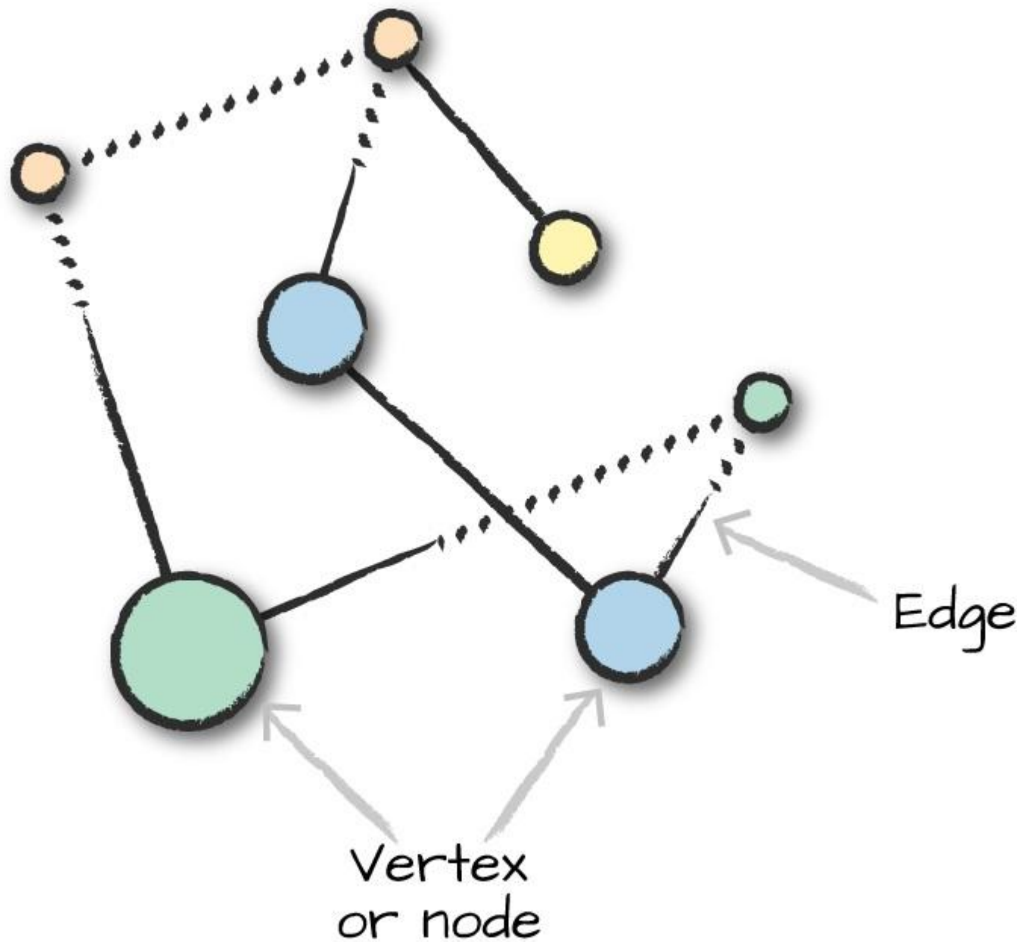
Graph Analytics in Spark

Graph analytics: Introduction

Graph analytics

- Graphs are data structures composed of nodes and edges
  - Nodes/vertexes are denoted as $V=\{v_1,v_2,\ldots,v_n\}$ and edges are denoted as $E=\{e_1,e_2,\ldots,e_n\}$
  - Graph analytics is the process of analyzing relationships between vertexes and edges

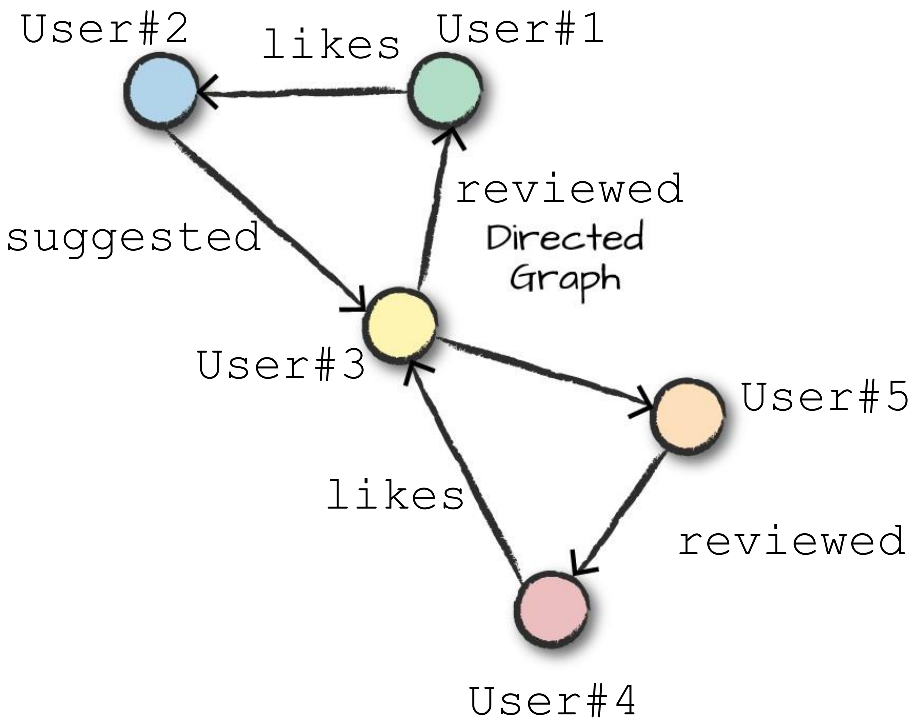
Graph analytics



Vertexes, edges and weights

- Graphs are undirected if edges do not have a direction
- Otherwise they are called directed graphs
- Vertexes and edges can have data associated with them
  - weight/label
    - e.g., an edge weight may represent the strength of the relationship
    - e.g., a vertex label may be the string associated with the name of the vertex

Vertexes, edges and weights

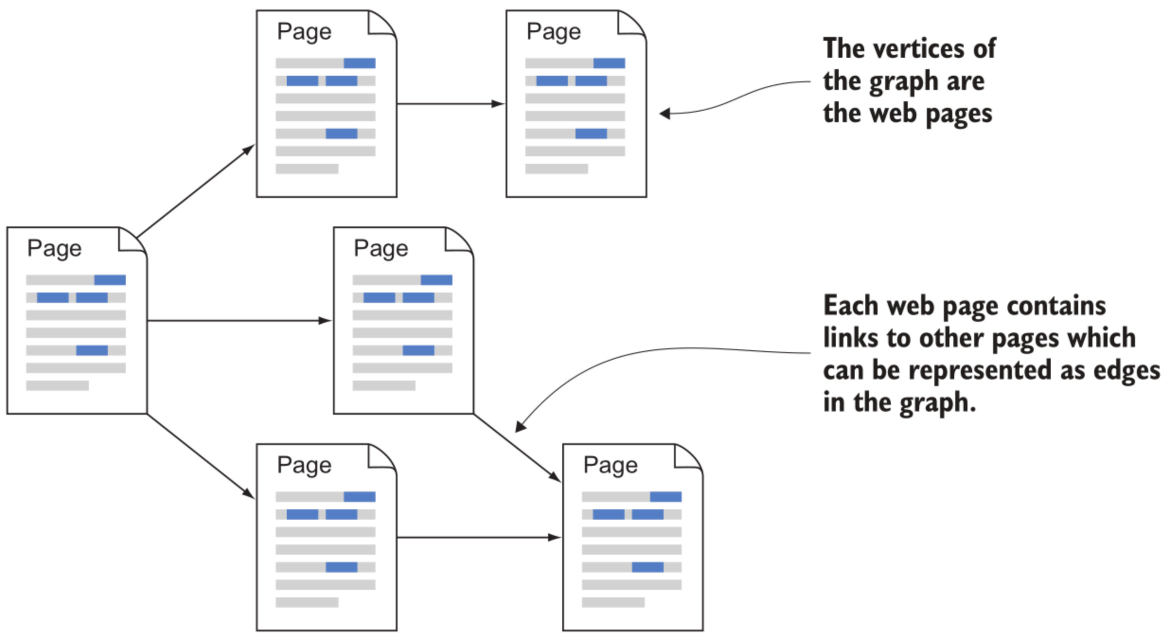


Why graph analytics?

- Graphs are natural way of describing relationships
- Practical example of analytics over graphs
  - Ranking web pages (Google PageRank)
  - Detecting group of friends
  - Determine importance of infrastructure in electrical networks
  - ...

