Carnegie Mellon University
Heinz College

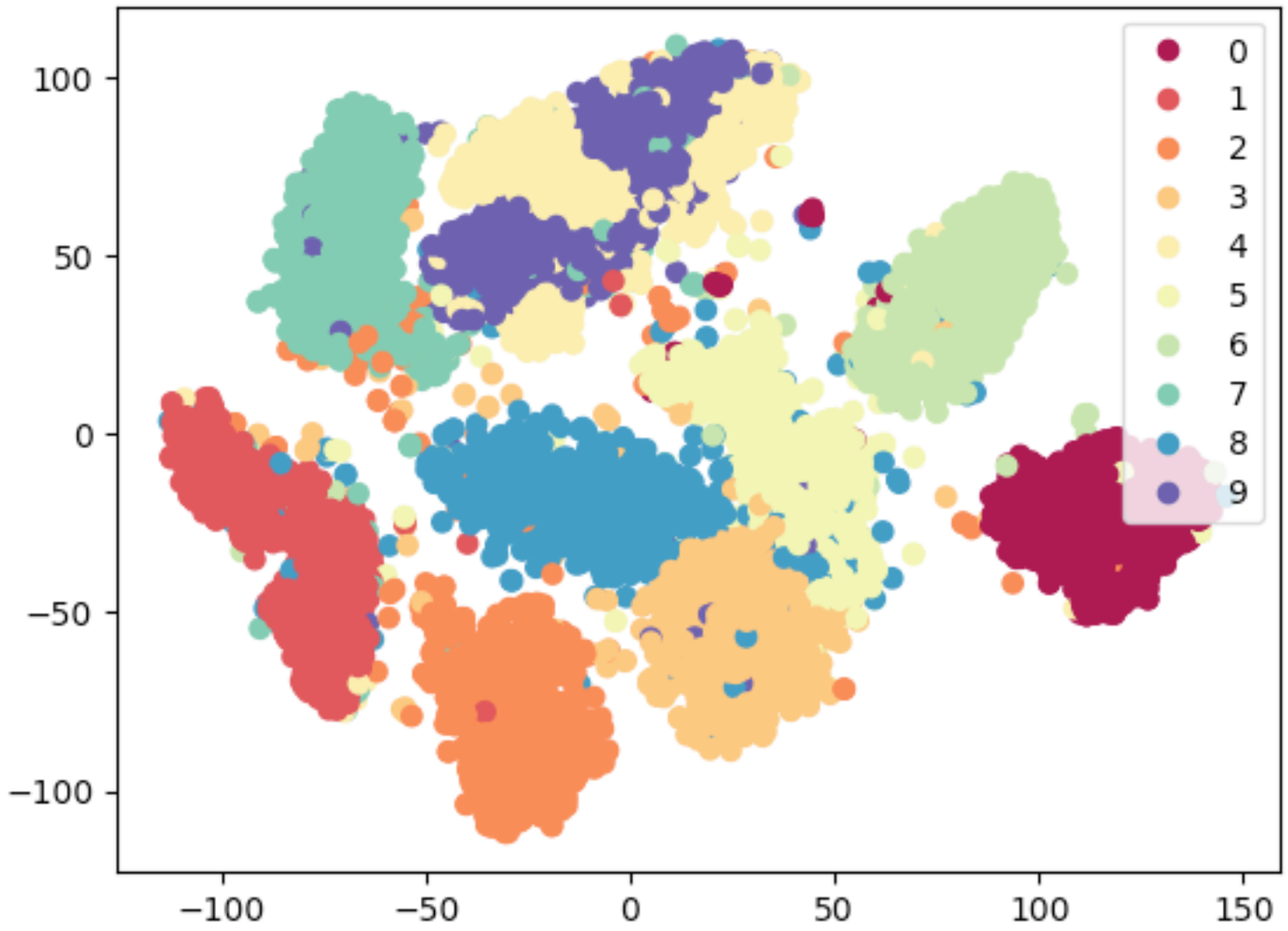
94-775 Unstructured Data Analytics for Policy

Lecture 7: Wrap up dimensionality reduction, start clustering

Slides by George H. Chen

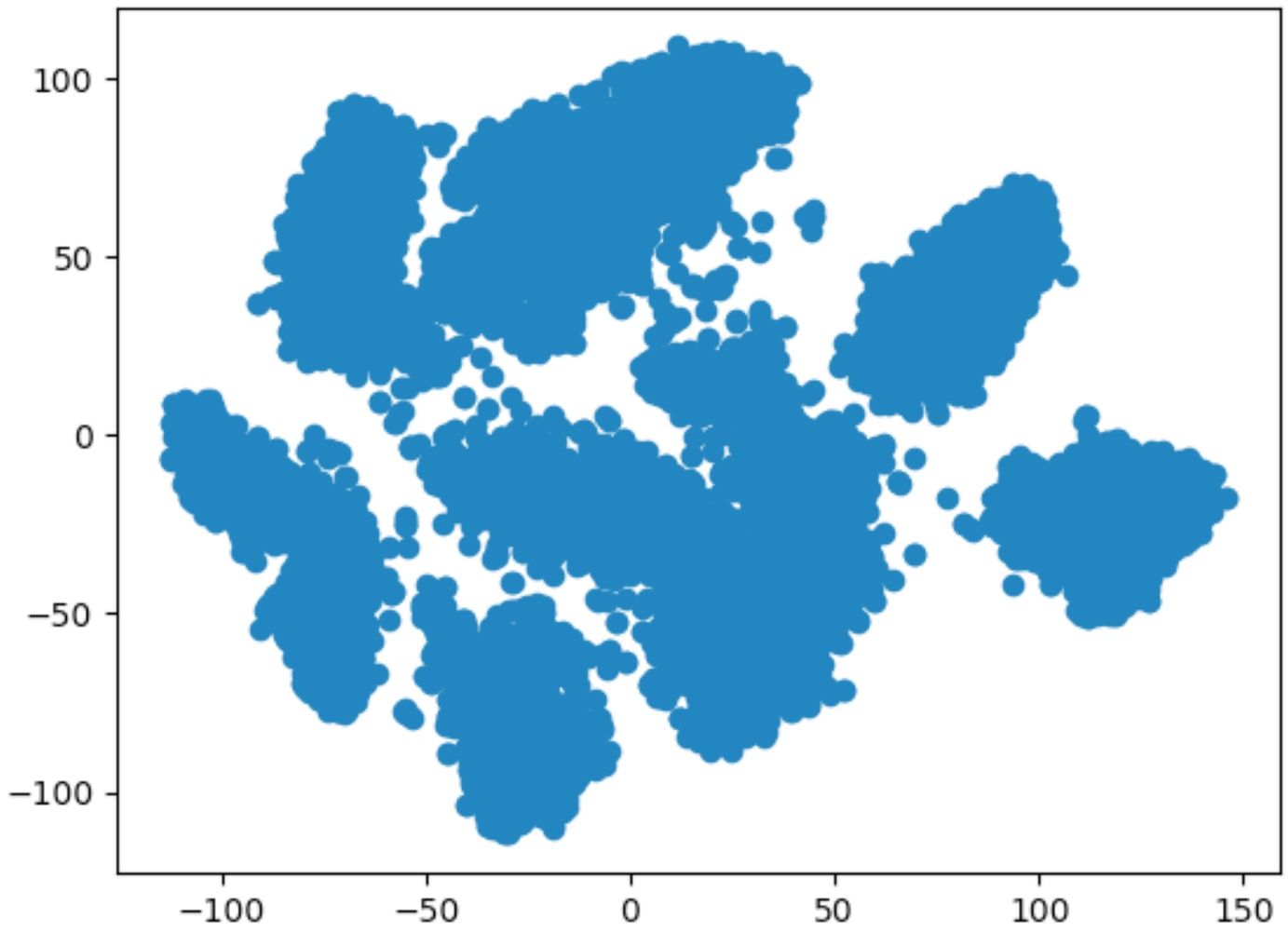
Dimensionality Reduction

- There are *many* methods for dimensionality reduction
- PCA (1901) is very well-understood; the new axes can be interpreted
- Nonlinear dimensionality reduction (manifold learning):
  new axes may not really be all that interpretable
- PCA is good to try first (look at plot & explained variance ratios)
  - If PCA works poorly, then t-SNE (2008) or UMAP (2018) could be a good 2nd thing to try (perhaps try UMAP first if your dataset is large)
  - In practice: if your data has clustering structure, t-SNE can often work well in showing (possibly even exaggerating) this clustering structure in the low-dim. space compared to UMAP
  - UMAP is *not* always better than t-SNE and t-SNE is *not* always better than UMAP — both are commonly used now though so it's good to know about the existence of both!



t-SNE plot of handwritten digit images shows clumps that correspond to digits — this is an example of **clustering structure** arising in real data

If we don't have label information, how do we color the points?



