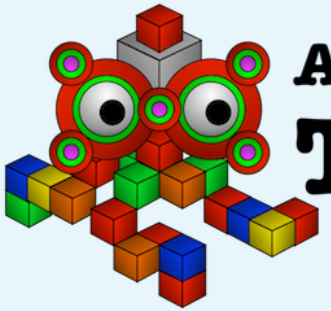
The Gremlin Graph Traversal Machine and Language



Apache TinkerPop

http://tinkerpop.incubator.apache.org

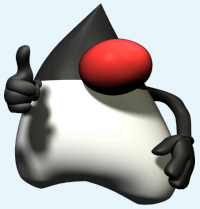
Dr. Marko A. Rodriguez
Director of Engineering at DataStax, Inc.
Project Management Committee, Apache TinkerPop

DATASTAX

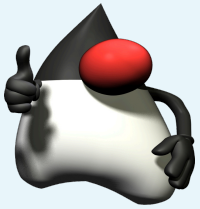
acm SIGPLAN

Database Programming Languages

Rodriguez, M.A., “The Gremlin Graph Traversal Machine and Language,” Proceedings of the ACM Database Programming Languages Conference, doi:10.1145/2815072.2815073, ACM, Pittsburg, Pennsylvania, October 2015.

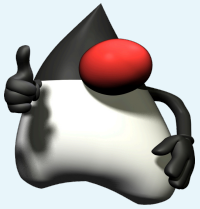


Java is a virtual machine.
Java is a programming language.



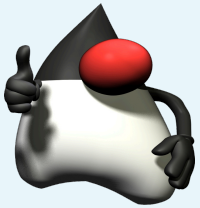
Java is a virtual machine.
Java is a programming language.

Gremlin is a traversal machine.
Gremlin is a traversal language.

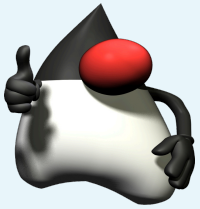


Java is a virtual machine.
Java is a programming language.

Gremlin is a traversal machine.
Gremlin is a traversal language.

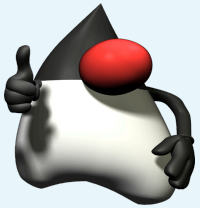


Java is operating system agnostic.
MacOSX, Linux, Windows, etc.



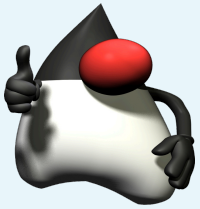
Java is a virtual machine.
Java is a programming language.

Gremlin is a traversal machine.
Gremlin is a traversal language.



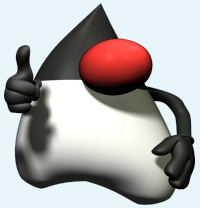
Java is operating system agnostic.
MacOSX, Linux, Windows, etc.

Gremlin is graph system agnostic.
Titan, Neo4j, Stardog, Giraph, Spark, Hadoop, etc.



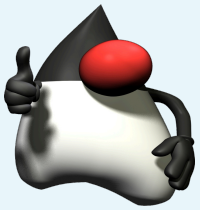
Java is a virtual machine.
Java is a programming language.

Gremlin is a traversal machine.
Gremlin is a traversal language.



Java is operating system agnostic.
MacOSX, Linux, Windows, etc.

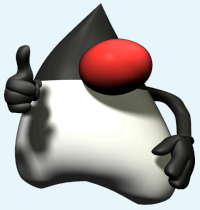
Gremlin is graph system agnostic.
Titan, Neo4j, Stardog, Giraph, Spark, Hadoop, etc.



Other languages compile to the Java virtual machine.
Groovy, Scala, Clojure, JavaScript Nashorn, etc.

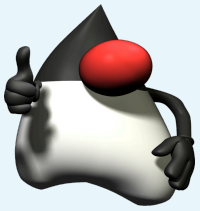
Java is a virtual machine.
Java is a programming language.

Gremlin is a traversal machine.
Gremlin is a traversal language.



Java is operating system agnostic.
MacOSX, Linux, Windows, etc.

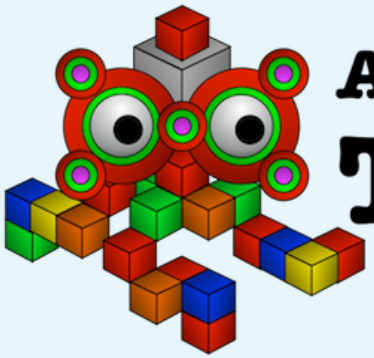
Gremlin is graph system agnostic.
Titan, Neo4j, Stardog, Giraph, Spark, Hadoop, etc.



Other languages compile to the Java virtual machine.
Groovy, Scala, Clojure, JavaScript Nashorn, etc.

Other languages compile to the Gremlin traversal machine.
Gremlin-Groovy, Gremlin-Scala, SPARQL, etc.

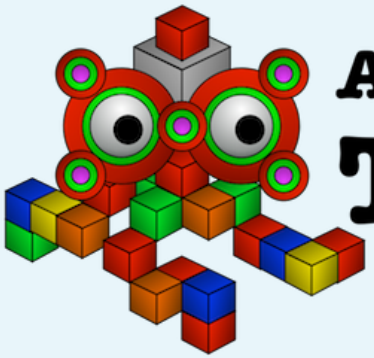
Rodriguez, M.A., Kuppitz, D., “The Benefits of the Gremlin Graph Traversal Machine," DataStax Engineering Blog, 2015.
http://www.datastax.com/dev/blog/the-benefits-of-the-gremlin-graph-traversal-machine



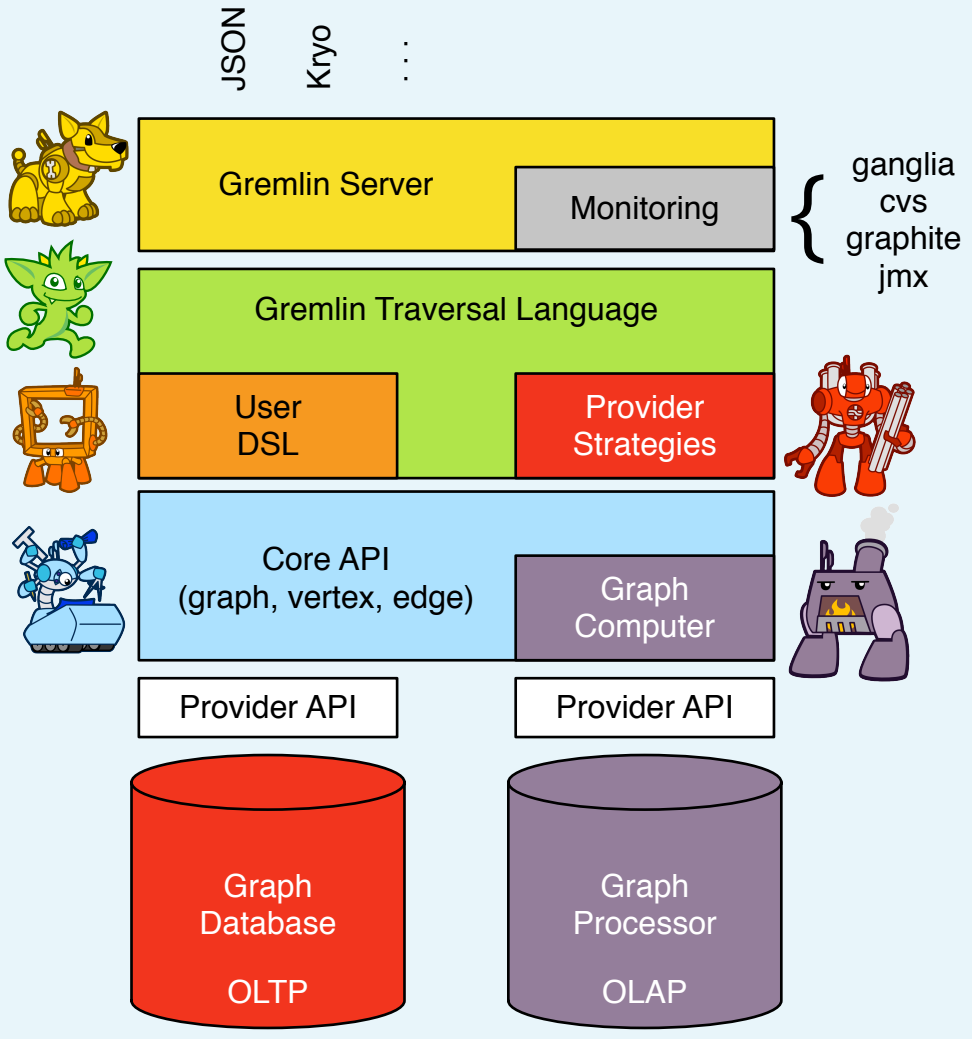
Apache TinkerPop

- TinkerPop is an open source graph computing framework.
- TinkerPop was started in 2009 by Marko A. Rodriguez, Josh Shinavier, and Peter Neubauer.
- TinkerPop joined the Apache Software Foundation January 2015 (currently in incubation).
- TinkerPop designs and develops the Gremlin graph traversal machine and language.
- TinkerPop is supported by nearly every graph system provider (both commercial and open source).
- TinkerPop3 has been in development for 2 years and was officially released July 2015.

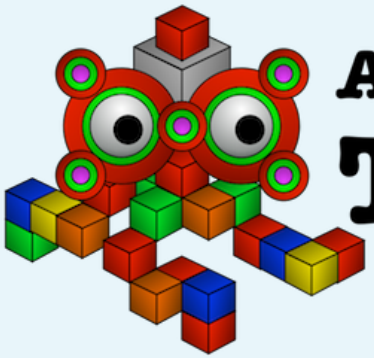
"What is The TinkerPop?"



Apache TinkerPop



"What is The TinkerPop?"



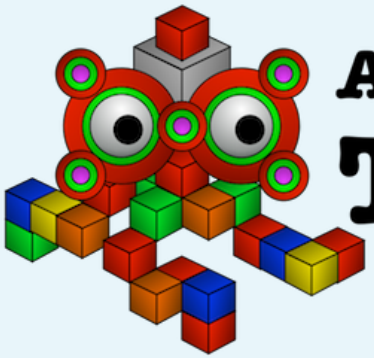
Apache TinkerPop

- TinkerPop provides graph computing capabilities to any data processing system.
- TinkerPop supports both OLTP (database) and OLAP (batch processor) systems.
- TinkerPop is used to process the order fulfillment network of Amazon.com
  ("100 billion vertices" -- at least factor 10x on edges?).

  https://videos.dabcc.com/aws-reinvent-2015-dat203-building-graph-databases-on-aws/
- TinkerPop is the sole interface for >50% of all graph systems in the market/wild.
- TinkerPop currently processes a near trillion edge graph represented over 1000 machines.

  (can't disclose company name)
- TinkerPop works with Apache Cassandra, Apache HBase, Apache Hadoop, Apache Spark, Apache Giraph, Apache Atlas, Apache Falcon, ...
- TinkerPop is always looking for more contributors and perhaps TinkerPop could be your future software/language design and development home.

"Why should you care about The TinkerPop?"

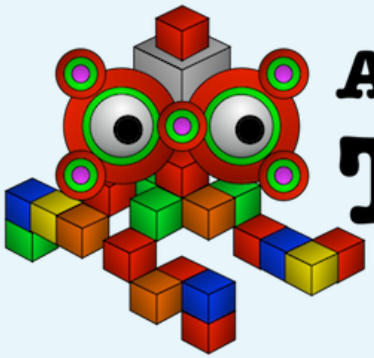


Apache TinkerPop

- TinkerPop provides graph computing capabilities to any data processing system.
- TinkerPop supports both OLTP (database) and OLAP (batch processor) systems.
- Tinkerpop has language bindings in every major programming language.
- TinkerPop is used to process the order fulfillment network of Amazon.com ("100 billion vertices" -- at least factor 10x on edges?).

  https://videos.dabcc.com/aws-reinvent-2015-dat203-building-graph-databases-on-aws/
- TinkerPop is the sole interface for >50% of all graph systems in the market/wild.
- TinkerPop currently processes a near trillion edge graph represented over 1000 machines.
- TinkerPop works with Apache Cassandra, Apache HBase, Apache Hadoop, Apache Spark, Apache Giraph, Apache Atlas, Apache Falcon, ...
- TinkerPop is always looking for more contributors and perhaps TinkerPop could be your future software/language design and development home.

"Why should you care about The TinkerPop?"



Apache TinkerPop

- TinkerPop provides graph computing capabilities to any data processing system.
- TinkerPop supports both OLTP (database) and OLAP (batch processor) systems.
- TinkerPop is used to process the order fulfillment network of Amazon.com ("100 billion vertices" -- at least factor 10x on edges?).

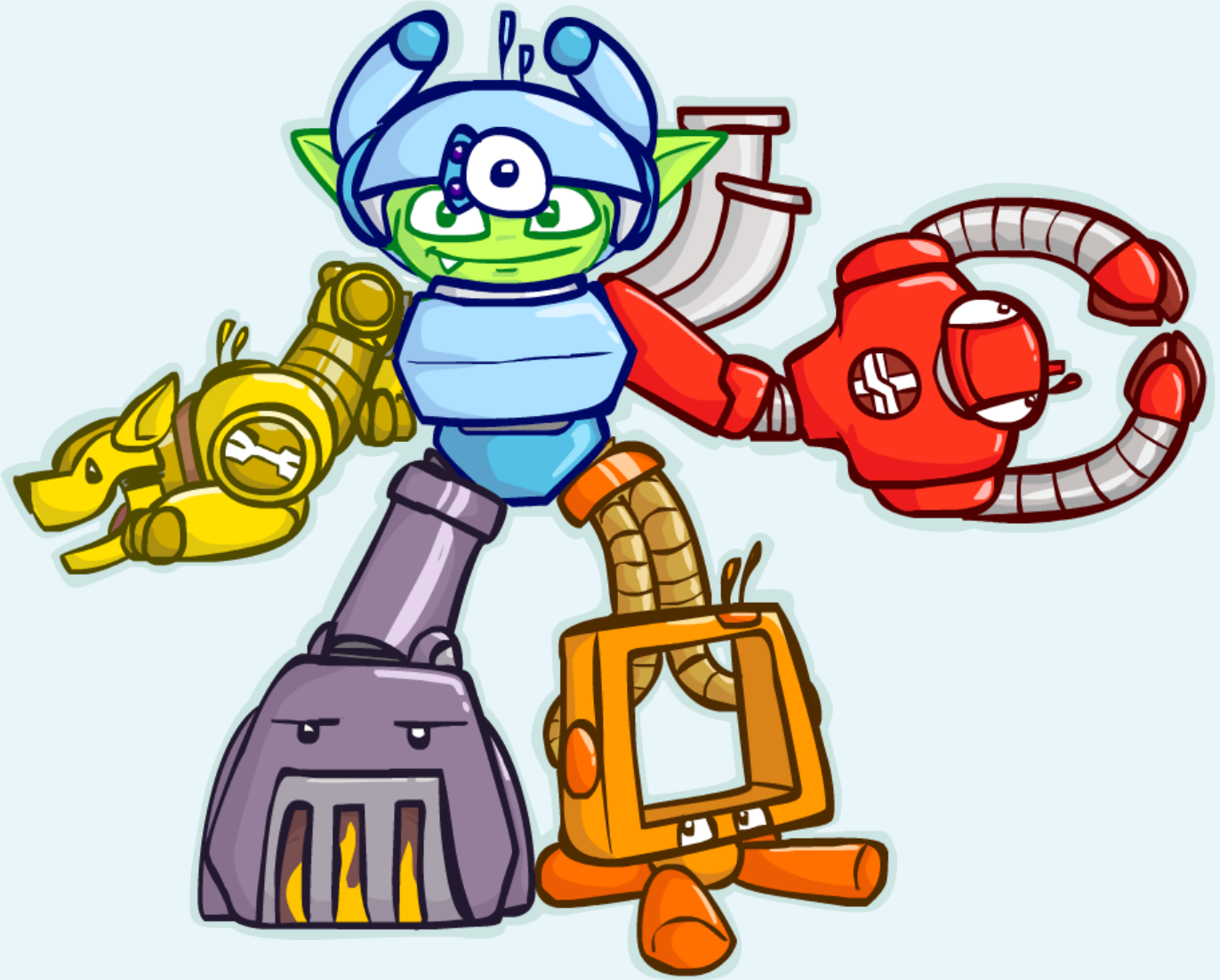
  https://videos.dabcc.com/aws-reinvent-2015-dat203-building-graph-databases-on-aws/
- TinkerPop is the sole interface for >50% of all graph systems in the market/wild.
- TinkerPop currently processes a near trillion edge graph represented over 1000 machines.
- TinkerPop works with Apache Cassandra, Apache HBase, Apache Hadoop, Apache Spark, Apache Giraph, Apache Atlas, Apache Falcon, ...
- TinkerPop is always looking for more contributors and perhaps TinkerPop could be your future software/language design and development home.

"Why should you care about The TinkerPop?"

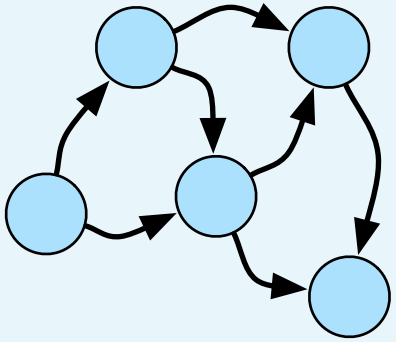
TinkerPop has language bindings in every major programming language.

TinkerPop is also used for small in-memory graphs.

Part 1: The Gremlin Traversal Machine



The Machine Components



The Graph

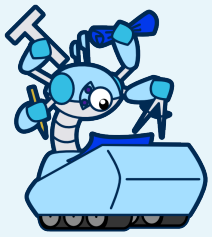
The Data

The Traverser

The CPU

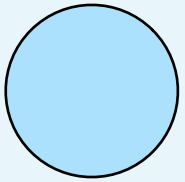
The Traversal

The Program

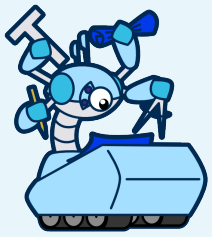


The Graph

vertex

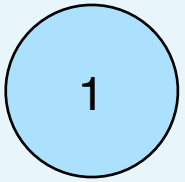


"The graph is the 'RAM'."

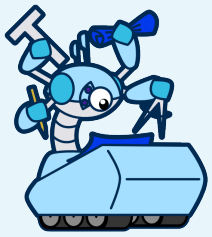


The Graph

vertex id

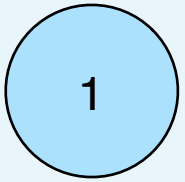


"Vertices and edges in the graph have unique ids (addresses)."

