



NETWORK ANALYSIS IN PYTHON II

Concept of projection

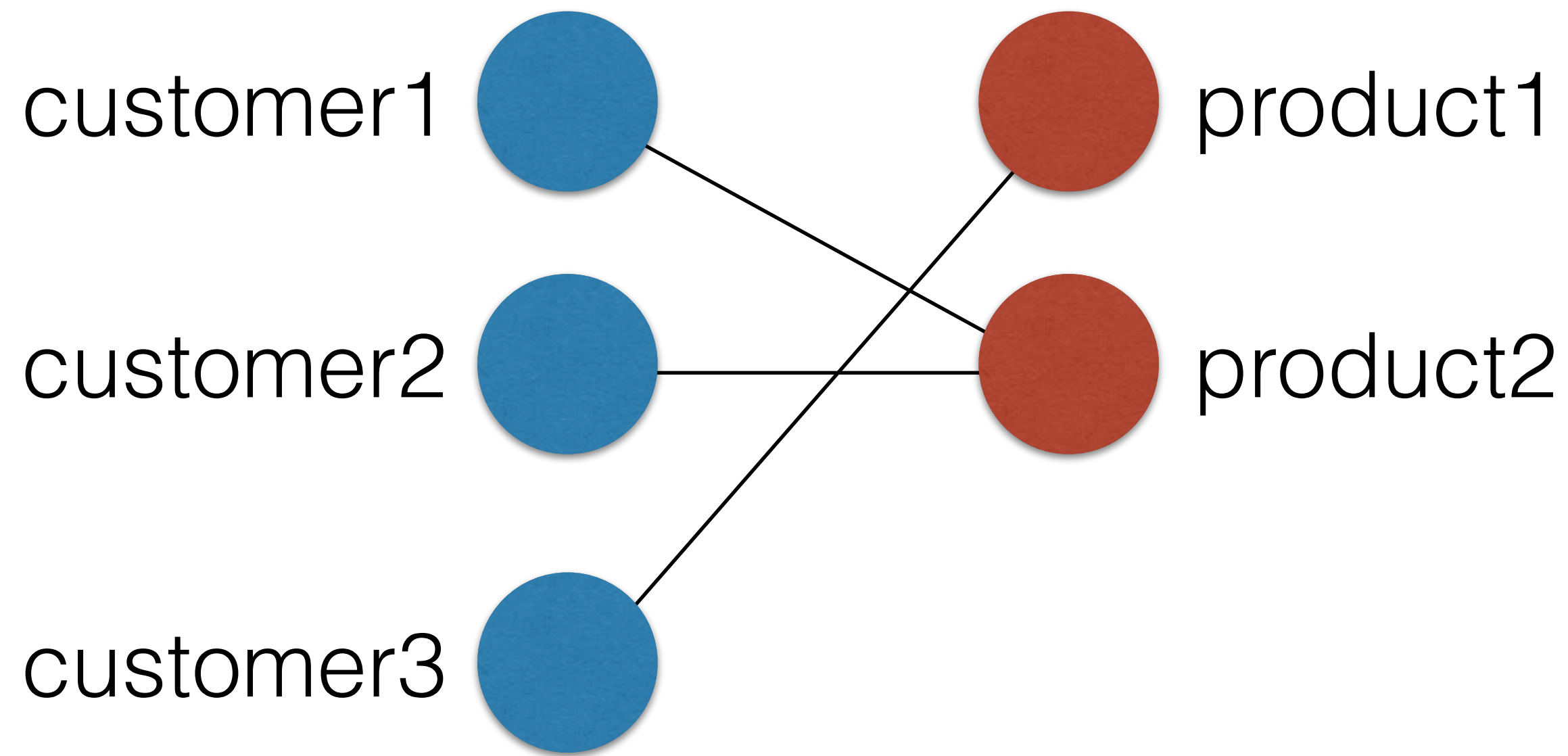
Projection

- Useful to investigate the relationships between nodes on one partition
- Conditioned on the connections to the nodes in the other partition



Projection

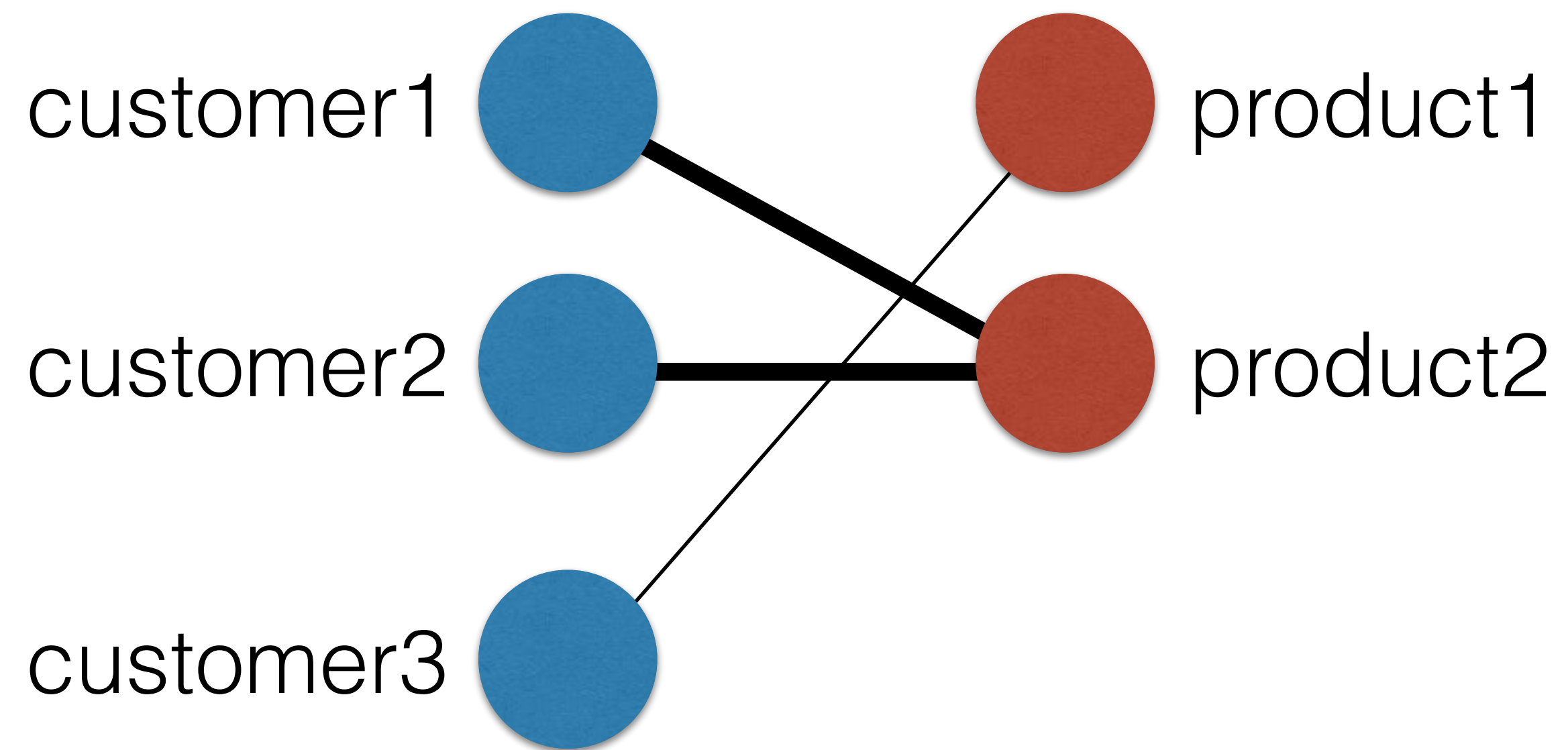
- Unipartite representation of bipartite connectivity





