Carnegie Mellon University
Heinz College

Unstructured Data Analysis

Lecture 8: More clustering

George Chen

(Flashback) 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

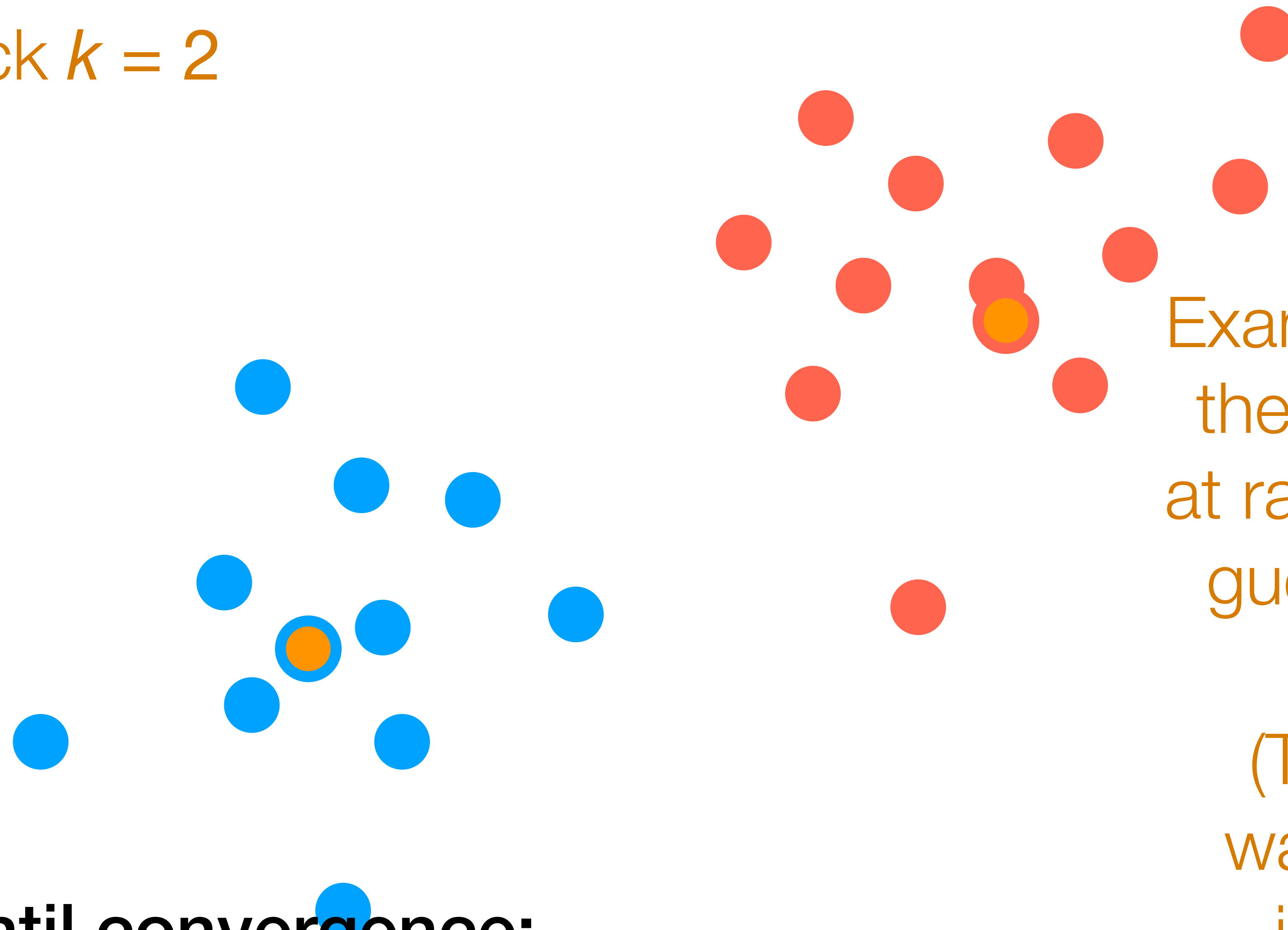
How do you turn this into a cluster assignment?

k-means

Step 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

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 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

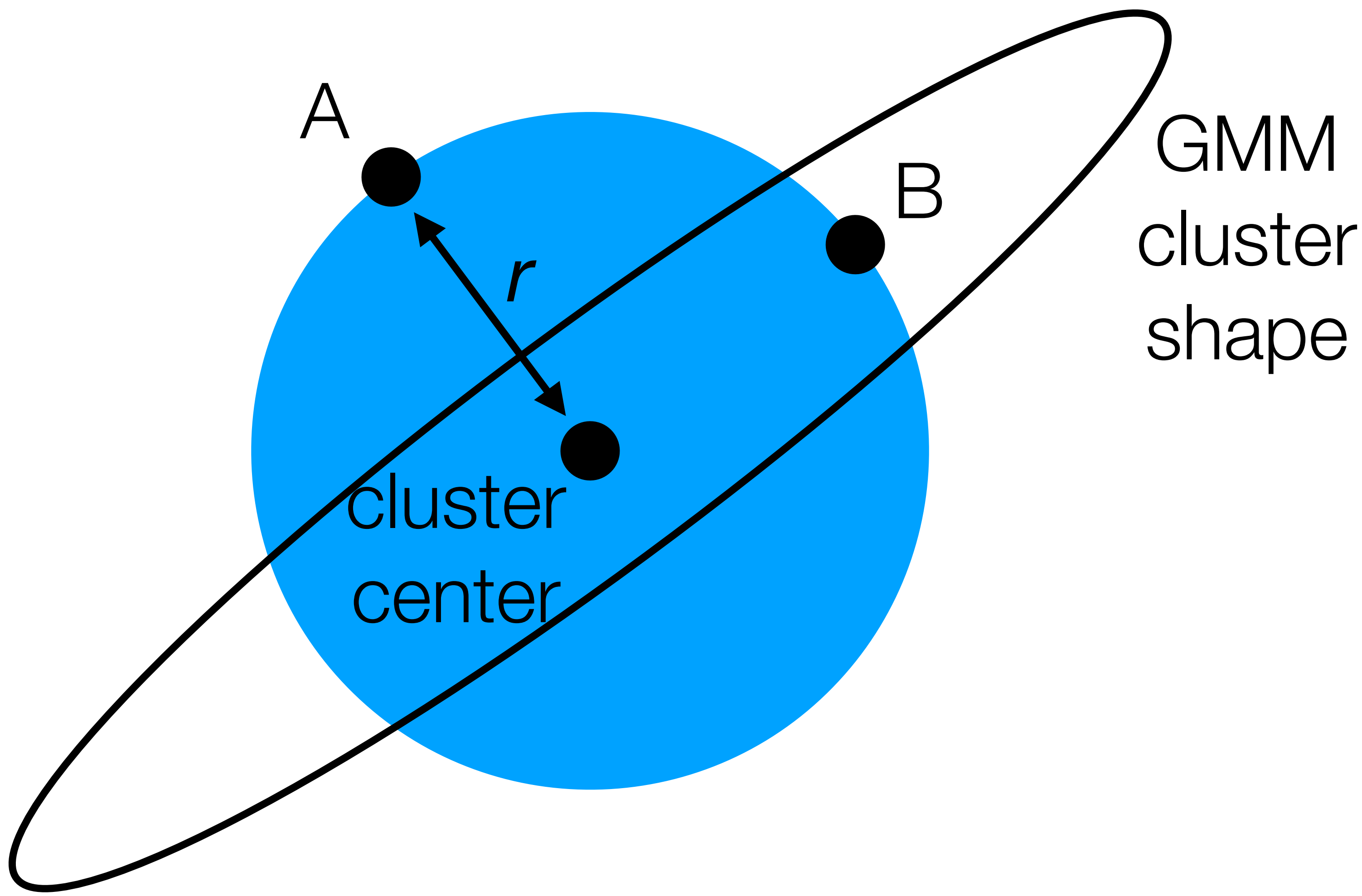
Step 3: Update **cluster probabilities, means, and covariances** carefully accounting for probabilities of each point belonging to each of the clusters

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)

(Rough Intuition) How Shape is Encoded by a GMM

For this ellipse-shaped Gaussian, point B is considered more similar to the cluster center than point A



k-means would think that point A and point B are the same distance from the cluster center since it does not keep track of cluster shape!!!

