

Analytics & Machine Learning in Data Systems (Part 3)

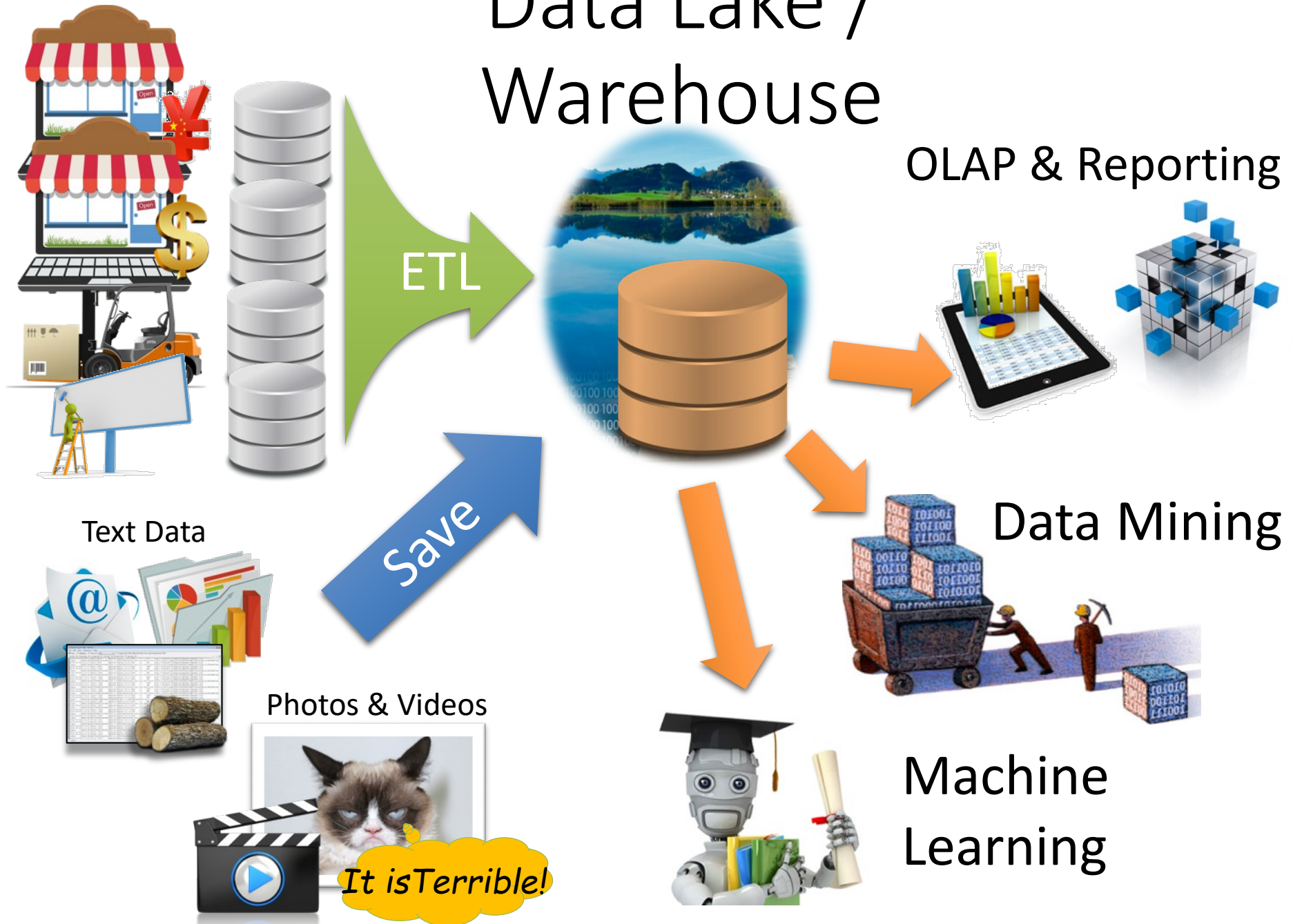
Course Textbook Chapters 26

Newer Material:

- Data Lake: https://en.wikipedia.org/wiki/Data_lake
- K-Means : https://en.wikipedia.org/wiki/K-means_clustering

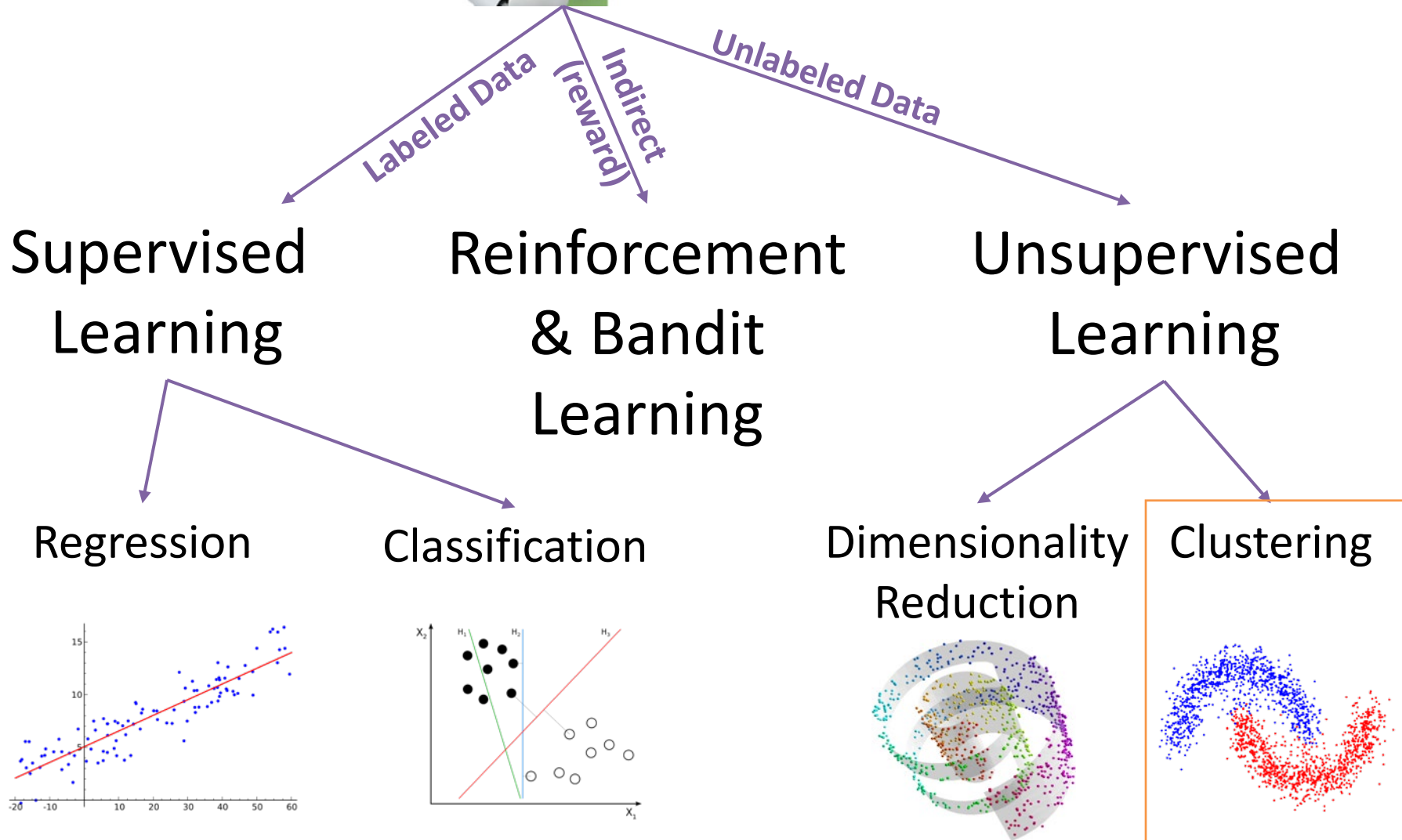
Joseph E. Gonzalez
jegonzal@cs.berkeley.edu

Data Lake / Warehouse





Taxonomy of Machine Learning



Clustering Images

- Given a collection of images cluster them into *meaningful groups*.



Clustering Images

- Given a collection of images cluster them into meaningful groups.

“Mountains”



“Forest”



“Beaches”



Clustering Images

- Given a collection of images cluster them into meaningful groups.

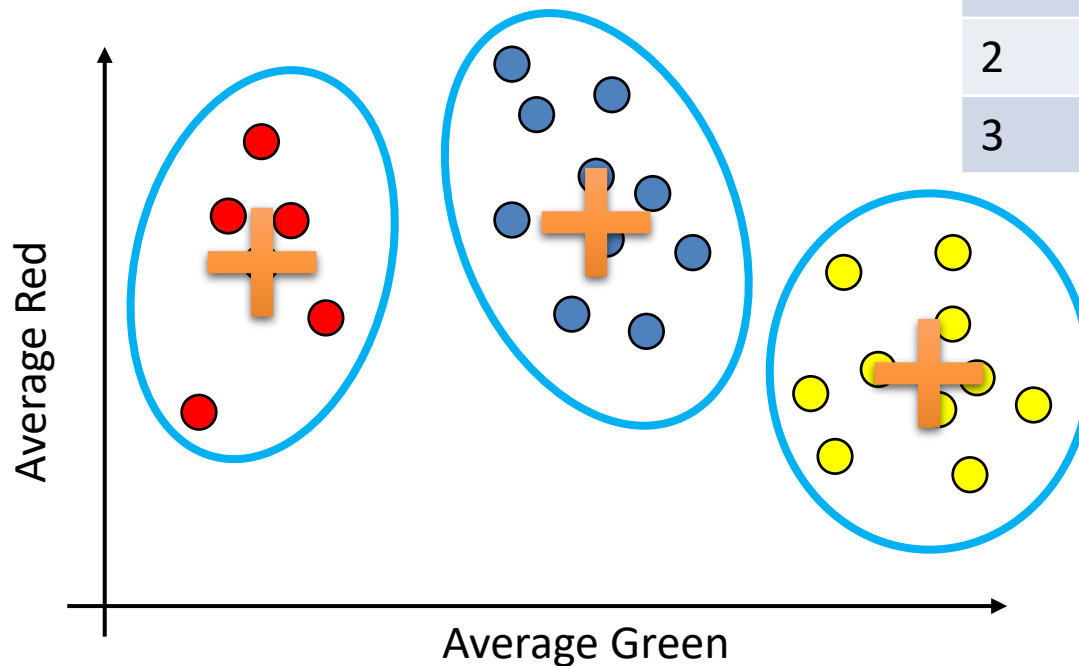


Image Id	Average Red	Average Green
1	123	200
2	212	103
3	55	35

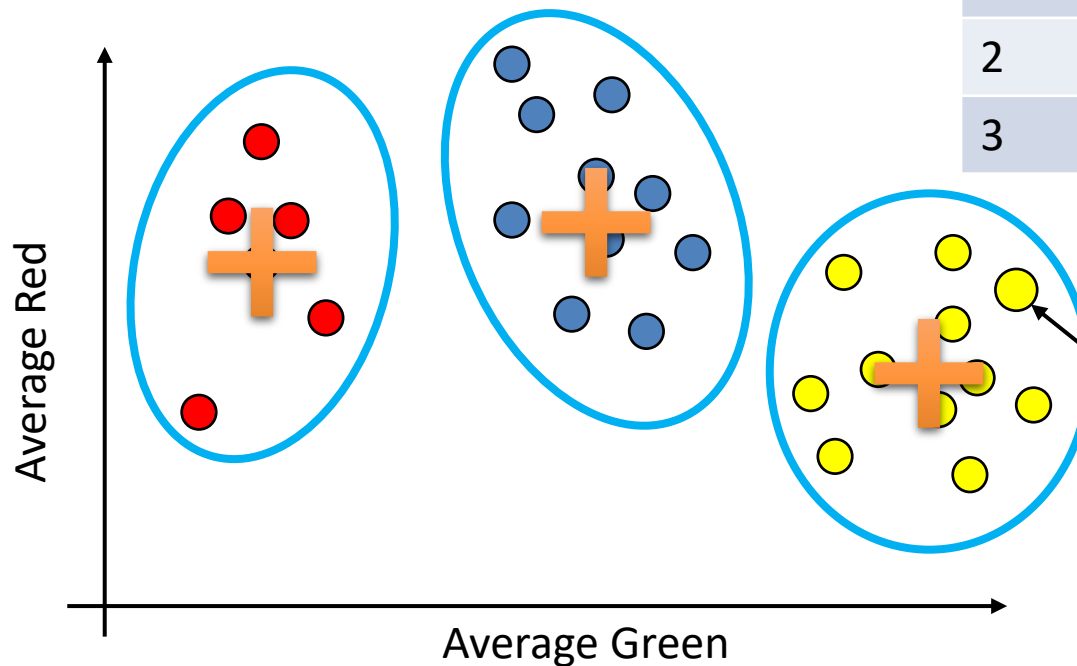
What makes a good clustering?

- All points are near the cluster center
- Spread between clusters > spread within clusters

Clustering Images

- Given a collection of images cluster them into meaningful groups.

Image Id	Average Red	Average Green
1	123	200
2	212	103
3	55	35



What happens when a new point arrives?

Predict “label” based on existing clusters (Yellow)

How do we Compute a Clustering?

Many different clustering models and algorithms:

➤ Feature Based Clustering: *Points in R^d*

- **K-Means:** EM on Symmetric Gaussians ← We will learn this one
- **Mixture Models:** Generalized k-means
- ...

