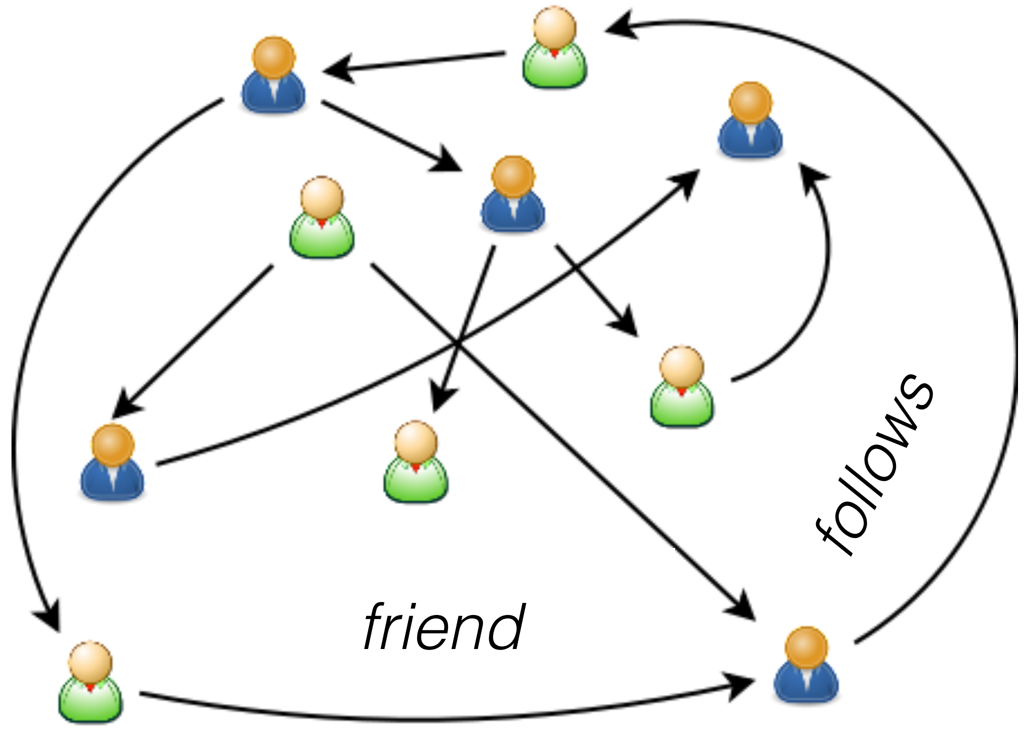
CS 591 K1: Data Stream Processing and Analytics

Spring 2020

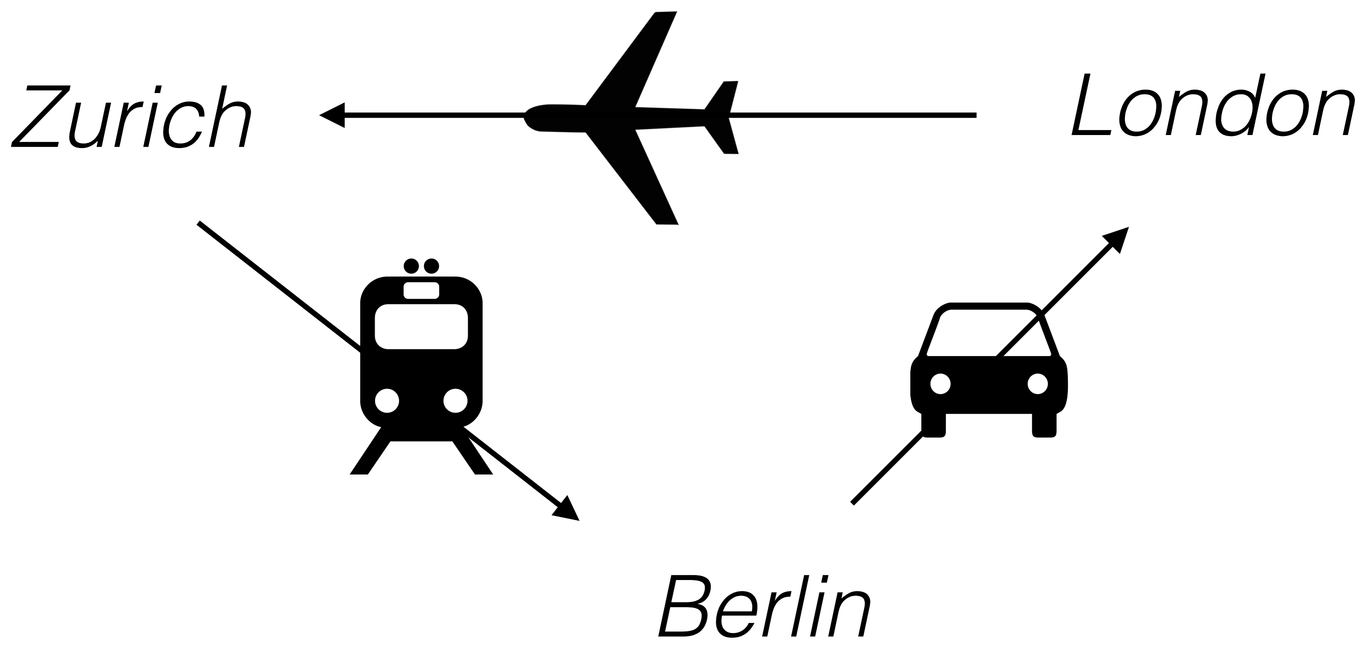
4/28: Graph Streaming

Vasiliki (Vasia) Kalavri
vkalavri@bu.edu

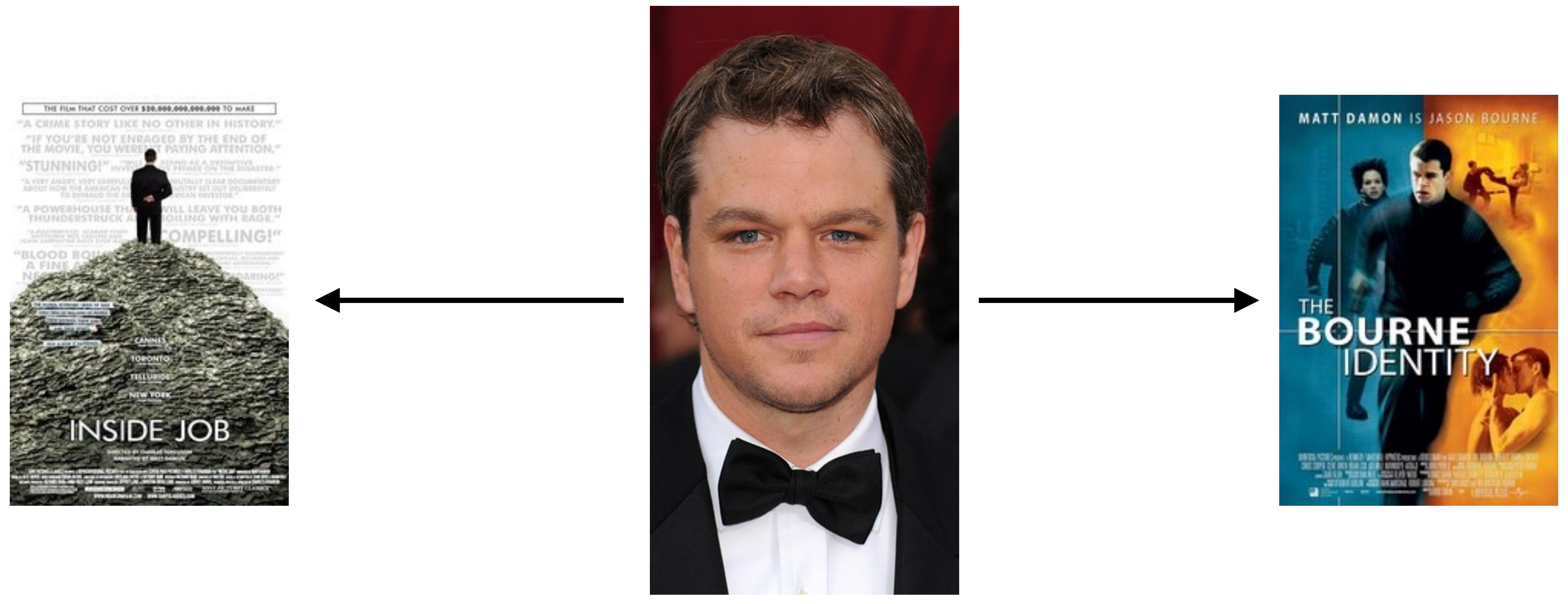
Modeling the world as a graph



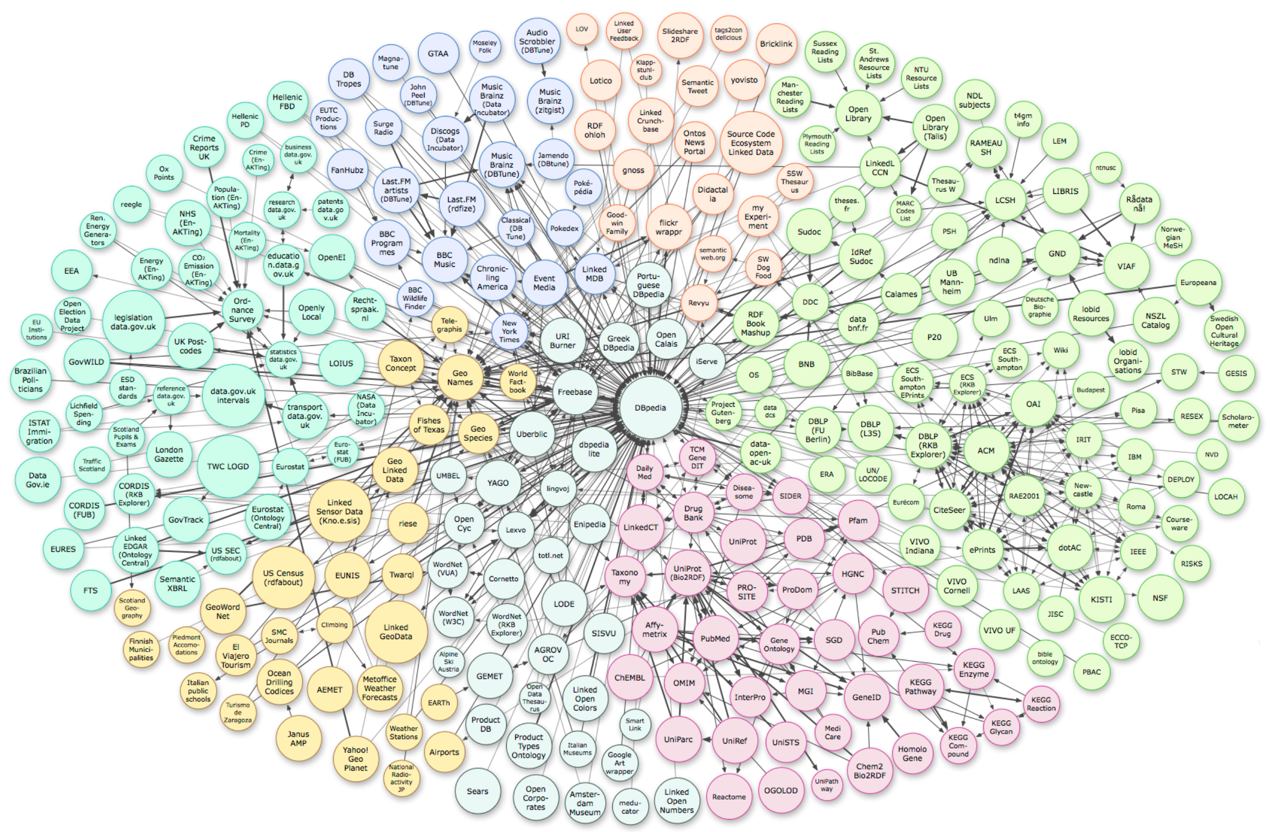
Social networks



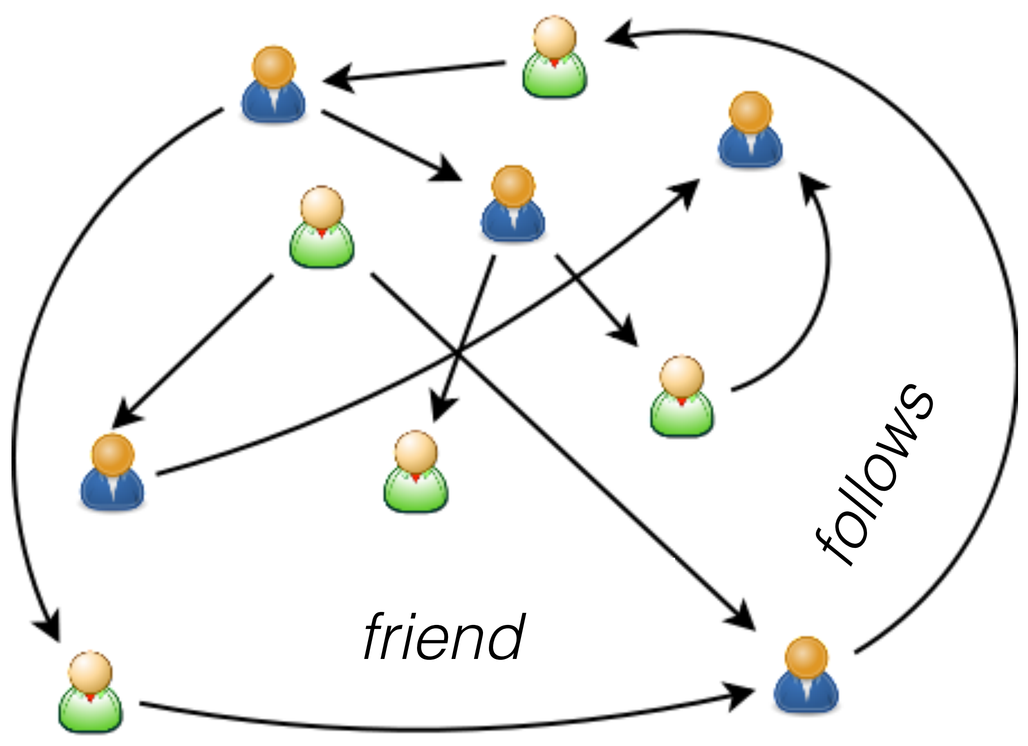
Transportation networks



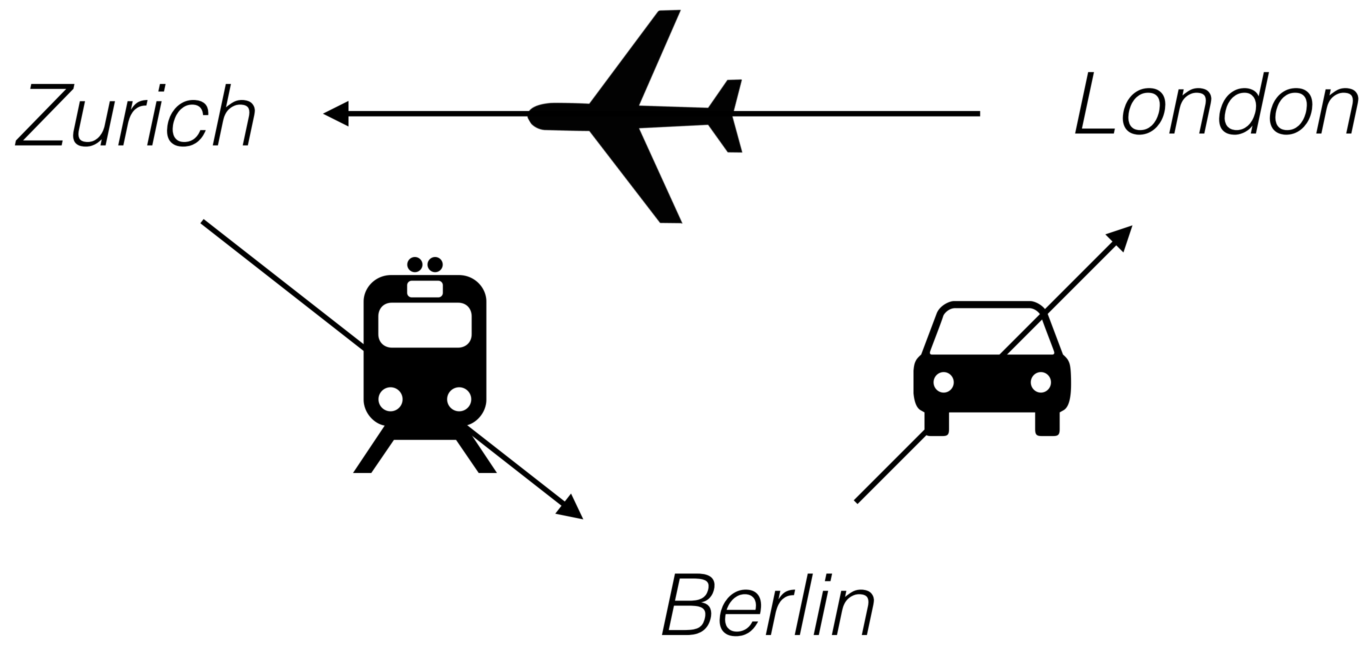
Actor-movie networks



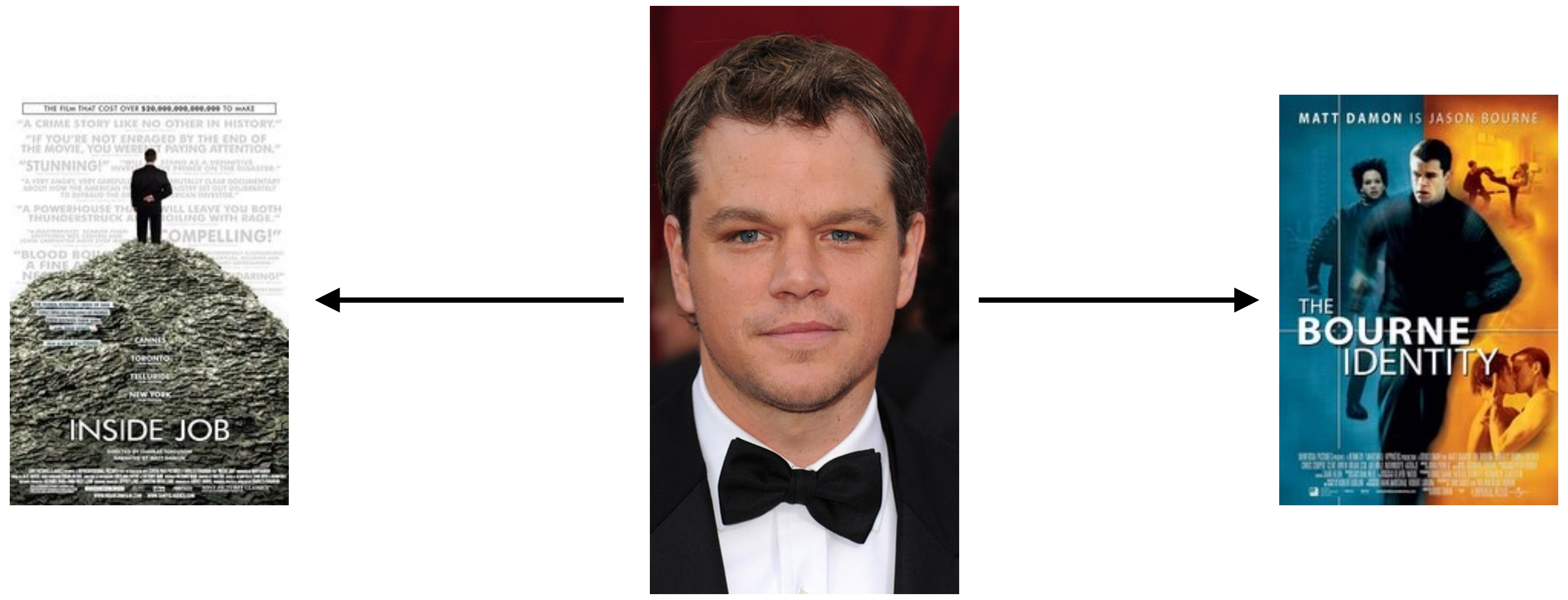
The web



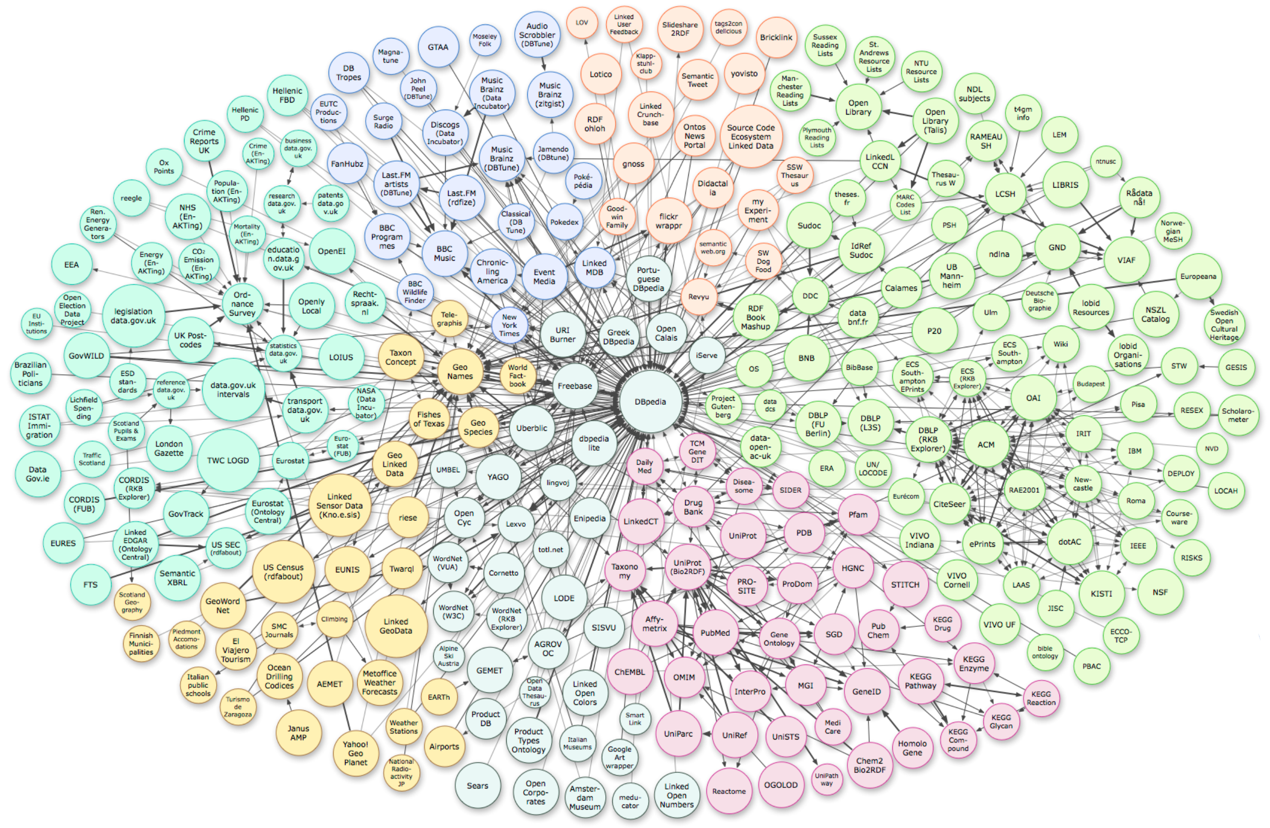
“conservative” “liberal”



What’s the cheapest way to reach Zurich from London through Berlin?

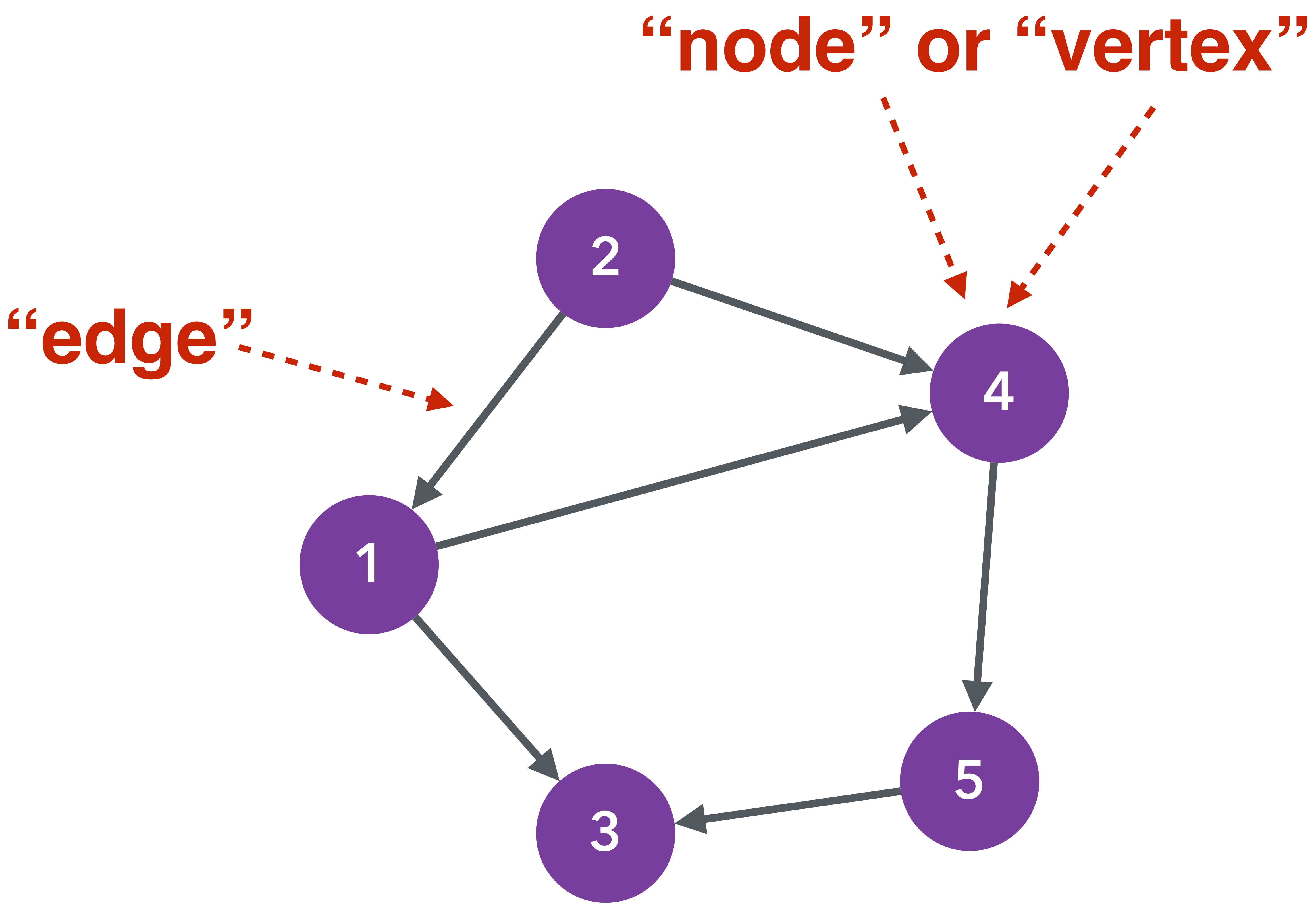


If you like “Inside job” you might also like “The Bourne Identity”



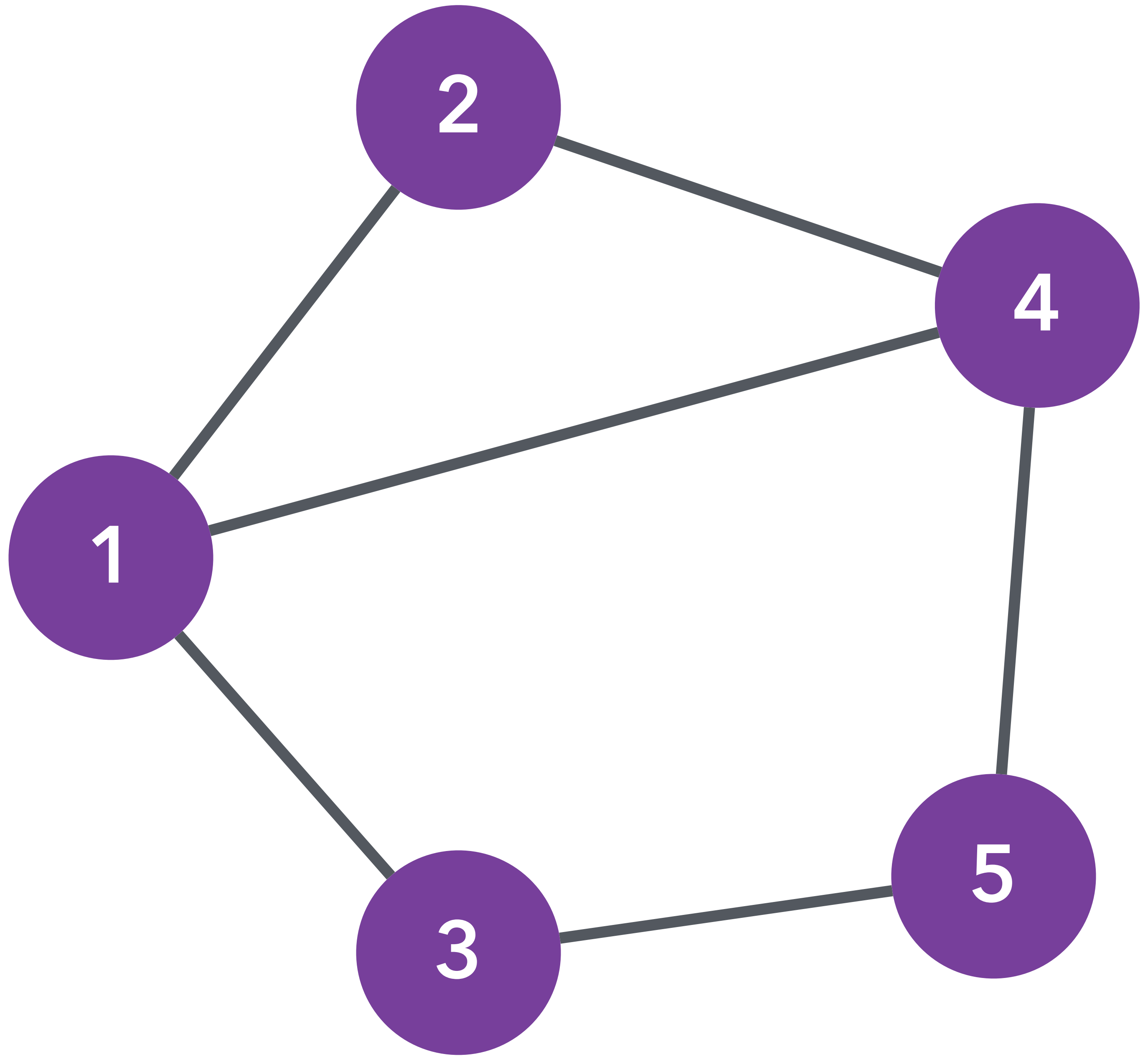
These are the top-10 relevant results for the search term “graph”

Basics



directed graph

undirected graph



Graph streams

Graph streams model interactions as **events** that update an underlying graph structure

Edge events:

A purchase, a movie rating, a like on an online post, a bitcoin transaction, a packet routed from a source to destination

Vertex events:

A new product, a new movie, a user

Vasia Kalavri @vkalavri · 9 Dec 2015

Just submitted a talk w/ @SenorCarbone at the FOSDEM @GraphDevroom! Have you submitted yours? CfP closes Dec 14 graphdevroom.github.io

Vasia Kalavri @vkalavri · 9 Dec 2015
Just submitted a talk w/ @SenorCarbone at the FOSDEM @GraphDevroom! Have you submitted yours? CfP closes Dec 14 graphdevroom.github.io

Vasia Kalavri @vkalavri · 9 Dec 2015
Just submitted a talk w/ @SenorCarbone at the FOSDEM @GraphDevroom! Have you submitted yours? CfP closes Dec 14 graphdevroom.github.io

GraphDevroom Retweeted
Vasia Kalavri @vkalavri · 9 Dec 2015
Just submitted a talk w/ @SenorCarbone at the FOSDEM @GraphDevroom! Have you submitted yours? CfP closes Dec 14 graphdevroom.github.io

