Autoencoders and dimensionality reduction

10-301/601

Geoff Gordon

w/ thanks to Matt Gormley and Henry Chai

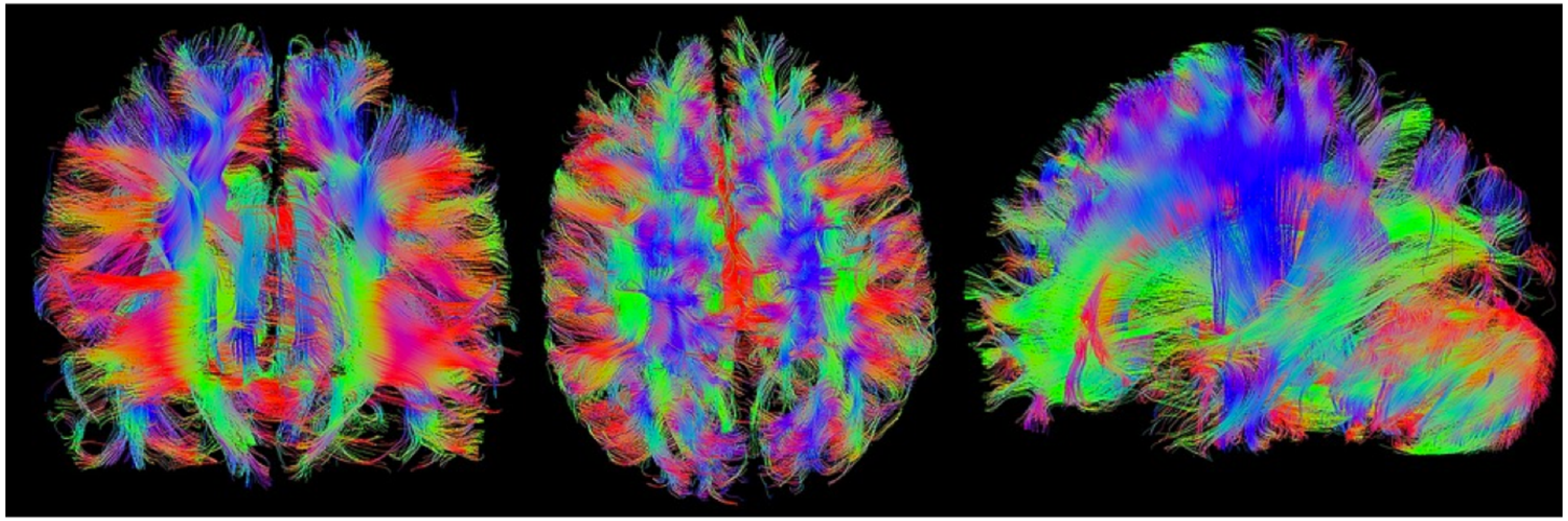
Unsupervised learning

Example use of unsupervised learning: high-dimensional (megapixels) photos

- Recall: ***unsupervised learning***
- Given dataset $\mathscr{D} = \{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(N)}\}$
  - no specific labels $y_1 \ldots y_N$
- Goal: better understand $\mathscr{D}$
  - e.g., exploratory analysis: figure out what info we have
  - e.g., so we can solve some downstream learning problem better

Geoff Gordon

image credit: https://pixabay.com/en/brain-mrt-magnetic-resonance-imaging-1728449/

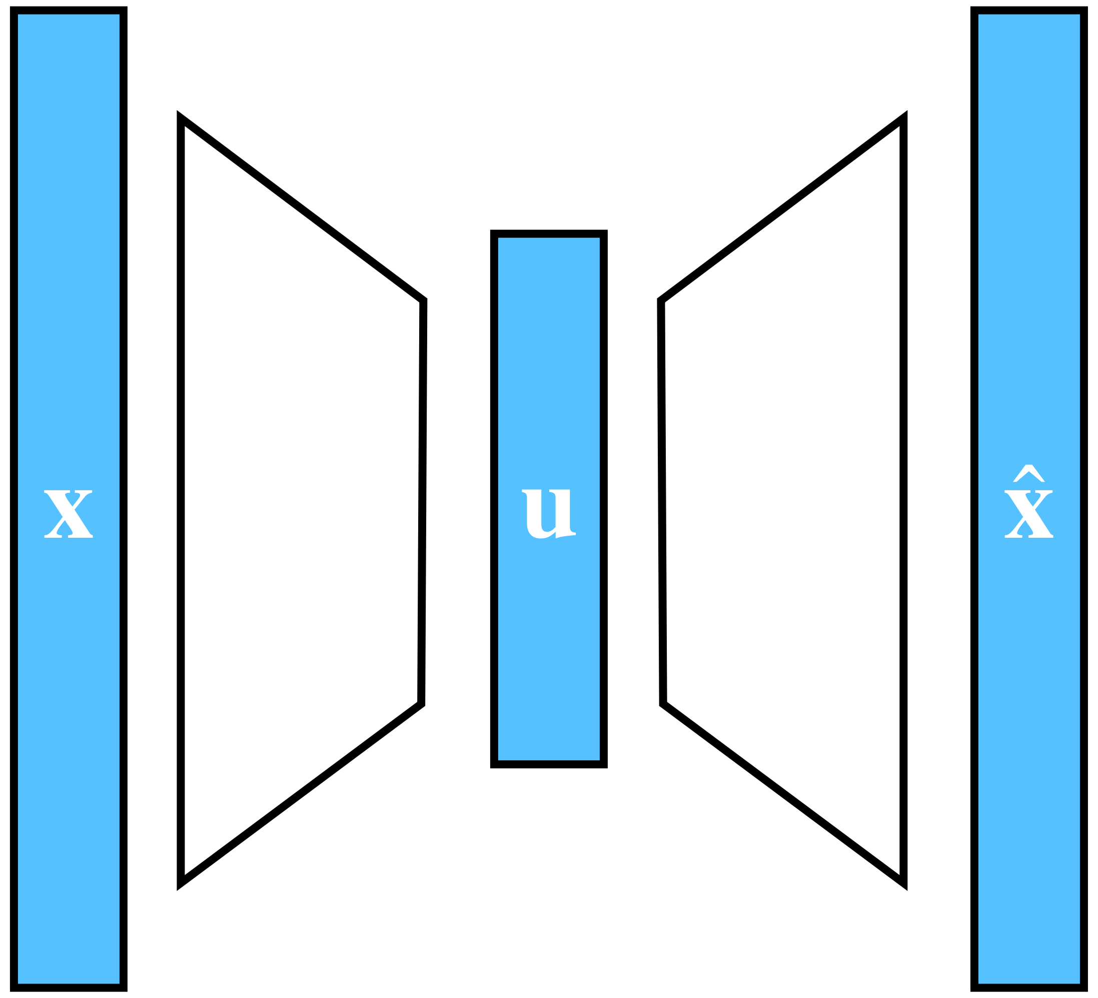


Example use of unsupervised learning: brain scans

Unsupervised learning

- Recall: ***unsupervised learning***
- Given dataset $\mathscr{D} = \{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(N)}\}$
  - no specific labels $y_1 \ldots y_N$
- Goal: better understand $\mathscr{D}$
  - e.g., exploratory analysis: figure out what info we have
  - e.g., so we can solve some downstream learning problem better

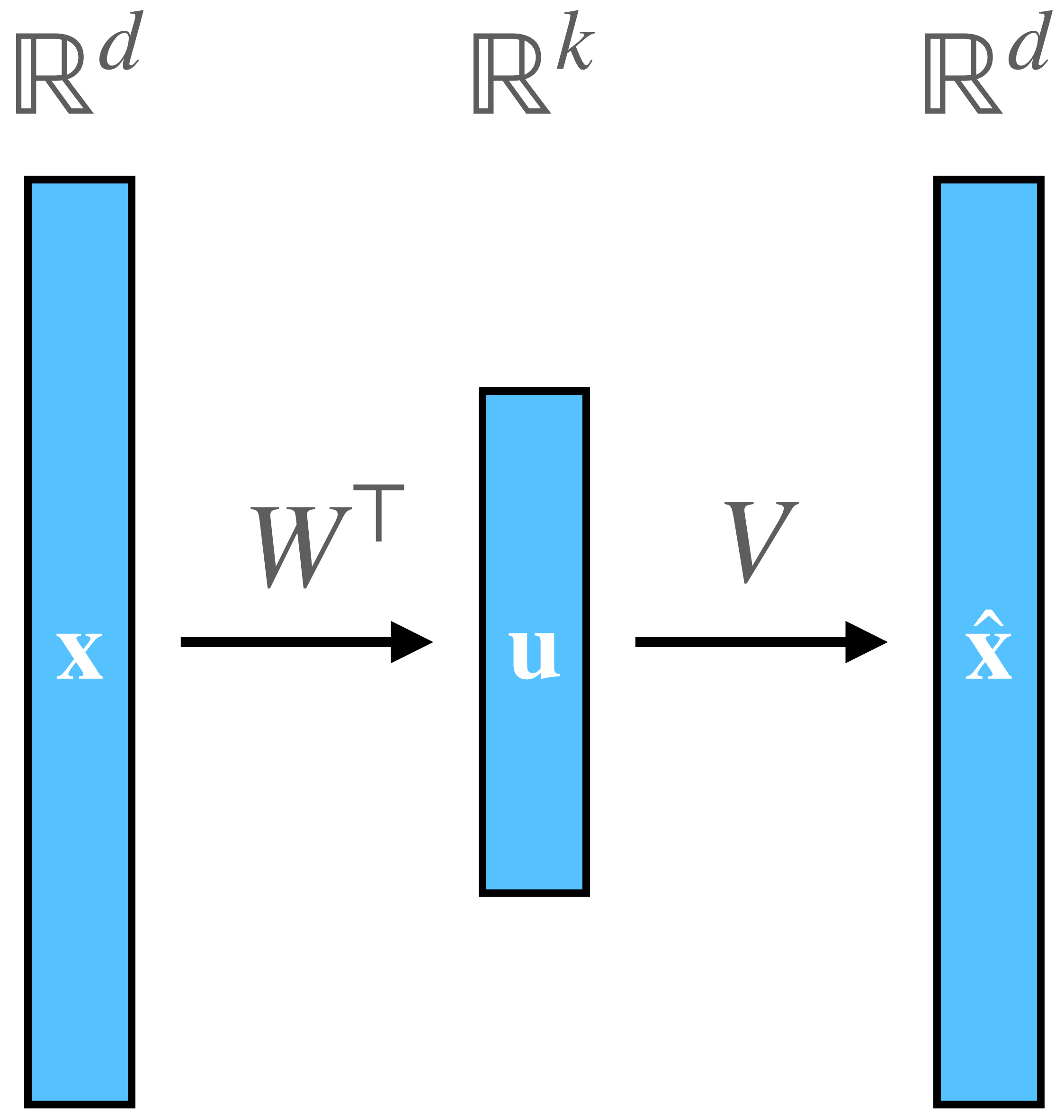
Autoencoder



Why the name? “Code” = hidden activations, so $\mathbf{x}$ encodes (and then decodes) itself

- Large, useful class of unsupervised model: *autoencoder*
- Train a model to predict $\mathbf{x}^{(i)}$ from $\mathbf{x}^{(i)}$ — sounds circular!
- The catch: something about the model (the ***bottleneck***) prevents us from just copying input to output
  - e.g., continuous hidden layer $\mathbf{u}^{(i)}$ w/ too few dimensions
  - e.g., discrete hidden layer $\mathbf{z}^{(i)}$ w/ too few bits
  - e.g., regularizer that disfavors straight copying

Linear autoencoder



$$\hat{\mathbf{x}} = V\mathbf{u} = VW^\top\mathbf{x}$$

$$V, W \in \mathbb{R}^{d \times k}$$

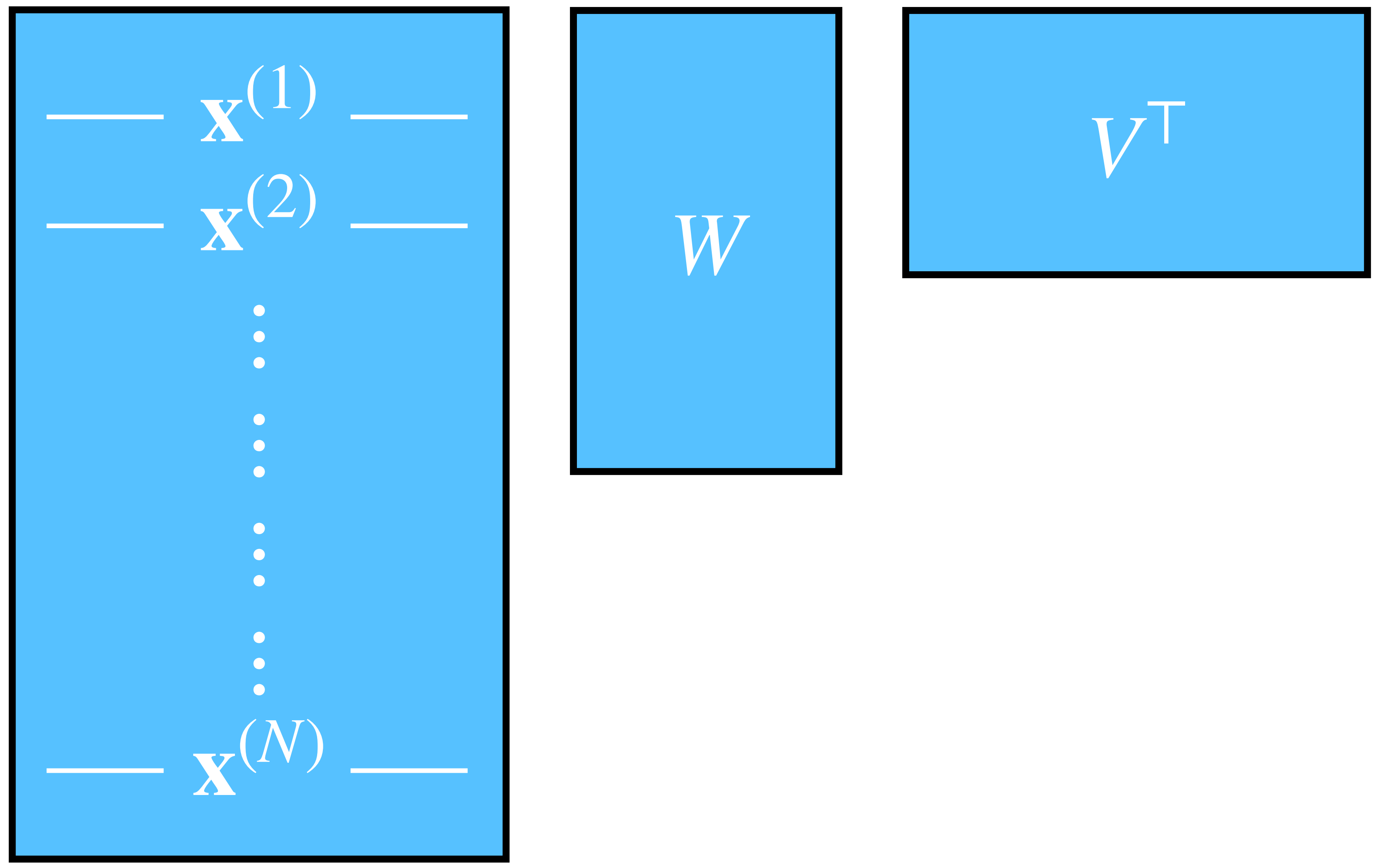
$$k \ll d$$

- Simplest autoencoder: one hidden layer, no nonlinearities
  - min sum squared error: $\min_{V,W} \sum_{i=1}^{N} ||VW^\top\mathbf{x}^{(i)} - \mathbf{x}^{(i)}||^2$
  - solve w/ SGD or alternating least squares

note: no bias weights for now (we'll deal with them later)

Geoff Gordon

Matrix form



$$\hat{\mathbf{x}} = V\mathbf{u} = VW^\top \mathbf{x}$$

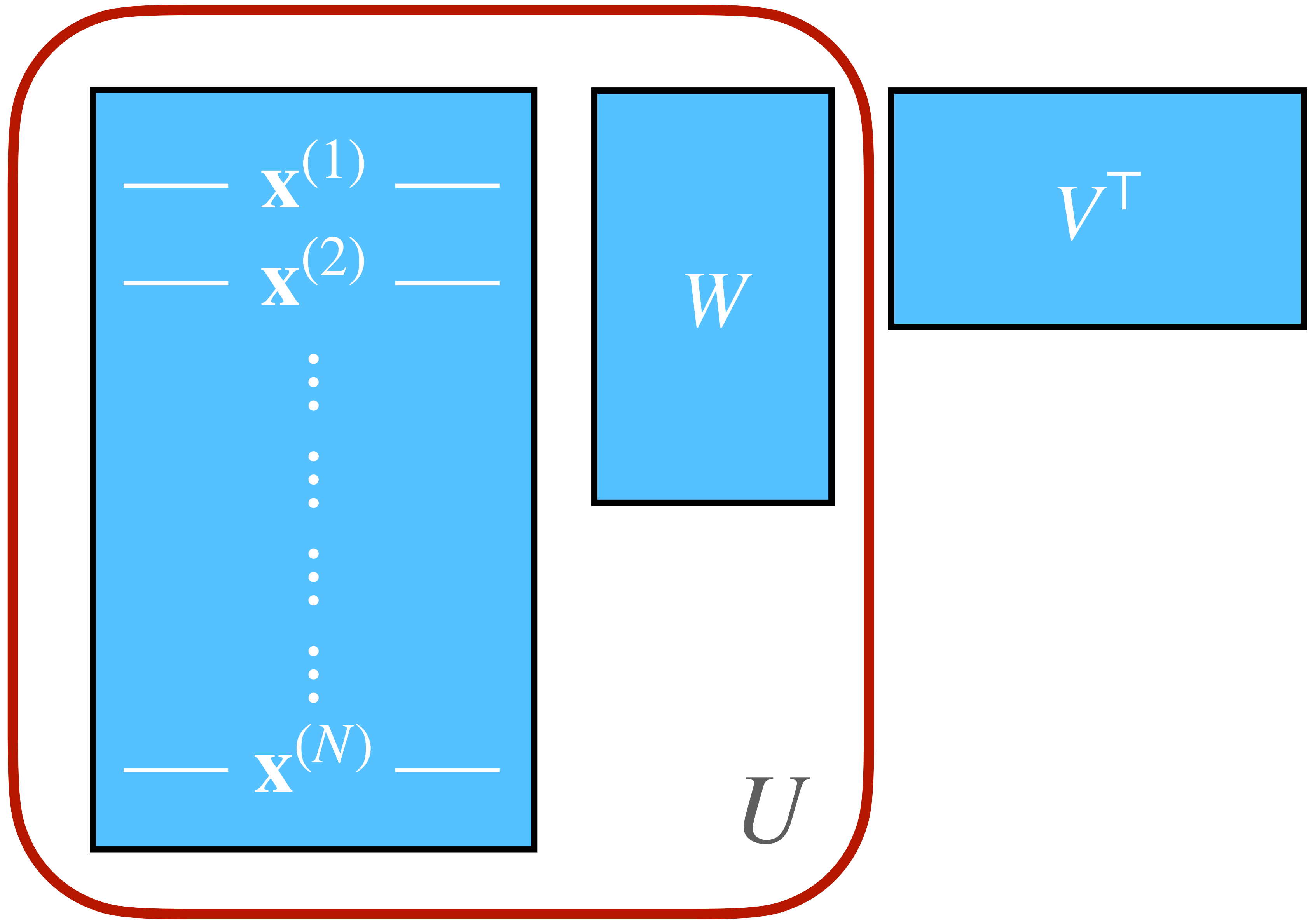
$$\hat{X} = UV^\top = XWV^\top$$

$$X \in \mathbb{R}^{N\times d}$$

$$U \in \mathbb{R}^{N\times k}$$

- Collect all examples $\mathbf{x}^{(i)}$ into a matrix X, one per row
- Collect latent vectors $\mathbf{u}^{(i)} = W^\top \mathbf{x}^{(i)}$ into matrix U
- Write $\mathbf{v}_j$ for jth row of V (= column of $V^\top$)

Matrix form



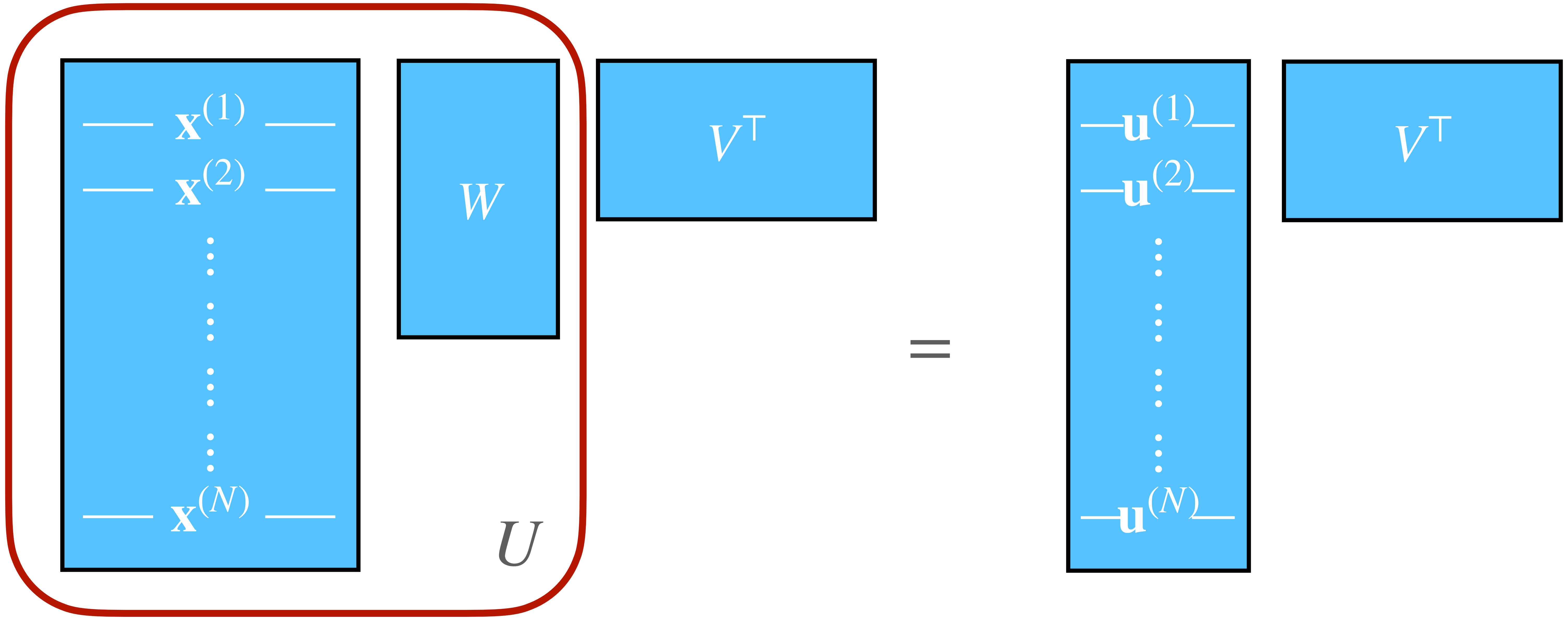
$$\hat{\mathbf{x}} = V\mathbf{u} = VW^\top\mathbf{x}$$
$$\hat{X} = UV^\top = XWV^\top$$

$$X \in \mathbb{R}^{N\times d}$$
$$U \in \mathbb{R}^{N\times k}$$

- Collect all examples $\mathbf{x}^{(i)}$ into a matrix X, one per row
- Collect latent vectors $\mathbf{u}^{(i)} = W^\top\mathbf{x}^{(i)}$ into matrix U
- Write $\mathbf{v}_j$ for jth row of V (= column of $V^\top$)

Geoff Gordon

Matrix form



$$\hat{\mathbf{x}} = V\mathbf{u} = VW^\top \mathbf{x}$$

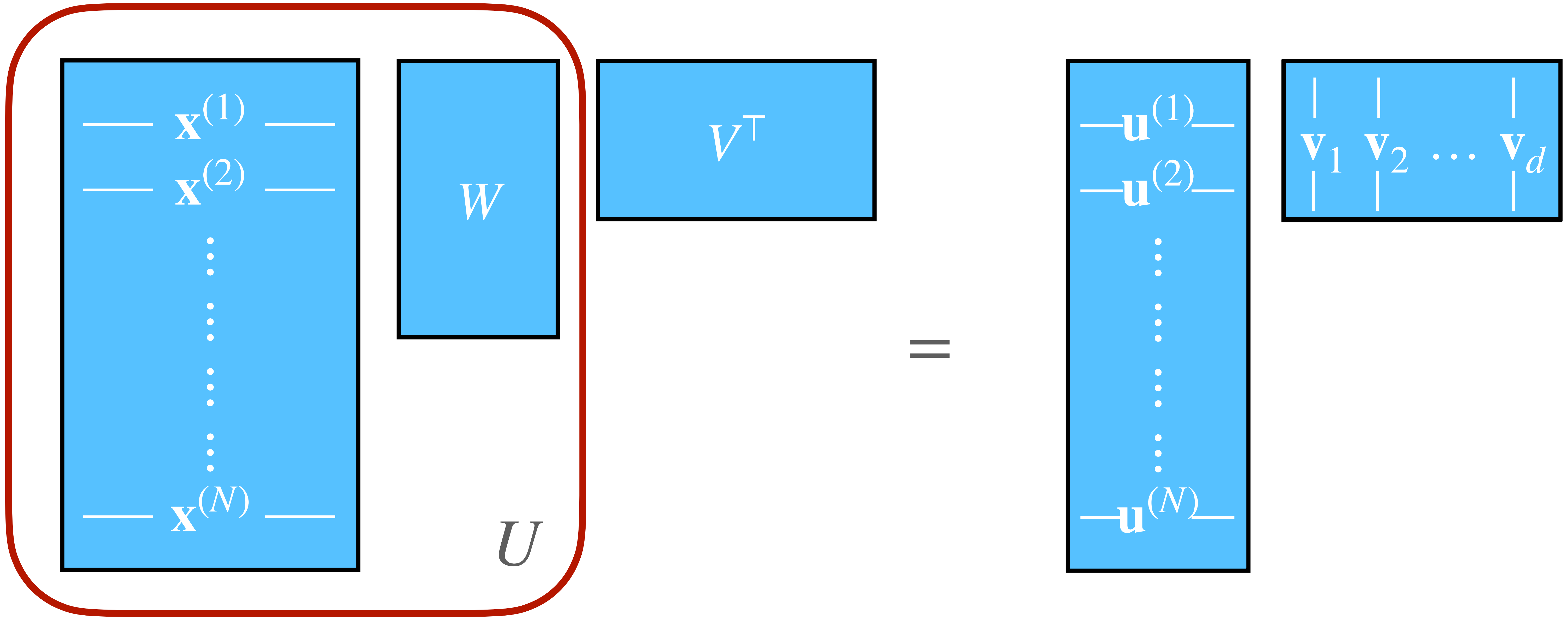
$$\hat{X} = UV^\top = XWV^\top$$

$$X \in \mathbb{R}^{N\times d}$$

$$U \in \mathbb{R}^{N\times k}$$

- Collect all examples $\mathbf{x}^{(i)}$ into a matrix X, one per row
- Collect latent vectors $\mathbf{u}^{(i)} = W^\top \mathbf{x}^{(i)}$ into matrix U
- Write $\mathbf{v}_j$ for jth row of V (= column of $V^\top$)

Matrix form



$$\hat{\mathbf{x}} = V\mathbf{u} = VW^\top\mathbf{x}$$

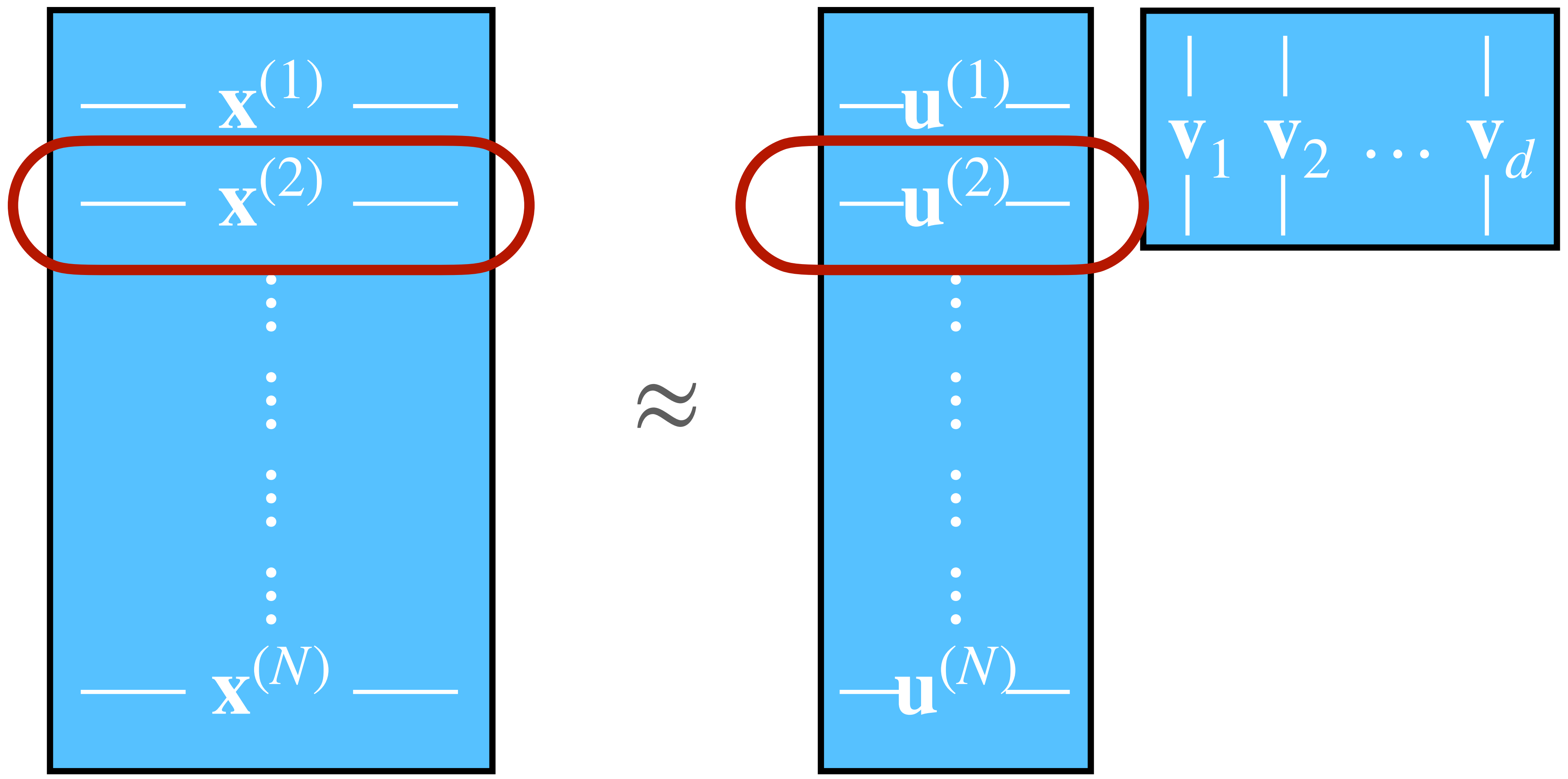
$$\hat{X} = UV^\top = XWV^\top$$

$$X \in \mathbb{R}^{N\times d}$$

$$U \in \mathbb{R}^{N\times k}$$

- Collect all examples $\mathbf{x}^{(i)}$ into a matrix X, one per row
- Collect latent vectors $\mathbf{u}^{(i)} = W^\top\mathbf{x}^{(i)}$ into matrix U
- Write $\mathbf{v}_j$ for jth row of V (= column of $V^\top$)

Best hidden activation vector $\mathbf{u}^{(i)}$



- Suppose we could choose $\mathbf{u}^{(i)}$ arbitrarily, instead of $\mathbf{u}^{(i)} = W^\top \mathbf{x}^{(i)}$ — solve for best $\mathbf{u}^{(i)}$, holding V fixed
- Regression objective: minimize $\sum_{j=1}^{d} (\mathbf{x}_j^{(i)} - \mathbf{v}_j^\top \mathbf{u}^{(i)})^2$
  - one "training example" for each dimension of $\mathbf{x}^{(i)}$
  - $\mathbf{v}_j$: feature vector for example j
  - $\mathbf{x}_j^{(i)}$: target output for example j
  - $\mathbf{u}^{(i)}$: learnable regression weights

Geoff Gordon

Best hidden activation vector $\mathbf{u}^{(i)}$

- Differentiate $\sum_{j=1}^{d} (\mathbf{x}_j^{(i)} - \mathbf{v}_j^\top \mathbf{u}^{(i)})^2$ wrt $\mathbf{u}^{(i)}$ and set to 0:

$$-2 \sum_{j=1}^{d} (x_j^{(i)} - v_j^T u^{(i)}) v_j = 0$$

$$\sum_{j=1}^{d} x_j^{(i)} v_j = \sum_{j=1}^{d} v_j v_j^T u^{(i)}$$

$$V^T x^{(i)} = V^T V u^{(i)}$$

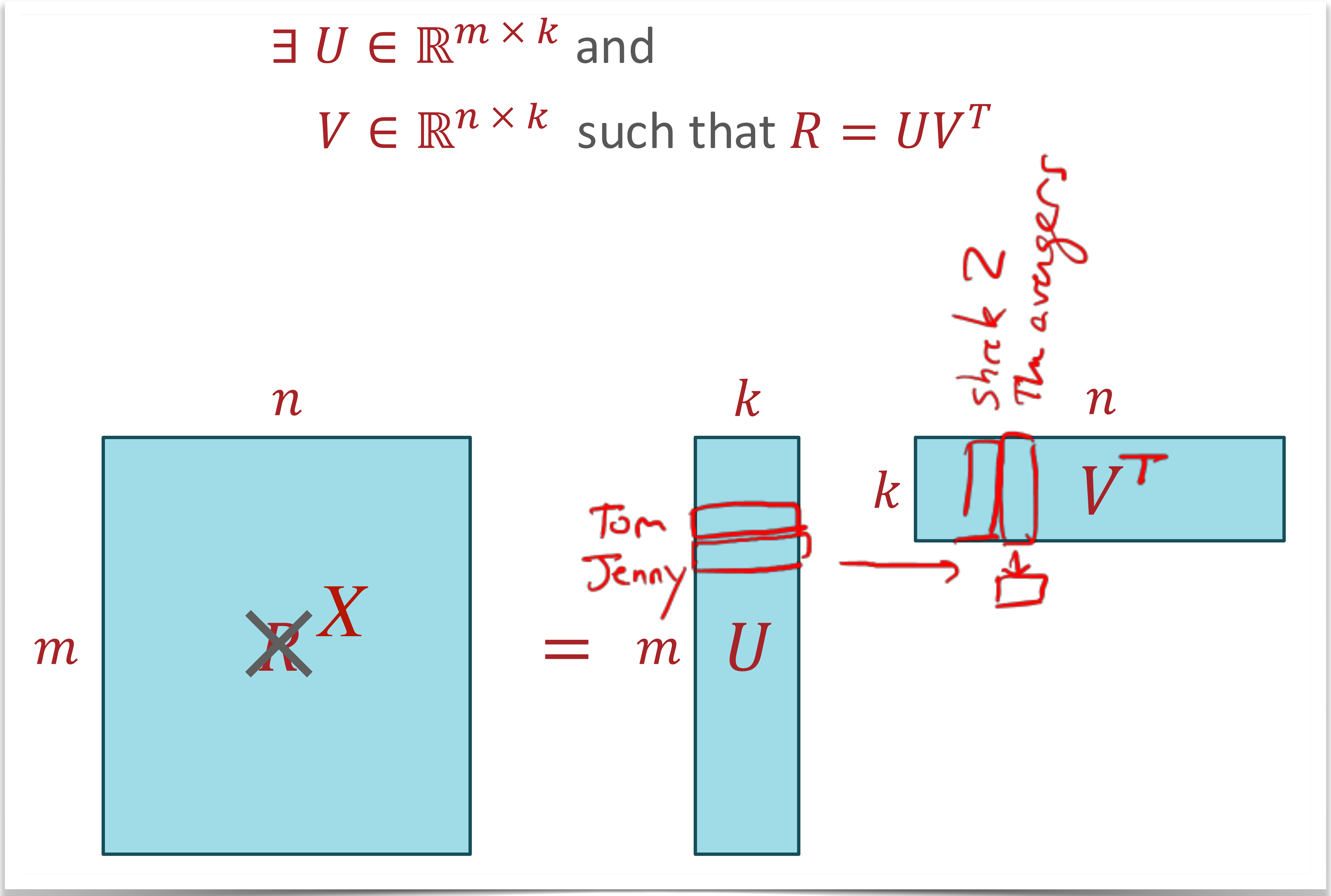
$$u^{(i)} = V^\dagger x^{(i)} \rightarrow (V^T V)^{-1} V^T x^{(i)}$$

$$W^T = V^\dagger$$

Find U, V first

- Min-norm solution: $\mathbf{u}^{(i)} = V^{\dagger}\mathbf{x}^{(i)} = W^{\top}\mathbf{x}^{(i)}$
  - or in matrix form $U = XW$
- Best $\mathbf{u}^{(i)}$ is a linear function of $\mathbf{x}^{(i)}$, even though we didn't constrain it to be
- So, to optimize a linear autoencoder, just need to find U and V, then calculate W as above
  - $\min_{\mathbf{u}^{(i)},\mathbf{v}_j} \sum_{ij} (\mathbf{v}_j^{\top}\mathbf{u}^{(i)} - \mathbf{x}_j^{(i)})^2$ or $\min_{U,V} \|UV^{\top} - X\|_F^2$
  - can use block coordinate descent (alternating optimization)

We saw almost the same model already!



from Lecture 23

- If we set data matrix X = users × movies ratings, we get collaborative filtering by matrix factorization!
  - in autoencoder, just like in collaborative filtering, it's OK if some elements of X are missing (not observed)

Centering and bias weights

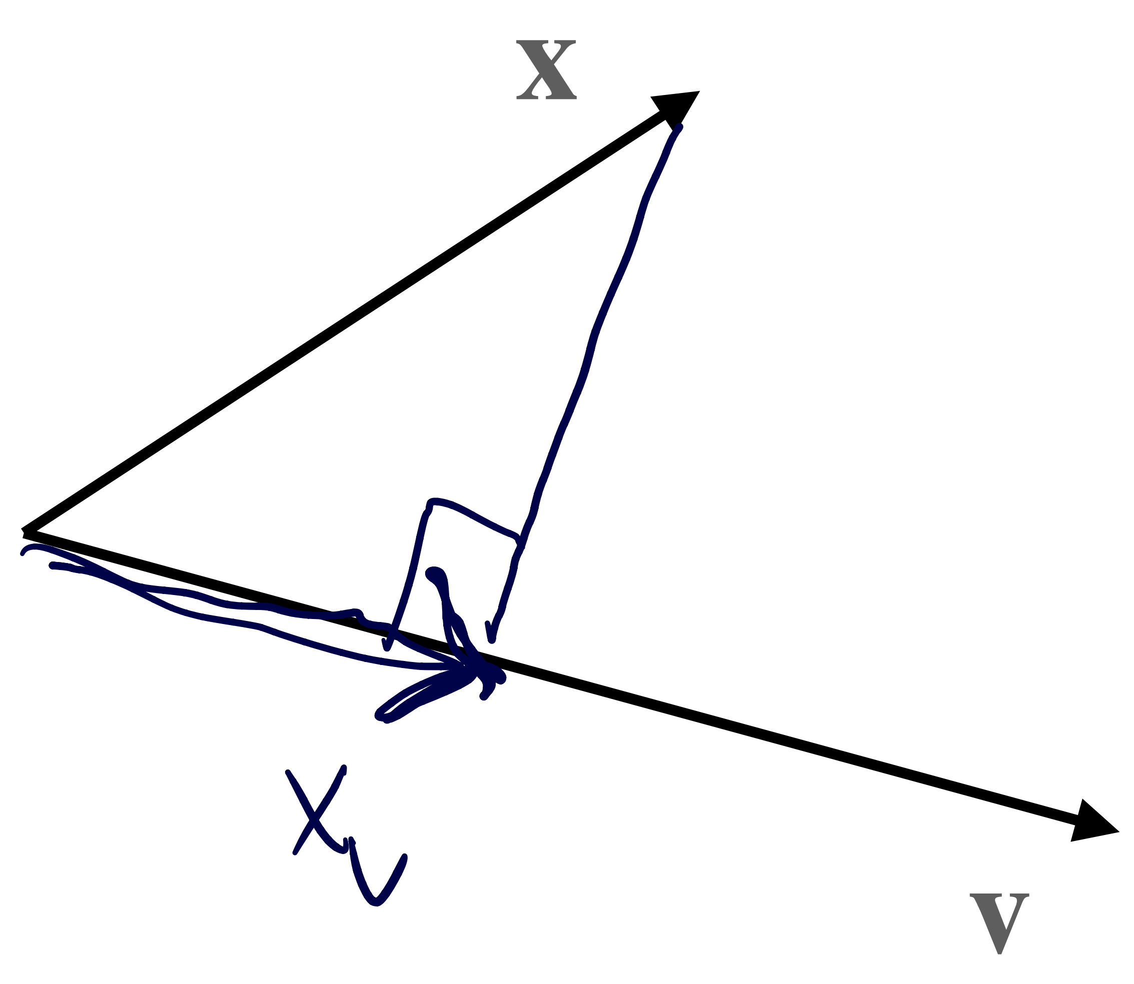
- What if we included bias weights:
  - $\hat{\mathbf{x}}^{(i)} = V\mathbf{u}^{(i)} + \mathbf{b}$ and $\mathbf{u}^{(i)} = W^\top \mathbf{x}^{(i)} + \mathbf{c}$
- Turns out the optimal biases satisfy
  - $\mathbf{b} = \bar{\mathbf{x}} = \frac{1}{n} \sum_{i=1}^{N} \mathbf{x}^{(i)}$ and $\mathbf{c} = -W^\top \bar{\mathbf{x}}$
  - i.e., subtract mean before fitting, $\mathbf{u}^{(i)} = W^\top (\mathbf{x}^{(i)} - \bar{\mathbf{x}})$
  - and add mean back in at end, $\hat{\mathbf{x}}^{(i)} = V\mathbf{u}^{(i)} + \bar{\mathbf{x}}$
- Algorithm:
  - first ***center*** data, $\mathbf{x}^{(i)}_{\text{center}} = \mathbf{x}^{(i)} - \bar{\mathbf{x}}$
  - then fit autoencoder to $\mathbf{x}^{(i)}_{\text{center}}$ *without* bias weights (block coordinate descent, above)
  - then set $\mathbf{b}, \mathbf{c}$ as above (i.e., add $\bar{\mathbf{x}}$ back into predictions)

