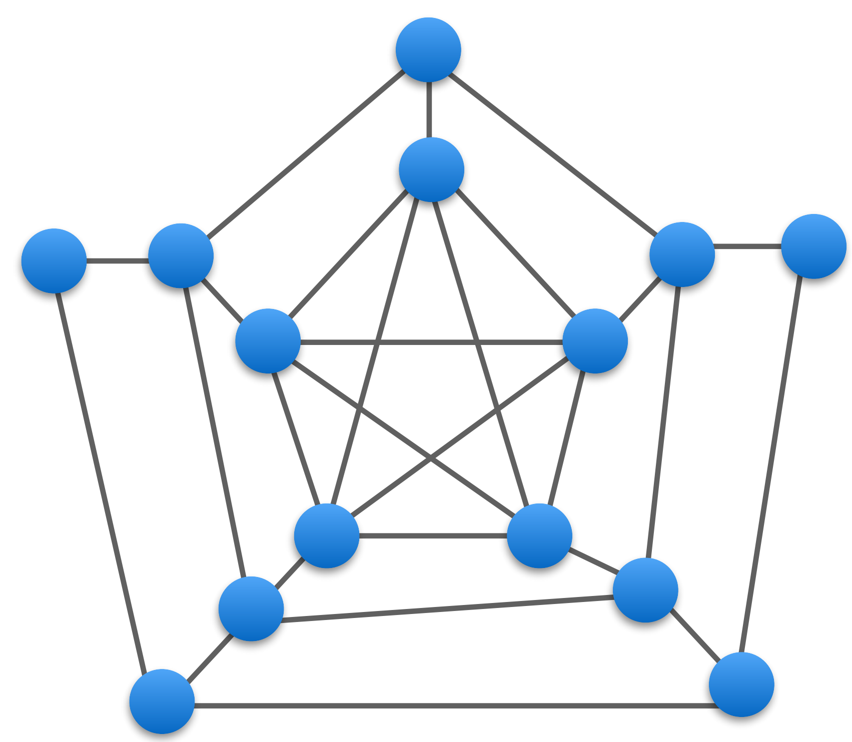
Remove all nodes of minimum degree from the graph and assign their current degree as their coreness number



Sequential algorithm

There is a simple algorithm to compute the coreness number of every node

Remove all nodes of minimum degree from the graph and assign their current degree as their coreness number



Approximating K-core

A $(1-\epsilon)$-approximate K-core is a subgraph where:
- every node has degree at least $(1-\epsilon)$K
- contains the K-core

Approximating K-core

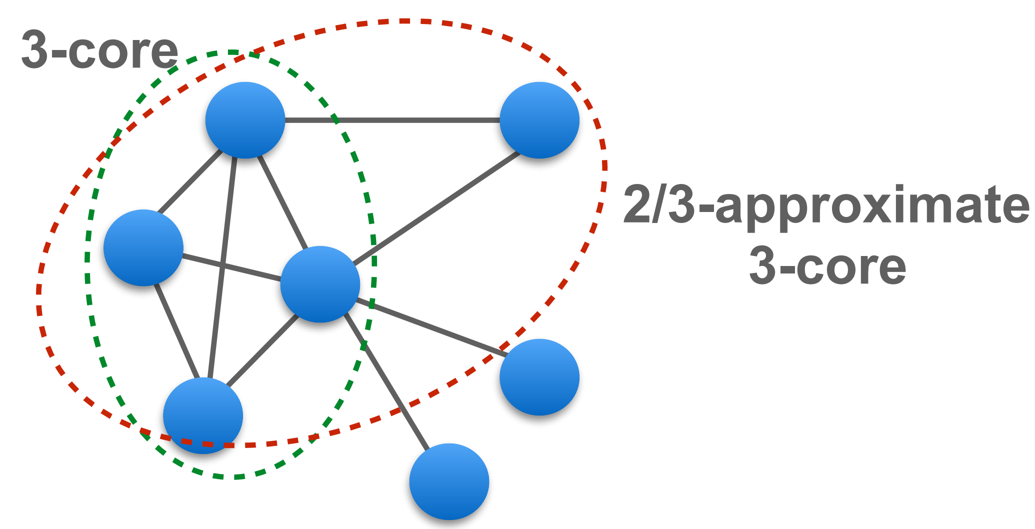
A $(1-\epsilon)$-approximate K-core is a subgraph where:
- every node has degree at least $(1-\epsilon)$K
- contains the K-core

The $(1-\epsilon)$-approximate coreness number of vertex v is maximum K for which v is part of a $(1-\epsilon)$-approximate K-core

Approximating K-core

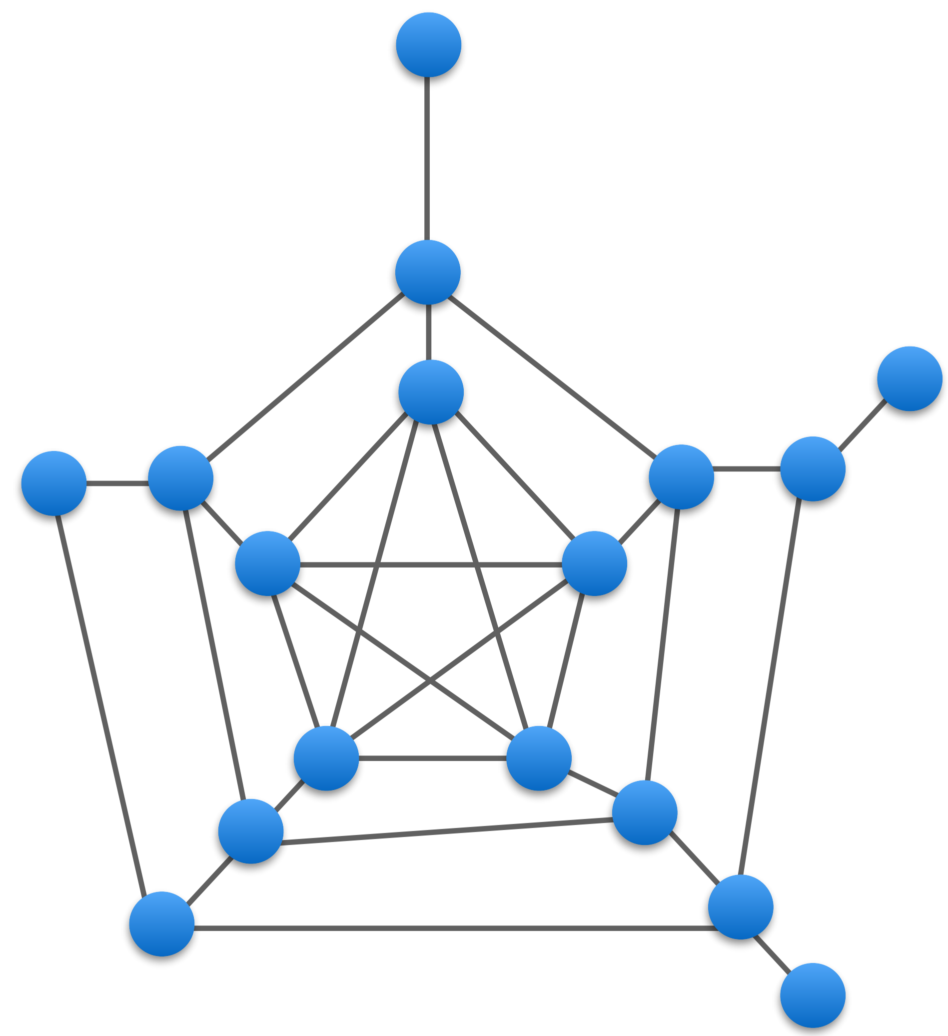
A $(1-\epsilon)$-approximate K-core is a subgraph where:
- every node has degree at least $(1-\epsilon)$K
- contains the K-core

The $(1-\epsilon)$-approximate coreness number of vertex v is maximum K for which v is part of a $(1-\epsilon)$-approximate K-core



An approximate sequential algorithm

Find a small summary that can be used to approximate the instance

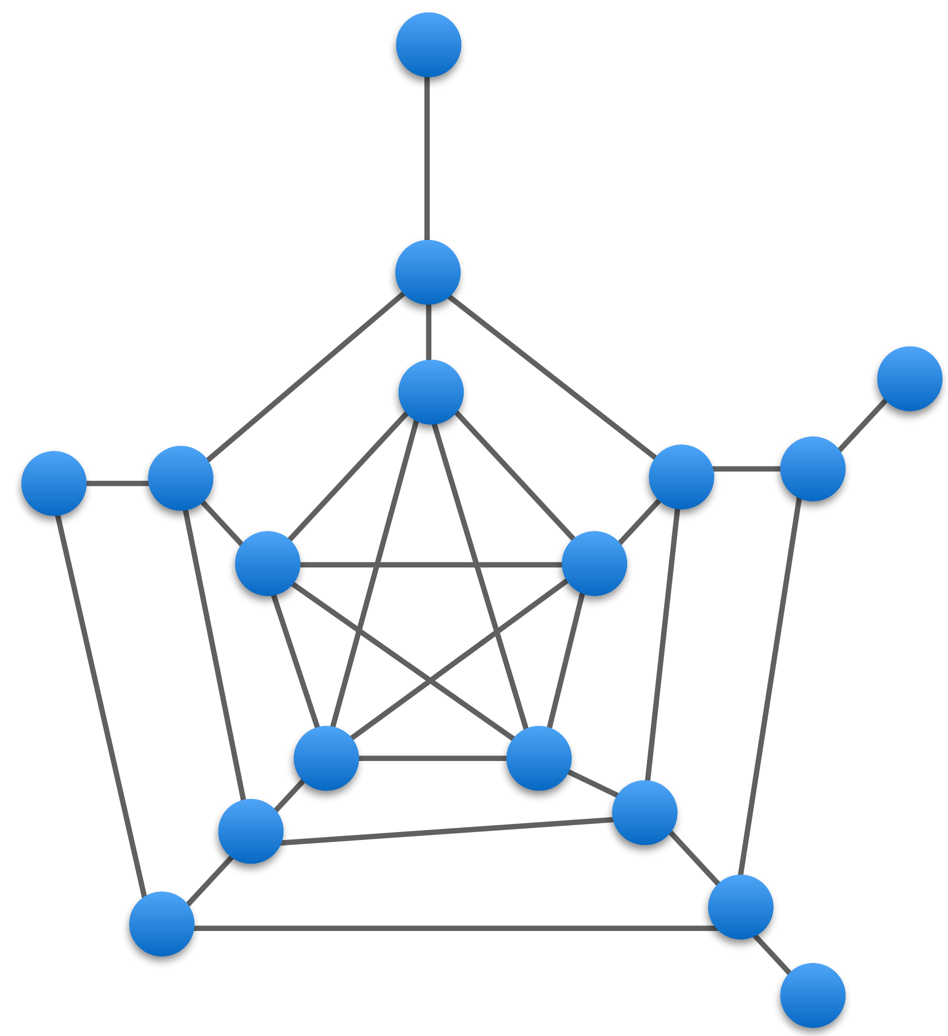


An approximate sequential algorithm

Find a small summary that can be used to approximate the instance

First idea:

Use uniform sampling to
sparsity the graph and
then use sequential algorithm

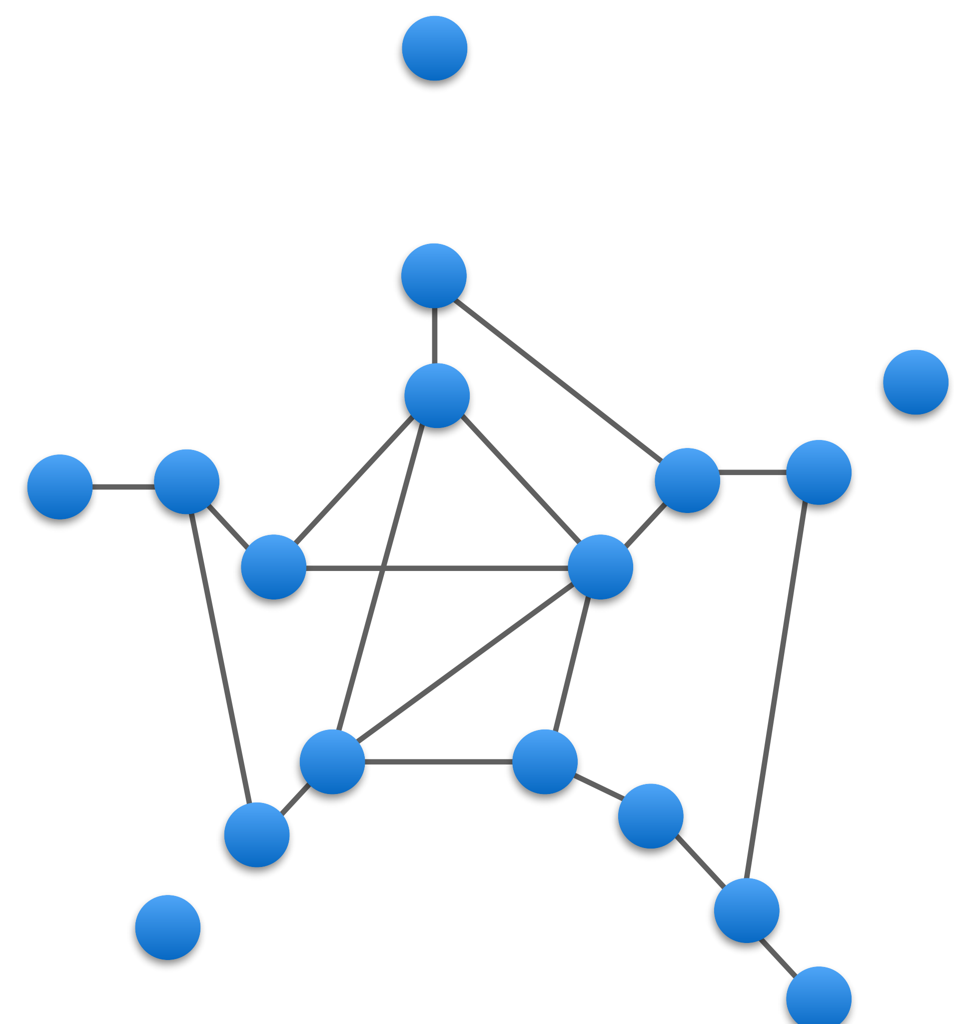


An approximate sequential algorithm

Find a small summary that can be used to approximate the instance

First idea:

Use uniform sampling to
sparsity the graph and
then use sequential algorithm



Sample every edge with
probability p

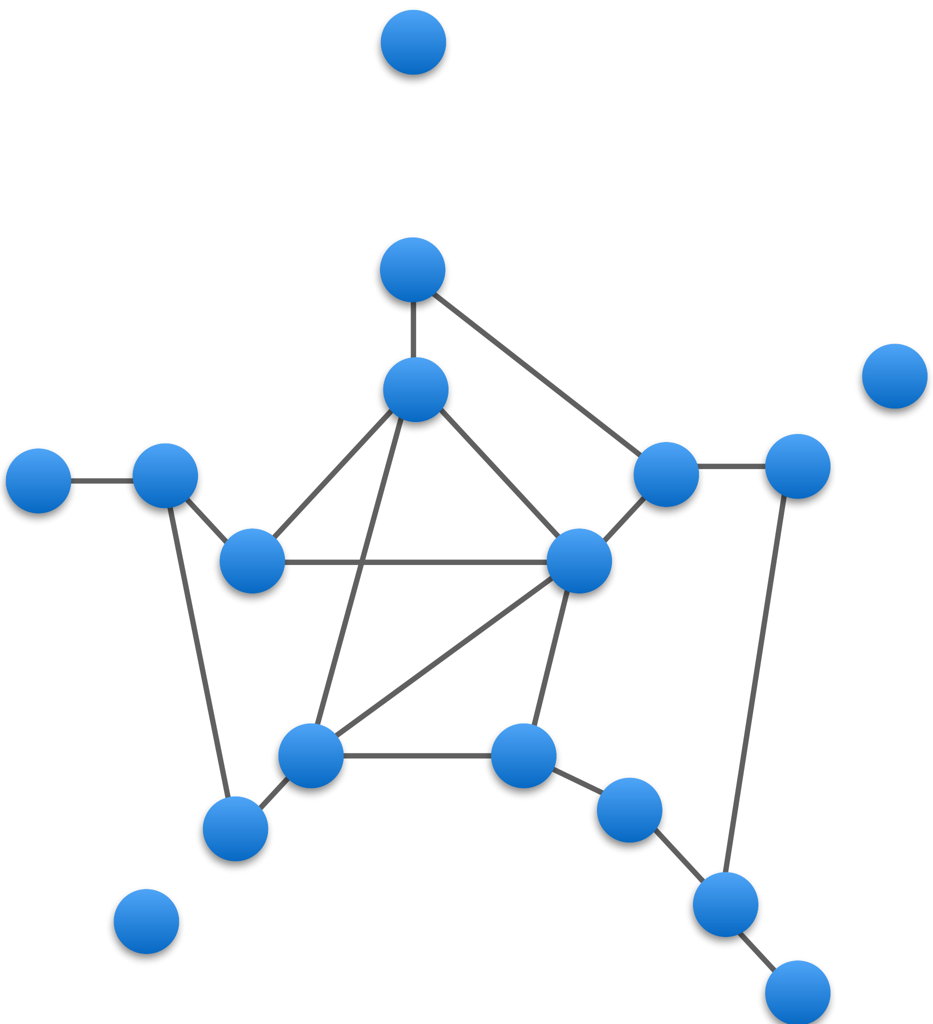
An approximate sequential algorithm

Find a small summary that can be used to approximate the instance

First idea:

Use uniform sampling to
sparsity the graph and
then use sequential algorithm

Issue:
We can estimate coreness
for high degree nodes only



Sample every edge with
probability p

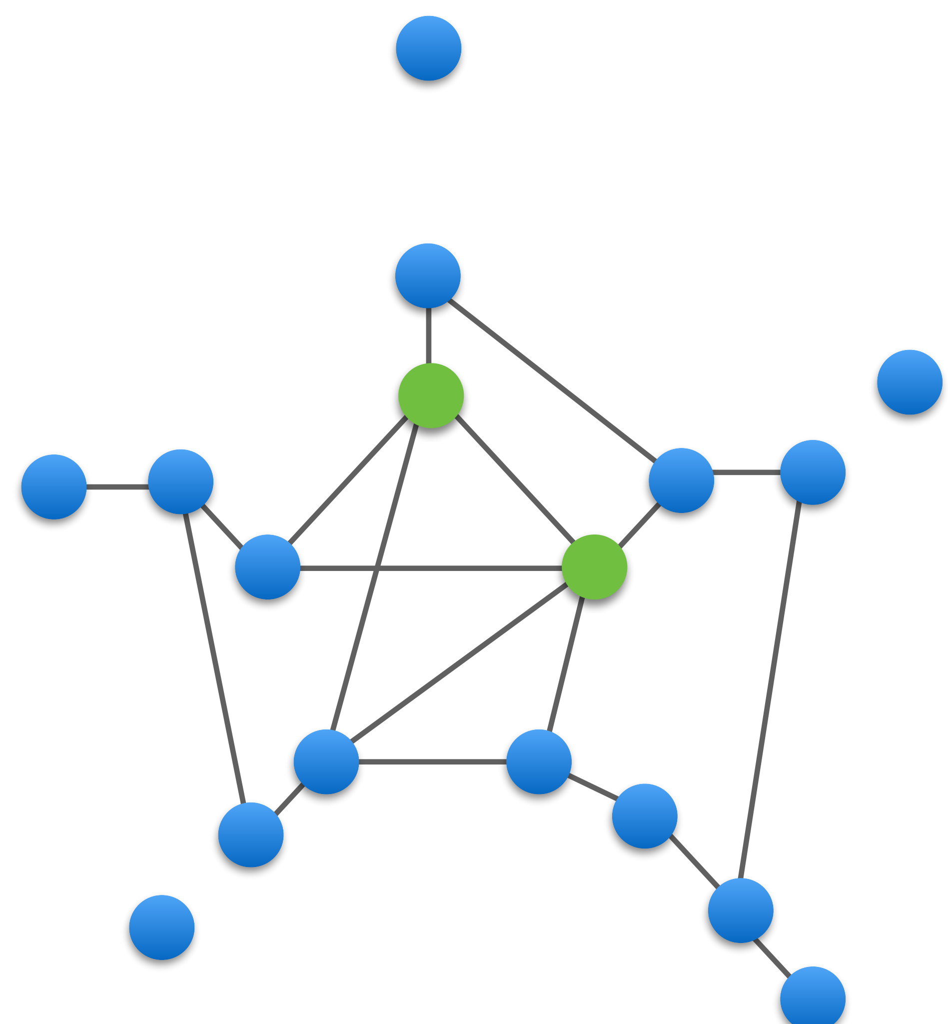
An approximate sequential algorithm

Find a small summary that can be used to approximate the instance

First idea:

Use uniform sampling to
sparsity the graph and
then use sequential algorithm

Issue:
We can estimate coreness
for high degree nodes only



Sample every edge with
probability p

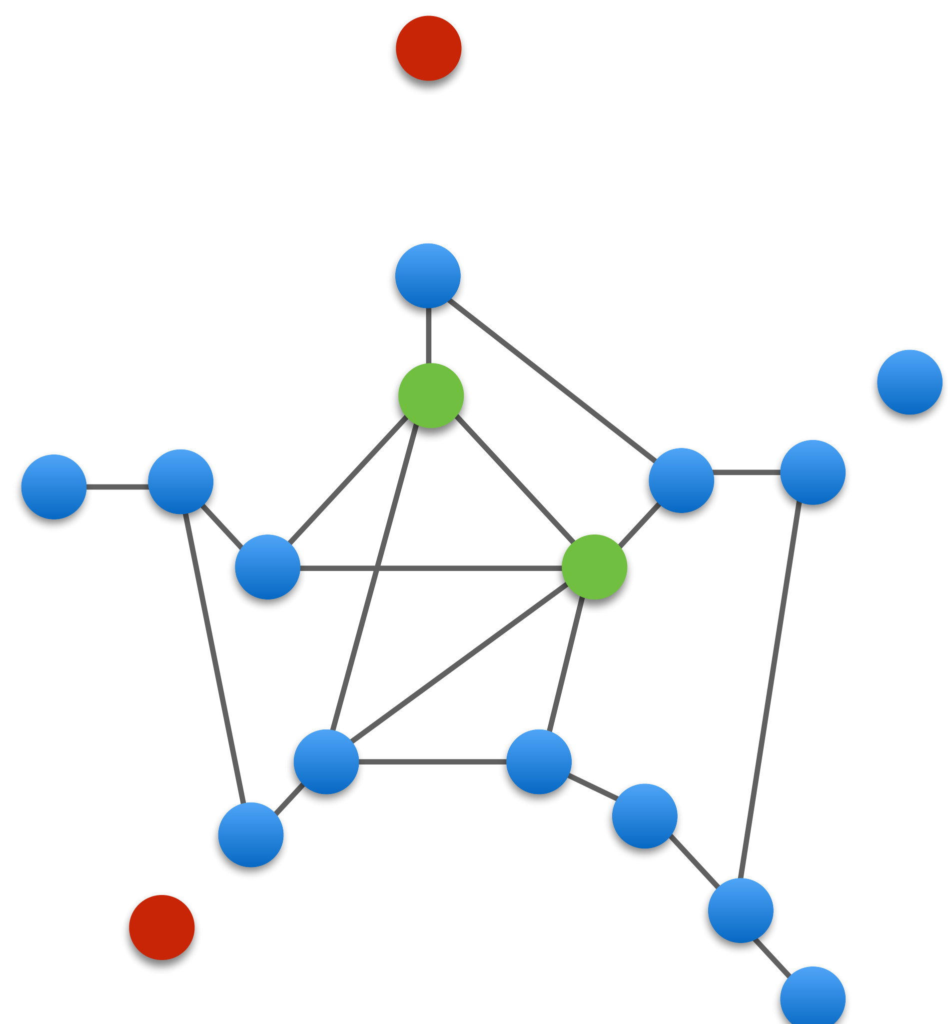
An approximate sequential algorithm

Find a small summary that can be used to approximate the instance

First idea:

Use uniform sampling to
sparsity the graph and
then use sequential algorithm

Issue:
We can estimate coreness
for high degree nodes only



Sample every edge with
probability p

An approximate sequential algorithm

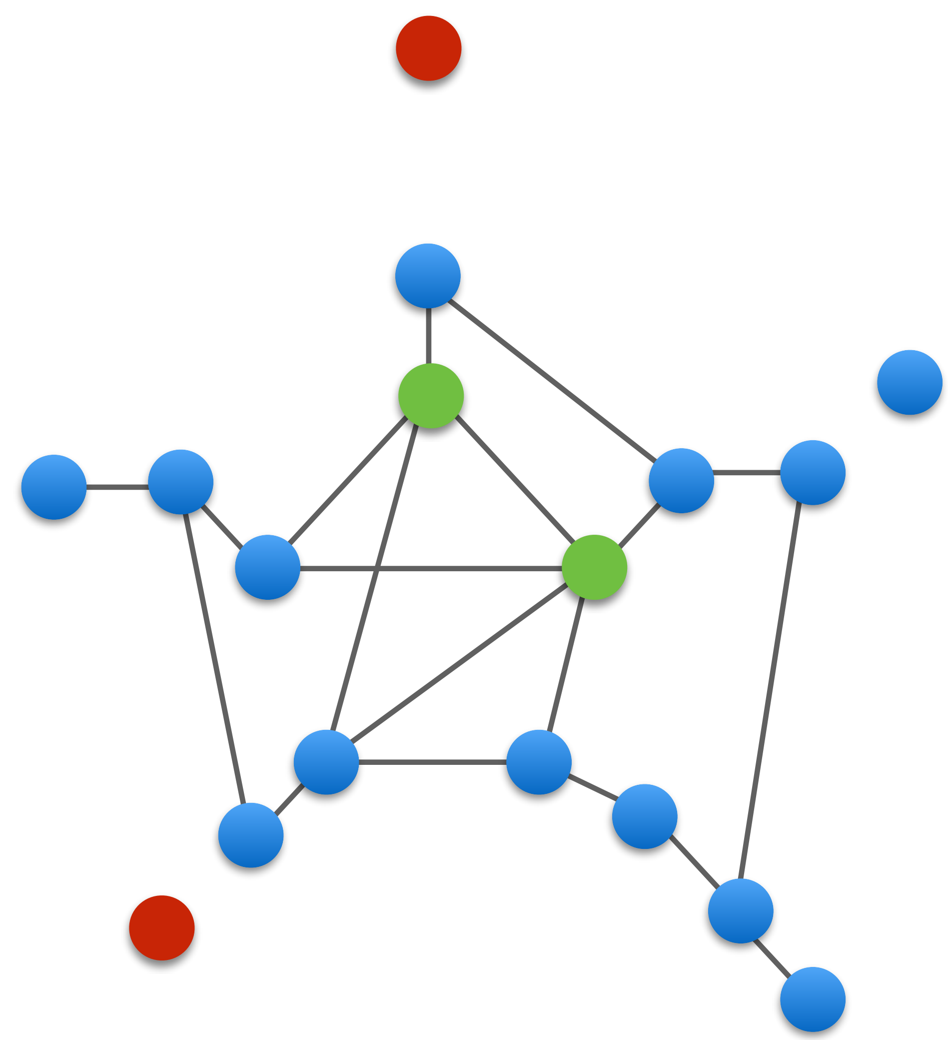
Find a small summary that can be used to approximate the instance

First idea:

Use uniform sampling to sparsity the graph and then use sequential algorithm

We want to estimate the coreness of every node

Issue:
We can estimate coreness for high degree nodes only

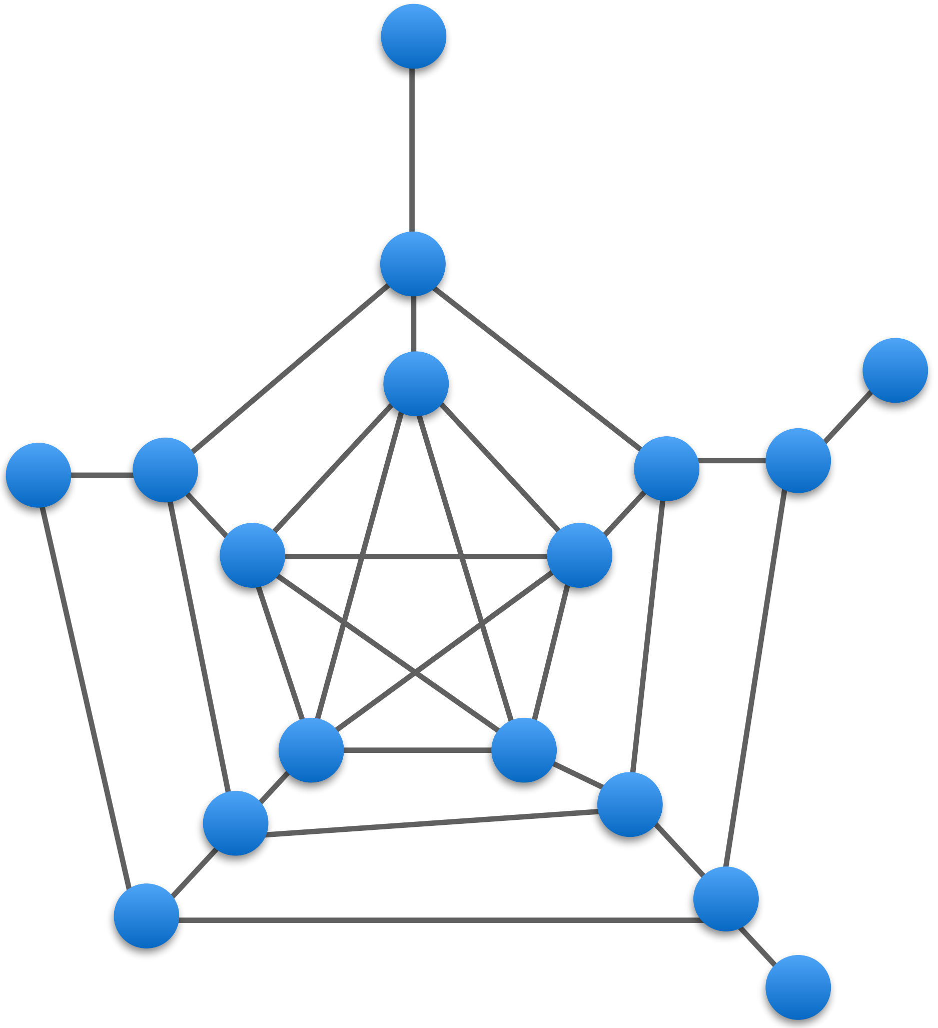


An approximate algorithm

Find a small summary that can be used to approximate the instance

Our algorithm:

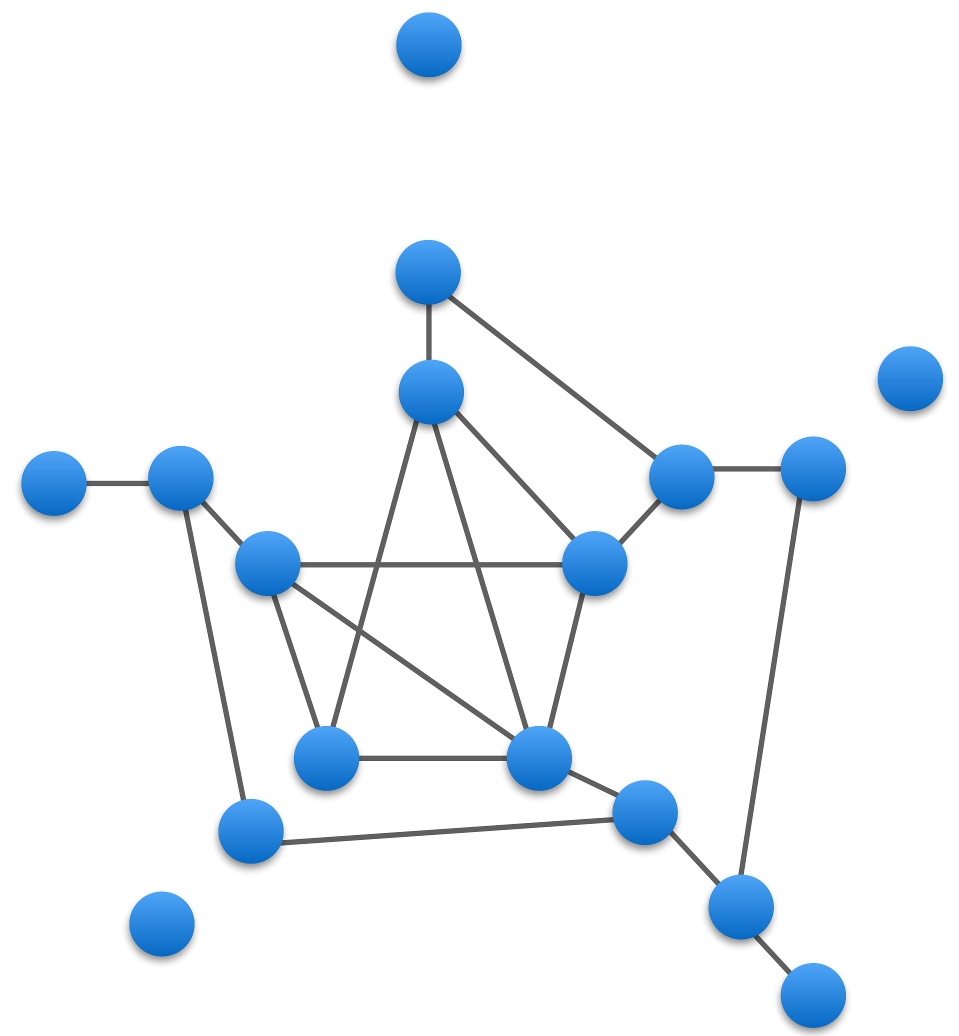
- Sample edge with probability p
- For nodes with high degree in the sample estimate the coreness number and add them to S
- Remove edges with both endpoints in S
- Multiply p by 2 and restart



An approximate algorithm

Find a small summary that can be used to approximate the instance

Our algorithm:
- Sample edge with probability p
- For nodes with high coreness number in the sample estimate the coreness number and add them to S
- Remove edges with both endpoints in S
- Multiply p by 2 and restart

