GRAPH + AI SUMMIT

organized by TigerGraph

Graph Modeling for Blockchain Forensics

(Graph For A Better Tokenomics)

Krishna Sankar @ksankar

Applicable to Non-Reciprocal, Sequenced, Temporal Networks

GRAPH FOR ALL
MILLION DOLLAR CHALLENGE
TigerGraph

Krishna Sankar
@ksankar

Graph For A Better Tokenomics a.k.a. Token Economy

Problem Statement

- Immutable, Distributed, Consenus-driven & *Secure* DLT & Blockchain stack is essential for Crypto currency & Web 3.0
  - *While the other properties are well implemented, security is still a challenge*
- Specifically, this talk addresses the detection of Fraud Rings (e.g., Money Laundering), Camouflaged Fraud (e.g., Fake Product Reviews) and other similar activities

Hypothesis

- Graph representation & algorithms are well suited to analyze payment networks & extract fraud rings of *arbitrary complexity*

Approach

- Tackle one of the complex fraud pattern in a scalable, extensible way

Agenda

1. Problem Statement
2. Demo
3. A few fantastic techniques (and where to find them!)
4. TigerGraph and the hard & (yet) unsolved challenges (in payment networks)

Non-Reciprocal, Sequenced, Temporal Networks

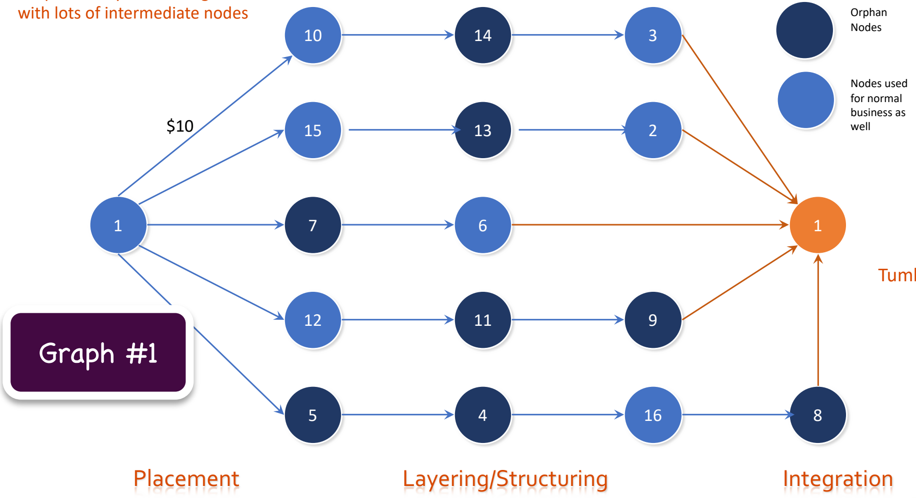
- Pragmas
  1. Relationships need not be symmetric or reciprocal
  2. There is a sequence to the edges i.e. Sequenced Edges
  3. The edges have a temporal, monotonic component i.e. Temporal Edges
- Examples
  1. Payment Networks Forensics
  2. Fake Reviews
  3. Fake News
- Challenges
  1. Breaks some of the assumptions in traditional graph algorithms
  2. Traditional Graph Algorithms are not as widely applicable
  3. In a payment network, edges rule
     - Algorithmically, nodes (while important) have a lesser role
  4. In short, we need more specific concepts
- Differences from traditional networks (social et al)
  1. PageRank doesn't make sense
     - Just because amzn paid someone doesn't make them any more important
  2. Community detection or k-clique are not that relevant
  3. Connected component is not that informative either

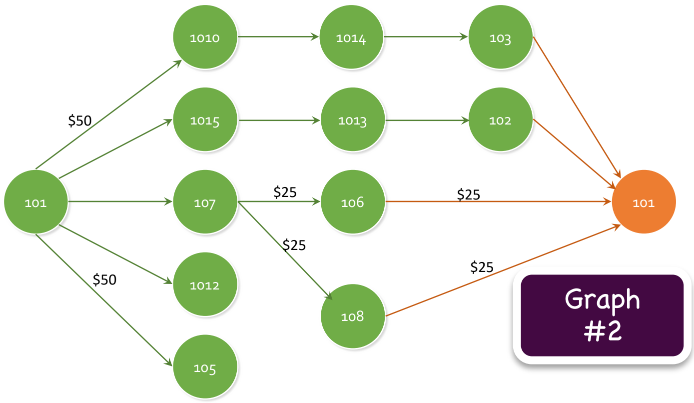
Let us take an example
Fan-In/Fan-Out Fraud Rings

