Carnegie Mellon University
Heinz College

94-775/95-865 Lecture 6: Clustering Part II

George Chen

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

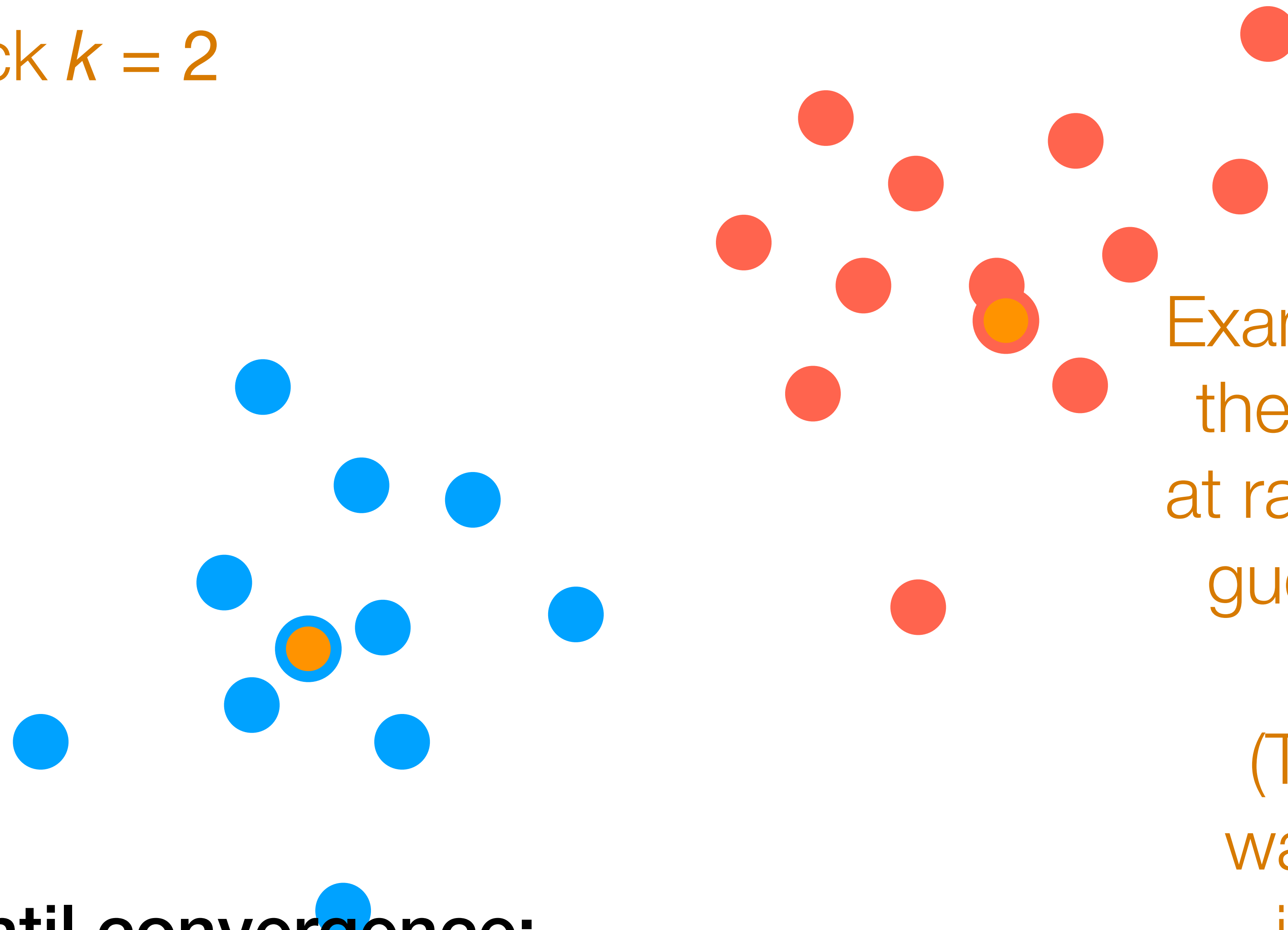
How do you turn this into a cluster assignment?

k-means

Step 0: Pick *k*

We'll pick *k* = 2

Step 1: Pick 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 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

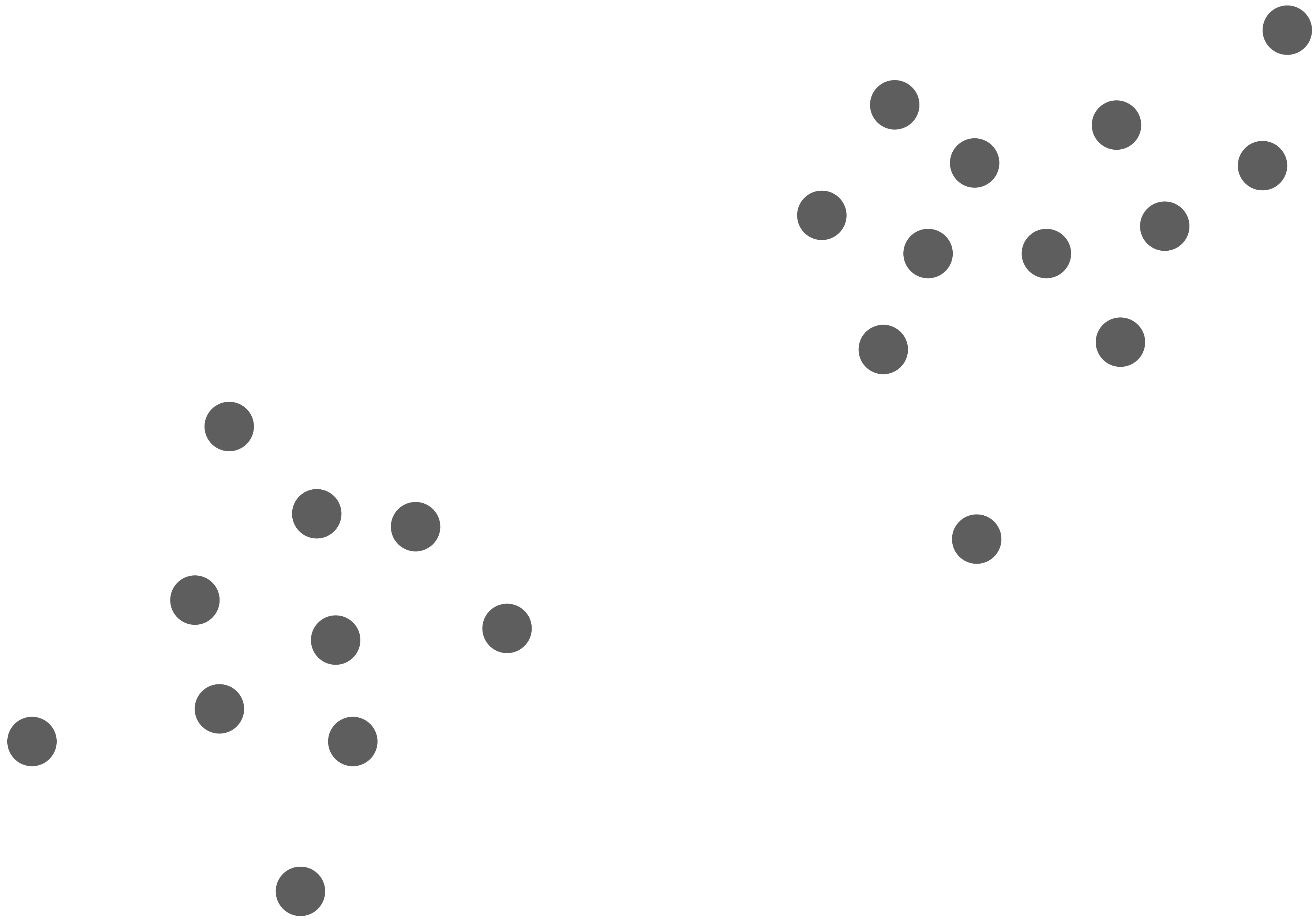
If the ellipses are all circles and have the same "skinniness" (e.g., in the 1D case it means they all have same std dev):

