

# Network Analysis and Visualization

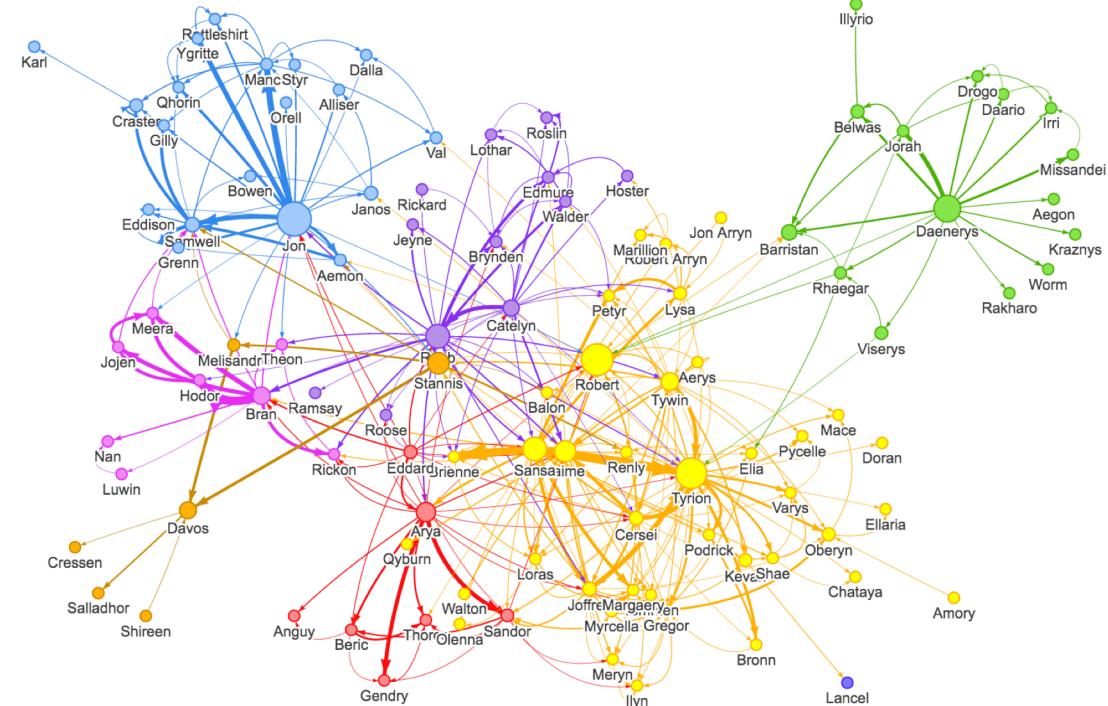
Chun Su

# Agenda

- Network analysis basics
  - R object IGRAPH "I/O"
  - object IGRAPH manipulation
- Graph advance
  - Graph measurement
  - Graph cluster
- Network visualization
  - Aesthetics of network elements
  - Network layout
- Network application show case
- Group Exercise – NYC flight

# What is network analysis?

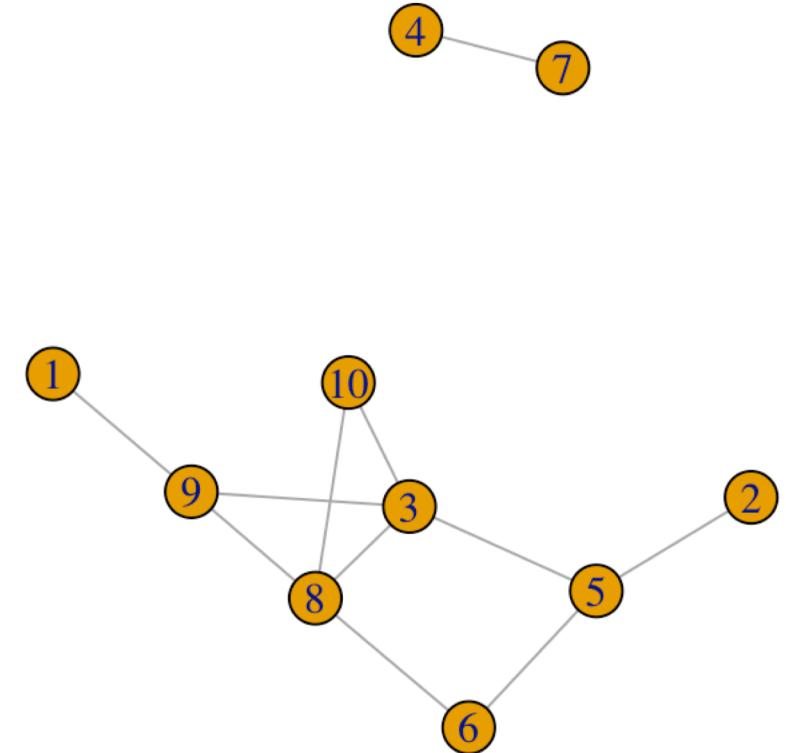
- Network analysis is to study the **inter-relationships** between actors of all sorts and provides an **architectural view** of individual connections. In mathematics, it is part of **graph theory**.
- Network analysis are widely used in
  - Computer science . eg. computational devices connection
  - Linguistics . eg. natural language flow
  - Social sciences. eg. social network
  - Biology. eg. gene regulation network



<https://www.lyonwj.com/2016/06/26/graph-of-thrones-neo4j-social-network-analysis/>

# Graph elements (Glossaries I)

- **Vertex**: is the node of network
- **Edge**: the connection of notes, sometimes called "link".
  - Directed
  - undirected
- **Graph**: constitute of vertices and edges, which represents the whole inter-relationship of nodes.



# IGRAPH object in R

- Package `igraph` have a class ‘`igraph`’.

```
g = make_graph(letters[1:10], directed = T)  
g
```

Named	Edge #
Directed	vertex #
## IGRAPH 8b3c740 DN--	10 5 --
## + attr: name (v/c)	Attribute can be associated with either vertex (v) or edge (e) or both.
## + edges from 8b3c740 (vertex names):	Special attributes, name, can be used to select vertices and edges ("from to").
## [1] a->b c->d e->f g->h i->j	

# Create IGRAPH

- IGRAPH can be created in multiple ways
  - Easy graph:
    - `graph_from_literal`
    - `make_graph`
  - Create from user data:
    - `graph_from_edgelist`
    - `graph_from_adjacency_matrix`
    - `graph_from_data_frame`
  - Classic random graphs:
    - `sample_gnp`
    - `sample_gnm`
    - `sample_pa`
    - `sample_smallworld`

# graph\_from\_adjacency\_matrix

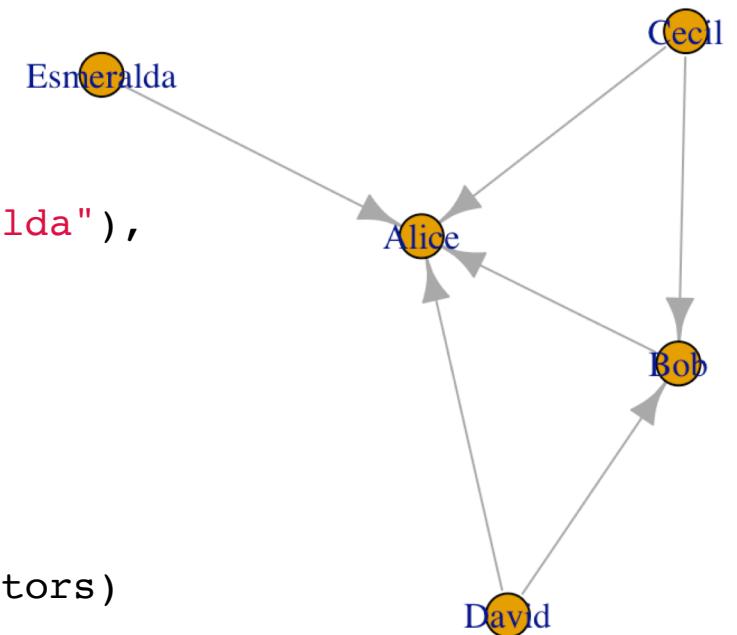
	a	b	c	d	e	f	g	h	i	j
a	0	1	1	1	0	0	0	0	0	0
b	0	0	0	1	0	0	0	0	0	0
c	0	0	0	0	0	0	0	0	0	0
d	0	0	1	0	0	0	0	0	0	0
e	1	0	0	0	0	0	0	0	0	0
f	0	1	0	0	0	0	0	0	0	0
g	0	0	0	0	0	0	0	1	0	0
h	1	0	0	0	0	0	0	1	0	0
i	0	0	0	0	1	0	0	0	0	0
j	0	1	0	0	0	0	0	0	0	0

```
g = graph_from_adjacency_matrix(adj_matrix,
mode = "undirected", weighted = T)
g
```

```
## IGRAPH 48cf731 UNW- 10 12 --
## + attr: name (v/c), weight (e/n)
## + edges from 48cf731 (vertex names):
## [1] a--b a--c a--d a--e a--h b--d b--f b--j c--d f--i g--i h--i
```

# graph\_from\_data\_frame

```
actors <- data.frame(  
  name=c("Alice", "Bob", "Cecil", "David", "Esmeralda"),  
  age=c(48, 33, 45, 34, 21),  
  gender=c("F", "M", "F", "M", "F"))  
)  
relations <- data.frame(  
  from=c("Bob", "Cecil", "Cecil", "David", "David", "Esmeralda"),  
  to=c("Alice", "Bob", "Alice", "Alice", "Bob", "Alice"),  
  same.dept=c(FALSE, FALSE, TRUE, FALSE, FALSE, TRUE),  
  friendship=c(4, 5, 5, 2, 1, 1),  
  advice=c(4, 5, 5, 4, 2, 3))  
)  
g <- graph_from_data_frame(relations, directed=TRUE, vertices=actors)  
g  
  
## IGRAPH 41a8cf9 DN-- 5 6 --  
## + attr: name (v/c), age (v/n), gender (v/c), same.dept (e/l),  
## | friendship (e/n), advice (e/n)  
## + edges from 41a8cf9 (vertex names):  
## [1] Bob      ->Alice Cecil    ->Bob    Cecil     ->Alice David     ->Alice  
## [5] David    ->Bob    Esmeralda->Alice
```



# Retrieve vertex and Edges from IGRAPH

- Vertex and Edges are saved as "igraph.vs" and "igraph.es" object respectively. Can be retrieved using V(g) and E(g).

```
V(g)$name  
V(g)$age  
V(g)$gender  
  
# [1] "Alice" "Bob" "Cecil" "David" "Esmeralda"  
# [1] 48 33 45 34 21  
# [1] "F" "M" "F" "M" "F"
```

```
E(g)$same.dept  
E(g)$friendship  
  
# [1] FALSE FALSE TRUE FALSE FALSE TRUE  
# [1] 4 5 5 2 1 1
```

