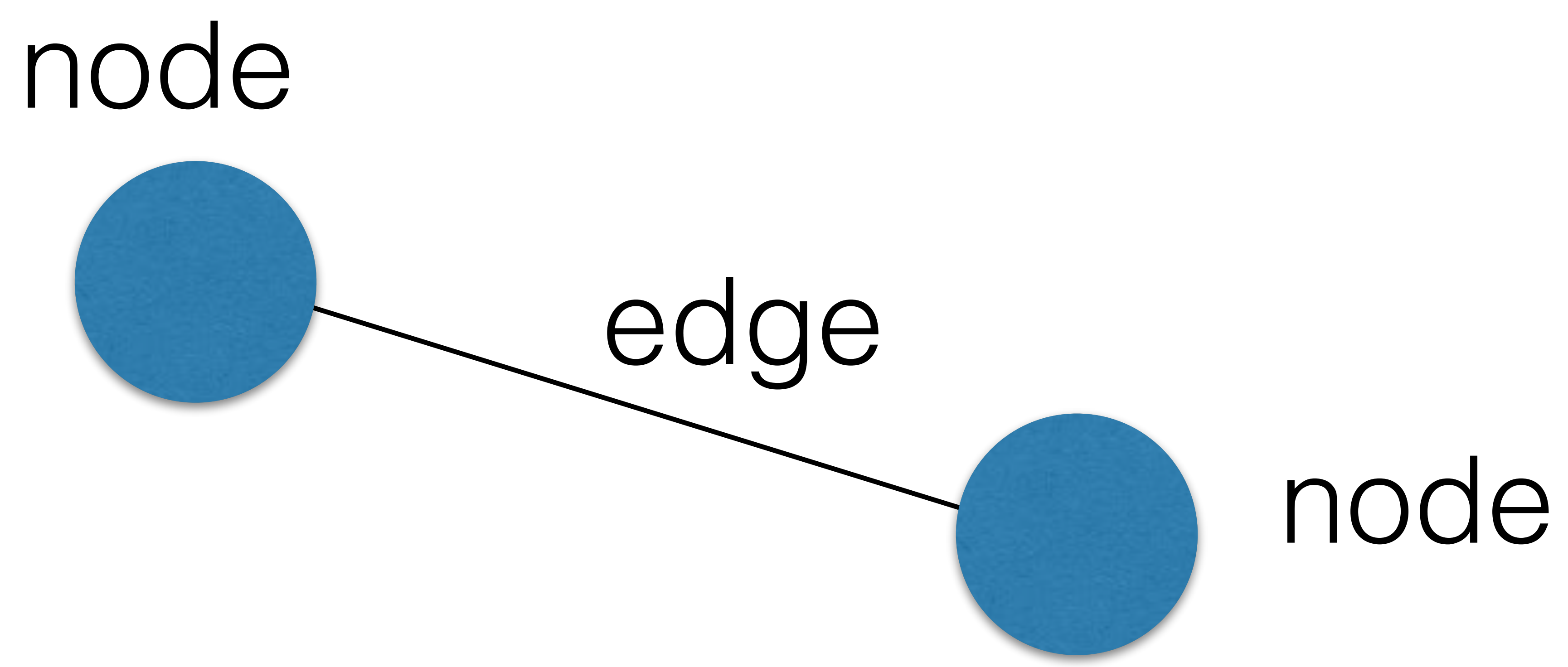
NETWORK ANALYSIS IN PYTHON II

# Definitions & basic recap

# Network/Graph

- Network = Graph = (nodes, edges)
- Directed or Undirected
  - Facebook: Undirected
  - Twitter: Directed
- `networkx`: API for analysis of graphs