(Flashback) *k*-means

Step 0: Pick *k*

Step 1: Pick guesses for where cluster centers are

Repeat until convergence:

Euclidean distance, does not account for any shape information

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

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

Relating *k*-means to GMM's

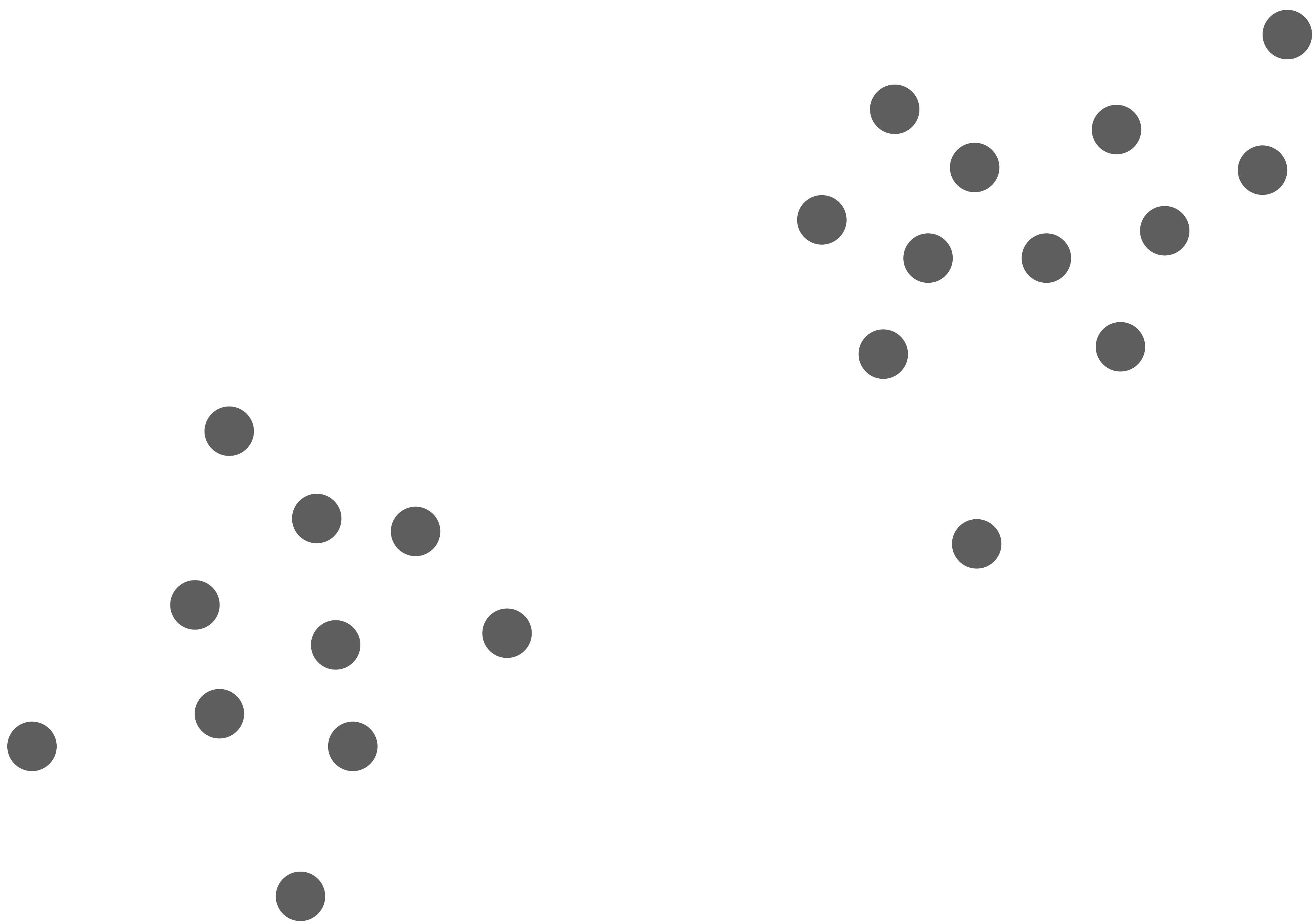
If the ellipses are all circles and have the same "skinniness" (e.g., in the 1D case it means they all have same std dev):

- *k*-means approximates the EM algorithm for GMM's (as there is no need to keep track of cluster shape)

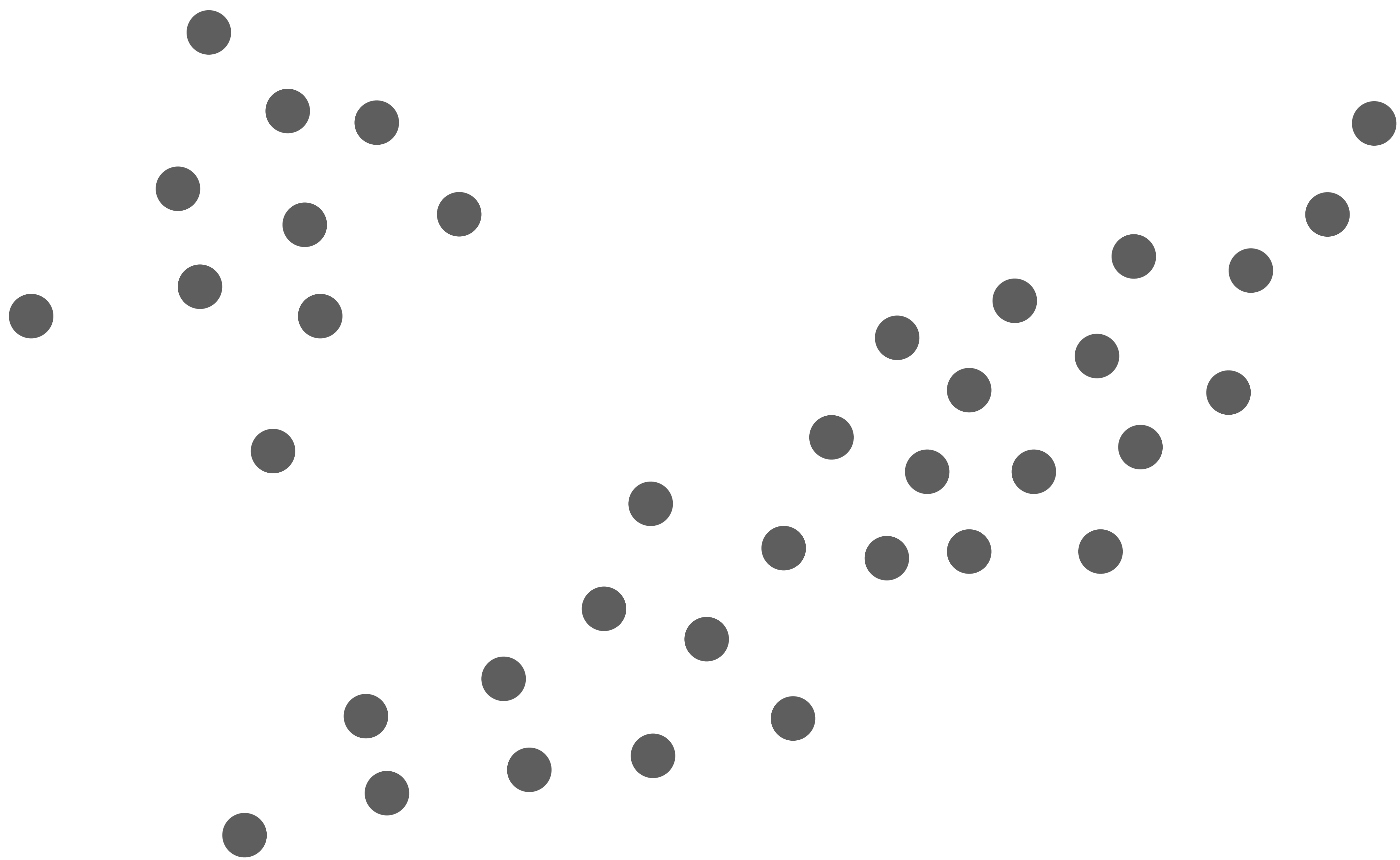
- *k*-means does a "hard" assignment of each point to a cluster, whereas the EM algorithm does a "soft" (probabilistic) assignment of each point to a cluster

Interpretation: *We now know a special case 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 of equal size"*

k-means should do well on this



But not on this



Learning a GMM

Demo

A Sketch of How to Interpret Clusters

Demo

Automatically Choosing *k*

For k = 2, 3, … up to some user-specified max value:

Fit model using k

Compute a score for the model

But what score function should we use?

Use whichever k has the best score

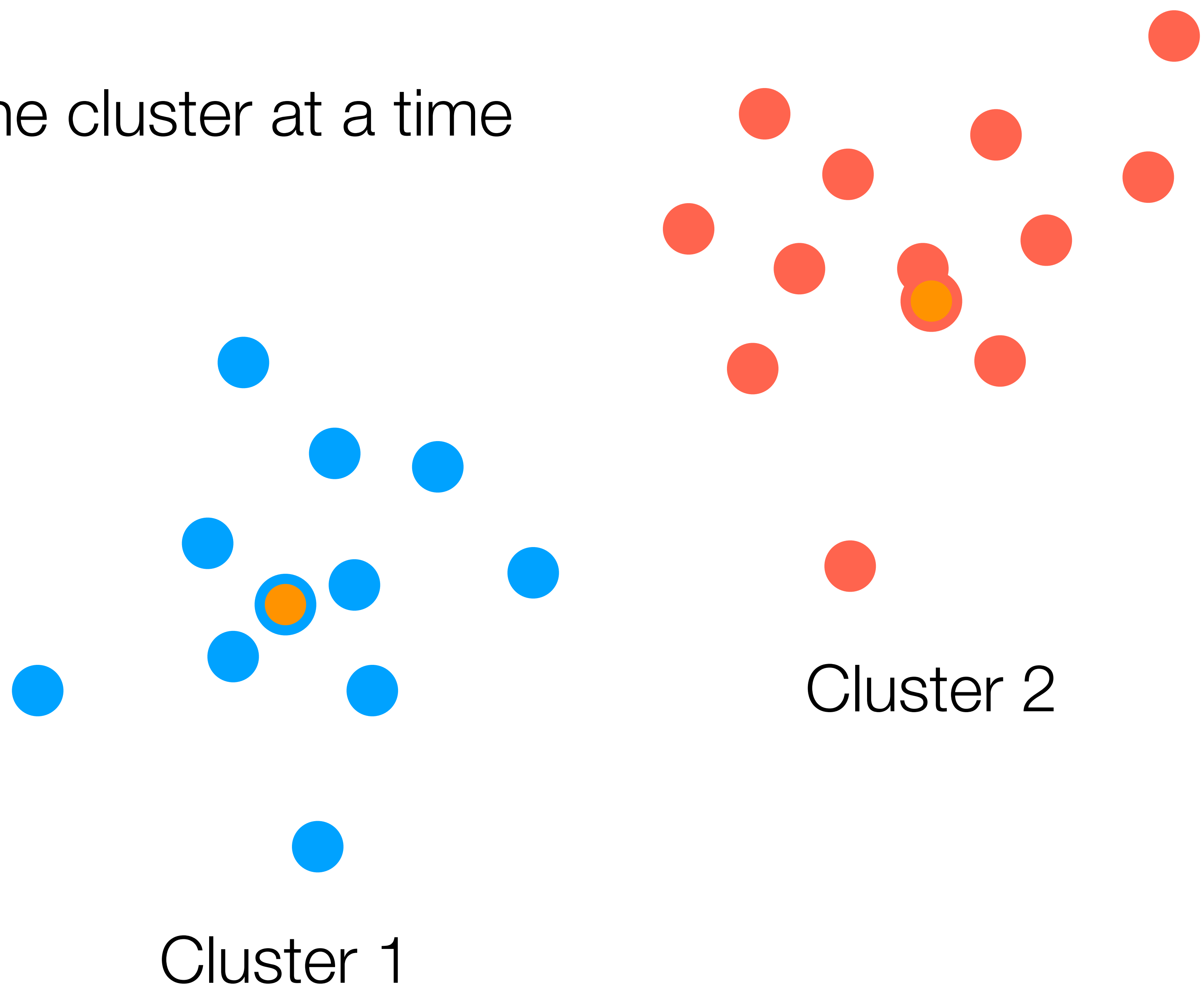
No single way of choosing k is the "best" way