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

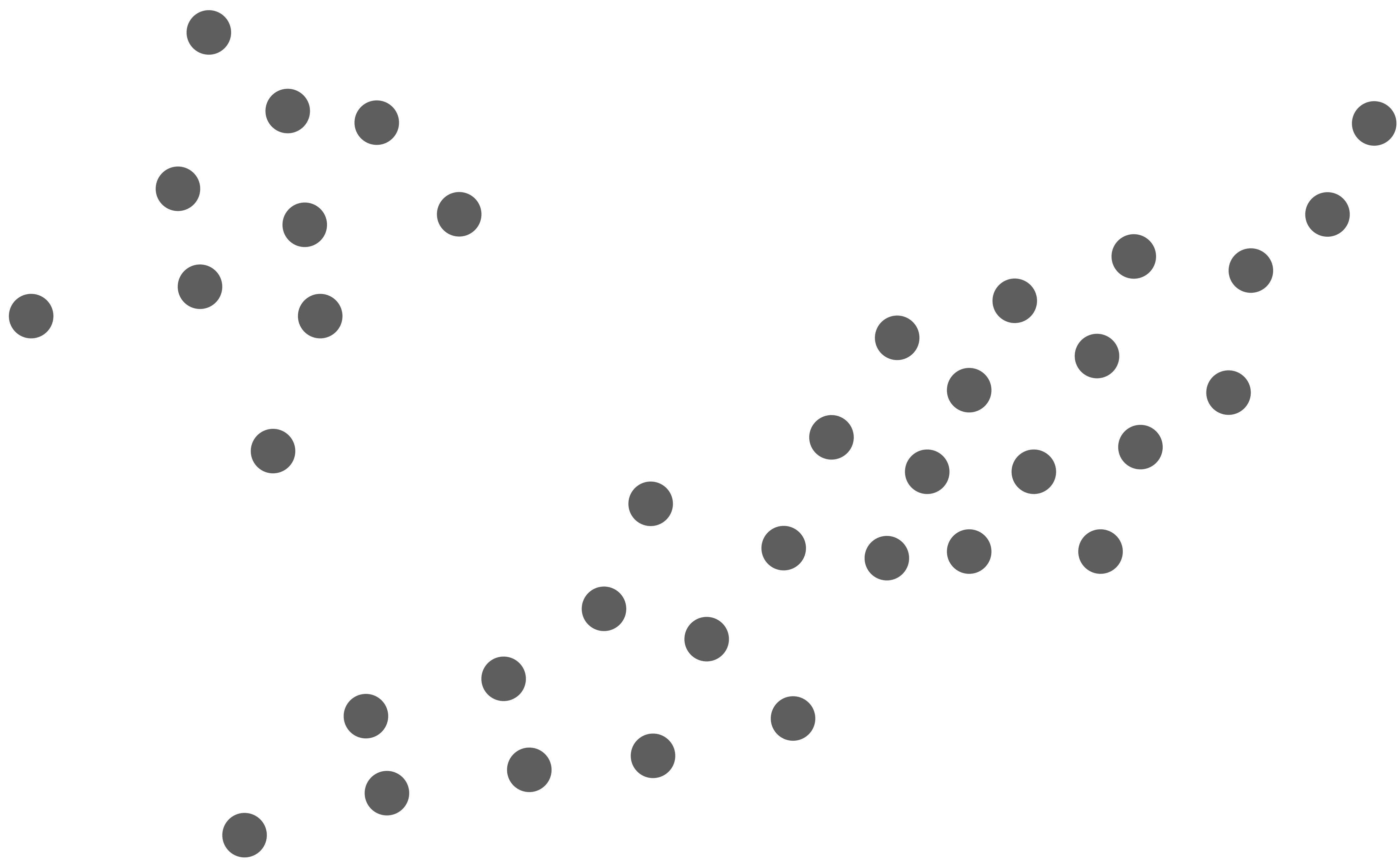
- Notice that *k*-means does a "hard" assignment of each point to a cluster, whereas the EM algorithm does a "soft" (probabilistic) assignment of each point to a cluster

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 and Interpreting a GMM

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

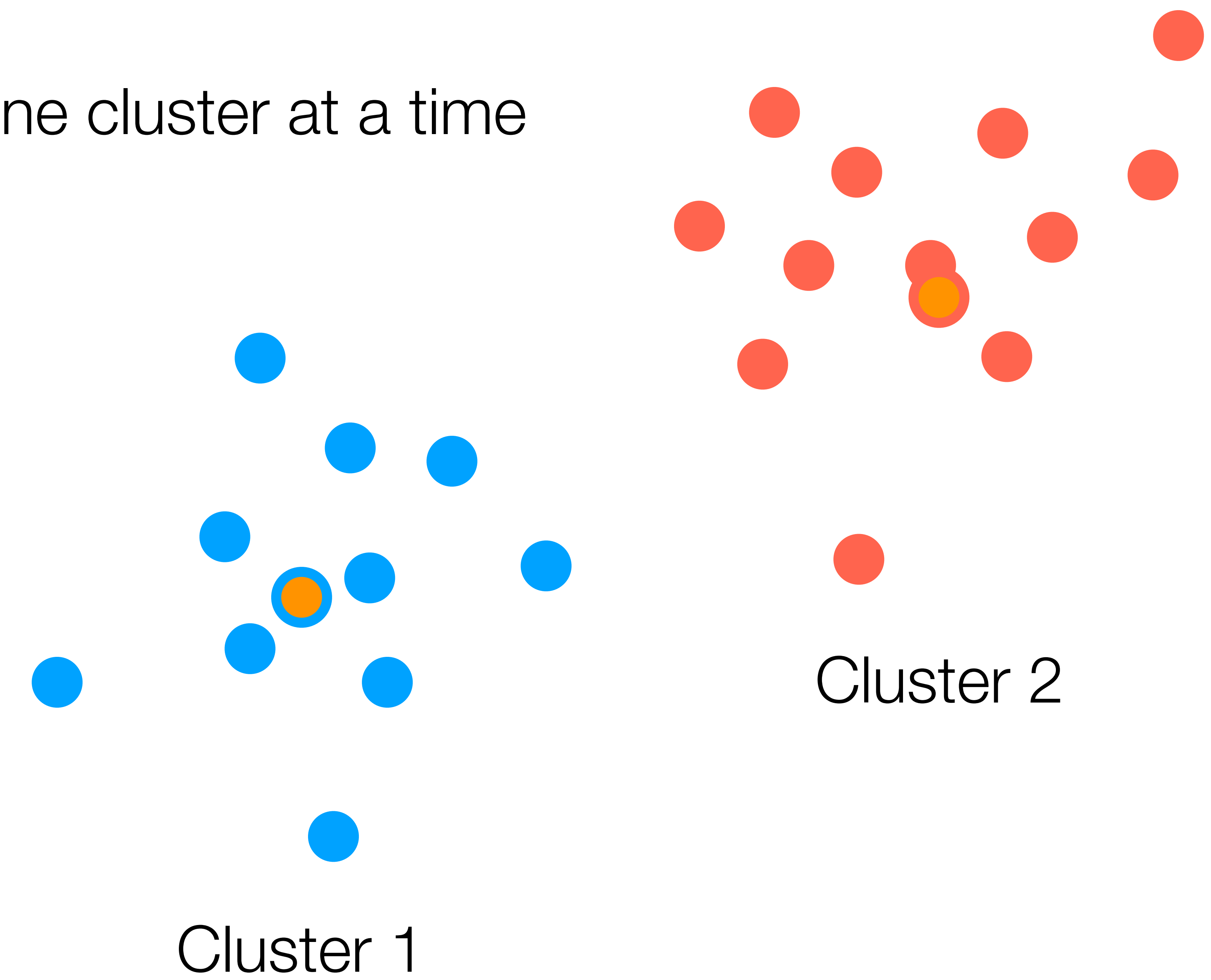
No single way of choosing k is the “best” way