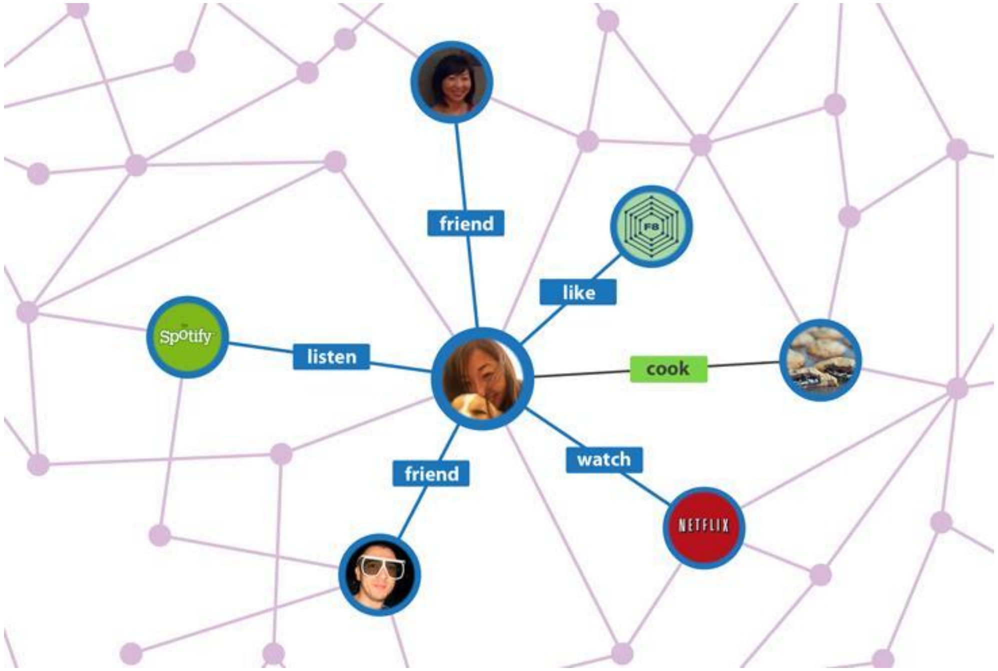
Graph structure in the web

- Importance and rank of web pages



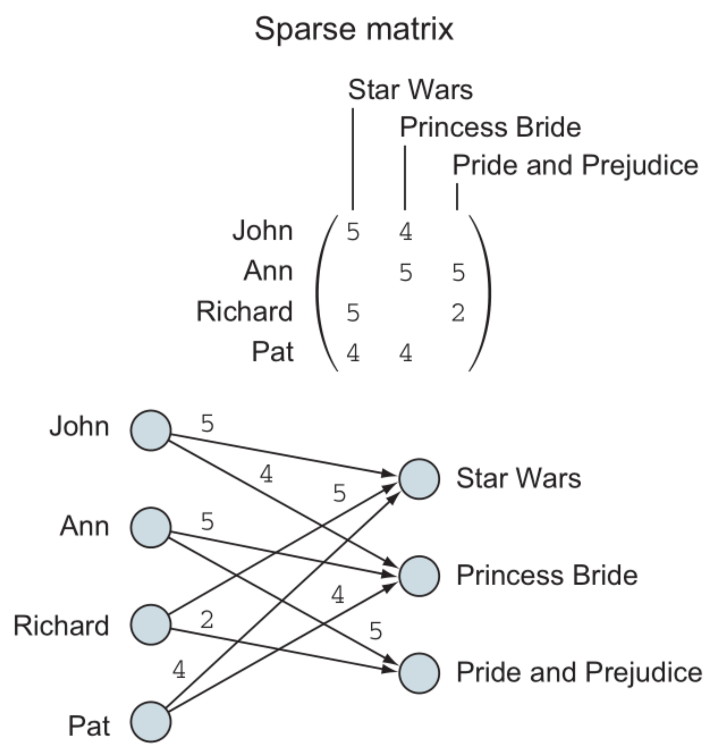
Graph structure in the web

- Social network structure and web usage



Graph structure in the web

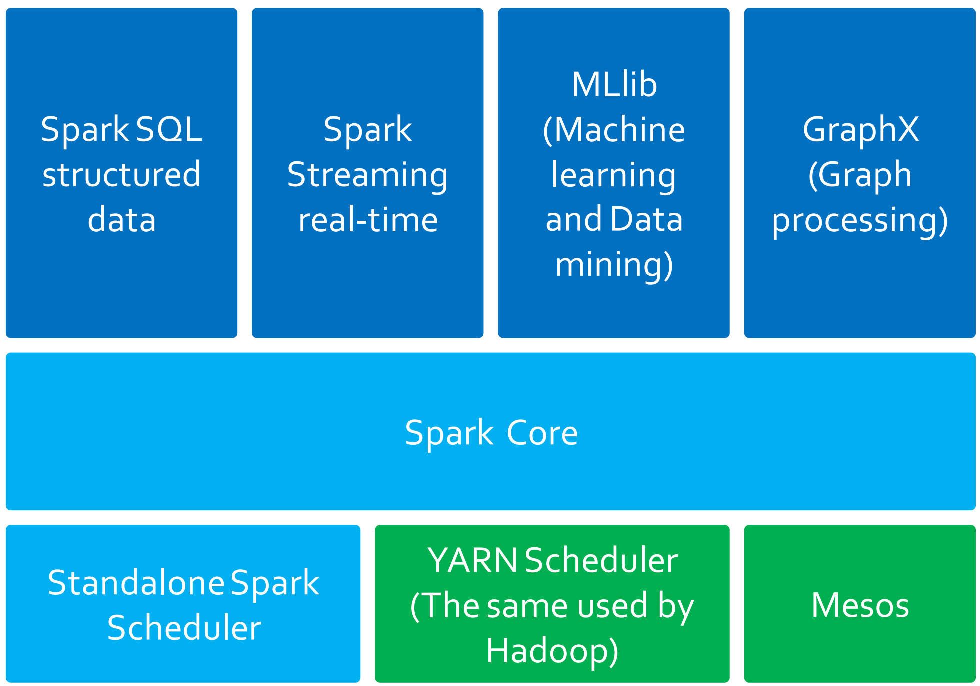
- Movies watched by users



Spark GraphX and GraphFrames

GraphX

- Spark RDD-based library for performing graph processing
- Core part of Spark



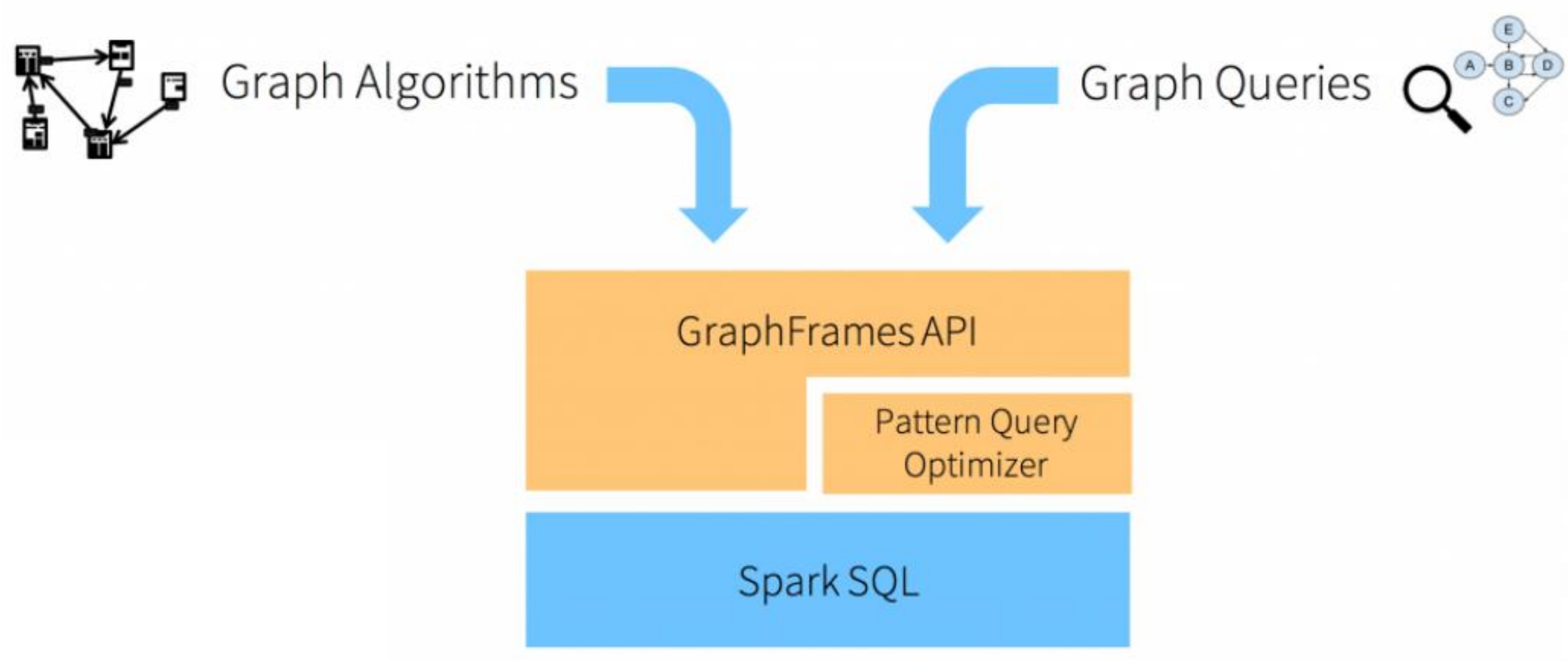
GraphX

- Low level interface with RDD
- Very powerful
  - Many application and libraries built on top of it
- However, not easy to use or optimize
- No Python version of the APIs

GraphFrames

- Library DataFrame-based for performing graph processing
- Spark external package built on top of GraphX
  - https://graphframes.github.io/graphframes/docs/_site/index.html

GraphFrames



Building and querying graphs with GraphFrames

Building a graph

- Define vertexes and edges of the graph
  - Vertexes and edges are represented by means of records inside DataFrames with specifically named columns
    - One DataFrame for the definition of the vertexes of the graph
    - One DataFrame for the definition of the edges of the graph

Building a graph

- The DataFrames that are used to represent nodes/vertexes
  - Contain **one record per vertex**
  - Must contain a column named "**id**" that stores unique vertex IDs
  - Can contain other columns that are used to characterize vertexes