Projection

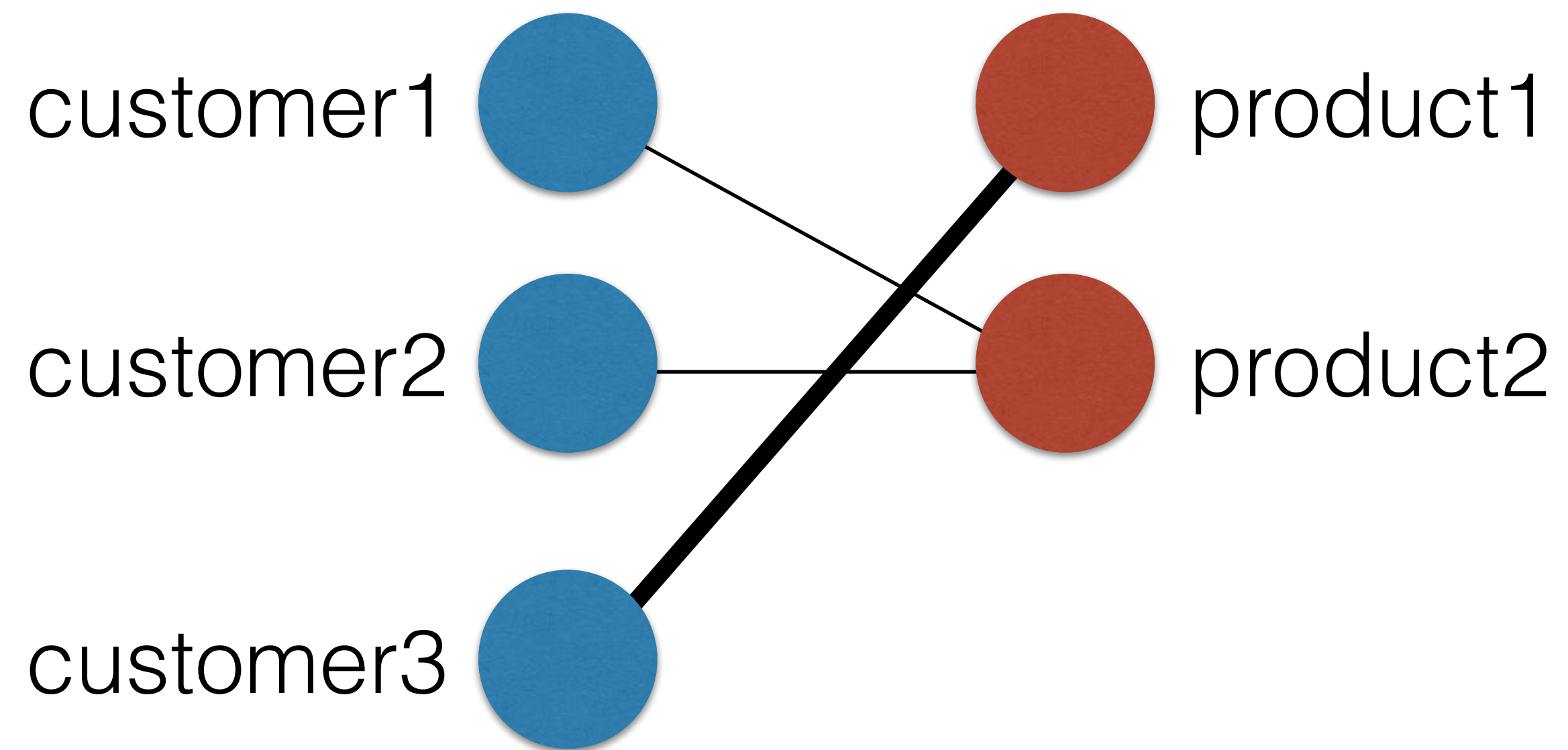
- Unipartite representation of bipartite connectivity





