# Introduction to Clustering

Similarity functions, *k*-means, Gaussian mixture models

slides by
George Chen
Carnegie Mellon University
Fall 2017

Image source: http://static3.businessinsider.com/image/58f900e37522cacd008b4ee9/scott-galloway-netflix-could-be-the-next-300-billion-company.jpg


Suppose Netflix asks you how to go about understanding what kind of TV show it should produce next. How would you go about doing it?

*Image source: http://static3.businessinsider.com/image/58f900e37522cacd008b4ee9/scott-galloway-netflix-could-be-the-next-300-billion-company.jpg*

We want to understand user tastes

# Movie Recommendation Data

# Movie Recommendation Data

User 1

User 2

⋮

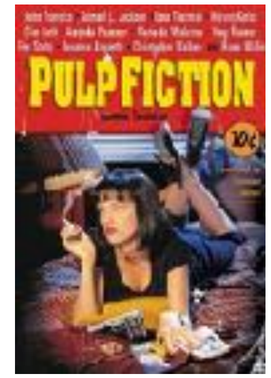
User *n*

# Movie Recommendation Data

Item 1

Item 2


Item 3

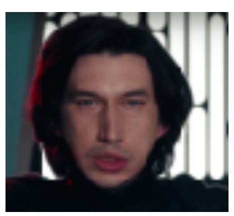



Item 4

…

Item *m*


User 1

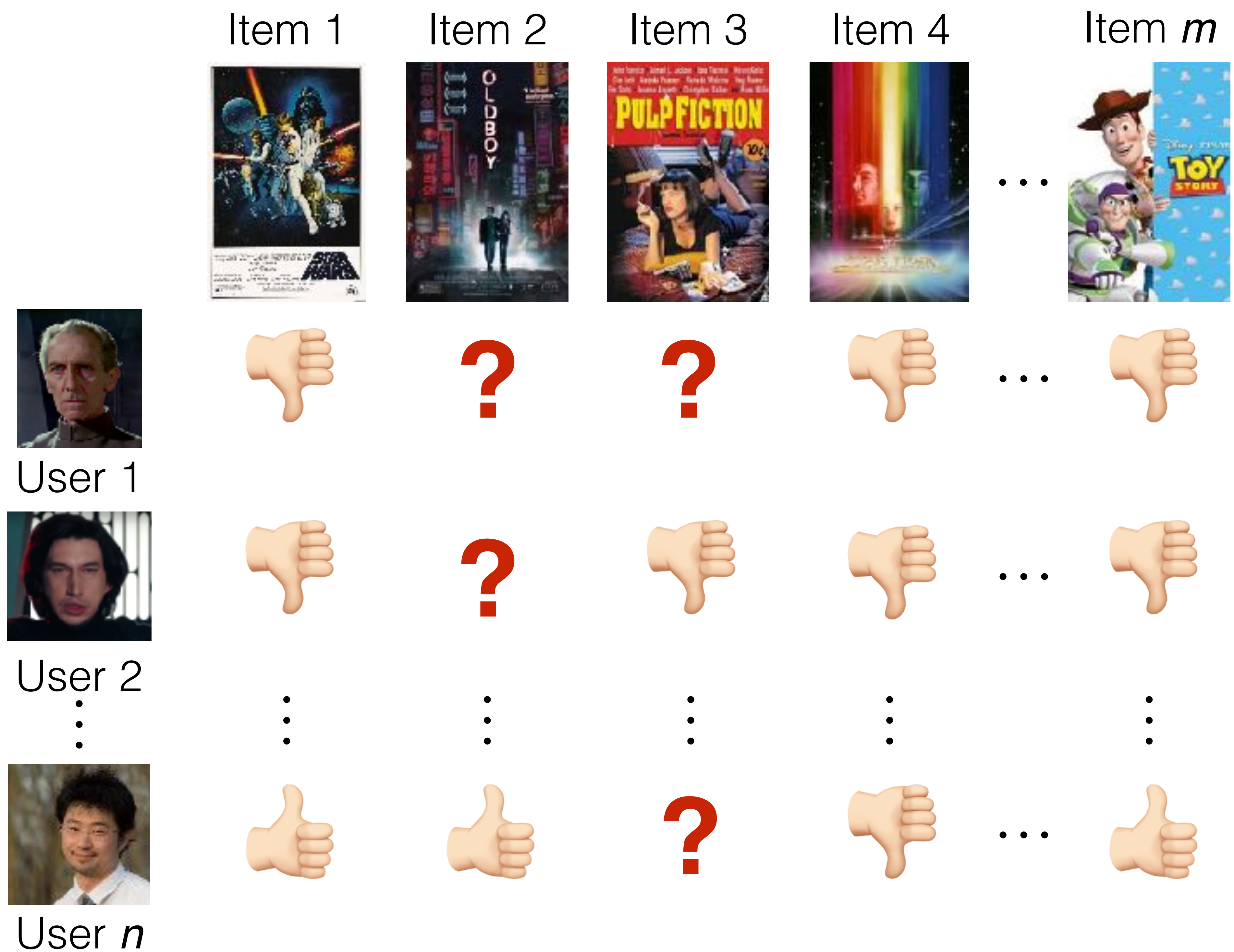


User 2

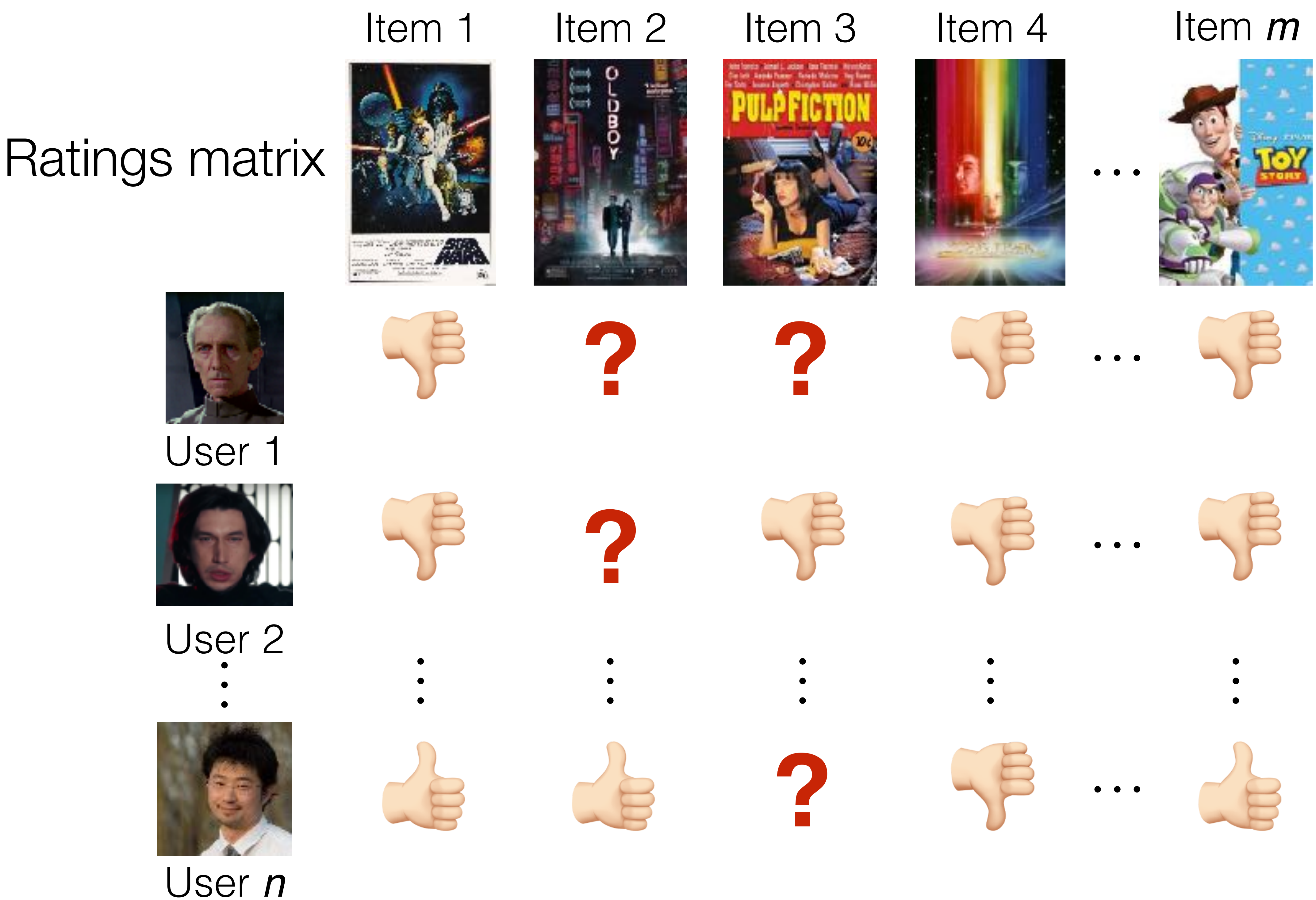
⋮

User *n*

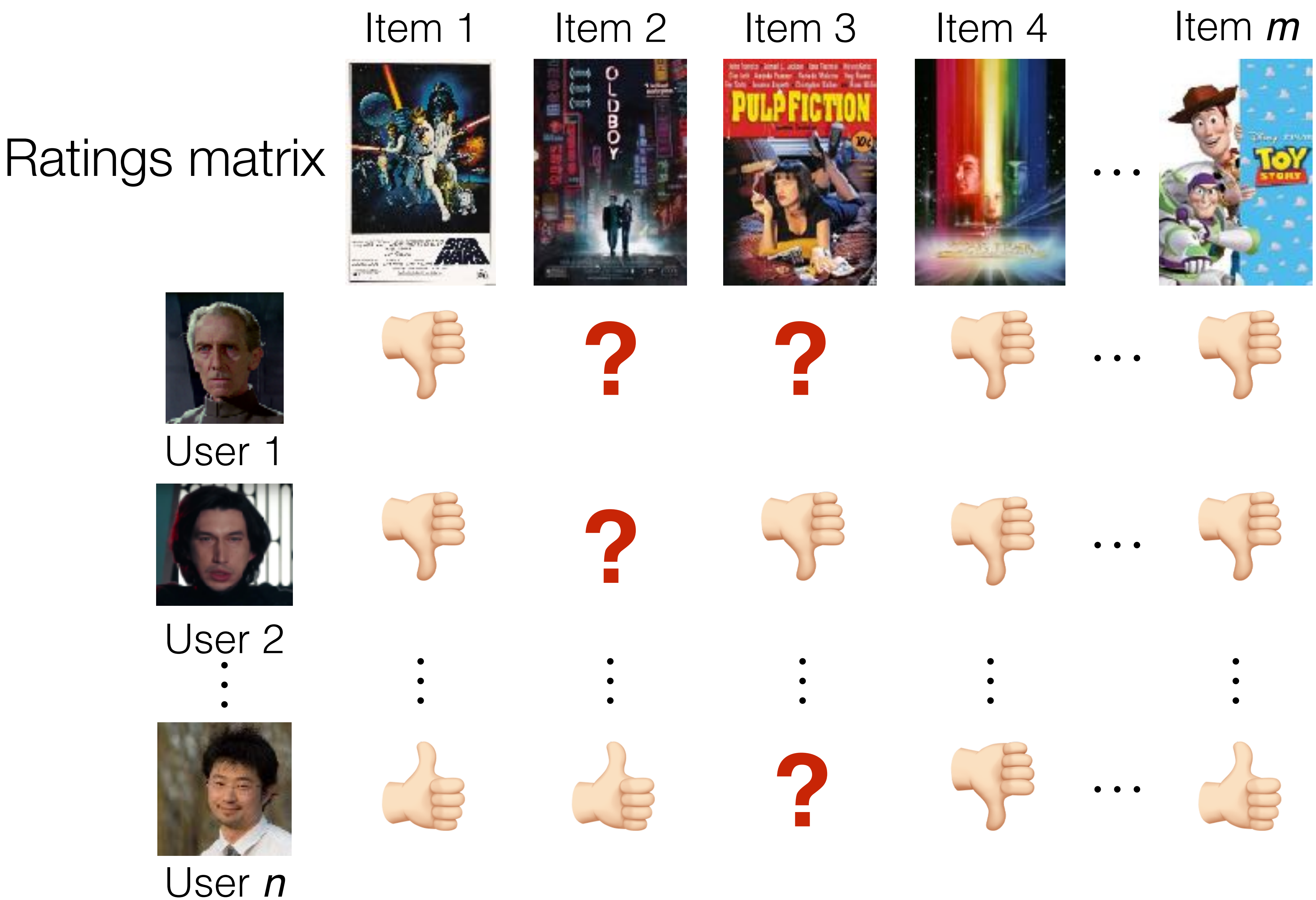
# Movie Recommendation Data



# Movie Recommendation Data



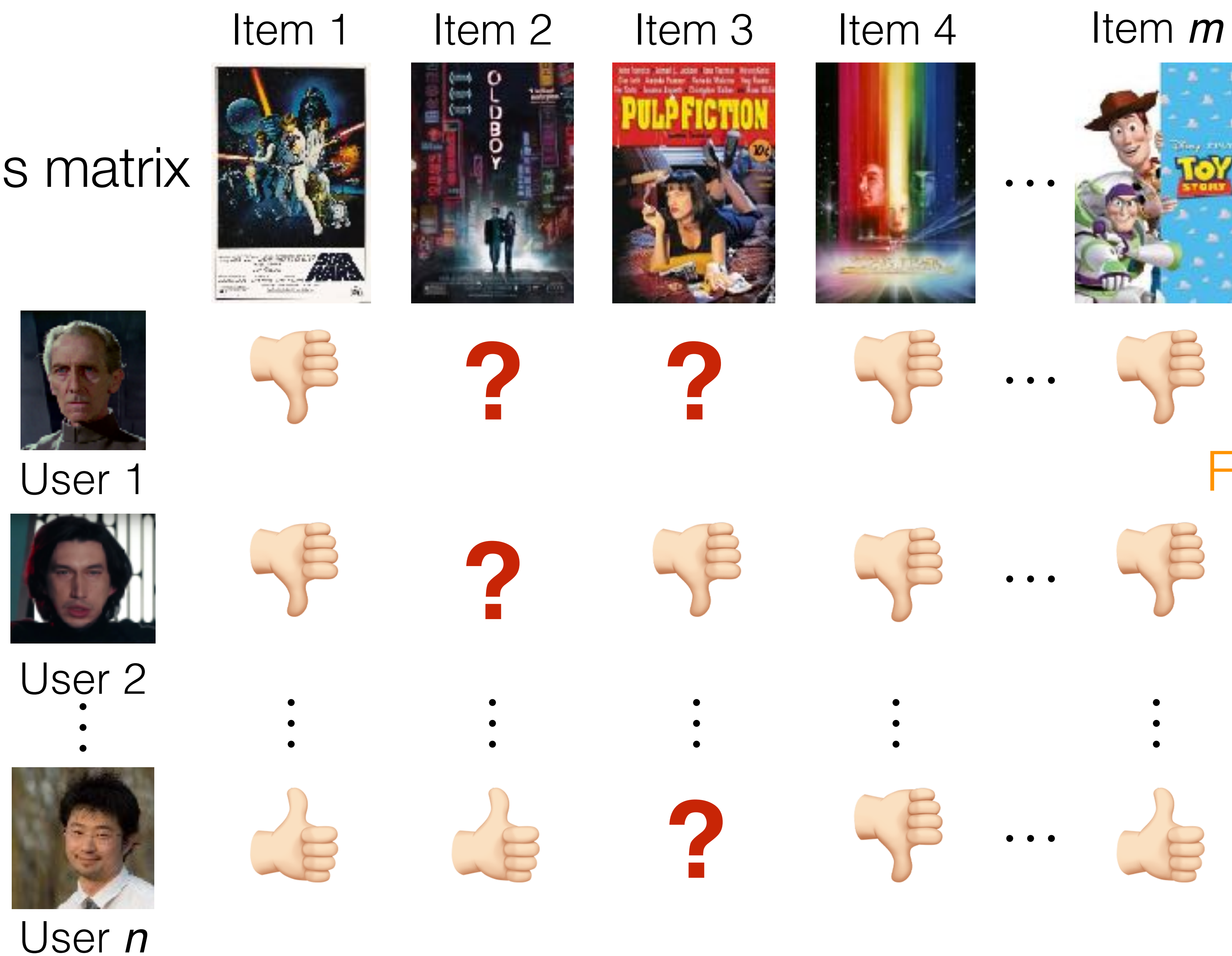
# Movie Recommendation Data




We can also scrape IMDb for a lot of semantic information (actresses, actors, genres, reviews, etc) about movies/TV shows

# Movie Recommendation Data

Ratings matrix

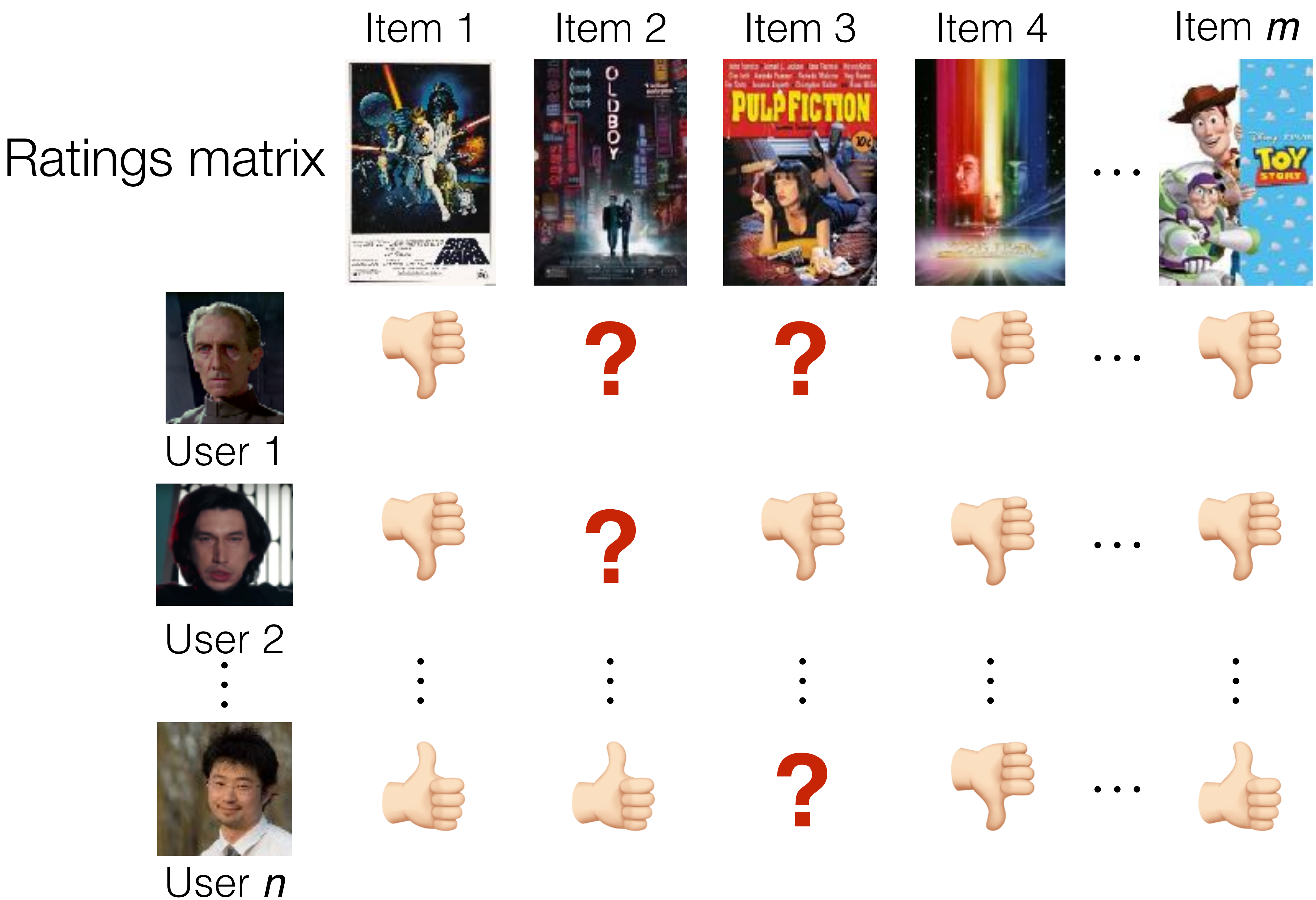



For simplicity: consider single snapshot in time

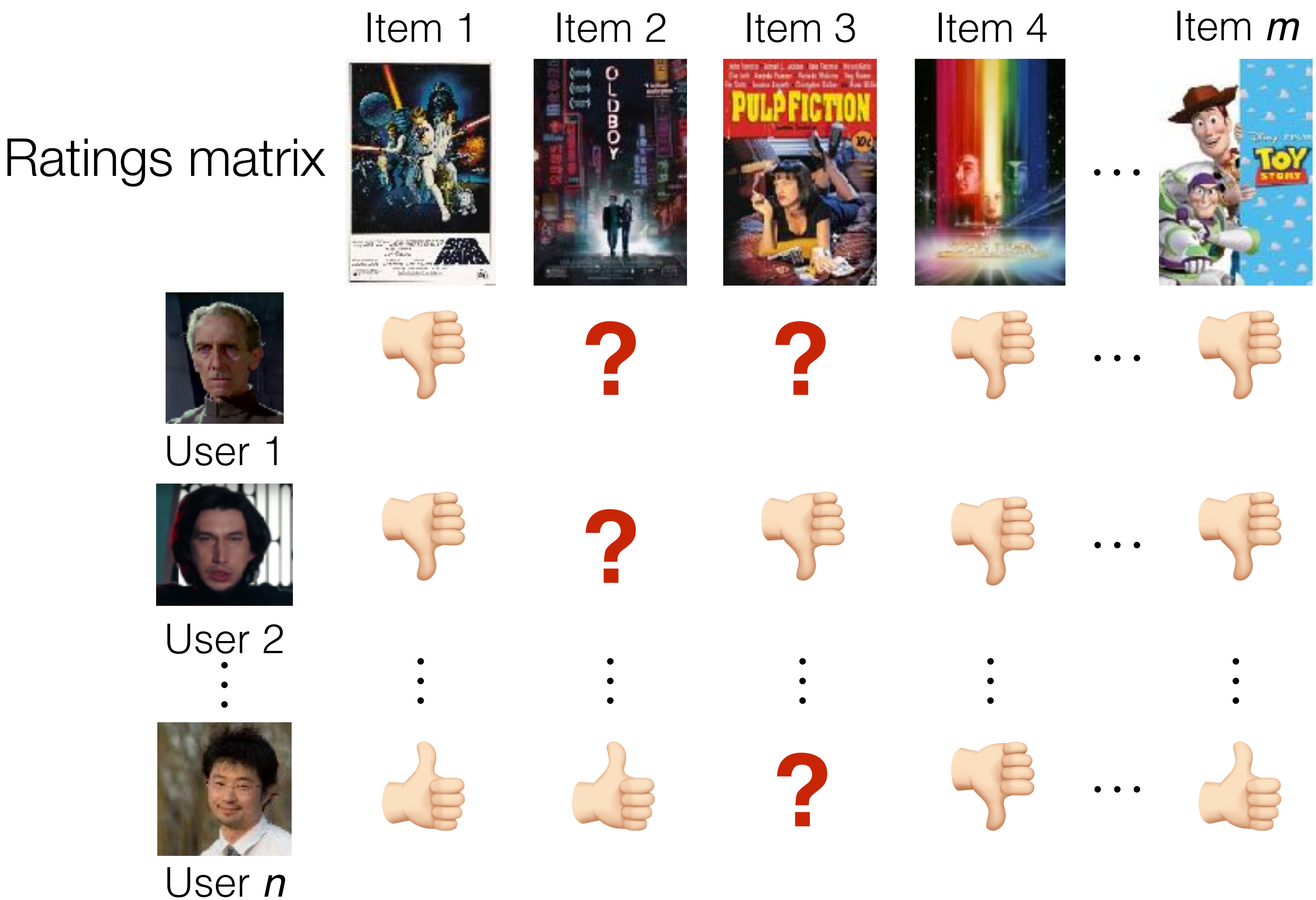
We can also scrape IMDb for a lot of semantic information (actresses, actors, genres, reviews, etc) about movies/TV shows

When looking for structure,
it's helpful to hypothesize
what structure there might be

# Movie Recommendation Data



# Movie Recommendation Data




Simple hypothesis: There are clusters of users with similar taste

# Movie Recommendation Data


Simple hypothesis: There are clusters of users with similar taste

# Movie Recommendation Data


Simple hypothesis: There are clusters of users with similar taste

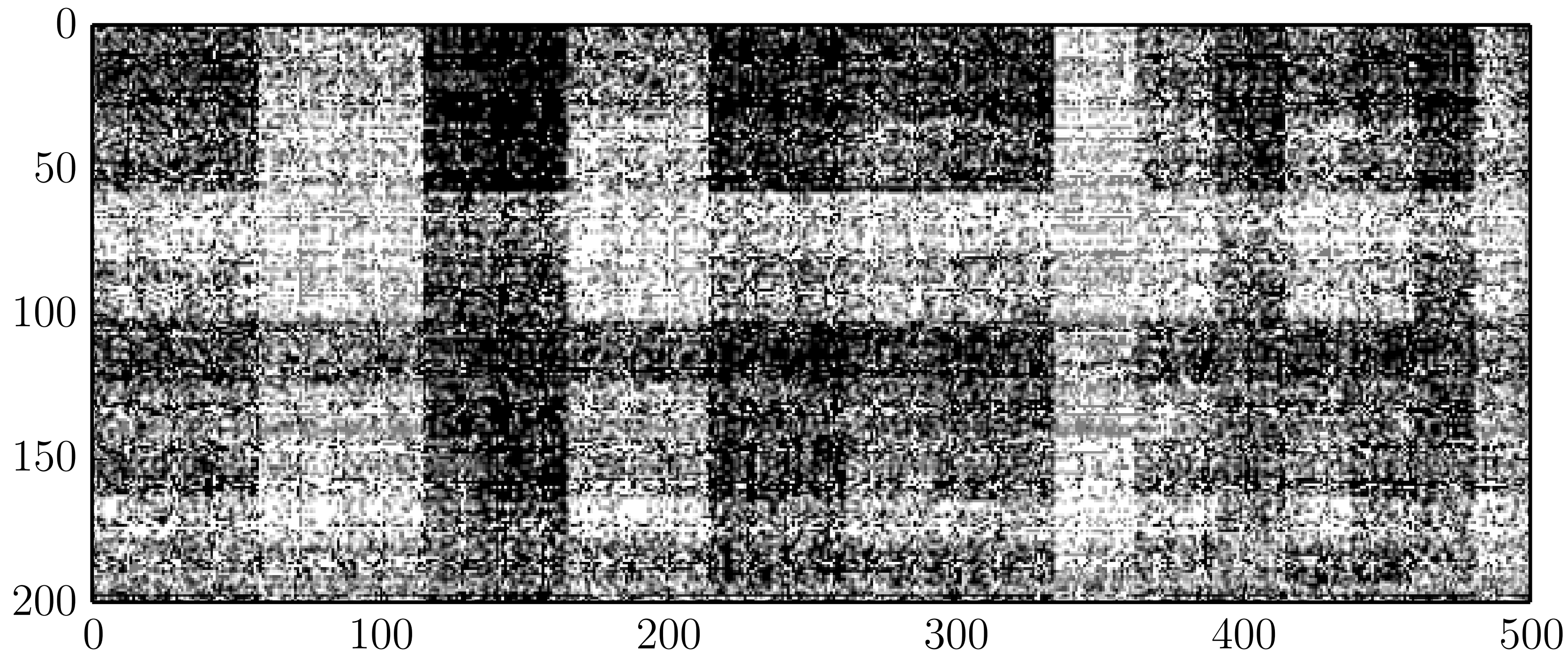
# Is the Hypothesis on Users True?

# Is the Hypothesis on Users True?

black = user dislikes movie

white = user likes movie

Users



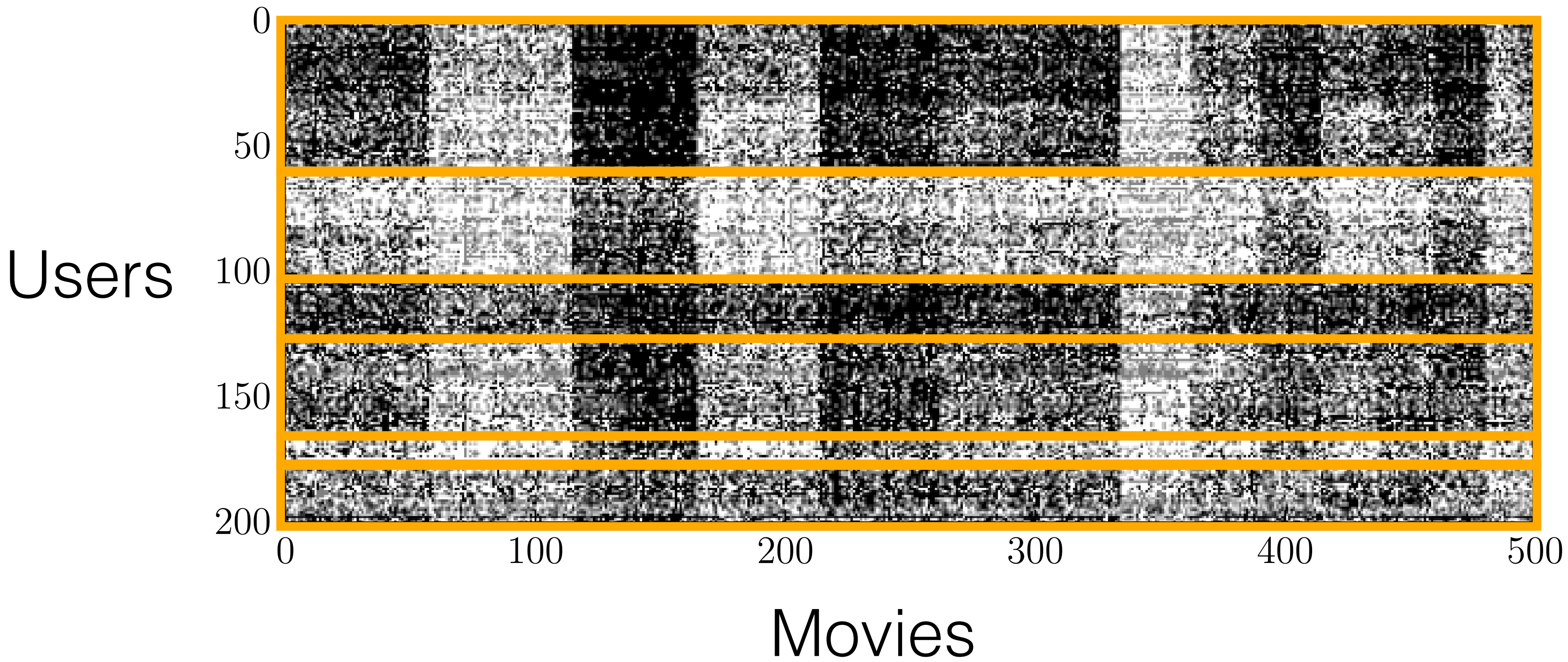

Movies

*Dense part of Netflix Prize data*

# Is the Hypothesis on Users True?

black = user dislikes movie

white = user likes movie



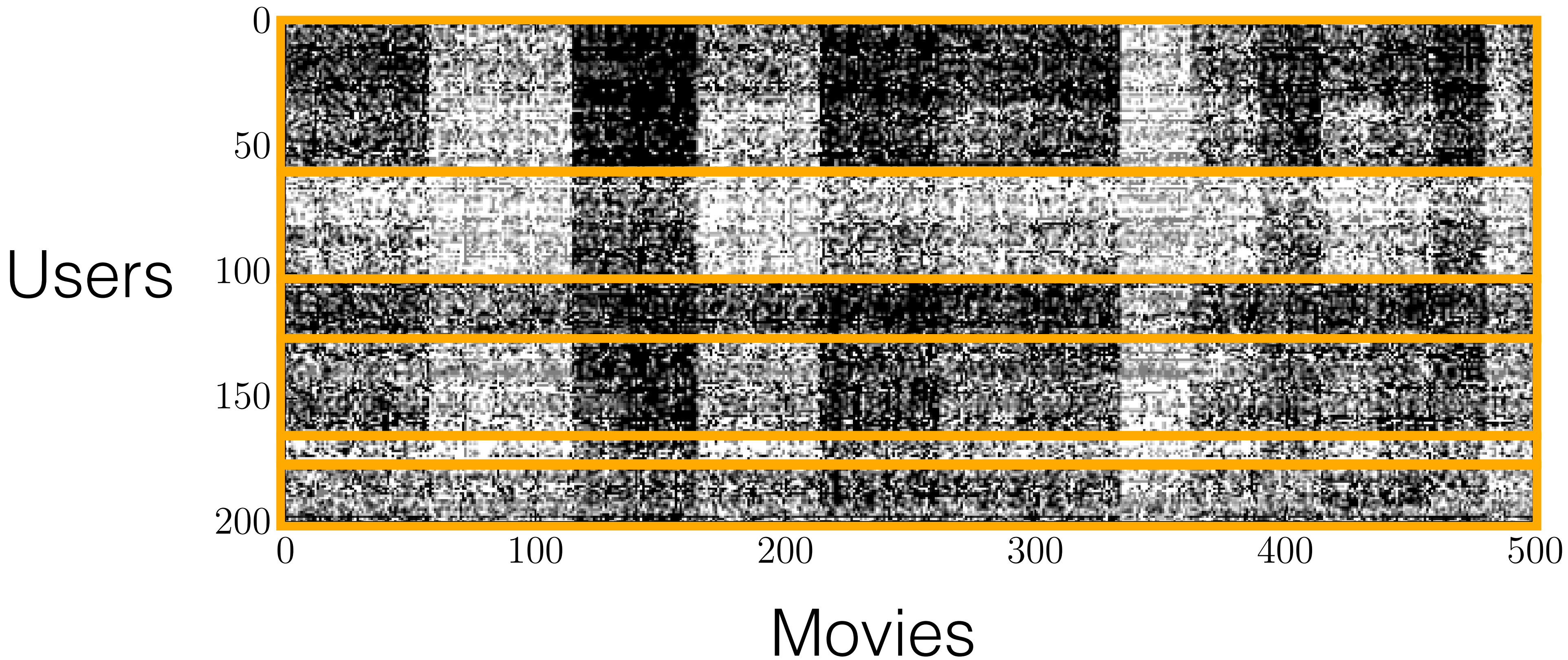

There are blocks of similar users!

*Dense part of Netflix Prize data*

# Is the Hypothesis on Users True?

black = user dislikes movie

white = user likes movie




There are blocks of similar users!

In fact there are blocks of similar items as well!