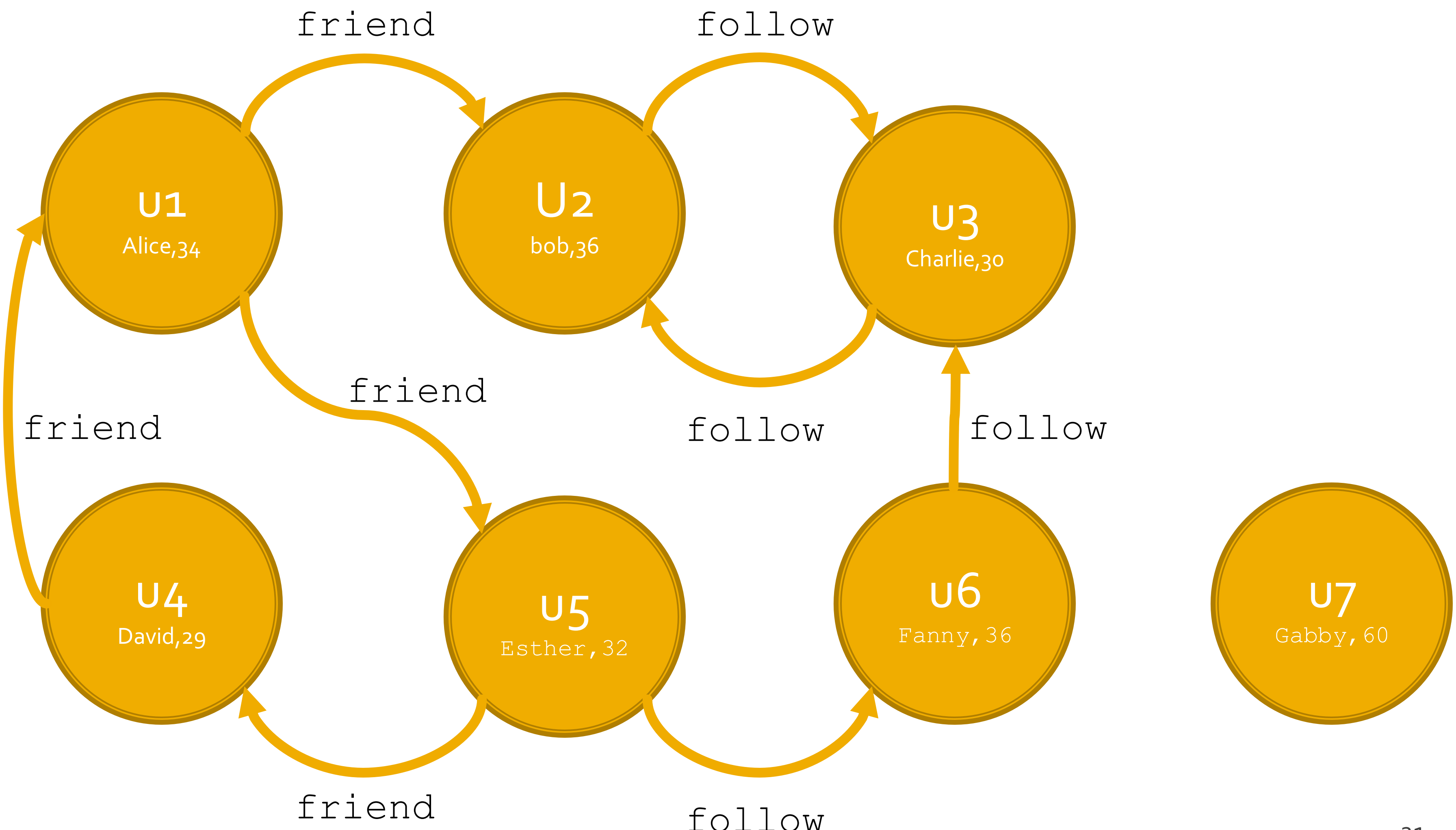
Building a graph

- The DataFrames that are used to represent edges
  - Contain **one record per edge**
  - Must contain two columns "**src**" and "**dst**" storing source vertex IDs and destination vertex IDs of edges
  - Can contain other columns that are used to characterize edges

Building a graph

- Create a graph of type graphframes.graphframe.GraphFrame by invoking the constructor **GraphFrame(v,e)**
  - v
    - The DataFrame containing the definition of the vertexes
  - e
    - The DataFrame containing the definition of the edges
- Graphs in graphframes are **directed graphs**

Building a graph: Example



Building a graph: Example

Vertex DataFrame

```
+---+-------+---+
| id|   name|age|
+---+-------+---+
| u1|  Alice| 34|
| u2|    Bob| 36|
| u3|Charlie| 30|
| u4|  David| 29|
| u5| Esther| 32|
| u6|  Fanny| 36|
| u7|  Gabby| 60|
+---+-------+---+
```

Edge DataFrame

```
+---+---+------------+
|src|dst|relationship|
+---+---+------------+
| u1| u2|      friend|
| u2| u3|      follow|
| u3| u2|      follow|
| u6| u3|      follow|
| u5| u6|      follow|
| u5| u4|      friend|
| u4| u1|      friend|
| u1| u5|      friend|
+---+---+------------+
```

Building a graph: Example

```
from graphframes import GraphFrame

# Vertex DataFrame
v = spark.createDataFrame([ ("u1", "Alice", 34),\
                            ("u2", "Bob", 36),\
                            ("u3", "Charlie", 30),\
                            ("u4", "David", 29),\
                            ("u5", "Esther", 32),\
                            ("u6", "Fanny", 36),\
                            ("u7", "Gabby", 60)],\
                           ["id", "name", "age"])
```

Building a graph: Example

```
# Edge DataFrame
e = spark.createDataFrame([ ("u1", "u2", "friend"),\
                            ("u2", "u3", "follow"),\
                            ("u3", "u2", "follow"),\
                            ("u6", "u3", "follow"),\
                            ("u5", "u6", "follow"),\
                            ("u5", "u4", "friend"),\
                            ("u4", "u1", "friend"),\
                            ("u1", "u5", "friend")],\
                          ["src", "dst", "relationship"])

# Create the graph
g = GraphFrame(v, e)
```

Directed vs undirected edges

- In undirected graphs the edges indicate a two-way relationship (each edge can be traversed in both directions)
- In GraphX you could use to_undirected() to create an undirected copy of the Graph
- Unfortunately **GraphFrames does not support it**
  - You can convert your graph by applying a flatMap function over the edges of the directed graph that creates symmetric edges and then create a new GraphFrame

Cache graphs

- As with RDD and DataFrame, you can cache graphs in GraphFrame
  - Convenient if the same (complex) graph result of (multiple) transformations is used multiple times in the same application
  - Simply invoke **cache()** on the GraphFrame you want to cache
    - It persists the DataFrame-based representation of vertexes and edges of the graph

Querying the graph

- Some specific methods are provided to execute queries on graphs
  - filterVertices(condition)
  - filterEdges(condition)
  - dropIsolatedVertices()
- The returned result is the filtered version of the input graph

Querying the graph: filterVertices

- **filterVertices(condition)**
  - condition contains an SQL-like condition on the values of the attributes of the vertexes
    - E.g., “age>35”
  - Selects only the vertexes for which the specified condition is satisfied and returns a new graph with only the subset of selected vertexes

Querying the graph: filterEdges

- **filterEdges(condition)**
  - condition contains an SQL-like condition on the values of the attributes of the edges
    - E.g., "relationship='friend' "
  - Selects only the edges for which the specified condition is satisfied and returns a new graph with only the subset of selected edges

Querying the graph: dropIsolatedVertices

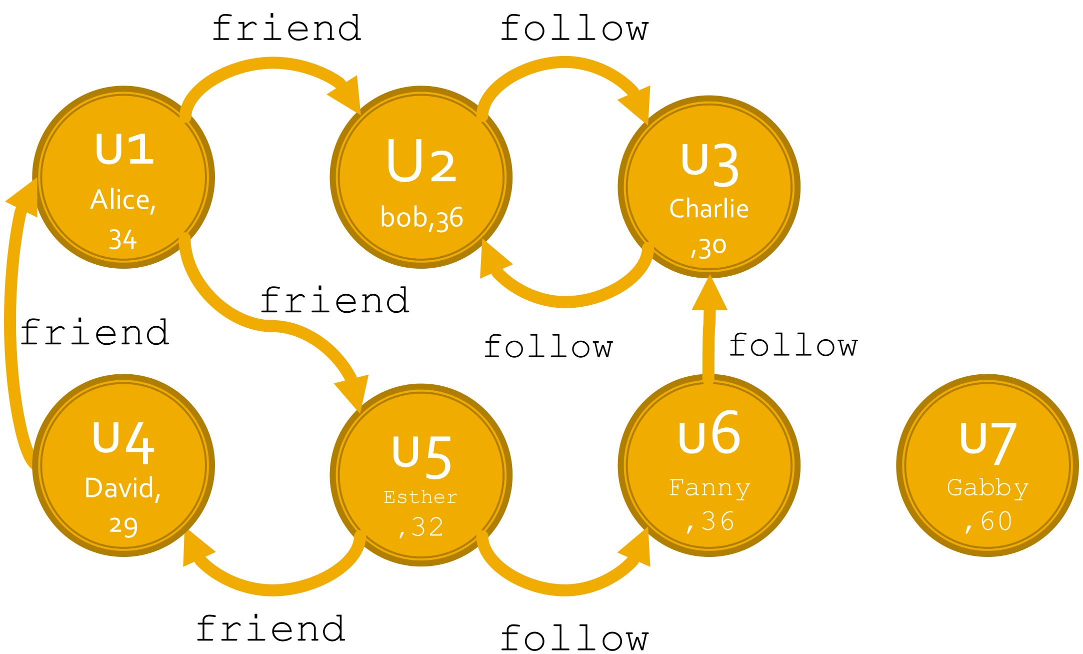
- **dropIsolatedVertices()**
  - Drops the vertexes that are not connected with any other node and returns a new graph without the dropped nodes

Querying the graph: Example 1

- Given the input graph, create a new subgraph including
  - Only the vertexes associated with users characterized by age between 29 and 50
  - Only the edges representing the friend relationship
  - Drop isolated vertexes

