Single-pass Graph Stream Analytics with Apache Flink

Vasia Kalavri <vasia@apache.org>
Paris Carbone <senorcarbone@apache.org>

Outline

- Why Graph Streaming?
- Single-Pass Algorithms Examples
- Apache Flink Streaming API
- The GellyStream API

Real Graphs are *dynamic*

Graphs are created from **events** happening in real-time

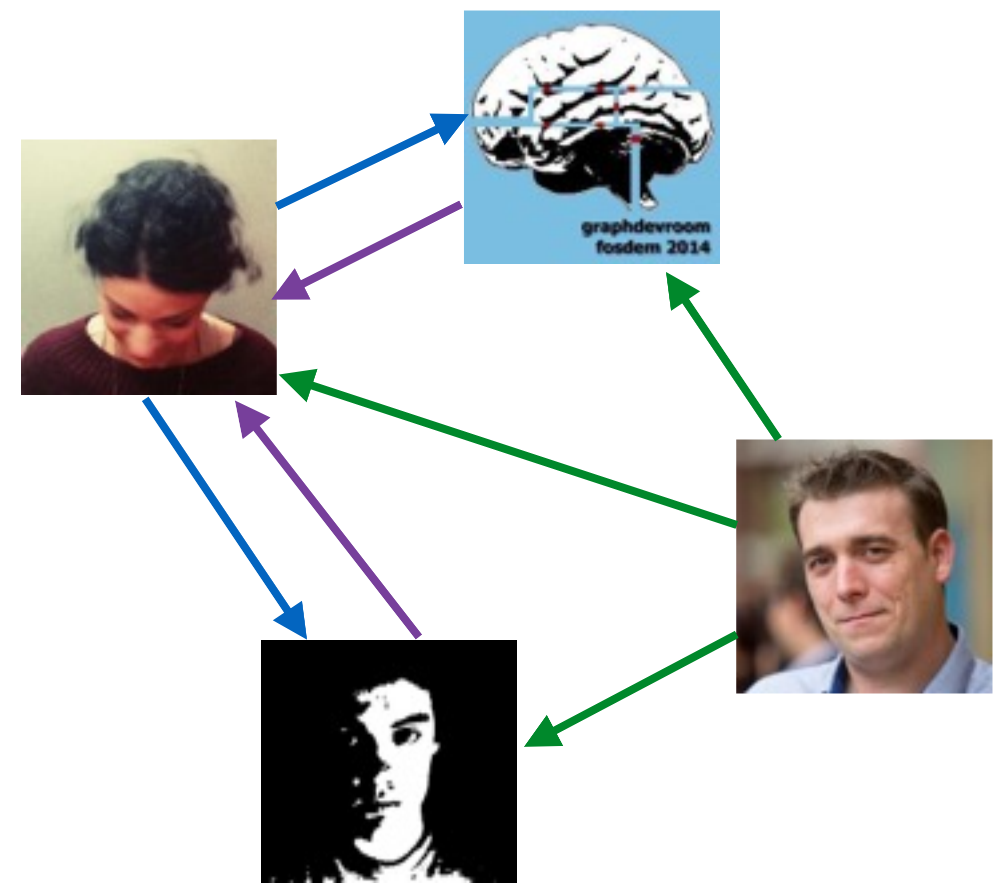
@GraphDevroom

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



Batch Graph Processing

We create and 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

Streaming Graph Processing

We consume events in **real-time**

- Get results *faster*
  - No need to wait for the job to finish
  - Sometimes, early approximations are better than late exact answers
- Get results *continuously*
  - Process *unbounded* number of events

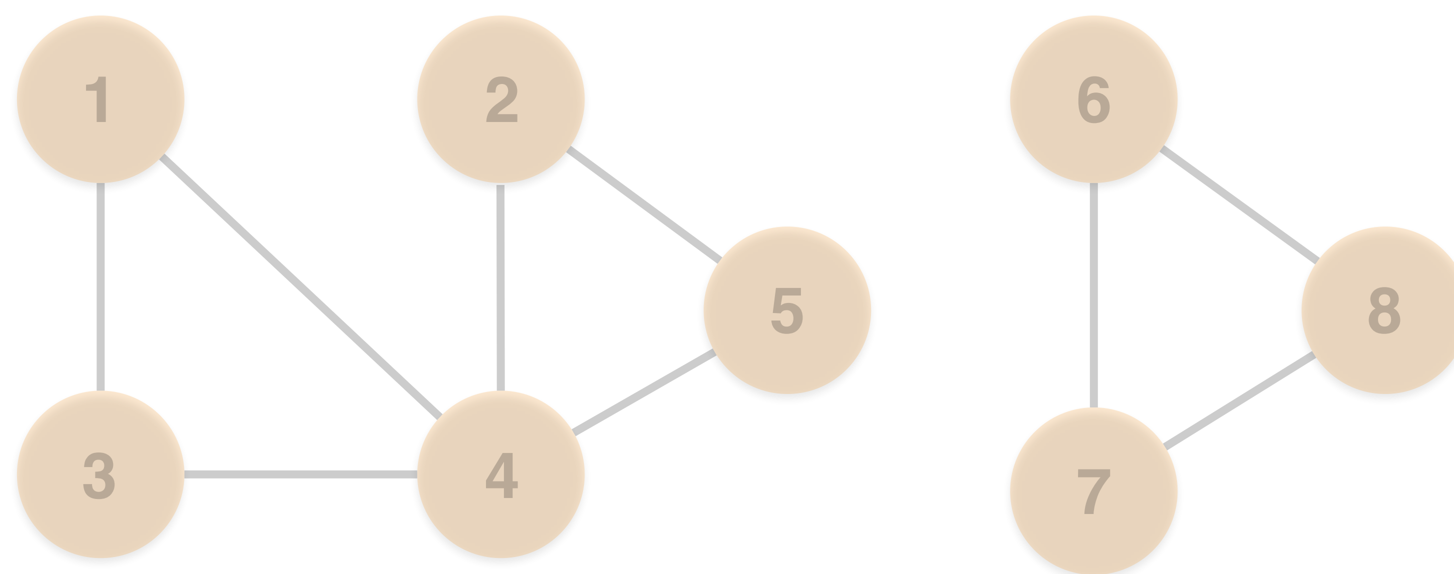
Challenges

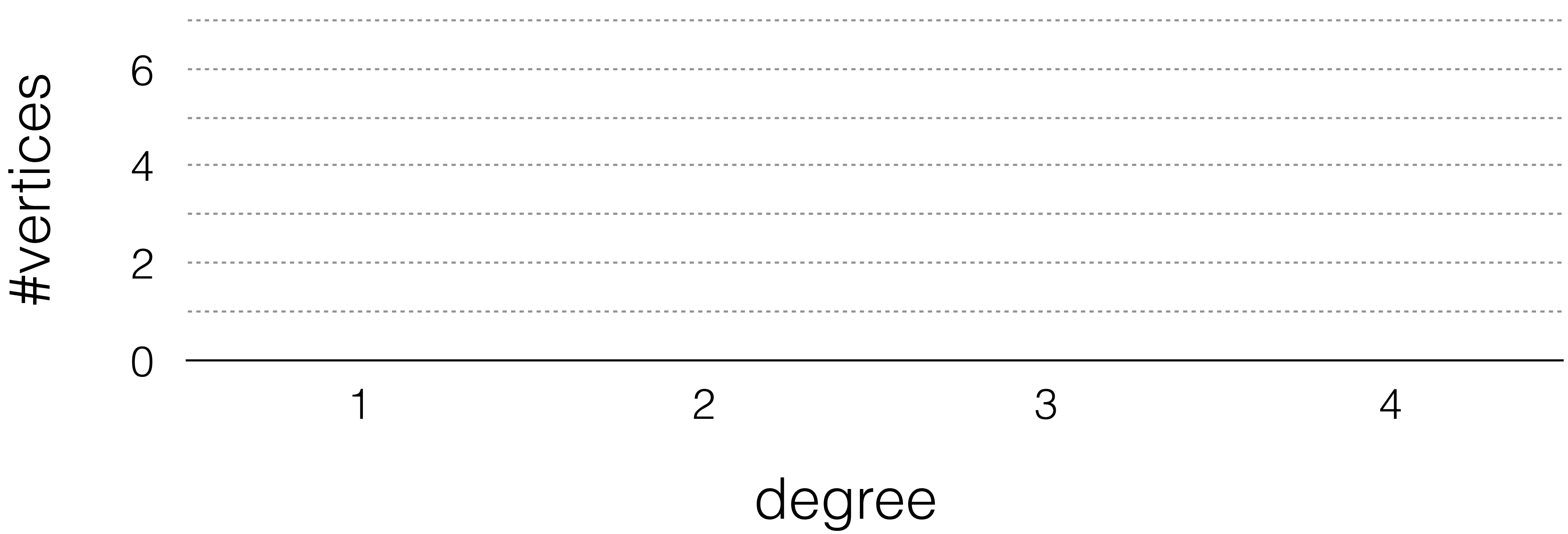
- Maintain the graph structure
  - How to apply state updates efficiently?
- Result updates
  - Re-run the analysis for each event?
  - Design an incremental algorithm?
  - Run separate instances on multiple snapshots?
- Computation on most recent events only

Single-Pass Graph Streaming

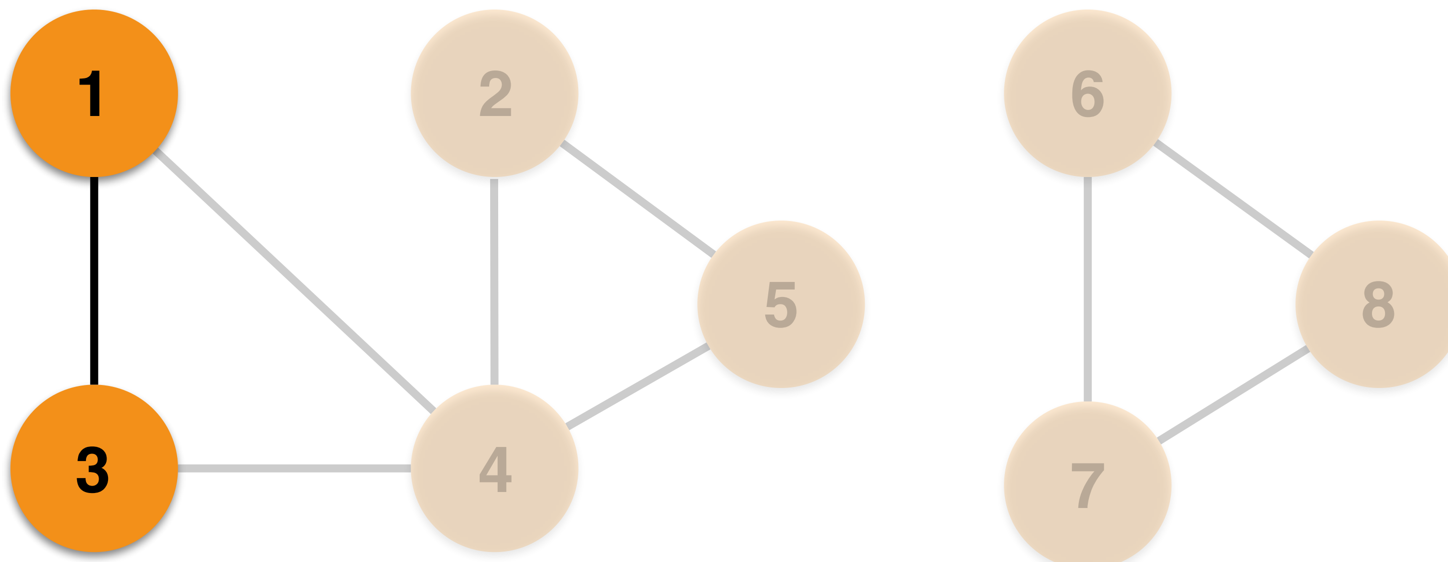
- Each event is an edge addition
- Maintains only a graph *summary*
- Recent events are grouped in graph *windows*