*Dense part of Netflix Prize data*

# Defining Similarity

The Art of

# Defining Similarity

# The Art of Defining Similarity

- There usually is no “best” way to define similarity

# The Art of Defining Similarity

- There usually is no “best” way to define similarity

**Example:** cosine similarity between users

# The Art of Defining Similarity

- There usually is no “best” way to define similarity

**Example:** cosine similarity between users

# The Art of Defining Similarity

- There usually is no “best” way to define similarity

**Example:** cosine similarity between users

User *u*

User *v*

# The Art of Defining Similarity

- There usually is no “best” way to define similarity

**Example:** cosine similarity between users

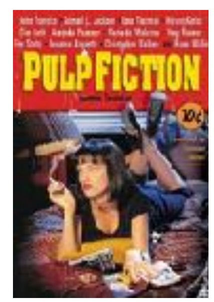
User *u*

User *v*

# The Art of Defining Similarity

- There usually is no “best” way to define similarity

**Example:** cosine similarity between users




User *u*

| | |
|---|---|
| +1 | −1 |

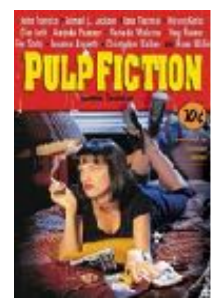
User *v*

| | |
|---|---|
| +1 | +1 |

# The Art of Defining Similarity

- There usually is no "best" way to define similarity

**Example:** cosine similarity between users



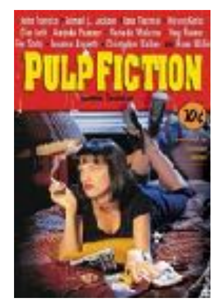

User $u$  $Y_u$ = [ $+1$ $-1$ ]

User $v$  $Y_v$ = [ $+1$ $+1$ ]

# The Art of Defining Similarity

- There usually is no “best” way to define similarity

**Example:** cosine similarity between users


User $u$  $Y_u$ = [ +1, −1 ]

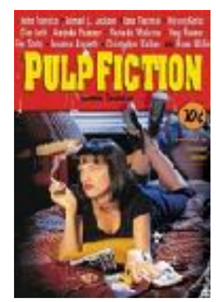
User $v$  $Y_v$ = [ +1, +1 ]

$$\frac{\langle Y_u, Y_v \rangle}{\|Y_u\| \|Y_v\|}$$

# The Art of Defining Similarity

- There usually is no "best" way to define similarity

**Example:** cosine similarity between users


User $u$  $Y_u$: | +1 | −1 |

User $v$  $Y_v$: | +1 | +1 |

$$\frac{\langle Y_u, Y_v \rangle}{\|Y_u\| \|Y_v\|} = 0$$

# The Art of Defining Similarity

- There usually is no “best” way to define similarity

**Example:** cosine similarity $$\frac{\langle Y_u, Y_v \rangle}{\| Y_u \| \| Y_v \|}$$

# The Art of Defining Similarity

- There usually is no "best" way to define similarity

**Example:** cosine similarity $\dfrac{\langle Y_u, Y_v \rangle}{\|Y_u\| \|Y_v\|}$

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

# The Art of Defining Similarity

- There usually is no “best” way to define similarity

**Example:** cosine similarity $\frac{\langle Y_u, Y_v \rangle}{\|Y_u\| \|Y_v\|}$

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

**Example:** Euclidean distance $\|Y_u - Y_v\|$

# The Art of Defining Similarity

- There usually is no “best” way to define similarity

**Example:** cosine similarity $\dfrac{\langle Y_u, Y_v \rangle}{\|Y_u\| \|Y_v\|}$

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

**Example:** Euclidean distance $\|Y_u - Y_v\|$

Turn into similarity with decaying exponential $\downarrow$

# The Art of Defining Similarity

- There usually is no "best" way to define similarity

**Example:** cosine similarity $\dfrac{\langle Y_u, Y_v \rangle}{\|Y_u\| \|Y_v\|}$

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

**Example:** Euclidean distance $\|Y_u - Y_v\|$

Turn into similarity with decaying exponential ↓

$$\exp(-\gamma \|Y_u - Y_v\|)$$

where $\gamma > 0$

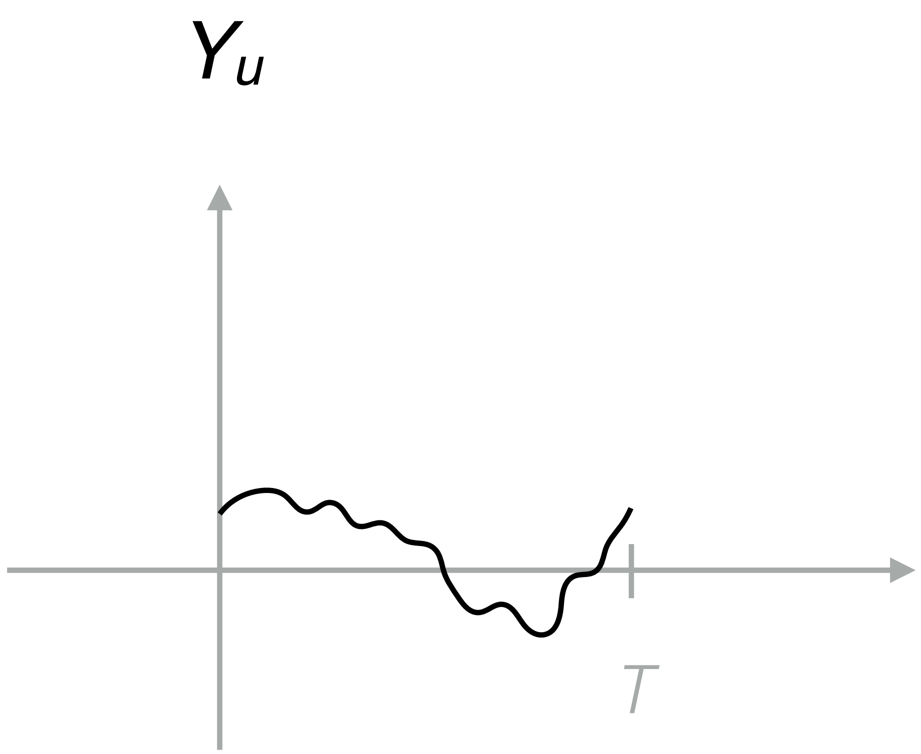
# Example: Time Series

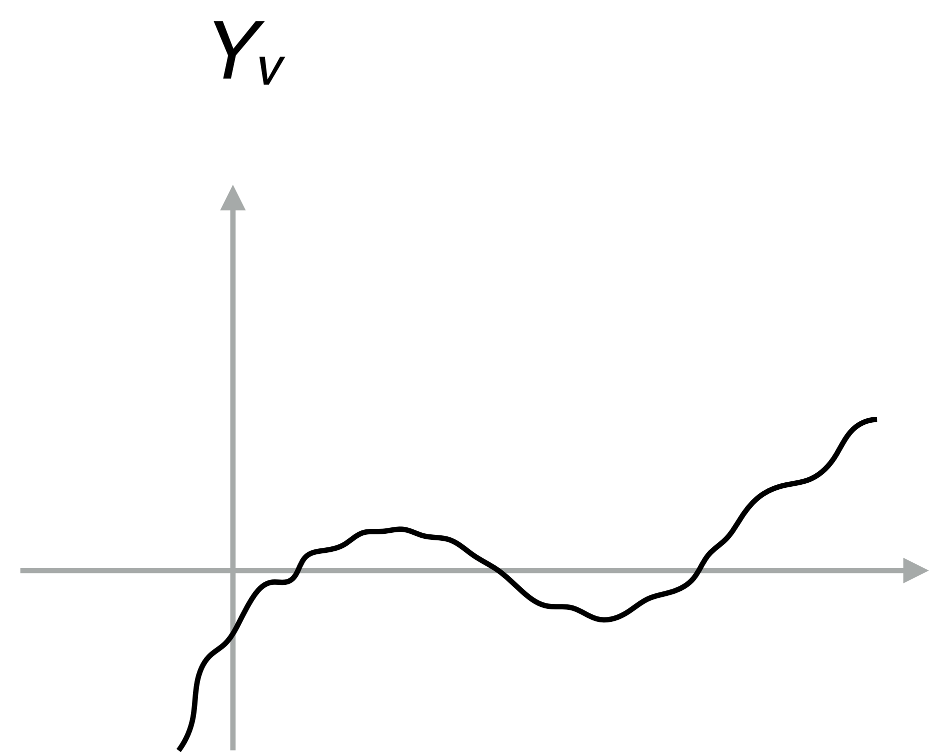
# Example: Time Series

How would you compute a distance between these?

# Example: Time Series

How would you compute a distance between these?

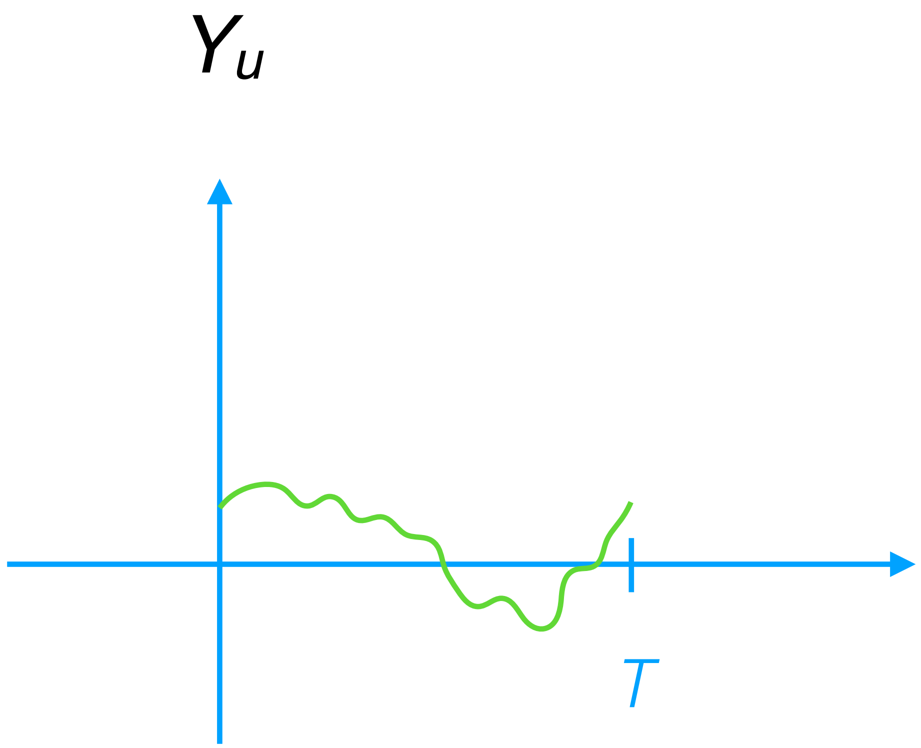


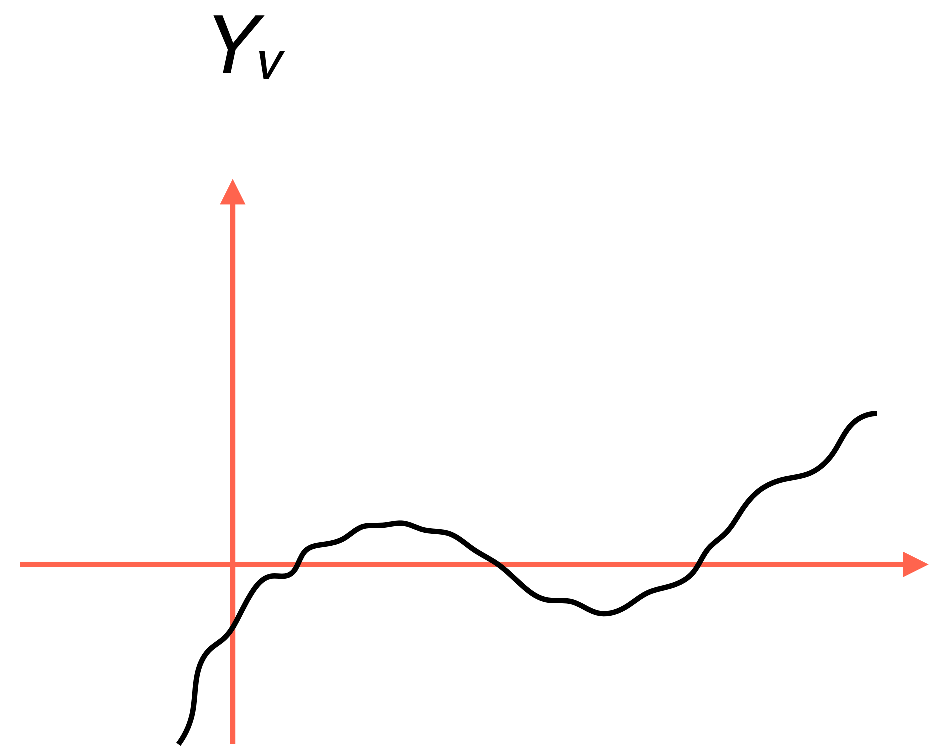

Only observe time steps between 0 and $T$

# Example: Time Series

How would you compute a distance between these?

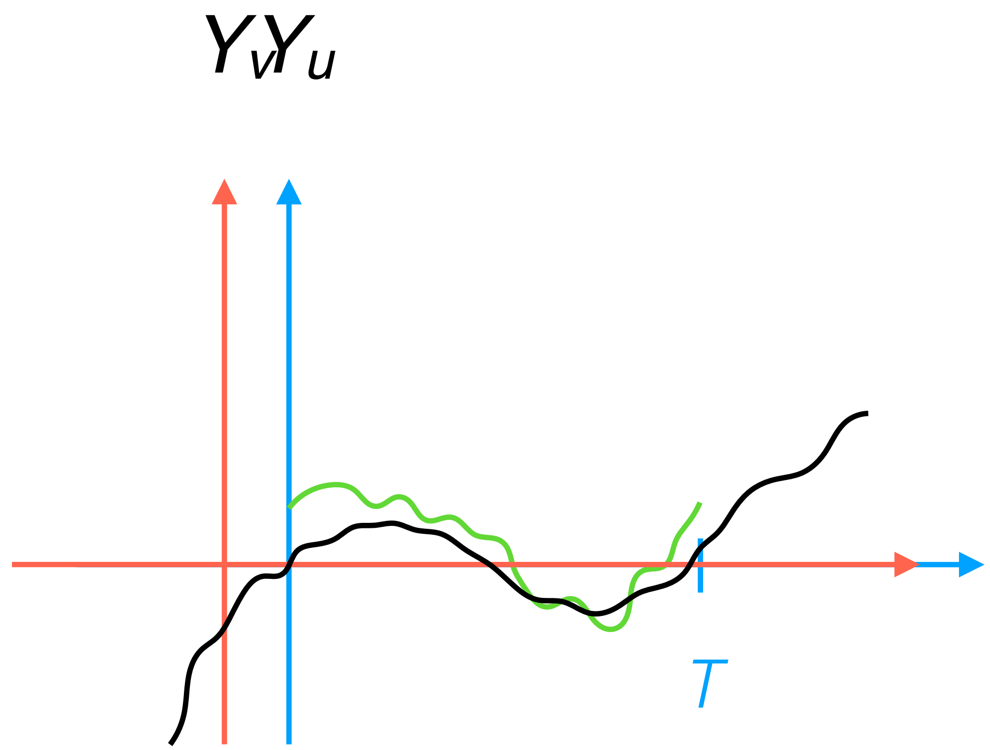



Only observe time steps between 0 and $T$



# Example: Time Series

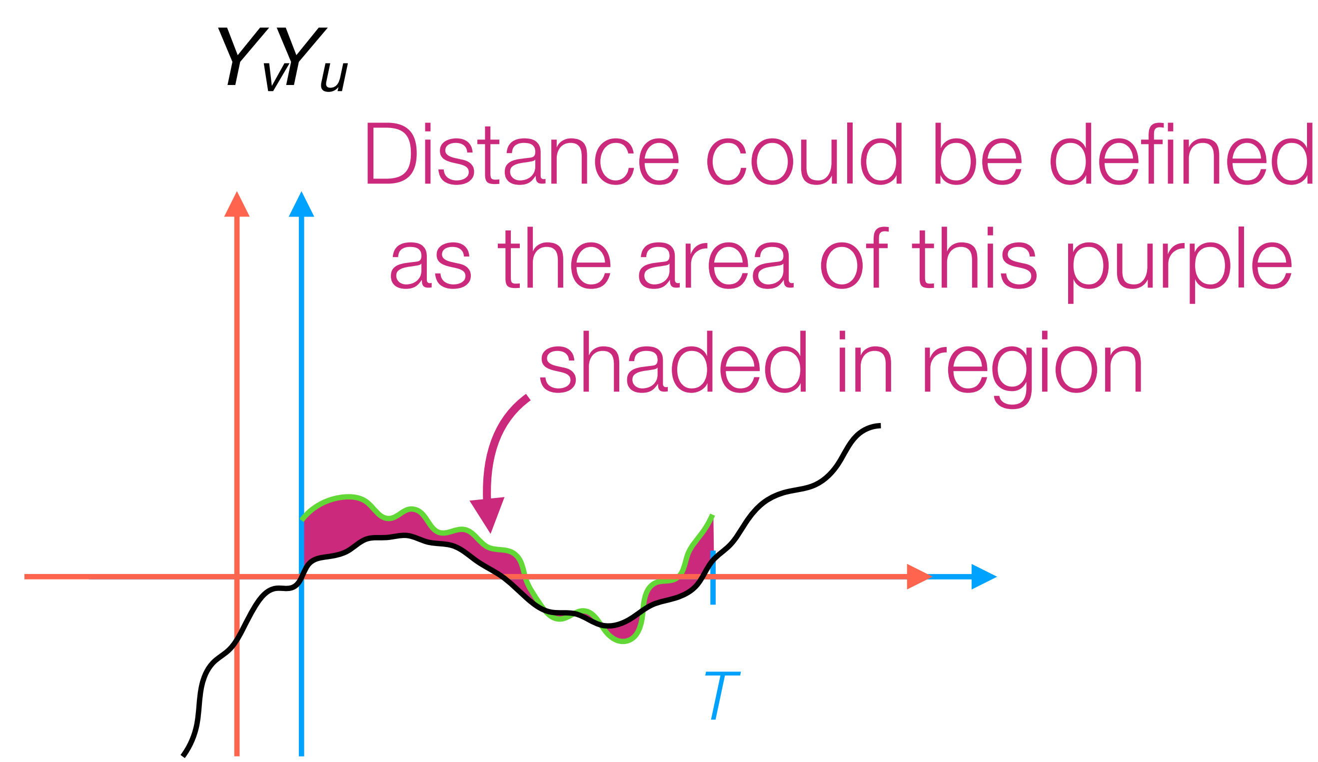
How would you compute a distance between these?




One solution: Align them first

# Example: Time Series

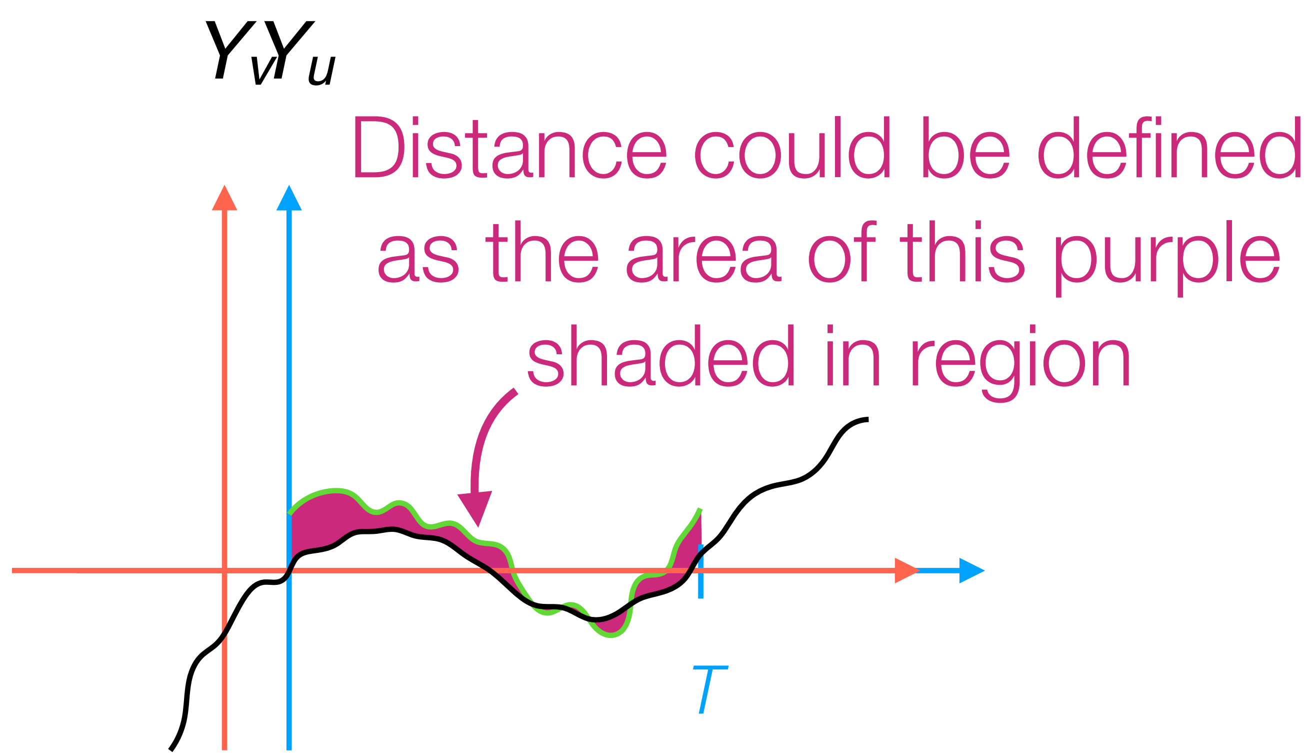
How would you compute a distance between these?




One solution: Align them first

# 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" to first align (it works kind of like how spell check does for words)

# Similarity Diagnostics

# Similarity Diagnostics

- As you try different similarity functions, easy thing to check:

# Similarity Diagnostics

- As you try different similarity functions, easy thing to check:
    - Pick any data point

# Similarity Diagnostics

- As you try different similarity functions, easy thing to check:
  - Pick any data point
  - Compute its similarity to all the other data points, and rank them in decreasing over from most similar to least similar

# Similarity Diagnostics

- As you try different similarity functions, easy thing to check:
  - Pick any data point
  - Compute its similarity to all the other data points, and rank them in decreasing over from most similar to least similar
  - Inspect the top most similar data points — do they seem reasonable?

# Similarity Diagnostics

- As you try different similarity functions, easy thing to check:
  - Pick any data point
  - Compute its similarity to all the other data points, and rank them in decreasing over from most similar to least similar
  - Inspect the top most similar data points — do they seem reasonable?

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

# Going from Similarities to Clusters

# Going from Similarities to Clusters

There's a whole zoo of clustering methods

# Going from Similarities to Clusters

There's a whole zoo of clustering methods

Two main categories we'll talk about:

# Going from Similarities to Clusters

There's a whole zoo of clustering methods

Two main categories we'll talk about:

**Generative models**

# Going from Similarities to Clusters

There's a whole zoo of clustering methods

Two main categories we'll talk about:

**Generative models**

1. Pretend data generated by specific model with parameters

# Going from Similarities to Clusters

There's a whole zoo of clustering methods

Two main categories we'll talk about:

**Generative models**

1. Pretend data generated by specific model with parameters

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

# Going from Similarities to Clusters

There's a whole zoo of clustering methods

Two main categories we'll talk about:

**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

# Going from Similarities to Clusters

There's a whole zoo of clustering methods

Two main categories we'll talk about:

**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

**Hierarchical clustering**

# Going from Similarities to Clusters

There's a whole zoo of clustering methods

Two main categories we'll talk about:

**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

**Hierarchical clustering**

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

# Going from Similarities to Clusters

There's a whole zoo of clustering methods

Two main categories we'll talk about:

**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

**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

# Going from Similarities to Clusters

There's a whole zoo of clustering methods

Two main categories we'll talk about:

**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 start here

**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

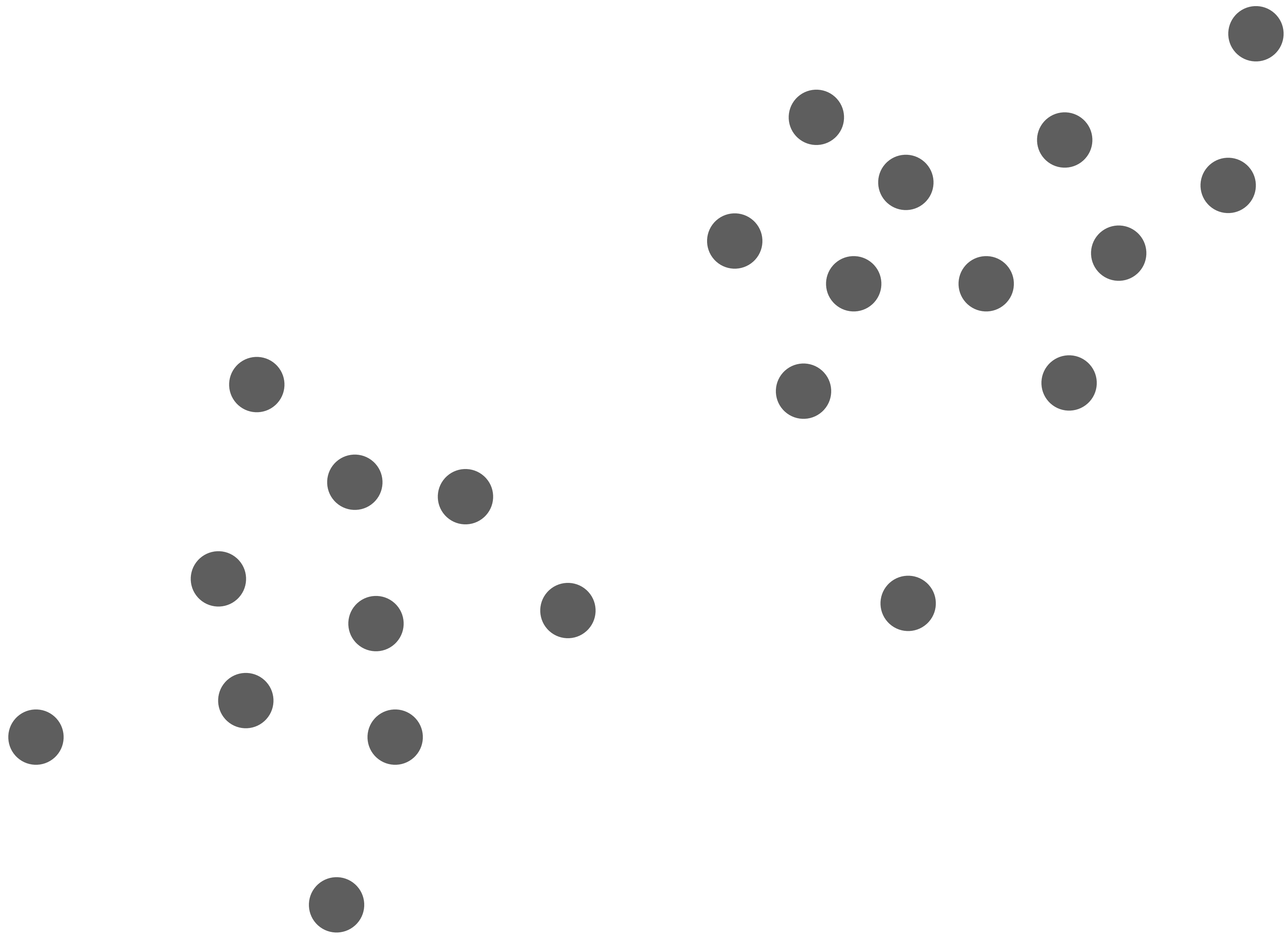
We're going to start with perhaps the most famous of clustering methods