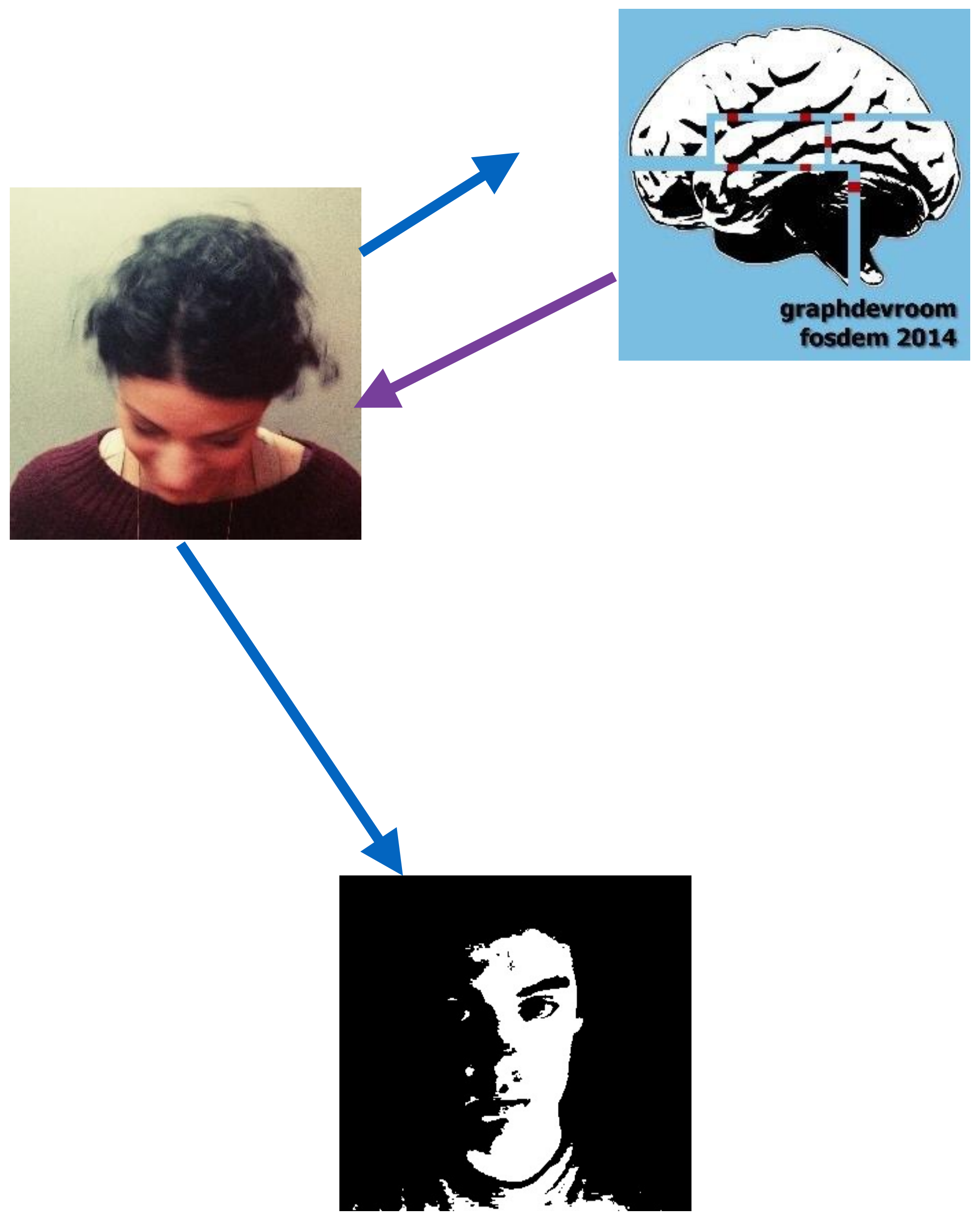
Vasia Kalavri @vkalavri · 9 Dec 2015
Just submitted a talk w/ @SenorCarbone at the FOSDEM @GraphDevroom! Have you submitted yours? CfP closes Dec 14 graphdevroom.github.io

GraphDevroom Retweeted
Vasia Kalavri @vkalavri · 9 Dec 2015
Just submitted a talk w/ @SenorCarbone at the FOSDEM @GraphDevroom! Have you submitted yours? CfP closes Dec 14 graphdevroom.github.io

Vasia Kalavri @vkalavri · 9 Dec 2015
Just submitted a talk w/ @SenorCarbone at the FOSDEM @GraphDevroom! Have you submitted yours? CfP closes Dec 14 graphdevroom.github.io

GraphDevroom Retweeted
Vasia Kalavri @vkalavri · 9 Dec 2015
Just submitted a talk w/ @SenorCarbone at the FOSDEM @GraphDevroom! Have you submitted yours? CfP closes Dec 14 graphdevroom.github.io

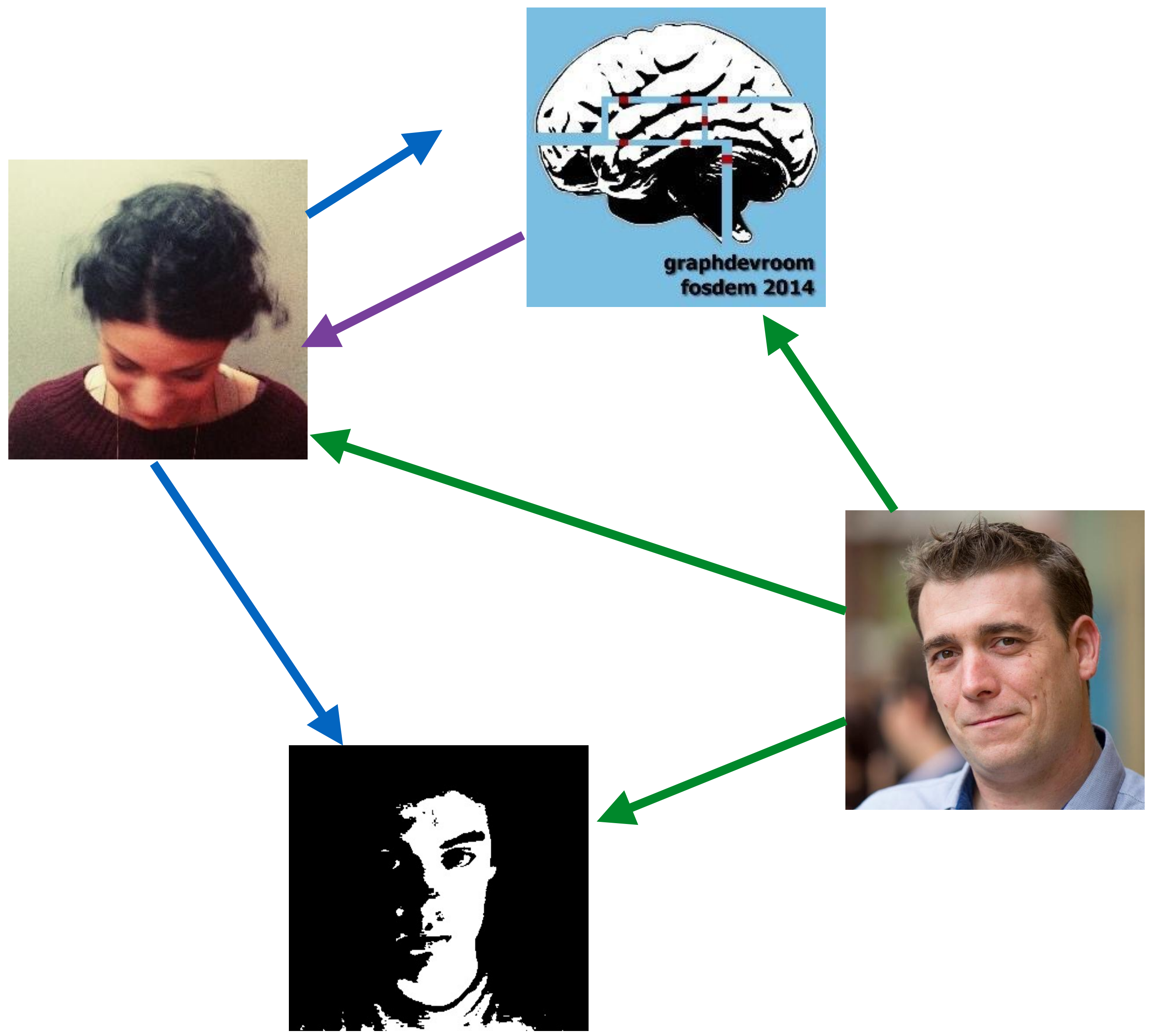
Christophe Willemsen @ikwattro · 9 Dec 2015
@vkalavri @SenorCarbone @GraphDevroom looking forward to your talk !!



Vasia Kalavri @vkalavri · 9 Dec 2015
Just submitted a talk w/ @SenorCarbone at the FOSDEM @GraphDevroom! Have you submitted yours? CfP closes Dec 14 graphdevroom.github.io

GraphDevroom Retweeted
Vasia Kalavri @vkalavri · 9 Dec 2015
Just submitted a talk w/ @SenorCarbone at the FOSDEM @GraphDevroom! Have you submitted yours? CfP closes Dec 14 graphdevroom.github.io

Christophe Willemsen @ikwattro · 9 Dec 2015
@vkalavri @SenorCarbone @GraphDevroom looking forward to your talk !!

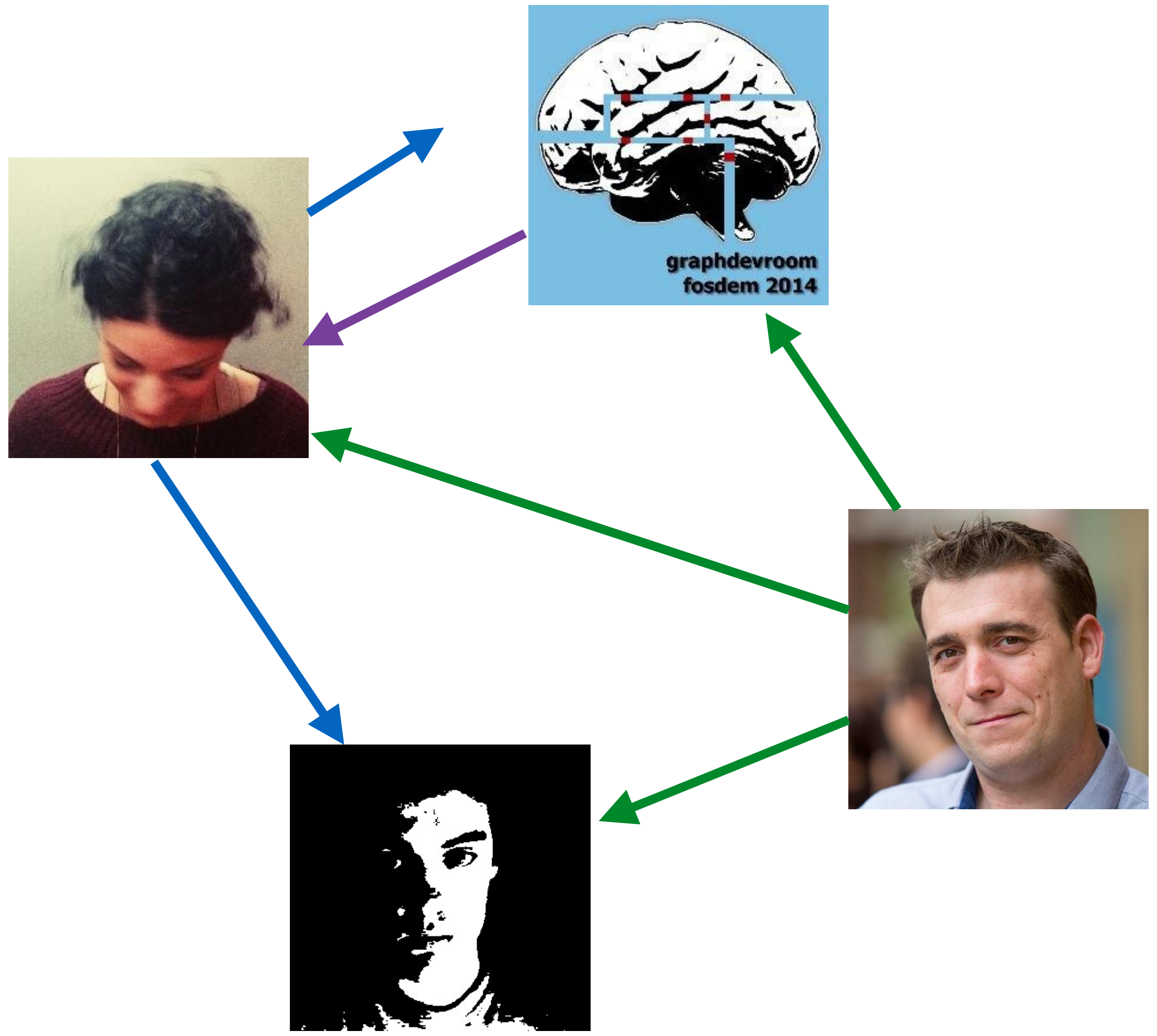


Vasia Kalavri @vkalavri · 9 Dec 2015
Just submitted a talk w/ @SenorCarbone at the FOSDEM @GraphDevroom! Have you submitted yours? CfP closes Dec 14 graphdevroom.github.io

GraphDevroom Retweeted
Vasia Kalavri @vkalavri · 9 Dec 2015
Just submitted a talk w/ @SenorCarbone at the FOSDEM @GraphDevroom! Have you submitted yours? CfP closes Dec 14 graphdevroom.github.io

Christophe Willemsen @ikwattro · 9 Dec 2015
@vkalavri @SenorCarbone @GraphDevroom looking forward to your talk !!

Paris Carbone Retweeted
Vasia Kalavri @vkalavri · 9 Dec 2015
Just submitted a talk w/ @SenorCarbone at the FOSDEM @GraphDevroom! Have you submitted yours? CfP closes Dec 14 graphdevroom.github.io

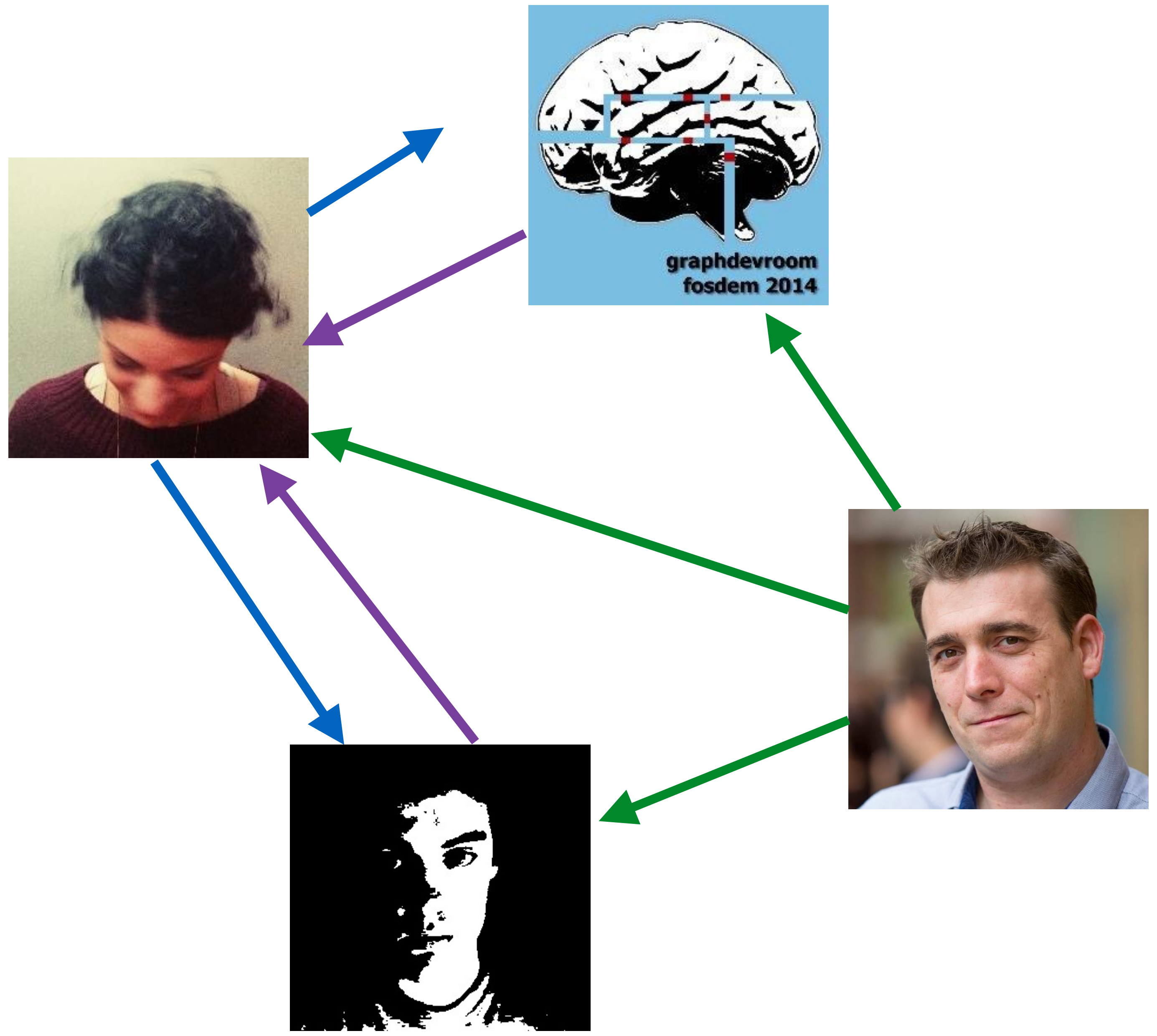


Vasia Kalavri @vkalavri · 9 Dec 2015
Just submitted a talk w/ @SenorCarbone at the FOSDEM @GraphDevroom! Have you submitted yours? CfP closes Dec 14 graphdevroom.github.io

GraphDevroom Retweeted
Vasia Kalavri @vkalavri · 9 Dec 2015
Just submitted a talk w/ @SenorCarbone at the FOSDEM @GraphDevroom! Have you submitted yours? CfP closes Dec 14 graphdevroom.github.io

Christophe Willemsen @ikwattro · 9 Dec 2015
@vkalavri @SenorCarbone @GraphDevroom looking forward to your talk !!

Paris Carbone Retweeted
Vasia Kalavri @vkalavri · 9 Dec 2015
Just submitted a talk w/ @SenorCarbone at the FOSDEM @GraphDevroom! Have you submitted yours? CfP closes Dec 14 graphdevroom.github.io



Preliminaries

Let $G(t) = (V(t), E(t))$ be the graph observed up to timestamp t.

For $t=0$, $V(t) = E(t) = \{\}$

For every $t > 0$, we receive one event:

- **Insert-only** edge stream: events indicate edge additions
- **Fully-dynamic** edge stream: events indicate edge additions or deletions

A $t+1$, the graph is obtained by inserting a new edge or deleting an existing edge (u, v) to $E(t+1)$.

If any of u, v do not already exist in $V(t)$, they are added to $V(t+1)$.

Vertex streams (not today)

Some algorithms model graph streams a sequence of vertex events.

A vertex stream consists of events that contain a vertex and all of its neighbors.

Although this model can enable a theoretical analysis of streaming algorithms, it cannot adequately model real-world unbounded streams, as the neighbors cannot be known in advance.

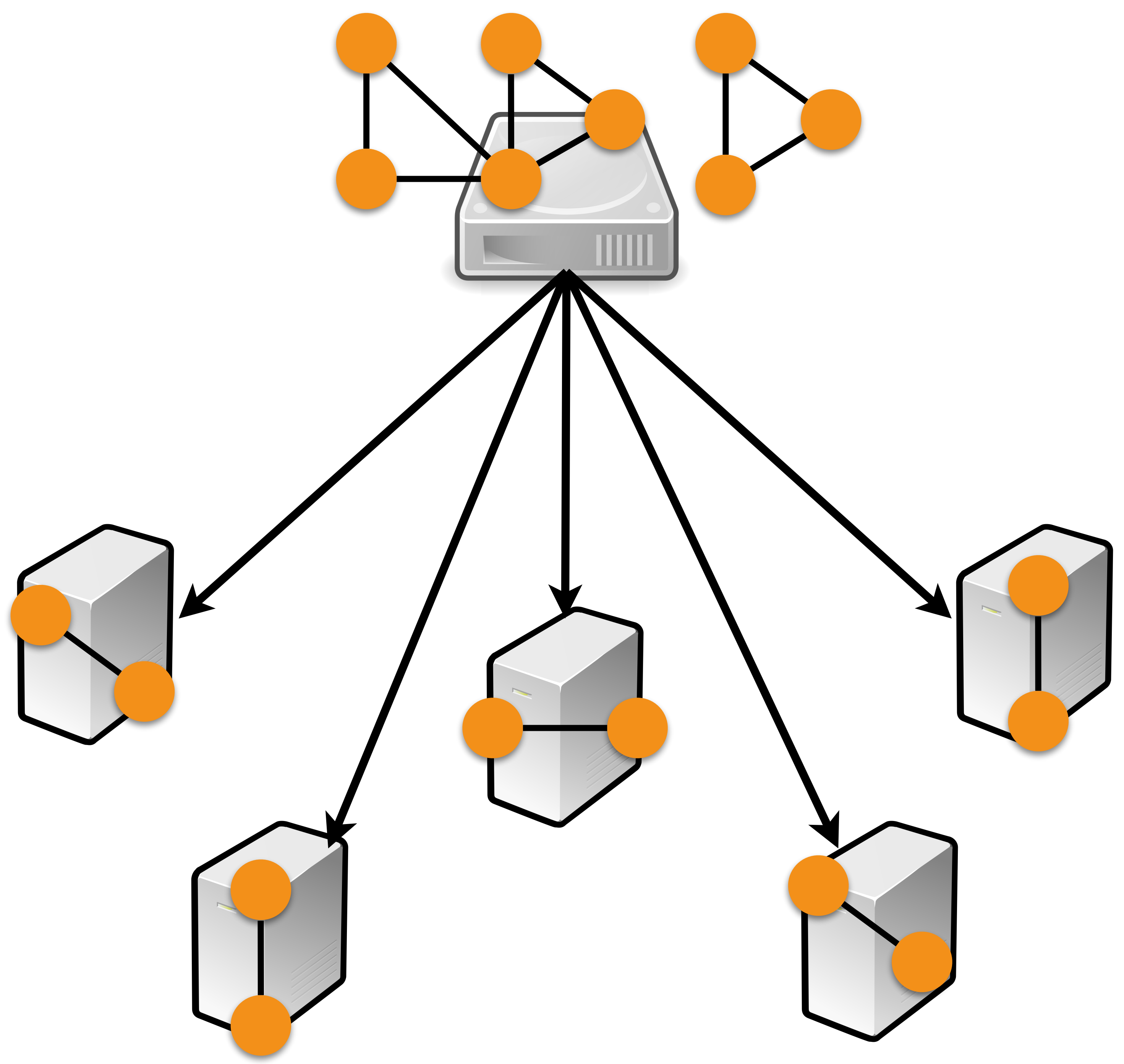
Batch Graph Processing

Batch graph processing systems, such as Apache Graph, GraphX, Pregel, operate **offline**.

They are built to analyze a **snapshot** of the real graph:

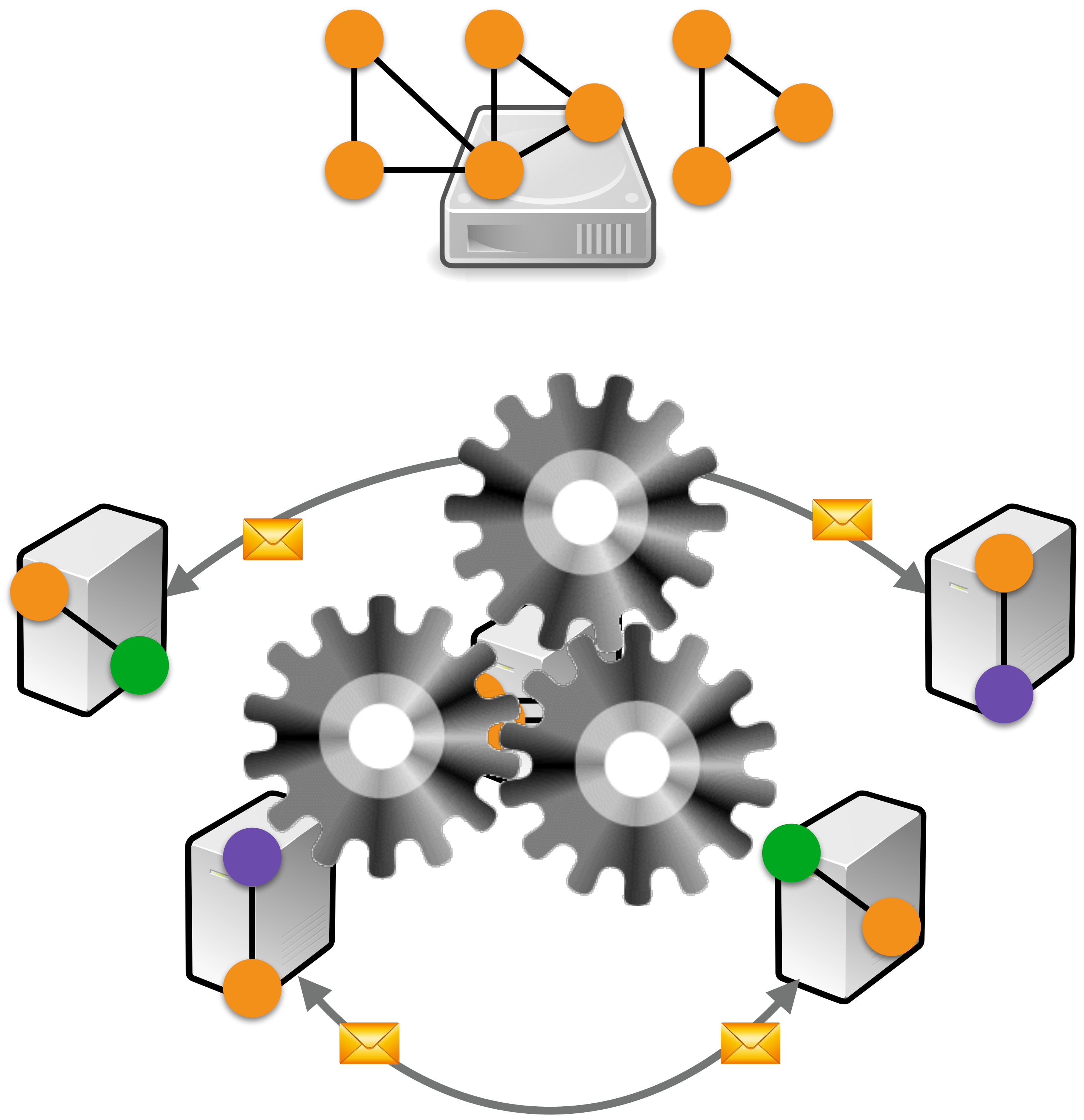
- the Facebook social network on January 30 2016
- user web logs gathered between March 1st 12:00 and 16:00
- retweets and replies for 24h after the announcement of the death of David Bowie

1. Load: read the graph from disk and partition it in memory



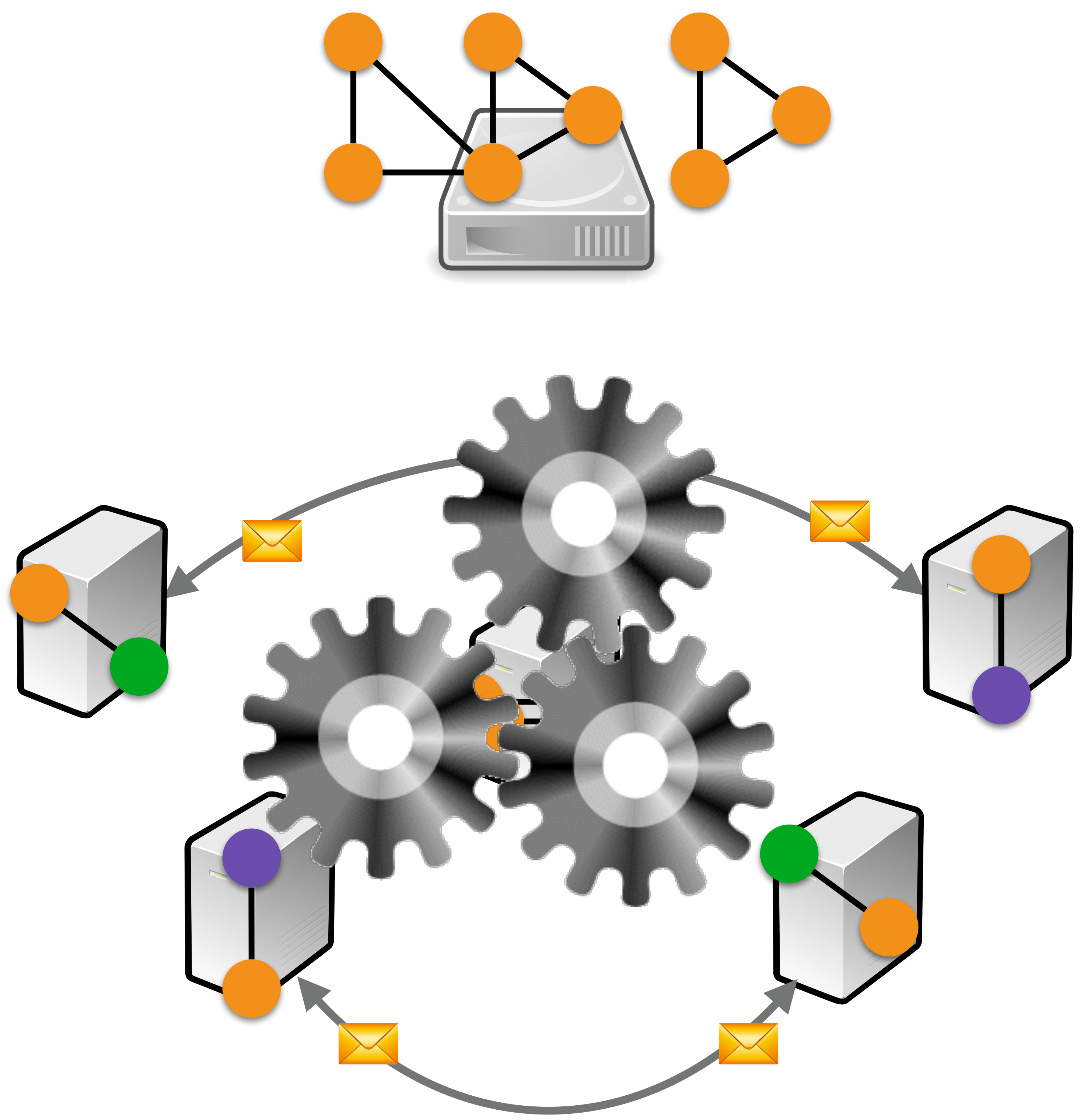
1. Load: read the graph from disk and partition it in memory

2. Compute: read and mutate the graph state



1. Load: read the graph from disk and partition it in memory

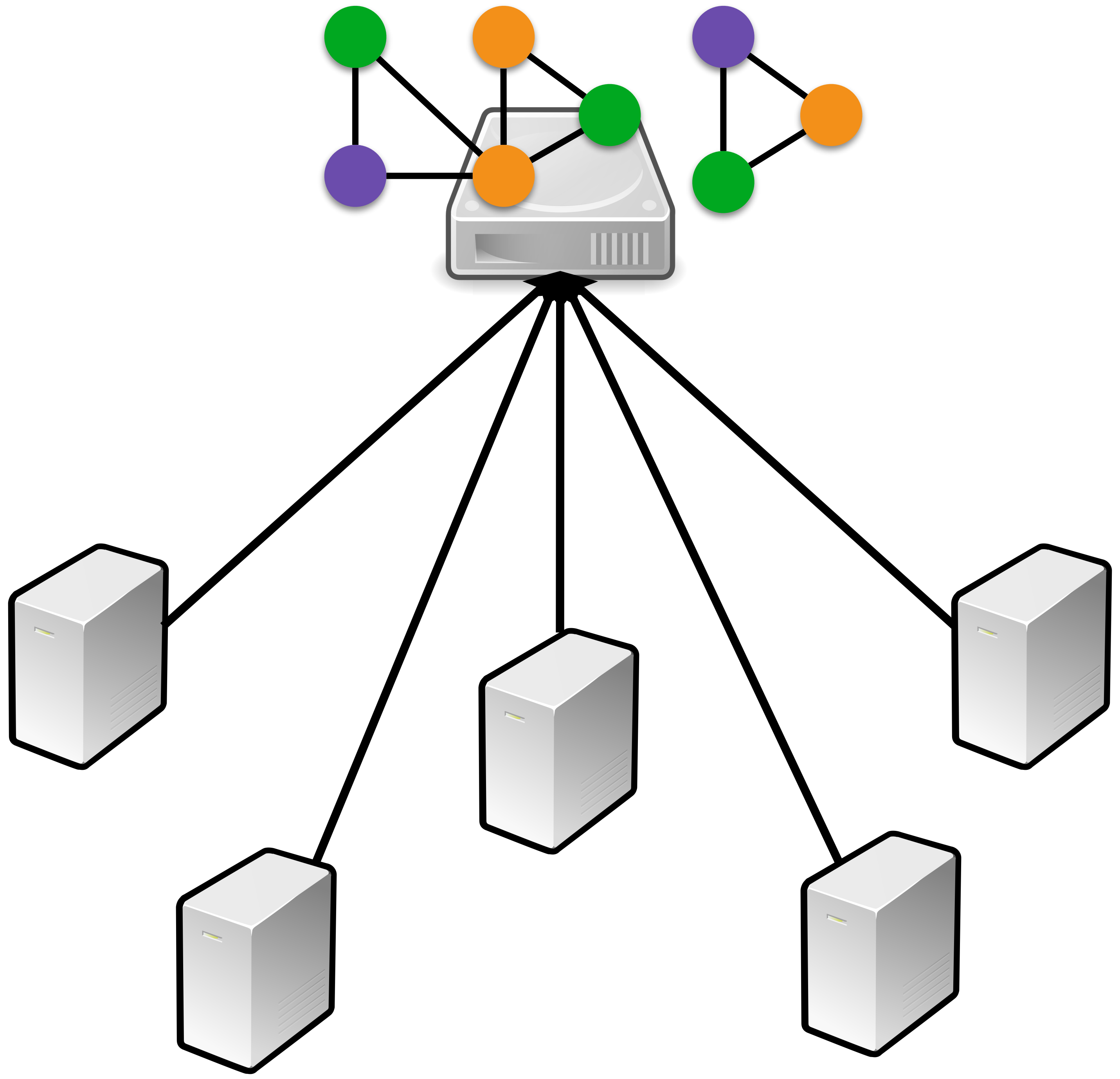
2. Compute: read and mutate the graph state