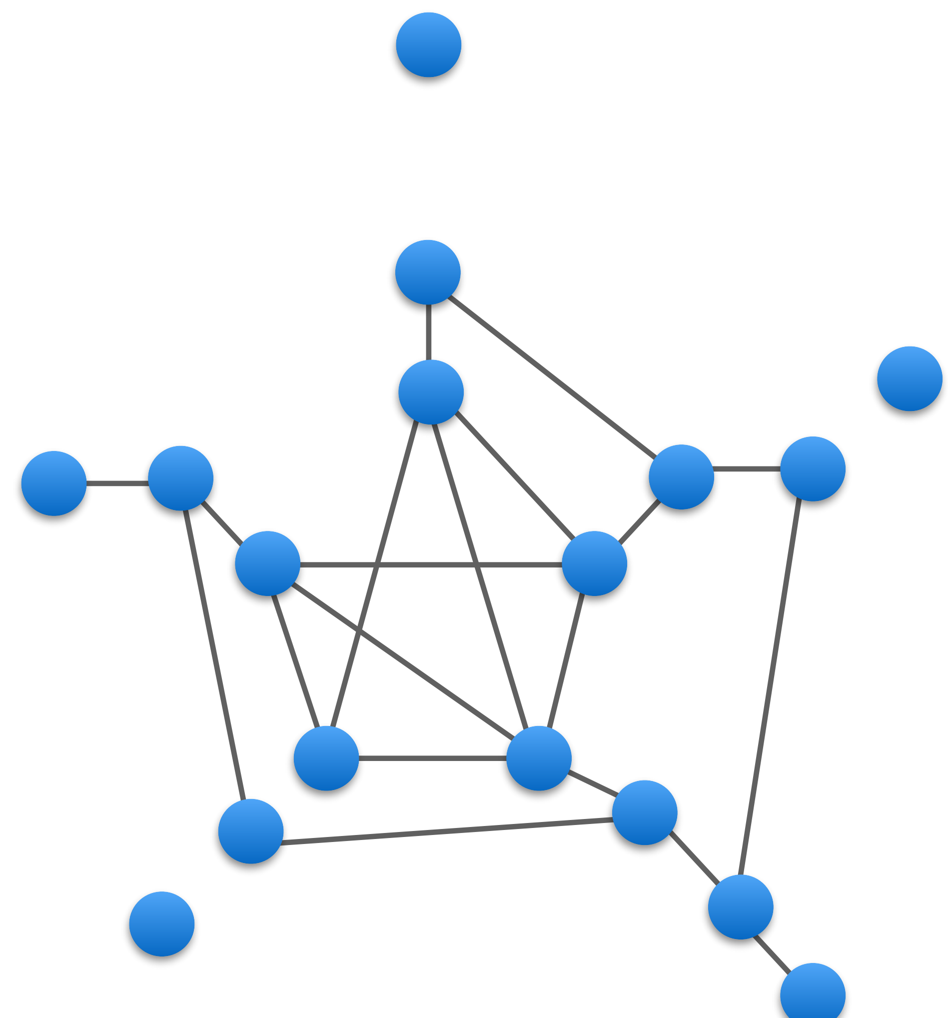


An approximate algorithm

Find a small summary that can be used to approximate the instance

Our algorithm:

- Sample edge with probability p
- For nodes with high coreness number in the sample estimate the coreness number and add them to S
- Remove edges with both endpoints in S
- Multiply p by 2 and restart



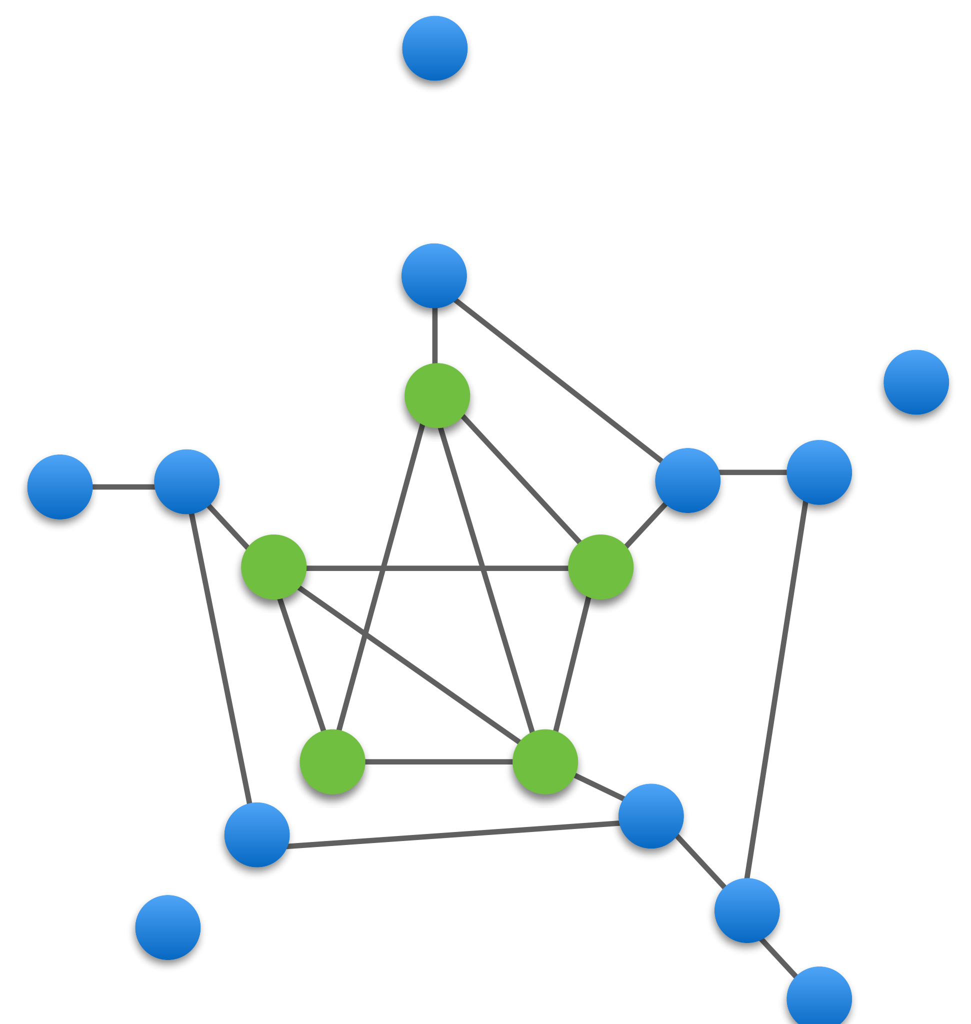
If a node has coreness bigger than 2 compute its coreness number

An approximate algorithm

Find a small summary that can be used to approximate the instance

Our algorithm:

- Sample edge with probability p
- For nodes with high coreness number in the sample estimate the coreness number and add them to S
- Remove edges with both endpoints in S
- Multiply p by 2 and restart



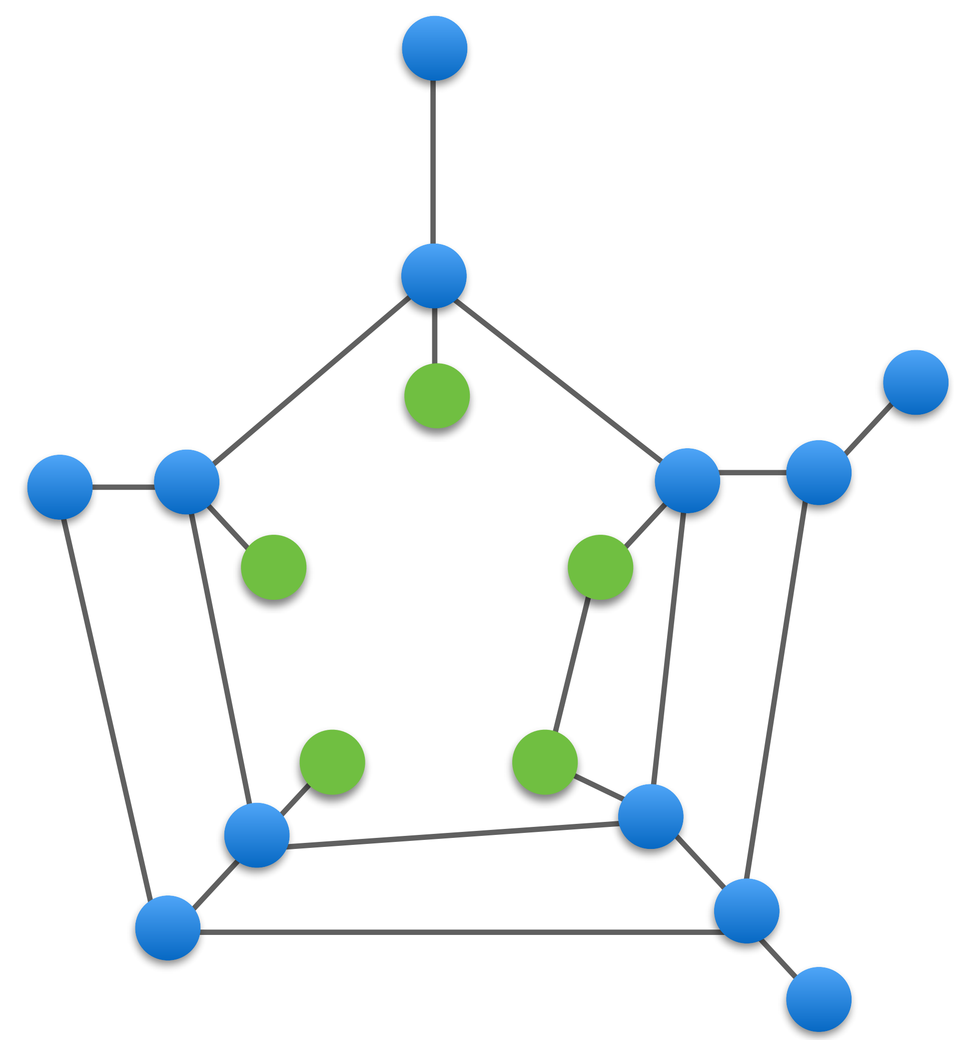
If a node has coreness bigger than 2 compute its coreness number

An approximate algorithm

Find a small summary that can be used to approximate the instance

Our algorithm:

- Sample edge with probability p
- For nodes with high coreness number in the sample estimate the coreness number and add them to S
- Remove edges with both endpoints in S
- Multiply p by 2 and restart