➤ Spectral Methods: *Similarity Function Between Items*

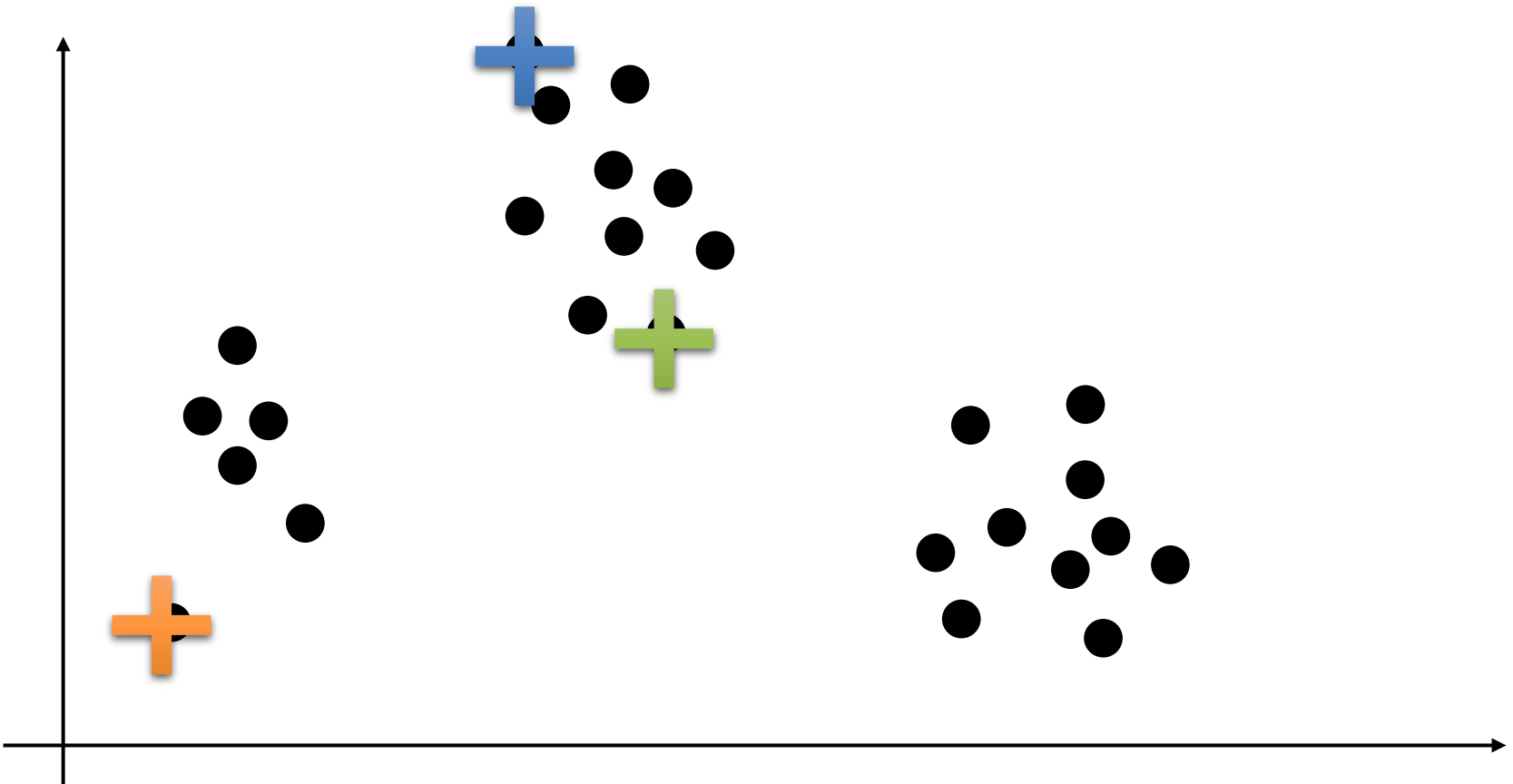
- **Similarity based clustering:** *A and B are co-purchased*
- **Graph clustering:** *Cities based on road network*
- ...

➤ Hierarchical Clustering: *clustering nested items*

- **Latent Dirichlet Allocation:** *Documents based on words*
 - *Developed at Berkeley and widely used!*
- ...

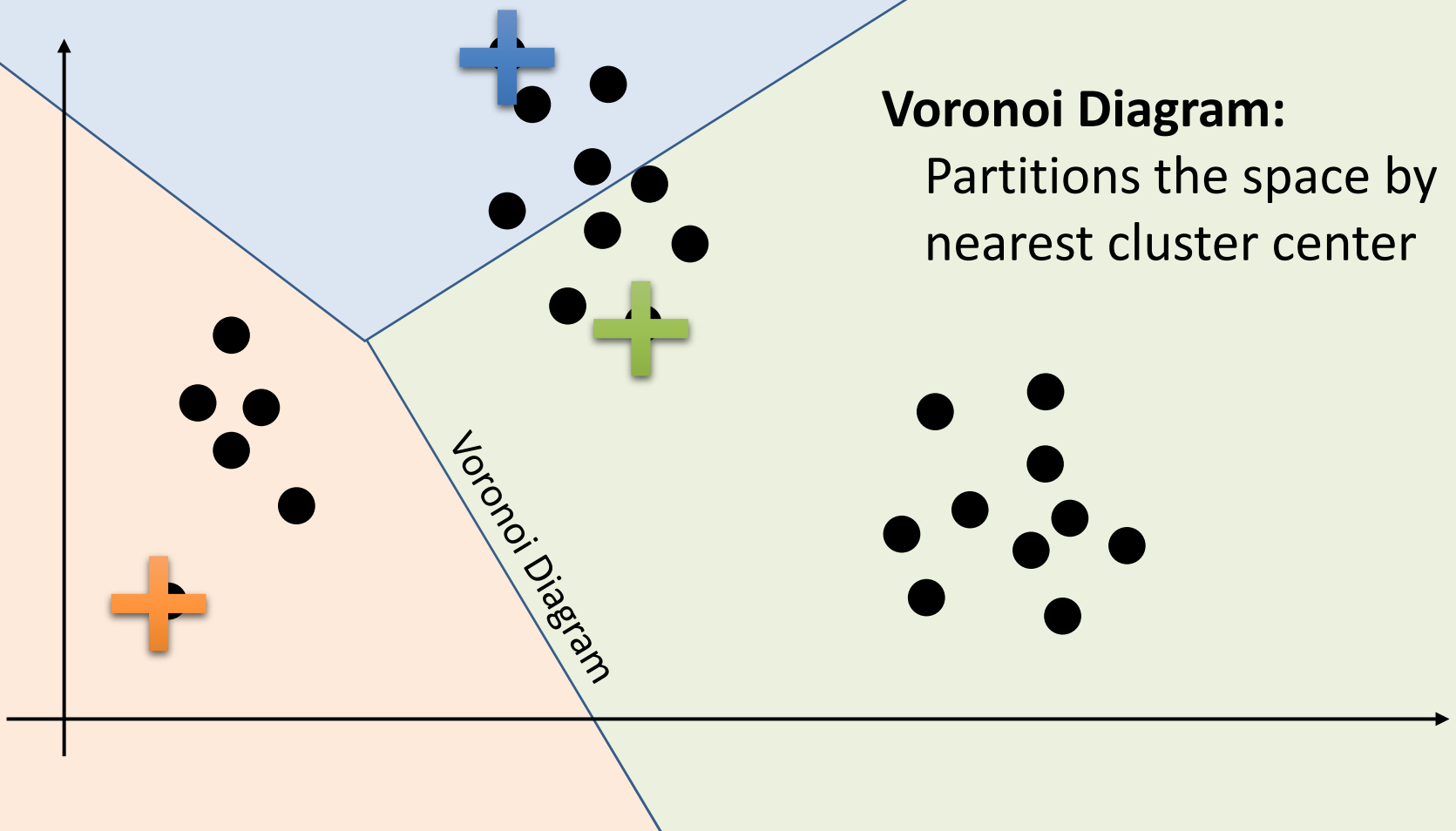
K-Means Clustering: *Intuition*

- Input K: The number of clusters to find
- Pick an initial set of points as cluster centers



