Advanced Clustering Algorithms

Clustering: a brief review

Group data points that are close to each other

Closeness defined by distance or density

k-means: clustering algorithm based on distance to nearest centroid

Clustering: when *k*-means isn't enough

k-means is usually first choice for a clustering algorithm due to its simplicity and ease of implementation

However, it has weakness that limit its use

In this lesson we look at other, more advanced clustering algorithms

Clustering: advanced clustering algorithms

Hierarchical agglomerative clustering
DBSCAN
Mean shift
Spectral clustering
Affinity propagation

Clustering: advanced clustering algorithms

Hierarchical agglomerative clustering
DBSCAN
Mean shift
Spectral clustering
Affinity propagation

https://scikit-learn.org/stable/modules/clustering.html

Hierarchical Agglomerative Clustering

Hierarchical agglomerative clustering (HAC)

A partitioning algorithm that merges the data sequentially

Points are merged using metric (such as Euclidean distance) to create fewer, larger clusters

Algorithm ends at pre-determined number of clusters or pre-determined distance between clusters

HAC: the algorithm

1. Each data point starts off as its own cluster

2. Nearest points are merged into one cluster

3. Next-nearest points are merged. Two possible outcomes:
   a. New cluster created
   b. Point merged into existing cluster

4. Repeat steps (2) and (3) until end condition reached

HAC: required inputs

1. Metric—function to calculate distance

e.g. Euclidean, Manhattan or cosine

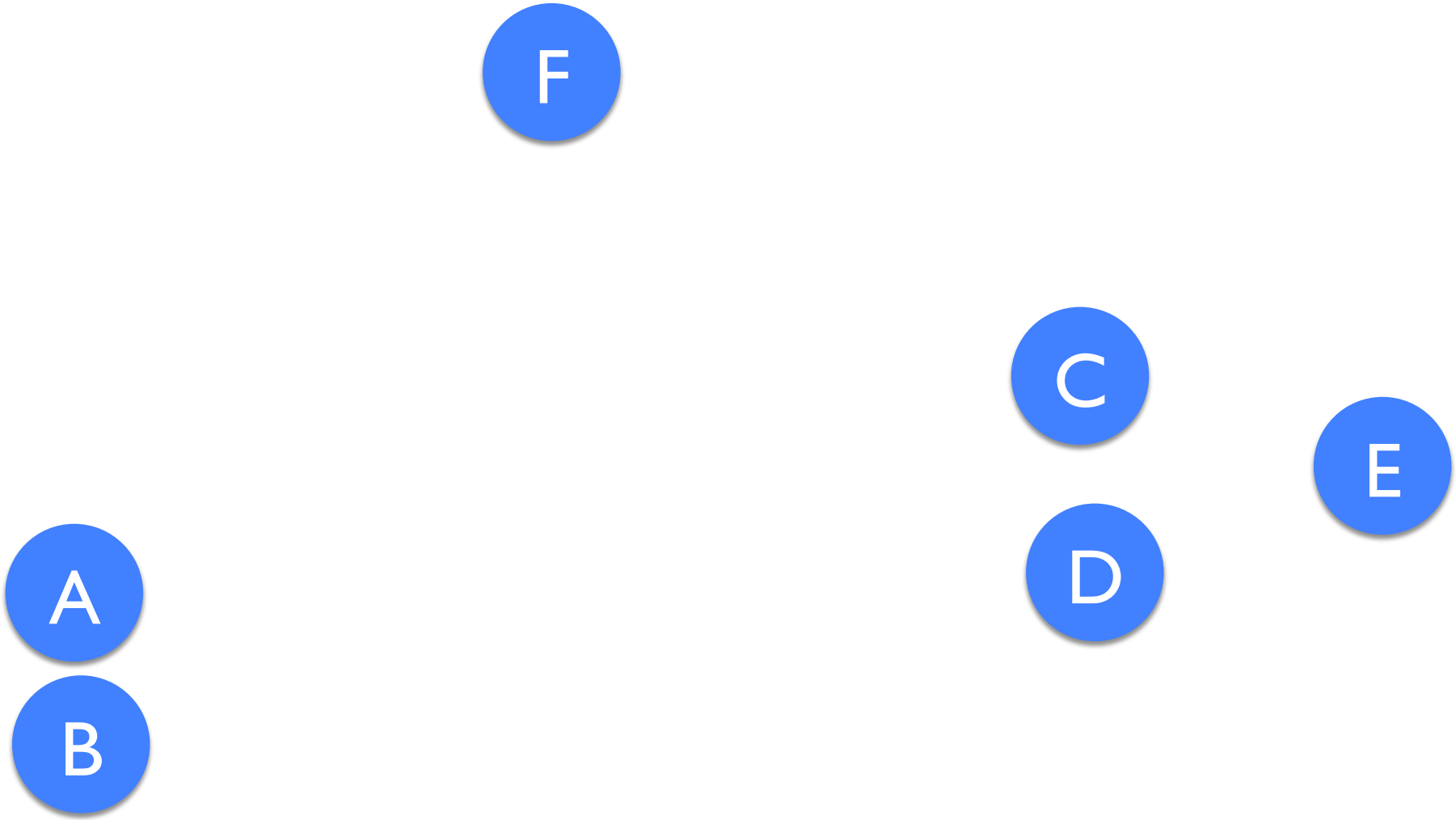
2. Linkage—how to merge clusters

e.g. use min/avg/max distance between points in cluster
or minimize variance ("inertia") of clusters being merged

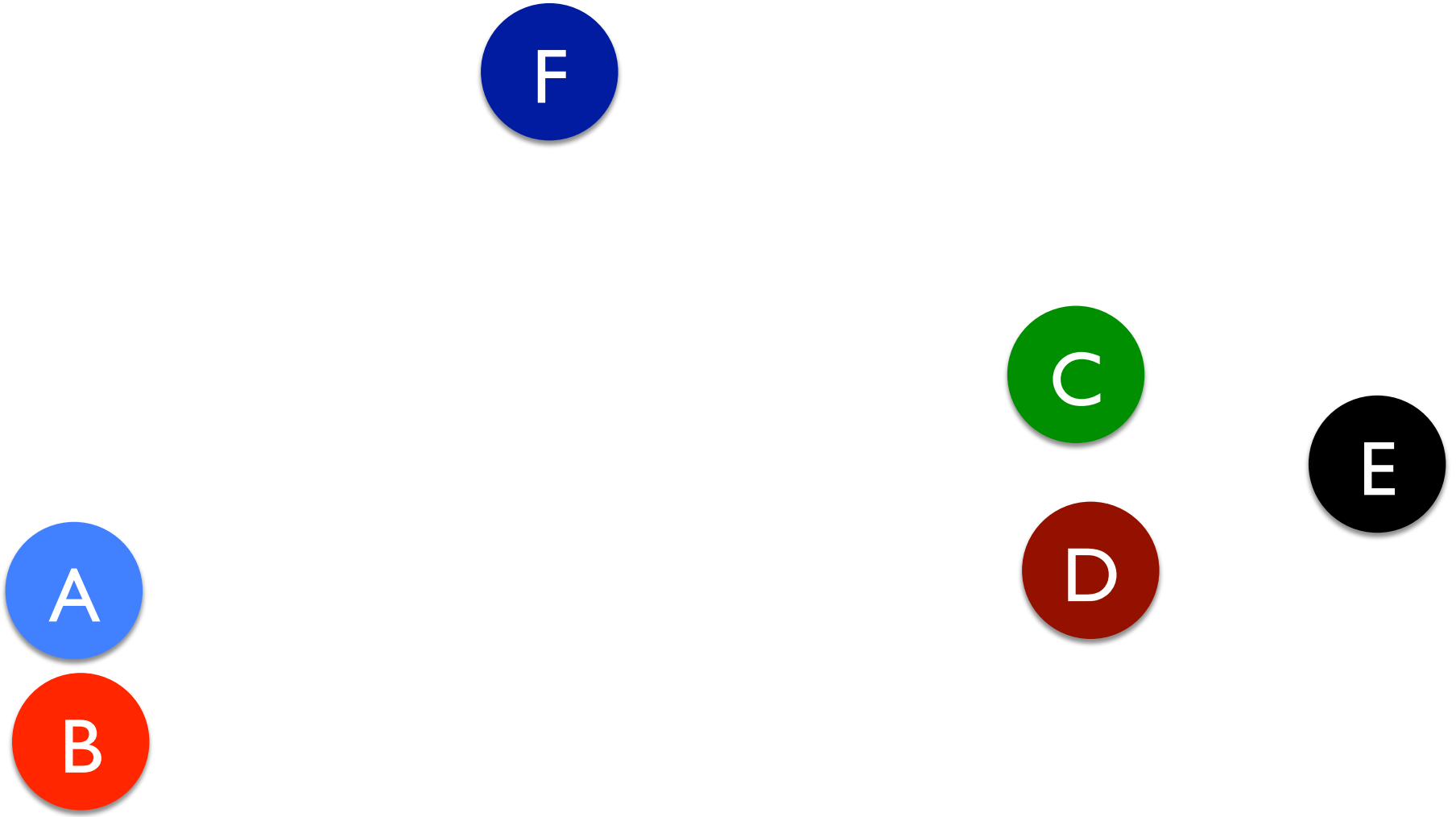
3. End—when to terminate algorithm

e.g. usually specify number of clusters
can also specify threshold for inter-cluster distance

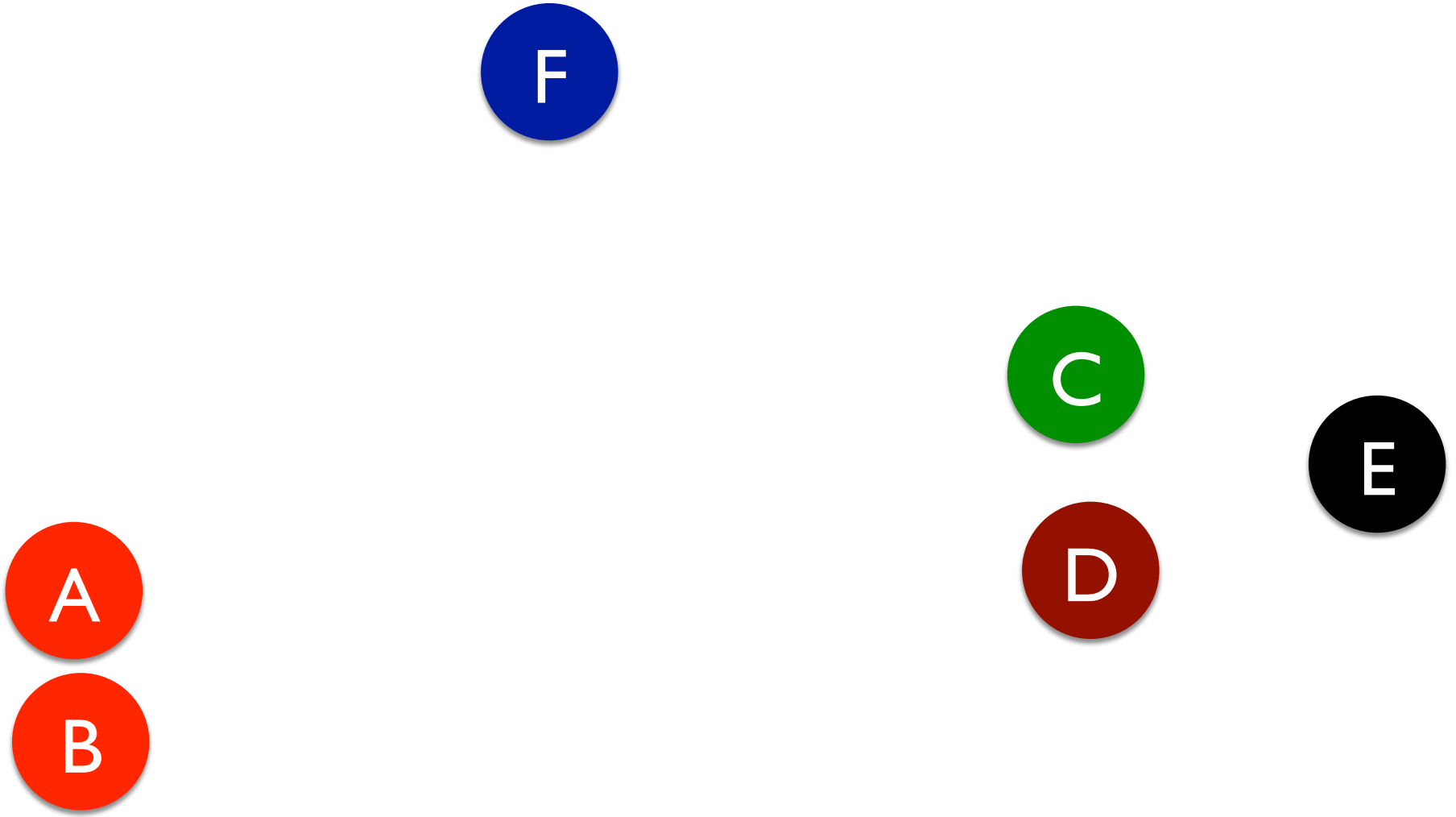
HAC: toy example



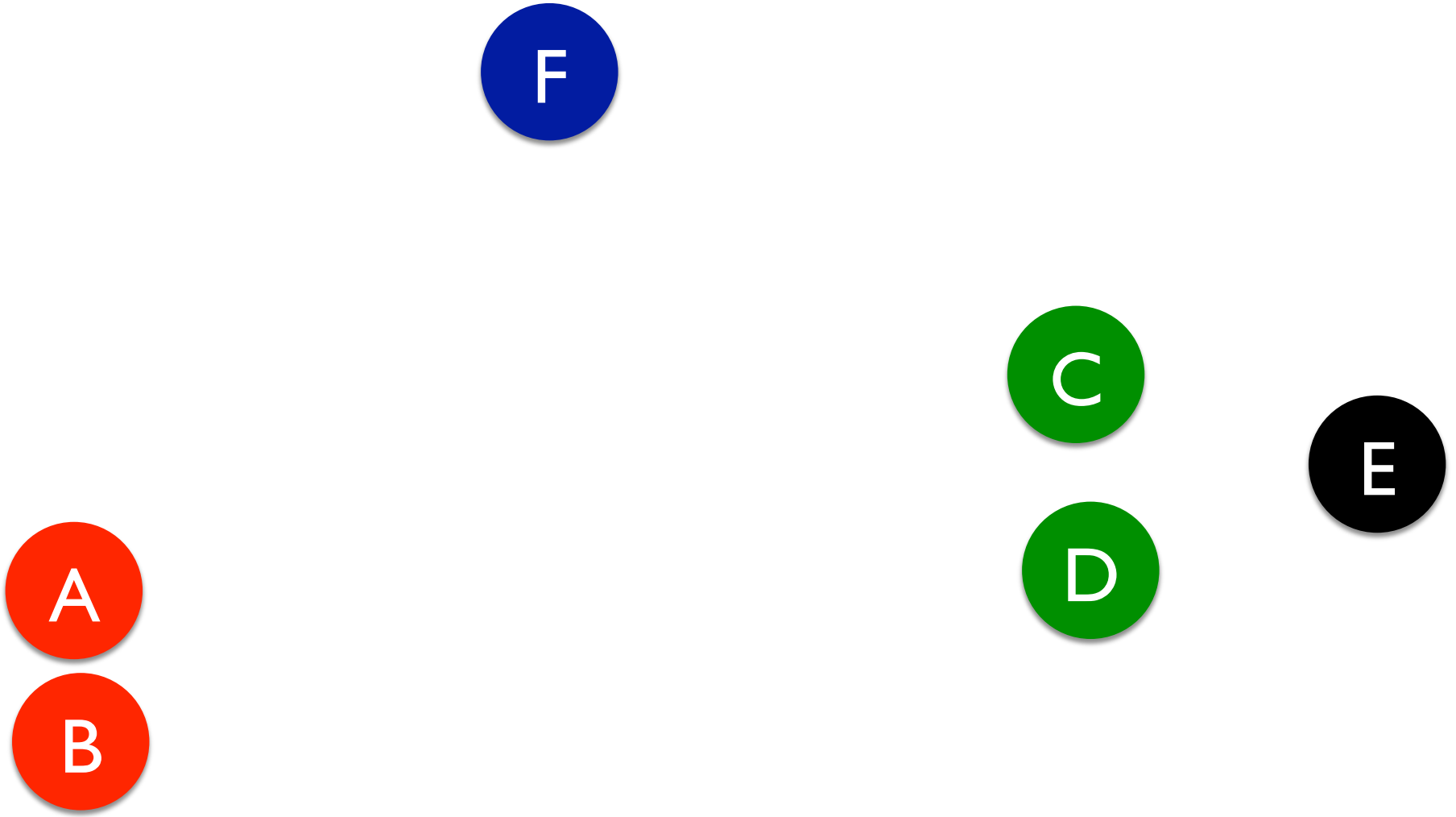
HAC: toy example



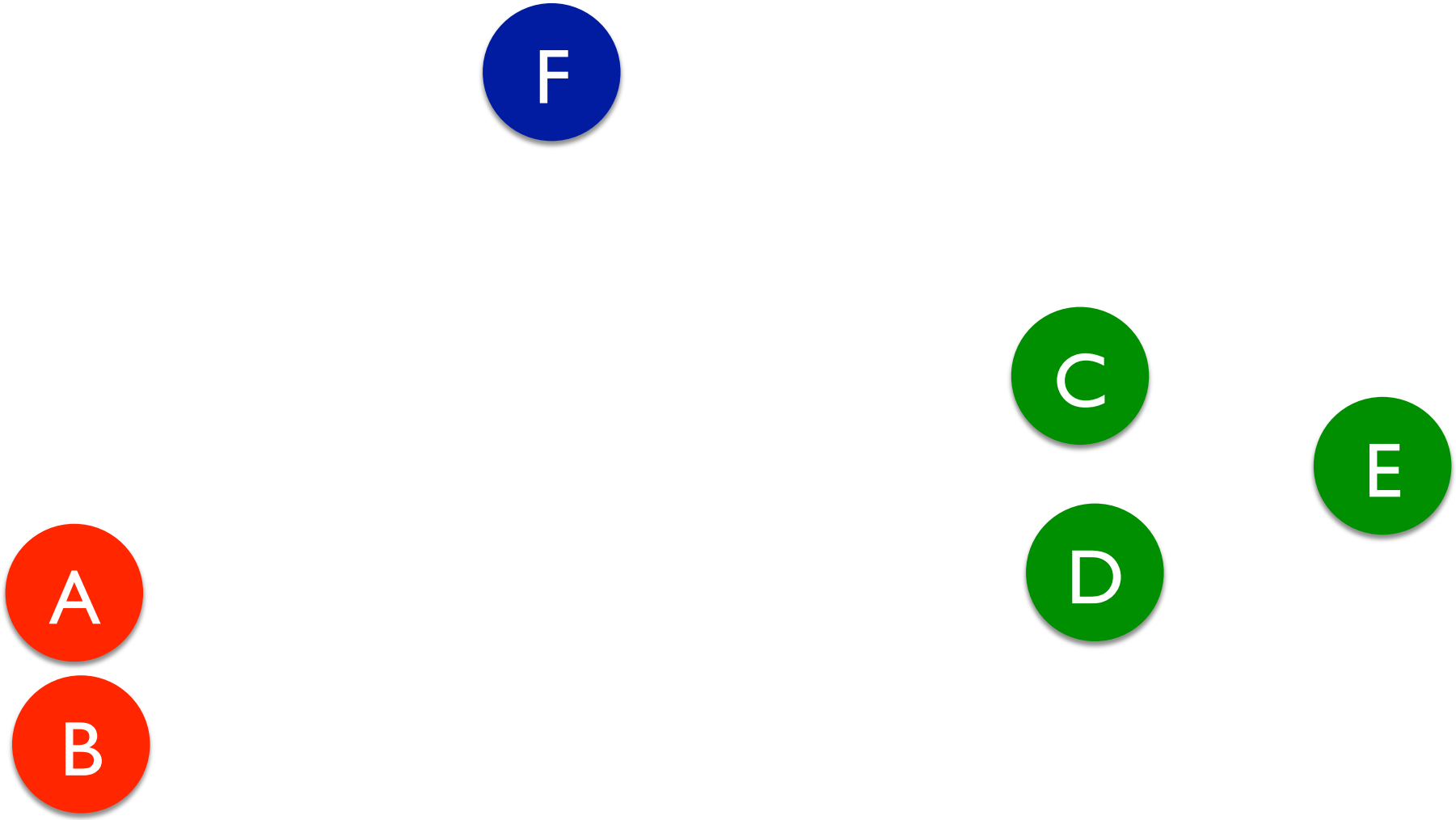
HAC: toy example



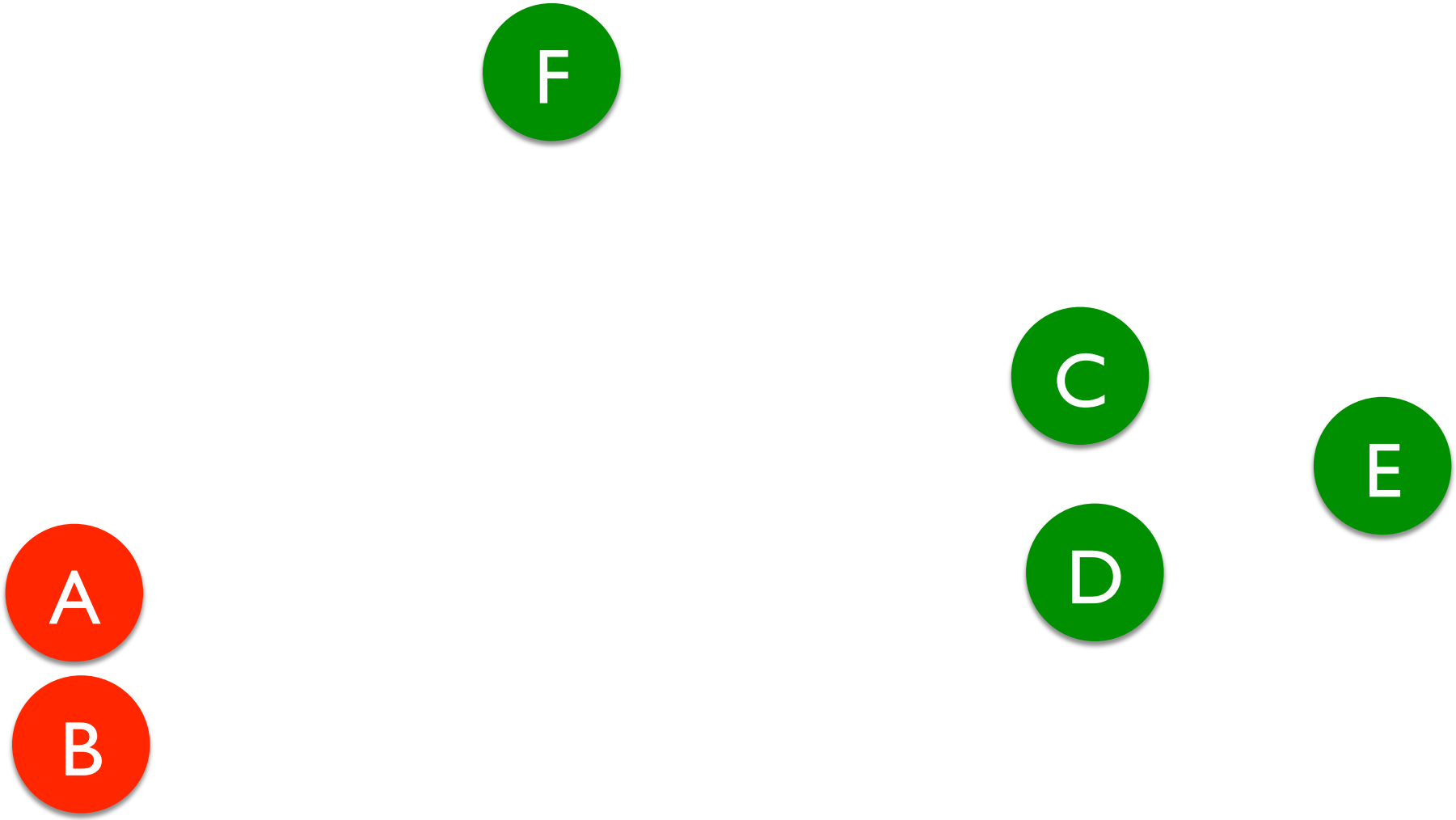
HAC: toy example



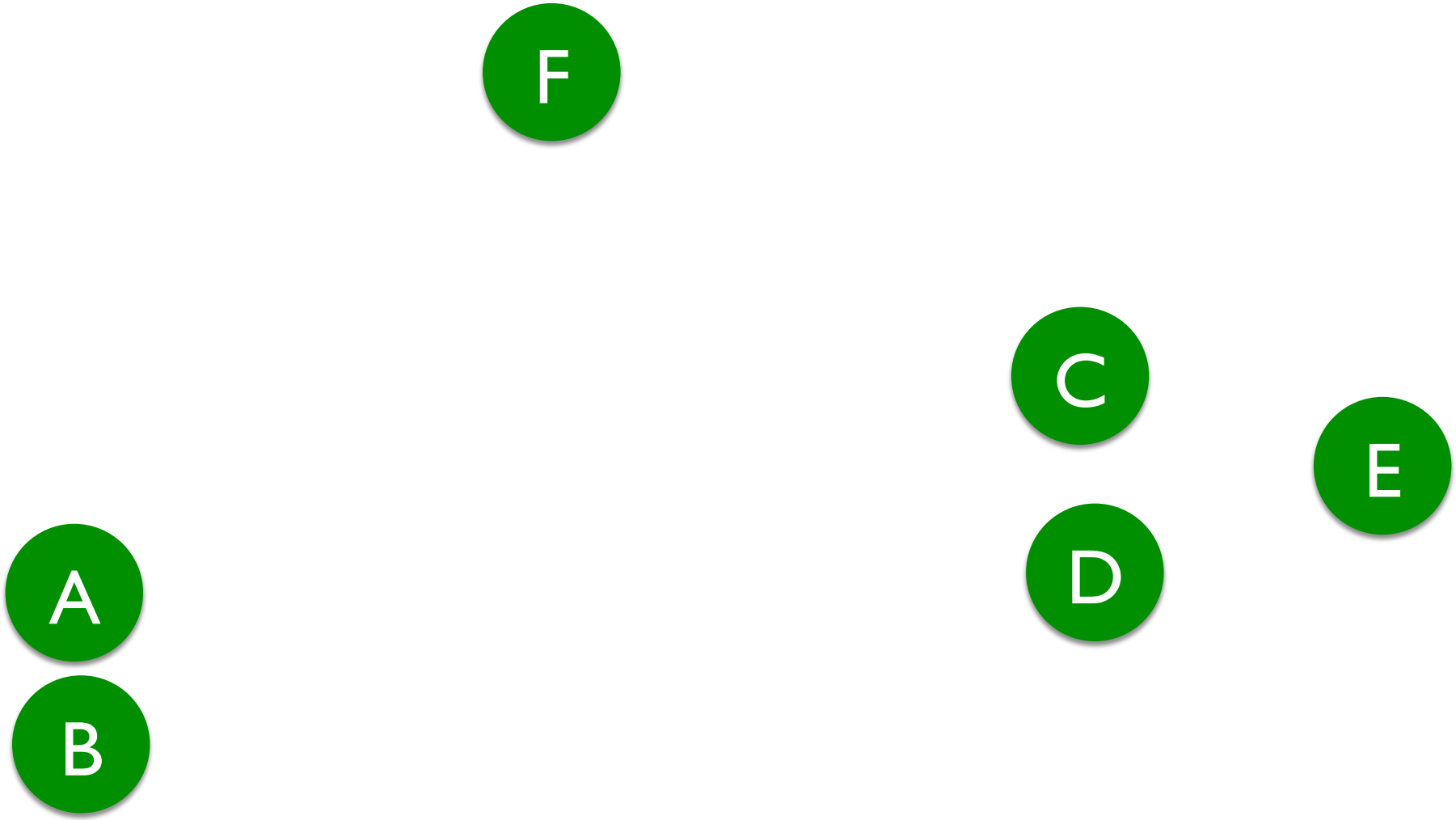
HAC: toy example



HAC: toy example

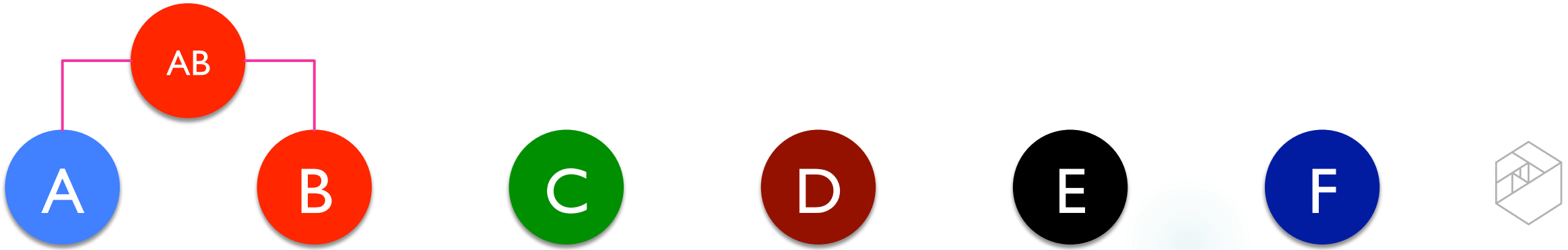


HAC: toy example

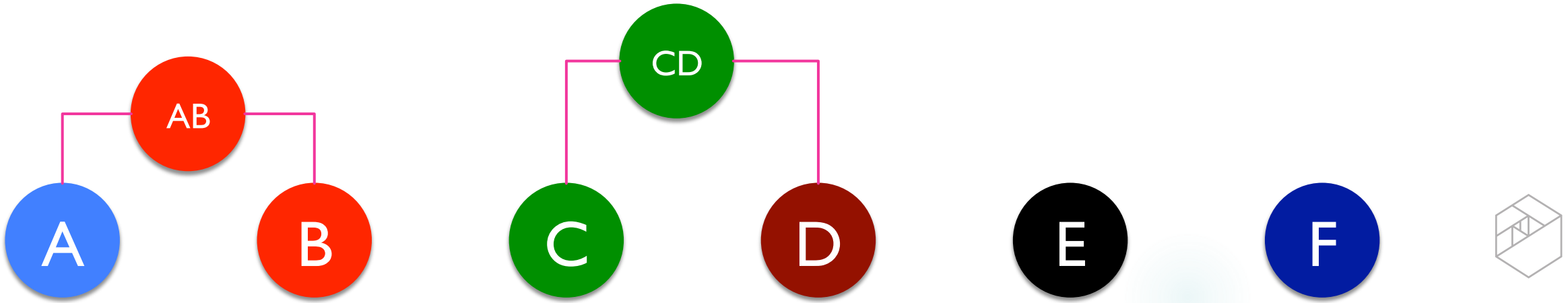


HAC: toy example as a dendrogram

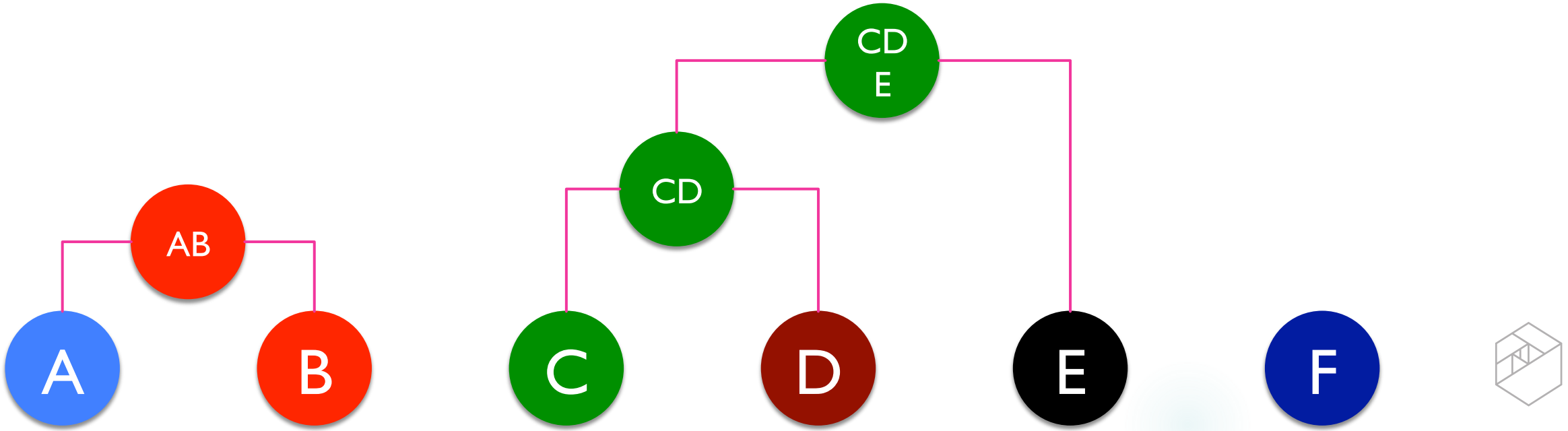
HAC: toy example as a dendrogram



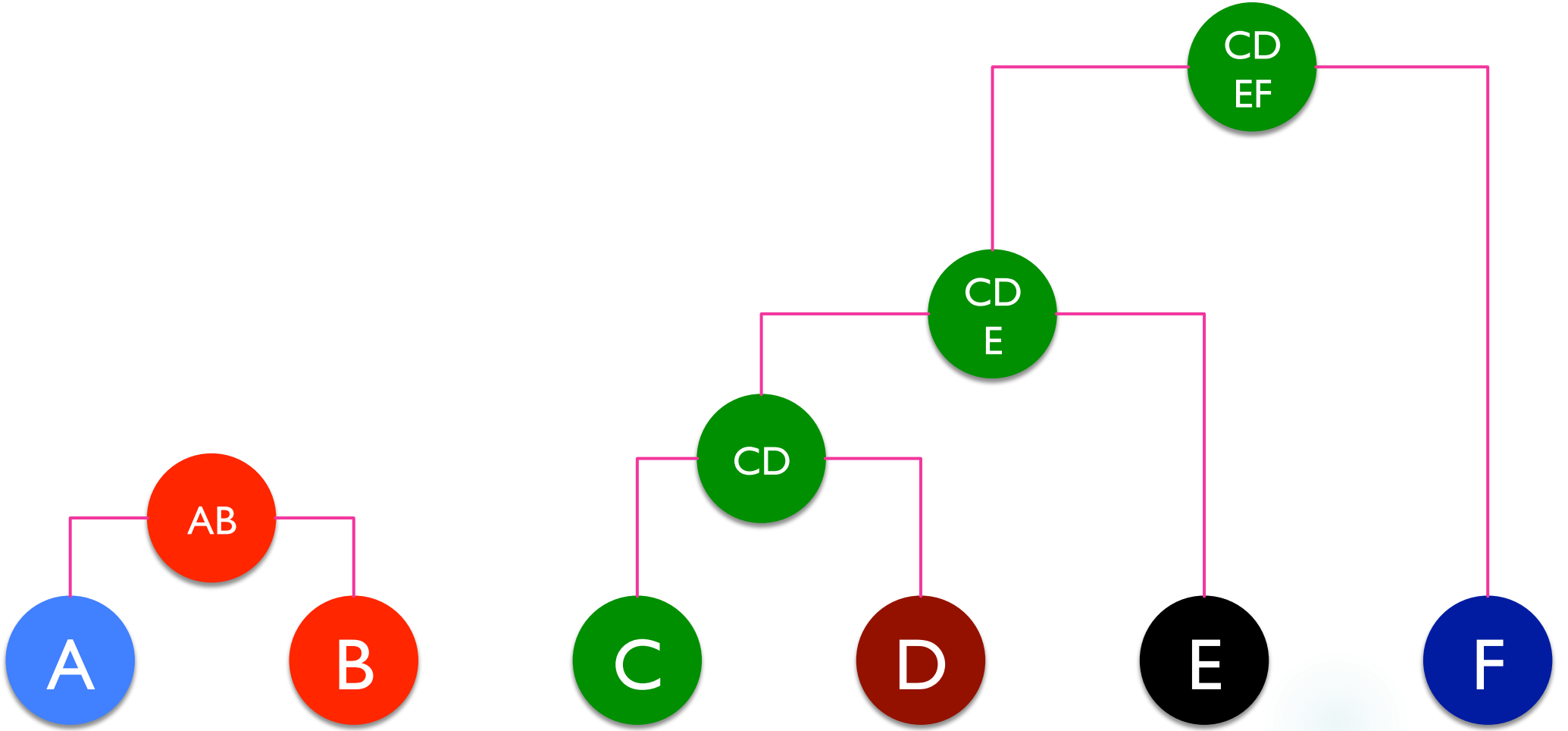
HAC: toy example as a dendrogram



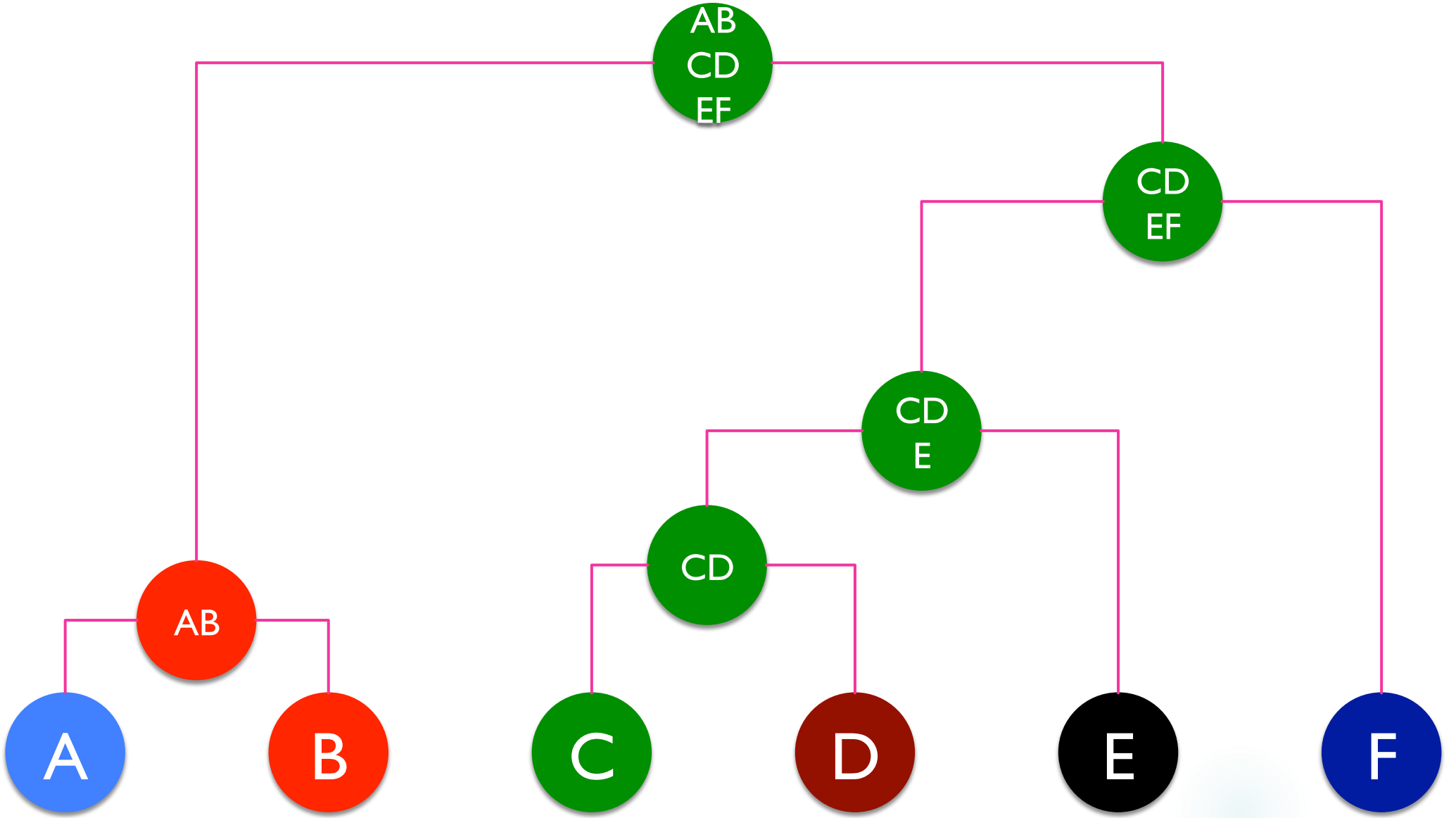
HAC: toy example as a dendrogram



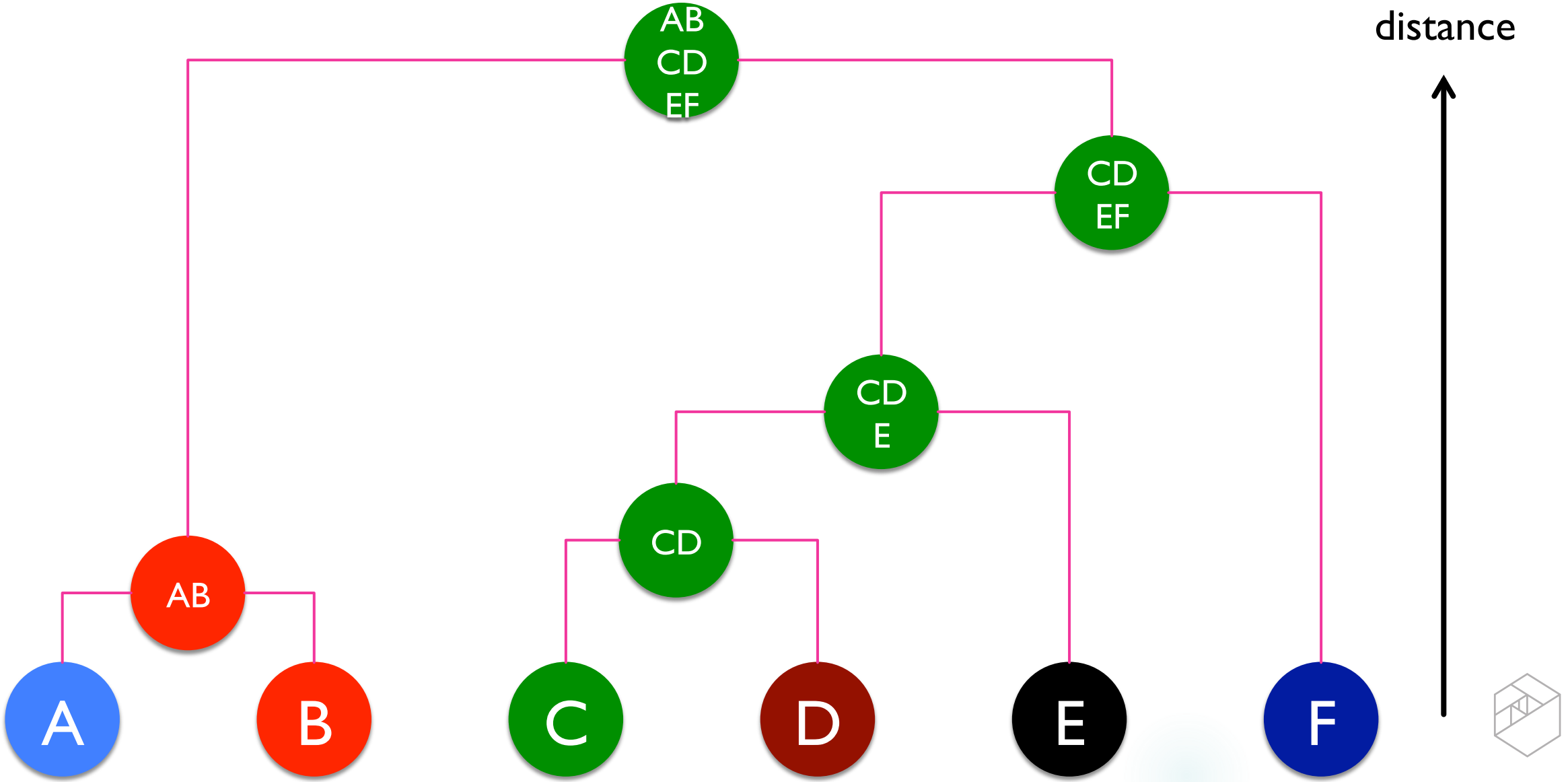
HAC: toy example as a dendrogram



HAC: toy example as a dendrogram



HAC: toy example as a dendrogram



HAC: strengths and weaknesses

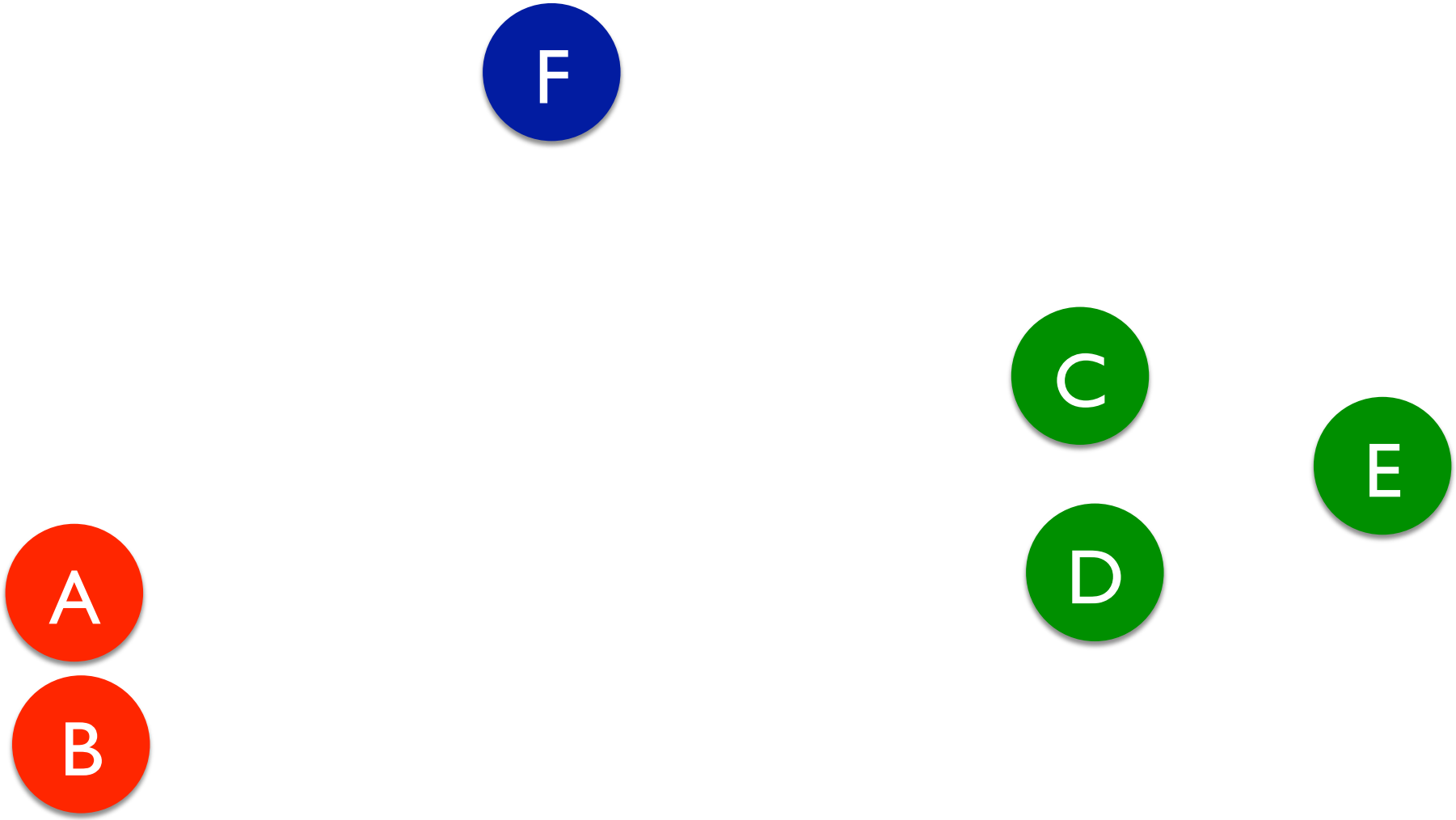
Strengths:

1. Easy to interpret
2. Easy to implement
3. Can get away without specifying number of clusters initially (if not used in termination condition)

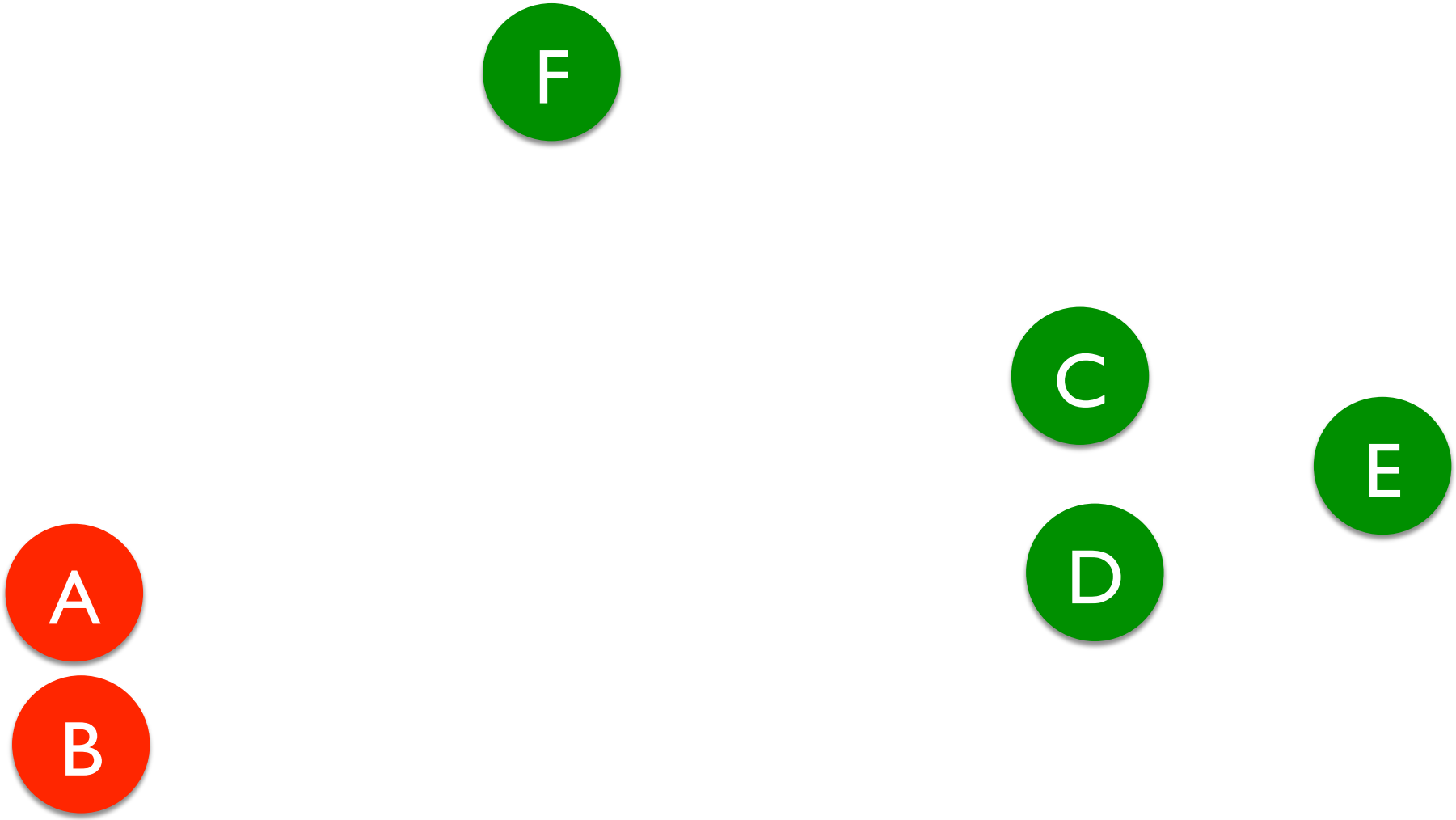
Weaknesses:

1. Slow: runtime is $O(n^3)$ for brute-force algorithms
2. Many (arbitrary) decisions
3. Hard to get robust results: sensitive to initial conditions and decisions made

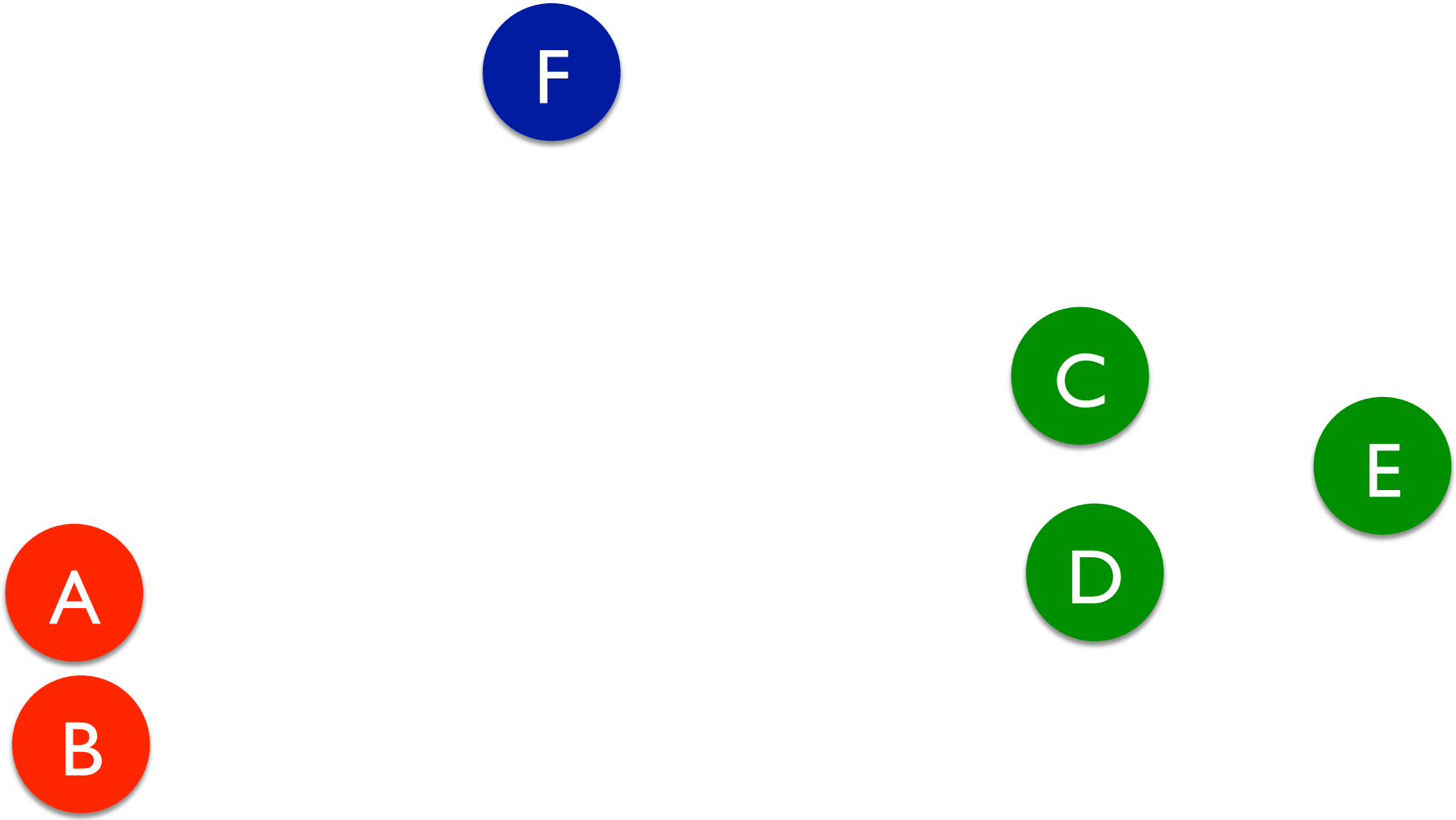
HAC: minimum distance linkage



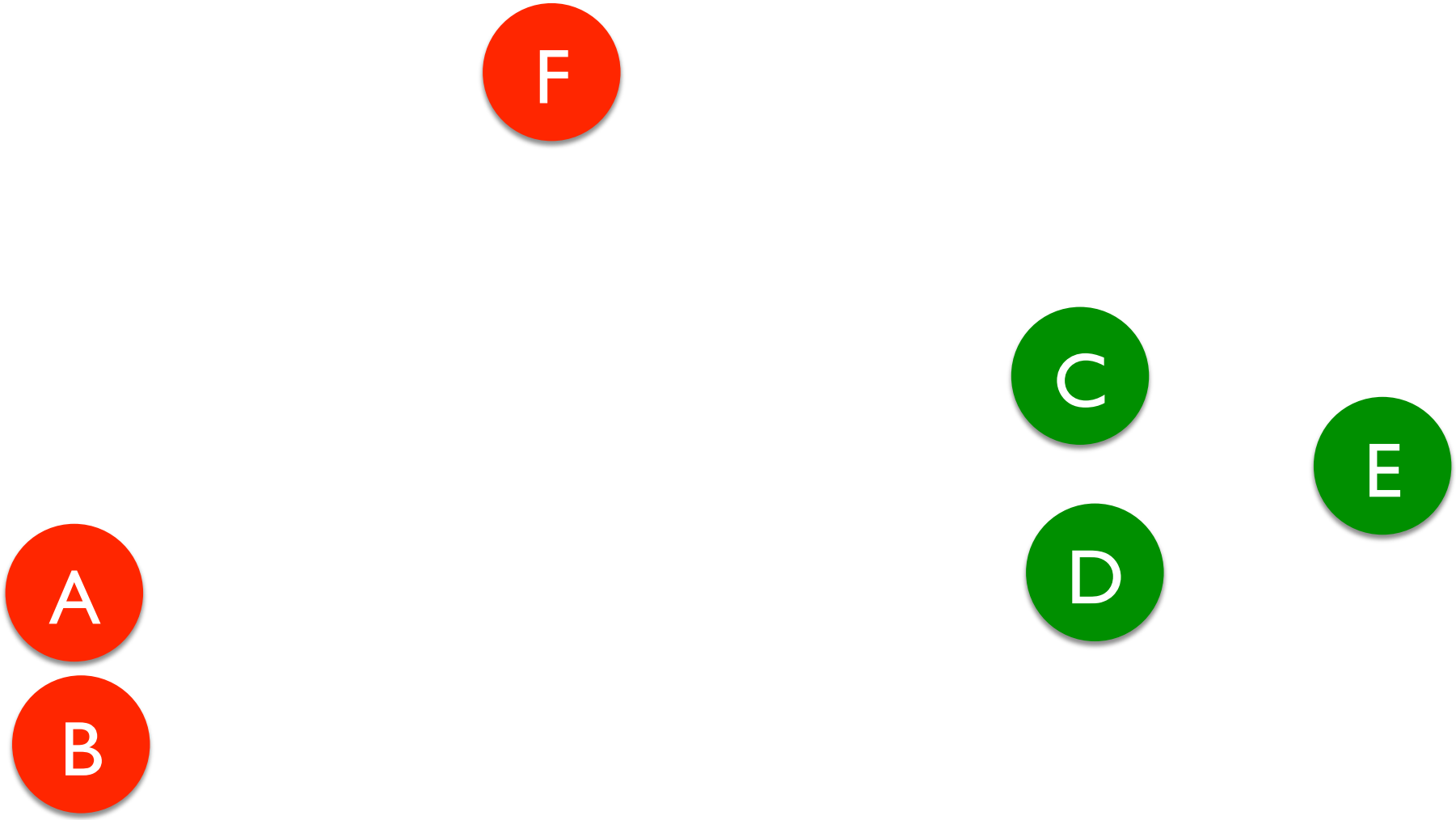
HAC: minimum distance linkage



HAC: maximum distance linkage



HAC: maximum distance linkage



HAC: How to get number of clusters

As with *k*-means,
if you have an external constraint or domain knowledge, use it!

If you don't have such knowledge or are doing
exploratory data analysis…no easy answer.

Can consider inertia or silhouette coefficients,
but given that both metric and linkage affect the clustering,
it can be tricky to get a robust result.

DBSCAN

Density-based spatial clustering of applications with noise

Density-based spatial clustering of applications with noise (DBSCAN)

A true clustering algorithm:
can have points that don't belong to any cluster

Points are clustered using density of local neighborhood:
finds core points in high density regions and expands clusters from them

Algorithm ends when all points have been classified as either belonging to a cluster or noise

DBSCAN: required inputs

1. Metric—function to calculate distance
2. Epsilon (eps, ε)—radius of local neighborhood
3. N_clu—determines density threshold (for fixed ε)

Core points are those which have more than n_clu neighbors in their local neighborhood ("ε-neighborhood")

DBSCAN: outputs

Three possible labels for any point:

Core: point which has more than n_clu neighbors
in their ε-neighborhood

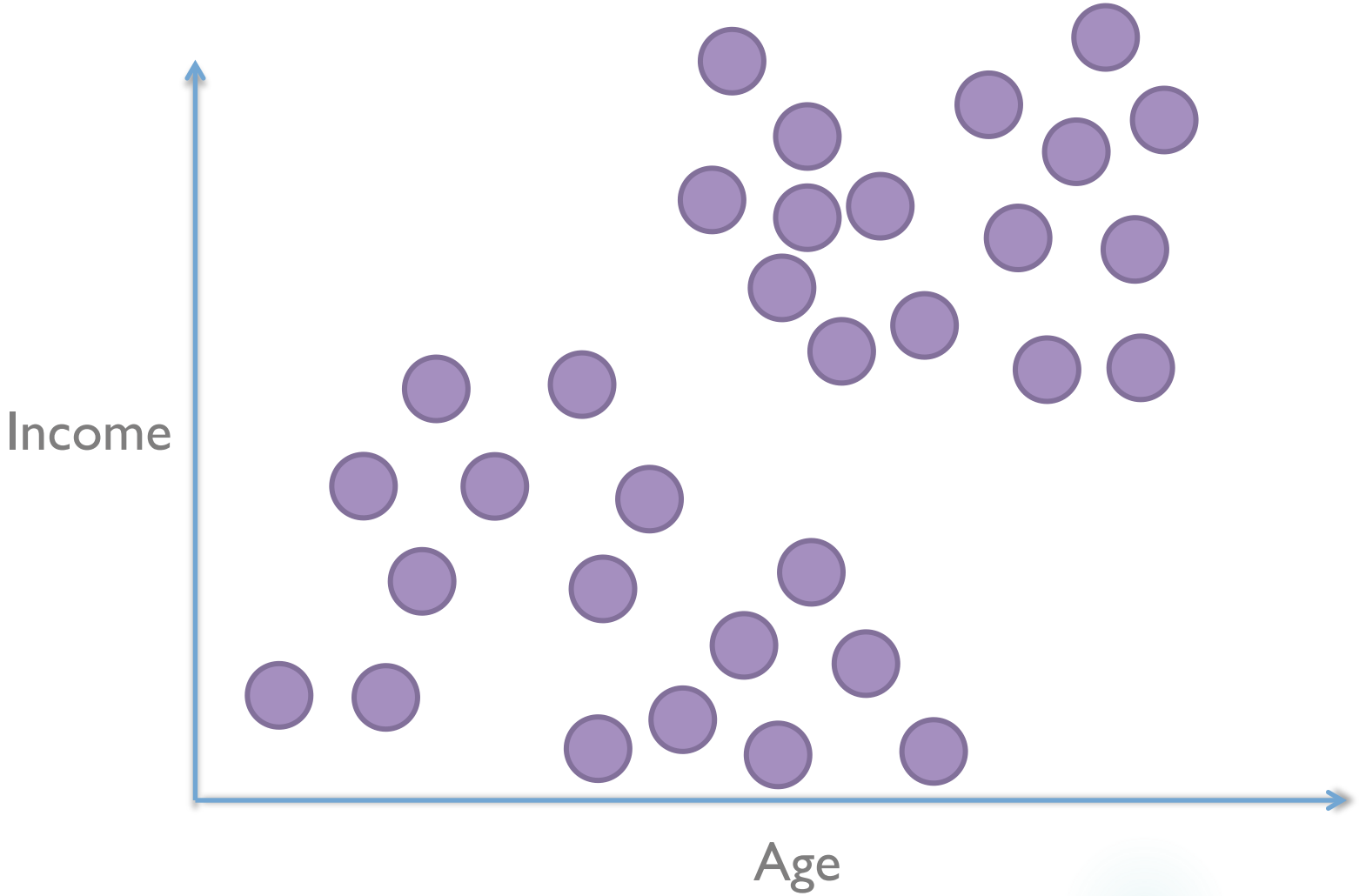
Density-reacheable: an ε-neighbor of a core point than has fewer than n_clu neighbors itself

Noise: a point that has no core points in its ε-neighborhood

Clusters: connected core + density-reachable points

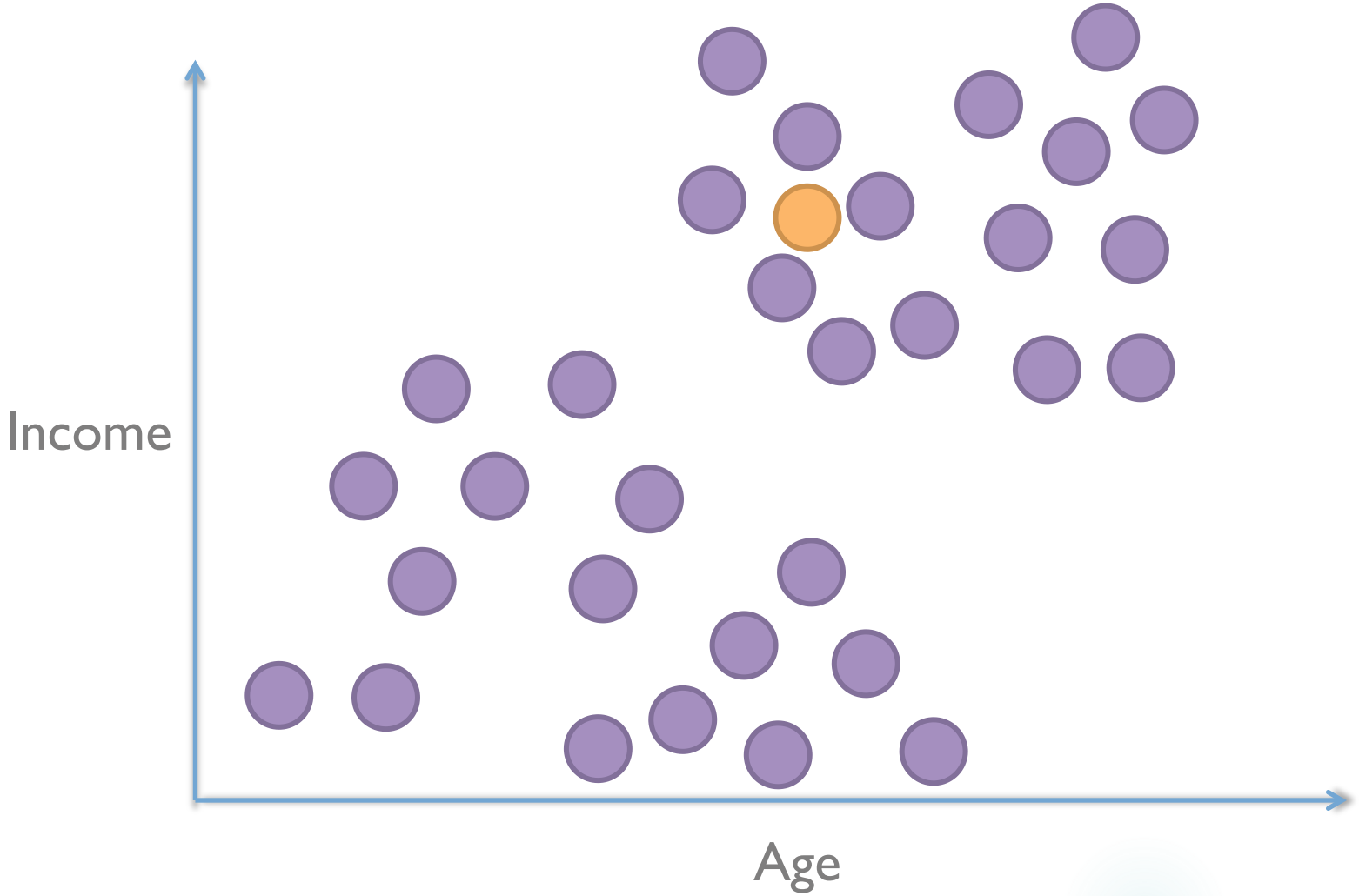
DBSCAN

Density---based spatial clustering of applications with noise



DBSCAN

Start at a random point

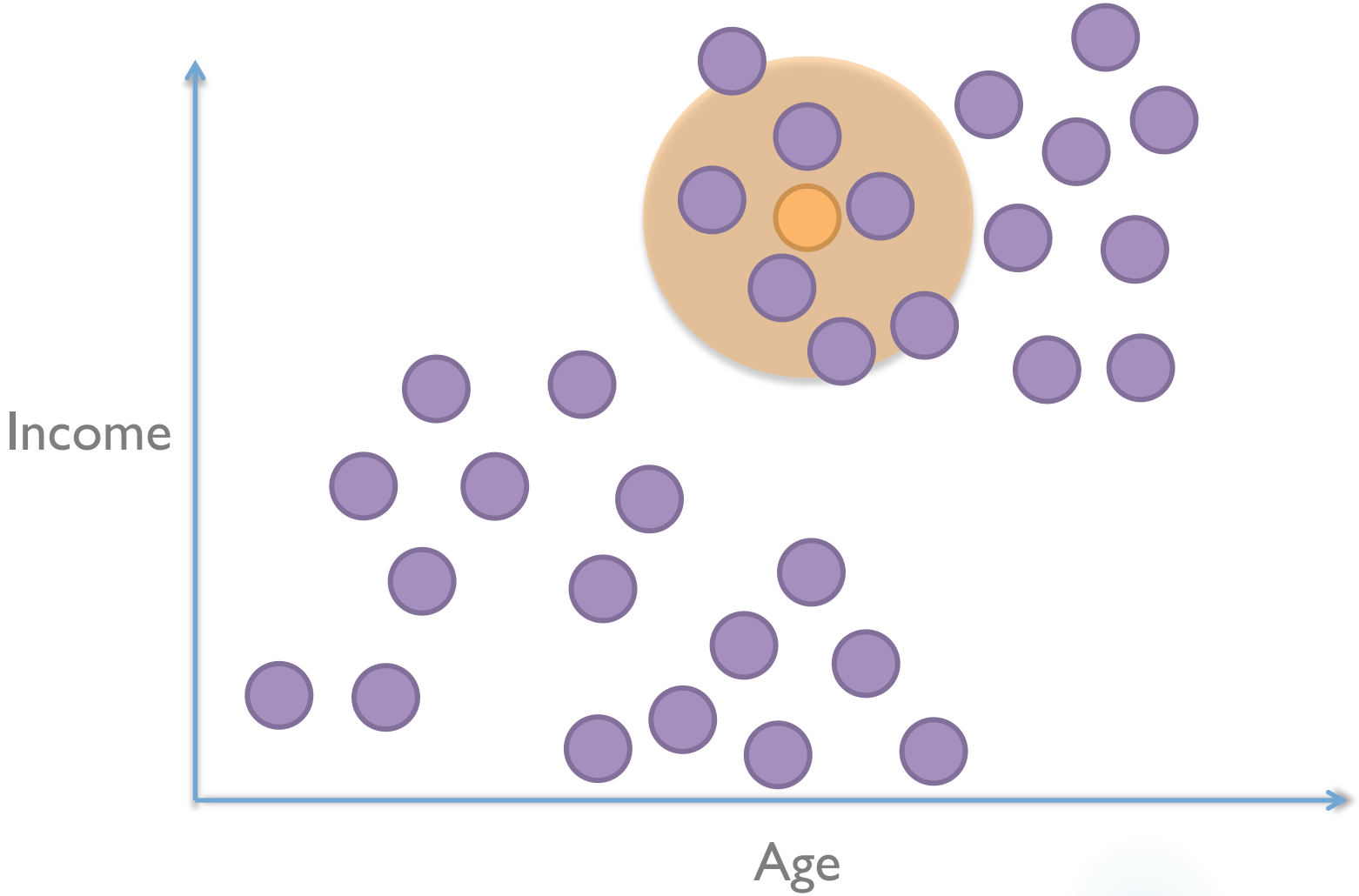


epsilon = 1.75
n_clu = 3

DBSCAN

Look at the radius *epsilon* around point

If enough points (*n_clu*) within circle, start cluster

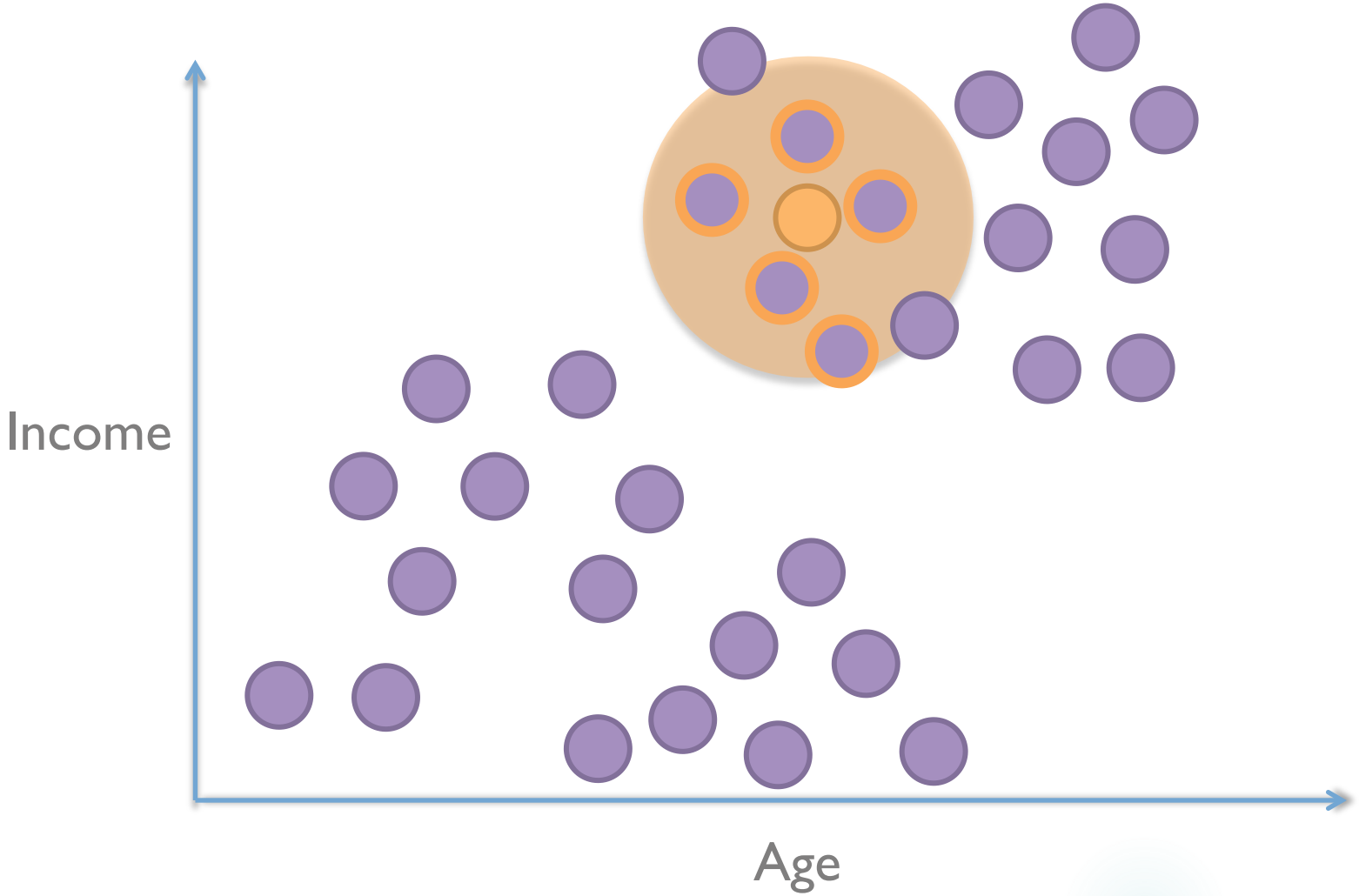


epsilon = 1.75
n_clu = 3

DBSCAN

Everybody within *epsilon* are part of the cluster.

Process each the same way.

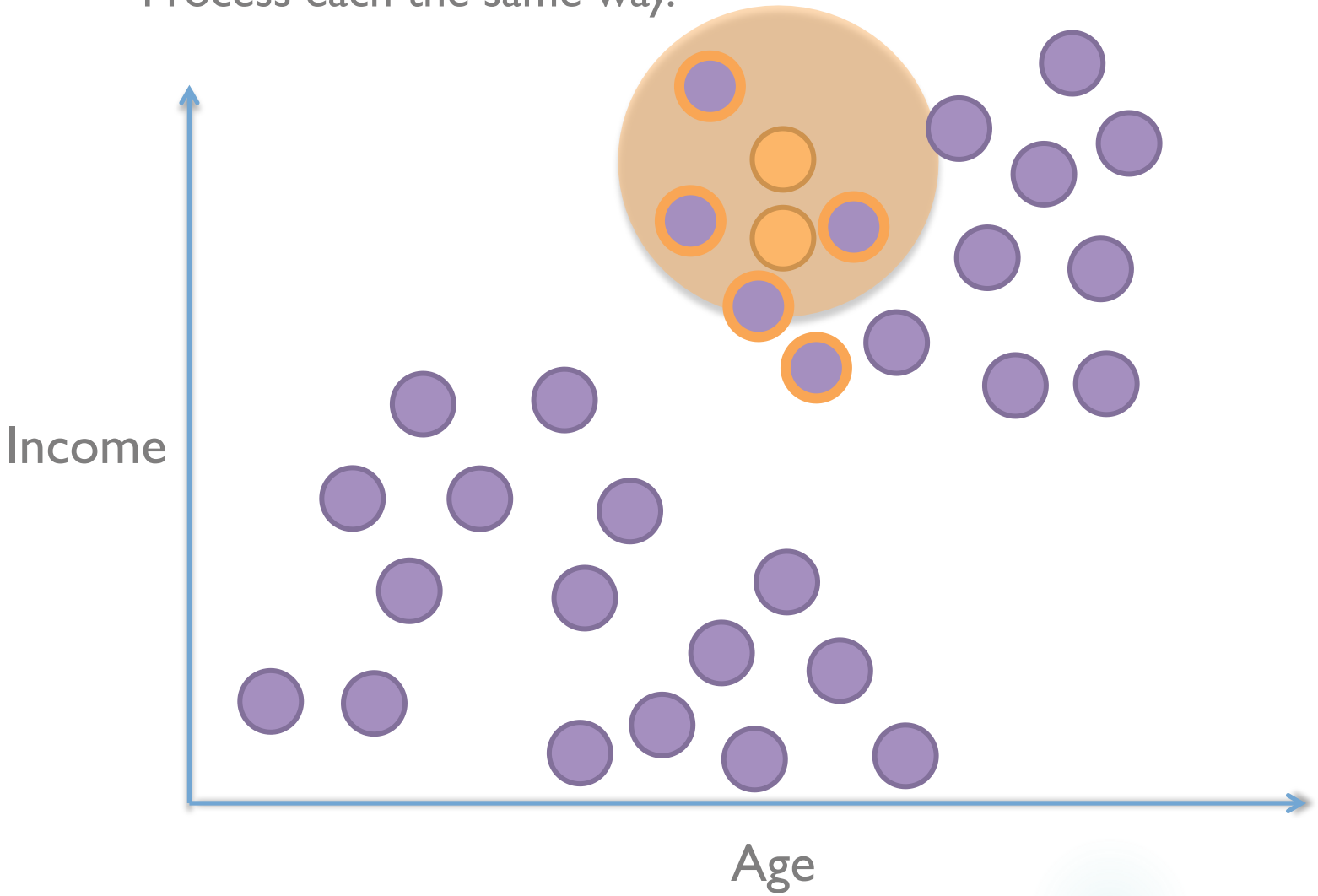


epsilon = 1.75
n_clu = 3

DBSCAN

Everybody within *epsilon* are part of the cluster.

Process each the same way.

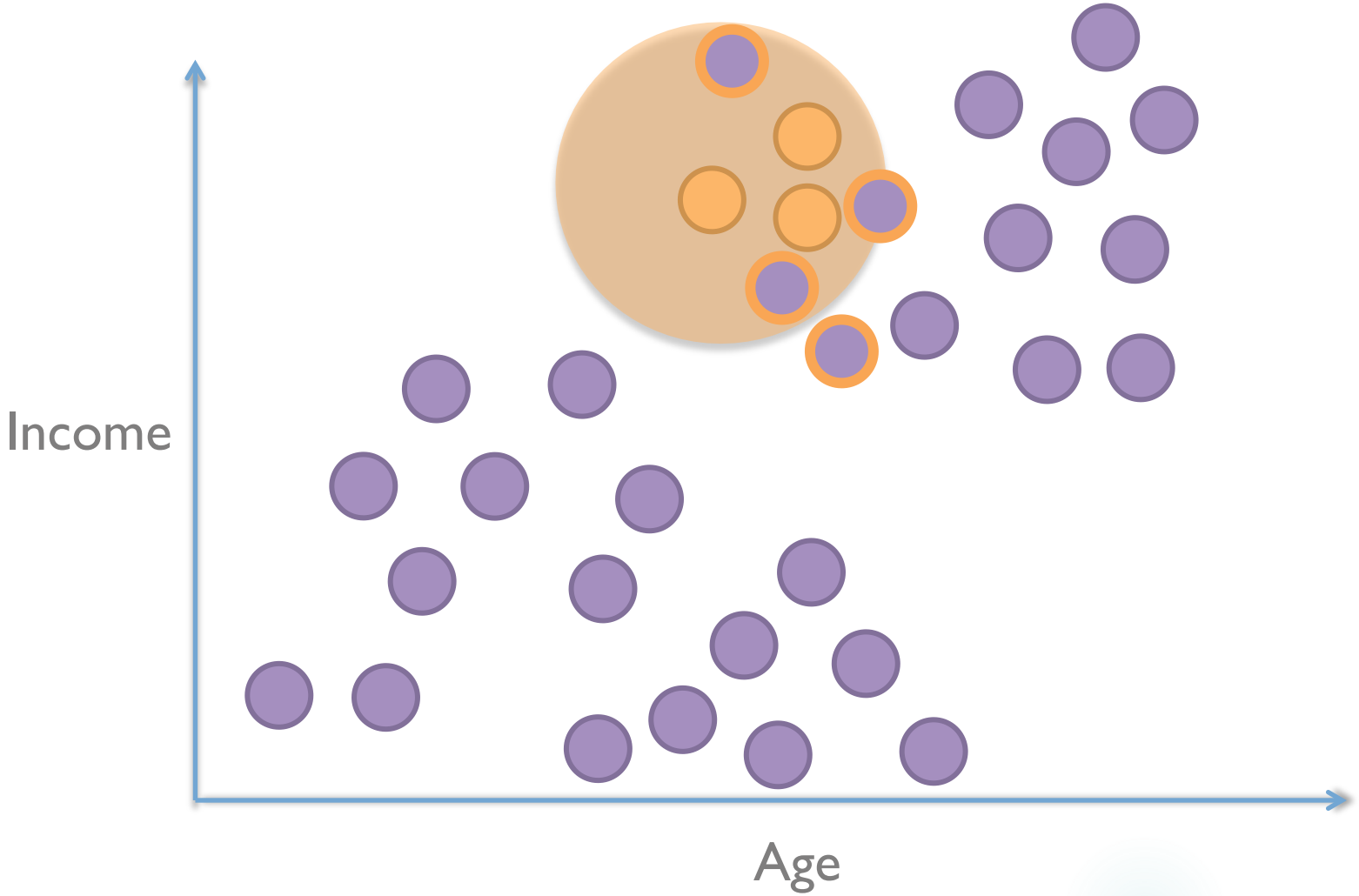


epsilon = 1.75
n_clu = 3

DBSCAN

Everybody within *epsilon* are part of the cluster.

