

# Identifying Nashville's Popular Pythonistas with NetworkX


---

NOVEMBER 16, 2017

STEPHEN BAILEY

[GITHUB.COM/STKBAILEY/NASHNETX](https://github.com/STKBAILEY/NASHNETX)





Who should I  
get to know?

What groups  
should I attend?

Where should  
I advertise?

g



# Popular Pythonistas

---

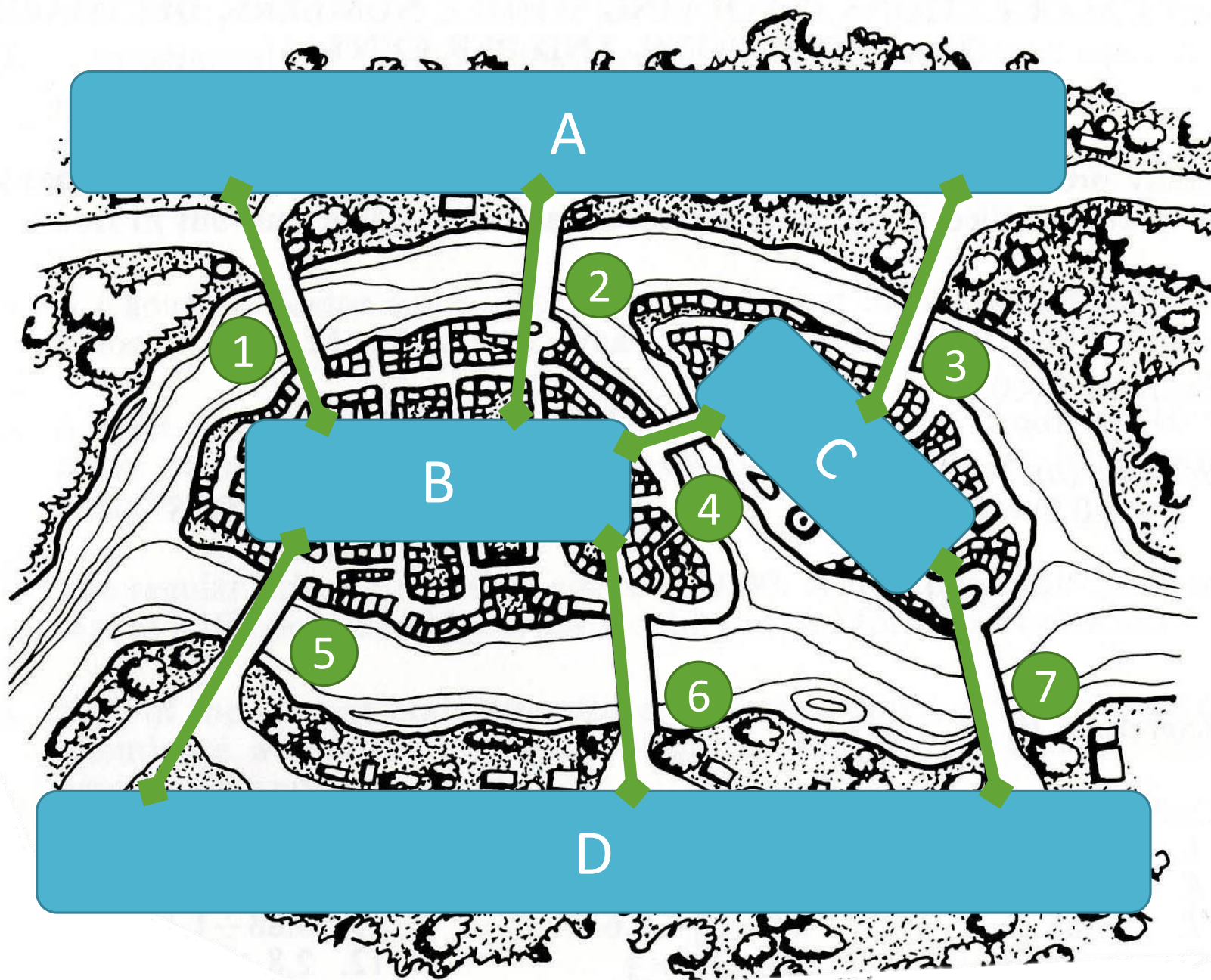
1. What is a network?
2. Building graphs
3. Who's who in PyNash?



Can you traverse every bridge just once?





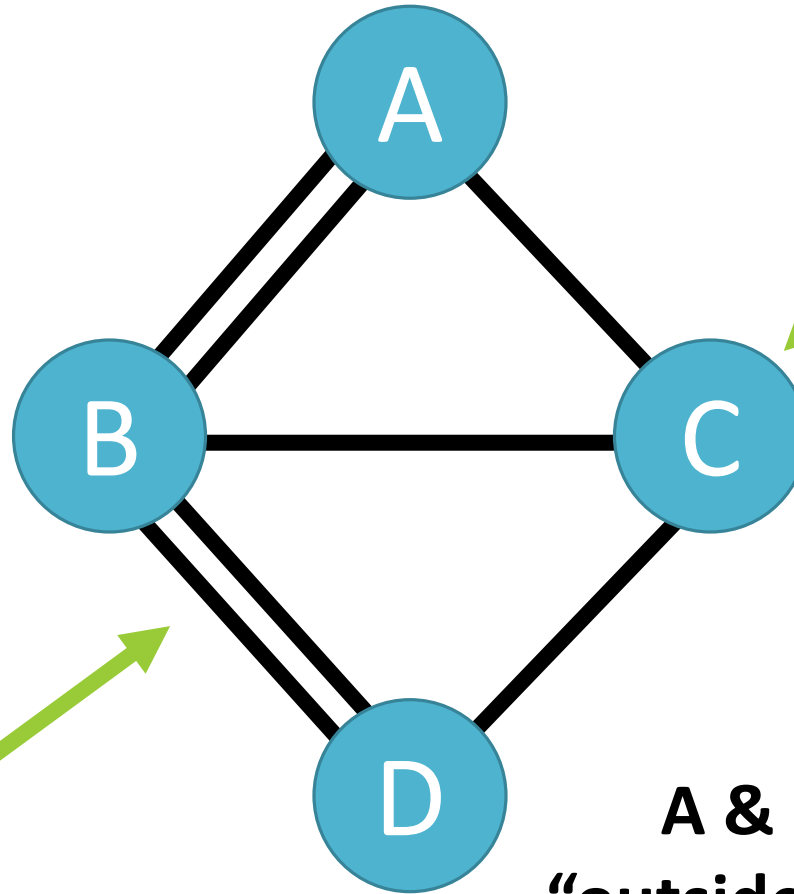


# Graph

**B has the most connections**

**Edge**

**Node**

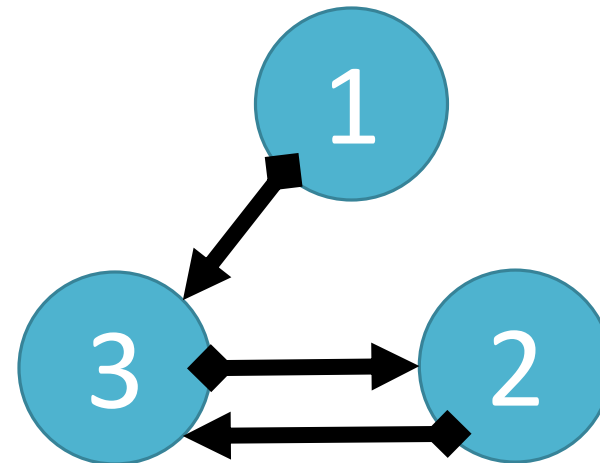
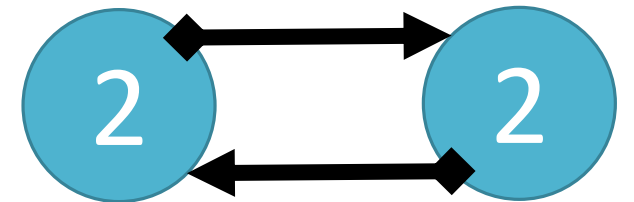
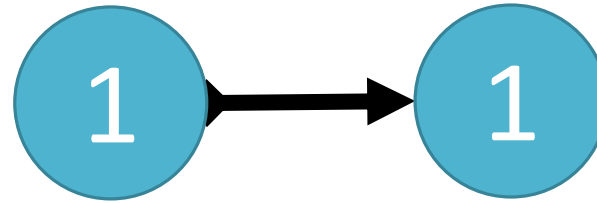


**A & D are on the “outside” – they are not connected to each other**

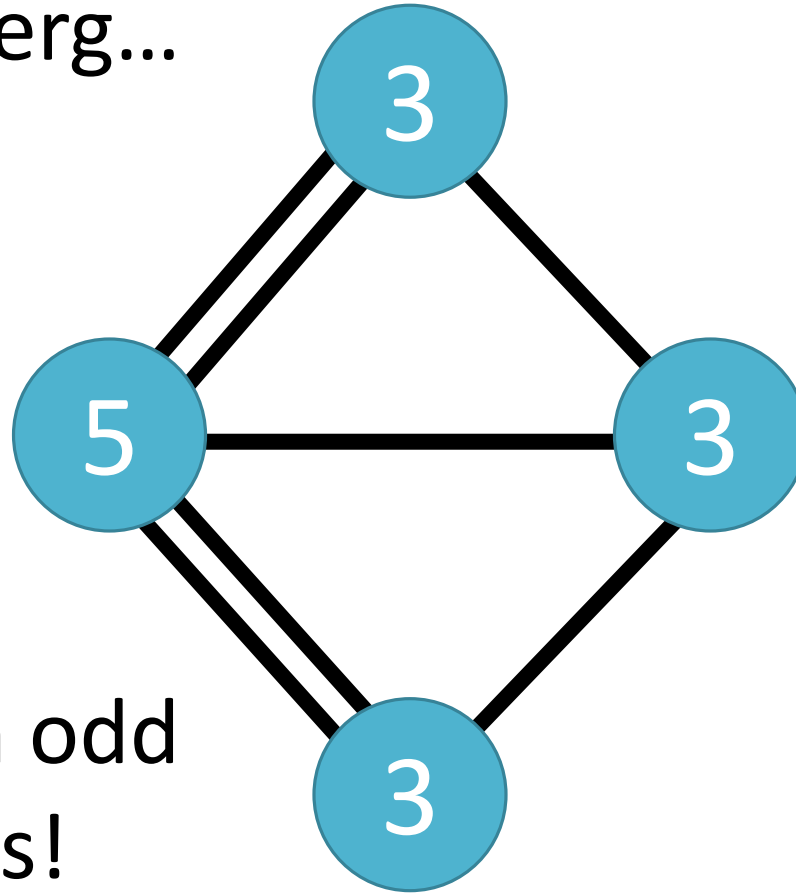
1.To travel into AND out of a node, requires 2 steps (or 4 or 6...).

2.Therefore... either:

- All nodes are even.
- Exactly two nodes are odd.



