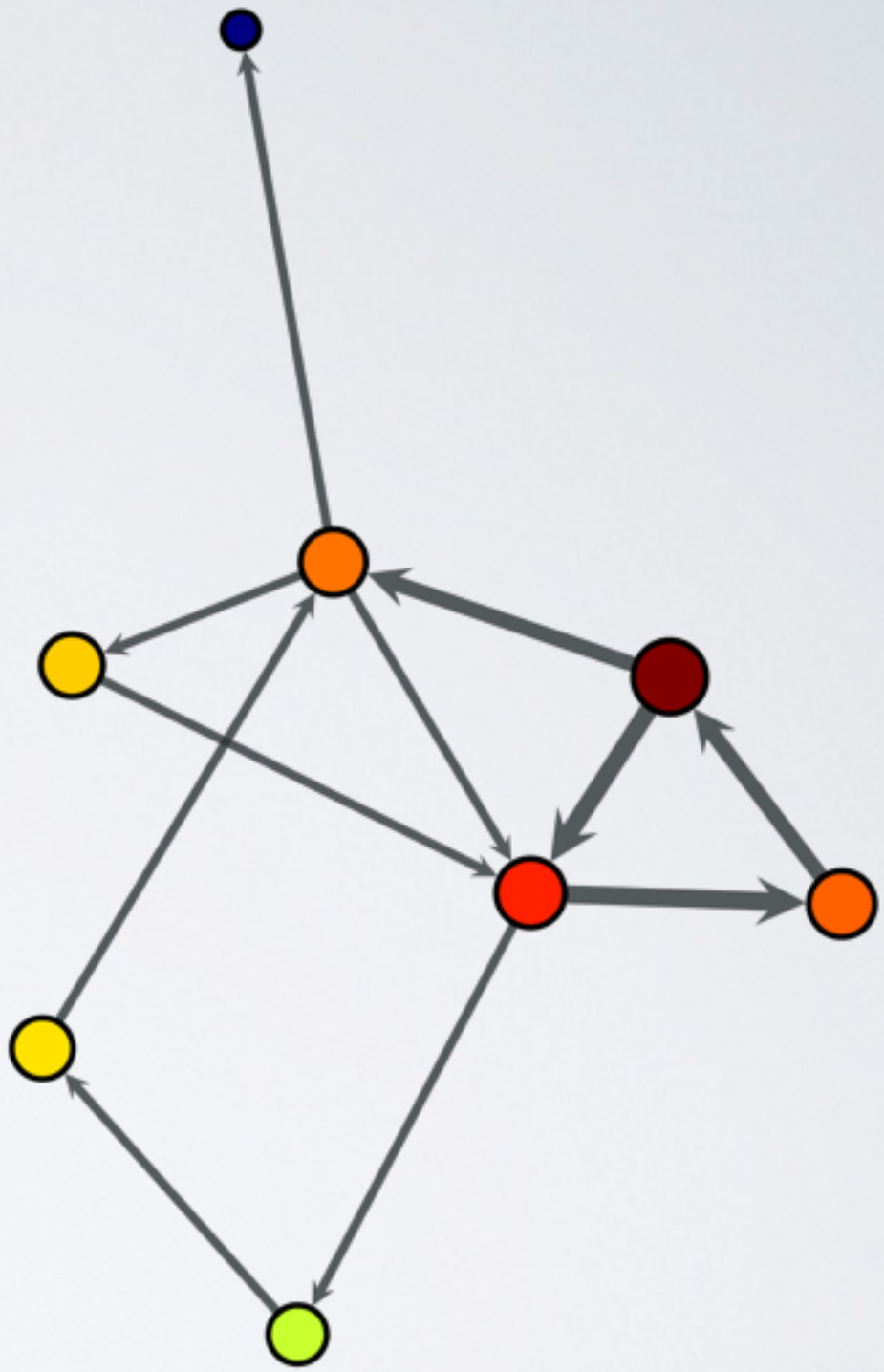
Graph-Tool

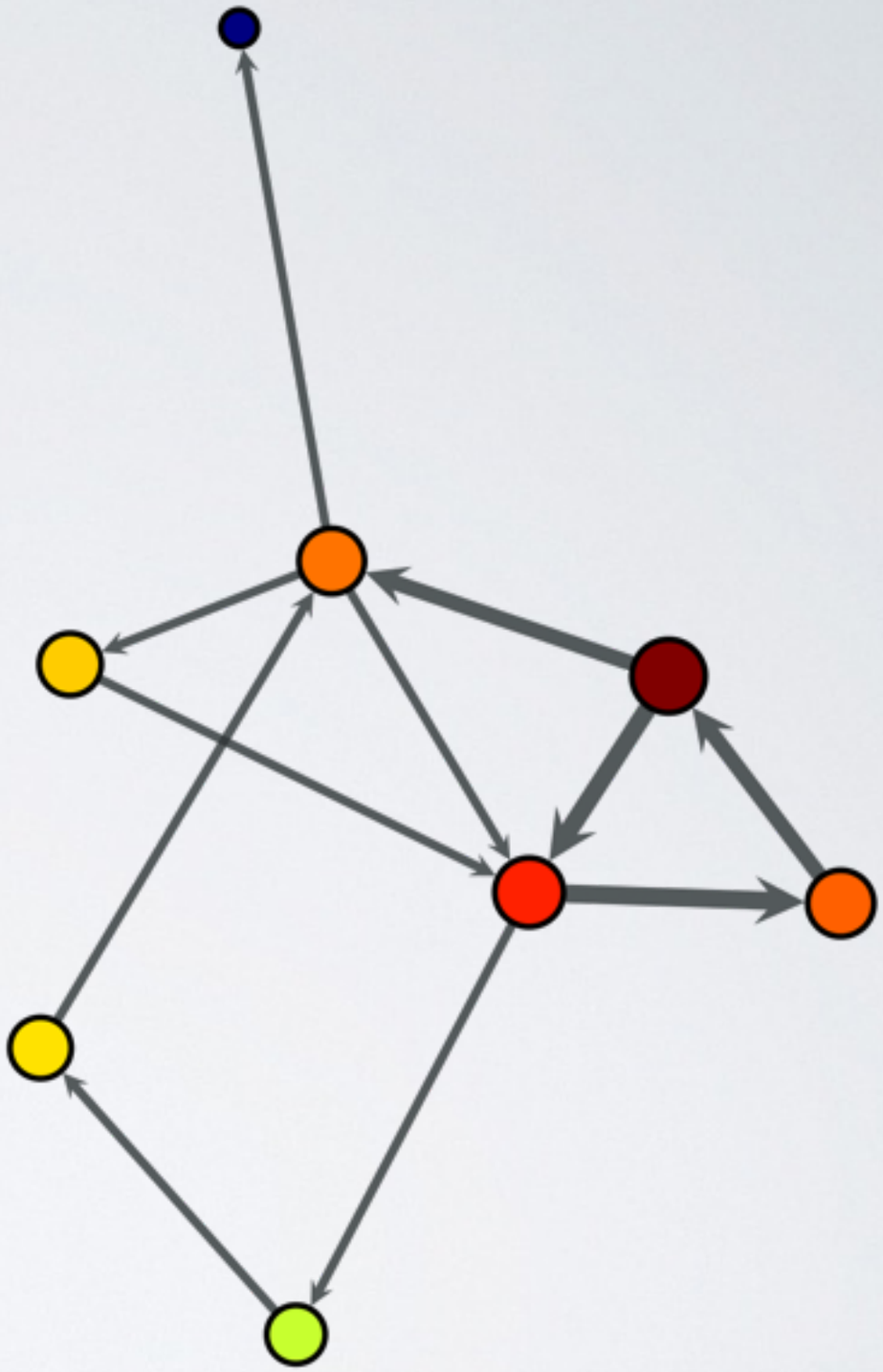
The Efficient Network Analyzing Tool for Python

Mosky



Graph-Tool in Practice

Mosky



MOSKY

MOSKY

- Python Charmer at Pinkoi

MOSKY

- Python Charmer at Pinkoi
- An author of the Python packages:
  - MoSQL, Clime, Uniout, ZIPCodeTW, …

MOSKY

- Python Charmer at Pinkoi
- An author of the Python packages:
  - MoSQL, Clime, Uniout, ZIPCodeTW, …
- A speaker of the conferences
  - 2014: PyCon APAC, OSDC; 2013: PyCon APAC, PyCon TW, COSCUP, …

MOSKY

- Python Charmer at Pinkoi
- An author of the Python packages:
  - MoSQL, Clime, Uniout, ZIPCodeTW, …
- A speaker of the conferences
  - 2014: PyCon APAC, OSDC; 2013: PyCon APAC, PyCon TW, COSCUP, …
- A Python instructor

MOSKY

- Python Charmer at Pinkoi
- An author of the Python packages:
  - MoSQL, Clime, Uniout, ZIPCodeTW, …
- A speaker of the conferences
  - 2014: PyCon APAC, OSDC; 2013: PyCon APAC, PyCon TW, COSCUP, …
- A Python instructor
- mosky.tw

OUTLINE

OUTLINE

- Introduction

OUTLINE

- Introduction
- Create Graph

OUTLINE

- Introduction
- Create Graph
- Visualize Graph

OUTLINE

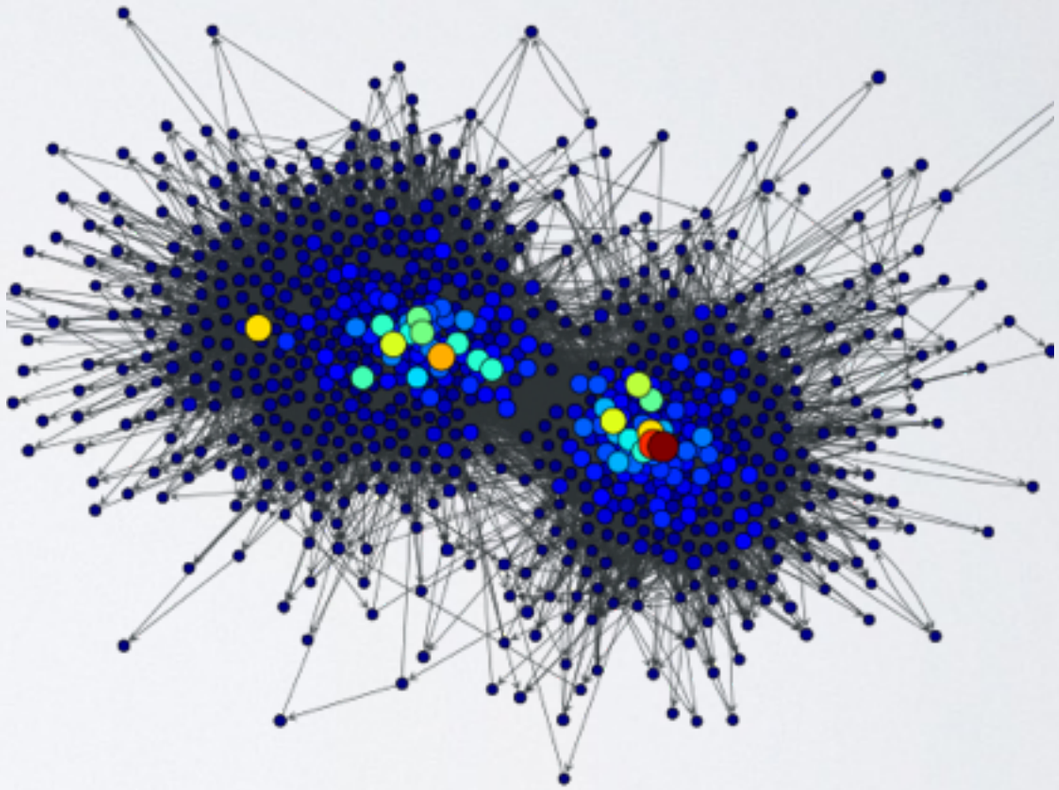
- Introduction
- Create Graph
- Visualize Graph
- Analyze Graph

OUTLINE

- Introduction
- Create Graph
- Visualize Graph
- Analyze Graph
- Conclusion

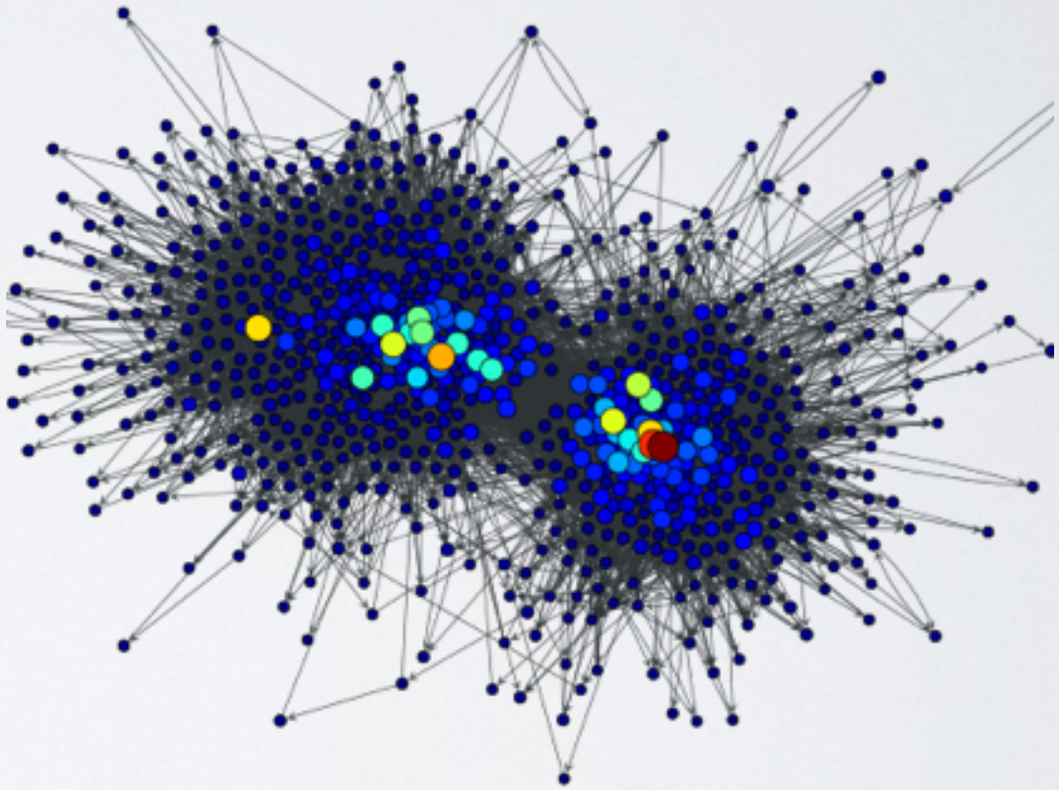
INTRODUCTION

GRAPH-TOOL



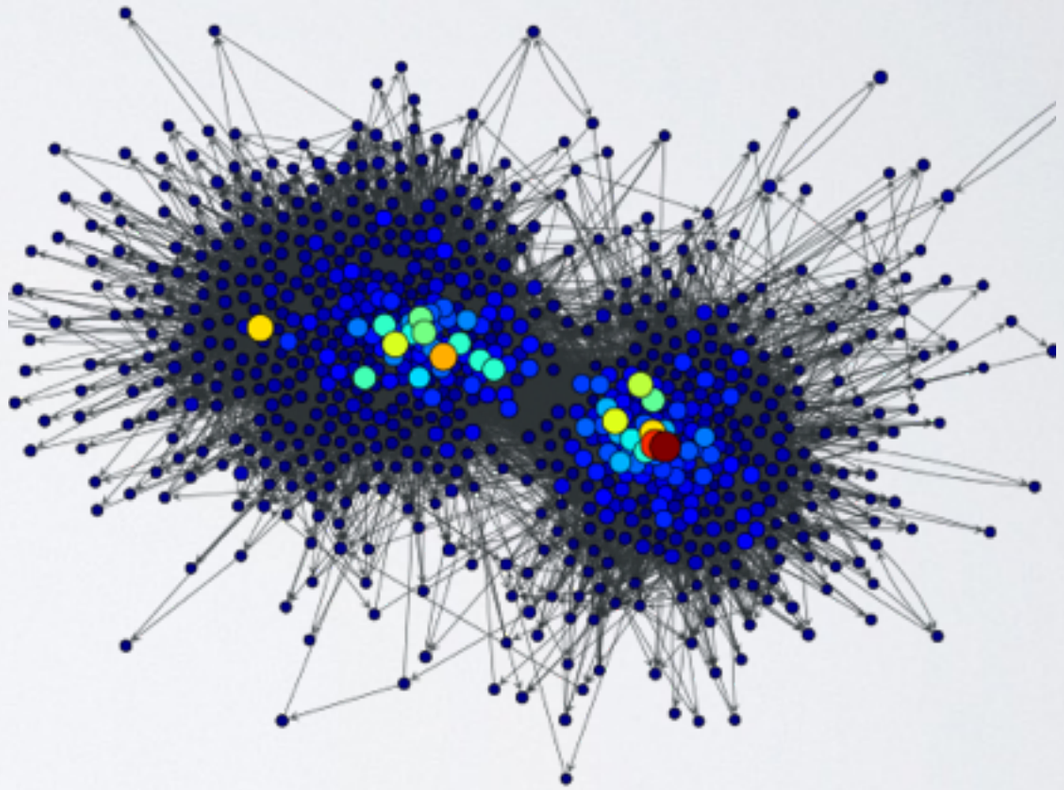
GRAPH-TOOL

- It's for analyzing graph.



