# MapReduce
# ML & Clustering Algorithms

Sergei Vassilvitskii

# Reminder

MapReduce:

- A trade-off between ease of use & possible parallelism

Graph Algorithms Approaches:

- Reduce input size (filtering)
- Graph specific optimizations (Pregel & Giraph)

# Today

Machine Learning

- More filtering -- reducing input size
- Machine Learning Optimizations & AllReduce

Applications:

- k-means clustering

# Machine Learning

Definition:

- Fitting a function to the data

# Machine Learning

Definition:

- Fitting a function to the data

Examples:

- Classification:
  - Data (x,y) pairs $x \in \mathbb{R}^d, y = \pm 1$
  - Temperature = 15, Pressure = 755mm, Cloud Cover = 90%. : No Rain
  - Temperature = 17, Pressure = 760mm, Cloud Cover = 75% : No Rain
  - Temperature = 23, Pressure = 766mm, Cloud Cover = 95% : Rain
  - Temperature = 19, Pressure = 740mm, Cloud Cover = 100% : ???

# Machine Learning

## Definition:

- Fitting a function to the data

## Examples:

- Classification.
- Regression:
  - Data (x,y) pairs $x \in \mathbb{R}^d, y \in \mathbb{R}$
  - Temperature = 15, Pressure = 755mm, Cloud Cover = 90%. : 1mm Rain
  - Temperature = 17, Pressure = 760mm, Cloud Cover = 75% : 0mm Rain
  - Temperature = 23, Pressure = 766mm, Cloud Cover = 95% : 9mm Rain
  - Temperature = 19, Pressure = 740mm, Cloud Cover = 100% : ???

# Machine Learning

Definition:

- Fitting a function to the data

Examples:

- Classification.
- Regression.
- Clustering:
    - Data $x \in \mathbb{R}^d$ , Goal: find a sensible grouping into $k$ groups

# Machine Learning

Definition:

- Fitting a function to the data

Examples:

- Classification.
- Regression.
- Clustering.

Today:

- Regression & Clustering

# Regression

Aarhus (yesterday)

Data:

- Temperature = 15, Pressure = 755mm, Cloud Cover = 90%. : 1mm
- Temperature = 17, Pressure = 760mm, Cloud Cover = 75% : 0mm
- Temperature = 23, Pressure = 766mm, Cloud Cover = 95% : 9mm
- Temperature = 11, Pressure = 740mm, Cloud Cover = 100% : 5mm

Matrix Form:

$$x = \begin{pmatrix} 15 & 755 & 90 \\ 17 & 760 & 75 \\ 23 & 766 & 95 \\ 11 & 740 & 100 \end{pmatrix} \qquad y = \begin{pmatrix} 1 \\ 0 \\ 9 \\ 5 \end{pmatrix}$$

Temperature Pressure Cloud Cover

# Linear Regression

$$x = \begin{pmatrix} 15 & 755 & 90 \\ 17 & 760 & 75 \\ 23 & 766 & 95 \\ 11 & 740 & 100 \end{pmatrix} \qquad y = \begin{pmatrix} 1 \\ 0 \\ 9 \\ 5 \end{pmatrix}$$

- Approximate $y$ by a linear function of $x$ :
- Example: 0.05 weight on Temperature, 0 on pressure , 0.1 on Humidity

# Linear Regression

$$x = \begin{pmatrix} 15 & 755 & 90 \\ 17 & 760 & 75 \\ 23 & 766 & 95 \\ 11 & 740 & 100 \end{pmatrix} \qquad y = \begin{pmatrix} 1 \\ 0 \\ 9 \\ 5 \end{pmatrix}$$

- Approximate $y$ by a linear function of $x$ :
- Example: 0.05 weight on Temperature, 0 on pressure , 0.1 on Humidity
- Predictions: $15 \cdot 0.05 + 0.1 \cdot 90 = 1.65$

$$17 \cdot 0.05 + 0.1 \cdot 75 = 1.6$$

$$\cdots$$

- Find $\theta$ that minimizes the squared distance: $\|x \cdot \theta - y\|^2$

# Linear Regression

$$x = \begin{pmatrix} 15 & 755 & 90 \\ 17 & 760 & 75 \\ 23 & 766 & 95 \\ 11 & 740 & 100 \end{pmatrix} \qquad y = \begin{pmatrix} 1 \\ 0 \\ 9 \\ 5 \end{pmatrix}$$

- Approximate $y$ by a linear function of $x$ :
- Example: 0.05 weight on Temperature, 0 on pressure , 0.1 on Humidity
- Predictions: $15 \cdot 0.05 + 0.1 \cdot 90 = 1.65$

$$17 \cdot 0.05 + 0.1 \cdot 75 = 1.6$$

$$\cdots$$

- Find $\theta$ that minimizes the squared distance: $\|x \cdot \theta - y\|^2$
- Very simple!
  - Is this complex enough to capture all of the data?

# Linear Regression

$$x = \begin{pmatrix} 15 & 755 & 90 \\ 17 & 760 & 75 \\ 23 & 766 & 95 \\ 11 & 740 & 100 \end{pmatrix} \qquad y = \begin{pmatrix} 1 \\ 0 \\ 9 \\ 5 \end{pmatrix}$$

- Approximate $y$ by a linear function of $x$ :
- Example: 0.05 weight on Temperature, 0 on pressure , 0.1 on Humidity
- Predictions: $15 \cdot 0.05 + 0.1 \cdot 90 = 1.65$

$$17 \cdot 0.05 + 0.1 \cdot 75 = 1.6$$

$$\cdots$$

- Find $\theta$ that minimizes the squared distance: $\|x \cdot \theta - y\|^2$
- Very simple!
  - Is this complex enough to capture all of the data?
  - Maybe if you have a lot of features

# Doing Regression

Problem:

- Examples $X \in \mathbb{R}^{n \times d}$, labels: $Y \in \mathbb{R}^{n \times 1}$
- Find a set of weights $\theta \in \mathbb{R}^{d \times 1}$ that minimizes: $\|X \cdot \theta - Y\|^2$

# Doing Regression

Problem:

- Examples $X \in \mathbb{R}^{n \times d}$, labels: $Y \in \mathbb{R}^{n \times 1}$
- Find a set of weights $\theta \in \mathbb{R}^{d \times 1}$ that minimizes: $\|X \cdot \theta - Y\|^2$

Exact Solution: $\theta = (X^T X)^{-1} X^T Y$

# Doing Regression

Problem:

- Examples $X \in \mathbb{R}^{n \times d}$, labels: $Y \in \mathbb{R}^{n \times 1}$
- Find a set of weights $\theta \in \mathbb{R}^{d \times 1}$ that minimizes: $\|X \cdot \theta - Y\|^2$

Exact Solution: $\theta = (X^T X)^{-1} X^T Y$

- If the number of examples $n$ is large..
- ...but the dimensionality $d$ is small
- Then:
  - $(X^T X) \in \mathbb{R}^{d \times d}$ and $X^T Y \in \mathbb{R}^{d \times 1}$ are small..

# Doing Regression

Problem:

- Examples $X \in \mathbb{R}^{n \times d}$, labels: $Y \in \mathbb{R}^{n \times 1}$
- Find a set of weights $\theta \in \mathbb{R}^{d \times 1}$ that minimizes: $\|X \cdot \theta - Y\|^2$

Exact Solution: $\theta = (X^T X)^{-1} X^T Y$

- If the number of examples $n$ is large..
- ...but the dimensionality $d$ is small
- Then:
  - $(X^T X) \in \mathbb{R}^{d \times d}$ and $X^T Y \in \mathbb{R}^{d \times 1}$ are small..
  - ..and therefore fit onto a single machine

# Doing Regression

Problem:

- Examples $X \in \mathbb{R}^{n \times d}$, labels: $Y \in \mathbb{R}^{n \times 1}$
- Find a set of weights $\theta \in \mathbb{R}^{d \times 1}$ that minimizes: $\|X \cdot \theta - Y\|^2$

Exact Solution: $\theta = (X^T X)^{-1} X^T Y$

- If the number of examples $n$ is large..
- ...but the dimensionality $d$ is small
- Then:
  - $(X^T X) \in \mathbb{R}^{d \times d}$ and $X^T Y \in \mathbb{R}^{d \times 1}$ are small..
  - ..and therefore fit onto a single machine

- Idea: Compute $X^T X, X^T Y$ in parallel, finish on a single machine

# Computation

Idea:

- Compute $X^T X, X^T Y$ in parallel, finish on a single machine

# Computation

Idea:

- Compute $X^T X, X^T Y$ in parallel, finish on a single machine

How hard?

- Computing Matrix-Matrix & Matrix-Vector products
- For square $\sqrt{n} \times \sqrt{n}$ matrices:
  - $O\left(\frac{n\sqrt{n}}{m\sqrt{M}}\right)$ time

Total memory

Machine memory

$O(1)$ when $m = n^{3/4}, M = n^{3/2}$

# Computation

## Idea:

- Compute $X^TX, X^TY$ in parallel, finish on a single machine

## How hard?

- Computing Matrix-Matrix & Matrix-Vector products
- For square $\sqrt{n} \times \sqrt{n}$ matrices:
  - $O\left(\frac{n\sqrt{n}}{m\sqrt{M}}\right)$ time

Total memory

$O(1)$ when $m = n^{3/4}, M = n^{3/2}$

  - and!

Machine memory

  - $\Omega\left(\frac{n\sqrt{n}}{m\sqrt{M}}\right)$

# How far can you go?

ML Theory

- Statistical query model
- Interact with data only via some $f(x, y)$ that's averaged over all of the examples.

$$f(X, Y) = \frac{1}{n} \sum_{(x,y) \in (X \times Y)} f(x, y)$$

- This is trivial to parallelize

# Statistical Query Model

What can you do using the statistical query model?

- Linear Regression
- Naive Bayes
- Logistic Regression
- Neural Networks
- Principle Component Analysis (PCA)

# Statistical Query Model

What can you do using the statistical query model?

But required runtime per step....

- Linear Regression $O(d^2)$
- Naive Bayes $O(d)$
- Logistic Regression $O(d^2)$
- Neural Networks $O(d)$
- Principle Component Analysis (PCA) $O(d^2)$

# Statistical Query Model

What can you do using the statistical query model?

But required runtime per step....

- Linear Regression $O(d^2)$
- Naive Bayes $O(d)$
- Logistic Regression $O(d^2)$
- Neural Networks $O(d)$
- Principle Component Analysis (PCA) $O(d^2)$

...Applicable only to low dimensional spaces

# Large Dimensionality

What if the dimension is too high?

Goal Minimize: $J(\theta) = \|X \cdot \theta - Y\|^2$

# Large Dimensionality

What if the dimension is too high?

Goal Minimize: $J(\theta) = \|X \cdot \theta - Y\|^2$

- greedy solution: gradient descent

$$\theta_{\text{new}} = \theta_{\text{old}} + \alpha \cdot \nabla J(\theta_{\text{old}})$$

- Single example gradient:

$$\frac{\partial}{\partial \theta_j} J(\theta) = (y - x \cdot \theta) \cdot x_j$$

# Batch Gradient Descent

Given examples:

- 1. Compute gradient for every example for every coordinate

$$\frac{\partial}{\partial \theta_j} J(\theta) = (y - x \cdot \theta) \cdot x_j$$

- Easy to parallelize!

# Batch Gradient Descent

Given examples:

- 1. Compute gradient for every example for every coordinate

$$\frac{\partial}{\partial \theta_j} J(\theta) = (y - x \cdot \theta) \cdot x_j$$

  - Easy to parallelize!

- 2. Update $\theta$ :

$$\theta_{\mathrm{new}(j)} = \theta_{\mathrm{old}(j)} + \alpha \sum_{i=1}^{n} (y^{(i)} - x^{(i)} \cdot \theta) \cdot x_j^{(i)}$$

# Batch Gradient Descent

Given examples:

- 1. Compute gradient for every example for every coordinate

$$\frac{\partial}{\partial \theta_j} J(\theta) = (y - x \cdot \theta) \cdot x_j$$

  - Easy to parallelize!

- 2. Update $\theta$ :

$$\theta_{\text{new}(j)} = \theta_{\text{old}(j)} + \alpha \sum_{i=1}^{n} (y^{(i)} - x^{(i)} \cdot \theta) \cdot x_j^{(i)}$$

- 3. Repeat until convergence
  - Will converge given mild conditions on $\alpha$

# Performance

Batch Gradient Descent in MR:

- Easy to write
- But requires many many rounds to converge...
- ...this is inefficient in MapReduce
- Remember, aimed for $O(1)$ rounds.

# Performance

Batch Gradient Descent in MR:

- Easy to write
- But requires many many rounds to converge...
- ...this is inefficient in MapReduce.
- Remember, aimed for $O(1)$ rounds.

Same problem exists sequentially:

- Batch Gradient Descent looks at all examples in every round!

# Sequential Improvement

What if we update the gradient after every example

- Read one example: $(x^{(i)}, y^{(i)})$

# Sequential Improvement

What if we update the gradient after every example

- Read one example: $(x^{(i)}, y^{(i)})$
- Update the parameter: $\theta_{\text{new}(j)} = \theta_{\text{old}(j)} + \alpha(y^{(i)} - x^{(i)} \cdot \theta) \cdot x_j^{(i)}$

# Sequential Improvement

What if we update the gradient after every example

- Read one example: $(x^{(i)}, y^{(i)})$
- Update the parameter: $\theta_{\mathrm{new}(j)} = \theta_{\mathrm{old}(j)} + \alpha(y^{(i)} - x^{(i)} \cdot \theta) \cdot x_j^{(i)}$
- Repeat until changes are minor
- Again, converges to somewhere near the local minimum

Known as Stochastic Gradient Descent

# Stochastic GS & All-Reduce

How to parallelize stochastic gradient descent?

- Main Loop:

$$\theta_{\text{new}(j)} = \theta_{\text{old}(j)} + \alpha(y^{(i)} - x^{(i)} \cdot \theta) \cdot x_j^{(i)}$$

- First compute $y^{(i)} - x^{(i)} \cdot \theta$
- Then: perform the update

Requires all coordinates

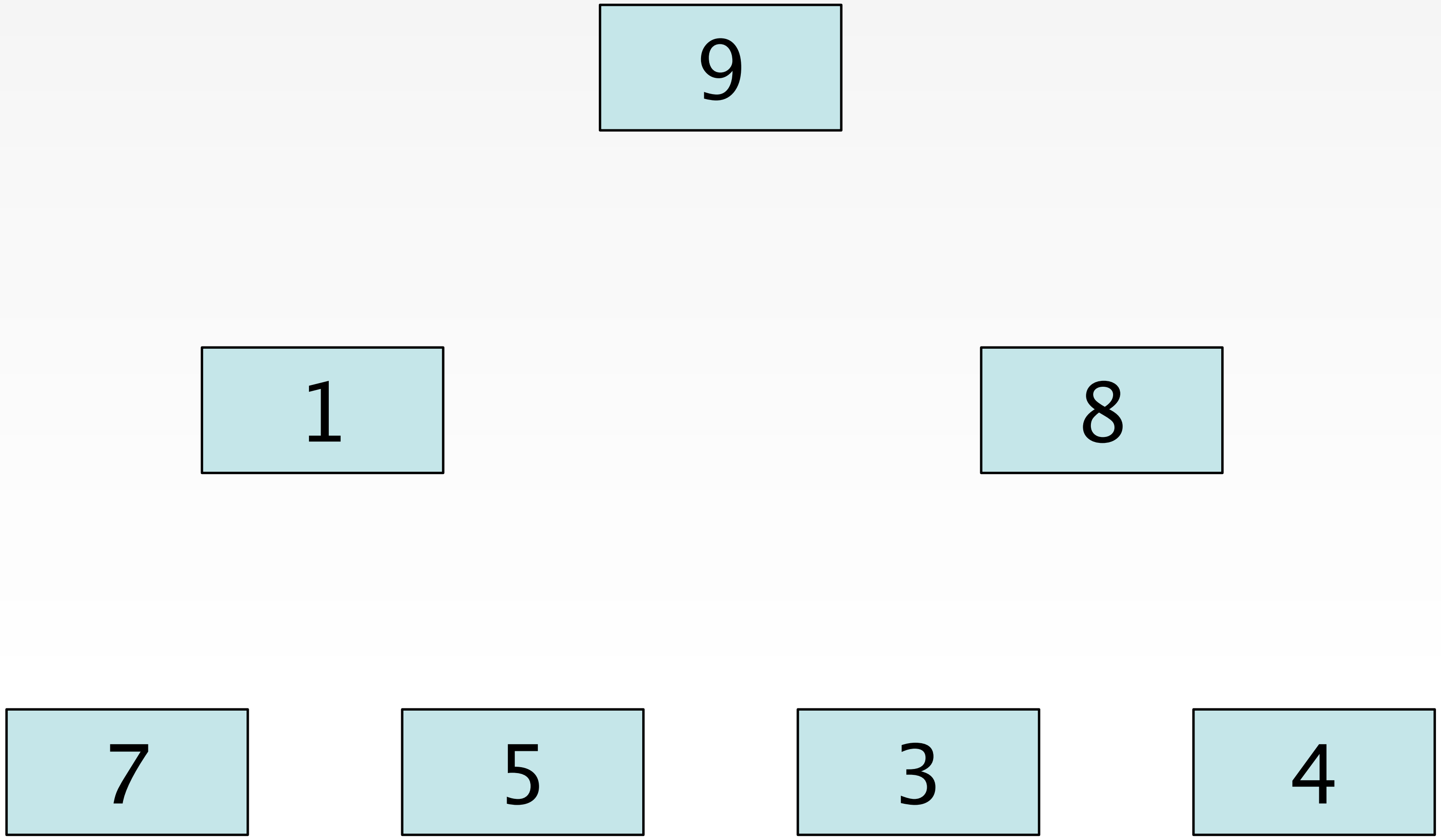
Easy to parallelize by coordinate

# All-Reduce

Optimizes taking a sum & propagating the updates.

# All-Reduce

Optimizes taking a sum & propagating the updates.