- Money Laundering goes through 3 stages
  - Placement,
  - Layering and
  - Integration
- Layering is the most complex, where the money is piped through a complex web of strategic transactions – mixers, tumblers, long chain of crypto transactions et al., obscuring the money trail
- As we will see later, one essential aspect in understanding the problem is to draw representative graphs – *the happy path 1st* (see next slide)

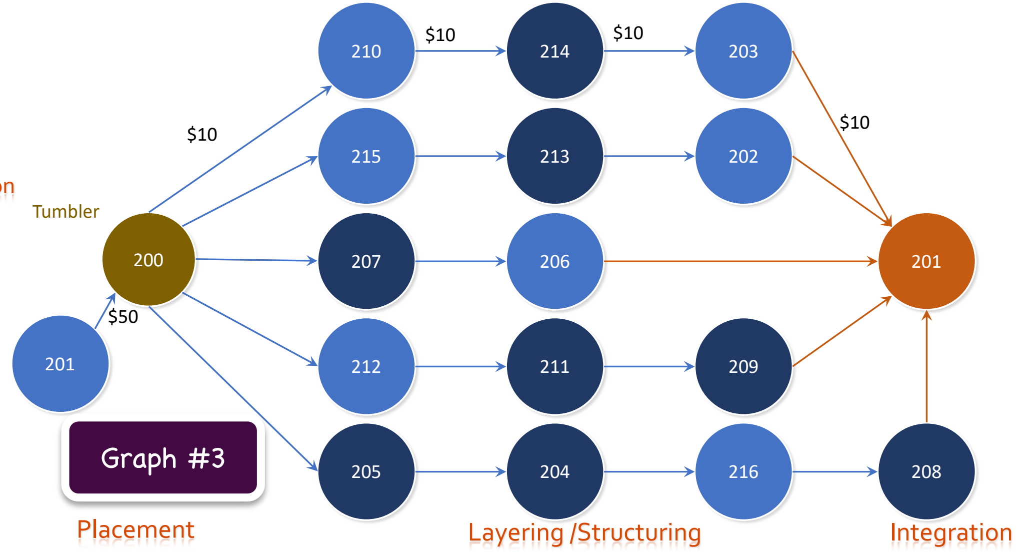
Money Laundering Graphs

Simple Money Laundering Scheme
with lots of intermediate nodes





Tumbler Network Money Laundering Scheme



From a traditional network perspective, there are multiple cycles; but from a payment network perspective the view is slightly different

