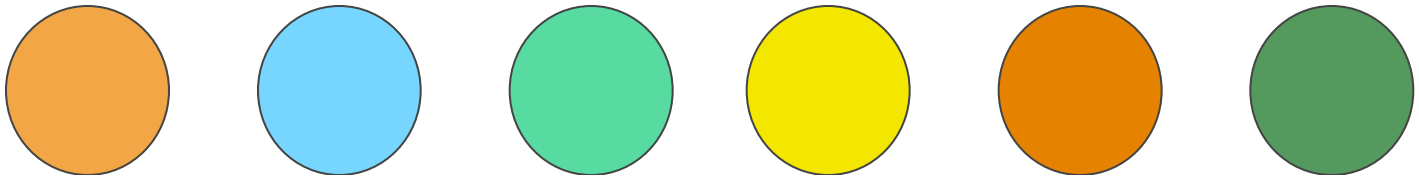
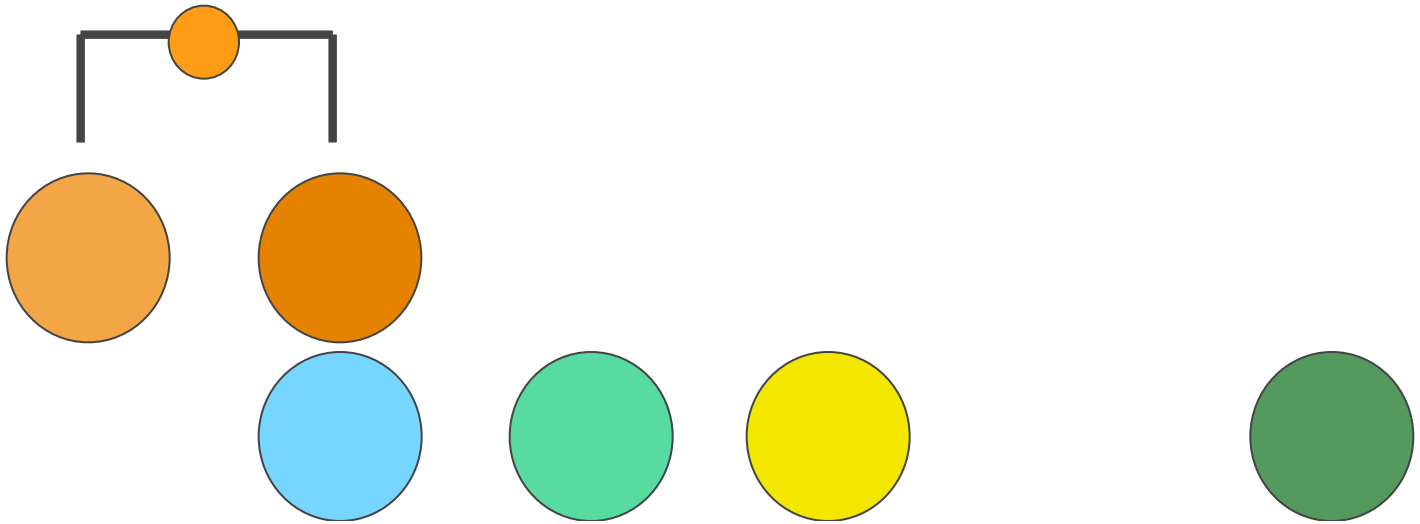
Hierarchical Clustering