Here’s an example of a score function you don’t want to use

But hey it’s worth a shot

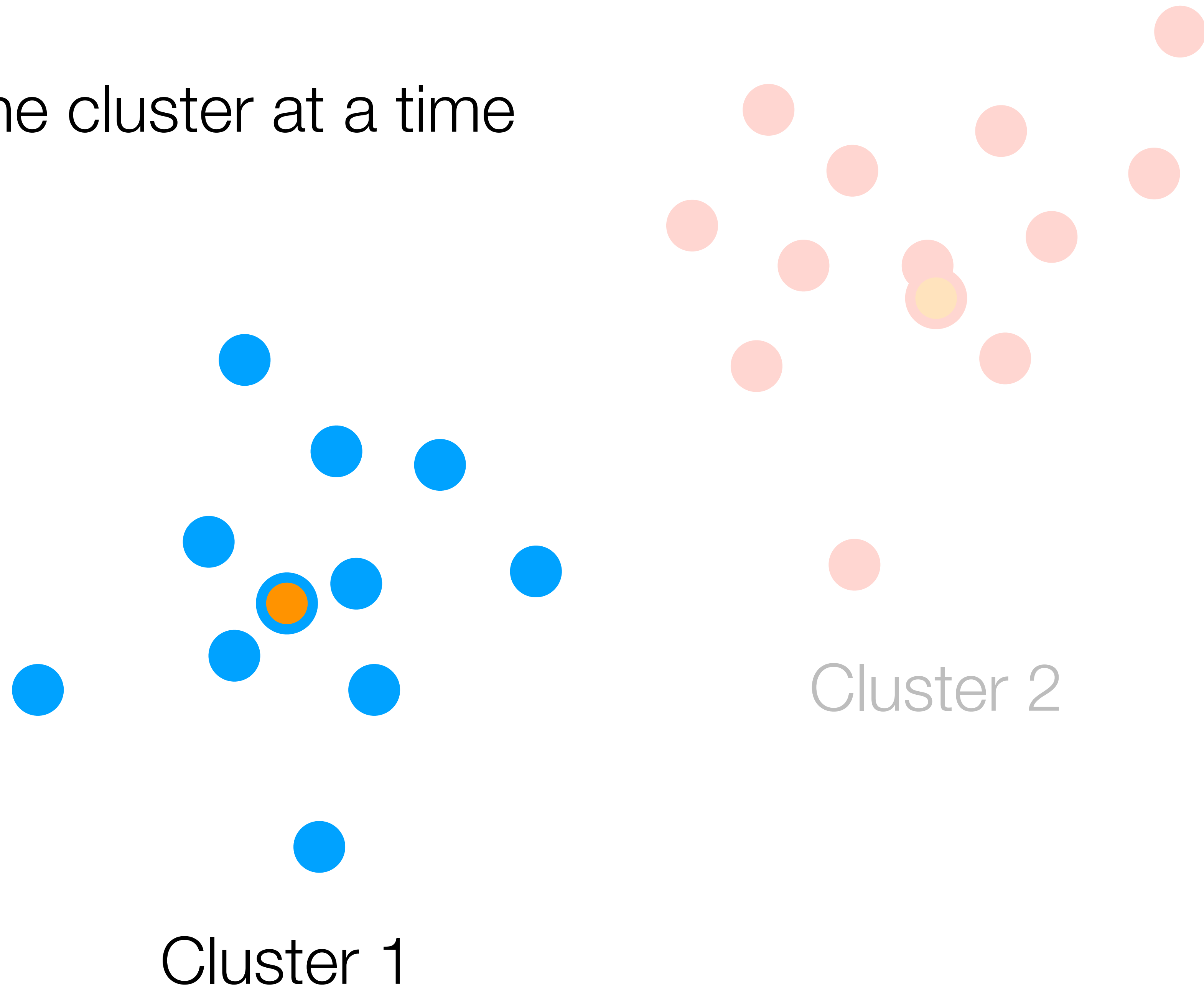
Residual Sum of Squares

Look at one cluster at a time



Residual Sum of Squares

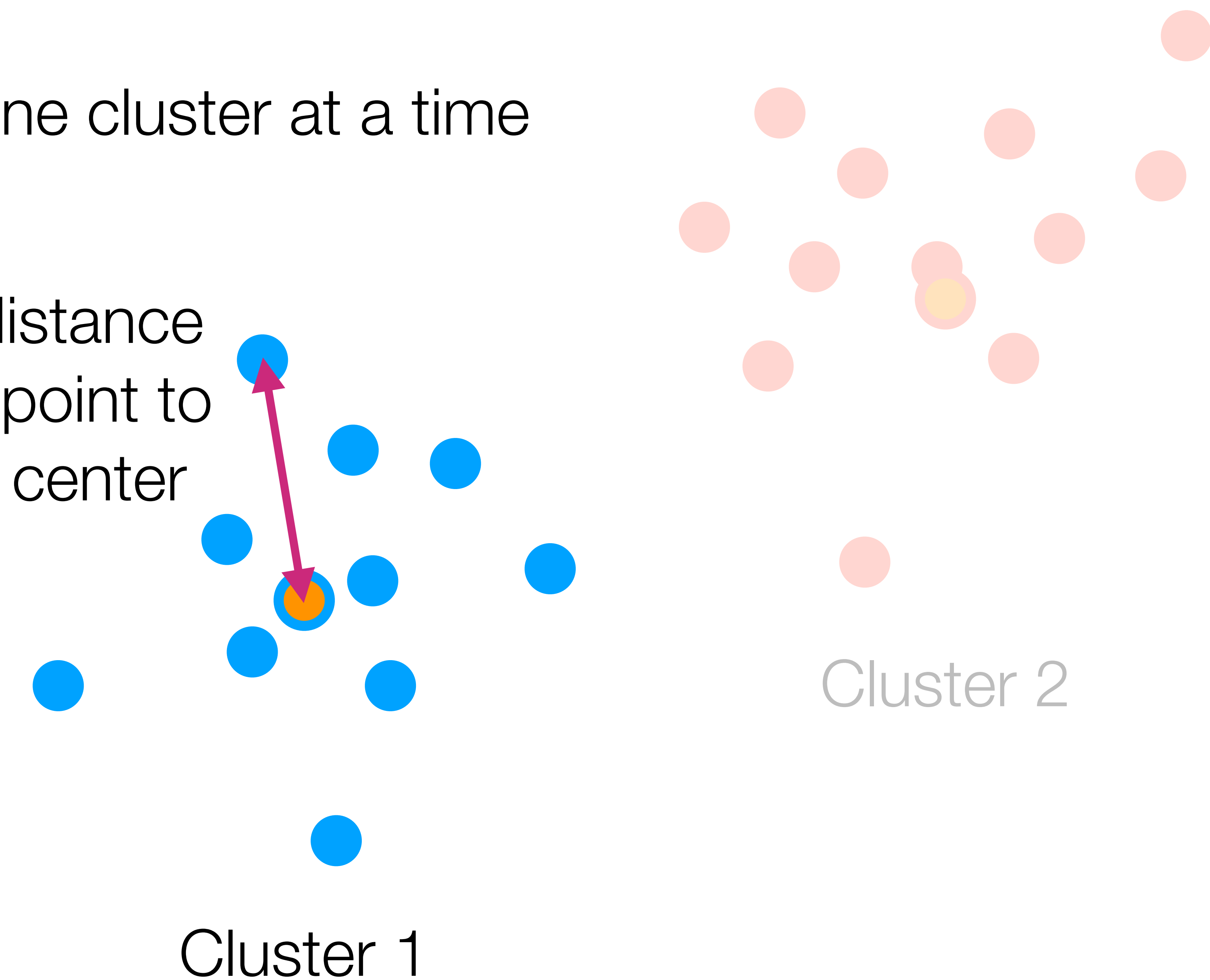
Look at one cluster at a time



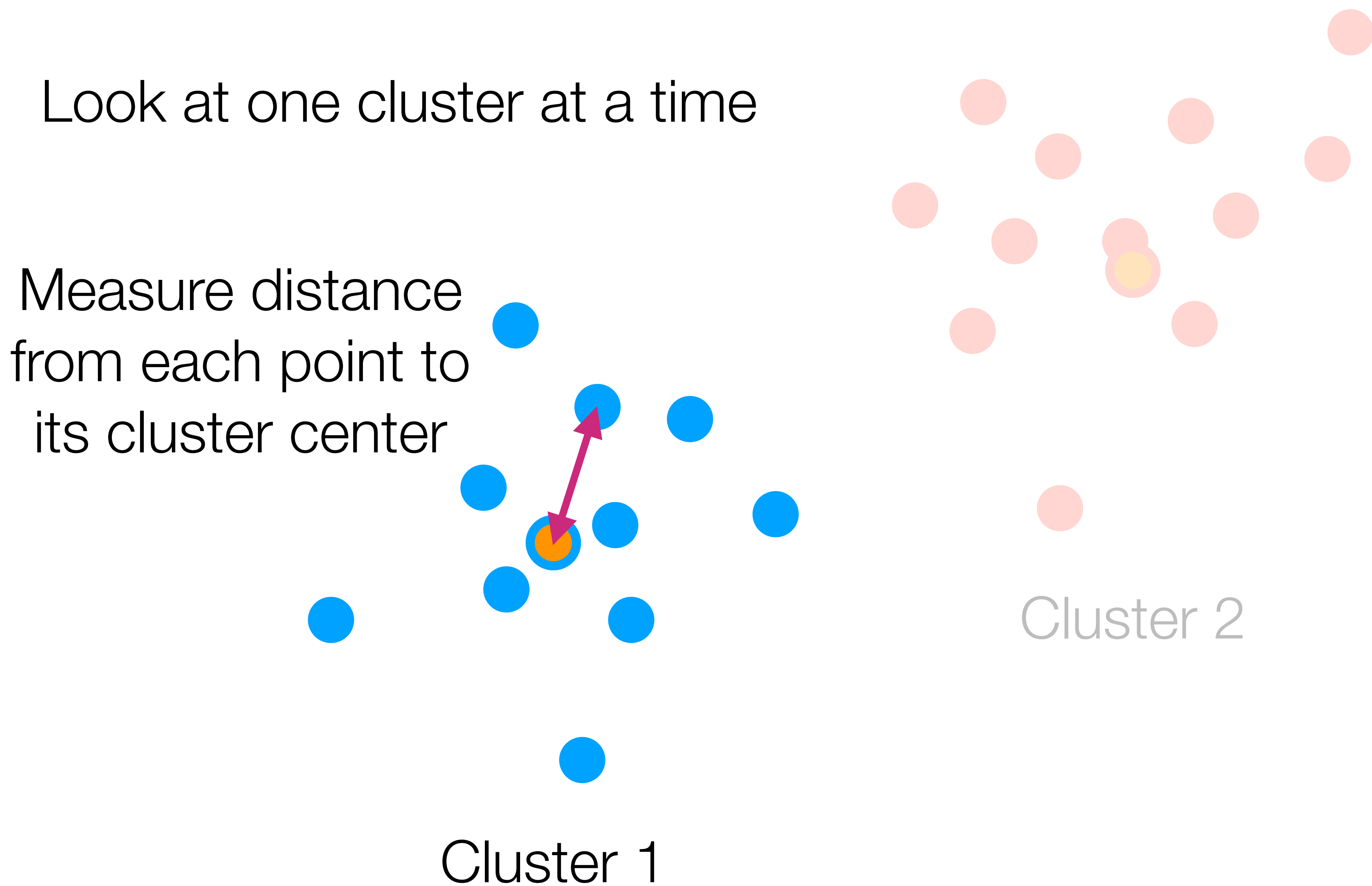
