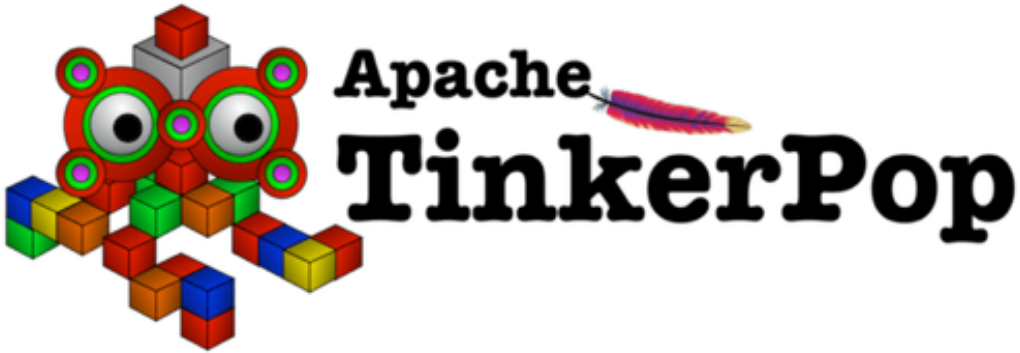
# Big Data Graphs and Apache TinkerPop 3



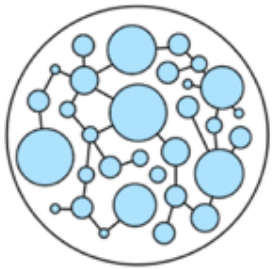

David Robinson, Software Engineer
April 14, 2015

IBM

# How This Talk Is Structured

- Part 1 Graph Background and Using Graphs
- Part 2 Apache TinkerPop 3

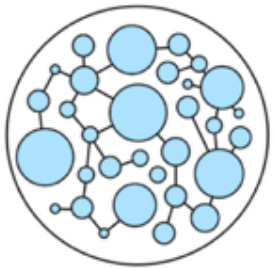
Part 1 builds a foundation to talk about Part 2, Apache TinkerPop 3 in a context



# Part 1 Graph Background

# Why would you want to use a graph ?

- Data fits a graph structure
- Intuitive way of thinking about a problem
- Data complexity can be modeled in a graph
  - Relationships as full objects with properties
- Deep or extended relationships through a set of entities
- Visualization of data in unique ways
- Effective algorithms
- Extensible schema and mixed pattern capability
  - Changes in production with minimal impact to existing code



# The sorts of things you can do with graphs

- Deriving conclusions through traversal
  - Terrorist G most likely knows Terrorist A because they both know 5 of the same people
- Objective measurement assignments and summing
  - Air route V93 LRP V39 ETX indicates the most congestion
- Identifying the most important nodes
  - Network route A is a bottleneck in the Austin to Raleigh network traffic
- Mixing Patterns
  - Bob's junk food intake is related to the playoff game of his favorite basketball team
- Sub-graph patterns
  - Triadic closures or looking for friends, missing links that indicate bot activity
- Ranking
  - what is the likelihood eboli spreads from region X to region Y ? Does an organization have technology cliques ?
- Shortest Path algorithms
  - Rerouting your dump trucks for minimum road taxes



# Real Life Graphs

- Person of Interest
- Telco Consumer Behavior
- Life Sciences
- Airline Routes and Schedules
- Retail Purchase affinities
- IT Infrastructure
- Email spam detection
- Medical Fraud detection
- Investment Recommendations
- …..

# Mix Patterns - The most interesting graphs combine multiple dimensions

- Graphs of one type of node connecting with other kinds of nodes

- People + Purchase Transactions + Local Weather + Economic data
- Package Deliver Stops + Traffic Status + Street Map + Road Weight Restrictions and Closures
- Protein Data + Gene Data + Drug Compound Data

- Graphs allow these “orthogonal models” to be stored and related



# Definition of graph for this conversation

- Property graph structure features:
  - Labels, key-value pairs, directed edges, multi-relational
  - the definition of the schema and the data elements