Thanks to Kevin Wayne for many of the slides

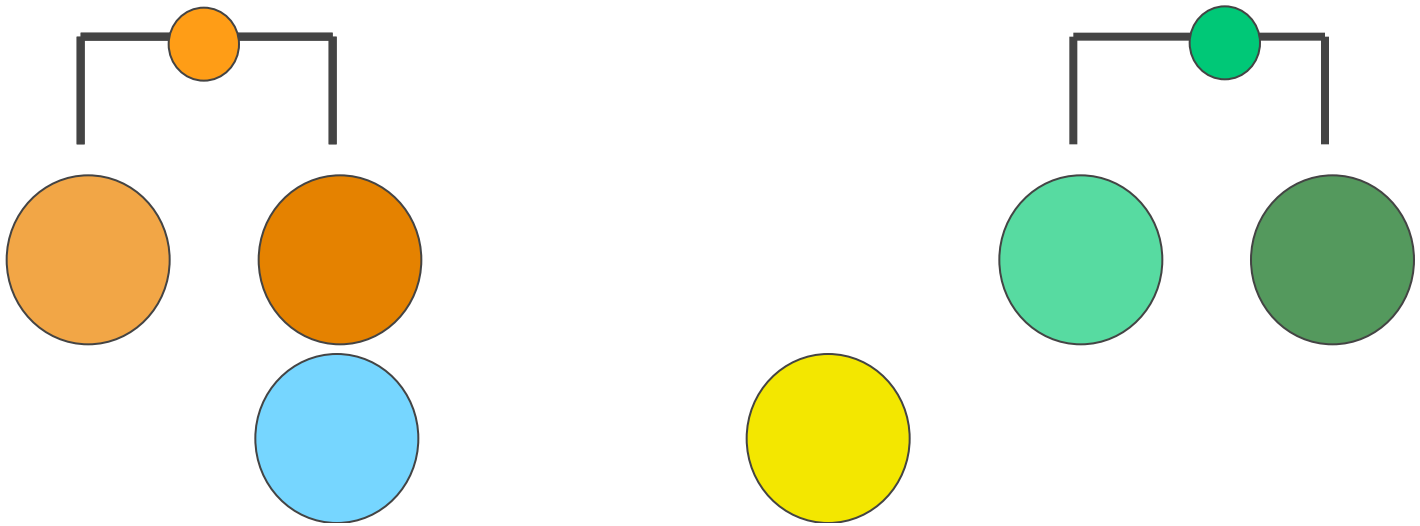
Hierarchical clustering



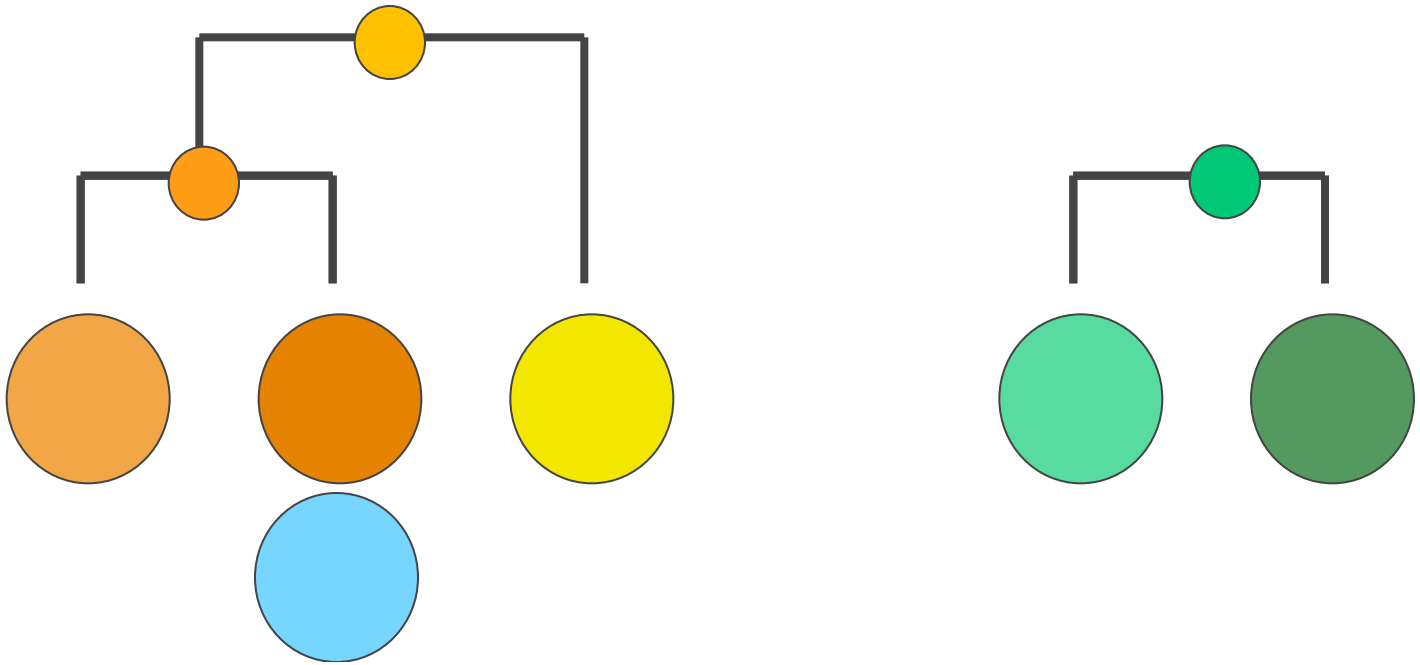
Hierarchical clustering



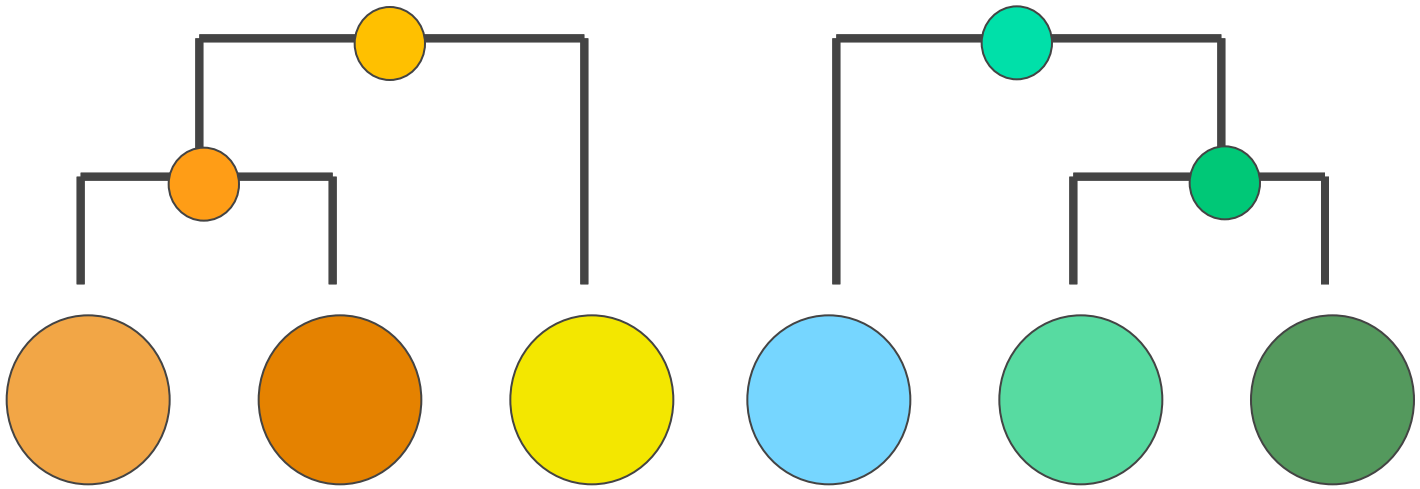
Hierarchical clustering



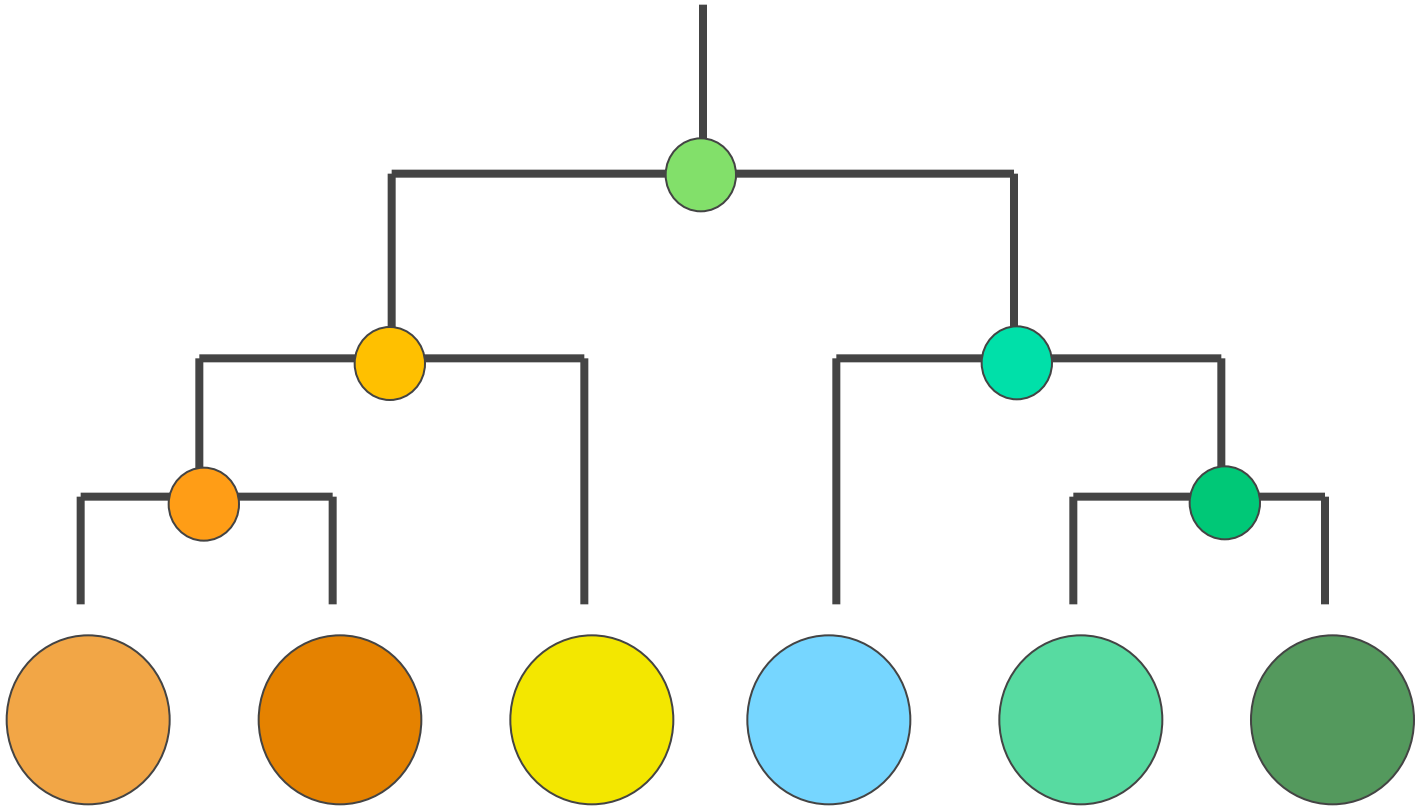
Hierarchical clustering



Hierarchical clustering

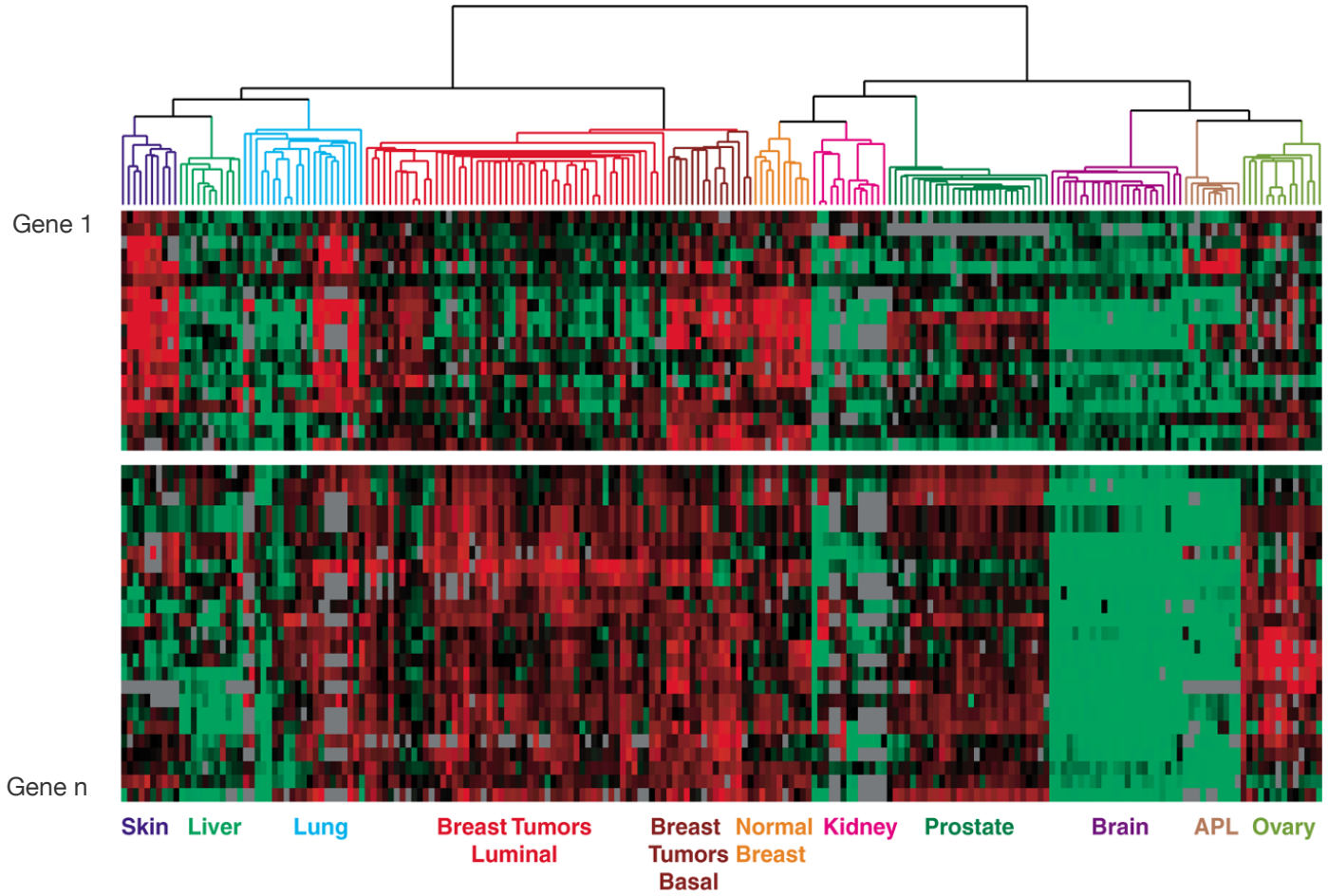


Hierarchical clustering



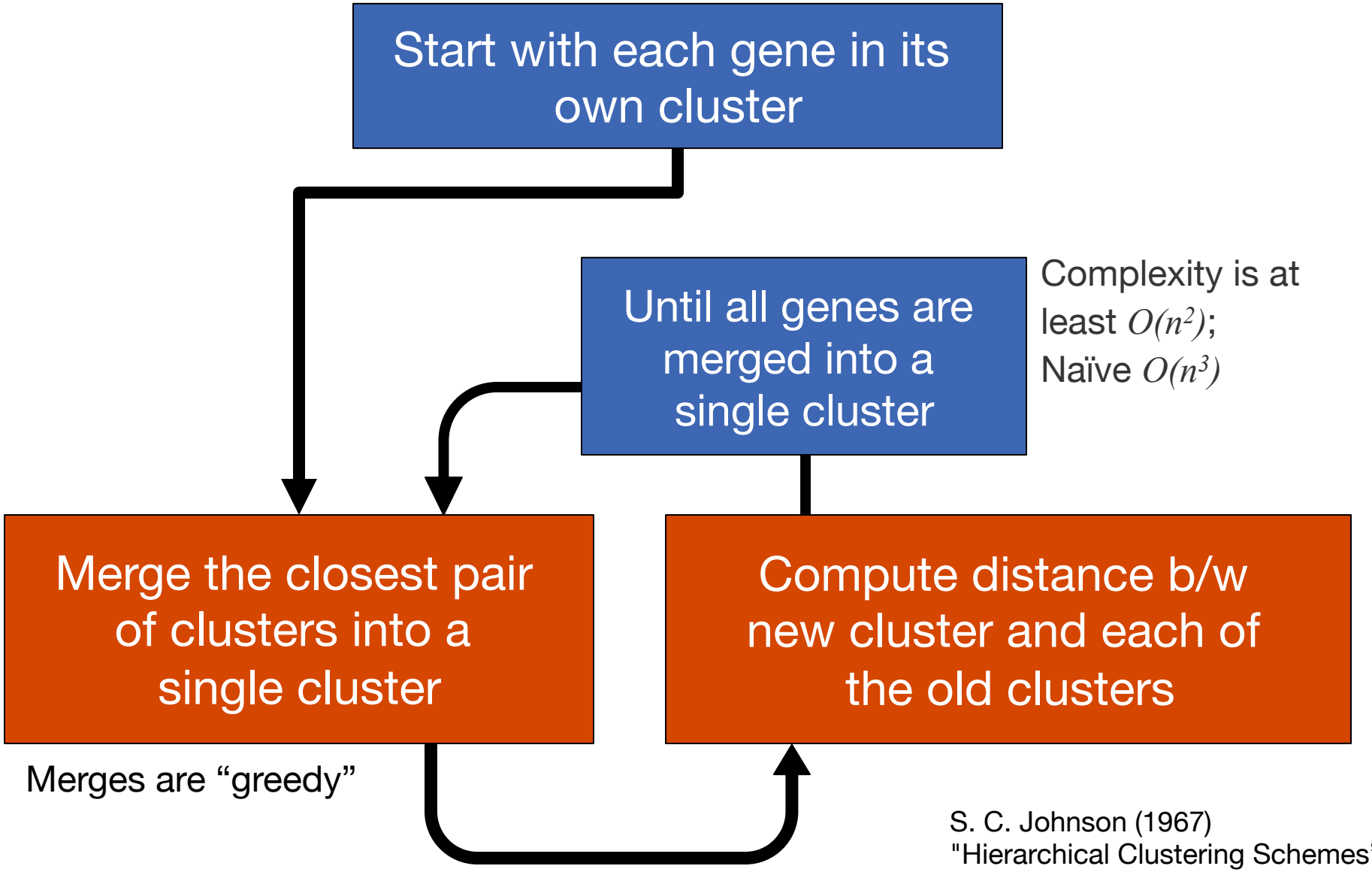
Hierarchical clustering

Tumors in similar tissues cluster together



Reference: Botstein & Brown group

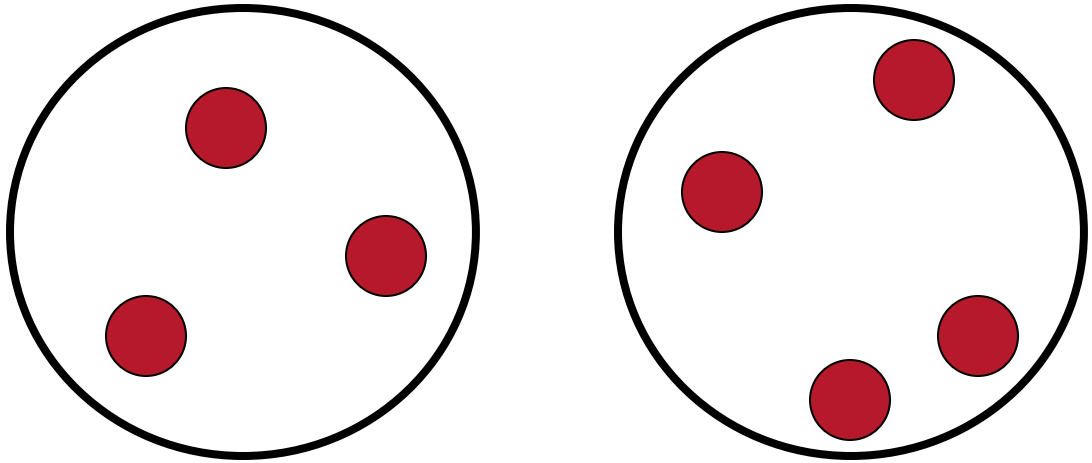
Hierarchical clustering



S. C. Johnson (1967)
"Hierarchical Clustering Schemes”
Psychometrika, 2:241-254

Hierarchical clustering

- Distance metric
- Linkage criteria