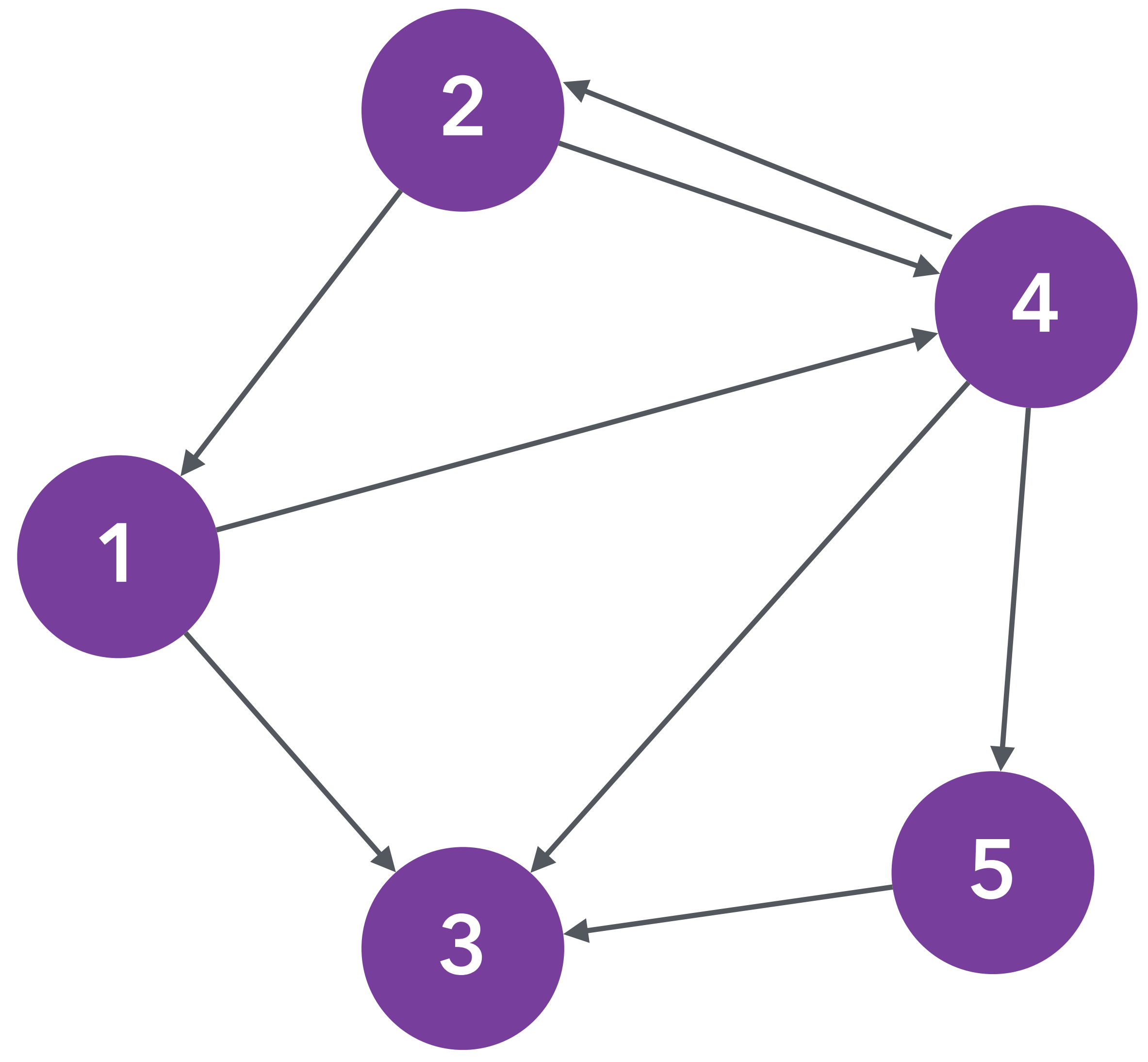
1. Load: read the graph from disk and partition it in memory

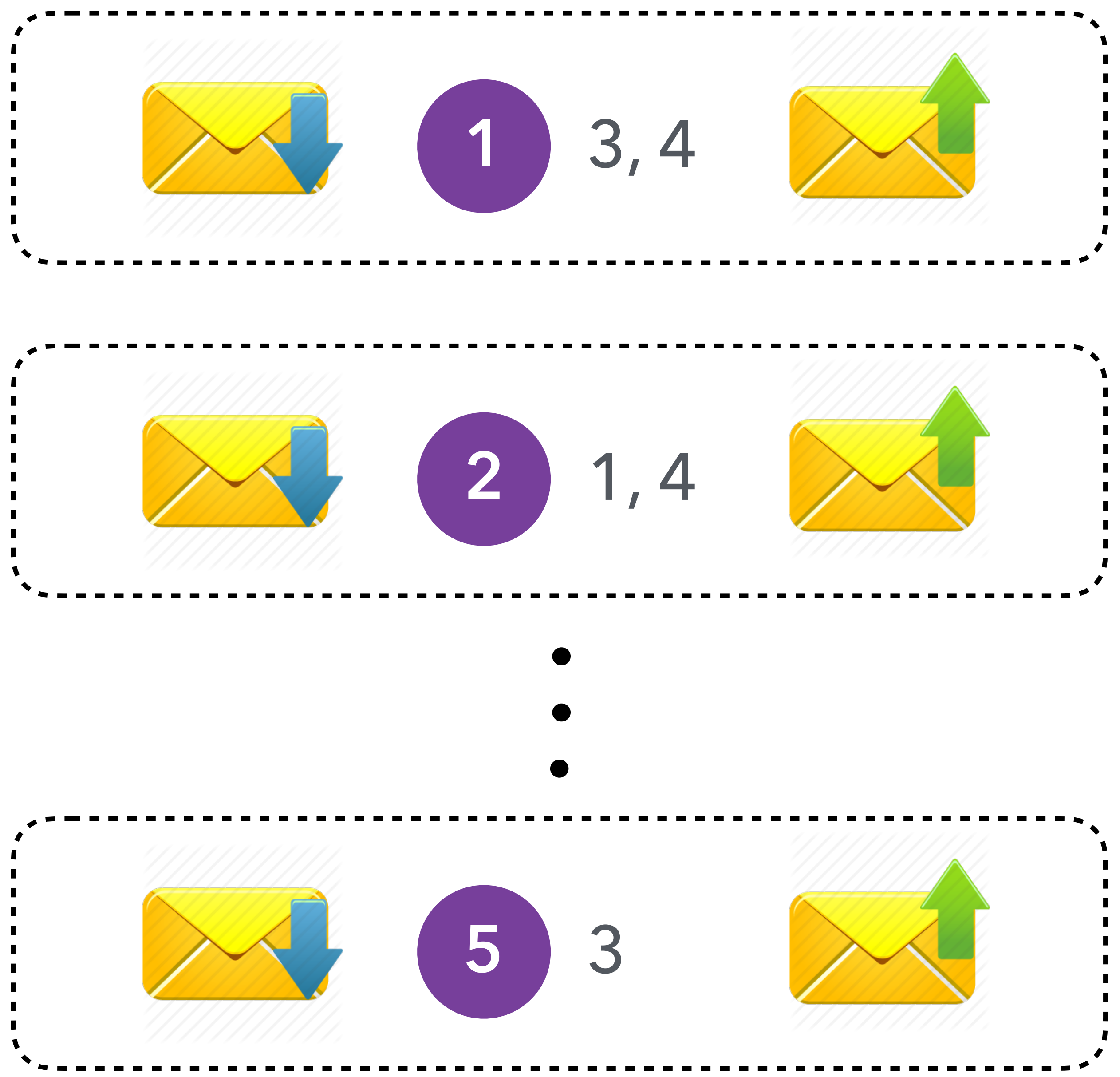
2. Compute: read and mutate the graph state

3. Store: write the final graph state back to disk

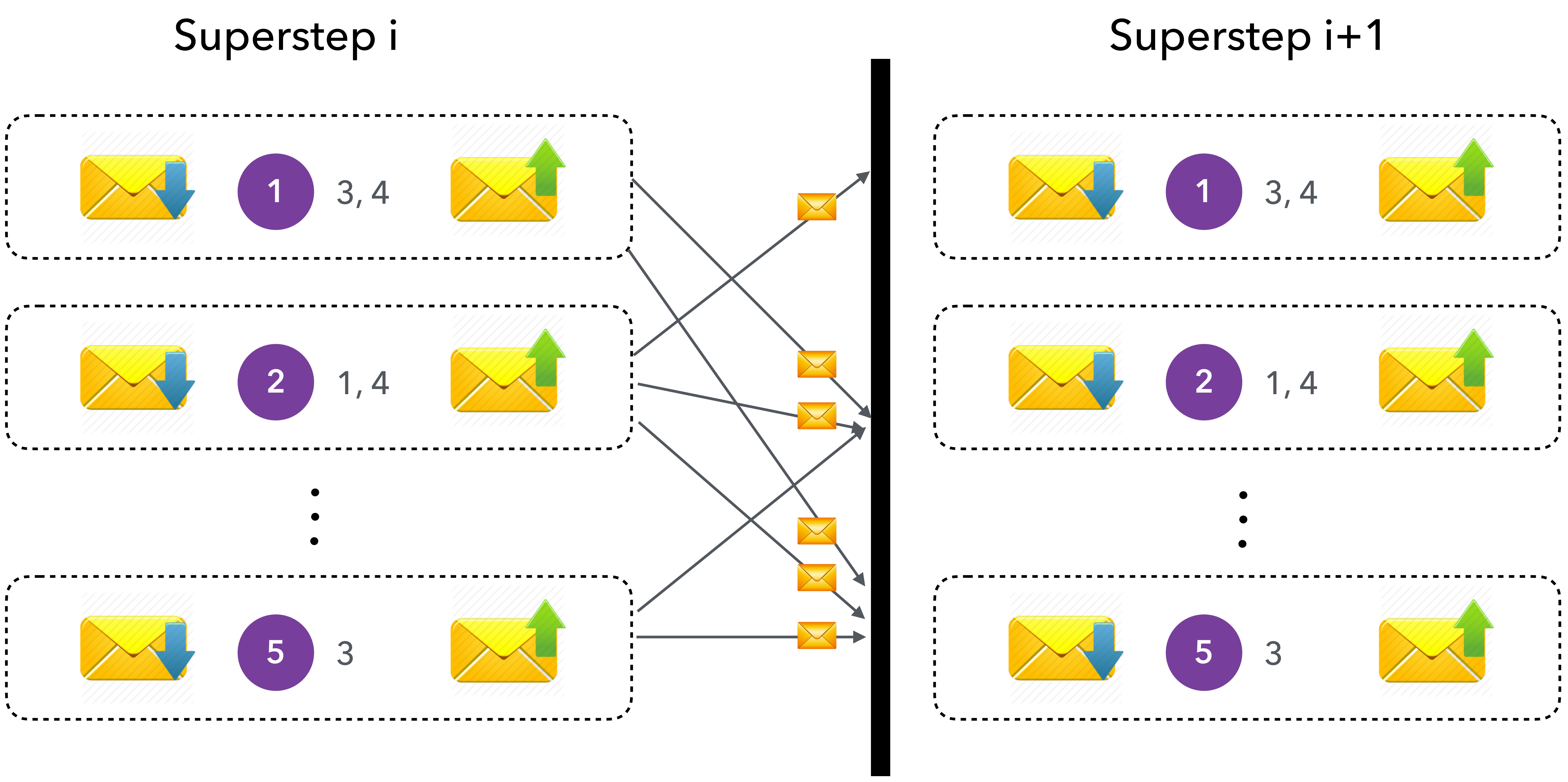


The vertex-centric model: *think like a vertex*





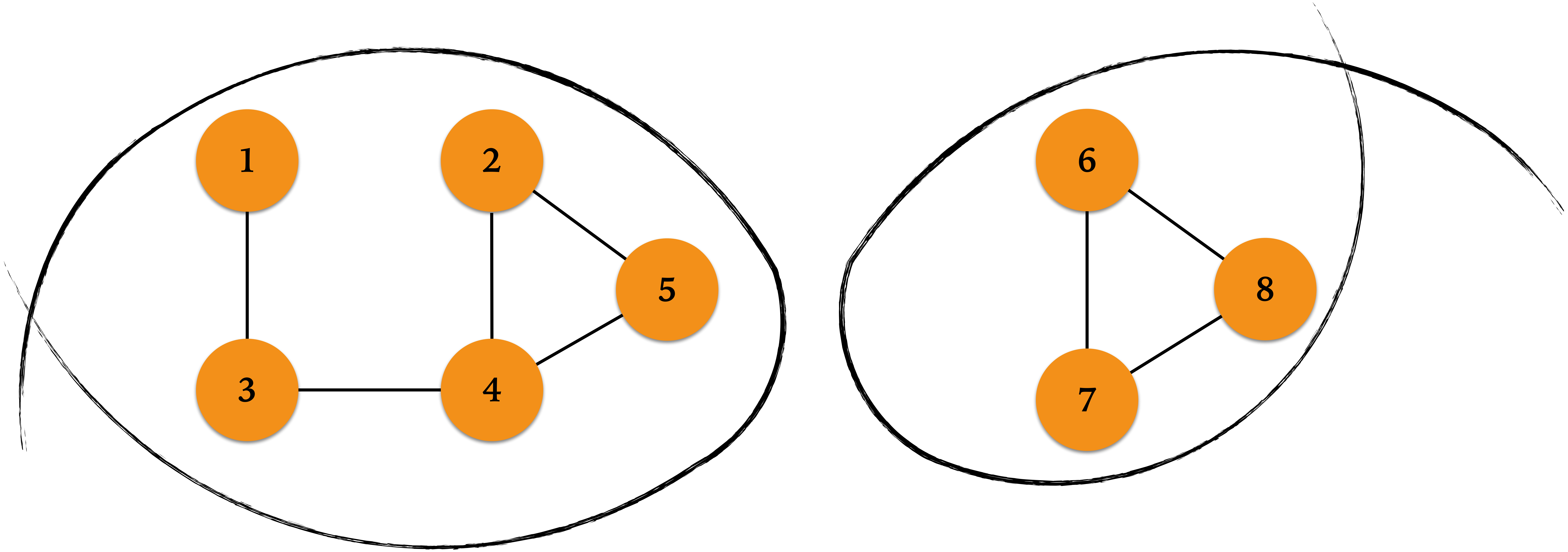
- We express the computation from the view of a single vertex
- Vertices communicate through **messages**
- The computation proceeds in synchronous iteration steps



$(V_{i+1}, \text{outbox}) \leftarrow \text{compute}(V_i, \text{inbox})$

Connected components

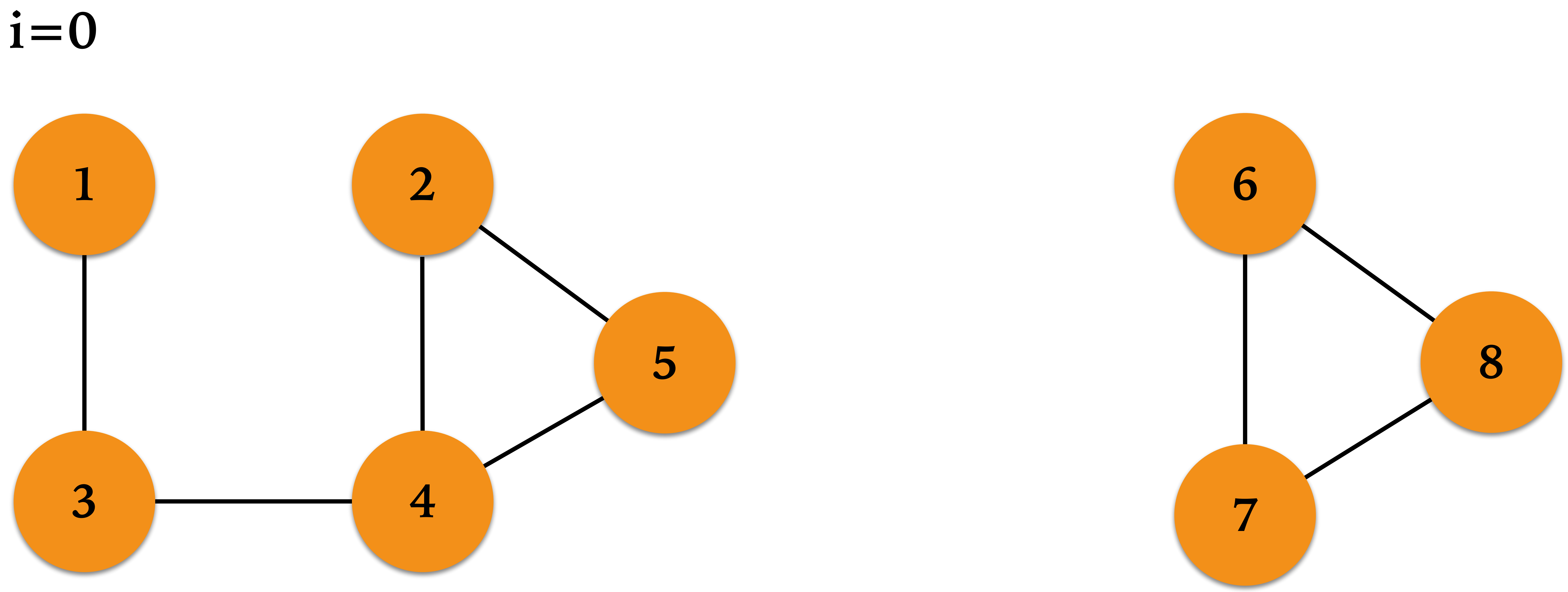
A component is a subgraph in which every vertex is reachable from all other vertices in the subgraph.



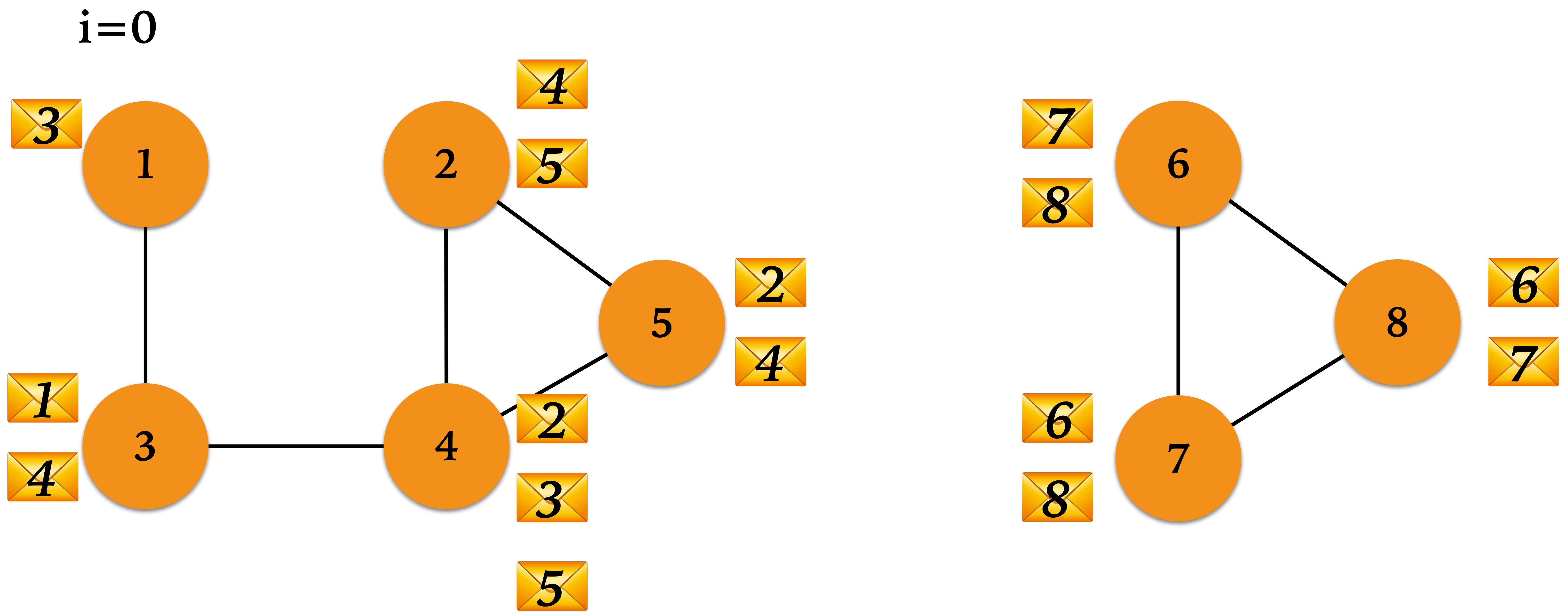
Batch Connected Components

- **State:** the graph and a component ID per vertex
  - initially equal to vertex ID
- **Iterative step:** For each vertex
  - choose the $\min$ of neighbors' component IDs and own component ID as the new ID
  - if the component ID changed since the last iteration, notify neighbors

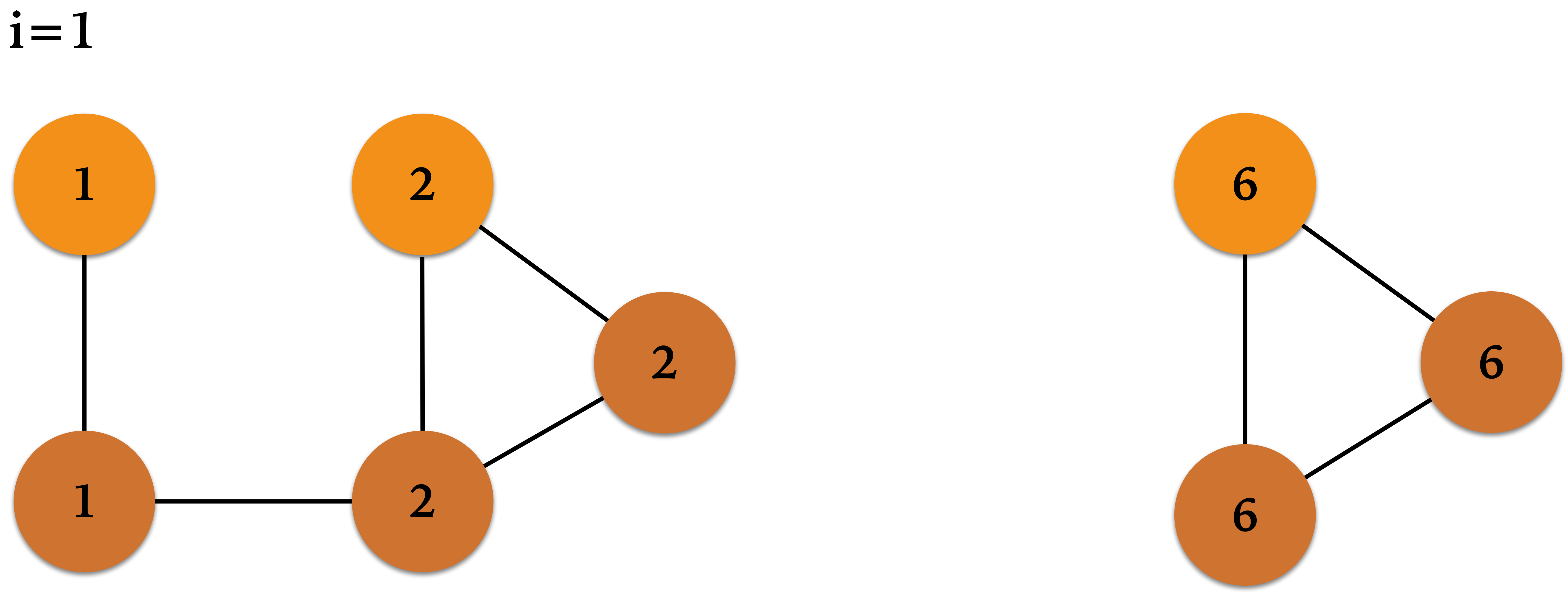
Batch Connected Components



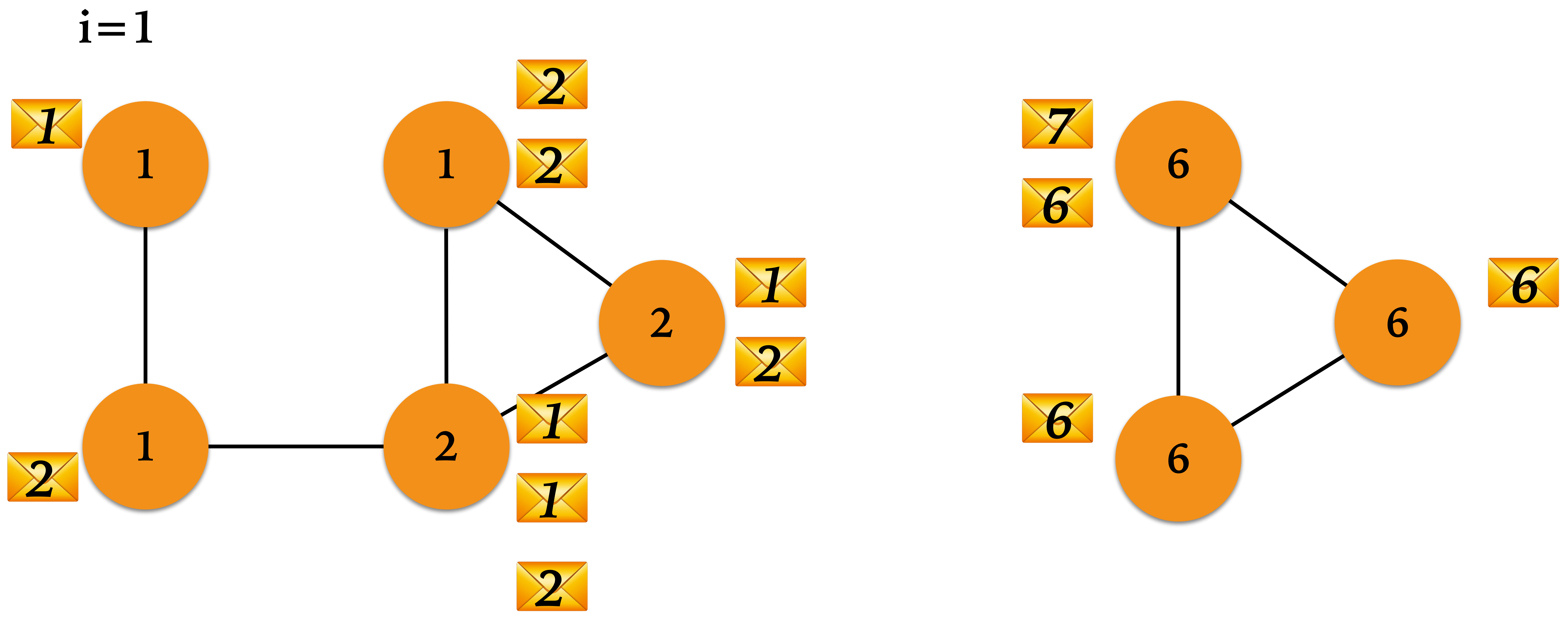
Batch Connected Components



Batch Connected Components

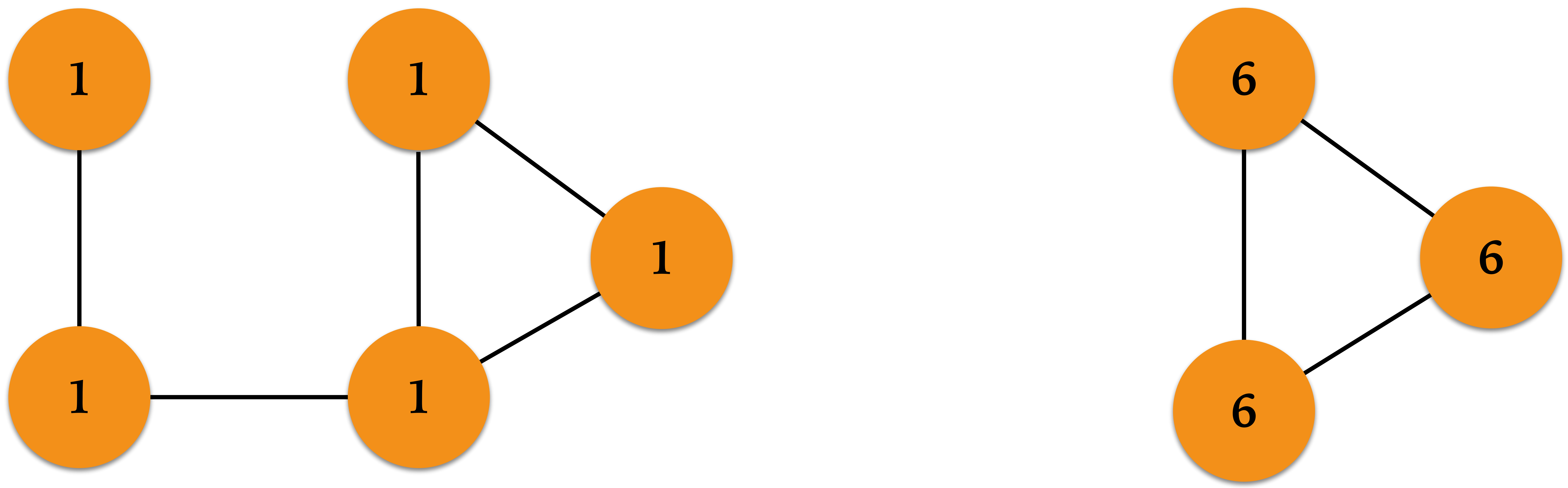


Batch Connected Components



Batch Connected Components

converged



compID = 1

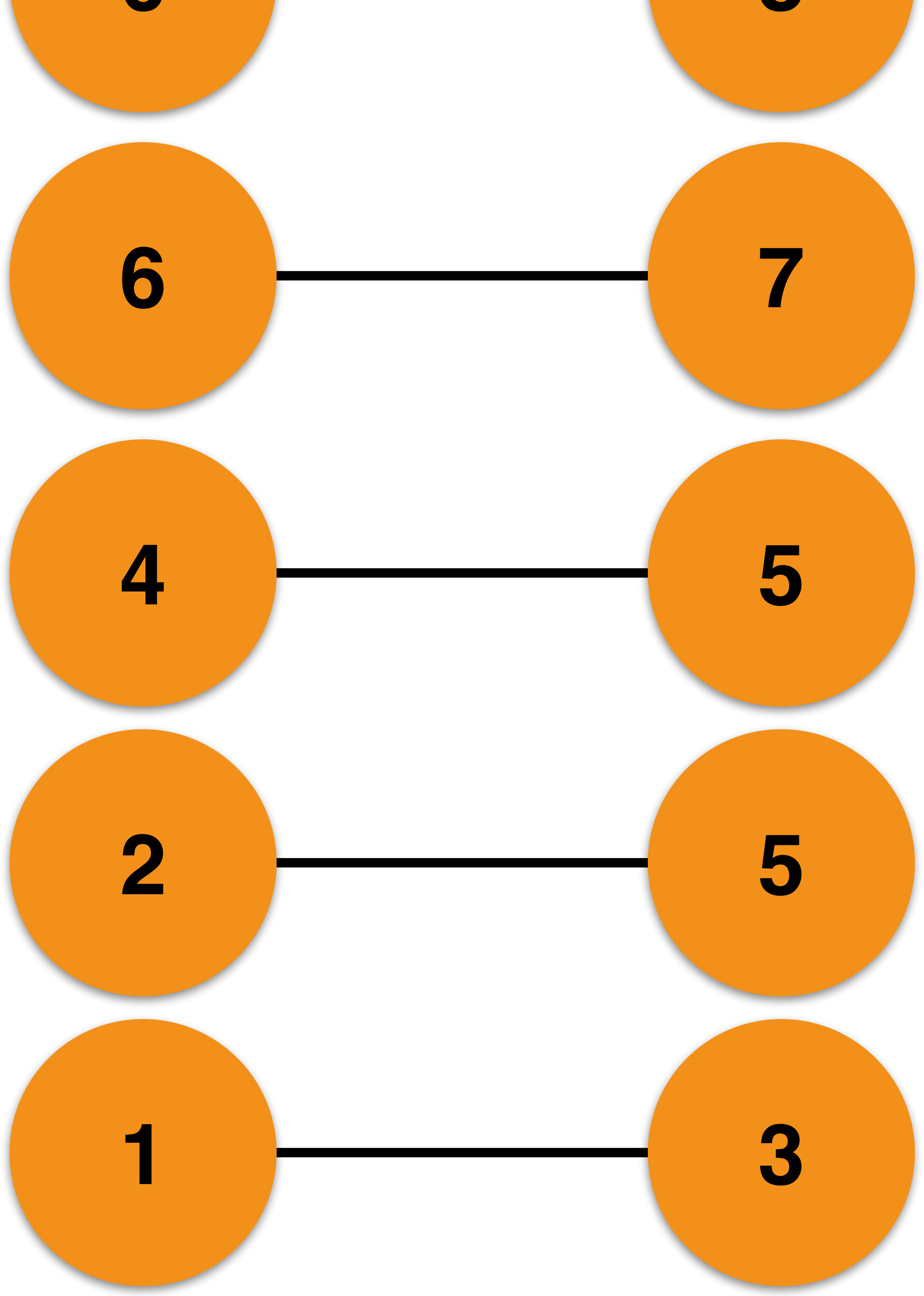
compID = 6

- How can we run such algorithms if the graph is continuously generated as a stream of edges?
- How can we perform iterative computation in a streaming dataflow engine? How can we propagate watermarks?
- Do we need to run the computation from scratch for every new edge?
- Can we use graph synopses and summaries and compute graph analytics in one-pass?

