Clustering

module 12

Clustering

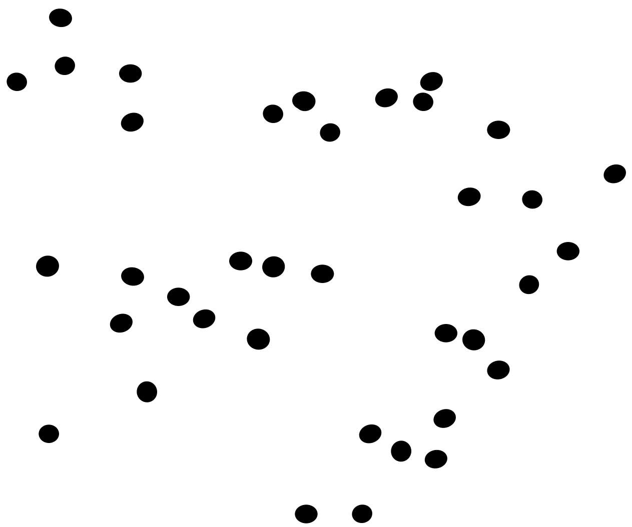
- Goal: Partition rows of a table into clusters, in order to:
  - Minimize difference within-cluster
  - Maximize difference extra-cluster
- Applications:
  - Find groups of similar customers
  - Find patterns in purchases
- We will see how to use it to explore data
- Many clustering techniques:
  - K-Means
  - Agglomerative clustering
  - DBScan
  - …

Agglomerative clustering

The Agglomerative Clustering Steps

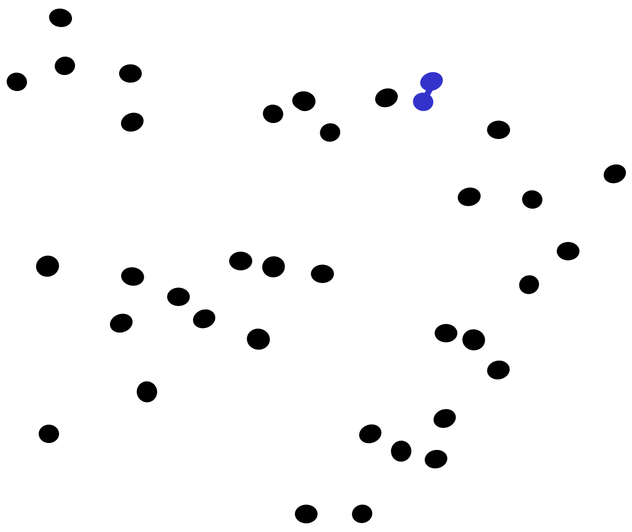
1. Start with *n* clusters (each record is its own cluster)
2. Merge two closest records into one cluster
3. At each successive step, the two clusters closest to each other are merged
4. Finish when the desired number of clusters is reached

Agglomerative Clustering

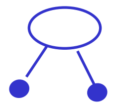


1. Say "Every point is its own cluster"

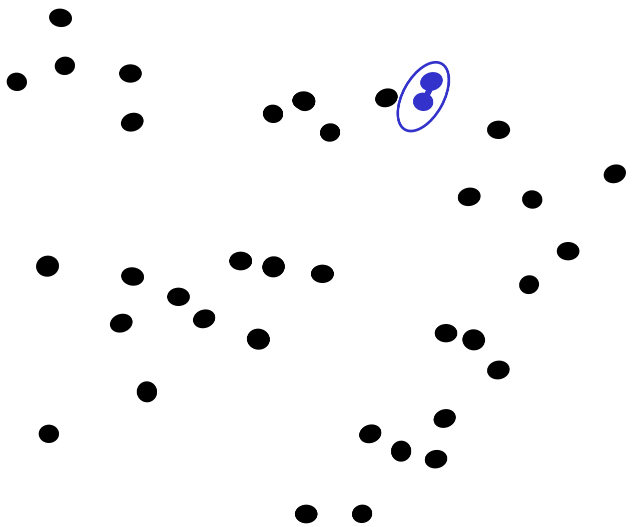
Agglomerative Clustering



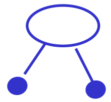
1. Say "Every point is its own cluster"
2. Find "most similar" pair of clusters



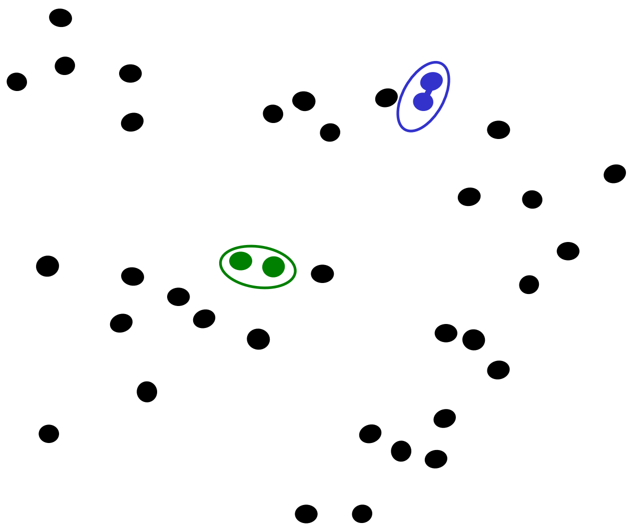
Agglomerative Clustering



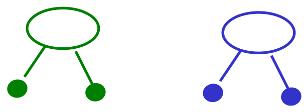
1. Say "Every point is its own cluster"
2. Find "most similar" pair of clusters
3. Merge it into a parent cluster



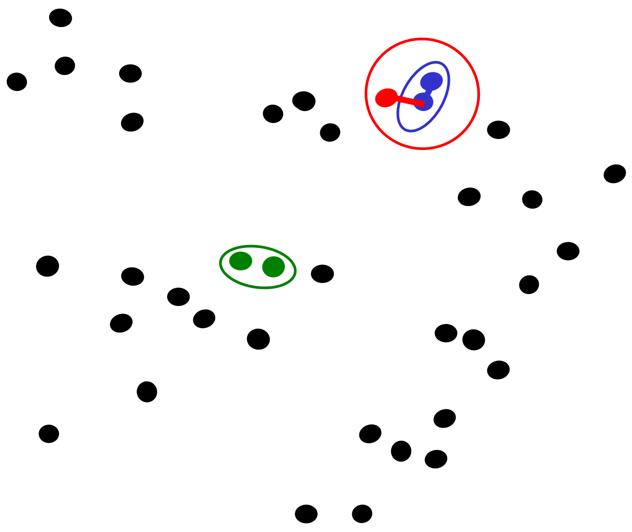
Agglomerative Clustering



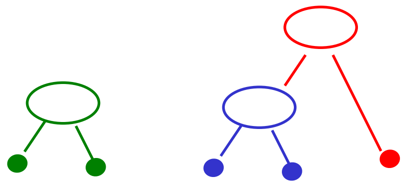
1. Say "Every point is its own cluster"
2. Find "most similar" pair of clusters
3. Merge it into a parent cluster
4. Repeat



Agglomerative Clustering



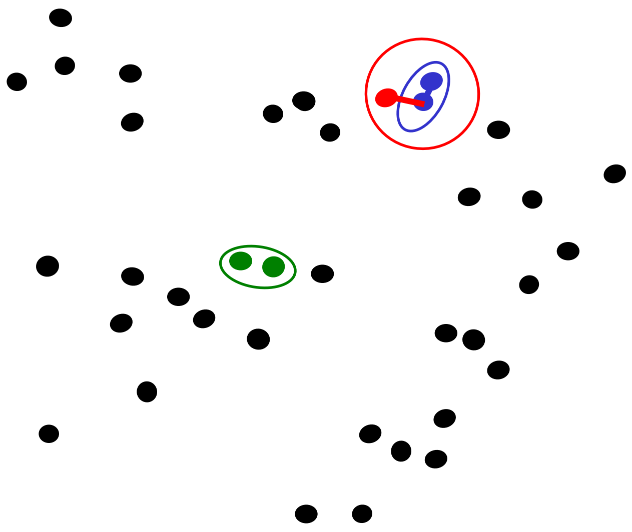
1. Say "Every point is its own cluster"
2. Find "most similar" pair of clusters
3. Merge it into a parent cluster
4. Repeat



Agglomerative Clustering

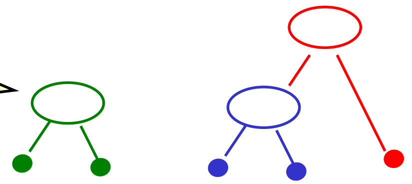
Based **on linkage**:
- Single linkage
- Complete linkage
- Average linkage

