A Hierarchical Algorithm for Extreme Clustering

Ari Kobren*, Nicholas Monath*
Akshay Krishnamurthy, Andrew McCallum

College of Information and Computer Science
University of Massachusetts Amherst

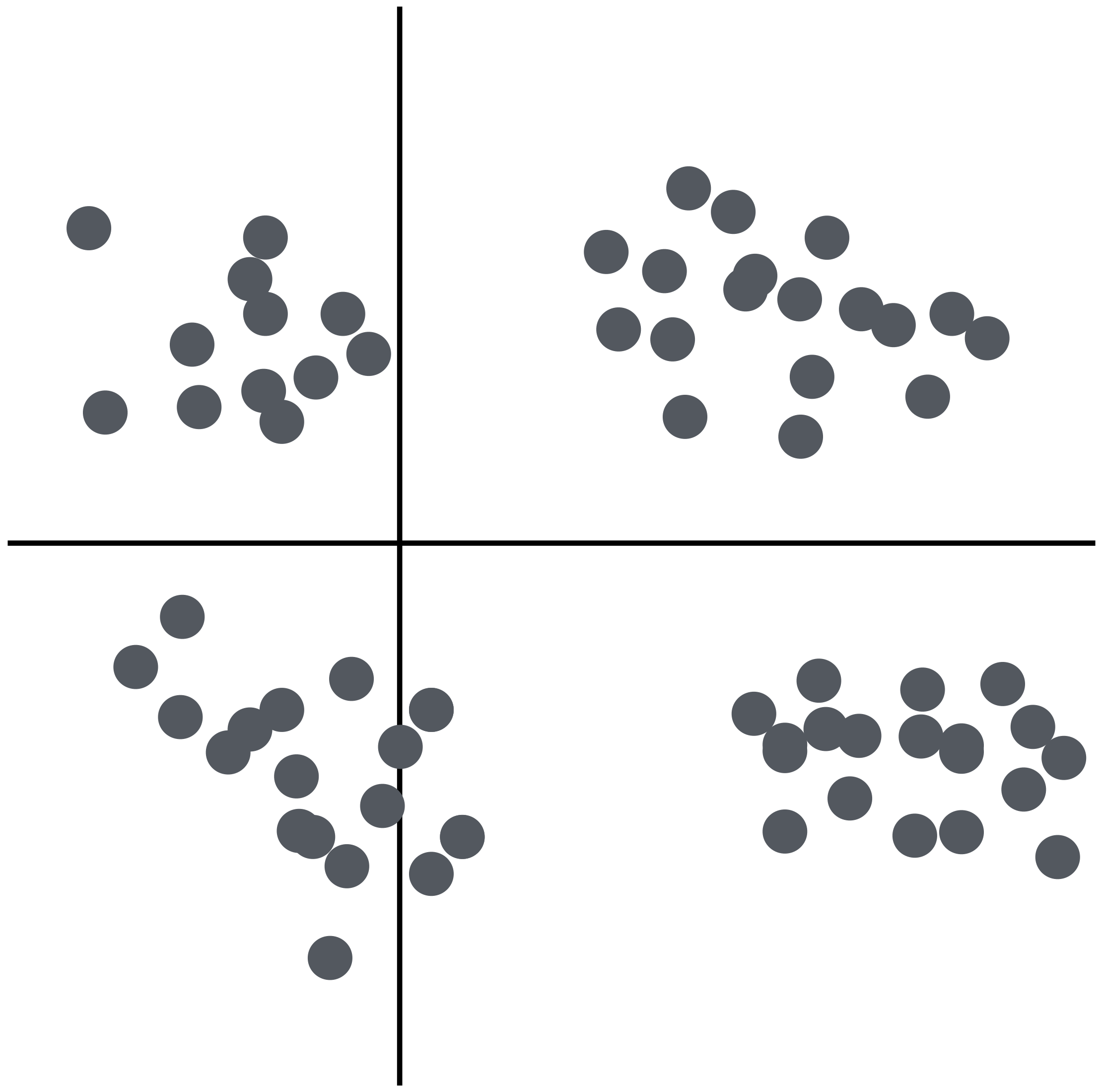
Clustering

Clustering

Partition dataset X into clusters $C_1 \ldots C_K$

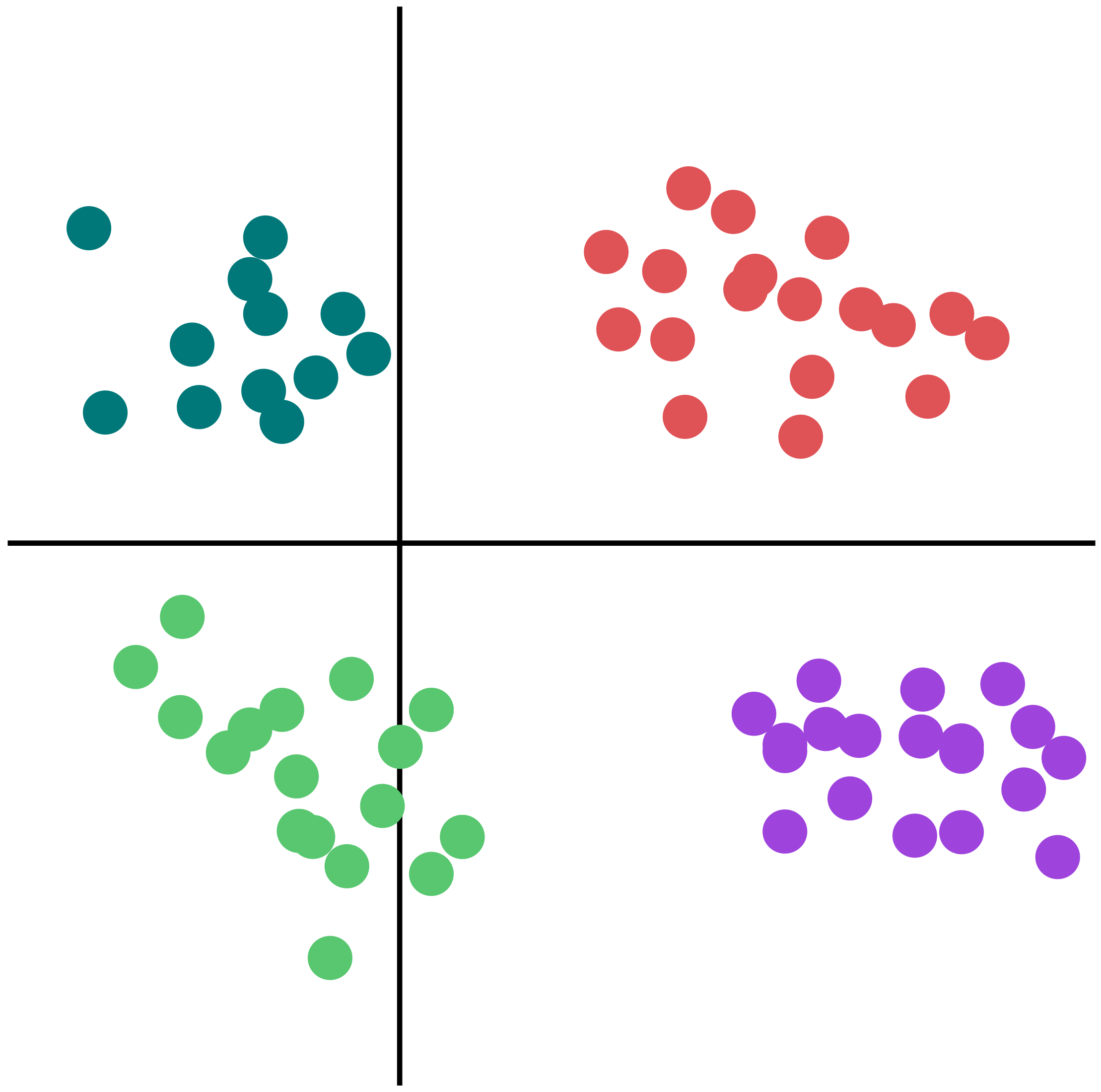
Clustering

Partition dataset X into clusters $C_1 \ldots C_K$



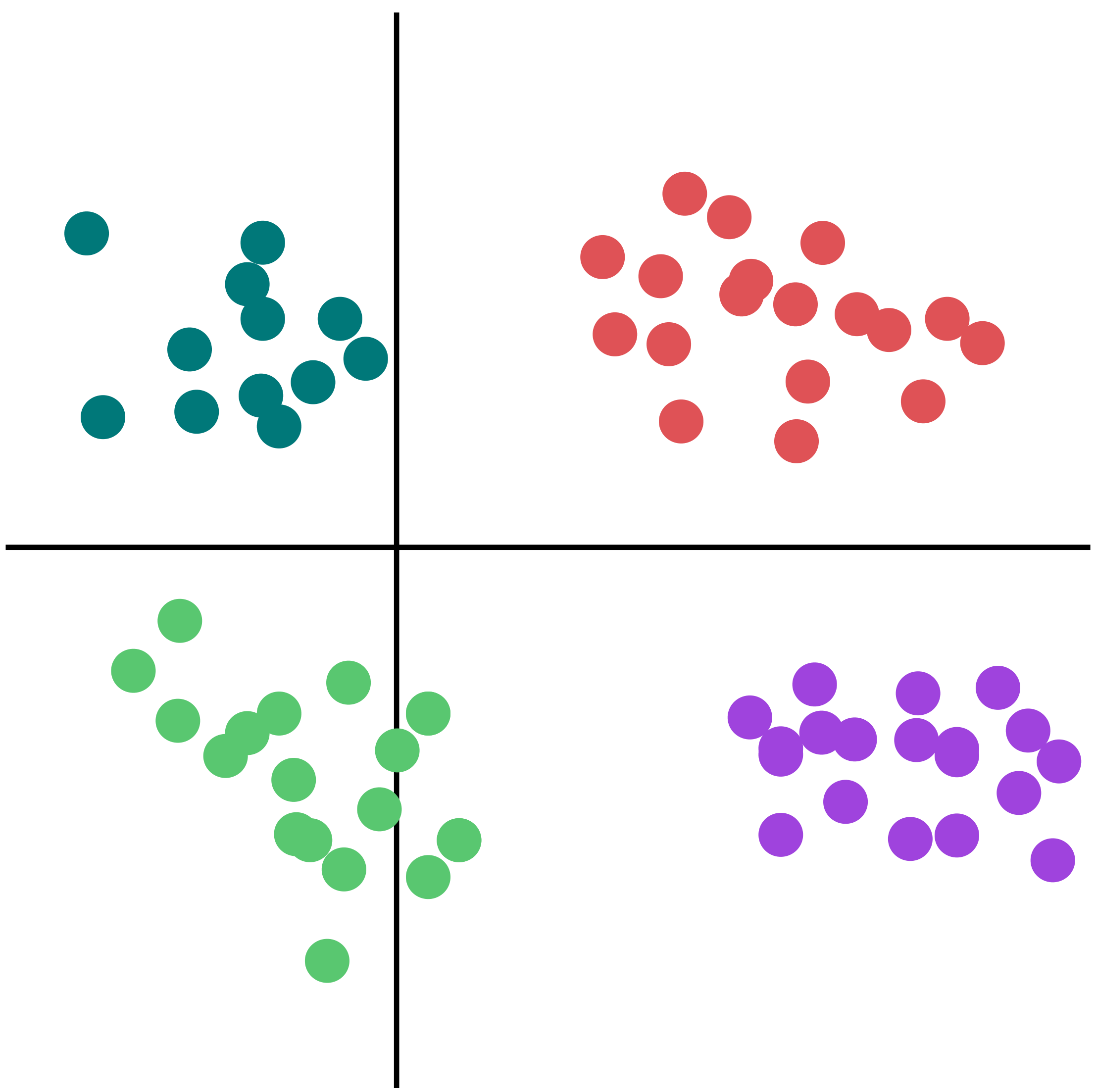
Clustering

Partition dataset X into clusters $C_1 \ldots C_K$

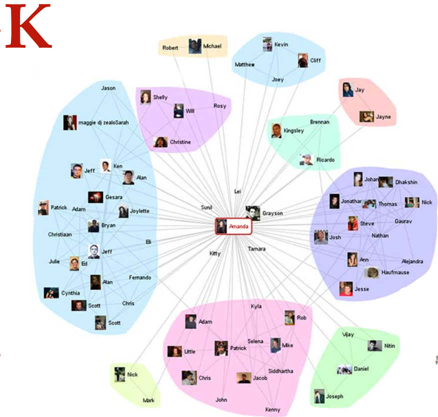


Clustering

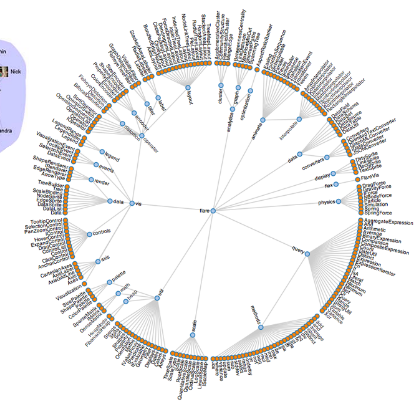
Partition dataset X into clusters $C_1 \dots C_K$



Analysis & Visualization



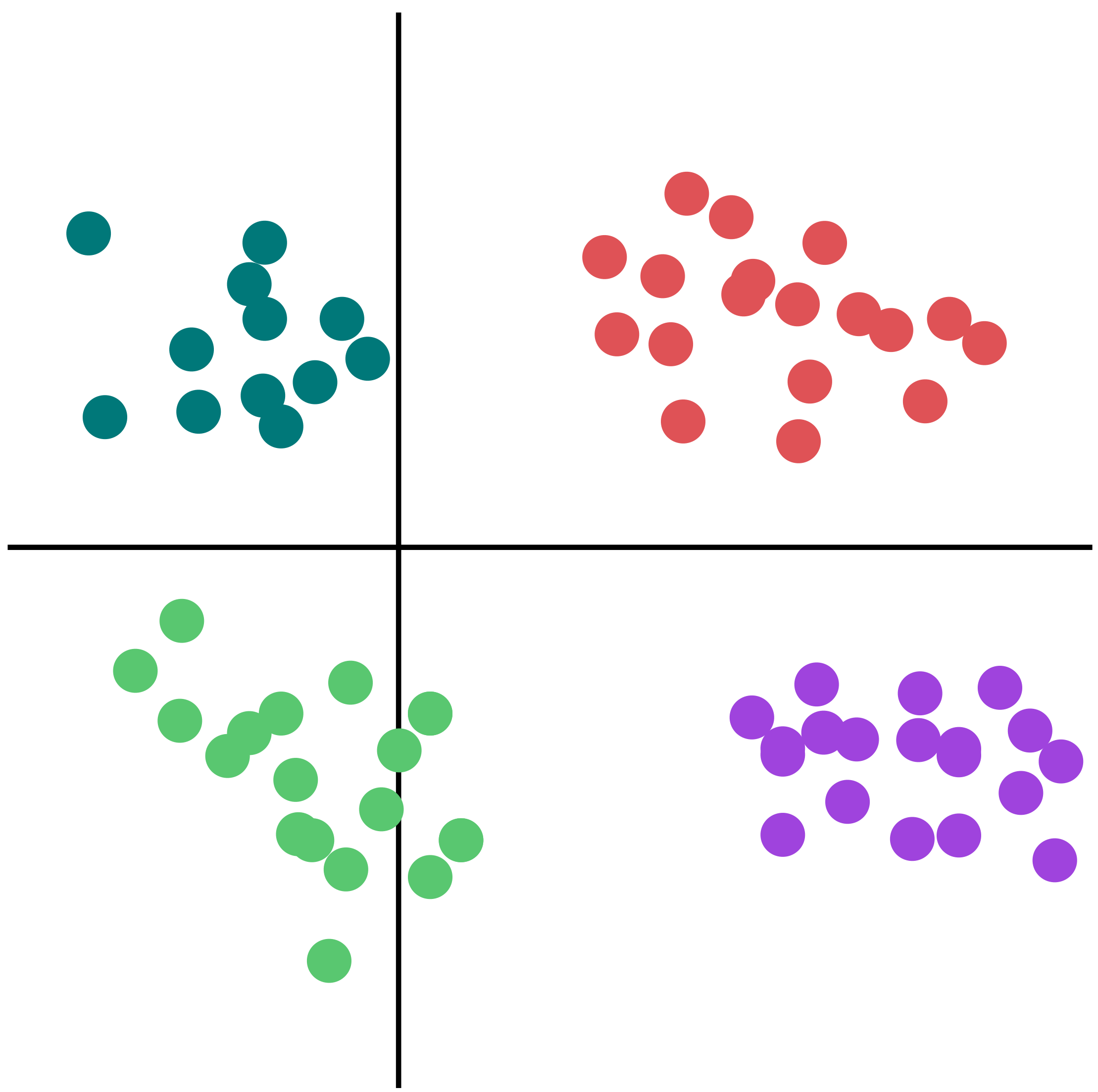
[Few, 2014]



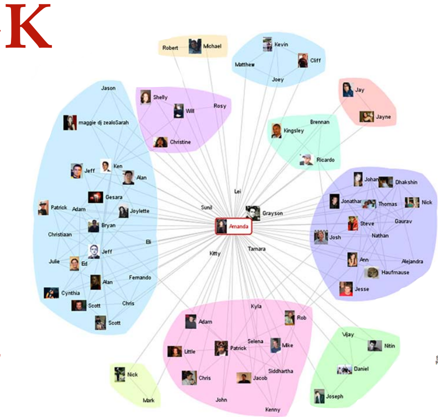
[Heer et al, 2014]

Clustering

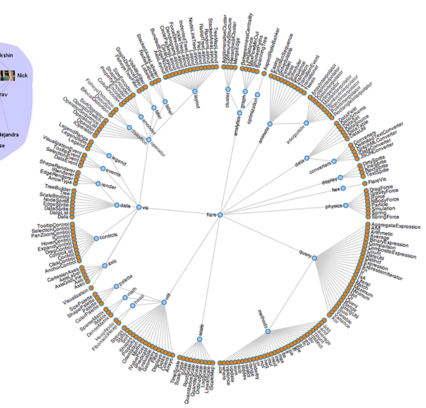
Partition dataset X into clusters $C_1 \ldots C_K$



Analysis & Visualization



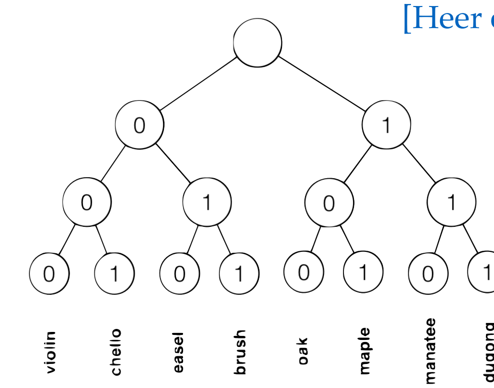
[Few, 2014]



[Heer et al, 2014]

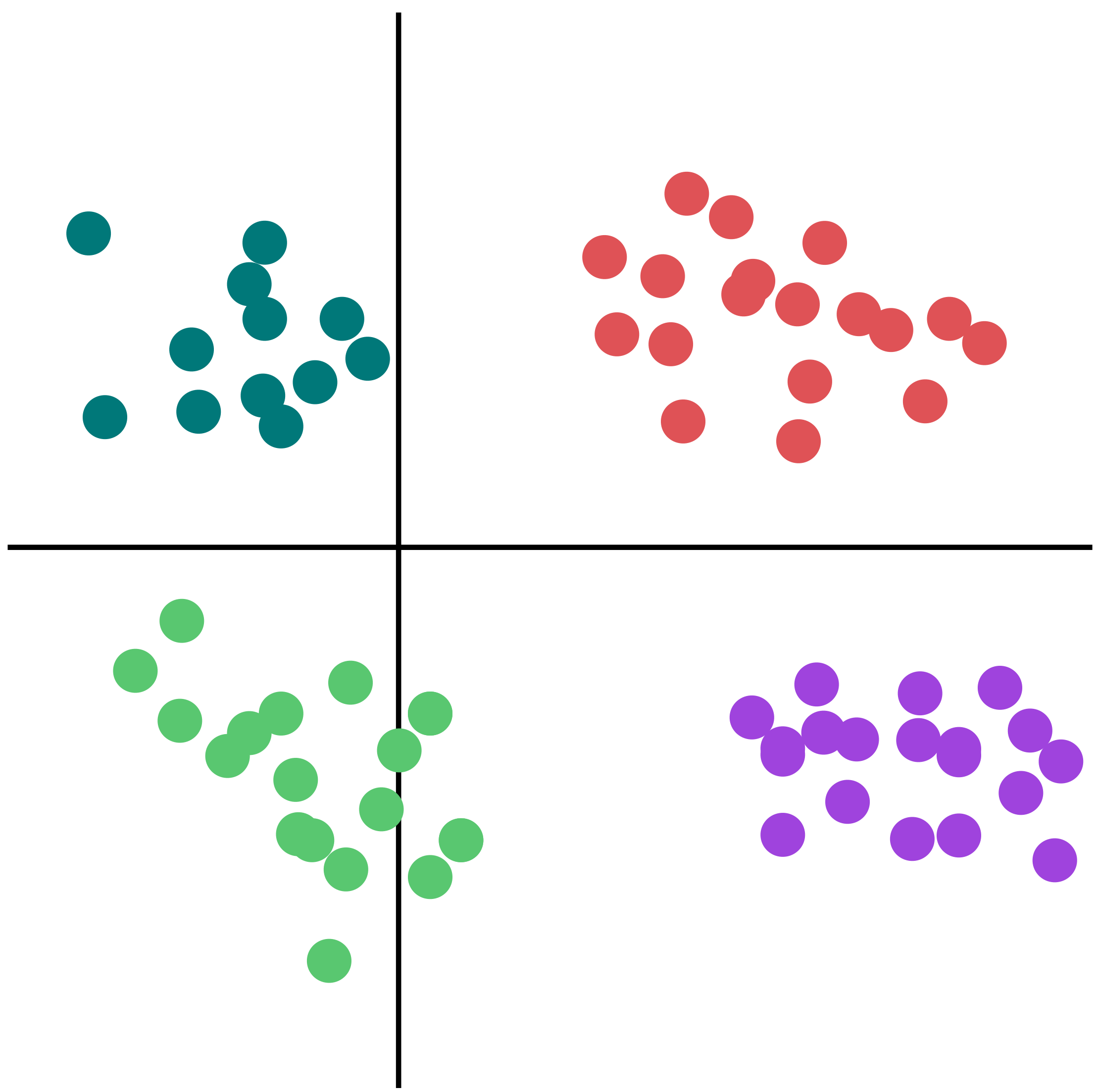
Feature Engineering

[Brown et al, 1993]

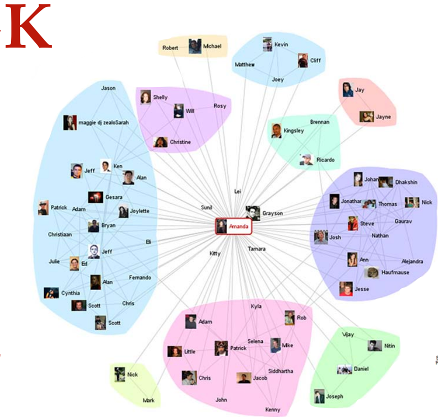


Clustering

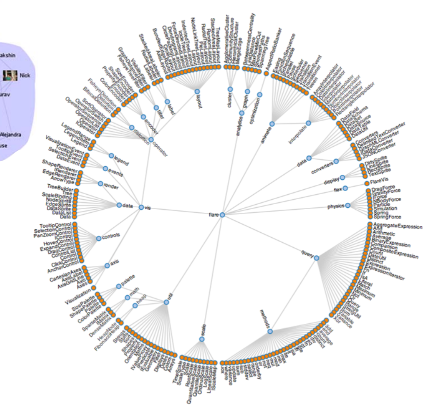
Partition dataset X into clusters $C_1 \ldots C_K$