# 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

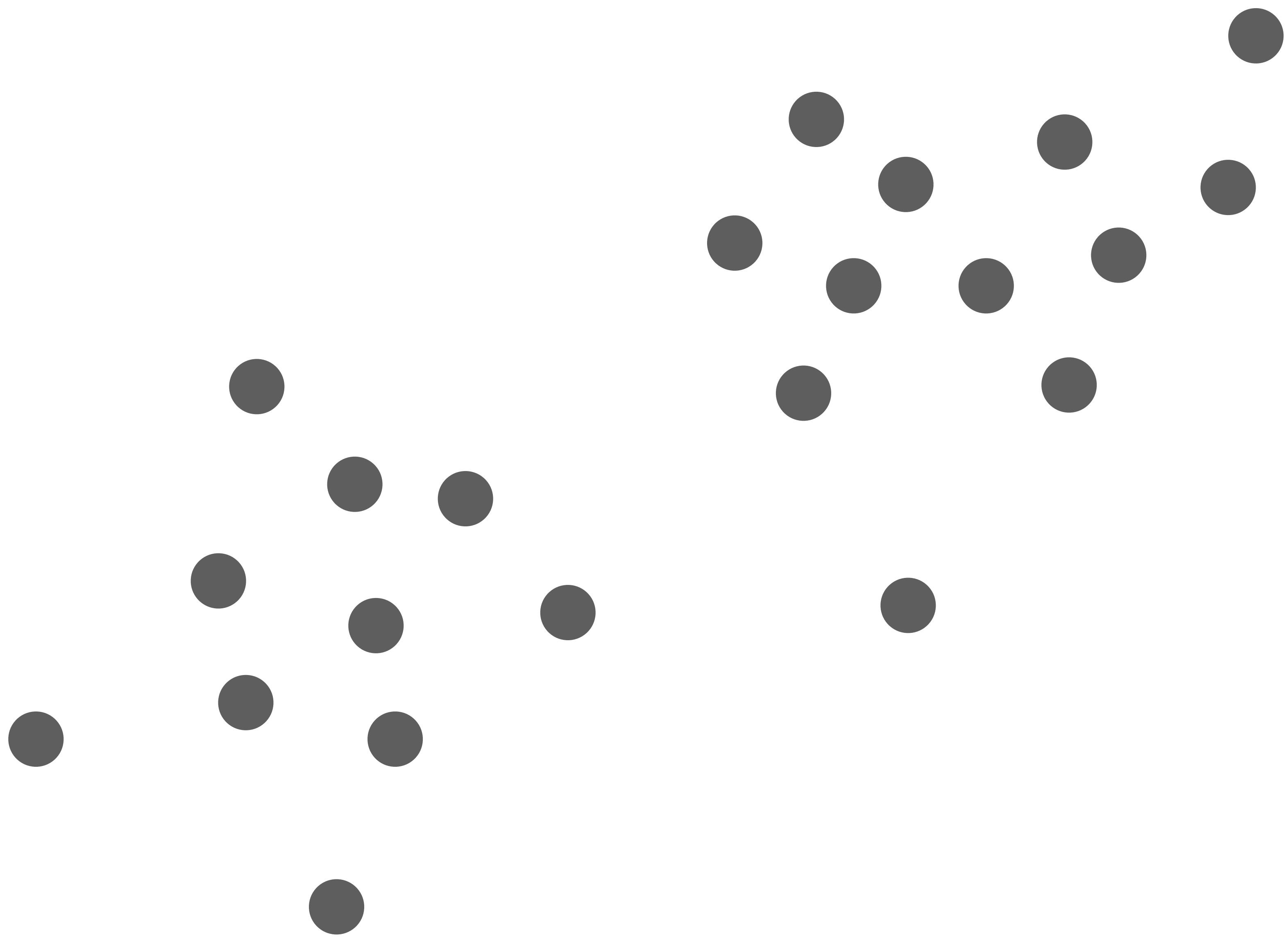
# *k*-means

# *k*-means



# *k*-means

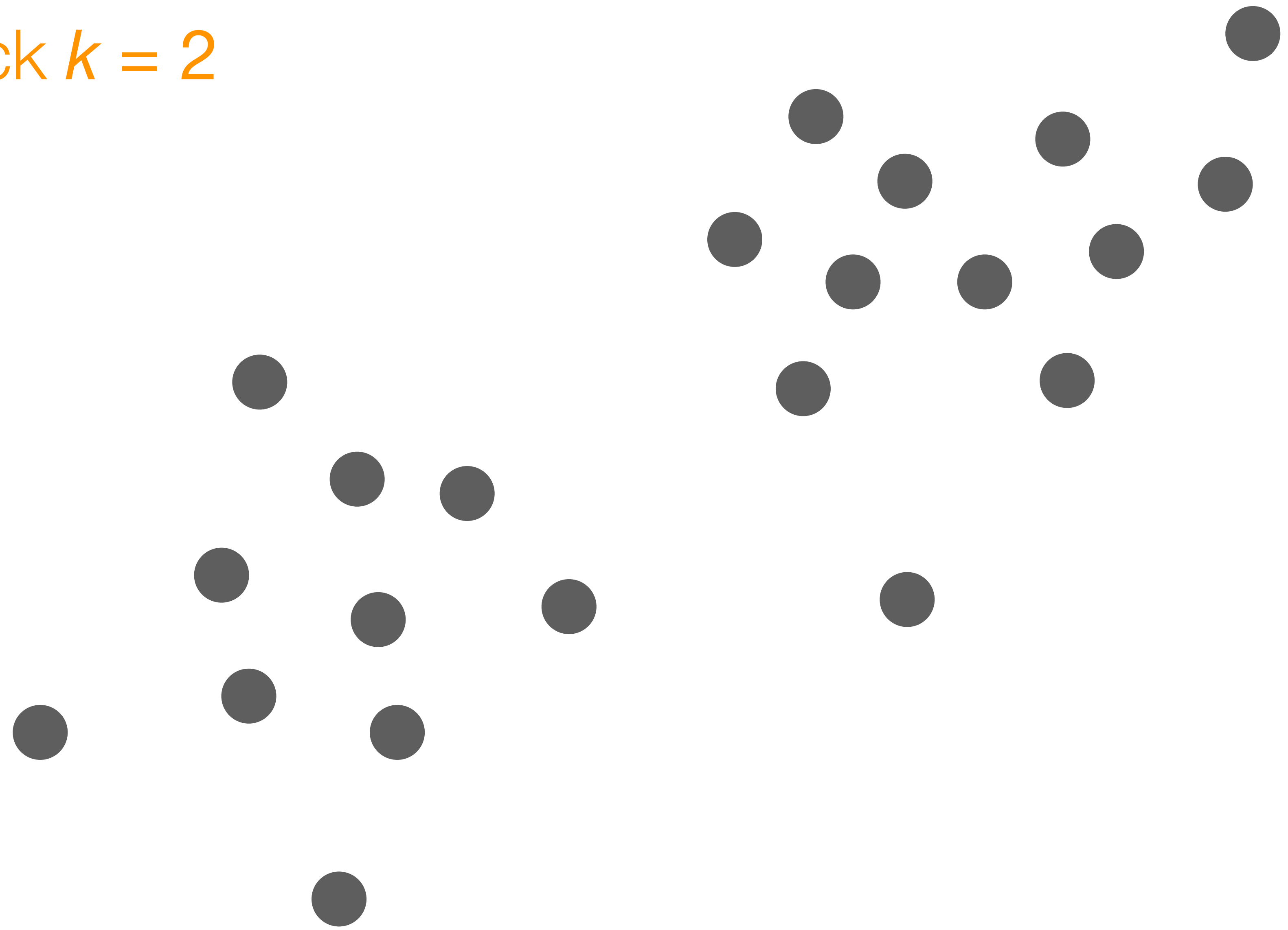
Step 0: Pick *k*



# *k*-means

Step 0: Pick *k*

We'll pick $k = 2$

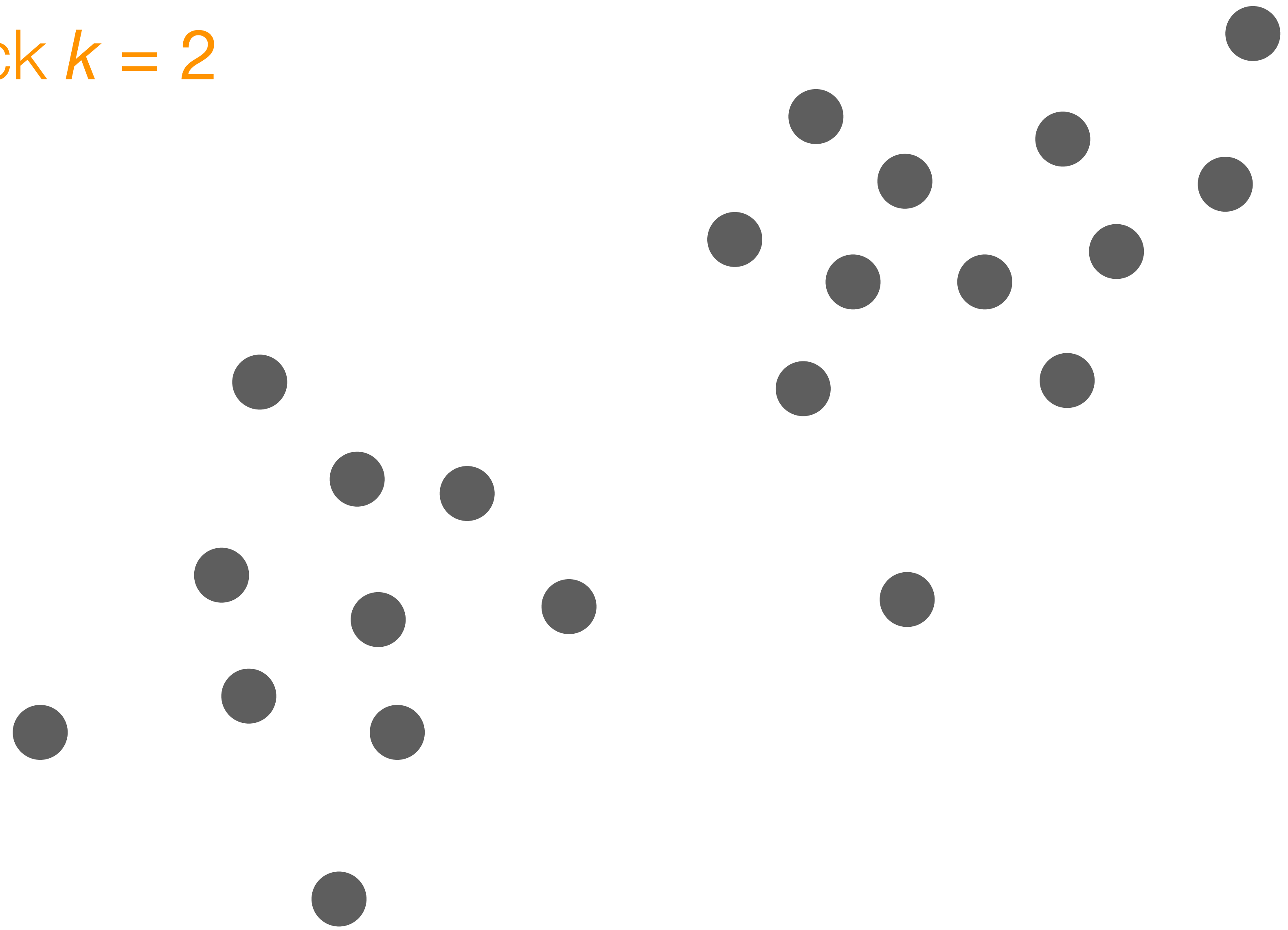


# *k*-means

Step 0: Pick *k*

We'll pick *k* = 2

Step 1: Pick guesses for where cluster centers are

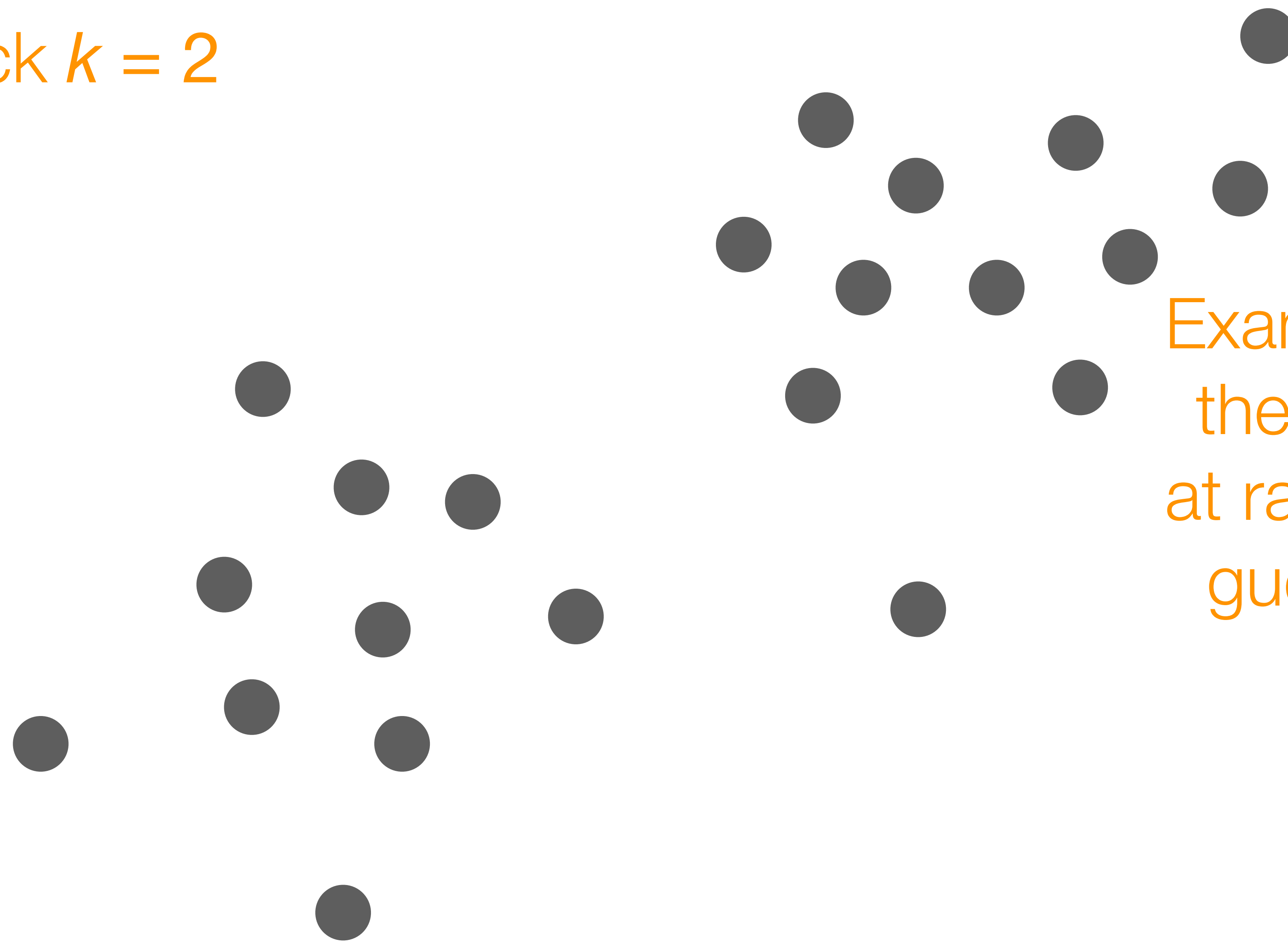


# *k*-means

Step 0: Pick *k*

We'll pick *k* = 2

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



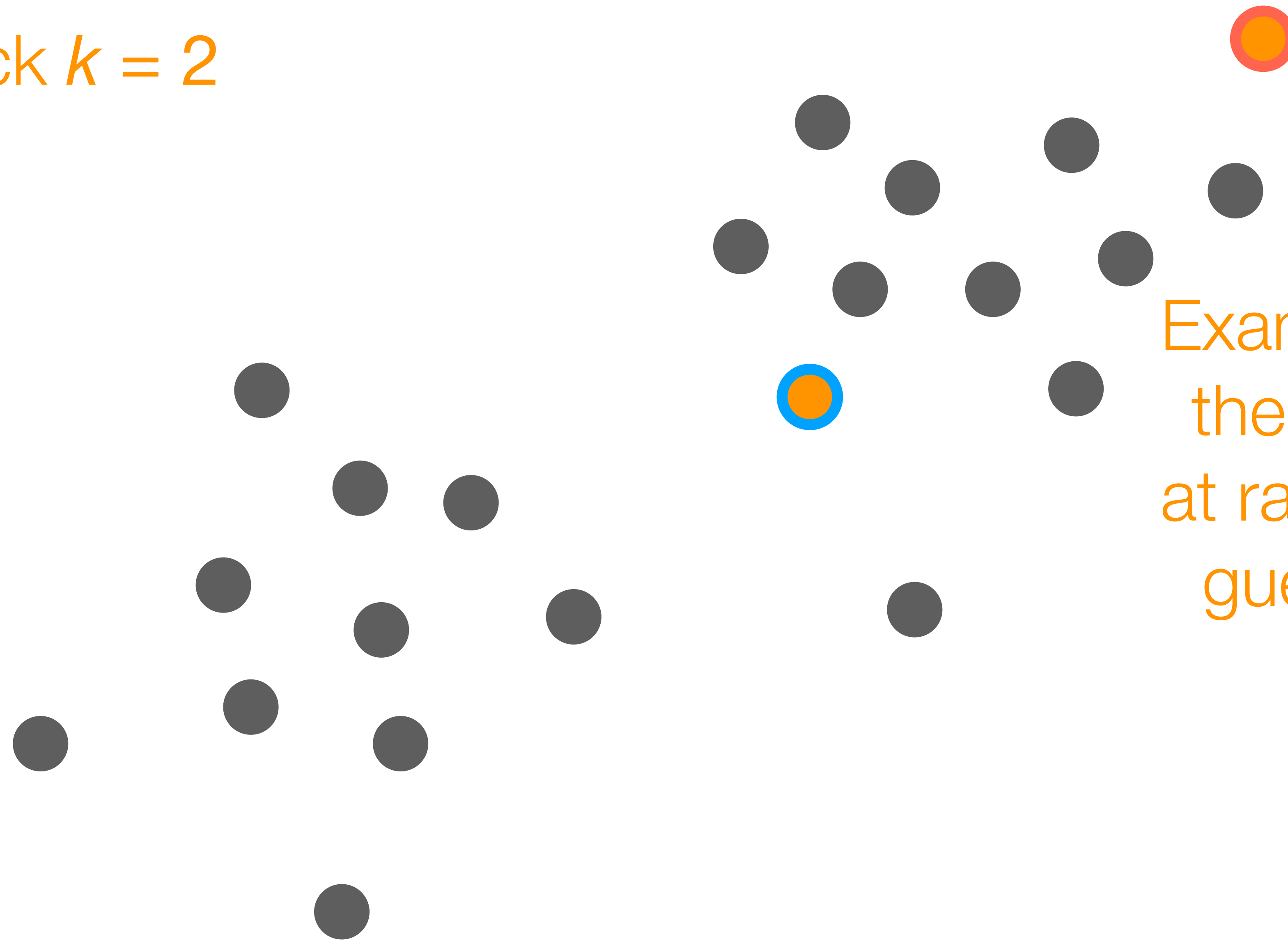
Example: choose *k* of the points uniformly at random to be initial guesses for cluster centers

# *k*-means

Step 0: Pick *k*

We'll pick *k* = 2

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



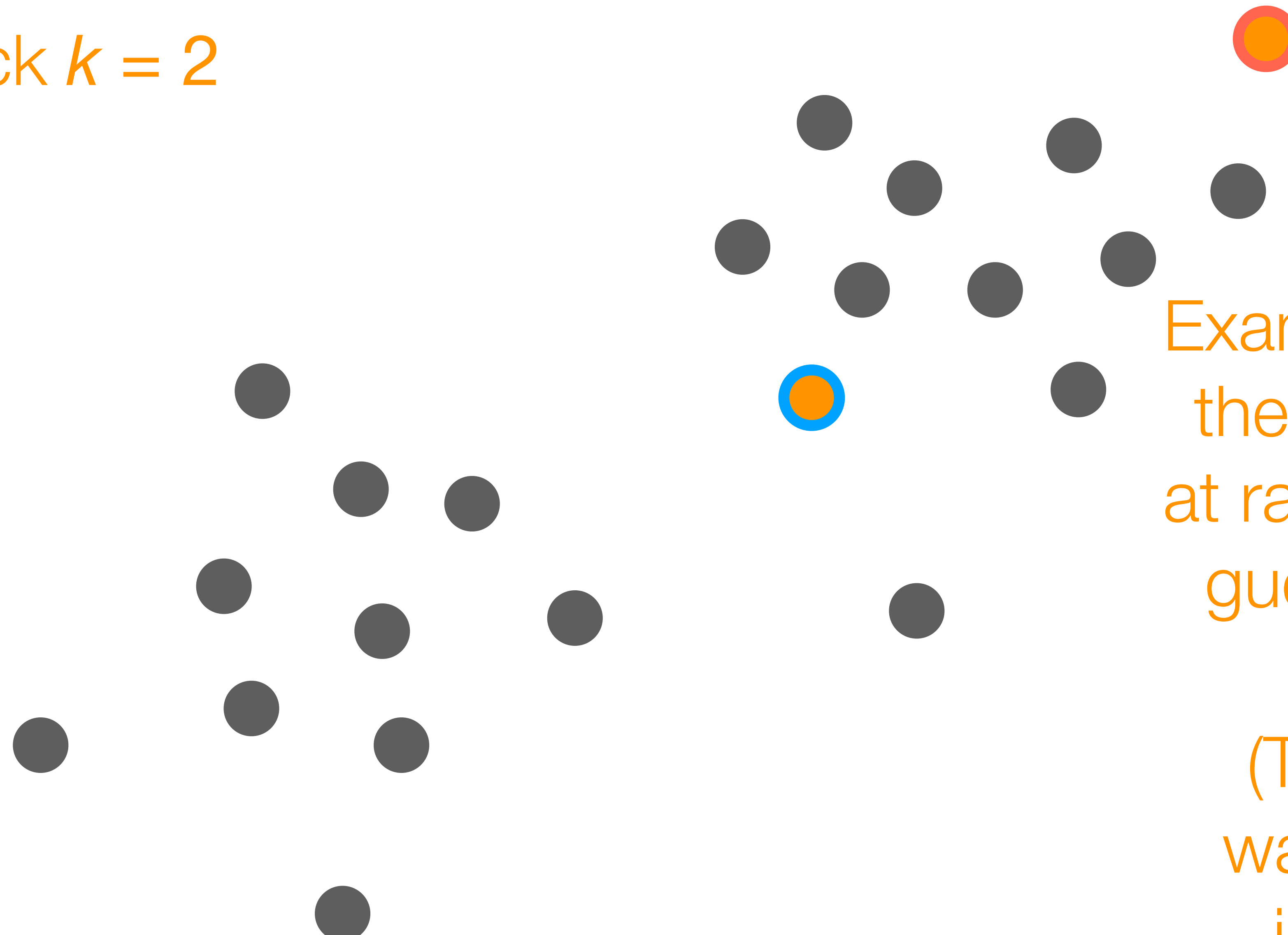
Example: choose *k* of the points uniformly at random to be initial guesses for cluster centers

# *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

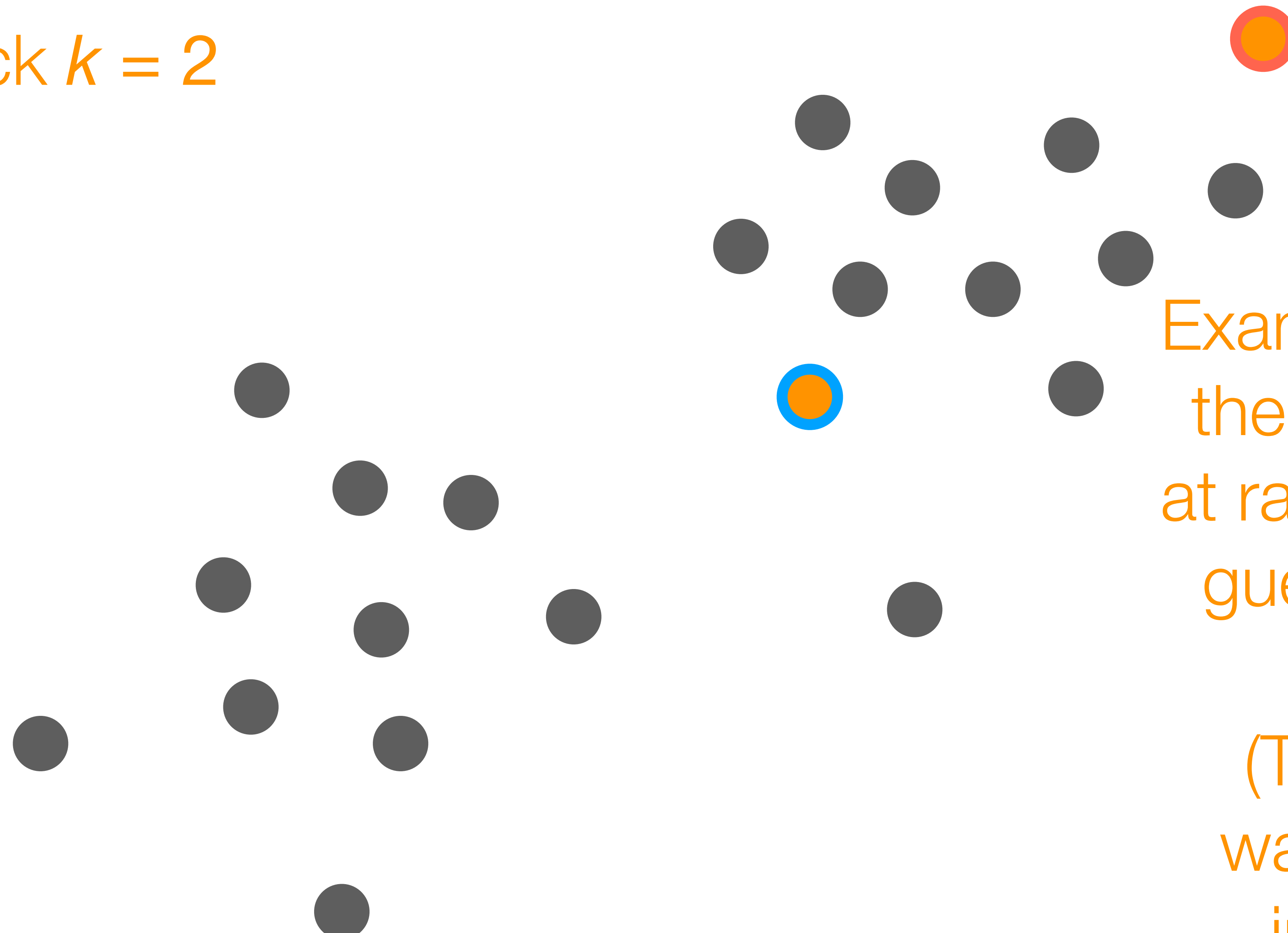
(There are many ways to make the initial guesses)

# *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)

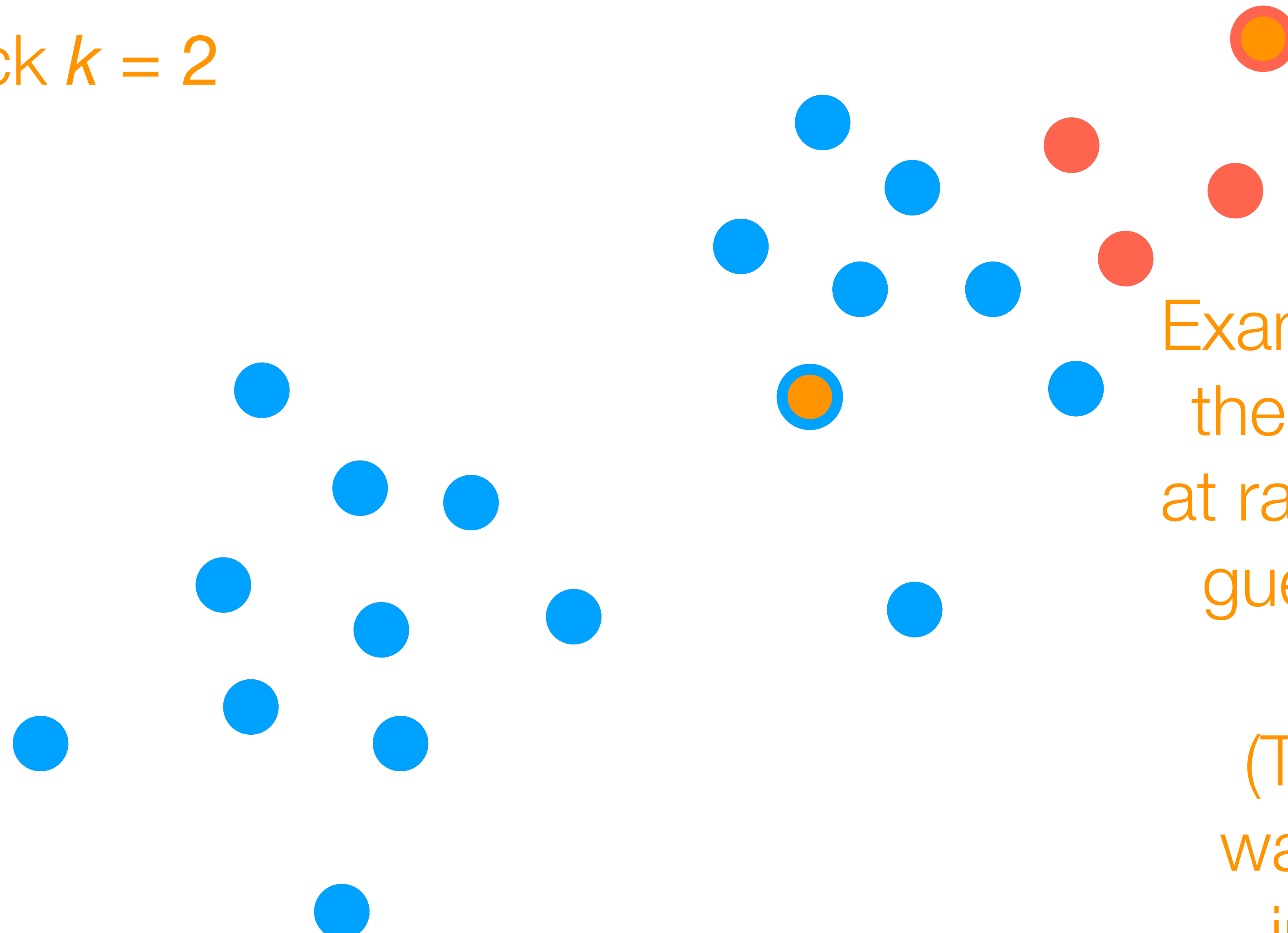
Step 2: Assign each point to belong to the closest cluster

# *k*-means

Step 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)

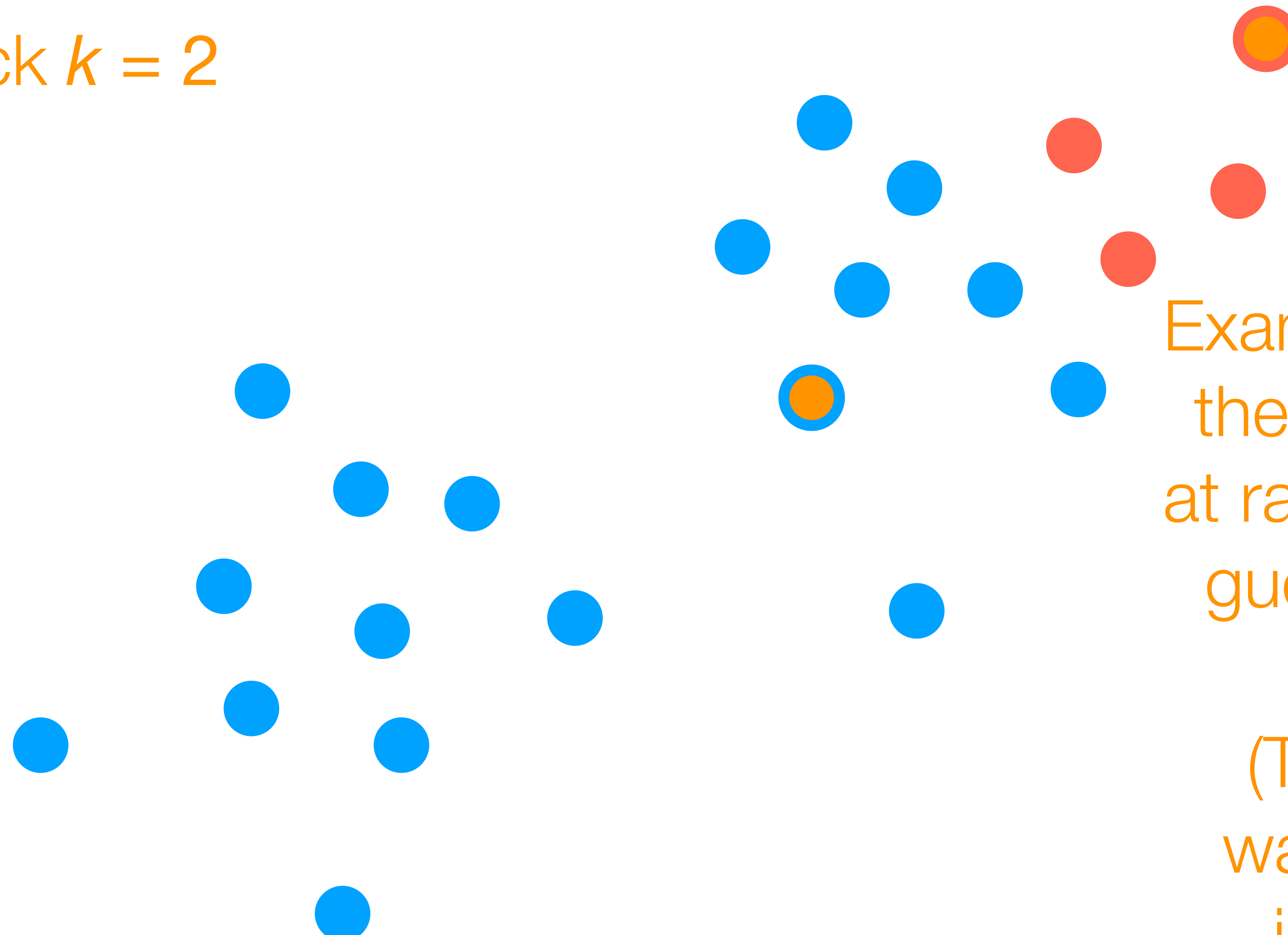
Step 2: Assign each point to belong to the closest cluster

# *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

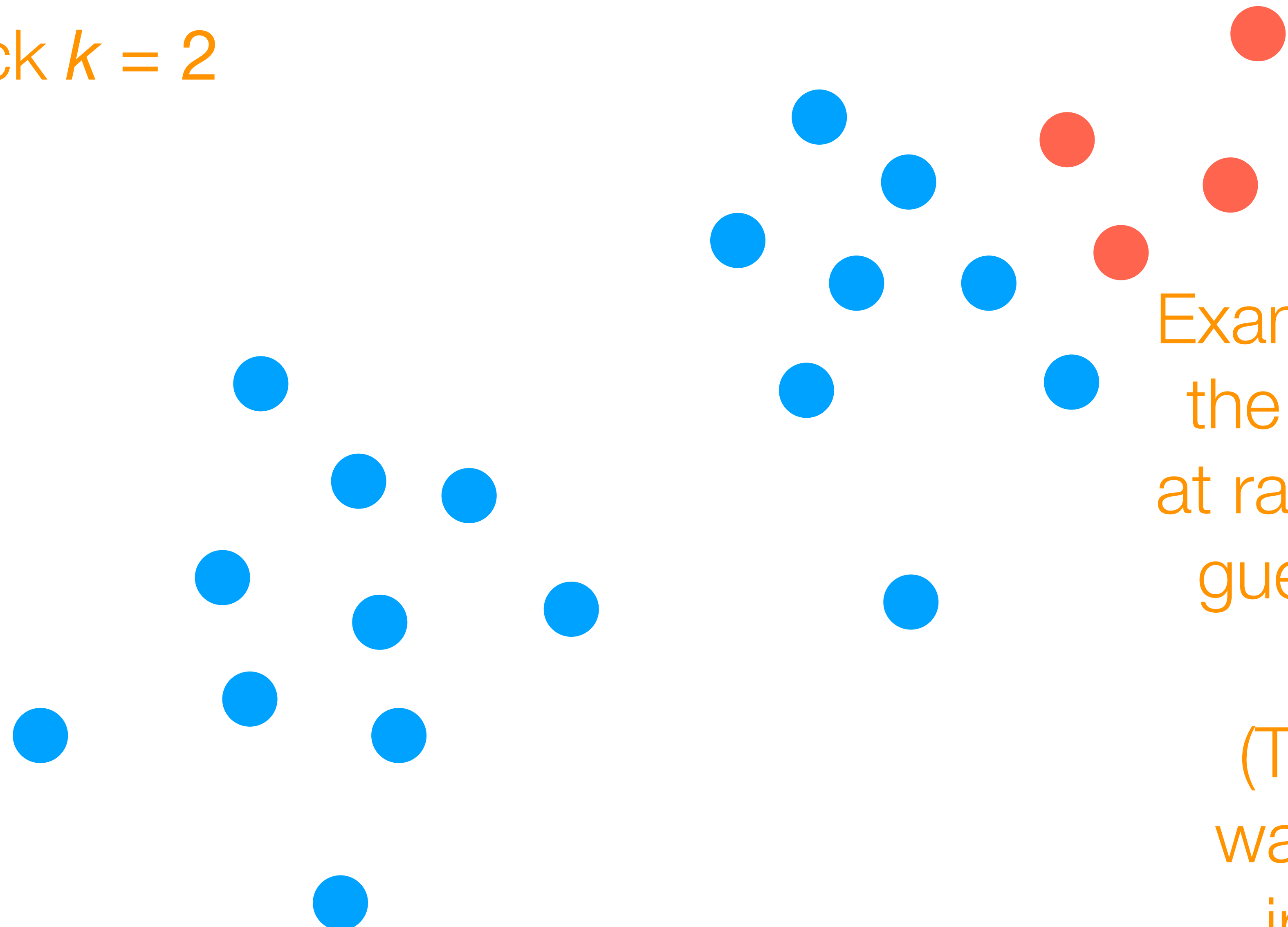
Step 3: Update cluster means (to be the center of mass per cluster)

# *k*-means

Step 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)

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

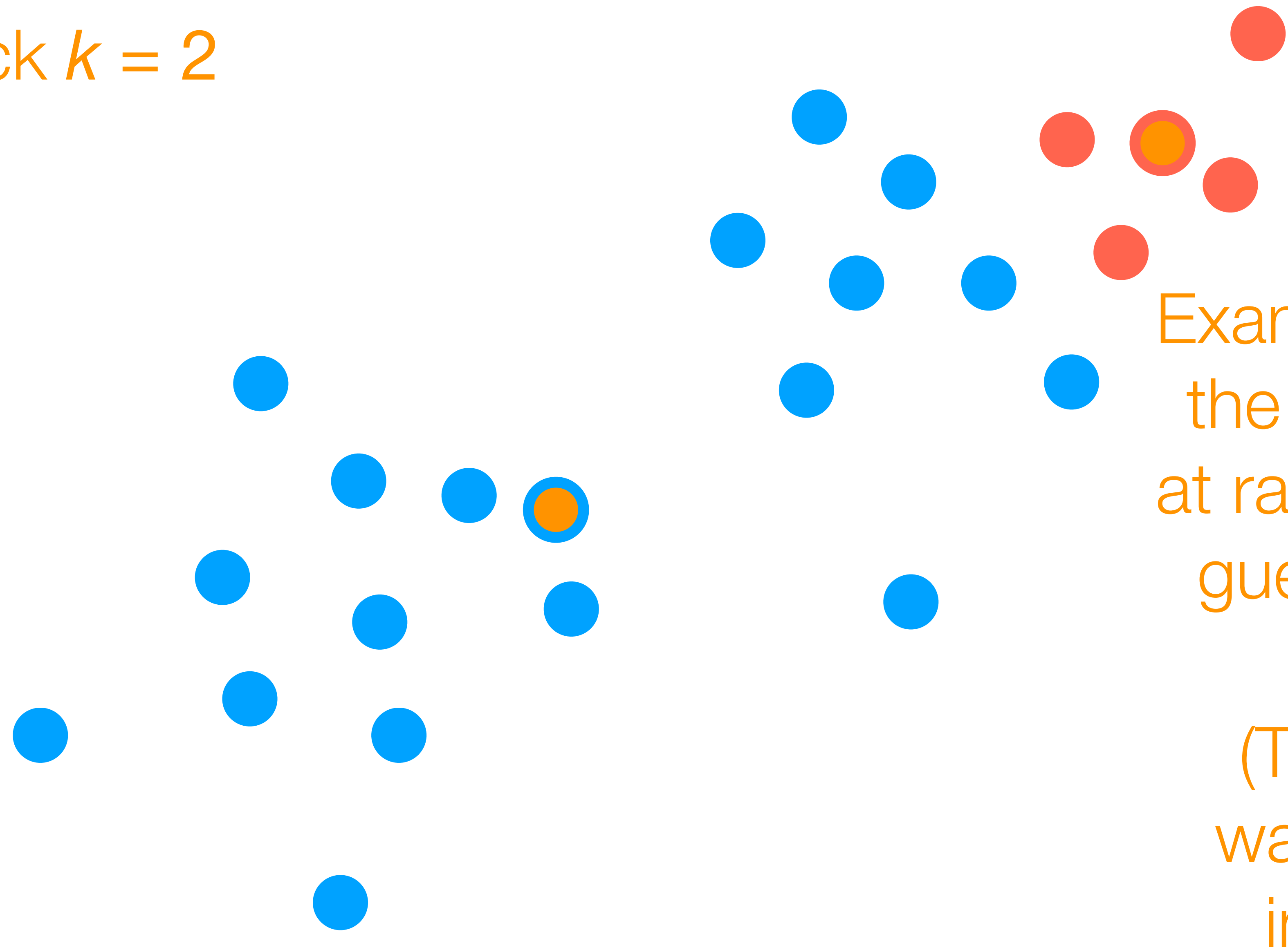
Step 3: Update cluster means (to be the center of mass per cluster)

# *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)

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

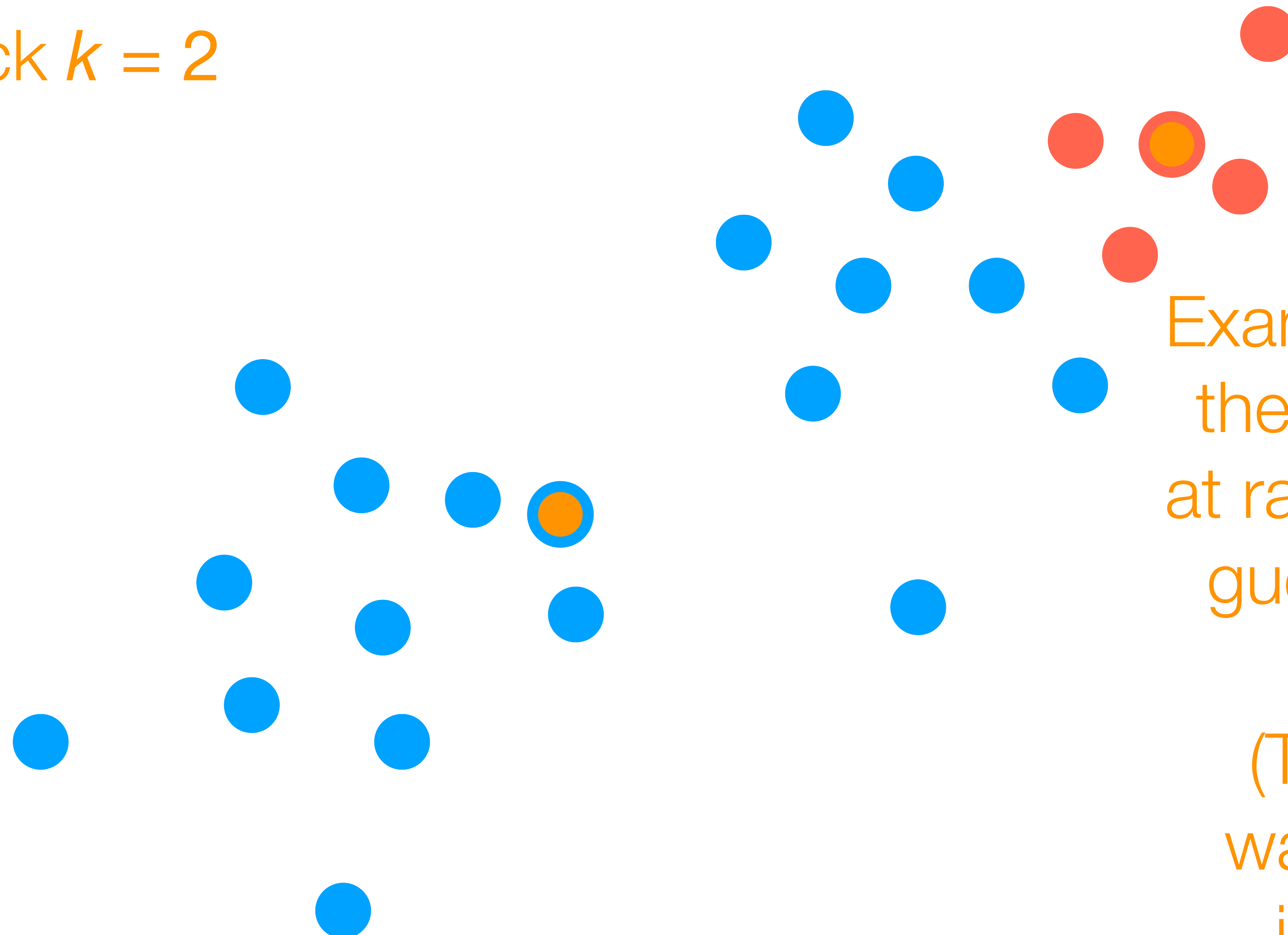
Step 3: Update cluster means (to be the center of mass per cluster)

# *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)

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

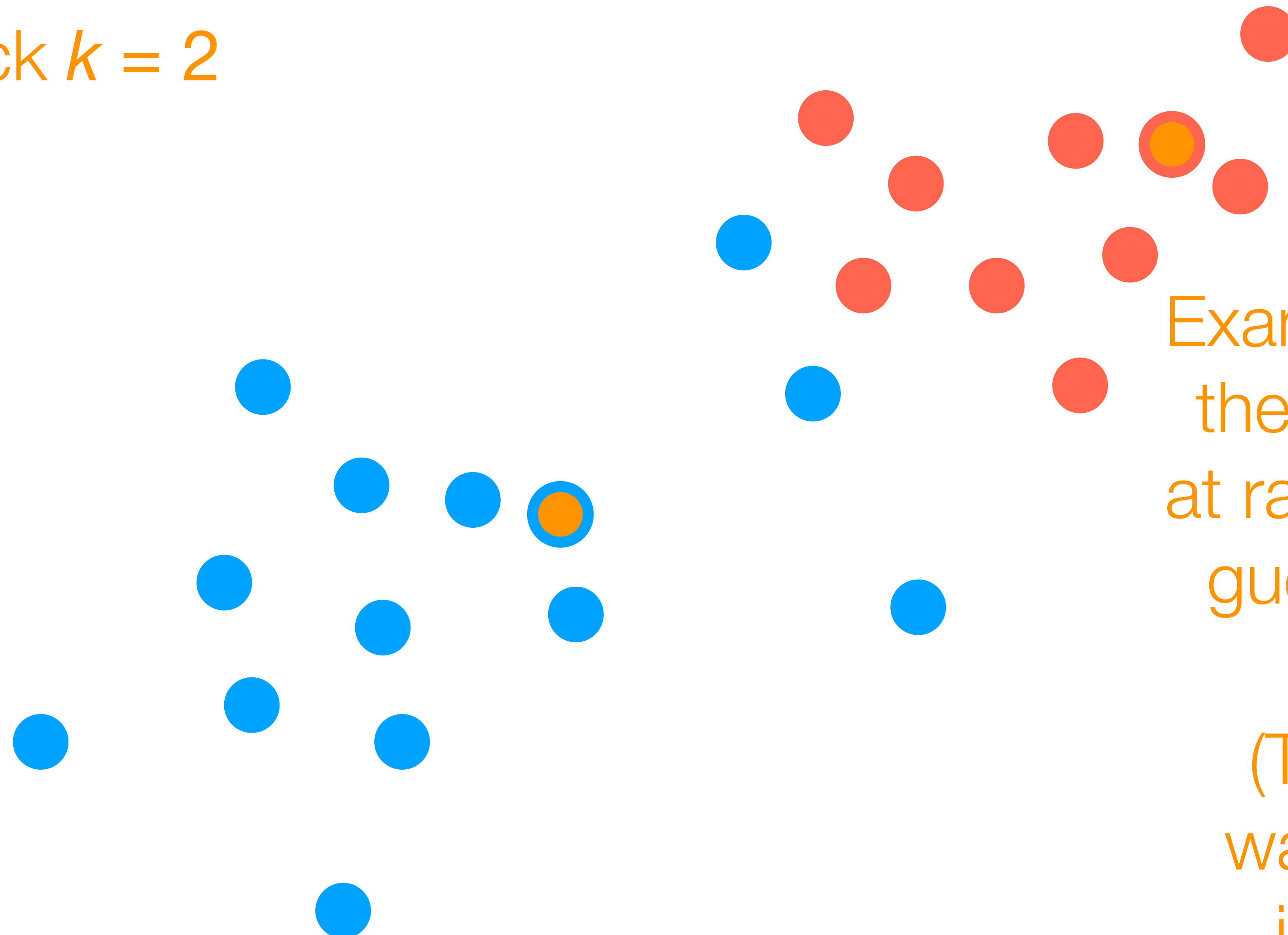
Step 3: Update cluster means (to be the center of mass per cluster)