And **on distance:**
- Euclidean
- Manhattan
- Hamming
- …

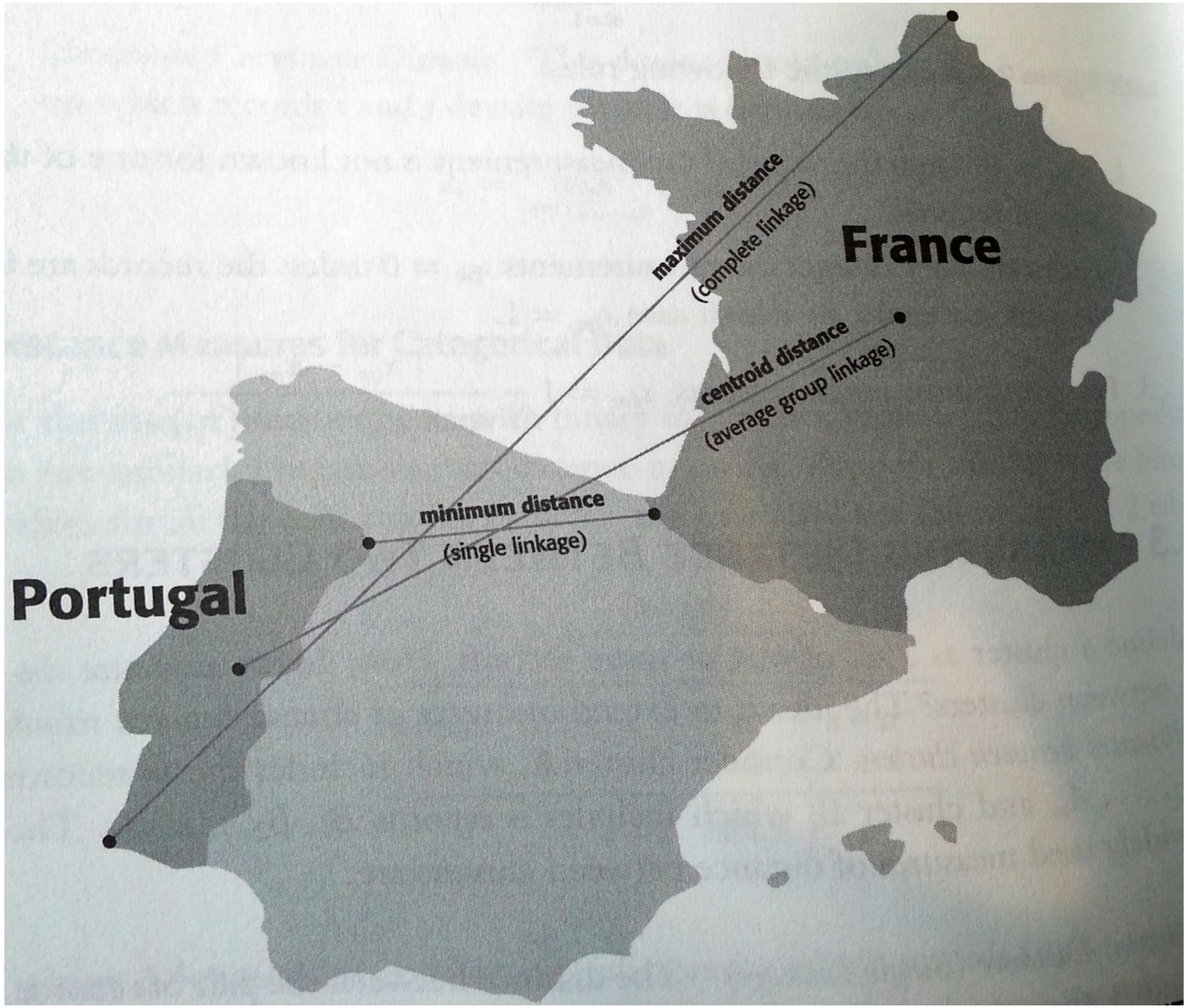


1. Say "Every point is its own cluster"
2. Find "most similar" pair of clusters
3. Merge it into a parent cluster
4. Repeat…until you've merged the whole dataset into one cluster

The algorithm will stop when it reaches the desired number of clusters



Linkage



France
Portugal
maximum distance
(complete linkage)
centroid distance
(average group linkage)
minimum distance
(single linkage)

K-Means

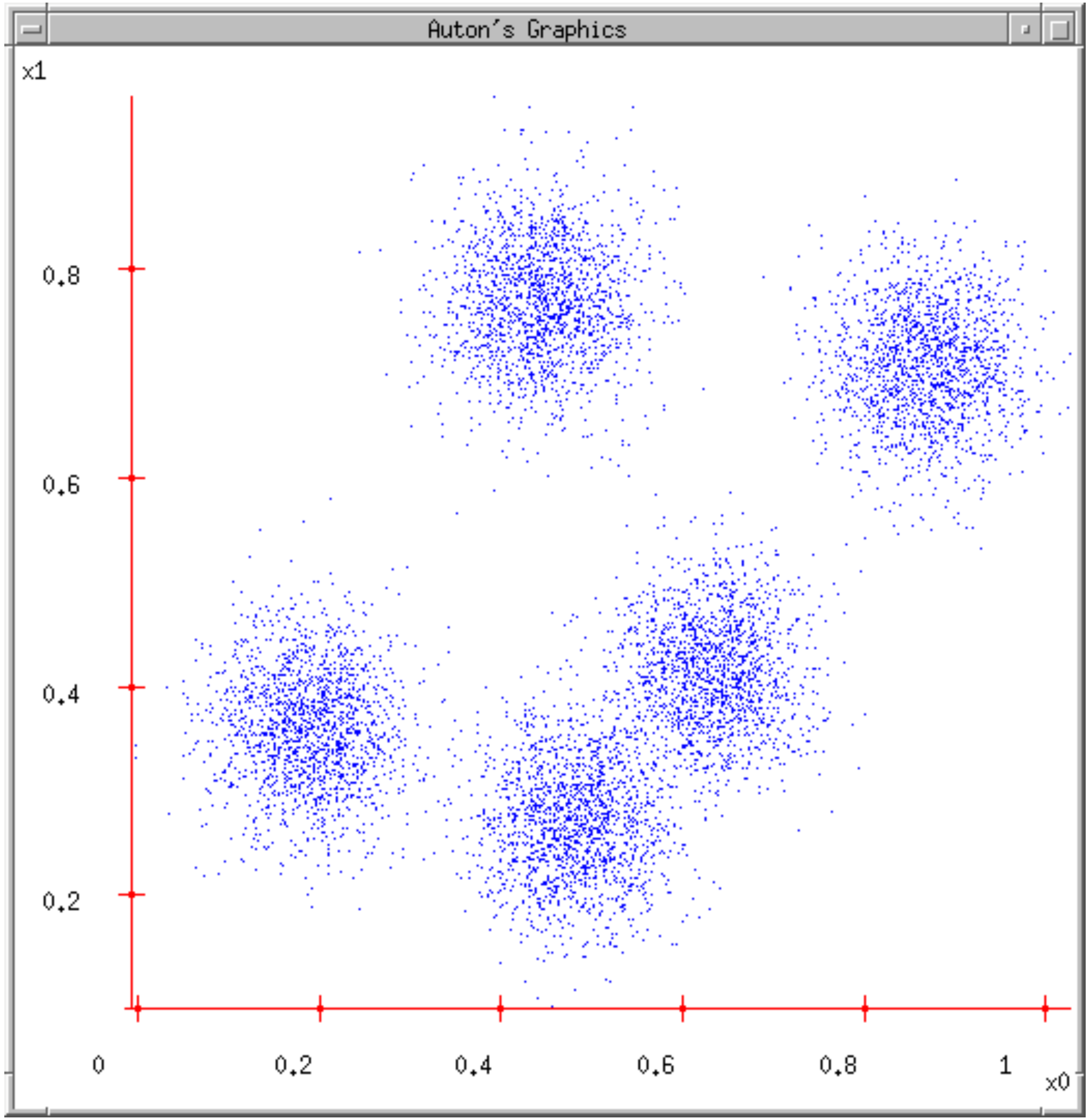
K-Means Clustering Algorithm

1. Choose # of clusters desired, *k*
2. Start with a partition into k clusters

   Often based on random selection of k centroids
3. At each step, move each record to cluster with closest centroid
4. Recompute centroids, repeat step 3
5. Stop when moving records increases within-cluster dispersion