Analysis & Visualization



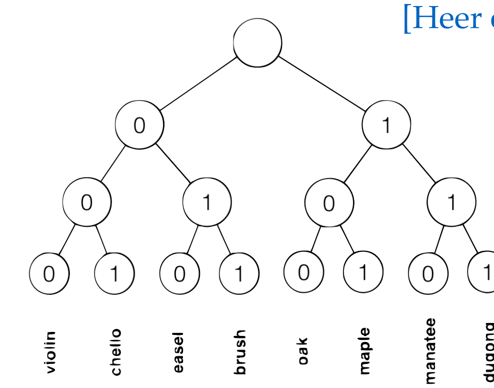
[Few, 2014]



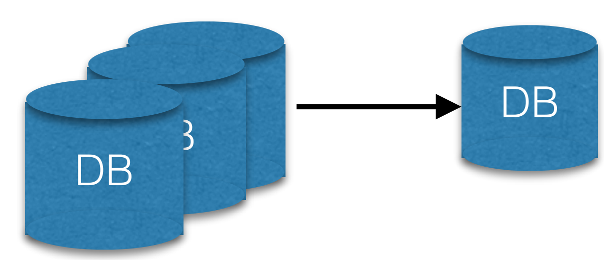
[Heer et al, 2014]

Feature Engineering

[Brown et al, 1993]

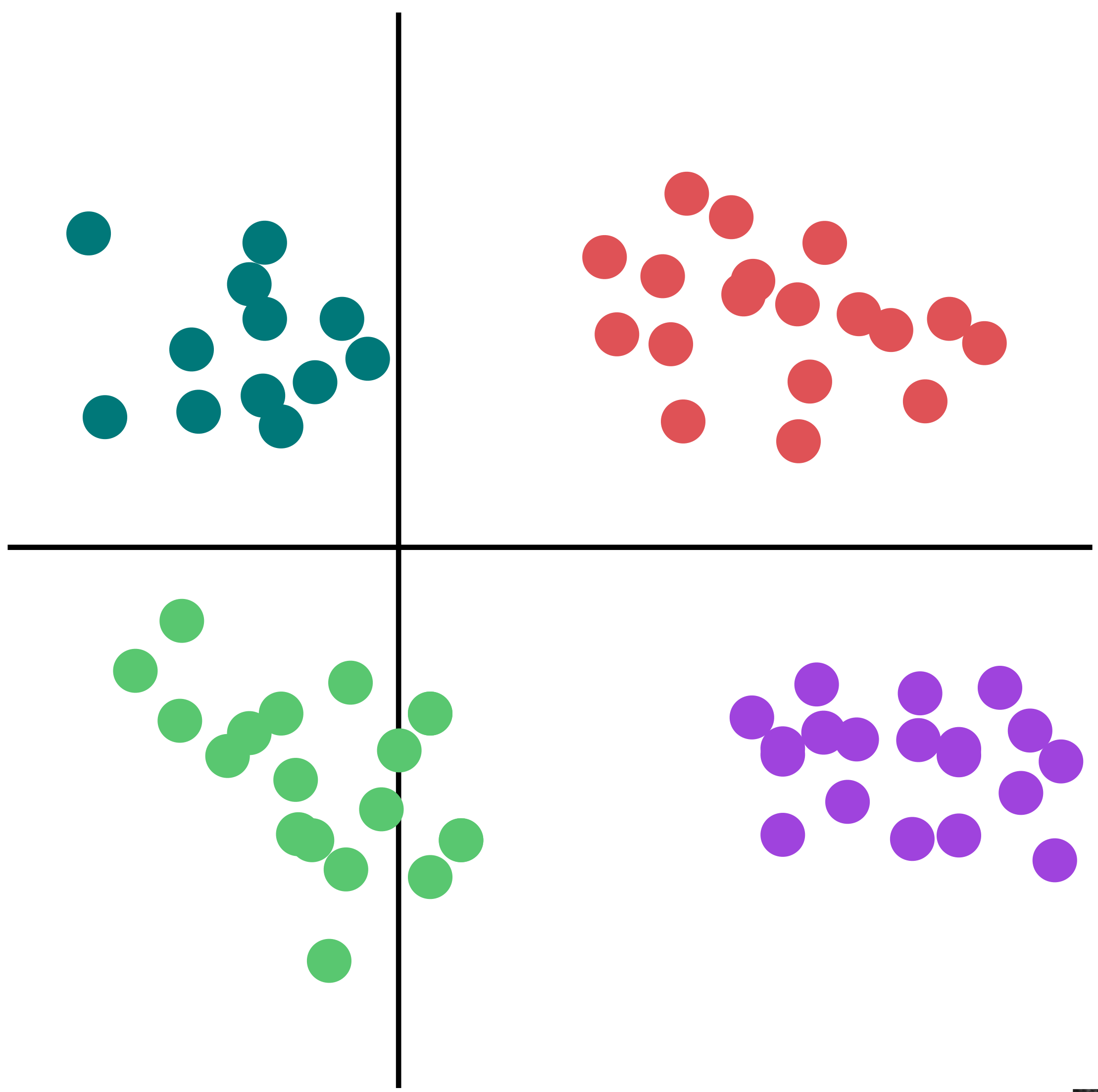


Deduplication

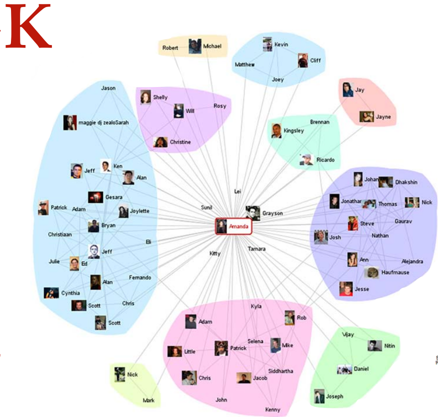


Clustering

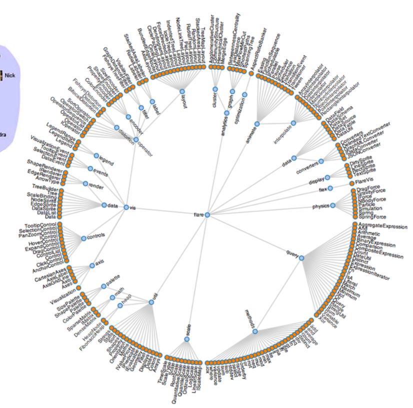
Partition dataset X into clusters $C_1 \ldots C_K$



Analysis & Visualization



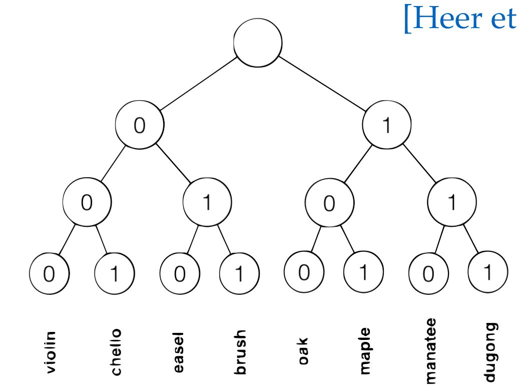
[Few, 2014]



[Heer et al, 2014]

Feature Engineering

[Brown et al, 1993]



Deduplication

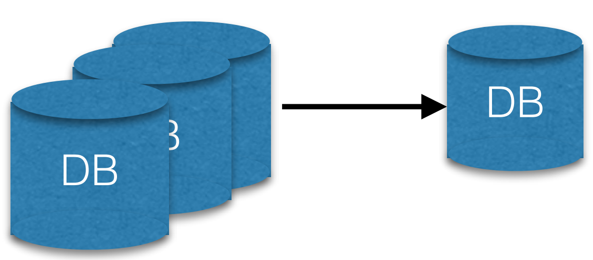
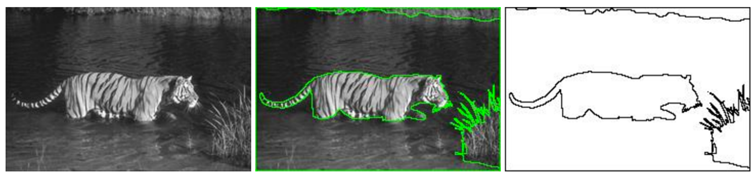


Image Segmentation



[Martin et al, 2001]

Extreme Clustering

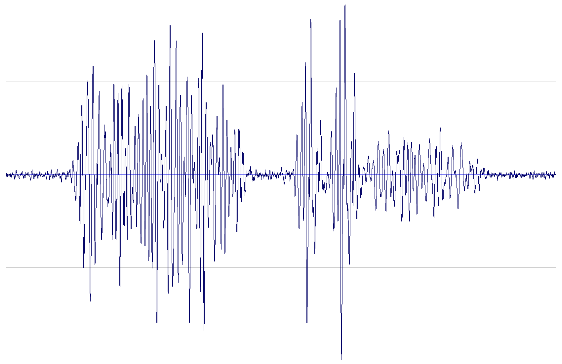
Extreme Clustering

Large Number of Clusters *K*
& Large Number of Points *N*

Extreme Clustering

Large Number of Clusters *K*
& Large Number of Points *N*

Speaker Recognition



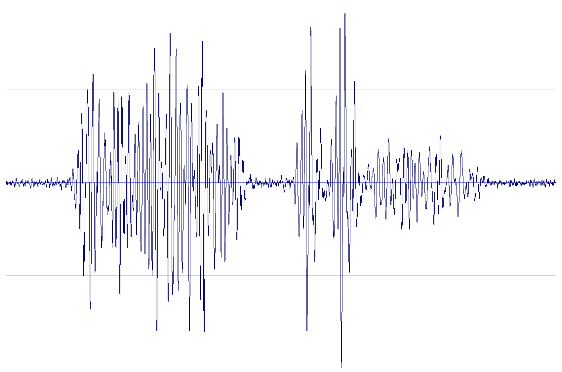
NIST I-VECTOR Challenge

N = 36,572 Samples
K = 4,958 Speakers

Extreme Clustering

Large Number of Clusters K
& Large Number of Points N

Speaker Recognition



NIST I-VECTOR Challenge

N = 36,572 Samples

K = 4,958 Speakers

Image Clustering

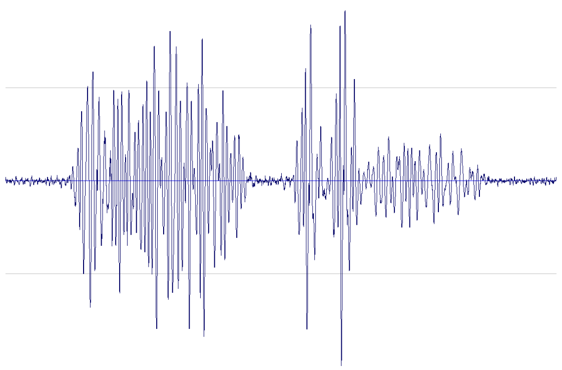
N =14 Million Images

K = 21,000+ Object Classes

Extreme Clustering

Large Number of Clusters K
& Large Number of Points N

Speaker Recognition



NIST I-VECTOR Challenge

N = 36,572 Samples

K = 4,958 Speakers

Entity Resolution

A. Banerjee, S. Chassang, E. Snowberg. *Decision Theoretic Approaches to Experiment Design and External Validity.* Handbook of Field Experiments. 2016.

Arindam Banerjee, S. Merugu, I. S. Dhillon, J. Ghosh. *Clustering with Bregman Divergences.* JMLR. 2006.

A. Banerjee, I. S. Dhillon, J. Ghosh, S. Sra. *Clustering on the Unit Hypersphere using von Mises-Fisher Distributions.* Journal of Machine Learning Research. 2005

Author Coreference. N=10M Records, K=1M Authors

Image Clustering

IMAGENET

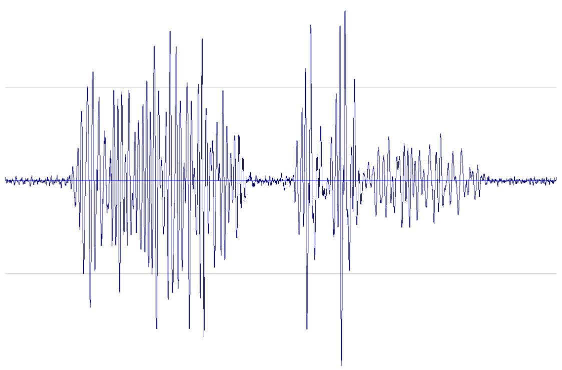
N =14 Million Images

K = 21,000+ Object Classes

Extreme Clustering

Large Number of Clusters K
& Large Number of Points N

Speaker Recognition



NIST I-VECTOR Challenge

N = 36,572 Samples

K = 4,958 Speakers

Image Clustering

IMAGENET

N =14 Million Images

K = 21,000+ Object Classes

Entity Resolution

A. Banerjee, S. Chassang, E. Snowberg. *Decision Theoretic Approaches to Experiment Design and External Validity.* Handbook of Field Experiments. 2016.

Arindam Banerjee, S. Merugu, I. S. Dhillon, J. Ghosh. *Clustering with Bregman Divergences.* JMLR. 2006.

A. Banerjee, I. S. Dhillon, J. Ghosh, S. Sra. *Clustering on the Unit Hypersphere using von Mises-Fisher Distributions.* Journal of Machine Learning Research. 2005

Author Coreference. N=10M Records, K=1M Authors

Extreme Clustering

Large Number of Clusters *K*
& Large Number of Points *N*

Extreme Clustering

Large Number of Clusters *K* & Large Number of Points *N*

```
def kmeans(x1...xN,K):
 until convergence
  for x in x1...xN:
   for c in clusters:
    if ||c - x|| < min_c:
     min_dist = ||c - x||
     min_c = x
   assign(x,min_c)
  update(clusters)
```

Extreme Clustering

Large Number of Clusters *K*
& Large Number of Points *N*

```
def kmeans(x1...xN,K):
 until convergence
  for x in x1...xN:
   for c in clusters:
    if ||c - x|| < min_c:
     min_dist = ||c - x||
     min_c = x
   assign(x,min_c)
  update(clusters)
```

Running time of k-means

Extreme Clustering

Large Number of Clusters *K*
& Large Number of Points *N*

```
def kmeans(x1...xN,K):
 until convergence
  for x in x1...xN:
   for c in clusters:
    if ||c - x|| < min_c:
     min_dist = ||c - x||
     min_c = x
   assign(x,min_c)
  update(clusters)
```

Running time of k-means

Extreme Clustering

Large Number of Clusters *K*
& Large Number of Points *N*

```
def kmeans(x1...xN,K):
 until convergence
  for x in x1...xN:
   for c in clusters:
    if ||c - x|| < min_c:
     min_dist = ||c - x||
     min_c = x
   assign(x,min_c)
  update(clusters)
```

Linear in K, O(NK)

Extreme Clustering

Large Number of Clusters *K*
& Large Number of Points *N*

```
def kmeans(x1...xN,K):
 until convergence
  for x in x1...xN:
   for c in clusters:
    if ||c - x|| < min_c:
     min_dist = ||c - x||
     min_c = x
   assign(x,min_c)
  update(clusters)
```

For large K, we'd like to be sublinear.

Existing Approaches

Existing Approaches

	Scales in N	Scales in K	Non-Greedy	In Practice

Existing Approaches

	Scales in N	Scales in K	Non-Greedy	In Practice
BIRCH [Zhang et al, 1998]	✓	✓	—	Fast, but low accuracy

Existing Approaches

	Scales in N	Scales in K	Non-Greedy	In Practice
BIRCH [Zhang et al, 1998]	✓	✓	—	Fast, but low accuracy
SGD/MiniBatch K-Means [Bottou and Bengio, 1995] [Sculley, 2010]	✓	✗	✓	Very effective, but does not scale with K

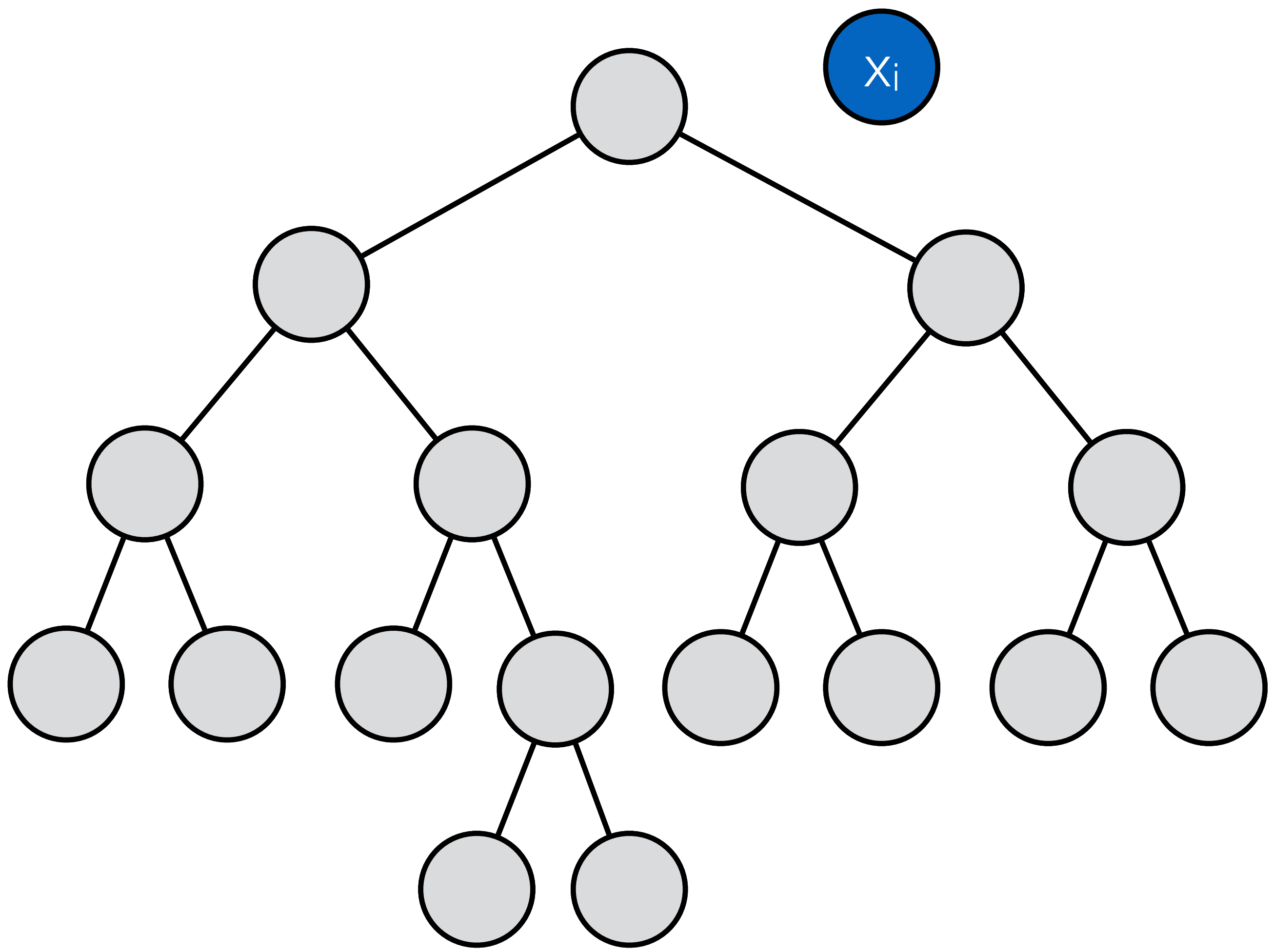
Existing Approaches

	Scales in N	Scales in K	Non-Greedy	In Practice
BIRCH [Zhang et al, 1998]	✓	✓	—	Fast, but low accuracy
SGD/MiniBatch K-Means [Bottou and Bengio, 1995] [Sculley, 2010]	✓	✗	✓	Very effective, but does not scale with K
StreamKM++ BICO [Ackermann et al, 2012] [Fichtenberger, et al, 2013]	✓	✗	—	Number of coresets does not scale with K