# *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)

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

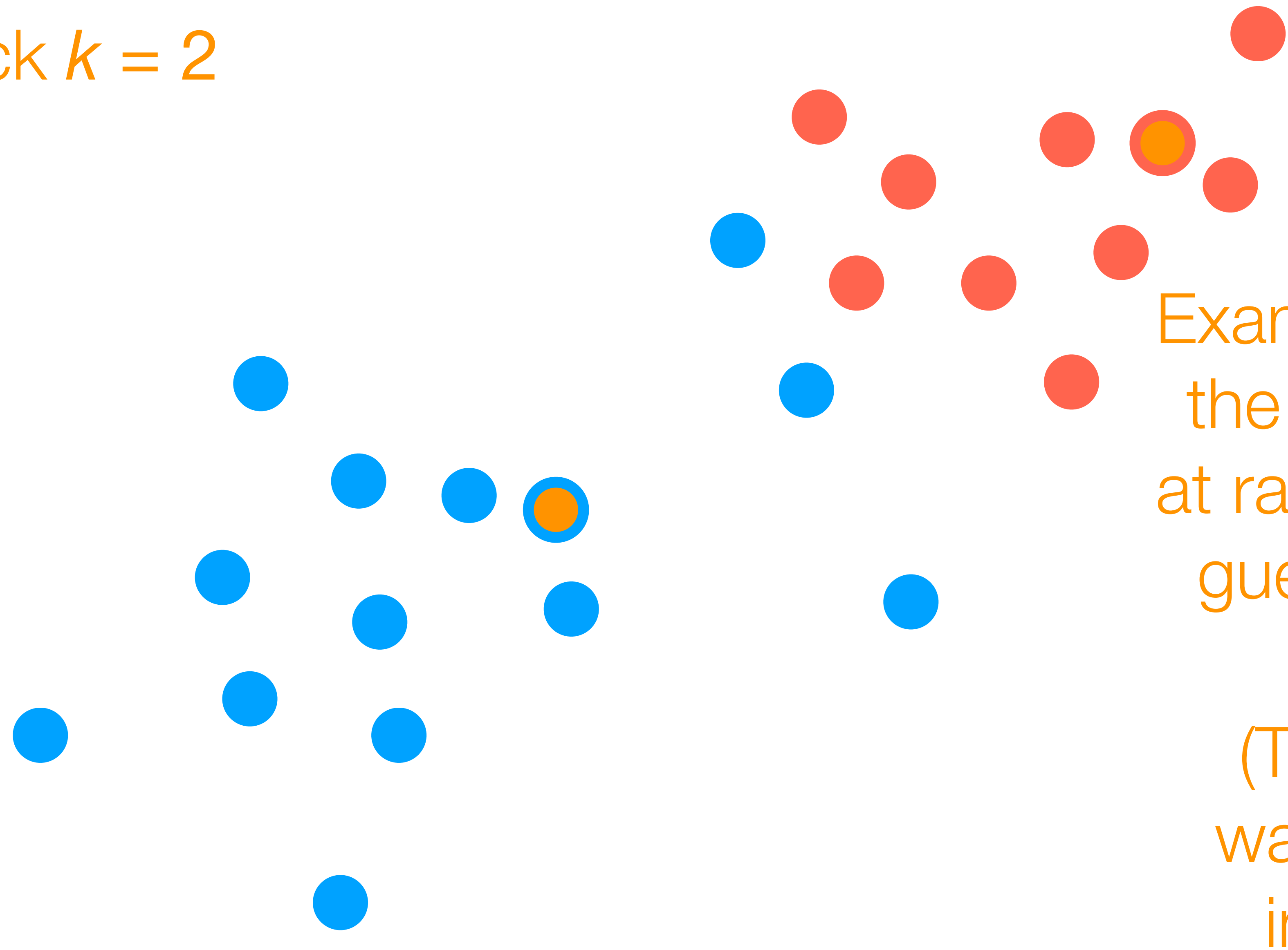
Step 3: Update cluster means (to be the center of mass per cluster)

# *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**

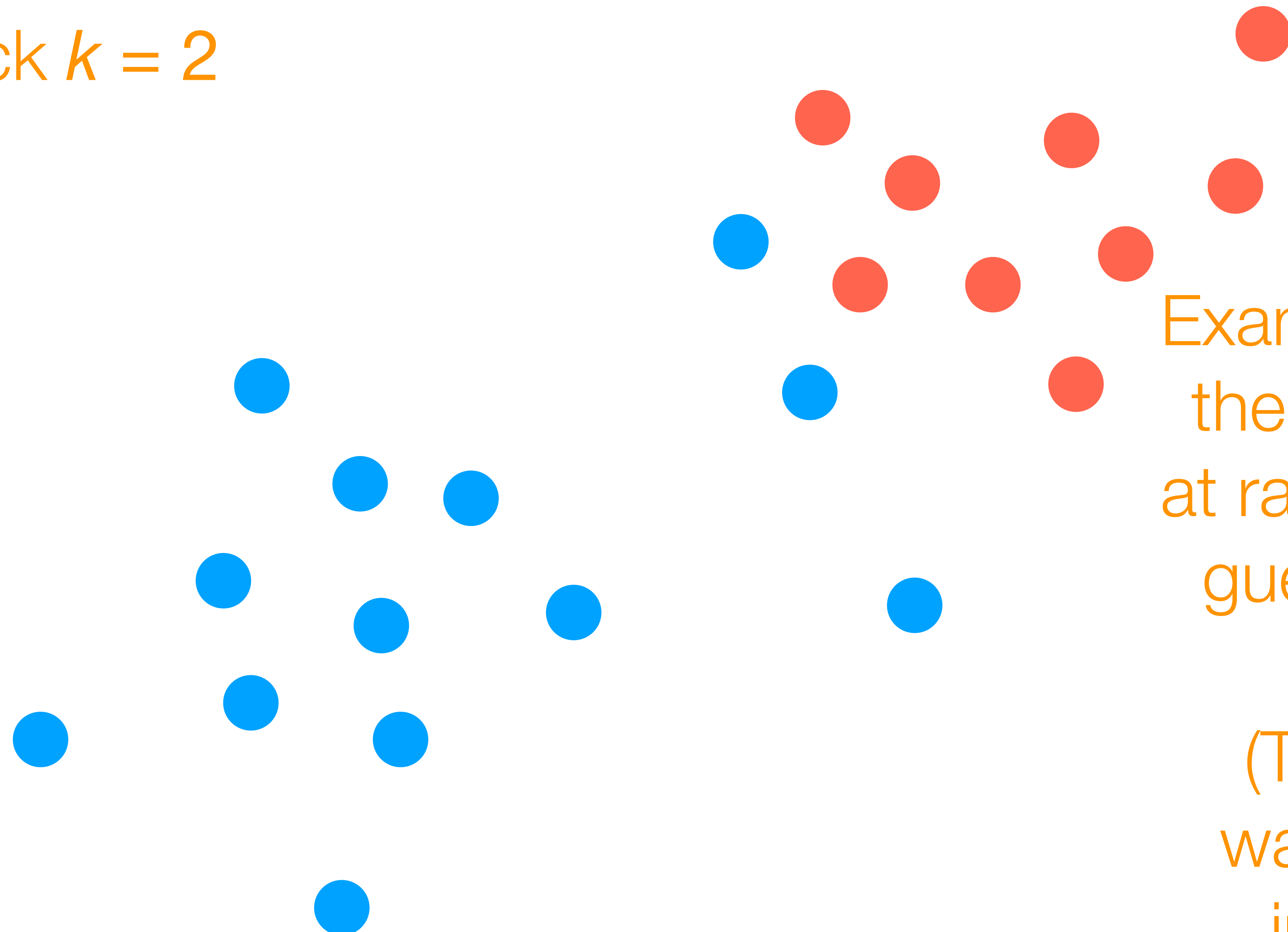
Step 3: Update cluster means (to be the center of mass per cluster)

# *k*-means

Step 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**

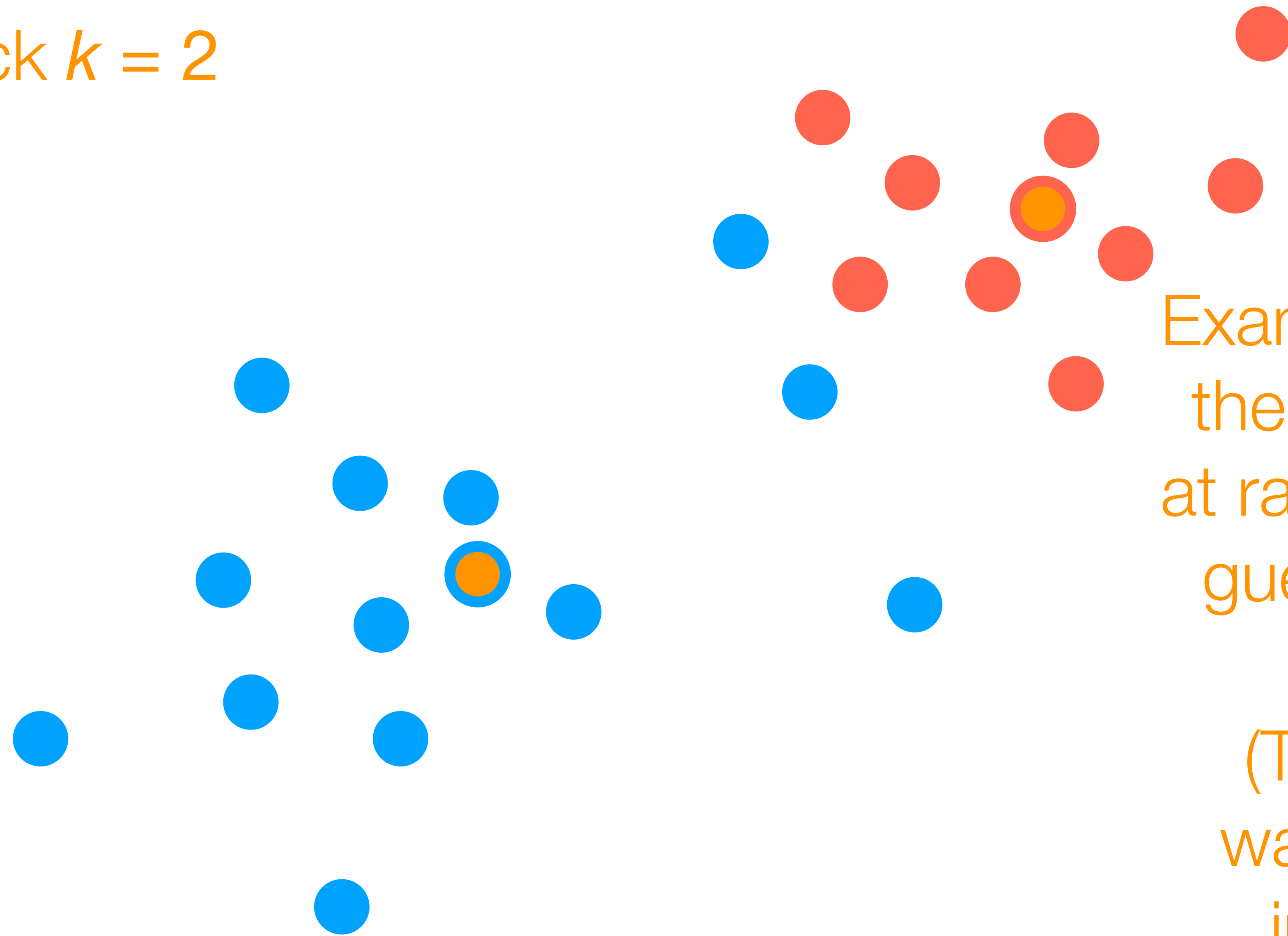
Step 3: Update cluster means (to be the center of mass per cluster)

# *k*-means

Step 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)

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

**Repeat**

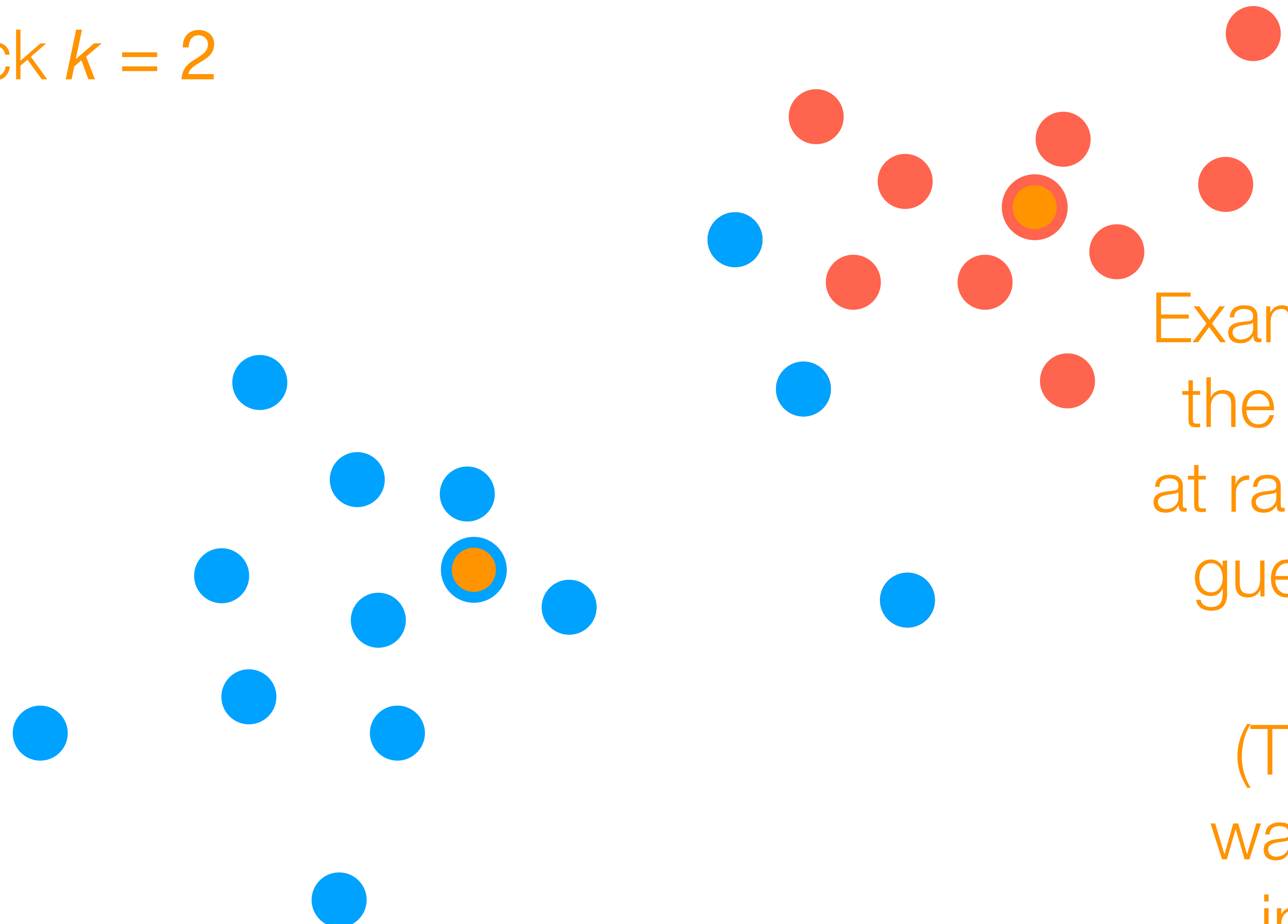
Step 3: Update cluster means (to be the center of mass per cluster)

# *k*-means

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

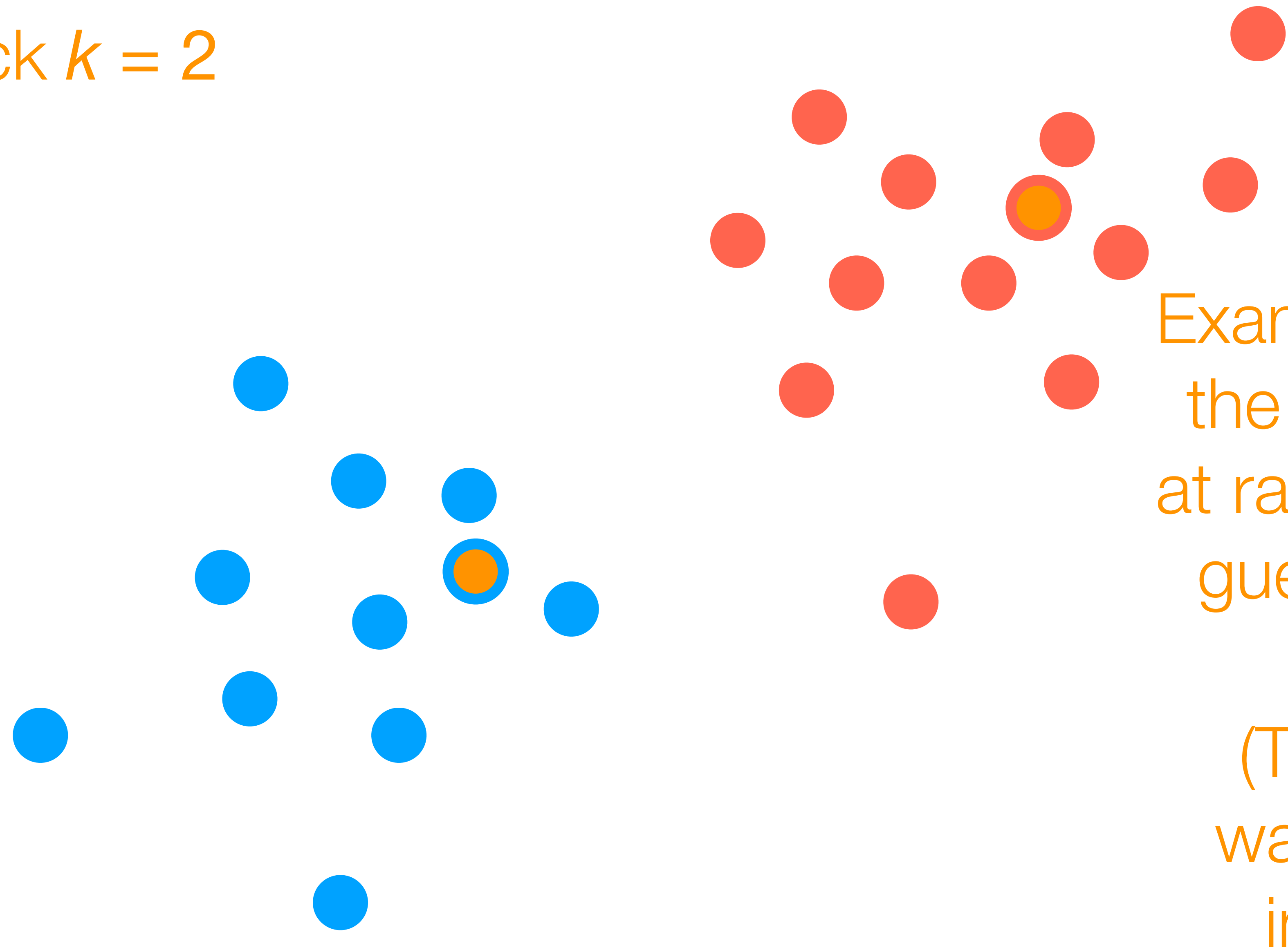
Step 3: Update cluster means (to be the center of mass per cluster)

# *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)

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

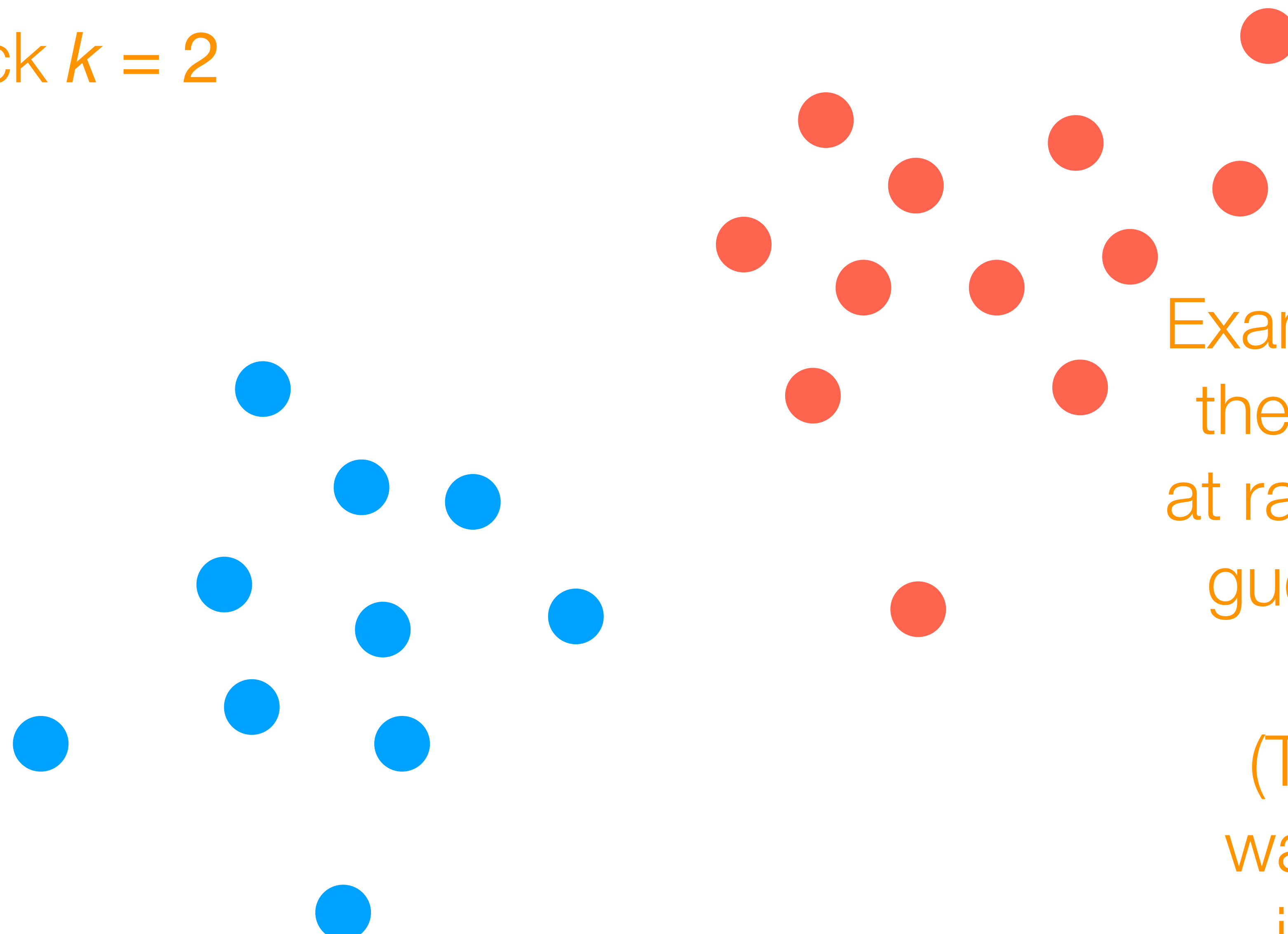
Step 3: Update cluster means (to be the center of mass per cluster)

# *k*-means

Step 0: Pick *k*

We'll pick *k* = 2

Step 1: Pick 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)

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

**Repeat**

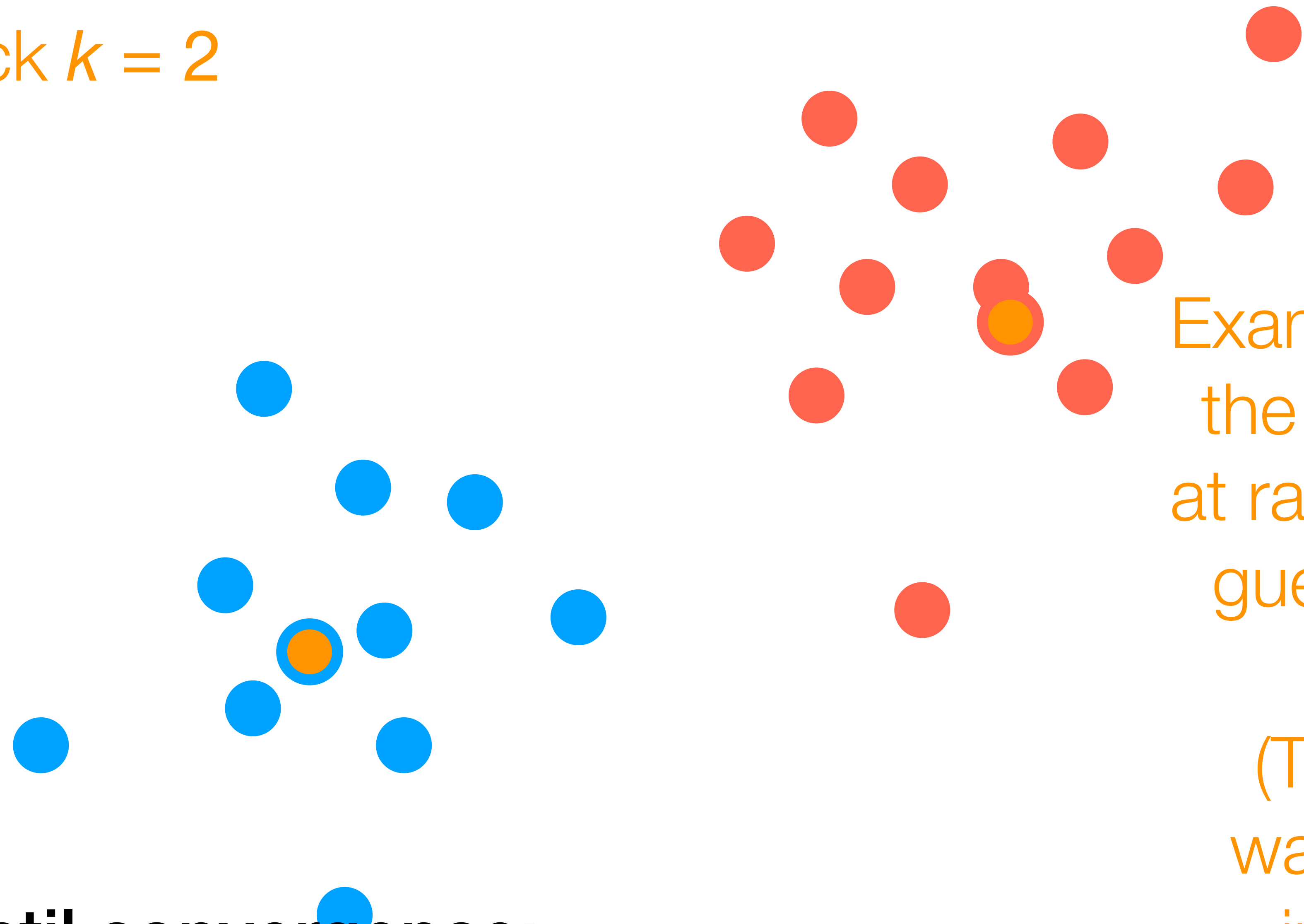
Step 3: Update cluster means (to be the center of mass per cluster)

# *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

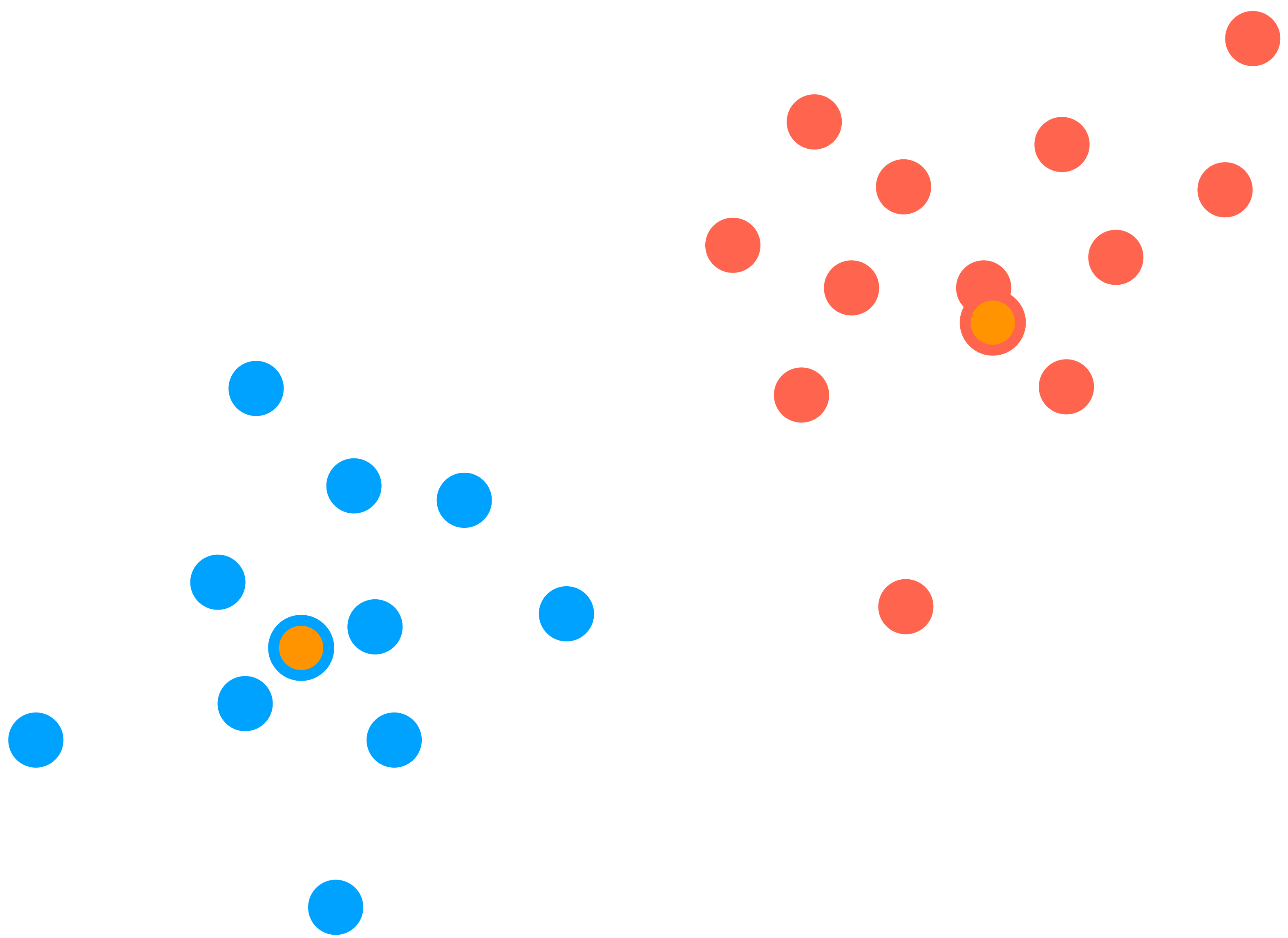
(There are many ways to make the initial guesses)

**Repeat until convergence:**

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

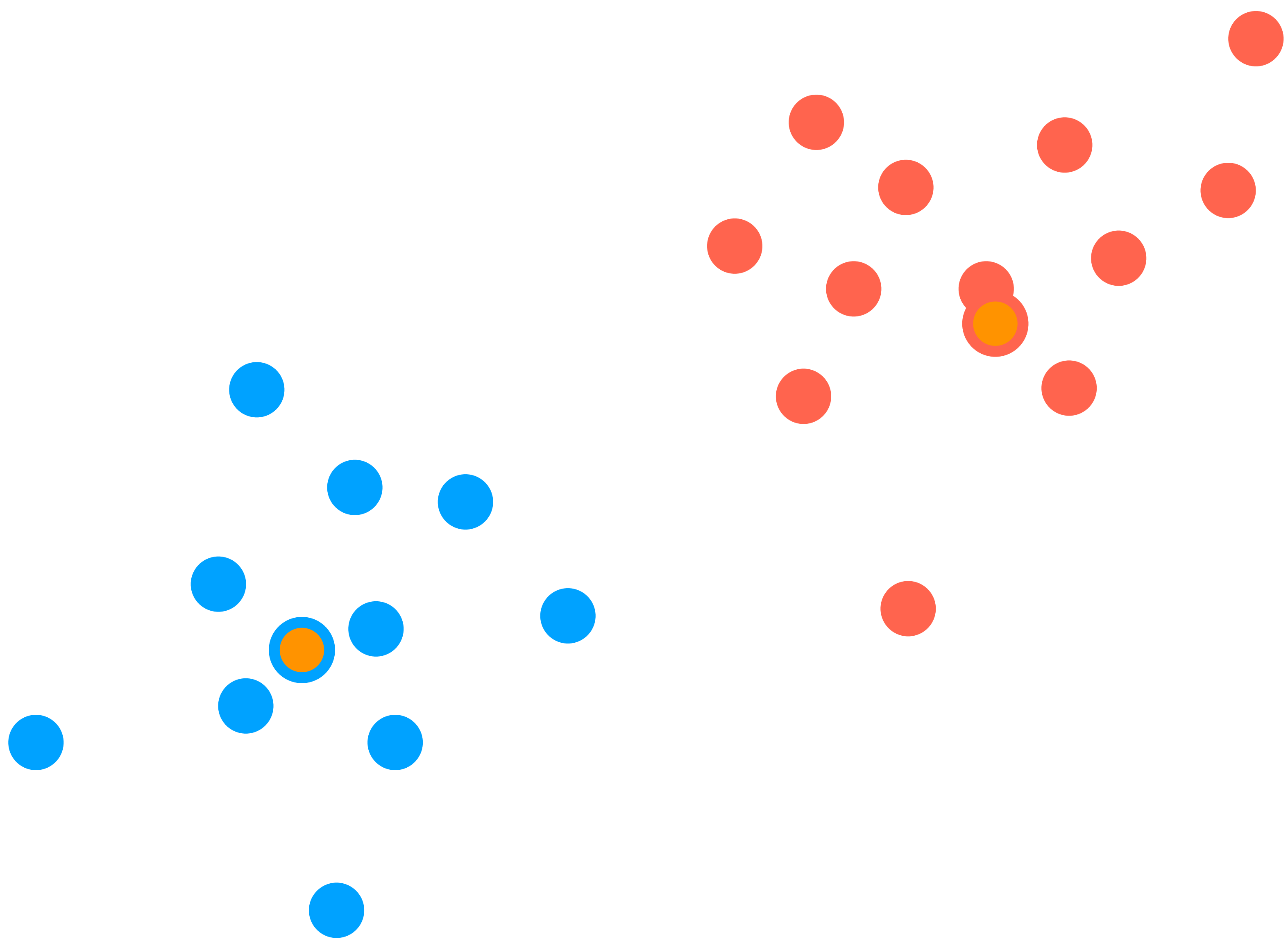
Step 3: Update cluster means (to be the center of mass per cluster)

# *k*-means



# *k*-means

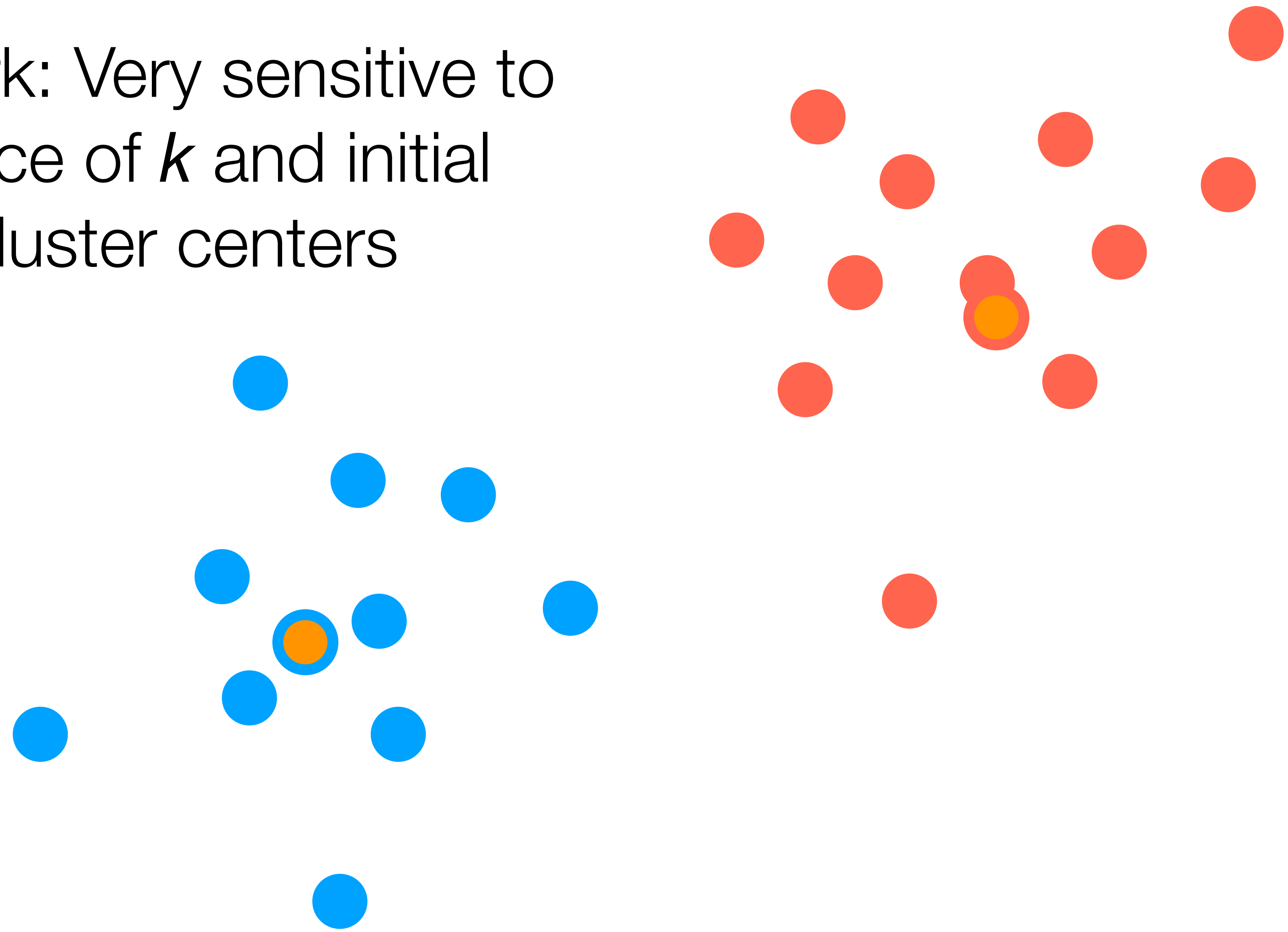
Final output: cluster centers, cluster assignment for every point