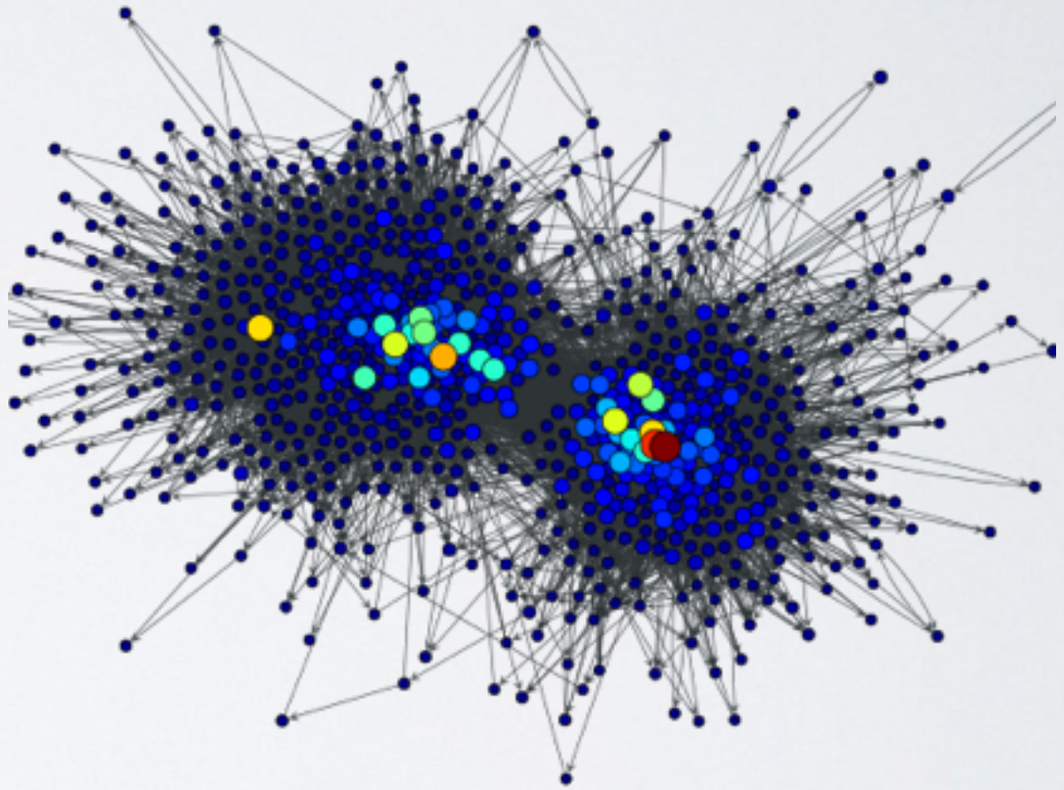
GRAPH-TOOL

- It's for analyzing graph.
- Fast. It bases on Boost Graph in C++.



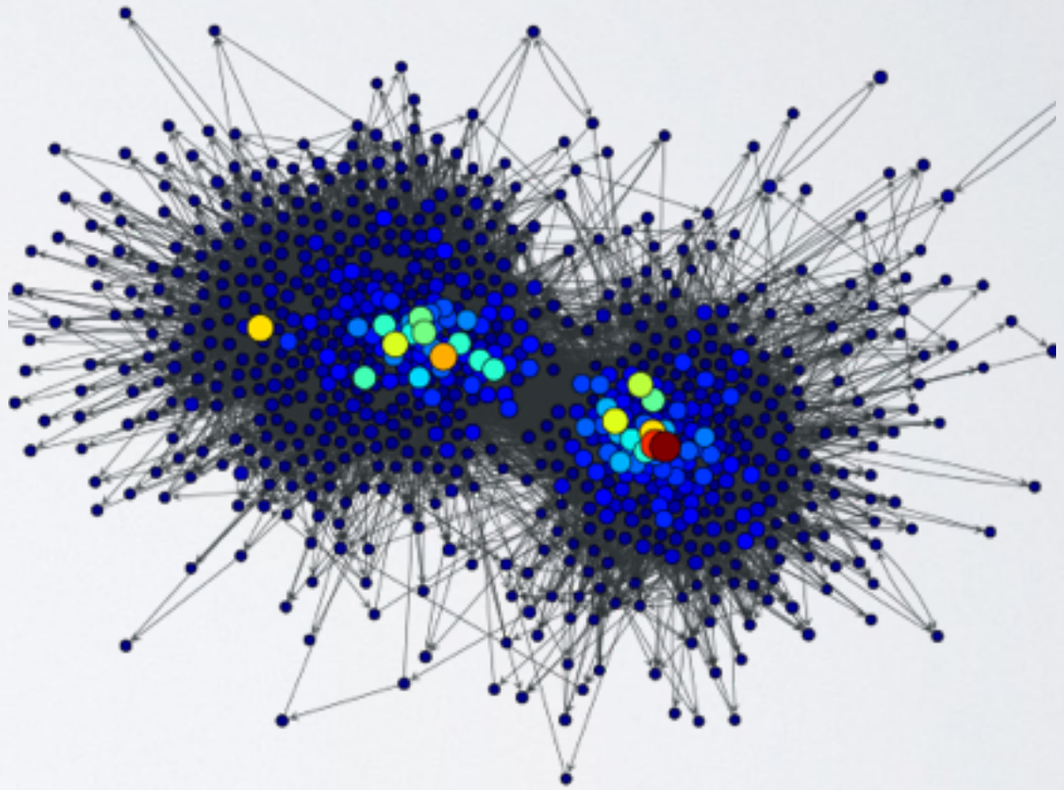
GRAPH-TOOL

- It's for analyzing graph.
- Fast. It bases on Boost Graph in C++.
- Powerful visualization



GRAPH-TOOL

- It's for analyzing graph.
- Fast. It bases on Boost Graph in C++.
- Powerful visualization
- Lot of useful algorithms



GET GRAPH-TOOL

GET GRAPH-TOOL

- Super easy on Debian / Ubuntu
  - http://graph-tool.skewed.de/download#debian

GET GRAPH-TOOL

- Super easy on Debian / Ubuntu
  - http://graph-tool.skewed.de/download#debian
- Super hard on Mac
  - http://graph-tool.skewed.de/download#macos
  - Install the dependencies by homebrew and pip.
    Then compile it from source.
  - Note it may take you 3~4 hours. I warned you!

CREATE GRAPH

BEFORE STARTING

BEFORE STARTING

- Define your problem.

BEFORE STARTING

- Define your problem.
- Convert it into a graphic form.

BEFORE STARTING

- Define your problem.
- Convert it into a graphic form.
- Parse raw data.

MY PROBLEM

MY PROBLEM

- To improve the duration of an online marketplace.

MY PROBLEM

- To improve the duration of an online marketplace.

- What's product browsing flow that users prefer?

IN GRAPHIC FORM

	What	Weight
Vertex	Product	Count
Edge	Directed Browsing	Count

PARSING

PARSING

- Regular expression
    - Filter garbages.

PARSING

- Regular expression
    - Filter garbages.
- Sorting

PARSING

- Regular expression
  - Filter garbages.
- Sorting
- Pickle
  - HIGHEST_PROTOCOL
  - Use tuple to save space/time.
  - Save into serial files.

VERTEX AND EDGE

```
import graph_tool.all as gt

g  = gt.Graph()
v1 = g.add_vertex()
v2 = g.add_vertex()
e  = g.add_edge(v1, v2)
```

PROPERTY

```
v_count_p = g.new_vertex_property('int')

# store it in our graph, optionally
g.vp['count'] = v_count_p
```

FASTER IMPORT

```
from graph_tool import Graph
```

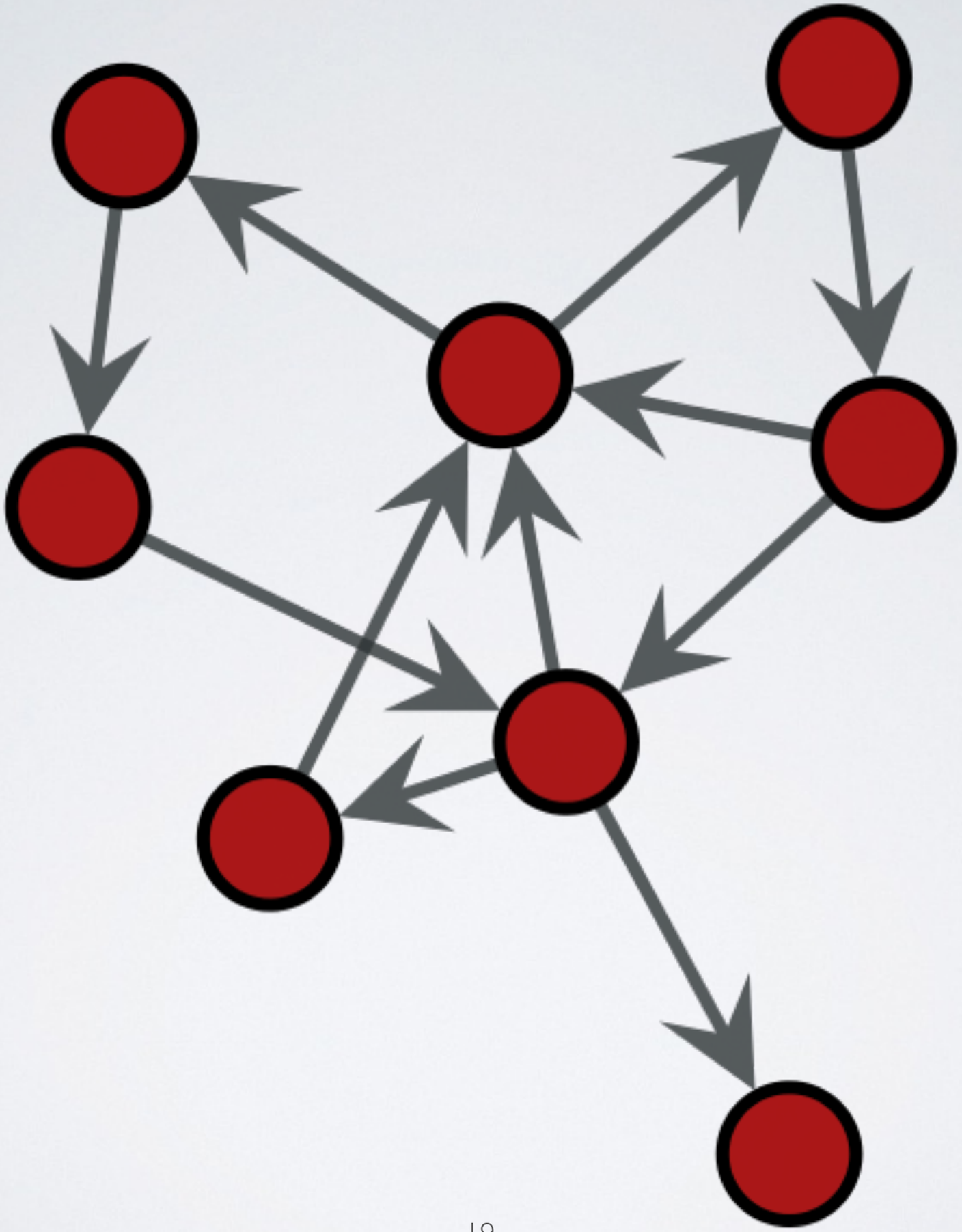
COUNTING

```
name_v_map = {}
for name in names:
    v = name_v_map.get(name)
    if v is None:
        v = g.add_vertex()
        v_count_p[v] = 0
        name_v_map[name] = v
    v_count_p[v] += 1
```

VISUALIZE GRAPH

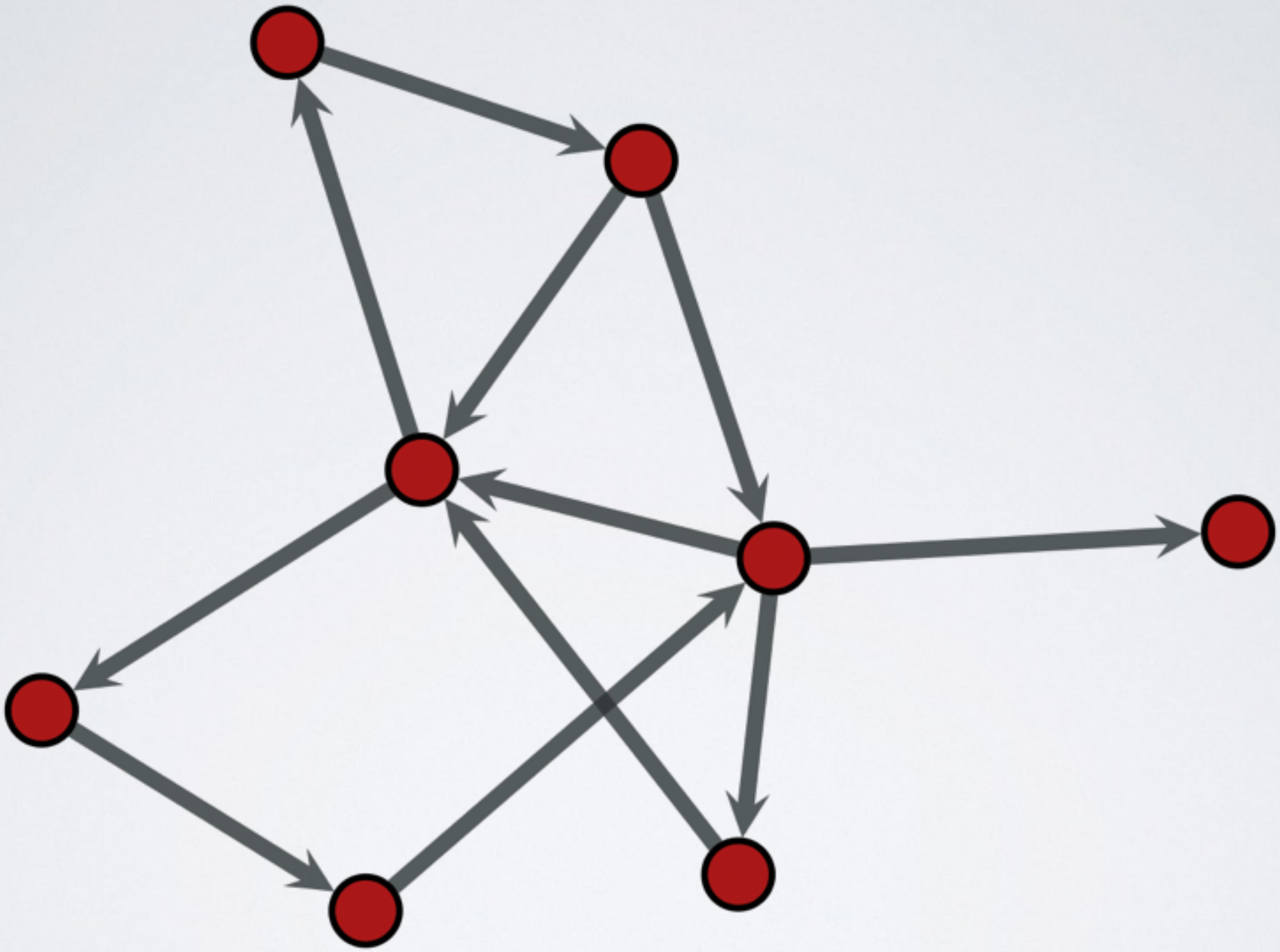
THE SIMPLEST

```
gt.graph_draw(
     g,
     output_path = 'output.pdf',
)

gt.graph_draw(
     g,
     output_path = 'output.png',
)
```