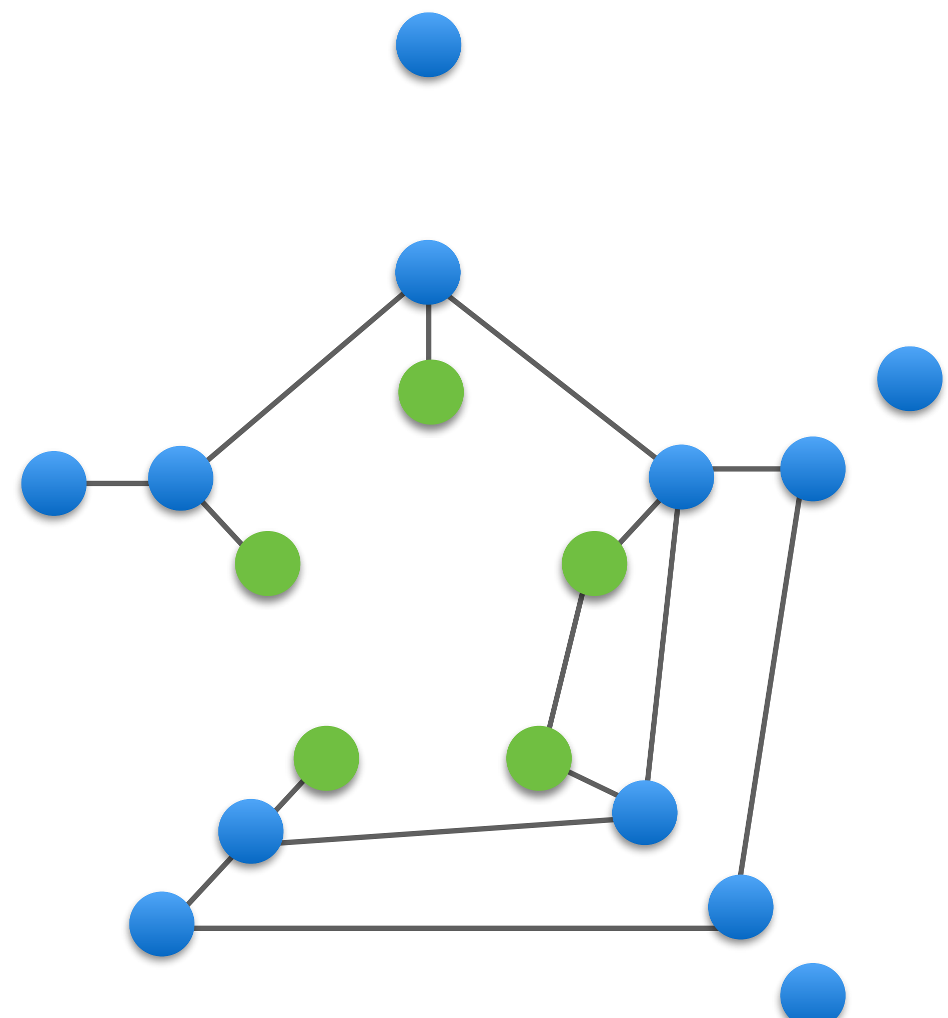


An approximate algorithm

Find a small summary that can be used to approximate the instance

Our algorithm:

- Sample edge with probability p
- For nodes with high coreness number in the sample estimate the coreness number and add them to S
- Remove edges with both endpoints in S
- Multiply p by 2 and restart



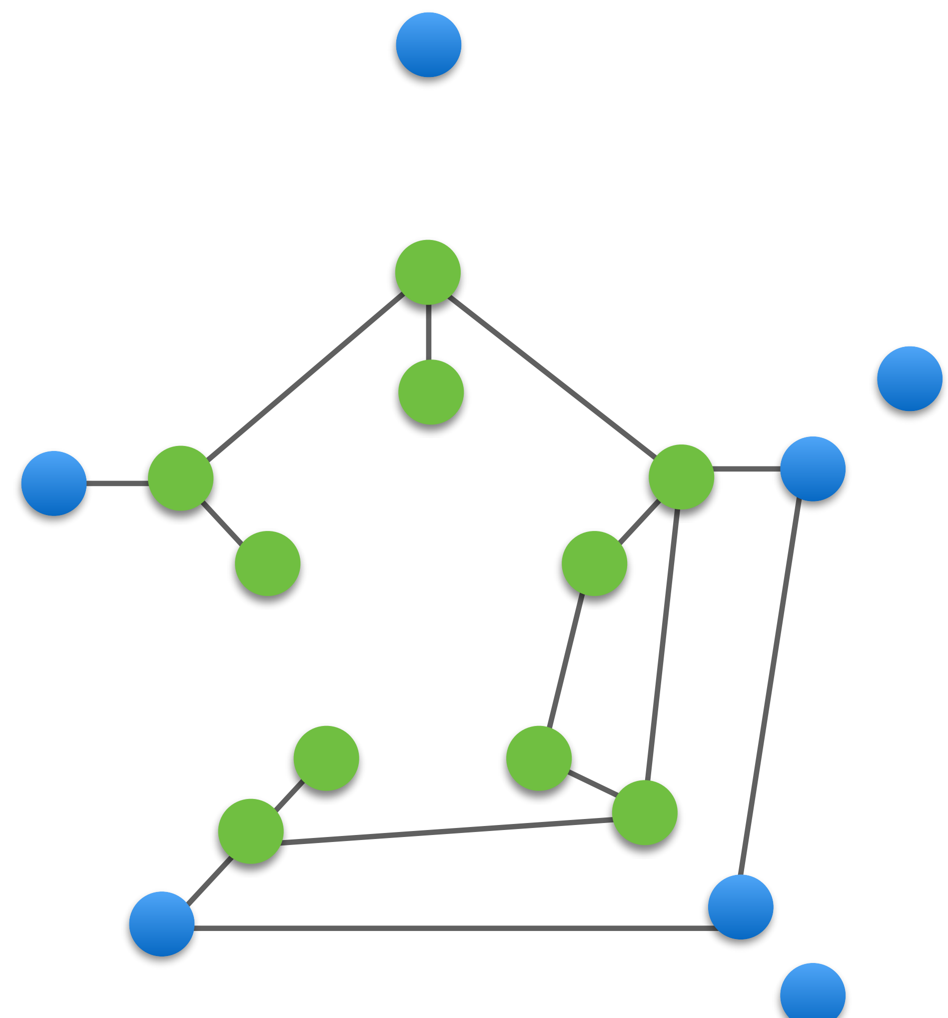
If a node has coreness bigger than 2 compute its coreness number

An approximate algorithm

Find a small summary that can be used to approximate the instance

Our algorithm:

- Sample edge with probability p
- For nodes with high coreness number in the sample estimate the coreness number and add them to S
- Remove edges with both endpoints in S
- Multiply p by 2 and restart



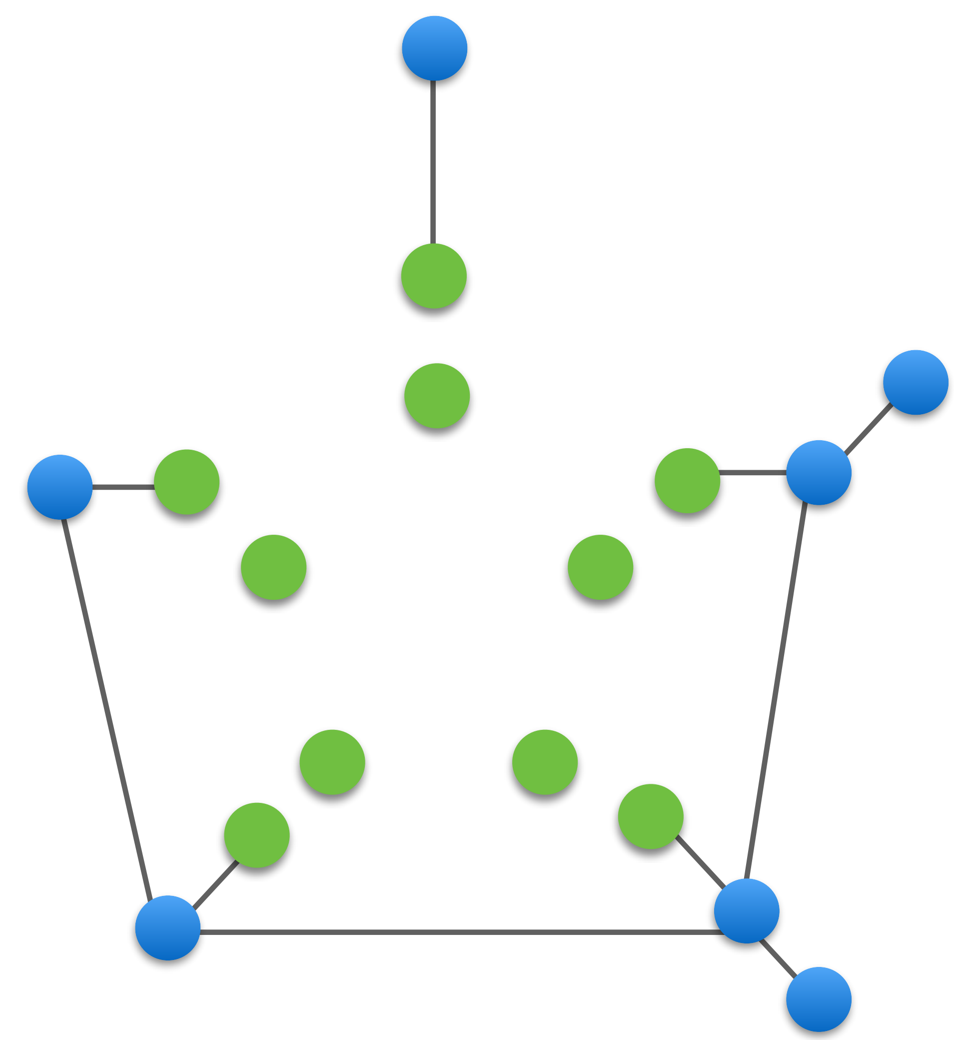
If a node has coreness bigger than 2 compute its coreness number

An approximate algorithm

Find a small summary that can be used to approximate the instance

Our algorithm:

- Sample edge with probability p
- For nodes with high coreness number in the sample estimate the coreness number and add them to S
- Remove edges with both endpoints in S
- Multiply p by 2 and restart



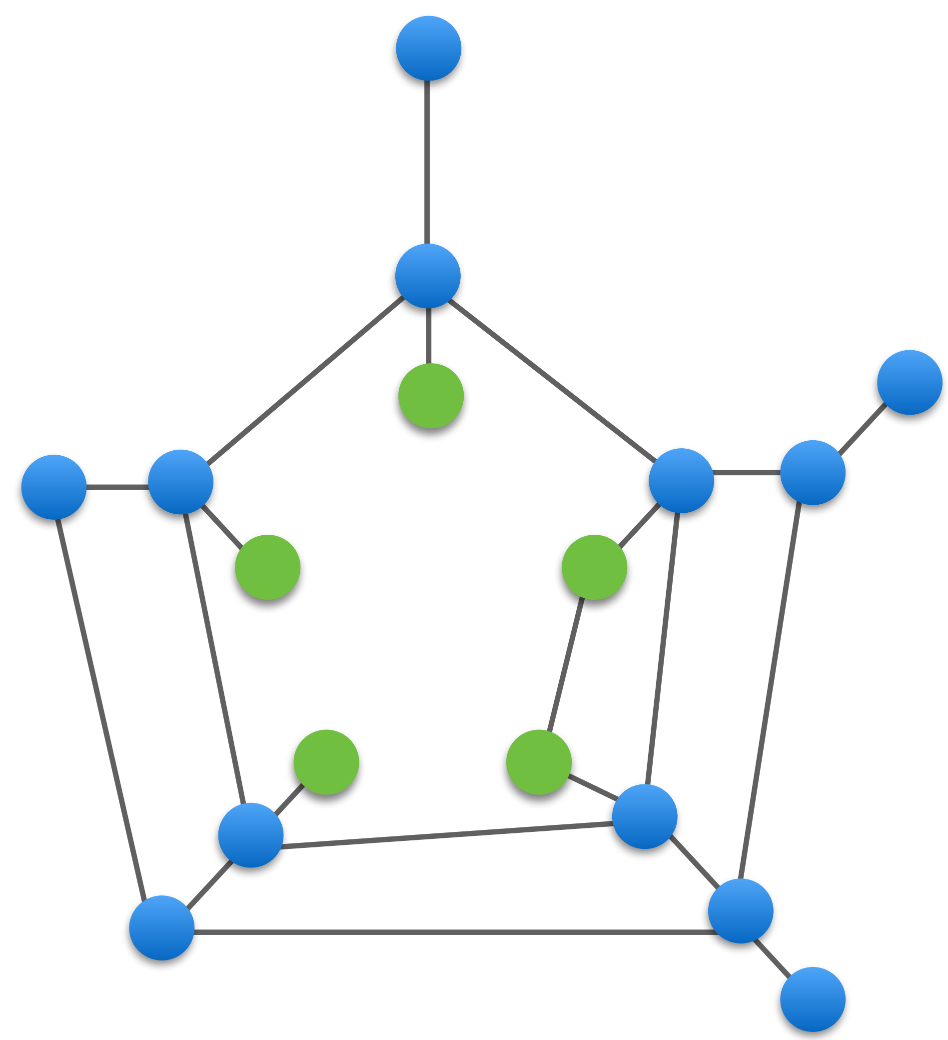
An approximate algorithm

Find a small summary that can be used to approximate the instance

Our algorithm:

- Sample edge with probability p
- For nodes with high coreness number in the sample estimate the coreness number and add them to S
- Remove edges with both endpoints in S
- Multiply p by 2 and restart

To estimate the coreness number, we run the sequential algorithm but we never remove nodes in S



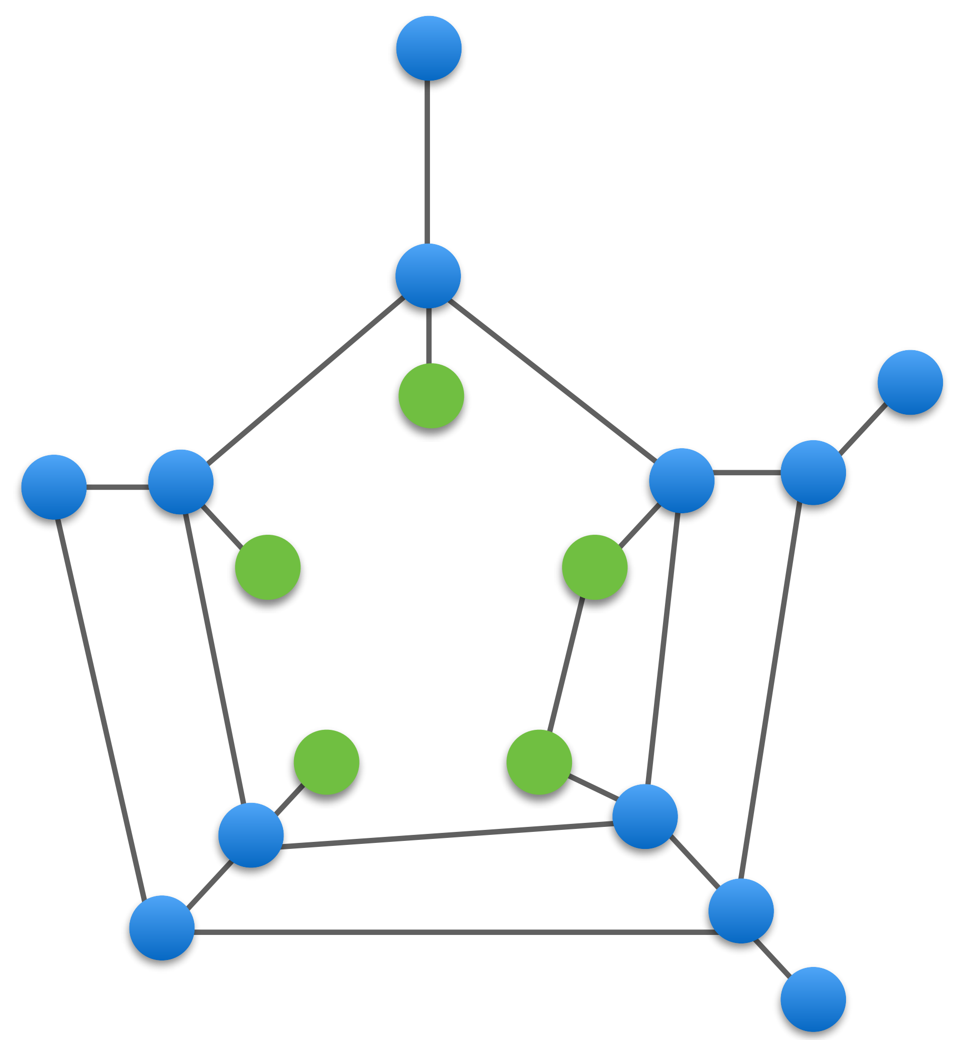
Main properties

Quality of the solution

If a node has expected logarithmic coreness after sampling, its coreness number can be estimated precisely

Size of the sample

After each sample the number of edges left in the graph is almost linear in the number of nodes



Main properties

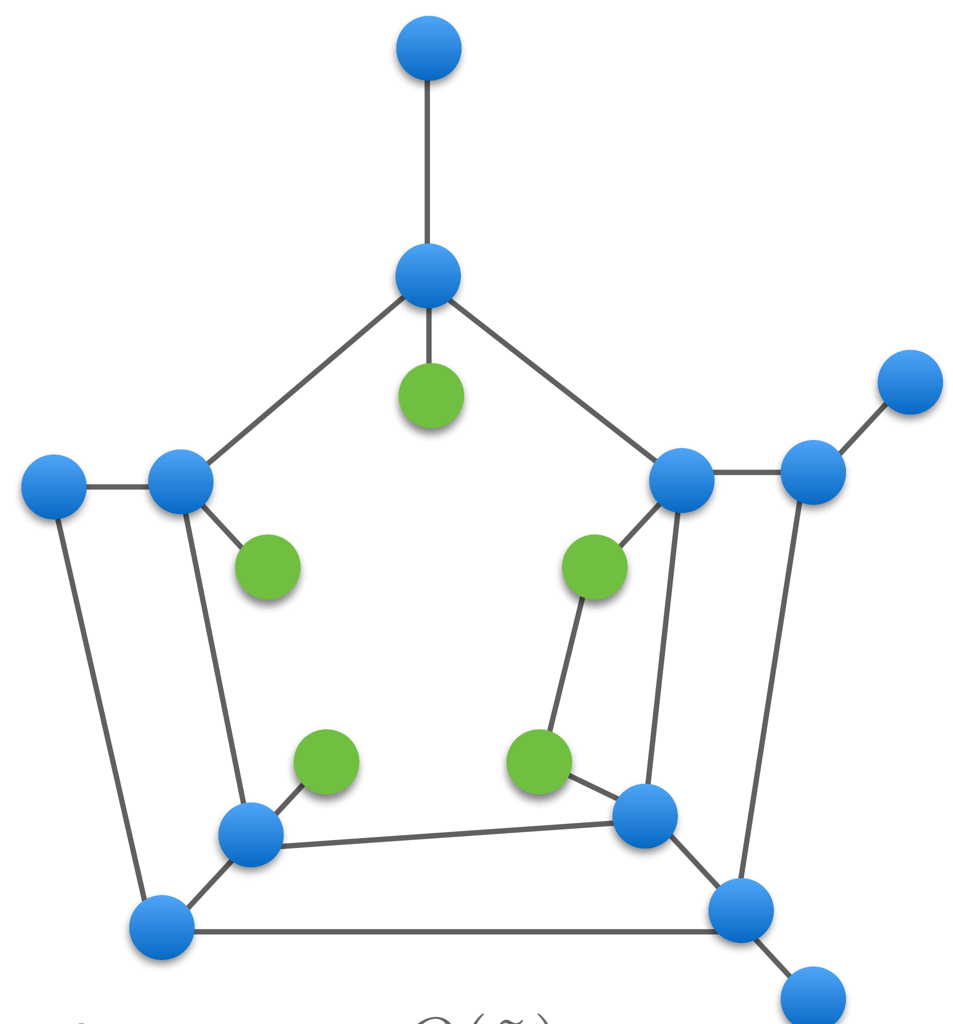
[Parallel and streaming algorithms for K-core decomposition]
Hossein Esfandiari, Silvio Lattanzi, Vahab S. Mirrokni
ICML 2018

Quality of the solution

If a node has expected logarithmic coreness after sampling, its coreness number can be estimated precisely

Size of the sample

After each sample the number of edges left in the graph is almost linear in the number of nodes

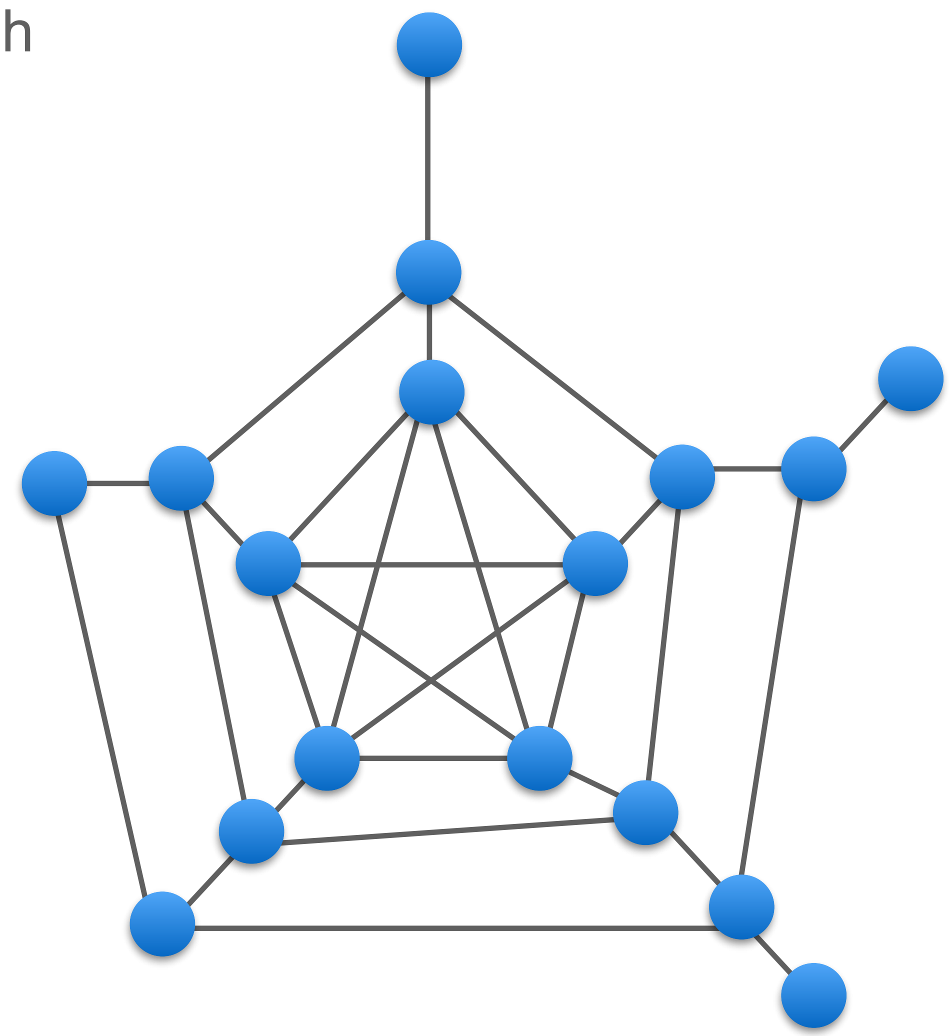


In $O(\log n)$ rounds we get a good approximation using memory $O(\tilde{n})$

Can we do better?

