Raphtory: A practical system for the analysis of temporal graphs

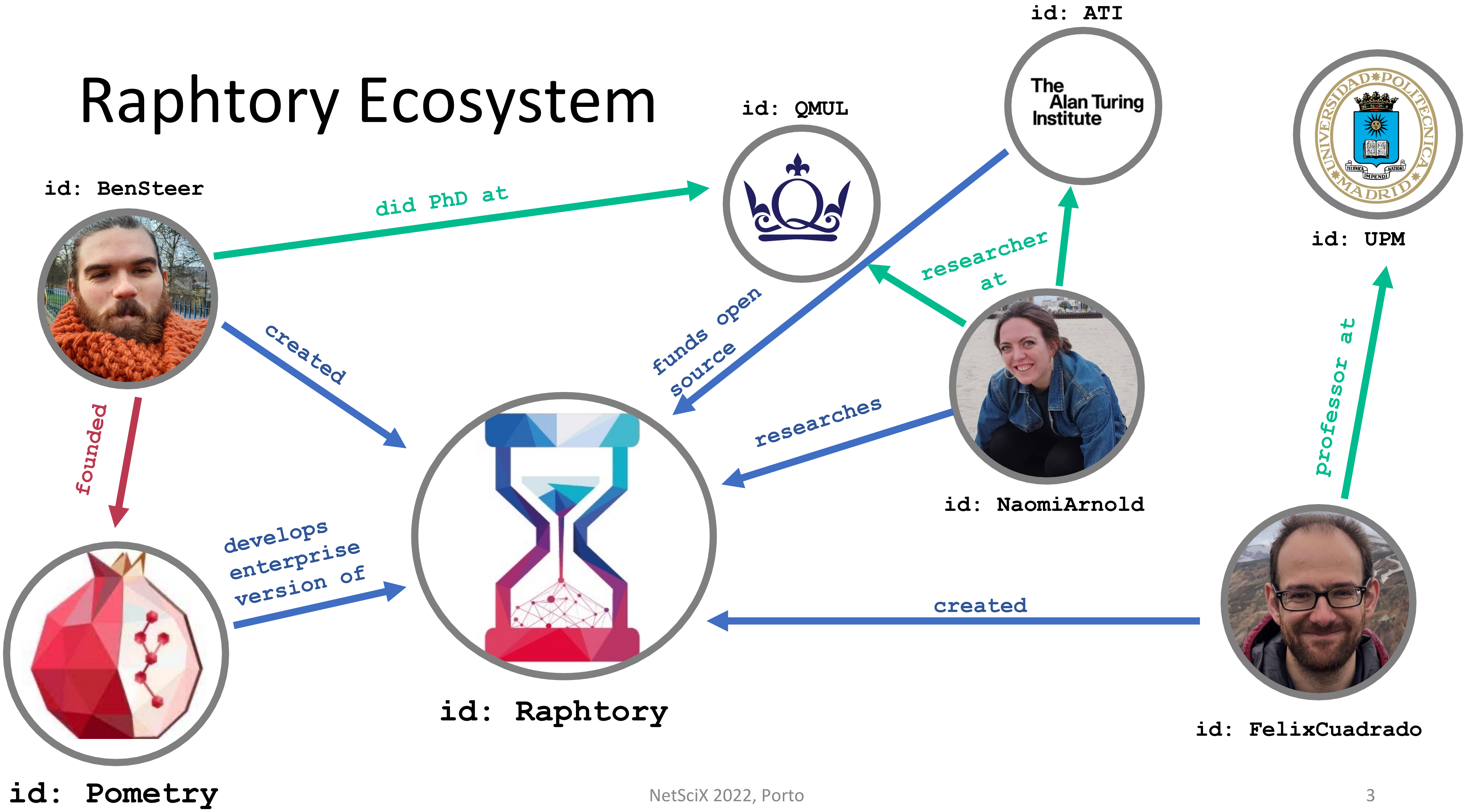
Ben Steer (Pometry), Naomi Arnold (QMUL, ATI), Felix Cuadrado (UPM, QMUL, ATI)

21/01/2022

Raphtory in a nutshell

- Platform for **temporal network analysis**
- **Ingest** graph data from anywhere
- Run a variety of **in-built algorithms,** or write your own
- Deploy on your **laptop**, across a **cluster** or in the **cloud**

Raphtory Ecosystem



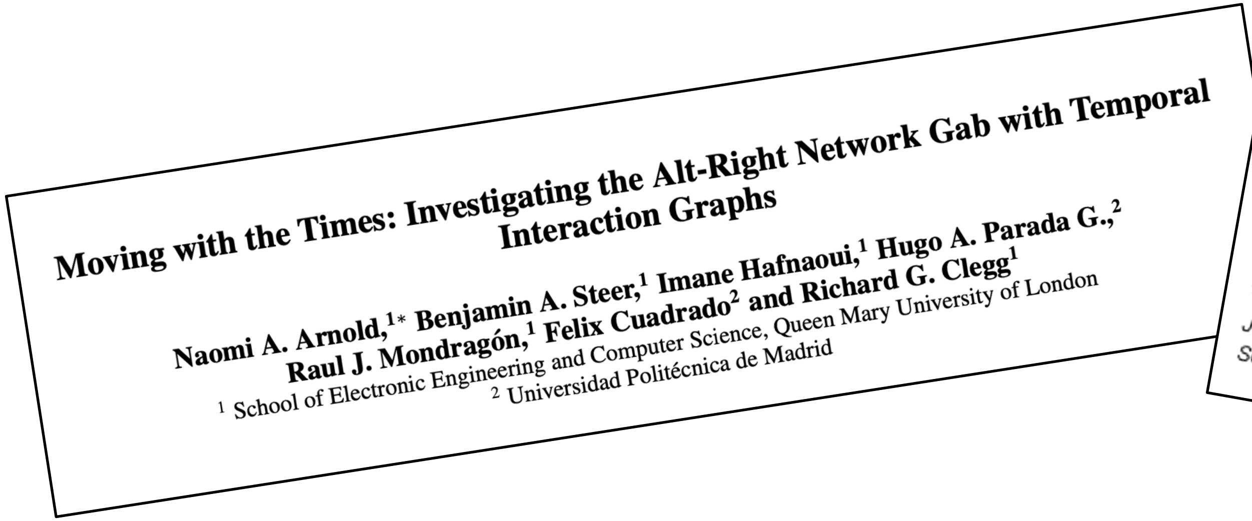
In this session we will

- Show you how **Raphtory works** and why it might be helpful for your research use-case.
- Show you round the **Raphtory API** with how to **write algorithms** and **make queries** on your data.
- Give you a **demo** with an example from Lord of the Rings!

What has Raphtory been used for?

What has Raphtory been used for?

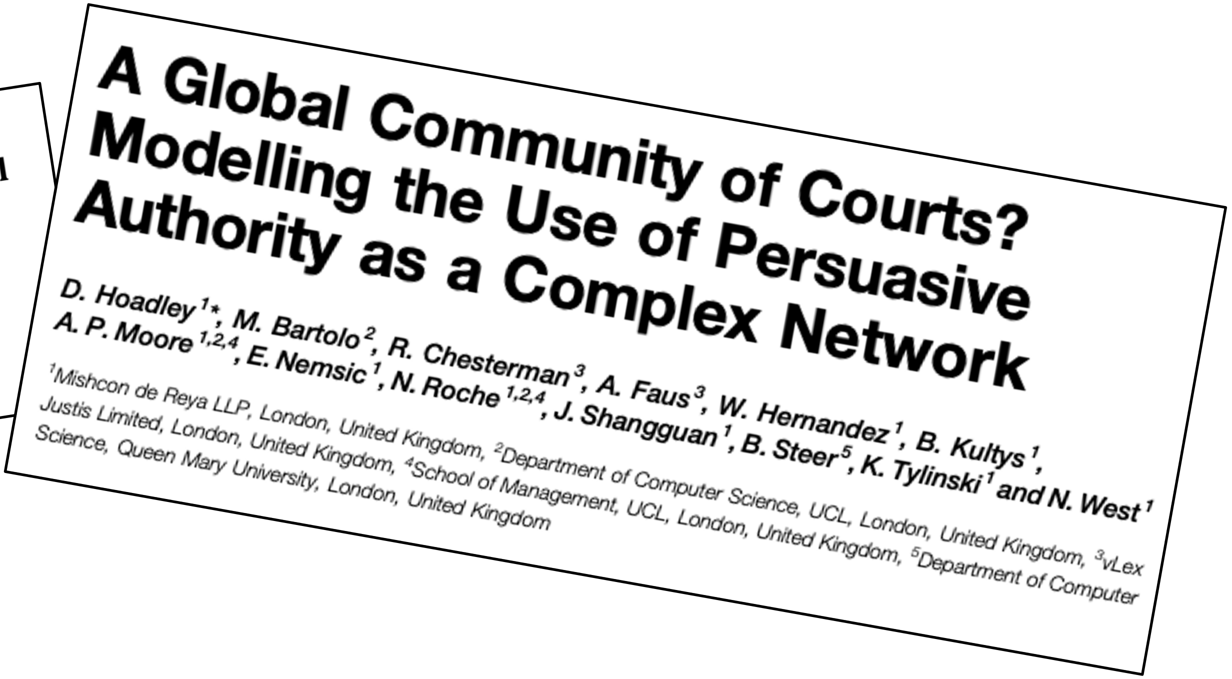
Across Industry and Academia



Moving with the Times: Investigating the Alt-Right Network Gab with Temporal Interaction Graphs

Naomi A. Arnold,[1*] Benjamin A. Steer,[1] Imane Hafnaoui,[1] Hugo A. Parada G.,[2] Raul J. Mondragón,[1] Felix Cuadrado[2] and Richard G. Clegg[1]

[1] School of Electronic Engineering and Computer Science, Queen Mary University of London
[2] Universidad Politécnica de Madrid

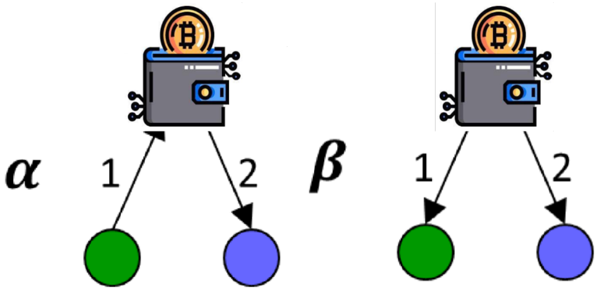


A Global Community of Courts? Modelling the Use of Persuasive Authority as a Complex Network

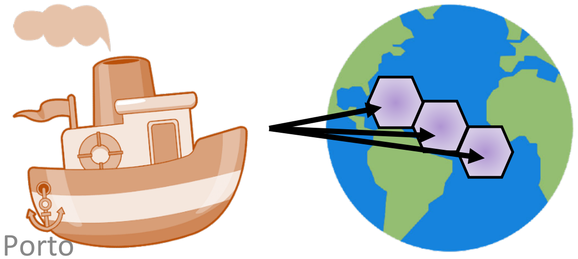
D. Hoadley [1], M. Bartolo [2], R. Chesterman [3], A. Faus [3], W. Hernandez [1], B. Kultys [1], A. P. Moore [1,2,4], E. Nemsic [1], N. Roche [1,2,4], J. Shangguan [1], B. Steer [5], K. Tylinski [1] and N. West [1]*

[1]Mishcon de Reya LLP, London, United Kingdom, [2]Department of Computer Science, UCL, London, United Kingdom, [3]vLex Justis Limited, London, United Kingdom, [4]School of Management, UCL, London, United Kingdom, [5]Department of Computer Science, Queen Mary University, London, United Kingdom