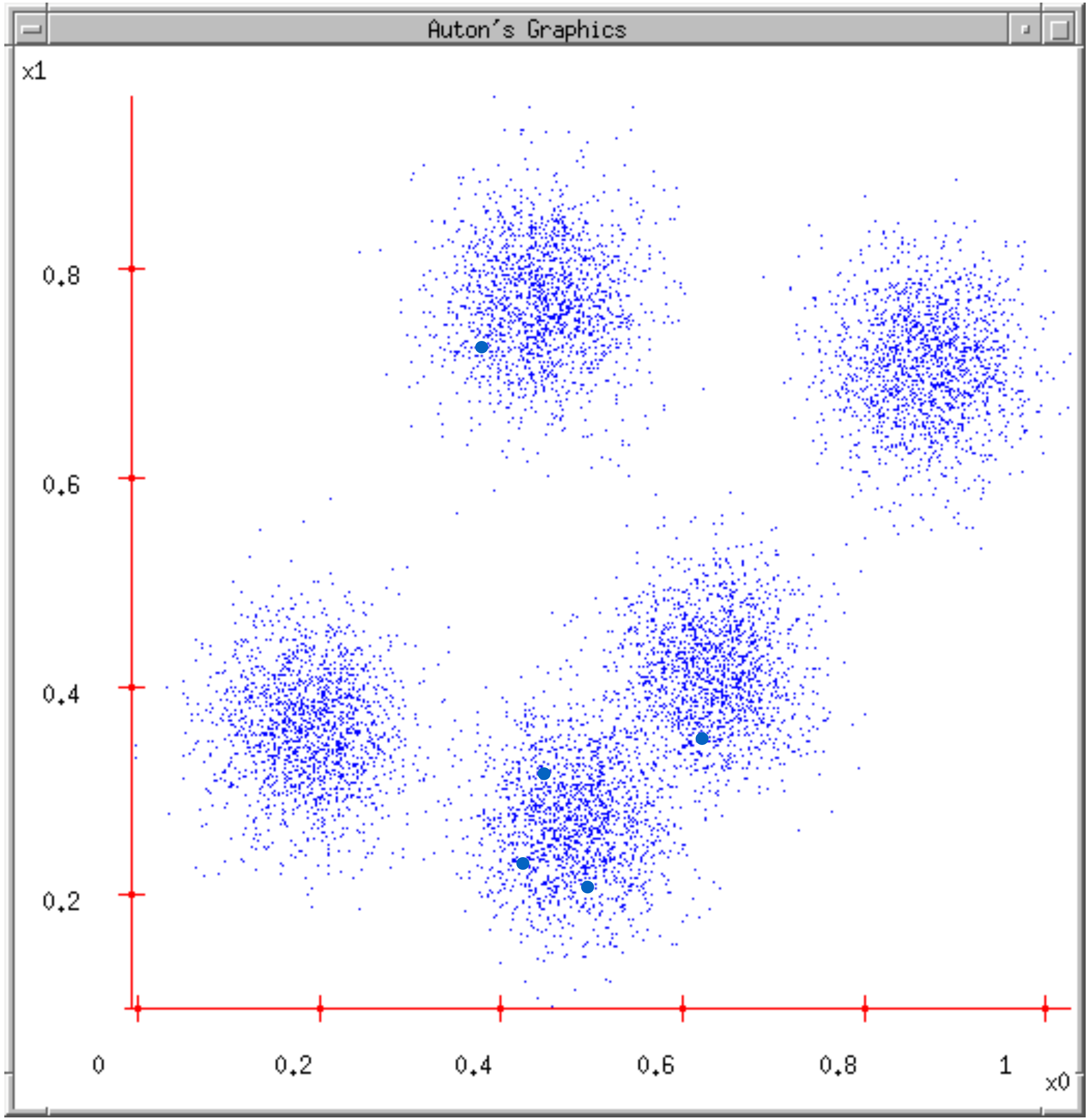
K-means

1. Ask user how many clusters they'd like. *(e.g. k=5)*



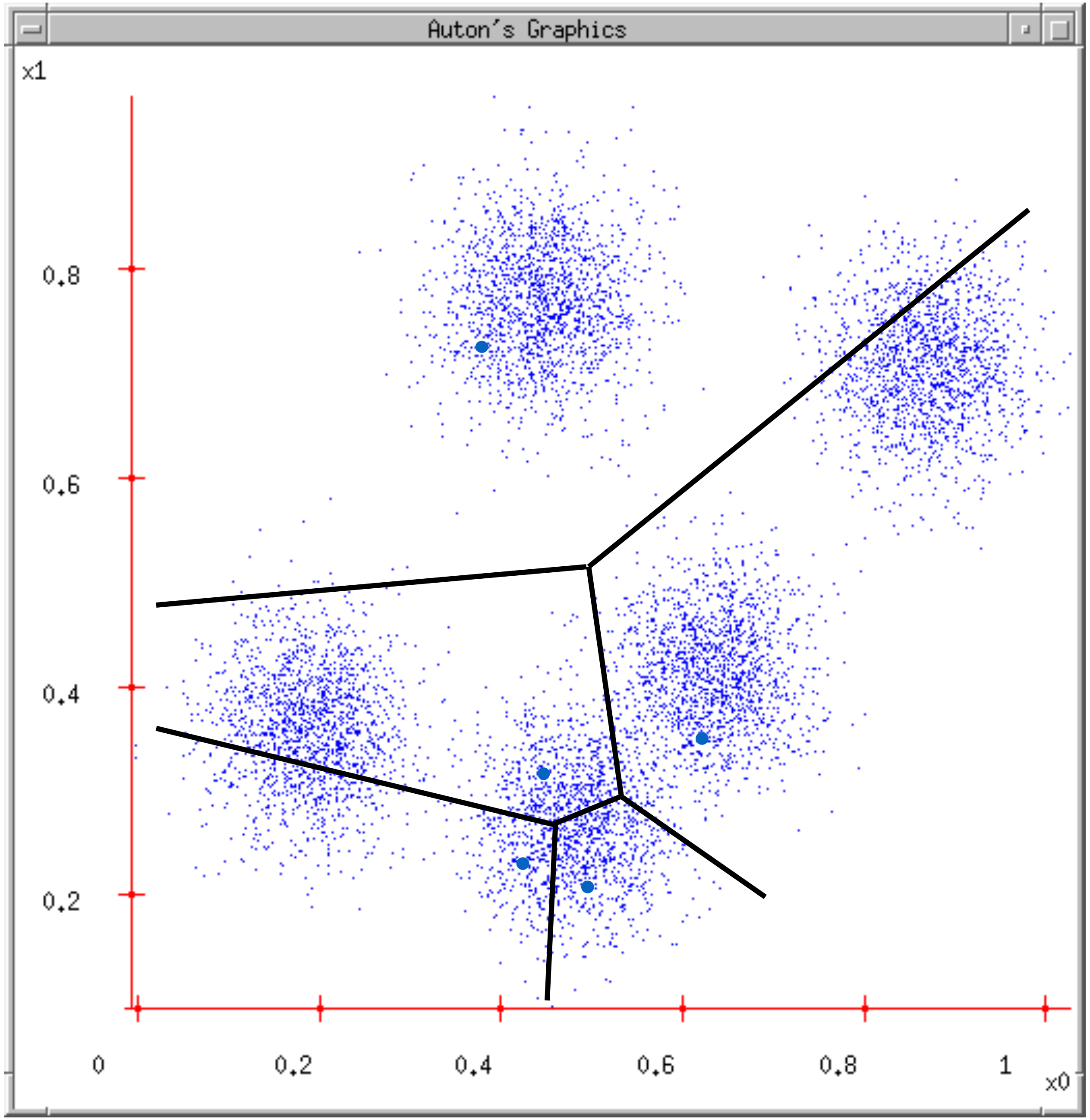
K-means

1. Ask user how many clusters they'd like. *(e.g. k=5)*
2. Randomly guess k cluster Center locations



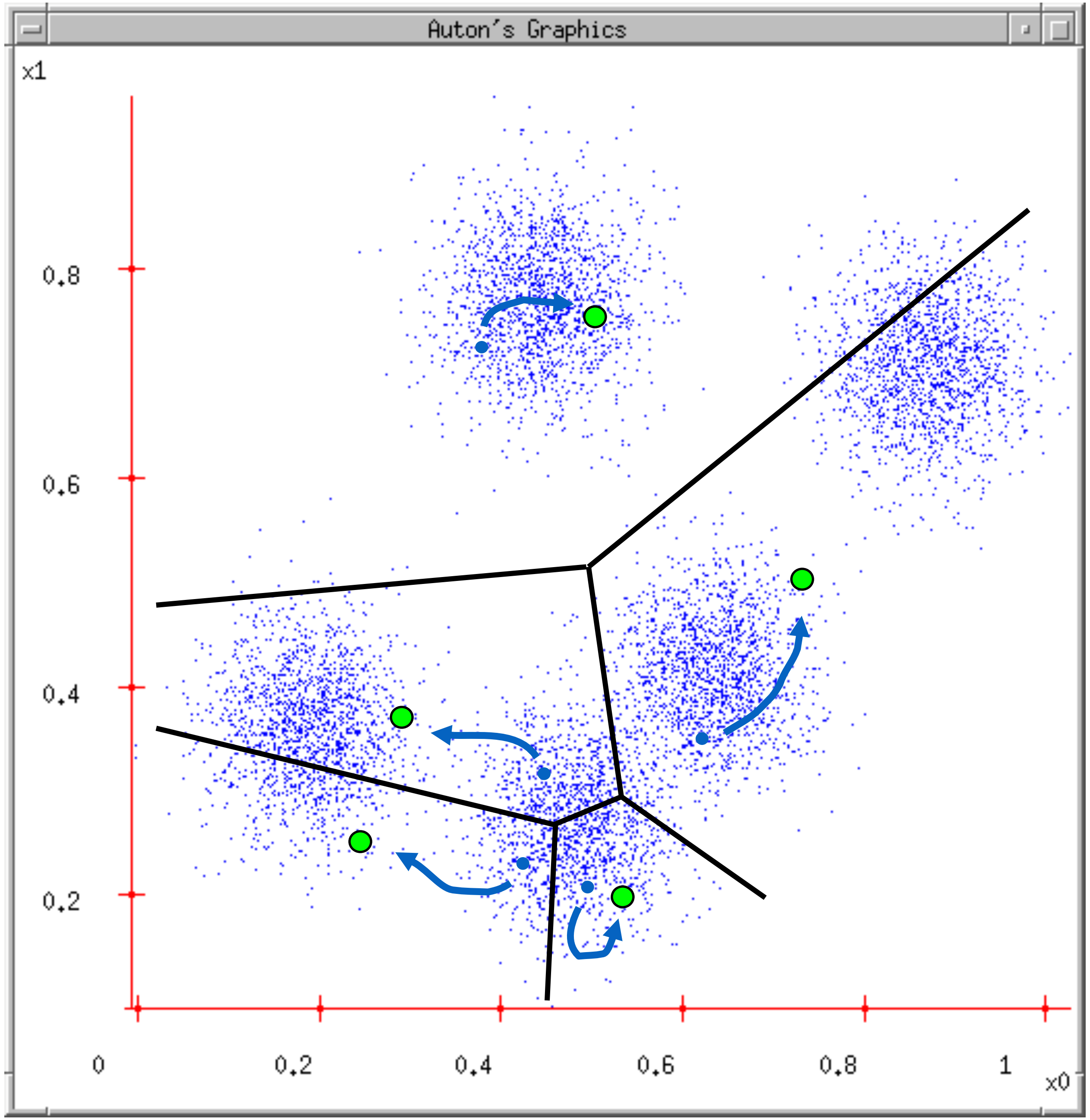
K-means

1. Ask user how many clusters they'd like. *(e.g. k=5)*
2. Randomly guess k cluster Center locations
3. Each datapoint (blue) finds out which Center it's closest to. (Thus each Center "owns" a set of datapoints). Black lines are the ownership boundaries of each centroid



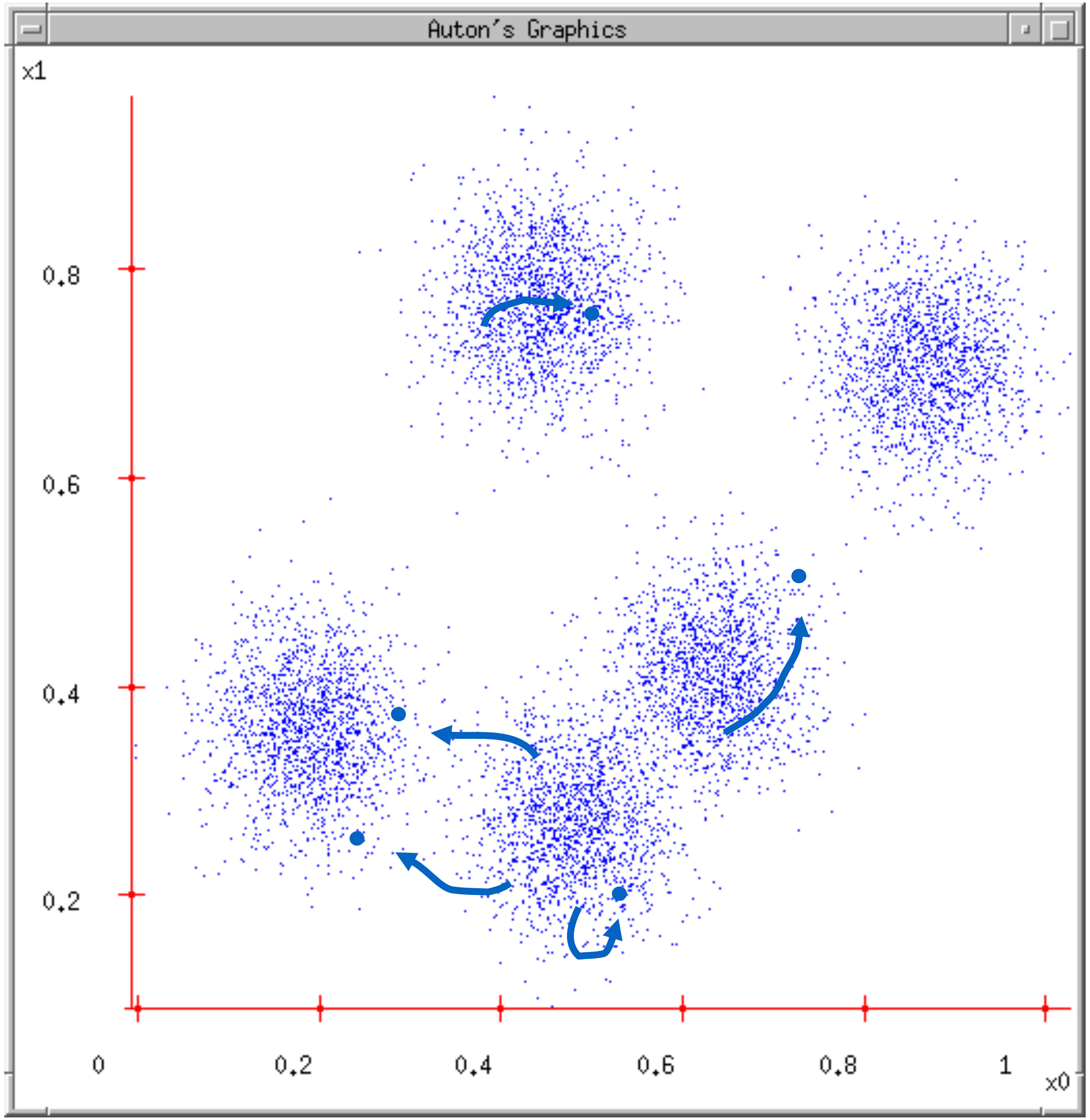
K-means

1. Ask user how many clusters they'd like. *(e.g. k=5)*
2. Randomly guess k cluster Center locations
3. Each datapoint finds out which Center it's closest to.
4. Each Center finds the centroid of the points it owns