# *k*-means

Final output: cluster centers, cluster assignment for every point

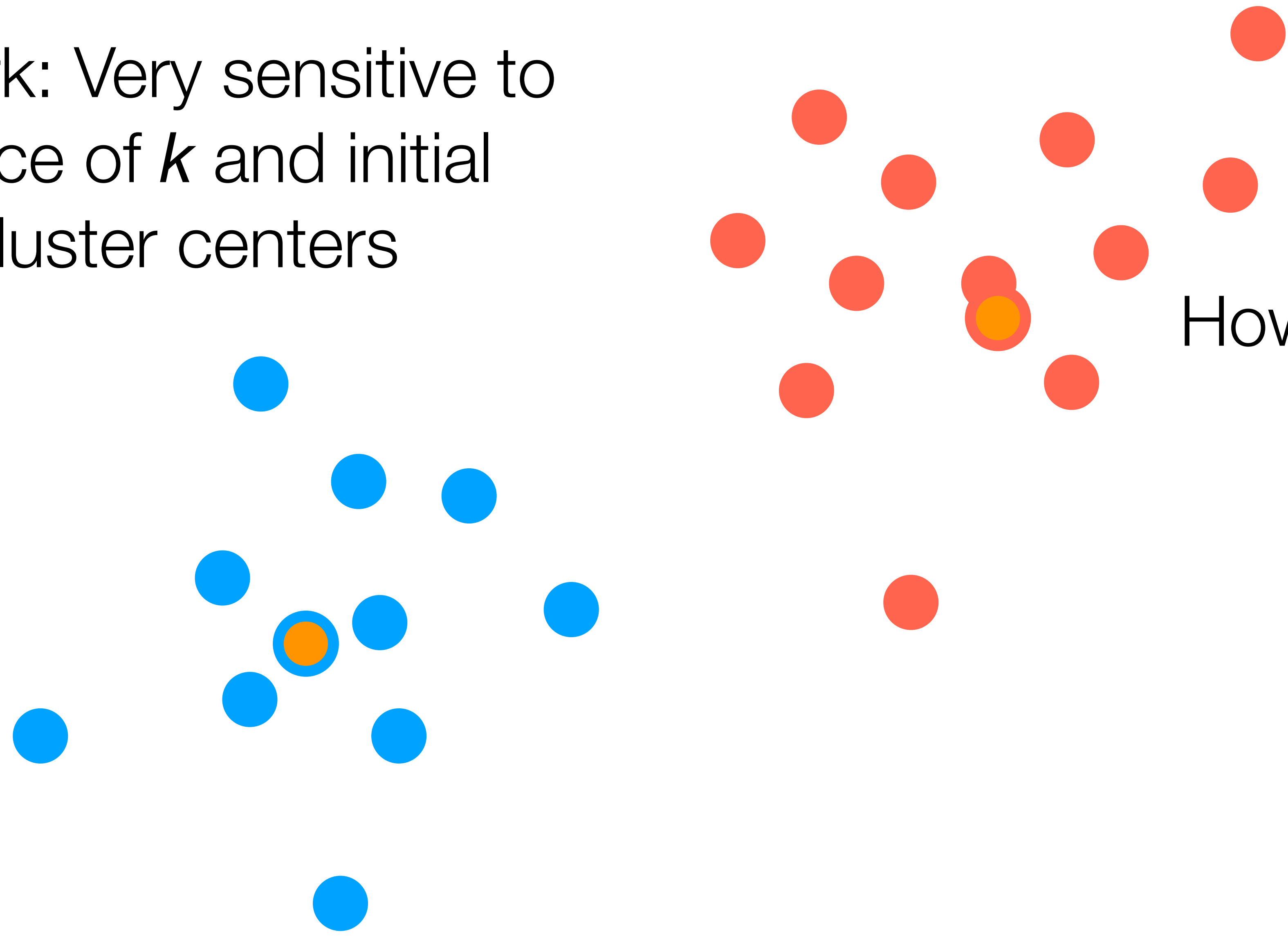
Remark: Very sensitive to choice of *k* and initial cluster centers



# *k*-means

Final output: cluster centers, cluster assignment for every point

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

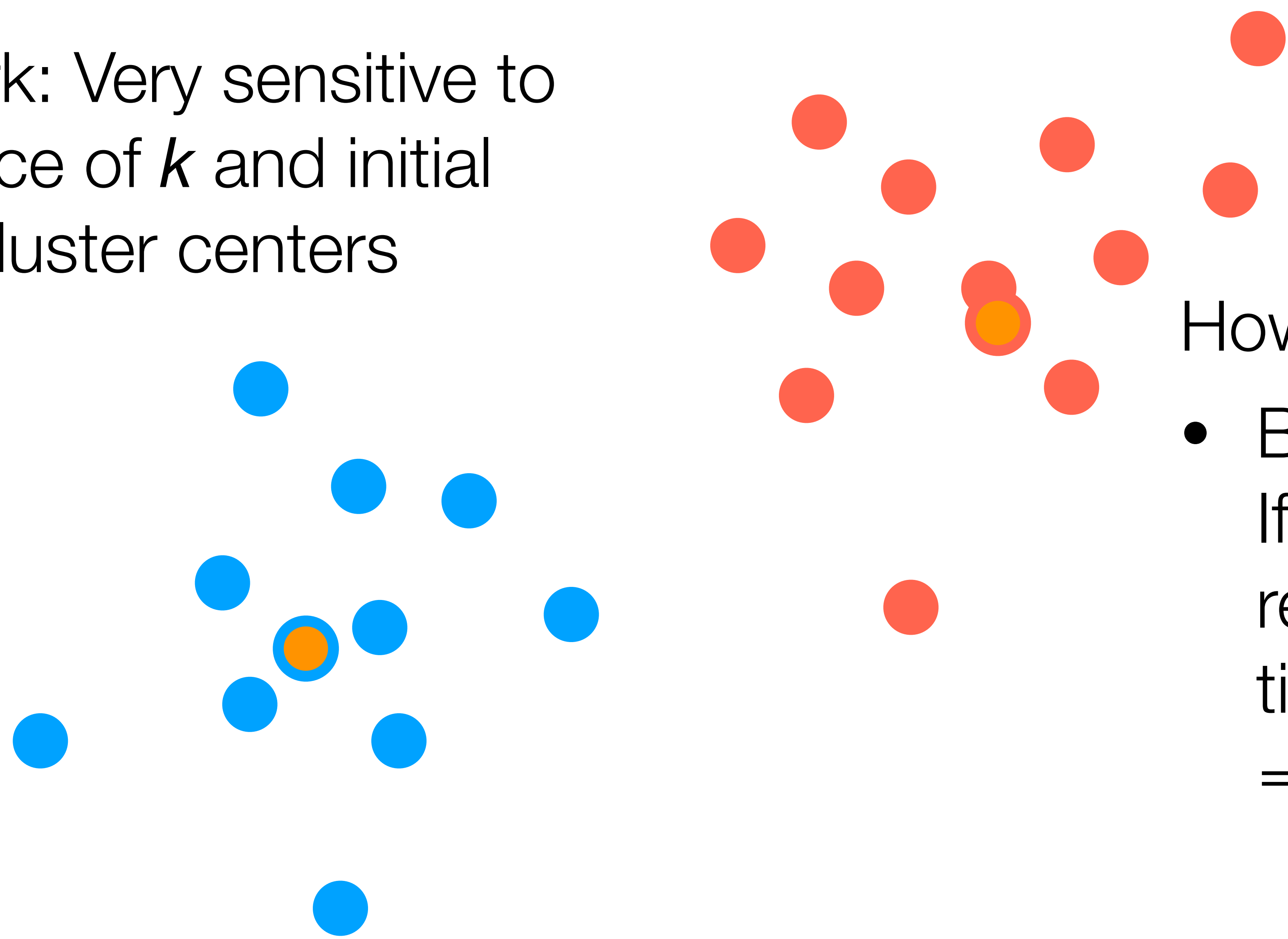


How to pick *k*?

# *k*-means

Final output: cluster centers, cluster assignment for every point

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



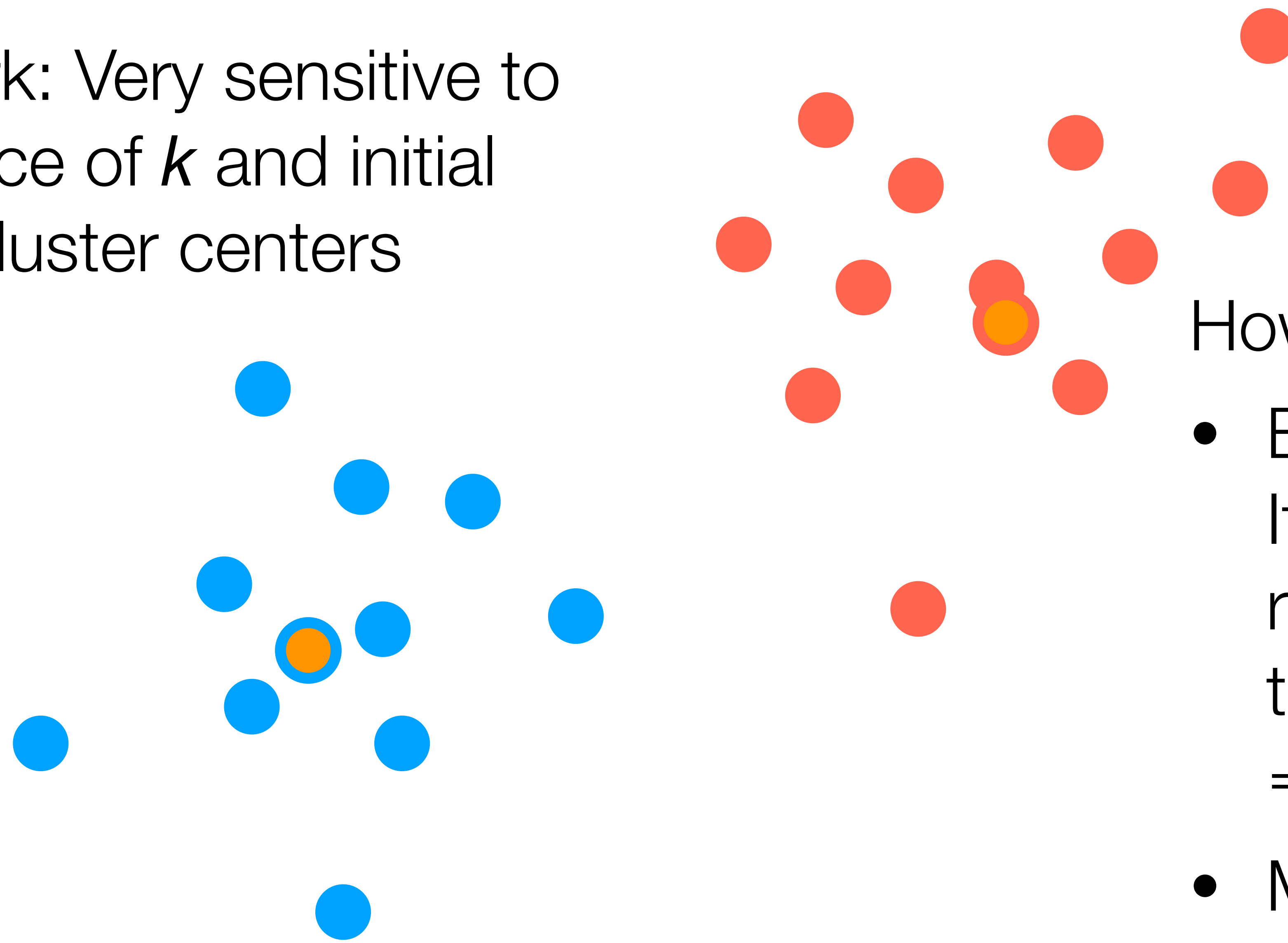
How to pick *k*?

- Basic check: If you have really, really tiny clusters ⇒ decrease *k*

# *k*-means

Final output: cluster centers, cluster assignment for every point

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



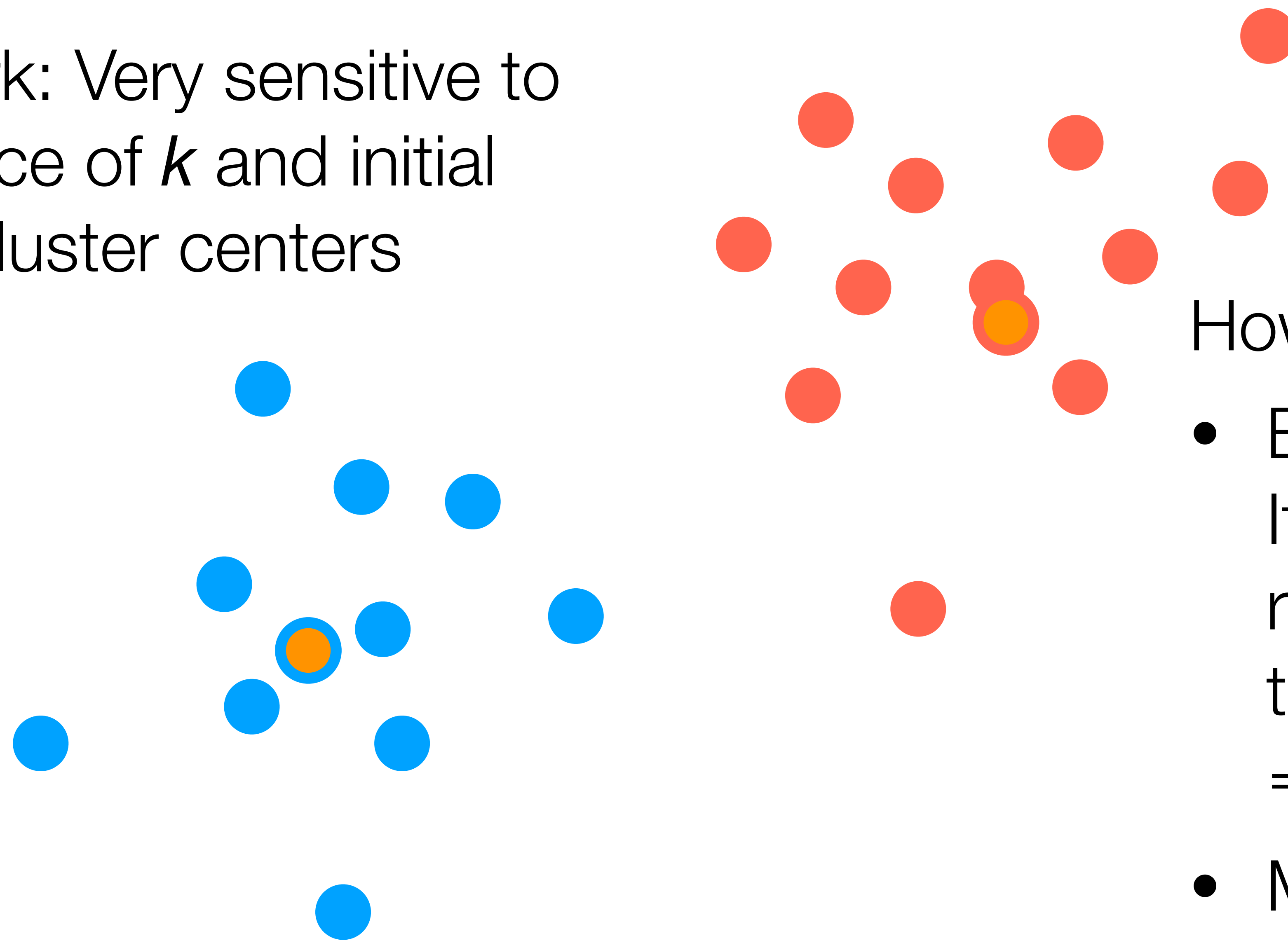
How to pick *k*?

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

# *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*?

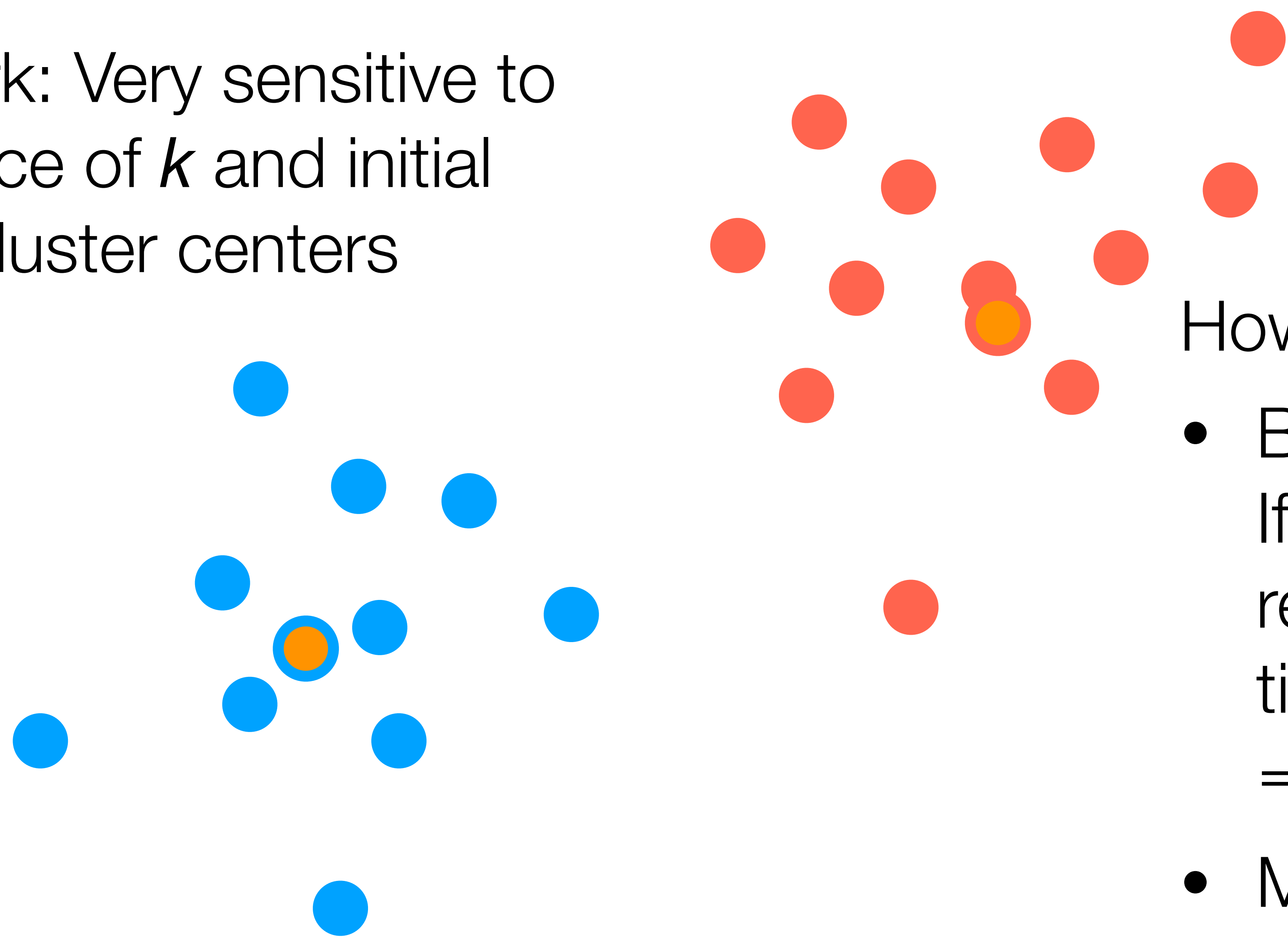
- Basic check: If you have really, really tiny clusters ⇒ decrease *k*
- More details later

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

# *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)

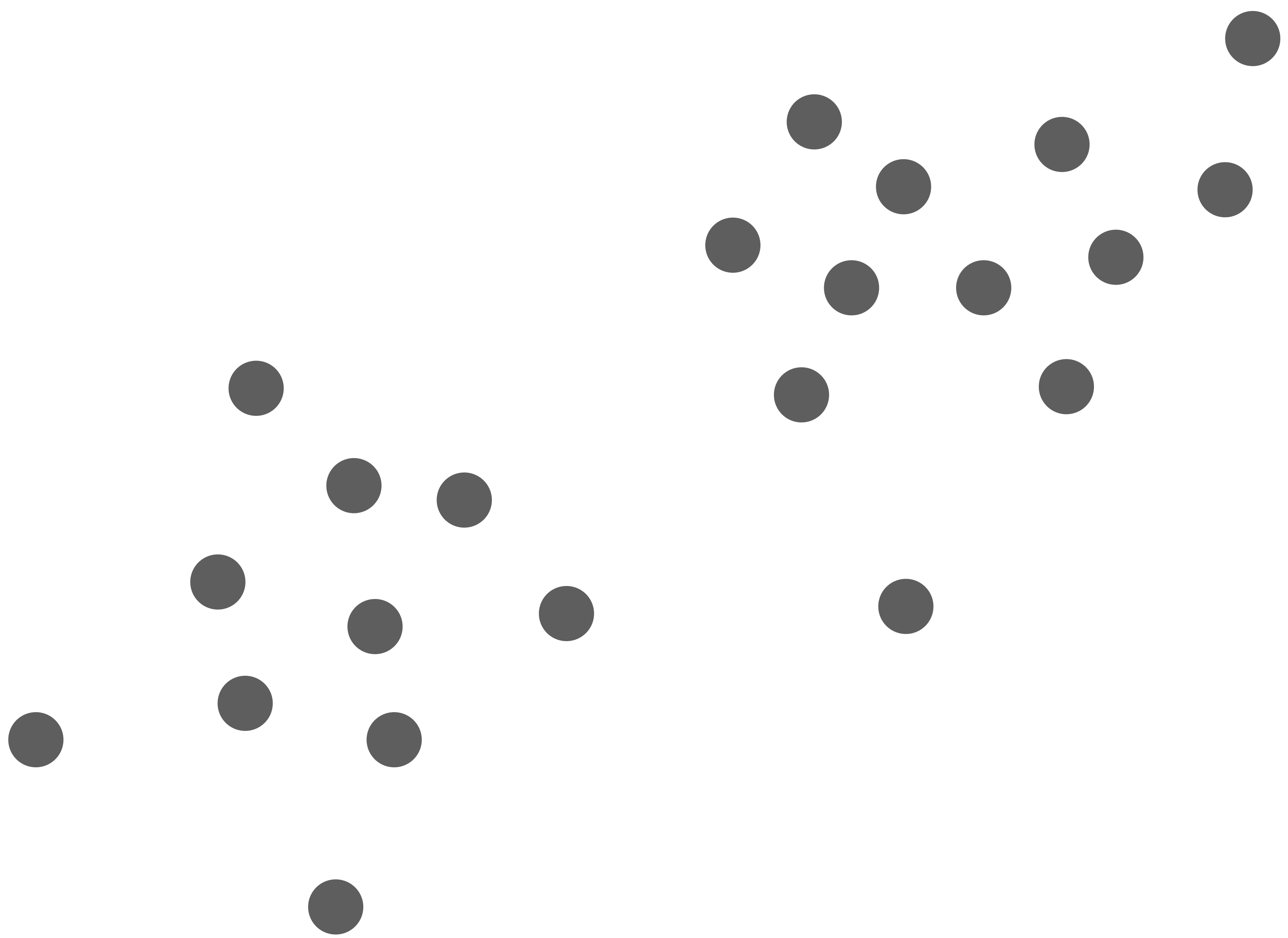
# When does *k*-means work well?

# When does *k*-means work well?

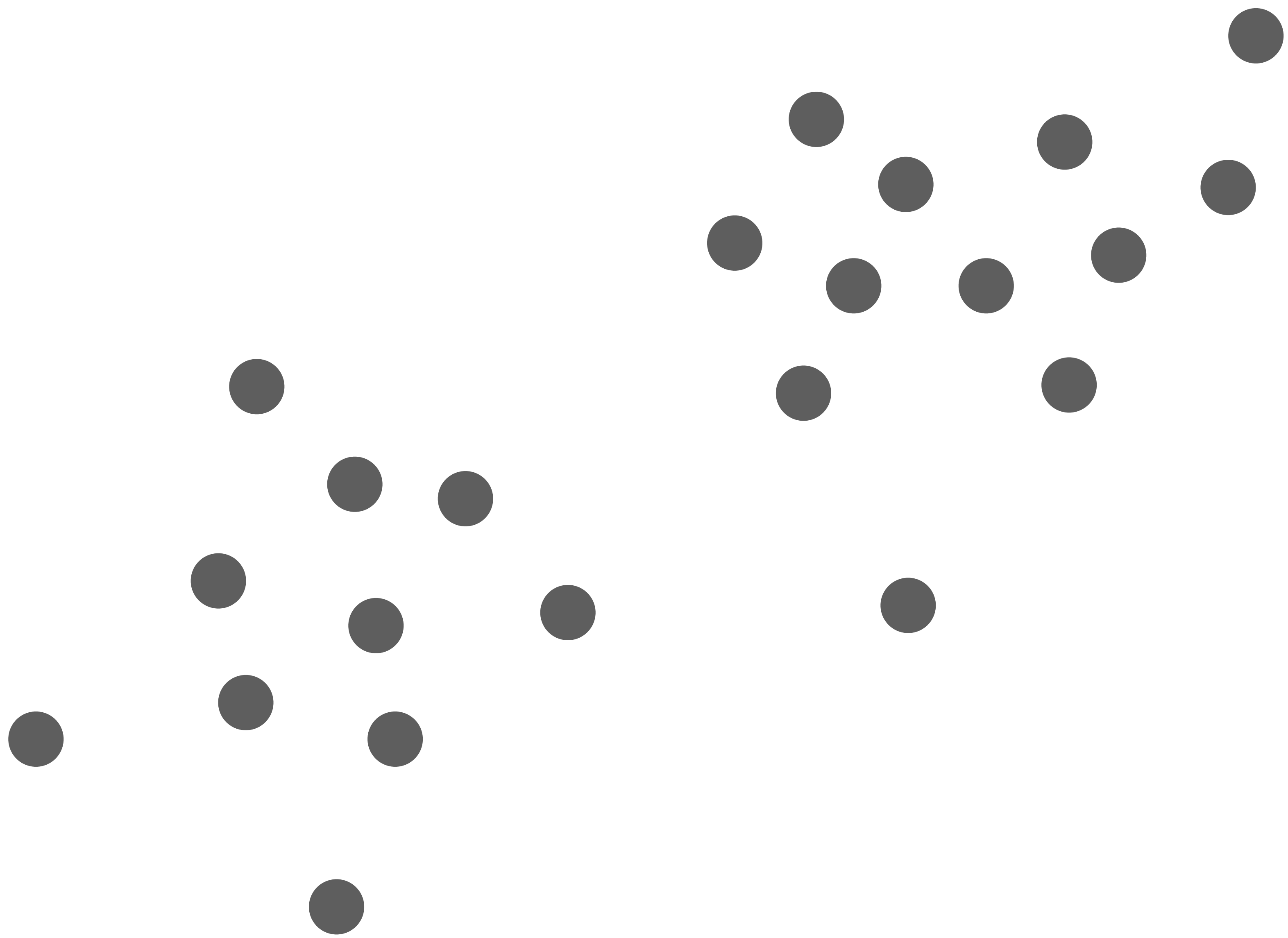
*k*-means is related to a more general model, which will help us understand *k*-means

# Gaussian Mixture Model (GMM)

# Gaussian Mixture Model (GMM)



# Gaussian Mixture Model (GMM)



What random process could have generated these points?

# Generative Process

# Generative Process

Think of flipping a coin

# Generative Process

Think of flipping a coin

each outcome:

# Generative Process

Think of flipping a coin

each outcome: heads or tails

# 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

# 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…*

# Generative Process

Think of flipping a coin

each outcome:        2D point

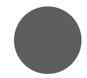
Each flip doesn't depend on any of the previous flips

*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…*

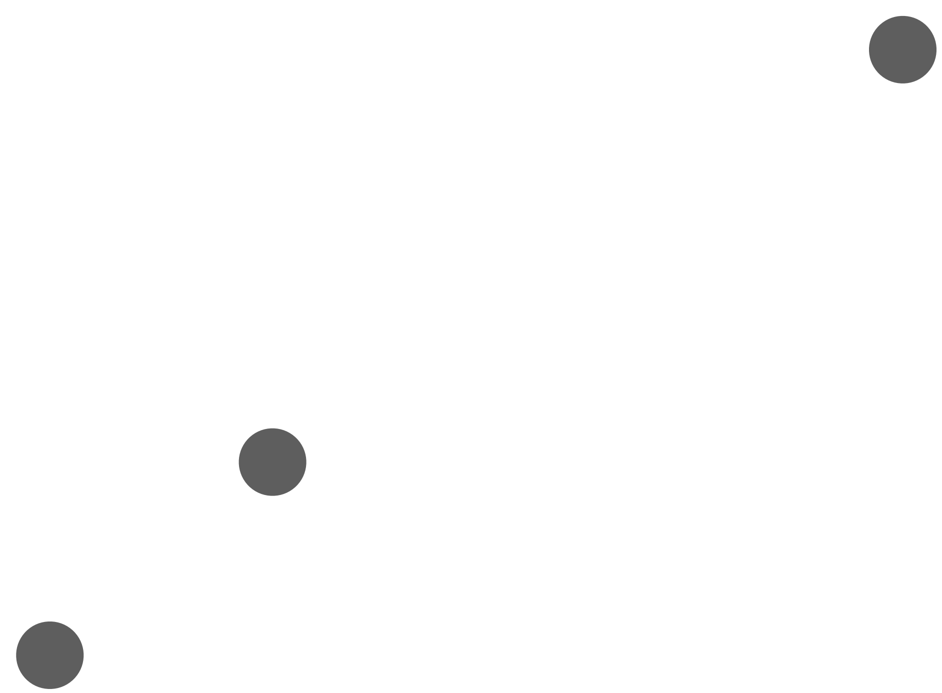
# Gaussian Mixture Model (GMM)

# Gaussian Mixture Model (GMM)



# Gaussian Mixture Model (GMM)

# Gaussian Mixture Model (GMM)

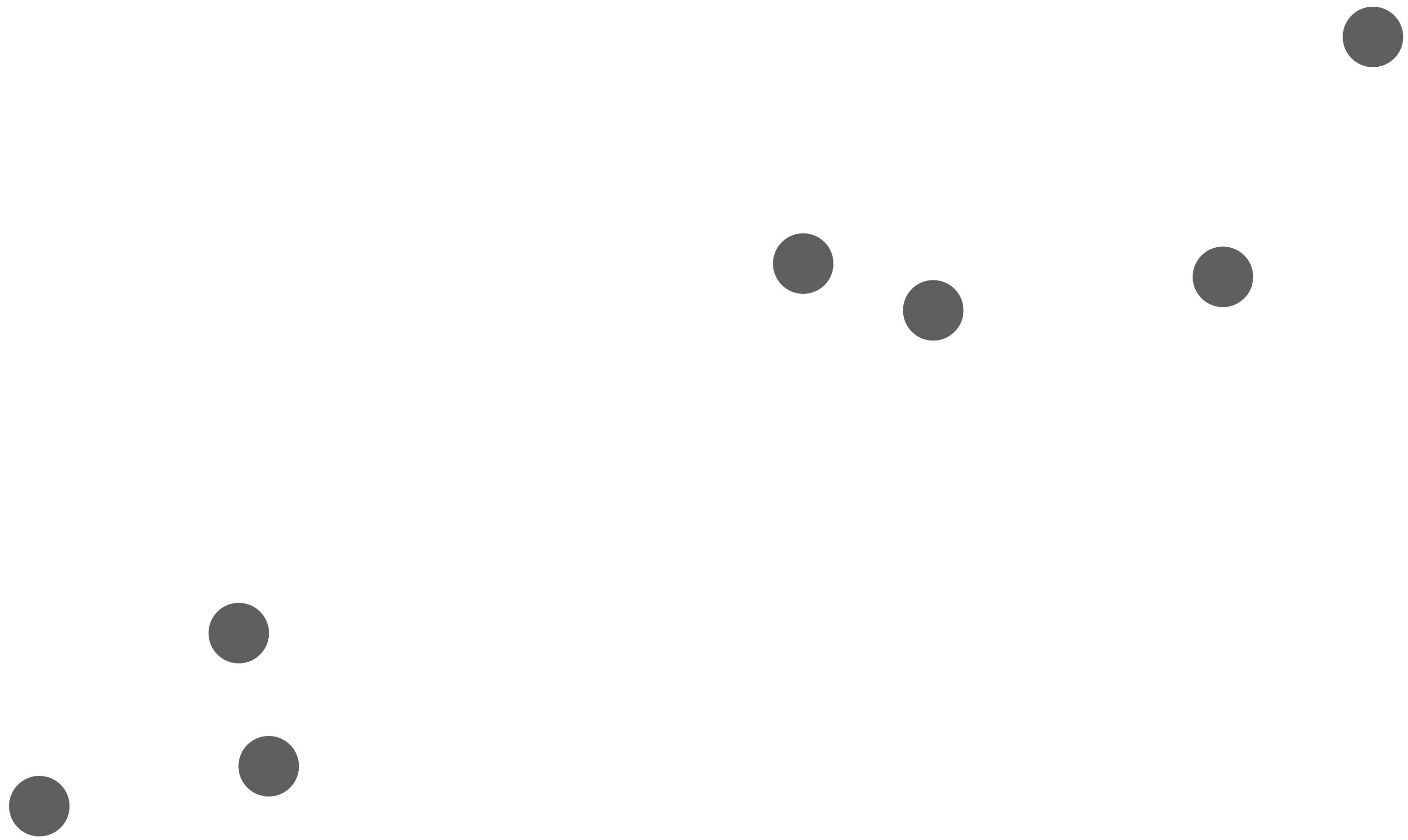


# Gaussian Mixture Model (GMM)

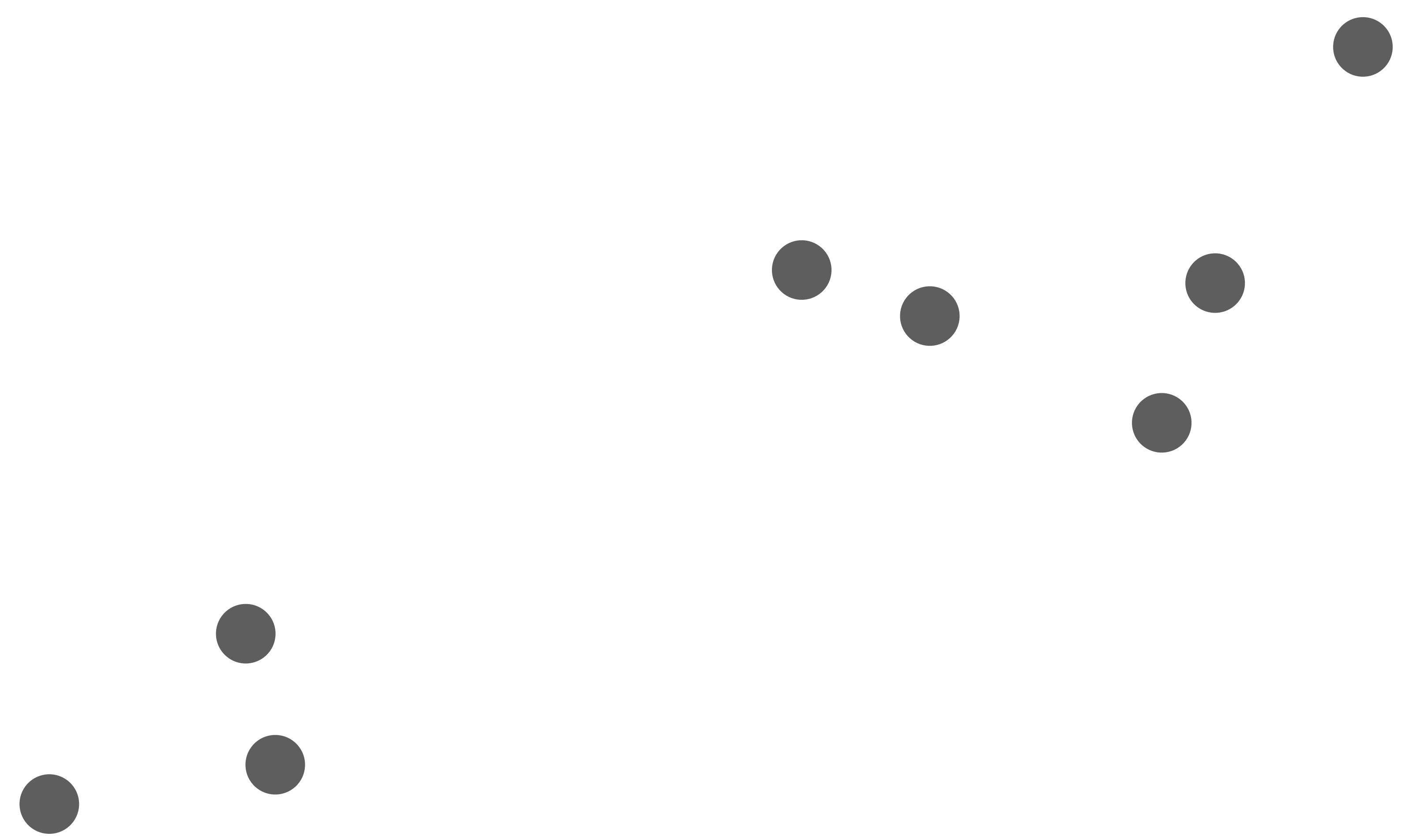
# Gaussian Mixture Model (GMM)