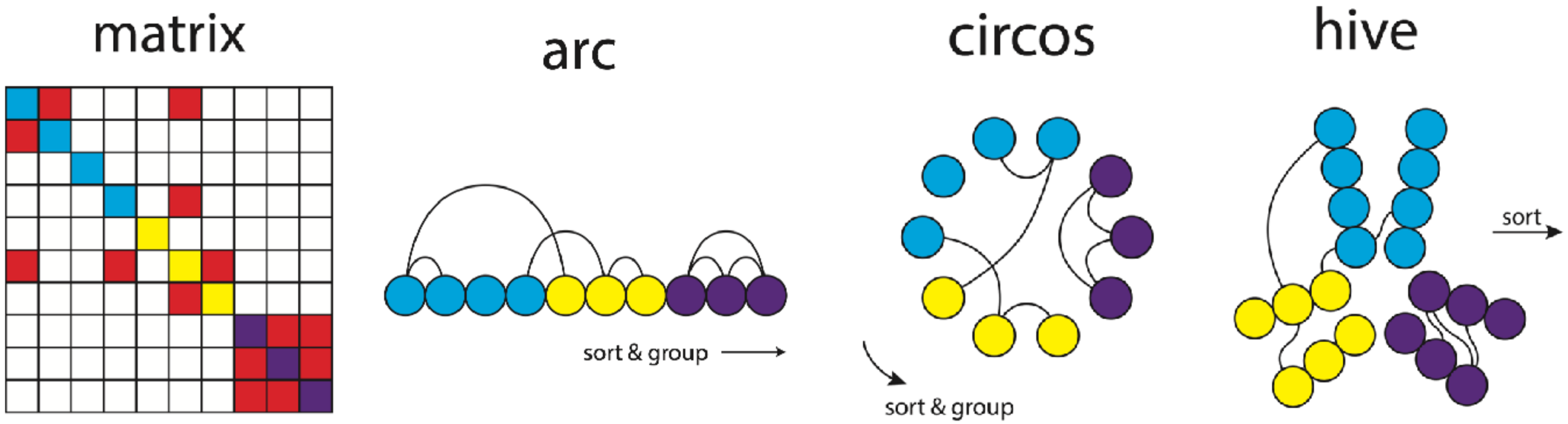


# Basic NetworkX API

```
In [1]: import networkx as nx

In [2]: G
Out[2]: <networkx.classes.graph.Graph at 0x10b192da0>

In [3]: G.nodes()
Out[3]:['customer1', 'customer3', 'customer2']

In [4]: len(G.nodes())
Out[4]: 3

In [5]: len(G.edges())
Out[5]: 2

In [6]: type(G)
Out[6]: networkx.classes.graph.Graph
```

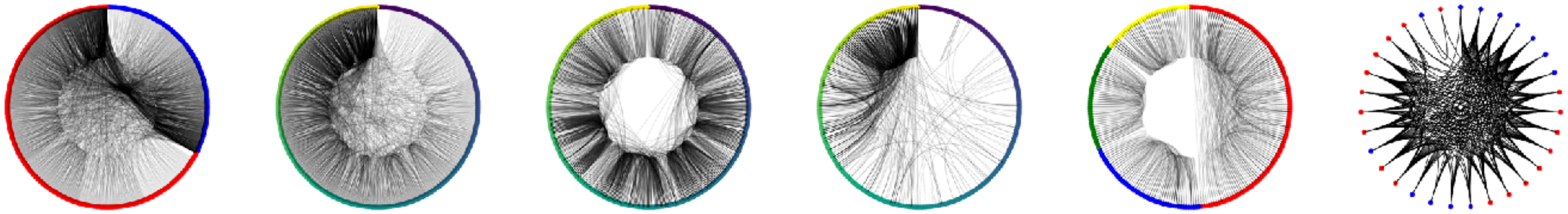
# Network visualization

- `nxviz`: API for creating beautiful and rational graph viz
- Prioritize placement of nodes



# Basic nxviz API

```
In [7]: import nxviz as nv

In [8]: import matplotlib.pyplot as plt

In [9]: c = nv.CircosPlot(G)

In[10]: c.draw()

In[11]: plt.show()
```