K-means

1. Ask user how many clusters they'd like. *(e.g. k=5)*
2. Randomly guess k cluster Center locations
3. Each datapoint finds out which Center it's closest to.
4. Each Center finds the centroid of the points it owns…
5. …and jumps there
6. …Repeat until terminated!

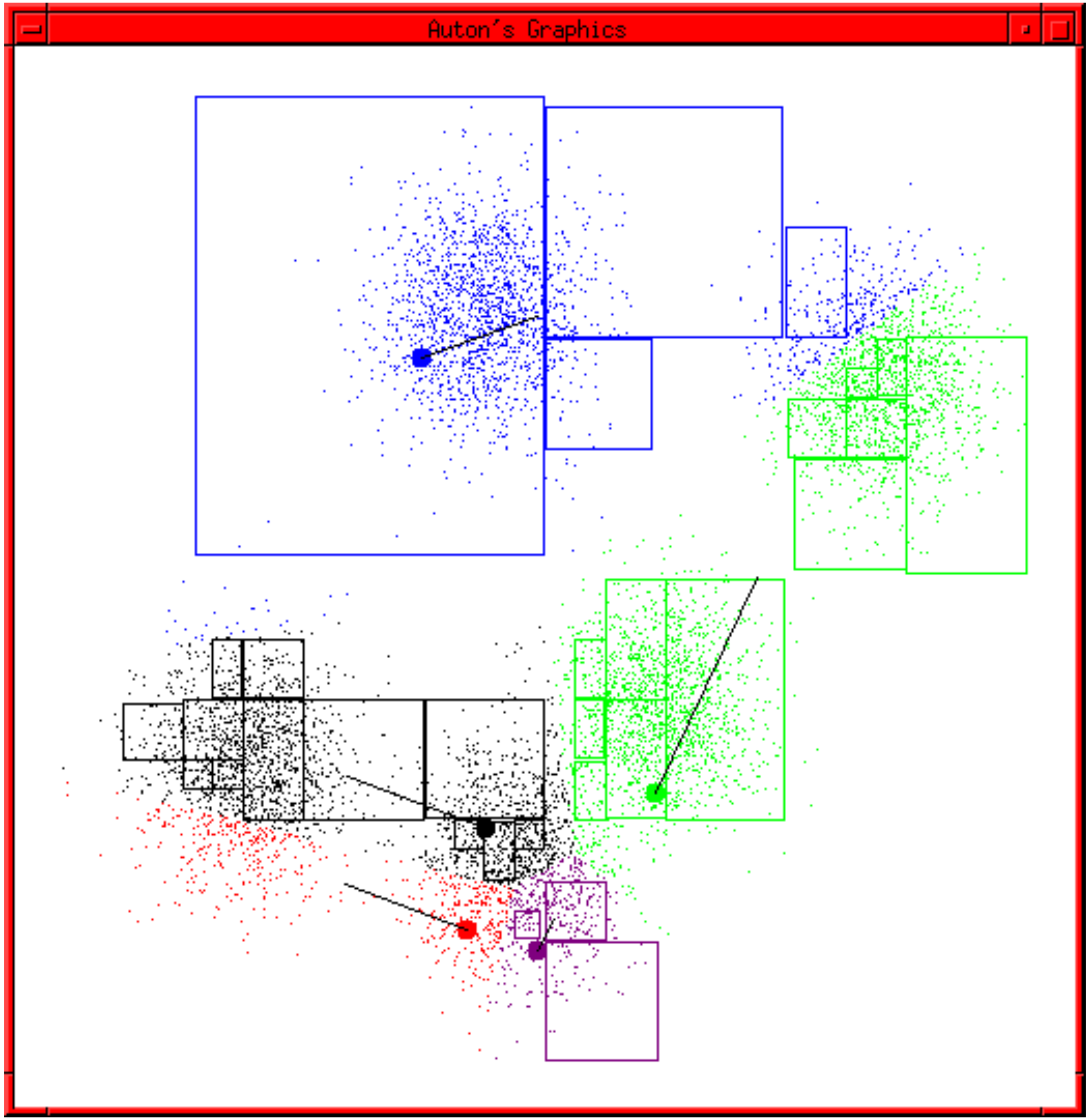


K-means Start

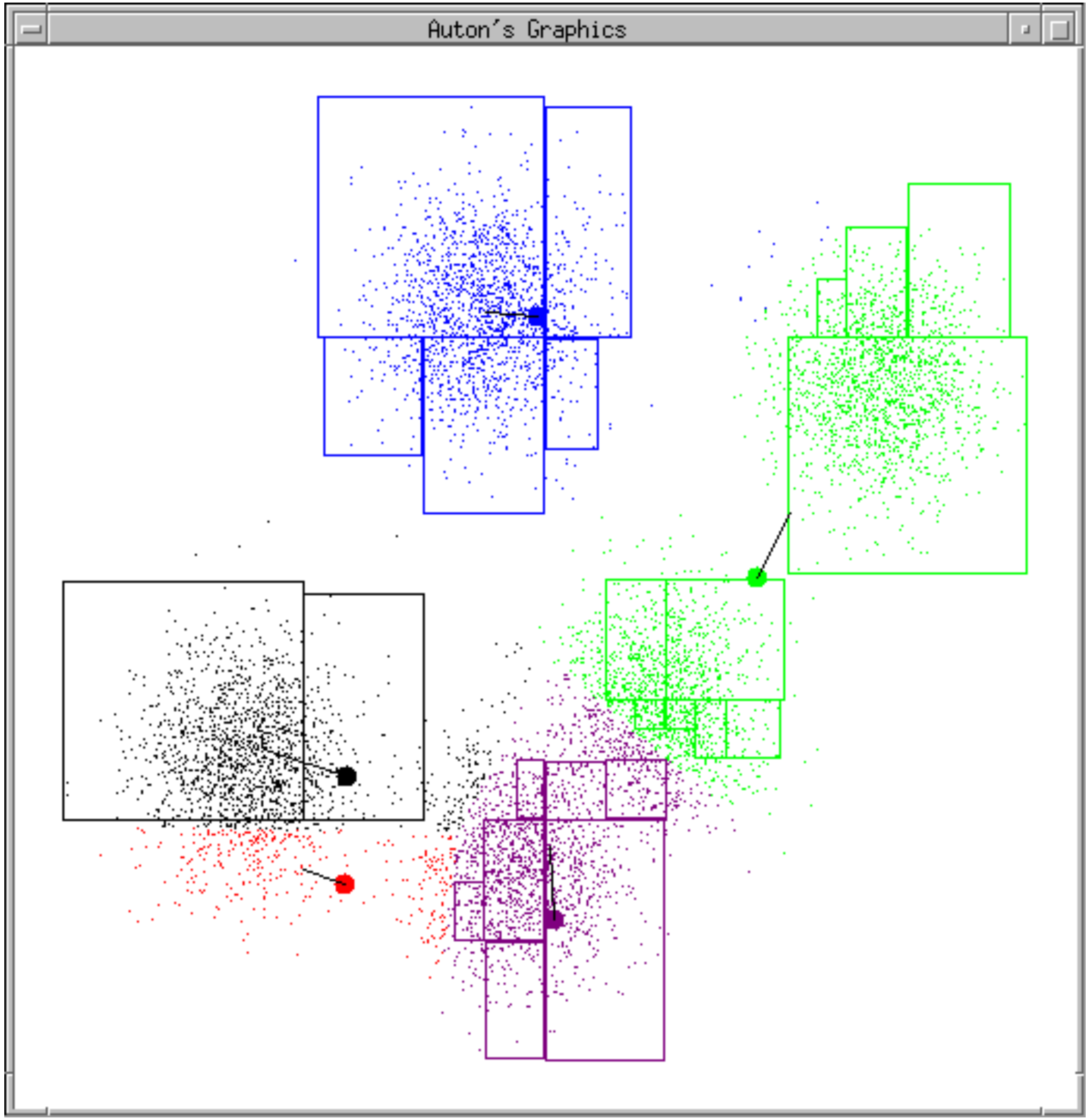
Advance apologies: in Black and White this example will deteriorate

Example generated by Dan Pelleg's super-duper fast K-means system:

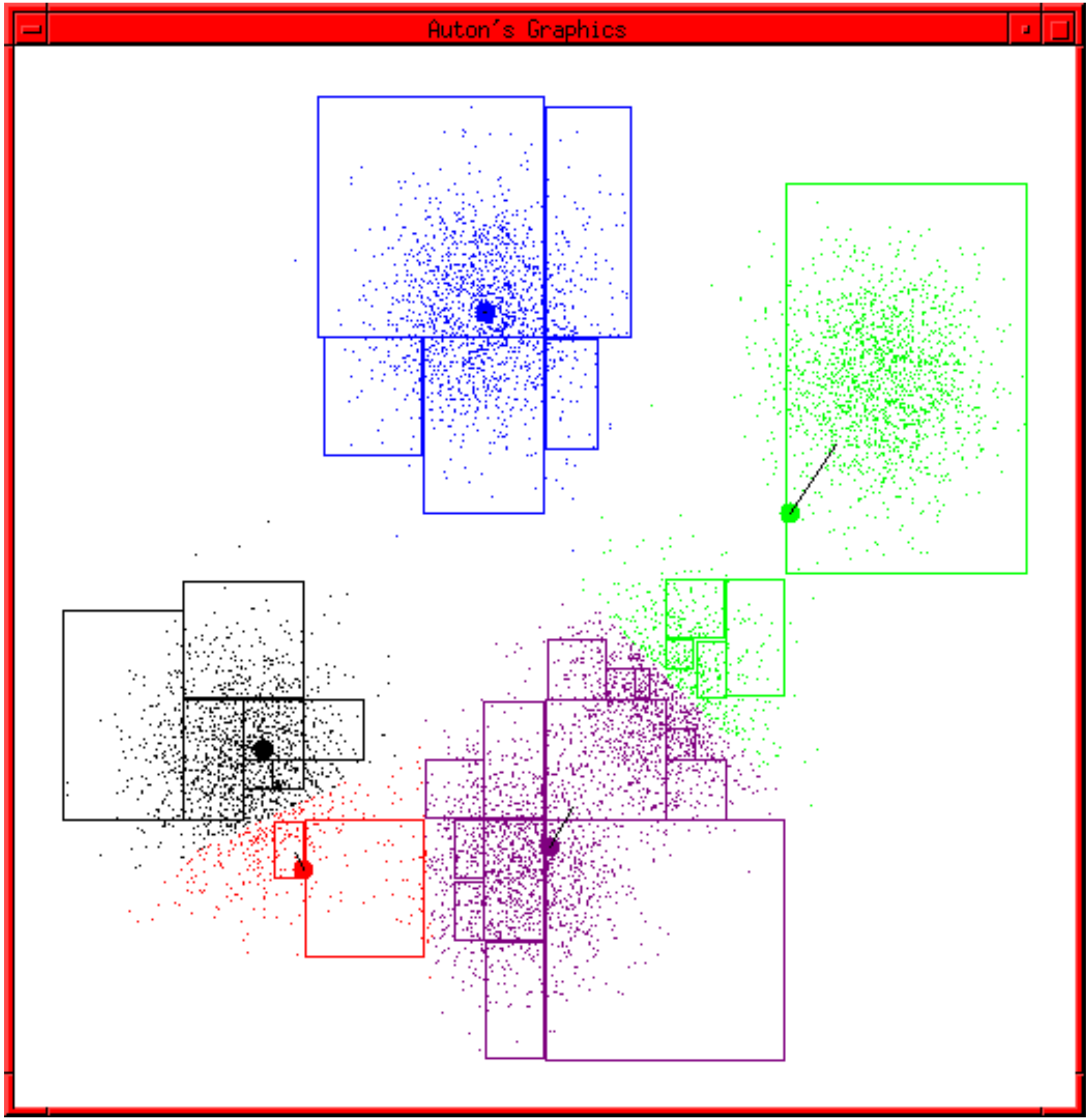
> *Dan Pelleg and Andrew Moore. Accelerating Exact k-means Algorithms with Geometric Reasoning. Proc. Conference on Knowledge Discovery in Databases 1999, (KDD99) (available on* www.autonlab.org/pap.html*)*