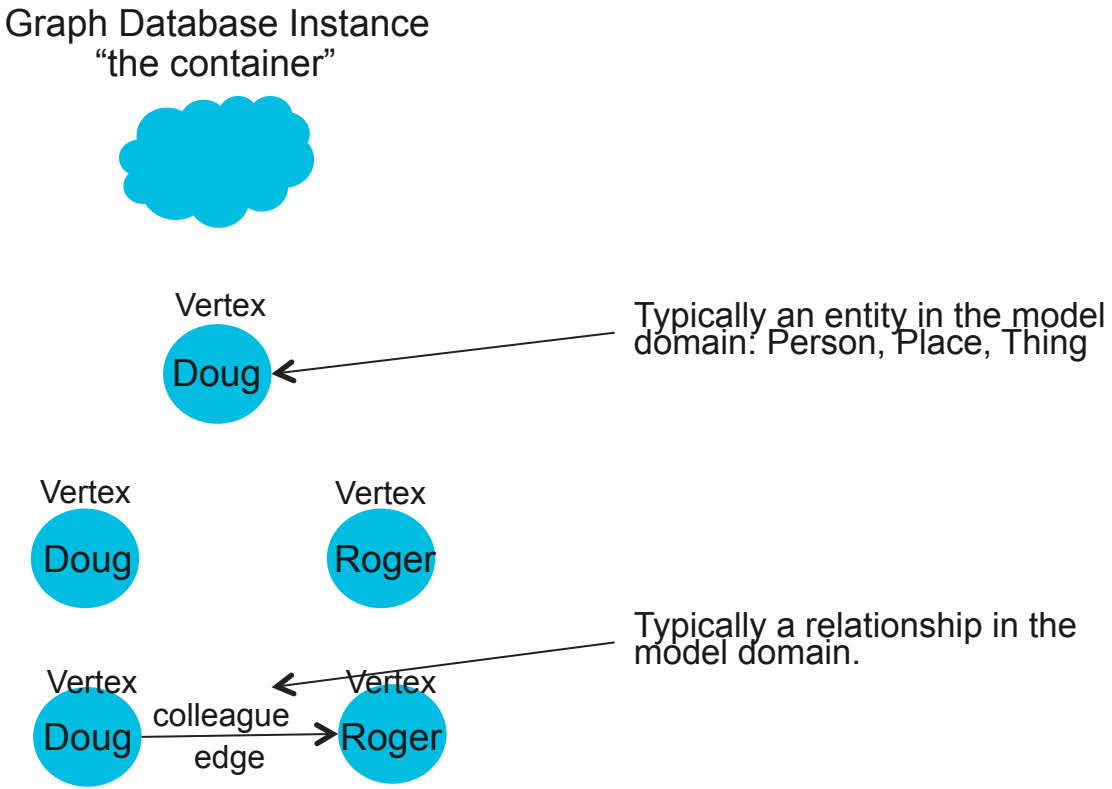
- Property graph process features:
  - Real time query and mutations possible
  - Substantial, or whole graph calculations (technically not a definition of property graph)

- Property graph structure + domain data + correct traversal yields:
  - New relational insights (prior discussion on use cases)

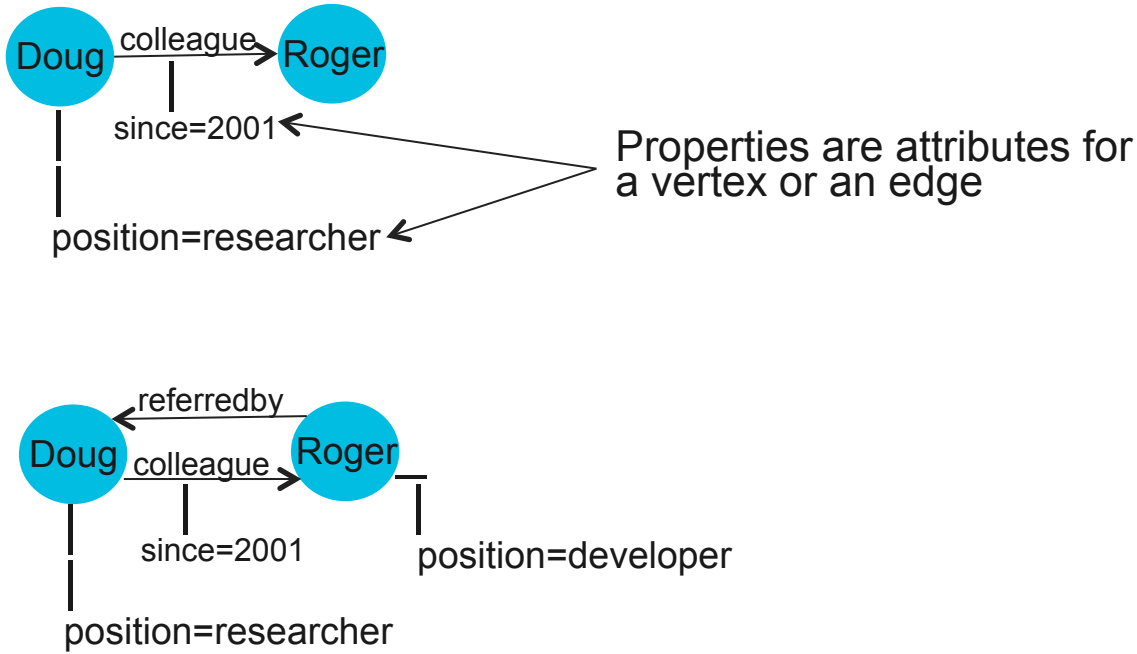
- Big data graphs are too big to fit in memory or disk of a single machine
  - other dimensions such as data locality come into play