Hierarchical Clustering

Advantages for Online Extreme Clustering

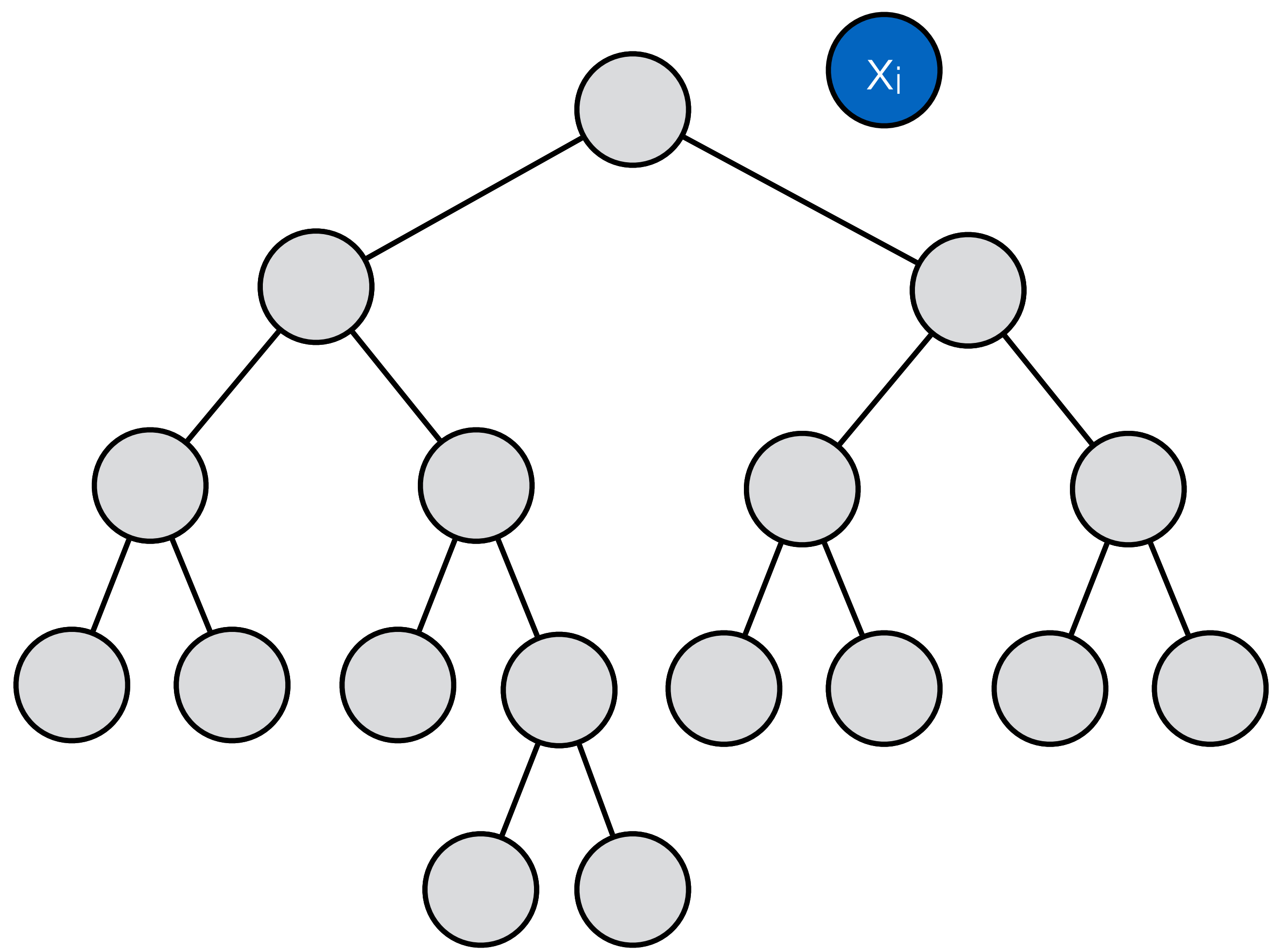
Efficiency



Hierarchical Clustering

Advantages for Online Extreme Clustering

Efficiency



Top-Down Log-Time Search

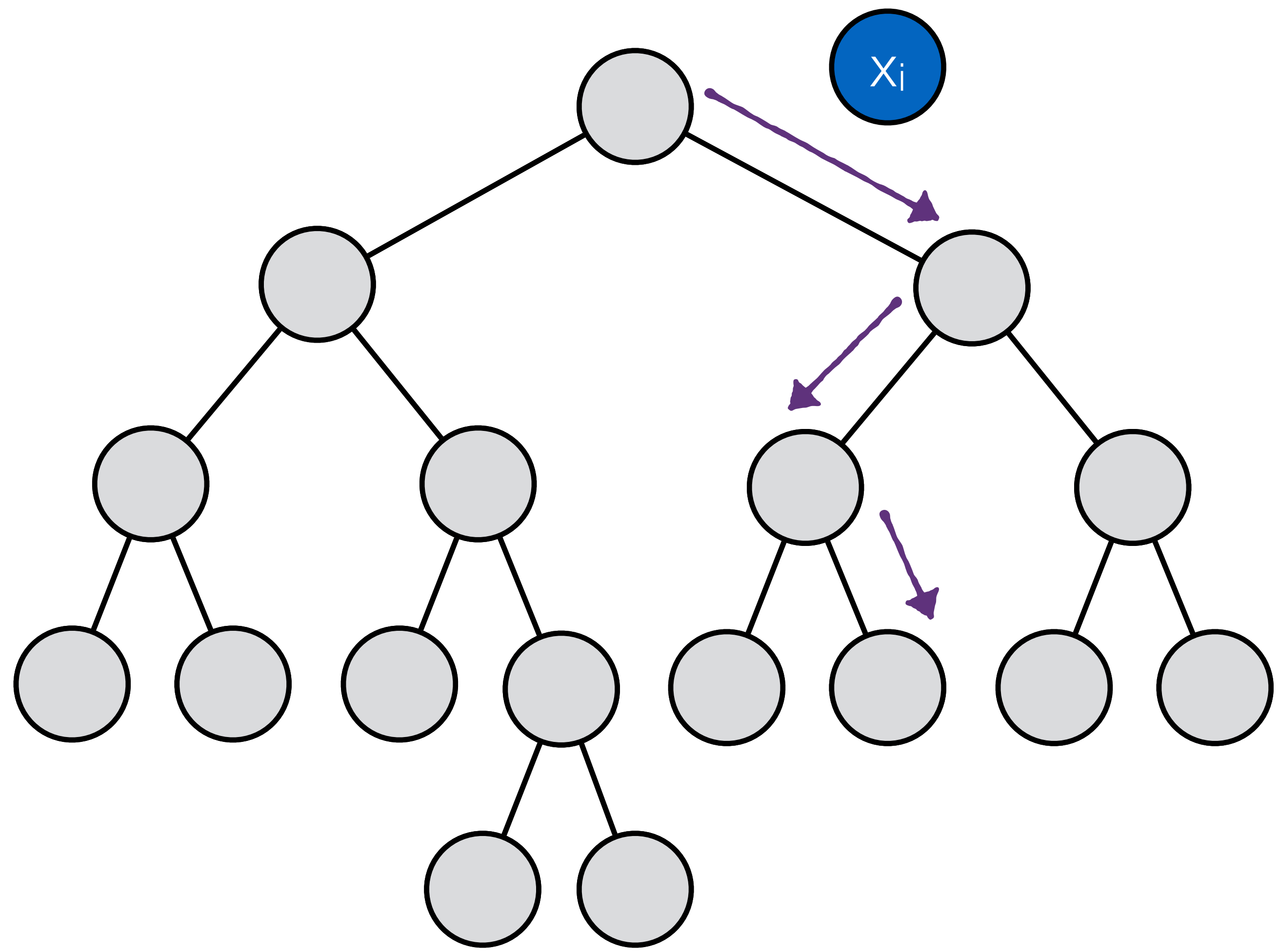
Extreme Multiclass Classification:

[Choromanska et al, 2015]
[Daumé III et al, 2016]

Hierarchical Clustering

Advantages for Online Extreme Clustering

Efficiency



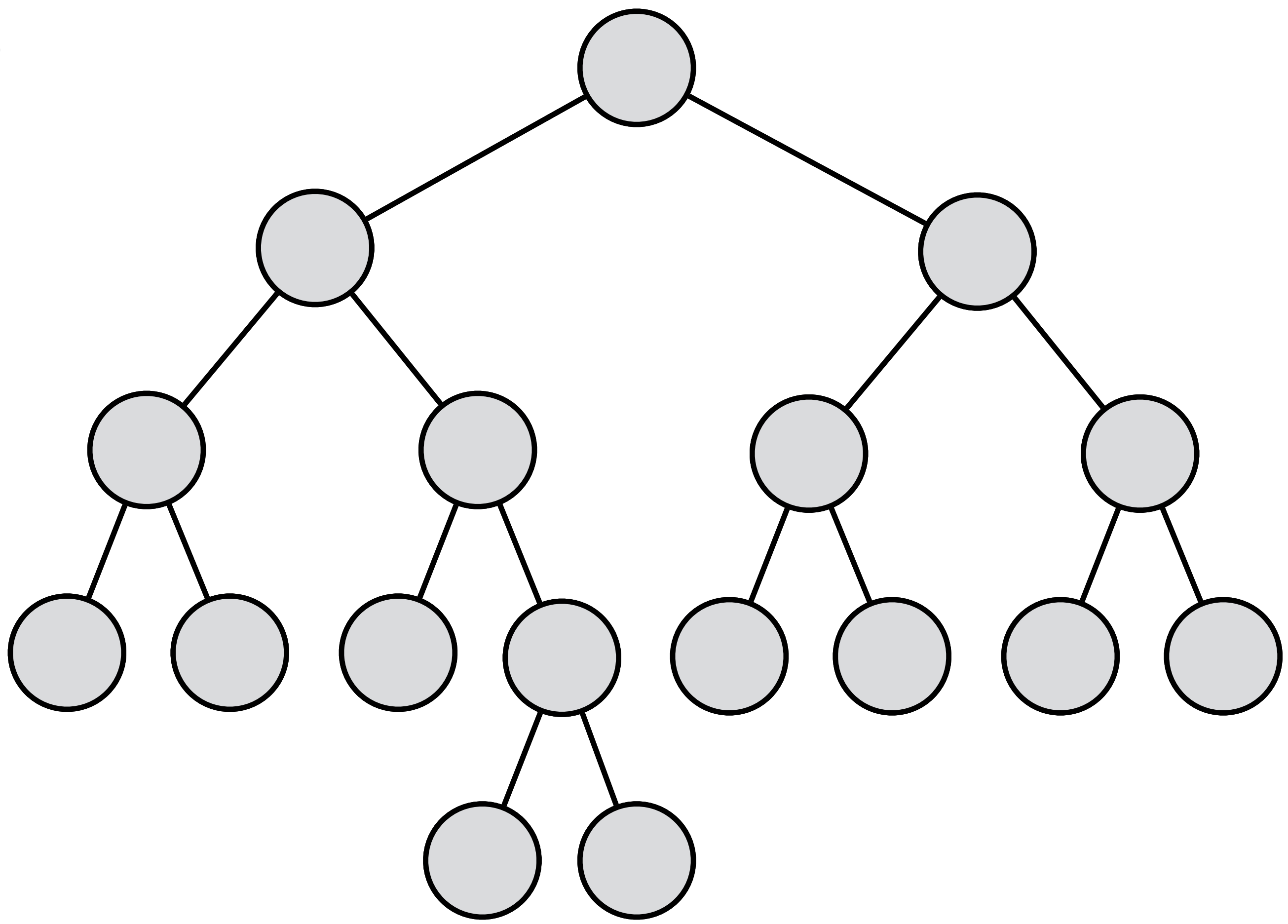
Top-Down Log-Time Search

Extreme Multiclass
Classification:
[Choromanska et al, 2015]
[Daumé III et al, 2016]