Goal of clustering algorithm: figure out how to color the points
(each color corresponds to a different cluster)

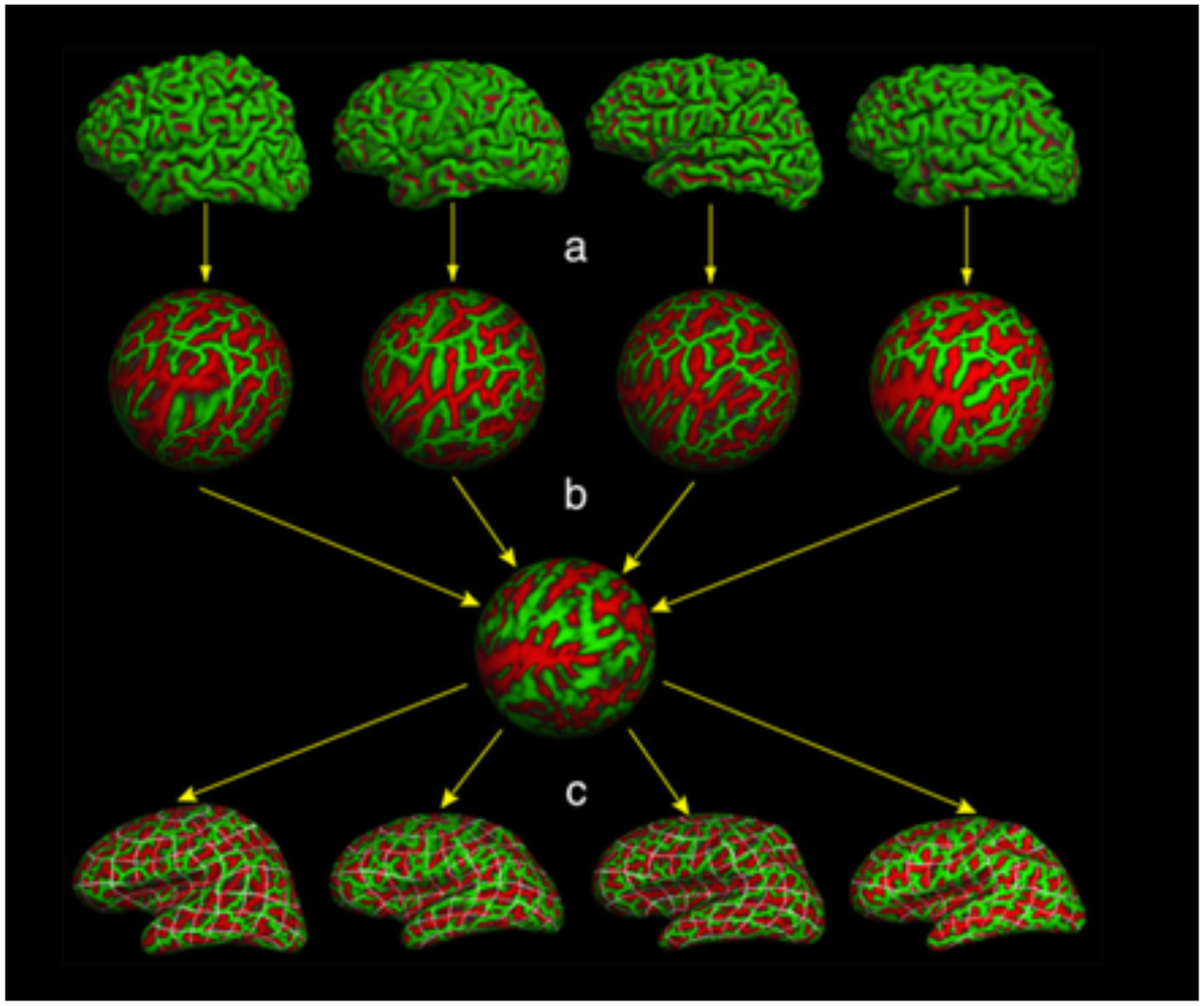
Want points within the same cluster to be "similar"
& points in different clusters to be "dissimilar"

But what does "similar" mean?

By far the most popular approach: if your data are represented as feature vectors, use Euclidean distance between feature vectors

You may have to cleverly define the feature vectors so that Euclidean distance between them actually is meaningful!

Example: Comparing Different People's Brain Scans



FreeSurfer software: nonlinearly "align" different people's brain scans by mapping them to a common coordinate system on a sphere

After nonlinearing mapping people to the common coordinate system, we can use Euclidean distance in that common coordinate system!

Example of Non-Euclidean Distance: 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 edit distance or Levenshtein distance)

Clustering

- There are *lots* of clustering methods out there!
- I know that y'all saw some clustering methods in 90-803
- In 94-775 this semester, we'll cover two main approaches, both of which can scale reasonably well to large datasets:
  - k-means (this should be a review from 90-803)
  - HDBSCAN (very good to try when you don't know how to choose the number of clusters)

By default, these use Euclidean distance

- In terms of analyzing unstructured data, the goal of clustering is often to help **summarize data**
  - We'll see this idea in demos

We're going to start with perhaps the most famous of clustering methods: *k*-means

k-means

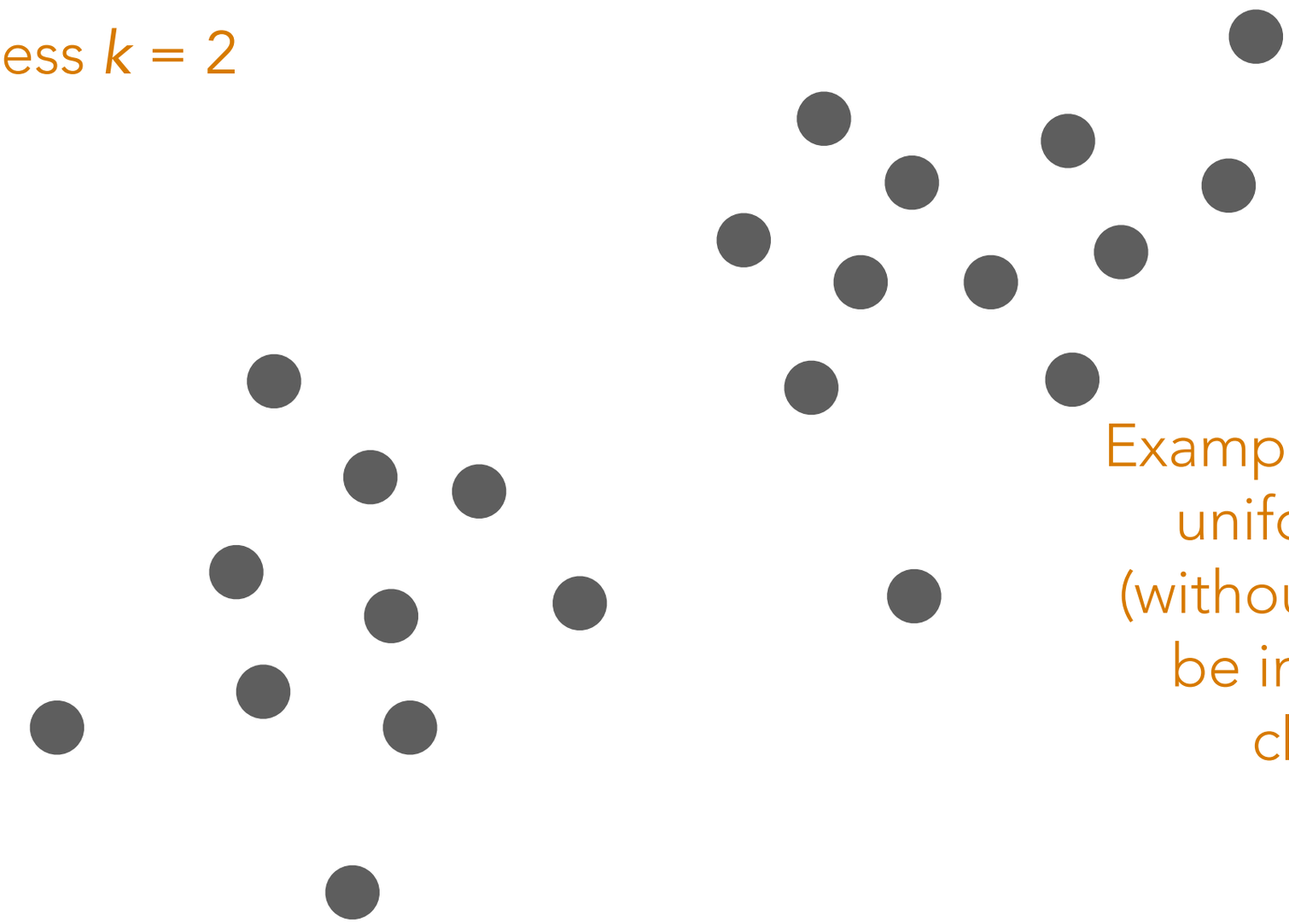
of clusters

Distance function: Euclidean

Step 0: Guess k

We'll guess $k = 2$

Step 1: Guess where cluster centers are



Example: choose k points uniformly at random (without replacement) to be initial guesses for cluster centers

