

---

---

# What The HELK?

Enabling Graph Analytics for Effective  
— Threat Hunting —



Label: Threat Hunter  
Twitter: @Cyb3rWard0g  
Name: Roberto Rodriguez

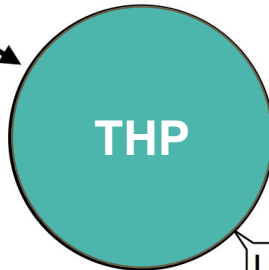


DEVELOPED



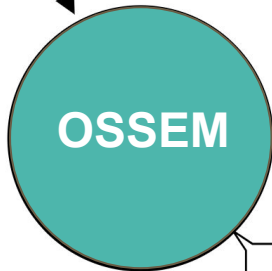
Label: GitHub Project  
Title: HELK  
Description: Hunting ELK

DEVELOPED



Label: GitHub Project  
Title: ThreatHunter-Playbook

DEVELOPED



Label: GitHub Project  
Title: OSSEM  
Description: Open Source Security Event Metadata

WORKS



Label: Company  
Company Name: SpecterOps  
Team: Adversary Detection

# Agenda

- Effective Threat Hunting?
- Current state of threat hunting programs
- Graph Theory
  - Definition
  - Origins
  - Types
- Graph Analytics (Queries, Algorithms, Analytics)
  - Blue & Red embracing graph analytics
- An ELK with Graphing capabilities
  - HELK
  - Spark & GraphFrames
- GraphFrames examples
- Further Research

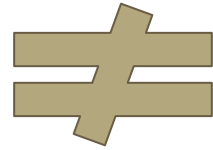
# \_\_\_\_\_ for Effective Threat Hunting

How are you effective?

What does being effective even mean?

# Efficiency

# Efficacy



# Effectiveness

<https://twitter.com/Cyb3rPandaH>

# Efficiency

The way resources are used (or wasted), How much I make the most of the resources I have



# Efficacy

It doesn't matter how we do it, but only on what we accomplish

# Effectiveness

Accomplishes the goals (to be efficacious) employing the best and most economic methodology (to be efficient).



# Efficiency



- Choosing an adversary model (MITRE ATT&CK)
- Do we even have the data?
  - Do we have the right data?
- Do we have the right technology
  - SQL, NOSQL, Graph Database
- The right skills in the team
- Prioritizing adversary techniques

# Efficacy



- Let's find evil!
- Uncovering Incidents vs Validating Detection of adversaries
- Detect all attack variations!! Can you?

# Current State of Threat Hunting

LOG IT ALL -> HUNT -> FIND EVIL ... Right??



# Threat Hunting

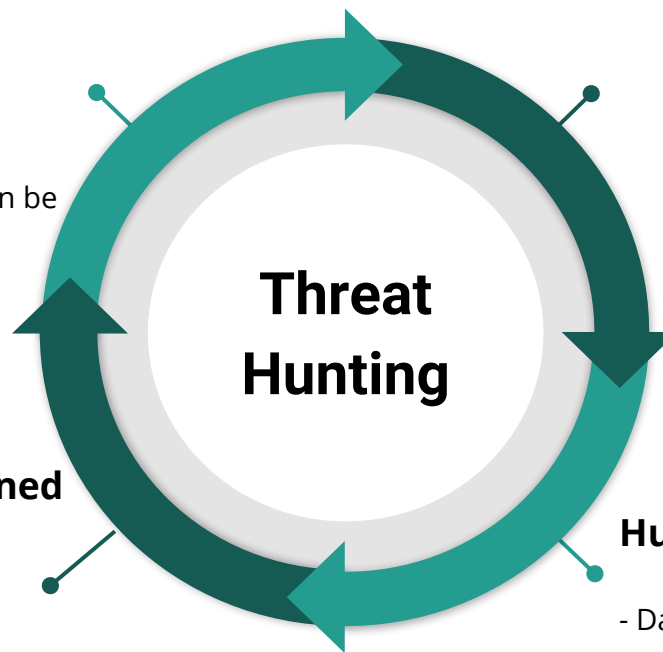


## What can be automated?

- Not everything can be automated
- Enhance SOC operations

## Lessons Learned

- Metrics
- Report Findings
- Transition to IR?
- What didn't work?



## Pre-Hunt

- Identify Data Sources
- Define Hunt Model
- Set Scope
- Define Team Roles
- Research
- Develop Hypothesis

## Hunt

- Data Analytics
  - > Behavioral
  - > Anomalies/Outliers
- Validate Detection

# Threat Hunting

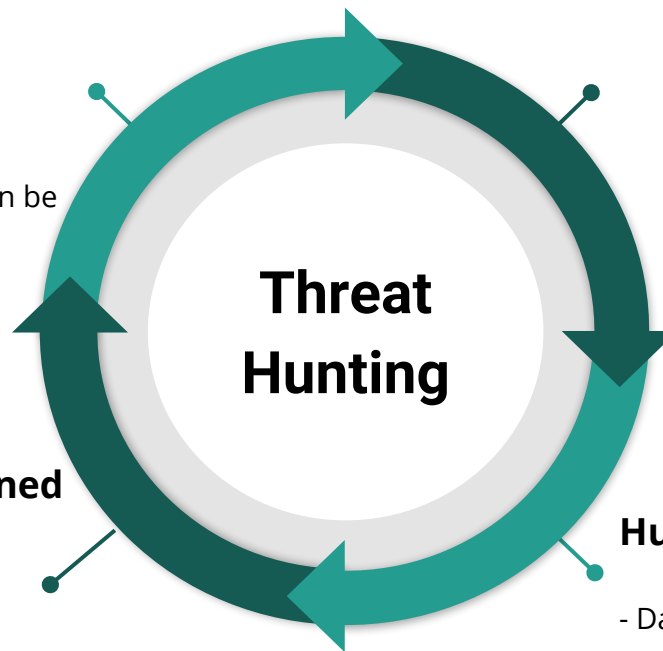


## What can be automated?

- Not everything can be automated
- Enhance SOC operations

## Lessons Learned

- Metrics
- Report Findings
- Transition to IR?
- What didn't work?



## Pre-Hunt

- Identify Data Sources
- Define Hunt Model
- Set Scope
- Define Team Roles
- Research
- Develop Hypothesis

## Hunt

- Data Analytics
  - > Behavioral
  - > Anomalies/Outliers
- Validate Detection

# Threat Hunting

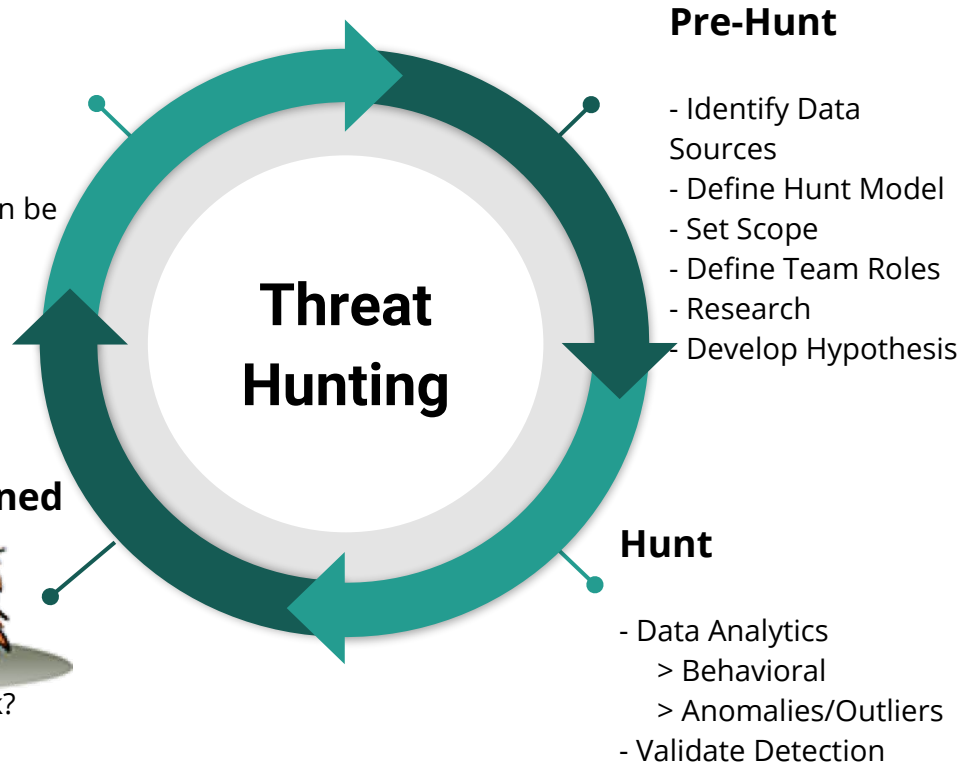


## What can be automated?

- Not everything can be automated
- Enhance SOC operations

## Lessons Learned

- Metrics
- Report Findings
- Transition to IR?
- What didn't work?