Hierarchical Clustering

Advantages for Online Extreme Clustering

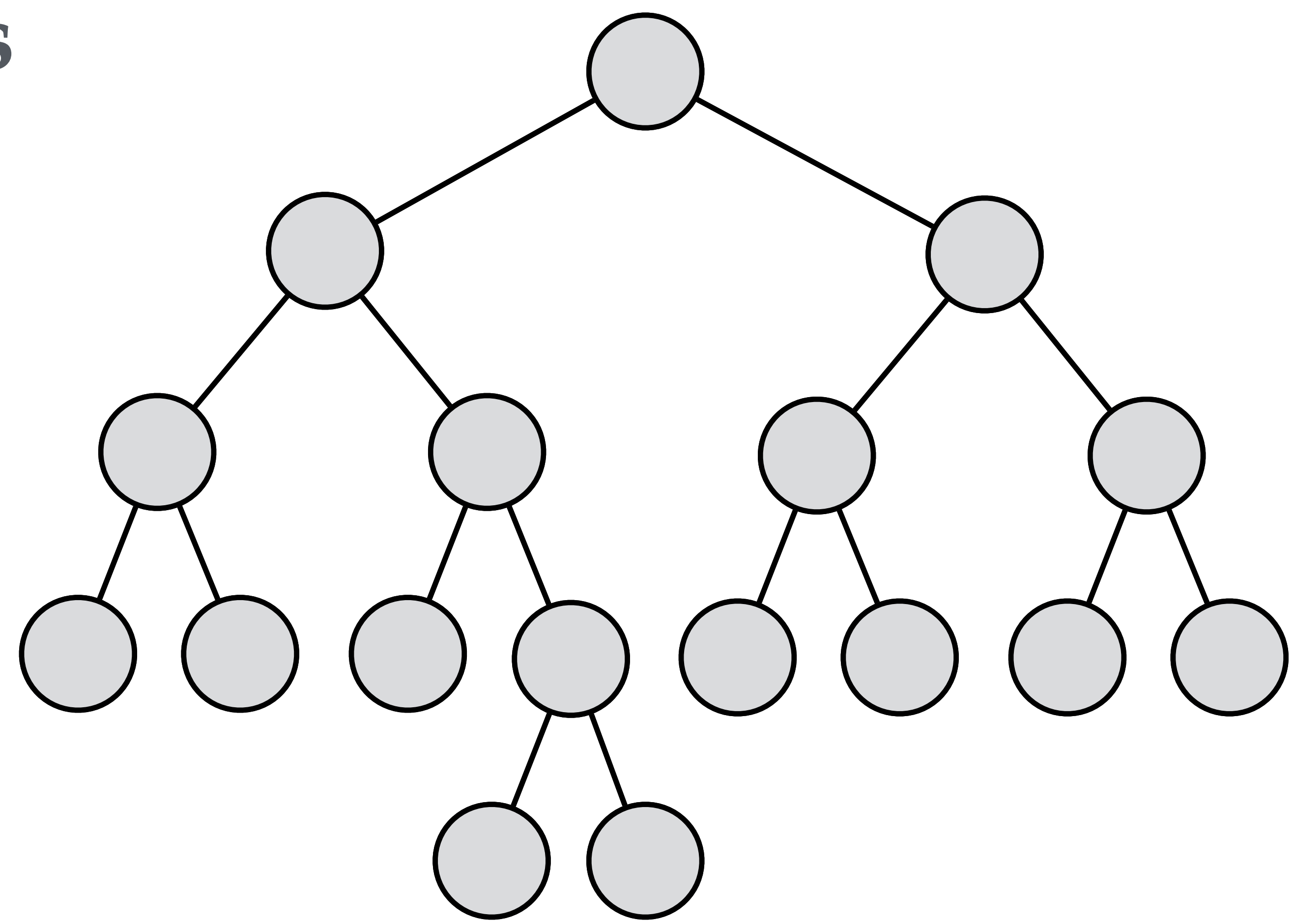
Non-greediness



Hierarchical Clustering

Advantages for Online Extreme Clustering

Non-greediness

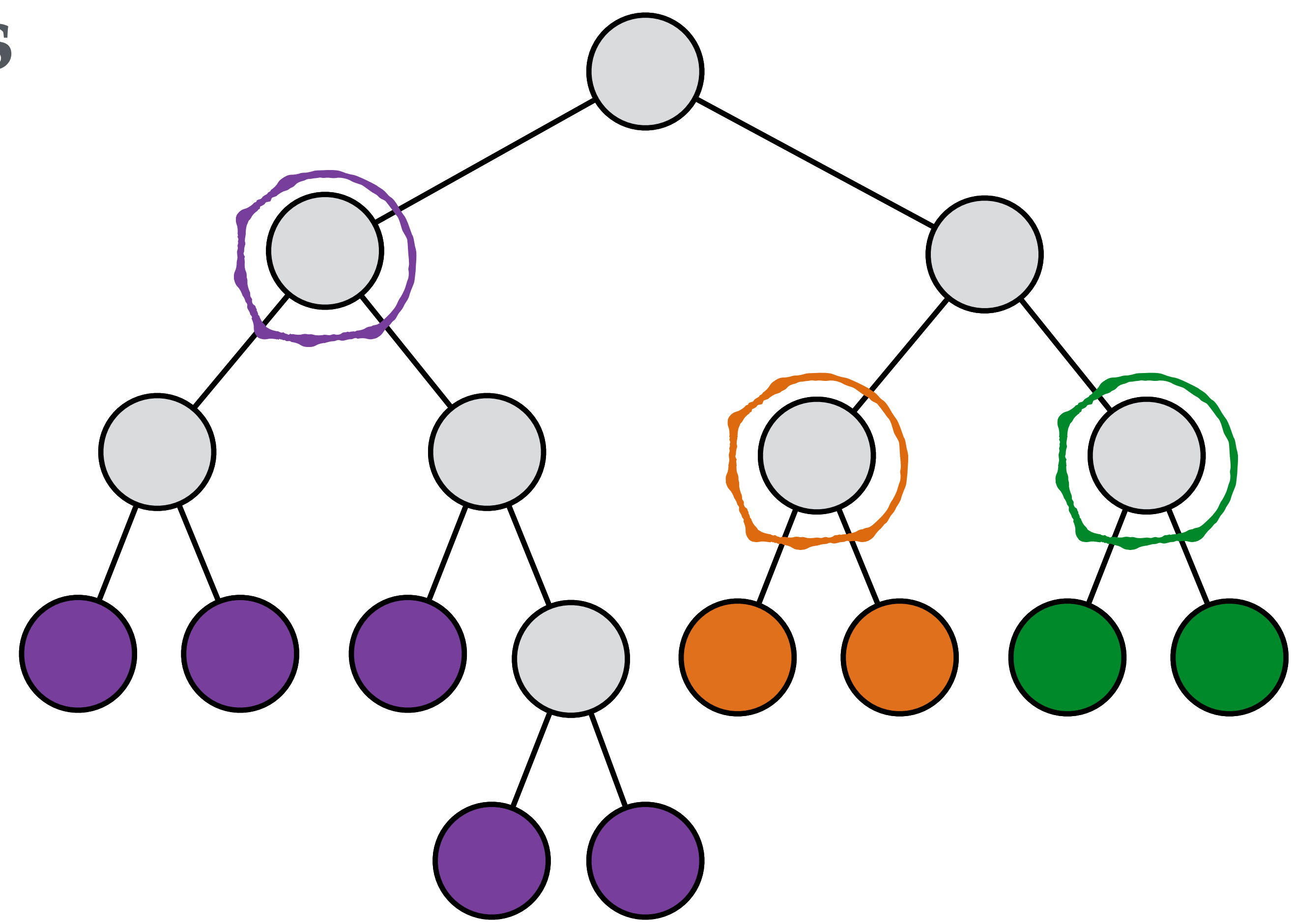


Simultaneously Represent
Multiple Alternative Clusterings

Hierarchical Clustering

Advantages for Online Extreme Clustering

Non-greediness

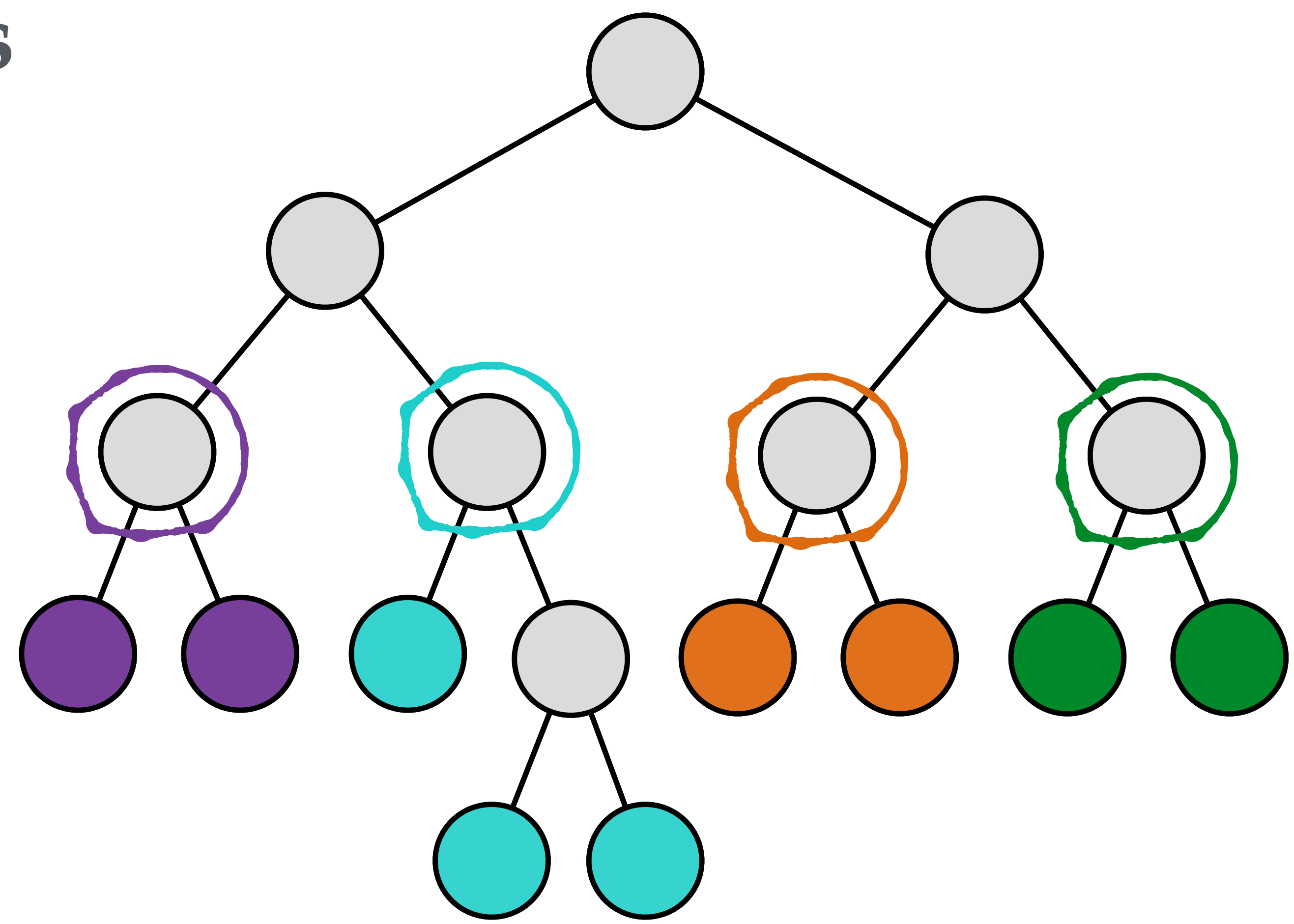


Simultaneously Represent
Multiple Alternative Clusterings

Hierarchical Clustering

Advantages for Online Extreme Clustering

Non-greediness

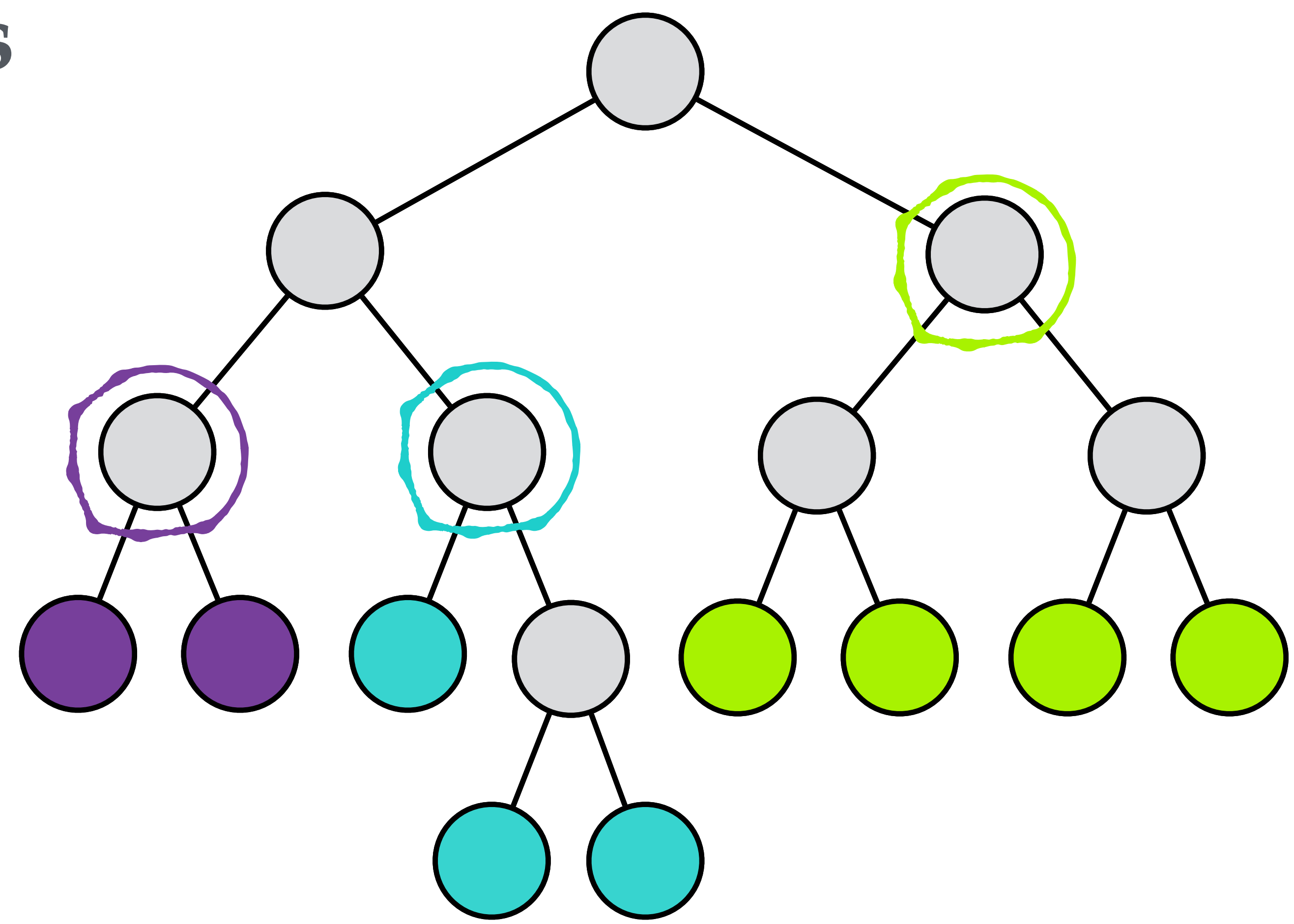


Simultaneously Represent
Multiple Alternative Clusterings

Hierarchical Clustering

Advantages for Online Extreme Clustering

Non-greediness



Simultaneously Represent
Multiple Alternative Clusterings

PERCH

Purity Enhancing Rotations for Cluster Hierarchies

PERCH

Purity Enhancing Rotations for Cluster Hierarchies

Incrementally build hierarchical clustering

PERCH

Purity Enhancing Rotations for Cluster Hierarchies

Incrementally build hierarchical clustering

Route point to nearest neighbor

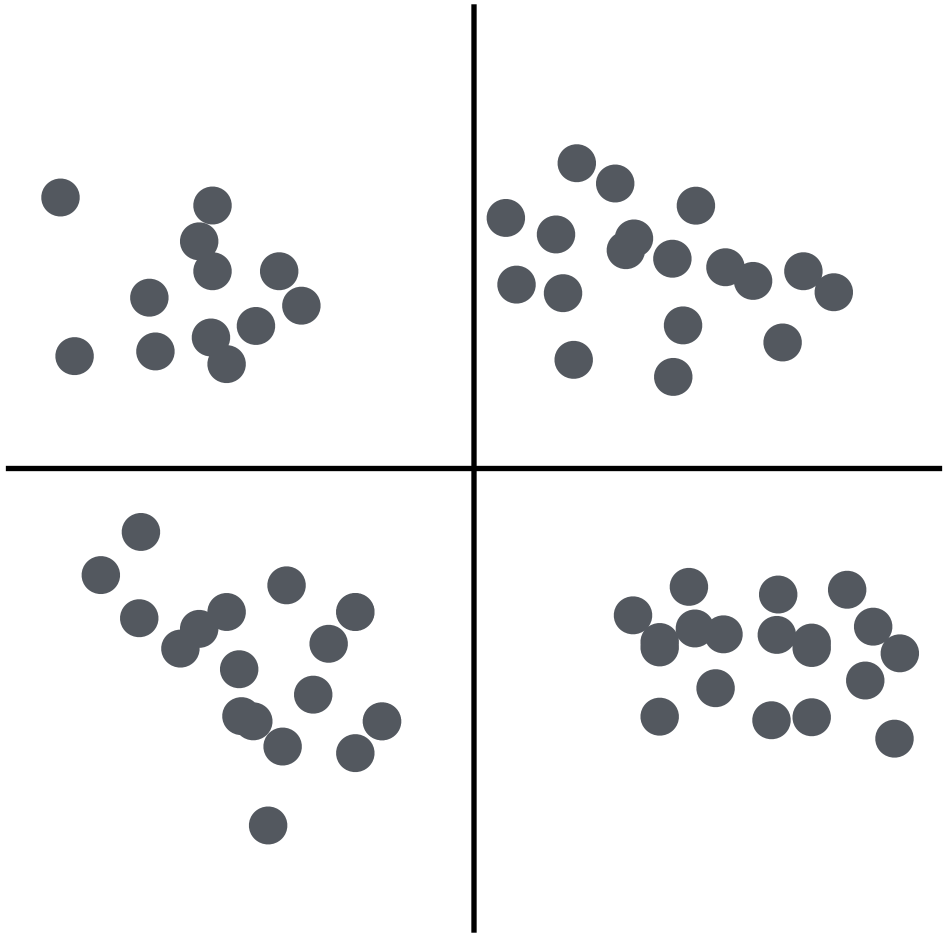
PERCH

Purity Enhancing Rotations for Cluster Hierarchies

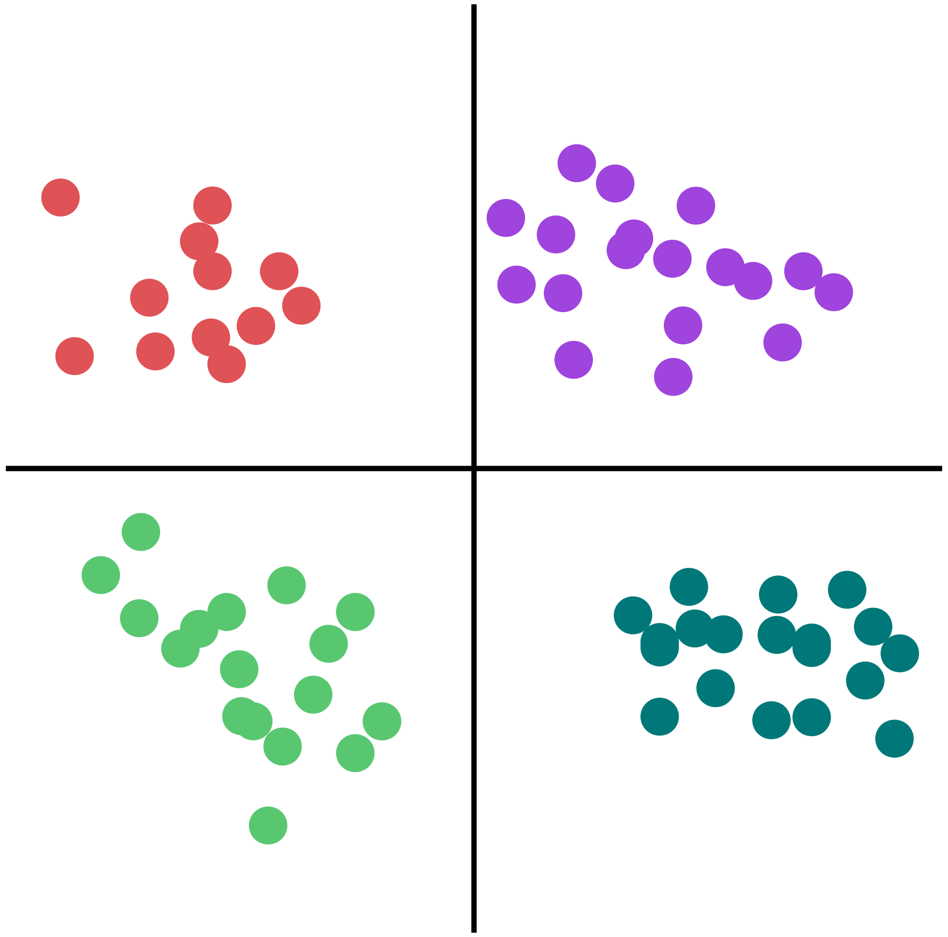
Incrementally build hierarchical clustering

Route point to nearest neighbor

Tree maintenance using rotation operations



Dataset

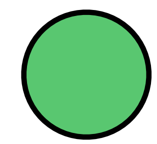


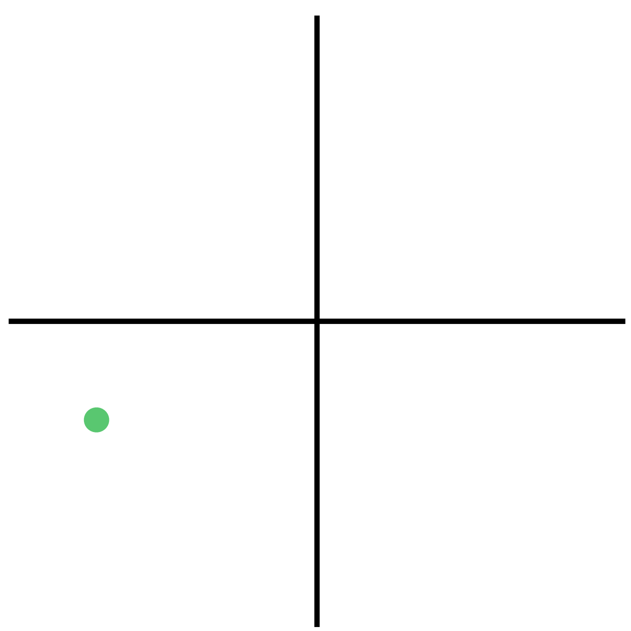
True Clustering

(labels withheld from clustering algorithm)

PERCH

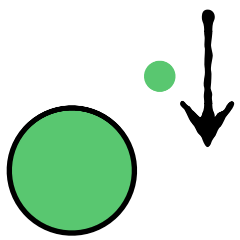
```
def perch(x_1…x_N,T):
 for x_i in x_1…x_N:
  n = nearestNeigh(x_i,T)
  v = split(n)
  recursiveRotate(v,T)
```

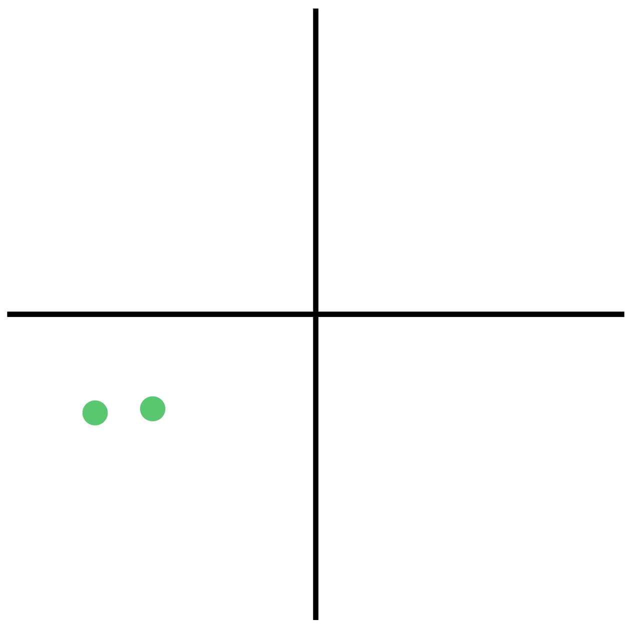




PERCH

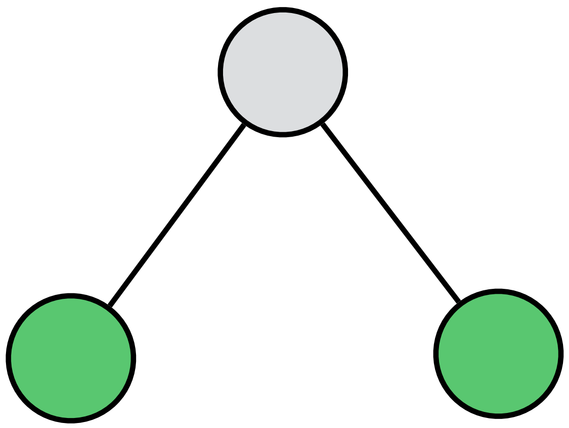
```
def perch(x1…xN,T):
 for xi in x1…xN:
  n = nearestNeigh(xi,T)
  v = split(n)
  recursiveRotate(v,T)
```

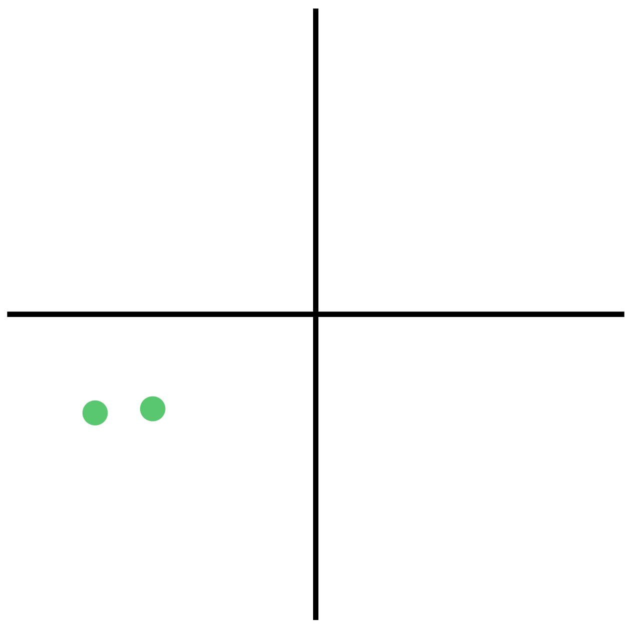




PERCH

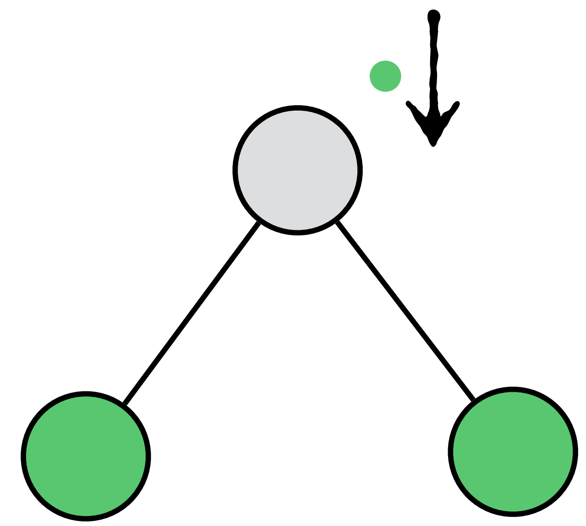
```
def perch(x1…xN,T):
 for xi in x1…xN:
  n = nearestNeigh(xi,T)
  v = split(n)
  recursiveRotate(v,T)
```

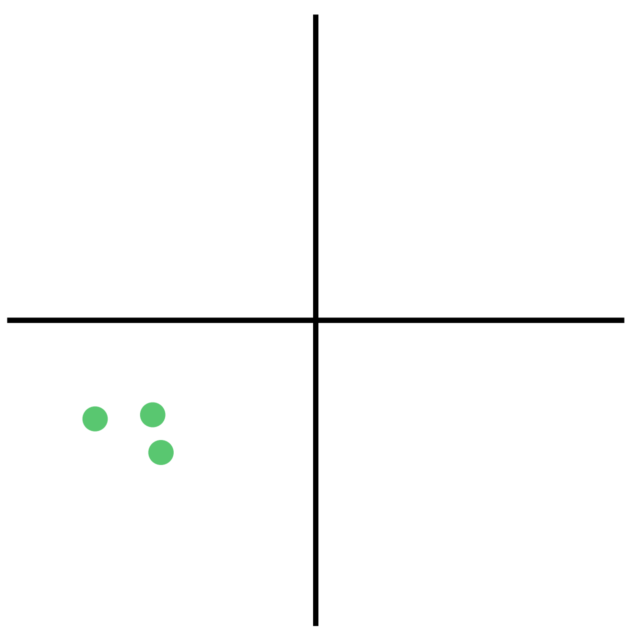




PERCH

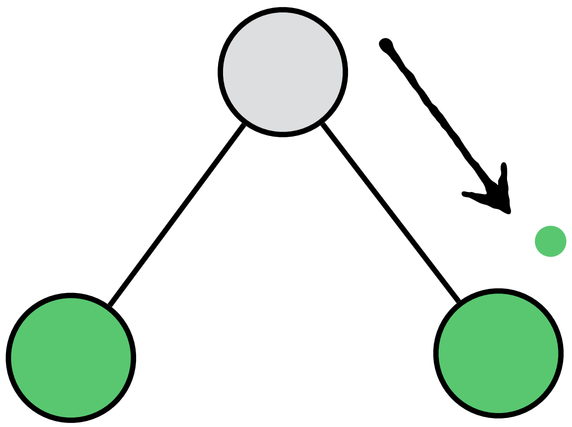
```
def perch(x1…xN,T):
 for xi in x1…xN:
  n = nearestNeigh(xi,T)
  v = split(n)
  recursiveRotate(v,T)
```

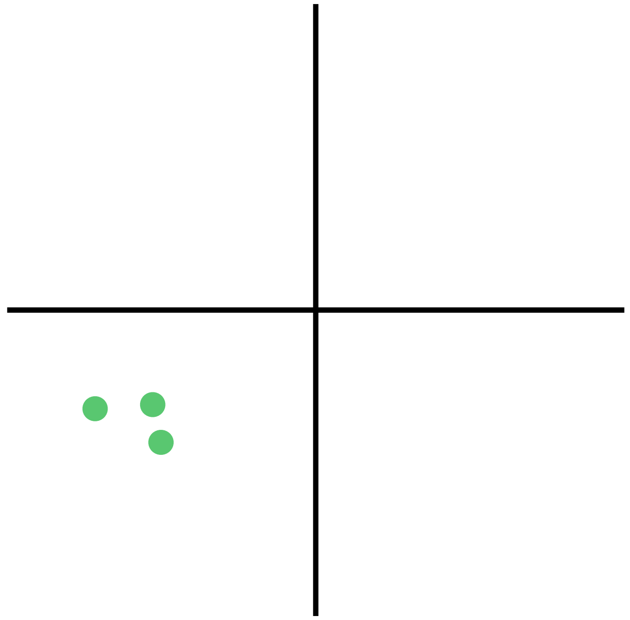




PERCH

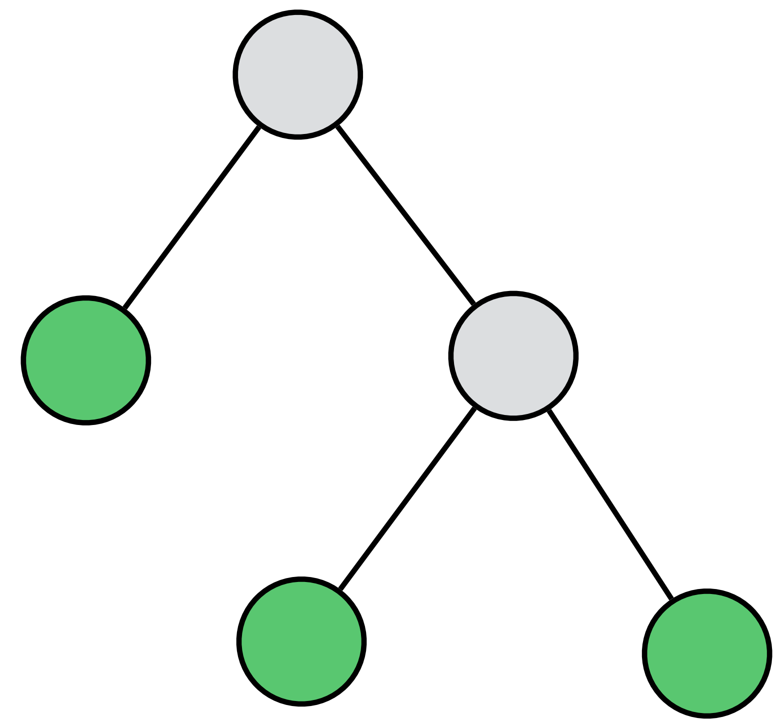
```
def perch(x_1…x_N,T):
 for x_i in x_1…x_N:
  n = nearestNeigh(x_i,T)
  v = split(n)
  recursiveRotate(v,T)
```

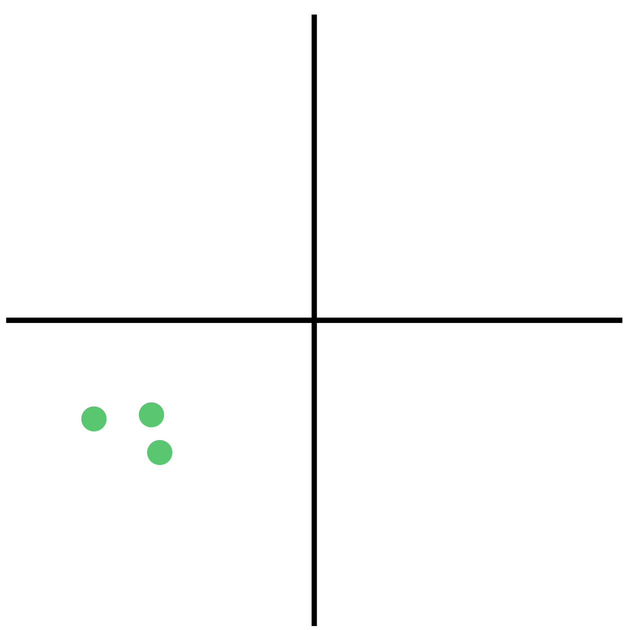




Perch

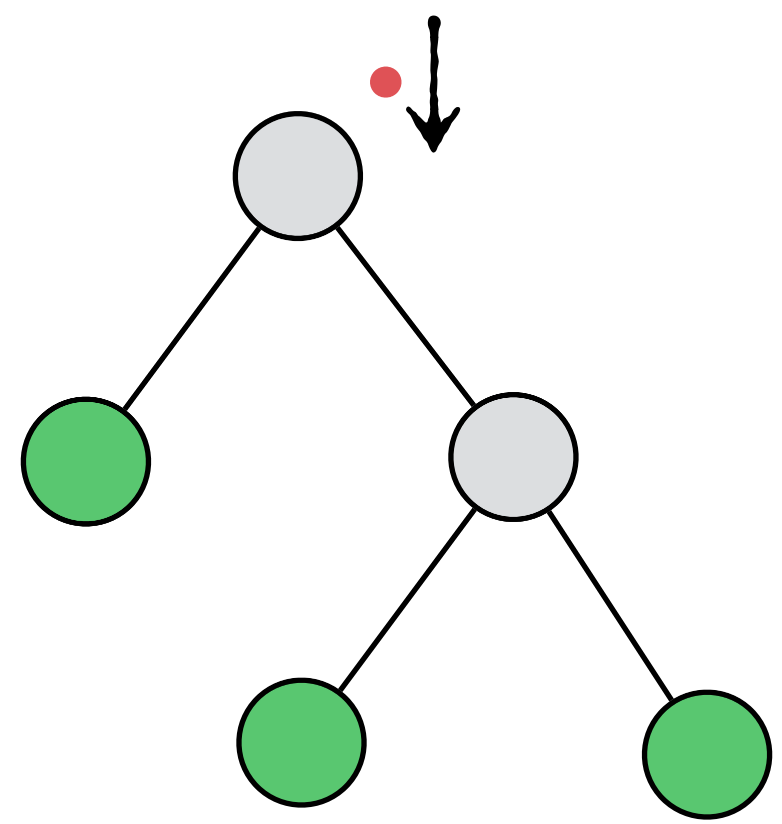
```
def perch(x1…xN,T):
 for xi in x1…xN:
  n = nearestNeigh(xi,T)
  v = split(n)
  recursiveRotate(v,T)
```