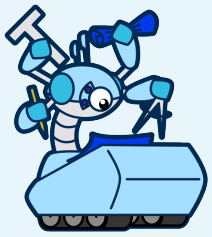


The Graph

vertex label

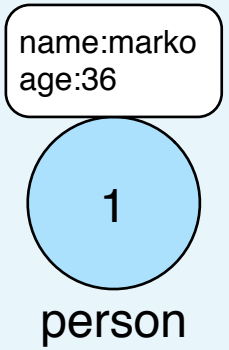


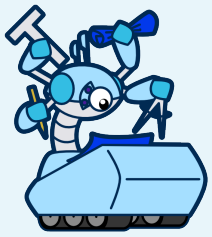
person



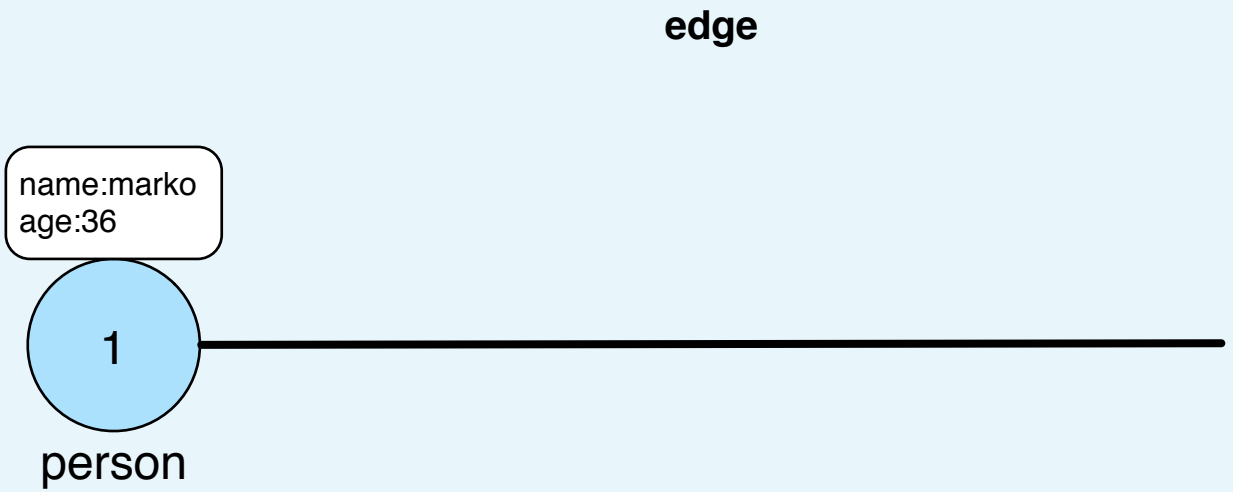
The Graph

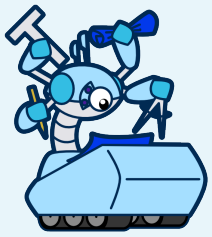
vertex properties



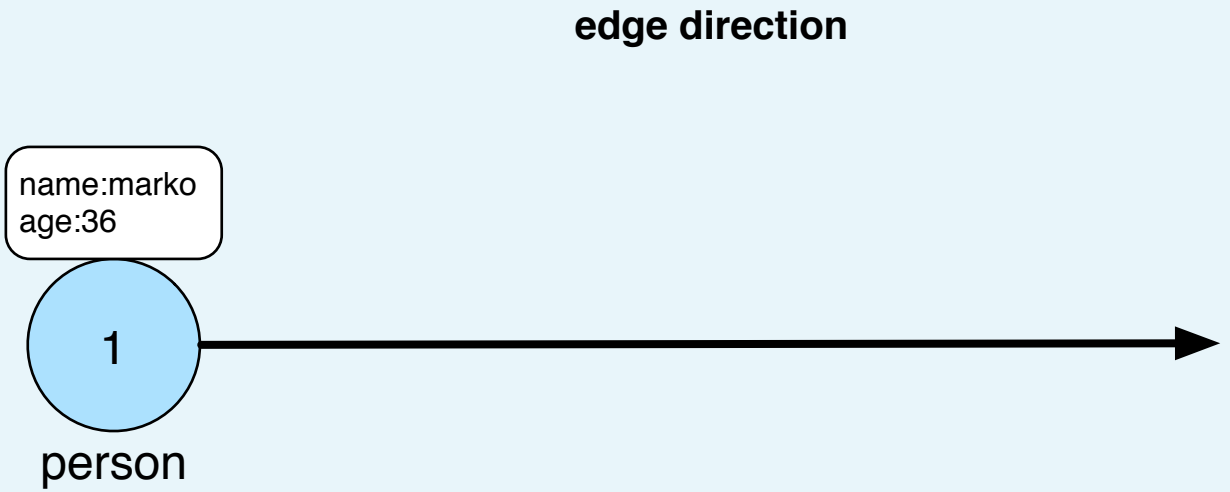


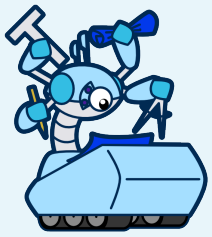
The Graph



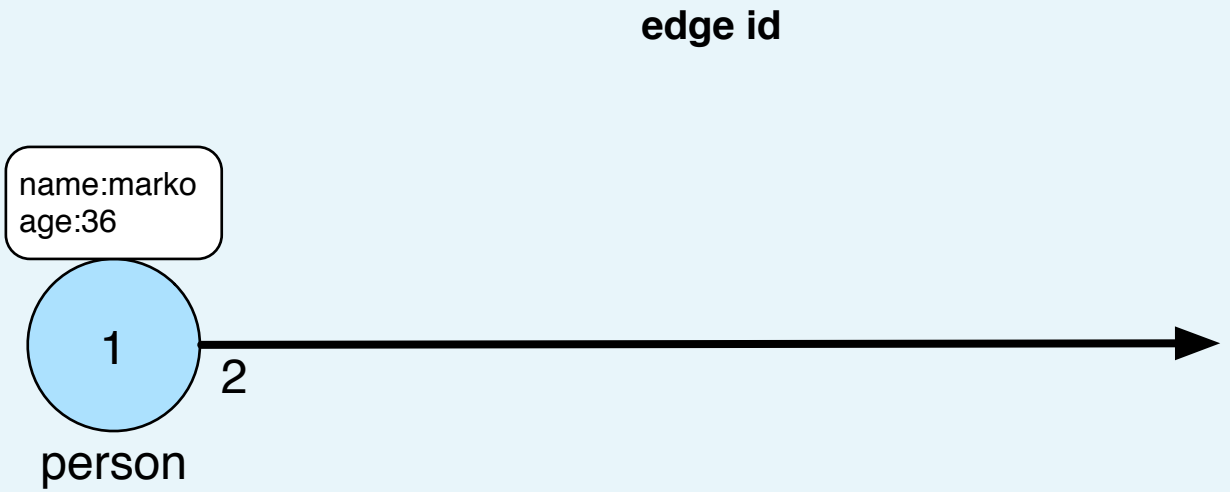


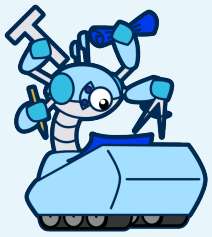
The Graph



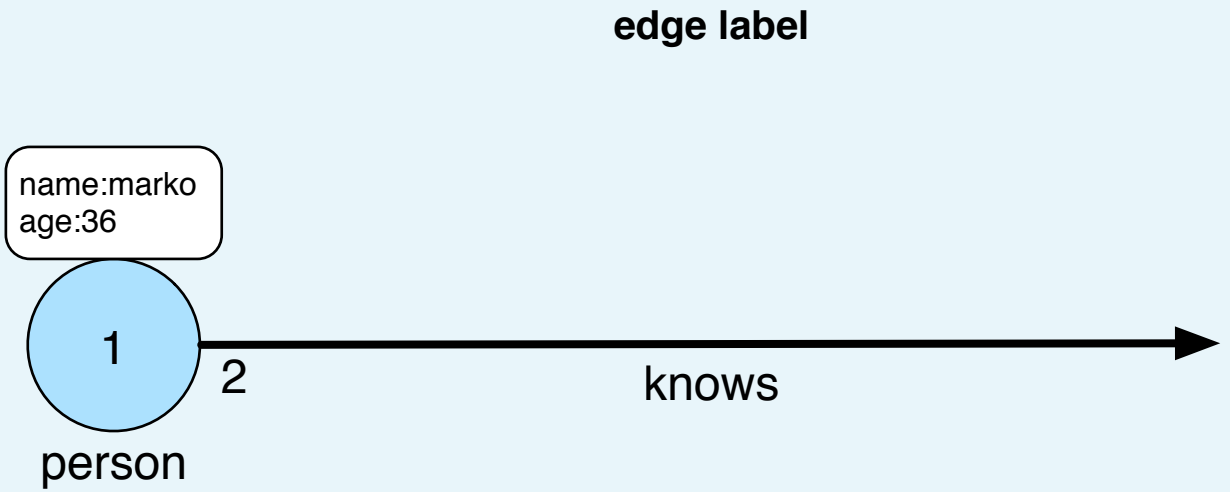


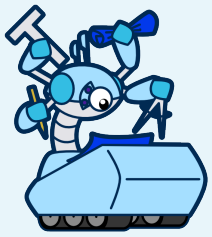
The Graph



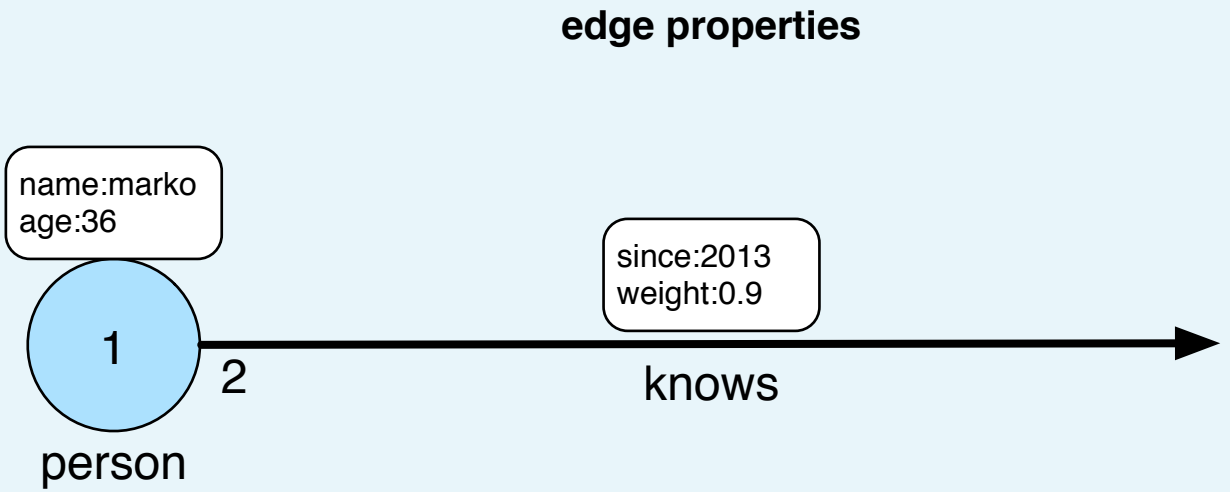


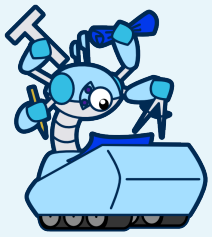
The Graph



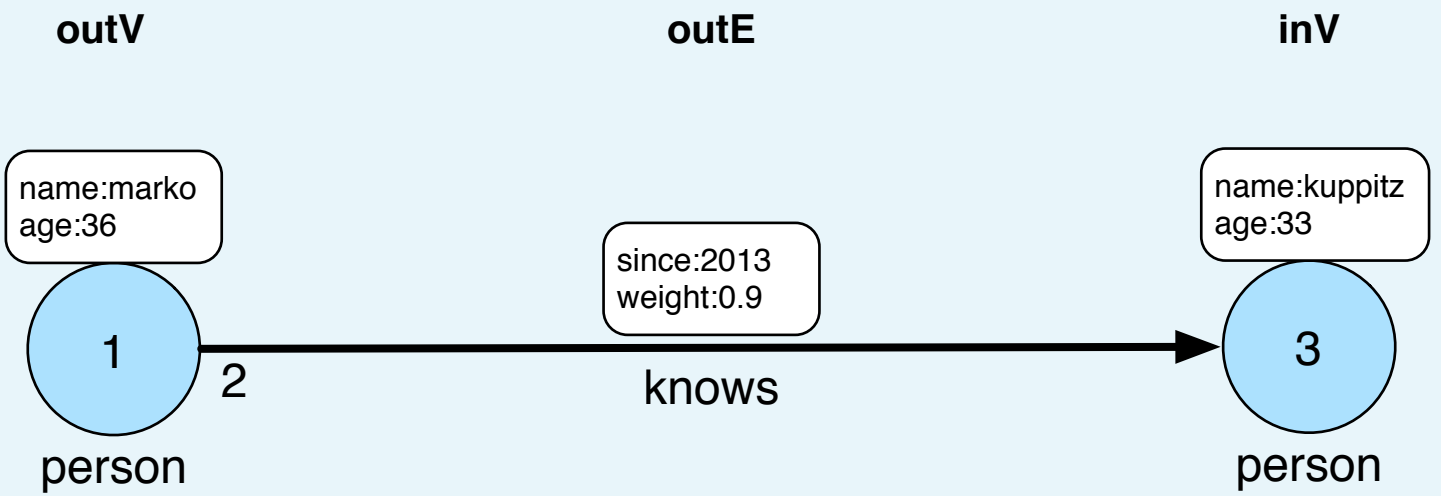


The Graph

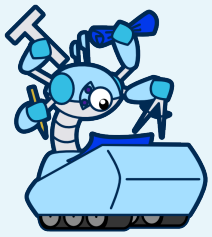

edge properties
name:marko
age:36
1
person
2
since:2013
weight:0.9
knows



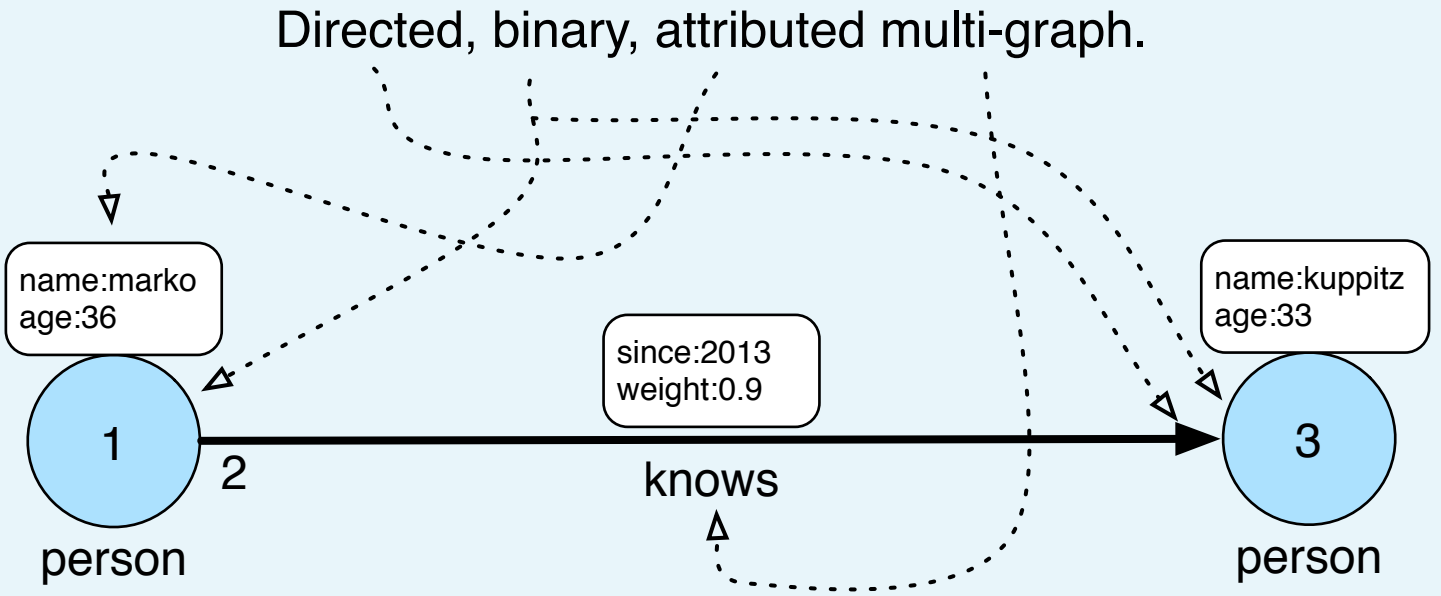
The Graph

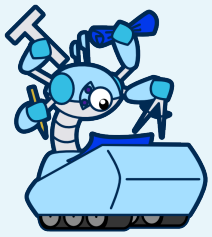


"Vertices are related by edges (addresses have pointers to each other)."



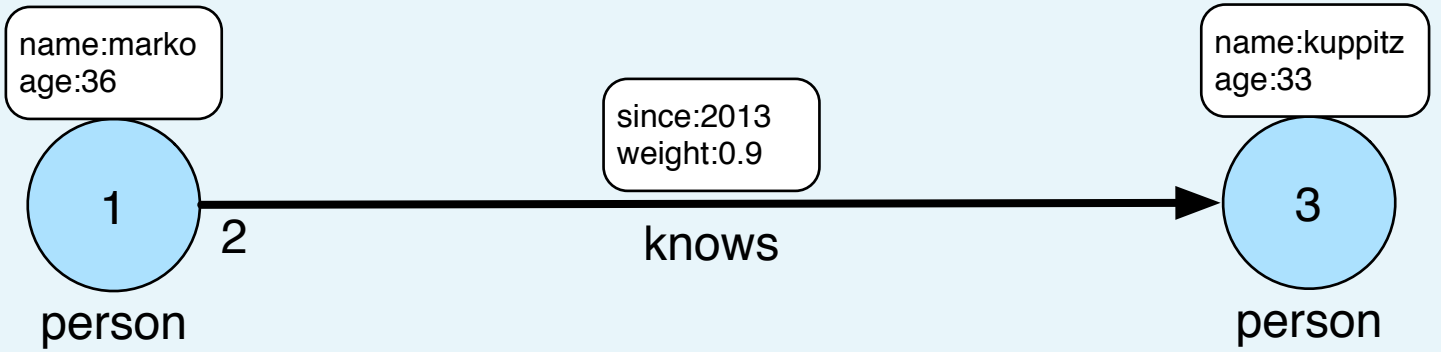
The Graph


Directed, binary, attributed multi-graph.
name:marko
age:36
since:2013
weight:0.9
name:kuppitz
age:33
1
2
knows
3
person
person



The Graph

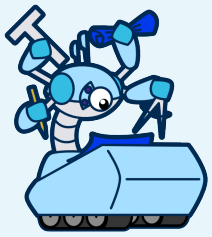
Directed, binary, attributed multi-graph.



edges — directed, binary — vertices or edges — property key (string) — properties can't reference vertices or edges

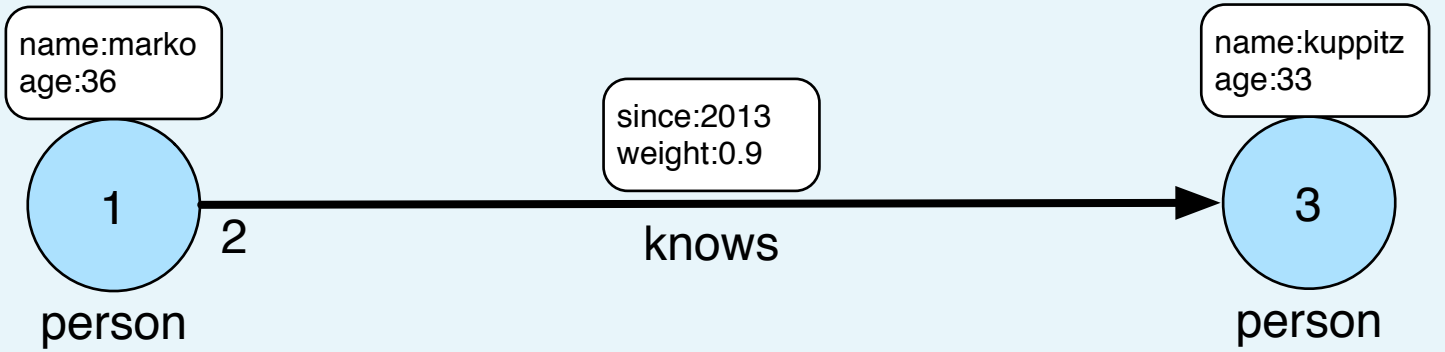
$$G = (V, E \subseteq (V \times V), \lambda : (V \cup E) \times \Sigma^* \rightarrow U \setminus (V \cup E))$$

vertices — labels+properties function — U = anything



The Graph

Directed, binary, attributed multi-graph.

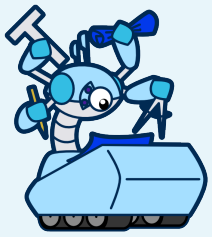


$$G = (V, E \subseteq (V \times V), \lambda : (V \cup E) \times \Sigma^* \rightarrow U \setminus (V \cup E))$$

$\lambda(v_1, \texttt{name}) \mapsto \texttt{marko}$

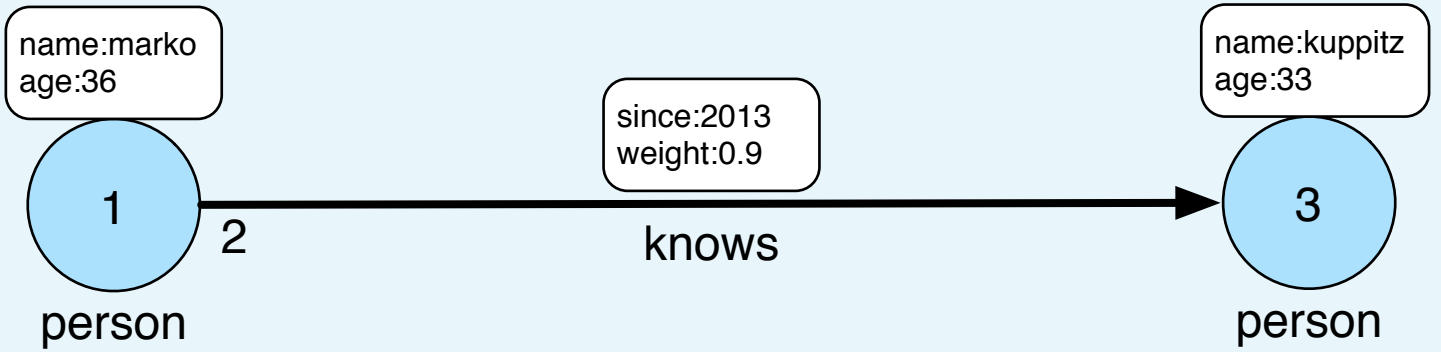
$\lambda(e_2, \texttt{label}) \mapsto \texttt{knows}$

$((e_2)_1, (e_2)_2) = (v_1, v_3)$



The Graph

~~Directed, binary, attributed multi-graph.~~



Property graph.

* Multi- and meta-properties are not discussed in this presentation nor in the associated conference article.
http://tinkerpop.incubator.apache.org/docs/3.0.2-incubating/#vertex-properties

```
~/tinkerpop3$ bin/gremlin.sh
```

```
gremlin>
```

```
~/tinkerpop3$ bin/gremlin.sh

         \,,,/
         (o o)
-----oOOo-(3)-oOOo-----
plugin activated: tinkerpop.server
plugin activated: tinkerpop.utilities
plugin activated: tinkerpop.tinkergraph
gremlin>
```

```
~/tinkerpop3$ bin/gremlin.sh

         \,,,/
         (o o)
-----oOOo-(3)-oOOo-----
plugin activated: tinkerpop.server
plugin activated: tinkerpop.utilities
plugin activated: tinkerpop.tinkergraph
gremlin> graph = TinkerGraph.open()
==>tinkergraph[vertices:0 edges:0]
gremlin>
```

"TinkerGraph is an in-memory graph system provided by TinkerPop."

```
~/tinkerpop3$ bin/gremlin.sh

         \,,,/
         (o o)
-----oOOo-(3)-oOOo-----
plugin activated: tinkerpop.server
plugin activated: tinkerpop.utilities
plugin activated: tinkerpop.tinkergraph
gremlin> graph = TinkerGraph.open()
==>tinkergraph[vertices:0 edges:0]
gremlin>
```

TitanGraph.open(…)

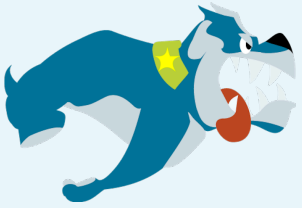
OrientGraph.open(…)

HadoopGraph.open(…)

DSEGraph.open(…)

HadoopGraph.
compute(GiraphGraphComputer)

Neo4jGraph.open(…)



StardogGraph.open(…)

HadoopGraph.
compute(SparkGraphComputer)

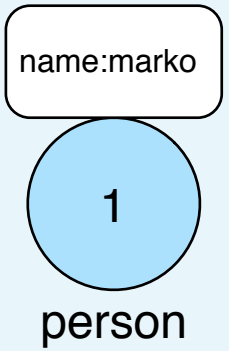
etc...

"Gremlin is agnostic to the underlying graph system."

```
~/tinkerpop3$ bin/gremlin.sh

         \,,,/
         (o o)
-----oOOo-(3)-oOOo-----
plugin activated: tinkerpop.server
plugin activated: tinkerpop.utilities
plugin activated: tinkerpop.tinkergraph
gremlin> graph = TinkerGraph.open()
==>tinkergraph[vertices:0 edges:0]
gremlin> v1 = graph.addVertex(id,1,label,'person')
==>v[1]
gremlin>
```

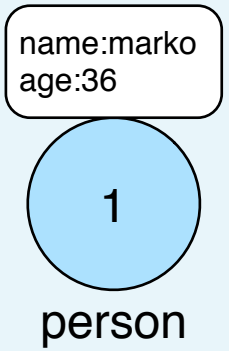
"A graph system provider is Gremlin-enabled if they implement the `gremlin.structure` API suite."

```
~/tinkerpop3$ bin/gremlin.sh

         \,,,/
         (o o)
-----oOOo-(3)-oOOo-----
plugin activated: tinkerpop.server
plugin activated: tinkerpop.utilities
plugin activated: tinkerpop.tinkergraph
gremlin> graph = TinkerGraph.open()
==>tinkergraph[vertices:0 edges:0]
gremlin> v1 = graph.addVertex(id,1,label,'person')
==>v[1]
gremlin> v1.property('name','marko')
==>vp[name->marko]
gremlin>
```

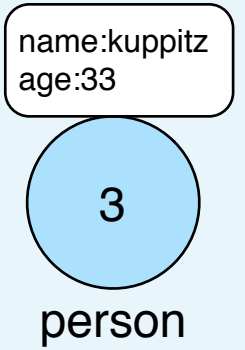


"The APIs have to do with CRUD operations on a graph structure."

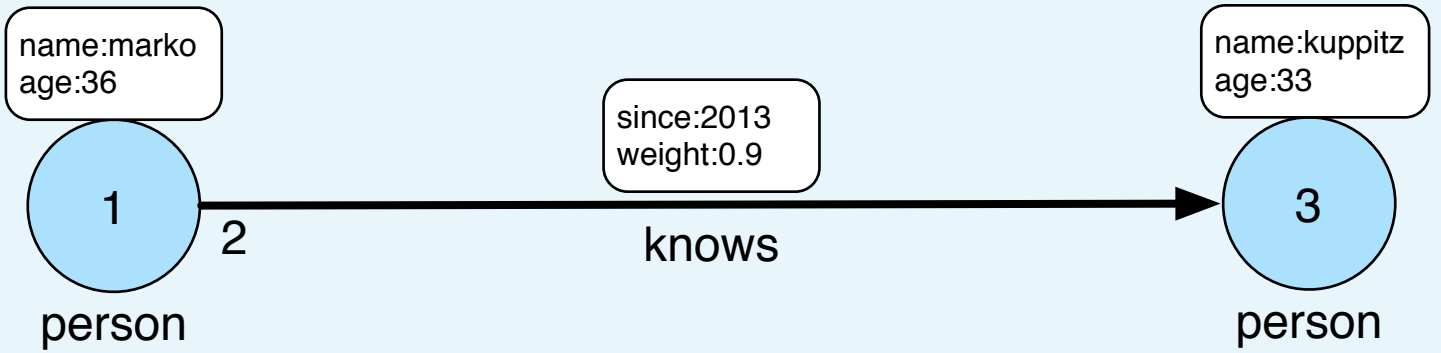
```
~/tinkerpop3$ bin/gremlin.sh

         \,,,/
         (o o)
-----oOOo-(3)-oOOo-----
plugin activated: tinkerpop.server
plugin activated: tinkerpop.utilities
plugin activated: tinkerpop.tinkergraph
gremlin> graph = TinkerGraph.open()
==>tinkergraph[vertices:0 edges:0]
gremlin> v1 = graph.addVertex(id,1,label,'person')
==>v[1]
gremlin> v1.property('name','marko')
==>vp[name->marko]
gremlin> v1.property('age',36)
==>vp[age->36]
gremlin>
```



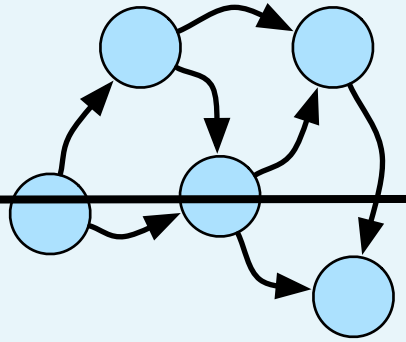
```
~/tinkerpop3$ bin/gremlin.sh

         \,,,/
         (o o)
-----oOOo-(3)-oOOo-----
plugin activated: tinkerpop.server
plugin activated: tinkerpop.utilities
plugin activated: tinkerpop.tinkergraph
gremlin> graph = TinkerGraph.open()
==>tinkergraph[vertices:0 edges:0]
gremlin> v1 = graph.addVertex(id,1,label,'person')
==>v[1]
gremlin> v1.property('name','marko')
==>vp[name->marko]
gremlin> v1.property('age',36)
==>vp[age->36]
gremlin> v3 = graph.addVertex(id,3,label,'person','name','kuppitz','age',33)
==>v[3]
gremlin>
```



```
~/tinkerpop3$ bin/gremlin.sh

         \,,,/
         (o o)
-----oOOo-(3)-oOOo-----
plugin activated: tinkerpop.server
plugin activated: tinkerpop.utilities
plugin activated: tinkerpop.tinkergraph
gremlin> graph = TinkerGraph.open()
==>tinkergraph[vertices:0 edges:0]
gremlin> v1 = graph.addVertex(id,1,label,'person')
==>v[1]
gremlin> v1.property('name','marko')
==>vp[name->marko]
gremlin> v1.property('age',36)
==>vp[age->36]
gremlin> v3 = graph.addVertex(id,3,label,'person','name','kuppitz','age',33)
==>v[3]
gremlin> v1.addEdge('knows',v3,id,2,'since',2013,'weight',0.9)
==>e[2][1-knows->3]
gremlin>
```



The Machine Components



The Graph

The Data

The Traverser

The CPU

The Traversal

The Program

The Traverser

"The traverser is a 'CPU core'."