# More data more problems?

- We are generating more data than ever!
- Collecting and storing security event data has become an inexpensive task for organizations of all sizes
- This has benefited security analysts from a data availability perspective!
- However, there is so much data that traditional SIEM capabilities are limiting the way that data can be described or analyzed by security analysts

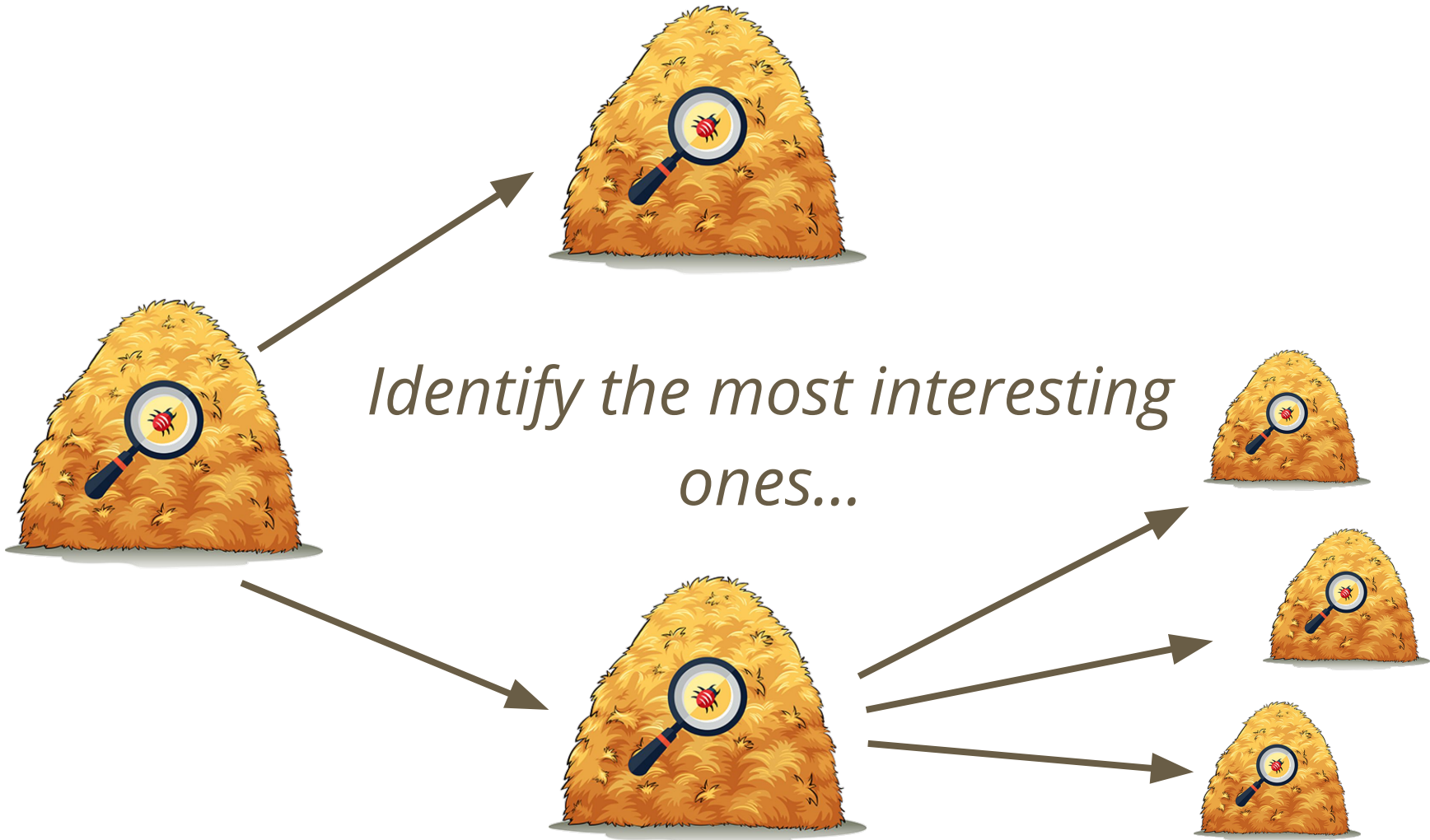


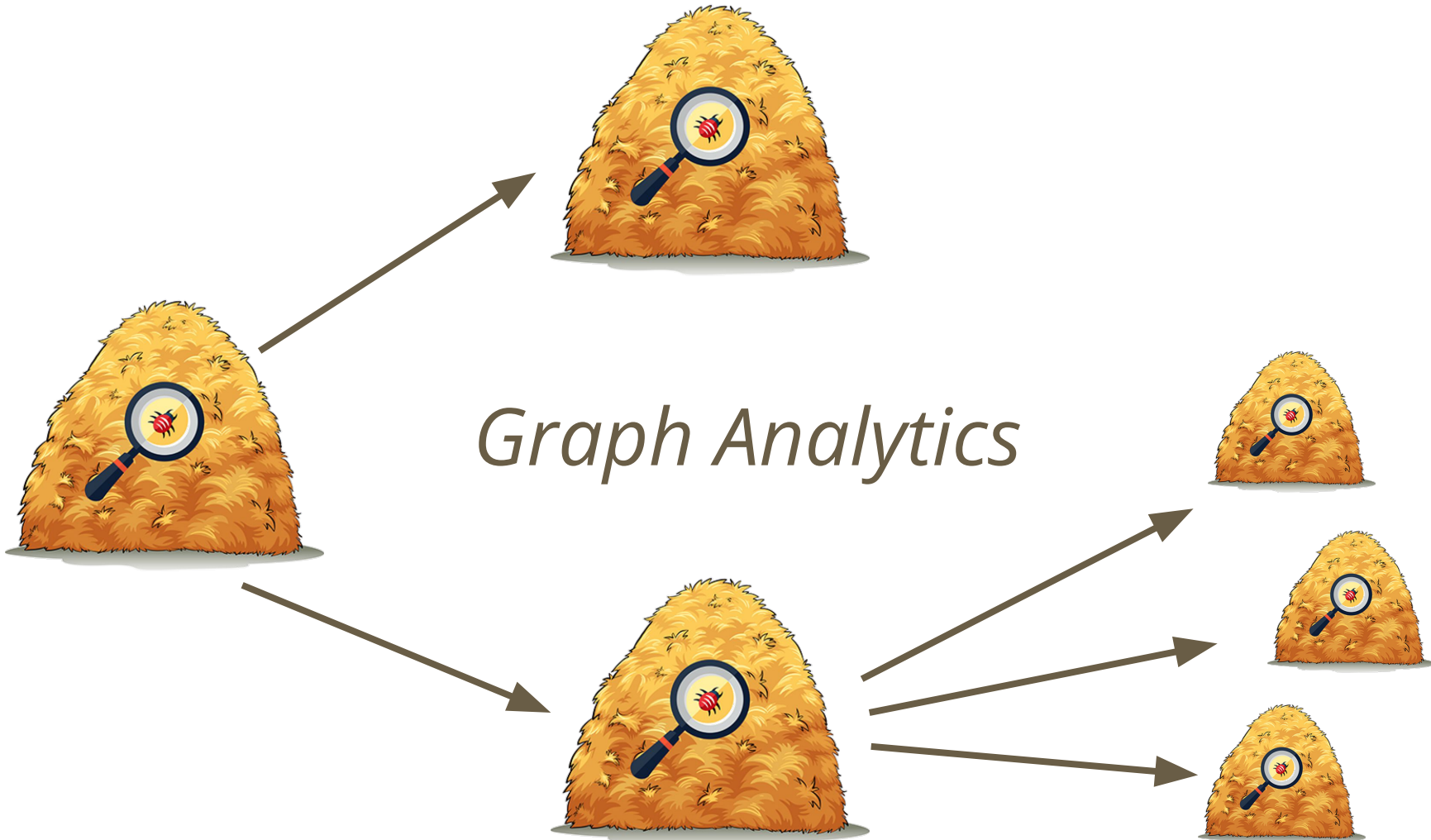
*Don't just try to find the needle in the  
haystack!*



*Find relationships & structural patterns*





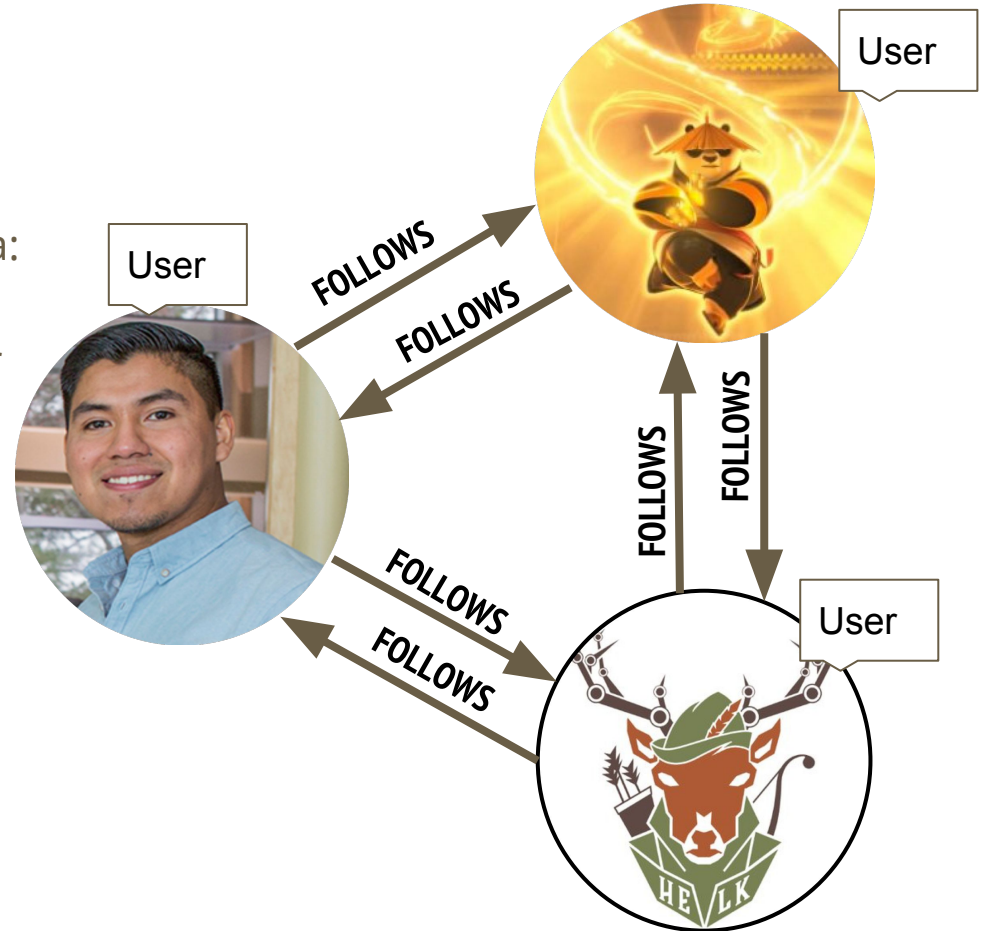


*Graph Analytics*



# What is a graph?

- A logical representation of data via:
  - Set of Vertices (Nodes)
  - Set of Edges (Relationships or links)
- Small network of Twitter
  - Vertices
    - @Cyb3rWard0g
    - @Cyb3rPandaH
    - @THE\_HELK
  - Edges
    - FOLLOWS
- Basic notation:  $\mathbf{G} = (\mathbf{V}, \mathbf{E})$