  - Single/Minimum linkage
  - Complete/Maximum linkage
  - Average linkage



Single-Link Hierarchical Clustering

Input. Pre-computed matrix of distances between all pairs of genes (i, j).

Scheme. Erase rows and columns in the distance matrix as old clusters are merged into new ones.

Begin.

Each gene in its own cluster.

For each cluster i, create/maintain index dmin[i] of closest cluster.

	dmin	dist
0	1	5.5
1	3	2.14
2	4	5.6
3	1	2.14
4	3	5.5

	0	1	2	3	4
gene0	-	5.5	7.3	8.9	5.8
1	5.5	-	6.1	2.14	5.6
2	7.3	6.1	-	7.8	5.6
3	8.9	2.14	7.8	-	5.5
4	5.8	5.6	5.6	5.5	-

Single-Link Hierarchical Clustering

Iteration.

- Find closest pair of clusters (i1, i2).
- Replace row i1 by min of row i1 and row i2.
- Infinity out row i2 and column i2.
- Update dmin[i] and change dmin[i'] to i1 if previously dmin[i'] = i2.

- Merging into a cluster
- Updating matrix with dist between new cluster & old clusters

Closest pair

	dmin	dist
0	1	5.5
1	3	2.14
2	4	5.6
3	1	2.14
4	3	5.5

	0	1	2	3	4
gene0	-	5.5	7.3	8.9	5.8
1	5.5	-	6.1	2.14	5.6
2	7.3	6.1	-	7.8	5.6
3	8.9	2.14	7.8	-	5.5
4	5.8	5.6	5.6	5.5	-

gene1 closest to gene3, dist = 2.14

	dmin	dist
0	1	5.5
1	0	5.5
2	4	5.6
3	-	-
4	1	5.5

	0	1	2	3	4
0	-	5.5	7.3	-	5.8
node1	5.5	-	6.1	-	5.5
2	7.3	6.1	-	-	5.6
3	-	-	-	-	-
4	5.8	5.5	5.6	-	-

New min distance

Single-Link Clustering: Main Loop

```
for (int s = 0; s < ? ; s++) {
```

Find closest pair of clusters (i1, i2).

Replace row i1 by min of row i1 and row i2.

Infinity out row i2 and column i2.

Update dmin[i]
Change dmin[i'] to i1 if previously dmin[i'] = i2.