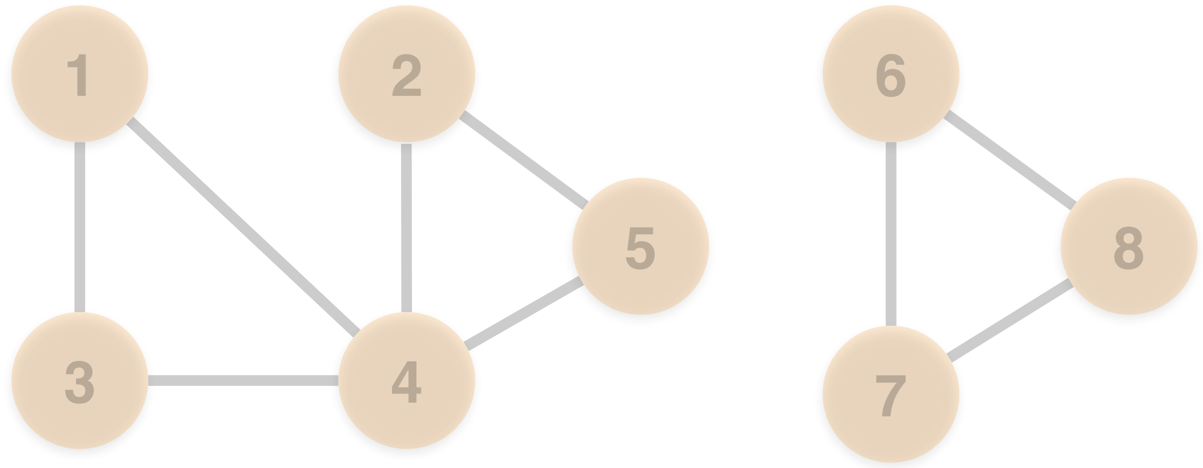
Connectivity & Bipartite property

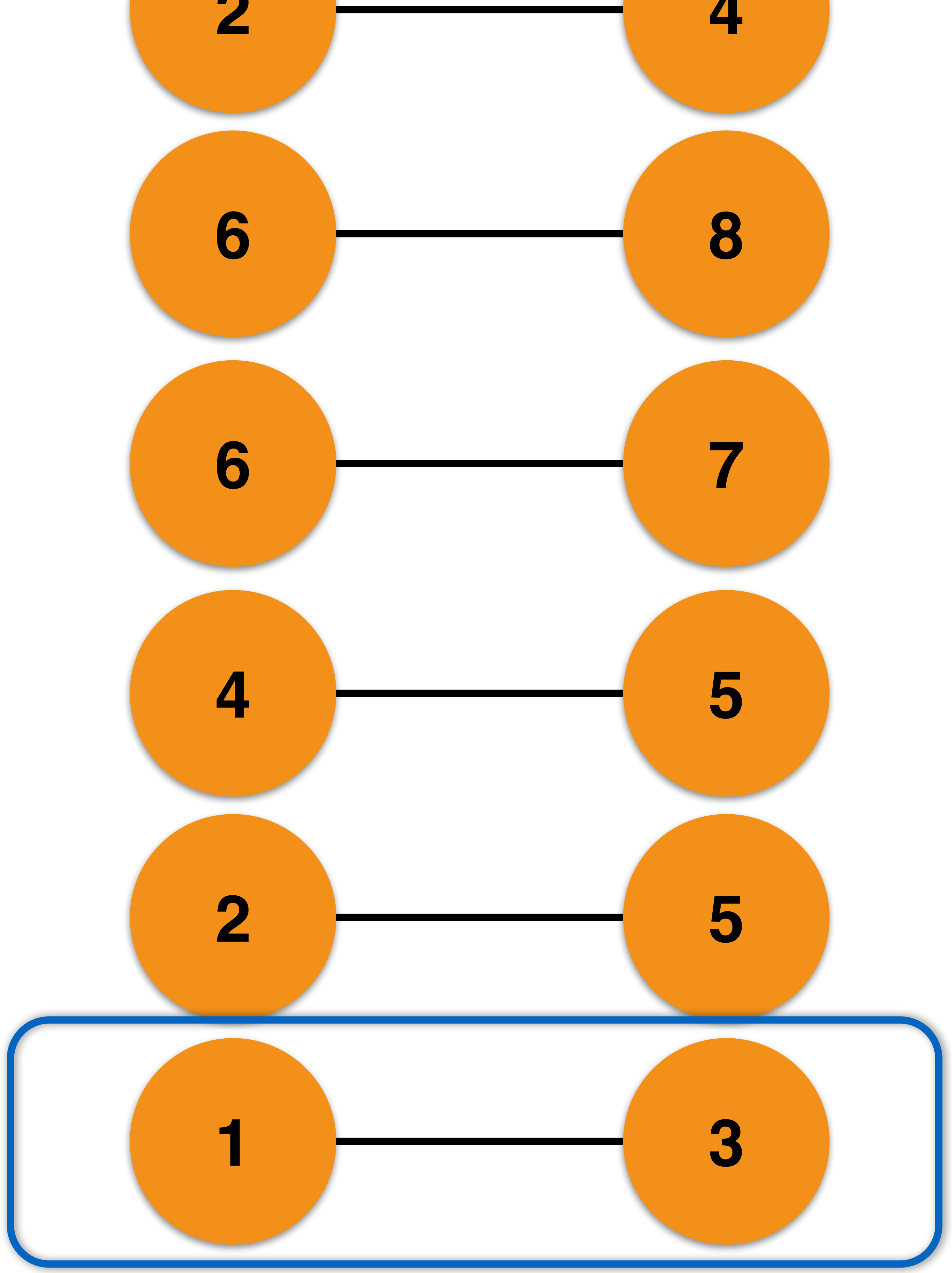
Streaming Connected Components

- **State:** a *disjoint set* (union-find) data structure for the components
  - it stores a set of elements partitioned in disjoint subsets
- **Single-pass computation:** For each edge
  - if seen for the 1st time, create a component with ID the min of the vertex IDs
  - if in different components, *merge* them and update the component ID to the min of the component IDs
  - if only one of the endpoints belongs to a component, add the other one to the same component

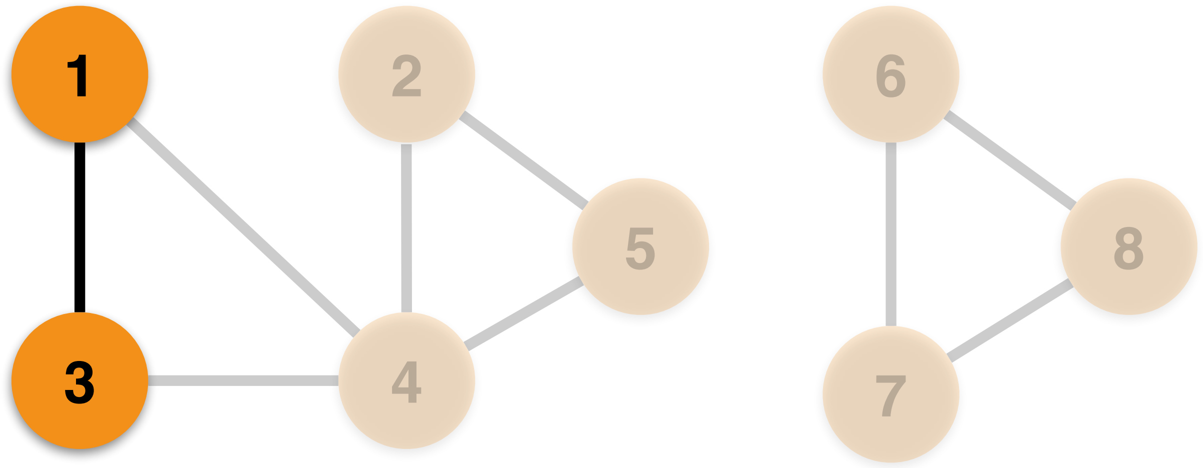


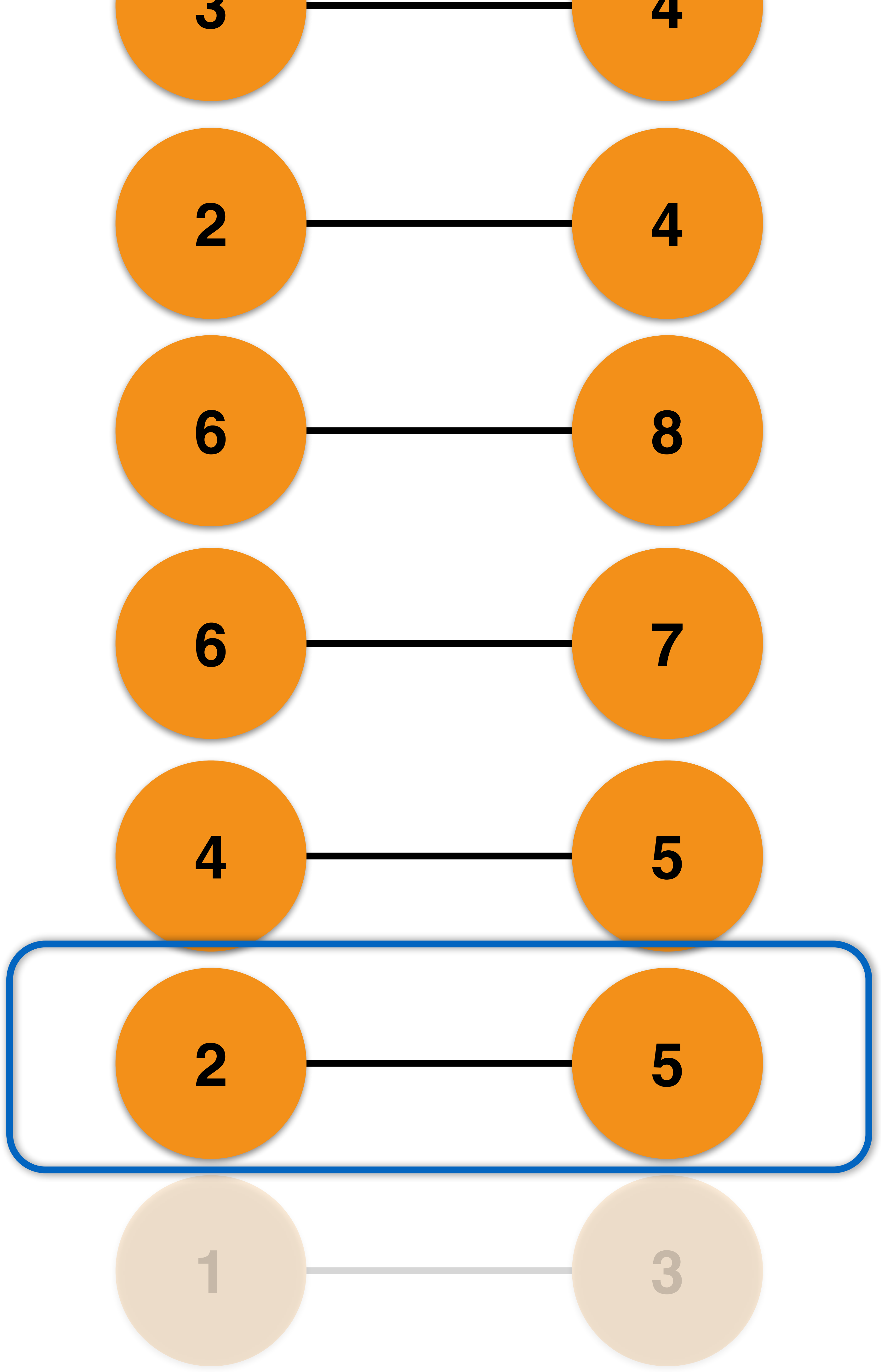
ComponentID	Vertices



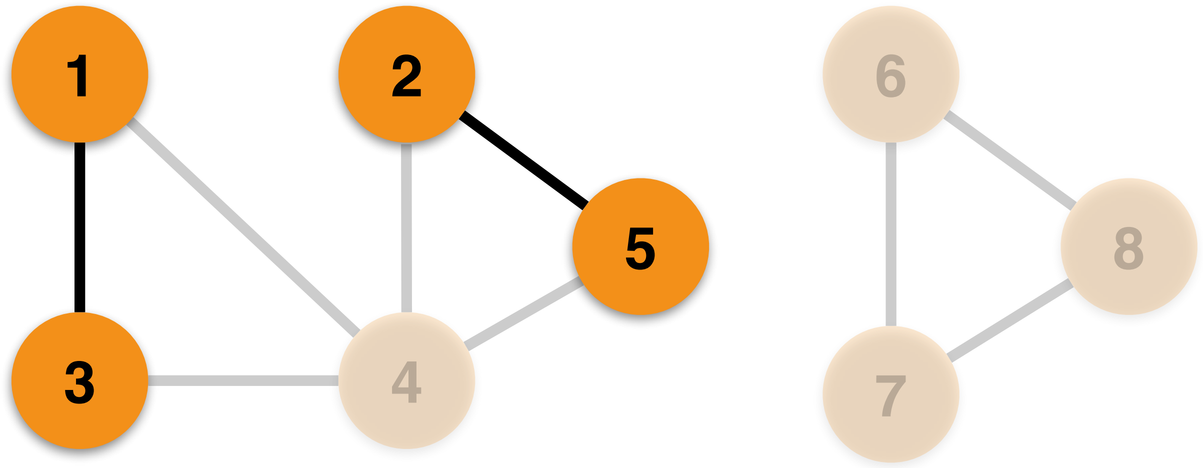


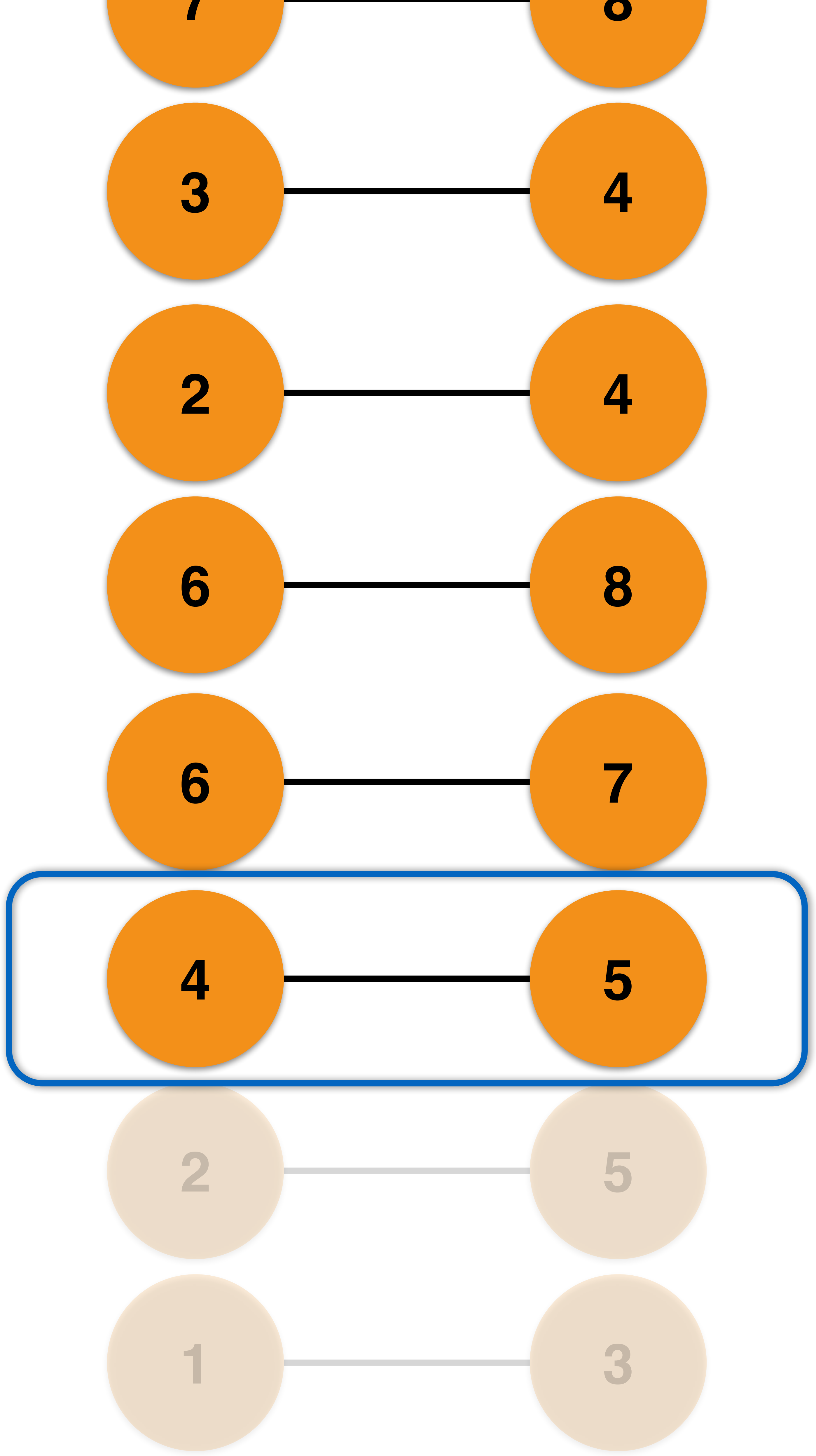
ComponentID	Vertices
1	1, 3



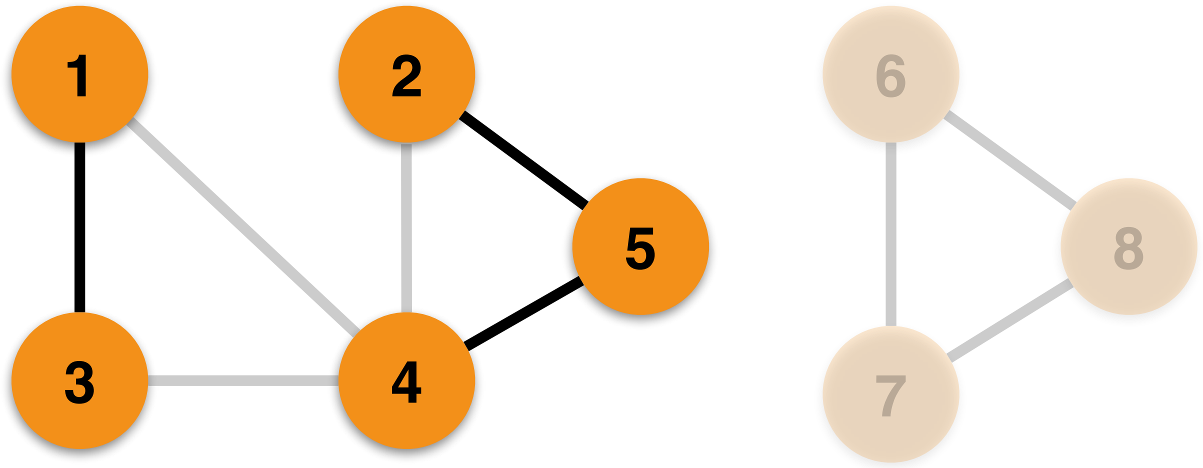


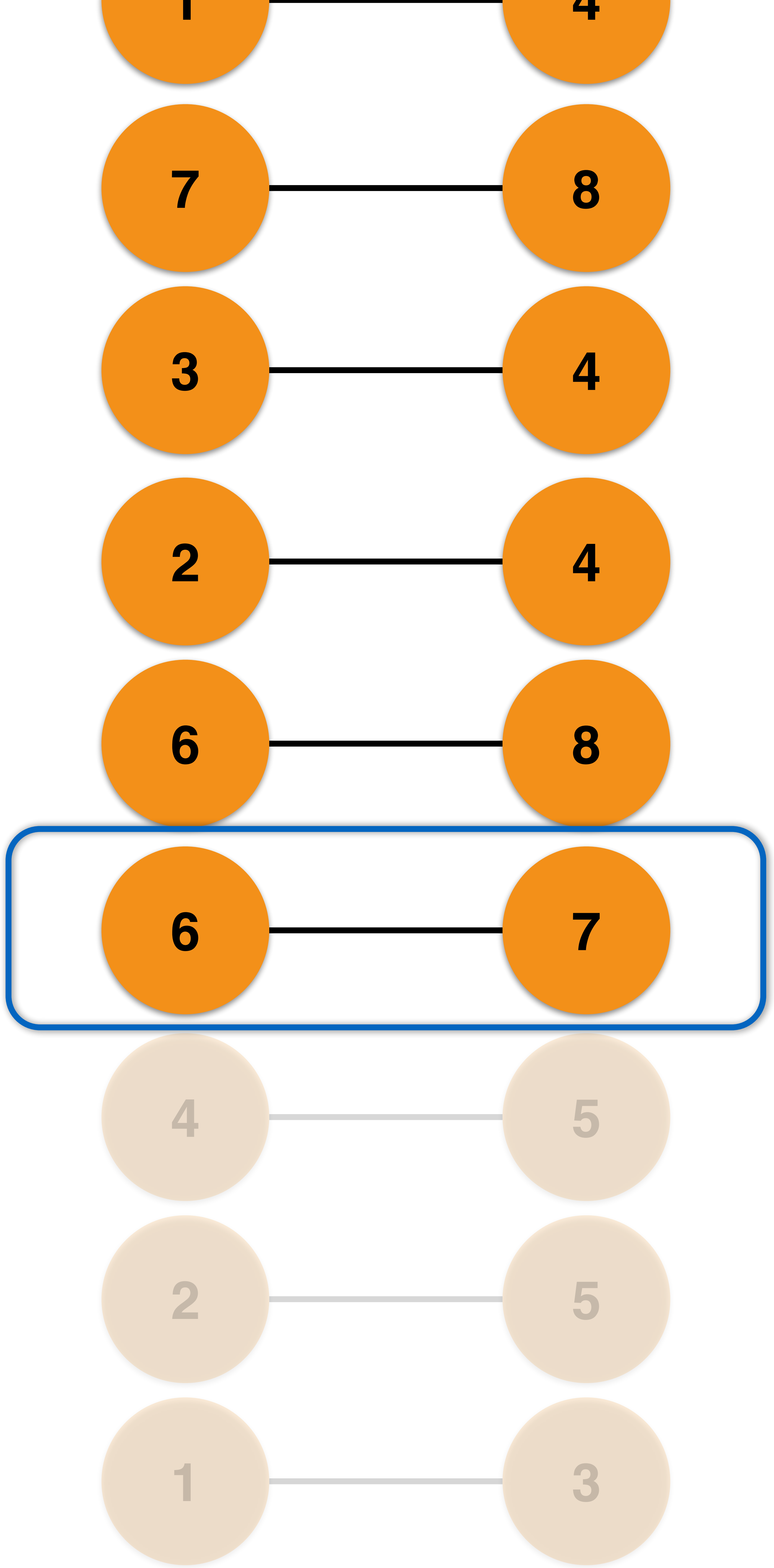
ComponentID	Vertices
1	1, 3
2	2, 5



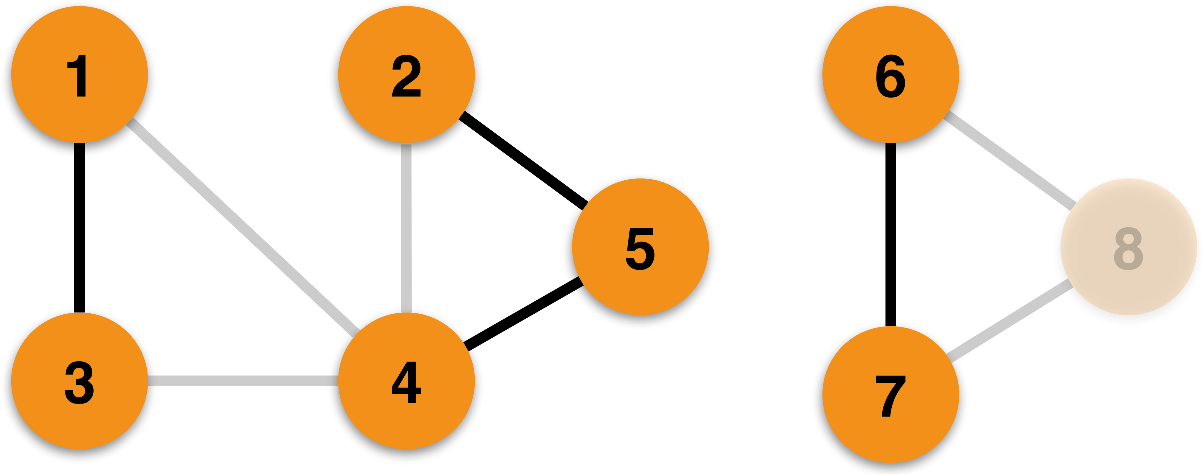


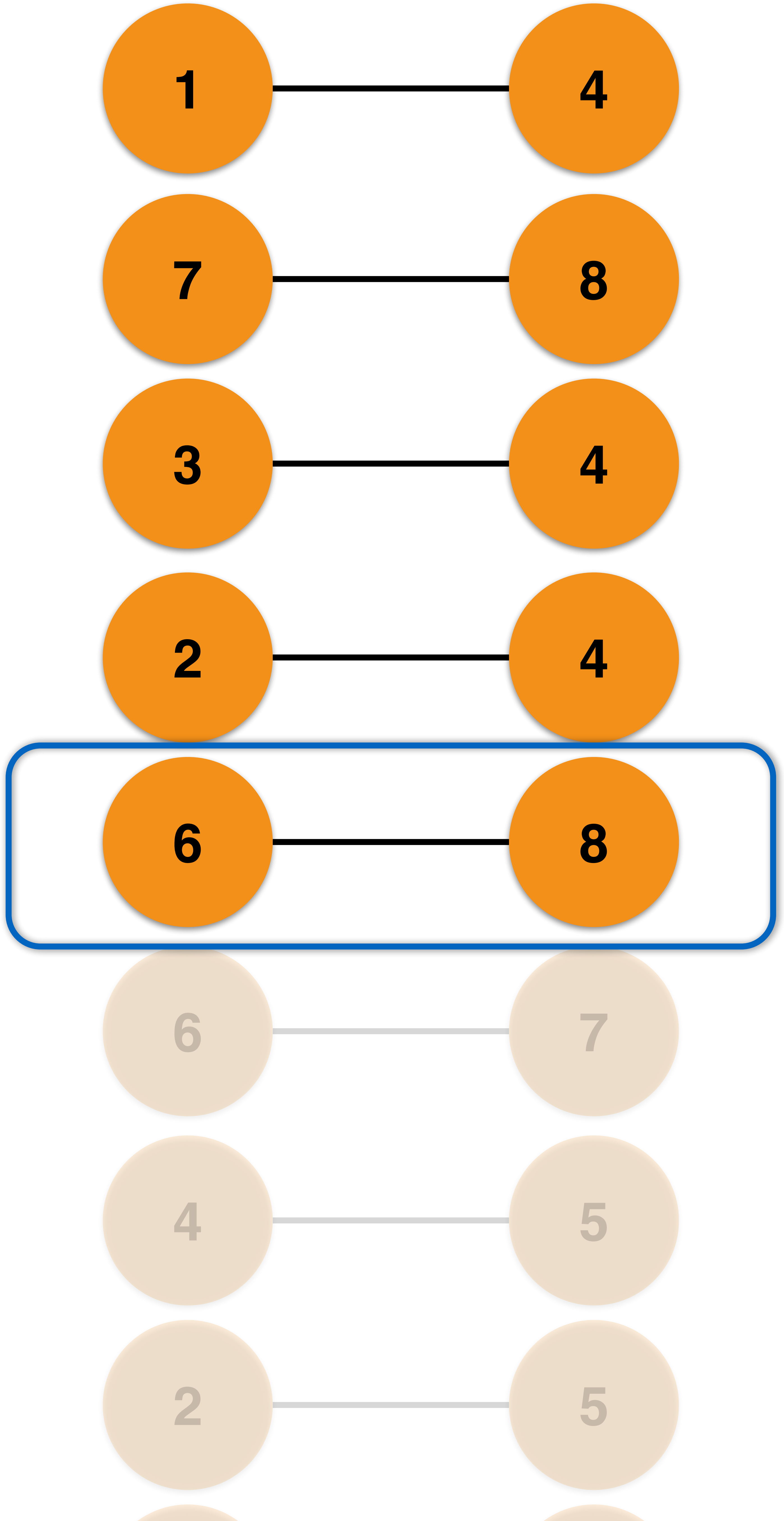
ComponentID	Vertices
1	1, 3
2	2, 4, 5



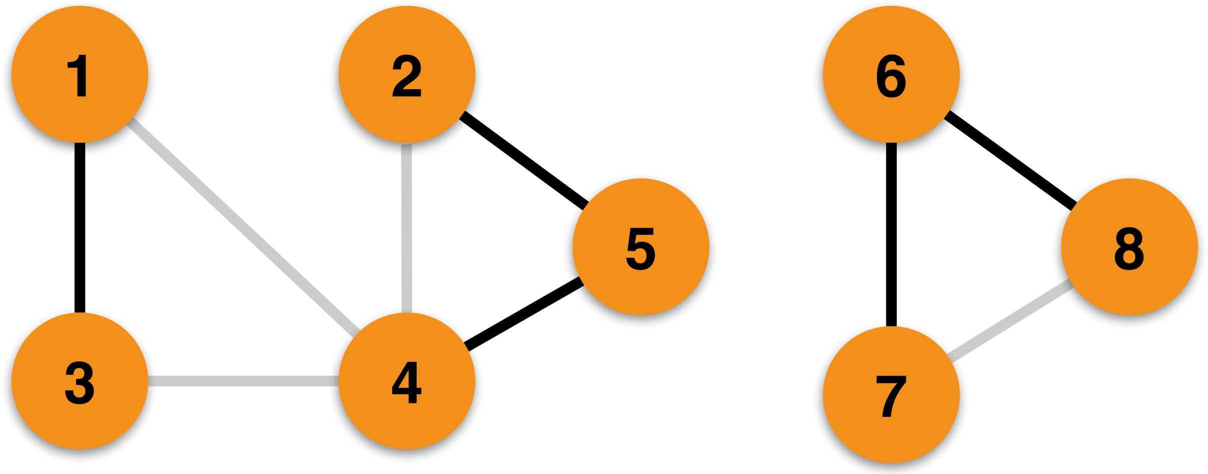


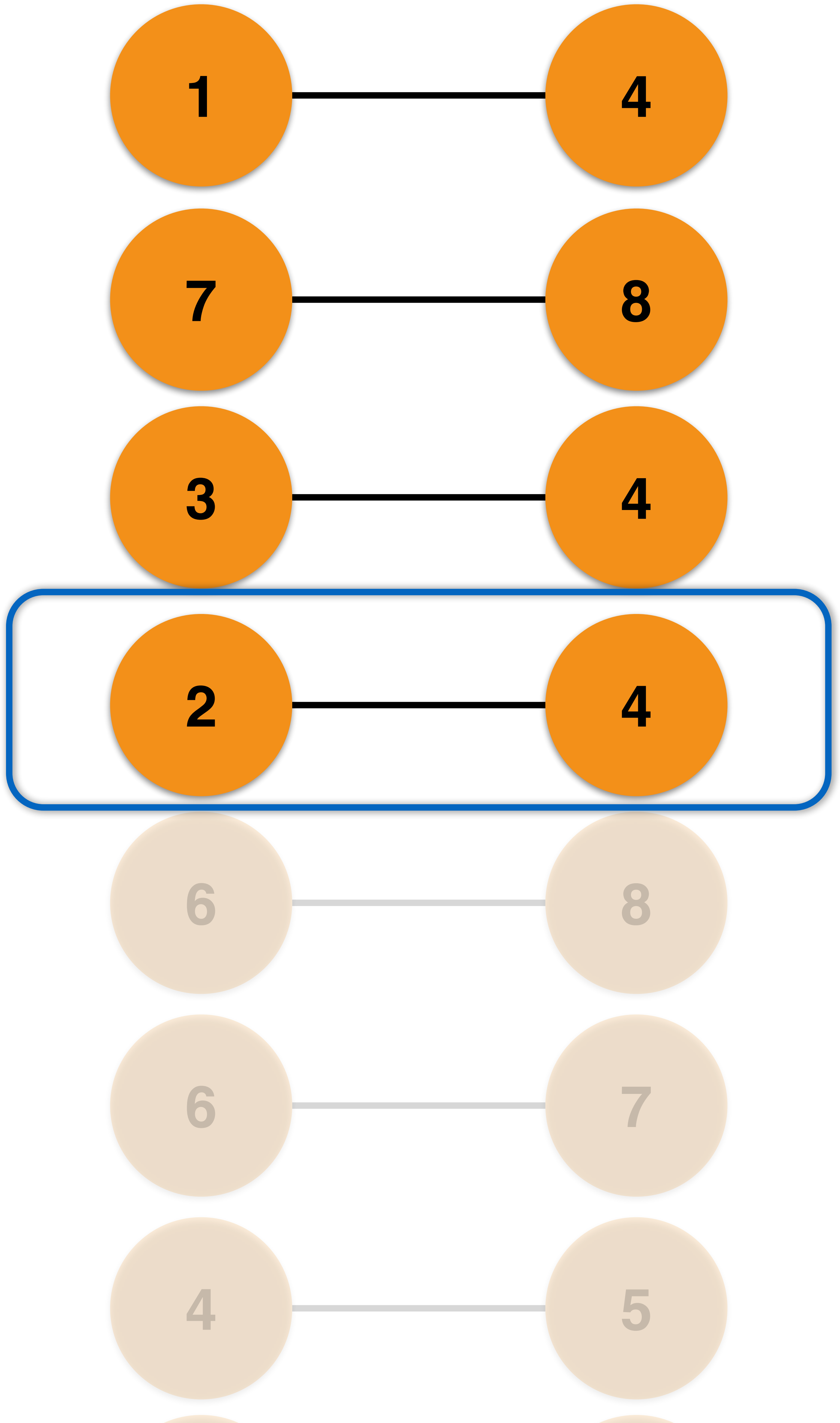
ComponentID	Vertices
1	1, 3
2	2, 4, 5
6	6, 7