# Basic Graph Terminology (A few)

- **Order:** Size of the vertex set in a graph
- **Path:** A walk (sequence of vertices and edges)
- **Size:** Number of edges that the graph has
- **Triangle:** A cycle of length 3 in a graph
- **Walk:** A walk is an alternating sequence of vertices and edges, starting and ending at a vertex, in which each edge is adjacent in the sequence to its two endpoints.
- **Isolated:** It is a vertex whose degree is zero

# Basic Graph Terminology (A few)

- **Adjacent:** Relation between two vertices that are both endpoints of the same edge
- **Degree:** Number of edges on a vertex
- **Depth:** It is the number of edges in the path from the root to the node (vertex)
- **Neighbor:** A vertex that is adjacent to a given vertex
- **Order:** Size of the vertex set in a graph

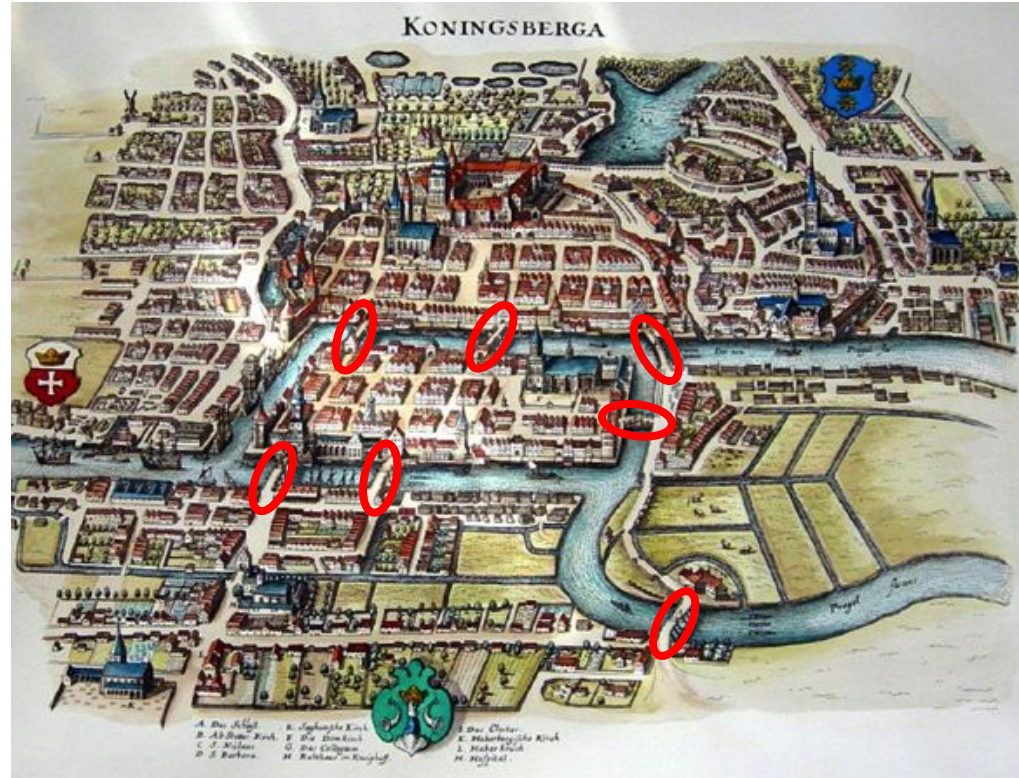
# The Origins of Graph Theory (Seven Bridges of Königsberg)

- Leonhard Euler in 1736
- Königsberg was a city in Germany that is now Kaliningrad, Russia built around a river Pregel River
- What was the problem?
  - Can you cross every single bridge (7 bridges) once and ONLY once?
- What did Leonhard do?
  - Considered each island as a node and each bridge as an edge
  - 4 vertices & 7 edges



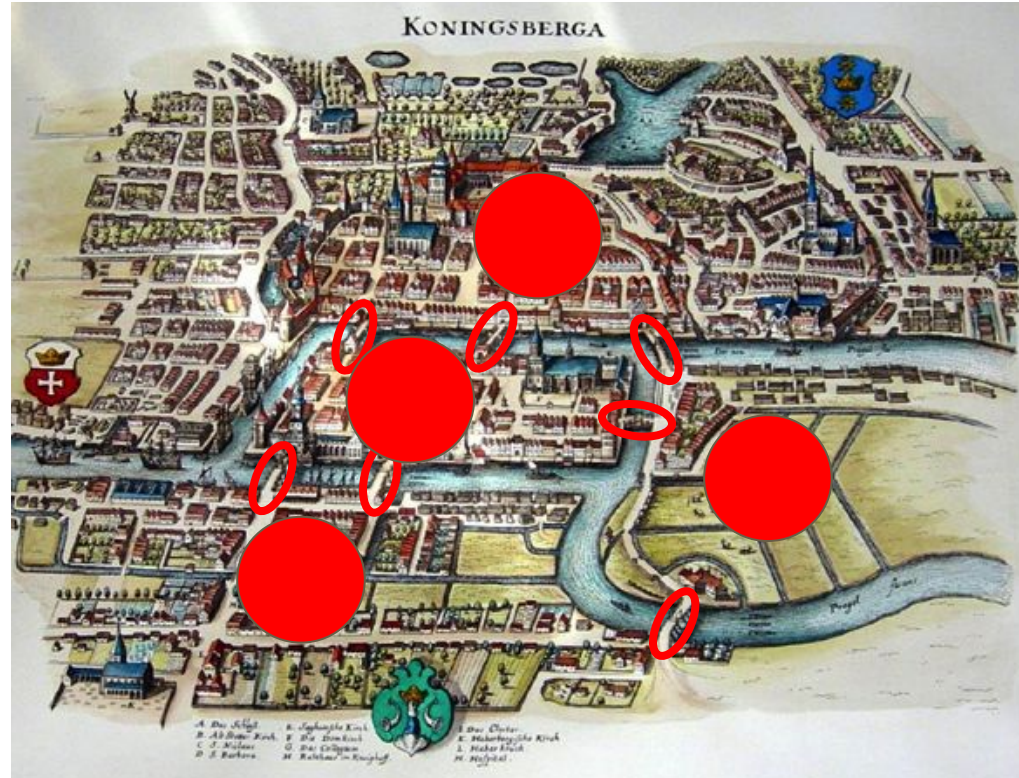
# The Origins of Graph Theory (Seven Bridges of Königsberg)

- Leonhard Euler in 1736
- Königsberg was a city in Germany that is now Kaliningrad, Russia built around a river Pregel River
- What was the problem?
  - Can you cross every single bridge (7 bridges) once and ONLY once?
- What did Leonhard do?
  - Considered each island as a node and each bridge as an edge
  - 4 vertices & 7 edges



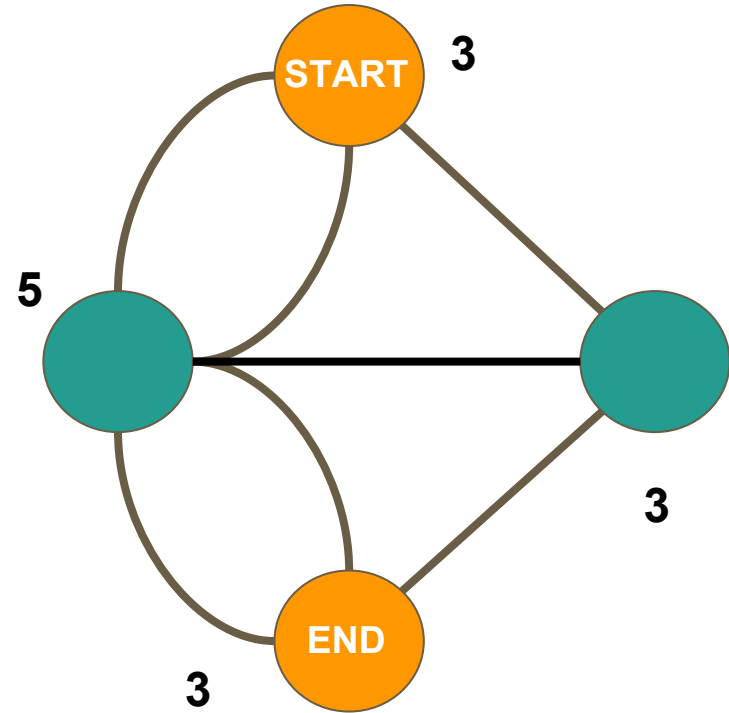
# The Origins of Graph Theory (Seven Bridges of Königsberg)

- Leonhard Euler in 1736
- Königsberg was a city in Germany that is now Kaliningrad, Russia built around a river Pregel River
- What was the problem?
  - Can you cross every single bridge (7 bridges) once and ONLY once?
- What did Leonhard do?
  - Considered each island as a node and each bridge as an edge
  - 4 vertices & 7 edges



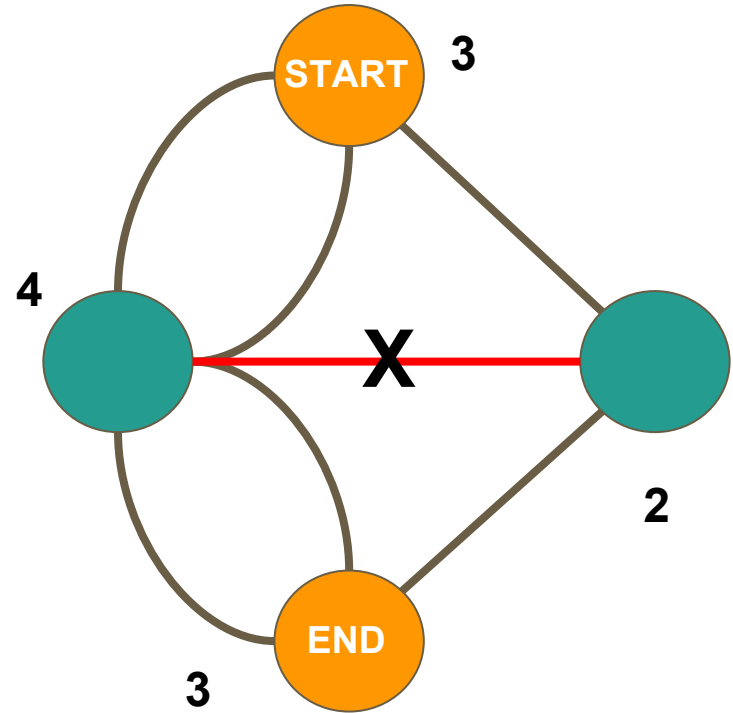
# The Origins of Graph Theory (Seven Bridges of Königsberg)