Temporal Motifs In
Crypto Networks



Spatial and Temporal
Co-location of Ships

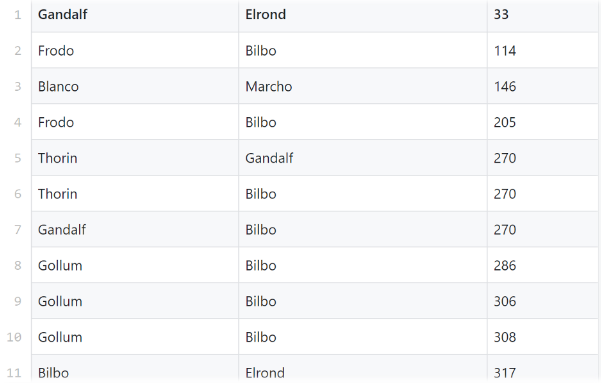


Raphtory Background

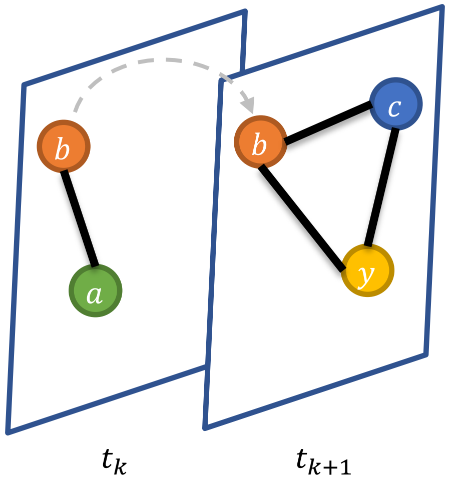
From Static To Temporal Graphs

Temporal graphs are graphs which are time-varying, changing in time

Character Interaction dataset

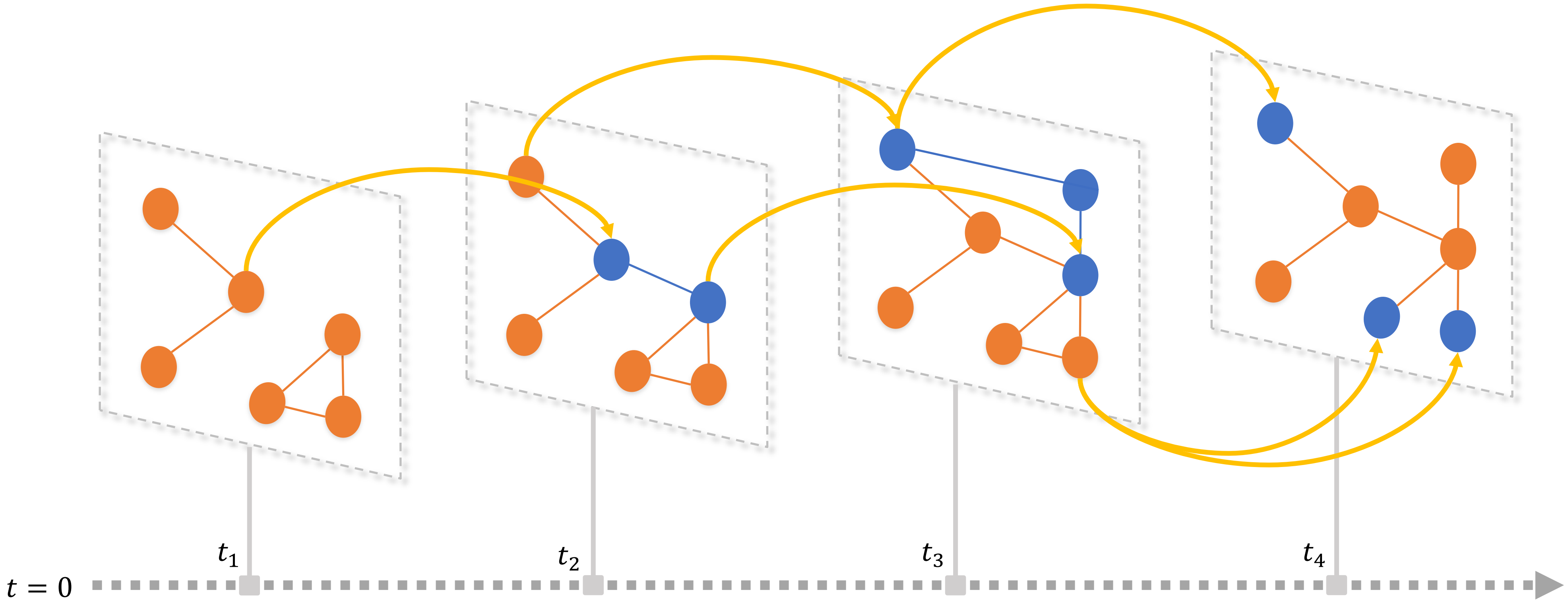


Gandalf	Elrond	33
Frodo	Bilbo	114
Blanco	Marcho	146
Frodo	Bilbo	205
Thorin	Gandalf	270
Thorin	Bilbo	270
Gandalf	Bilbo	270
Gollum	Bilbo	286
Gollum	Bilbo	306
Gollum	Bilbo	308
Bilbo	Elrond	317



From Static To Temporal Graphs

Temporal graphs are graphs which are time-varying, changing in time



The challenges of wrangling temporal datasets

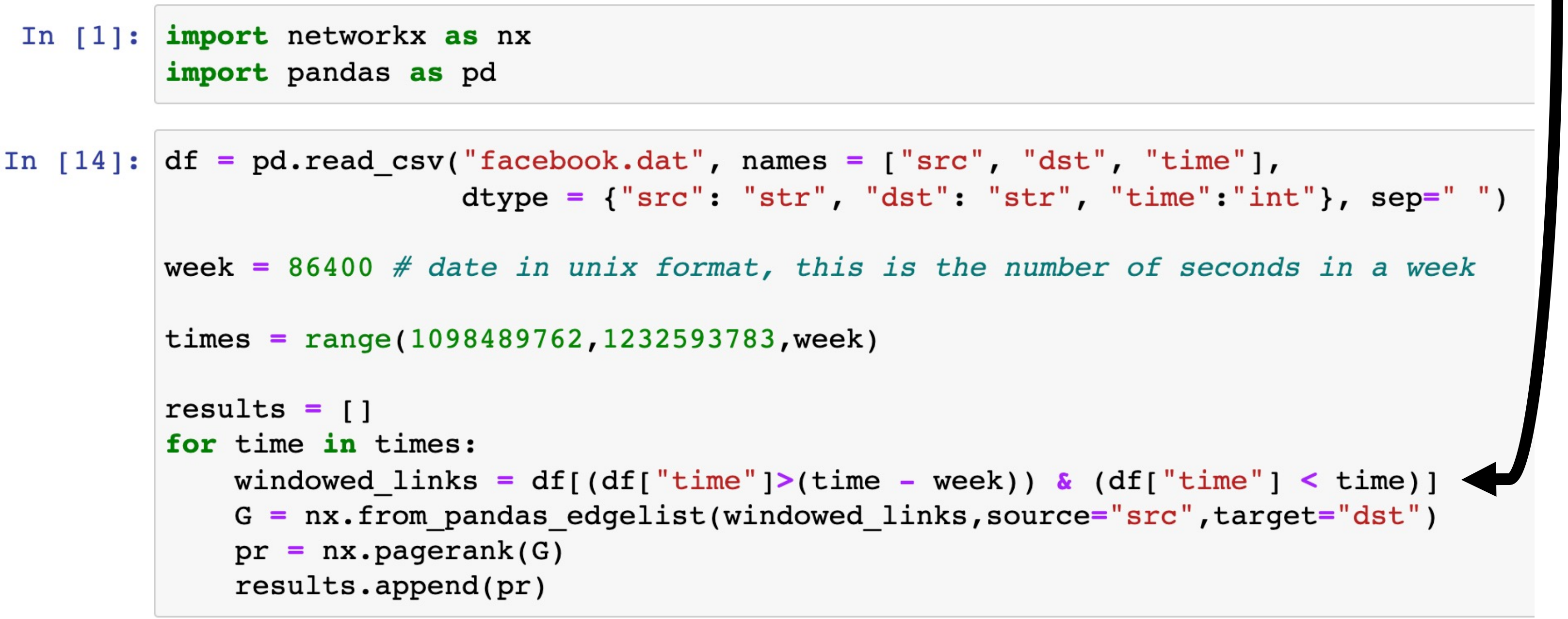
Read: why we made Raphtory ;)

The code can be hacky and gets messy quickly

E.g. “I want to get the weekly PageRank scores in this social network from the start of the dataset to the end of the dataset.”

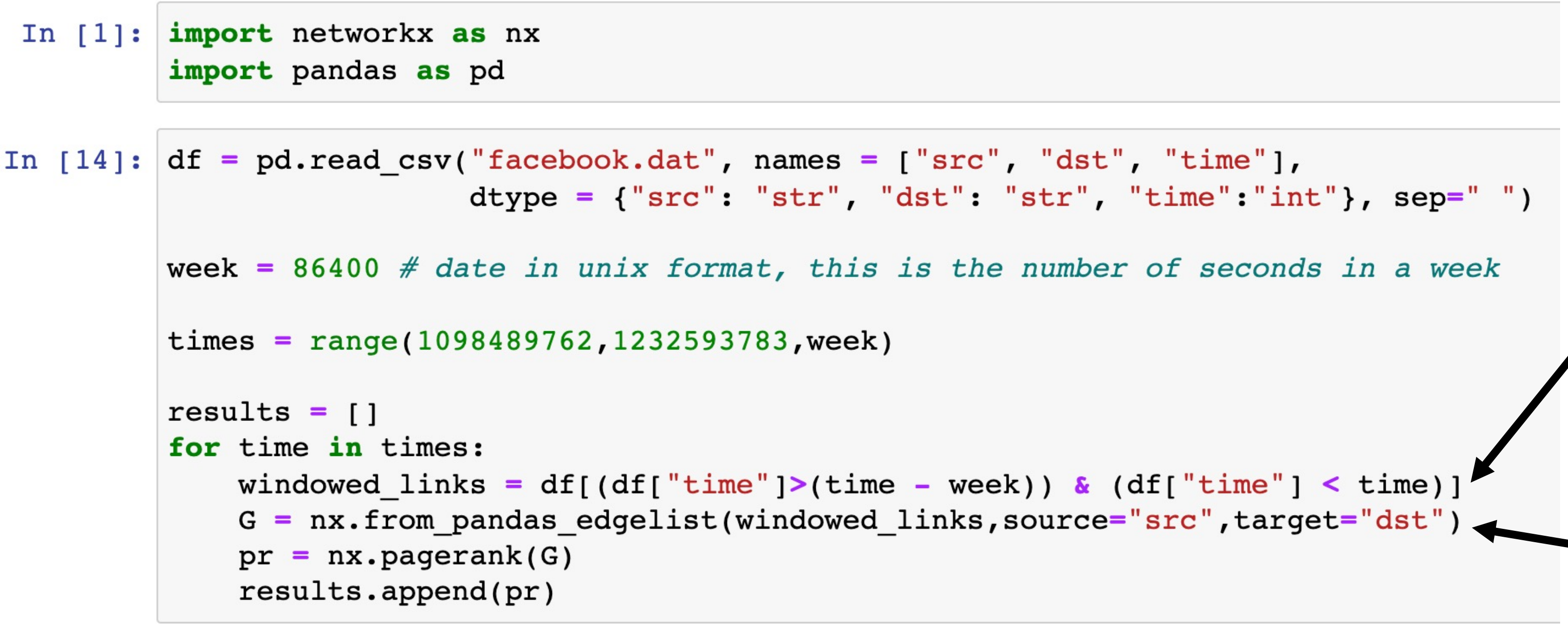
What about if I want monthly/annual/all-time? Or extra filtering?

Needed to fix some pandas errors getting the time filter to work



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

In [14]: df = pd.read_csv("facebook.dat", names = ["src", "dst", "time"],
                          dtype = {"src": "str", "dst": "str", "time":"int"}, sep=" ")

         week = 86400 # date in unix format, this is the number of seconds in a week

         times = range(1098489762,1232593783,week)

         results = []
         for time in times:
             windowed_links = df[(df["time"]>(time - week)) & (df["time"] < time)]
             G = nx.from_pandas_edgelist(windowed_links,source="src",target="dst")
             pr = nx.pagerank(G)
             results.append(pr)
```

The code can be inefficient and not scale well



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

In [14]: df = pd.read_csv("facebook.dat", names = ["src", "dst", "time"],
                          dtype = {"src": "str", "dst": "str", "time":"int"}, sep=" ")

         week = 86400 # date in unix format, this is the number of seconds in a week

         times = range(1098489762,1232593783,week)

         results = []
         for time in times:
             windowed_links = df[(df["time"]>(time - week)) & (df["time"] < time)]
             G = nx.from_pandas_edgelist(windowed_links,source="src",target="dst")
             pr = nx.pagerank(G)
             results.append(pr)
```

Making several passes over the data, one for each time point

Constructing and discarding many individual graph objects

Not ideal for **large** networks or **long time periods.**

Other models needing bespoke solutions

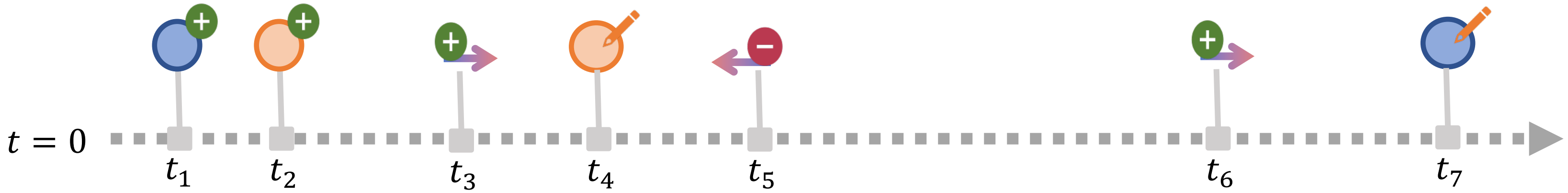
- What about if the dataset was of **network snapshots**?
- What about if you wanted stats of the **aggregate graph**?
- What about nodes/edges with **duration**?

