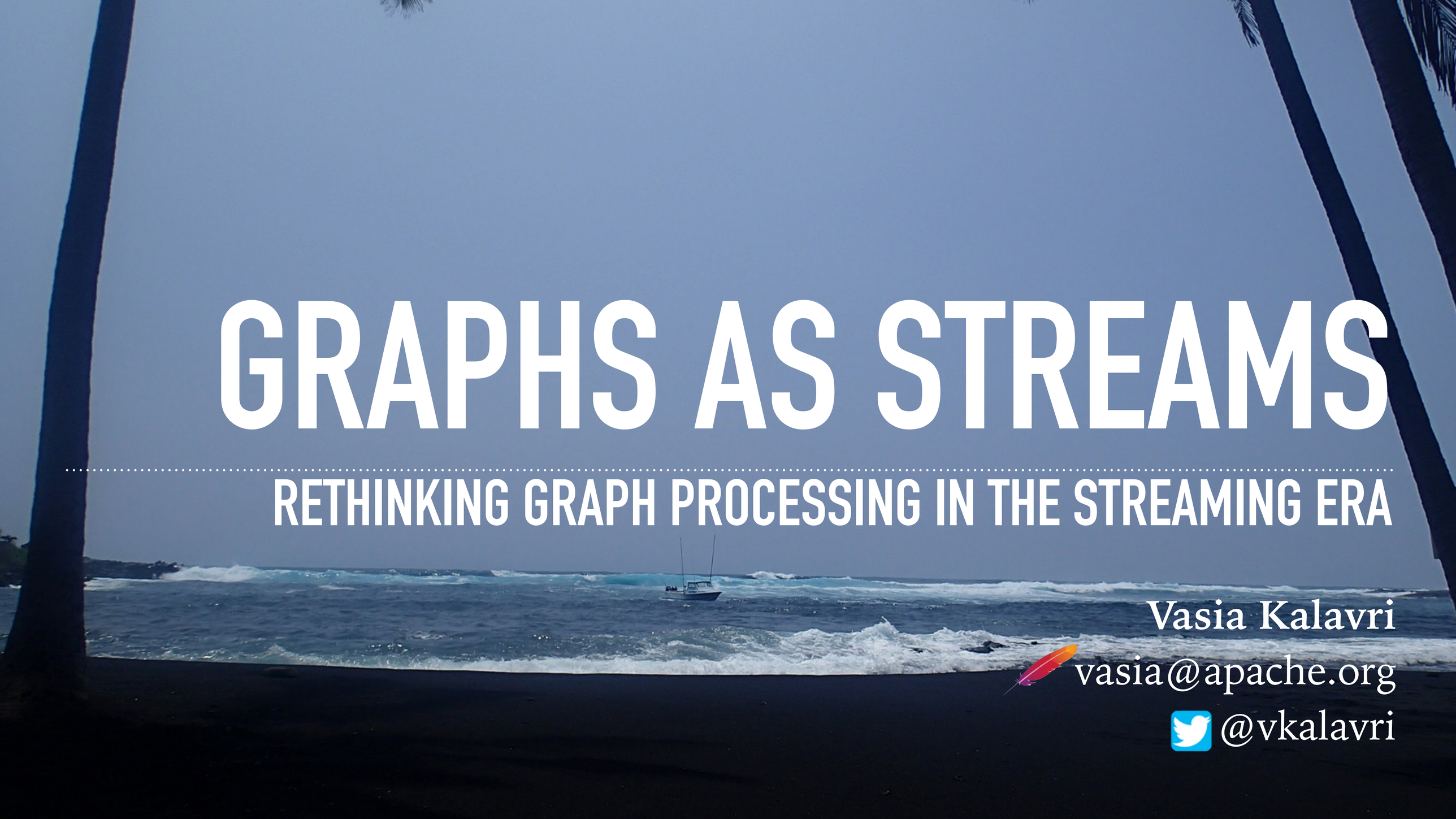
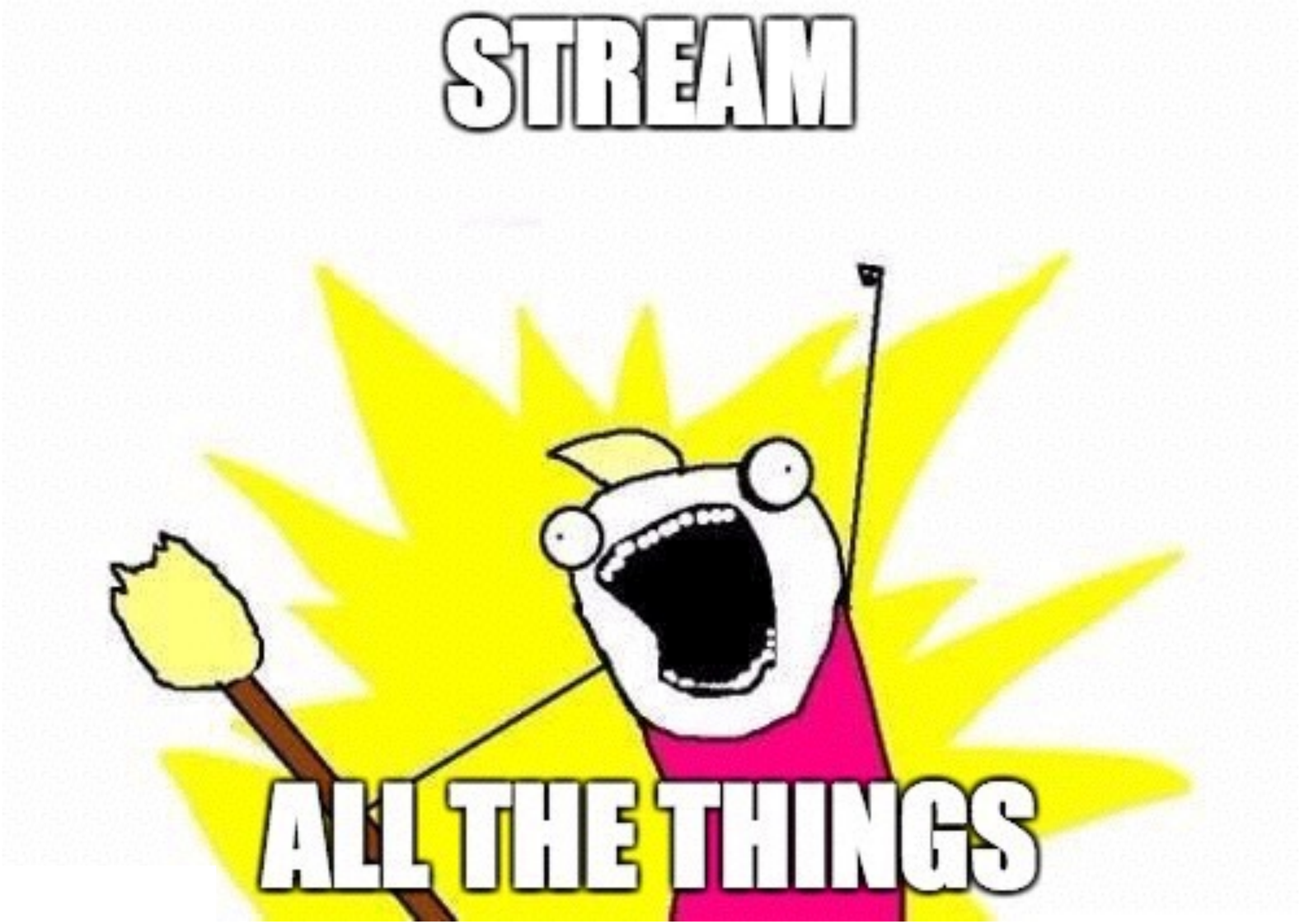


GRAPHS AS STREAMS

RETHINKING GRAPH PROCESSING IN THE STREAMING ERA

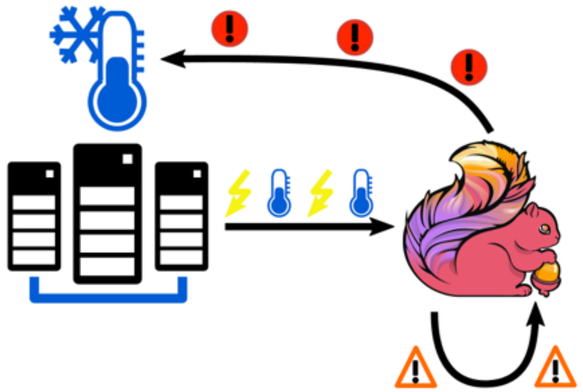
Vasia Kalavri
vasia@apache.org
@vkalavri


STREAM
ALL THE THINGS

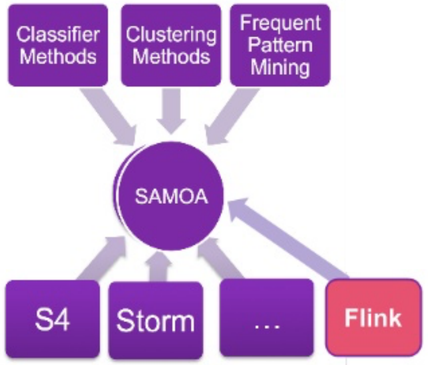
MODERN STREAMING TECHNOLOGY

- sub-second latencies
- high throughput
- dynamic topologies
- powerful semantics
- ecosystem integration

MORE THAN COUNTING WORDS



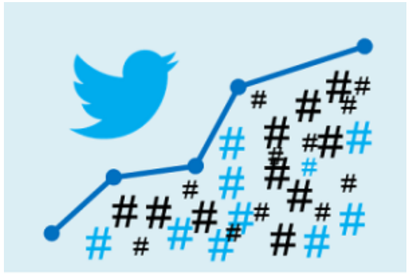
Complex Event Processing

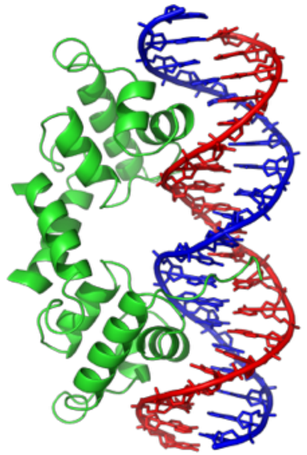


Online Machine Learning

Streaming SQL

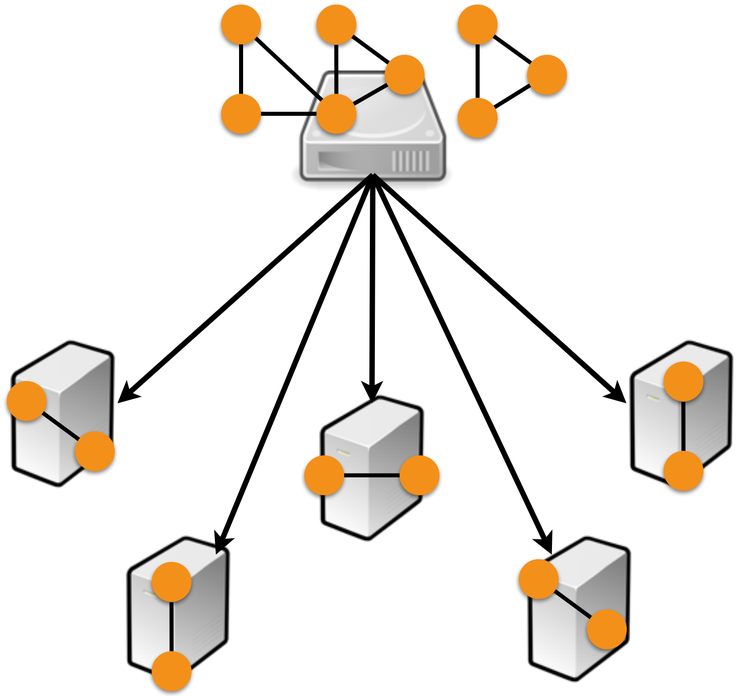
WHAT ABOUT GRAPH PROCESSING?