The Traverser

$t \in T$

"The traverser *t* is in a set of traversers *T*."

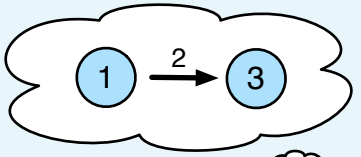
The Traverser

$t \in T$

$\mu(t) \mapsto v_3$

"The traverser's current graph location is vertex 3."

The Traverser



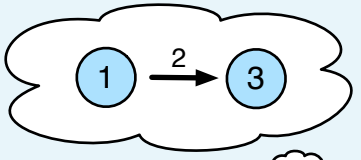
$\Delta(t) \mapsto ((\emptyset, v_1), (\emptyset, e_2), (\emptyset, v_3))$

$t \in T$

$\mu(t) \mapsto v_3$

"The traverser *may* remember his history in the graph."

The Traverser



$\Delta(t) \mapsto ((\emptyset, v_1), (\emptyset, e_2), (\emptyset, v_3))$

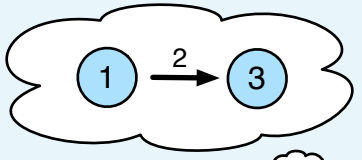
$t \in T$

$\mu(t) \mapsto v_3$

$\iota(t) \mapsto 0$

"The traverser records how many times he has been through a loop sequence."

The Traverser



$\Delta(t) \mapsto ((\emptyset, v_1), (\emptyset, e_2), (\emptyset, v_3))$

$t \in T$

$\beta(t) \mapsto 1$

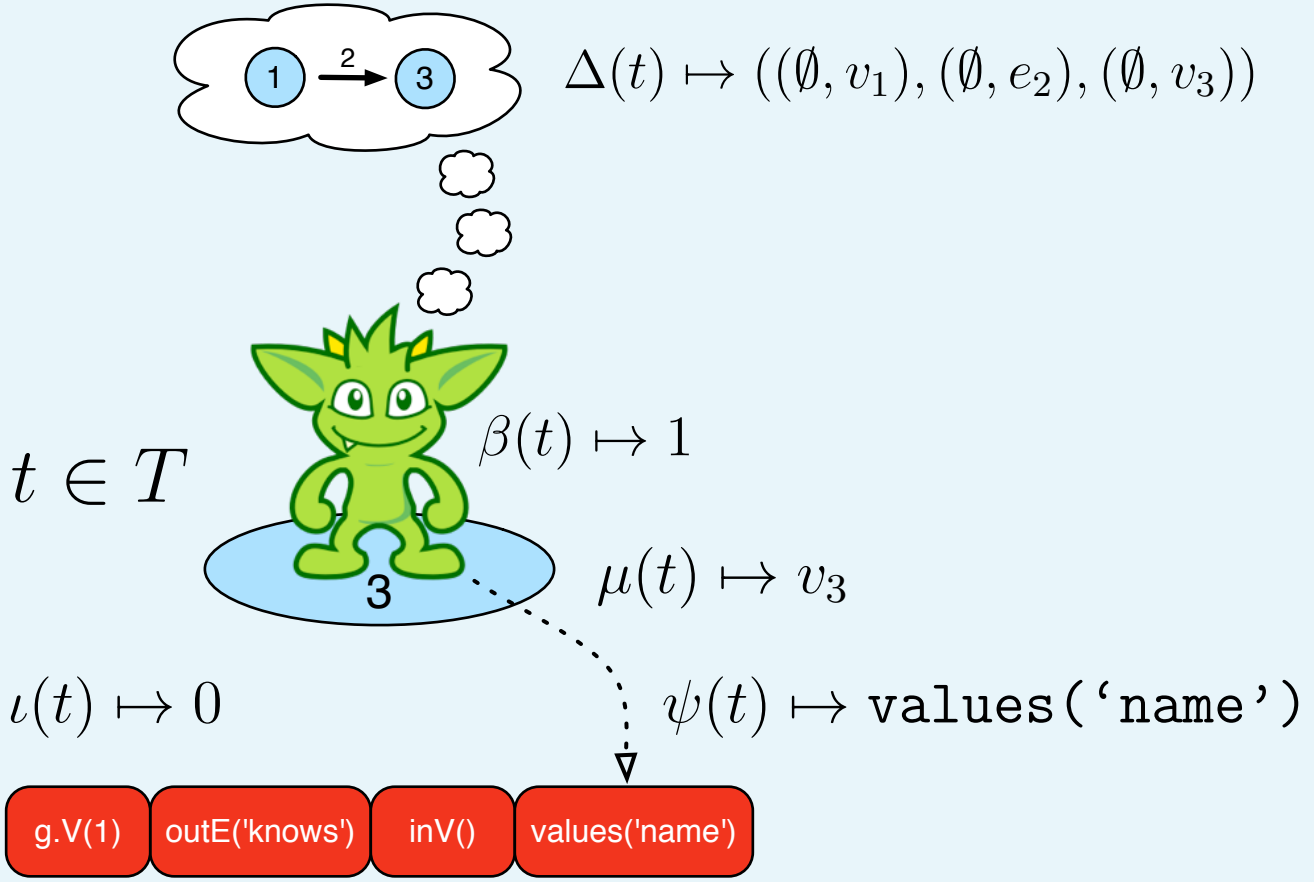
$\mu(t) \mapsto v_3$

$\iota(t) \mapsto 0$

"The traverser can represent more than one traverser."

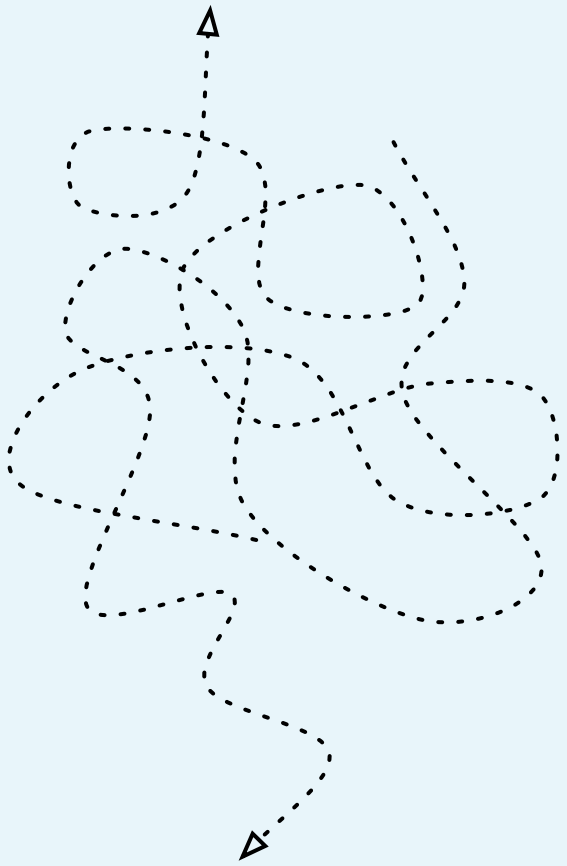
It's what you would do if you thought in terms of "energy flows. "

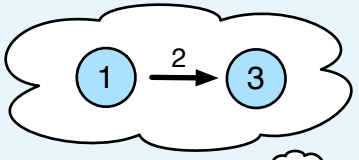
The Traverser



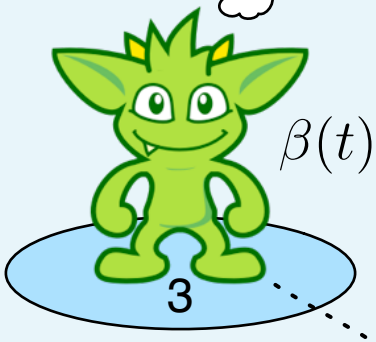
"The traverser has a reference to his location in the traversal (program counter)."

The Traverser





$\Delta(t) \mapsto ((\emptyset, v_1), (\emptyset, e_2), (\emptyset, v_3))$



$t \in T$

$\beta(t) \mapsto 1$

$\mu(t) \mapsto v_3$

$\iota(t) \mapsto 0$

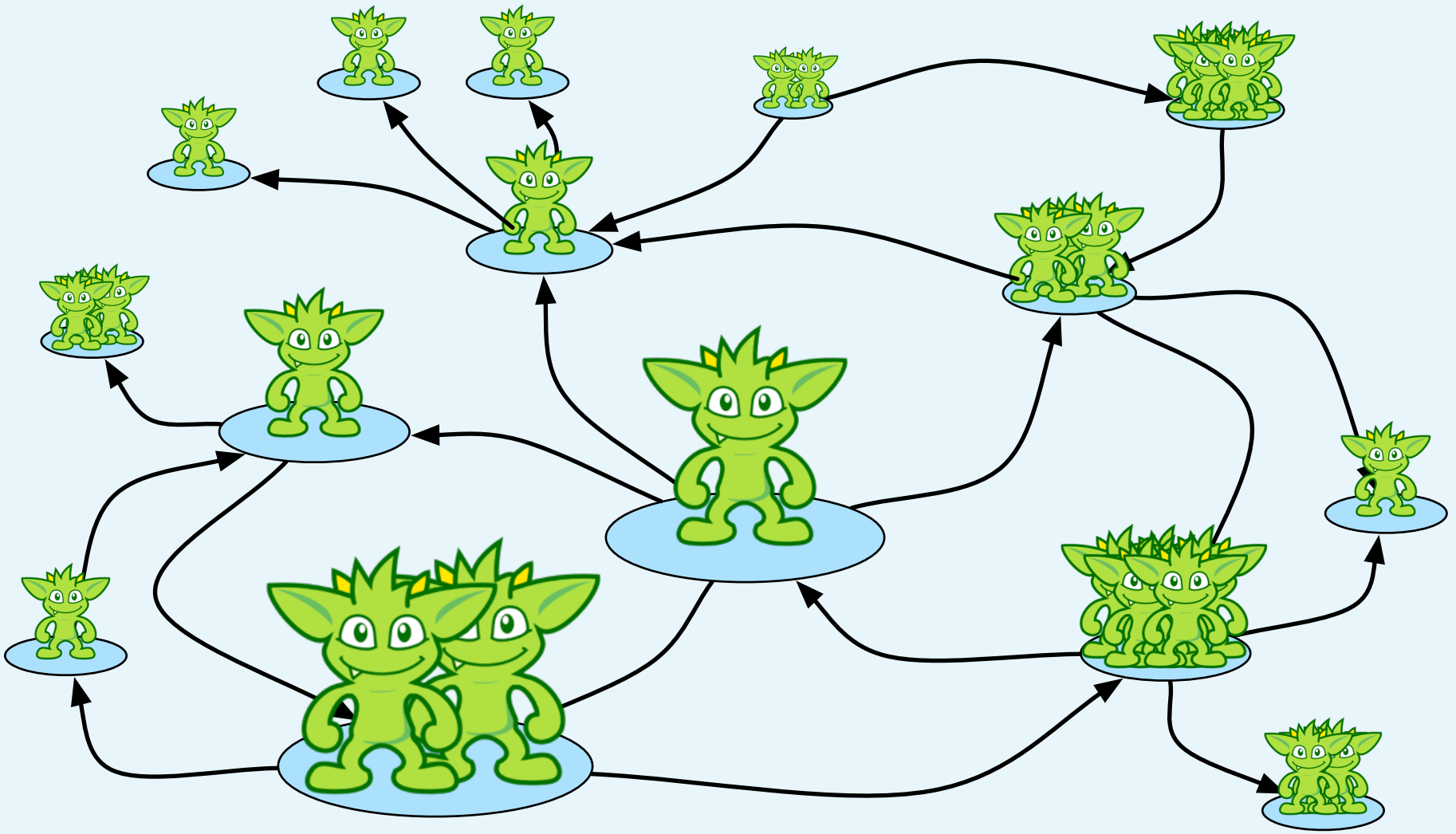
$\psi(t) \mapsto$ `values('name')`

g.V(1) outE('knows') inV() values('name')

$T \subseteq \left(U \times \Psi \times (\mathcal{P}(\Sigma^*) \times U)^* \times \mathbb{N}^+ \times U \times \mathbb{N}^+\right)$

"Gremlin sack" (not discussed here)

The Traverser



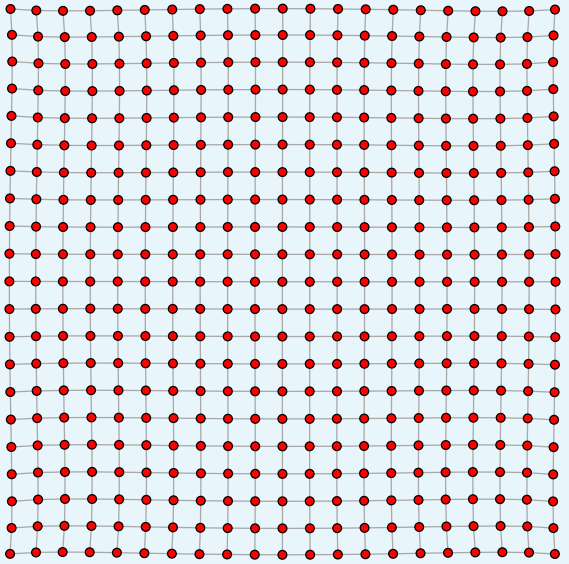
Combinatoric explosion

"Numerous (many many many) traversers are spawned during a traversal (computation)."

many

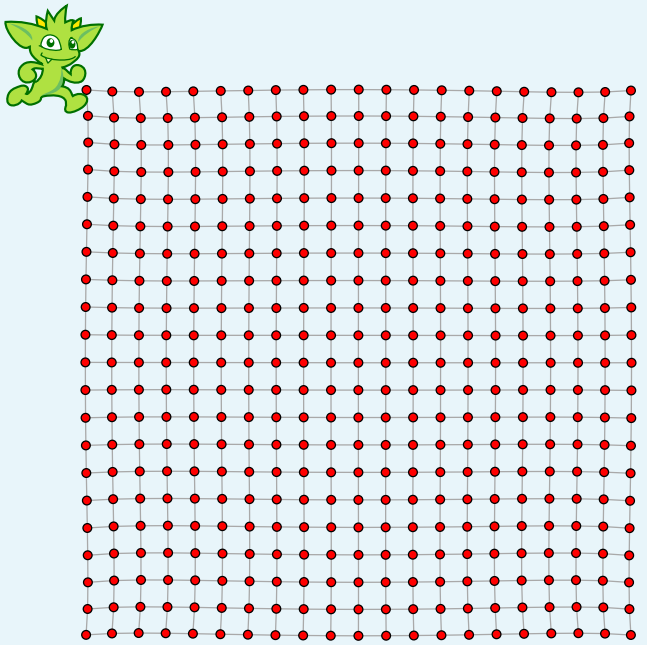
many

The Traverser



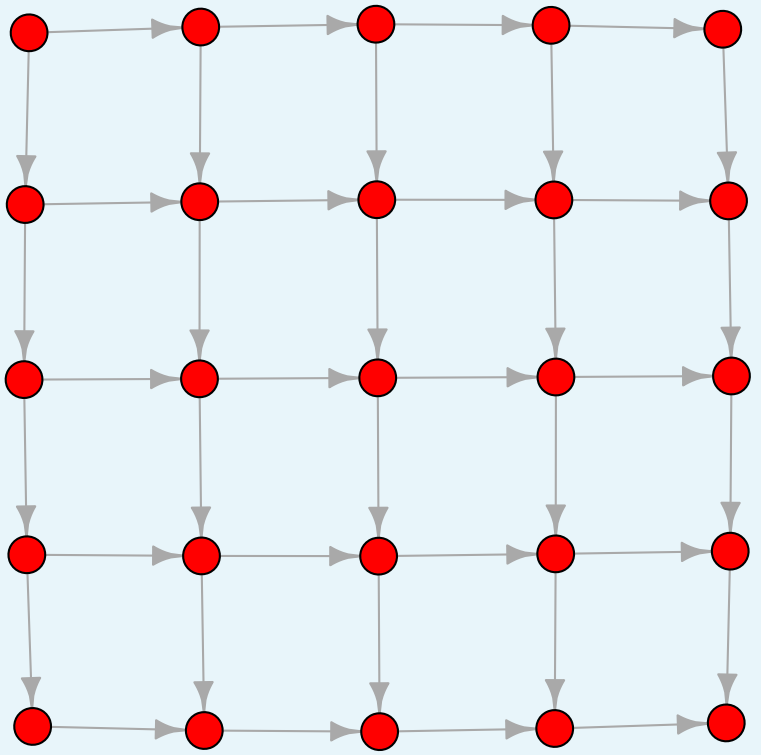
"A 21x21 vertex lattice -- 20x20 edges."

The Traverser



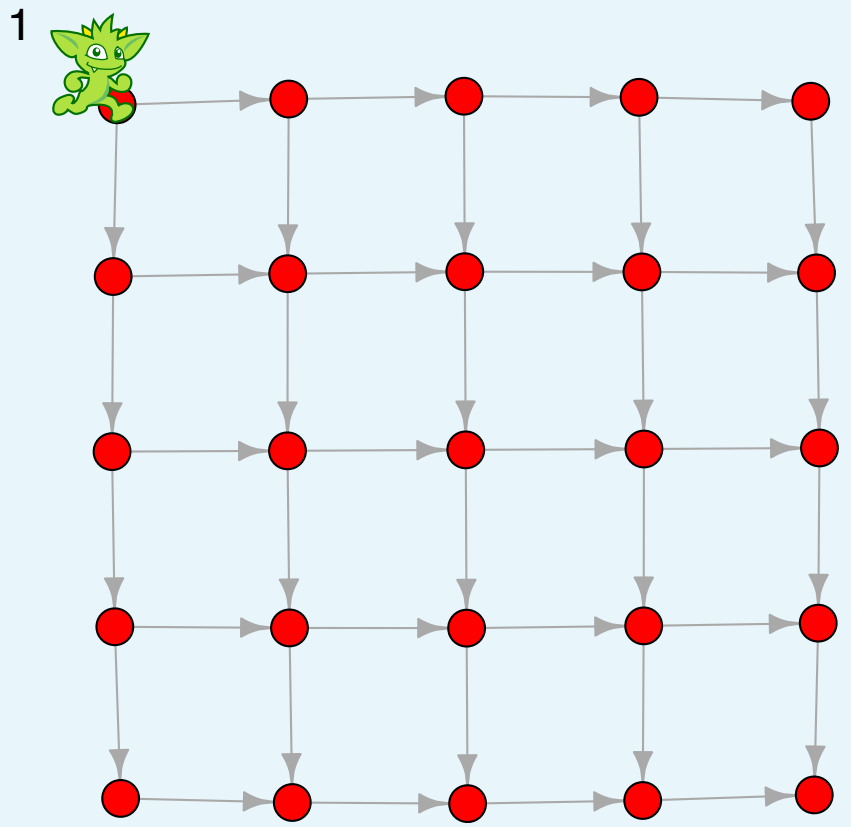
"A Gremlin starts at (0,0) and can only go right and down to reach (20,20).
How many traversers (paths) ultimately reach (20,20)?"

The Traverser



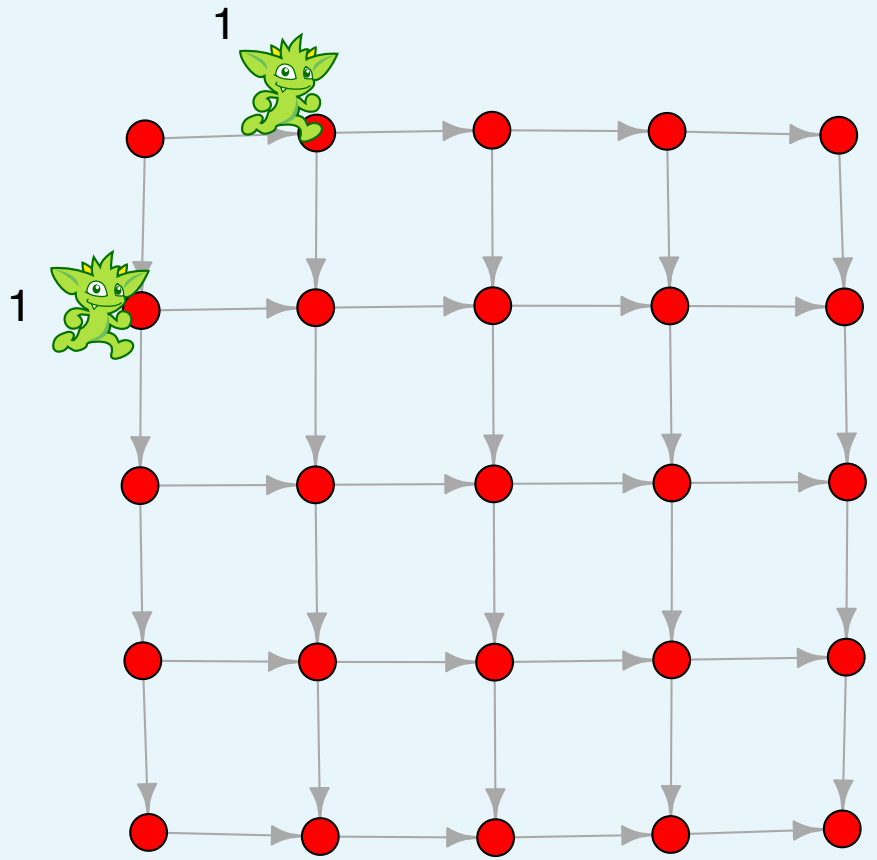
"For the sake of demonstration, lets look at a 5x5 lattice (4x4 edges)."