All these things are hard to adapt existing tools for, because they are <u>not designed with temporal networks in mind.</u>

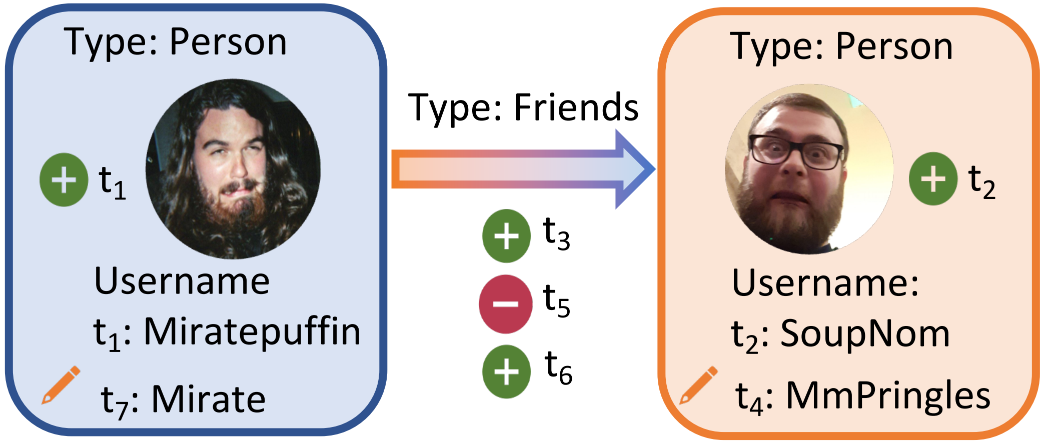
Raphtory Graph Model

Raphtory's Graph Model

- Input: stream of **graph changes**
- Happening at **specific timestamps**



- Storage: full **graph history**
- Node and edge **property history**



Type: Person

t_1

Username

t_1: Miratepuffin

t_7: Mirate

Type: Friends

t_3

t_5

t_6

Type: Person

t_2

Username:

t_2: SoupNom

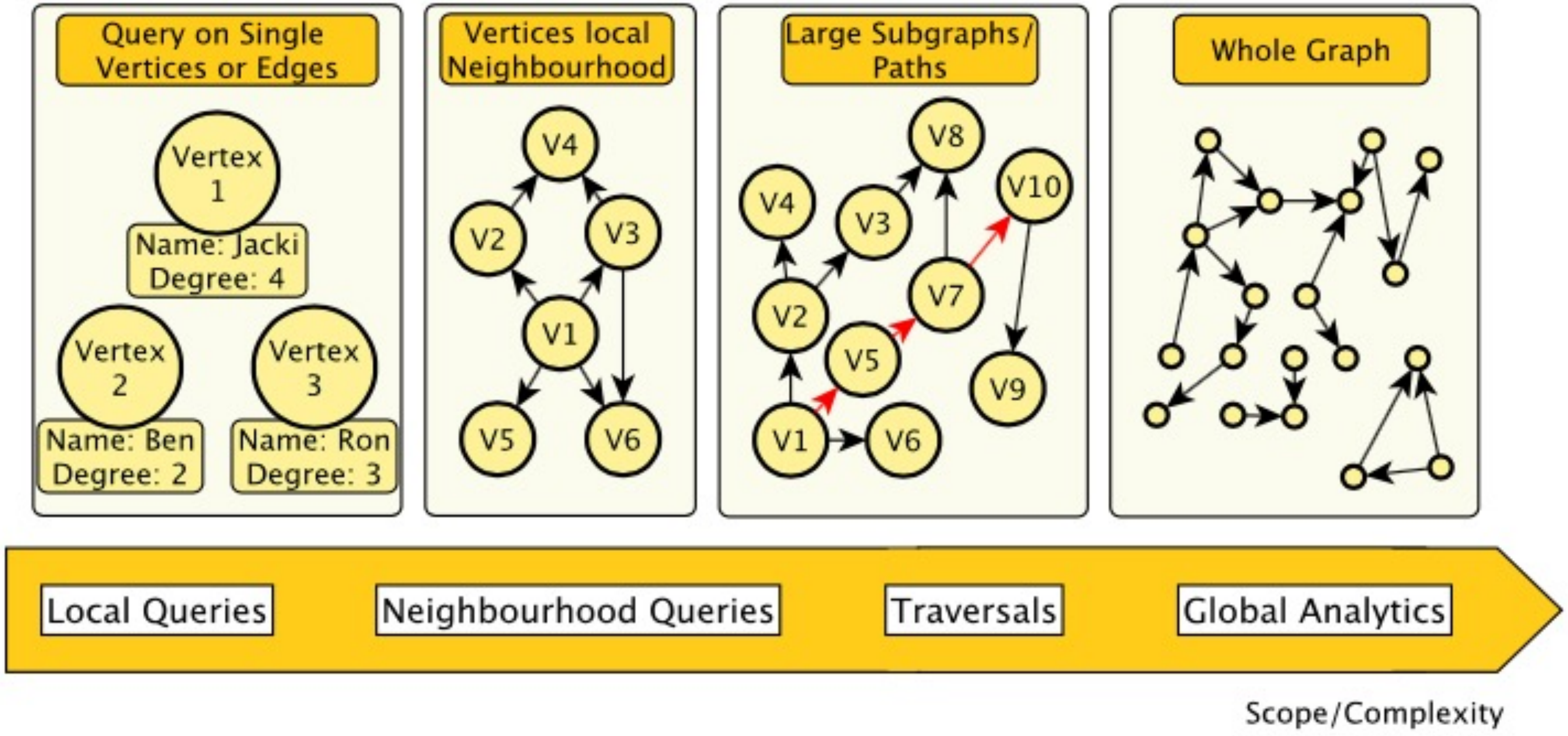
t_4: MmPringles

What temporal graph models can Raphtory understand?

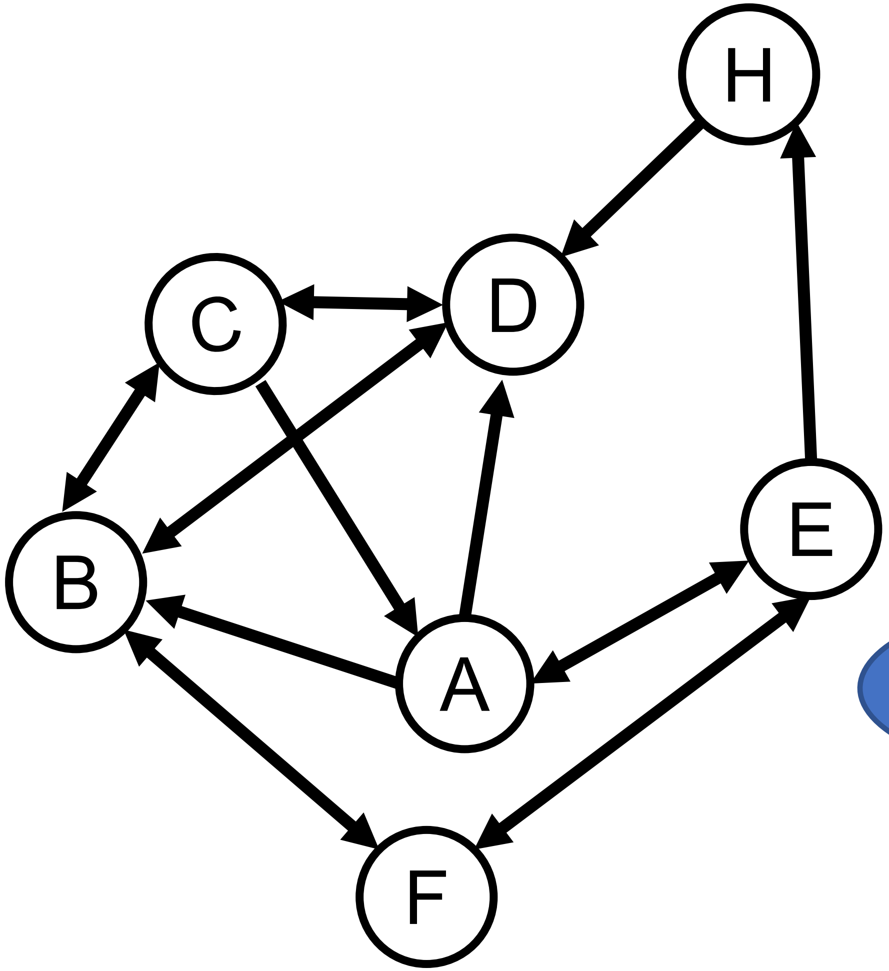
- **Snapshots** – individual datasets for each time point
- **Link streams** – instantaneous events between vertices such as online interactions
- **Interval graphs** – vertices and edges have a creation and removal time
- **Time windows** – the graph according to updates within a given time interval

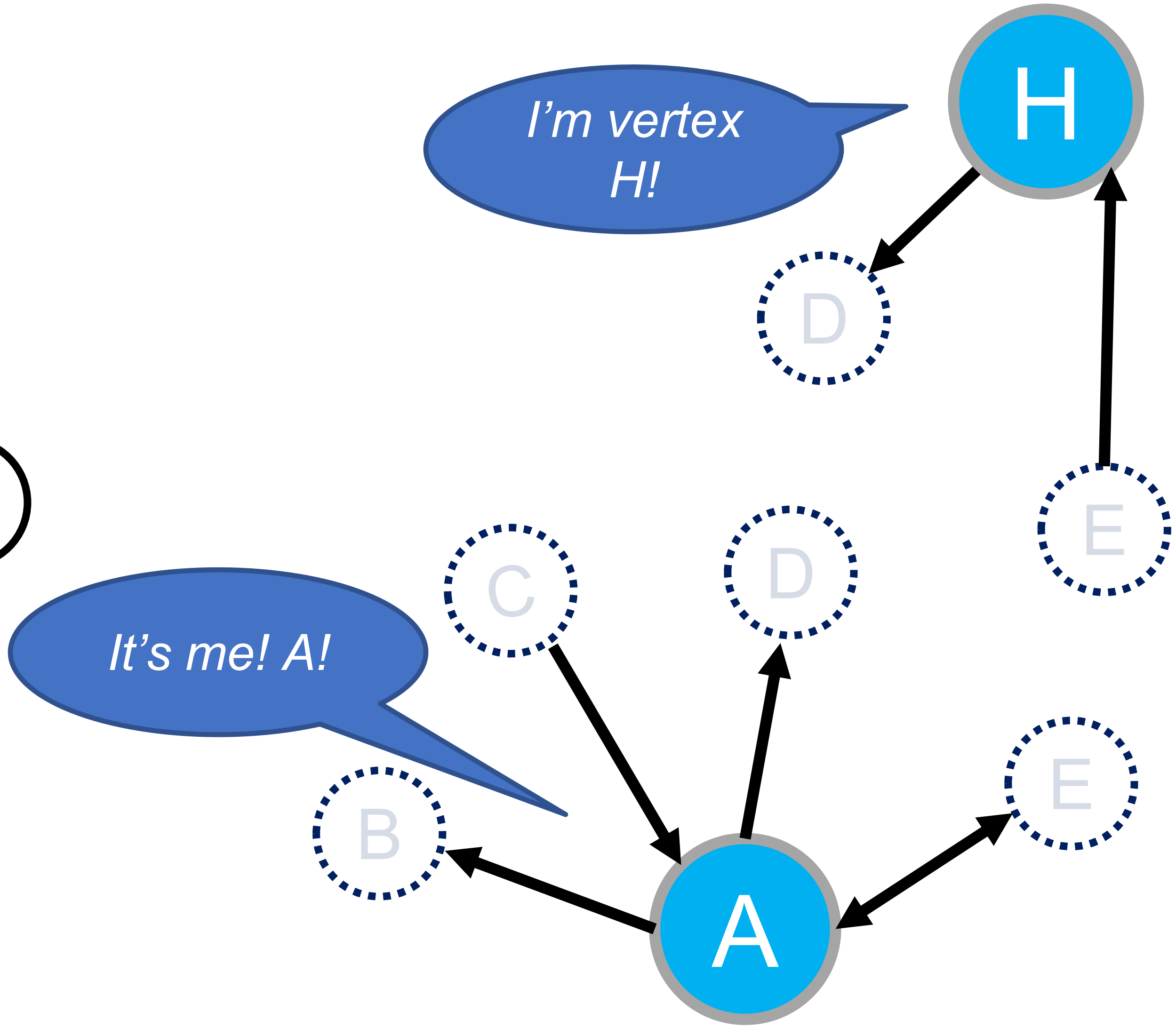
Algorithms in Raphtory

Algorithms in Raphtory



Thinking of graphs -> thinking like a vertex

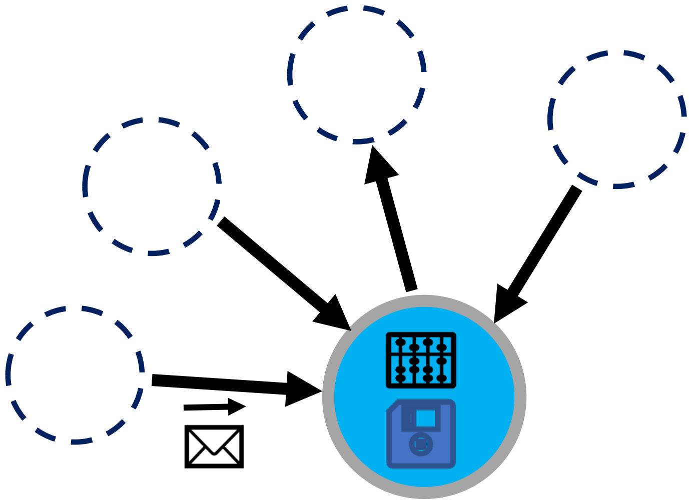




Thinking like a vertex - Gather Apply Scatter

Each vertex **knows**:

- Its own **ID**, **properties**, **history** (within the perspective scope)
- Information on its incoming and outgoing edges – including the ID of nodes **directly connected** to it



ID:
Properties:

Each vertex **can**:

✉ Send and receive **messages** along incoming and outgoing edges.