Projection

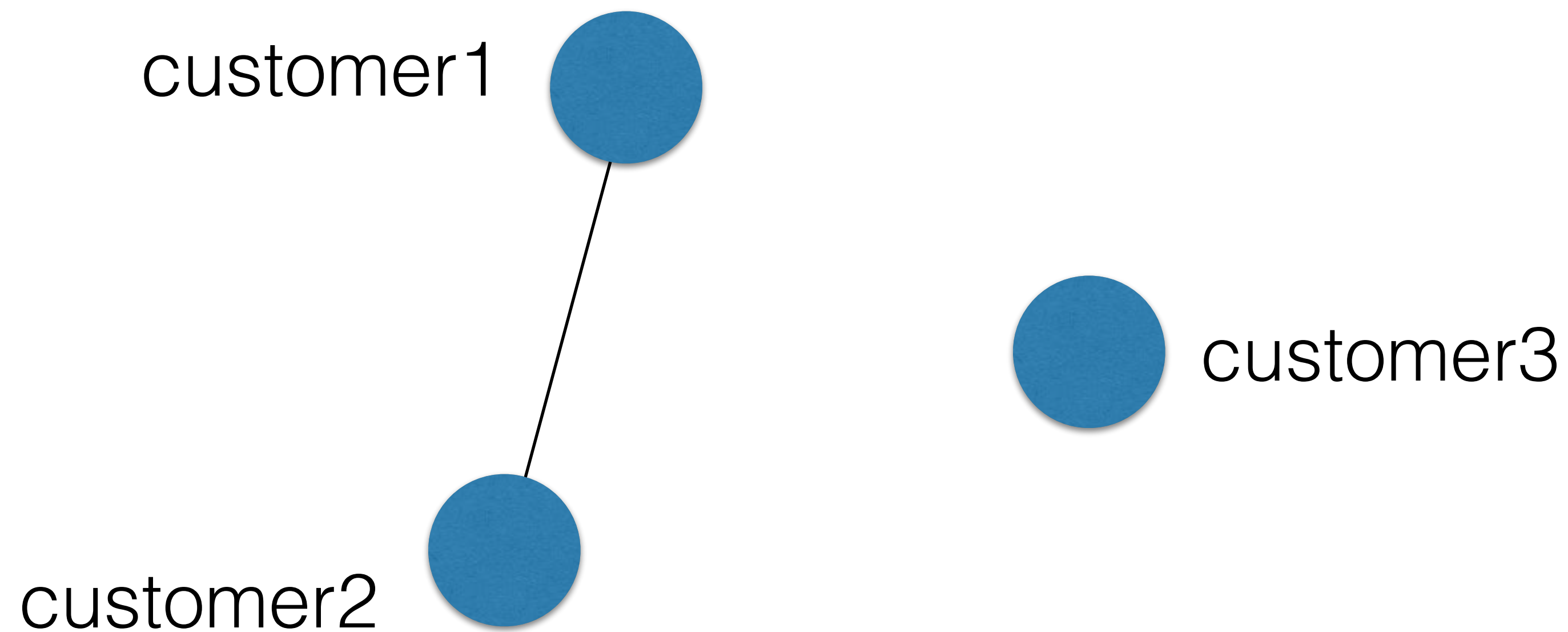
- Unipartite representation of bipartite connectivity





Projection

- Unipartite representation of bipartite connectivity





Graphs on Disk

- Flat edge lists
- CSV files: nodelist + metadata, edgelist + metadata



Reading network data

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

```
In [2]: G = nx.read_edgelist('american-revolution.txt')
```

```
In [3]: G.edges(data=True)[0:5]
```

```
Out[3]:
```

```
[('Parkman.Elias', 'LondonEnemies', {'weight': 1}),  
 ('Parkman.Elias', 'NorthCaucus', {'weight': 1}),  
 ('English.Alexander', 'StAndrewsLodge', {'weight': 1}),  
 ('NorthCaucus', 'Chadwell.Mr', {'weight': 1}),  
 ('NorthCaucus', 'Pearce.IsaacJun', {'weight': 1})]
```

```
----Text File----
```

```
Barrett.Samuel LondonEnemies {'weight': 1}  
Barrett.Samuel StAndrewsLodge {'weight': 1}  
Marshall.Thomas LondonEnemies {'weight': 1}  
Eaton.Joseph TeaParty {'weight': 1}  
Bass.Henry LondonEnemies {'weight': 1}
```




Bipartite projection

```
In [4]: G.nodes()
Out[4]: ['product2', 'customer3', 'customer1', 'product3',
...: 'customer2', 'product1']
```

```
In [5]: G.edges()
Out[5]:
[('product2', 'customer1'),
 ('product2', 'customer2'),
 ('customer3', 'product1')]
```

```
In [6]: cust_nodes = [n for n in G.nodes() if G.node[n]
...:                  ['bipartite'] == 'customers']
```

```
In [7]: cust_nodes
Out[7]: ['customer3', 'customer1', 'customer2']
```



Bipartite projection

```
In [8]: G_cust = nx.bipartite.projected_graph(G, cust_nodes)
```

```
In [9]: G_cust.nodes()
```

```
Out[9]: ['customer1', 'customer3', 'customer2']
```

```
In [10]: G_cust.edges()
```

```
Out[10]: [('customer1', 'customer2')]
```



Degree centrality

- Recall degree centrality definition

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

- Denominator: number of nodes on opposite partition



Bipartite degree centrality

```
In [11]: nx.bipartite.degree_centrality(G, cust_nodes)
```

```
Out[11]:
```

```
{'customer1': 0.333333333333333333,  
 'customer2': 0.333333333333333333,  
 'customer3': 0.333333333333333333,  
 'product1': 0.333333333333333333,  
 'product2': 0.666666666666666666,  
 'product3': 0.0}
```

```
In [12]: nx.degree_centrality(G)
```

```
Out[12]:
```

```
{'customer1': 0.2,  
 'customer2': 0.2,  
 'customer3': 0.2,  
 'product1': 0.2,  
 'product2': 0.4,  
 'product3': 0.0}
```



NETWORK ANALYSIS IN PYTHON II

Let's practice!