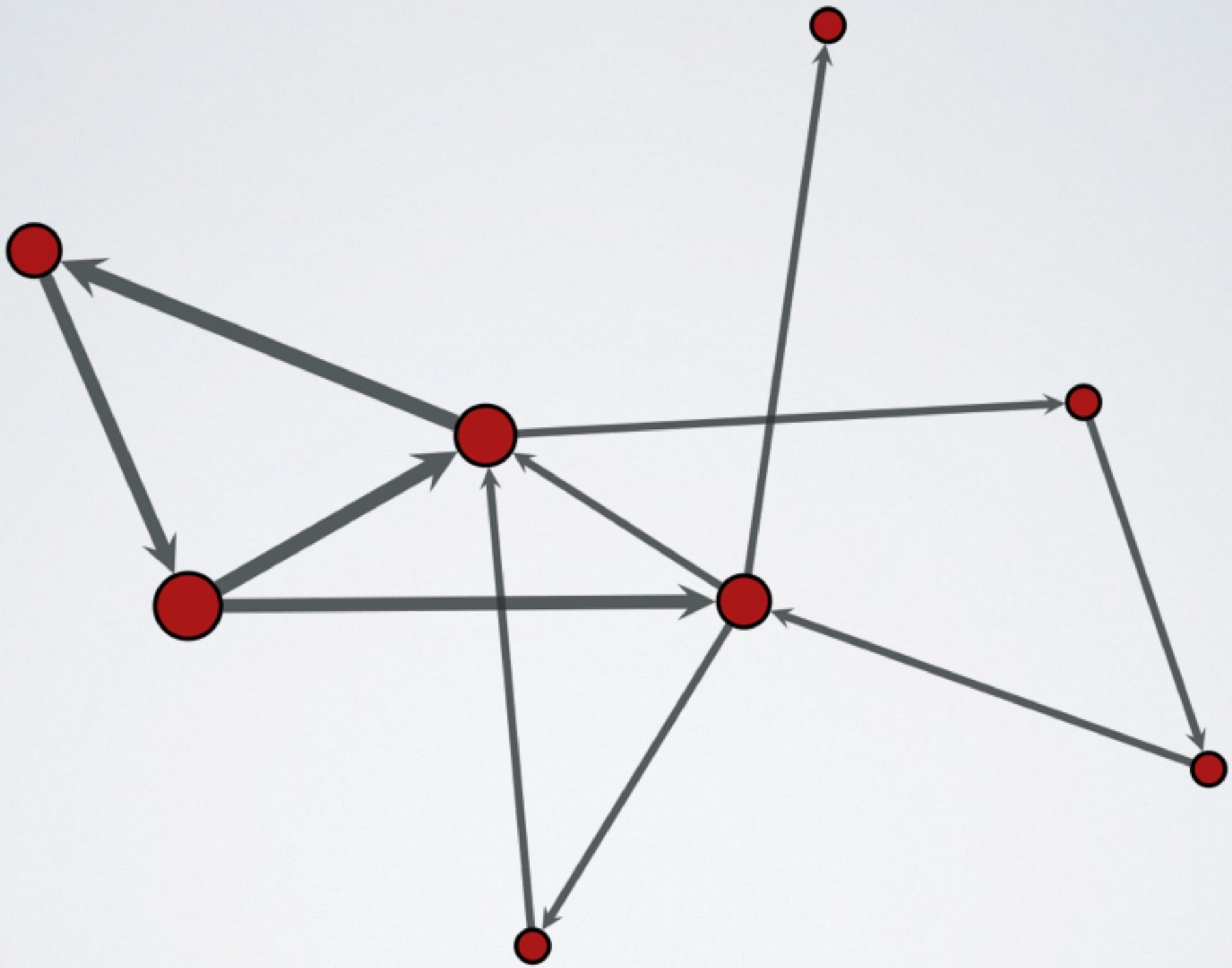
USE CONSTANTS

```
SIZE      = 400
V_SIZE    = SIZE / 20.
E_PWIDTH = V_SIZE / 4.
gt.graph_draw(
      …
      output_size = (SIZE, SIZE),
      vertex_size = V_SIZE,
      edge_pen_width = E_PWDITH,
)
```



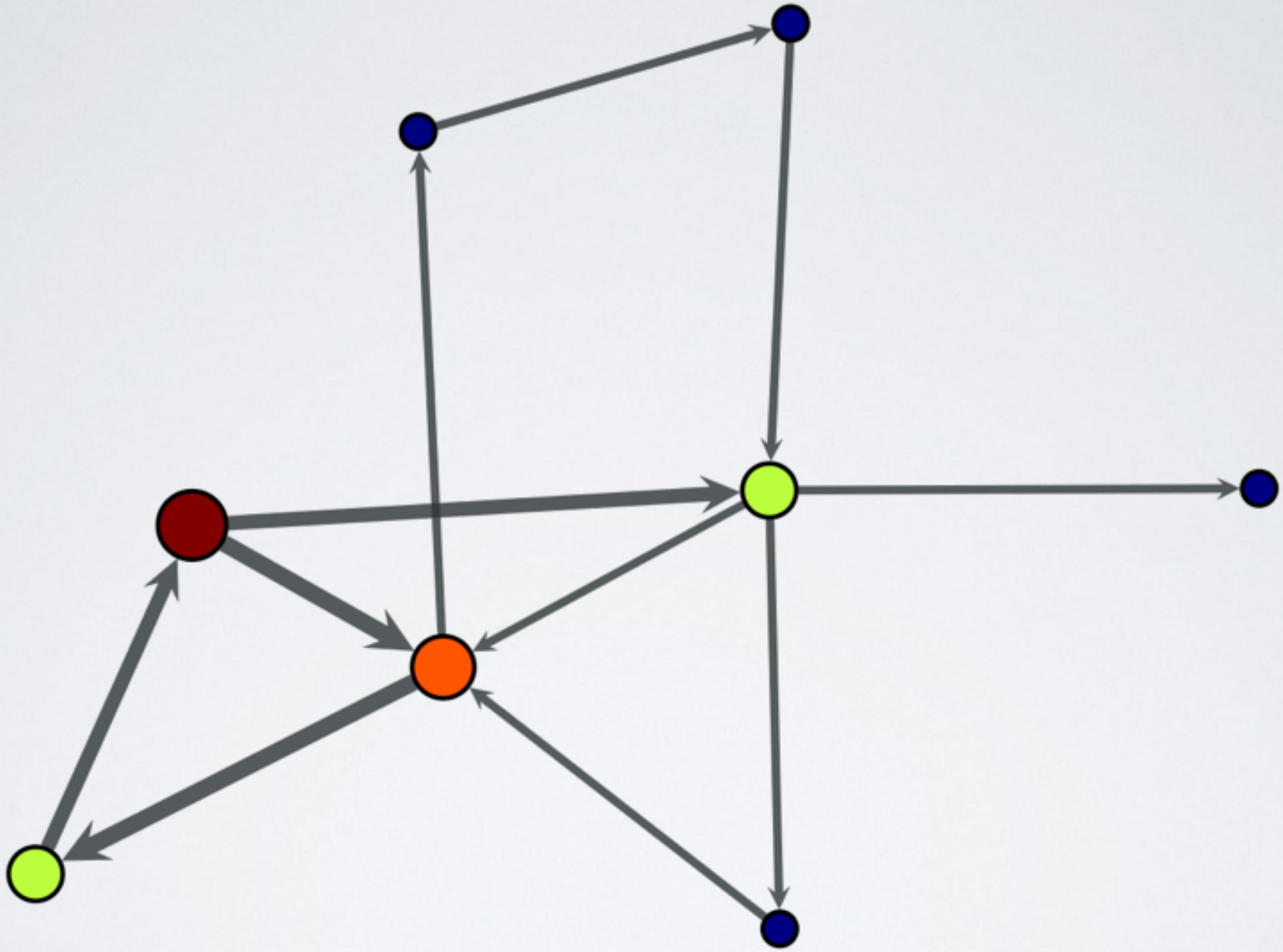
USE PROP_TO_SIZE

```
v_size_p = gt.prop_to_size(
    v_count_p,
    MI_V_SIZE,
    MA_V_SIZE,
)
…
gt.graph_draw(
    …
    vertex_size = v_size_p,
    edge_pen_width = e_pwidth_p,
)
```



USE FILL_COLOR

```
gt.graph_draw(
      …
      vertex_fill_color = v_size_p,
)
```



ANALYZE GRAPH

CHOOSE AN ALGORITHM

CHOOSE AN ALGORITHM

- Search algorithms
  - BFS search …

CHOOSE AN ALGORITHM

- Search algorithms
  - BFS search …
- Assessing graph topology
  - shortest path …

CHOOSE AN ALGORITHM

- Search algorithms
  - BFS search …
- Assessing graph topology
  - shortest path …
- Centrality measures
  - pagerank, betweenness, closeness …

- Maximum flow algorithms

- Maximum flow algorithms
- Community structures

- Maximum flow algorithms
- Community structures
- Clustering coefficients

CENTRALITY MEASURES

CENTRALITY MEASURES

- Degree centrality
  - the number of links incident upon a node
  - the immediate risk of taking a node out

CENTRALITY MEASURES

- Degree centrality
  - the number of links incident upon a node
  - the immediate risk of taking a node out
- Closeness centrality
  - sum of a node's distances to all other nodes
  - the cost to spread information to all other nodes

- Betweenness centrality
  - the number of times a node acts as a bridge
  - the control of a node on the communication between other nodes

- Betweenness centrality
  - the number of times a node acts as a bridge
  - the control of a node on the communication between other nodes
- Eigenvector centrality
  - the influence of a node in a network
  - Google's PageRank is a variant of the Eigenvector centrality measure

MY CHOICE

MY CHOICE

- Centrality measures - Closeness centrality

MY CHOICE

- Centrality measures - Closeness centrality

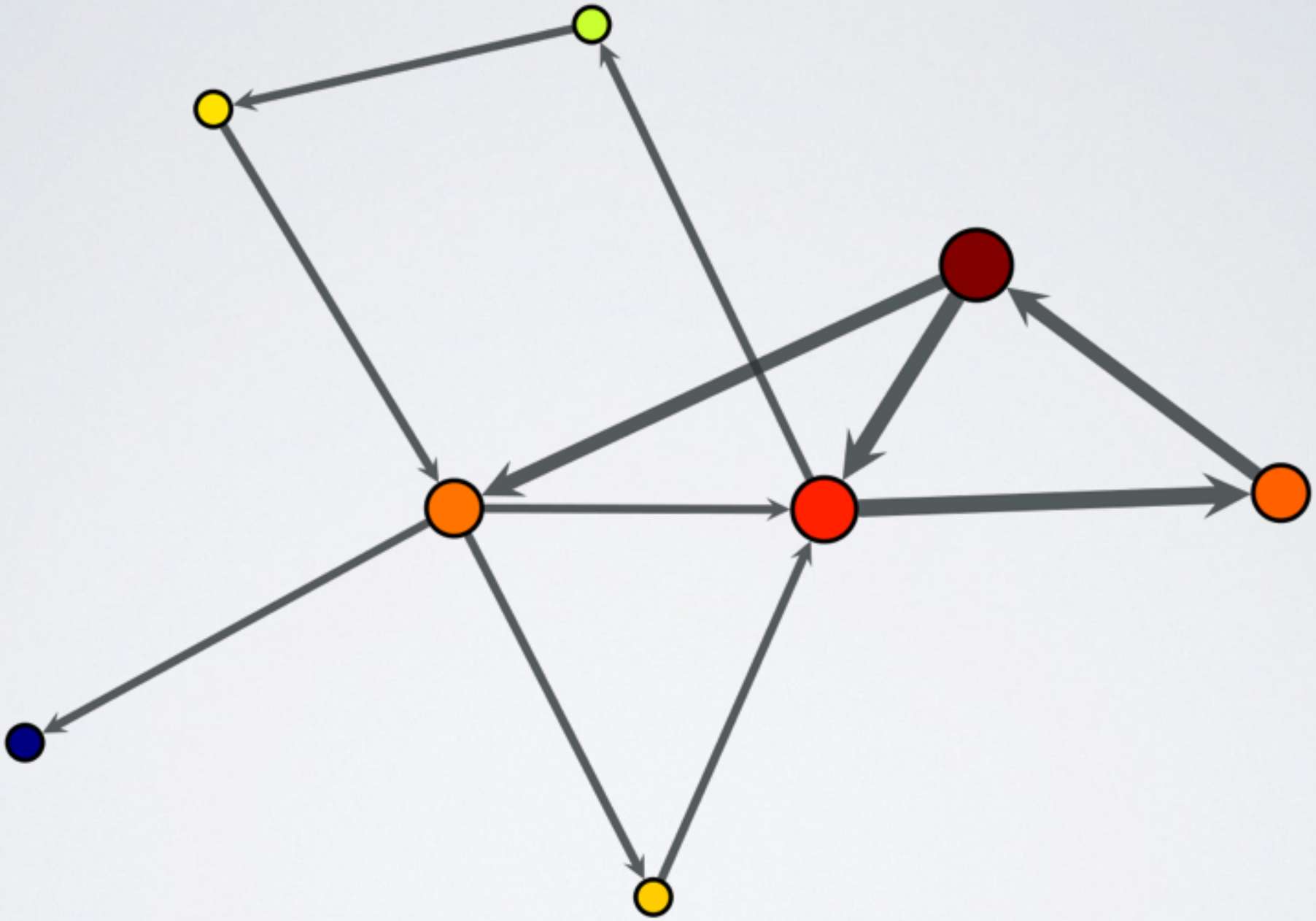
- Get the products are easier to all other products.