Even simpler

- Simplest linear autoencoder: $k = 1$ hidden dimension
  - solution is some $\mathbf{v}_1 \in \mathbb{R}^d$ and (from above) $\mathbf{w}_1 = \frac{\mathbf{v}_1}{\mathbf{v}_1^\top \mathbf{v}_1}$
  - scaling ambiguity: might as well pick $\|\mathbf{v}_1\| = 1$, $\mathbf{w}_1 = \mathbf{v}_1$
- That is, the $k = 1$ autoencoder picks a unit vector $\mathbf{v}_1$ and projects $\mathbf{x}^{(i)}$ onto $\mathbf{v}_1$

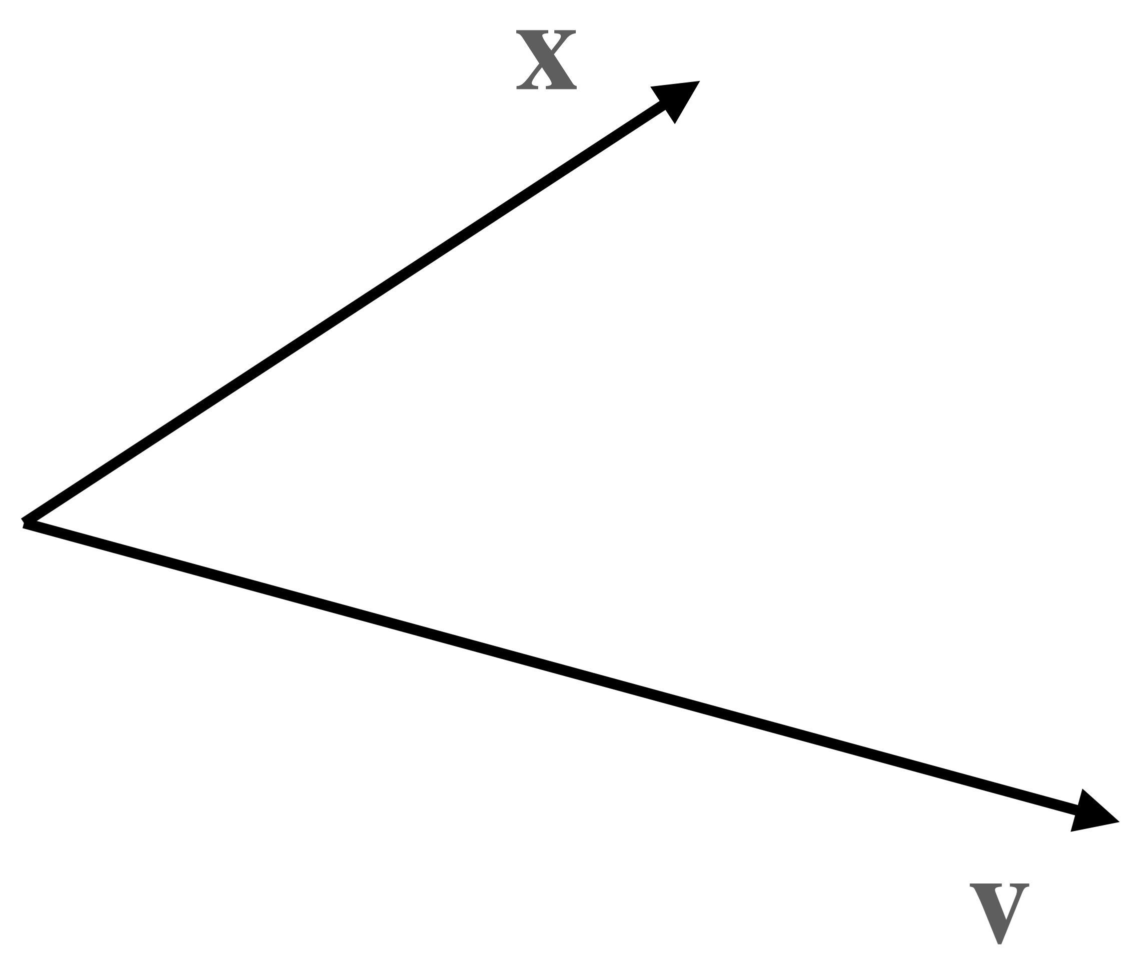
Reminder: vector projection

- Length of projection of $\mathbf{x}$ onto $\mathbf{v}$:
  - $a = \mathbf{v}^\top \mathbf{x}$ if $\|\mathbf{v}\| = 1$
- Projection of $\mathbf{x}$ onto $\mathbf{v}$:
  - $\hat{\mathbf{x}} = \mathbf{v}(\mathbf{v}^\top \mathbf{x})$ if $\|\mathbf{v}\| = 1$



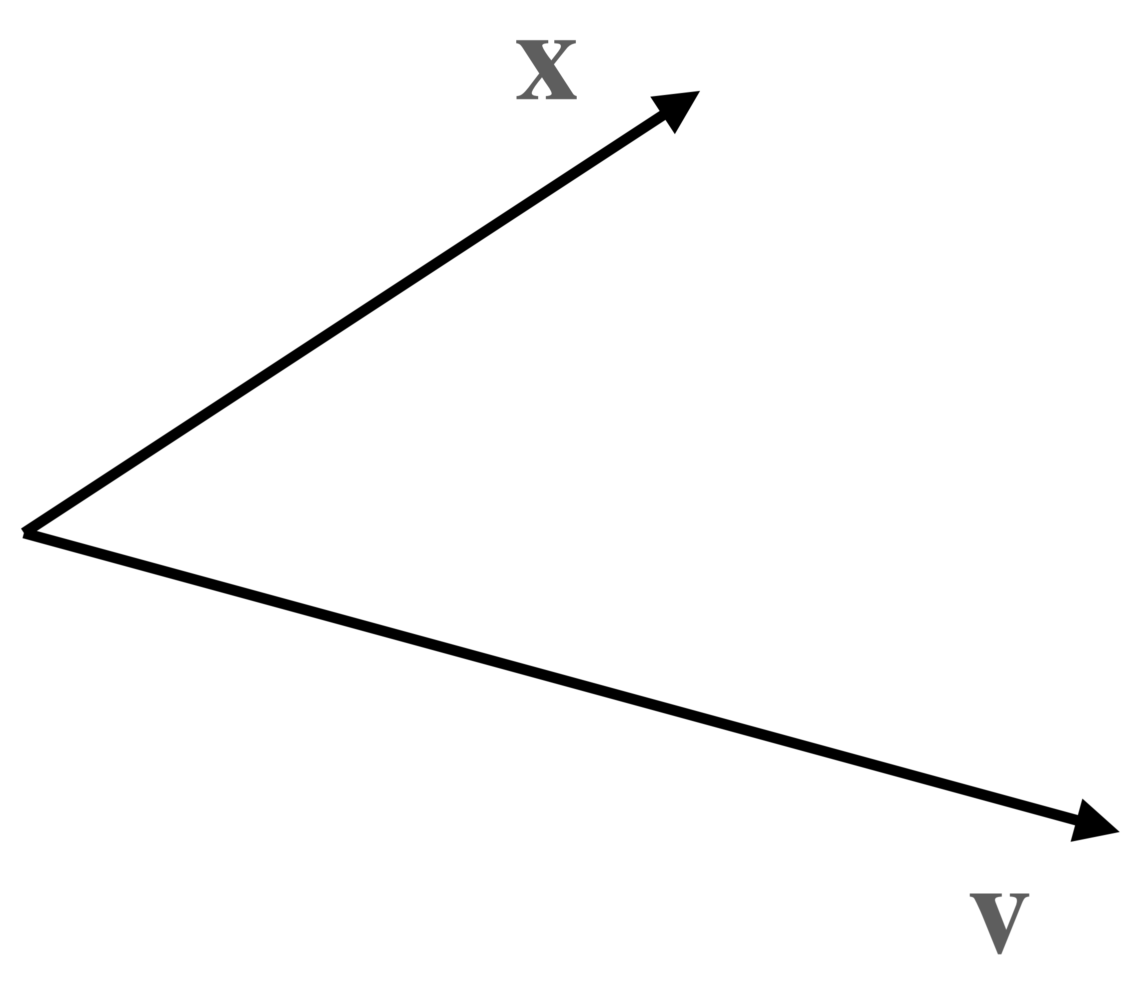
Reminder: vector projection

- Length of projection of $\mathbf{x}$ onto $\mathbf{v}$:
  - $a = \frac{\mathbf{v}^\top \mathbf{x}}{\|\mathbf{v}\|}$ if $\|\mathbf{v}\| = 1$ o/w
- Projection of $\mathbf{x}$ onto $\mathbf{v}$:
  - $\hat{\mathbf{x}} = \mathbf{v}(\mathbf{v}^\top \mathbf{x})$ if $\|\mathbf{v}\| = 1$

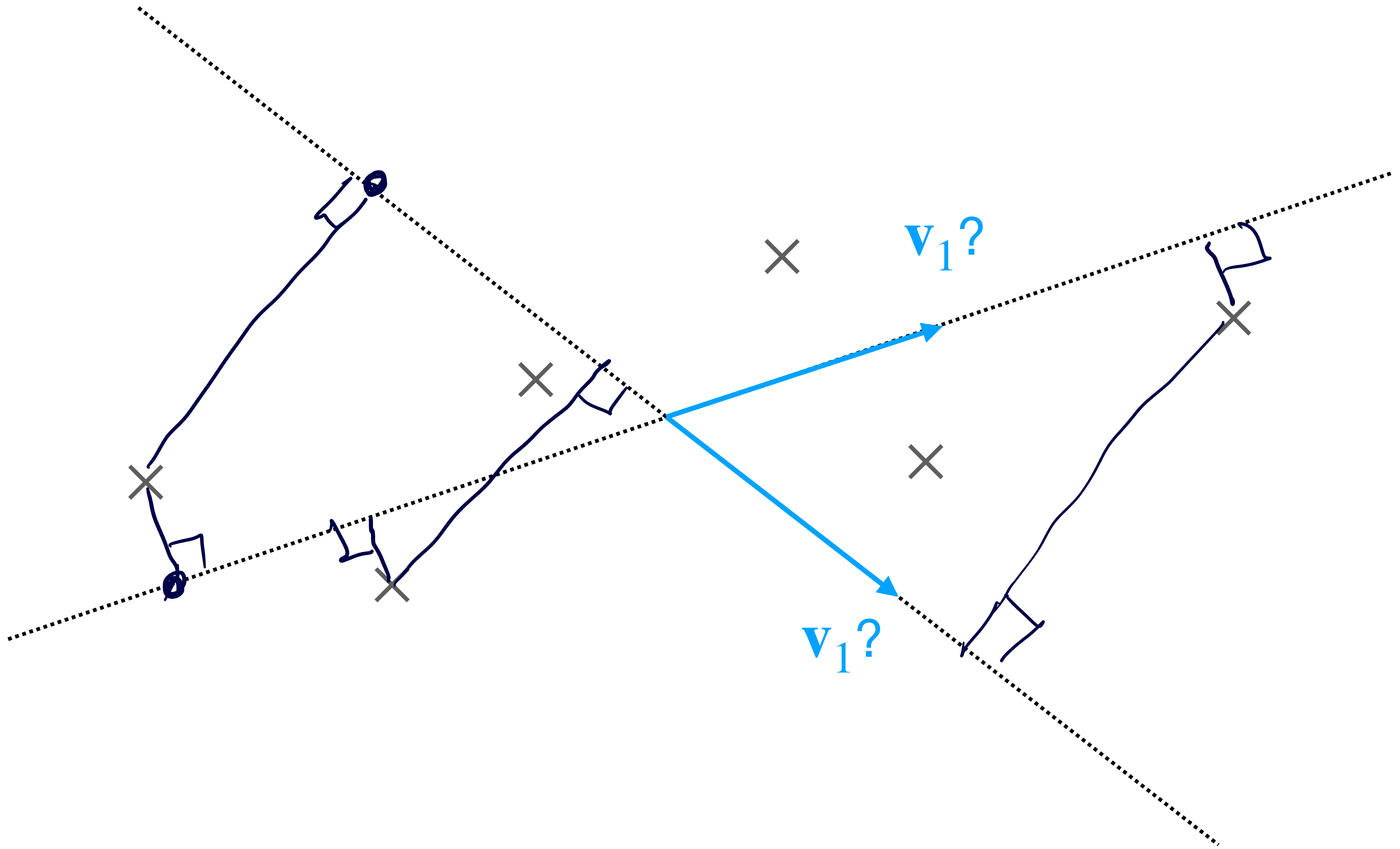


Reminder: vector projection

- Length of projection of $\mathbf{x}$ onto $\mathbf{v}$:
  - $a = \dfrac{\mathbf{v}^\top \mathbf{x}}{\|\mathbf{v}\|}$ if $\|\mathbf{v}\| = 1$ o/w
- Projection of $\mathbf{x}$ onto $\mathbf{v}$:
  - $\hat{\mathbf{x}} = \dfrac{\mathbf{v}(\mathbf{v}^\top \mathbf{x})}{\|\mathbf{v}\|^2}$ if $\|\mathbf{v}\| = 1$ o/w



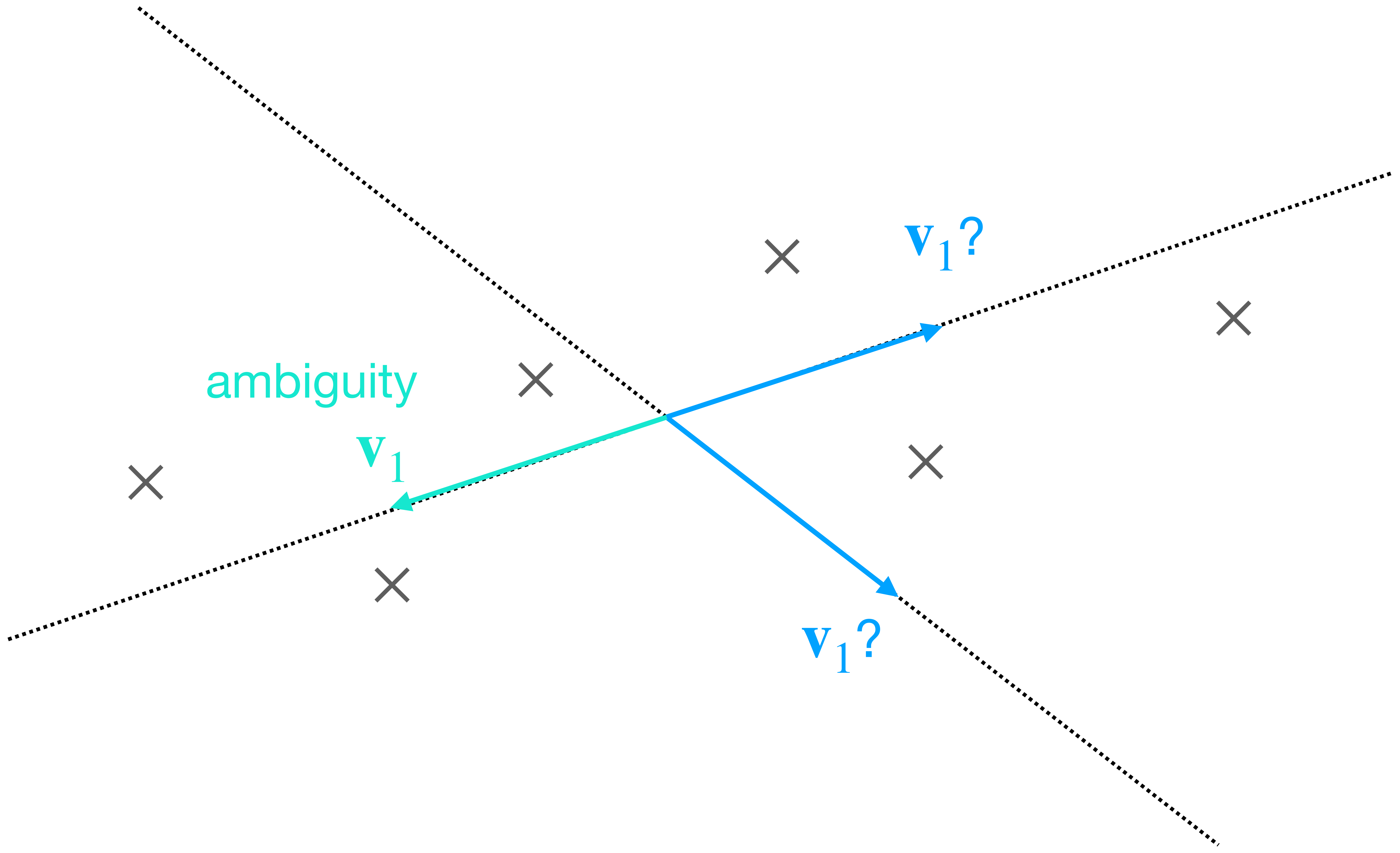
Which vector?



- We find $\mathbf{v}_1$ by minimizing (sum or mean) squared reconstruction error
  - intuitively: if datapoints are spread out, $\mathbf{v}_1$ should point along the long direction

Geoff Gordon

Which vector?



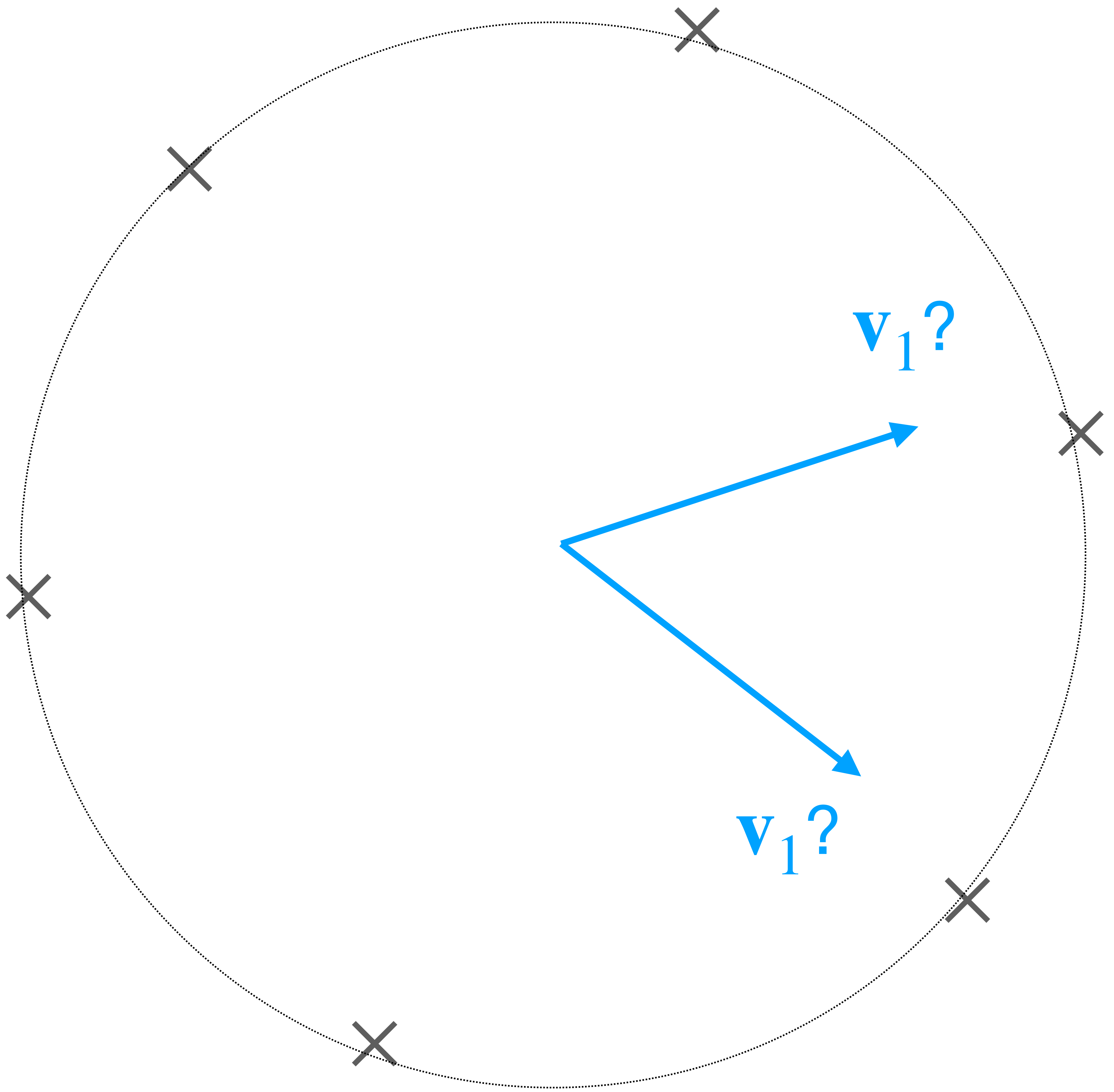
- We find $\mathbf{v}_1$ by minimizing (sum or mean) squared reconstruction error
  - intuitively: if datapoints are spread out, $\mathbf{v}_1$ should point along the long direction

Ambiguity

- Could be a tie for best $\mathbf{v}_1$ if points are equally spread out in two (or more) directions
  - if so, infinitely many solutions
  - we can break the tie arbitrarily

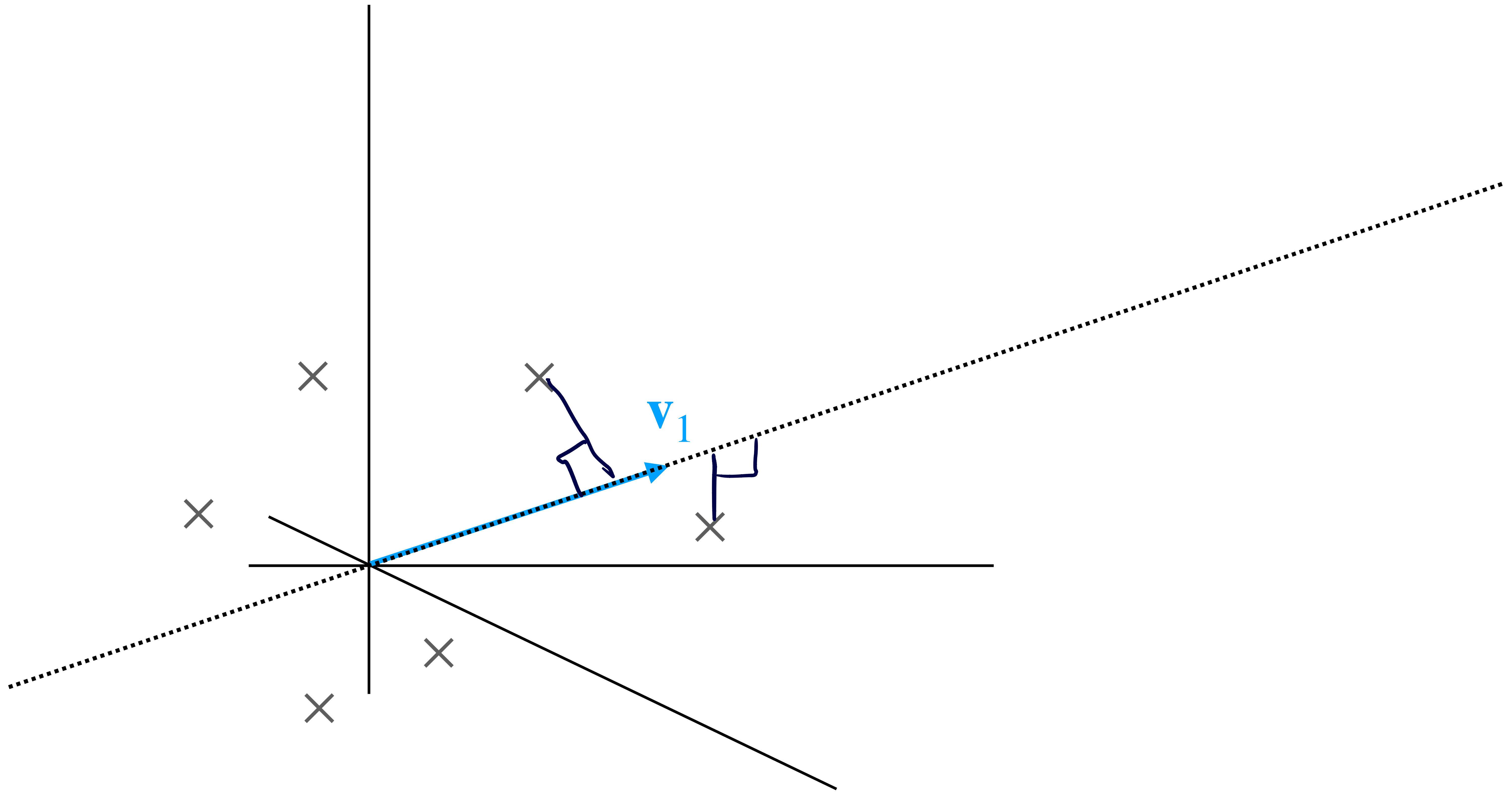
Geoff Gordon

Ambiguity



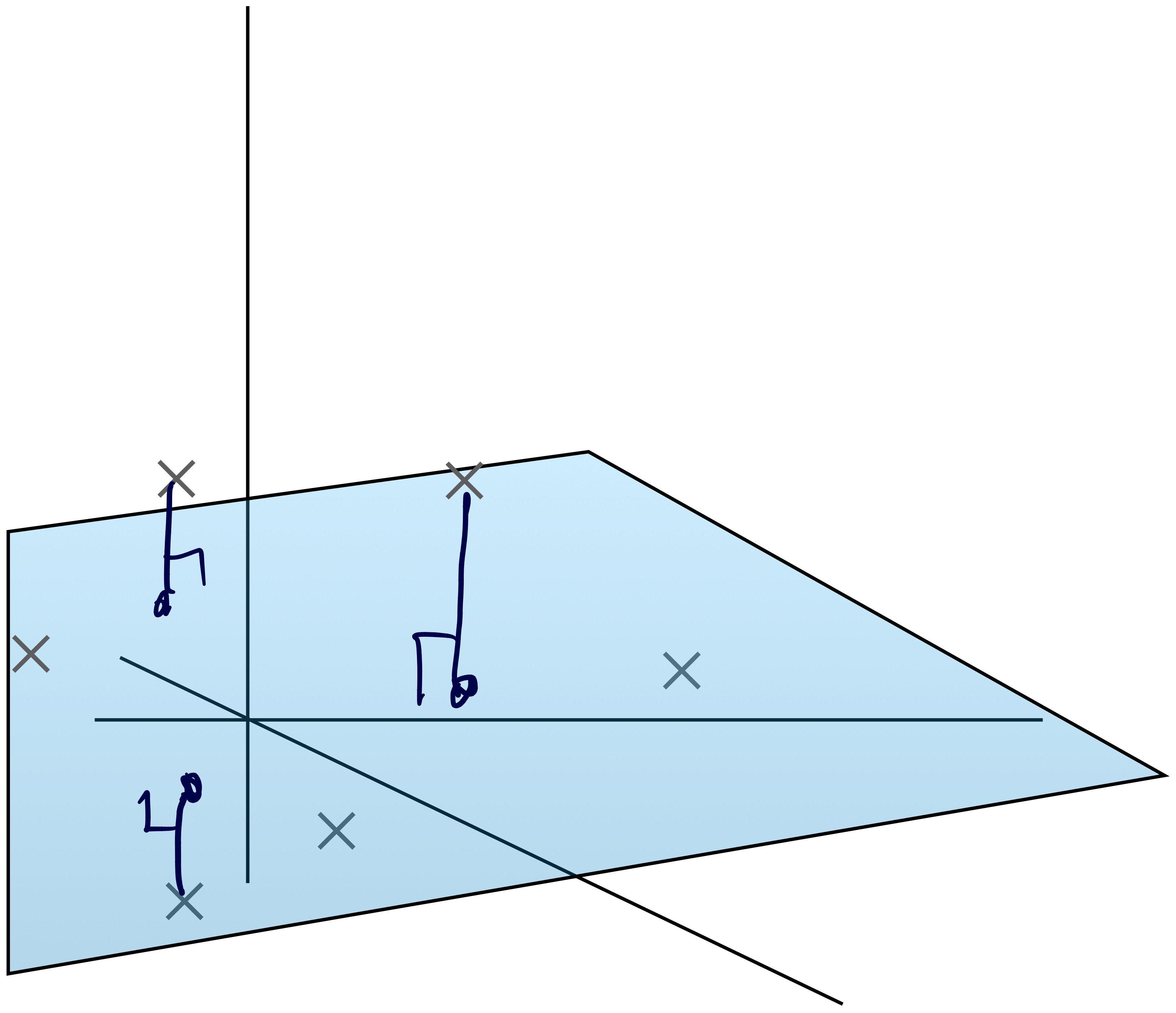
- Could be a tie for best $\mathbf{v}_1$ if points are equally spread out in two (or more) directions
  - if so, infinitely many solutions
  - we can break the tie arbitrarily

An example in 3D



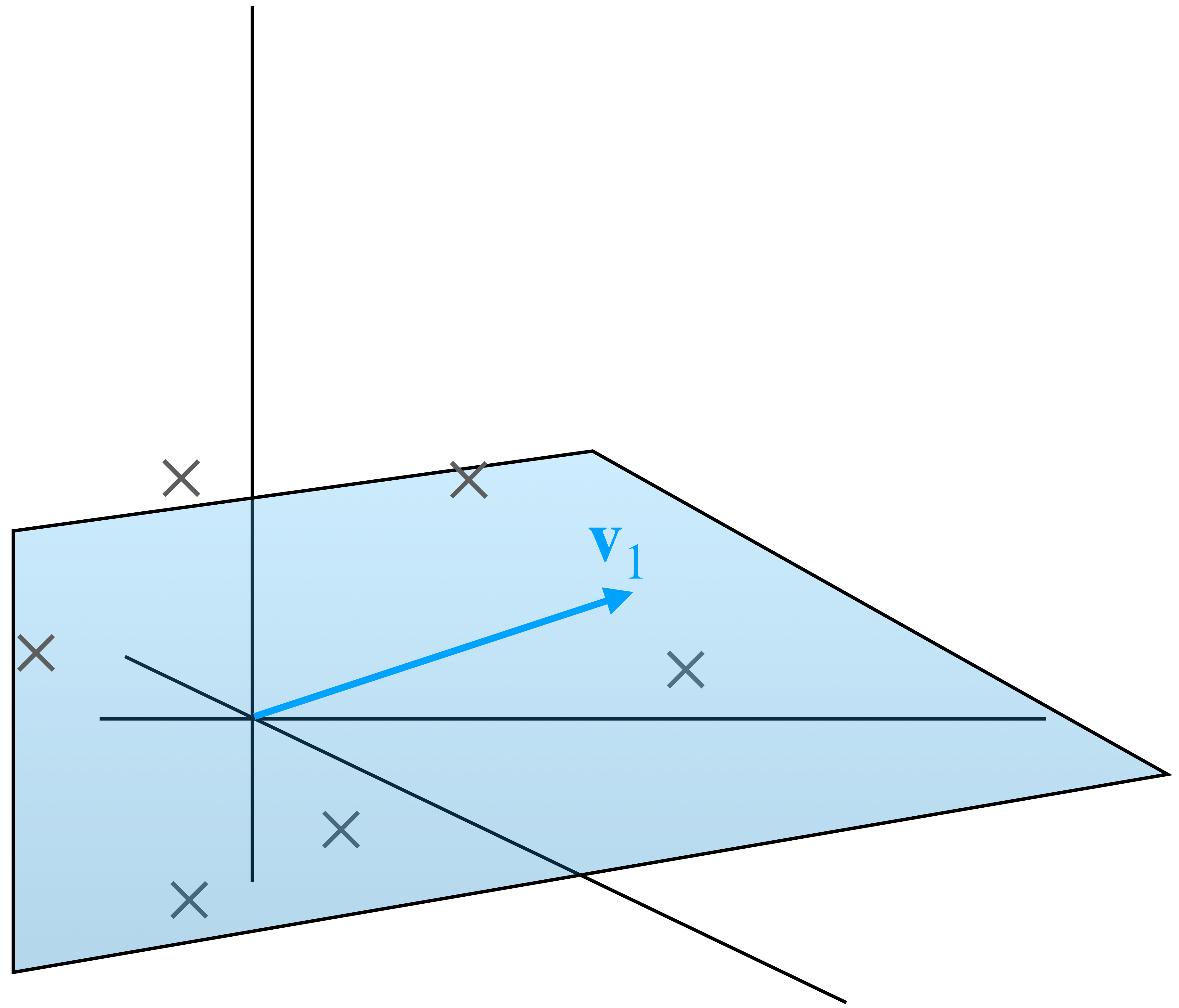
- $\mathbf{v}_1$ defines a line in 3D — but MSE is still based on orthogonal distance to the line

Best plane



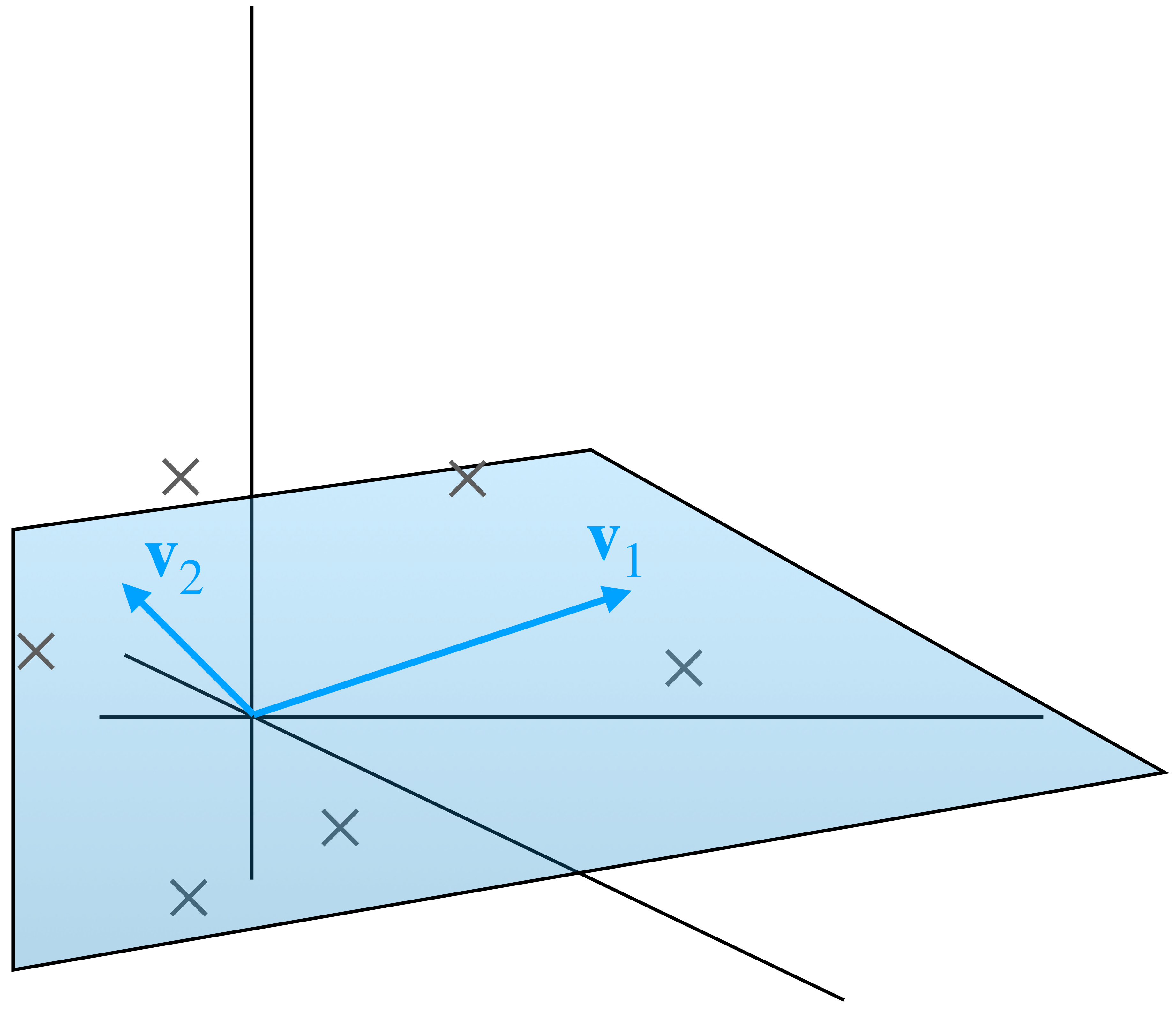
- What about $k = 2$?
  - project onto best 2D plane (min MSE)

Nesting



- Best plane contains the $\mathbf{v}_1$ from $k = 1$ solution
  - else we'd reduce MSE by rotating the plane towards $\mathbf{v}_1$
  - i.e., solutions are ***nested:*** if $k < k'$, solution for k is contained in solution for k' (considered as subspaces)

Orthogonal basis



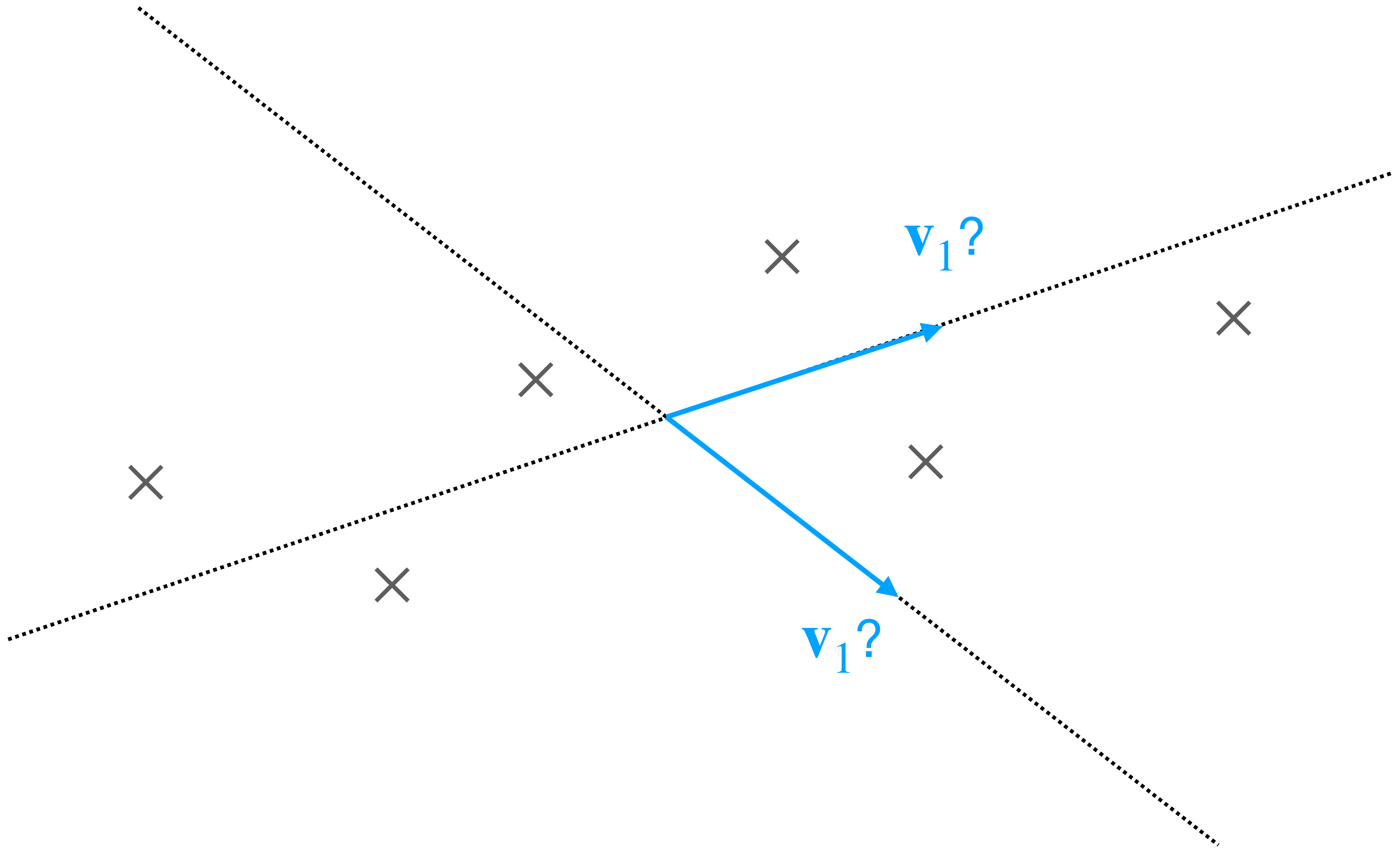
- Can choose any basis for the plane — might as well pick $\mathbf{v}_1$ and a vector $\mathbf{v}_2$ that's orthogonal to $\mathbf{v}_1$ (ie, $\mathbf{v}_2^\top \mathbf{v}_1 = 0$)

PCA

nb: not "principle"!