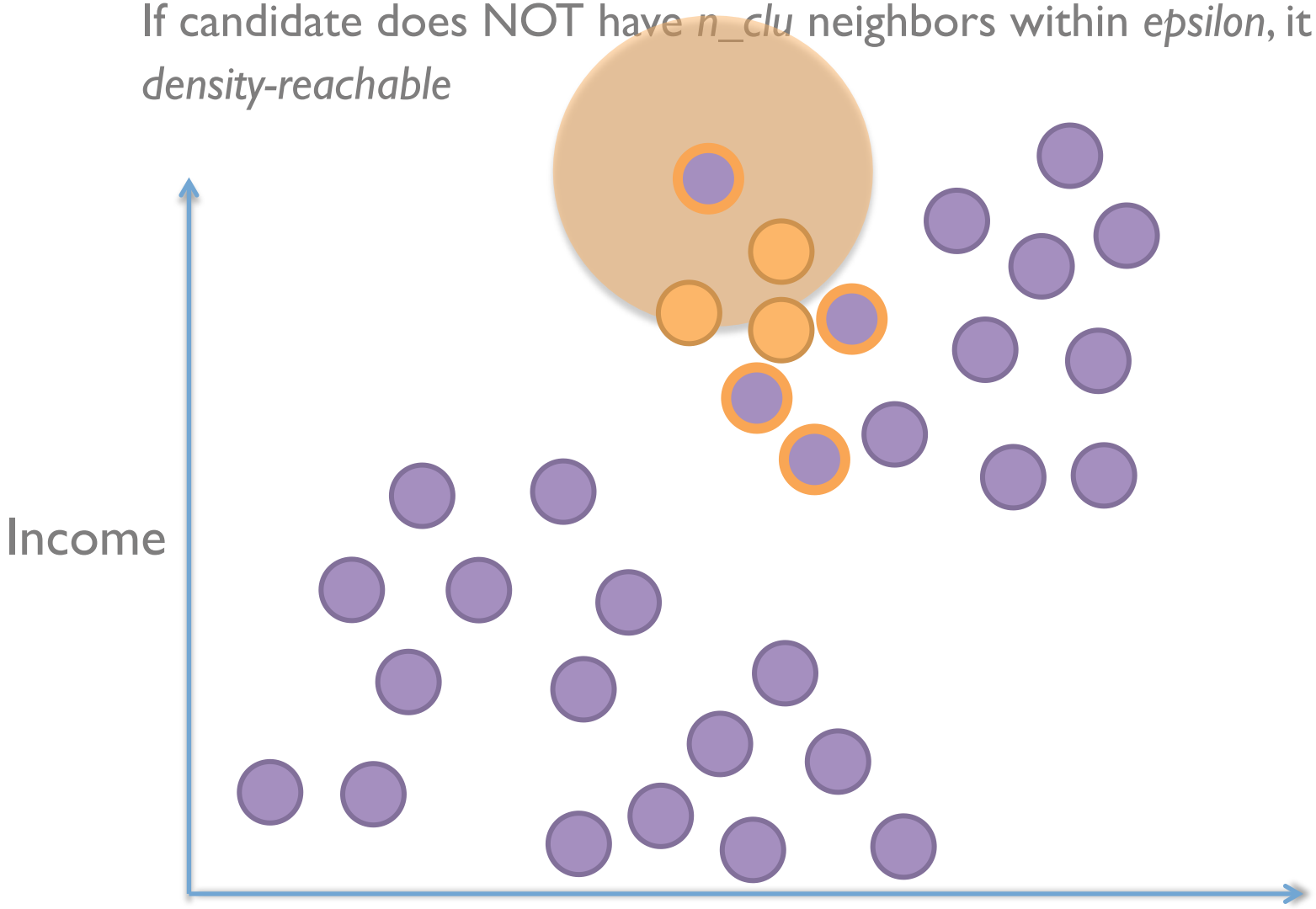
Process each the same way.



epsilon = 1.75
n_clu = 3

DBSCAN

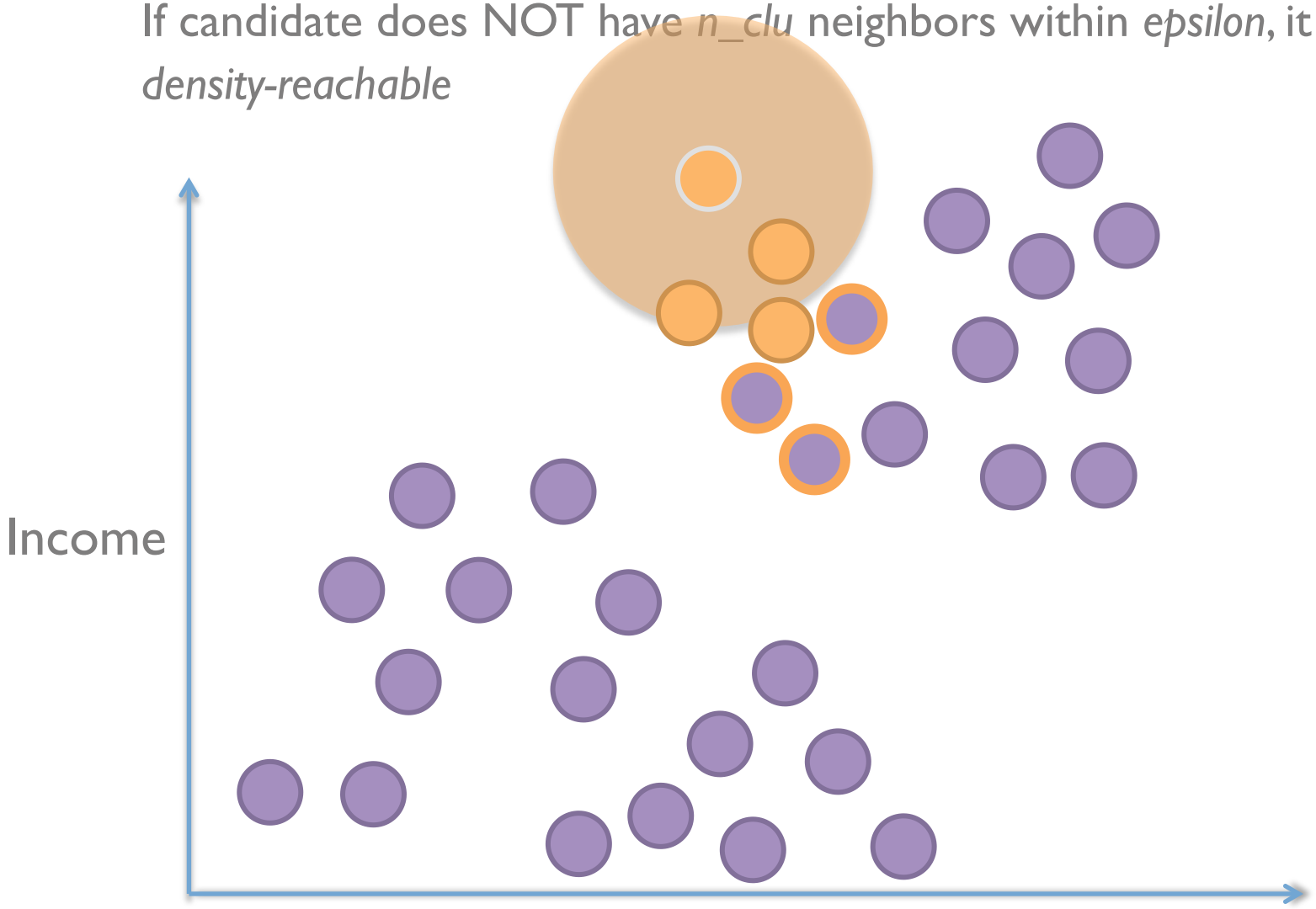
If candidate does NOT have *n_clu* neighbors within *epsilon*, it is *density-reachable*



epsilon = 1.75
n_clu = 3

DBSCAN

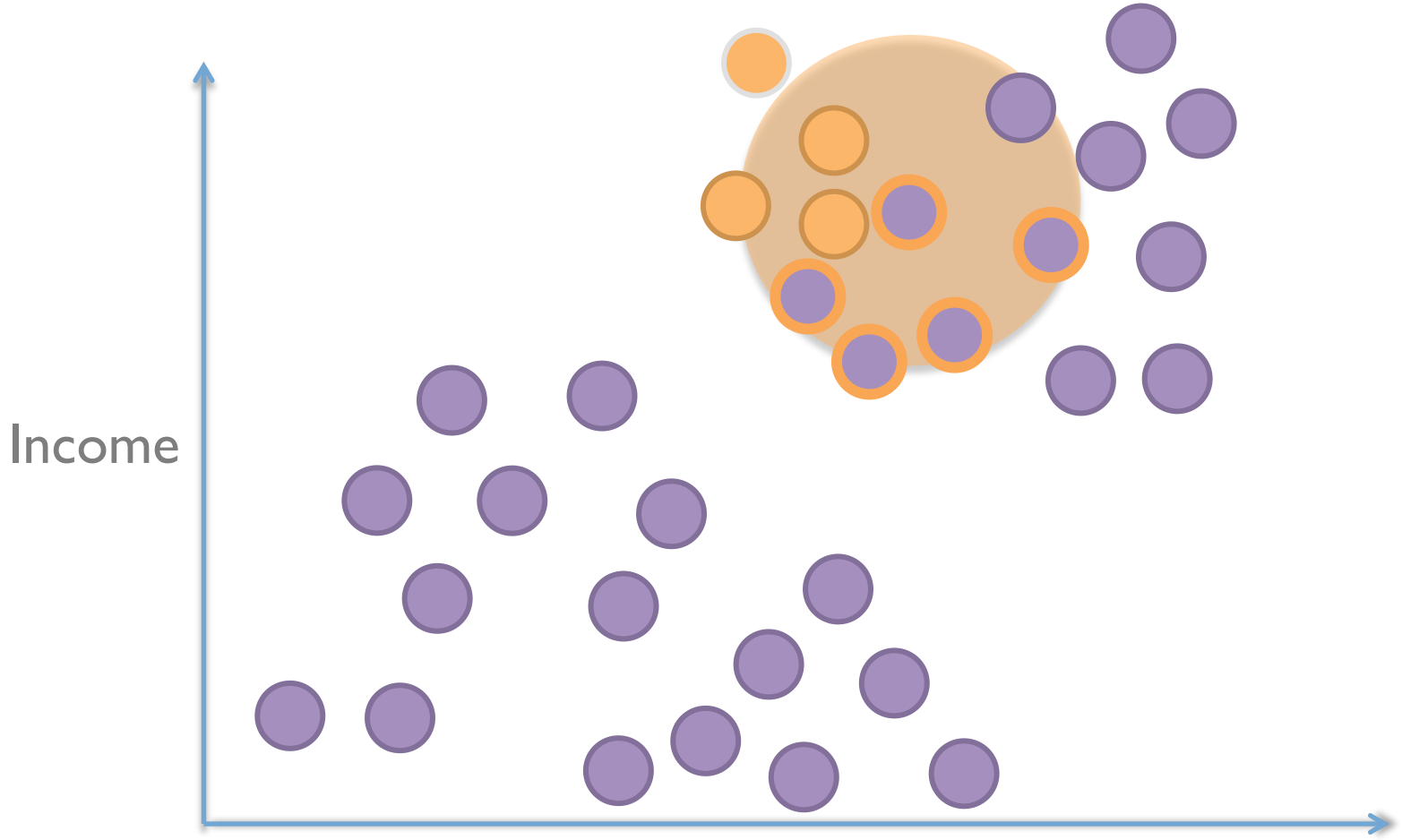
If candidate does NOT have *n_clu* neighbors within *epsilon*, it is *density-reachable*



epsilon = 1.75
n_clu = 3

DBSCAN

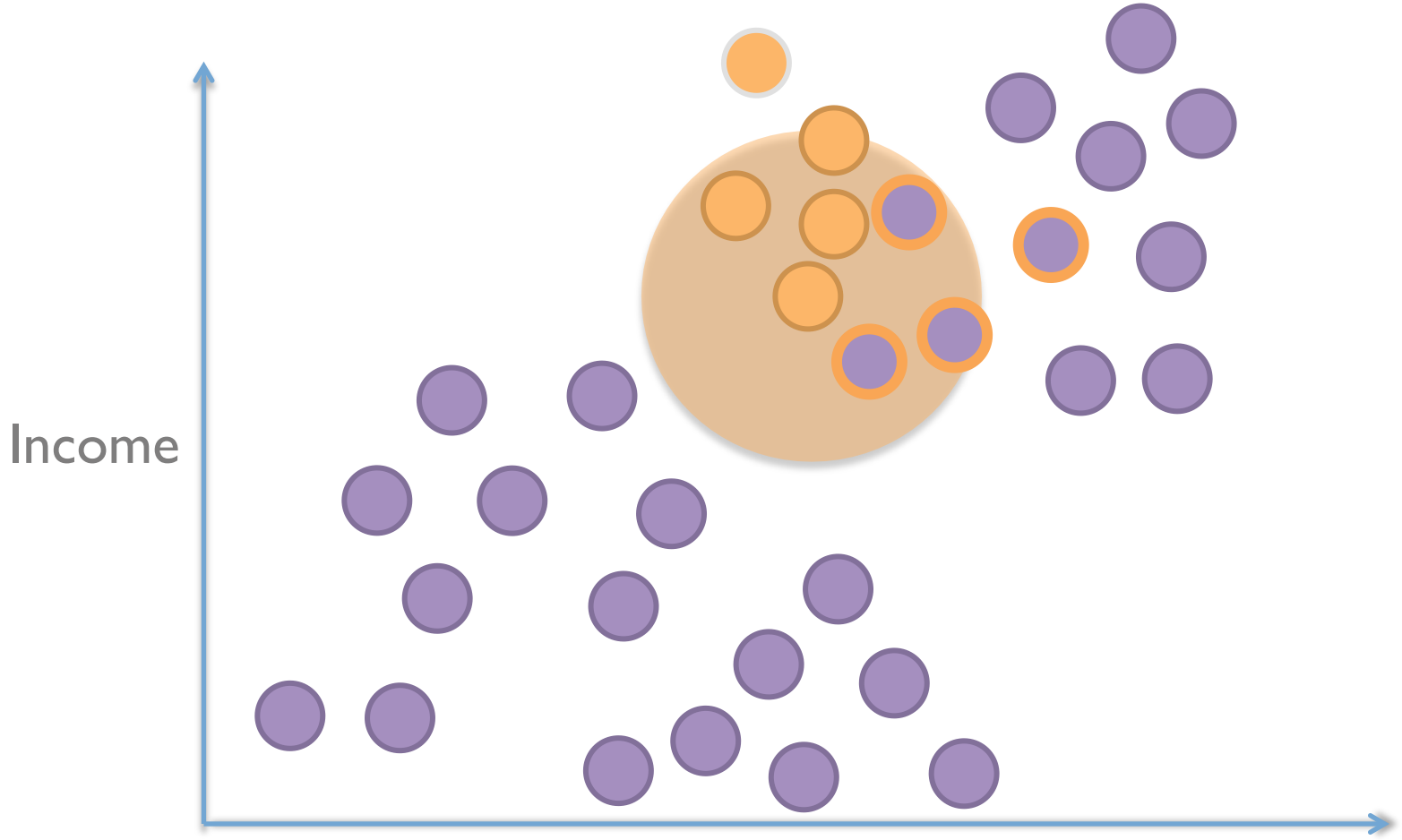
Keep going until you find the entire cluster and no point is left unvisited by this chain reaction



epsilon = 1.75
n_clu = 3

DBSCAN

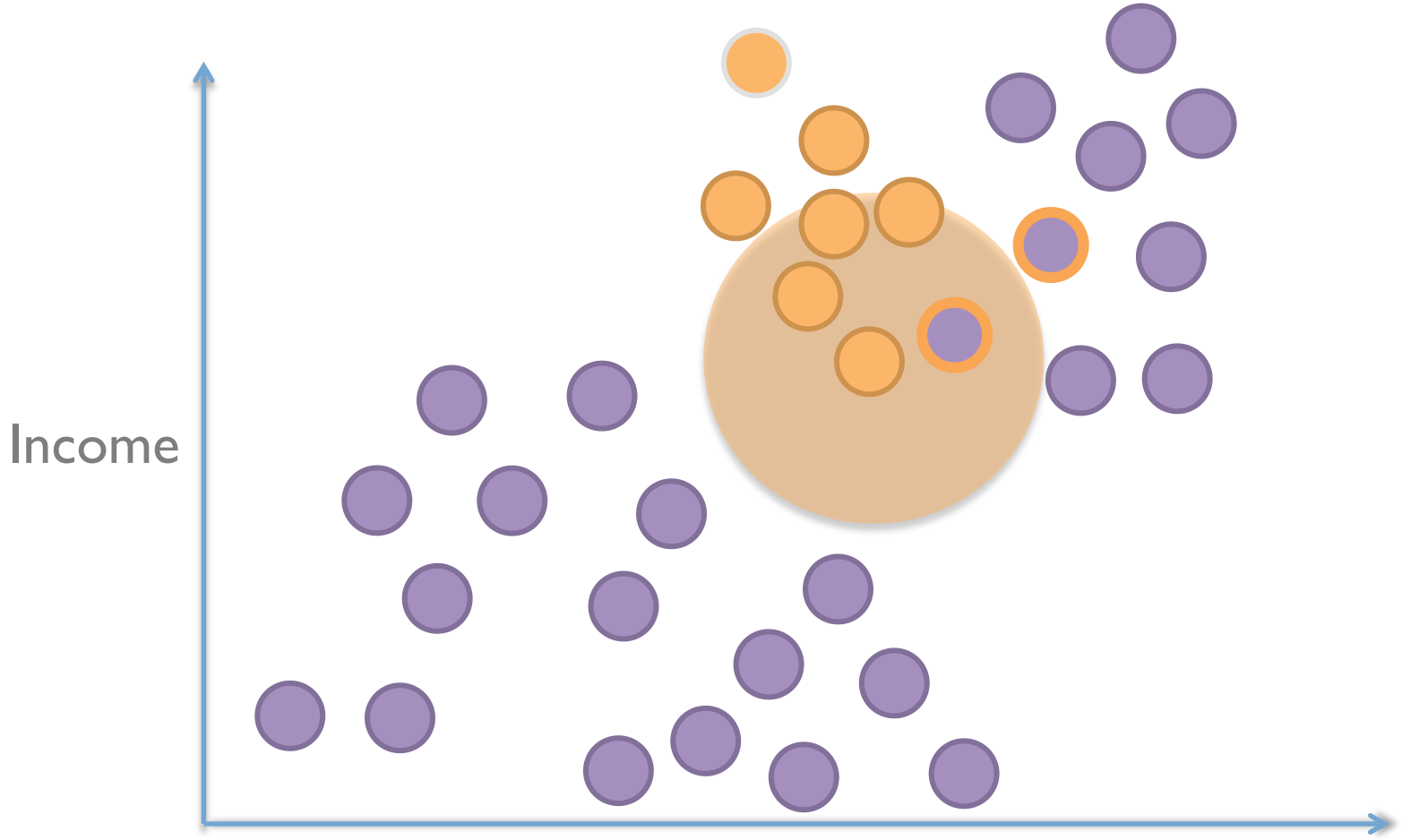
Keep going until you find the entire cluster and no point is left unvisited by this chain reaction



epsilon = 1.75
n_clu = 3

DBSCAN

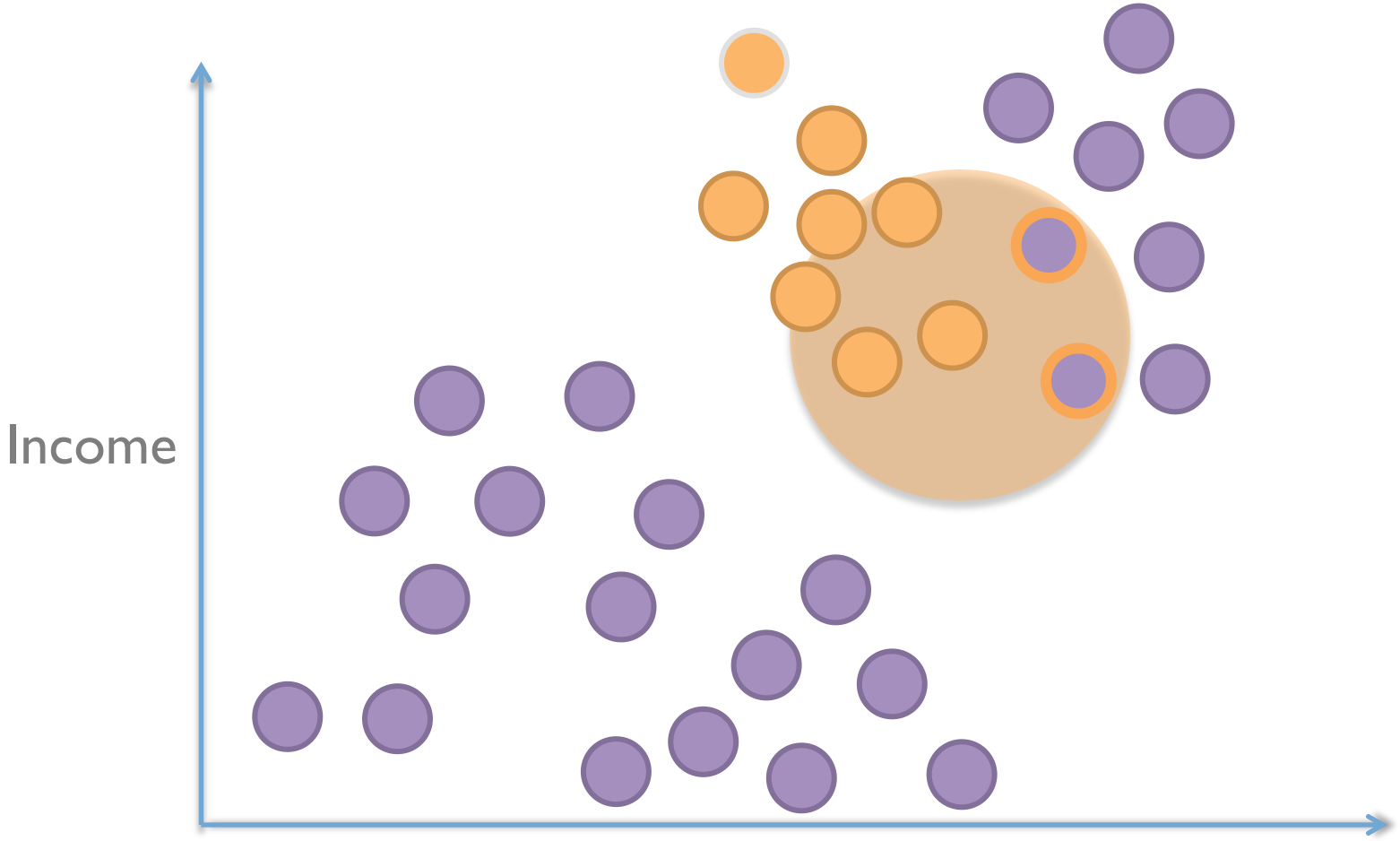
Keep going until you find the entire cluster and no point is left unvisited by this chain reaction



epsilon = 1.75
n_clu = 3

DBSCAN

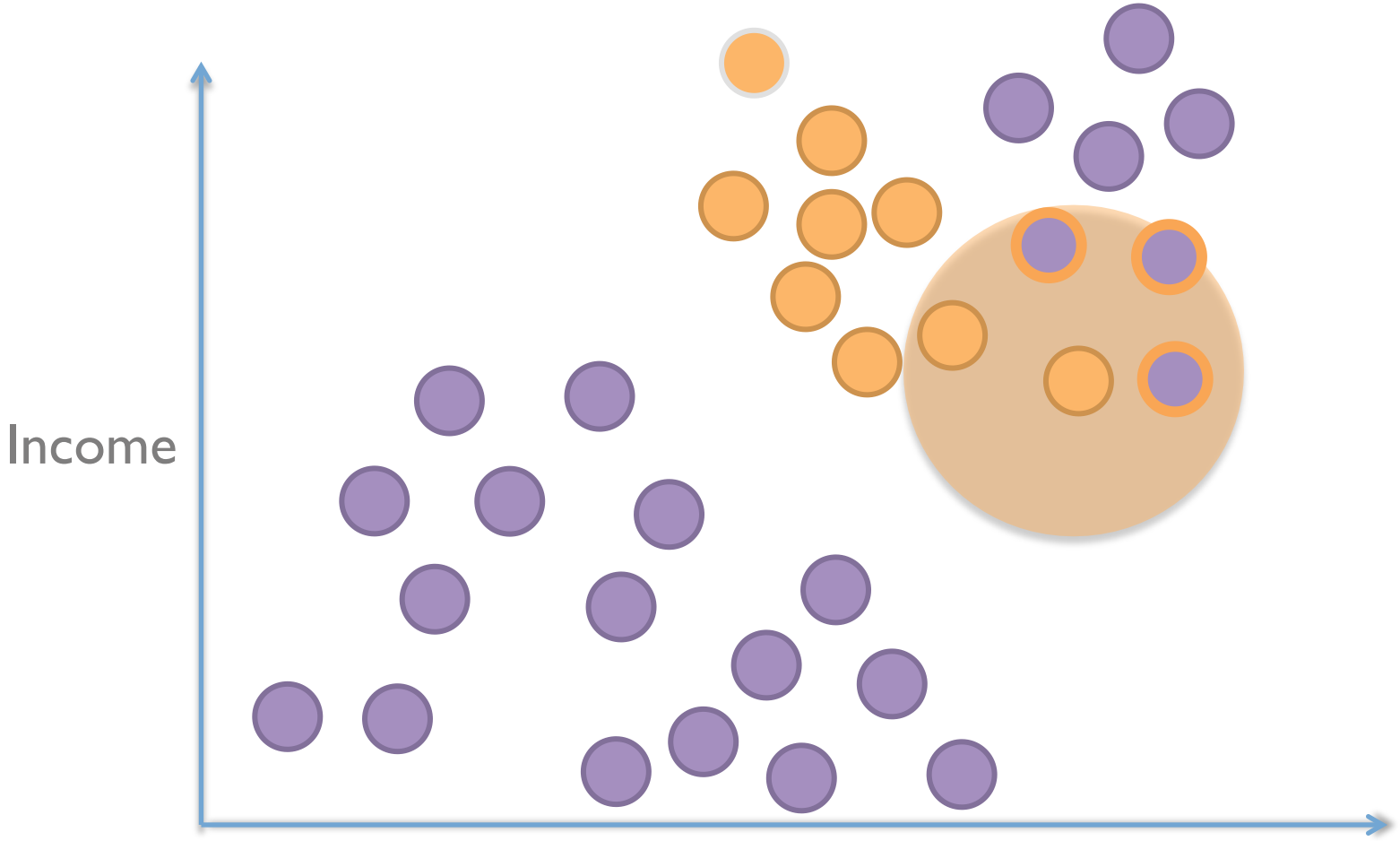
Keep going until you find the entire cluster and no point is left unvisited by this chain reaction



epsilon = 1.75
n_clu = 3

DBSCAN

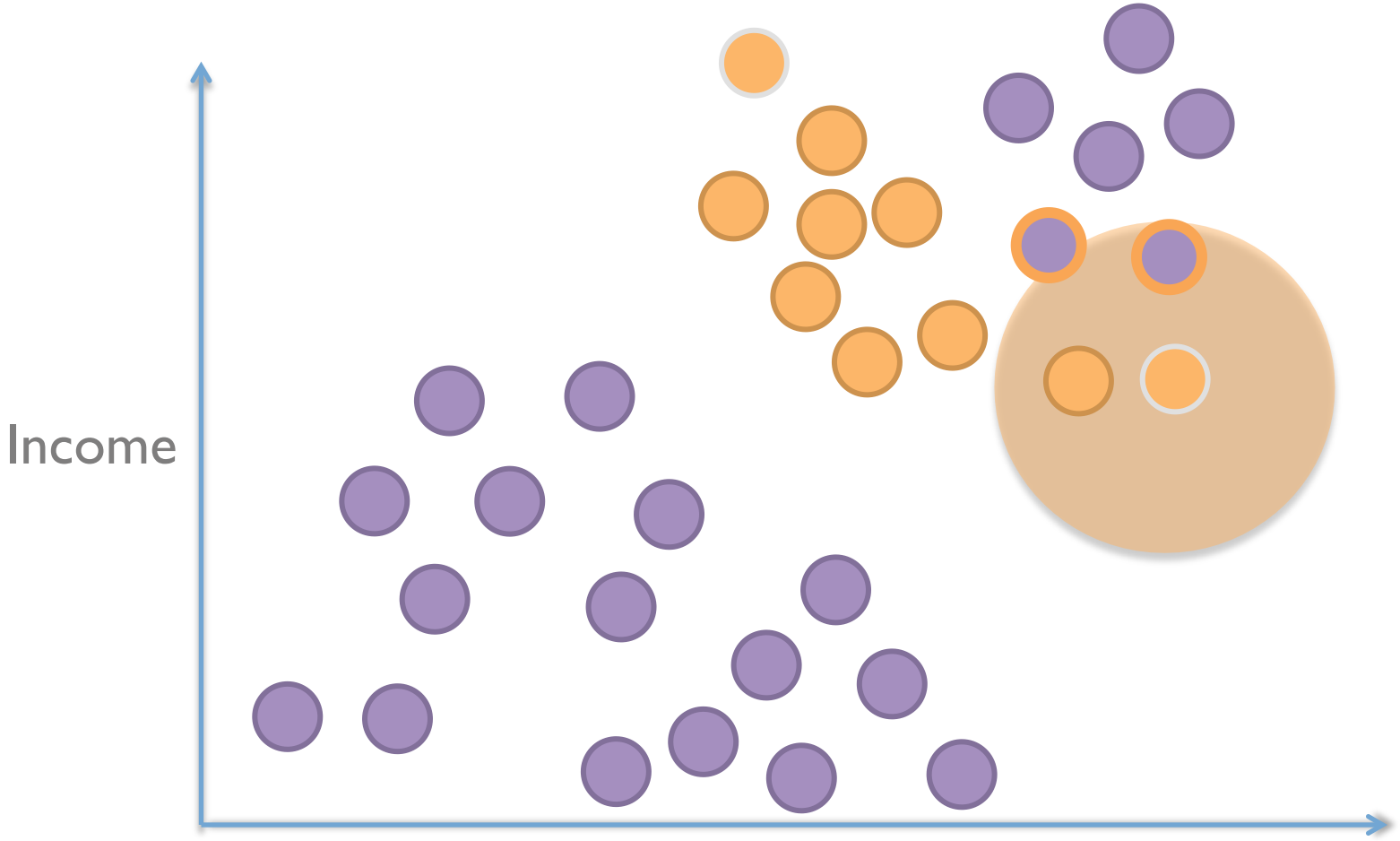
Keep going until you find the entire cluster and no point is left unvisited by this chain reaction



epsilon = 1.75
n_clu = 3

DBSCAN

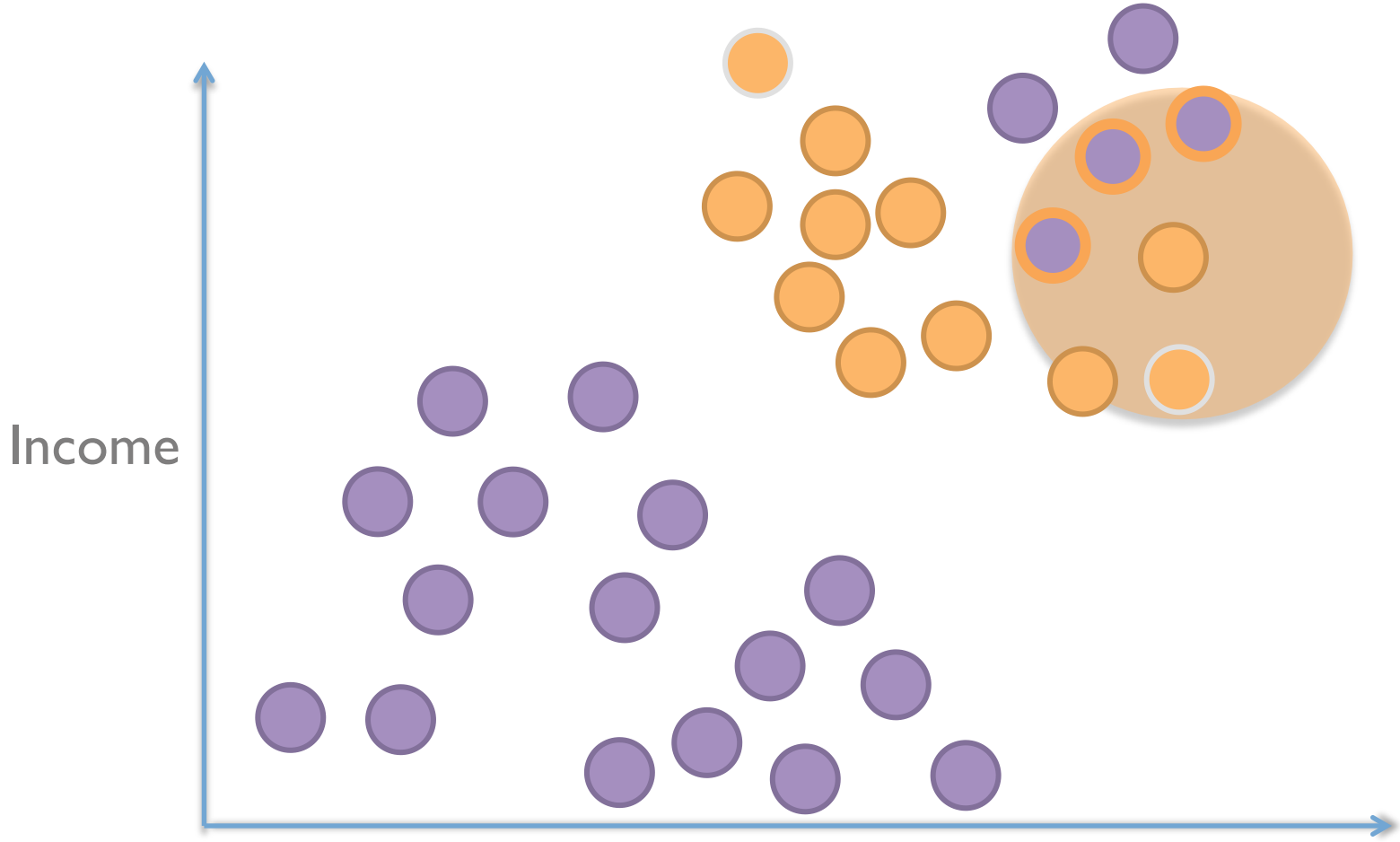
Keep going until you find the entire cluster and no point is left unvisited by this chain reaction



epsilon = 1.75
n_clu = 3

DBSCAN

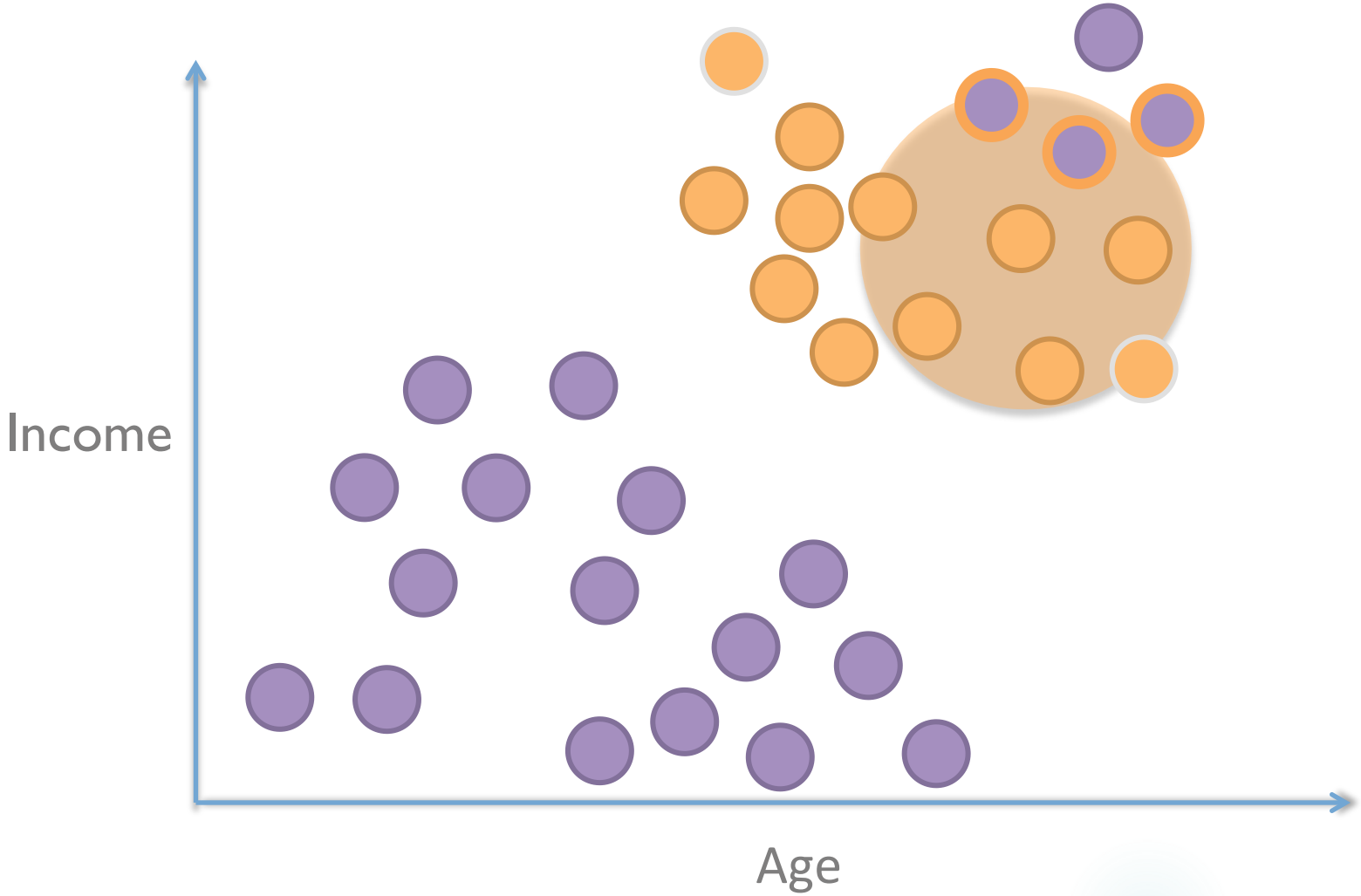
Keep going until you find the entire cluster and no point is left unvisited by this chain reaction



epsilon = 1.75
n_clu = 3

DBSCAN

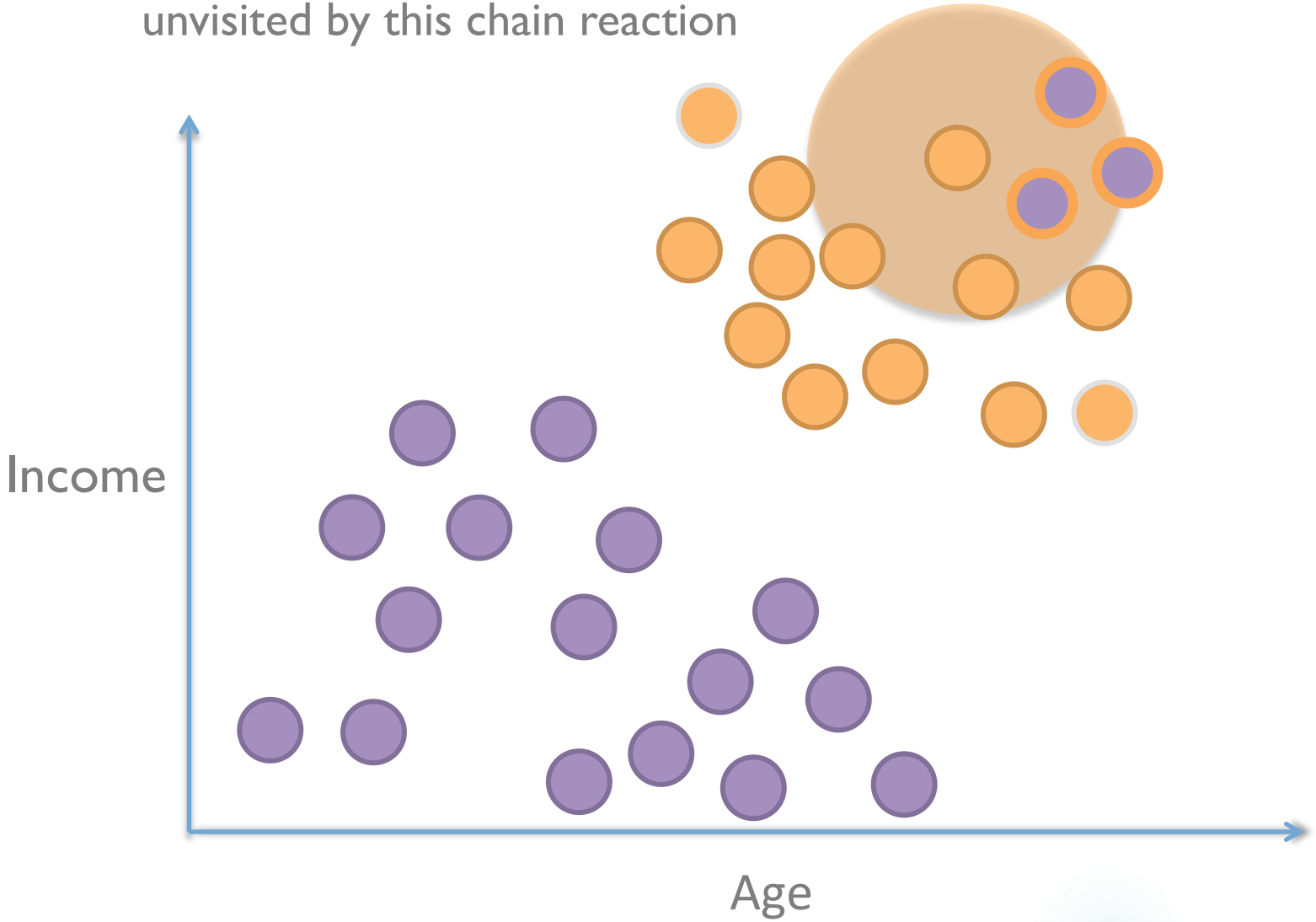
Keep going until you find the entire cluster and no point is left unvisited by this chain reaction



epsilon = 1.75
n_clu = 3

DBSCAN

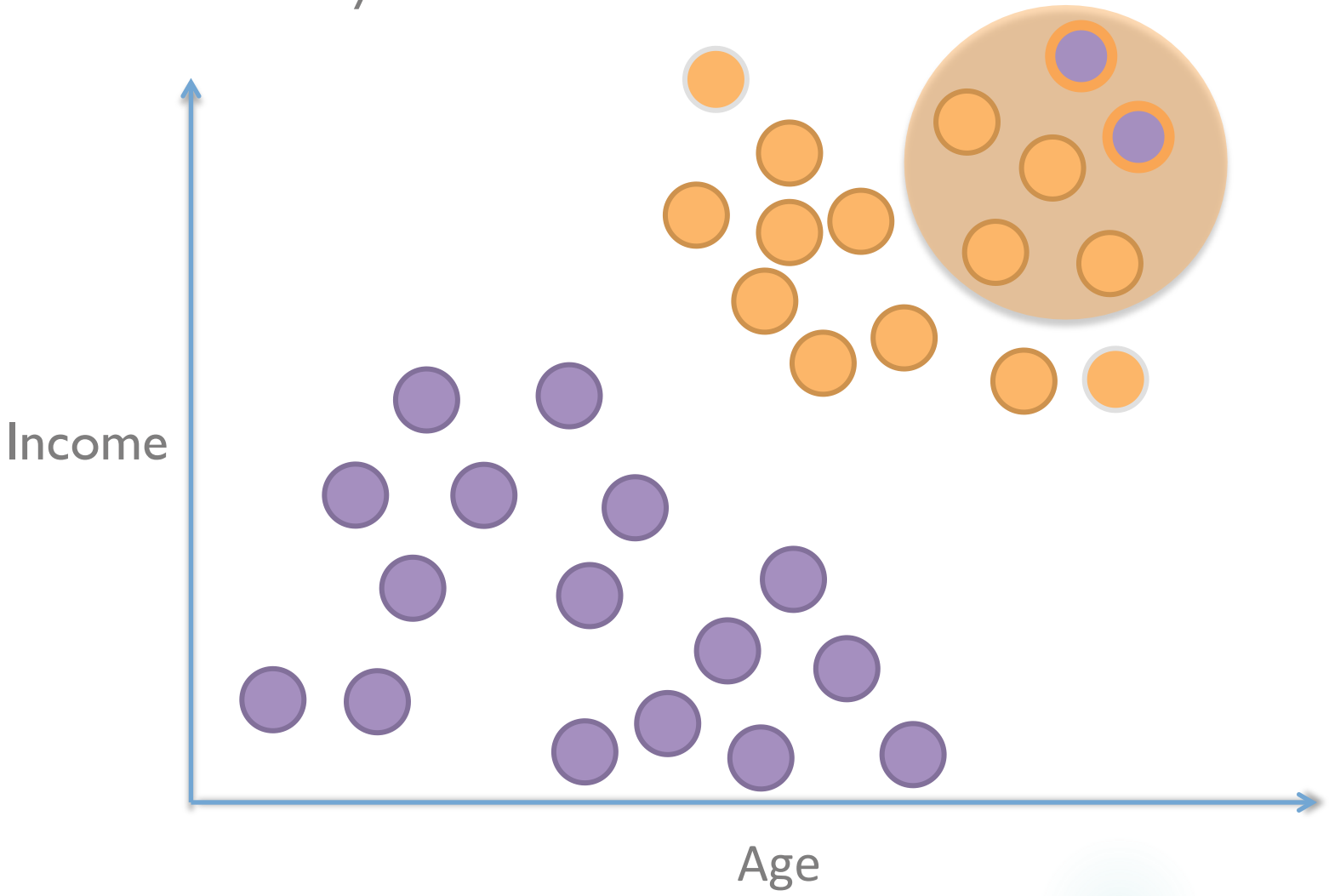
Keep going until you find the entire cluster and no point is left unvisited by this chain reaction



epsilon = 1.75
n_clu = 3

DBSCAN

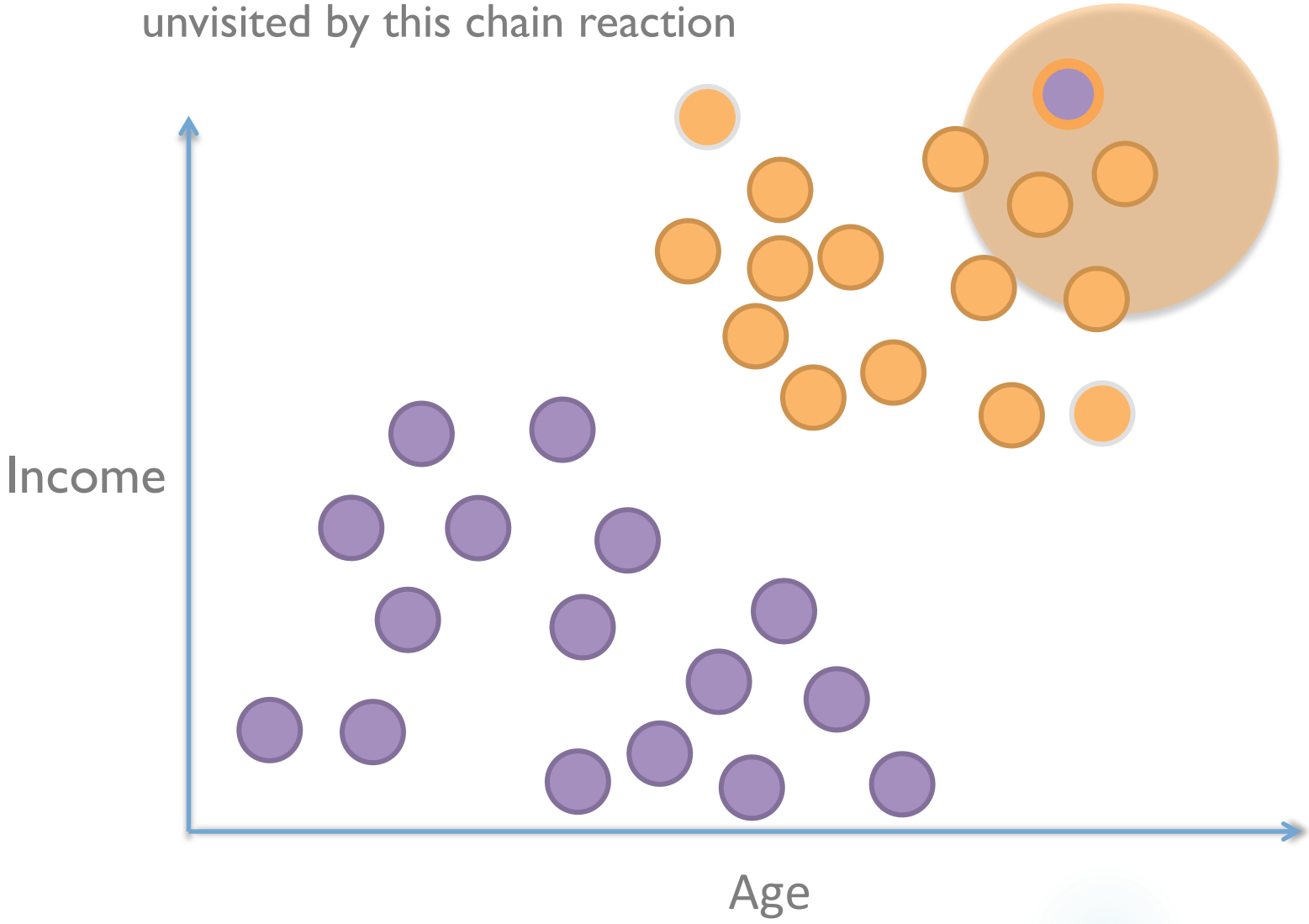
Keep going until you find the entire cluster and no point is left unvisited by this chain reaction