# Retrieve vertex and Edges from IGRAPH

```
as_data_frame(g, what = "vertices")
```

```
##          name age gender
## Alice      Alice  48     F
## Bob        Bob   33     M
## Cecil     Cecil  45     F
## David     David  34     M
## Esmeralda Esmeralda 21     F
```

```
as_data_frame(g, what = "edges")
```

##	from	to	same.dept	friendship	advice
## 1	Alice	Bob	FALSE	4	4
## 2	Bob	David	FALSE	5	5
## 3	Cecil	Alice	TRUE	5	5
## 4	David	Alice	FALSE	2	4
## 5	David	Bob	FALSE	1	2
## 6	Esmeralda	Alice	TRUE	1	3

# Add vertex and edges to IGRAPH

```
# add vertices
g %>% add_vertices(2, name=c("Lisa", "Zack")) # the first argument is
# number of vertex
g + vertices(c("Lisa", "Zack"))

# add connected edges (even number of vertices). The vertices
# must be from known vertices already in the graph
g %>% add_edges(c("Alice", "Bob"))
g + edge(c("Alice", "Bob"))

# add paths. The vertices must be from known vertices already in the graph
g + path("Alice", "Bob", "Cecil")

# If a new vertex needs to be added to current graph, using add graph method
# instead
g + make_graph(c("Alice", "Bob", "Bob", "Melisa"))
```

# Delete vertex and edges from IGGRAPH

```
### remove the vertices whose age is younger than 30
vertex_df = g %>% as_data_frame(what="vertices") %>% tbl_df %>%
  mutate(index=row_number()) %>% filter(age < 30)
# remove vertices by index number
g %>% delete_vertices(vertex_df$index)
# remove vertice by name
g %>% delete_vertices(vertex_df$name)

### remove the edges with friendship <= 1
edge_df = g %>% as_data_frame(what="edges") %>% tbl_df %>%
  mutate(index=row_number()) %>% mutate(name=paste(from,to,sep=" | "))
%>% filter(friendship <= 1)
#remove vertice by index
g %>% delete_edges(edge_df$index)
#remove vertice by name
g %>% delete_edges(edge_df$name)
```

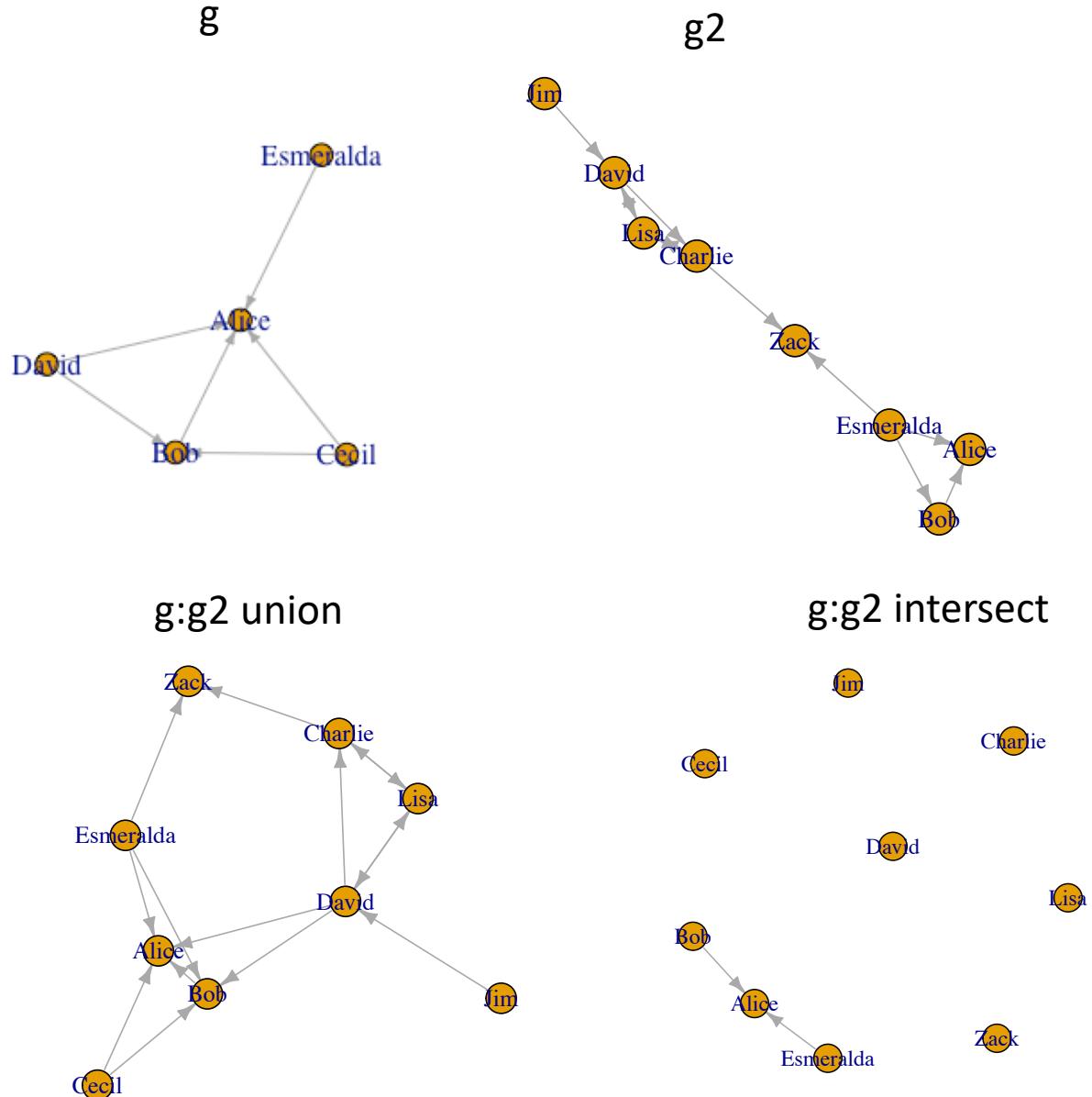
# Add and delete vertex and edge attributes to IGRAPH

- Add/change new edge attributes using `set_edge_attr()`.
- Add/change new vertice attributes using `set_vertex_attr()`
- Delete edge attributes using `delete_edge_attr(g, attr_name)`.
- Delete vertice attributes using `delete_Vertice_attr(g, attr_name, index, attr_value)`

```
# add a new attr "relationship" for people  
in the same dept  
edge_df = g %>%  
as_data_frame(what="edges") %>%  
mutate(  
    relationship=ifelse(same.dept, "colle  
gue", NA)  
)  
g %>%  
set_edge_attr(  
    "relationship",  
    which(!is.na(edge_df$relationship)),  
    edge_df$relationship[!is.na(edge_df$  
relationship)]  
)
```

# Merge Graph

```
#### create new graph g2
g2 = graph_from_literal(
    "David"-+"Charlie"+-+ "Lisa",
    "Lisa"+-+ "David"+- "Jim",
    "Zack"+- "Esmeralda"-+ "Bob",
    "Zack"+- "Charlie",
    "Lisa"+- "Lisa",
    "Bob"-+ "Alice"+- "Esmeralda"
)
#### union graph
g3 = igraph::union(g,g2)
#### graph intersection
g4 = igraph::intersection(g,g2)
```



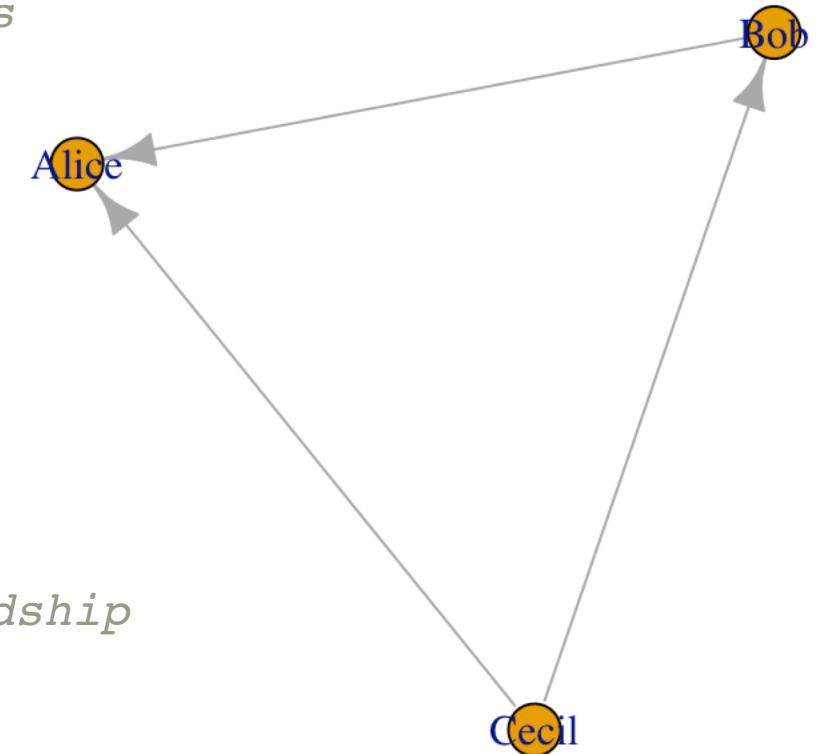
# Subgraph

- Subgraph can be induced by vertice and edge names or igraph.es

```
# induce a subgraph using a list of vertice names  
induced_subgraph(g, v=c("Alice", "Bob", "Cecil"))
```

```
# induce a subgraph using edges  
subgraph.edges(g,  
c("Bob|Alice", "Cecil|Bob", "Cecil|Alice"),  
delete.vertices = TRUE)
```

```
# induce a subgraph using edges attribute (friendship  
score stronger than 3)  
e1 = E(g)[E(g)$friendship > 3]  
subgraph.edges(g, e1, delete.vertices = TRUE)
```



# Agenda

- Network analysis basics
  - R object IGRAPH "I/O"
  - object IGRAPH manipulation
- Graph advance
  - Graph measurement
  - Graph cluster
- Network visualization
  - Aesthetics of network elements
  - Network layout
- Network application show case
- Group Exercise— NYC flight

# Graph measurement – degree and strength

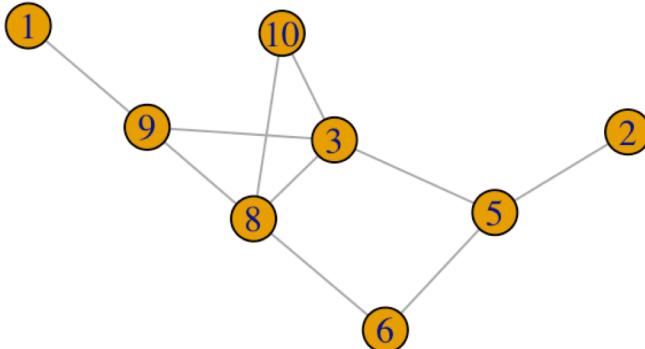
- **Degree:** the number of adjacent vertex.

```
# generate random graph  
set.seed(12)  
gr <- sample_gnp(10, 0.3)  
plot(gr)
```