CALCULATE CLOSENESS

```
e_icount_p = g.new_edge_property('int')
e_icount_p.a = e_count_p.a.max()-e_count_p.a

v_cl_p = closeness(g, weight=e_icount_p)

import numpy as np
v_cl_p.a = np.nan_to_num(v_cl_p.a)
```

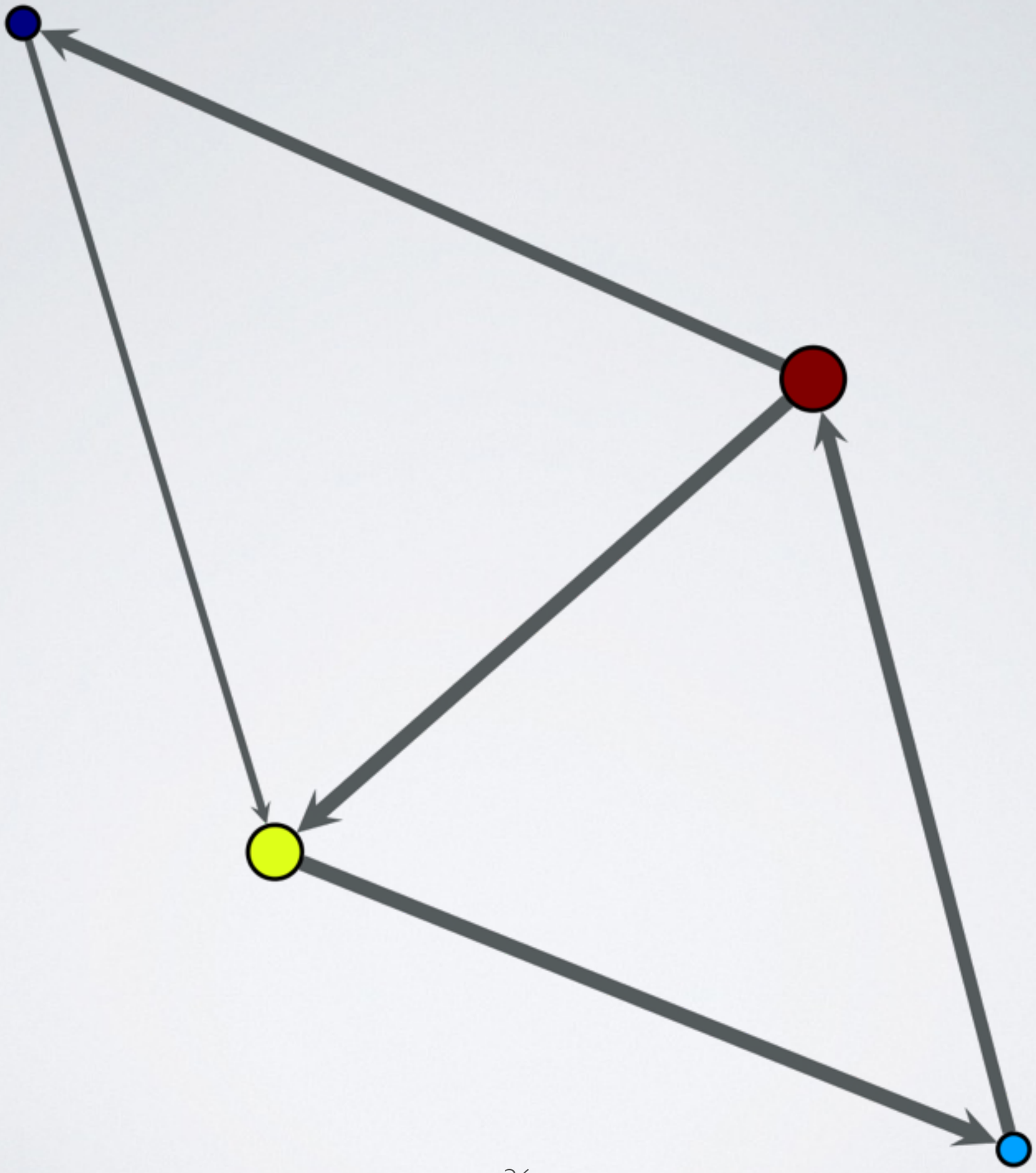
DRAW CLOSENESS

```
v_cl_size_p = gt.prop_to_size(
    v_cl_p,
    MI_V_SIZE,
    MA_V_SIZE,
)
…
gt.graph_draw(
    …
    vertex_fill_color = v_cl_size_p,
)
```



ON THE FLY FILTERING

```
v_pck_p = g.new_vertex_property('bool')
v_pck_p.a = v_count_p.a > v_count_p.a.mean()

g.set_vertex_filter(v_pck_p)
# g.set_vertex_filter(None) # unset
```

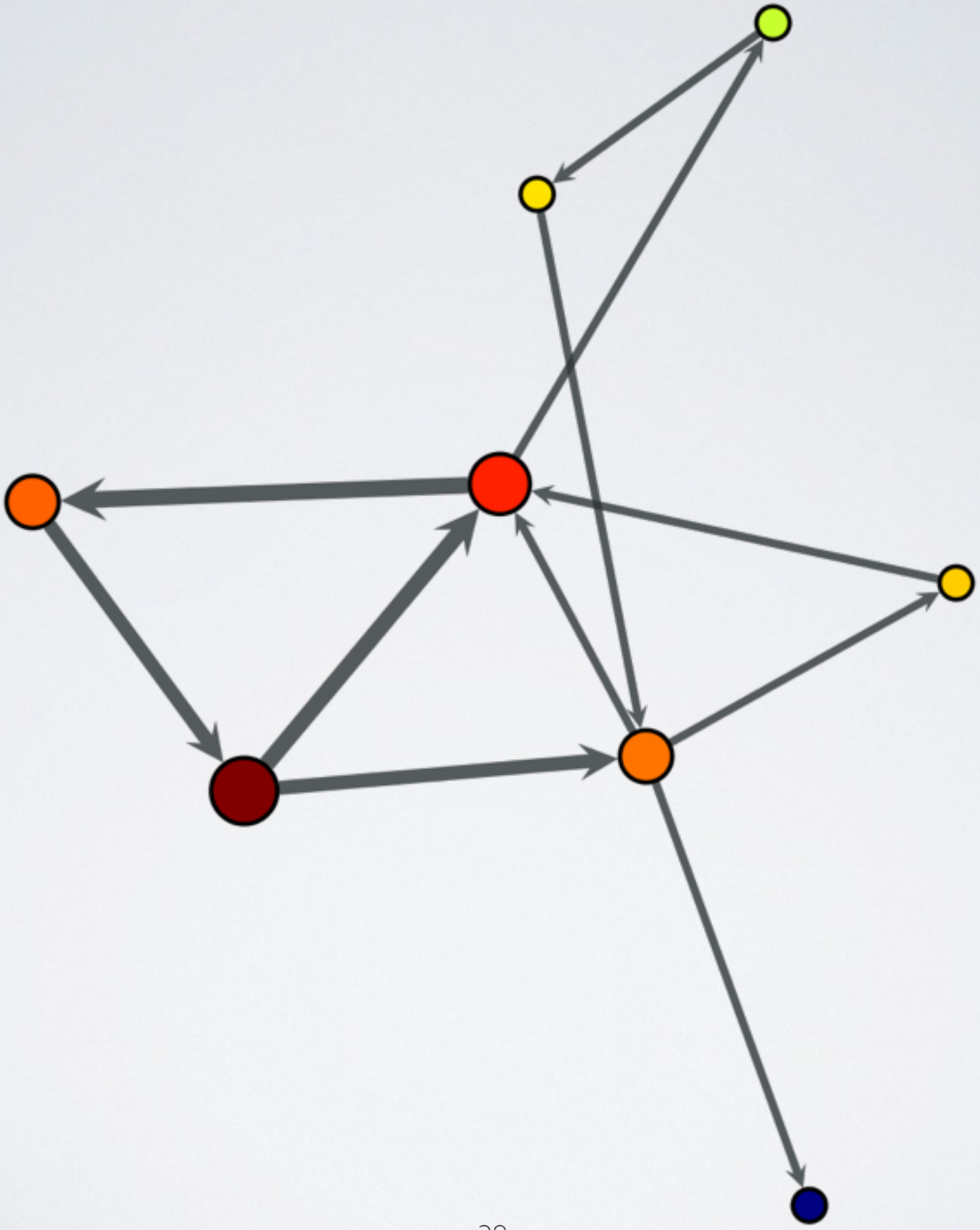


TOP N

```
t10_idxs = v_count_p.a.argsort()[-10:][::-1]

t1_idx   = t10_idxs[0]
t1_v     = g.vertex(t1_idx)
t1_name  = v_name_p[t1_v]
t1_count = v_count_p[t1_v]
```

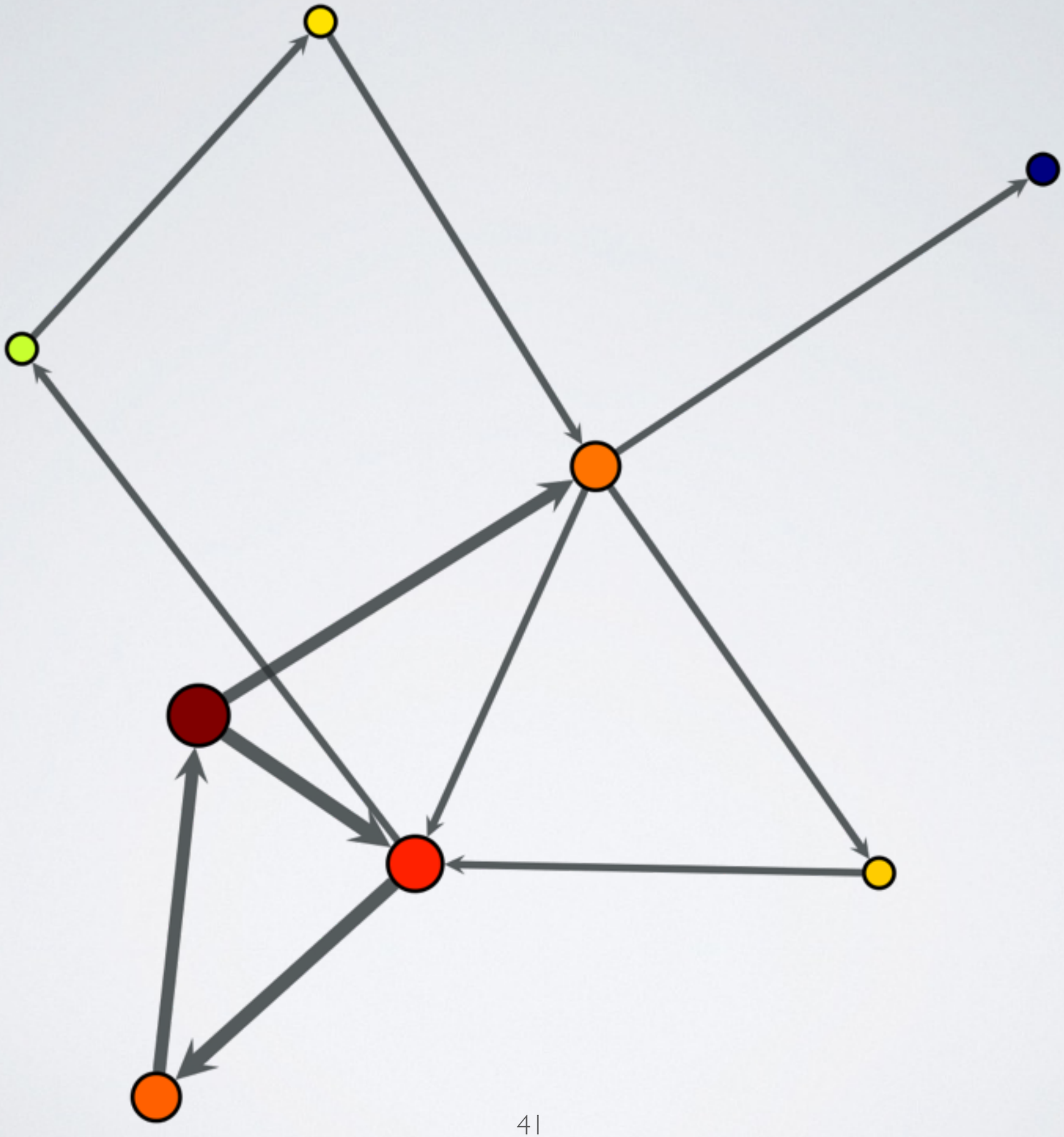
SFDF LAYOUT

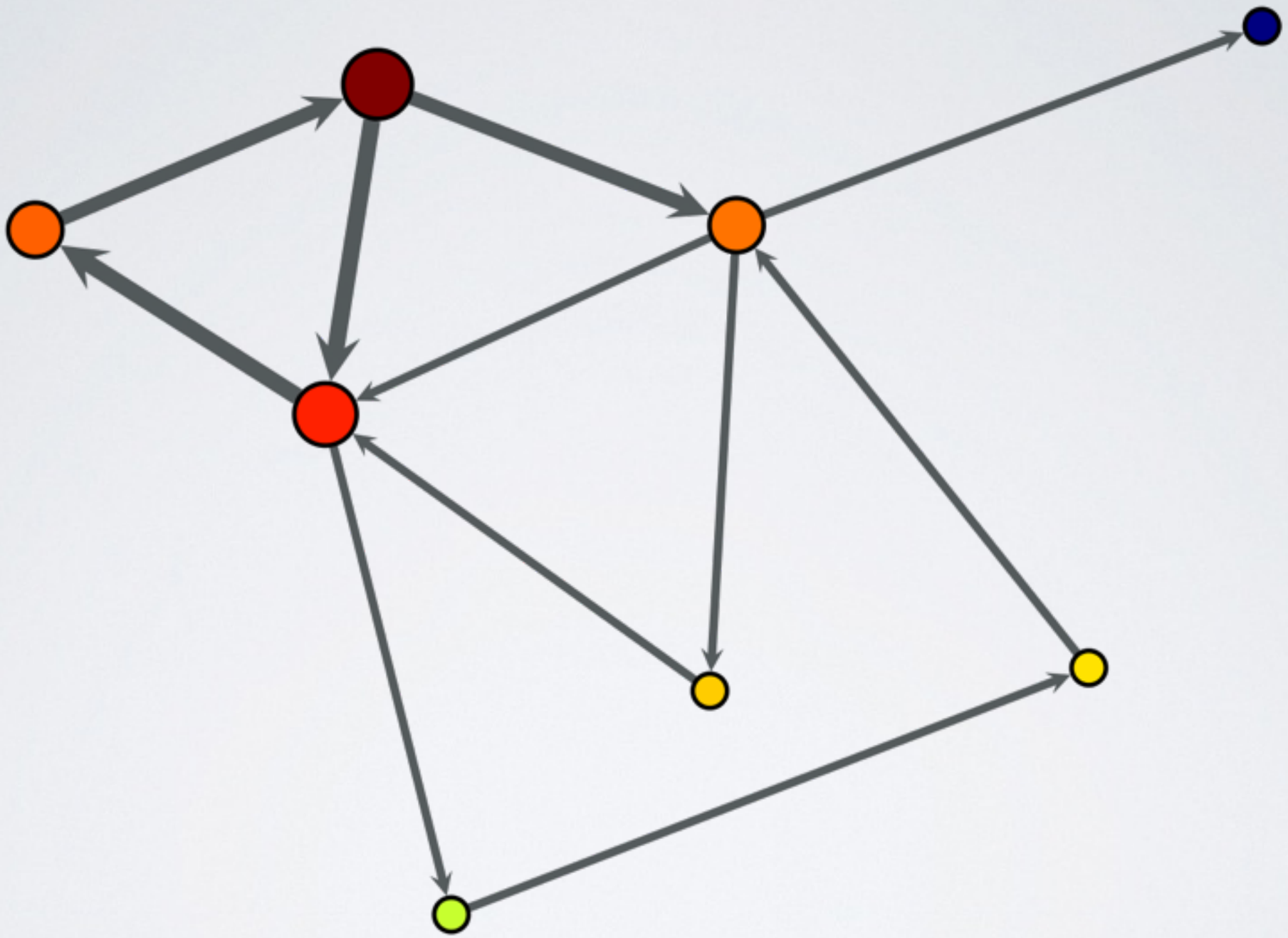
```
gt.graph_draw(
      …
      pos = gt.sfdp_layout(g),
)
```