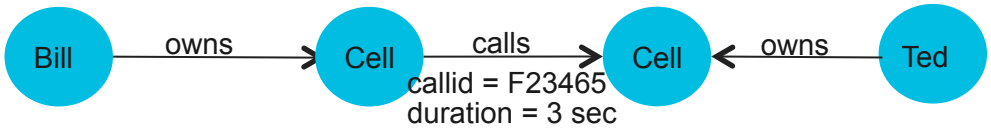
# Property graph structure – building the model



# Property graph structure – building the model



# Property graph structure – building a useful model



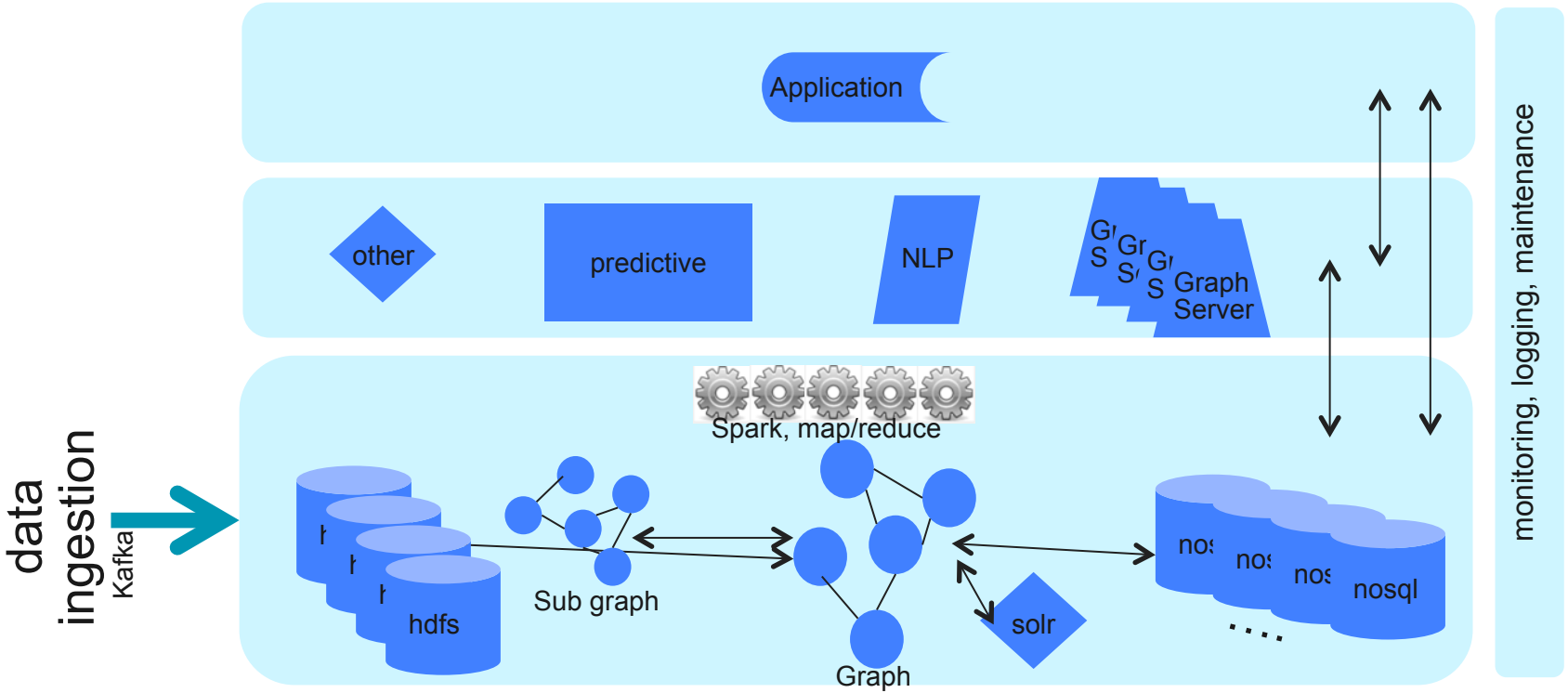

What can you do with this sort of graph ?

# A Graph *adds* to your bag of tricks

- Sometimes, is a replacement
  - more often, an addition
- Rarely the only data store for a larger application
  - Polyglot. Combined with one or more stores
- Often not the only analytics approach used in larger applications
  - Combine the strengths of different approaches to get to a better overall answer
- Often not the only visualization
  - Can provide a very useful view of the data



# What might it look like in an application?



# Graph structure and traversal appear enticingly simple to begin, deceptively difficult to optimize

- Inappropriate models
  - Examples: no properties,  generic edge labels, no time boundaries, no/poor indexes, etc
- Inappropriate queries
  - “Batch” algorithms run as real time with a user waiting
  - Selecting back unfiltered “fan-out” of 4 pulls back 400 million objects….why is this so slow ?
- Graphs at scale expose bad models and access patterns
  - What works with 6 nodes/8 edges fails with 500 million nodes/1 billion edges
- Many potential pitfalls in reconciliation, data ingestion, hydrating, indexing, sub-graphing, integration
  - The real world!