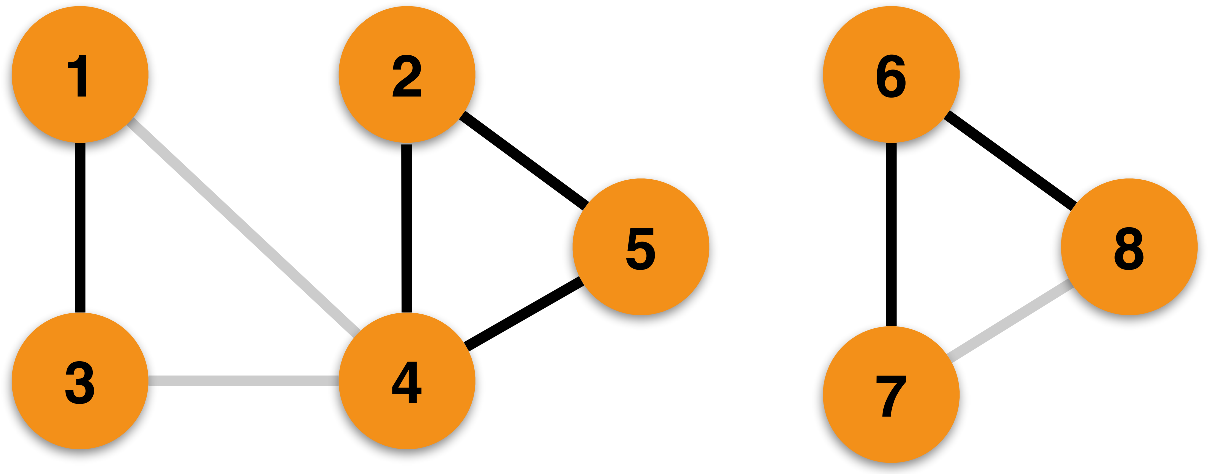


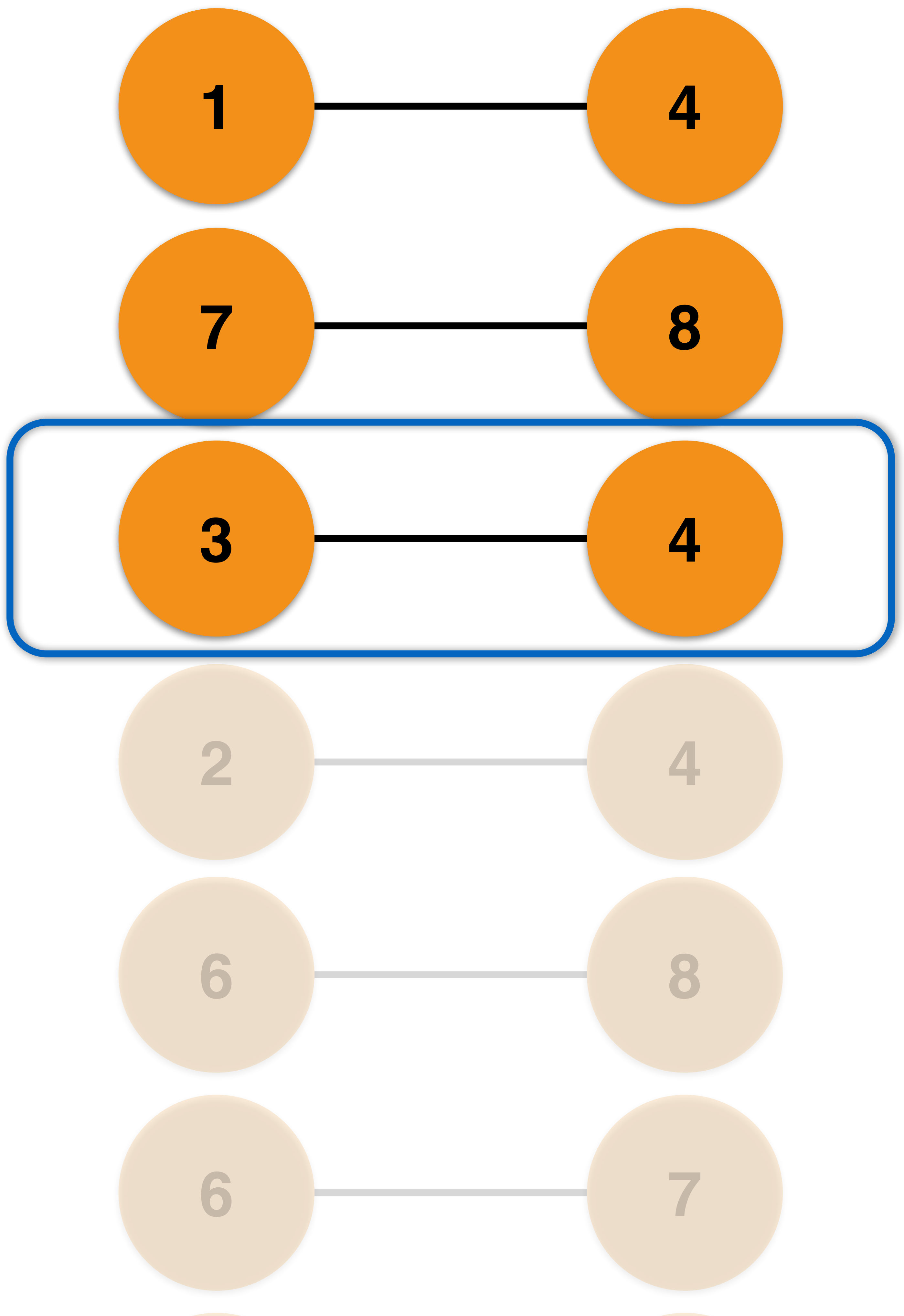
ComponentID	Vertices
1	1, 3
2	2, 4, 5
6	6, 7, 8



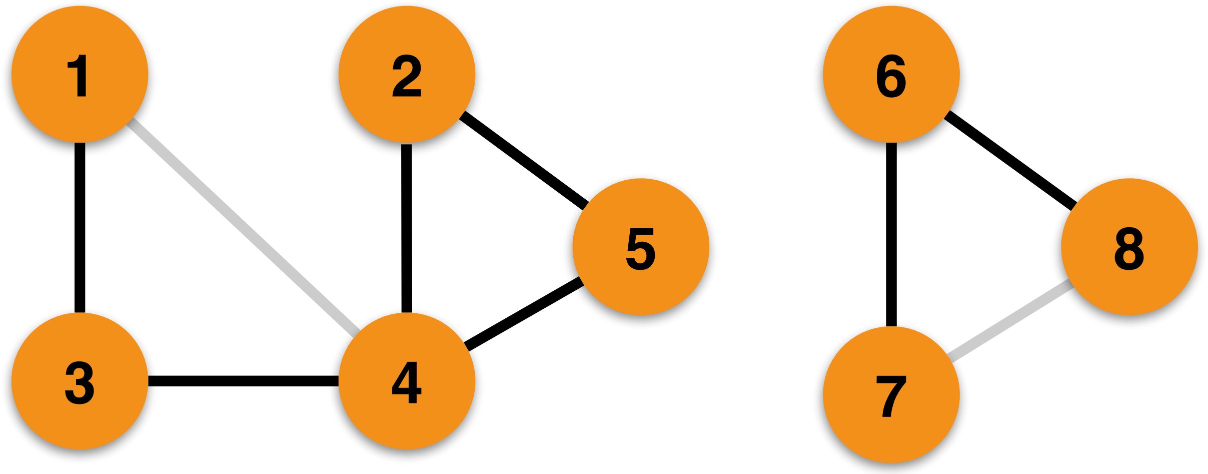


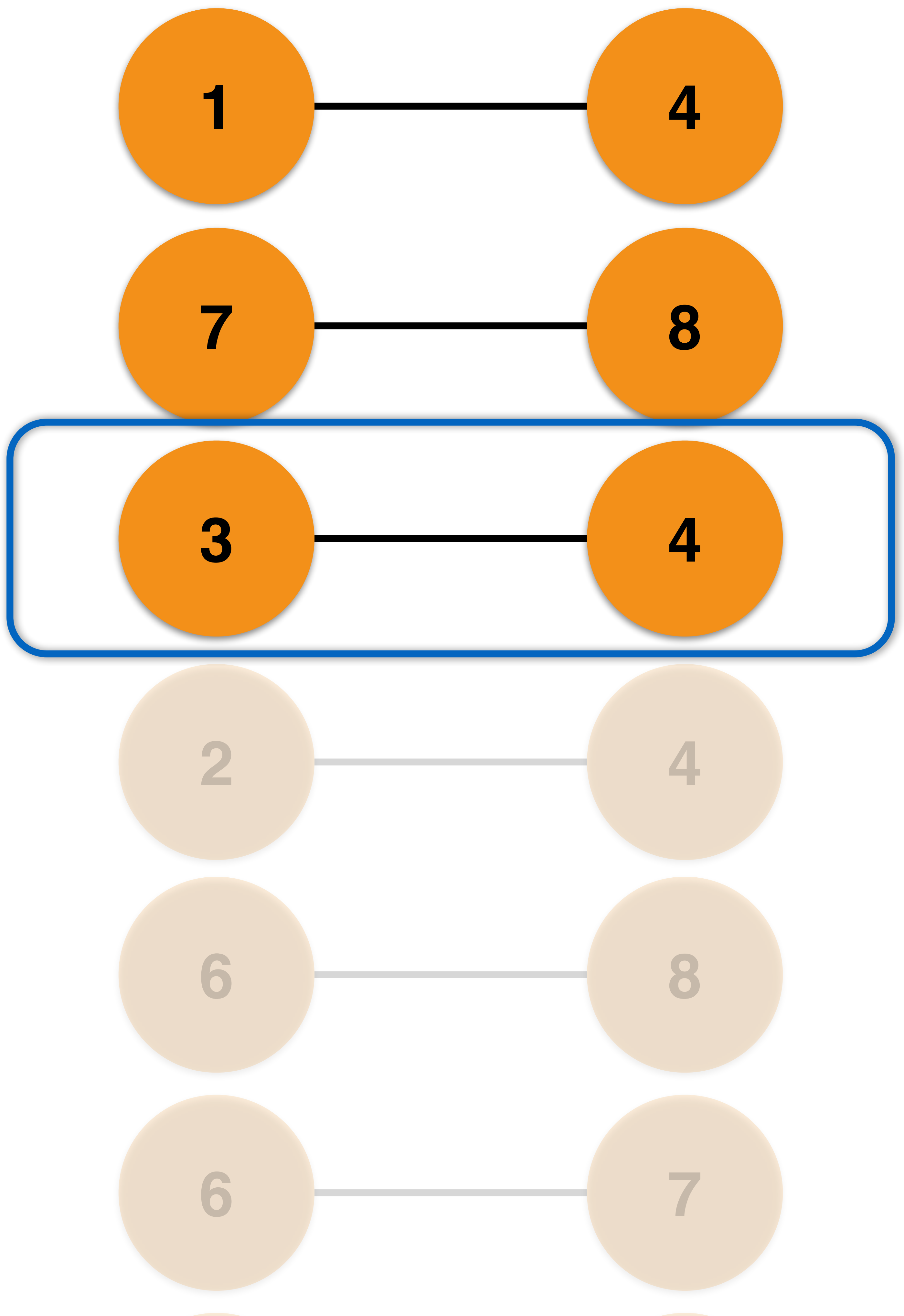
ComponentID	Vertices
1	1, 3
2	2, 4, 5
6	6, 7, 8



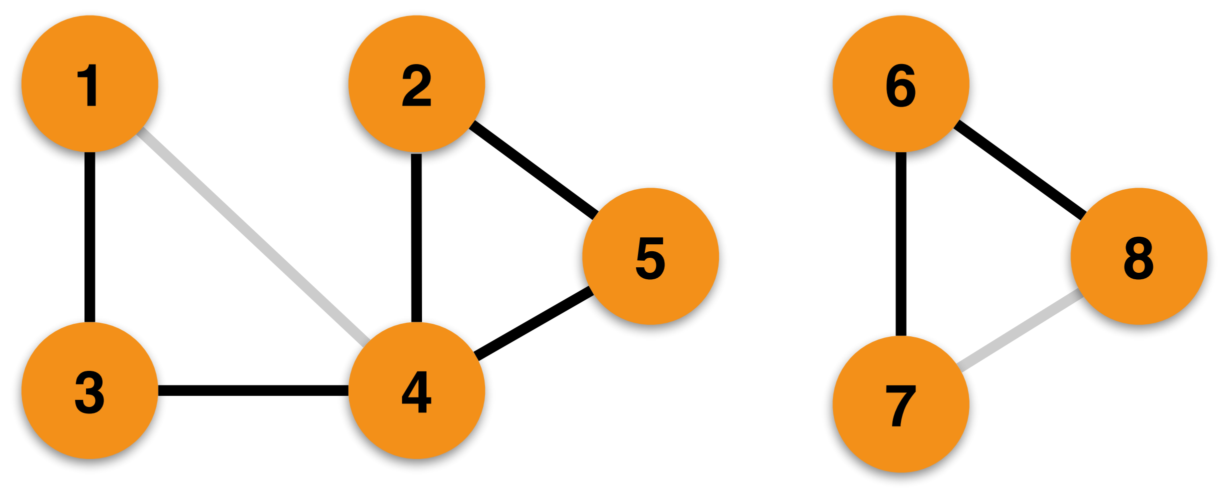


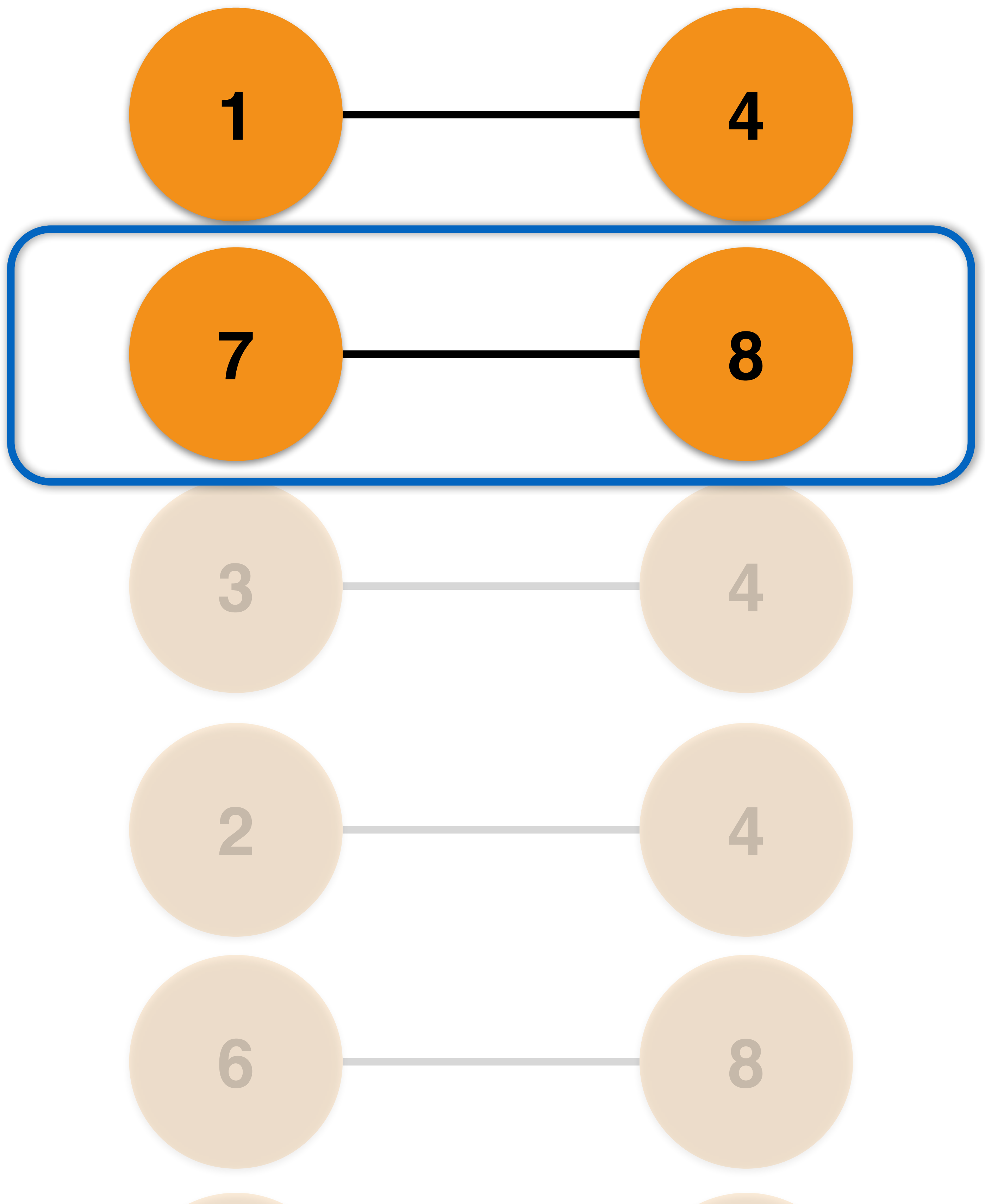
ComponentID	Vertices
1	1, **3**
2	2, **4**, 5
6	6, 7, 8



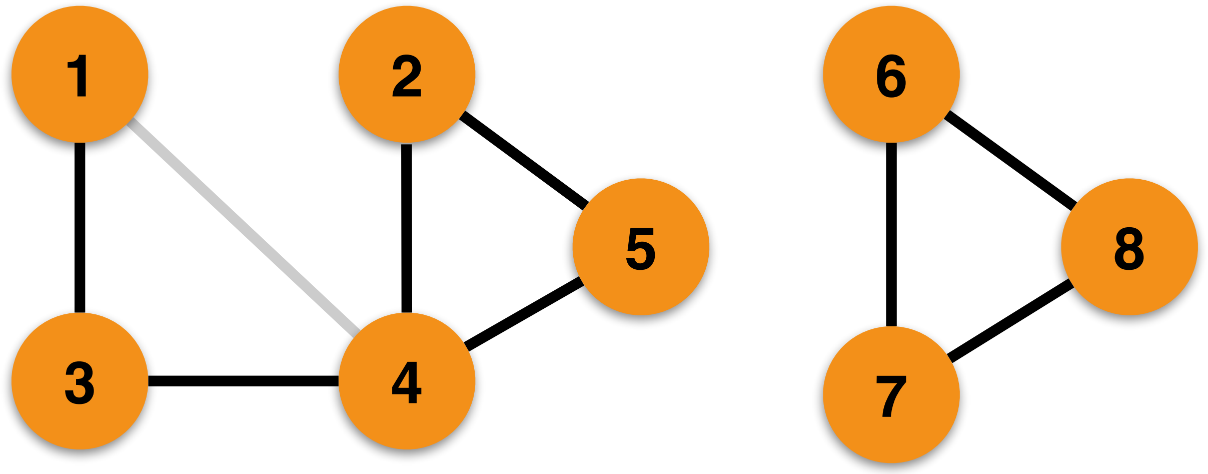


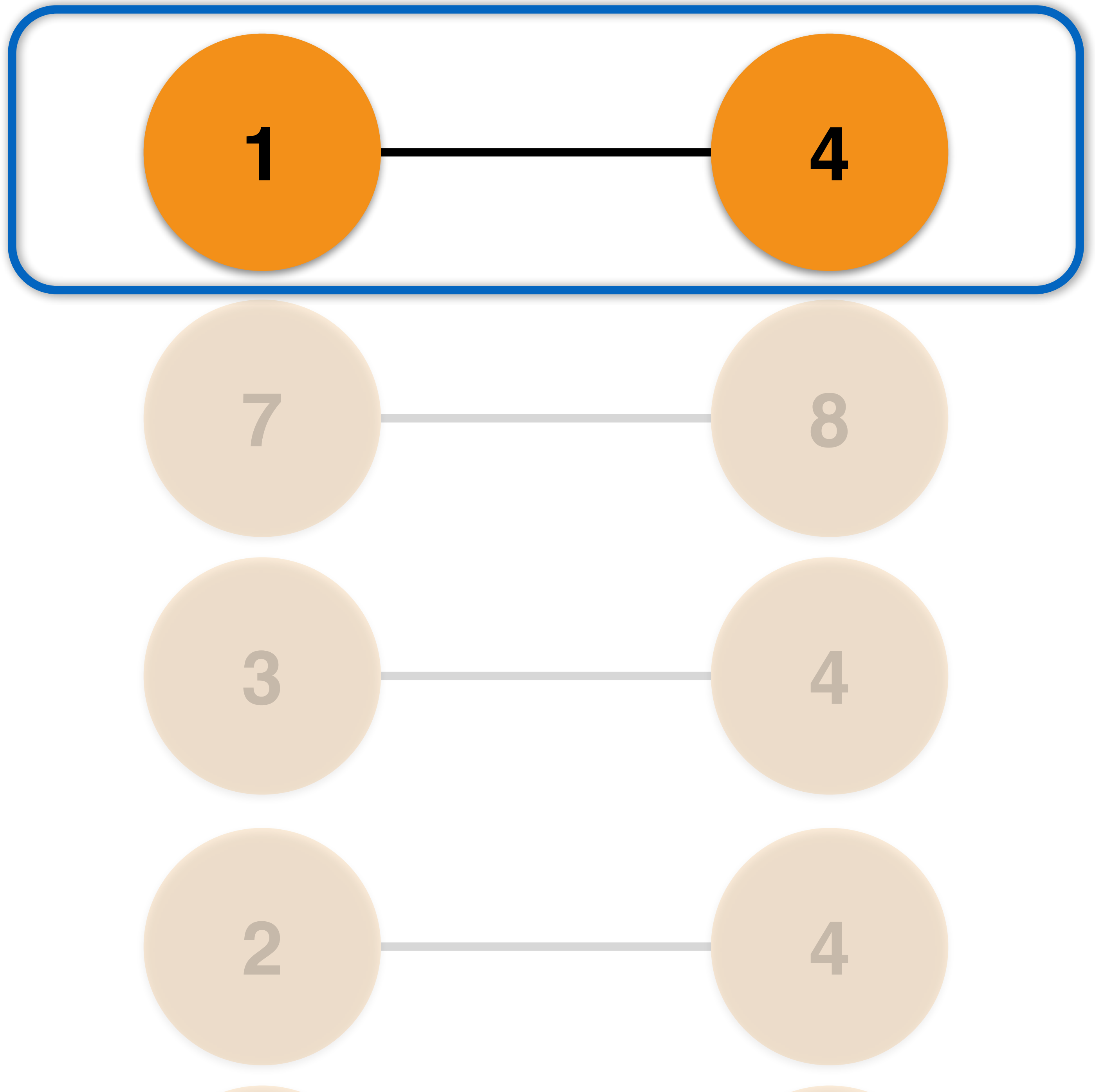
ComponentID	Vertices
1	1, 2, 3, 4, 5
6	6, 7, 8



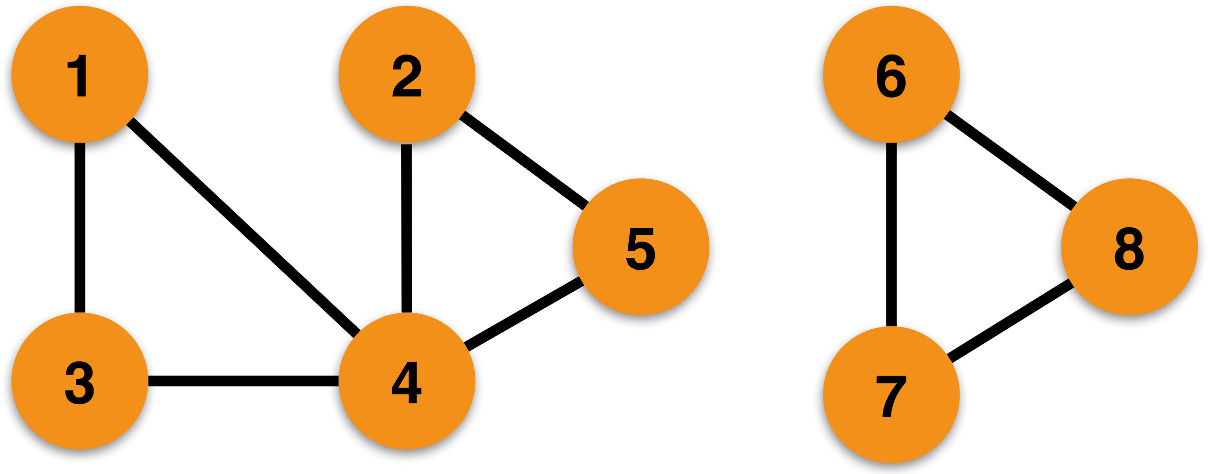


ComponentID	Vertices
1	1, 2, 3, 4, 5
6	6, 7, 8

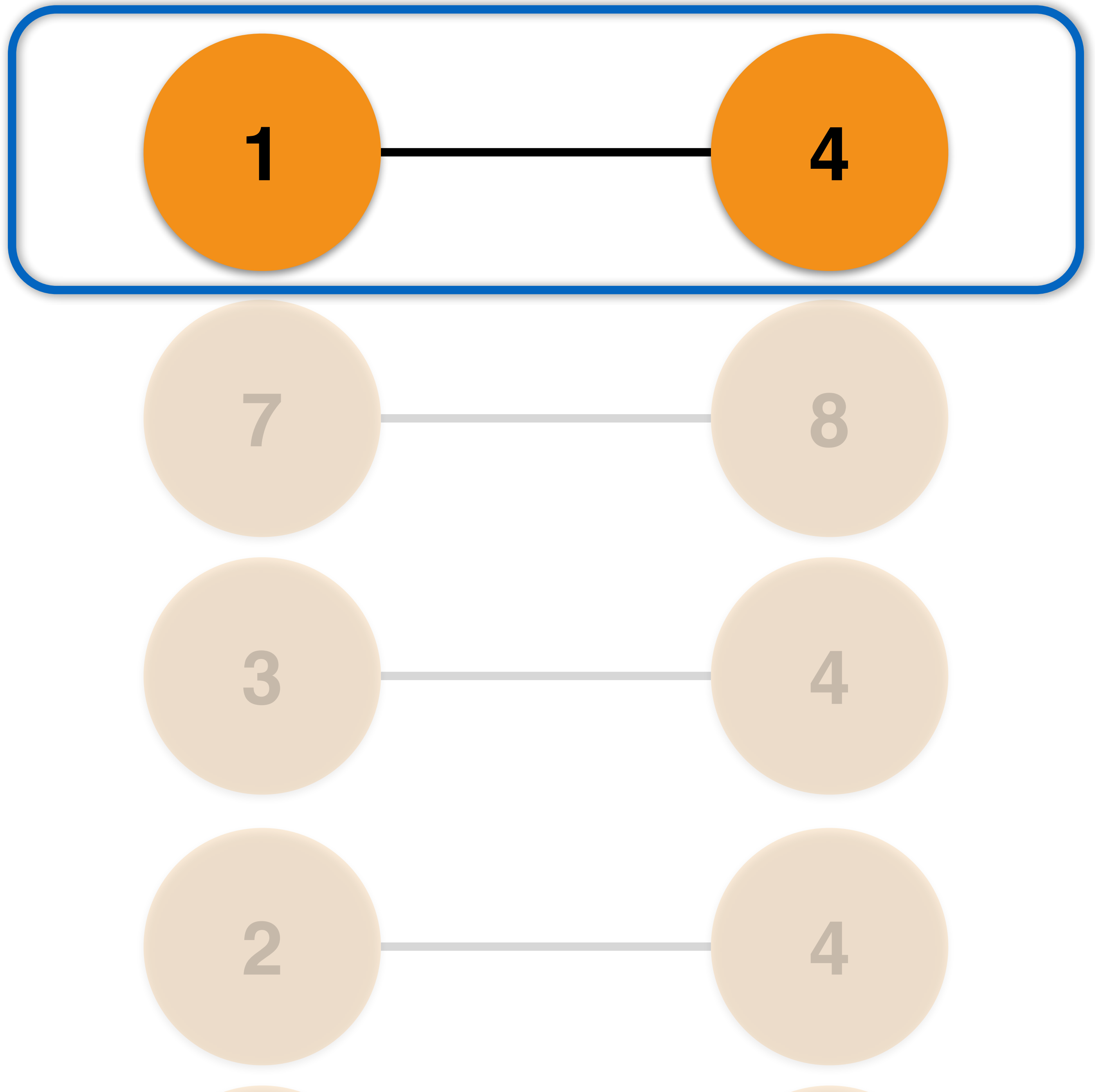




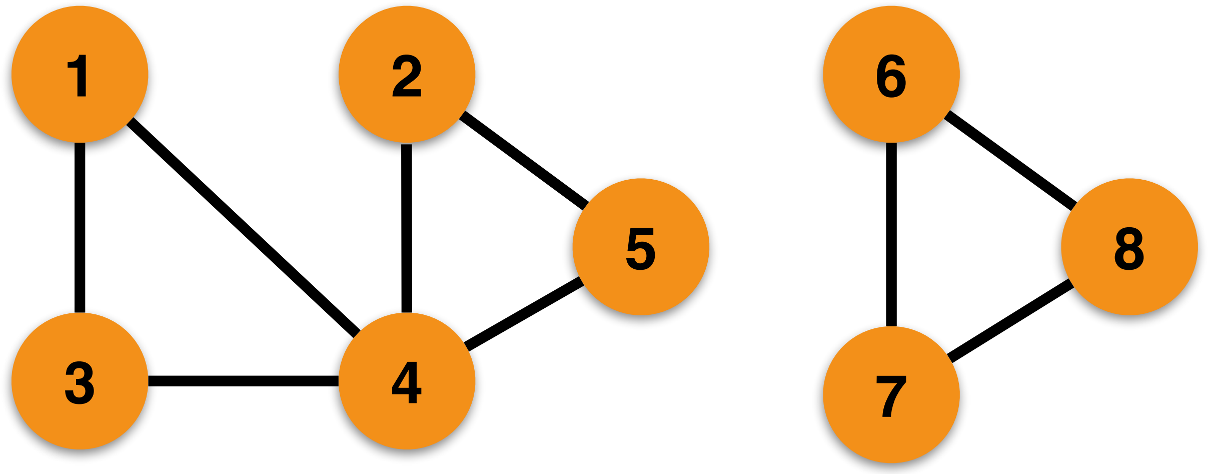
ComponentID	Vertices
1	1, 2, 3, 4, 5
6	6, 7, 8



How would you implement this in Flink?

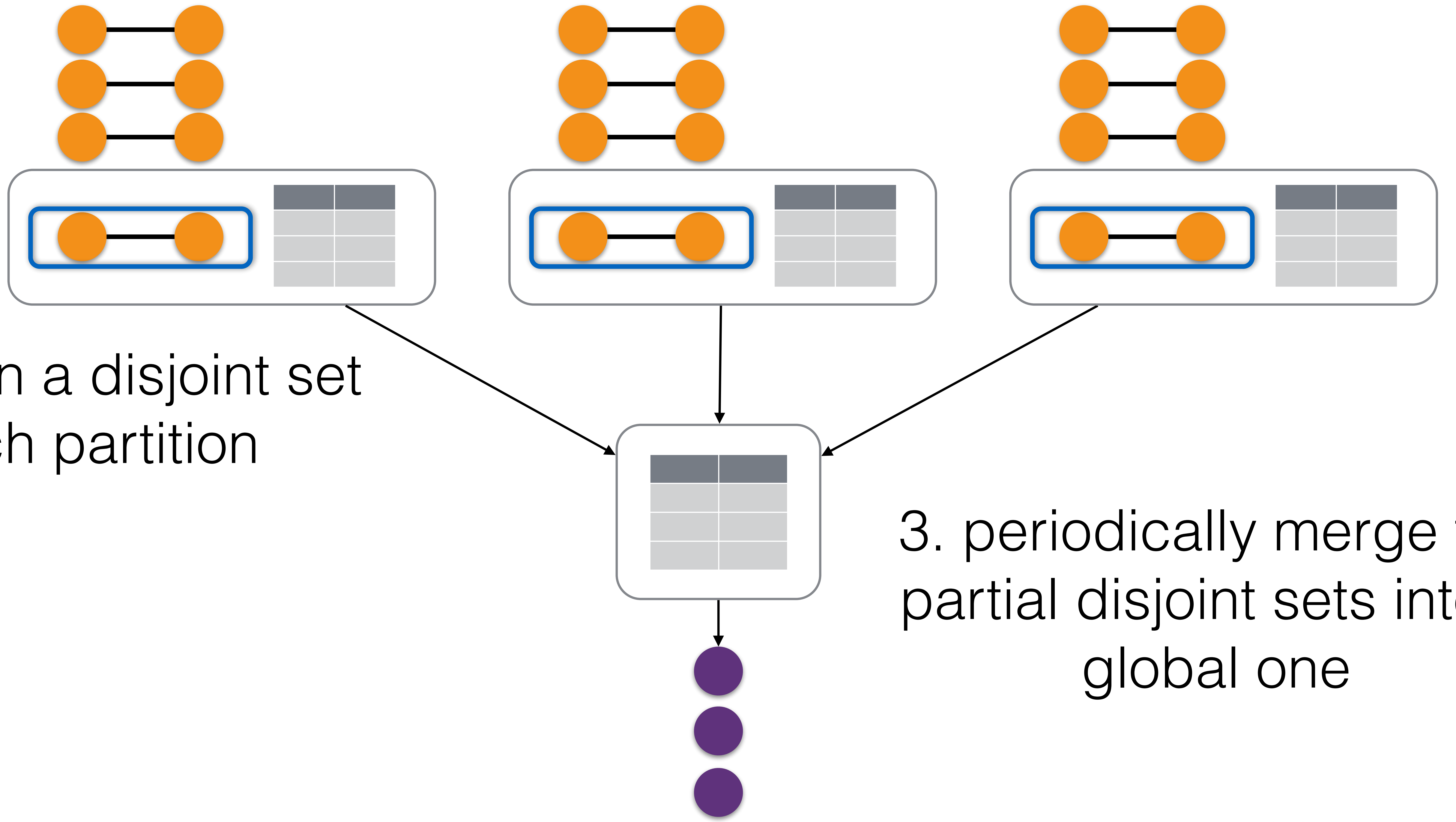


ComponentID	Vertices
1	1, 2, 3, 4, 5
6	6, 7, 8



Distributed Stream Connected Components

1. partition the edge stream, e.g. by source Id



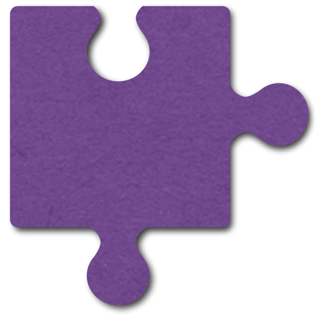
2. maintain a disjoint set in each partition

3. periodically merge the partial disjoint sets into a global one

Connected components in Flink

```
DataStream<DisjointSet> cc =
  edgeStream
     .keyBy(0)
     .timeWindow(Time.of(100, TimeUnit.MILLISECONDS))
     .process(new UpdateDisjointSet()) // ephemeral partial state
     .flatMap(new Merger()) // global state
     .setParallelism(1); // merging on one task
```

Connected components in Flink



Will this scale?

```
DataStream<DisjointSet> cc =
  edgeStream
     .keyBy(0)
     .timeWindow(Time.of(100, TimeUnit.MILLISECONDS))
     .process(new UpdateDisjointSet()) // ephemeral partial state
     .flatMap(new Merger()) // global state
     .setParallelism(1); // merging on one task
```

Connected components in Flink

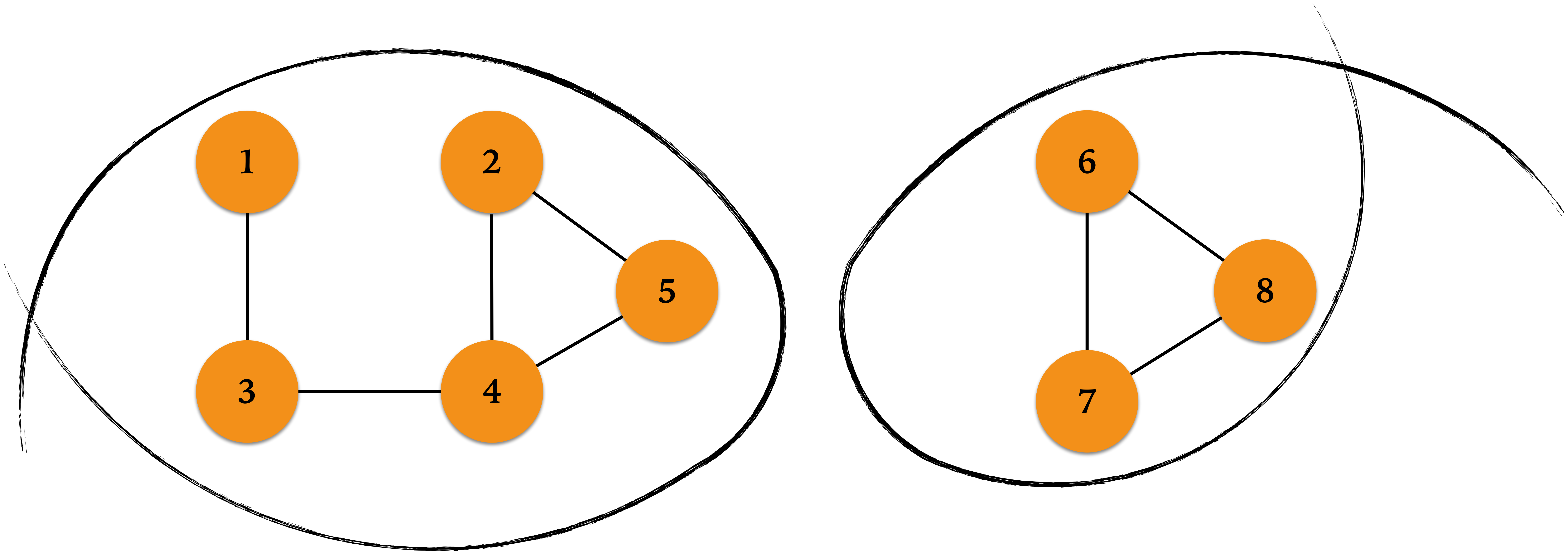
Will this scale?