🧮 Perform **computations**

💾 Store **state** or values for ongoing computation

Thinking like a vertex: What this means

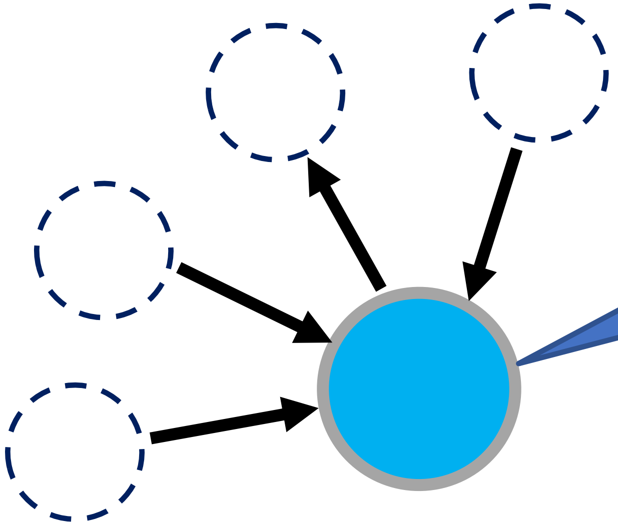
Through repeated message passing and computation steps, knowledge of **wider graph quantities** can be obtained

Vertices can be **partitioned** on multiple machines/cores offering **scalability benefits**

Many useful graph algorithms can be **expressed** using this paradigm

Local (0-step) algorithms

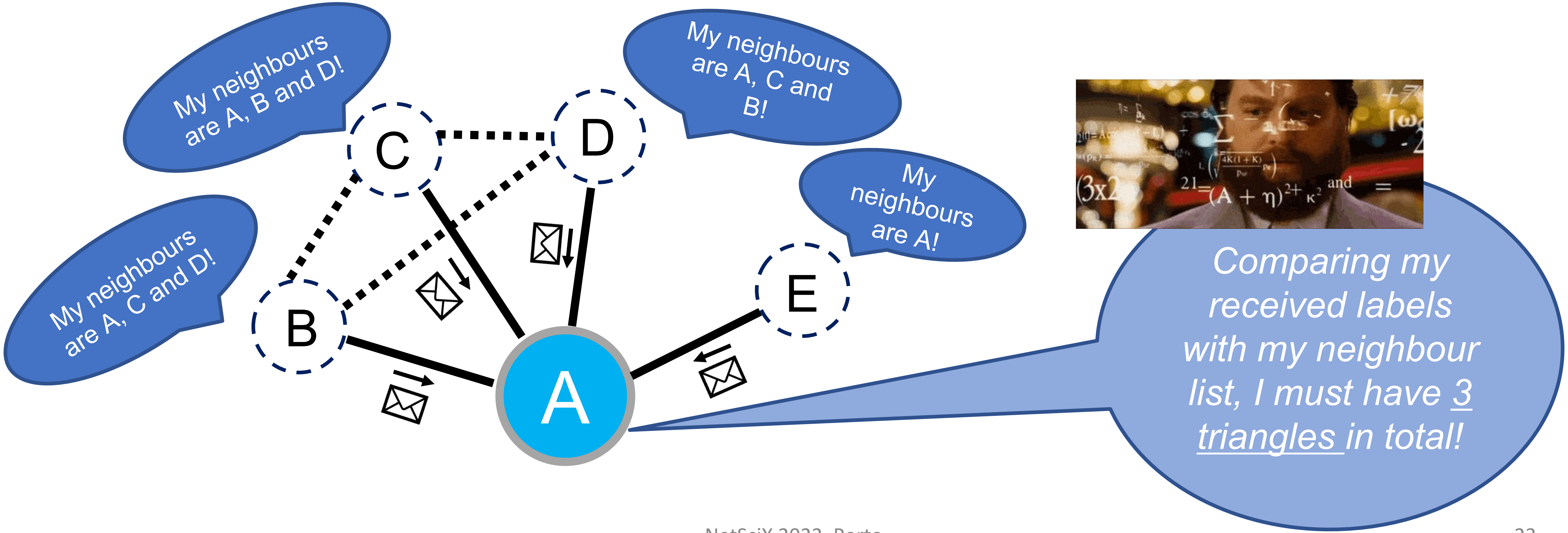
Some algorithms require no messaging step such as degree or obtaining a property



My indegree is 3 and my outdegree is 1!

One-step algorithm

Algorithms with one messaging step include **triangle count**, clustering coefficient, average neighbour degree.



My neighbours are A, B and D!

My neighbours are A, C and B!

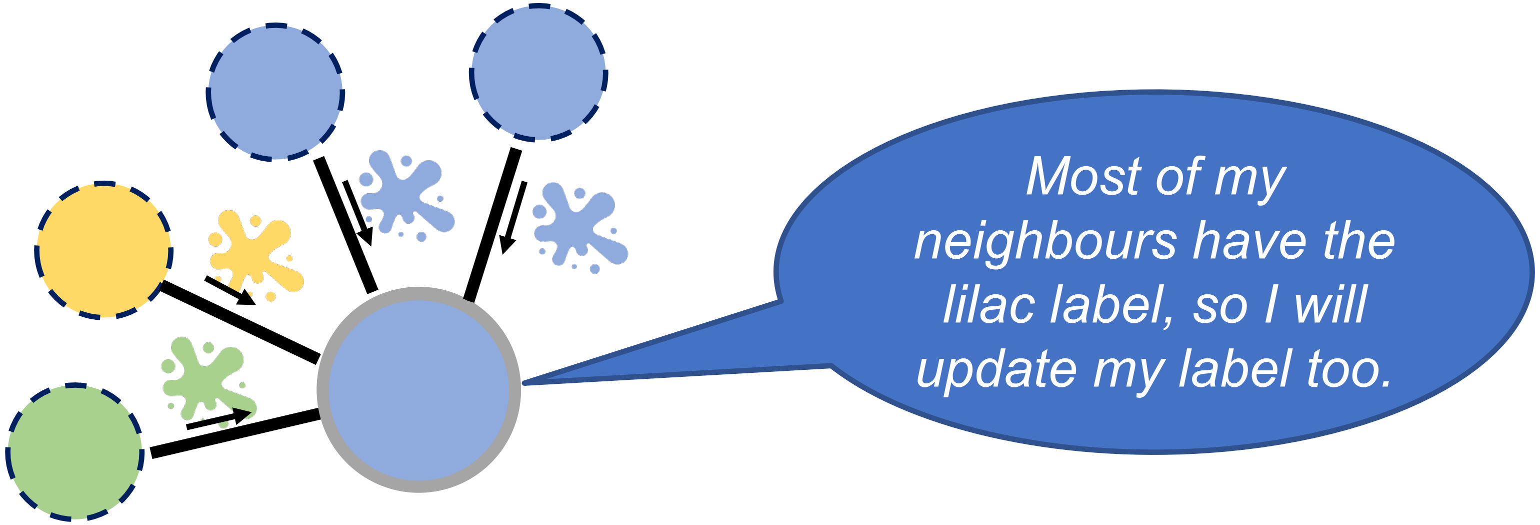
My neighbours are A, C and D!

My neighbours are A!

Comparing my received labels with my neighbour list, I must have 3 triangles in total!

Iterative algorithms

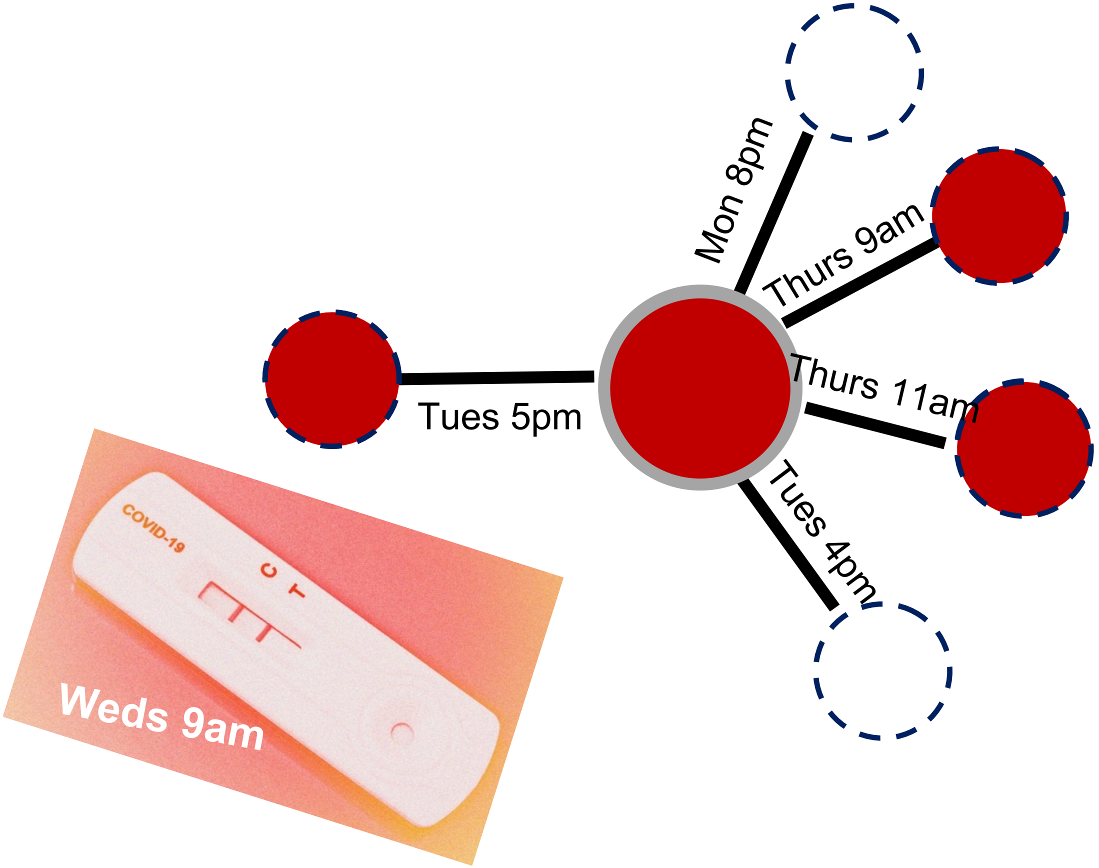
Some algorithms require an unknown number of message steps before convergence, e.g. PageRank, connected components, **community detection.**


Most of my neighbours have the lilac label, so I will update my label too.

Carry on until no labels change, this will give a *community partition* of the network

Temporal algorithms

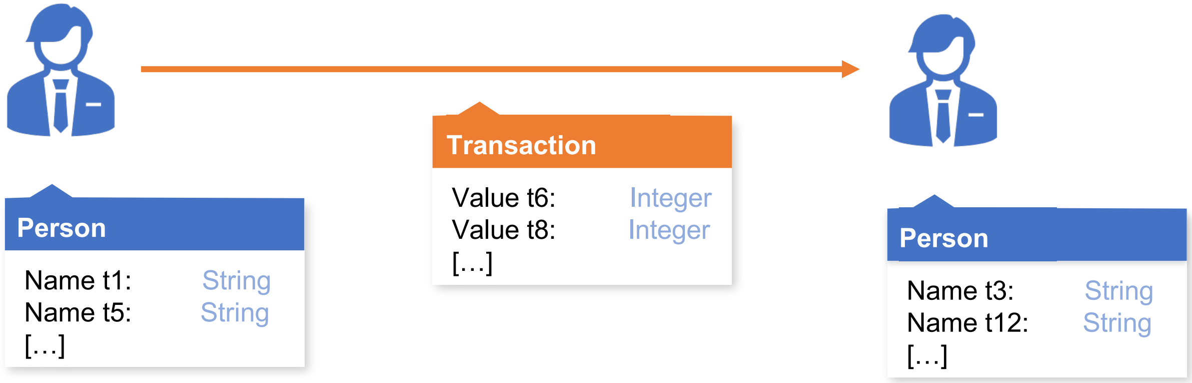
Time information can also be incorporated into these algorithms.