- Definition: ***principal components analysis***
  - the 1st principal component is the unit vector that minimizes mean-squared reconstruction error $\mathbf{v}_1 = \arg\min_{\mathbf{v}} \frac{1}{N}\sum_{i=1}^{N} \|\mathbf{x}^{(i)} - (\mathbf{v}^\top \mathbf{x}^{(i)})\mathbf{v}\|^2$ st $\|\mathbf{v}\| = 1$
  - construct residuals $\mathbf{e}_1^{(i)} = \mathbf{x}^{(i)} - (\mathbf{v}_1^\top \mathbf{x}^{(i)})\mathbf{v}_1$
  - the 2nd PC is the unit vector that minimizes mean-squared reconstruction error of the residuals, *while remaining orthogonal to* $\mathbf{v}_1$ $\mathbf{v}_2 = \arg\min_{\mathbf{v}} \frac{1}{N}\sum_{i=1}^{N} \|\mathbf{e}_1^{(i)} - (\mathbf{v}^\top \mathbf{e}_1^{(i)})\mathbf{v}\|^2$ st $\|\mathbf{v}\| = 1,\ \mathbf{v}^\top \mathbf{v}_1 = 0$
  - construct residuals $\mathbf{e}_2^{(i)} = \mathbf{e}_1^{(i)} - (\mathbf{v}_2^\top \mathbf{e}_1^{(i)})\mathbf{v}_2$
  - …
  - the kth principal component is the vector that minimizes mean-squared reconstruction error *while remaining orthogonal to* $\mathbf{v}_1 \ldots \mathbf{v}_{k-1}$

Thinking about objectives for PCA



- Best $\mathbf{v}_1$ (min MSE) points along the long direction
  - side effect: the projections are spread out more
  - maybe another reasonable objective: maximize variance of the projections

Geoff Gordon

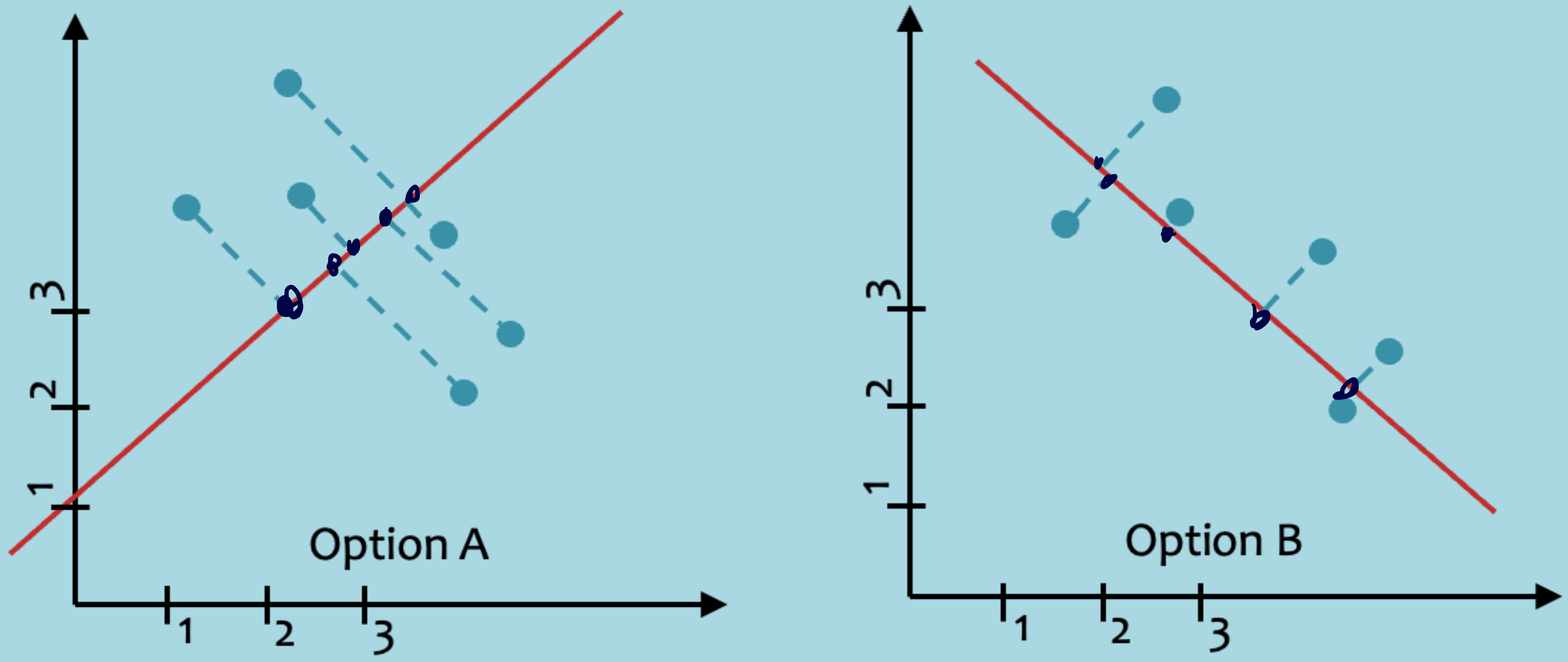
C toxic

Comparison of objectives

Below are two plots of the same dataset D. Consider the two projections shown.

1. **Poll Question 1:** Which maximizes the variance?
2. **Poll Question 2:** Which minimizes the reconstruction error?

Answer:



Geoff Gordon

Equivalence

- Minimizing the reconstruction error is ***the same as*** maximizing the variance of projections
  - $||\mathbf{x} - (\mathbf{v}^\top \mathbf{x})\mathbf{v}||^2 =$ *reconstruction error*

$$x^T x - 2\underbrace{x^T v\,(v^T x)}_{(v^T x)^2} + (v^T x)^2\, v^T v$$

$$x^T x - (v^T x)^2$$

  - $\min_{\mathbf{v}} \frac{1}{N} \sum_{i=1}^{N} [(\mathbf{x}^{(i)})^\top \mathbf{x}^{(i)} - (\mathbf{v}^\top \mathbf{x}^{(i)})^2] =$

$$\max \frac{1}{N} \sum_{i=1}^{N} (v^T x^{(i)})^2$$

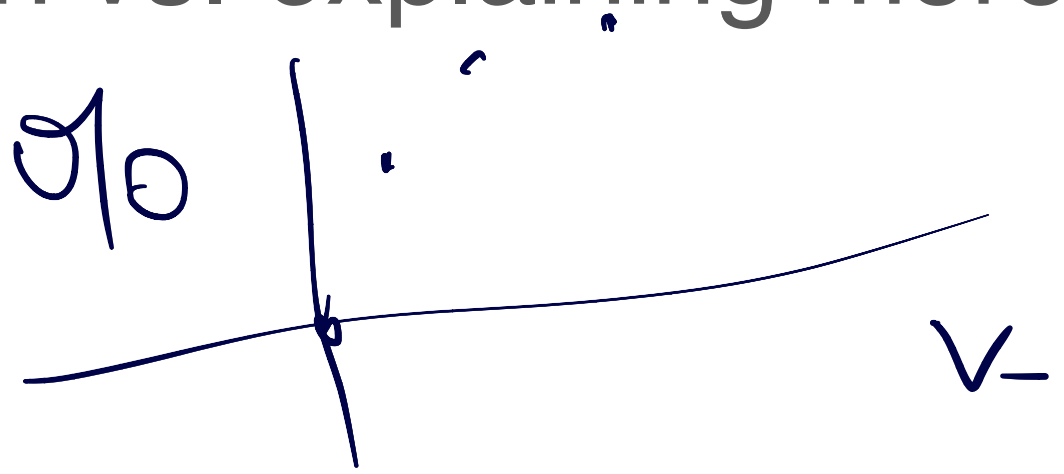
variance of projections

Principal values

- Definition: the jth ***principal value*** is the variance of the projections onto the jth principal component $\mathbf{v}_j$
  - because PCs are orthogonal, the sum of the principal values is equal to the variance of $\mathbf{x}^{(i)}$
- Often report ***fraction of variance explained*** by a PC:

$$\frac{\frac{1}{N}\sum_i (\mathbf{v}_j^\top \mathbf{x}^{(i)})^2}{\frac{1}{N}\sum_i (\mathbf{x}^{(i)})^\top \mathbf{x}^{(i)}} = \frac{\text{variance of projection}}{\text{variance of } \mathbf{x}^{(i)}}$$

- Or plot the ***cumulative sum*** of variance explained (sum over first j PCs vs. j)
  - to decide how many PCs to use (trade off small latent representation vs. explaining more variance)



Covariance matrix

- Maximize variance of projections

$$\frac{1}{N}\sum_i (\mathbf{v}_j^\top \mathbf{x}^{(i)})^2 = \frac{1}{N}\sum_i \mathbf{v}_j^\top \left[\mathbf{x}^{(i)}(\mathbf{x}^{(i)})^\top\right] \mathbf{v}_j = \mathbf{v}_j^\top \Sigma \mathbf{v}_j$$

- Here $\Sigma = \frac{1}{N}\sum_i \mathbf{x}^{(i)}(\mathbf{x}^{(i)})^\top$ is the (sample) ***covariance matrix*** of datapoints $\mathbf{x}^{(i)}$
  - diagonal elements: variance of each feature $\mathbf{x}_j^{(i)}$
  - off-diagonal: covariance of pair of features

ex: $X = \begin{pmatrix} 1 & -1 \\ -1 & 1 \\ 2 & 2 \\ -2 & -2 \end{pmatrix}$ $\quad \Sigma = \begin{pmatrix} \frac{5}{2} & \frac{3}{2} \\ \frac{3}{2} & \frac{5}{2} \end{pmatrix}$

Geoff Gordon

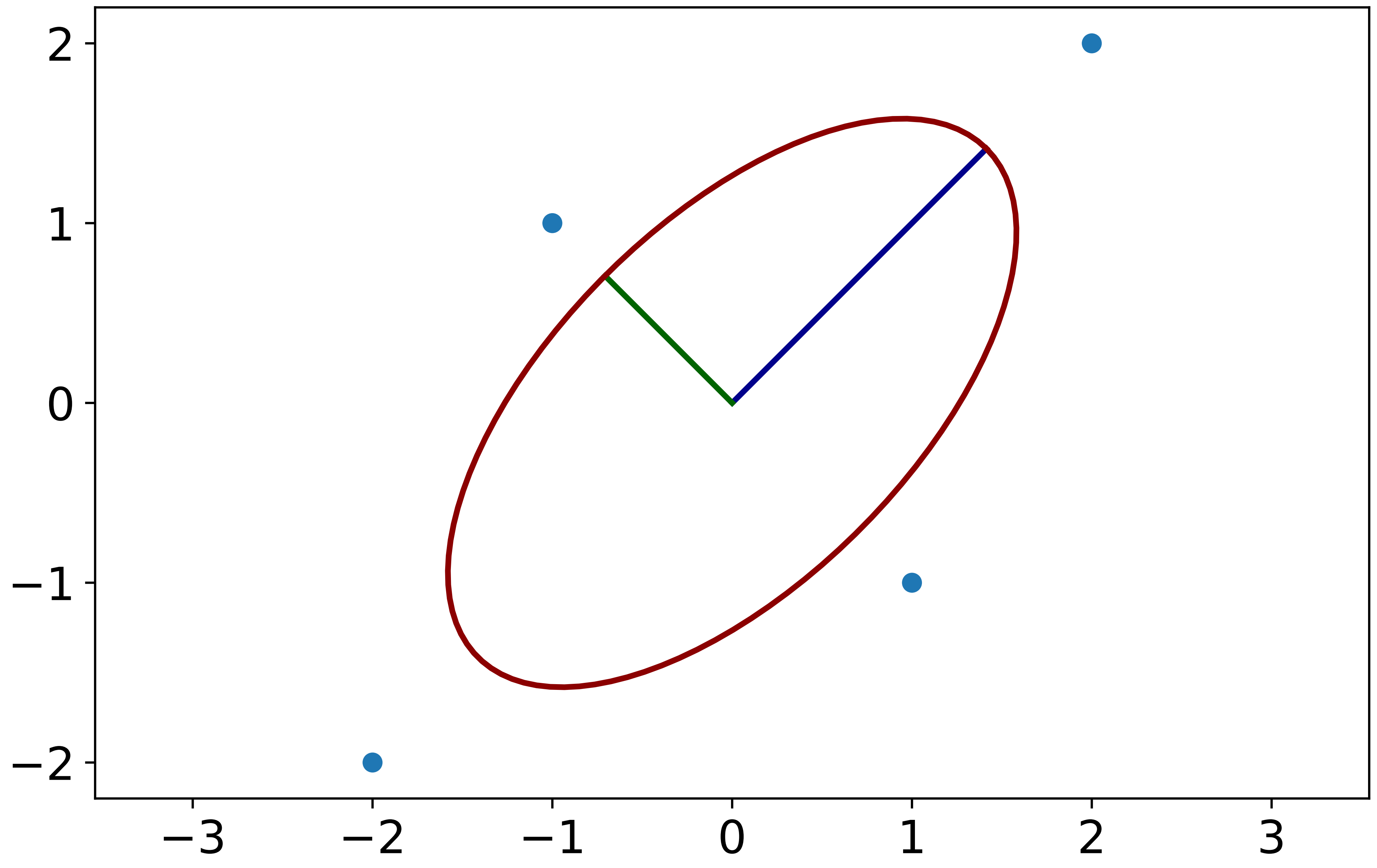
Eigenvalues and eigenvectors

$$\lambda = \max_{||\mathbf{v}||=1} \mathbf{v}^\top \Sigma \mathbf{v}$$

- This is exactly the definition of largest eigenvalue of Σ
  - and the arg max $\mathbf{v}_1$ is the corresponding eigenvector
- Similarly, if we maximize over $\mathbf{v}$ with $\mathbf{v}^\top \mathbf{v}_1 = 0$ we get second largest eigenvalue (and its eigenvector)
- So, we can solve PCA by finding eigenvalues and eigenvectors of the covariance matrix!
  - PyTorch has a built-in function for this: `torch.pca_lowrank`

Graphically

$$X = \begin{pmatrix} 1 & -1 \\ -1 & 1 \\ 2 & 2 \\ -2 & -2 \end{pmatrix}$$



PCA finds the longest axes

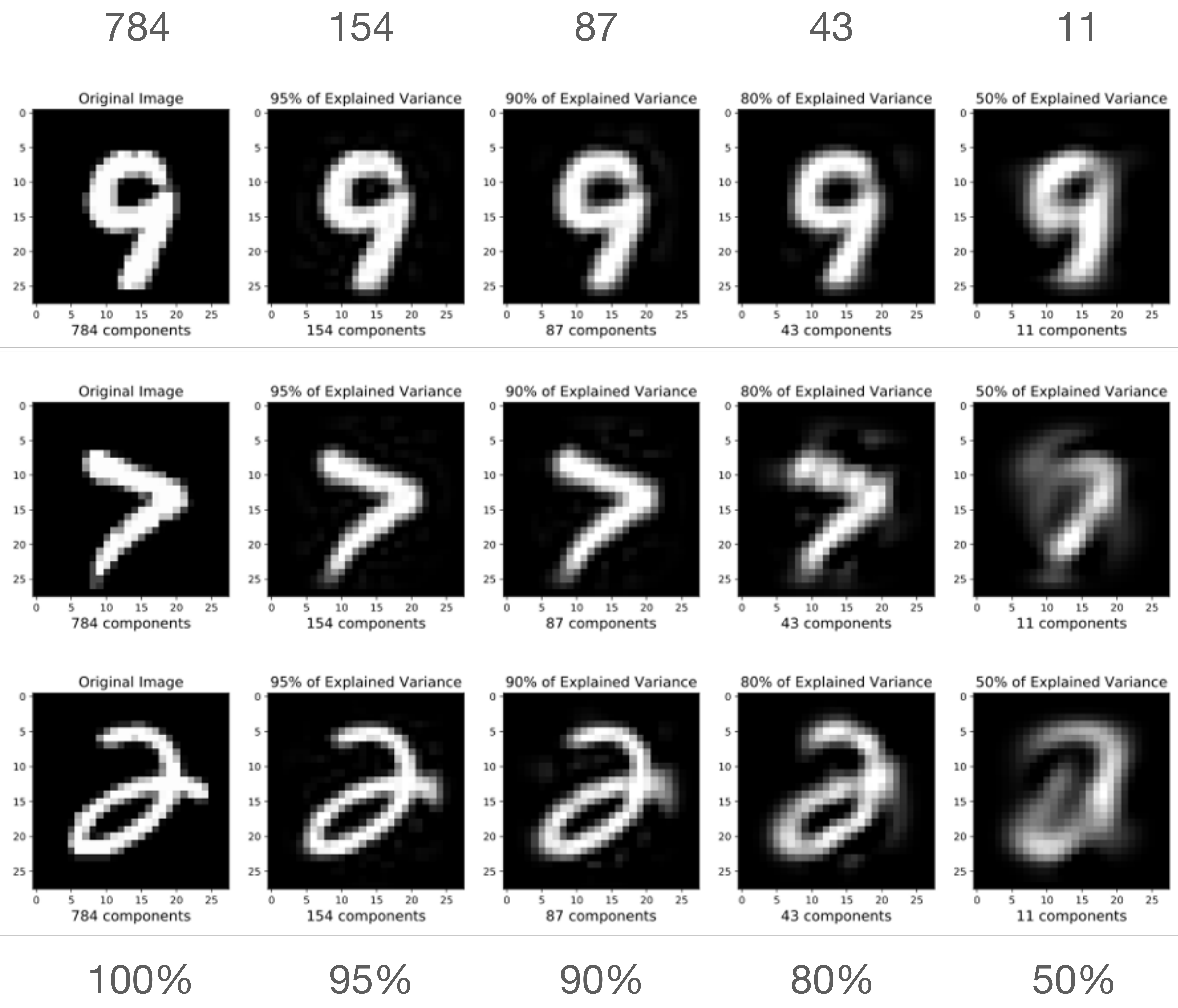
- Visualize Σ by looking at PDF of a Gaussian distribution with covariance Σ (even if our data aren't Gaussian)
- Contours of PDF are ellipses (or ellipsoids in higher D)
- Each eigenvector (each principal component) corresponds to an axis of the ellipse
  - corresponding eigenvalue = (axis length)^2 = variance

Geoff Gordon

PCA example: MNIST digits

- Task: for each latent dimensionality k
  - take each 28×28 image of a digit (a vector $\mathbf{x}^{(i)}$ of length 784) and project it down to $\mathbb{R}^k$
  - report percent of variance explained
  - then project back up to 28×28 image (a vector $\hat{\mathbf{x}}^{(i)}$ of length 784) to visualize what information was preserved

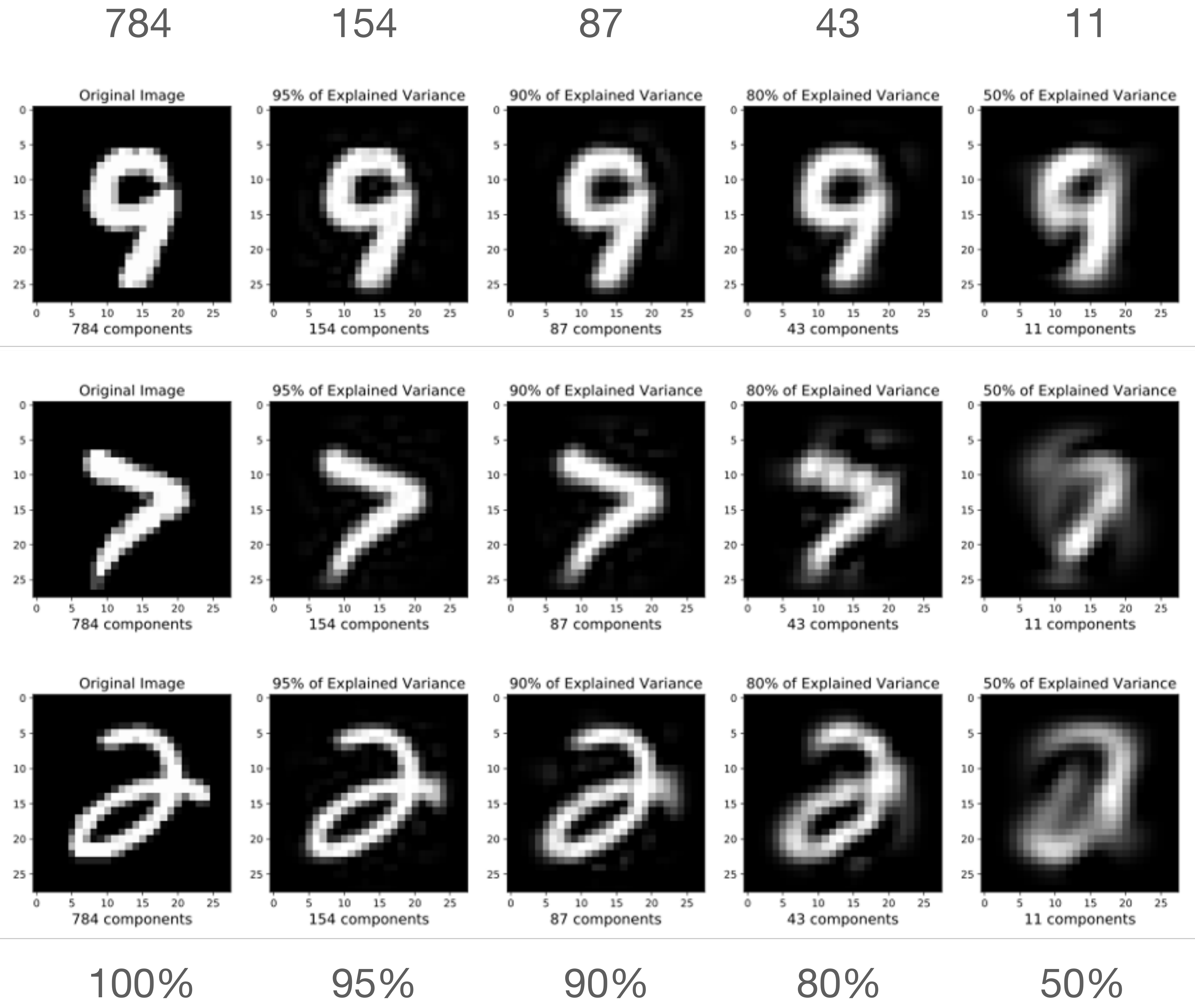
PCA example: MNIST digits



Geoff Gordon

Takeaway:
Using fewer principal components k leads to higher reconstruction error.

But even a small number (say 43) still preserves a lot of information about the original image.

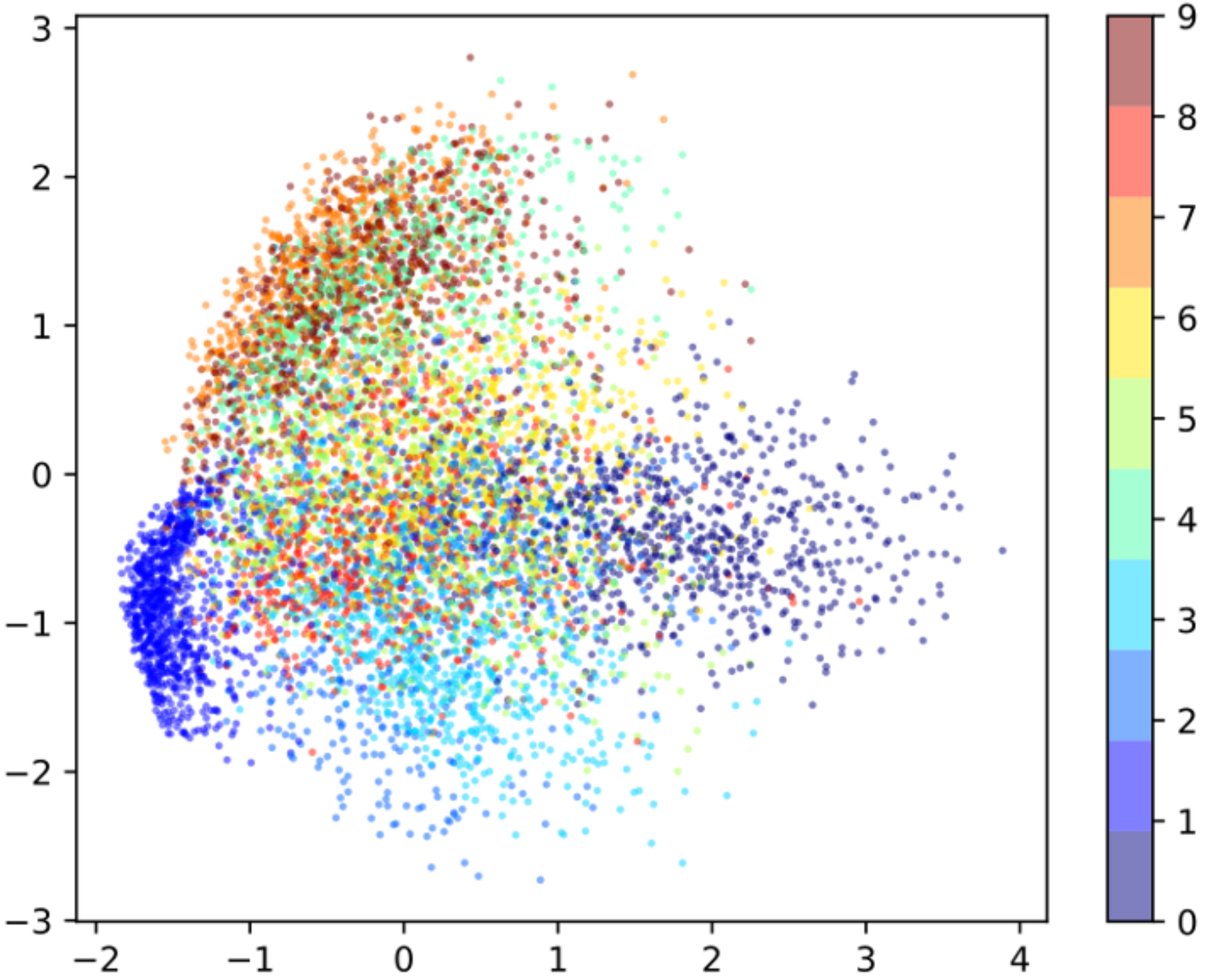


Geoff Gordon

PCA example: MNIST digits

- Task: fix latent dimension $k = 2$
  - take each 28×28 image of a digit (a vector $\mathbf{x}^{(i)}$ of length 784) and project it down to $\mathbb{R}^k$
  - plot the 2D points and color them according to the digit label $y^{(i)}$ (which is unknown to PCA)

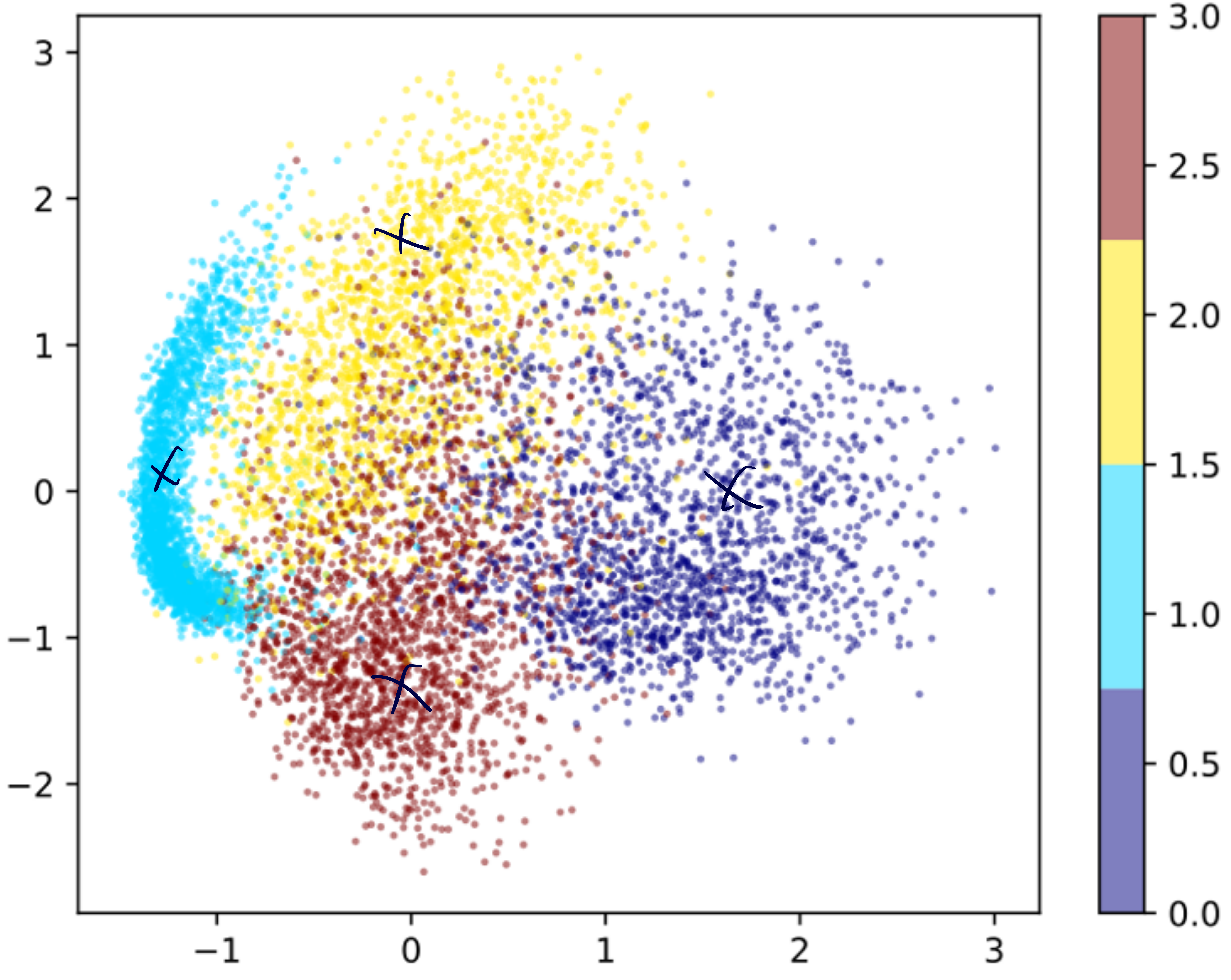
PCA example: MNIST digits



- All 10 digits 0–9

Geoff Gordon

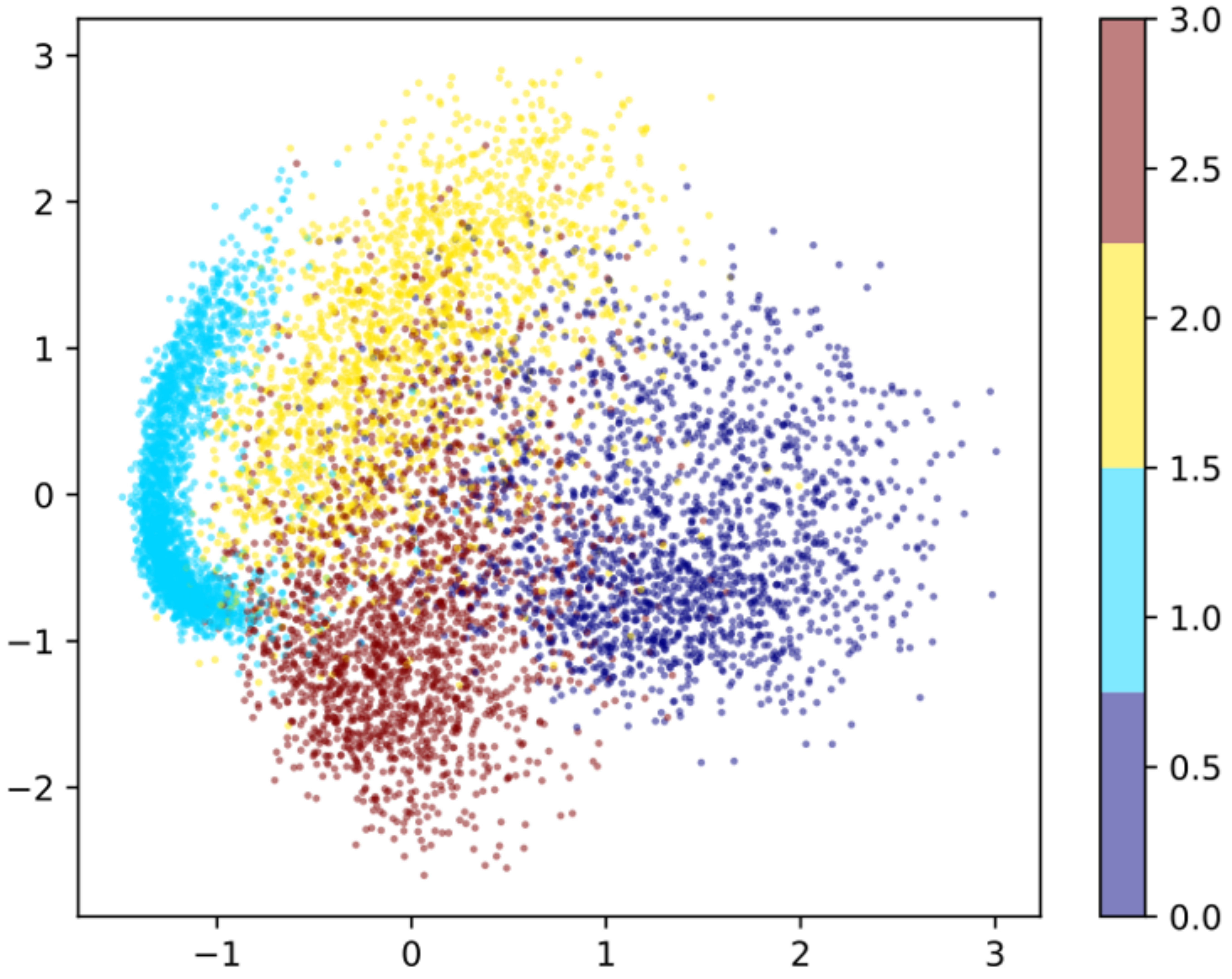
PCA example: MNIST digits



- Just the four digits 0–3

Geoff Gordon

Takeaway:
Even with a tiny number of principal components $k = 2$, PCA learns a representation that captures **latent** information: the type of digit

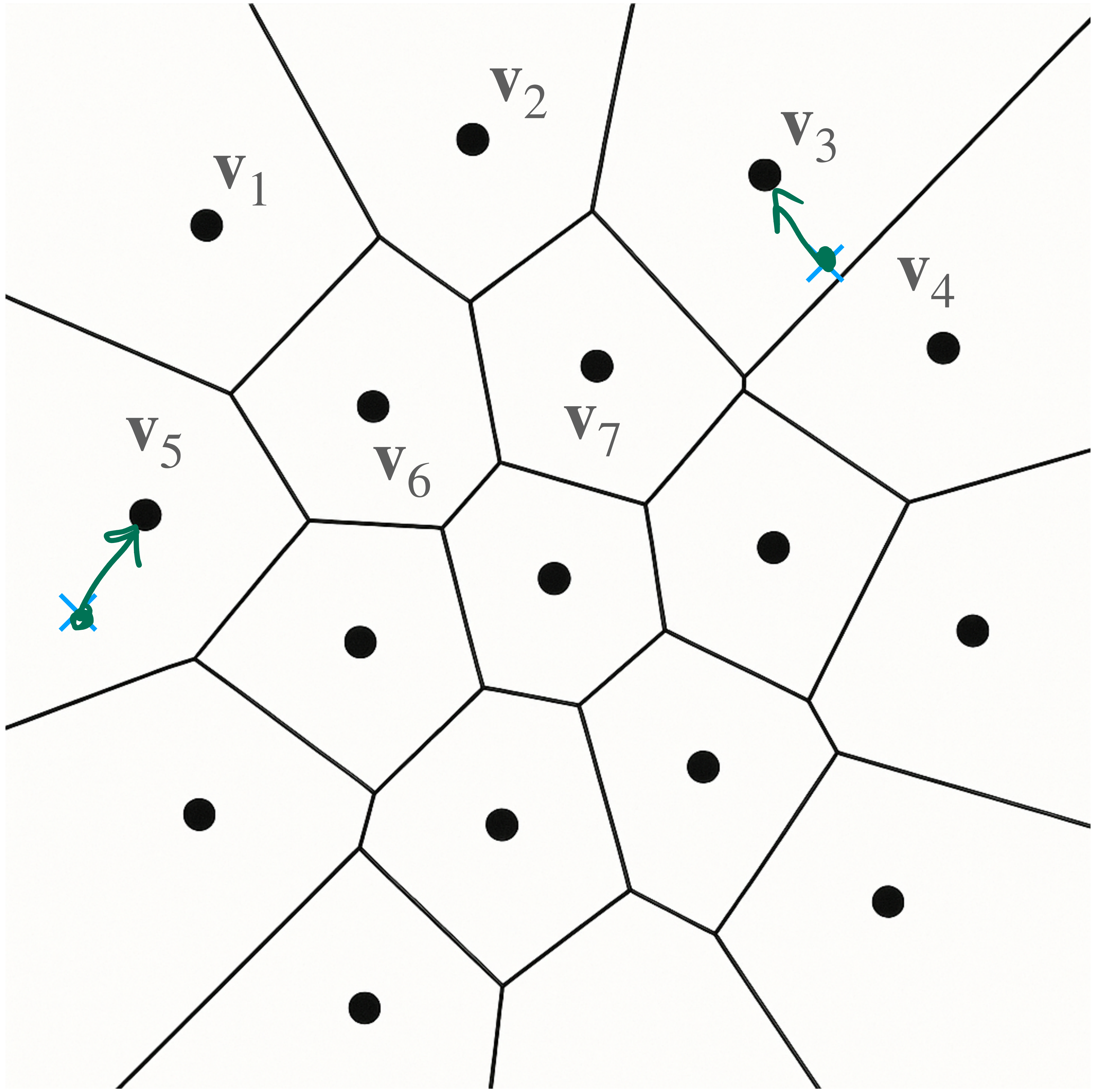


- Just the four digits 0–3

Learning objectives: PCA / dimensionality reduction

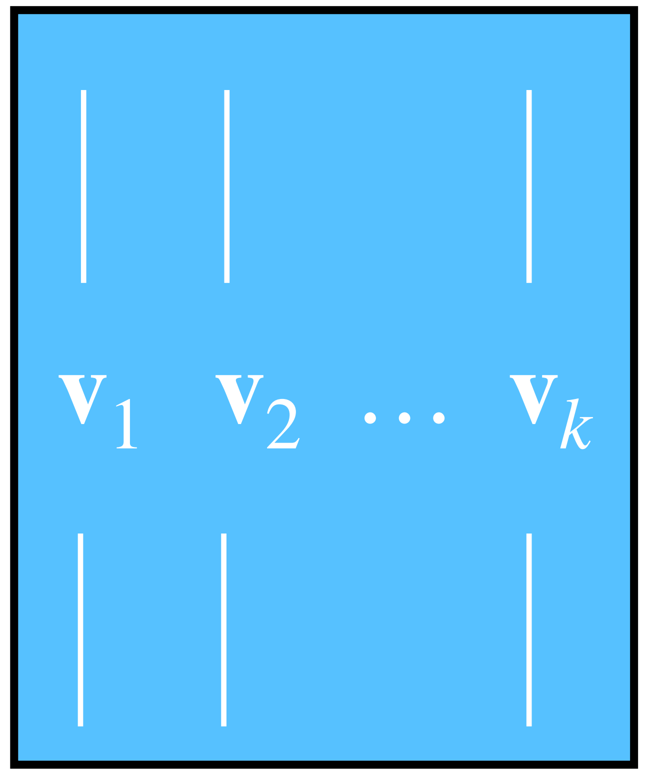
- Identify examples of high dimensional data and common use cases for dimensionality reduction
- Define the sample mean, sample variance, and sample covariance of a vector-valued dataset
- Draw the principal components of a given low-D dataset
- Establish the equivalence of minimization of reconstruction error with maximization of variance
- Given a set of principal components, project from high to low-D space; do the reverse to produce a reconstruction
- Explain the connection between PCA, eigenvectors, eigenvalues, and covariance matrix
- Use common methods in linear algebra to obtain the principal components