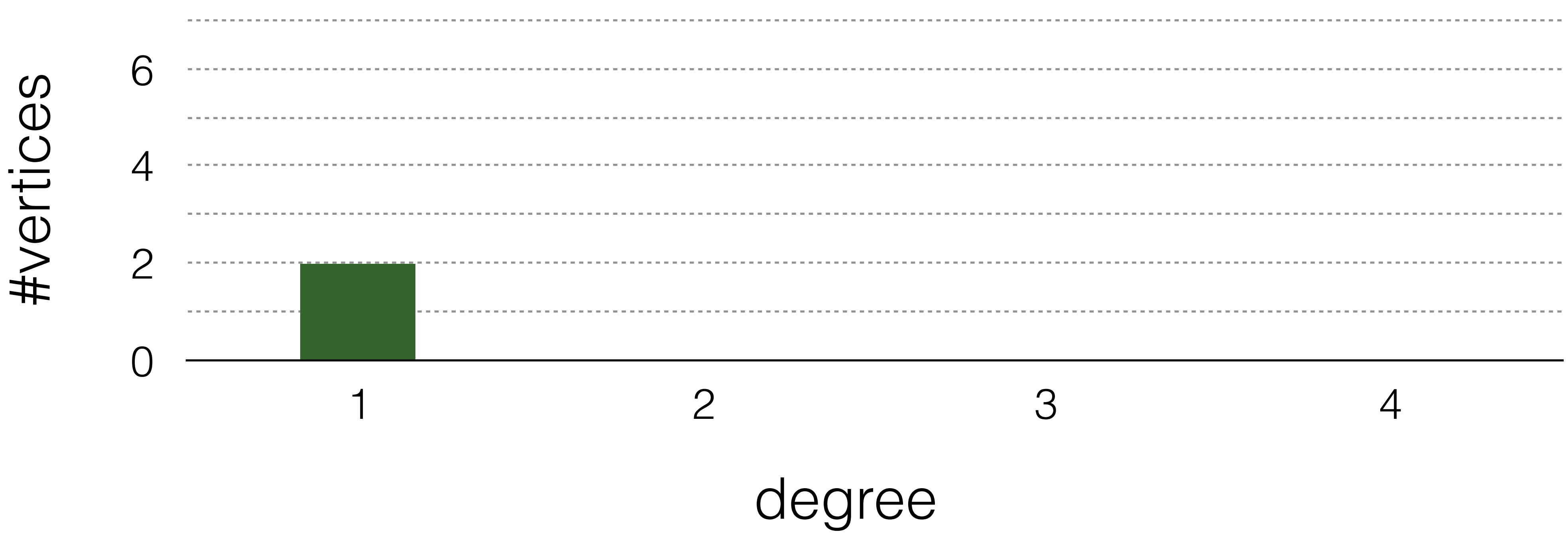
Streaming Degrees Distribution



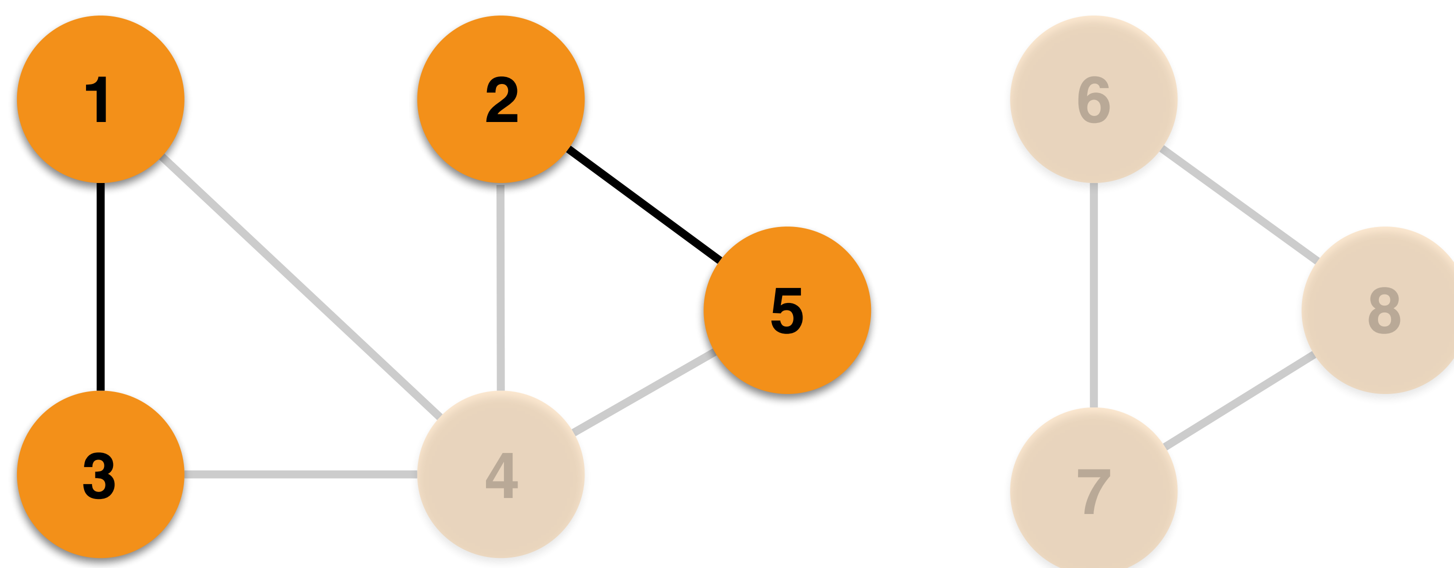


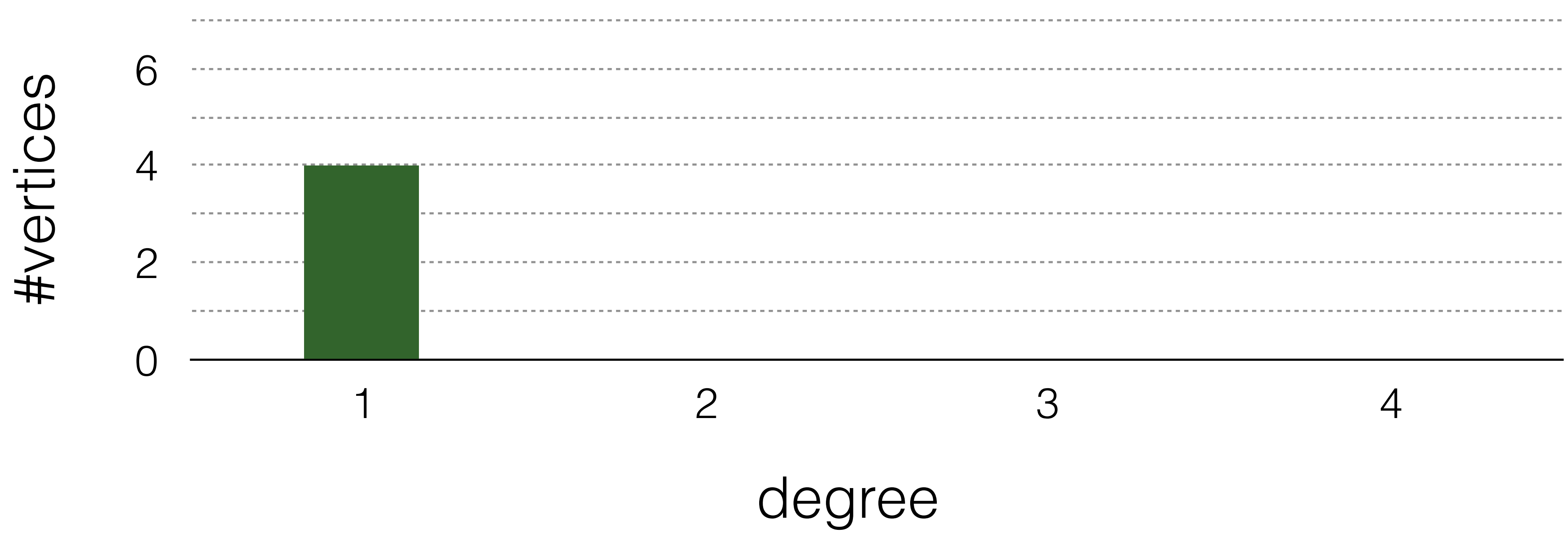
Streaming Degrees Distribution



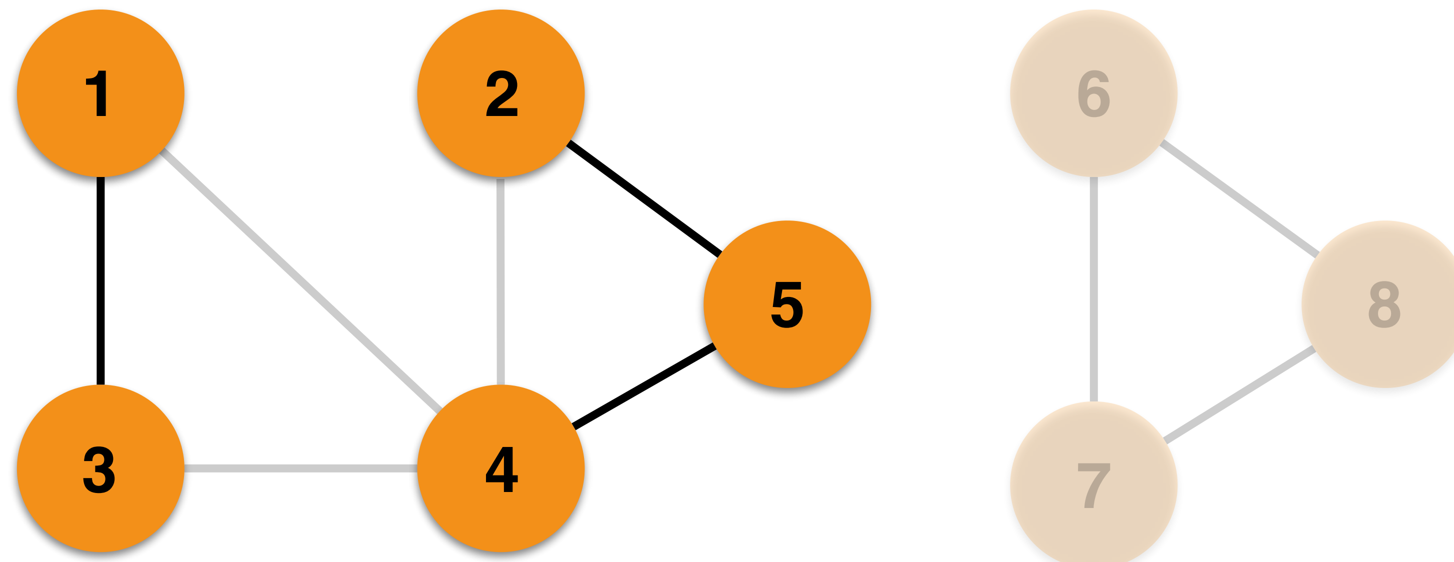


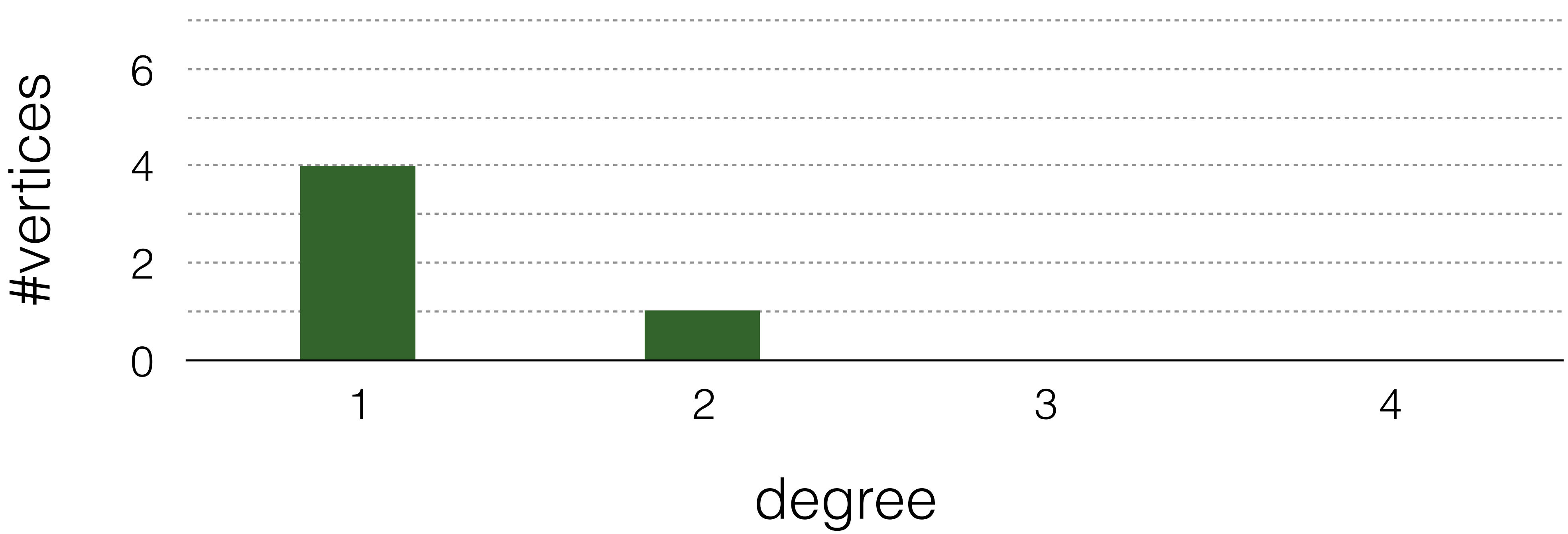
Streaming Degrees Distribution



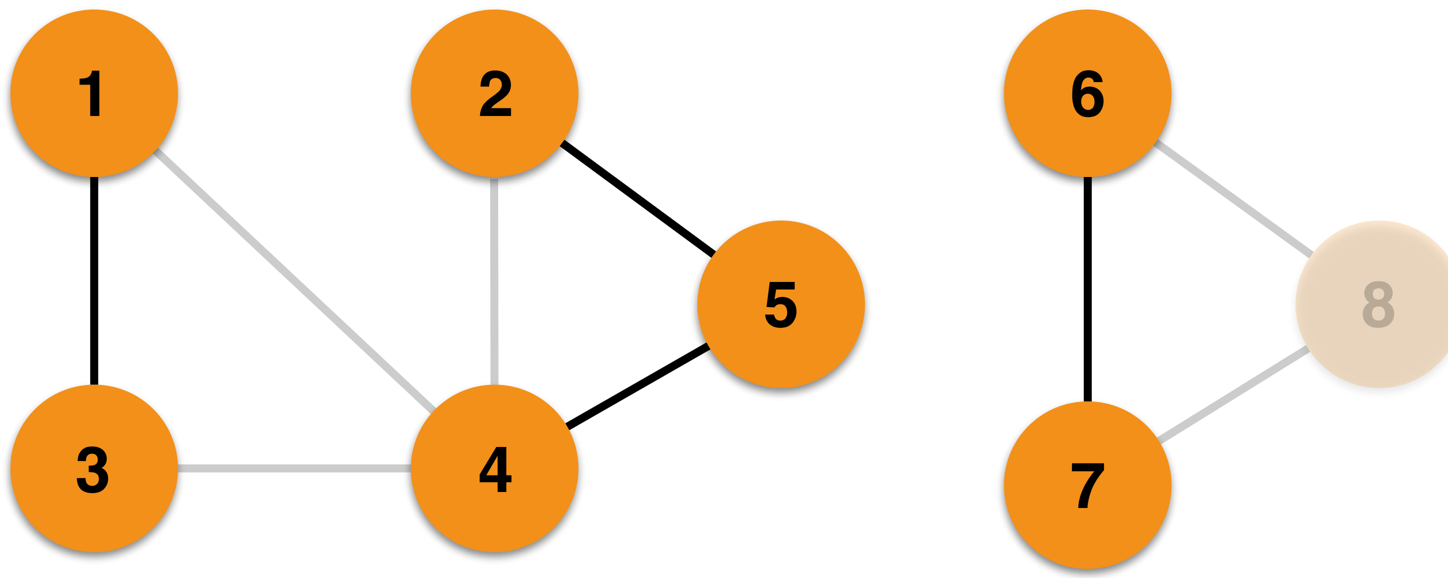


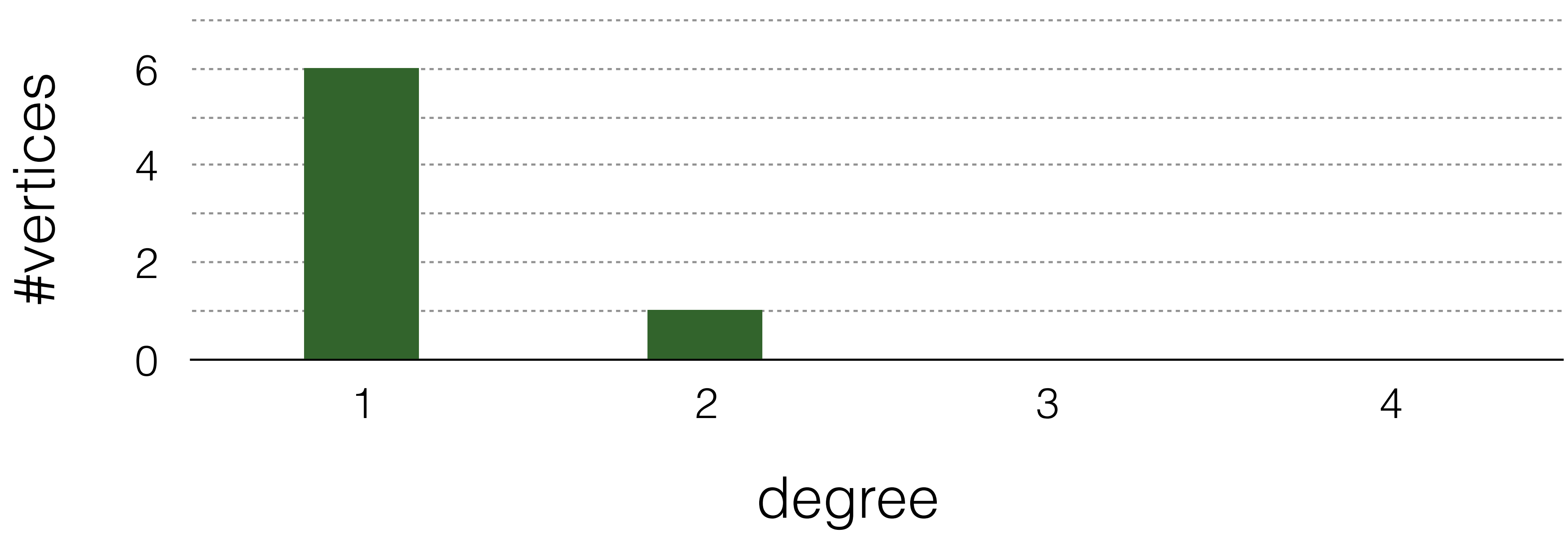
Streaming Degrees Distribution



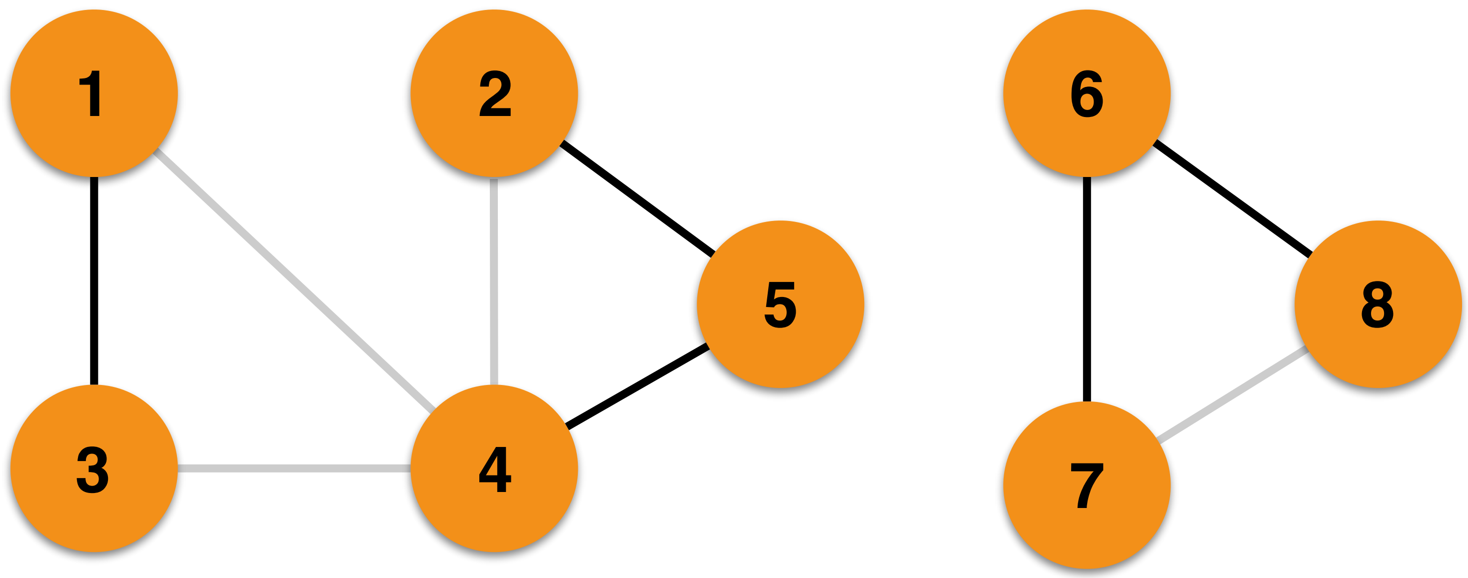


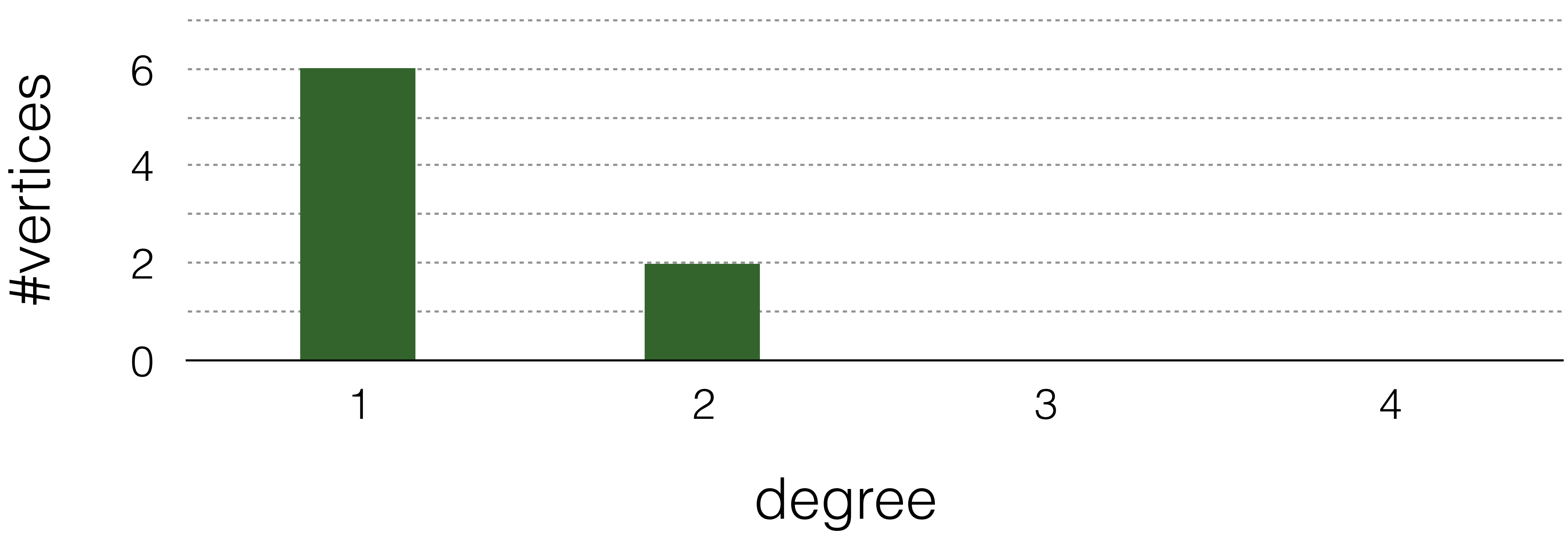
Streaming Degrees Distribution



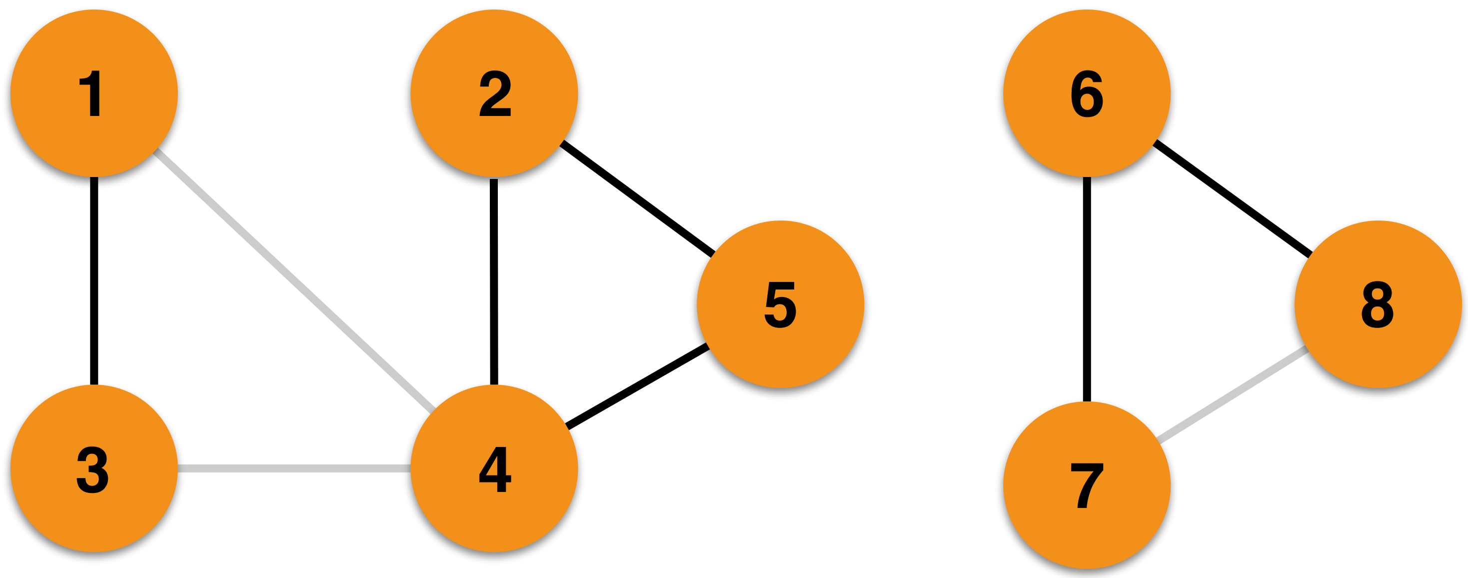


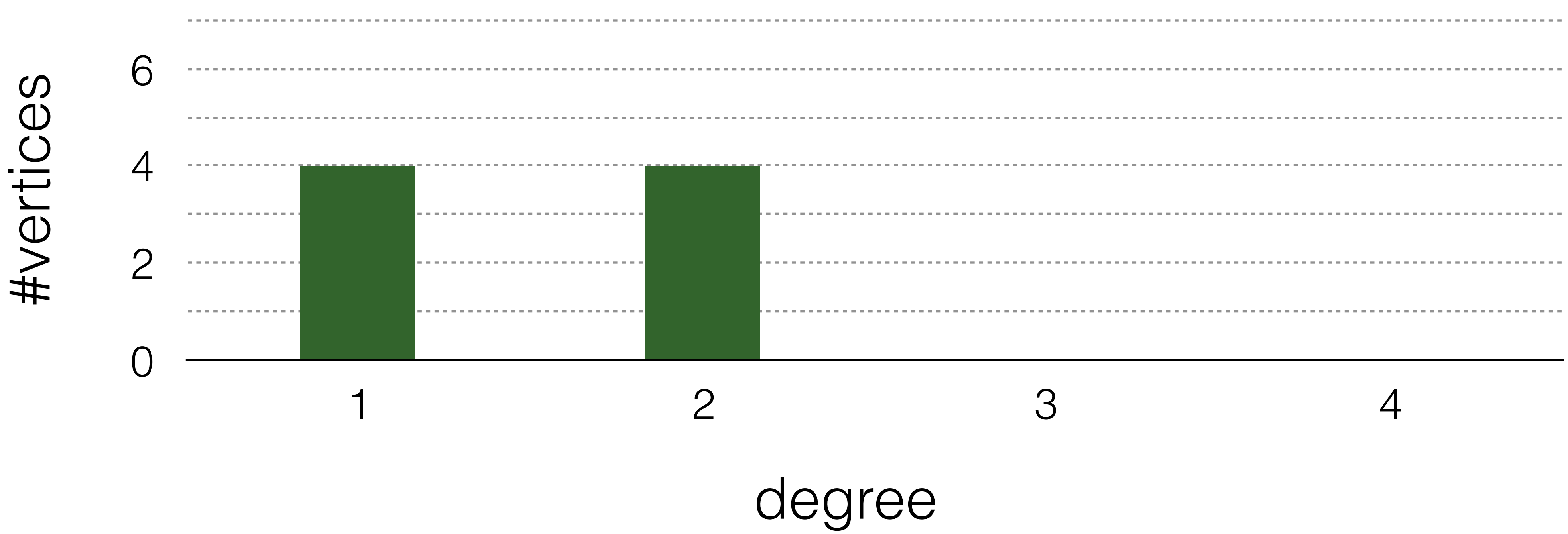
Streaming Degrees Distribution



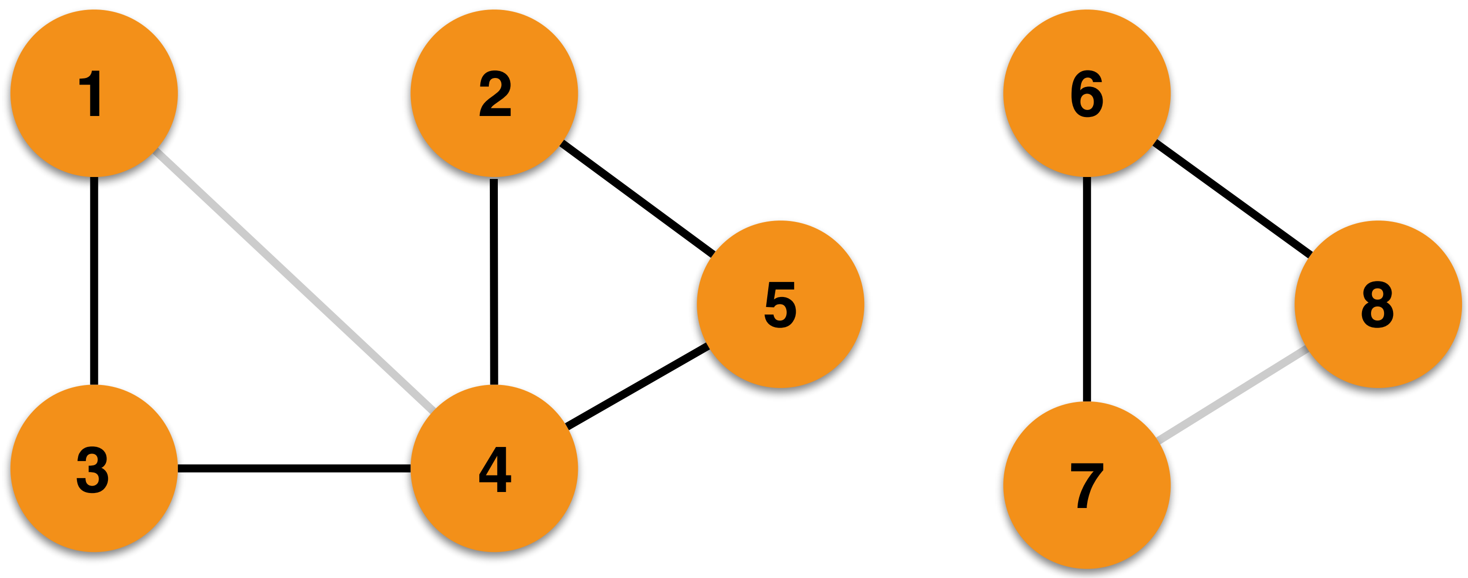


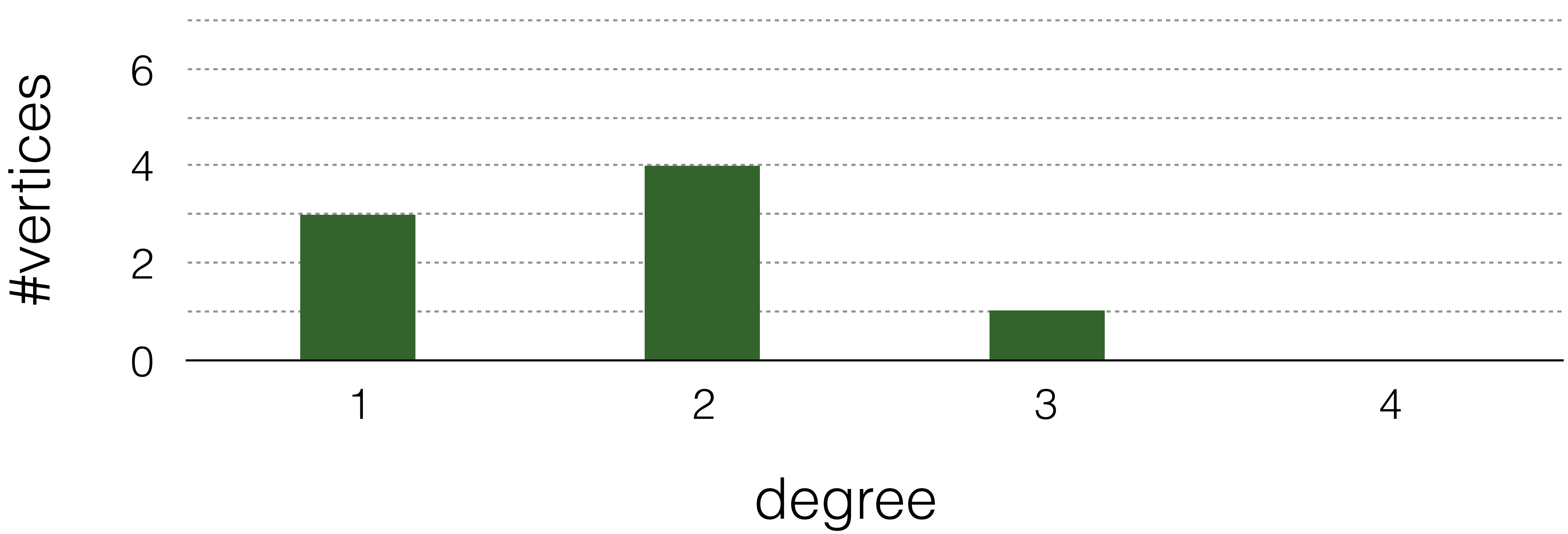
Streaming Degrees Distribution



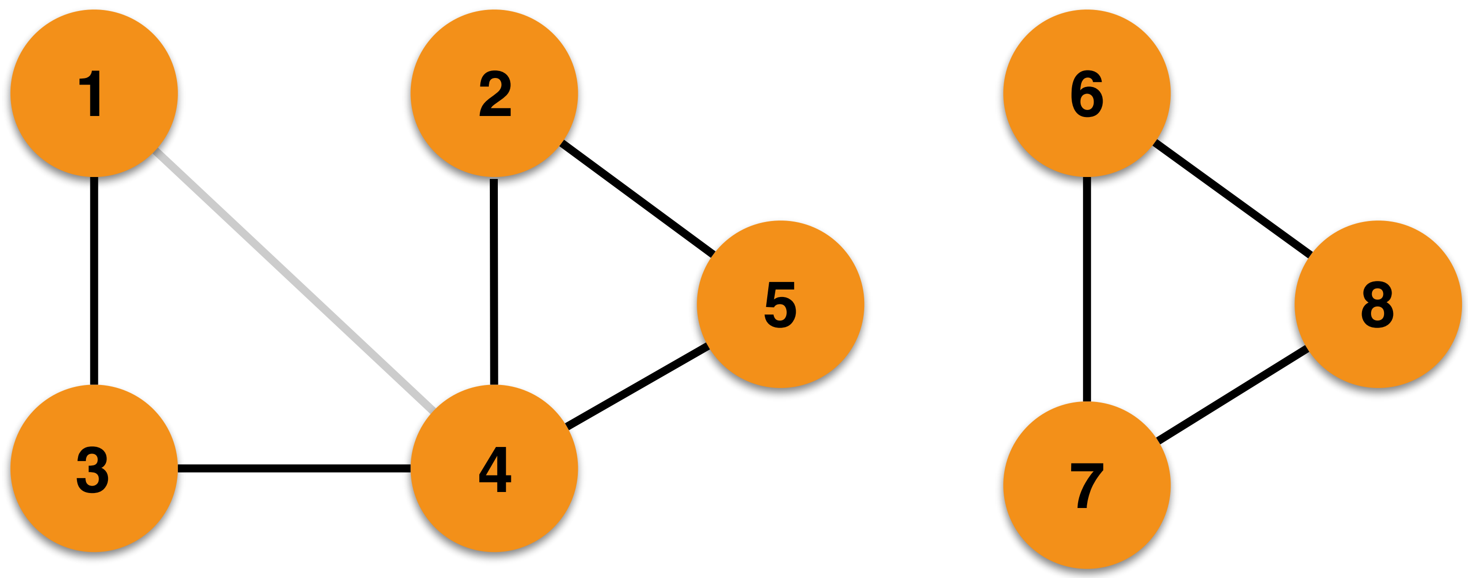


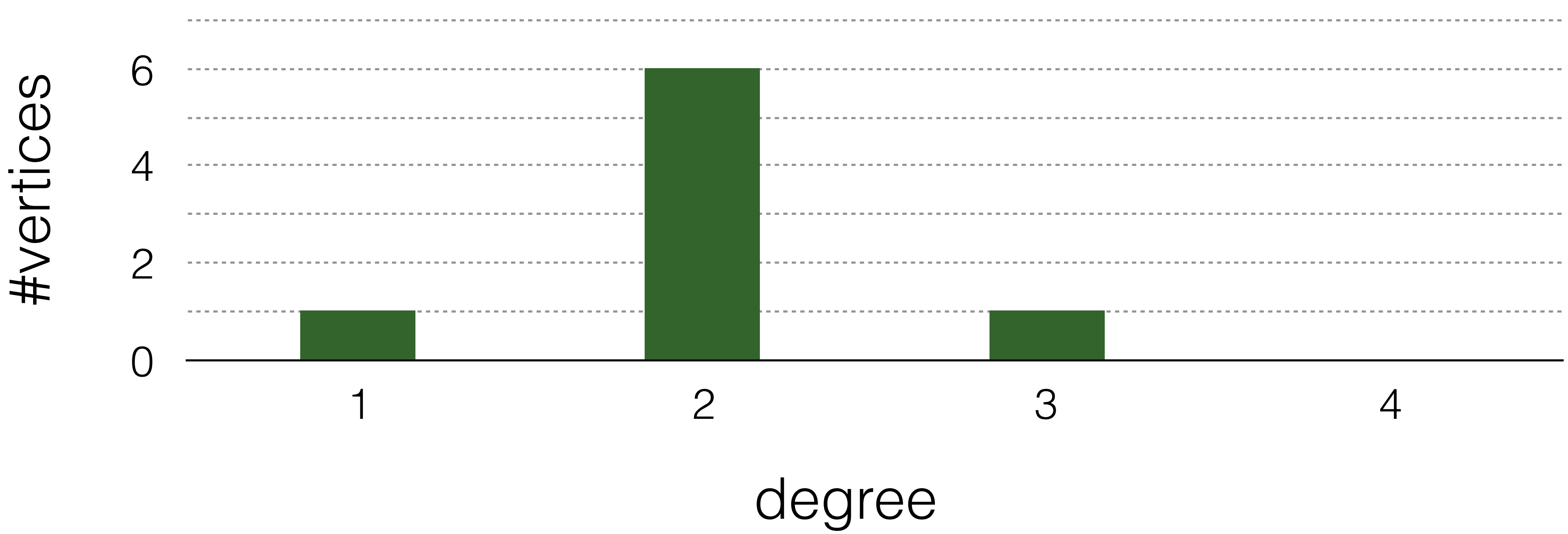
Streaming Degrees Distribution



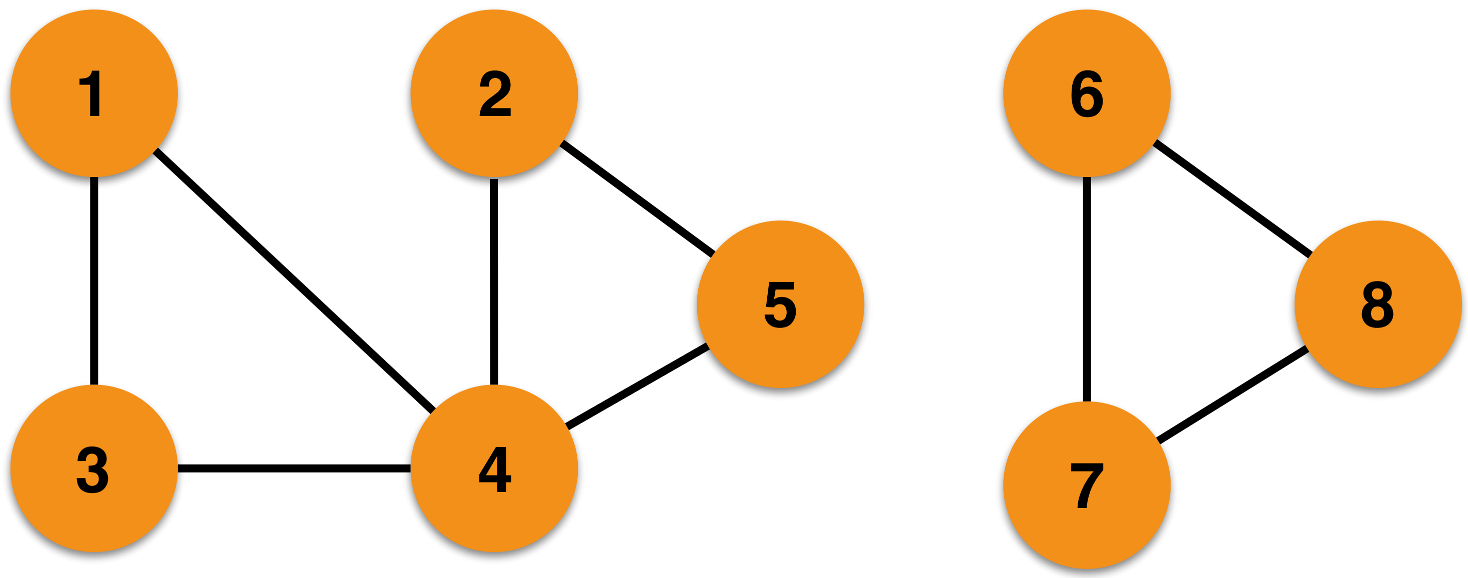


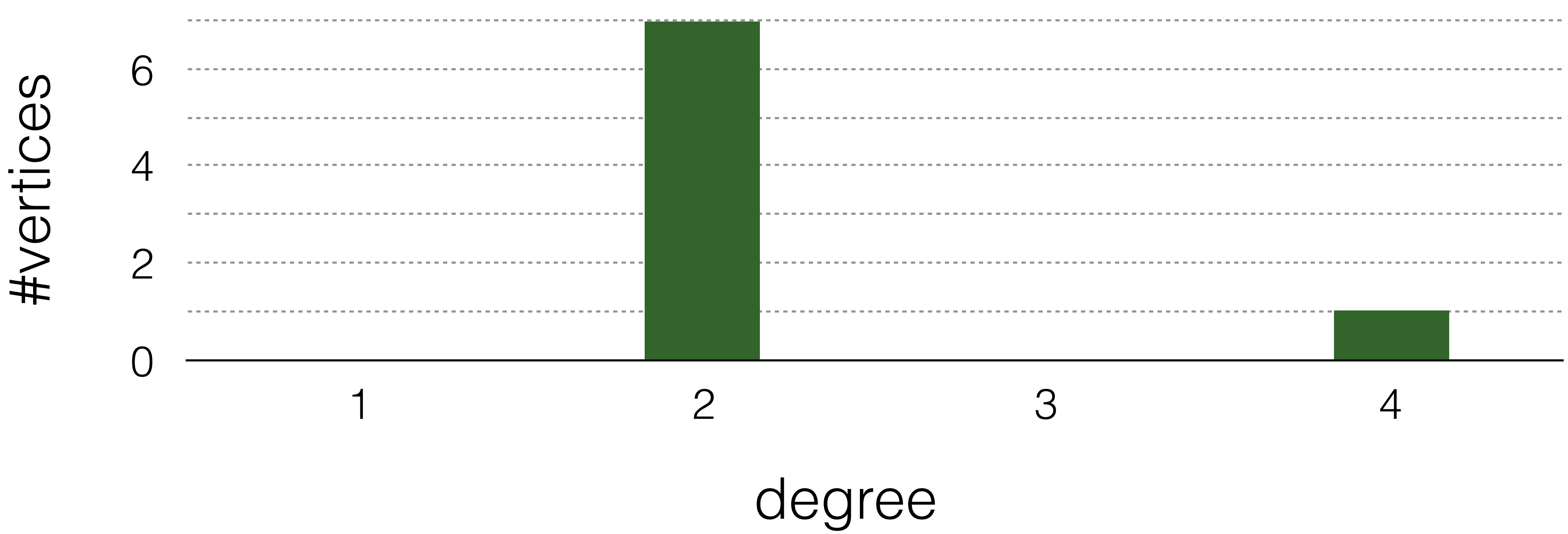
Streaming Degrees Distribution