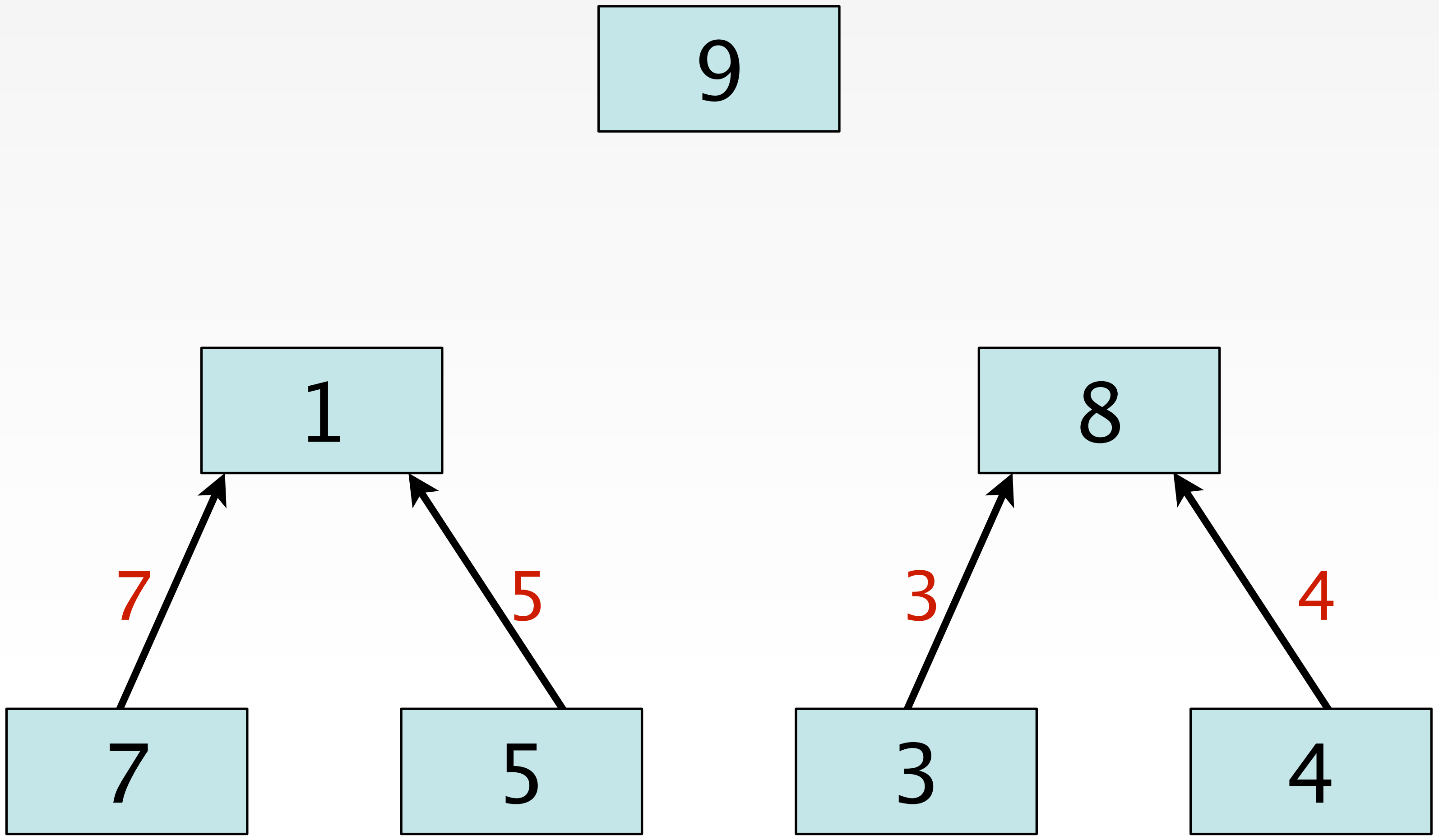
9

1 8

7 5 3 4

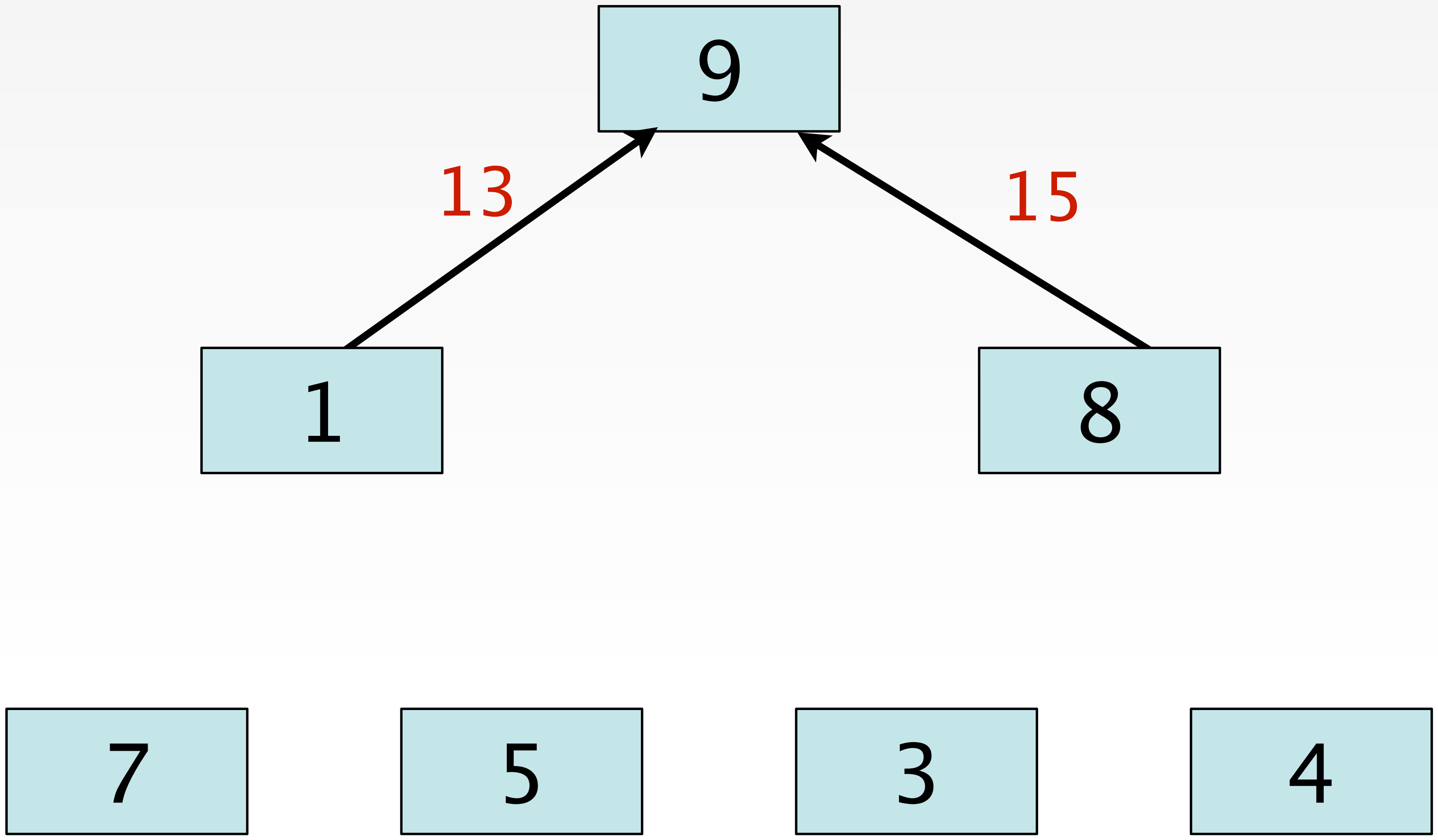
# All-Reduce

Optimizes taking a sum & propagating the updates.



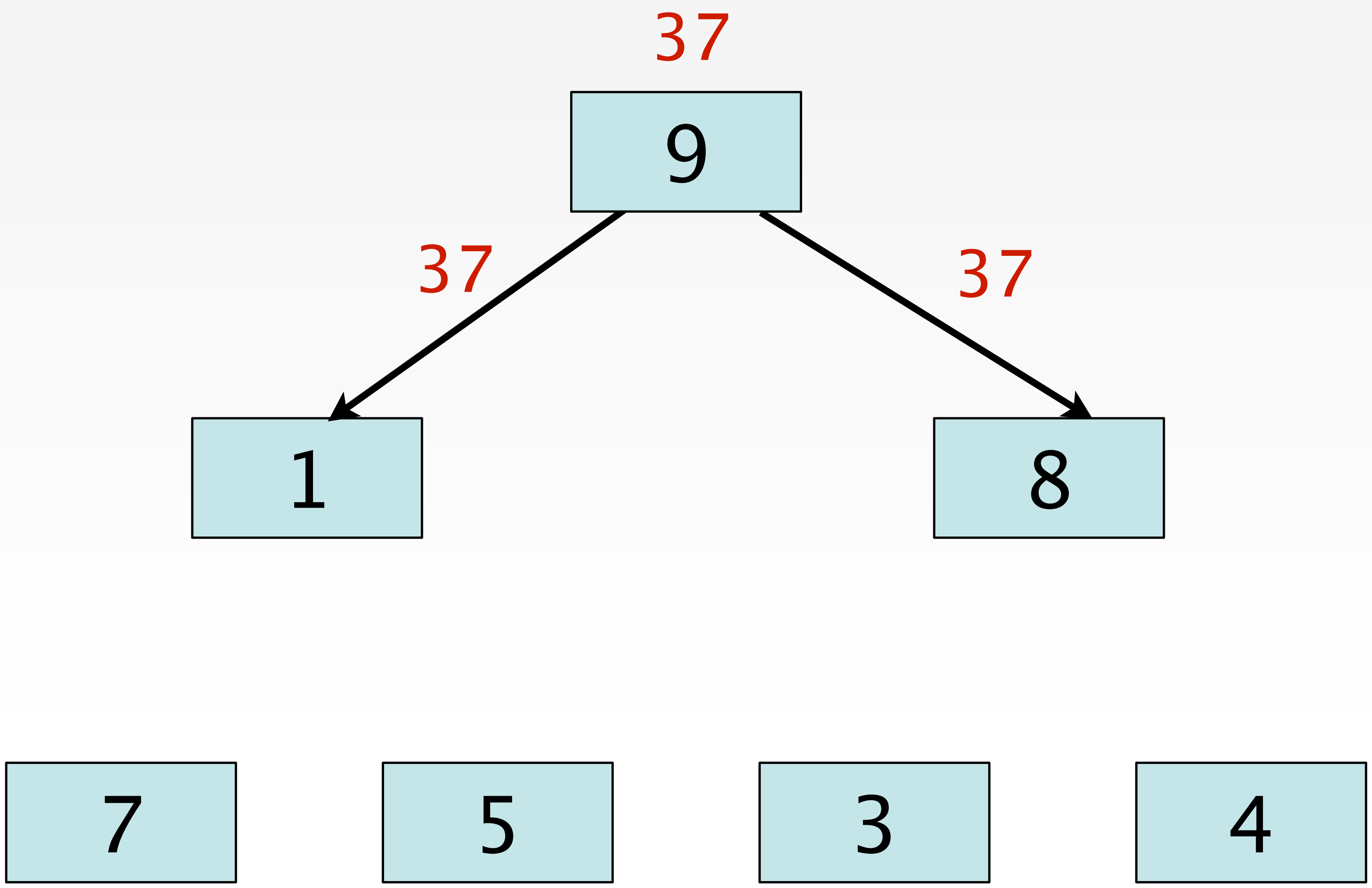
# All-Reduce

Optimizes taking a sum & propagating the updates.



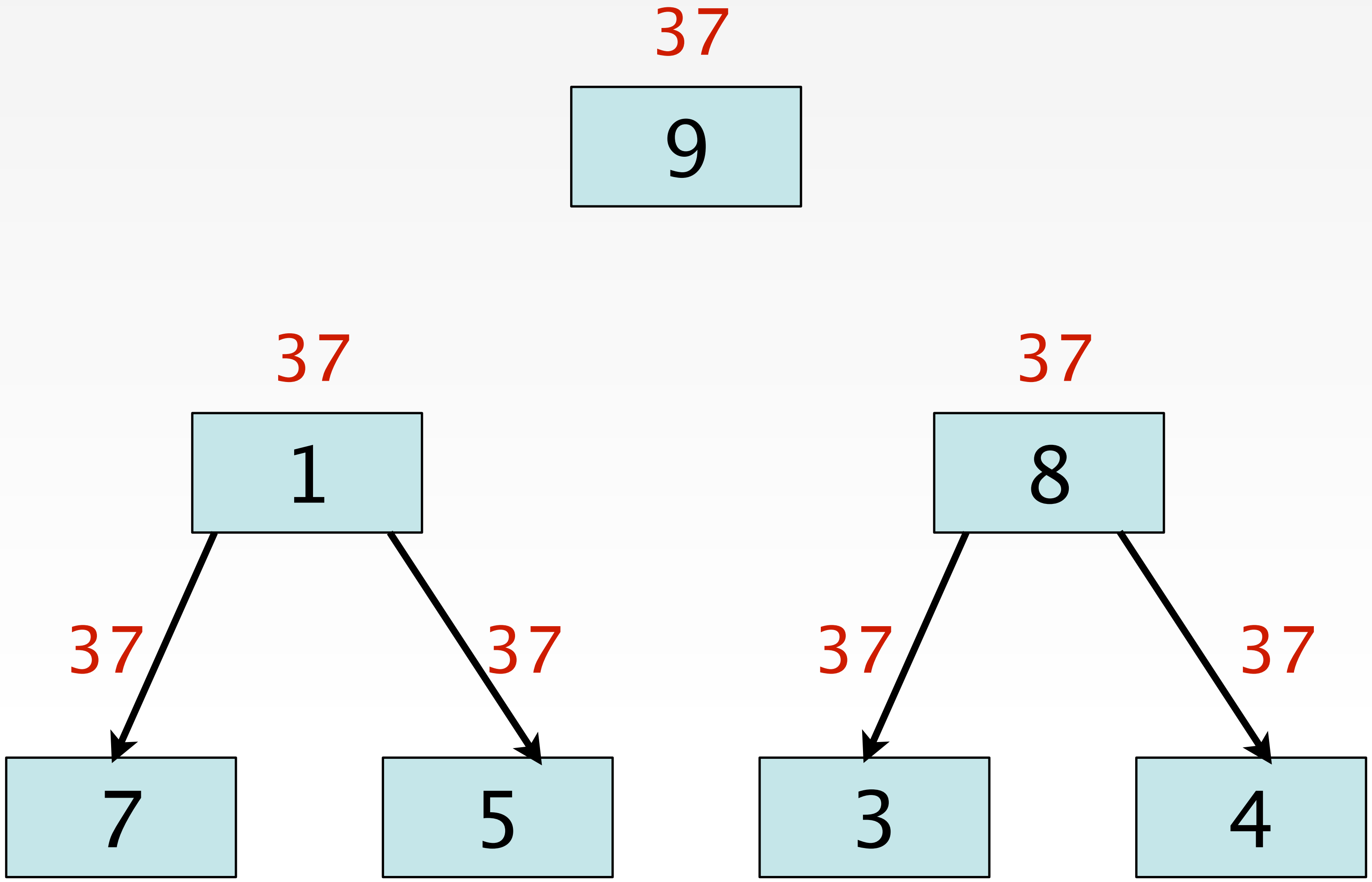
# All-Reduce

Optimizes taking a sum & propagating the updates.



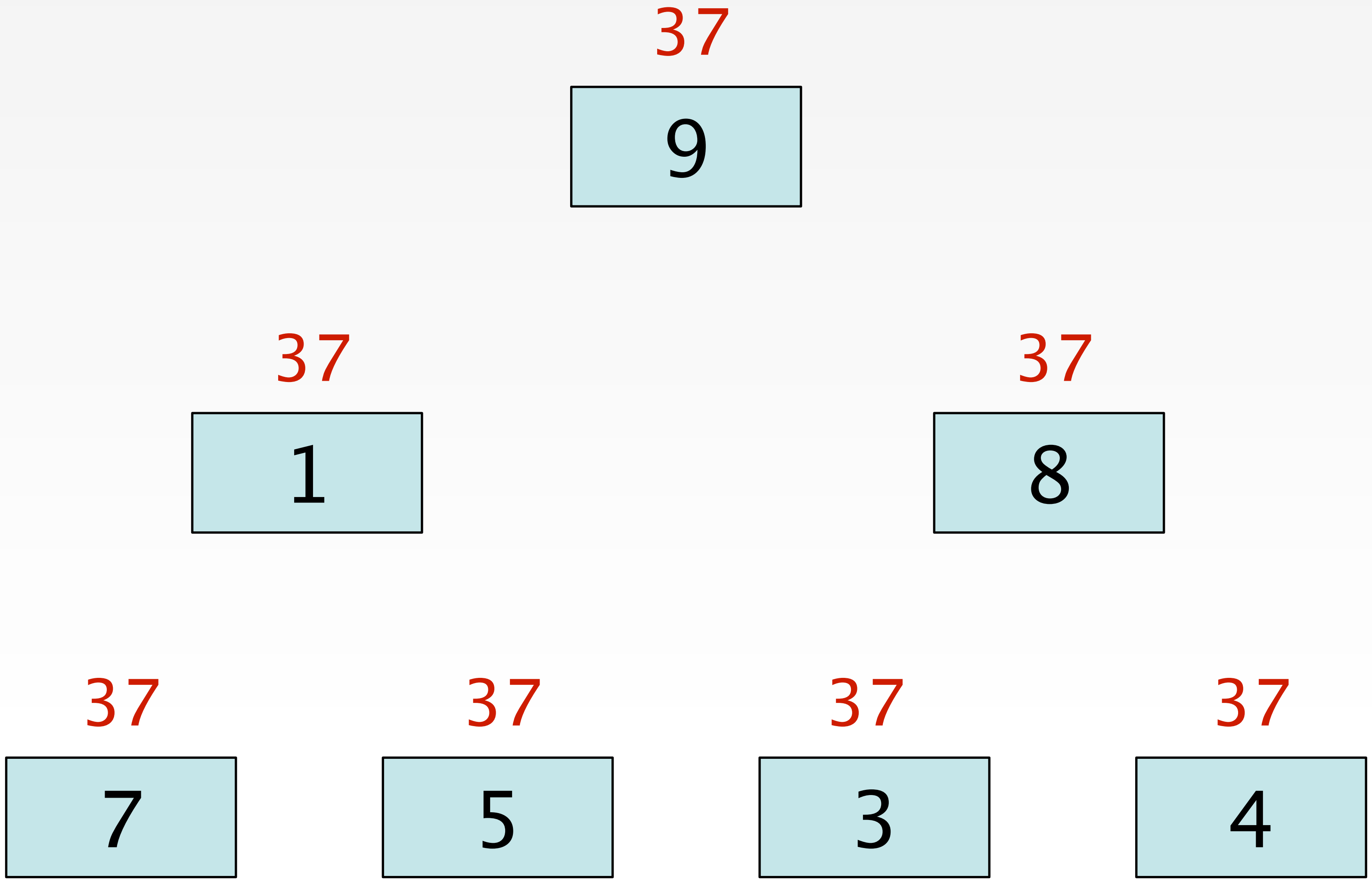
# All-Reduce

Optimizes taking a sum & propagating the updates.