Challenges

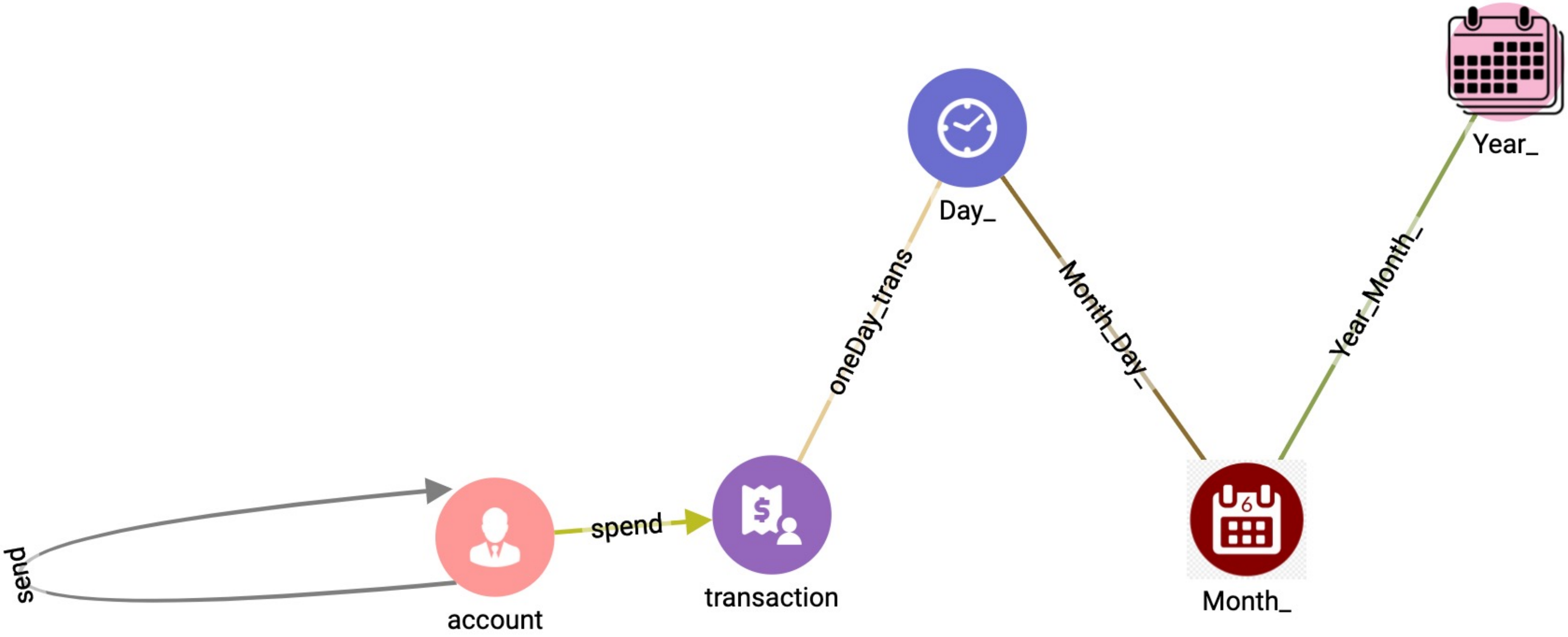
1. Non-Reciprocal, Sequenced, Temporal Networks
2. Payment network is very different from social or other networks
    - Temporal, monotonic time
    - Not symmetric
    - Other attributes (like amount) matters
    - Many of the concepts like connected components and page rank do not mean much
3. Money Laundering Layering is not just one cycle
    - But, multiple ordered cycles with a fraud actor at the helm
4. Detecting fraud ring is literally finding the proverbial needle in a haystack
    - Class imbalance, Long Fraud Chains spanning different crypto currency ledgers

Approach

1. Layered approach with well defined pipeline stages
    - *Overlay fraud rings progressively*
    - *Narrow down and organize the vertices as we progress*
    - *Always keep time sequencing in the processing*
2. Customize relevant gsql graph algorithms
    - *e.g., Rocha-Thatte Cycle Detection, but it is unordered*
    - *Also need to combine cycles and find the fraud actor at the helm*
3. Add runtime attributes to vertices
    - *That will help the processing downstream the pipeline*
4. Start with a simple schema and add more elements as required
5. Stay in TigerGraph as much as possible

Demo

Design Schema



Schema Information

Vertex information:

- Vertex: account
  - (PRIMARY ID) id: STRING
  - v_type: INT
- Vertex: transaction
  - (PRIMARY ID) id: STRING
  - amt: DOUBLE
  - date_time: DATETIME
  - from_id: UINT
  - to_id: UINT
- Vertex: Year_
  - (PRIMARY ID) id: STRING
- Vertex: Month_
  - (PRIMARY ID) id: STRING
- Vertex: Day_
  - (PRIMARY ID) id: STRING

Edge information:

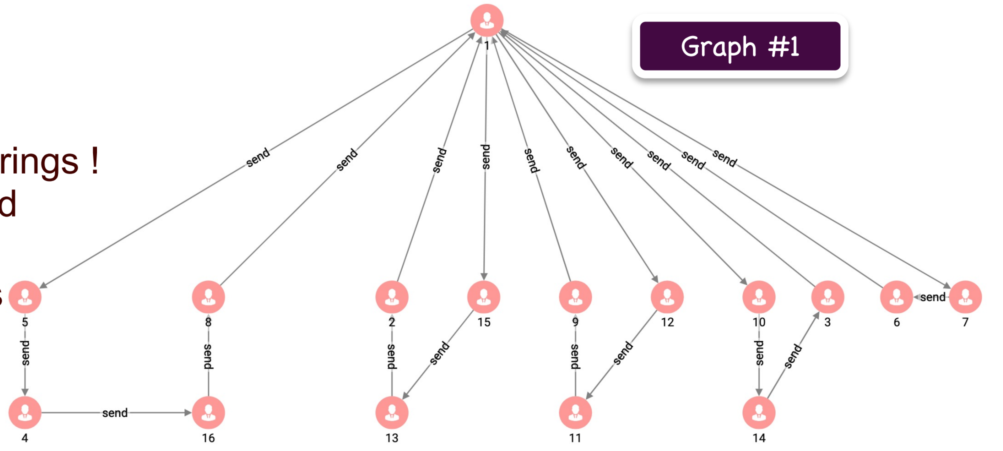
- Edge: send
  - Source: account
  - Target: account
  - amount: DOUBLE
  - date_time: DATETIME
- reverse edge: reverse_send
- Edge: spend
  - Source: account
  - Target: transaction
- reverse edge: reverse_spend
- Edge: Year_Month_
  - Source: Year_
  - Target: Month_
- Edge: Month_Day_
  - Source: Month_
  - Target: Day_

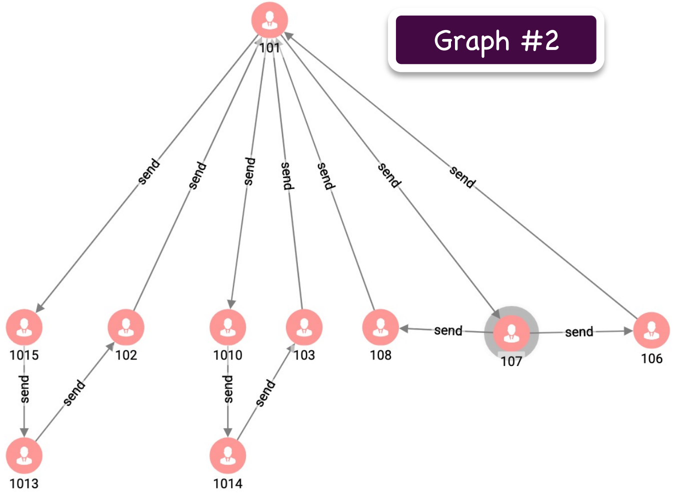
Edge: oneDay_trans

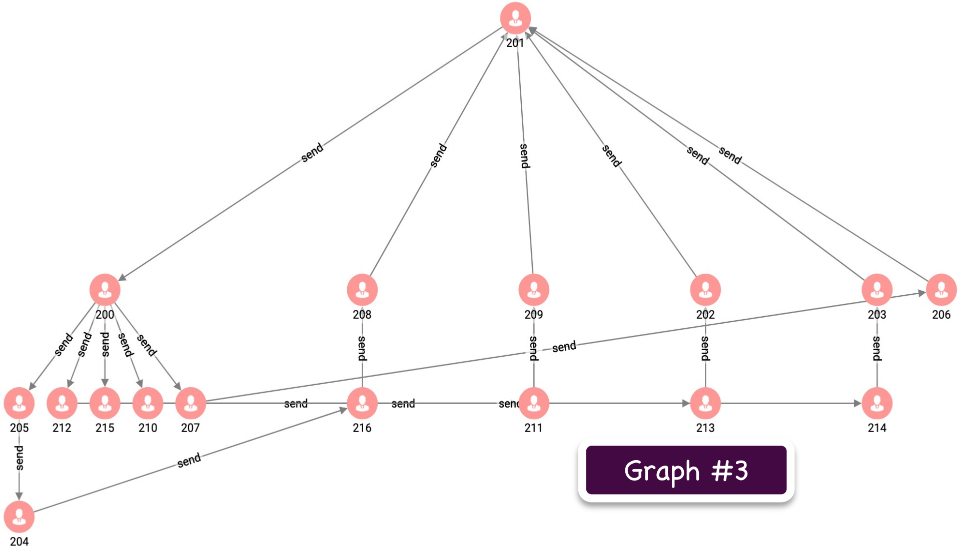
- Source: transaction
- Target: Day_
- date_time: DATETIME