- Euler realized that there was no way to cross each bridge only once
- Eulerian graph/walk was born
  - Vertices that are not a start or end vertex must have even degree
  - We can have a start vertex which is different than the end vertex
    - We can have only odd degree at most twice (Start & End)
  - Number of odd degree vertices is 0 or 2
- This was one of the first examples of what a graph was and how it was used



# The Origins of Graph Theory (Seven Bridges of Königsberg)

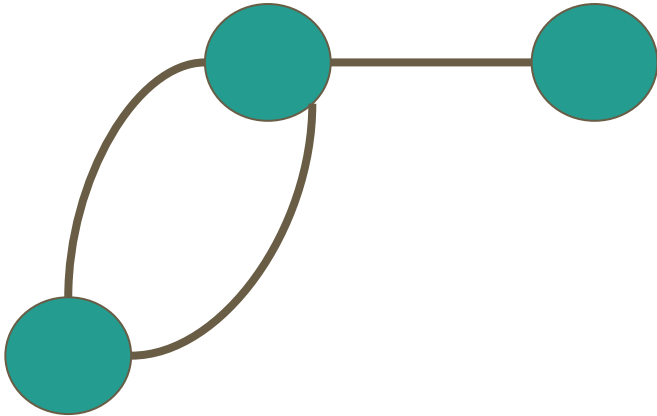
- Euler realized that there was no way to cross each bridge only once
- Eulerian graph/walk was born
  - Vertices that are not a start or end vertex must have even degree
  - We can have a start vertex which is different than the end vertex
    - We can have only odd degree at most twice (Start & End)
  - Number of odd degree vertices is 0 or 2
- This was one of the first examples of what a graph was and how it was used



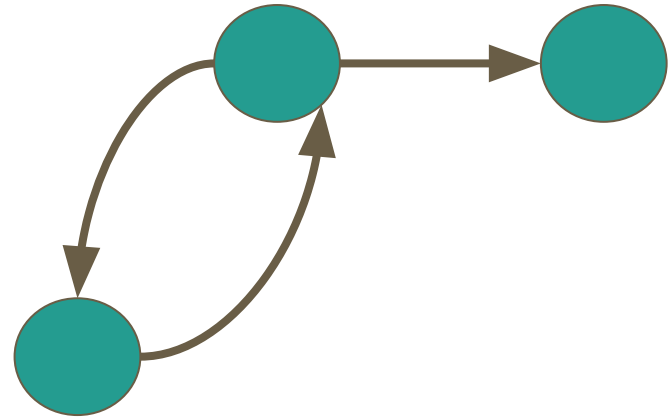


# A Few Graph Types

- **Undirected graph:** a graph that doesn't have a particular direction for edges.

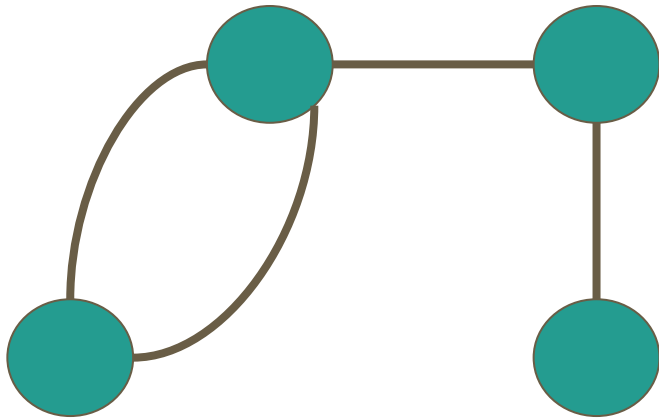


- **Directed graph:** a graph in which edges have a particular direction.

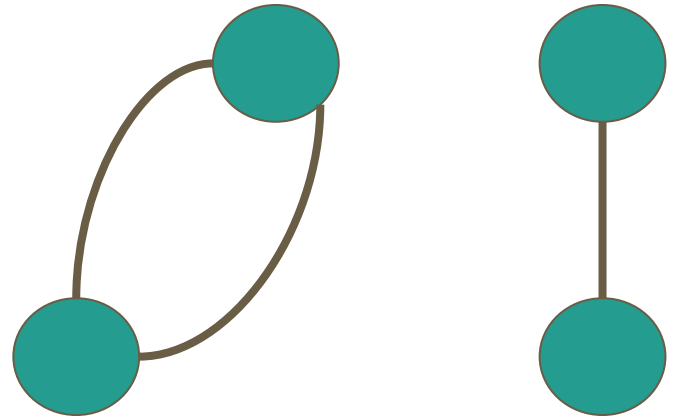


# A Few Graph Types

- **Connected graph:** A graph where there is no unreachable vertex. There must be a path between every pair of vertices.

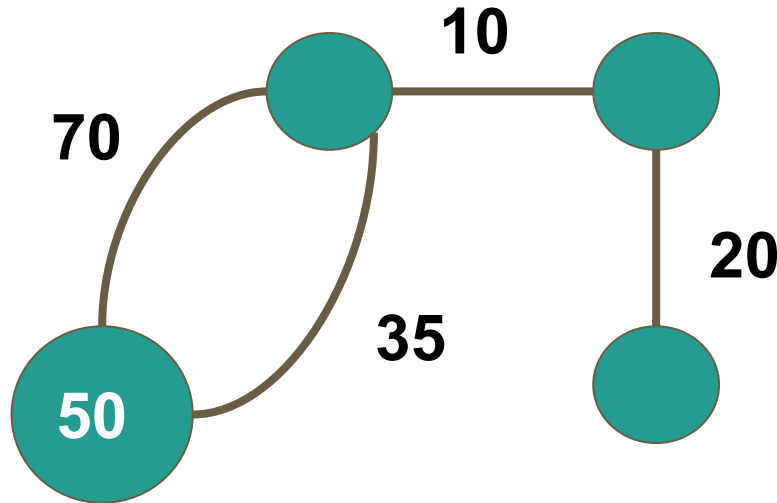


- **Disconnected graph:** A graph where there are unreachable vertices. There is not a path between every pair of vertices.



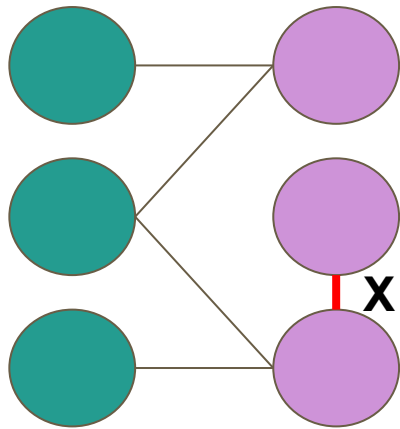
# A Few Graph Types

- **Weighted Graph:** A graph whose vertices or edges have been assigned weights.



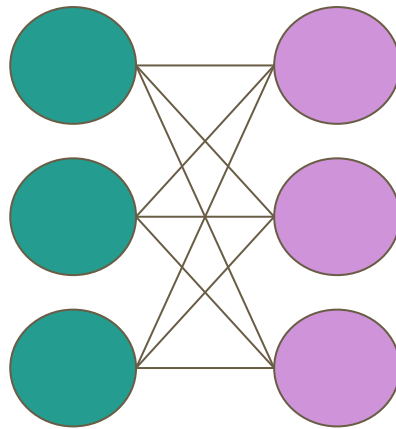
# A Few Graph Types

**Bipartite Graphs:** Type of graph whose vertex sets can be partitioned in two sets

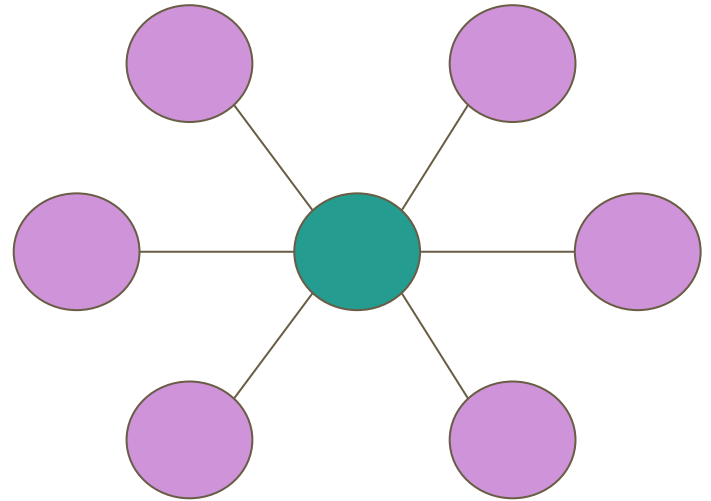


V1

V2



COMPLETE



STAR

# Walk, Path and Cycle

## Open Walk:

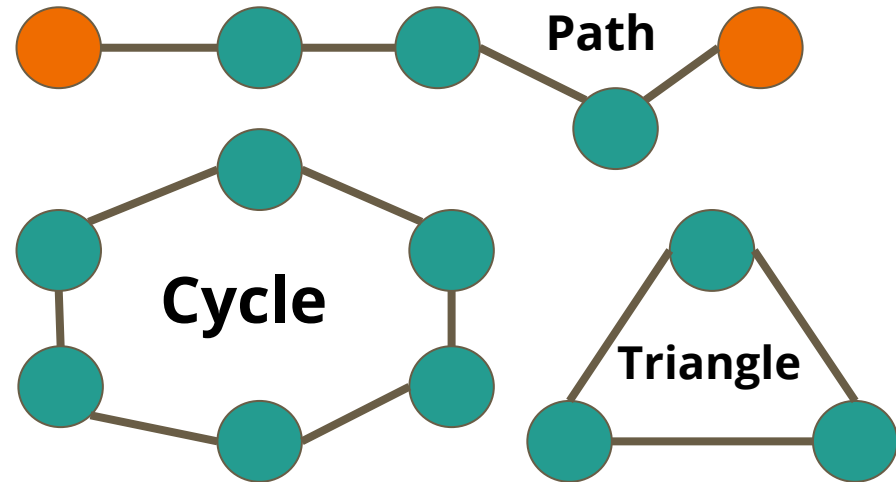
- When no vertex appears more than once is called a **path**
- A path does not intersect itself unless it is a closed walk