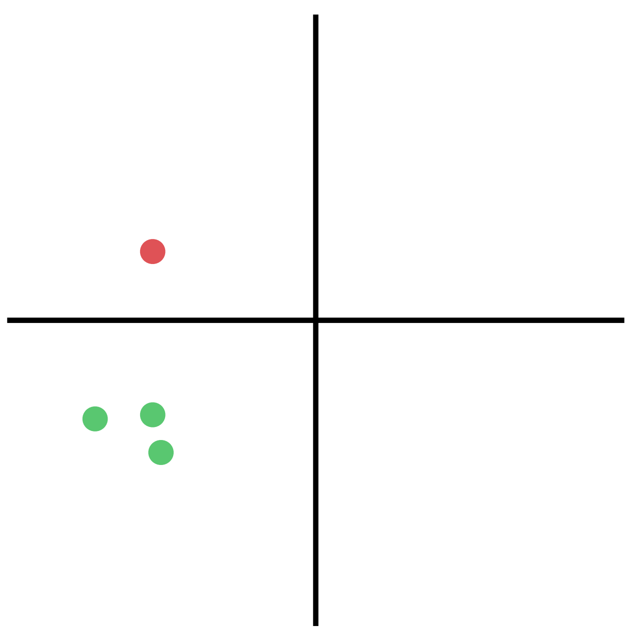




PERCH

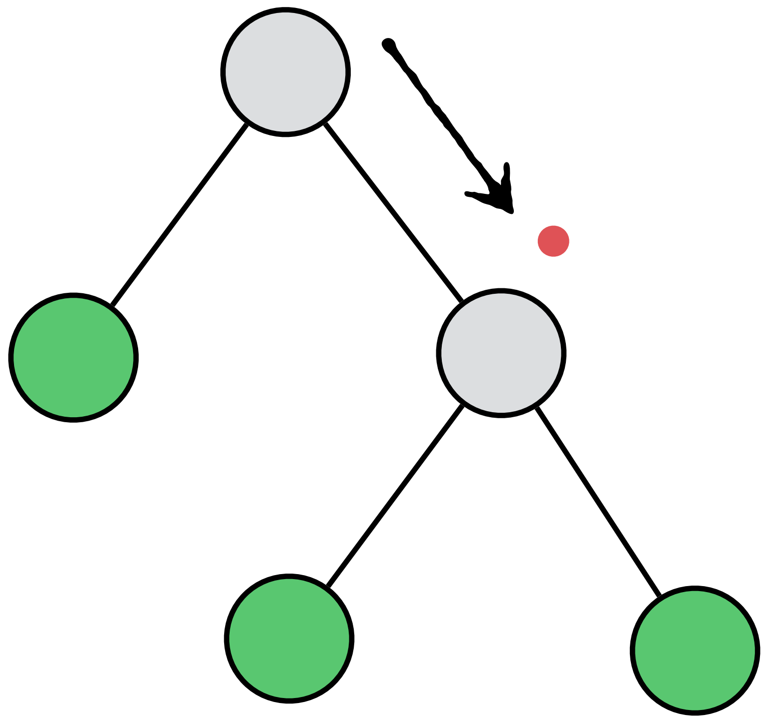
```
def perch(x1…xN,T):
 for xi in x1…xN:
  n = nearestNeigh(xi,T)
  v = split(n)
  recursiveRotate(v,T)
```

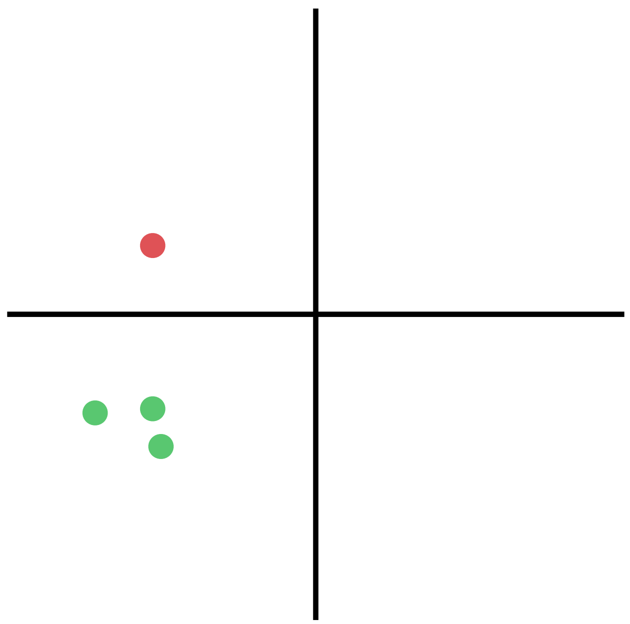




PERCH

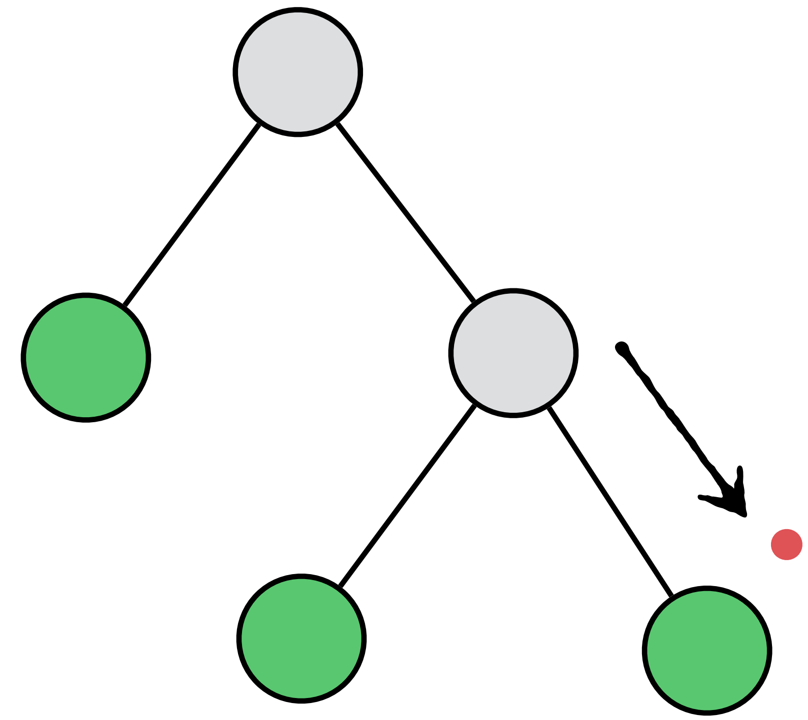
```
def perch(x1…xN,T):
 for xi in x1…xN:
  n = nearestNeigh(xi,T)
  v = split(n)
  recursiveRotate(v,T)
```

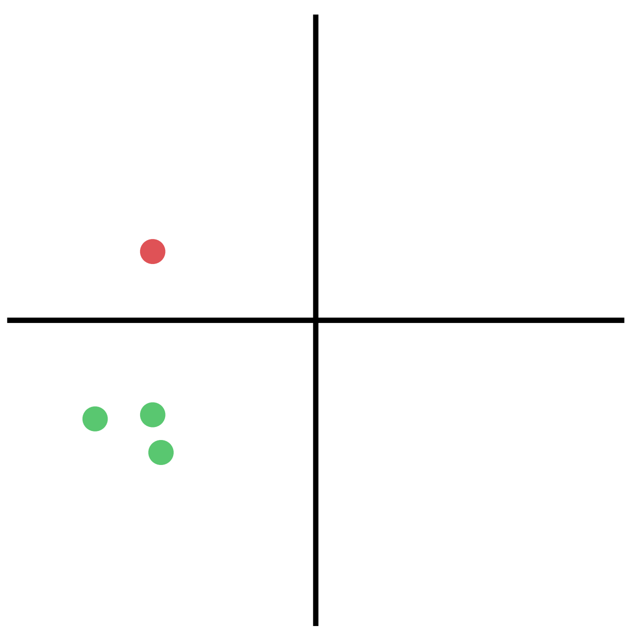




PERCH

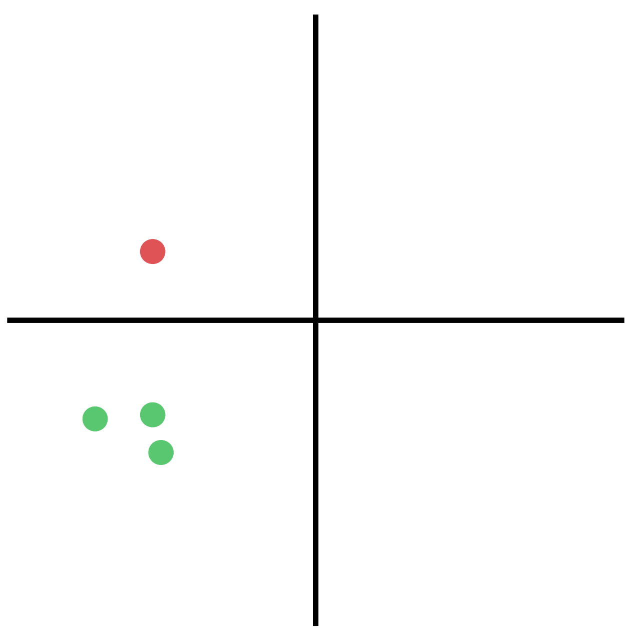
```
def perch(x1…xN,T):
 for xi in x1…xN:
  n = nearestNeigh(xi,T)
  v = split(n)
  recursiveRotate(v,T)
```

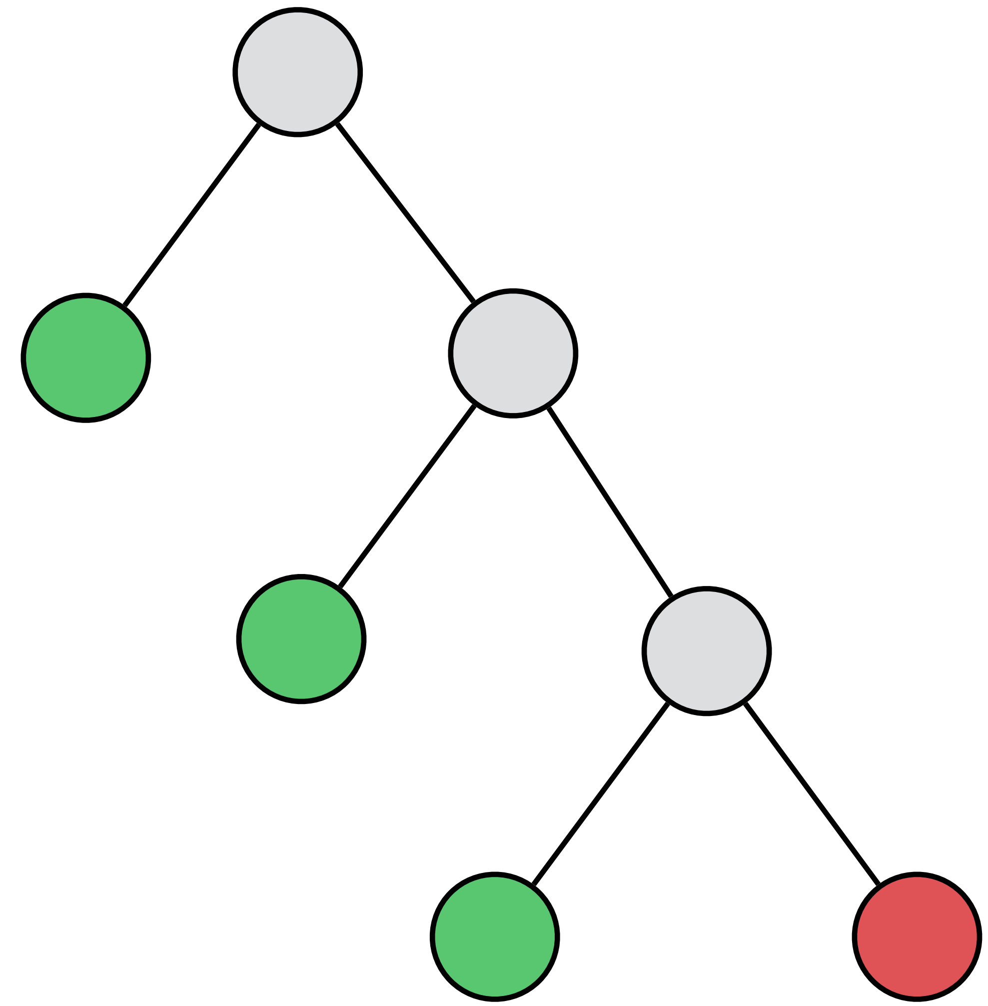




PERCH

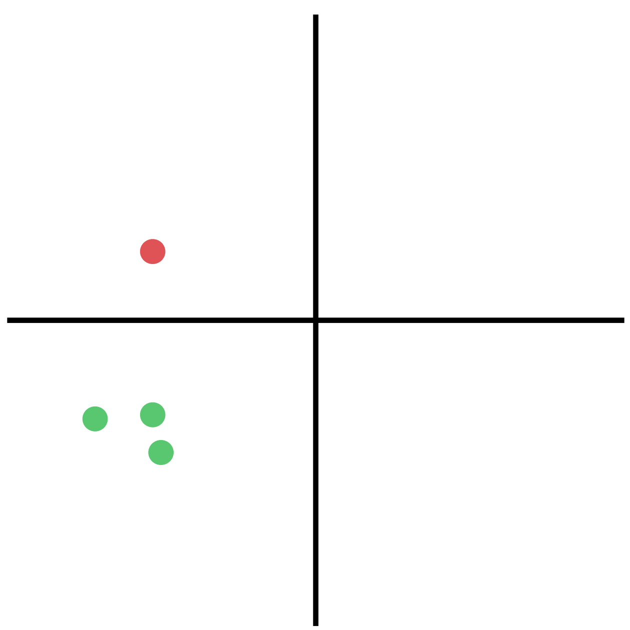
```
def perch(x1…xN,T):
 for xi in x1…xN:
  n = nearestNeigh(xi,T)
  v = split(n)
  recursiveRotate(v,T)
```

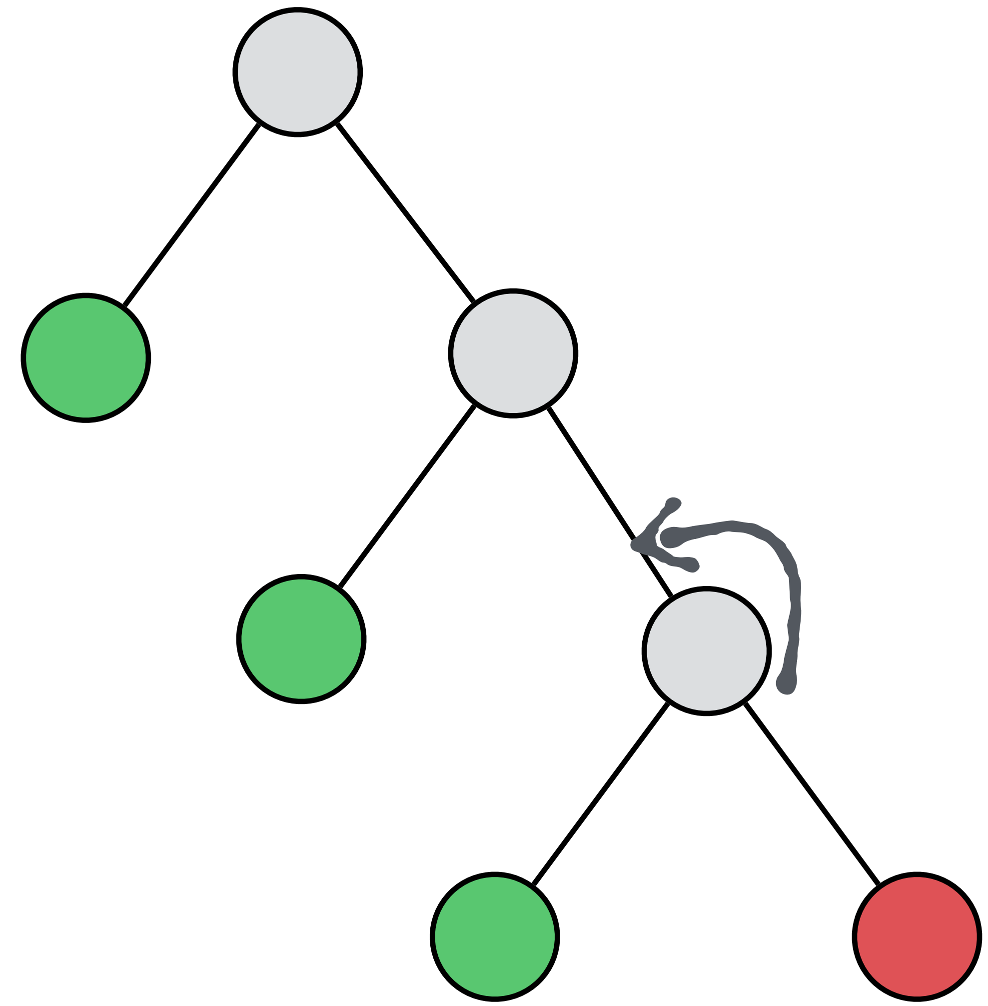




PERCH

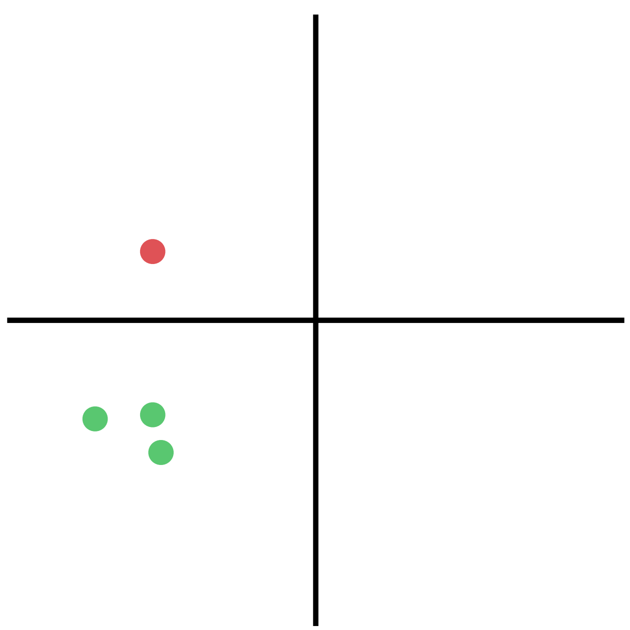
```
def perch(x₁…x_N,T):
 for xᵢ in x₁…x_N:
  n = nearestNeigh(xᵢ,T)
  v = split(n)
  recursiveRotate(v,T)
```

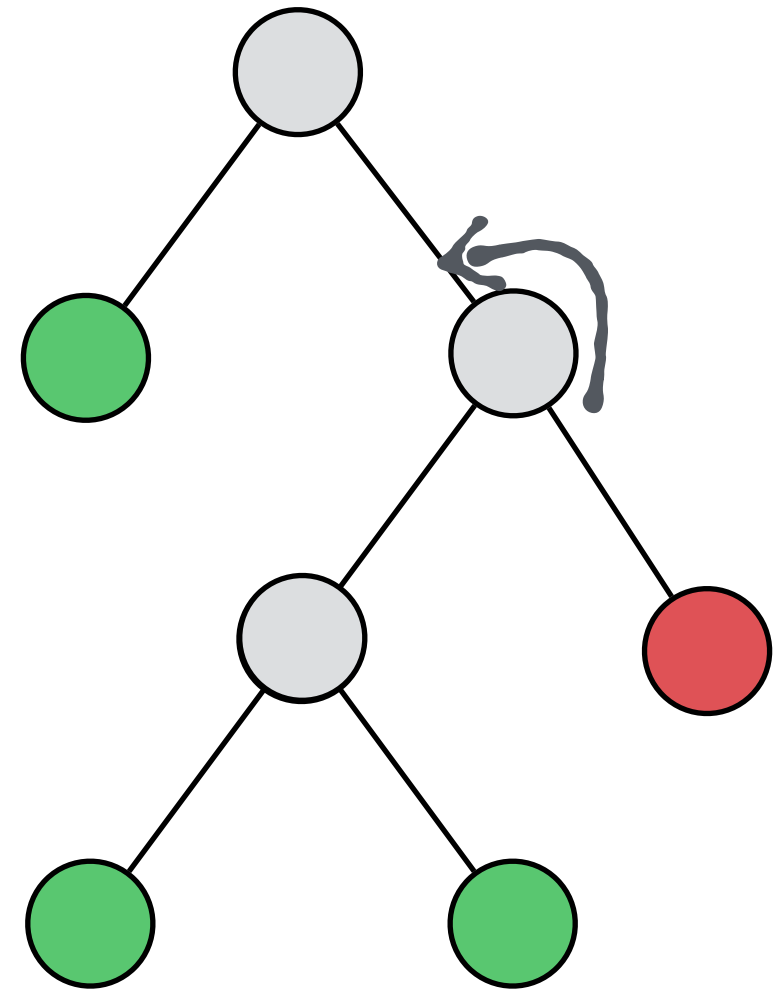




PERCH

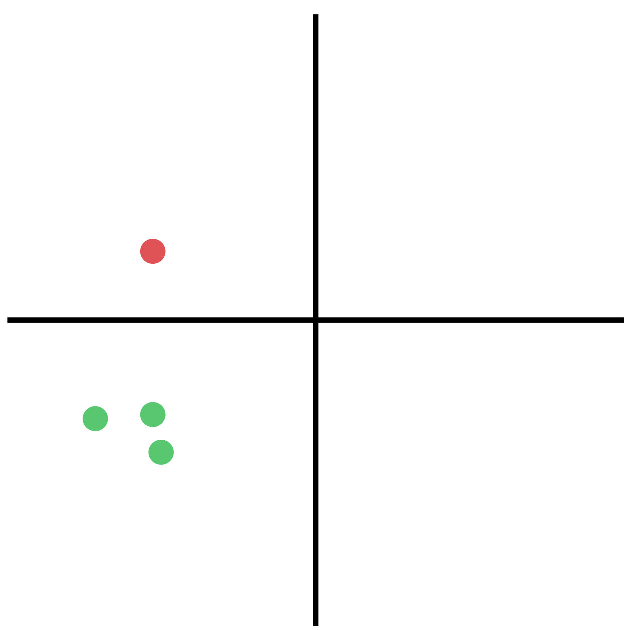
```
def perch(x1…xN,T):
 for xi in x1…xN:
  n = nearestNeigh(xi,T)
  v = split(n)
  recursiveRotate(v,T)
```

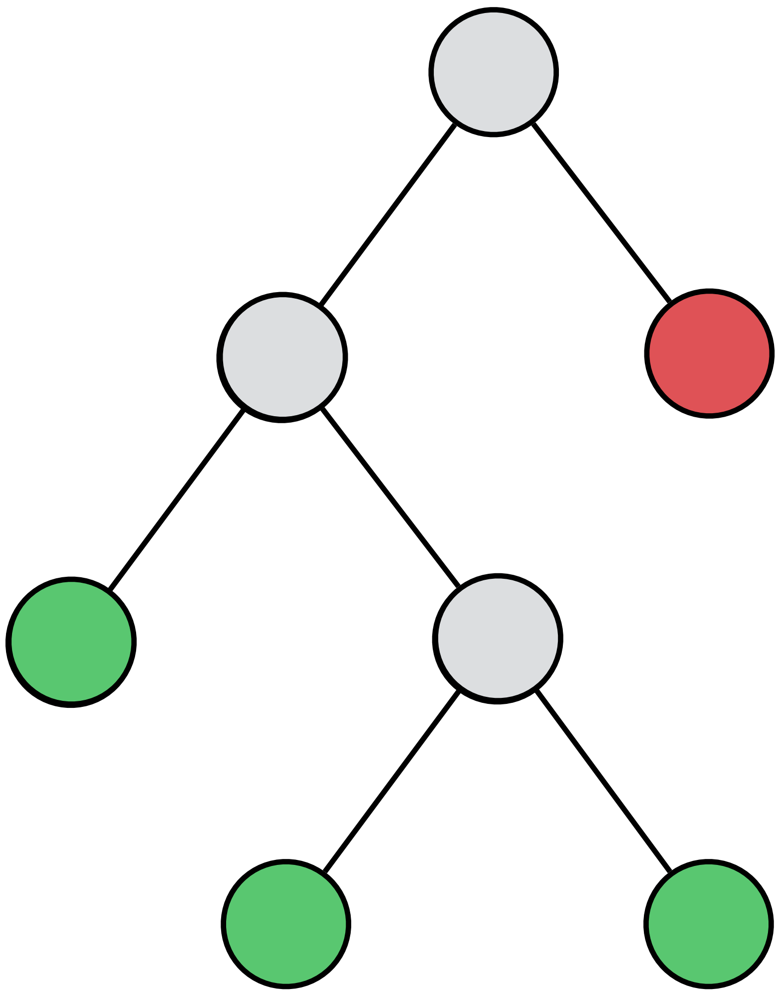




PERCH

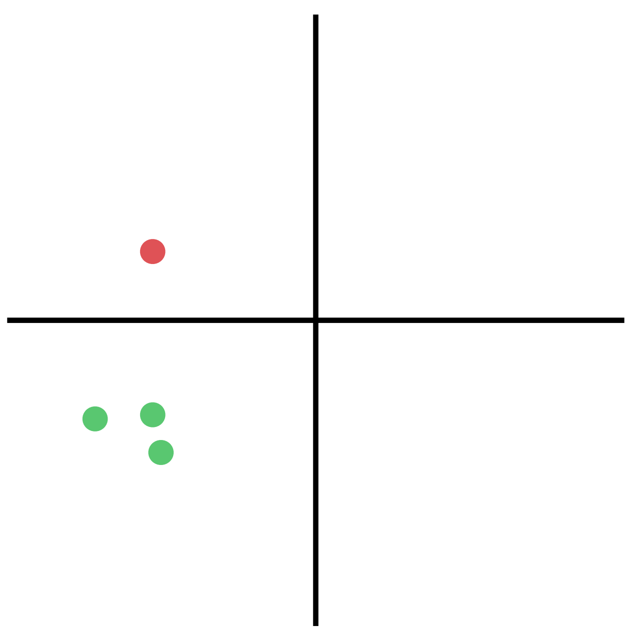
```
def perch(x1…xN,T):
 for xi in x1…xN:
  n = nearestNeigh(xi,T)
  v = split(n)
  recursiveRotate(v,T)
```

