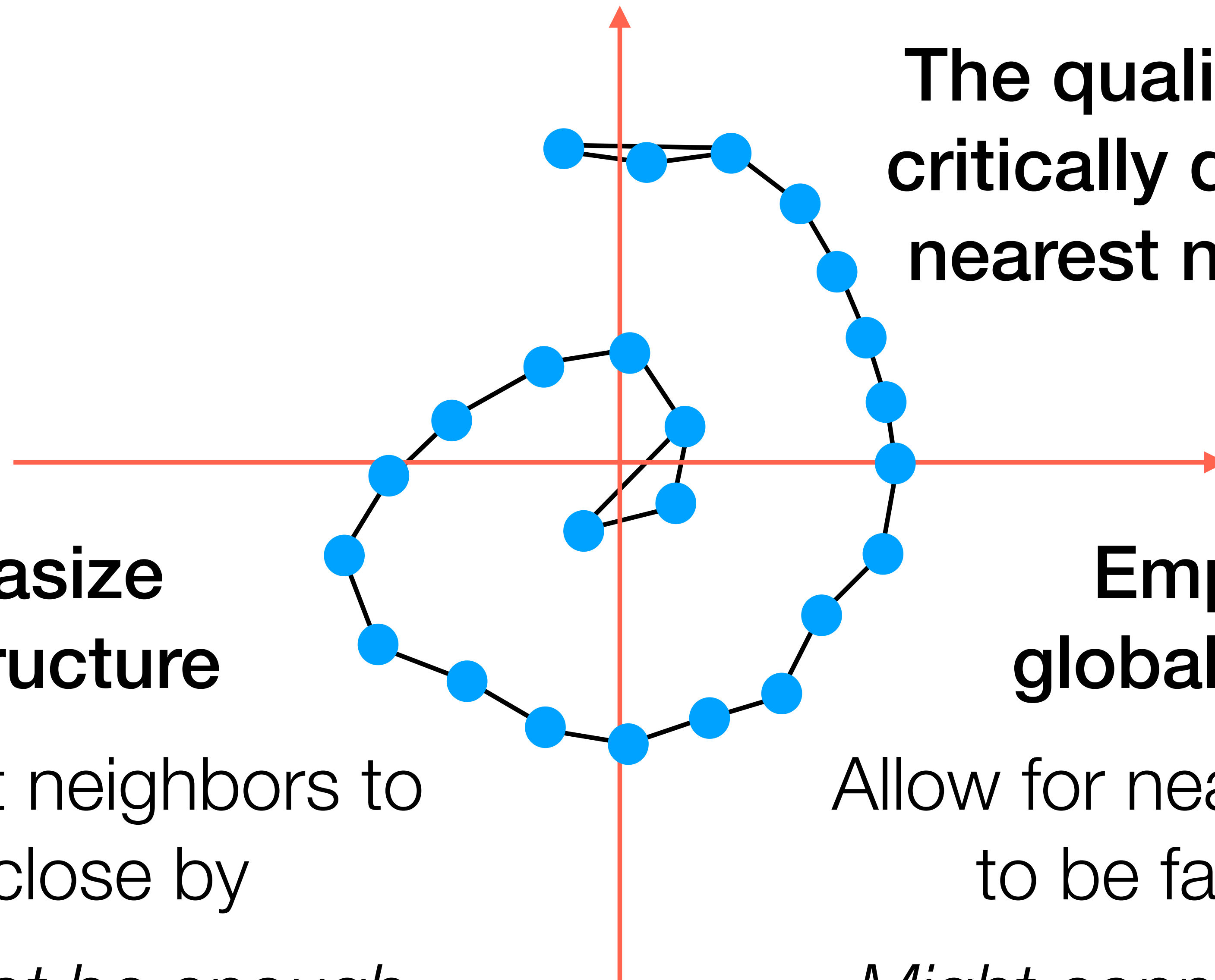
Carnegie Mellon University
Heinz College

Unstructured Data Analysis

Lecture 6: Wrap up manifold learning (t-SNE), a first look at analyzing images, and an introduction to clustering phenomena

George Chen

(Flashback) Some Observations on Isomap



The quality of the result critically depends on the nearest neighbor graph

Emphasize local structure

Ask for nearest neighbors to be really close by

There might not be enough edges

Emphasize global structure

Allow for nearest neighbors to be farther away

Might connect points that shouldn't be connected

In general: try different parameters for nearest neighbor graph construction when using Isomap + visualize

(Flashback) Isomap

Build k-NN graph, computed shortest distances

If k is set too large and we connect everything: Isomap just becomes MDS

Original high-dim. data

Distance table (for high-dim. points)

Make these two as close as possible

(Euclidean dist)

Low-dim. data

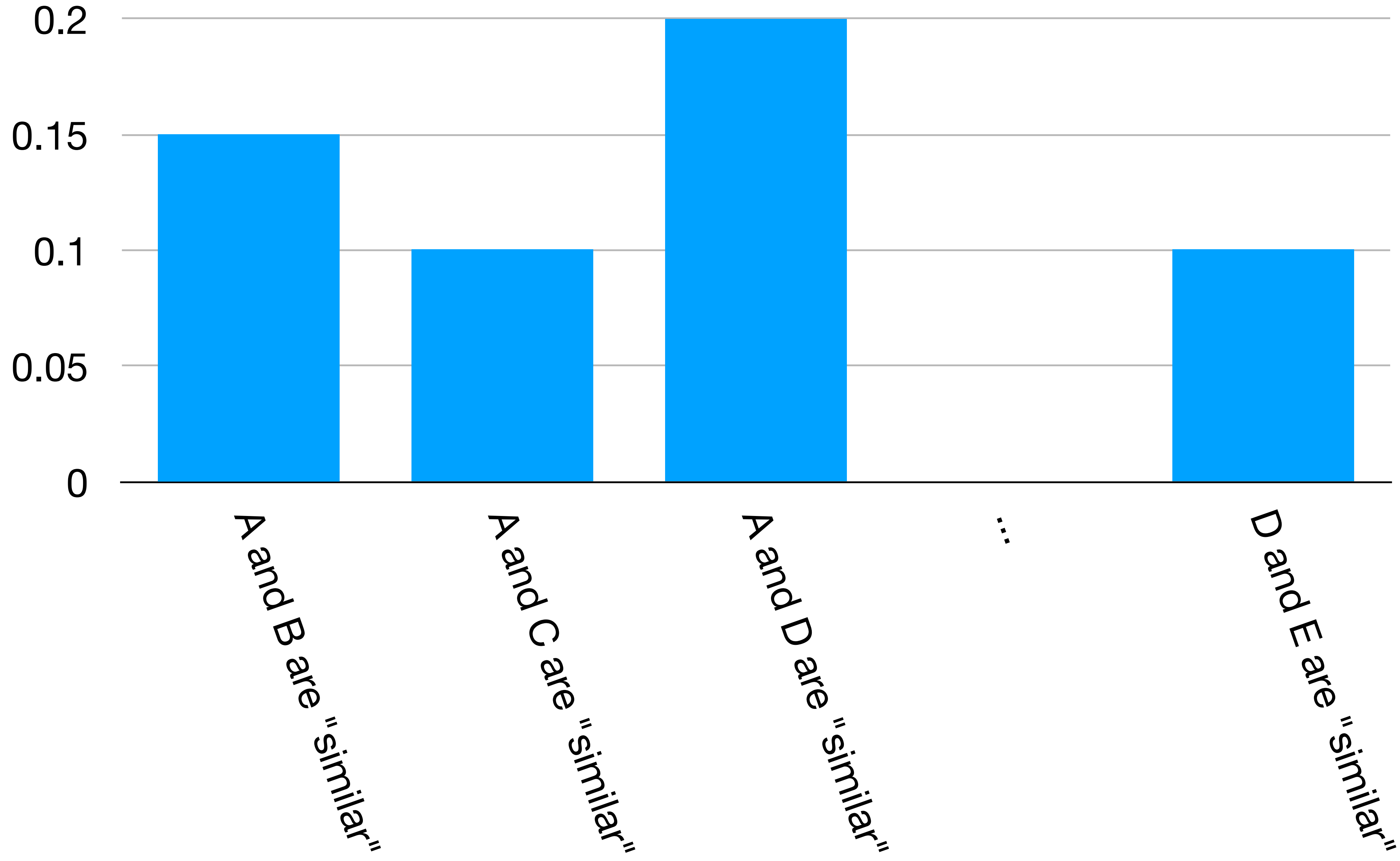
Distance table (for low-dim. points)

Compute Euclidean distances between all pairs of low-dimensional points

t-SNE
(t-distributed stochastic neighbor embedding)

t-SNE High-Level Idea #1

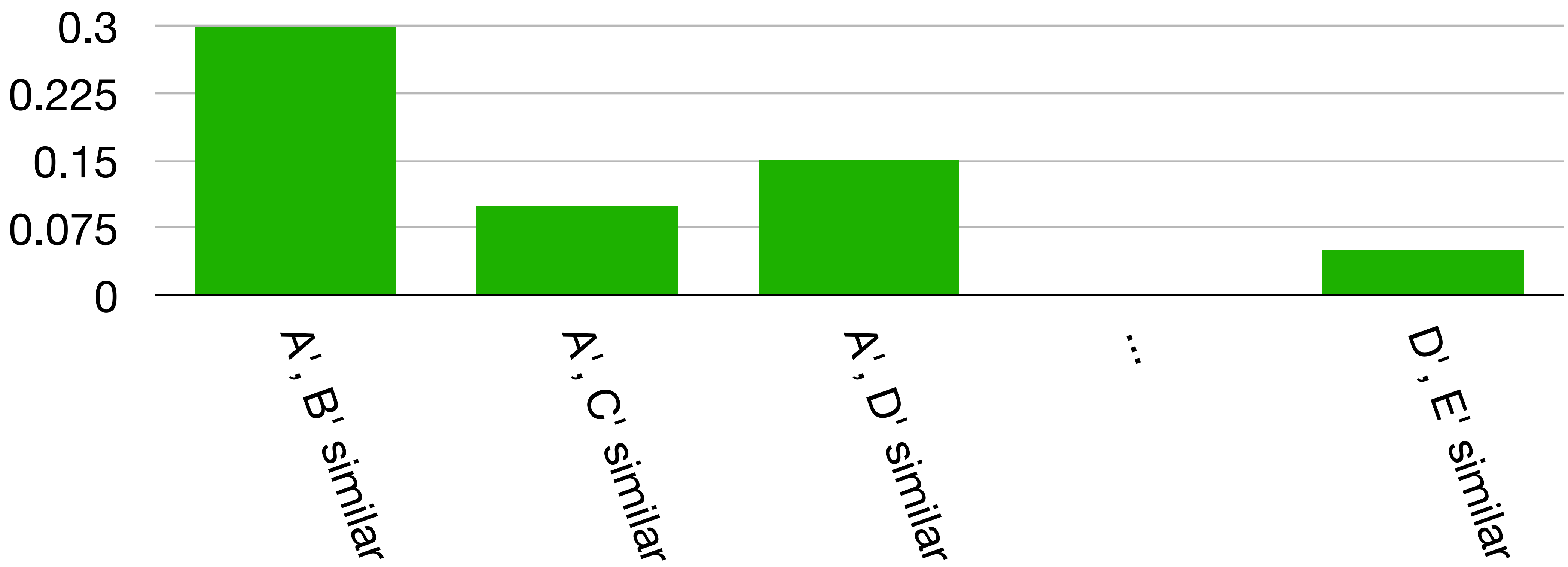
- Don't use deterministic definition of which points are neighbors
- Use probabilistic notation instead



t-SNE High-Level Idea #2

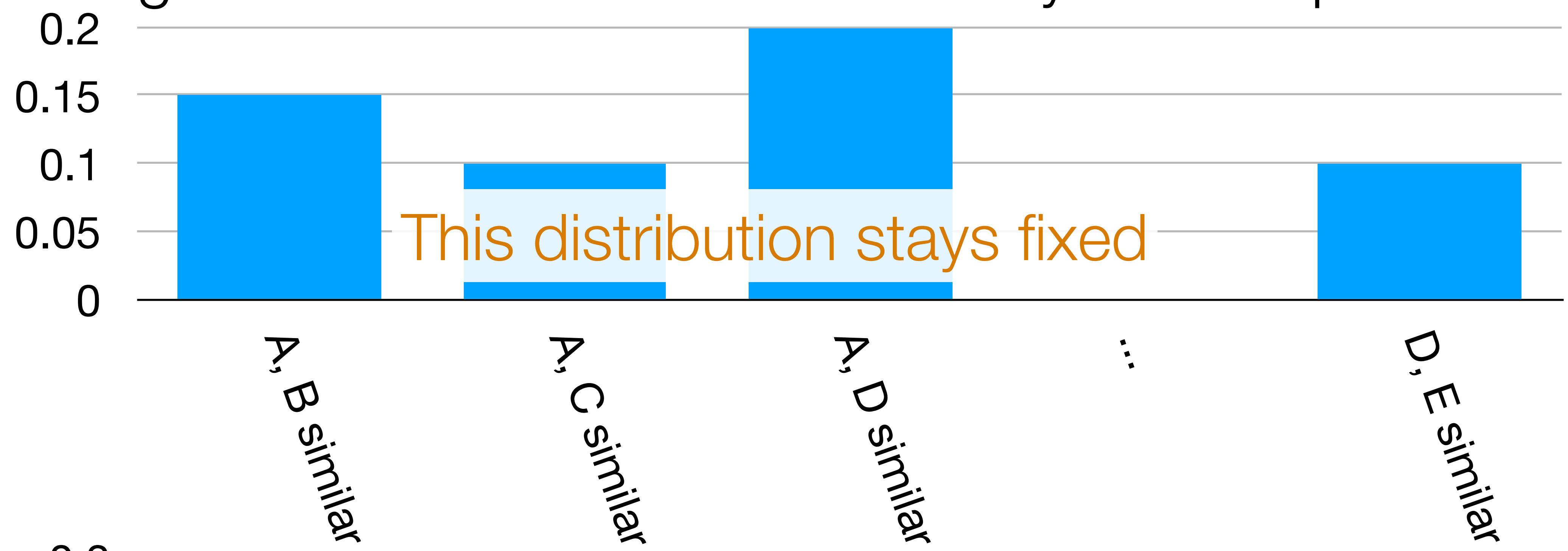
- In low-dim. space (e.g., 1D), suppose we just randomly assigned coordinates as a candidate for a low-dimensional representation for A, B, C, D, E (I'll denote them with primes):

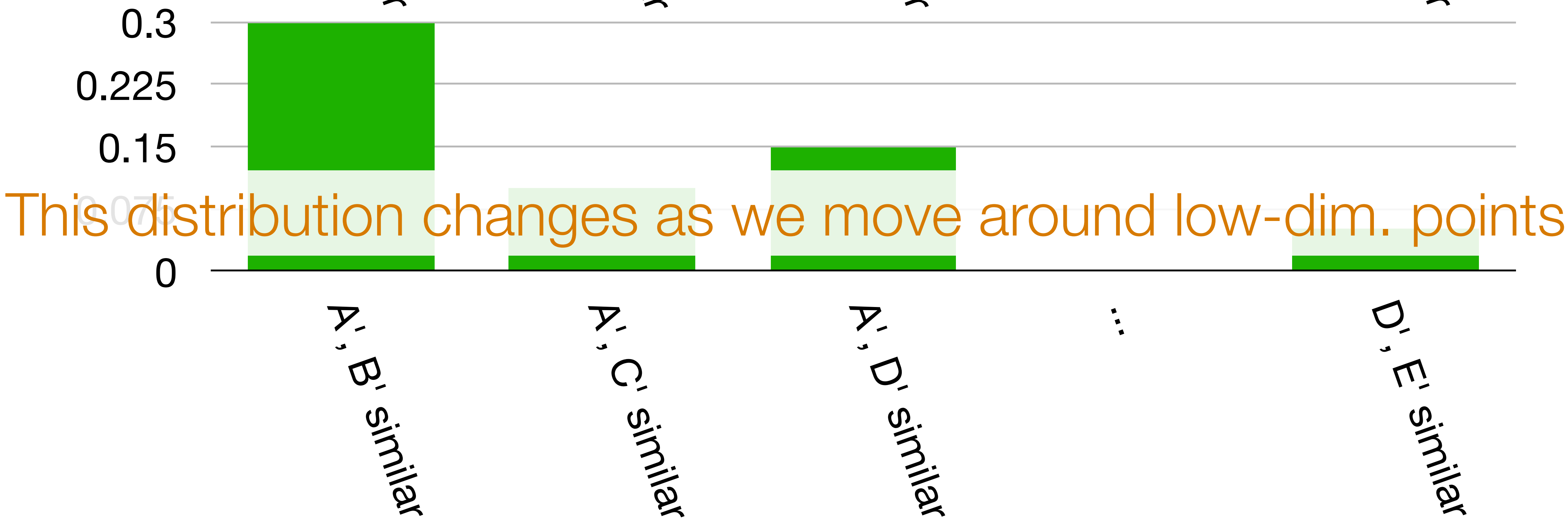
- With any such candidate choice, we can define a probability distribution for these <u>low-dimensional</u> points being similar



t-SNE High-Level Idea #3

- Keep improving low-dimensional representation to make the following two distributions look as closely alike as possible





t-SNE

Technical detail: creates probabilities
based on Gaussian distribution

Original high-dim. data → → Probability table
(for high-dim. points)