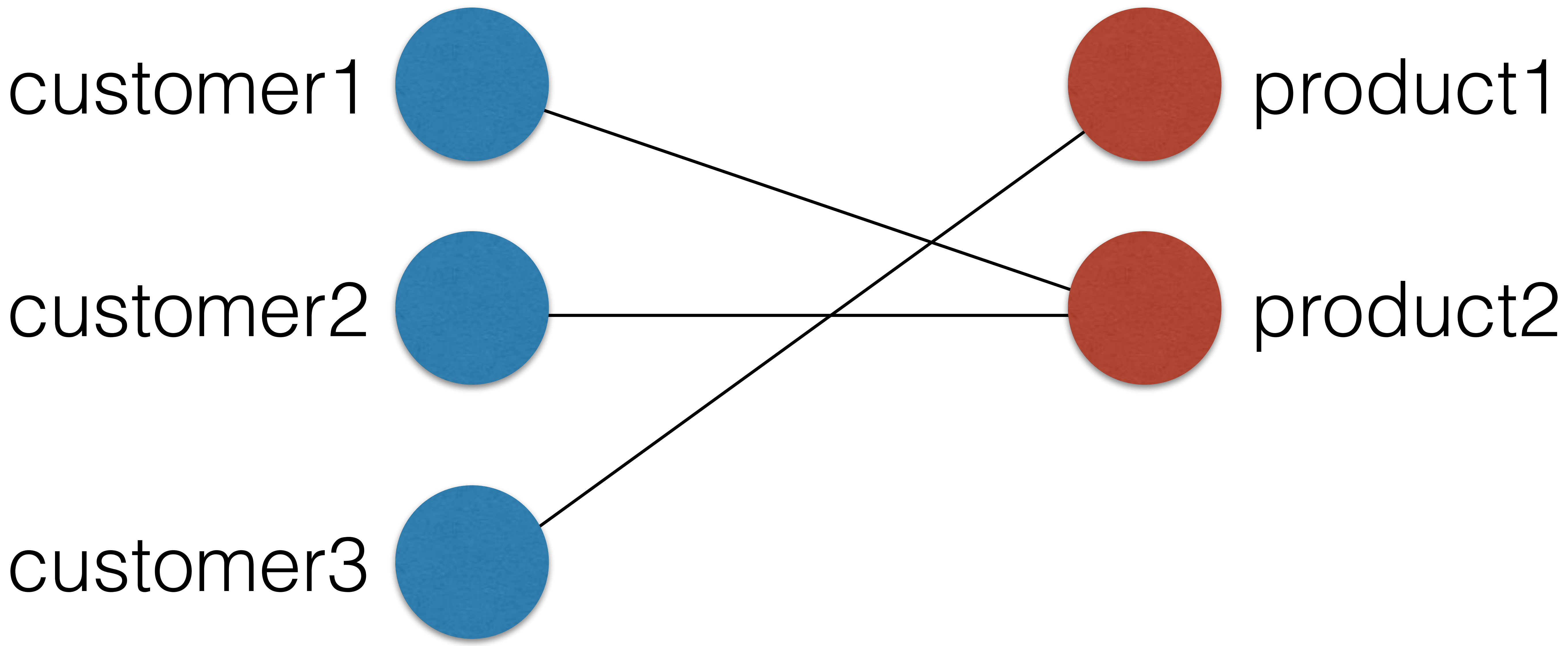
# Let's practice!

# Bipartite graphs

# Bipartite graphs

- A graph that is partitioned into two sets
- Nodes are only connected to nodes in other partitions
- Contrast: “unipartite”

# Bipartite graphs: Example



# Bipartite graphs in NetworkX

```
In [1]: import networkx as nx

In [2]: G = nx.Graph()

In [3]: numbers = range(3)

In [4]: G.add_nodes_from(numbers, bipartite='customers')

In [5]: letters = ['a', 'b']

In [6]: G.add_nodes_from(letters, bipartite='products')
```

# Bipartite graphs in NetworkX

```
In [7]: G.nodes(data=True)
Out[7]:
[(0, {'bipartite': 'customers'}),
 (1, {'bipartite': 'customers'}),
 (2, {'bipartite': 'customers'}),
 ('b', {'bipartite': 'products'}),
 ('a', {'bipartite': 'products'})]
```