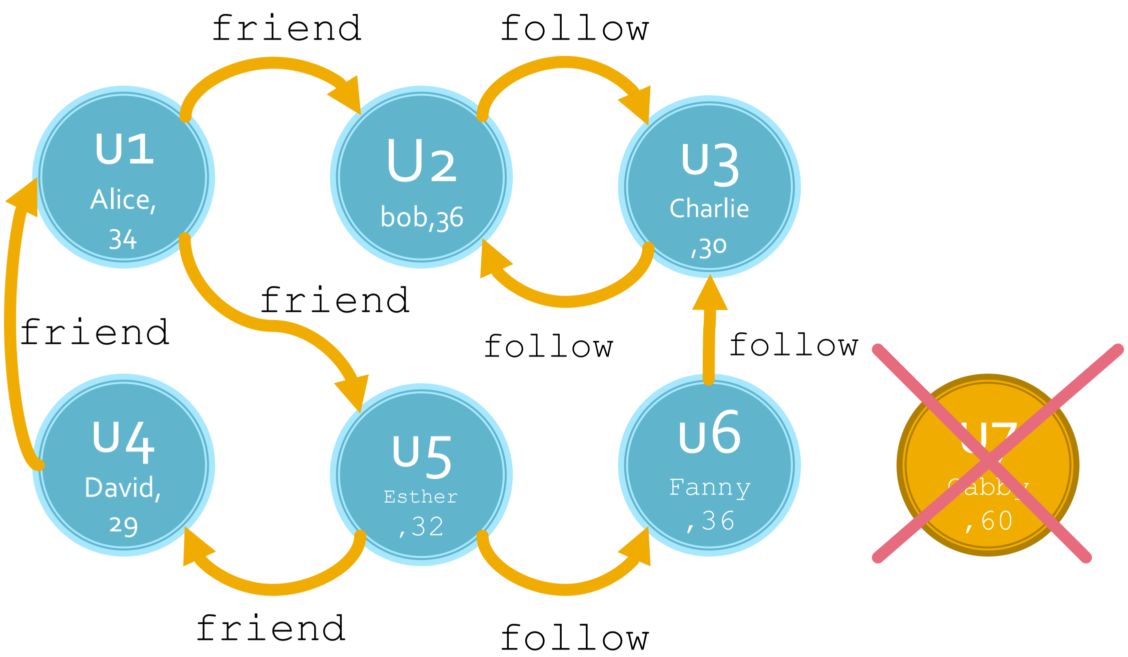
Querying the graph: Example 1

Input graph



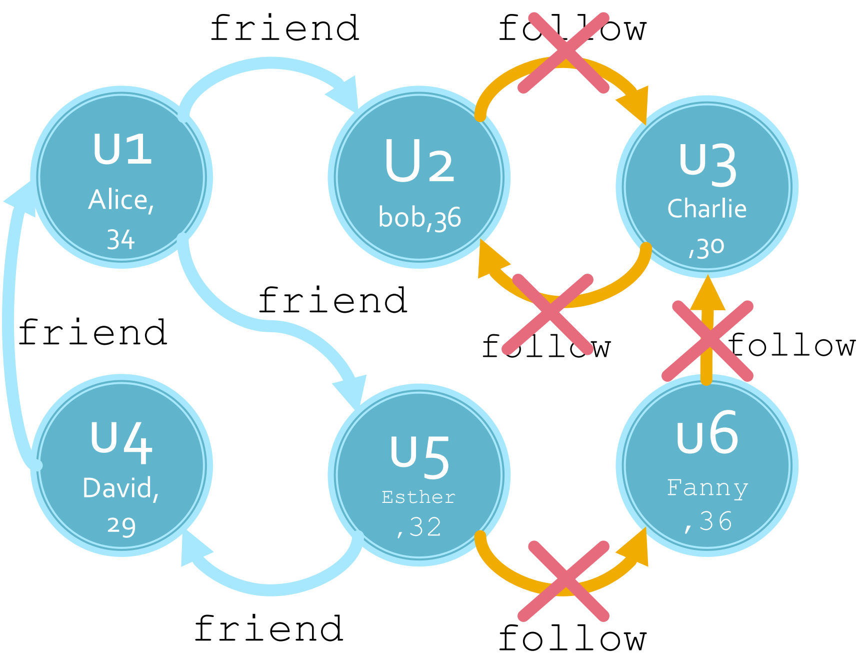
Querying the graph: Example 1

Filter vertexes



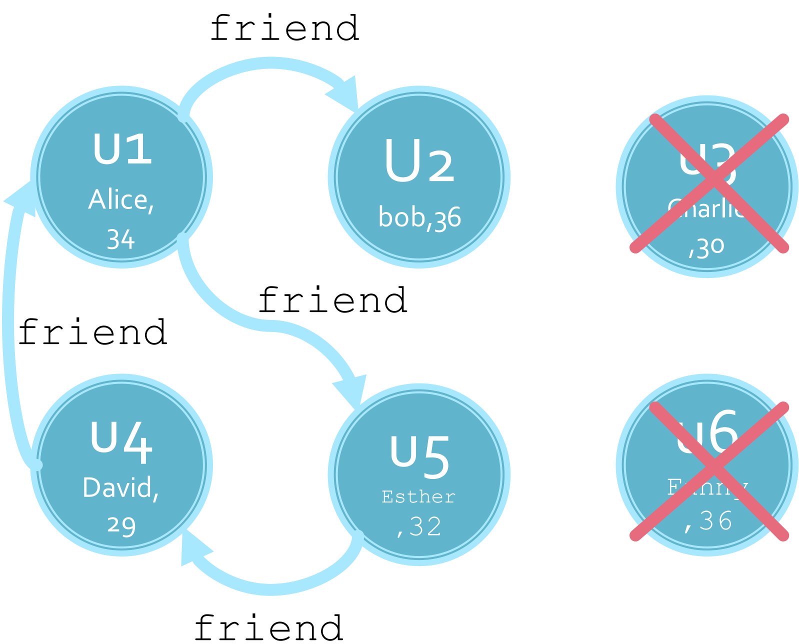
Querying the graph: Example 1

Filter edges



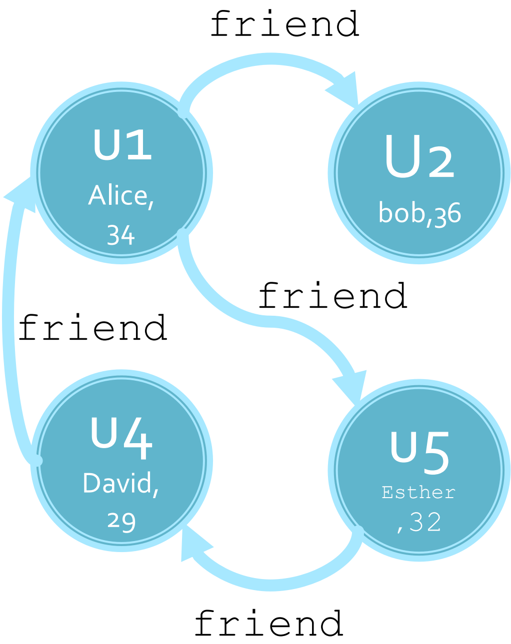
Querying the graph: Example 1

Drop isolated vertexes

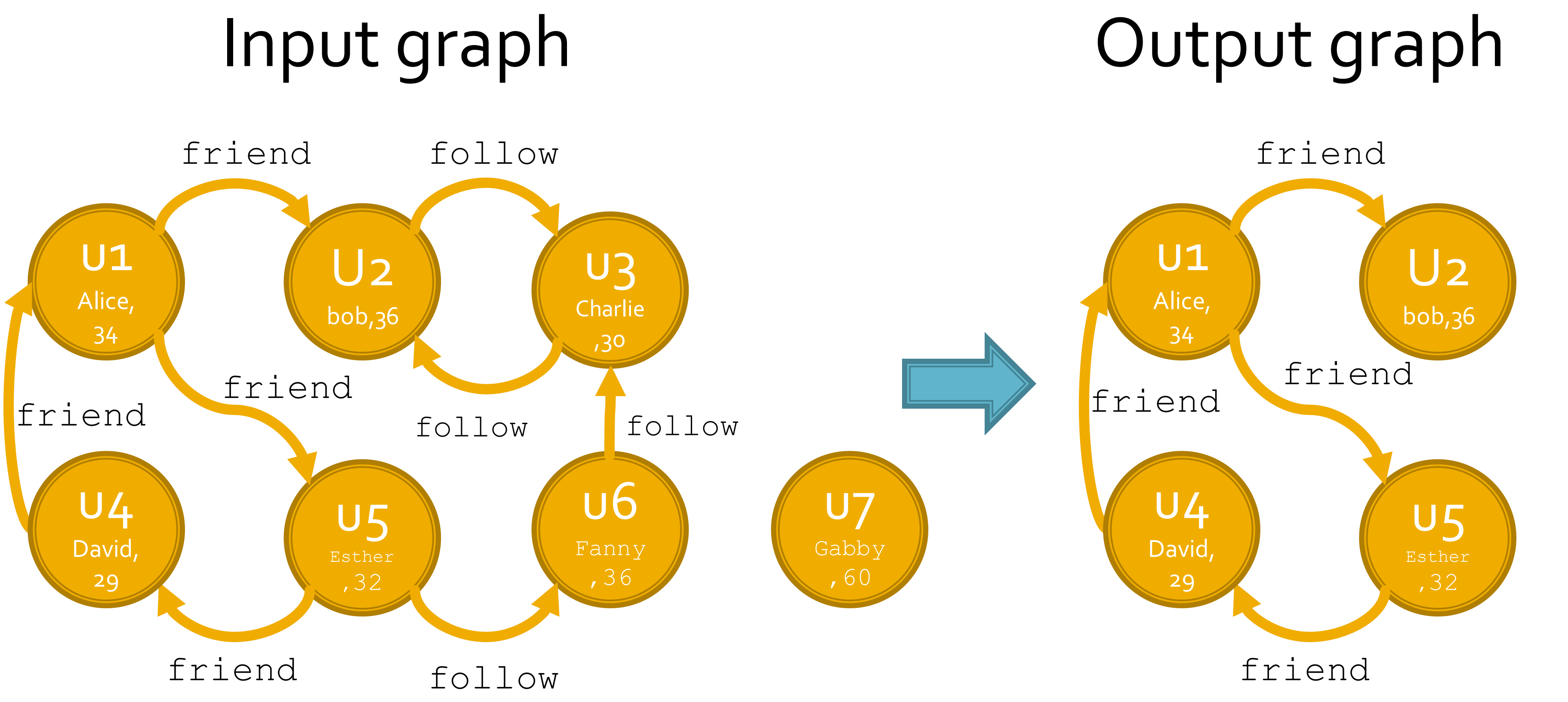


Querying the graph: Example 1

Output graph



Querying the graph: Example 1



Querying the graph: Example 1

```
from graphframes import GraphFrame

# Vertex DataFrame
v = spark.createDataFrame([ ("u1", "Alice", 34),\
                            ("u2", "Bob", 36),\
                            ("u3", "Charlie", 30),\
                            ("u4", "David", 29),\
                            ("u5", "Esther", 32),\
                            ("u6", "Fanny", 36),\
                            ("u7", "Gabby", 60)],\
                           ["id", "name", "age"])
```

Querying the graph: Example 1

```
# Edge DataFrame
e = spark.createDataFrame([ ("u1", "u2", "friend"),\
                            ("u2", "u3", "follow"),\
                            ("u3", "u2", "follow"),\
                            ("u6", "u3", "follow"),\
                            ("u5", "u6", "follow"),\
                            ("u5", "u4", "friend"),\
                            ("u4", "u1", "friend"),\
                            ("u1", "u5", "friend")],\
                           ["src", "dst", "relationship"])

# Create the graph
g = GraphFrame(v, e)
```

Querying the graph: Example 1

```
selectedUsersandFriendRelGraph = g\
.filterVertices("age>=29 AND age<=50")\
.filterEdges("relationship='friend'")
. dropIsolatedVertices()
```

Querying the graph

- Given a GraphFrame, we can easily access its vertexes and edges
  - g.vertices returns the DataFrame associated with the vertexes of the input graph
  - g.edges returns the DataFrame associated with the edges of the input graph

Querying the graph

- All the standard DataFrame transformations/ actions are available also for the DataFrames that are used to store vertexes and edges
  - For example, the number of vertexes and the number of edges can be computed by invoking the count() action on the DataFrames vertices and edges, respectively

Querying the graph: Example 2

- Given the input graph
  - Count how many vertexes and edges has the graph
  - Find the smallest value of age (i.e., the age of the youngest user in the graph)
  - Count the number of edges of type "follow" in the graph

Querying the graph: Example 2

```
from graphframes import GraphFrame

# Vertex DataFrame
v = spark.createDataFrame([ ("u1", "Alice", 34),\
                            ("u2", "Bob", 36),\
                            ("u3", "Charlie", 30),\
                            ("u4", "David", 29),\
                            ("u5", "Esther", 32),\
                            ("u6", "Fanny", 36),\
                            ("u7", "Gabby", 60)],\
                           ["id", "name", "age"])
```

Querying the graph: Example 2

```
# Edge DataFrame
e = spark.createDataFrame([ ("u1", "u2", "friend"),\
                            ("u2", "u3", "follow"),\
                            ("u3", "u2", "follow"),\
                            ("u6", "u3", "follow"),\
                            ("u5", "u6", "follow"),\