Can propagate messages along **time respecting edges** to trace the path of processes happening on the network.

Once we have an algorithm what can we do with it?

Graph perspectives and queries



Person

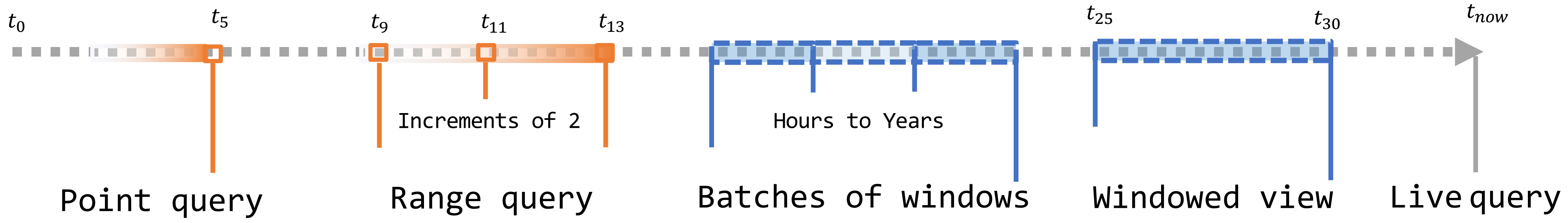
Name t1: String
Name t5: String
[…]

Transaction

Value t6: Integer
Value t8: Integer
[…]

Person

Name t3: String
Name t12: String
[…]



Increments of 2

Hours to Years

Point query

Range query

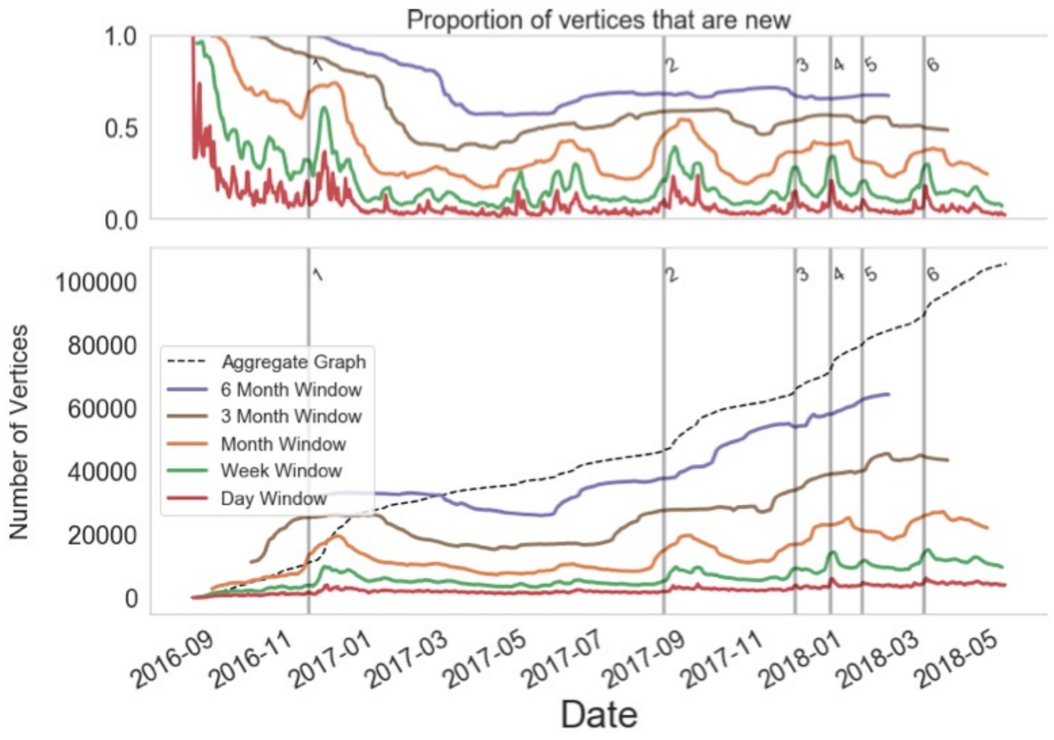
Batches of windows

Windowed view

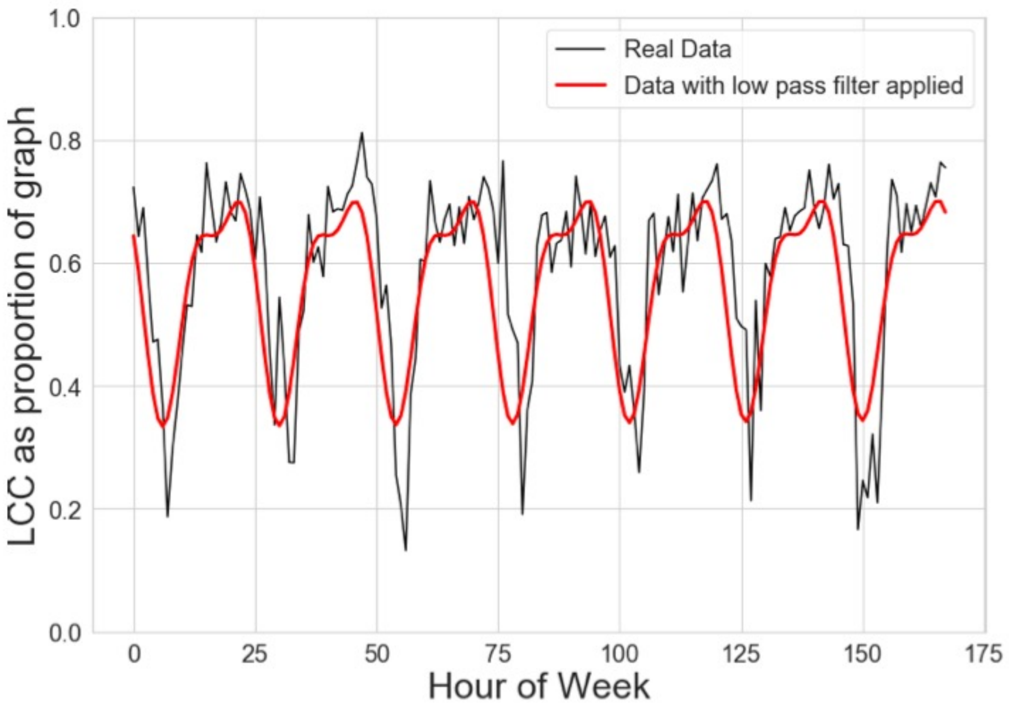
Live query

Putting it all together

Quick dive into our analysis of the social network Gab.ai

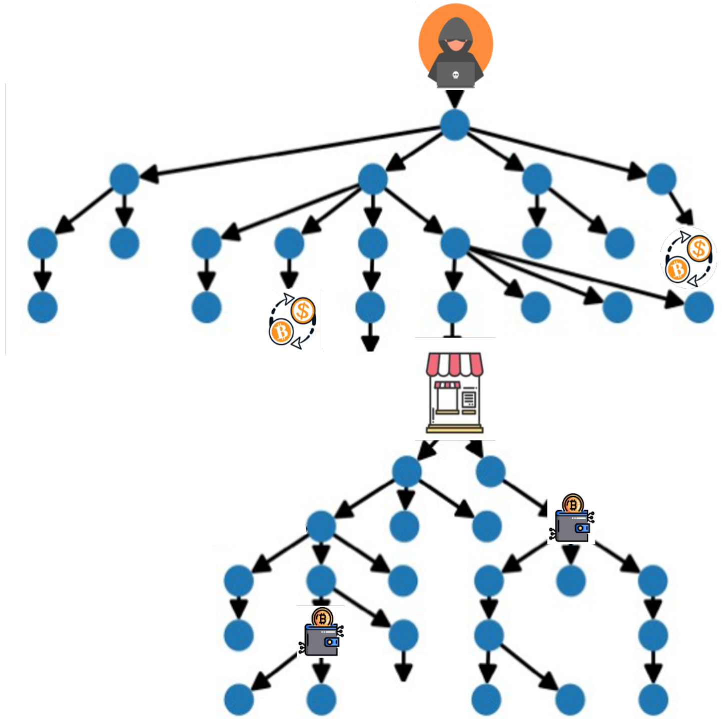


(a) Number of users

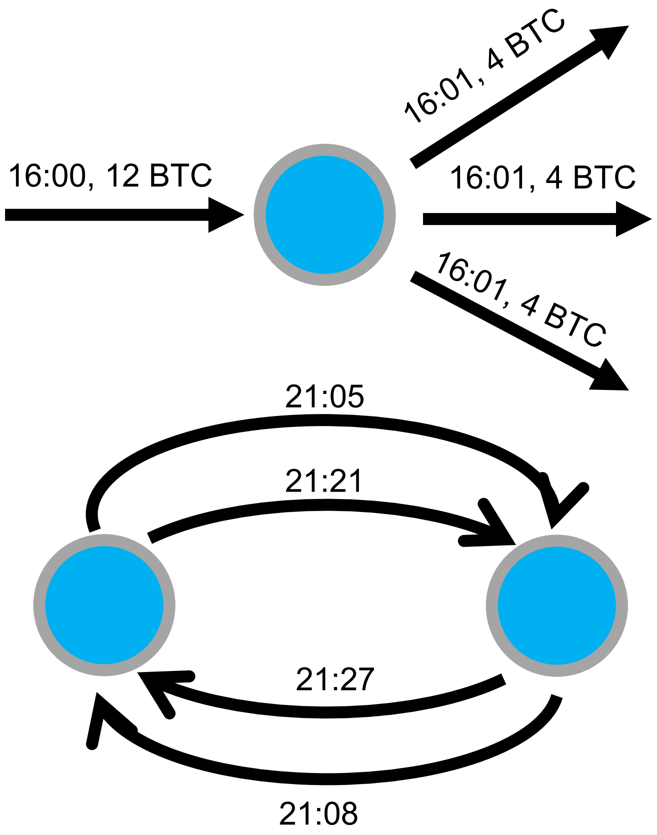


Taken from: https://dl.acm.org/doi/10.1145/3479591

Temporal algorithms



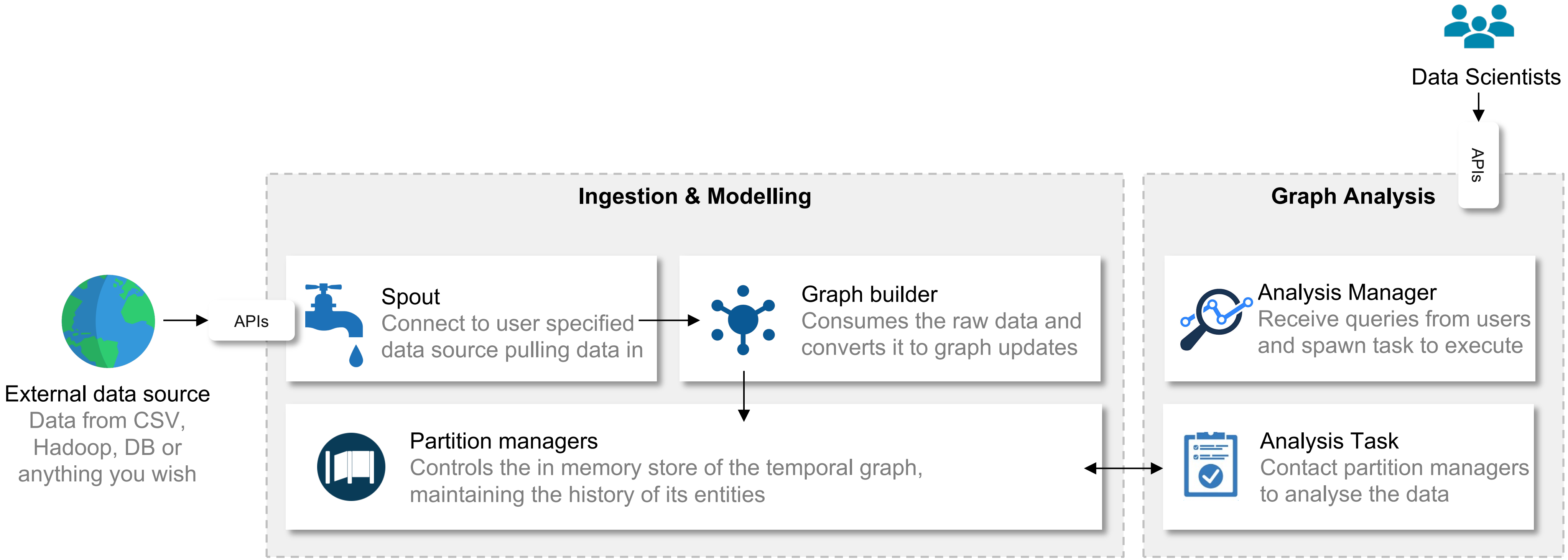
Finding time respecting paths through a cryptocurrency transaction network to track the flow of ”bad” money.