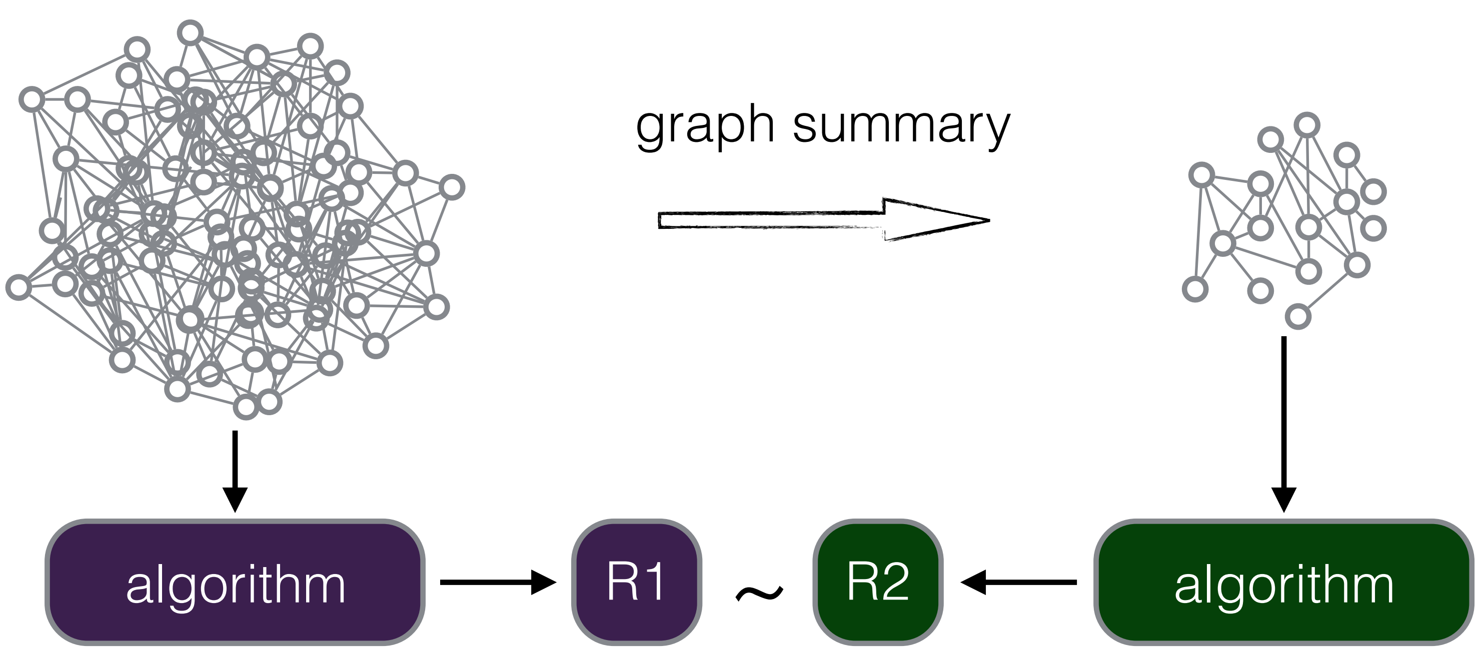
Streaming Degrees Distribution





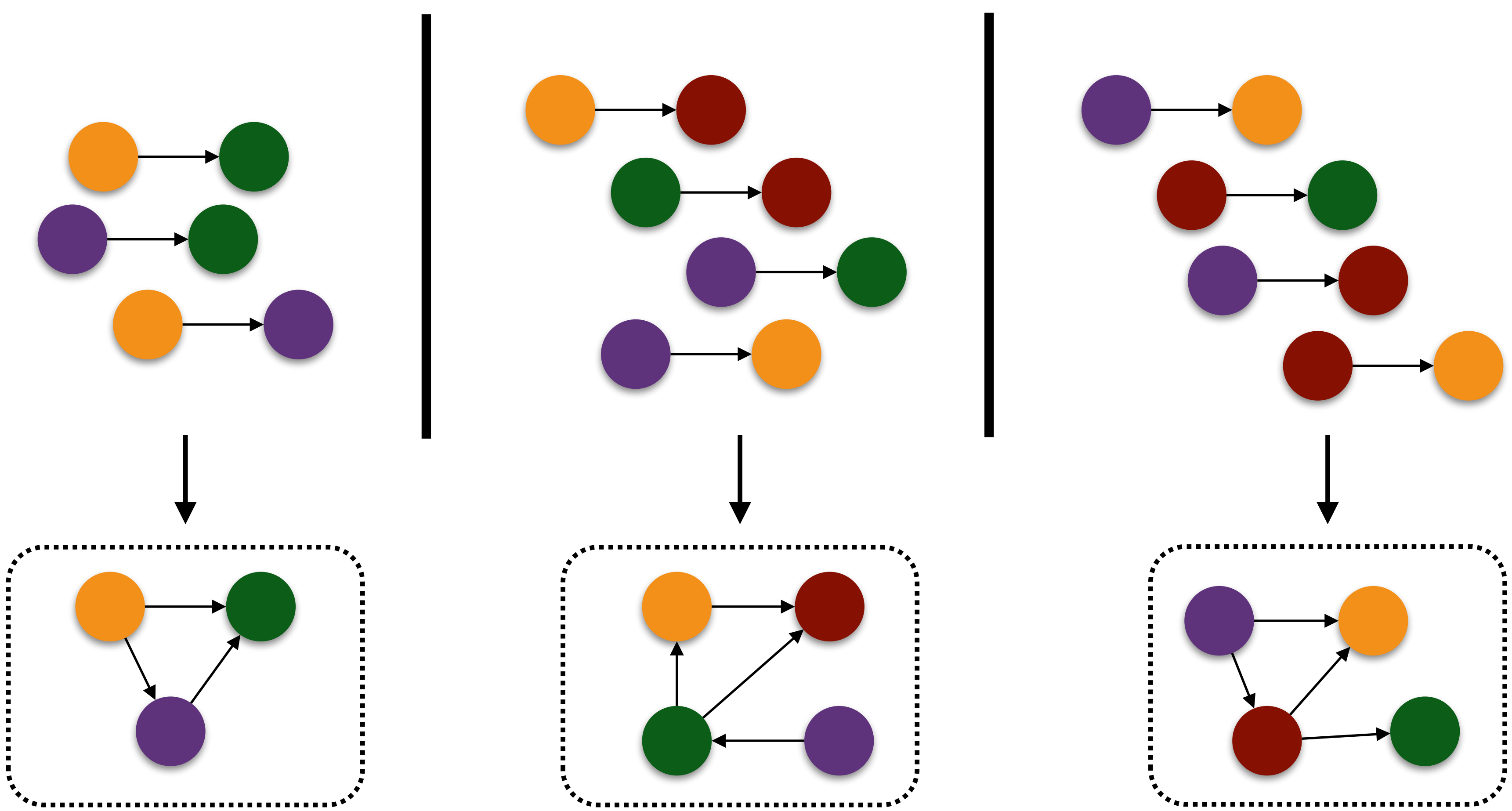
Graph Summaries

- **spanners** for distance estimation
- **sparsifiers** for cut estimation
- **sketches** for homomorphic properties



Window Aggregations

Neighborhood aggregations on windows



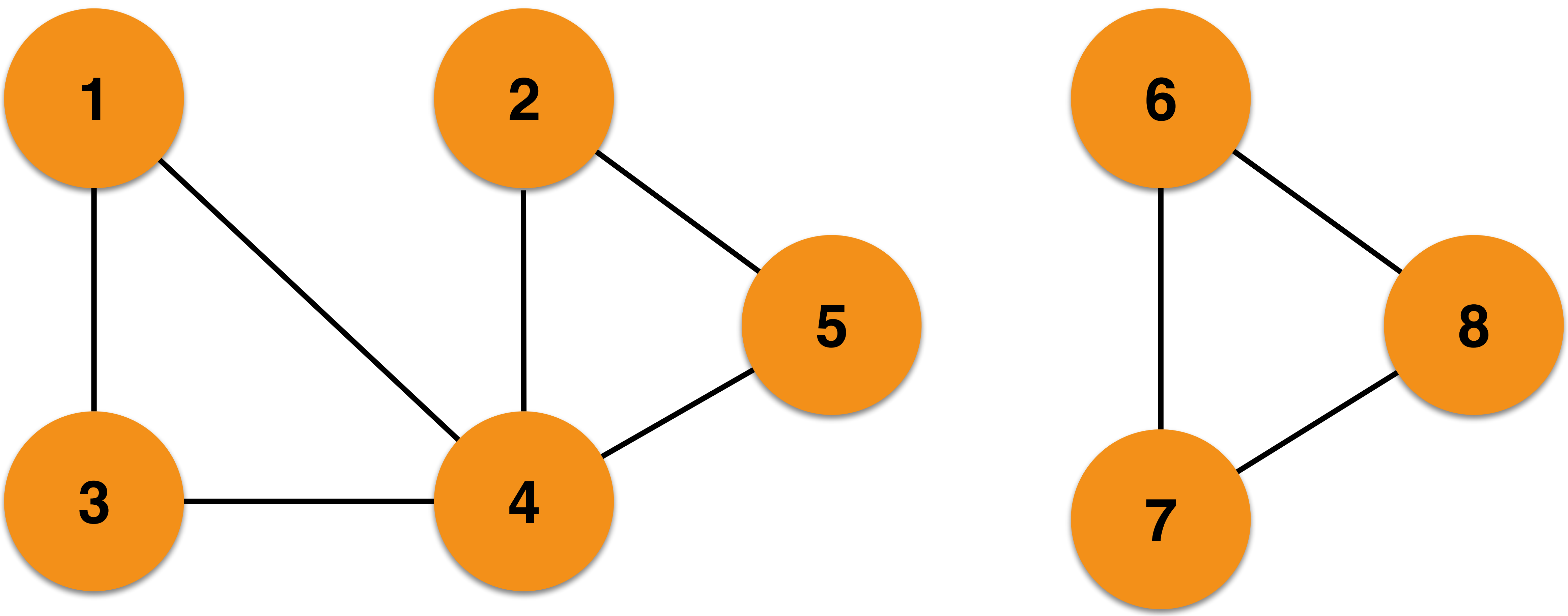
Examples

Batch Connected Components

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

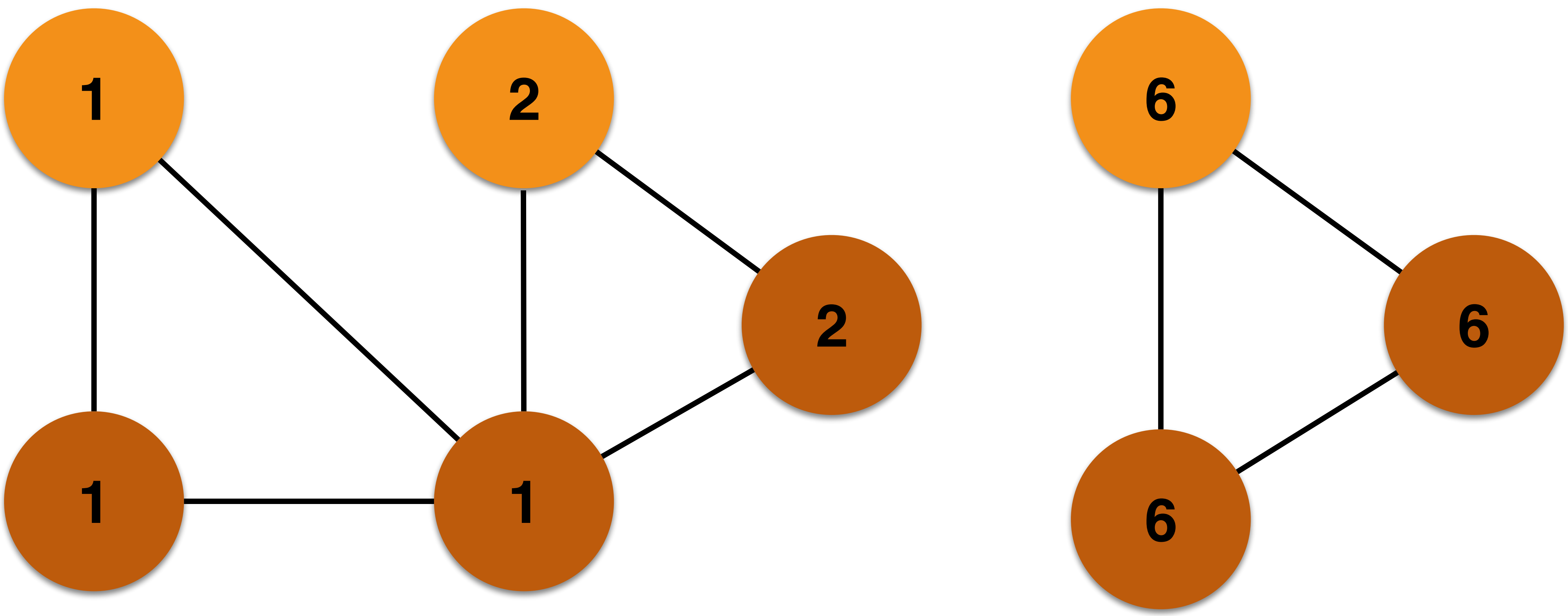
Batch Connected Components

i=0



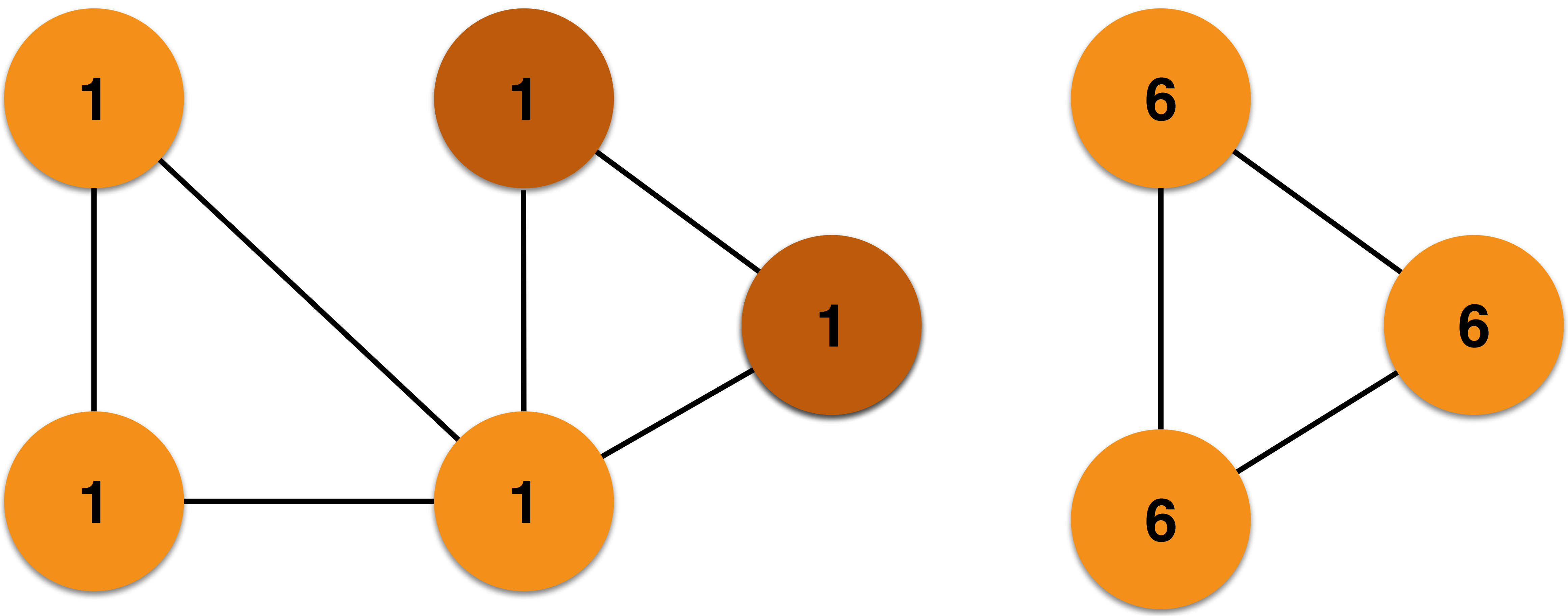
Batch Connected Components

i=1



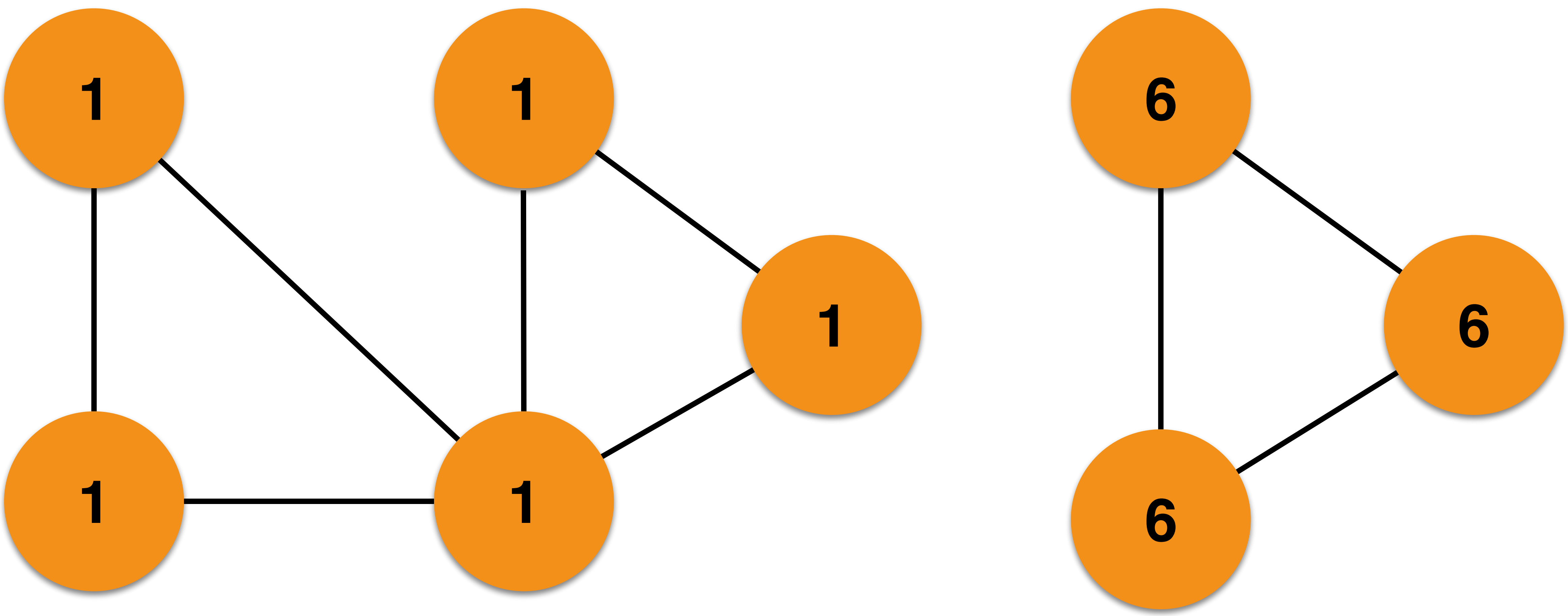
Batch Connected Components

i=2



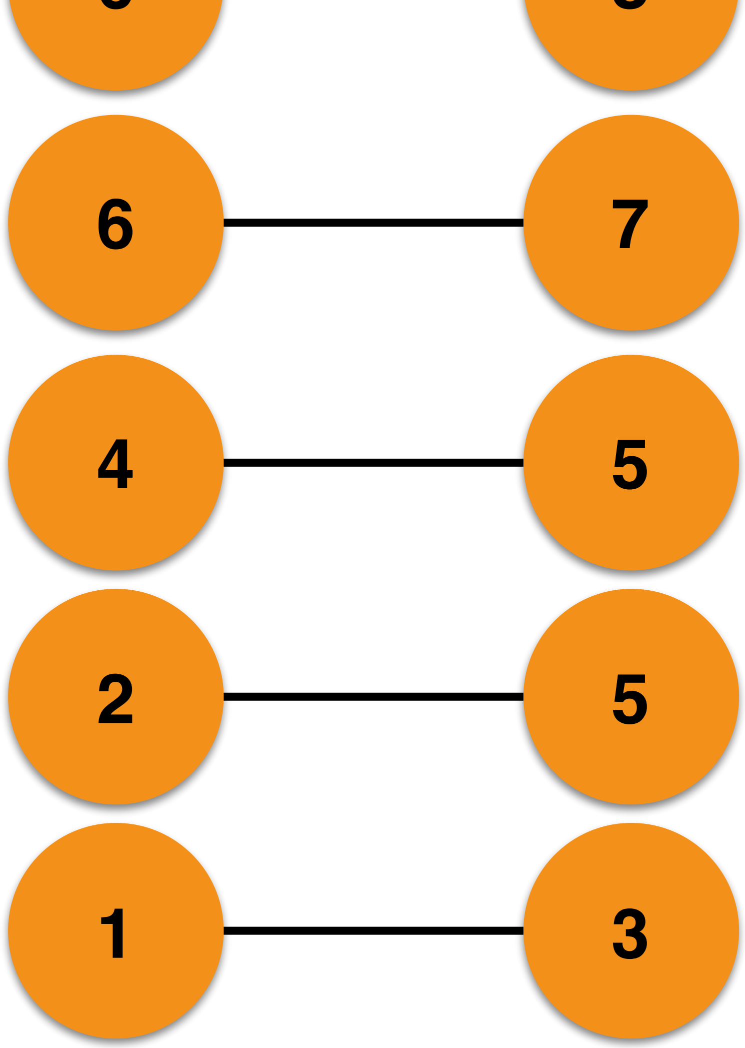
Batch Connected Components

i=3

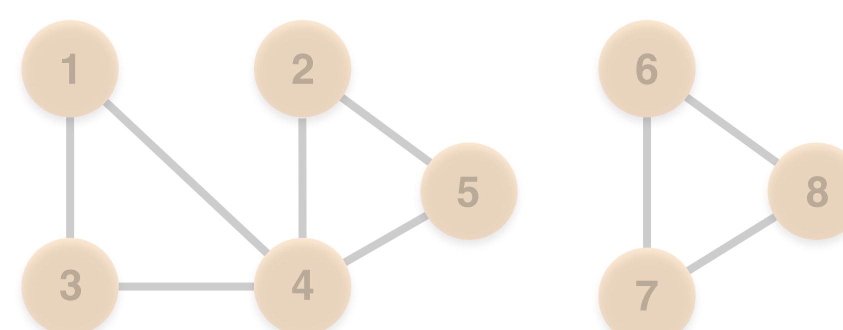


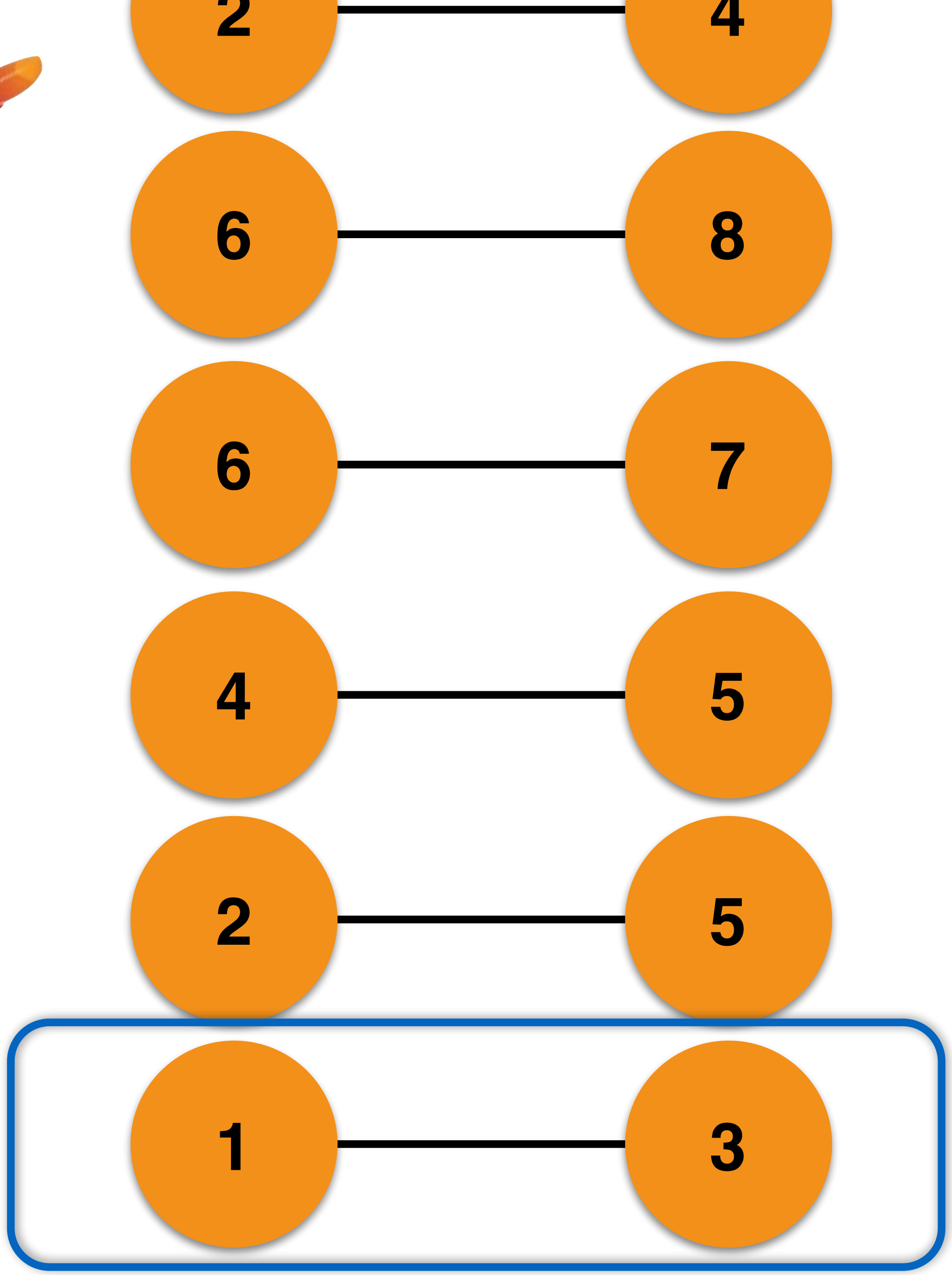
Stream Connected Components

- **State:** a *disjoint set* data structure for the components
- **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