Make these two as
close as possible

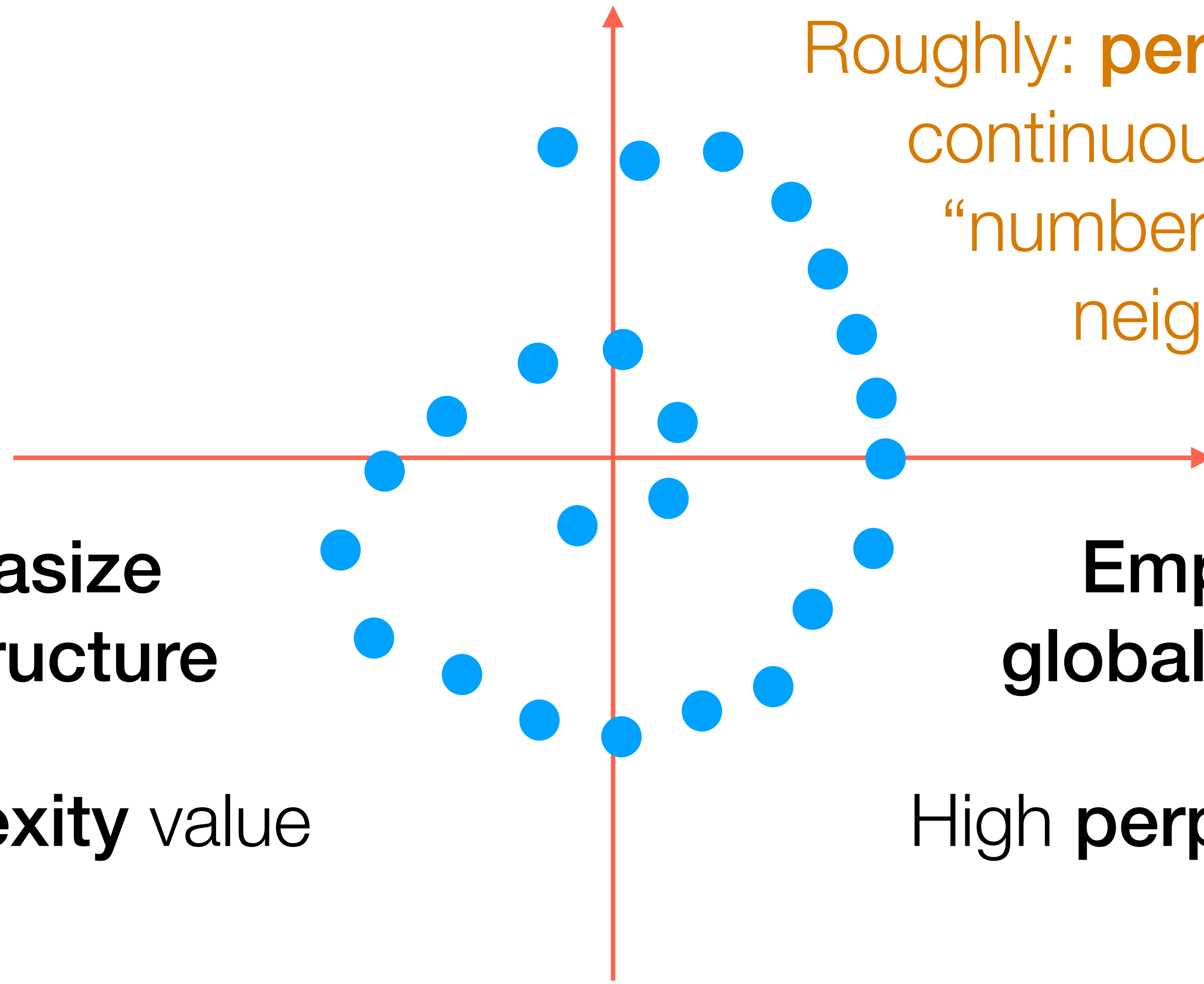
(Technical detail:
KL divergence)

Low-dim. data → → Probability table
(for low-dim. points)

Technical detail: creates probabilities
based on Student's *t*-distribution

Technical details are in separate slides (posted on webpage)

t-SNE



Roughly: **perplexity** is like a continuous version of "number of nearest neighbors"

Emphasize local structure

Low **perplexity** value

Emphasize global structure

High **perplexity** value

Also: play with learning rate, # iterations

In practice, often people initialize with PCA

Manifold Learning with t-SNE

Demo

t-SNE Interpretation

https://distill.pub/2016/misread-tsne/

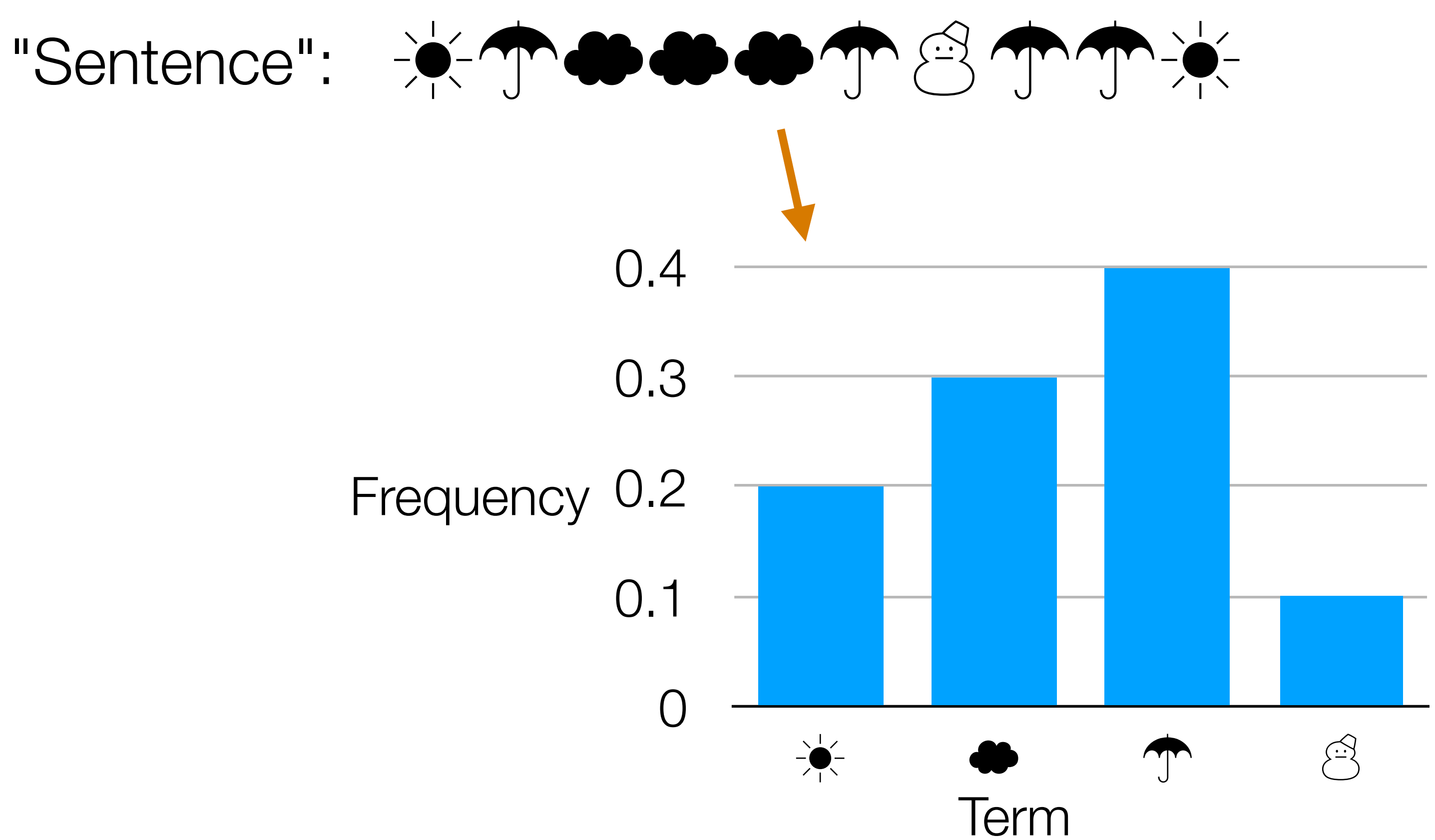
Dimensionality Reduction for Visualization

- There are *many* methods (I've posted a link on the course webpage to a scikit-learn example using ~10 methods)
- PCA is very well-understood; the new axes can be interpreted
- Nonlinear dimensionality reduction: new axes may not really be all that interpretable (you can scale axes, shift all points, etc)
- PCA and t-SNE are good candidates for methods to try first
- If you have good reason to believe that only certain features matter, of course you could restrict your analysis to those!

Let's look at images

(Flashback) Recap: Basic Text Analysis

- Represent text in terms of "features"
  (such as how often each word/phrase appears)
  - Can repeat this for different documents:
    represent each document as a "feature vector"



$$\begin{bmatrix} 0.2 \\ 0.3 \\ 0.4 \\ 0.1 \end{bmatrix}$$

This is a point in 4-dimensional space, $\mathbb{R}^4$

dimensions = number of terms

In general (not just text): first represent data as feature vectors

Example: Representing an Image

0: black
1: white

$$\begin{bmatrix} 0 \\ \\ \\ \end{bmatrix}$$

Go row by row and look at pixel values

Image source: *The Mandalorian*

Example: Representing an Image

0: black
1: white

$$\begin{bmatrix} 0 \\ 0 \\ \vdots \\ \\ \\ \end{bmatrix}$$

Go row by row and look at pixel values

Image source: *The Mandalorian*

Example: Representing an Image

0: black
1: white

$$\begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0.9 \\ \vdots \\ \\ \\ \\ \end{bmatrix}$$

Go row by row and look at pixel values

Image source: *The Mandalorian*

Example: Representing an Image

0: black
1: white

$$\begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0.9 \\ \vdots \\ \\ 0.1 \end{bmatrix}$$

Go row by row and look at pixel values

dimensions = image width × image height

Very high dimensional!

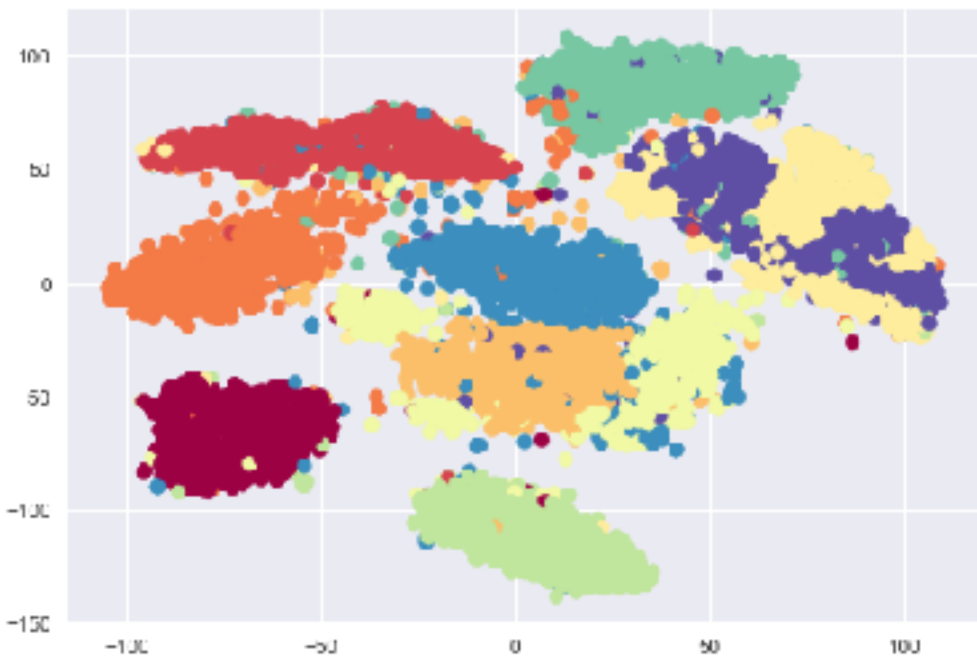
Image source: *The Mandalorian*

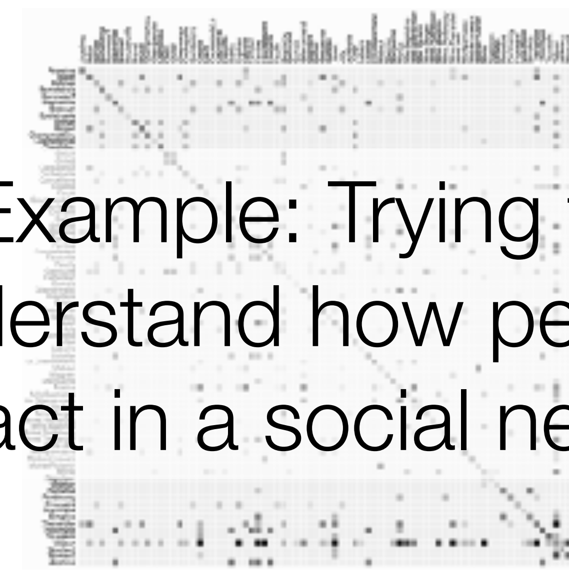
Dimensionality Reduction for Images

Demo

Visualization

is a way of debugging data analysis!





Example: Trying to understand how people interact in a social network

Important:
Handwritten digit demo is a **toy example** where we know which images correspond to digits 0, 1, …, 9

Many real UDA problems:
The data are **messy** and it's not obvious what the "correct" labels/answers look like, and "correct" is ambiguous!

Later on in the course (when we cover predictive analytics), we look at how to take advantage of knowing the true "correct" answers

Top right image source: https://bost.ocks.org/mike/miserables/

Let's look at a *structured* dataset
(easier to explain clustering):
drug consumption data

Drug Consumption Data

Demo

Intermission

Unstructured Data Analysis

Lecture 7: Distance and similarity functions, clustering

George Chen

Clustering Shows Up Often in Real Data!

- Example: crime might happen more often in specific hot spots

- Example: people applying for micro loans have a few specific uses in mind (education, electricity, healthcare, etc)

- Example: users in a recommendation system can share similar taste in products

To come up with clusters, we first need to define what it means for two things to be “similar”

The Art of Defining Similarity

- Popular: define a distance first and then turn it into a similarity

Example: Euclidean distance $\|X_i - X_j\|$

Turn into similarity with decaying exponential $\downarrow$

$$\exp(-\gamma\|X_i - X_j\|^2)$$

where $\gamma > 0$

- There is no “best” distance function to use

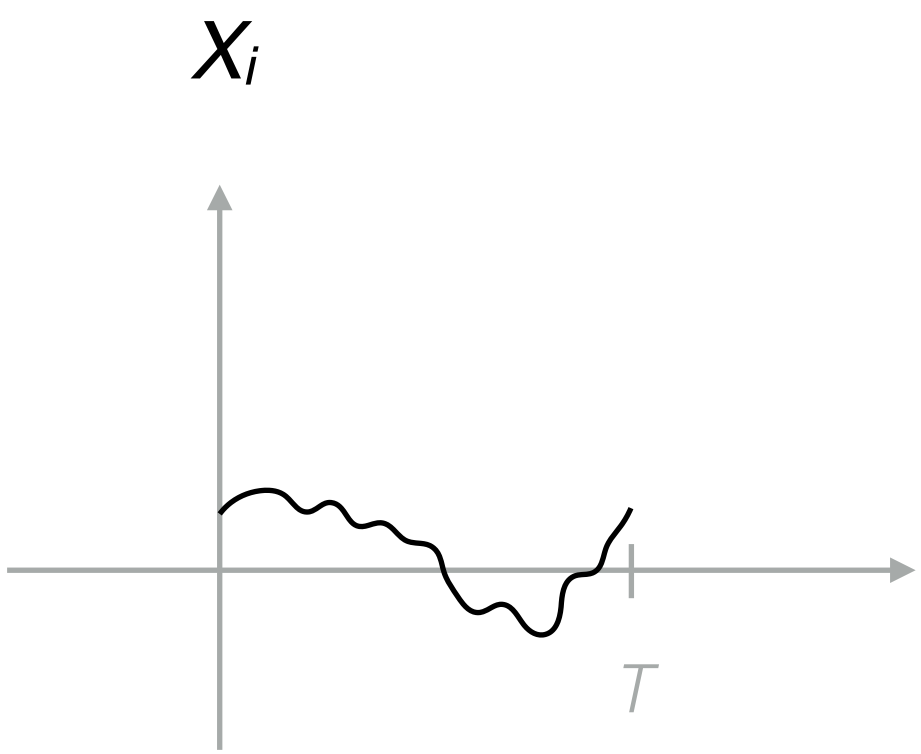
- Can also directly define similarity function

Example: cosine similarity $\dfrac{\langle X_i, X_j \rangle}{\|X_i\|\|X_j\|}$

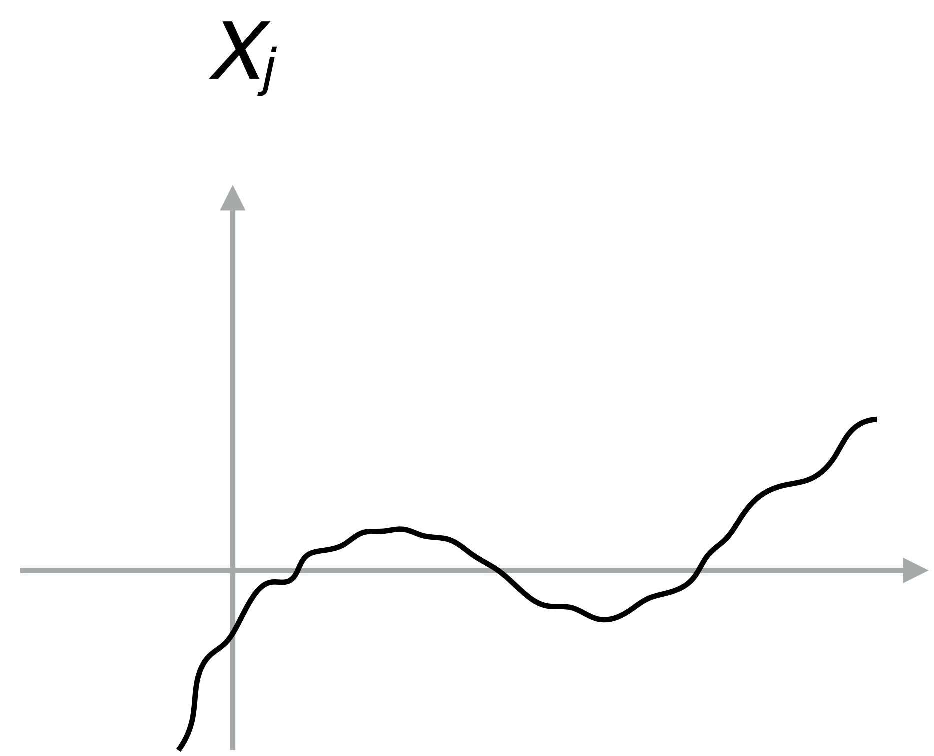
There exist methods for automatically learning distance or similarity functions

Example: Time Series

How would you compute a distance between these?