k-means

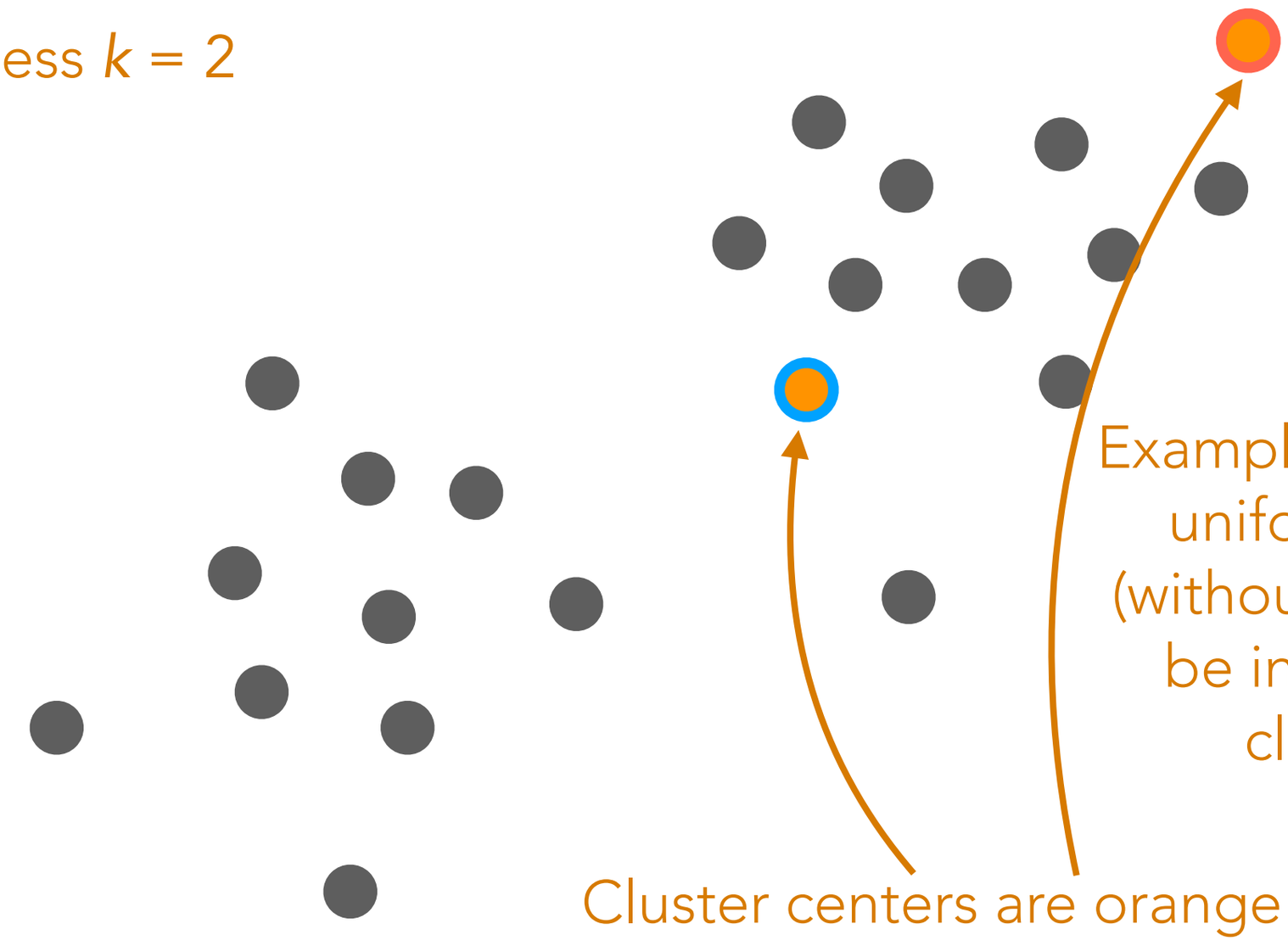
of clusters

Distance function: Euclidean

Step 0: Guess k

We'll guess $k = 2$

Step 1: Guess where cluster centers are



Example: choose k points uniformly at random (without replacement) to be initial guesses for cluster centers

Cluster centers are orange points (outline says whether it is the blue or red cluster)

k-means

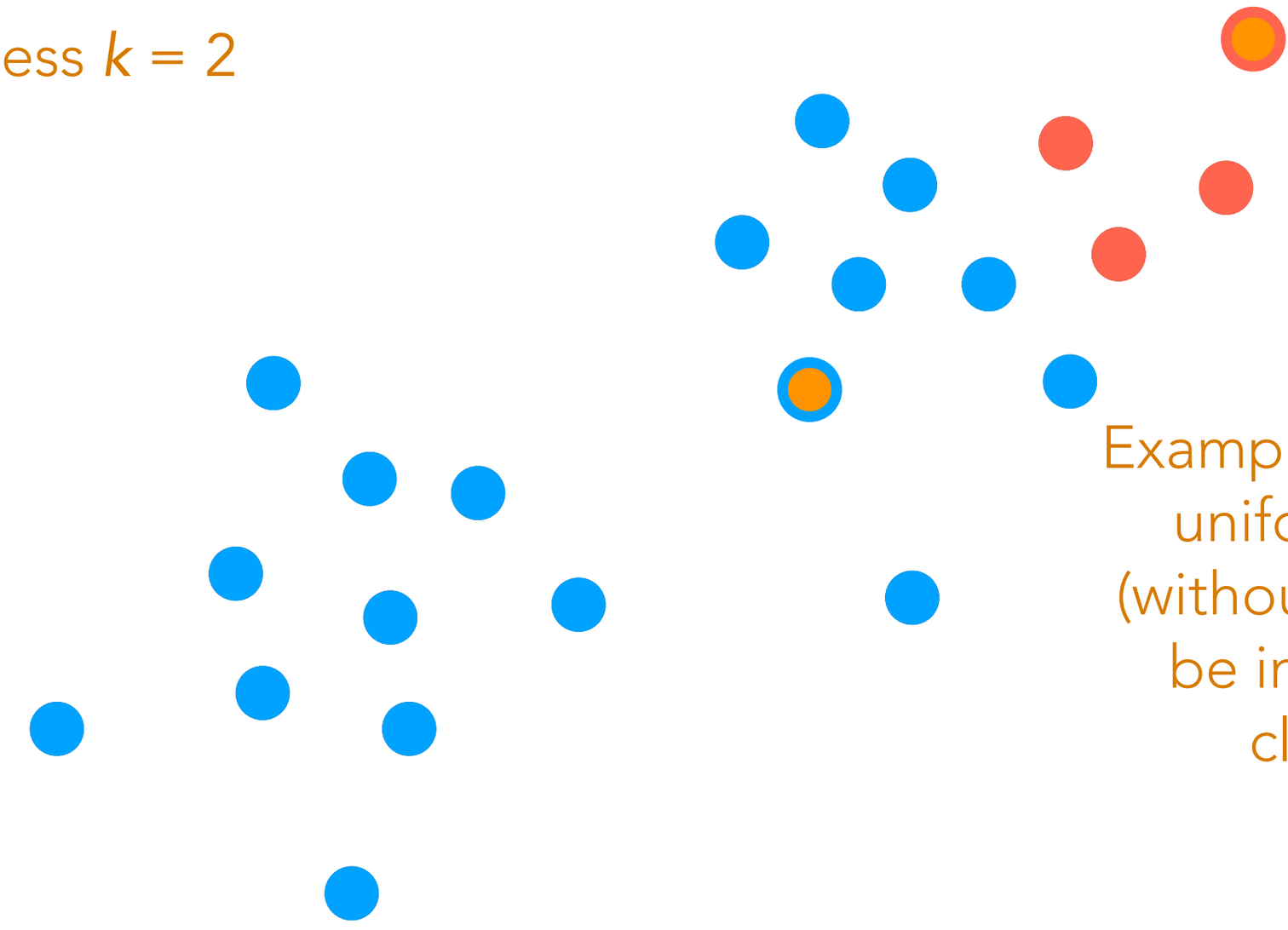
of clusters

Distance function: Euclidean

Step 0: Guess k

We'll guess $k = 2$

Step 1: Guess where cluster centers are



Example: choose k points uniformly at random (without replacement) to be initial guesses for cluster centers