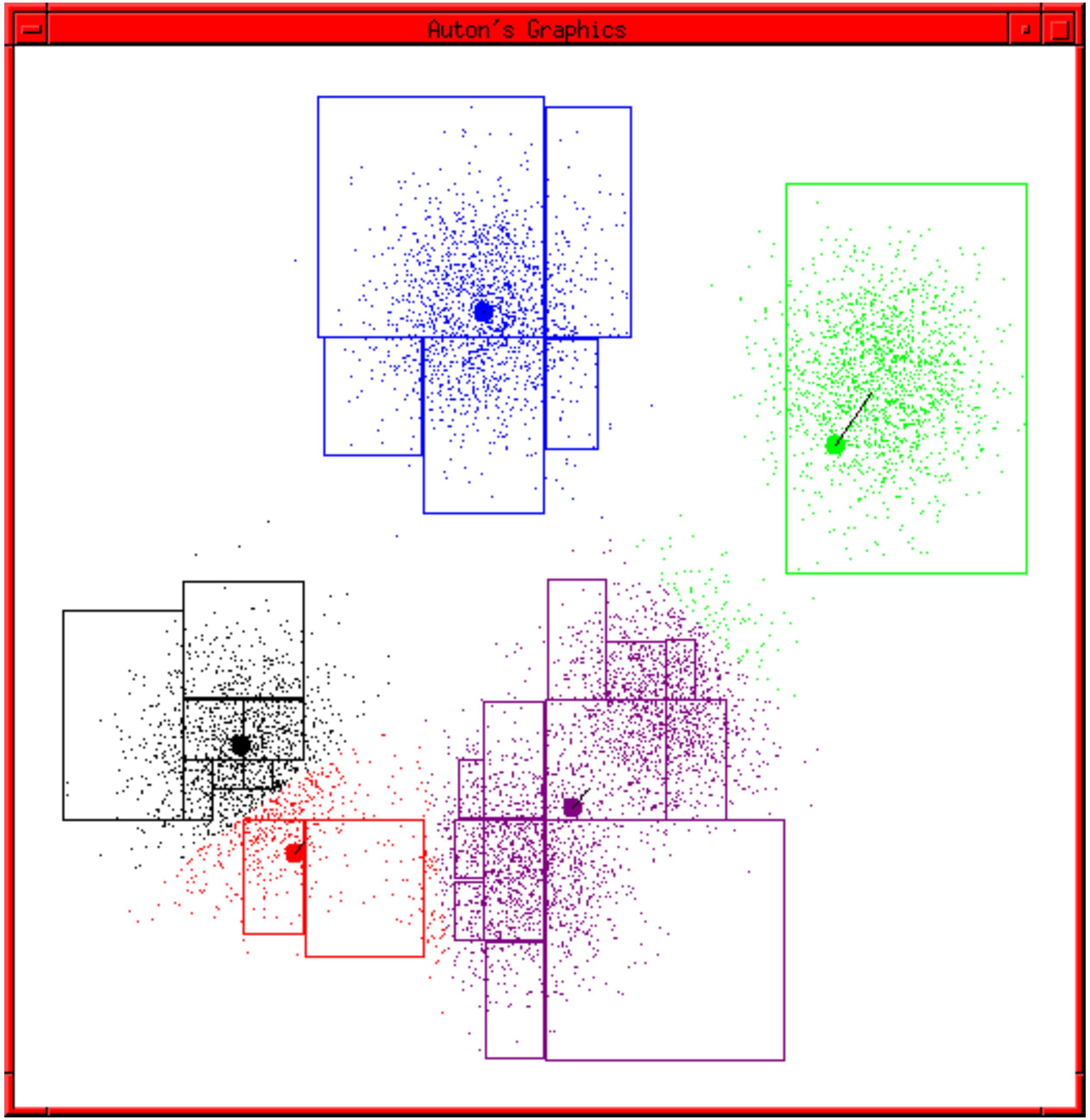
K-means continues …



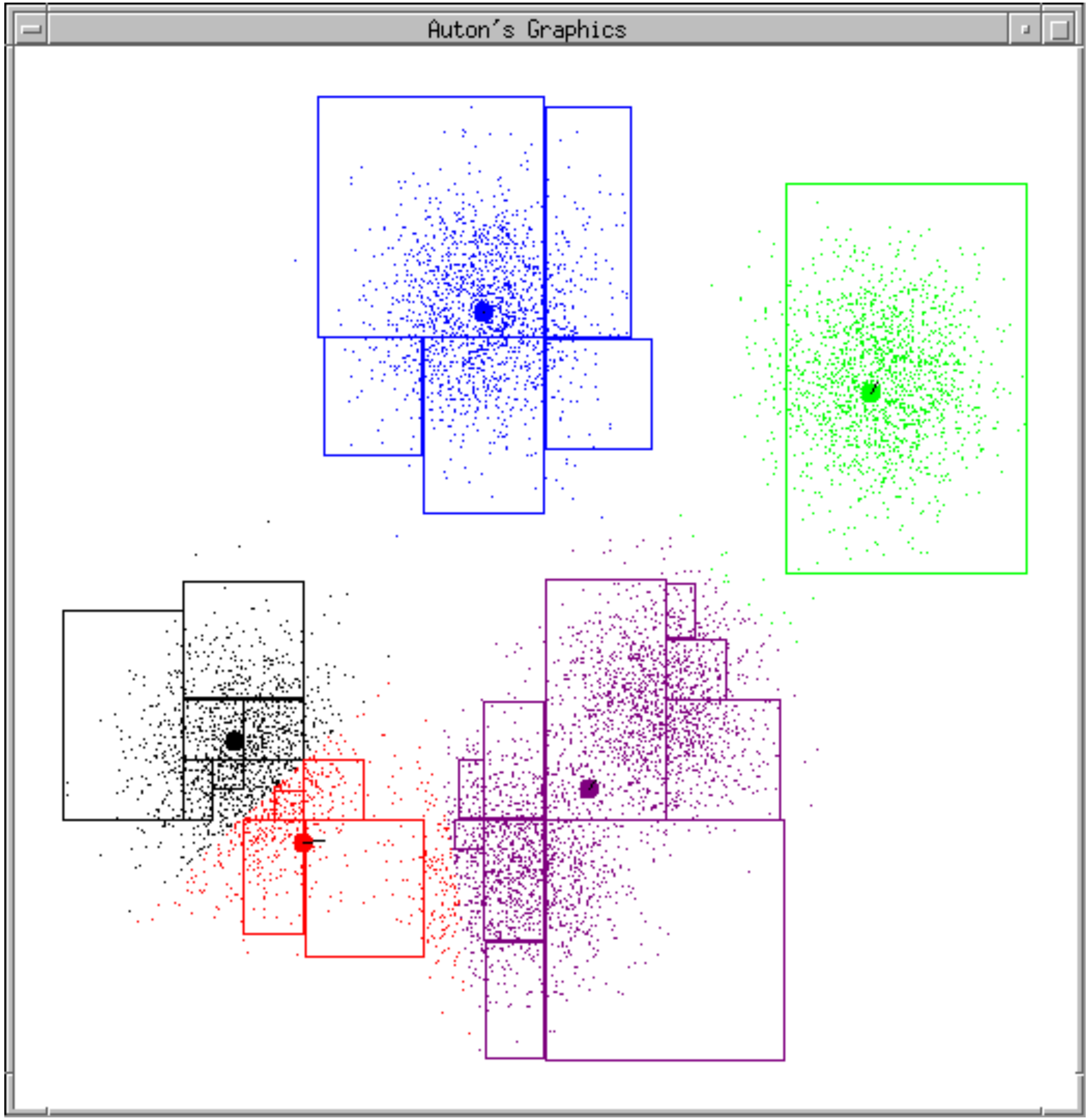
K-means continues …



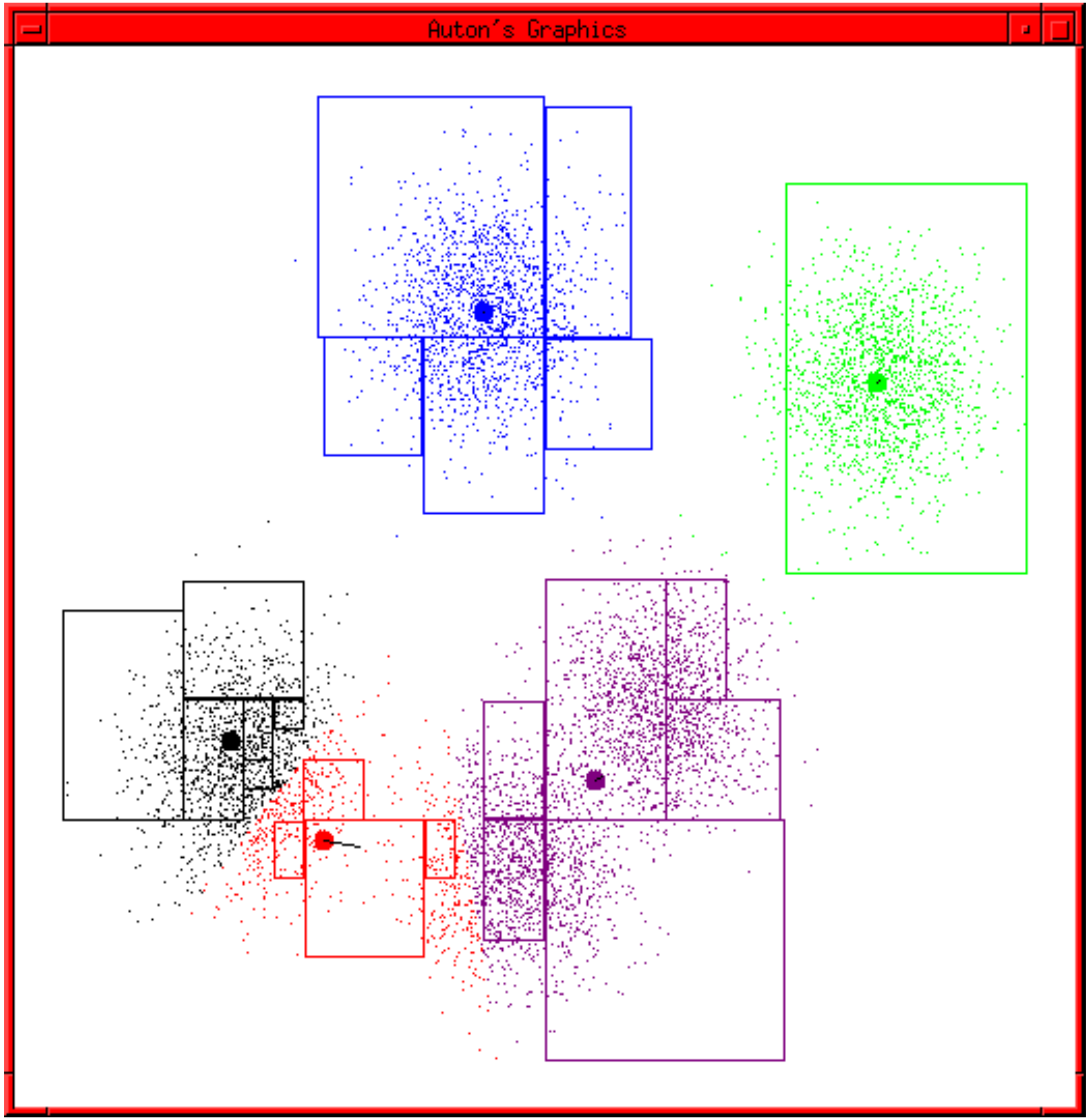
K-means continues …



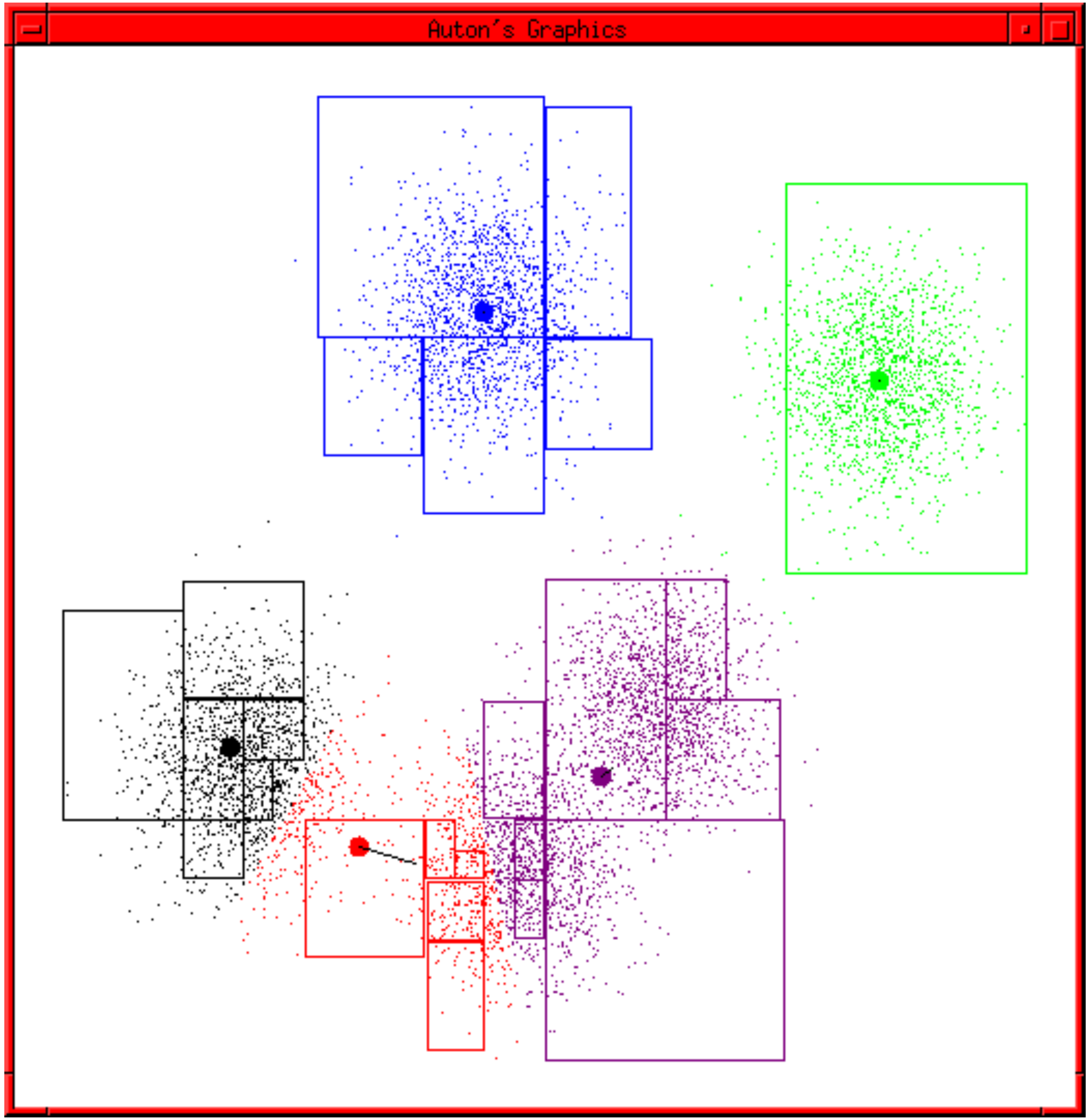
K-means continues …



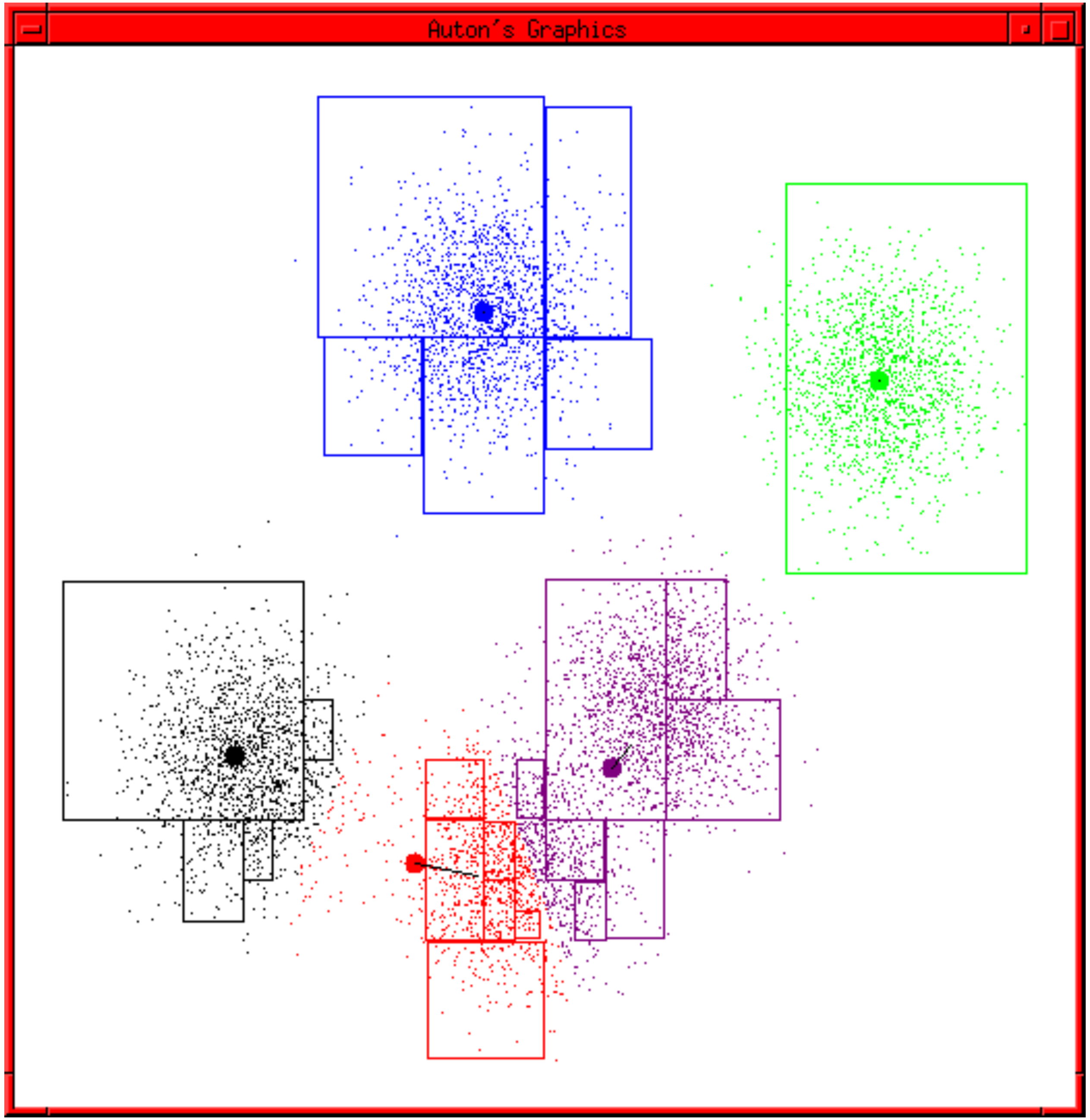
K-means continues …



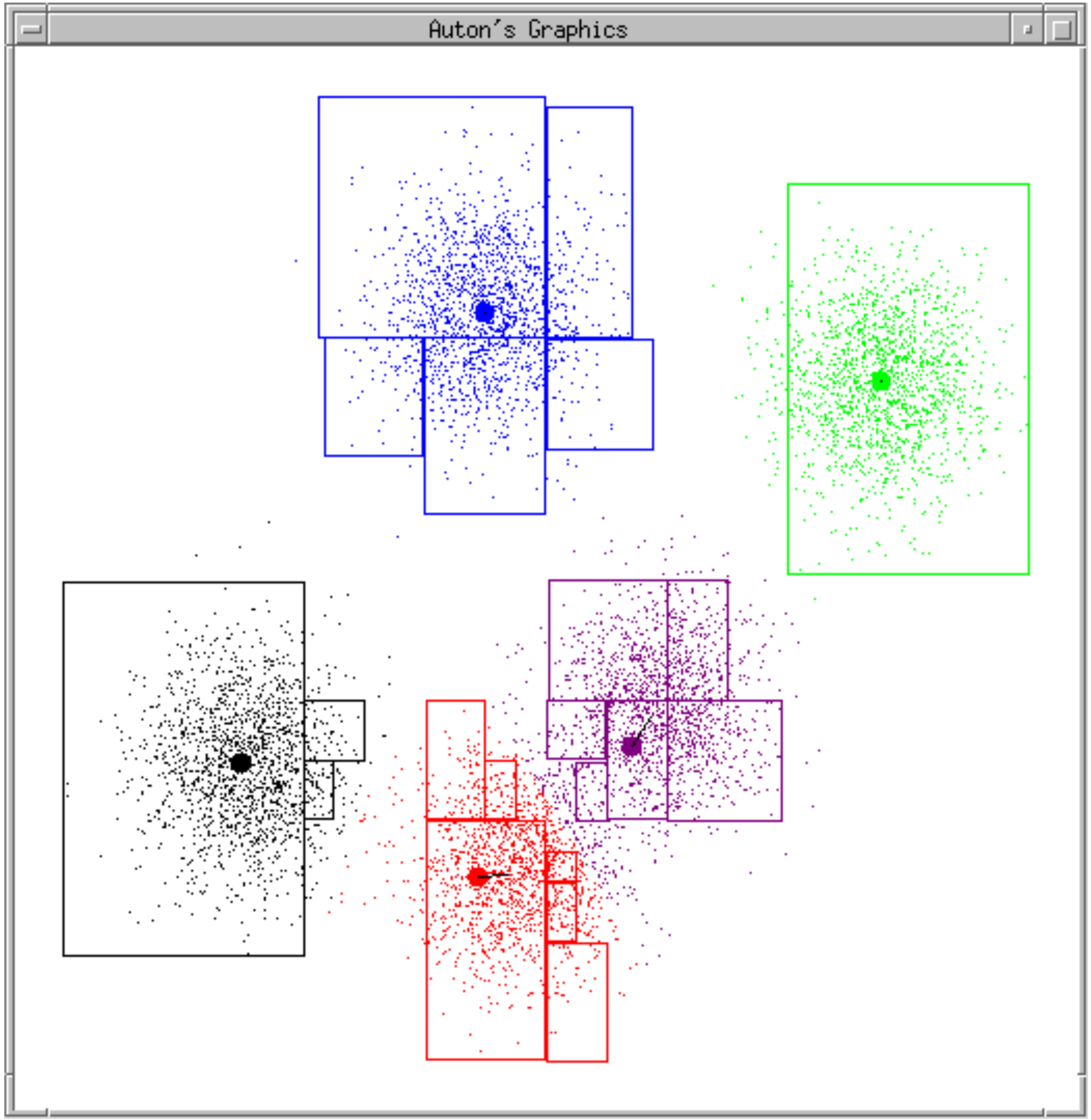
K-means continues …



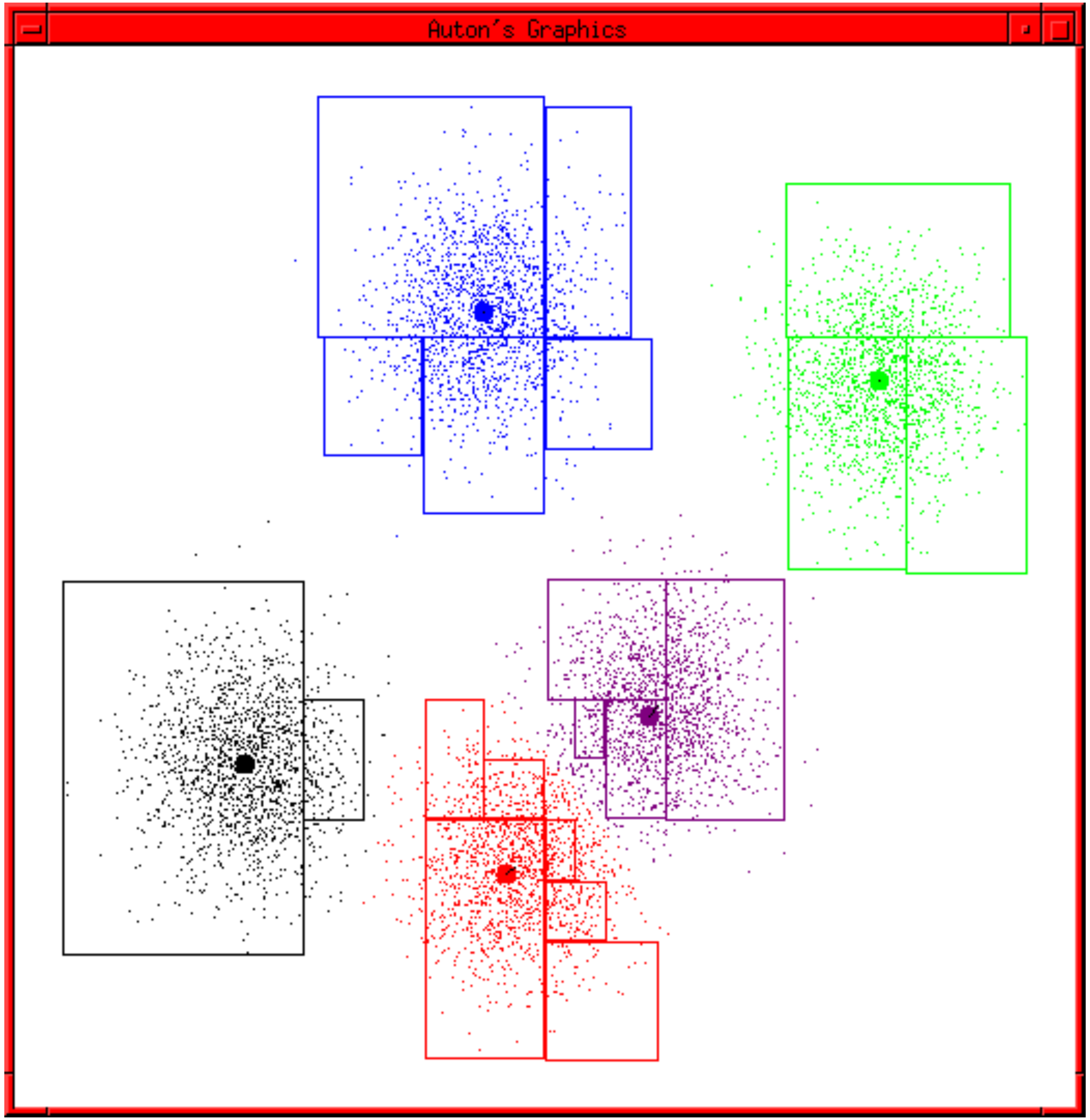
K-means continues …



K-means continues …



K-means terminates



Clustering for data exploration

Today’s data set

- Affairs.csv:
  - One row = one person
  - Columns:
    - **age, children**: age and number of children
    - **religious**: the person's religiousness
    - **educ**: the person's education level
    - **occupation**: a code that identifies the person's occupation
    - **rate_marriage**: how the person rates his or her marriage,
    - **yrs_married**: length of the marriage, in years
    - **affairs**: time spent, in hours/week, in extra-marital affairs

  - **Our goal is to find groups of homogeneous people**

Cluster the rows of the table

```
df = pd.read_csv('affairs.csv', index_col=0)
```

```
df = pd.get_dummies(columns=['occupation'],data=df)
```

```
from sklearn.cluster import KMeans
clu = KMeans(n_clusters=3, random_state=0)
clu.fit(df)
```

Each row in the data set is labeled with a **categorical value** that indicates its cluster

Retrieve the cluster labels:

```
clu.labels_[:20]
```

```
array([2, 2, 0, 1, 2, 2, 1, 1, 0, 0, 0, 0, 1, 0, 0, 2, 0, 1, 1, 2])
```

Summary information on cluster

```
df2=pd.DataFrame.copy(df)
df2['cluster'] = clu.labels_
df2.groupby('cluster').mean()
```

	rate_marriage	age	yrs_married	children	religious	educ	affairs	occupation_1	occupation_2
cluster									
0	4.215116	24.095785	3.358430	0.496366	2.330814	14.348547	0.876362	0.009012	0.127616
1	3.945343	39.979215	20.461124	3.060431	2.649731	13.883757	0.394723	0.003849	0.120862
2	4.017824	30.927474	11.814382	1.972649	2.449293	14.177013	0.591874	0.003073	0.161647