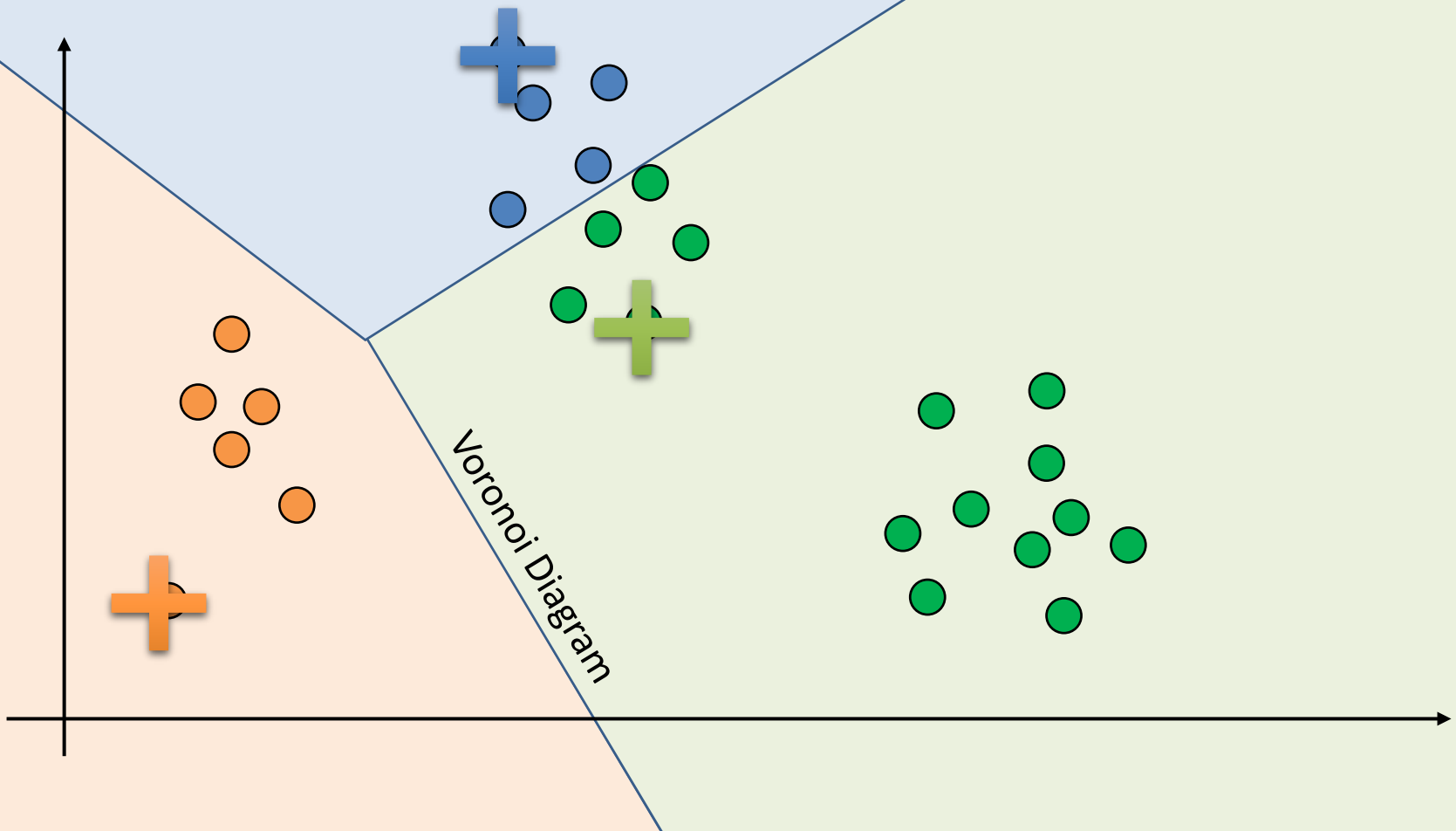
K-Means Clustering: *Intuition*

- For each data point find the cluster nearest center



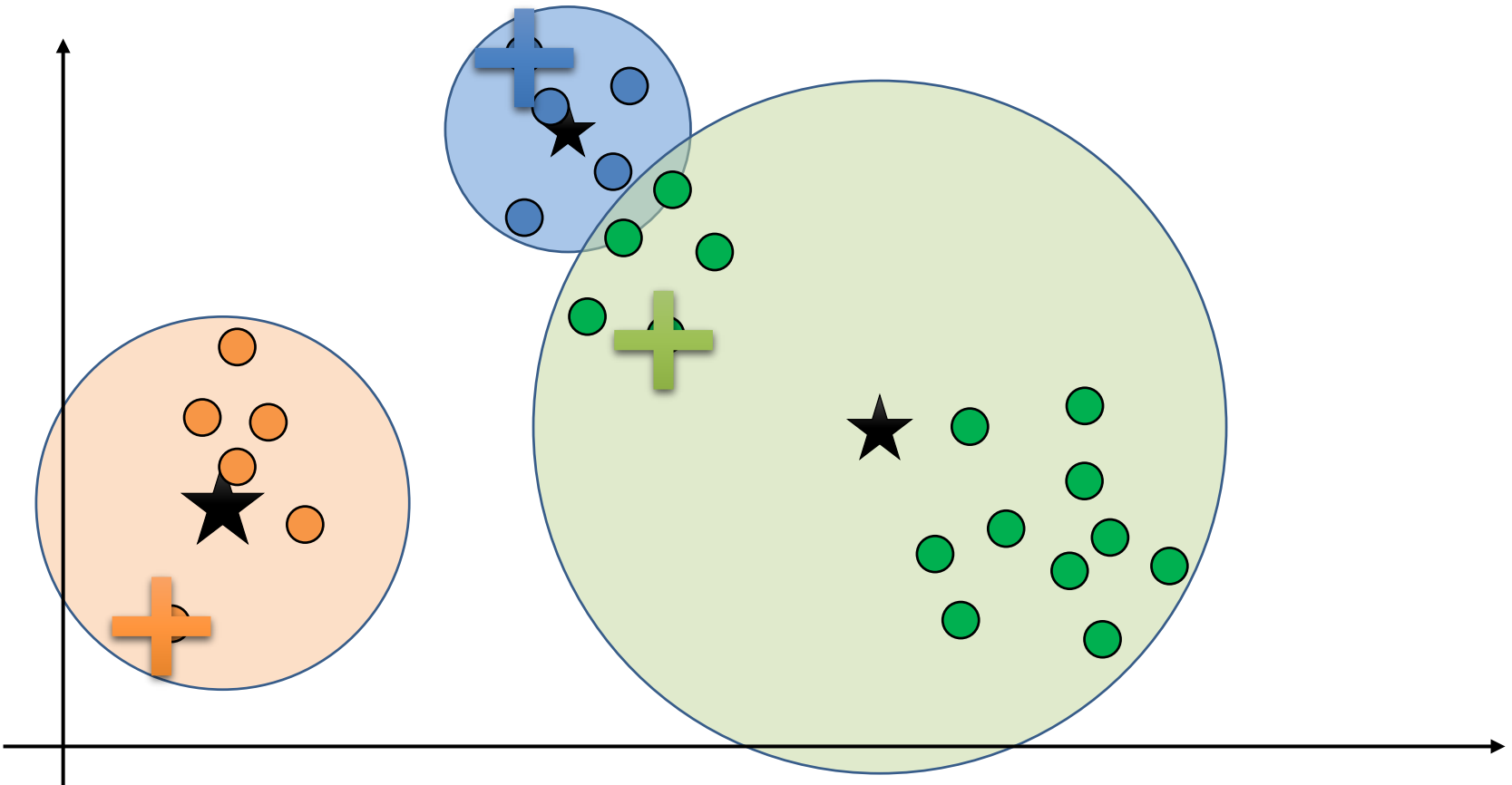
K-Means Clustering: *Intuition*

- For each data point find the cluster nearest center



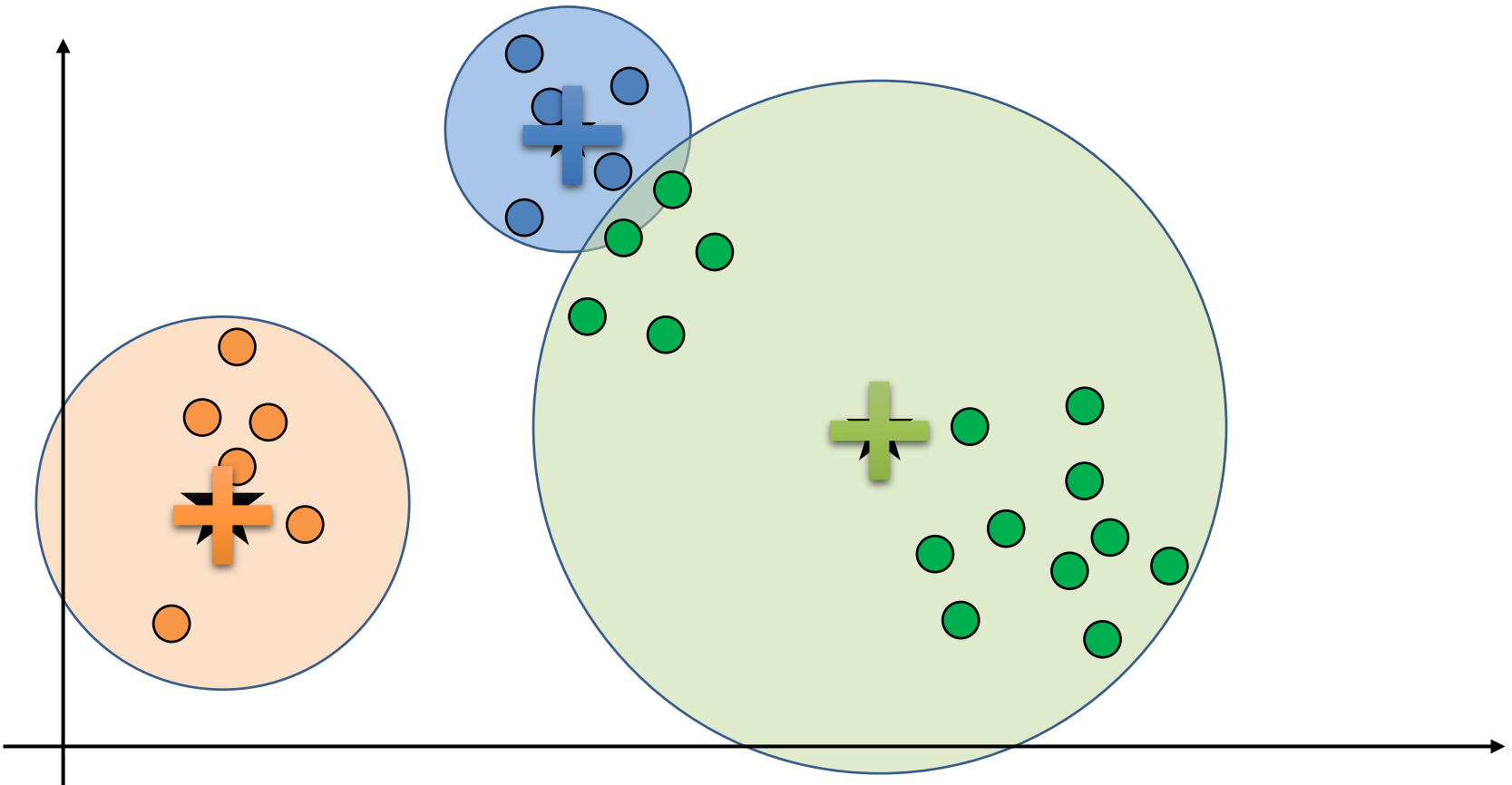
K-Means Clustering: *Intuition*

- Compute mean of points in each “cluster”