Vector quantization



k centers in $\mathbb{R}^d$

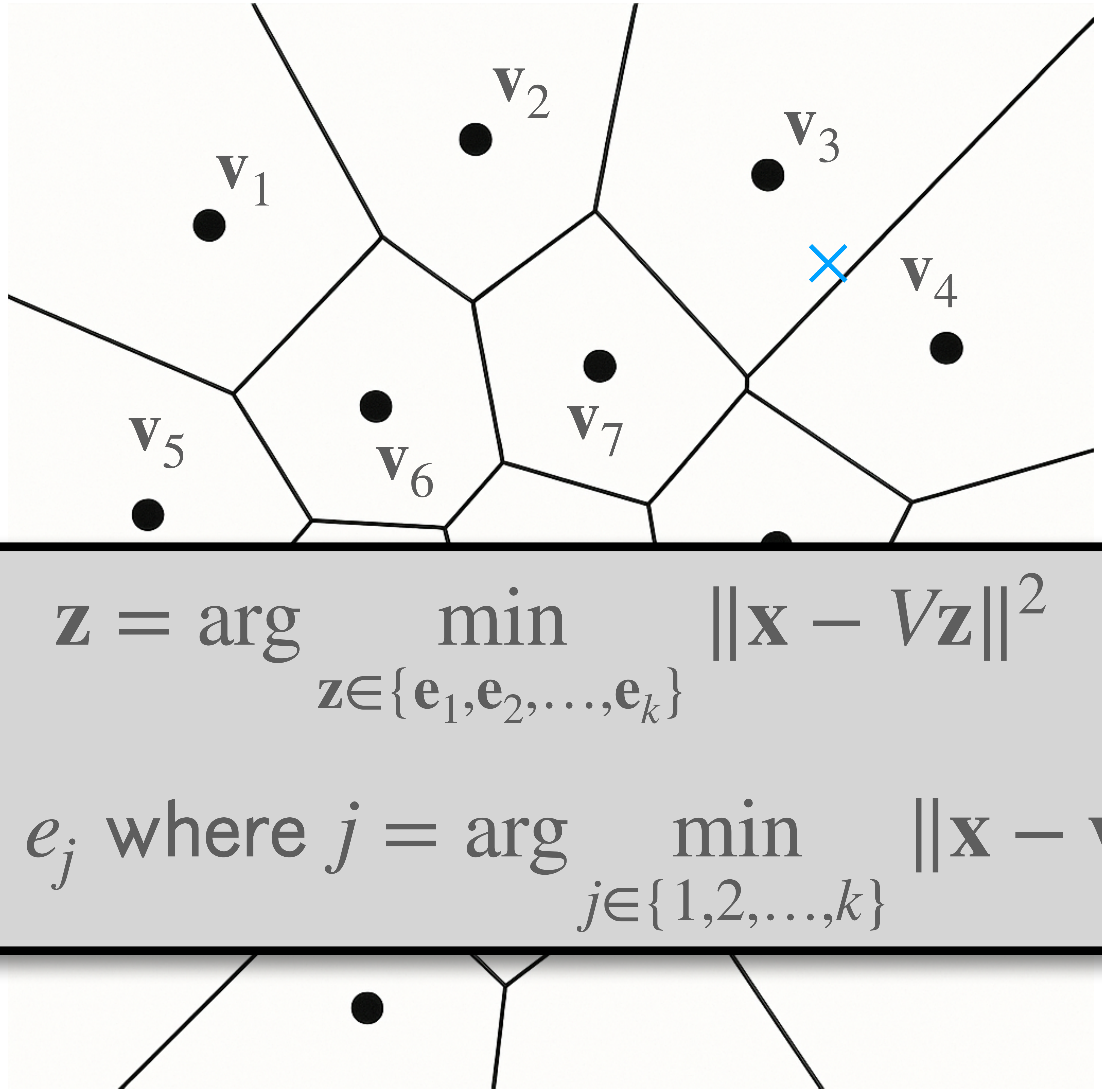
$V \in \mathbb{R}^{d \times k}$



- Operation: map a point to closest center in Voronoi diagram
  - write $\mathbf{z} = \mathrm{VQ}(\mathbf{x}, V)$ where $\mathbf{z}$ is 1-hot and $V \in \mathbb{R}^{d \times k}$ is matrix of centers

Geoff Gordon

Vector quantization

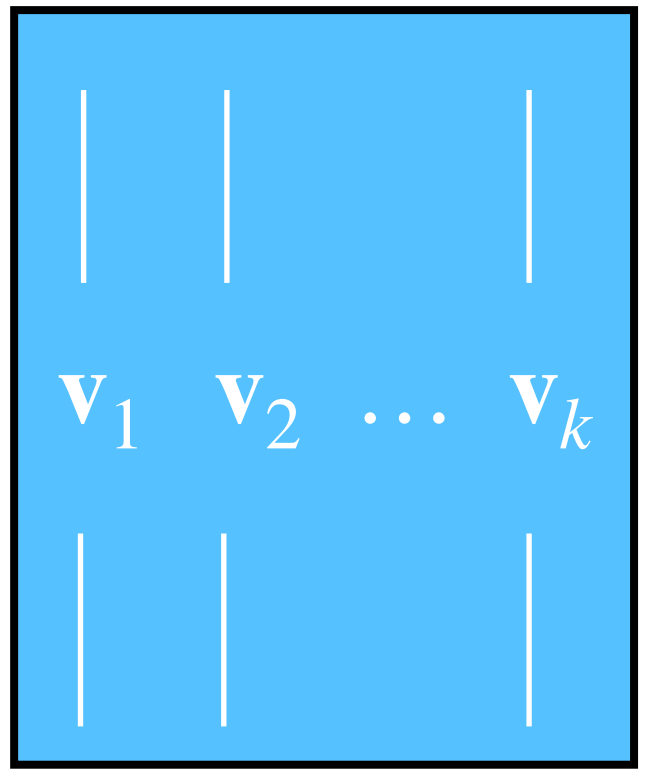


$$\mathbf{z} = \arg\min_{\mathbf{z}\in\{\mathbf{e}_1,\mathbf{e}_2,\ldots,\mathbf{e}_k\}} \|\mathbf{x} - V\mathbf{z}\|^2$$

$$\mathbf{z} = e_j \text{ where } j = \arg\min_{j\in\{1,2,\ldots,k\}} \|\mathbf{x} - \mathbf{v}_j\|^2$$

k centers in $\mathbb{R}^d$

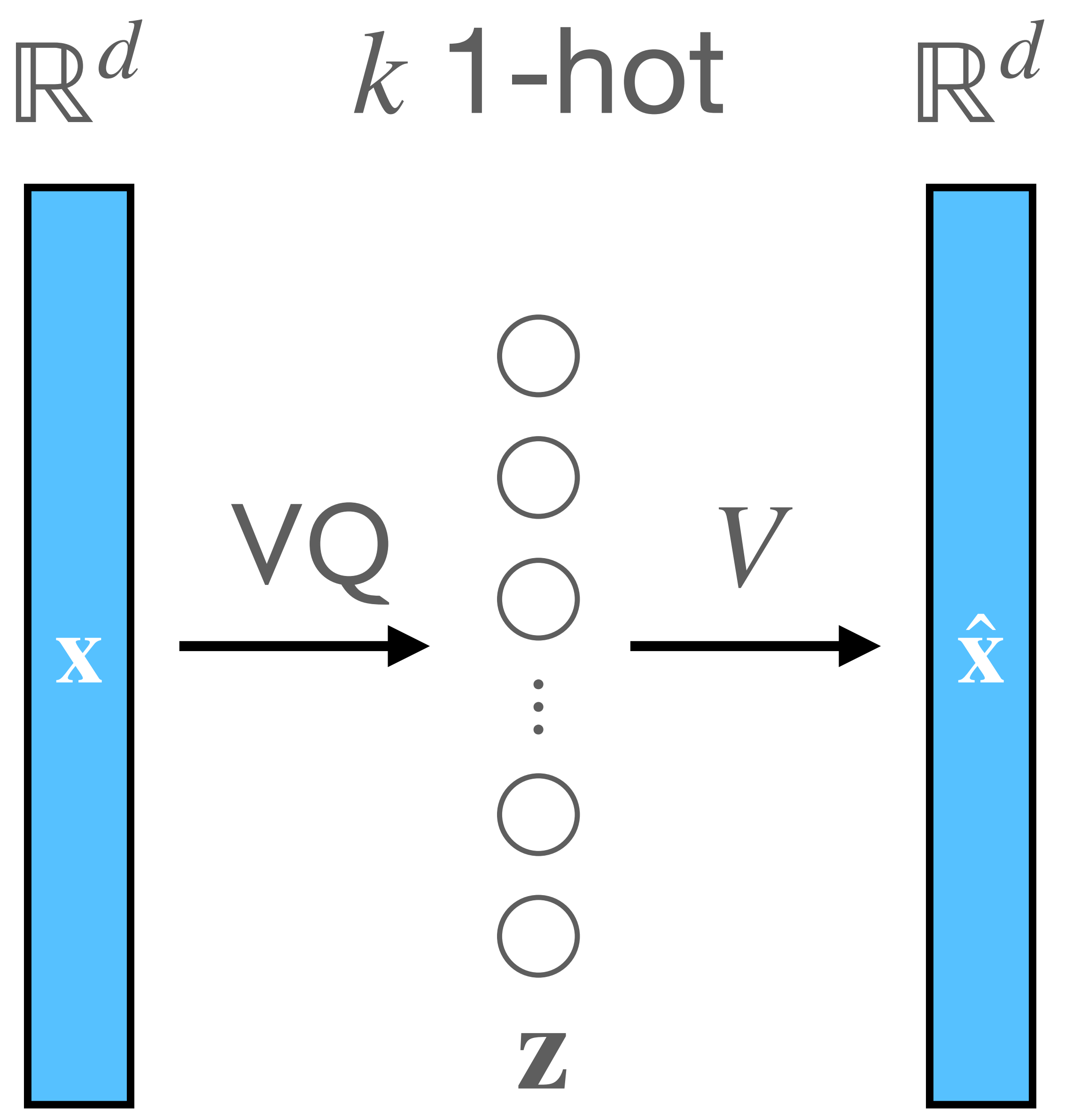
$V \in \mathbb{R}^{d\times k}$



- Operation: map a point to closest center in Voronoi diagram
  - write $\mathbf{z} = \mathrm{VQ}(\mathbf{x}, V)$ where $\mathbf{z}$ is 1-hot and $V \in \mathbb{R}^{d\times k}$ is matrix of centers

Geoff Gordon

Discrete autoencoder



$$\hat{\mathbf{x}} = V\mathbf{z} = V \times \text{VQ}(\mathbf{x}, V)$$

$$\hat{X} = ZV^\top$$

$$V \in \mathbb{R}^{d \times k}$$

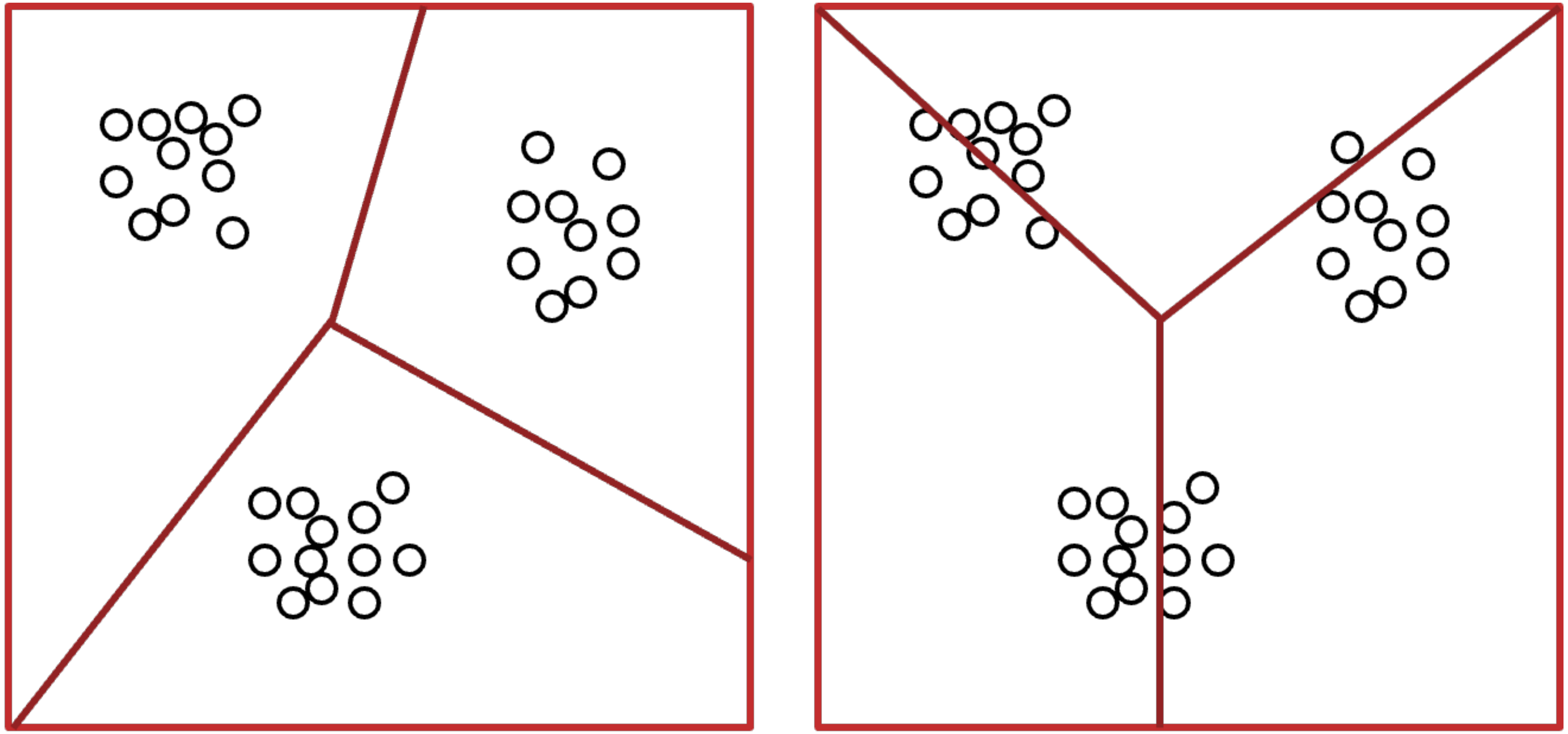
$$Z \in \{0,1\}^{N \times k}$$ rows 1-hot

- Discrete autoencoder (nonlinear):
  - map $\mathbf{x}$ to discrete hidden variable $\mathbf{z}$ using VQ
  - prediction is just the center that corresponds to $\mathbf{z}$
  - train to minimize MSE $\frac{1}{N}\sum_{i=1}^{N} ||\mathbf{x}^{(i)} - \hat{\mathbf{x}}^{(i)}||^2$

Clustering

- Discrete autoencoder implements ***clustering****:* learns to partition unlabeled data into groups of nearby points
  - this version called k-means
- Applications:
  - ***topic modeling:*** group news articles or web pages by topic
  - ***sequence analysis:*** group protein sequences by function or genes by expression profile
  - ***community detection:*** group social network users by interest
  - ***fraud detection:*** spot groups of unusual transactions
  - ***astronomy:*** find groups of similar objects in sky survey
  - …

Clustering: the picture



- Which of these partitions is better? Why?

k-means objective

- Reconstruction error: distance from $\mathbf{x}^{(i)}$ to closest center
  - closest center: $\arg\min_j \|\mathbf{x}^{(i)} - \mathbf{v}_j\|^2$
  - distance to closest center: $\min_j \|\mathbf{x}^{(i)} - \mathbf{v}_j\|^2$
  - or $\min_{\mathbf{z}} \|\mathbf{x}^{(i)} - V\mathbf{z}\|^2$ where $\mathbf{z} \in \{\mathbf{e}_1 \ldots \mathbf{e}_k\}$
- Find best V:
  - $\arg\min_V \sum_{i=1}^N \min_{\mathbf{z}} \|\mathbf{x}^{(i)} - V\mathbf{z}\|^2$
- Get rid of nested minimizations:
  - write $\mathbf{z}^{(i)}$ = latent for $\mathbf{x}^{(i)}$
  - $\arg\min_V \sum_{i=1}^N \min_{\mathbf{z}^{(i)}} \|\mathbf{x}^{(i)} - V\mathbf{z}^{(i)}\|^2$
  - $\arg\min_{V,Z} \sum_{i=1}^N \|\mathbf{x}^{(i)} - V\mathbf{z}^{(i)}\|^2$

$$\arg\min_{V,Z} \sum_{i=1}^{N} \|\mathbf{x}^{(i)} - V\mathbf{z}^{(i)}\|^2$$

k-means algorithm

- To optimize, use block coordinate descent
- Given feature vectors $\mathscr{D} = \{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(N)}\}$
- Initialize matrix of centers V (each column is a center $\mathbf{v}_j$)
- Repeat:
  - minimize wrt Z: for each i, set $\mathbf{z}^{(i)}$ to map $\mathbf{x}^{(i)}$ to its closest center
  - minimize wrt V: for each j, minimize MSE from $\mathbf{v}_j$ to its assigned points

Geoff Gordon

$$\arg\min_{V,Z} \sum_{i=1}^{N} \|\mathbf{x}^{(i)} - V\mathbf{z}^{(i)}\|^2$$

k-means algorithm

- To optimize, use block coordinate descent
- Given feature vectors $\mathscr{D} = \{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(N)}\}$
- Initialize matrix of centers V (each column is a center $\mathbf{v}_j$)
- Repeat:
  - minimize wrt Z: for each i, set $\mathbf{z}^{(i)}$ to map $\mathbf{x}^{(i)}$ to its closest center
  - minimize wrt V: for each j, minimize MSE from $\mathbf{v}_j$ to its assigned points

Can solve each i and j independently

Geoff Gordon

$$\arg\min_{V,Z} \sum_{i=1}^{N} \|\mathbf{x}^{(i)} - V\mathbf{z}^{(i)}\|^2$$

k-means algorithm

- To optimize, use block coordinate descent
- Given feature vectors $\mathscr{D} = \{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(N)}\}$
- Initialize matrix of centers V (each column is a center $\mathbf{v}_j$)
- Repeat:
  - minimize wrt Z: for each i, set $\mathbf{z}^{(i)}$ to map $\mathbf{x}^{(i)}$ to its closest center
  - minimize wrt V: for each j, minimize MSE from $\mathbf{v}_j$ to its assigned points

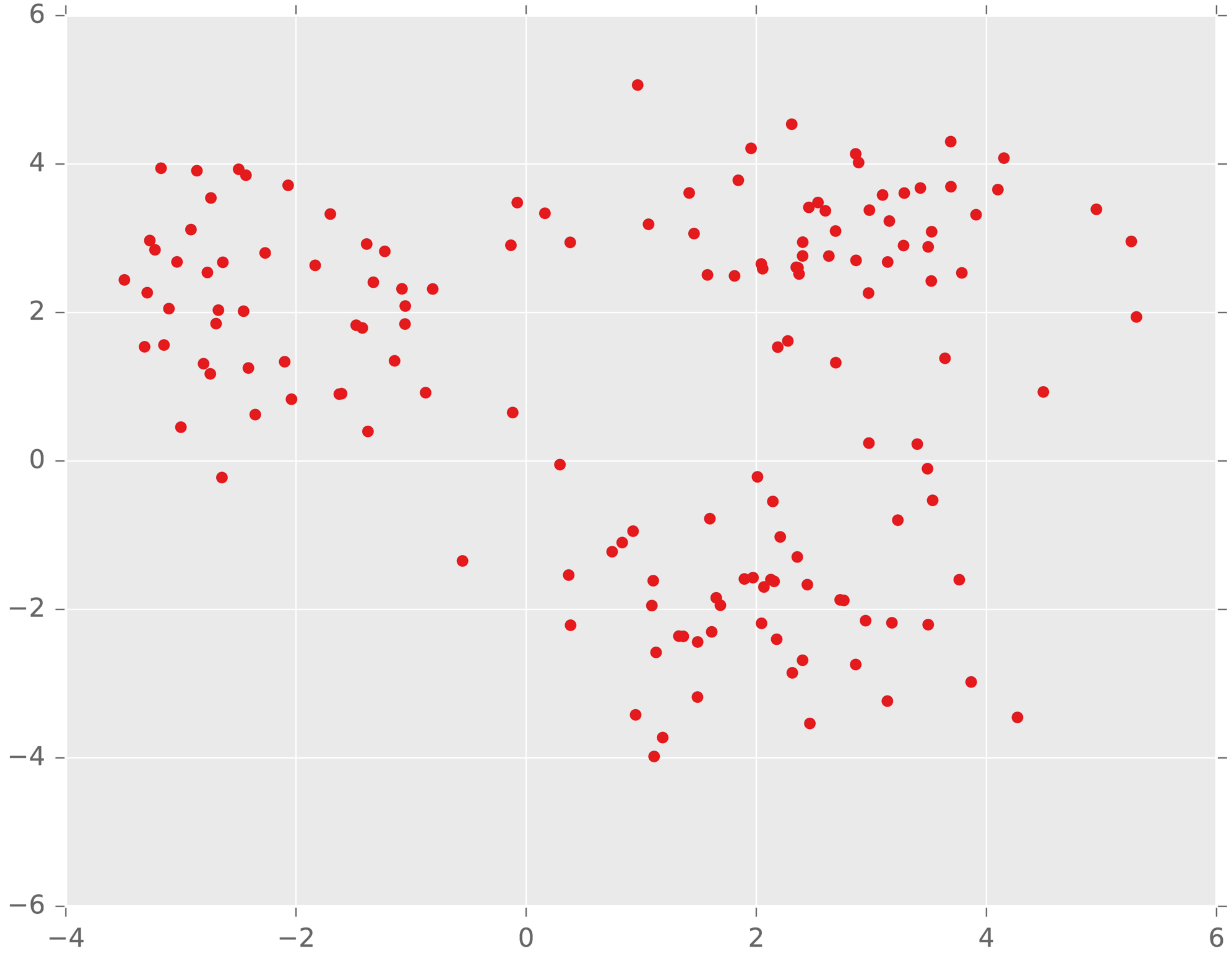
Can solve each i and j independently

this is the same as setting $\mathbf{v}_j$ = mean of $\{\mathbf{x}^{(i)} \mid \mathbf{z}^{(i)} = \mathbf{e}_j\}$