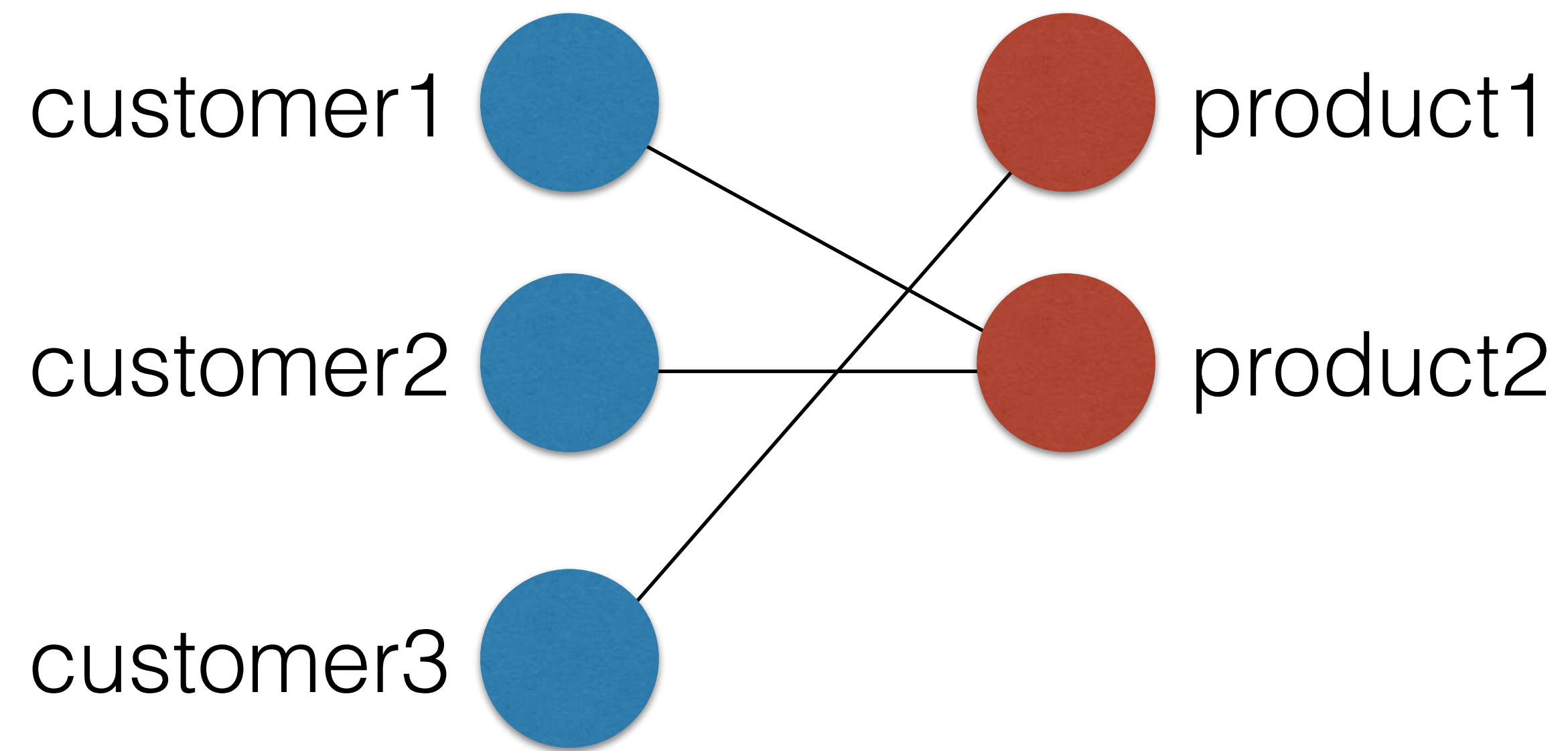
NETWORK ANALYSIS IN PYTHON II

Bipartite graphs as matrices

Matrix representation

- Rows: nodes on one partition
- Columns: nodes on other partition
- Cells: 1 if edge present, else 0

Matrix representation



	1	2
1		
2		
3		



Example code

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

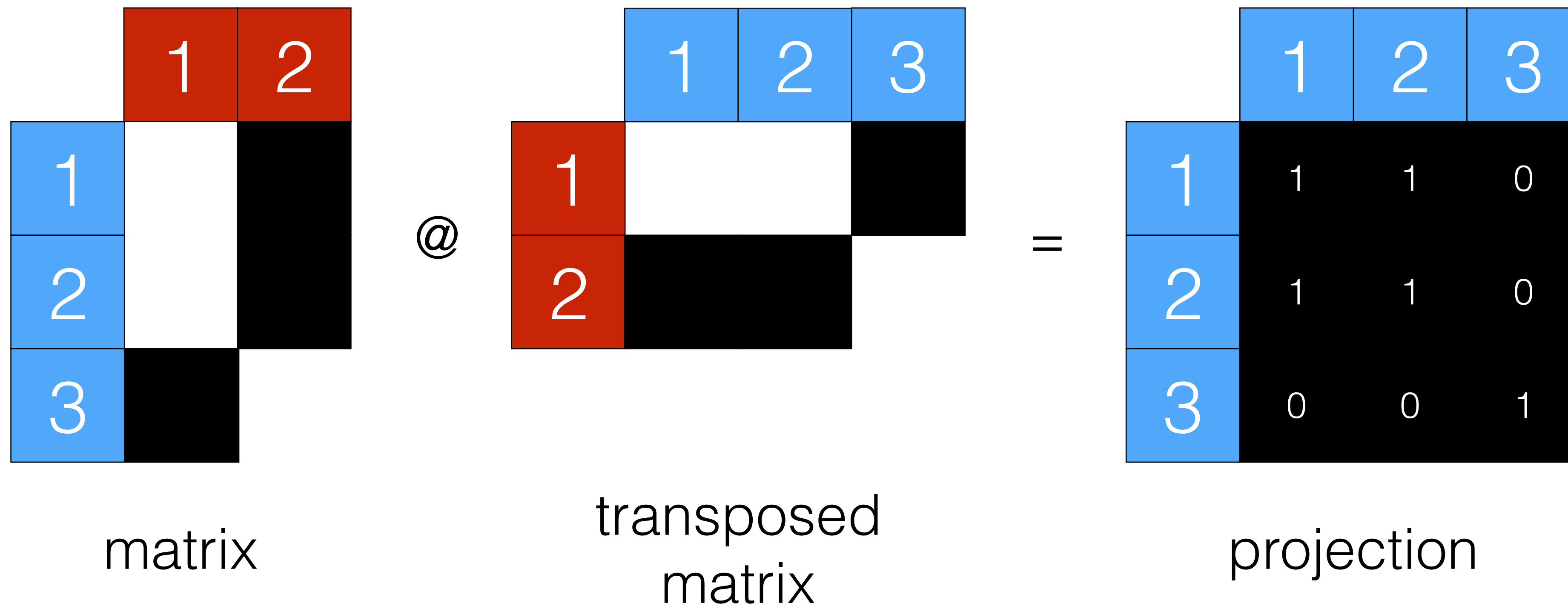
In [2]: prod_nodes = [n for n in G.nodes() if G.node[n]
....:                ['bipartite'] == 'products']

In [3]: mat = nx.bipartite.biadjacency_matrix(G,
....:                row_order=cust_nodes, column_order=prod_nodes)