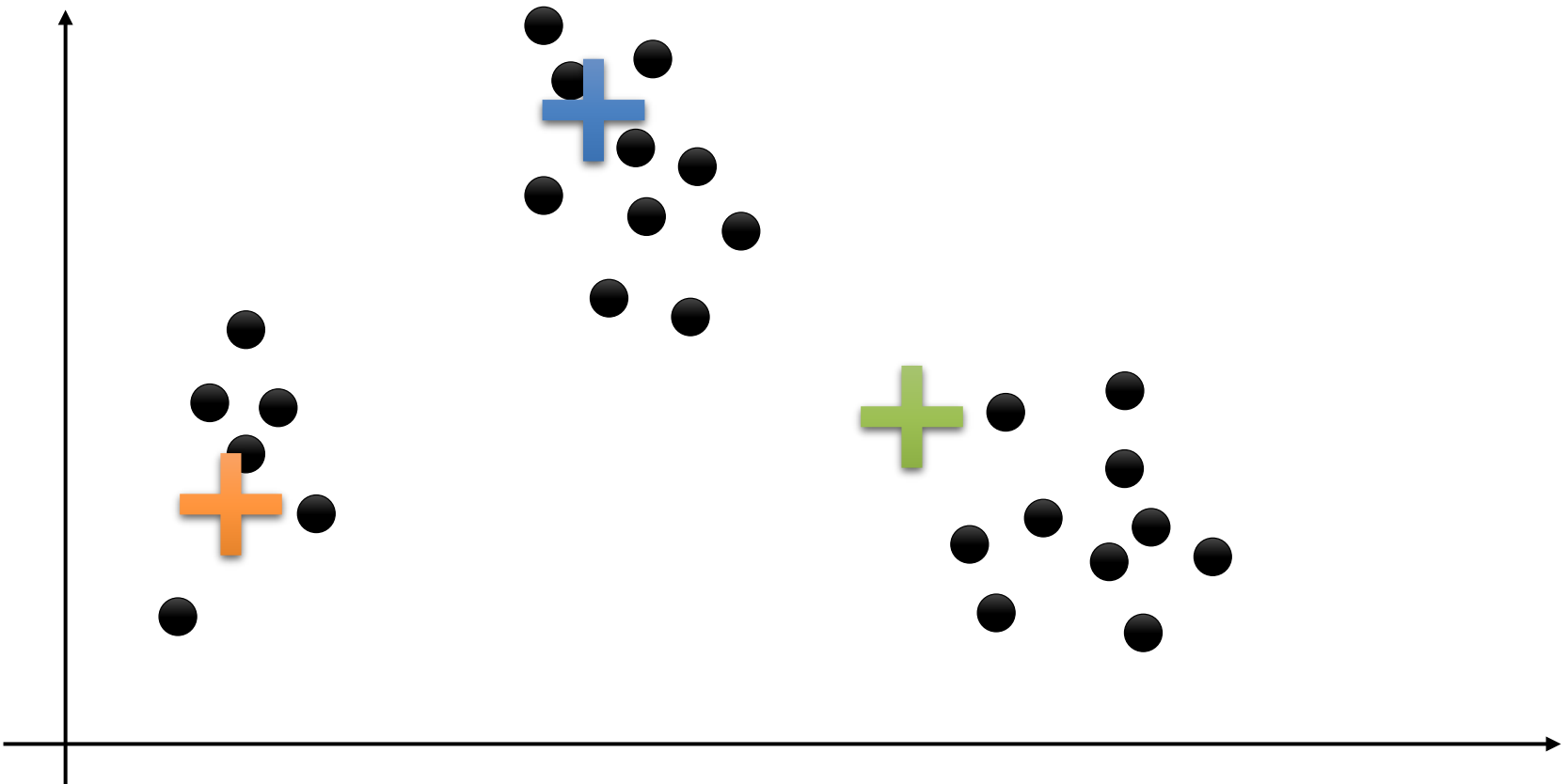
K-Means Clustering: *Intuition*

- Adjust cluster centers to be the mean of the cluster



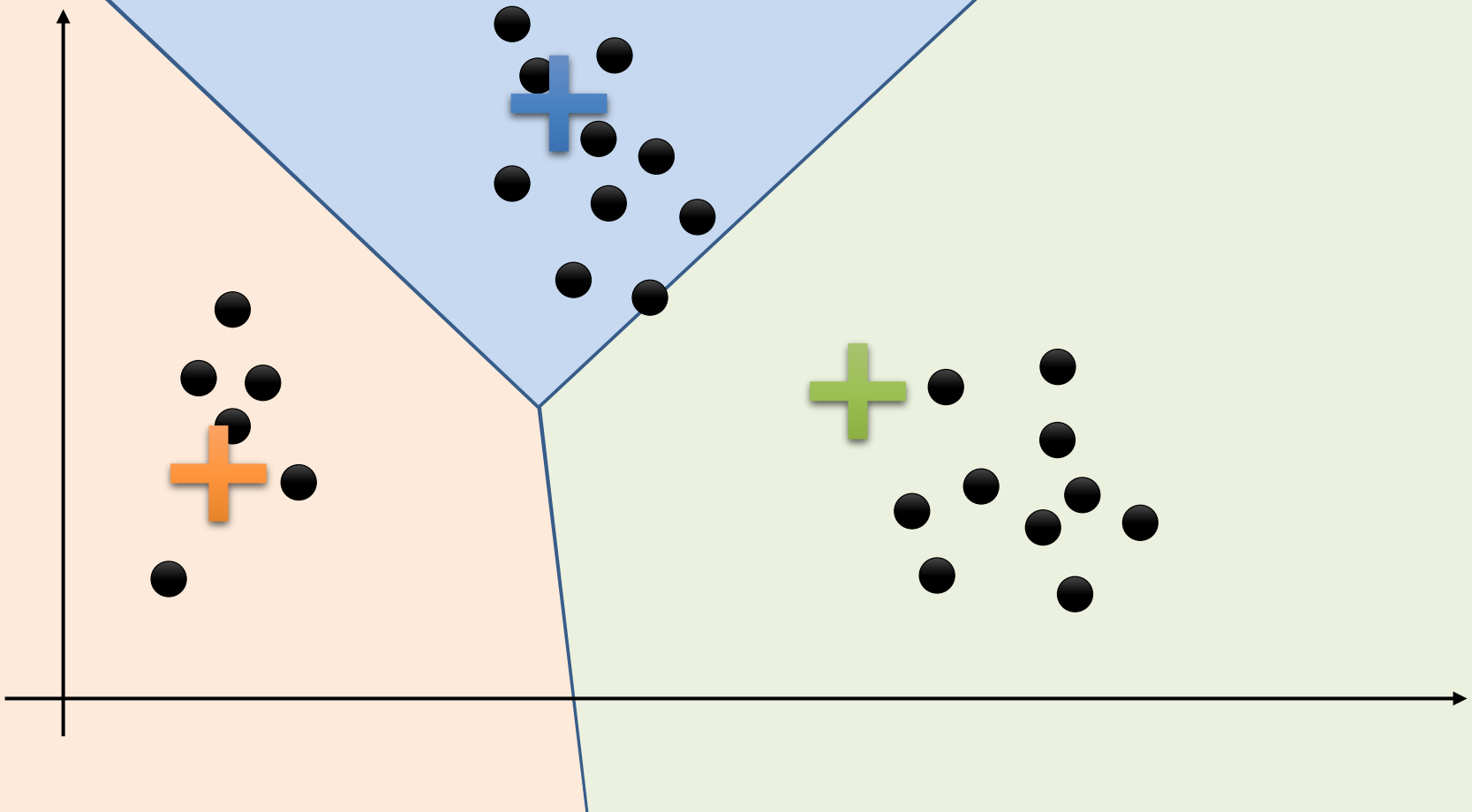
K-Means Clustering: *Intuition*

- Improved?
- Repeat



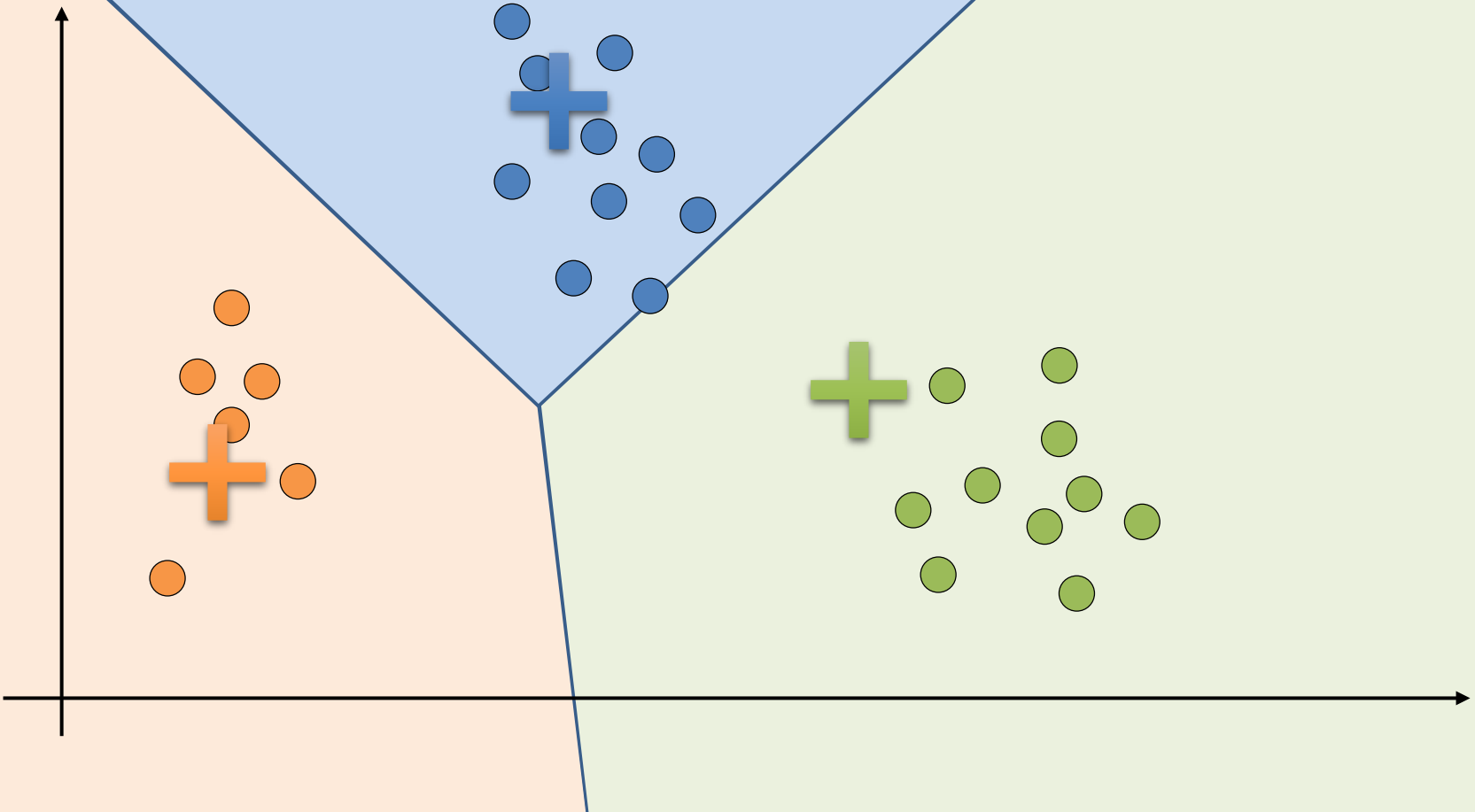
K-Means Clustering: *Intuition*

➤ Assign Points



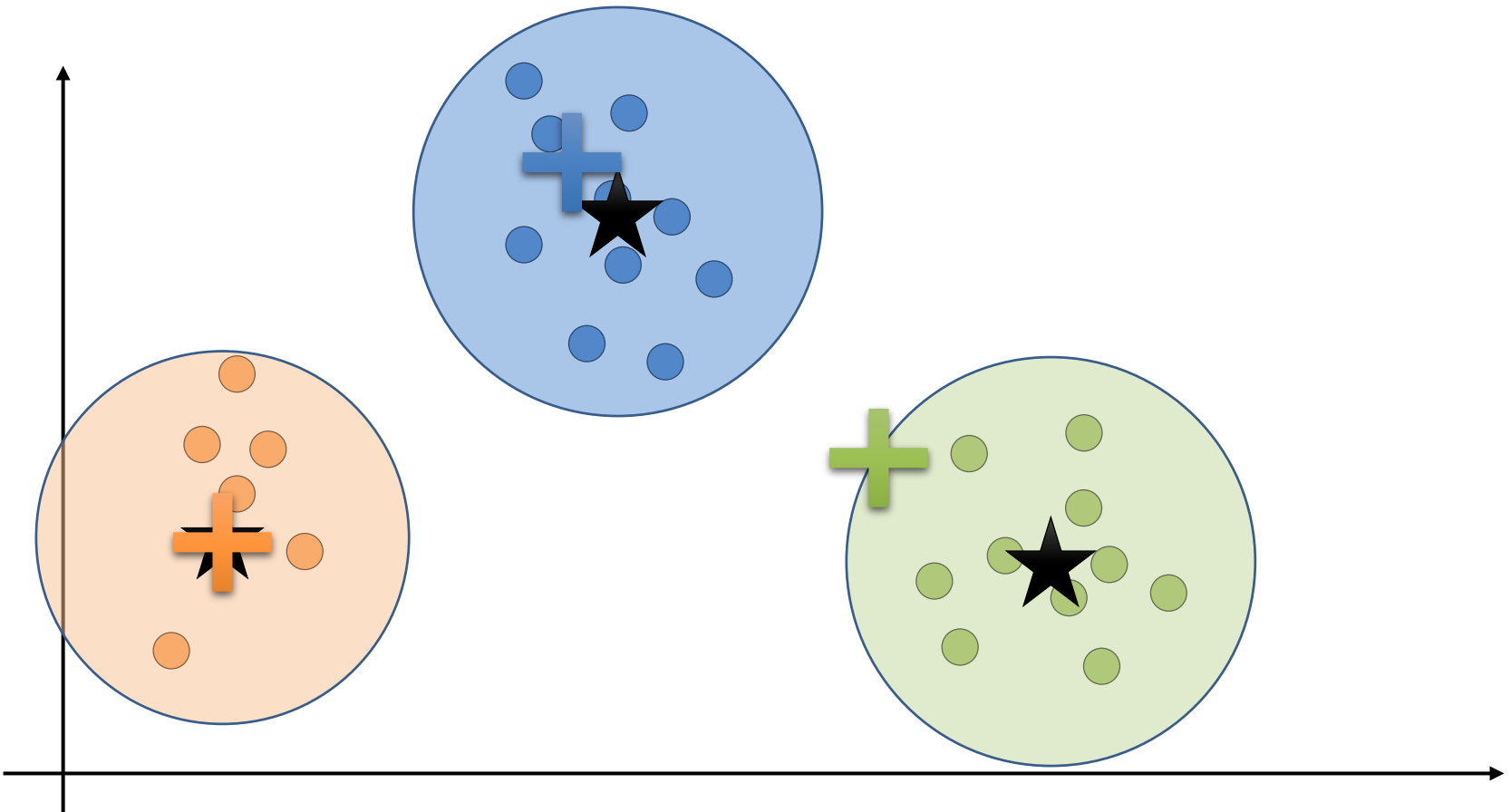
K-Means Clustering: *Intuition*

➤ Assign Points



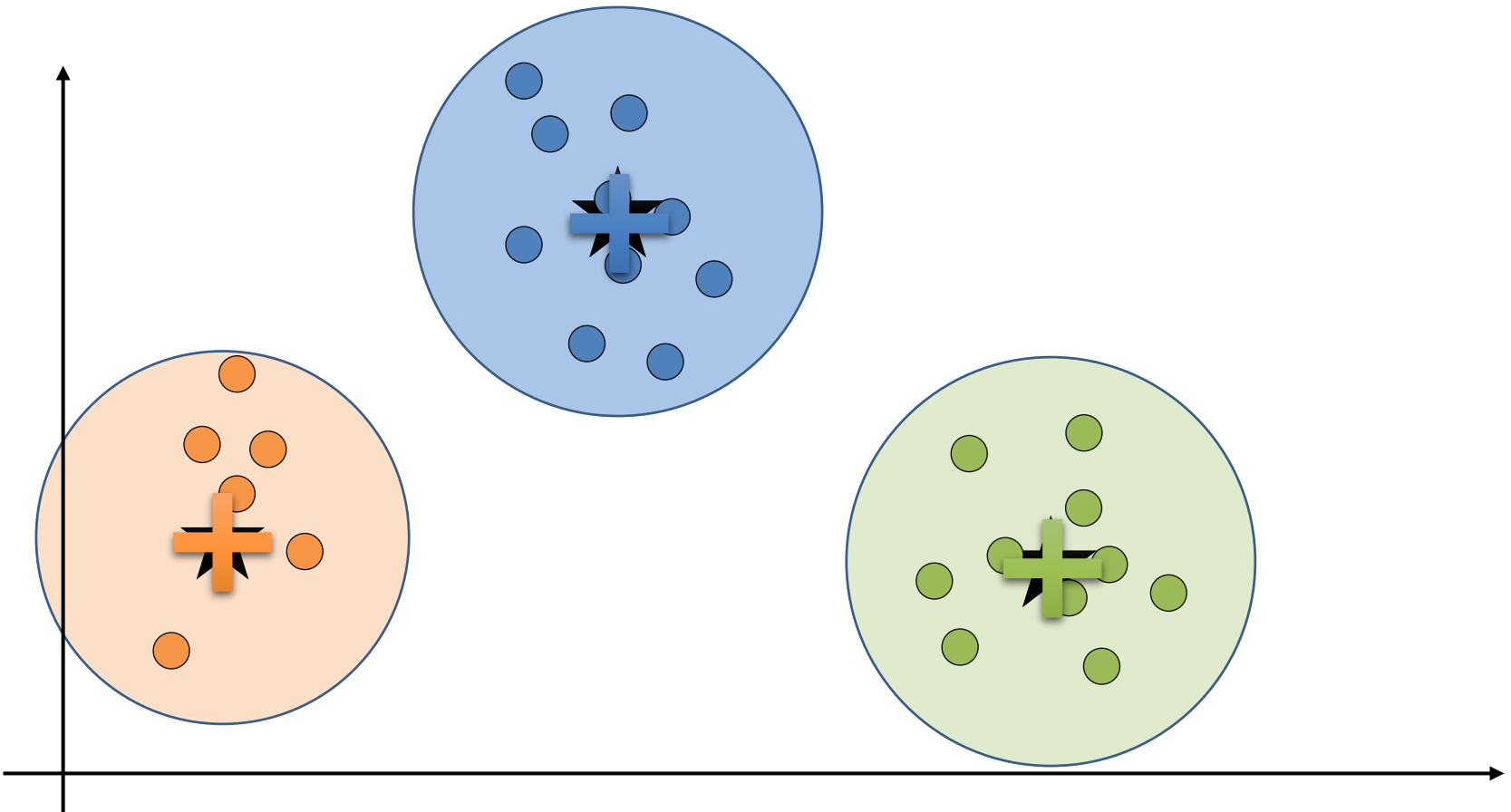
K-Means Clustering: *Intuition*

- Compute cluster means



K-Means Clustering: *Intuition*

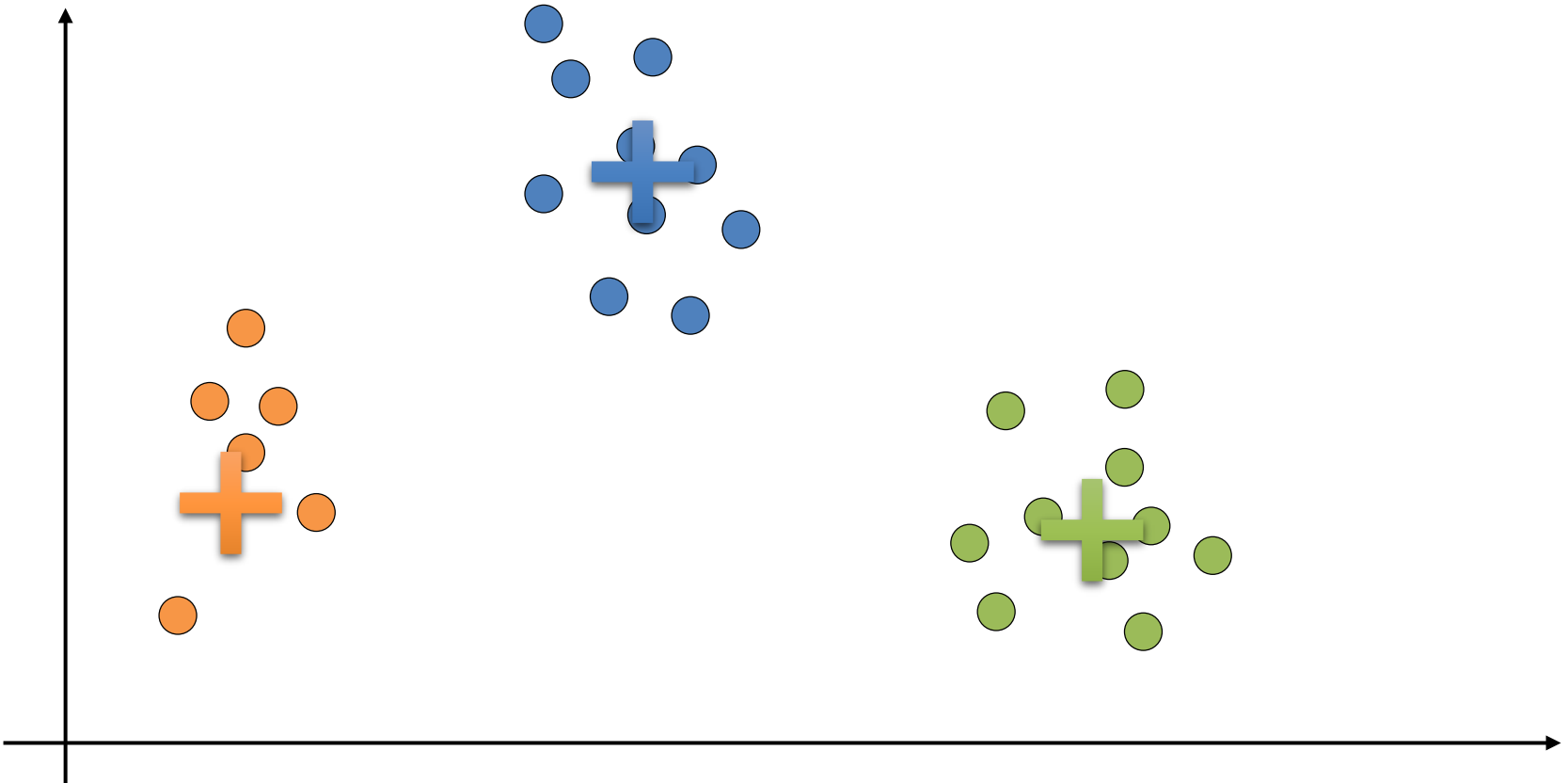
- Update cluster centers



K-Means Clustering: *Intuition*

➤ Repeat?

- Yes to check that nothing changes → Converged!



K-Means Algorithm: Details

```
centers ← pick k initial Centers
```

```
while (centers are changing) {
```

```
    // Compute the assignments (E-Step)
```

```
    asg ← [(x, nearest(centers, x)) for x in data]
```

What do we mean by “nearest”:

A: Euclidean Distance

$$\arg \min_{c \in \text{centers}} \|c - x\|_2^2 = \sum_{i=1}^d (c_i - x_i)^2$$

K-Means Algorithm: Details

```
centers ← pick k initial Centers
```

Compute the
“Expected” Assignment

```
while (centers are changing) {
```

```
    // Compute the assignments (E-Step)
```

```
    asg ← [(x, nearest(centers, x)) for x in data]
```

```
    // Compute the new centers (M-Step)
```

```
    for i in range(k):
```

```
        centers[i] =
```

Find centers that maximize the
data “likelihood”

```
            mean([x for (x, c) in asg if c == i])
```

```
}
```

K-Means Algorithm: Details

```
centers ← pick k initial Centers
```

```
while (centers are changing) {  
    // Compute the assignments (E-Step)  
    asg ← [(x, nearest(centers, x)) for x in data]  
  
    // Compute the new centers (M-Step)  
    for i in range(k):  
        centers[i] =  
            mean([x for (x, c) in asg if c == i])  
}
```

Guaranteed to
converge!