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

k-means

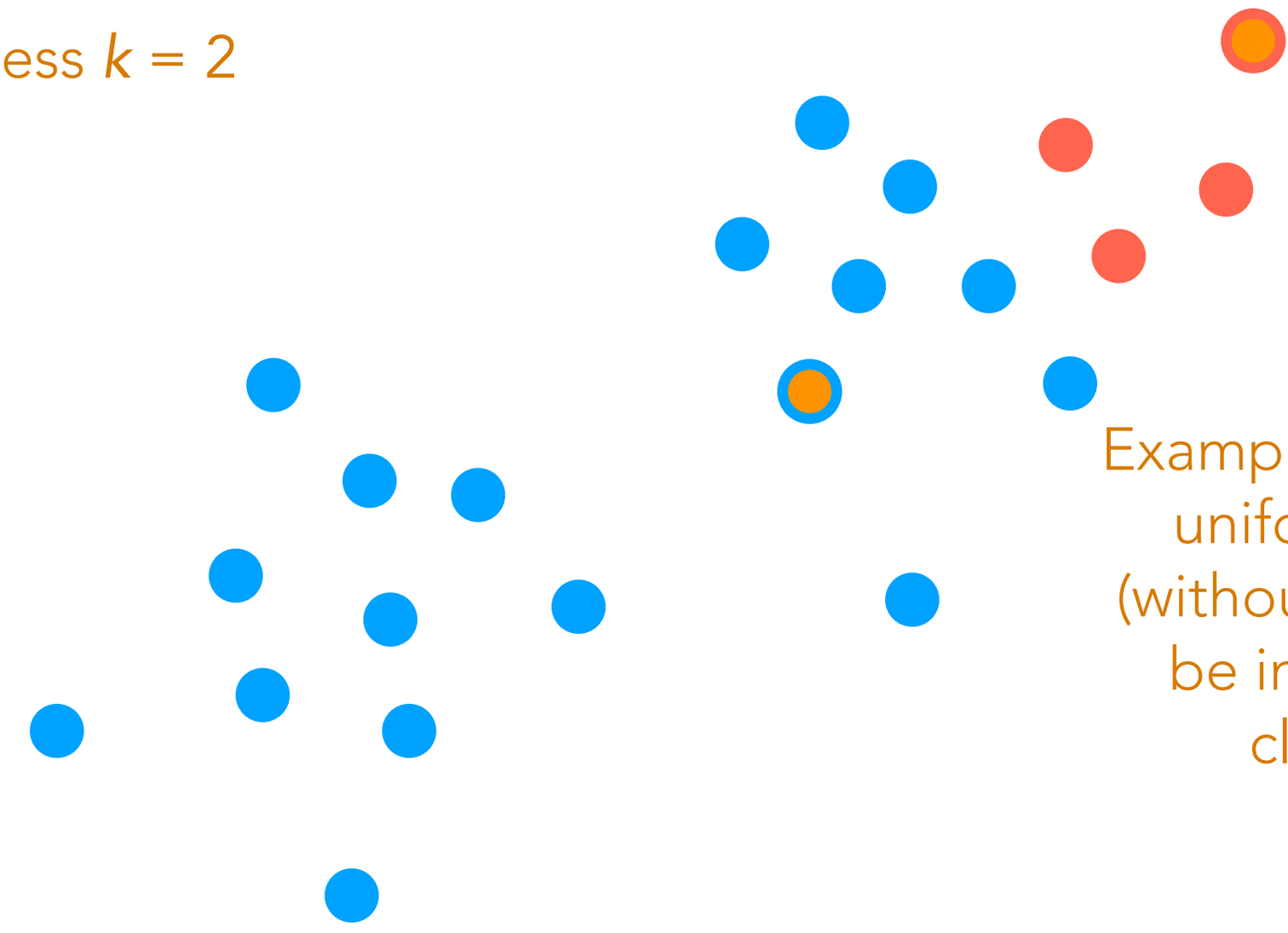
of clusters

Distance function: Euclidean

Step 0: Guess k

We'll guess $k = 2$

Step 1: Guess where cluster centers are



Example: choose k points uniformly at random (without replacement) to be initial guesses for cluster centers

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

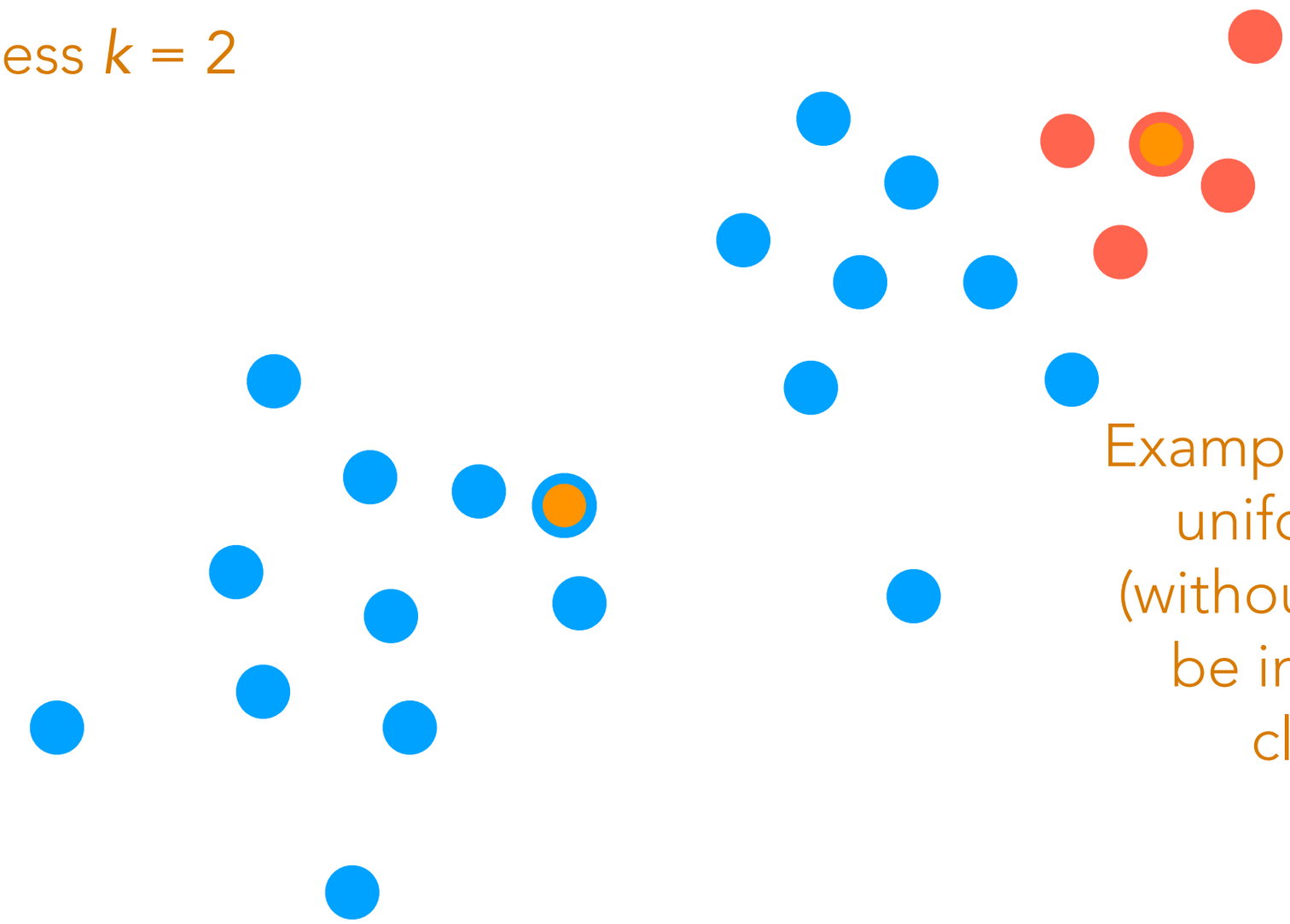
of clusters

Distance function: Euclidean

Step 0: Guess k

We'll guess $k = 2$

Step 1: Guess where cluster centers are



Example: choose k points uniformly at random (without replacement) to be initial guesses for cluster centers

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

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

# of clusters

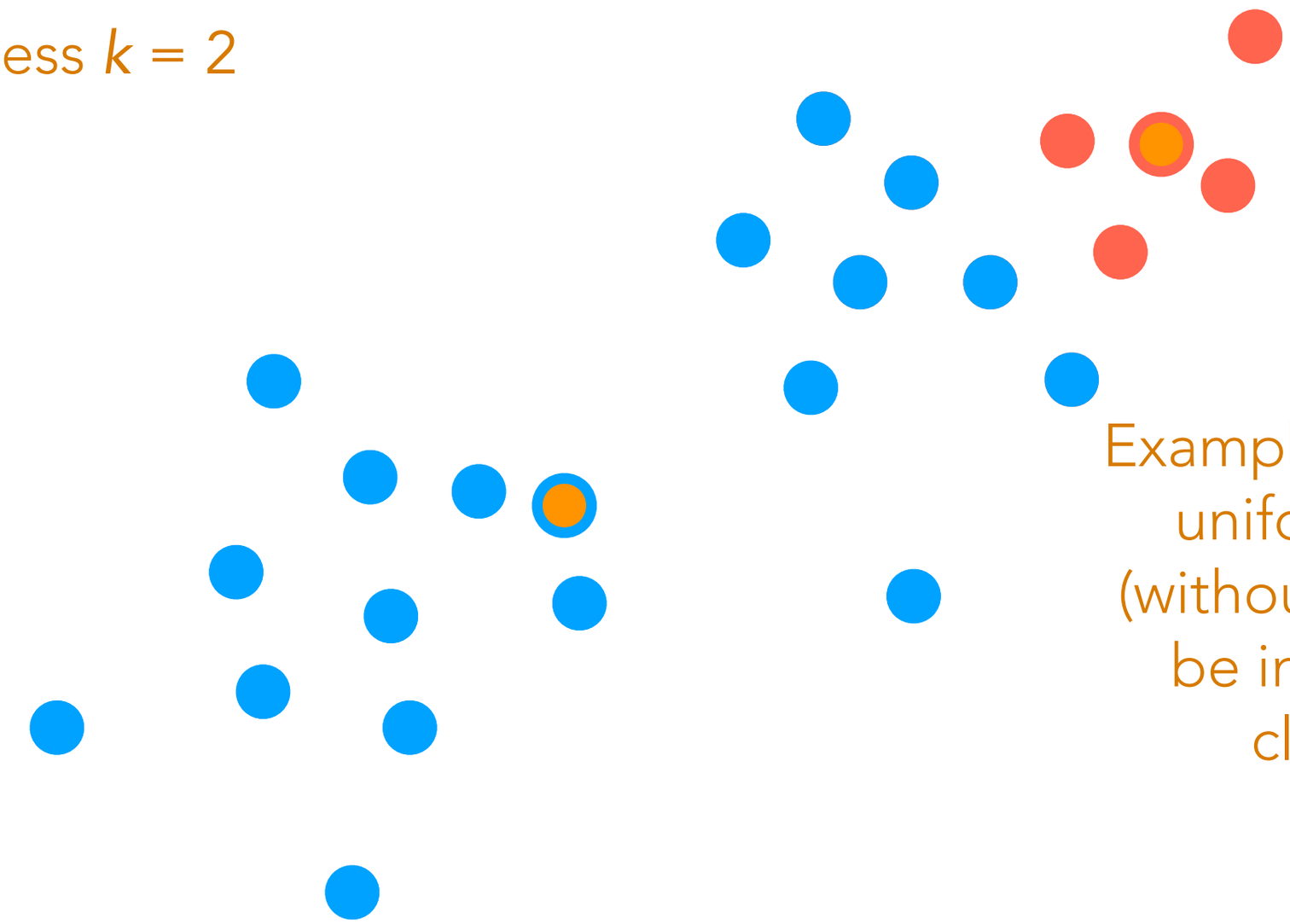
k-means

Distance function: Euclidean

Step 0: Guess *k*

We'll guess *k* = 2

Step 1: Guess where cluster centers are



Example: choose *k* points uniformly at random (without replacement) to be initial guesses for cluster centers