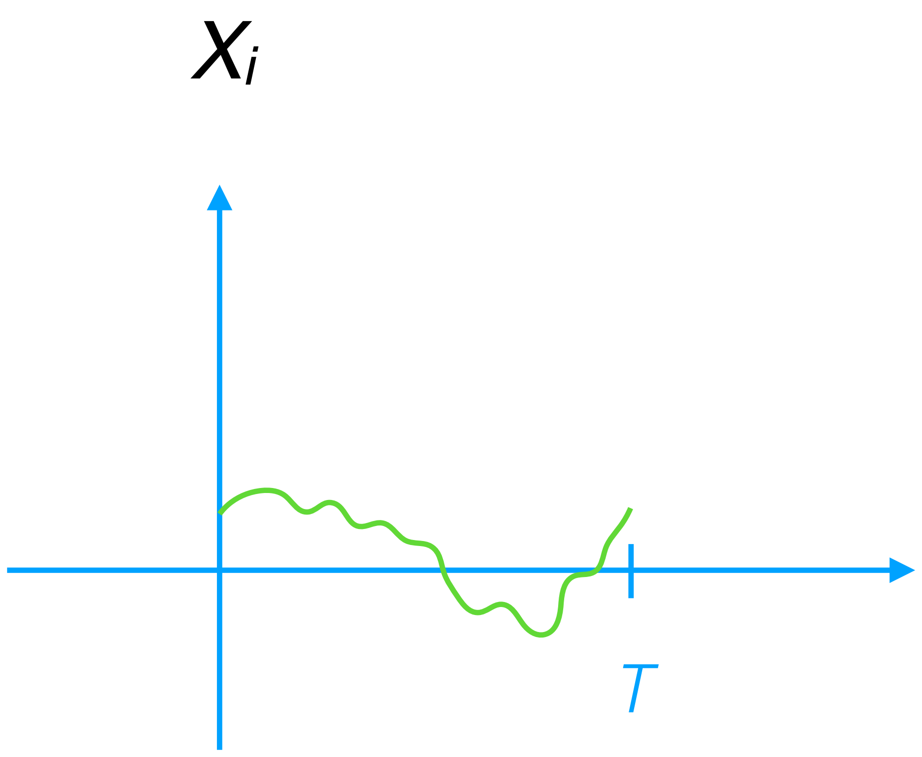


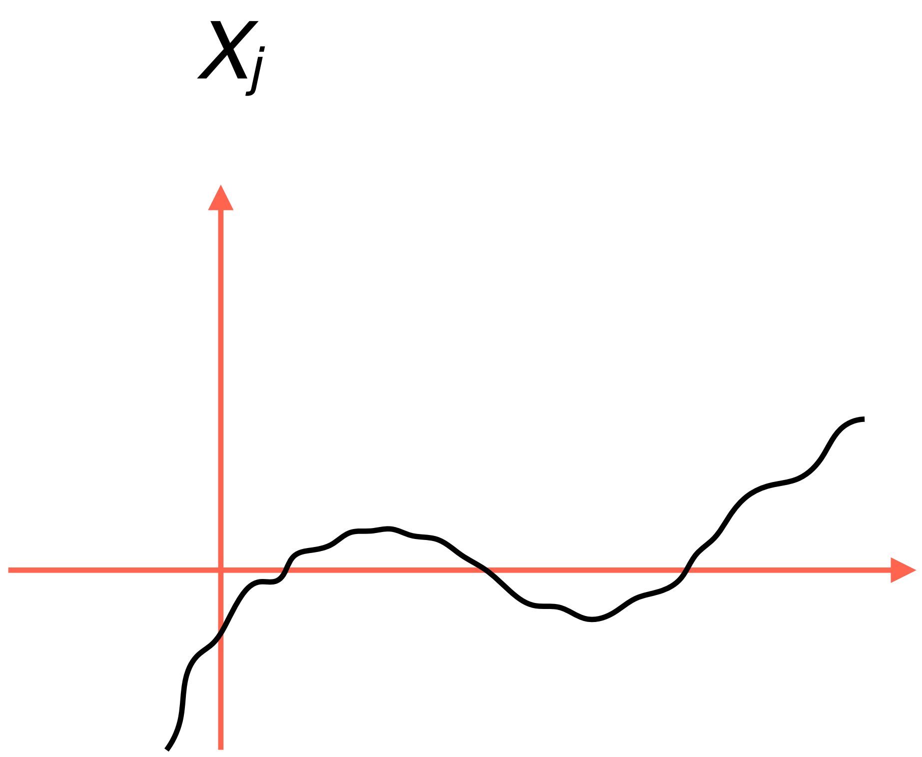
Only observe time steps
between 0 and T



Example: Time Series

How would you compute a distance between these?

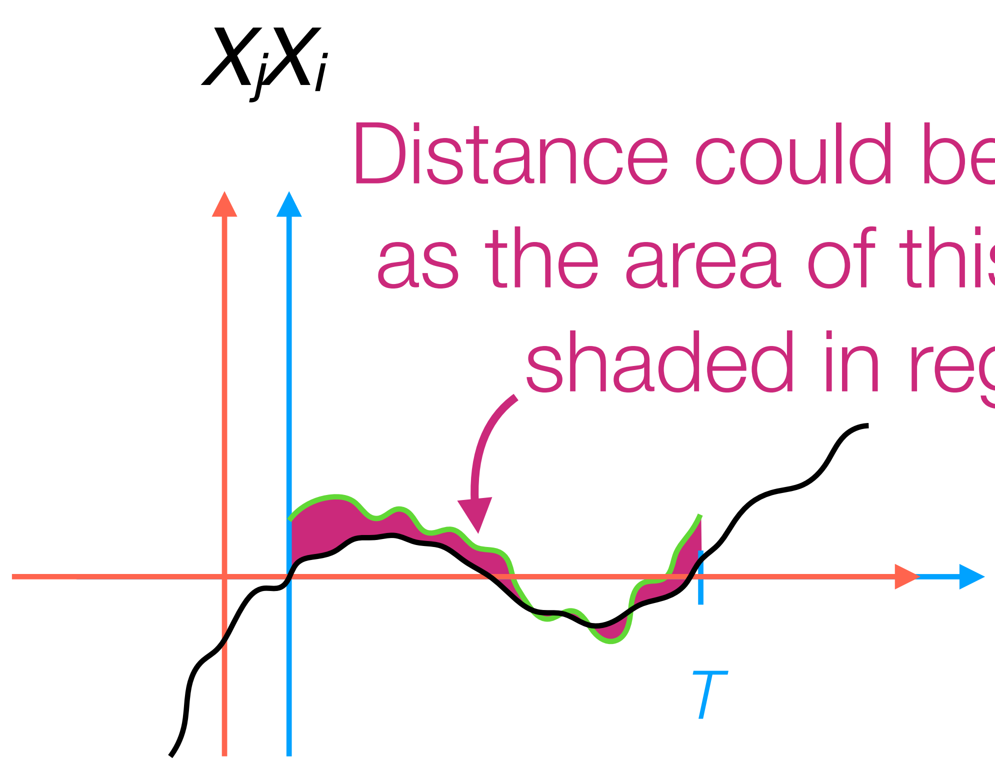




Only observe time steps between 0 and T

Example: Time Series

How would you compute a distance between these?



One solution: Align them first

In practice: for time series, very popular to use "dynamic time warping" (aligns two time series in a nonlinear manner)

Dynamic Time Warping aims to align time series into some common coordinate system

Then in the common coordinate system, can use usual distance functions like Euclidean, Manhattan, etc

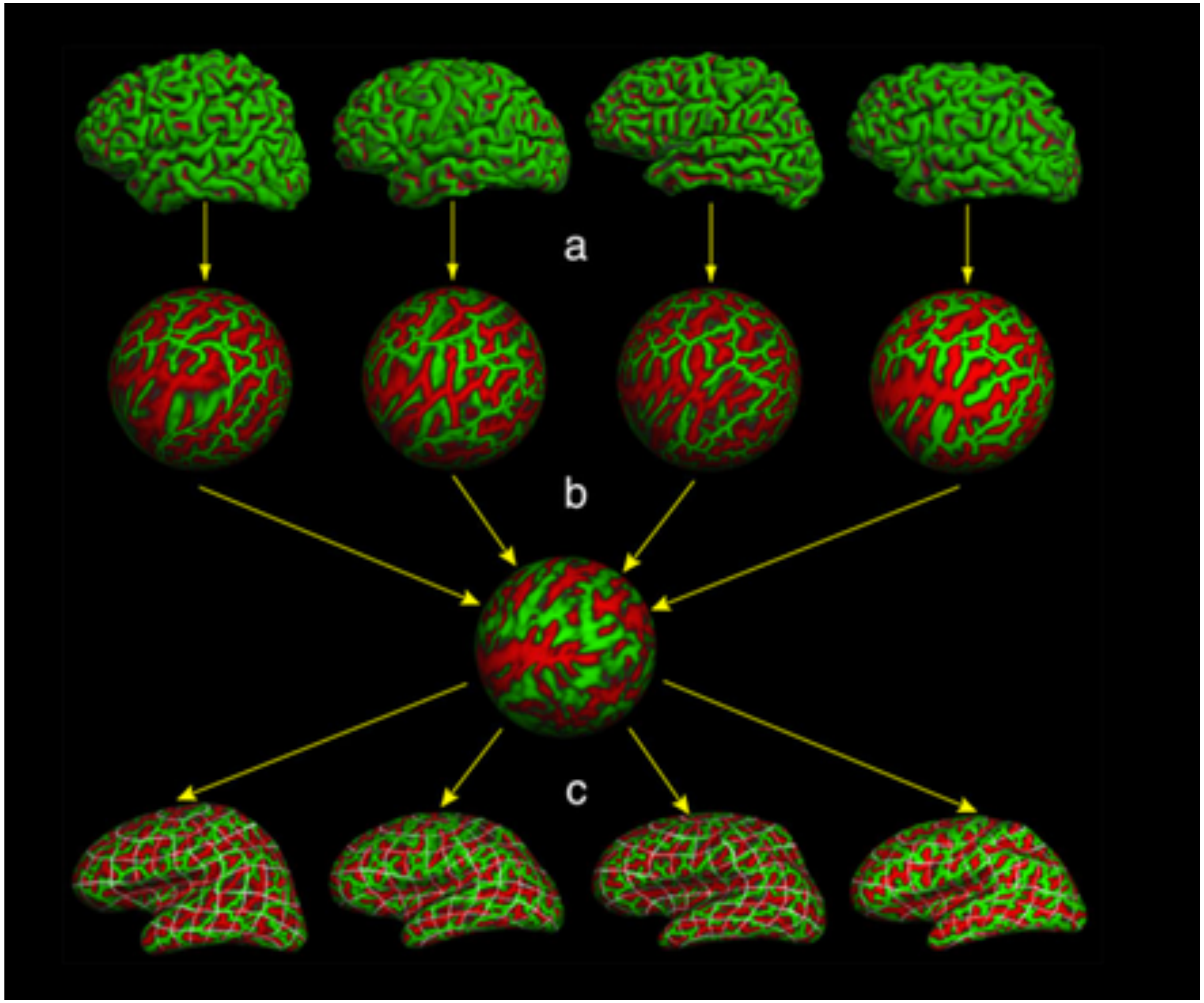
“Aligning” data points is important in other problems too, not just for time series analysis

Example: Spell Check

Distance between “apple” and “ap;ple”?

One way to compute: find minimum number of single-letter insertions/deletions/substitutions to convert one to the other
(called the Levenshtein distance)

Brain Image “Alignment”



FreeSurfer software: convert different people’s brain scans into spherical coordinates for comparison

Is a Distance/Similarity Function Any Good?

Easy thing to try:

- Pick a data point (for example, randomly)
- Compute its similarity to all the other data points, and sort them from most similar to least similar (or smallest distance to largest)
- Manually examine the most similar (closest) data points

If the most similar/closest points are not interpretable, it's quite likely that your distance/similarity function isn't very good =(

Clustering methods aim to group together data points that are "similar" into "clusters", while having different clusters be "dissimilar"

Clustering methods will either directly assume a specific choice of distance/similarity function, or some allow you to specify the distance/similarity

Going from Similarities to Clusters

There's a whole zoo of clustering methods

Several main categories (although there are other categories!):

Generative models

1. Pretend data generated by specific model with parameters

2. Learn the parameters ("fit model to data")

3. Use fitted model to determine cluster assignments

We mainly focus on this

Hierarchical clustering

Top-down: Start with everything in 1 cluster and decide on how to recursively split

Bottom-up: Start with everything in its own cluster and decide on how to iteratively merge clusters

Density-based clustering

Based on finding parts of the data with higher density

We're going to start with perhaps the most famous of clustering methods

It won't yet be apparent what this method has to do with generative models

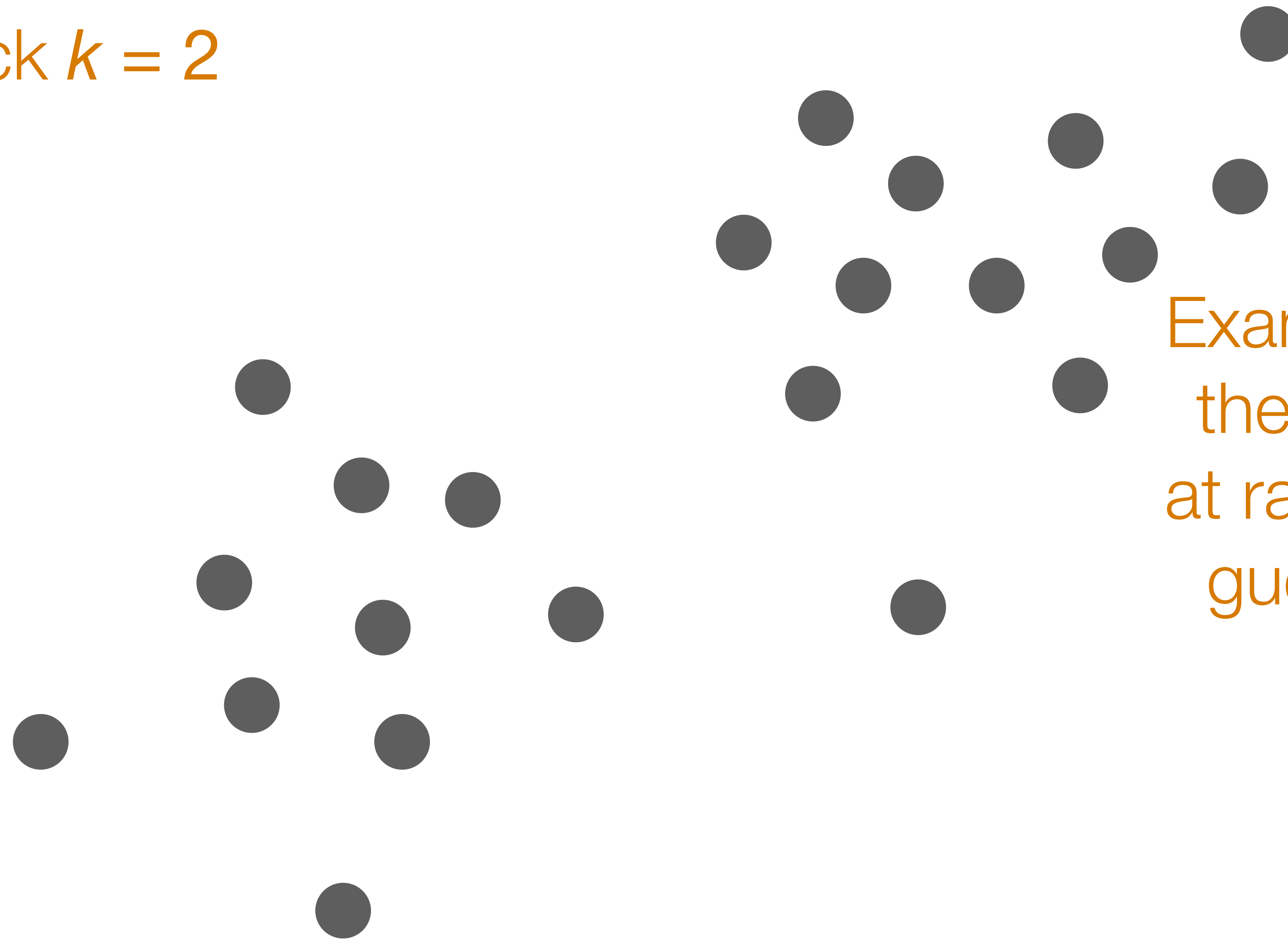
k-means

Step 0: Pick *k*

We'll pick $k = 2$

Step 1: Pick guesses for where cluster centers are

Example: choose *k* of the points uniformly at random to be initial guesses for cluster centers

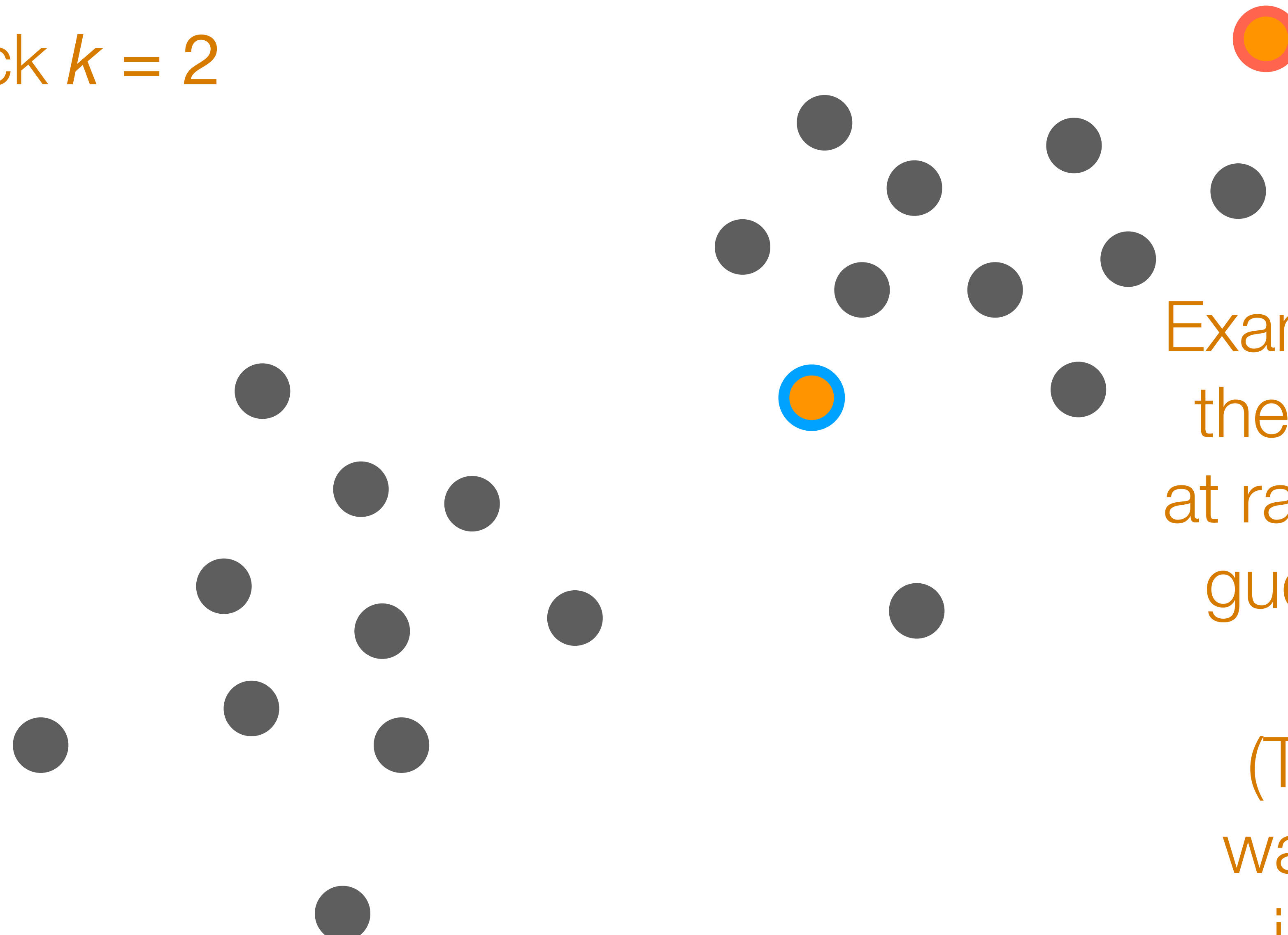


k-means

Step 0: Pick k

We'll pick $k = 2$

Step 1: Pick <u>guesses</u> for where cluster centers are



Example: choose k of the points uniformly at random to be initial guesses for cluster centers