... to what?

To a local
optimum. 😞

Depends on
Initial Centers

K-Means Algorithm: Details

```
centers ← pick k initial Centers
```

How do we pick initial centers?

```
while (centers are changing) {  
    // Compute the assignments (E-Step)  
    asg ← [(x, nearest(centers, x)) for x in data]  
  
    // Compute the new centers (M-Step)  
    for i in range(k):  
        centers[i] =  
            mean([x for (x, c) in asg if c == i])  
}
```

Guaranteed to
converge!

... to what?

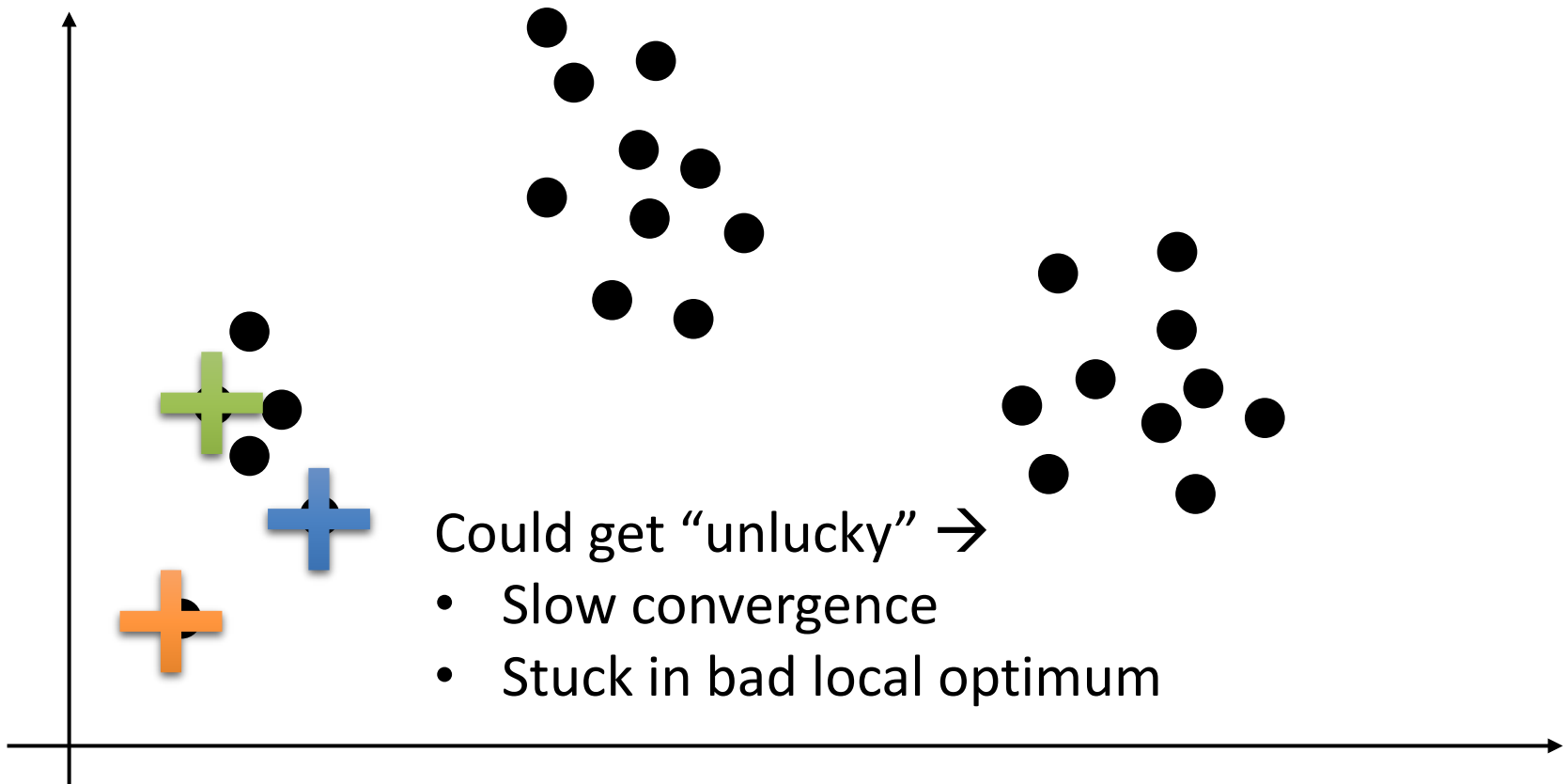
To a local
optimum. 😞

Depends on
Initial Centers

Picking the Initial Centers

➤ **Simple Strategy:** select k points at random

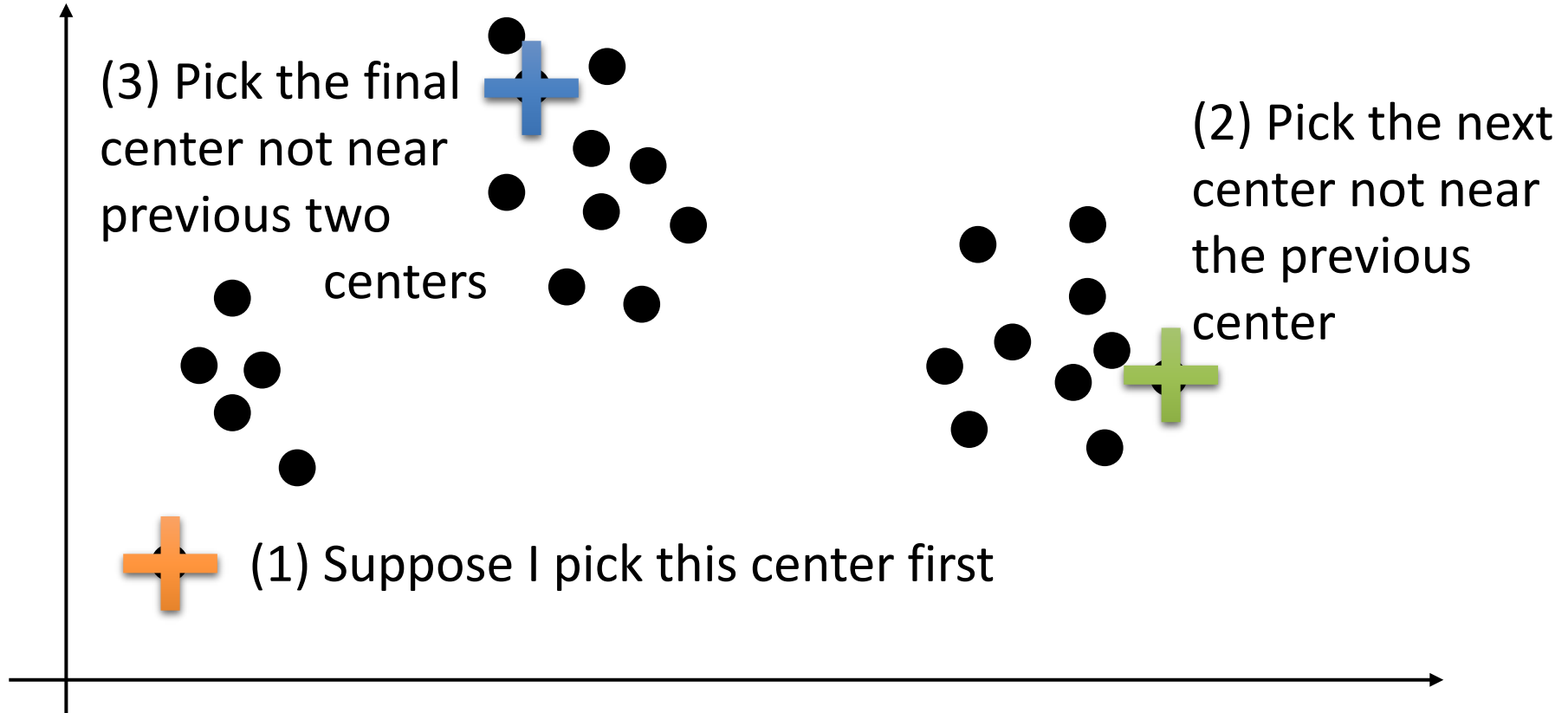
- What could go wrong?



Picking the Initial Centers

➤ **Better Strategy:** kmeans++

- Randomized approx. algorithm
- Intuition select points that are not near existing centers



K-Means++ Algorithm

```
centers ← set(randomly select a single point)
```

```
while len(centers) < k:
```

```
    # Compute the distance of each point
```

```
    # to its nearest center  $dSq = d^2$ 
```

```
    dSq ← [(x, dist_to_nearest(centers, x)^2) for x in data]
```

```
    # Sample a new point with probability
```

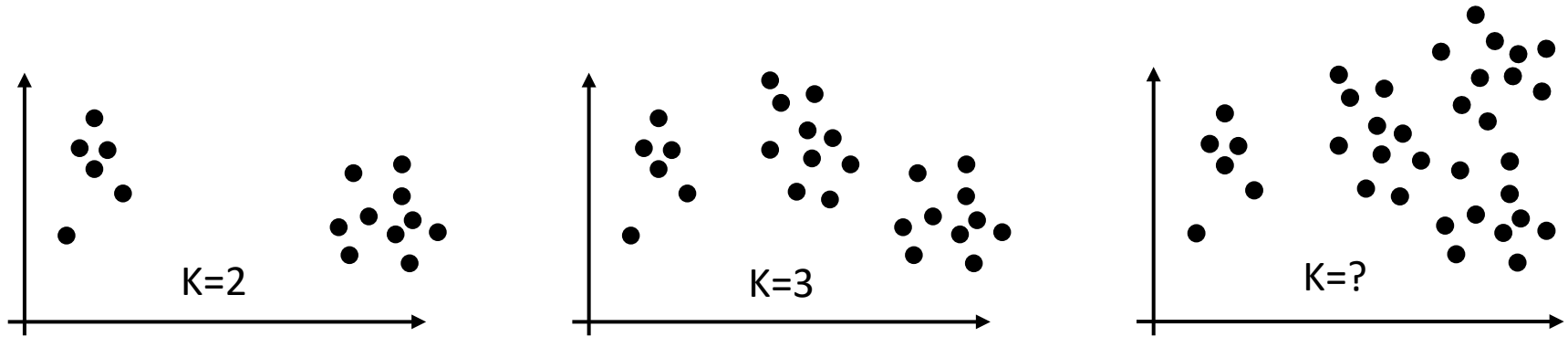
```
    # proportional to dSq
```

```
    c ← sample_one(data, prob = dSq / sum(dSq))
```

```
    # Update the clusters
```

```
    centers.add(c)
```


How do we choose K?

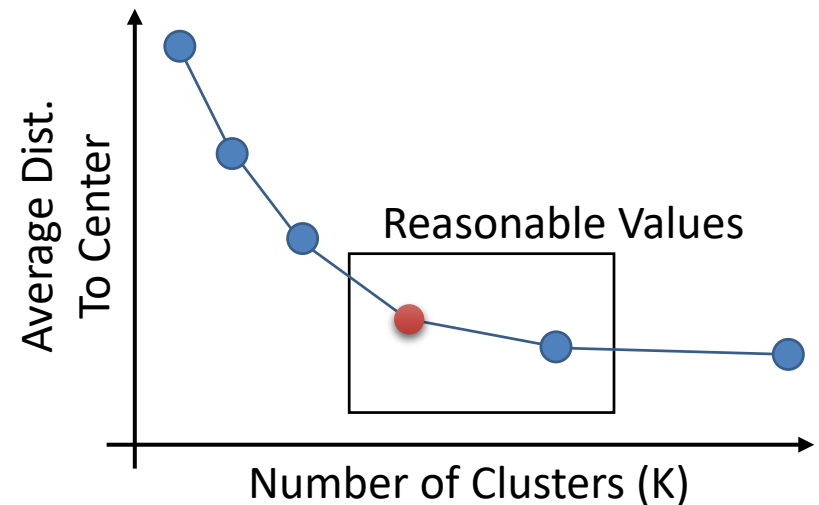


➤ Basic Elbow Method (Easy and what you do in HW)

- Try range of K-values and plot average distance to centers

➤ Cross-Validation (Better)

- Repeatedly split the data into training and validation datasets
- Cluster the training dataset
- Measure Avg. Dist. To Centers on validation data



K-Means +



How do we run k-means on the
data warehouse / data lake?

Interacting With the Data

Good for smaller datasets

- Faster more natural interaction
- Lots of tools!

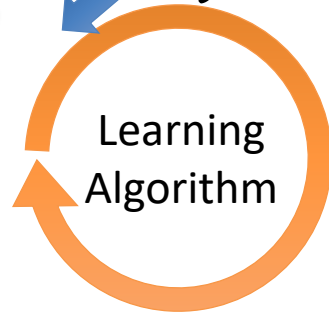
Compute
Locally

$$\Sigma = \bigoplus_{r \in \text{Data}} f_{\theta}(r)$$



Request Data Sample

Sample of Data



Can we send the
computation to
the data?

Yes!