# All-Reduce

Optimizes taking a sum & propagating the updates.



# All-Reduce

Optimizes taking a sum & propagating the updates

- Very optimized !
- Can get a significant speed up over straightforward Hadoop
- Mostly maintain fault tolerance given by Hadoop
- Pipelining means nodes are never idle
  - Delay propagation of gradient by a few rounds

# All-Reduce

Optimizes taking a sum & propagating the updates

- Very optimized !
- Can get a significant speed up over straightforward Hadoop
- Mostly maintain fault tolerance given by Hadoop
- Pipelining means nodes are never idle
  - Delay propagation of gradient by a few rounds
- Gets good results!
- Can be made to work with other Statistical Query Algorithms

# Regression Overview

Two approaches:

- Exact Computation
  - Works only if the dimension is small (quadratic algorithms on dimension allowed)
- Streaming style computation
  - Works even if dimension is large
  - AllReduce makes MapReduce more scalable

# Today

Machine Learning

- More filtering -- reducing input size
- Machine Learning Optimizations & AllReduce

Applications:

- k-means clustering

# Clustering

Clustering:

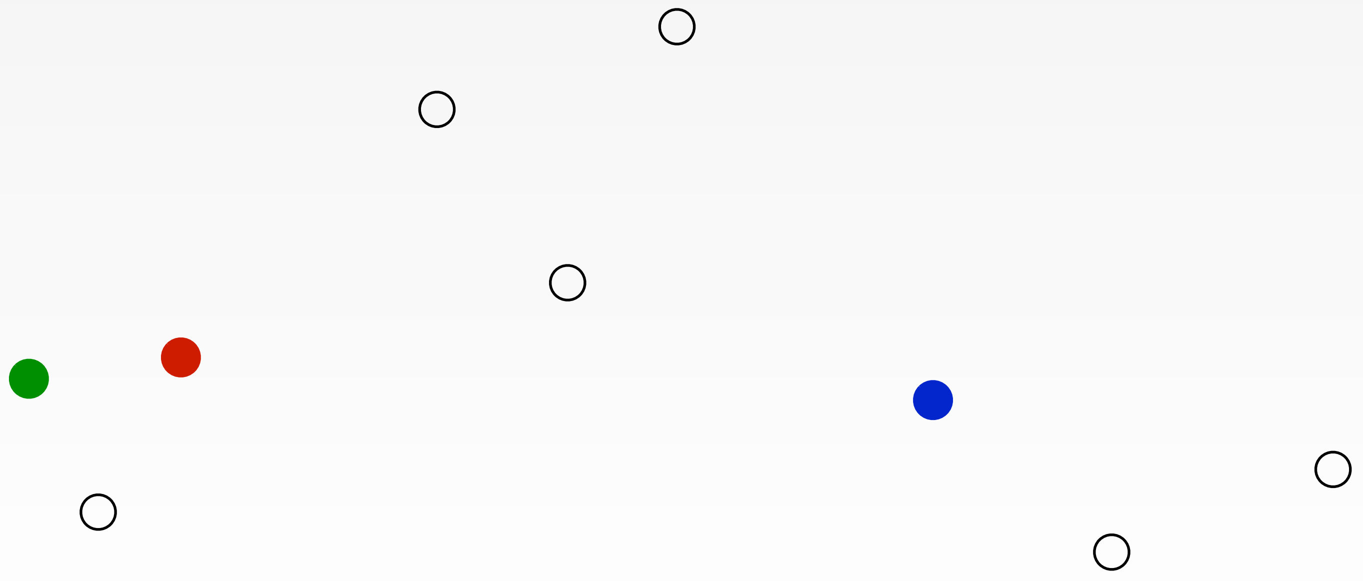
- Group similar items together

One of the oldest problems in CS:

- Thousands of papers
- Hundreds of algorithms

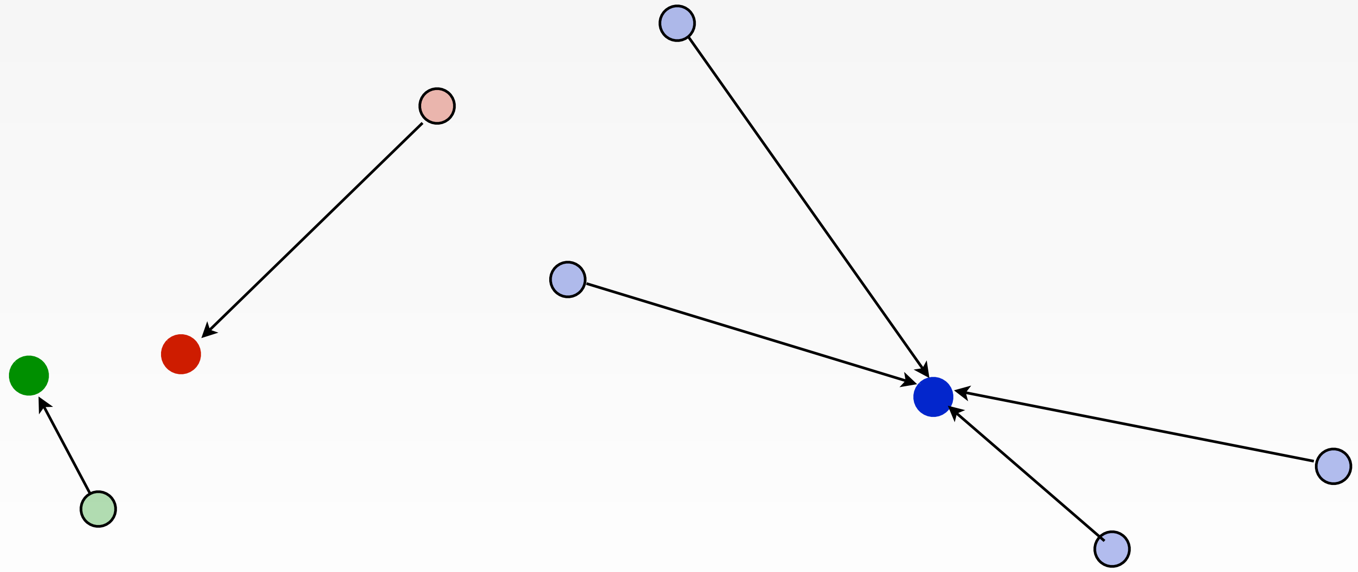
# Lloyd’s Method: k-means

Initialize with random clusters



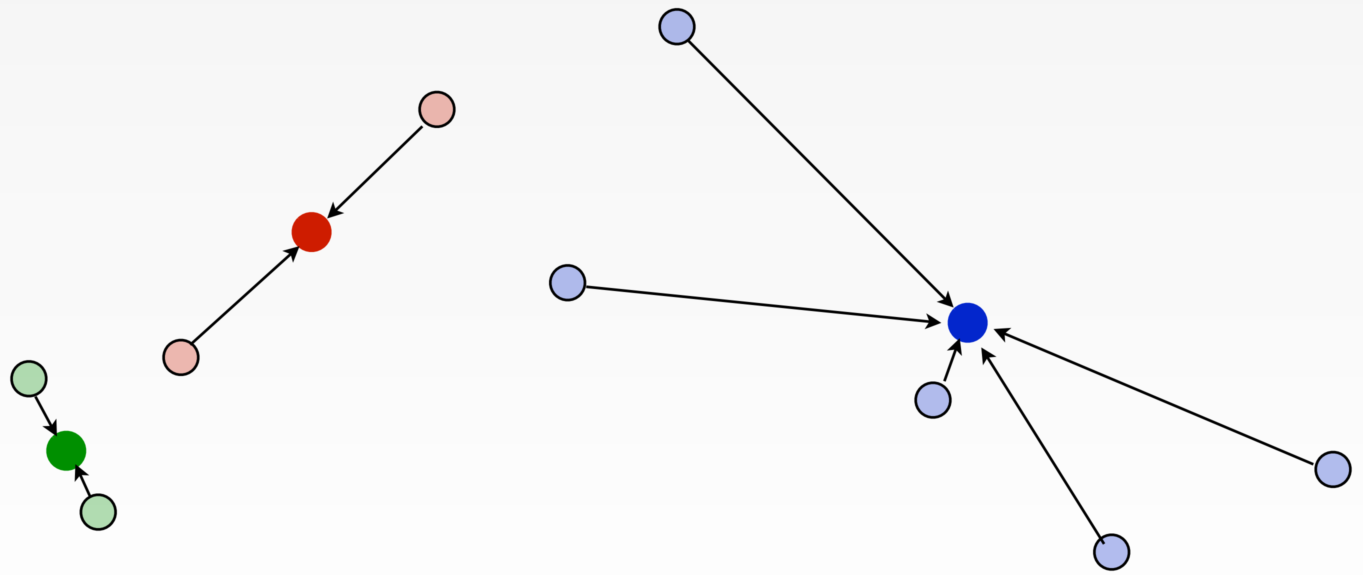
# Lloyd’s Method: k-means

Assign each point to nearest center



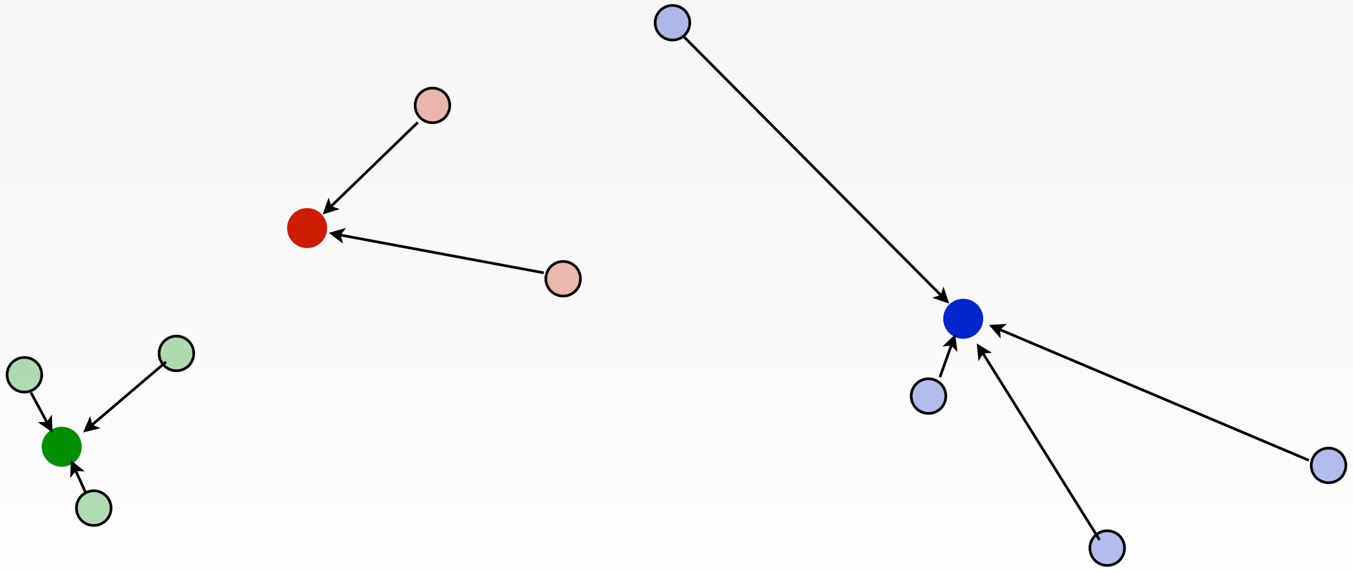
# Lloyd’s Method: k-means

Recompute optimum centers (means)



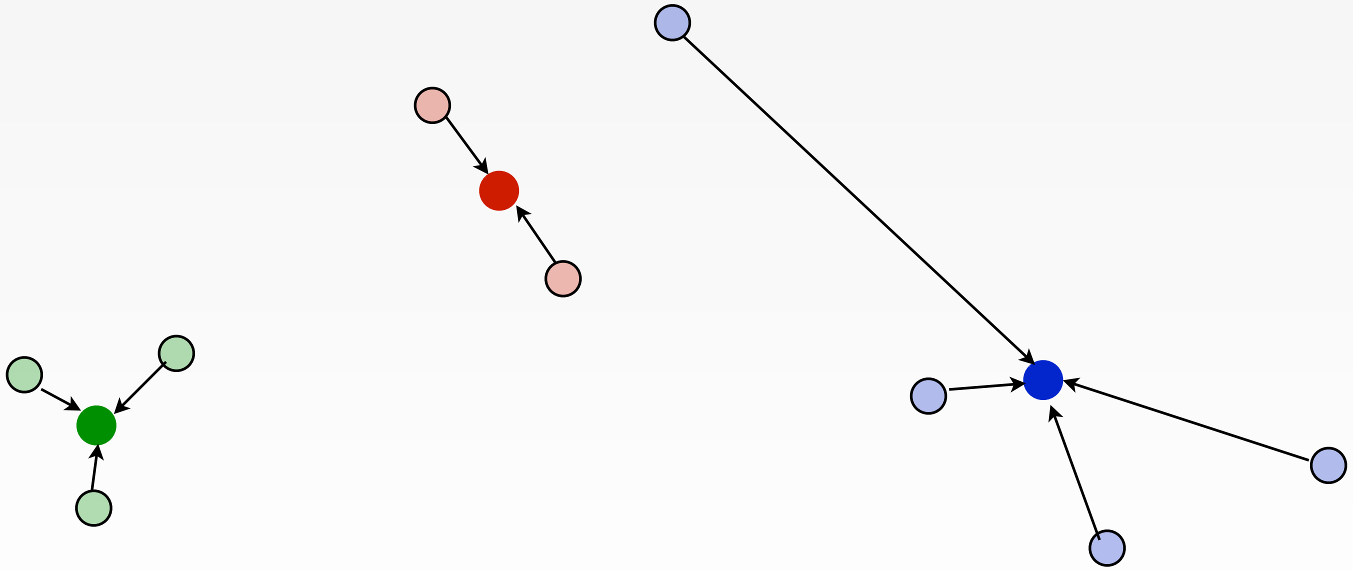
# Lloyd’s Method: k-means

Repeat: Assign points to nearest center



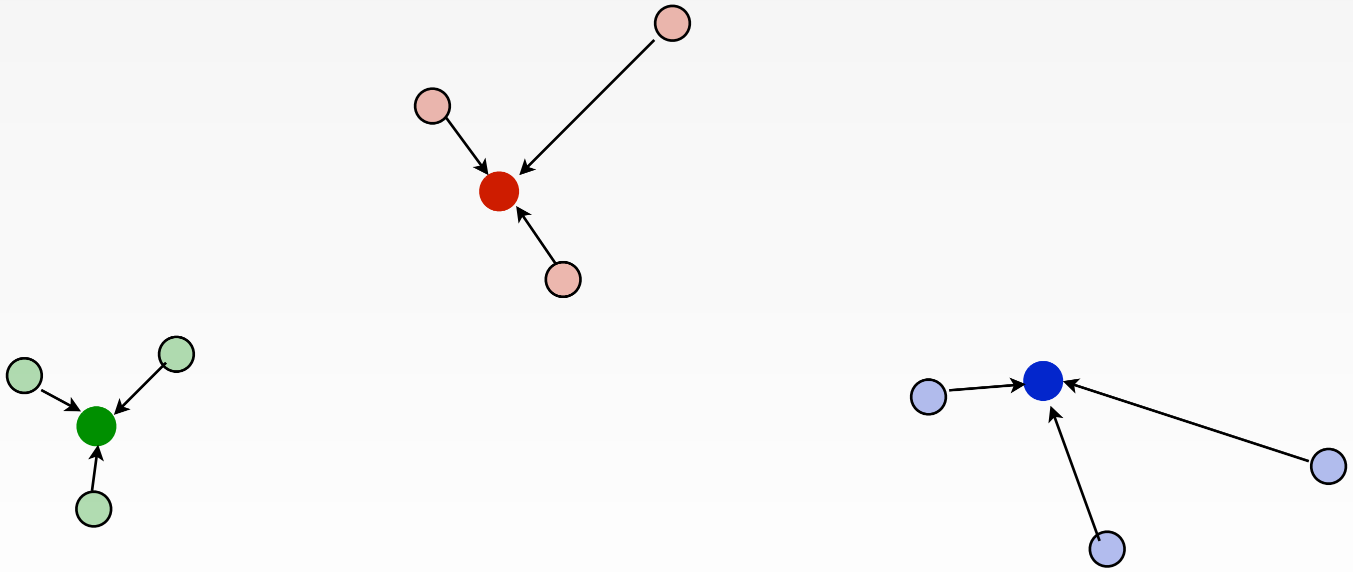
# Lloyd’s Method: k-means

Repeat: Recompute centers



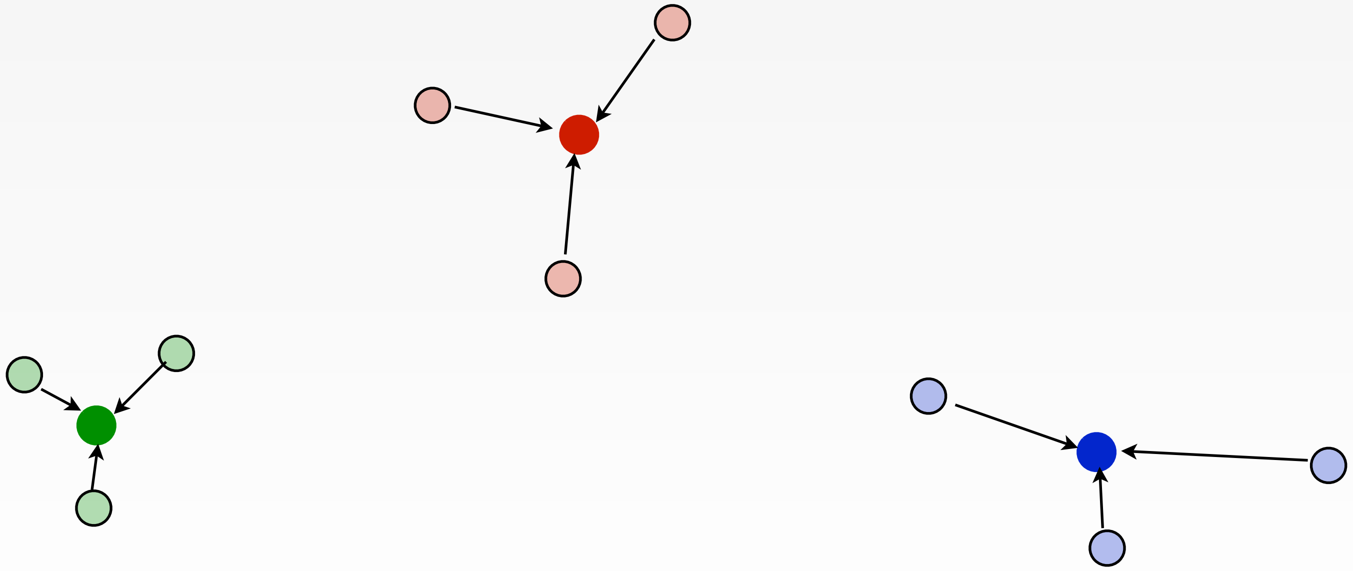
# Lloyd’s Method: k-means

Repeat...