## Closed Walk:

- Initial and final vertex appear more than once in the walk is called a circuit. A Circuit is also called a **Cycle**
- In a cycle, vertices can be arranged in a cyclic sequence

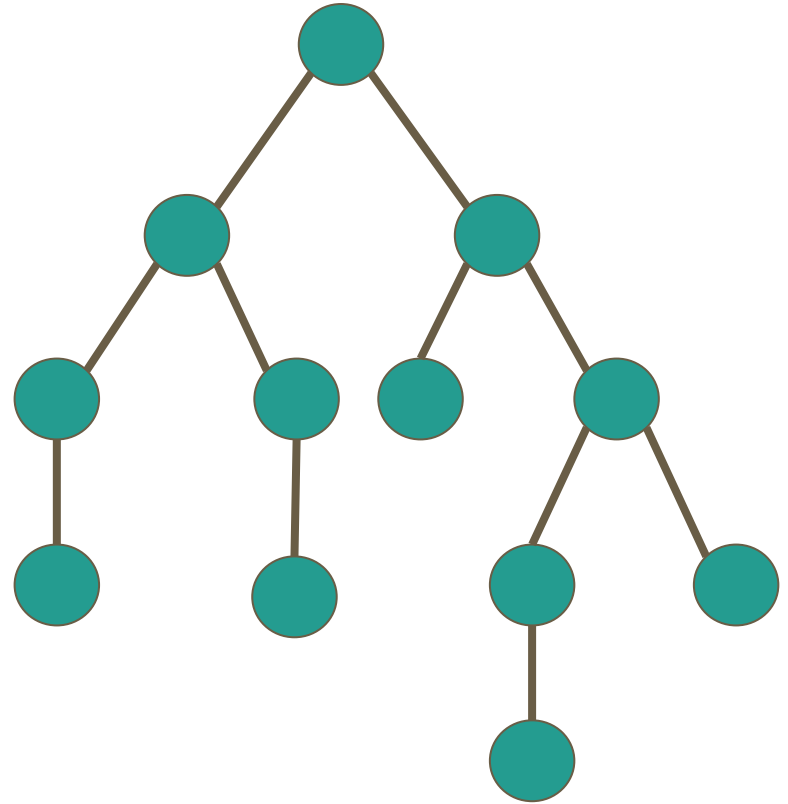
## Path:

- Start & End vertices have degree ONE
- Intermediate vertices have degree 2
- Vertices arranged in a sequence



# Tree Graphs

- Limited versions of a graph
- Directed Acyclic Graphs (DAG)
- Graph traversal applies to trees too
- Traversing to a tree is a little different than traversing a graph
  - We usually check or update the nodes
- Walking through a tree then involves not passing through the same node twice
- Order of the tree traversal helps to classify the different traversal algorithms
- Order matters!! We either go deep or go wide when traversing a tree



# Graph Analytics

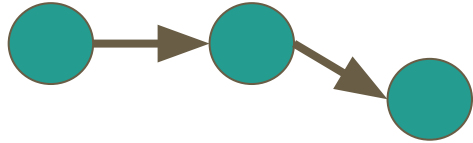
What are graph analytics?

Is it a graph query?

# Graph Analytics (3 Levels)

## Graph Queries

$(V1)-[E1]->(V2); (V2)-[E2]->(V3)$

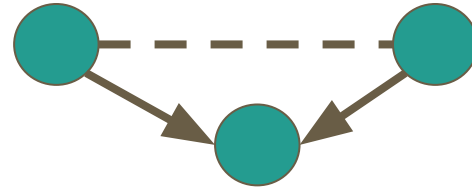


## Graph Analytics:

Via label propagation, identify reasonable partitions within my graph (statistical approach)

## Graph Algorithms:

Apply graph reasoning





# Graph Algorithms

**Pathfinding:** Finds the shortest path or evaluate route availability

- Breadth-First & Depth-First Search
- Shortest Path

**Centrality:** Determines the importance of distinct nodes in the network

- Page Rank
- In-Degree & Out-Degree

**Community-Detection:**

Evaluates how a group is clustered or partitioned

- Connected Components
- Strongly Connected
- Label Propagation

# Pathfinding

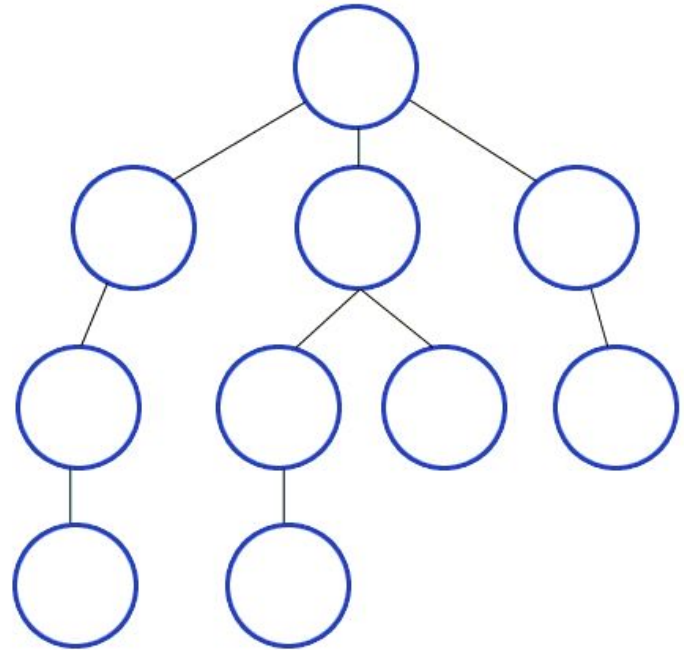
**Breadth-First Search**

**Depth-First Search**

**Shortest Path**

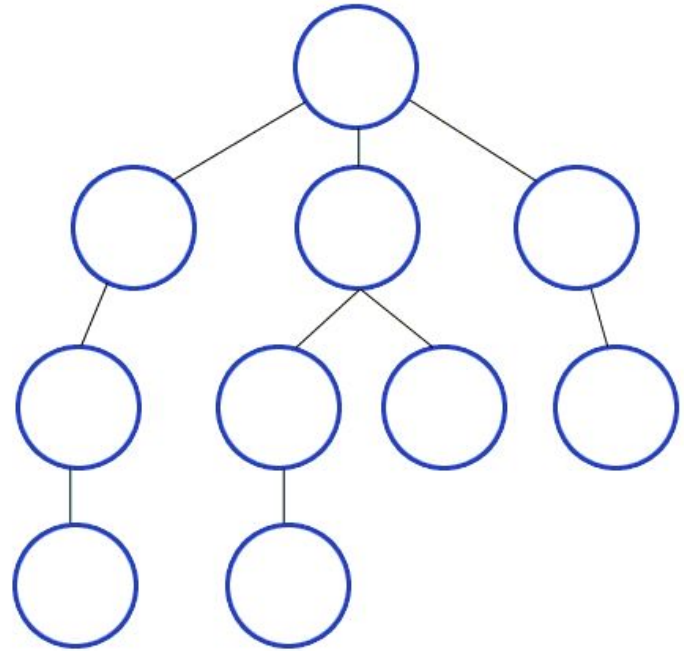
# Depth-First Search

- Go deep into one node before asking all the childrens (neighbors)
- Deep traversal into a data structure
- Difference between graph and tree traversal is deciding where to start searching!
- Stack process:
  - 1,2
  - 1,2,3
  - 1,2,3,4
  - 1,2,3
  - 1,2
  - 1,5 -> 1,5,6 -> 1,5,6,7



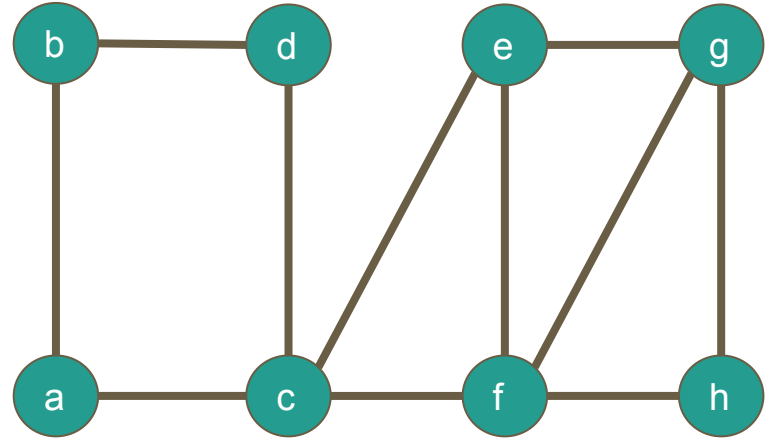
# Breadth-First Search

- Searching through a tree or data structure level by level out
- Broad traversal into a data structure
- Wider before going deep
- Difference between graph and tree traversal is deciding where to start searching!
- Queue Process
  - 1,2,3,4
  - **1**,2,3,4,5,6,7,8
  - **1,2,3,4**,5,6,7,8,9,10