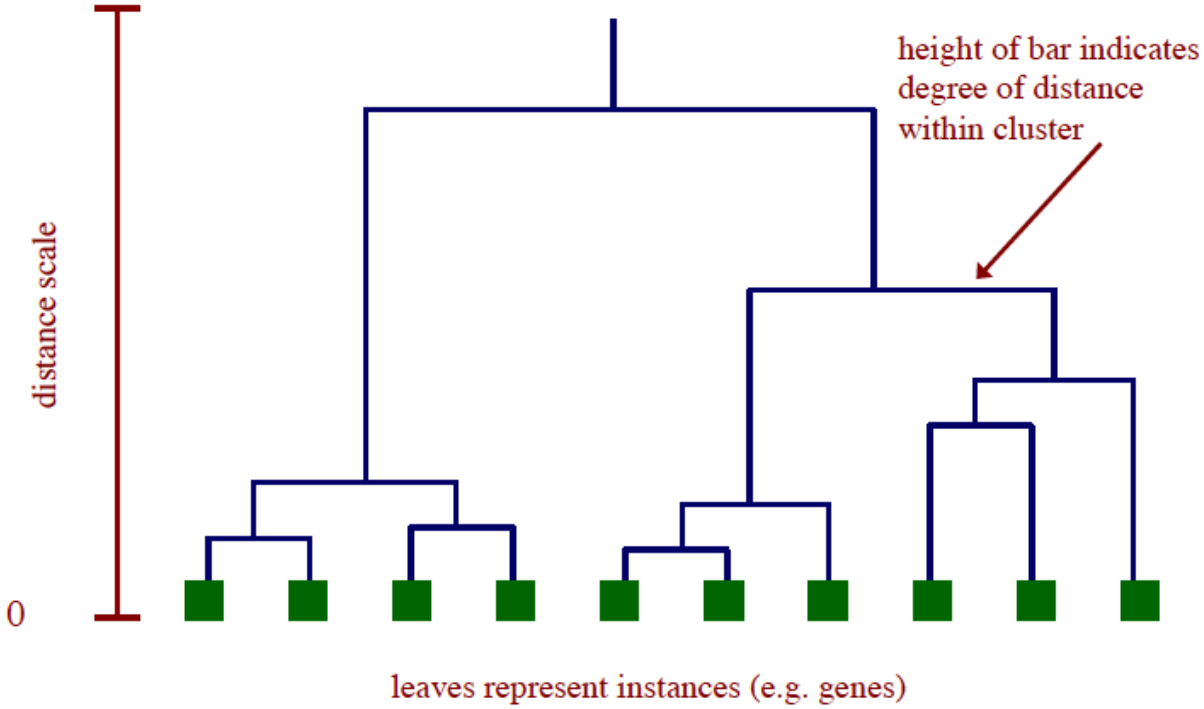
```
}
```

Single-Link Clustering: Main Loop

```
for (int s = 0; s < N-1; s++) {
   // find closest pair of clusters (i1, i2)
   int i1 = 0;
   for (int i = 0; i < N; i++)
      if (d[i][dmin[i]] < d[i1][dmin[i1]]) i1 = i;
   int i2 = dmin[i1];

   // overwrite row i1 with minimum of entries in row i1 and i2
   for (int j = 0; j < N; j++)
      if (d[i2][j] < d[i1][j]) d[i1][j] = d[j][i1] = d[i2][j];
   d[i1][i1] = INFINITY;

   // infinity-out old row i2 and column i2
   for (int i = 0; i < N; i++)
      d[i2][i] = d[i][i2] = INFINITY;

   // update dmin and replace ones that previous pointed to
   // i2 to point to i1
   for (int j = 0; j < N; j++) {
      if (dmin[j] == i2) dmin[j] = i1;
      if (d[i1][j] < d[i1][dmin[i1]]) dmin[i1] = j;
   }
}
```

Dendrogram

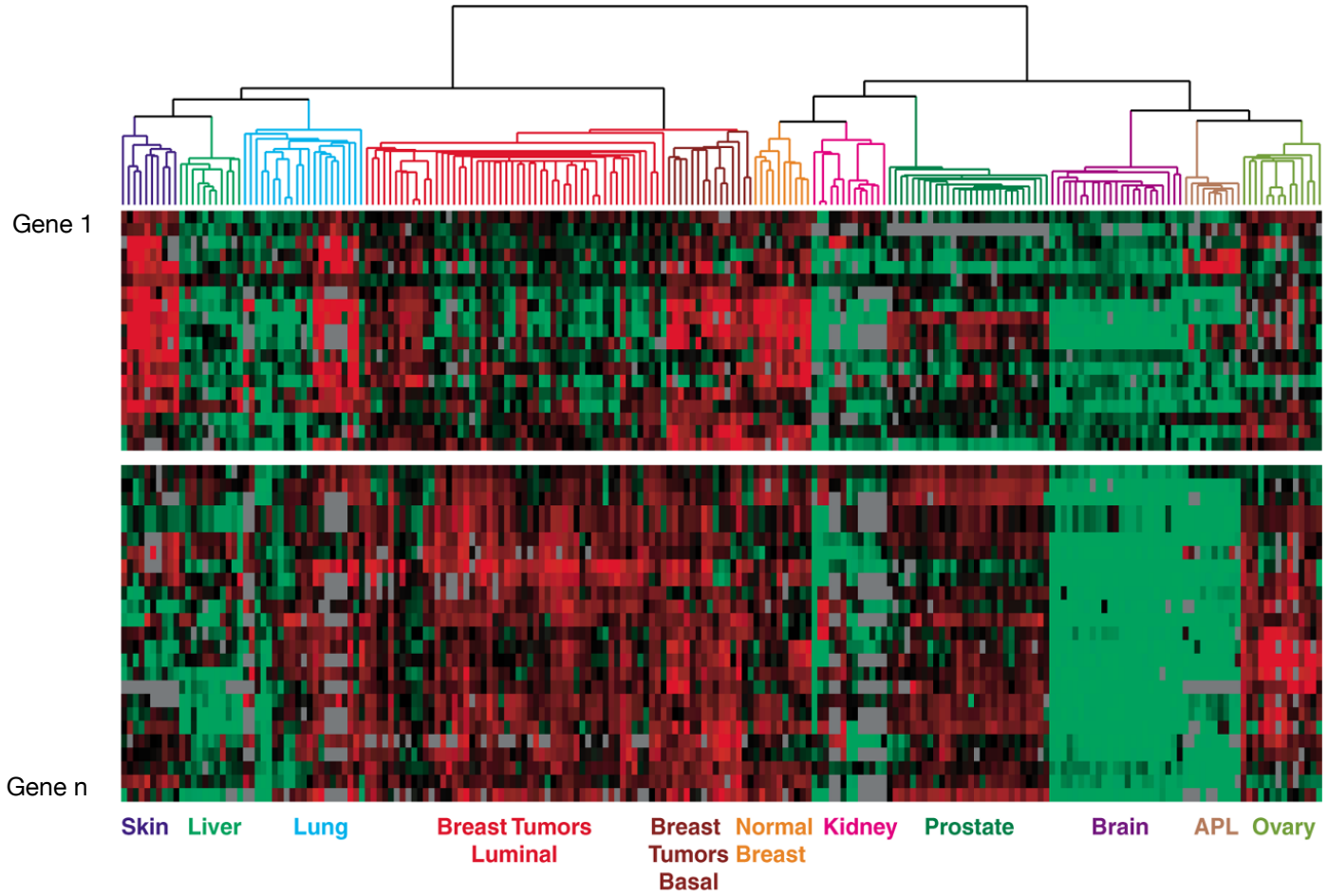
- Dendrogram: Scientific visualization of hypothetical sequence of evolutionary events.
  - Leaves = genes.
  - Internal nodes = hypothetical ancestors.