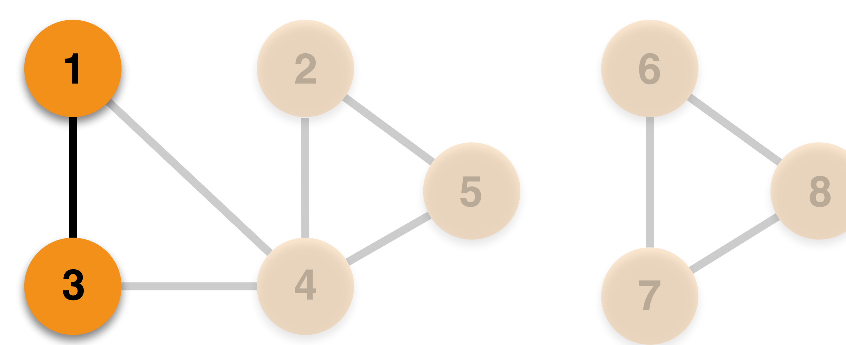


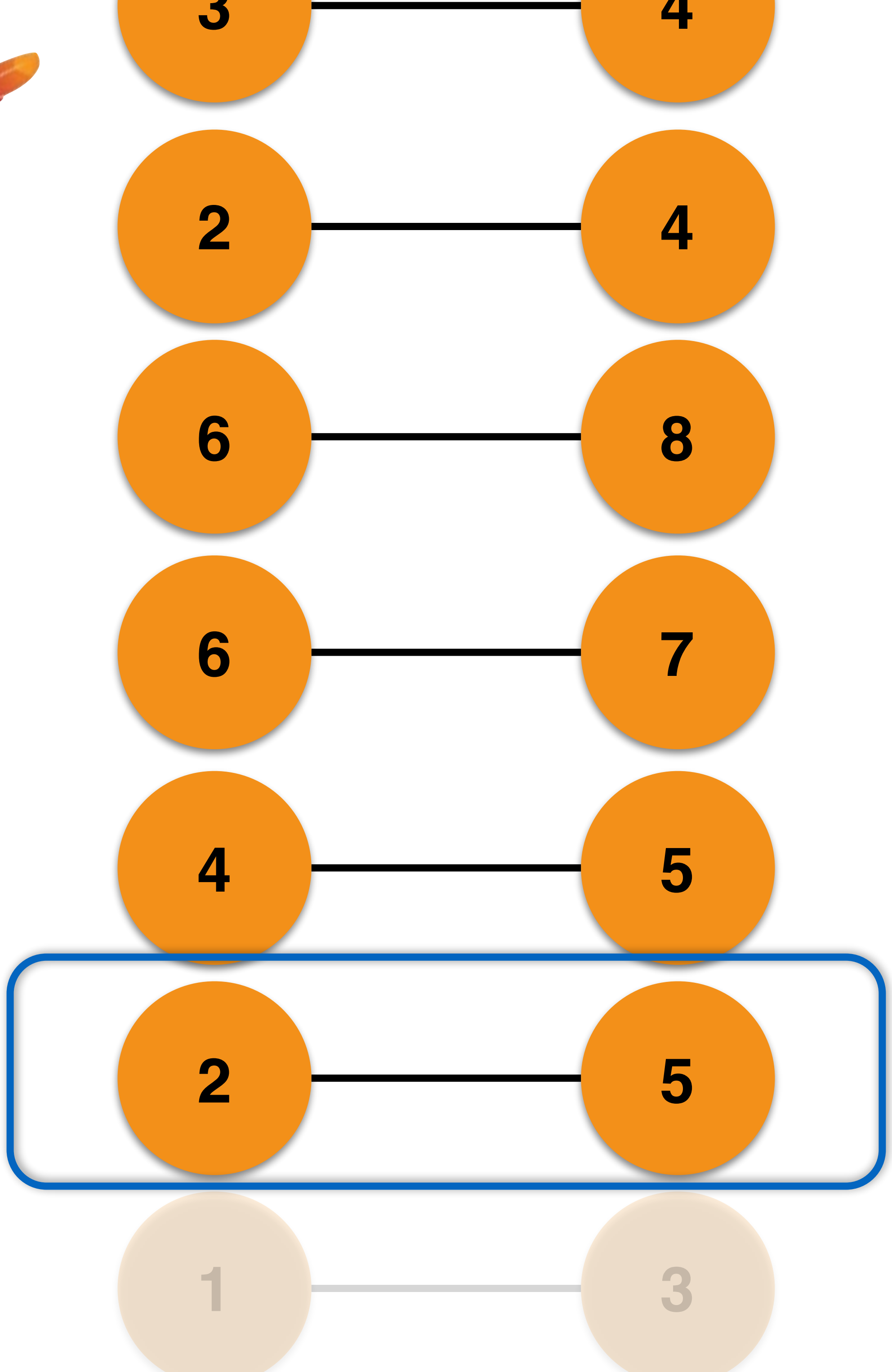
ComponentID	Vertices



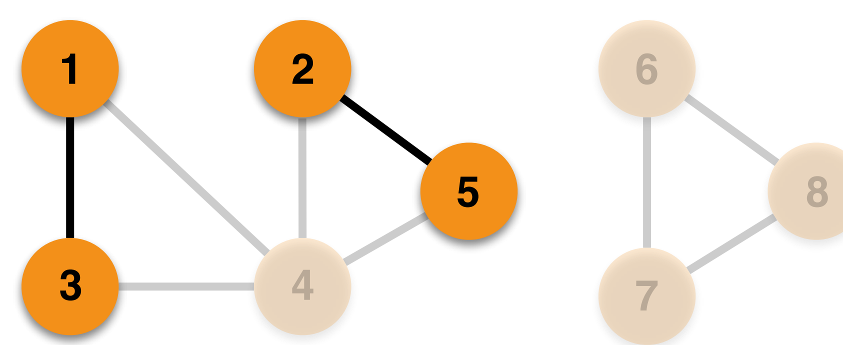


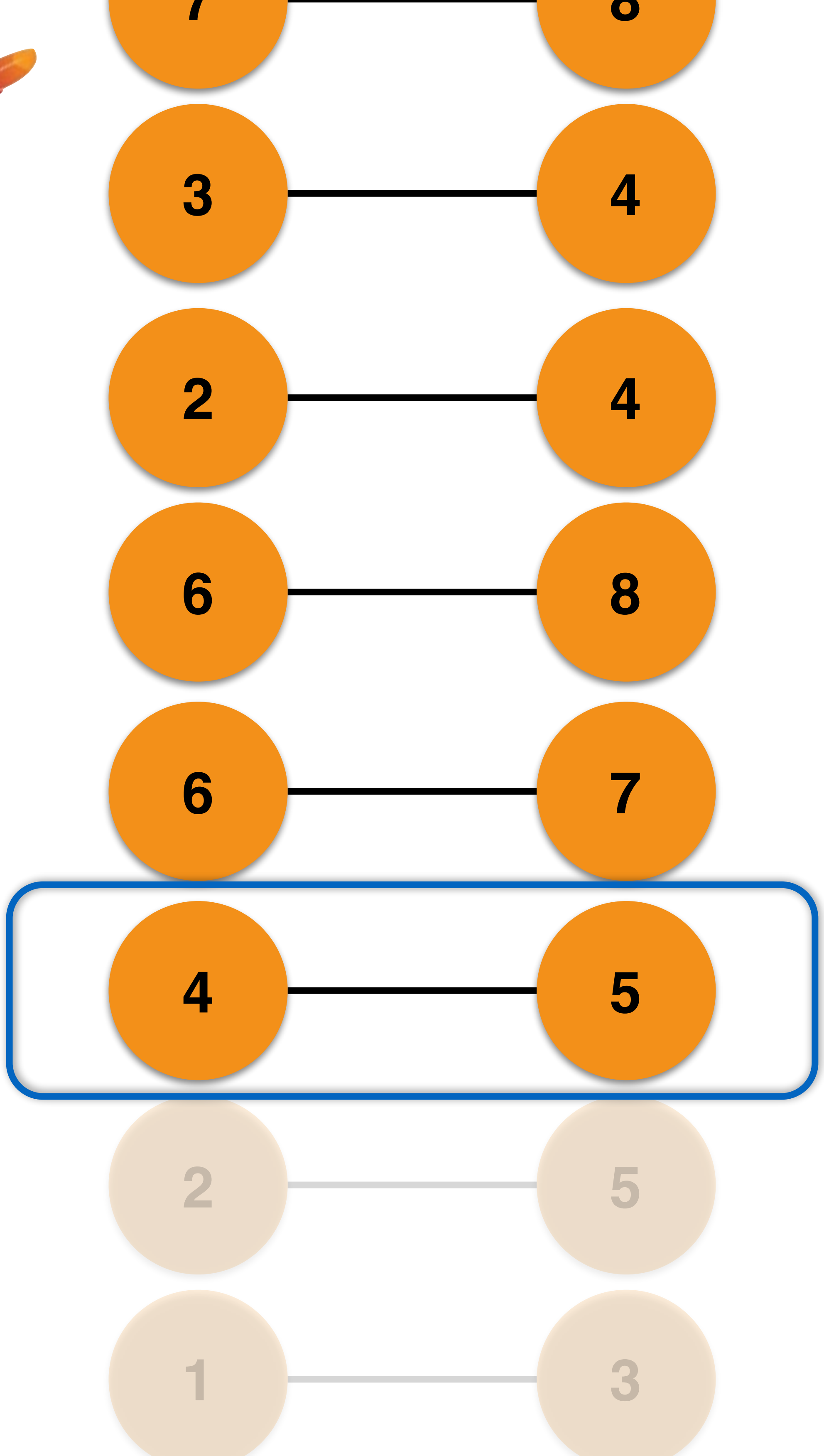
ComponentID	Vertices
1	1, 3



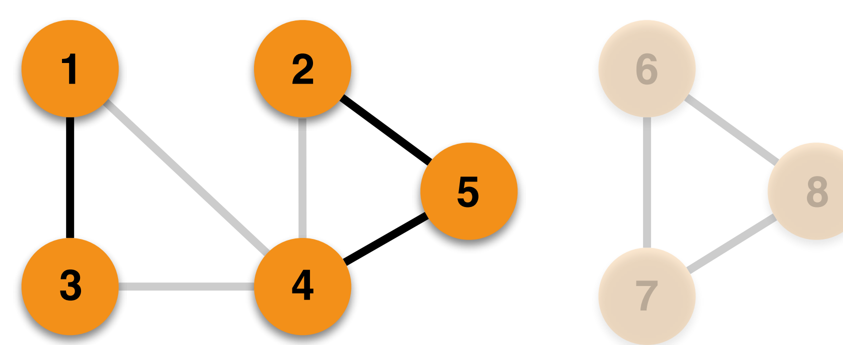


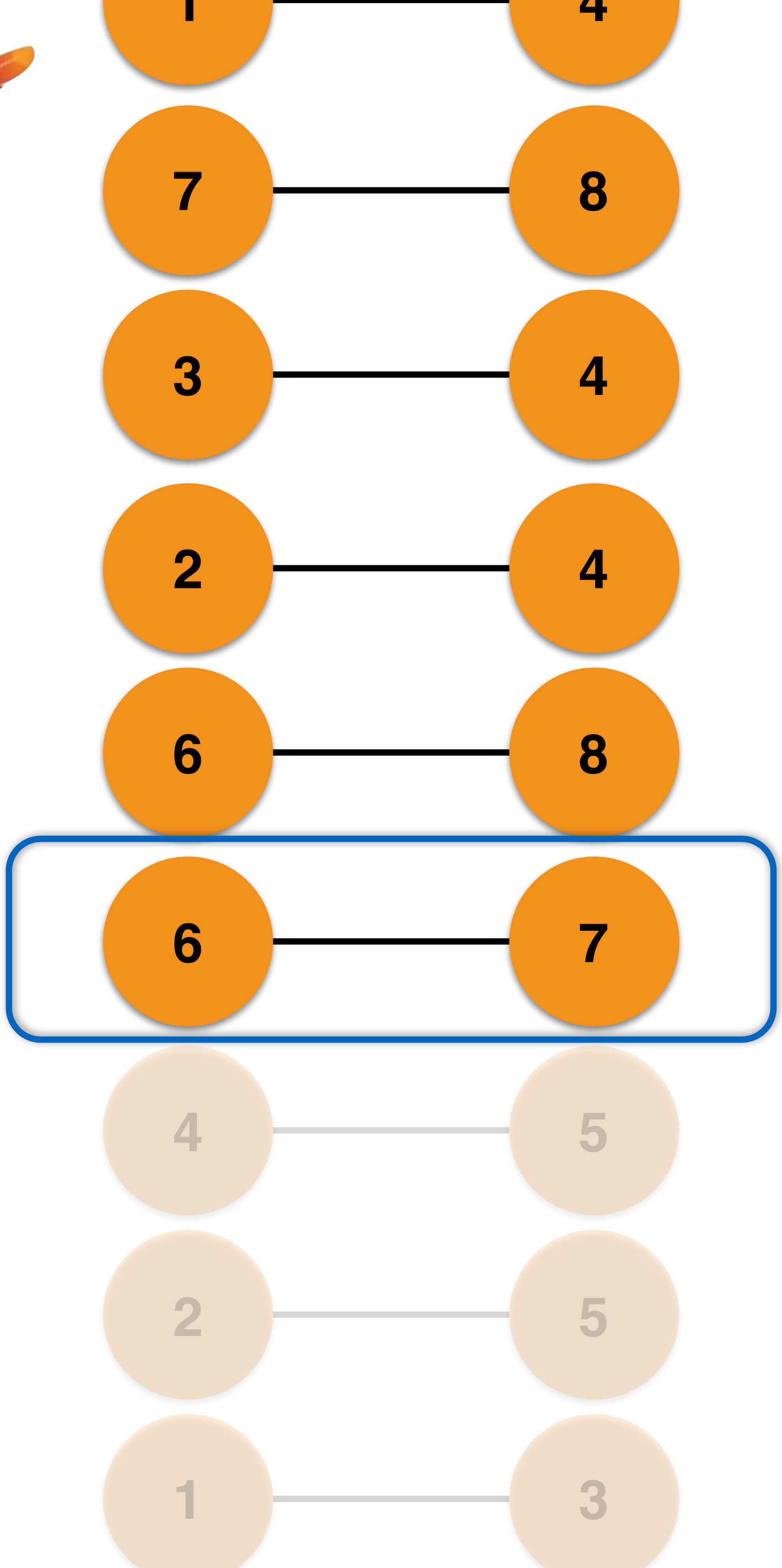
ComponentID	Vertices
1	1, 3
2	2, 5



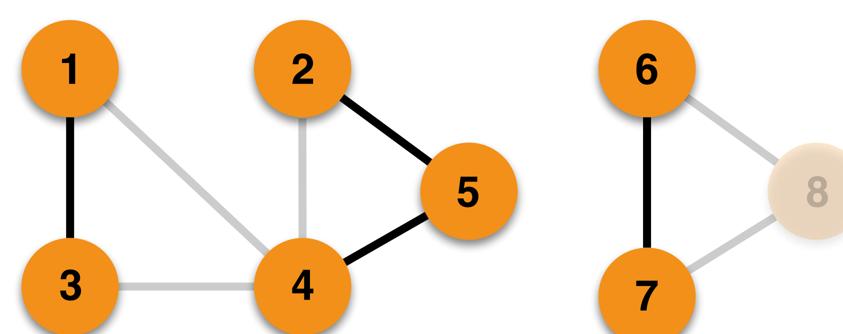


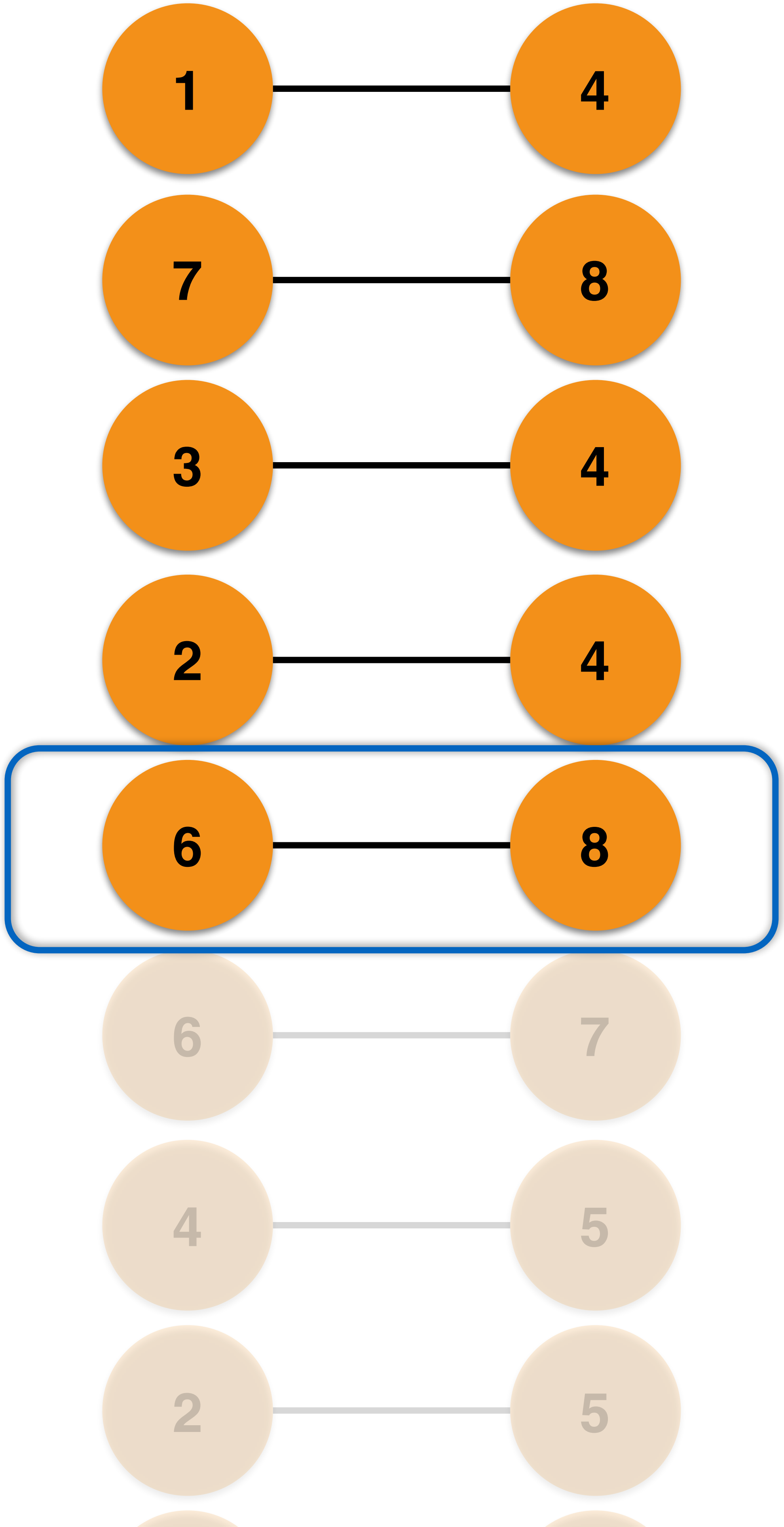
ComponentID	Vertices
1	1, 3
2	2, 4, 5



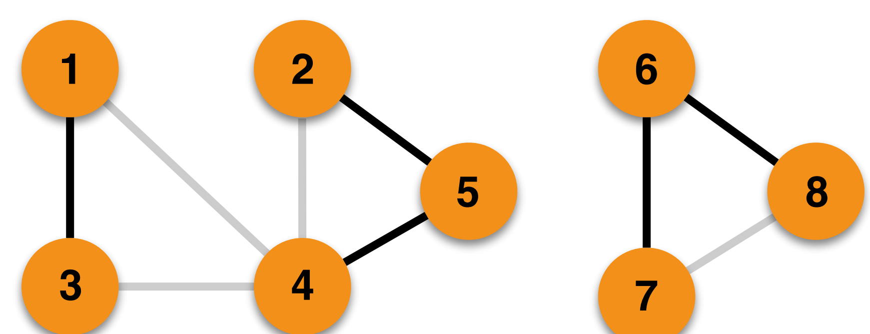


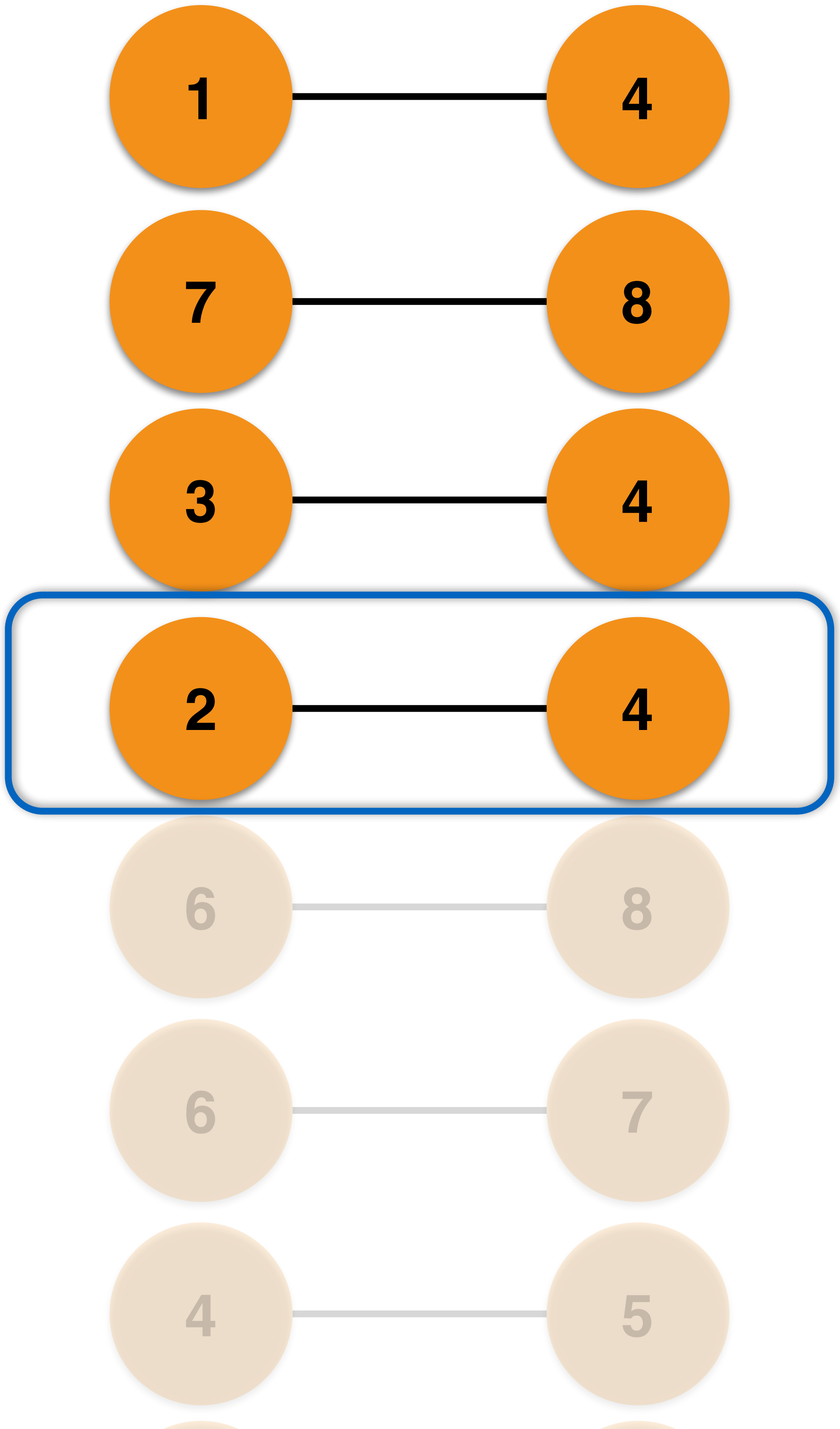
ComponentID	Vertices
1	1, 3
2	2, 4, 5
6	6, 7



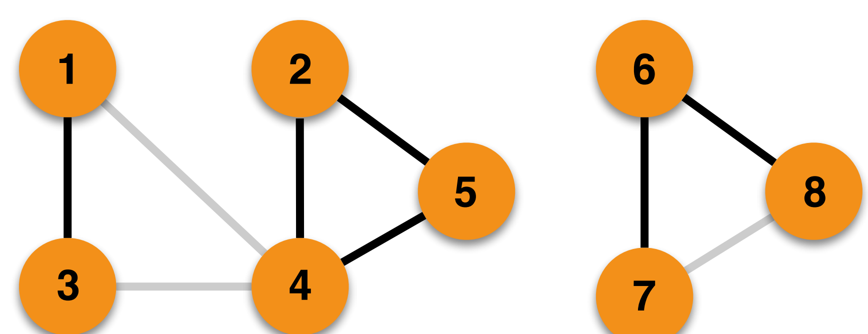


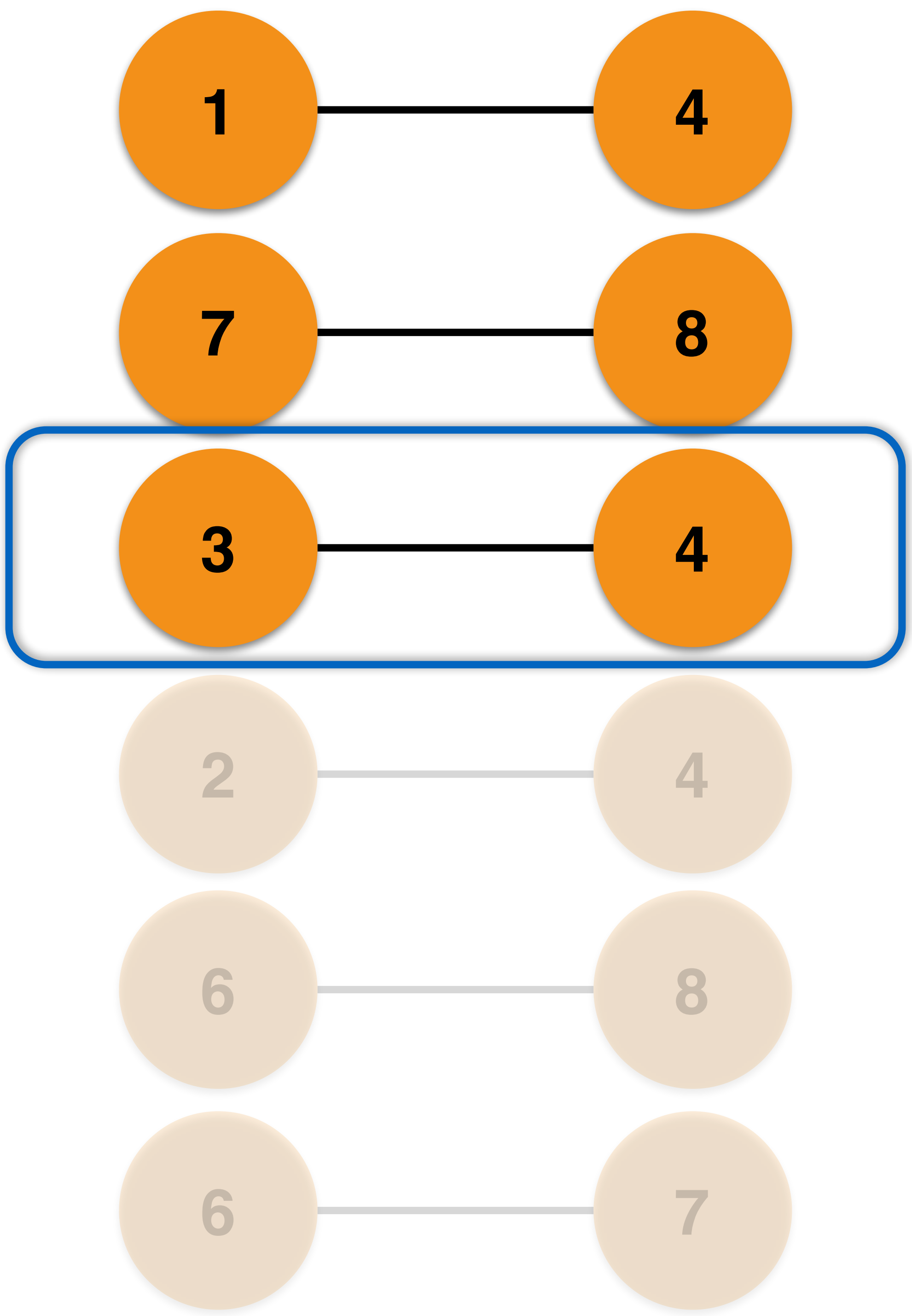
ComponentID	Vertices
1	1, 3
2	2, 4, 5
6	6, 7, 8



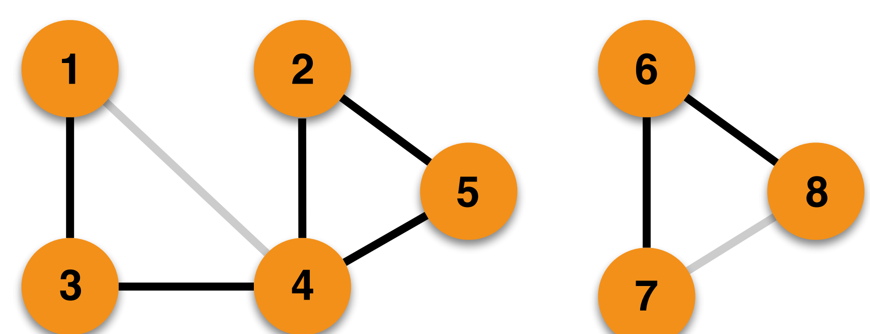


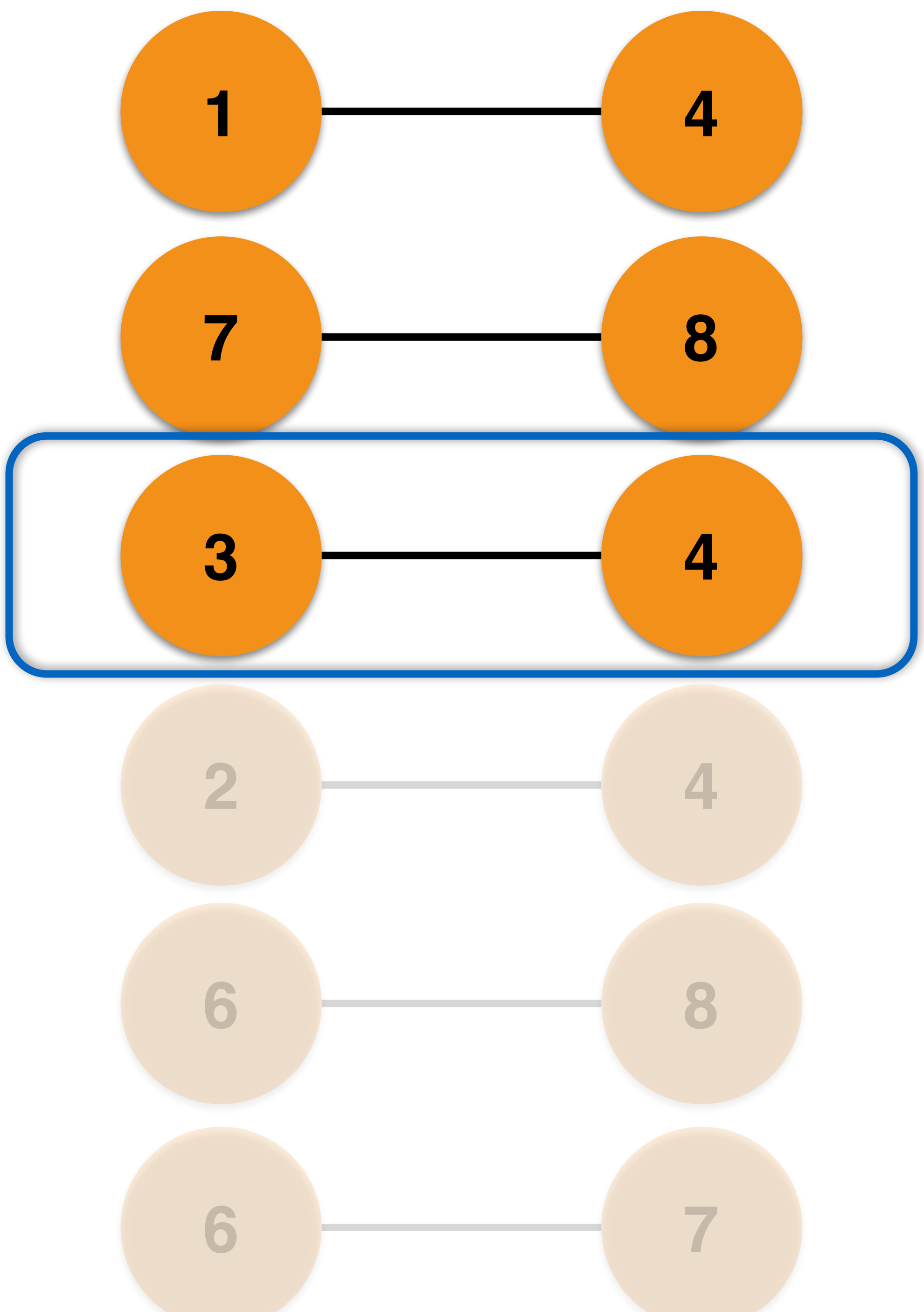
ComponentID	Vertices
1	1, 3
2	2, 4, 5
6	6, 7, 8



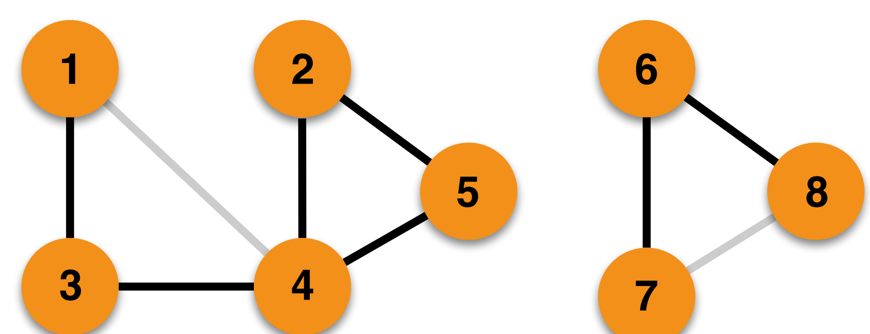


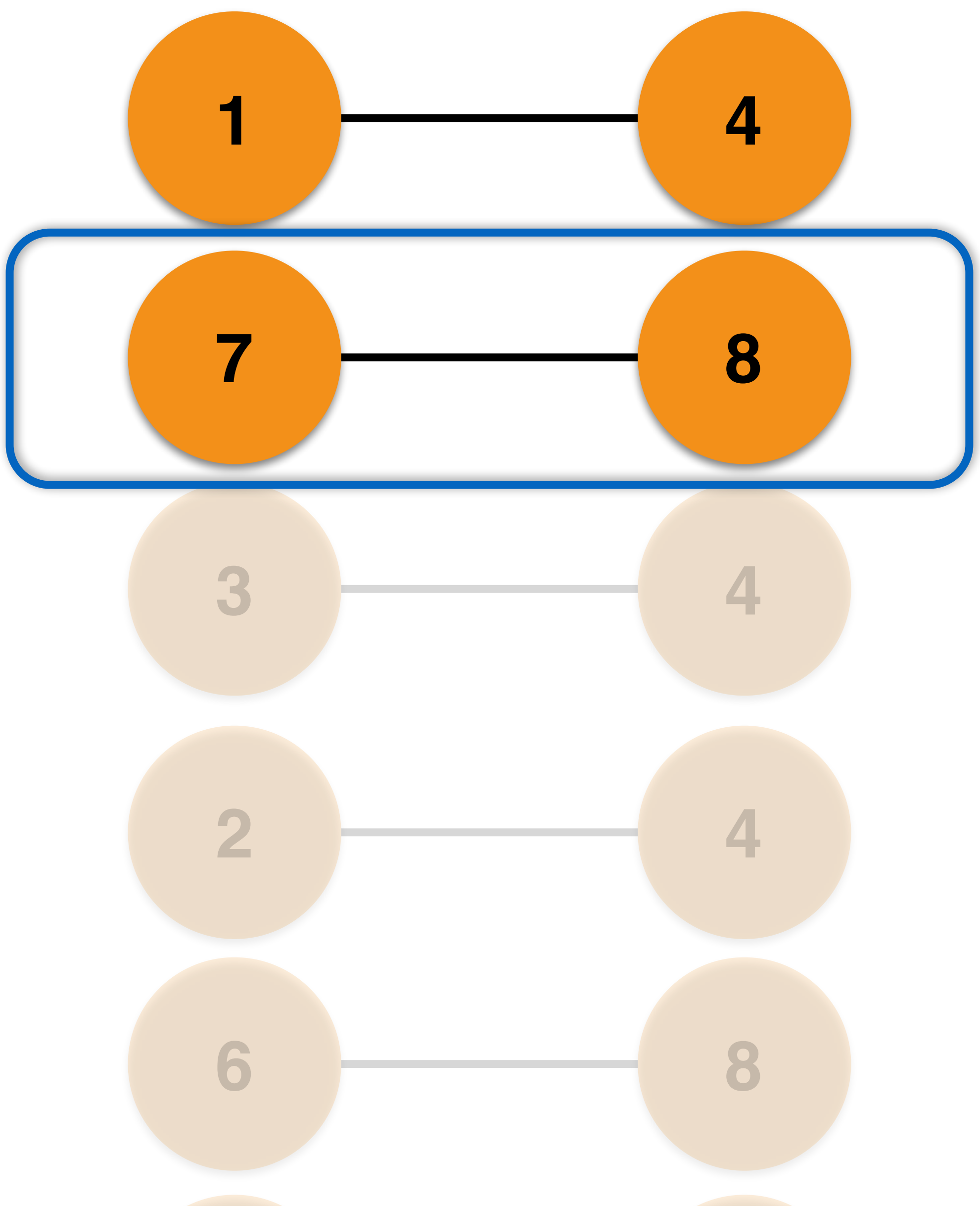
ComponentID	Vertices
1	1, **3**
2	2, **4**, 5
6	6, 7, 8