Residual Sum of Squares

Look at one cluster at a time

Measure distance from each point to its cluster center



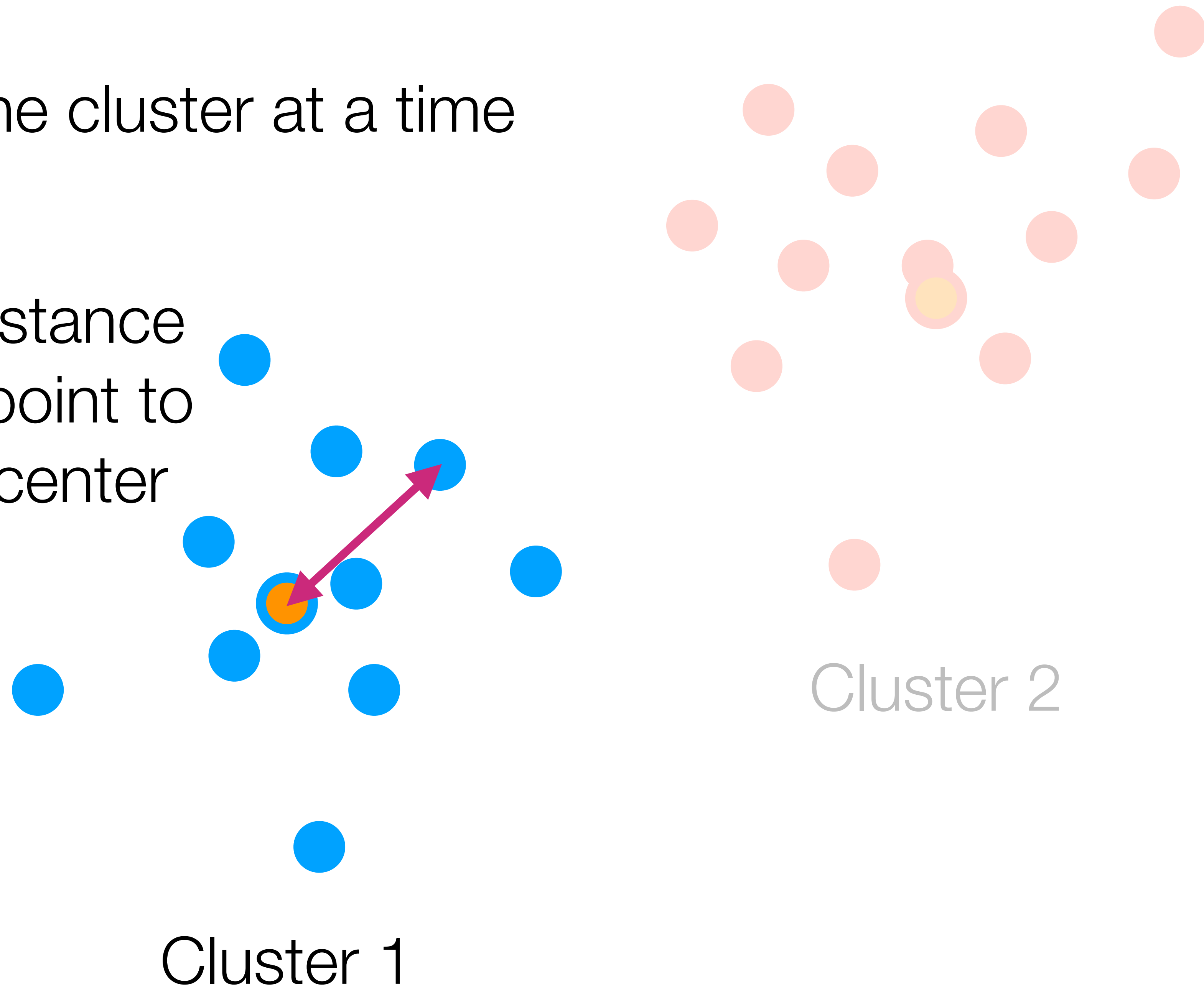
Residual Sum of Squares



Residual Sum of Squares

Look at one cluster at a time

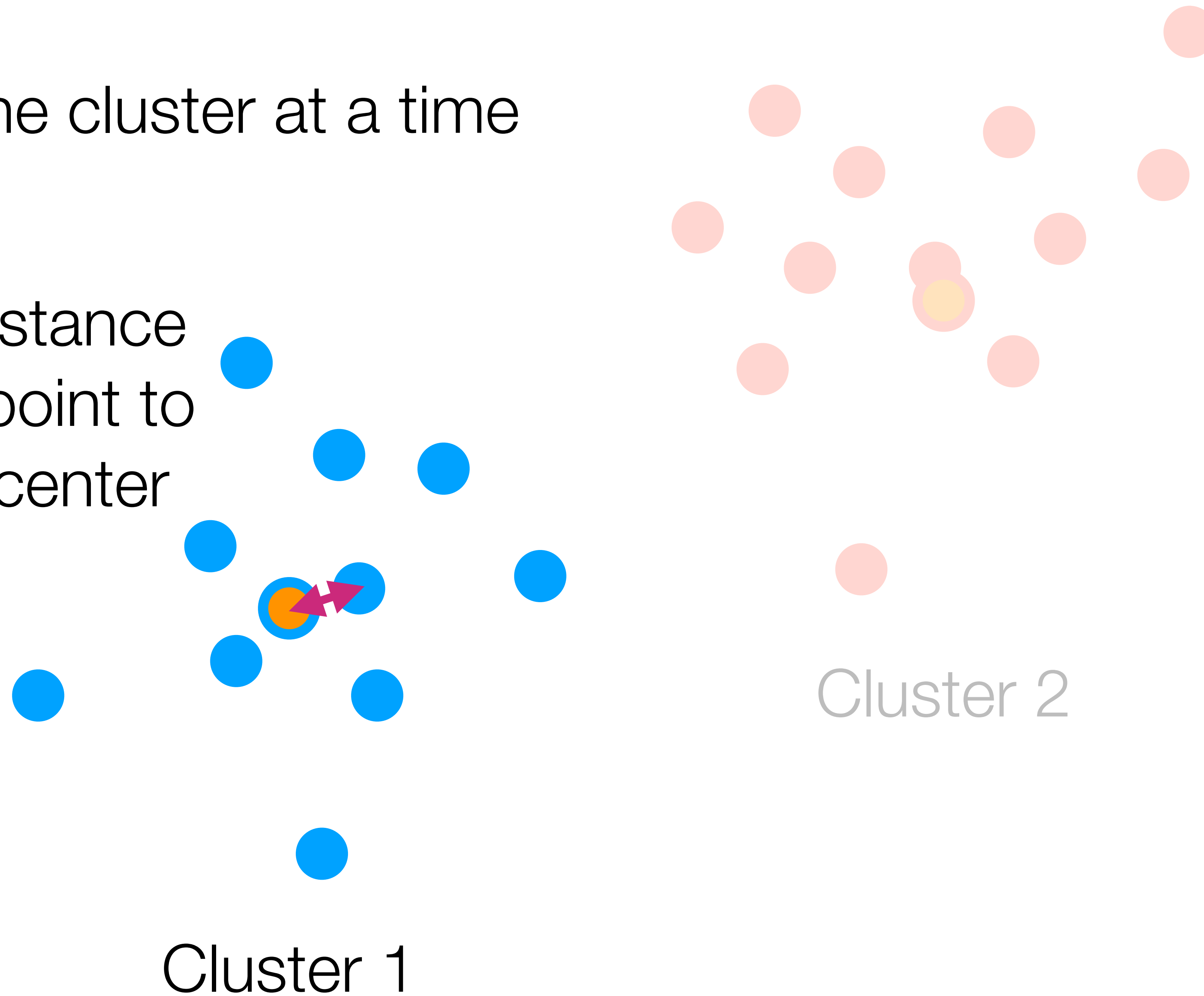
Measure distance from each point to its cluster center