# Gaussian Mixture Model (GMM)

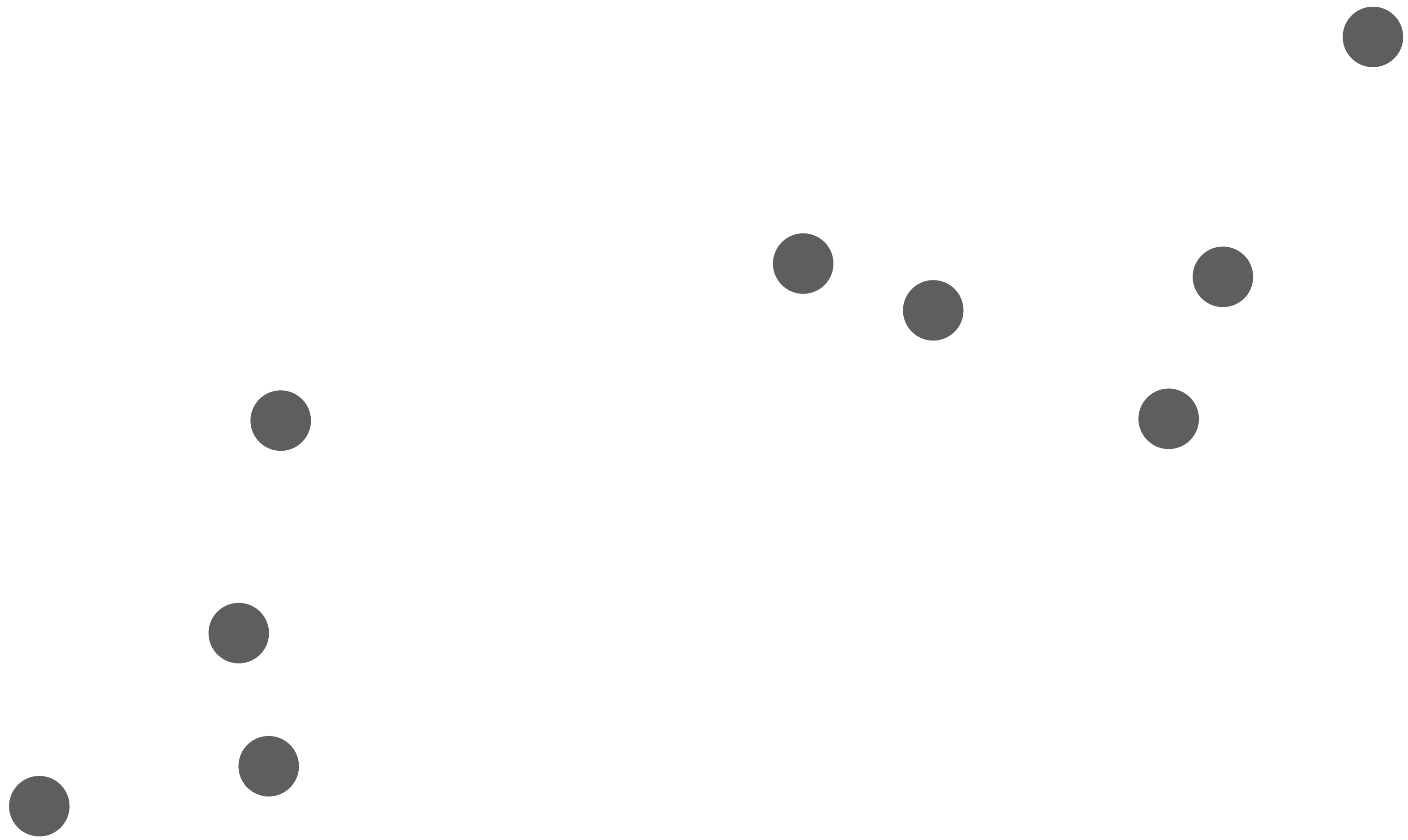
# Gaussian Mixture Model (GMM)



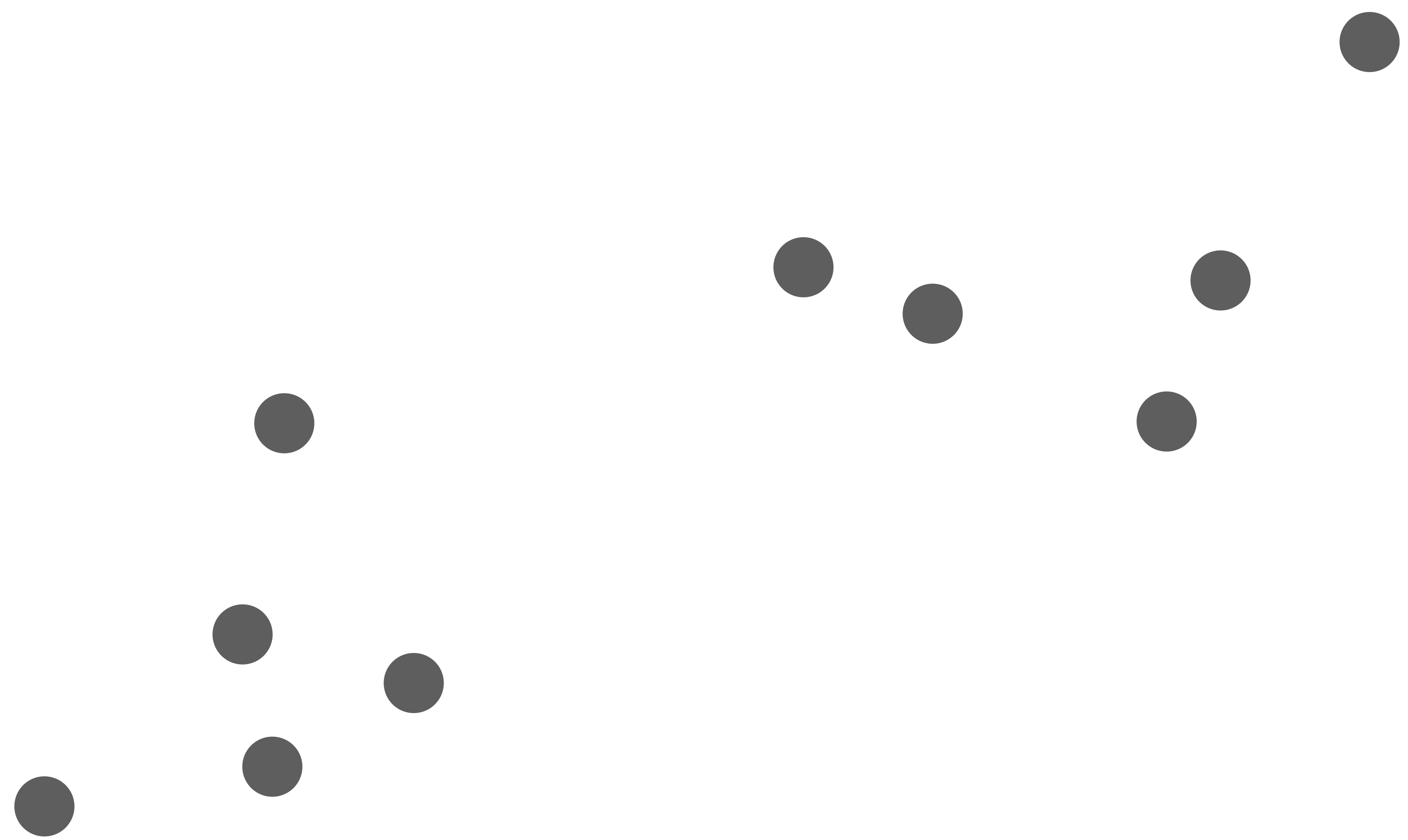
# Gaussian Mixture Model (GMM)



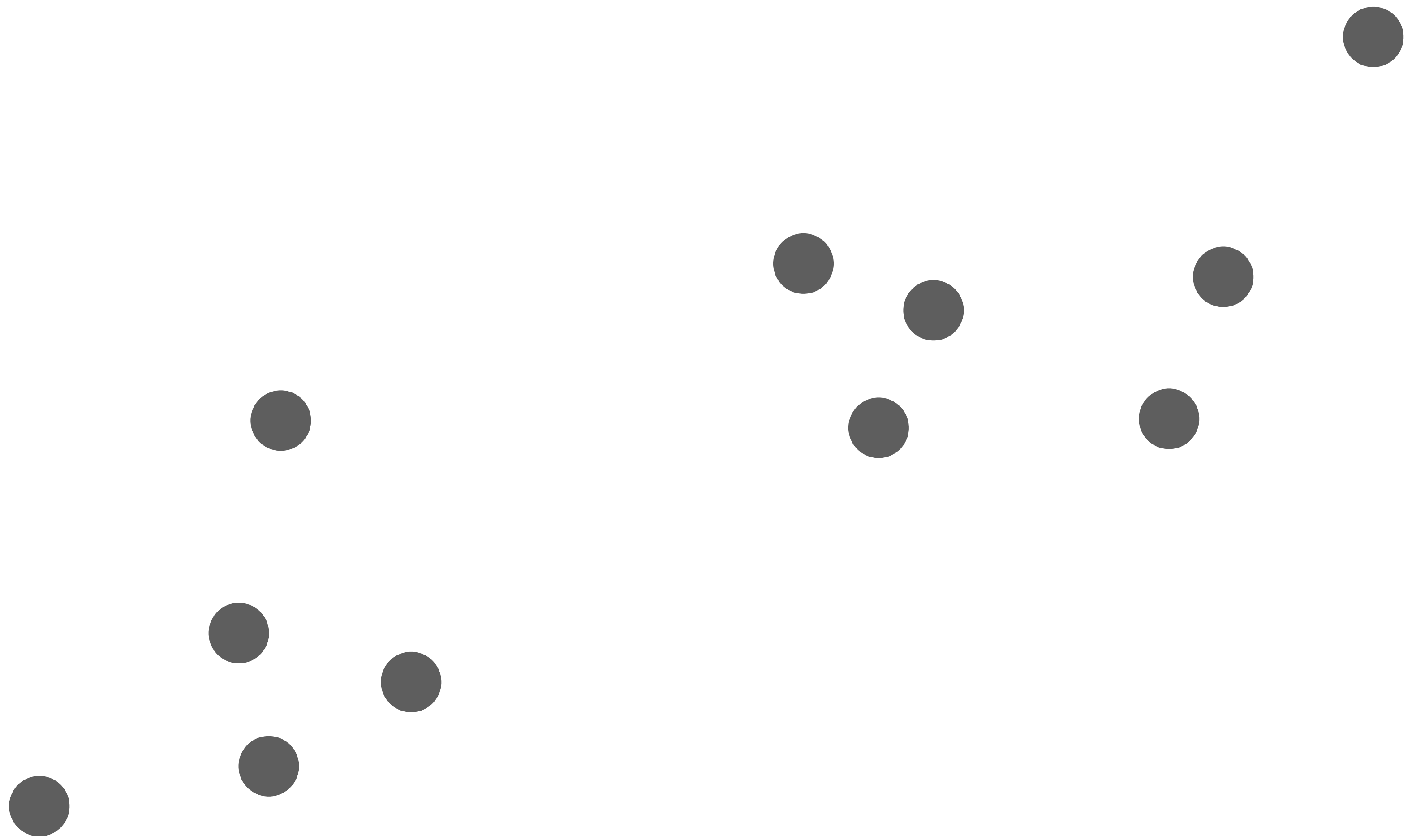
# Gaussian Mixture Model (GMM)



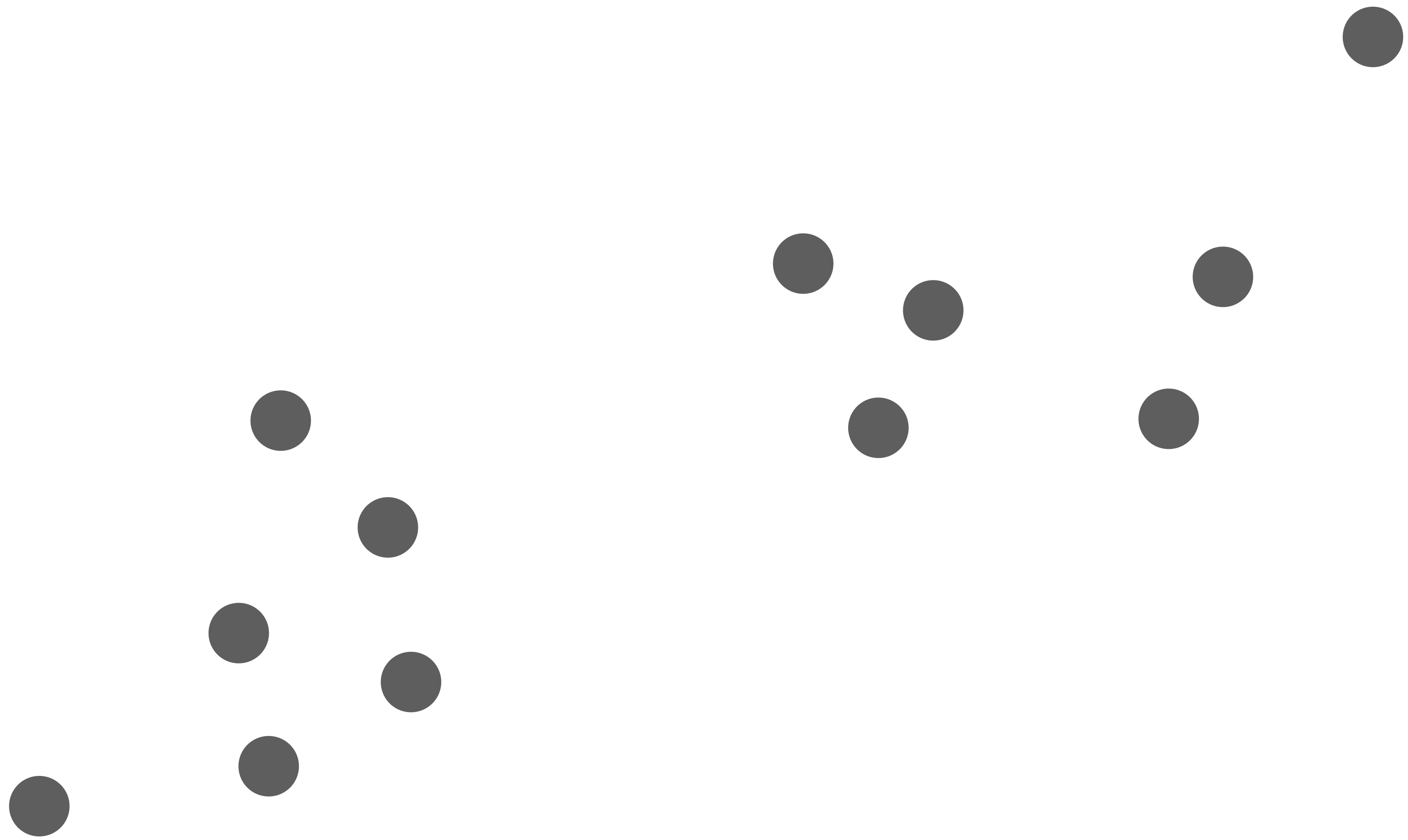
# Gaussian Mixture Model (GMM)



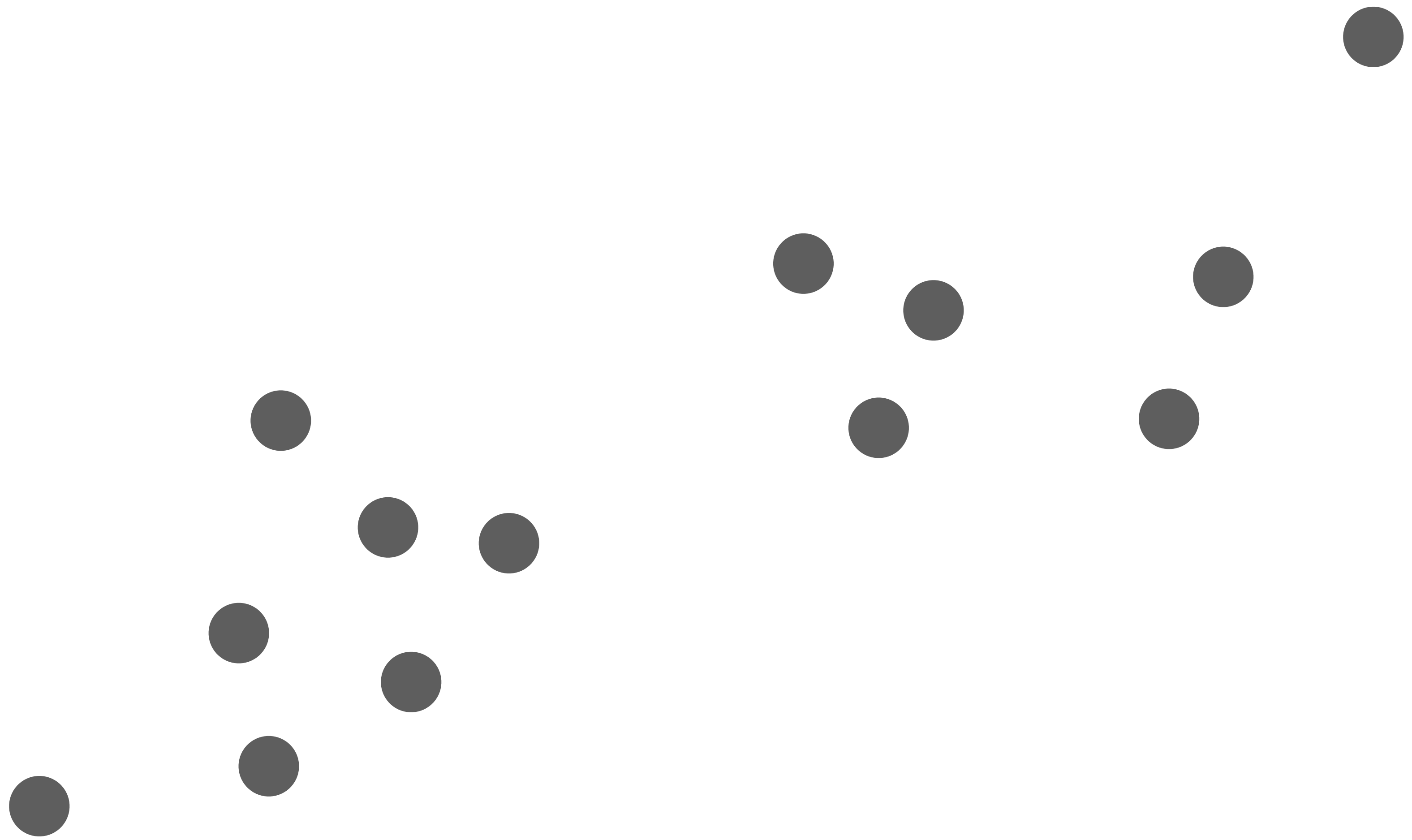
# Gaussian Mixture Model (GMM)



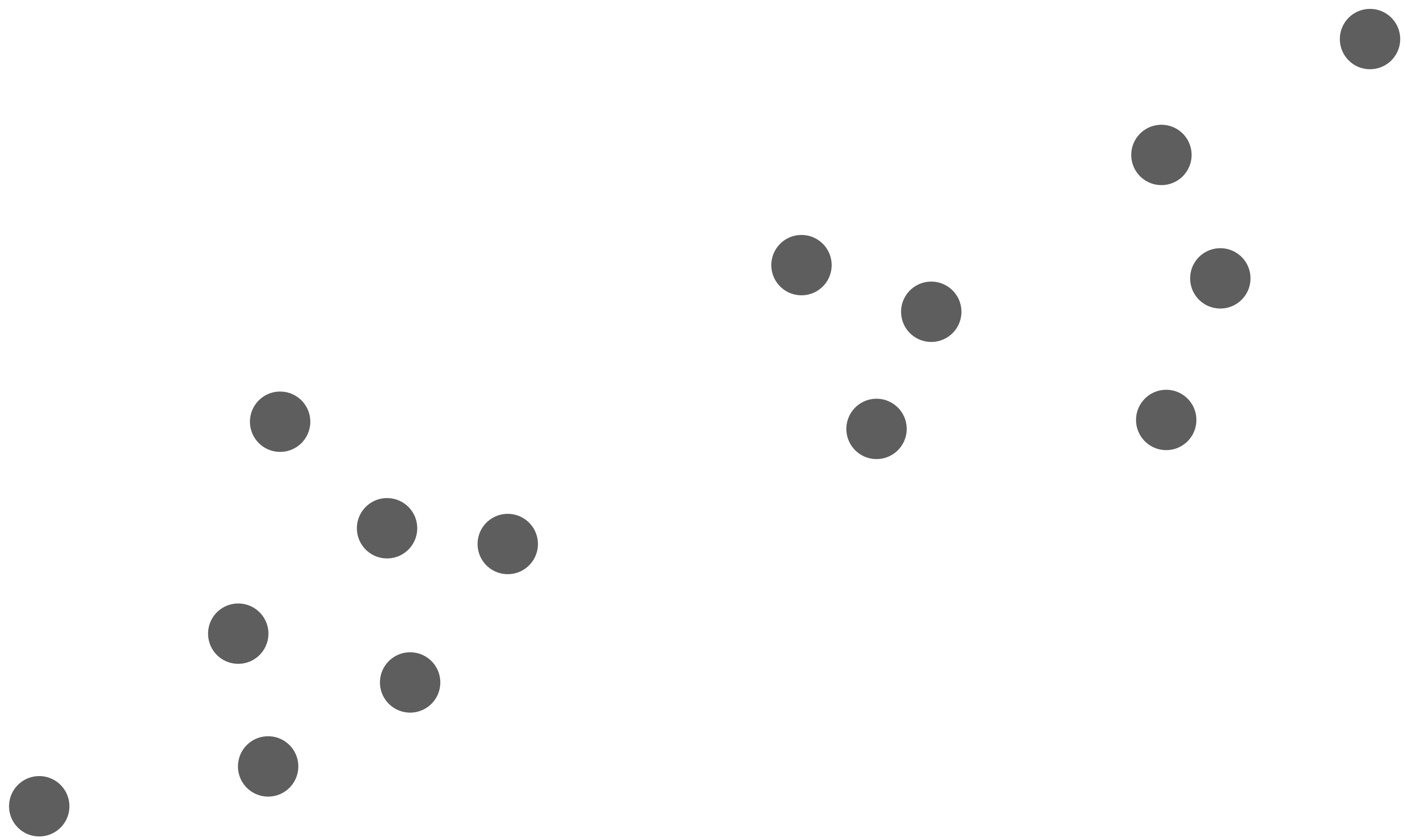
# Gaussian Mixture Model (GMM)



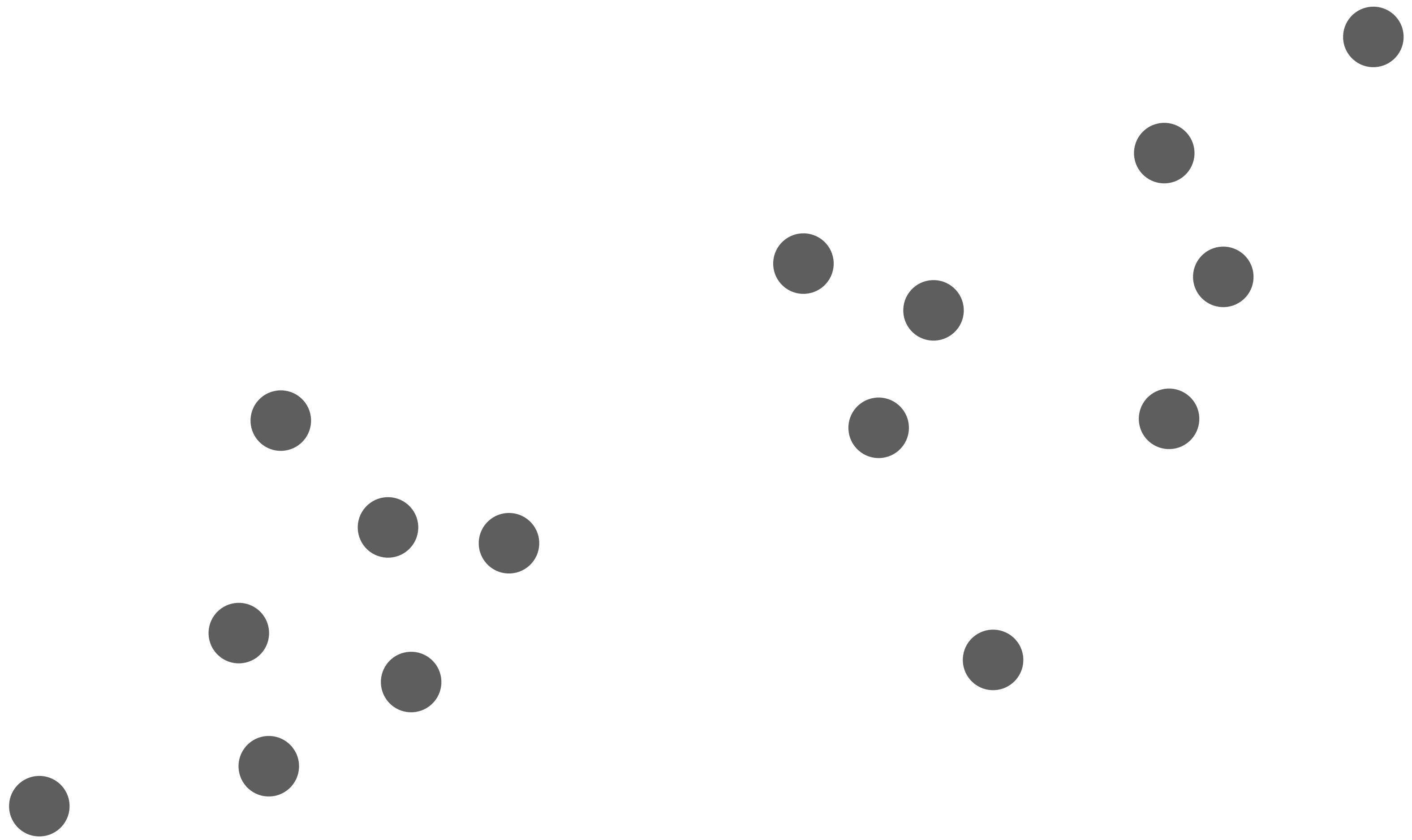
# Gaussian Mixture Model (GMM)



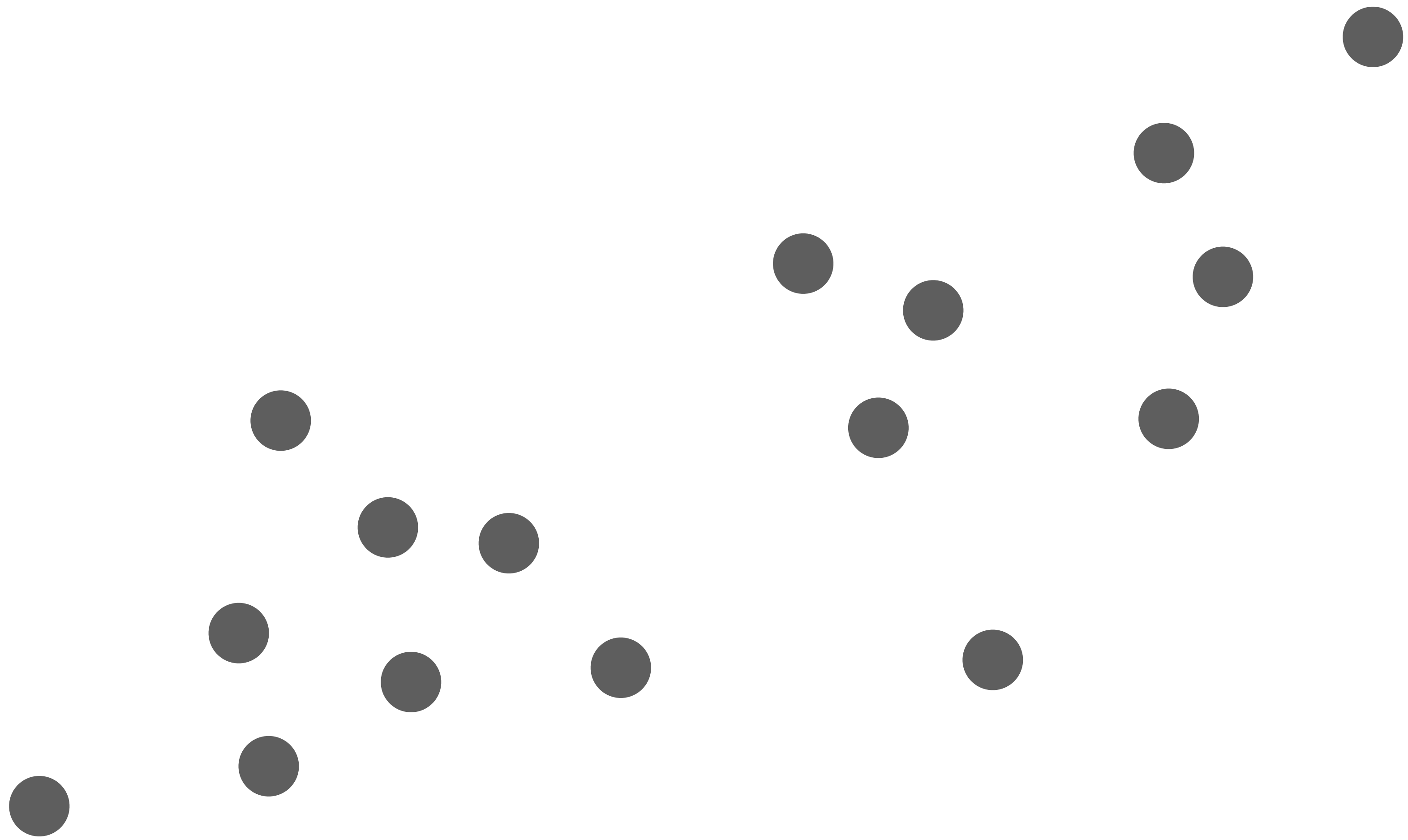
# Gaussian Mixture Model (GMM)



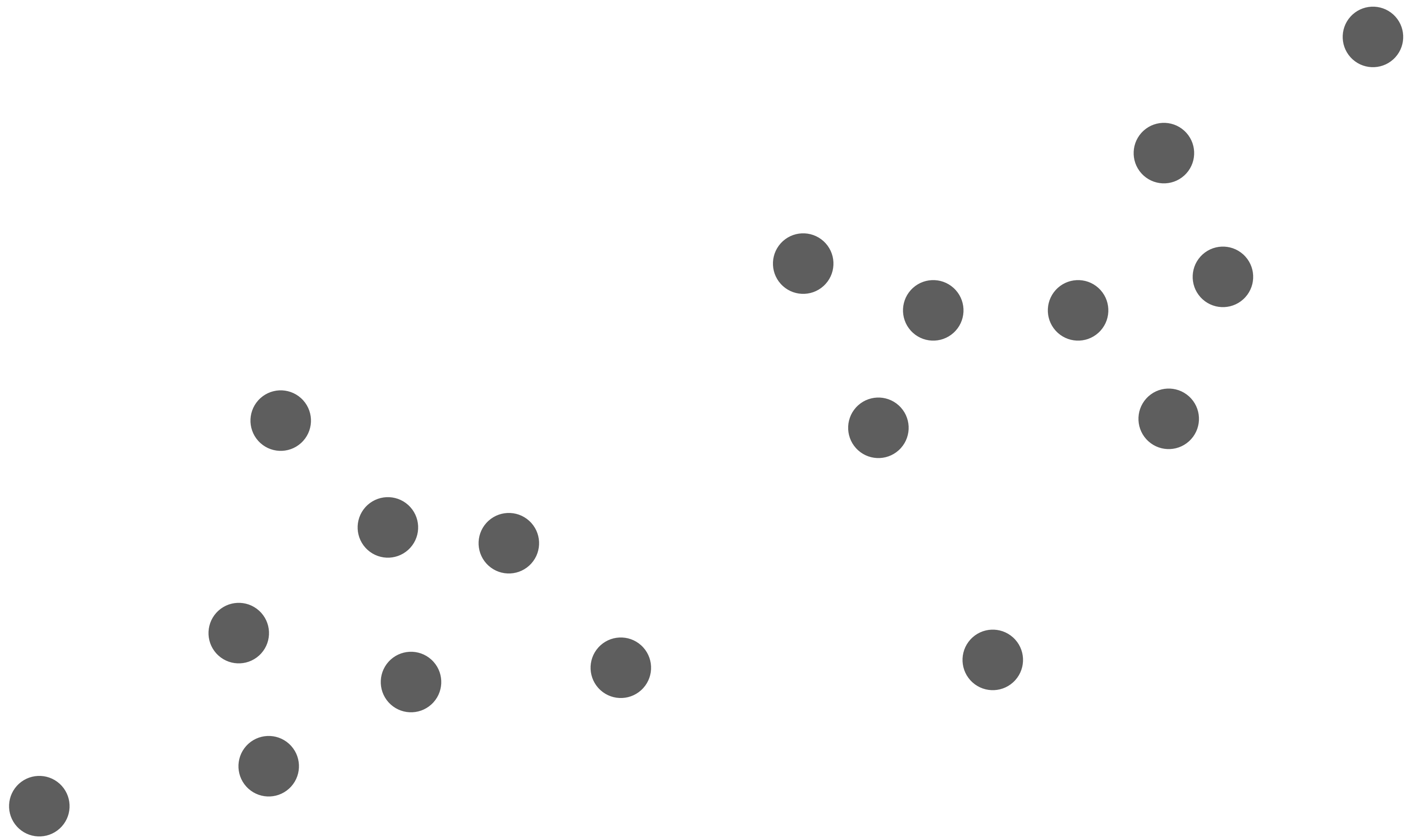
# Gaussian Mixture Model (GMM)



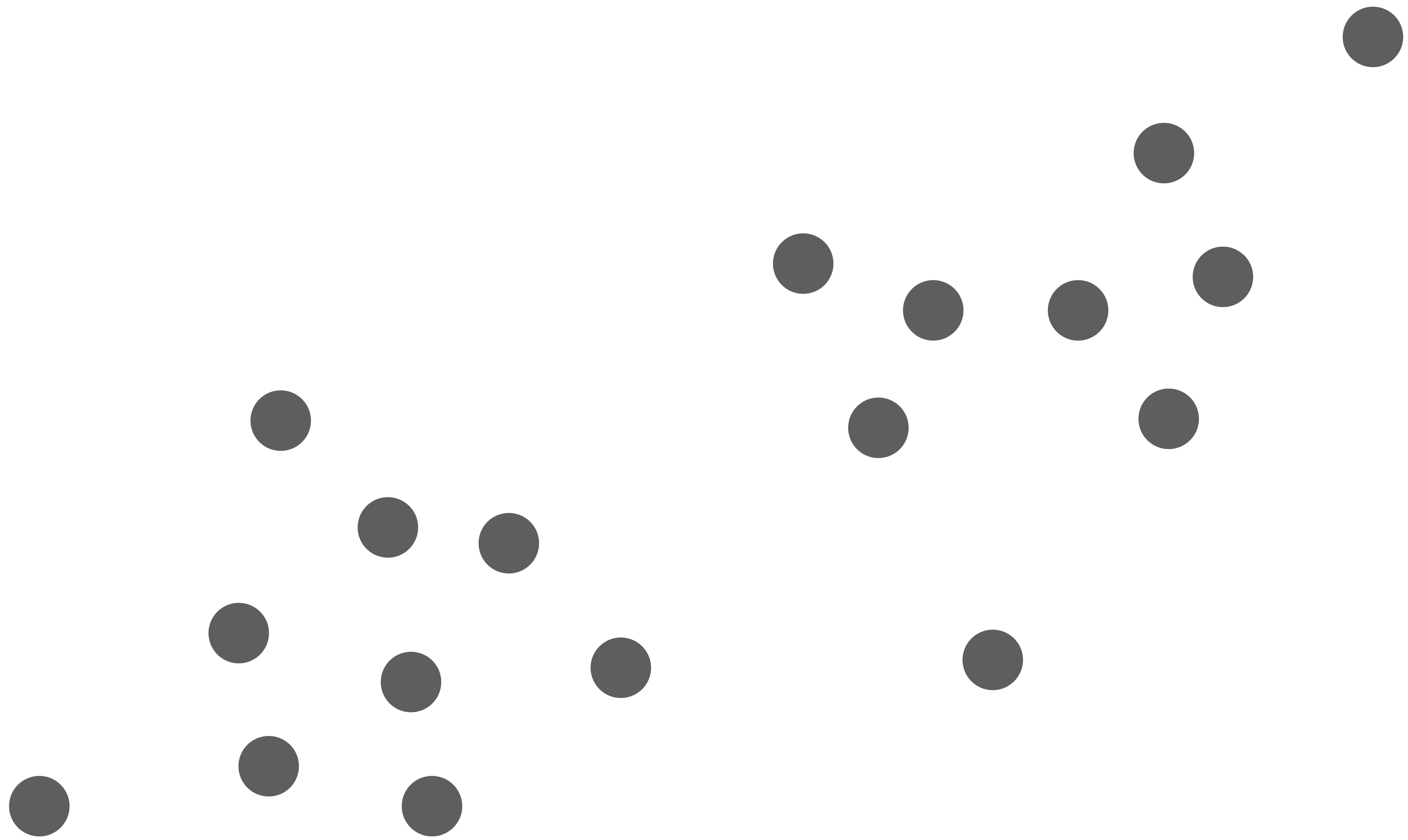
# Gaussian Mixture Model (GMM)