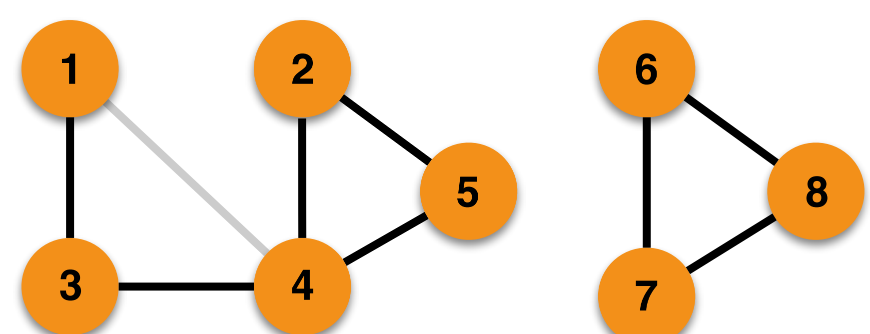


ComponentID	Vertices
1	1, 2, 3, 4, 5
6	6, 7, 8

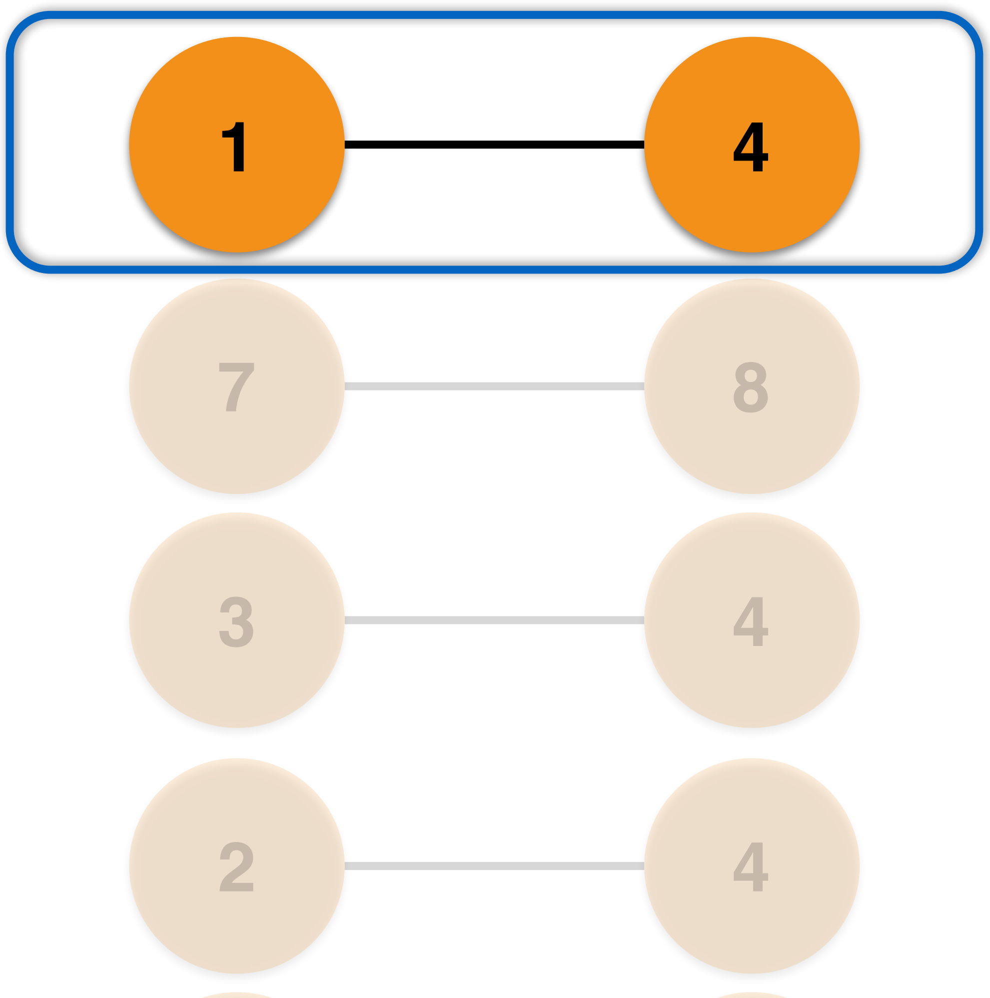


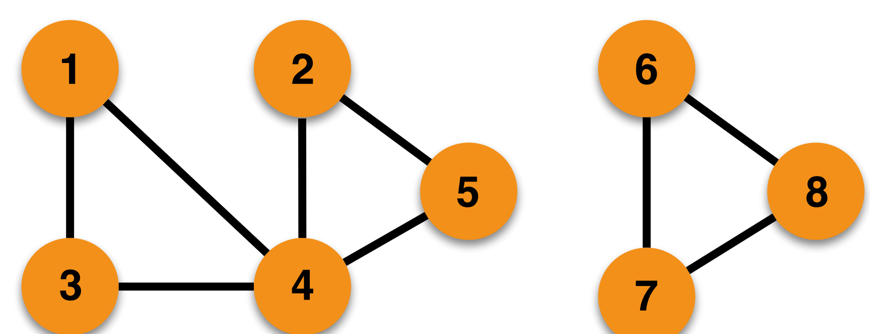


ComponentID	Vertices
1	1, 2, 3, 4, 5
6	6, 7, 8

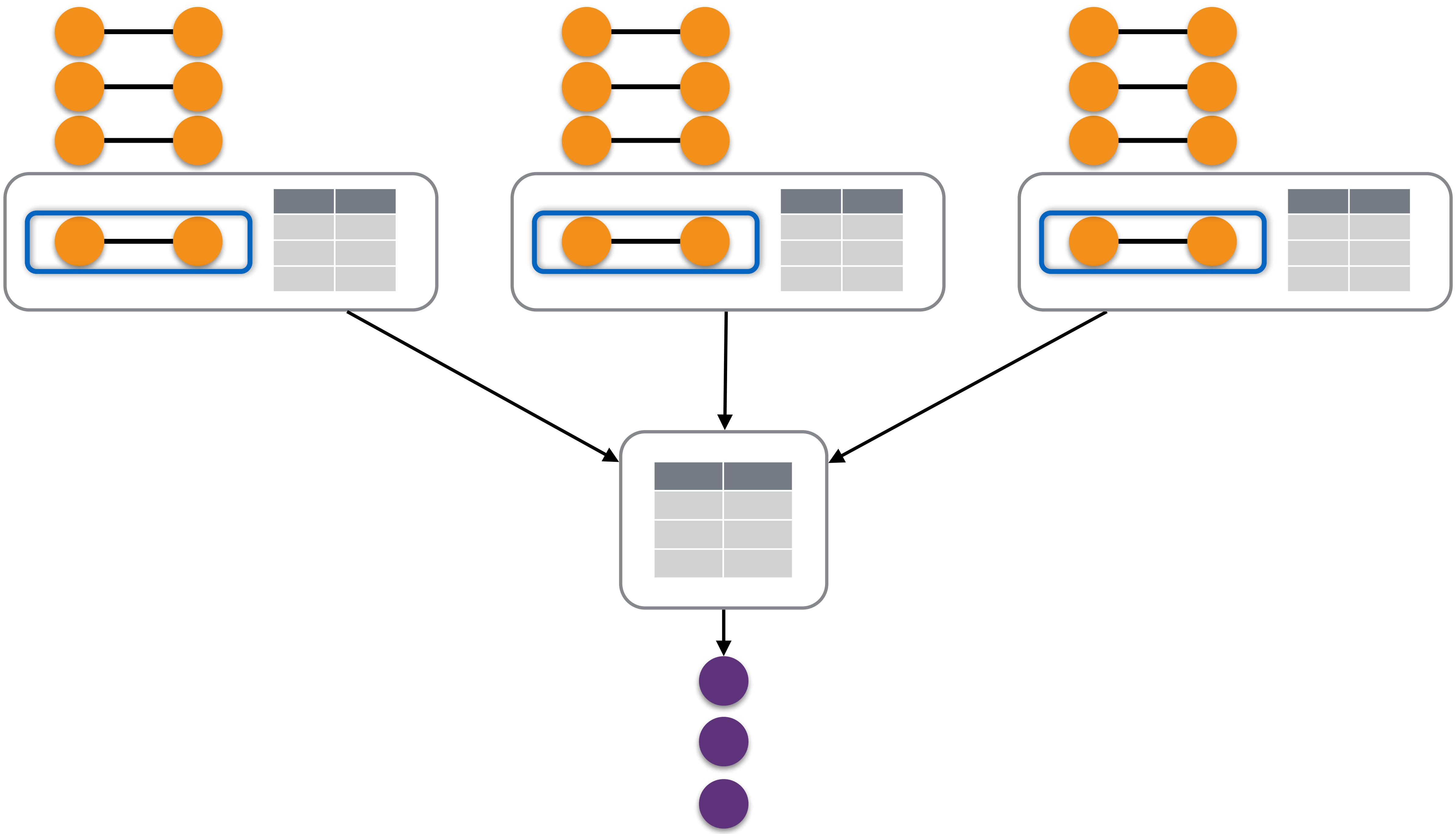


ComponentID	Vertices
1	1, 2, 3, 4, 5
6	6, 7, 8





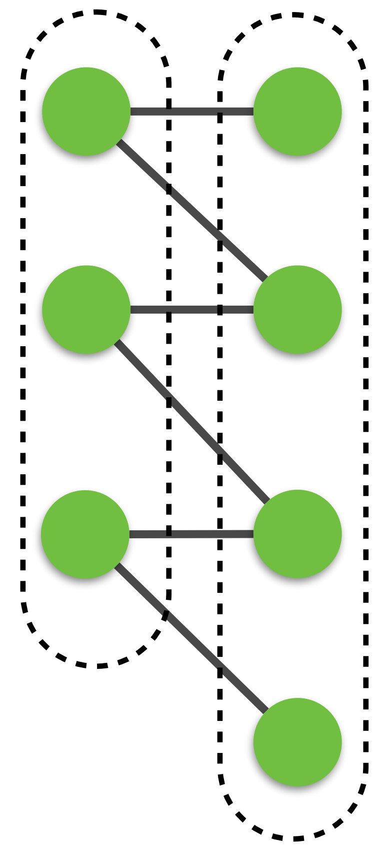
Distributed Stream Connected Components



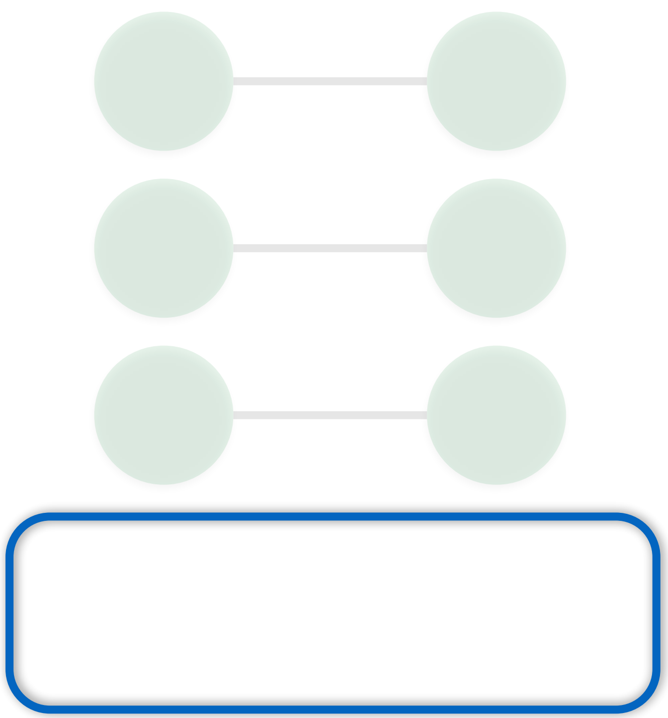
Stream Bipartite Detection

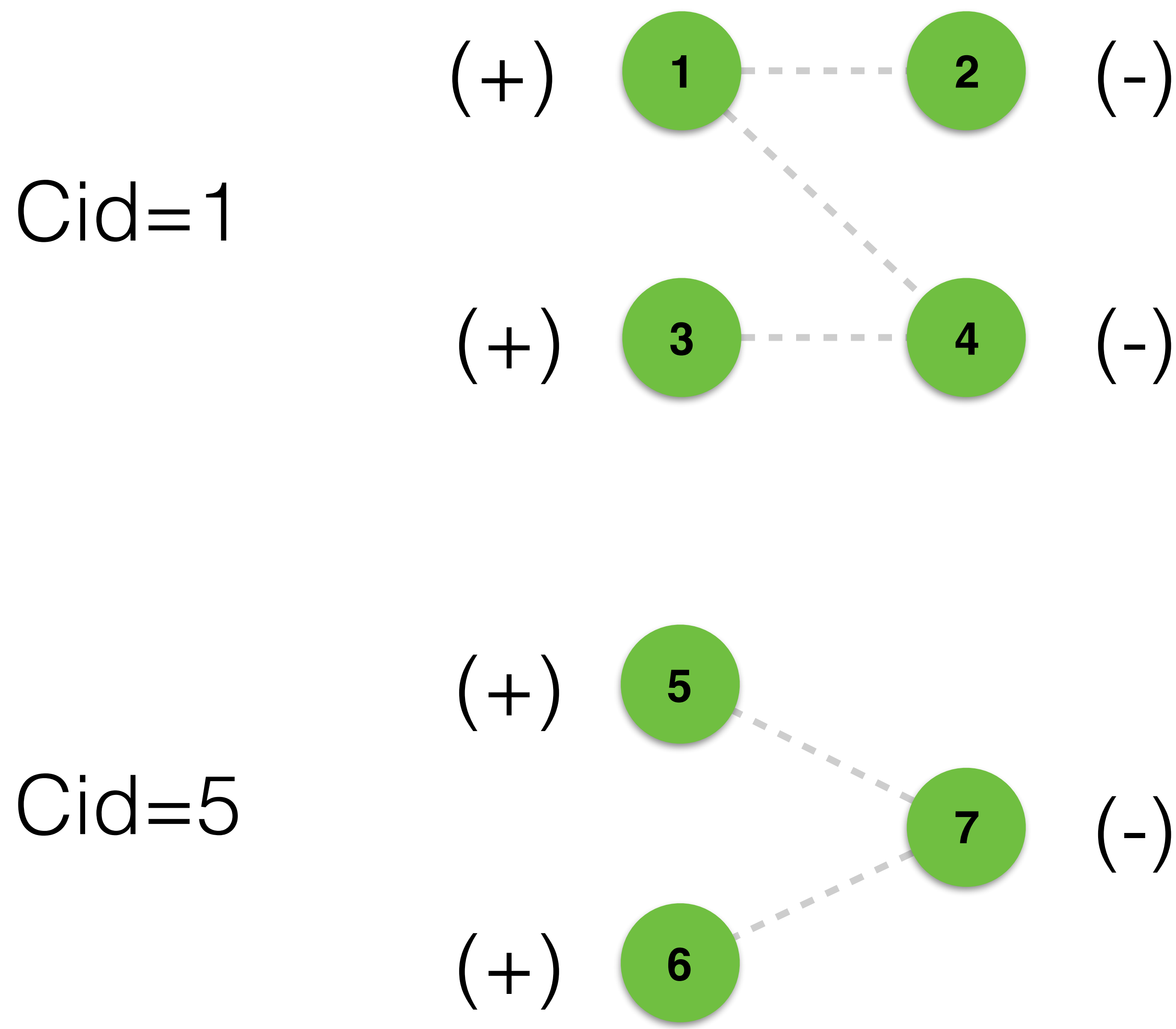
Similar to connected components, but

- 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

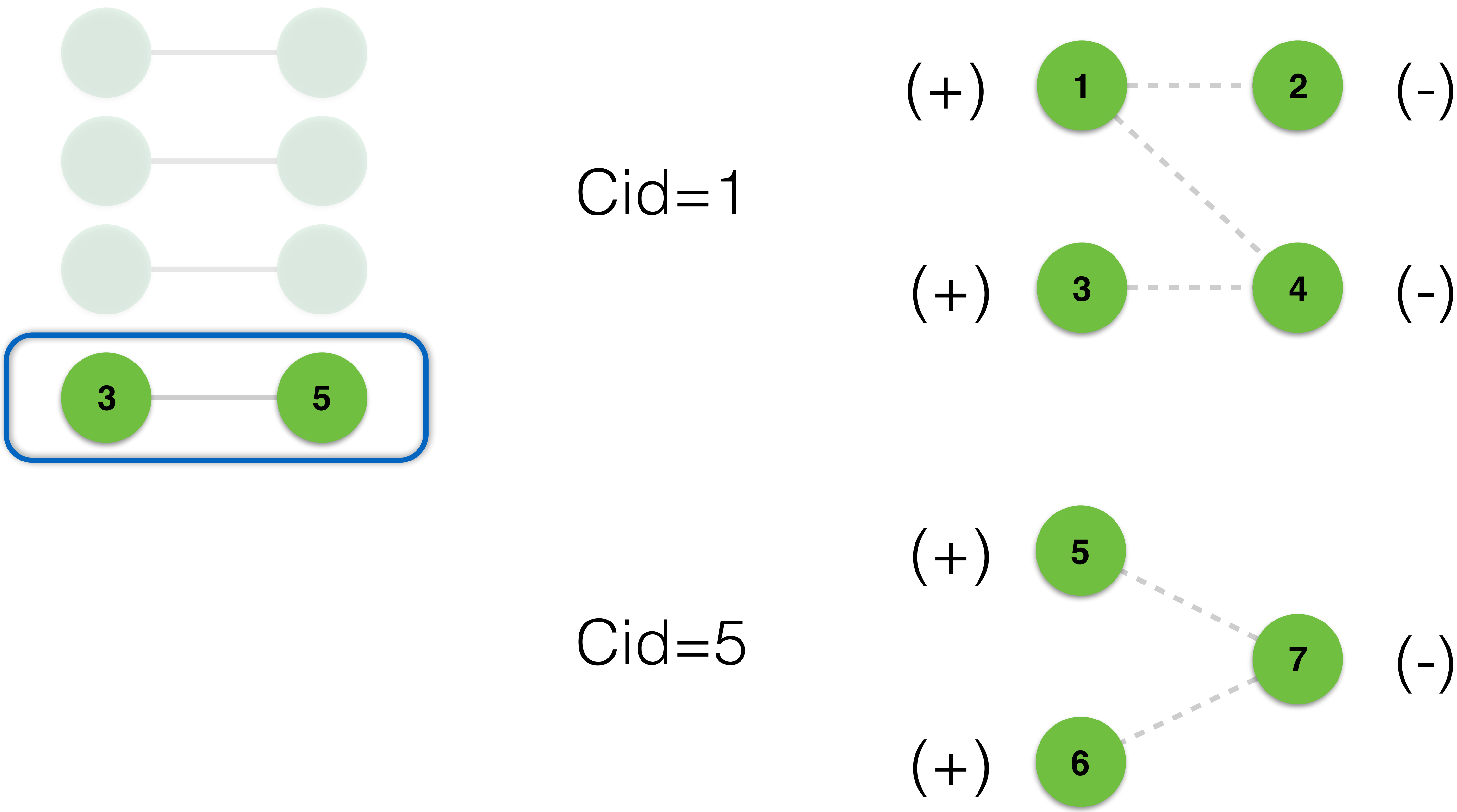


Stream Bipartite Detection

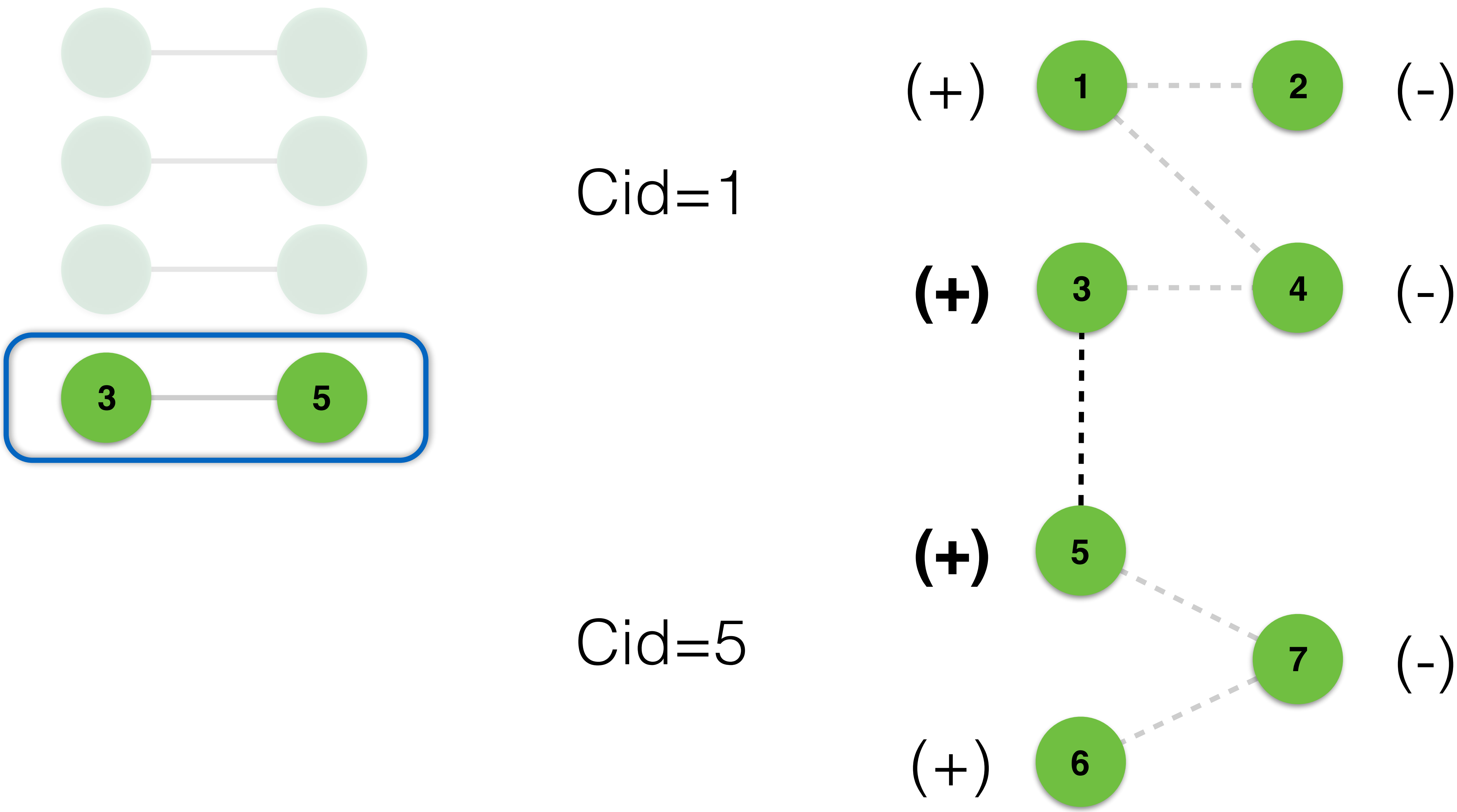




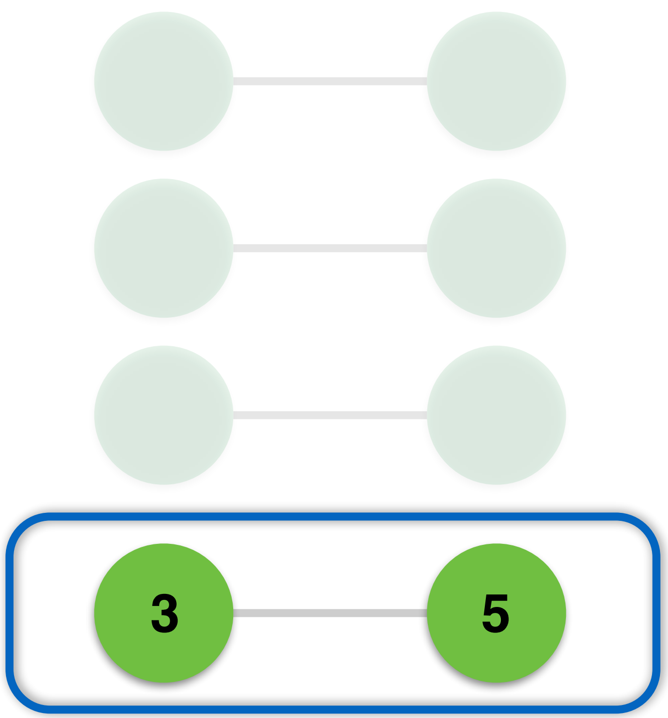
Stream Bipartite Detection



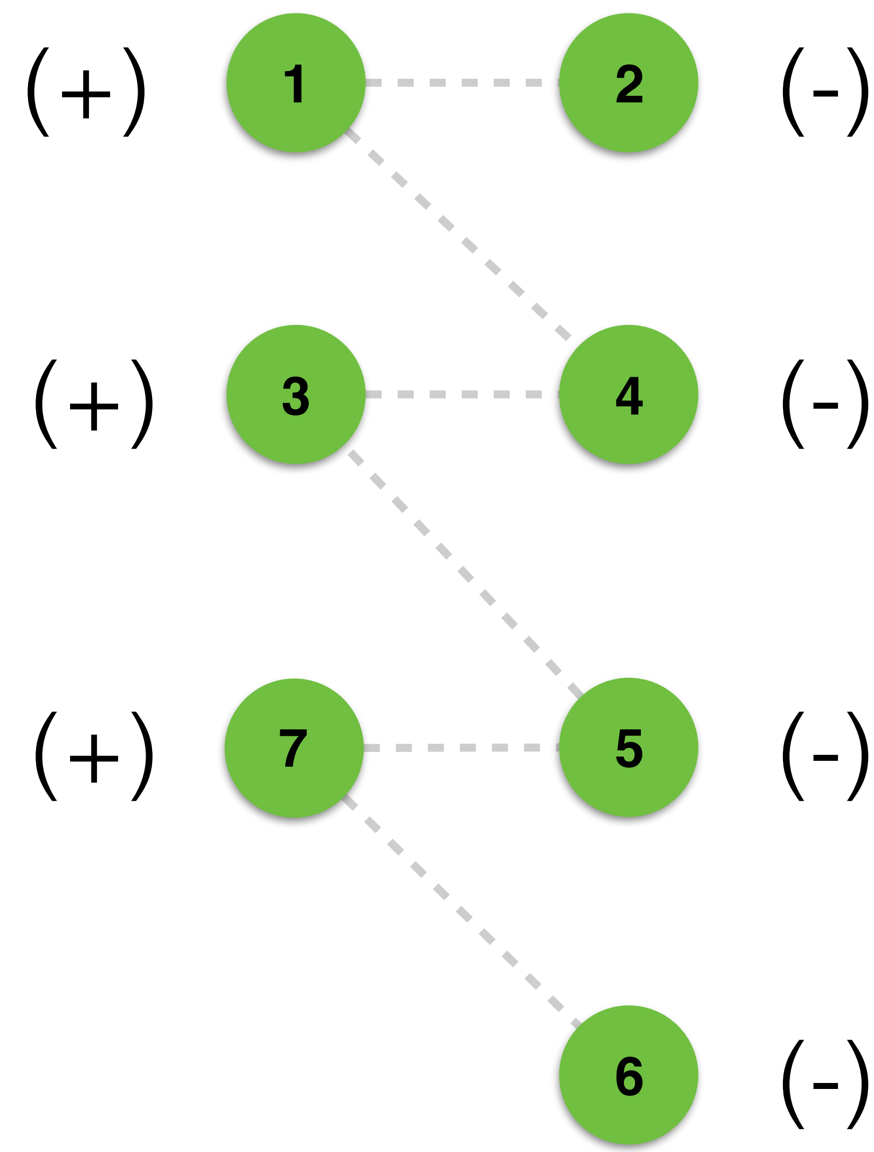
Stream Bipartite Detection



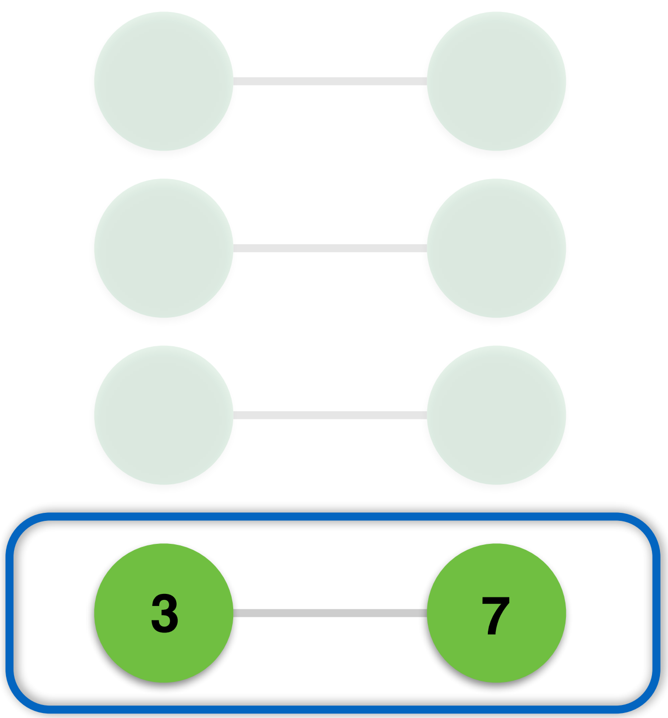
Stream Bipartite Detection



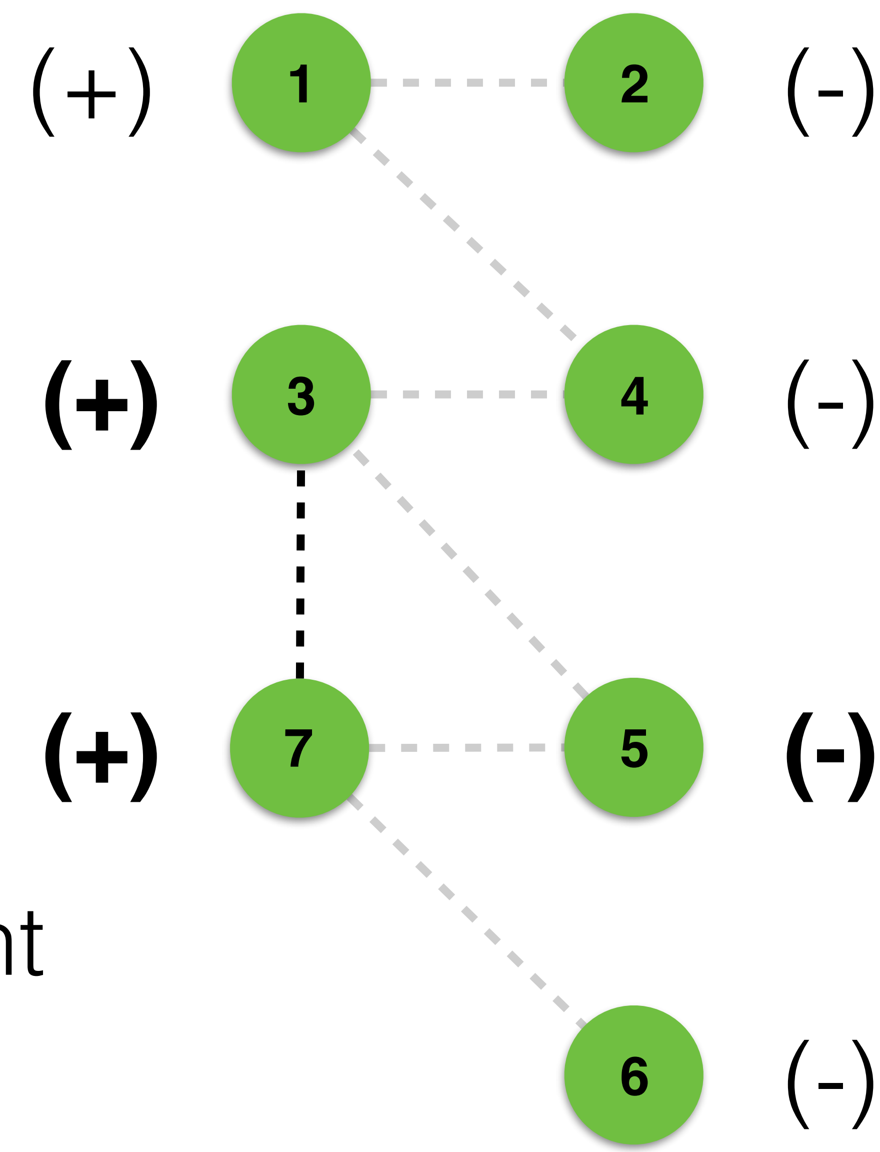
Cid=1



Stream Bipartite Detection



Cid=1



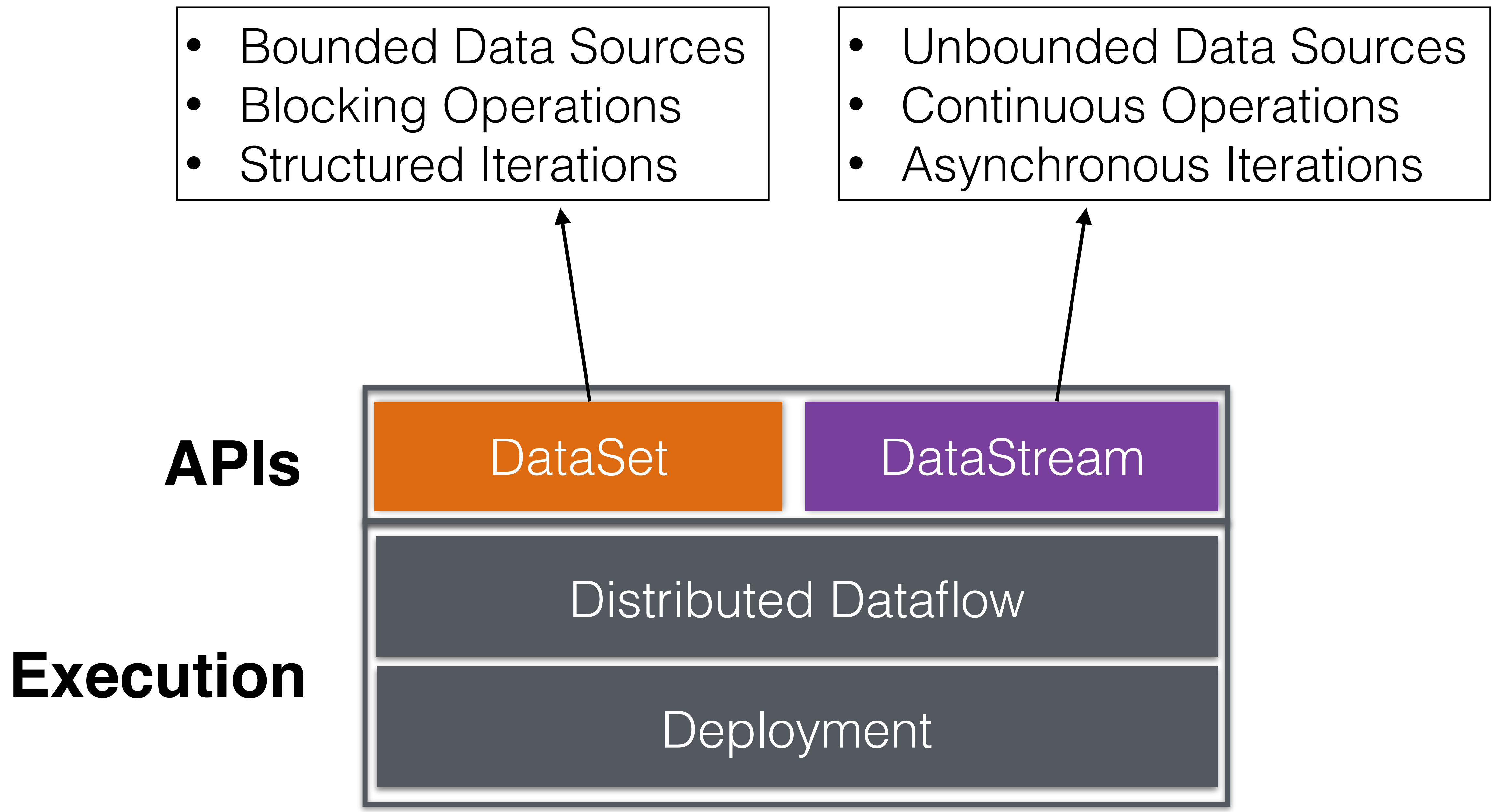
Can't flip signs and stay consistent
=> not bipartite!

API Requirements

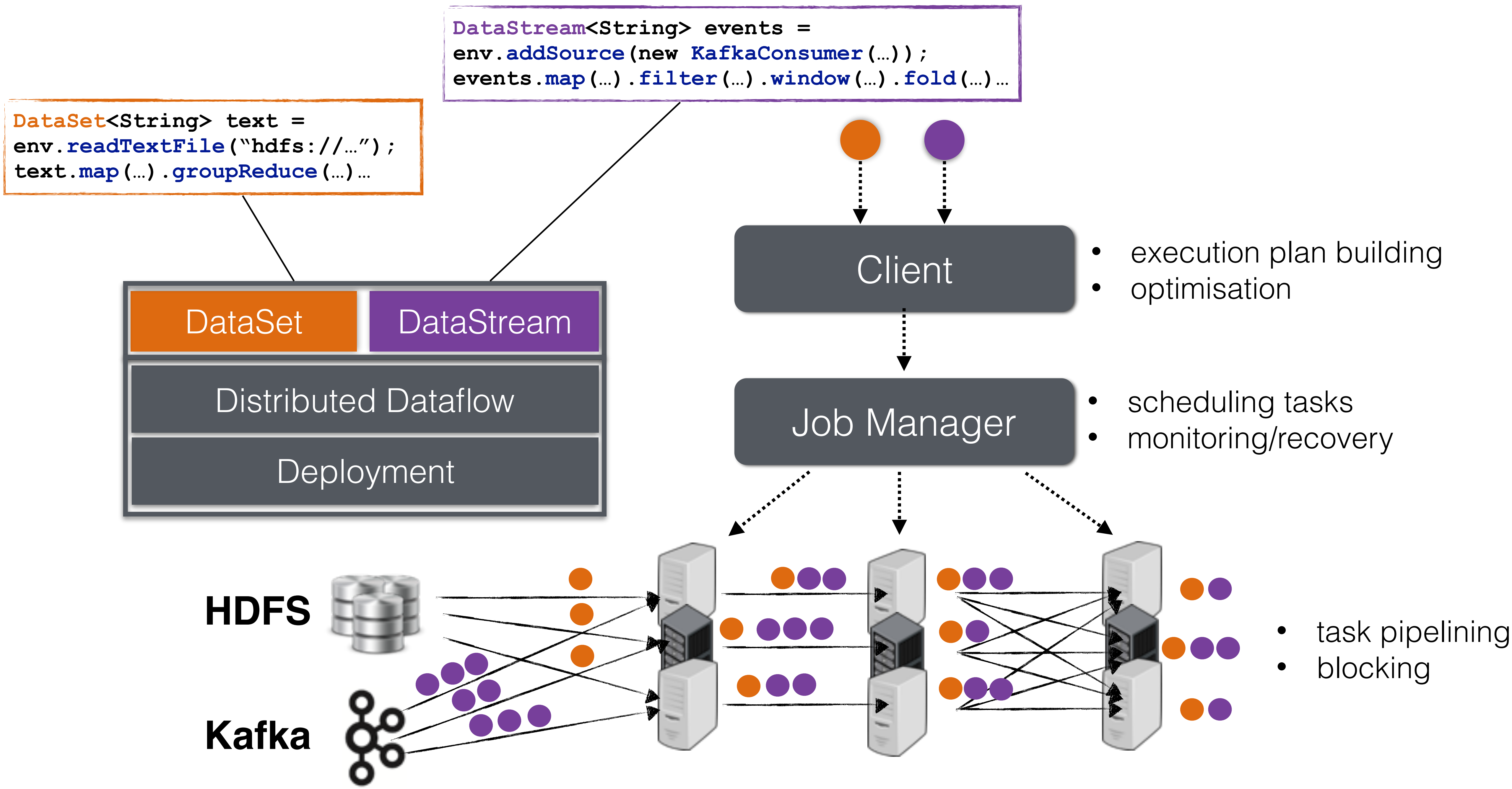
- Continuous aggregations on edge streams
- Global graph aggregations
- Support for windowing

@GraphDevroom

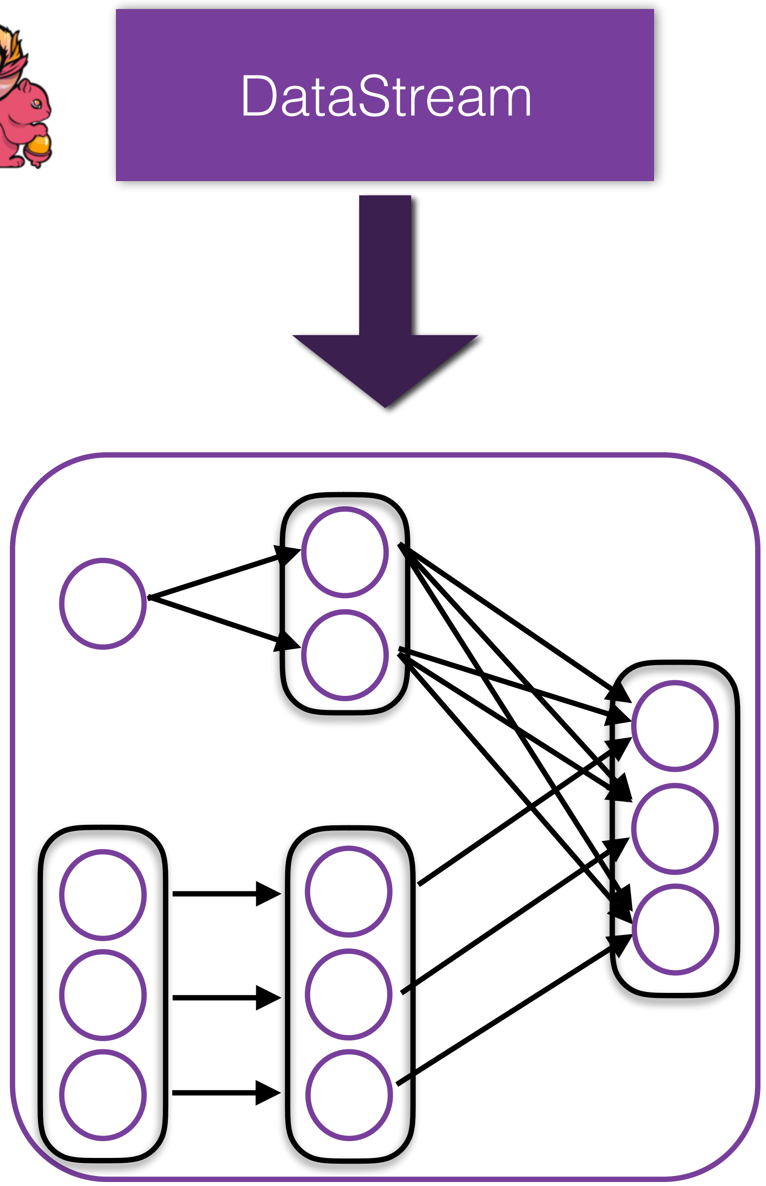
The Apache Flink Stack



Unifying Data Processing



Data Streams as ADTs

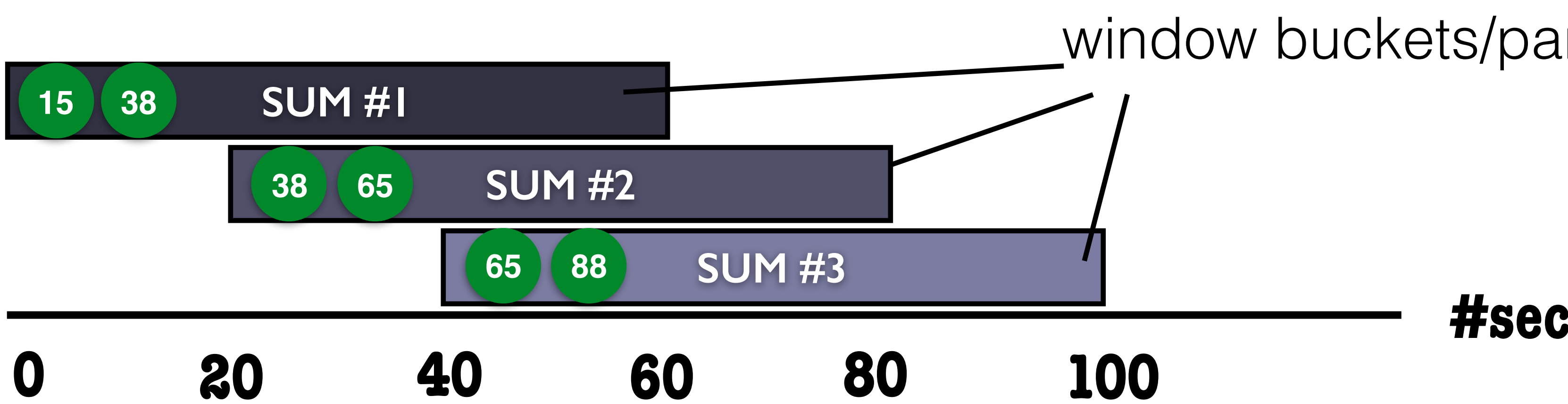


- **Transformations:** map, flatmap, filter, union…
- **Aggregations:** reduce, fold, sum
- **Partitioning:** forward, broadcast, shuffle, keyBy
- **Sources/Sinks:** custom or Kafka, Twitter, Collections…

- **Tasks** are long running in a pipelined execution.
- **State** is kept within tasks.