How to represent the state?

```
DataStream<DisjointSet> cc =
  edgeStream
     .keyBy(0)
     .timeWindow(Time.of(100, TimeUnit.MILLISECONDS))
     .process(new UpdateDisjointSet()) // ephemeral partial state
     .flatMap(new Merger()) // global state
     .setParallelism(1); // merging on one task
```

Bipartite graph checking

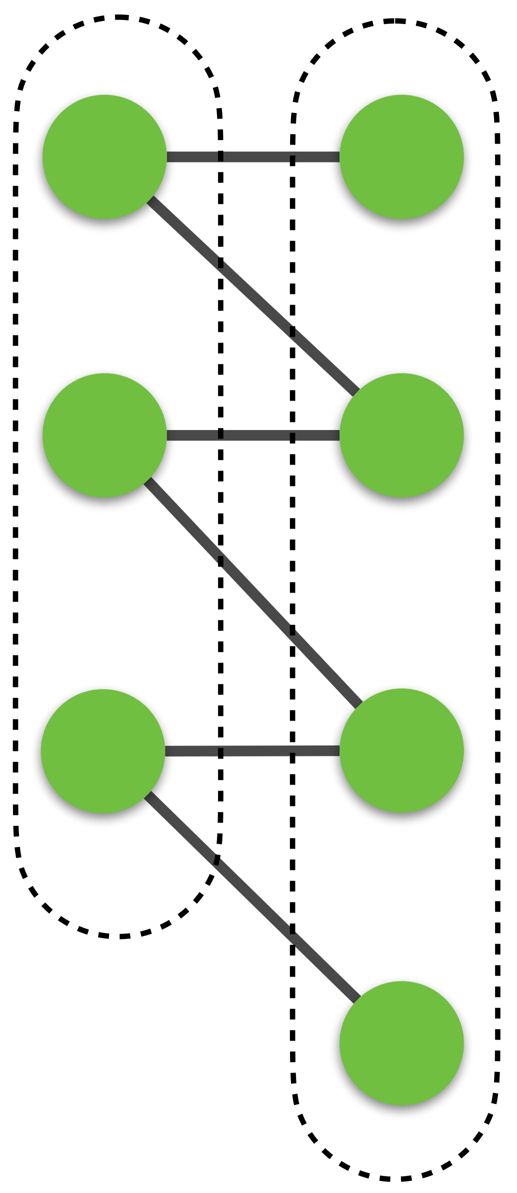
A component is a subgraph in which every vertex is reachable from all other vertices in the subgraph.



Bipartite graph checking

A graph is bipartite if its vertex set can be divided into two disjoint independent sets $\mathbf{U}$, $\mathbf{V}$, such that every edge connects a vertex in $\mathbf{U}$ to a vertex in $\mathbf{V}$ (no edges between vertices in the same part).

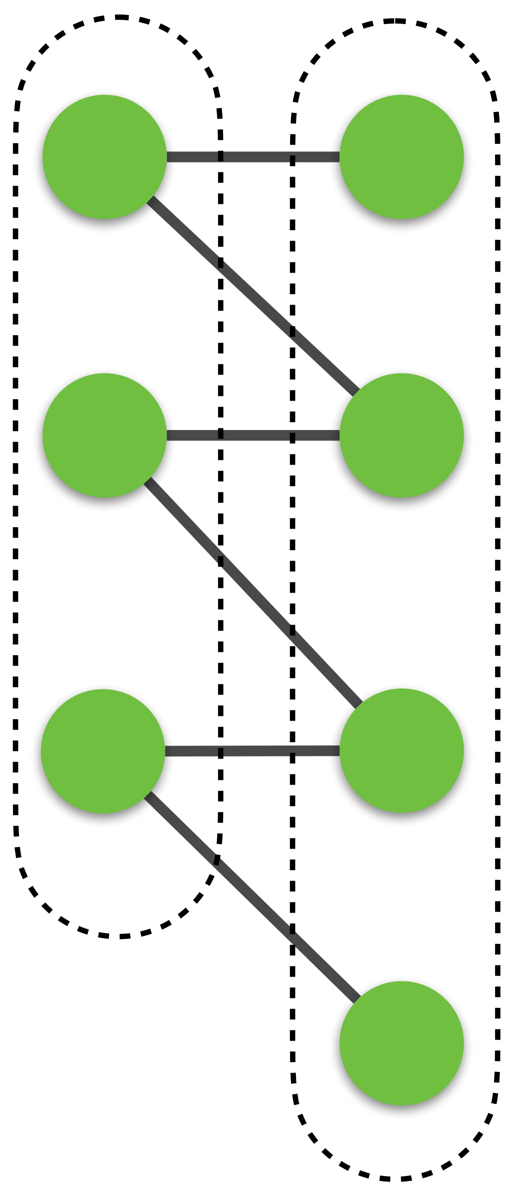
A bipartite graph has no odd-length cycles (thus, no triangles).



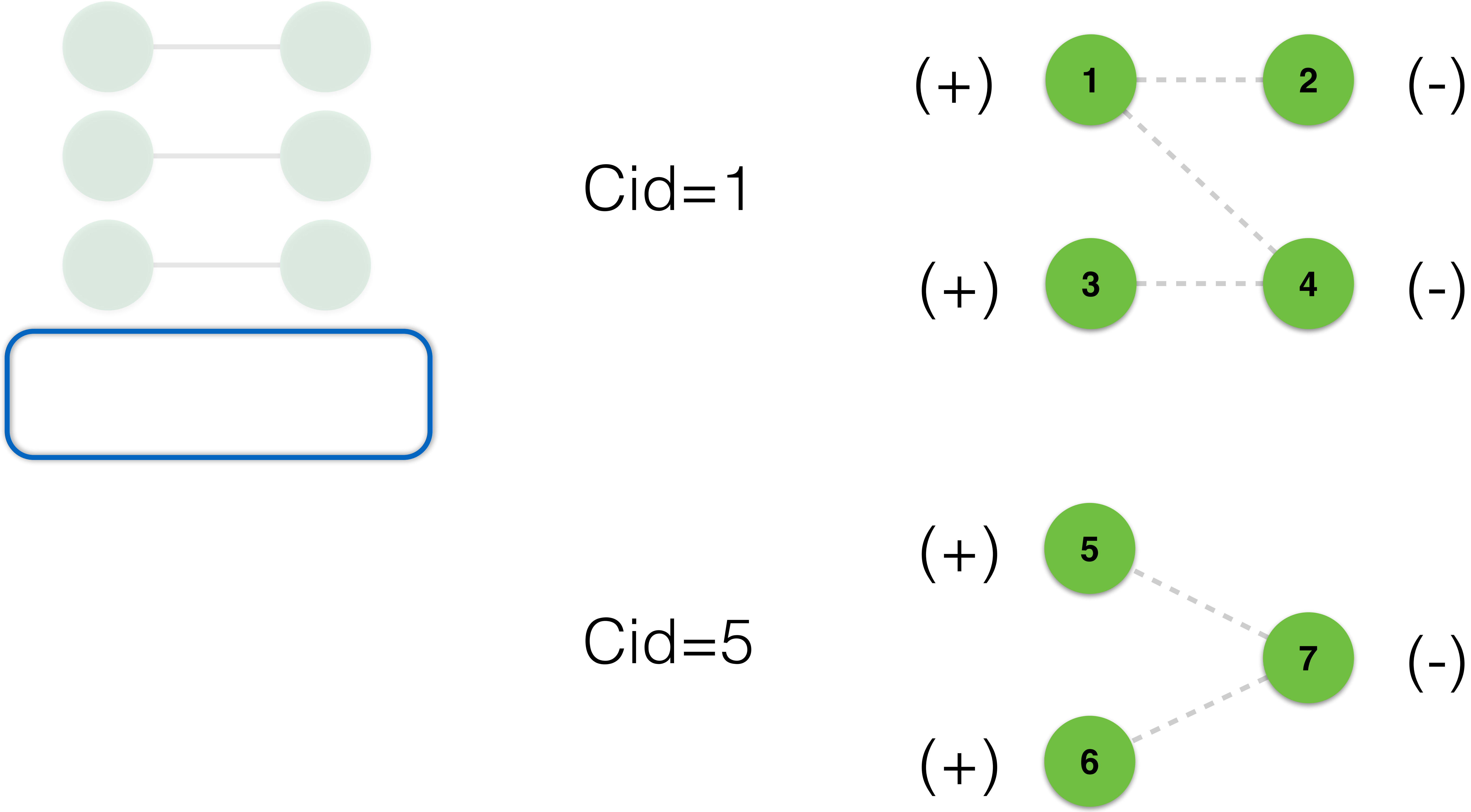
Bipartite graph checking

Similar to connected components, but

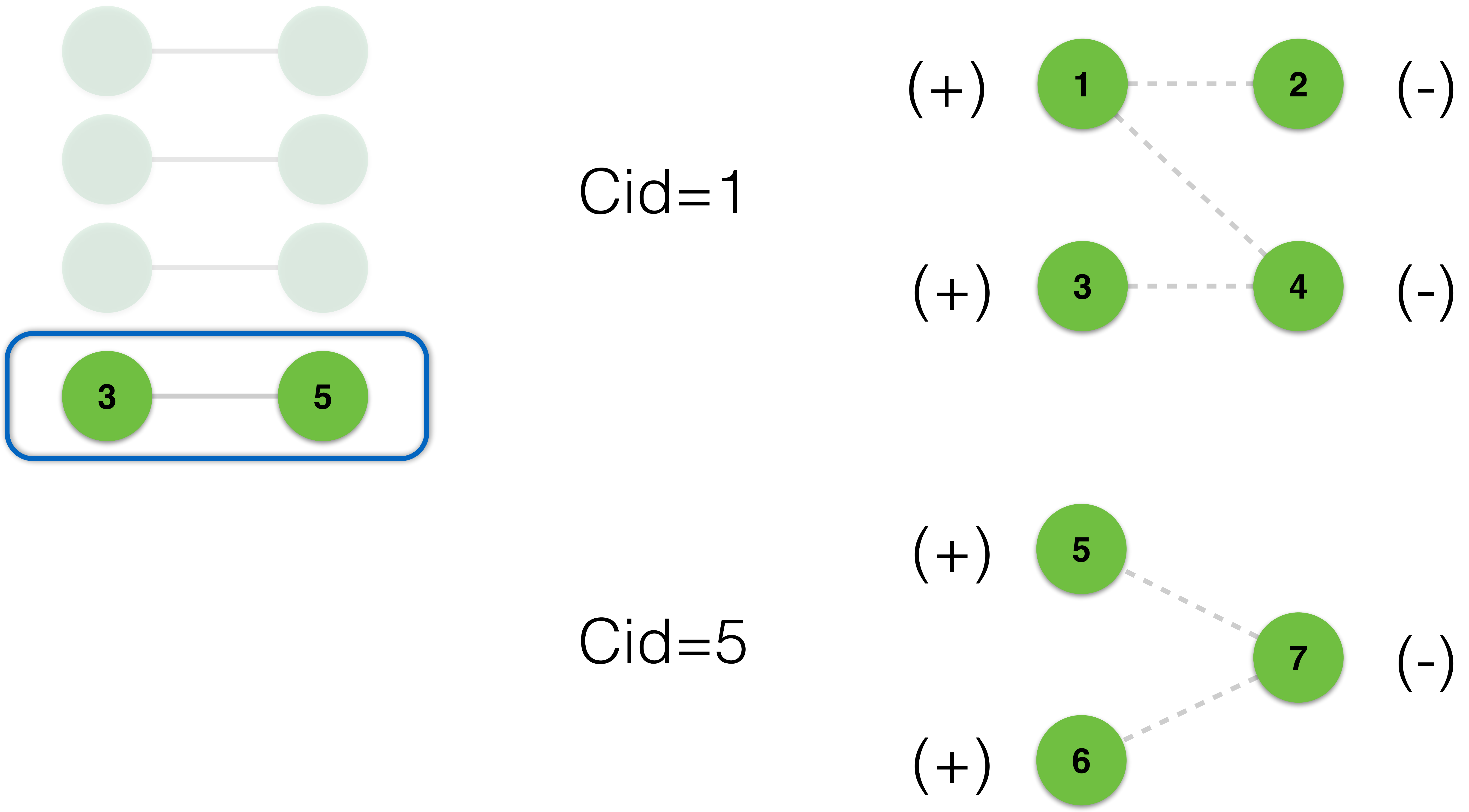
- Each vertex is also assigned a sign, (+) or (-)
- Edge endpoints must have different signs
- When merging components, if flipping all signs doesn't work => the graph is not bipartite



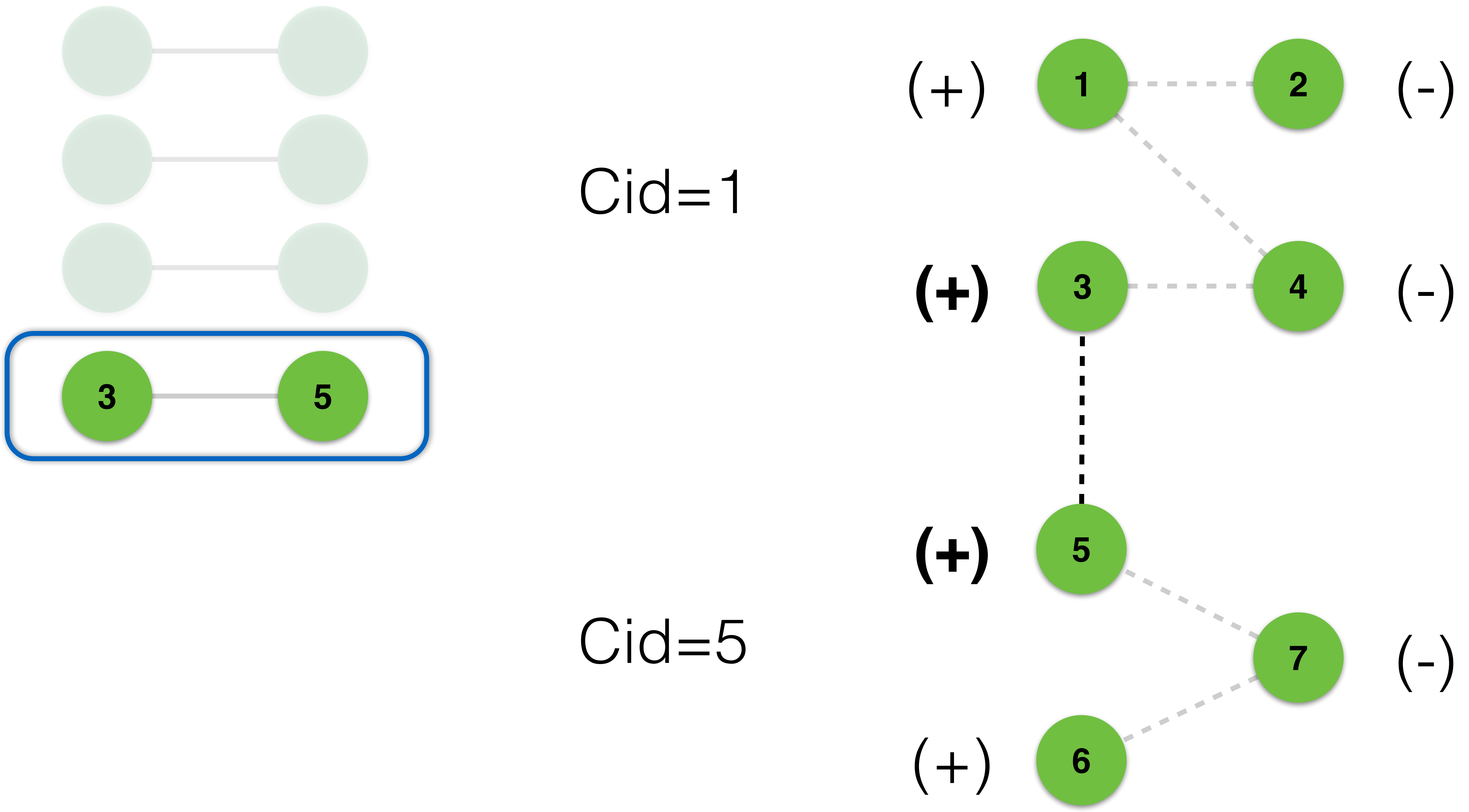
Bipartite graph checking



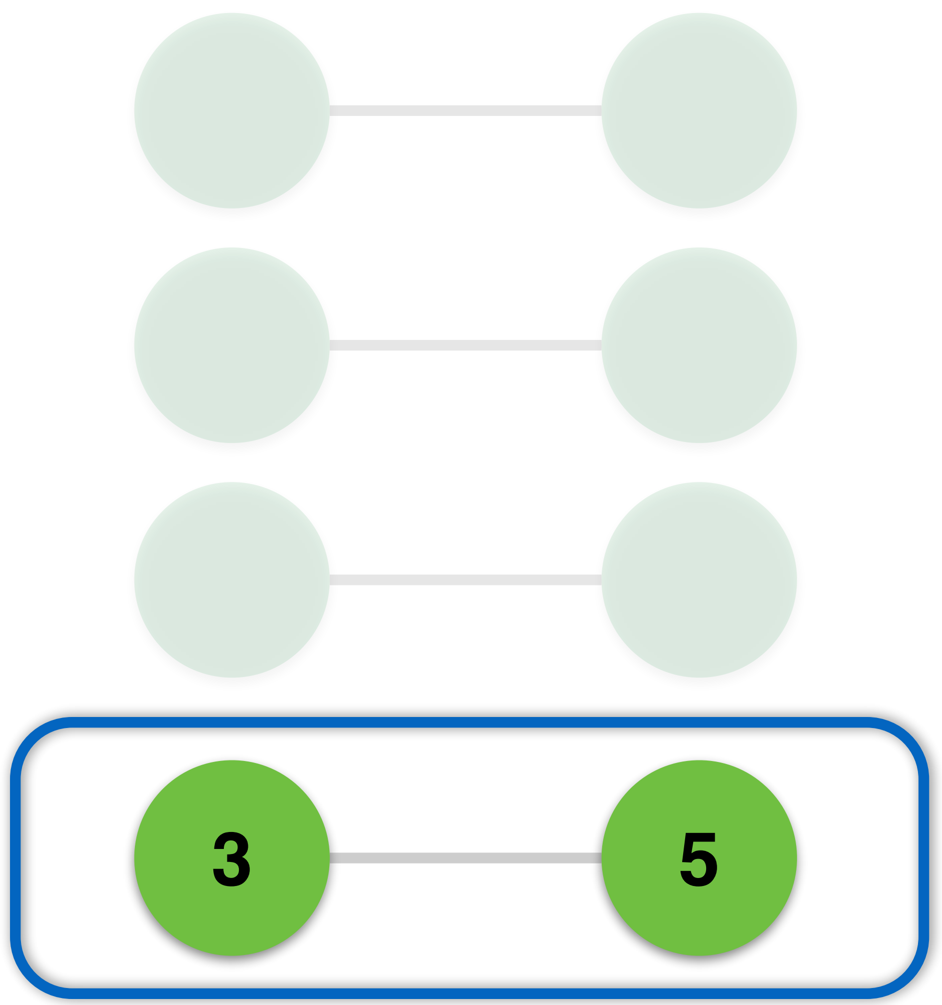
Bipartite graph checking



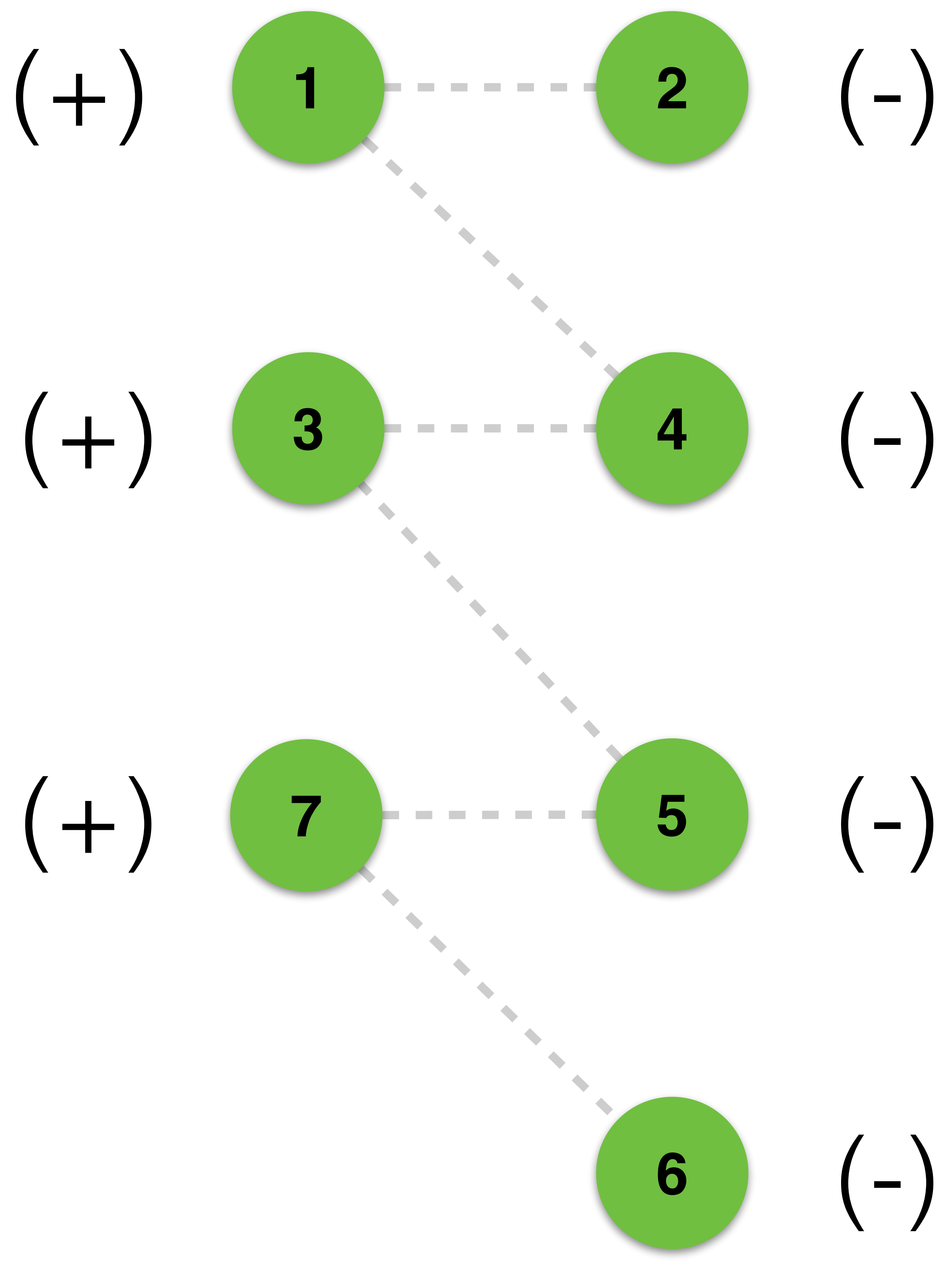
Bipartite graph checking



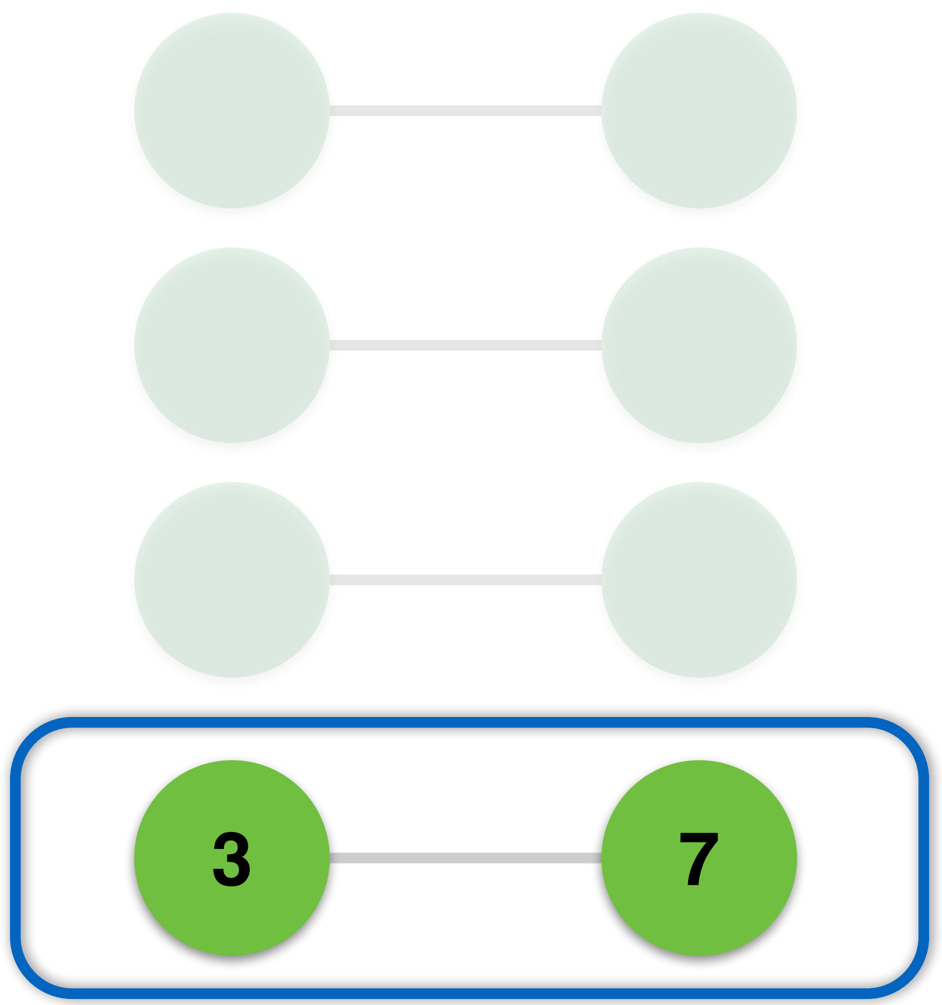
Bipartite graph checking



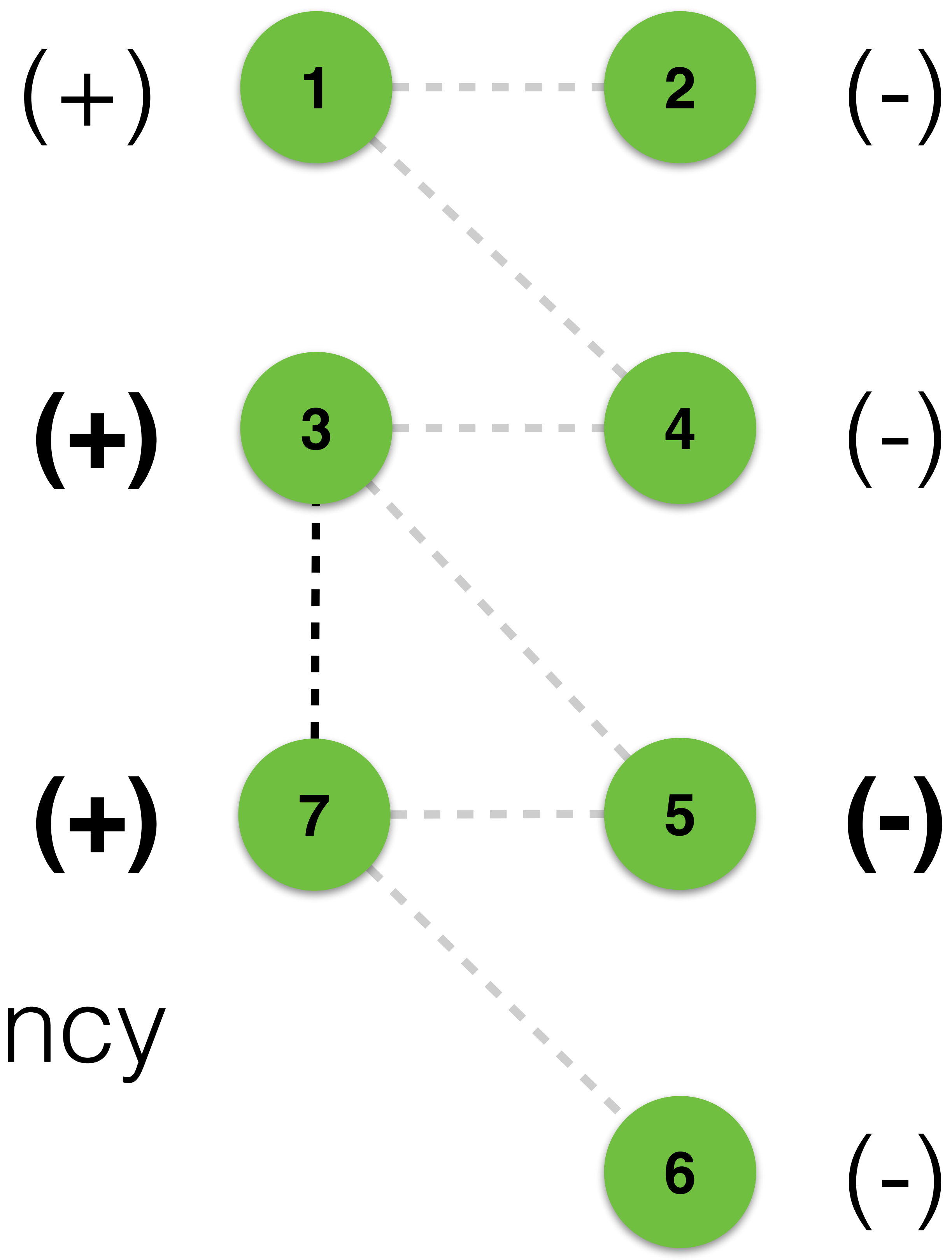
Cid=1



Bipartite graph checking



Cid=1



Can't flip signs and maintain consistency
=> not bipartite.