Is there anything that strikes you as unexpected?
Complete here

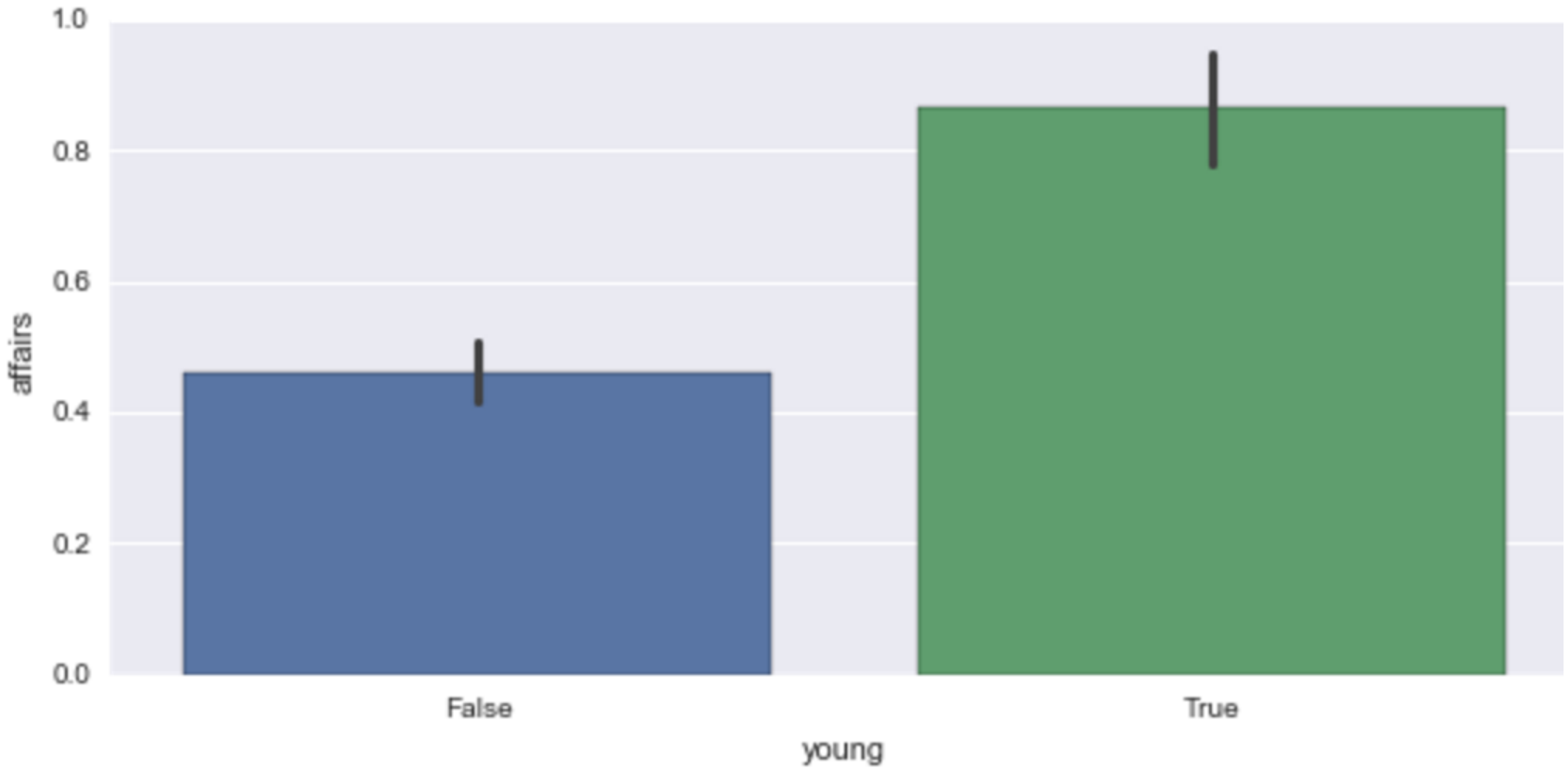
Let’s verify it

```
df3 = df2.copy()
df3['young']=(df3['age'] <= 27)
df3.groupby('young').agg({'affairs' : ['size', 'mean'], 'rate_marriage' : 'mean'})
```

	rate_marriage	affairs	
	mean	size	mean
young			
False	3.991987	2496	0.459645
True	4.185530	3870	0.863859

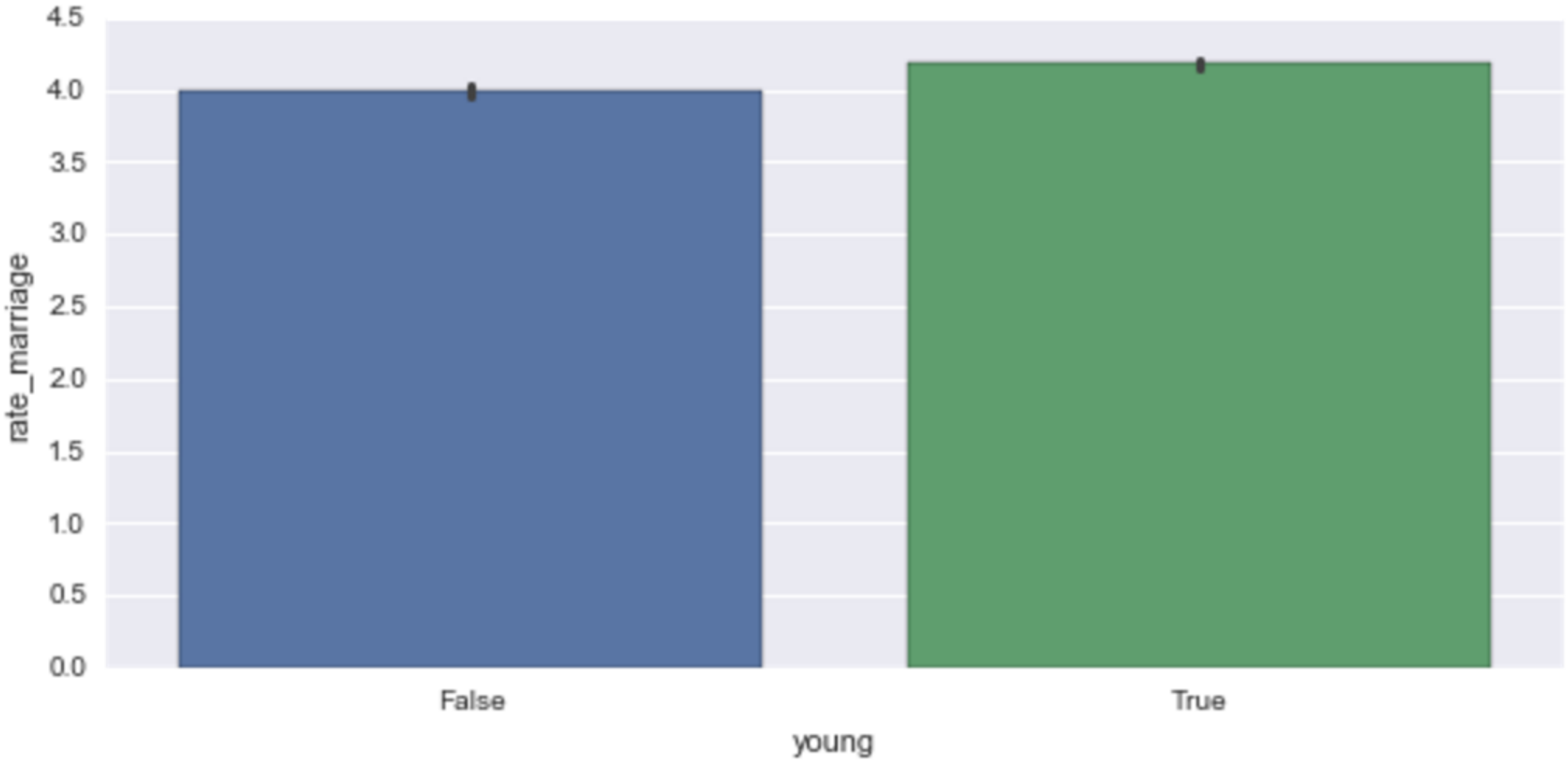
```python
import seaborn as sns
sns.factorplot(x='young', y='affairs', data=df3,kind='bar', aspect=2)
```

```
<seaborn.axisgrid.FacetGrid at 0x16bb7c18>
```



```python
import seaborn as sns
sns.factorplot(x='young', y='rate_marriage', data=df3,kind='bar', aspect=2)
```

```
<seaborn.axisgrid.FacetGrid at 0x162b6f98>
```



Tips for clustering

- Use few attributes
  - Too many attributes make it very hard to see the differences between the clusters
- Do not have correlated attributes
  - E.g., in the previous example, we could have removed at least one among age, children, and yrs_married
- Use a small number of clusters
  - Or else, it may be very hard to tell the difference between them

Optimizing the clustering

- For each clustering technique t that you want to try:
  - For each k=2…10
    - Run technique t with k clusters
    - Measure the quality of the clusters obtained (see next slide)

- Return the best combination of technique t and k

Measuring the cluster quality – Silhouette Coefficient

- Input:
  - Data
  - Cluster assignment
- Output:
  - A score from -1 to 1 where
    - 0 is random clustering
    - 1 is perfect clustering
- For each point i, compute:
  - a_i = mean distance between i and all other points in the same cluster
  - b_i = mean distance between i and the points in the next nearest cluster
  - $s_i = \frac{b_i - a_i}{\max(a_i, b_i)}$
- The overall silhouette coefficient is the mean value of $s_i, i = 1 \ldots n$

Finding the best clustering technique and number of clusters

```
from sklearn import metrics
from sklearn.cluster import KMeans
from sklearn.cluster import Birch
from sklearn.cluster import AgglomerativeClustering

bestSil = -1
for k in range(2,10):
    clus = [KMeans(n_clusters=k,n_jobs=-1), Birch(n_clusters=k), AgglomerativeClustering(n_clusters=k)]
    for cl in clus:
        res = cl.fit(df)
        sil = metrics.silhouette_score(X, res.labels_)
        print (str(cl)[:10] + ' with k =' +str(k) + ": " + str(round(sil,4)))
        if (sil > bestSil):
            bestSil = sil
            bestCl = cl
```