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

But hey it's worth a shot

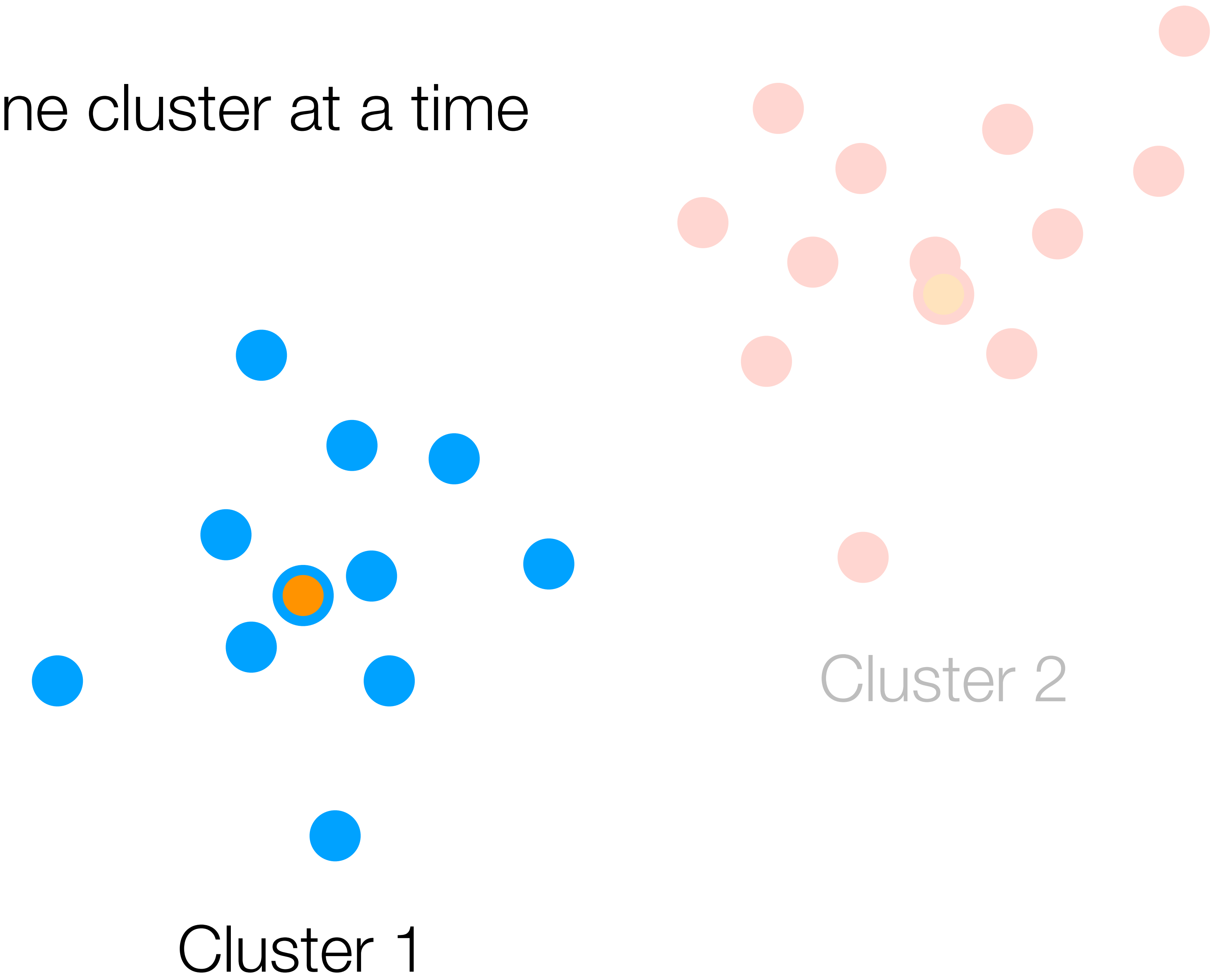
Residual Sum of Squares

Look at one cluster at a time

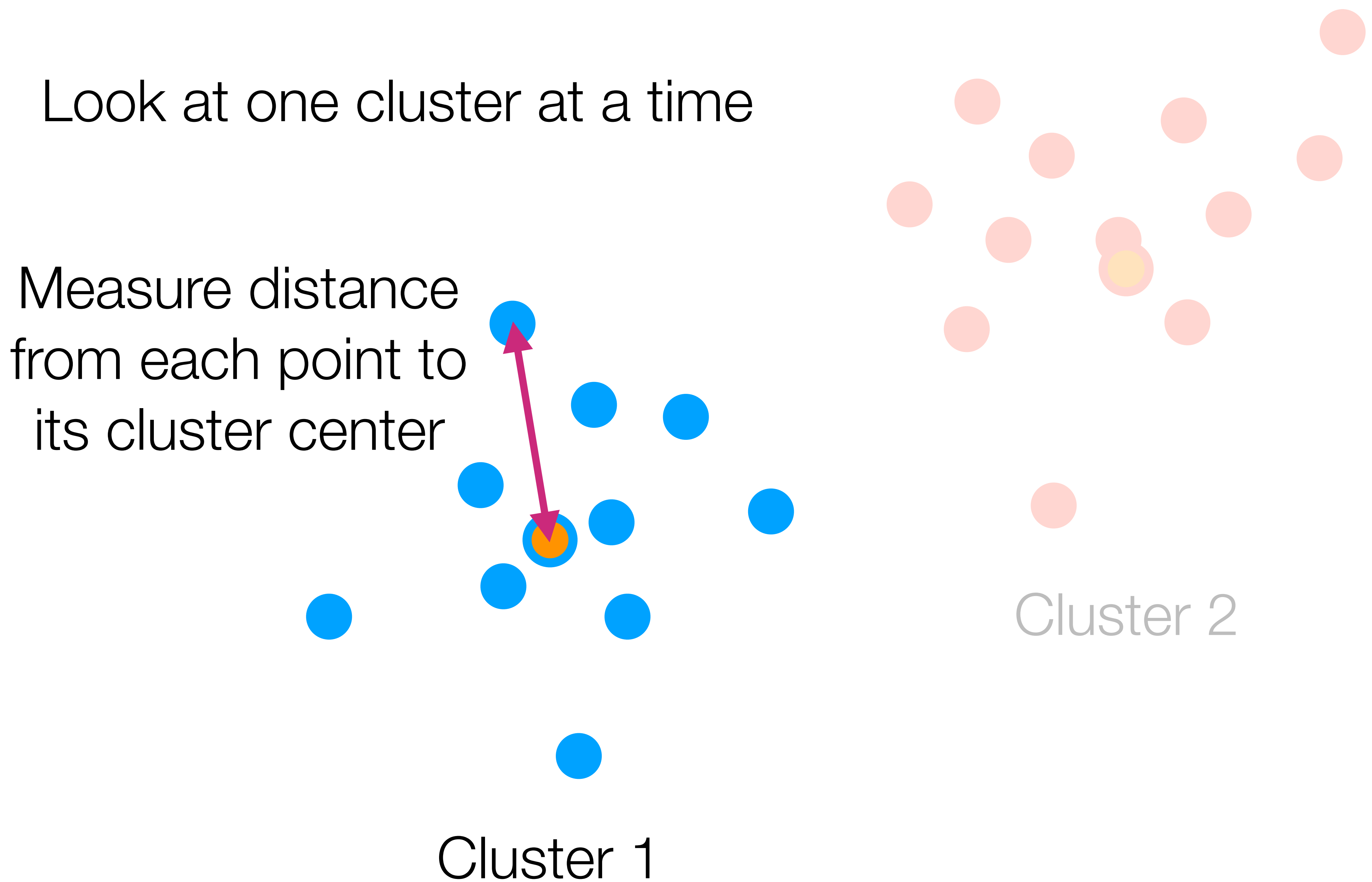


Residual Sum of Squares

Look at one cluster at a time



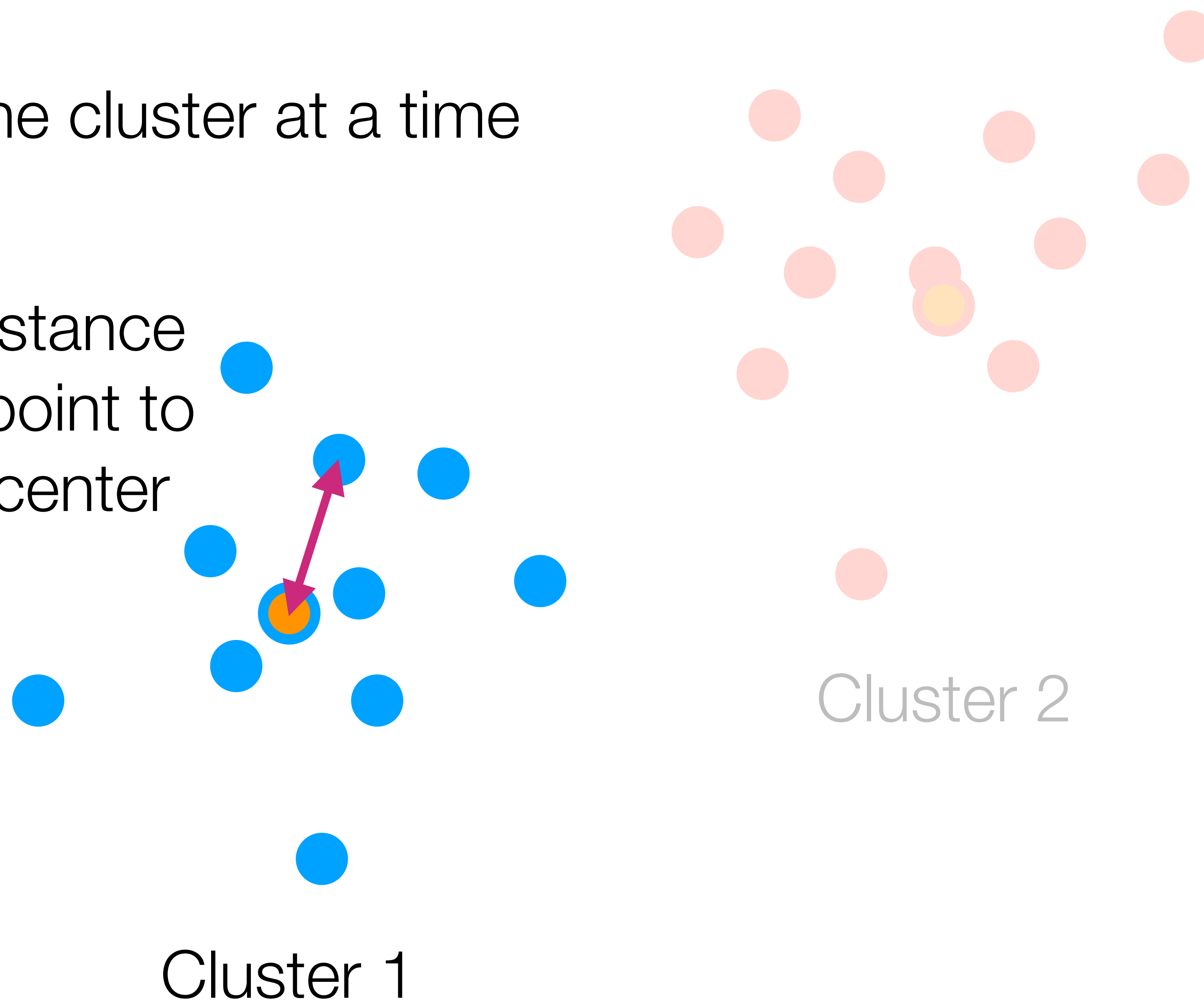