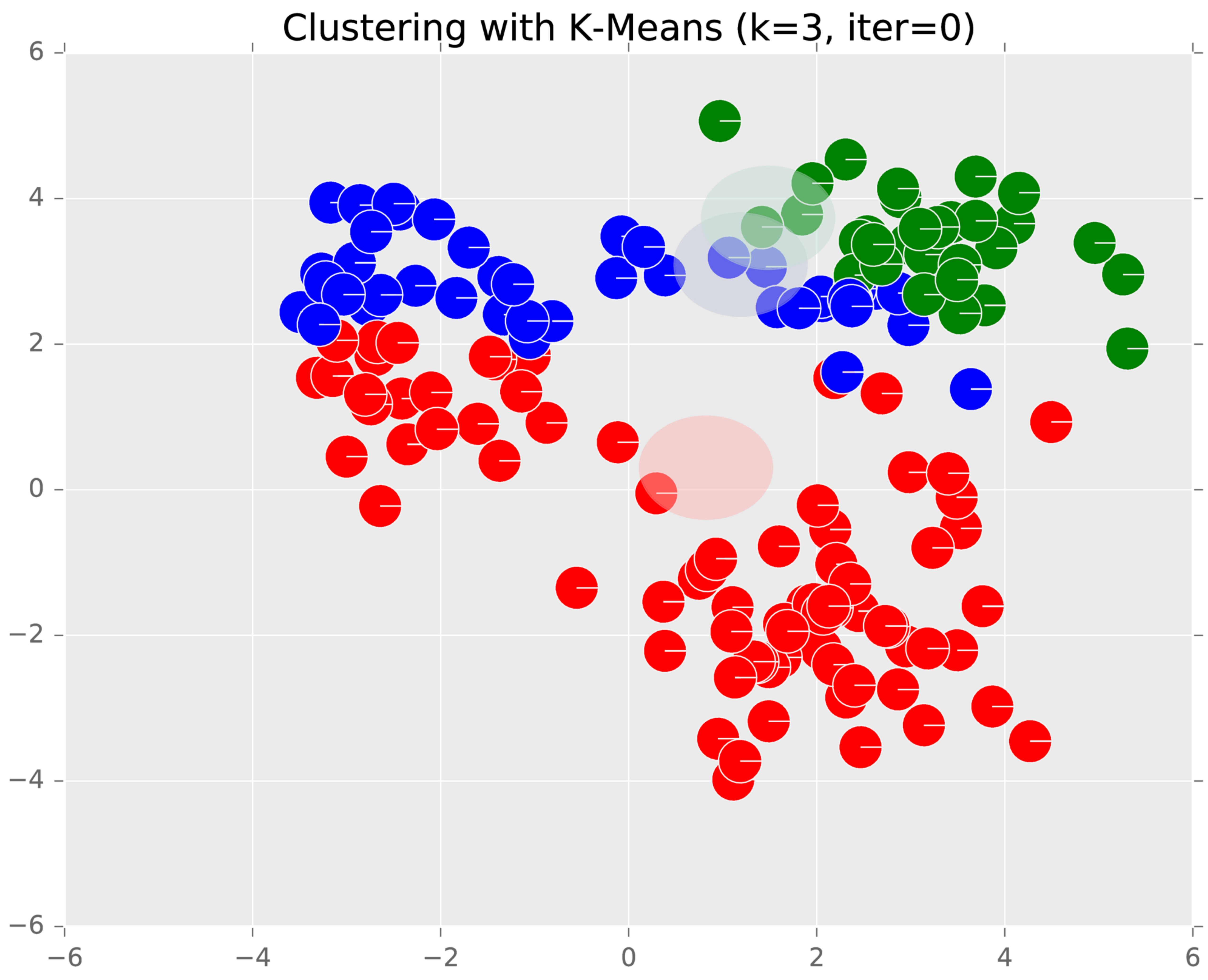
Example: K-Means

True k is 3

Example: K-Means

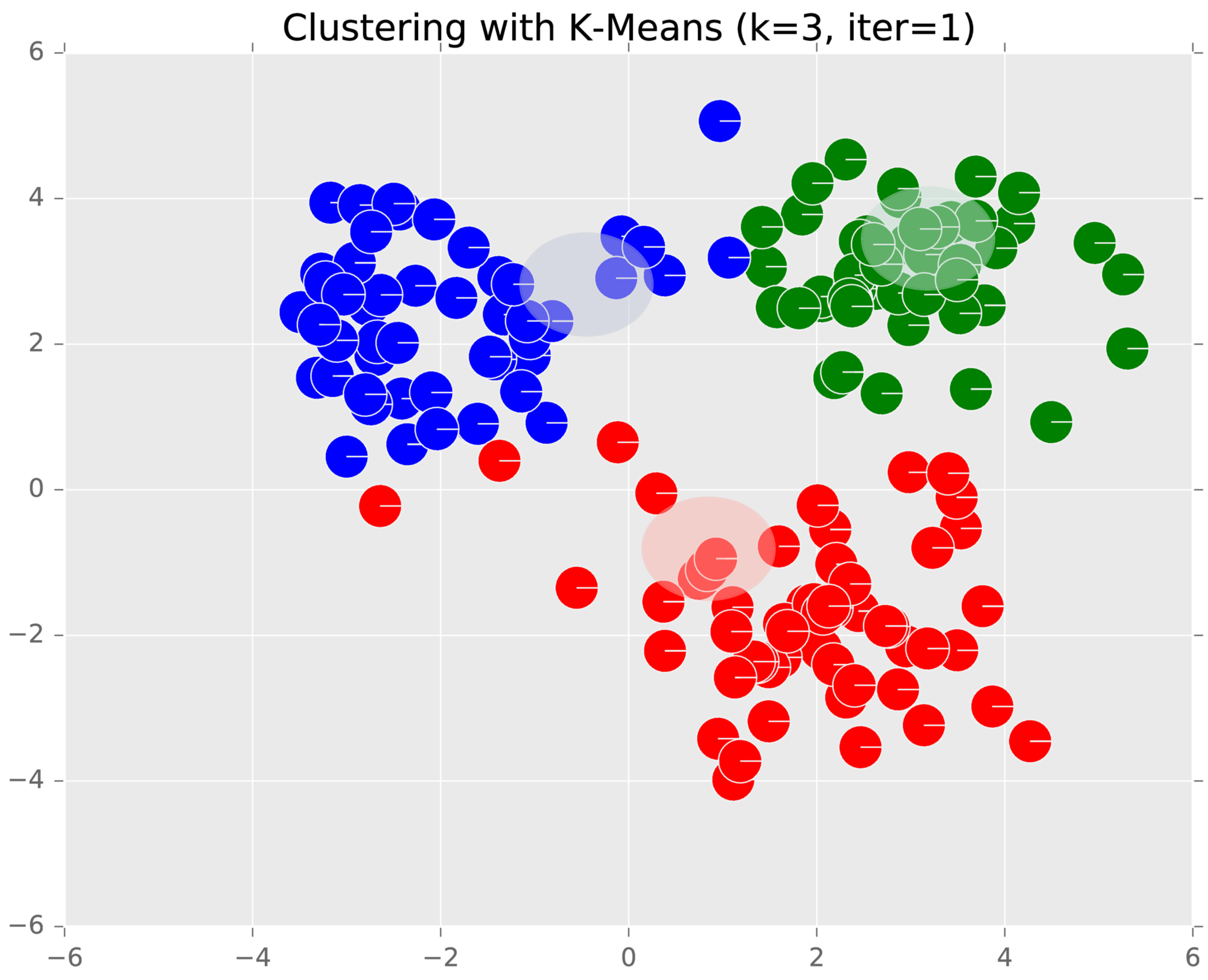


Example: K-Means

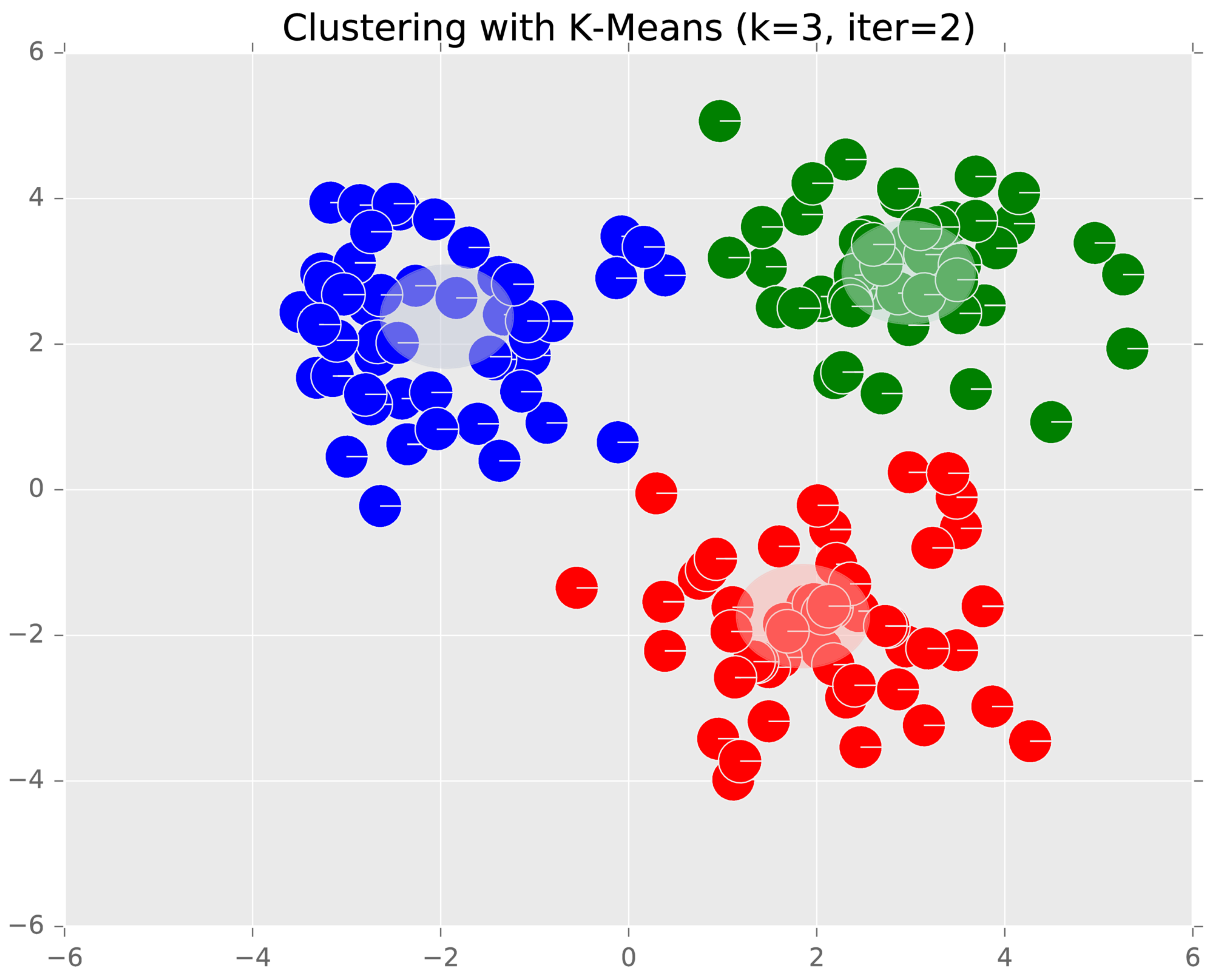


Use $k = 3$

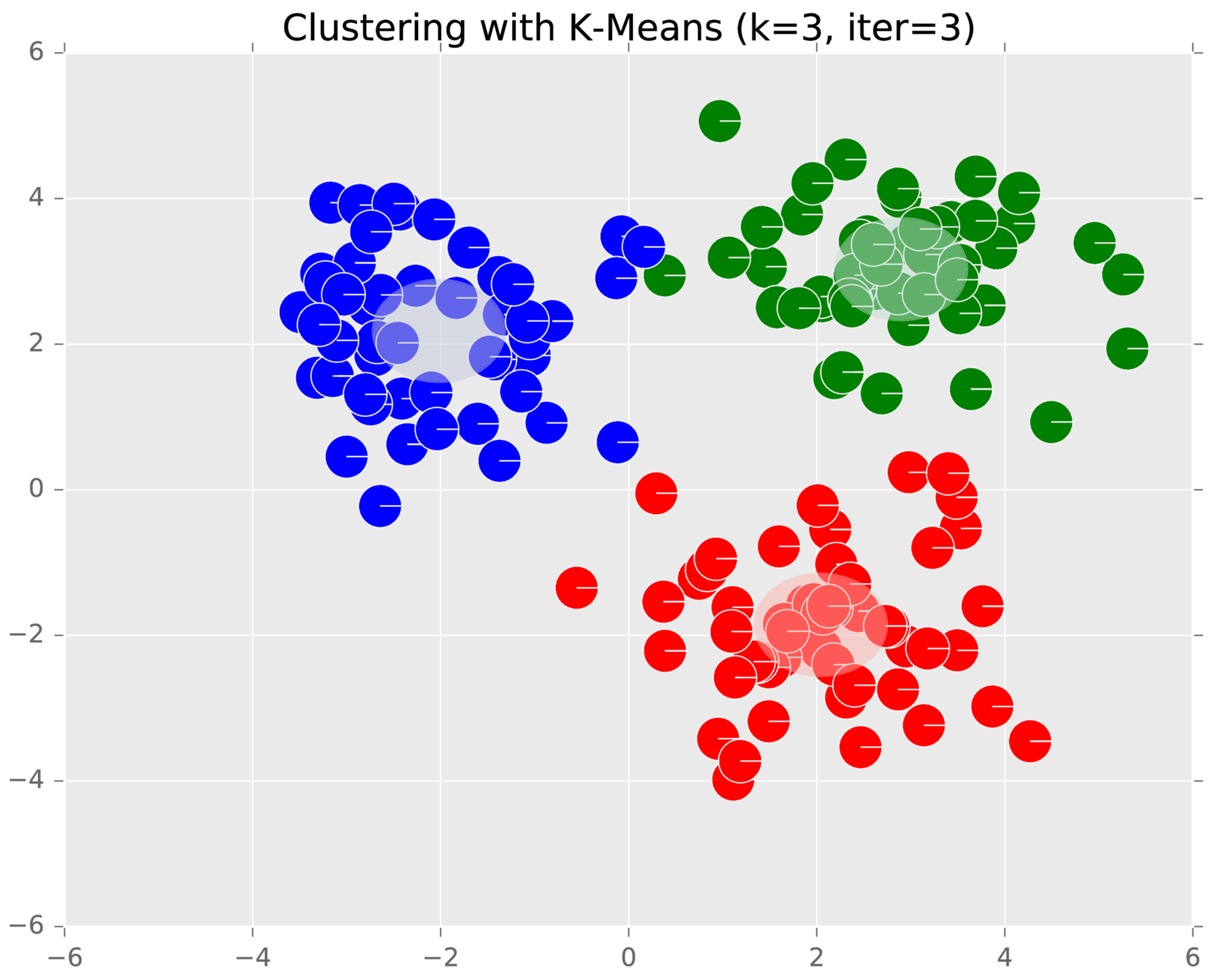
Example: K-Means



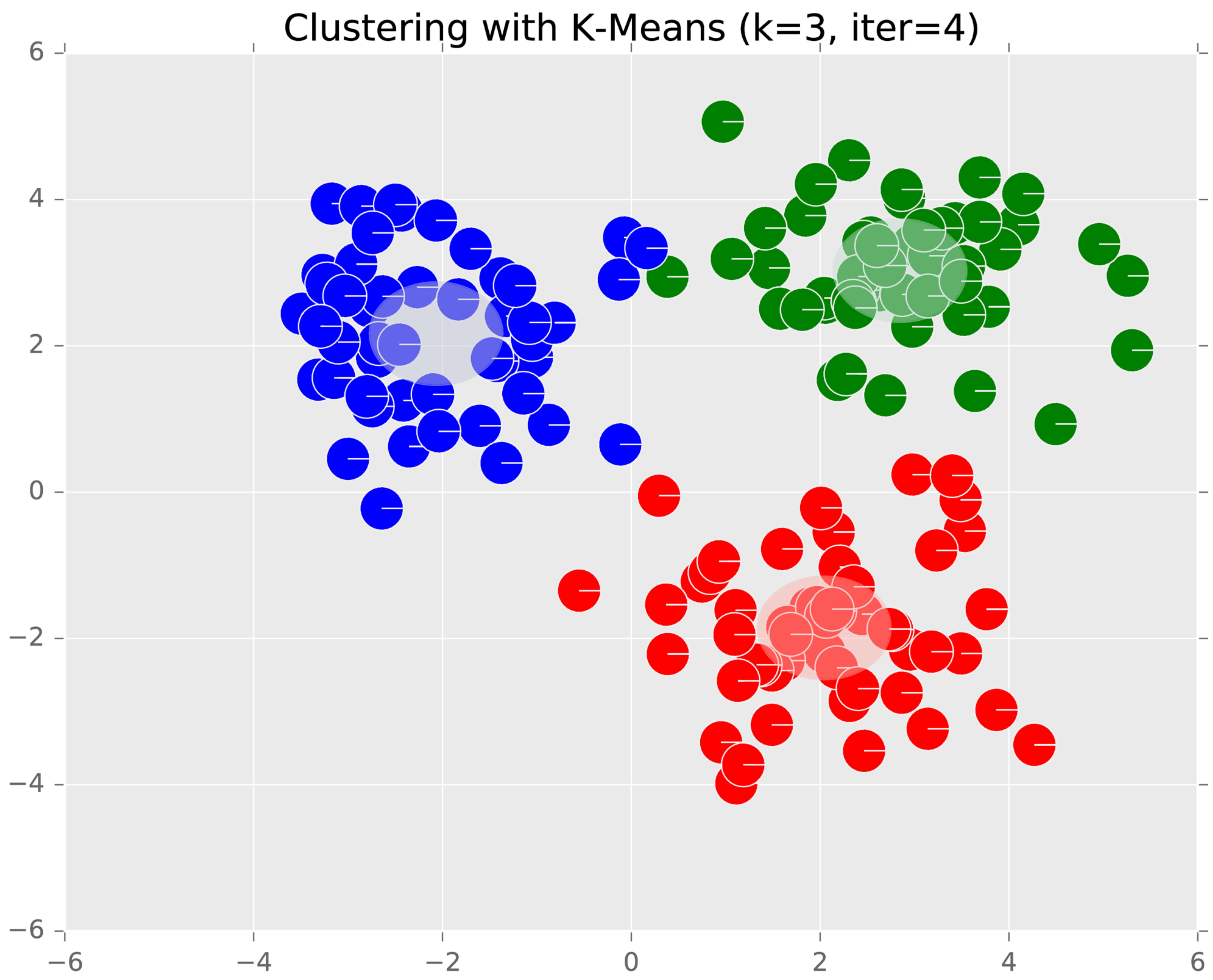
Example: K-Means



Example: K-Means



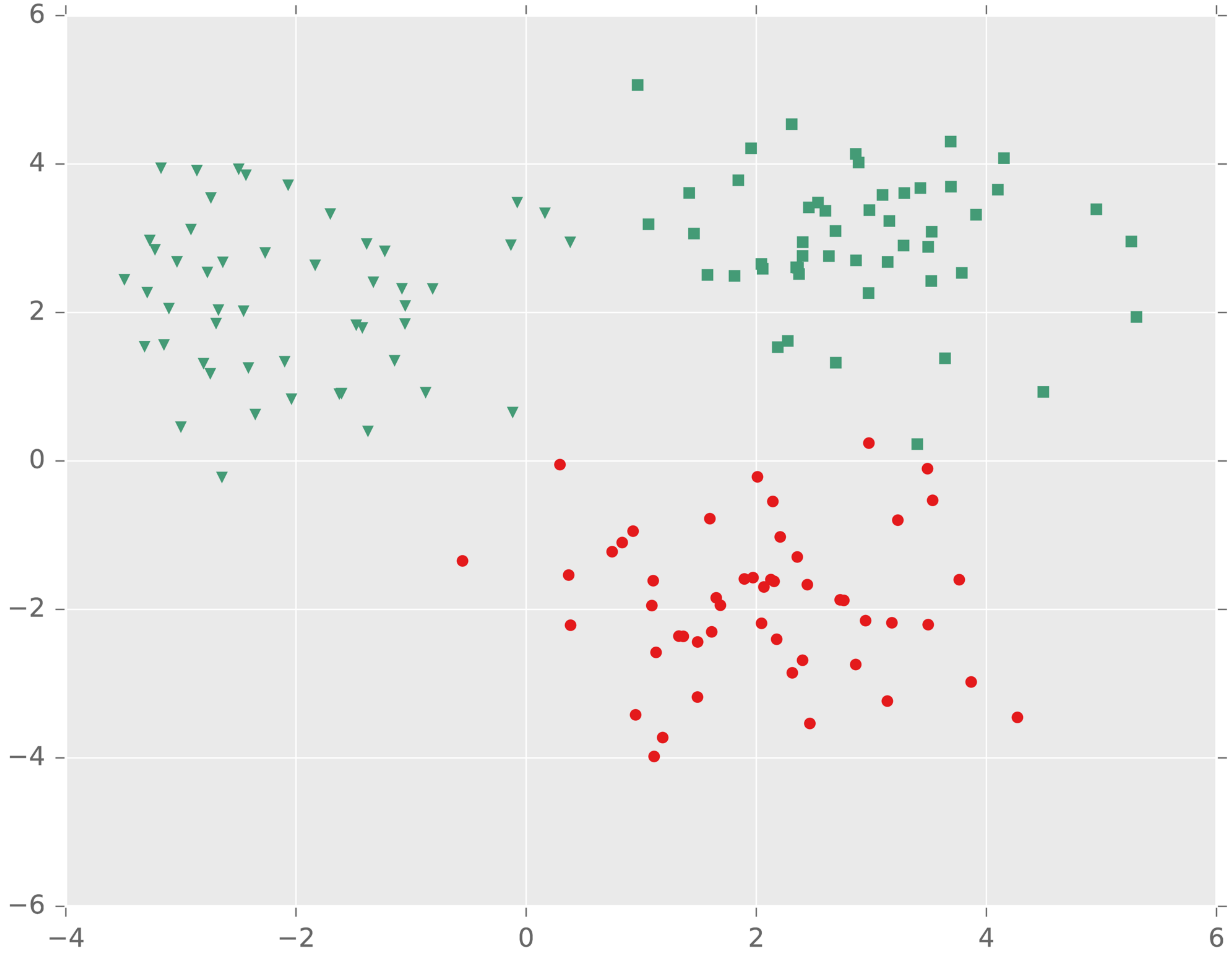
Example: K-Means



Example: K-Means

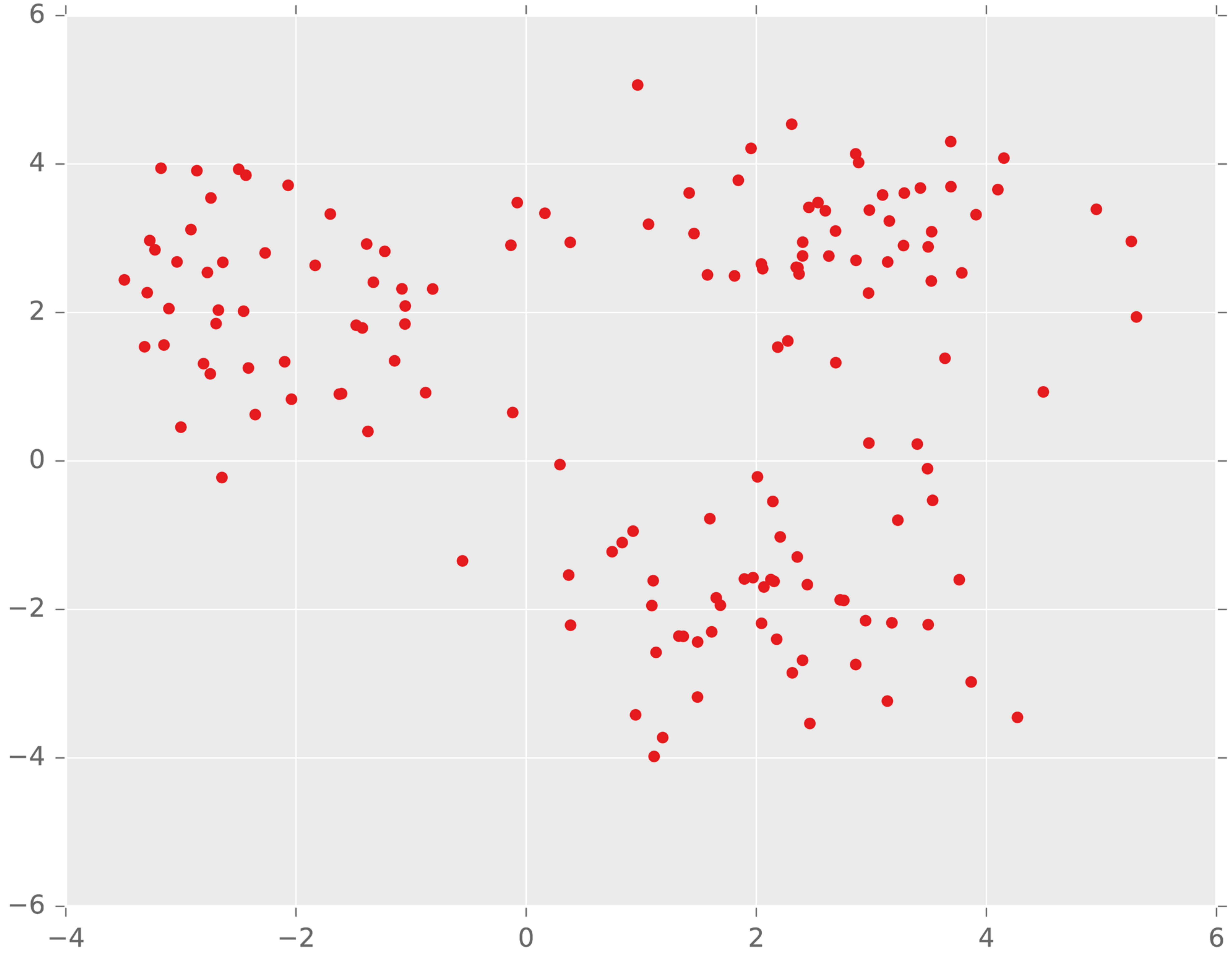
converged

Example: K-Means

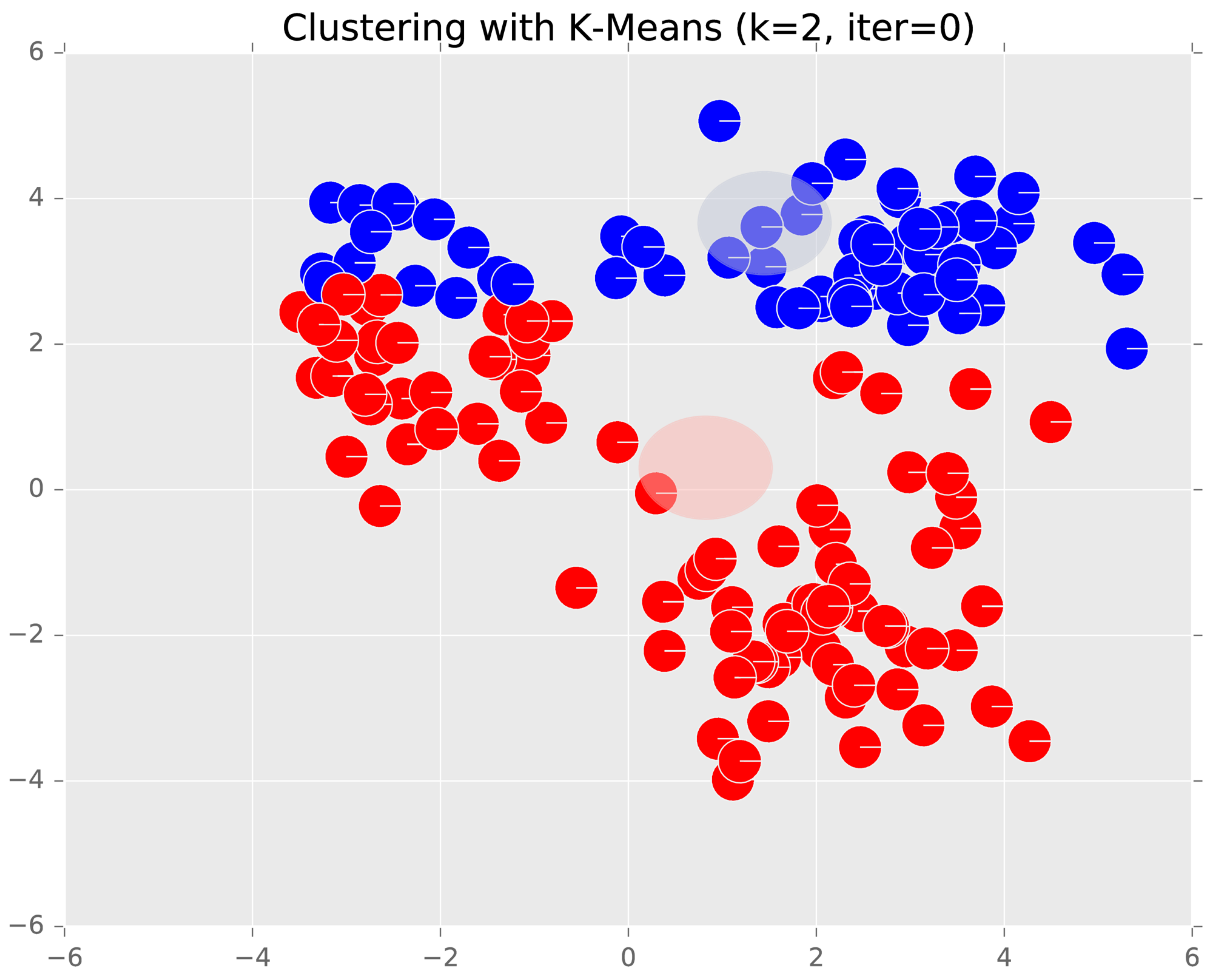


True k is 3

Example: K-Means

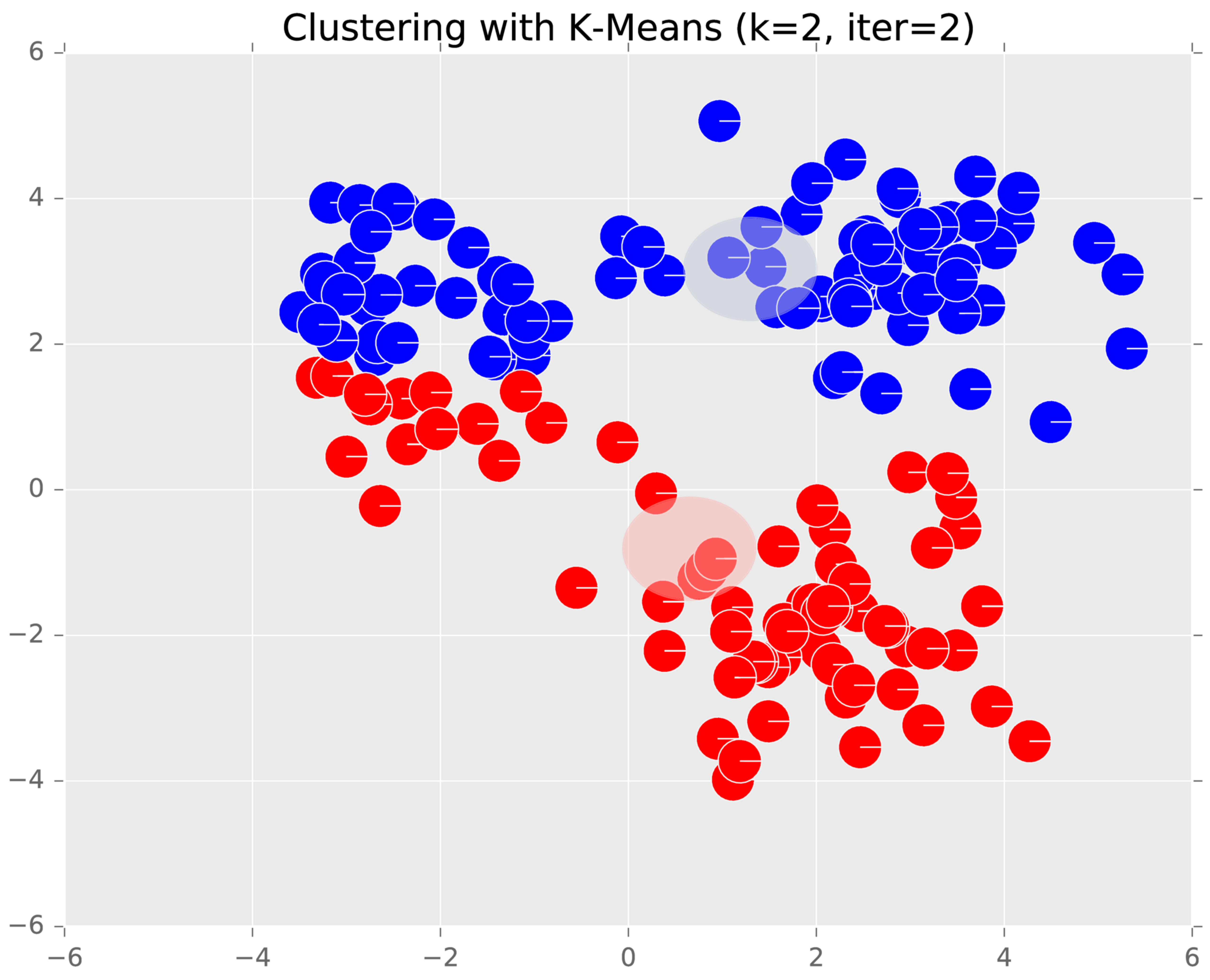


Example: K-Means

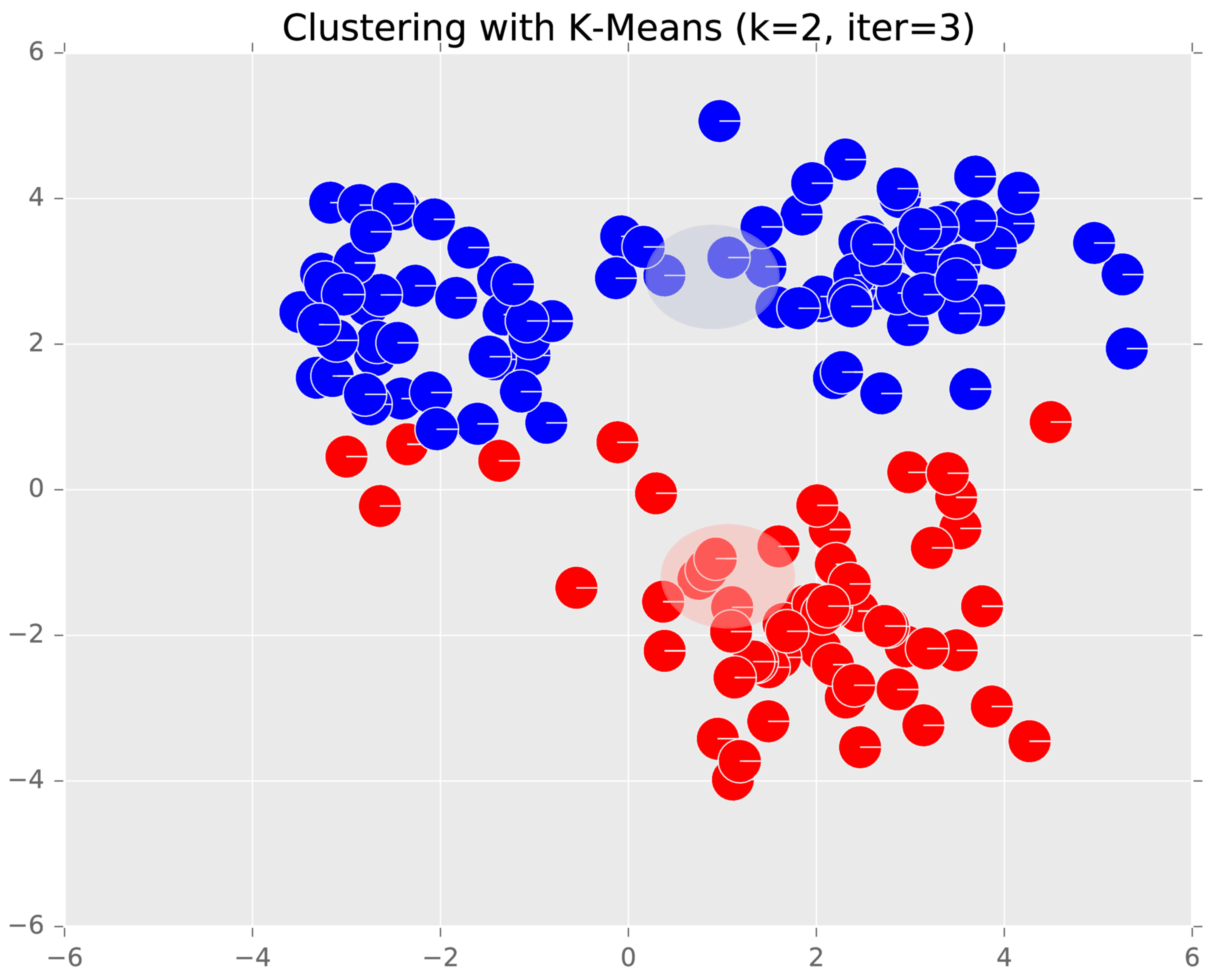


Use $k = 2$

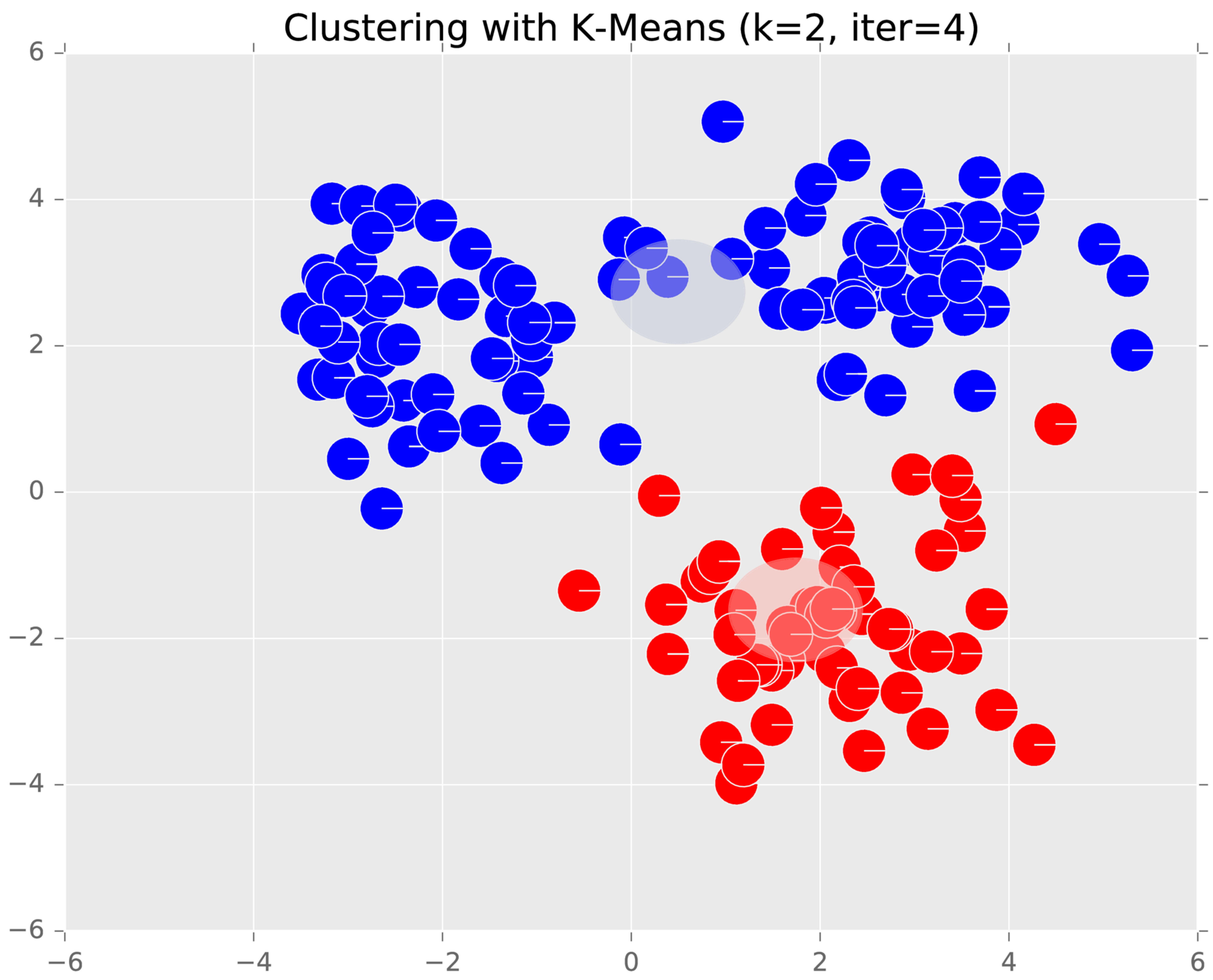
Example: K-Means



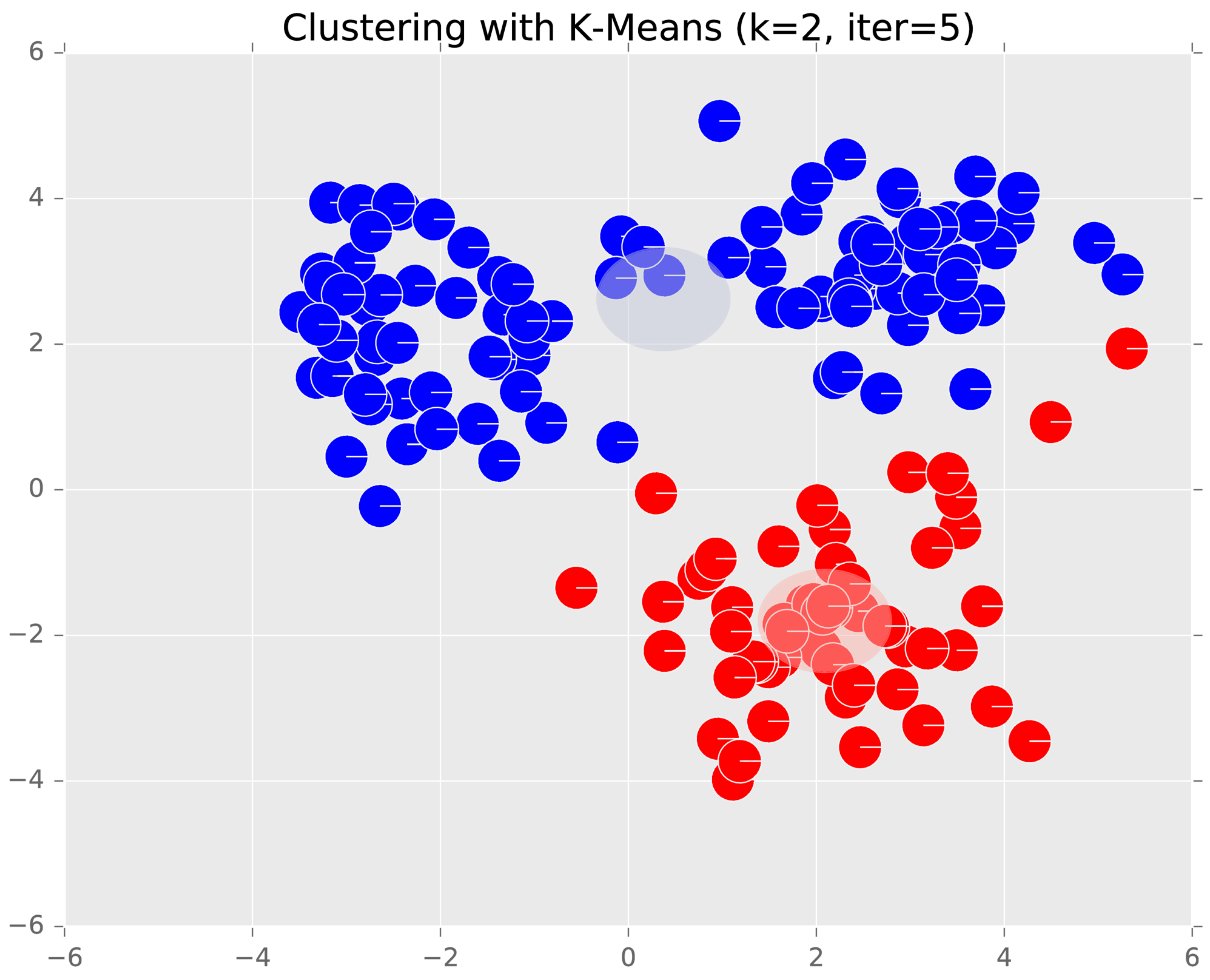
Example: K-Means



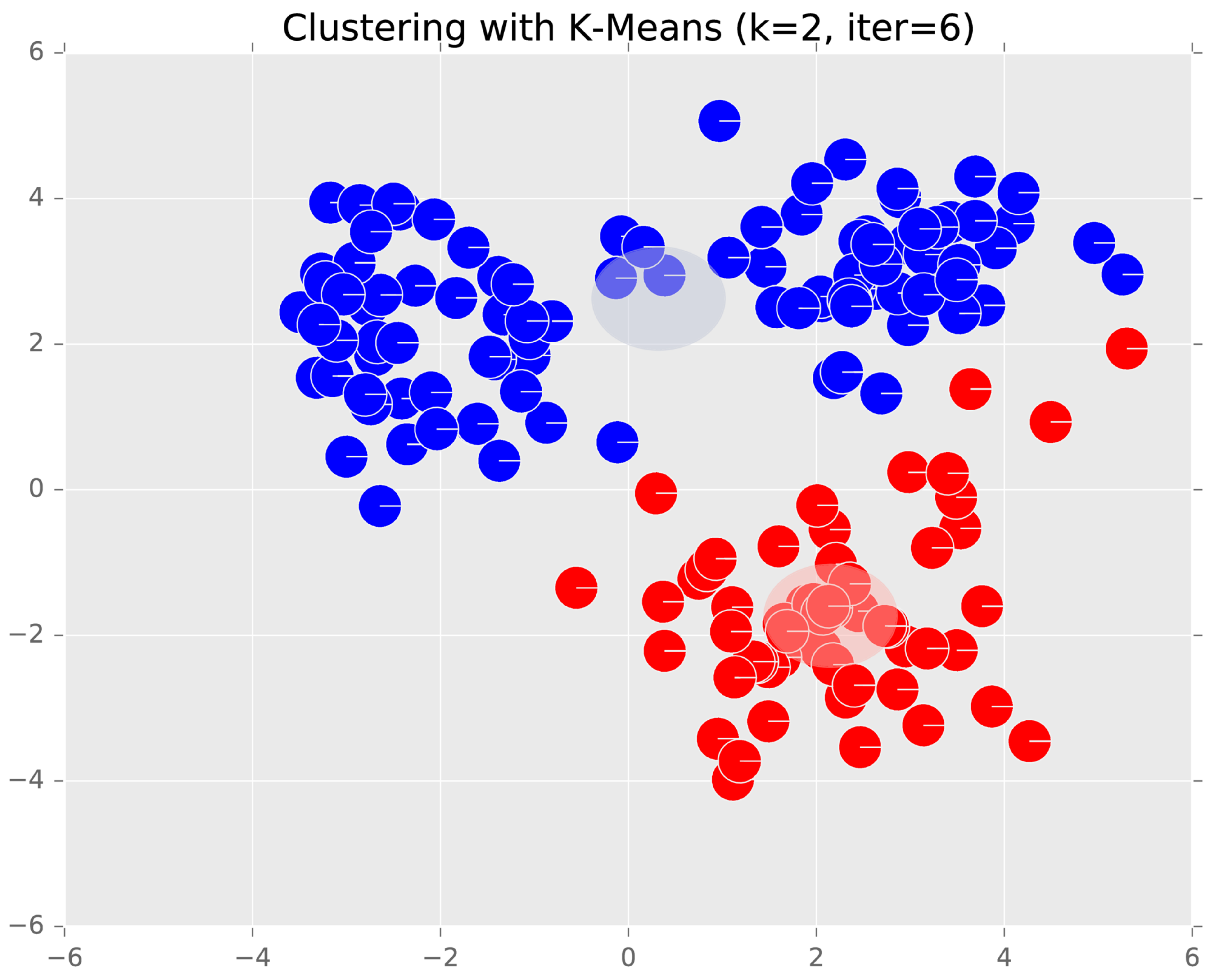
Example: K-Means



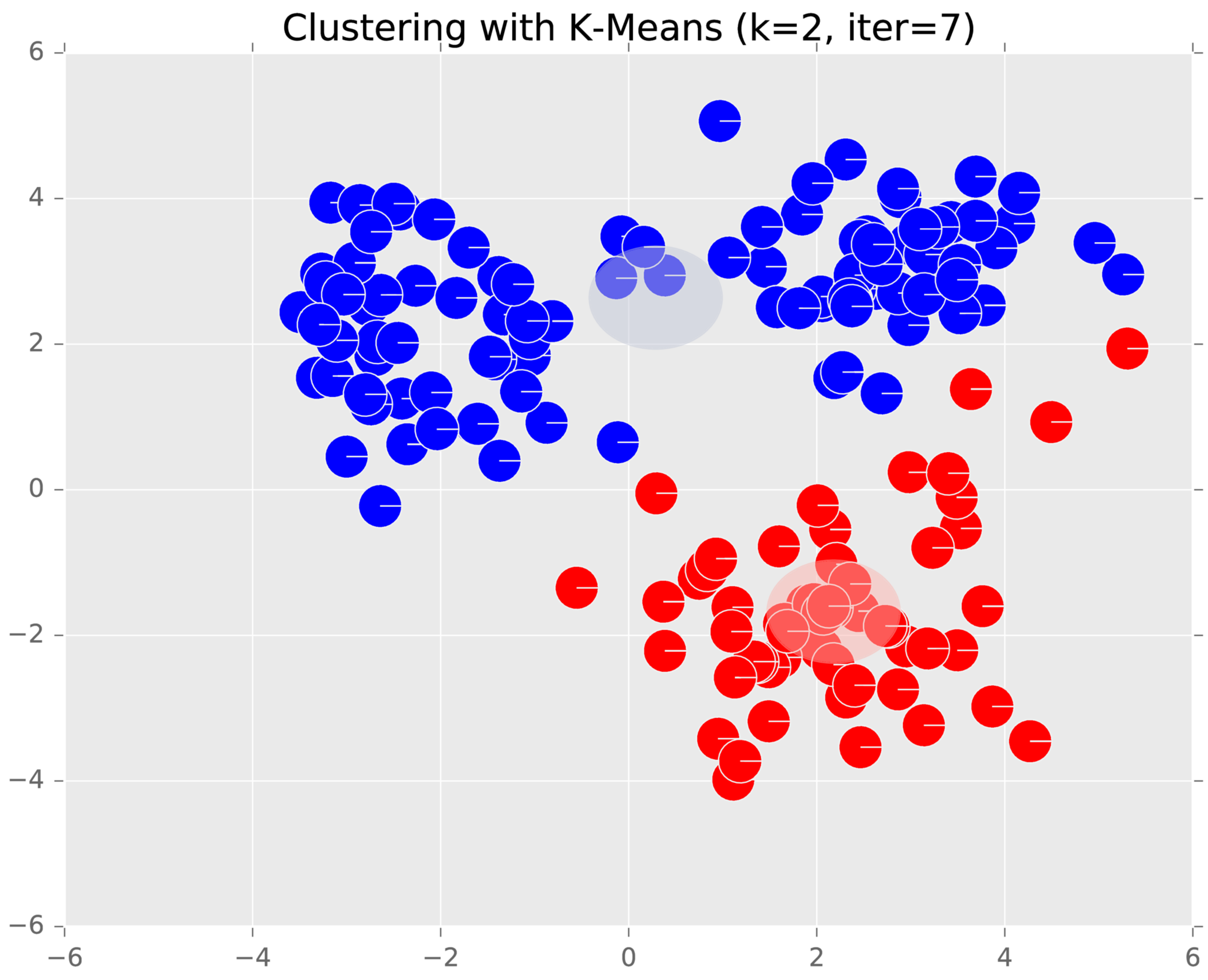
Example: K-Means



Example: K-Means



Example: K-Means



converged

Initializing k-means

- Given feature vectors $\mathscr{D} = \{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(N)}\}$
- Initialize matrix of centers V (each column is a center $\mathbf{v}_j$)
- Repeat:
  - minimize wrt Z: for each i, set $\mathbf{z}^{(i)}$ to assign $\mathbf{x}^{(i)}$ to its closest center
  - minimize wrt V: for each j, minimize MSE from $\mathbf{v}_j$ to its assigned points

Geoff Gordon

Initializing k-means

- Given feature vectors $\mathscr{D} = \{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(N)}\}$
- Initialize matrix of centers V (each column is a center $\mathbf{v}_j$)
- Repeat:
  - minimize wrt Z: for eac
    closest center
  - minimize wrt V: for eac
    assigned points

Remaining question: how should we initialize cluster centers?