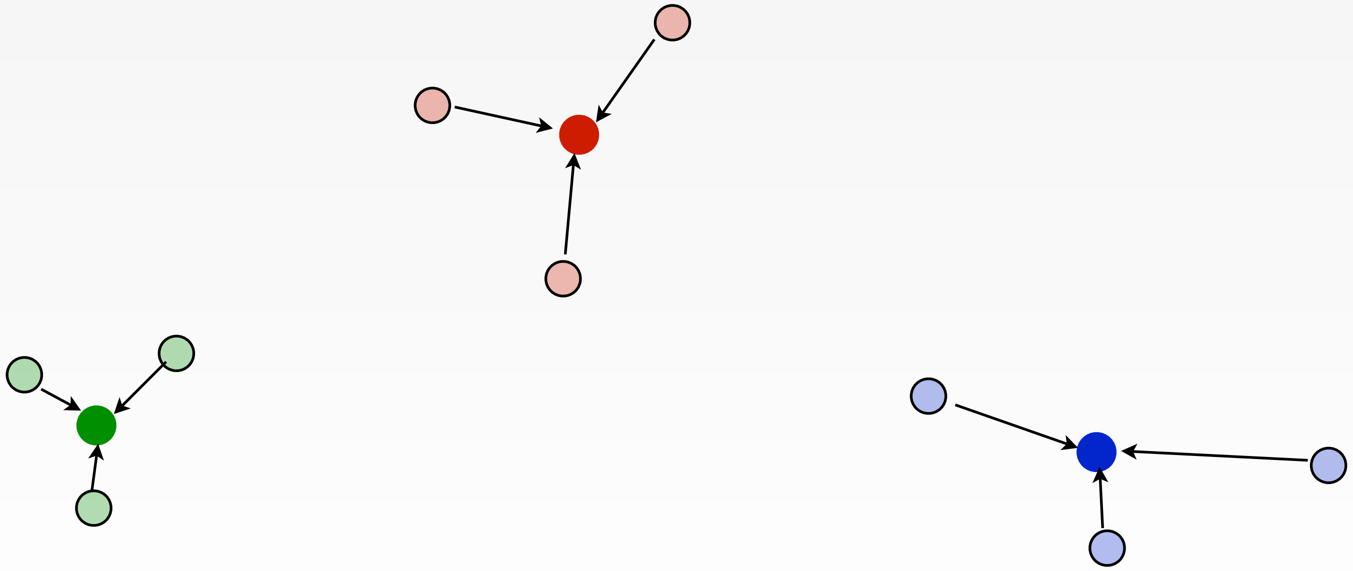
# Lloyd’s Method: k-means

Repeat...Until clustering does not change



# Lloyd’s Method: k-means

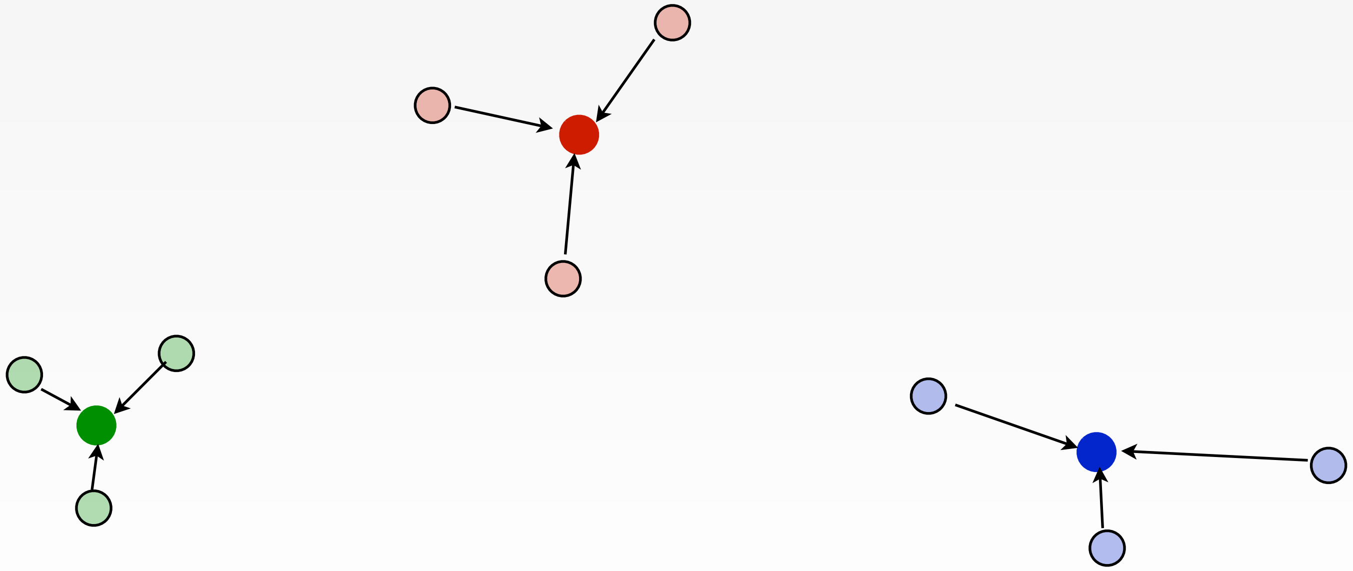
Repeat...Until clustering does not change



Total error reduced at every step - guaranteed to converge.

# Lloyd's Method: k-means

Repeat...Until clustering does not change



Total error reduced at every step - guaranteed to converge.

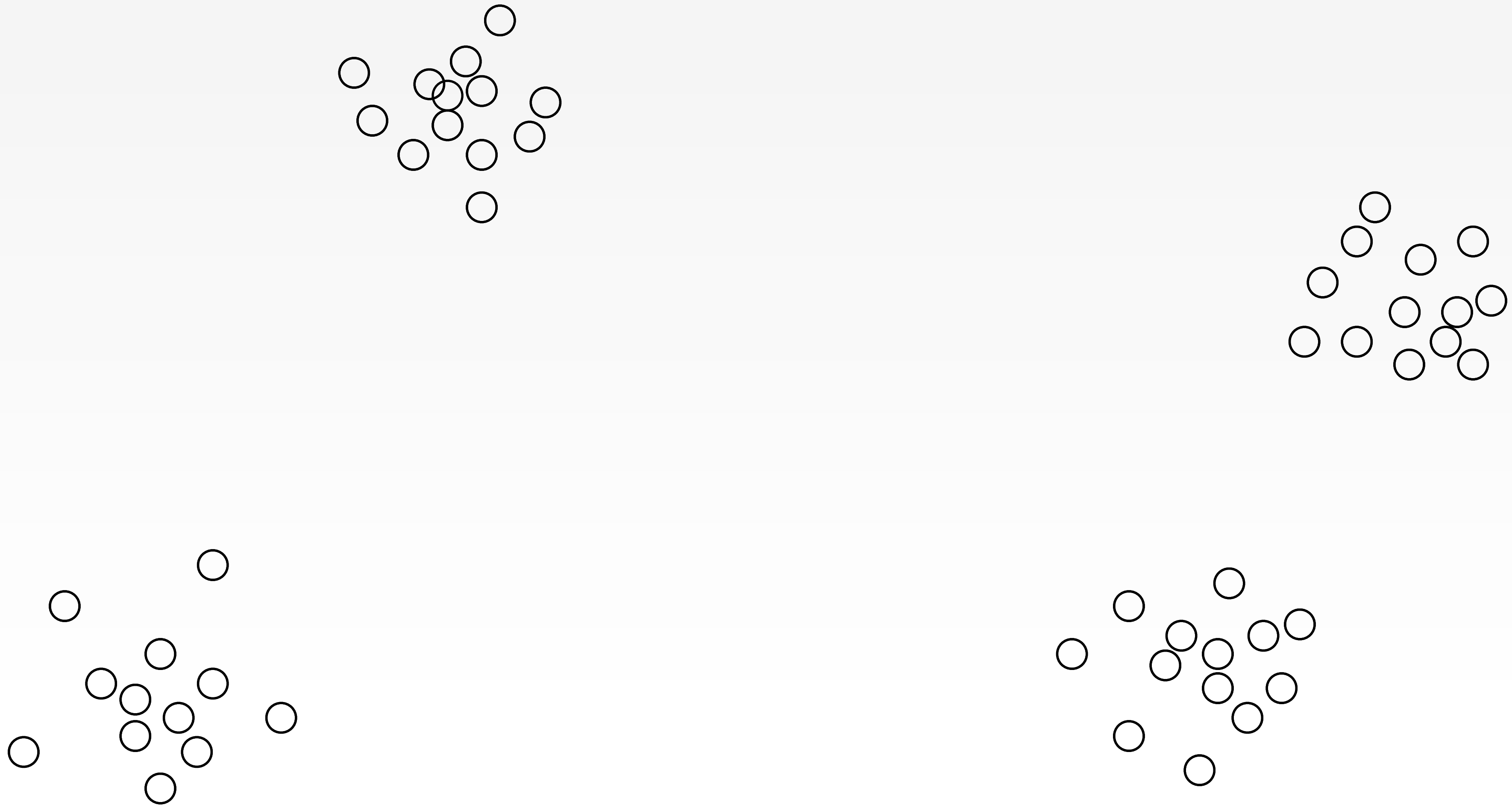
Minimizes: $\phi(X,C) = \sum_{x \in X} d(x,C)^2$

# k-means Initialization

Random?

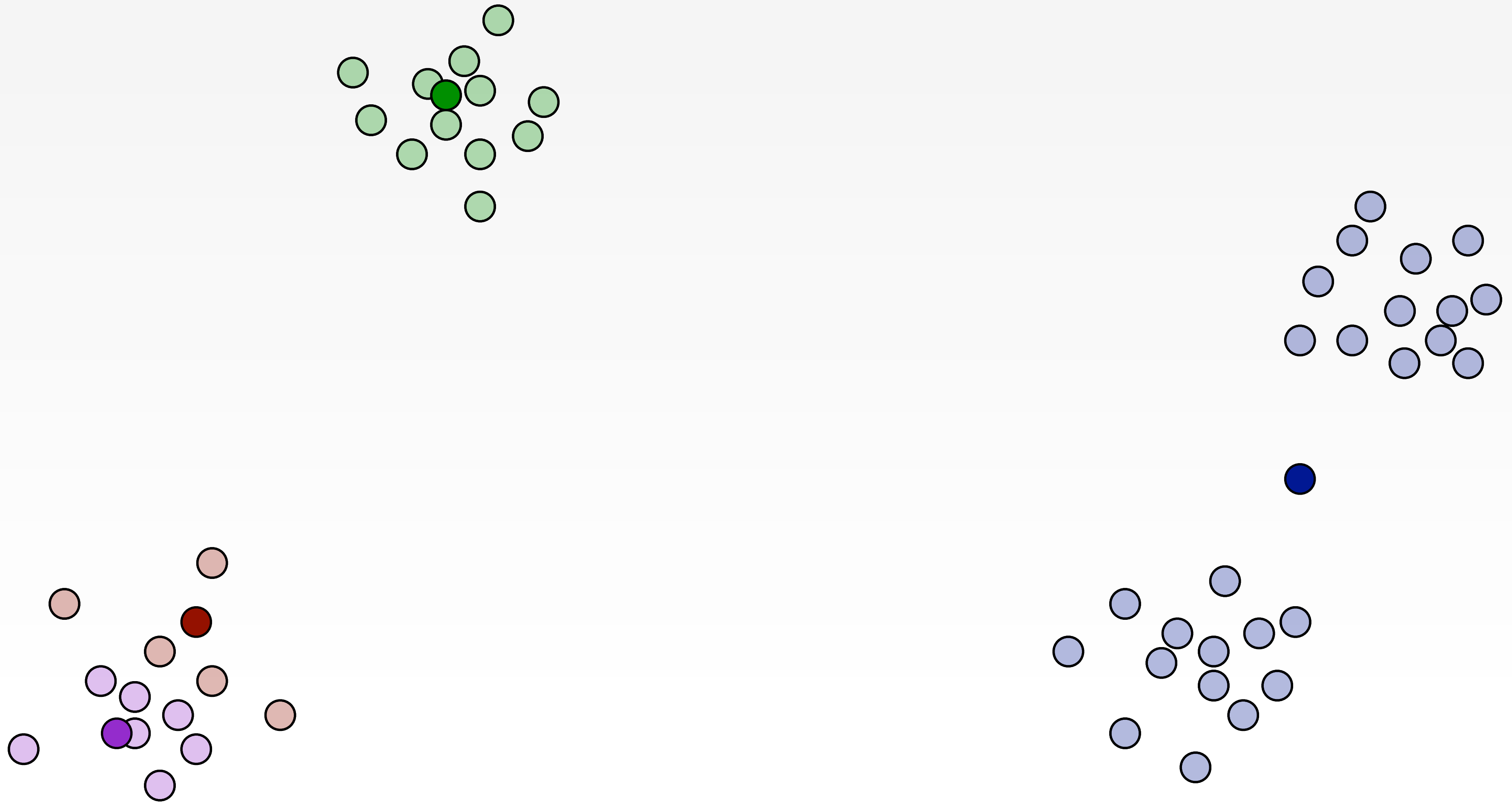
# k-means Initialization

Random?