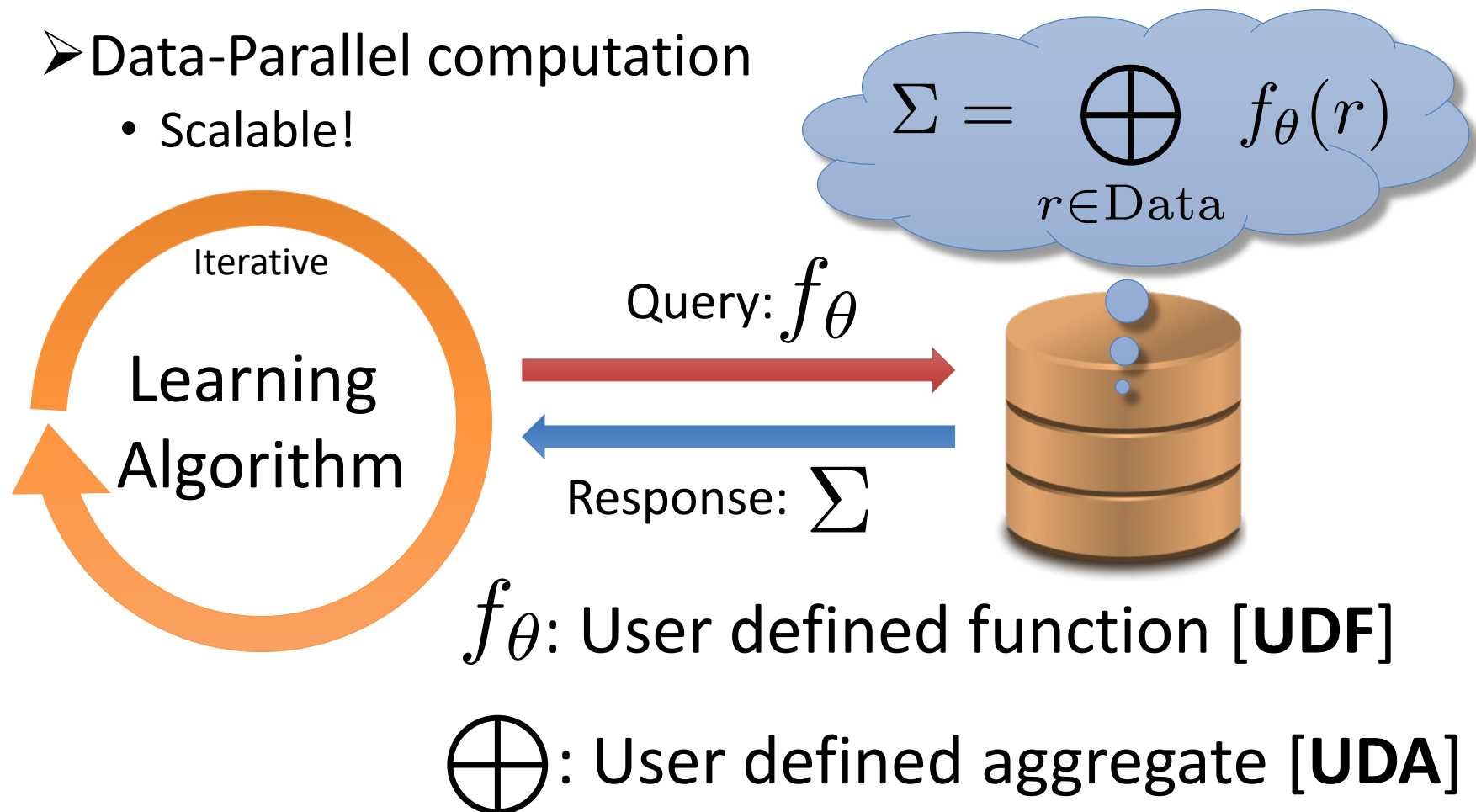
Computation



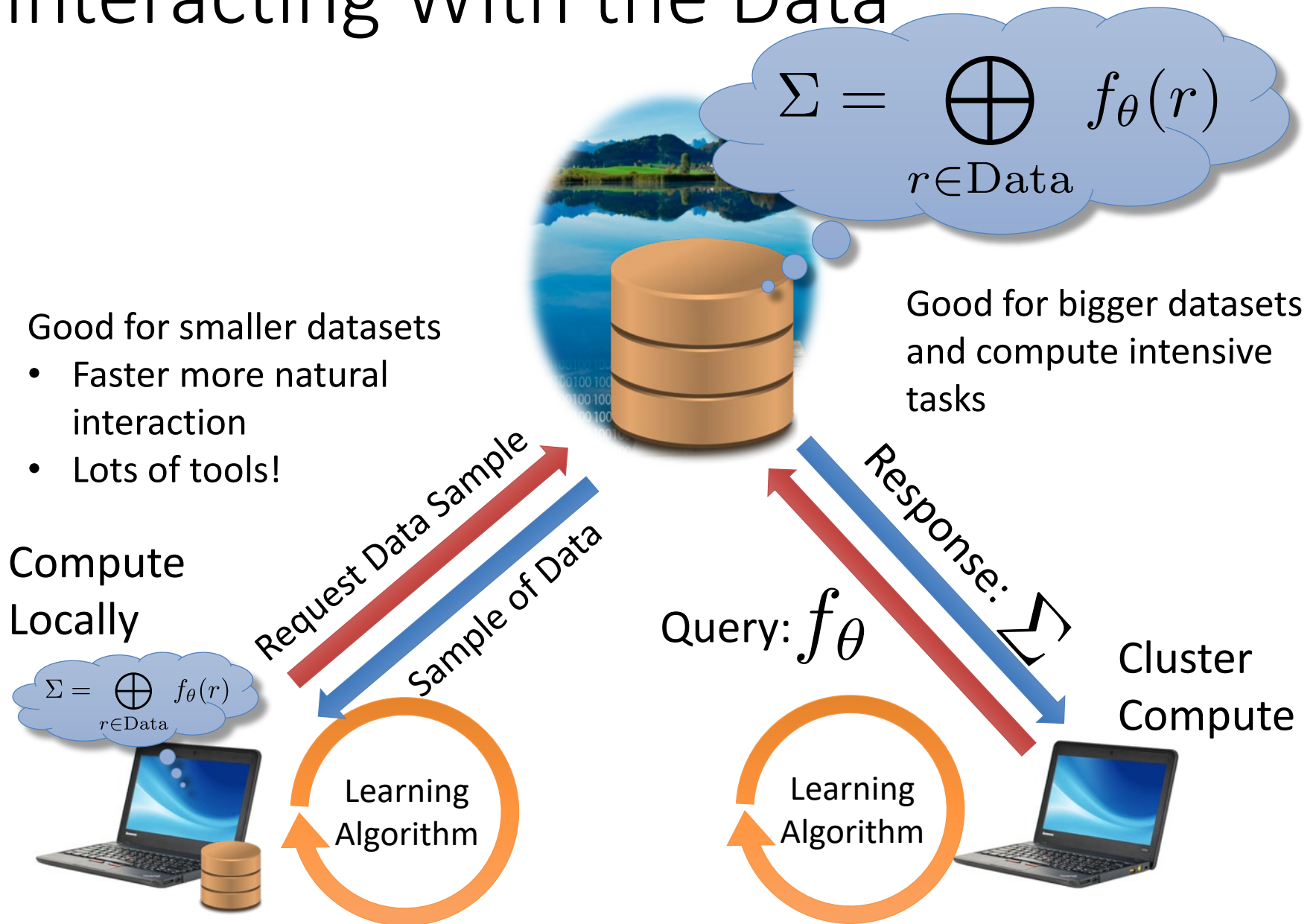
Statistical Query Pattern

Common Machine Learning Pattern

- Computing aggregates of user defined functions
- Data-Parallel computation
 - Scalable!



Interacting With the Data



Can we express K-Means in the Statistical Query Pattern?

```
centers ← pick k initial Centers
```

```
while (centers are changing):
```

```
    // Compute the assignments (E-Step)
```

```
    asg ← [(x, nearest(centers, x)) for x in data]
```

```
    for i in range(k): // Compute the new centers (M-Step)
```

```
        centers[i] = mean([x for (x, c) in asg if c == i])
```

Query returns all
the data ...

Merge with M-Step



Statistical Query
Pattern

```
centers ← pick k initial Centers
```

```
while (centers are changing):
```

```
    for i in range(k):
```

```
        new_centers[i] =
```

```
            mean([x for x in data if nearest(centers, x) == i])
```

```
    centers = new_centers
```


Can we express K-Means in the Statistical Query Pattern?

```
centers ← pick k initial Centers
```

```
while (centers are changing):
```

```
    for i in range(k):
```

```
        new_centers[i] =
```

```
            mean([x for x in data if nearest(centers, x) == i])
```

```
    centers = new_centers
```

Group by query:

```
SELECT nearest_UDF(centers, x) AS cid, mean_UDA(x)
FROM data GROUPBY cid
```

K-Means in Map-Reduce

➤ **MapFunction**(*old_centers*, *x*)

- Compute the index of the nearest old center
- Return (**key** = *nearest_centers*, **value** = (*x*, 1))

➤ **ReduceFunction** combines values and counts

- For each cluster center (Group By)

➤ Data system returns aggregate statistics:

$$s_i = \sum_{x \in \text{Cluster } i} x_i \quad \text{and} \quad n_i = \sum_{x \in \text{Cluster } i} 1$$

➤ ML algorithm computes new centers: $\mu_i = s_i / n_i$

Can we express K-Means++ in the Statistical Query Pattern?

➤ Yes, however there is a better version: **K-Means||**

- More complex but much faster

➤ Or you can parallelize **K-Means++** directly

- Requires more passes

➤ Challenging Step?

- Parallel weighted sampling:

```
sample_one(data, prob = dSq / sum(dSq))
```

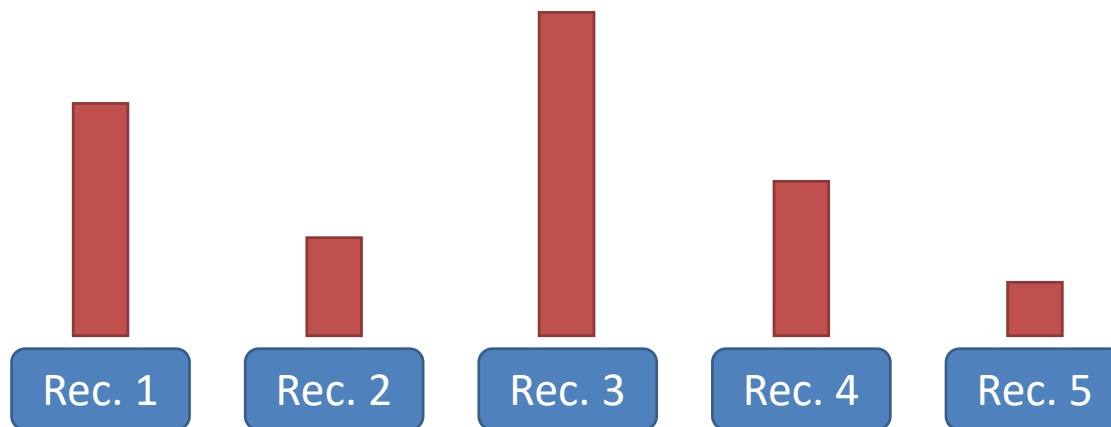
- How do you select one point uniformly at random?

Res-A: weighted reservoir sampling

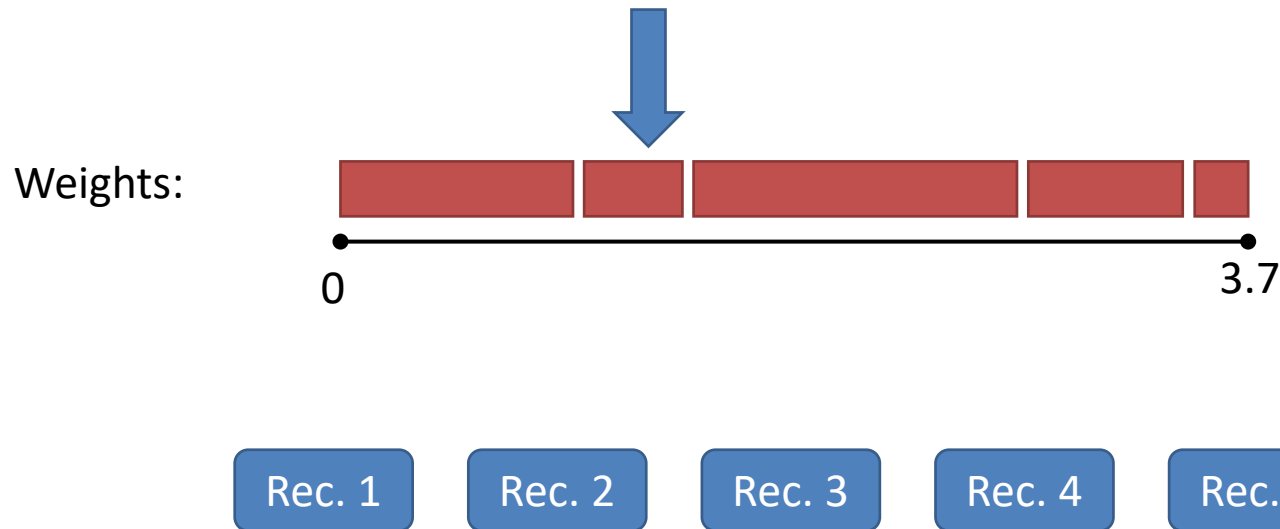
- **Goal:** *Sample k records from a stream where record i is included in the sample with probability proportional to w_i*

How would we normally sample k records?