                            ("u5", "u4", "friend"),\
                            ("u4", "u1", "friend"),\
                            ("u1", "u5", "friend")],\
                          ["src", "dst", "relationship"])

# Create the graph
g = GraphFrame(v, e)
```

Querying the graph: Example 2

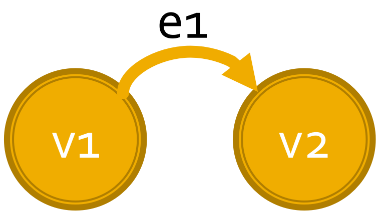
```
# Count how many vertexes and edges has the graph
print("Number of vertexes: ",g.vertices.count())
print("Number of edges: ",g.edges.count())

# Print on the standard output the smallest value of age
# (i.e., the age of the youngest user in the graph)
g.vertices.agg({"age":"min"}).show()

# Print on the standard output
# the number of "follow" edges in the graph.
numFollows = g.edges.filter("relationship = 'follow' ").count()

print(numFollows)
```

Motif finding

- **Motif** finding refers to searching for **structural patterns** in graphs
- A simple Domain-Specific Language (DSL) is used to specify the structure of the patterns we are interested in
  - The paths/subgraphs in the graph matching the specified structural pattern are selected

DSL for Motif finding

- The **basic unit** of a pattern is a connection between vertexes



  - **(v1) – [e1] -> (v2)**

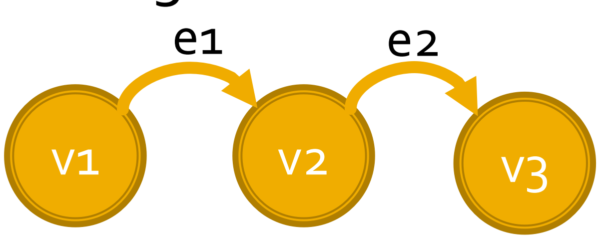
    means

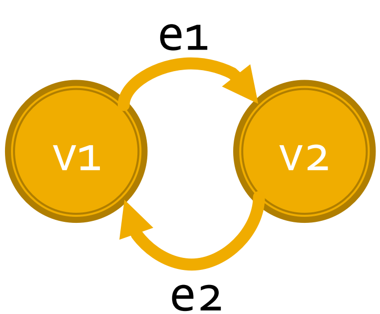
  - An arbitrary edge [e1] from an arbitrary vertex (v1) to another arbitrary vertex (v2)
- **Edges** are denoted by **square brackets**
  - [e1]
- **Vertexes** are expressed by **round brackets**
  - (v1), (v2)

DSL for Motif finding

- Patterns are chains of basic units
  - (v1) – [e1] -> (v2); (v2) – [e2] -> (v3)

    means
  - An arbitrary edge from an arbitrary vertex v1 to another arbitrary vertex v2 and another arbitrary edge from v2 to another arbitrary vertex v3
    - v3 and v1 can be the same vertex

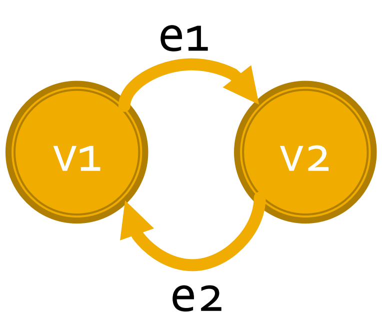




DSL for Motif finding

- The same vertex name is used in a pattern to have a reference to the same vertex
  - (v1) – [e1] -> (v2); (v2) – [e2] -> (v1)

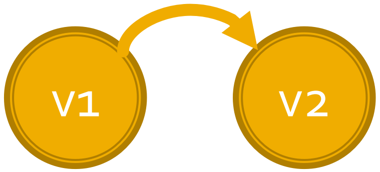
    means
  - An arbitrary edge from an arbitrary vertex v1 to another arbitrary vertex v2 and vice-versa



DSL for Motif finding

- It is acceptable to omit names for vertices or edges in patterns when not needed
  - (v1)-[]->(v2)

    expresses an arbitrary edge between two arbitrary vertexes v1,v2 but does not assign a name to the edge

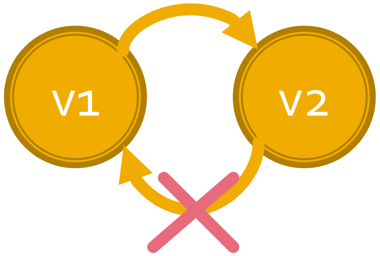


- These are called **anonymous** vertexes and edges

DSL for Motif finding

- A basic unit (an edge between two vertexes) can be negated to indicate that the edge should not be present in the graph
  - (v1)-[]->(v2); !(v2)-[]->(v1)

    means
  - Edges from v1 to v2 but no edges from v2 to v1



DSL for Motif finding

- The **find(motif)** method of GraphFrame is used to select motifs
  - motif
    - DSL representation of the structural pattern

DSL for Motif finding

- find() returns a DataFrame of all the paths matching the structural motif/pattern
  - One path per record
  - The returned DataFrame will have a column for each of the named elements (vertexes and edges) in the structural pattern/motif
    - Each column is a struct
      - The fields of each struct are the labels/features of the associated vertex or edge
  - It can return duplicate rows/records
    - If there are many paths connecting the same nodes