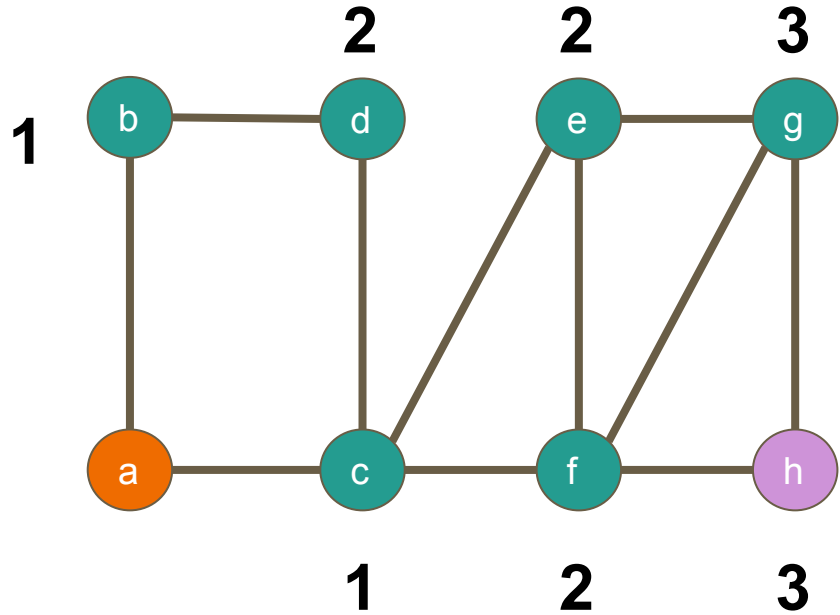
# Breadth-First Search

- Searching through a tree or data structure level by level out
- Broad traversal into a data structure
- Wider before going deep
- Level order in a binary tree
- Difference between graph and tree traversal is deciding where to start searching!
- Queue Process
  - 1,2,3,4
  - **1**,2,3,4,5,6,7,8
  - **1,2,3,4**,5,6,7,8,9,10



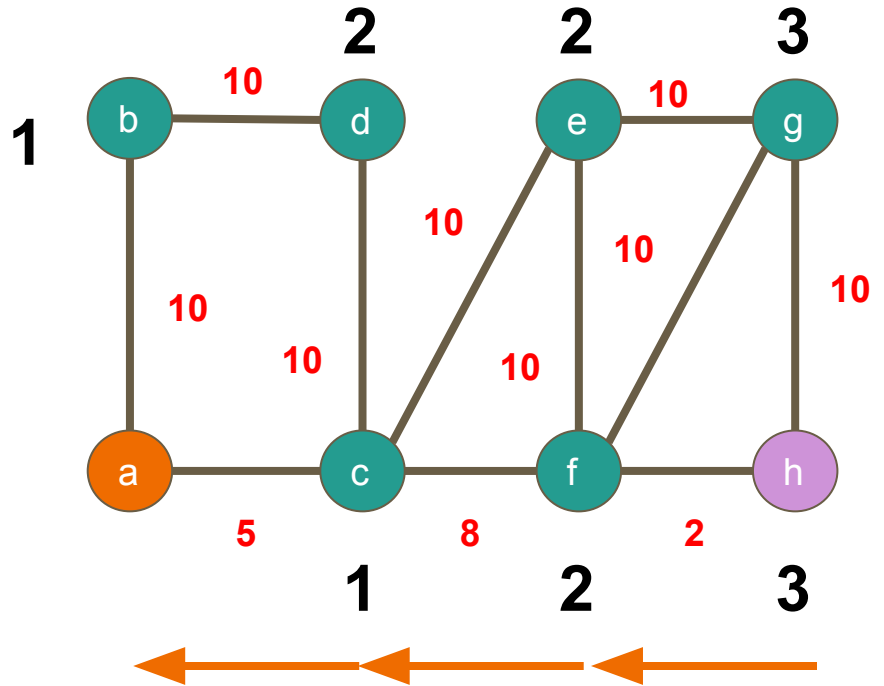
# Breadth-First Search

- Searching through a tree or data structure level by level out
- Broad traversal into a data structure
- Wider before going deep
- Level order in a binary tree
- Difference between graph and tree traversal is deciding where to start searching!
- Queue Process
  - 1,2,3,4
  - **1**,2,3,4,5,6,7,8
  - **1,2,3,4**,5,6,7,8,9,10



# Shortest Path

- BFS algorithms will keep track of every single node's "parent" and the nodes that come before it.
- We can then use the pointers of the path that we took in order to determine a shortest path in the graph.
- Dijkstra's algorithm uses weights
  - GPS (Distance)
- Single source shortest path
  - One vertex to the other vertices



# Adversarial Graph Application - BloodHound

- Adversaries used to rely on extremely tedious manual “derivative local admin” methodology – used to take days or weeks to find paths to domain admin.
- PowerView enabled easy local admin, user session, and security group enumeration, all with just domain authenticated access (no local admin needed!)
- BloodHound uses a directed graph using the above information to find privilege escalation attack paths from any node to any other node (e.g.: attack paths to the Domain Admins group)
- BloodHound compresses the time needed to escalate privileges in a domain from days or weeks to minutes or hours.
- See our DEFCON 24 video for more information:

<https://www.youtube.com/watch?v=wP8ZCczC1OU>





# Defensive Graph Application - BloodHound

- Adversaries use BloodHound to find not just one, but ALL attack paths in a domain that rely on stolen credentials, AD object control, and GPO control.
- Defenders use BloodHound to find the same attack paths and shut them down before an attacker can.
- BloodHound also enables easy, graphical local admin auditing and nested security group memberships.
- Several blue teams run BloodHound collection and analyze the results on a monthly basis to see how attack paths are evolving over time in their environments.
- See our blog post for more info:  
<https://posts.specterops.io/introducing-the-adversary-resilience-methodology-part-one-e38e06ffd604>

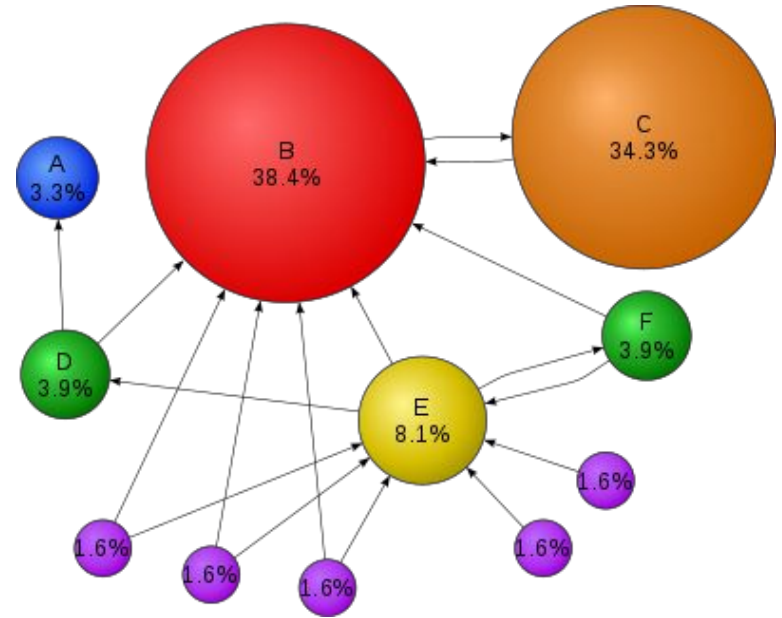


# Centrality

- Page Rank
- In-Degree & Out-Degree

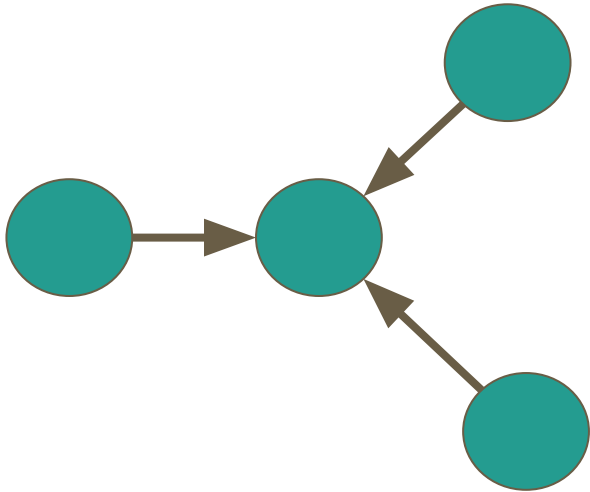
# Page Rank

- Larry Page, cofounder of Google, created the algorithm to rank websites in their search engine results.
- If you have links from other sites that have high page rank, you get a high page rank (Recursive)
- It counts links to a page to determine a rough estimate of how important the website is.
- More important websites are likely to receive more links from other websites.

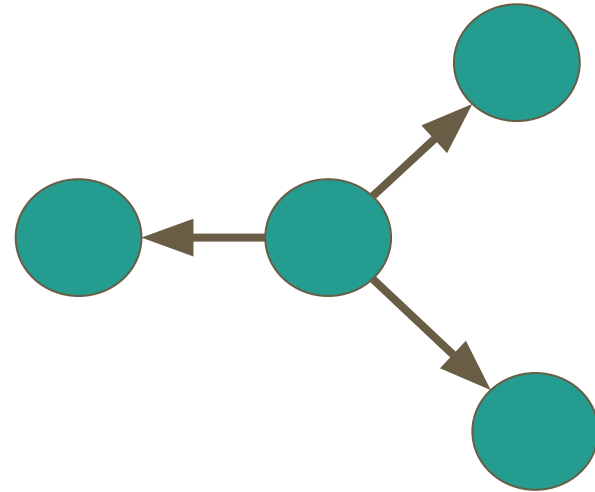


# In-Degree & Out-Degree

In- Degree



Out-Degree:

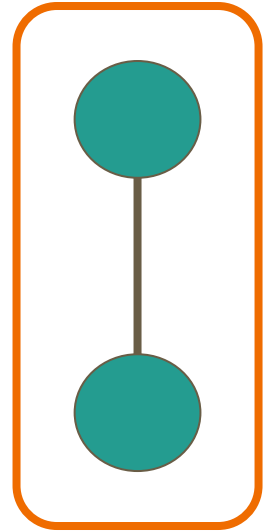
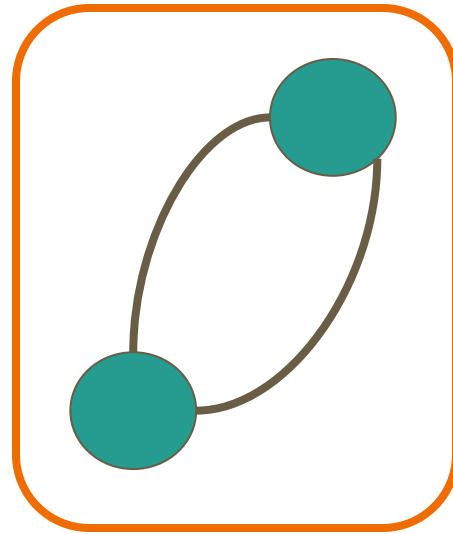