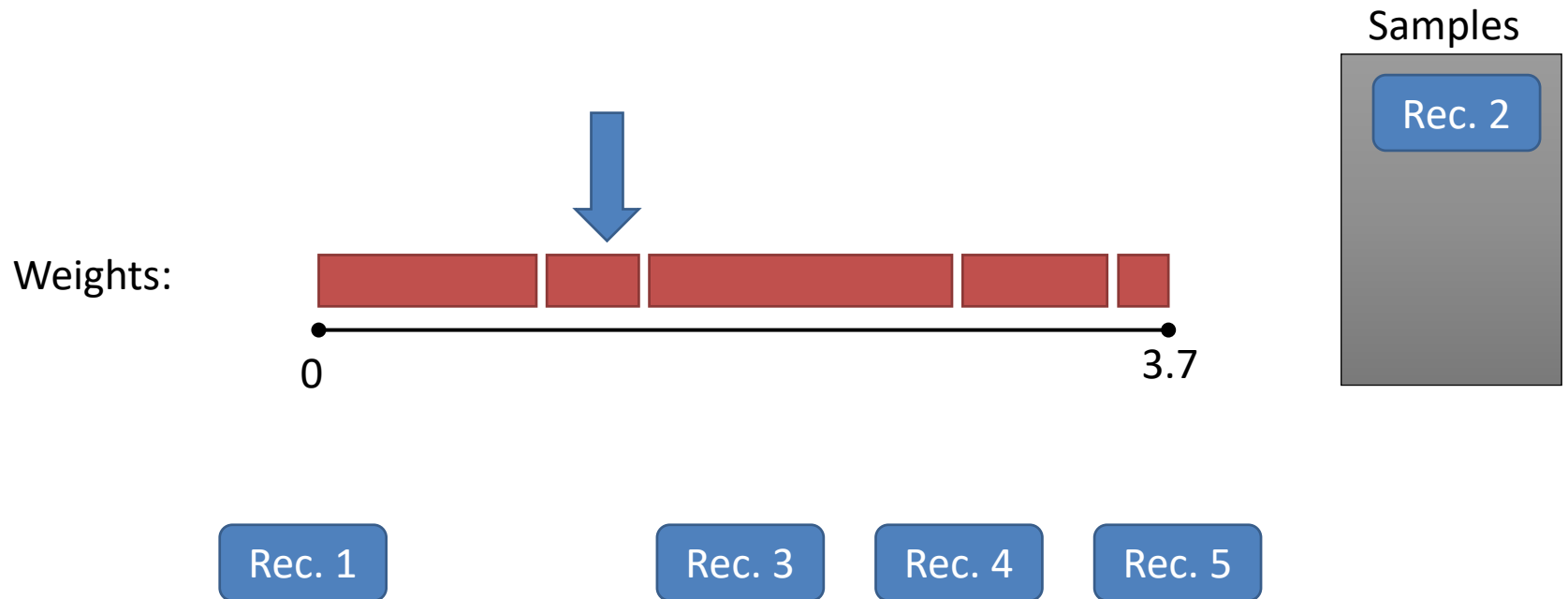
Weights:



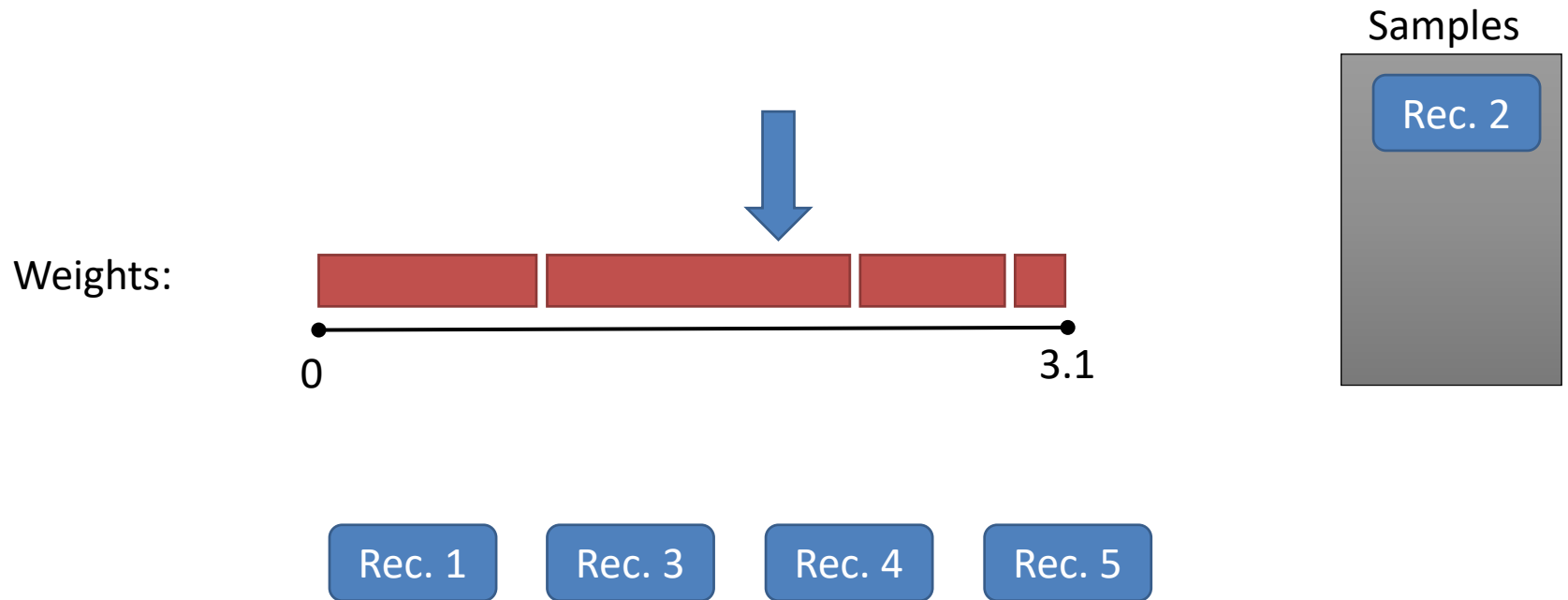
Draw a random number uniformly between **0** and **3.7**



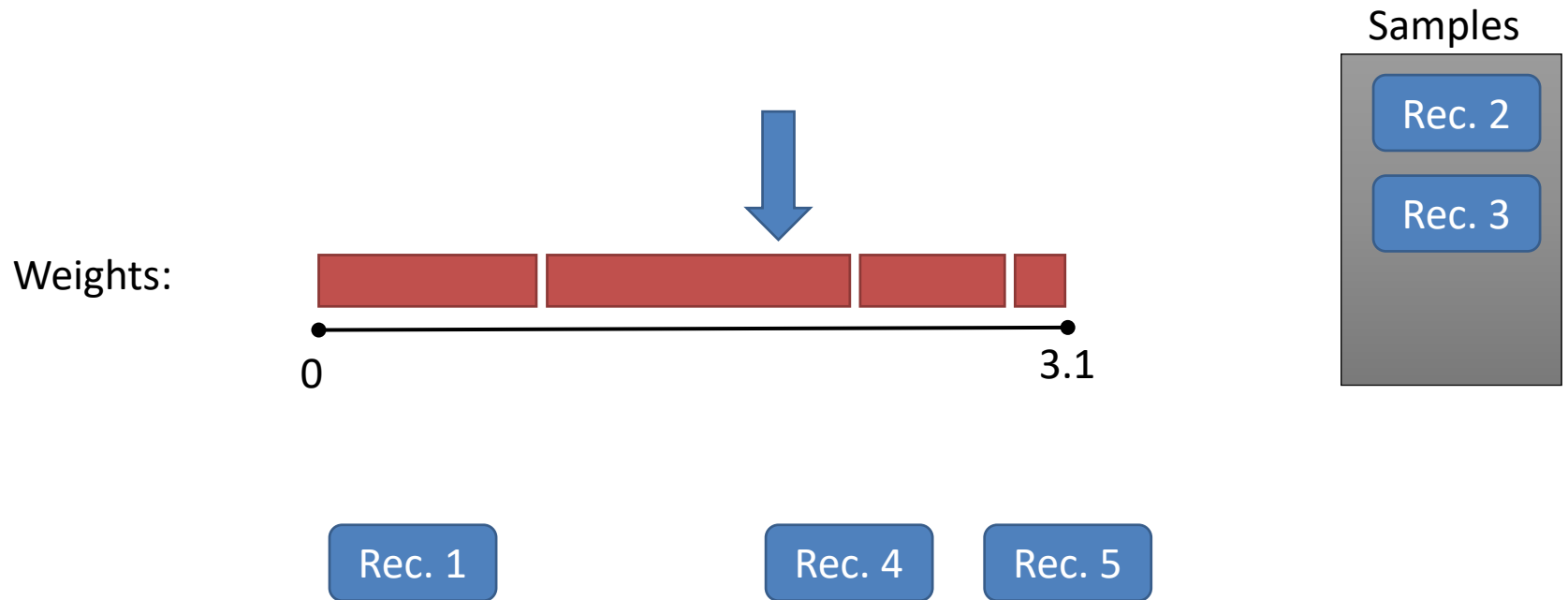
Sample the corresponding record and remove the weight.



Draw a random number uniformly between **0** and **3.1**



We want to do this in **one pass** *without* ever knowing the **sum** of the weights!



Res-A: weighted reservoir sampling

➤ **Goal:** *Sample k records from a stream where record i is included in the sample with probability proportional to w_i*

➤ **Algorithm:**

- For each record i draw a uniform random number:

$$u_i \sim \text{Unif}(0, 1)$$

- Select the top- k records ordered by: u_i^{1/w_i}

➤ **Common ML Pattern?**

- **Query Function:** $[pow(rand(), 1 / record.w), record]$
- **Agg. Function:** *top-k heap*

Illustrating Res-A Algorithm

Weights:		Uniform Random Number	u_i^{1/w_i}
1.0	Rec. 1	0.31	0.31
2.0	Rec. 2	0.20	0.45
1.4	Rec. 3	0.45	0.56
2.5	Rec. 4	0.14	0.46
0.7	Rec. 5	0.55	0.43
3.0	Rec. 6	0.69	0.88

Top 2

The diagram illustrates the Res-A algorithm. It shows a table with four columns: 'Weights:', 'Recommendations', 'Uniform Random Number', and the transformed value u_i^{1/w_i} . The weights are 1.0, 2.0, 1.4, 2.5, 0.7, and 3.0. The recommendations are Rec. 1 through Rec. 6. The uniform random numbers are 0.31, 0.20, 0.45, 0.14, 0.55, and 0.69. The transformed values are 0.31, 0.45, 0.56, 0.46, 0.43, and 0.88. An arrow points to the top two values, 0.56 and 0.88, which are labeled 'Top 2'.

Basic Analysis Behind Res-A

➤ Define the random variable: $X_i = u_i^{1/w_i}$

➤ Then:

$$\mathbf{P}(X_i < \alpha) = \mathbf{P}(u_i^{1/w_i} < \alpha) = \mathbf{P}(u_i < \alpha^{w_i}) = \alpha^{w_i}$$

$$\mathbf{p}(X_i = \alpha) = w_i \alpha^{w_i - 1}$$

Derivative of CDF → PDF

➤ Suppose we want to pick just one element (k=1)

- Probability of selecting X_i is:

$$\begin{aligned} \int_0^1 \mathbf{p}(X_i = \alpha) \prod_{j \neq i} \mathbf{P}(X_j < \alpha) d\alpha &= \int_0^1 (w_i \alpha^{w_i - 1}) \prod_{j \neq i} \alpha^{w_j} d\alpha \\ &= \frac{w_i}{\sum_j w_j} \end{aligned}$$

We won't test you
on this derivation

People who like Res-A also like...

➤ Algorithm R

- Another reservoir filtering algorithm (recitation?)