HOW WE'VE DONE GRAPH PROCESSING SO FAR

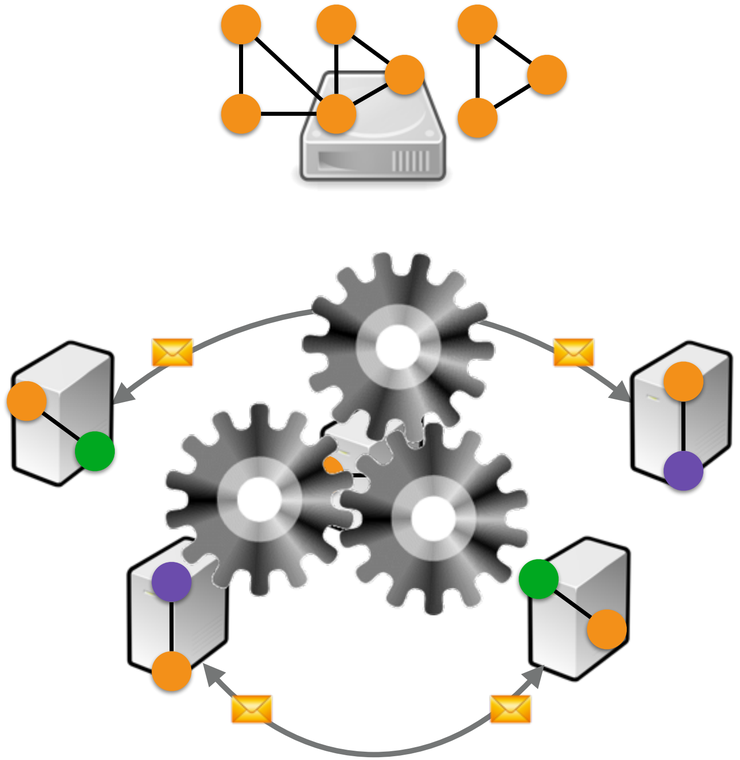
1. **Load**: read the graph from disk and partition it in memory



HOW WE'VE DONE GRAPH PROCESSING SO FAR

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

2. **Compute**: read and mutate the graph state

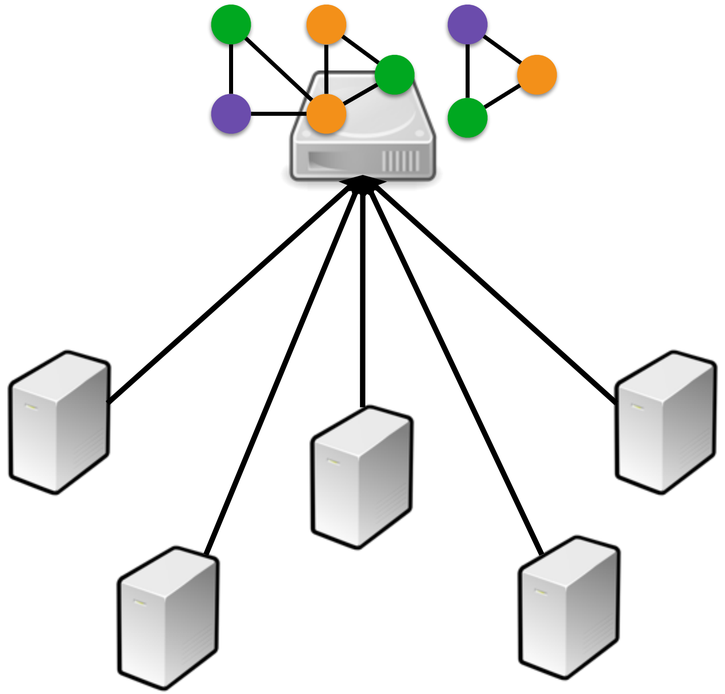


HOW WE'VE DONE GRAPH PROCESSING SO FAR

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



“

If what you need

is to analyze a **static** graph

over and over again

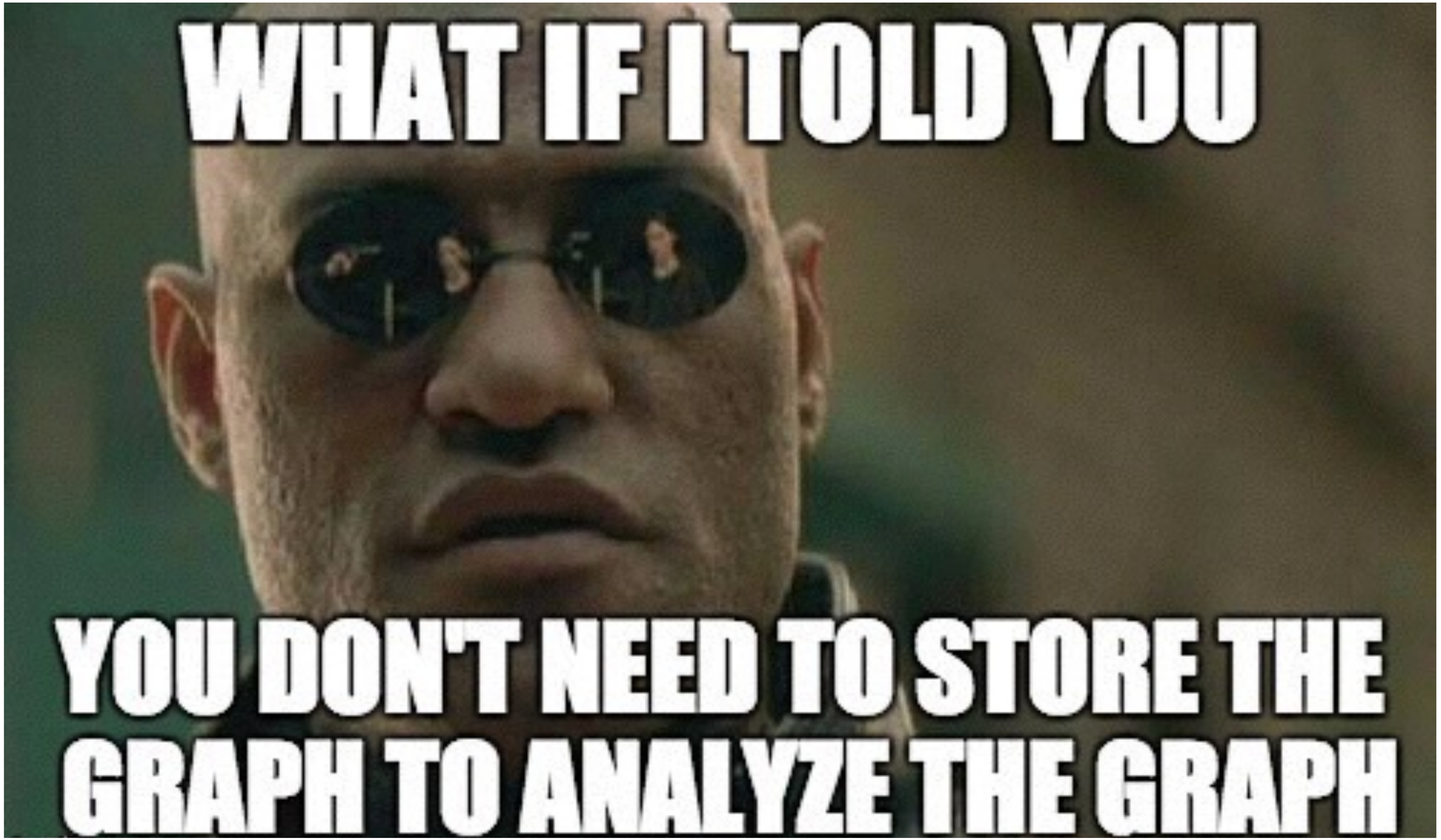
then this model is great!

WHAT'S WRONG WITH THIS MODEL?

- It is **slow**
  - wait until the computation is over before you see any result
  - pre-processing and partitioning
- It is **expensive**
  - lots of memory and CPU required in order to scale
- It requires **re-computation** for graph changes
  - no efficient way to deal with updates

GRAPH STREAMING CHALLENGES

- Maintain the dynamic graph **structure**
- Provide **up-to-date** results with low latency
- Compute on **fresh state** only

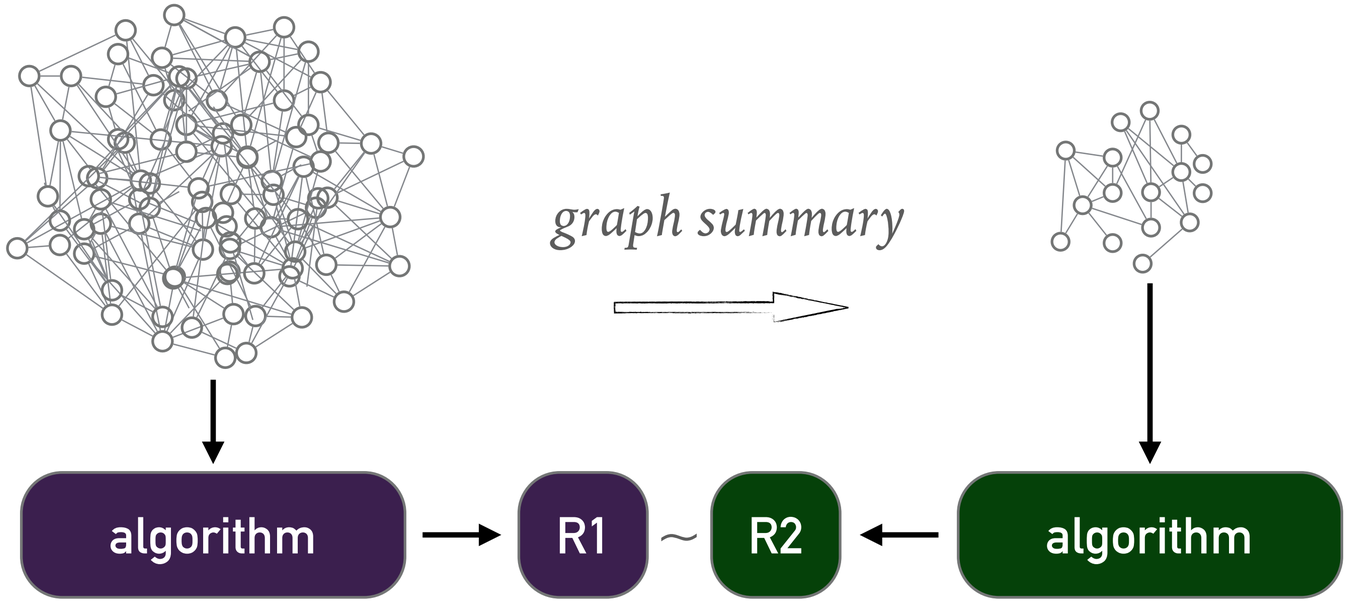

WHAT IF I TOLD YOU
YOU DON'T NEED TO STORE THE GRAPH TO ANALYZE THE GRAPH

ACADEMIA TO THE RESCUE

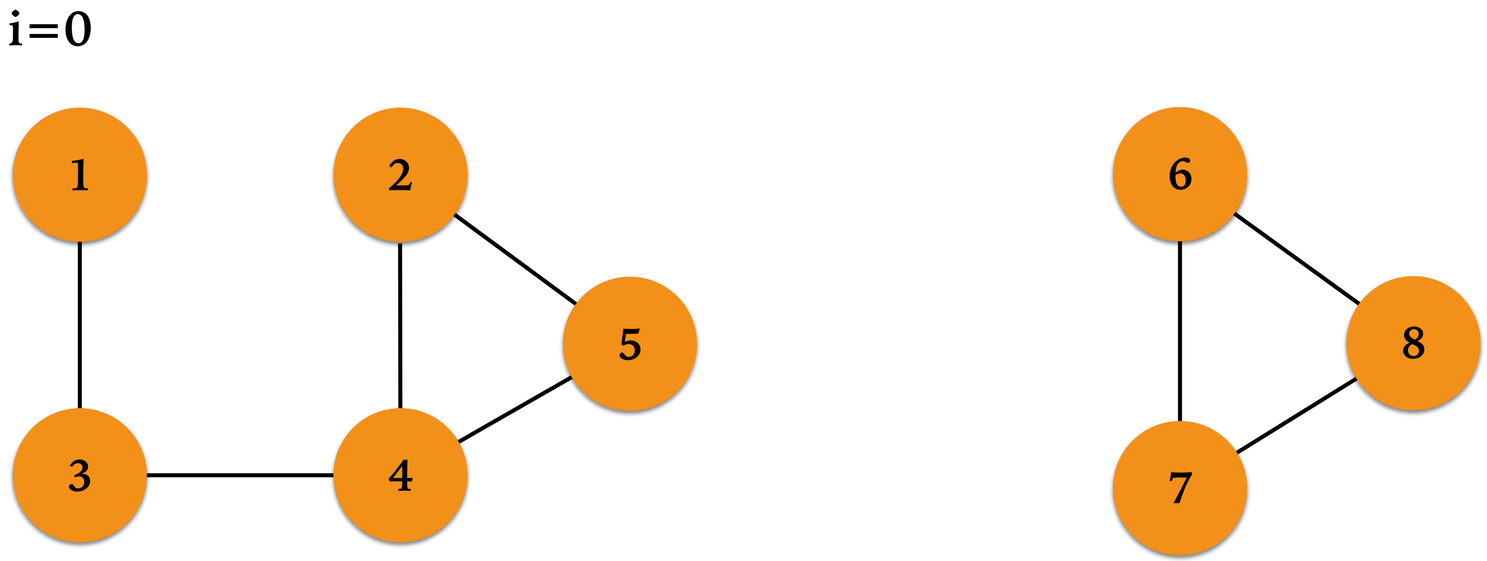
- Graph streaming in the 90s-00s
  - input fits in **secondary storage**
  - **limited memory**
  - **few passes** over the input data
  - compact graph representations and **summaries**