# Computing over a property graph

- Real time queries (OLTP)
  - Recommendations (may have been pre-computed and stored)
  - Depth first, lazy eval, serial
- Semi-real time calculations (Grey Area)
  - Sub-graphs, traversing large single dimensions
  - Have to figure out tooling and domain
- Bulk graph transformations (OLAP)
  - Clustering, bulk graph transformations
  - Breadth first, parallel

# Back to the beginning…

- In relational databases there is DDL and SQL
  - Concepts are the same between vendors
  - Syntax (minus extensions) is often identical
  - Greater innovation, user flexibility, larger skill base with interchangeable skills

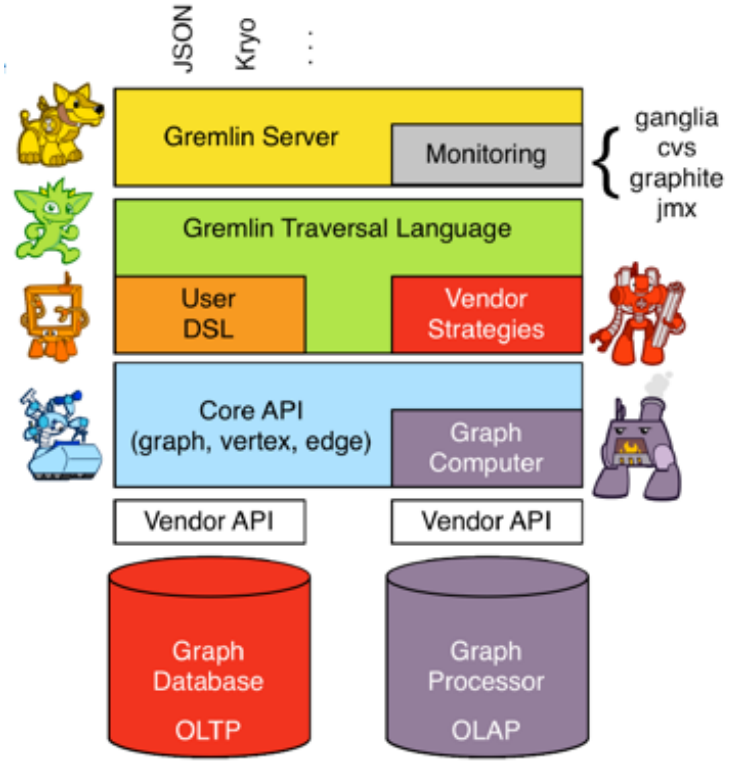
- Why not try to have the same thing for graph data stores and graph computation ?
  - ……Wait for it…..



# Part 2 TinkerPop 3

# Apache TinkerPop 3 Features

- A standard, vendor-agnostic graph API
  - Was called Blueprints
- A standard, vendor-agnostic graph query language
  - Still called Gremlin
- OLTP and OLAP engines for evaluating query
- Sample Data Server
  - “Improved Rexster”
- Reference TinkerPop 3 graph implementation
  - Tinkergraph
- Reference command line shell
  - Gremlin shell
- Reference Visualization
  - Gephi Integration



# What's New With TinkerPop 3

- Vertex Labels
- Properties on properties
- Integration into one stack
  - No more Blueprints, Pipes, Furnace, etc.
- Gremlin Server
  - Leverages netty and web sockets
- Significantly enhanced "step" library
- Interfaces for Graph Compute Engines
  - Much boarder support including Hadoop M/R. Spark, Giraph
- Java 8 Support

( Last version was TinkerPop 2.6 )



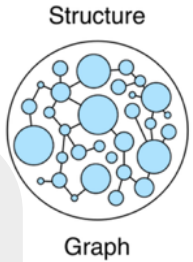
# What is the status of Apache TinkerPop 3 ?