➤ Bloom Filters

- Efficient set membership with limited memory

➤ Count-Min

- Efficient key-counting with limited memory

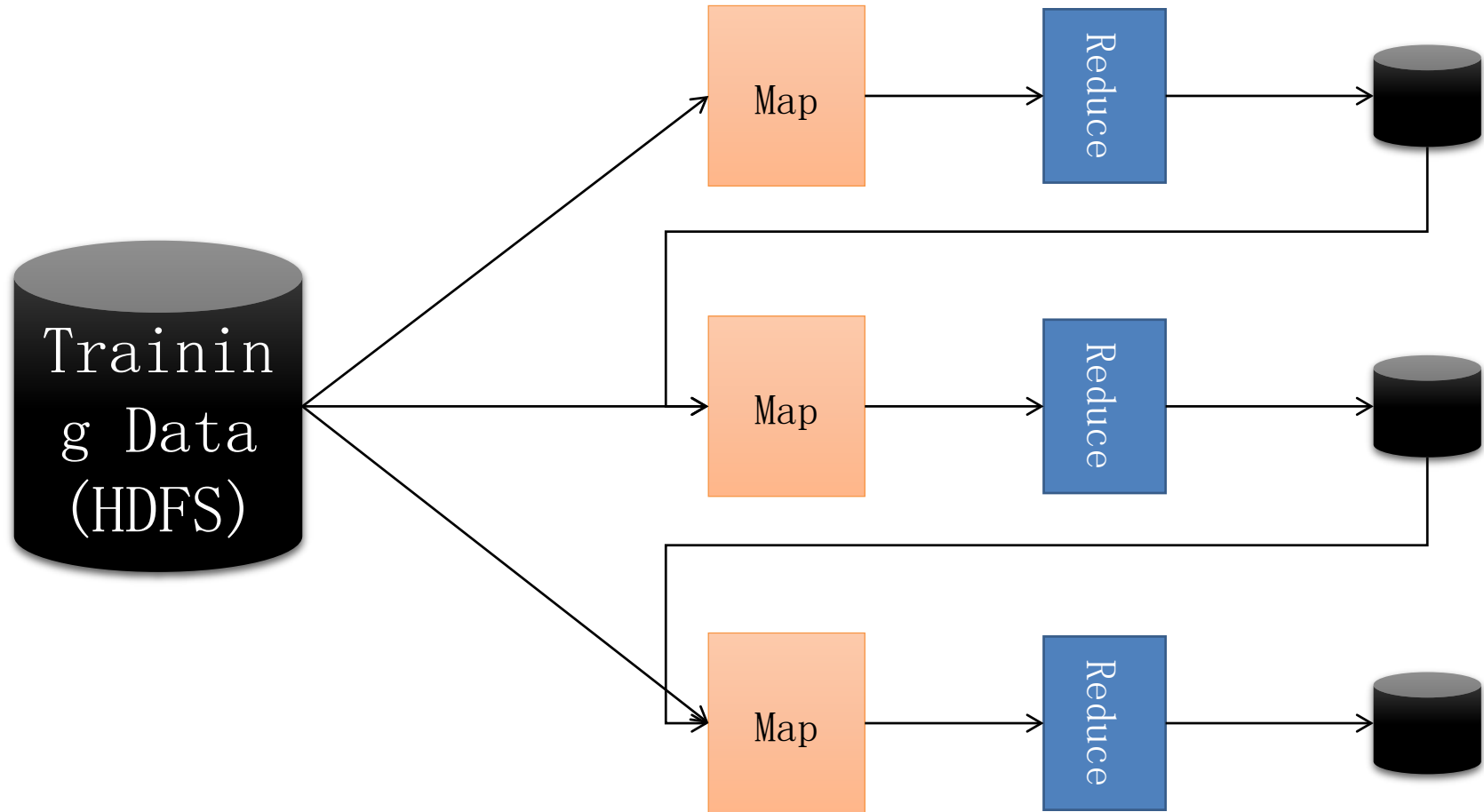
➤ Heavy Hitters Sketch

- Top-k Elements with limited memory

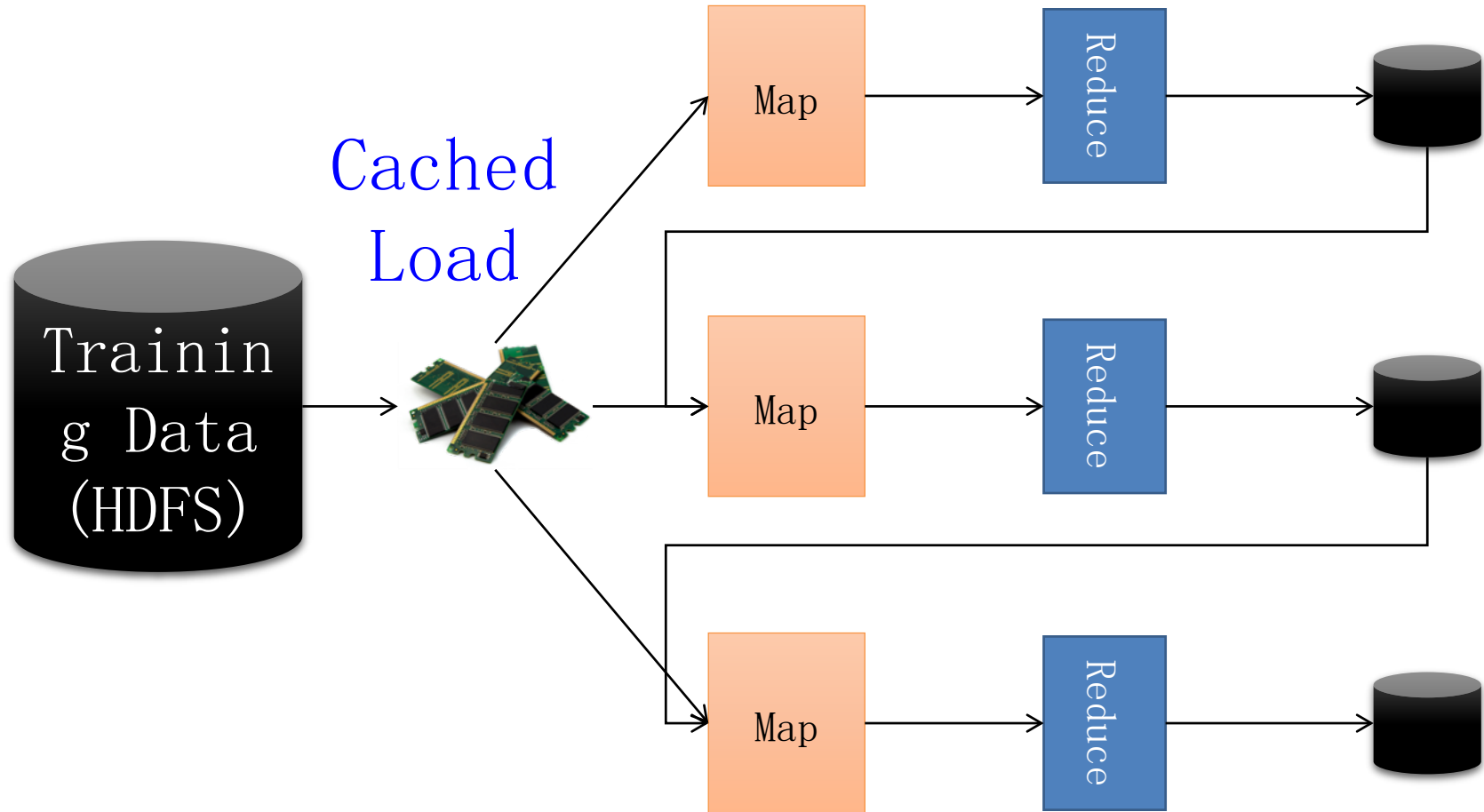
Implementation Details: Statistical Query Pattern

- **Iterative ML** ➔ Data caching is important
 - Motivation behind Spark project

Map Reduce Dataflow View

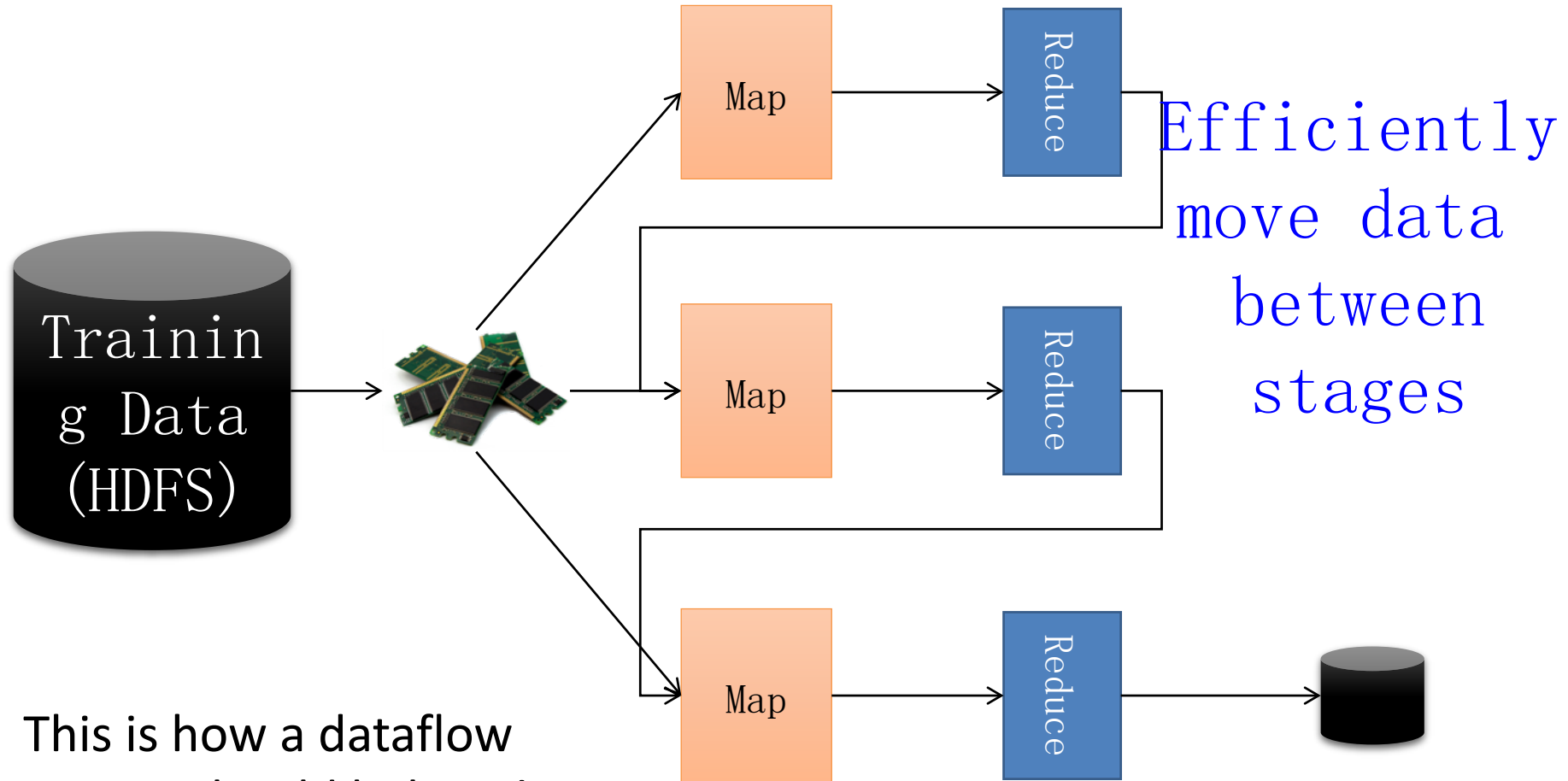


Spark Opt. Dataflow



10–100× faster than network and disk 52

Spark Opt. Dataflow View

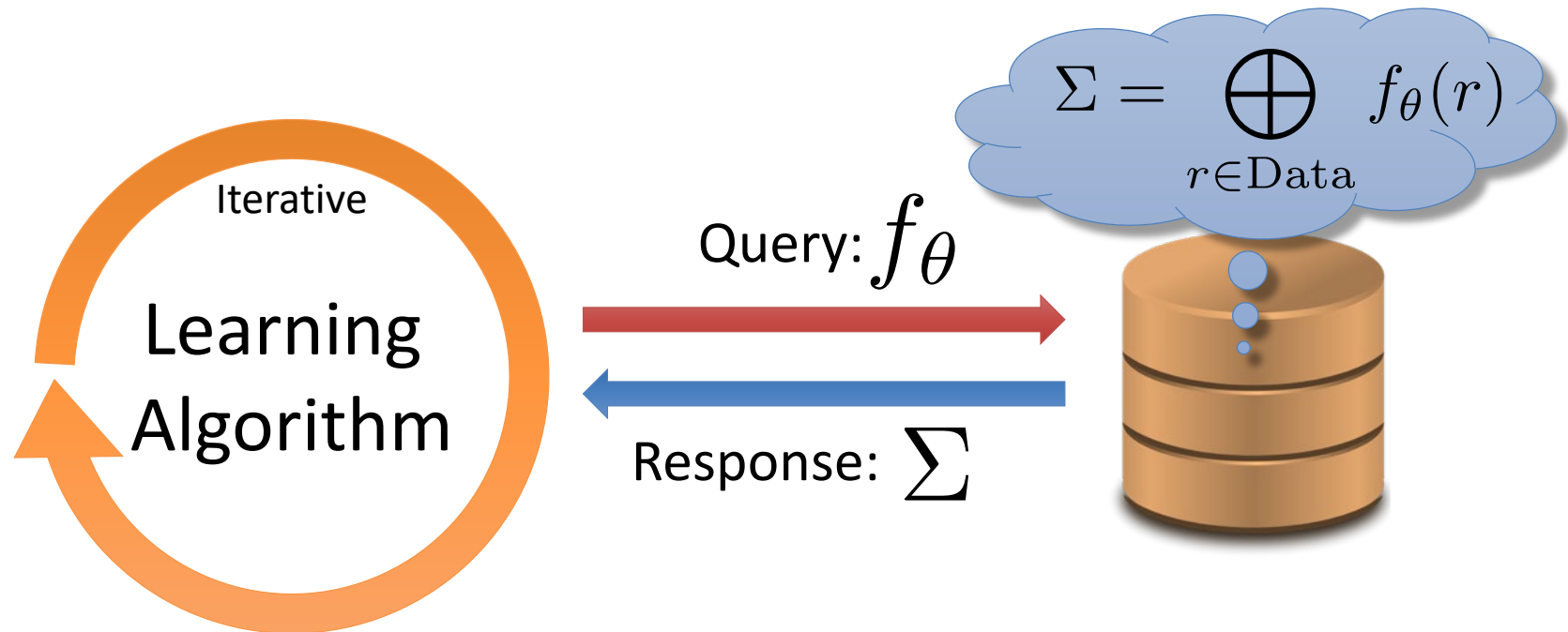


This is how a dataflow system should behave!

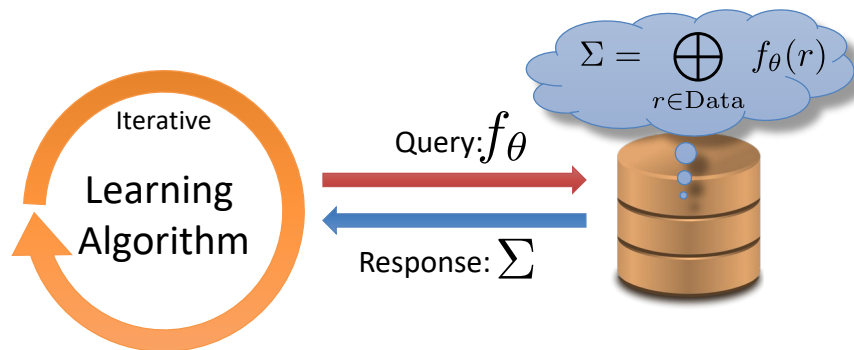
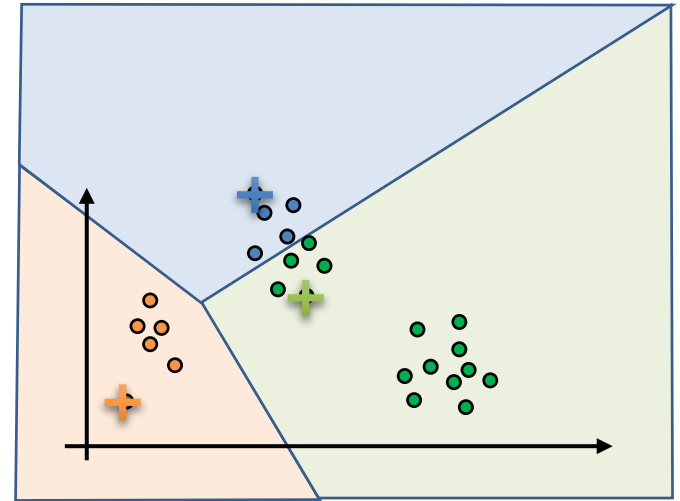
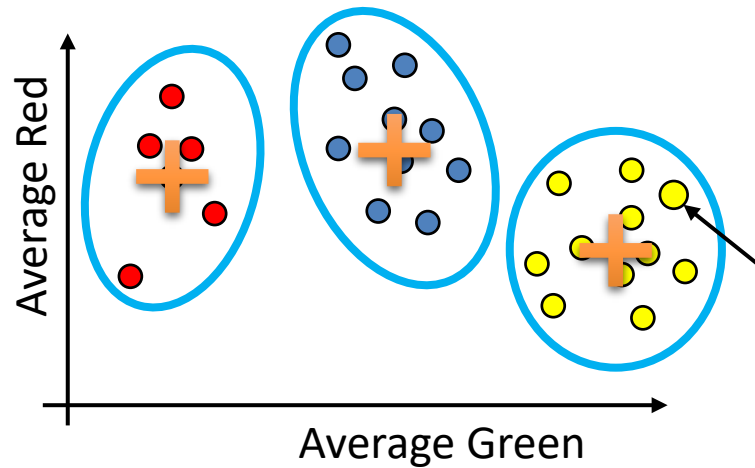
- What happened to map-reduce?

Implementation Details: Common Machine Learning Pattern

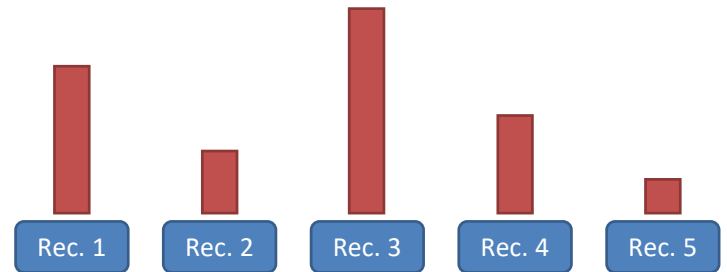
- Iterative ML → Data caching is important
 - Motivation behind Spark project
- Need to watch out for large θ and Σ



Summary of Clustering



```
SELECT nearest_UDF(centers, x) AS cid, mean_UDA(x)
FROM data GROUPBY cid
```





Taxonomy of Machine Learning