Finding occurrences of **temporal motifs** to identify patterns and anomalies.

How Raphtory Works

Raphtory Component Overview



Data Scientists

APIs

External data source
Data from CSV, Hadoop, DB or anything you wish

APIs

Ingestion & Modelling

Spout
Connect to user specified data source pulling data in

Graph builder
Consumes the raw data and converts it to graph updates

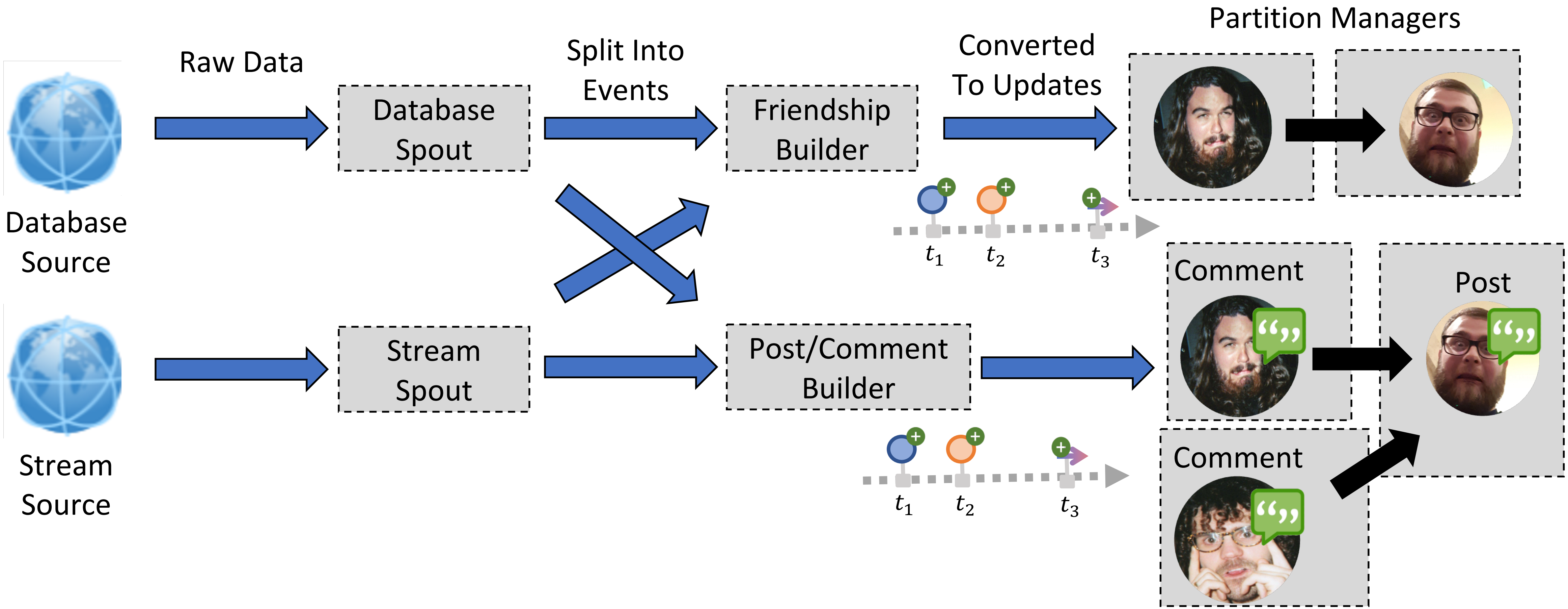
Partition managers
Controls the in memory store of the temporal graph, maintaining the history of its entities

Graph Analysis

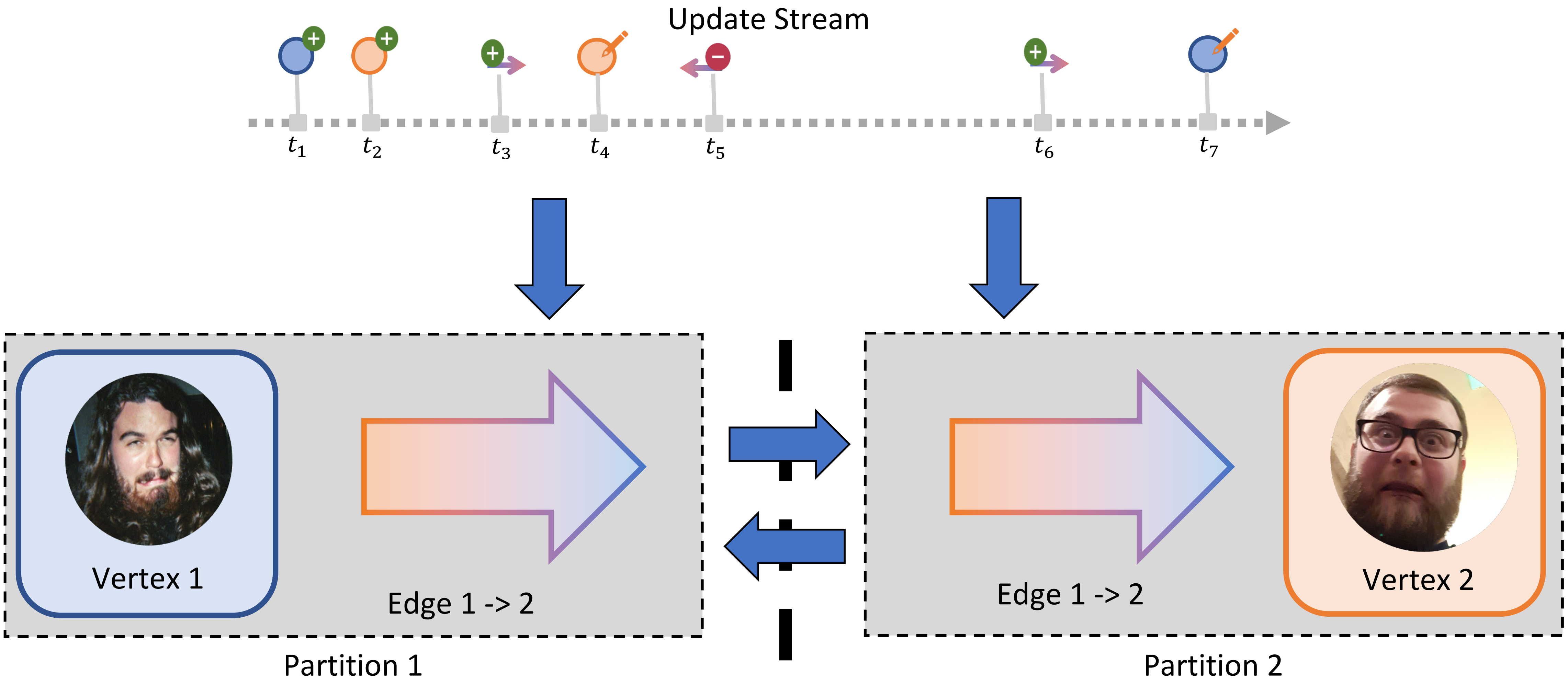
Analysis Manager
Receive queries from users and spawn task to execute

Analysis Task
Contact partition managers to analyse the data

Getting your data in

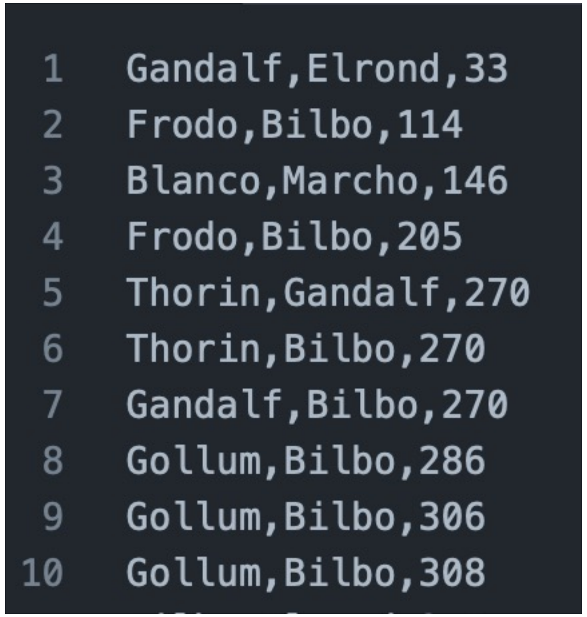


Partition Managers



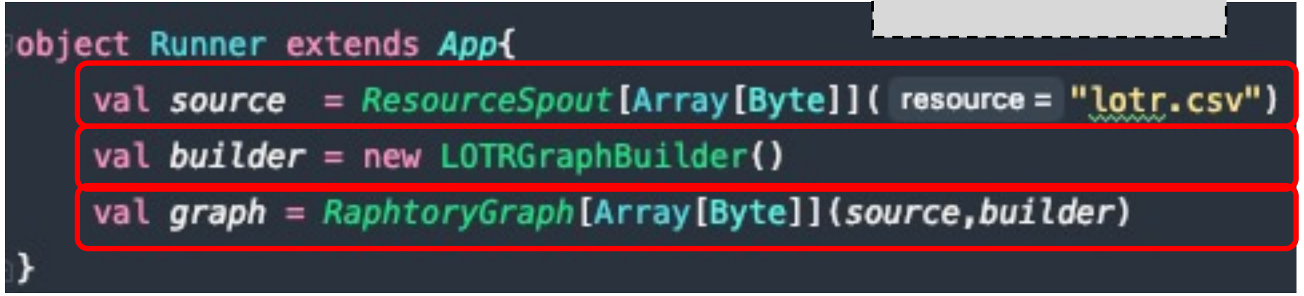
Building your graph in practice

Character Interaction dataset



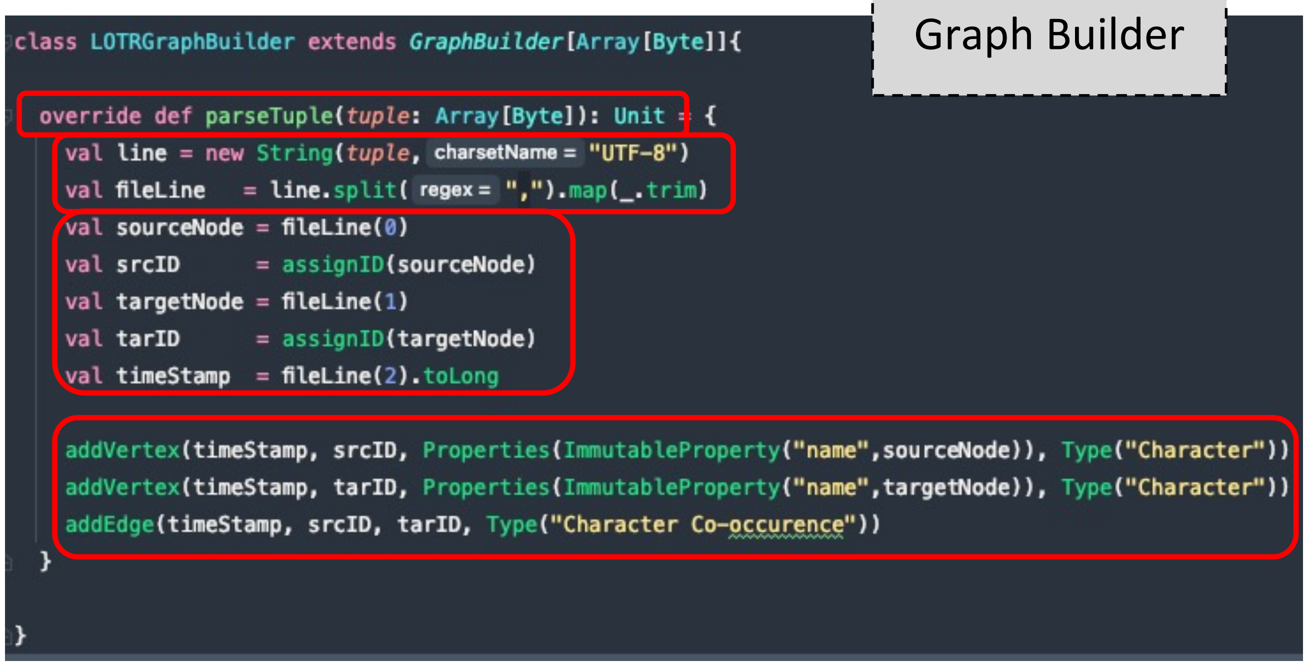
```
Gandalf,Elrond,33
Frodo,Bilbo,114
Blanco,Marcho,146
Frodo,Bilbo,205
Thorin,Gandalf,270
Thorin,Bilbo,270
Gandalf,Bilbo,270
Gollum,Bilbo,286
Gollum,Bilbo,306
Gollum,Bilbo,308
```