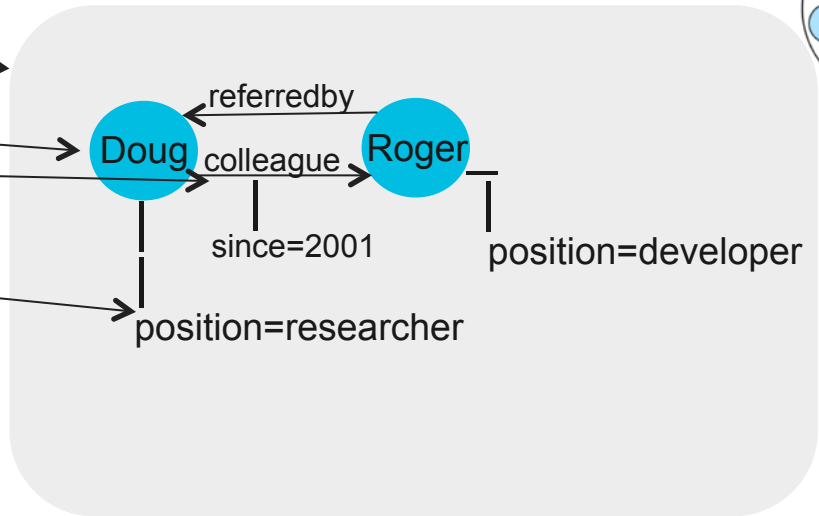
- Incubating project
  - Been around since 2009 as an open source project.
  - Just moving to Apache
- 2015 release schedule
  - April 5, 2015  TinkerPop 3 M8
  - May 5, 2015  TinkerPop 3 M9
  - May 25, 2015 TinkerPop 3 GA
- Focus on stabilizing and locking down API signatures



# Elements Of A TinkerPop 3 Graph *Structure*

- Graph
- Vertex
- Edge
- Property



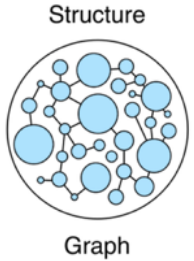




TinkerPop/gremlin/tinkergraph/structure

- TinkerEdge.java
- TinkerElement.java
- TinkerFactory.java
- TinkerGraph.java
- TinkerGraphVariables.java
- TinkerHelper.java
- TinkerIndex.java
- TinkerProperty.java
- TinkerVertex.java
- TinkerVertexProperty.java

# TinkerPop 3 graph structure APIs in action

Graph “Database” Instance
“the container”

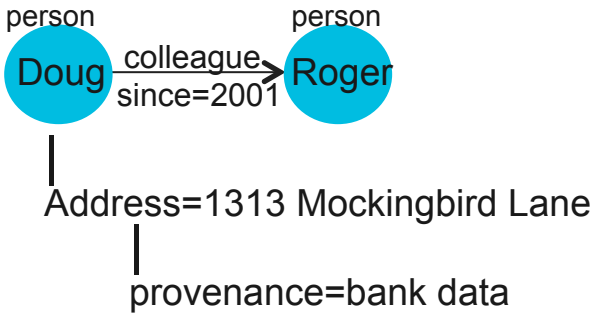



g = TinkerGraph.open()

Optional label (not shown in all examples to prevent clutter


v1 = g.addVertex(T.label, "person”, "name", ”Doug”)

v2 = g.addVertex(T.label, "person”, "name", ”Roger”)




e1 = v1.addEdge(‘colleague’, v2, ‘since’, ‘2001’ )
p1 = v1.property(“address”, “1313 Mockingbird Lane”)
pp1 = p1.property(“provenance”, “bank data”)

There is the notion of transactions

- Often implementation dependent
- tinkerpop/gremlin/structure/Transaction.java

# What does this look like with different graph databases ?

## Titan

```
g = SpecificGraphFactory.open()
v1 = g.addVertex(T.label, "person”, "name", ”Doug”)
v2 = g.addVertex(T.label, "person”, "name", ”Roger”)
e1 = v1.addEdge(‘colleague’, v2, ‘since’, ‘2001’ )
p1 = v1.property(“address”, “1313 Mockingbird Lane”)
pp1 = p1.property(“provenance”, “bank data”
```

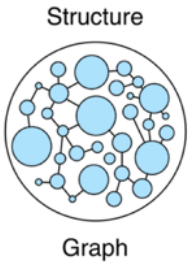
## InfiniteGraph

```
g = SpecificGraphFactory.open()
v1 = g.addVertex(T.label, "person”, "name", ”Doug”)
v2 = g.addVertex(T.label, "person”, "name", ”Roger”)
e1 = v1.addEdge(‘colleague’, v2, ‘since’, ‘2001’ )
p1 = v1.property(“address”, “1313 Mockingbird Lane”)
pp1 = p1.property(“provenance”, “bank data”
```

## OrientDb

```
g = SpecificGraphFactory.open()
v1 = g.addVertex(T.label, "person”, "name", ”Doug”)
v2 = g.addVertex(T.label, "person”, "name", ”Roger”)
e1 = v1.addEdge(‘colleague’, v2, ‘since’, ‘2001’ )
p1 = v1.property(“address”, “1313 Mockingbird Lane”)
pp1 = p1.property(“provenance”, “bank data”
```

There seems to be a pattern here….