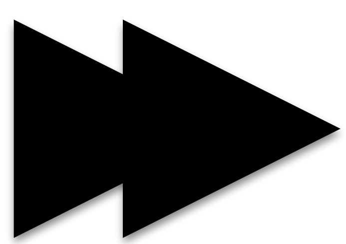




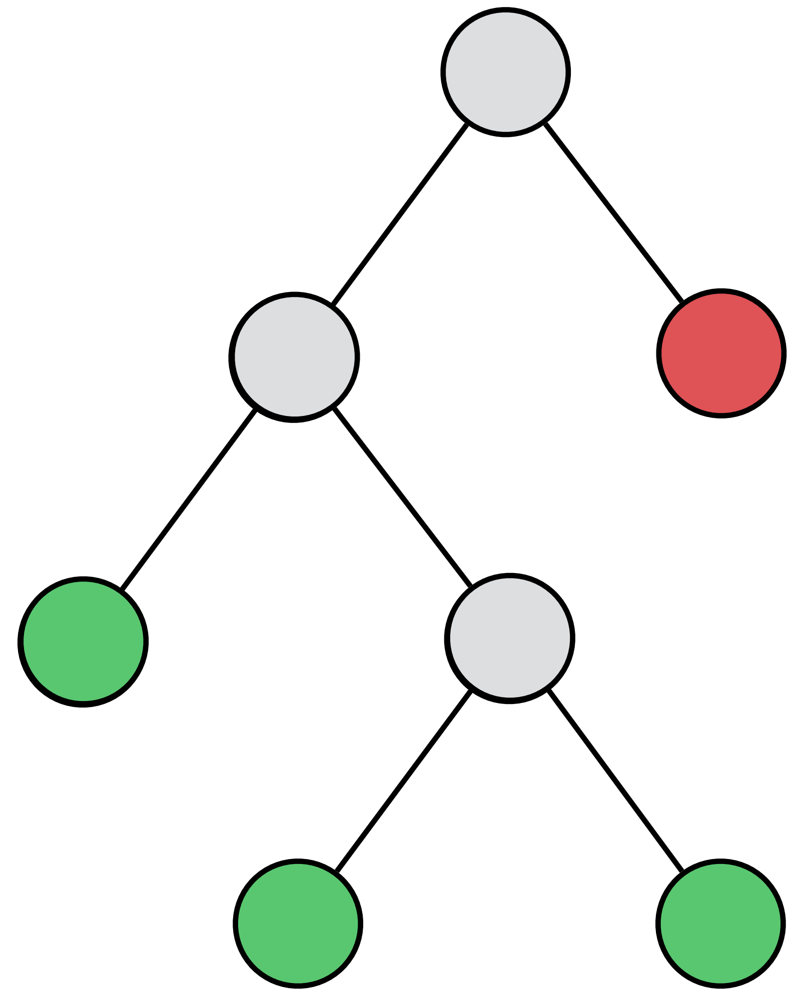
PERCH

```
def perch(x_1…x_N,T):
 for x_i in x_1…x_N:
  n = nearestNeigh(x_i,T)
  v = split(n)
  recursiveRotate(v,T)
```



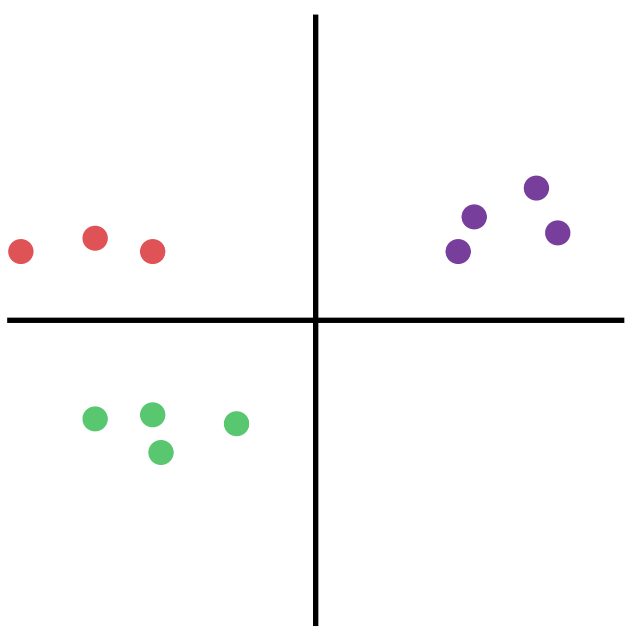


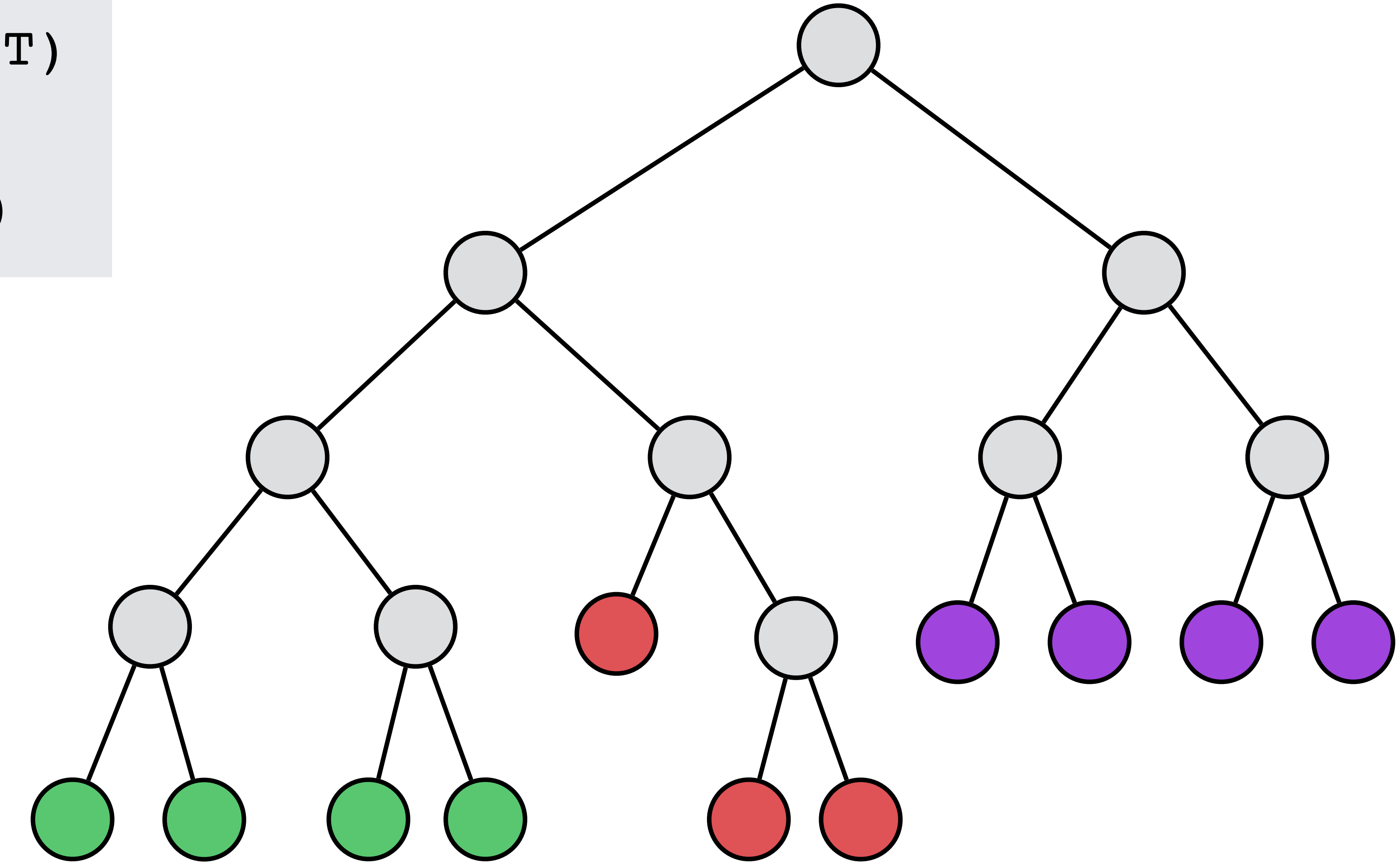
Fast Forward



PERCH

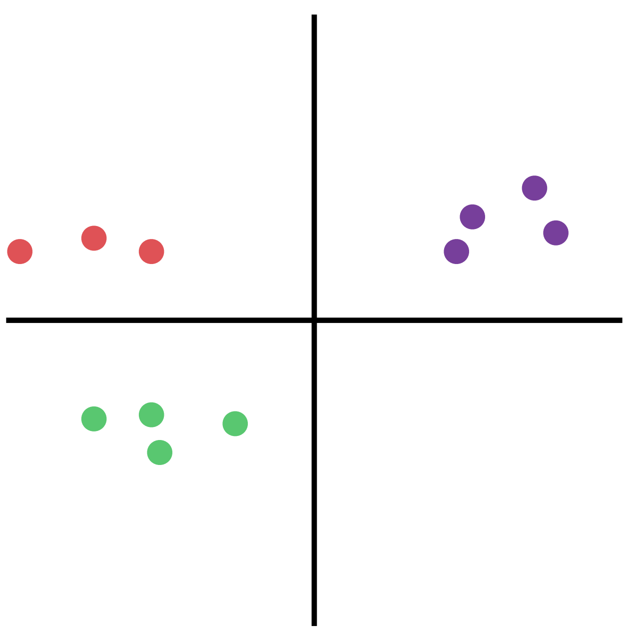
```
def perch(x1…xN,T):
 for xi in x1…xN:
  n = nearestNeigh(xi,T)
  v = split(n)
  recursiveRotate(v,T)
```

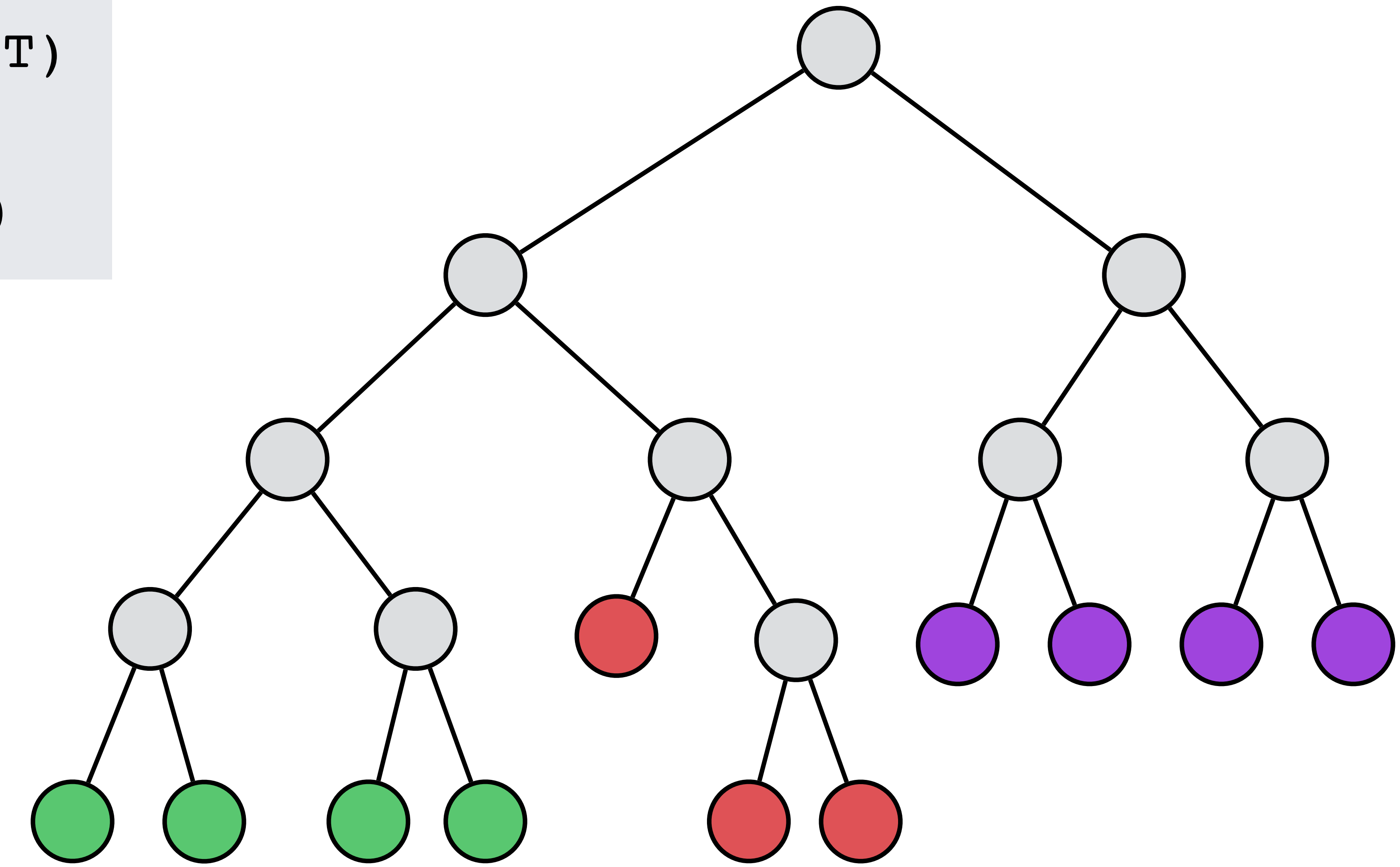




Perch

```
def perch(x1…xN,T):
 for xi in x1…xN:
  n = nearestNeigh(xi,T)
  v = split(n)
  recursiveRotate(v,T)
```

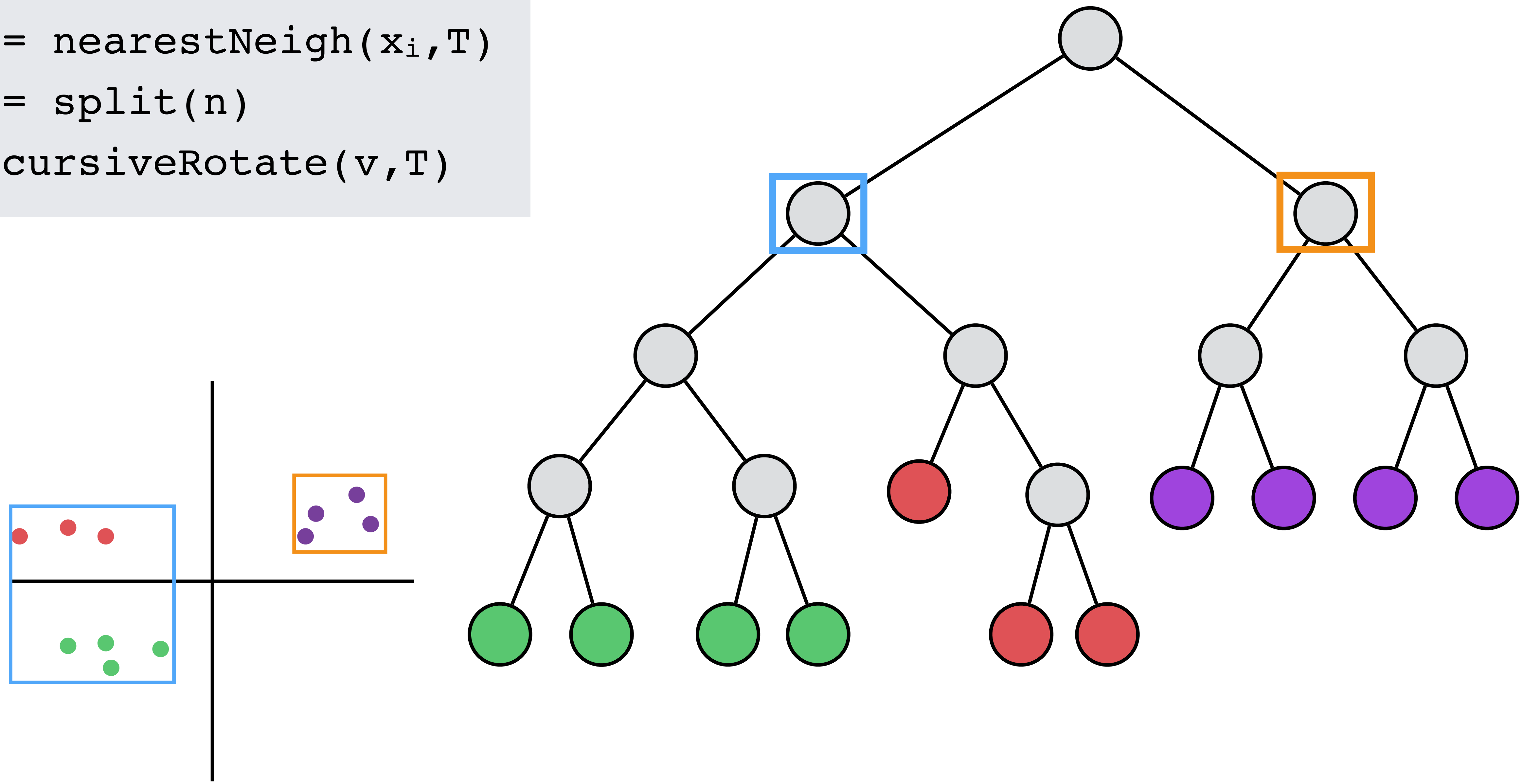




Data structures used for efficiency?

PERCH

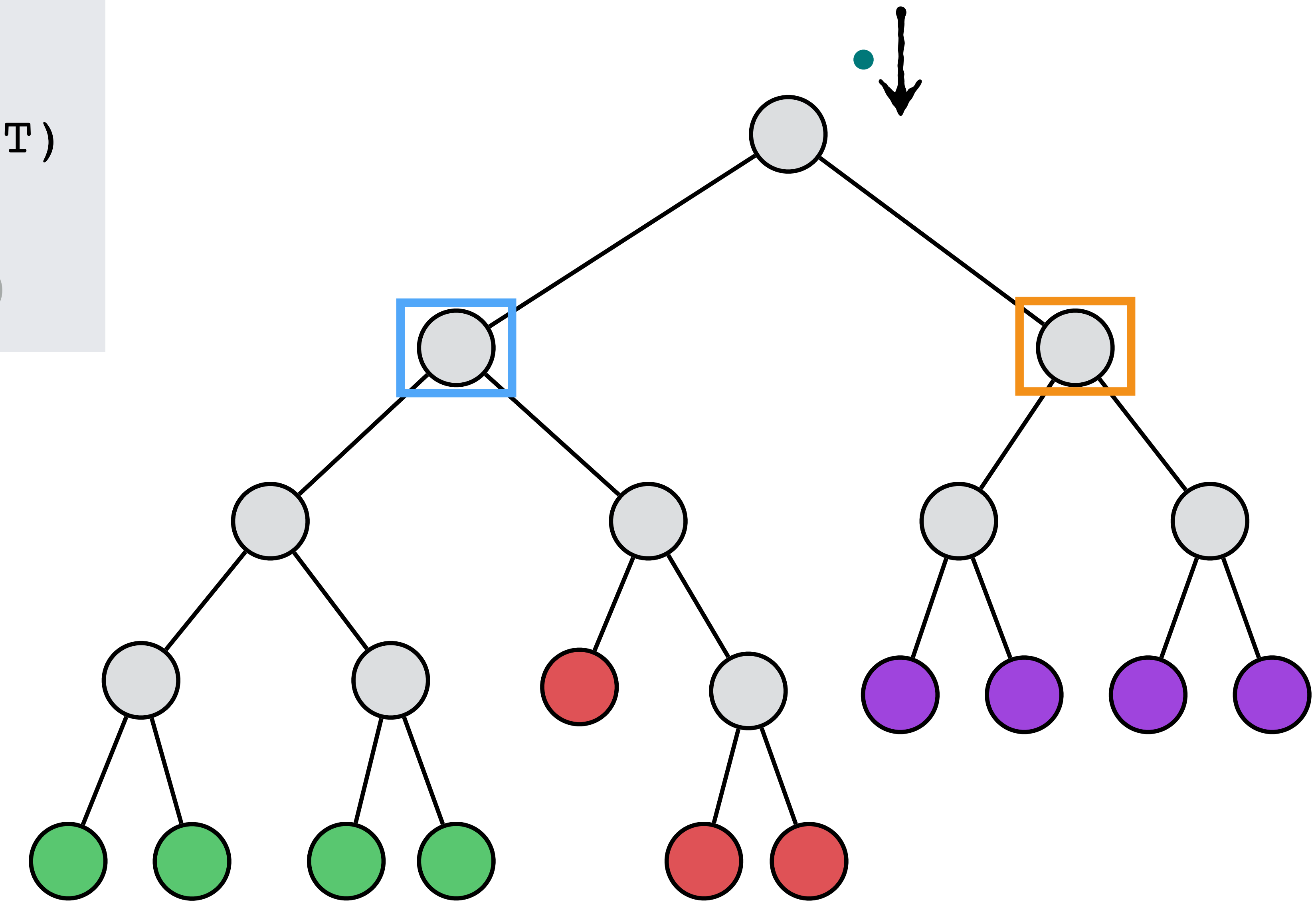
```
def perch(x1…xN,T):
 for xi in x1…xN:
  n = nearestNeigh(xi,T)
  v = split(n)
  recursiveRotate(v,T)
```