Raphtory Graph



```
object Runner extends App{
    val source  = ResourceSpout[Array[Byte]]( resource = "lotr.csv")
    val builder = new LOTRGraphBuilder()
    val graph = RaphtoryGraph[Array[Byte]](source,builder)
}
```

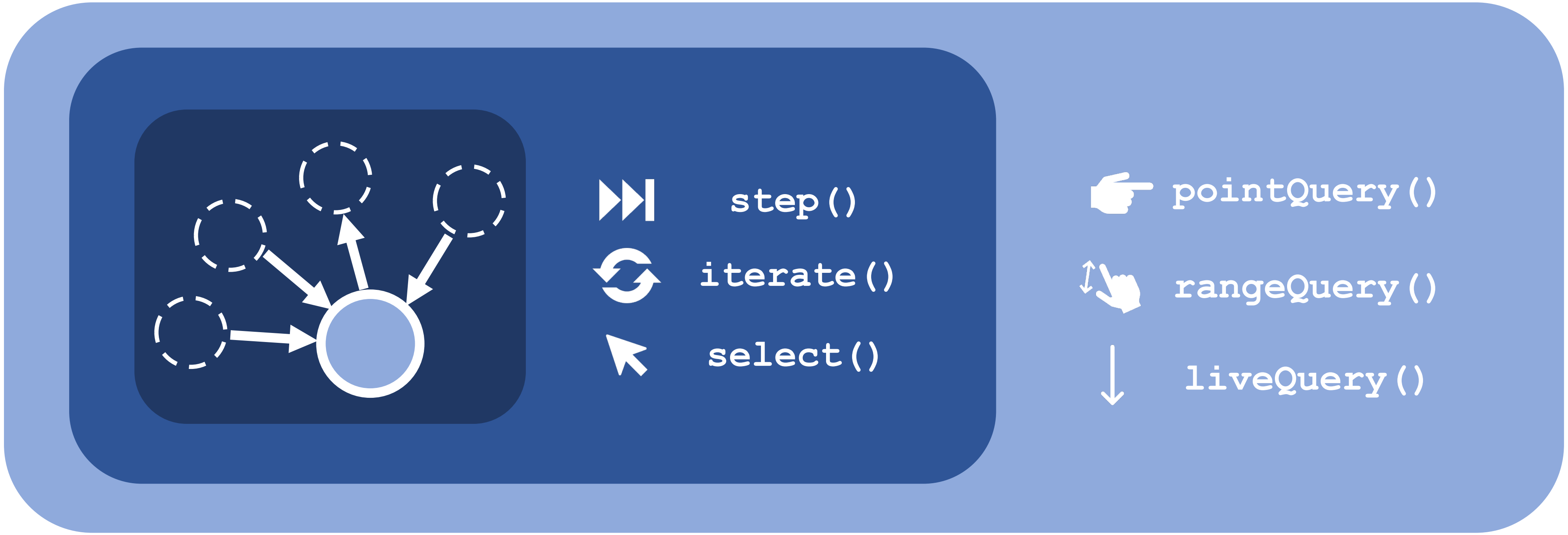
Graph Builder



```
class LOTRGraphBuilder extends GraphBuilder[Array[Byte]]{

  override def parseTuple(tuple: Array[Byte]): Unit = {
    val line = new String(tuple, charsetName = "UTF-8")
    val fileLine   = line.split( regex = ",").map(_.trim)
    val sourceNode = fileLine(0)
    val srcID      = assignID(sourceNode)
    val targetNode = fileLine(1)
    val tarID      = assignID(targetNode)
    val timeStamp  = fileLine(2).toLong

    addVertex(timeStamp, srcID, Properties(ImmutableProperty("name",sourceNode)), Type("Character"))
    addVertex(timeStamp, tarID, Properties(ImmutableProperty("name",targetNode)), Type("Character"))
    addEdge(timeStamp, srcID, tarID, Type("Character Co-occurence"))
  }

}
```

Algorithm APIs

Three level API



step()

iterate()

select()

pointQuery()

rangeQuery()

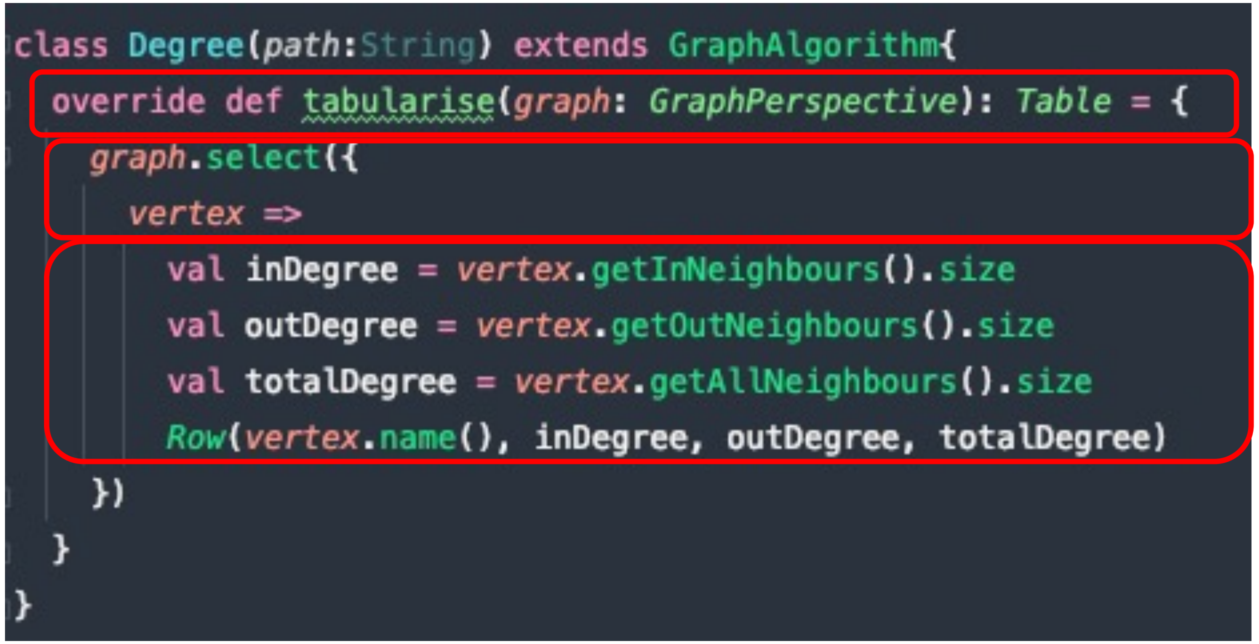
liveQuery()

Vertex and edge visitors:
Low-level querying on entities such as messaging neighbours or extracting vertex properties

Algorithm:
Defining the overall flow of an algorithm using step, iterate and select functions

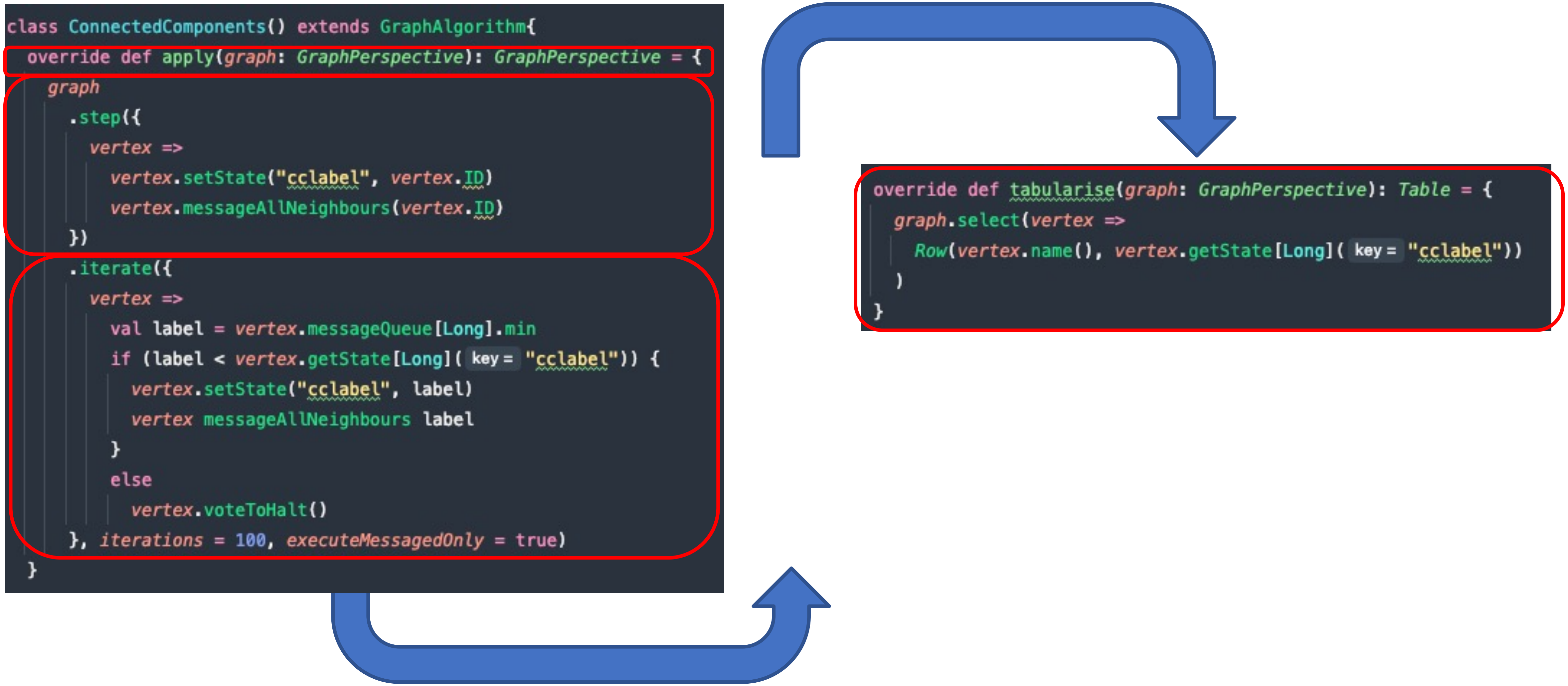
Time selection:
Defining the time scope of the graph for the algorithm to apply to

Local (0-step) algorithm - Degree



```scala
class Degree(path:String) extends GraphAlgorithm{
  override def tabularise(graph: GraphPerspective): Table = {
    graph.select({
      vertex =>
        val inDegree = vertex.getInNeighbours().size
        val outDegree = vertex.getOutNeighbours().size
        val totalDegree = vertex.getAllNeighbours().size
        Row(vertex.name(), inDegree, outDegree, totalDegree)
    })
  }
}
```