Spanners

Distance estimation

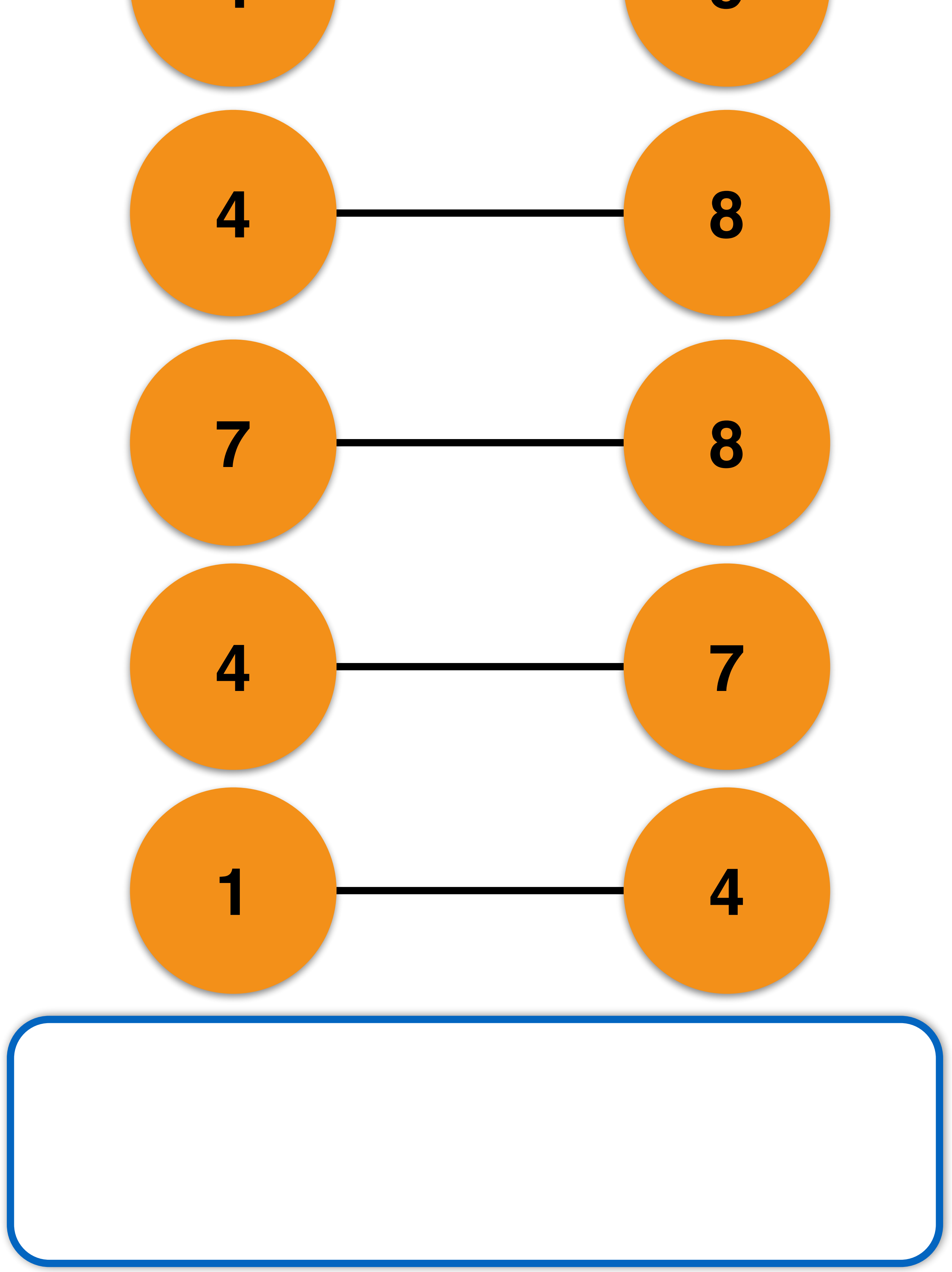
- Consider an undirected unweighted graph, $G = (V, E)$.
- We want to estimate the distance between any pair of nodes u, v as the length of the shortest path between them.
- A **spanner** H of graph G is a subgraph of G with fewer edges and the same set of vertices: $E(H) \subseteq E(G), V(H) = V(G)$.

The k-spanner synopsis

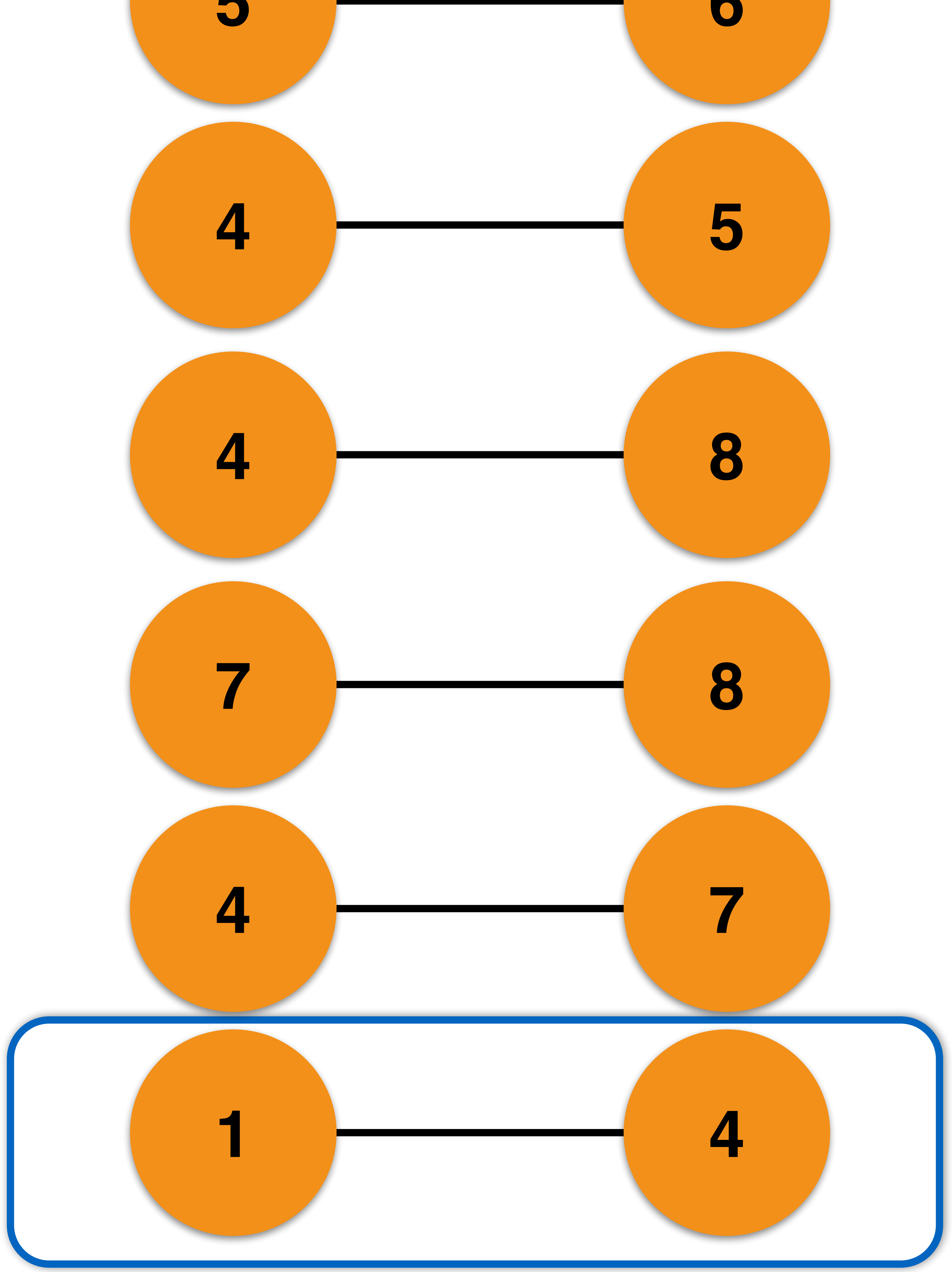
A k-spanner is a graph synopsis that preserves the distances between any pair of nodes up to a factor of k:

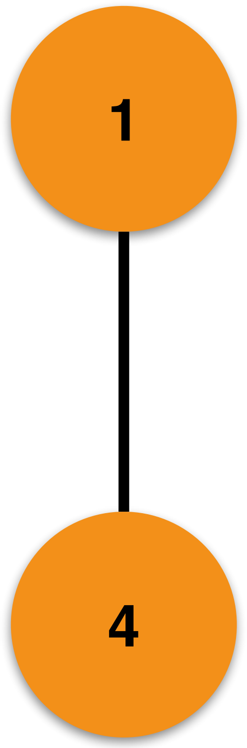
$$\forall (u, v) \in V, d_G(u, v) \leq d_H(u, v) \leq k \cdot d_G(u, v)$$

```
initialize all distances to maxValue
E(H) = {}
for (u, v) in input do
  if dH(u,v) > k then
    E(H).add((u,v))
```