Residual Sum of Squares



Residual Sum of Squares

Look at one cluster at a time

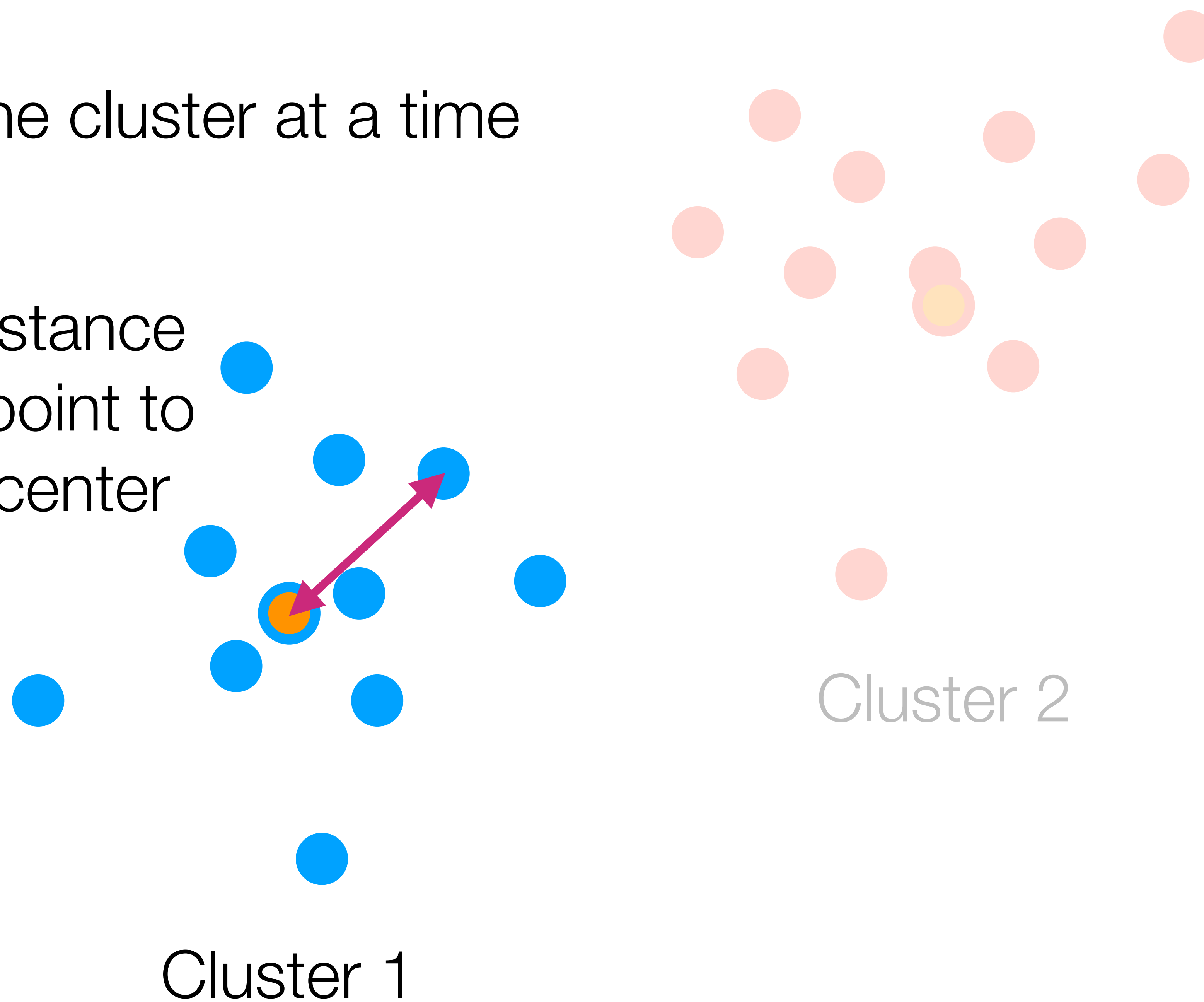
Measure distance
from each point to
its cluster center



Residual Sum of Squares

Look at one cluster at a time

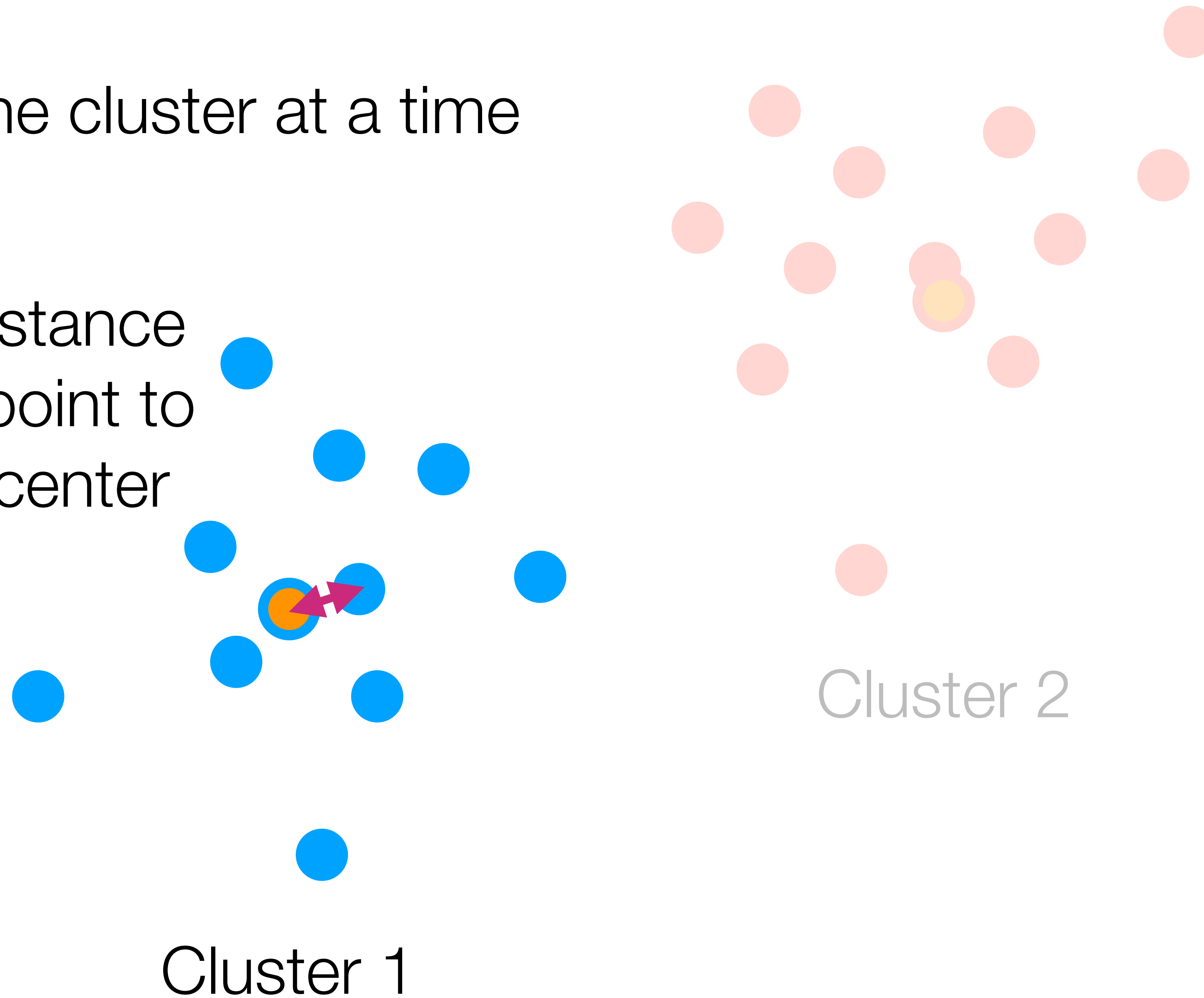
Measure distance from each point to its cluster center