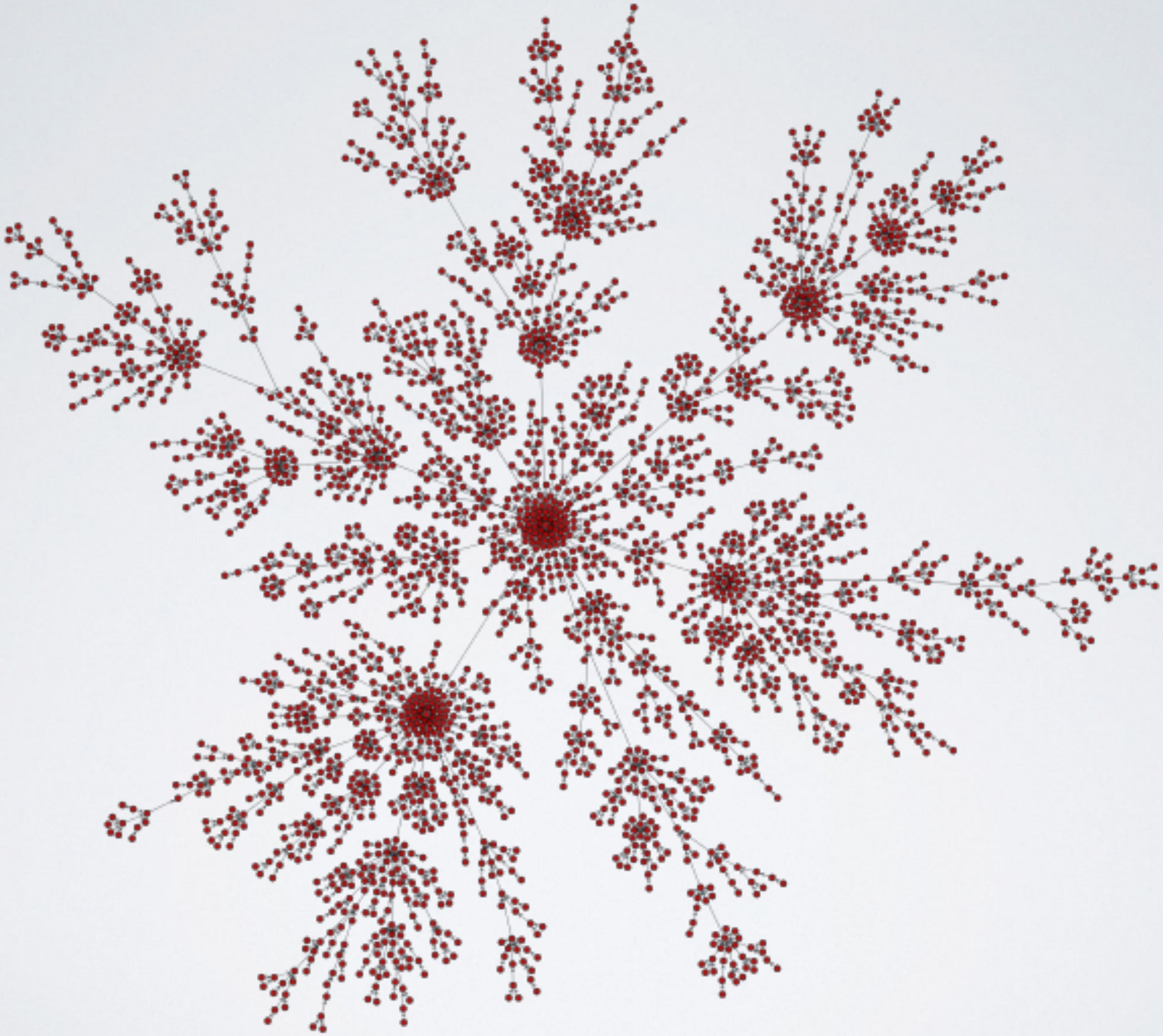


```
gt.graph_draw(
      …
      pos = gt.sfdp_layout(
            g, eweight=e_count_p
      ),
)
```

```
gt.graph_draw(
      …
      pos = gt.sfdp_layout(
            g,
            eweight=e_count_p, vweight=v_count_p
      ),
)
```

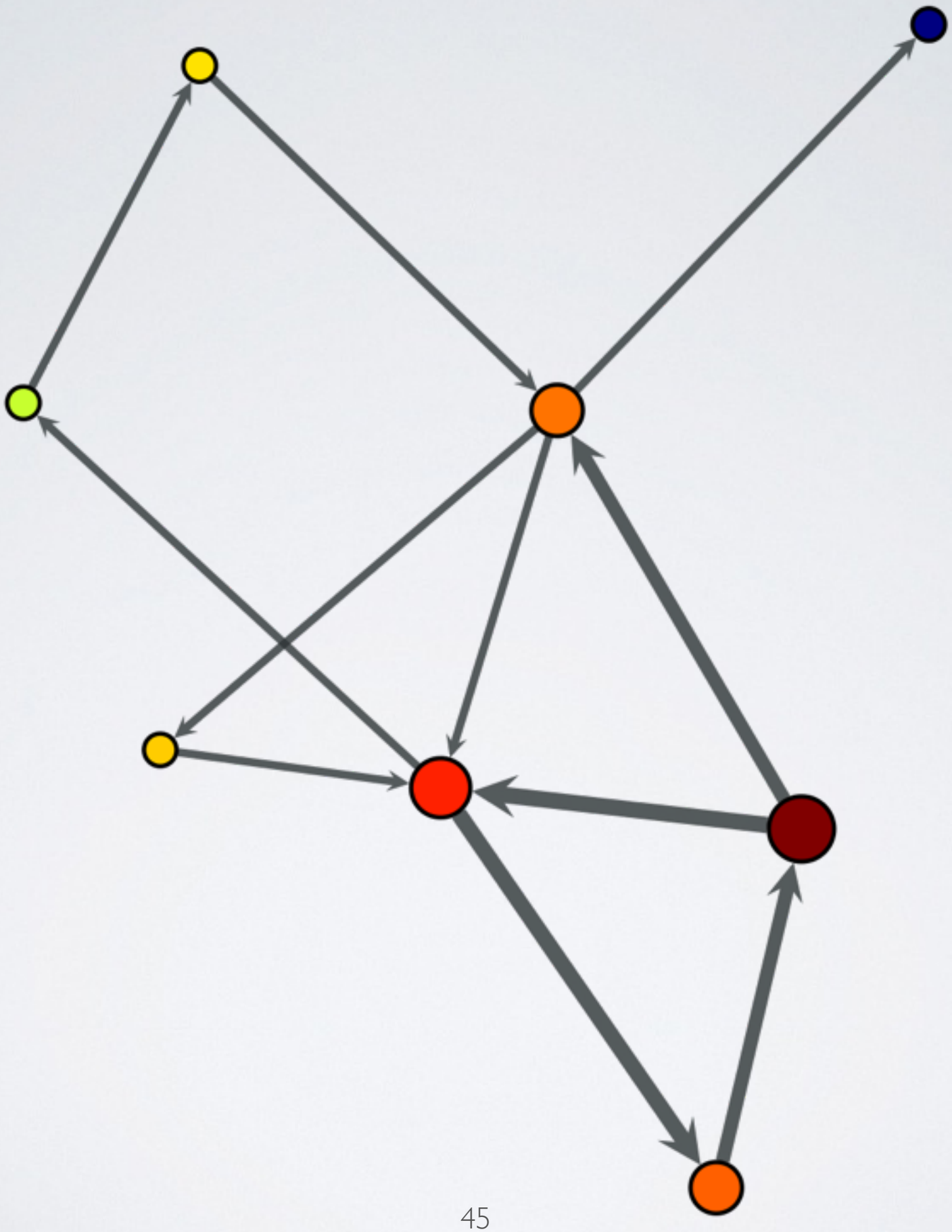


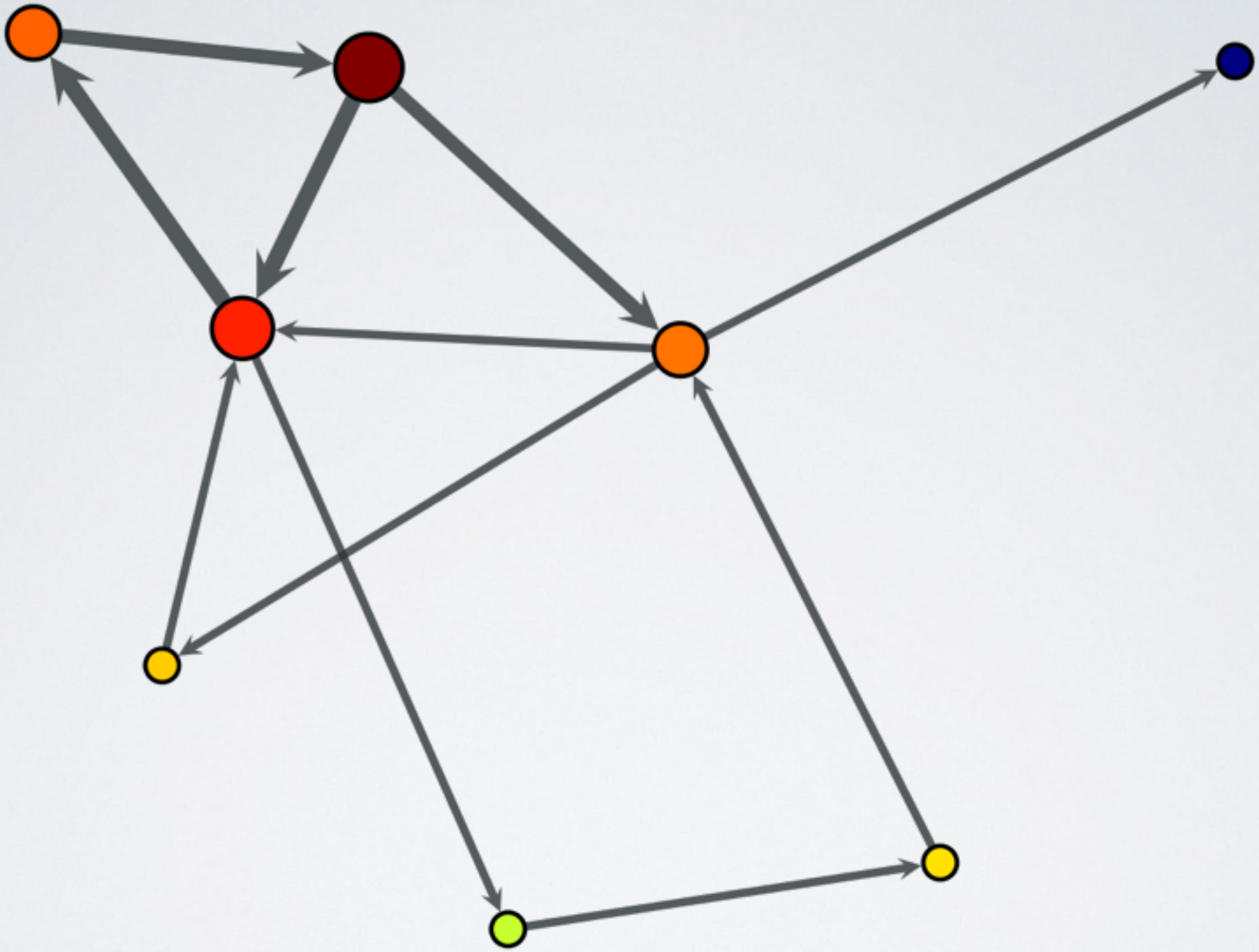


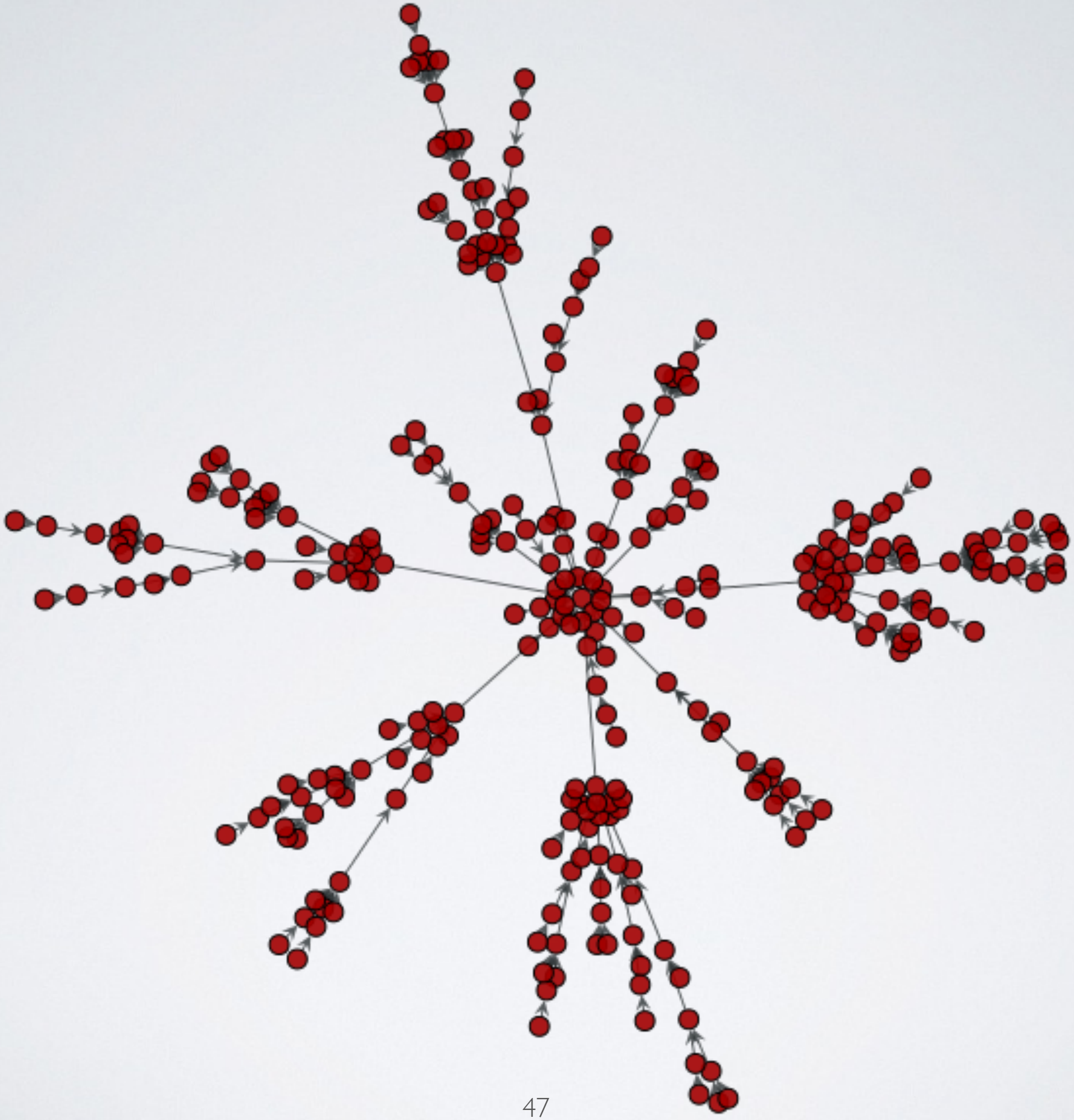


FR LAYOUT

```
gt.graph_draw(
      …
      pos = gt.fruchterman_reingold_layout(g),
)

gt.graph_draw(
      …
      pos = gt.fruchterman_reingold_layout(
            g, weight=e_count_p
      ),
)
```