GRAPH SUMMARIES

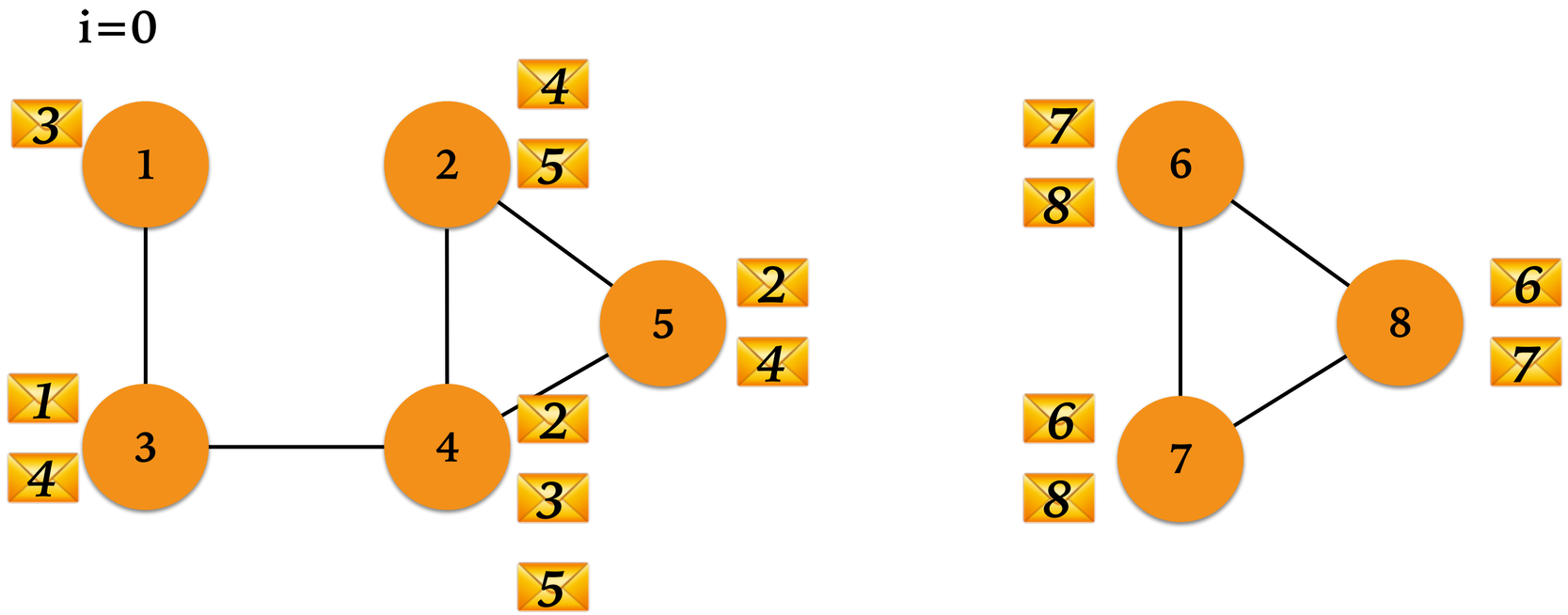
- **spanners**
  - connectivity, distance
- **sparsifiers**
  - cut estimation
- neighborhood **sketches**