Residual Sum of Squares

Look at one cluster at a time

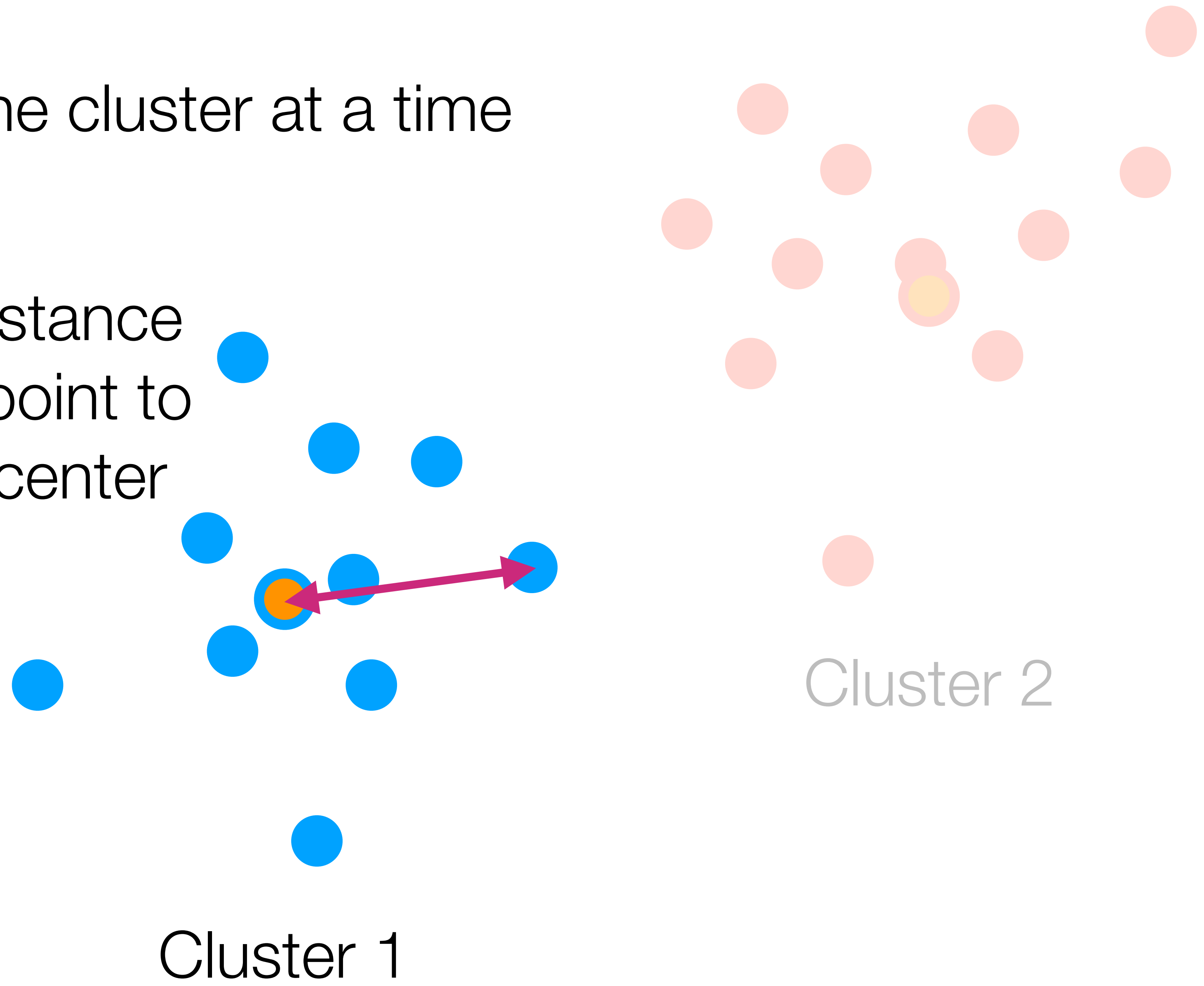
Measure distance
from each point to
its cluster center



Residual Sum of Squares

Look at one cluster at a time

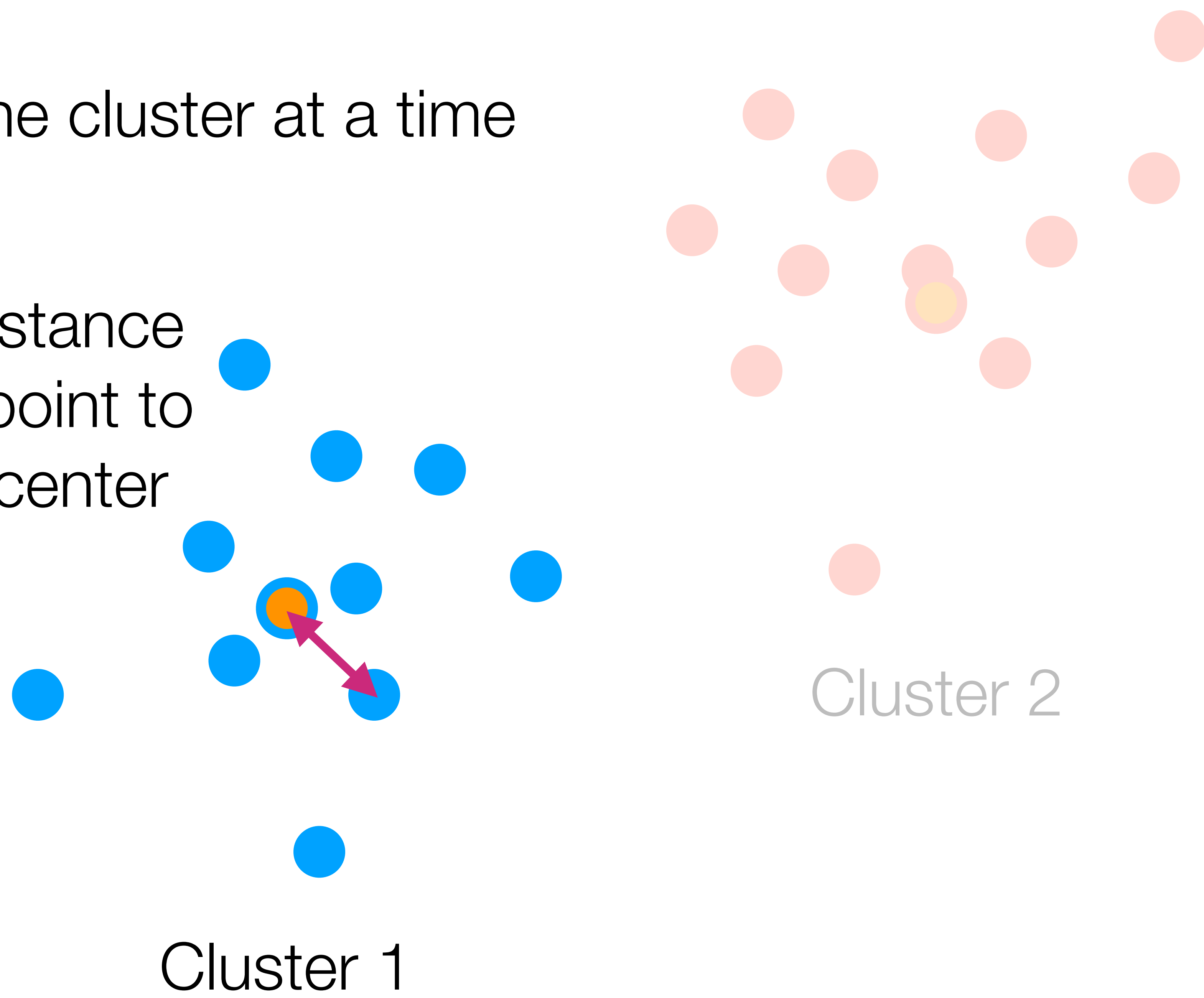
Measure distance from each point to its cluster center