The Traverser



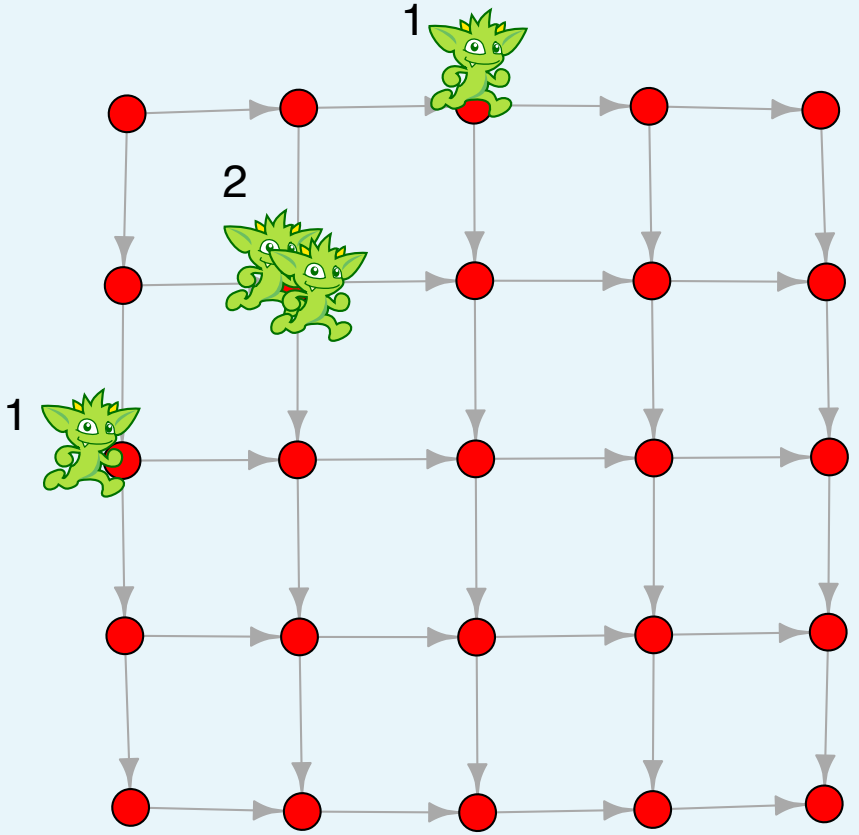
"Total = 1"

The Traverser



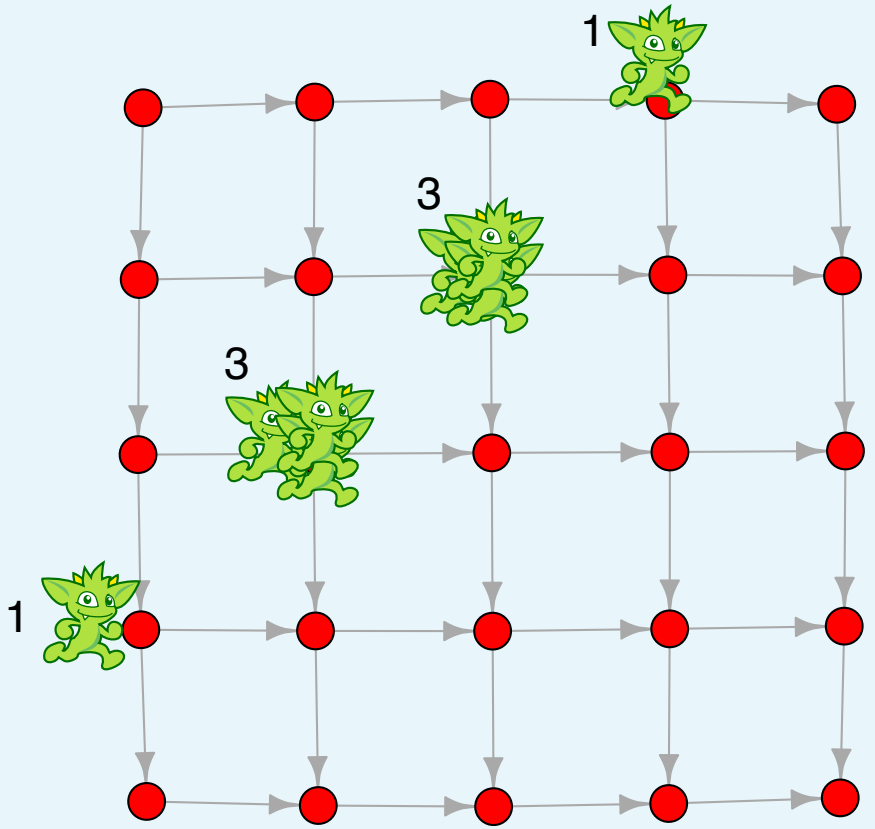
"Total = 2"

The Traverser



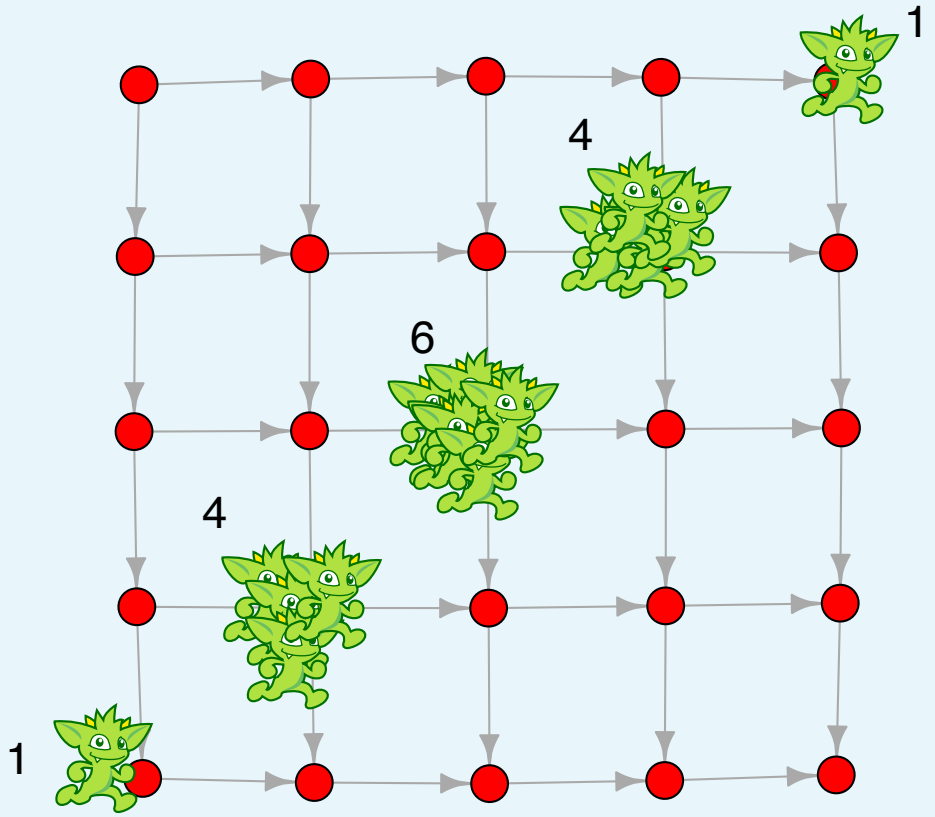
"Total = 4"

The Traverser



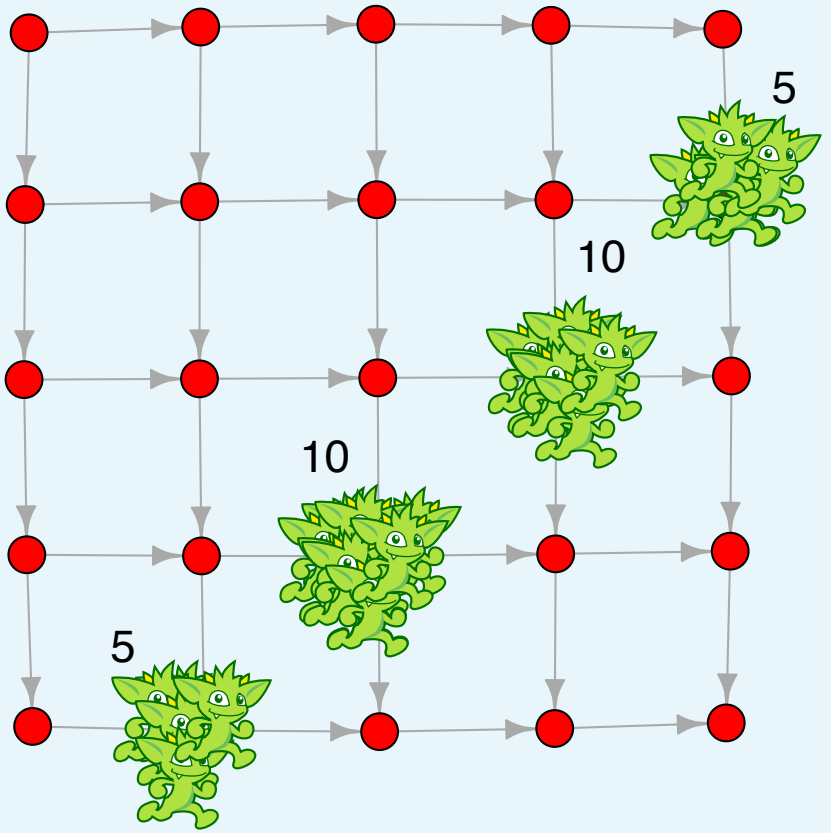
"Total = 8"

The Traverser



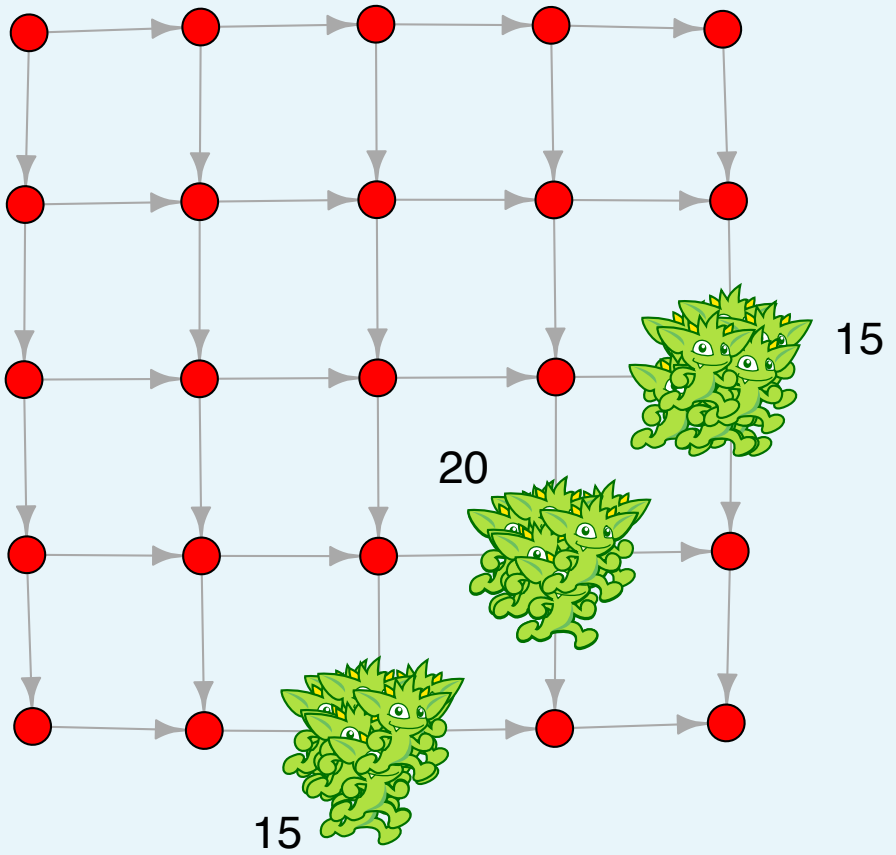
"Total = 16"

The Traverser



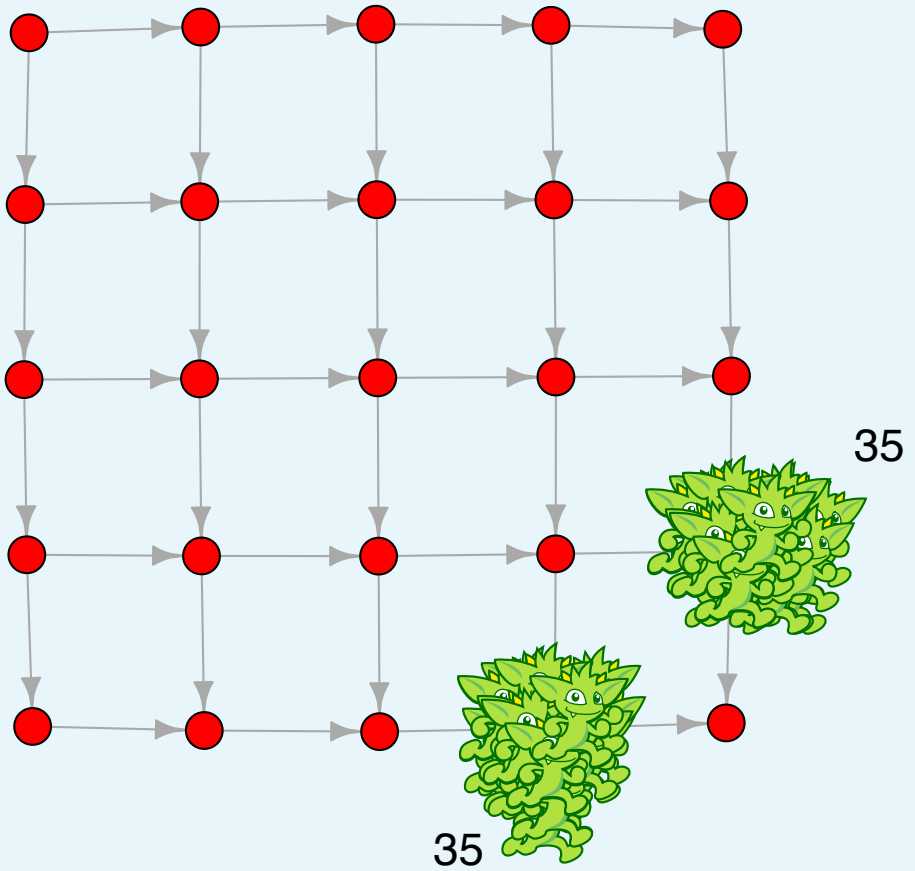
"Total = 20"

The Traverser



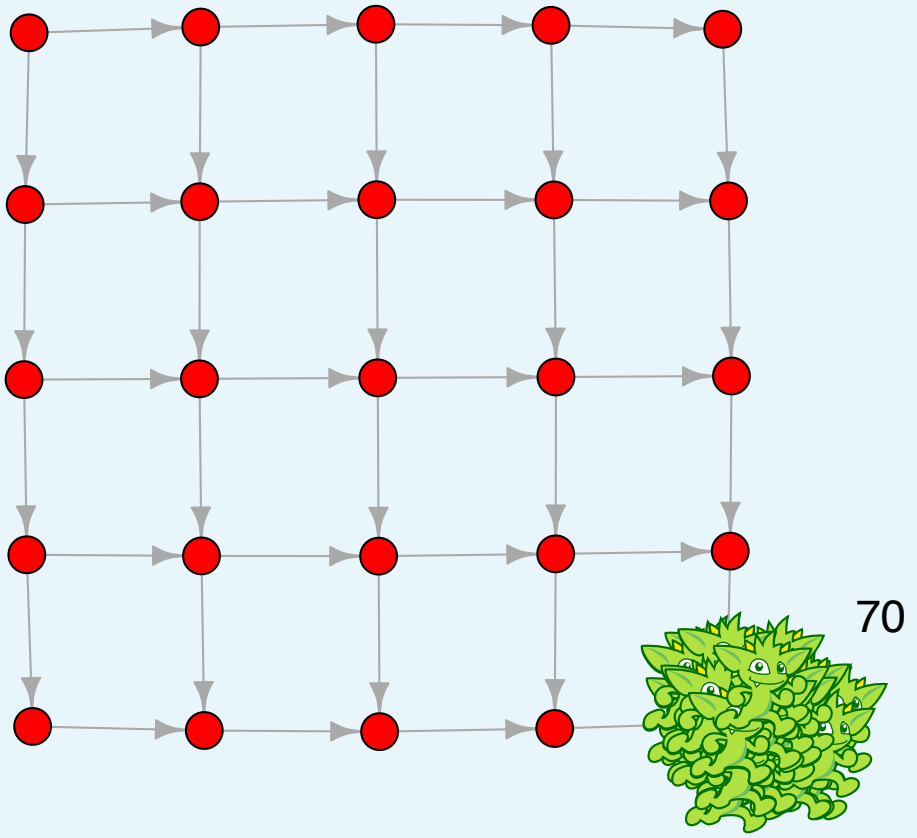
"Total = 50"

The Traverser



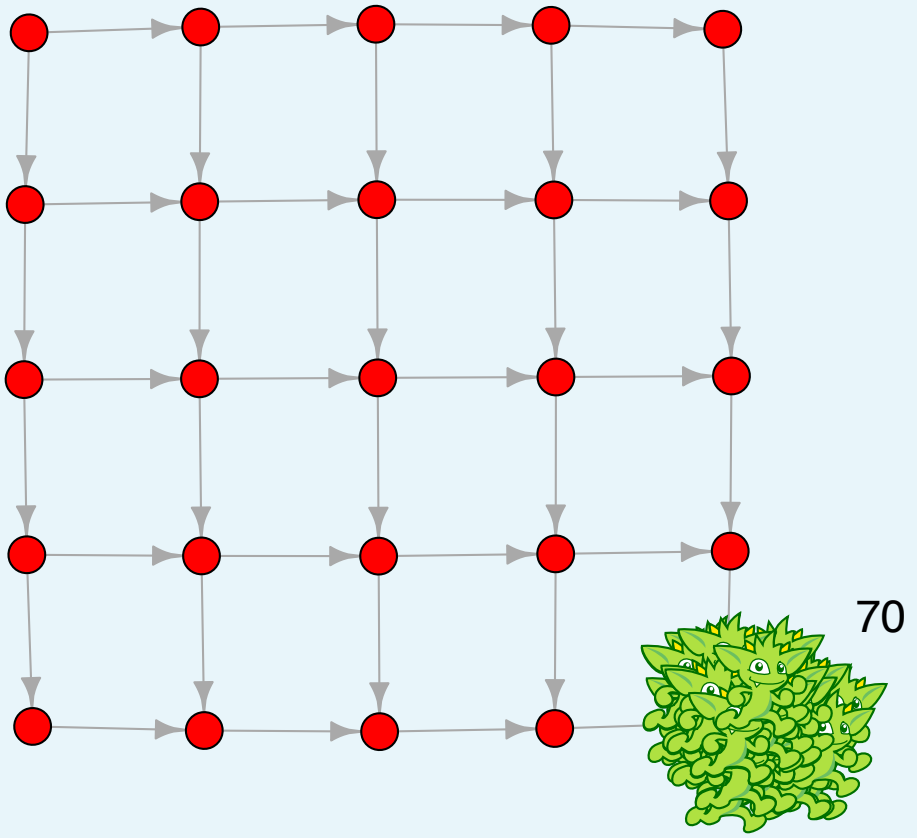
"Total = 70"

The Traverser



"Total = 70"

The Traverser

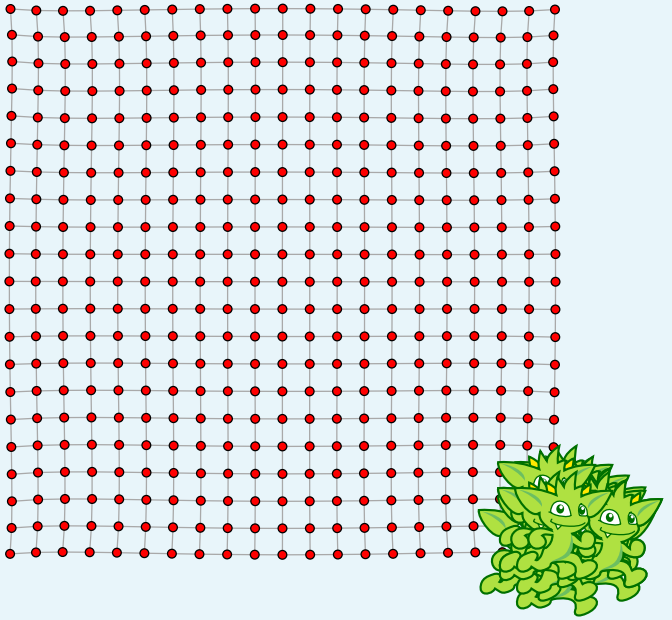


8 choose 4 = 70

$$\binom{2n}{n} = \frac{(2n)!}{(n!)^2} = \frac{(2n)(2n-1)(2n-2)\dots 1}{(n(n-1)(n-2)\dots 1)^2}$$

n = number of edges on a side

The Traverser

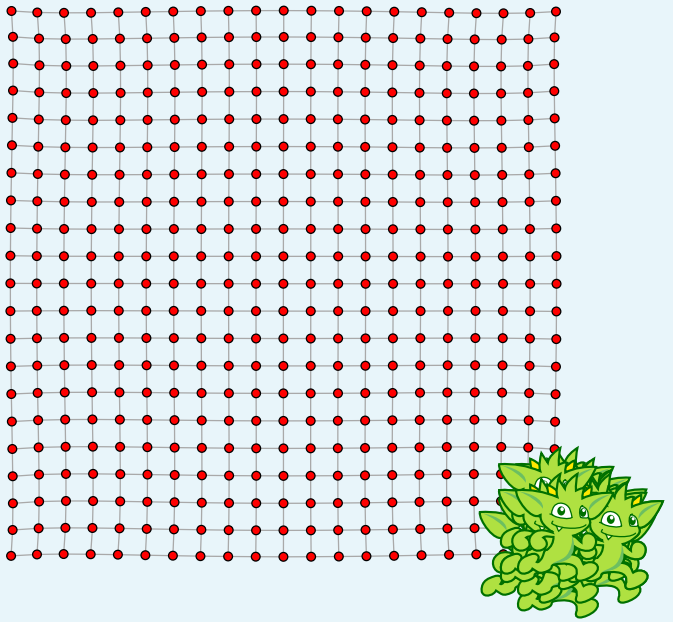


40 choose 20

$$\binom{2n}{n} = \frac{(2n)!}{(n!)^2} = \frac{(2n)(2n-1)(2n-2)\dots 1}{(n(n-1)(n-2)\dots 1)^2}$$

"Analytically, ~137 billion traversers will exist at (20,20)."

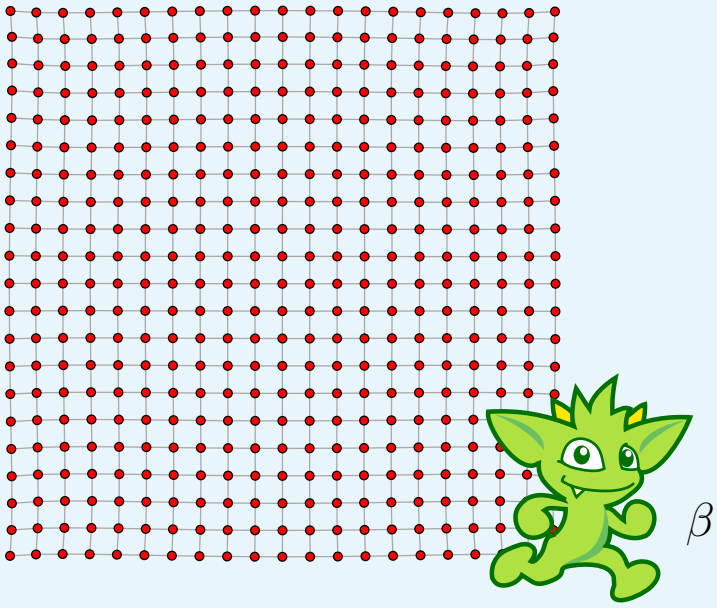
The Traverser



```
gremlin> g.V(0).repeat(out('down','right')).times(40).count()
==>137846528820
gremlin>
```

"That was fast… The Gremlin machine spawned 137 billion traversers?!"

The Traverser



$\beta(t) \mapsto 137846528820$

```
gremlin> g.V(0).repeat(out('down','right')).times(40).count()
==>137846528820
gremlin> clock(100){g.V(0).repeat(out('down','right')).times(40).count().next()}
==>0.8890942
```

less than a millisecond

Bulking is a crucial component of OLAP when near trillion edge graphs are moving traversers between machines.

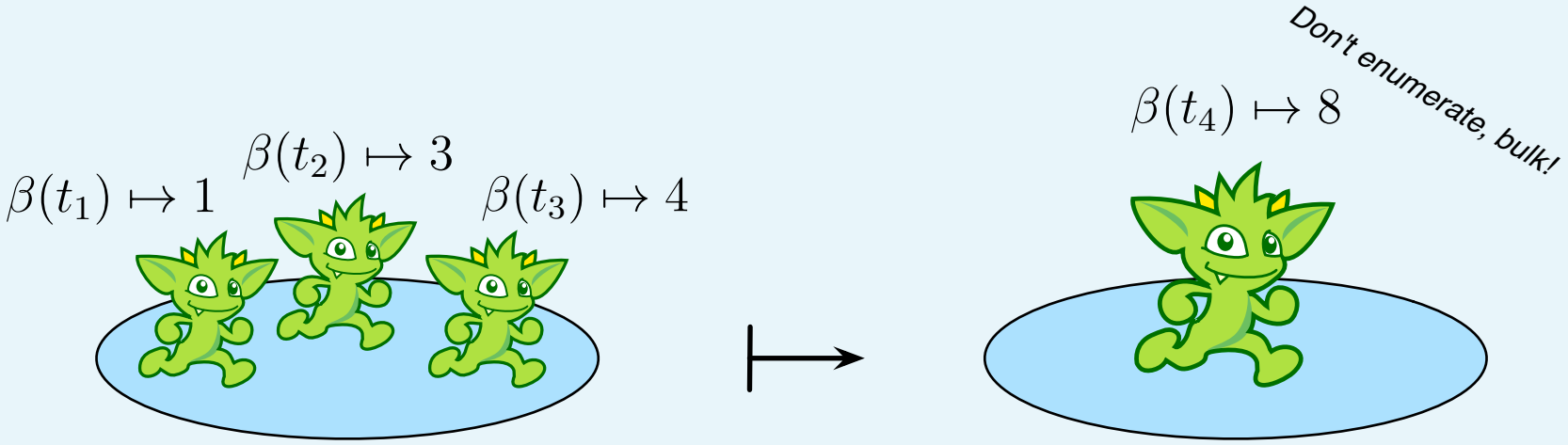
"No. At vertex 400 (20,20), there exists 1 traverser with a bulk of ~137 billion."