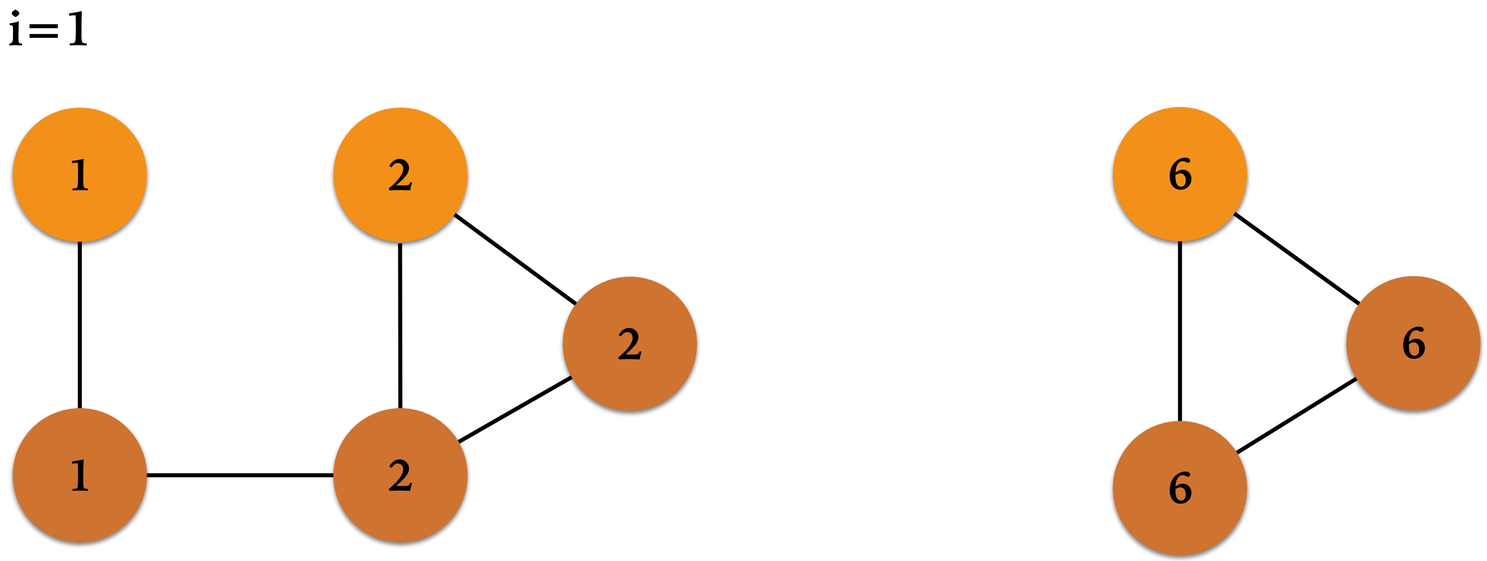
BATCH CONNECTED COMPONENTS



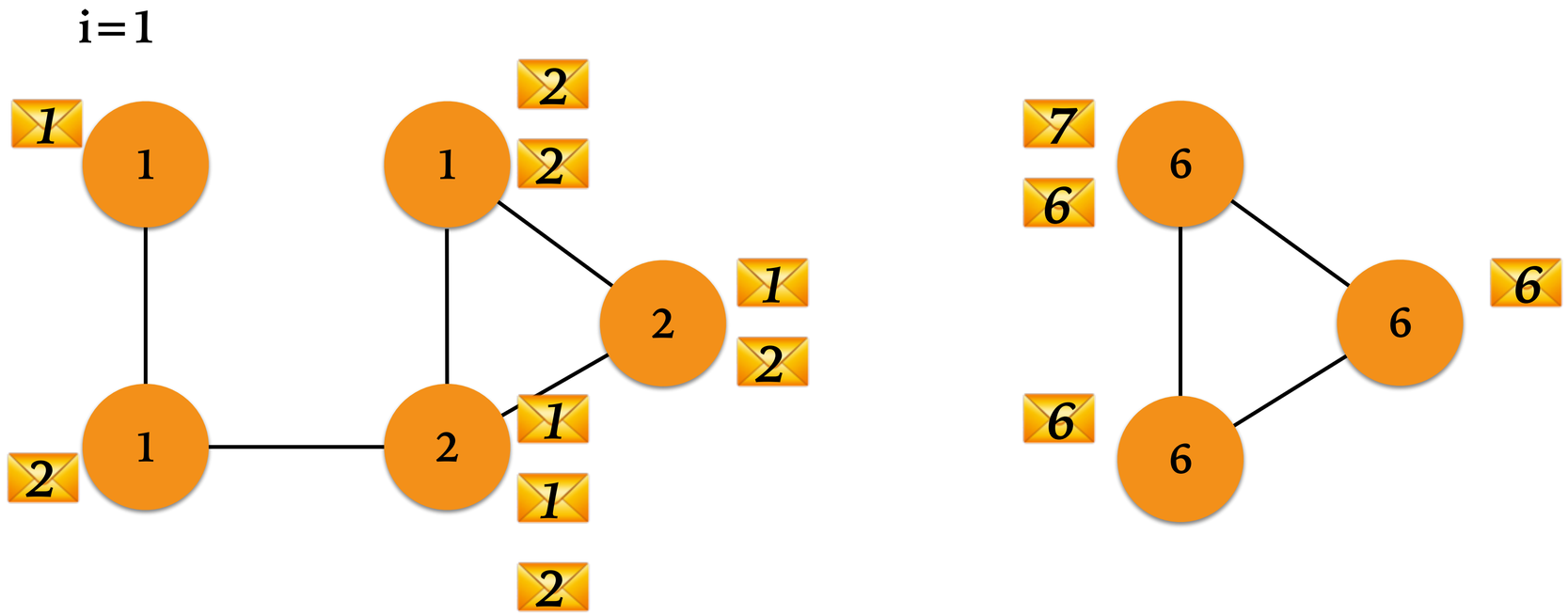
BATCH CONNECTED COMPONENTS



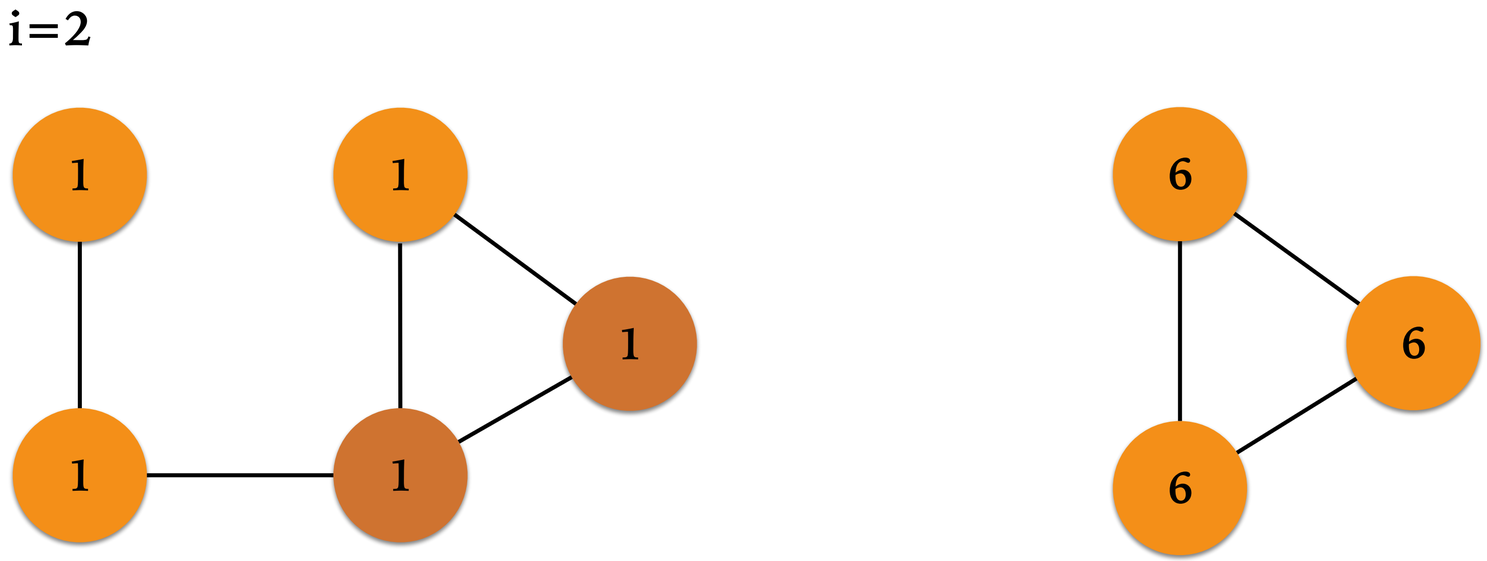
BATCH CONNECTED COMPONENTS

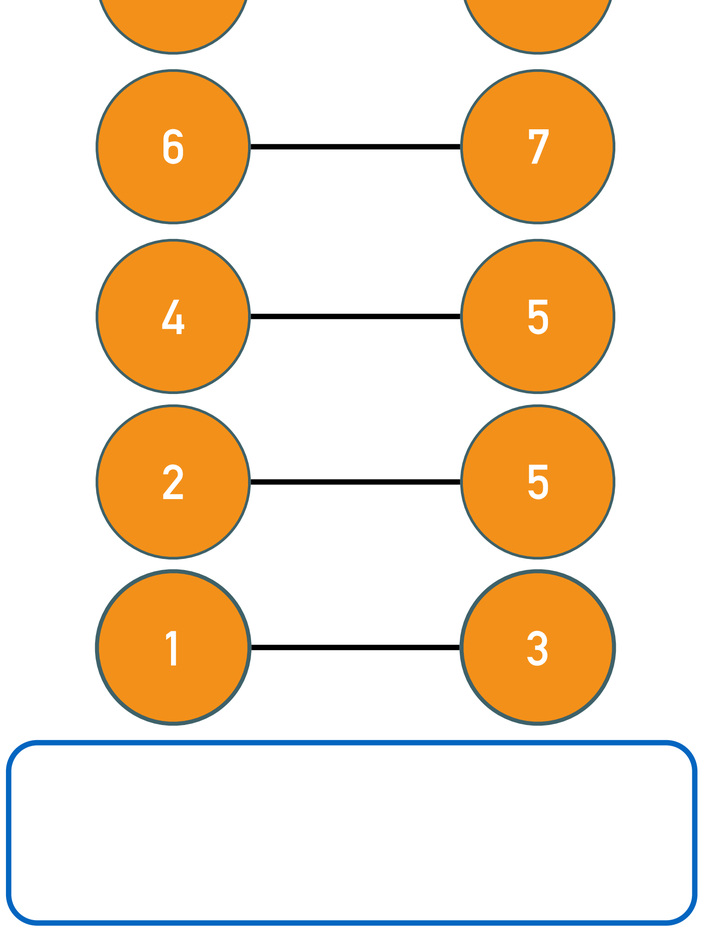


BATCH CONNECTED COMPONENTS



BATCH CONNECTED COMPONENTS

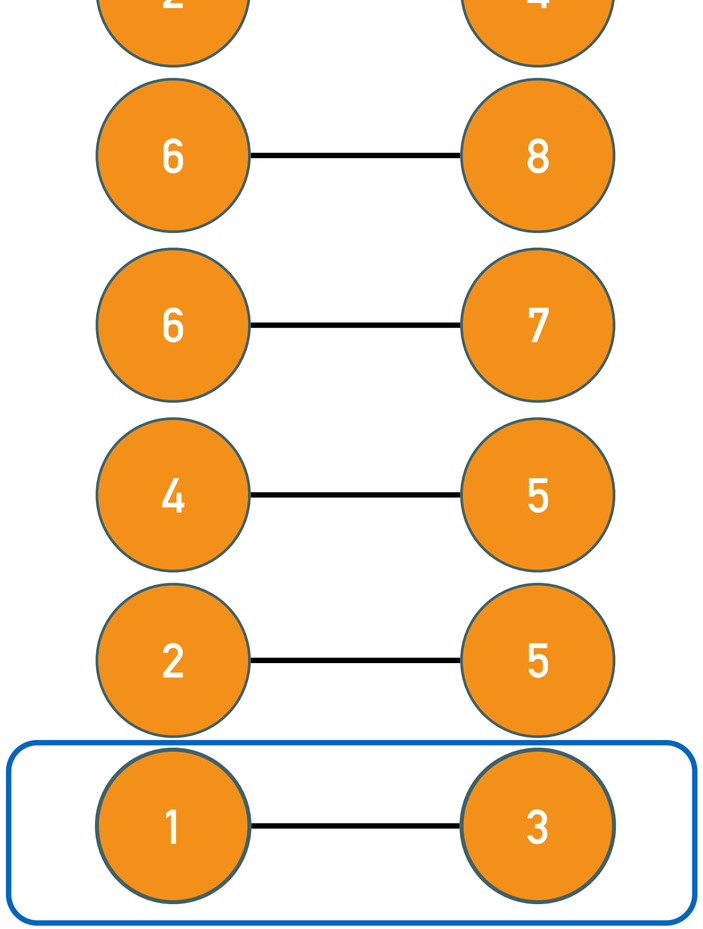


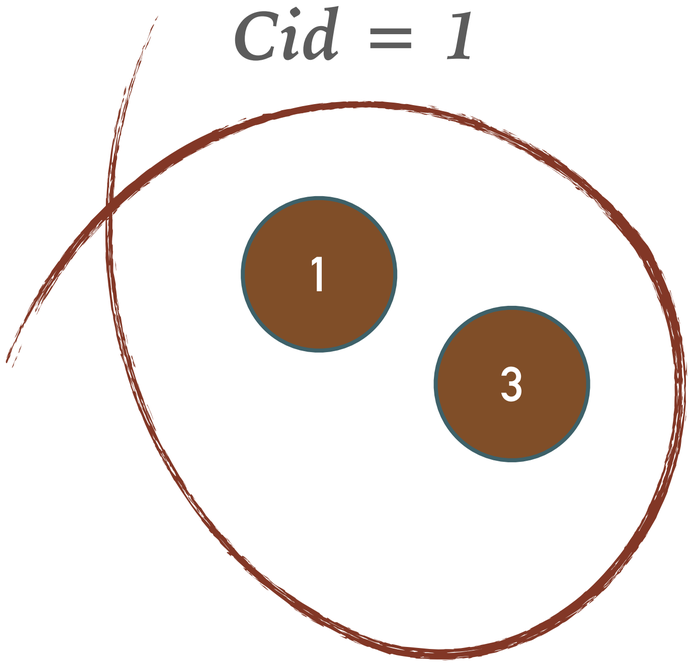


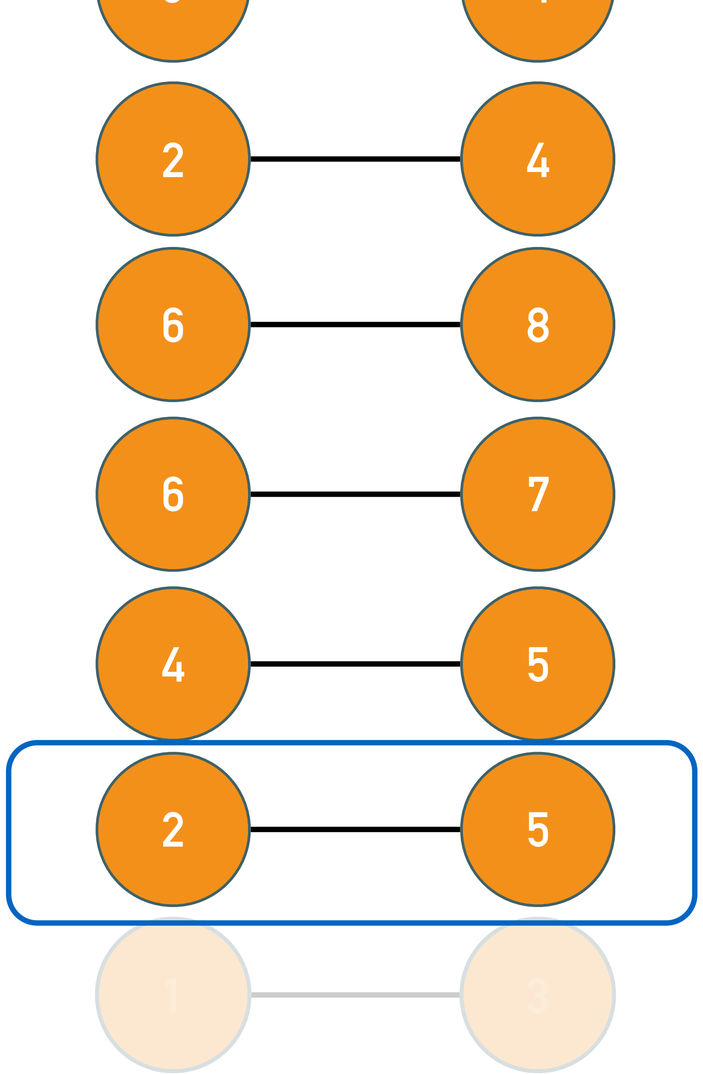
STREAM CONNECTED COMPONENTS

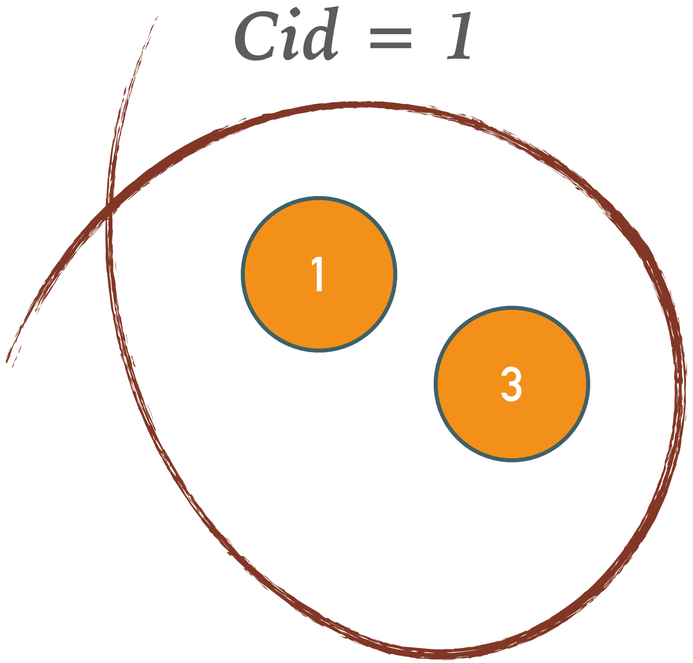
Graph Summary: **Disjoint Set** (Union-Find)

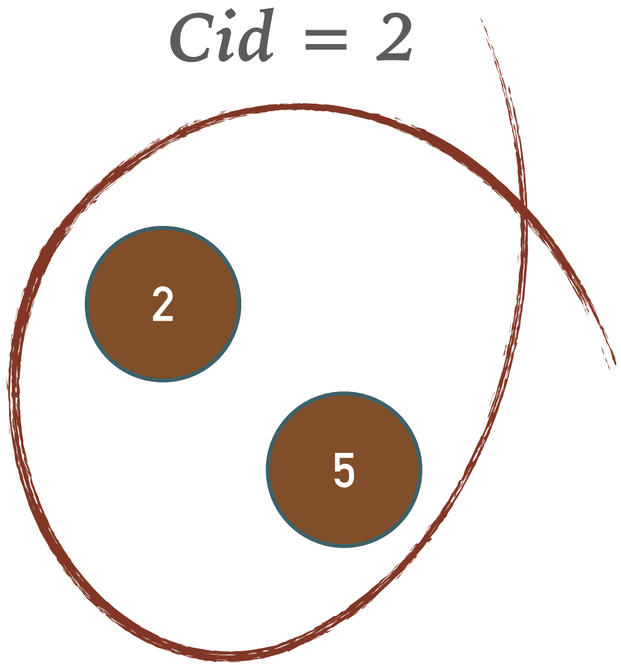
➤ Only store component IDs and vertex IDs