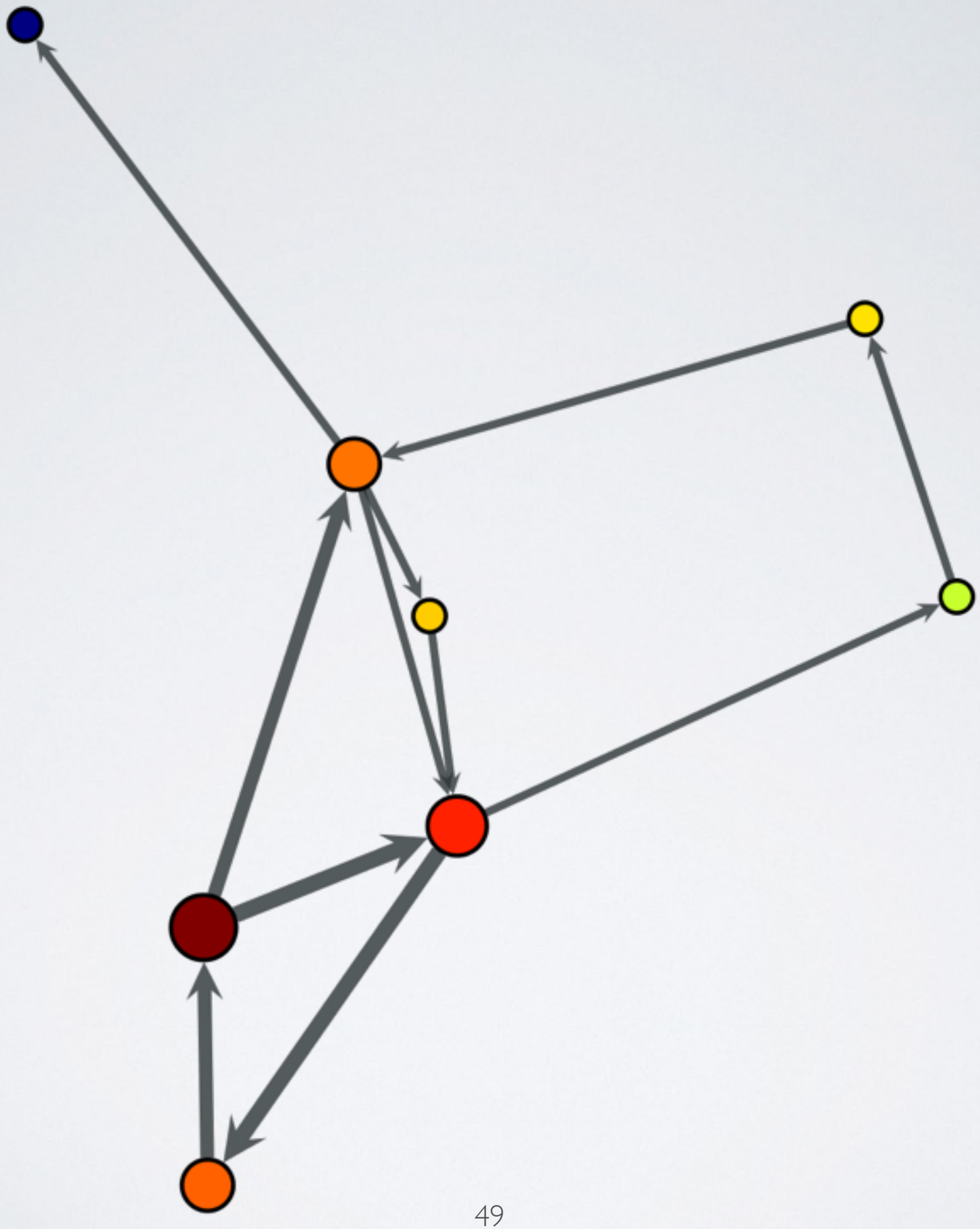


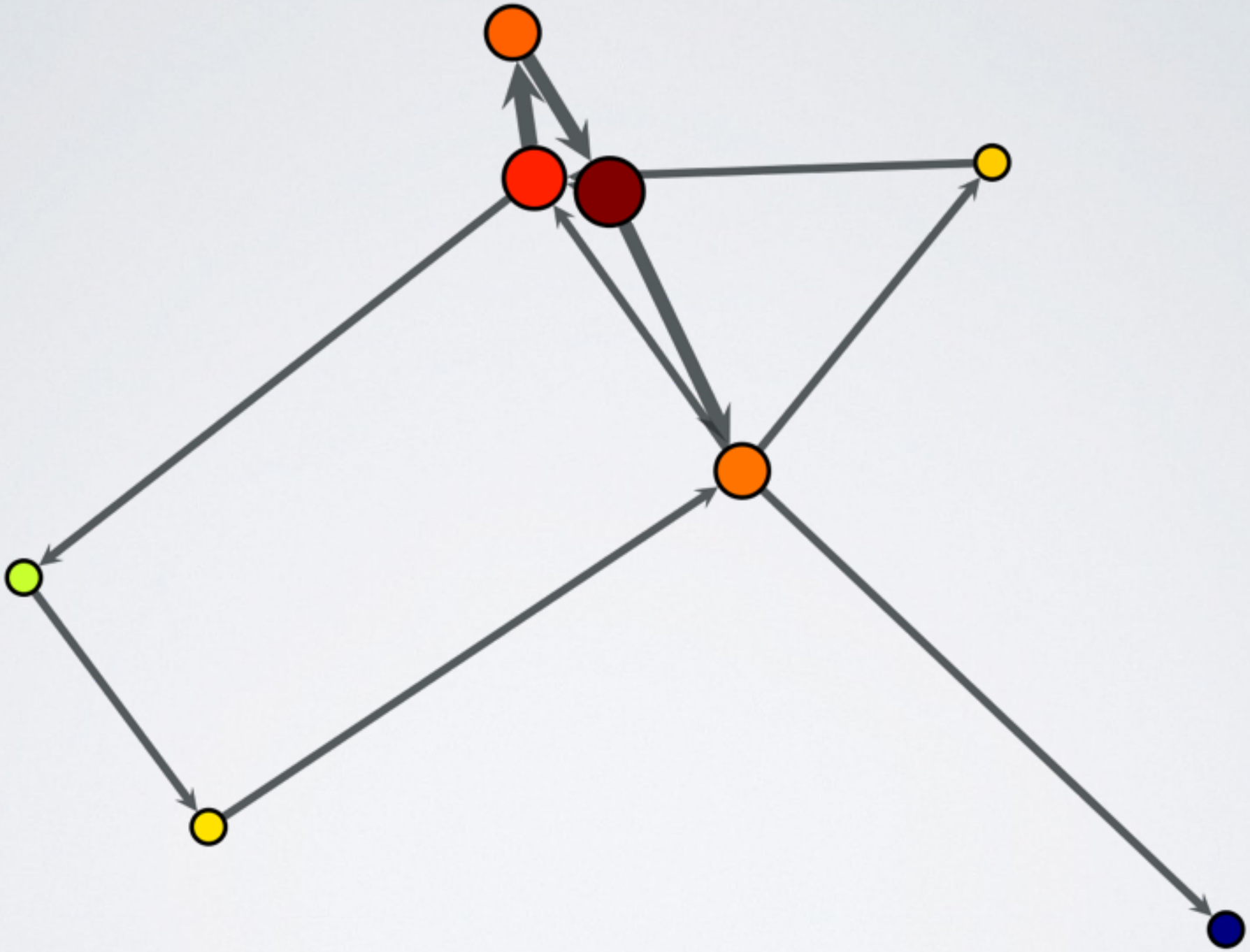


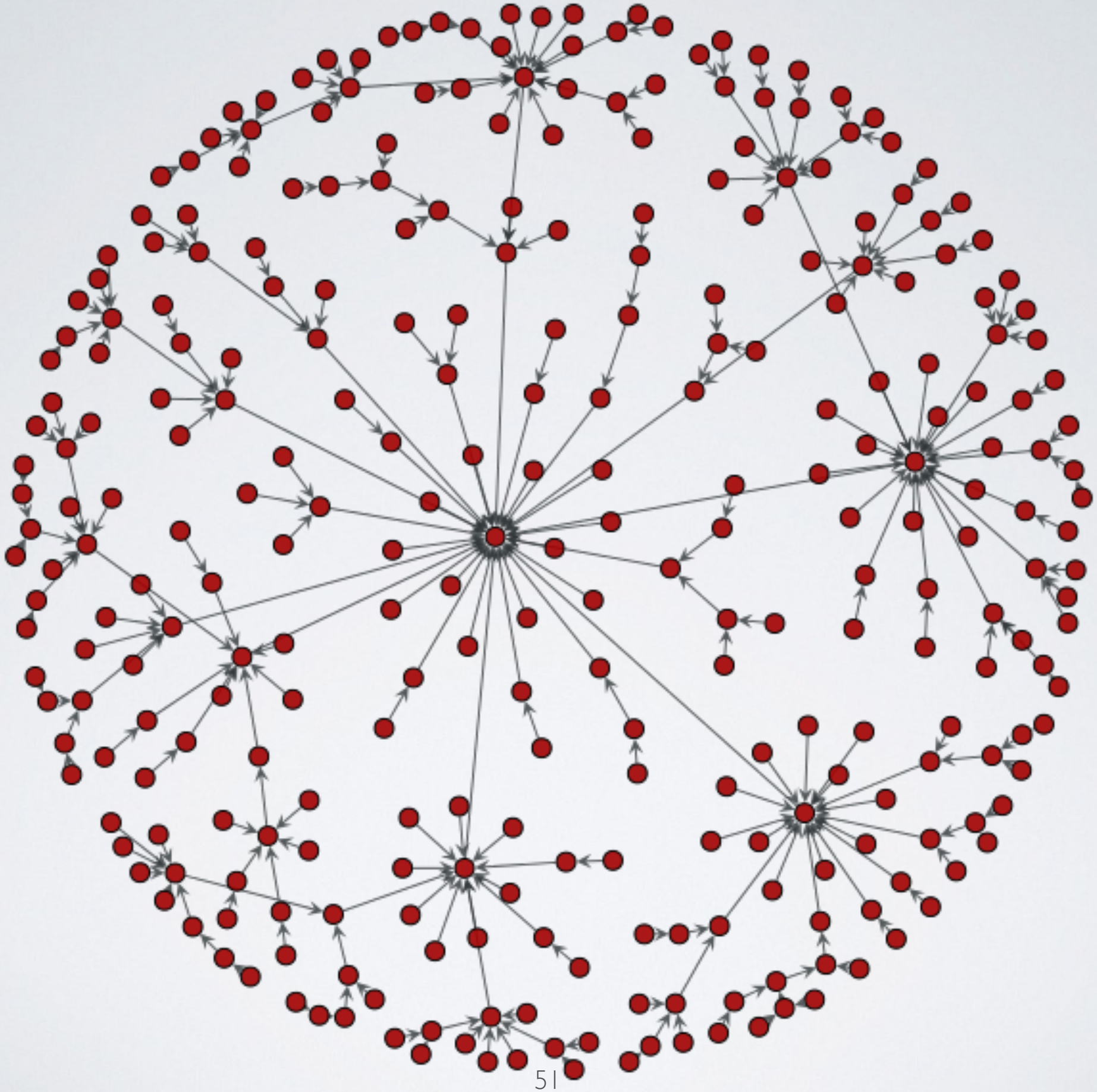


ARF LAYOUT

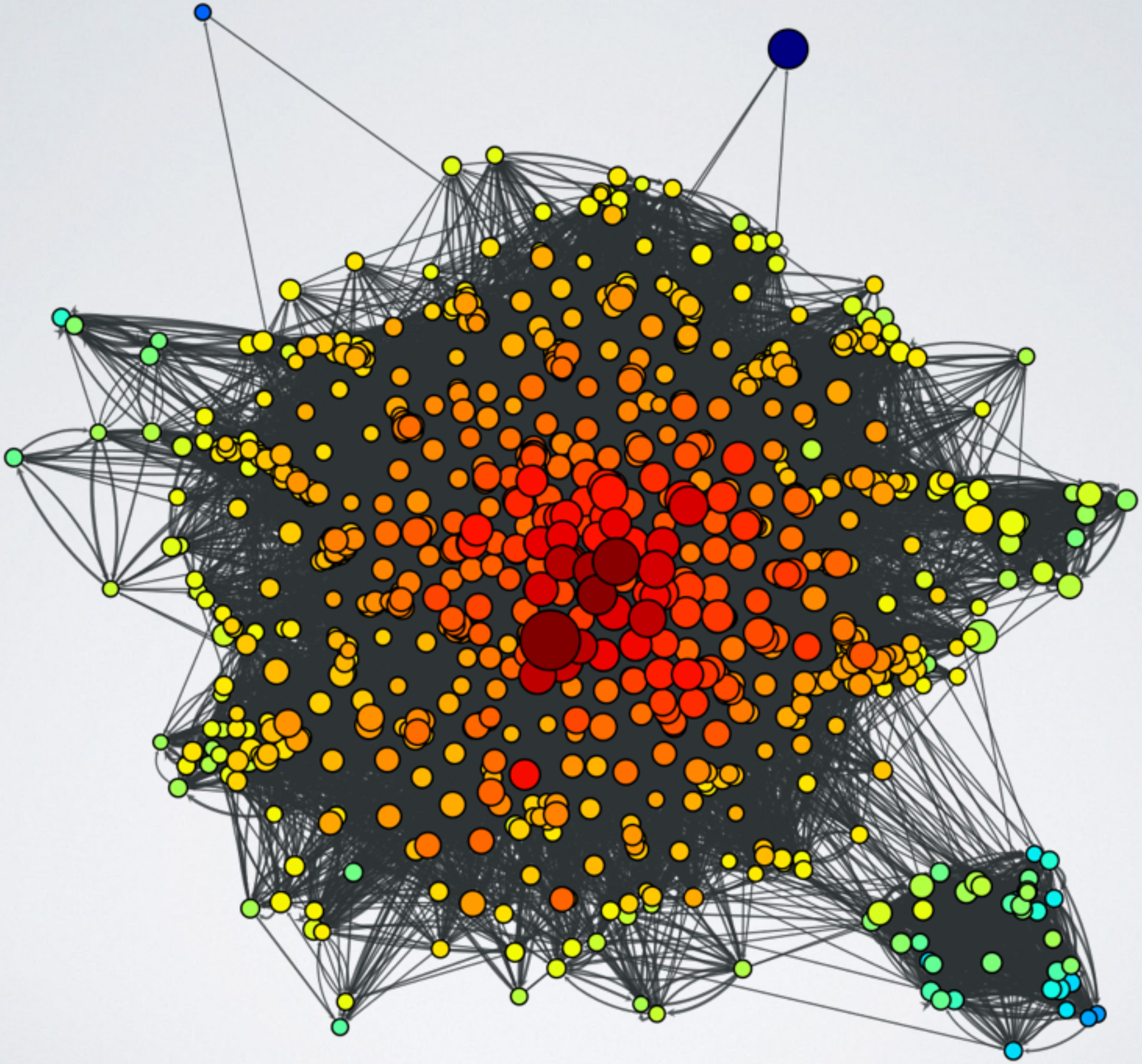
```
gt.graph_draw(
      …
      pos = gt.arf_layout(g),
)

gt.graph_draw(
      …
      pos = gt.arf_layout(
            g, weight=e_count_p
      ),
)
```