DSL for Motif finding

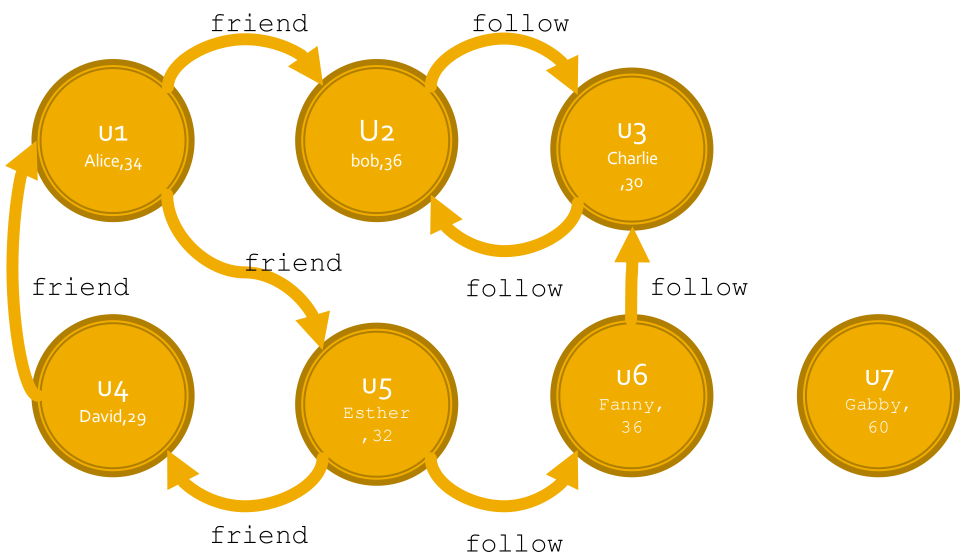
- More complex queries on the structure and content of the patterns can be expressed by applying filters to the result DataFrame
  - i.e., more complex queries can be applied by combing find() and filter()

Motif finding: Example 1

- Find the paths/subgraphs matching the pattern

  (v1) – [e1] -> (v2); (v2) – [e2] -> (v1)

- Store the result in a DataFrame

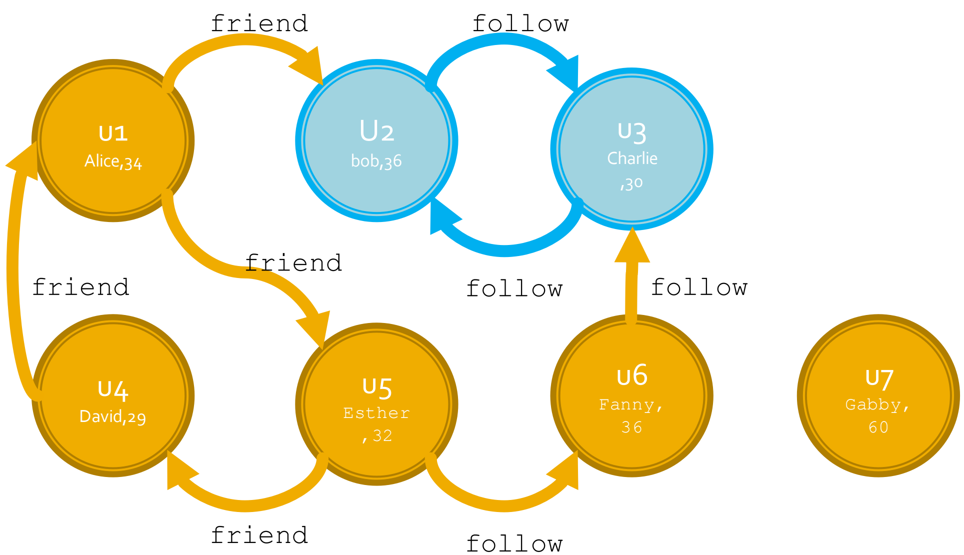


Motif finding: Example 1

- Find the paths/subgraphs matching the pattern

  (v1) – [e1] -> (v2); (v2) – [e2] -> (v1)

- Store the result in a DataFrame

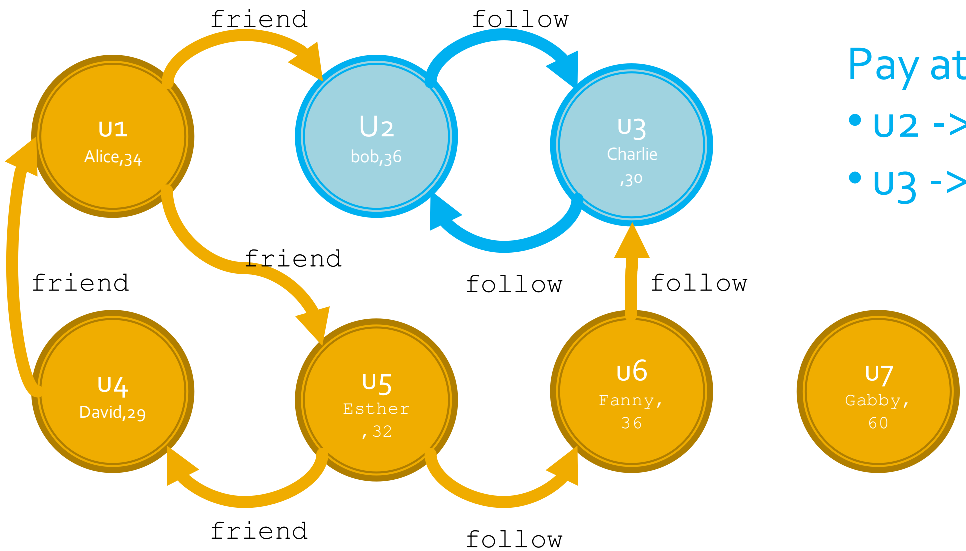


Motif finding: Example 1

- Find the paths/subgraphs matching the pattern

(v1) – [e1] -> (v2); (v2) – [e2] -> (v1)

- Store the result in a DataFrame



Pay attention that two paths are returned:
- u2 -> follow -> u3 -> follow ->u2
- u3 -> follow -> u2 -> follow ->u3

Motif finding: Example 1

- Find the paths/subgraphs matching the pattern

  (v1) – [e1] -> (v2); (v2) – [e2] -> (v1)

- Content of the returned DataFrame

v1	e1	v2	e2
[u2, Bob, 36]	[u2, u3, follow]	[u3, Charlie, 30]	[u3, u2, follow]
[u3, Charlie, 30]	[u3, u2, follow]	[u2, Bob, 36]	[u2, u3, follow]

Motif finding: Example 1

- Find the paths/subgraphs matching the pattern

  (v1) – [e1] -> (v2); (v2) – [e2] -> (v1)

- Content of the returned DataFrame

v1	e1	v2	e2
[u2, Bob, 36]	[u2, u3, follow]	[u3, Charlie, 30]	[u3, u2, follow]
[u3, Charlie, 30]	[u3, u2, follow]	[u2, Bob, 36]	[u2, u3, follow]

There is one column for each (distinct) named vertex and edge of the structural pattern

Motif finding: Example 1

- Find the paths/subgraphs matching the pattern

(v1) – [e1] -> (v2); (v2) – [e2] -> (v1)

- Content of the returned DataFrame

```
+------------------+------------------+------------------+--------------------+
|        v1        |        e1        |        v2        |         e2         |
+------------------+------------------+------------------+--------------------+
| [u2, Bob, 36]    | [u2, u3, follow] | [u3, Charlie, 30]| [u3, u2, follow]   |
| [u3, Charlie, 30]| [u3, u2, follow] | [u2, Bob, 36]    | [u2, u3, follow]   |
+------------------+------------------+------------------+--------------------+
```

The records are associated with the vertexes and edges of the selected paths

Motif finding: Example 1

- Find the paths/subgraphs matching the pattern

(v1) – [e1] -> (v2); (v2) – [e2] -> (v1)

- Content of the returned DataFrame

```
+-------------------+-------------------+-------------------+---------------------+
|         v1        |         e1        |         v2        |         e2          |
+-------------------+-------------------+-------------------+---------------------+
| [u2, Bob, 36]     | [u2, u3, follow]  | [u3, Charlie, 30] | [u3, u2, follow]    |
| [u3, Charlie, 30] | [u3, u2, follow]  | [u2, Bob, 36]     | [u2, u3, follow]    |
+-------------------+-------------------+-------------------+---------------------+
```

All columns are associated with the data type “struct”.
Each struct has the same “schema/features” of the associated vertex or edge.

Motif finding: Example 1

```
from graphframes import GraphFrame

# Vertex DataFrame
v = spark.createDataFrame([ ("u1", "Alice", 34),\
                            ("u2", "Bob", 36),\
                            ("u3", "Charlie", 30),\
                            ("u4", "David", 29),\
                            ("u5", "Esther", 32),\
                            ("u6", "Fanny", 36),\
                            ("u7", "Gabby", 60)],\
                          ["id", "name", "age"])
```

Motif finding: Example 1

```
# Edge DataFrame
e = spark.createDataFrame([ ("u1", "u2", "friend"),\
                            ("u2", "u3", "follow"),\
                            ("u3", "u2", "follow"),\
                            ("u6", "u3", "follow"),\
                            ("u5", "u6", "follow"),\
                            ("u5", "u4", "friend"),\
                            ("u4", "u1", "friend"),\
                            ("u1", "u5", "friend")],\
                           ["src", "dst", "relationship"])

# Create the graph
g = GraphFrame(v, e)
```