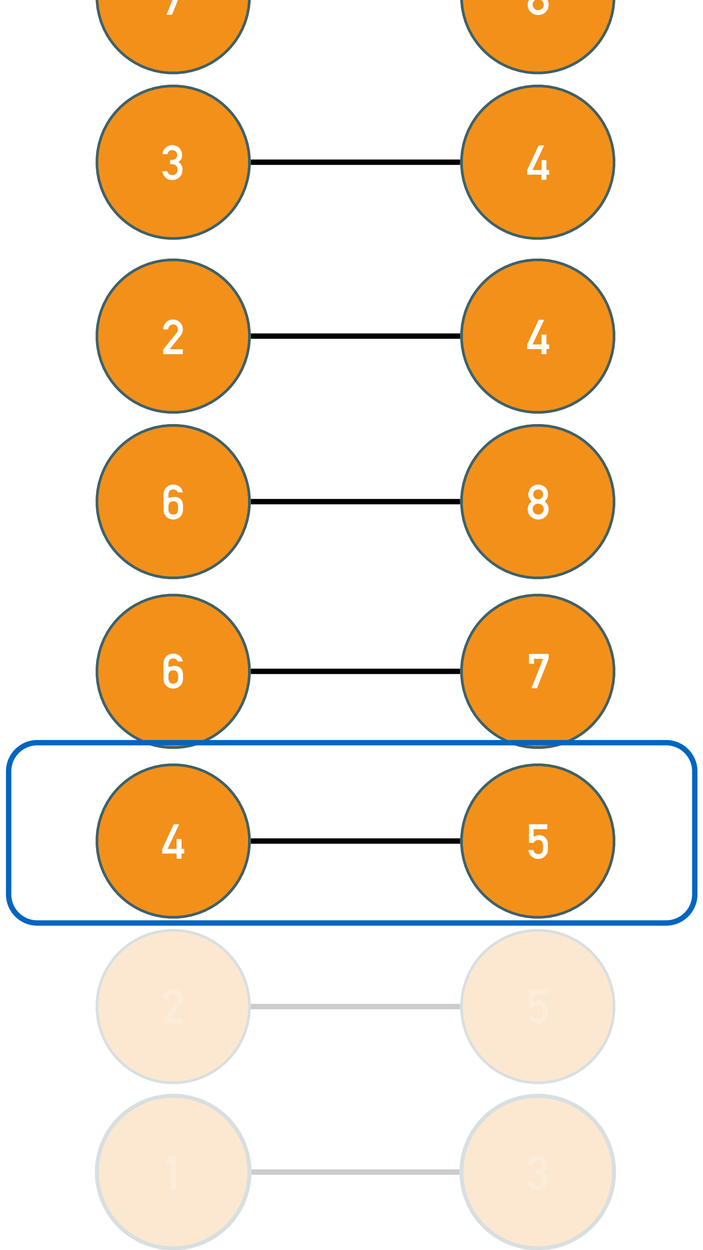

6
8
6
7
4
5
2
5
1
3

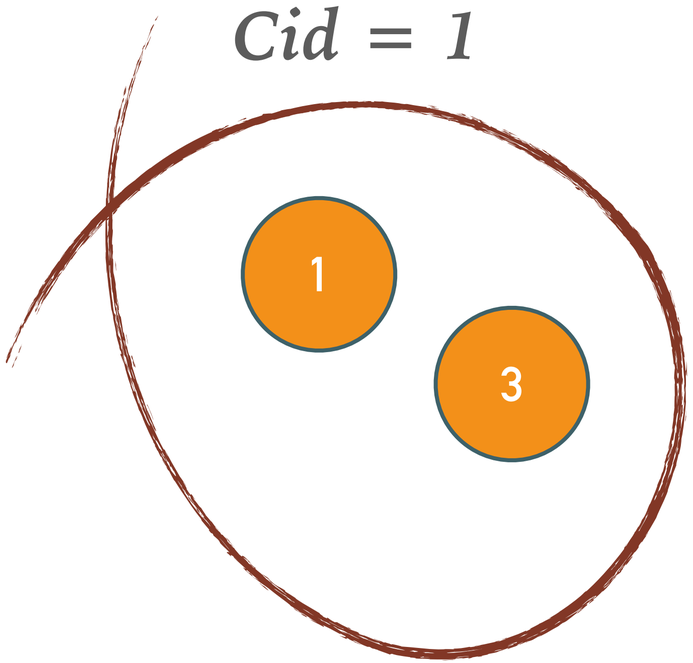

Cid = 1
1
3

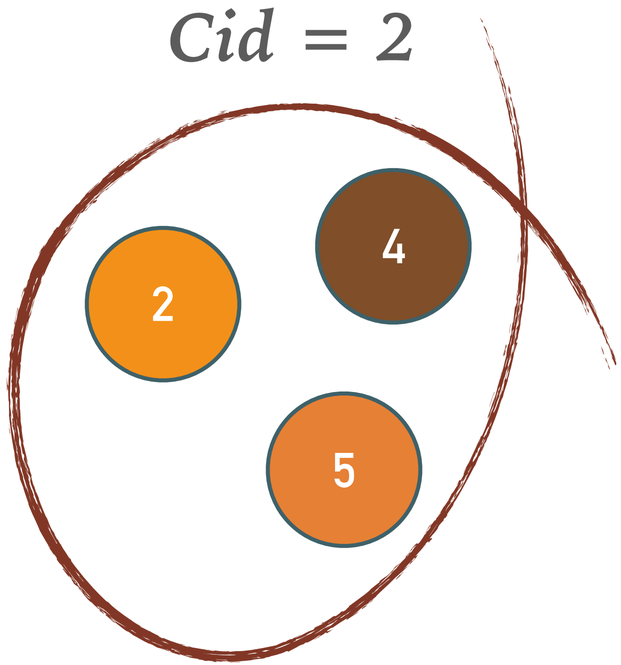

2
4
6
8
6
7
4
5
2
5
1
3

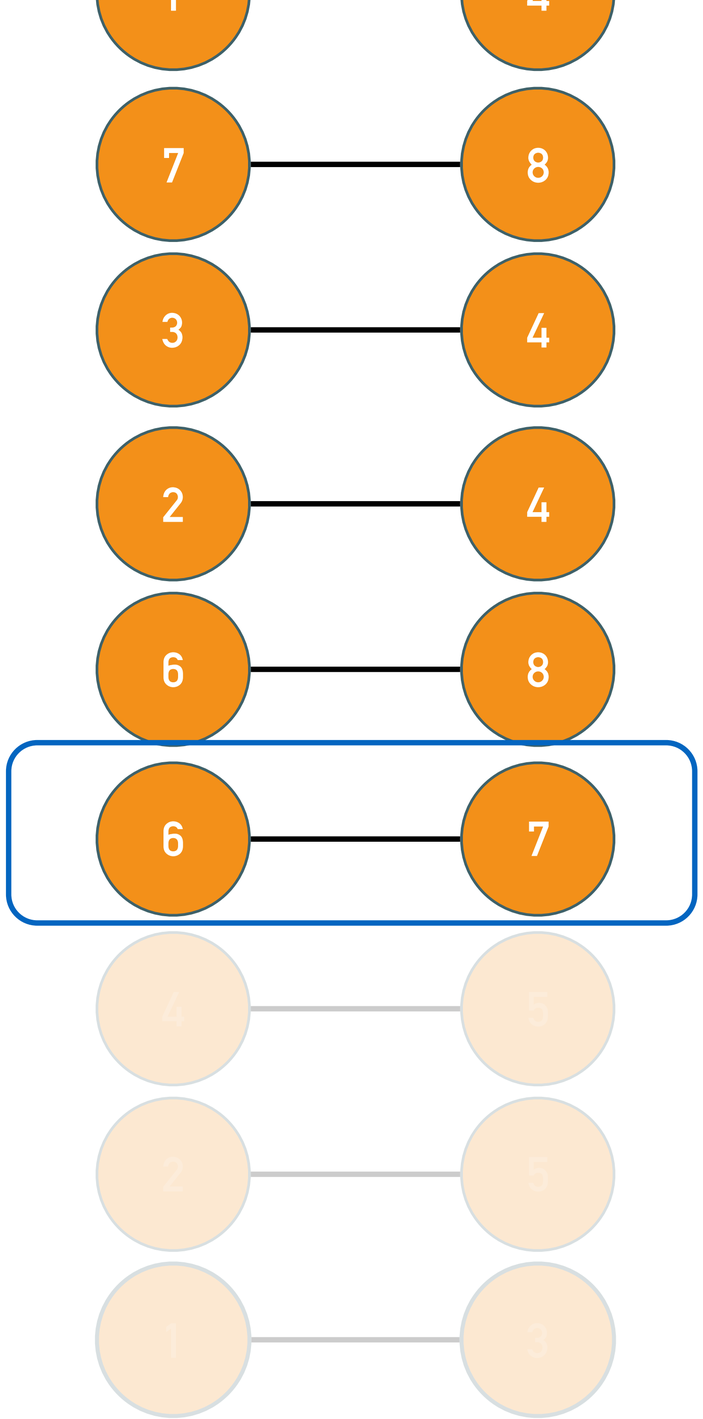

Cid = 1
1
3

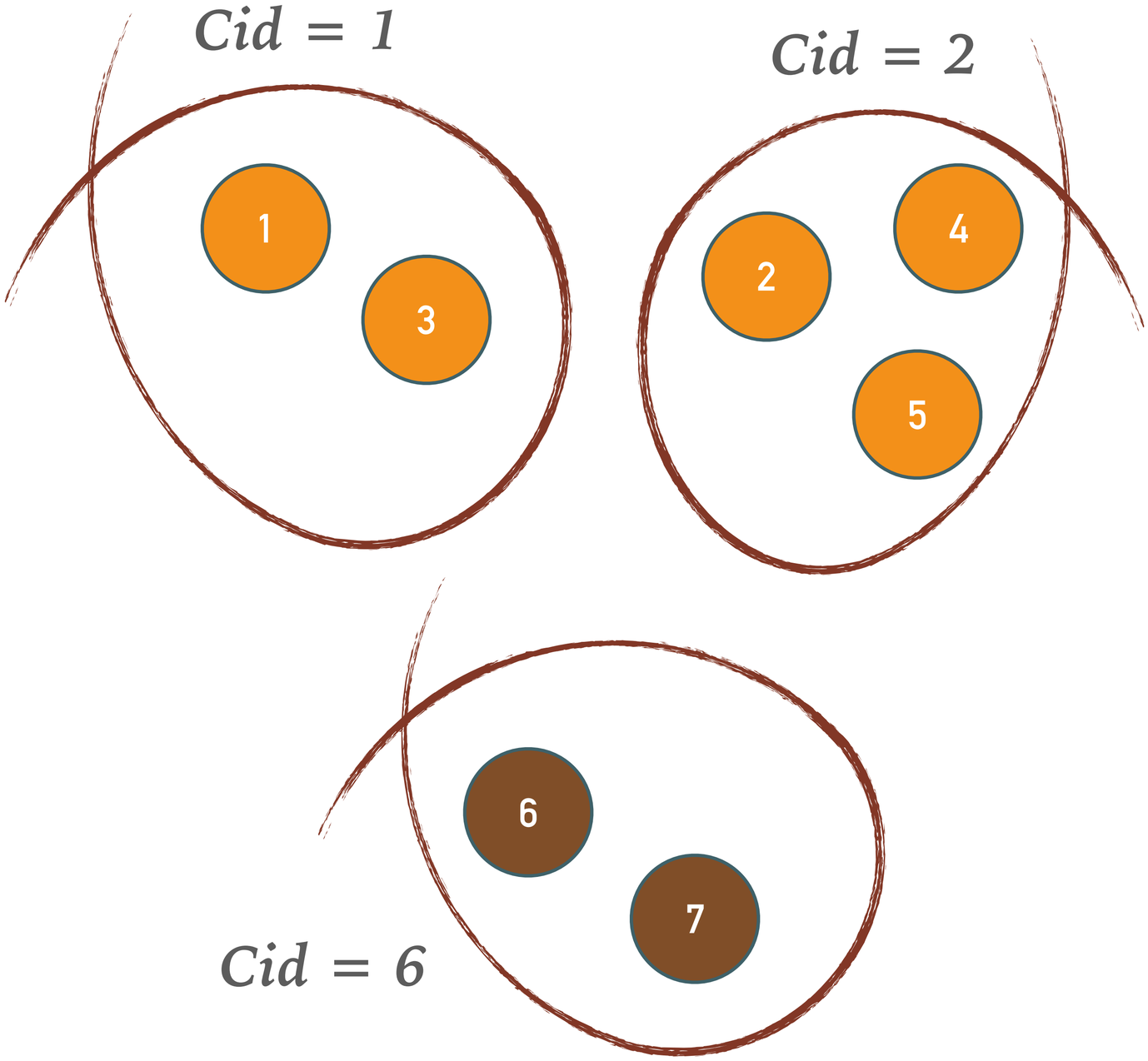

Cid = 2
2
5

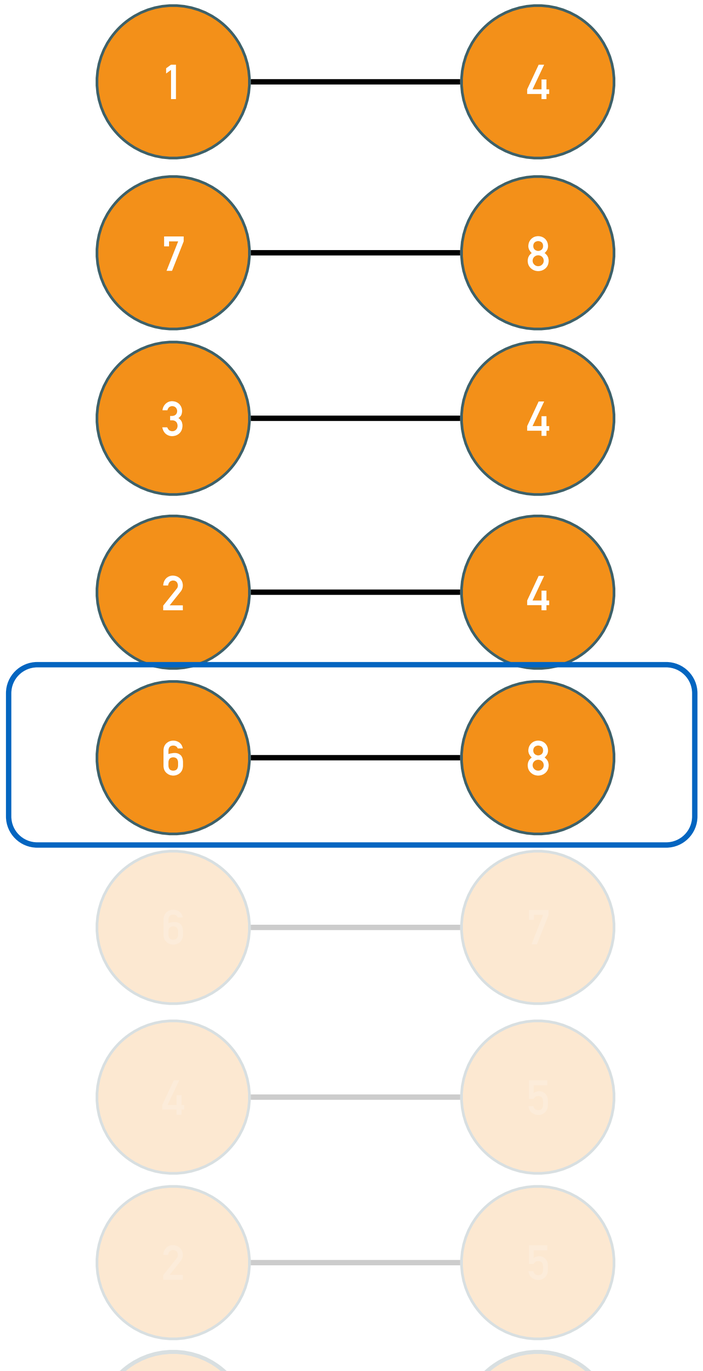

3
4
2
4
6
8
6
7
4
5
2
5
1
3

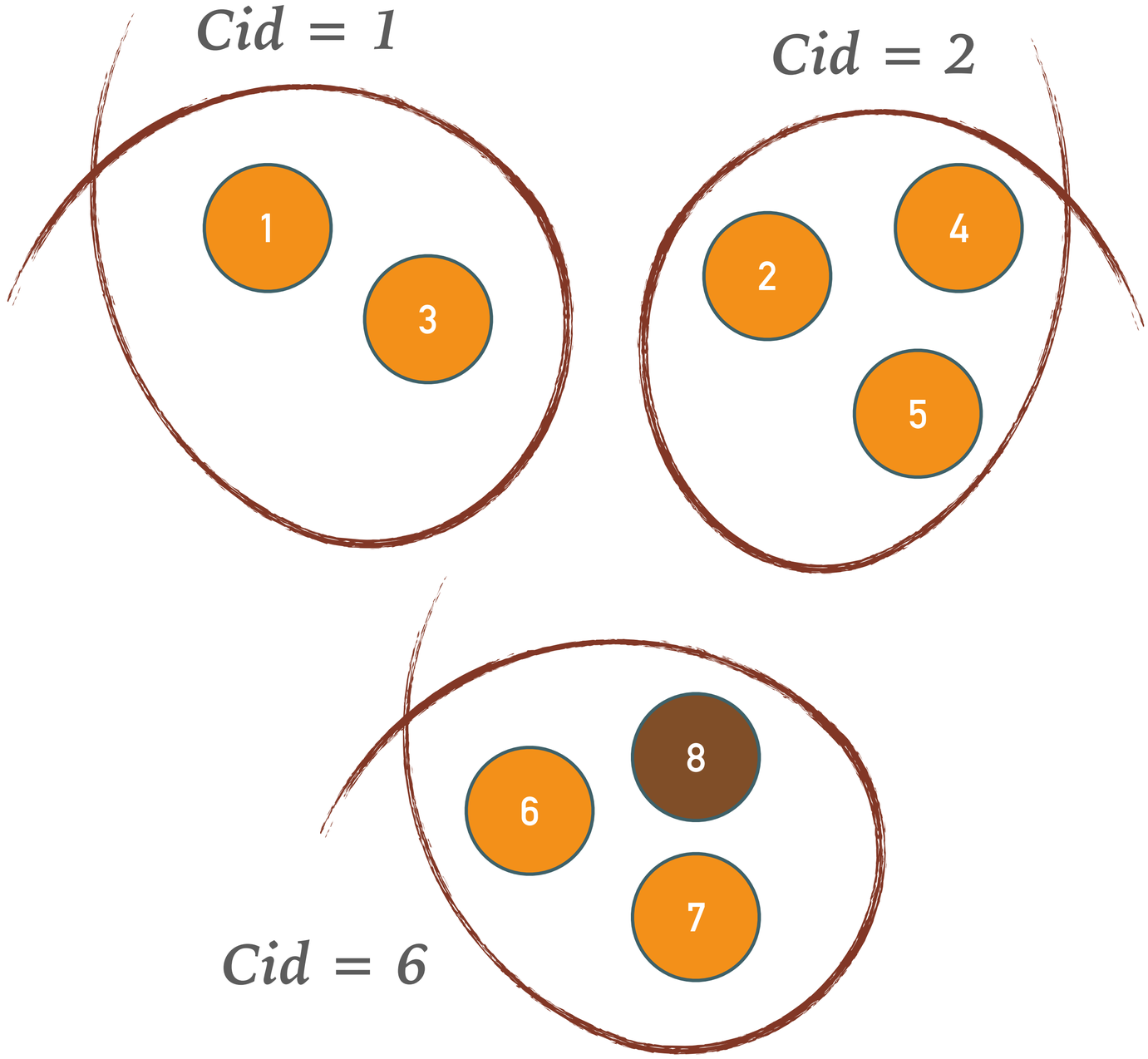

Cid = 1
1
3

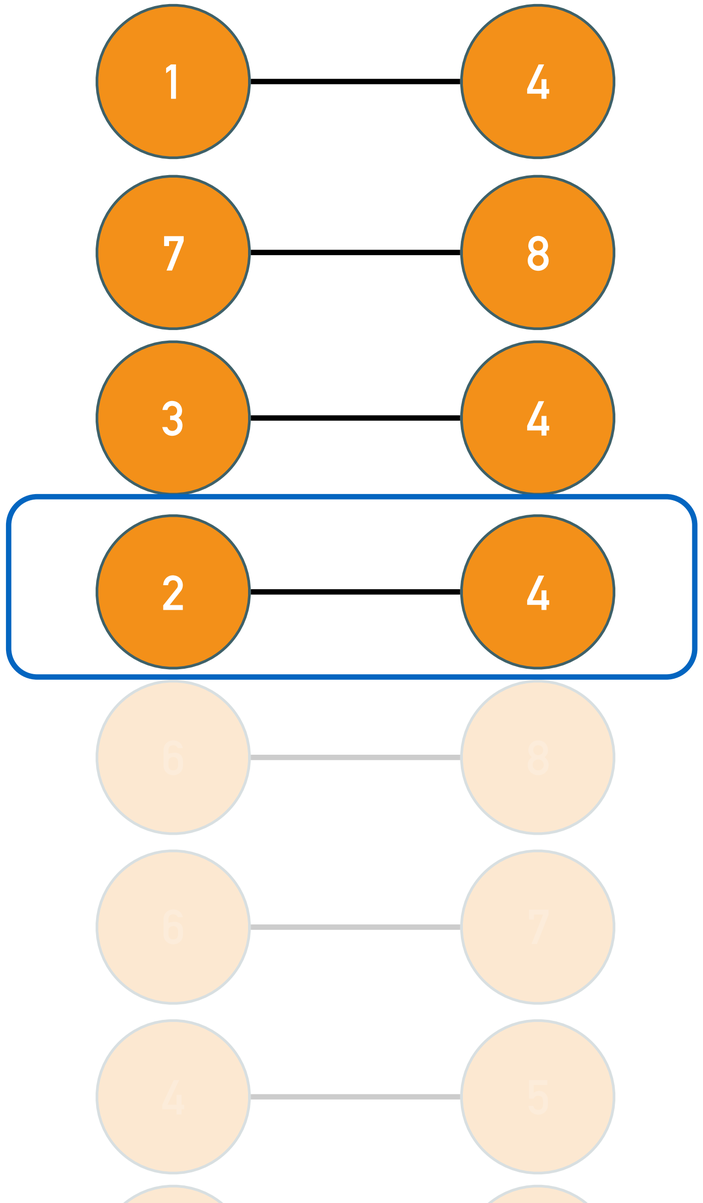