epsilon = 1.75
n_clu = 3

DBSCAN

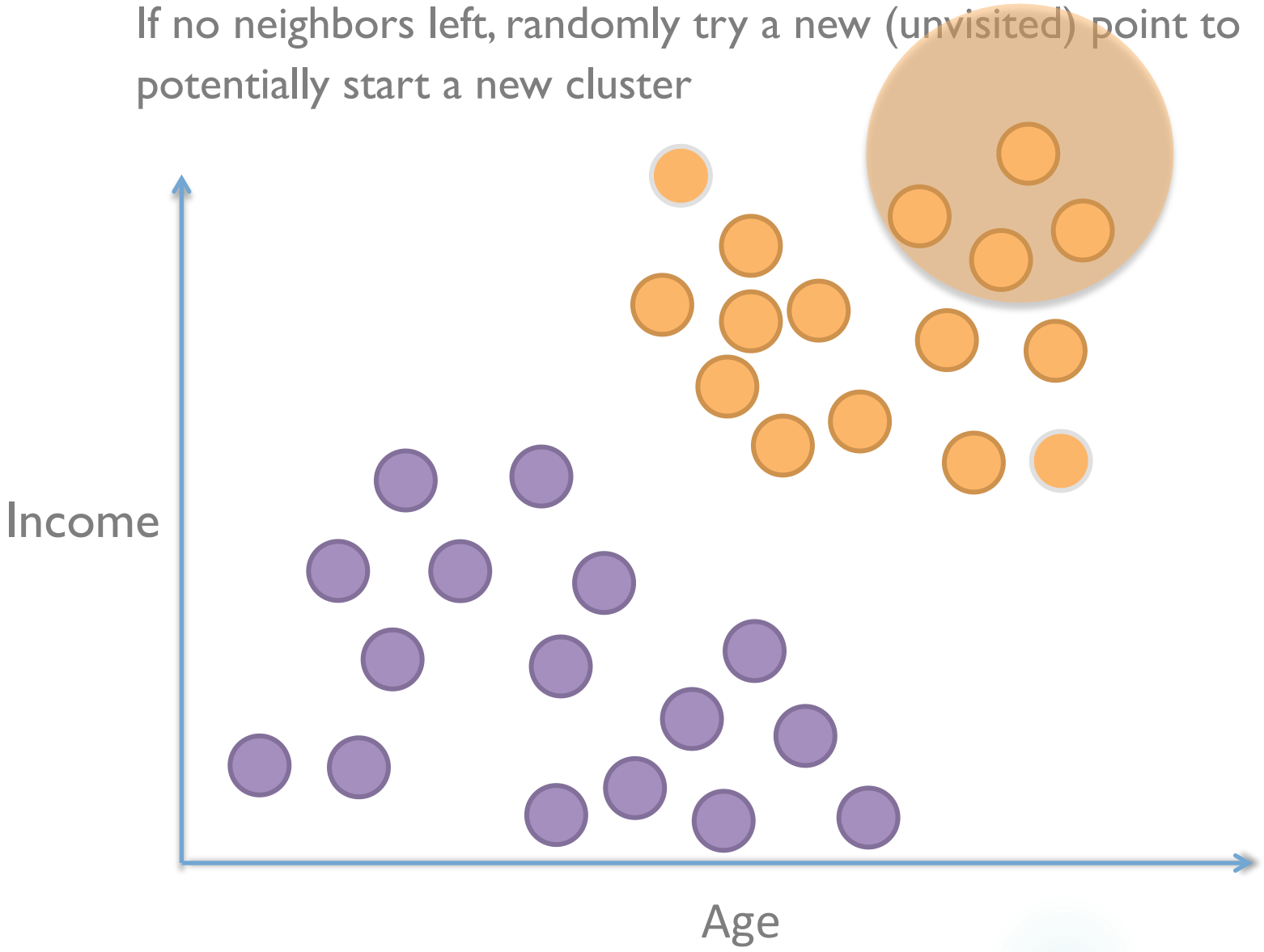
Keep going until you find the entire cluster and no point is left unvisited by this chain reaction



epsilon = 1.75
n_clu = 3

DBSCAN

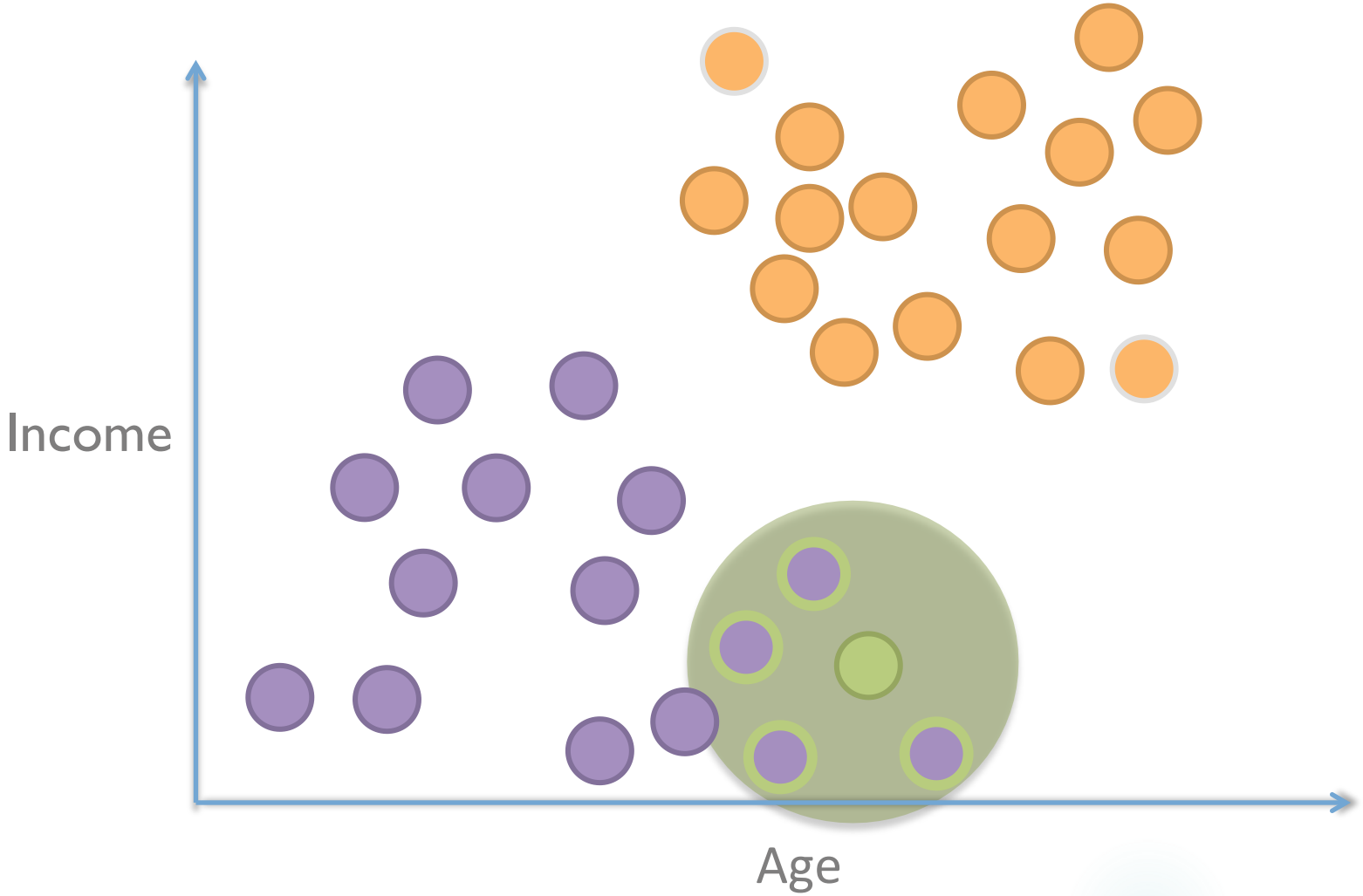
If no neighbors left, randomly try a new (unvisited) point to potentially start a new cluster



epsilon = 1.75
n_clu = 3

DBSCAN

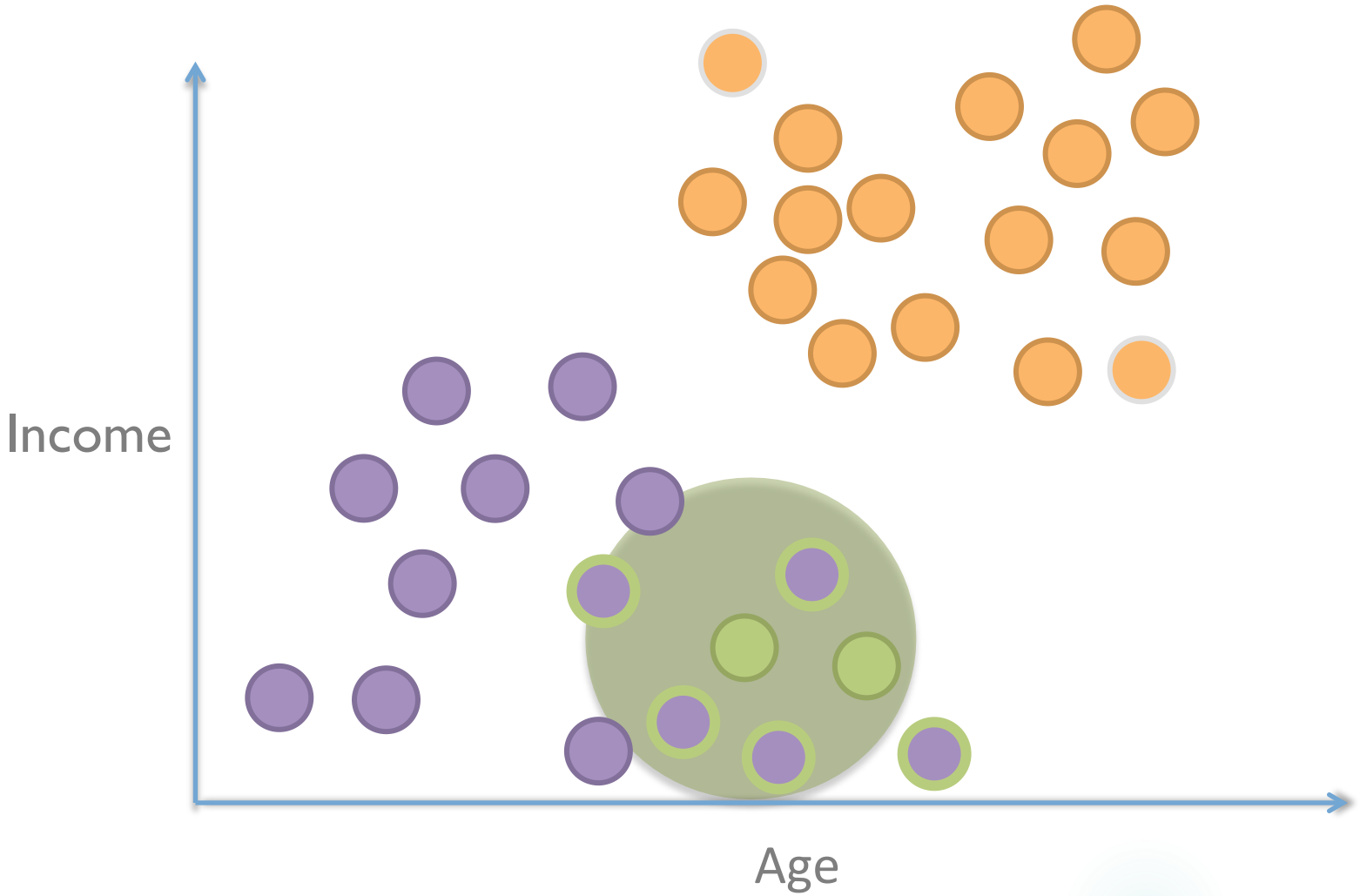
If no neighbors left, randomly try a new (unvisited) point to potentially start a new cluster



epsilon = 1.75
n_clu = 3

DBSCAN

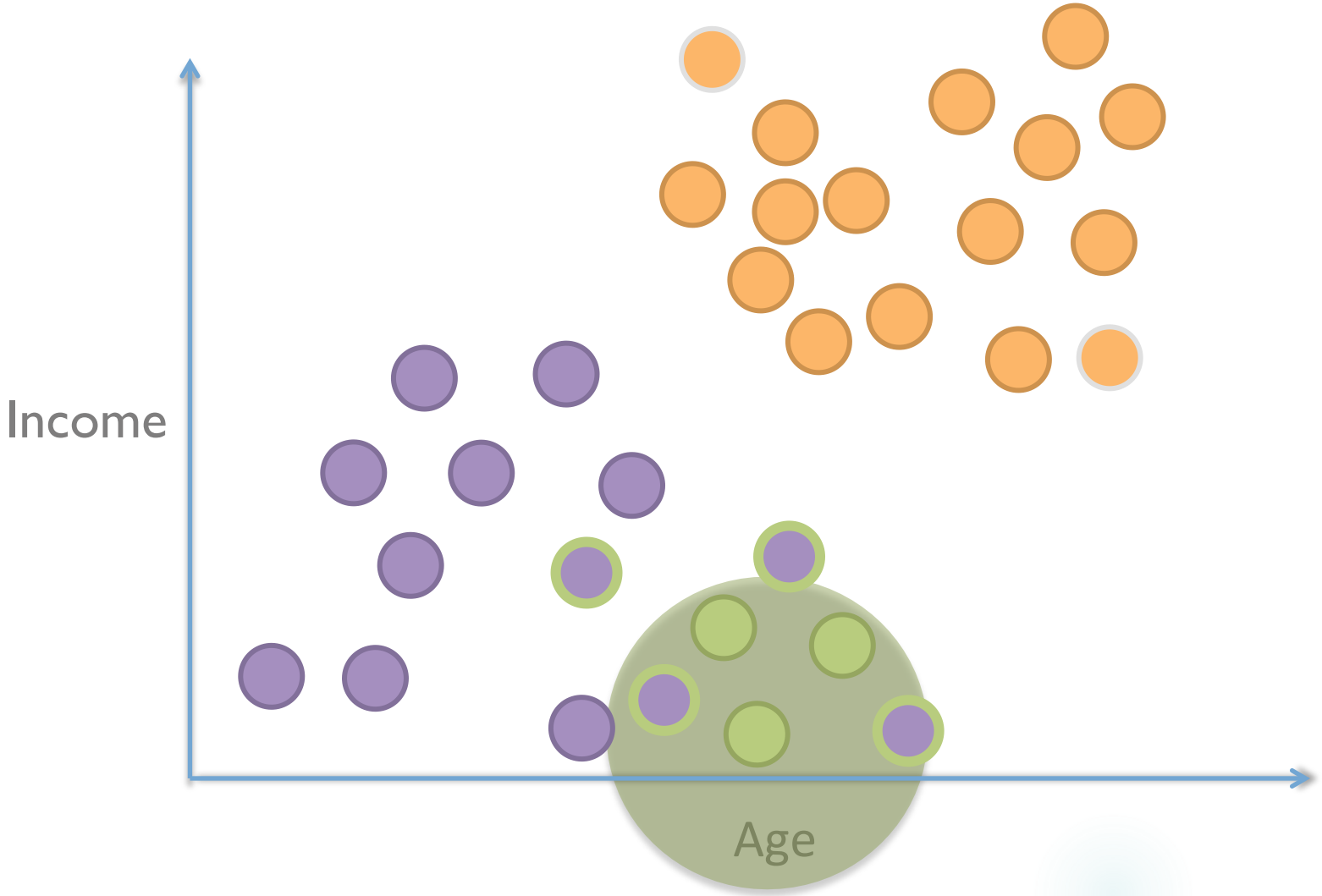
Keep adding neighbors within *epsilon* as chain reaction



epsilon = 1.75
n_clu = 3

DBSCAN

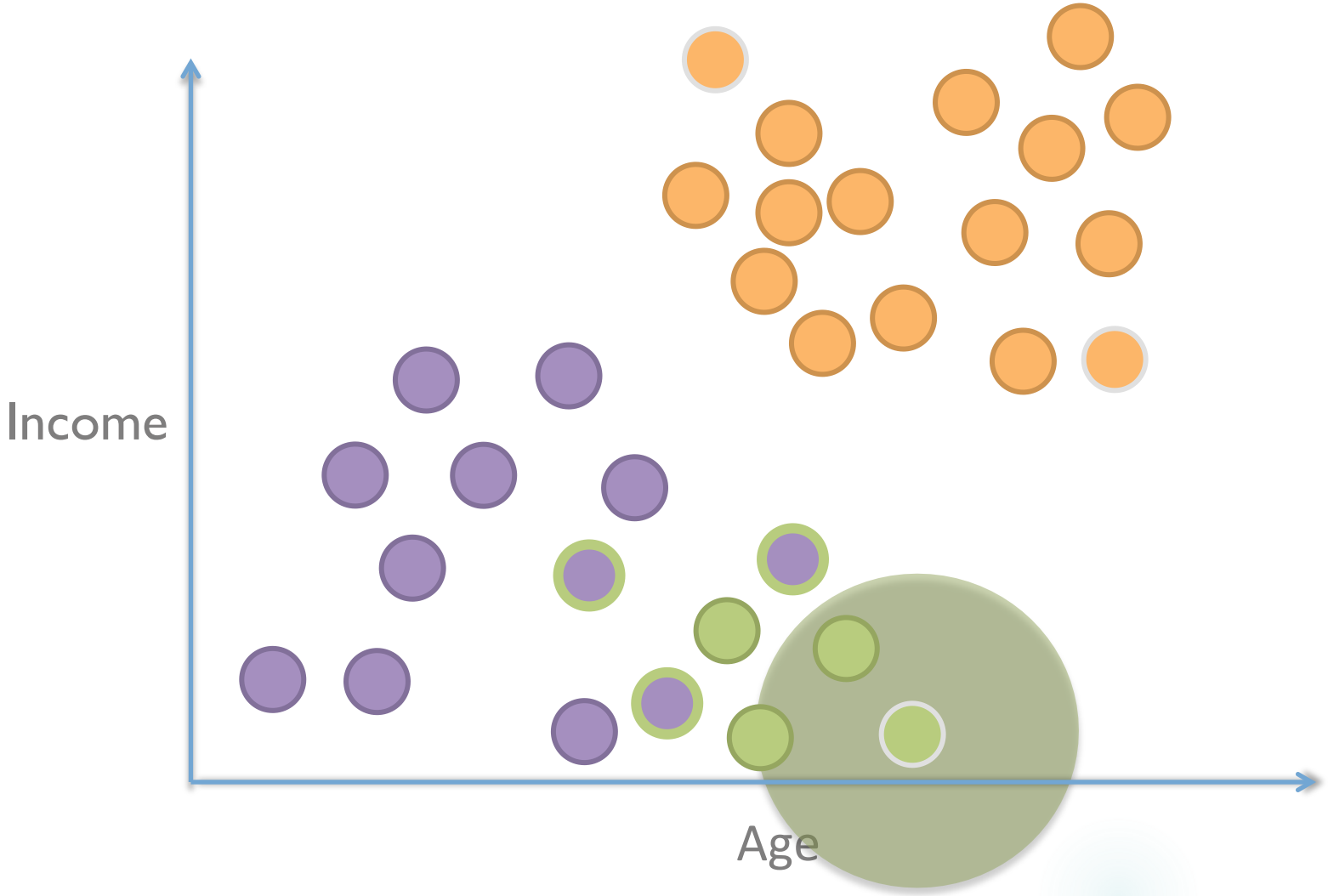
Keep adding neighbors within *epsilon* as chain reaction



epsilon = 1.75
n_clu = 3

DBSCAN

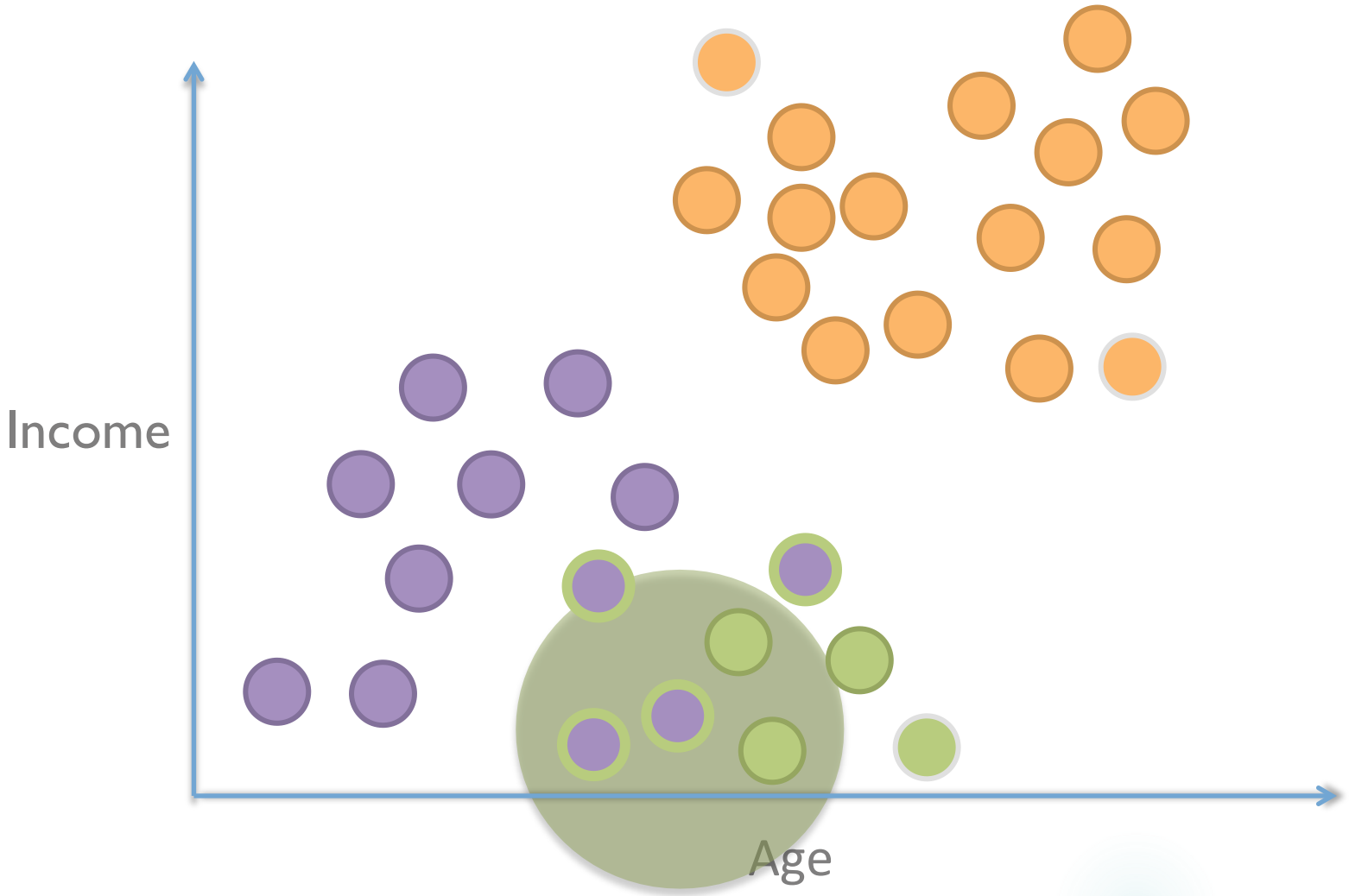
Keep adding neighbors within *epsilon* as chain reaction



epsilon = 1.75
n_clu = 3

DBSCAN

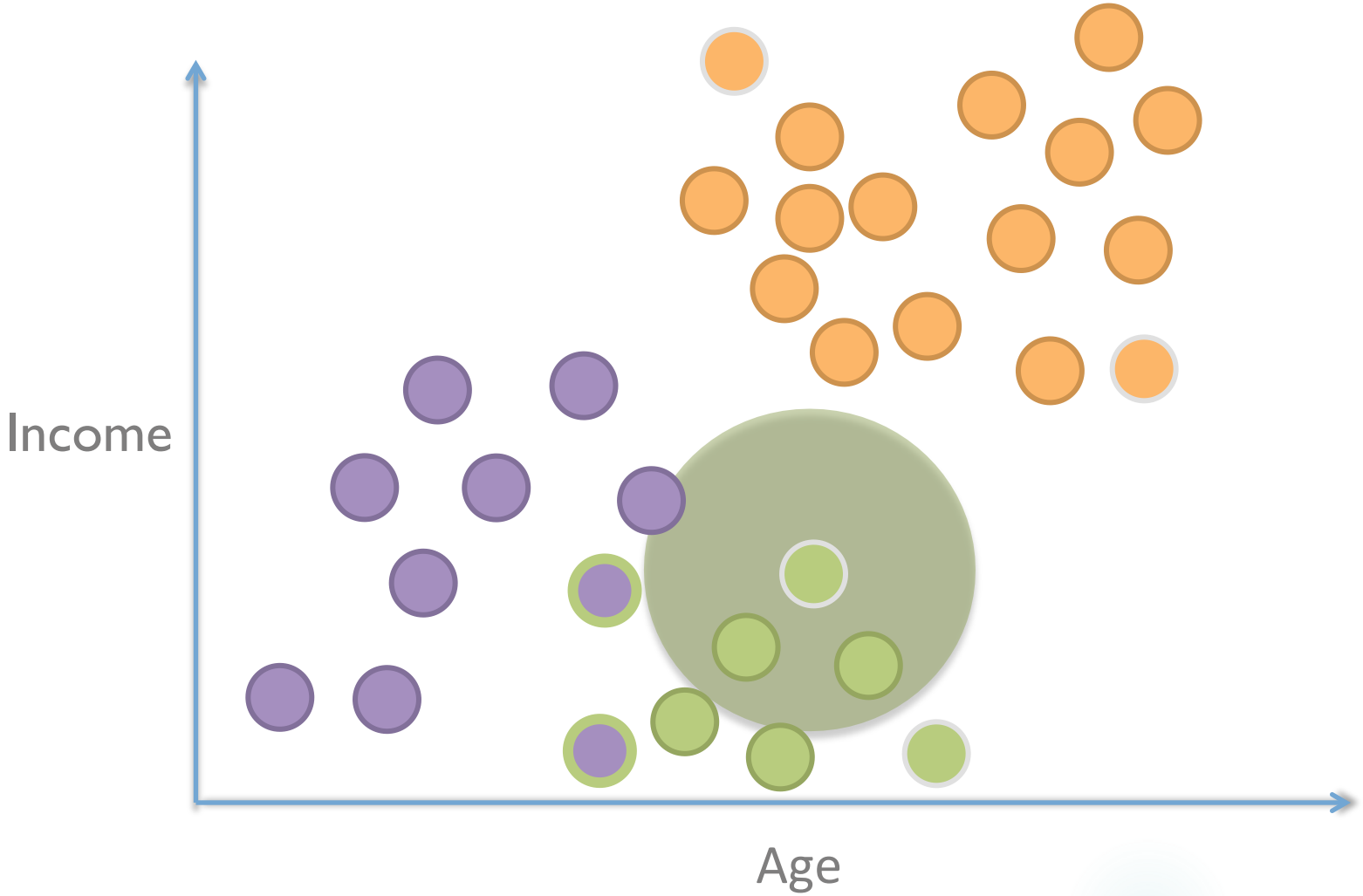
Keep adding neighbors within *epsilon* as chain reaction



epsilon = 1.75
n_clu = 3

DBSCAN

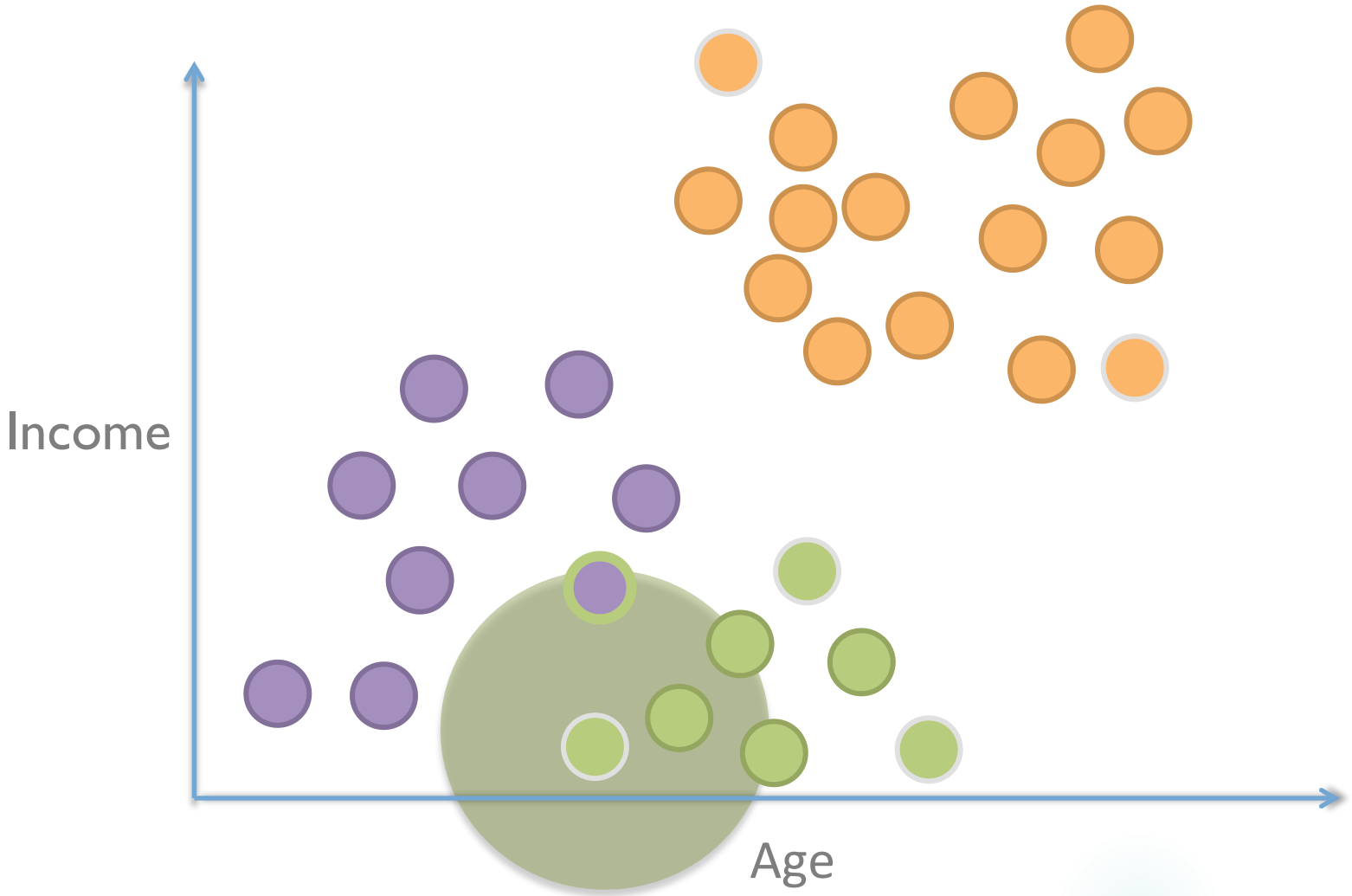
Keep adding neighbors within *epsilon* as chain reaction



epsilon = 1.75
n_clu = 3

DBSCAN

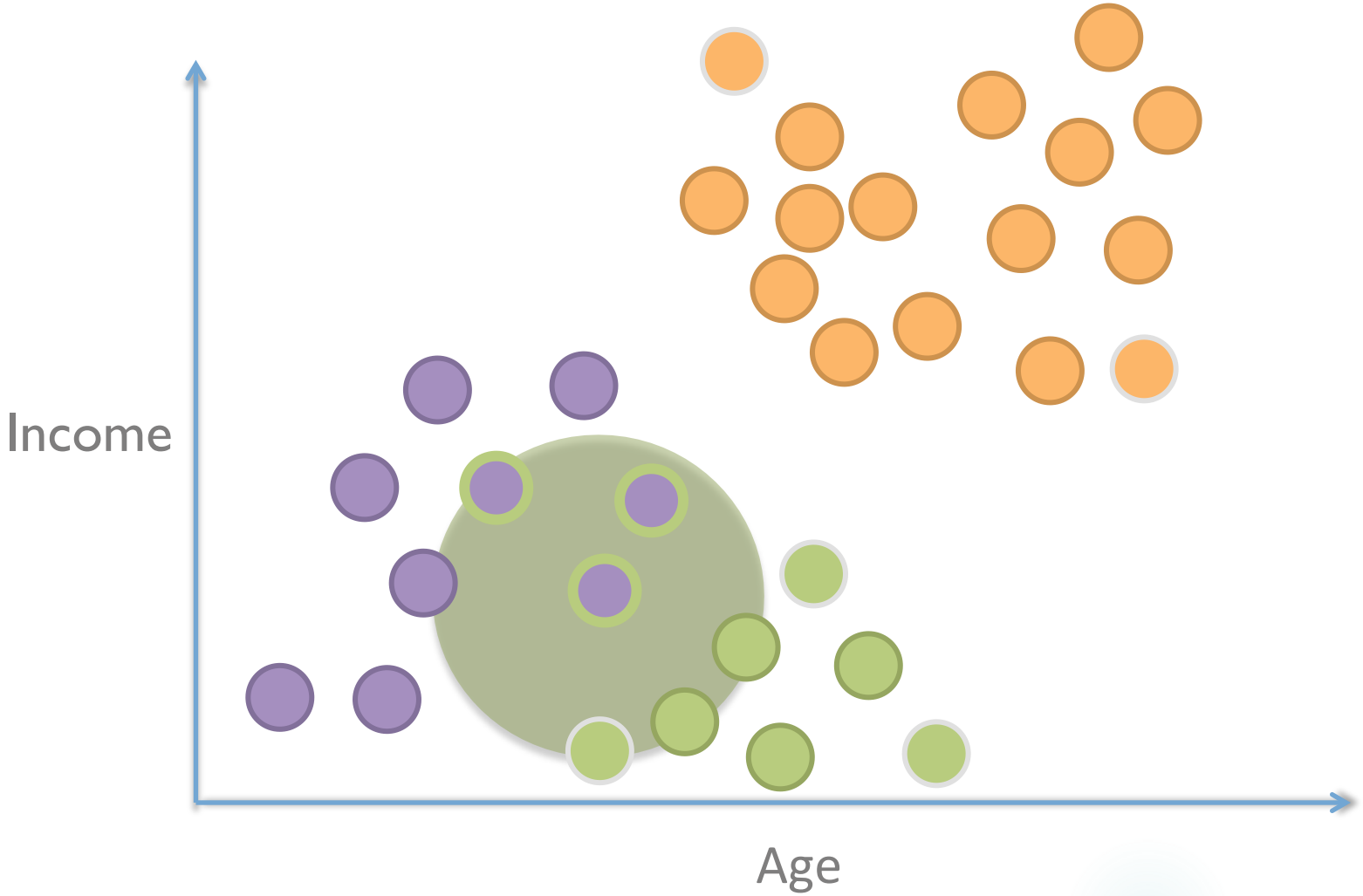
Keep adding neighbors within *epsilon* as chain reaction



epsilon = 1.75
n_clu = 3

DBSCAN

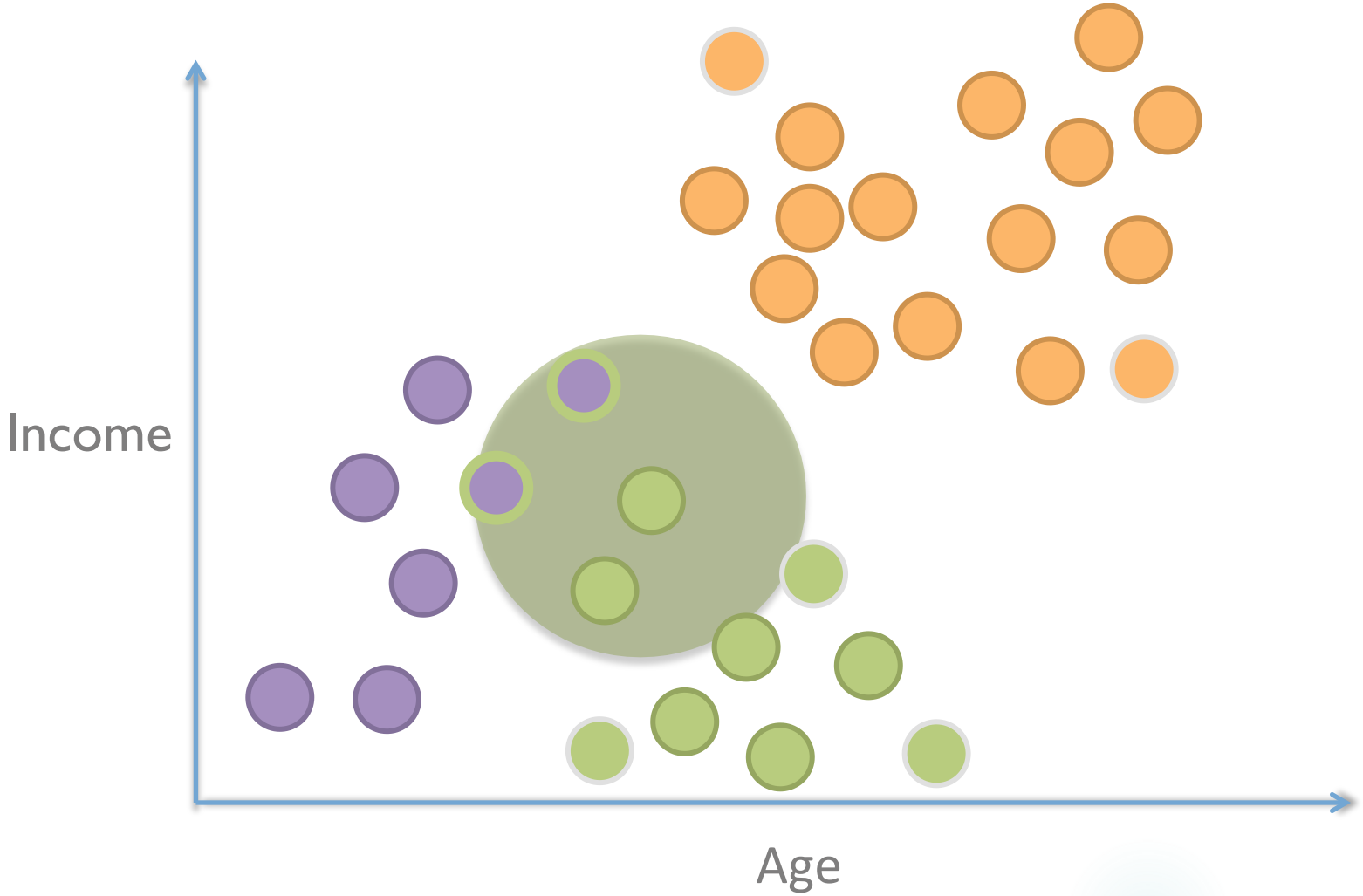
Keep adding neighbors within *epsilon* as chain reaction



epsilon = 1.75
n_clu = 3

DBSCAN

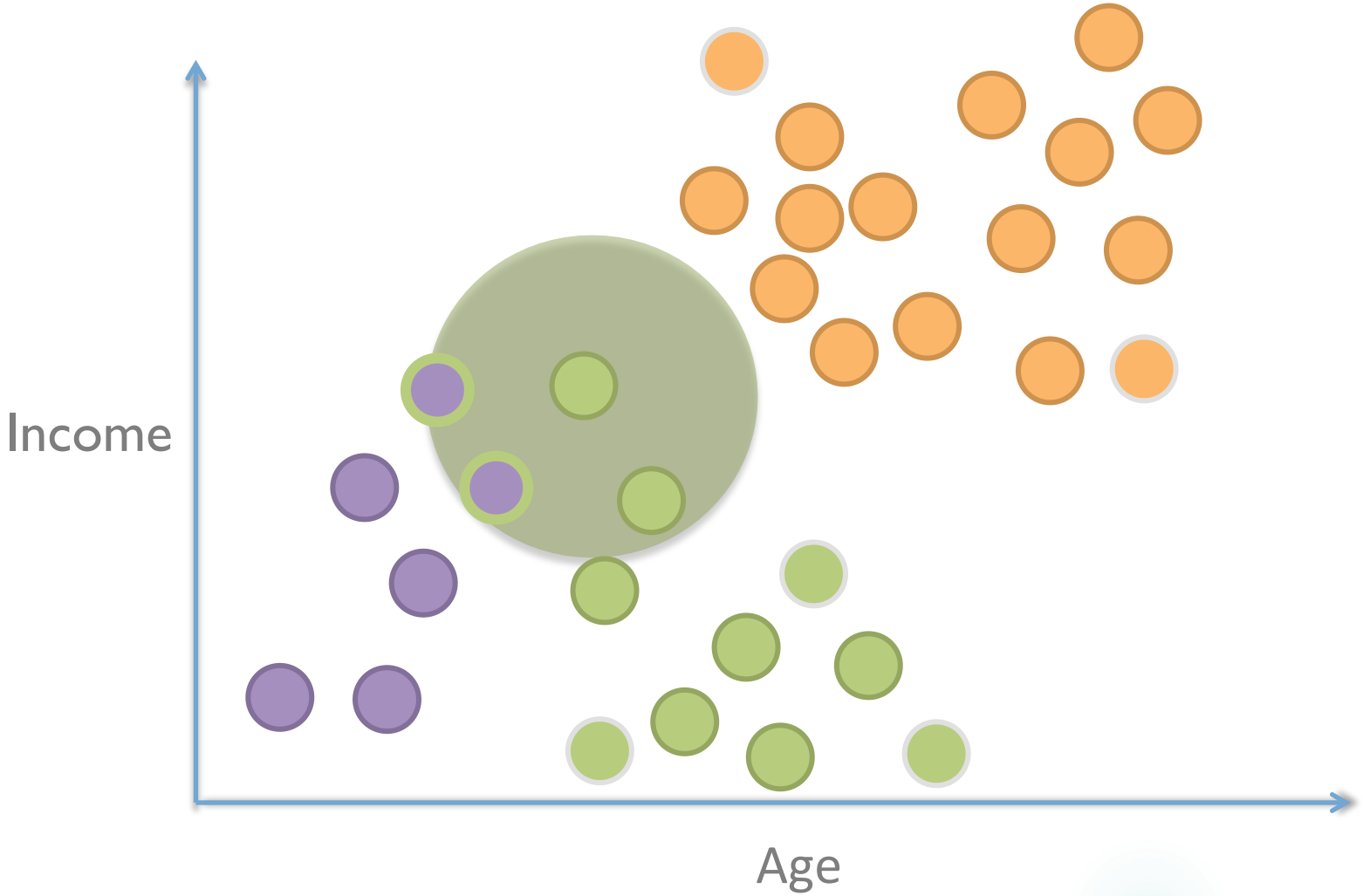
Keep adding neighbors within *epsilon* as chain reaction