Motif finding: Example 1

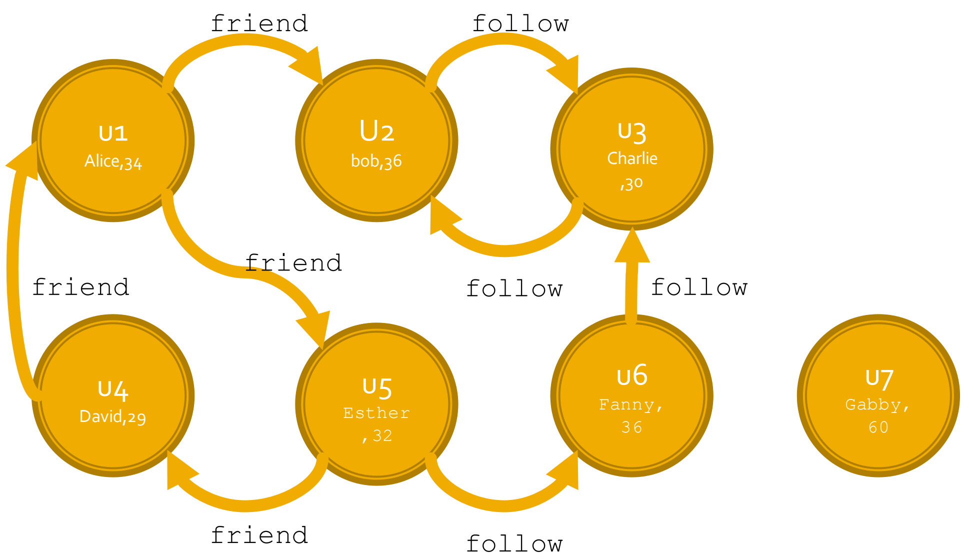
```
# Retrieve the motifs associated with the pattern
# vertex ->  edge -> vertex ->  edge ->vertex
motifs = g.find("(v1)-[e1]->(v2); (v2)-[e2]->(v1)")
```

Motif finding: Example 2

- Find the paths/subgraphs matching the pattern

  (v1)- [friend] -> (v2); (v2)- [follow] -> (v3)

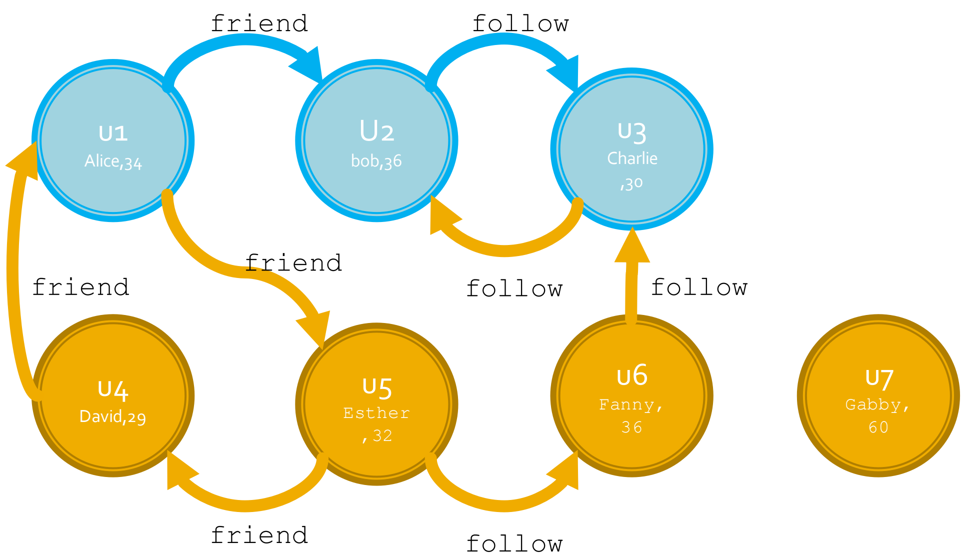
- Store the result in a DataFrame



Motif finding: Example 2

- Find the paths/subgraphs matching the pattern

  (v1)- [friend] -> (v2); (v2)- [follow] -> (v3)
- Store the result in a DataFrame



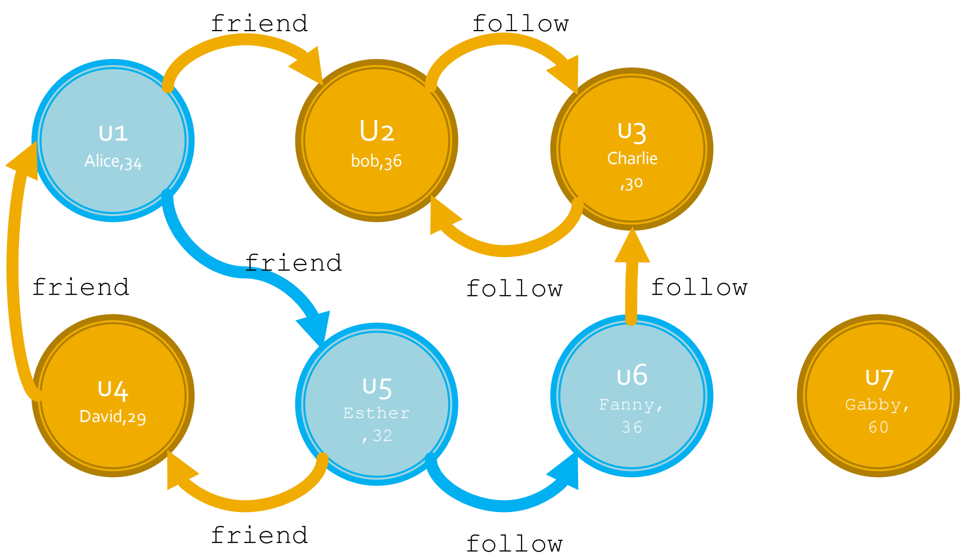
First selected path

Motif finding: Example 2

- Find the paths/subgraphs matching the pattern

  (v1)- [friend] -> (v2); (v2)- [follow] -> (v3)

- Store the result in a DataFrame



Second selected path

Motif finding: Example 2

```
from graphframes import GraphFrame

# Vertex DataFrame
v = spark.createDataFrame([ ("u1", "Alice", 34),\
                            ("u2", "Bob", 36),\
                            ("u3", "Charlie", 30),\
                            ("u4", "David", 29),\
                            ("u5", "Esther", 32),\
                            ("u6", "Fanny", 36),\
                            ("u7", "Gabby", 60)],\
                          ["id", "name", "age"])
```

Motif finding: Example 2

```
# Edge DataFrame
e = spark.createDataFrame([ ("u1", "u2", "friend"),\
                            ("u2", "u3", "follow"),\
                            ("u3", "u2", "follow"),\
                            ("u6", "u3", "follow"),\
                            ("u5", "u6", "follow"),\
                            ("u5", "u4", "friend"),\
                            ("u4", "u1", "friend"),\
                            ("u1", "u5", "friend")],\
                          ["src", "dst", "relationship"])

# Create the graph
g = GraphFrame(v, e)
```

Motif finding: Example 2

```
# Retrieve the motifs associated with the pattern
# vertex ->  edge -> vertex ->  edge ->vertex
motifs = g.find("(v1)-[friend]->(v2); (v2)-[follow]->(v3)")

# Filter the motifs (the content of the motifs DataFrame)
# Select only the ones matching the pattern
# vertex ->  friend-> vertex ->  follow ->vertex
motifsFriendFollow = motifs\
.filter("friend.relationship='friend' AND follow.relationship='follow' ")
```

Motif finding: Example 2

```
# Retrieve the motifs associated with the pattern
# vertex -> edge -> vertex -> edge ->vertex
motifs = g.find("(v1)-[friend]->(v2); (v2)-[follow]->(v3)")

# Filter the motifs (the content of the motifs DataFrame)
# Select only the ones matching the pattern
# vertex -> friend-> vertex -> follow ->vertex
motifsFriendFollow = motifs\
.filter("friend.relationship='friend' AND follow.relationship='follow' ")
```

Columns friend and follow are structs with three fields/attributes
- src
- dst
- relationship

Motif finding: Example 2

```
# Retrieve the motifs associated with the pattern
# vertex -> edge -> vertex -> edge ->vertex
motifs = g.find("(v1)-[friend]->(v2); (v2)-[follow]->(v3)")

# Filter the motifs (the content of the motifs DataFrame)
# Select only the ones matching the pattern
# vertex -> friend-> vertex -> follow ->vertex
motifsFriendFollow = motifs\
.filter("friend.relationship='friend' AND follow.relationship='follow' ")
```

To access a field of a struct column use the syntax columnName.field

Basic statistics

- Some specific properties are provided to compute basic statistics on the degrees of the vertexes
  - degrees
  - inDegrees
  - outDegrees
- The returned result of each of this property is a DataFrame with
  - id
  - (in/out)Degree value

Basic statistics: degrees

- **degrees**
  - Returns the degree of each vertex
    - i.e., the number of edges associated with each vertex
  - The **result** is stored in a DataFrame with **Columns** (vertex) “**id**” and “**degree**”
    - One record per vertex
    - Only the vertexes with degree>=1 are stored in the returned DataFrame

Basic statistics: inDegrees

- **inDegrees**
  - Returns the in-degree of each vertex
    - i.e., the number of in-edges associated with each vertex
  - The **result** is stored in a DataFrame with **Columns** (vertex) “**id**” and “**inDegree**”
    - One record per vertex
    - Only the vertexes with in-degree>=1 are stored in the returned DataFrame

Basic statistics: outDegrees

- **outDegrees**
  - Returns the out-degree of each vertex
    - i.e., the number of out-edges associated with each vertex
  - The **result** is stored in a DataFrame with **Columns** (vertex) “**id**” and “**outDegree**”
    - One record per vertex
    - Only the vertexes with out-degree>=1 are stored in the returned DataFrame

Basic statistics: Example 1

- Given the input graph, compute
  - Degree of each vertex
  - inDegree of each vertex
  - outDegree of each vertex

Basic statistics: Example 1

```
from graphframes import GraphFrame

# Vertex DataFrame
v = spark.createDataFrame([ ("u1", "Alice", 34),\
                           ("u2", "Bob", 36),\
                           ("u3", "Charlie", 30),\
                           ("u4", "David", 29),\
                           ("u5", "Esther", 32),\
                           ("u6", "Fanny", 36),\
                           ("u7", "Gabby", 60)],\
                          ["id", "name", "age"])
```