In [4]: mat
Out[4]:
<3x2 sparse matrix of type '<class 'numpy.int64'>'
      with 3 stored elements in Compressed Sparse Row format>
```

Matrix projection

- Projection computable using matrix multiplication



The diagram illustrates the computation of a projection matrix using matrix multiplication. It shows three matrices: the original matrix, its transposed matrix, and the resulting projection matrix.

matrix

	1	2
1		
2		
3		

@

transposed matrix

	1	2	3
1			
2			

=

projection

	1	2	3
1	1	1	0
2	1	1	0
3	0	0	1



Matrix projection

- Projection computable using matrix multiplication

The diagram illustrates the projection operation using matrix multiplication. It shows three matrices: Matrix A (3x3), Matrix B (3x4), and Matrix C (3x4), connected by an '@' symbol and an equals sign.

Matrix A (3x3):

	1	2
1		
2		
3		

Matrix B (3x4):

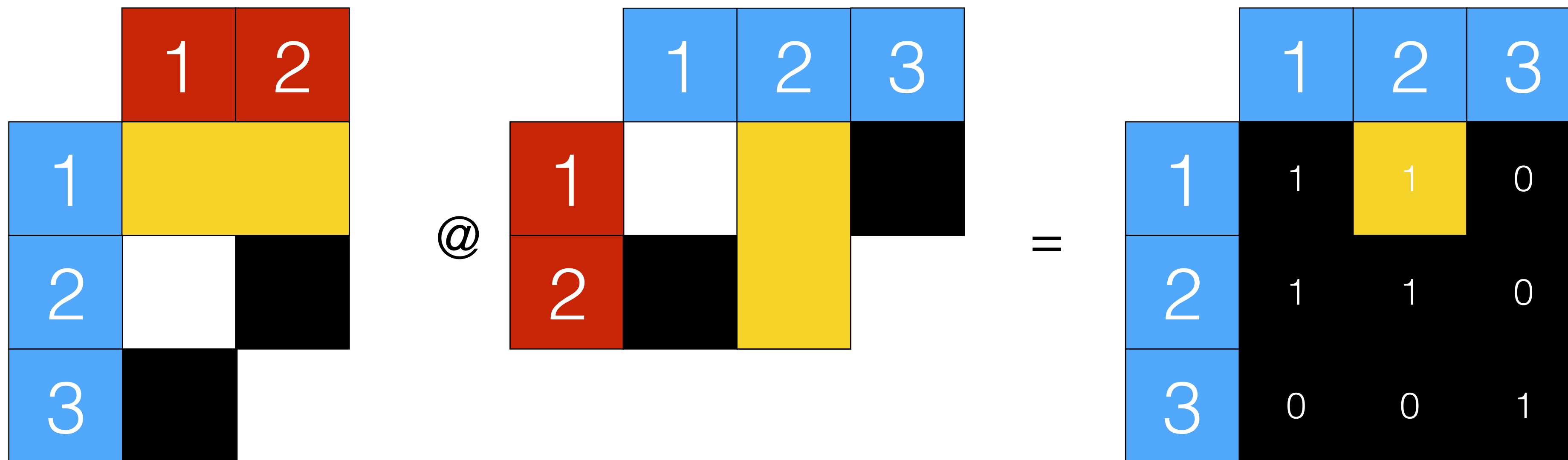
	1	2	3
1			
2			

Matrix C (3x4):

	1	2	3
1	1	1	0
2	1	1	0
3	0	0	1

Matrix projection

- Projection computable using matrix multiplication



The diagram illustrates the projection of a 3x3 matrix onto a 3x3 matrix using matrix multiplication. The first matrix (left) is a 3x3 matrix with columns 1 and 2 highlighted in red and column 3 in blue. The second matrix (middle) is a 3x3 matrix with columns 1 and 2 highlighted in red and column 3 in blue. The result matrix (right) is a 3x3 matrix with columns 1 and 2 highlighted in blue and column 3 in blue. The result matrix is the product of the first two matrices.

	1	2	3
1	1	1	0
2	1	1	0
3	0	0	1

@

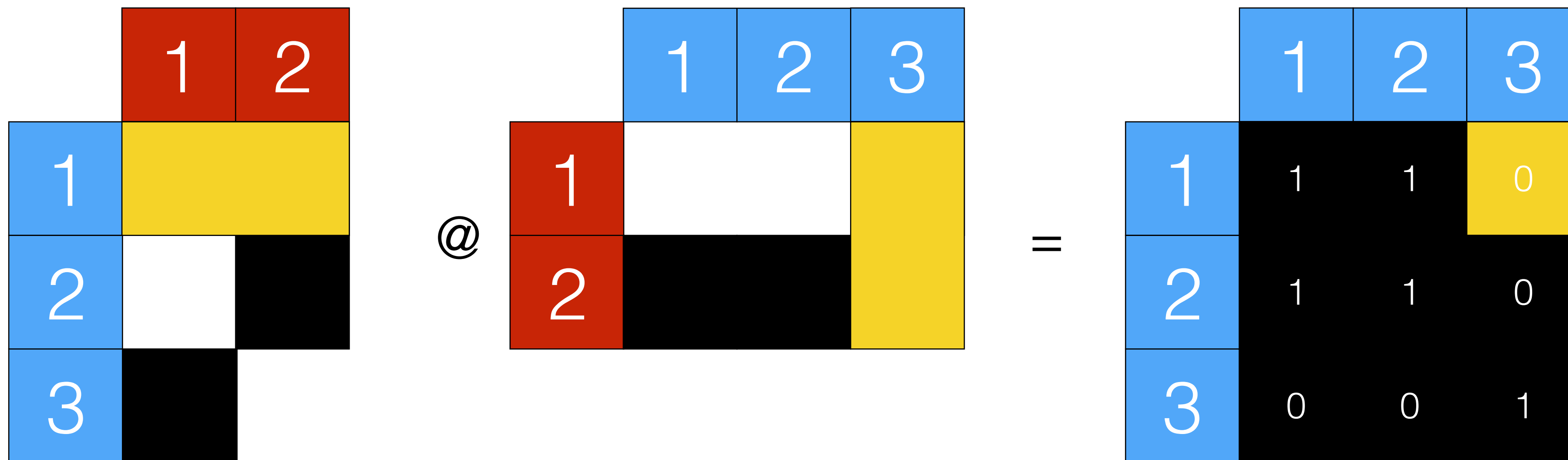
	1	2	3
1	1	1	0
2	1	1	0
3	0	0	1

=

	1	2	3
1	1	1	0
2	1	1	0
3	0	0	1

Matrix projection

- Projection computable using matrix multiplication