epsilon = 1.75
n_clu = 3

DBSCAN

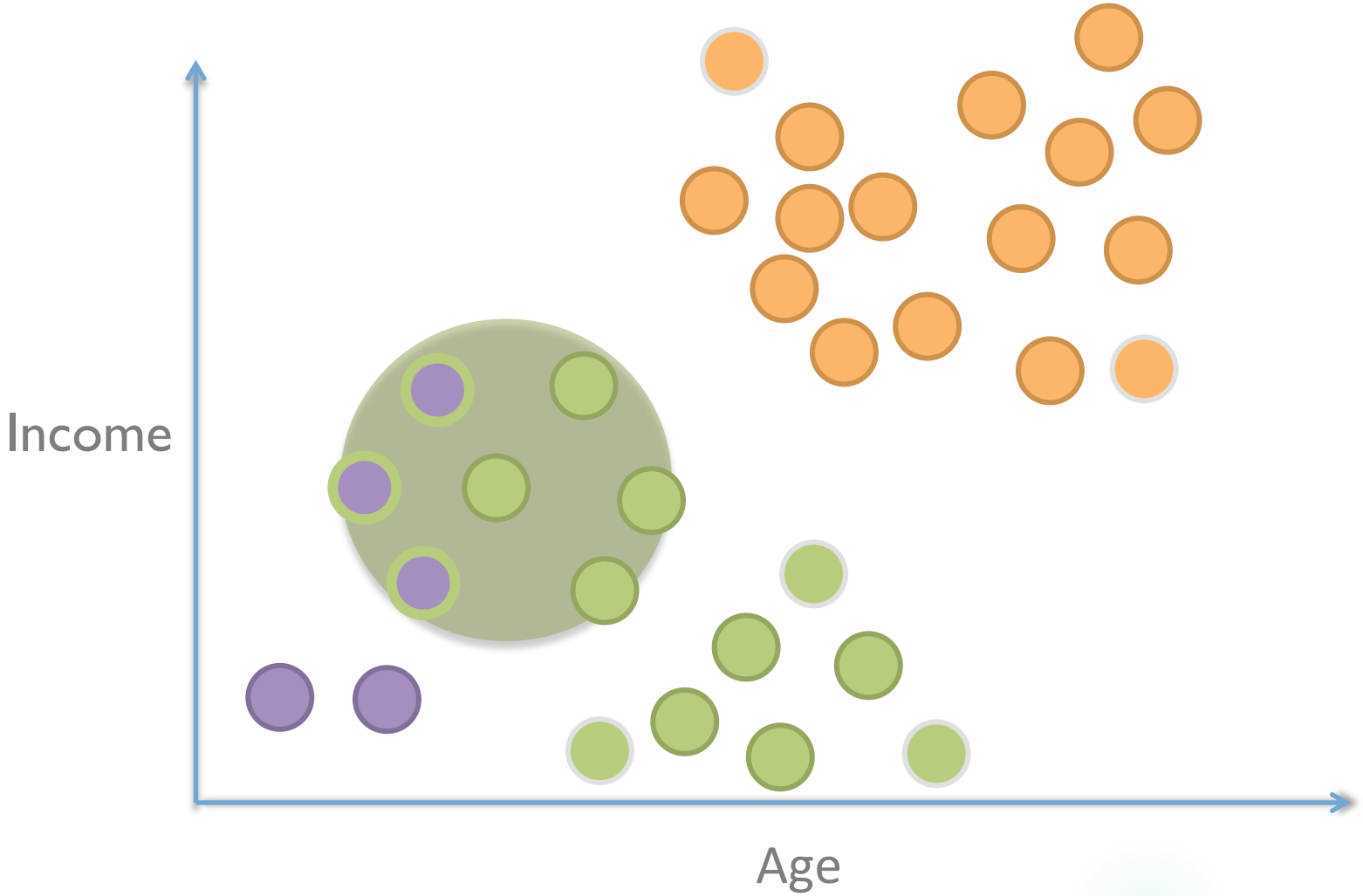
Keep adding neighbors within *epsilon* as chain reaction



epsilon = 1.75
n_clu = 3

DBSCAN

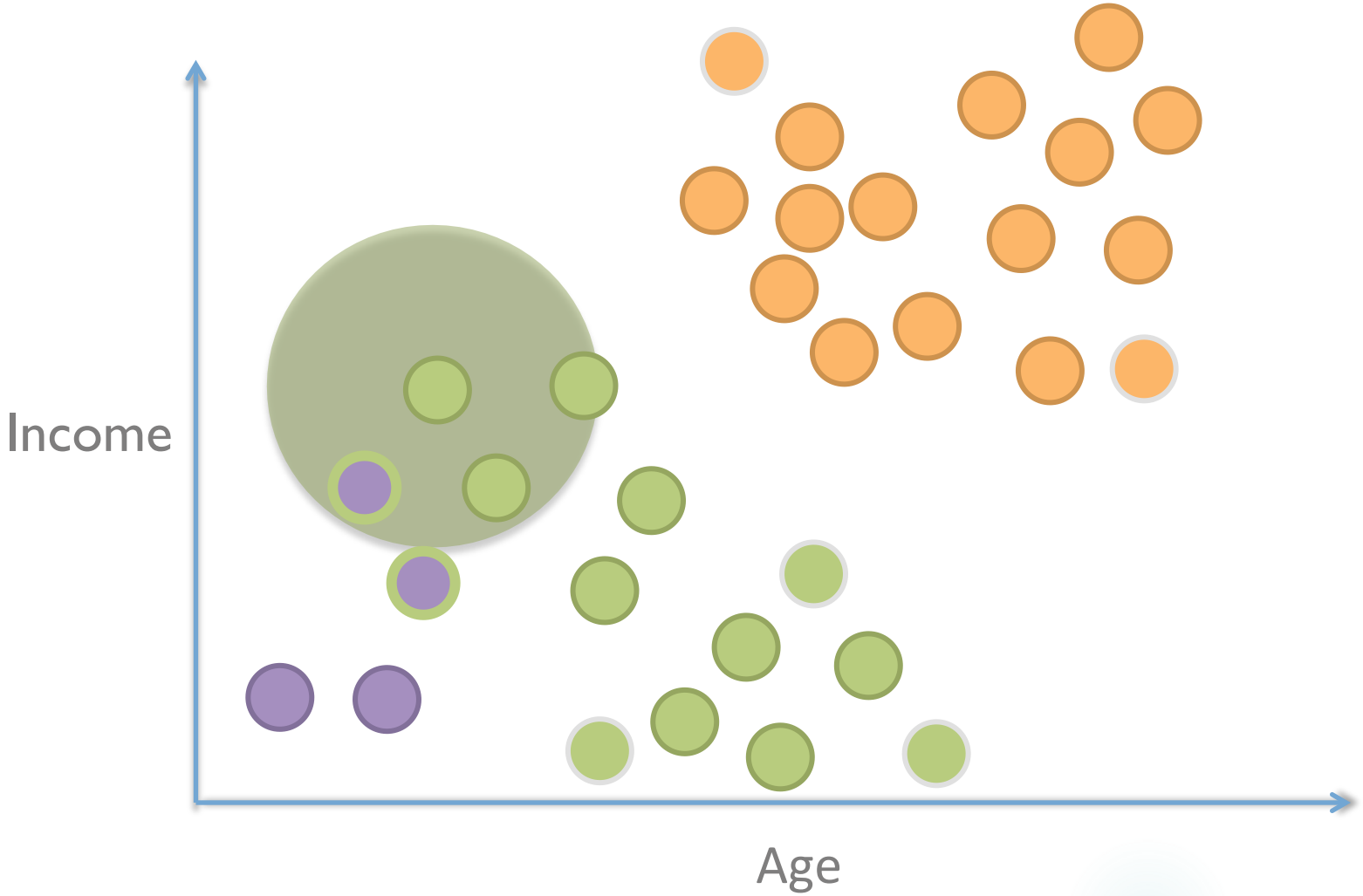
Keep adding neighbors within *epsilon* as chain reaction



epsilon = 1.75
n_clu = 3

DBSCAN

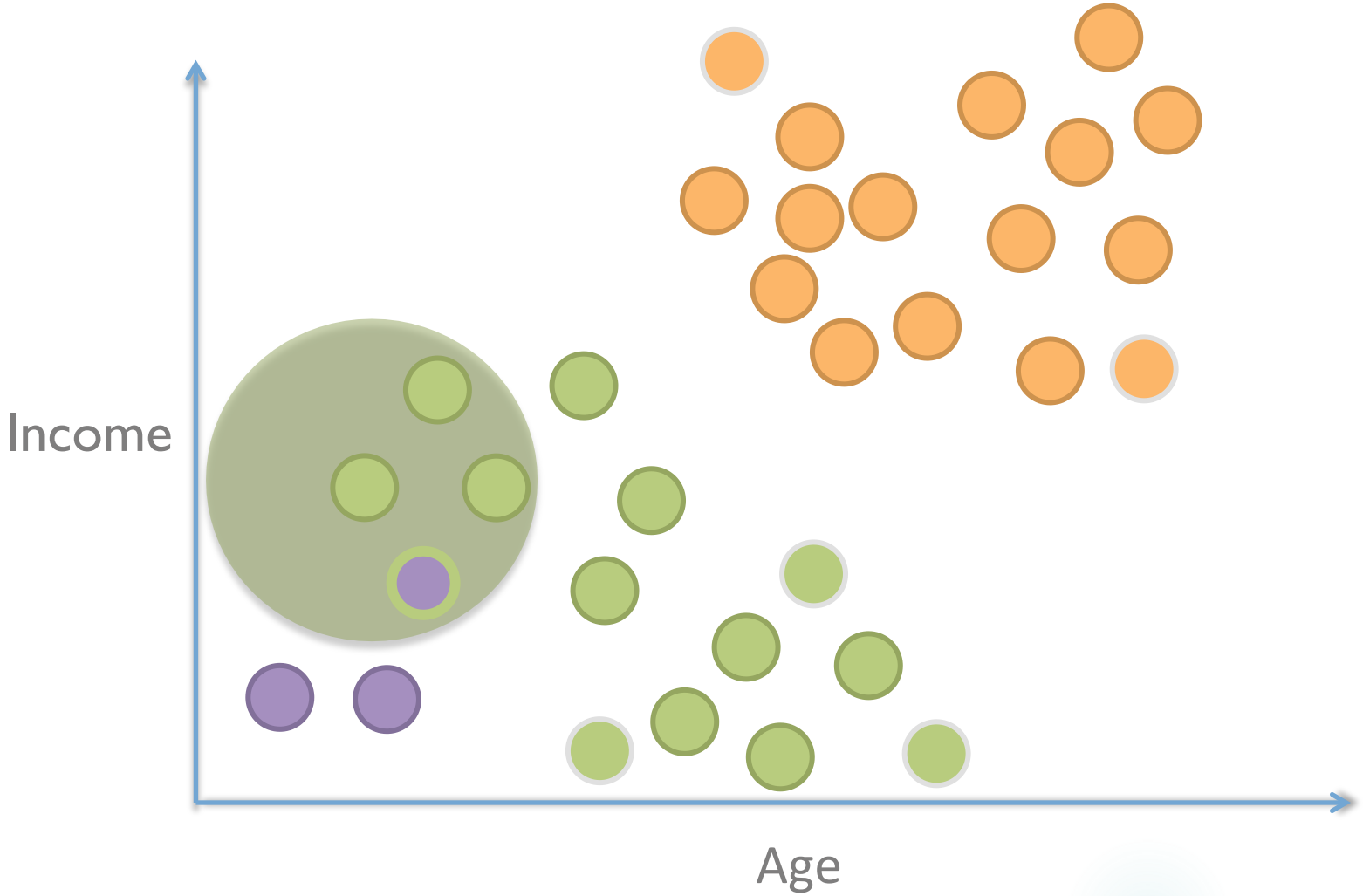
Keep adding neighbors within *epsilon* as chain reaction



epsilon = 1.75
n_clu = 3

DBSCAN

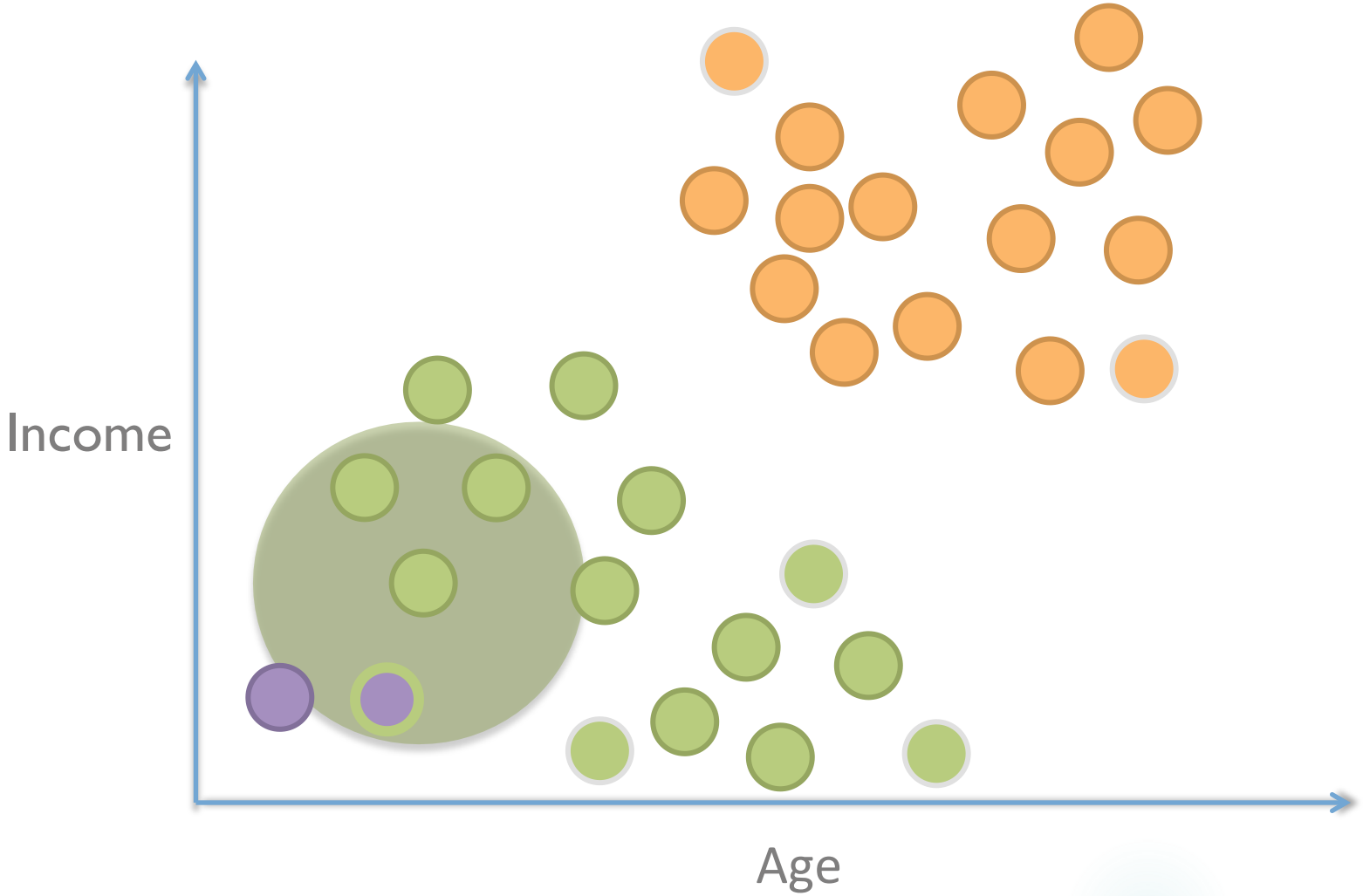
Keep adding neighbors within *epsilon* as chain reaction



epsilon = 1.75
n_clu = 3

DBSCAN

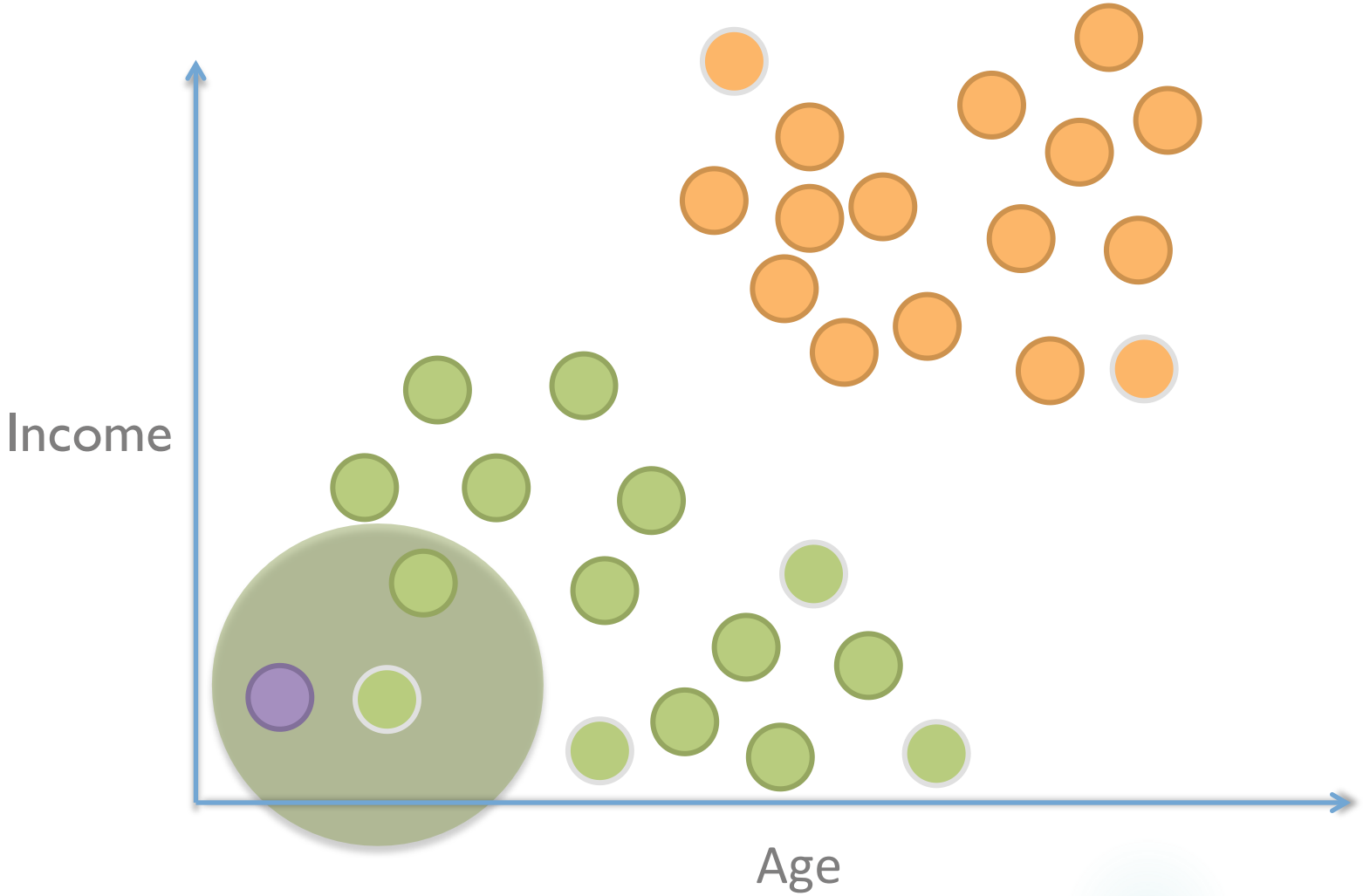
Keep adding neighbors within *epsilon* as chain reaction



epsilon = 1.75
n_clu = 3

DBSCAN

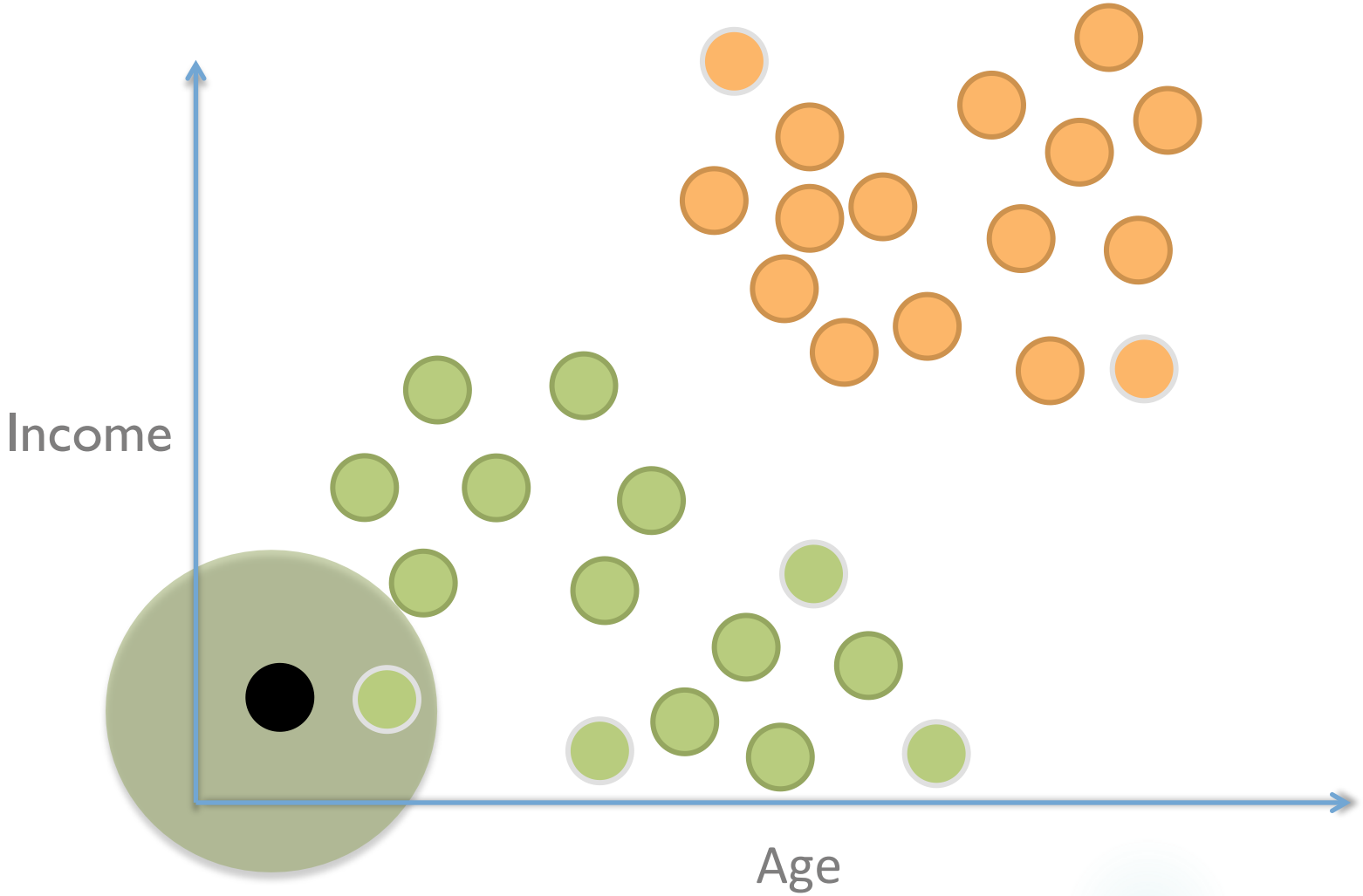
Keep adding neighbors within *epsilon* as chain reaction



epsilon = 1.75
n_clu = 3

DBSCAN

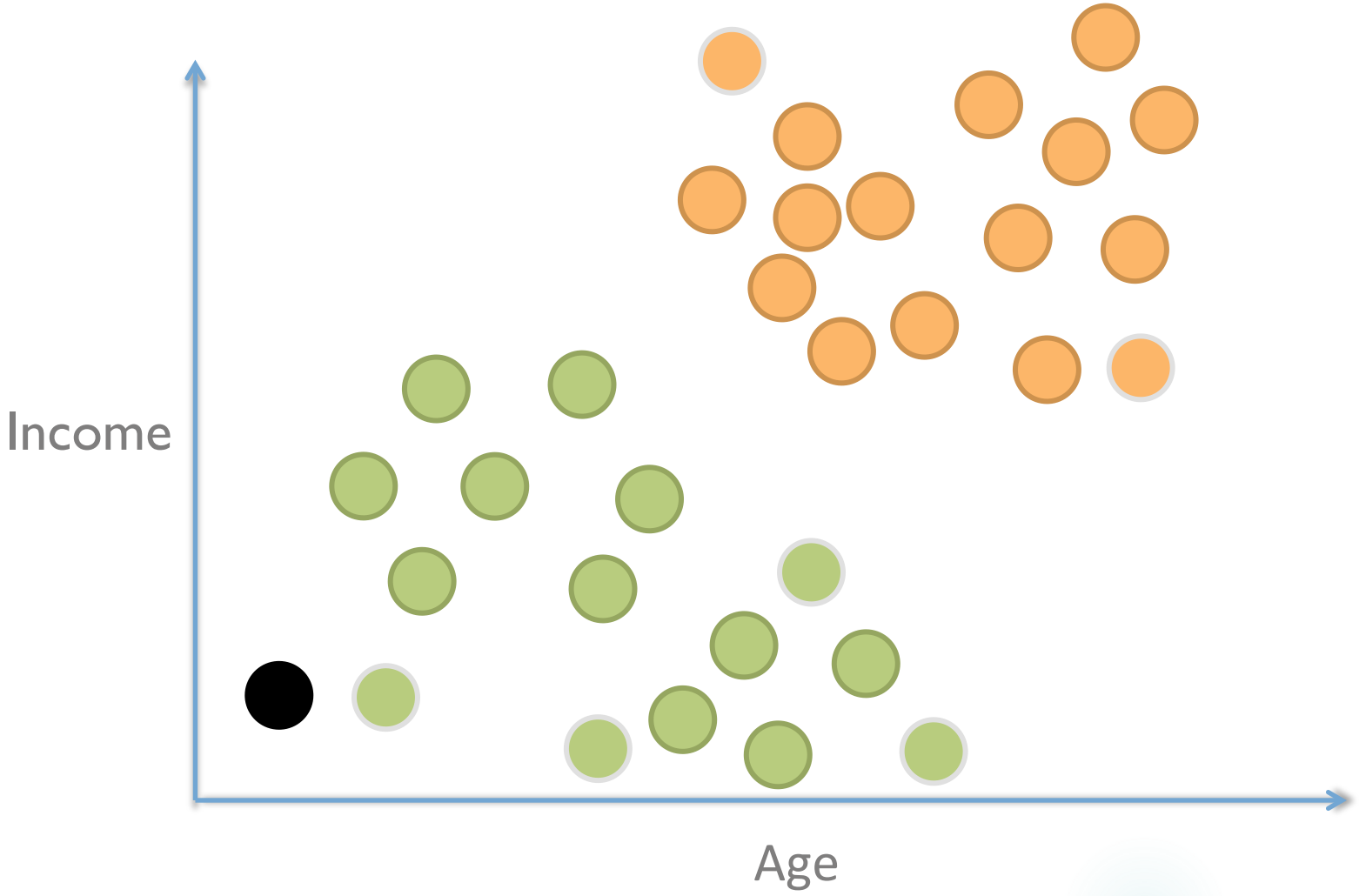
If there is a point that does not have *n_clu* neighbors and is not reached from a core point: It is a **noise** point.



epsilon = 1.75
n_clu = 3

DBSCAN

Ta--daa! Notice the core points, density-reachable points (at the borders) and the **noise** point.



DBSCAN: strengths and weaknesses

Strengths:

1. No need to specify number of clusters (cf. *k*-means)
2. Allows for noise
3. Can handle arbitrary-shaped clusters

Weaknesses:

1. Requires two parameters (vs. one for *k*-means)
2. Finding appropriate values of ε and n_clu can be tricky
3. Does not do well with clusters of different density

Mean Shift

Mean shift

A partitioning algorithm that assigns points to nearest cluster centroid

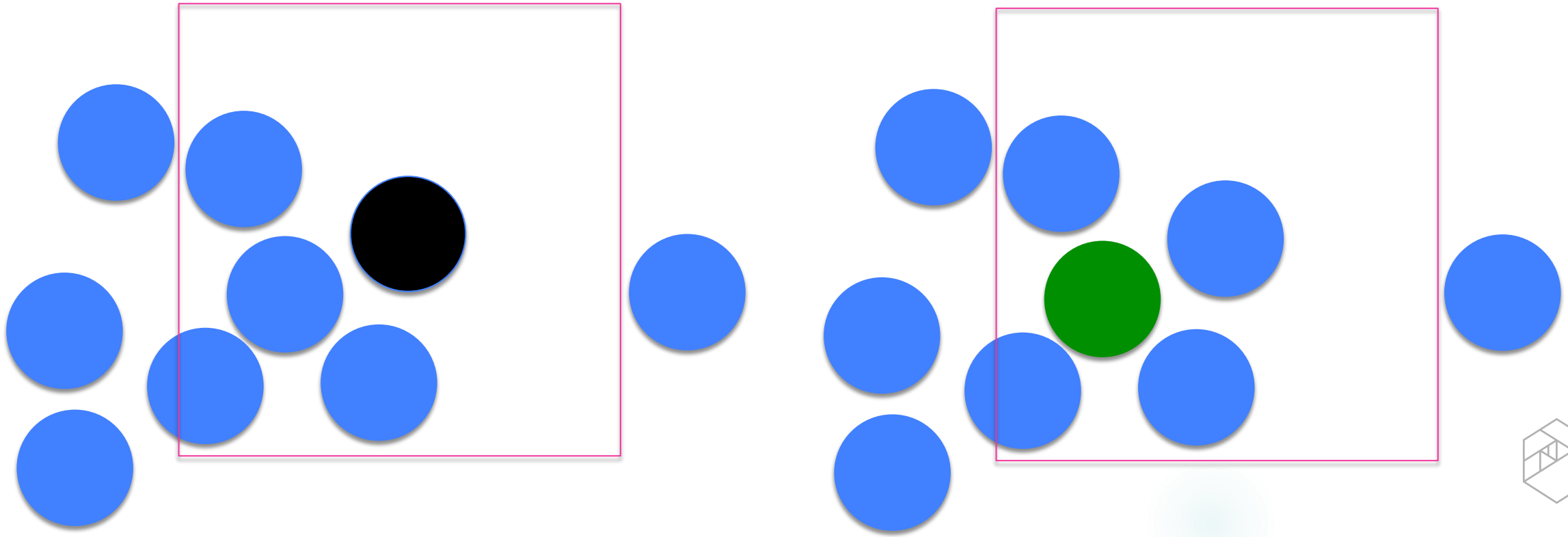
Centroid: point of highest local density

Algorithm ends when all points assigned to a cluster

Mean shift: local density

How do you calculate local density?

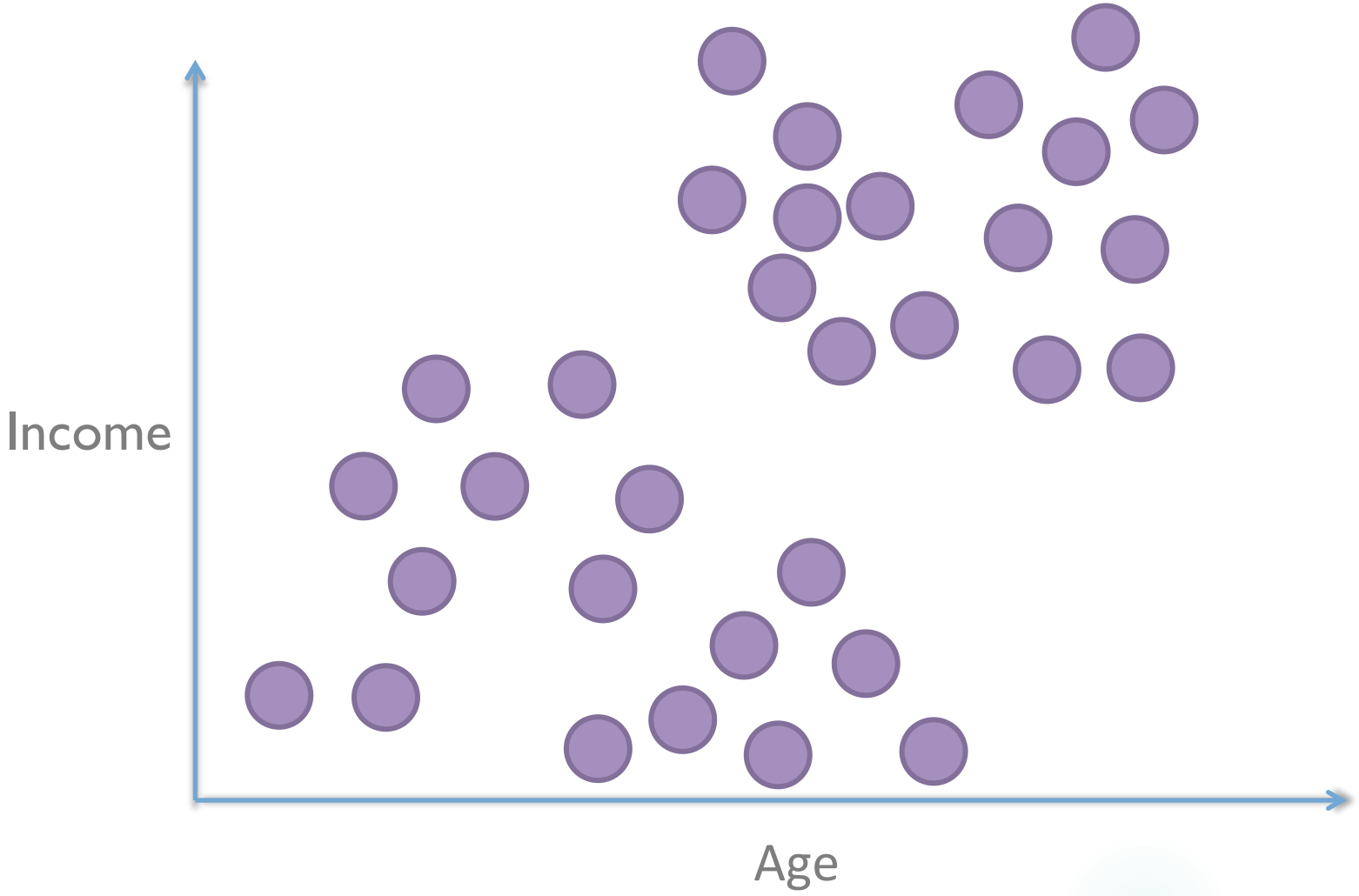
Evaluate weighted mean around each point



Mean shift: the algorithm

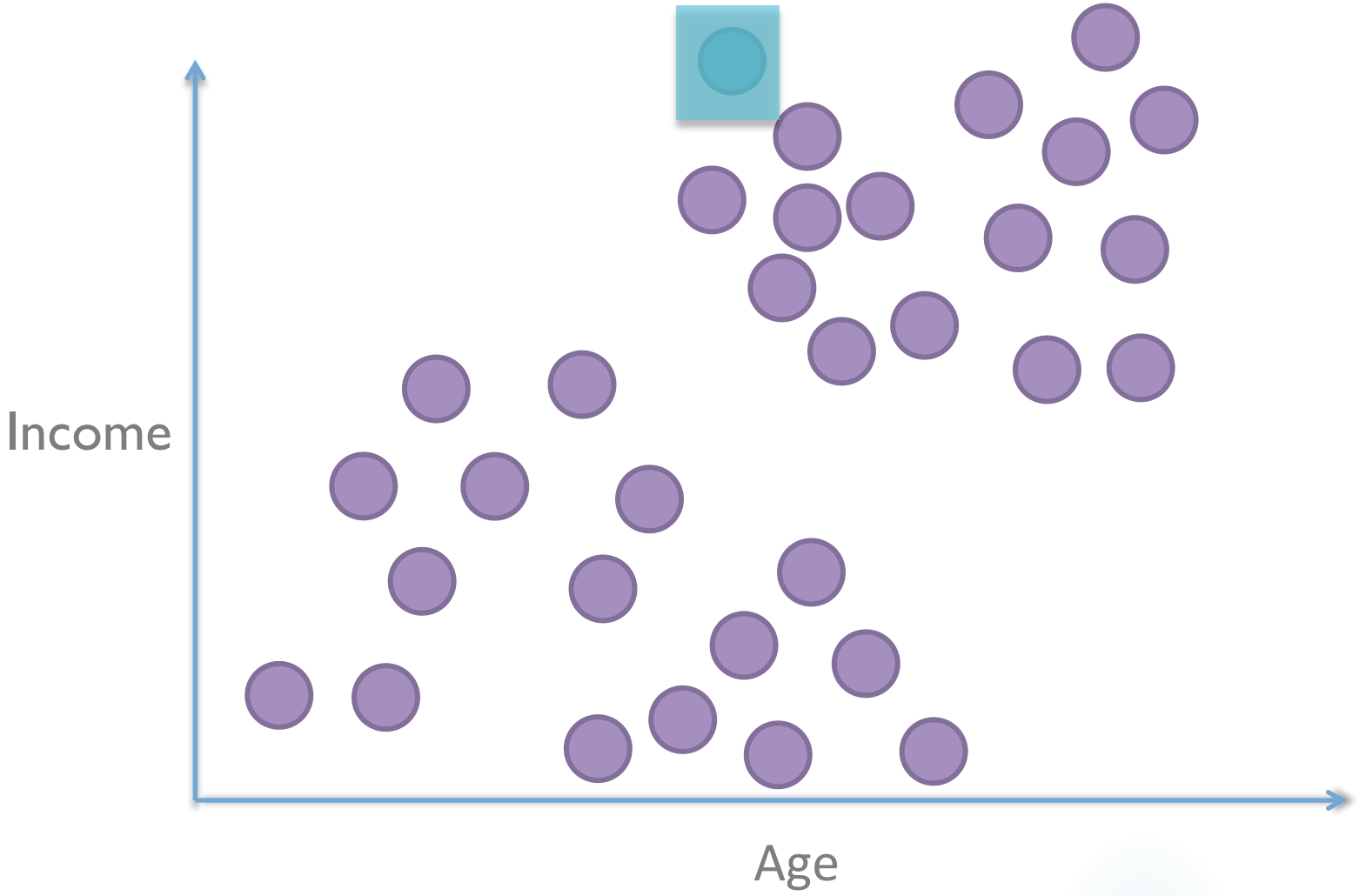
1. Choose a point and window *W*
2. Calculate weighted mean in *W*
3. Shift center of window to new mean
4. Repeat steps (2) and (3) until convergence (no shift)
i.e. until local density maximum ("mode") is reached
5. Repeat steps (1-4) for all data points
6. Data points that lead to same mode are grouped into same cluster