The Traverser

Traverser Equivalence Class

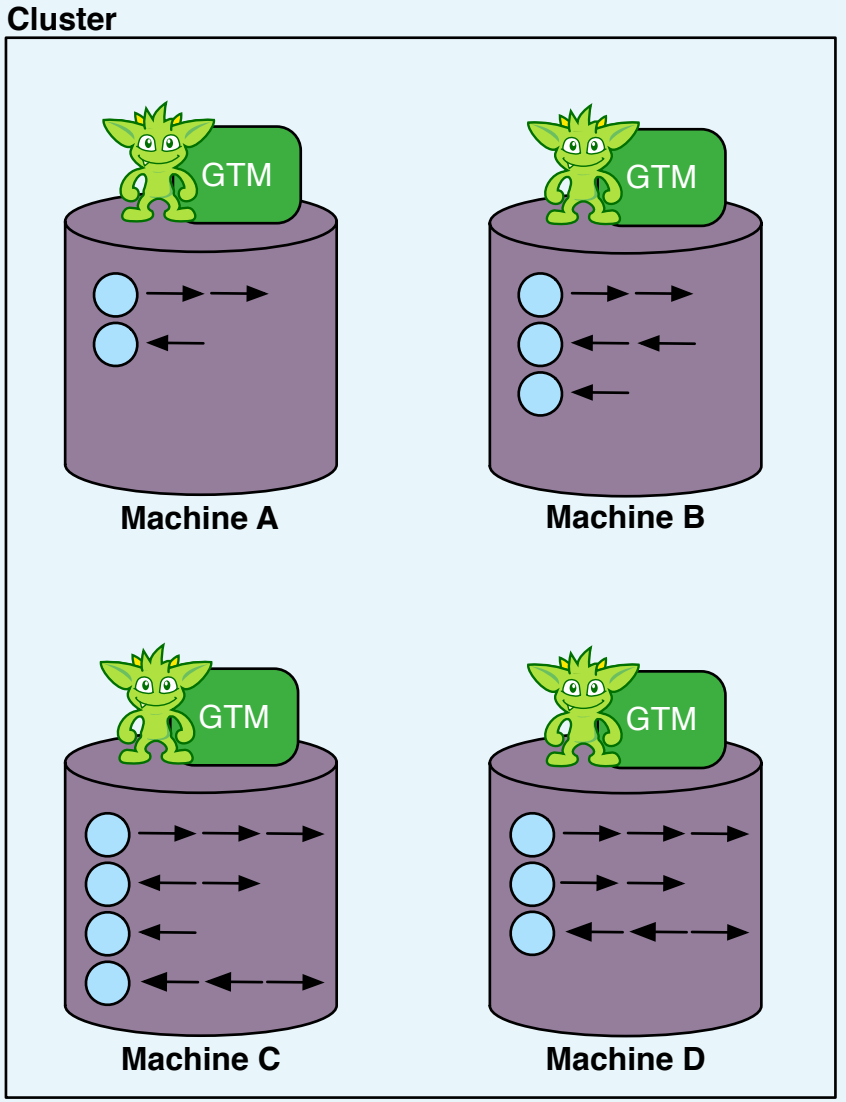
$$[t] = \{t' \in T \mid \begin{array}{rcll} \mu(t) & = & \mu(t') & \wedge \\ \psi(t) & = & \psi(t') & \wedge \\ \Delta(t) & = & \Delta(t') & \wedge \\ \iota(t) & = & \iota(t') & \}. \end{array}$$

graph location	*data location*
traversal location	*program counter*
path history	*MAY NOT BE REQUIRED*
same loop counter	*recursion of program counter*



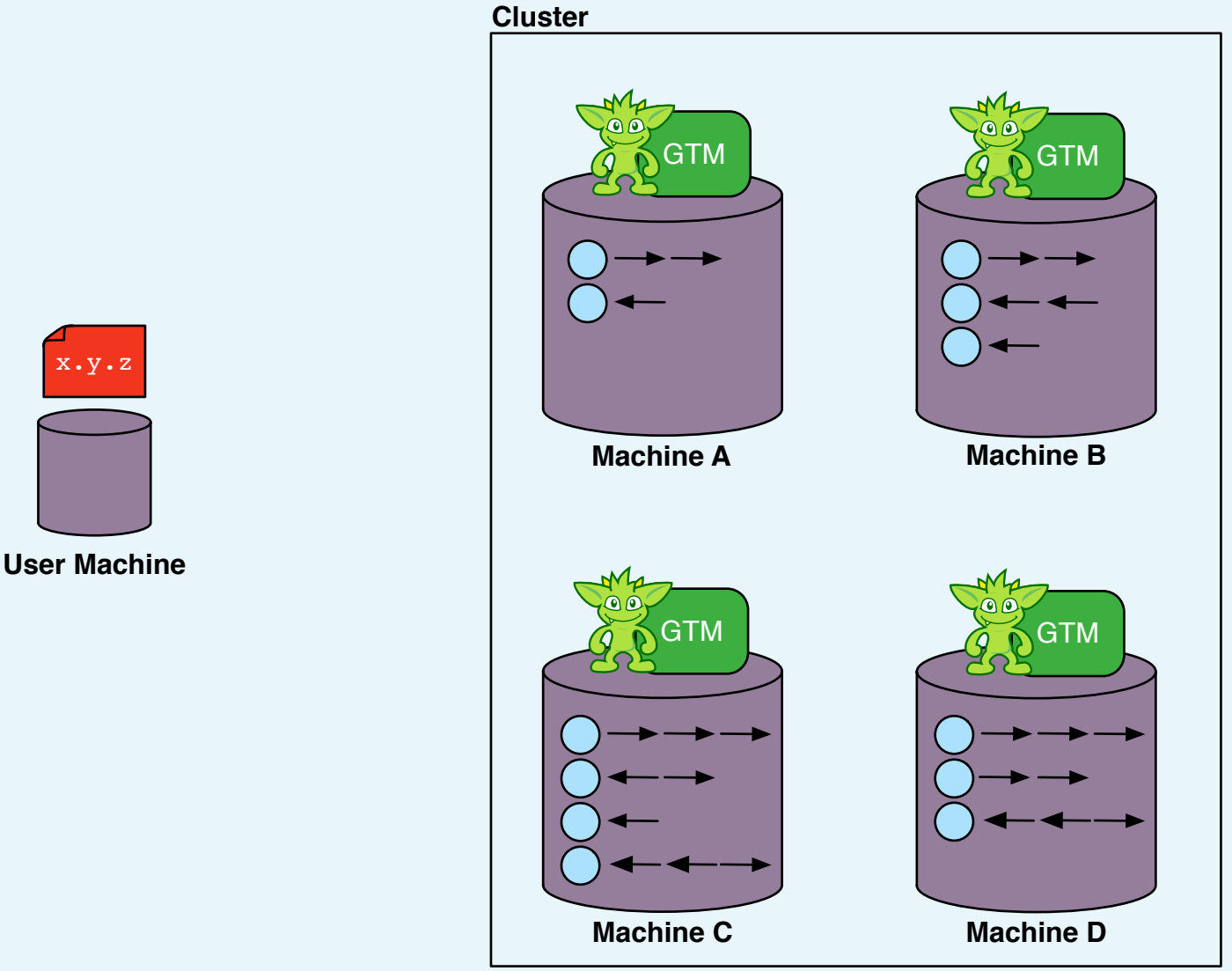
"If 2 traversers (each with bulk 1) are at the same location in the graph and have similar other properties, make 1 traverser with a bulk of 2."

The Traverser



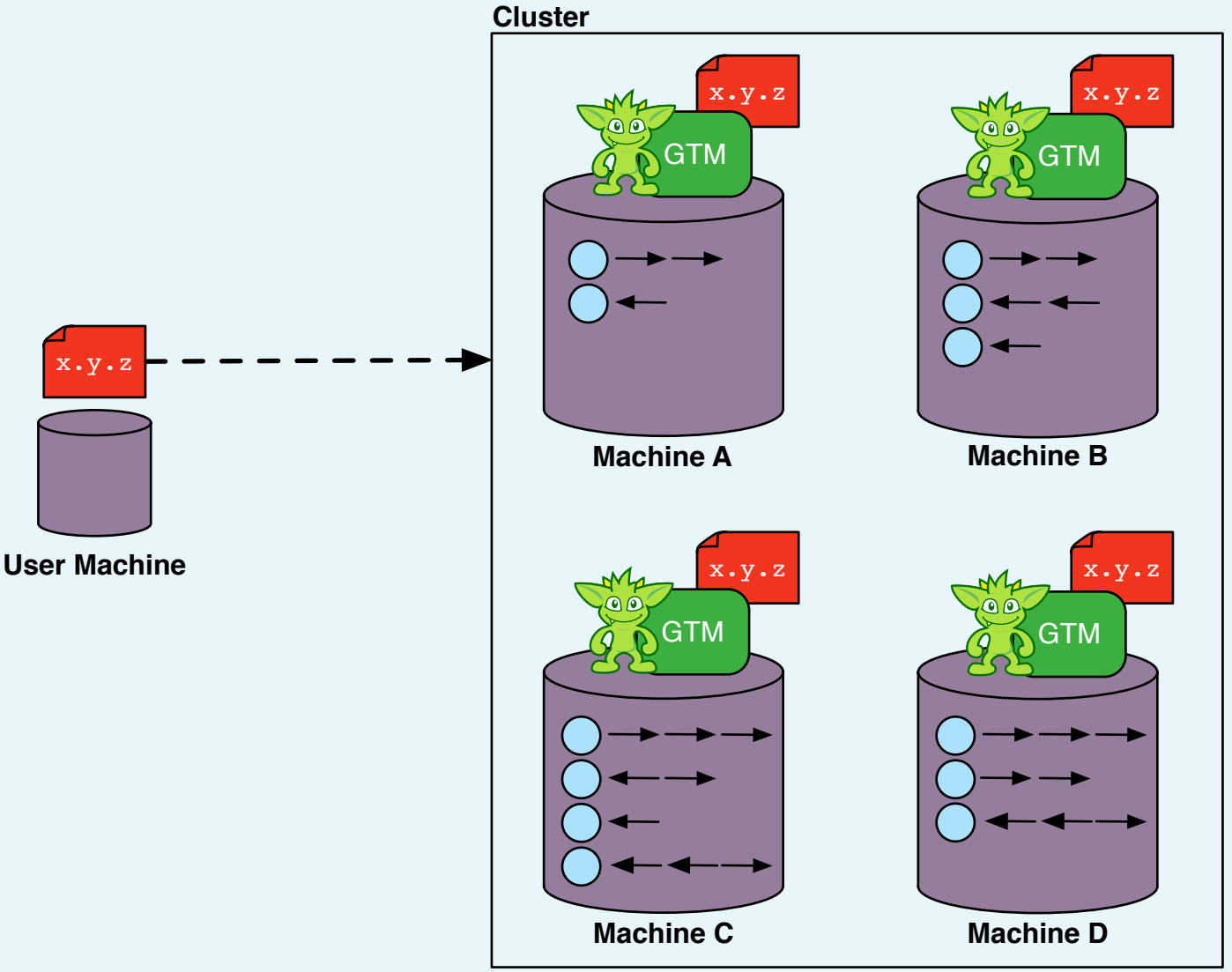
"In Gremlin OLAP, the graph is represented across a distributed system (e.g. Hadoop) as a partitioned adjacency list."

The Traverser



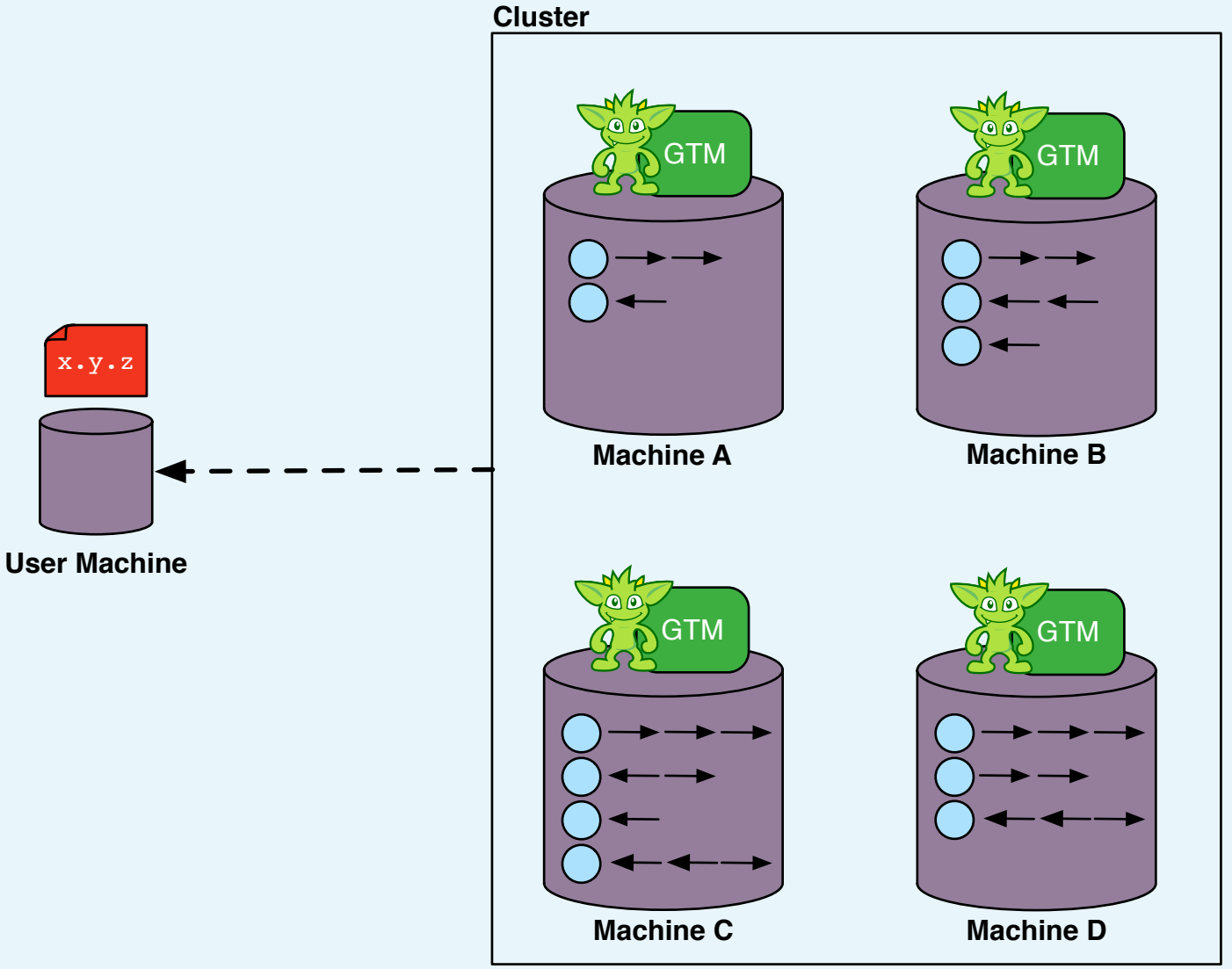
"The user creates a traversal (compiled from any language)."

The Traverser



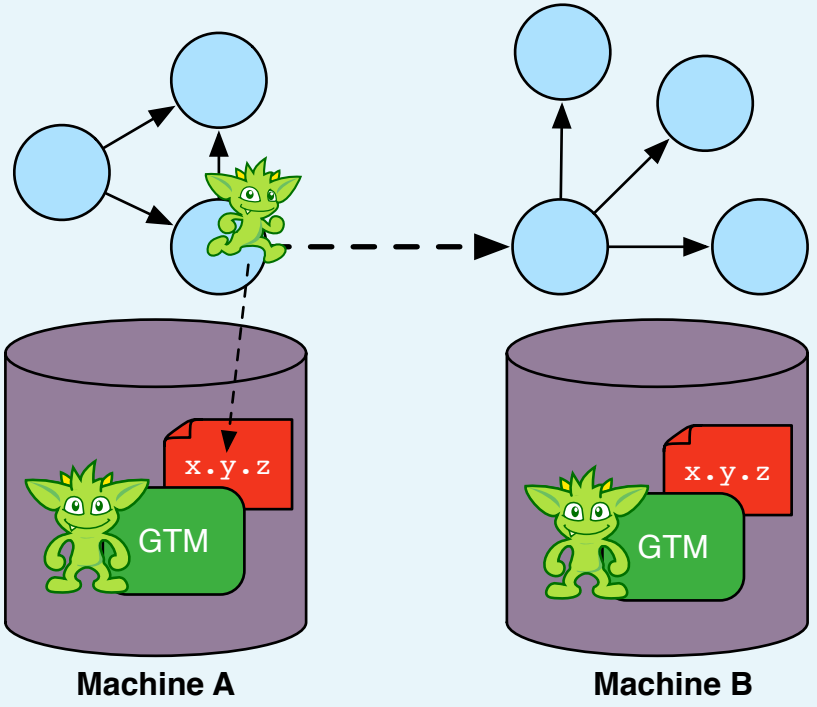
"The compiled traversal is sent to every machine in the cluster which has a piece of the graph and a Gremlin traversal machine."

The Traverser



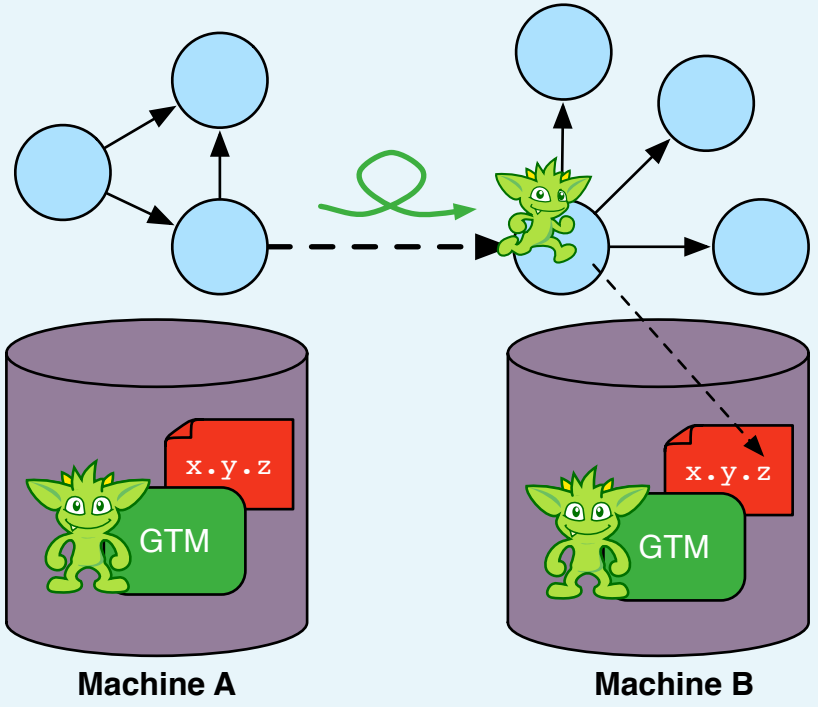
"When the graph computation is complete, a reference to the result is returned. For instance, in HadoopGremlin, HDFS stores both the graph and sideEffect data."

The Traverser



"The OLAP algorithm is the classic Bulk Synchronous Parallel algorithm popularized by Google Pregel and Apache Giraph."

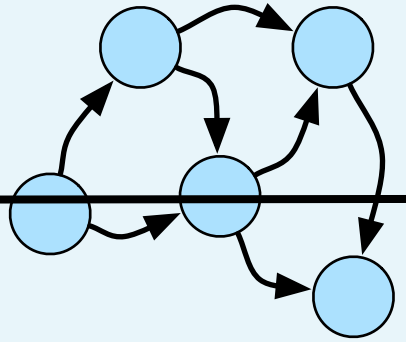
The Traverser



"The messages are traversers. Given the traverser equivalence class [t], they can be bulked."

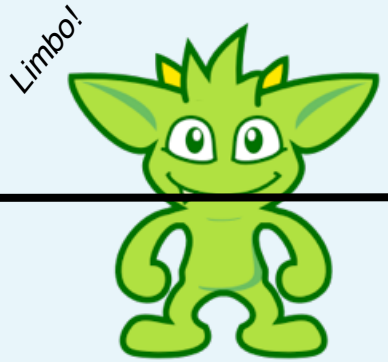
"Its just "complex energy."

The Machine Components



~~The Graph~~

~~The Data~~

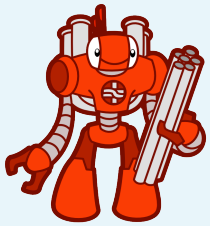


~~The Traverser~~

~~The CPU~~

The Traversal

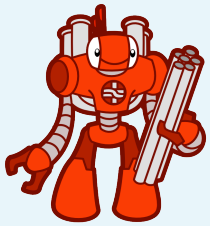
The Program



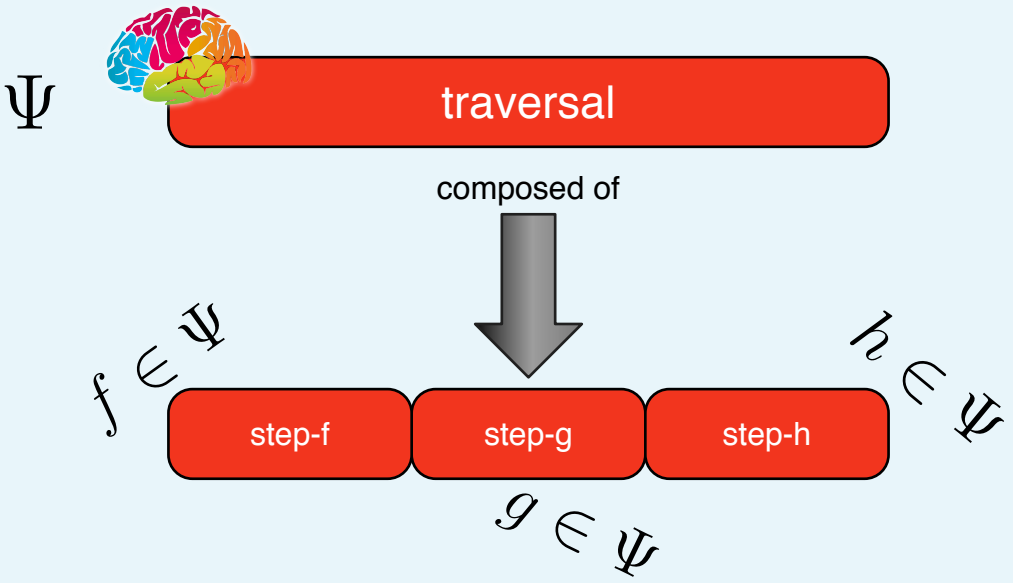
The Traversal

Ψ  traversal

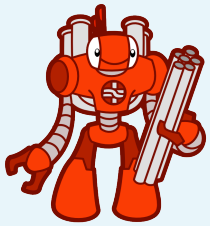
"The traversal is the 'software program'."



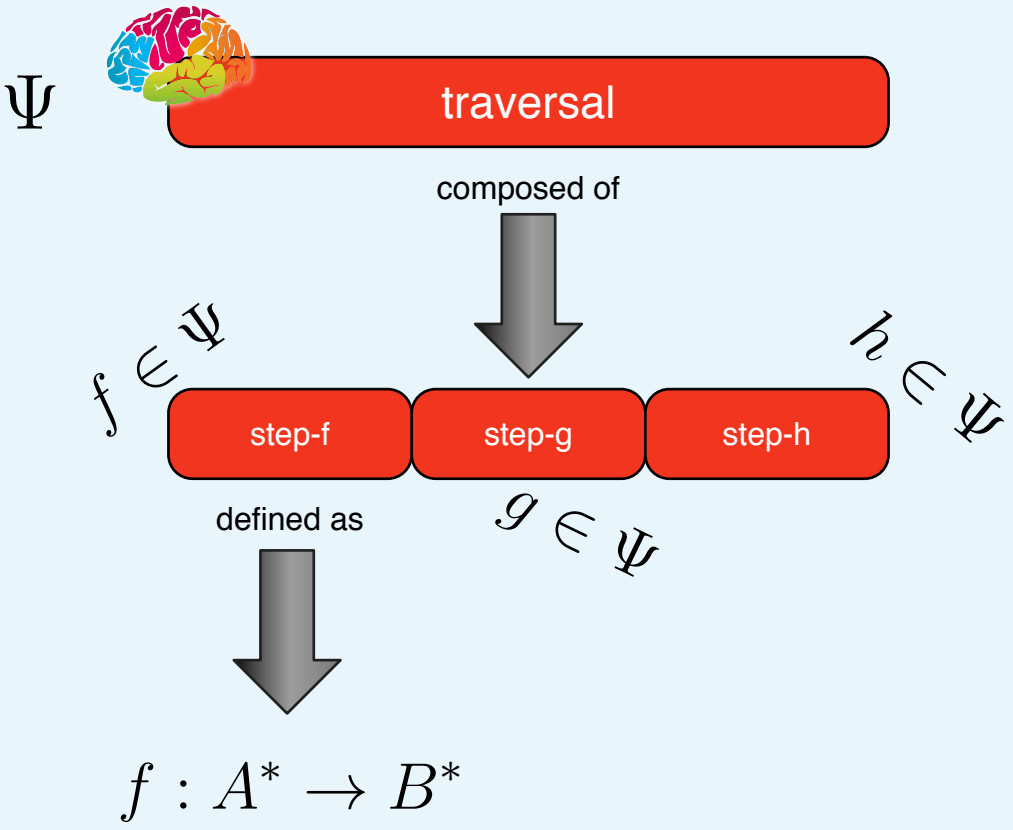
The Traversal



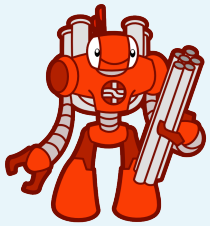
"The steps are the 'instructions'."