Reference: http://www.biostat.wisc.edu/bmi576/fall-2003/lecture13.pdf

Dendrogram of Human tumors

Tumors in similar tissues cluster together.



Reference: Botstein & Brown group

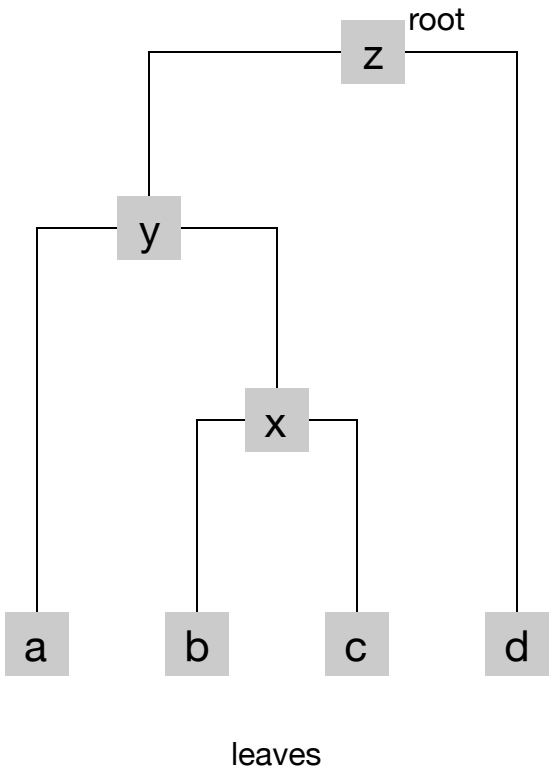
Ancestor Tree

Root. Node with no parent.

Leaf. Node with no children.

Depth. Length of path from node to root.

Least Common Ancestor. Common ancestor with largest depth.



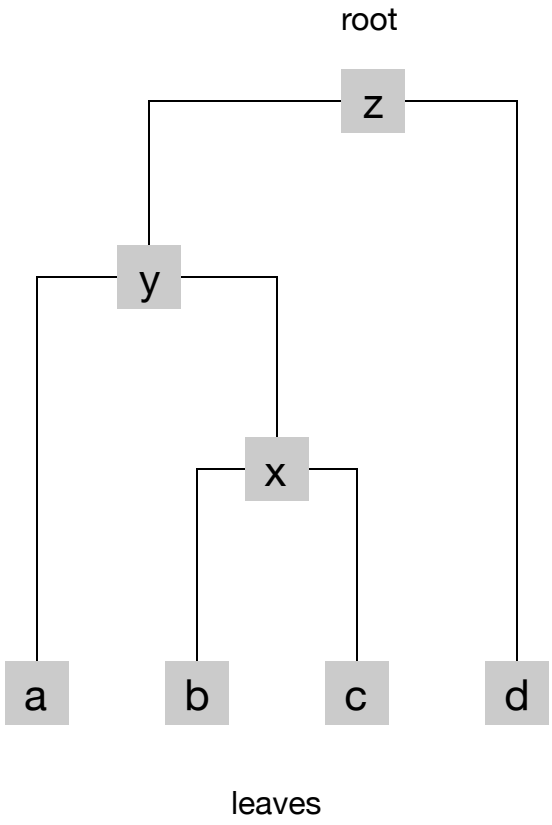

```
public static void main(String[] args) {
   TreeNode a = new TreeNode("GENE1");
   TreeNode b = new TreeNode("GENE2");
   TreeNode c = new TreeNode("GENE3");
   TreeNode d = new TreeNode("GENE4");

   TreeNode x = new TreeNode("NODE1", b, c);
   TreeNode y = new TreeNode("NODE2", a, x);
   TreeNode z = new TreeNode("NODE3", d, y);

   System.out.println(a.lca(b));
   a.lca(b).showLeaves();
}
```

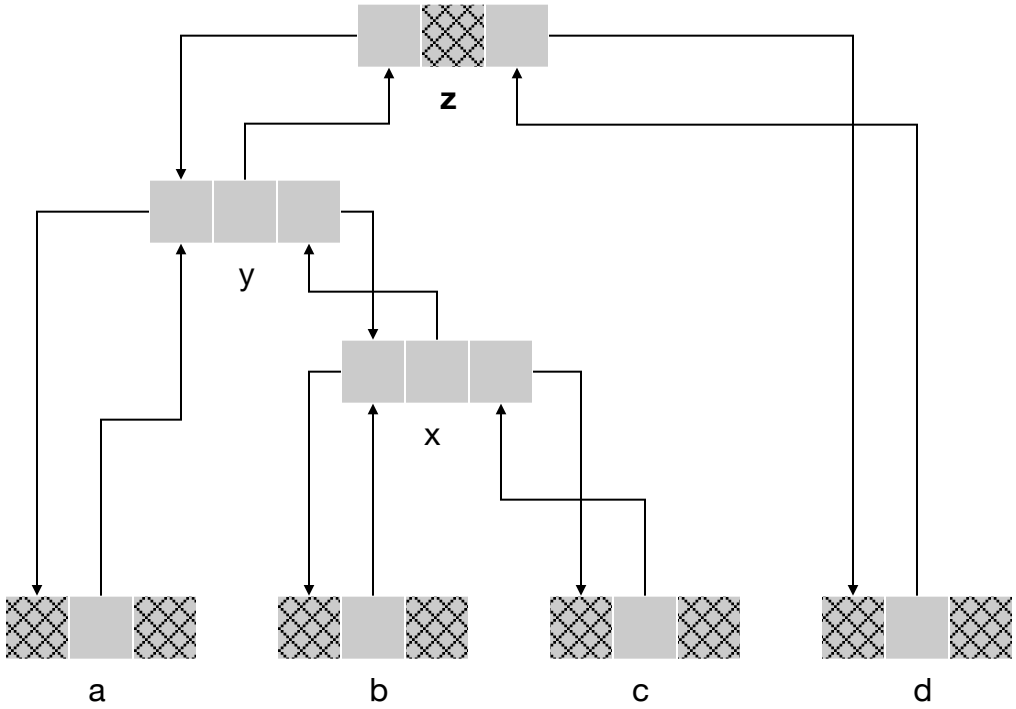
lca(a, b) = y, leaves of y = { a, b, c }

Ancestor Tree: Implementation

How would you represent a node in java to be able to find LCAs or Children?