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)

# of clusters

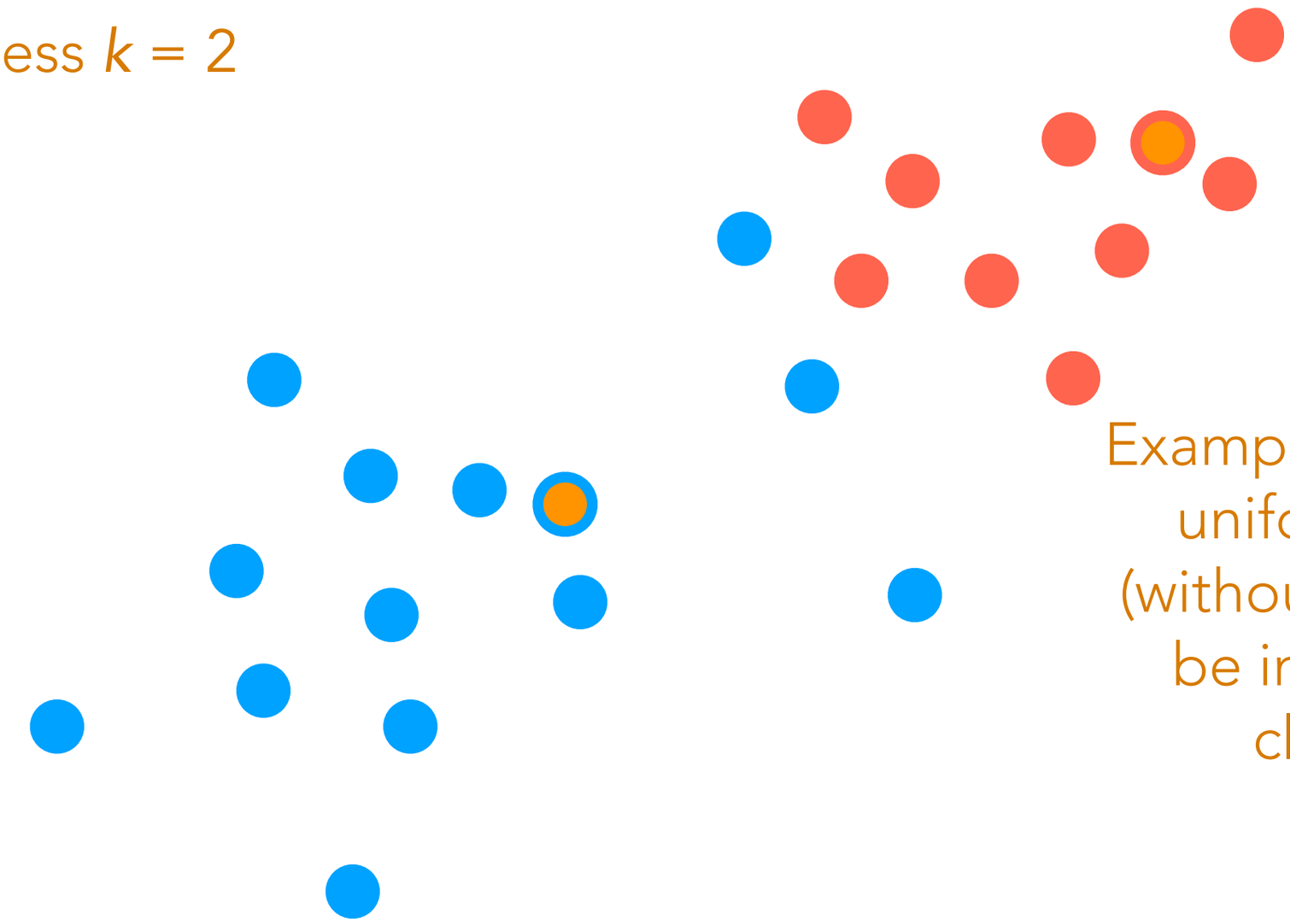
k-means

Distance function: Euclidean

Step 0: Guess *k*

We'll guess *k* = 2

Step 1: Guess where cluster centers are



Example: choose *k* points uniformly at random (without replacement) to be initial guesses for cluster centers

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

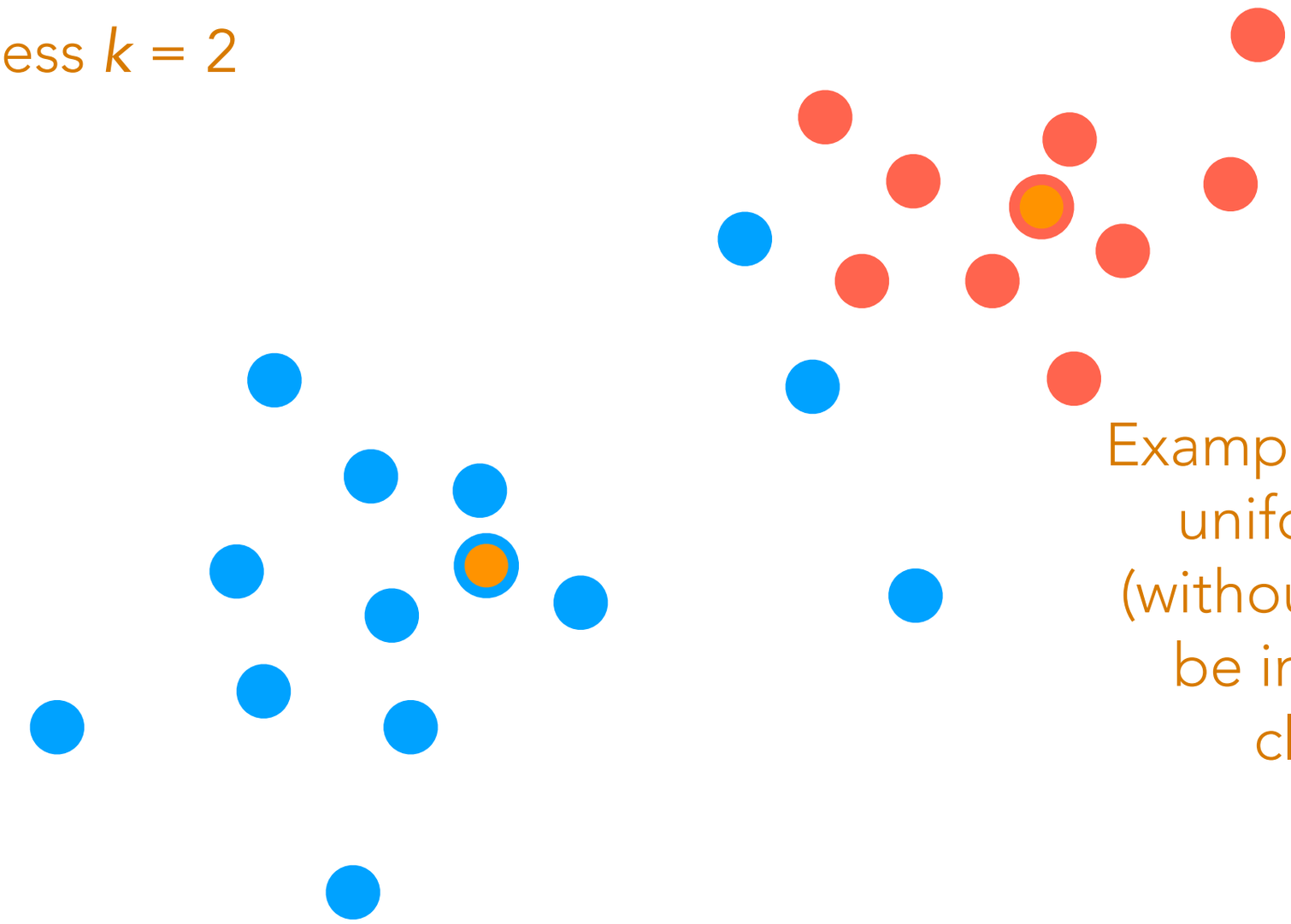
of clusters

Distance function: Euclidean

Step 0: Guess k

We'll guess $k = 2$

Step 1: Guess where cluster centers are



Example: choose k points uniformly at random (without replacement) to be initial guesses for cluster centers