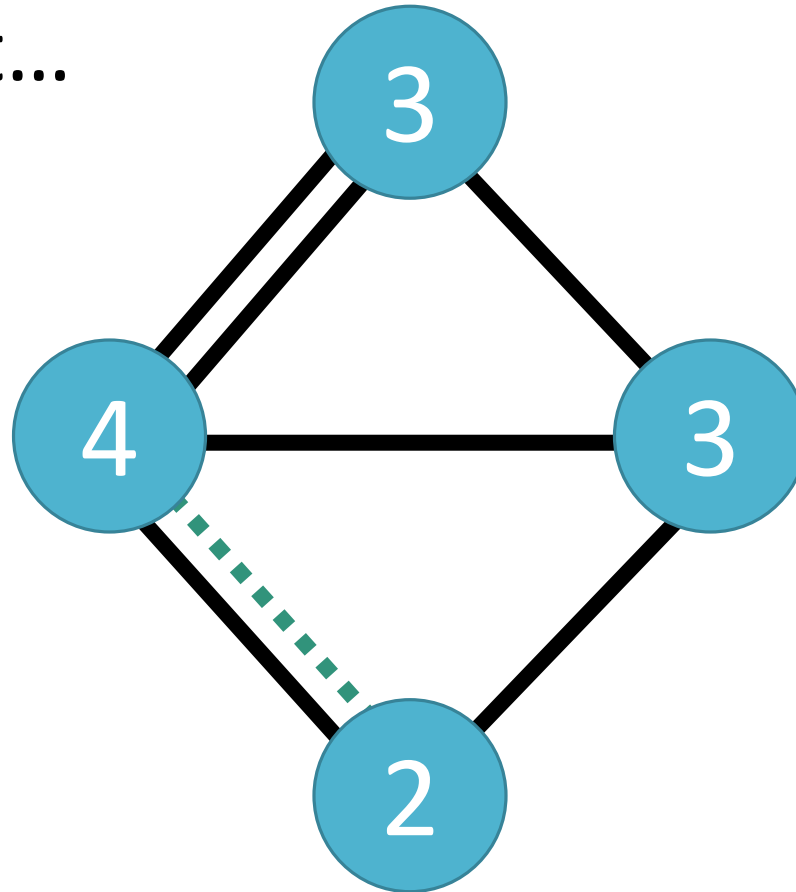
Back to Königsberg...



Four nodes with odd number of edges!



That's the ticket...



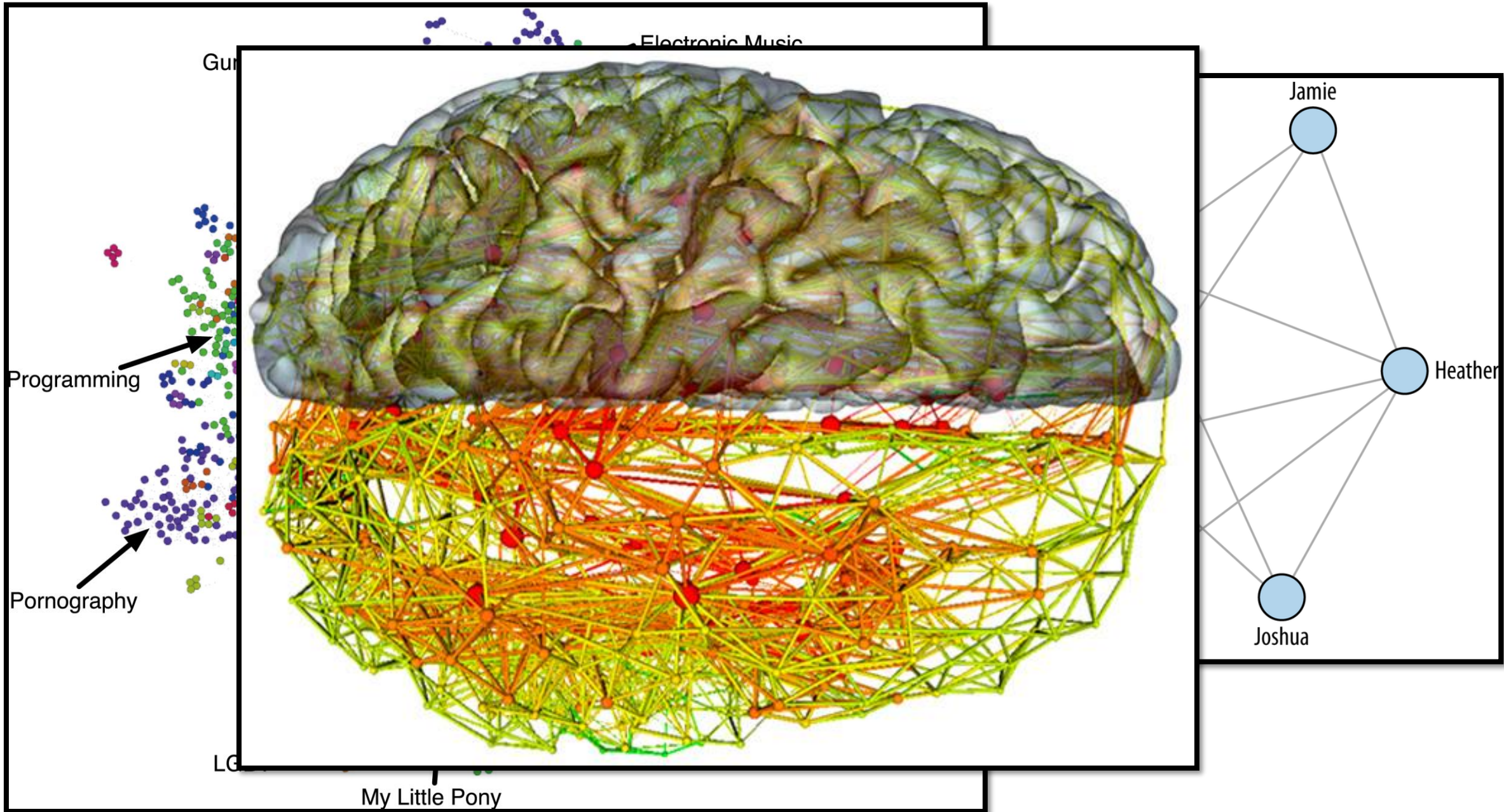
# Graphs

---

Useful for understanding **relationships**.

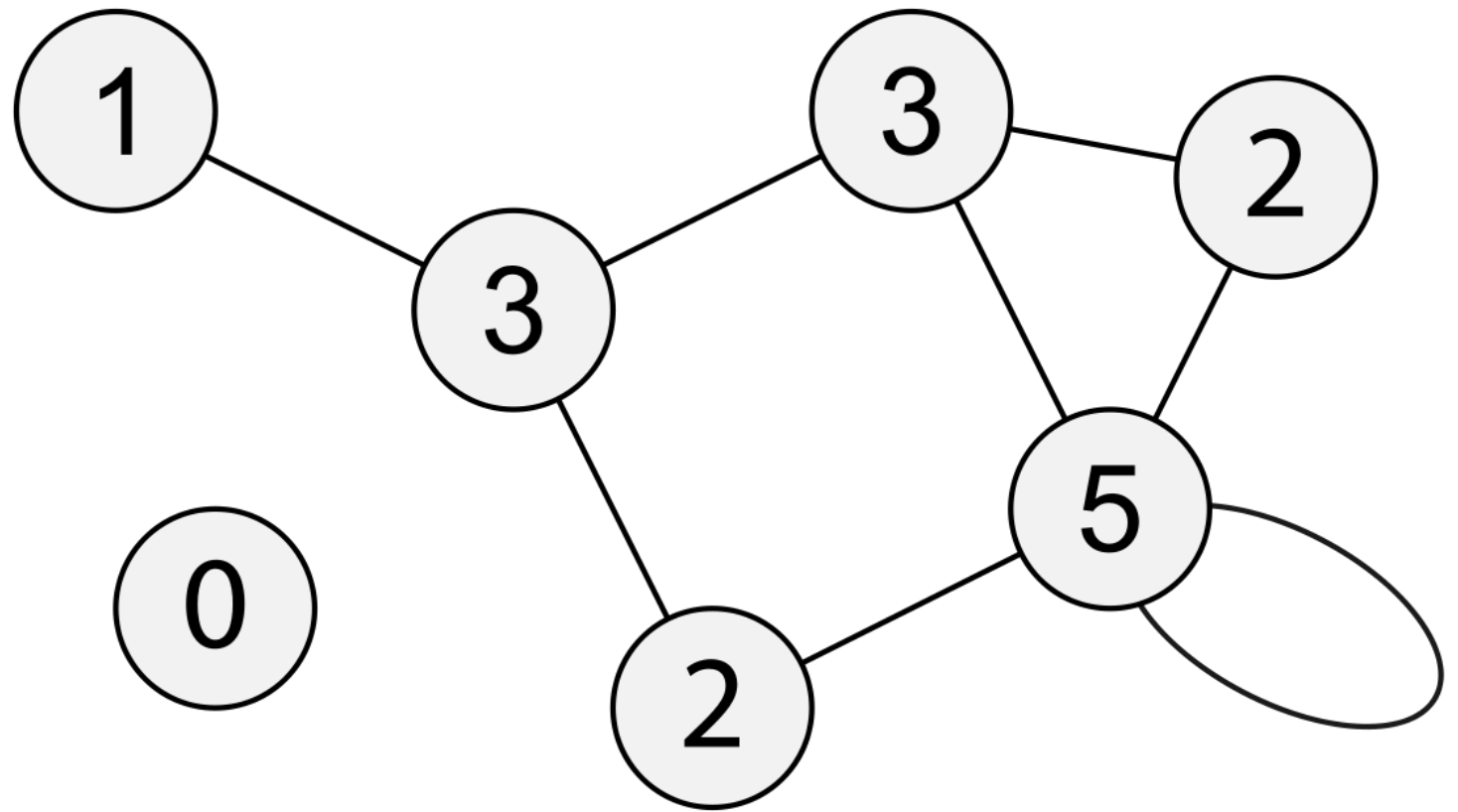
Describe **topology** (arrangement) of networks.

Many possible measurements to derive.