# Degree centrality

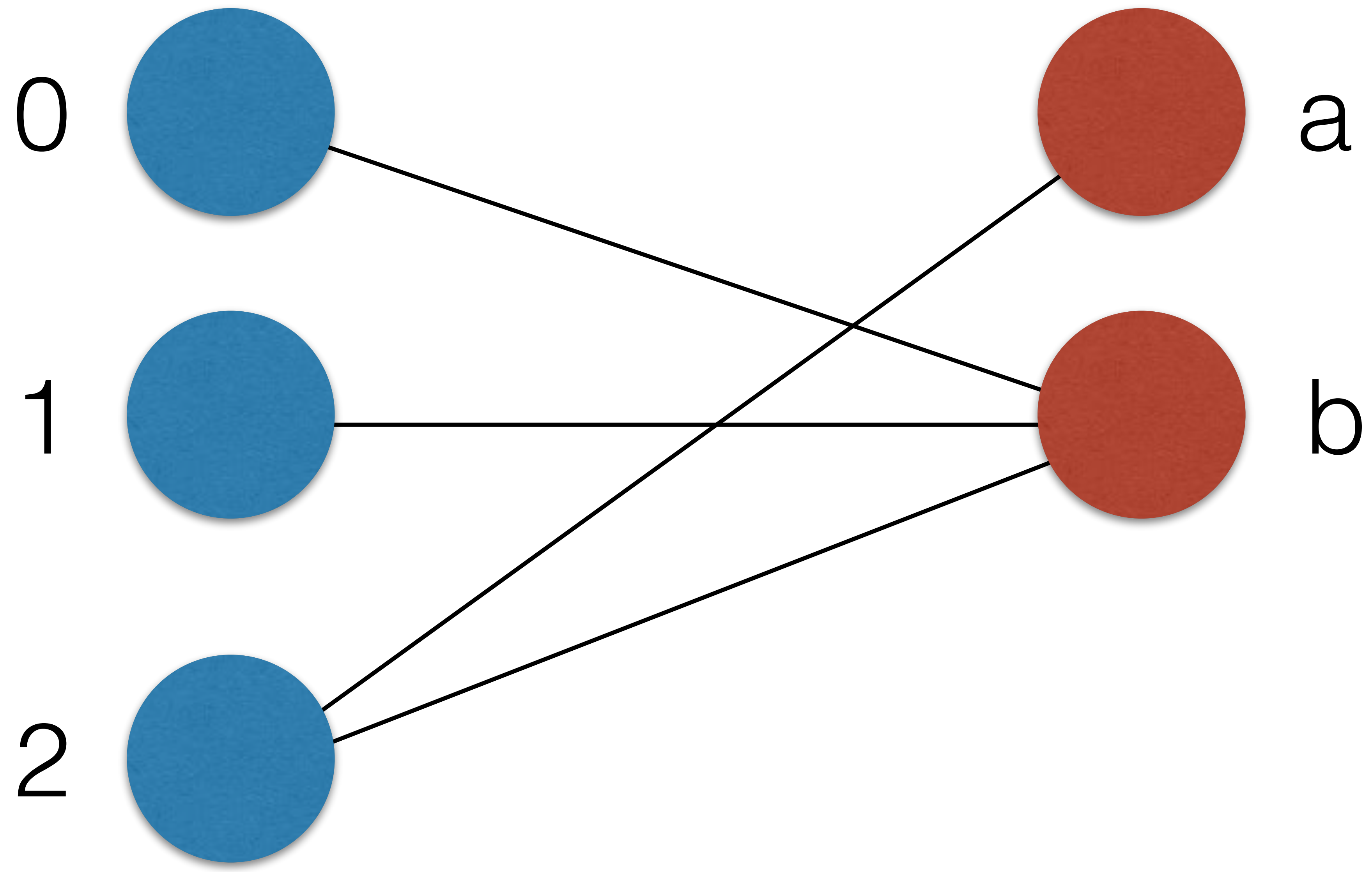
- Definition:

$$\frac{\text{number of neighbors}}{\text{number of possible neighbors}}$$

- Number of possible neighbors depends on graph type

# Bipartite centrality metrics

- Denominator: number of nodes in opposite partition, rather than all other nodes



# Filtering graphs

```
In [1]: cust_nodes = [n for n, d in G.nodes(data=True) if
   ...:                d['bipartite'] == 'customers']

In [2]: cust_nodes
Out[2]:
[(0, {'bipartite': 'customers'}),
 (1, {'bipartite': 'customers'}),
 (2, {'bipartite': 'customers'})]

In [3]: nx.bipartite.degree_centrality(G, cust_nodes)
Out[3]:
{0: 0.5,
 1: 0.5,
 2: 1.0,
 'a': 0.333,
 'b': 1.0}
```