Residual Sum of Squares

Look at one cluster at a time

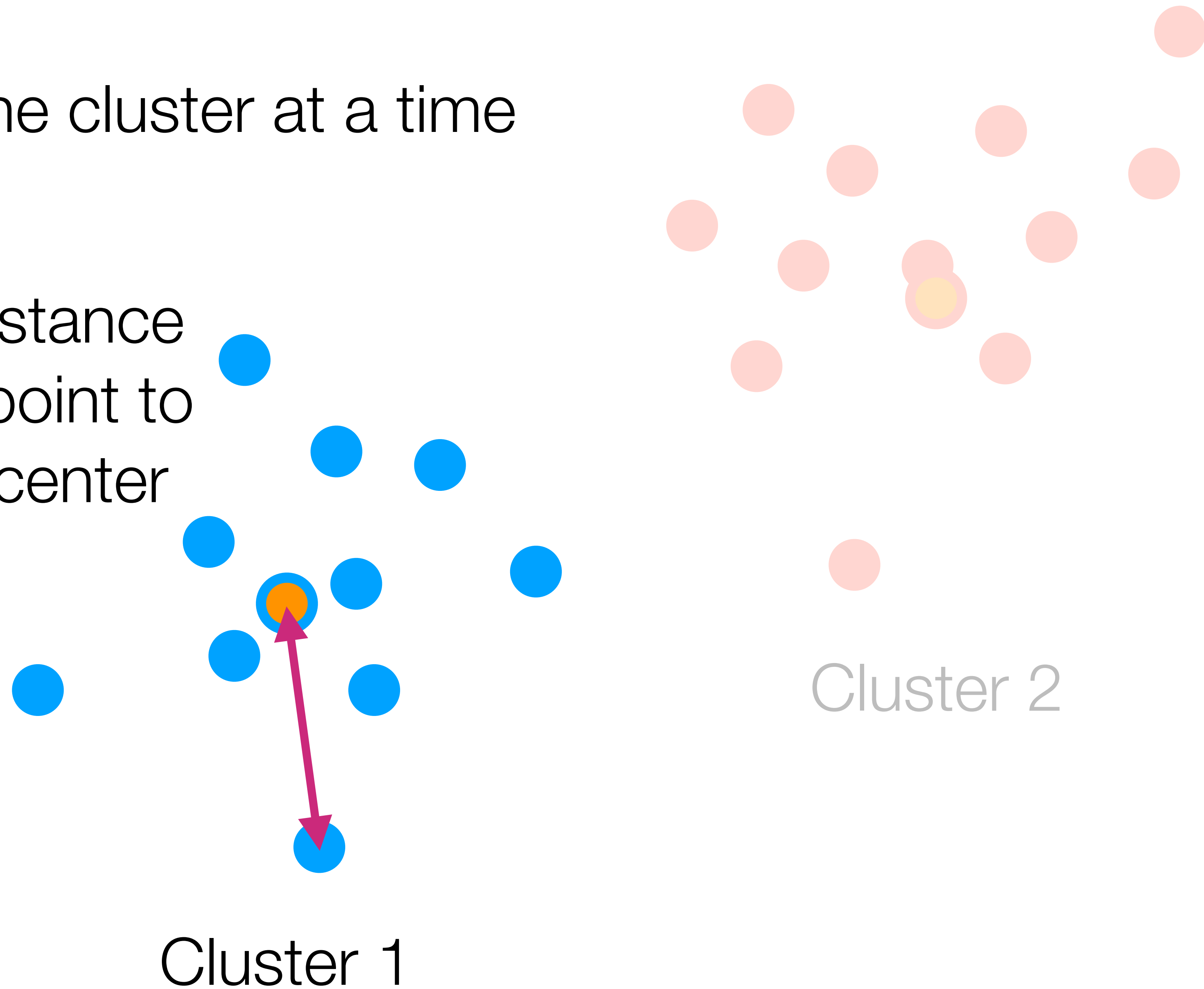
Measure distance from each point to its cluster center



Residual Sum of Squares

Look at one cluster at a time

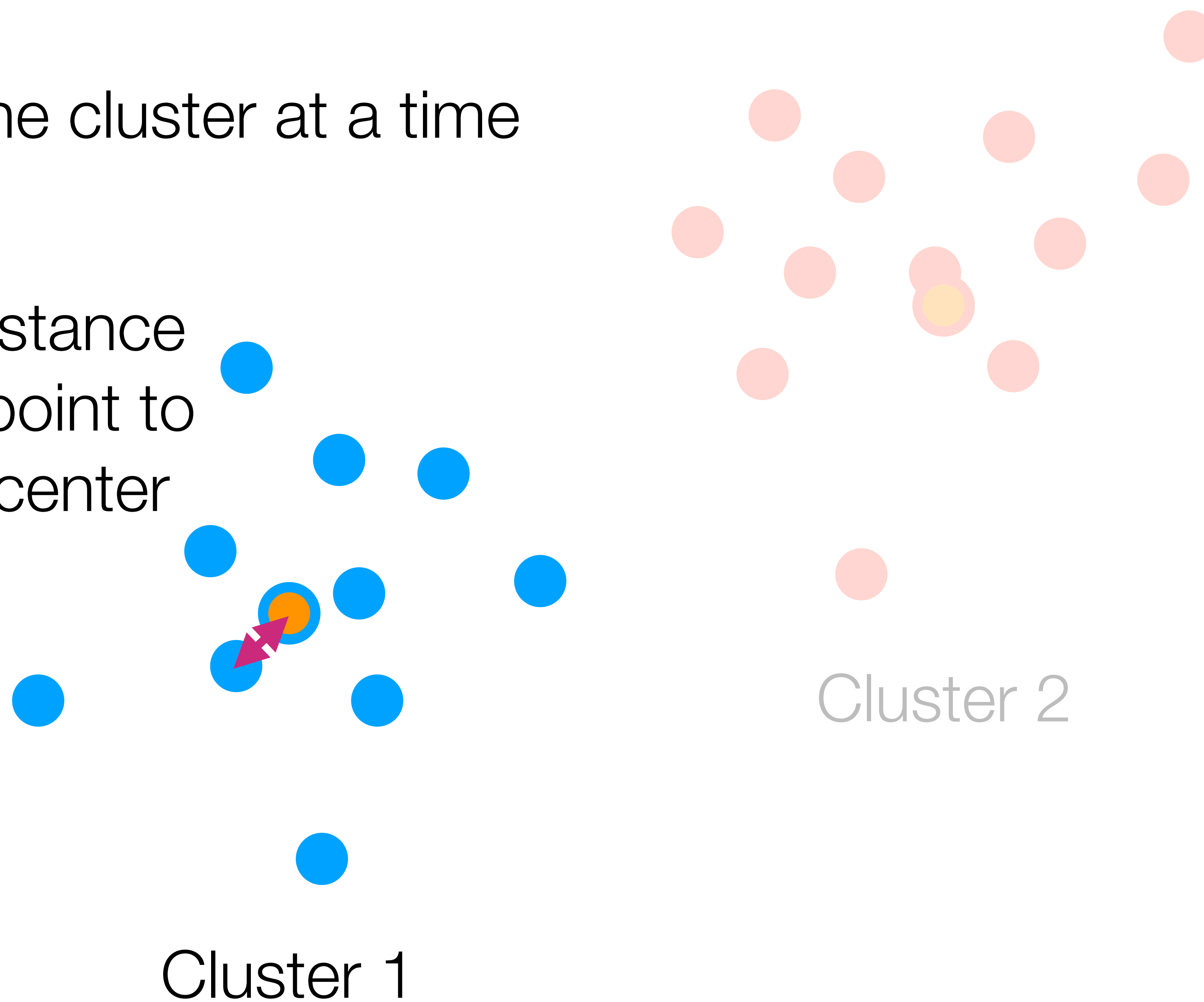
Measure distance from each point to its cluster center