# Common Measures

## Degree

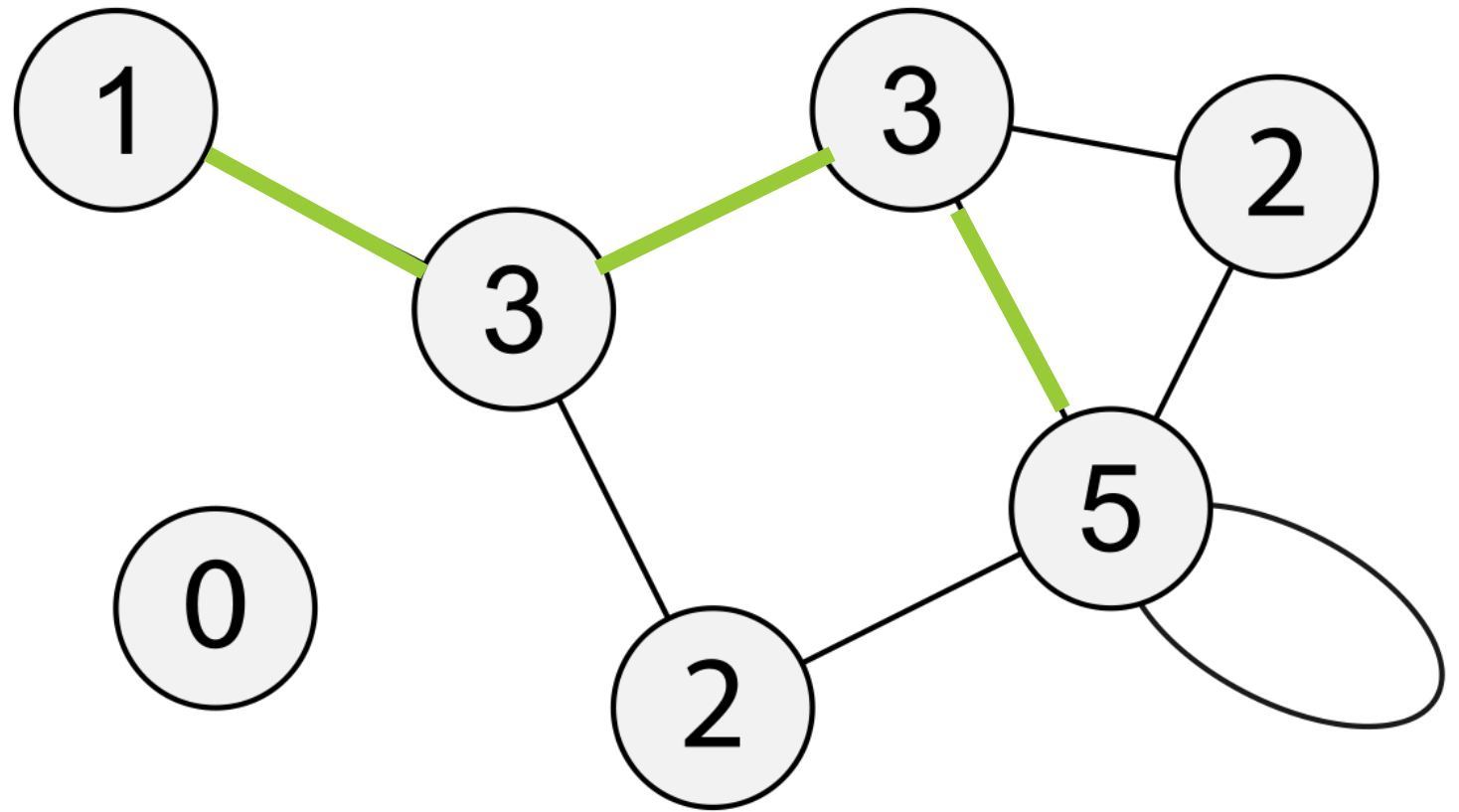


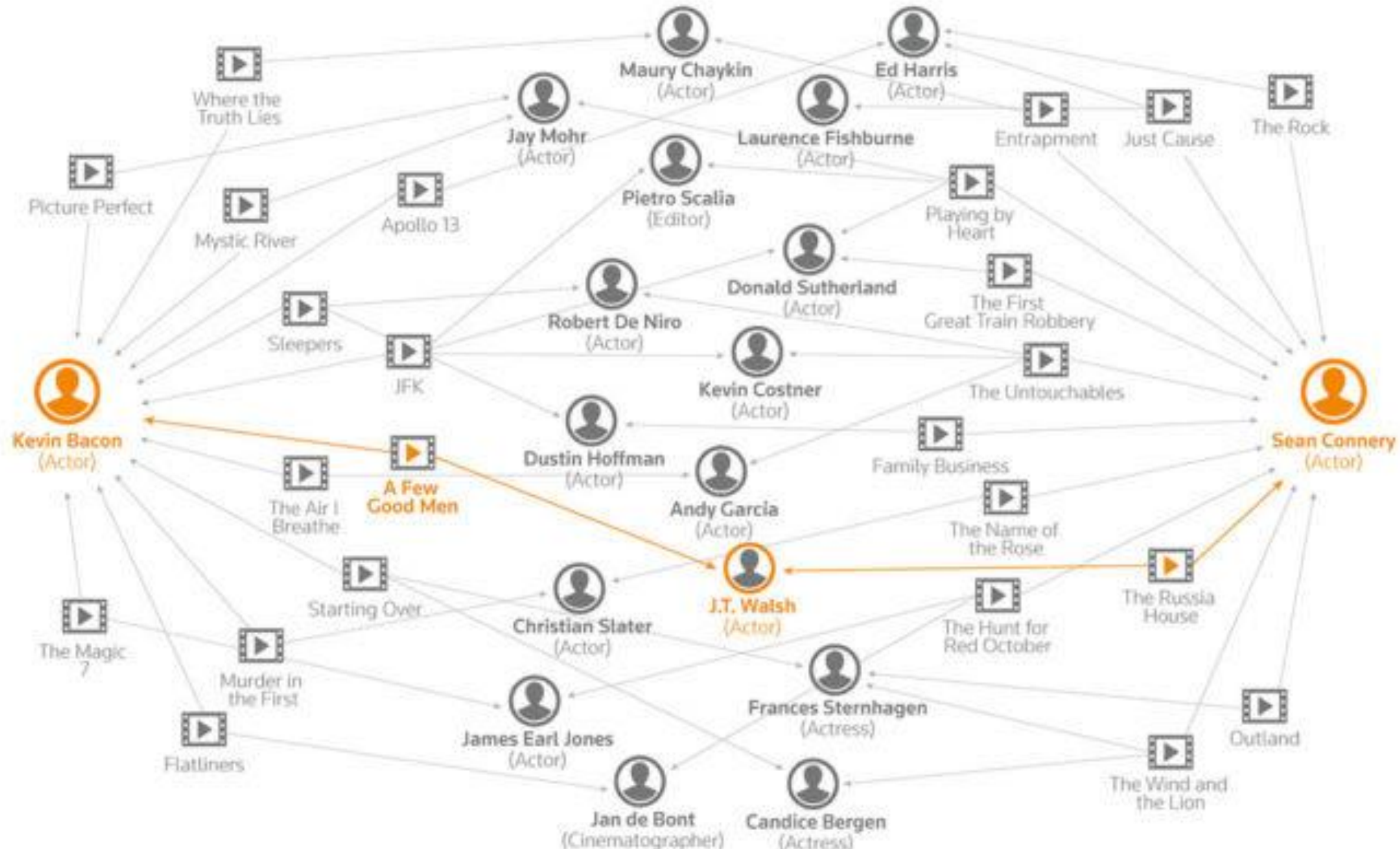


# Common Measures

Degree

Path Length



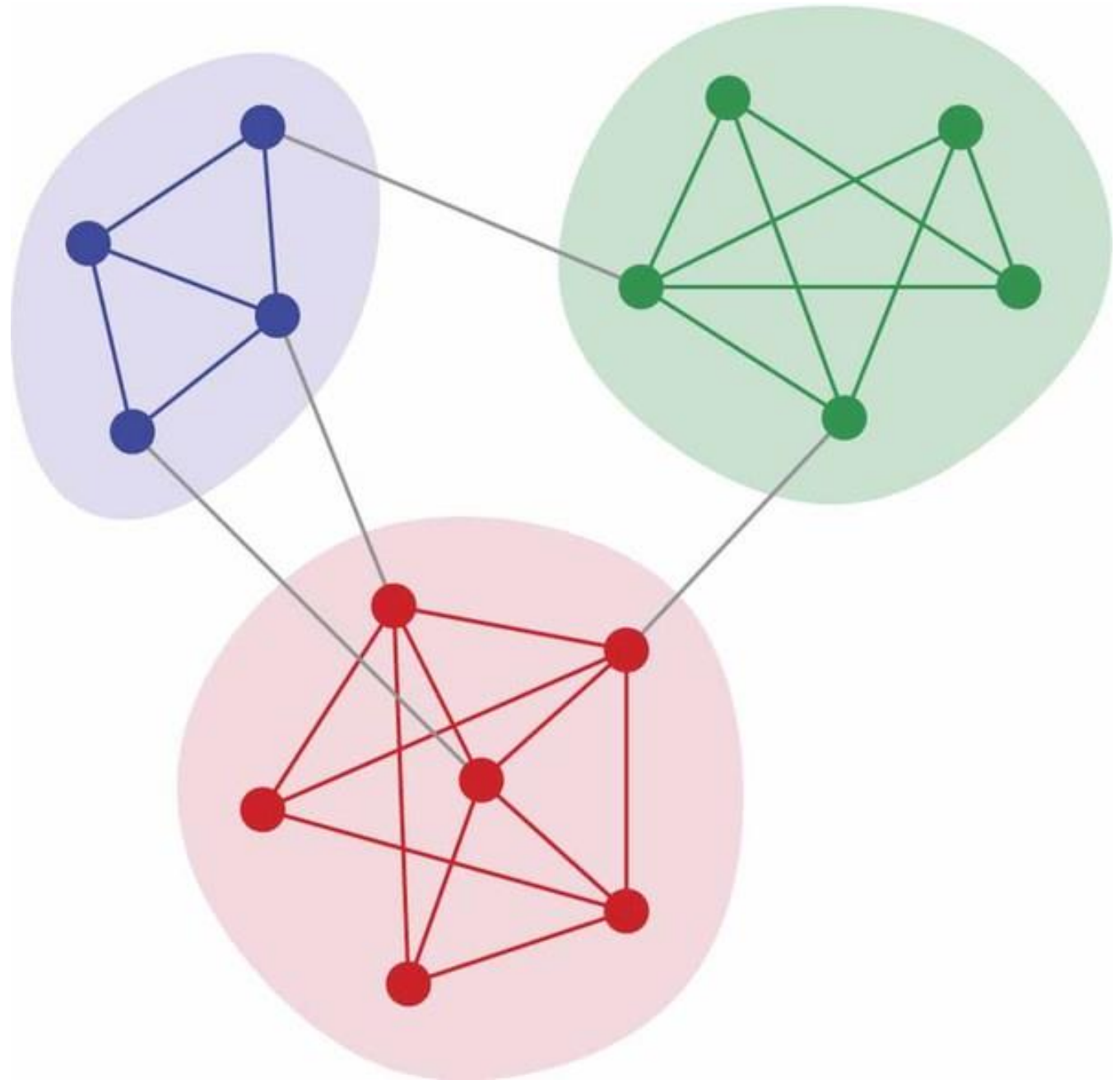


# Common Measures

Degree

Path Length

Clustering



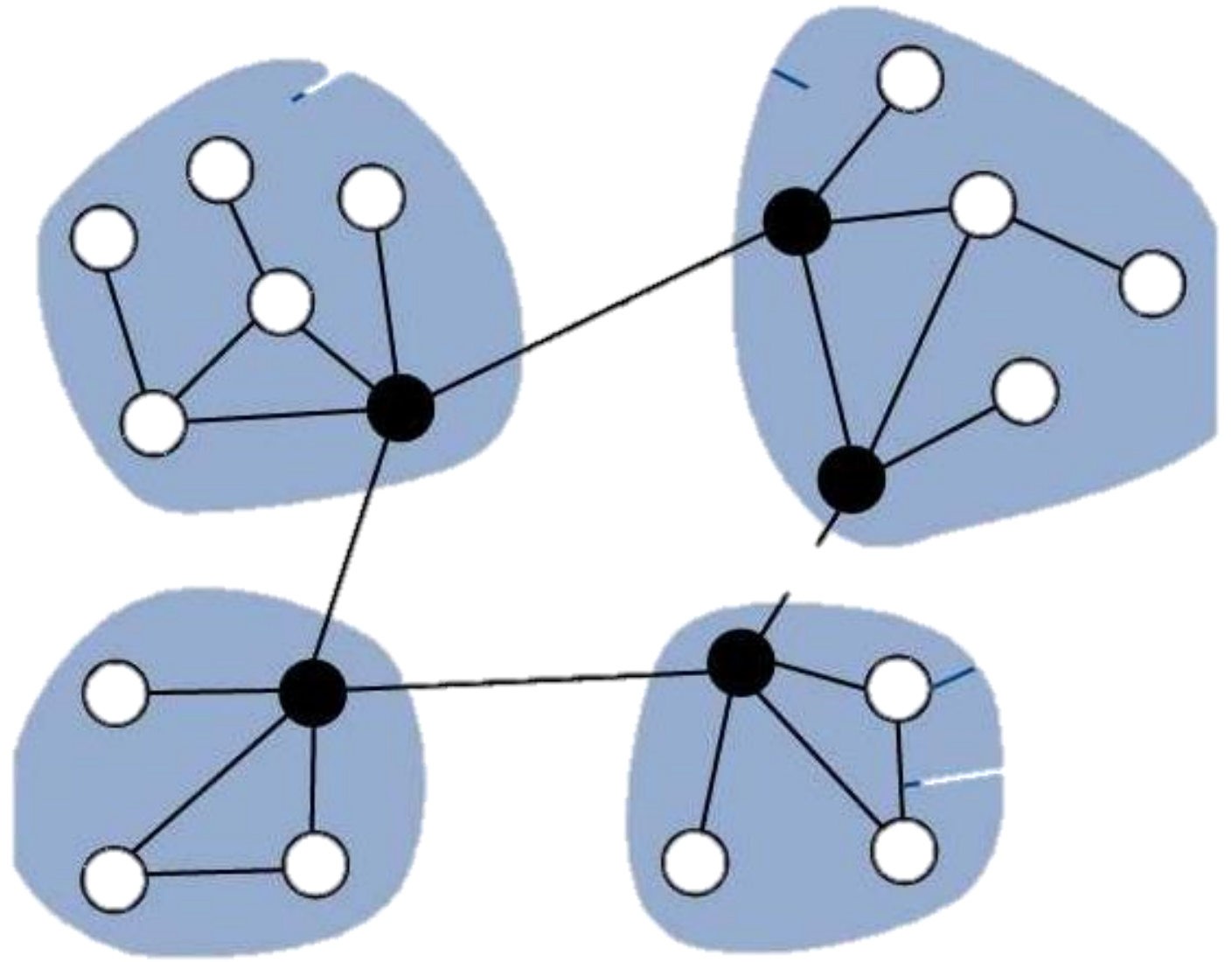
# Common Measures

Degree

Path Length

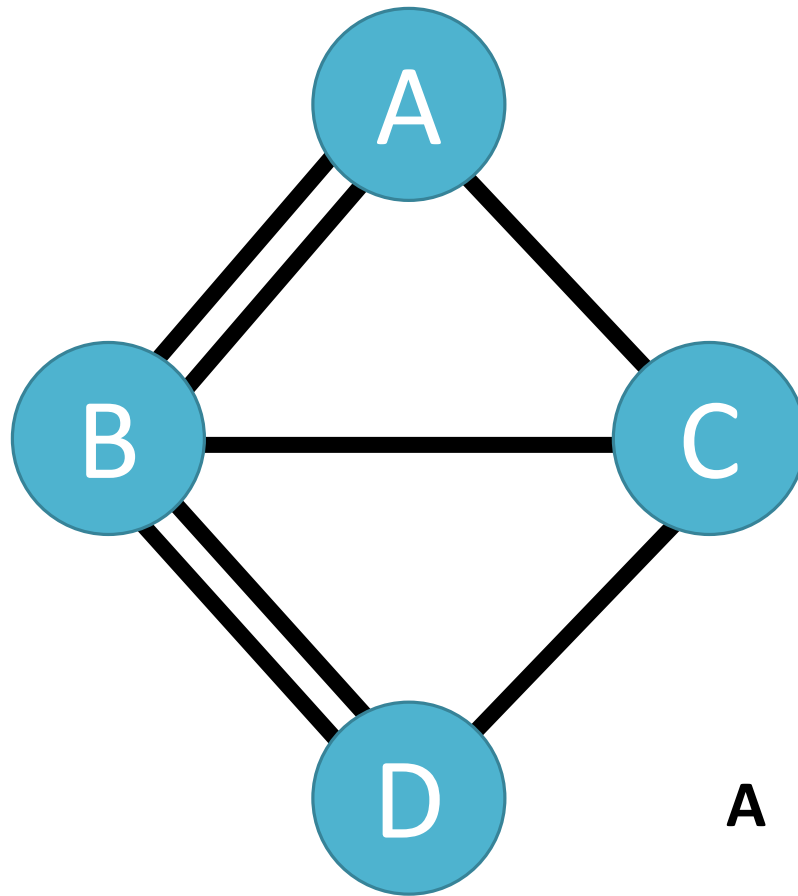
Clustering

Centrality





# Graphs as matrices



	A	B	C	D
A	-	2	1	0
B	2	-	1	2
C	1	1	-	1
D	0	2	1	-



# Popular Pythonistas

1. What is a network?
2. Building graphs
3. Who's who in PyNash?

Guiding Question:  
Who are PyNash's  
“glue guys/gals”?

---

# The MeetUp Data

---

Used MeetUp API to get the following information:

1. All groups within 25 miles of Nashville.
2. All members of those groups.
3. All events for each group and who RSVPed yes.

(Everything is on Github)





# PyNash Relationships

---

We have >1400 members. But not all of them come to every meeting. (And many don't come to any meetings.)

So we want to build a graph where...

1. each node is EITHER a person or event
2. Each edge represents attendance at that event

## Building a graph: Event data

	name	time	yes_rsvp_count
event_id			
<b>244222379</b>	An October Two-fer: Refactoring, Extra Code In...	2017-10- 26 23:00:00	40
<b>cldxflywnbgb</b>	PyNash Lunch!	2017-10- 04 16:00:00	19
<b>243161760</b>	Creating Better Beer Through Data Science	2017-09- 28 23:00:00	69
<b>cldxflywmbjb</b>	PyNash Lunch!	2017-09- 06 16:00:00	15
<b>242470062</b>	You and I and the PyNash API	2017-08- 24 23:00:00	59

## Building a graph: Member data

	name	hometown	city	state
member_id				
6144606	Aaric Pittman	Smyrna	Smyrna	TN
65072762	Aaron	Saratoga	Brentwood	TN
225546676	Aaron Barfoot	NaN	Nashville	TN