Sample vertices instead of edges

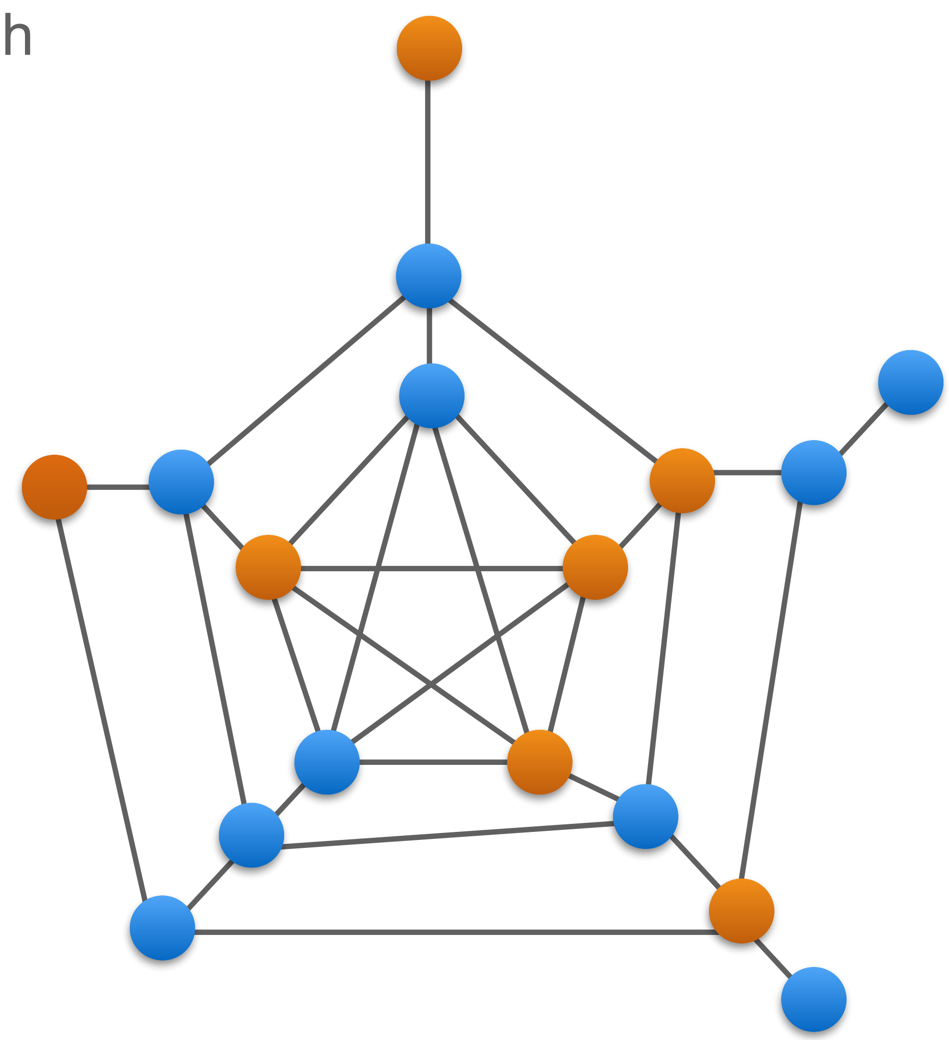
Sample vertices and consider the induce subgraph



Sampling vertices

Sample vertices instead of edges

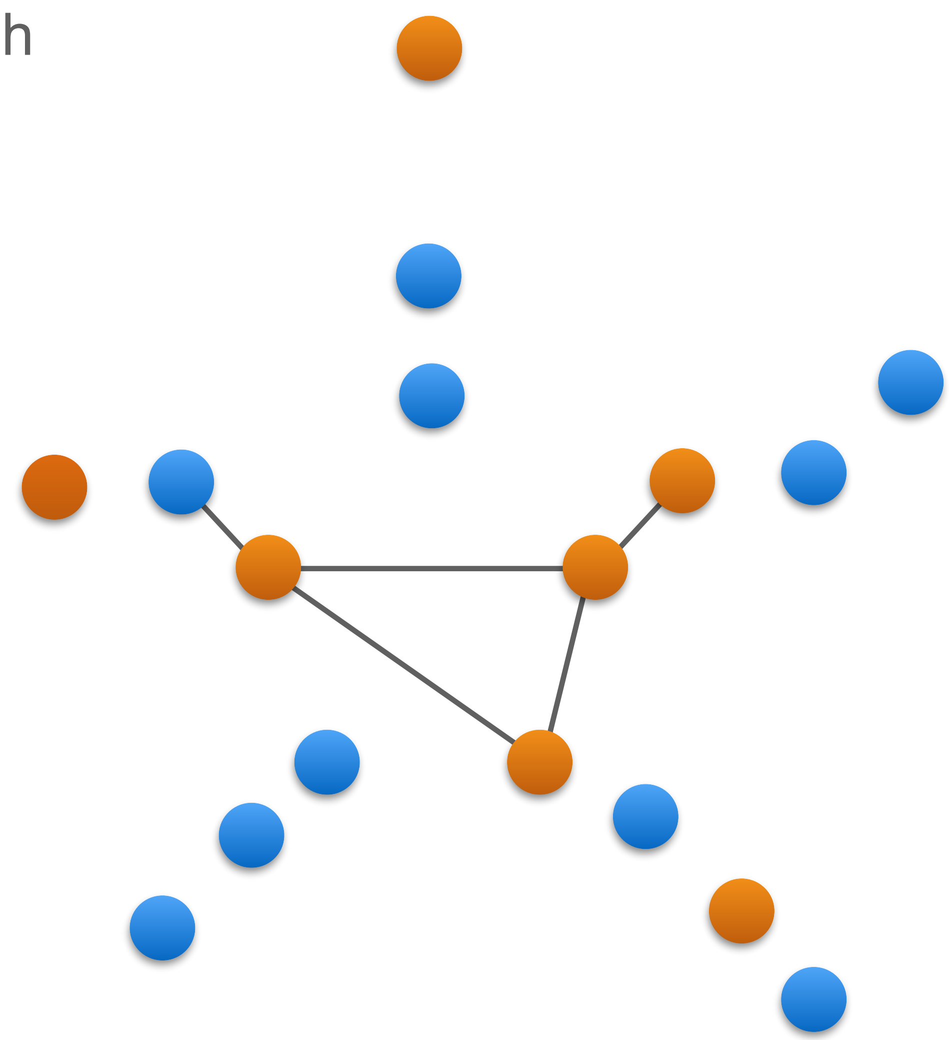
Sample vertices and consider the induce subgraph



Sampling vertices

Sample vertices instead of edges

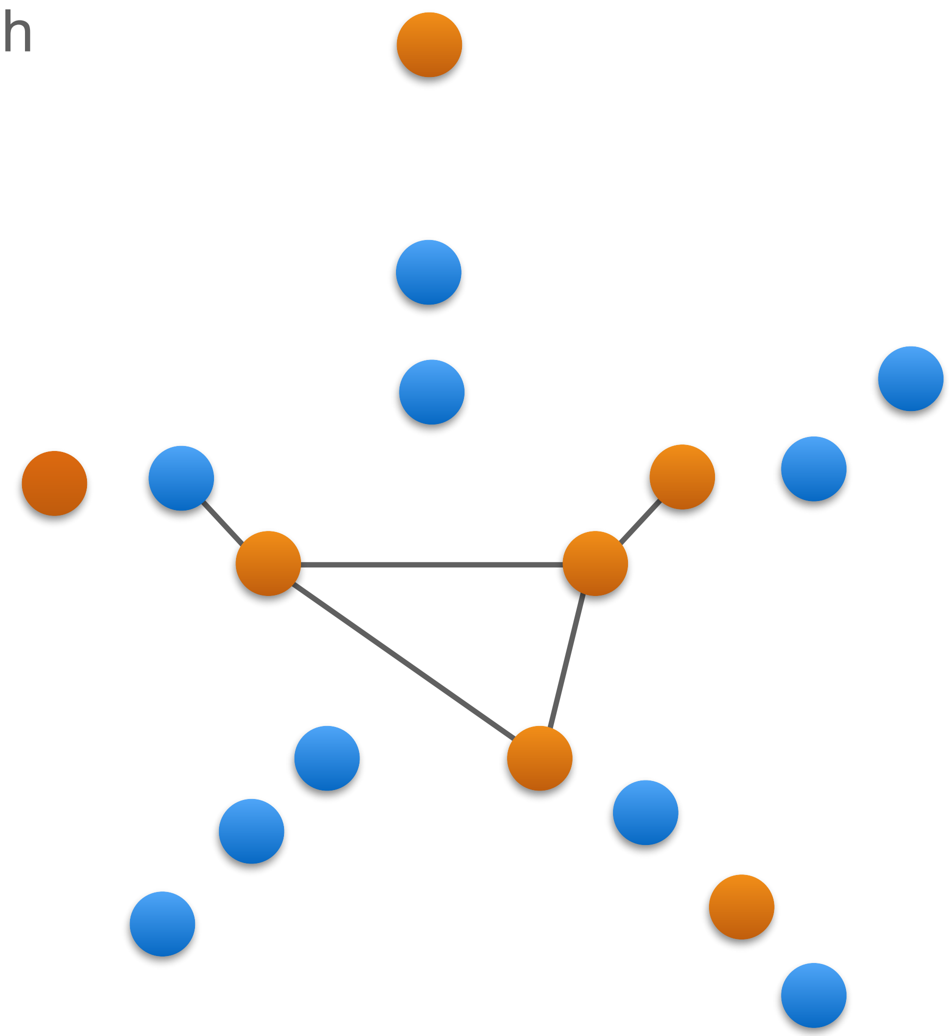
Sample vertices and consider the induce subgraph