Residual Sum of Squares

Look at one cluster at a time

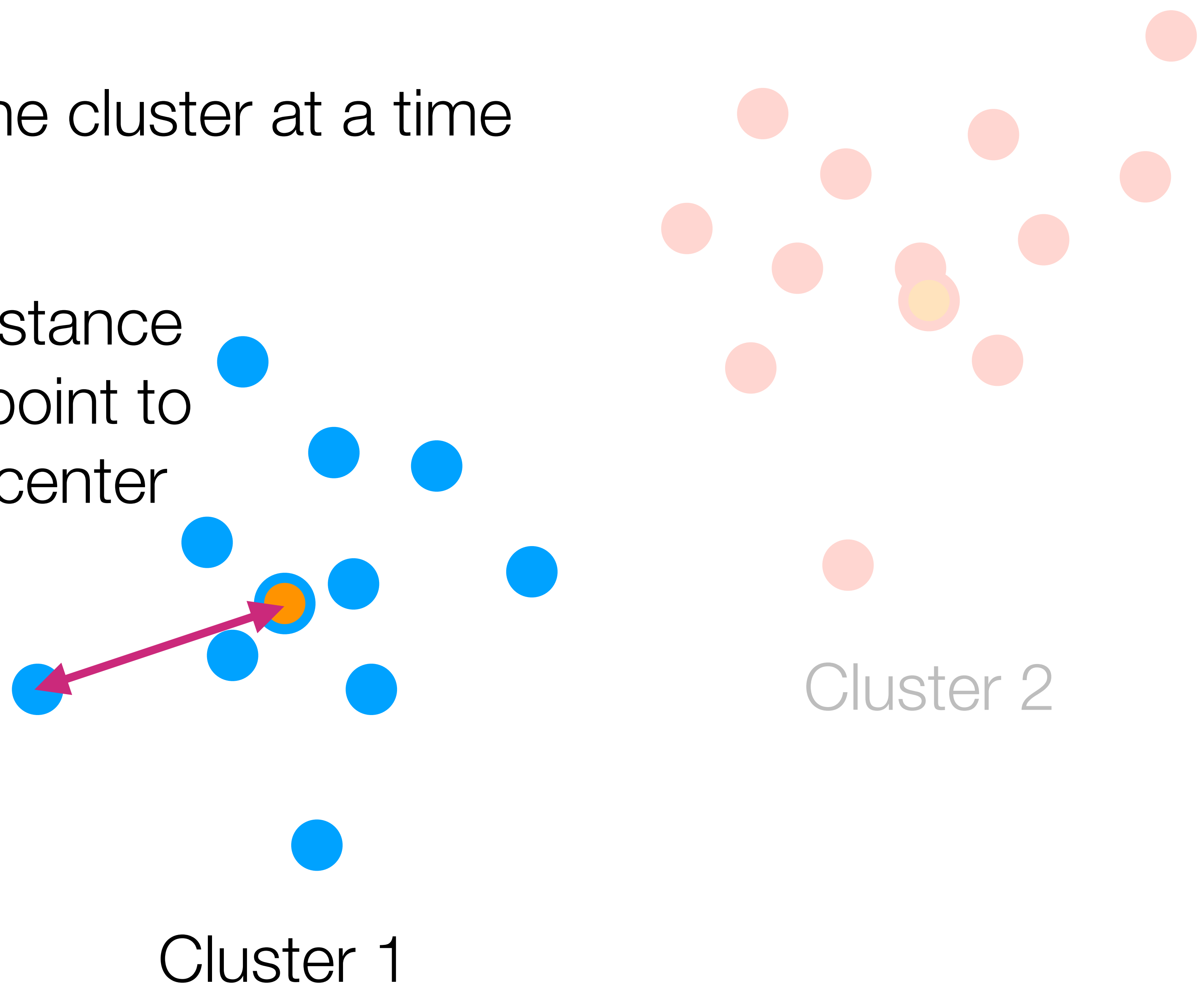
Measure distance
from each point to
its cluster center



Residual Sum of Squares

Look at one cluster at a time

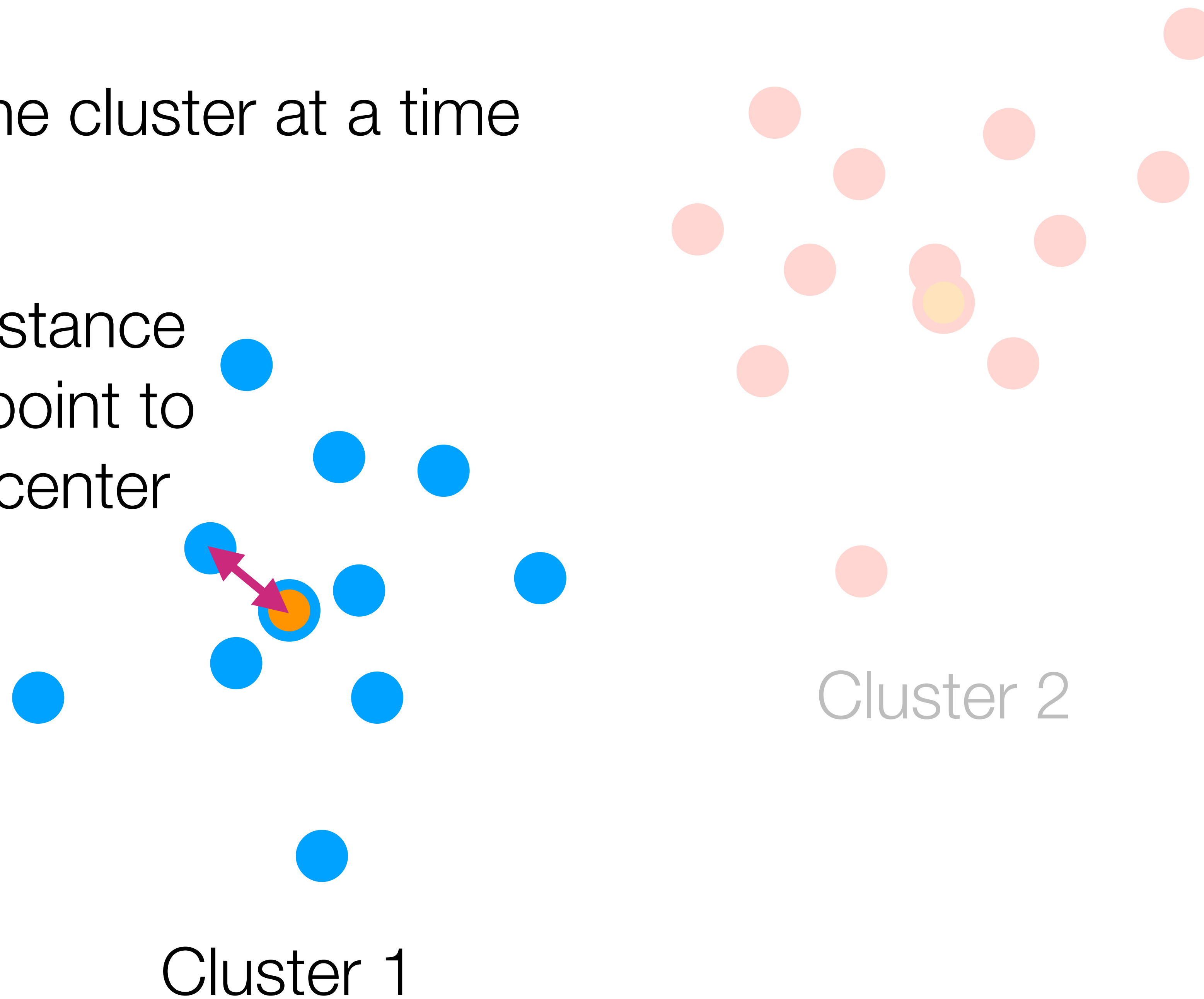
Measure distance from each point to its cluster center