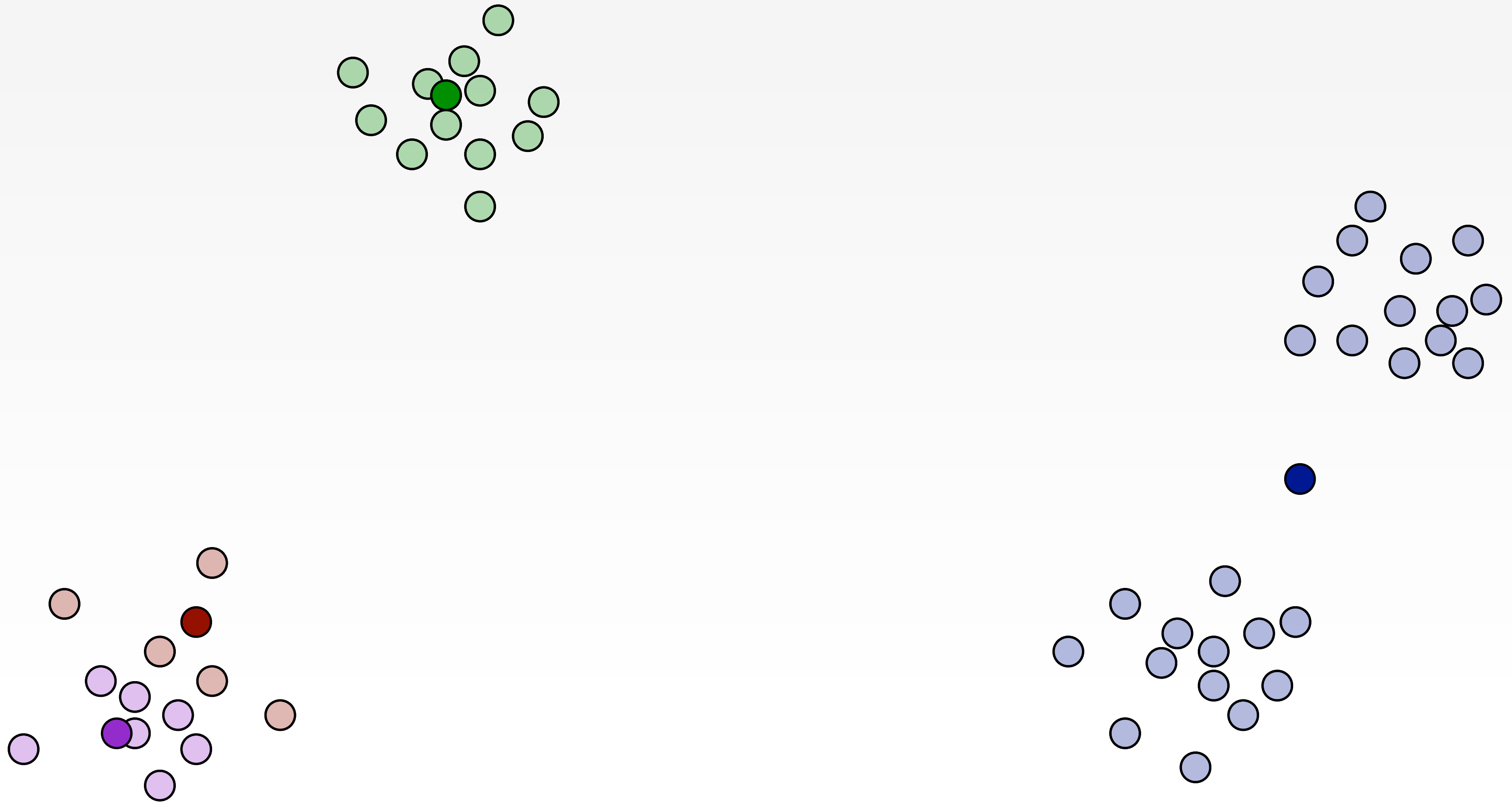
# k-means Initialization

Random? A bad idea



# k-means Initialization

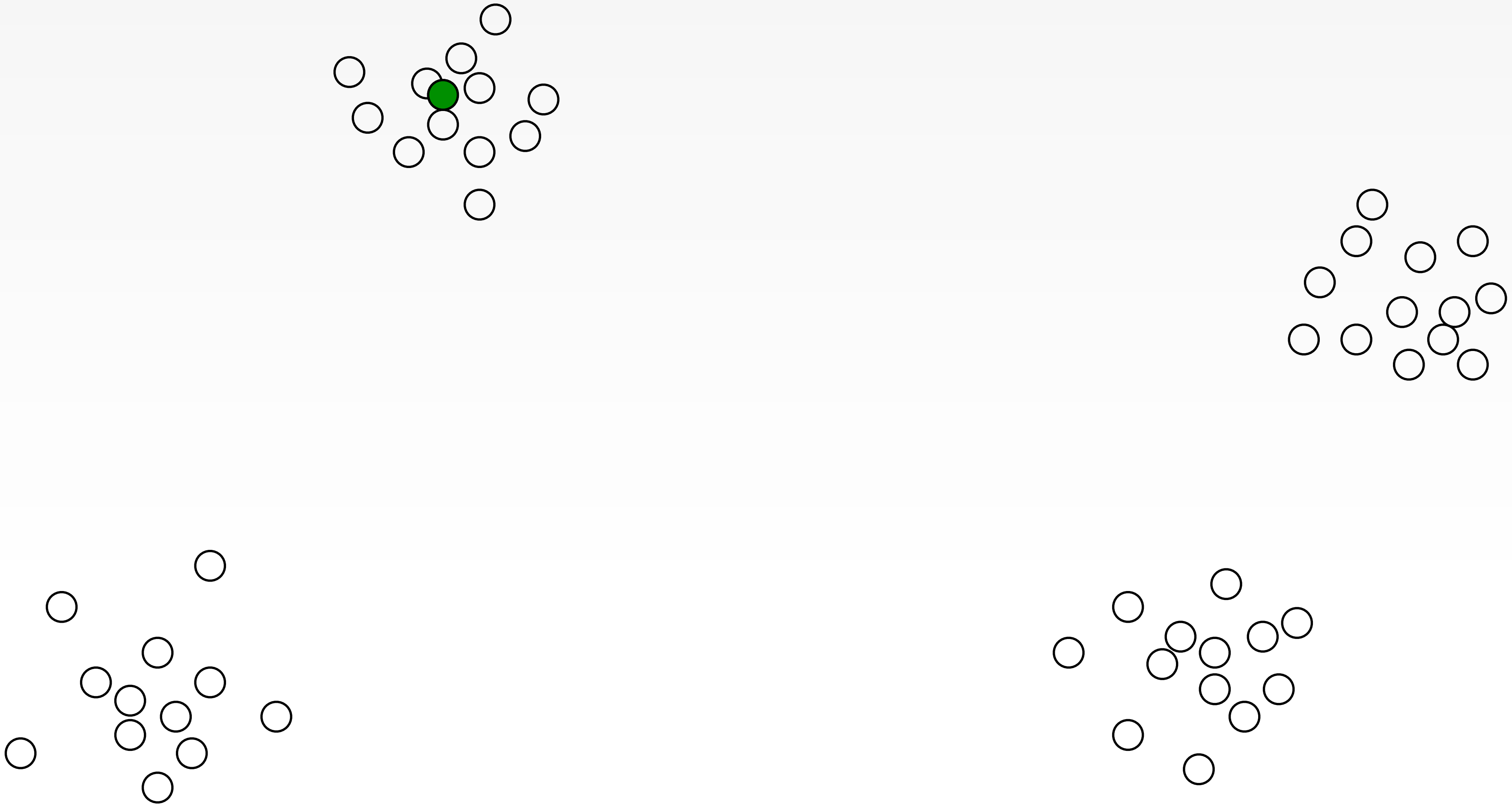
Random? A bad idea



Even with many random restarts!

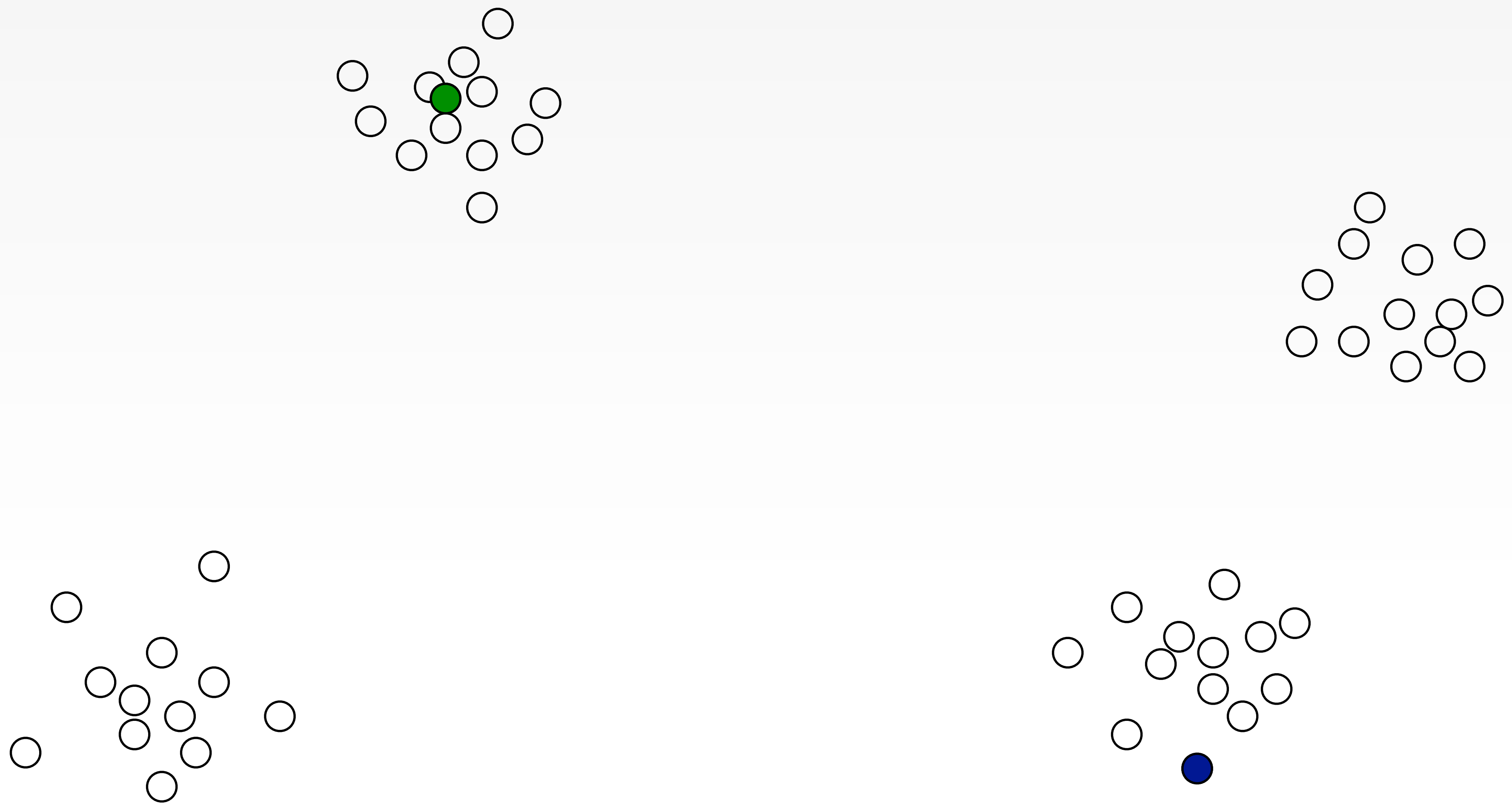
# Easy Fix

Select centers using a furthest point algorithm (2-approximation to k-Center clustering).



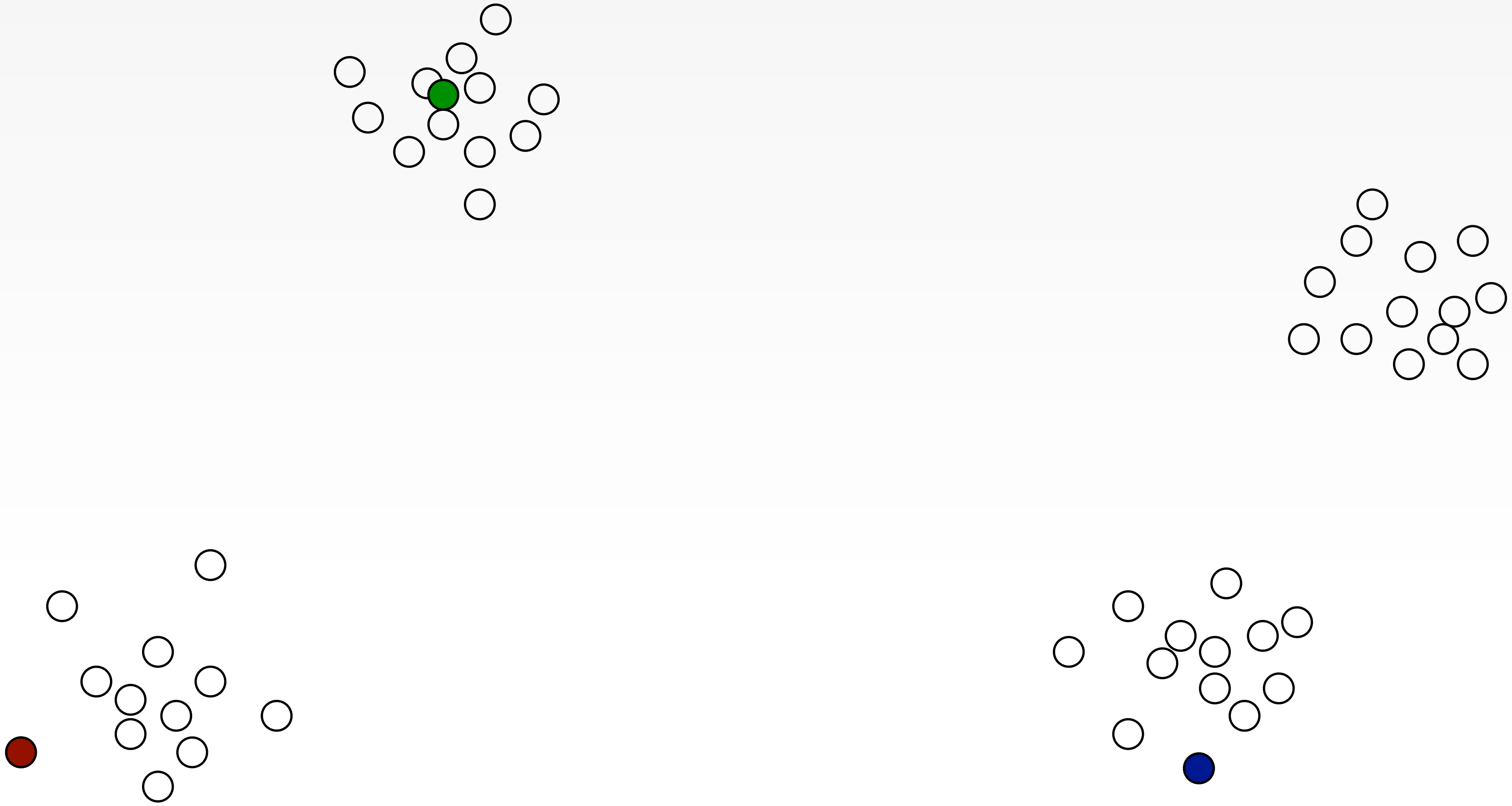
# Easy Fix

Select centers using a furthest point algorithm (2-approximation to k-Center clustering).



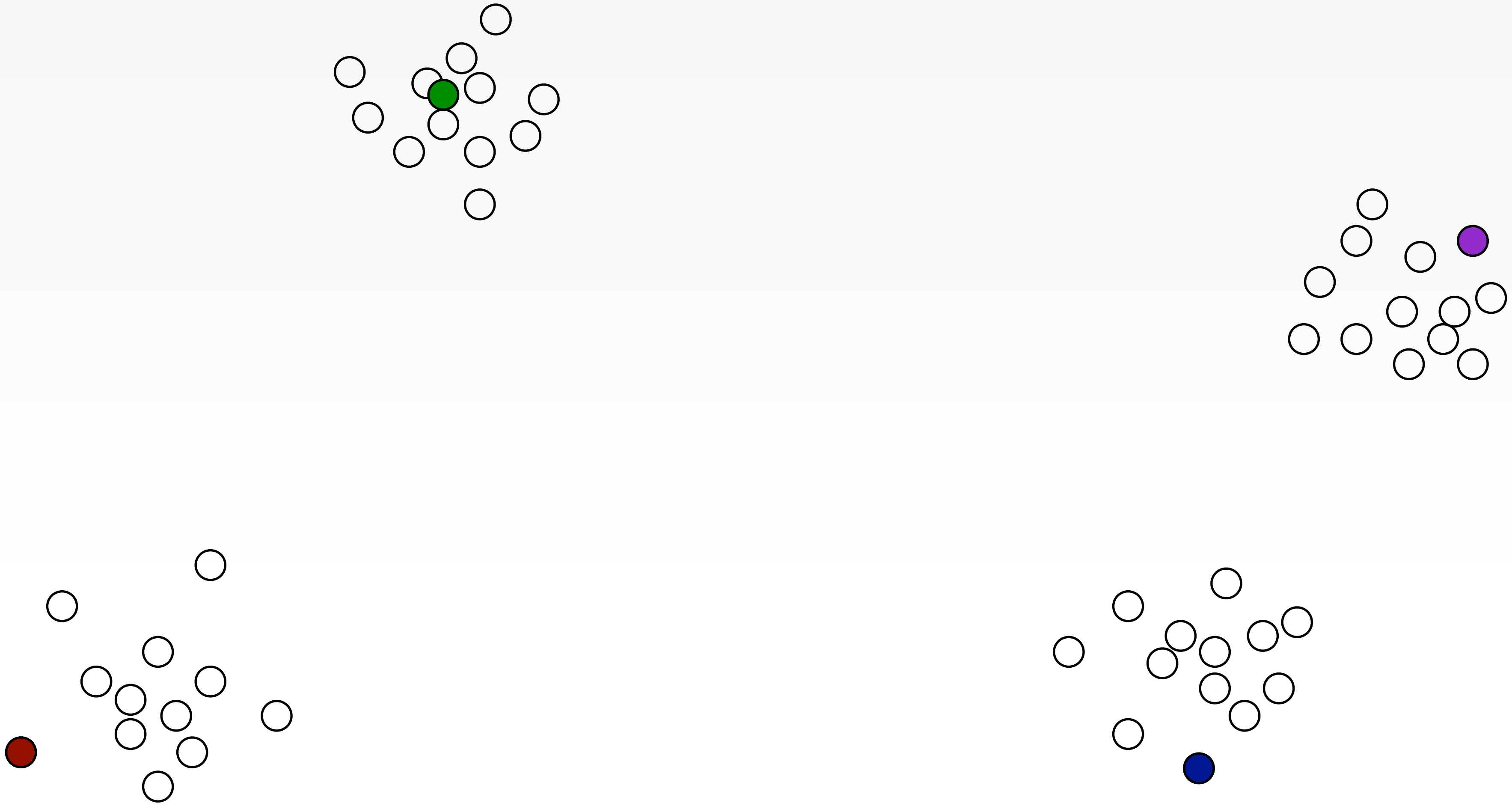
# Easy Fix

Select centers using a furthest point algorithm (2-approximation to k-Center clustering).



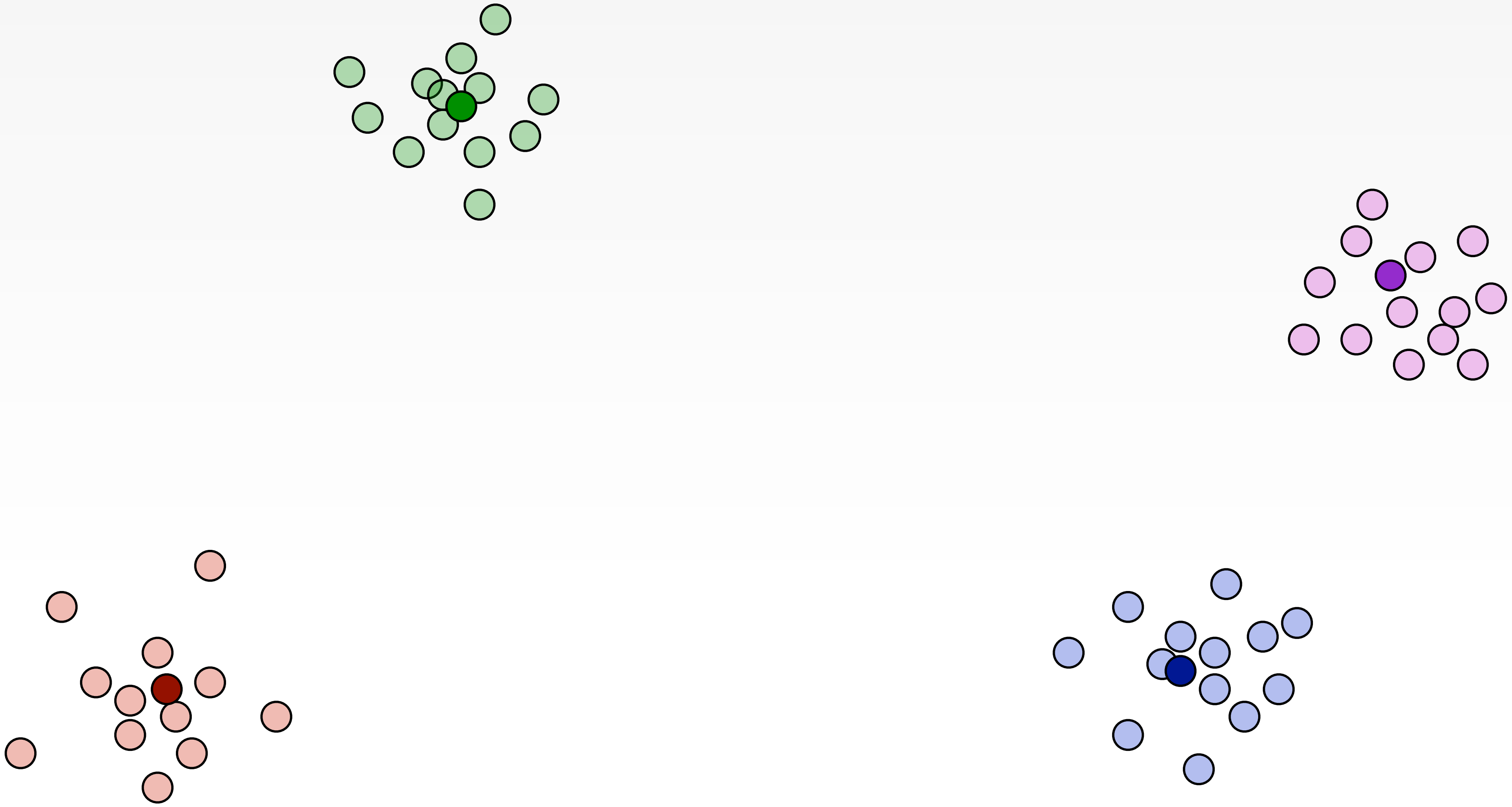
# Easy Fix

Select centers using a furthest point algorithm (2-approximation to k-Center clustering).