The diagram illustrates the matrix projection operation $A @ B = C$ using 3x3 matrices. The first matrix (A) has a blue first column with values 1, 2, 3 and a red top row with values 1, 2. The second matrix (B) has a red first column with values 1, 2 and a blue top row with values 1, 2, 3. The resulting matrix (C) has a blue first column with values 1, 2, 3 and a black 2x2 submatrix with values 1, 1, 0, 0, 0, 1.

	1	2
1	1	2
2	1	0
3	1	0

@

	1	2	3
1	1	0	0
2	1	0	0
3	0	0	1

=

	1	2	3
1	1	1	0
2	1	1	0
3	0	0	1



Matrix projection

- Projection computable using matrix multiplication

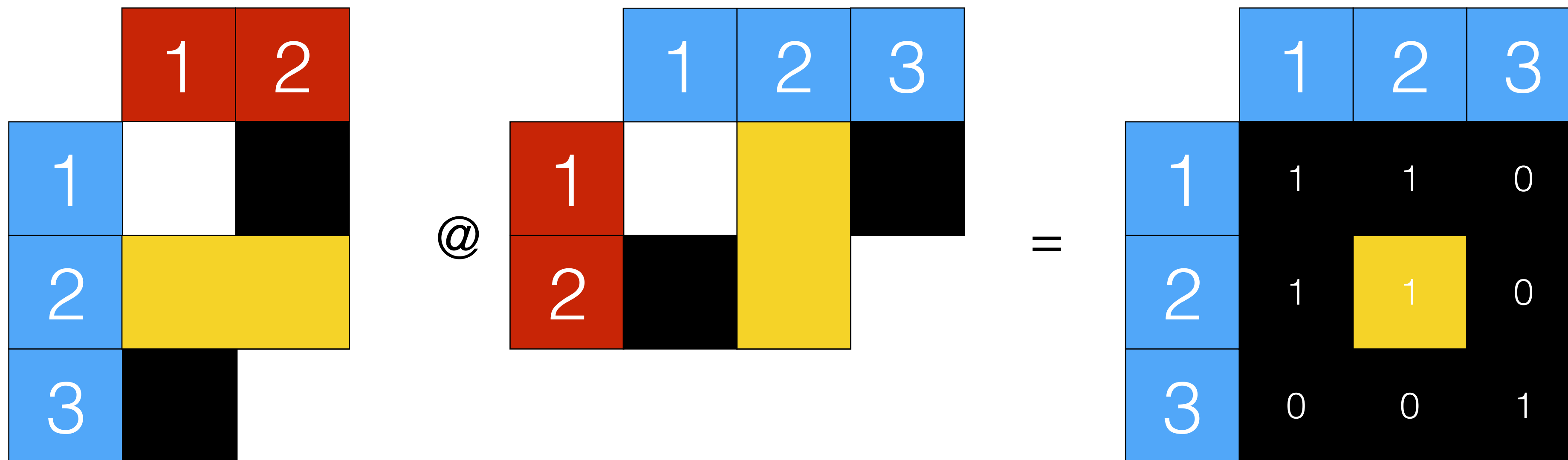
Matrix A (3x3) \otimes Matrix B (3x4) = Matrix C (3x4)

	1	2	
1	1	1	0
2	1	1	0
3	0	0	1



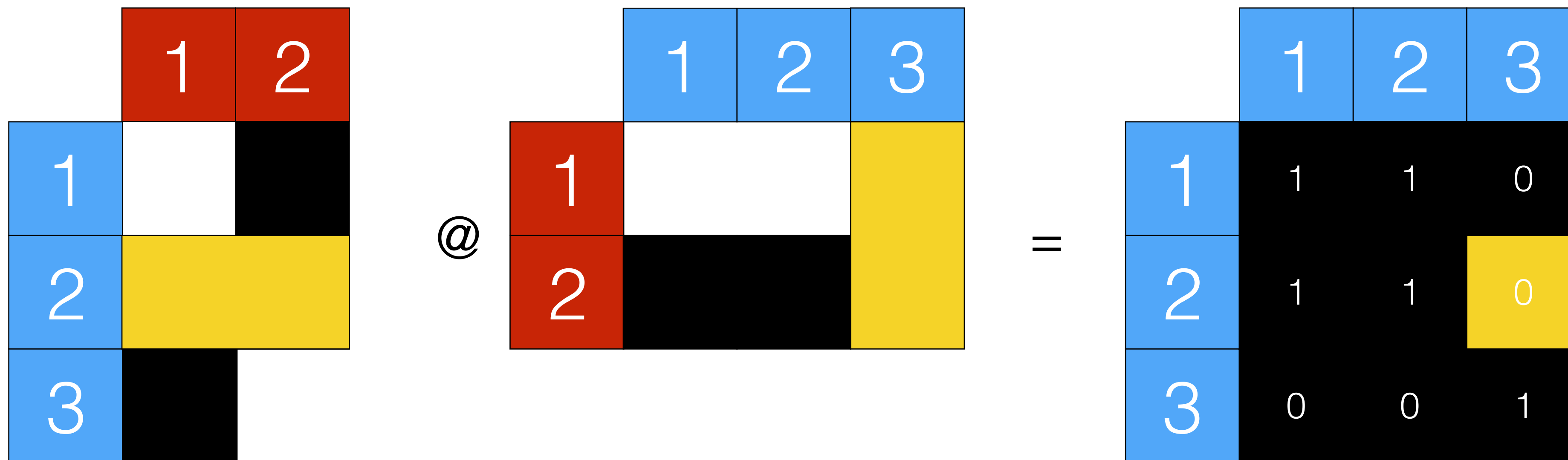
Matrix projection

- Projection computable using matrix multiplication



Matrix projection

- Projection computable using matrix multiplication



The diagram illustrates the matrix projection operation $A @ B = C$, where A is the adjacency matrix, B is the node feature matrix, and C is the resulting projected matrix.

Matrix A (Adjacency Matrix): A 3x3 matrix with rows and columns indexed 1 to 3. The top row has red cells with values 1 and 2. The middle row has a blue cell with value 1, a white cell, and a black cell. The bottom row has a blue cell with value 2, a yellow cell, and a black cell.

Matrix B (Node Feature Matrix): A 3x3 matrix with rows and columns indexed 1 to 3. The top row has blue cells with values 1, 2, and 3. The middle row has a red cell with value 1, a white cell, and a yellow cell. The bottom row has a red cell with value 2, a black cell, and a yellow cell.

Matrix C (Projected Matrix): A 3x3 matrix with rows and columns indexed 1 to 3. The top row has blue cells with values 1, 2, and 3. The middle row has a blue cell with value 1, a black cell with value 1, a black cell with value 1, and a black cell with value 0. The bottom row has a blue cell with value 2, a black cell with value 1, a black cell with value 1, and a yellow cell with value 0.