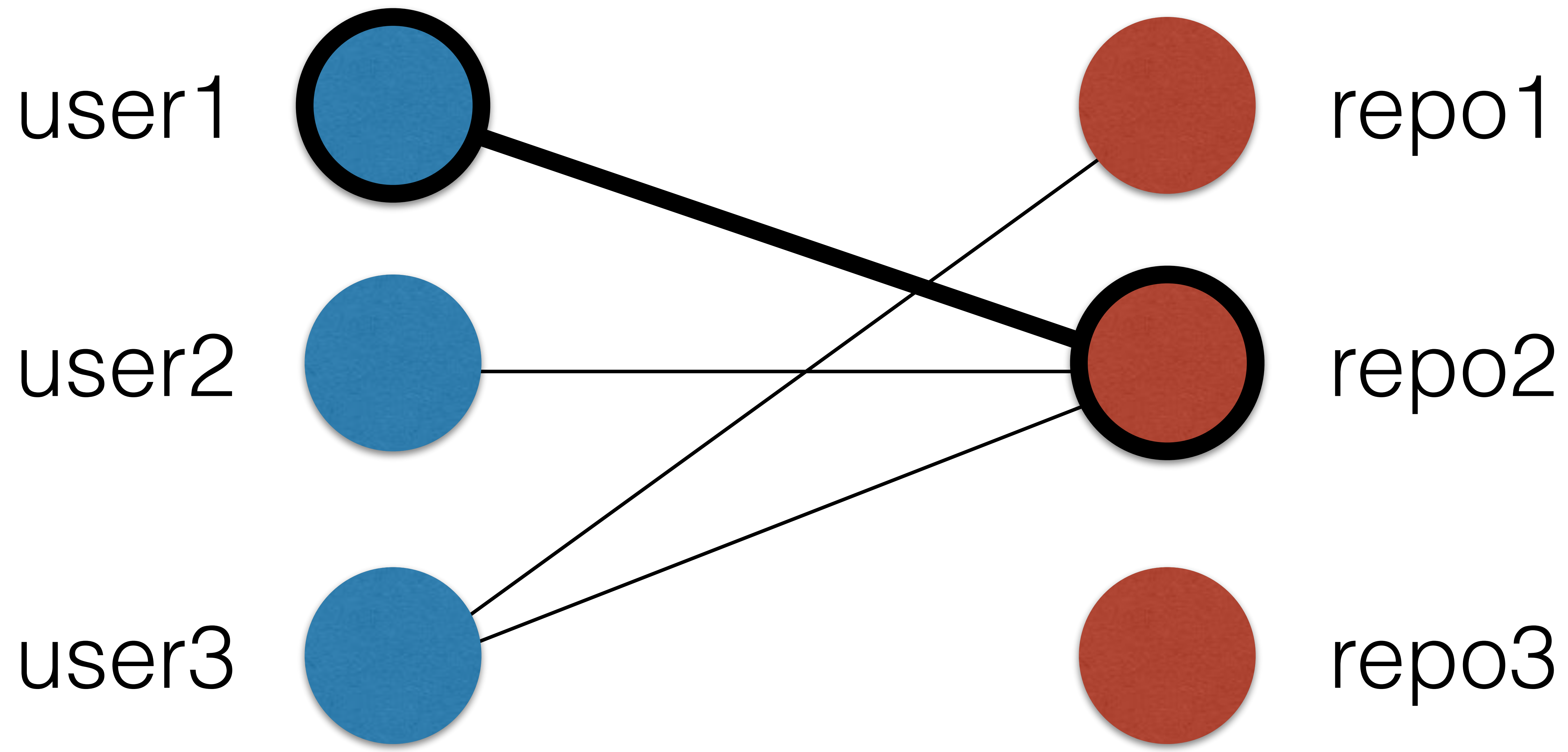
# Let's practice!