We'll try three solutions: (1) at random, (2) furthest point heuristic, (3) k-means++

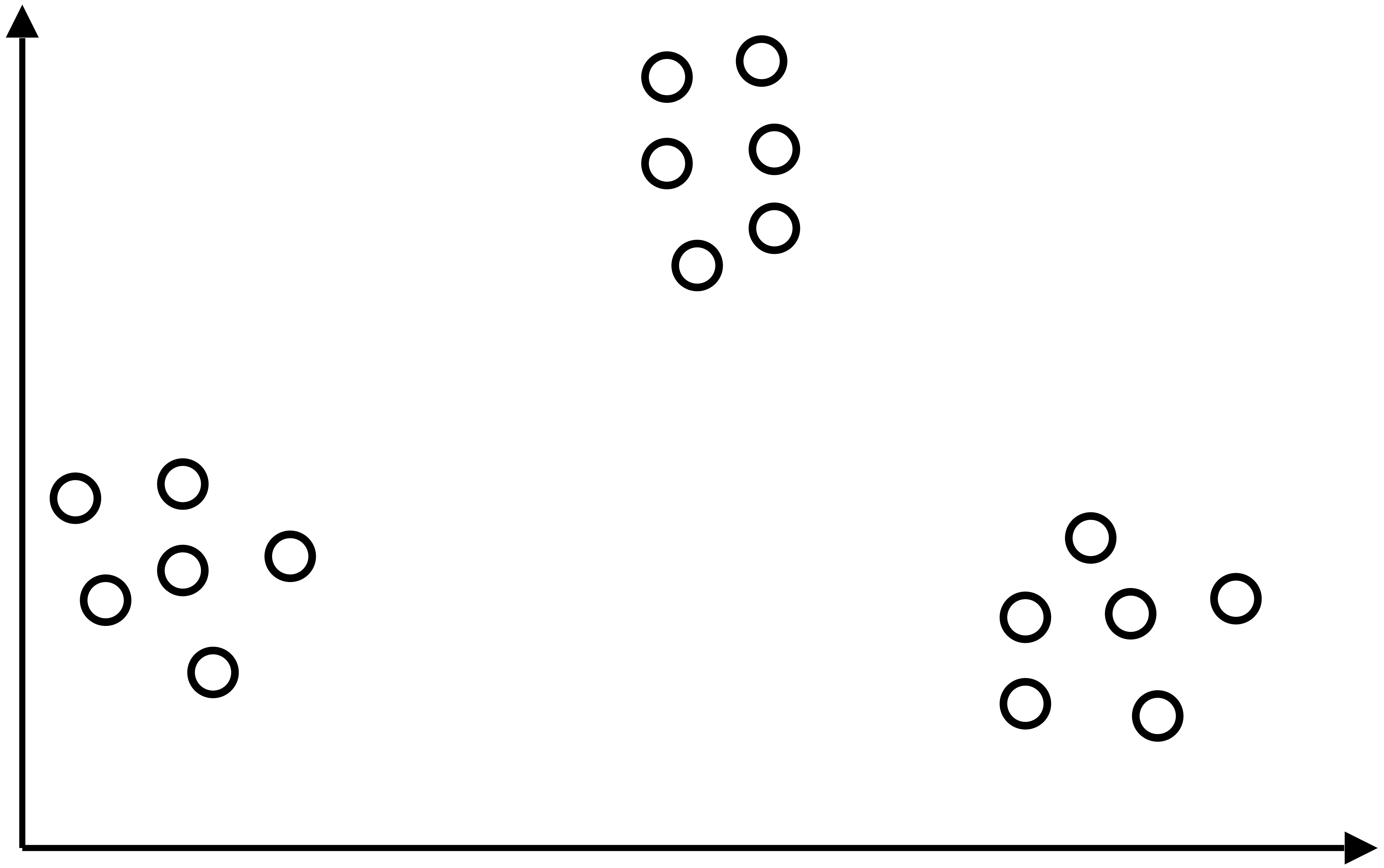
Geoff Gordon

Initialization for K-Means

Algorithm #1: Random Initialization
Select each cluster center uniformly at random from the data points in the training data

Observations:
Even when data comes from well-separated Gaussians…

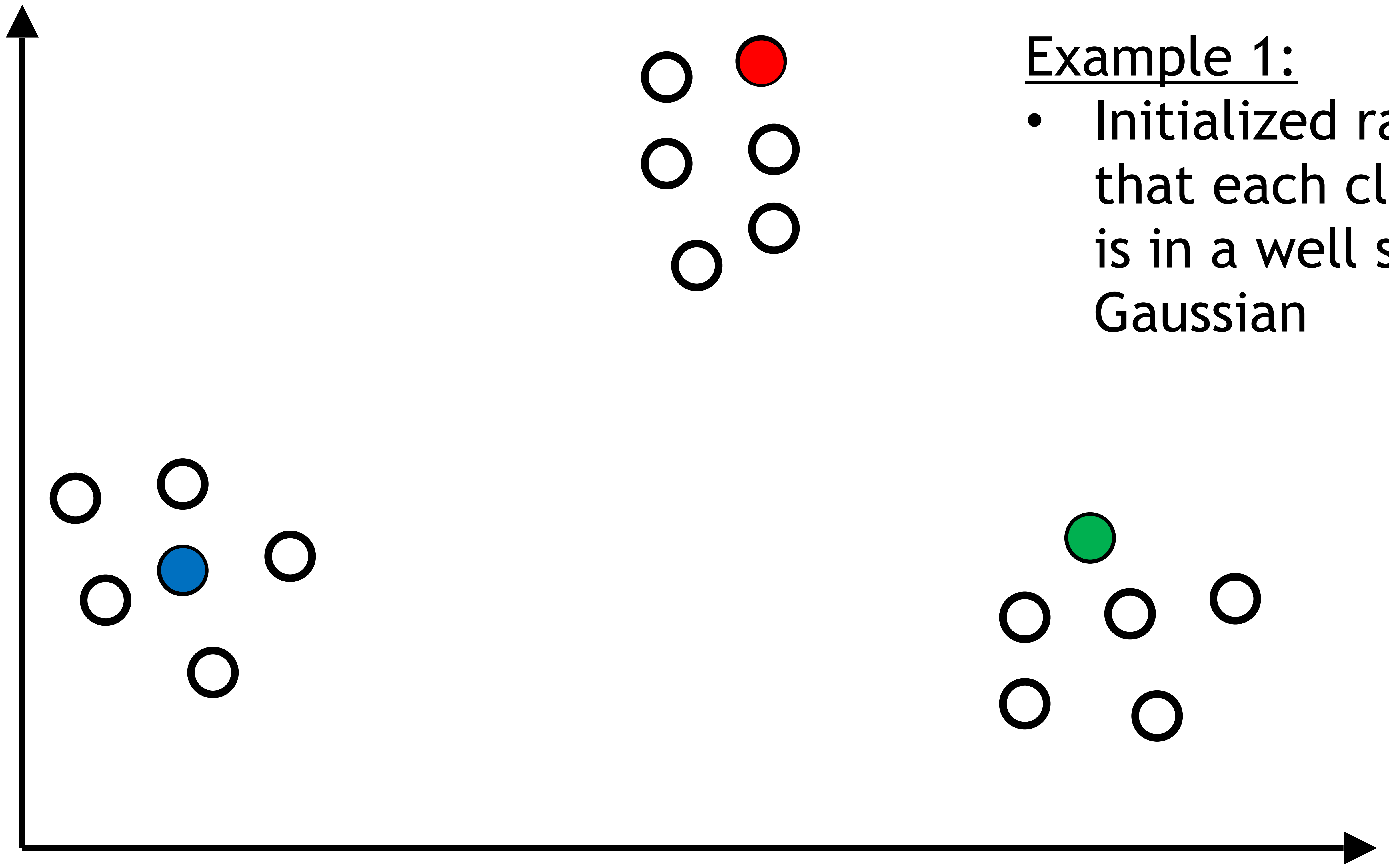
- …sometimes works great!
- …sometimes get stuck in poor local optima.



Initialization for K-Means

Algorithm #1: Random Initialization
Select each cluster center uniformly at random from the data points in the training data

Observations:
Even when data comes from well-separated Gaussians…
- …sometimes works great!
- …sometimes get stuck in poor local optima.

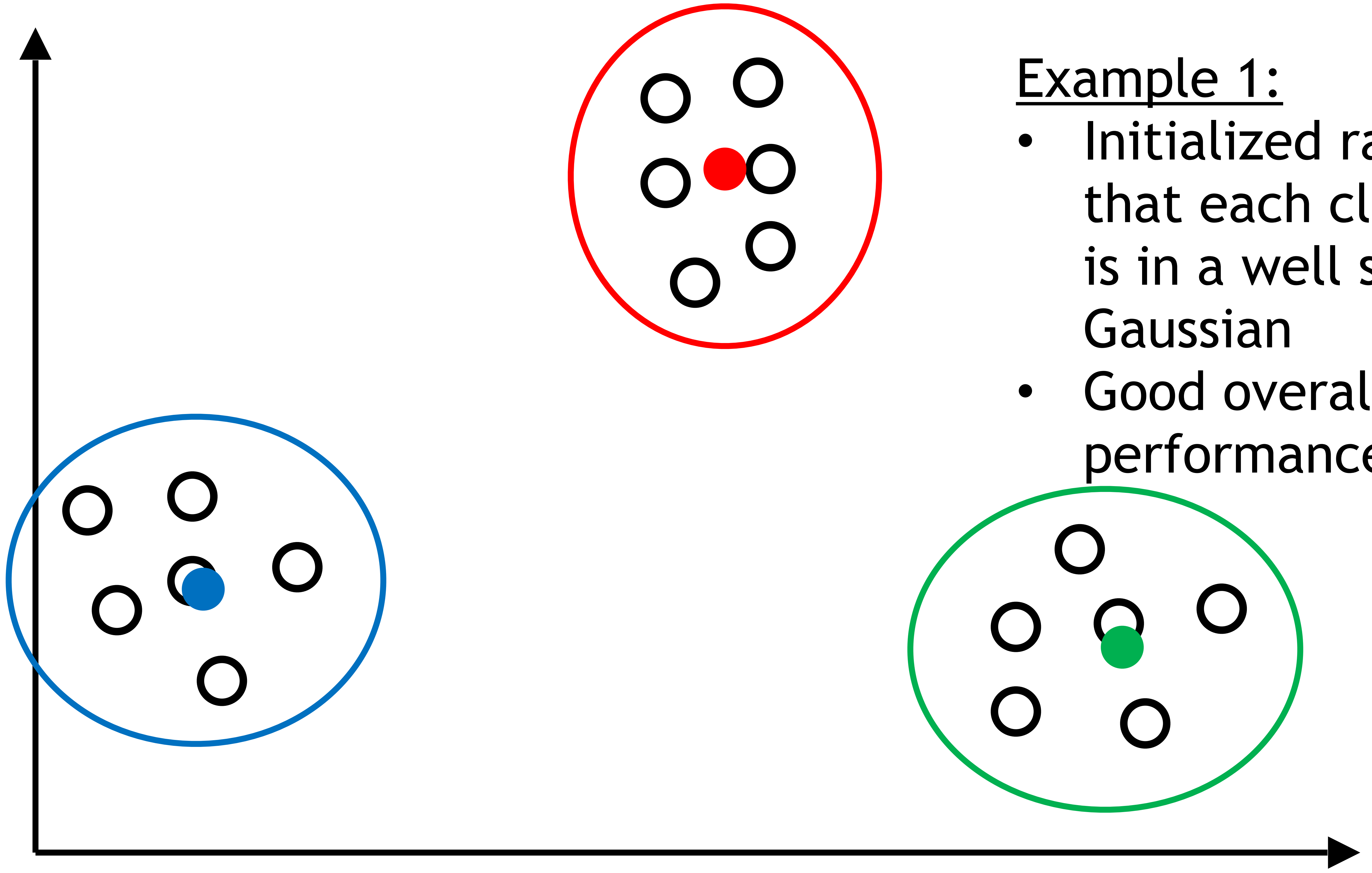


Example 1:
- Initialized randomly such that each cluster center is in a well separated Gaussian

Initialization for K-Means

<u>Algorithm #1: Random Initialization</u>
Select each cluster center uniformly at random from the data points in the training data

<u>Observations:</u>
Even when data comes from well-separated Gaussians…
- …sometimes works great!
- …sometimes get stuck in poor local optima.

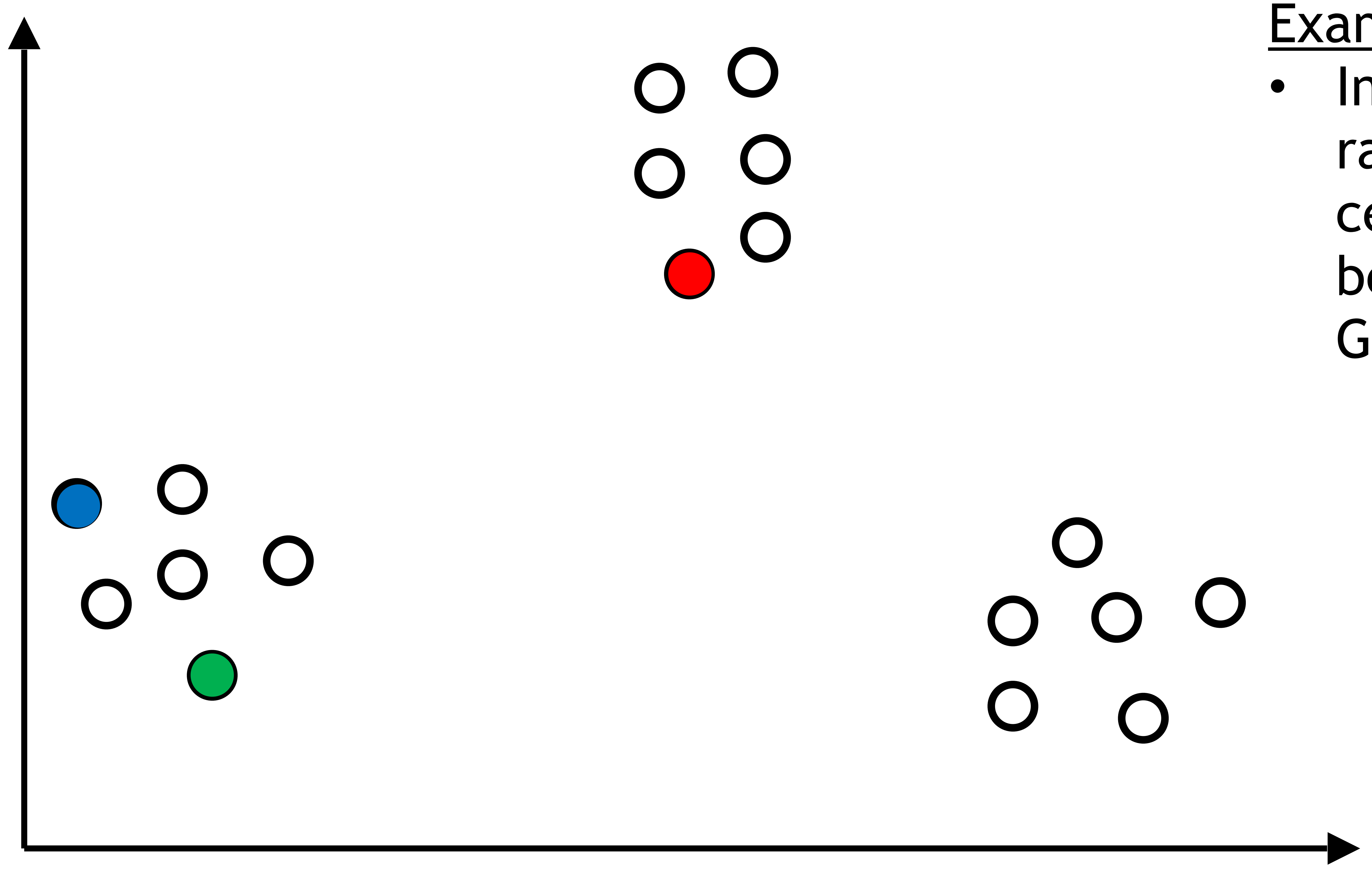


<u>Example 1:</u>
- Initialized randomly such that each cluster center is in a well separated Gaussian
- Good overall performance

Initialization for K-Means

<u>Algorithm #1: Random Initialization</u>
Select each cluster center uniformly at random from the data points in the training data

<u>Observations:</u>
Even when data comes from well-separated Gaussians…
- …sometimes works great!
- …sometimes get stuck in poor local optima.

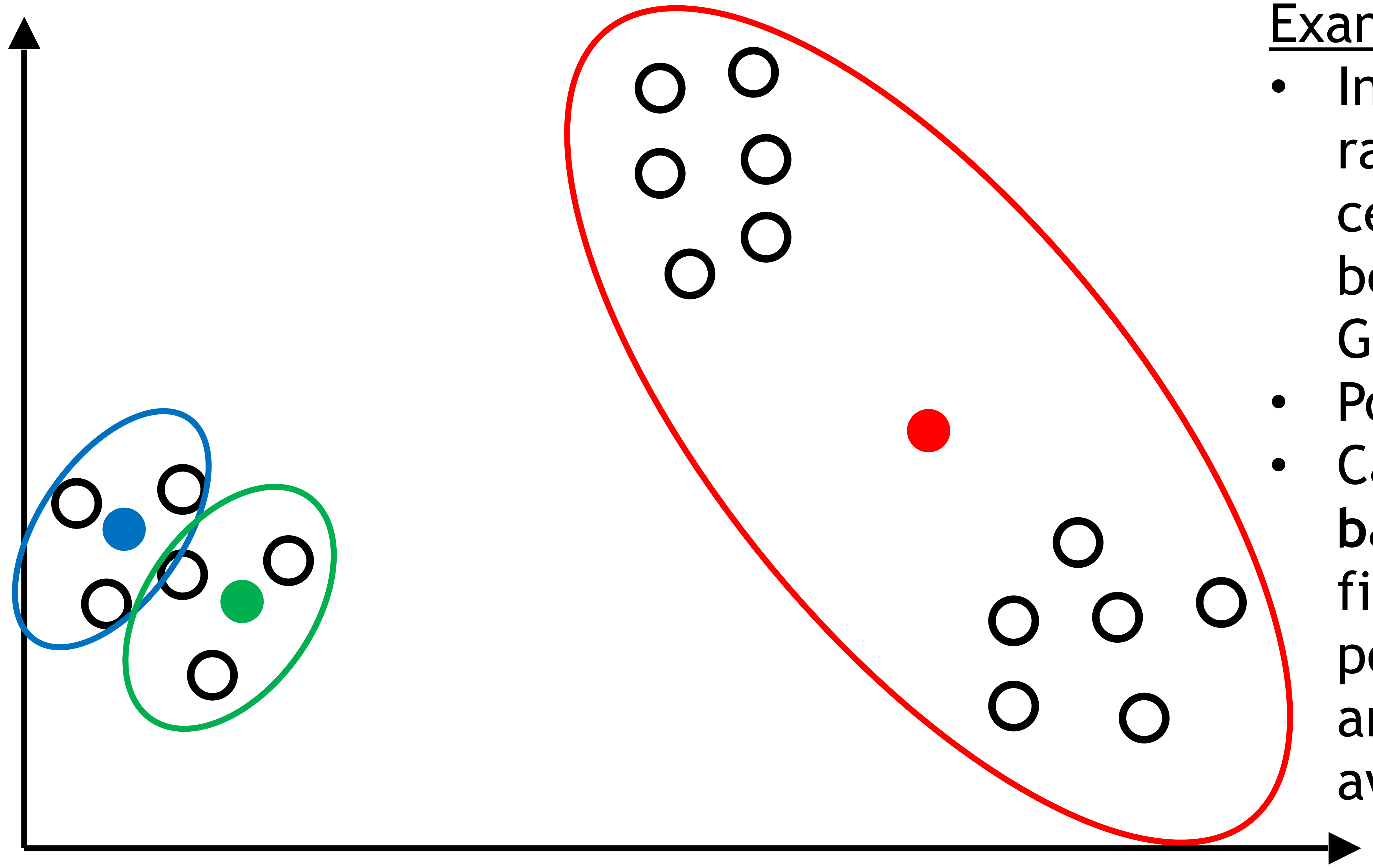


<u>Example 2:</u>
- Initialized randomly but two centers happen to be in same Gaussian cluster

Initialization for K-Means

<u>Algorithm #1: Random Initialization</u>
Select each cluster center uniformly at random from the data points in the training data

<u>Observations:</u>
Even when data comes from well-separated Gaussians…
- …sometimes works great!
- …sometimes get stuck in poor local optima.



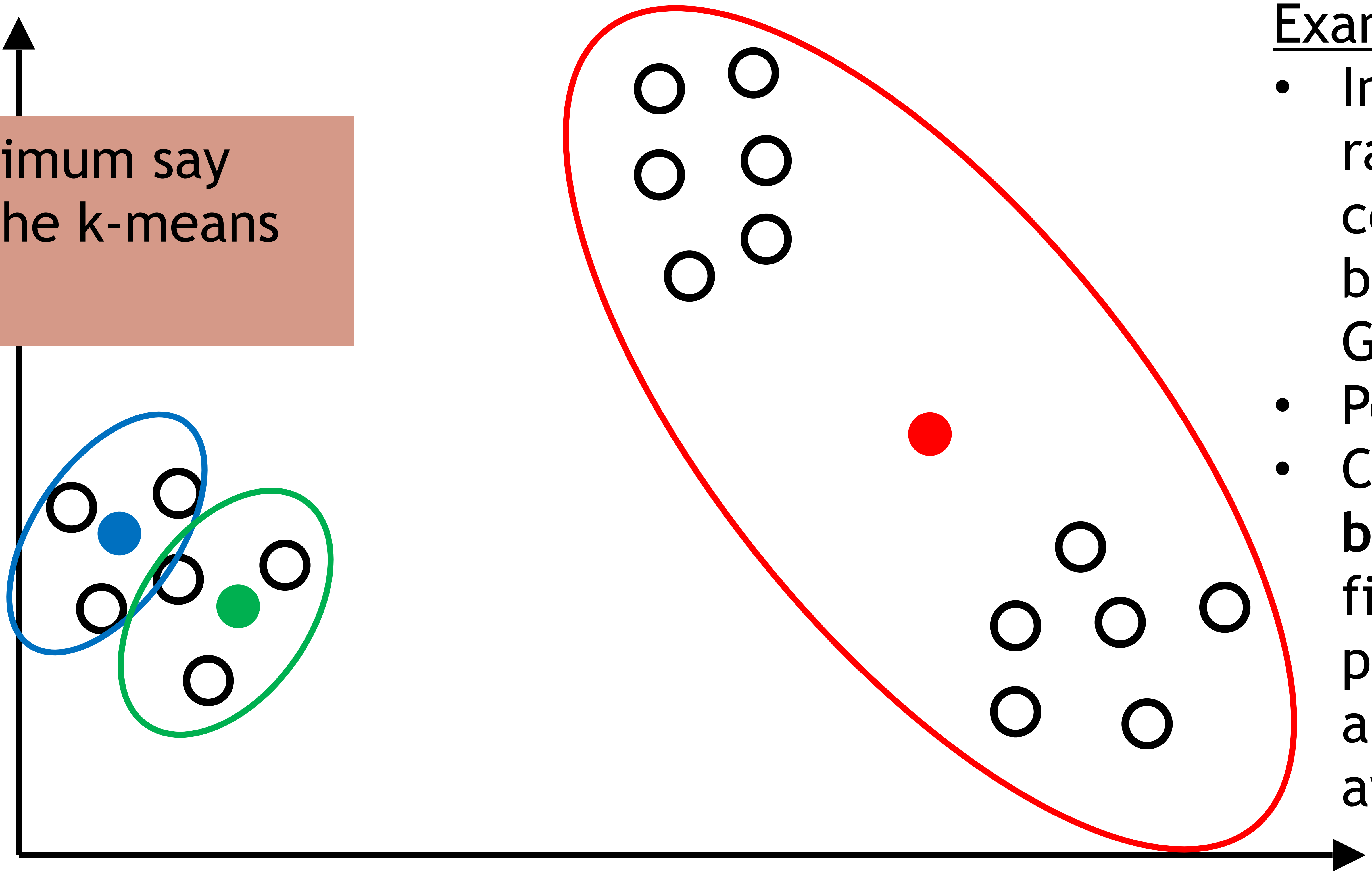
<u>Example 2:</u>
- Initialized randomly but two centers happen to be in same Gaussian cluster
- Poor performance
- Can be **arbitrarily bad** (imagine the final red cluster points moving arbitrarily far away!)

Initialization for K-Means

Algorithm #1: Random Initialization
Select each cluster center uniformly at random from the data points in the training data

Observations:
Even when data comes from well-separated Gaussians…
- …sometimes works great!
- …sometimes get stuck in poor local optima.

What does this local optimum say about the convexity of the k-means objective function?



Example 2:
- Initialized randomly but two centers happen to be in same Gaussian cluster
- Poor performance
- Can be **arbitrarily bad** (imagine the final red cluster points moving arbitrarily far away!)

Initialization for K-Means

k-means Performance (with Random Initialization)

If we do **random initialization**, as k increases, it becomes more likely we won't have perfectly picked one center per Gaussian in our initialization (so k-means will output a bad solution).

- For k equal-sized Gaussians,

$$\Pr[\text{each initial center is in a different Gaussian}] \approx \frac{k!}{k^k} \approx \frac{1}{e^k}$$

- Becomes unlikely as k gets large.

Slide courtesy of Nina Balcan

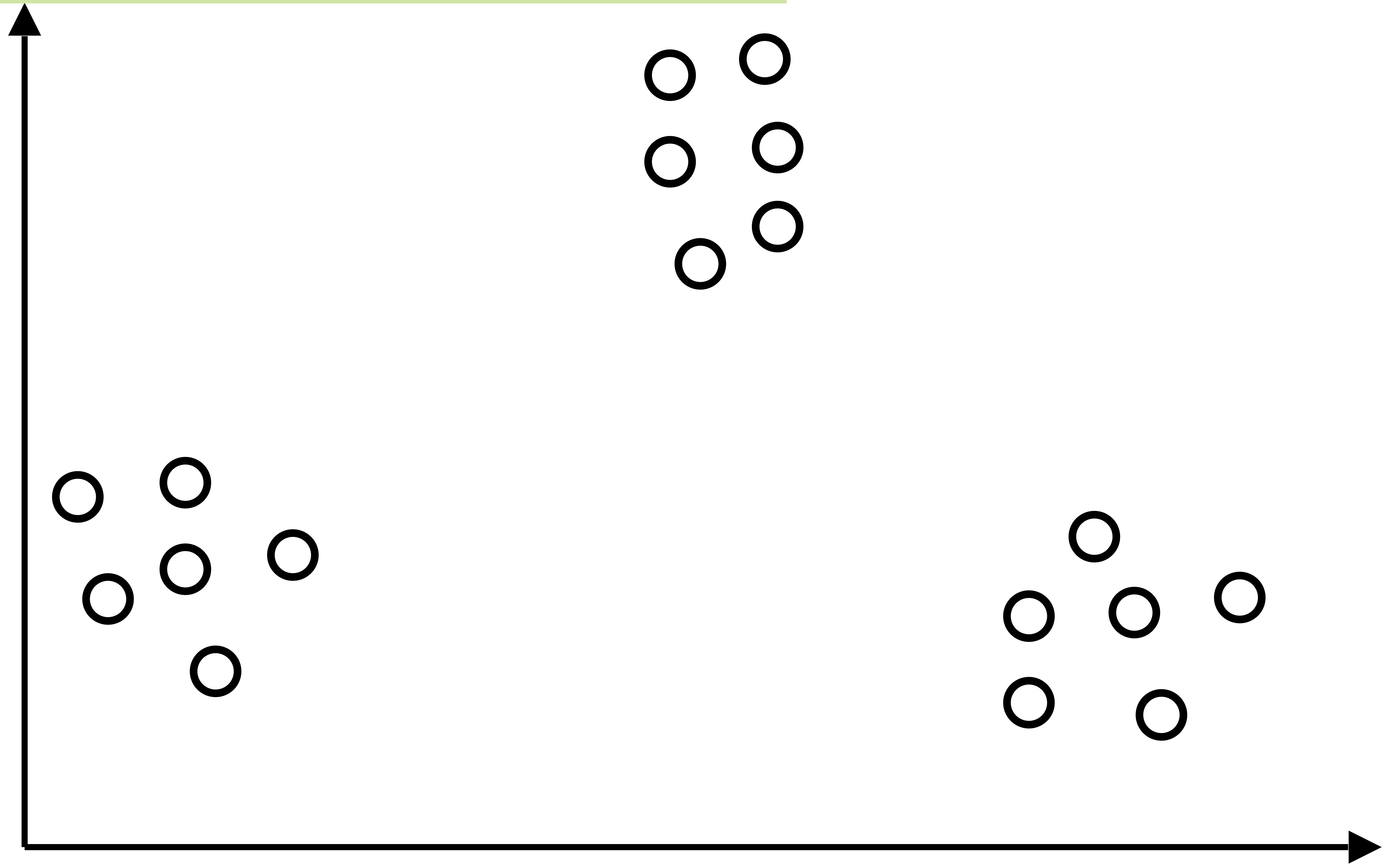
Initialization for K-Means

<u>Algorithm #2: Furthest Point Heuristic</u>

1. Pick the first cluster center $\mathbf{c}_1$ **randomly**
2. Pick each subsequent center $\mathbf{c}_j$ so that it is **as far as possible** from closest previously chosen center $\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_{j-1}$

<u>Observations:</u>

- OK if data is purely Gaussian
- But outliers pose a new problem!



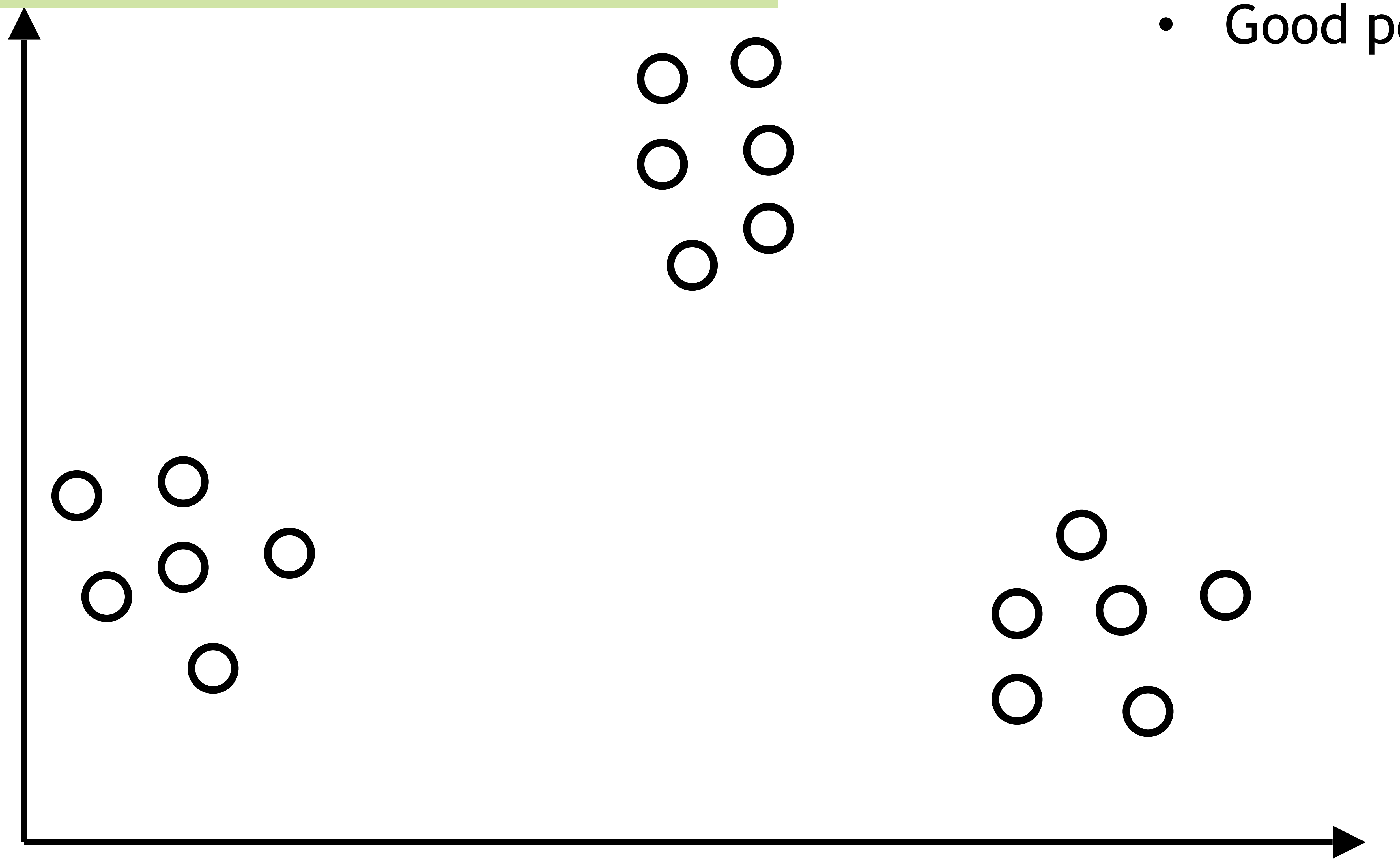
Initialization for K-Means

<u>Algorithm #2: Furthest Point Heuristic</u>

1. Pick the first cluster center $\mathbf{c}_1$ **randomly**
2. Pick each subsequent center $\mathbf{c}_j$ so that it is **as far as possible** from closest previously chosen center $\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_{j-1}$

<u>Observations:</u>

- OK if data is purely Gaussian
- But outliers pose a new problem!



<u>Example 1:</u>

- No outliers
- Good performance

Initialization for K-Means

Algorithm #2: Furthest Point Heuristic

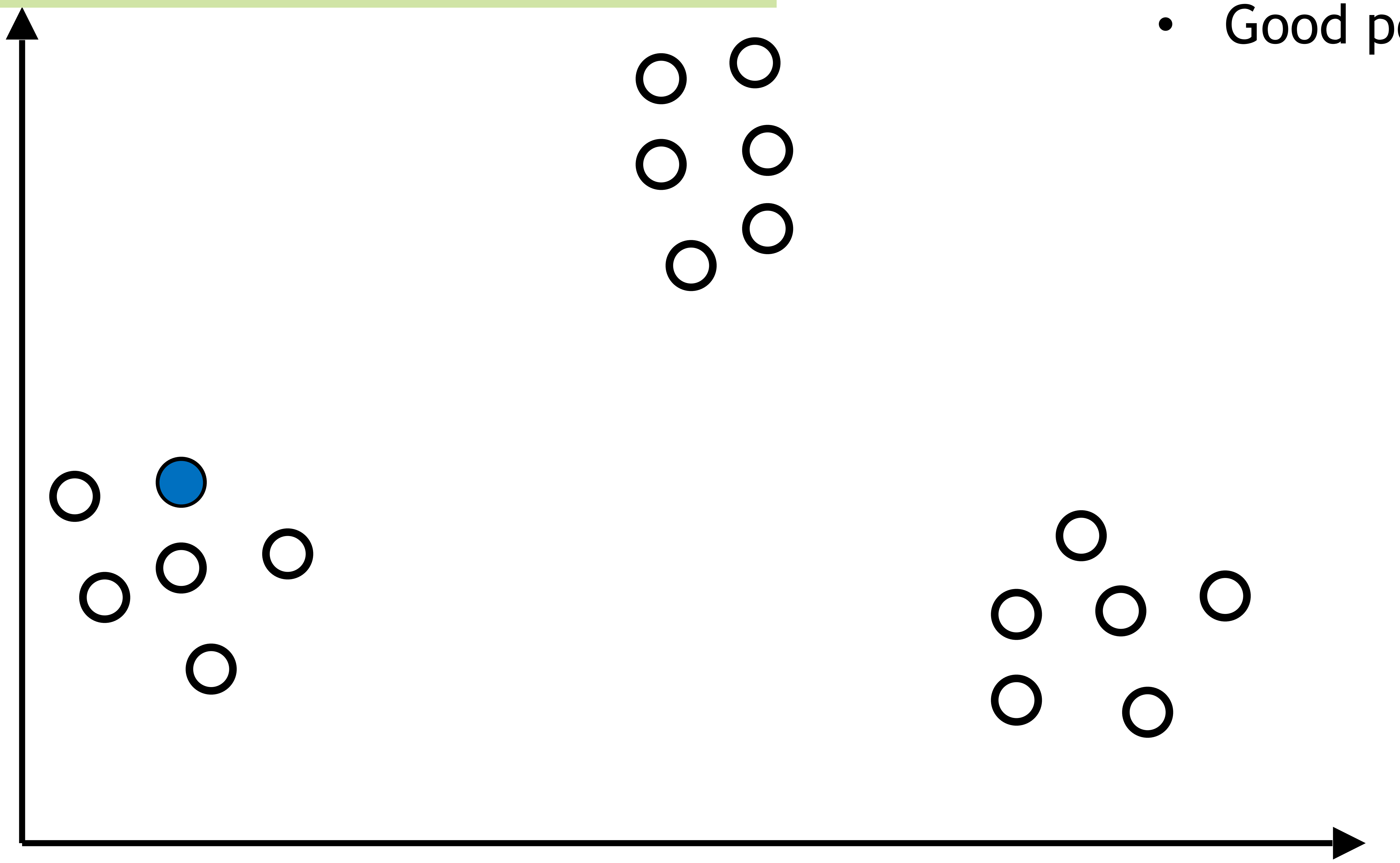
1. Pick the first cluster center $\mathbf{c}_1$ **randomly**
2. Pick each subsequent center $\mathbf{c}_j$ so that it is **as far as possible** from closest previously chosen center $\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_{j-1}$

Observations:

- OK if data is purely Gaussian
- But outliers pose a new problem!

Example 1:

- No outliers
- Good performance



Initialization for K-Means

Algorithm #2: Furthest Point Heuristic

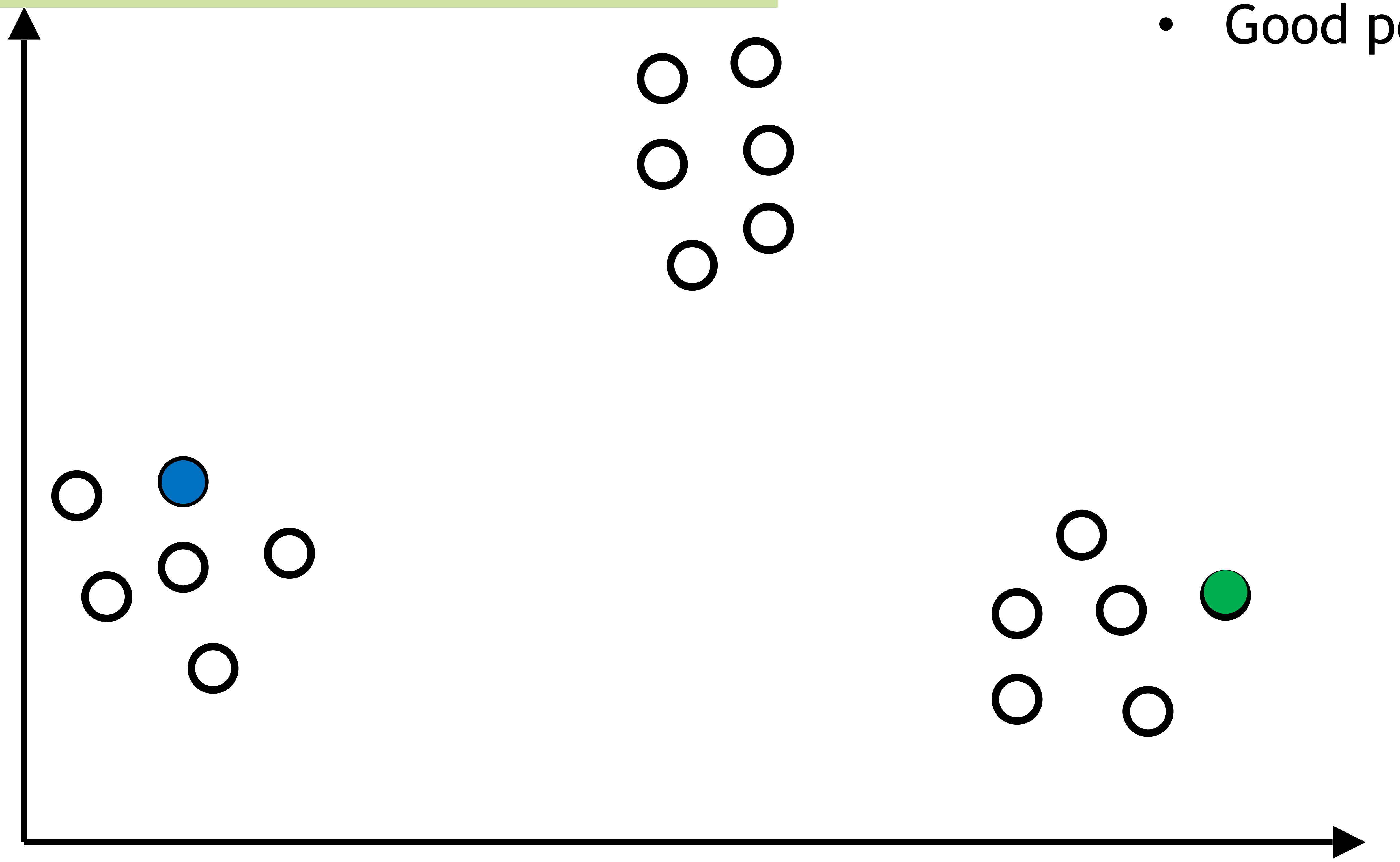
1. Pick the first cluster center $\mathbf{c}_1$ **randomly**
2. Pick each subsequent center $\mathbf{c}_j$ so that it is **as far as possible** from closest previously chosen center $\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_{j-1}$

Observations:

- OK if data is purely Gaussian
- But outliers pose a new problem!

Example 1:

- No outliers
- Good performance



Initialization for K-Means

Algorithm #2: Furthest Point Heuristic

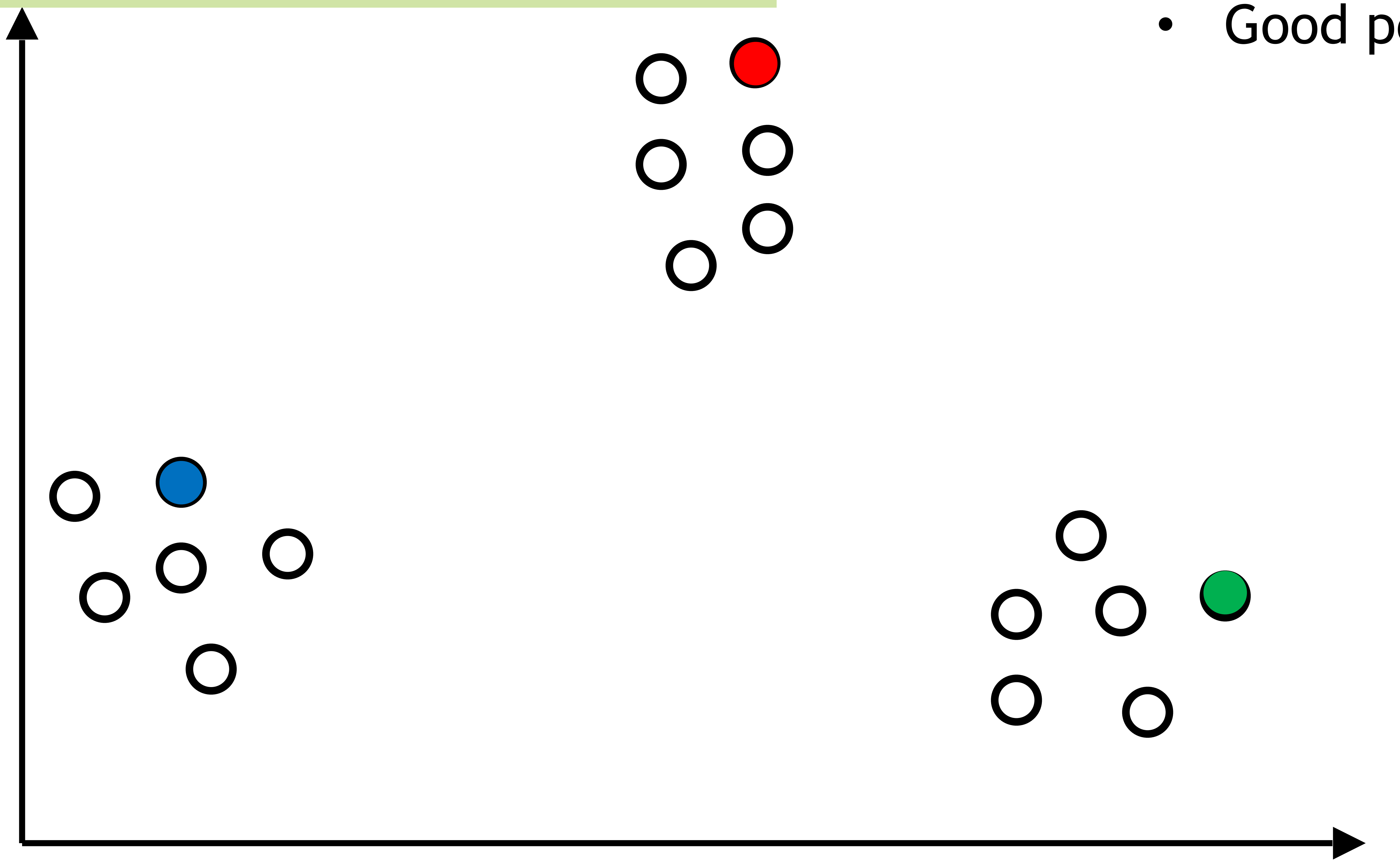
1. Pick the first cluster center c_1 **randomly**
2. Pick each subsequent center c_j so that it is **as far as possible** from closest previously chosen center $c_1, c_2, \ldots, c_{j-1}$

Observations:

- OK if data is purely Gaussian
- But outliers pose a new problem!