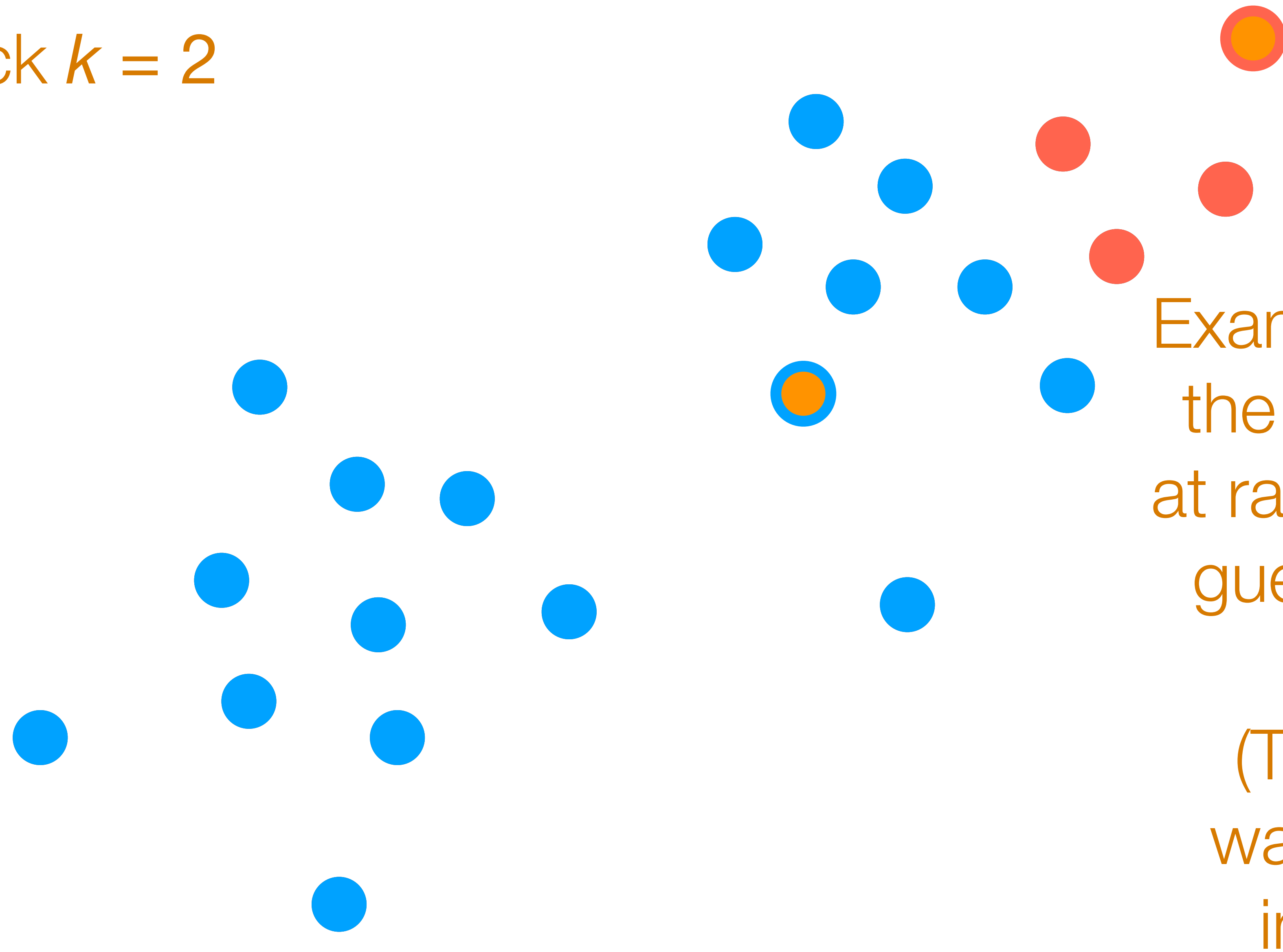
(There are many ways to make the initial guesses)

k-means

Step 0: Pick *k*

We'll pick $k = 2$

Step 1: Pick <u>guesses</u> for where cluster centers are



Example: choose *k* of the points uniformly at random to be initial guesses for cluster centers

(There are many ways to make the initial guesses)

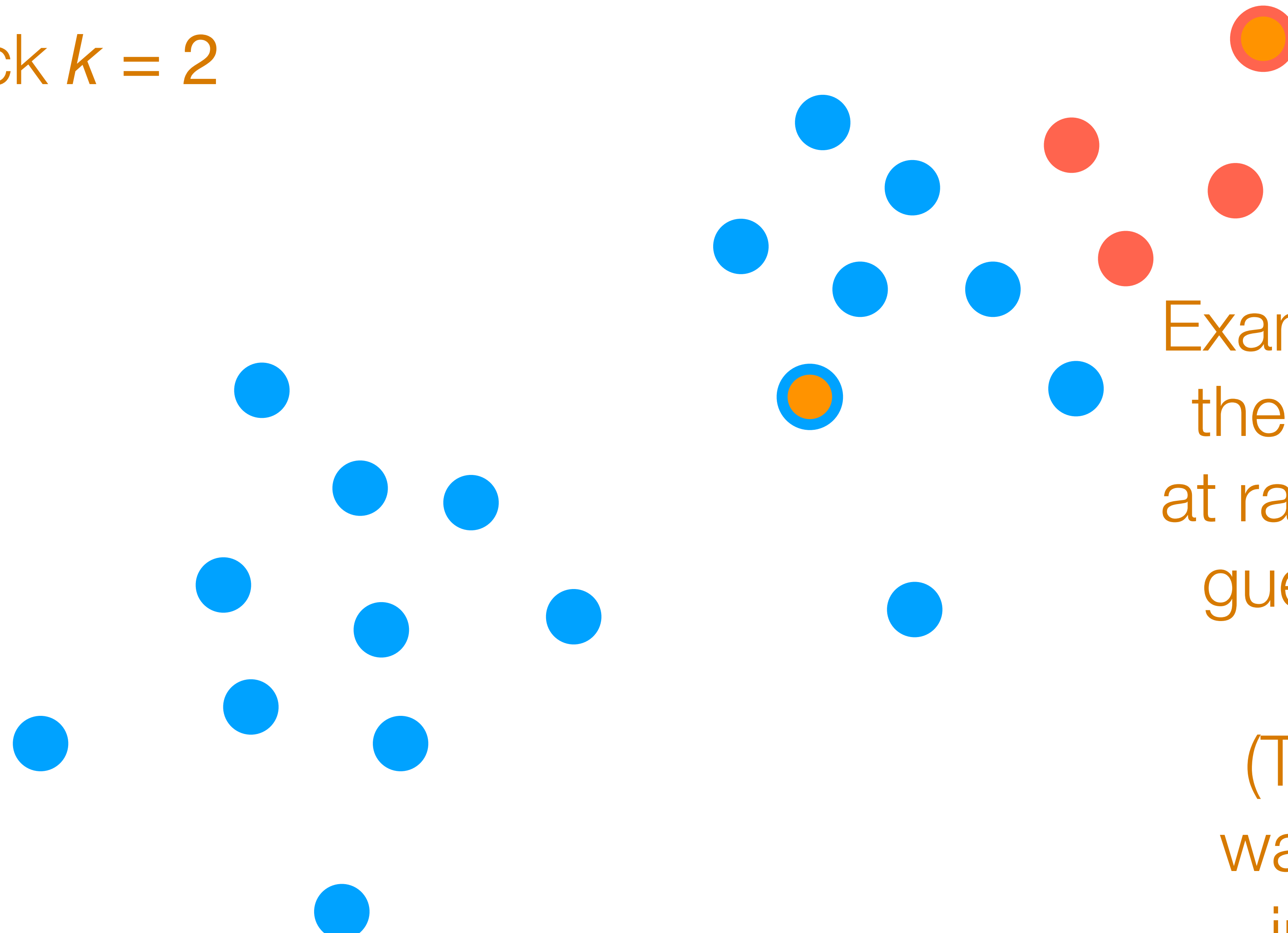
Step 2: Assign each point to belong to the closest cluster

k-means

Step 0: Pick *k*

We'll pick *k* = 2

Step 1: Pick <u>guesses</u> for where cluster centers are



Example: choose *k* of the points uniformly at random to be initial guesses for cluster centers

(There are many ways to make the initial guesses)

Step 2: Assign each point to belong to the closest cluster

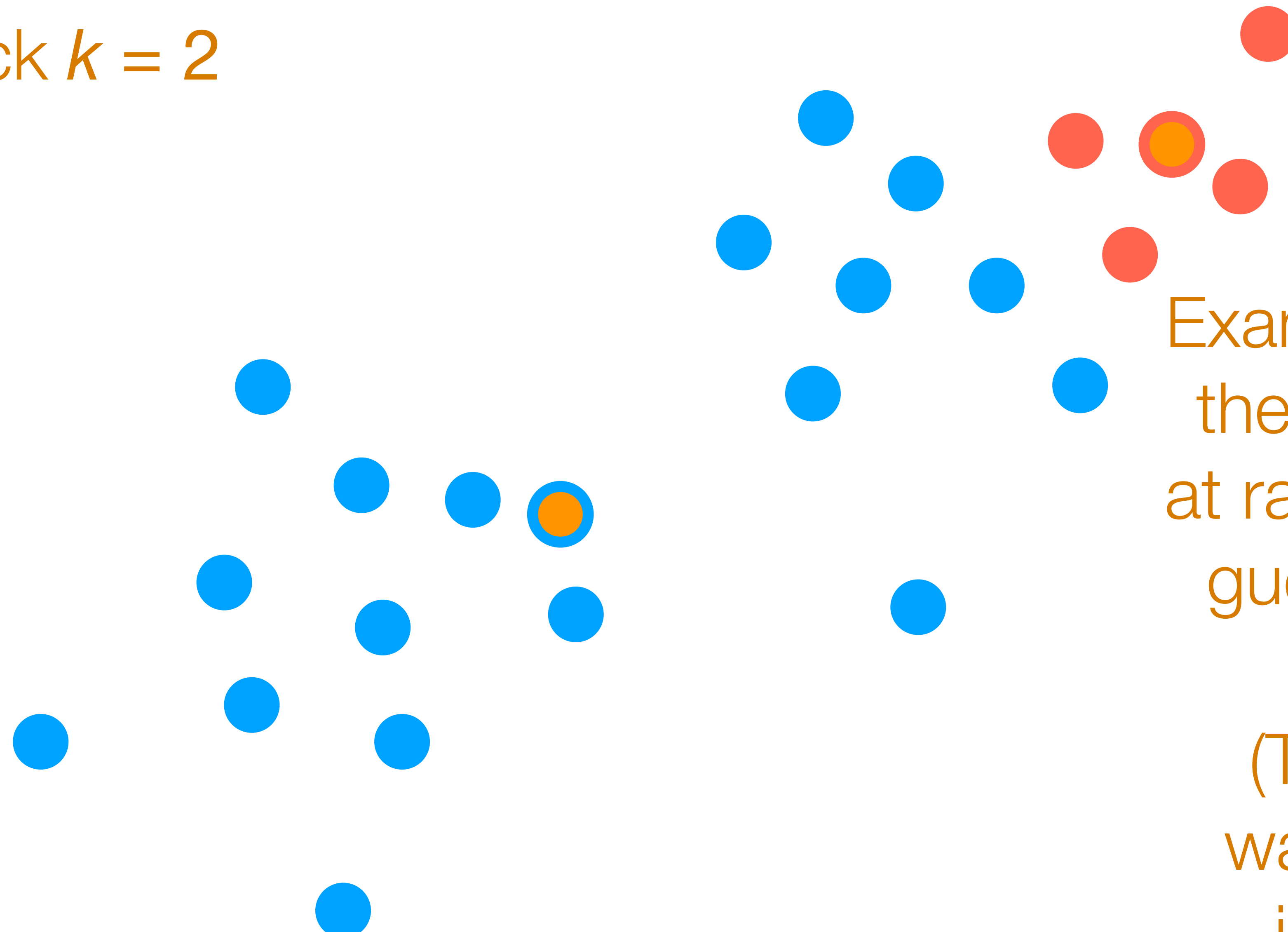
Step 3: Update cluster means (to be the center of mass per cluster)

k-means

Step 0: Pick *k*

We'll pick *k* = 2

Step 1: Pick <u>guesses</u> for where cluster centers are



Example: choose *k* of the points uniformly at random to be initial guesses for cluster centers

(There are many ways to make the initial guesses)

Step 2: Assign each point to belong to the closest cluster

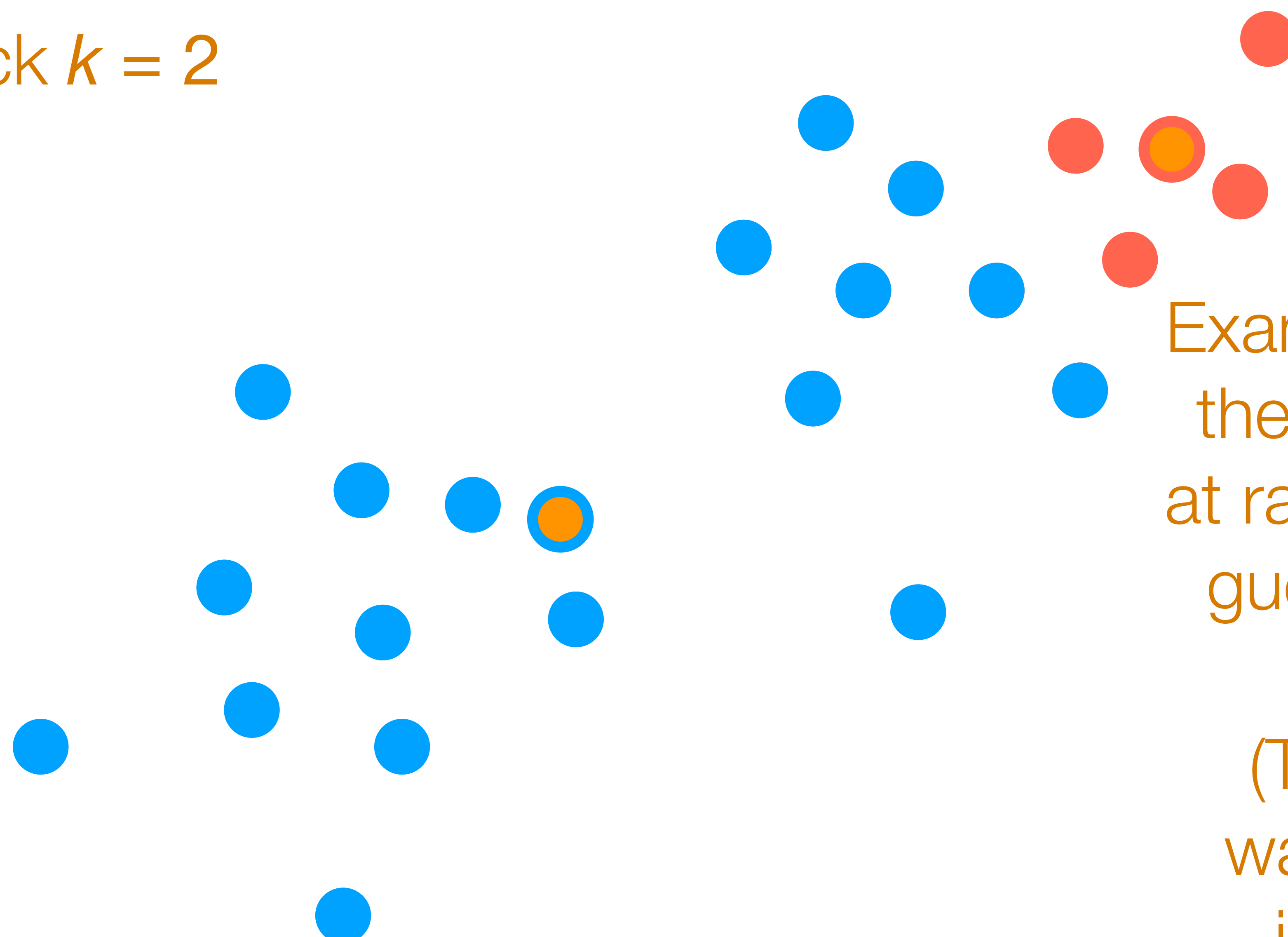
Step 3: Update cluster means (to be the center of mass per cluster)

k-means

Step 0: Pick *k*

We'll pick *k* = 2

Step 1: Pick guesses for where cluster centers are



Example: choose *k* of the points uniformly at random to be initial guesses for cluster centers

(There are many ways to make the initial guesses)