- **Transformations** are applied per-record or window.

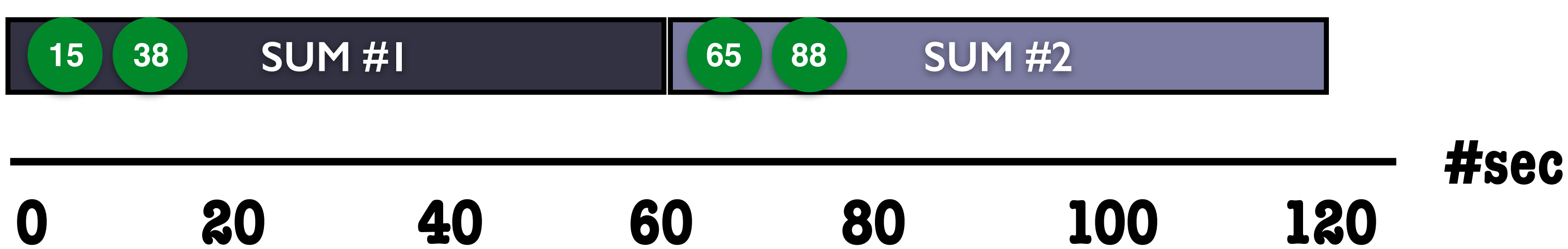
Working with Windows

Why windows?
We are often interested in *fresh* data!



1) Sliding windows

```
myKeyStream.timeWindow(
    Time.of(60, TimeUnit.SECONDS),
    Time.of(20, TimeUnit.SECONDS));
```



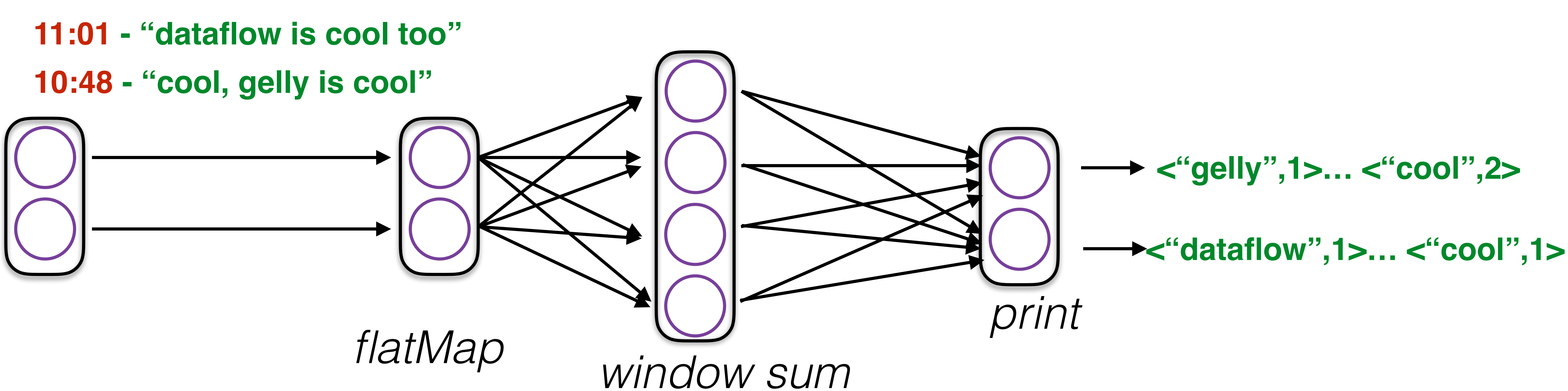
2) Tumbling windows

```
myKeyStream.timeWindow(
    Time.of(60, TimeUnit.SECONDS));
```

Highlight: Flink can form and trigger windows consistently under different notions of **time** and deal with late events!

Example

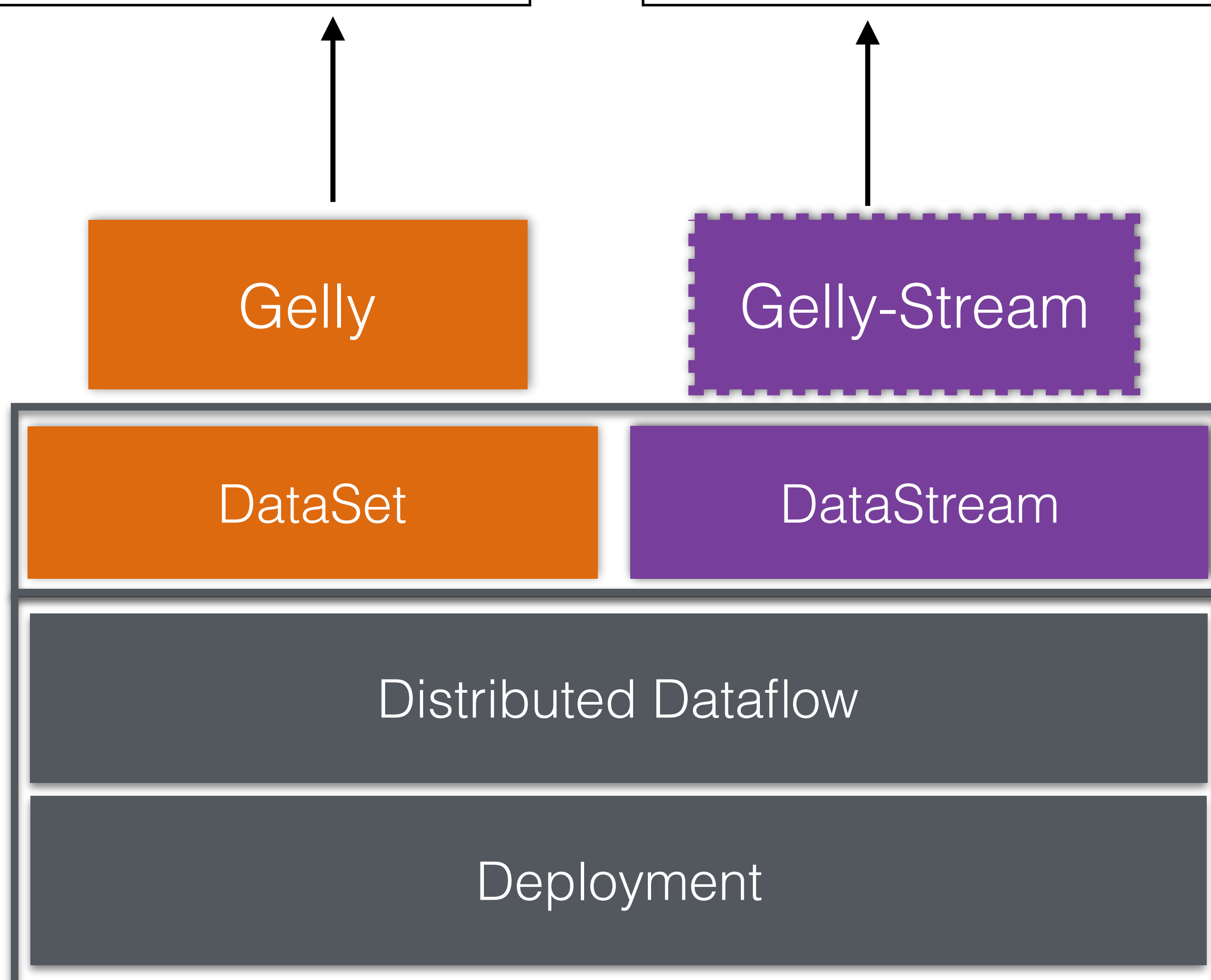
```
myTextStream
    .flatMap(new Splitter()) //transformation
    .keyBy(0) //partitioning
    .window(Time.of(5, TimeUnit.MINUTES))
    .sum(1) //rolling aggregation
    .setParallelism(4);
counts.print();
```



Gelly on Streams

- Static Graphs
- Multi-Pass Algorithms
- Full Computations

- Dynamic Graphs
- Single-Pass Algorithms
- Approximate Computations



Introducing Gelly-Stream

Gelly-Stream enriches the DataStream API with two new additional ADTs:

- GraphStream:
  - A representation of a data **stream of edges**.
  - Edges can have **state** (e.g. weights).
  - Supports **property** streams, **transformations** and **aggregations.**
- GraphWindow:
  - A “time-slice” of a graph stream.
  - It enables neighbourhood aggregations

GraphStream Operations

Property Streams

GraphStream -> DataStream

```
.getEdges()
.getVertices()
.numberOfVertices()
.numberOfEdges()
.getDegrees()
.inDegrees()
.outDegrees()
```

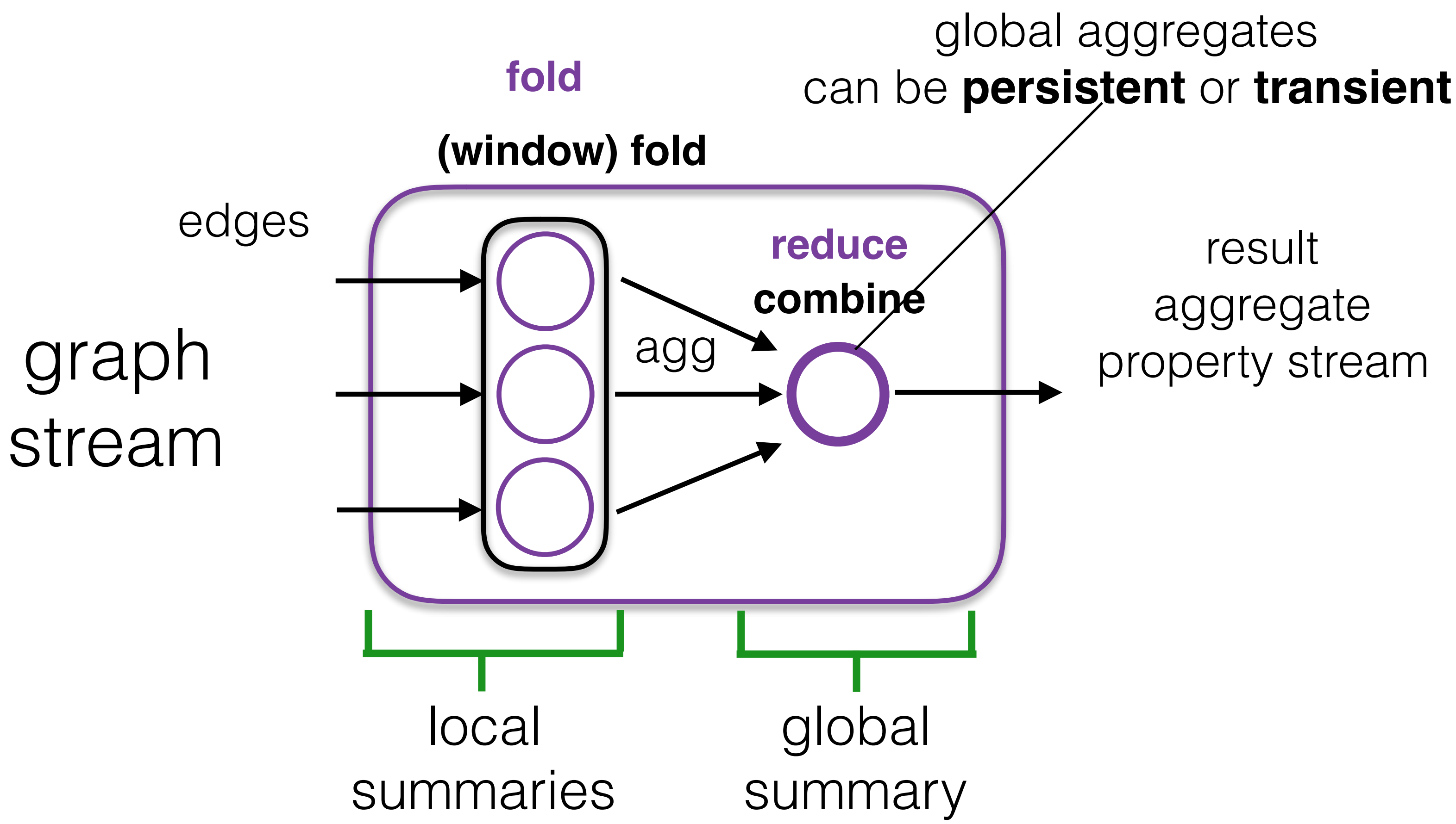
Transformations

GraphStream -> GraphStream

```
.mapEdges();
.distinct();
.filterVertices();
.filterEdges();
.reverse();
.undirected();
.union();
```