Example 1:

- No outliers
- Good performance



Initialization for K-Means

Algorithm #2: Furthest Point Heuristic

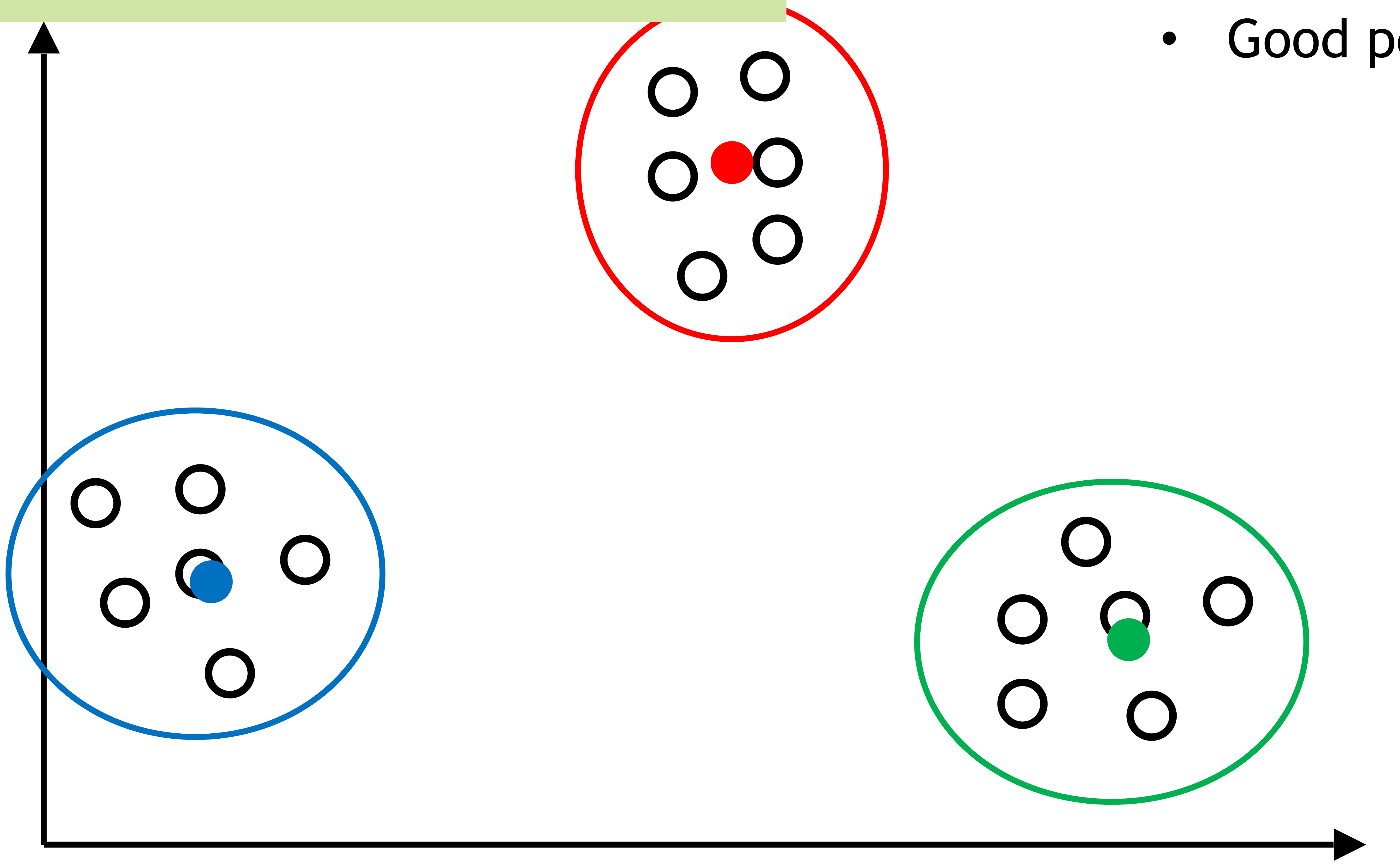
1. Pick the first cluster center $\mathbf{c}_1$ **randomly**
2. Pick each subsequent center $\mathbf{c}_j$ so that it is **as far as possible** from closest previously chosen center $\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_{j-1}$

Observations:

- OK if data is purely Gaussian
- But outliers pose a new problem!

Example 1:

- No outliers
- Good performance



Initialization for K-Means

Algorithm #2: Furthest Point Heuristic

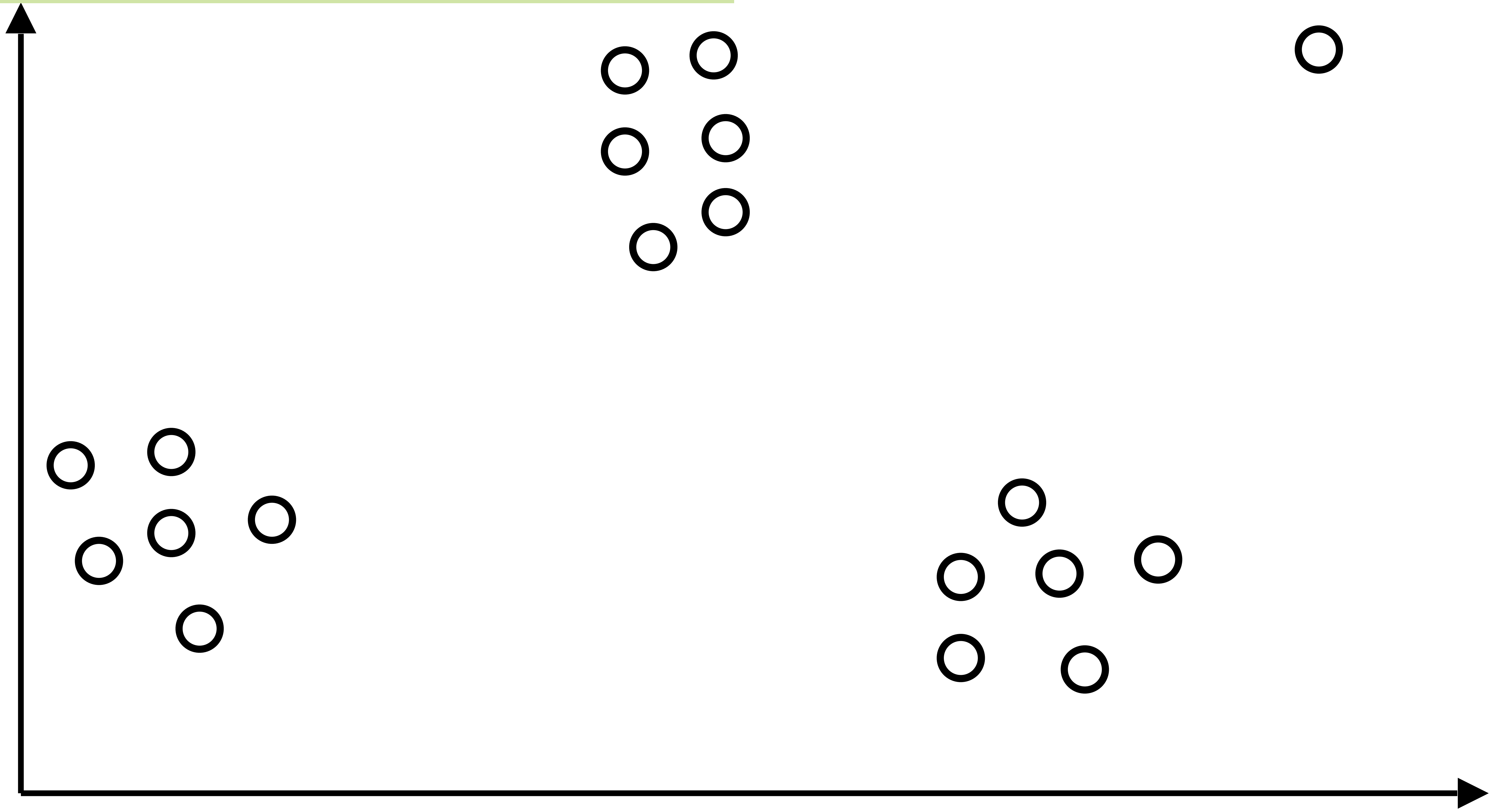
1. Pick the first cluster center $\mathbf{c}_1$ **randomly**
2. Pick each subsequent center $\mathbf{c}_j$ so that it is **as far as possible** from closest previously chosen center $\mathbf{c}_1$, $\mathbf{c}_2$,…, $\mathbf{c}_{j-1}$

Observations:

- OK if data is purely Gaussian
- But outliers pose a new problem!

Example 2:

- One outlier throws off the algorithm
- Poor performance



Initialization for K-Means

<u>Algorithm #2: Furthest Point Heuristic</u>

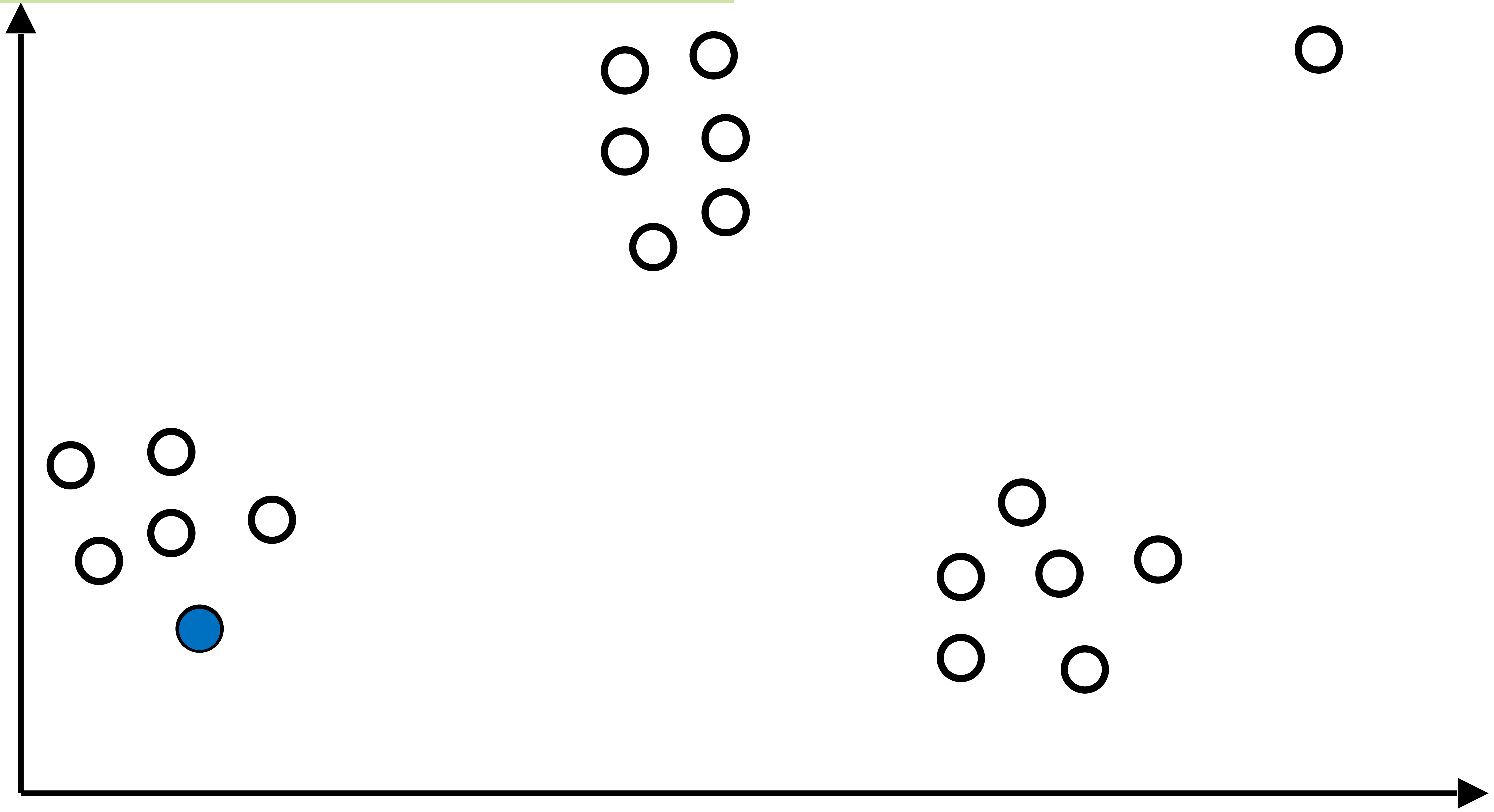
1. Pick the first cluster center $\mathbf{c}_1$ **randomly**
2. Pick each subsequent center $\mathbf{c}_j$ so that it is **as far as possible** from closest previously chosen center $\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_{j-1}$

<u>Observations:</u>

- OK if data is purely Gaussian
- But outliers pose a new problem!

<u>Example 2:</u>

- One outlier throws off the algorithm
- Poor performance



Initialization for K-Means

<u>Algorithm #2: Furthest Point Heuristic</u>

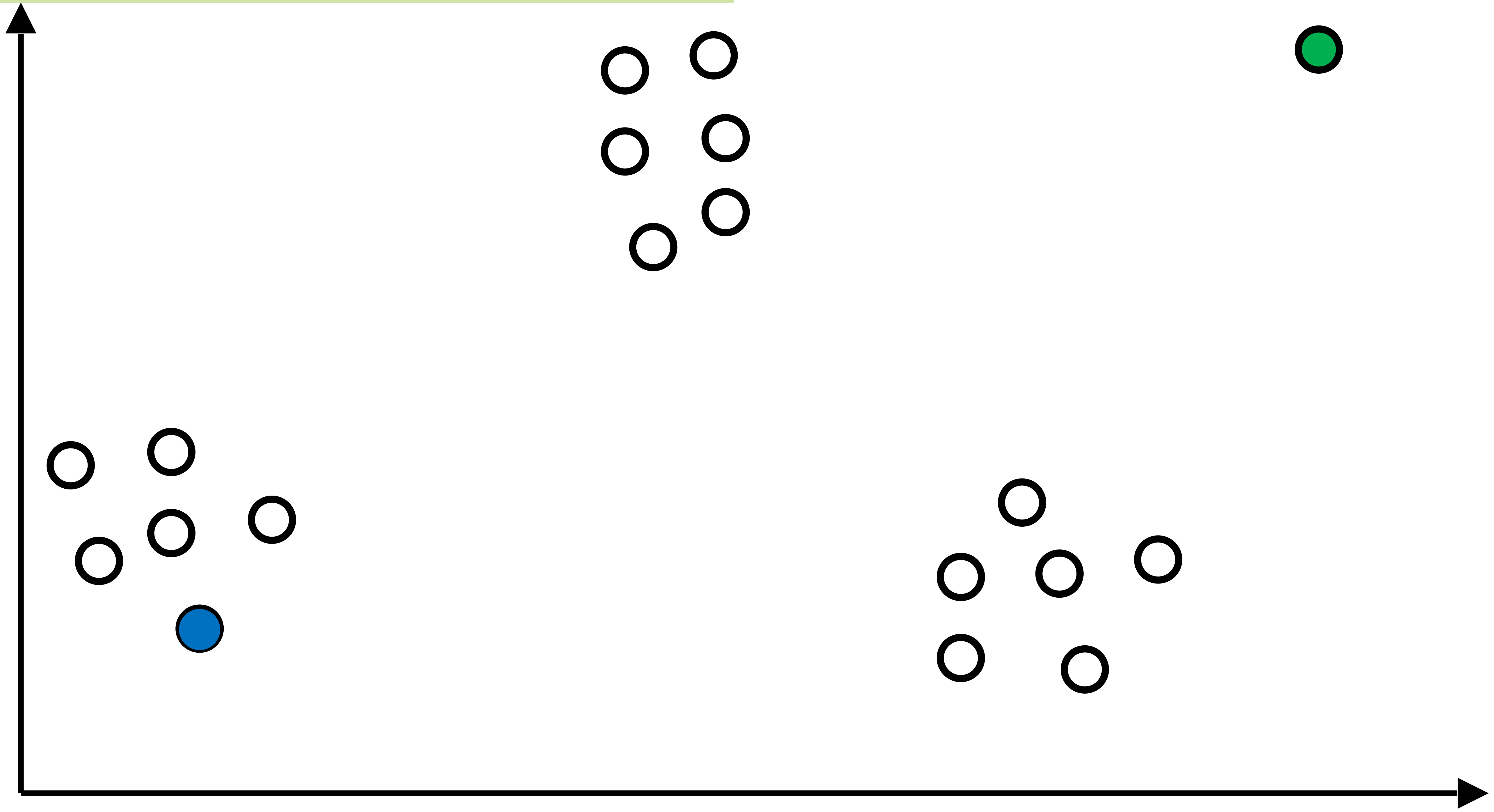
1. Pick the first cluster center $\mathbf{c}_1$ **randomly**
2. Pick each subsequent center $\mathbf{c}_j$ so that it is **as far as possible** from closest previously chosen center $\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_{j-1}$

<u>Observations:</u>

- OK if data is purely Gaussian
- But outliers pose a new problem!

<u>Example 2:</u>

- One outlier throws off the algorithm
- Poor performance



Initialization for K-Means

<u>Algorithm #2: Furthest Point Heuristic</u>

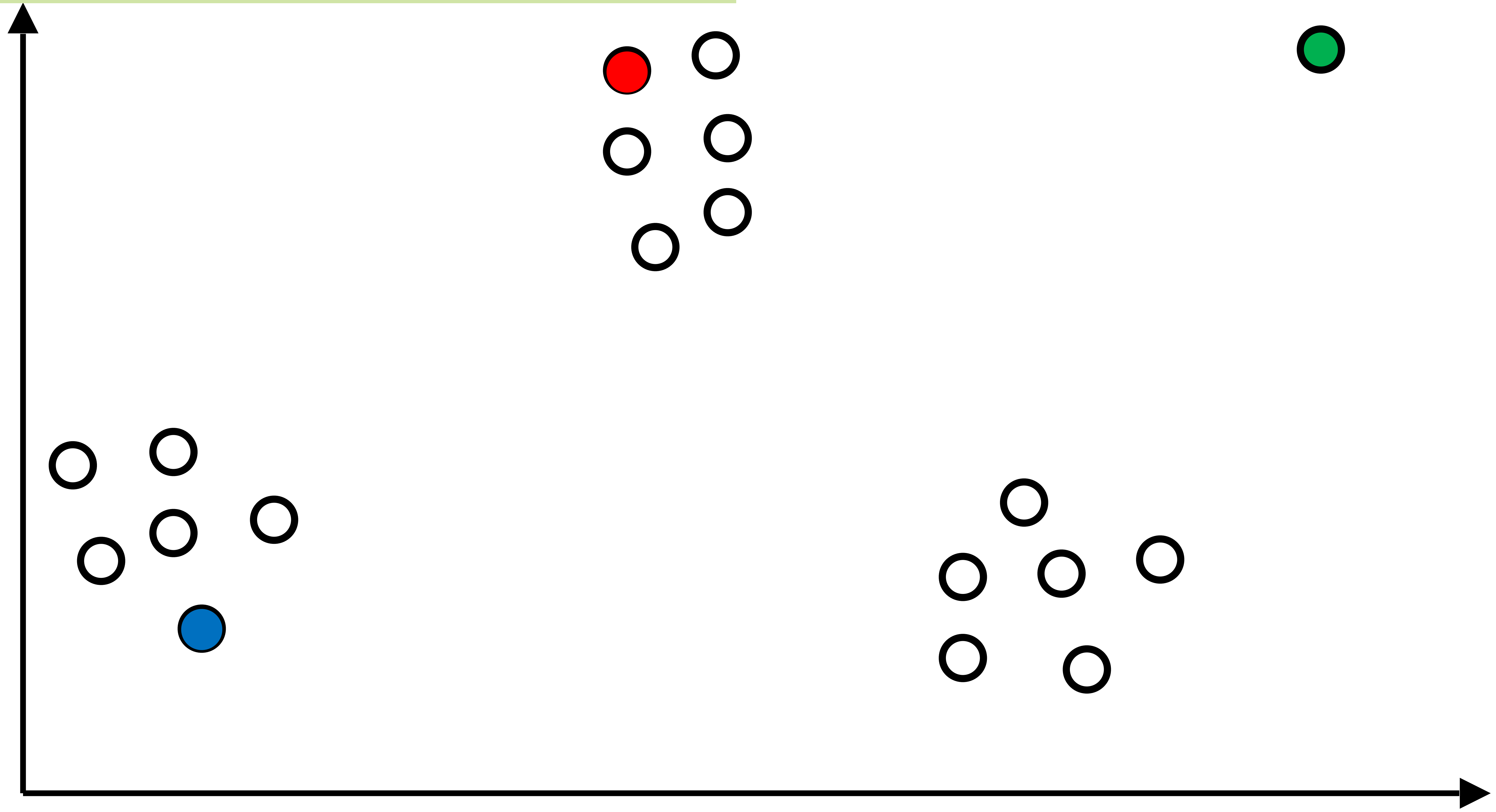
1. Pick the first cluster center $\mathbf{c}_1$ **randomly**
2. Pick each subsequent center $\mathbf{c}_j$ so that it is **as far as possible** from closest previously chosen center $\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_{j-1}$

<u>Observations:</u>

- OK if data is purely Gaussian
- But outliers pose a new problem!

<u>Example 2:</u>

- One outlier throws off the algorithm
- Poor performance

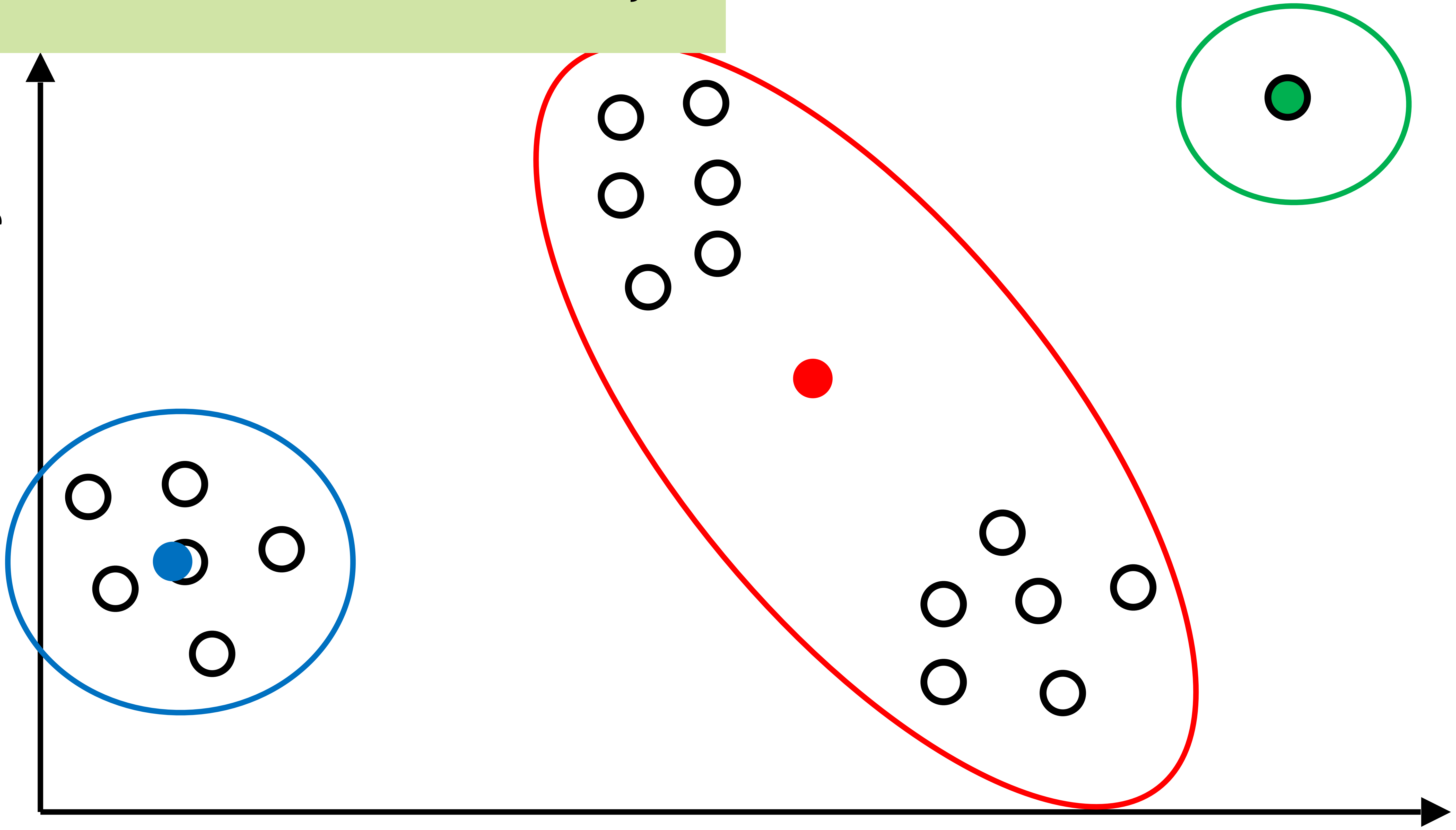


Initialization for K-Means

Algorithm #2: Furthest Point Heuristic
1. Pick the first cluster center $\mathbf{c}_1$ **randomly**
2. Pick each subsequent center $\mathbf{c}_j$ so that it is **as far as possible** from closest previously chosen center $\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_{j-1}$

Observations:
- OK if data is purely Gaussian
- But outliers pose a new problem!

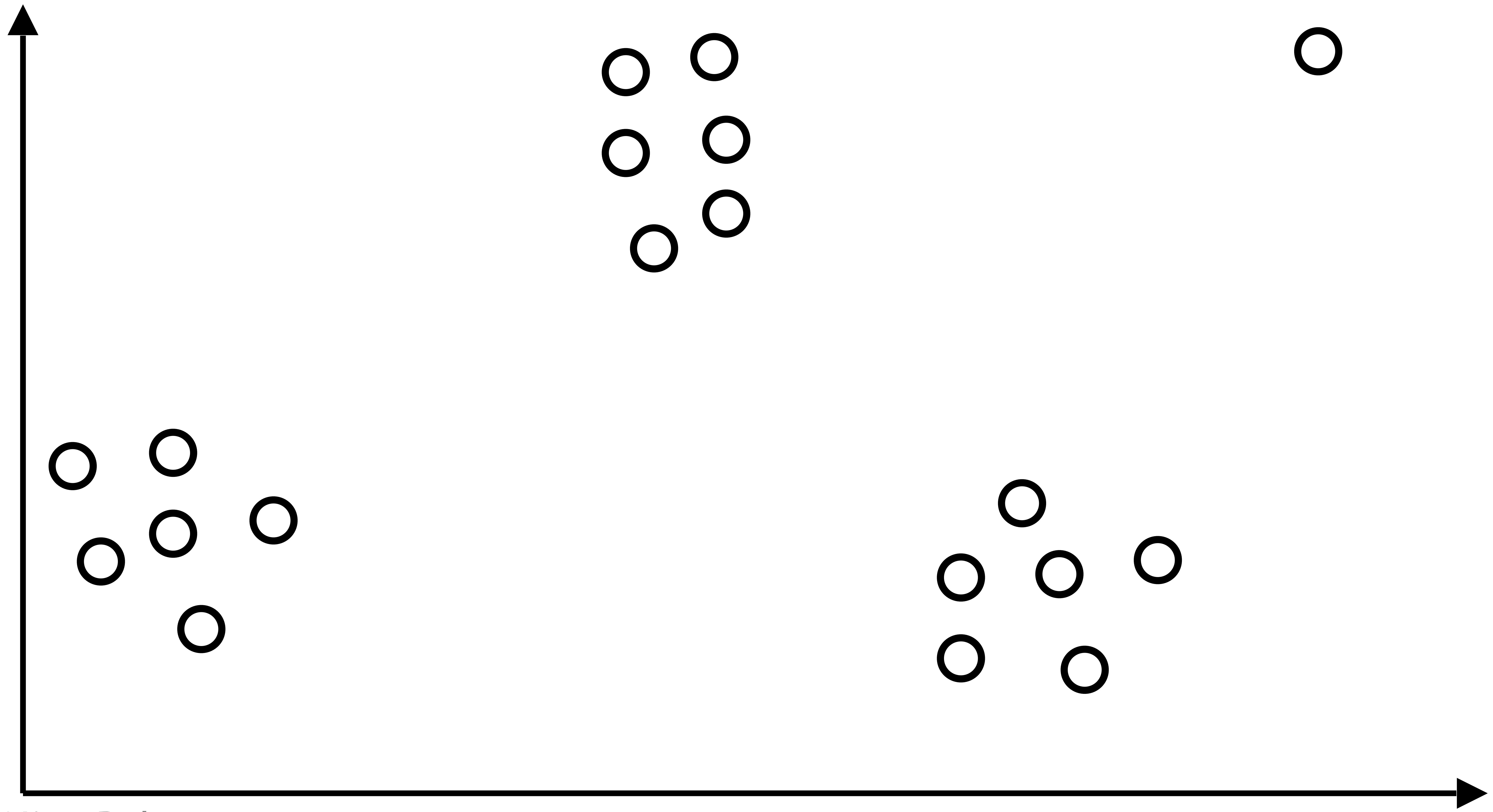
Example 2:
- One outlier throws off the algorithm
- Poor performance



Initialization for K-Means

<u>Algorithm #3: K-Means++</u>

- Let D(x) be the distance between a point x and its nearest center. Choose next center proportional to $D(x)^2$. (1st one uniformly random.)



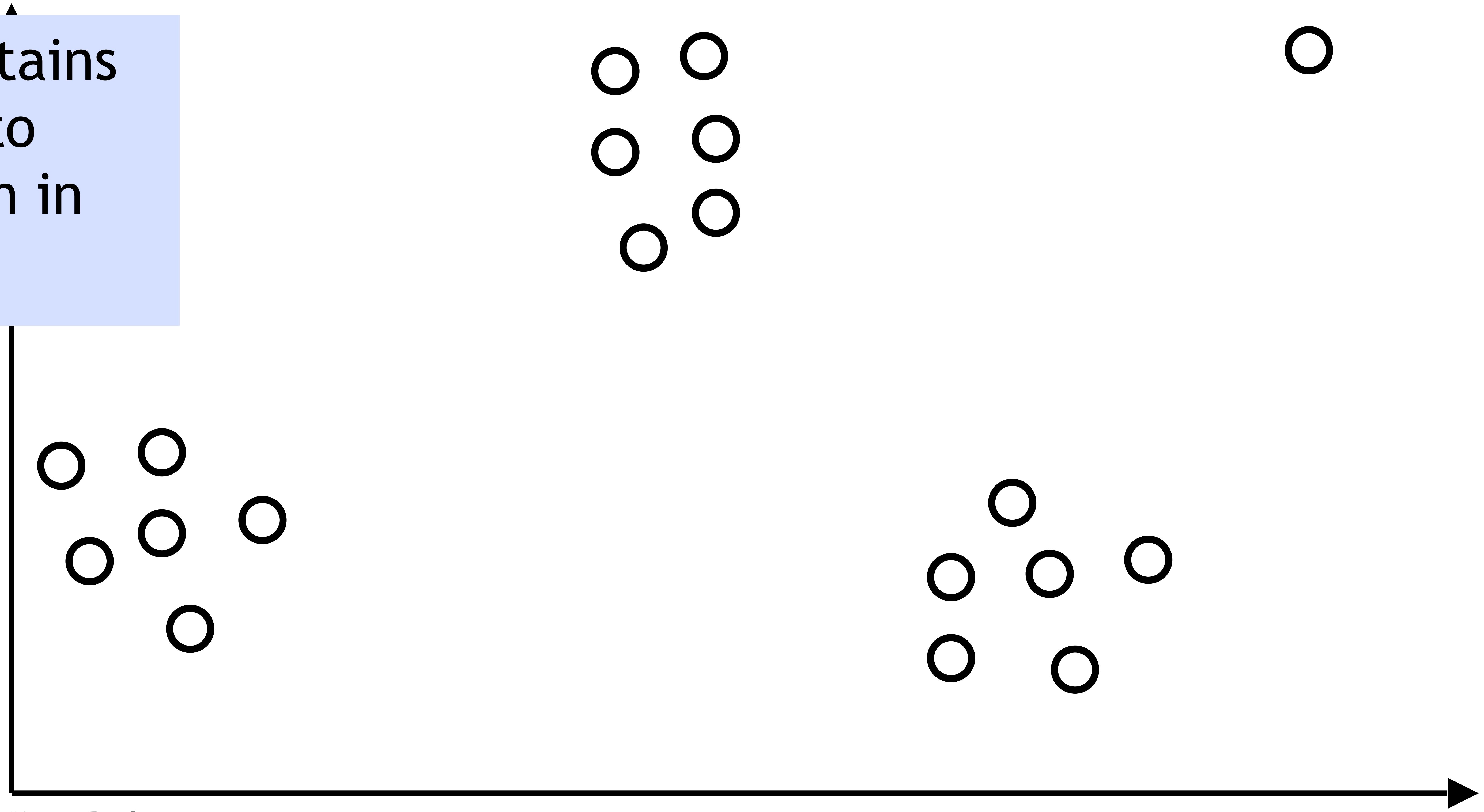
Slide adapted from Nina Balcan

Initialization for K-Means

Algorithm #3: K-Means++

- Let $D(x)$ be the distance between a point x and its nearest center. Choose next center proportional to $D(x)^2$. (1st one uniformly random.)

Theorem: K-Means++ attains $O(\log k)$ approximation to optimal K-Means solution in expectation.

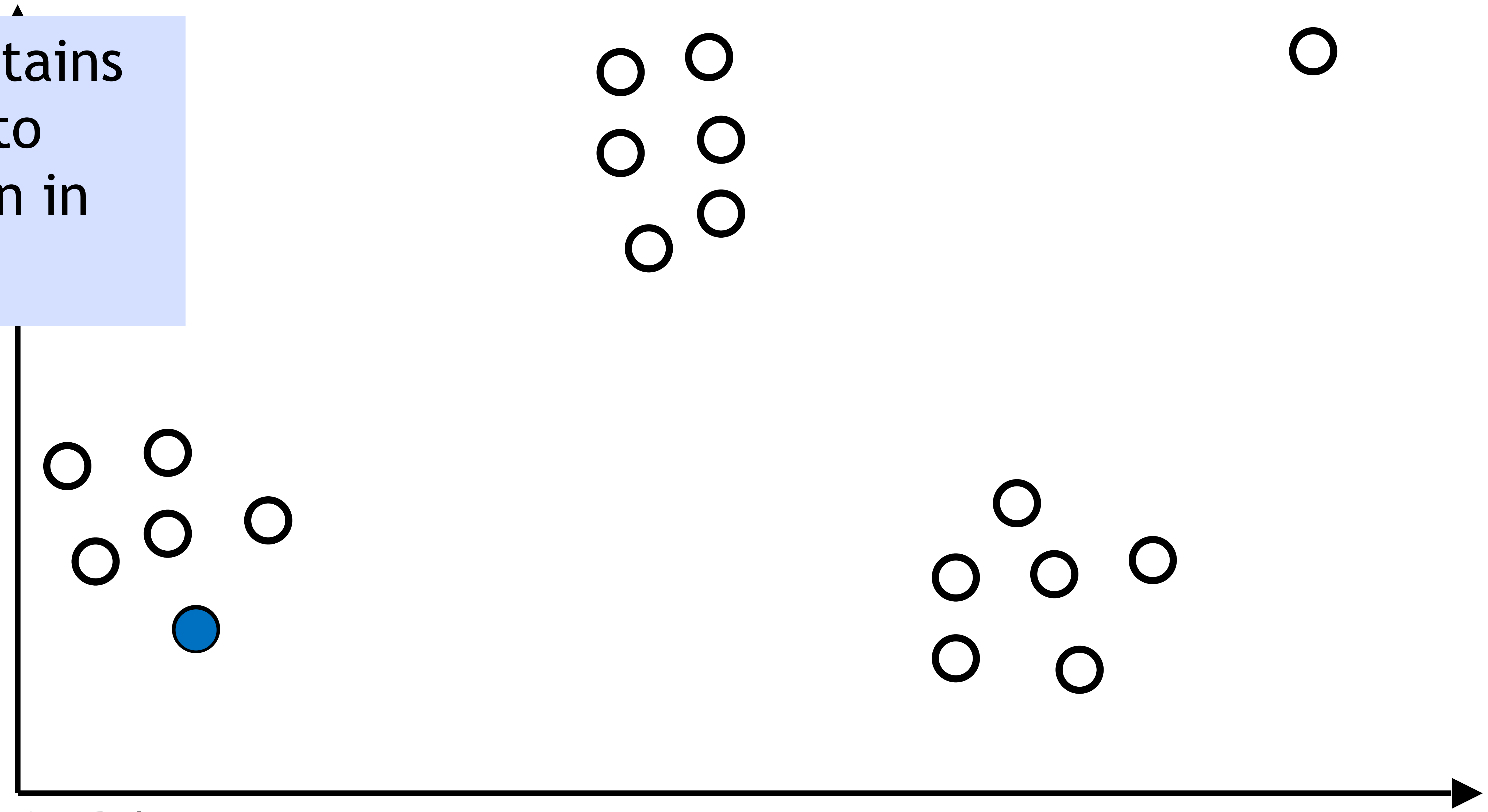


Slide adapted from Nina Balcan

Initialization for K-Means

Algorithm #3: K-Means++
- Let $D(x)$ be the distance between a point x and its nearest center. Choose next center proportional to $D(x)^2$. (1st one uniformly random.)

Theorem: K-Means++ attains $O(\log k)$ approximation to optimal K-Means solution in expectation.