Matrix projection

- Projection computable using matrix multiplication

The diagram illustrates the projection of a 3x3 matrix A onto a 3x3 matrix B, resulting in a 3x3 matrix C. The matrices are defined as follows:

Matrix A (3x3):

	1	2	3
1	1	0	0
2	0	1	0
3	0	0	1

Matrix B (3x3):

	1	2	3
1	1	0	0
2	0	1	0
3	0	0	1

Matrix C (3x3):

	1	2	3
1	1	1	0
2	1	1	0
3	0	0	1



Matrix projection

- Projection computable using matrix multiplication

	1	2
1		
2		
3		

 @

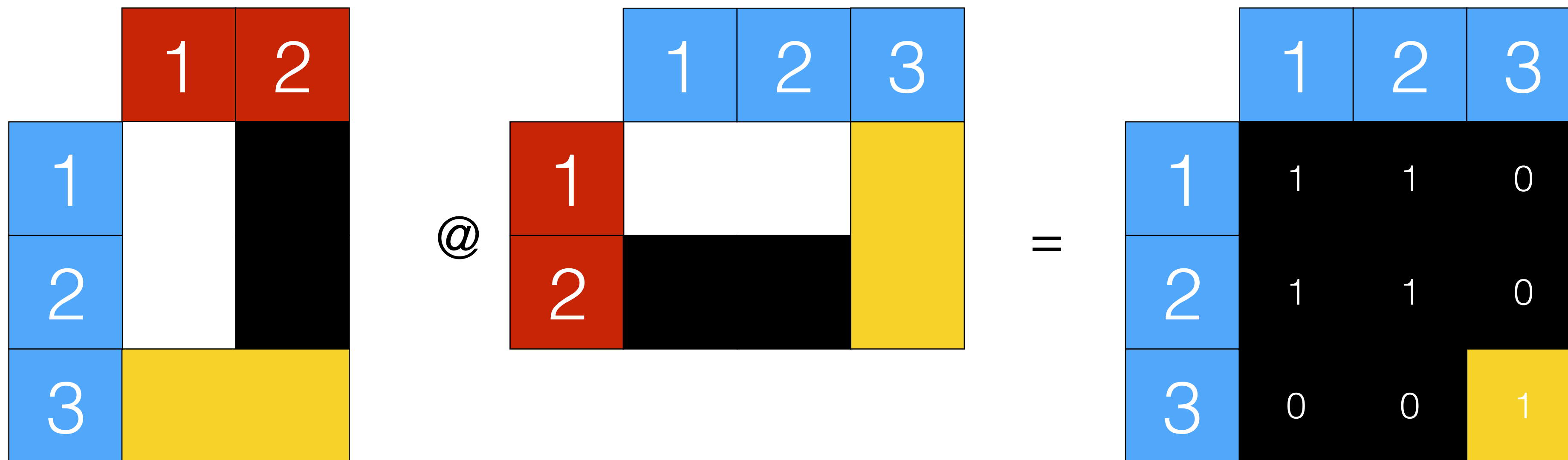
	1	2	3
1			
2			

 =

	1	2	3
1	1	1	0
2	1	1	0
3	0	0	1

Matrix projection

- Projection computable using matrix multiplication



The diagram illustrates the matrix projection operation $A @ B = C$, where A is the adjacency matrix, B is the node attribute matrix, and C is the resulting projected matrix.

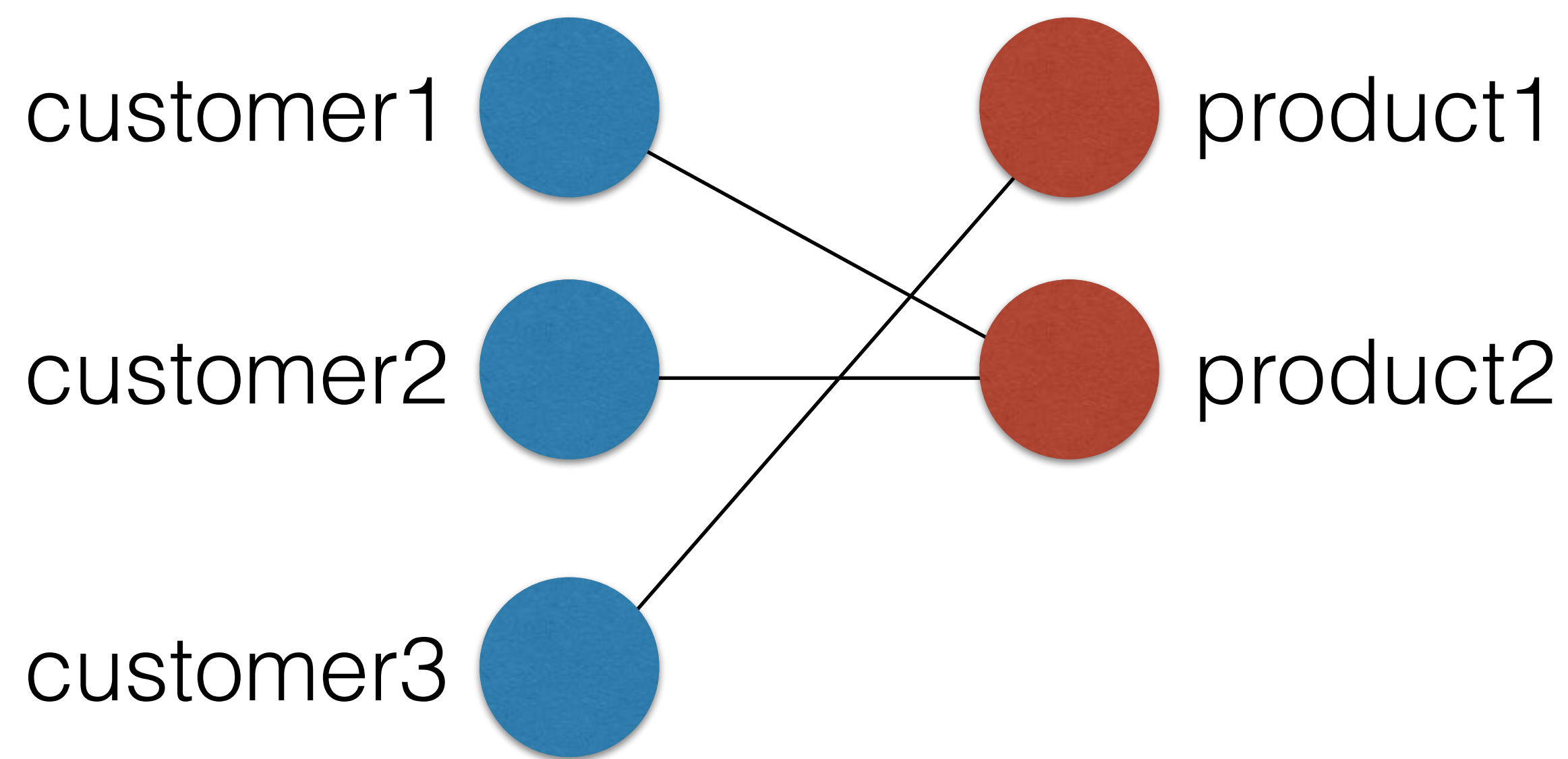
Matrix A (Adjacency Matrix): A 3x3 matrix with columns labeled 1, 2, 3. The first column has values 1, 2, 3. The second column has values 1, 2. The third column has values 1, 2.

Matrix B (Node Attribute Matrix): A 3x3 matrix with columns labeled 1, 2, 3. The first column has values 1, 2. The second column has values 1, 2. The third column has values 1, 2.

Matrix C (Projected Matrix): A 3x3 matrix with columns labeled 1, 2, 3. The first column has values 1, 2, 3. The second column has values 1, 1, 0. The third column has values 0, 0, 1.

Matrix projection

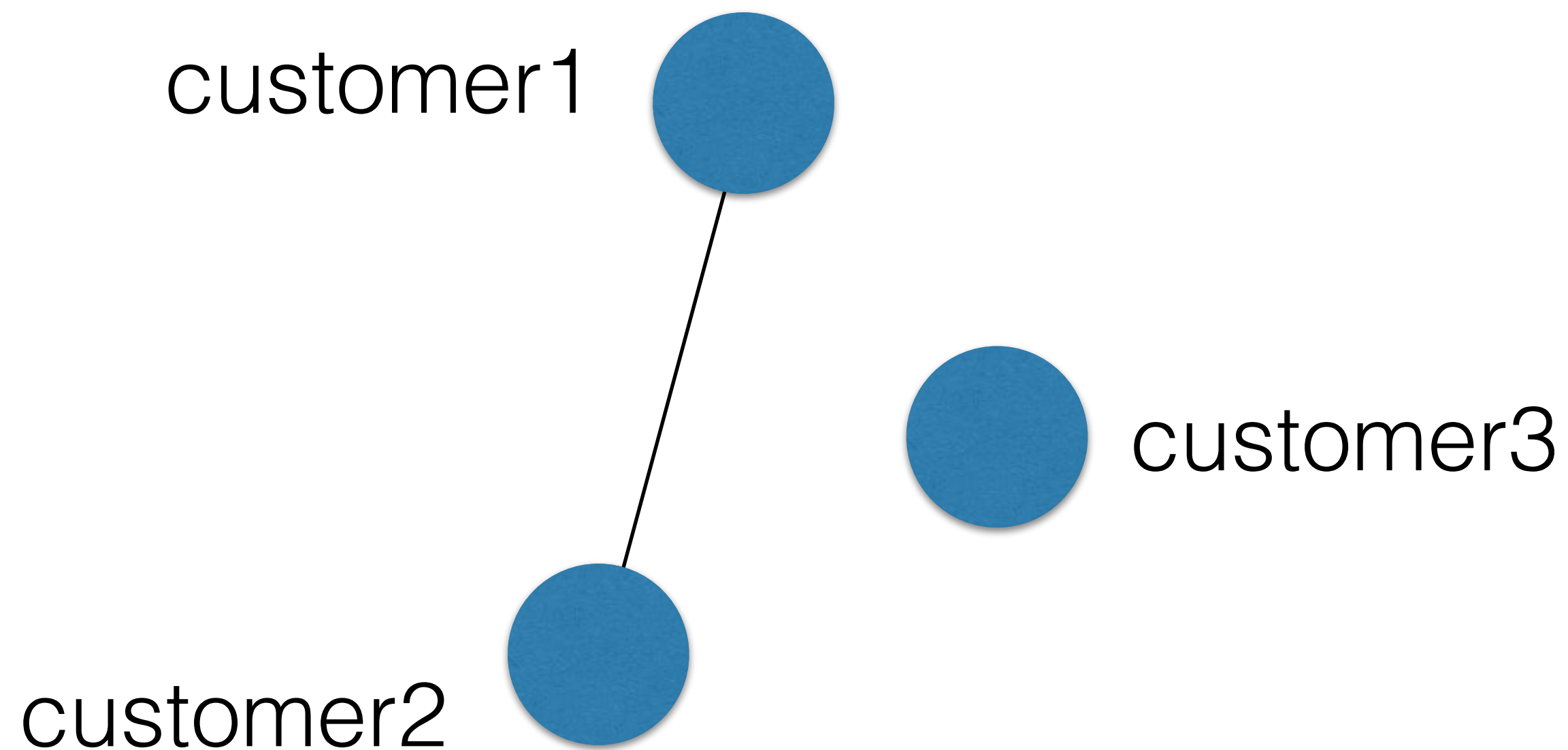
- Projection computable using matrix multiplication



	1	2	3
1	1	1	0
2	1	1	0
3	0	0	1

Matrix projection

- Projection computable using matrix multiplication



	1	2	3
1	1	1	0
2	1	1	0
3	0	0	1

Matrix multiplication in Python