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

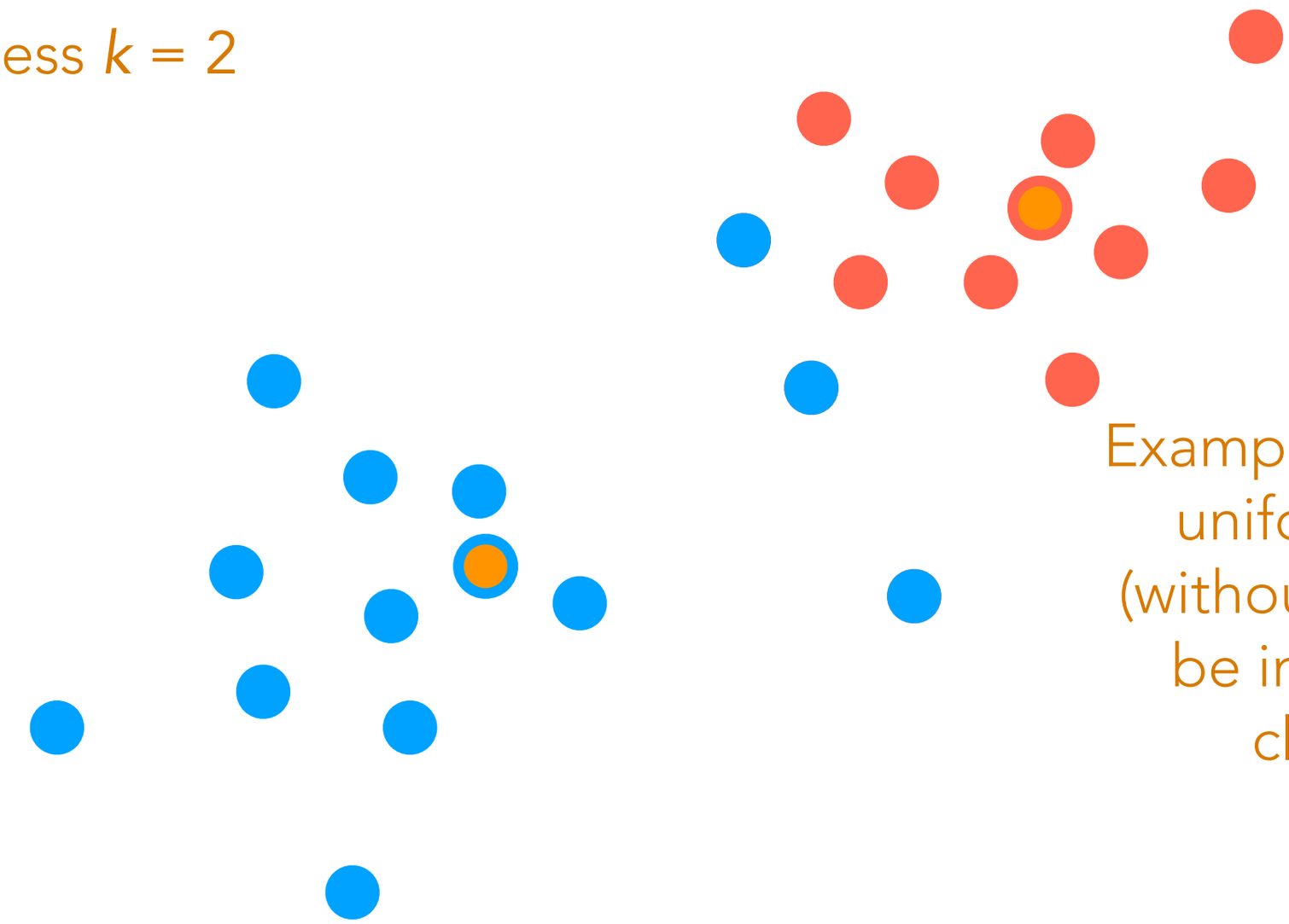
of clusters

Distance function: Euclidean

Step 0: Guess k

We'll guess $k = 2$

Step 1: Guess where cluster centers are



Example: choose k points uniformly at random (without replacement) to be initial guesses for cluster centers

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)

# of clusters

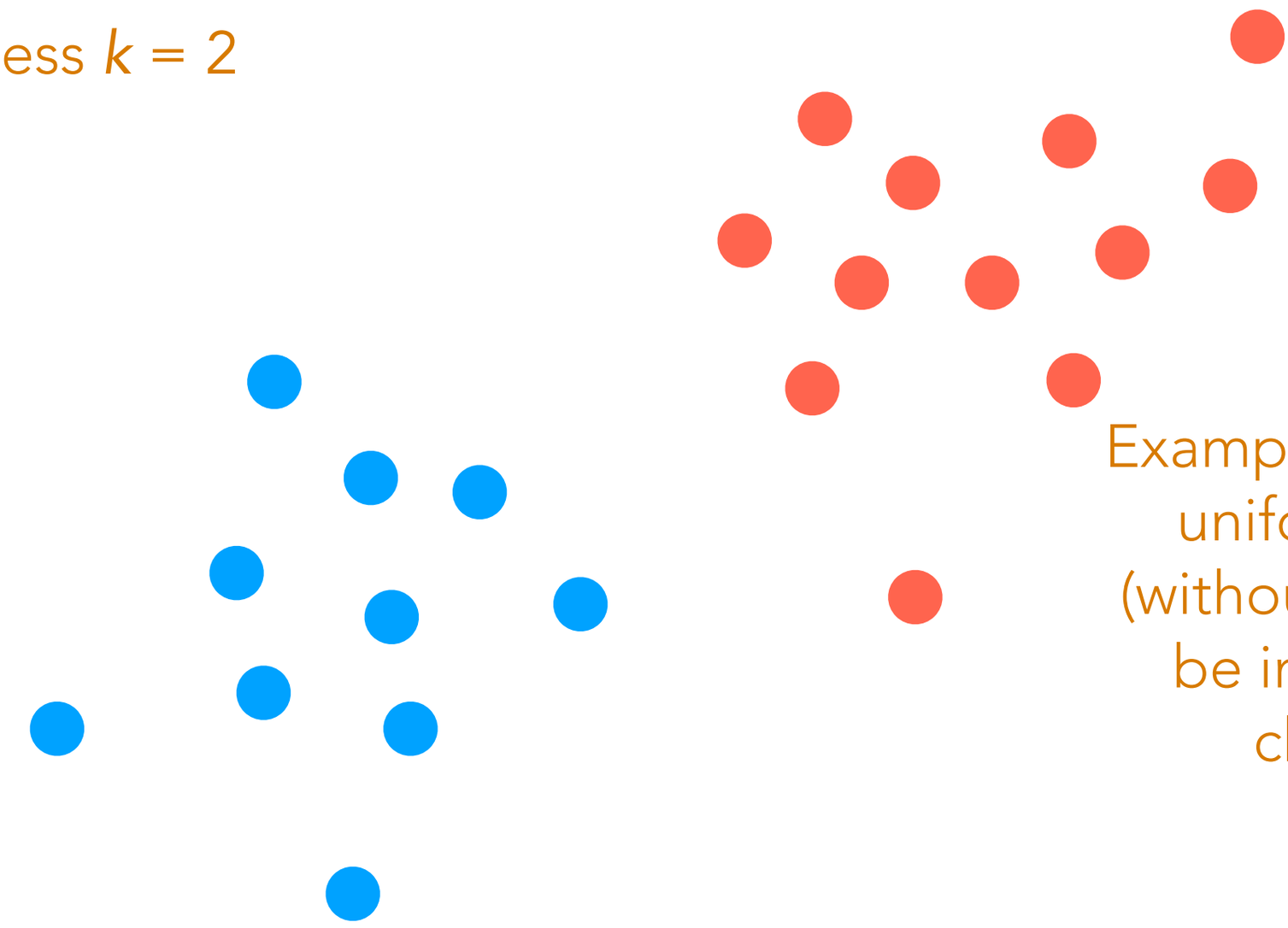
k-means

Distance function: Euclidean

Step 0: Guess *k*

We'll guess *k* = 2

Step 1: Guess where cluster centers are



Example: choose *k* points uniformly at random (without replacement) to be initial guesses for cluster centers