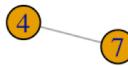
```
# get degree for each node  
degree(gr, v=1:10))  
# [1] 1 1 4 1 3 2 1 4 3 2  
  
# probability for degree  
degree_distribution(g)  
# [1] 0.0 0.4 0.2 0.2 0.2
```

- **Strength:** is weighted version of degree, by summing up the edge weights of the adjacent edges for each vertex



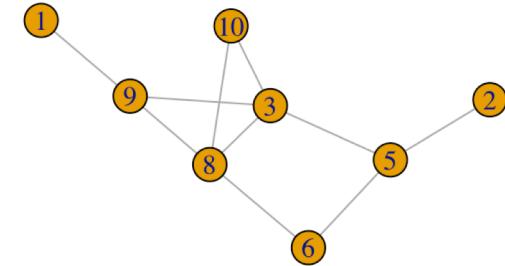
```
# add random weight attribute  
set.seed(12)  
gr = gr %>% set_edge_attr(  
  "weight", index=E(gr),  
  value=sample(seq(0,1,0.05), size=length(E(gr))))  
)  
# calculate strength  
strength(gr, weights = E(gr)$weight)  
# [1] 0.95 0.05 1.90 0.20 1.70 0.85 0.20 1.75 2.10 1.50
```

# Graph measurement – distance and path



- **Order/Distances** : how far away from one vertex to another.

```
# count all edges from 1 to 10, regardless of direction
distances(gr, v=1, to=10, mode="all",
weights = NA)
#      [,1]
# [1,]    3
# pairwise distance table
distances(gr, mode="all")
[,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
[1,]   0   4   2 Inf   3   3 Inf   2   1   3
[2,]   4   0   2 Inf   1   2 Inf   3   3   3
[3,]   2   2   0 Inf   1   2 Inf   1   1   1
[4,] Inf Inf Inf   0 Inf Inf   1 Inf Inf Inf
[5,]   3   1   1 Inf   0   1 Inf   2   2   2
[6,]   3   2   2 Inf   1   0 Inf   1   2   2
[7,] Inf Inf Inf   1 Inf Inf   0 Inf Inf Inf
[8,]   2   3   1 Inf   2   1 Inf   0   1   1
[9,]   1   3   1 Inf   2   2 Inf   1   0   2
[10,]  3   3   1 Inf   2   2 Inf   1   2   0
```



**Path:** a route between any two vertices

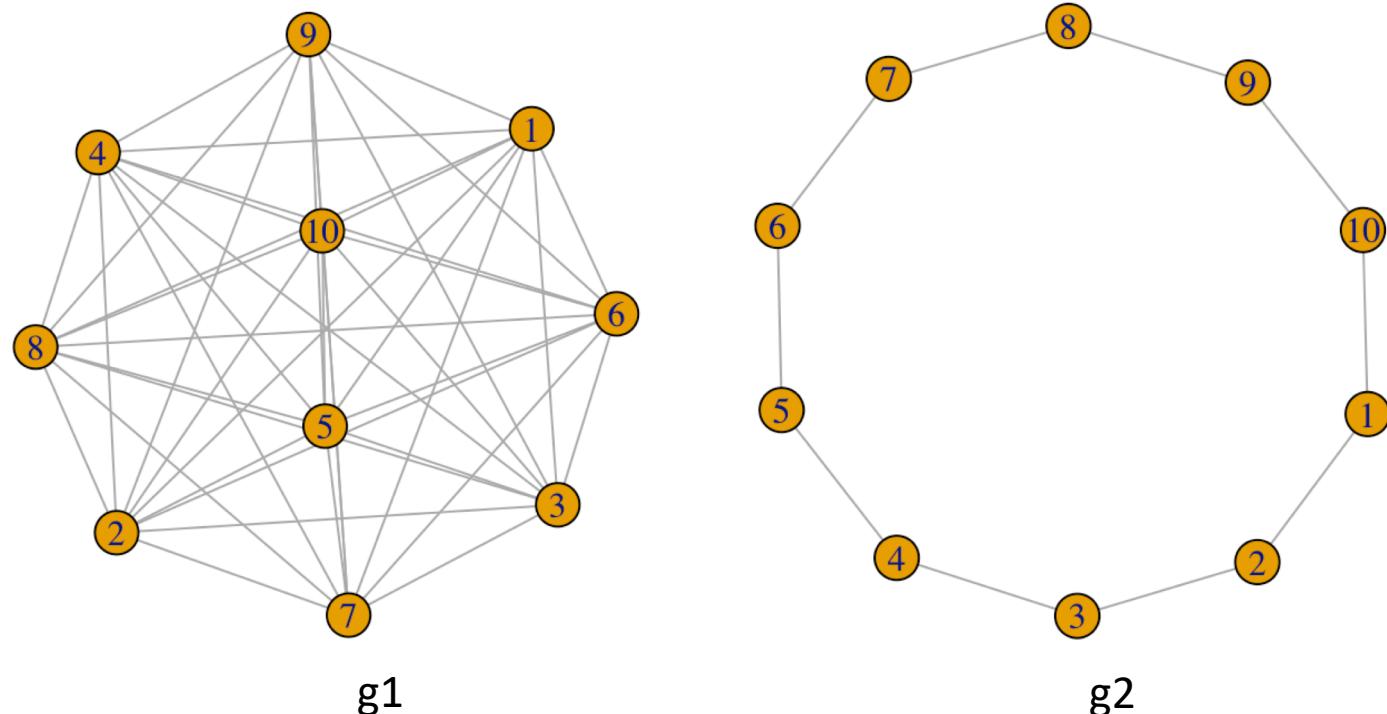
```
# shortest path to connect
all_shortest_paths(gr, 1,10)$res
# $res[[1]]
# + 4/10 vertices, from 82b58b9:
# [1] 1 9 8 10
# $res[[2]]
# + 4/10 vertices, from 82b58b9:
# [1] 1 9 3 10
Both return a list of igraph.vs
# all path to connect
all_simple_paths(gr, 1,10)
```

# Graph measurement – transitivity

- **Transitivity** measures the probability that the adjacent vertices of a vertex are connected. also called the **clustering coefficient**

```
# two extreme classes -- full graph and
# ring graph
g1 = make_full_graph(10)
transitivity(g1)
## [1] 1

g2 = make_ring(10)
transitivity(g2)
## [1] 0
```

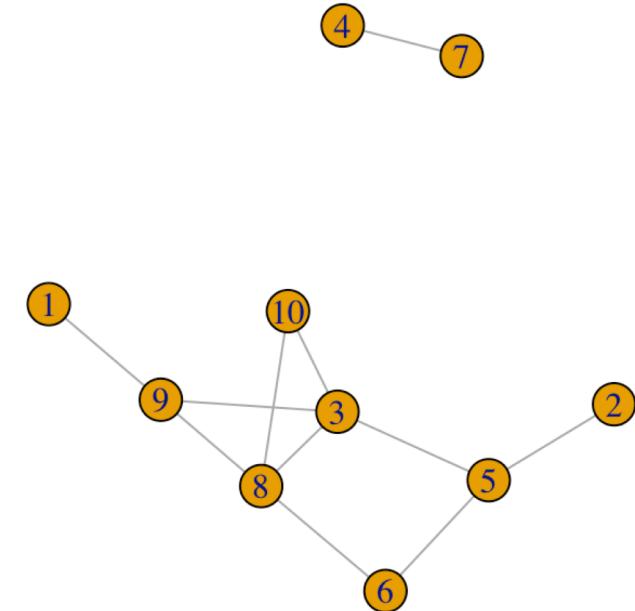


A property very important in social networks,  
and to a lesser degree in other networks

# Graph measurement -- centrality

- **Centrality:** measures the **importance** of a node in a network.
- However, this “importance” can be conceived in two ways:
  - in relation to a type of flow or transfer across the network.
  - involvement in the cohesiveness of the network
- degree centrality (`centr_degree`)
- closeness centrality (`centr_clo`)
- betweenness centrality (`centr_betw`)
- eigenvector centrality (`centr_eigen`)
- .....

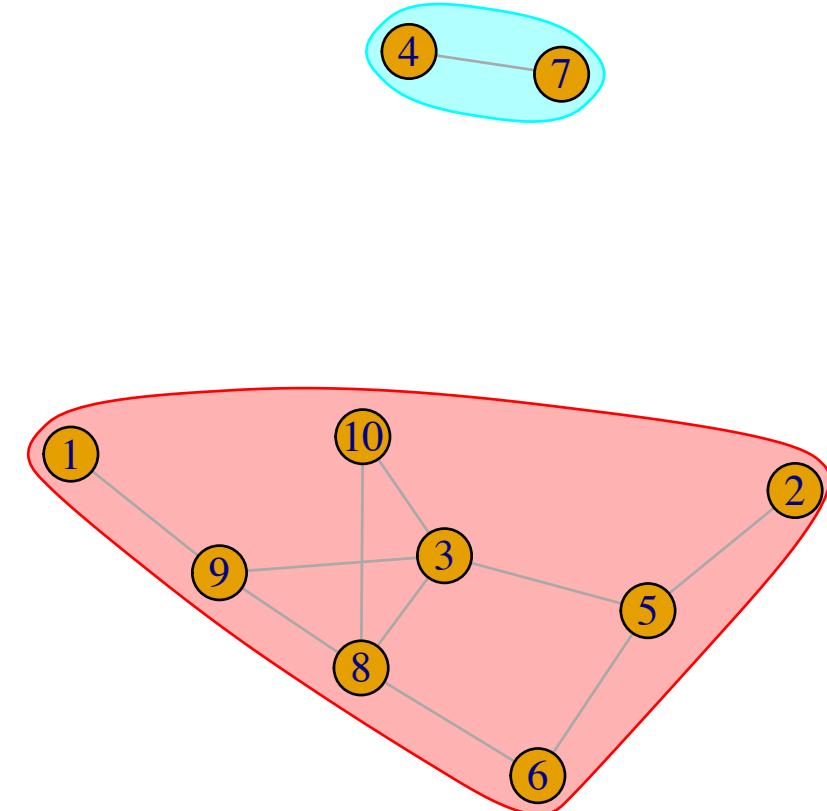
```
centr_betw(gr, directed = FALSE)
## $res
## [1] 0.0 0.0 8.0 0.0 6.5 1.0 0.0 4.5 6.0 0.0
## $centralization
## [1] 0.1666667
## $theoretical_max
## [1] 324
```