Repeat Step 2: Assign each point to belong to the closest cluster

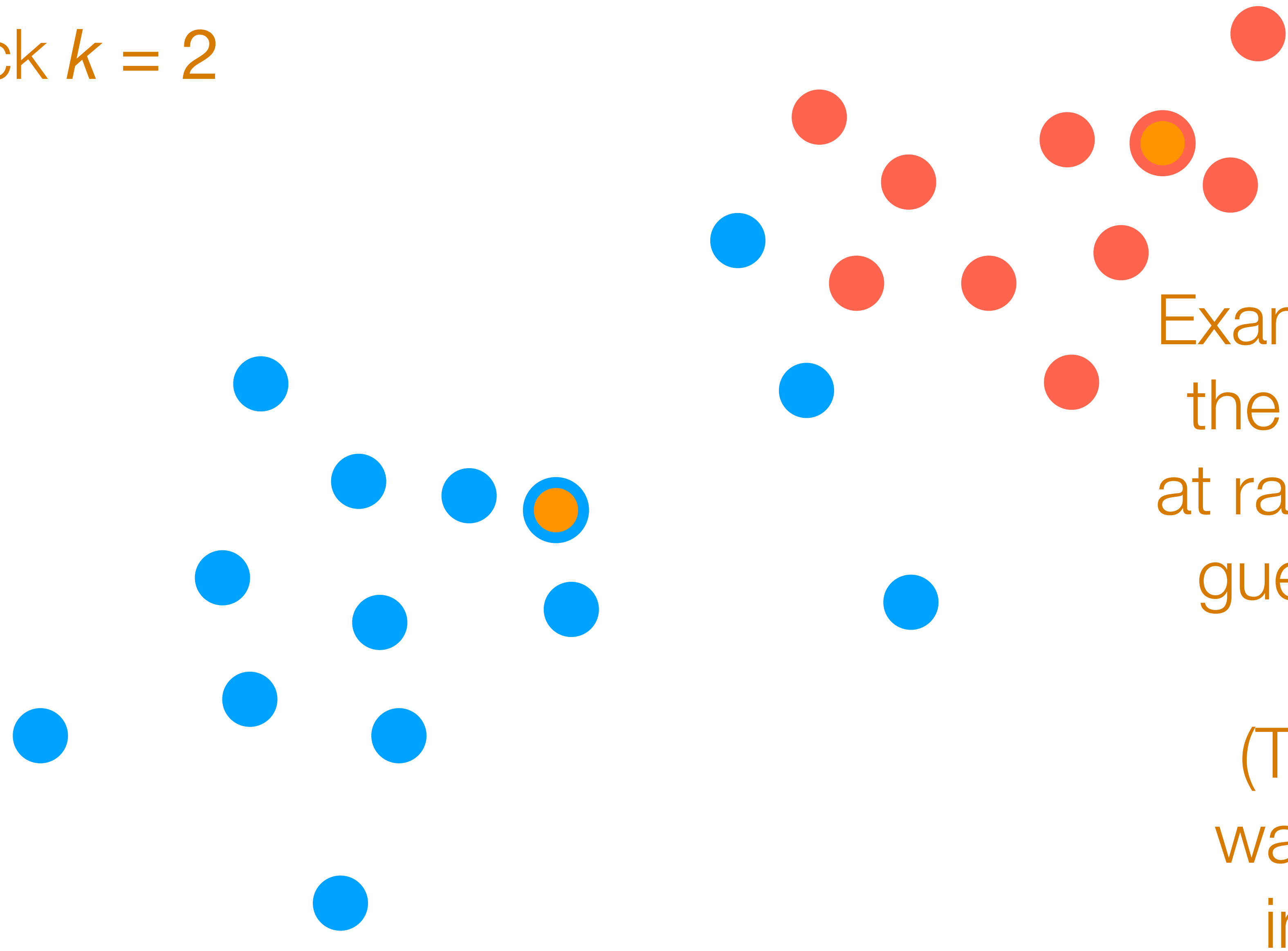
Step 3: Update cluster means (to be the center of mass per cluster)

k-means

Step 0: Pick *k*

We'll pick *k* = 2

Step 1: Pick <u>guesses</u> for where cluster centers are



Example: choose *k* of the points uniformly at random to be initial guesses for cluster centers

(There are many ways to make the initial guesses)

Step 2: Assign each point to belong to the closest cluster

Repeat

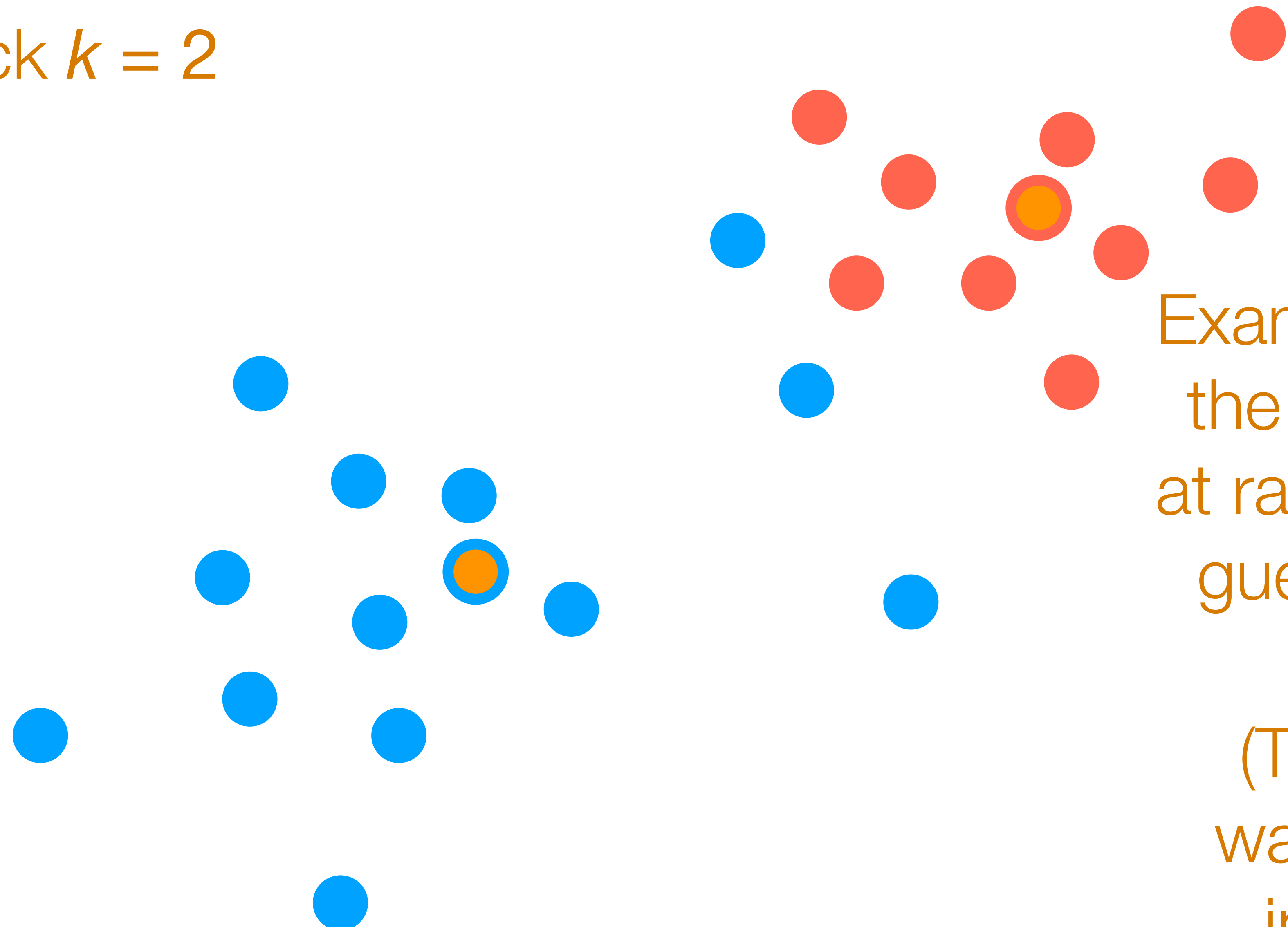
Step 3: Update cluster means (to be the center of mass per cluster)

k-means

Step 0: Pick *k*

We'll pick *k* = 2

Step 1: Pick <u>guesses</u> for where cluster centers are



Example: choose *k* of the points uniformly at random to be initial guesses for cluster centers

(There are many ways to make the initial guesses)

Step 2: Assign each point to belong to the closest cluster

Repeat

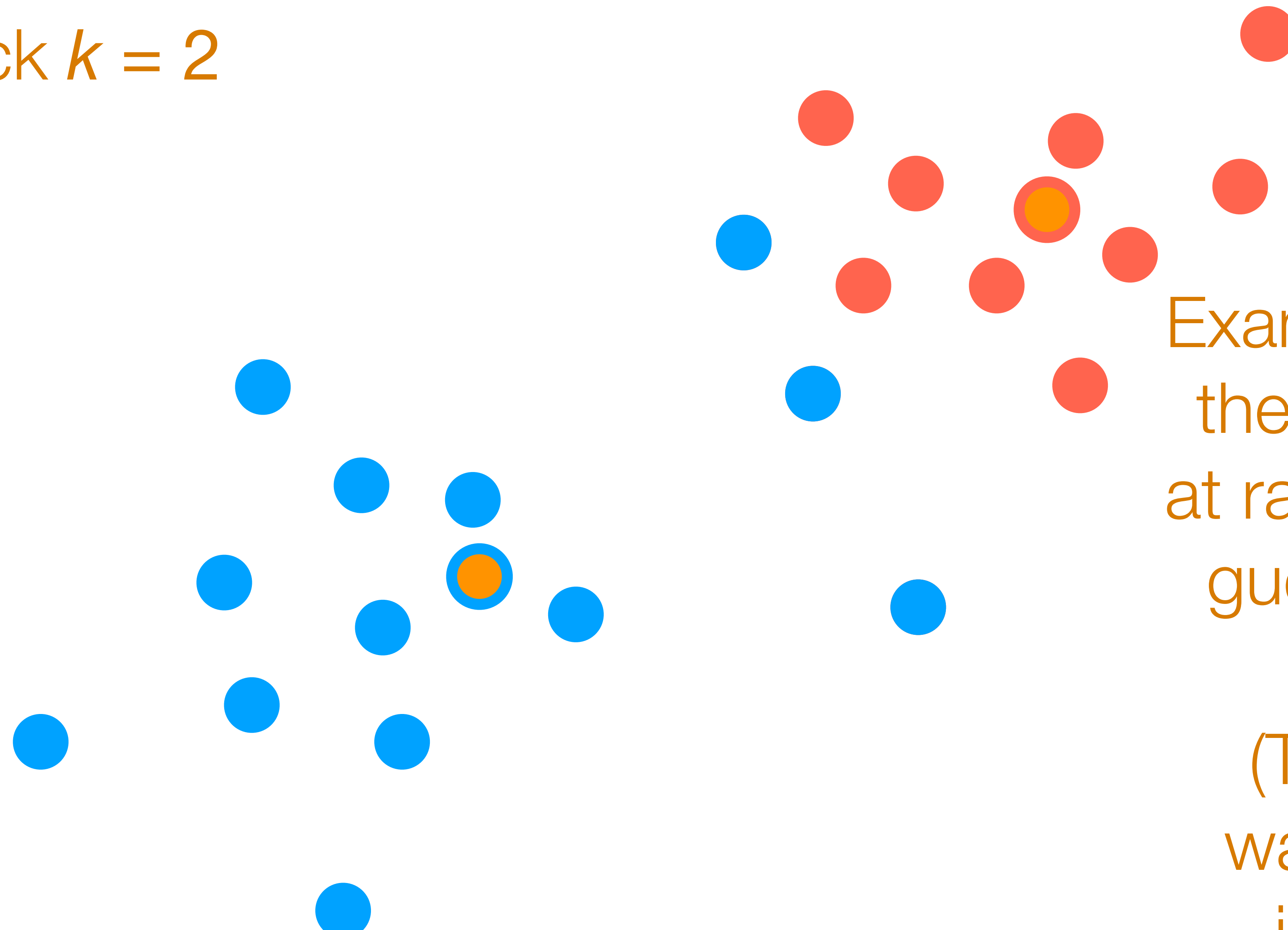
Step 3: Update cluster means (to be the center of mass per cluster)

k-means

Step 0: Pick *k*

We'll pick *k* = 2

Step 1: Pick <u>guesses</u> for where cluster centers are



Example: choose *k* of the points uniformly at random to be initial guesses for cluster centers

(There are many ways to make the initial guesses)

Repeat Step 2: Assign each point to belong to the closest cluster

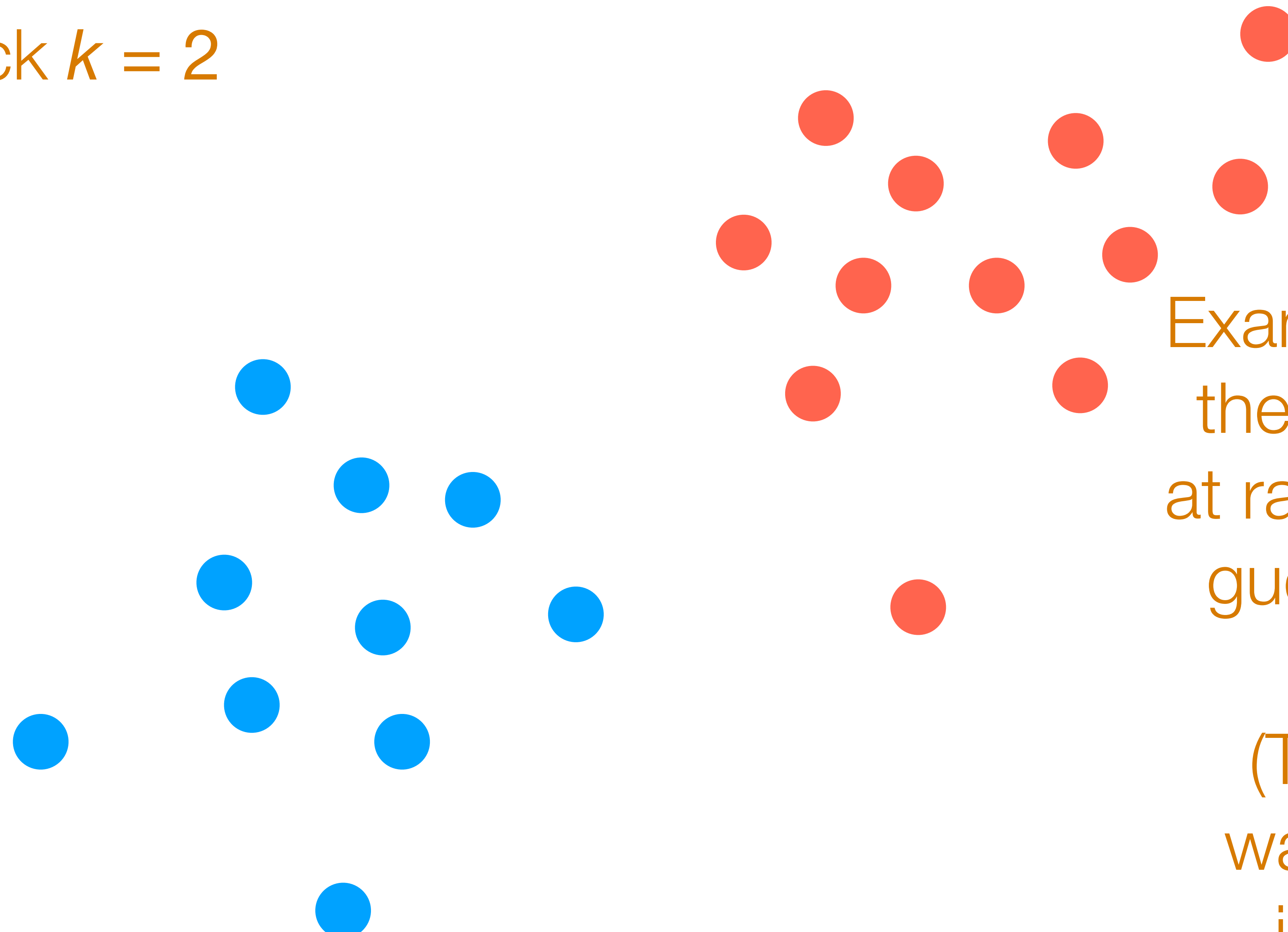
Step 3: Update cluster means (to be the center of mass per cluster)

k-means

Step 1: Pick guesses for where cluster centers are

Step 0: Pick *k*

We'll pick *k* = 2



Example: choose *k* of the points uniformly at random to be initial guesses for cluster centers

(There are many ways to make the initial guesses)

Step 2: Assign each point to belong to the closest cluster

Repeat

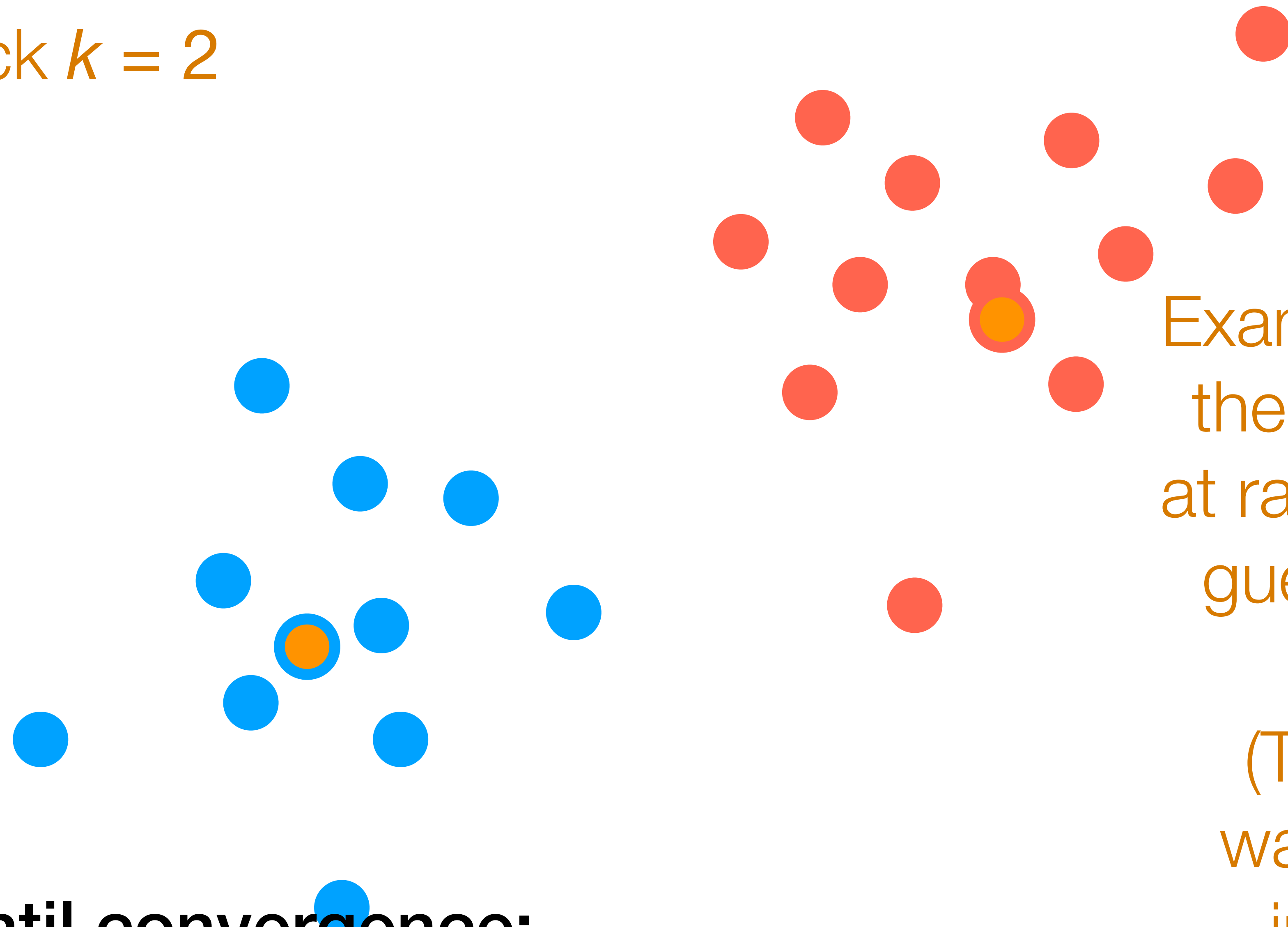
Step 3: Update cluster means (to be the center of mass per cluster)

k-means

Step 0: Pick *k*

We'll pick *k* = 2

Step 1: Pick <u>guesses</u> for where cluster centers are



Example: choose *k* of the points uniformly at random to be initial guesses for cluster centers