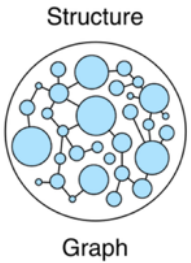


# Many more TinkerPop 3 Structure API method signatures

- See
  - Documentation ( http://tinkerpop.incubator.apache.org/docs/3.0.0-SNAPSHOT/ )
  - Javadoc (http://tinkerpop.incubator.apache.org/javadocs/3.0.0-SNAPSHOT/full/ )

  - The code…
    - tinkerpop3/tinkergraph-gremlin/src/test/java/org/apache/tinkerpop/gremlin/tinkergraph/structure

- The specific set of APIs supported depends on the open source or vendor implementation
  - the underlying technology often dictates what can be supported



# Gremlin: A graph query language

- Imperative graph traversal language
  - Sequences of ‘steps’ for the computation
  - Contrast to declarative: logic without specific steps
- Must understand structure of graph
  - This isn’t so different…
- Most easily written in Groovy
  - Prior to Java 8, writing with “Pipes” in Java was not for the faint of heart
- User defined functions (steps) possible
- User defined DSL capability
  - Write a domain specific language if you want

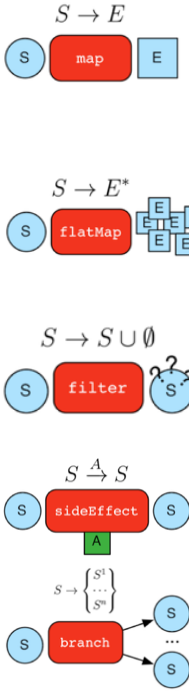
# Gremlin can be challenging

- Empirical observation: most frequent question on mailing lists:
  - "I need help with this query"
  - Understanding how to string step commands together requires a depth of knowledge
- OLTP turns into OLAP queries
  - "select *" queries on big data not quite as obvious
- App developers don't get
  - "I just want to write my program"
  - "I want an object mapping layer"

    hint: DSL can help
- Complex traversals and step assembly is hard
  - If you don't do it all day, every day, skill fades fast
  - An acquired skill
- *Must* understand the object types flowing through the 'pipe'
  - How ……DO I get xyz out of this pipe or this object ?
  - "Traverser" objects flow through the query pipe

# Traversals in Apache TinkerPop 3
# All about that Lambda…

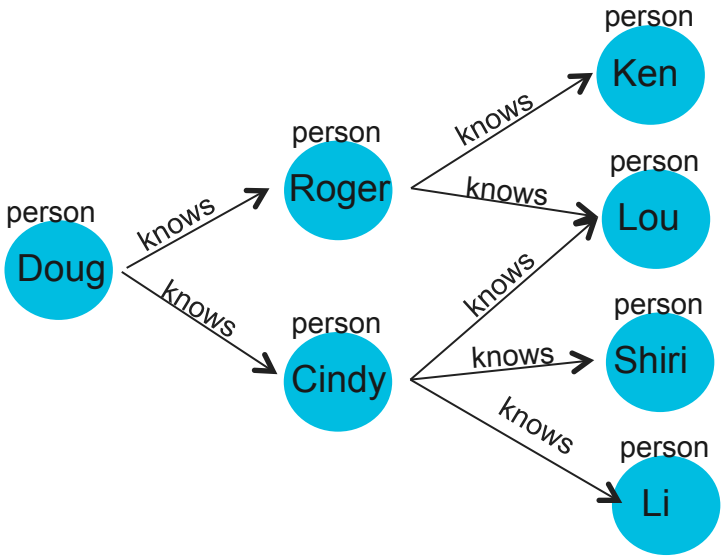
- There are five conceptual graph
  - Reality is underlying stores don't know how to optimize lambda steps
  - map(Function<Traverser<S>, E>)
    - map the traverser to some object of type E for the next step to process
  - flatMap(Function<Traverser<S>, Iterator<E>>)
    - map the traverser to an iterator of E objects that are streamed to the next step
  - filter(Predicate<Traverser<S>>)
    - map the traverser to either true or false, where false will not pass the traverser to the next step.
  - sideEffect(Consumer<Traverser<S>>)
    - perform some operation on the traverser and pass it to the next step.
  - branch(Function<Traverser<S>,M>)
    - split the traverser to all the traversals indexed by the M token.



# Example OLTP Gremlin Query

g.V(1).out('knows').out('knows').count()  or g.V(1).repeat(out()).times(2).count()

Starting with the vertex with id=1, follow all the out 'knows' edges to the adjacent vertices, then follow all of their out 'knows' edges to the next set of vertices and return the count of vertices visited.