# Decompose graph to components

- A **component** is a subgraph in which all nodes are inter-connected.

```
# decompose graph to connected components
dg <- decompose.graph(gr)
dg
## [[1]]
## IGRAPH ec2223e U--- 8 10 -- Erdos renyi (gnp) graph
## + attr: name (g/c), type (g/c), loops (g/l), p (g/n)
## + edges from ec2223e:
## [1] 2--4 3--4 4--5 3--6 5--6 1--7 3--7 6--7 3--8 6--8
##
## [[2]]
## IGRAPH bef9e40 U--- 2 1 -- Erdos renyi (gnp) graph
## + attr: name (g/c), type (g/c), loops (g/l), p (g/n)
## + edge from bef9e40:
## [1] 1--2 ## $membership
##    [1] 1 1 1 2 1 1 2 1 1 1
#summarize components ##
components(gr) ## $csizes
## [1] 8 2
## $no
## [1] 2
```

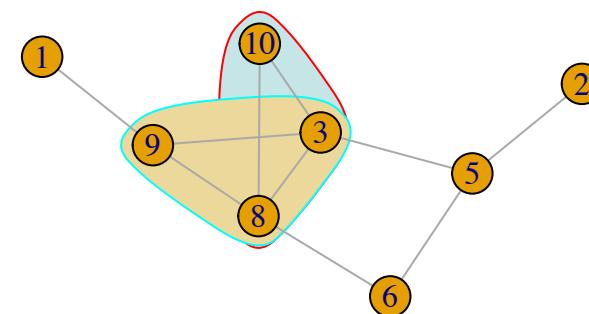
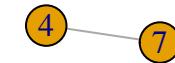


# Graph cluster -- cliques

- **Clique**: fully connected subgraph in which every vertex connects with every other vertex.

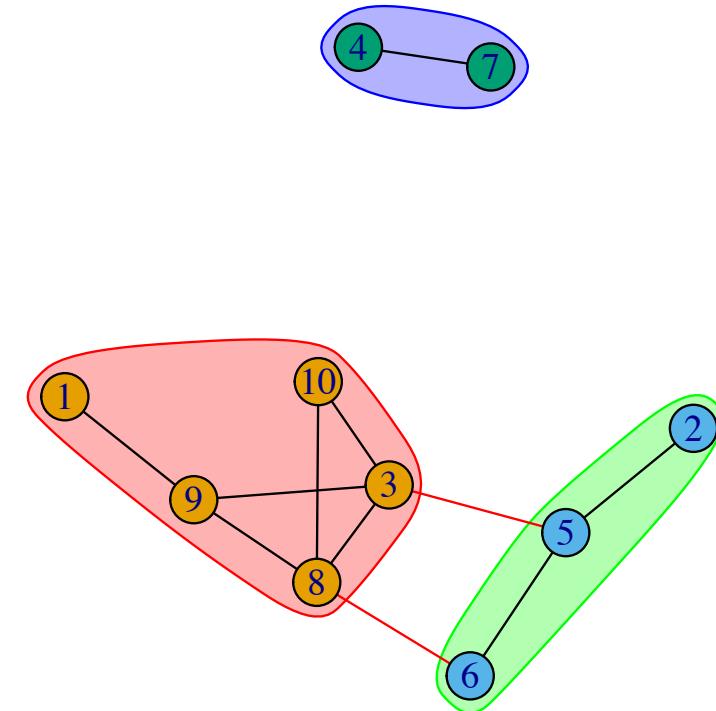
```
# extract cliques that contain more than 3  
vertices  
cliques(g, min=3)
```

```
# get cliques with largest number of vertices
largest_cliques(g)
```



# Graph cluster – community/module

- **Communities:** defined if the nodes of the network can be easily grouped into (potentially overlapping) **sets of nodes** such that each set of nodes is **densely connected internally**
- **Modularity:** measure the strength of division of a network into community, used in optimization methods for detecting community structure in networks



[https://en.wikipedia.org/wiki/Modularity\\_\(networks\)#Modularity](https://en.wikipedia.org/wiki/Modularity_(networks)#Modularity)

# Cluster graph – community/module (cont.)

- There are many ways to cluster graph to communities. The functions in igraph packages include
  - `cluster_edge_betweenness` - a hierarchical decomposition process where edges are removed in the decreasing order of their edge betweenness scores
  - `cluster_optimal` - a top-down hierarchical approach that optimizes the modularity function
  - `cluster_walktrap` - an approach based on random walks
  - `cluster_fast_greedy`
  - `cluster_label_prop`
  - `cluster_leading_eigen`
  - `cluster_Louvain`
  - `cluster_spinglass`

<https://stackoverflow.com/questions/9471906/what-are-the-differences-between-community-detection-algorithms-in-igraph>

[https://en.wikipedia.org/wiki/Community\\_structure](https://en.wikipedia.org/wiki/Community_structure)

# Cluster graph – community/module (cont.)

```
# cluster graph using walktrap method, turn a "communities" object
```

```
wtc <- cluster_walktrap(gr)
```

```
wtc
```

```
IGRAPH clustering walktrap, groups: 3, mod: 0.33
```

```
+ groups:
```

```
$`1`
```

```
[1] 1 3 8 9 10
```

```
$`2`
```

```
[1] 2 5 6
```

```
$`3`
```

```
[1] 4 7
```

```
# find membership for each vertex
```

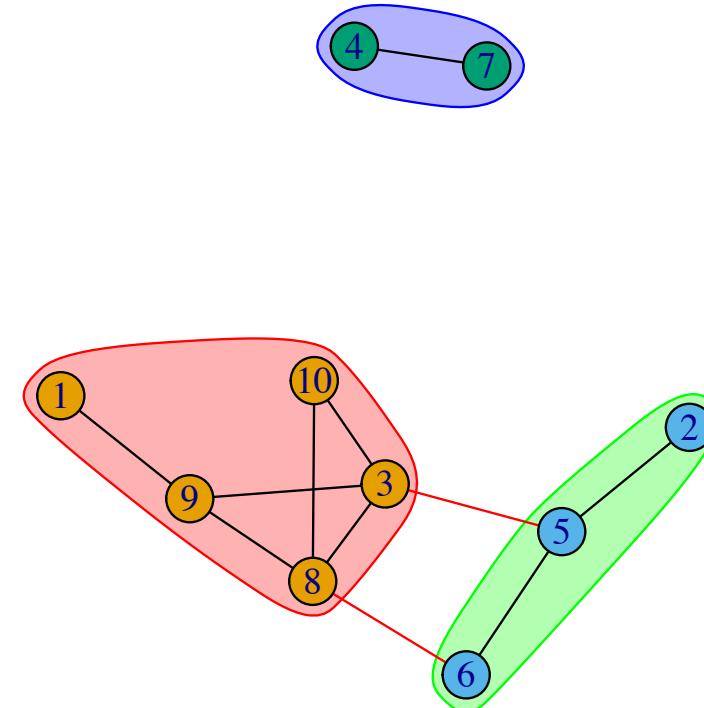
```
membership(wtc)
```

```
## [1] 1 2 1 3 2 2 3 1 1 1
```

```
# calculate modularity for walktrap clustering on this graph
```

```
modularity(wtc)
```

```
## [1] 0.3305785
```



# Agenda

- Network analysis basics
  - R object IGRAPH "I/O"
  - object IGRAPH manipulation
- Graph advance
  - Graph measurement
  - Graph cluster
- **Network visualization**
  - Aesthetics of network elements
  - Network layout
- Network application show case
- Group Exercise— NYC flight

# Aesthetics of network elements

- The aesthetics of both vertices and edges can be manipulated at **color, transparency**.
- Specially for vertices, we can also manipulate its **shape, size** and **fill**.
- For edges, we can manipulate its **width/thickness, linetype, arrow** and so on.

```
actors <- data.frame(  
  name=c("Alice", "Bob", "Cecil", "David", "Esmeralda"),  
  age=c(48, 33, 45, 34, 21),  
  gender=c("F", "M", "F", "M", "F"))  
relations <- data.frame(  
  from=c("Bob", "Cecil", "Cecil", "David", "David", "Esmeralda"),  
  to=c("Alice", "Bob", "Alice", "Alice", "Bob", "Alice"),  
  same.dept=c(FALSE, FALSE, TRUE, FALSE, FALSE, TRUE),  
  friendship=c(4, 5, 5, 2, 1, 1),  
  advice=c(4, 5, 5, 4, 2, 3))  
g <- graph_from_data_frame(relations, directed=TRUE, vertices=actors)
```

# Vertice aesthetics – default igraph.plot

```
# make female and male color different  
v = as_data_frame(g, what="vertice") %>% tbl_df %>%  
mutate(color=case_when(gender=="F" ~ "red", gender=="M"  
~ "blue"))  
g = g %>% set_vertex_attr("color", value=v$color)
```

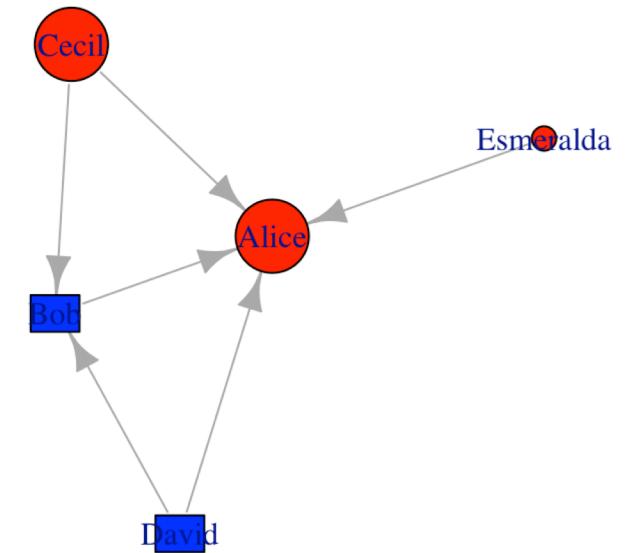


```
# make age as size  
v = v %>% mutate(size=case_when(age < 30 ~ 10, age < 40  
& age > 30 ~ 20, age > 40 ~ 30))  
g = g %>% set_vertex_attr("size", value=v$size)
```

# make gender as shape and put into vertex.shape para.  
In plot

```
v = v %>%  
mutate(shape=case_when(gender=="F" ~ "circle",  
gender=="M" ~ "rectangle"))
```