Output

- TigerGraph finds all the 3 fraud rings !
- And, correctly identifies the fraud actors at the helm
- See next slides for comparisons

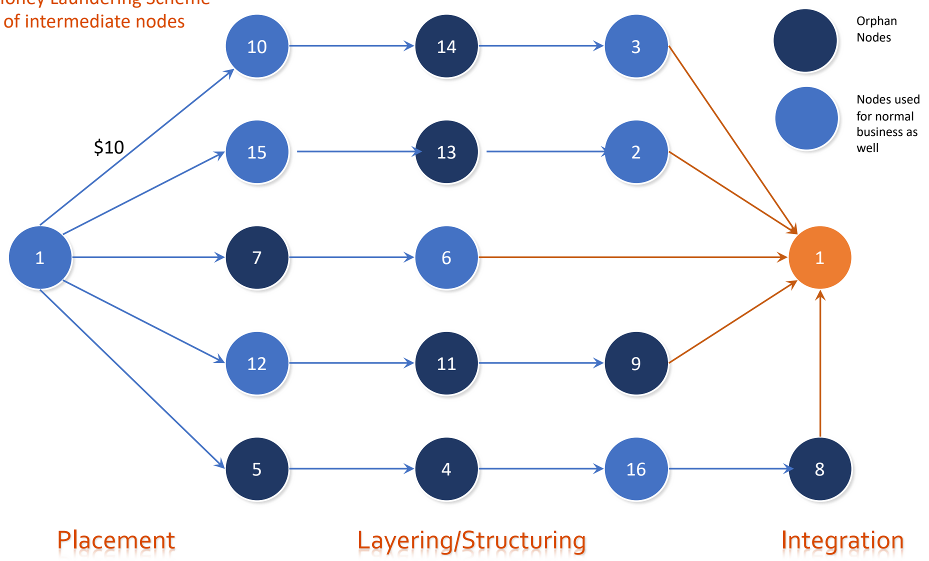


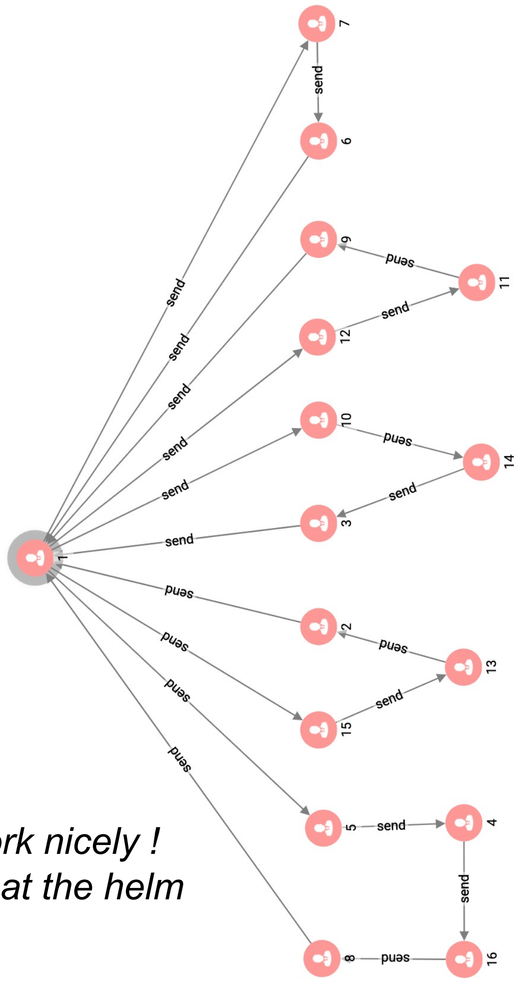




Graph #1

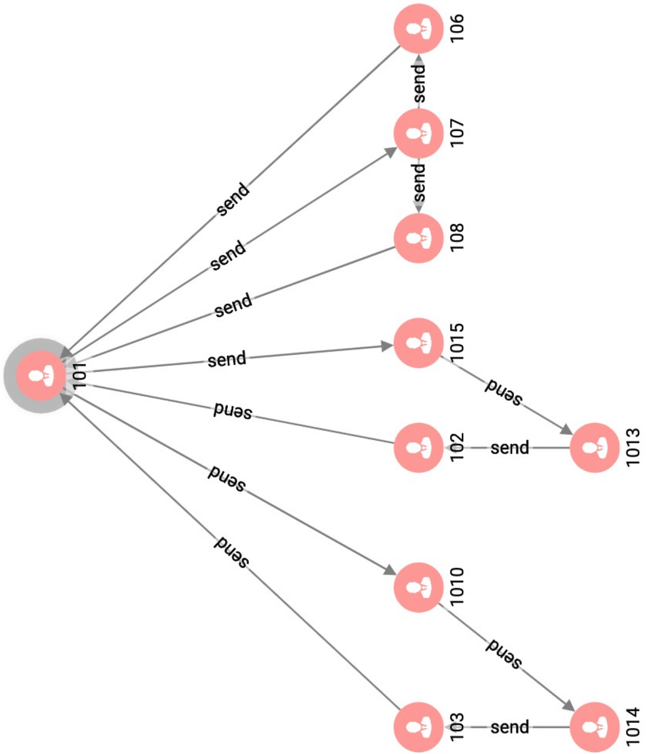
Simple Money Laundering Scheme
with lots of intermediate nodes





- TigerGraph does well
- Finds the money laundering network nicely !
- Correctly identifies the fraud actor at the helm