Residual Sum of Squares

Look at one cluster at a time

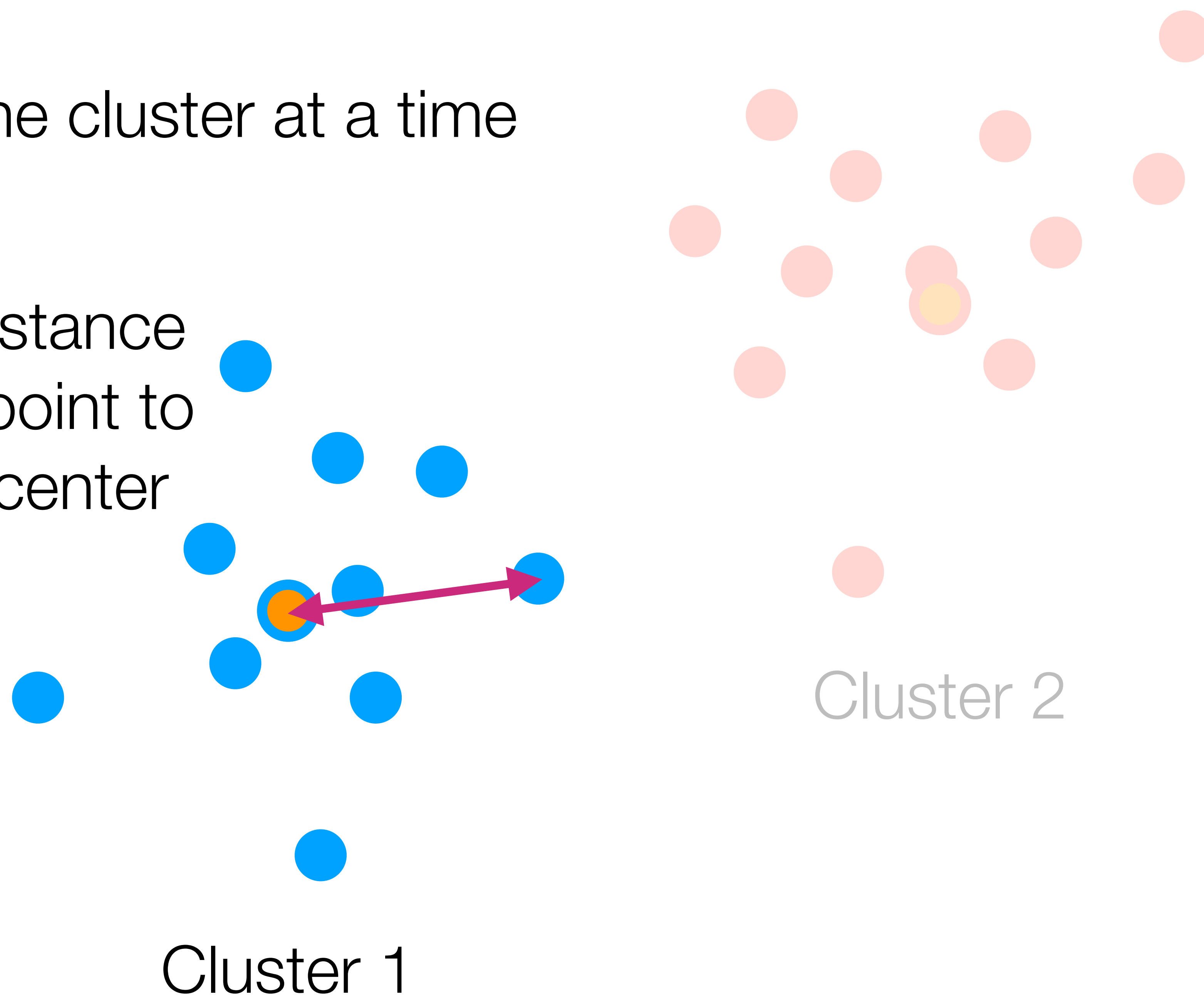
Measure distance
from each point to
its cluster center



Residual Sum of Squares

Look at one cluster at a time

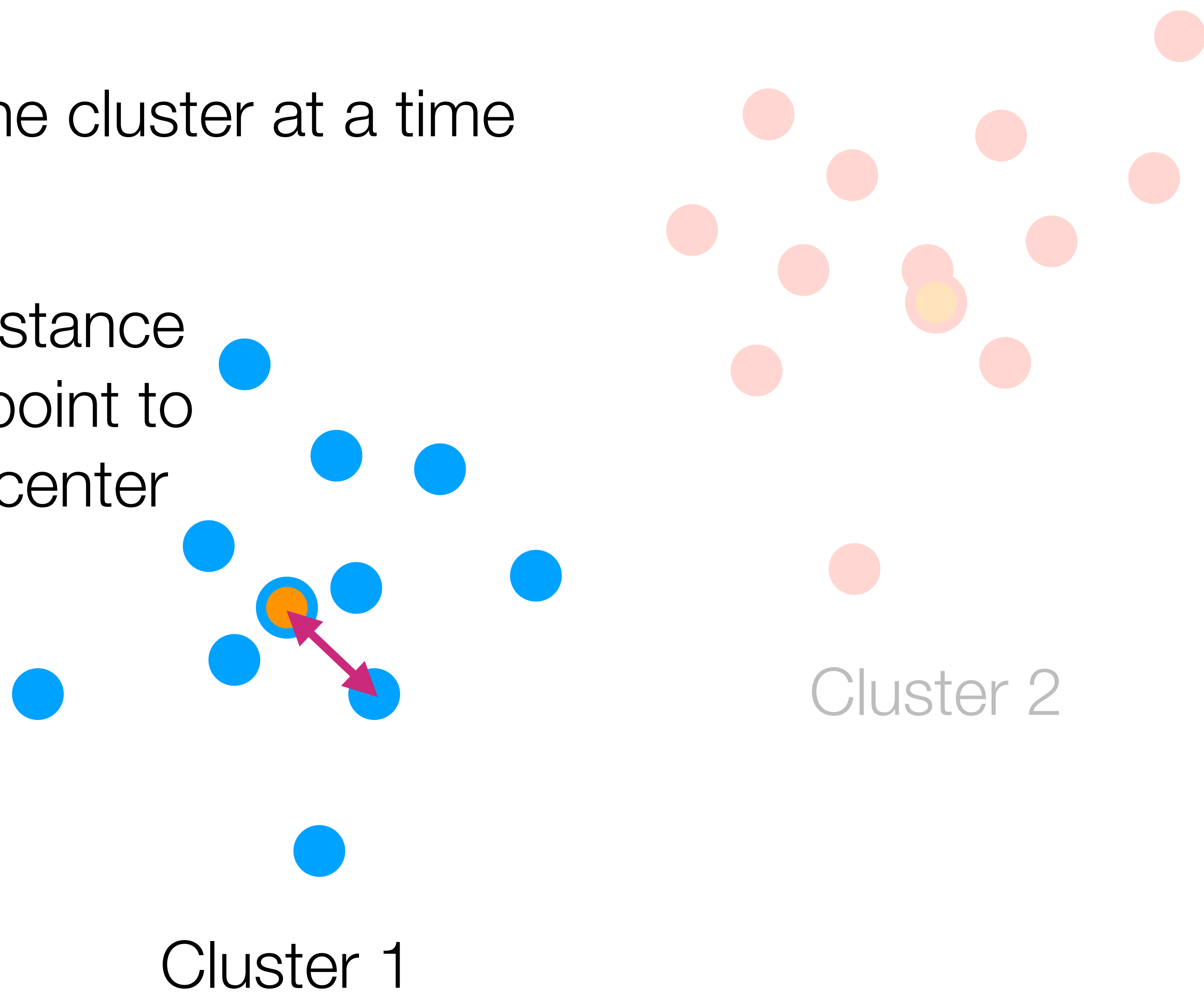
Measure distance
from each point to
its cluster center