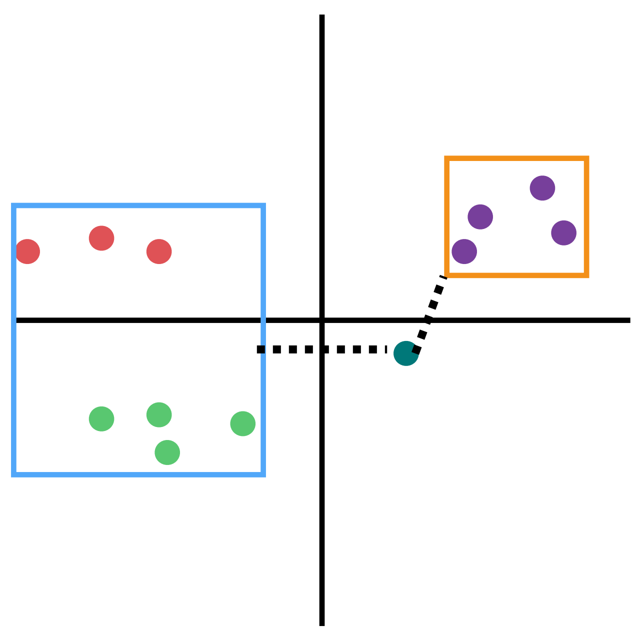


Bounding Boxes

PERCH

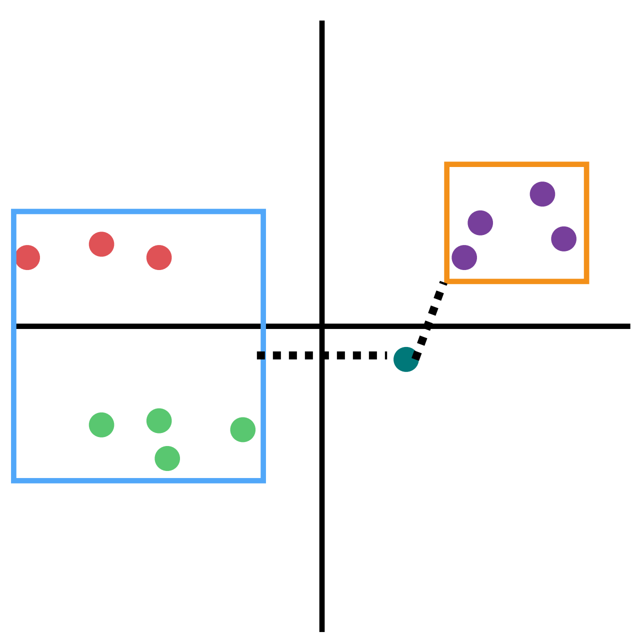
```
def perch(x1…xN,T):
 for xi in x1…xN:
  n = nearestNeigh(xi,T)
  v = split(n)
  recursiveRotate(v,T)
```

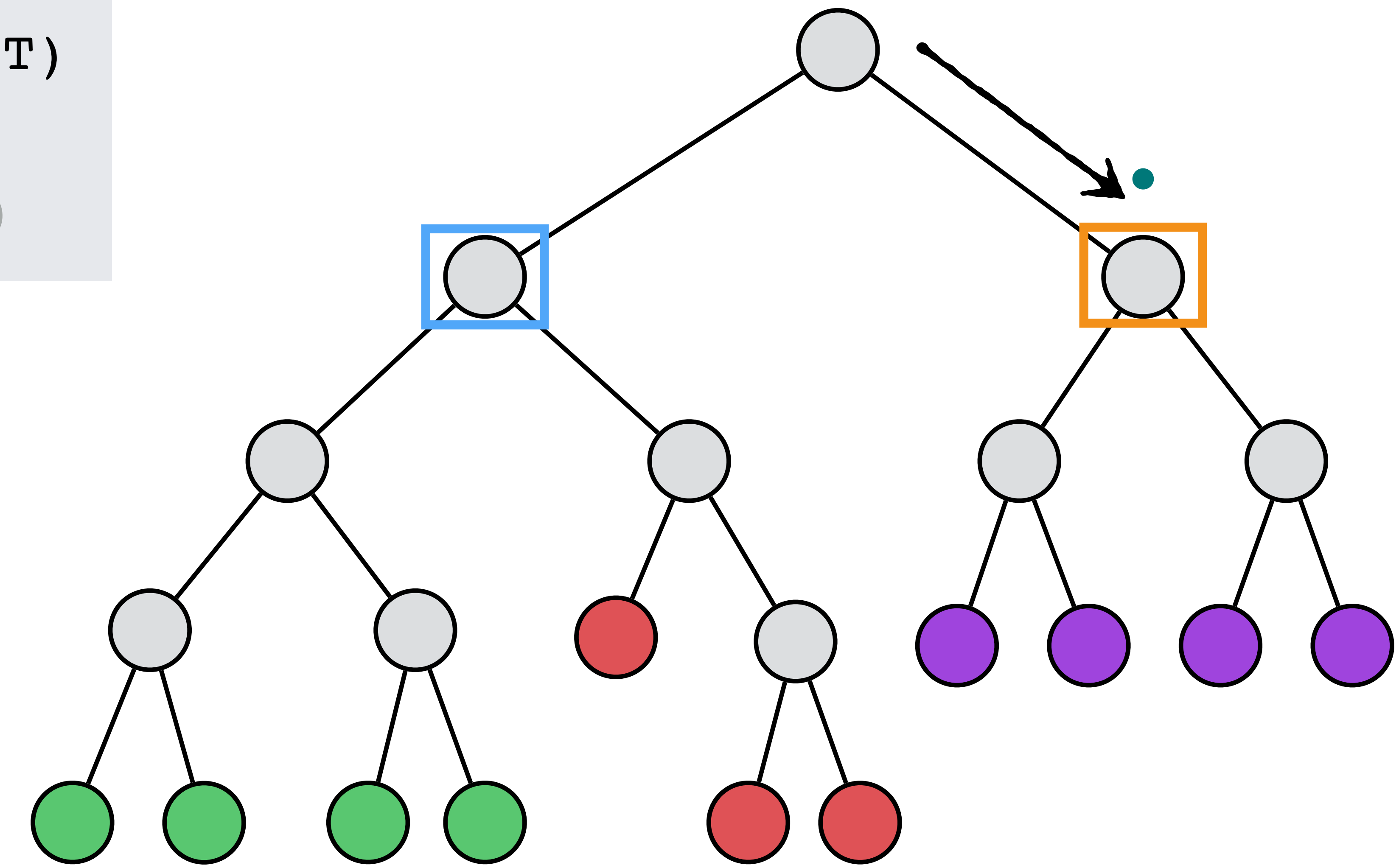




PERCH

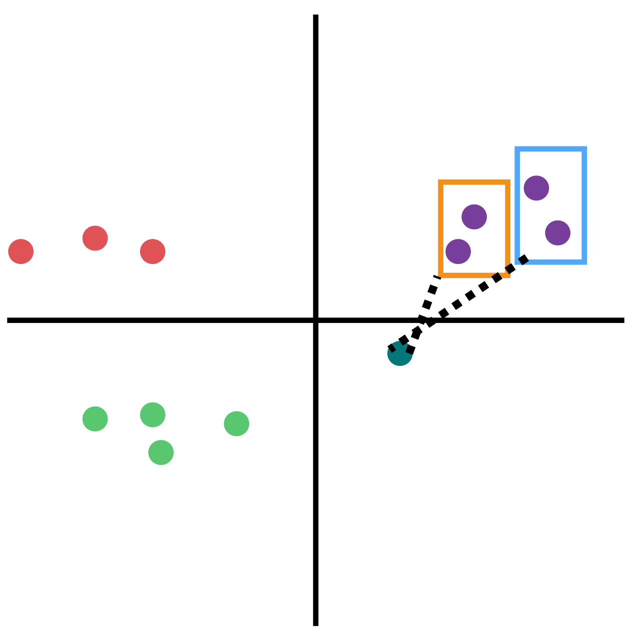
```
def perch(x₁…xN,T):
 for xi in x₁…xN:
  n = nearestNeigh(xi,T)
  v = split(n)
  recursiveRotate(v,T)
```

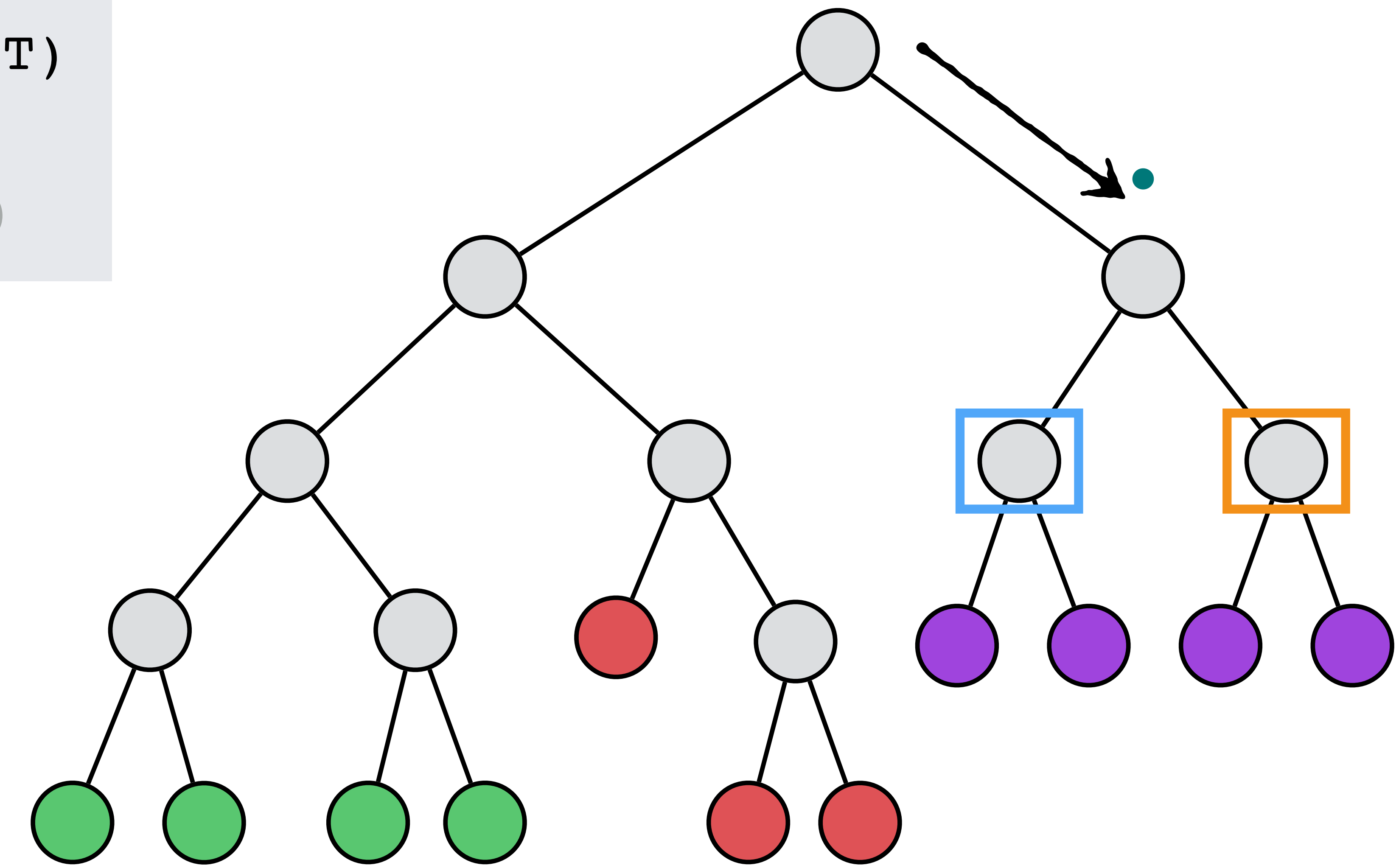




PERCH

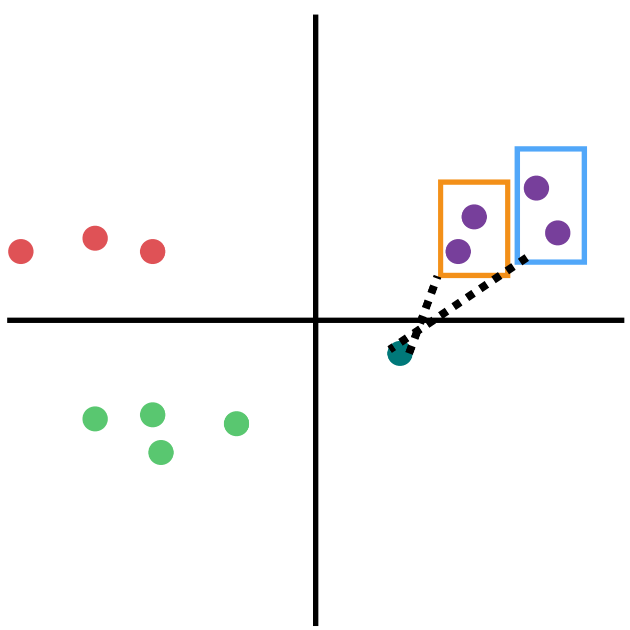
```
def perch(x1…xN,T):
 for xi in x1…xN:
  n = nearestNeigh(xi,T)
  v = split(n)
  recursiveRotate(v,T)
```

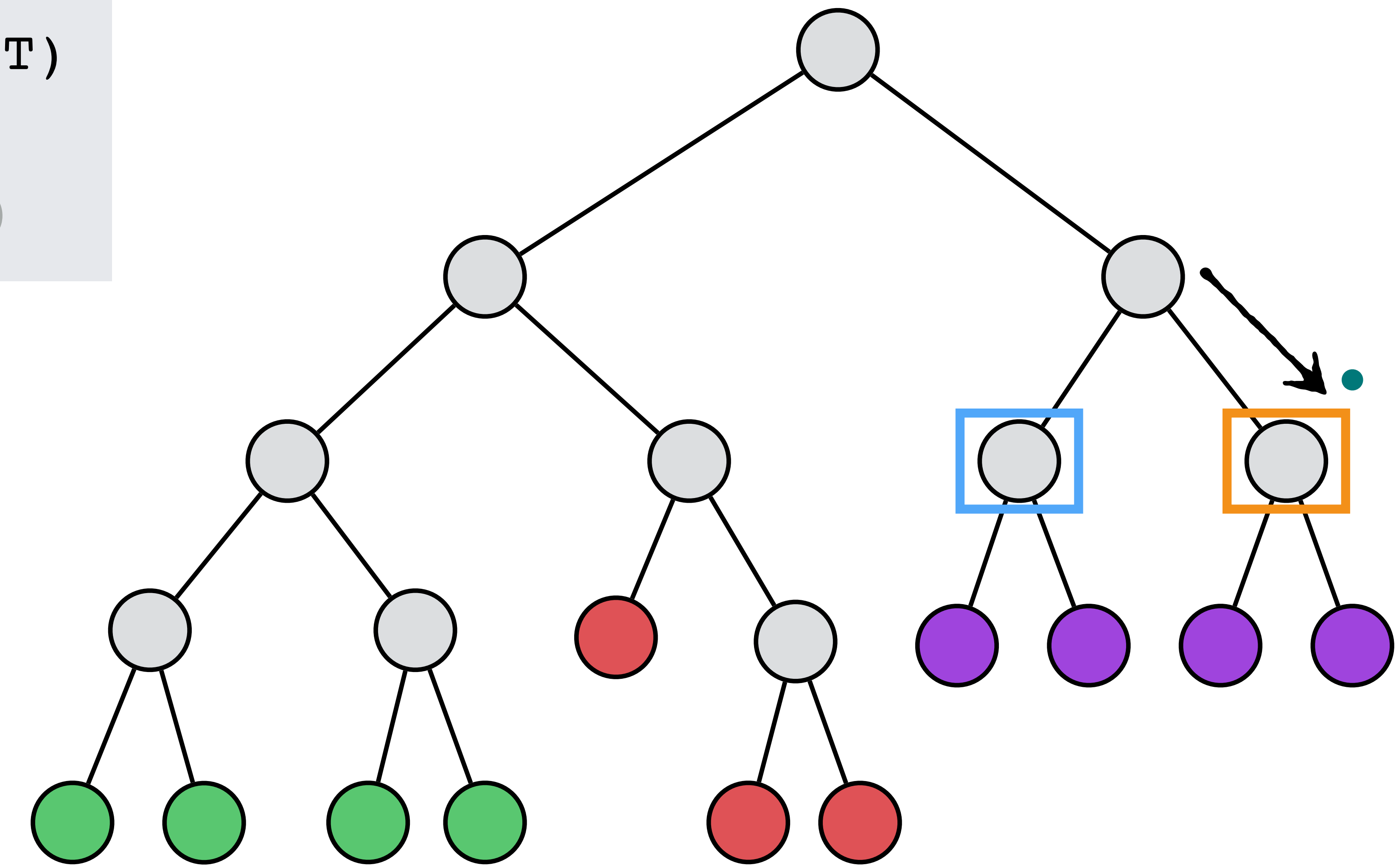




PERCH

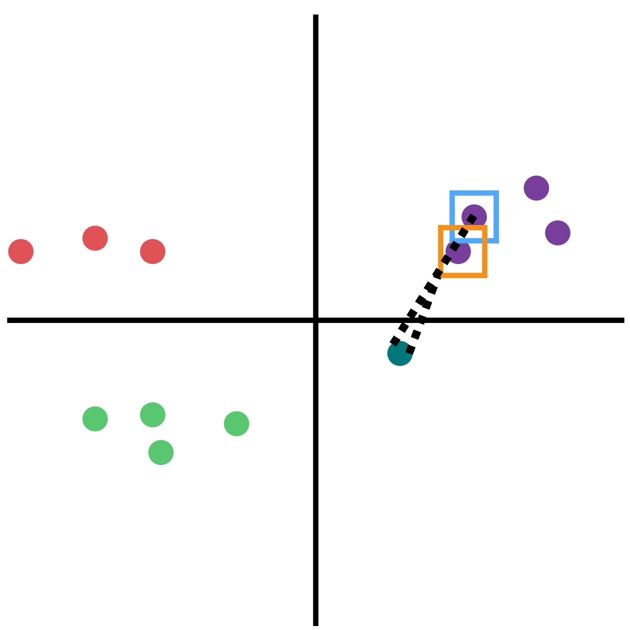
```
def perch(x1…xN,T):
 for xi in x1…xN:
  n = nearestNeigh(xi,T)
  v = split(n)
  recursiveRotate(v,T)
```

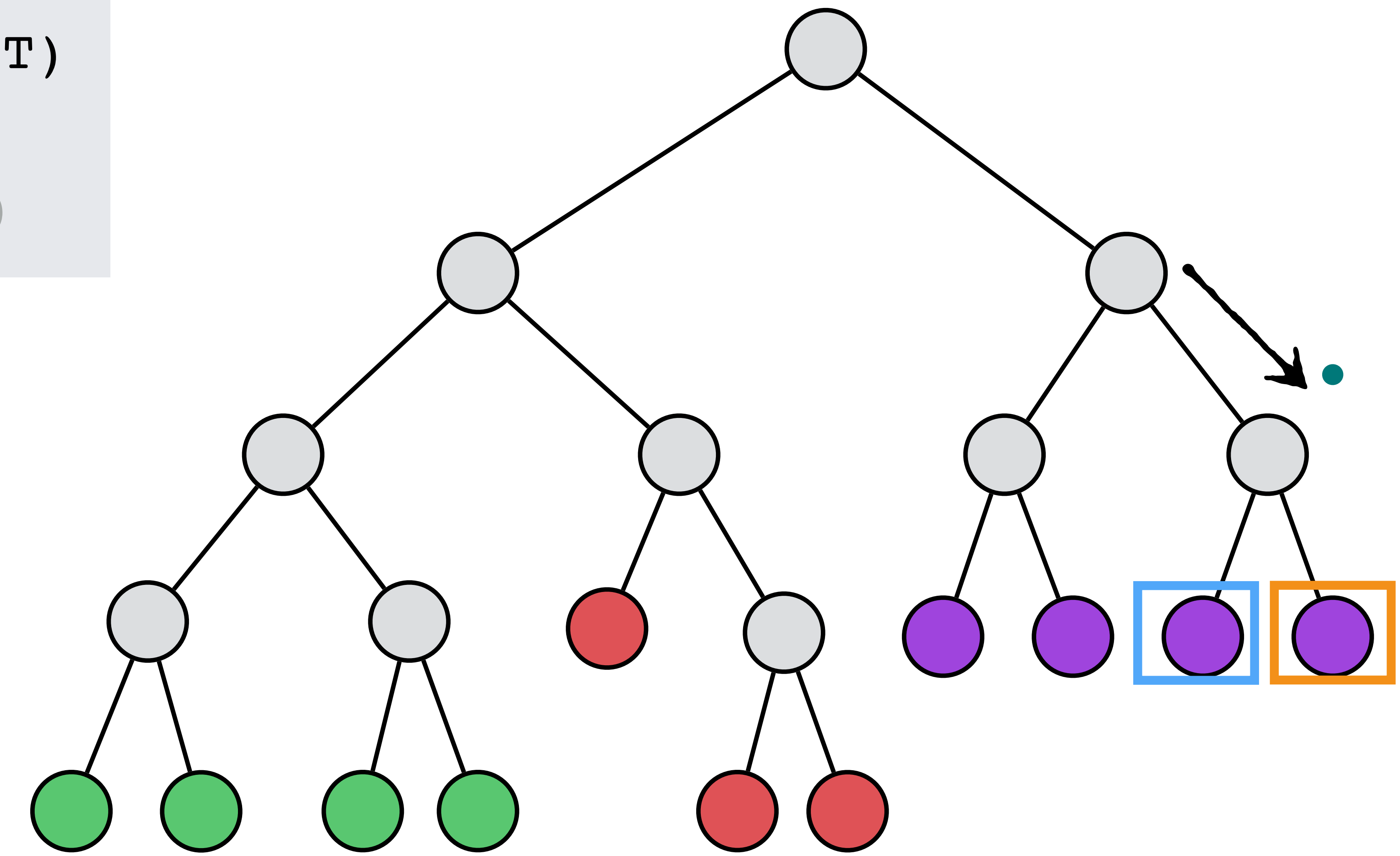




Perch

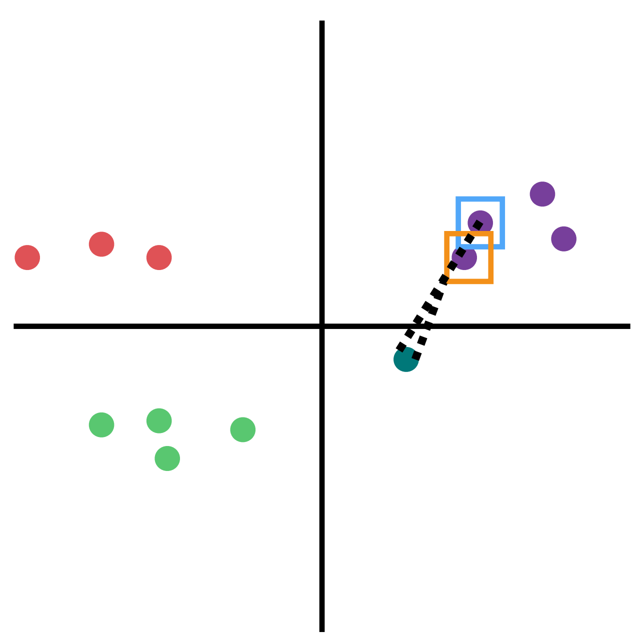
```
def perch(x_1…x_N,T):
 for x_i in x_1…x_N:
  n = nearestNeigh(x_i,T)
  v = split(n)
  recursiveRotate(v,T)
```