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

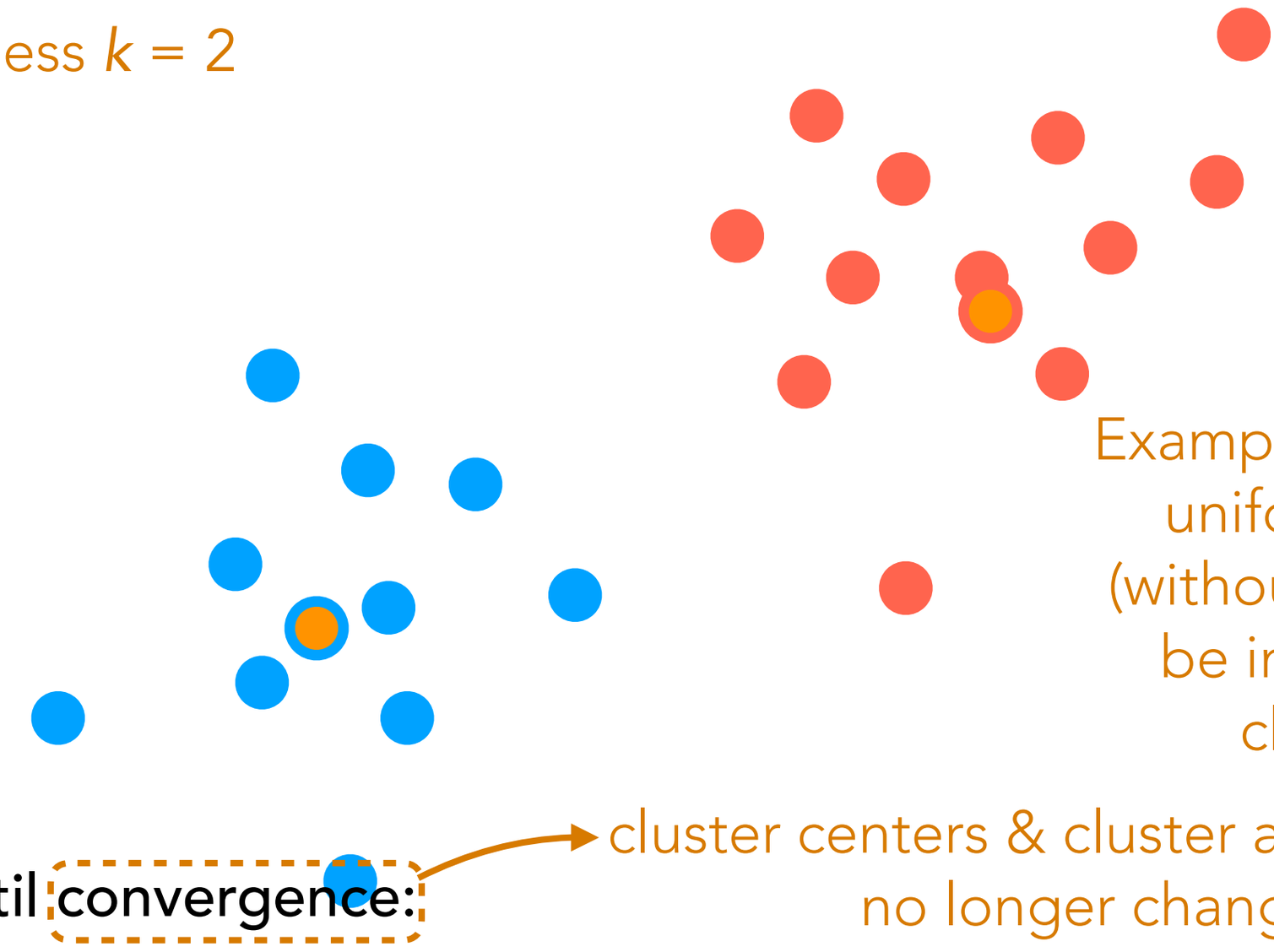
of clusters

Distance function: Euclidean

Step 0: Guess k

We'll guess $k = 2$

Step 1: Guess where cluster centers are



Example: choose k points uniformly at random (without replacement) to be initial guesses for cluster centers

Repeat until convergence:

cluster centers & cluster assignments no longer change

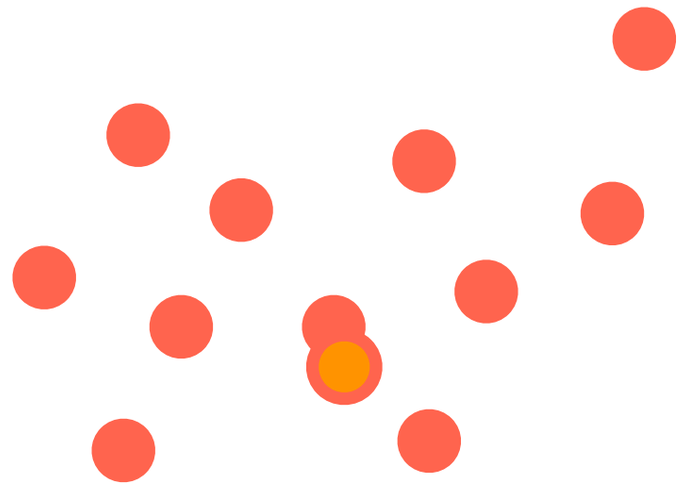
Step 2: Assign each point to belong to the closest cluster

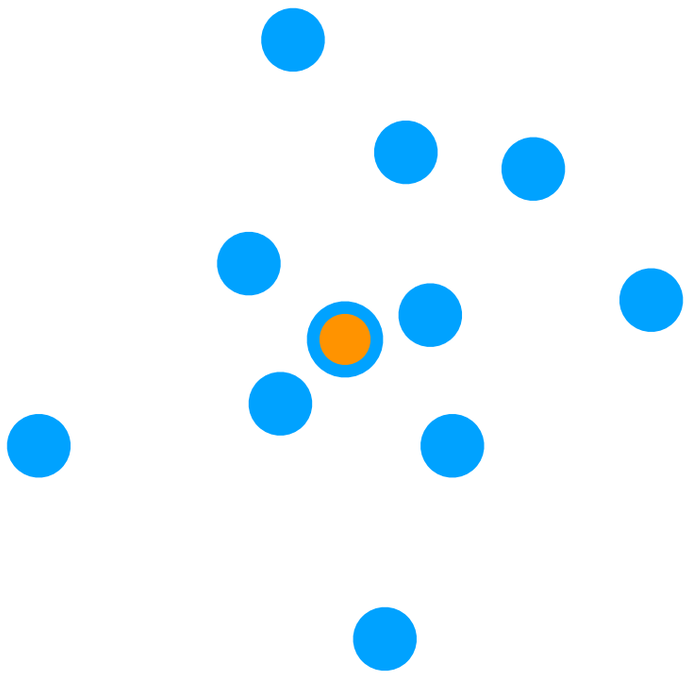
Step 3: Update cluster means (to be the center of mass per cluster)

k-means

Final output: cluster centers, cluster assignment for every point

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





How to choose *k*?

We'll discuss this next lecture

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)