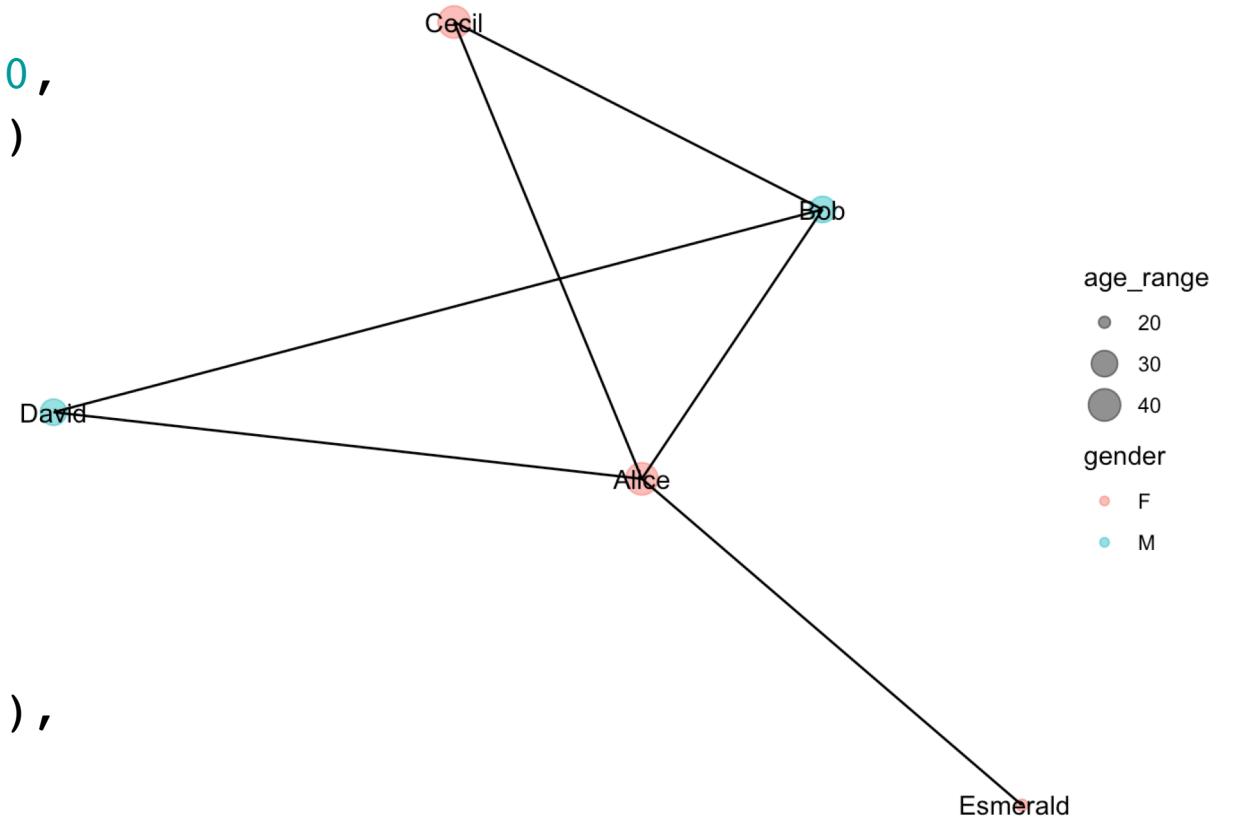
```
# plot  
plot(g, vertex.shape=v$shape)  
legend('topleft', legend=unique(v$gender), pch=c(21,  
22), pt.bg=c("red", "blue"))
```



# Vertice aesthetics – ggraph

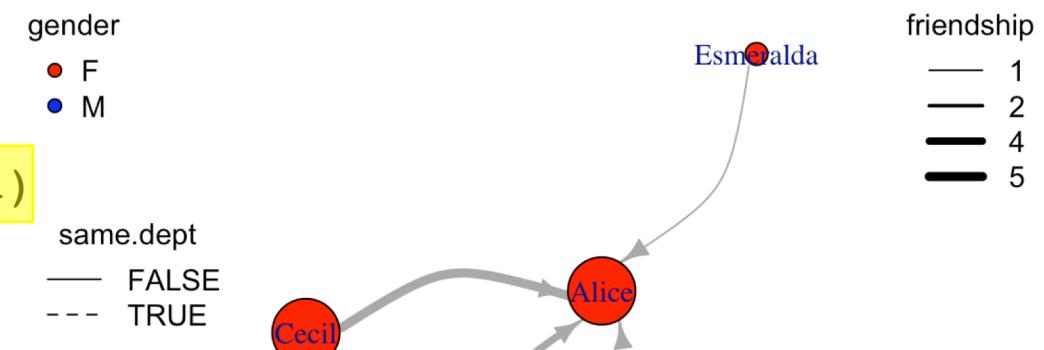
```
# add age_range attr
v = v %>%
  mutate(age_range=case_when(age < 30 ~ 20,
  age < 40 & age > 30 ~ 30, age > 40 ~ 40))
g = g %>% set_vertex_attr("age_range",
  value=v$age_range)

# plot in ggraph
ggraph(g, layout = "kk") +
  geom_node_point(aes(size=age_range,
  color=gender), alpha=0.5) +
  geom_node_text(aes(label=name)) +
  geom_edge_link() +
  scale_size_continuous(breaks=c(20,30,40),
  range = c(2, 6)) + theme_void()
```



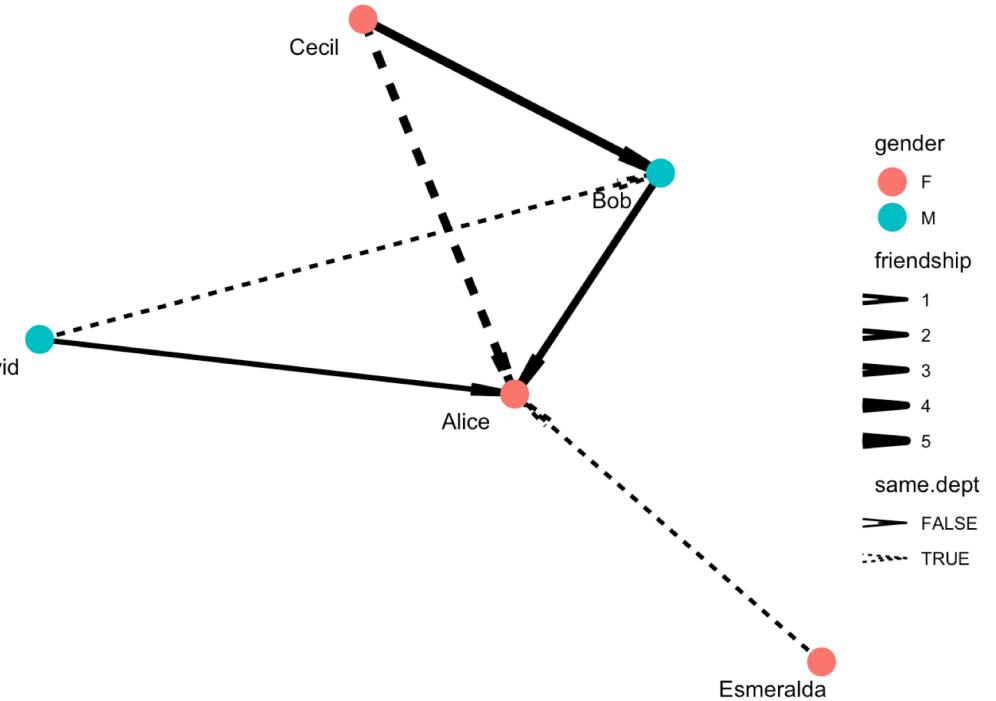
# Edge aesthetics -- default igraph.plot

```
# use linetype present whether come from same  
department, and line width presents friendship  
e = as_data_frame(g, what="edges") %>%tbl_df %>%  
mutate(width=friendship) %>%  
mutate(lty=ifelse(same.dept,1,2))  
# plot  
plot(  
  g %>% set_edge_attr("width", value=e$width)  
%>% set_edge_attr("lty", value=e$lty),  
  edge.arrow.size=0.8, edge.curved=T  
)  
# legend  
legend("topleft",  
  legend=unique(v$gender), pch=21, pt.bg=c("red", "blue"),  
  title="gender", box.lty=0)  
legend("left", legend=unique(e$same.dept), lty=c(1,2),  
  title = "same.dept", box.lty=0) legend("topright",  
  legend=sort(unique(e$friendship)),  
  lwd=sort(unique(e$friendship)), title="friendship",  
  box.lty=0)
```



# Edge aesthetics -- ggraph

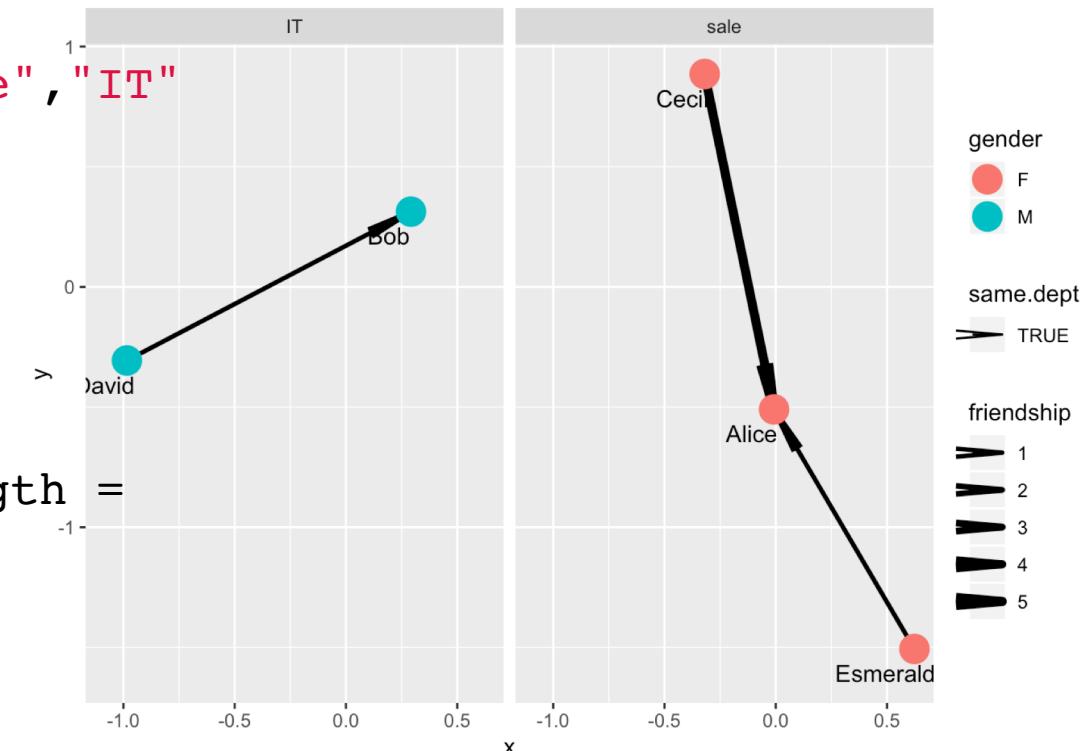
```
ggraph(g, layout="kk") +  
  geom_edge_link(  
    aes(edge_width=friendship,  
        edge_linetype=same.dept),  
    arrow = arrow(angle=5, length =  
      unit(0.3, "inches"))  
  ) +  
  geom_node_point(aes(color=gender), size=6) +  
  geom_node_text(aes(label=name), nudge_y = -  
    0.1, nudge_x = -0.1) +  
  scale_edge_width(range  
    = c(1, 2)) +  
  theme_void()
```