41795222	Aaron Carlton	NaN	Murfreesboro	TN
190499461	Aaron Hall	NaN	Nolensville	TN
205279429	Aaron K	NaN	Nashville	TN
66862502	Aaron Price	NaN	Nashville	TN
7768516	Aaron Webb	Urbandale, IA	Springfield	TN
5739275	Aaron West	Clarksville, TN	Mount Juliet	TN
198023758	Abby Fleming	NaN	Lewisburg	TN

## Building a graph: RSVP data

	<b>event_id</b>	<b>member_id</b>
<b>74389</b>	cldxflywnbgb	237869773
<b>74390</b>	cldxflywnbgb	53439252
<b>74391</b>	cldxflywnbgb	127214502
<b>74392</b>	cldxflywnbgb	156626172
<b>74393</b>	cldxflywnbgb	234179814



# NetworkX

## Software for complex networks

Stable (notes)

2.0 – September 2017

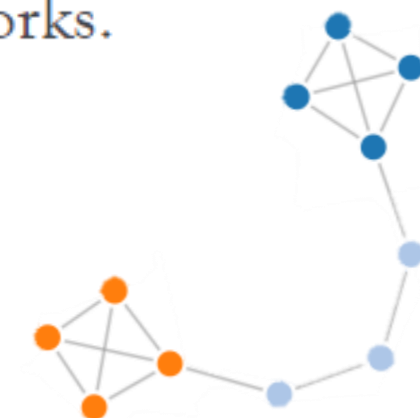
[download](#) | [doc](#) | [pdf](#)

Latest (notes)

2.1 development

[github](#) | [doc](#) | [pdf](#)

NetworkX is a Python package for the creation, manipulation, and study of the structure, dynamics, and functions of complex networks.



# NetworkX Setup

---

## NETWORKX FUNCTIONS

```
nx.Graph()
```

```
nx.DiGraph()
```

```
nx.degree(G)
```

```
nx.clustering(G)
```

```
nx.draw_networkx(G)
```

## GRAPH METHODS / SLICING

```
G.nodes()
```

```
G.edges()
```

```
G.degree()
```

```
G[node]
```

```
G[node][node]
```

## Building a graph: Declarative Syntax

```
# Create an (undirected) Graph instance.  
g = nx.Graph()  
  
# Add nodes - a few members and the event itself.  
g.add_node(237869773)  
g.add_node(53439252)  
g.add_node(127214502)  
g.add_node('cldxflywnbgb')  
  
# Add edges between members and the event.  
g.add_edge(127214502, 'cldxflywnbgb')  
g.add_edge(53439252, 'cldxflywnbgb')
```

```
g
```

```
<networkx.classes.graph.Graph at 0x1a715fbf518>
```

## Slicing the graph

Graphs are setup as  
dicts-of-dicts-of-dicts.