Residual Sum of Squares

Look at one cluster at a time

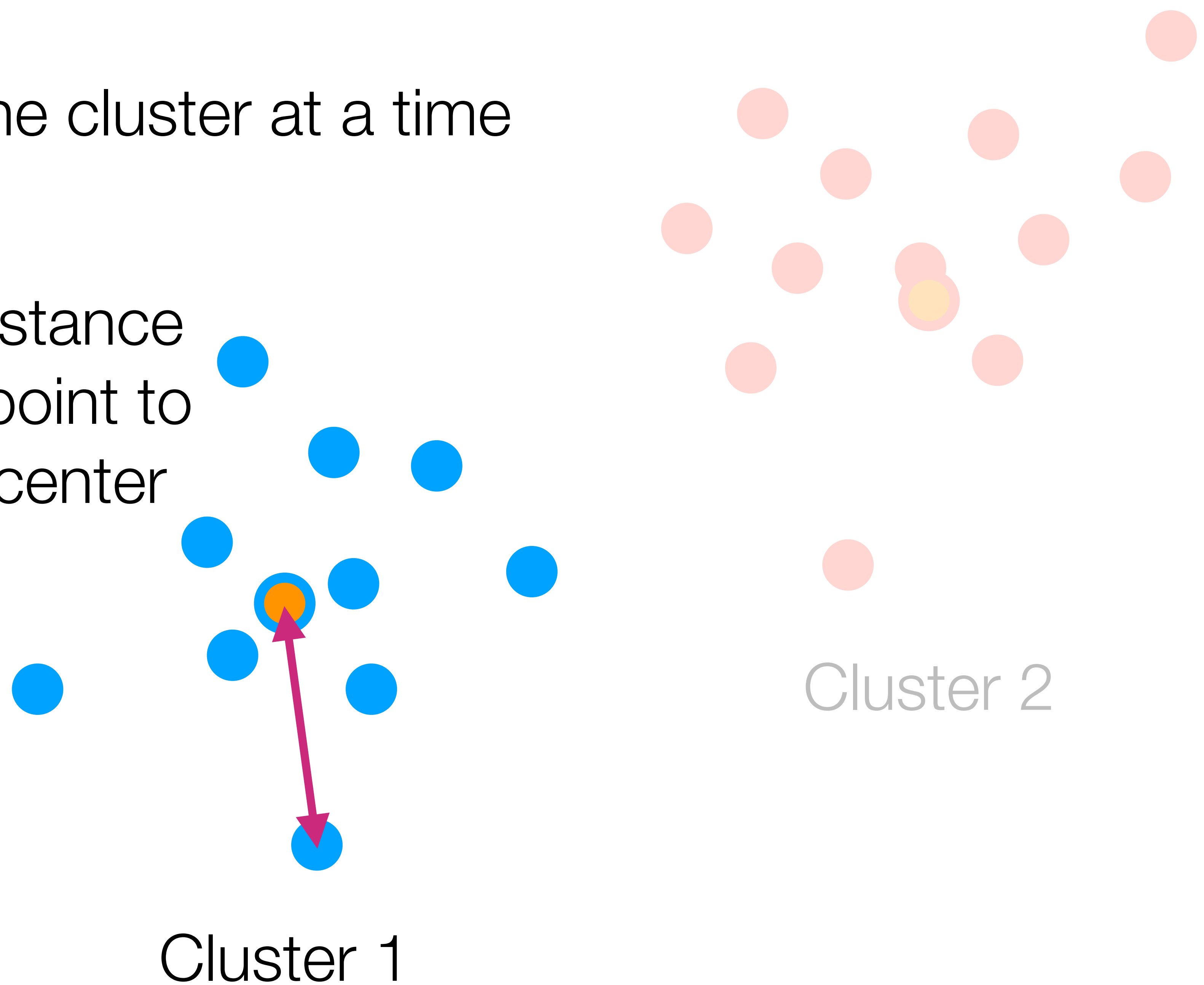
Measure distance from each point to its cluster center



Residual Sum of Squares

Look at one cluster at a time

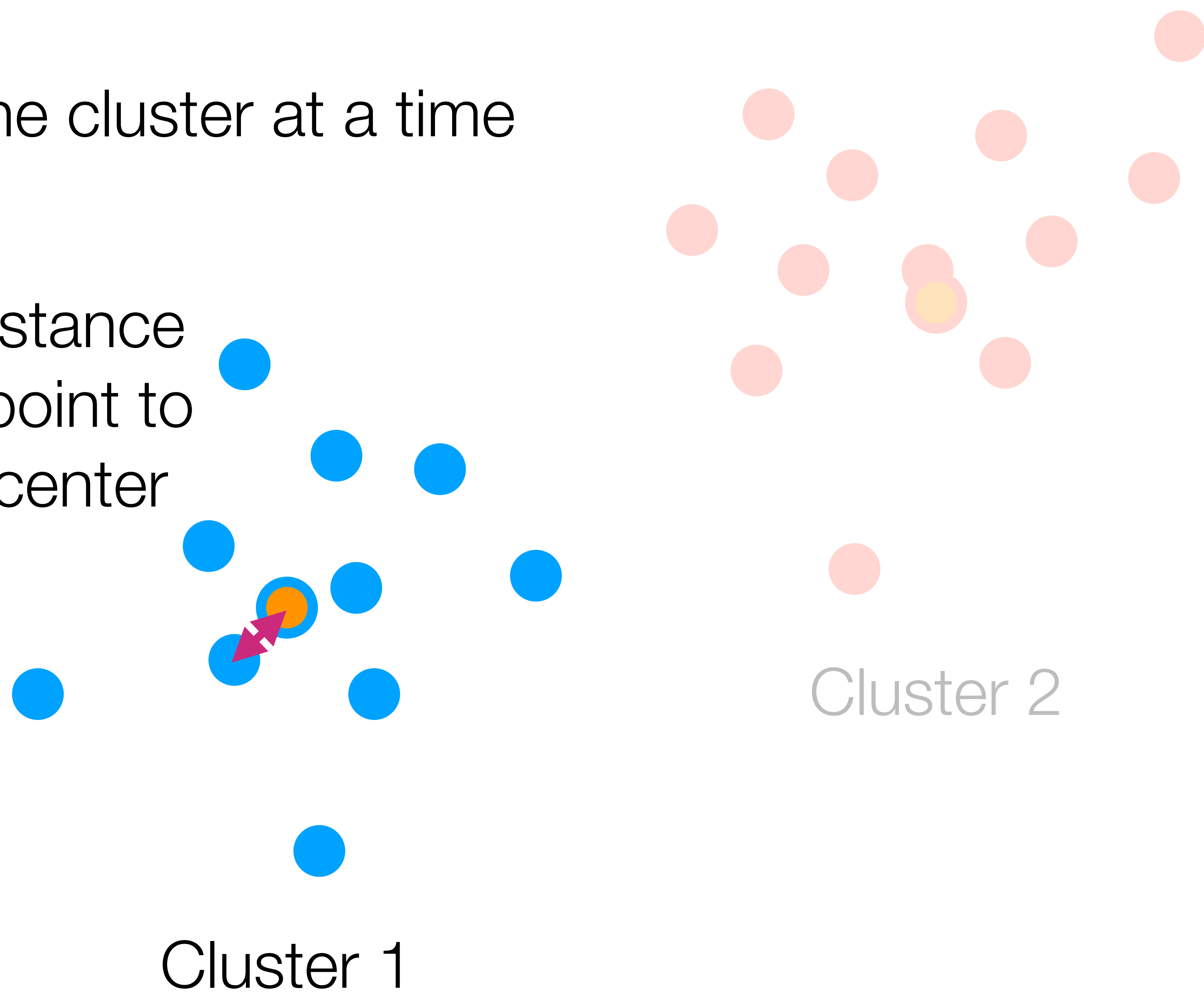
Measure distance from each point to its cluster center