# Facet

- Facet in ggraph can be `facet_edges`, `facet_nodes` or `facet_graph`.
- The first two are similar to `facet_wrap` while `facet_graphs` is comparable to `facet_grid` in ggplot

```
# add dept to graph
g = g %>%
  set_vertex_attr("dept", value=c("sale", "IT", "sale", "IT",
  , "sale")) %>%
  set_edge_attr("same.dept", value=c(F, F, T, F, T, T))
# facet based on the dept
ggraph(g, layout="kk") +
  facet_nodes(~dept, drop = F) +
  geom_edge_link(aes(edge_width=friendship,
  linetype=same.dept), arrow = arrow(angle=5, length =
  unit(0.3, "inches"))) +
  geom_node_point(aes(color=gender), size=6) +
  geom_node_text(aes(label=name), nudge_y = -0.1,
  nudge_x = -0.1) +
  scale_edge_width(range = c(1, 2))
```



# Network layout

- Network visualization layout depends on the coordinates of node at canvas. There are many internal-programmed layout in both igraph and ggraph

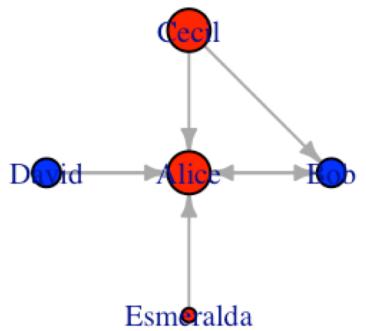
- **Standard layouts**

- **star**: place one node in the center and the rest equidistantly around it. `as_star()`
    - **circle**: place nodes in a circle in the order of their index. `in_circle()`
    - **nicely**: default, tries to pick an appropriate layout. `nicely`
    - **sphere**: place nodes uniformly on a sphere - less relevant for 2D visualizations of networks. `with_sphere()`
    - **fr**: places nodes according to the force-directed algorithm of Fruchterman and Reingold. `with_fr()`
    - **kk**: uses the spring-based algorithm by Kamada and Kawai to place nodes. `with_kk()`
    - ..... [https://igraph.org/r/doc/layout\\_.html](https://igraph.org/r/doc/layout_.html) ....

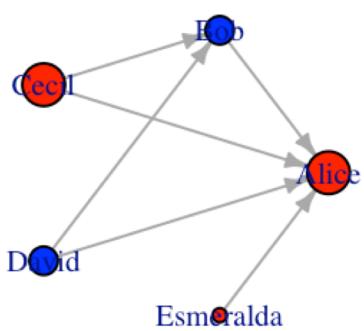
- **Hierarchical layouts**

- **tree**: parent with the parent centered above its children. `as_tree()`
    - **sugiyama**: user-defined layers

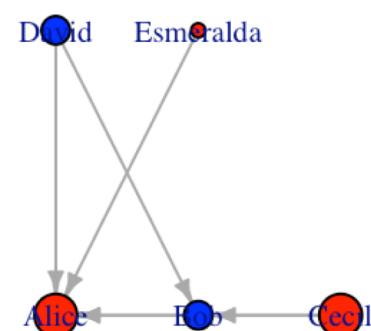
**start**



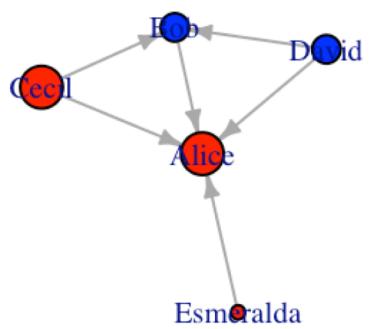
**circle**



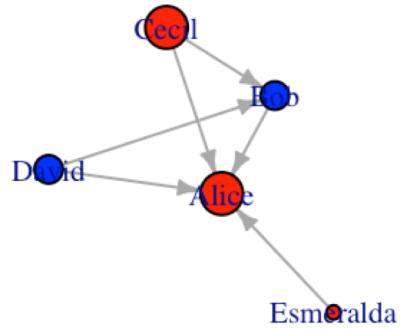
**grid**



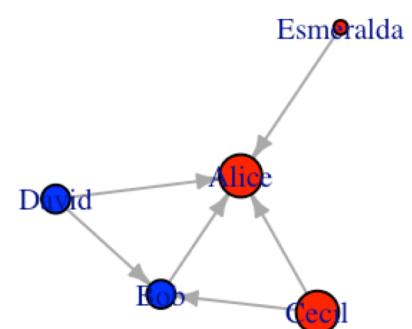
**nicely**



**Kamada and Kawai(kk)**

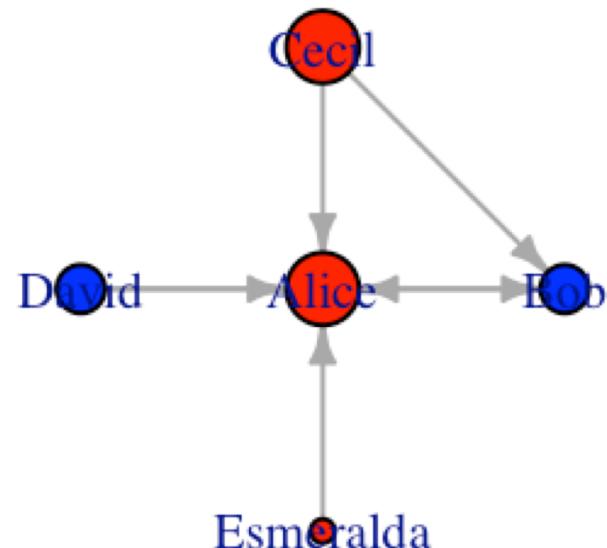


**force-directed(fr)**



# Specify layout in plot

```
# star layout in igraph default plot  
coords <- layout_(g, as_star())  
plot(g, layout = coords,  
edge.arrow.size=0.4)  
title("star")
```



```
# star layout in ggraph  
ggraph(g, layout="star") +  
geom_edge_link(aes(edge_width=friendship,  
edge_linetype=same.dept), arrow =  
arrow(angle=5, length = unit(0.3,  
"inches")))+  
geom_node_point(aes(color=gender), size=6)  
+ geom_node_text(aes(label=name), nudge_y =  
-0.1, nudge_x = -0.1) +  
scale_edge_width(range = c(1, 2)) +  
theme_void()
```

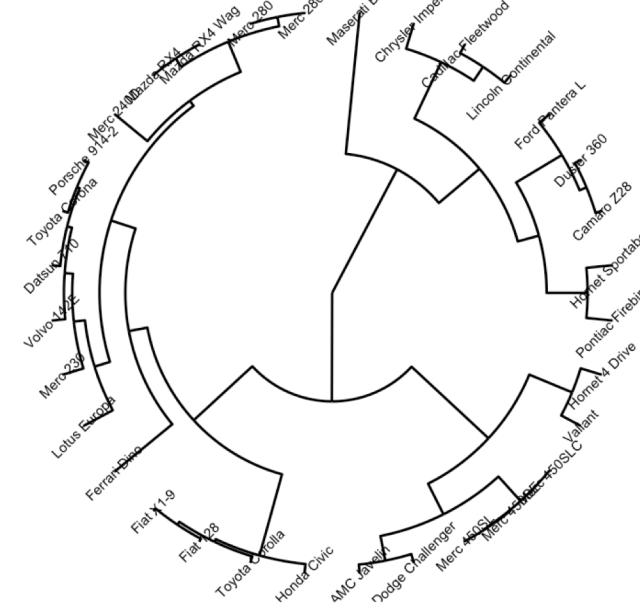
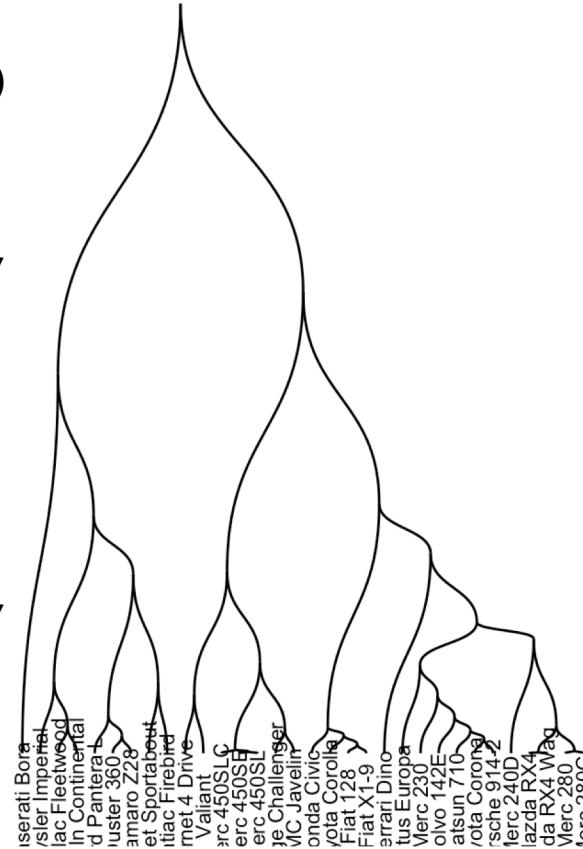
# Extra layout in ggraph

- Dendrogram: dendrogram layout not only take in graph object but also dendrogram object.

```
den <- as.dendrogram(hclust(dist(mtcars)))
p1 = ggraph(den, 'dendrogram') +
geom_edge_diagonal() +
geom_node_text(aes(label=label), angle=90,
nudge_y=-30, size=3) +
theme_void()

p2 = ggraph(den, 'dendrogram', circular =
TRUE) + geom_edge_elbow() +
geom_node_text(aes(label=label), angle=45,
size=2) + coord_fixed()+
theme_void()

grid.arrange(p1,p2,ncol=2)
```