Hint: think pointers!

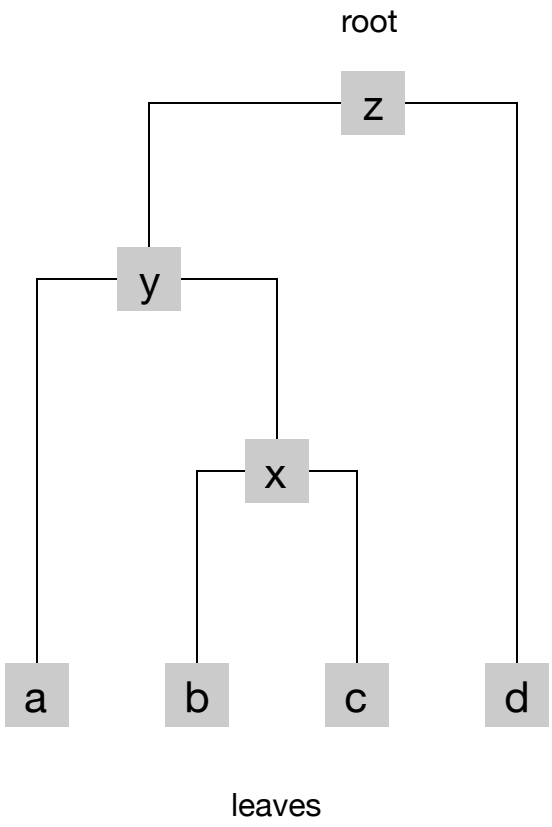


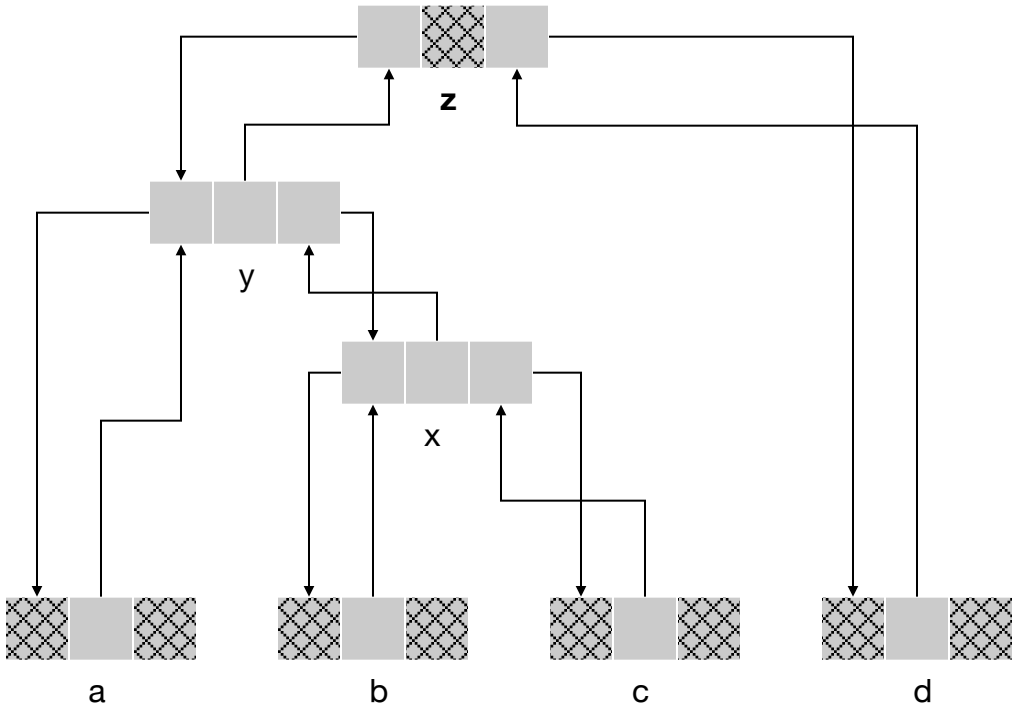


Ancestor Tree: Implementation

Node. Left pointer, right pointer, parent pointer.

Consequence. Can go up or down the tree.





Ancestor Tree

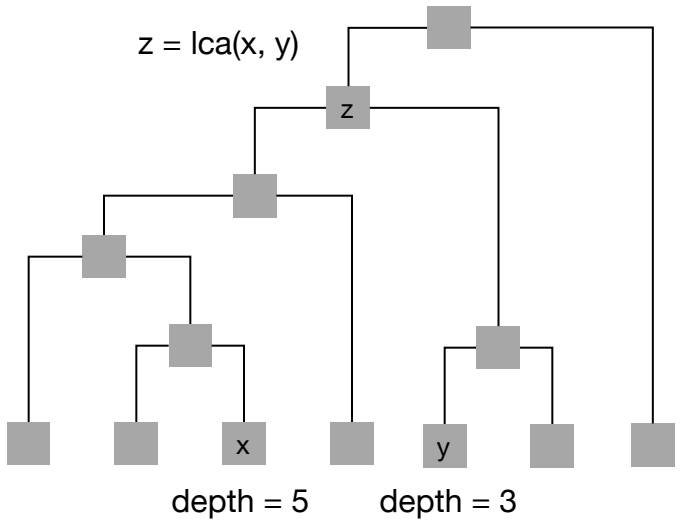
```
public class TreeNode {
    private TreeNode parent;          // parent
    private TreeNode left, right;     // two children
    private String name;              // name of node

    // create a leaf node
    public TreeNode(String name) {
        this.name = name;
    }

    // create an internal node that is the parent of x and y
    public TreeNode(String name, TreeNode x, TreeNode y) {
        this.name  = name;
        this.left  = x;
        this.right = y;
        x.parent = this;
        y.parent = this;
    }
```

Ancestor Tree: Helper Functions

```
// return depth of this node in the tree
// depth of root = 0
public int depth() {
   int depth = 0;
   for (TreeNode x = this; x.parent != null; x = x.parent)
      depth++;
   return depth;
}

// return root
public TreeNode root() {
   TreeNode x = this;
   while (x.parent != null)
      x = x.parent;
   return x;
}
```

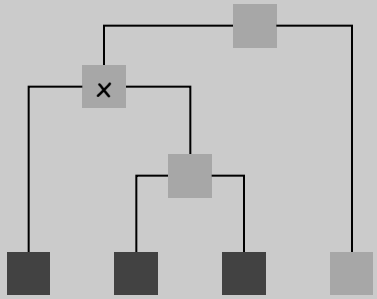
Ancestor Tree: Least Common Ancestor

```
// return the lca of node x and y
public TreeNode lca(TreeNode y) {
        TreeNode x = this;
        int dx = x.depth();
        int dy = y.depth();
        if (dx < dy) {
            for (int i = 0; i < dy-dx; i++) y = y.parent;
        }
        else {
            for (int i = 0; i < dx-dy; i++) x = x.parent;
        }
        while (x != y) {
            x = x.parent;
            y = y.parent;
        }
        return x;
    }
```



Ancestor Tree: Tree Traversal

```
// return string representation
public String toString() { return name; }

// print all leaves in tree rooted at this node
public void showLeaves() {
   if (left == null && right == null) System.out.println(this);
   else {
      left.showLeaves();
      right.showLeaves();
   }
}

// print the tree rooted at this node
public void show() {
   if (left == null && right == null) return;
   System.out.println(name + " " + left.name + " " + right.name);
   left.show();
   right.show();
}
```



Hierarchical clustering implementation

Vector Data Type

Vector data type.