Clustering on Text

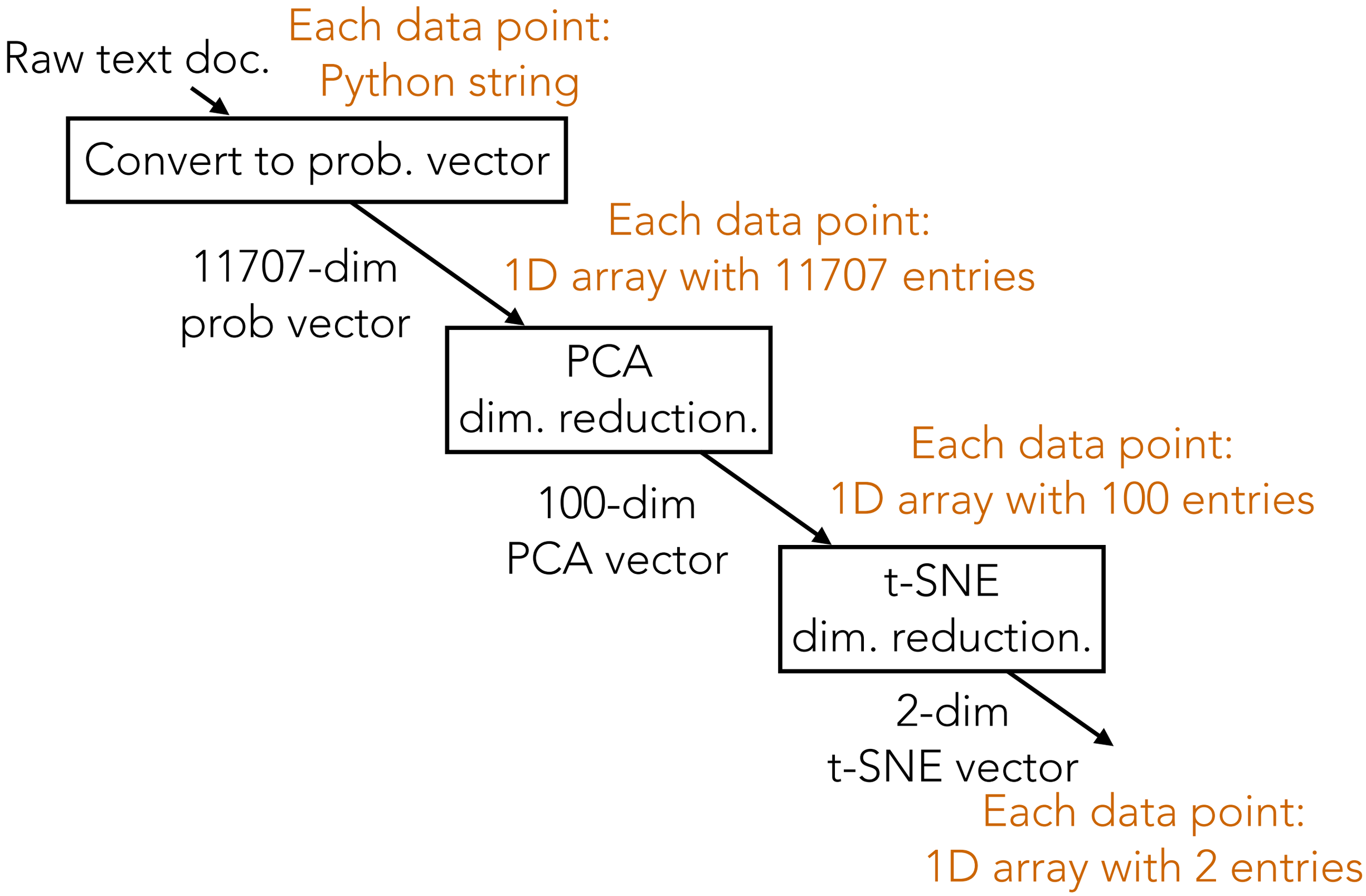
Demo

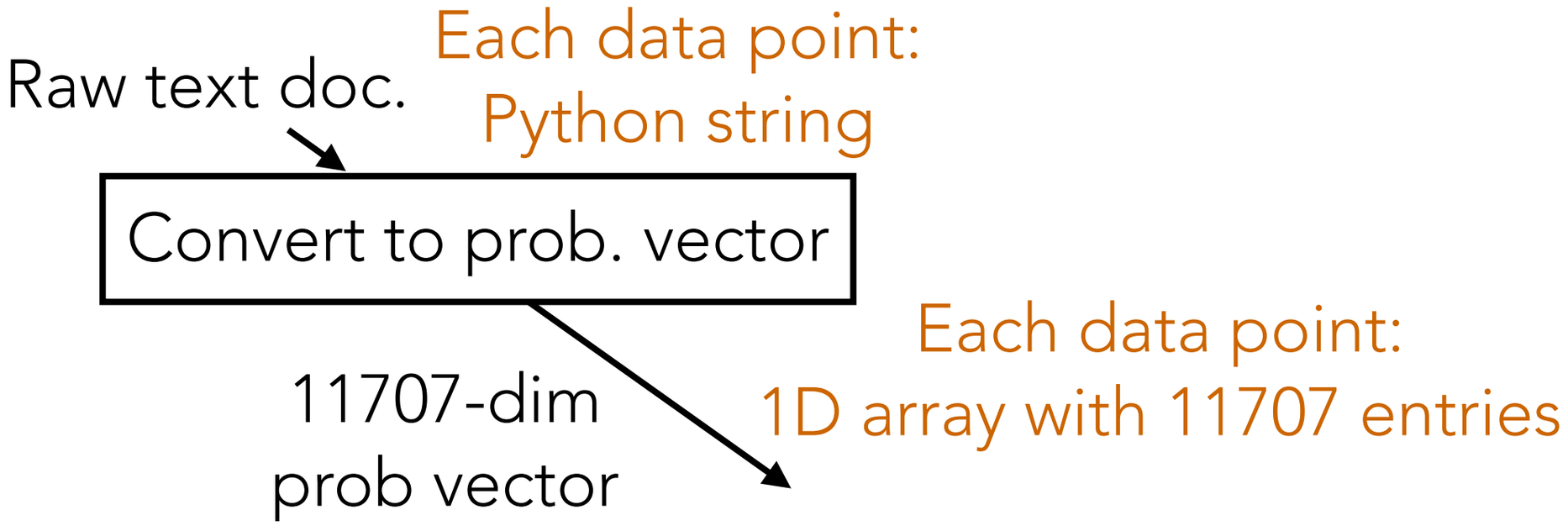
Warning: This demo is very action-packed &
we might not complete it until next week

The initial chunk of it isn't even about clustering &
is on preprocessing the data *before* clustering

Recap of Start of Clustering on Text Demo

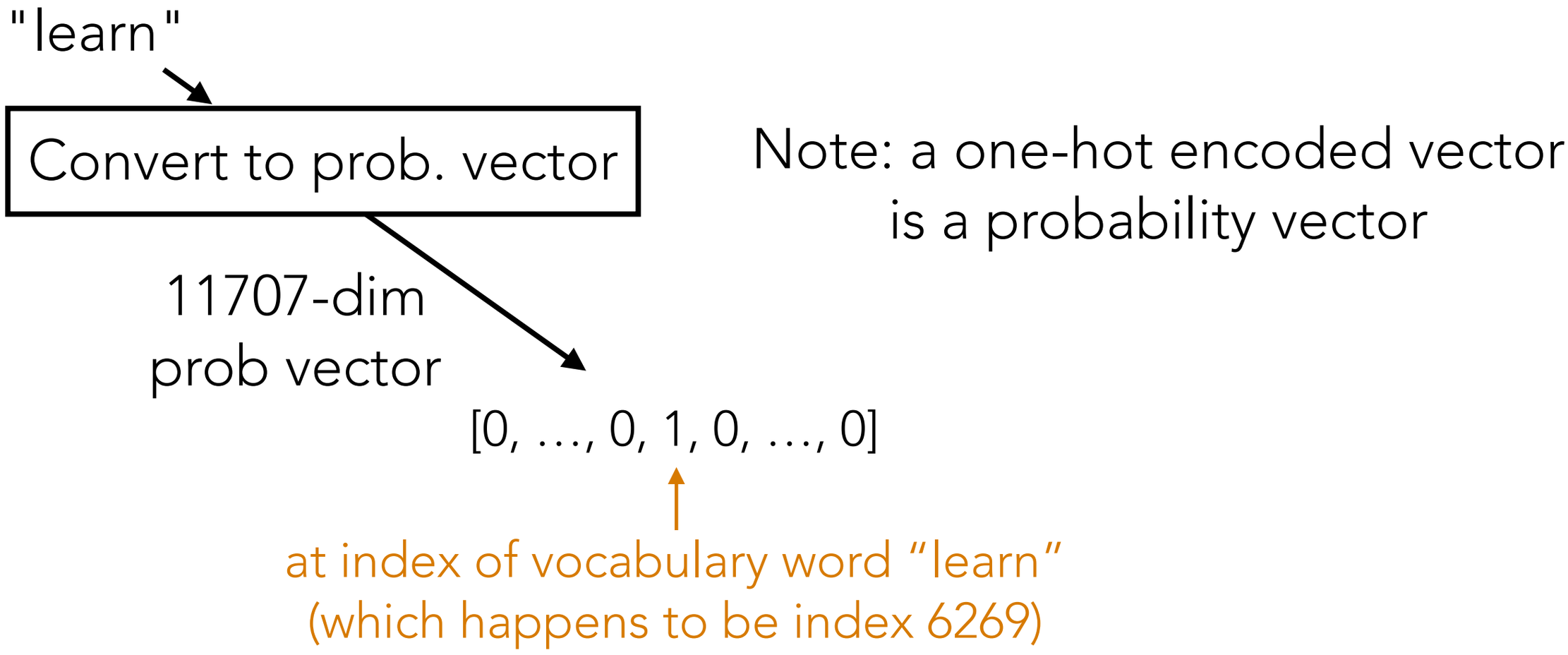
- We're clustering on the 20 Newsgroups dataset (preprocessed by lemmatizing every token)
- We filtered out some documents that are likely not in English
- We filtered out vocab words that showed up in too many or too few documents
    - Resulting 2D table of feature vectors: `filtered_tf`
    - Resulting 1D table of vocabulary words: `filtered_vocab`
- After filtering, text documents still varied wildly in length
    - Convert each row of `filtered_tf` into a probability vector
        - Resulting 2D table of probability vectors: `prob_vectors`


Raw text doc.
Each data point:
Python string
Convert to prob. vector
11707-dim
prob vector
Each data point:
1D array with 11707 entries
PCA
dim. reduction.
100-dim
PCA vector
Each data point:
1D array with 100 entries
t-SNE
dim. reduction.
2-dim
t-SNE vector
Each data point:
1D array with 2 entries



Does this 11707-dimensional space capture interesting semantic structure?

In general, a text doc can have lots of words, but let's consider when there's just a single word for a moment



Note: a one-hot encoded vector
is a probability vector

"learn"

Convert to prob. vector

11707-dim prob vector

[0, …, 0, 1, 0, …, 0]

at index of vocabulary word “learn” (which happens to be index 6269)

"car"

Convert to prob. vector

11707-dim prob vector

[0, …, 0, 1, 0, …, 0]

index 2134

"study"

Convert to prob. vector

11707-dim prob vector

[0, …, 0, 1, 0, …, 0]

index 10117

Intuitively, "learn" & "study" should be more semantically similar so we would like them to be closer to each other compared to "learn" & "car"

What happens when we compute these pairwise distances in the 11707-dim prob vector space?

Clustering on Text

Resuming the demo…

"learn"

Convert to prob. vector

11707-dim prob vector

[0, …, 0, 1, 0, …, 0]

at index of vocabulary word “learn” (which happens to be index 6269)

"car"

Convert to prob. vector

11707-dim prob vector

[0, …, 0, 1, 0, …, 0]

index 2134

"study"

Convert to prob. vector

11707-dim prob vector

[0, …, 0, 1, 0, …, 0]

index 10117

Intuitively, "learn" & "study" should be more semantically similar so we would like them to be closer to each other compared to "learn" & "car"

What happens when we compute these pairwise distances in the 11707-dim prob vector space?

"learn"

Convert to prob. vector

11707-dim
prob vector

PCA
dim. reduction.

100-dim
PCA vector

"car"

Convert to prob. vector

11707-dim
prob vector

PCA
dim. reduction.

100-dim
PCA vector

"study"

Convert to prob. vector

11707-dim
prob vector

PCA
dim. reduction.

100-dim
PCA vector

Let's see what happens when we compute pairwise distances using the 100-dim PCA vectors instead

Clustering on Text

Resuming the demo…