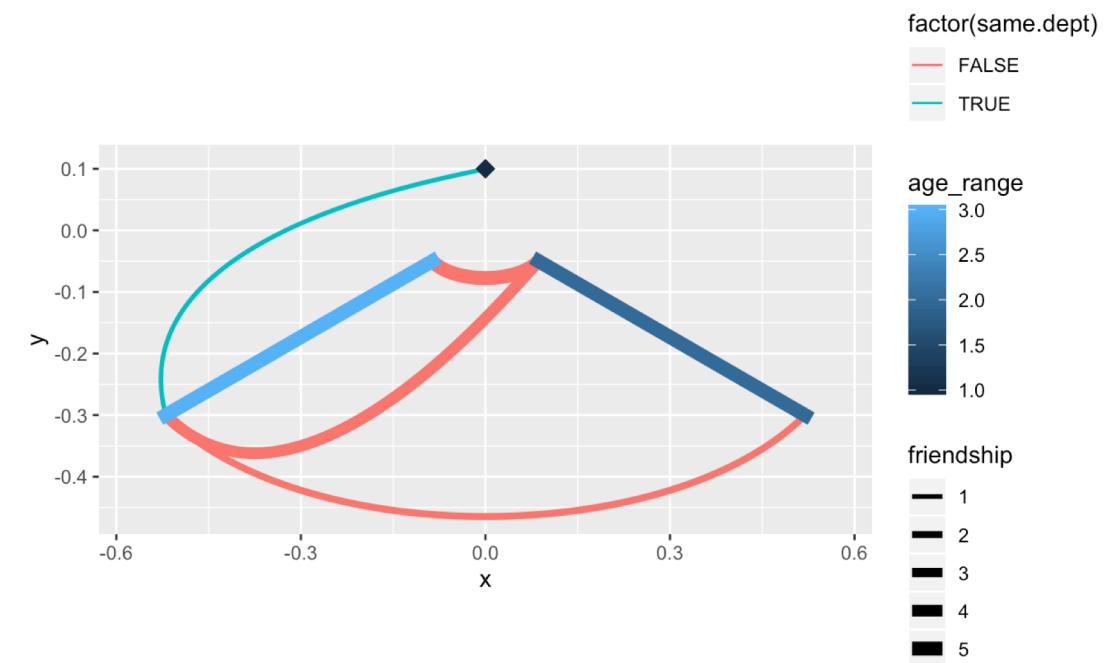


# Extra layout in ggraph

- Hive: make nodes group into a axis and connecting axis instead.

```
# group age_range into axis
V(g)$age_range = factor(V(g)$age_range)

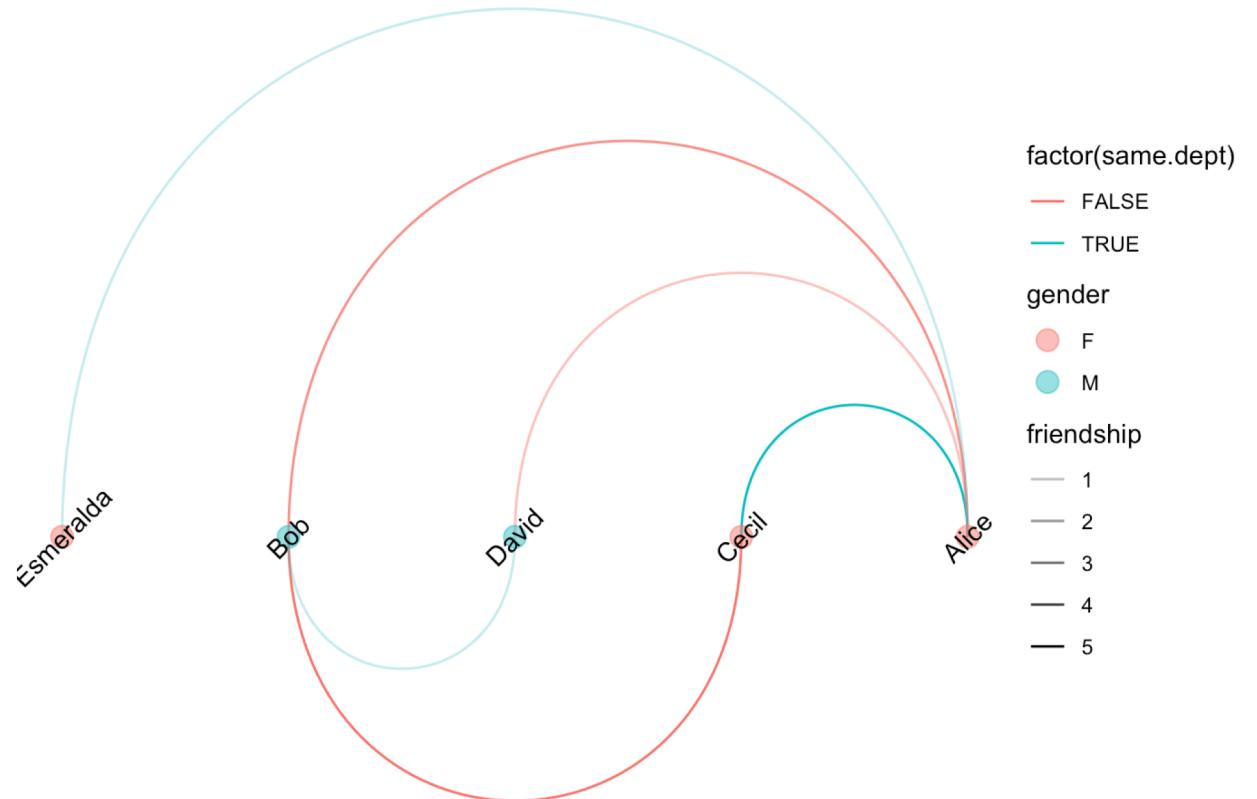
ggraph(g, 'hive', axis = 'age_range',
sort.by = 'age') +
geom_edge_hive(aes(color =
factor(same.dept), edge_width=friendship)) +
geom_axis_hive(aes(color = age_range), size
= 3, label = FALSE) +
coord_fixed() +
scale_edge_width(range=c(1,3))
```



# Extra layout in ggraph

- Linear: all nodes at same line

```
ggraph(g, layout = 'linear',
sort.by = 'age') +
geom_edge_arc(aes(color =
factor(same.dept),
edge_alpha=friendship)) +
geom_node_point(aes(color=gender),
size=4, alpha=0.5) +
geom_node_text(aes(label=name),
angle=45) +
theme_void() +
scale_edge_alpha(range=c(0.3,1))
```