(There are many ways to make the initial guesses)

Repeat until convergence:

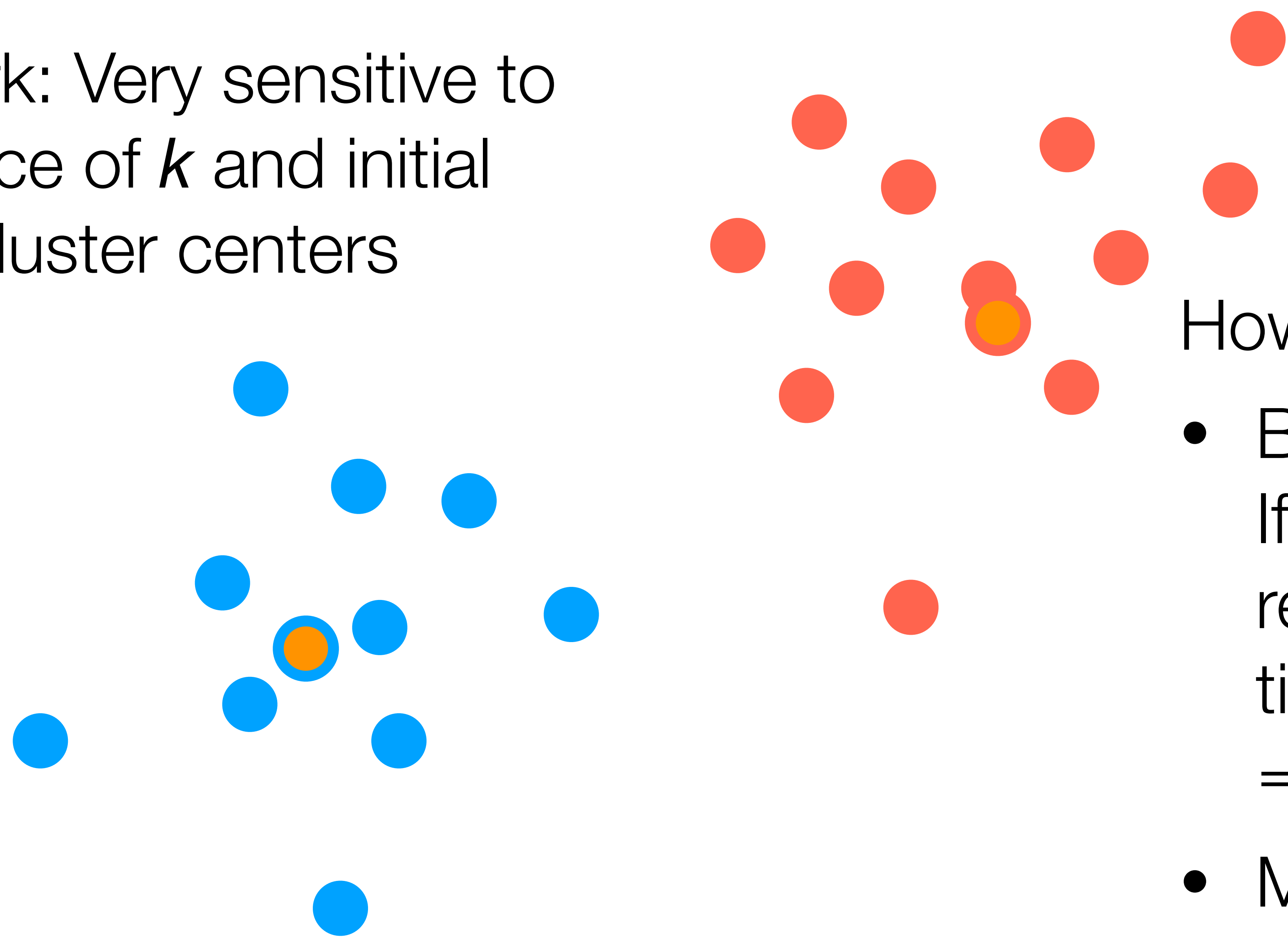
Step 2: Assign each point to belong to the closest cluster

Step 3: Update cluster means (to be the center of mass per cluster)

k-means

Final output: cluster centers, cluster assignment for every point

Remark: Very sensitive to choice of *k* and initial cluster centers



How to pick *k*?

- Basic check: If you have really, really tiny clusters $\Rightarrow$ decrease *k*
- More details later

Suggested way to pick initial cluster centers: "*k*-means++" method

(rough intuition: incrementally add centers; favor adding center far away from centers chosen so far)

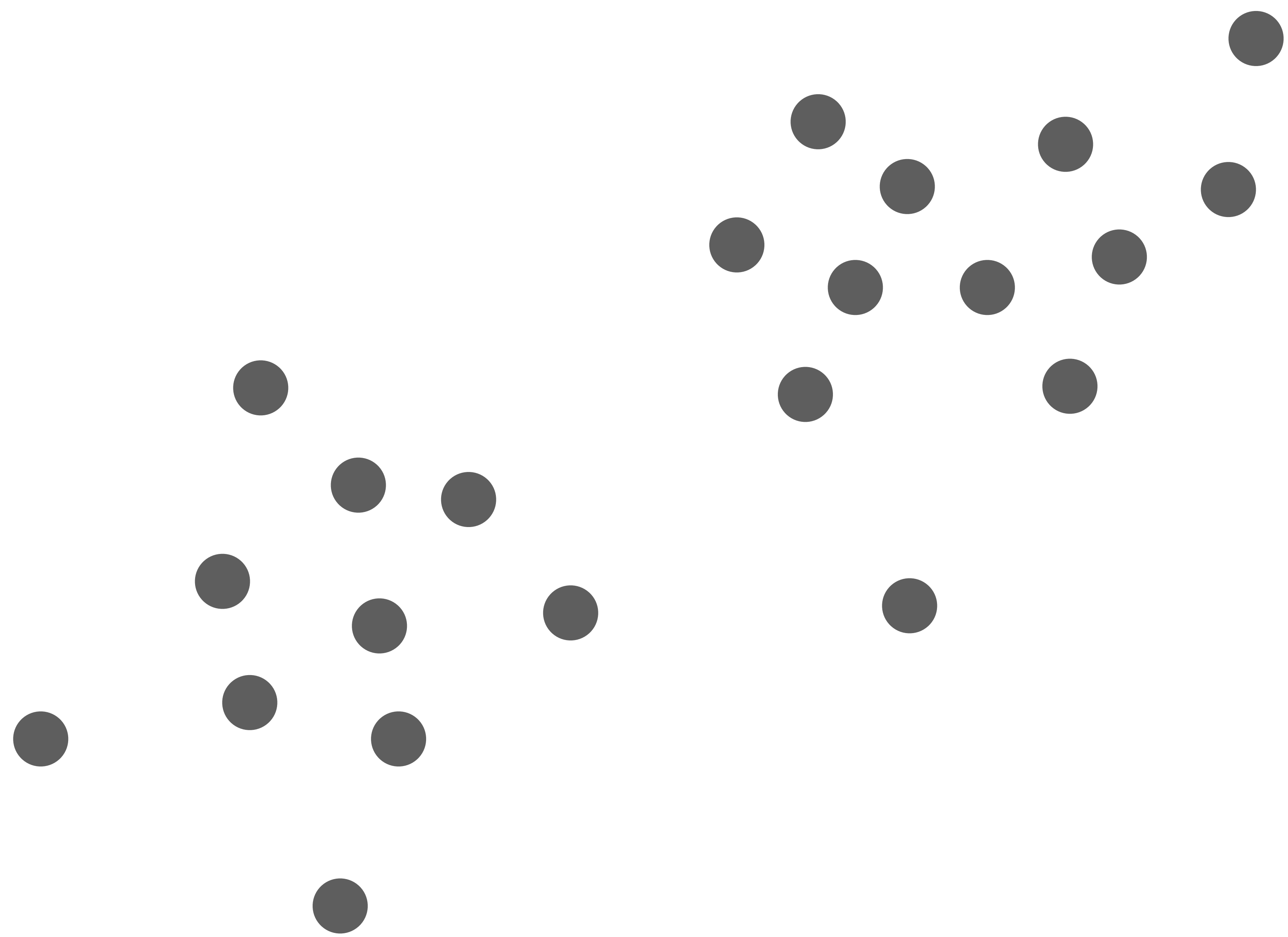
When does *k*-means work well?

k-means is related to a more general model, which will help us understand *k*-means

When does *k*-means work well?

k-means is related to a more general model, which will help us understand *k*-means

Gaussian Mixture Model (GMM)



What random process could have generated these points?

Generative Process

Think of flipping a coin

each outcome: heads or tails

Each flip doesn't depend on any of the previous flips

Generative Process

Think of flipping a coin

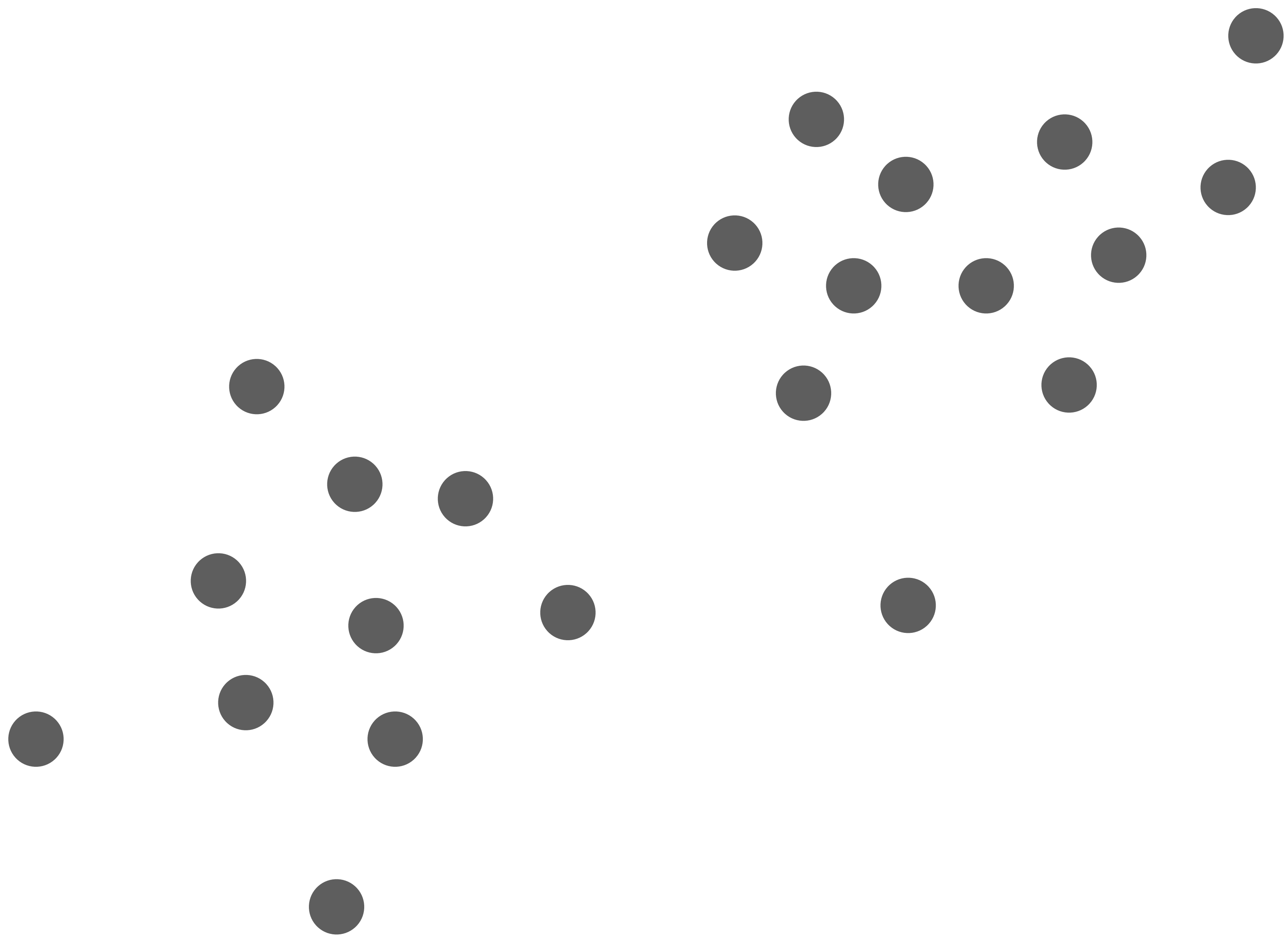
each outcome: 2D point

Each flip doesn't depend on any of the previous flips

Okay, maybe it's bizarre to think of it as a coin…

If it helps, just think of it as you pushing a button and a random 2D point appears…

Gaussian Mixture Model (GMM)



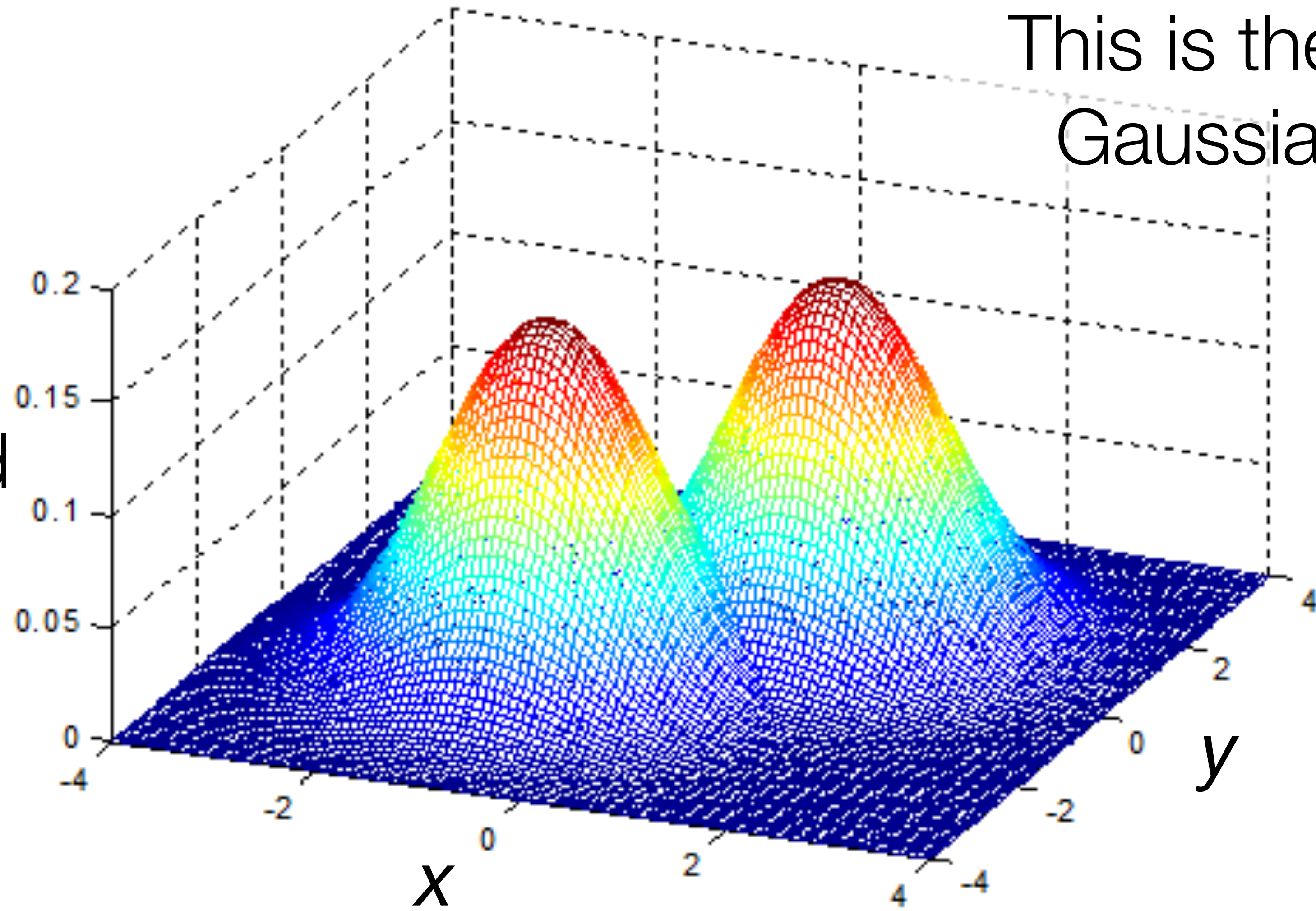
We now discuss a way to generate points in this manner

Gaussian Mixture Model (GMM)

Assume: points sampled independently from a probability distribution

This is the sum of two 2D Gaussian distributions!