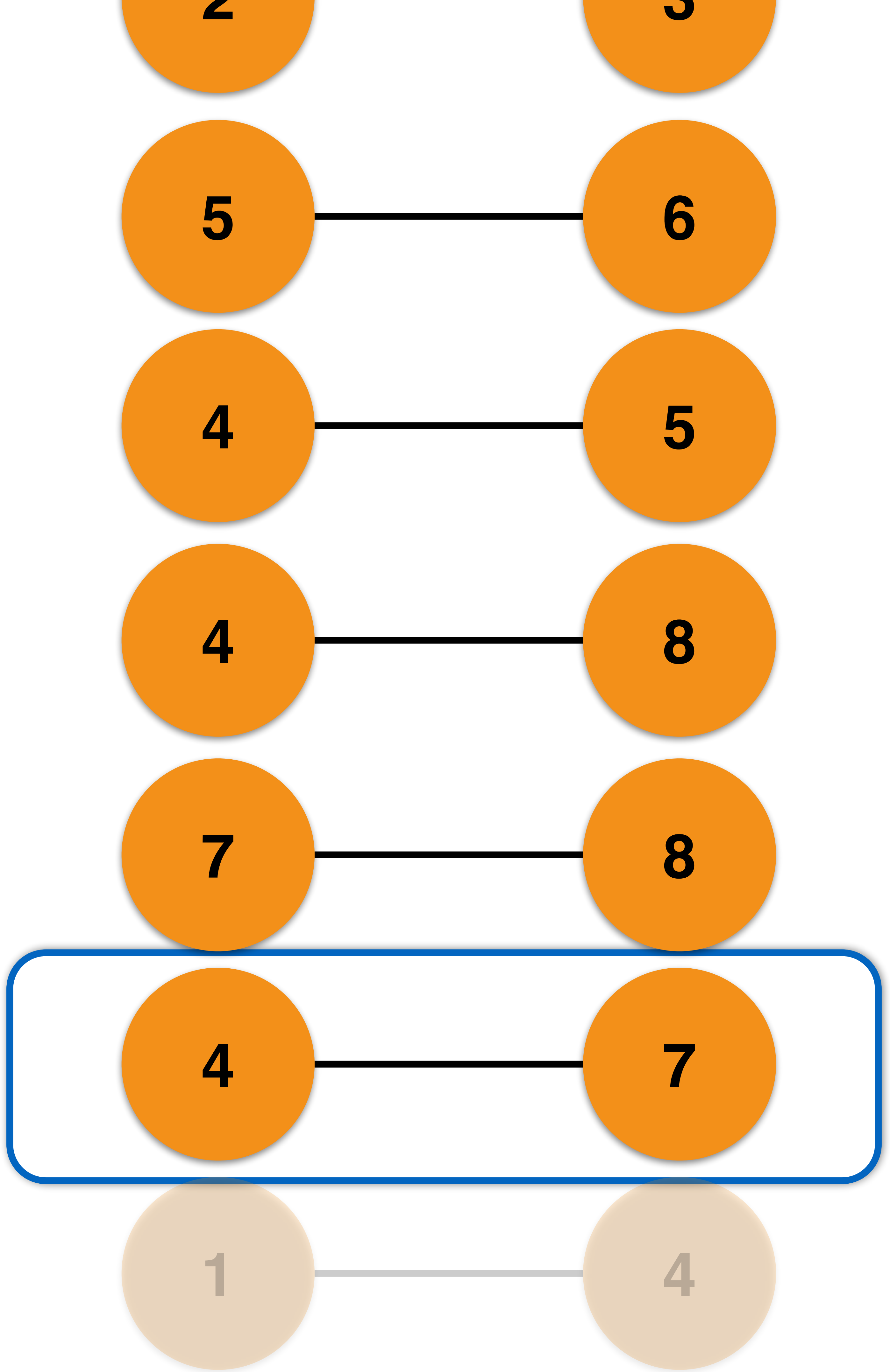
k=3

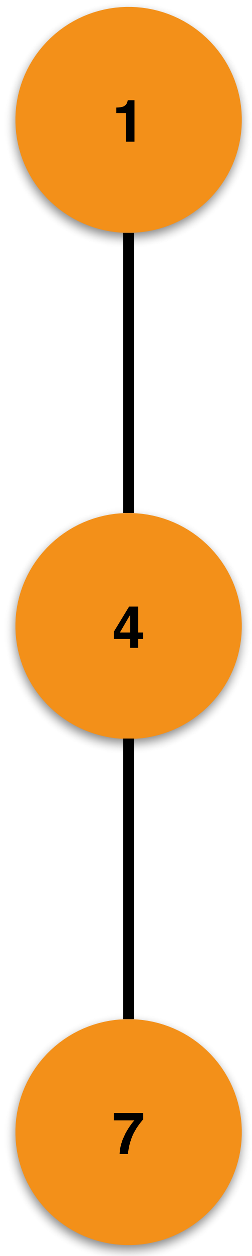




$k=3$

$d(1, 4) = 1$

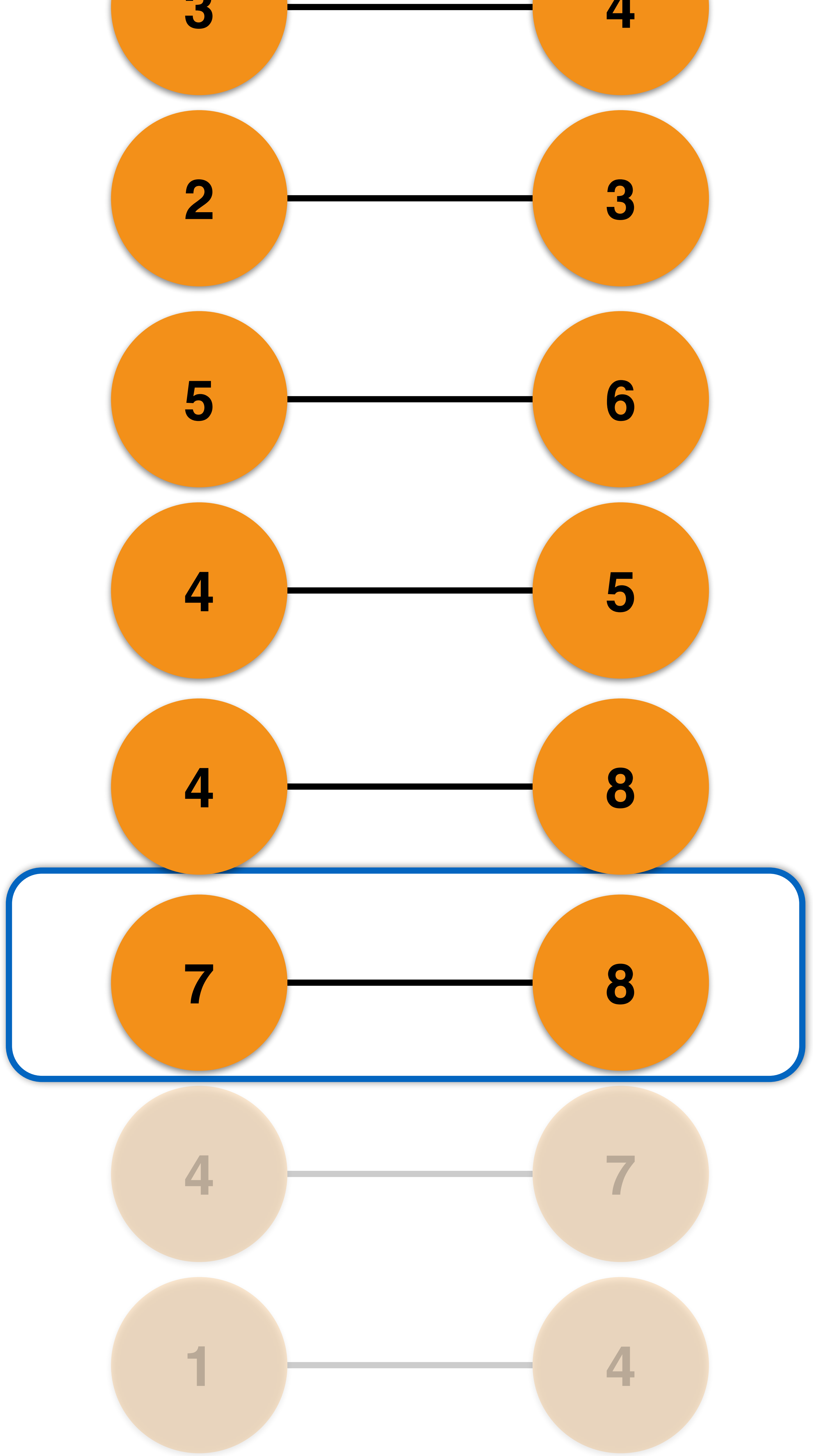


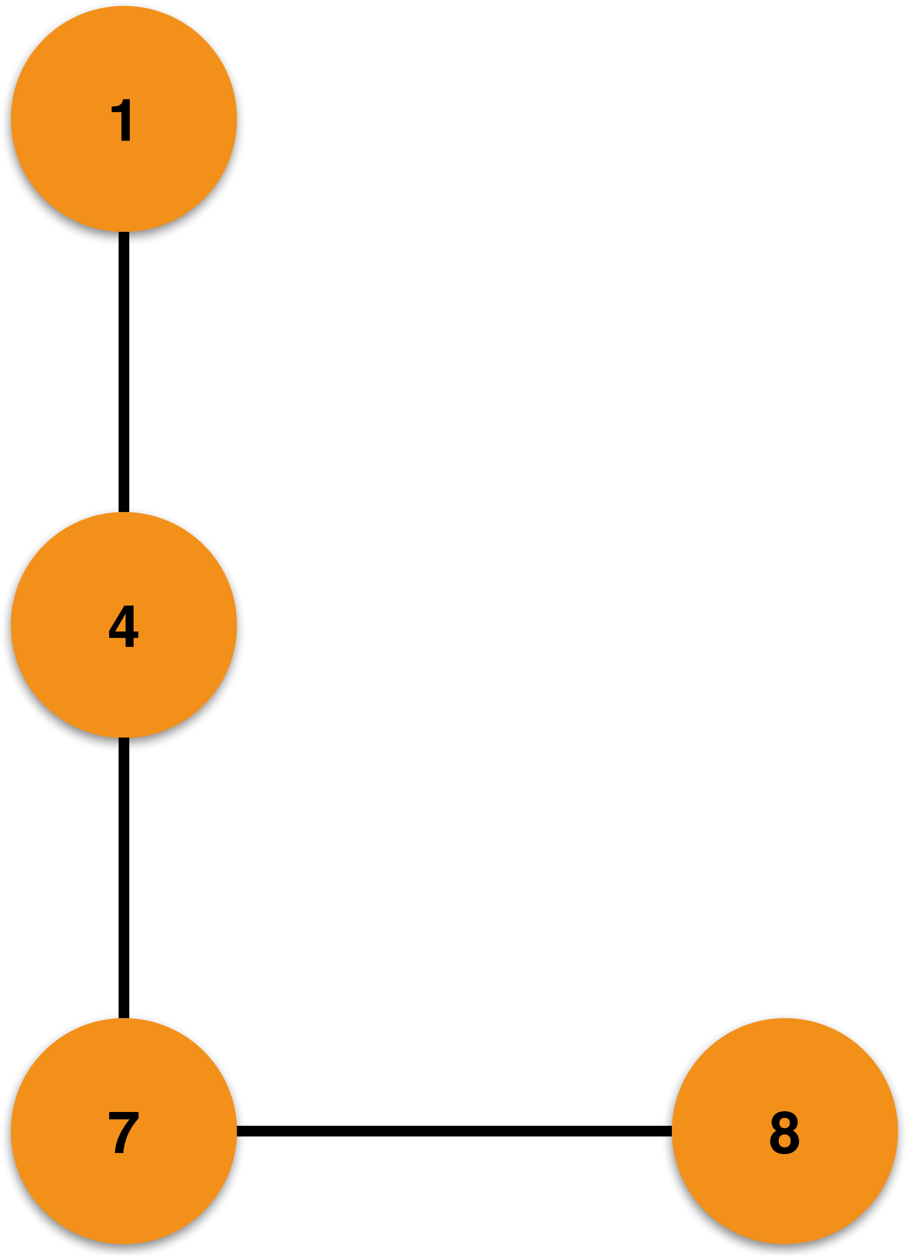


k=3

d(1, 4) = 1

d(4, 7) = 1



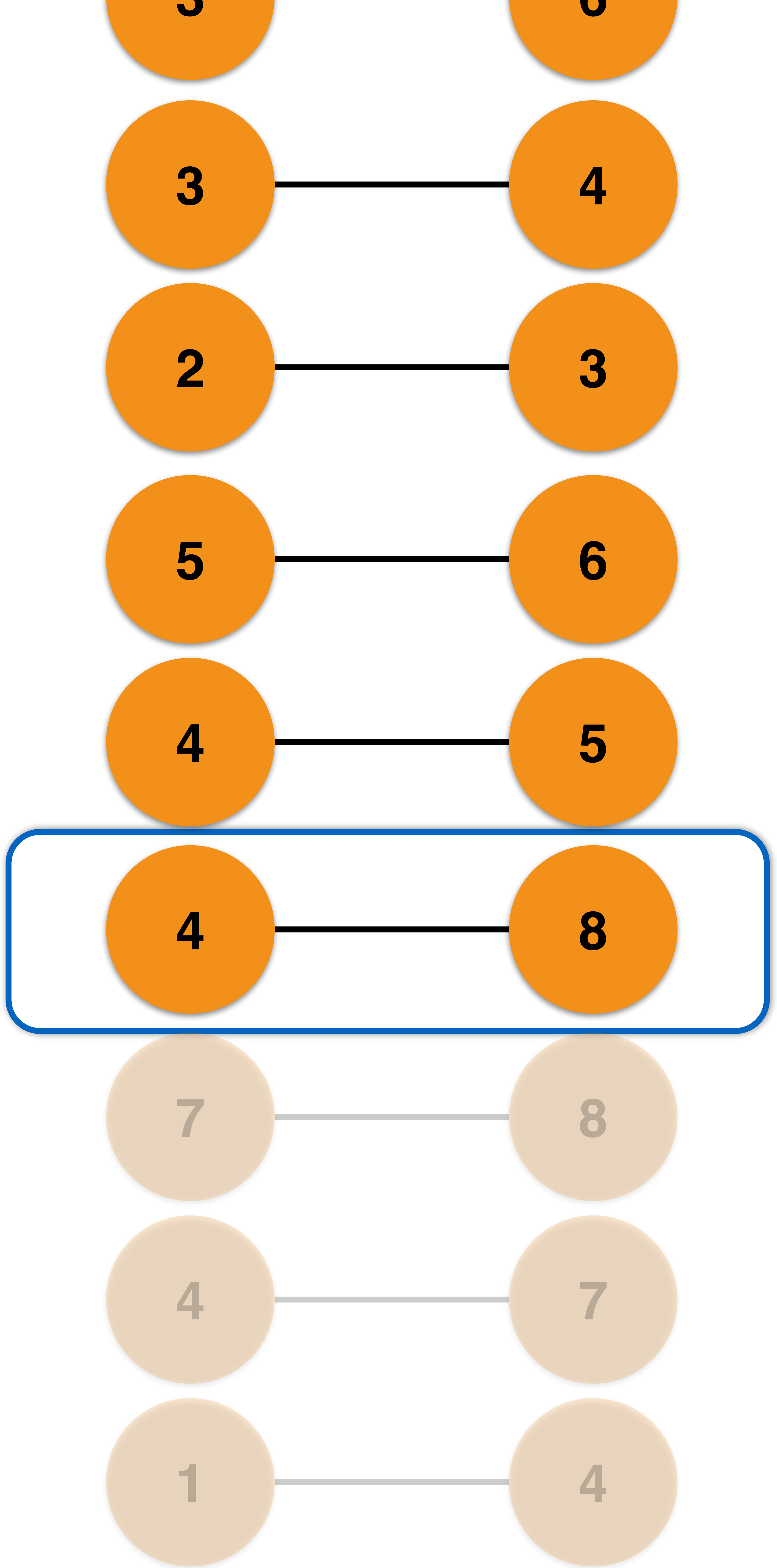


k=3

d(1, 4) = 1

d(4, 7) = 1

d(7, 8) = 1

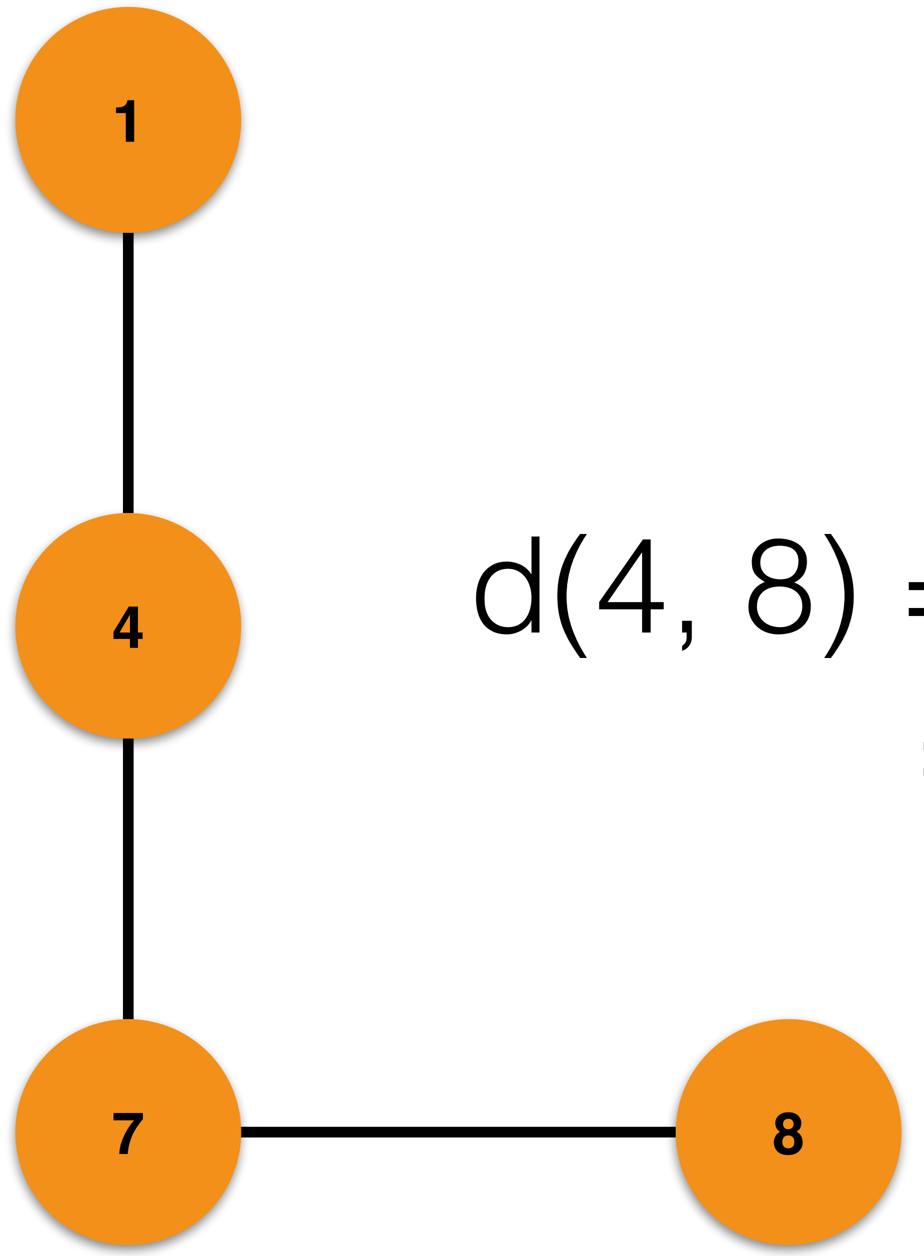


k=3

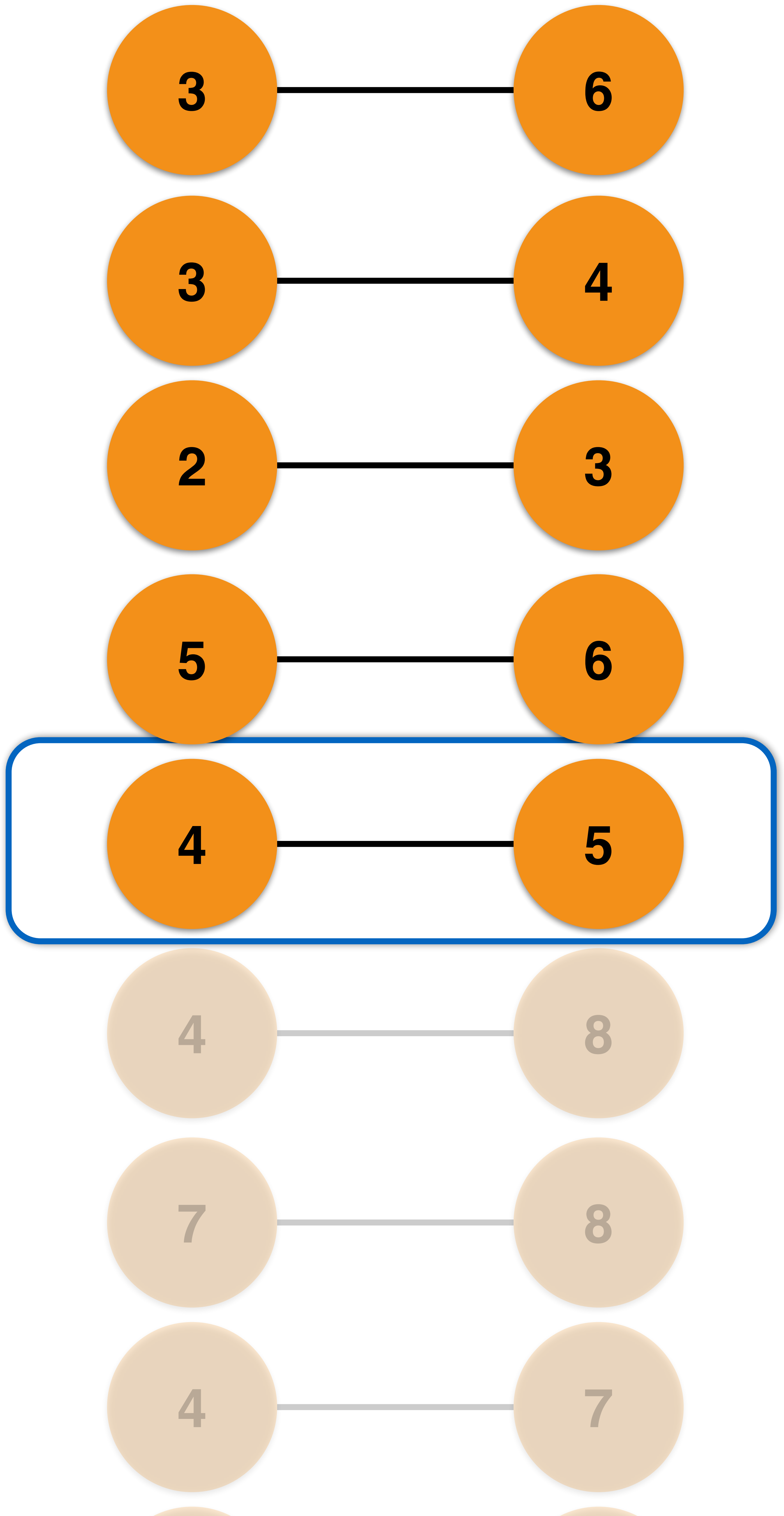
$d(1, 4) = 1$

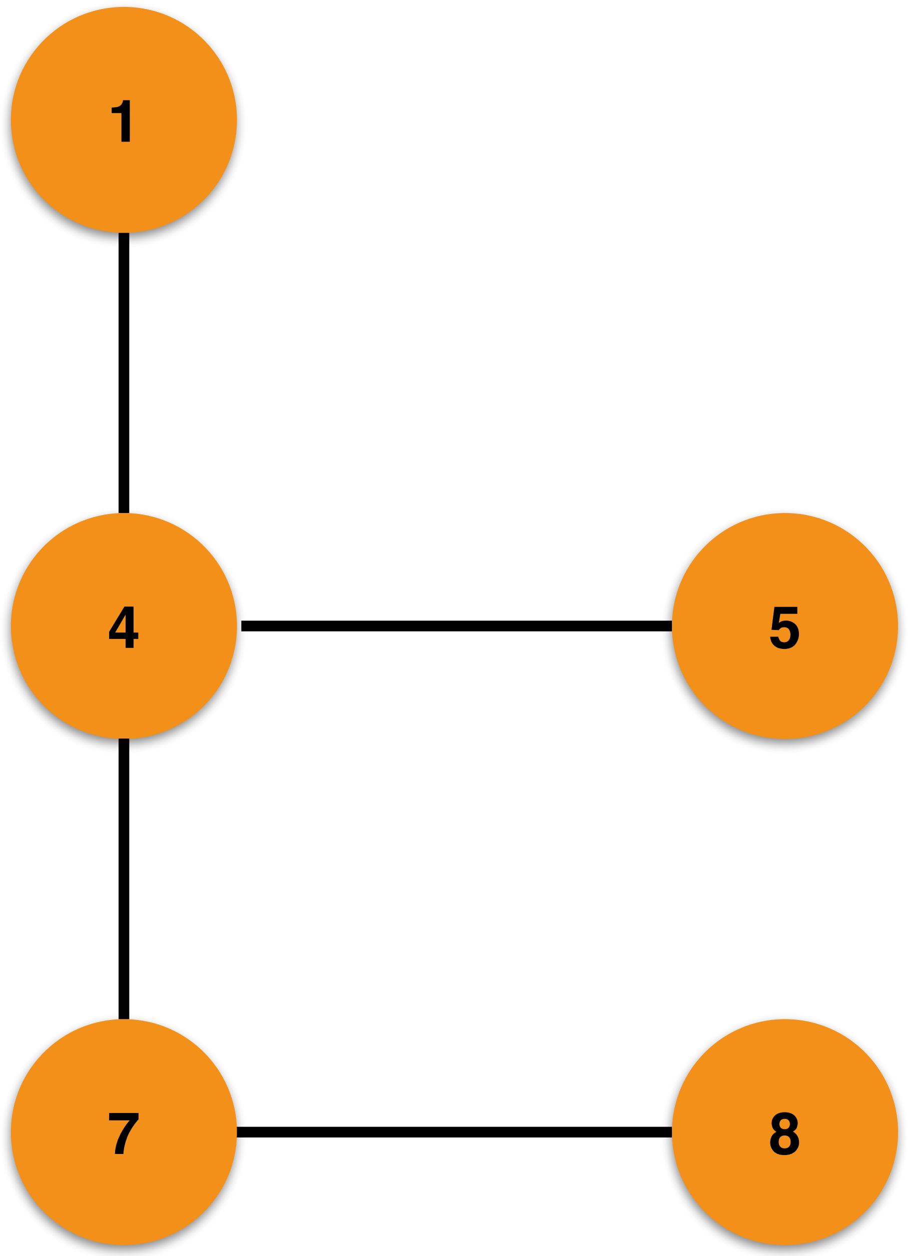
$d(4, 7) = 1$

$d(7, 8) = 1$



$$d(4, 8) = d(4, 7) + d(7, 8) = 2 < 3$$





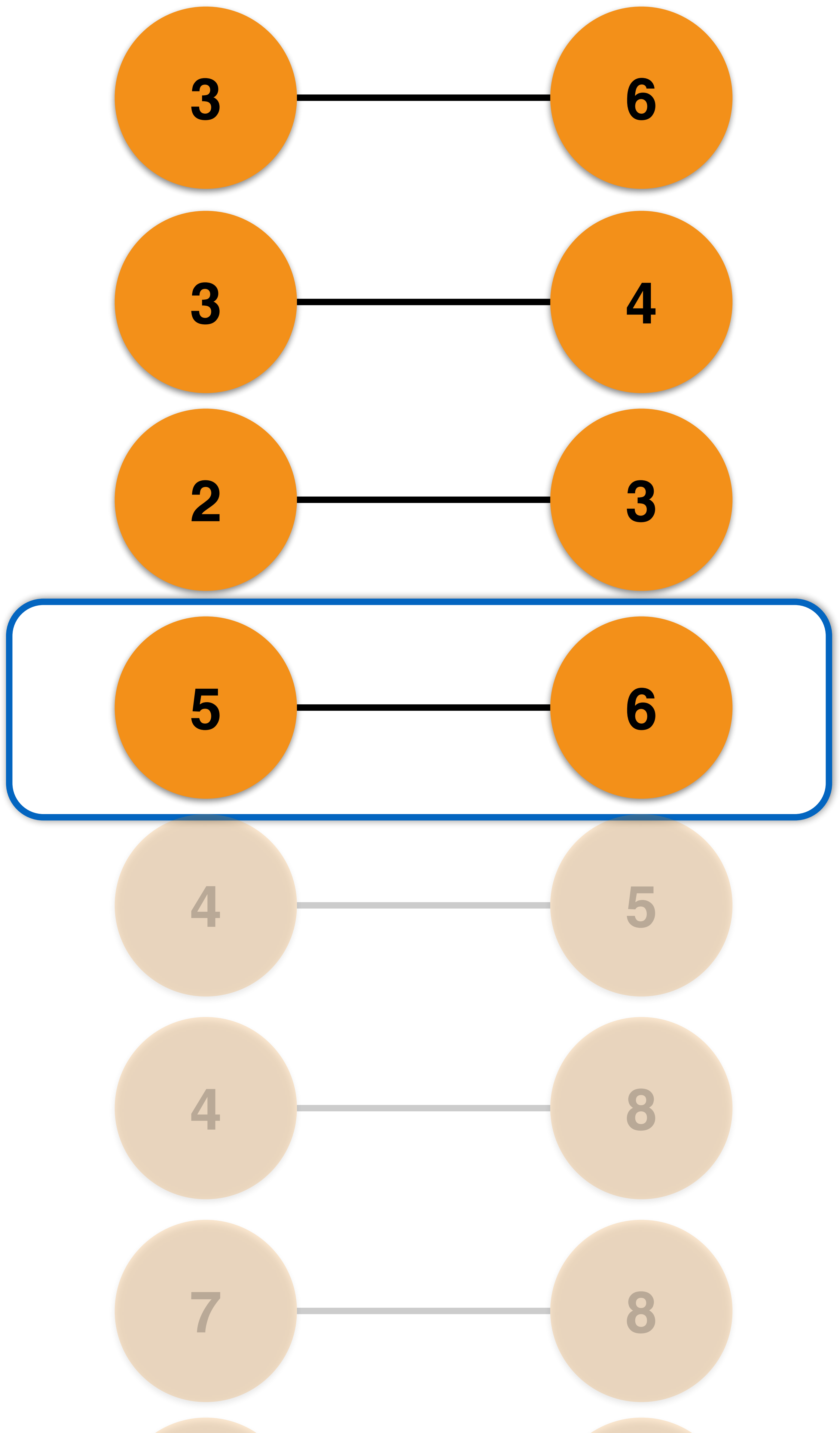
k=3

d(1, 4) = 1

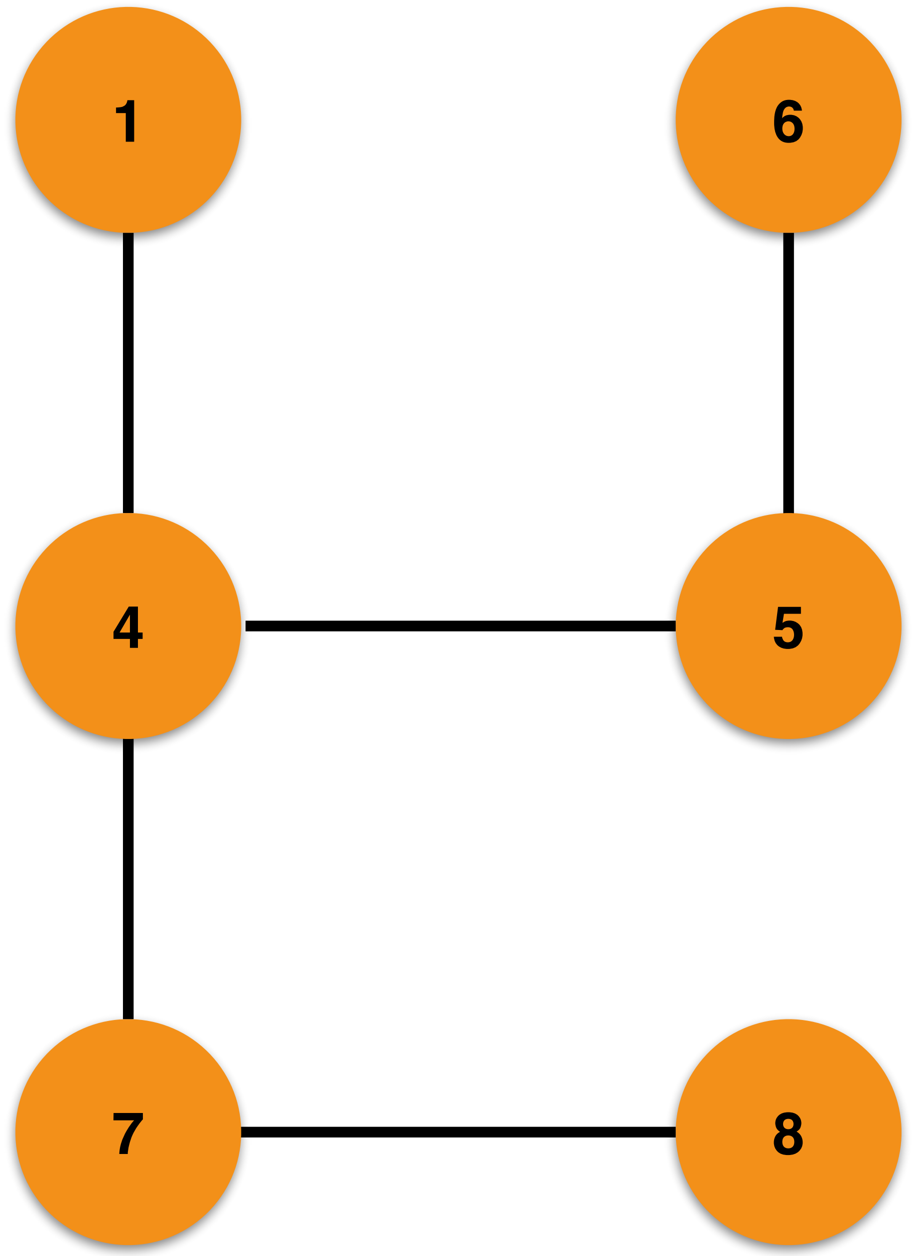
d(4, 7) = 1

d(7, 8) = 1

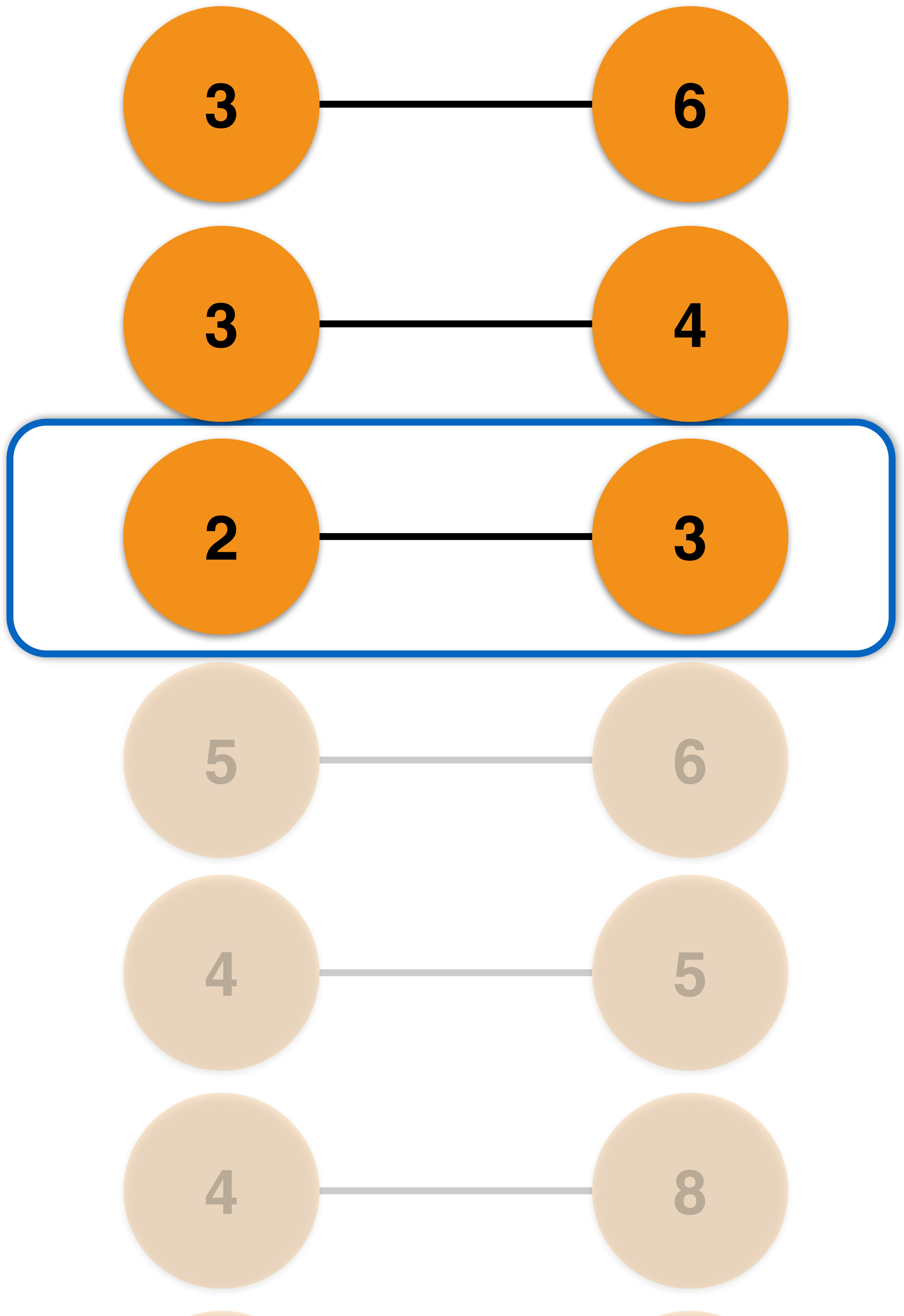
d(4, 5) = 1



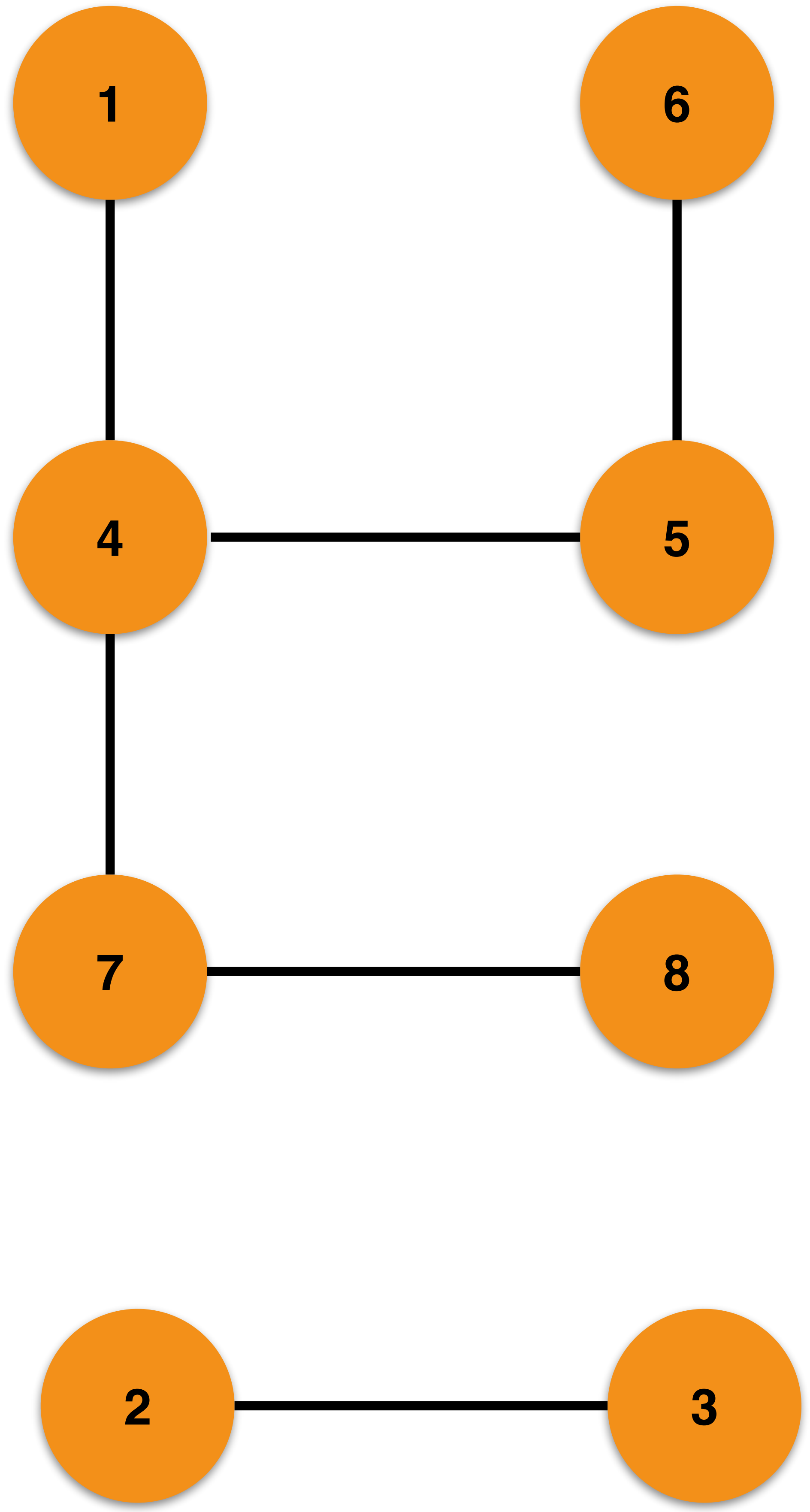
k=3



d(1, 4) = 1
d(4, 7) = 1
d(7, 8) = 1
d(4, 5) = 1
d(5, 6) = 1

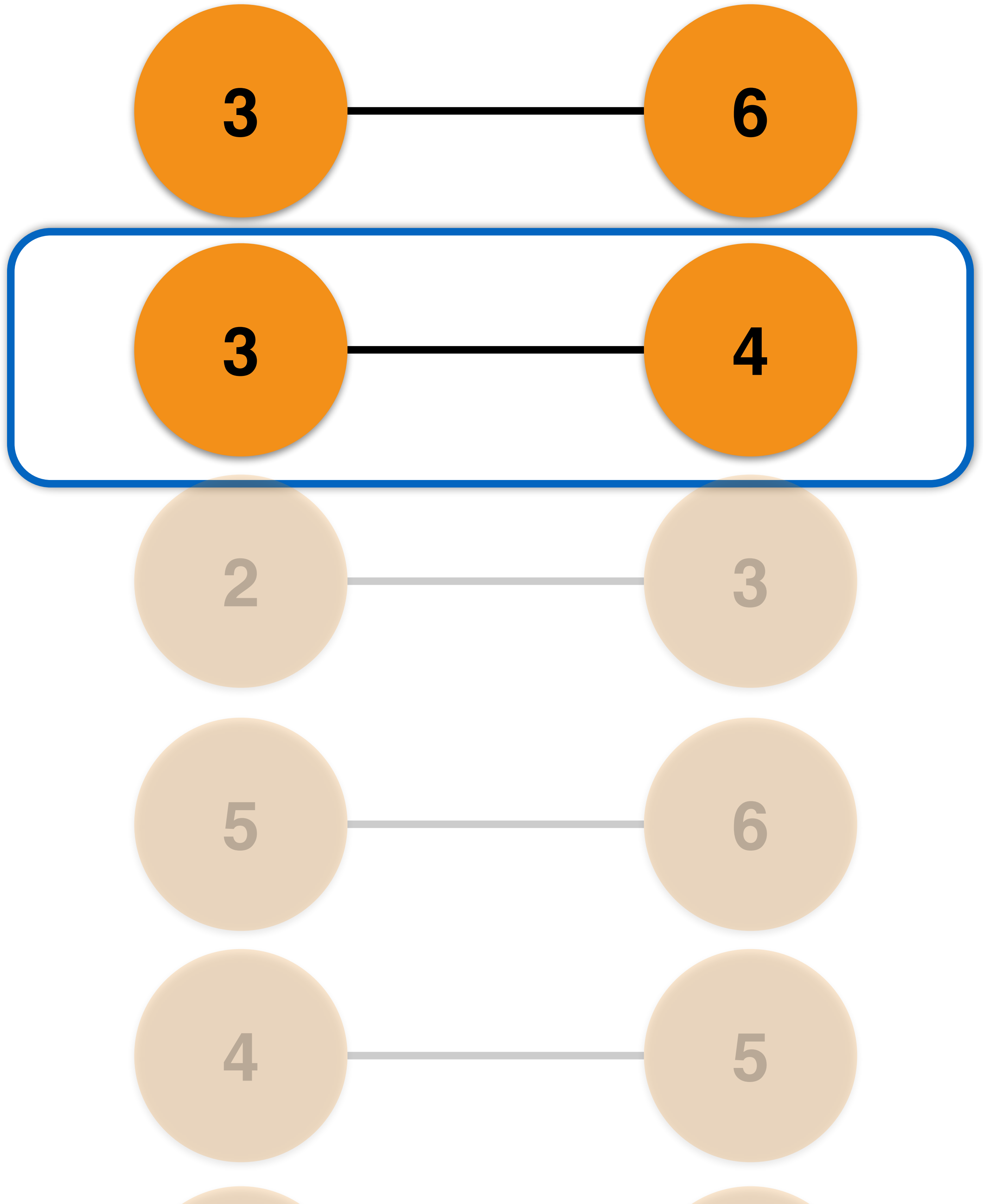


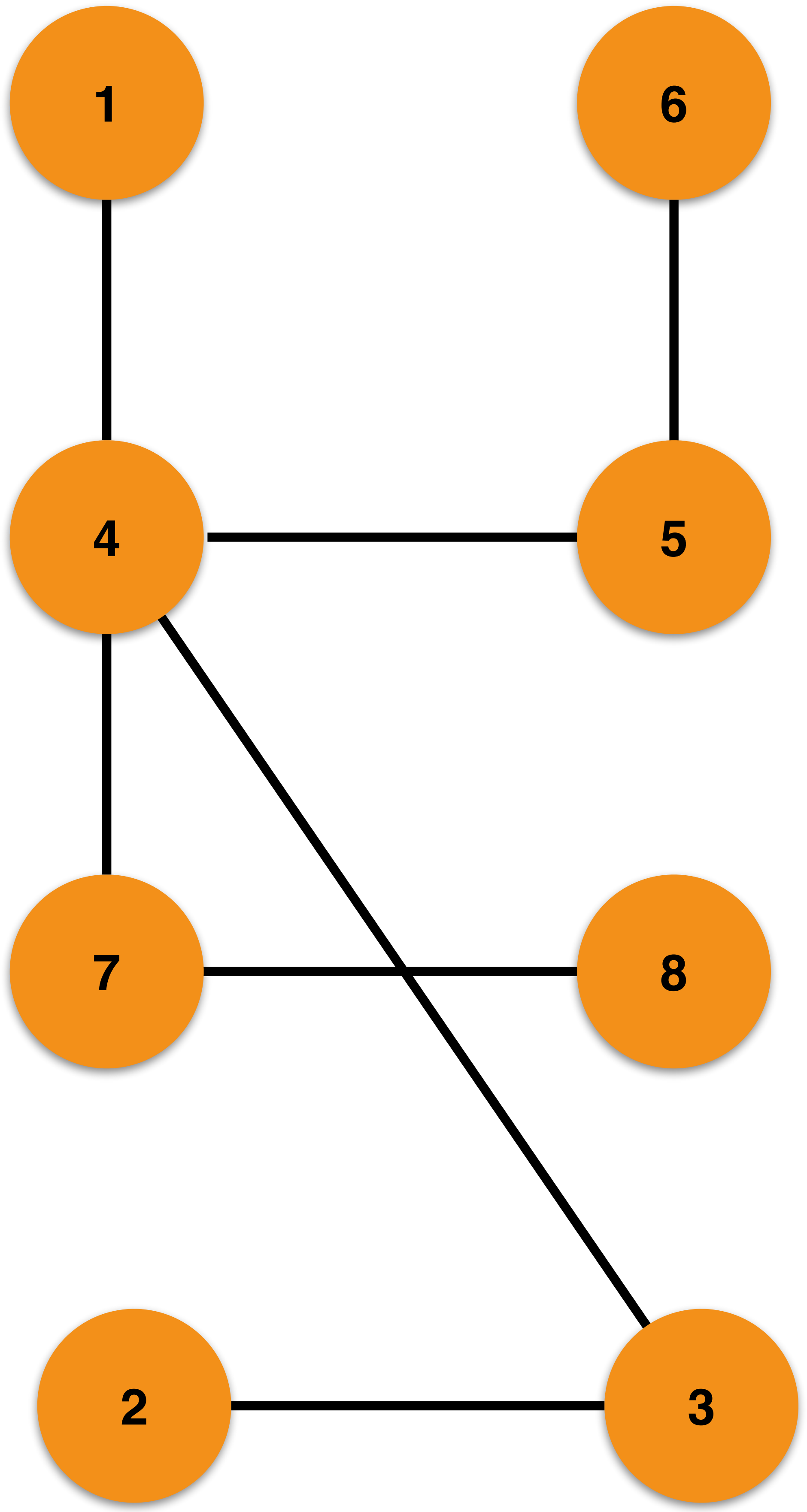
k=3



d(1, 4) = 1
d(4, 7) = 1
d(7, 8) = 1
d(4, 5) = 1
d(5, 6) = 1
d(2, 3) = 1

k=3





$d(1, 4) = 1$

$d(4, 7) = 1$

$d(7, 8) = 1$

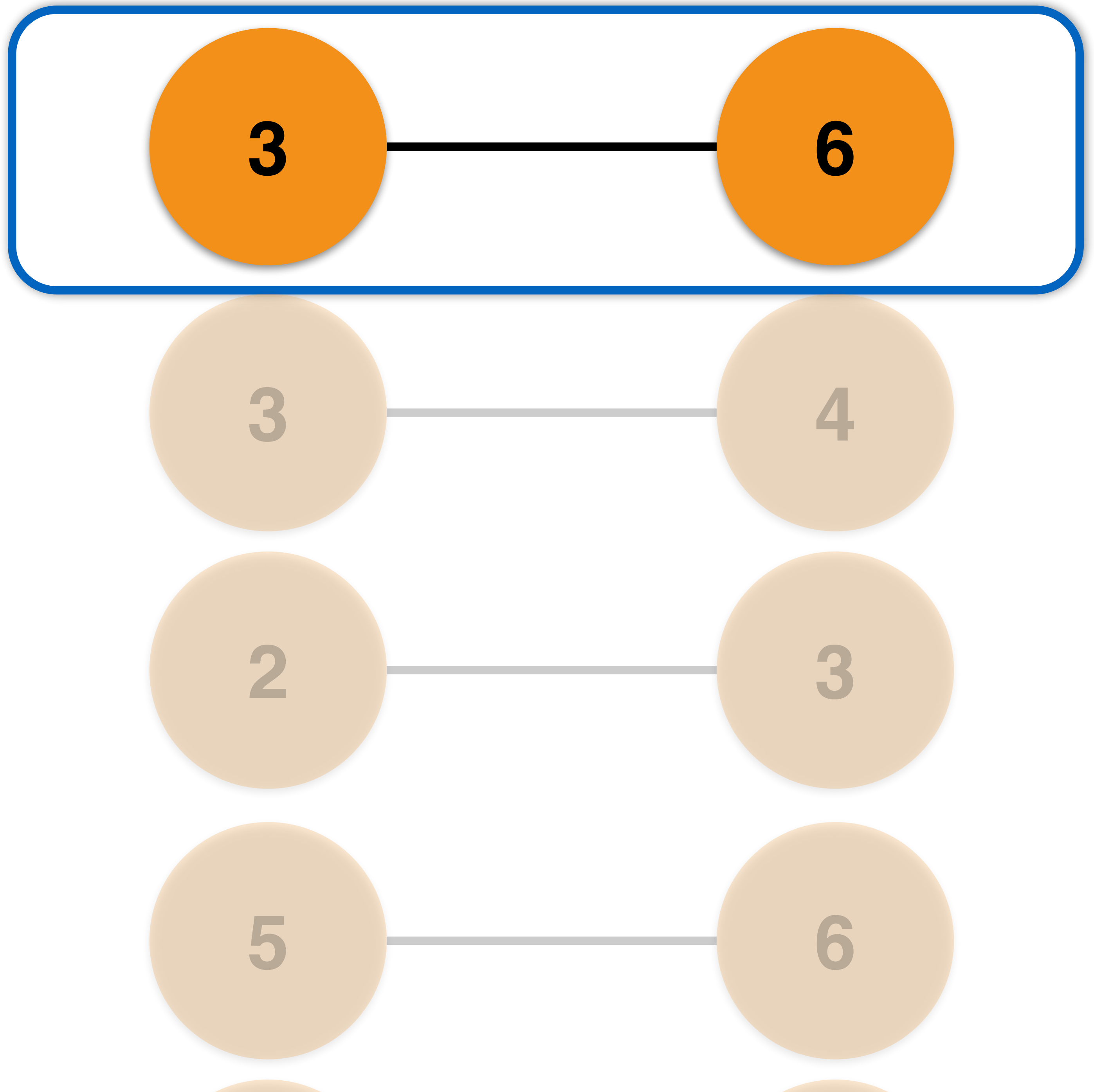
$d(4, 5) = 1$

$d(5, 6) = 1$

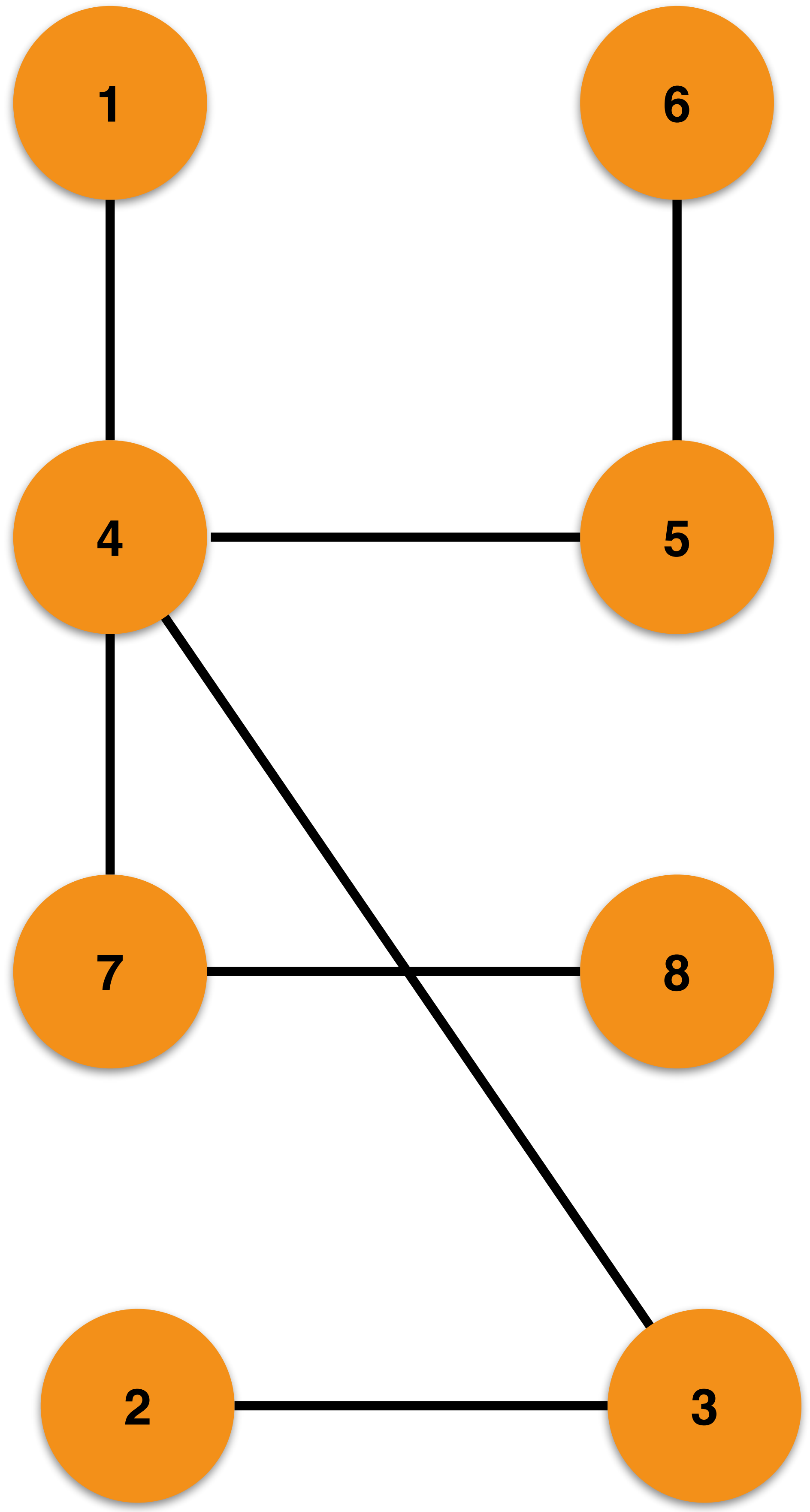
$d(2, 3) = 1$

$d(3, 4) = 1$

$$d(3, 6) = d(3, 4) + d(4, 5) + d(5, 6)$$
$$= 3$$



k=3



$d(1, 4) = 1$
$d(4, 7) = 1$
$d(7, 8) = 1$
$d(4, 5) = 1$
$d(5, 6) = 1$
$d(2, 3) = 1$
$d(3, 4) = 1$

Data-parallel streaming spanners on Flink?

- Similar challenges exist for a data-parallel implementation of spanners
- How to represent the spanner? As an adjacency list? which state primitives are suitable? Is RocksDB a suitable backend for graph state?
- How to compute the distance between edges? Do we need to do that for every incoming edge? Can we compute the distances in separate partitions and then merge them?

Further reading

- McGregor, Andrew. **Graph stream algorithms: a survey**. *ACM SIGMOD Record* 43.1 (2014). https://dl.acm.org/doi/pdf/10.1145/2627692.2627694
- Stanton, Isabelle, and Gabriel Kliot. **Streaming graph partitioning for large distributed graphs**. *ACM SIGKDD,* 2012. https://www.microsoft.com/en-us/research/wp-content/uploads/2012/08/kdd325-stanton.pdf
- Stefani, Lorenzo De, et al. **Triest: Counting local and global triangles in fully dynamic streams with fixed memory size**. *TKDD* 2017. https://www.kdd.org/kdd2016/papers/files/rfp0465-de-stefaniA.pdf