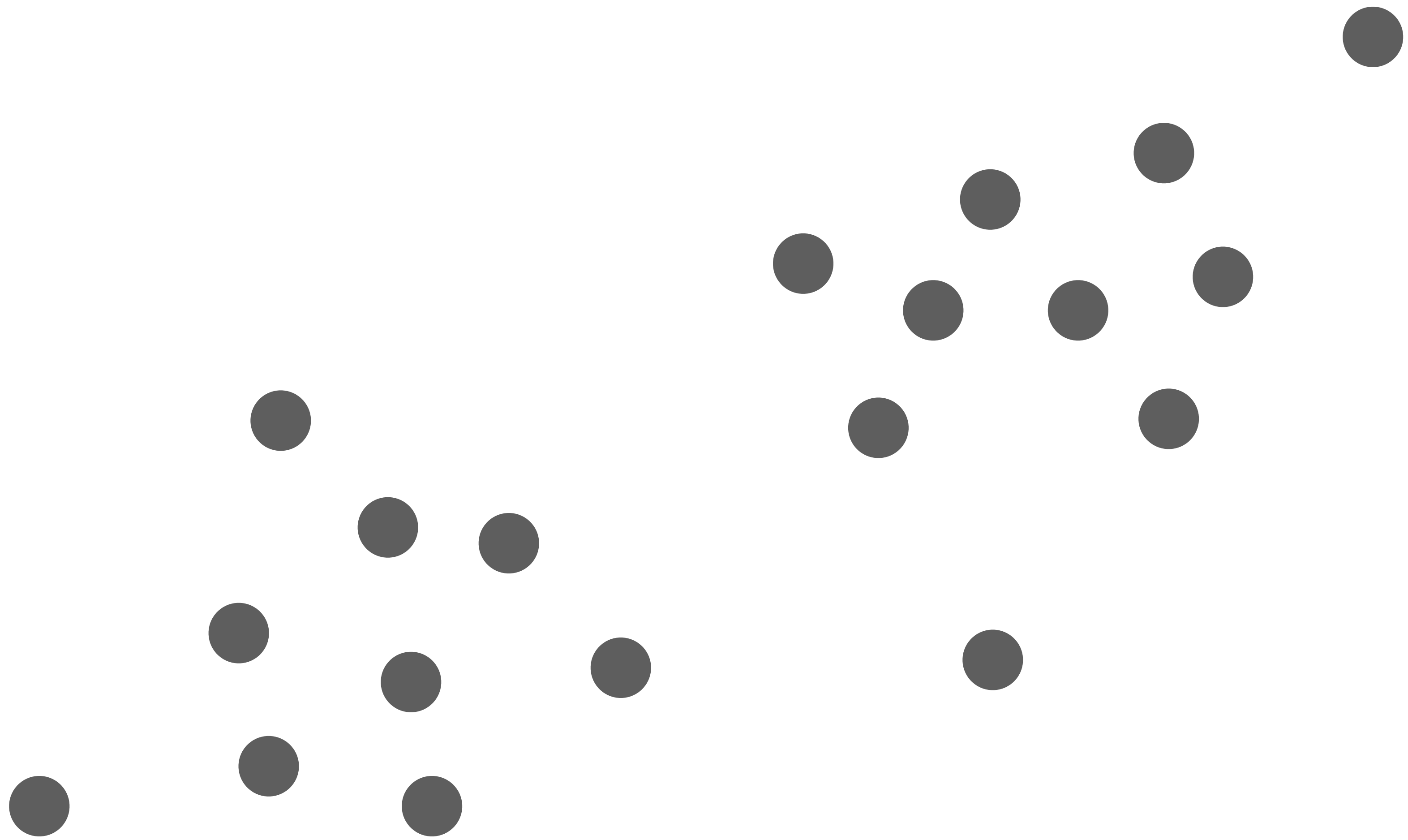
# Gaussian Mixture Model (GMM)



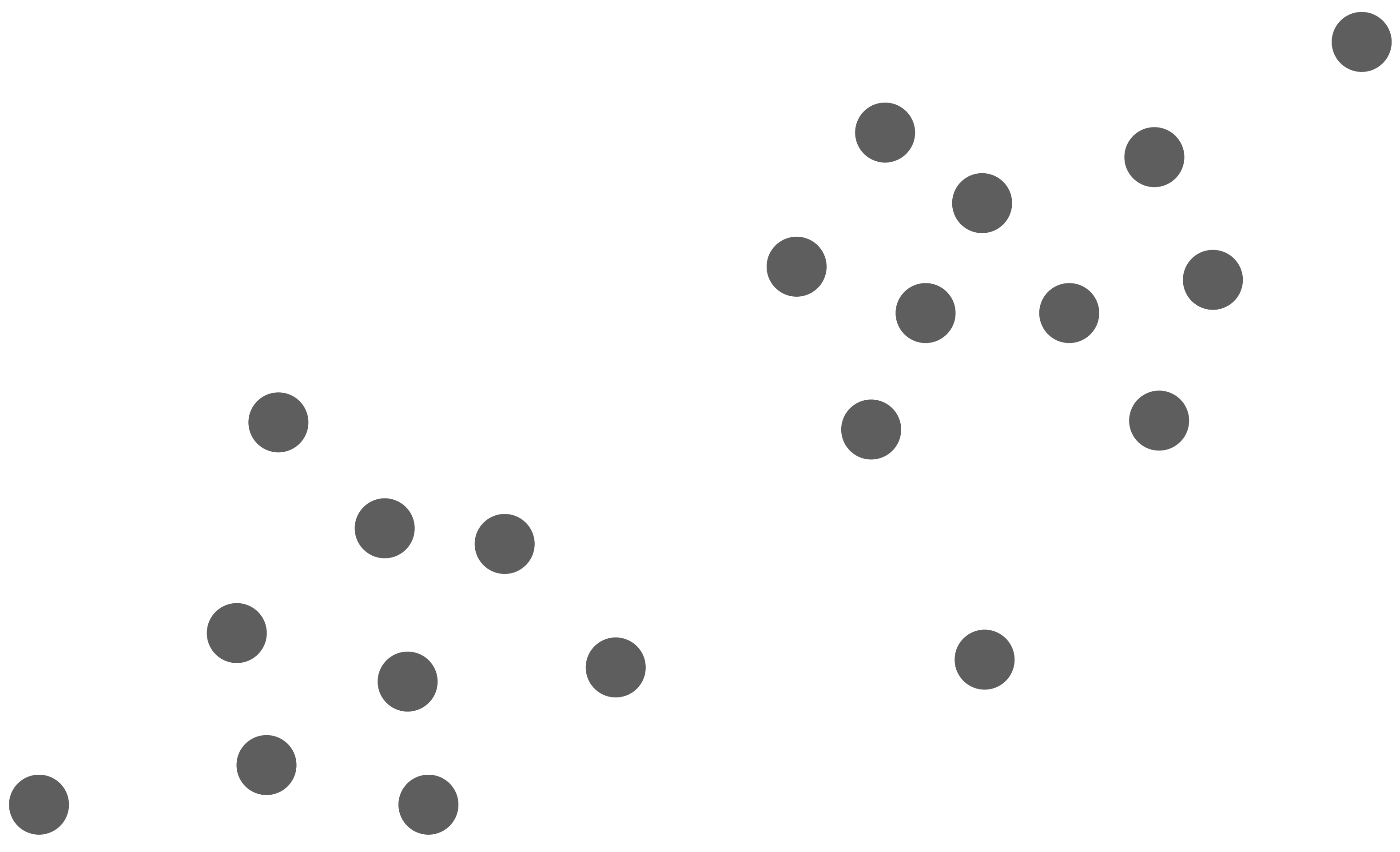
# Gaussian Mixture Model (GMM)



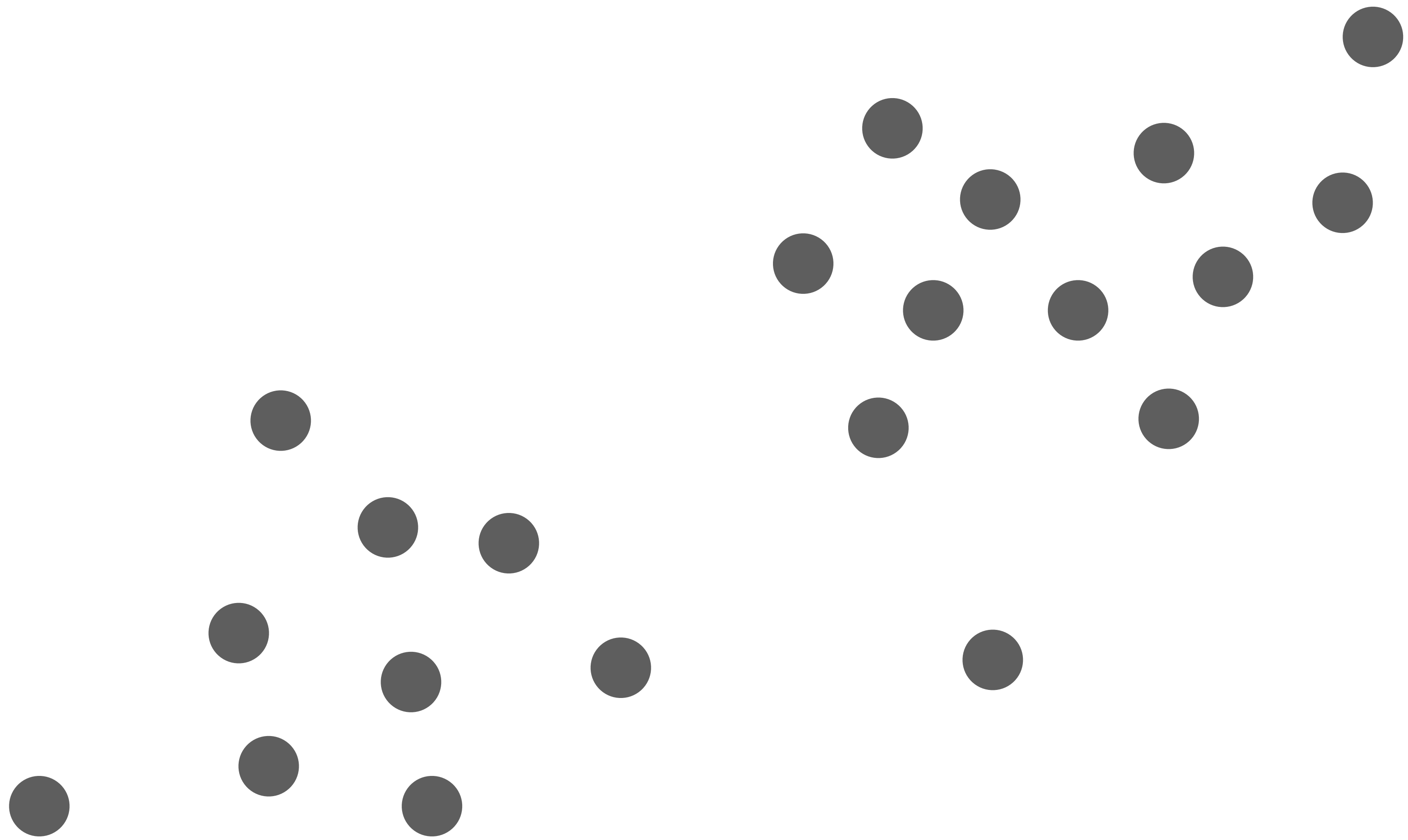
# Gaussian Mixture Model (GMM)



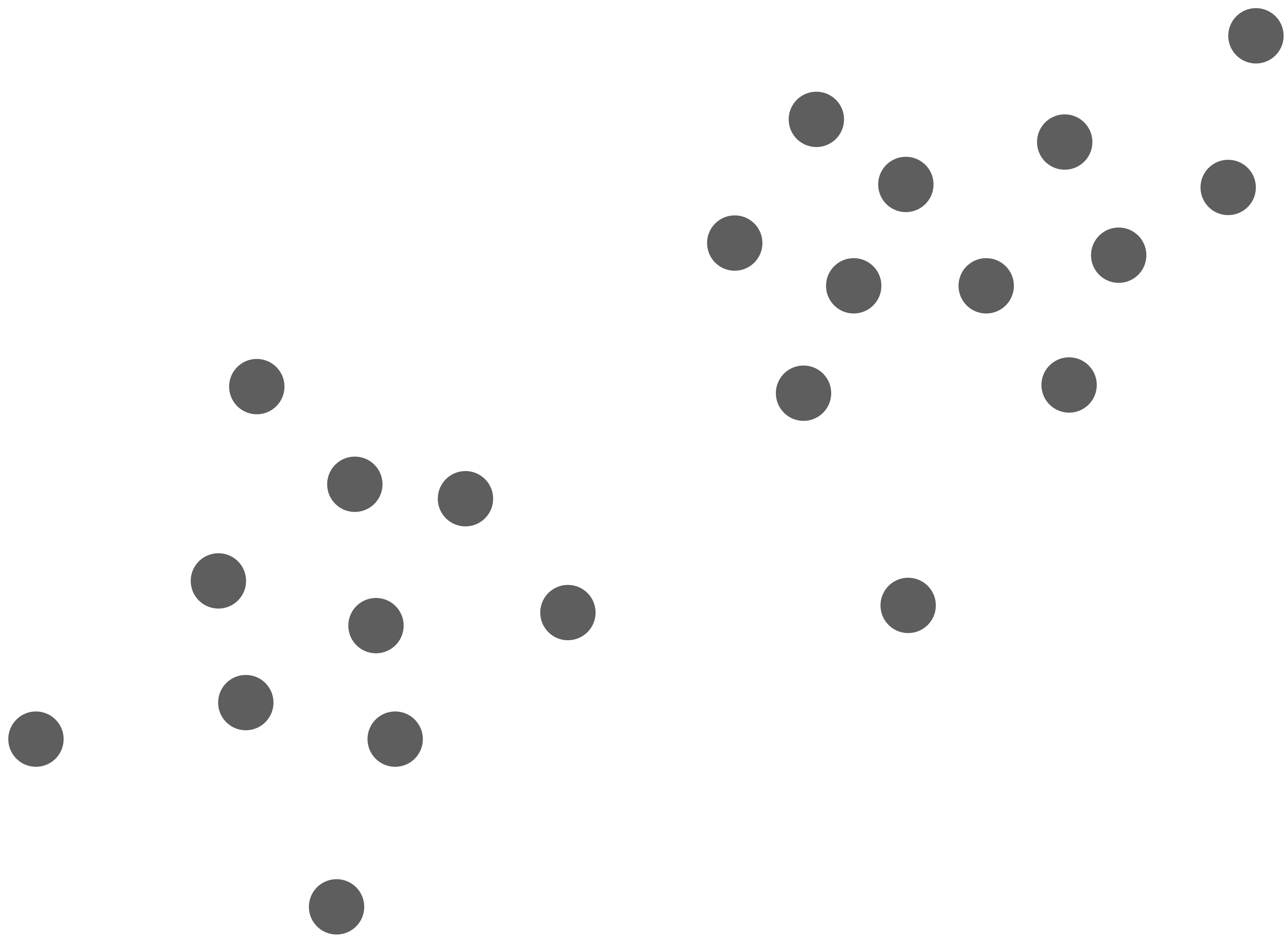
# Gaussian Mixture Model (GMM)



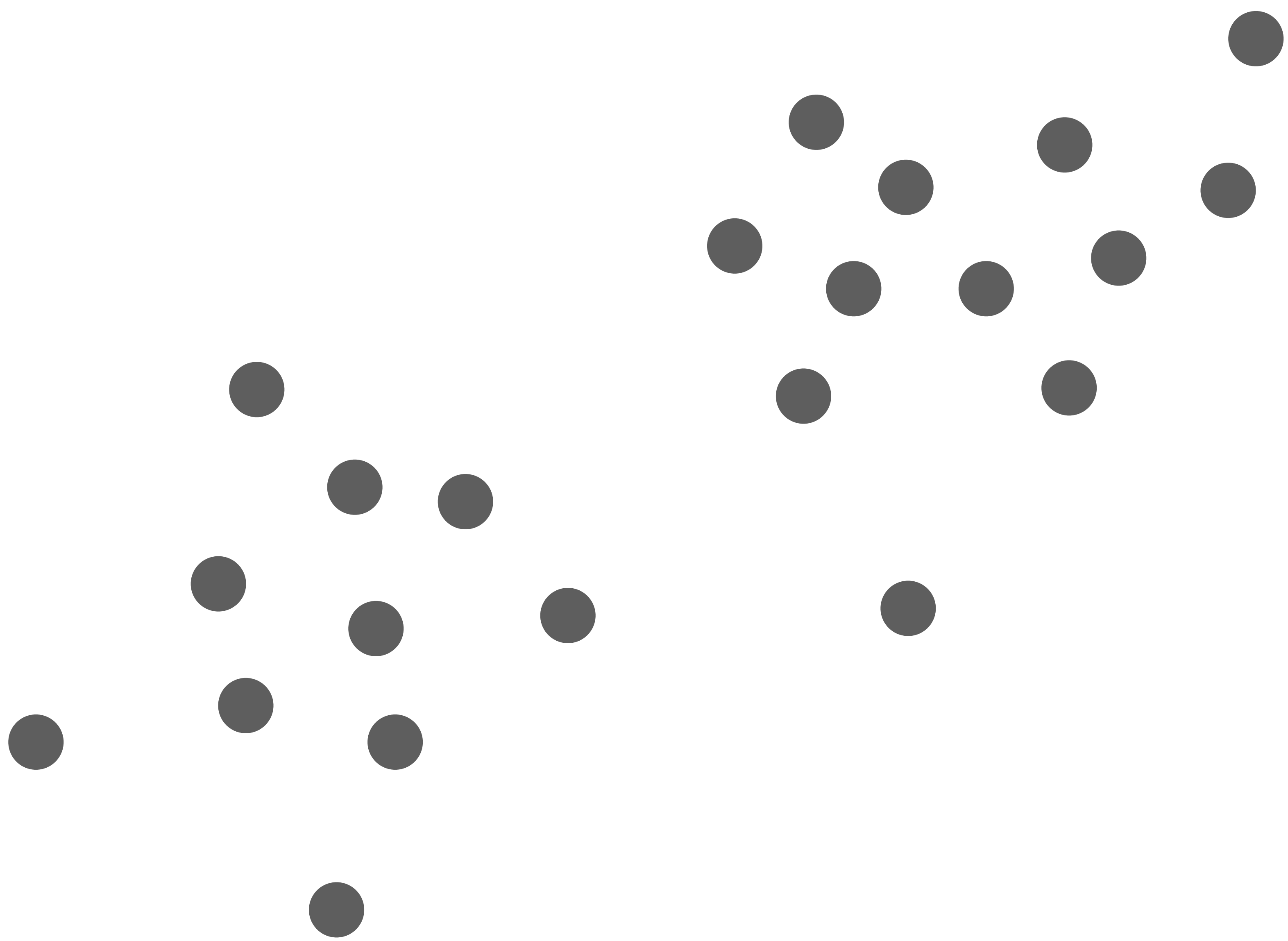
# Gaussian Mixture Model (GMM)



# Gaussian Mixture Model (GMM)



# Gaussian Mixture Model (GMM)



We now discuss a way to generate points in this manner

# Gaussian Mixture Model (GMM)

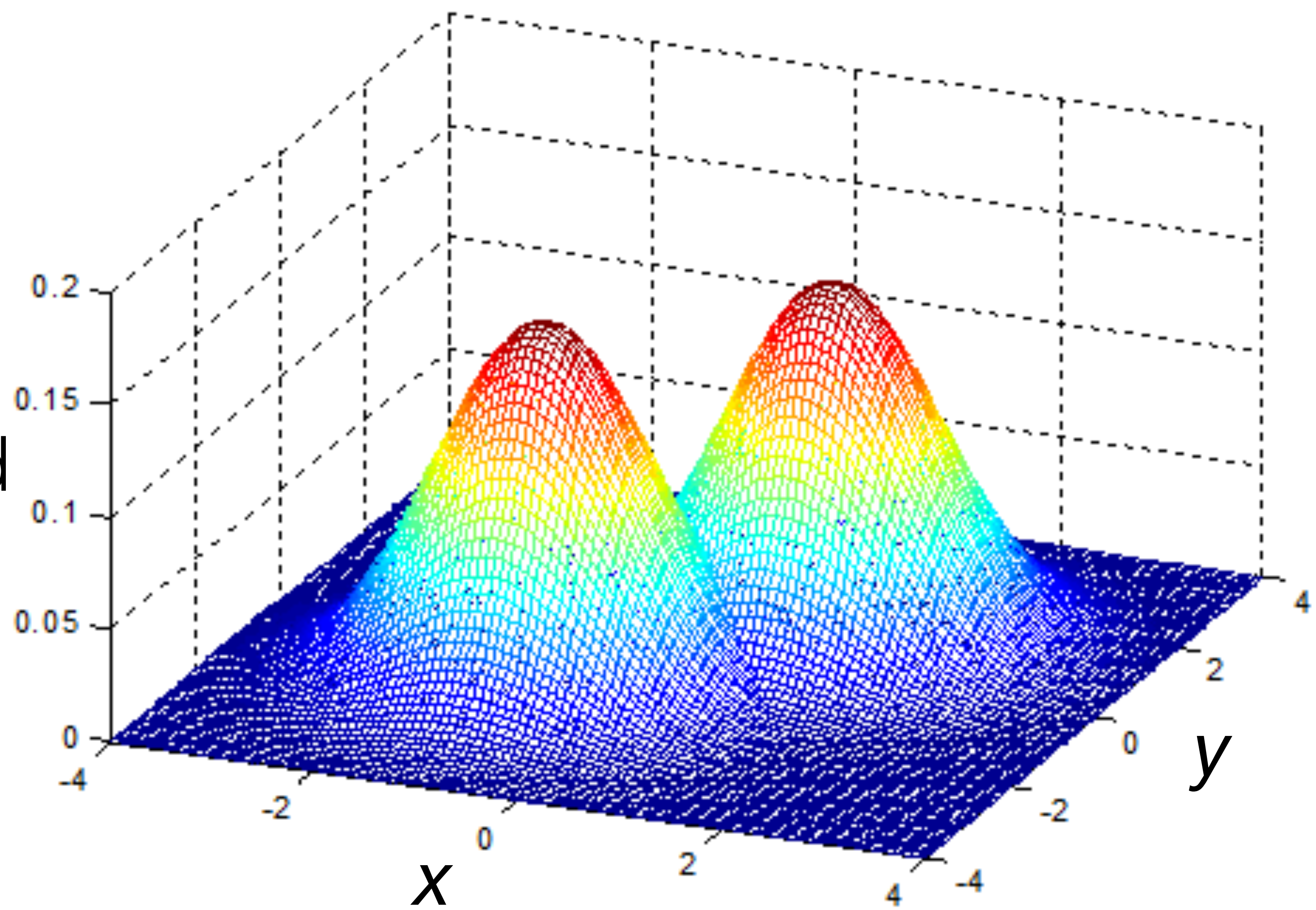
Assume: points sampled independently from a probability distribution

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

# Gaussian Mixture Model (GMM)

Assume: points sampled independently from a probability distribution

how probable point generated at ($x$, $y$) is




Red = more likely

Blue = less likely

Example of a 2D probability distribution

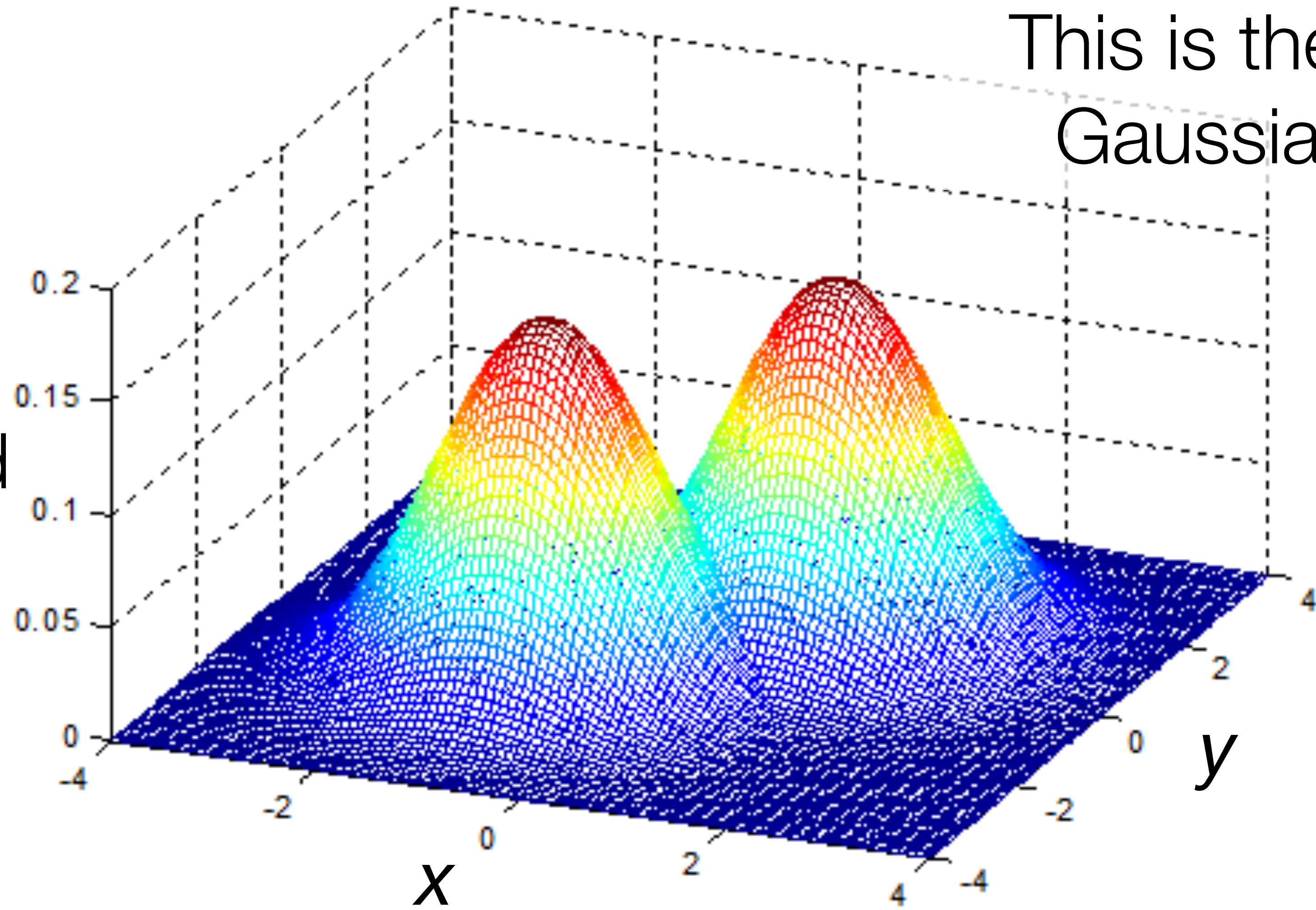
*Image source: https://www.intechopen.com/source/html/17742/media/image25.png*

# Gaussian Mixture Model (GMM)

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



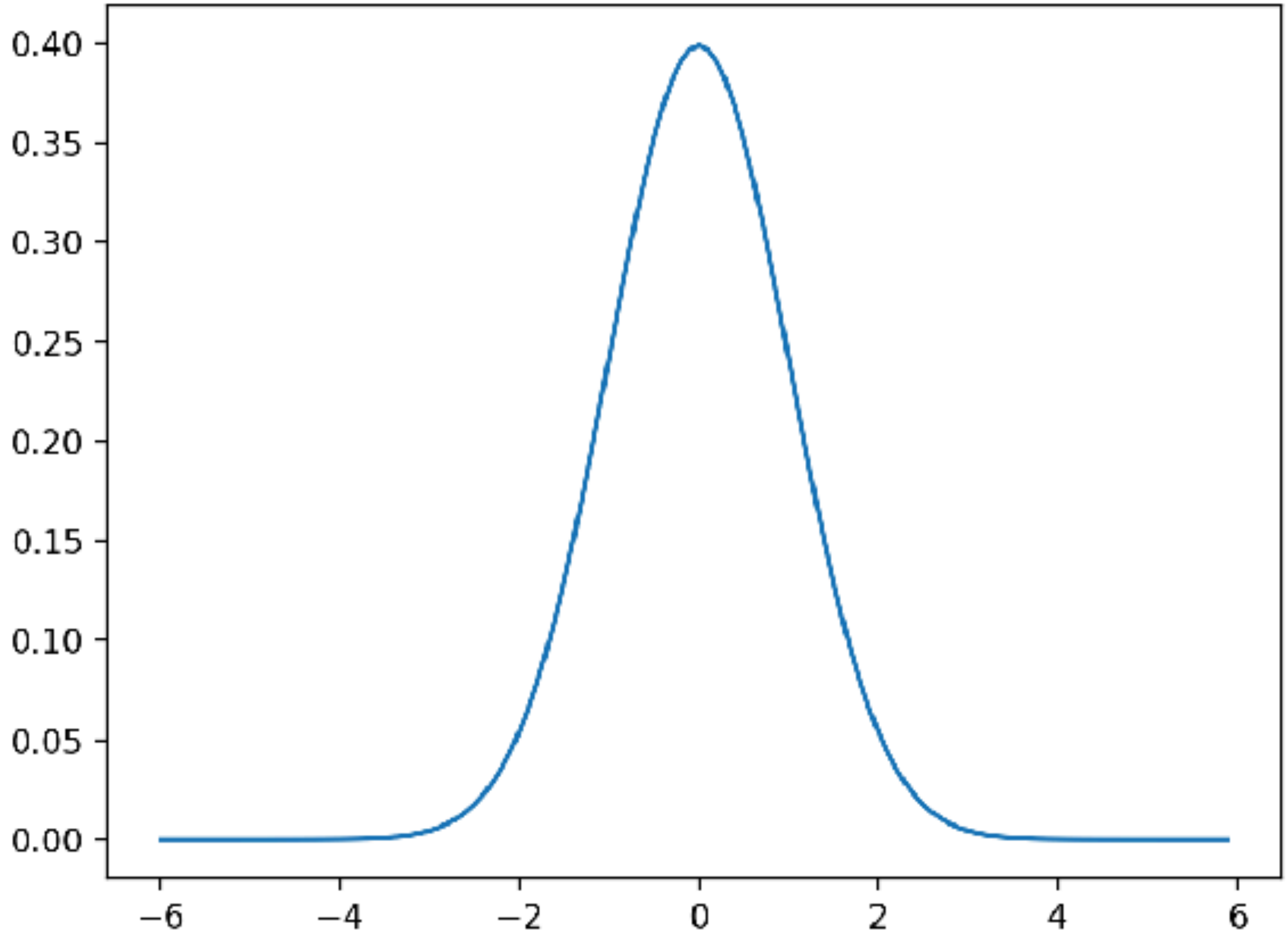

Red = more likely

Blue = less likely

Example of a 2D probability distribution

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

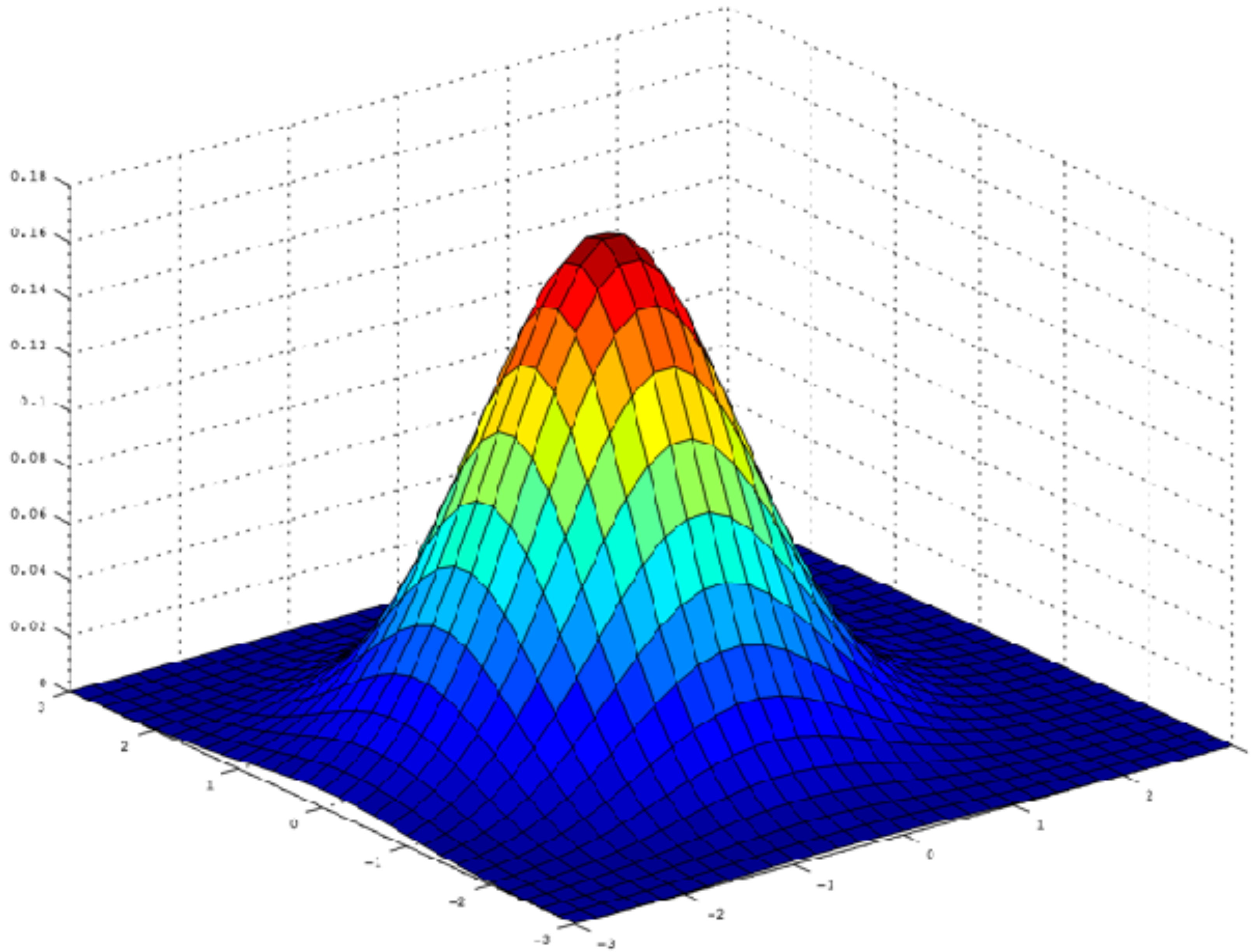
# Quick Reminder: 1D Gaussian




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

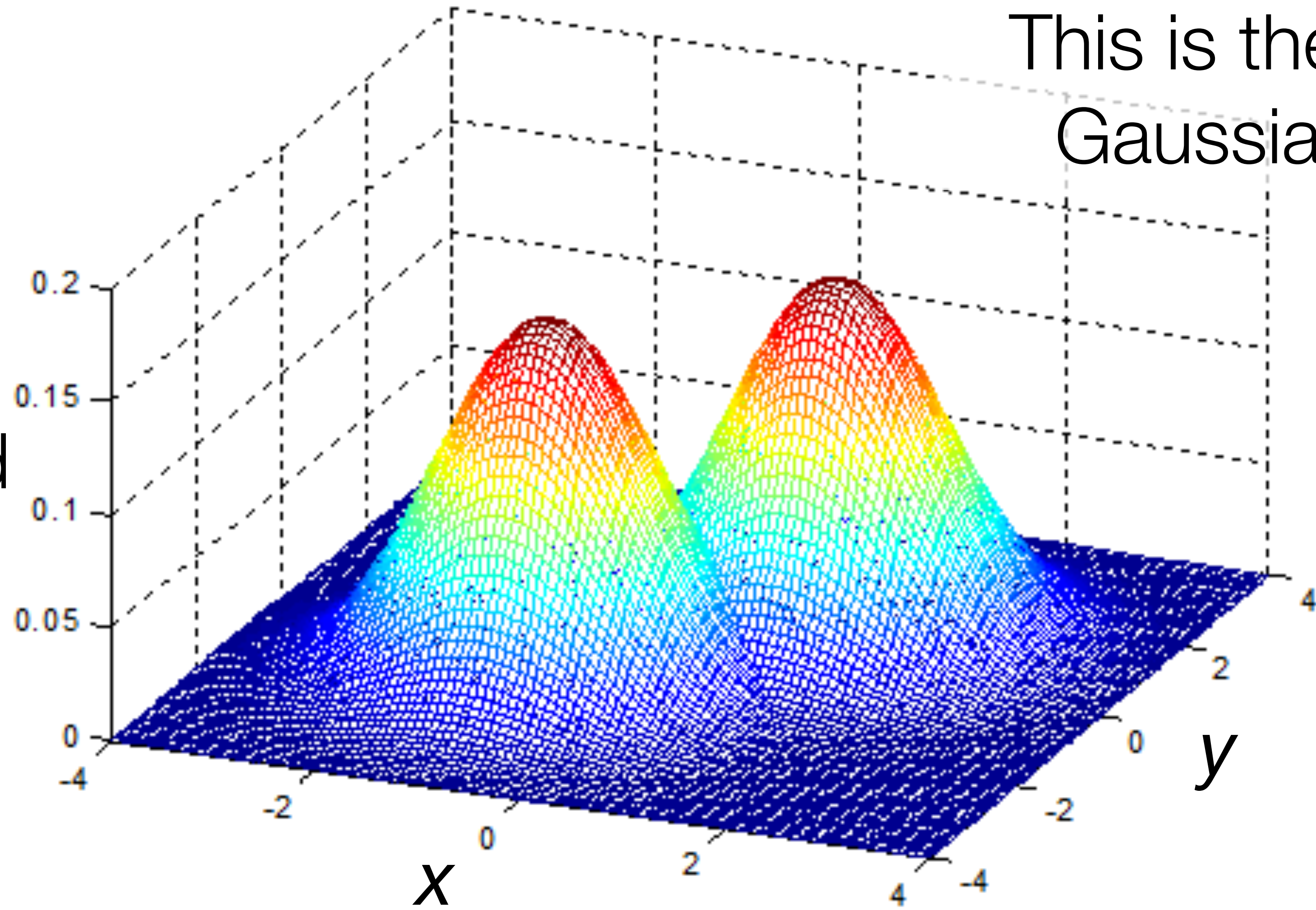
*Image source: https://i.stack.imgur.com/OlWce.png*