# Community-Detection

- **Connected Components**
- **Strongly Connected**
- **Label Propagation**

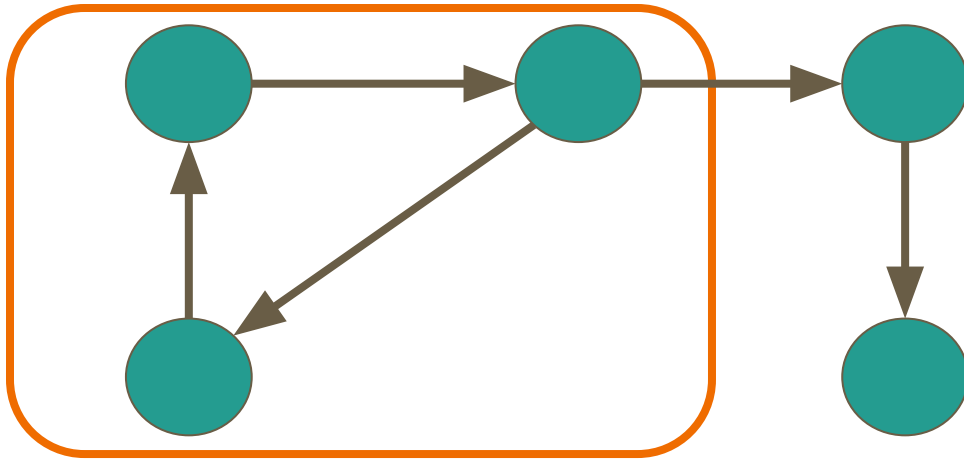
# Connected Components

- I want to partition my graph into parts that are connected
- Identifiers get applied to the vertices of the same group
- A connected component defines an **(undirected) subgraph** that has connections to itself but does not connect to the greater graph



# Strongly Connected

- It takes directionality into account
- It is a subgraph that has paths between all pairs of vertices inside it.



# Label Propagation

- Not a label defined as a property of the vertex
  - It is more like a tag
- At the end similar vertices will have the same tags
- semi-supervised machine learning algorithm that assigns labels to previously unlabeled data points
- Each node in the network is initially assigned to its own community. At every superstep, nodes send their community affiliation to all neighbors and update their state to the mode community affiliation of incoming messages.



**OK, let just graph  
EVERYTHING... right?**

# HELK

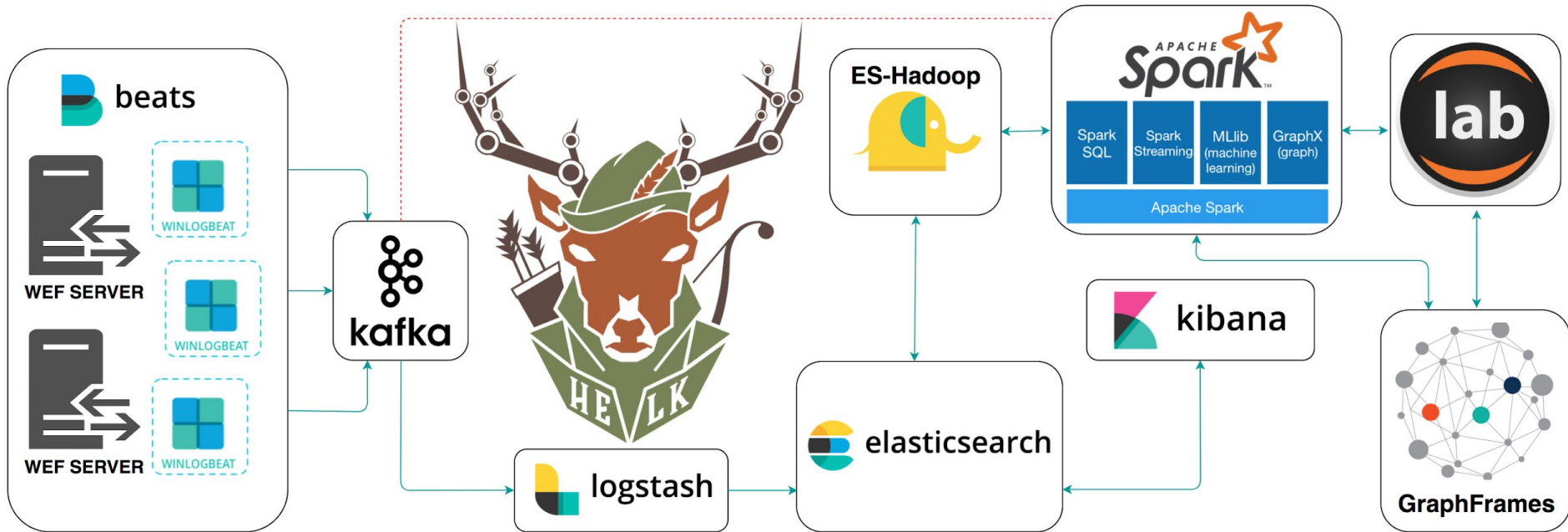
An open source ELK with  
Advanced Analytics Capabilities

<https://github.com/Cyb3rWard0g/HELK>

[ ALPHA ]



# Architecture

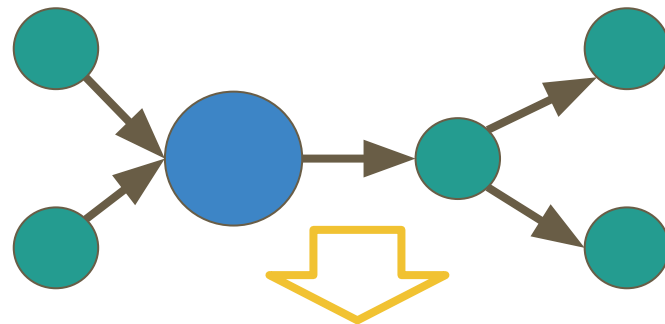


# Main Features Definitions

- **Kafka:** A distributed publish-subscribe messaging system that is designed to be fast, scalable, fault-tolerant, and durable.
- **Elasticsearch:** A highly scalable open-source full-text search and analytics engine.
- **Logstash:** A data collection engine with real-time pipelining capabilities.
- **Kibana:** An open source analytics and visualization platform designed to work with Elasticsearch.
- **ES-Hadoop:** An open-source, stand-alone, self-contained, small library that allows Spark to interact with Elasticsearch.
- **Spark:** A fast and general-purpose cluster computing system. It provides high-level APIs in Java, Scala, Python and R, and an optimized engine that supports general execution graphs.
- **GraphFrames:** A package for Apache Spark which provides DataFrame-based Graphs.
- **Jupyter Notebook:** An open-source web application that allows you to create and share documents that contain live code, equations, visualizations and narrative text.

# GraphFrames

- Currently an Apache Spark Package
- Graph library based on DataFrames
- Python, Java & Scala APIs
- All algorithms from GraphX
- Queries via SparkSQL APIs & DataFrames
- Cypher-Like graph queries
- Supports DataFrame data sources, allowing writing and reading graphs using many formats like Parquet, JSON, and CSV



**GraphFrames API**

**Spark**  **SQL**



**ES-Hadoop**



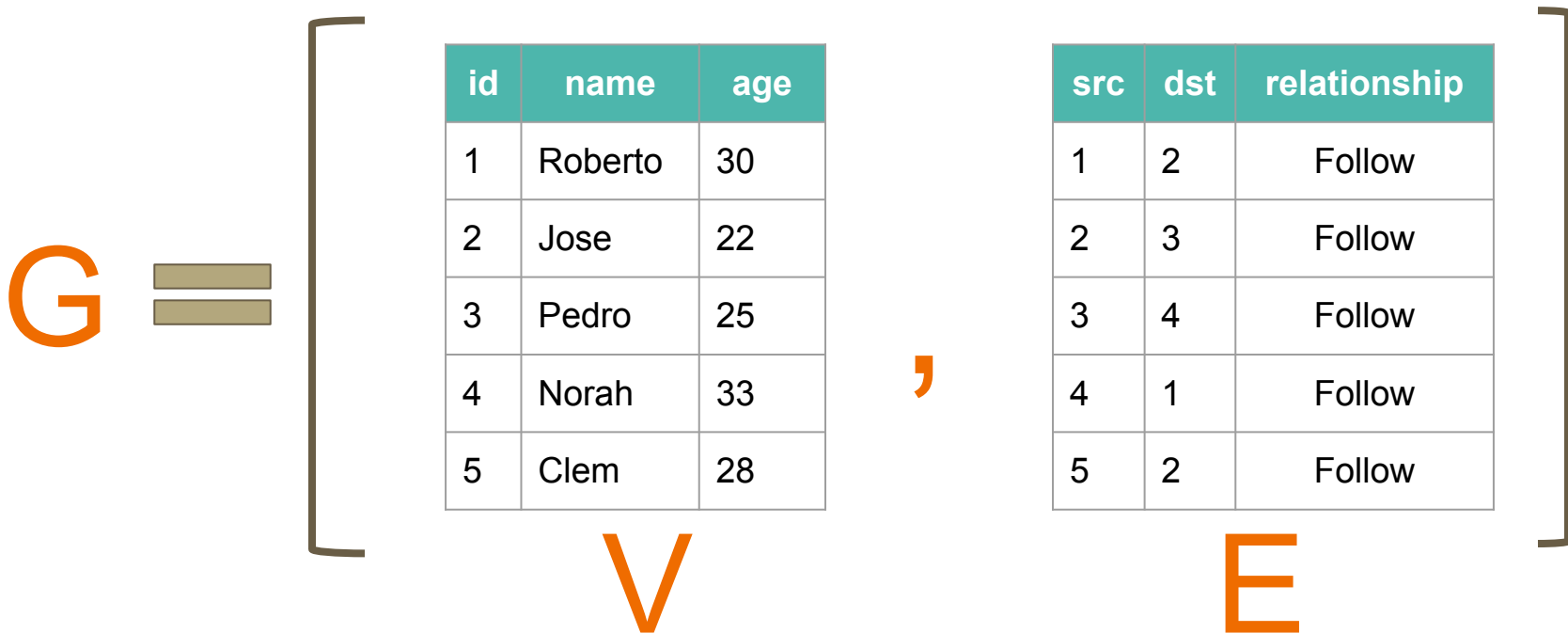
**elasticsearch**

## How does GraphFrames compare to graph databases?

- Spark is not a database
- Spark is a distributed computation engine
- Spark does not store data long-term
- Spark is capable to build a graph computation via Graphframes

# How do you build a Graph?

A graph is composed by two dataframes (Vertices & Edges)



# Graph Queries: MOTIF Finding

- Motifs are a way of expressing structural patterns in a graph
- We are querying for patterns in the data instead of actual data.
- vertex connects to another vertex:

**(a)-[ab]->(b)**

- The letters inside of parenthesis or brackets do not signify values but signify what the columns should be named in the resulting DataFrame

**(a)-[]->(b)**



# Graph Algorithms

**Pathfinding:** Finds the shortest path or evaluate route availability

- Breadth-First & Depth-First Search
- Shortest Path

**Centrality:** Determines the importance of distinct nodes in the network

- Page Rank
- In-Degree & Out-Degree

**Community-Detection:**

Evaluates how a group is clustered or partitioned

- Strongly Connected
- Label Propagation
- Triangle Counting

# HELK - Show me now!

- Red Team Training Data set
- Data Sources: Sysmon EID 3 (Network Connections)
- Explore the data!