- Set of values: sequence of N real numbers.
- Set of operations: distanceTo, scale, plus.

Ex. p = (1, 2, 3, 4), q = (5, 2, 4, 1).

- dist(p, q) = sqrt($4^2 + 0^2 + 1^2 + 3^2$) = 5.099.
- t = 1/4 p + 3/4 q = (4, 2, 3.75, 1.75).

```
public static void main(String[] args) {
   double[] pdata = { 1.0, 2.0, 3.0, 4.0 };
   double[] qdata = { 5.0, 2.0, 4.0, 1.0 };
   Vector p = new Vector(pdata);
   Vector q = new Vector(qdata);
   double dist = p.distanceTo(q);
   Vector r = p.scale(1.0/4.0);
   Vector s = q.scale(3.0/4.0);
   Vector t = r.plus(s);
   System.out.println(t);
}
```

Vector: Array Implementation

```
public class Vector {
   private int N;           // dimension
   private double[] data;   // components

   // create a vector from the array d
   public Vector(double[] d) {
      N = d.length;
      data = d;
   }

   // return Euclidean distance from this vector a to b
   public double distanceTo(Vector b) {
      Vector a = this;
      double sum = 0.0;
      for (int i = 0; i < N; i++)
         sum += (a.data[i] - b.data[i])*(a.data[i] - b.data[i]);
      return Math.sqrt(sum);
   }
```

Vector: Array Implementation

```
public String toString() {
   String s = "";
   for (int i = 0; i < N; i++)
      s = s + data[i] + " ";
   return s;
}
```

return string representation of this vector

```
public Vector plus(Vector b) {
   Vector a = this;
   double[] d = new double[N];
   for (int i = 0; i < N; i++)
      d[i] = a.data[i] + b.data[i];
   return new Vector(d);
}
```

return vector sum of this vector a and vector b

(could add error checking to ensure dimensions agree)

Single-Link Hierarchical Clustering

Iteration.

- Closest pair of clusters (i, j) is one with the smallest dist value.
- Replace row i by min of row i and row j.
- Infinity out row j and column j.
- Update dmin[i] and change dmin[i'] to i if previously dmin[i'] = j.

Closest pair

	dmin	dist
0	1	5.5
1	3	2.14
2	4	5.6
3	1	2.14
4	3	5.5

	0	1	2	3	4
gene0	-	5.5	7.3	8.9	5.8
1	5.5	-	6.1	2.14	5.6
2	7.3	6.1	-	7.8	5.6
3	8.9	2.14	7.8	-	5.5
4	5.8	5.6	5.6	5.5	-

gene1 closest to gene3, dist = 2.14

---

	dmin	dist
0	1	5.5
1	0	5.5
2	4	5.6
3	-	-
4	1	5.5

	0	1	2	3	4
0	-	5.5	7.3	-	5.8
node1	5.5	-	6.1	-	5.5
2	7.3	6.1	-	-	5.6
3	-	-	-	-	-
4	5.8	5.5	5.6	-	-

New min dist

Single-Link Clustering: Java Implementation

Single-link clustering.

- Read in the data.

```
public static void main(String[] args) {
   int M = StdIn.readInt();
   int N = StdIn.readInt();

   // read in N vectors of dimension M
   Vector[] vectors = new Vector[N];
   String[] names   = new String[N];
   for (int i = 0; i < N; i++) {
   names[i] = StdIn.readString();
   double[] d = new double[M];
   for (int j = 0; j < M; j++)
      d[j] = StdIn.readDouble();
   vectors[i] = new Vector(d);
   }
```

Single-Link Clustering: Java Implementation

Single-link clustering.

- Read in the data.
- Precompute d[i][j] = distance between cluster i and j.
- For each cluster i, maintain index dmin[i] of closest cluster.

```
double INFINITY = Double.POSITIVE_INFINITY;
double[][] d = new double[N][N];
int[] dmin = new int[N];
for (int i = 0; i < N; i++) {
   for (int j = 0; j < N; j++) {
      if (i == j) d[i][j] = INFINITY;
      else        d[i][j] = vectors[i].distanceTo(vectors[j]);
      if (d[i][j] < d[i][dmin[i]]) dmin[i] = j;
   }
}
```

Single-Link Clustering: Main Loop

```
for (int s = 0; s < N-1; s++) {
   // find closest pair of clusters (i1, i2)
   int i1 = 0;
   for (int i = 0; i < N; i++)
      if (d[i][dmin[i]] < d[i1][dmin[i1]]) i1 = i;
   int i2 = dmin[i1];

   // overwrite row i1 with minimum of entries in row i1 and i2
   for (int j = 0; j < N; j++)
      if (d[i2][j] < d[i1][j]) d[i1][j] = d[j][i1] = d[i2][j];
   d[i1][i1] = INFINITY;

   // infinity-out old row i2 and column i2
   for (int i = 0; i < N; i++)
      d[i2][i] = d[i][i2] = INFINITY;

   // update dmin and replace ones that previous pointed to
   // i2 to point to i1
   for (int j = 0; j < N; j++) {
      if (dmin[j] == i2) dmin[j] = i1;
      if (d[i1][j] < d[i1][dmin[i1]]) dmin[i1] = j;
   }
}
```

Store Centroids in Each Internal Node

Cluster analysis.

Centroids distance / similarity.

Easy modification to TreeNode data structure.

- Store Vector in each node.
  - leaf nodes: directly corresponds to a gene
  - internal nodes: centroid = average of all leaf nodes beneath it
- Maintain count field in each TreeNode, which equals the number of leaf nodes beneath it.
- When setting z to be parent of x and y,
  - set z.count = x.count + y.count
  - set z.vector = αp + (1-α)q, where p = x.vector and q = y.vector, and α = x.count / z.count

Analysis and Micro-Optimizations

Running time. Proportional to MN^2 (N genes, M arrays)
Memory. Proportional to N^2.

Ex. [M = 50, N = 6,000] Takes 280MB, 48 sec on fast PC.

← input size proportional to MN

Some optimizations.

- Use float instead of double
- Store only lower triangular part of distance matrix
- Use squares of distances instead of distances.

How much do you think would this help?

The End