Residual Sum of Squares

Look at one cluster at a time

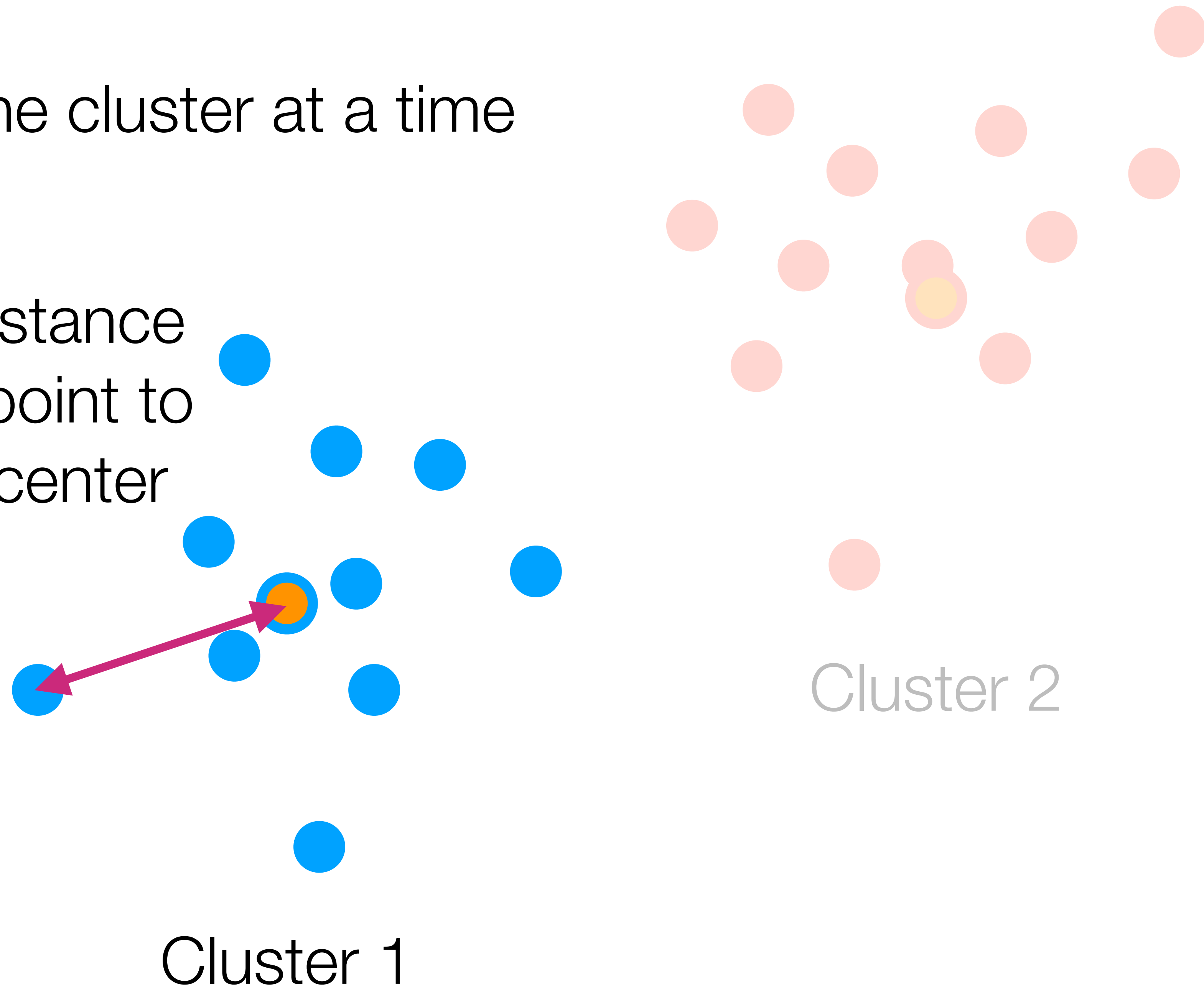
Measure distance from each point to its cluster center



Residual Sum of Squares

Look at one cluster at a time

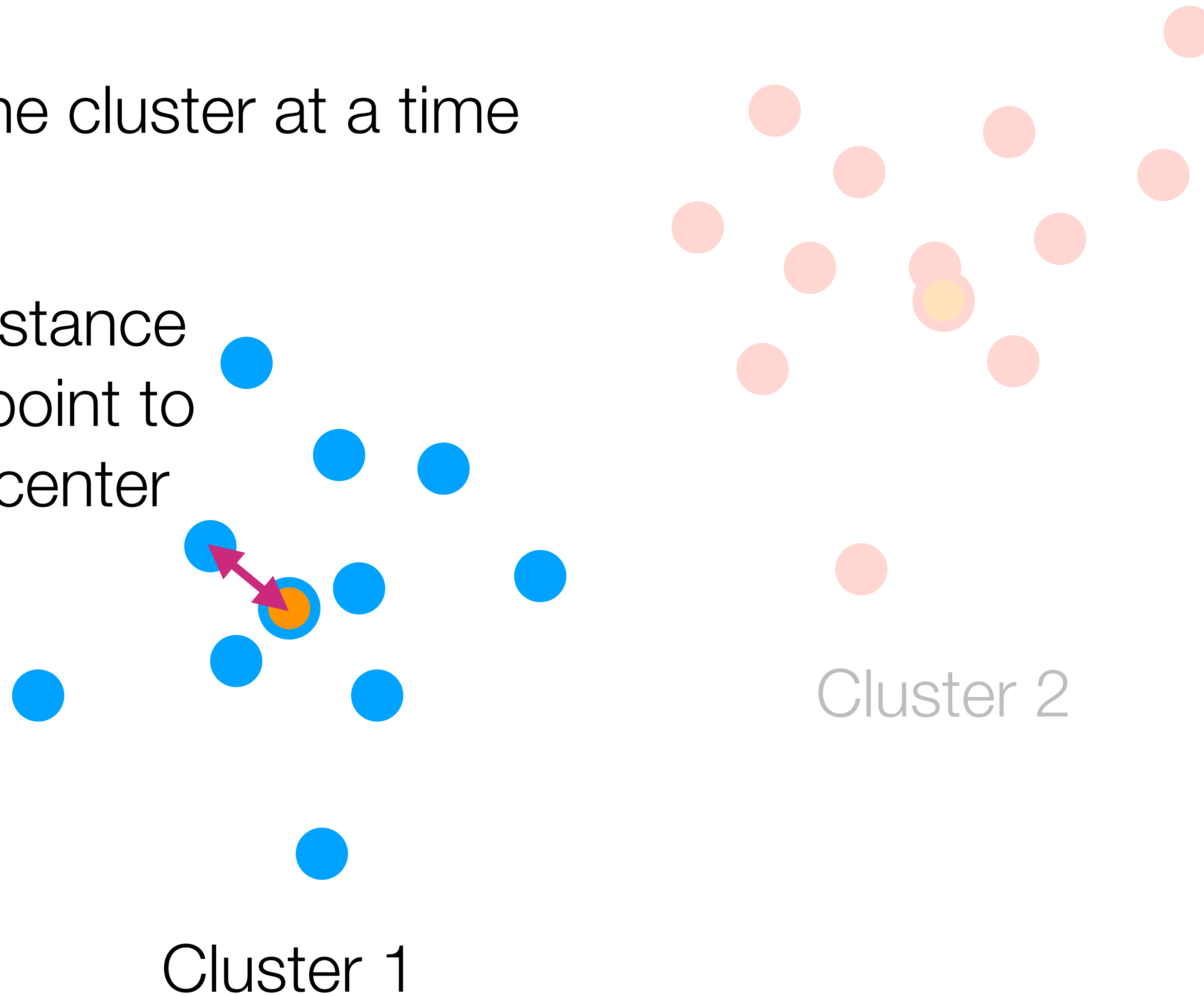
Measure distance from each point to its cluster center



Residual Sum of Squares

Look at one cluster at a time

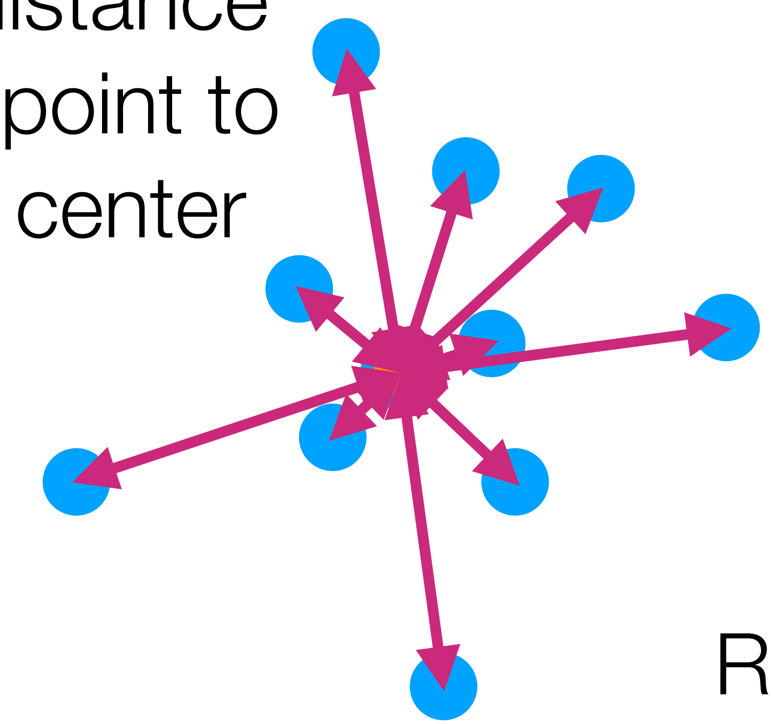
Measure distance from each point to its cluster center