Cid = 2
2
4
5

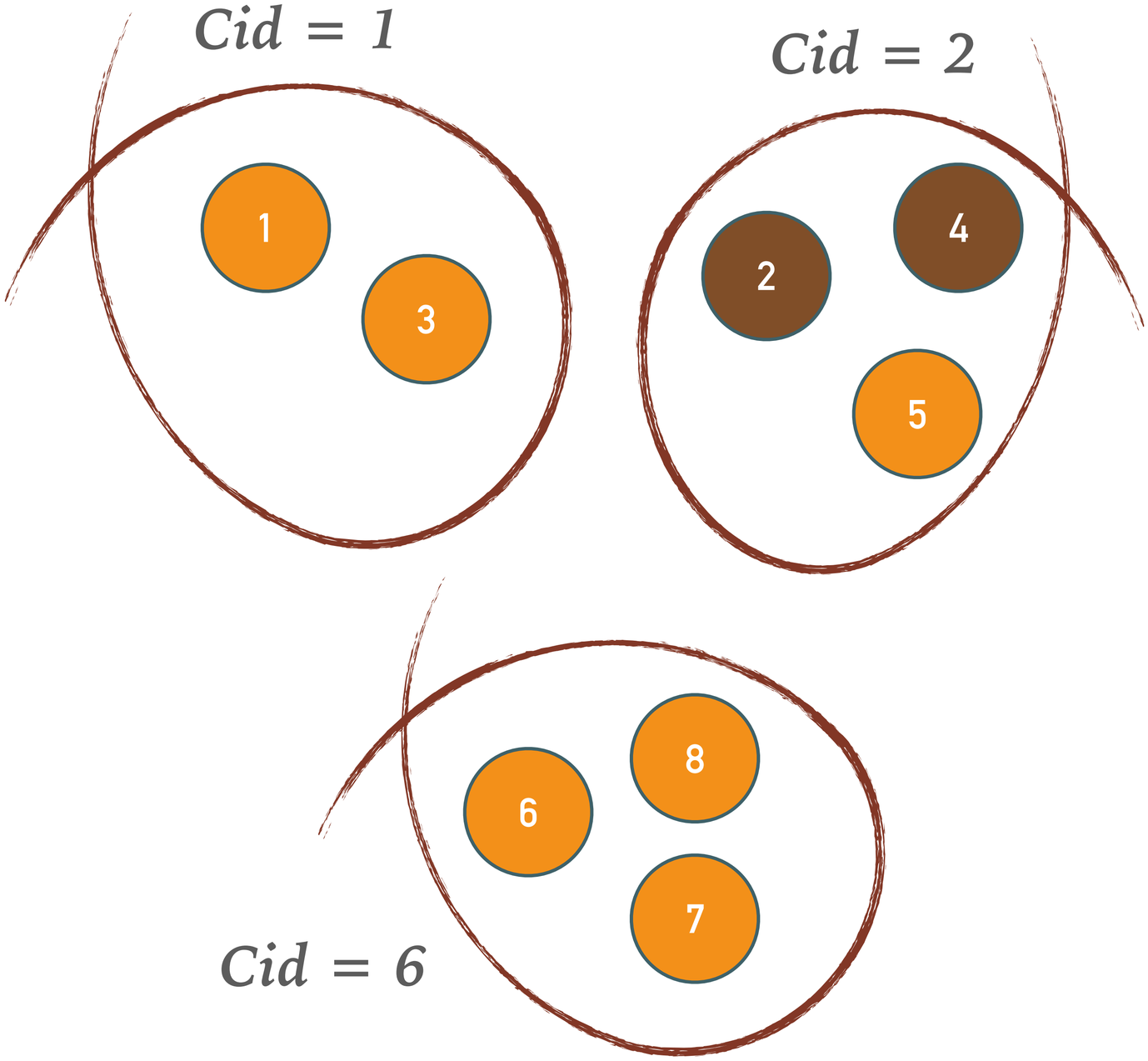

7
8
3
4
2
4
6
8
6
7
4
5
2
5
1
3

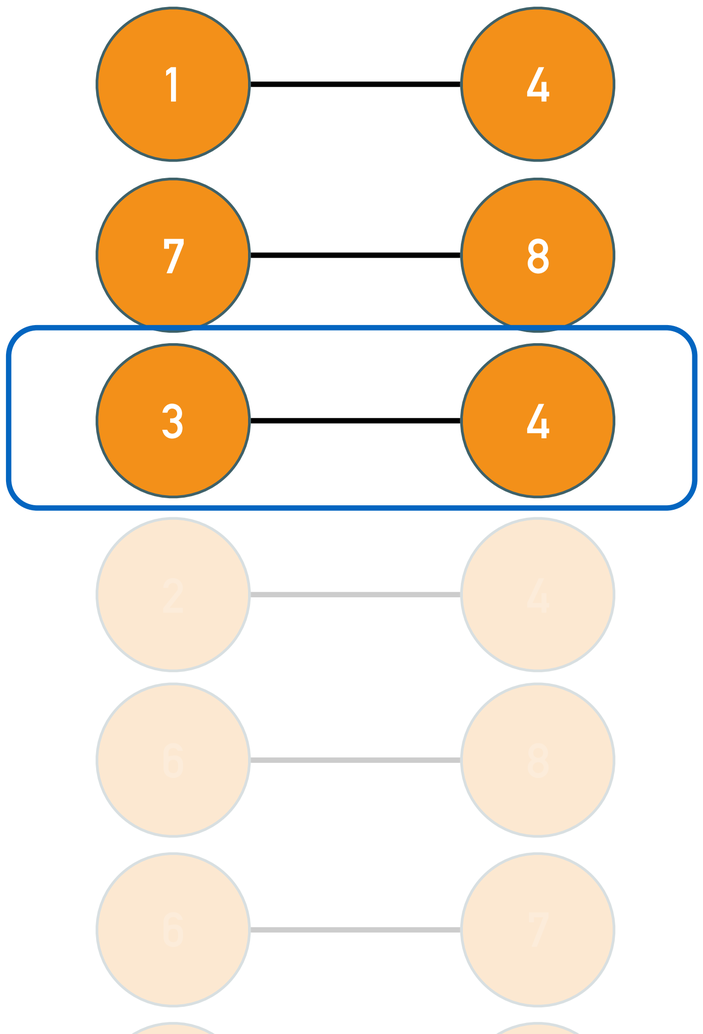

Cid = 1
1
3
Cid = 2
2
4
5
6
7
Cid = 6

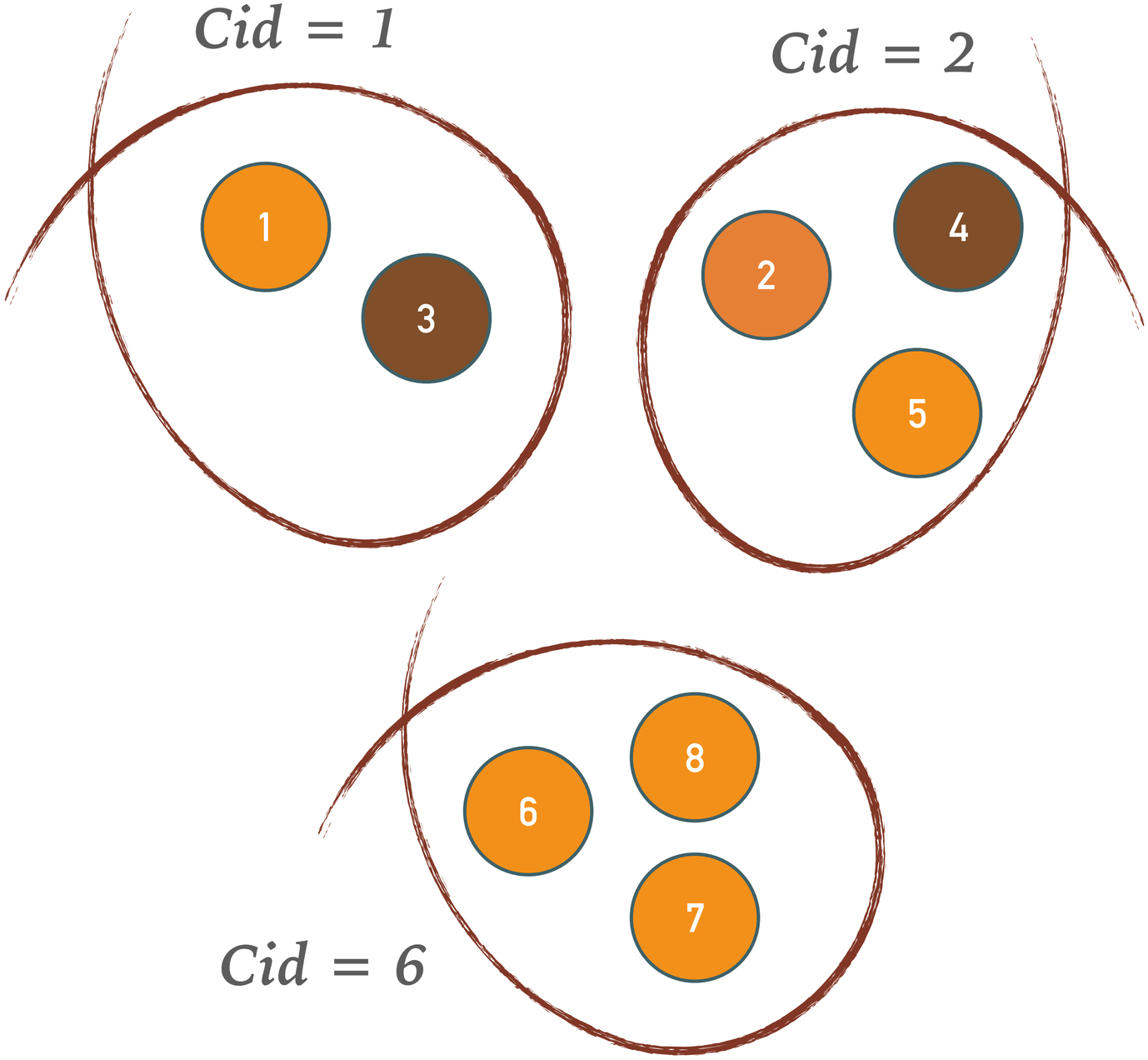

1
4
7
8
3
4
2
4
6
8
6
7
4
5
2
5

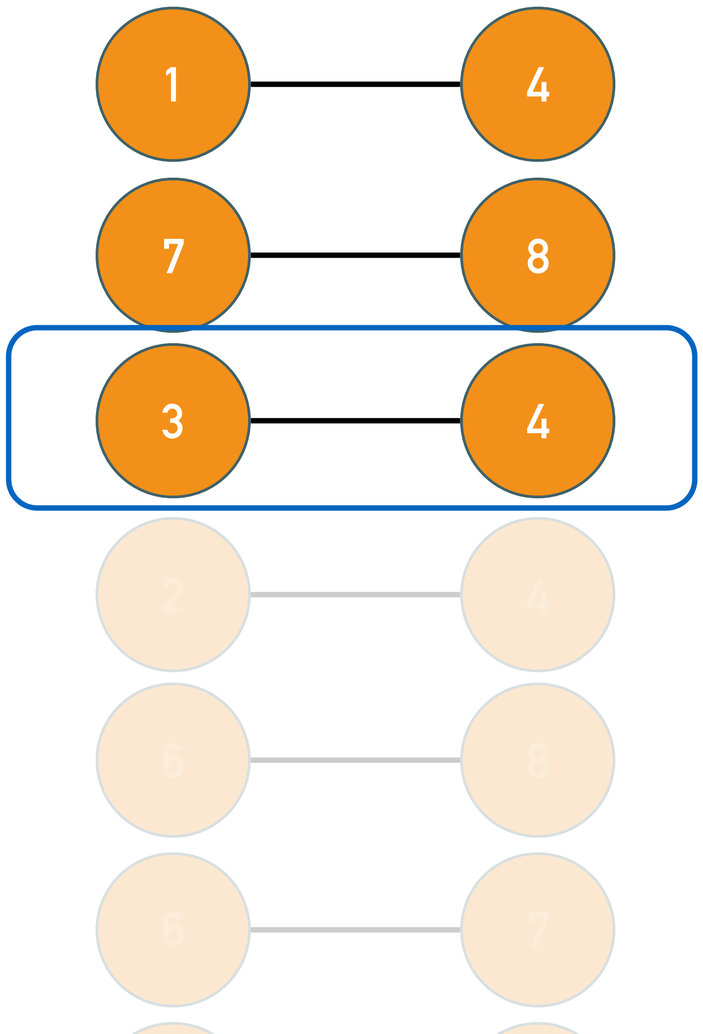

Cid = 1
1
3
Cid = 2
2
4
5
Cid = 6
6
8
7

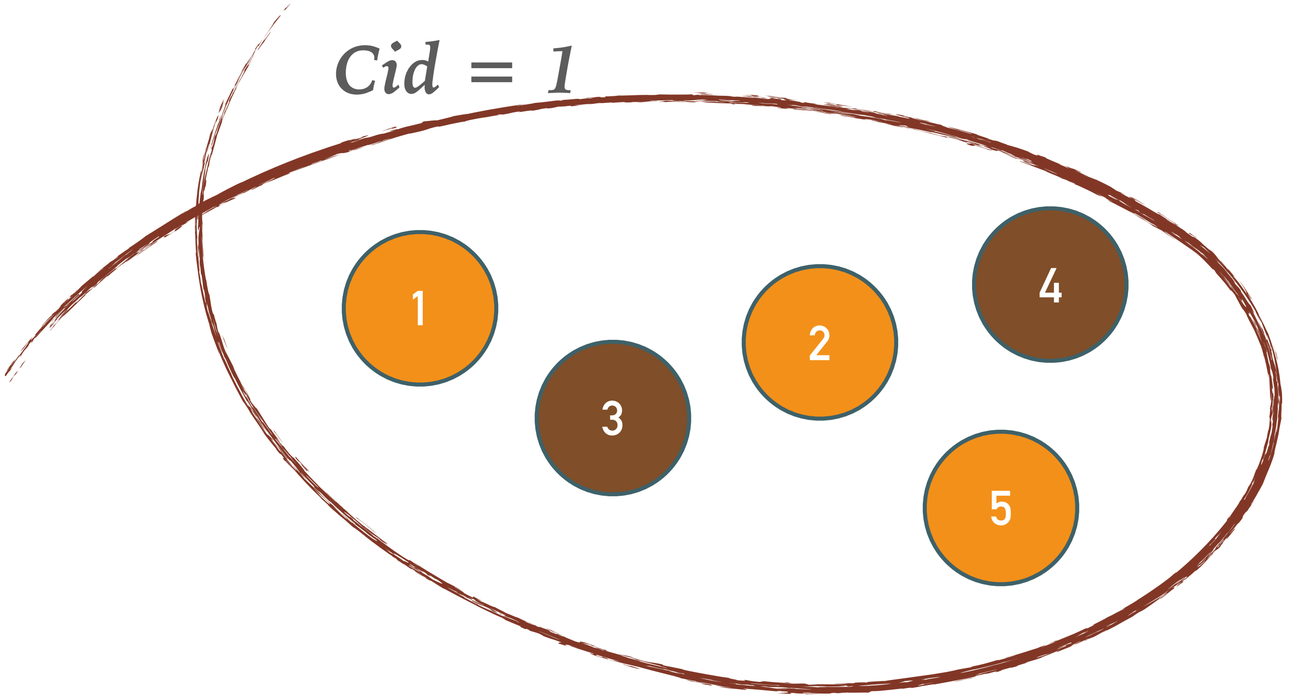

1
4
7
8
3
4
2
4
6
8
6
7
4
5

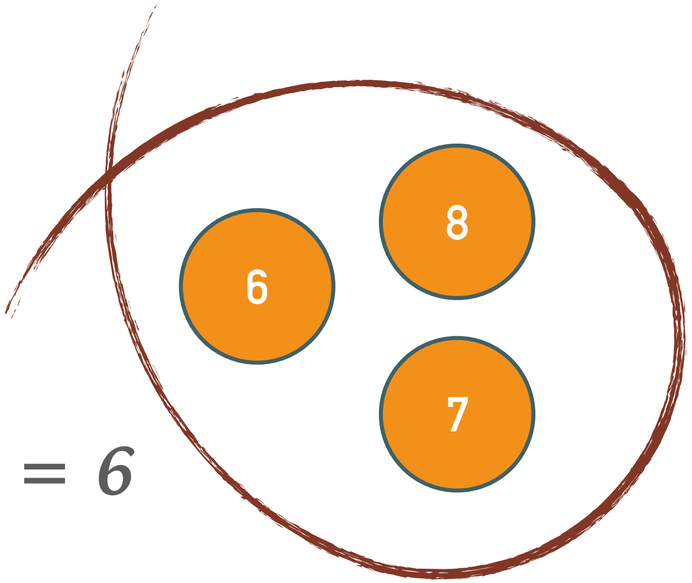