# HELK - Build A Graph

- Elasticsearch OR CSV
- Sysmon Index
  - Event ID 3 : Network Connection

# Connect to Elasticsearch -> Read Sysmon Index

```
%python
```

```
spark = SparkSession.builder \  
  .appName("HELK") \  
  .config("es.read.field.as.array.include", "tags") \  
  .config("es.nodes", "helk-elasticsearch:9200") \  
  .getOrCreate()  
  
df = spark.read.format("org.elasticsearch.spark.sql").\  
load("logs-endpoint-winevent-sysmon-*/doc")
```

# Select Network Events and specific field names

```
%python
```

```
df = df.filter(df.event_id == 3)
```

```
df = df.select("process_name","host_name",\  
"src_ip","src_port_number",\  
"Dst_host","dst_ip","dst_port_number",\  
"process_guid","user_name","action")
```

## Read from CSV

```
%python
```

```
rto_df = spark_graph\  
    .read\  
    .option("inferSchema", "true")\  
    .option("header", "true")\  
    .csv("/tmp/RTO0518_EID3_PART0.csv")
```

# Define Vertices

```
%python
```

```
vertices = rto_df.withColumn("id", rto_df.ip_src)\  
  .filter(rto_df.ip_src != "10.10.10.255")\  
  .filter(rto_df.user_name != "NETWORK SERVICE")\  
  .select("id", "user_name", "process_name",  
"port_dst_number")\  
  .distinct()
```

# Define Edges

```
%python
```

```
edges = rto_df.selectExpr\  
    ("ip_src as src","ip_dst as dst")
```



# HELK - Graph Queries

# MOTIF Finding

```
%python
```

```
G.find("(a)-[]->(b);(b)-[]->(a)")
```

# HELK - Graph Algorithms

- **Centrality: In-Out Degrees & PageRank**
- **Community: Connected Components**

# PageRank Algorithm

```
%python
ranks = subgraph.pageRank(resetProbability=0.15,
maxIter=10)
ranks.vertices\
  .orderBy(desc("pagerank"))\
  .select("id", "pagerank")\
  .show(10)
```

# IN & OUT Degrees

Using the following query, you can find interesting people in the social network that might have more influence than others.

```
in_degree = subgraph.inDegrees
```

```
in_degree.orderBy(desc("inDegree")).show(5, False)
```

```
out_degree = subgraph.outDegrees
```

```
out_degree.orderBy(desc("outDegree")).show(5, False)
```

# Connected Components

- Computes the connected component membership of each vertex and returns a graph with each vertex assigned a component ID
- **Applications:**
  - Clustering
  - Anomaly Detection
  - Fraud

```
%python
```

```
spark.sparkContext.setCheckpointDir("/tmp/checkpoints")
```

```
SubGraph = Graph(Vertices,  
Edges.sample(False, 0.2))
```

```
cc = SubGraph.connectedComponents()
```

# HELK - Future

- Cyphe for Apache Spark (CAPS)
- Zeppelin Notebook
- Kibana Vega Visualizations

# Cypher for Apache Spark

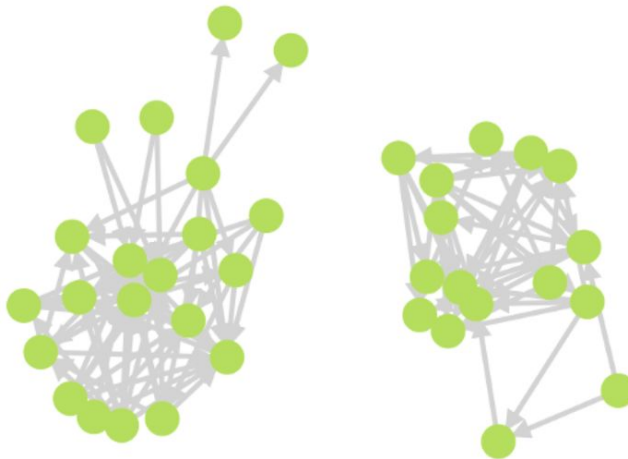
```
val CITYFRIENDS_NA = SN_NA.cypher(
  """
  | MATCH (a:Person)-[:IS_LOCATED_IN]->(city:City)<-[:IS_LOCATED_IN]-(b:Person),
  |       (a)-[:KNOWS*1..2]->(b)
  | WHERE city.name = "New_York" OR city.name = "San_Francisco"
  | RETURN GRAPH result OF (a)-[r:SIMILAR_CIRCLE]->(b)
  """ .stripMargin).graphs("result").cache
```

CITYFRIENDS\_NA.asZeppelinGraph




Nodes **37**: Person

Relationships **99**: SIMILAR\_CIRCLE






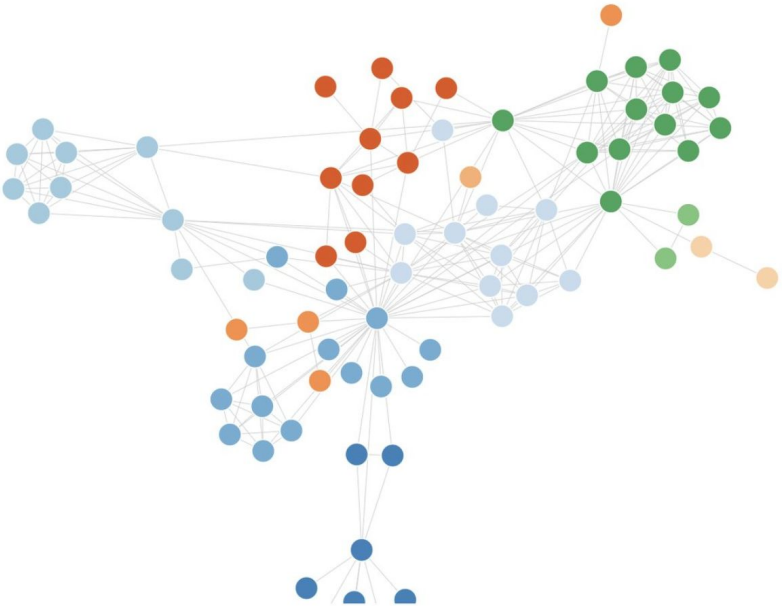
# Kibana Vega Viz

Visualize / New Visualization (unsaved) Save Share Refresh  Auto-re

Search... (e.g. status:200 AND extension:PHP)

 This visualization is marked as experimental.

```
68 |
69 | "scales": [
70 |   {
71 |     "name": "color",
72 |     "type": "ordinal",
73 |     "range": {"scheme": "category20c"}
74 |   }
75 | ],
76 |
77 | "marks": [
78 |   {
79 |     "name": "nodes",
80 |     "type": "symbol",
81 |     "zindex": 1,
82 |
83 |     "from": {"data": "node-data"},
84 |     "on": [
85 |       {
86 |         "trigger": "fix",
87 |         "modify": "node",
88 |         "values": "fix === 1 ? {fx:node.x, fy:
                        :node.y} : {fx:x(), fy:y()}"
89 |       },
90 |       {
91 |         "trigger": "!fix",
92 |         "modify": "node", "values": "{fx:
                        null, fy: null}"
93 |       }
94 |     ],
95 |   }
96 | "encode": {
```



The visualization displays a network graph with nodes and edges. The nodes are colored based on their category, using a categorical color scheme (category20c). The nodes are arranged in a complex, interconnected structure, with some clusters and some isolated nodes. The edges are thin lines connecting the nodes, representing relationships between them. The overall layout is a dense network of connections.

# Information

<https://github.com/Cyb3rWard0g>

@Cyb3rWard0g

# Resources

- <https://posts.specterops.io/welcome-to-helk-enabling-advanced-analytics-capabilities-f0805d0bb3e8>
- <https://www.youtube.com/watch?v=wP8ZCczC1OU>
- <https://posts.specterops.io/introducing-the-adversary-resilience-methodology-part-one-e38e06ffd604>
- <https://graphframes.github.io/user-guide.html#graph-algorithms>
- <https://medium.com/basecs/leaf-it-up-to-binary-trees-11001aaf746d>
- <https://neo4j.com/blog/graph-algorithms-neo4j-15-different-graph-algorithms-and-what-they-do/?platform=hootsuite>
- <https://neo4j.com/blog/top-13-resources-graph-theory-algorithms/>
- <https://neo4j.com/graph-analytics/?ref=blog>
- <https://neo4j.com/blog/graph-algorithms-neo4j-connections-drive-discoveries/>
- <https://medium.com/@KerrySheldon/breadth-first-search-in-apache-spark-d274403494ca>
- <https://www.forbes.com/sites/danwoods/2018/04/30/improve-your-graph-iq-what-are-graph-queries-graph-algorithms-and-graph-analytics/#3c63c3731961>
- [https://neo4j.com/blog/cypher-for-apache-spark/?utm\\_content=buffer44adf&utm\\_medium=social&utm\\_source=twitter.com&utm\\_campaign=buffer](https://neo4j.com/blog/cypher-for-apache-spark/?utm_content=buffer44adf&utm_medium=social&utm_source=twitter.com&utm_campaign=buffer)
- <https://neo4j.com/blog/top-13-resources-graph-theory-algorithms/>