how probable point generated at (x, y) is



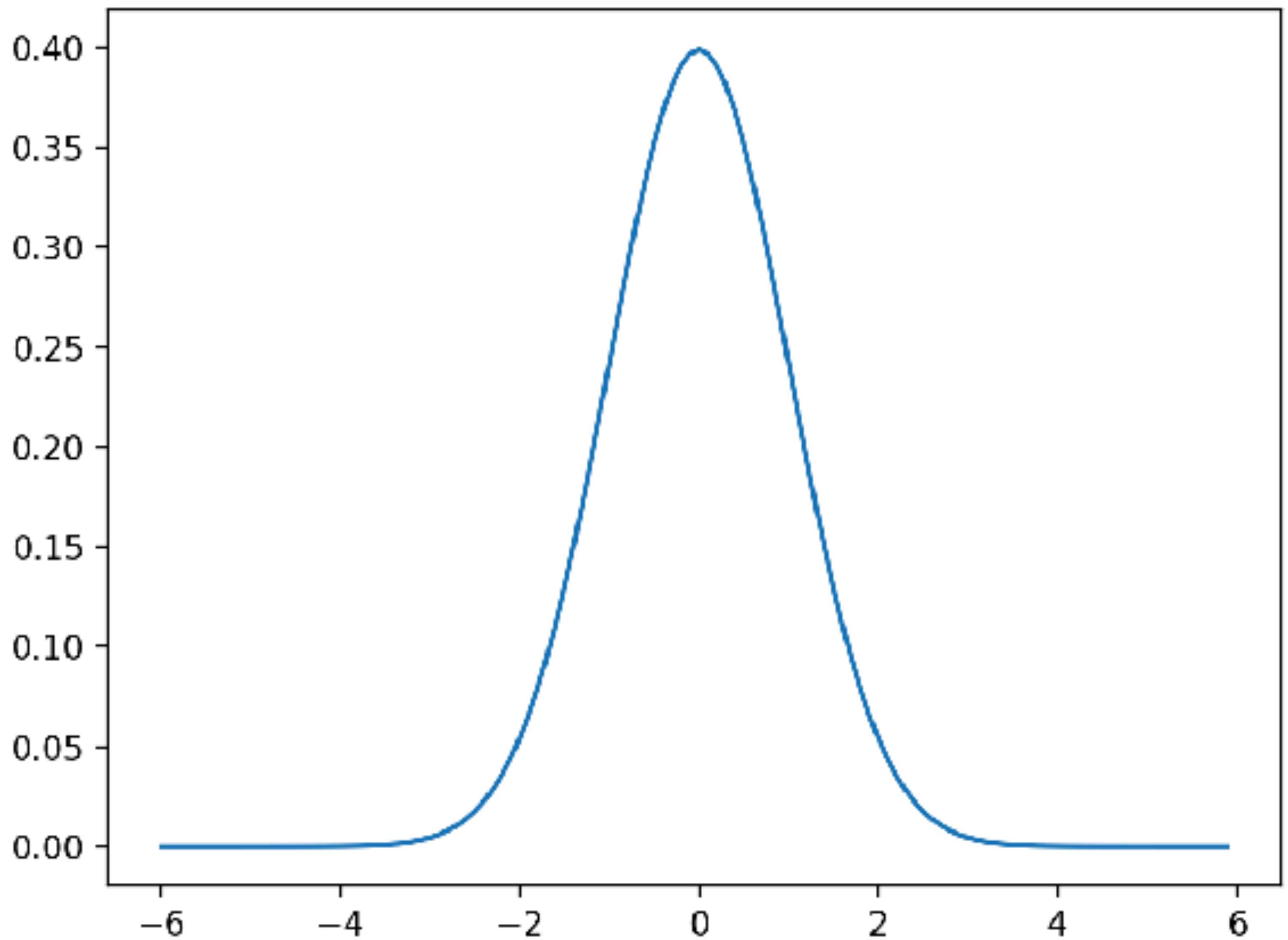
Red = more likely

Blue = less likely

Example of a 2D probability distribution

Image source: https://www.intechopen.com/source/html/17742/media/image25.png

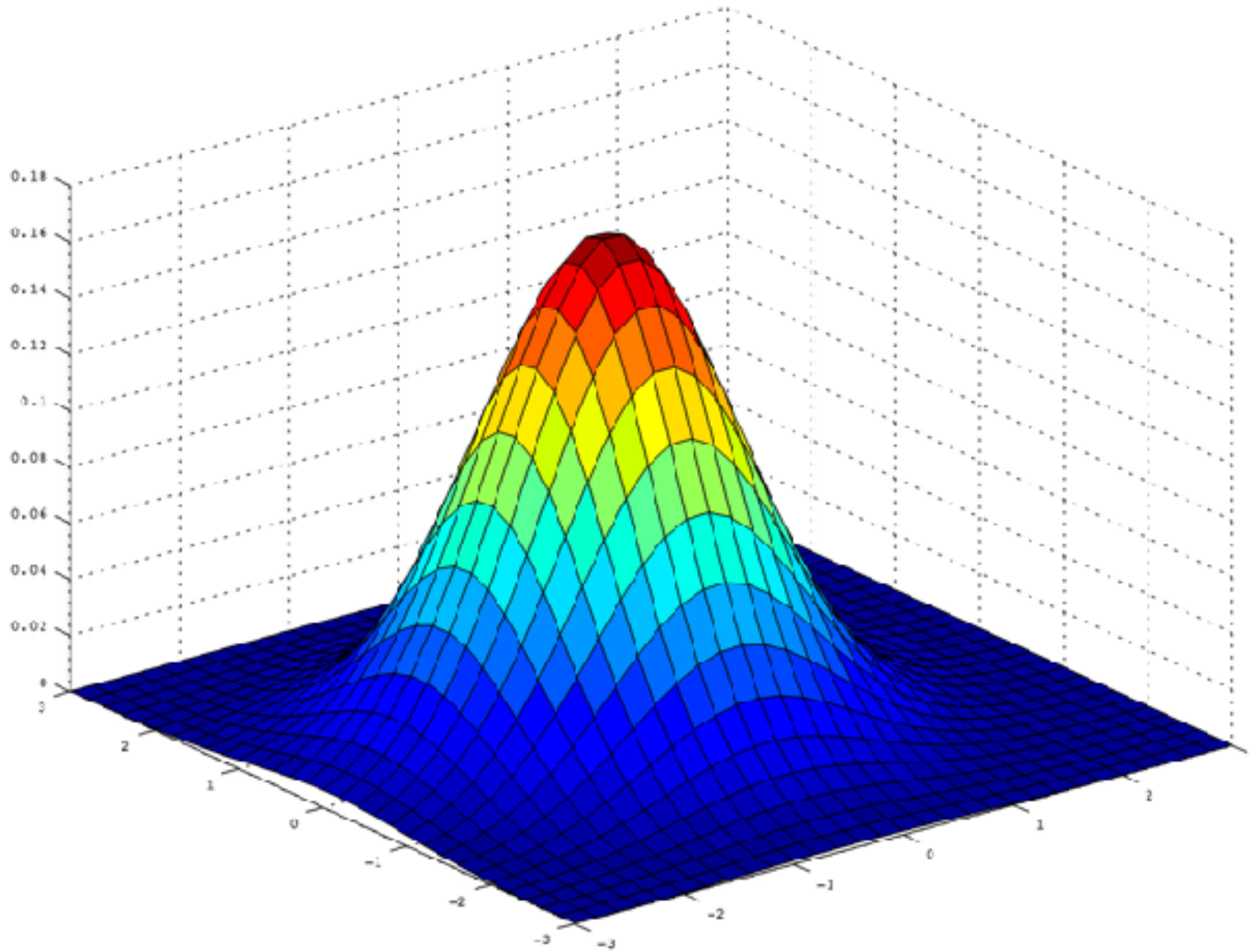
Quick Reminder: 1D Gaussian



This is a 1D Gaussian distribution

Image source: https://matthew-brett.github.io/teaching//smoothing_intro-3.hires.png

2D Gaussian



This is a 2D Gaussian distribution

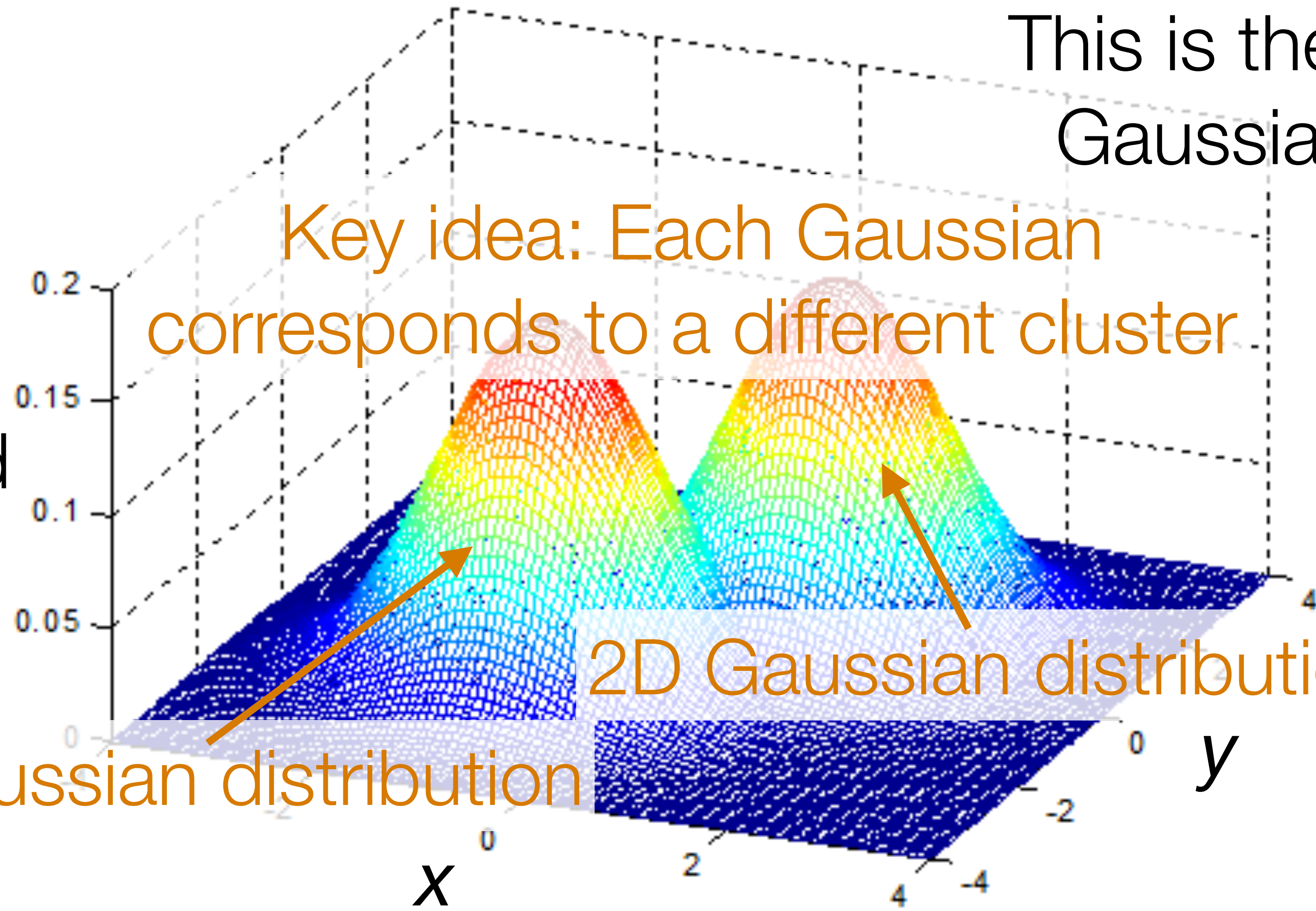
Image source: https://i.stack.imgur.com/OIWce.png

Gaussian Mixture Model (GMM)

Assume: points sampled independently from a probability distribution

This is the sum of two 2D Gaussian distributions!

how probable point generated at (x, y) is



Red = more likely

Blue = less likely

Example of a 2D probability distribution

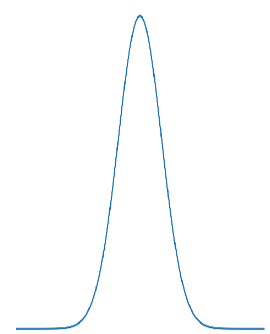
Image source: https://www.intechopen.com/source/html/17742/media/image25.png

Gaussian Mixture Model (GMM)

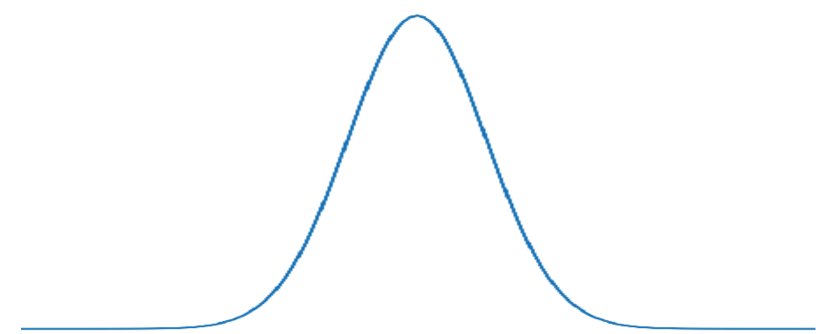
- For a fixed value *k* and dimension *d*, a GMM is the sum of *k* *d*-dimensional Gaussian distributions so that the overall probability distribution looks like *k* mountains (We've been looking at *d* = 2)
  - Each mountain corresponds to a different cluster
  - Different mountains can have different peak heights
  - One missing thing we haven't discussed yet:
    different mountains can have different shapes

2D Gaussian Shape

In 1D, you can have a skinny Gaussian or a wide Gaussian



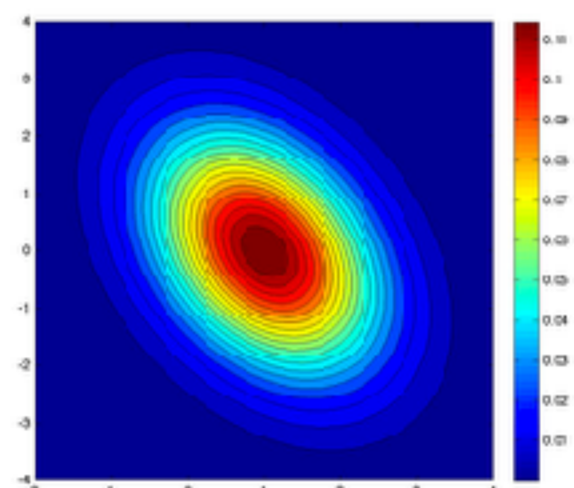
Less uncertainty



More uncertainty

In 2D, you can more generally have ellipse-shaped Gaussians

Ellipse enables encoding relationship between variables



Can't have arbitrary shapes

Top-down view of an example 2D Gaussian distribution

Image source: https://www.cs.colorado.edu/~mozer/Teaching/syllabi/ProbabilisticModels2013/homework/assign5/a52dgauss.jpg

Gaussian Mixture Model (GMM)

- For a fixed value *k* and dimension *d*, a GMM is the sum of *k* *d*-dimensional Gaussian distributions so that the overall probability distribution looks like *k* mountains (We've been looking at *d* = 2)
    - Each mountain corresponds to a different cluster
    - Different mountains can have different peak heights
    - Different mountains can have different ellipse shapes (captures "covariance" information)

Example: 1D GMM with 2 Clusters

Cluster 1

Probability of generating a point from cluster 1 = 0.5

Gaussian mean = –5

Gaussian std dev = 1

Cluster 2

Probability of generating a point from cluster 2 = 0.5

Gaussian mean = 5

Gaussian std dev = 1

What do you think this looks like?

Example: 1D GMM with 2 Clusters

<u>Cluster 1</u>

Probability of generating a point from cluster 1 = 0.5

Gaussian mean = –5

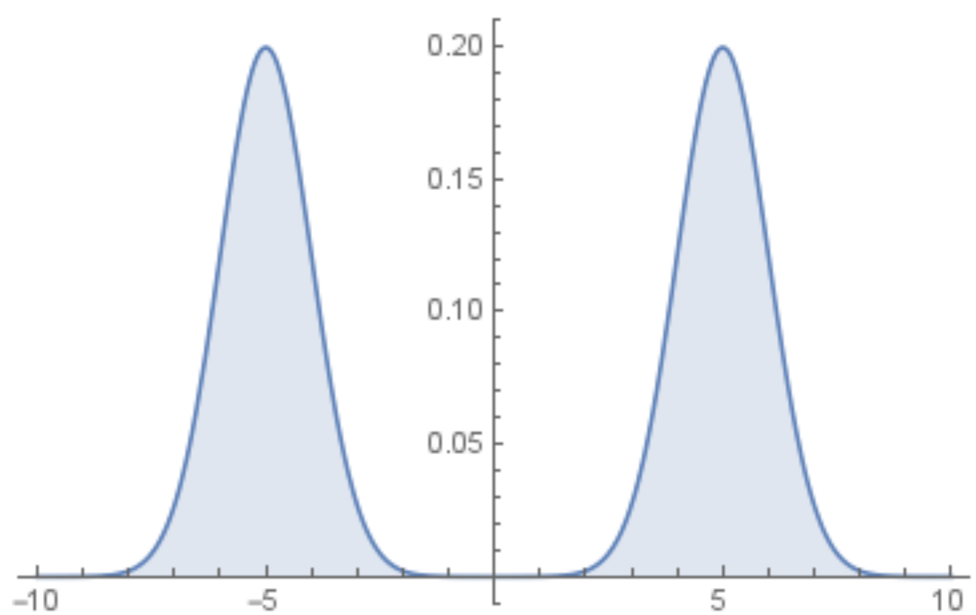
Gaussian std dev = 1

<u>Cluster 2</u>

Probability of generating a point from cluster 2 = 0.5

Gaussian mean = 5

Gaussian std dev = 1



Example: 1D GMM with 2 Clusters

Cluster 1

Probability of generating a point from cluster 1 = **0.7**

Gaussian mean = –5

Gaussian std dev = 1

Cluster 2

Probability of generating a point from cluster 2 = **0.3**

Gaussian mean = 5

Gaussian std dev = 1

What do you think this looks like?