Cid = 1
1
3
Cid = 2
2
4
5
Cid = 6
6
8
7


1
4
7
8
3
4
2
4
6
8
6
7


Cid = 1
1
3
Cid = 2
2
4
5
Cid = 6
6
8
7

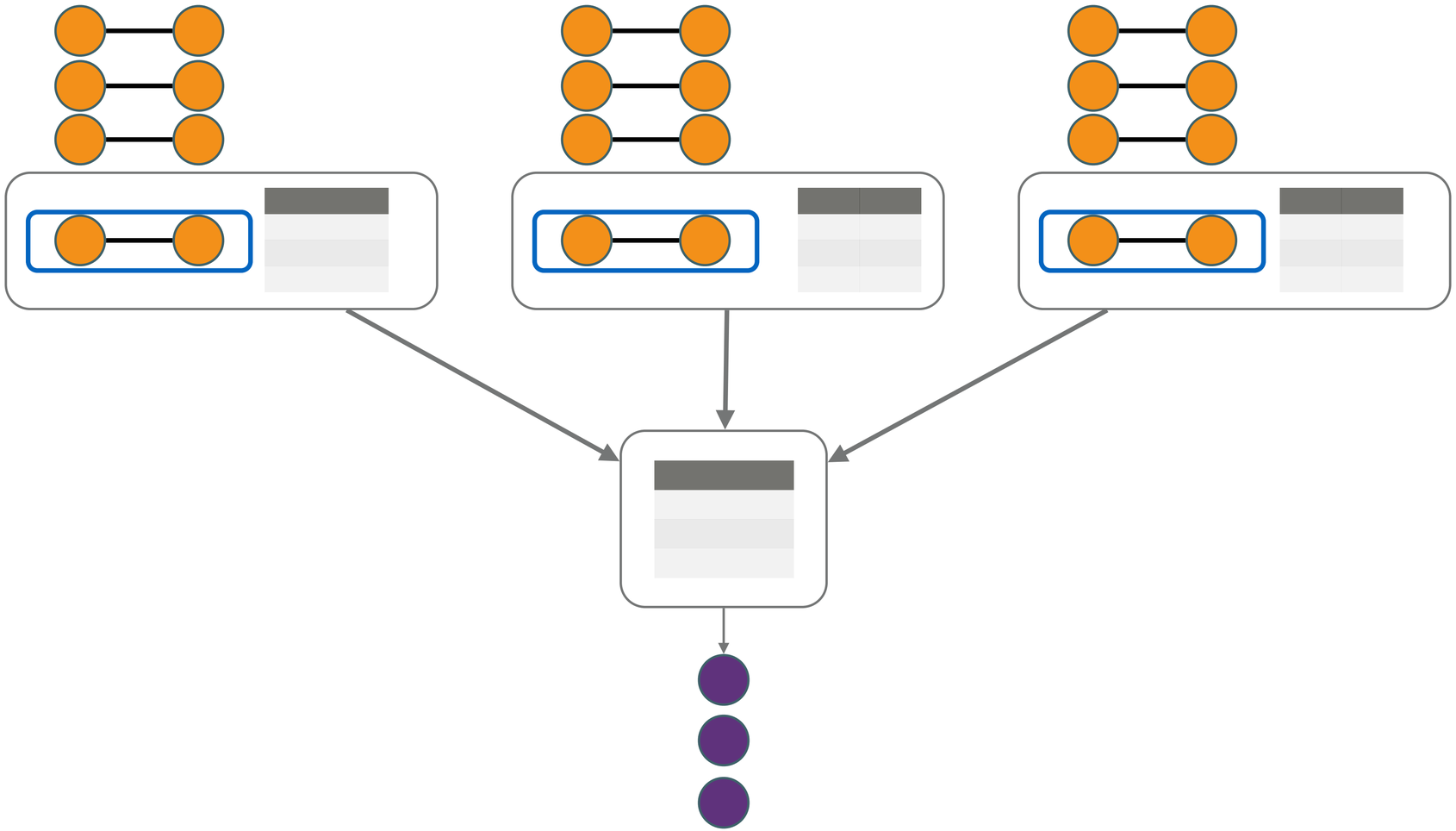






Cid = 6

DISTRIBUTED STREAM CONNECTED COMPONENTS



THE BAD NEWS

- A *slightly* different motivation
  - finite graph stored in disk vs. unbounded graph arriving in real-time
  - some algorithms assume we know |V|, |E|
  - most algorithms designed for single-node execution