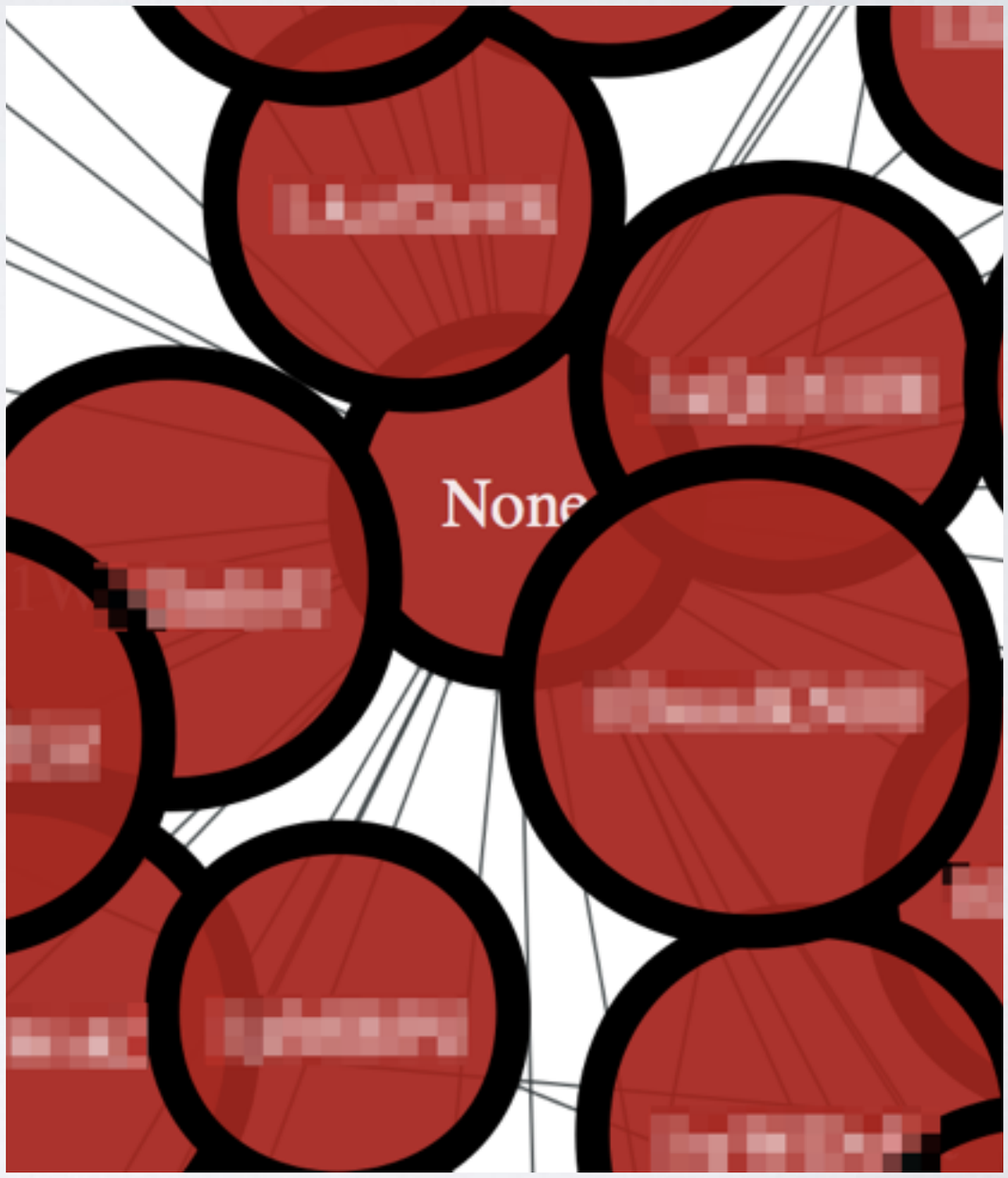


MY GRAPH



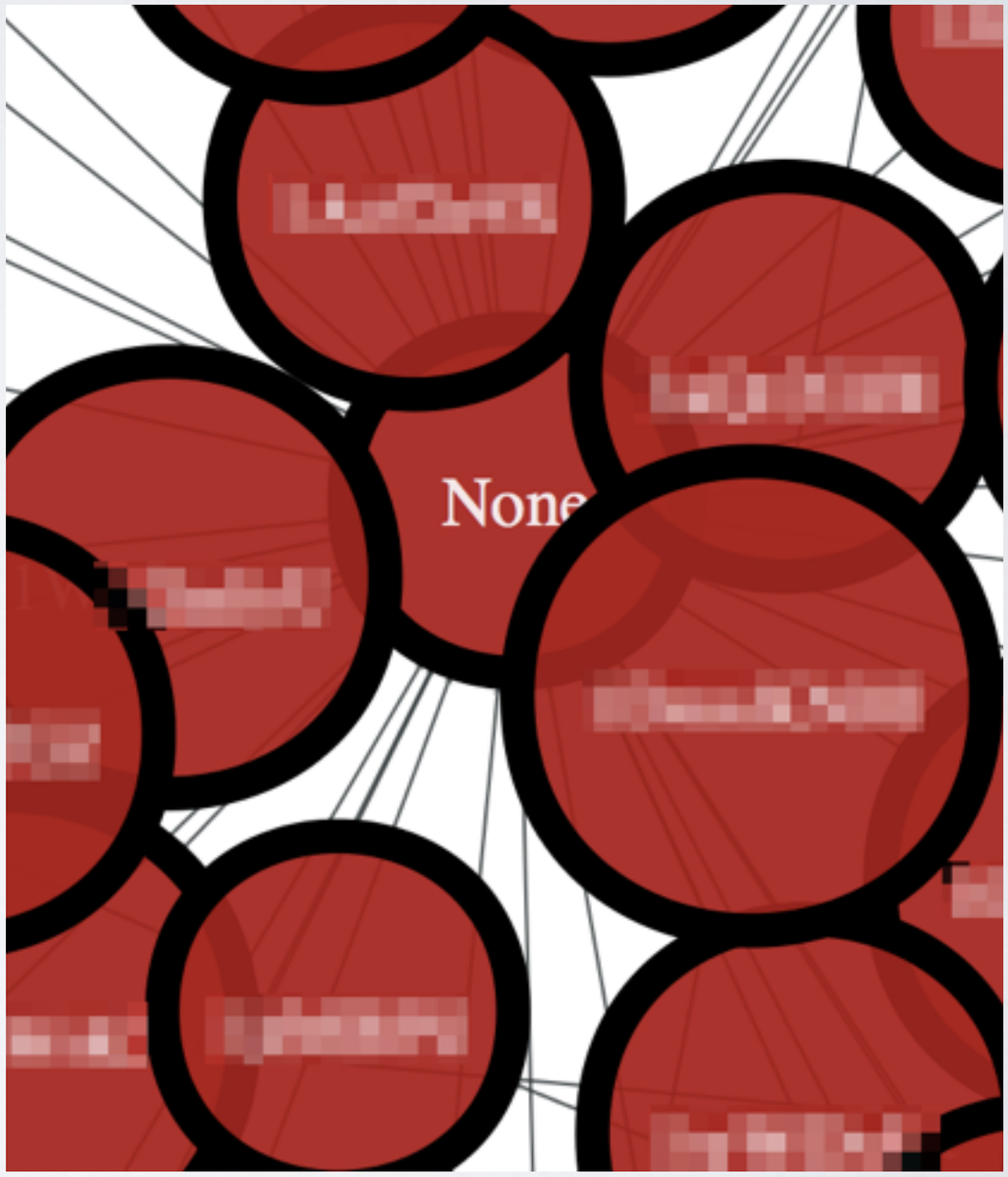
CONCLUSION

CONCLUSION



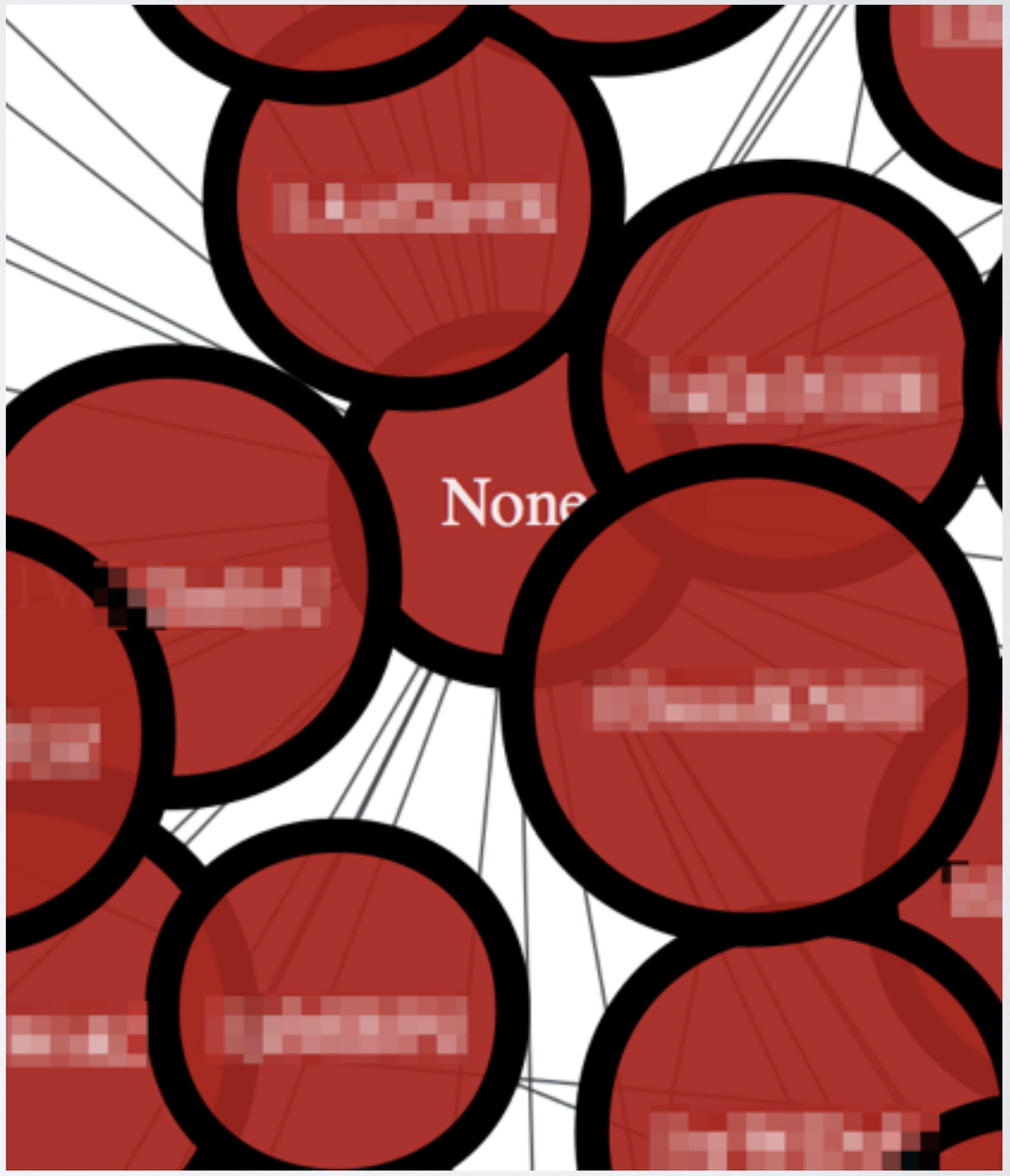
CONCLUSION

- Define problem in graphic form.



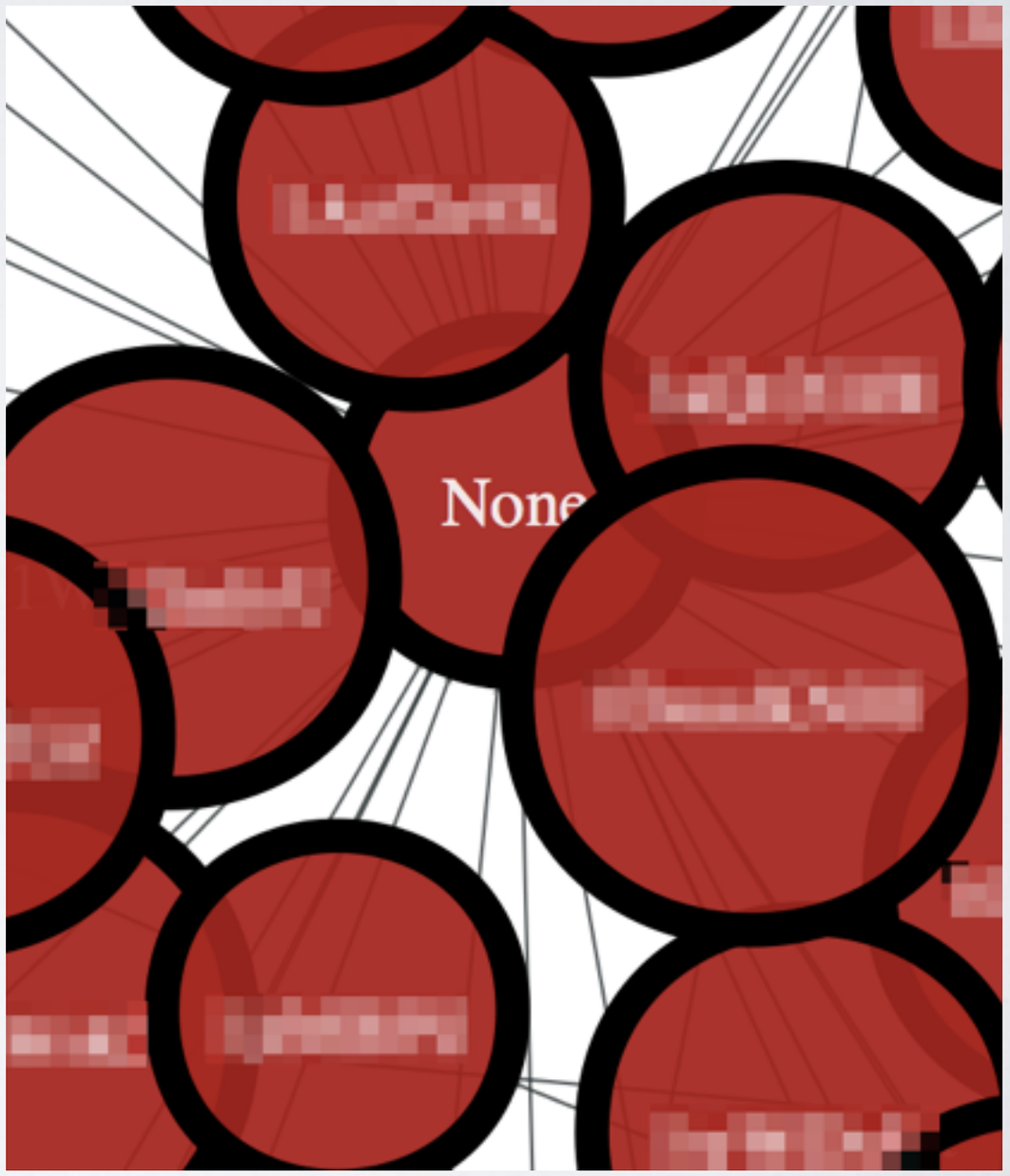
CONCLUSION

- Define problem in graphic form.
- Parse raw data.
  - Watch out!
    Your data will bite you. →