Source: Target: Attrs

```
g.nodes()
```

```
NodeView((237869773, 53439252, 127214502, 'cldxflywnbgb'))
```

```
g.edges()
```

```
EdgeView([(53439252, 'cldxflywnbgb'), (127214502, 'cldxflywnbgb')])
```

```
g[127214502]
```

```
AtlasView({'cldxflywnbgb': {}})
```

```
g[127214502]['cldxflywnbgb']
```

```
{}
```

# Building a Graph: From Pandas DF

```
g = nx.from_pandas_dataframe(lunch_rsvps,
                             source = 'member_id',
                             target='event_id',
                             edge_attr=['group_urlname'])
```

```
pprint(dict(g.edges()))
```

```
{('cldxflywnbgb', 2069): {'group_urlname': 'PyNash'},
 ('cldxflywnbgb', 2896514): {'group_urlname': 'PyNash'},
 ('cldxflywnbgb', 30123762): {'group_urlname': 'PyNash'},
 ('cldxflywnbgb', 53439252): {'group_urlname': 'PyNash'},
 ('cldxflywnbgb', 111917572): {'group_urlname': 'PyNash'},
 ('cldxflywnbgb', 127214502): {'group_urlname': 'PyNash'},
 (237869773, 'cldxflywnbgb'): {'group_urlname': 'PyNash'},
 ('cldxflywnbgb', 43237102): {'group_urlname': 'PyNash'},
 ('cldxflywnbgb', 156626172): {'group_urlname': 'PyNash'},
 ('cldxflywnbgb', 182591120): {'group_urlname': 'PyNash'},
 ('cldxflywnbgb', 183564319): {'group_urlname': 'PyNash'},
 ('cldxflywnbgb', 184389351): {'group_urlname': 'PyNash'},
 ('cldxflywnbgb', 191186119): {'group_urlname': 'PyNash'},
 ('cldxflywnbgb', 204129504): {'group_urlname': 'PyNash'},
 ('cldxflywnbgb', 230753190): {'group_urlname': 'PyNash'},
 ('cldxflywnbgb', 232676405): {'group_urlname': 'PyNash'}}
```

The lunch graph

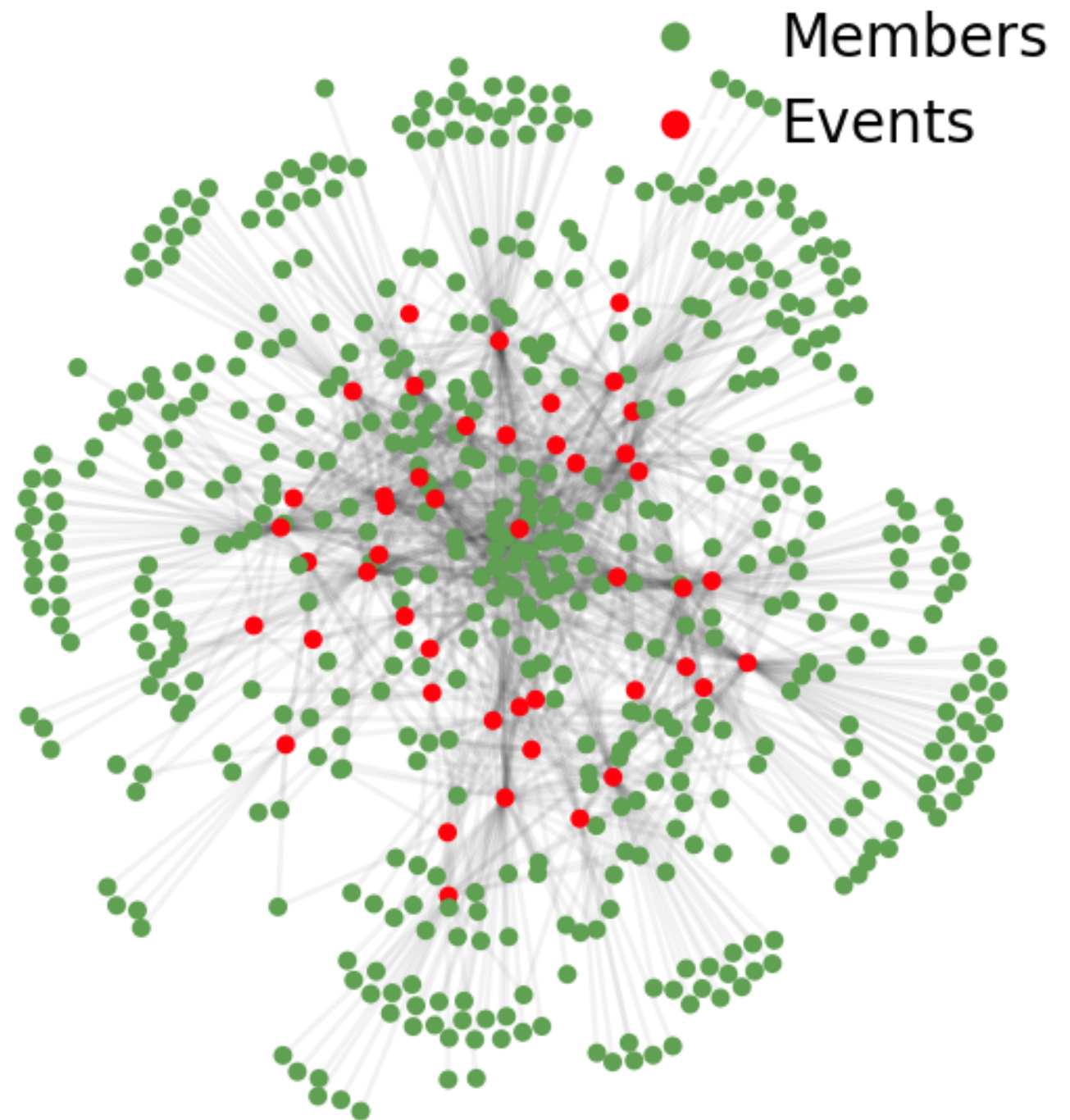
Degree of the  
event = 19  
(RSVPs)

Degree of the  
members = 1  
(event attended)





Scaling it up  
to all events





# Popular Pythonistas

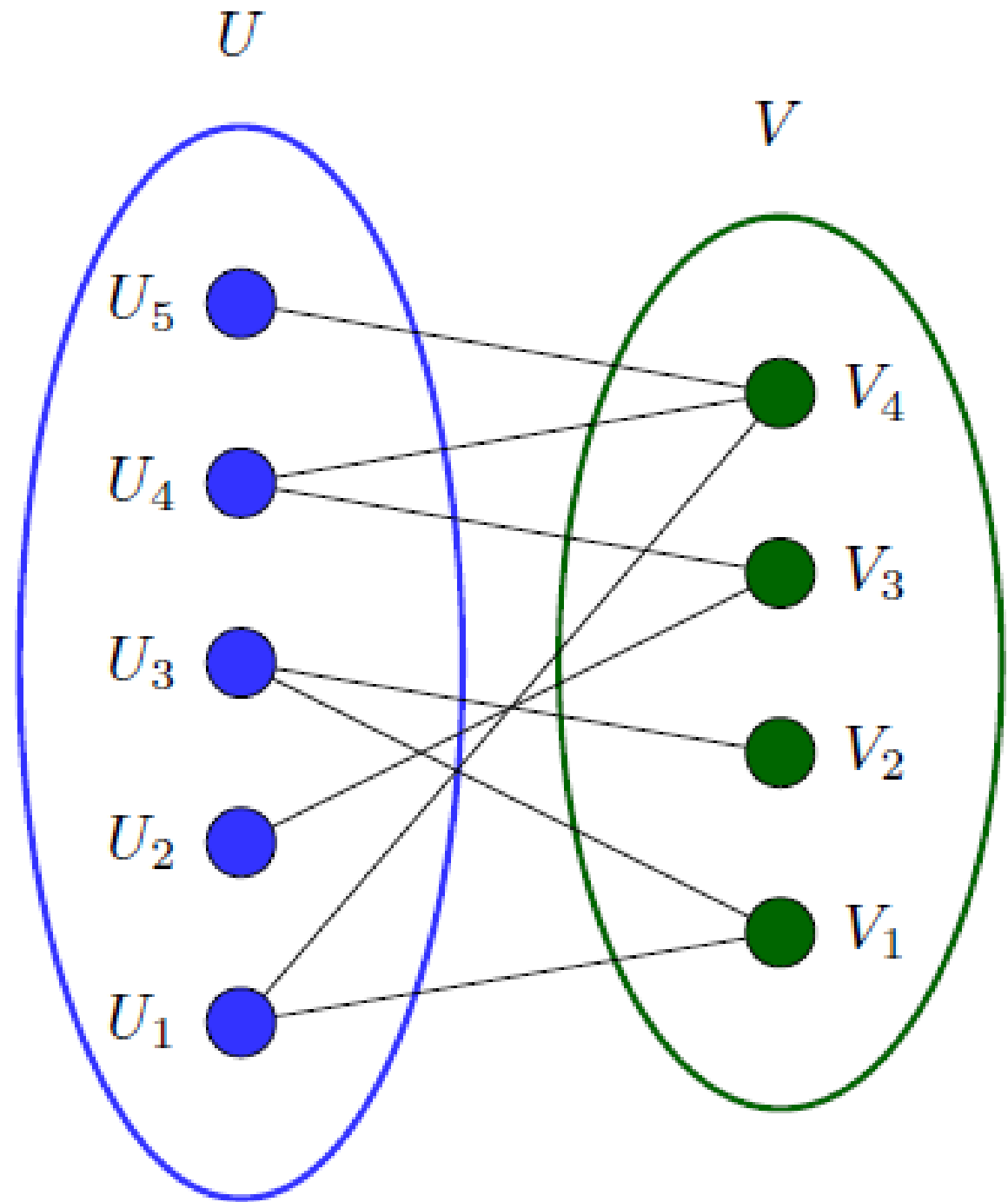
1. What is a network?
2. Building graphs
3. Who's who in PyNash?

# Bipartite Graphs

Great initial starting place for some datasets.

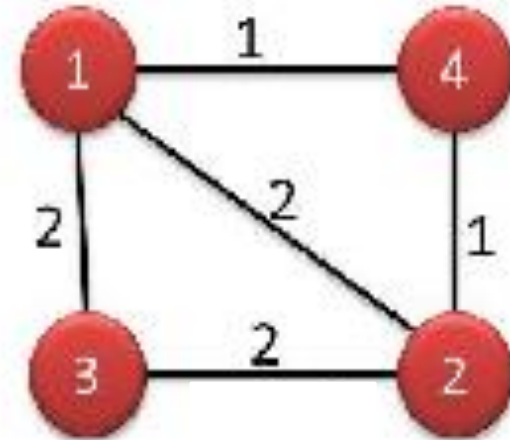
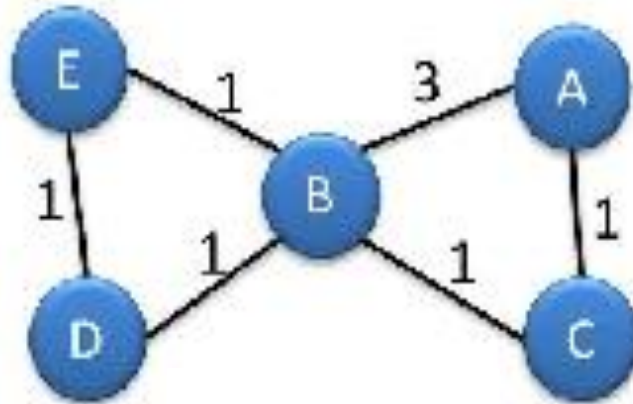
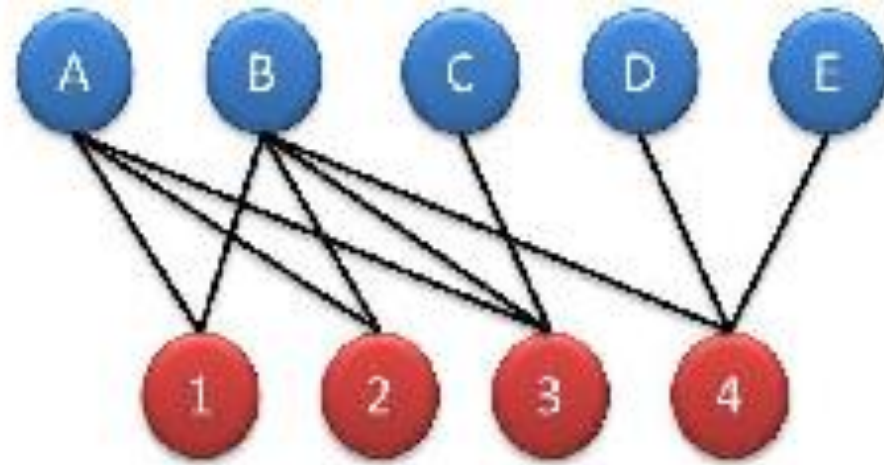
Can take the form of  
(Person node  $\rightarrow$   
Affiliation node)

e.g. (Sports fan  $\rightarrow$   
Sports team)



# Projecting bipartiate graphs

(Person, Event) to  
(Person, Person)



# Projecting the graph

---

```
gm = nx.bipartite.weighted_projected_graph(g, member_nodes, ratio=False)  
ge = nx.bipartite.weighted_projected_graph(g, event_nodes, ratio=False)
```

```
print('There are {} members in the Member graph.'.format(len(gm.nodes)))  
print('There are {} event in the Events graph.'.format(len(ge.nodes)))
```

There are 526 members in the Member graph.  
There are 46 event in the Events graph.