Mean Shift



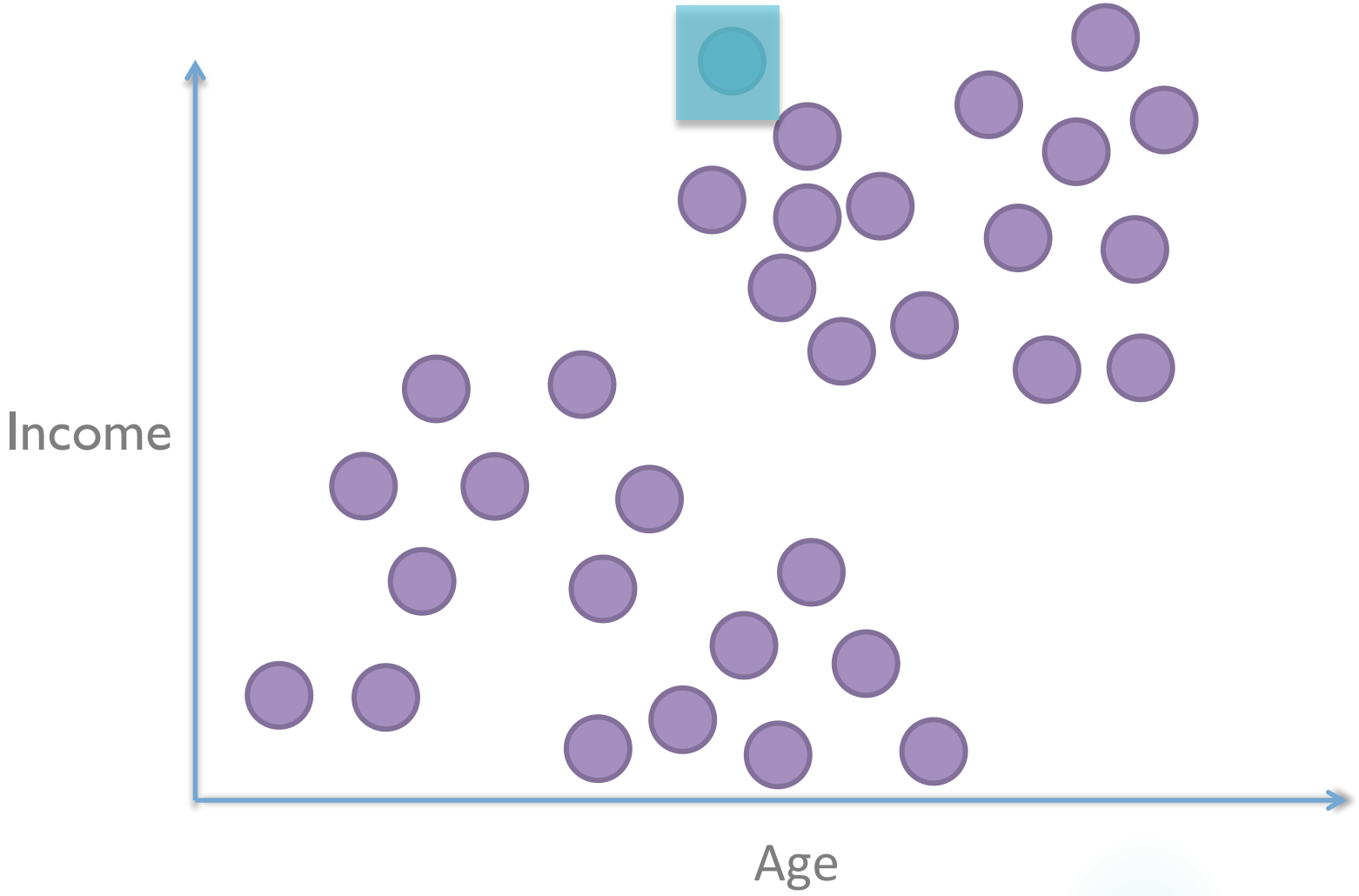
Mean Shift

Start with a centroid at a point



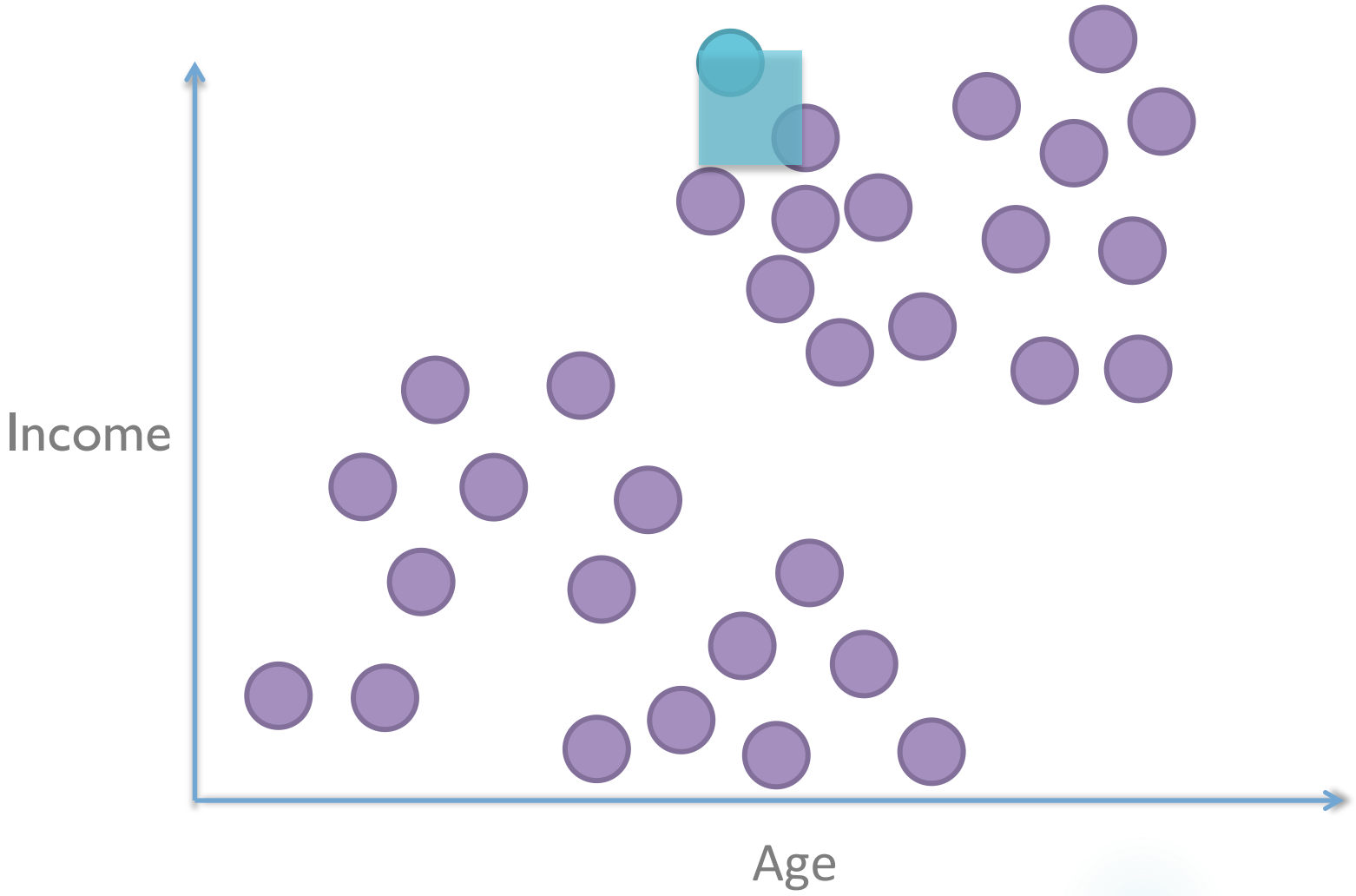
Mean Shift

Sample local density, follow gradient towards denser direction



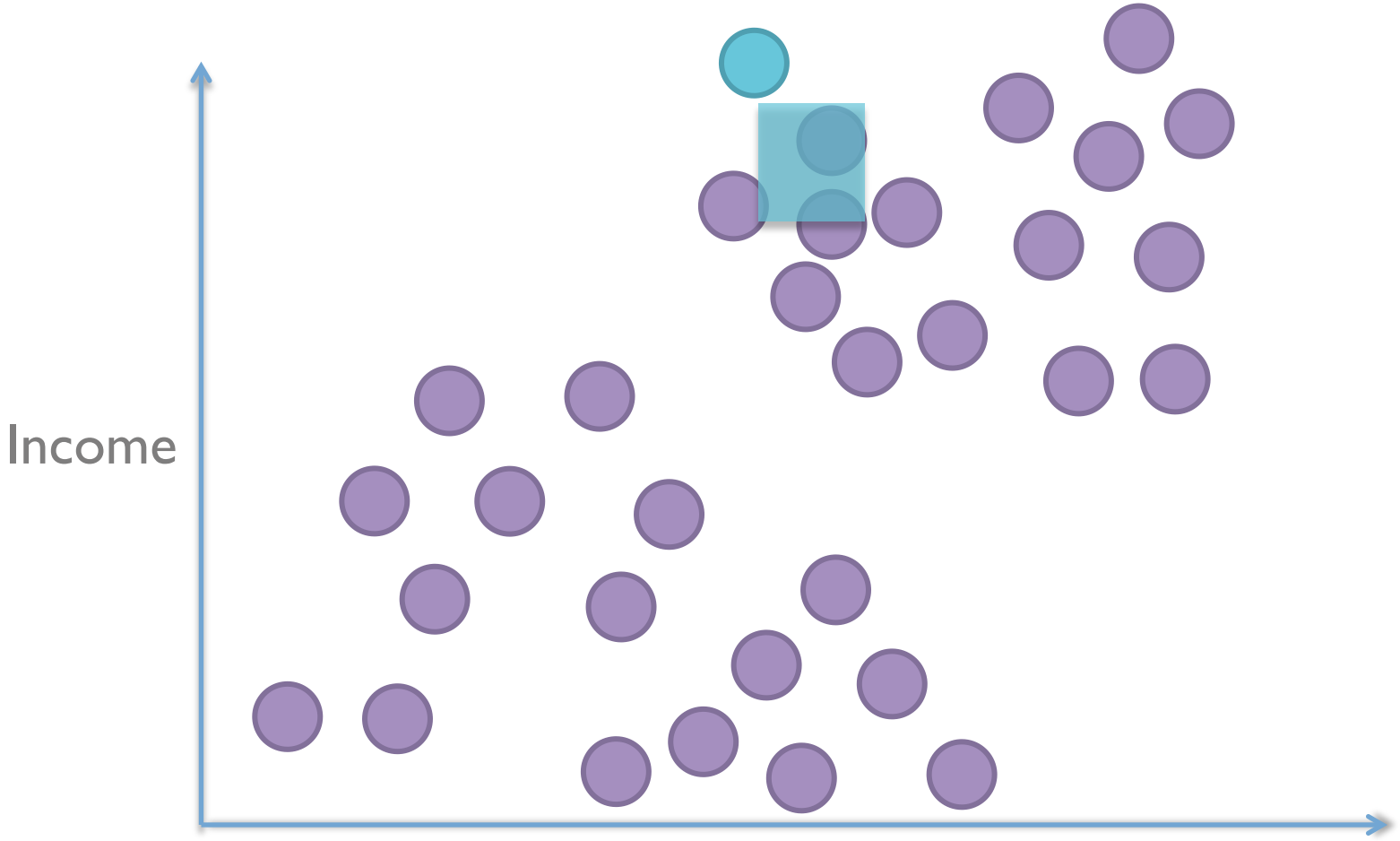
Mean Shift

Sample local density, follow gradient towards denser direction



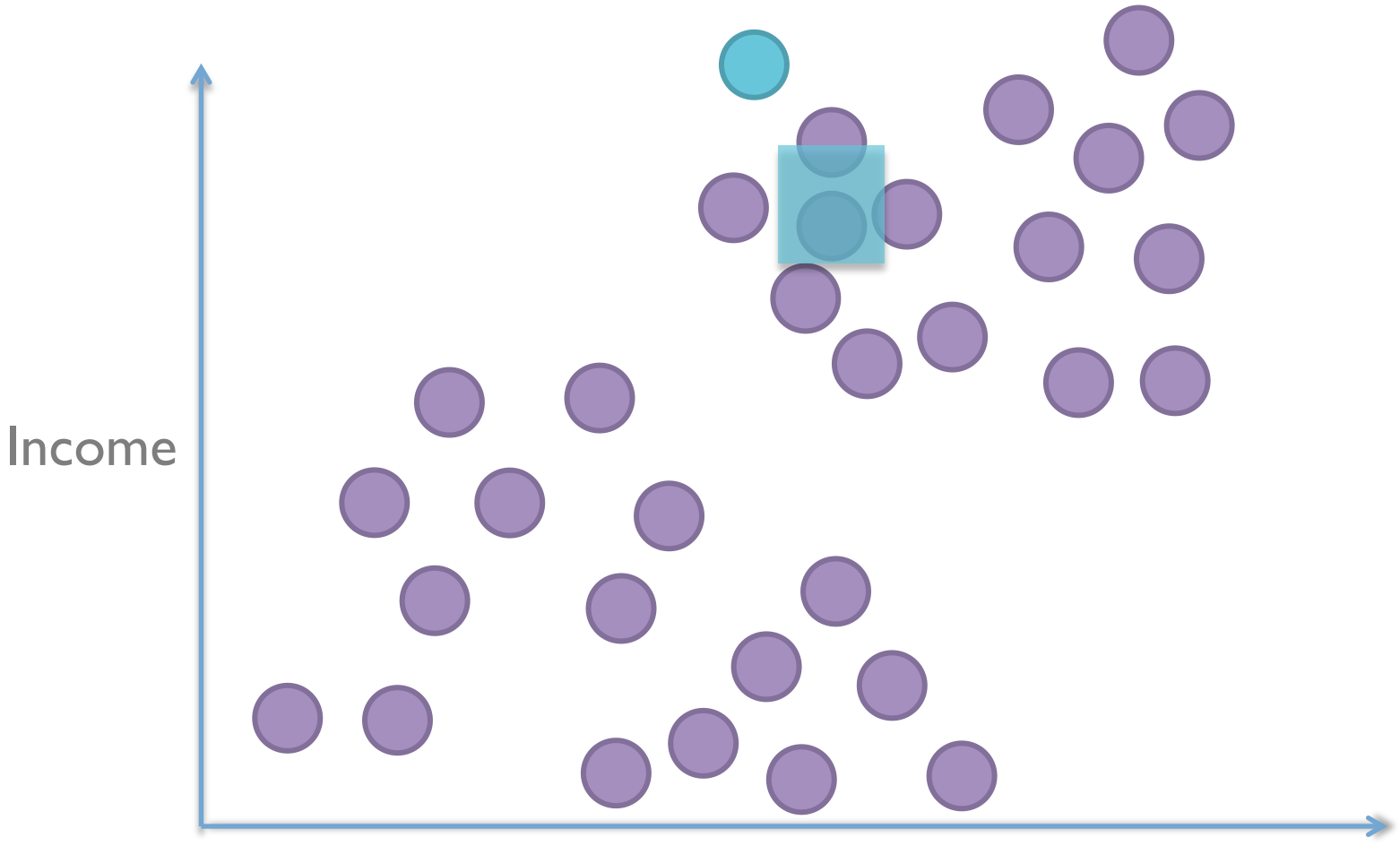
Mean Shift

Sample local density, follow gradient towards denser direction



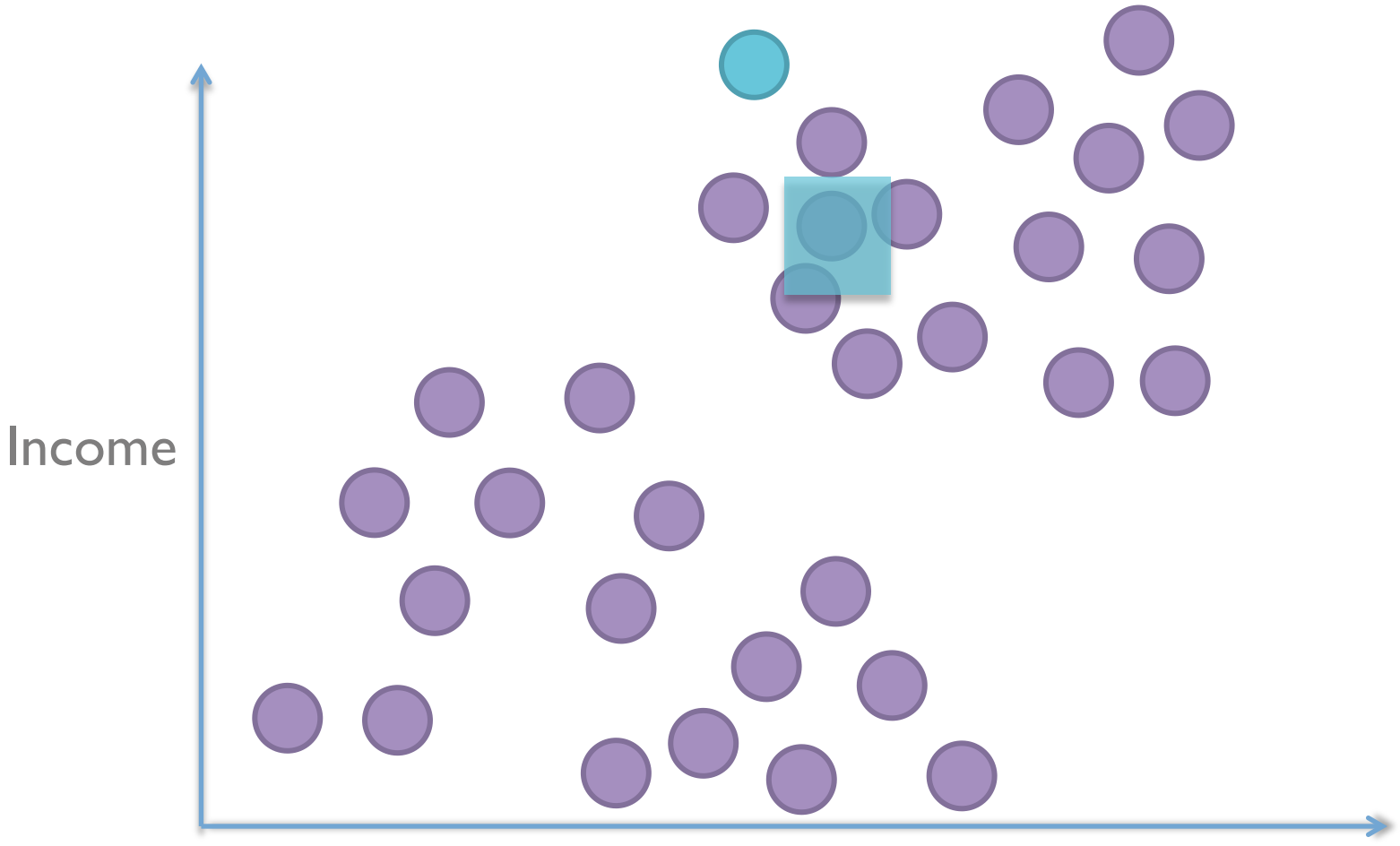
Mean Shift

Sample local density, follow gradient towards denser direction



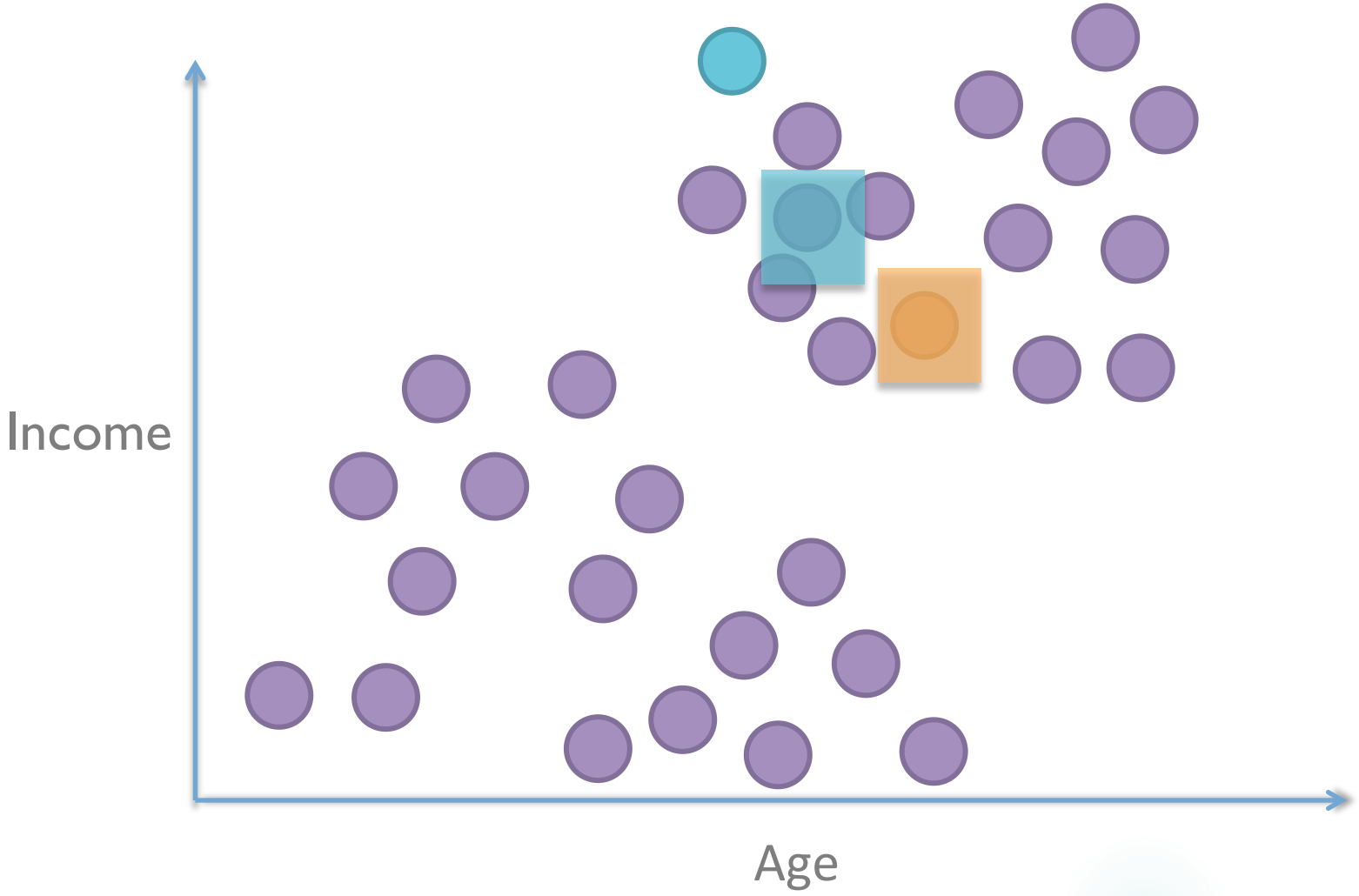
Mean Shift

Found local density maximum! Stop.



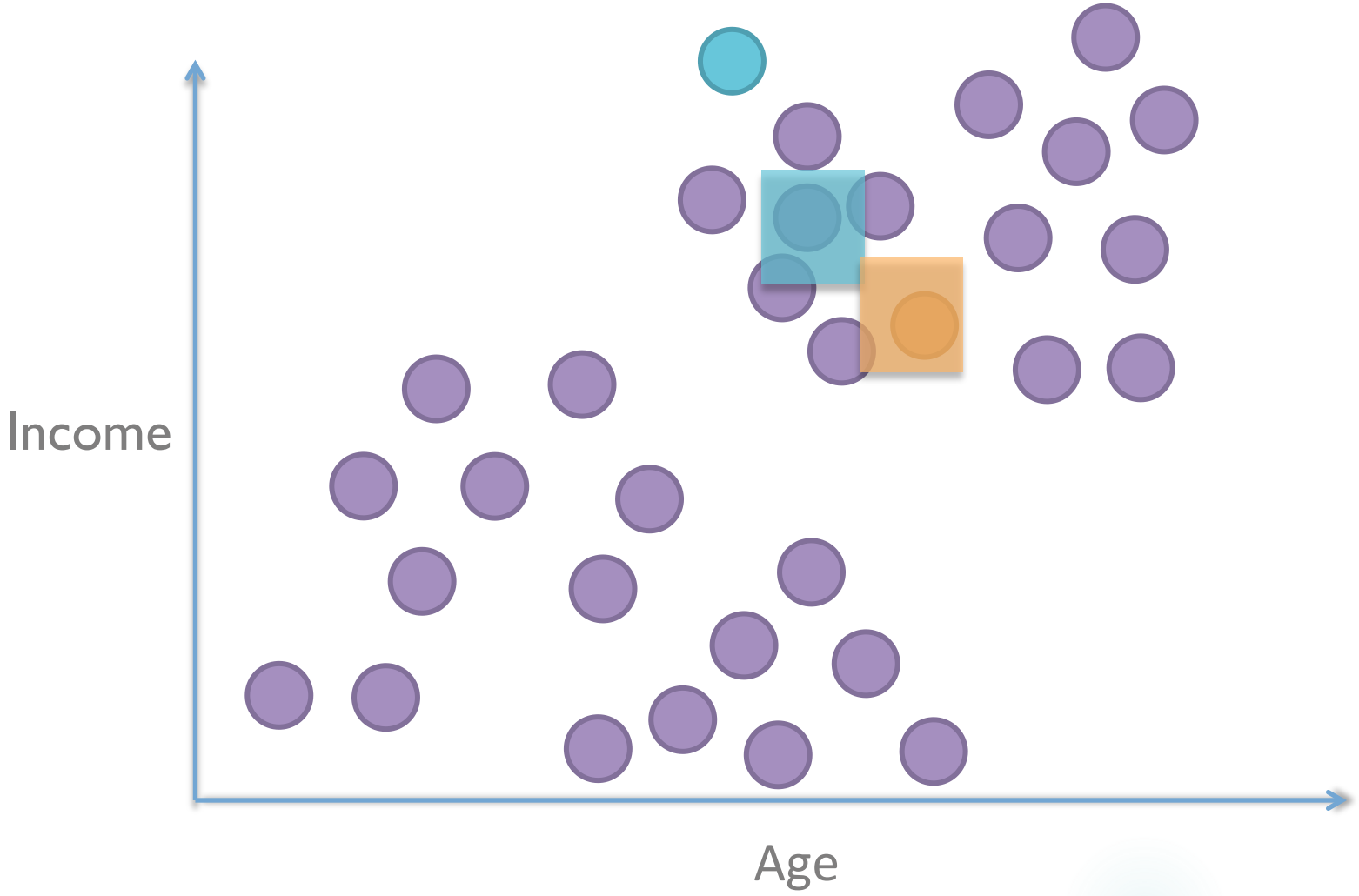
Mean Shift

Start at another point.