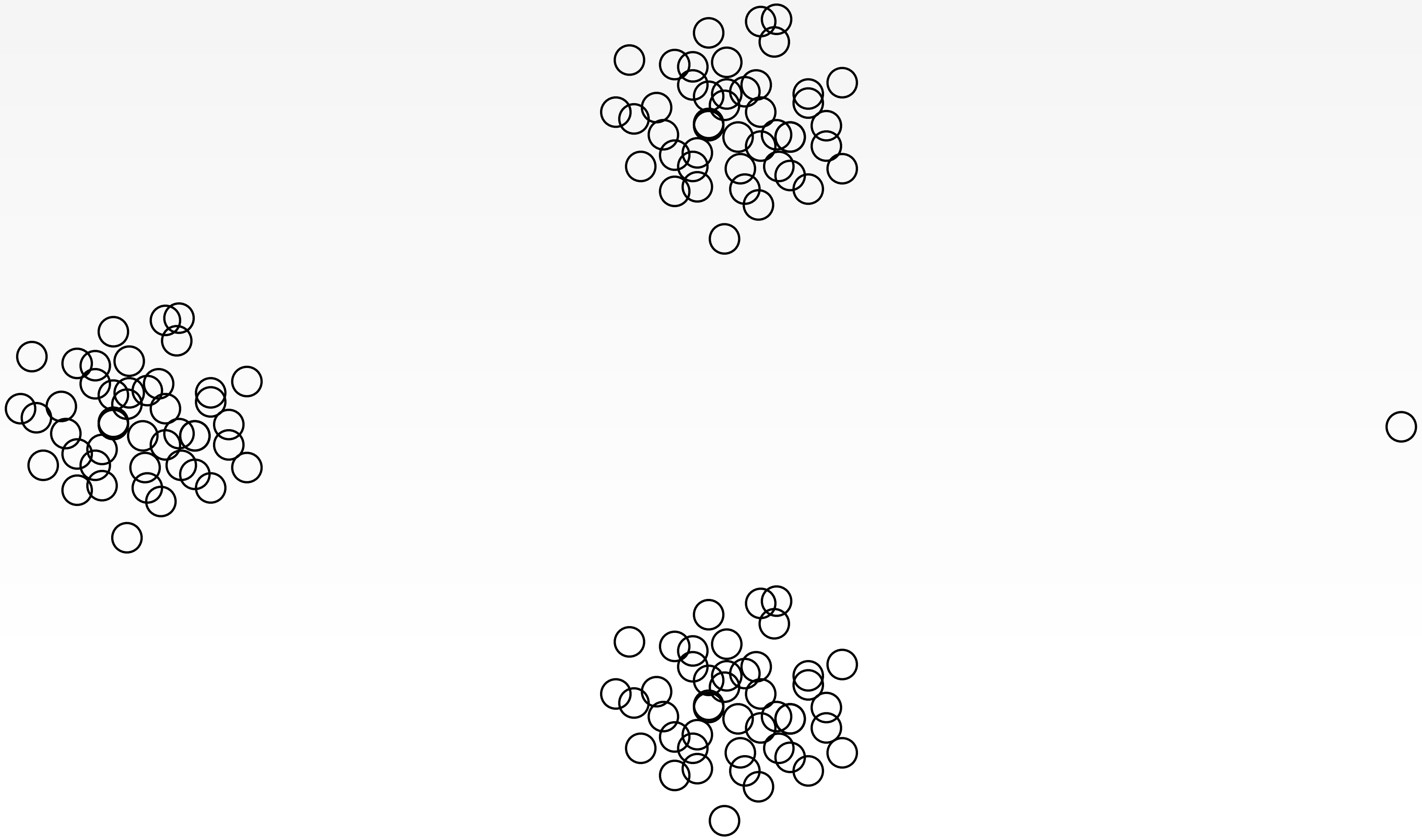


# Easy Fix

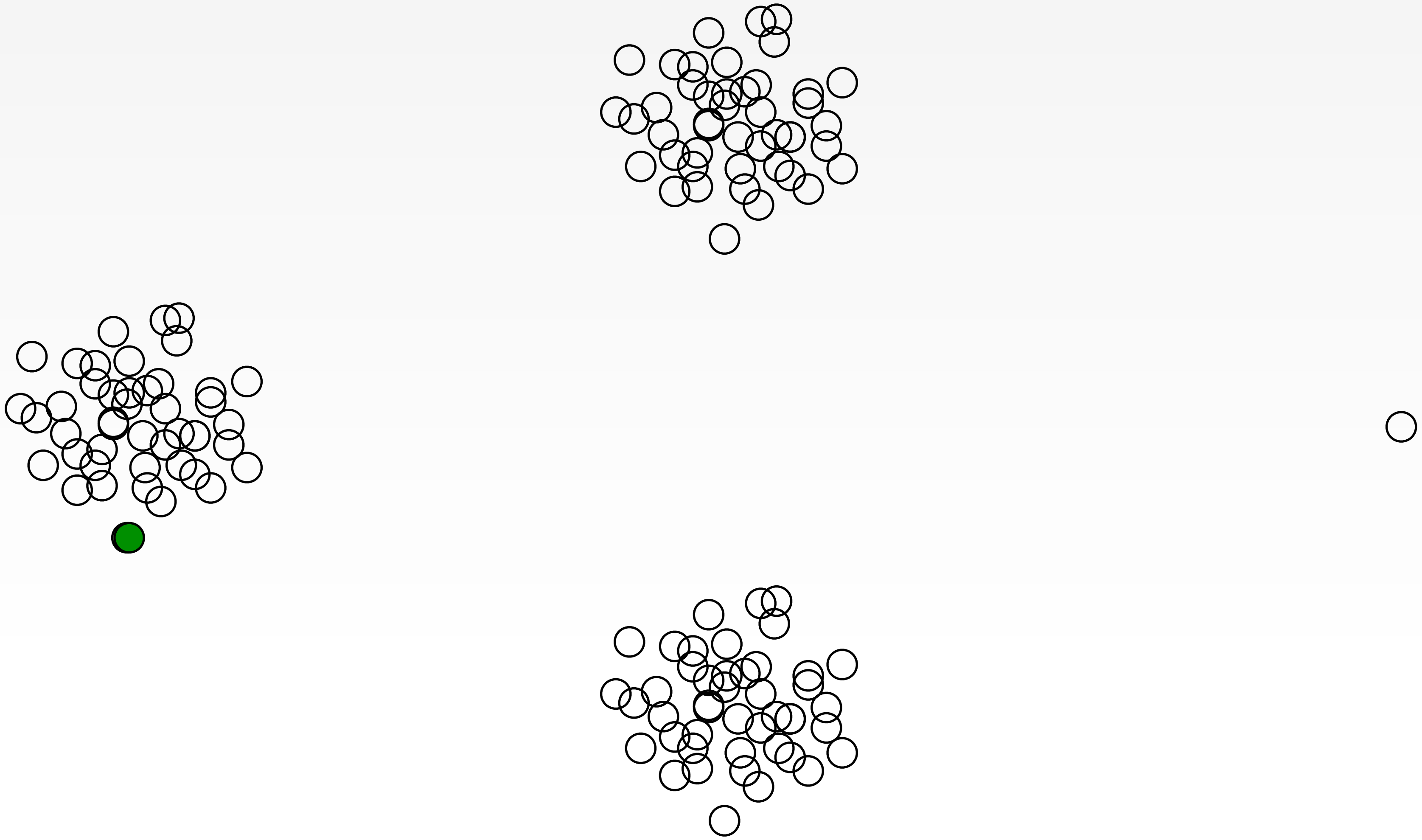
Select centers using a furthest point algorithm (2-approximation to k-Center clustering).



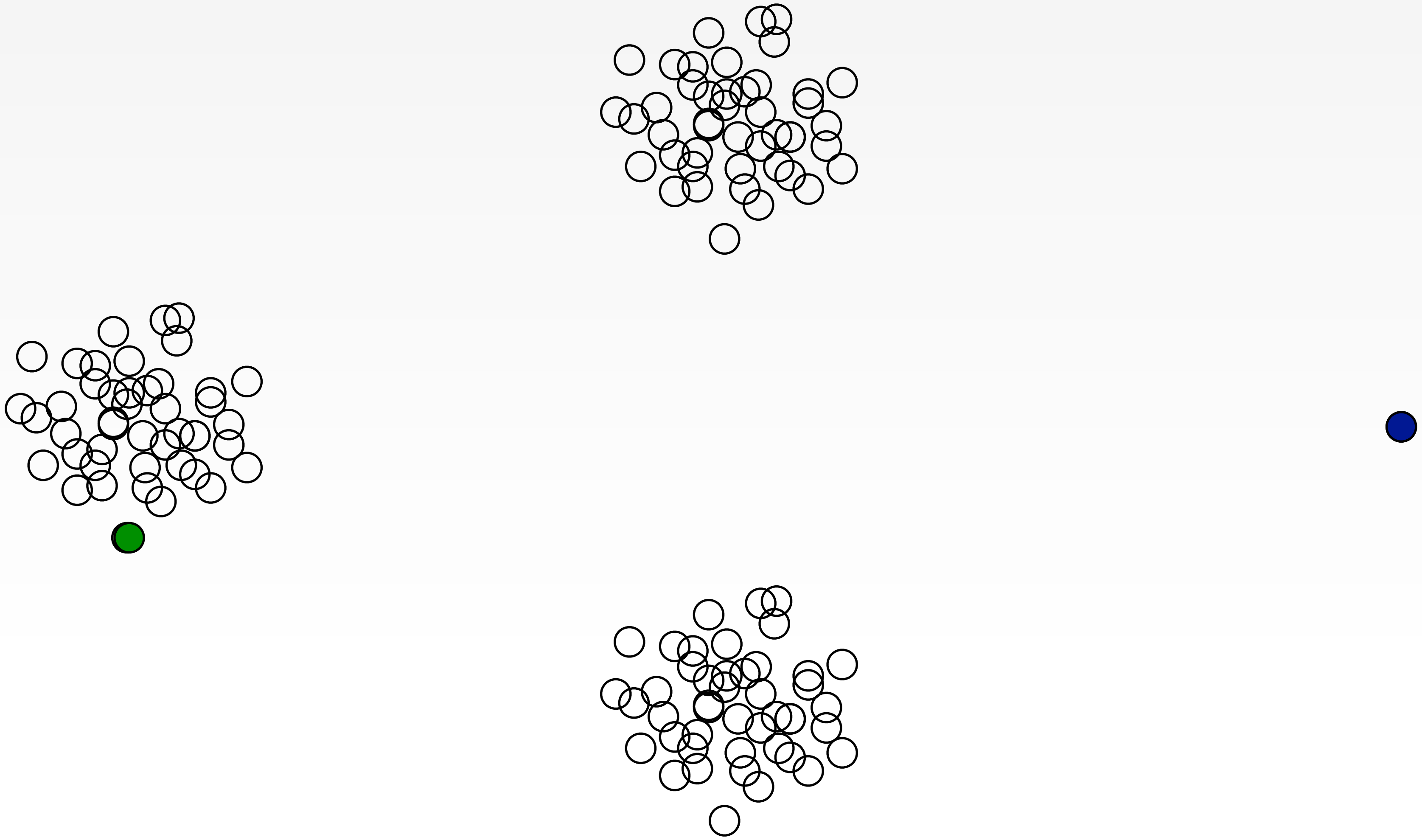
# Sensitive to Outliers



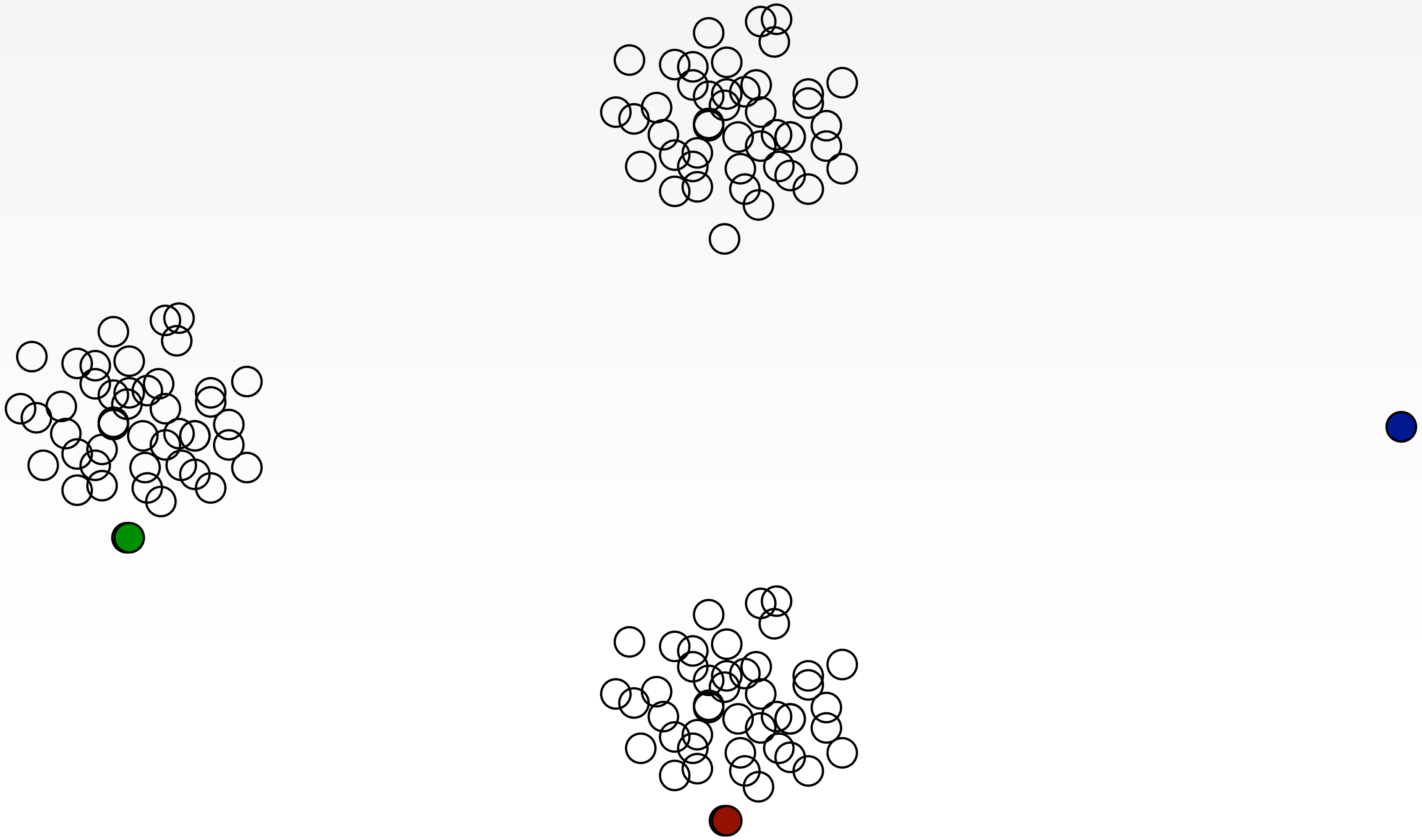
# Sensitive to Outliers



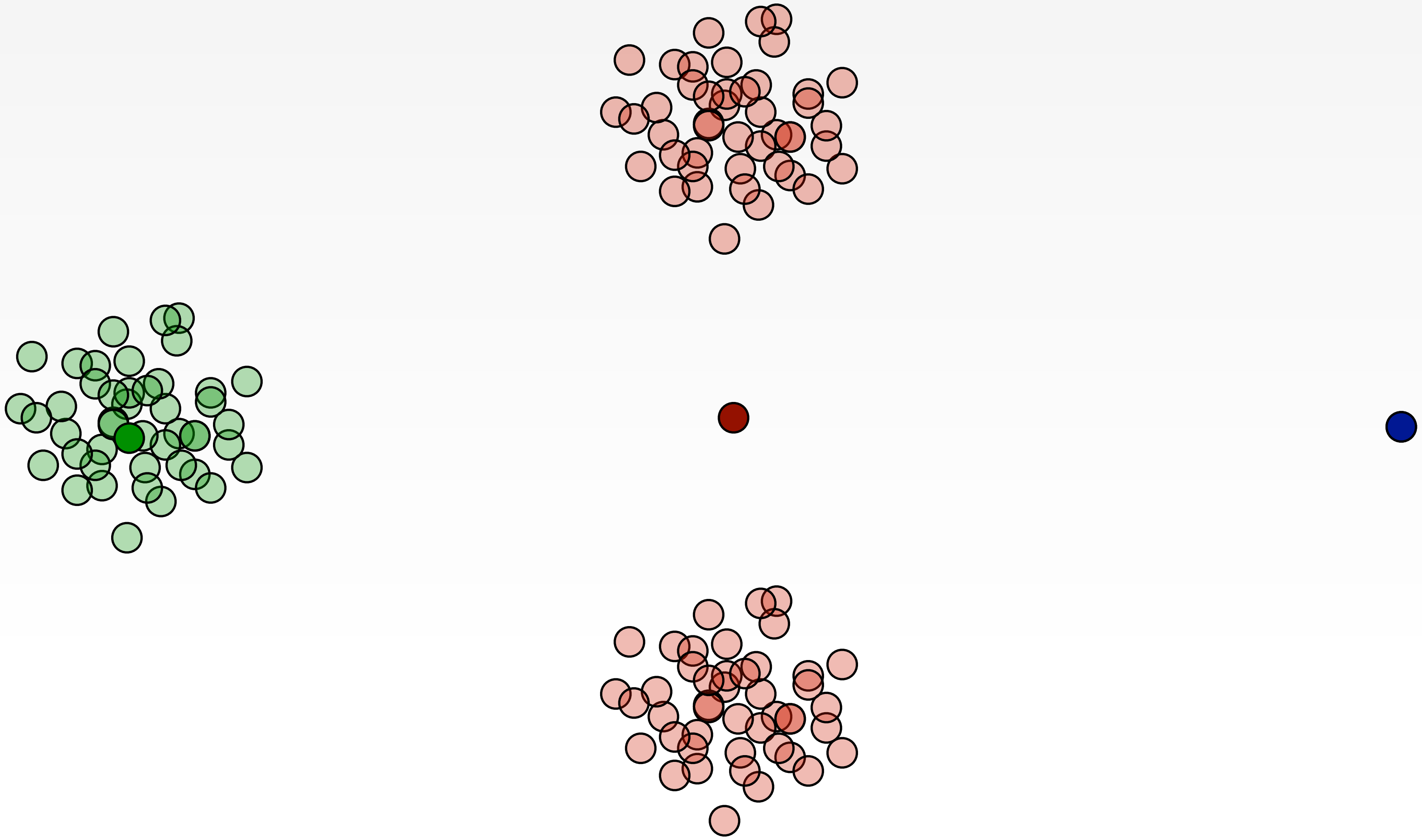
# Sensitive to Outliers



# Sensitive to Outliers



# Sensitive to Outliers



# k-means++

Interpolate between two methods. Give preference to further points.