# The “members” graph

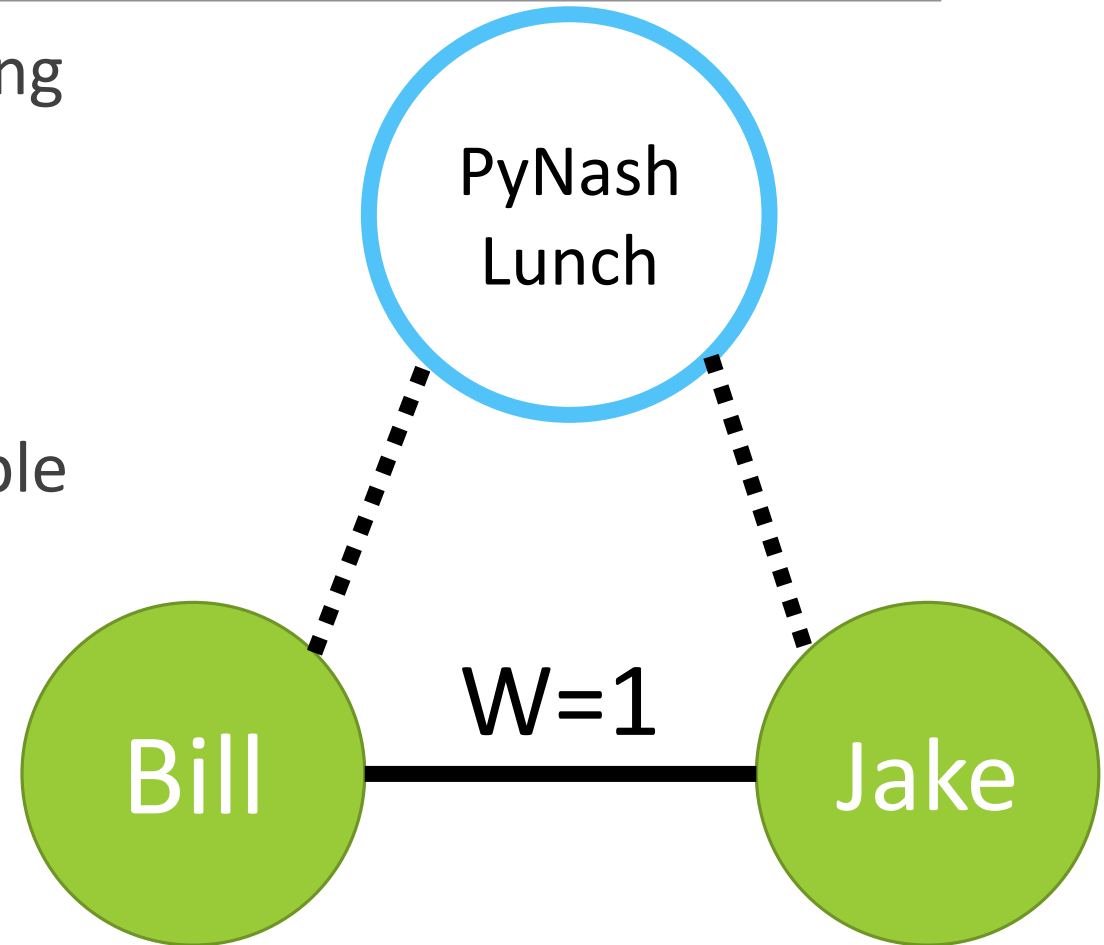
---

Now we have a graph with the following attributes:

**Node:** a person who attended at least one event in the past two years

**Edge:** shared event between two people

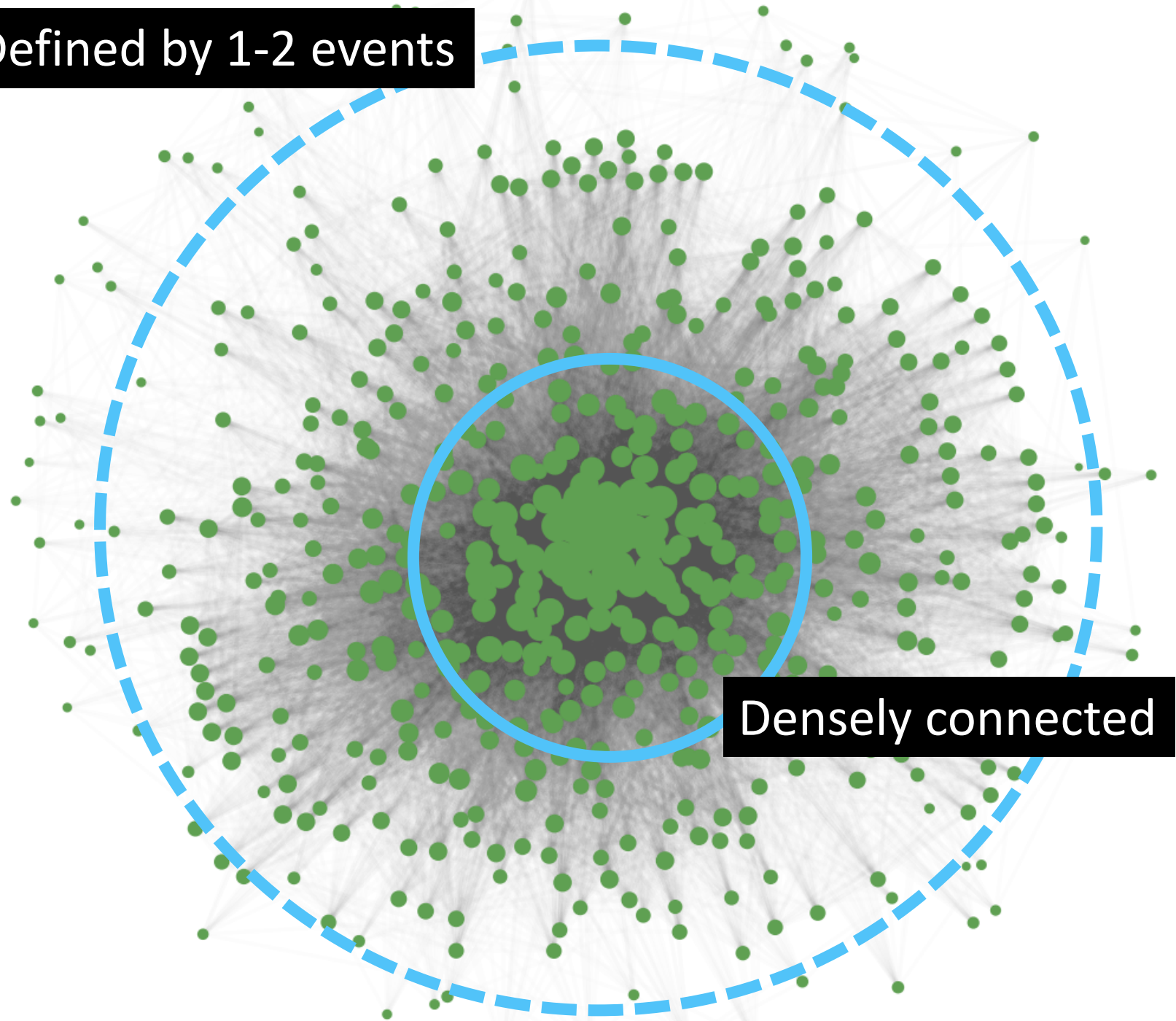
**Edge weight:** the number of shared events



# Member Structure

Large “core” of highly connected individuals, with many others on the periphery.