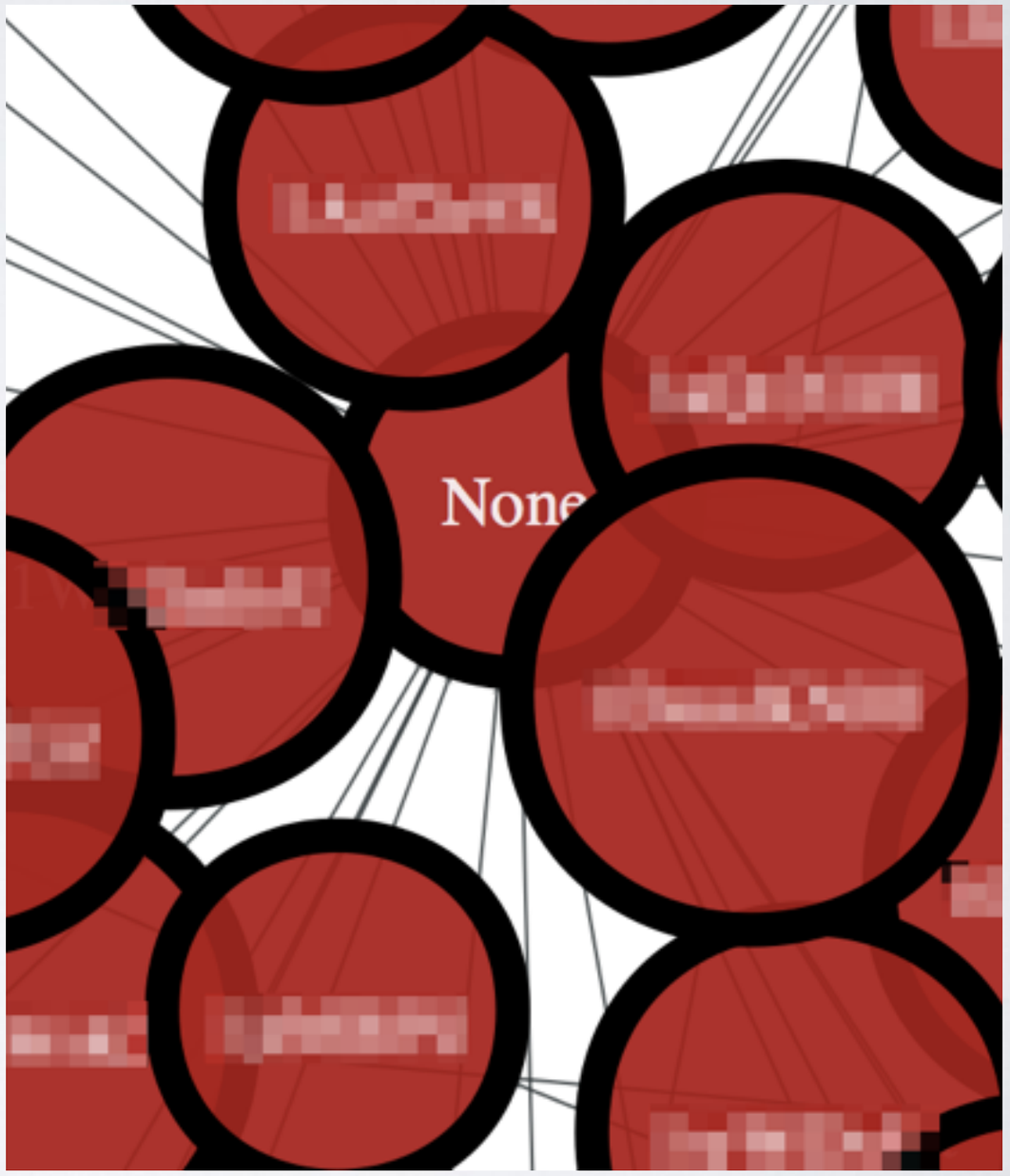
CONCLUSION

- Define problem in graphic form.
- Parse raw data.
  - Watch out!
    Your data will bite you. →
- Visualize to understand.



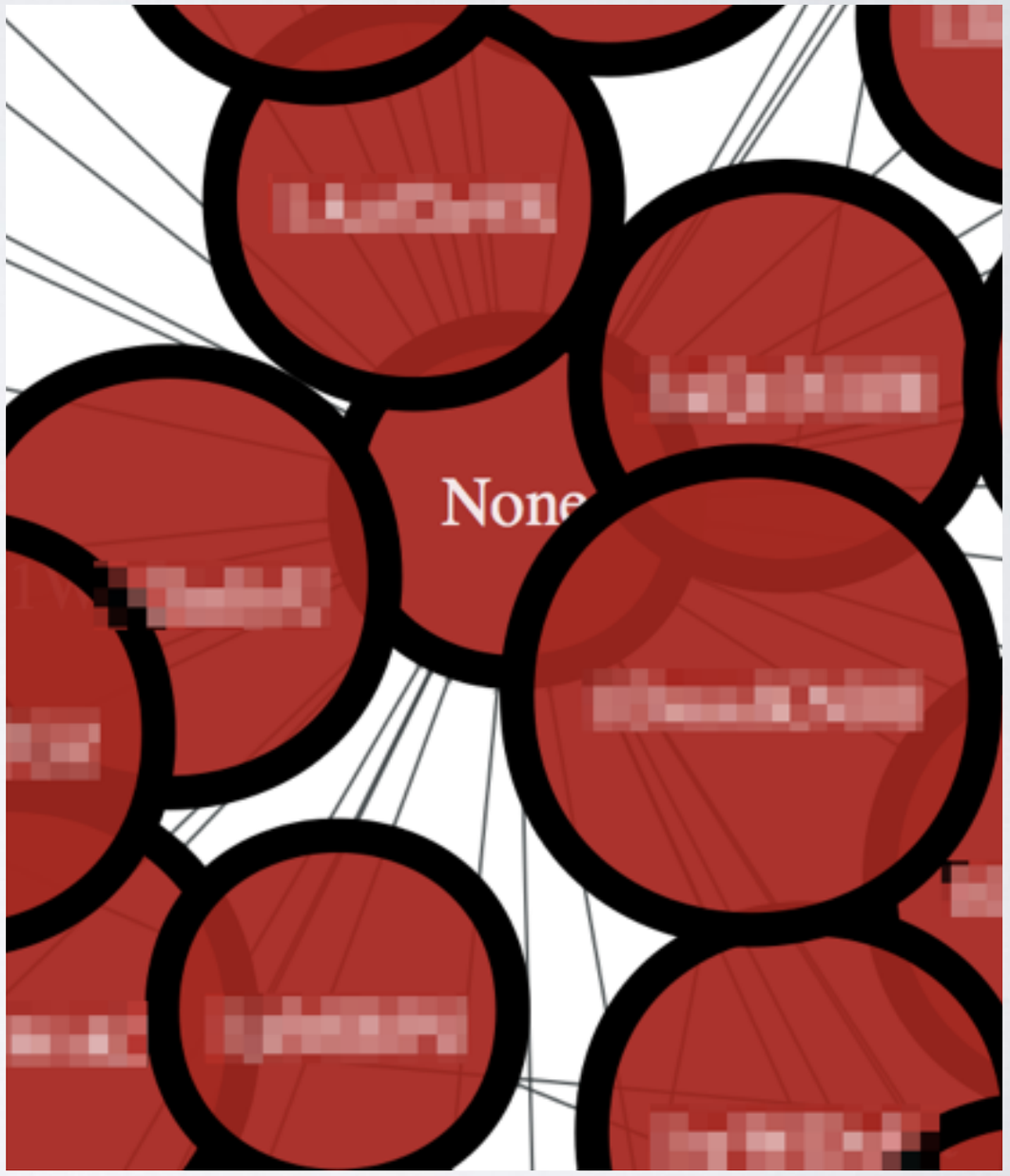
CONCLUSION

- Define problem in graphic form.
- Parse raw data.
  - Watch out!
    Your data will bite you. →
- Visualize to understand.
- Choose a proper algorithms.



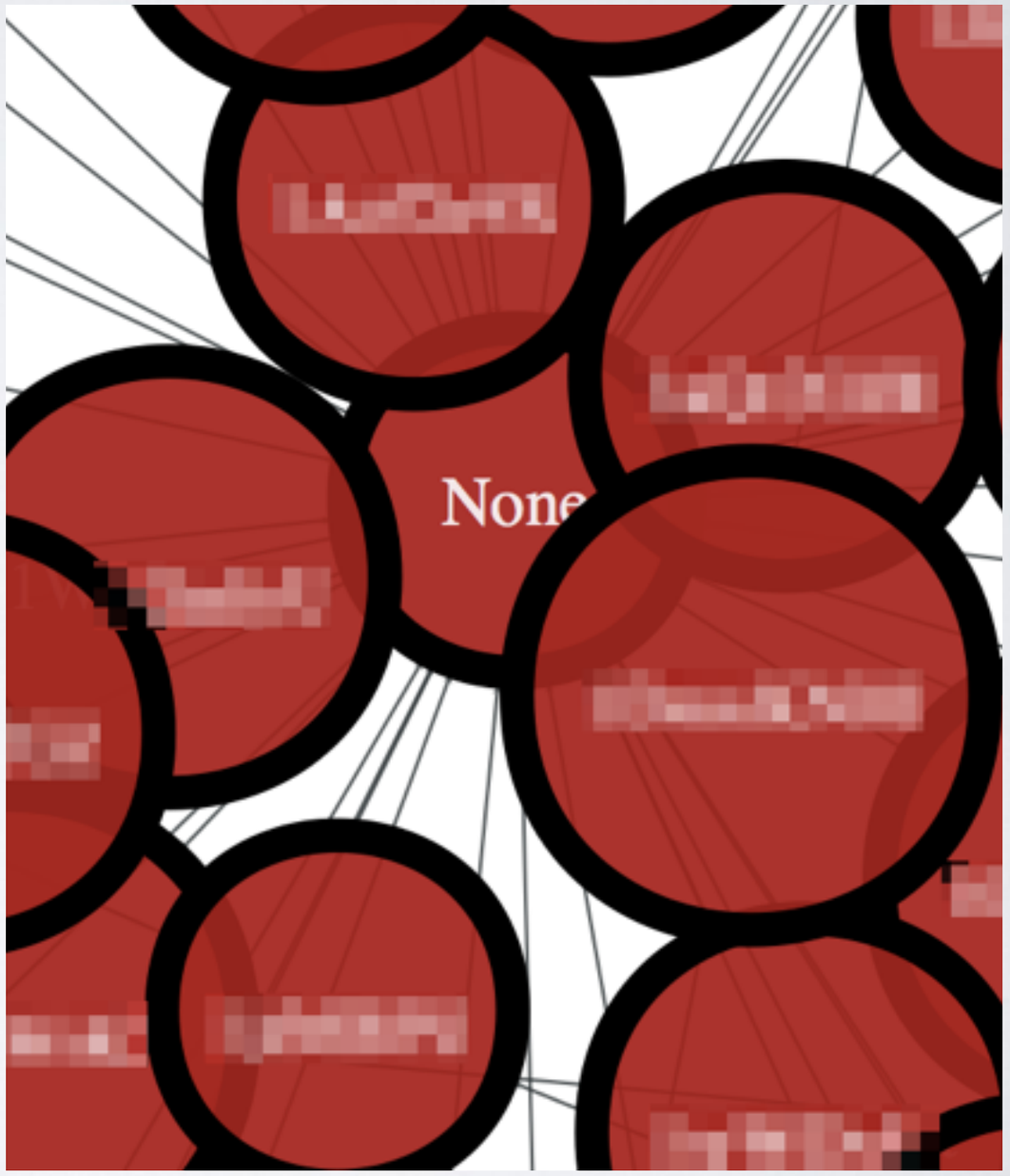
CONCLUSION

- Define problem in graphic form.
- Parse raw data.
  - Watch out!
    Your data will bite you. →
- Visualize to understand.
- Choose a proper algorithms.
- Filter data which interest you.



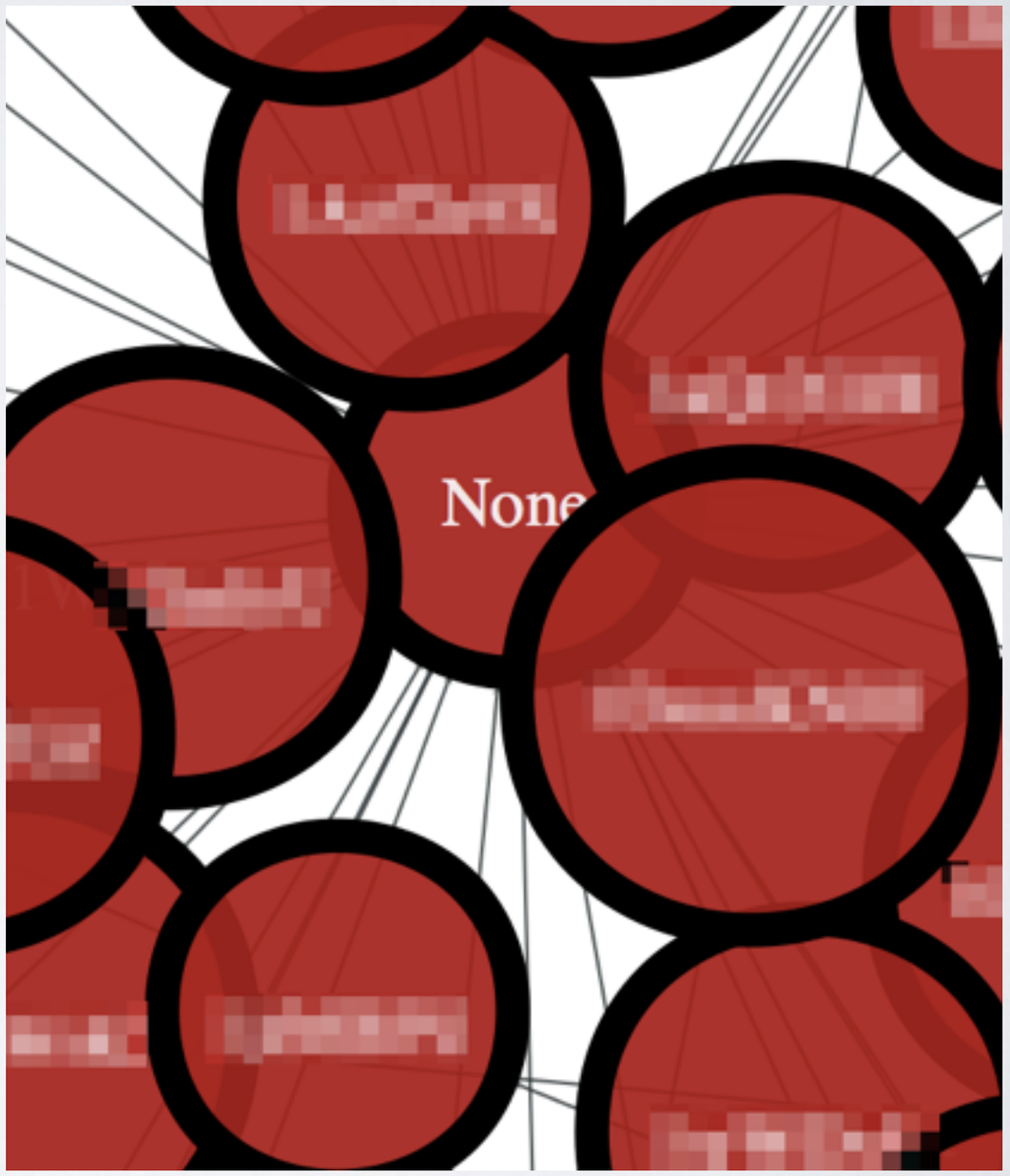
CONCLUSION

- Define problem in graphic form.
- Parse raw data.
  - Watch out!
    Your data will bite you. →
- Visualize to understand.
- Choose a proper algorithms.
- Filter data which interest you.
- Visualize again to convince.



CONCLUSION

- Define problem in graphic form.
- Parse raw data.
  - Watch out!
    Your data will bite you. →
- Visualize to understand.
- Choose a proper algorithms.
- Filter data which interest you.
- Visualize again to convince.
- mosky.tw



DEMO

COSCUP 2014

2014.07.19 - 2014.07.20 | Academia Sinica, Taipei, Taiwan

LINKS

- Quick start using graph-tool
  http://graph-tool.skewed.de/static/doc/quickstart.html

- Learn more about Graph object
  http://graph-tool.skewed.de/static/doc/graph_tool.html

- The possible property value types
  http://graph-tool.skewed.de/static/doc/graph_tool.html#graph_tool.PropertyMap

- Graph drawing and layout
  http://graph-tool.skewed.de/static/doc/draw.html

- Available subpackages - Graph-Tool
  http://graph-tool.skewed.de/static/doc/graph_tool.html#available-subpackages

- Centrality - Wiki
  http://en.wikipedia.org/wiki/Centrality

- NumPy Reference
  http://docs.scipy.org/doc/numpy/reference/