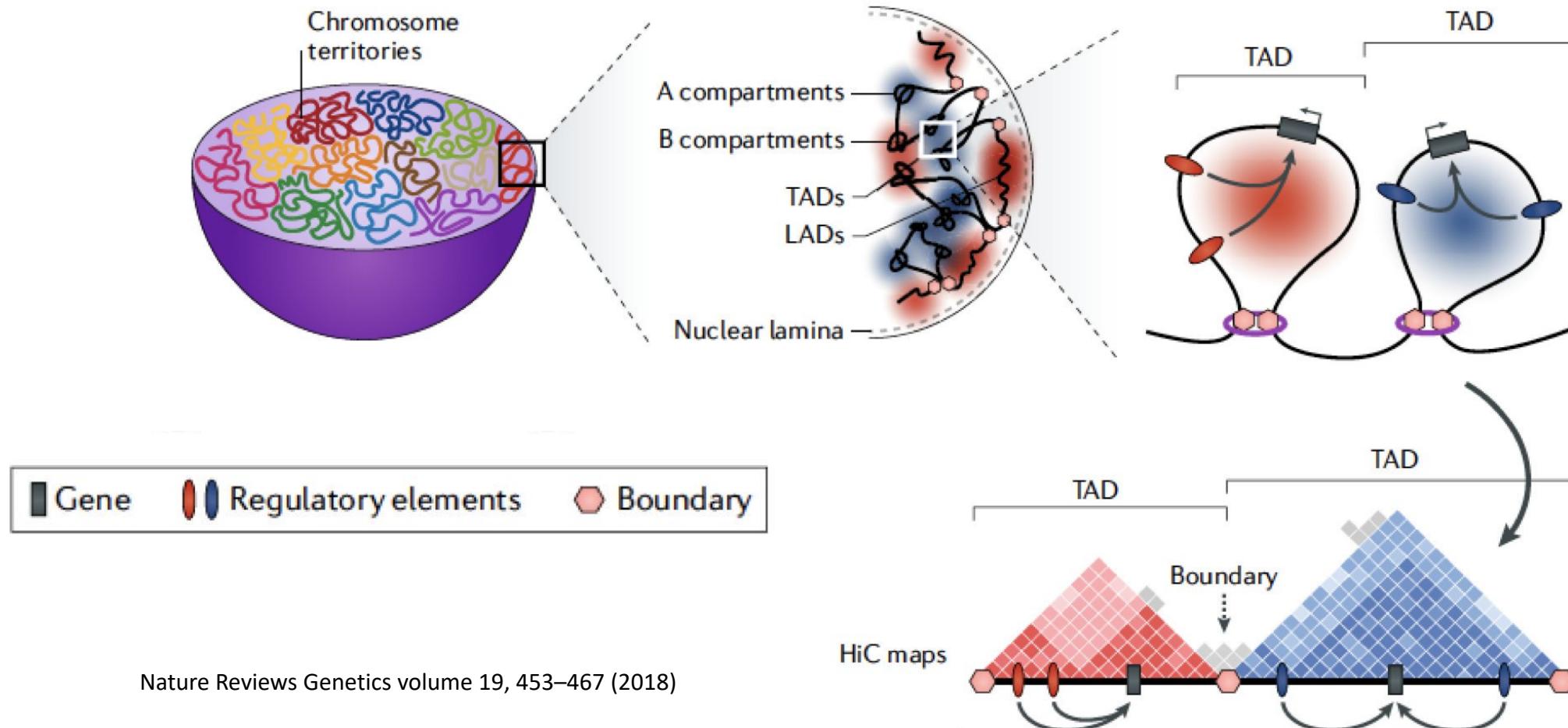


# Agenda

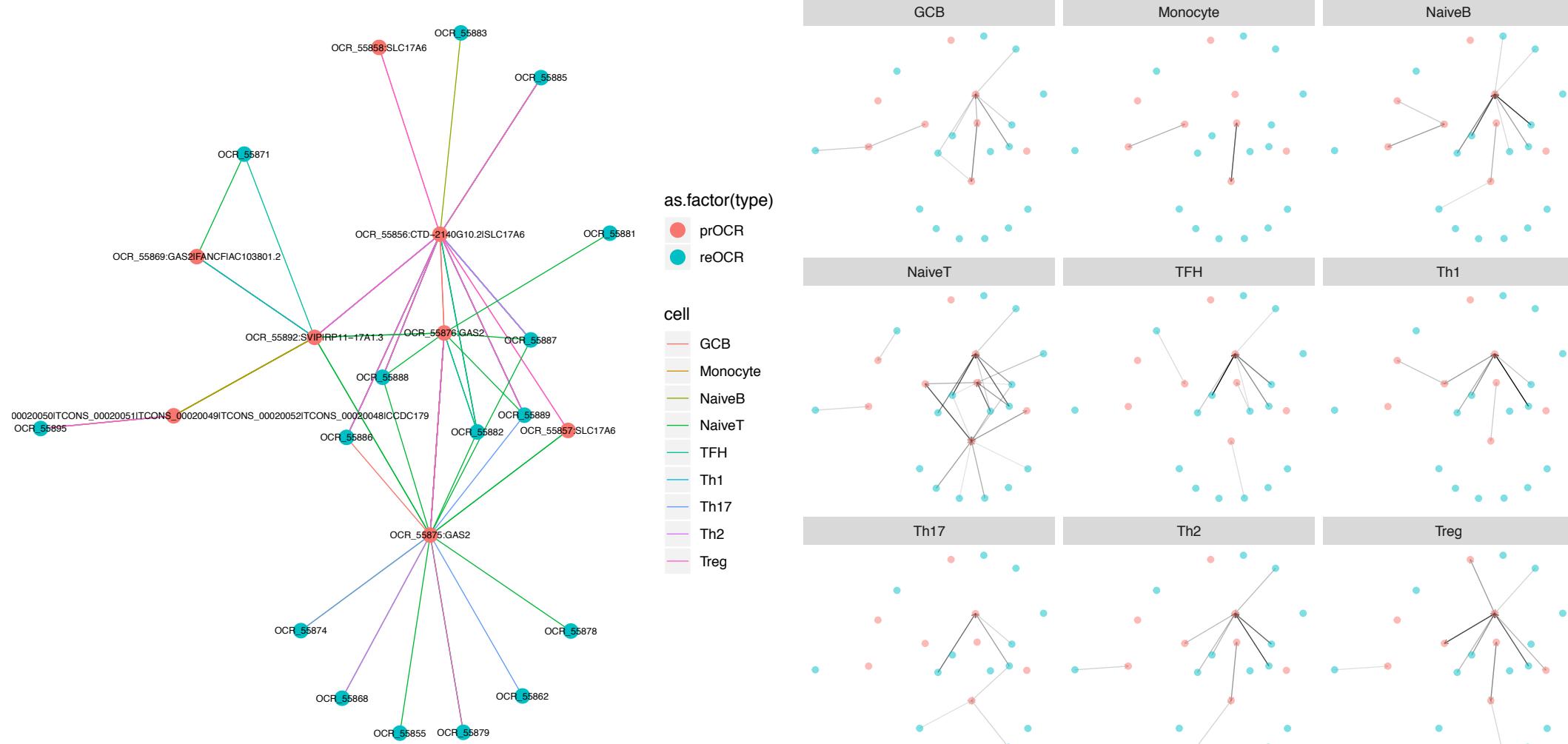
- Network analysis basics
  - R object IGRAPH "I/O"
  - object IGRAPH manipulation
- Graph advance
  - Graph measurement
  - Graph cluster
- Network visualization
  - Aesthetics of network elements
  - Network layout
- **Network application show case in Biology**
- Group Exercise – NYC flight

# Network application in gene regulation

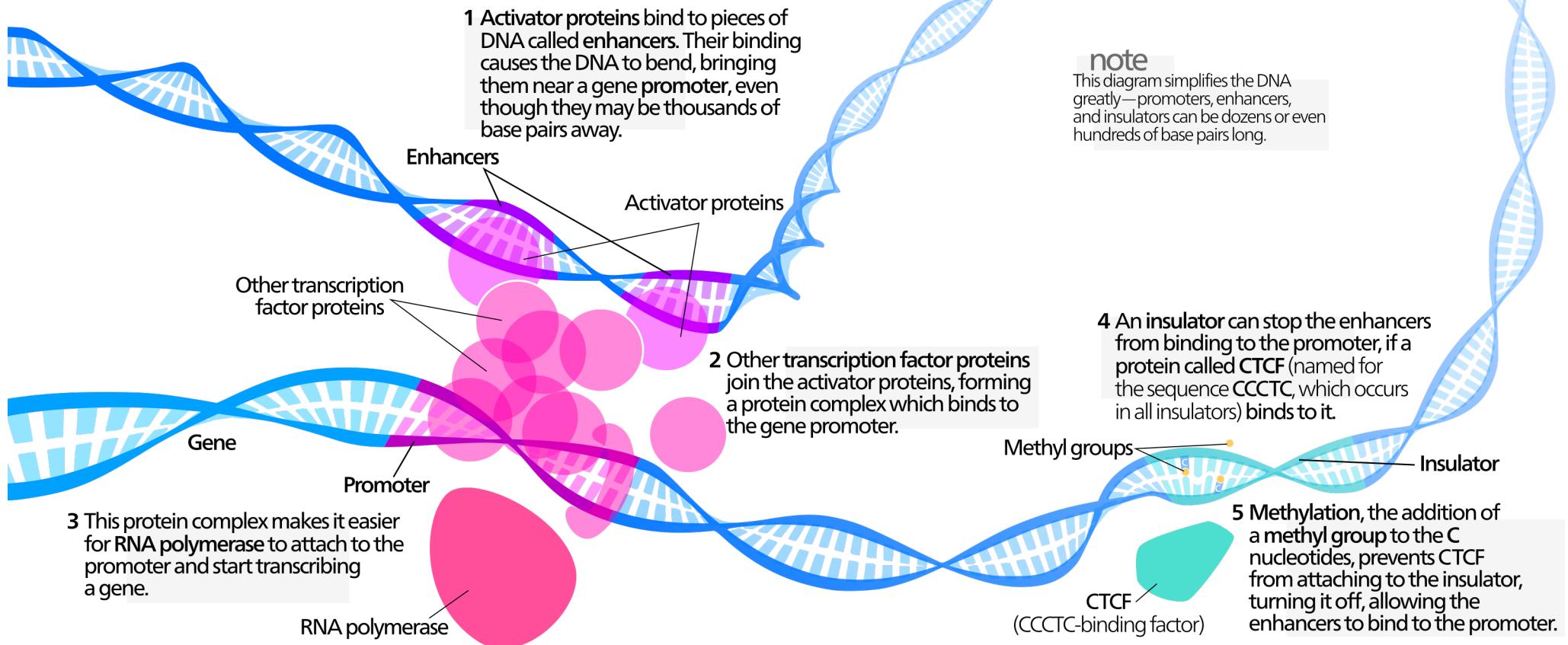
- Topologically Associating Domains (TADs)



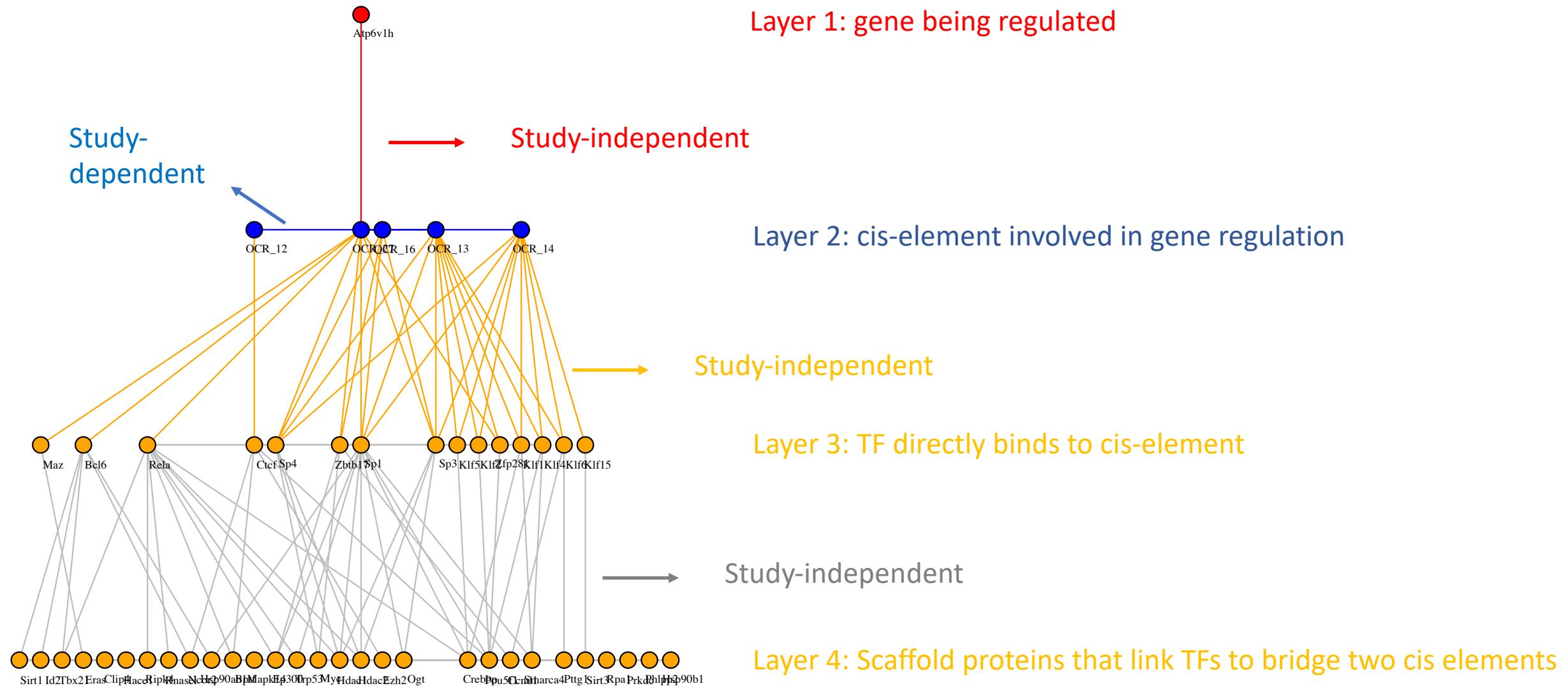
# Cell-specific gene regulation network within TADs



# transcription factors of eukaryotic cells



# Gene regulation network with transcriptional factors



# Other network packages

- Network analysis tool:
  - Statnet: <https://statnet.org/trac/wiki>
- Network visualization:
  - ggnet: <https://briatte.github.io/ggnet/>
- Interactive network :
  - visNetwork: <https://datastorm-open.github.io/visNetwork/>
  - jstree: <https://bwlewis.github.io/rthreejs/>
  - Ndtv:  
[http://statnet.csde.washington.edu/workshops/SUNBELT/current/ndtv/ndtv\\_workshop.html](http://statnet.csde.washington.edu/workshops/SUNBELT/current/ndtv/ndtv_workshop.html)