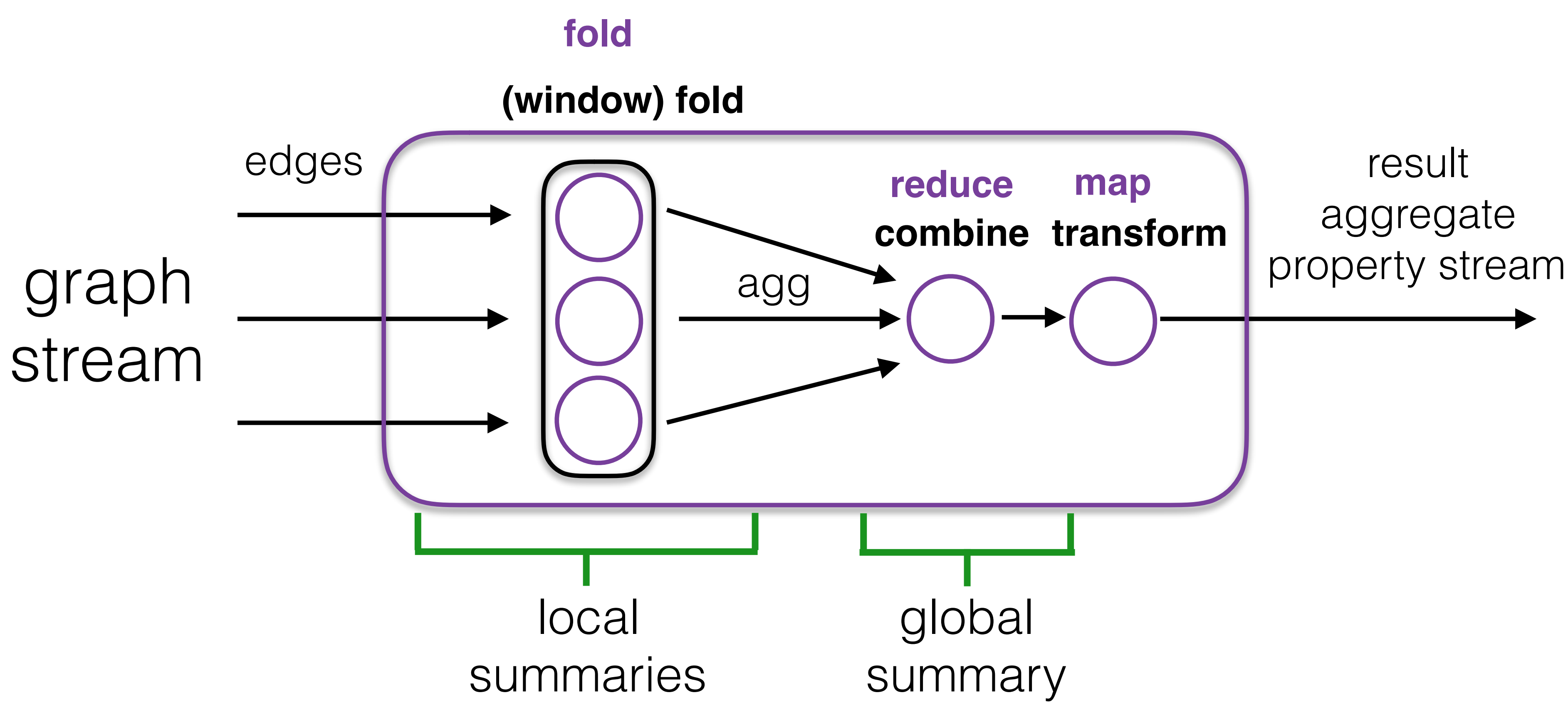
Graph Stream Aggregations

```
graphStream.aggregate(
    new MyGraphAggregation(window, fold, combine, transform))
```



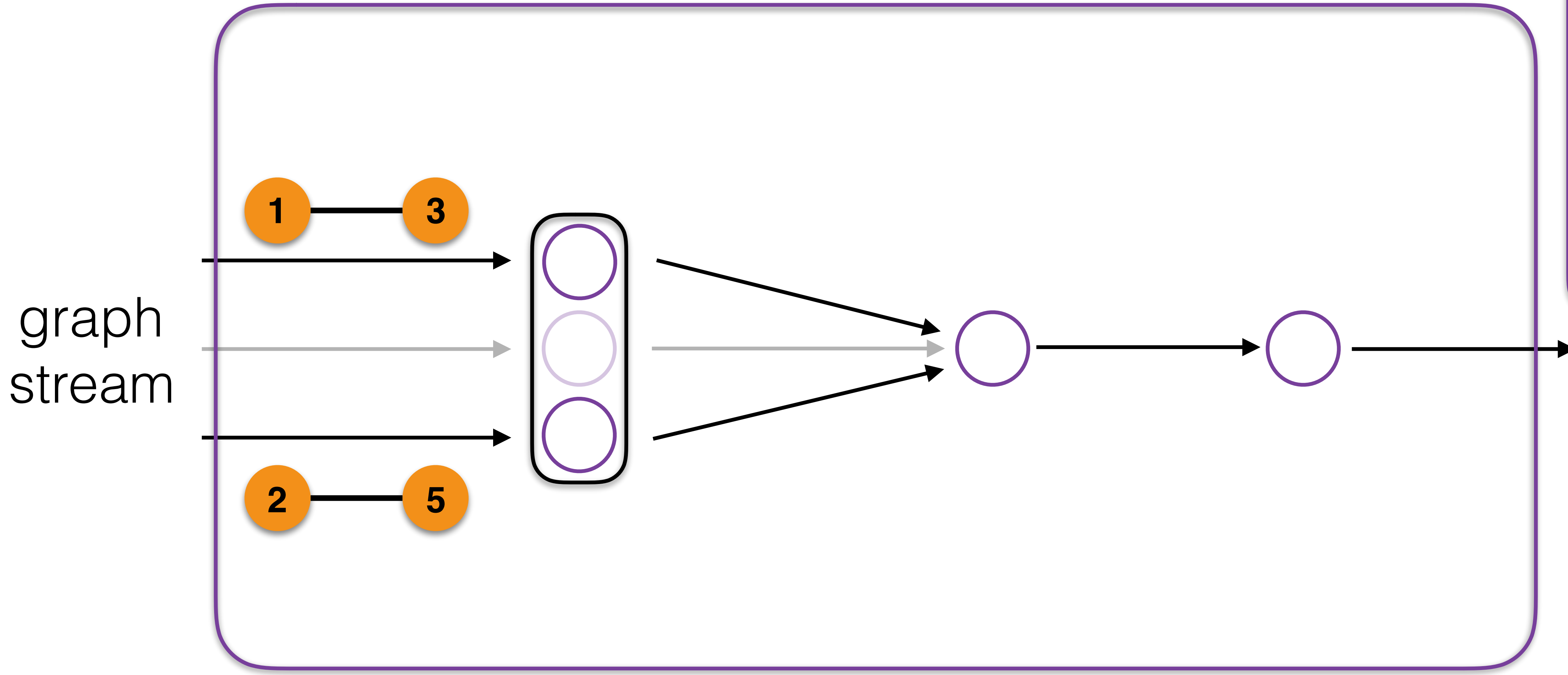
Graph Stream Aggregations

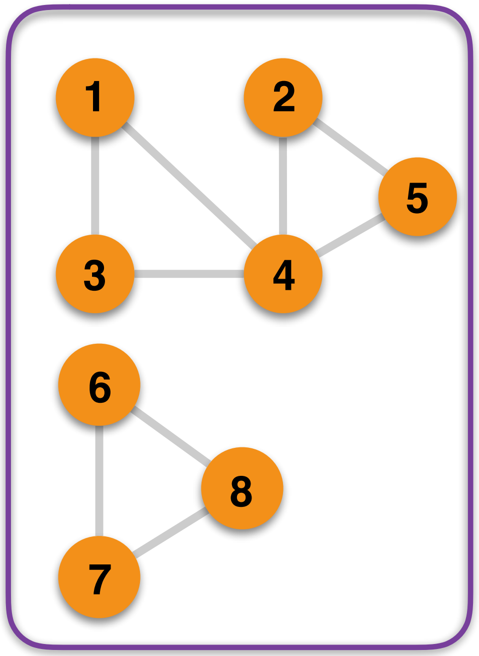
```
graphStream.aggregate(
     new MyGraphAggregation(window, fold, combine, transform))
```



Connected Components

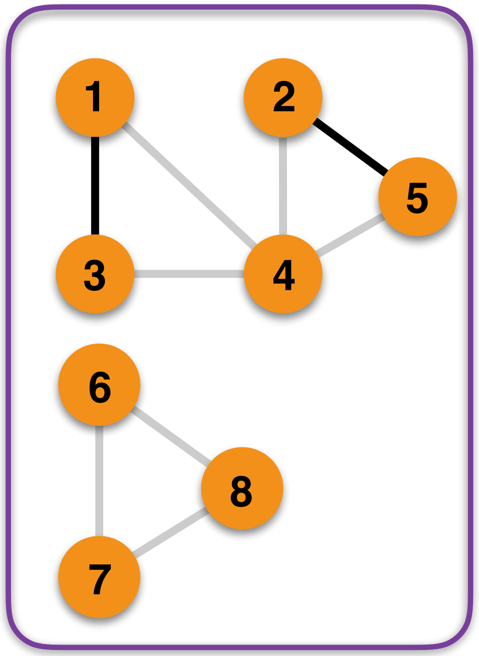
```
graphStream.aggregate(
new ConnectedComponents(window,fold,combine,transform))
```



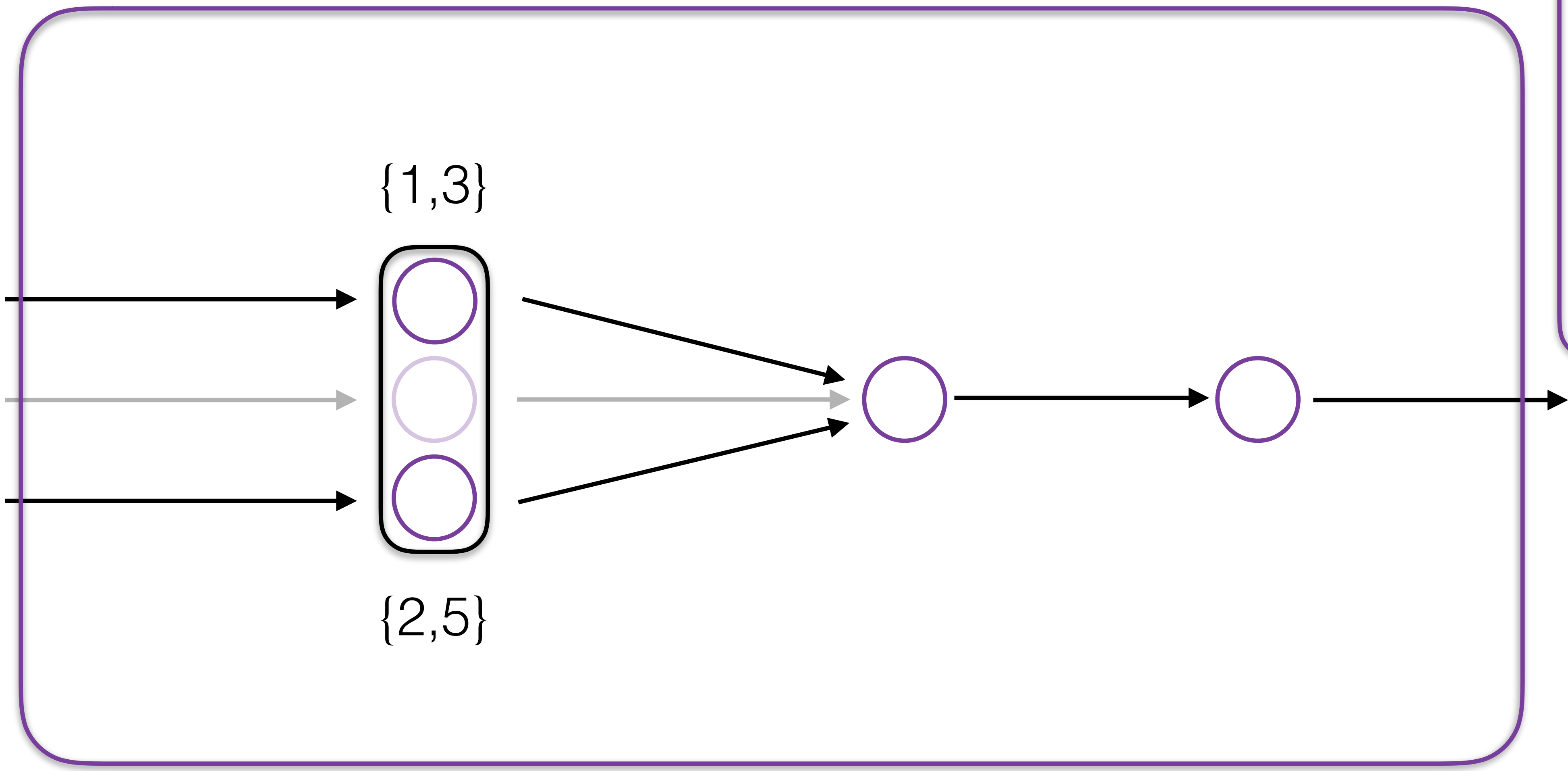


Connected Components

```
graphStream.aggregate(
new ConnectedComponents(window,fold,combine,transform))
```



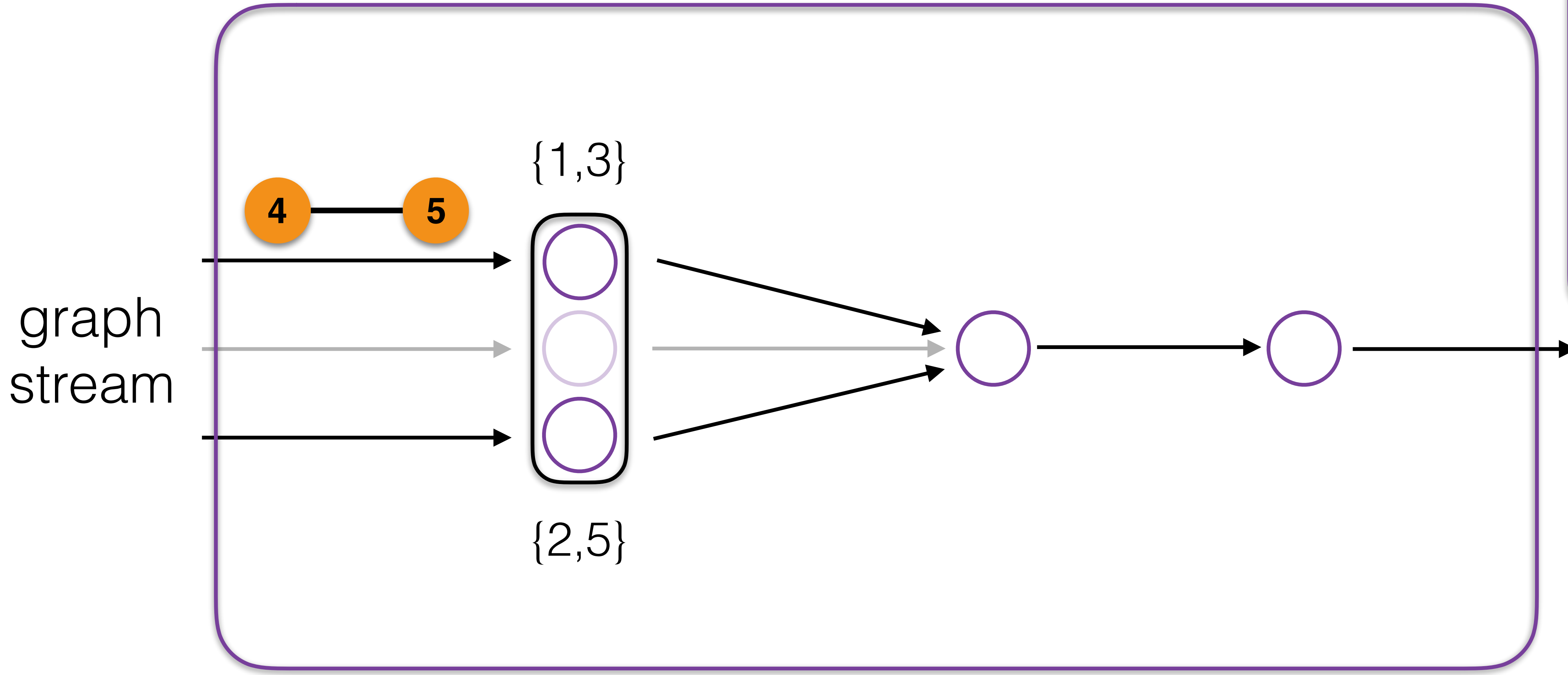
graph stream

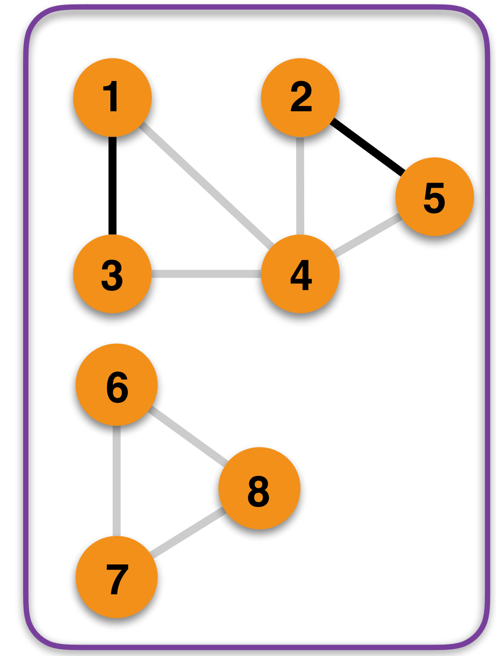


#components

Connected Components

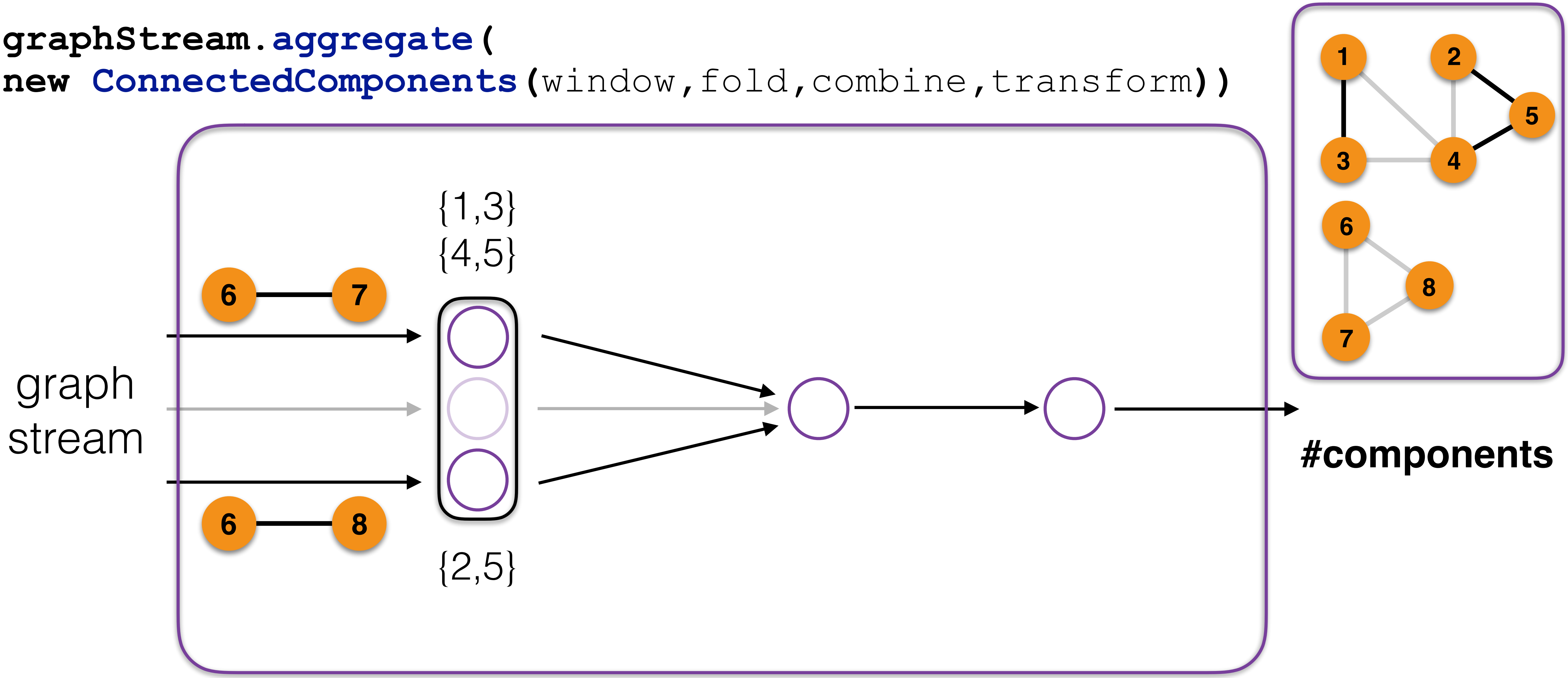
```
graphStream.aggregate(
new ConnectedComponents(window,fold,combine,transform))
```





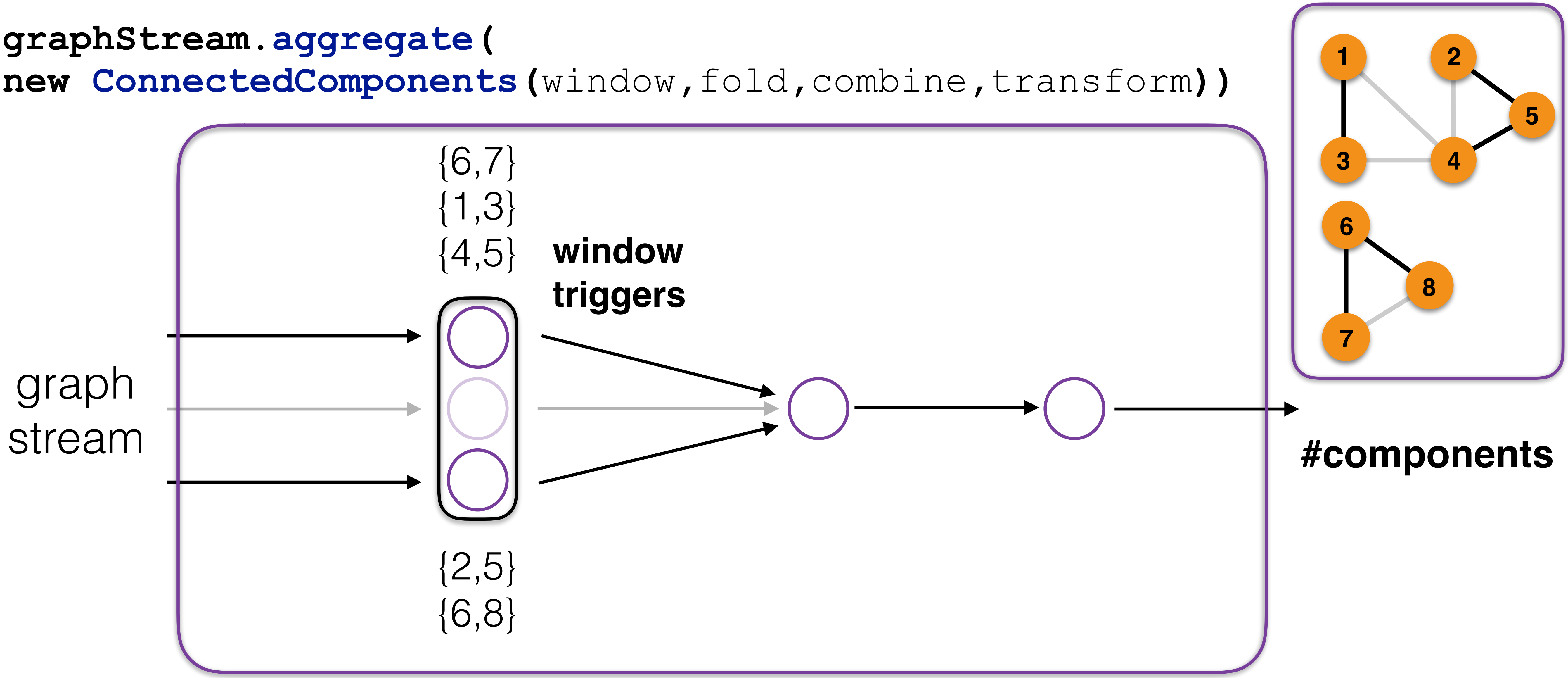
Connected Components

```
graphStream.aggregate(
new ConnectedComponents(window,fold,combine,transform))
```



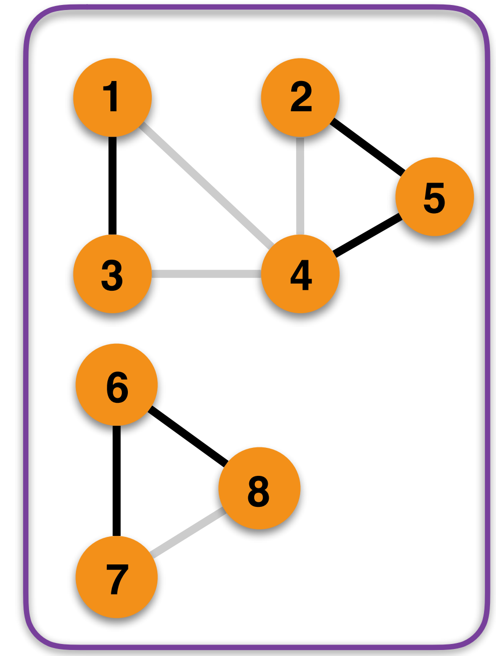
Connected Components

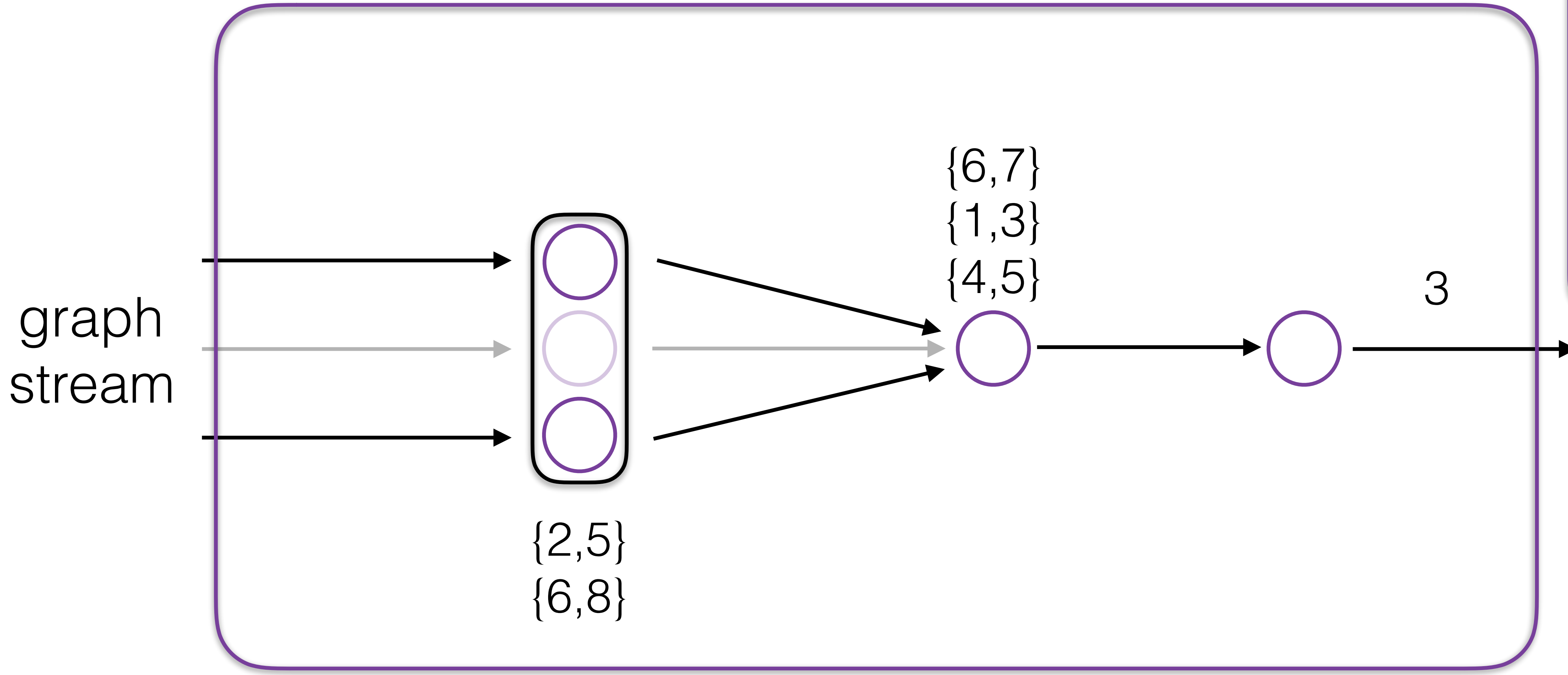
```
graphStream.aggregate(
new ConnectedComponents(window,fold,combine,transform))
```