# Cool concepts in TinkerPop 3

- Local vs global
  - Scoping of the step actions
- By step modulator
  - The general pattern is step().by()
  - A main reason why things like “filter” are not needed anymore
- Match step
  - Enables more of a declarative form of querying
  - Good for larger graphs where statistics of edge counts, for example, are unknown
- Profile step
  - Side effect step allowing developer to understand their query better
  - Leverages the new Traverser pattern used in the query pipeline
- Sack step
  - Passing data through the traversal is now much easier
- barriers
  - When a computation in a pipe is not lazy, a "barrier step" exists



# Apache TinkerPop 3 Gremlin Steps – Lots of Options

AddEdge
Aggregate
Back
Between
Chose
Coin
CyclicPath
Dedup
Except
Fold
Group
GroupCount
Has
Inject
Limit
Local
Match
Order
Path
Profile
Range
Repeat
Retain
Sack
Sample
Select
Shuffle
SimplePath
Store
Subgraph
TimeLimit
Tree
Unfold
Union
ValueMap
Vertex

# Apache TinkerPop 3 Transactions and Schema

- Still dependent on the underling graph database
- More flexibility in controlling “auto” vs “manual’, etc.

# Apache TinkerPop 3 and Graph Computing (OLAP)

- Conceptually think BSP (bulk synchronous parallel)
  - Leverages messaging passing concepts of underlying store
- Still use Gremlin
  - **Sweet !**
  - subset of Gremlin
  - Masks differences OLAP engines
    - Still must understand tuning and behavioral aspects of the specific engine
- When to OLAP ? (just examples)
  - Graph back ups
  - Copying to an analytics cluster
  - Whole graph mutations for migration or schema changes
  - Whole graph analytics, like clustering, k-means, etc.
- Primarily uses the message passing technology of the underlying compute engine
  - Spark, Giraph, Hama, etc



# Apache TinkerPop 3 and Graph Computing (OLAP)

- Hadoop-Gremlin supports 3 GraphComputer implementations:
- GiraphGraphComputer - Uses Giraph as the execution engine
  - Graph size restrictions may apply
- SparkGraphComputer - Uses Spark as the execution engine
  - Supports larger graphs. Uses disks (and is slower) for larger graphs
- MapReduceGraphComputer - Hadoop's MapReduce
  - Massive graphs. Slowest traversals. Largest scale.

- Tradeoffs between the different graph computing options
- Can use multiple approaches in the same implementation
- Can write your own by extending the classes



# Apache TinkerPop 3 Traversal Strategies

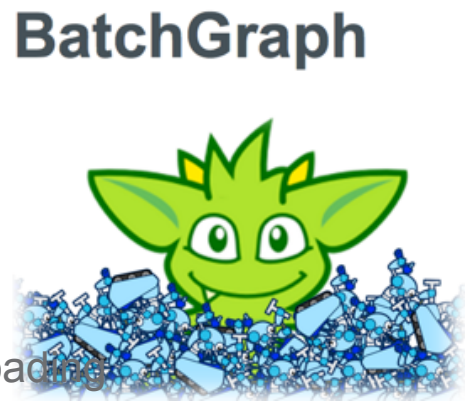
- Troubles with “strategies” right now
  - Hopefully this gets fixed before GA

- Concept of wrapping the basic graph to provide additional, custom behavior
  - Read only graph
  - Sub Graphs
  - Partitioned Graphs
  - Event Graphs
  - User Assigned IDs
  - Etc.



# Apache TinkerPop 3 Data Ingestion

- For bulk loading of “larger” data sets
- Limitations around edge property setting
- Presorting data makes batch loading faster
  - Cache hits
- Works with incremental or new loading
- Best for single machine graph databases




- Personal experience…
  - Can still feel slow with large datasets
  - Not a solution for distributed installations and parallel loading
- Use OLAP loading across a cluster for most efficient large data loading

# Apache TinkerPop 3 Data I/O

- Reduced supported formats to 3
  - GraphML
    - Best for integration with other tools
    - Lossy in that type information is not conveyed
  - GraphSON
    - Slightly different from previous TInkerPops due to new features such as vertex labels and multi-properties
    - Slightly lossy in that JSON data types don't represent all Java types
    - Design graph based on JSON data types
  - Gryo for binary serialization
    - Kryo serialization
    - Serialization of individual graph elements, graph migrations.
- Customizable
  - vendor specific implementations