Residual Sum of Squares

Look at one cluster at a time

Measure distance from each point to its cluster center



Cluster 1

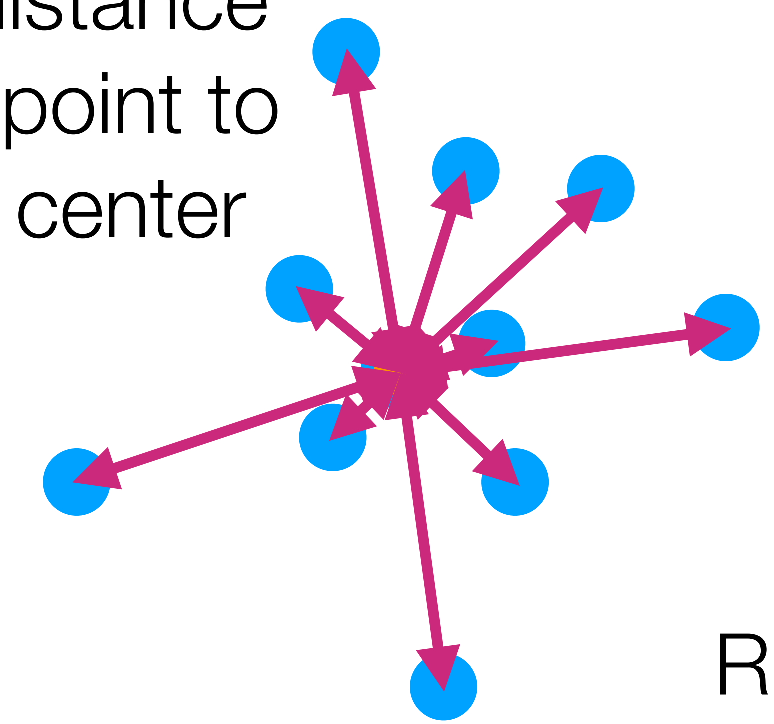
Cluster 2

Residual sum of squares for cluster 1: sum of *squared* purple lengths

Residual Sum of Squares

Look at one cluster at a time

Measure distance from each point to its cluster center



Cluster 1

Cluster 2

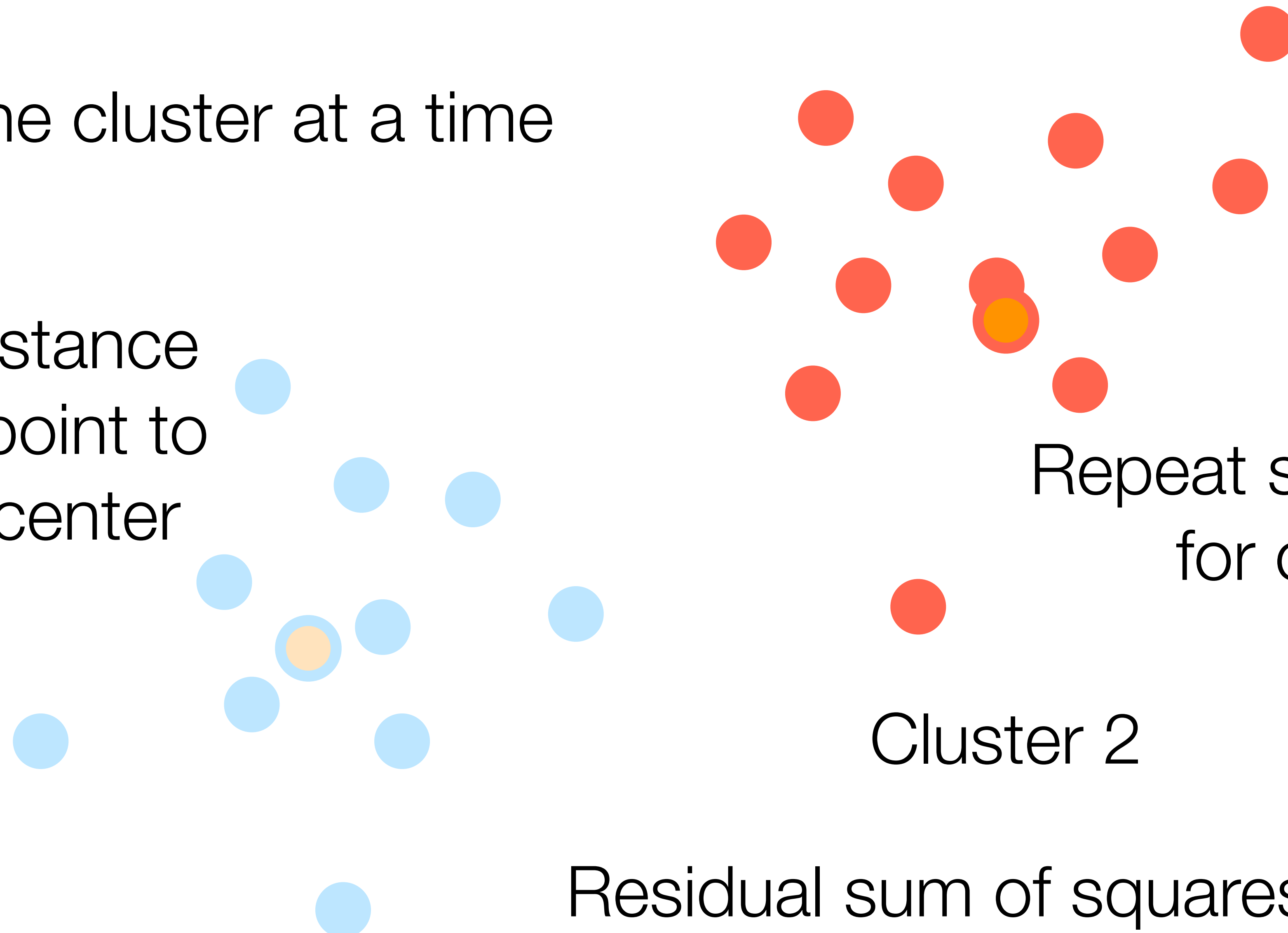
Residual sum of squares for cluster 1:

$$\mathrm{RSS}_1 = \sum_{x \in \text{cluster 1}} \|x - \mu_1\|^2$$

Residual Sum of Squares

Look at one cluster at a time

Measure distance from each point to its cluster center



Repeat similar calculation for other cluster

Cluster 2

Cluster 1

Residual sum of squares for cluster 2:

$$\mathrm{RSS}_2 = \sum_{x \in \text{cluster 2}} \left\| x - \mu_2 \right\|^2$$

Residual Sum of Squares

$$\text{RSS} = \text{RSS}_1 + \text{RSS}_2 = \sum_{x \in \text{cluster } 1} \|x - \mu_1\|^2 + \sum_{x \in \text{cluster } 2} \|x - \mu_2\|^2$$

In general if there are *k* clusters:

$$\text{RSS} = \sum_{g=1}^{k} \text{RSS}_g = \sum_{g=1}^{k} \sum_{x \in \text{cluster } g} \|x - \mu_g\|^2$$

Remark: *k*-means *tries* to minimize RSS
(it does so *approximately,* with no guarantee of optimality)

RSS only really makes sense for clusters that look like circles

Why is minimizing RSS a bad way to choose *k*?

What happens when *k* is equal to the number of data points?

A Good Way to Choose *k*

RSS measures *within-cluster variation*

$$W = \text{RSS} = \sum_{g=1}^{k} \text{RSS}_g = \sum_{g=1}^{k} \sum_{x \in \text{cluster } g} \|x - \mu_g\|^2$$

Want to also measure *between-cluster variation*

$$B = \sum_{g=1}^{k} (\text{\# points in cluster } g) \|\mu_g - \mu\|^2$$

mean of *all* points

Called the **CH index**
[Calinski and Harabasz 1974]

A good score function to use for choosing *k*:

$$\text{CH}(k) = \frac{B \cdot (n - k)}{W \cdot (k - 1)}$$

n = total # points

Pick *k* with highest **CH(*k*)**

(Choose *k* among 2, 3, … up to pre-specified max)

Automatically Choosing *k*

Demo

More on Automatic Selection of *k*

Dirichlet Process Gaussian Mixture Model (DP-GMM):

- Number of clusters is effectively random, and *can grow with the amount of data you have!*

- While you don't have to choose *k*, you have to choose a different parameter which says how large clusters are

DP-GMM High-Level Idea

Cluster 1	Cluster 2	Cluster 3	
Probability of generating a point from cluster 1 = π_1	π_2	π_3	…
Gaussian mean = μ_1	μ_2	μ_3	It goes on forever!
Gaussian covariance = Σ_1	Σ_2	Σ_3	

There is a parameter that controls how these π values roughly decay

There are an infinite number of parameters

(Rough idea) How to generate points from this DP-GMM:

1. Flip biased ∞-sided coin (the sides have probabilities π_1, π_2, π_3, …)
2. Let Z be the side that we got (it is a positive integer)
3. Sample 1 point from Gaussian mean μ_Z, covariance Σ_Z

Remark: For any given dataset, when learning the DP-GMM, there aren't going to be an infinite number of clusters found

More on Automatic Selection of *k*

Dirichlet Process Gaussian Mixture Model (DP-GMM):

- Number of clusters is effectively random, and *can grow with the amount of data you have!*

- While you don't have to choose *k*, you have to choose a different parameter which says how large clusters are

- An example of a *Bayesian nonparametric model* (roughly: a generative model with an *infinite number of parameters*, where the *parameters are random*)

Learning a DP-GMM

Common approach: finite approximation where you specify some maximum number of clusters (the algorithm will find up to that many clusters)

- Algorithm is somewhat similar to *k*-means/EM for GMMs
- Algorithm output: very similar to regular GMM fitting

Learning a DP-GMM

Demo

k-means approximates (a special case of) learning GMM's.

What approximates learning DP-GMMs?

This next algorithm will give you a sense of how we get around specifying the number of clusters directly