Connected Components

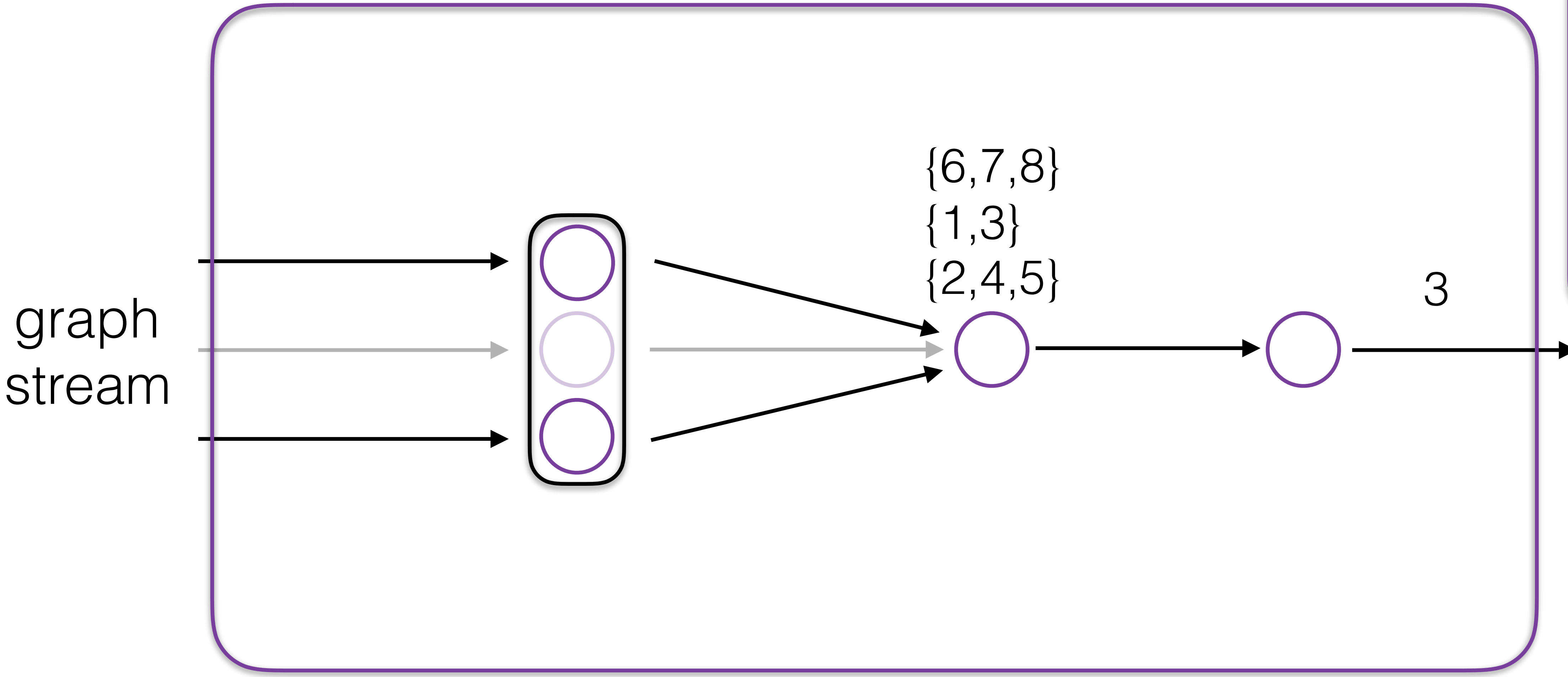
```
graphStream.aggregate(
new ConnectedComponents(window,fold,combine,transform))
```

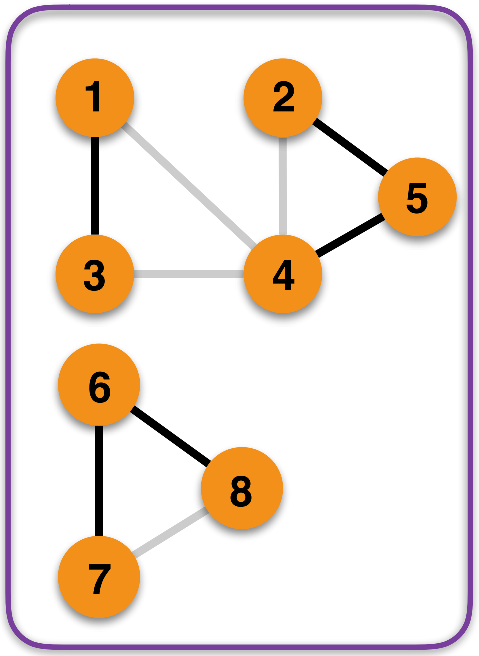




Connected Components

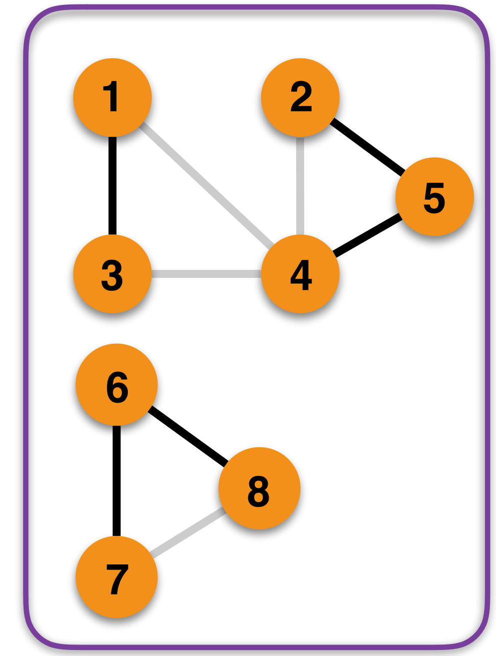
```
graphStream.aggregate(
new ConnectedComponents(window,fold,combine,transform))
```

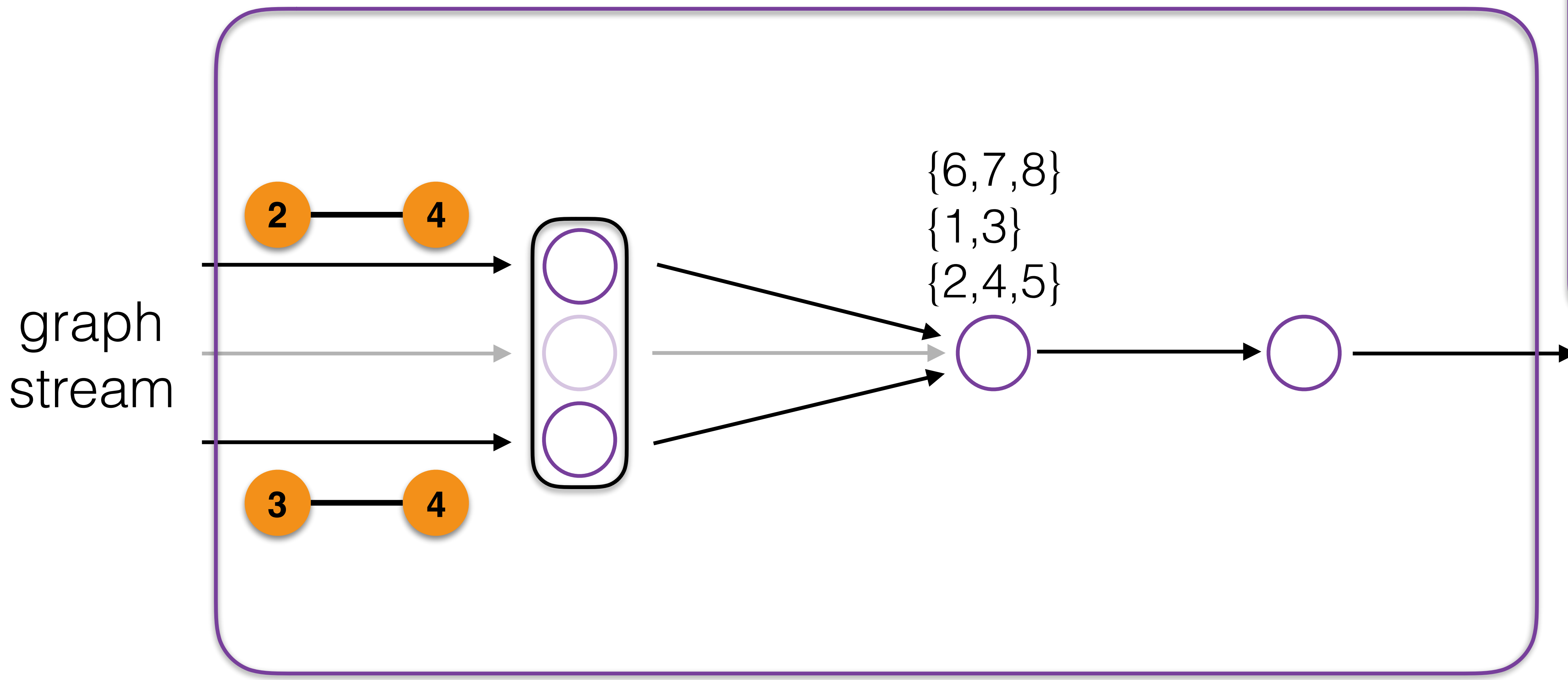




Connected Components

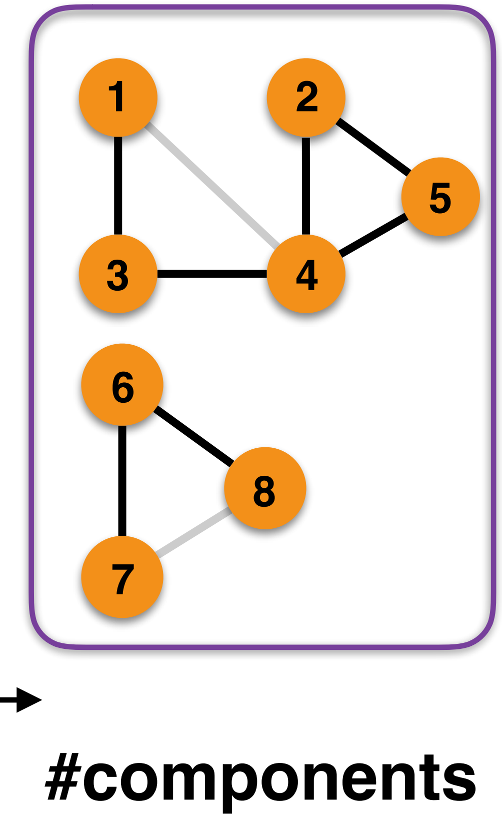
```
graphStream.aggregate(
new ConnectedComponents(window,fold,combine,transform))
```

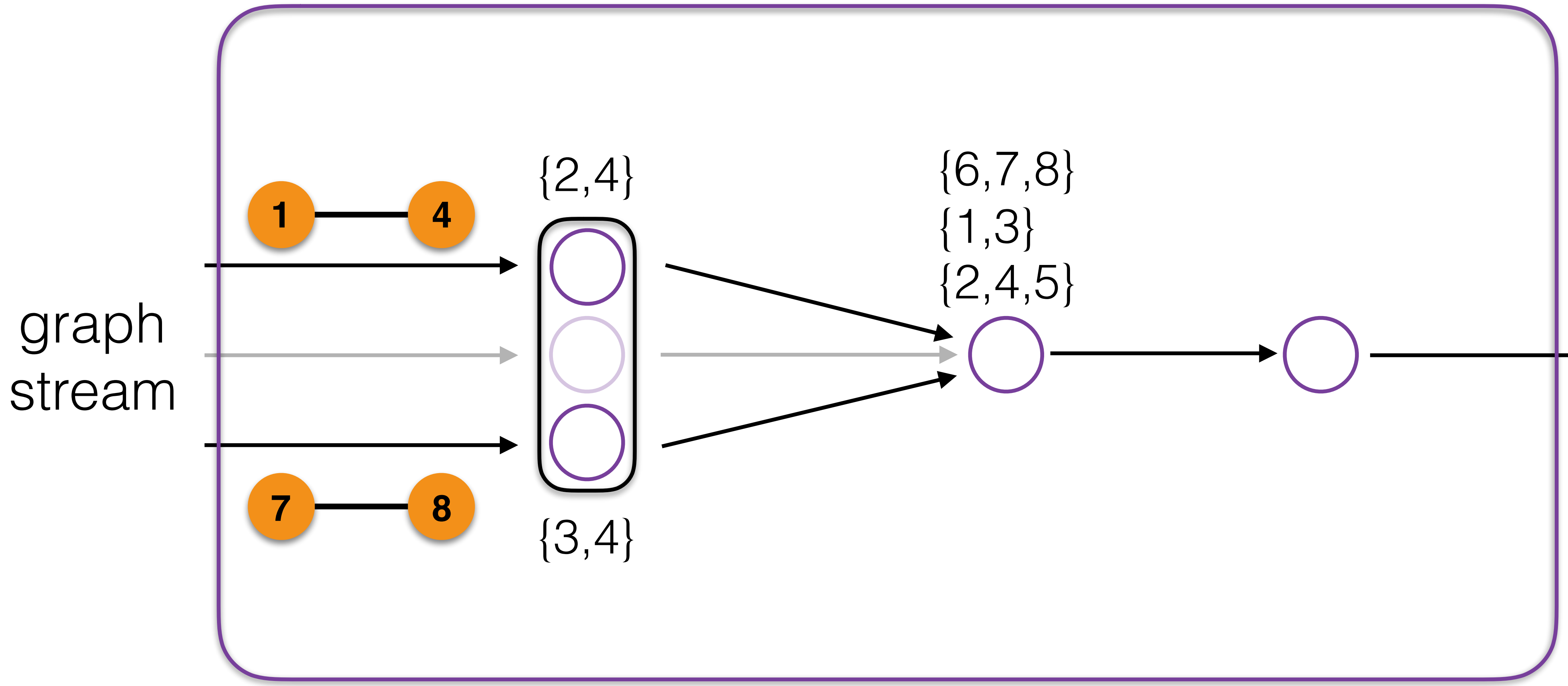




Connected Components

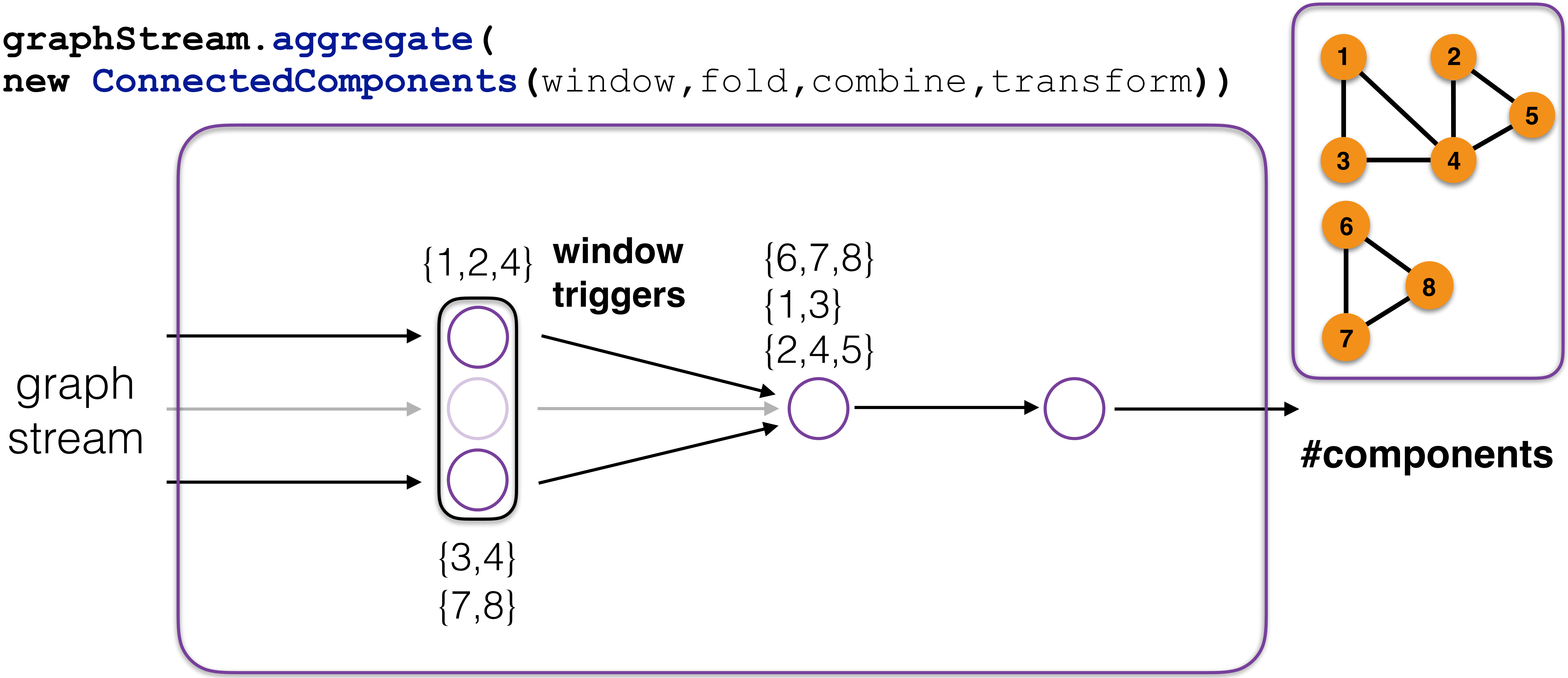
```
graphStream.aggregate(
new ConnectedComponents(window,fold,combine,transform))
```





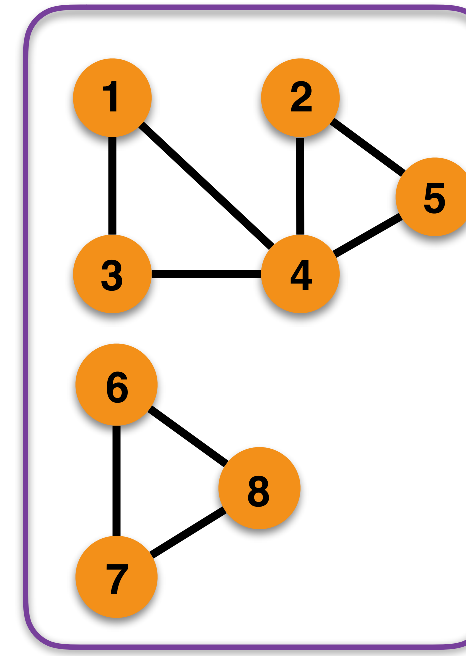
Connected Components

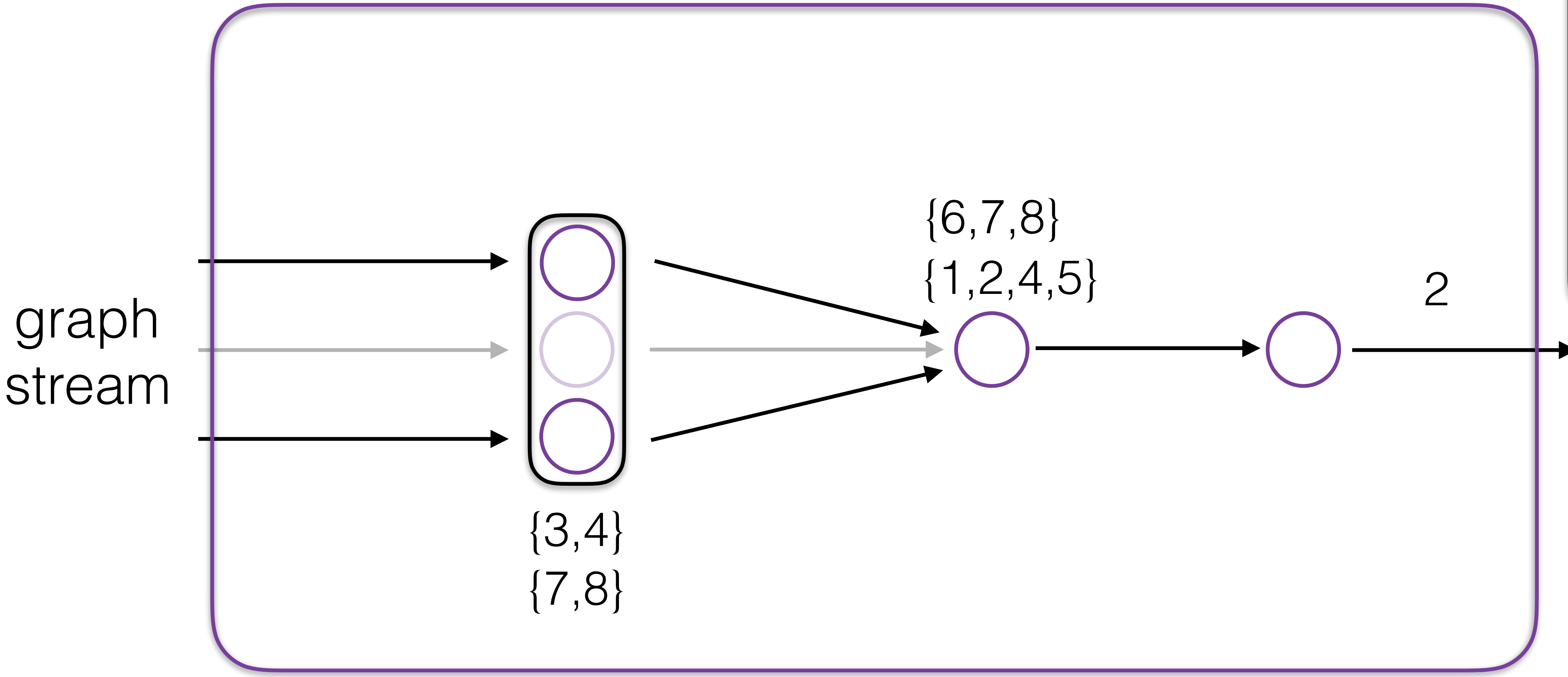
```
graphStream.aggregate(
new ConnectedComponents(window,fold,combine,transform))
```



Connected Components

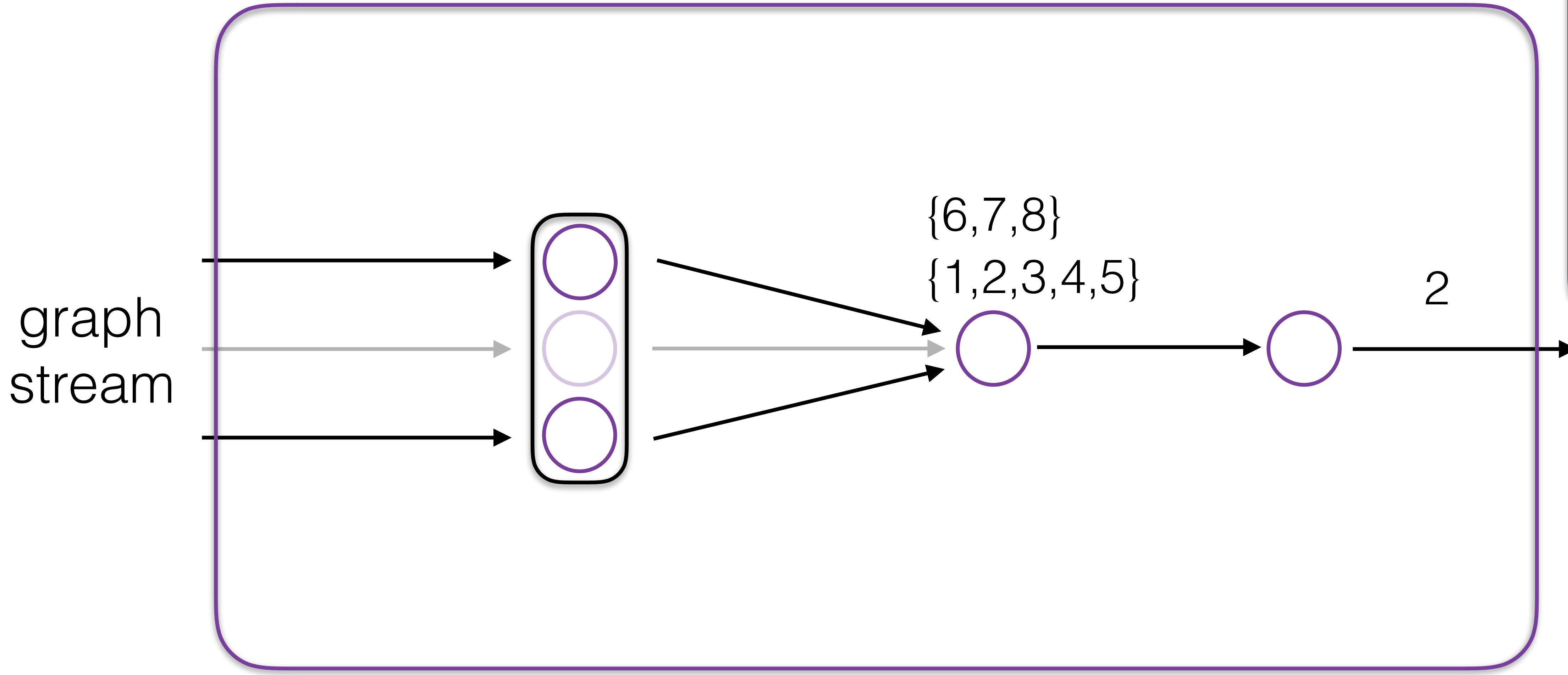
```
graphStream.aggregate(
new ConnectedComponents(window,fold,combine,transform))
```

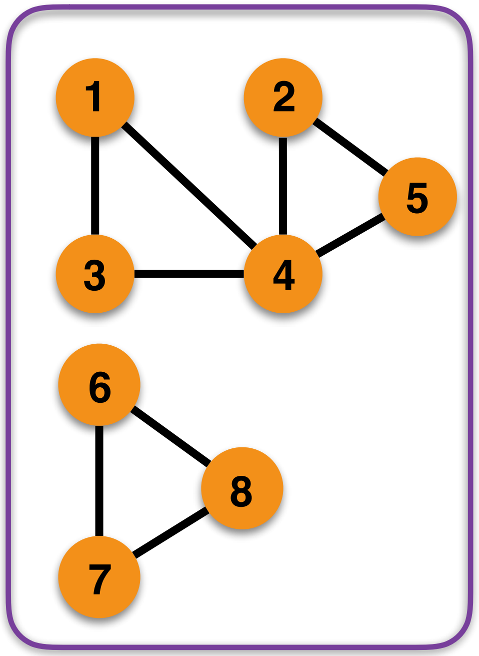




Connected Components

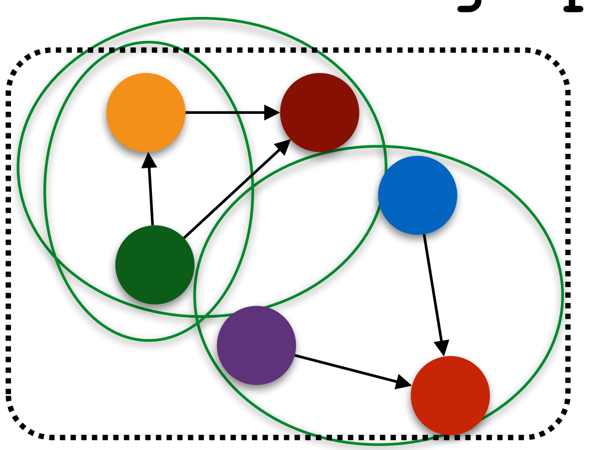
```
graphStream.aggregate(
new ConnectedComponents(window,fold,combine,transform))
```





Aggregating Slices

```
graphStream.slice(Time.of(1, MINUTE), direction)
```

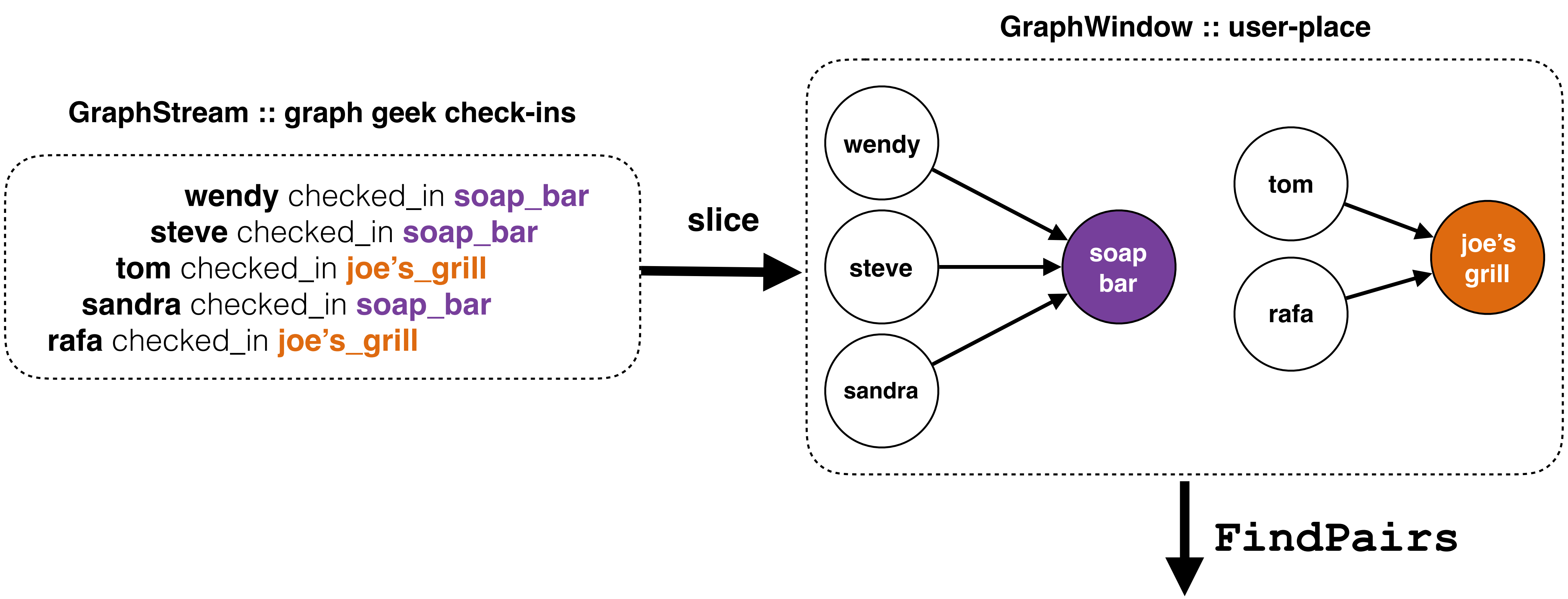
<u>Aggregations</u>

```
.reduceOnEdges();
.foldNeighbors();
.applyOnNeighbors();
```

- Slicing collocates edges by vertex information
- Neighbourhood aggregations are now enabled on sliced graphs

Finding Matches Nearby

```
graphStream.filterVertices(GraphGeeks())
    .slice(Time.of(15, MINUTE), EdgeDirection.IN)
    .applyOnNeighbors(FindPairs())
```




```
{wendy, steve}
{steve, sandra}
{wendy, sandra}
{tom, rafa}
```

Feeling Gelly?

- **Gelly-Stream**: https://github.com/vasia/gelly-streaming
- **Apache Flink**: https://github.com/apache/flink
- **An interesting read**: http://users.dcc.uchile.cl/~pbarcelo/mcg.pdf
- **A cool thesis**: http://urn.kb.se/resolve?urn=urn:nbn:se:kth:diva-170425
- **Twitter:** @vkalavri , @senorcarbone