# 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

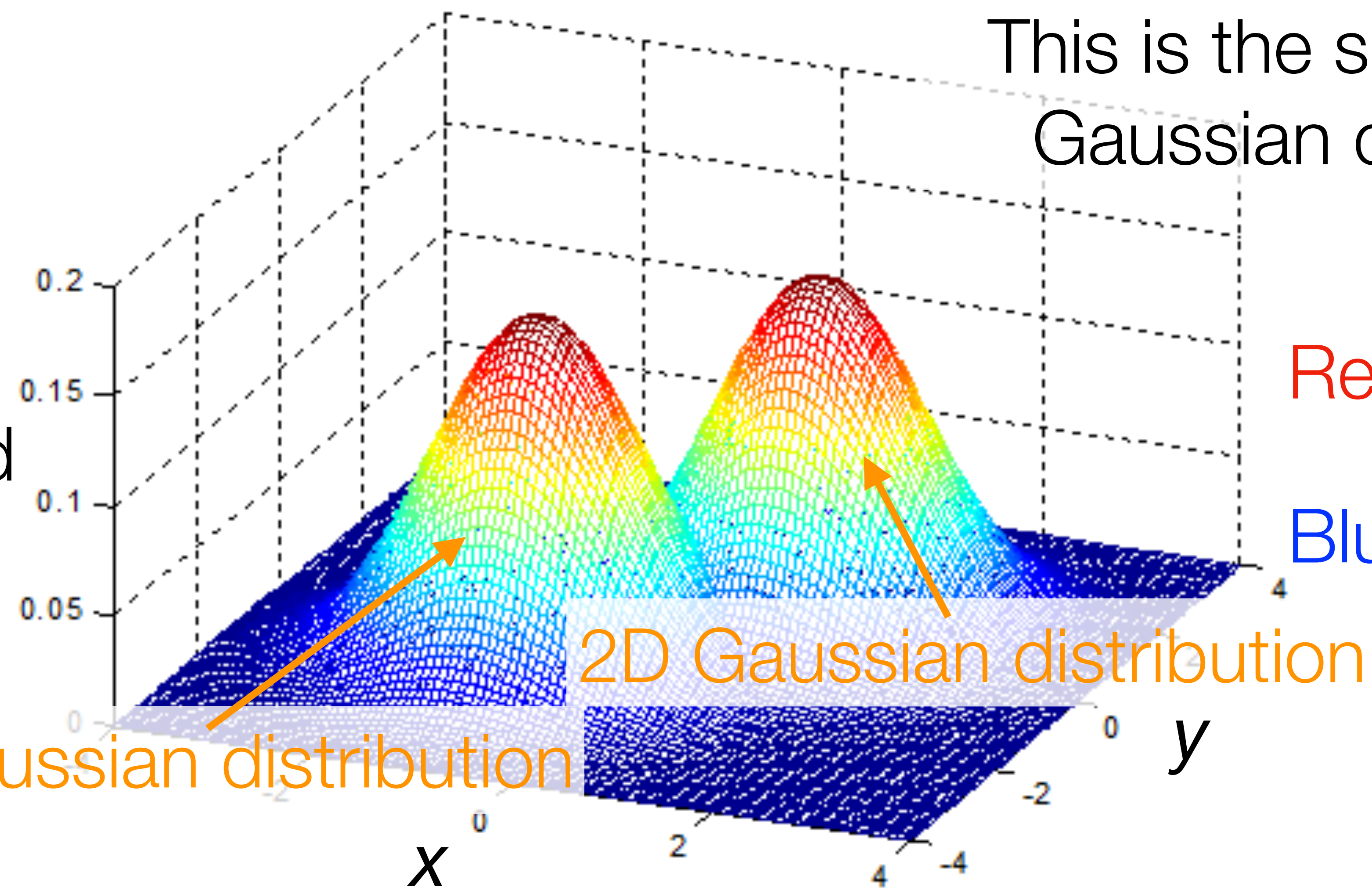
*Image source: https://www.intechopen.com/source/html/17742/media/image25.png*

# Gaussian Mixture Model (GMM)

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

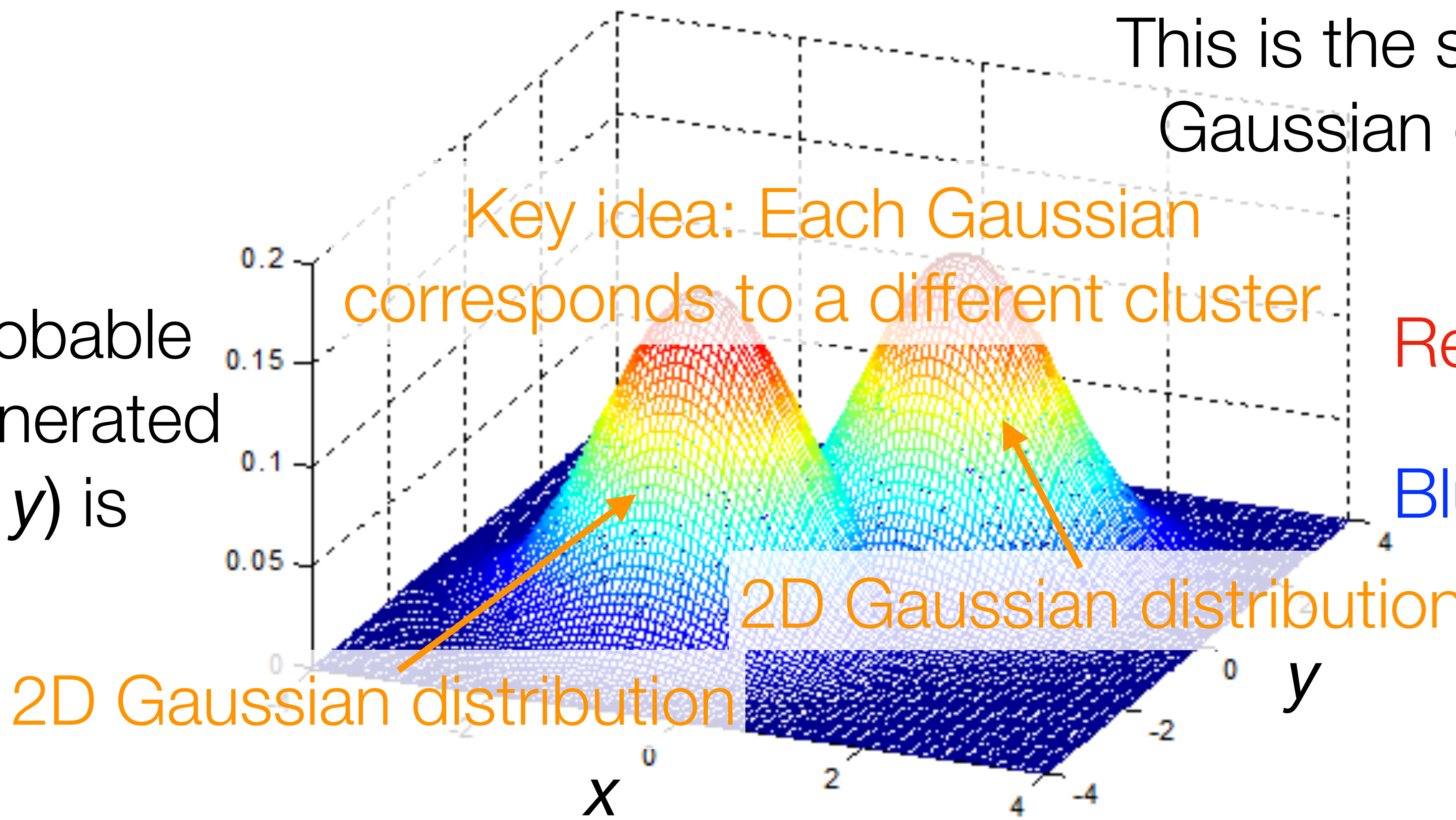
*Image source: https://www.intechopen.com/source/html/17742/media/image25.png*

# Gaussian Mixture Model (GMM)

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)

# 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

# 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)

# 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

# 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

# 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

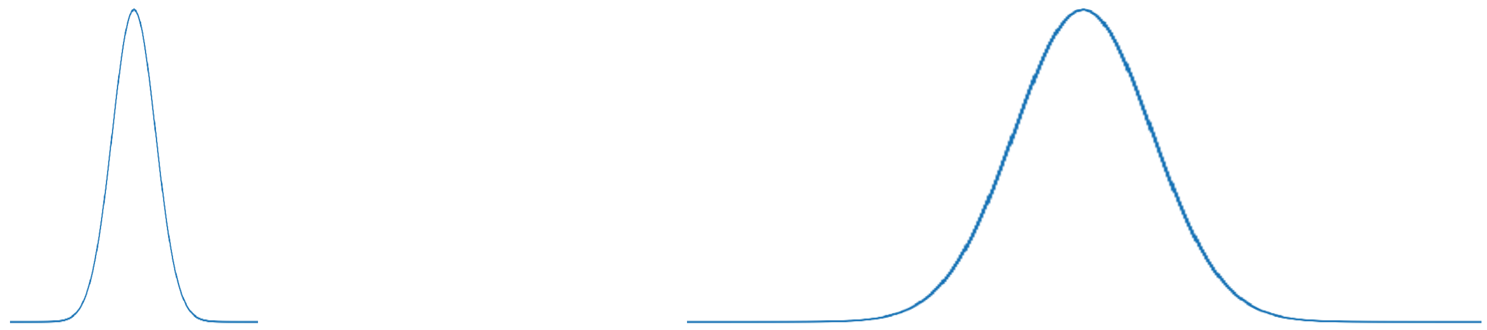
# 2D Gaussian Shape

# 2D Gaussian Shape

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

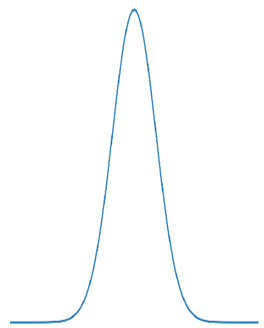
# 2D Gaussian Shape

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

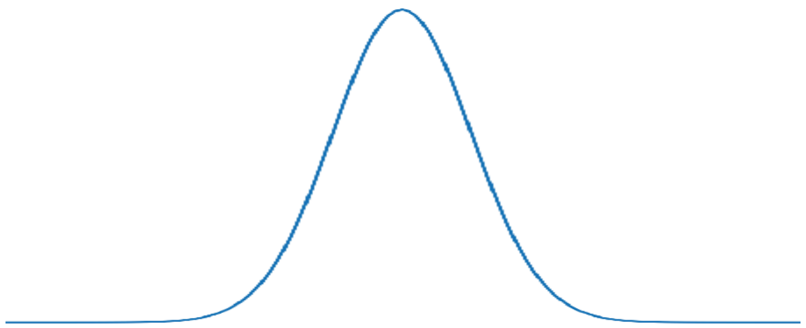


# 2D Gaussian Shape

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



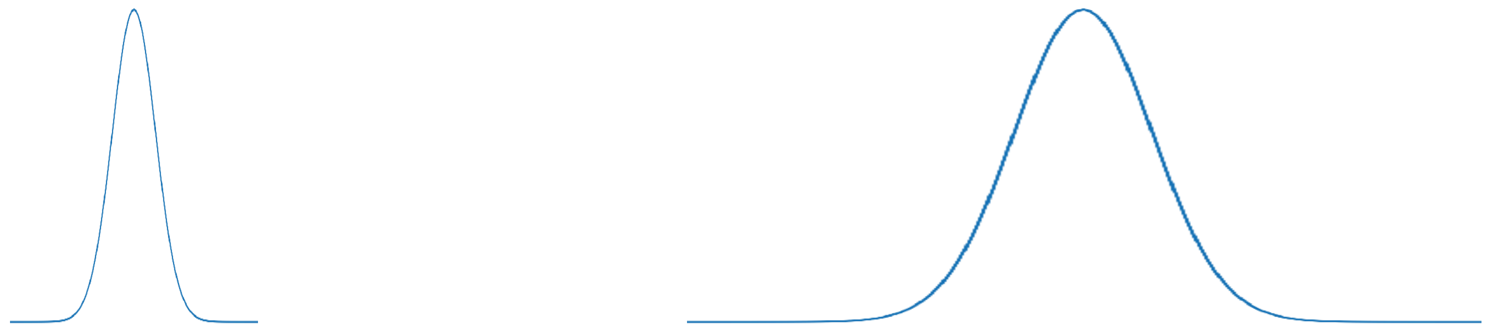
Less uncertainty



More uncertainty

# 2D Gaussian Shape

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



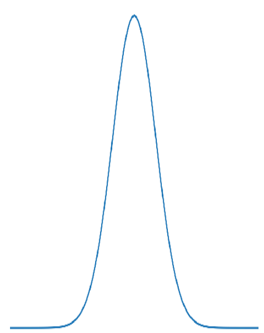
Less uncertainty

More uncertainty

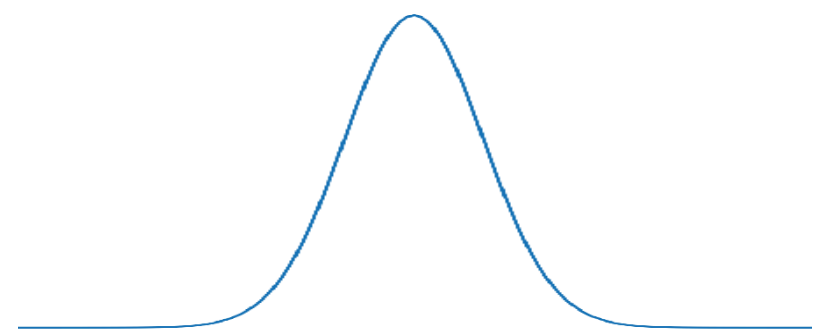
In 2D, you can more generally have ellipse-shaped Gaussians

# 2D Gaussian Shape

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

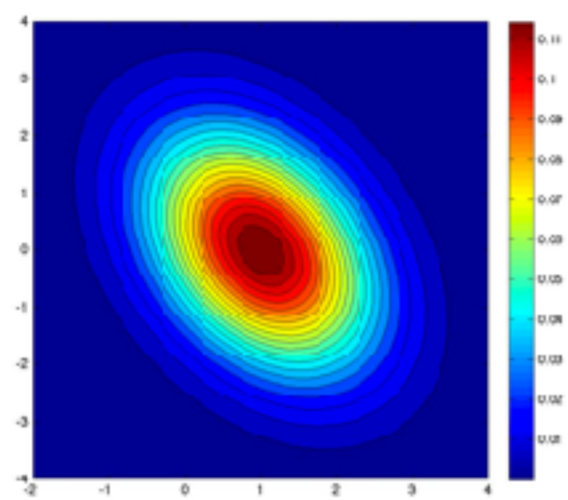


Less uncertainty



More uncertainty

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

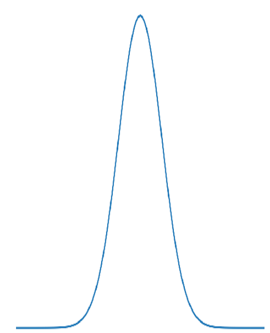


Top-down view of an example 2D Gaussian distribution

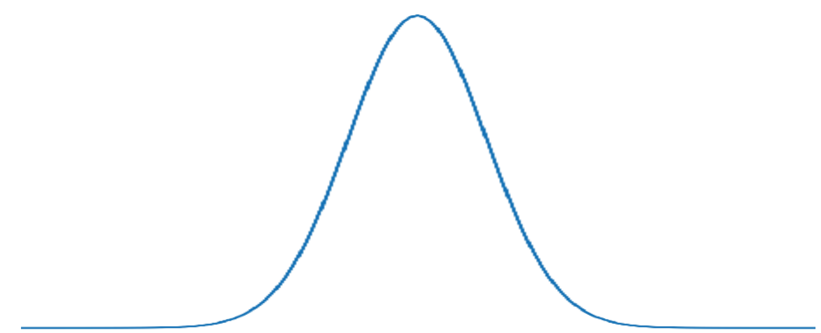
*Image source: https://www.cs.colorado.edu/~mozer/Teaching/syllabi/ProbabilisticModels2013/homework/assign5/a52dgauss.jpg*

# 2D Gaussian Shape

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



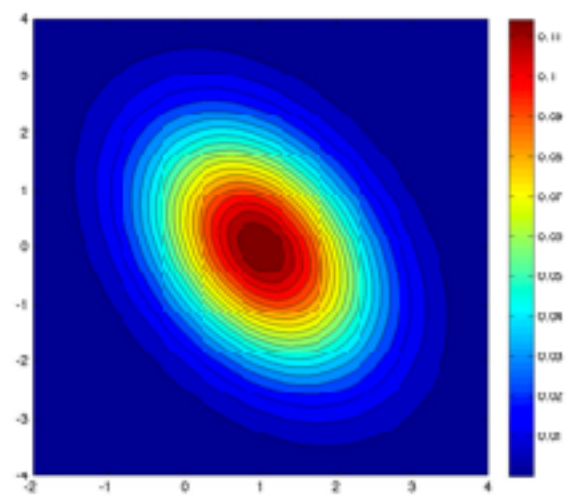
Less uncertainty



More uncertainty

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

Ellipse enables encoding relationship between variables

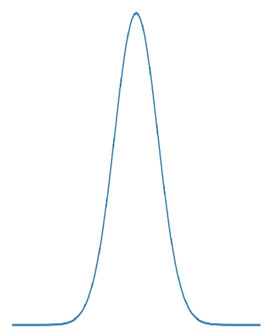


Top-down view of an example 2D Gaussian distribution

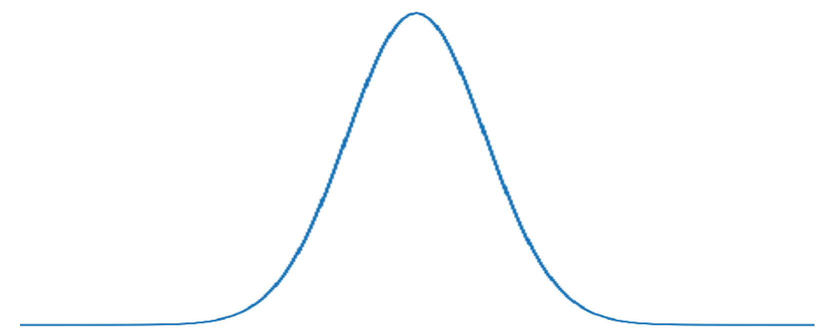
*Image source: https://www.cs.colorado.edu/~mozer/Teaching/syllabi/ProbabilisticModels2013/homework/assign5/a52dgauss.jpg*

# 2D Gaussian Shape

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



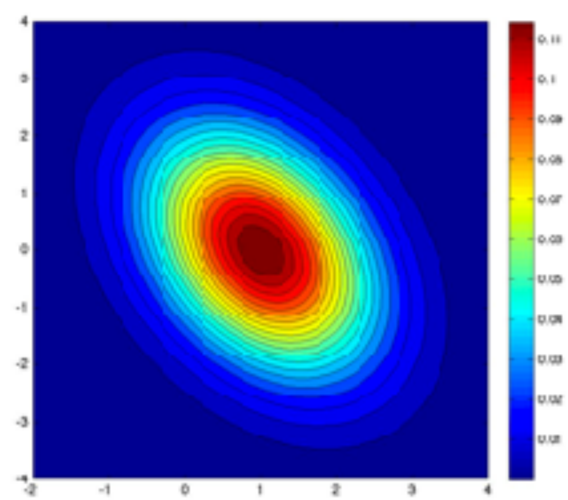
Less uncertainty



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

# 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

Cluster 1

Probability of generating a point from cluster 1 = 0.5

Gaussian mean = –5

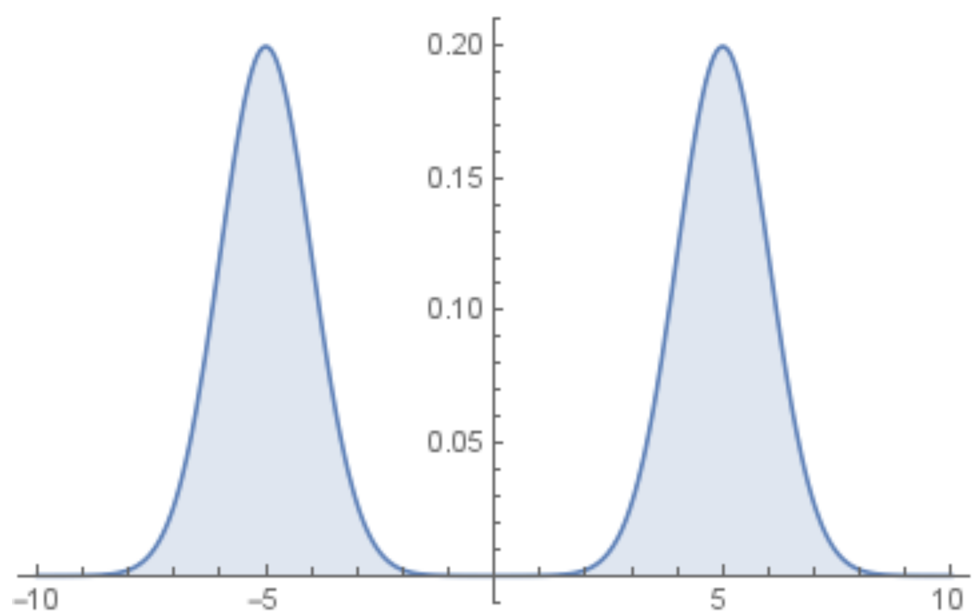
Gaussian std dev = 1

Cluster 2

Probability of generating a point from cluster 2 = 0.5

Gaussian mean = 5

Gaussian std dev = 1



# 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

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

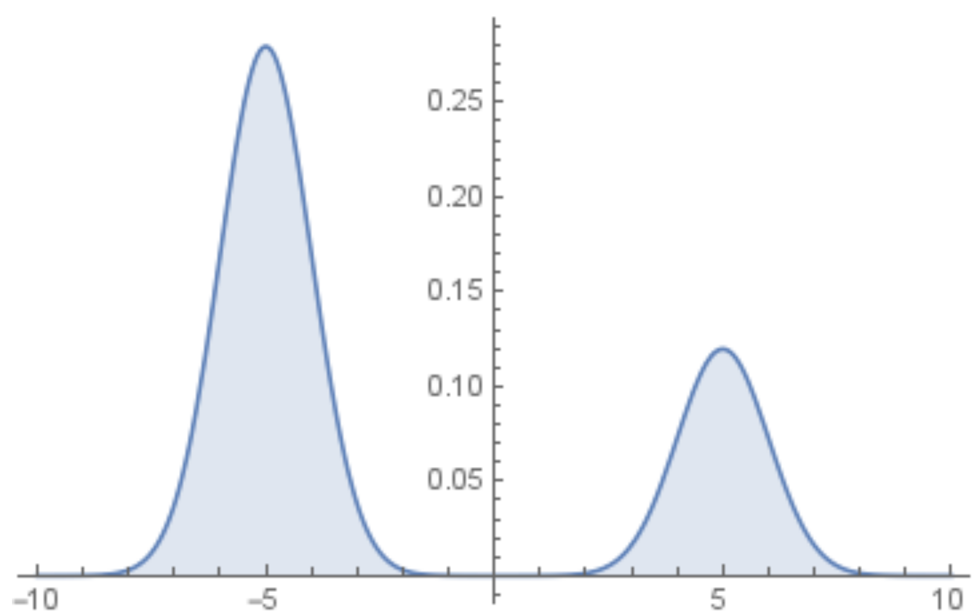
Gaussian std dev = 1

<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

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

# 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:

# 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

How to generate 1D points from this GMM:

1. Flip biased coin (with probability of heads 0.7)

# Example: 1D GMM with 2 Clusters

| Cluster 1 | Cluster 2 |
|---|---|
| Probability of generating a point from cluster 1 = 0.7 | Probability of generating a point from cluster 2 = 0.3 |
| Gaussian mean = –5 | Gaussian mean = 5 |
| Gaussian std dev = 1 | 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

# 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

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

<u>Cluster 1</u>

Probability of generating a point from cluster 1 = $\pi_1$

Gaussian mean = $\mu_1$

Gaussian std dev = $\sigma_1$

<u>Cluster 2</u>

Probability of generating a point from cluster 2 = $\pi_2$

Gaussian mean = $\mu_2$

Gaussian std dev = $\sigma_2$

How to generate 1D points from this GMM:

1. Flip biased coin (with probability of heads $\pi_1$)
2. If heads: sample 1 point from Gaussian mean $\mu_1$, std dev $\sigma_1$

   If tails: sample 1 point from Gaussian mean $\mu_2$, std dev $\sigma_2$

# Example: 1D GMM with *k* Clusters

Cluster 1

Probability of generating a point from cluster 1 = $\pi_1$

Gaussian mean = $\mu_1$

Gaussian std dev = $\sigma_1$

…

Cluster *k*

Probability of generating a point from cluster *k* = $\pi_k$

Gaussian mean = $\mu_k$

Gaussian std dev = $\sigma_k$

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 $1, \ldots, k$)
3. Sample 1 point from Gaussian mean $\mu_Z$, std dev $\sigma_Z$

# Example: 2D GMM with *k* Clusters

Cluster 1

Probability of generating a point from cluster 1 = $\pi_1$

Gaussian mean = $\mu_1$

Gaussian **covariance** = $\Sigma_1$

…

Cluster *k*

Probability of generating a point from cluster ***k*** = $\pi_k$

Gaussian mean = $\mu_k$

Gaussian **covariance** = $\Sigma_k$

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, …, *k*)
3. Sample 1 point from Gaussian mean $\mu_Z$, **covariance** $\Sigma_Z$

# Example: 2D GMM with *k* Clusters

Cluster 1

Probability of generating a point from cluster 1 = $\pi_1$

Gaussian mean = $\mu_1$ 2D point

Gaussian **covariance** = $\Sigma_1$

…

Cluster *k*

Probability of generating a point from cluster *k* = $\pi_k$

Gaussian mean = $\mu_k$ 2D point

Gaussian **covariance** = $\Sigma_k$

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 $\mu_Z$, **covariance** $\Sigma_Z$

# Example: 2D GMM with $k$ Clusters

<u>Cluster 1</u>

Probability of generating a point from cluster 1 = $\pi_1$

Gaussian mean = $\mu_1$ 2D point

Gaussian **covariance** = $\Sigma_1$

2x2 matrix

…

<u>Cluster $k$</u>

Probability of generating a point from cluster $k$ = $\pi_k$

Gaussian mean = $\mu_k$ 2D point

Gaussian **covariance** = $\Sigma_k$

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 $\mu_Z$, **covariance** $\Sigma_Z$

# GMM with $k$ Clusters

| Cluster 1 | | Cluster $k$ |
|---|---|---|
| Probability of generating a point from cluster 1 = $\pi_1$ | … | Probability of generating a point from cluster $k$ = $\pi_k$ |
| Gaussian mean = $\mu_1$ | | Gaussian mean = $\mu_k$ |
| Gaussian covariance = $\Sigma_1$ | | Gaussian covariance = $\Sigma_k$ |

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 $\mu_Z$, covariance $\Sigma_Z$

# High-Level Idea of GMM

# High-Level Idea of GMM

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

# 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!

# 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!

**“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!

# 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

# 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>

- Learning ("fitting") the parameters of a GMM
  - Input: *d*-dimensional data points, your guess for *k*

# 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: $\pi_1, \ldots, \pi_k, \mu_1, \ldots, \mu_k, \Sigma_1, \ldots, \Sigma_k$

# 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:

# 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

# 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

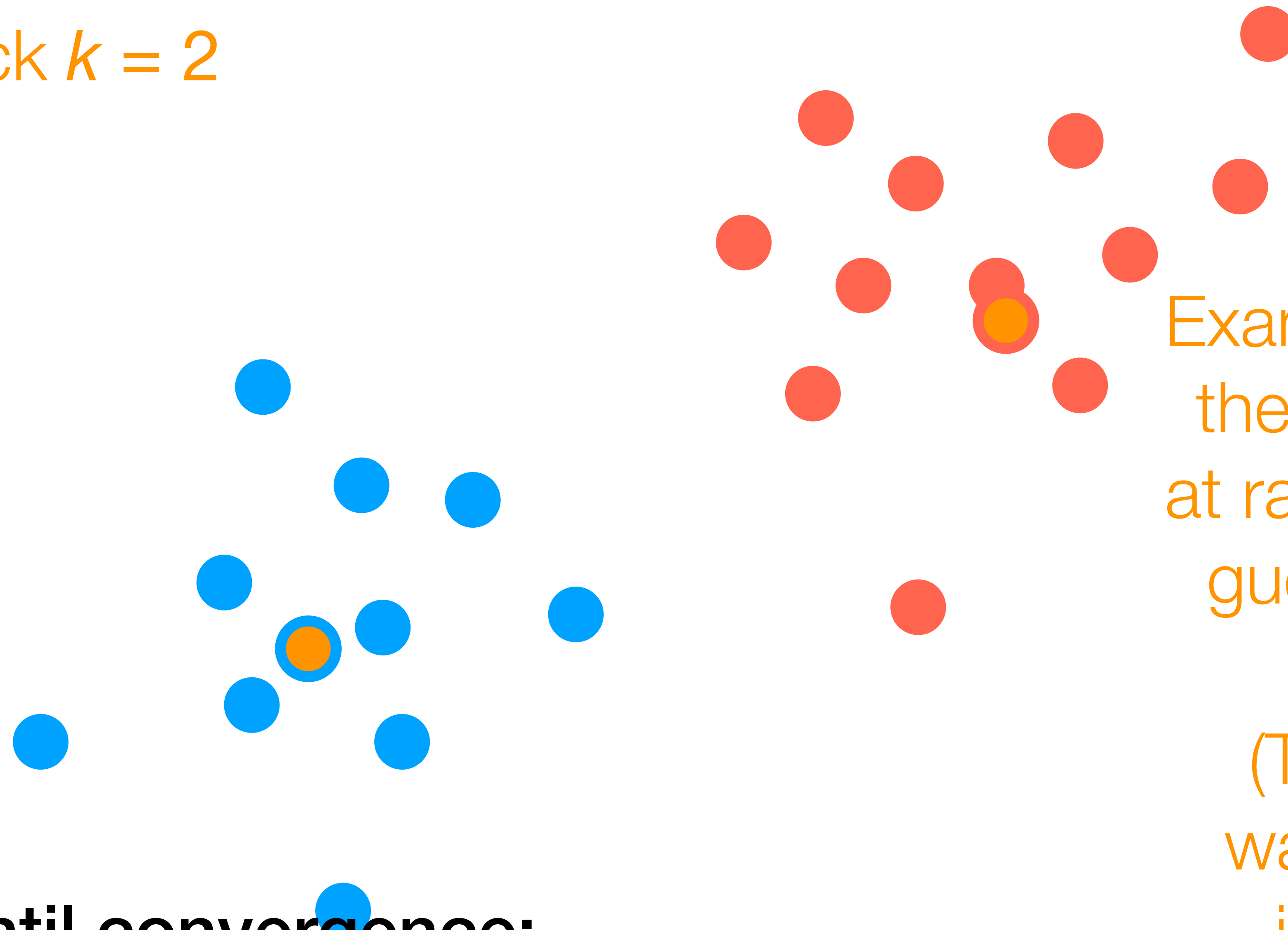
*How do you turn this into a cluster assignment?*

# *k*-means

Step 1: Pick 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)

**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 guesses 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 guesses for **cluster means and covariances**

**Repeat until convergence:**

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

Step 3: Update **cluster 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

# 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):

# 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

# 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

# Relating *k*-means to GMM's

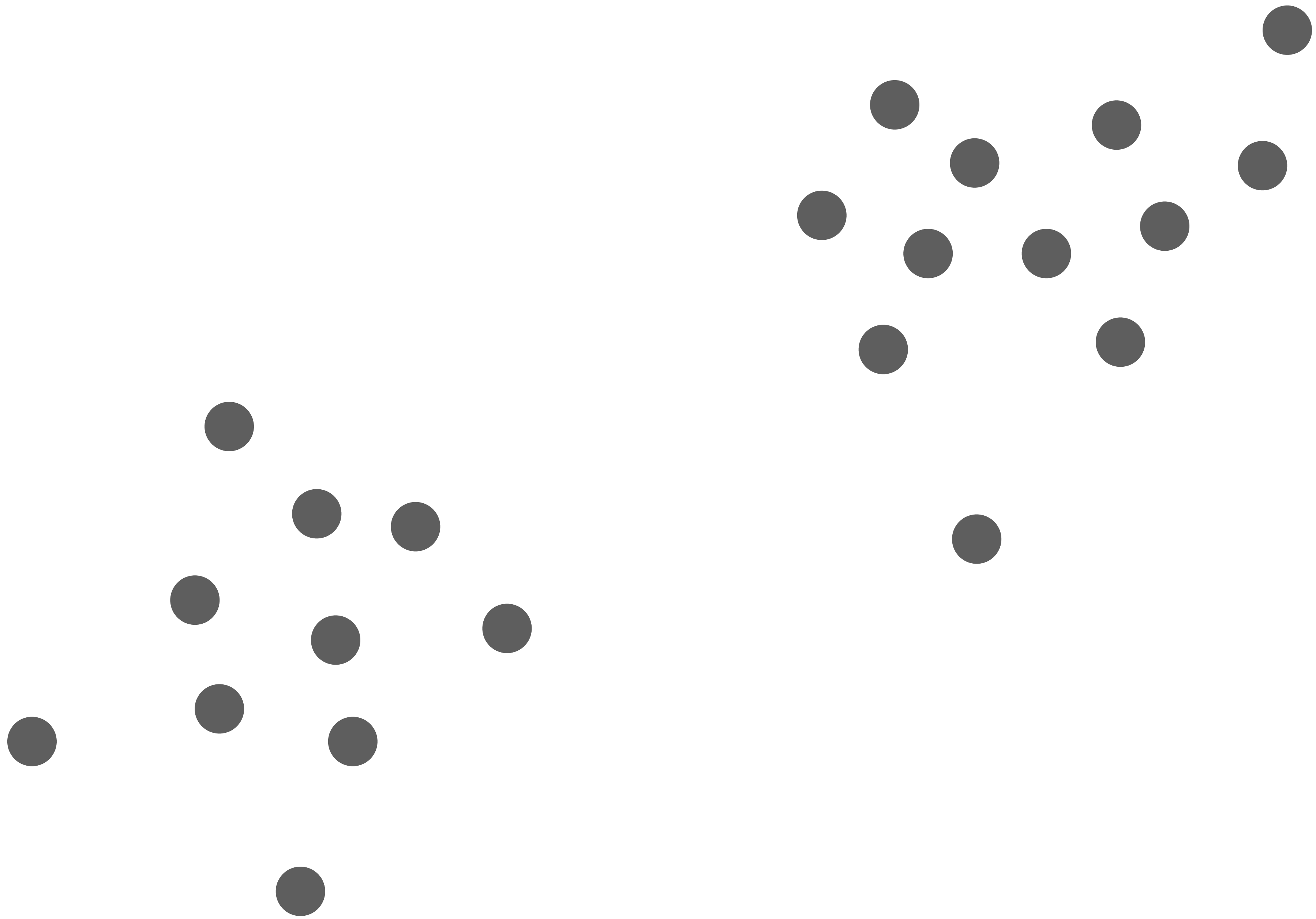
If the ellipses are all circles and have the same "skinniness" (e.g., in the 1D case it means they all have same std dev):

- *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