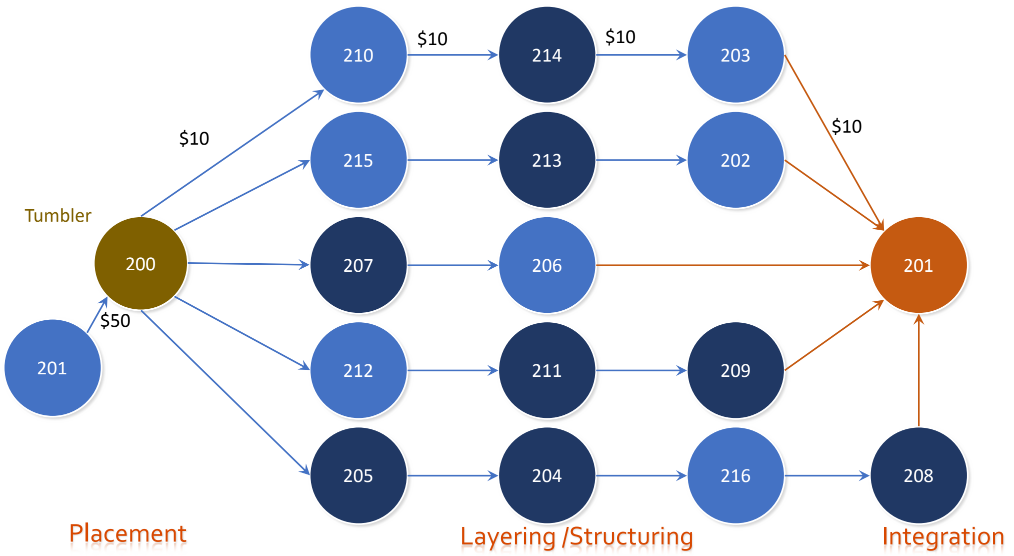
Graph #2

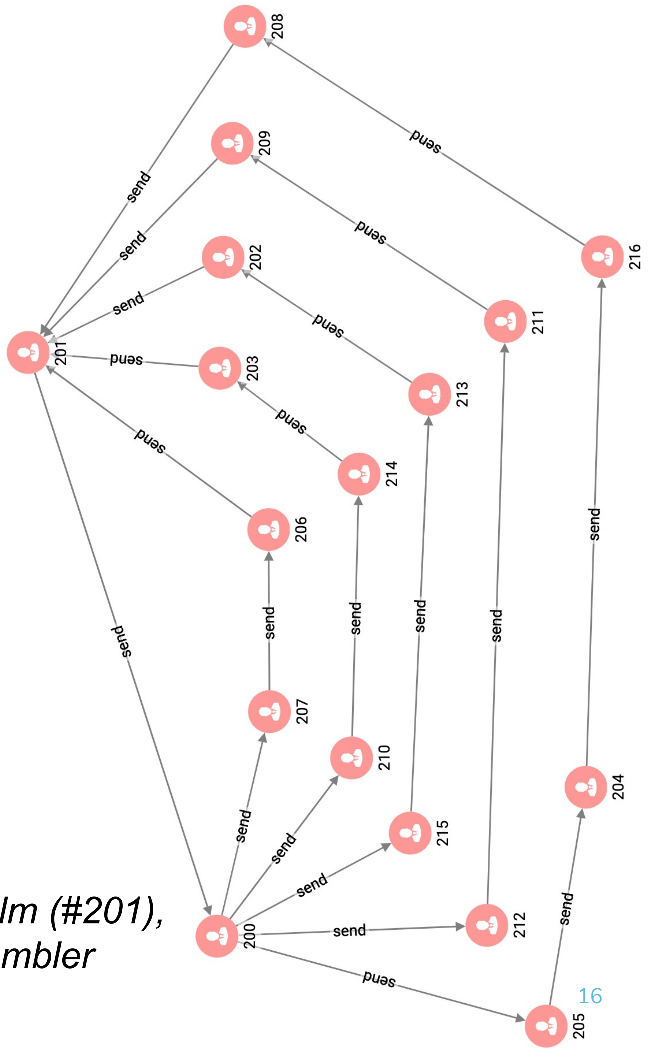


- *Easier to see*
- *The fraud ring is evident*

Graph #3

Tumbler Network Money Laundering Scheme





- *Slightly convoluted, routing-wise*
- *But, finds the tumbler network nicely !*
- *Correctly identifies the fraud actor at the helm (#201), in spite of the attempted subterfuge via a tumbler*

Challenges / TigerGraph Feature Requests

1. res = SELECT s FROM Start:s - (send:e) -> account:tgt

   WHERE tgt IN @@cycle_set

```
// Find all edges and heap them by payment date to get the head node
//ListAccum<VERTEX<account>> @@aNode = c_list.get(0);
Start = {c_list};
res = SELECT tgt FROM Start:s - (send:e) -> Start:tgt