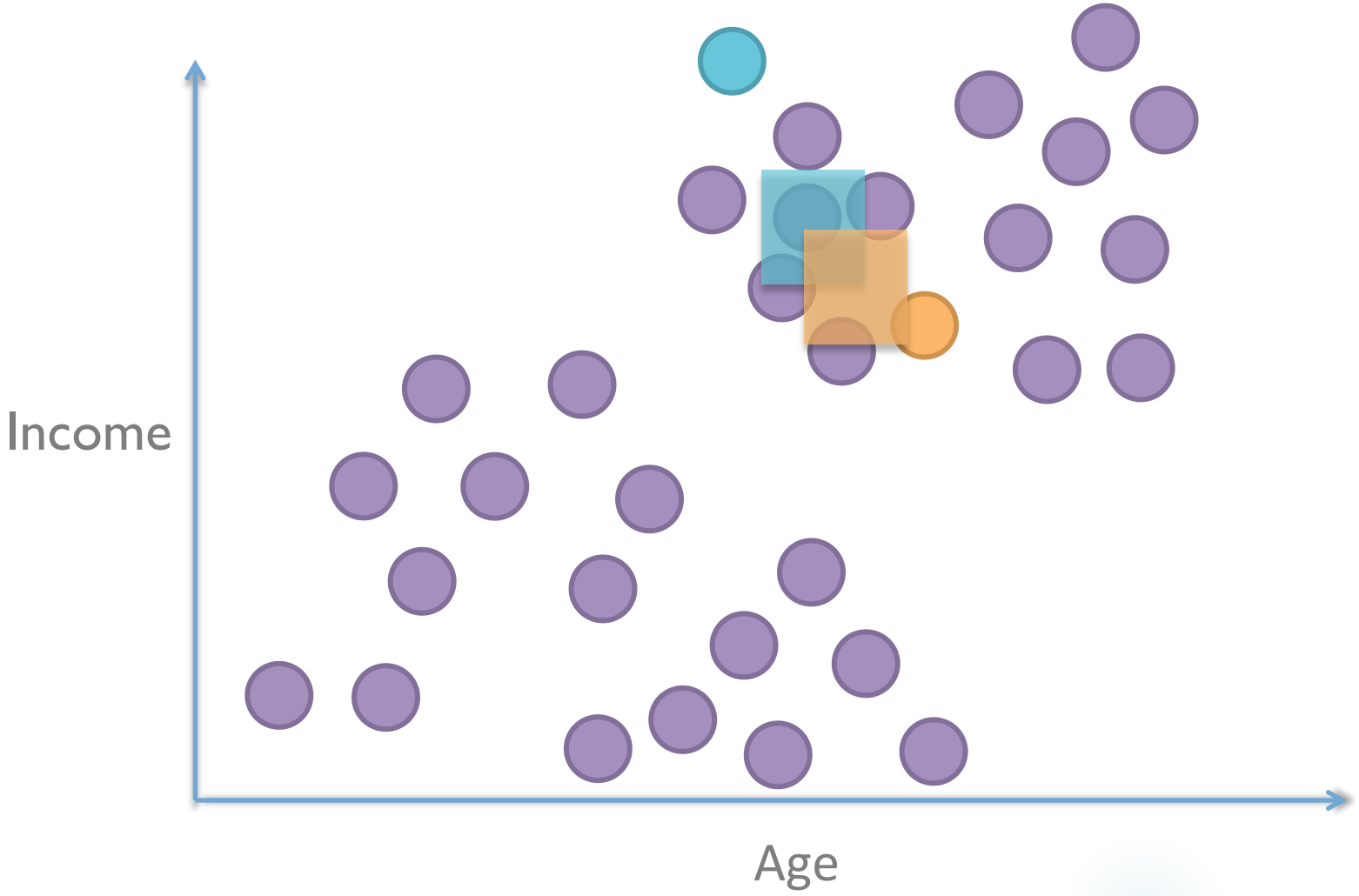
Mean Shift

Sample local density, follow gradient towards denser direction



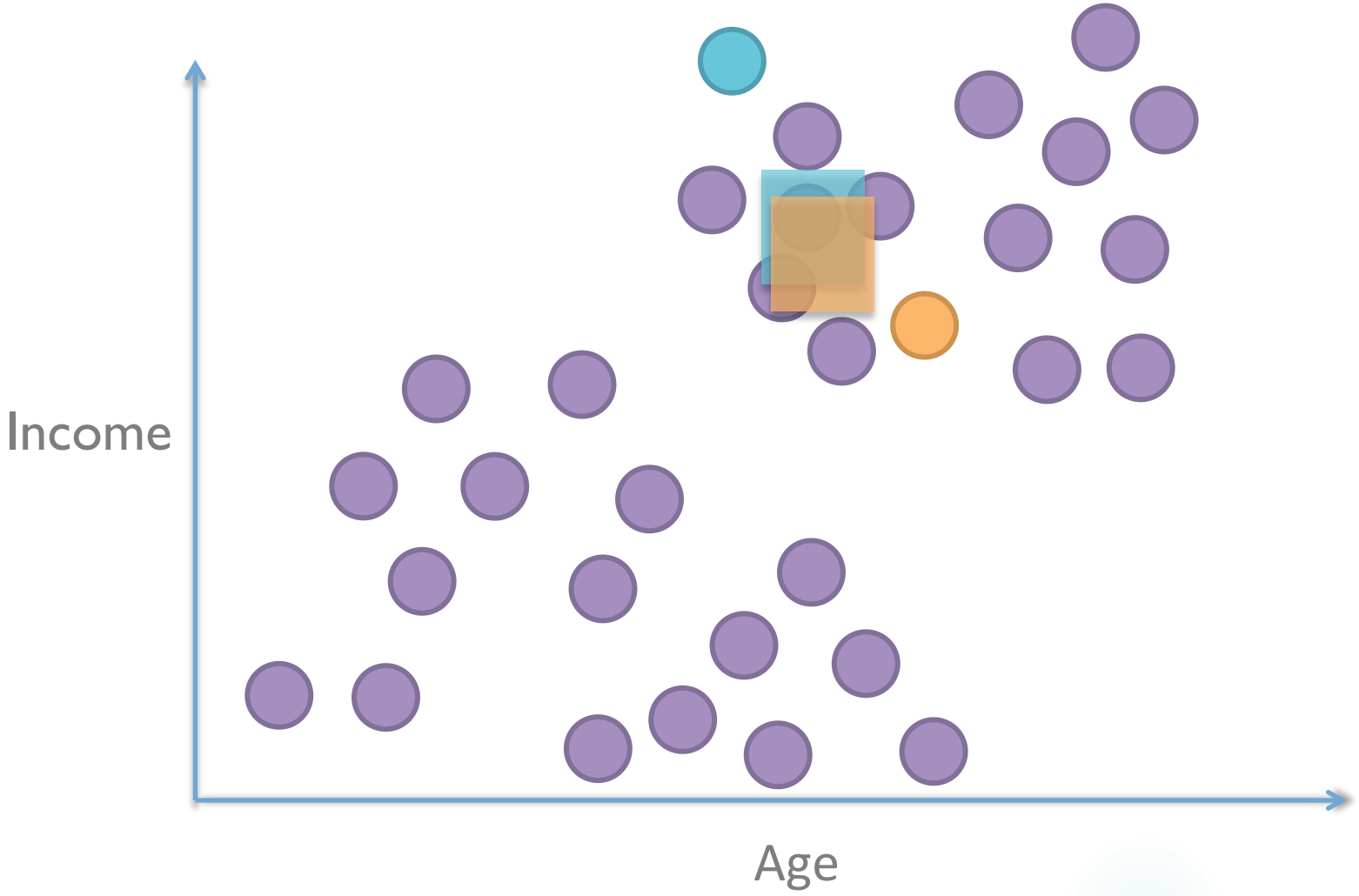
Mean Shift

Sample local density, follow gradient towards denser direction



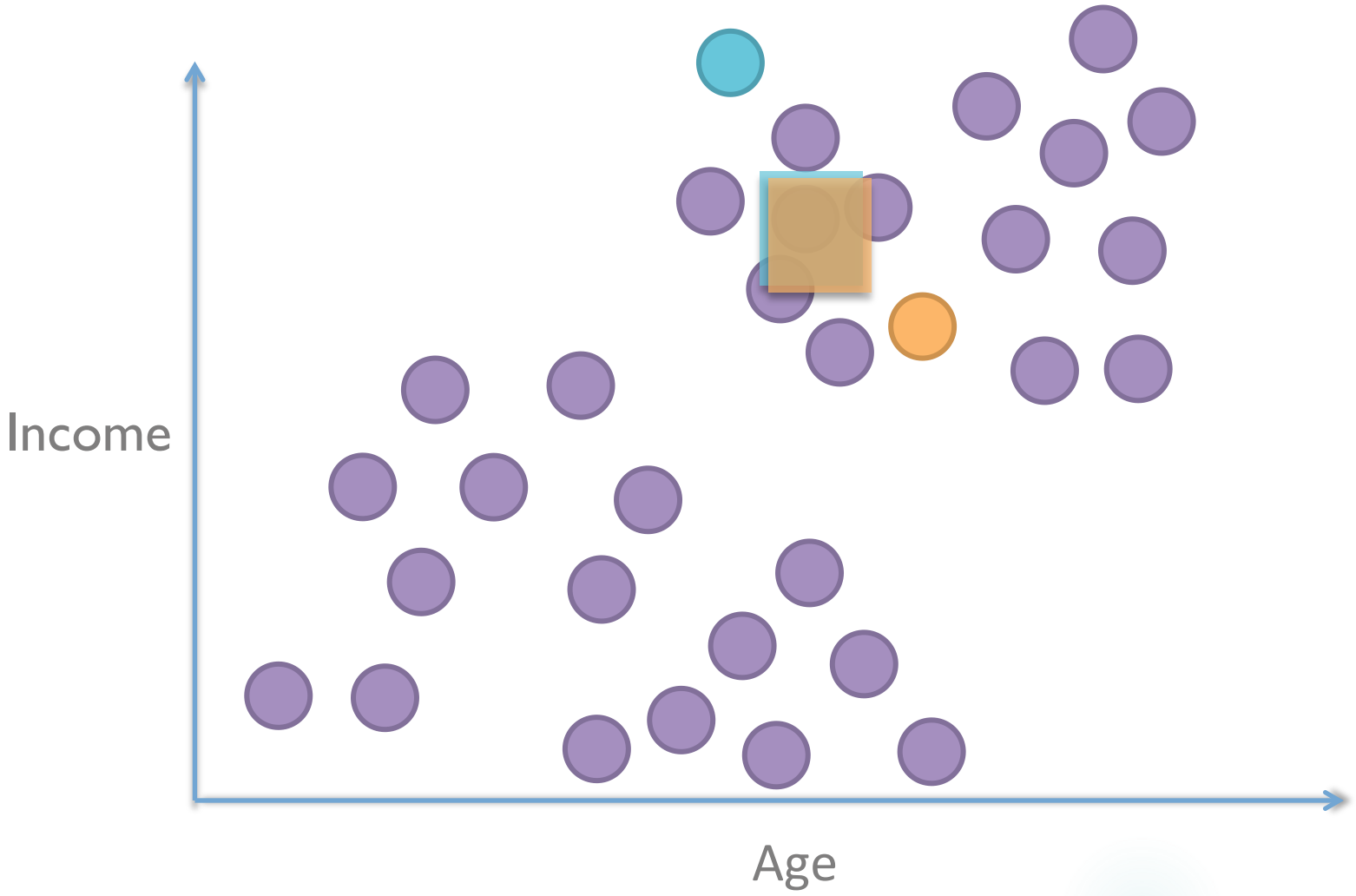
Mean Shift

Sample local density, follow gradient towards denser direction



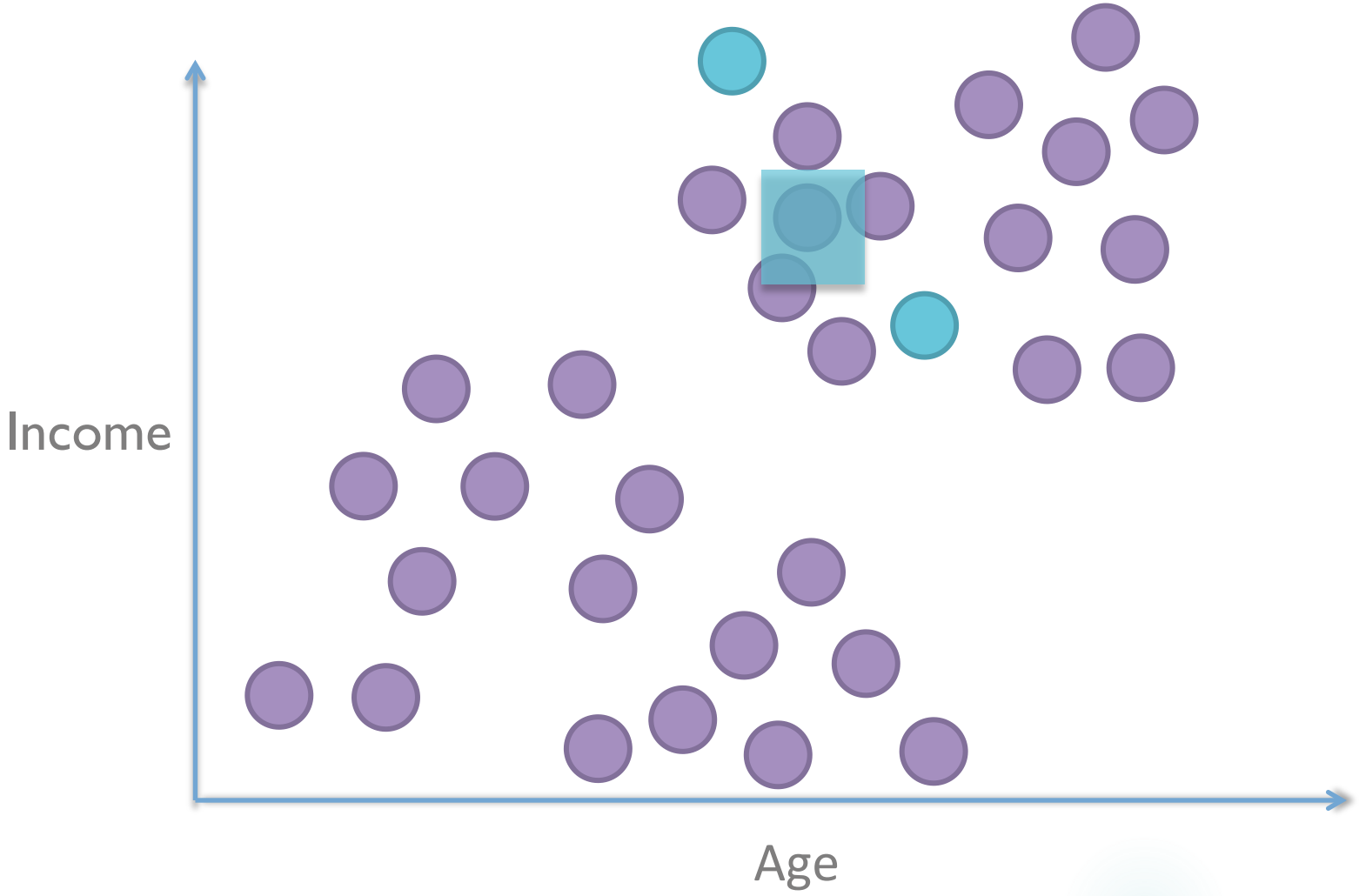
Mean Shift

Sample local density, follow gradient towards denser direction



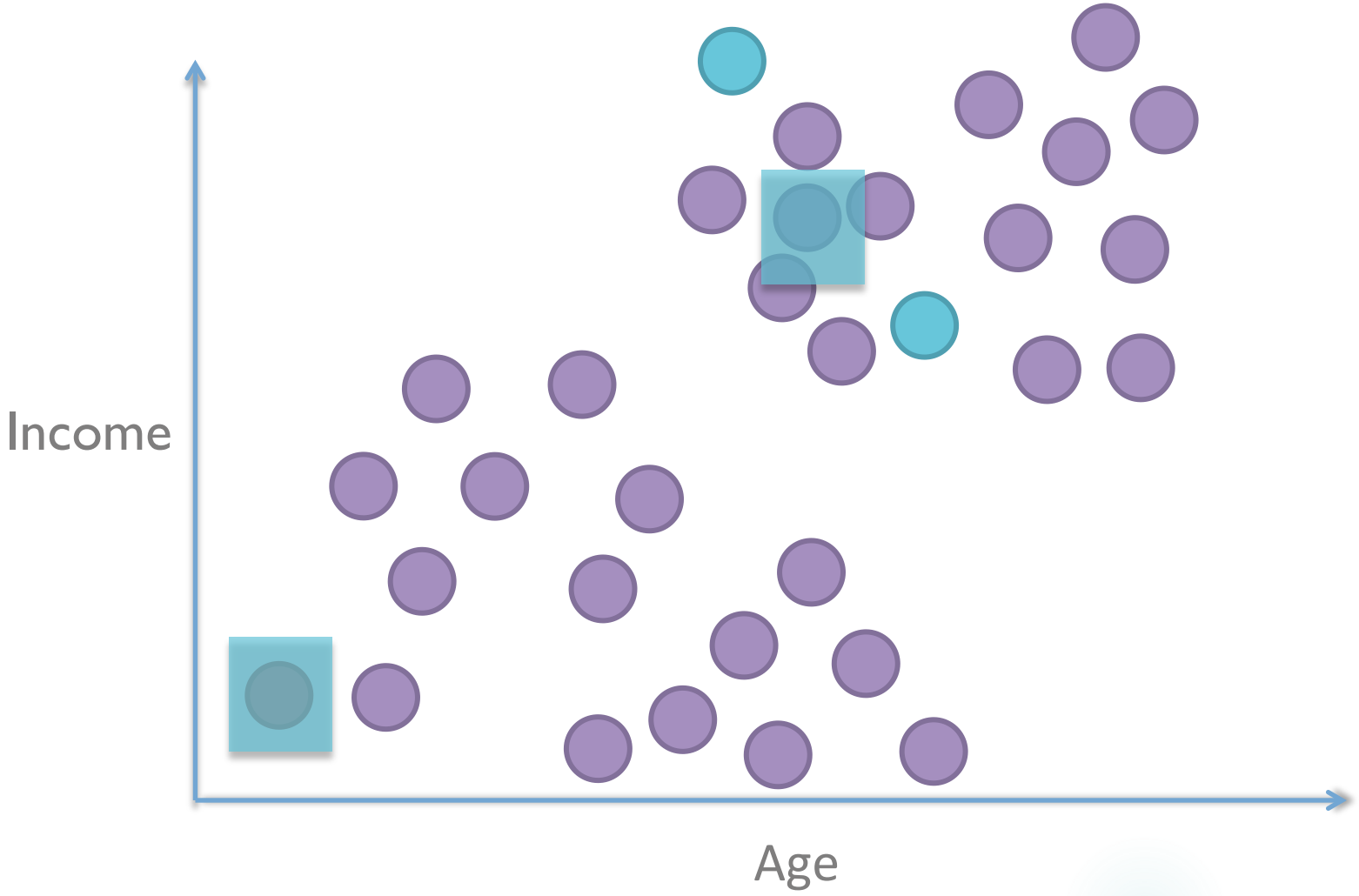
Mean Shift

Found the same (close) local maximum, same cluster



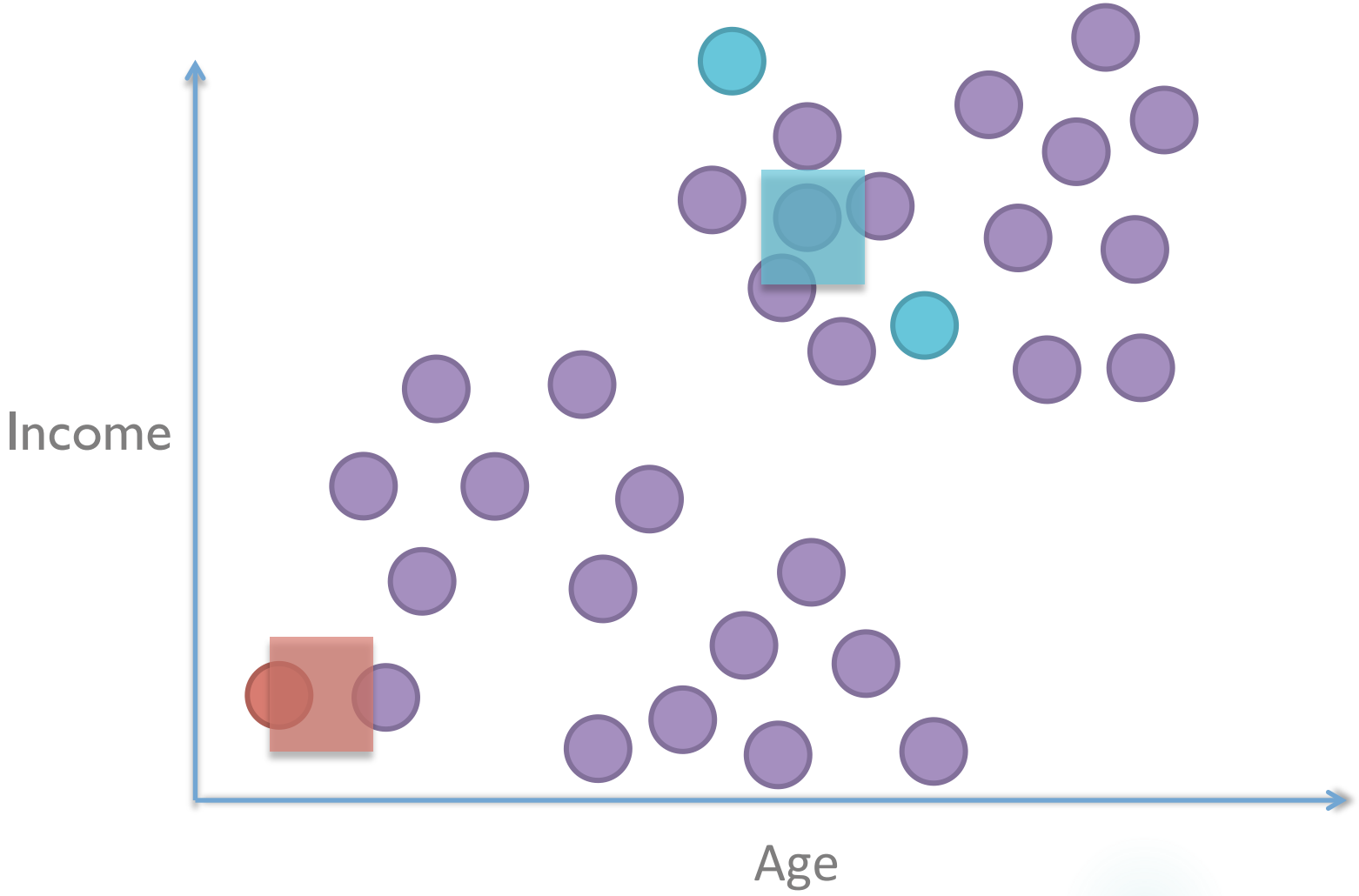
Mean Shift

Start at another point



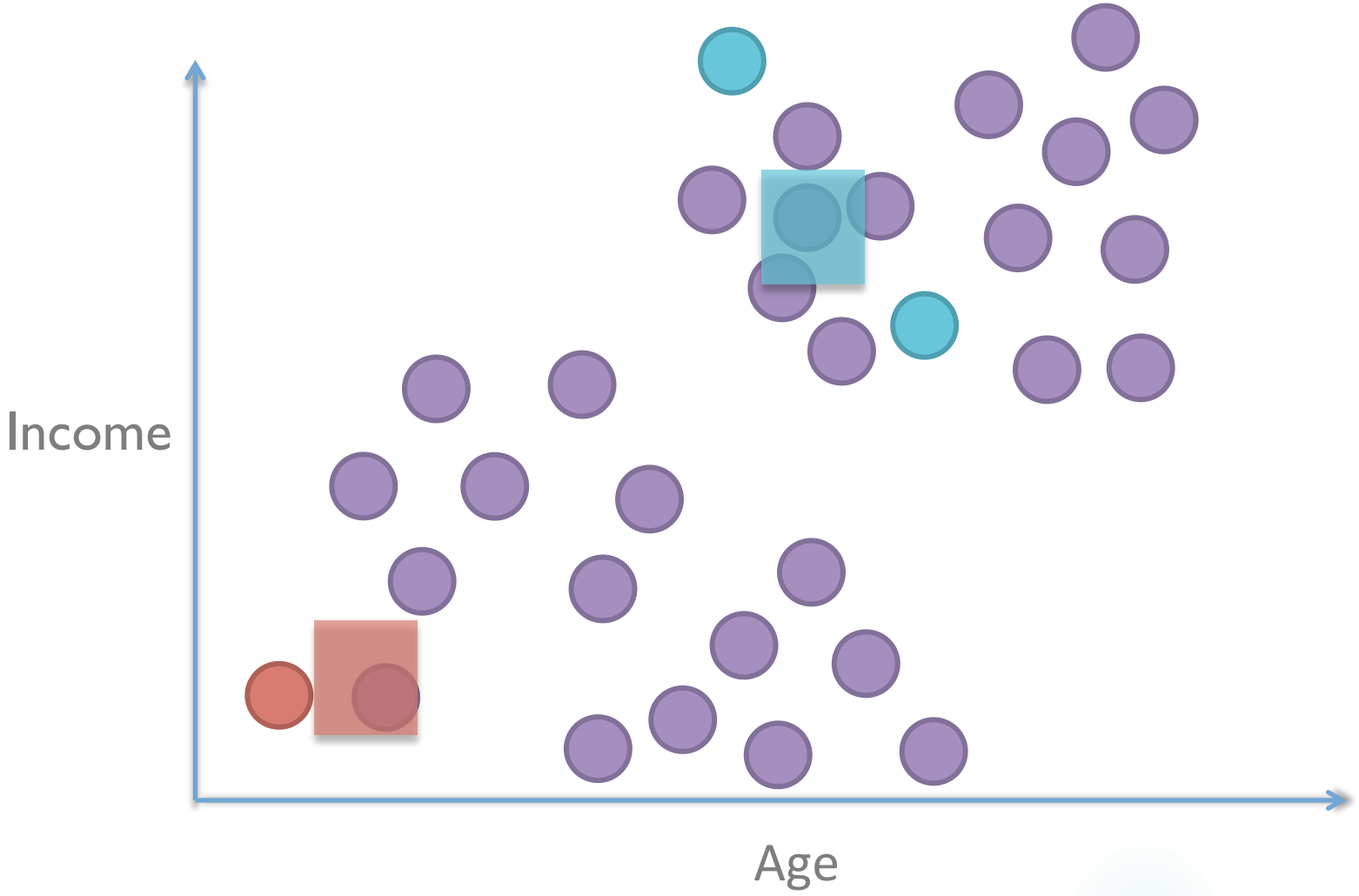
Mean Shift

Sample local density, follow gradient towards denser direction



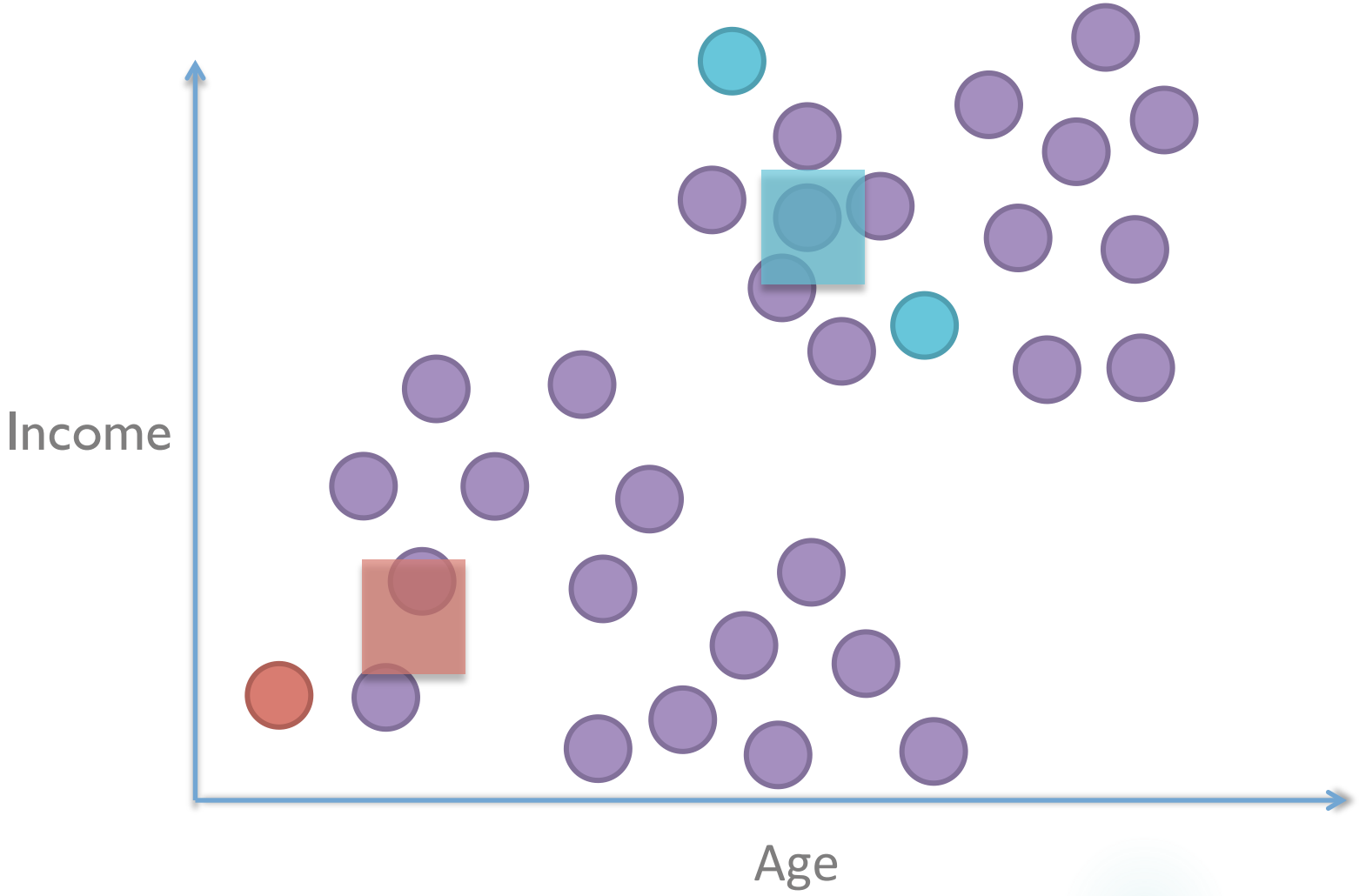
Mean Shift

Sample local density, follow gradient towards denser direction



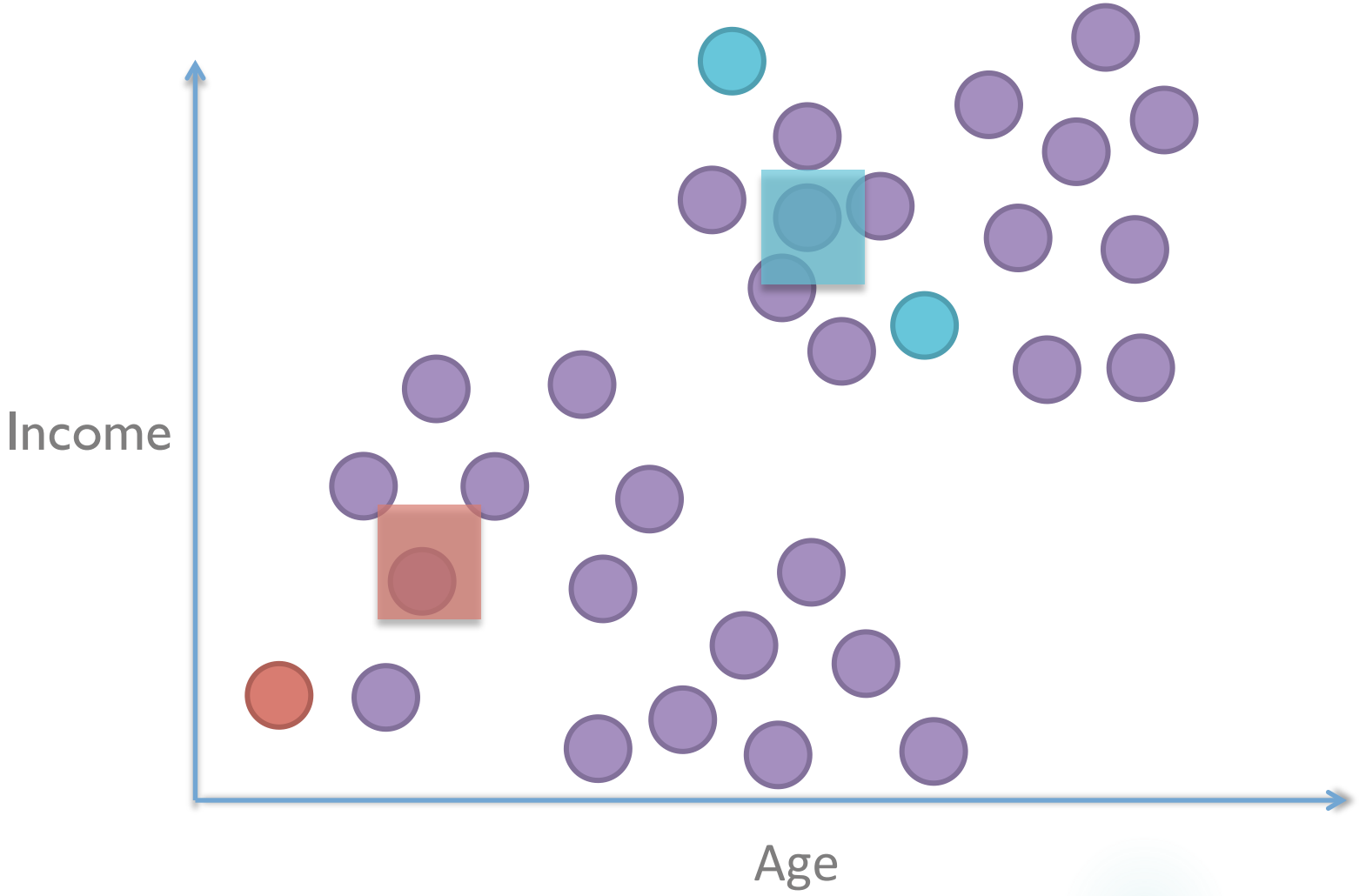
Mean Shift

Sample local density, follow gradient towards denser direction



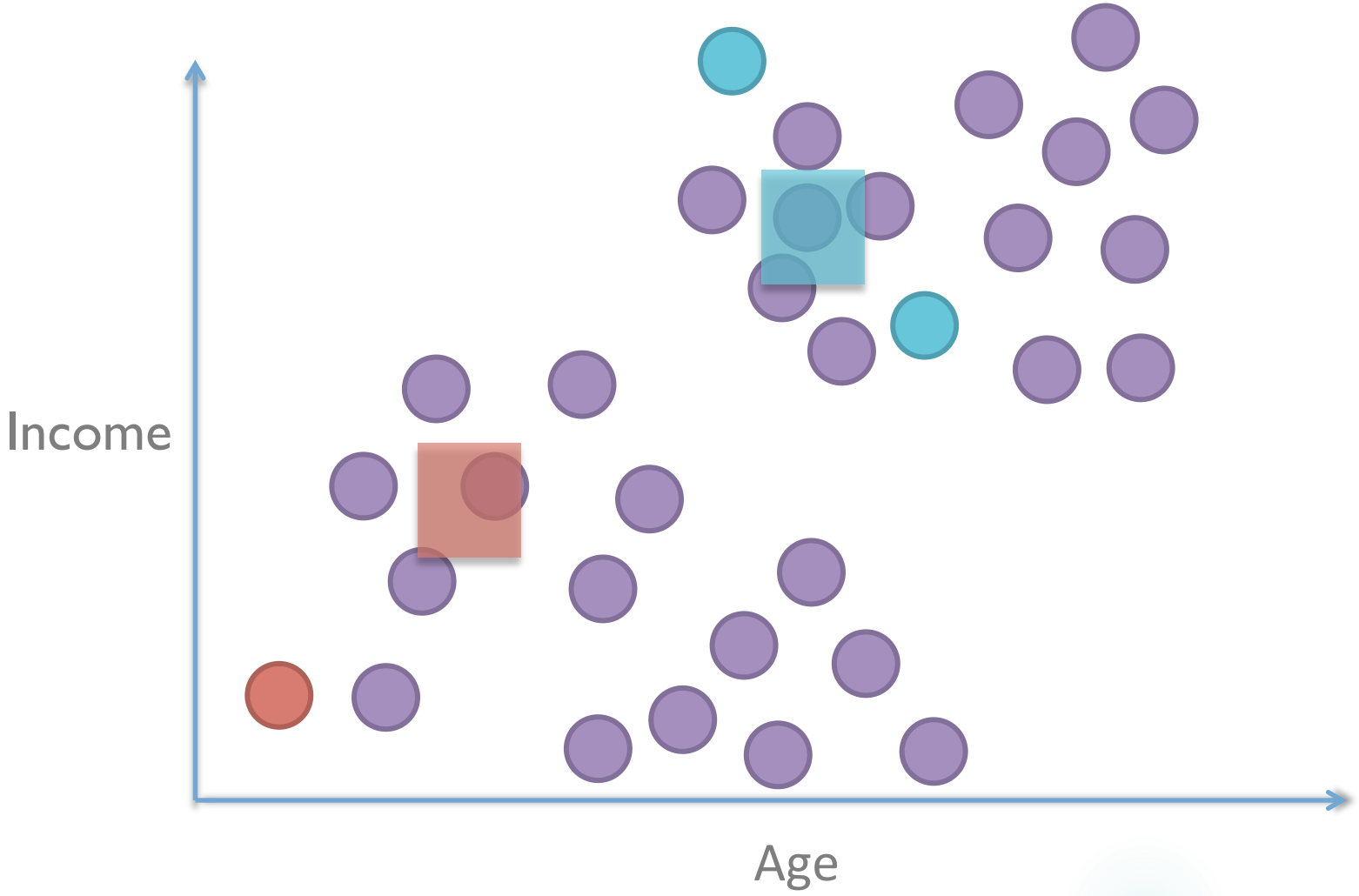
Mean Shift

Sample local density, follow gradient towards denser direction



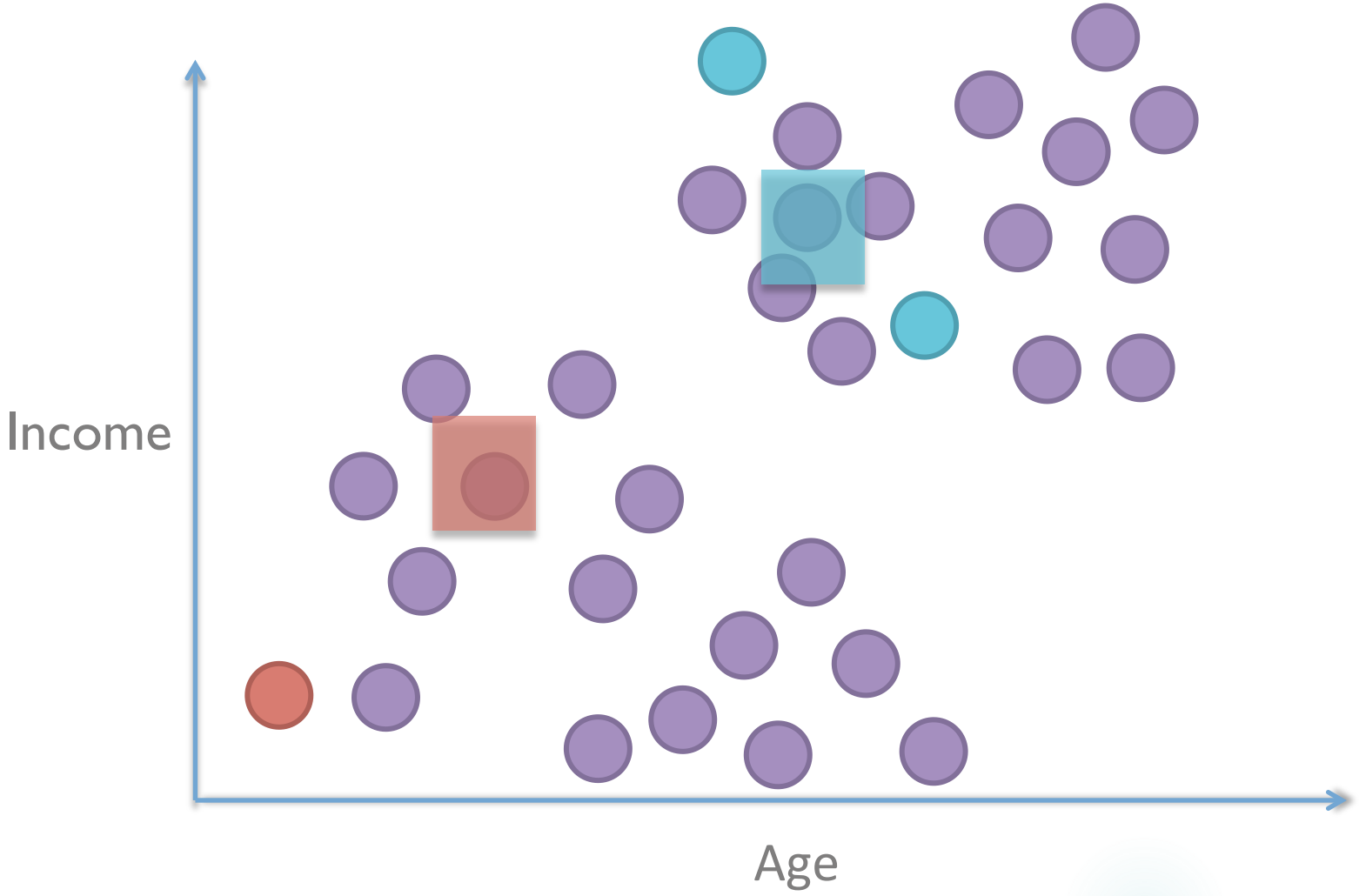
Mean Shift

Sample local density, follow gradient towards denser direction



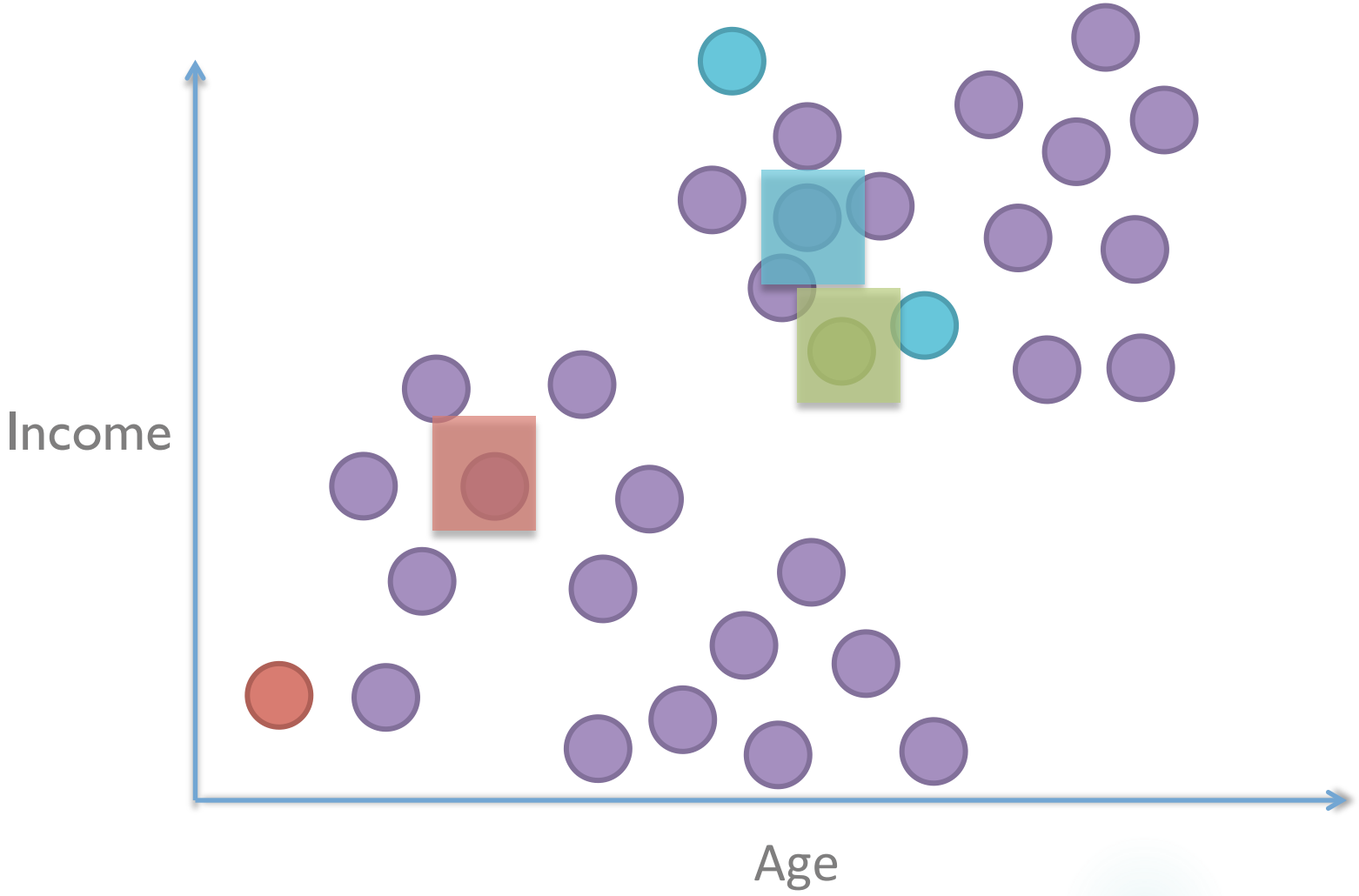
Mean Shift

Found local maximum. Stop.



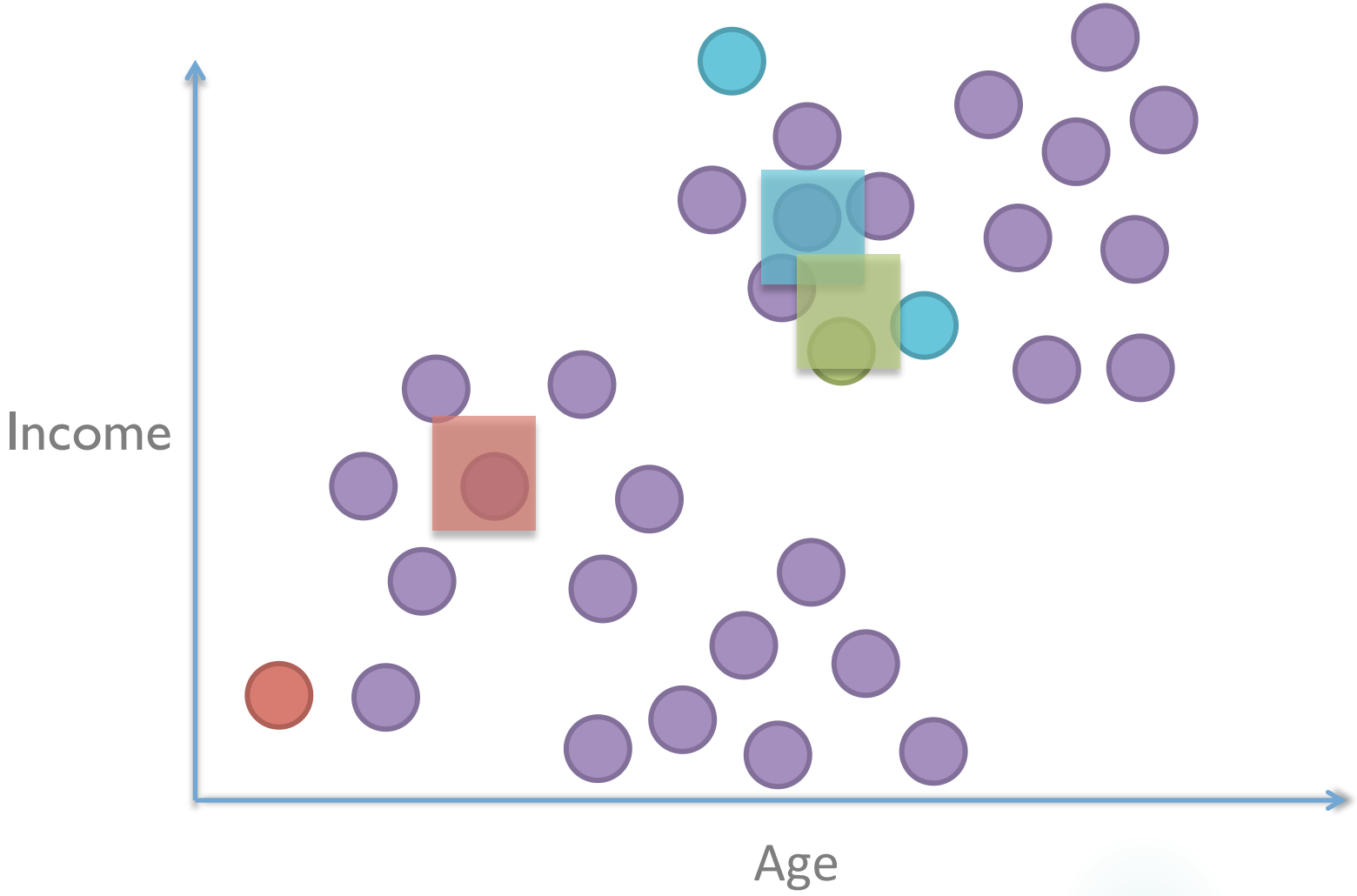
Mean Shift

Start at another point



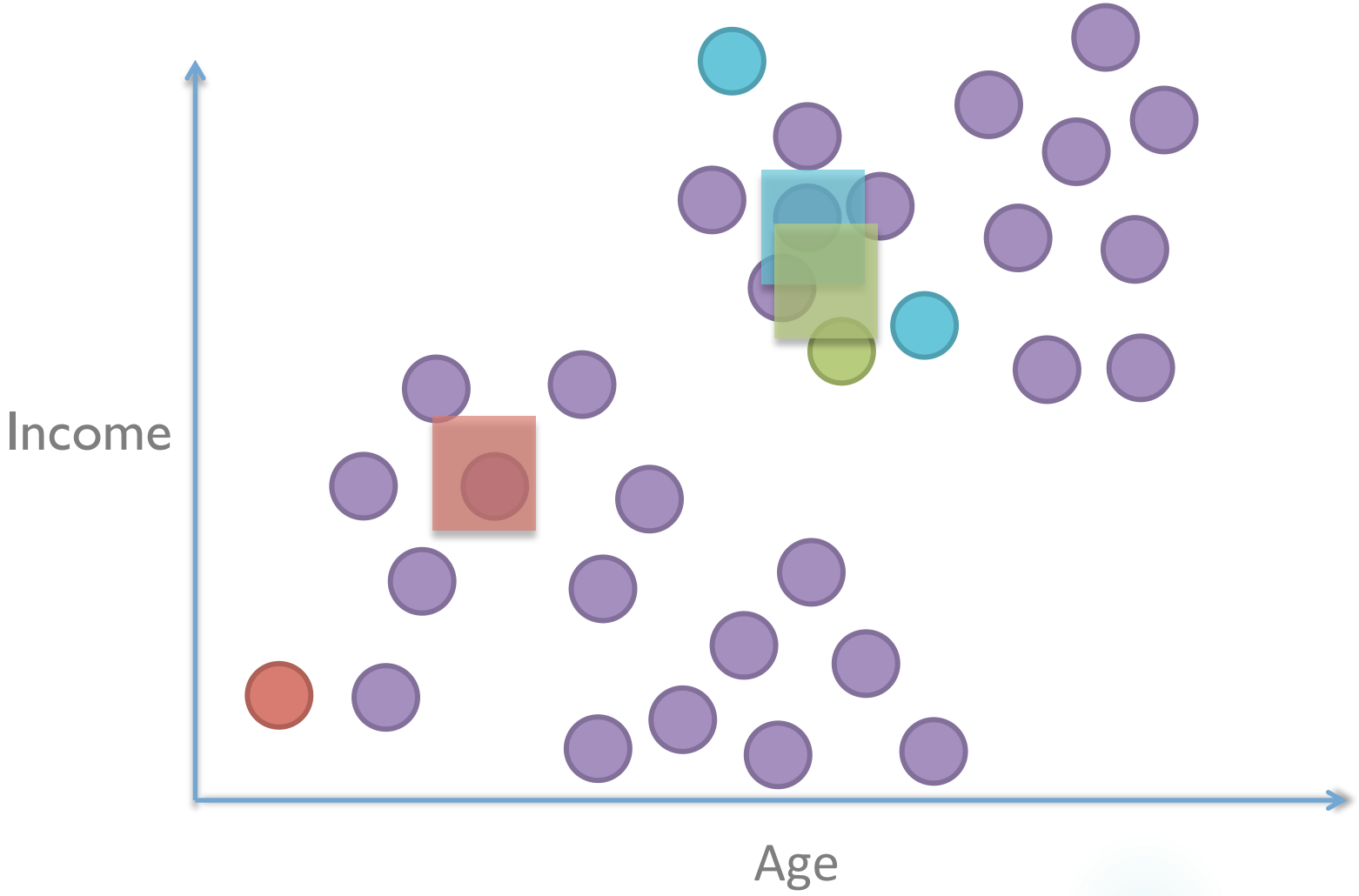
Mean Shift

Search for local density maximum



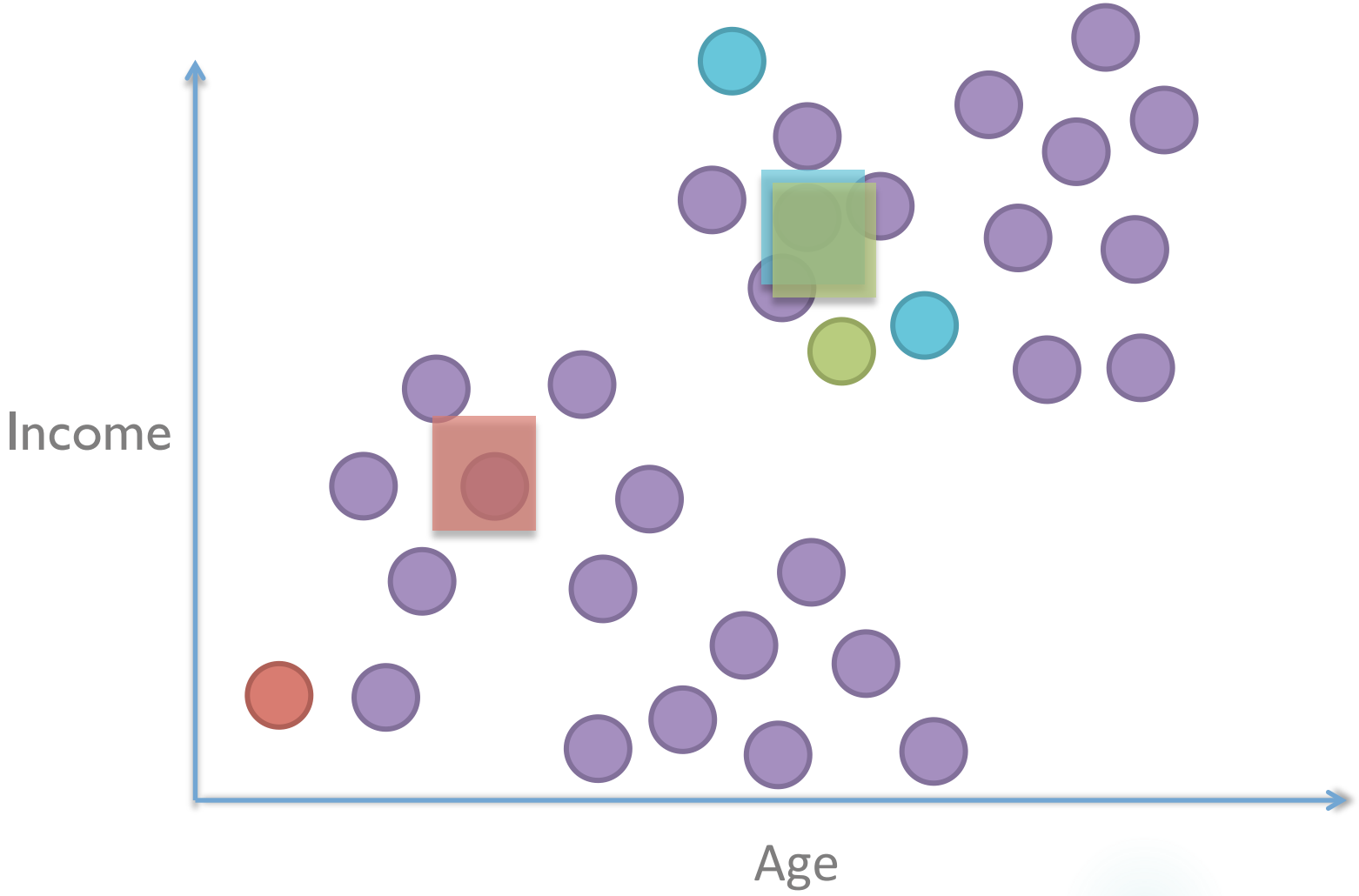
Mean Shift

Search for local density maximum



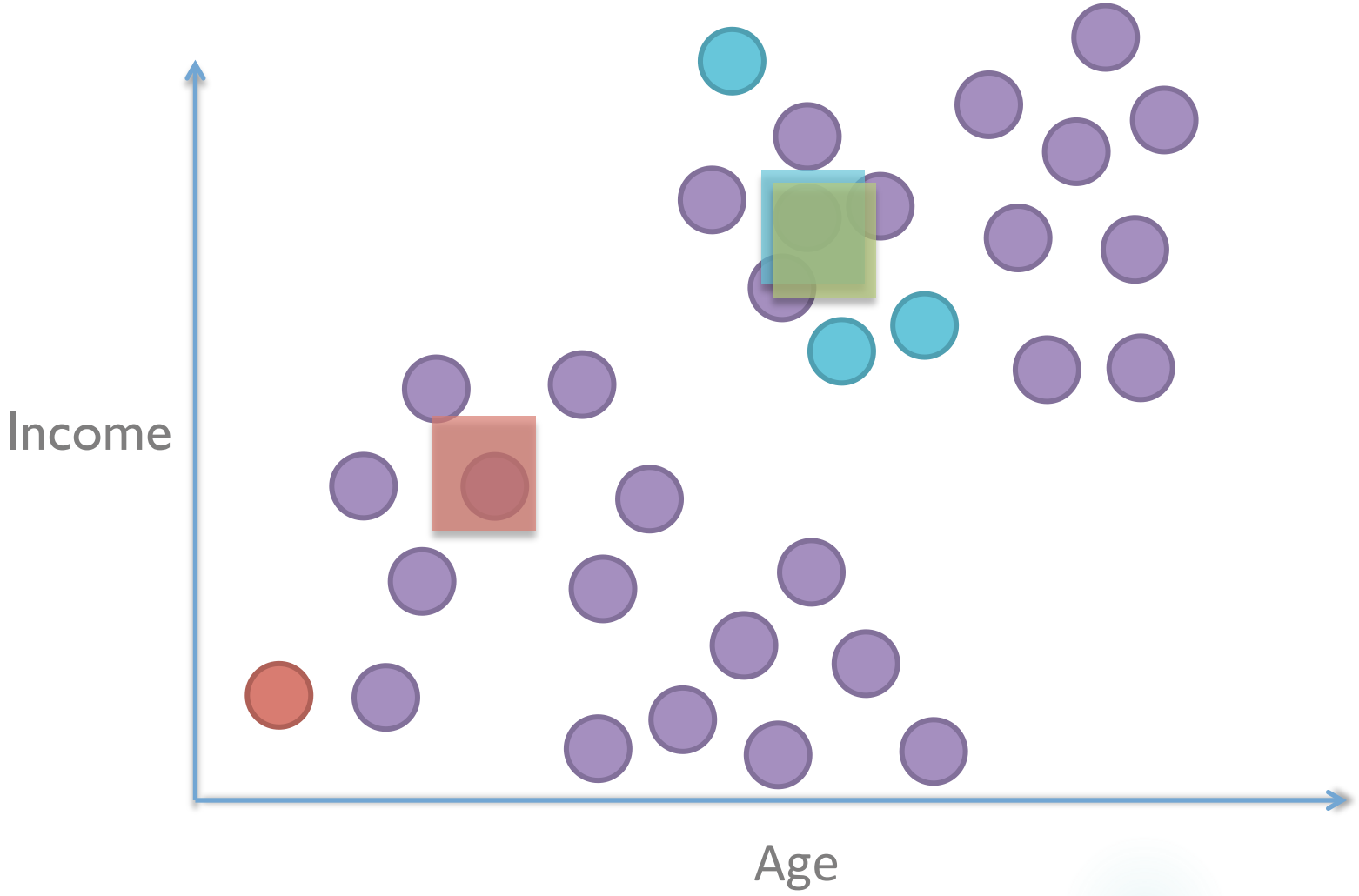
Mean Shift

Found same maximum, same cluster



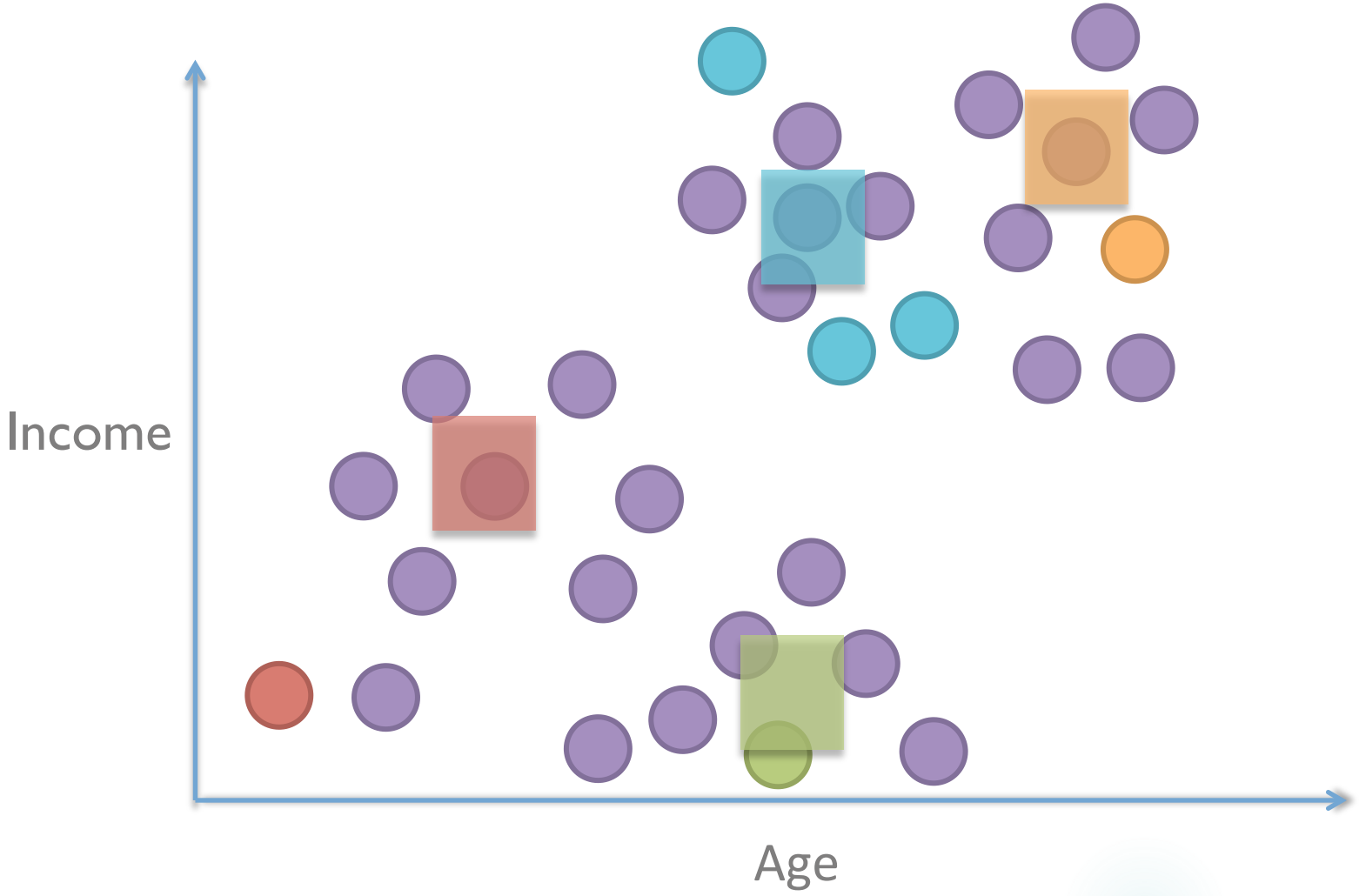
Mean Shift

Found same maximum, same cluster



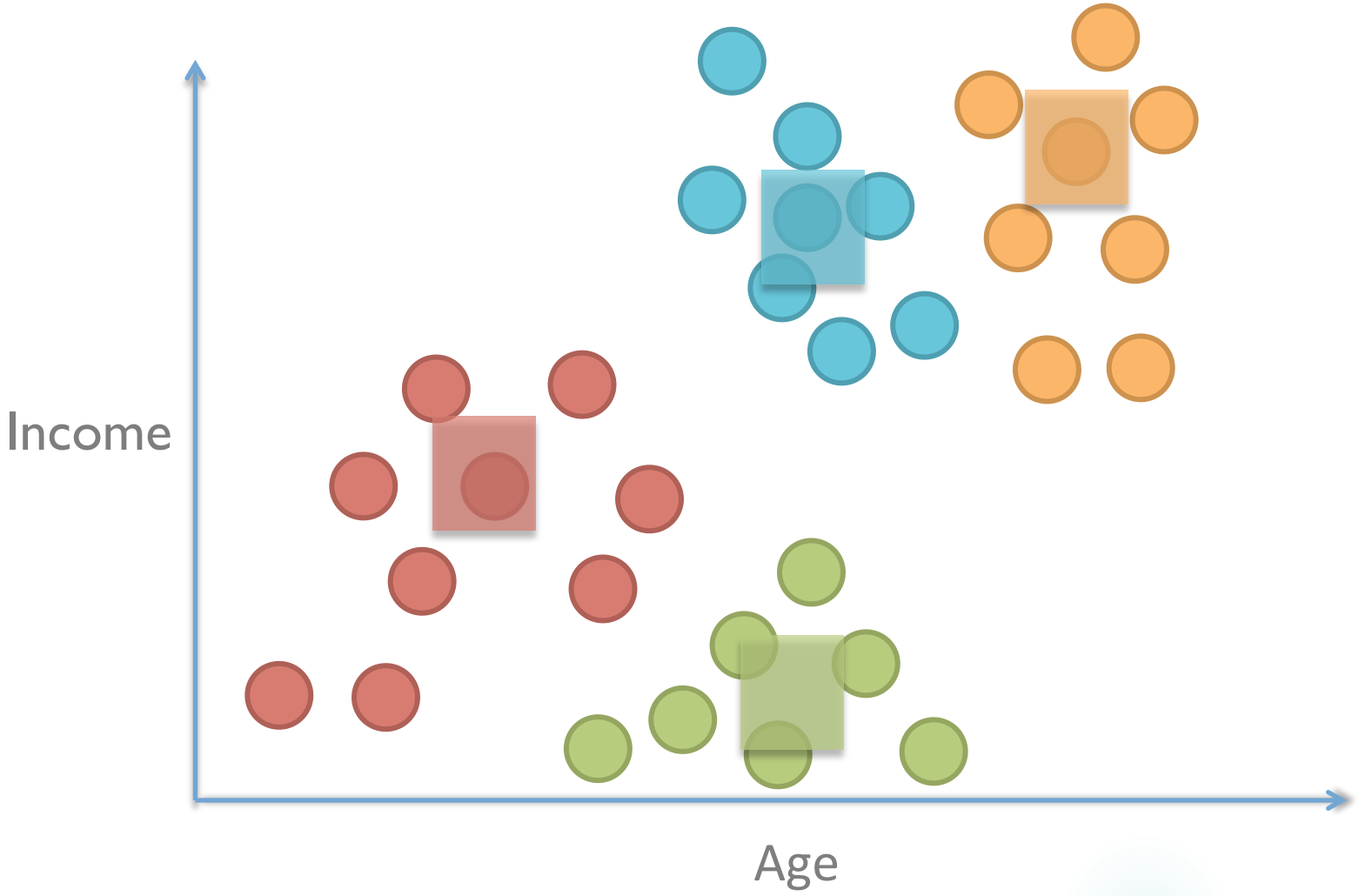
Mean Shift

Keep going like this, eventually it finds four unique local maxima



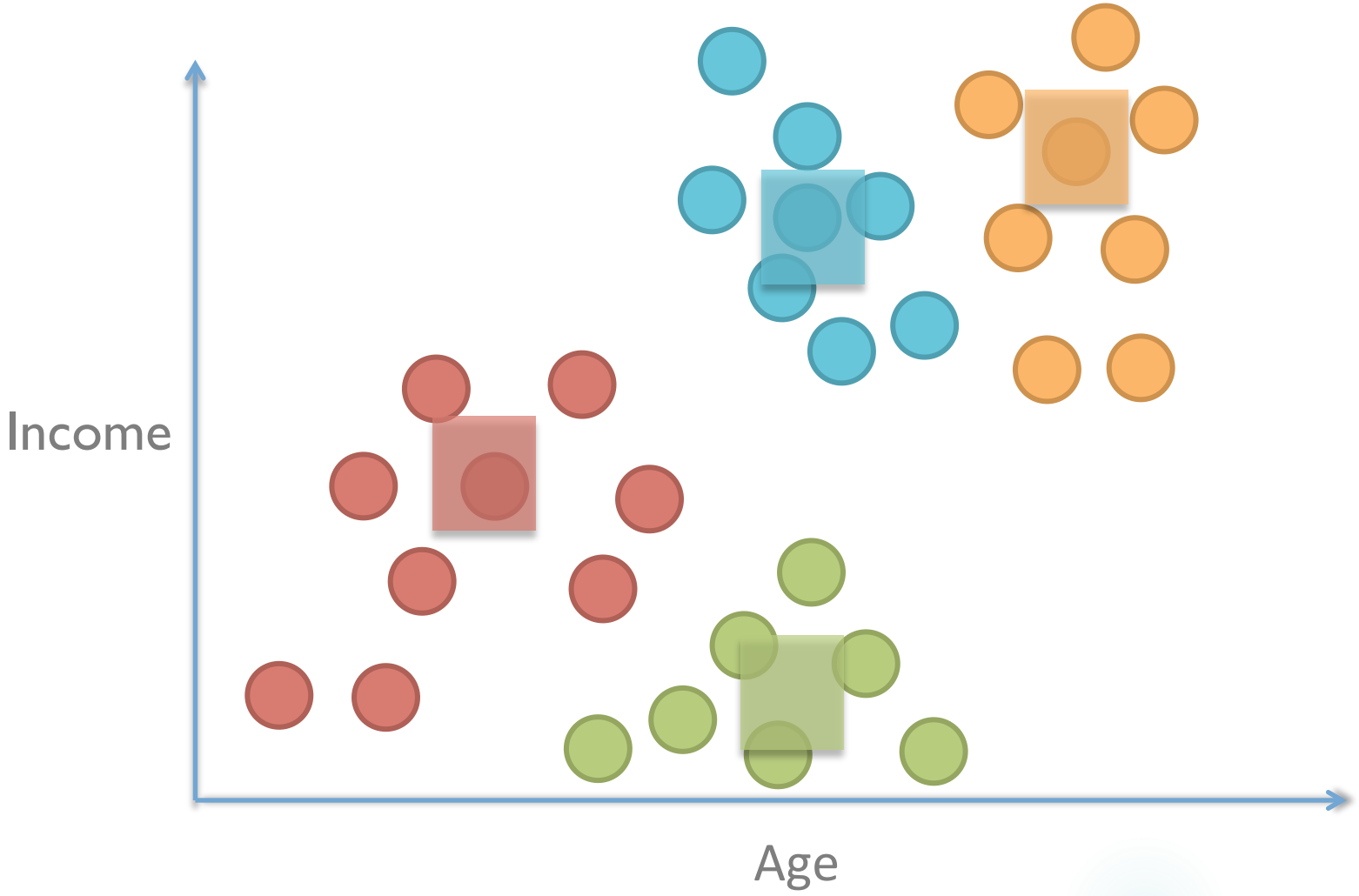
Mean Shift

Assigns points to centroids they fall to



Mean Shift

No cluster number or distance parameters



Mean shift: weighted mean

$$m(x) = \frac{\sum_{x_i \in W} x_i K(x_i - x)}{\sum_{x_i \in W} K(x_i - x)}$$

new mean

previous mean

sum over points in window

weighting ("kernel") function

Mean shift: strengths and weaknesses

Strengths:

1. Model-free: does assume number or shape of clusters
2. Can use just one parameter: window size
3. Robust to outliers

Weaknesses:

1. Result depends on window size
2. Selection of window size is not easy
3. Slow: $O(mn^2)$ for m iterations and n data points

Spectral Clustering Affinity Propagation

A brief exposition

Spectral Clustering and Affinity Propagation

Both methods cluster the data based on a similarity (adjacency) matrix

Spectral clustering: use similarity matrix to collapse data into lower dimensional space for clustering

Affinity propagation: use similarity matrix to find a subset of representative examples around which clusters are assembled

Similarity matrix

For *n* data points, the similarity matrix is a symmetric *nxn* matrix

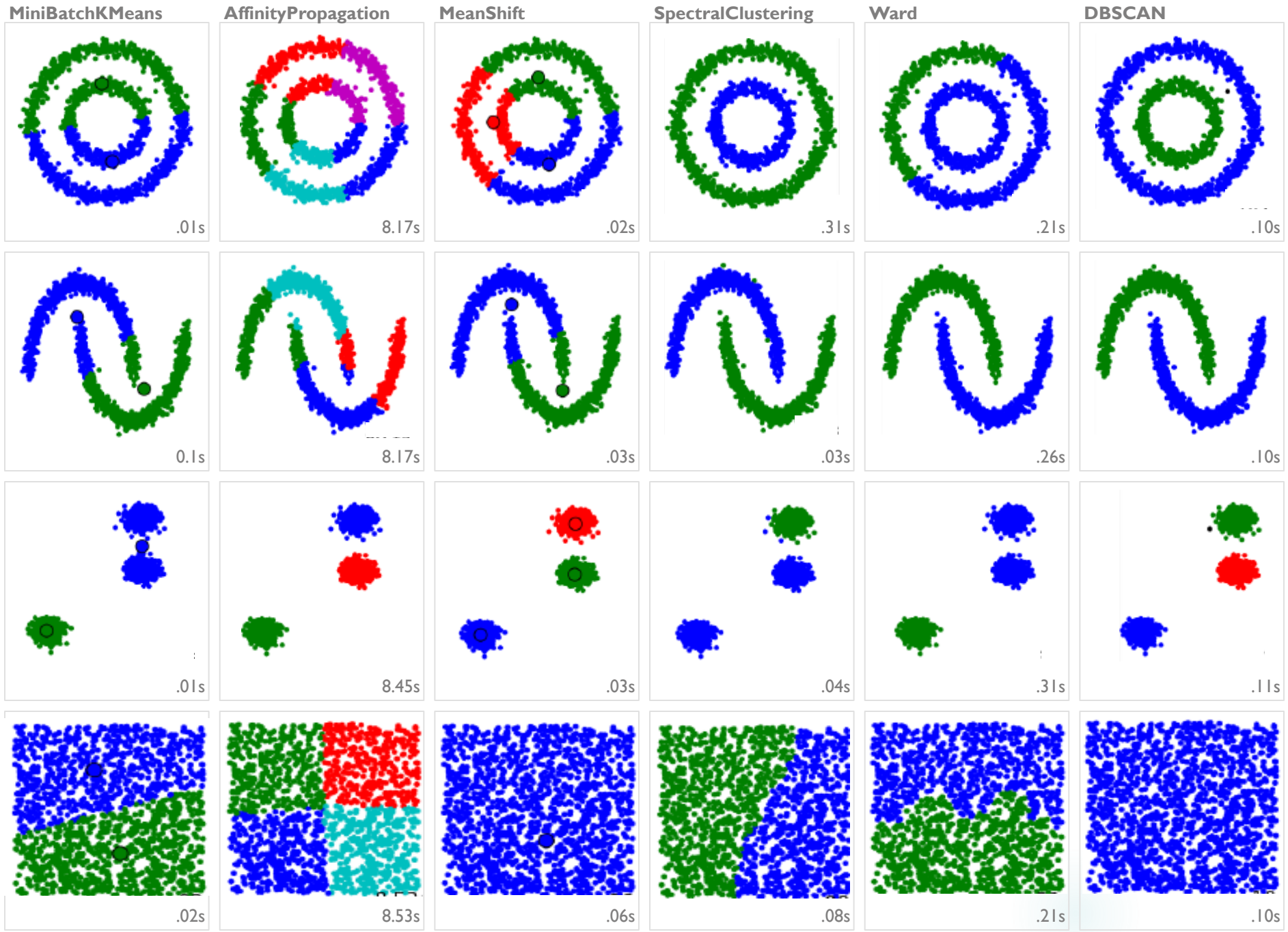
Entry (*i*, *j*) of matrix is similarity function *s(i, j)*: measures how similar are points *i* and *j*

Examples: $s(i, j) = -|\boldsymbol{r}_i - \boldsymbol{r}_j|^2$

$= (\boldsymbol{r}_i * \boldsymbol{r}_j)/|\boldsymbol{r}_i||\boldsymbol{r}_j|$

Comparing algorithms

A NOTE OF CAUTION

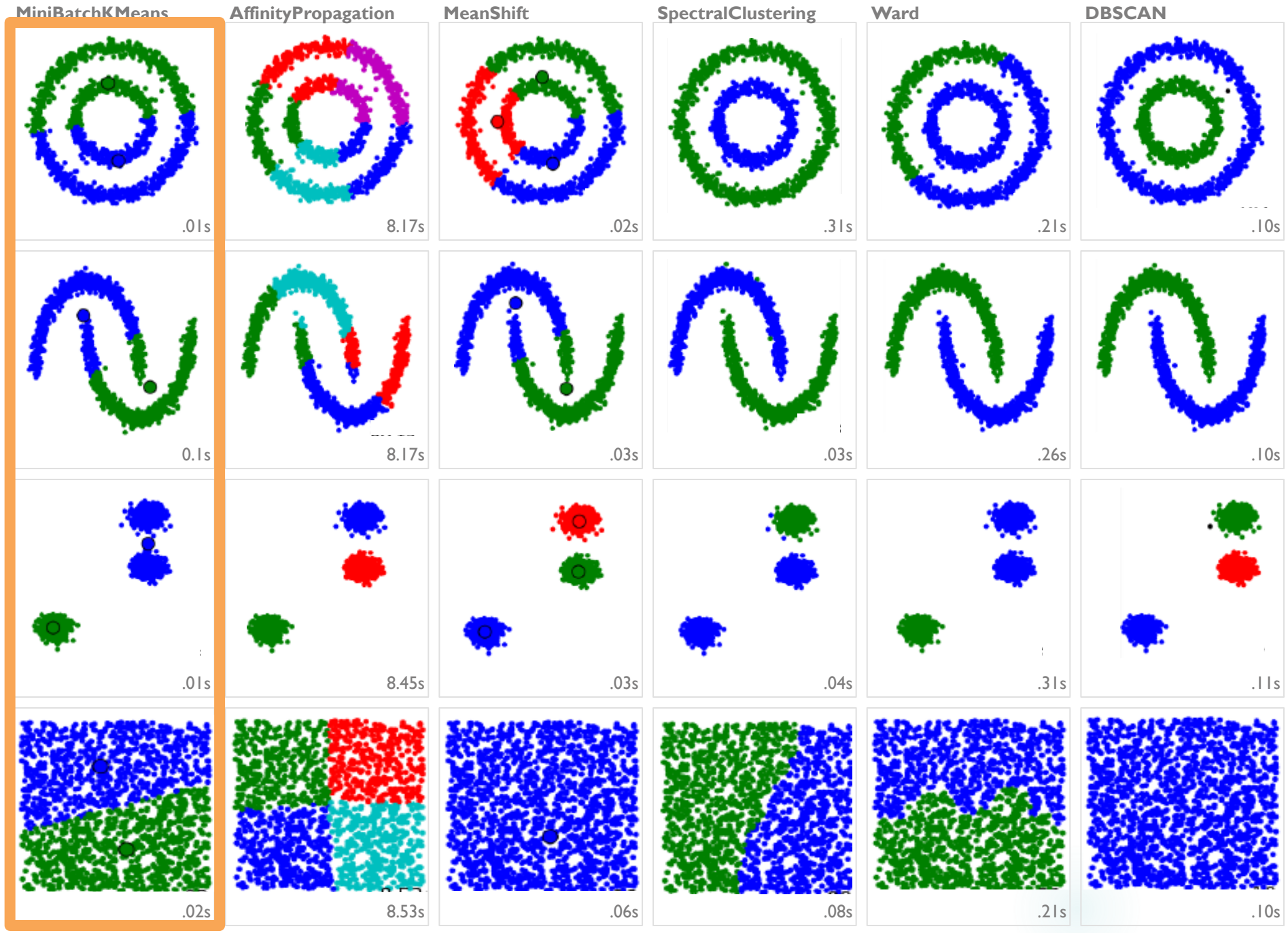

MiniBatchKMeans
AffinityPropagation
MeanShift
SpectralClustering
Ward
DBSCAN
.01s
8.17s
.02s
.31s
.21s
.10s
0.1s
8.17s
.03s
.03s
.26s
.10s
.01s
8.45s
.03s
.04s
.31s
.11s
.02s
8.53s
.06s
.08s
.21s
.10s

Method name	Parameters	Scalability	Usecase	Geometry (metric used)
K-means	Number of clusters	Very large *n_samples* medium *n_clusters* with *MiniBatch code*	General purpose, even cluster size, flat geometry, not too many clusters	Distances between points
Affinity propagation	Damping, sample preferences	Not scalable with n_samples	Many clusters, uneven cluster size, non-flat geometry	Graph distance (e.g. nearest-neighbor graph)
Mean-shift	Bandwidth	Not scalable with n_samples	Many clusters, uneven cluster size, non-flat geometry	Distances between points
Spectral clustering	Number of clusters	Medium *n_samples*, small *n_clusters*	Few clusters, even cluster size, non-flat geometry	Graph distance (e.g. nearest-neighbor graph)
Hierarchical clustering	Number of clusters	Large *n_samples* and *n_clusters*	Many clusters, possibly connectivity constraints	Distances between points
DBSCAN	Neighborhood size	Very large *n_samples*, medium *n_clusters*	Non-flat geometry, uneven cluster sizes	Distances between nearest points

Method name	Parameters	Scalability	Usecase	Geometry (metric used)
K-means	Number of clusters	Very large *n_samples* medium *n_clusters* with *MiniBatch code*	General purpose, even cluster size, flat geometry, not too many clusters	Distances between points
Affinity propagation	Damping, sample preferences	Not scalable with n_samples	Many clusters, uneven cluster size, non-flat geometry	Graph distance (e.g. nearest-neighbor graph)
Mean-shift	Bandwidth	Not scalable with n_samples	Many clusters, uneven cluster size, non-flat geometry	Distances between points
Spectral clustering	Number of clusters	Medium *n_samples*, small *n_clusters*	Few clusters, even cluster size, non-flat geometry	Graph distance (e.g. nearest-neighbor graph)
Hierarchical clustering	Number of clusters	Large *n_samples* and *n_clusters*	Many clusters, possibly connectivity constraints	Distances between points
DBSCAN	Neighborhood size	Very large *n_samples*, medium *n_clusters*	Non-flat geometry, uneven cluster sizes	Distances between nearest points

K Means:

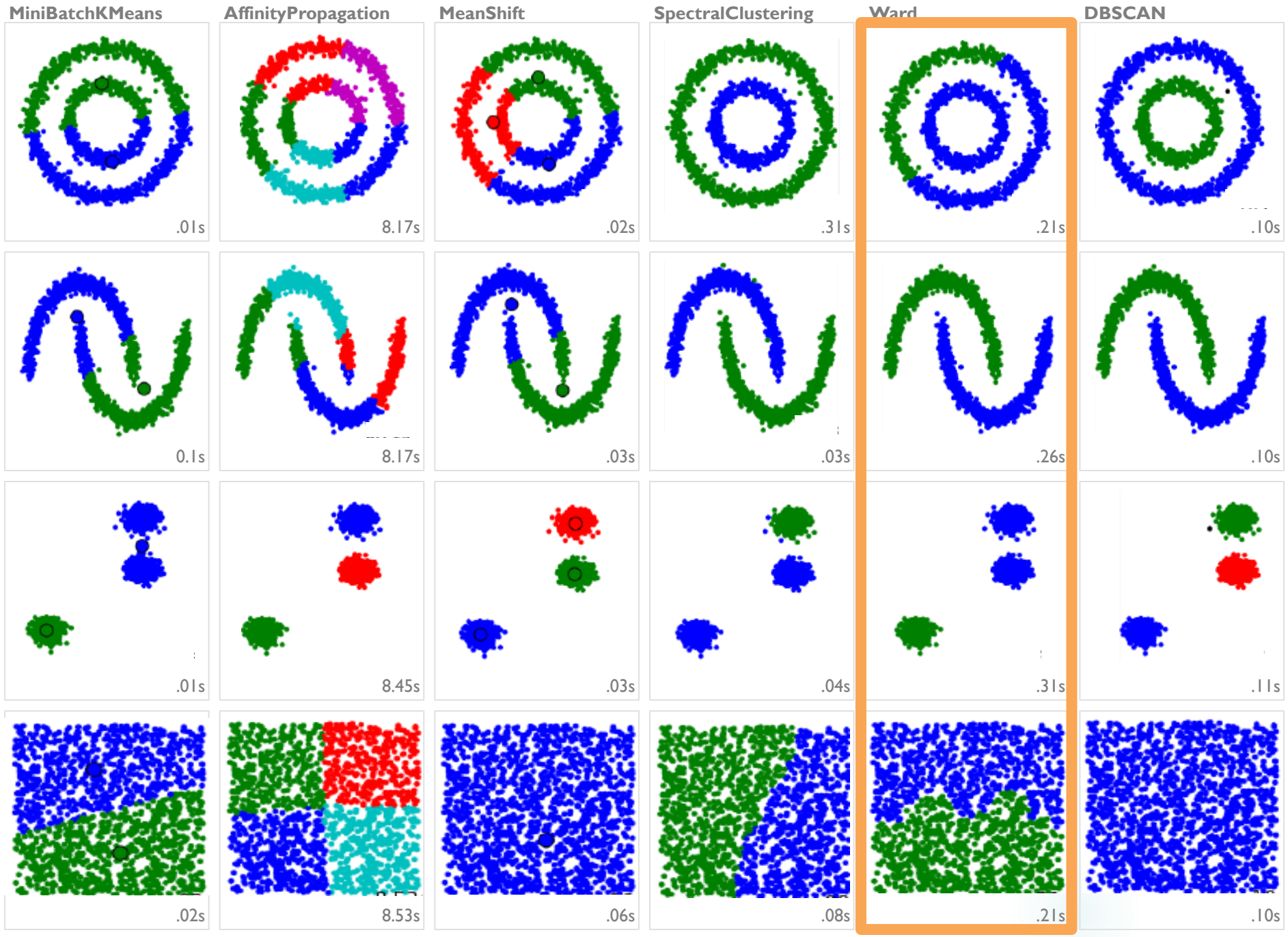
- MiniBatch version is fast (big data).
- Have to try k values (k not too big).
- Tends to find even sized clusters.
- Bad with non-spherical cluster shapes.


MiniBatchKMeans
AffinityPropagation
MeanShift
SpectralClustering
Ward
DBSCAN
.01s
8.17s
.02s
.31s
.21s
.10s
0.1s
8.17s
.03s
.03s
.26s
.10s
.01s
8.45s
.03s
.04s
.31s
.11s
.02s
8.53s
.06s
.08s
.21s
.10s

Method name	Parameters	Scalability	Usecase	Geometry (metric used)
K-means	Number of clusters	Very large *n_samples* medium *n_clusters* with *MiniBatch code*	General purpose, even cluster size, flat geometry, not too many clusters	Distances between points
Affinity propagation	Damping, sample preferences	Not scalable with n_samples	Many clusters, uneven cluster size, non-flat geometry	Graph distance (e.g. nearest-neighbor graph)
Mean-shift	Bandwidth	Not scalable with n_samples	Many clusters, uneven cluster size, non-flat geometry	Distances between points
Spectral clustering	Number of clusters	Medium *n_samples*, small *n_clusters*	Few clusters, even cluster size, non-flat geometry	Graph distance (e.g. nearest-neighbor graph)
Hierarchical clustering	Number of clusters	Large *n_samples* and *n_clusters*	Many clusters, possibly connectivity constraints	Distances between points
DBSCAN	Neighborhood size	Very large *n_samples*, medium *n_clusters*	Non-flat geometry, uneven cluster sizes	Distances between nearest points

Hierarchical Clustering:

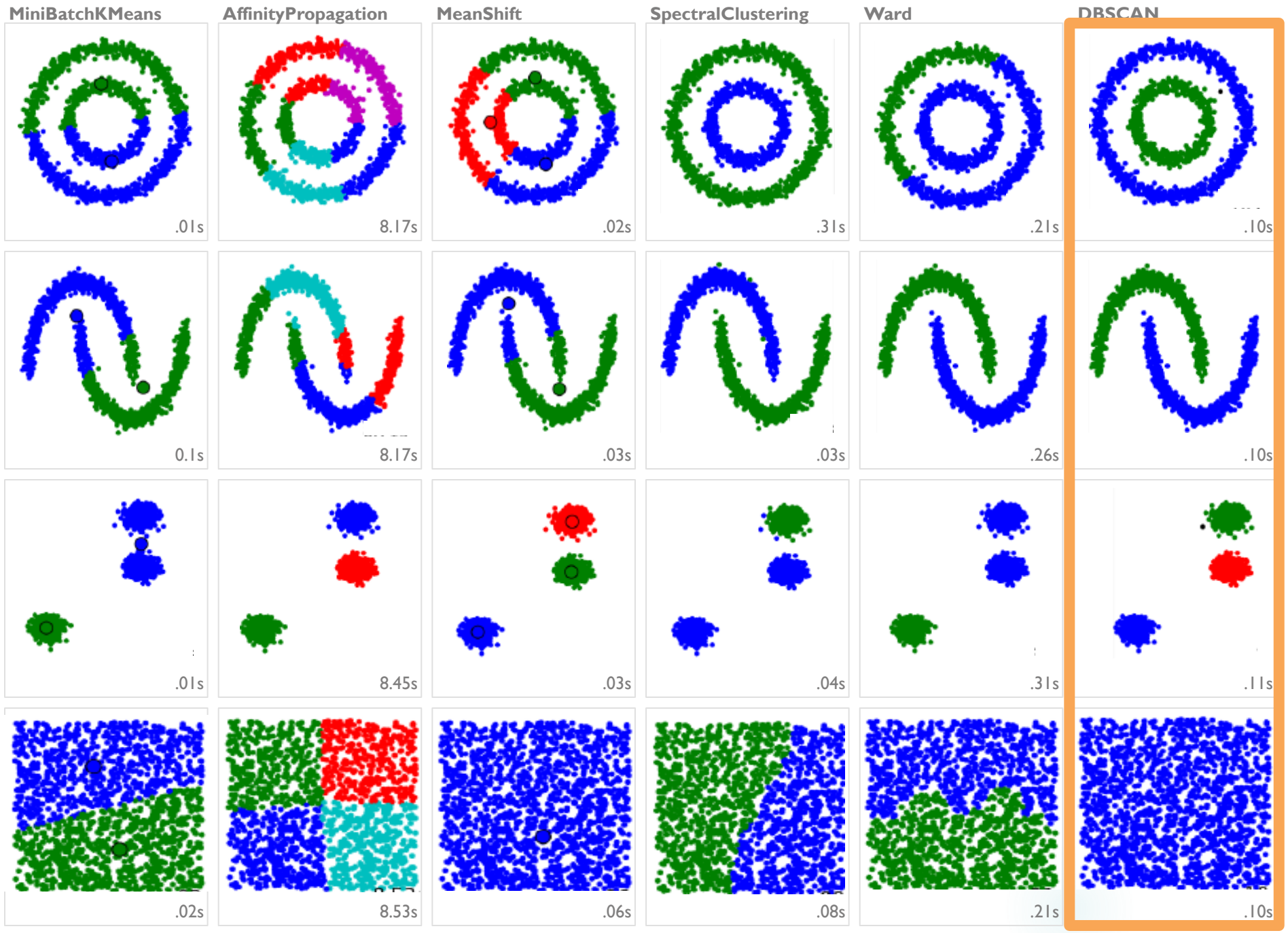
- You get a full hierarchy tree. Useful for some problems.
- Have to try k values.
- Finds uneven cluster sizes (one is big, some are tiny).
  A lot of distance metric and linkage options.
- Slow: O(n^2).


MiniBatchKMeans
AffinityPropagation
MeanShift
SpectralClustering
Ward
DBSCAN
.01s
8.17s
.02s
.31s
.21s
.10s
0.1s
8.17s
.03s
.03s
.26s
.10s
.01s
8.45s
.03s
.04s
.31s
.11s
.02s
8.53s
.06s
.08s
.21s
.10s

Method name	Parameters	Scalability	Usecase	Geometry (metric used)
K-means	Number of clusters	Very large *n_samples* medium *n_clusters* with *MiniBatch code*	General purpose, even cluster size, flat geometry, not too many clusters	Distances between points
Affinity propagation	Damping, sample preferences	Not scalable with n_samples	Many clusters, uneven cluster size, non-flat geometry	Graph distance (e.g. nearest-neighbor graph)
Mean-shift	Bandwidth	Not scalable with n_samples	Many clusters, uneven cluster size, non-flat geometry	Distances between points
Spectral clustering	Number of clusters	Medium *n_samples*, small *n_clusters*	Few clusters, even cluster size, non-flat geometry	Graph distance (e.g. nearest-neighbor graph)
Hierarchical clustering	Number of clusters	Large *n_samples* and *n_clusters*	Many clusters, possibly connectivity constraints	Distances between points
DBSCAN	Neighborhood size	Very large *n_samples*, medium *n_clusters*	Non-flat geometry, uneven cluster sizes	Distances between nearest points

DBSCAN

Density based, on the money with the right parameters.
Have to try *epsilon* (and *num_clu*) values.
Can find uneven cluster sizes.
Full distance metric options.
Can handle tons of data and weird shapes.
Too small epsilon (too many clusters) is not trustworthy.

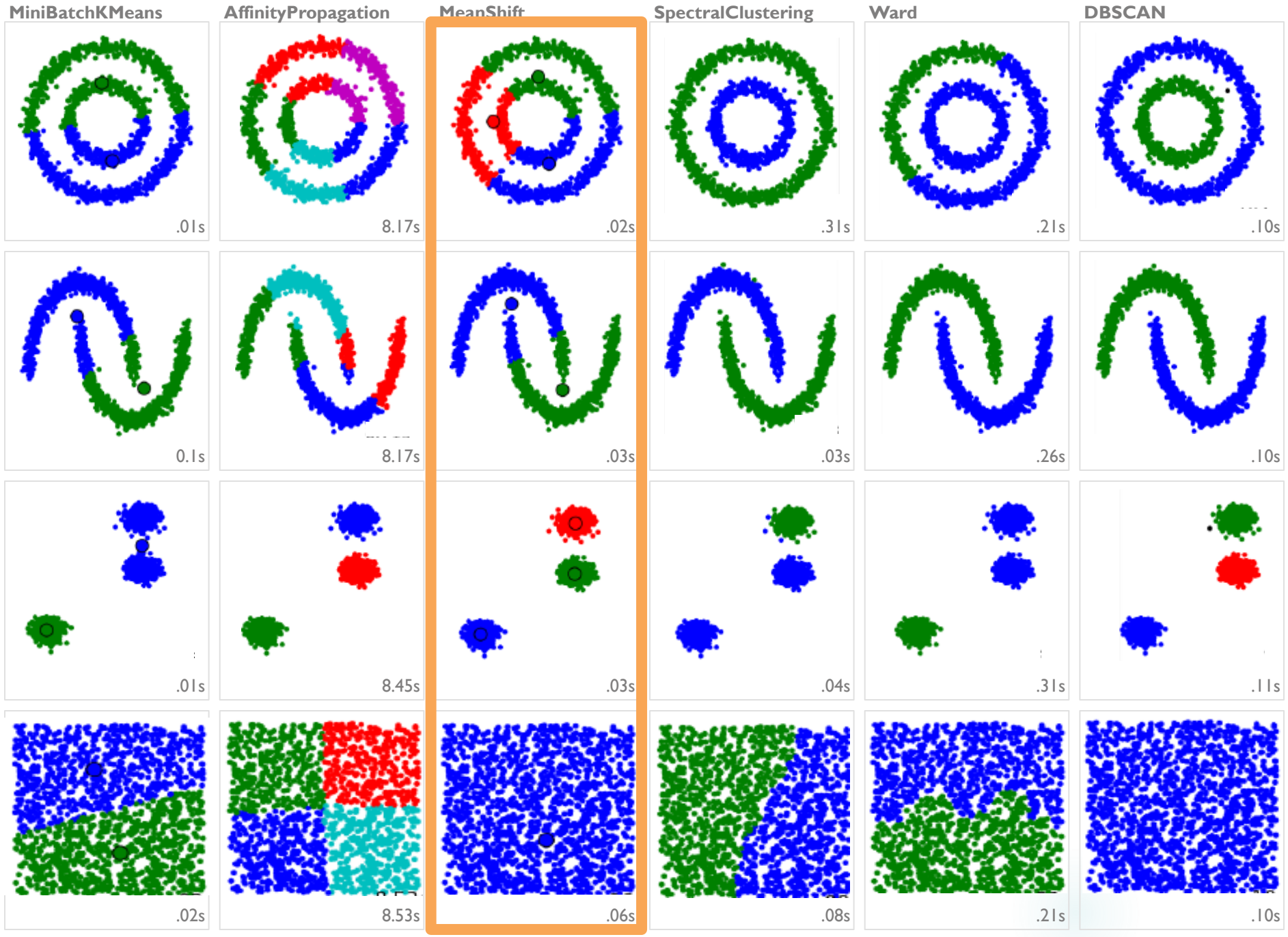

MiniBatchKMeans
AffinityPropagation
MeanShift
SpectralClustering
Ward
DBSCAN
.01s
8.17s
.02s
.31s
.21s
.10s
0.1s
8.17s
.03s
.03s
.26s
.10s
.01s
8.45s
.03s
.04s
.31s
.11s
.02s
8.53s
.06s
.08s
.21s
.10s

Method name	Parameters	Scalability	Usecase	Geometry (metric used)
K-means	Number of clusters	Very large *n_samples* medium *n_clusters* with *MiniBatch code*	General purpose, even cluster size, flat geometry, not too many clusters	Distances between points
Affinity propagation	Damping, sample preferences	Not scalable with n_samples	Many clusters, uneven cluster size, non-flat geometry	Graph distance (e.g. nearest-neighbor graph)
Mean-shift	Bandwidth	Not scalable with n_samples	Many clusters, uneven cluster size, non-flat geometry	Distances between points
Spectral clustering	Number of clusters	Medium *n_samples*, small *n_clusters*	Few clusters, even cluster size, non-flat geometry	Graph distance (e.g. nearest-neighbor graph)
Hierarchical clustering	Number of clusters	Large *n_samples* and *n_clusters*	Many clusters, possibly connectivity constraints	Distances between points
DBSCAN	Neighborhood size	Very large *n_samples*, medium *n_clusters*	Non-flat geometry, uneven cluster sizes	Distances between nearest points

Mean-shift

- You DON'T have to guess k or other parameters.
- Can find uneven cluster sizes.
- Slow with a lot of data
- Lots of clusters? No problem
- Doesn't handle weird shapes well.

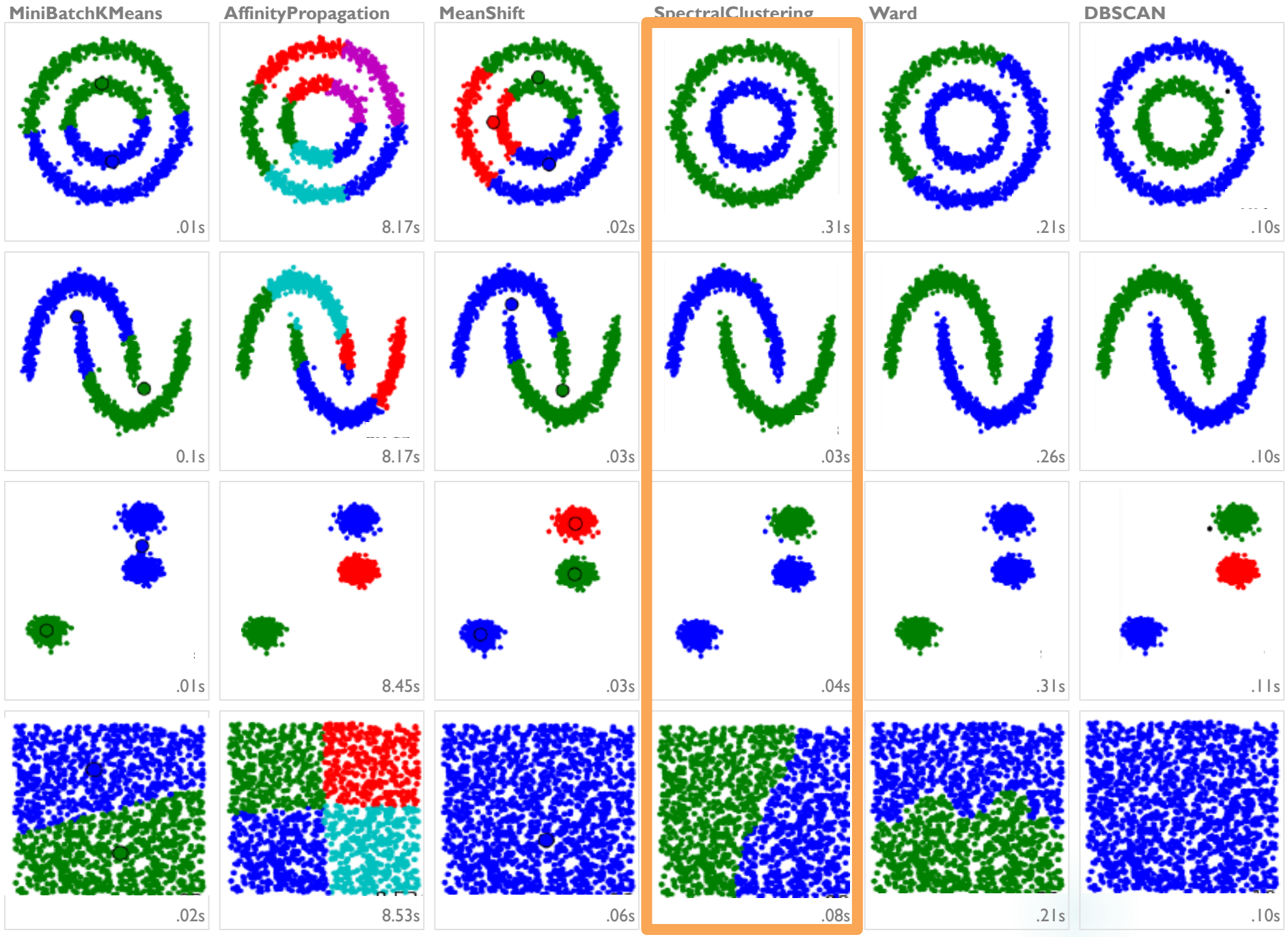
Euclidean distance only.


MiniBatchKMeans
AffinityPropagation
MeanShift
SpectralClustering
Ward
DBSCAN
.01s
8.17s
.02s
.31s
.21s
.10s
0.1s
8.17s
.03s
.03s
.26s
.10s
.01s
8.45s
.03s
.04s
.31s
.11s
.02s
8.53s
.06s
.08s
.21s
.10s

Method name	Parameters	Scalability	Usecase	Geometry (metric used)
K-means	Number of clusters	Very large *n_samples* medium *n_clusters* with *MiniBatch code*	General purpose, even cluster size, flat geometry, not too many clusters	Distances between points
Affinity propagation	Damping, sample preferences	Not scalable with n_samples	Many clusters, uneven cluster size, non-flat geometry	Graph distance (e.g. nearest-neighbor graph)
Mean-shift	Bandwidth	Not scalable with n_samples	Many clusters, uneven cluster size, non-flat geometry	Distances between points
Spectral clustering	Number of clusters	Medium *n_samples*, small *n_clusters*	Few clusters, even cluster size, non-flat geometry	Graph distance (e.g. nearest-neighbor graph)
Hierarchical clustering	Number of clusters	Large *n_samples* and *n_clusters*	Many clusters, possibly connectivity constraints	Distances between points
DBSCAN	Neighborhood size	Very large *n_samples*, medium *n_clusters*	Non-flat geometry, uneven cluster sizes	Distances between nearest points

Spectral clustering:

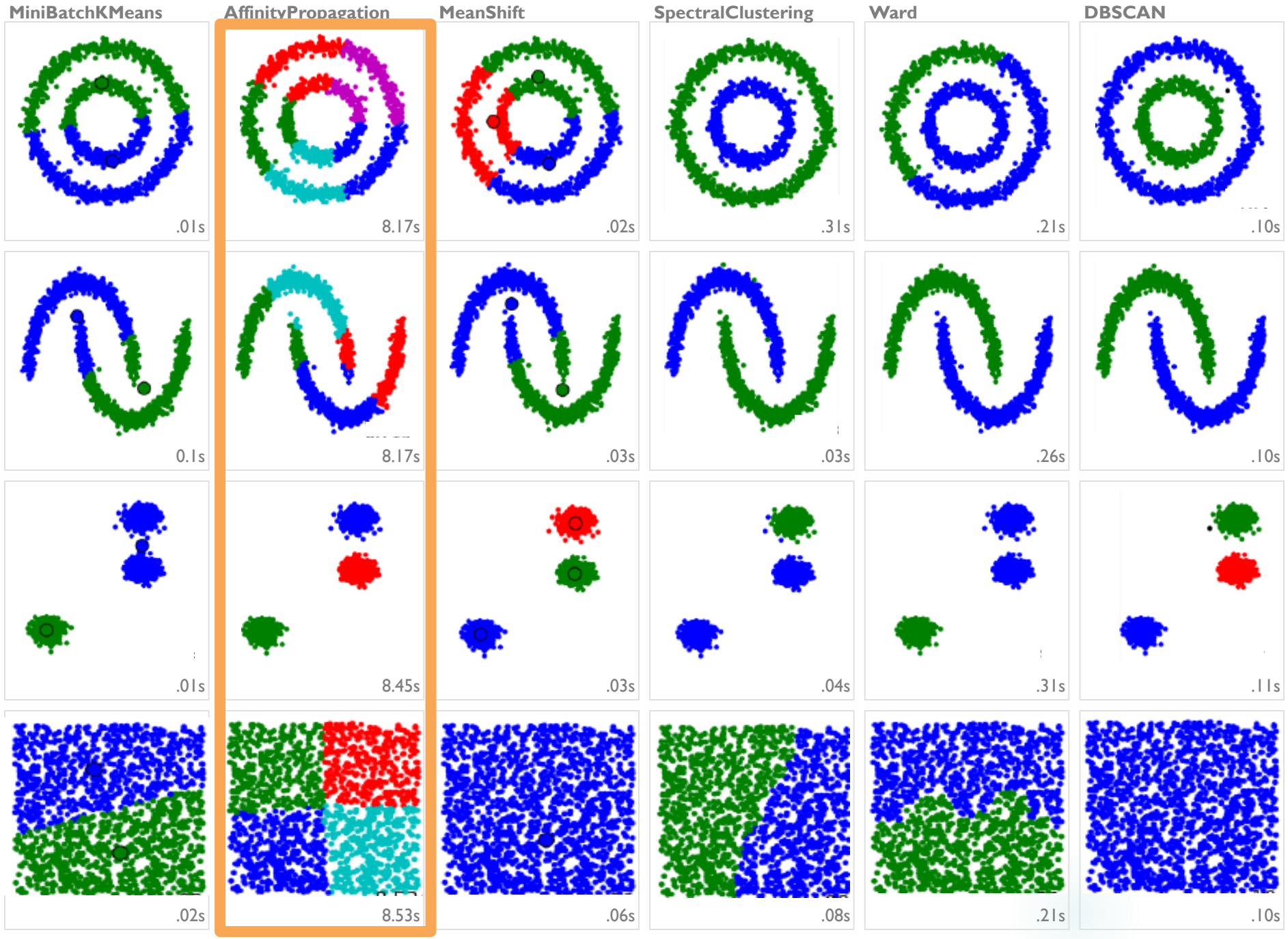
- Best with sparse distances.
- Tends to find even sized clusters.
- Good with only a few clusters, poor performance with many.
- Can handle weirdest connectivity shapes (like concentric circles)
- Euclidean distance only.


MiniBatchKMeans
AffinityPropagation
MeanShift
SpectralClustering
Ward
DBSCAN
.01s
8.17s
.02s
.31s
.21s
.10s
0.1s
8.17s
.03s
.03s
.26s
.10s
.01s
8.45s
.03s
.04s
.31s
.11s
.02s
8.53s
.06s
.08s
.21s
.10s

Method name	Parameters	Scalability	Usecase	Geometry (metric used)
K-means	Number of clusters	Very large *n_samples* medium *n_clusters* with *MiniBatch code*	General purpose, even cluster size, flat geometry, not too many clusters	Distances between points
Affinity propagation	Damping, sample preferences	Not scalable with n_samples	Many clusters, uneven cluster size, non-flat geometry	Graph distance (e.g. nearest-neighbor graph)
Mean-shift	Bandwidth	Not scalable with n_samples	Many clusters, uneven cluster size, non-flat geometry	Distances between points
Spectral clustering	Number of clusters	Medium *n_samples*, small *n_clusters*	Few clusters, even cluster size, non-flat geometry	Graph distance (e.g. nearest-neighbor graph)
Hierarchical clustering	Number of clusters	Large *n_samples* and *n_clusters*	Many clusters, possibly connectivity constraints	Distances between points
DBSCAN	Neighborhood size	Very large *n_samples*, medium *n_clusters*	Non-flat geometry, uneven cluster sizes	Distances between nearest points

Affinity propagation:

- DON'T have to guess k.
- Tends to find uneven sized clusters.
- Good with many clusters.
- Required input: damping factor
- Euclidean distance typically used


MiniBatchKMeans
AffinityPropagation
MeanShift
SpectralClustering
Ward
DBSCAN
.01s
8.17s
.02s
.31s
.21s
.10s
0.1s
8.17s
.03s
.03s
.26s
.10s
.01s
8.45s
.03s
.04s
.31s
.11s
.02s
8.53s
.06s
.08s
.21s
.10s

Summary

Summary

- Clustering: unsupervised learning technique for grouping data

- Advanced clustering algorithms:

Hierarchical agglomerative clustering
DBSCAN
Mean shift
Spectral clustering
Affinity propagation

- All implemented in scikit-learn

https://scikit-learn.org/stable/modules/clustering.html

Questions?