Let $D(p)$ be the distance between $p$ and the nearest cluster center. Sample next center proportionally to $D^{\alpha}(p)$.

# k-means++

Interpolate between two methods. Give preference to further points.

Let $D(p)$ be the distance between $p$ and the nearest cluster center. Sample next center proportionally to $D^\alpha(p)$.

```
kmeans++:
    Select first point uniformly at random
    for (int i=1; i < k; ++i){
        Select next point p with probability $\frac{D^\alpha(p)}{\sum_x D^\alpha(p)}$ ;
        UpdateDistances();
    }
```

# k-means++

Interpolate between two methods. Give preference to further points.

Let $D(p)$ be the distance between $p$ and the nearest cluster center.
Sample next center proportionally to $D^\alpha(p)$.

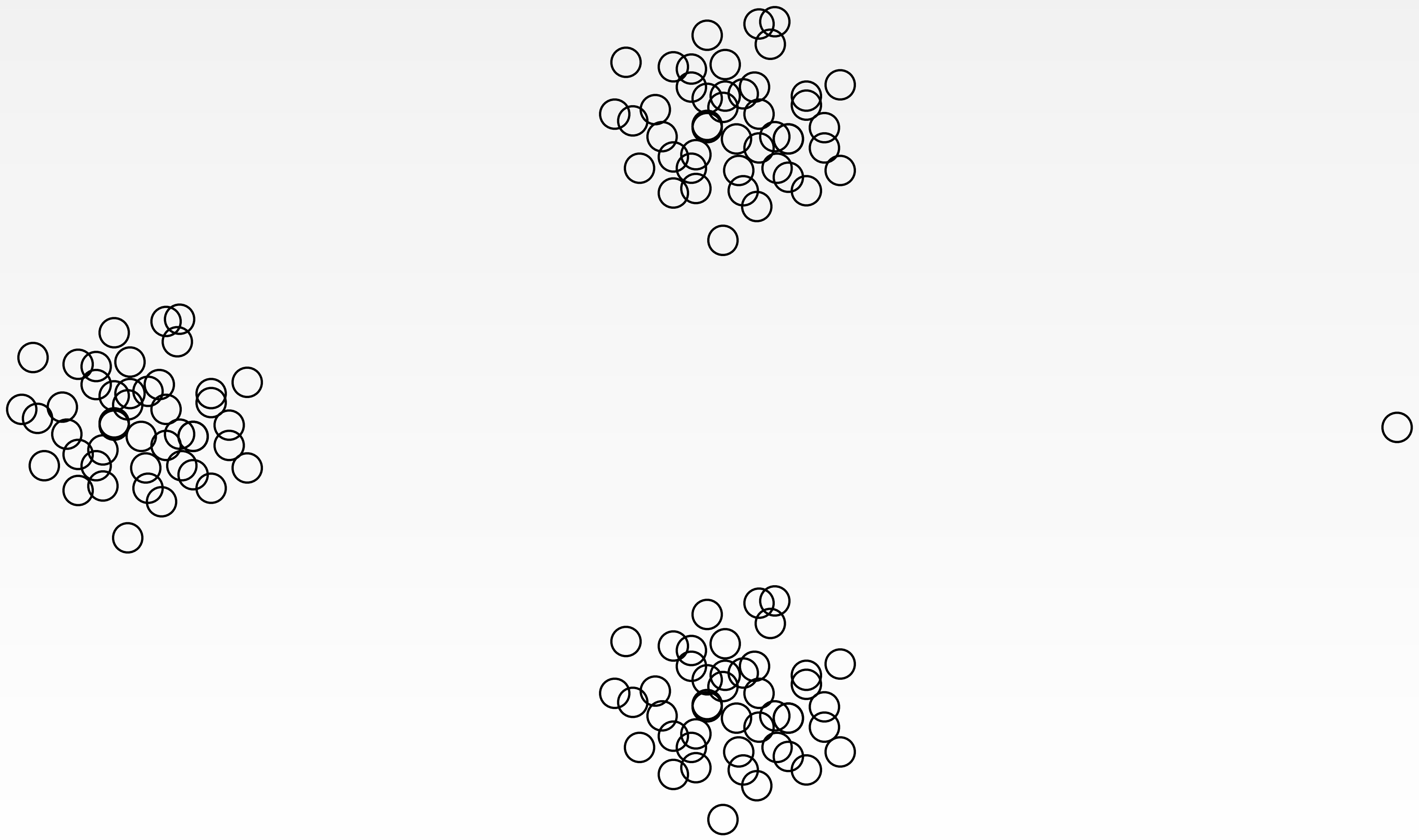
```
kmeans++:
   Select first point uniformly at random
   for (int i=1; i < k; ++i){
      Select next point p with probability $\frac{D^\alpha(p)}{\sum_x D^\alpha(p)}$ ;
      UpdateDistances();
   }
```