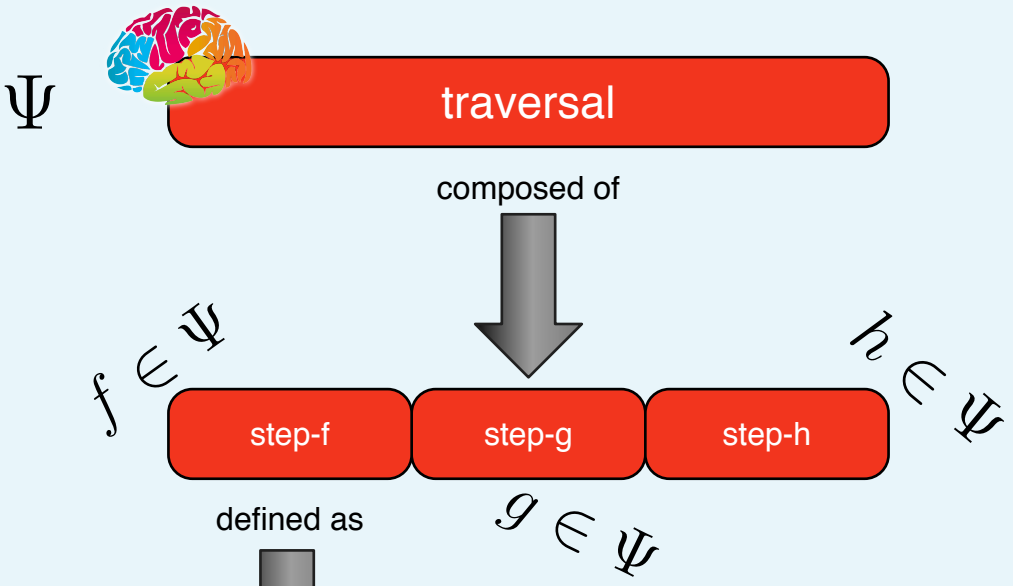
The Traversal



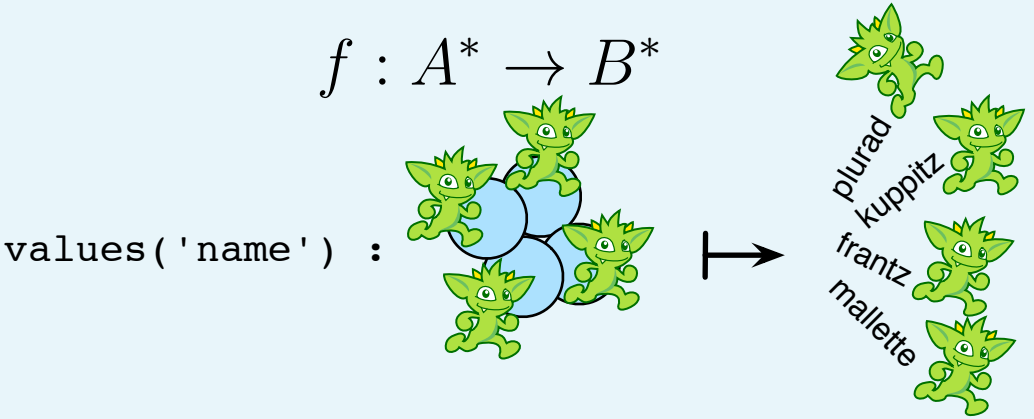
"Step *f* maps a stream (multi-set) of traversers located at objects of type *A* to a stream (multi-set) of traversers located at objects of type *B*."



The Traversal



$$f : A^* \rightarrow B^*$$



"Step *values('name')* maps a stream (multi-set) of traversers located at vertices to a stream (multi-set) of traversers located at strings."

The Traversal

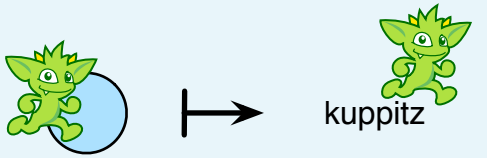
$\texttt{map} : A^* \rightarrow B^*$

`values('name') :`  $\mapsto$ 

"For a traverser at *a*, move the traverser to an object at *b*."

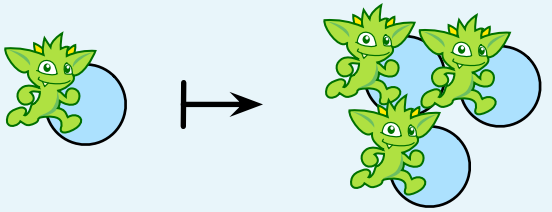
The Traversal

$\texttt{map} : A^* \rightarrow B^*$



"For a traverser at *a*, move the traverser to an object at *b*."

$\texttt{flatMap} : A^* \rightarrow B^*$



"For a traverser at *a*, clone the traverser across multiple objects in *B*."

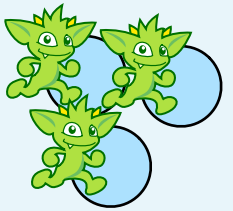
The Traversal

$\texttt{map} : A^* \rightarrow B^*$

`values('name') :`  ↦  kuppitz

"For a traverser at *a*, move the traverser to an object at *b*."

$\texttt{flatMap} : A^* \rightarrow B^*$

`out('knows') :`  ↦ 

"For a traverser at *a*, clone the traverser across multiple objects in *B*."

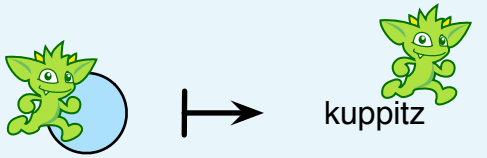
$\texttt{filter} : A^* \rightarrow A^*$

`has('age',lt(30)) :`  ↦ 

"For a traverser at *a*, either kill the traverser or leave him alone."

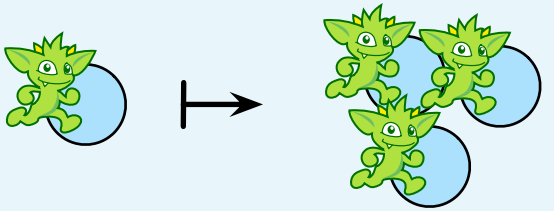
The Traversal

$\texttt{map} : A^* \rightarrow B^*$

`values('name') :` 

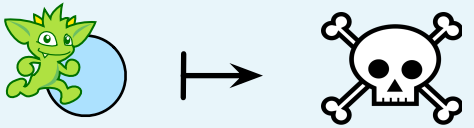
"For a traverser at *a*, move the traverser to an object at *b*."

$\texttt{flatMap} : A^* \rightarrow B^*$

`out('knows') :` 

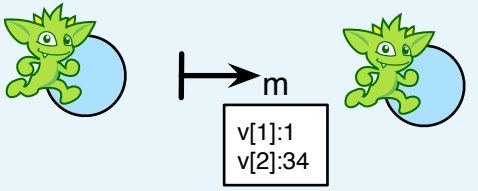
"For a traverser at *a*, clone the traverser across multiple objects in *B*."

$\texttt{filter} : A^* \rightarrow A^*$

`has('age',lt(30)) :` 

"For a traverser at *a*, either kill the traverser or leave him alone."

$\texttt{sideEffect} : A^* \rightarrow_x A^*$

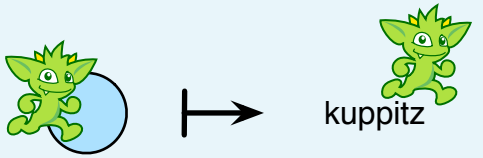
`groupCount('m') :` 

"For a traverser at *a*, leave him alone though manipulate some data structure *x*."

The Traversal

$\texttt{map} : A^* \rightarrow B^*$

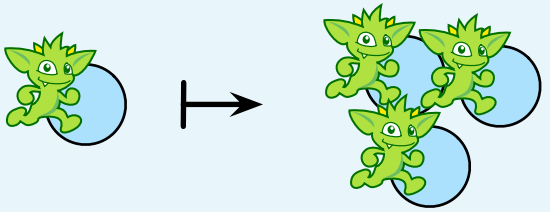
`values('name') :`



"For a traverser at *a*, move the traverser to an object at *b*."

$\texttt{flatMap} : A^* \rightarrow B^*$

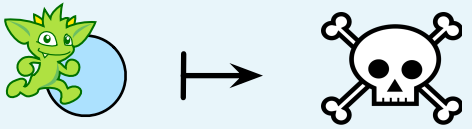
`out('knows') :`



"For a traverser at *a*, clone the traverser across multiple objects in *B*."

$\texttt{filter} : A^* \rightarrow A^*$

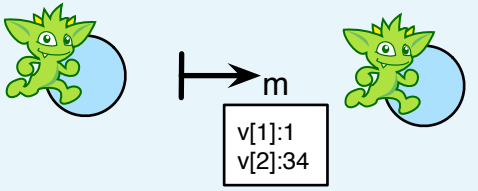
`has('age',lt(30)) :`



"For a traverser at *a*, either kill the traverser or leave him alone."

$\texttt{sideEffect} : A^* \rightarrow_x A^*$

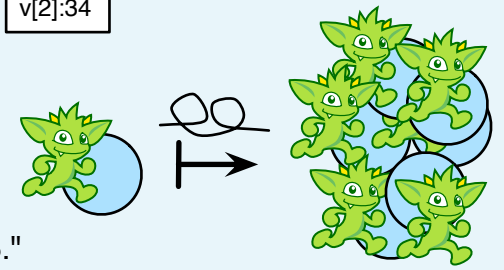
`groupCount('m') :`



"For a traverser at *a*, leave him alone though manipulate some data structure *x*."

$\texttt{branch} : A^* \rightarrow^b B^*$

`repeat(out('knows')).times(5) :`



"For a traverser at *a*, choose some internal branch *b* to ultimately yield traversers at objects of type B."

The Traversal

map	flatMap	filter	sideEffect	branch
LabelStep	VertexStep	AndStep	GroupCountStep	BranchStep
IdStep	EdgeVertexStep	CoinStep	GroupStep	ChooseStep
PathStep	CoalesceStep	CyclicPathStep	StoreStep	LocalStep
SackStep	...	DedupStep	AggregateStep	RepeatStep
SelectStep		DropStep	InjectStep	UnionStep
MatchStep		FilterStep	SubgraphStep	...
ConstantStep		HasStep	TreeStep	
...		IsStep	IdentityStep	
		NotStep	...	
		OrStep		
		RangeStep		
		...		

"This is the step library (instruction set) of the Gremlin graph traversal machine (virtual machine)."

The Traversal

Linear Motif

$$f \circ g \circ h$$

f g h

```
g.V(1).out('knows').has('age',lt(30)).values('name')
```

"The 'byte code' of Gremlin is a sequence of steps .... "

The Traversal

Linear Motif

$$f \circ g \circ h$$

f g h

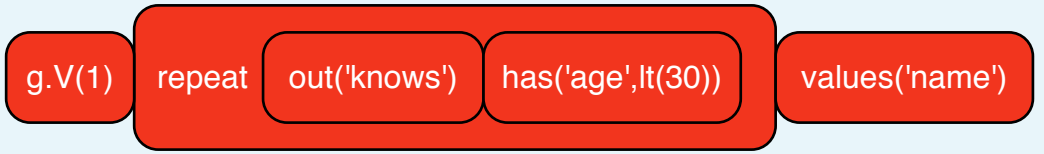
```
g.V(1).out('knows').has('age',lt(30)).values('name')
```

Nested Motif

$$f(g \circ h) \circ k$$

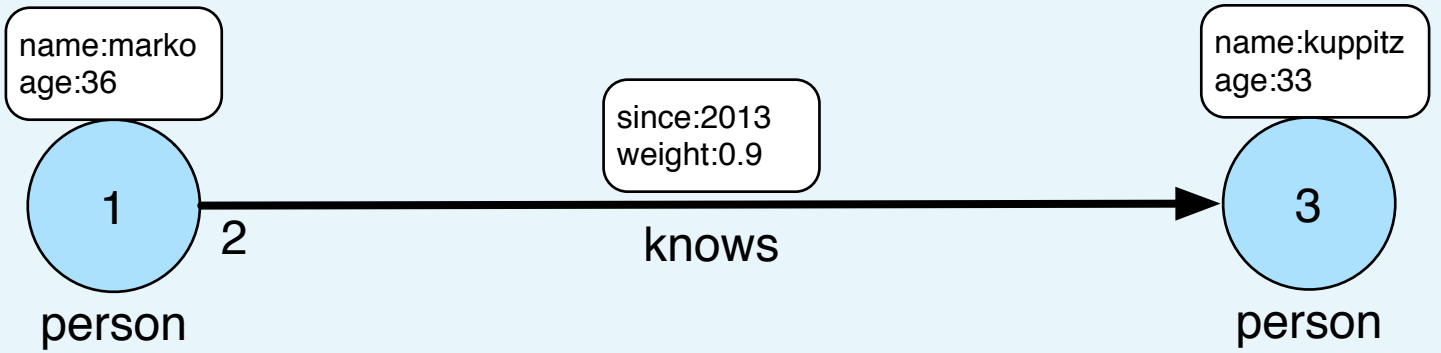
f g h k

```
g.V(1).repeat(out('knows').has('age',lt(30))).values('name')
```



"The 'byte code' of Gremlin is a sequence of steps with some steps nested within each other."

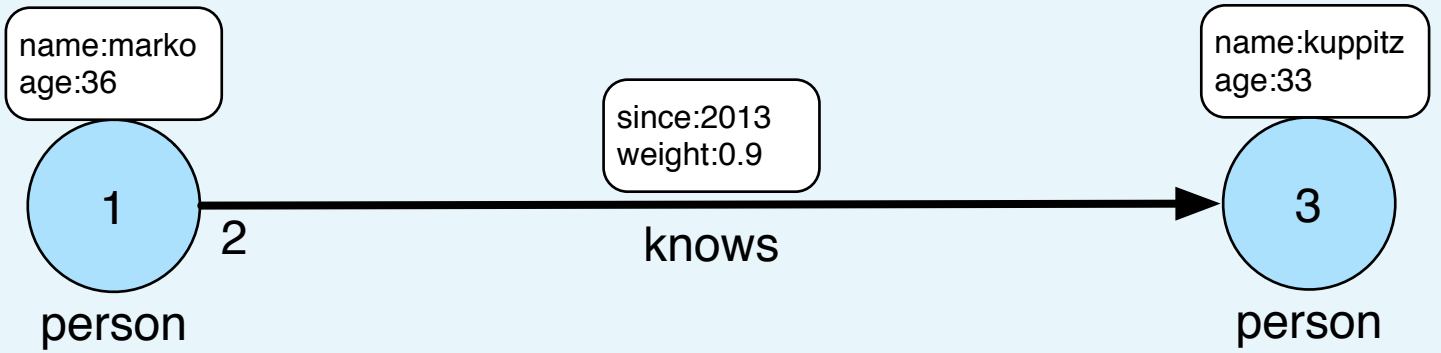
```
gremlin> graph
==>tinkergraph[vertices:2 edges:1]
gremlin>
```



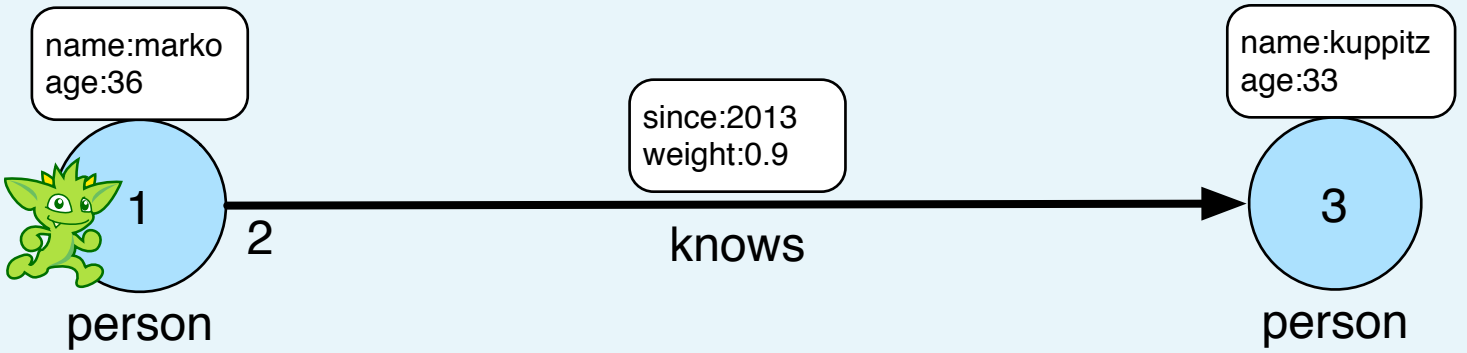
"Back to our super baby toy graph."

```
gremlin> graph
==>tinkergraph[vertices:2 edges:1]
gremlin> g = graph.traversal()
==>graphtraversalsource[tinkergraph[vertices:2 edges:1], standard]
gremlin>
```

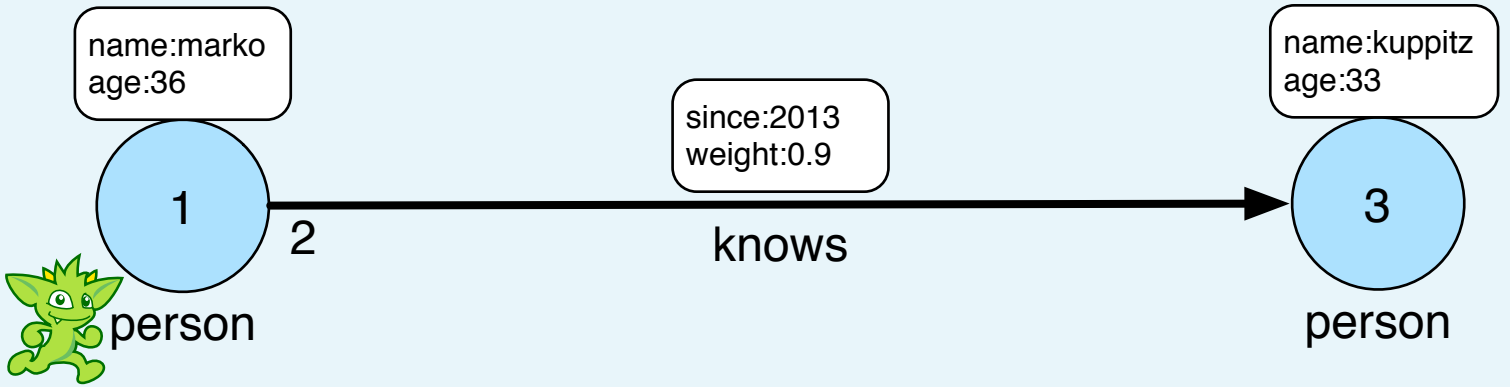
OLTP vs. OLAP
graph database vs. processor
Neo4j vs. Giraph



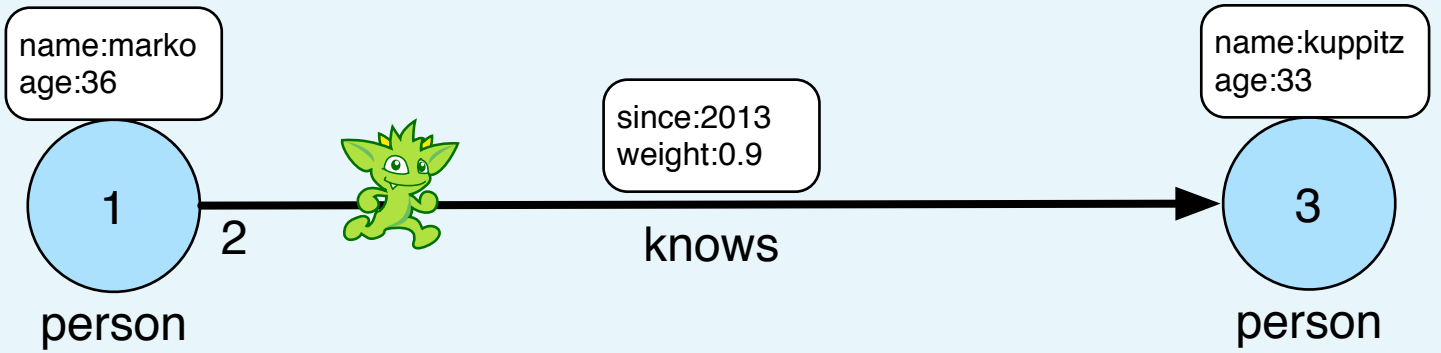
```
gremlin> graph
==>tinkergraph[vertices:2 edges:1]
gremlin> g = graph.traversal()
==>graphtraversalsource[tinkergraph[vertices:2 edges:1], standard]
gremlin> g.V(1)
==>v[1]
gremlin>
```



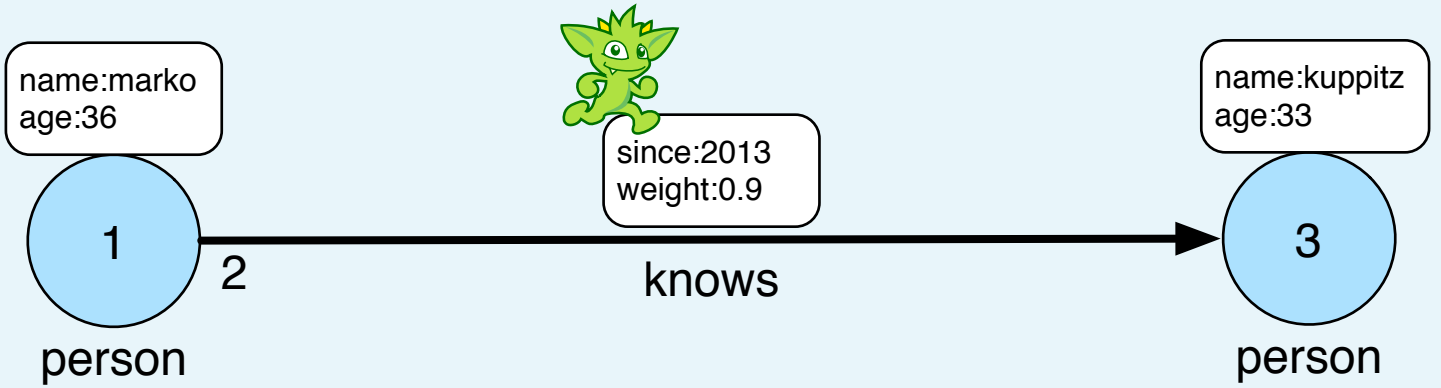
```
gremlin> graph
==>tinkergraph[vertices:2 edges:1]
gremlin> g = graph.traversal()
==>graphtraversalsource[tinkergraph[vertices:2 edges:1], standard]
gremlin> g.V(1)
==>v[1]
gremlin> g.V(1).label()
==>person
gremlin>
```



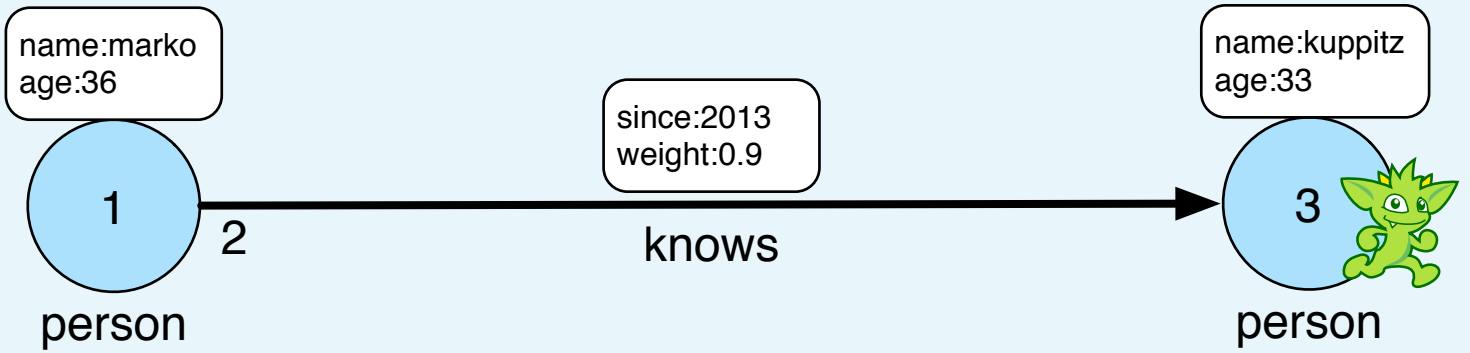
```
gremlin> graph
==>tinkergraph[vertices:2 edges:1]
gremlin> g = graph.traversal()
==>graphtraversalsource[tinkergraph[vertices:2 edges:1], standard]
gremlin> g.V(1)
==>v[1]
gremlin> g.V(1).label()
==>person
gremlin> g.V(1).outE()
==>e[2][1-knows->3]
gremlin>
```