# Bipartite graphs and recommendation systems

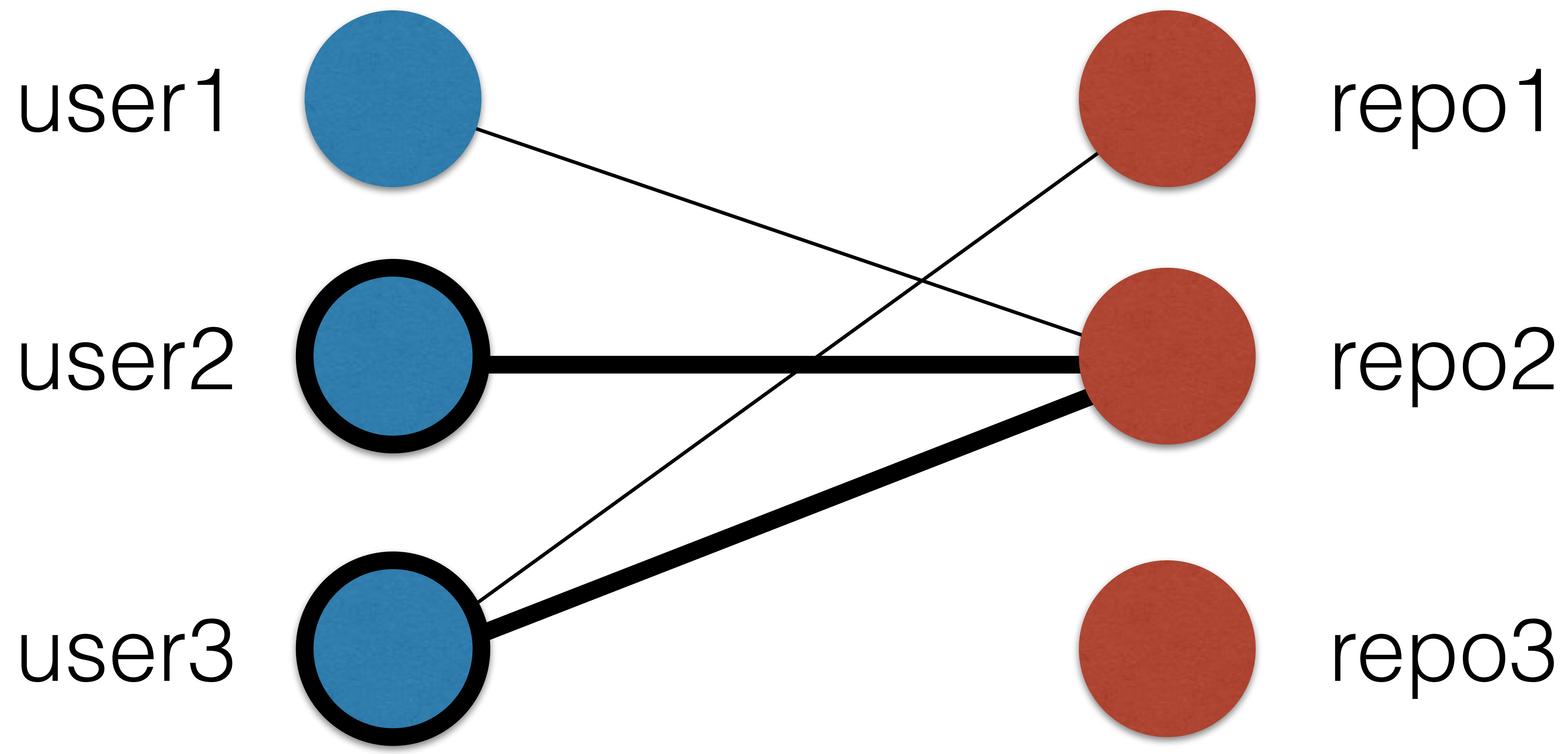
# Recommendation systems

- Previously: Recommended users to connect with one another
- Graph: "unipartite" (or users-only) version
- Now: "bipartite" or (repo-users) version
- Recommending repositories for users to work on