Defined by 1-2 events



Densely connected

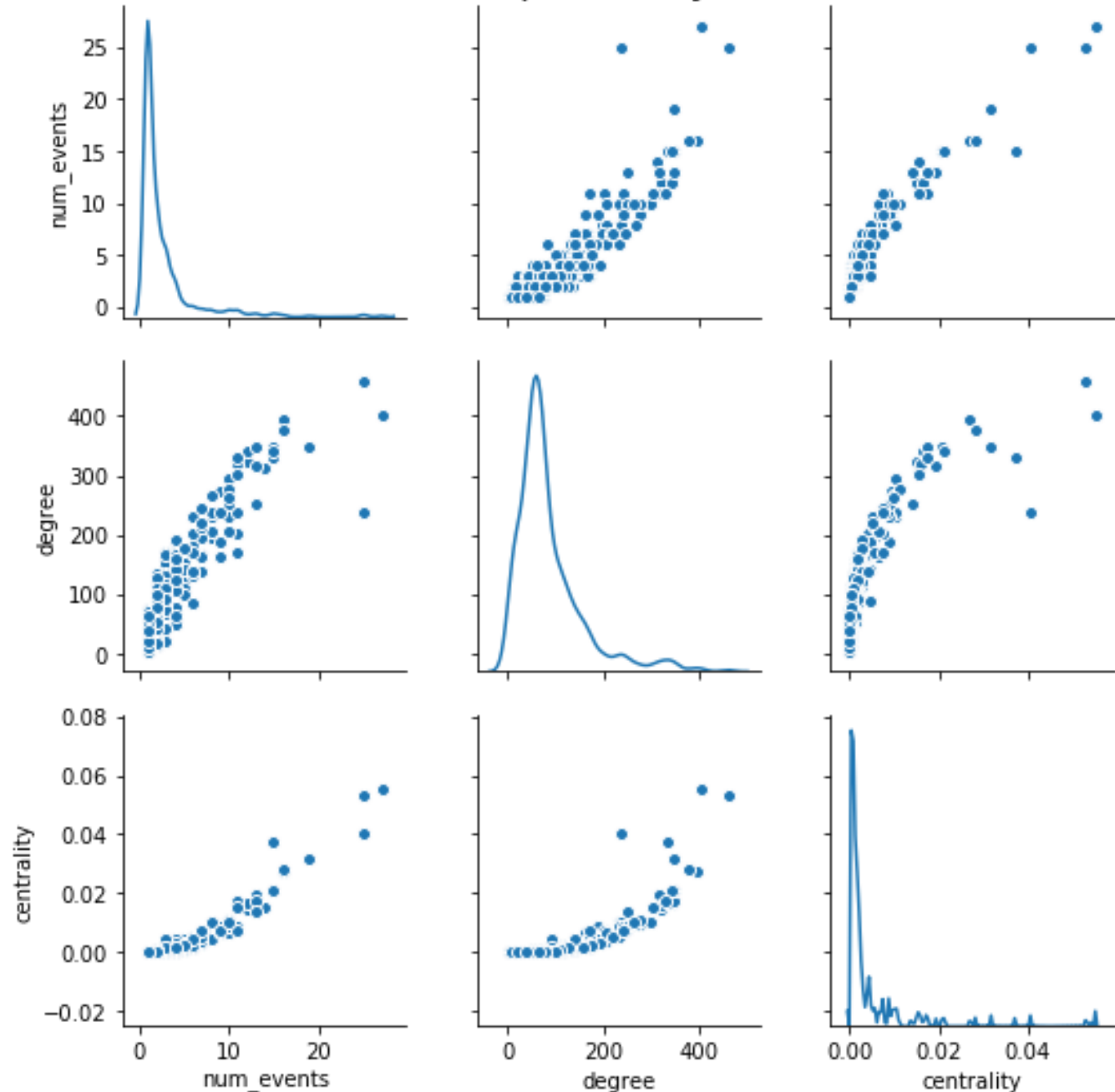


# Getting network measures

Degree = number of members w/ shared attendance at an event

Centrality = how much a person “connects” two other people

Distributions and Correlations  
Between Graph Theory Measures



*\*drumroll\**

---

# Our top “connectors”

	<b>name</b>	<b>num_events</b>	<b>degree</b>	<b>clustering</b>	<b>centrality</b>
<b>121334792</b>	Greg Back	27	403	0.224732	0.055124
<b>57907252</b>	Chad Upjohn	25	459	0.203471	0.053080
<b>30123762</b>	Jason Myers	25	237	0.237753	0.040602
<b>2896514</b>	Trey Brooks	15	331	0.245372	0.037175
<b>184547023</b>	Chris Jarvis	19	347	0.264380	0.031768
<b>12140530</b>	Bill Israel	16	378	0.249323	0.028138
<b>202882025</b>	Michael mead	16	396	0.246465	0.027081
<b>13606604</b>	Alex Simonian	15	343	0.263362	0.021315
<b>126309962</b>	Aliva Gifford	15	348	0.269618	0.020821

# Event Attendance for Upjohn and Myers

---

## CHAD UPJOHN

DEGREE = 459

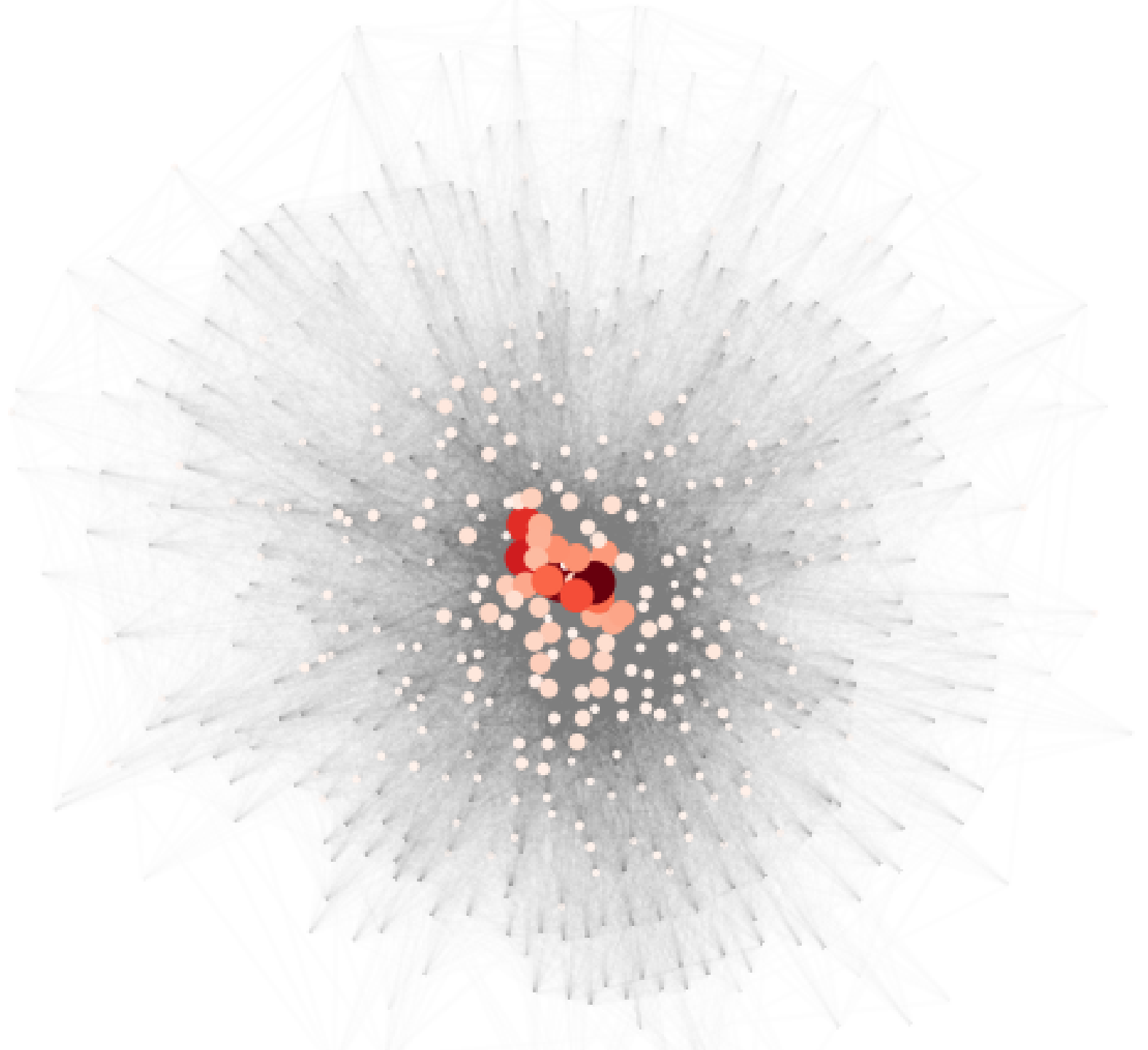
1. An October Two-fer: Refactoring, Extra Code Included / PDFs Against Humanity
2. Creating Better Beer Through Data Science
3. You and I and the PyNash API
4. PyNash x Penny U: An Evening of Learning
5. Interactive Python Environments: IPython, Jupyter, and Beaker, Oh My!
6. PyNash Lunch!
7. Logging beyond /dev/null -- ELK Stack for Log Visualization and Analysis
8. PyNash Lunch!

## JASON MYERS

DEGREE = 237

1. PyNash Lunch!
2. PyNash Lunch!
3. You and I and the PyNash API
4. PyNash Lunch!
5. PyNash Lunch!
6. PyNash Lunch!
7. PyNash Lunch!
8. PyNash Lunch!

PyNash  
Members:  
Emphasizing  
centrality



# Now, to Nashville!

---



# Making a splash

We now know who to get in touch within PyNash – but what about outside of PyNash?

There are likely going to be **clusters** of groups

- Tech groups
- Outdoors groups
- Professional groups



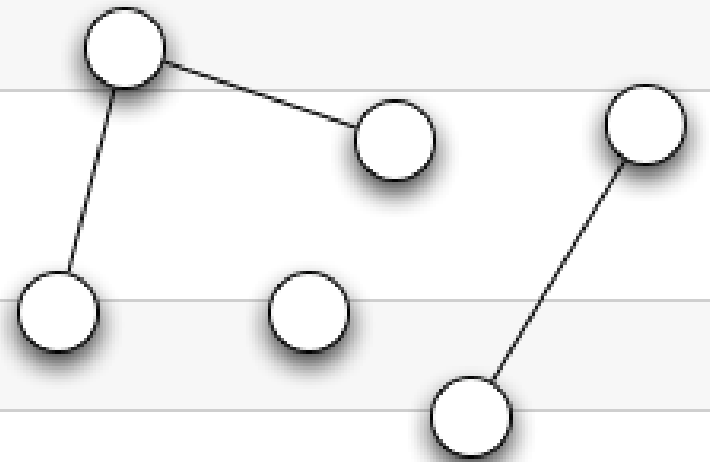
```
# Build DataFrame from pynash_rsvps
g = nx.from_pandas_dataframe(rsvps,
                             source='member_id',
                             target='event_id',
                             edge_attr='group_urlname')
```

# A new problem!!

```
nx.is_connected(g)
```

False

```
# Get the connected subcomponents of the main graph
components = list(nx.connected_components(g))
connected_nodes = sorted(components, key=len)[-1]
g = g.subgraph(connected_nodes)
```