Basic statistics: Example 1

```
# Edge DataFrame
e = spark.createDataFrame([ ("u1", "u2", "friend"),\
                            ("u2", "u3", "follow"),\
                            ("u3", "u2", "follow"),\
                            ("u6", "u3", "follow"),\
                            ("u5", "u6", "follow"),\
                            ("u5", "u4", "friend"),\
                            ("u4", "u1", "friend"),\
                            ("u1", "u5", "friend")],\
                           ["src", "dst", "relationship"])

# Create the graph
g = GraphFrame(v, e)
```

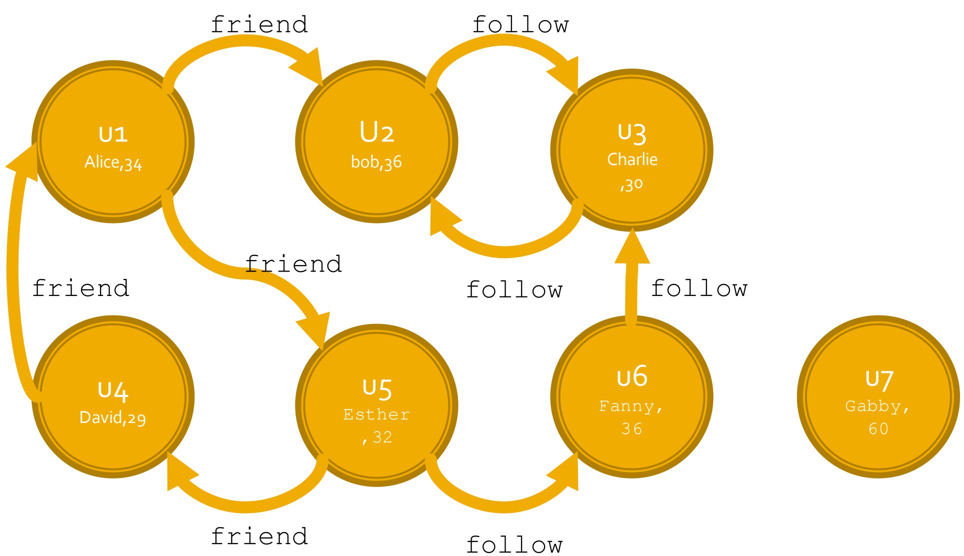
Basic statistics: Example 1

```
# Retrieve the DataFrame with the information about the degree of
# each vertex
vertexesDegreesDF = g.degrees

# Retrieve the DataFrame with the information about the in-degree of
# each vertex
vertexesInDegreesDF = g.inDegrees

# Retrieve the DataFrame with the information about the out-degree of
# each vertex
vertexesOutDegreesDF = g.outDegrees
```

Basic statistics: Example 2

- Given the input graph, select only the ids of the vertexes with at least 2 in-edges

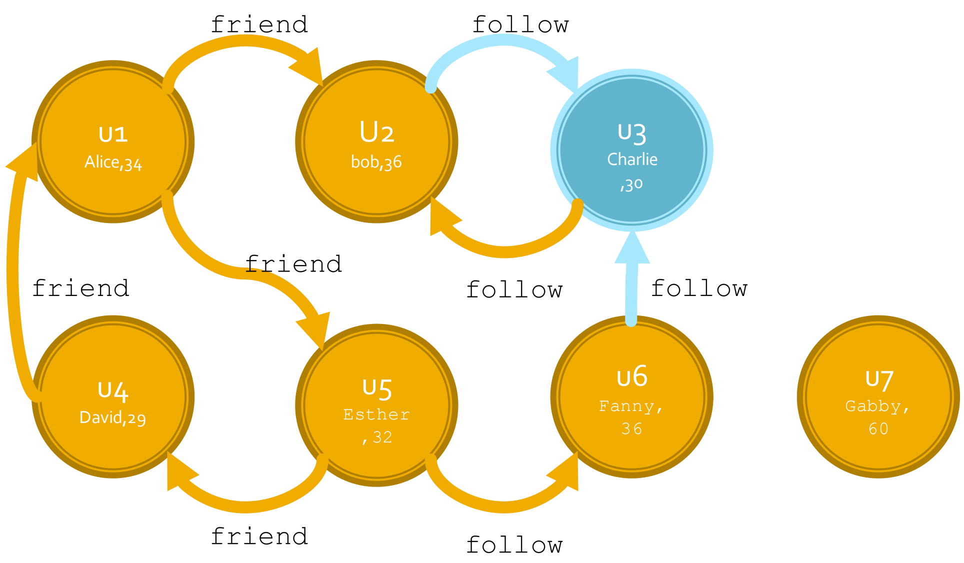
Input graph



Basic statistics: Example 2

- Given the input graph, select only the ids of the vertexes with at least 2 in-edges

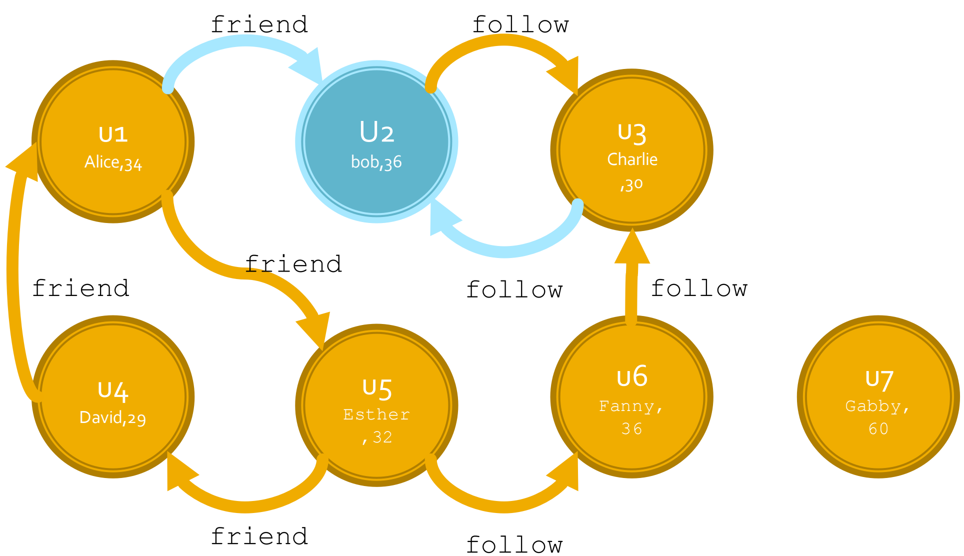
Input graph



Basic statistics: Example 2

- Given the input graph, select only the ids of the vertexes with at least 2 in-edges

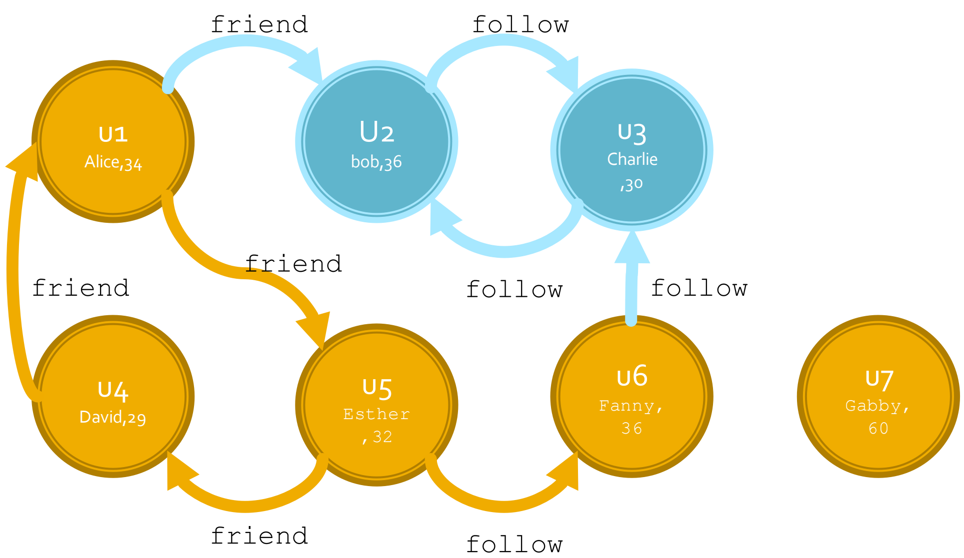
Input graph



Basic statistics: Example 2

- Given the input graph, select only the ids of the vertexes with at least 2 in-edges

Input graph



Selected IDs

U2

U3

Basic statistics: Example 2

```
from graphframes import GraphFrame

# Vertex DataFrame
v = spark.createDataFrame([ ("u1", "Alice", 34),\
                            ("u2", "Bob", 36),\
                            ("u3", "Charlie", 30),\
                            ("u4", "David", 29),\
                            ("u5", "Esther", 32),\
                            ("u6", "Fanny", 36),\
                            ("u7", "Gabby", 60)],\
                          ["id", "name", "age"])
```

Basic statistics: Example 2

```
# Edge DataFrame
e = spark.createDataFrame([ ("u1", "u2", "friend"),\
                            ("u2", "u3", "follow"),\
                            ("u3", "u2", "follow"),\
                            ("u6", "u3", "follow"),\
                            ("u5", "u6", "follow"),\
                            ("u5", "u4", "friend"),\
                            ("u4", "u1", "friend"),\
                            ("u1", "u5", "friend")],\
                          ["src", "dst", "relationship"])

# Create the graph
g = GraphFrame(v, e)
```

Basic statistics: Example 2

```
# Retrieve the DataFrame with the information about the in-degree of
# each vertex
vertexesInDegreesDF = g.inDegrees

# Select only the vertexes with and in-degree value >=2
selectedVertexesDF = vertexesInDegreesDF.filter("inDegree>=2")

# Select only the content of Column id
selectedVertexesIDsDF = selectedVertexesDF.select("id")
```