```
In [5]: mat @ mat.T
Out[5]:
<5x5 sparse matrix of type '<class 'numpy.int64'>'
  with 23 stored elements in Compressed Sparse Row format>

In [6]: mat.T @ mat
Out[6]:
<10x10 sparse matrix of type '<class 'numpy.int64'>'
  with 50 stored elements in Compressed Sparse Column format>
```



NETWORK ANALYSIS IN PYTHON II

Let's practice!



NETWORK ANALYSIS IN PYTHON II

Representing network data with pandas

CSV files for network data storage

```
----CSV File—  
person,party,weight  
Barrett.Samuel,LondonEnemies,1  
Barrett.Samuel,StAndrewsLodge,1  
Marshall.Thomas,LondonEnemies,1  
Eaton.Joseph,TeaParty,1  
Bass.Henry,LondonEnemies,1
```

CSV files for network data storage

- Advantages:
 - Human-readable
 - Do further analysis with pandas
- Disadvantages:
 - Repetitive; disk space
- Two DataFrames: node and edge lists

Node list and edge list

- Node list
 - Each row is one node
 - The columns represent metadata attached to that node
- Edge list
 - Each row is one edge
 - The columns represent the metadata attached to that edge



Pandas and graphs

```
In [1]: G.nodes(data=True)
```

```
Out[1]:
```

```
[(0, {'bipartite': 0}),  
(1, {'bipartite': 0}),  
(2, {'bipartite': 0}),  
...]
```

```
In [2]: nodelist = []
```

```
In [3]: for n, d in G.nodes(data=True):
```

```
....:     node_data = dict()
```

```
....:     node_data['node'] = n
```

```
....:     node_data.update(d)
```

```
....:     nodelist.append(node_data)
```

Pandas and graphs

```
In [4]: nodelist
Out[4]:
[{'bipartite': 0, 'node': 0},
 {'bipartite': 0, 'node': 1},
 {'bipartite': 0, 'node': 2},
 {'bipartite': 0, 'node': 3},
 {'bipartite': 0, 'node': 4},...]
```



Pandas and graphs

```
In [5]: import pandas as pd
```

```
In [6]: pd.DataFrame(nodelist)
```

```
Out[6]:
```

	bipartite	node
0	0	0
1	0	1
2	0	2
3	0	3
4	0	4
5	1	5
6	1	6
7	1	7

```
In [7]: pd.DataFrame(nodelist).to_csv('my_file.csv')
```



NETWORK ANALYSIS IN PYTHON II

Let's practice!