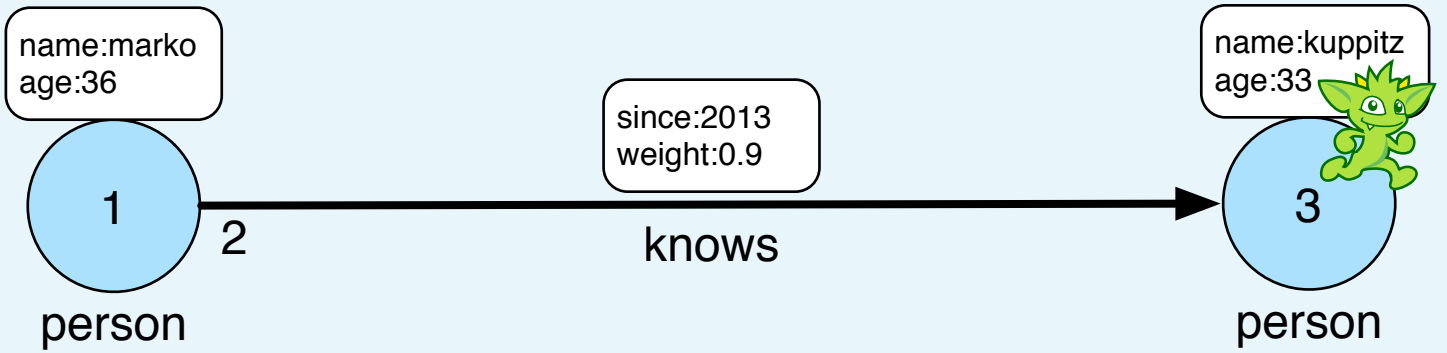
```
gremlin> graph
==>tinkergraph[vertices:2 edges:1]
gremlin> g = graph.traversal()
==>graphtraversalsource[tinkergraph[vertices:2 edges:1], standard]
gremlin> g.V(1)
==>v[1]
gremlin> g.V(1).label()
==>person
gremlin> g.V(1).outE()
==>e[2][1-knows->3]
gremlin> g.V(1).outE().valueMap()
==>[weight:0.9, since:2013]
gremlin>
```



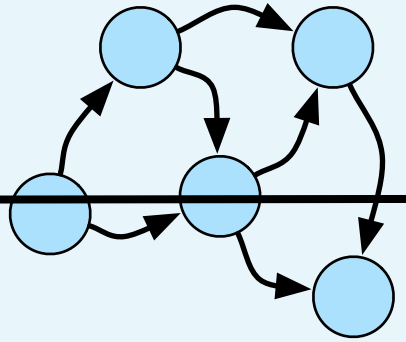
```
gremlin> graph
==>tinkergraph[vertices:2 edges:1]
gremlin> g = graph.traversal()
==>graphtraversalsource[tinkergraph[vertices:2 edges:1], standard]
gremlin> g.V(1)
==>v[1]
gremlin> g.V(1).label()
==>person
gremlin> g.V(1).outE()
==>e[2][1-knows->3]
gremlin> g.V(1).outE().valueMap()
==>[weight:0.9, since:2013]
gremlin> g.V(1).out()
==>v[3]
gremlin>
```



```
gremlin> graph
==>tinkergraph[vertices:2 edges:1]
gremlin> g = graph.traversal()
==>graphtraversalsource[tinkergraph[vertices:2 edges:1], standard]
gremlin> g.V(1)
==>v[1]
gremlin> g.V(1).label()
==>person
gremlin> g.V(1).outE()
==>e[2][1-knows->3]
gremlin> g.V(1).outE().valueMap()
==>[weight:0.9, since:2013]
gremlin> g.V(1).out()
==>v[3]
gremlin> g.V(1).out().values('age')
==>33
gremlin>
```

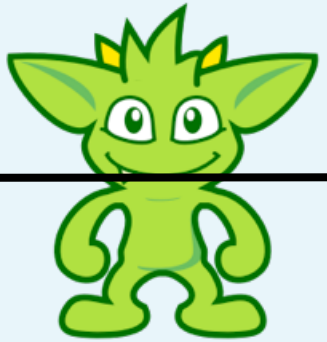


The Machine Components



The Graph

The Data



The Traverser

The CPU

The Traversal

The Program

The Machine Components

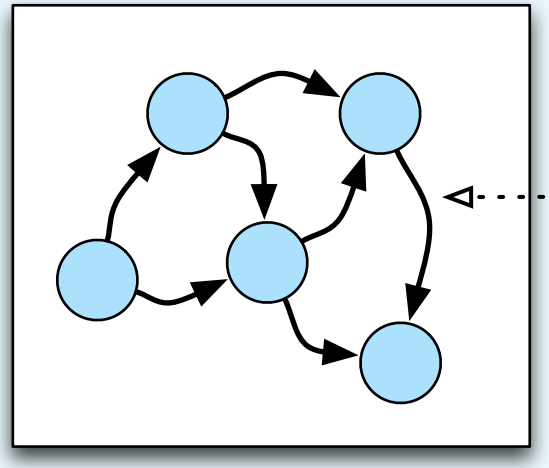
$$G \longleftarrow \mu \quad \frac{t \in T}{\{\Delta, \beta, \varsigma, \iota\}} \quad \psi \longrightarrow \Psi$$

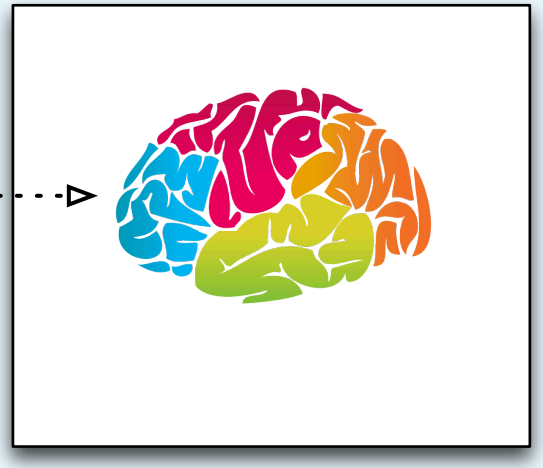
The Machine Components

$$G \longleftarrow \mu \quad \frac{t \in T}{\{\Delta, \beta, \varsigma, \iota\}} \quad \psi \longrightarrow \Psi$$

The Machine Components

$$G \longleftarrow \mu \frac{t \in T}{\{\Delta, \beta, \varsigma, \iota\}} \psi \longrightarrow \Psi$$





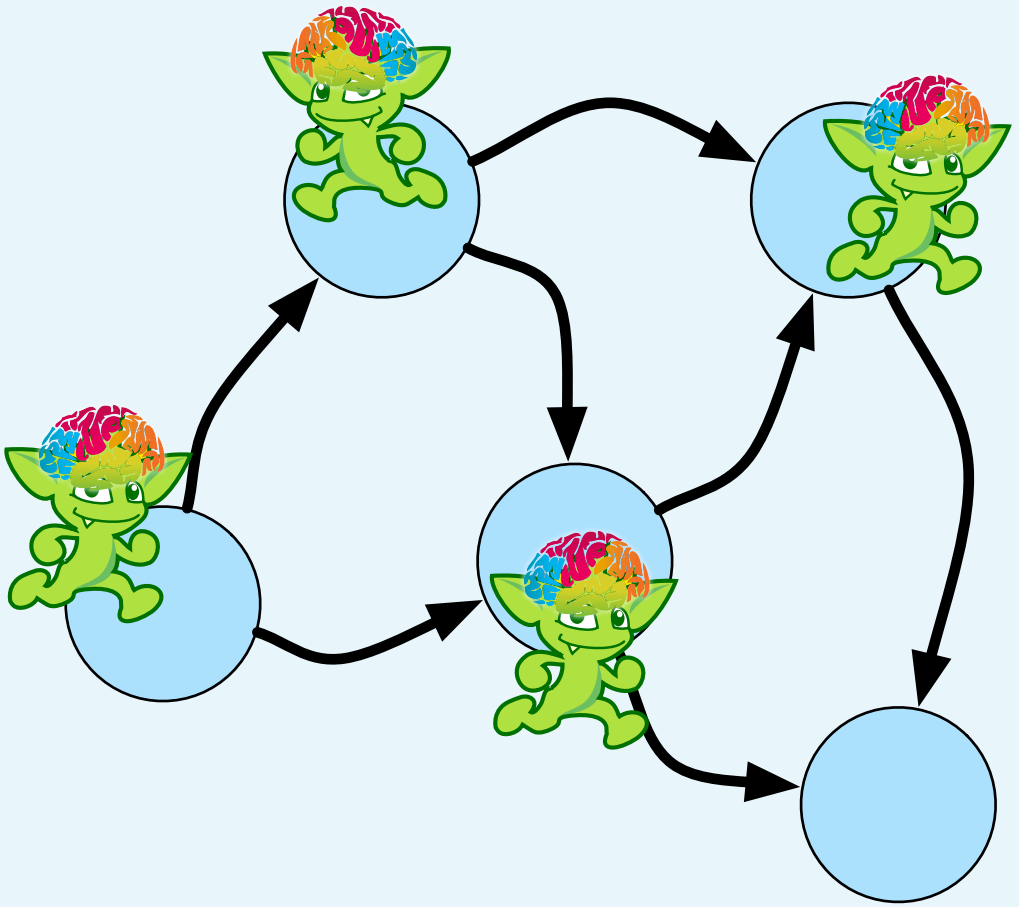
Graph Structure
(Data)

+

Graph Process
(Algorithm)

= Graph Computing

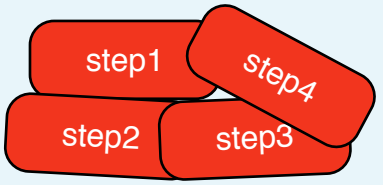
The Machine Components



"Traversers move about a graph as instructed by their traversal. The result of the computation is
1.) the final location of all halted traversers and
2.) the state of any side-effect data structures."

Part 2: The Gremlin Traversal Language

Gremlin's step library is the instruction set of the Gremlin traversal machine.



A linear/nested composition of steps forms a traversal which is executed by the Gremlin traversal machine.

=

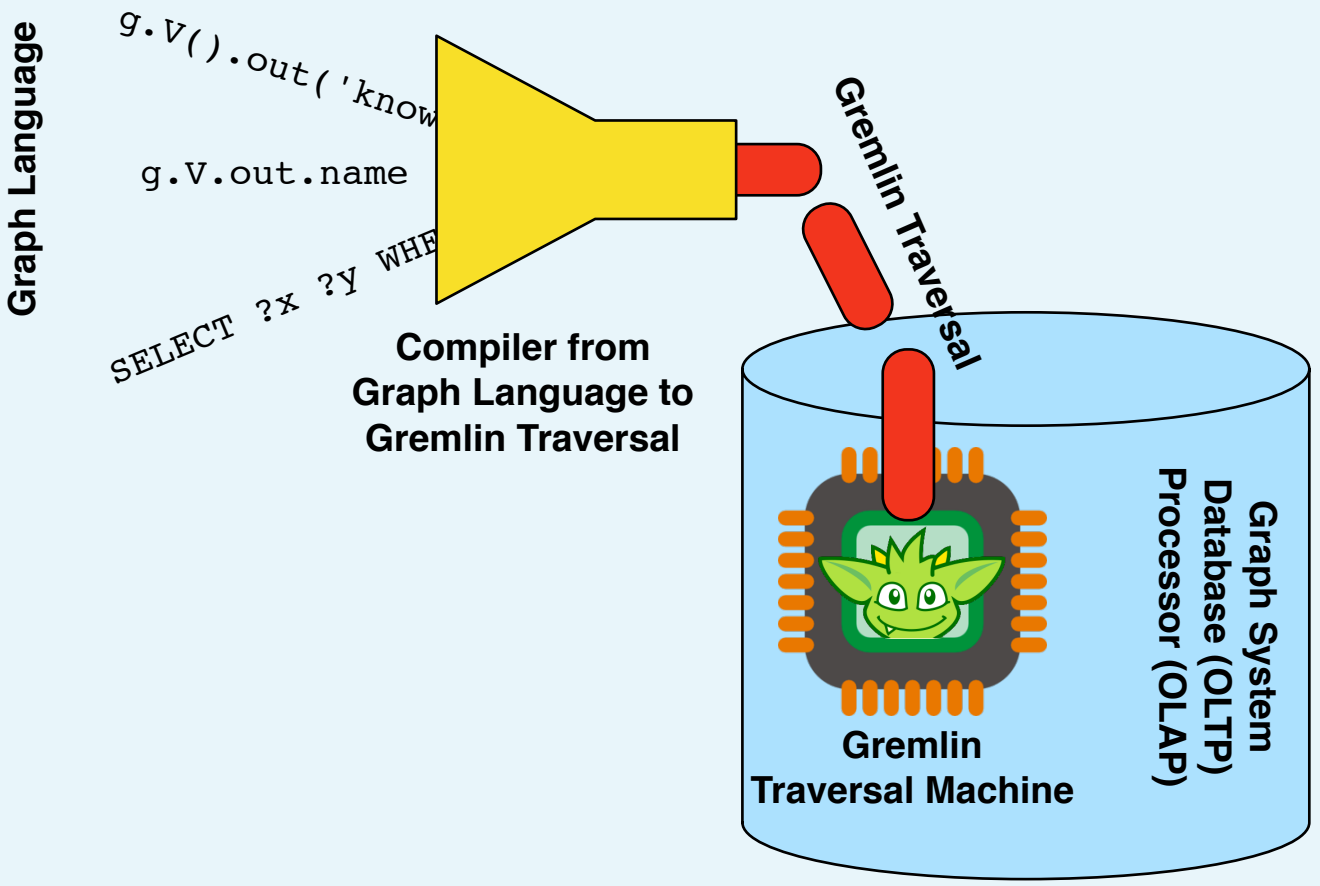
Gremlin-Java8 is the language provided by TinkerPop, though any language (with respective compiler) can be used to write Gremlin traversals.

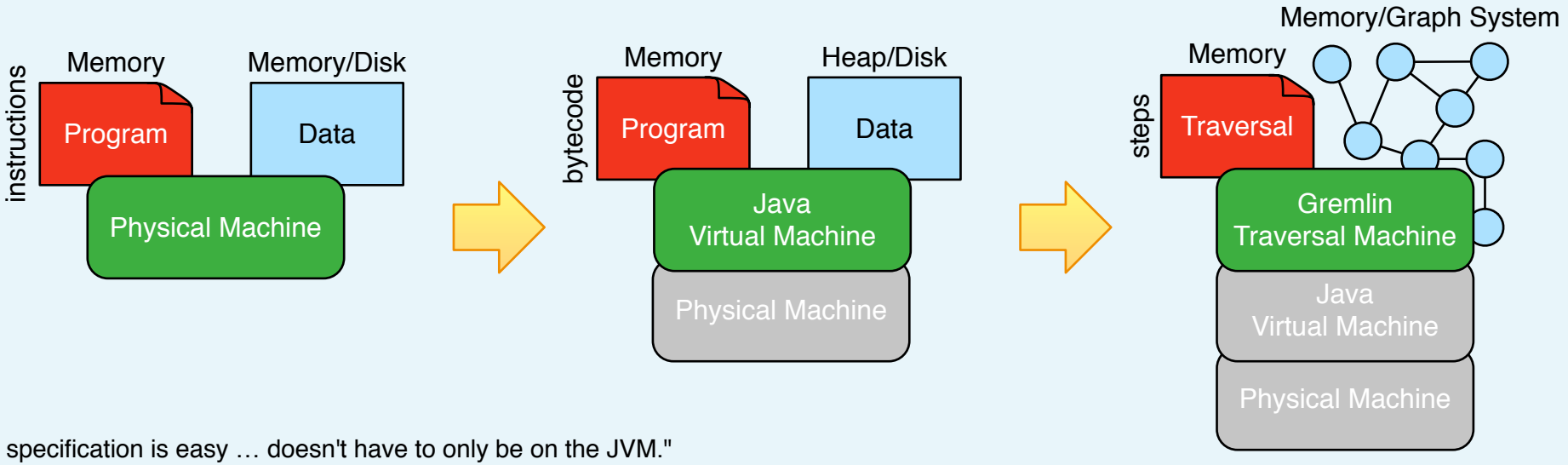
Gremlin-Java8

```
g.V(1).as("a").
  out("knows").as("b").
  select("a","b")
```


```
SELECT ?a ?b WHERE {
  ?a e:knows ?b
  ?a v:id ?c
  FILTER (?c == 1)
}
```

Written by Daniel Kuppitz
https://github.com/dkuppitz/sparql-gremlin





"The specification is easy … doesn't have to only be on the JVM."

```
~/tinkerpop3$ bin/gremlin.sh
```

```
gremlin>
```

"Gremlin-Groovy can be executed in the Gremlin Console REPL and thus, is good for demonstrations."

```
~/tinkerpop3$ bin/gremlin.sh

         \,,,/
         (o o)
-----oOOo-(3)-oOOo-----
plugin activated: tinkerpop.server
plugin activated: tinkerpop.utilities
plugin activated: tinkerpop.tinkergraph
gremlin>
```

```
~/tinkerpop3$ bin/gremlin.sh

         \,,,/
         (o o)
-----oOOo-(3)-oOOo-----
plugin activated: tinkerpop.server
plugin activated: tinkerpop.utilities
plugin activated: tinkerpop.tinkergraph
gremlin> graph = TinkerGraph.open()
==>tinkergraph[vertices:0 edges:0]
gremlin>
```

```
~/tinkerpop3$ bin/gremlin.sh

         \,,,/
         (o o)
-----oOOo-(3)-oOOo-----
plugin activated: tinkerpop.server
plugin activated: tinkerpop.utilities
plugin activated: tinkerpop.tinkergraph
gremlin> graph = TinkerGraph.open()
==>tinkergraph[vertices:0 edges:0]
gremlin> graph.io(graphml()).readGraph('data/grateful-dead.xml')
==>null
gremlin>
```

```
~/tinkerpop3$ bin/gremlin.sh

         \,,,/
         (o o)
-----oOOo-(3)-oOOo-----
plugin activated: tinkerpop.server
plugin activated: tinkerpop.utilities
plugin activated: tinkerpop.tinkergraph
gremlin> graph = TinkerGraph.open()
==>tinkergraph[vertices:0 edges:0]
gremlin> graph.io(graphml()).readGraph('data/grateful-dead.xml')
==>null
gremlin>
```

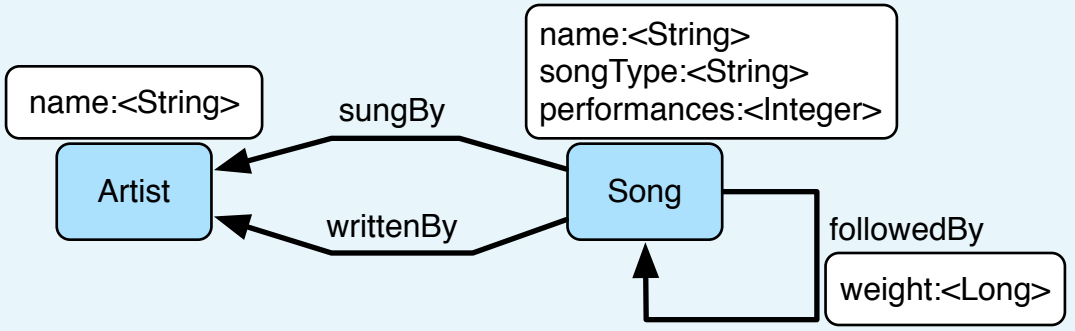
Gryo

GraphSON

GraphML

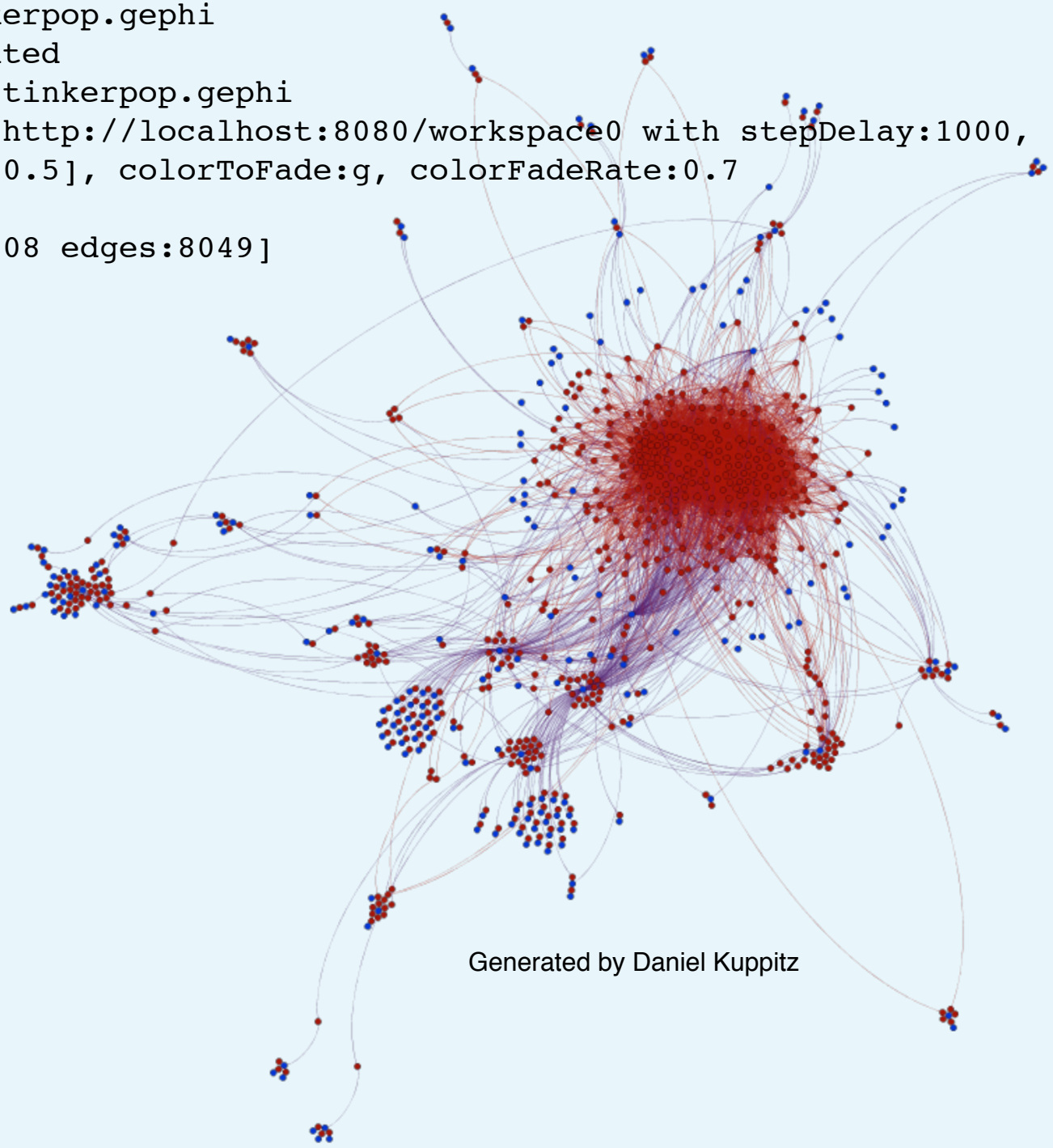
"TinkerPop supports three I/O graph formats. Though the user can add their own format/parser."

```
~/tinkerpop3$ bin/gremlin.sh

         \,,,/
         (o o)
-----oOOo-(3)-oOOo-----
plugin activated: tinkerpop.server
plugin activated: tinkerpop.utilities
plugin activated: tinkerpop.tinkergraph
gremlin> graph = TinkerGraph.open()
==>tinkergraph[vertices:0 edges:0]
gremlin> graph.io(graphml()).readGraph('data/grateful-dead.xml')
==>null
gremlin>
```



```
gremlin> :plugin use tinkerpop.gephi
==>tinkerpop.gephi activated
gremlin> :remote connect tinkerpop.gephi
==>Connection to Gephi - http://localhost:8080/workspace0 with stepDelay:1000,
startRGBColor:[0.0, 1.0, 0.5], colorToFade:g, colorFadeRate:0.7
gremlin> :> graph
==>tinkergraph[vertices:808 edges:8049]
gremlin>
```

http://gephi.org



Generated by Daniel Kuppitz

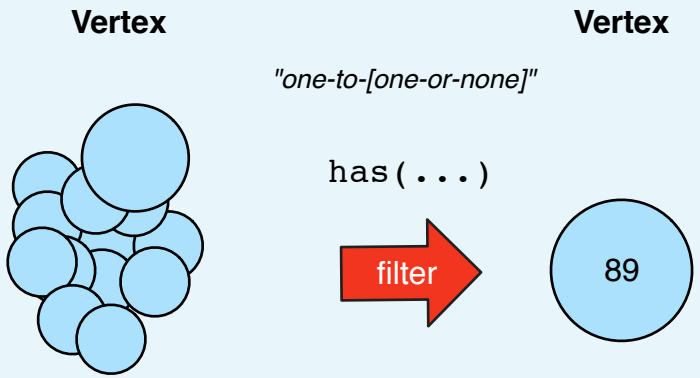
Rodriguez, M.A., Kuppitz, D., Yim, K., “Tales From the TinkerPop," DataStax Engineering Blog, 2015.
http://www.datastax.com/dev/blog/tales-from-the-tinkerpop

This is an actual Gremlin/R session I performed for this presentation.
I was interested in understanding why Grateful Dead concerts continue to fascinate me.