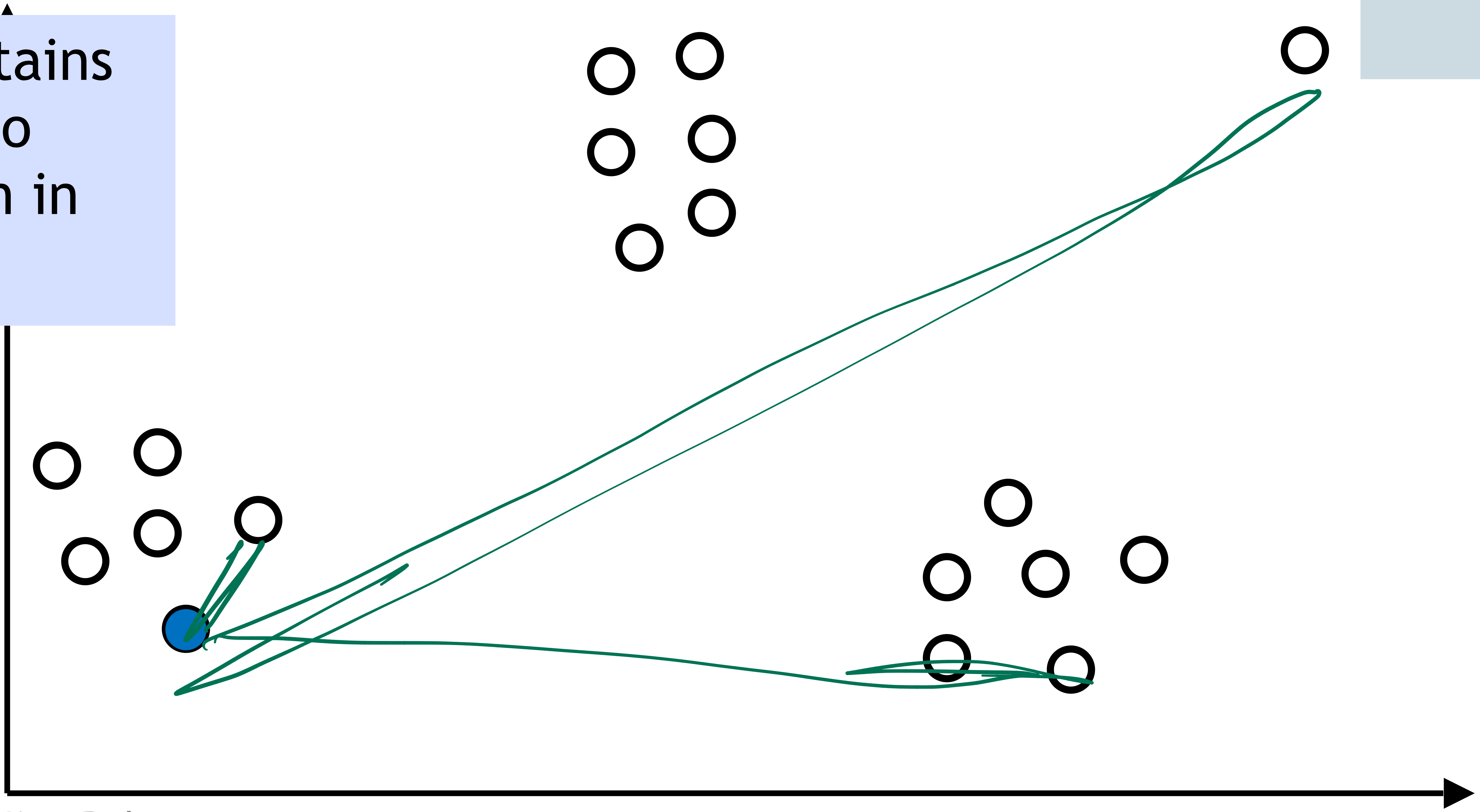
Slide adapted from Nina Balcan

Initialization for K-Means

Algorithm #3: K-Means++
- Let D(x) be the distance between a point x and its nearest center. Choose next center proportional to $D(x)^2$. (1st one uniformly random.)

i	D(x)	$D^2(x)$	$P(v_2=x^{(i)})$
1	3	9	9/137
2	2	4	4/137
…			
7	4	16	16/137
…			
N	3	9	9/137
	Sum:	137	1.0

Theorem: K-Means++ attains O(log k) approximation to optimal K-Means solution in expectation.



Slide adapted from Nina Balcan

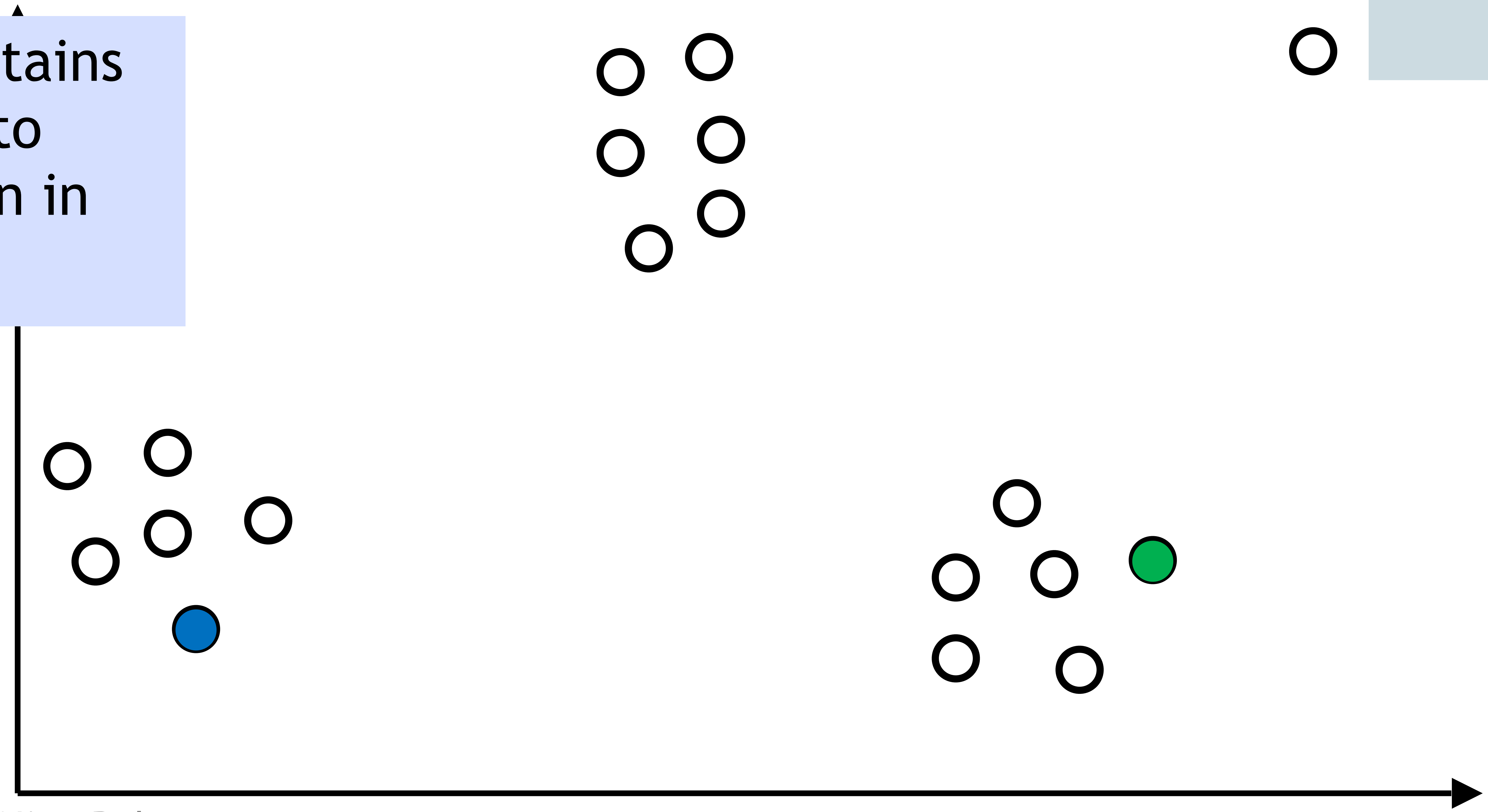
Initialization for K-Means

i	D(x)	$D^2(x)$	$P(v_2=x^{(i)})$
1	3	9	9/137
2	2	4	4/137
…			
7	4	16	16/137
…			
N	3	9	9/137
	Sum:	137	1.0

<u>Algorithm #3: K-Means++</u>

- Let D(x) be the distance between a point **x** and its nearest center. Choose next center proportional to $D(x)^2$. (1st one uniformly random.)

Theorem: K-Means++ attains O(log k) approximation to optimal K-Means solution in expectation.



Slide adapted from Nina Balcan

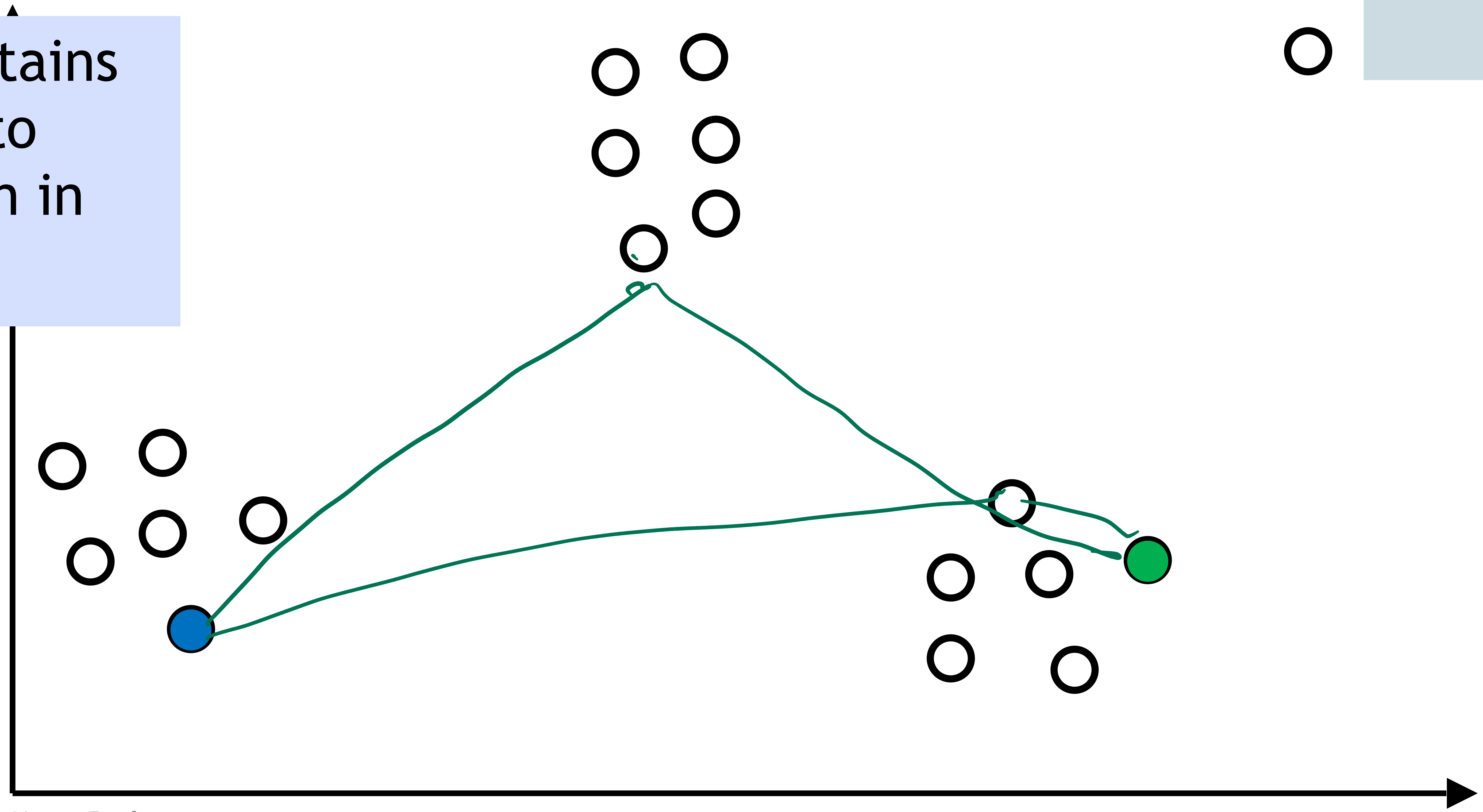
Initialization for K-Means

i	D(x)	$D^2(x)$	$P(v_2=x^{(i)})$
1	3	9	9/102
2	2	4	4/102
…			
7	3	9	9/102
…			
N	2	4	4/102
	Sum:	102	1.0

Algorithm #3: K-Means++

- Let D(x) be the distance between a point **x** and its nearest center. Choose next center proportional to $D(x)^2$. (1st one uniformly random.)

Theorem: K-Means++ attains O(log k) approximation to optimal K-Means solution in expectation.



Slide adapted from Nina Balcan

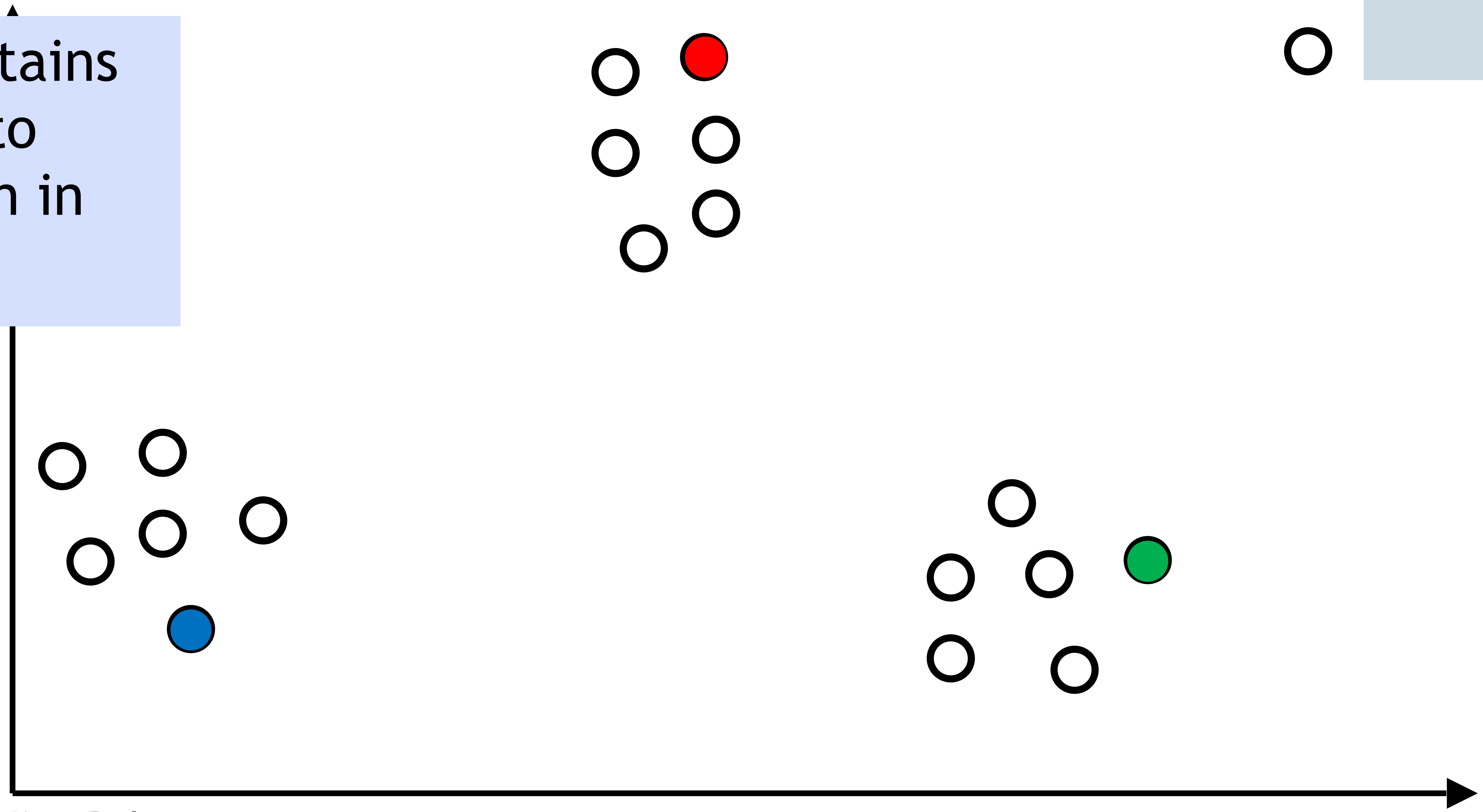
Initialization for K-Means

i	D(x)	$D^2(x)$	$P(v_2=x^{(i)})$
1	3	9	9/102
2	2	4	4/102
…			
7	3	9	9/102
…			
N	2	4	4/102
	Sum:	102	1.0

<u>Algorithm #3: K-Means++</u>

- Let D(x) be the distance between a point **x** and its nearest center. Choose next center proportional to $D(x)^2$. (1st one uniformly random.)

Theorem: K-Means++ attains O(log k) approximation to optimal K-Means solution in expectation.



Slide adapted from Nina Balcan

Initialization for K-Means

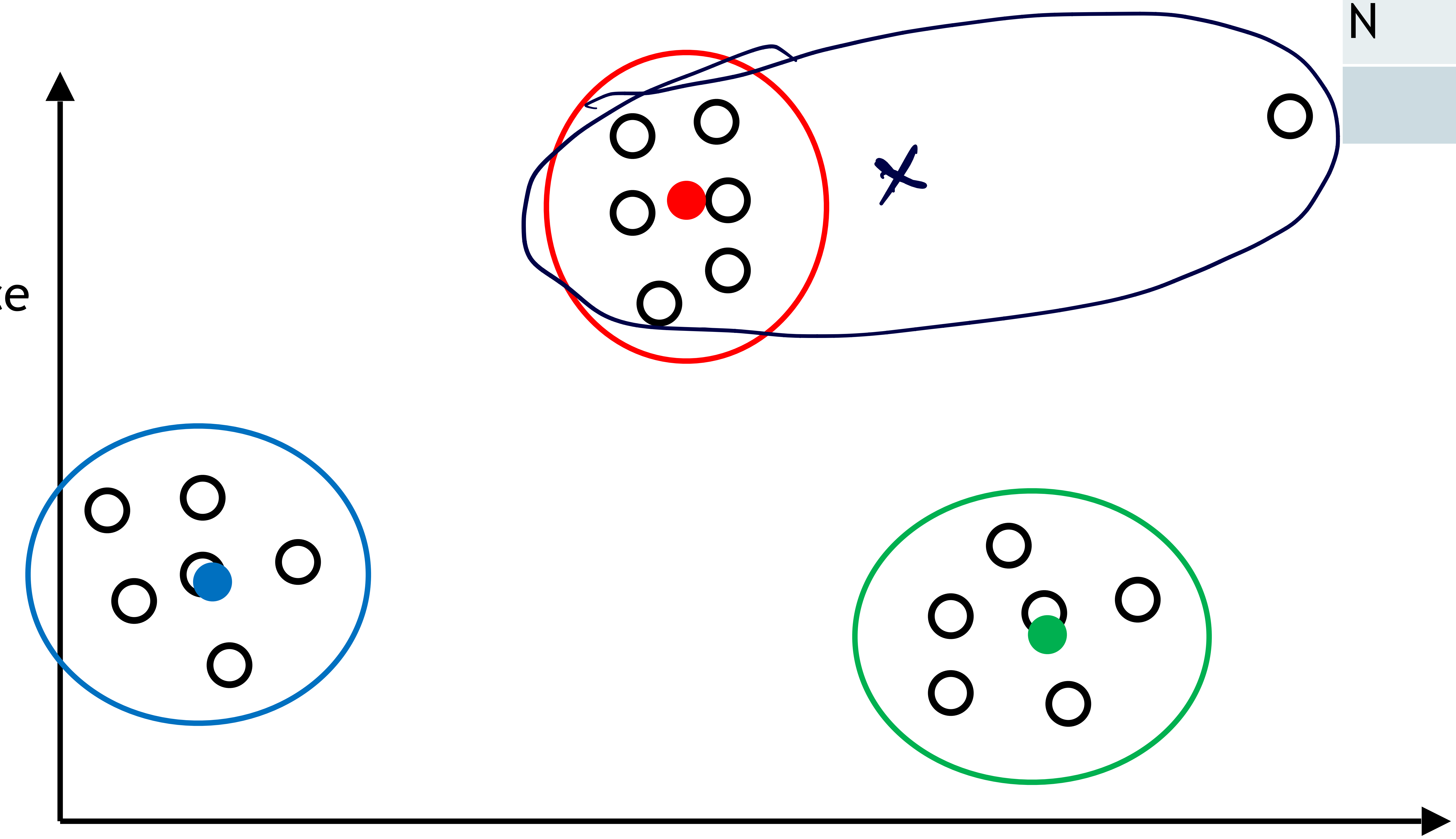
i	D(x)	$D^2(x)$	$P(v_2=x^{(i)})$
1	3	9	9/137
2	2	4	4/137
…			
7	4	16	16/137
…			
N	3	9	9/137
	Sum:	137	1.0

Algorithm #3: K-Means++

- Let D(x) be the distance between a point x and its nearest center. Choose next center proportional to $D(x)^2$. (1st one uniformly random.)

Example 1:

- One outlier
- Good performance



Initialization for K-Means

<u>Algorithm #3: K-Means++</u>

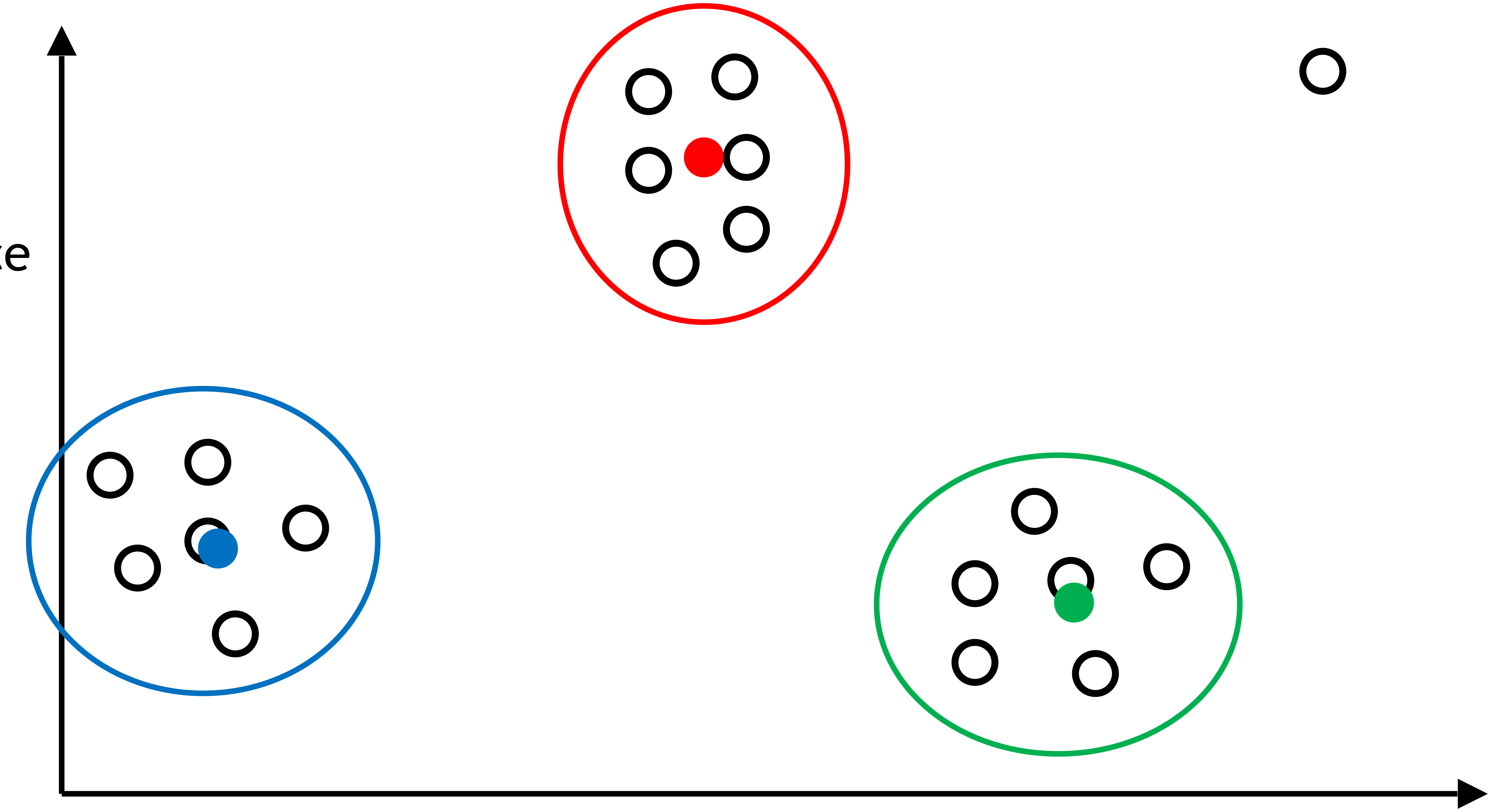
- Let D(x) be the distance between a point **x** and its nearest center. Choose next center proportional to $D(x)^2$. (1st one uniformly random.)

<u>Observations:</u>

- Interpolates between random and farthest point initialization
- Solves the problem with Gaussian data
- **And** solves the outlier problem

<u>Example 1:</u>

- One outlier
- Good performance



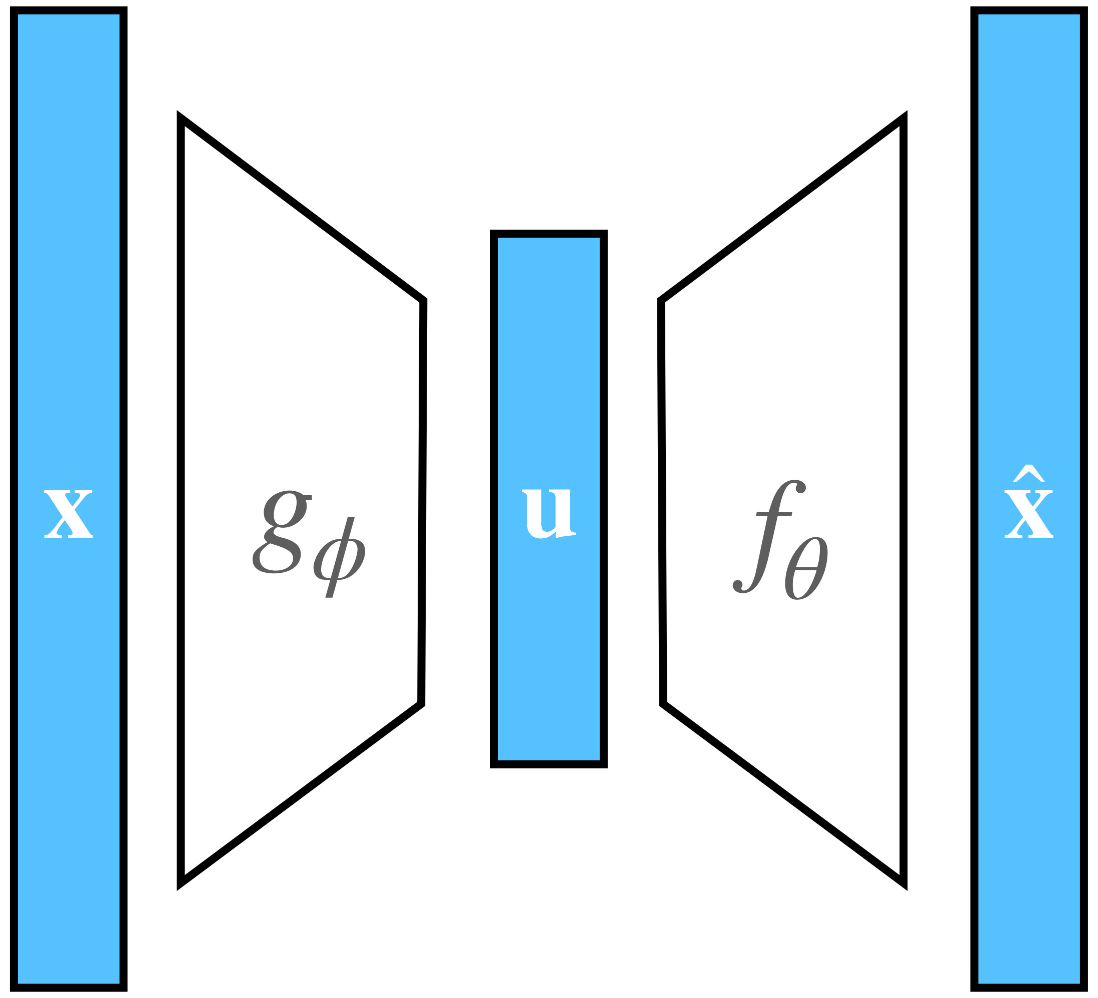
Learning Objectives

K-Means

You should be able to…

1. Distinguish between coordinate descent and block coordinate descent
2. Define an objective function that gives rise to a "good" clustering (preferring each point to be close to nearest center)
3. Apply block coordinate descent to this objective function to obtain the K-Means algorithm
4. Implement the K-Means algorithm
5. Connect the non-convexity of the K-Means objective function with the (possibly) poor performance of random initialization

Deep autoencoder

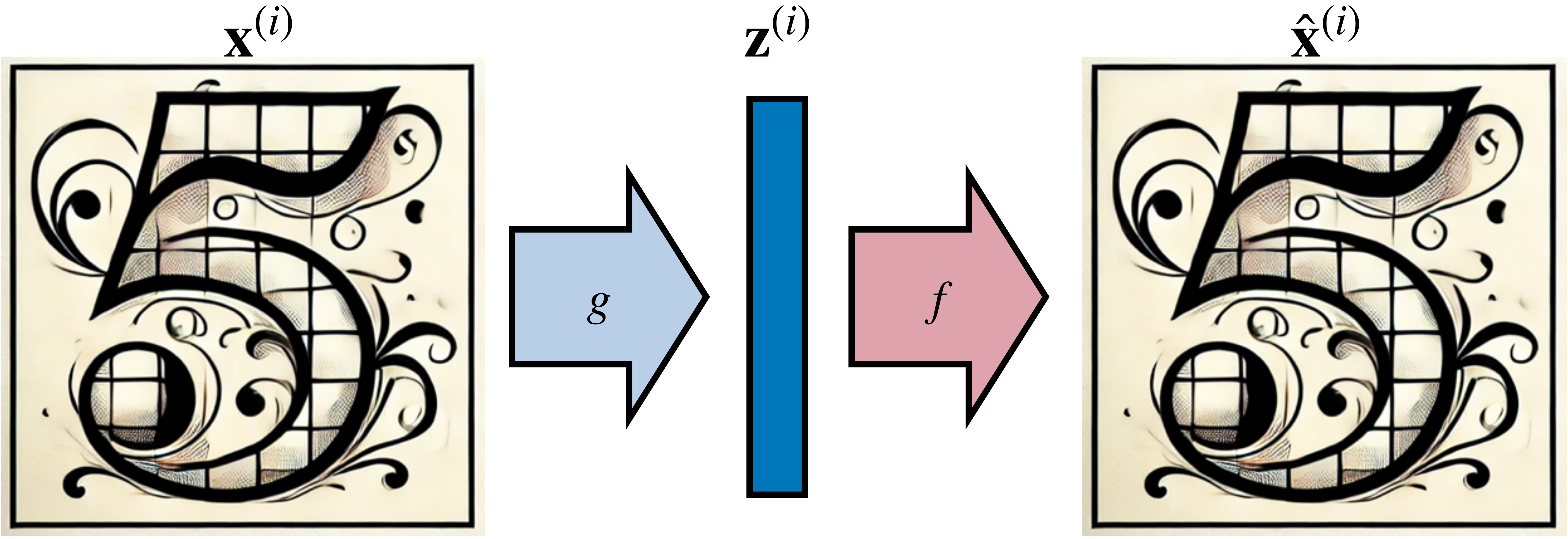


- Can design versions of all of the above autoencoders that use deep nets instead of simpler functions
- E.g., deep autoencoder (train by SGD on $\|\hat{\mathbf{x}}^{(i)} - \mathbf{x}^{(i)}\|$):
  - $\mathbf{u} = g_\phi(\mathbf{x})$
  - $\hat{\mathbf{x}} = f_\theta(\mathbf{u})$
  - f_θ, g_ϕ are deep nets: convolutional layers for images, transformer blocks for text, and all the tools we've covered like residual connections, layer norm, …

Latent distribution

- Current SOTA autoencoder models (VAE, diffusion models) use one more modification on top of above
- So far: hidden layer was $\mathbb{R}^k$ (continuous) or $\{\mathbf{e}_1, \ldots \mathbf{e}_k\}$ (discrete)
- Mod: hidden layer is a *probability distribution*
  - over a set like $\mathbb{R}^k$ or $\{\mathbf{e}_1, \ldots \mathbf{e}_k\}$ or $\{\mathbf{e}_1, \ldots \mathbf{e}_k\}^m$ (a grid)
  - continuous even if it's a distribution over a discrete set
- To fit, need a new tool: ***variational methods***

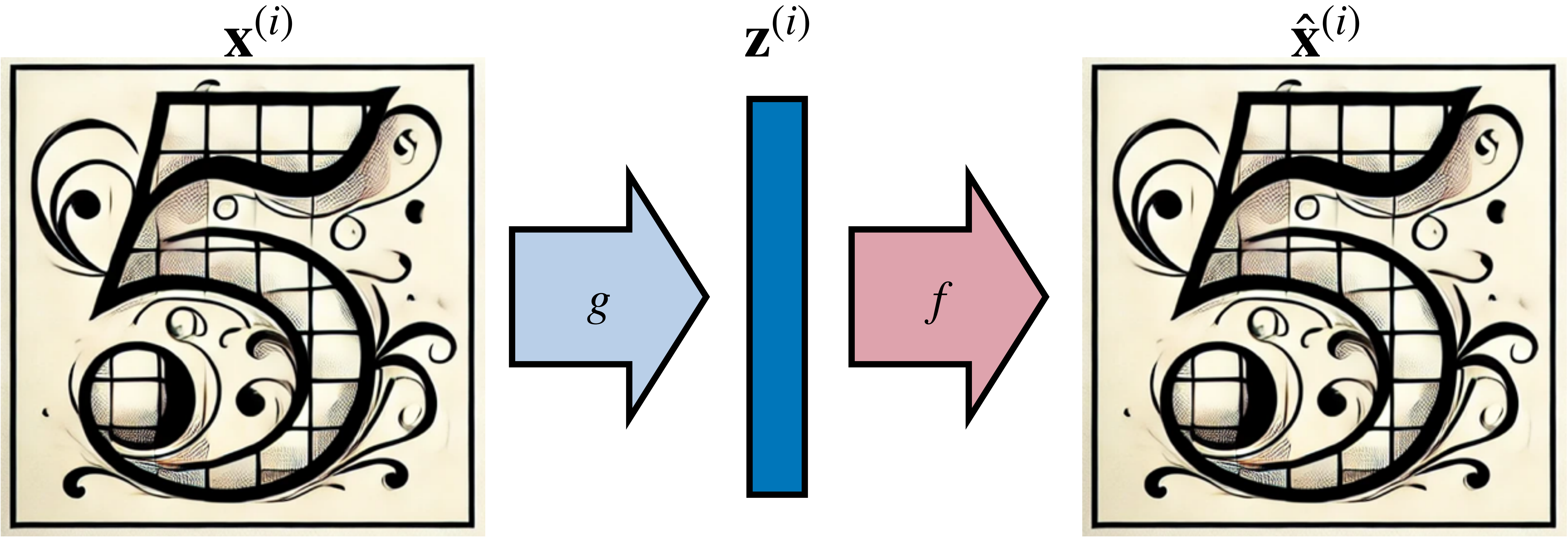
Variational autoencoder (VAE)



- VAE is a complete probabilistic generative model (unlike previous autoencoders this lecture)
  - $\mathbf{z}^{(i)} \sim N(0, I), \ \hat{\mathbf{x}}^{(i)} = f_\theta(\mathbf{z}^{(i)})$ + noise
  - f_θ a deep net — the *decoder*
- Auxiliary deep net $g_\phi(\mathbf{x}^{(i)})$ — the *encoder*
  - g_ϕ is not part of the generative model
  - instead, approximates posterior $P(\mathbf{z}^{(i)} \mid \mathbf{x}^{(i)}, \theta)$

Geoff Gordon

Variational autoencoder (VAE)



- Overall goal: maximize log-likelihood

$$\max_\theta \sum_{i=1}^{N} \ln P(\mathbf{x}^{(i)} \mid \theta)$$

- Log-likelihood is intractable to compute: need an integral over posterior of $\mathbf{z}^{(i)}$

$$\max_\theta \sum_{i=1}^{N} \ln \mathbb{E}_{\mathbf{z} \sim P(\mathbf{z}^{(i)} \mid \mathbf{x}^{(i)}, \theta)} P(\mathbf{x}^{(i)} \mid \mathbf{z}^{(i)}, \theta)$$

- Approximate posterior from encoder g_ϕ will help us work around this problem

Geoff Gordon

VAE training

- At any point in training, we have an approximate posterior for each example's distribution over latents
  - $P(\mathbf{z}^{(i)} \mid \mathbf{x}^{(i)}, \theta) \approx g_\phi(\mathbf{x}^{(i)})$
- Use samples from approximate posterior to make a lower bound on log-likelihood $\ln P(\mathbf{x}^{(i)} \mid \theta)$ — what we really want to maximize
  - need bound because true log-likelihood is intractable
  - bound is called the ***ELBO*** (evidence lower bound)
- Take SGD steps to maximize ELBO wrt θ
  - increasing the lower bound also pushes up on true log likelihood, allowing us to increase it while avoiding an intractable gradient calculation

VAE example

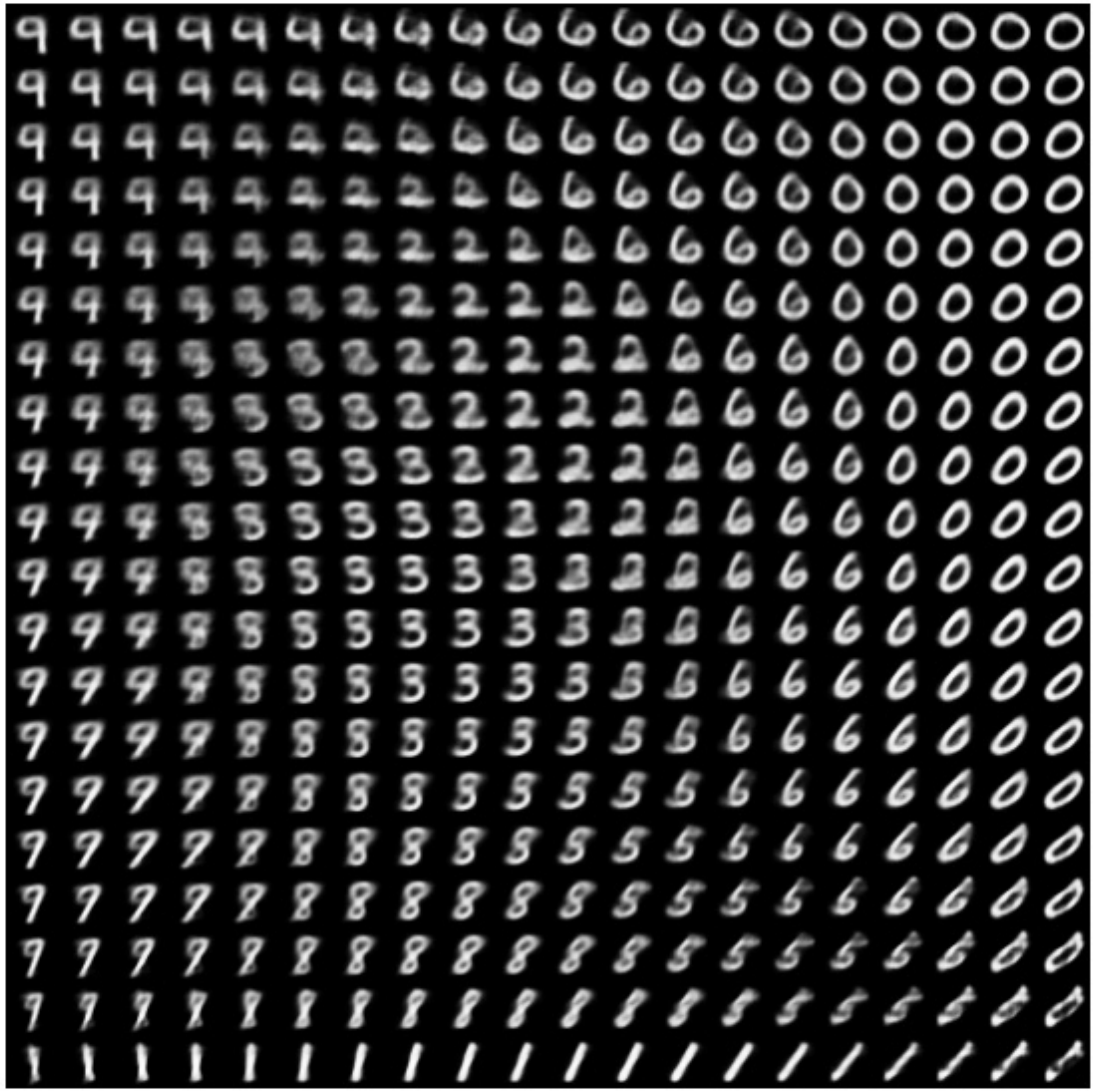


image credit: [TensorFlow]

- Task: compress MNIST digits from observed $\mathbb{R}^{28\times28}$ to latent $\mathbb{R}^2$, then generate samples from the learned model, scanning $\mathbf{z}$ across a grid in $\mathbb{R}^2$