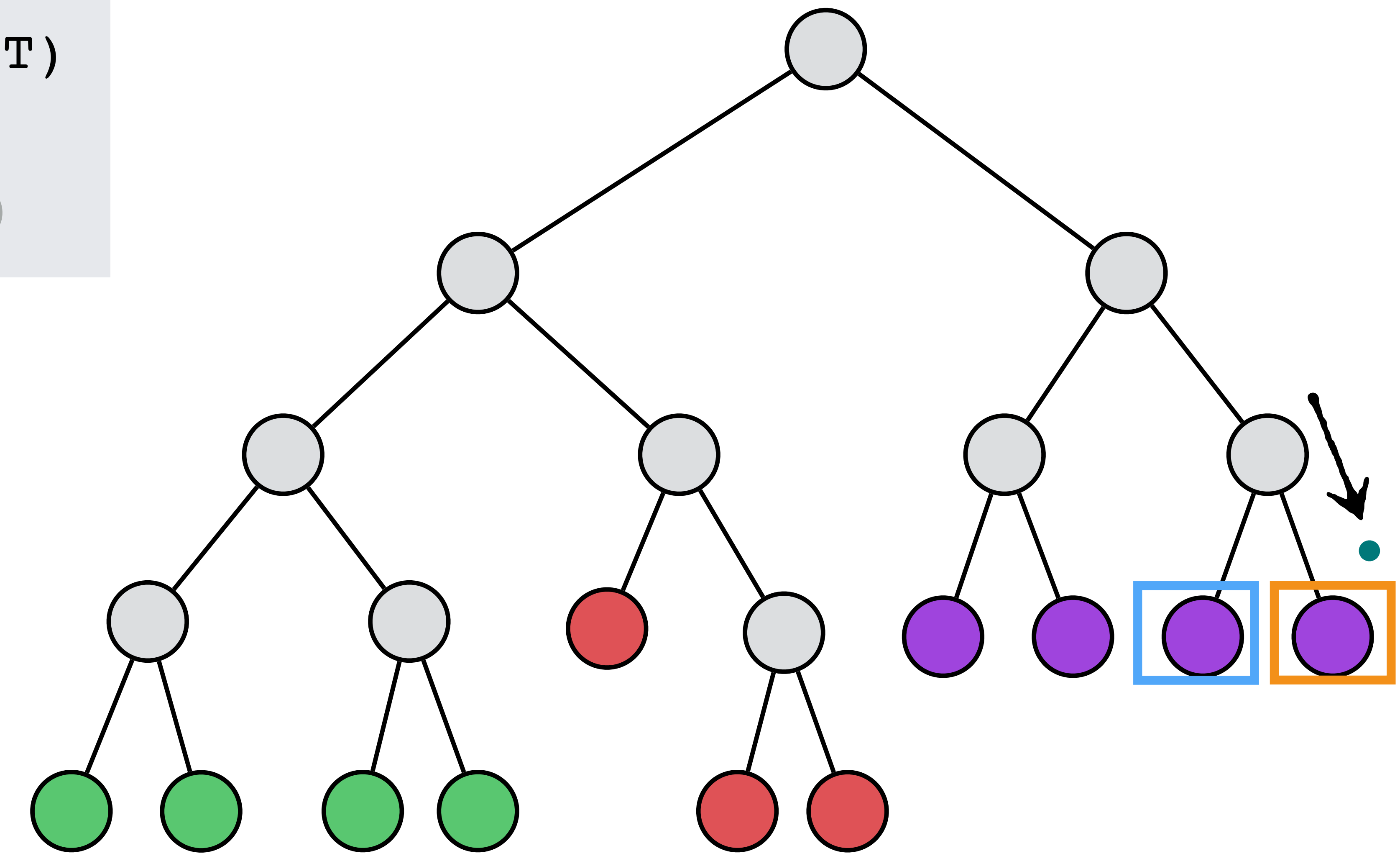




PERCH

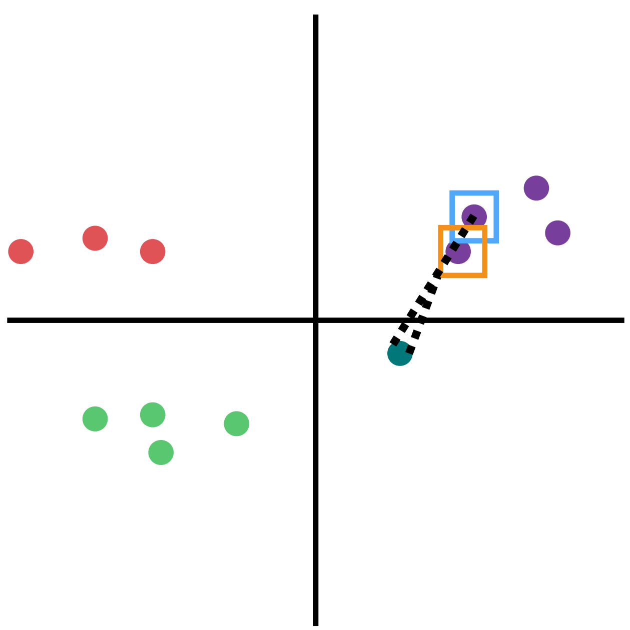
```
def perch(x_1…x_N,T):
 for x_i in x_1…x_N:
  n = nearestNeigh(x_i,T)
  v = split(n)
  recursiveRotate(v,T)
```

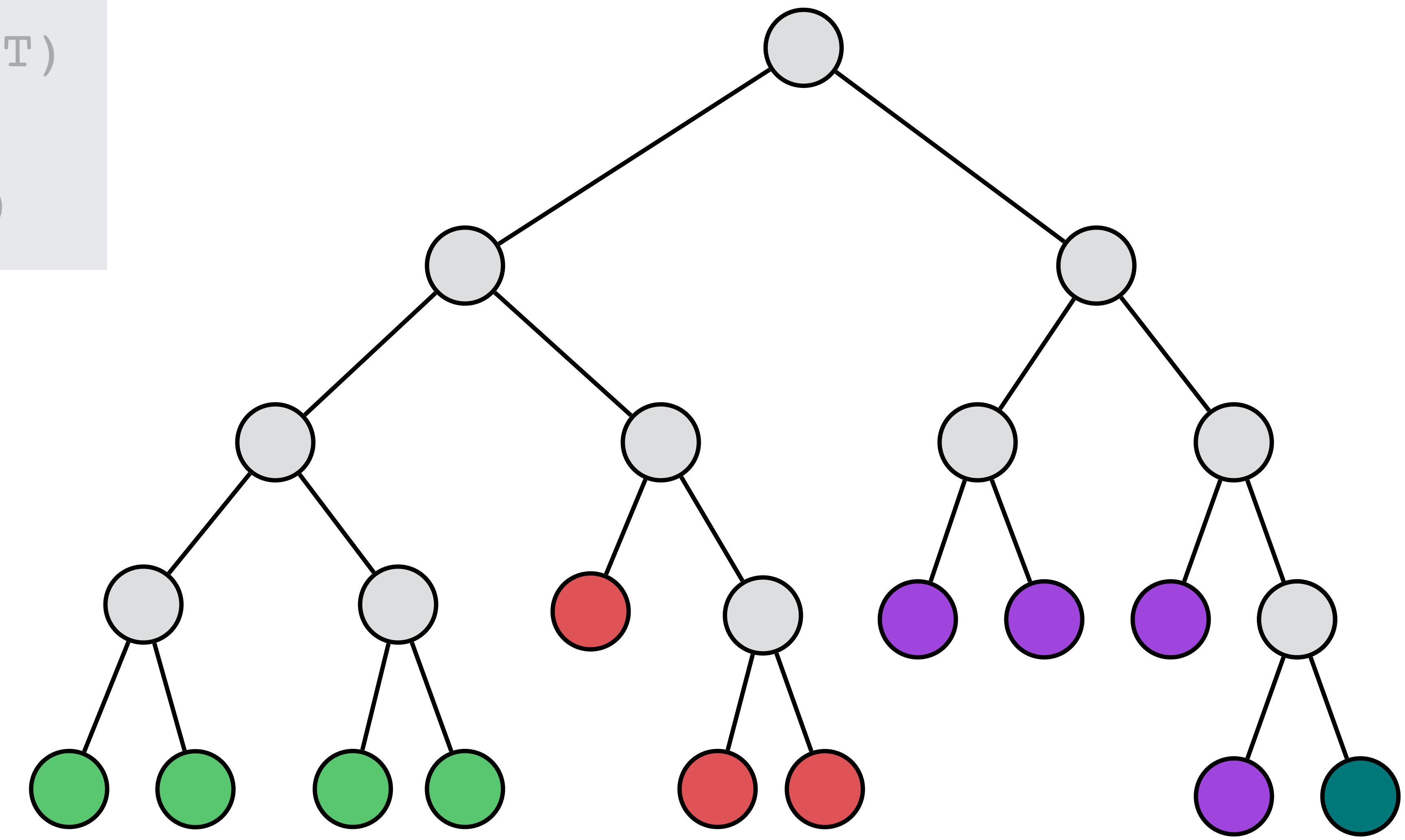




PERCH

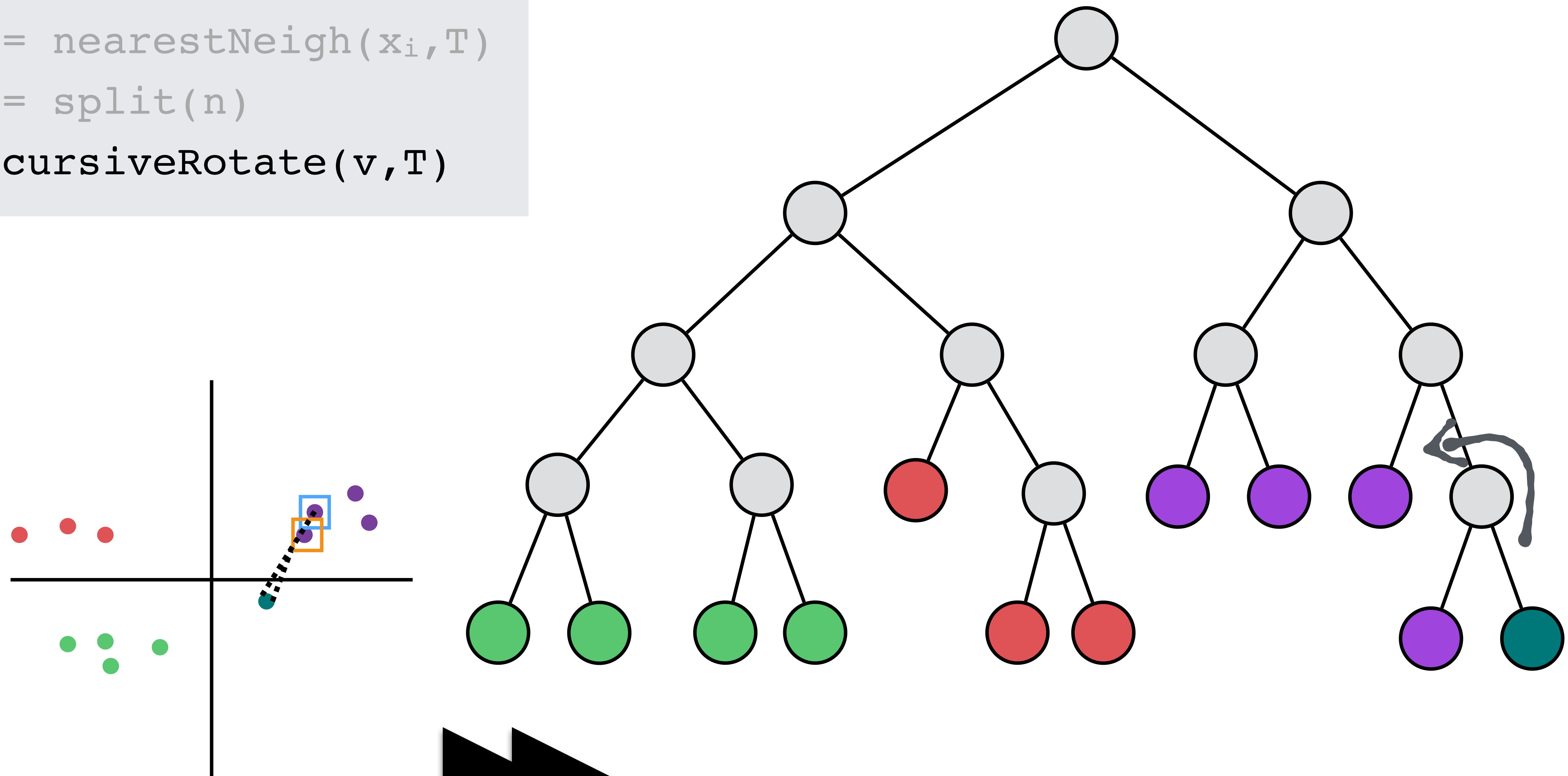
```
def perch(x1…xN,T):
 for xi in x1…xN:
  n = nearestNeigh(xi,T)
  v = split(n)
  recursiveRotate(v,T)
```





PERCH

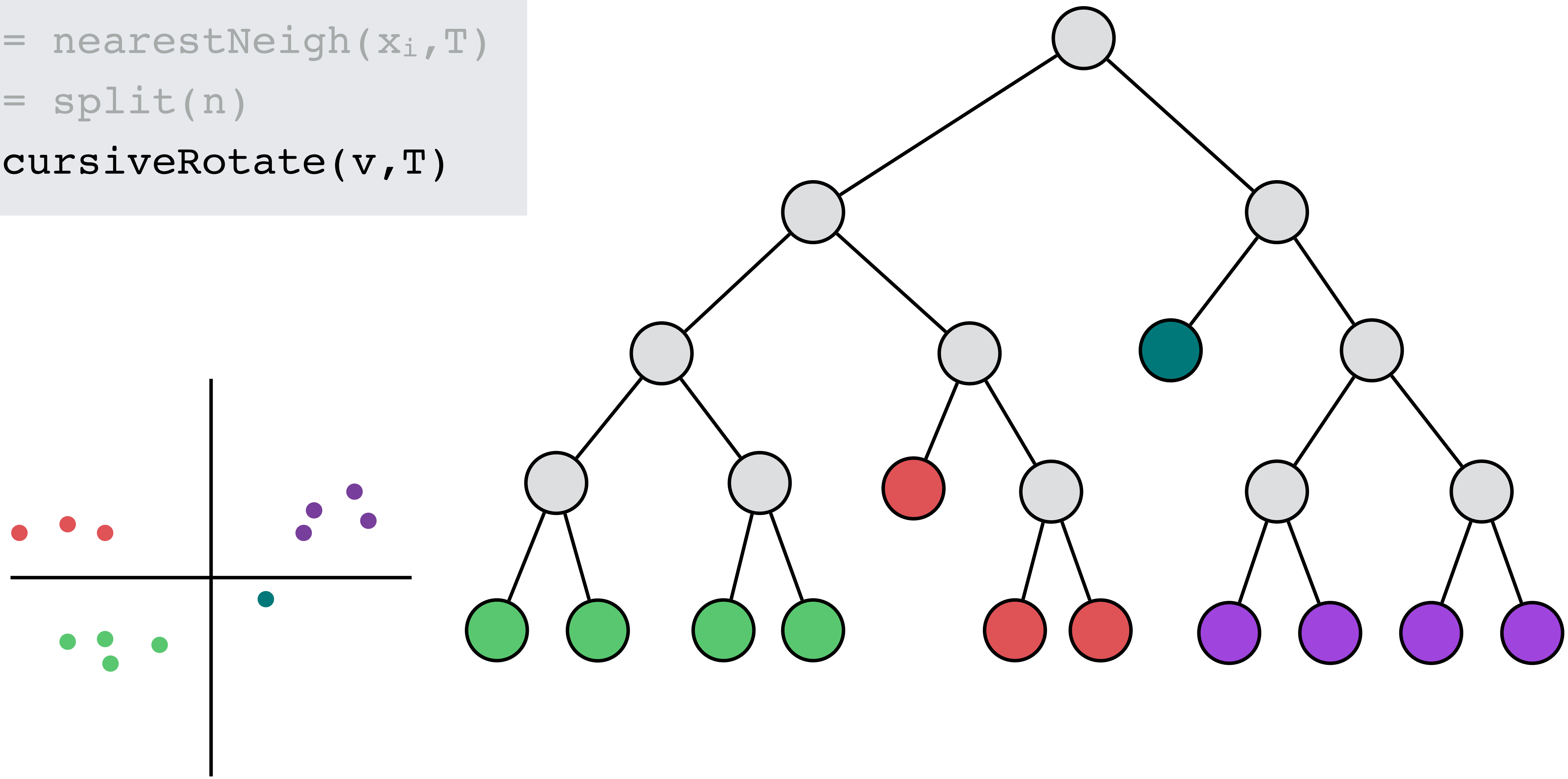
```
def perch(x1…xN,T):
 for xi in x1…xN:
  n = nearestNeigh(xi,T)
  v = split(n)
  recursiveRotate(v,T)
```



Fast Forward

PERCH

```
def perch(x1…xN,T):
 for xi in x1…xN:
  n = nearestNeigh(xi,T)
  v = split(n)
  recursiveRotate(v,T)
```



Perch

PERCH

Additional Efficiency Components:

PERCH

Additional Efficiency Components:

Balance Rotations

- Improve tree balance without sacrificing purity
- Speed up nearest neighbor search

PERCH

Additional Efficiency Components:

Balance Rotations

- Improve tree balance without sacrificing purity
- Speed up nearest neighbor search

Collapsed Mode

- Restrict the number of nodes in the tree
- Allows for clustering data that doesn’t fit in memory

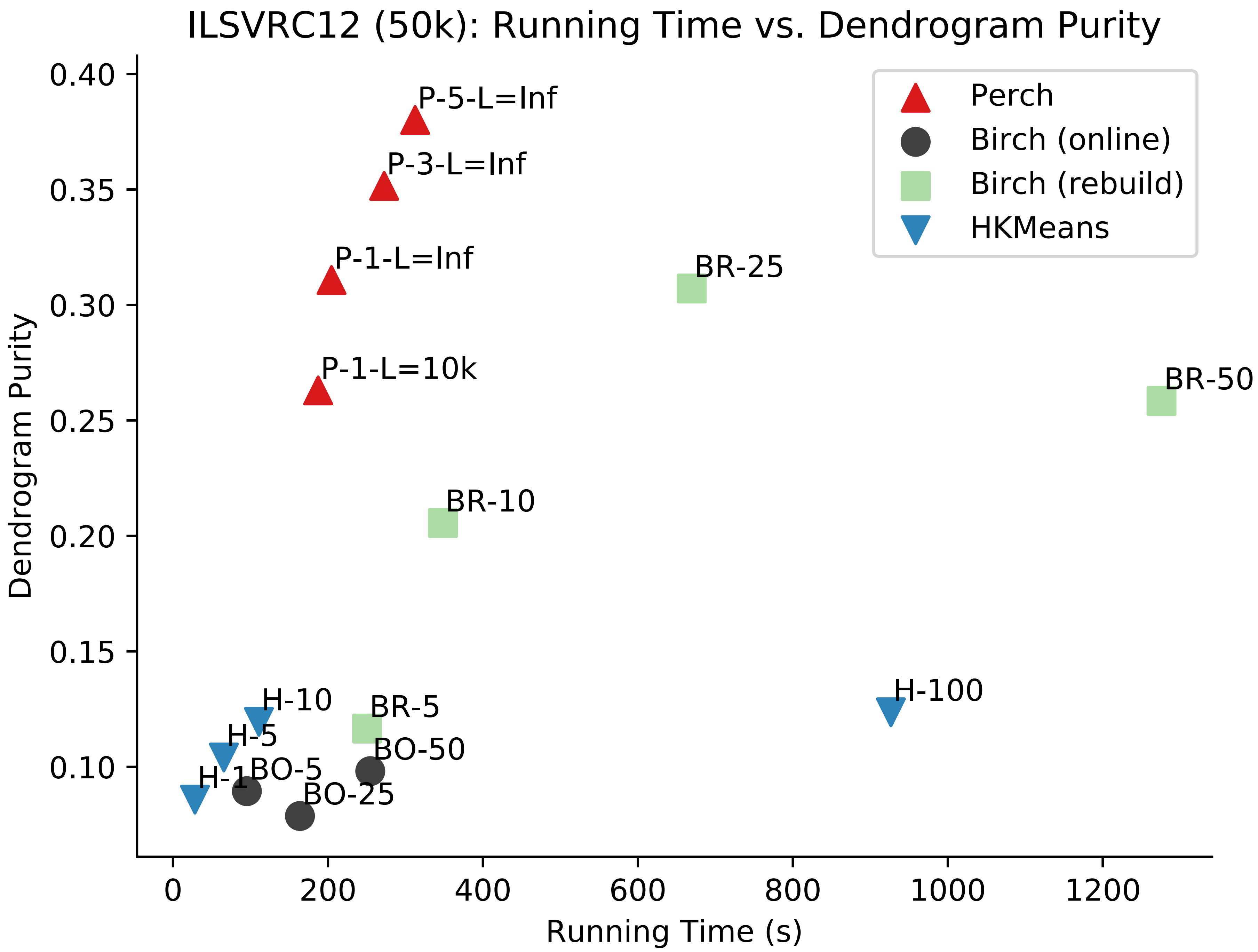
PERCH

Theoretical Guarantees:

On *separated* data, PERCH constructs trees with dendrogram purity 1.0, even when using balance rotations and collapsing.

Results

Results


ILSVRC12 (50k): Running Time vs. Dendrogram Purity
Dendrogram Purity
Running Time (s)
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0
200
400
600
800
1000
1200
Perch
Birch (online)
Birch (rebuild)
HKMeans
P-5-L=Inf
P-3-L=Inf
P-1-L=Inf
P-1-L=10k
BR-25
BR-50
BR-10
BR-5
H-10
H-5
H-1
H-100
BO-5
BO-50
BO-25

Results

Method	CovType	ILSVRC12 (50k)	ALOI	ILSVRC 12	Speaker	ImageNet (100k)
PERCH	**0.45 ± 0.004**	**0.53 ± 0.003**	**0.44 ± 0.004**	—	**0.37 ± 0.002**	**0.07 ± 0.00**
PERCH-BC	0.45 ± 0.004	0.36 ± 0.005	0.37 ± 0.008	**0.21 ± 0.017**	0.09 ± 0.001	0.03 ± 0.00
BIRCH (online)	0.44 ± 0.002	0.09 ± 0.006	0.21 ± 0.004	0.11 ± 0.006	0.02 ± 0.002	0.02 ± 0.00
MB-HAC-Com.	—	0.43 ± 0.005	0.15 ± 0.003	—	0.01 ± 0.002	—
MB-HAC-Cent.	0.44 ± 0.005	0.02 ± 0.000	0.30 ± 0.002	—	—	—
HKMmeans	0.44 ± 0.001	0.12 ± 0.002	**0.44 ± 0.001**	0.11 ± 0.003	0.12 ± 0.002	0.02 ± 0.00
BIRCH (rebuild)	0.44 ± 0.002	0.26 ± 0.003	0.32 ± 0.002	—	0.22 ± 0.006	0.03 ± 0.00

(a) Dendrogram Purity for Hierarchical Clustering.

Method	CoverType	ILSVRC 12 (50k)	ALOI	ILSVRC 12	Speaker	ImageNet (100K)
PERCH	22.96 ± 0.7	**54.30 ± 0.3**	**44.21 ± 0.2**	—	**31.80 ± 0.1**	**6.178 ± 0.0**
PERCH-BC	22.97 ± 0.8	37.98 ± 0.5	37.48 ± 0.7	25.75 ± 1.7	1.05 ± 0.1	4.144 ± 0.04
SKM++	23.80 ± 0.4	28.46 ± 2.2	37.53 ± 1.0	—	—	—
BICO	**24.53 ± 0.4**	45.18 ± 1.0	32.984 ± 3.4	—	—	—
MB-KM	24.27 ± 0.6	51.73 ± 1.8	40.84 ± 0.5	**56.17 ± 0.4**	1.73 ± 0.141	5.642 ± 0.00
DBSCAN	—	16.95	—	—	22.63	—

(b) Pairwise F1 for Flat Clustering.

Thanks!

Questions?

https://arxiv.org/abs/1704.01858

https://github.com/iesl/xcluster