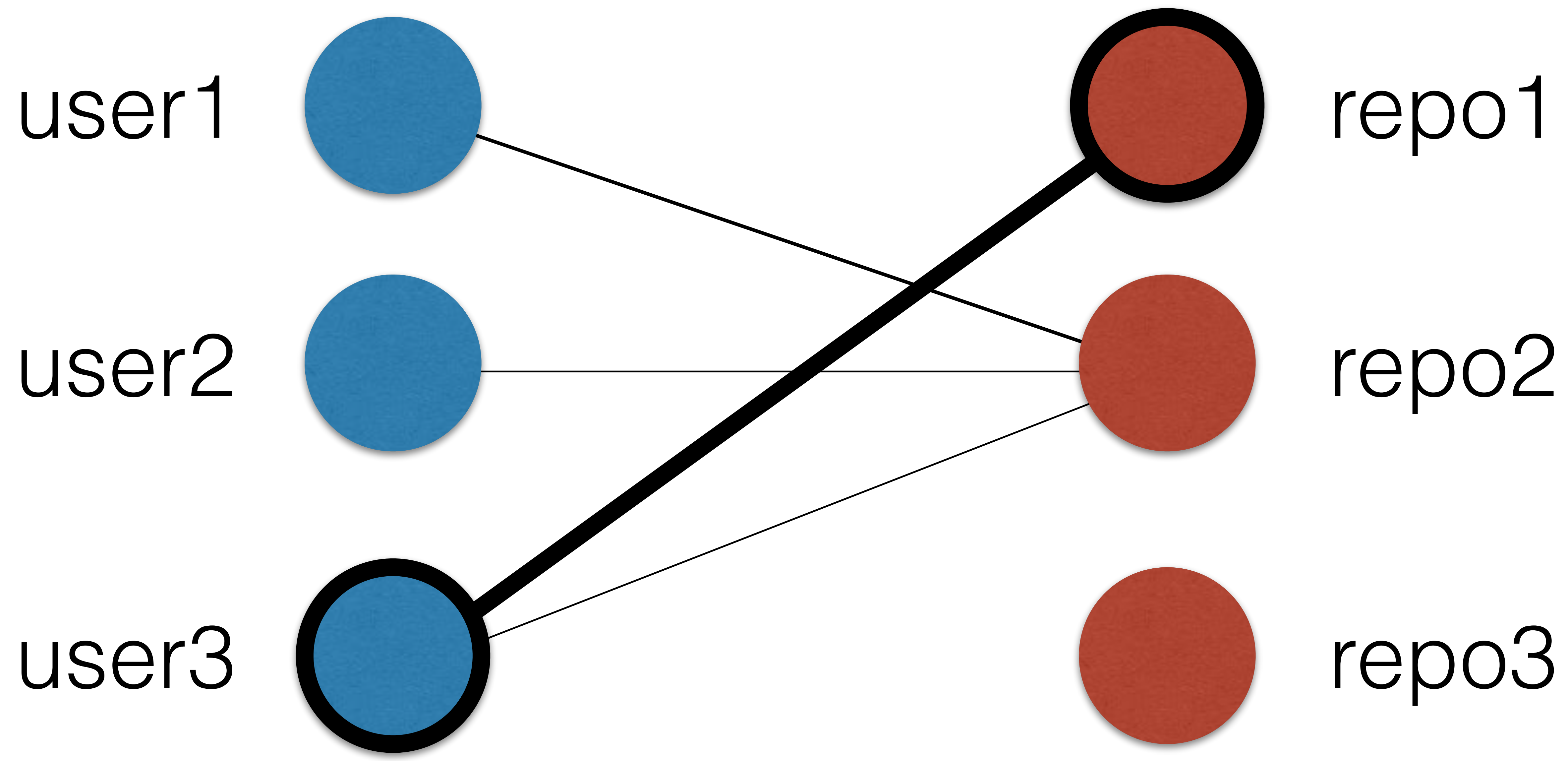
# Recommendation systems



# Recommendation systems



# Recommendation systems



# Code: Node sets

```
In [1]: G.nodes(data=True)
Out[1]:
[('repo3', {'bipartite': 'repositories'}),
 ('repo1', {'bipartite': 'repositories'}),
 ('user1', {'bipartite': 'users'}),
 ('user2', {'bipartite': 'users'}),
 ('repo2', {'bipartite': 'repositories'}),
 ('user3', {'bipartite': 'users'})]

In [2]: G.edges()
Out[2]:
[('repo1', 'user3'),
 ('user1', 'repo2'),
 ('user2', 'repo2'),
 ('repo2', ‘user3')]
```

# Code: Node sets

```
In [3]: user1_nbrs = G.neighbors('user1')

In [4]: user1_nbrs
Out[4]: ['repo2']

In [5]: user3_nbrs = G.neighbors('user3')

In [6]: user3_nbrs
Out[6]: ['repo2', 'repo1']

In [7]: set(user1_nbrs).intersection(user3_nbrs)
Out[7]: {'repo2'}

In [8]: set(user3_nbrs).difference(user1_nbrs)
Out[8]: {'repo1'}
```

# Let's practice!