```

The query has mixed usage of v1 and v2 syntax.

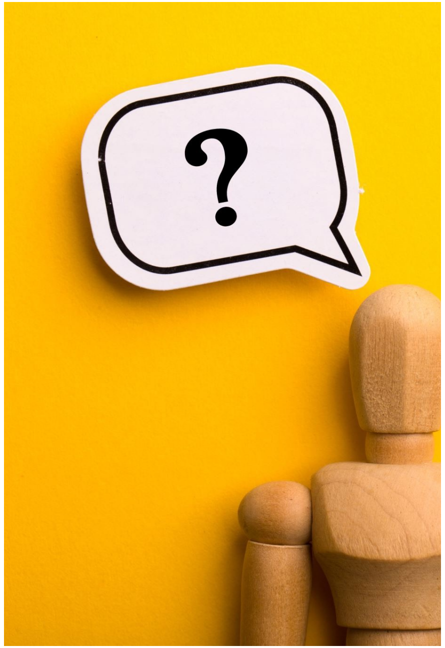
2. Lots of object conversions because of container restrictions in the object hierarchy
   - *<TG Feature Request>* ListAccum<SetAccum<.. *Won't work*
   - *<TG Feature Request>* Edges are not 1st class objects
   - *<TG Feature Request>* HeapAccum<EDGE> (100, .date_time ASC) *would be very helpful*
3. Fine grained temporals : No native support for Time Series or to unroll time
   - *<TG Feature Request> : Add Date Tree as an internal default optional property*
4. *<TG Feature Request>* Feature Engineering for Graph Neural Networks
   - *I know this is a priority for TigerGraph product management*