Original Lloyd's: $\alpha = 0$

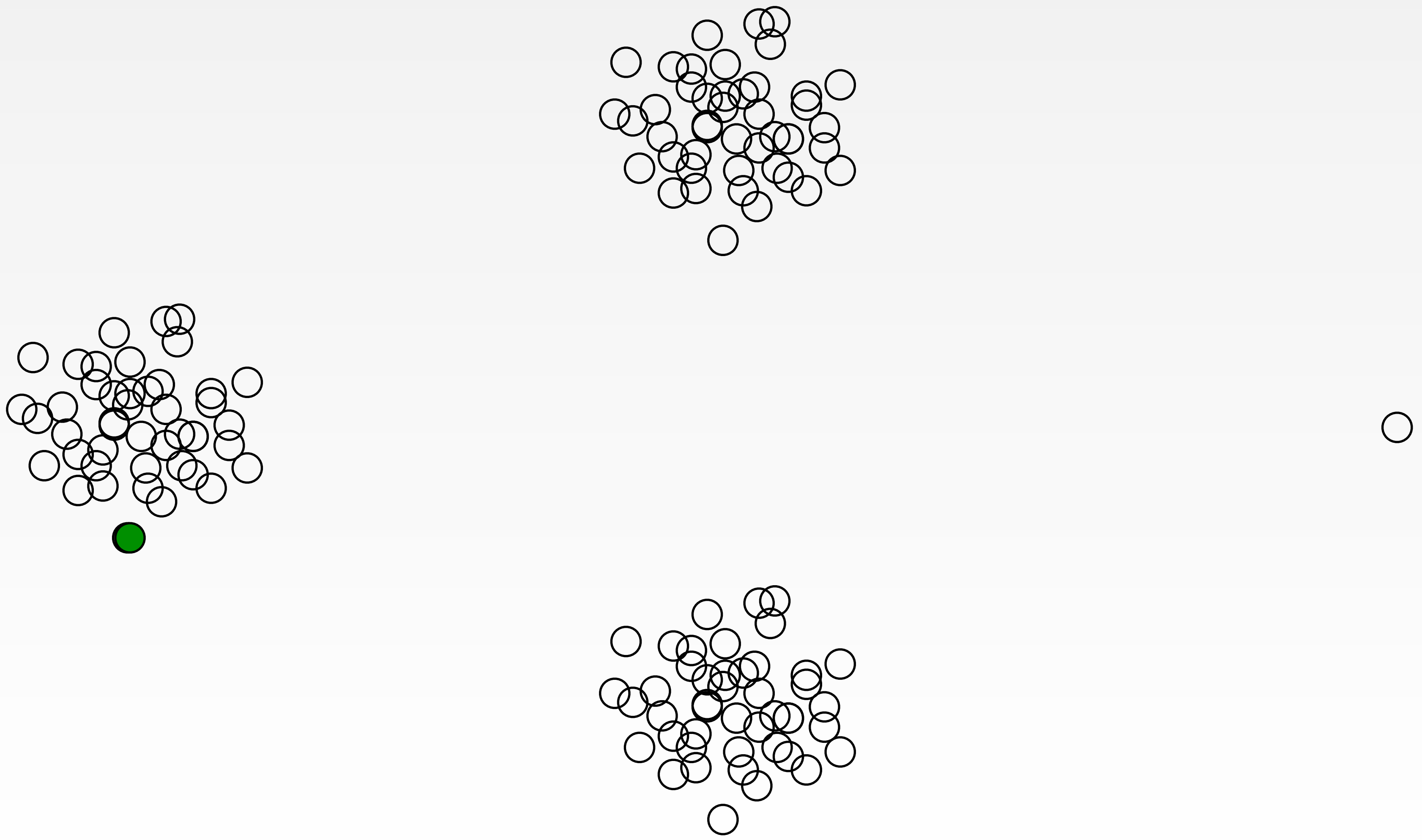
Furthest Point: $\alpha = \infty$

k-means++: $\alpha = 2$

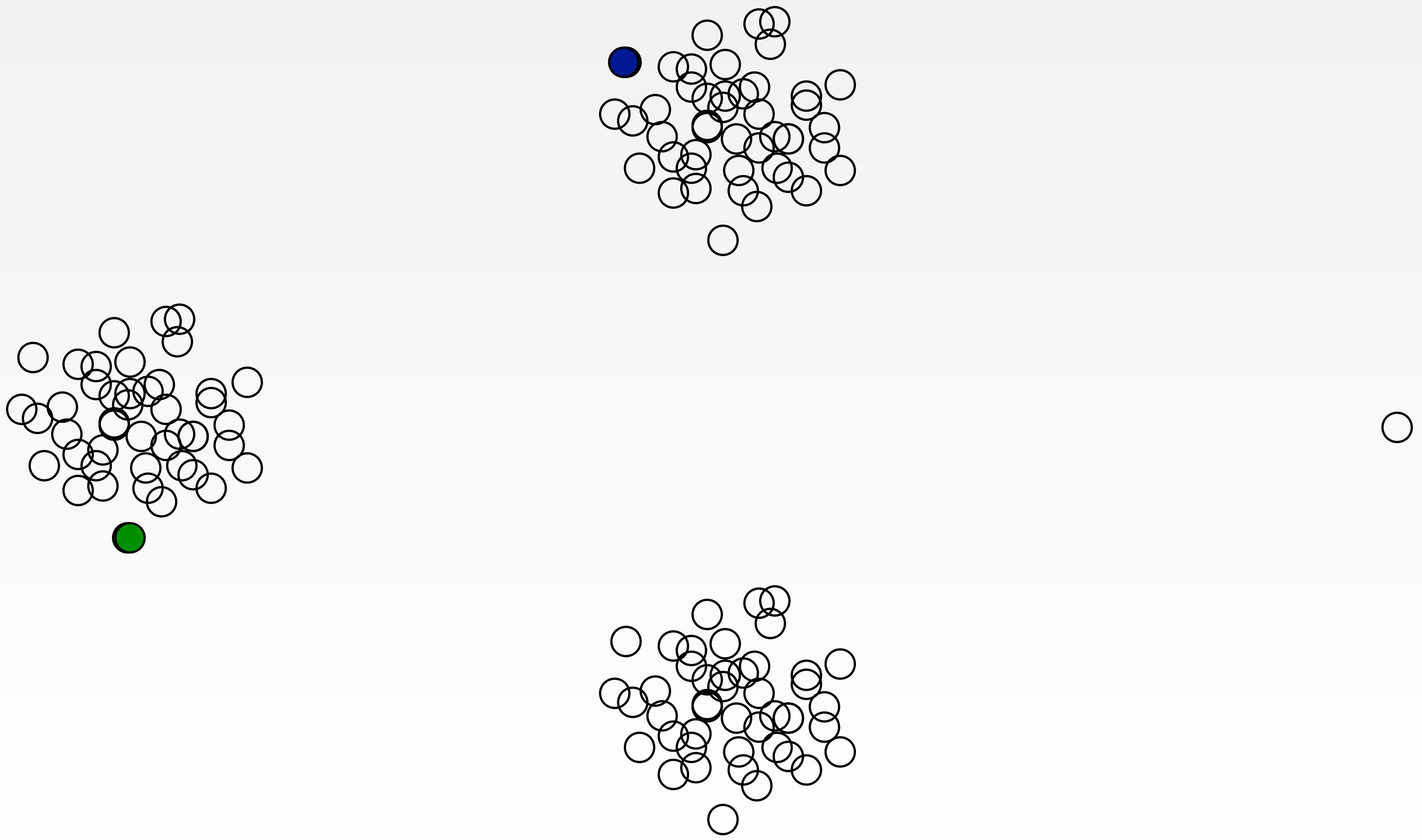
# k-means++



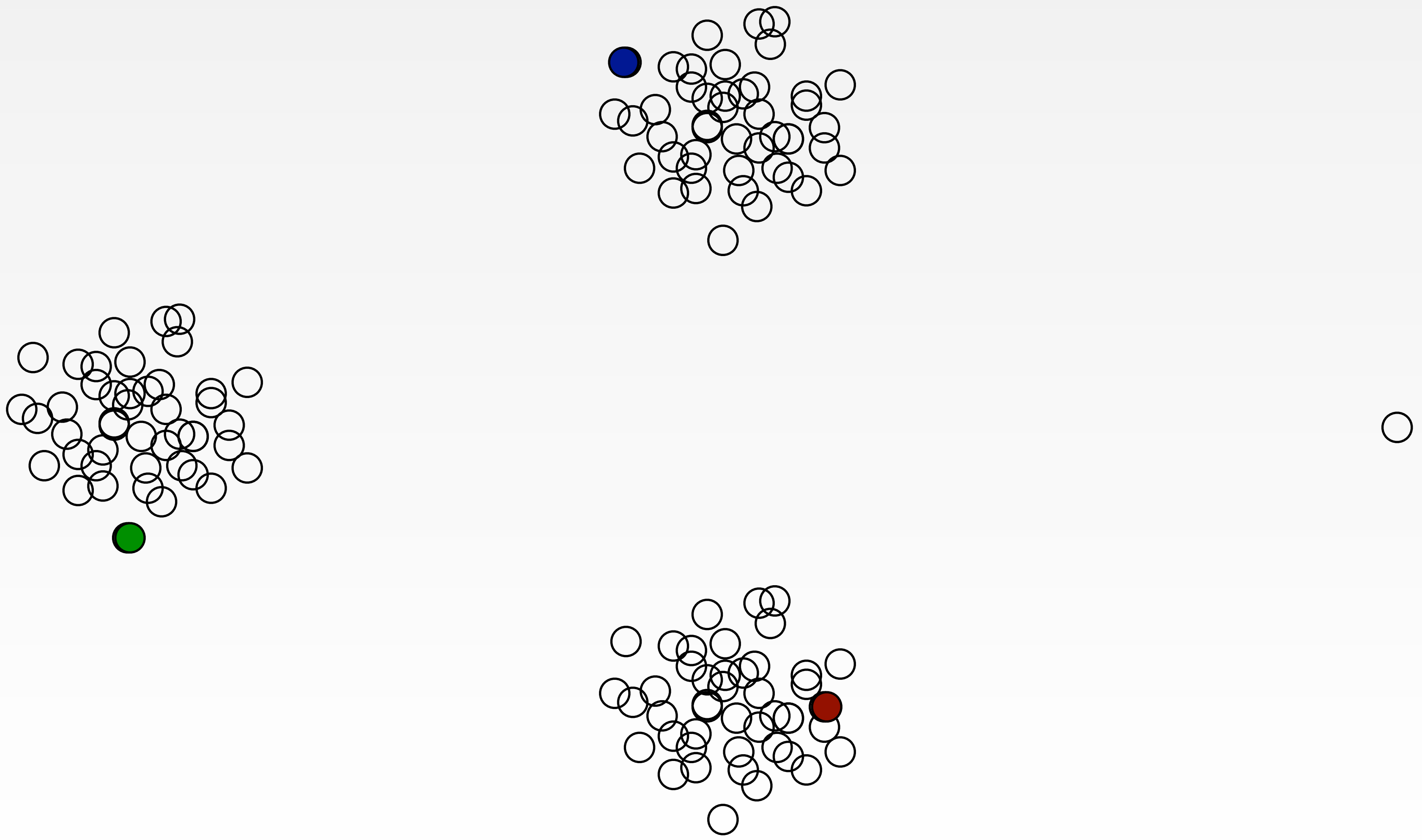
# k-means++



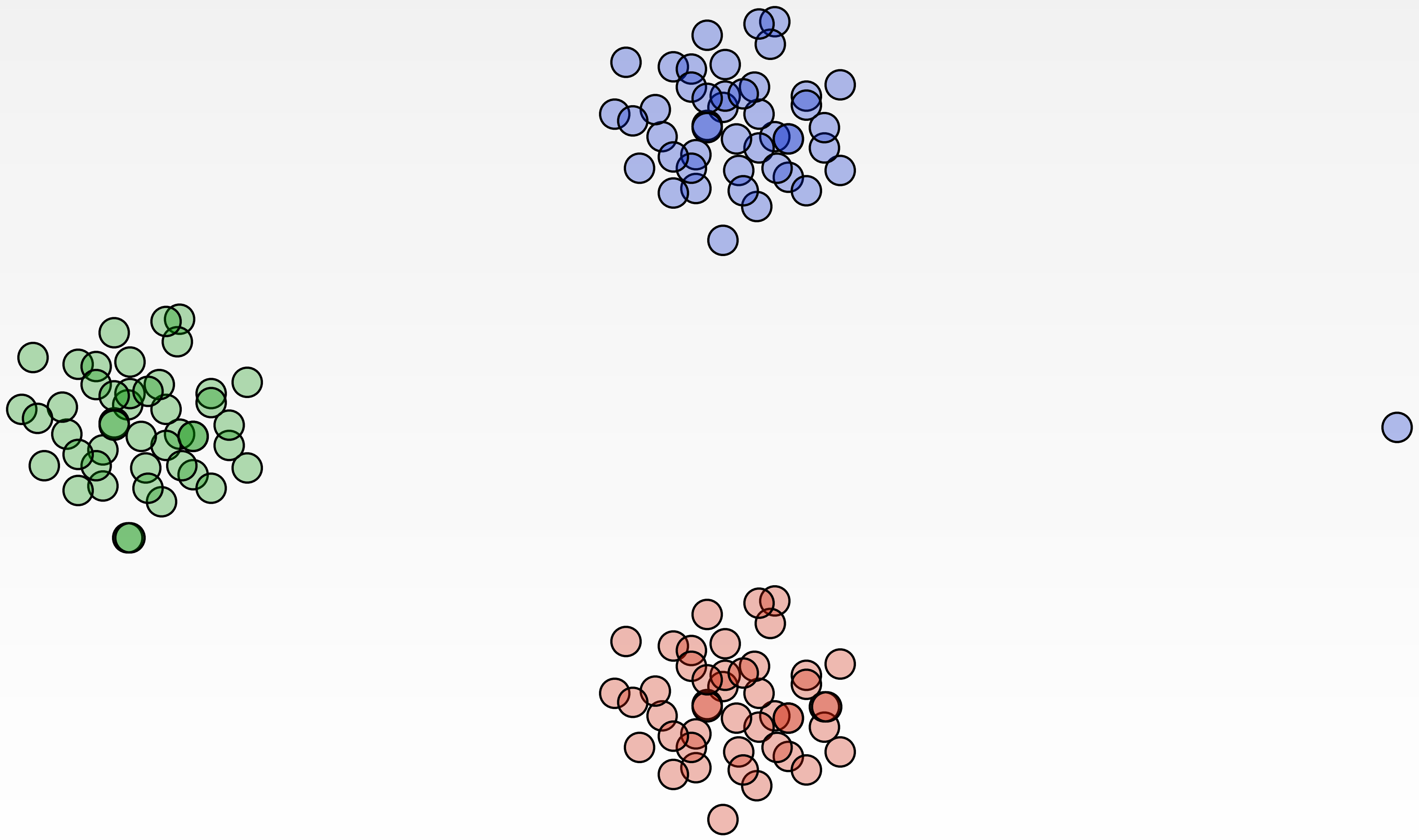
# k-means++



# k-means++



# k-means++



Theorem [AV ’07]: k-means++ guarantees a $\Theta(\log k)$ approximation

# Algorithm

## Initialization:

```
kmeans++:
    Select first point uniformly at random
    for (int i=1; i < k; ++i){
        Select next point p with probability $\frac{D^2(p)}{\sum_p D^2(p)}$ ;
        UpdateDistances();
    }
```

## Very Sequential!

- Must update all distances before selecting next cluster

# Goal: Simulate k-means++

k-means++:

- Is a “soft” greedy algorithm
- Adapt sample & prune technique
- Sample multiple points in each round
- In the end, prune back down to k points

# k-means||

```
kmeans++:
    Select first point uniformly at random
    for (int i=1; i < k; ++i){
        Select next point p with probability $\frac{D^2(p)}{\sum_p D^2(p)}$ ;
        UpdateDistances();
    }
}
```

# k-means||

```
kmeans++:
    Select first point c uniformly at random
    for (int i=1; i < $\log_\ell(\phi(X,c))$; ++i){
        Select point p independently with probability $k \cdot \ell \cdot \frac{D^\alpha(p)}{\sum_x D^\alpha(p)}$
        UpdateDistances();
    }
    Prune to k points total by clustering the clusters
}
```

# k-means||

```
kmeans++:
    Select first point c uniformly at random
    for (int i=1; i < $\log_\ell(\phi(X,c))$; ++i){
        Select point p independently with probability $k \cdot \ell \cdot \frac{D^\alpha(p)}{\sum_x D^\alpha(p)}$
        UpdateDistances();
    }
    Prune to k points total by clustering the clusters
}
```

Independent selection

Easy MR

# k-means||

Oversampling Parameter

```
kmeans++:
    Select first point c uniformly at random
    for (int i=1; i < $\log_\ell(\phi(X,c))$; ++i){
        Select point p independently with probability $k \cdot \ell \cdot \frac{D^\alpha(p)}{\sum_x D^\alpha(p)}$
        UpdateDistances();
    }
    Prune to k points total by clustering the clusters
}
```

Independent selection

Easy MR

# k-means||

Oversampling Parameter

```
kmeans++:
    Select first point c uniformly at random
    for (int i=1; i < $\log_\ell(\phi(X,c))$; ++i){
        Select point p independently with probability $k \cdot \ell \cdot \frac{D^\alpha(p)}{\sum_x D^\alpha(p)}$
        UpdateDistances();
    }
    Prune to k points total by clustering the clusters
}
```

Independent selection

Easy MR

Re-clustering step

# k-means||: Analysis

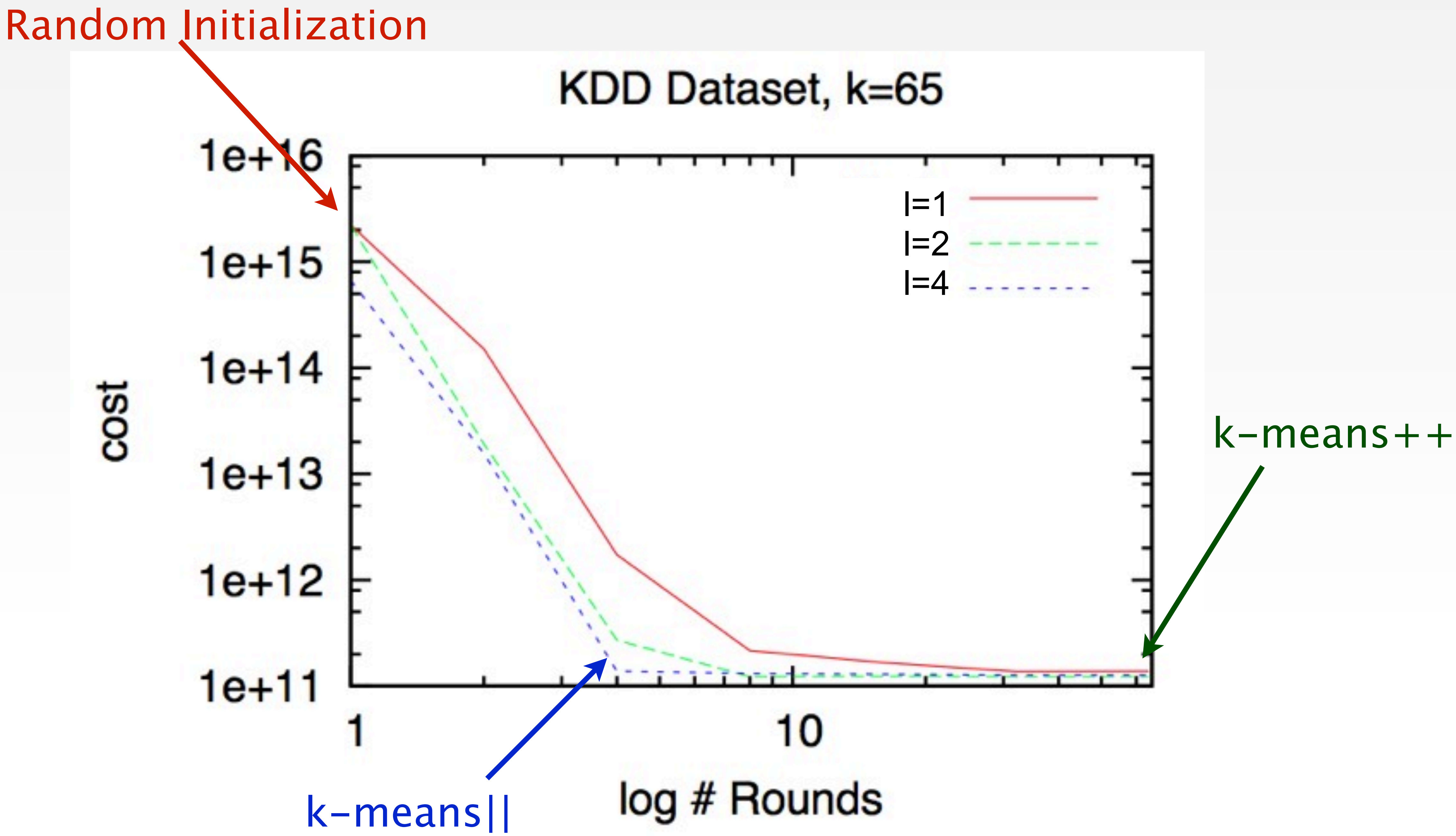
## How Many Rounds?

- Theorem: After $O(\log_\ell(n\Delta))$ rounds, guarantee $O(1)$ approximation
- In practice: fewer iterations are needed
- Need to re-cluster $O(k\ell \log_\ell(n\Delta))$ intermediate centers

## Discussion:

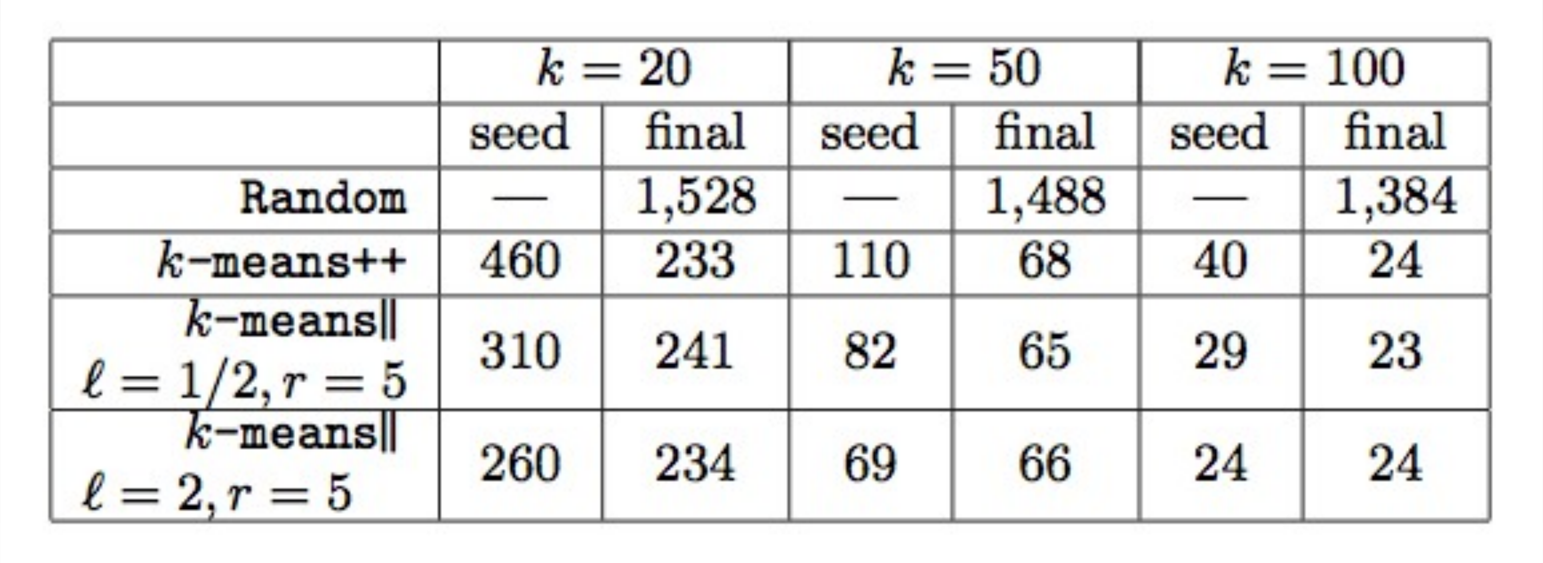
- Number of rounds independent of k
- Tradeoff between number of rounds and memory

# How well does this work?



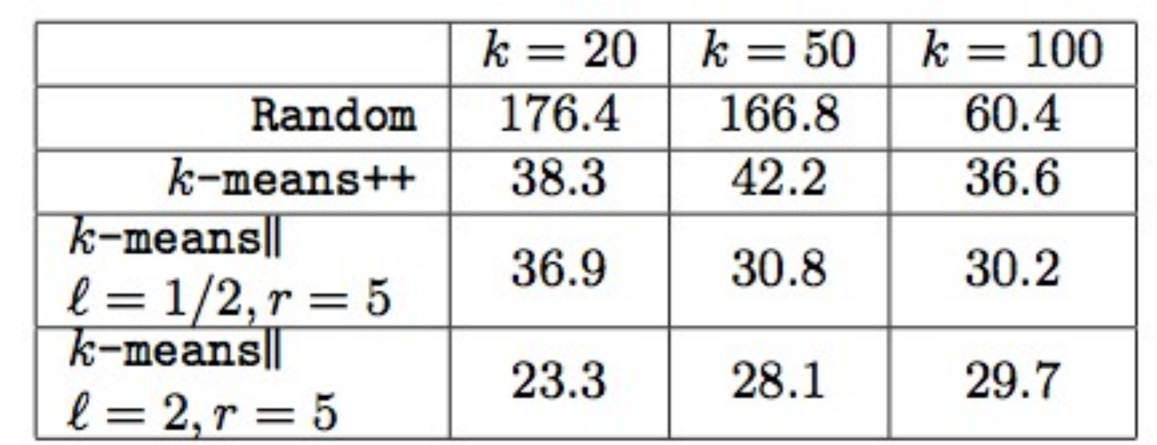
# Performance vs. k-means++

- Even better on small datasets: 4600 points, 50 dimensions (SPAM)
- Accuracy:



| | $k = 20$ | | $k = 50$ | | $k = 100$ | |
|---|---|---|---|---|---|---|
| | seed | final | seed | final | seed | final |
| Random | — | 1,528 | — | 1,488 | — | 1,384 |
| $k$-means++ | 460 | 233 | 110 | 68 | 40 | 24 |
| $k$-means‖<br>$\ell = 1/2, r = 5$ | 310 | 241 | 82 | 65 | 29 | 23 |
| $k$-means‖<br>$\ell = 2, r = 5$ | 260 | 234 | 69 | 66 | 24 | 24 |

- Time (iterations):



| | $k = 20$ | $k = 50$ | $k = 100$ |
|---|---|---|---|
| Random | 176.4 | 166.8 | 60.4 |
| $k$-means++ | 38.3 | 42.2 | 36.6 |
| $k$-means‖<br>$\ell = 1/2, r = 5$ | 36.9 | 30.8 | 30.2 |
| $k$-means‖<br>$\ell = 2, r = 5$ | 23.3 | 28.1 | 29.7 |

# Conclusion: ML

Three prevalent ideas:

- If the dimension is small: prune (in a smart way)
- If the dimension is large, figure out what to optimize All-Reduce
- For clustering, adapt methods by oversampling & pruning

# Conclusion: MapReduce

Overall:

- A robust implementation of the BSP model
- Easy to work with, easy to think about

Things to Keep in Mind:

- The data will be skewed!
- For specific classes of problems, additional optimizations possible
  - Graphs with Pregel, ML with All-Reduce
- Wisely sampling the input & using the sample gets you very far

# Conclusion: MapReduce

Overall:

- A robust implementation of the BSP model
- Easy to work with, easy to think about

Things to Keep in Mind:

- The data will be skewed!
- For specific classes of problems, additional optimizations possible
  - Graphs with Pregel, ML with All-Reduce
- Wisely sampling the input & using the sample gets you very far

- Apparently it never rains in Aarhus :-)

# References

- Map-Reduce for Machine Learning on Multicore. Cheng-Tao Chu, Sang Kyun Kim, Yi-An Lin, YuanYuan Yu, Gary Bradski, Andrew Ng, Kunle Olukotun, NIPS 2006.
- Space-Round Tradeoffs for MapReduce Computations. Andrea Pietracaprina, Geppino Pucci, Matteo Riondato, Francesco Silvestri, Eli Upfal, ICS 2012.
- Efficient Noise-Tolerant Learning from Statistical Queries. Michael Kearns, STOC 1993.
- A Reliable Effective Terascale Linear Learning System. Alekh Agarwal, Olivier Chapelle, Miroslav Dudik, John Langford, ArXiv 2011.
- k-means++: The Advantages of Careful Seeding. David Arthur, S.V., SODA 2007.
- Scalable k-means++. Bahman Bahmani, Benjamin Moseley, Andrea Vattani, Ravi Kumar, S.V., VLDB 2012.