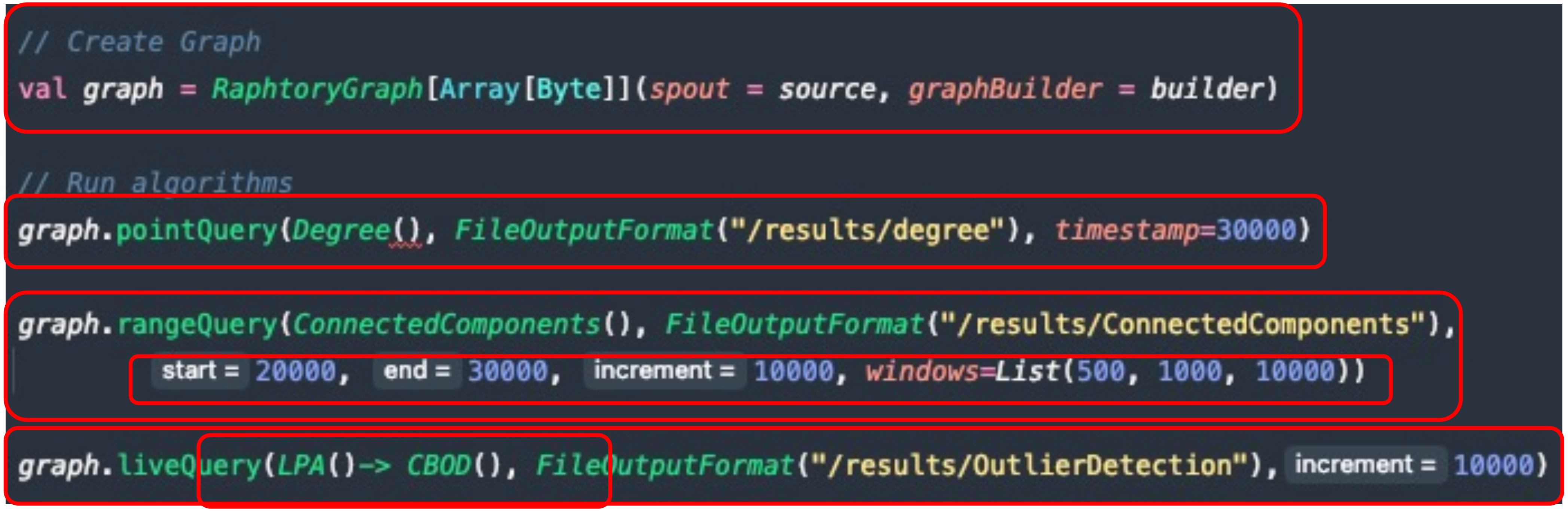
Iterative algorithms – Connected Components



```
class ConnectedComponents() extends GraphAlgorithm{
  override def apply(graph: GraphPerspective): GraphPerspective = {
    graph
      .step({
        vertex =>
          vertex.setState("cclabel", vertex.ID)
          vertex.messageAllNeighbours(vertex.ID)
      })
      .iterate({
        vertex =>
          val label = vertex.messageQueue[Long].min
          if (label < vertex.getState[Long]( key = "cclabel")) {
            vertex.setState("cclabel", label)
            vertex messageAllNeighbours label
          }
          else
            vertex.voteToHalt()
      }, iterations = 100, executeMessagedOnly = true)
  }
```

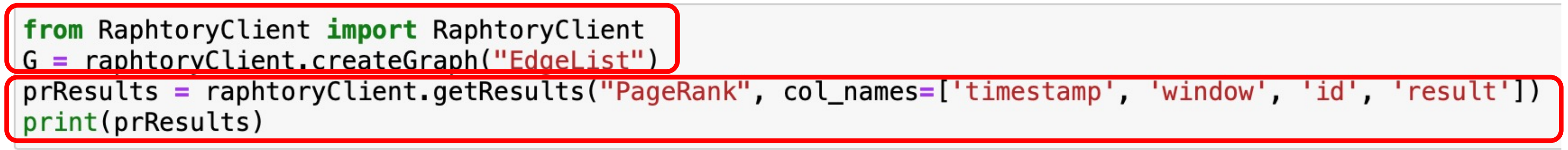
```
override def tabularise(graph: GraphPerspective): Table = {
  graph.select(vertex =>
    Row(vertex.name(), vertex.getState[Long]( key = "cclabel"))
  )
}
```

Submitting these queries

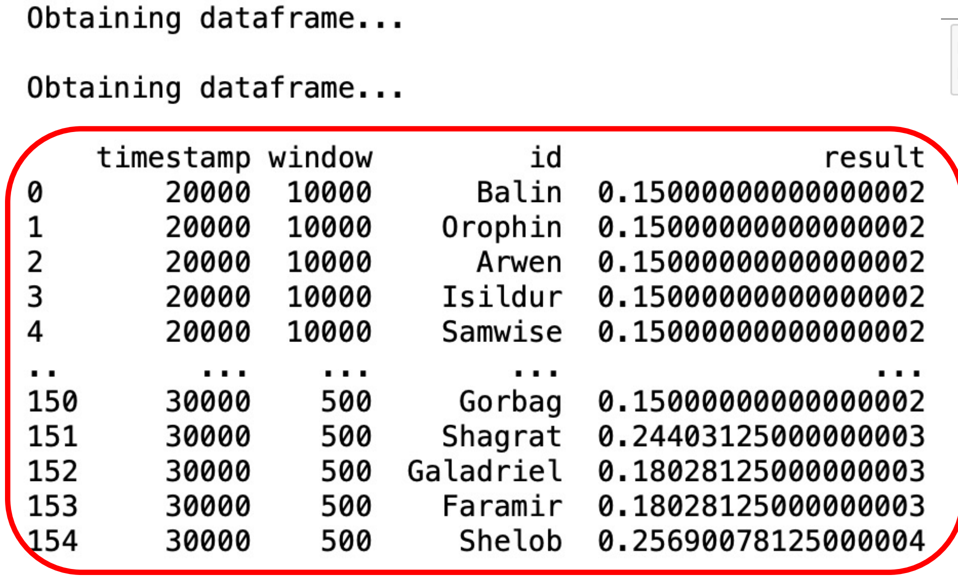


```
// Create Graph
val graph = RaphtoryGraph[Array[Byte]](spout = source, graphBuilder = builder)

// Run algorithms
graph.pointQuery(Degree(), FileOutputFormat("/results/degree"), timestamp=30000)

graph.rangeQuery(ConnectedComponents(), FileOutputFormat("/results/ConnectedComponents"),
        start = 20000, end = 30000, increment = 10000, windows=List(500, 1000, 10000))

graph.liveQuery(LPA()-> CBOD(), FileOutputFormat("/results/OutlierDetection"), increment = 10000)
```

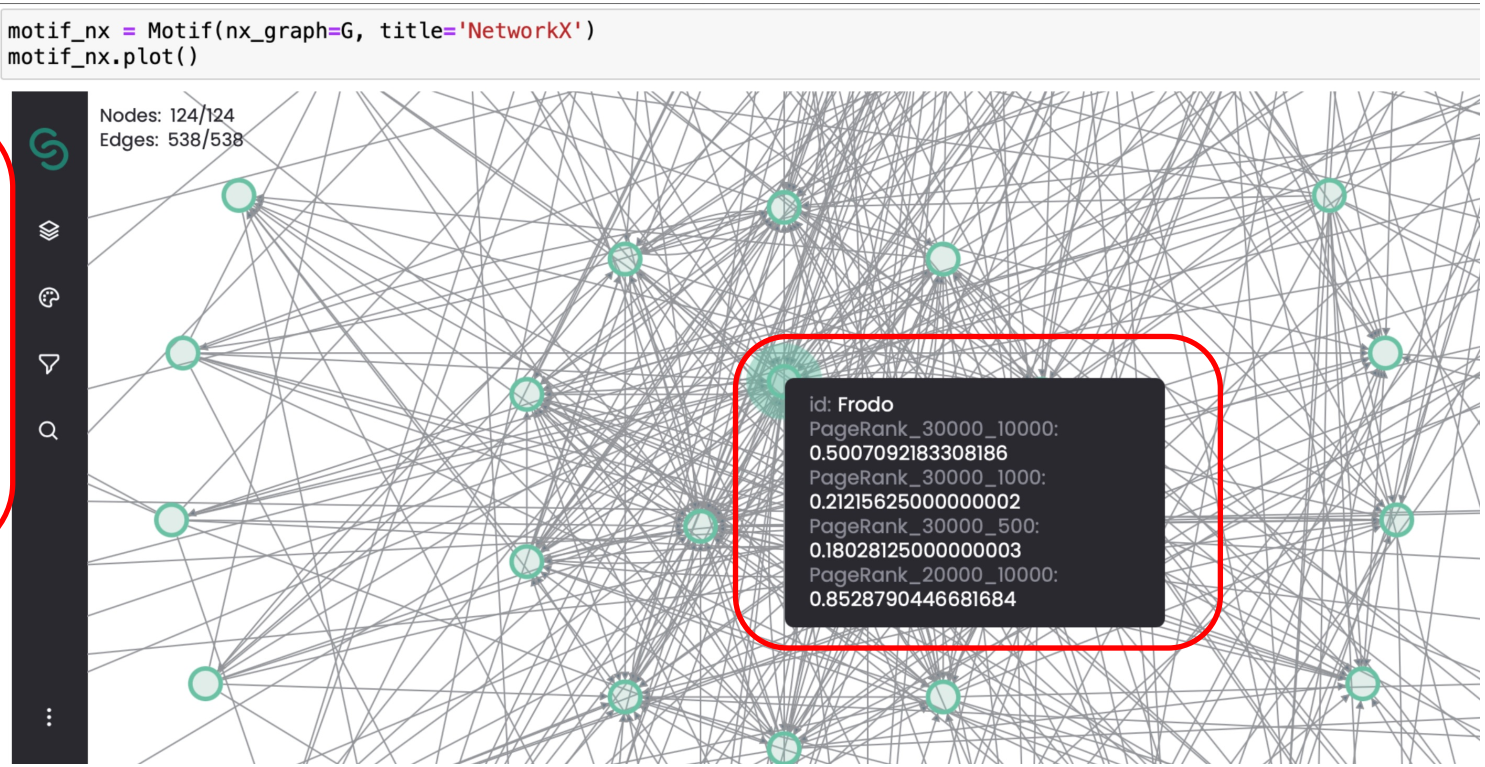
Getting your results into Jupyter



```
from RaphtoryClient import RaphtoryClient
G = raphtoryClient.createGraph("EdgeList")
prResults = raphtoryClient.getResults("PageRank", col_names=['timestamp', 'window', 'id', 'result'])
print(prResults)
```



```
Obtaining dataframe...

Obtaining dataframe...

     timestamp window        id               result
0        20000  10000     Balin  0.15000000000000002
1        20000  10000   Orophin  0.15000000000000002
2        20000  10000     Arwen  0.15000000000000002
3        20000  10000   Isildur  0.15000000000000002
4        20000  10000   Samwise  0.15000000000000002
..         ...    ...       ...                  ...
150      30000    500    Gorbag  0.15000000000000002
151      30000    500   Shagrat  0.24403125000000003
152      30000    500 Galadriel  0.18028125000000003
153      30000    500   Faramir  0.18028125000000003
154      30000    500    Shelob  0.25690078125000004
```



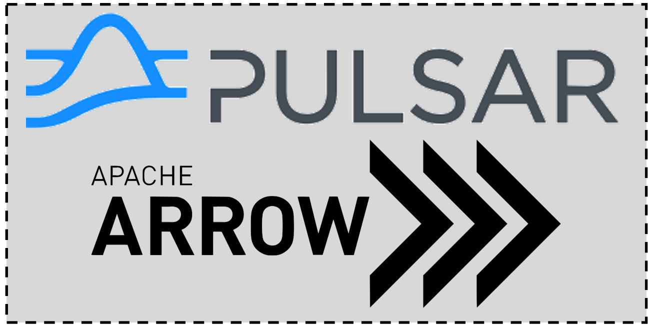
```
motif_nx = Motif(nx_graph=G, title='NetworkX')
motif_nx.plot()
```

Nodes: 124/124
Edges: 538/538

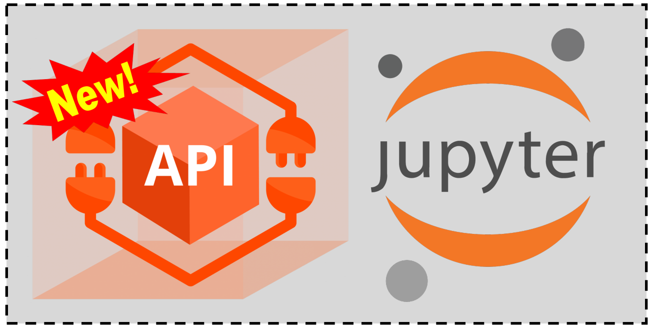
id: Frodo
PageRank_30000_10000:
0.5007092183308186
PageRank_30000_1000:
0.21215625000000002
PageRank_30000_500:
0.18028125000000003
PageRank_20000_10000:
0.8528790446681684

On the horizon

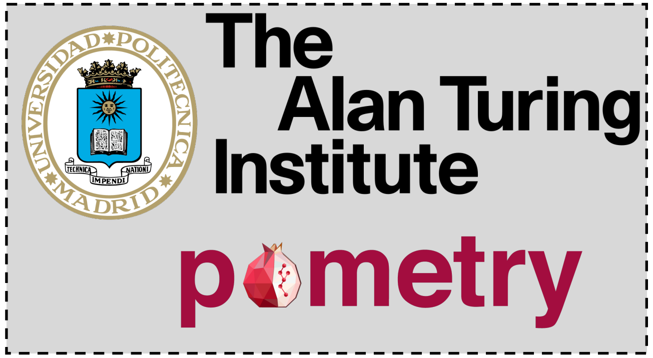
What’s on the horizon



Rebuilding the platform
for the largest of datasets



New API’s and tool integration
to best interrogate temporal networks



Research and Industry positions
for those that are interested

Best ways to get involved

Join the conversation on slack

tinyurl.com/5n8vrt6e

Come give Raphtory a try

github.com/Raphtory/Raphtory

github.com/Raphtory/Raphtory

@raphtory

raphtory.readthedocs.io

Thanks for listening!

What are your questions?

tinyurl.com/5n8vrt6e

ben.steer@pometry.com

n.a.arnold@qmul.ac.uk