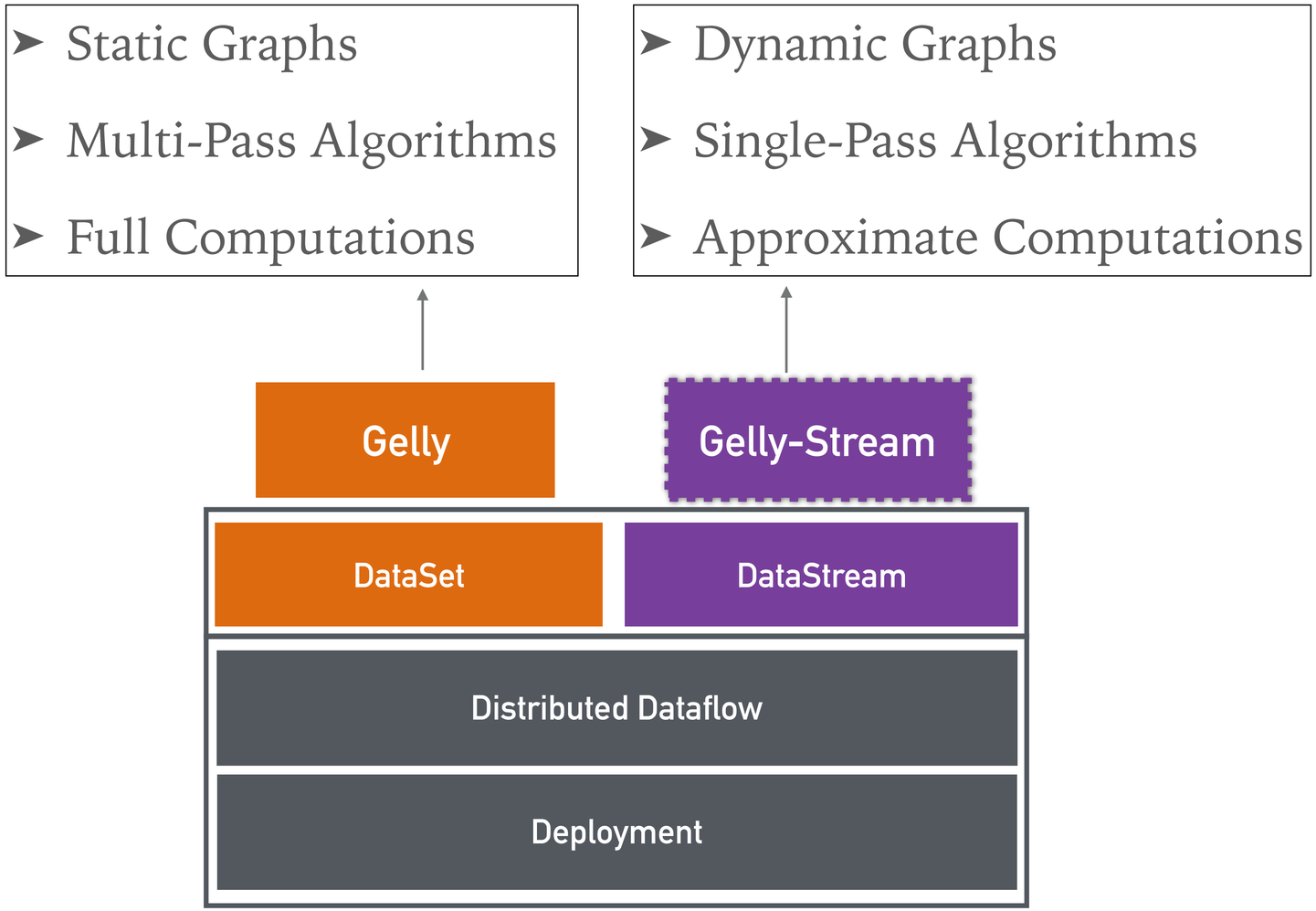
THE GOOD NEWS

- A *quite* different reality
  - memory is getting bigger
  - … and cheaper
  - we know how to design distributed algorithms

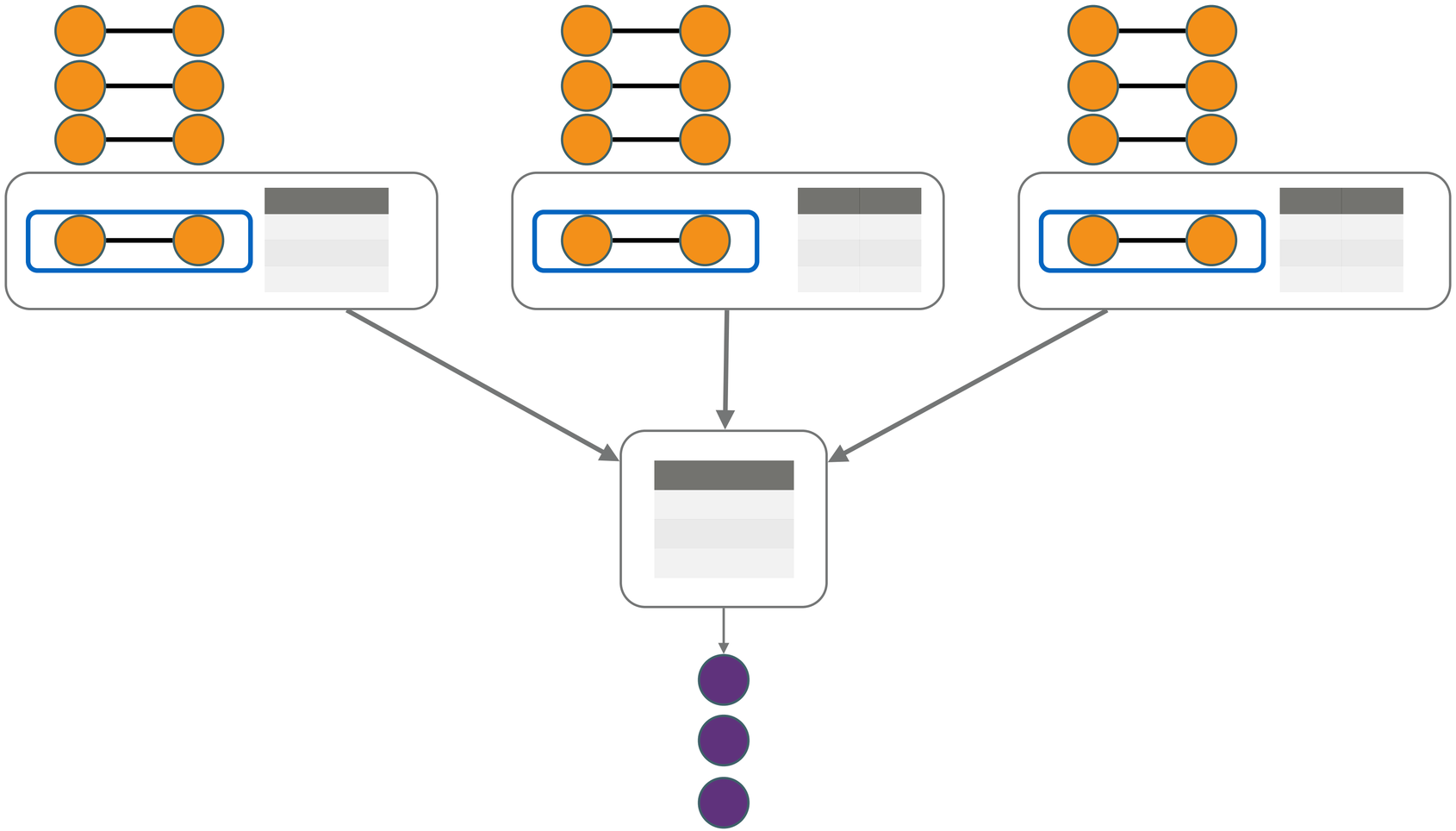
GELLY-STREAM

SINGLE-PASS STREAM GRAPH PROCESSING WITH APACHE FLINK

GELLY ON STREAMS



DISTRIBUTED STREAM CONNECTED COMPONENTS



STREAM CONNECTED COMPONENTS WITH FLINK

```
DataStream<DisjointSet> cc =
   edgeStream
      .keyBy(0)
      .timeWindow(Time.of(100, TimeUnit.MILLISECONDS))
      .fold(new DisjointSet(), new UpdateCC())
      .flatMap(new Merger())
      .setParallelism(1);
```

STREAM CONNECTED COMPONENTS WITH FLINK

```
DataStream<DisjointSet> cc =
   edgeStream
      .keyBy(0)
      .timeWindow(Time.of(100, TimeUnit.MILLISECONDS))
      .fold(new DisjointSet(), new UpdateCC())
      .flatMap(new Merger())
      .setParallelism(1);
```

Partition the edge stream

STREAM CONNECTED COMPONENTS WITH FLINK

```
DataStream<DisjointSet> cc =
   edgeStream
      .keyBy(0)
      .timeWindow(Time.of(100, TimeUnit.MILLISECONDS))
      .fold(new DisjointSet(), new UpdateCC())
      .flatMap(new Merger())
      .setParallelism(1);
```

Define the merging frequency

STREAM CONNECTED COMPONENTS WITH FLINK

```
DataStream<DisjointSet> cc =
   edgeStream
      .keyBy(0)
      .timeWindow(Time.of(100, TimeUnit.MILLISECONDS))
      .fold(new DisjointSet(), new UpdateCC())
      .flatMap(new Merger())
      .setParallelism(1);
```

merge locally

STREAM CONNECTED COMPONENTS WITH FLINK

```
DataStream<DisjointSet> cc =
   edgeStream
      .keyBy(0)
      .timeWindow(Time.of(100, TimeUnit.MILLISECONDS))
      .fold(new DisjointSet(), new UpdateCC())
      .flatMap(new Merger())
      .setParallelism(1);
```

merge globally

GELLY-STREAM STATUS

- Properties and Metrics
- Transformations
- Aggregations
- Discretization
- Neighborhood Aggregations

- Graph Streaming Algorithms
  - Connected Components
  - Bipartiteness Check
  - Window Triangle Count
  - Triangle Count Estimation
  - Continuous Degree Aggregate

FEELING GELLY?

➤ **Gelly-Stream Repository**

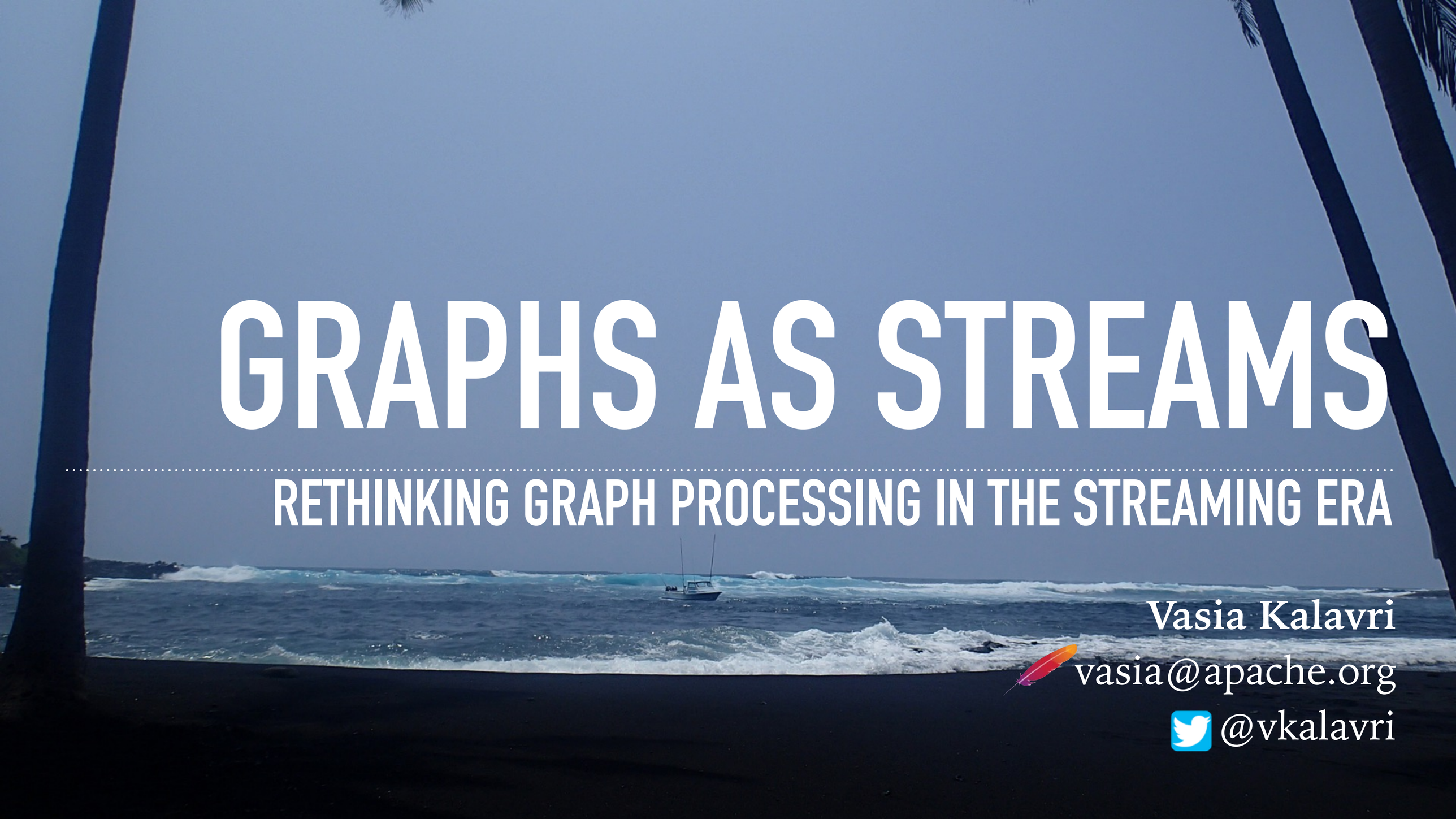
github.com/vasia/gelly-streaming

➤ **A list of graph streaming papers**

citeulike.org/user/vasiakalavri/tag/graph-streaming

➤ **A related talk at FOSDEM'16**

slideshare.net/vkalavri/gellystream-singlepass-graph-streaming-analytics-with-apache-flink



GRAPHS AS STREAMS

RETHINKING GRAPH PROCESSING IN THE STREAMING ERA

Vasia Kalavri
vasia@apache.org
@vkalavri