Lessons Learned

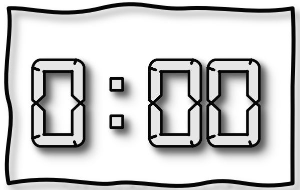
1. Graph representation is appropriate for DLT/Blockchain Fraud Detection
    - *TigerGraph is a feature rich, flexible & scalable Graph Database well suited for this problem*
    - *But it requires disciplined thinking, at times different than what we are used to !*
2. Spend time thinking about & understanding the problem
3. Make simplified assumptions and relax them as you progress (Layered approach)
4. Think Graphs & more specifically Parallel Graphs – It will take a little time to get used to that concept
    - *Read, write & learn the patterns from the GSQL Graph Algorithms and the GSQL code*
5. Draw graph diagrams to visualize the problem - *Draw the happy path 1st & then edge cases*
6. Create datasets depicting multiple scenarios – *1st use a small dataset to test the algorithms*
7. Do as much in TigerGraph as possible, staying true to the platform
    - *It is tempting to process a list outside (say in python), after a quick GSQL algorithm; don't stop there, persist (using GSQL) until you have exhausted all graph ideas*

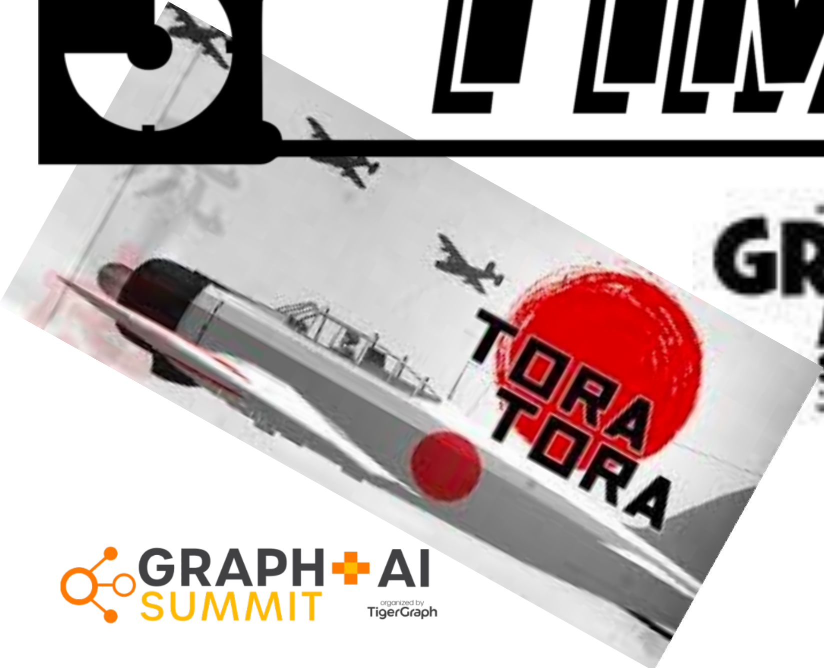
Things To Try Next … *This is only the beginning !*

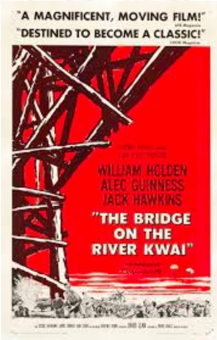
1. Scale ! Load Bitcoin/Ethereum blockchains and apply the algorithms
    - *Cross-Ledger tracking of fraud rings*
2. Fine grained temporals
    - *There could be many such rings by the same actors, so need to separate the rings by time*
    - *Solution : Time Tree "If you need to filter use vertices" – TigerGraph pragma*
    - *Opportunity for supporting Payment Networks natively in TigerGraph Graph Algorithms*
        - *Refer to <TG Feature Request> in slide #17*
3. More expressive Graph schema with derived runtime attributes, especially to track cross-ledger behaviors
4. Explore Graph Motif extraction, Weighted Graphs
5. Graph Neural Networks leveraging the extended dynamic attributes
6. Entity Resolution
    - *Need to understand heavy spans & differentiate between Exchanges, Tumblers, Mixers - Add Vertex type based customized logic*
    - *Probably via highest measure of eigenvalue centrality and Graph Neural Networks*



TOOL TIME WEEKLY







GRAPH+AI SUMMIT
organized by TigerGraph