Sampling vertices

Sample vertices instead of edges

Sample vertices and consider the induce subgraph



Main advantage

Every edge is present with probability p^2

For every sampled node a neighbour is sampled with probability p

Sampling vertices

Sample vertices instead of edges

Sample vertices and consider the induce subgraph

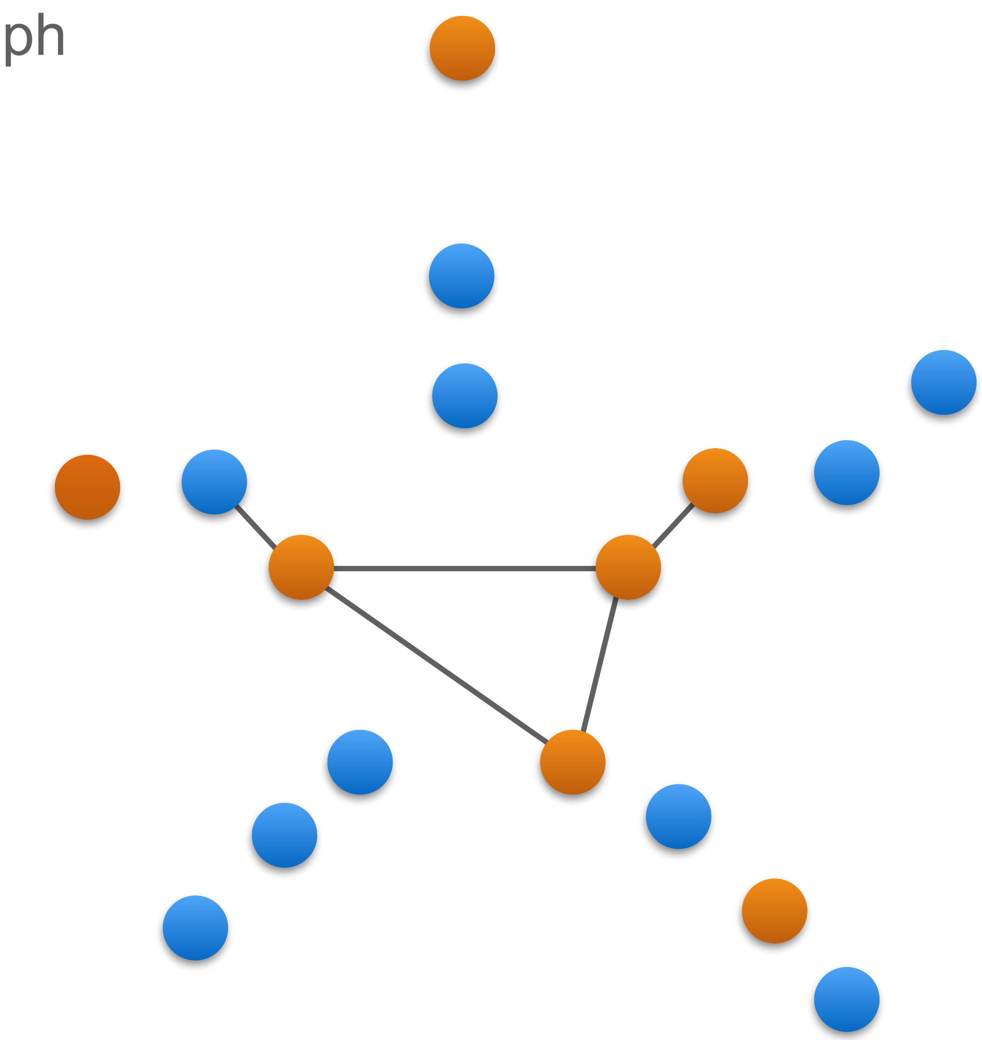
Main advantage

Every edge is present with probability p^2

For every sampled node a neighbour is sampled with probability p

Reduction in a parallel round

Every round we can reduce the maximum corners number exponentially



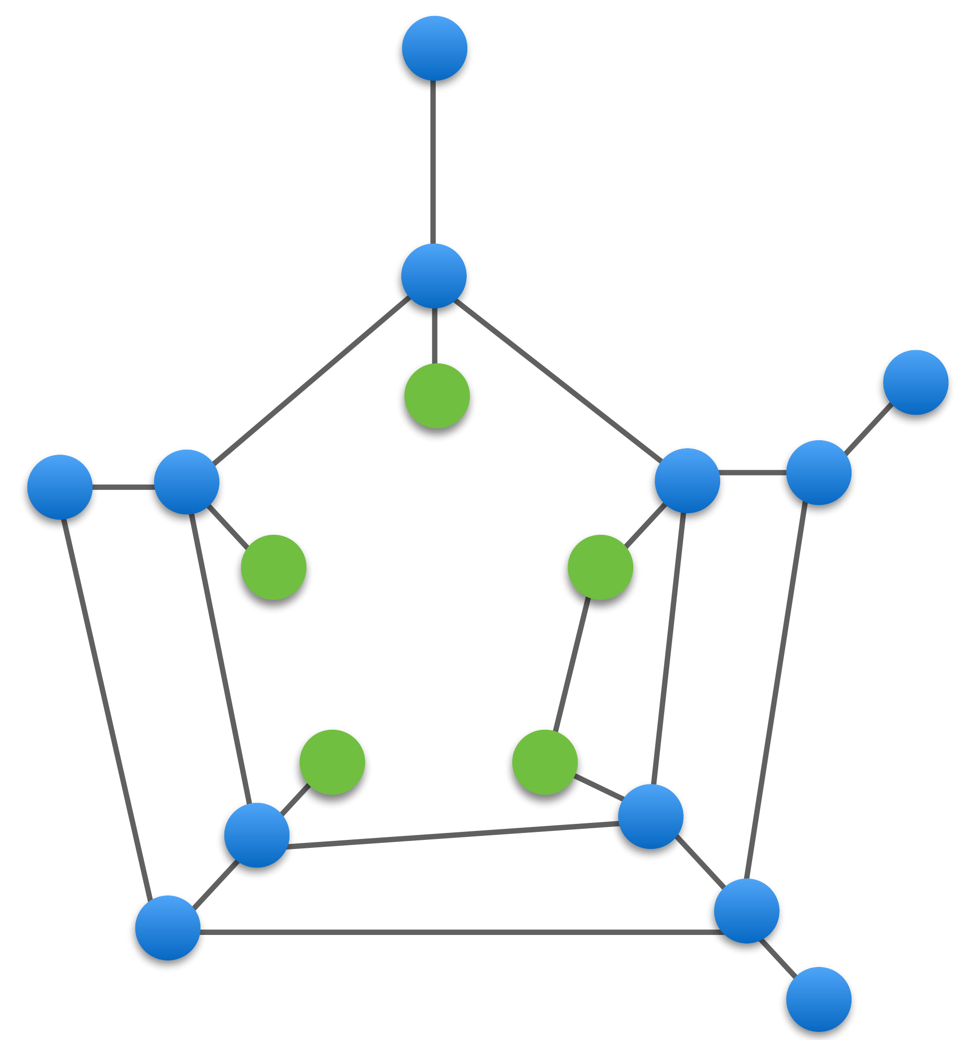
An approximate algorithm

Find a small summary that can be used to approximate the instance

Our algorithm:

- Sample nodes with probability p
- For nodes with high coreness number in the sample estimate the coreness number and add them to S
- Remove edges with both endpoints in S
- Let $p = p^{0.9}$

To estimate the coreness number, we run the sequential algorithm but we never remove nodes in S

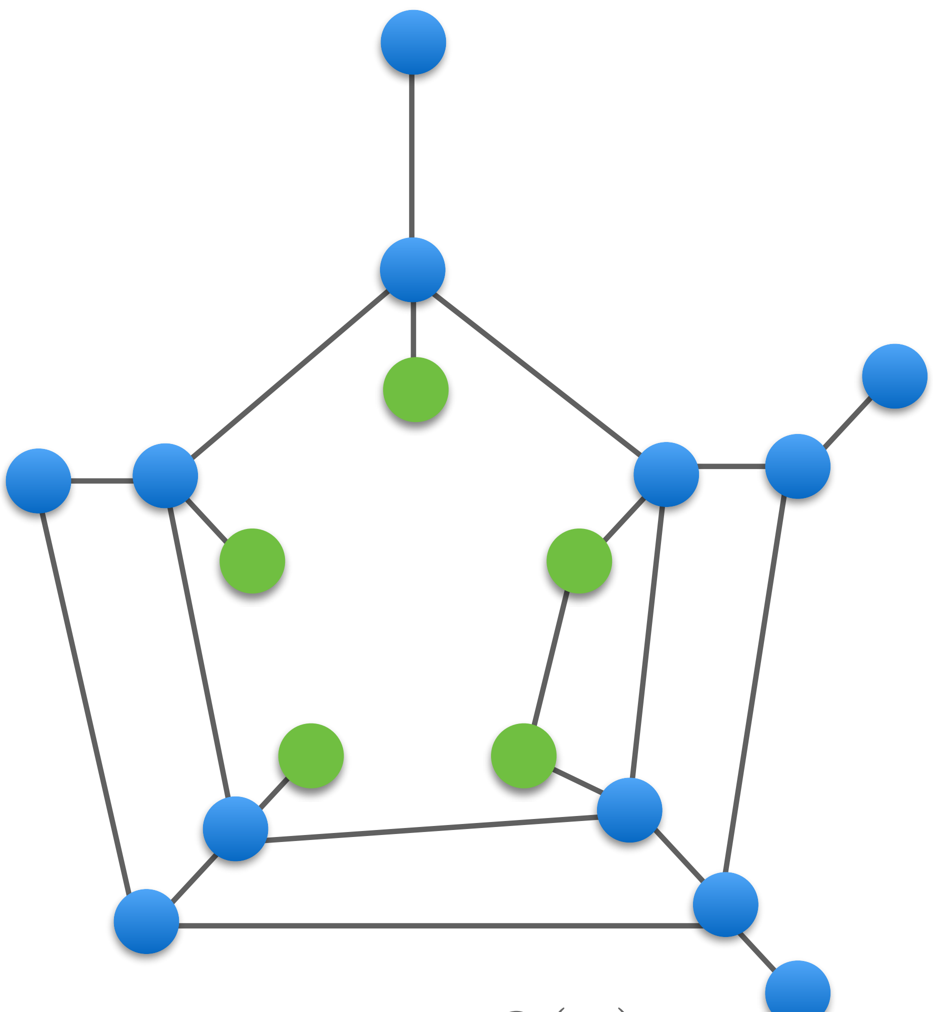


Main properties

[Improved Parallel Algorithms for Density-Based Network Clustering](#)
Mohsen Ghaffari, Silvio Lattanzi, Slobodan Mitrovic
ICML 2019

Quality of the solution

If a node has expected logarithmic coreness after sampling, its coreness number can be estimated precisely



Size of the sample

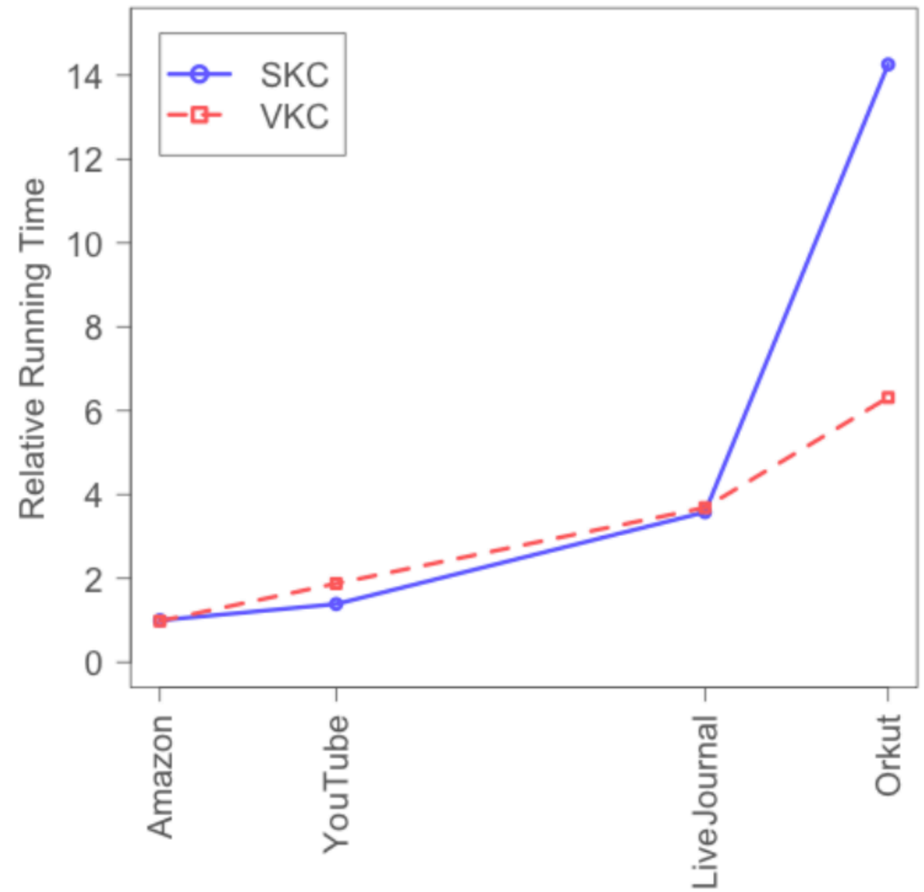
After each sample the number of edges left in the graph is almost linear in the number of nodes

In $O(\log \log n)$ rounds we get a good approximation using memory $O(\tilde{n})$

Experiments

Improved Parallel Algorithms for Density-Based Network Clustering
Mohsen Ghaffari, Silvio Lattanzi, Slobodan Mitrovic
ICML 2019

Graph	# Nodes	# Edges
Amazon	334,863	925,872
Youtube	1,134,890	2,987,624
LiveJournal	3,997,962	34,681,189
Orkut	3,072,441	117,185,083



Conclusions and Future Work

Conclusions and Future Work

- Nice model that captures many real world scenarios
- Very active area of research with many interesting results
- Many open problems with practical applications

Thanks