# Creating a metric DataFrame

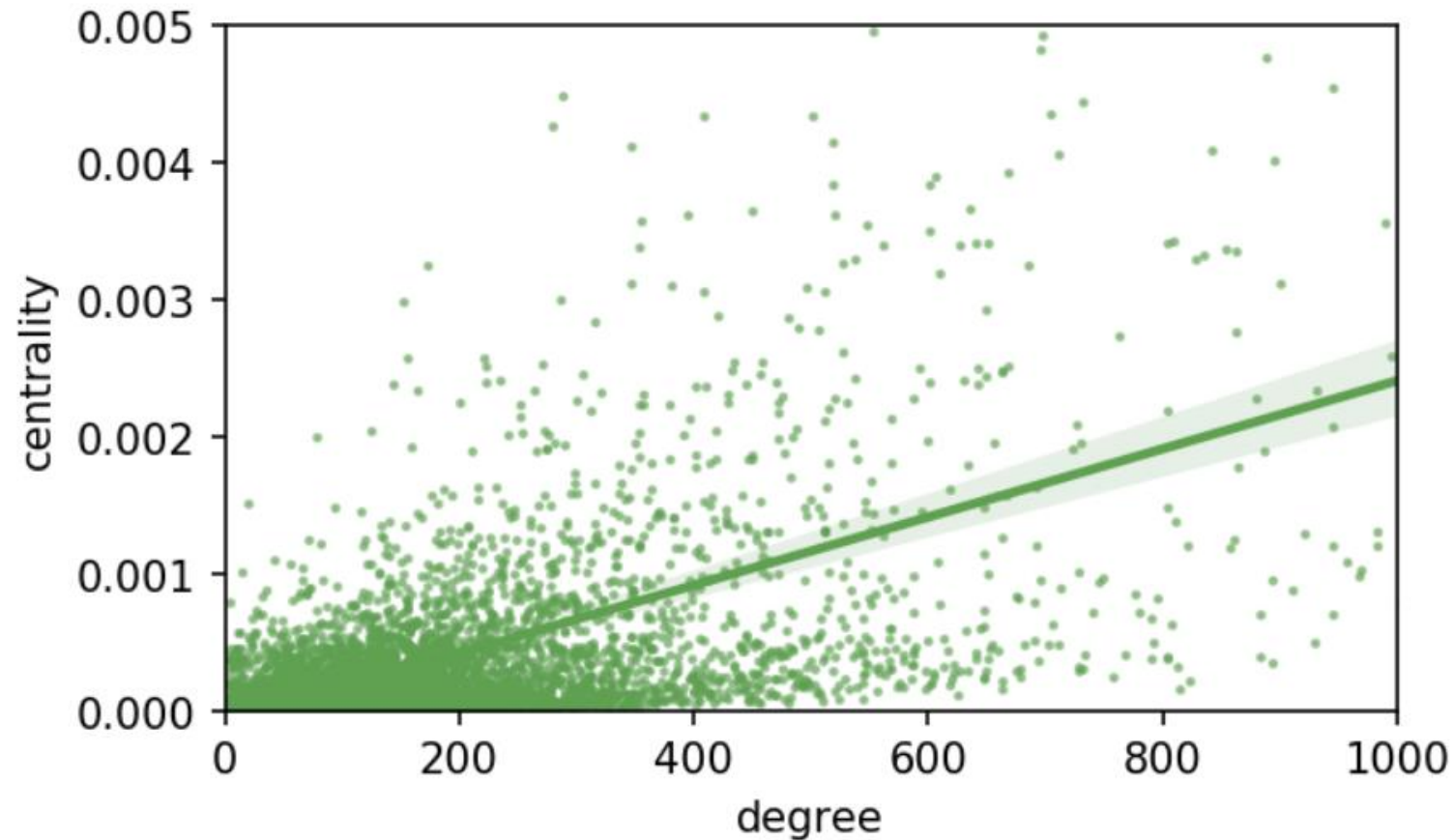
---

```
df_members = pd.DataFrame(index=member_nodes)
df_members['in_pynash'] = [True if m in pynash_member_ids
                           else False for m in member_nodes]
df_members['num_events'] = pd.Series(dict(nx.degree(g, member_nodes)))
df_members['degree'] = pd.Series(dict(nx.degree(gm)))
df_members['clustering'] = pd.Series(nx.clustering(gm))
df_members['centrality'] = pd.Series(nx.betweenness_centrality(gm, k=3000,
                                                                normalized=True,
                                                                weight='weight'))
```

Takes a LONG time – can be parallelized.

In larger networks, there is a weaker relationship between degree and centrality

---

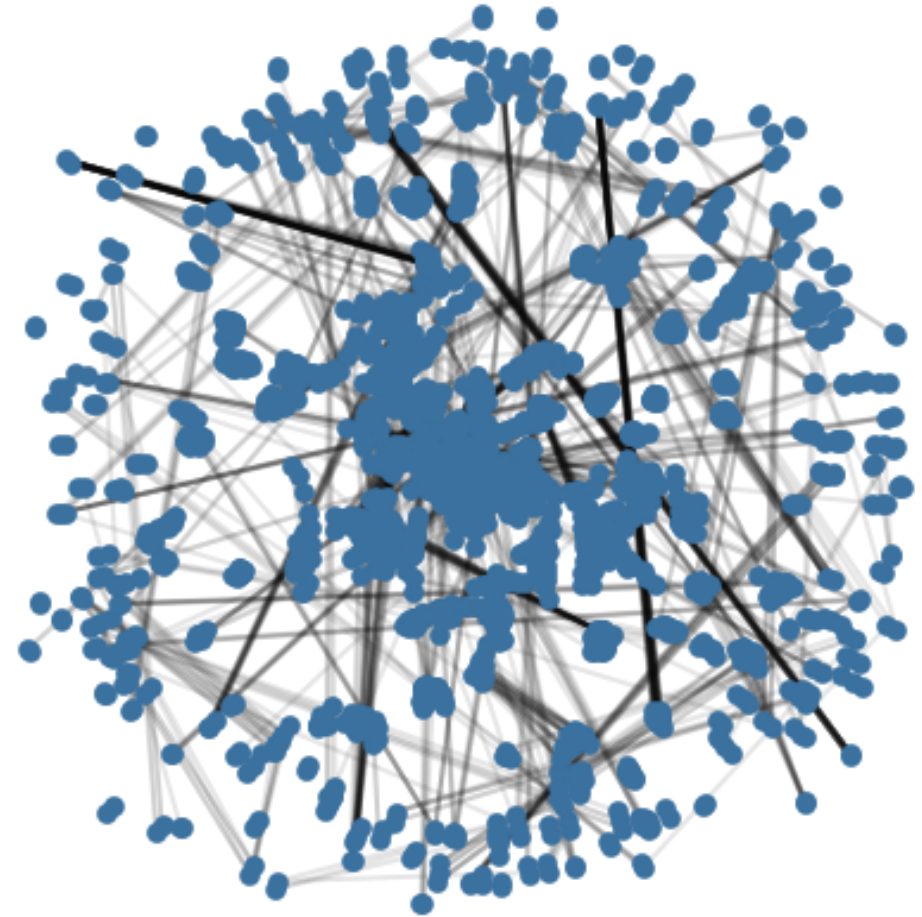


# How about the groups?

Event projection – edges represent shared members.

- **High degree** for an event means it has (at least one) shared attendees with a lot of other events.
- **Low clustering** means that it does not draw from the same pool of shared attendees each time.

Aggregated events by group and summarized!





## LOWEST CLUSTERING (MOST “NEW PEOPLE”)

---

MusicCityDrinkingBuddies  
Movie-Lovers  
Nashville-Psychotherapy-Institute  
SavvyCoders  
Tennessee-Real-Estate-Investors  
Free & Cheap Events  
NashvilleMusician  
funhappeningsocialgroup  
paddleadventuresunlimited

## HIGHEST DEGREE (MOST SHARED ATTENDANCE)

nashjs  
MTN-40  
Nashville-NET-User-Group  
Sunday-Assembly-Nashville  
dnd-49  
nashville-ux  
Data-Science-Nashville  
NashvilleFlagFootball  
steppingoutsocialdance



# The big picture

---

## APPLICATIONS

Recommendation engine – based on your interests, who might you connect with?

Social status measures beyond “number of events” attended.

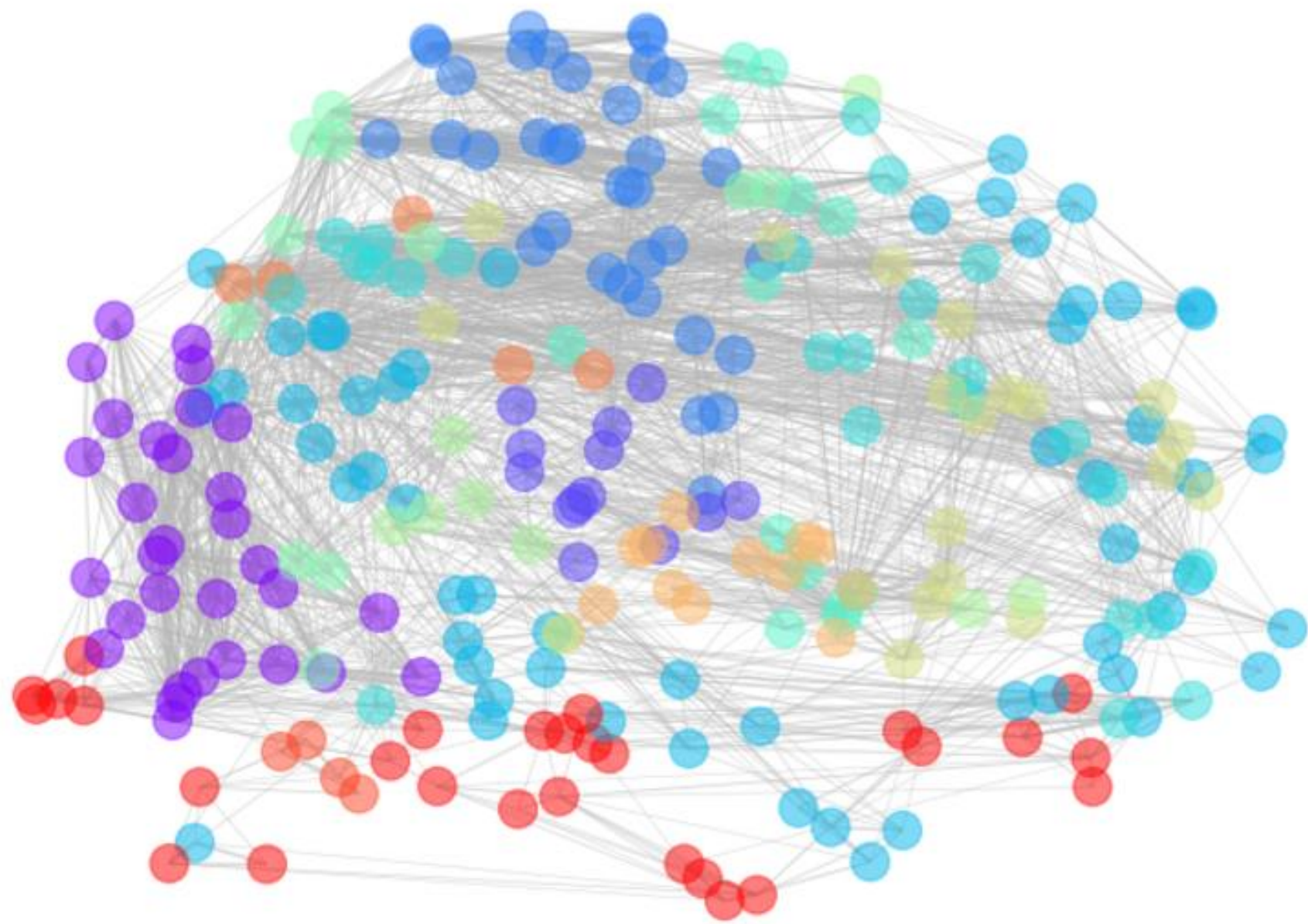
Target “influential” people for advertising / recruiting.

## LIMITATIONS

This data is based on “rsvps” – may not reflect real attendance.

Attendance  $\neq$  participation

Wide number of connections does not represent influence – though it may represent persistence.



# Recap of NetworkX

---

## BUILD GRAPHS

```
g = nx.Graph()
g = nx.DiGraph()

nx.from_edge_list()
nx.from_pandas_dataframe()
```

## INDEXING GRAPHS

```
n = g[node]
e = g[node][node]
w = g[node][node][attribute]

[n for n in g.nodes]
[e['weight'] for e in
                             g.edges]
```

# Recap of NetworkX

---

## BUILD GRAPHS

```
nx.degree(g)  
nx.betweenness_centrality(g)  
nx.clustering(g)
```

## DRAW GRAPHS

```
pos = nx.spring_layout(g)  
pos = nx.circular_layout(g)  
  
nx.draw_networkx(g, pos)
```