Example: 1D GMM with 2 Clusters

<u>Cluster 1</u>

Probability of generating a point from cluster 1 = 0.7

Gaussian mean = –5

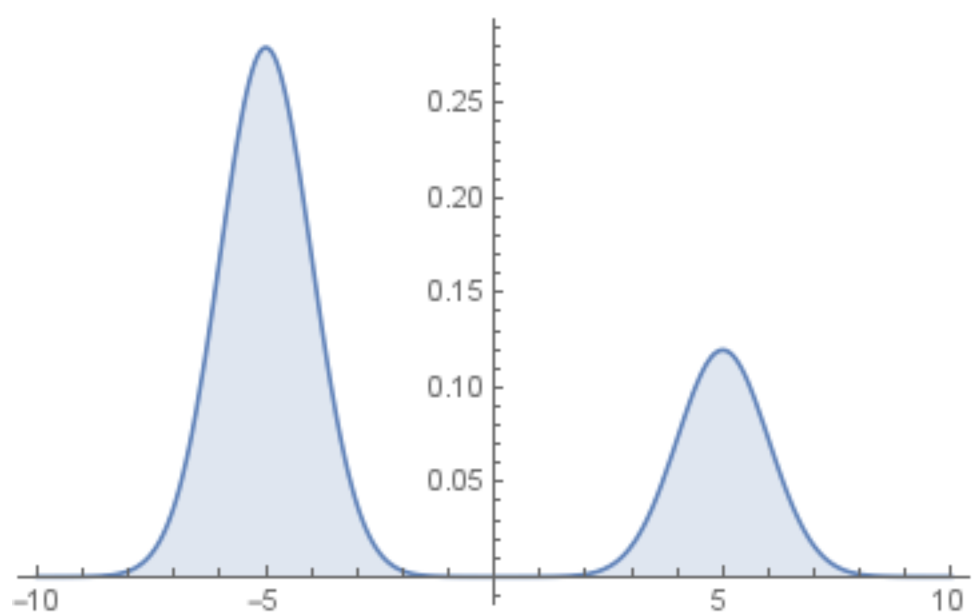
Gaussian std dev = 1

<u>Cluster 2</u>

Probability of generating a point from cluster 2 = 0.3

Gaussian mean = 5

Gaussian std dev = 1



Example: 1D GMM with 2 Clusters

<u>Cluster 1</u>

Probability of generating a point from cluster 1 = 0.7

Gaussian mean = –5

Gaussian std dev = 1

<u>Cluster 2</u>

Probability of generating a point from cluster 2 = 0.3

Gaussian mean = 5

Gaussian std dev = 1

How to generate 1D points from this GMM:

1. Flip biased coin (with probability of heads 0.7)
2. If heads: sample 1 point from Gaussian mean -5, std dev 1

   If tails: sample 1 point from Gaussian mean 5, std dev 1

Example: 1D GMM with 2 Clusters

Cluster 1

Probability of generating a point from cluster 1 = π_1

Gaussian mean = μ_1

Gaussian std dev = σ_1

Cluster 2

Probability of generating a point from cluster 2 = π_2

Gaussian mean = μ_2

Gaussian std dev = σ_2

How to generate 1D points from this GMM:

1. Flip biased coin (with probability of heads π_1)
2. If heads: sample 1 point from Gaussian mean μ_1, std dev σ_1
   If tails: sample 1 point from Gaussian mean μ_2, std dev σ_2

Example: 1D GMM with *k* Clusters

<u>Cluster 1</u>

Probability of generating a point from cluster 1 = π_1

Gaussian mean = μ_1

Gaussian std dev = σ_1

…

<u>Cluster *k*</u>

Probability of generating a point from cluster k = π_k

Gaussian mean = μ_k

Gaussian std dev = σ_k

How to generate 1D points from this GMM:

1. Flip biased *k*-sided coin (the sides have probabilities $\pi_1, \ldots, \pi_k$)
2. Let Z be the side that we got (it is some value 1, …, k)
3. Sample 1 point from Gaussian mean μ_Z, std dev σ_Z

Example: 2D GMM with k Clusters

Cluster 1

Probability of generating a point from cluster 1 = π_1

Gaussian mean = μ_1 2D point

Gaussian **covariance** = Σ_1

2x2 matrix

…

Cluster k

Probability of generating a point from cluster k = π_k

Gaussian mean = μ_k 2D point

Gaussian **covariance** = Σ_k

2x2 matrix

How to generate **2D** points from this GMM:

1. Flip biased k-sided coin (the sides have probabilities $\pi_1, \ldots, \pi_k$)
2. Let Z be the side that we got (it is some value $1, \ldots, k$)
3. Sample 1 point from Gaussian mean μ_Z, **covariance** Σ_Z

GMM with k Clusters

<u>Cluster 1</u>

Probability of generating a point from cluster 1 = π_1

Gaussian mean = μ_1

Gaussian covariance = Σ_1

…

<u>Cluster k</u>

Probability of generating a point from cluster k = π_k

Gaussian mean = μ_k

Gaussian covariance = Σ_k

How to generate points from this GMM:

1. Flip biased k-sided coin (the sides have probabilities $\pi_1, \ldots, \pi_k$)
2. Let Z be the side that we got (it is some value $1, \ldots, k$)
3. Sample 1 point from Gaussian mean μ_Z, covariance Σ_Z

High-Level Idea of GMM

- Generative model that gives a *hypothesized* way in which data points are generated

<u>In reality, data are unlikely generated the same way!</u>

<u>In reality, data points might not even be independent!</u>

“All models are wrong, but some are useful.”

–George Edward Pelham Box

High-Level Idea of GMM

- Generative model that gives a *hypothesized* way in which data points are generated

In reality, data are unlikely generated the same way!

In reality, data points might not even be independent!

- Learning ("fitting") the parameters of a GMM
  - Input: *d*-dimensional data points, your guess for *k*
  - Output: $\pi_1, \ldots, \pi_k, \mu_1, \ldots, \mu_k, \Sigma_1, \ldots, \Sigma_k$
- *After* learning a GMM:
  - For *any* *d*-dimensional data point, can figure out probability of it belonging to each of the clusters

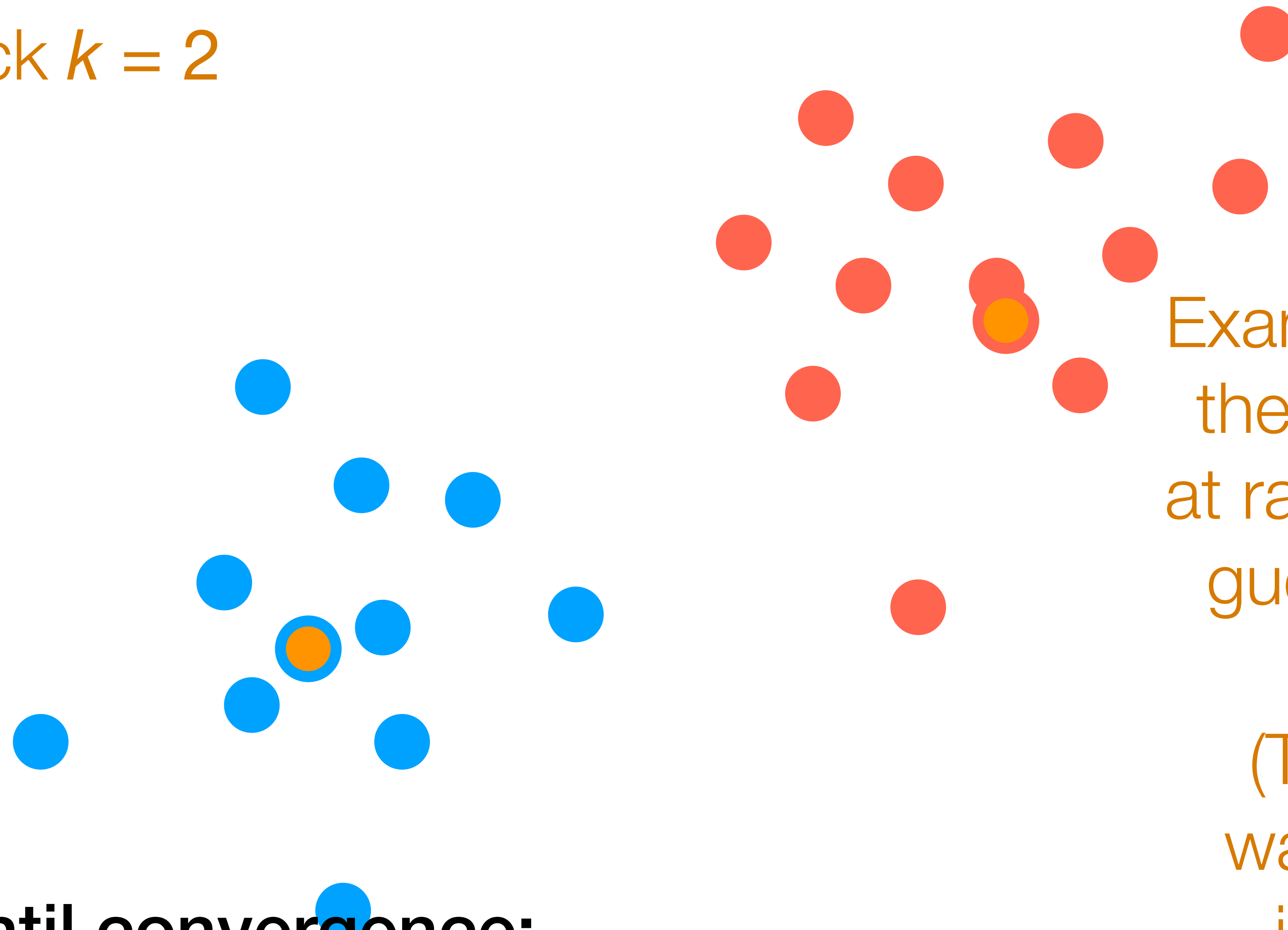
How do you turn this into a cluster assignment?

k-means

Step 1: Pick <u>guesses</u> for where cluster centers are

Step 0: Pick *k*

We'll pick *k* = 2



Example: choose *k* of the points uniformly at random to be initial guesses for cluster centers

(There are many ways to make the initial guesses)

Repeat until convergence:

Step 2: Assign each point to belong to the closest cluster

Step 3: Update cluster means (to be the center of mass per cluster)

k-means

Step 0: Pick *k*

Step 1: Pick <u>guesses</u> for
where cluster centers are

Repeat until convergence:

Step 2: Assign each point to belong to the closest cluster

Step 3: Update cluster means (to be the center of mass per cluster)

(Rough Intuition) Learning a GMM

Step 0: Pick *k*

Step 1: Pick <u>guesses</u> for **cluster probabilities, means, and covariances** (often done using *k*-means)

Repeat until convergence:

Step 2: Compute probability of each point belonging to each of the *k* clusters

Step 3: Update **cluster probabilities, means, and covariances** carefully accounting for probabilities of each point belonging to each of the clusters

This algorithm is called the Expectation-Maximization (EM) algorithm specifically for GMM's (and approximately does maximum likelihood)

(Note: EM by itself is a general algorithm not just for GMM's)

Relating *k*-means to GMM's

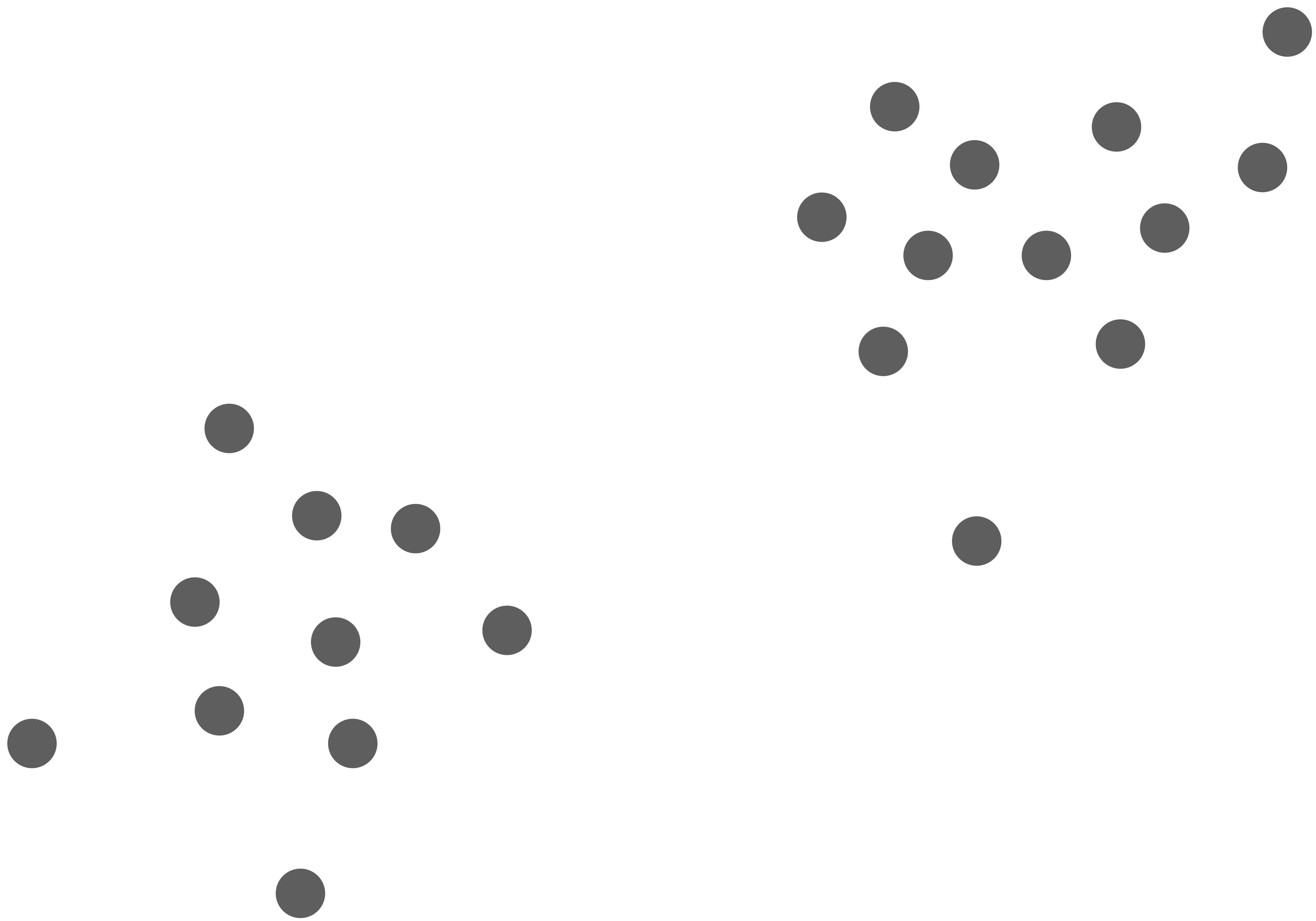
If the ellipses are all circles and have the same "skinniness" (e.g., in the 1D case it means they all have same std dev):

- *k*-means approximates the EM algorithm for GMM's

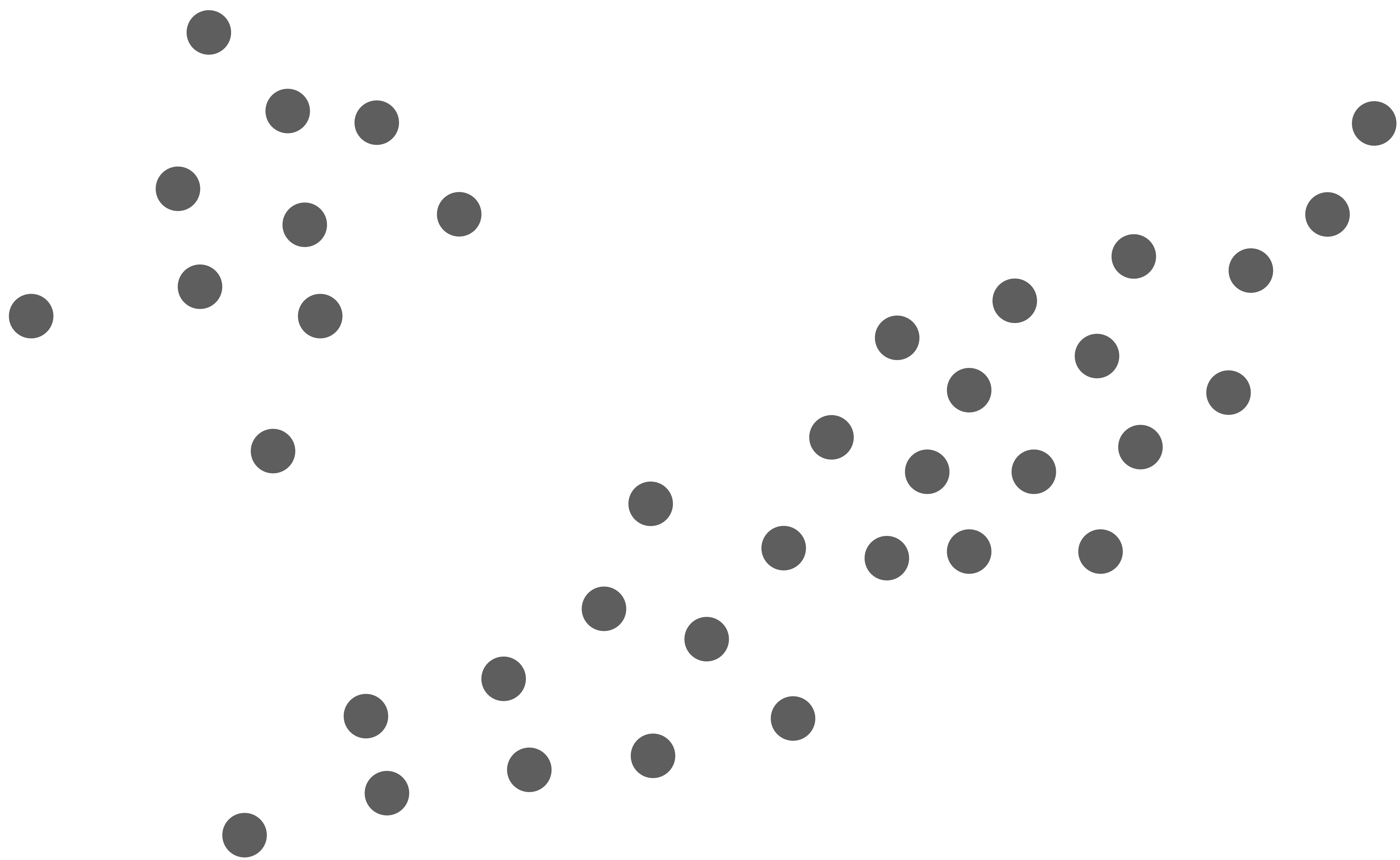
- Notice that *k*-means does a "hard" assignment of each point to a cluster, whereas the EM algorithm does a "soft" (probabilistic) assignment of each point to a cluster

Interpretation: *We know when k-means should work! It should work when the data appear as if they're from a GMM with true clusters that "look like circles"*

k-means should do well on this



But not on this



Learning a GMM

Demo