Residual Sum of Squares

Look at one cluster at a time

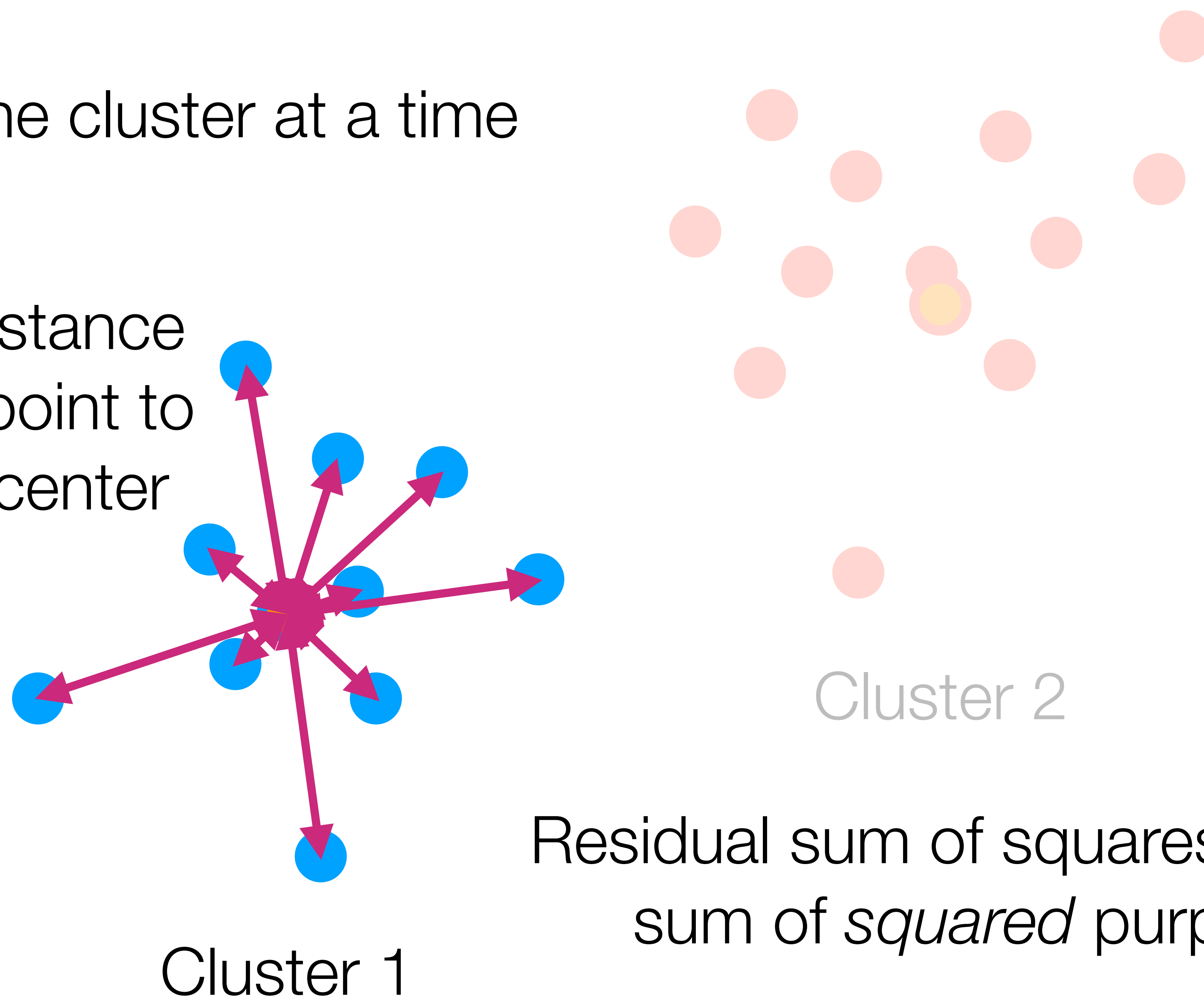
Measure distance from each point to its cluster center



Residual Sum of Squares

Look at one cluster at a time

Measure distance from each point to its cluster center

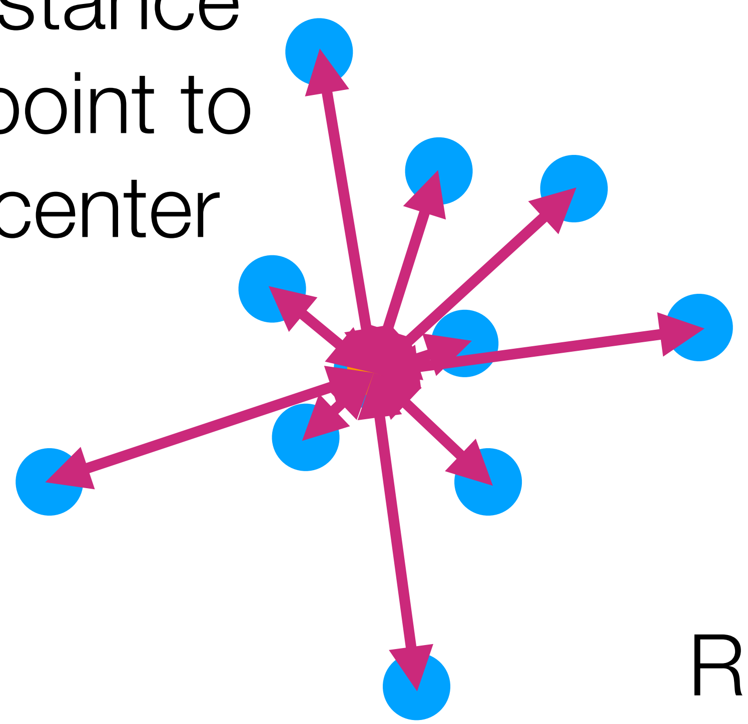


Residual sum of squares for cluster 1: sum of *squared* purple lengths

Residual Sum of Squares

Look at one cluster at a time

Measure distance from each point to its cluster center



Cluster 1

Cluster 2

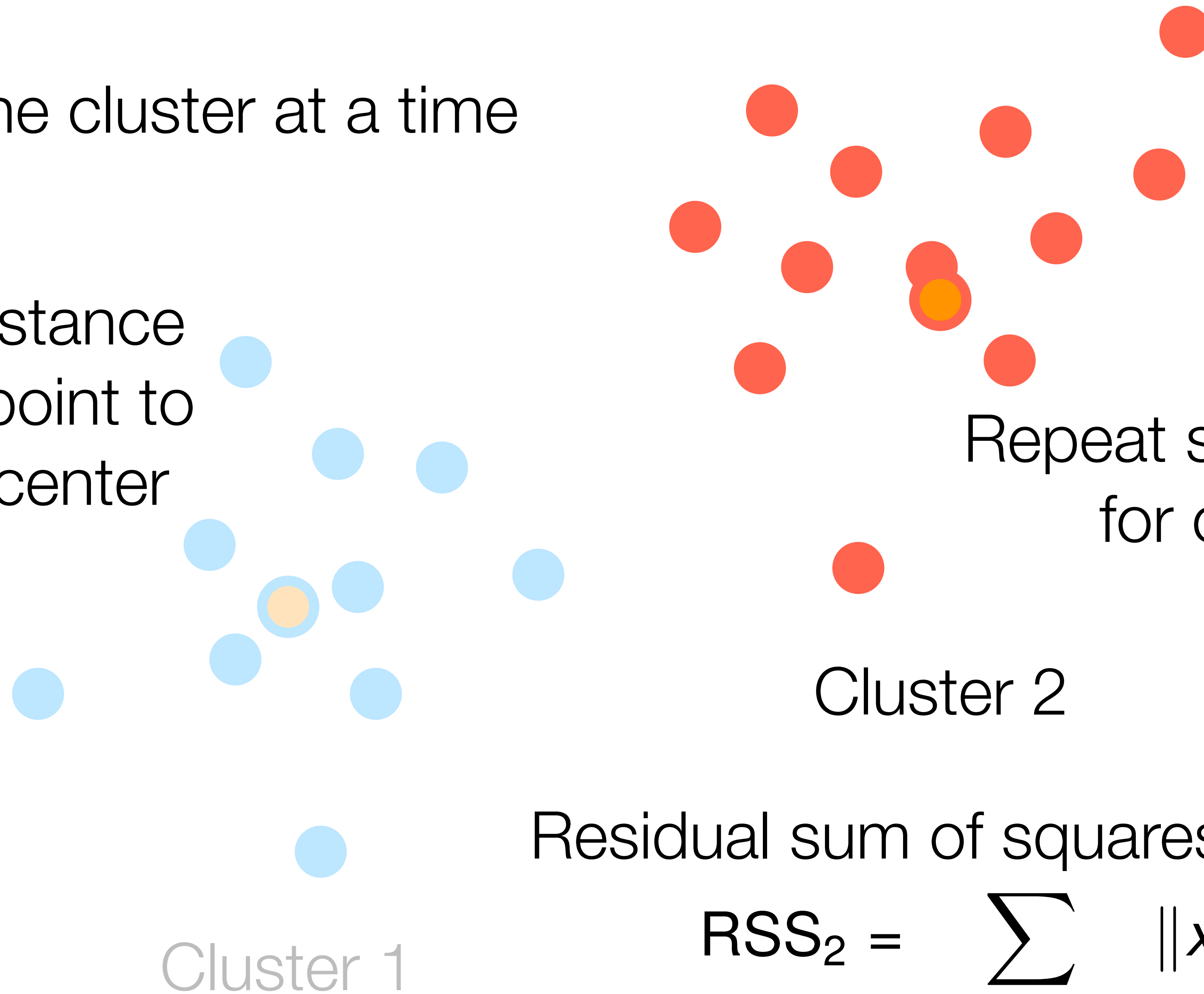
Residual sum of squares for cluster 1:

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

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

Residual sum of squares for cluster 2:

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

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