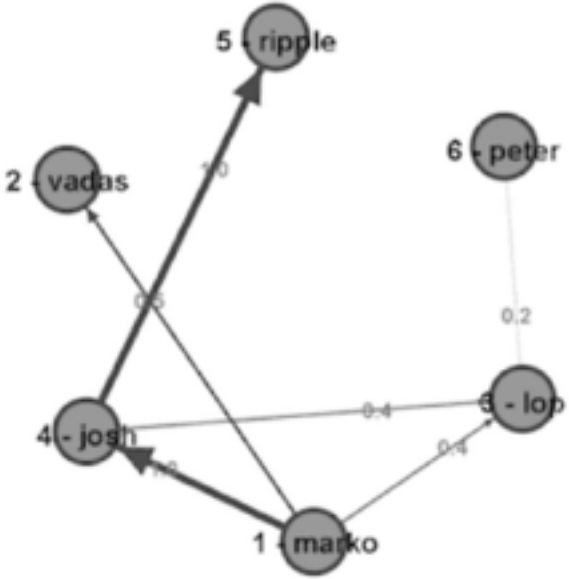
# Apache TinkerPop 3 Gremlin Console

- Useful for development, verifying graph, quick maintenance a variety of graph related activities

```
gremlin> g.V.has(outE('knows') & outE('created')).name //(1)
==>marko
gremlin> t = g.V.has(outE('knows') | inE('created')).name; null //(2)
==>null
gremlin> t.toString()
==>[GraphStep(vertex), HasTraversalStep([VertexStep(OUT,[knows],edge), OrMarker, VertexStep(IN,[created],edge)]),
gremlin> t
==>marko
==>lop
==>ripple
gremlin> t.toString()
==>[TinkerGraphStep(vertex), HasTraversalStep([OrStep([[VertexStep(OUT,[knows],edge)], [VertexStep(IN,[created],ed
```

# Apache TinkerPop 3 and Visualization

- Pre-built integration for Visualization
- Subsetting for large graphs required
  - Roll your own subsetting




# Apache TinkerPop 3 Vendor Integration

- Extensive documentation so you can write your own graph driver
  - Get all the benefits of TinkerPop 3 for your new implementation

# Apache TinkerPop 3 Areas for growth ?

- Ingestion patterns and performance
- Subgraphing
- Gremlin usability (?)
- Continued improvements as new processing techniques appear
  - GPU acceleration (?)
- Tooling
  - Query assistance
  - Visualization
  - Profiling
  - Modeling
  - Test data generation
- Other ?



# Apache TinkerPop 3 – A call to action

- TinkerPop 3 is a significant stride in standardizing graph usage patterns across implementations
  - Improved and hardened from field experience
  - Does not mean it has “arrived”
- Contribute your experience to make Apache TinkerPop 3 better
- Port your existing graph database to Apache TinkerPop 3 from older versions
- Encourage your open source project or vendor to implement TInkerPop 3
  - A better ecosystem
  - Protecting application investment



# Thank You



# Credits

Thank you to Marko Rodriguez, Stephen Mallet for their tireless pursuit of an open graph ecosystem, the concept of the TinkerPop 3 project in the first place and their investment in walking me through TInkerPop 3 concepts.

Thank you to Jason Plurad for his review and suggestions for the material included in this presentation.

Thank you to Ketrina Yim for the best open source graphics of any project I have seen.

Thank you to the entire Tinkerpop 3 team and mailing list participants.

Many of the slides in this presentation come directly from the TinkerPop 3 documentation.



# References

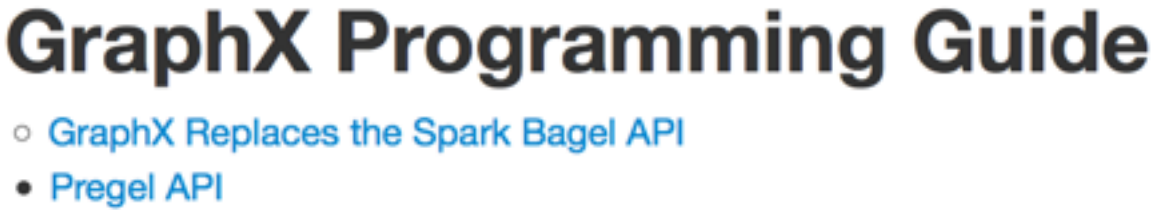
- http://TinkerPop.incubator.apache.org/
- http://en.wikipedia.org/wiki/Category:Graph_algorithms
- http://www.slideshare.net/slidarko
- http://gephi.github.io/
- https://spark.apache.org/
- http://giraph.apache.org/
- https://hadoop.apache.org/



# Backup

# Apache TinkerPop 3 provides a vendor-agnostic way of interacting with property graphs

- Let's say we want to use GraphX…
  - We can learn GraphX specifics…




- Or, we can use TinkerPop

# Apache TinkerPop 3 provides a vendor-agnostic way of interacting with property graphs

- What if we decide that we need Apache Giraph to perform graph calculations ?
  - We can learn how to write Giraph Java Programs….


- Or, we can use TinkerPop

`GiraphGraphComputer`: Leverages Giraph to execute TinkerPop3 OLAP computations.

# Apache TinkerPop 3 provides a vendor-agnostic way of interacting with property graphs

- What if we decide we also need a graph database in addition to a graph compute engine?


- OrientDB adapter to use it inside Gremlin
- OrientDB implementation of TinkerPop Blueprints

Titan

- Native integration with the TinkerPop graph stack:
  - Gremlin graph query language