SPOILER ALERT: No local correlations -- correlations exist in the stationary distribution. EigenDead..the long run."

```
gremlin> graph = TinkerGraph.open()
==>tinkergraph[vertices:0 edges:0]
gremlin> graph.io(graphml()).readGraph('data/grateful-dead.xml')
==>null
gremlin> g = graph.traversal()
==>graphtraversalsource[tinkergraph[vertices:808 edges:8049], standard]
gremlin>
```

```
gremlin> graph = TinkerGraph.open()
==>tinkergraph[vertices:0 edges:0]
gremlin> graph.io(graphml()).readGraph('data/grateful-dead.xml')
==>null
gremlin> g = graph.traversal()
==>graphtraversalsource[tinkergraph[vertices:808 edges:8049], standard]
gremlin> g.V().has('name','DARK STAR')
==>v[89]
gremlin>
```

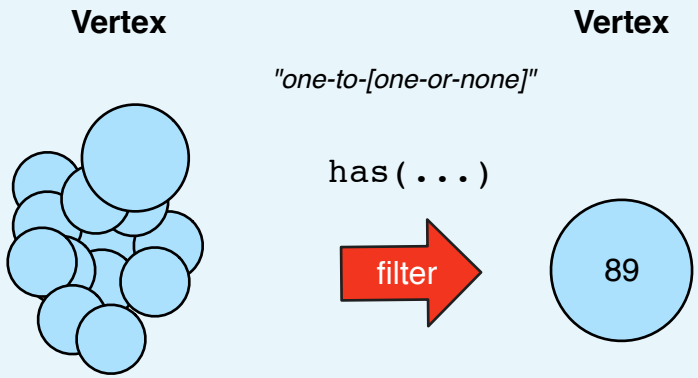


"Get the vertex named Dark Star."

```
gremlin> graph = TinkerGraph.open()
==>tinkergraph[vertices:0 edges:0]
gremlin> graph.io(graphml()).readGraph('data/grateful-dead.xml')
==>null
gremlin> g = graph.traversal()
==>graphtraversalsource[tinkergraph[vertices:808 edges:8049], standard]
gremlin> g.V().has('name','DARK STAR')
==>v[89]
gremlin>
```

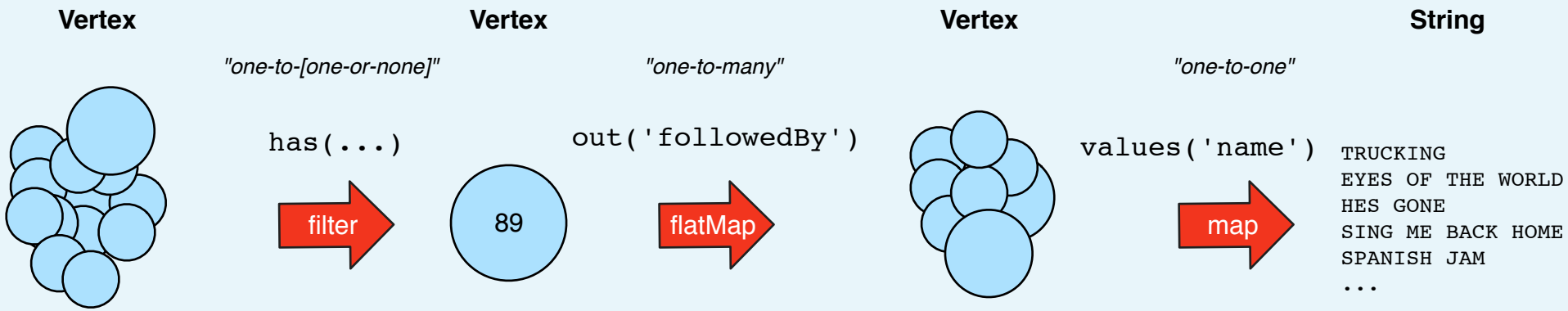
`ProviderOptimizationStrategy`

* OLTP graph system providers turn this into an index-lookup.



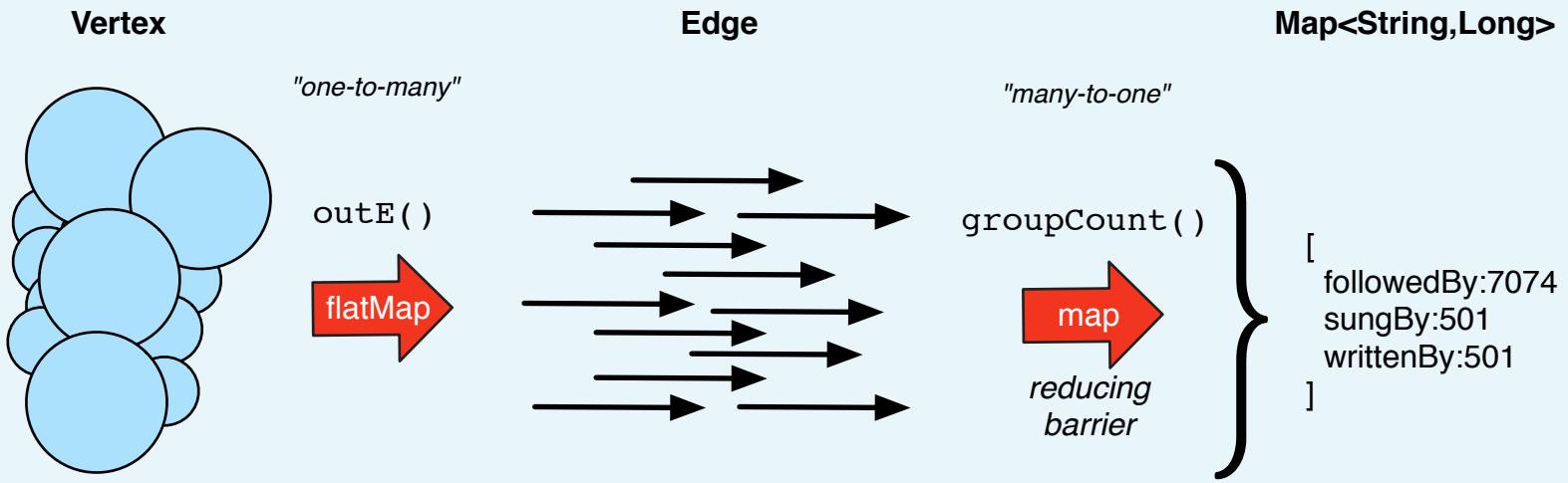
"TraversalStrategies are an important part of Gremlin (discussed at length in the conference article)."

```
gremlin> graph = TinkerGraph.open()
==>tinkergraph[vertices:0 edges:0]
gremlin> graph.io(graphml()).readGraph('data/grateful-dead.xml')
==>null
gremlin> g = graph.traversal()
==>graphtraversalsource[tinkergraph[vertices:808 edges:8049], standard]
gremlin> g.V().has('name','DARK STAR')
==>v[89]
gremlin> g.V().has('name','DARK STAR').out('followedBy').values('name')
==>TRUCKING
==>EYES OF THE WORLD
==>HES GONE
==>SING ME BACK HOME
==>SPANISH JAM
==>CHINA DOLL
==>MORNING DEW
==>WHARF RAT
==>THE OTHER ONE
==>MIND LEFT BODY JAM
...
gremlin>
```



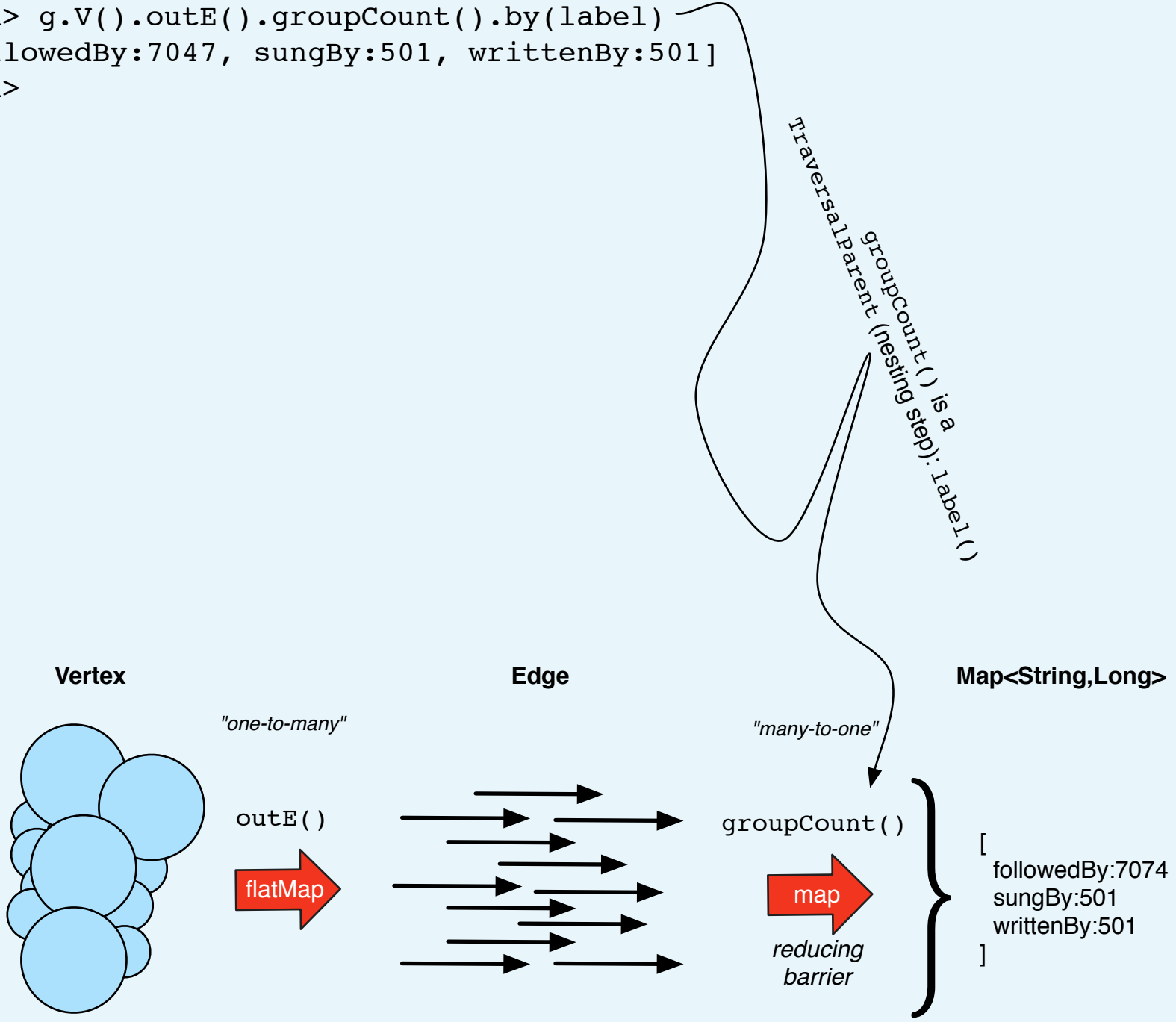
"What are the names of the songs that have followed Dark Star in concert?"

```
gremlin> g.V().outE().groupCount().by(label)
==>[followedBy:7047, sungBy:501, writtenBy:501]
gremlin>
```



"What is the distribution of edge labels in the graph?"

```
gremlin> g.V().outE().groupCount().by(label)
==>[followedBy:7047, sungBy:501, writtenBy:501]
gremlin>
```



"What is the distribution of edge labels in the graph?"

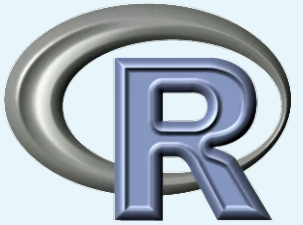
```
gremlin> g.V().groupCount().by(outE().count())
==>0=224
==>1=26
==>2=254
==>3=45
==>4=24
==>5=20
==>6=10
==>7=10
==>8=6
==>9=8
...
gremlin>
```

"What is the out-degree distribution of the graph?"

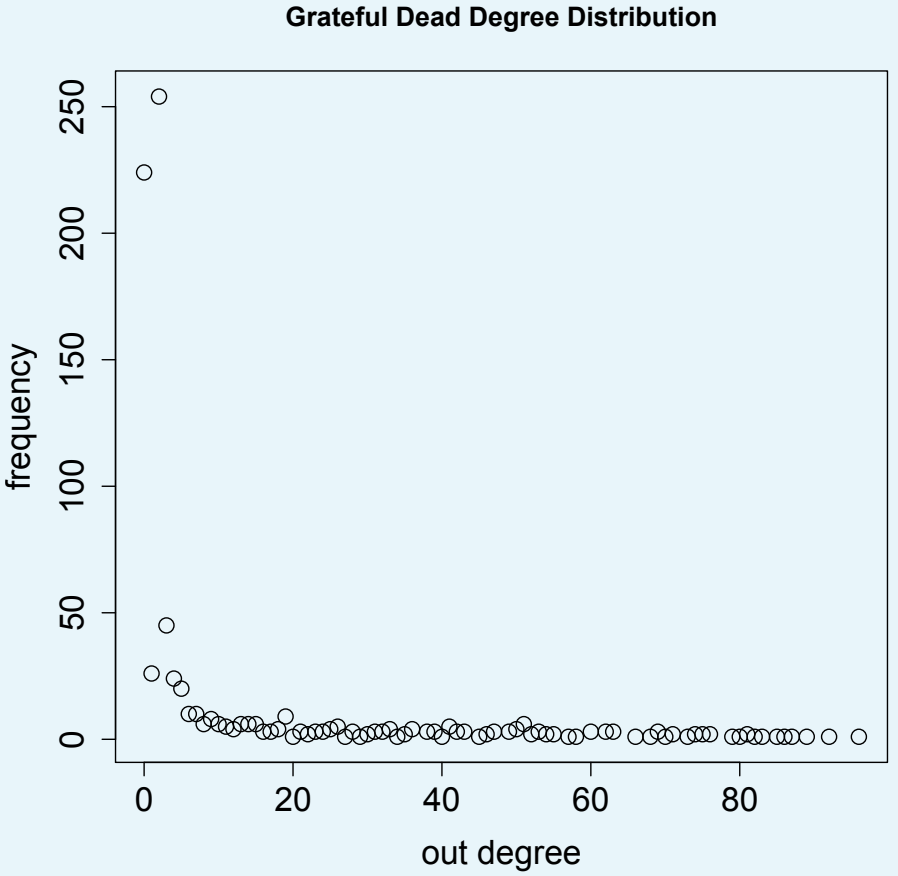
```
gremlin> g.V().groupCount().by(outE().count())
==>0=224
==>1=26
==>2=254
==>3=45
==>4=24
==>5=20
==>6=10
==>7=10
==>8=6
==>9=8
...
gremlin> f = new File('grateful-dead-distribution.txt')
==>grateful-dead-distribution.txt
gremlin> g.V().groupCount().by(outE().count()).
           next().each{degree,freq -> f.append(degree + '\t' + freq + '\n')}
gremlin> :quit
```

"Save result to a tab delimitated file for further analysis."

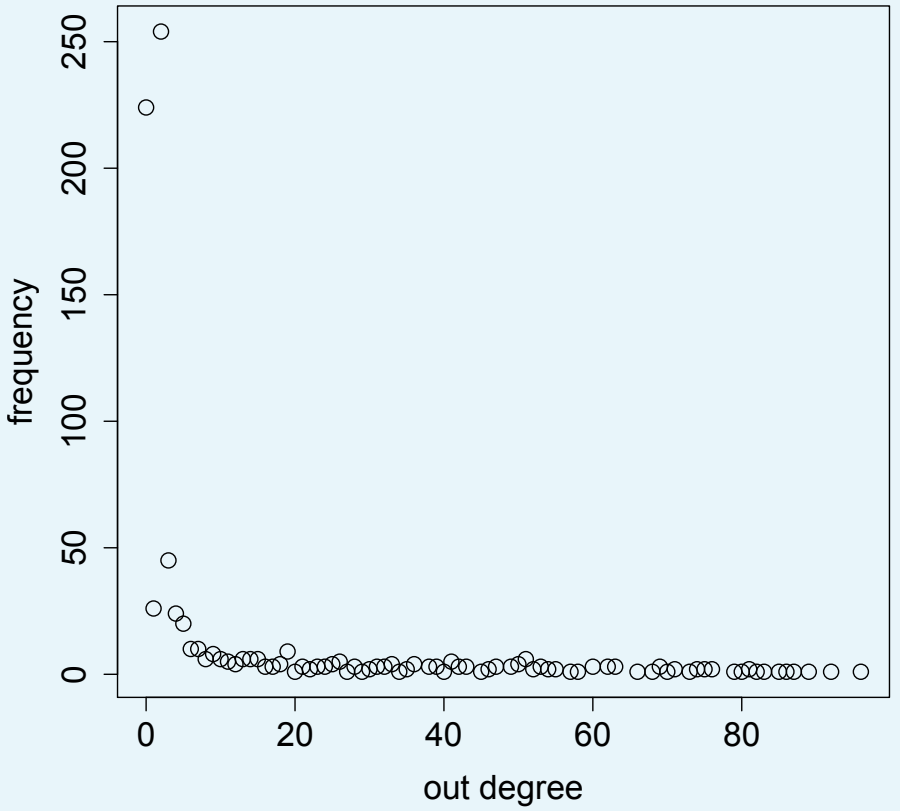
```
$ r
> t <- read.table("grateful-dead-distribution.txt")
>
```

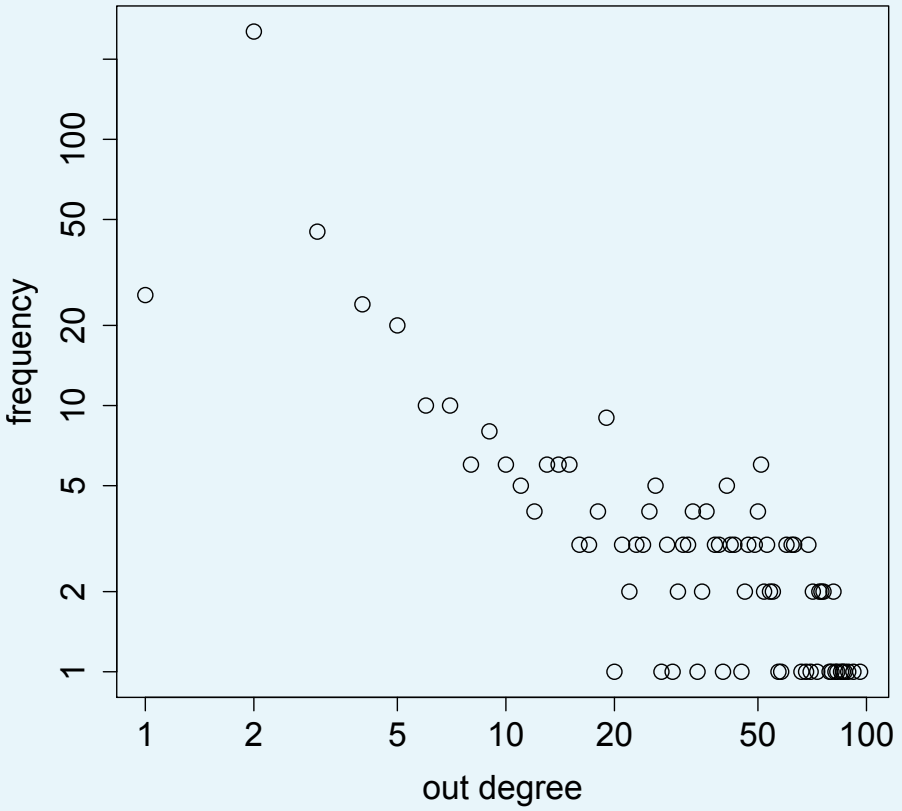


```
$ r
> t <- read.table("grateful-dead-distribution.txt")
> plot(t,xlab="out degree", ylab="frequency",cex=1.5,cex.lab=1.5,cex.axis=1.5,
        main="Grateful Dead Degree Distribution")
>
```



```
$ r
> t <- read.table("grateful-dead-distribution.txt")
> plot(t,xlab="out degree", ylab="frequency",cex=1.5,cex.lab=1.5,cex.axis=1.5,
       main="Grateful Dead Degree Distribution")
> plot(t,xlab="out degree", ylab="frequency",cex=1.5,cex.lab=1.5,cex.axis=1.5,
       main="Grateful Dead Degree Distribution",log="xy")
>
```





"Yea, yet another power law. Its 2005 and I get a free publication!"

```
gremlin> g.V().as('a').out('followedBy').as('b').
           select('a','b').by('performances')
==>[a:5, b:1]
==>[a:5, b:531]
==>[a:5, b:394]
==>[a:5, b:293]
==>[a:5, b:1]
==>[a:1, b:473]
==>[a:1, b:24]
==>[a:531, b:87]
==>[a:531, b:41]
==>[a:531, b:254]
...
gremlin> f = new File('performance-assortativity.txt')
==>performance-assortativity.txt
gremlin> g.V().as('a').out('followedBy').as('b').
           select('a','b').by('performances').
             each{f.append(it.a + '\t' + it.b + '\n')}
```

"Gremlins of a green, bulk together."

"Are songs that follow each other assortative by the number of times they are played in concert?"

https://en.wikipedia.org/wiki/Assortativity

```
gremlin> g.V().as('a').out('followedBy').as('b').
           select('a','b').by('performances')
==>[a:5, b:1]
==>[a:5, b:531]
==>[a:5, b:394]
==>[a:5, b:293]
==>[a:5, b:1]
==>[a:1, b:473]
==>[a:1, b:24]
==>[a:531, b:87]
==>[a:531, b:41]
==>[a:531, b:254]
...
gremlin> f = new File('performance-assortativity.txt')
==>performance-assortativity.txt
gremlin> g.V().as('a').out('followedBy').as('b').
           select('a','b').by('performances').
             each{f.append(it.a + '\t' + it.b + '\n')}
```

"...you know: birds and feathers and flocking togethers."

```
> cor.test(t[,1],t[,2], test='pearson')
   Pearson's product-moment correlation

data:  t[, 1] and t[, 2]
t = -3.9525, df = 7045, p-value = 7.809e-05
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 -0.07030966 -0.02371587
sample estimates:
        cor
-0.04703835
```

No correlation between songs by performance.

```
gremlin> g.V().as('a').out('followedBy').as('b').
           select('a','b').by('songType')
==>[a:cover, b:cover]
==>[a:cover, b:cover]
==>[a:cover, b:original]
==>[a:cover, b:cover]
==>[a:cover, b:cover]
==>[a:cover, b:original]
==>[a:cover, b:original]
==>[a:cover, b:original]
==>[a:cover, b:cover]
==>[a:cover, b:cover]
...
gremlin> f = new File('songType-assortativity.txt')
==>songType-assortativity.txt
gremlin> g.V().as('a').out('followedBy').as('b').
           select('a','b').by('songType').
             each{f.append((it.a == 'cover' ? 0 : 1) + '\t' +
                           (it.b == 'cover' ? 0 : 1) + '\n')}
```

"Hmmm.. need numbers -- ah! only two song types."

"Are songs that follow each other assortative by either being a cover or an original song?"

```
gremlin> g.V().as('a').out('followedBy').as('b').
           select('a','b').by('songType')
==>[a:cover, b:cover]
==>[a:cover, b:cover]
==>[a:cover, b:original]
==>[a:cover, b:cover]
==>[a:cover, b:cover]
==>[a:cover, b:original]
==>[a:cover, b:original]
==>[a:cover, b:original]
==>[a:cover, b:cover]
==>[a:cover, b:cover]
...
gremlin> f = new File('songType-assortativity.txt')
==>songType-assortativity.txt
gremlin> g.V().as('a').out('followedBy').as('b').
           select('a','b').by('songType').
             each{f.append((it.a == 'cover' ? 0 : 1) + '\t' +
                           (it.b == 'cover' ? 0 : 1) + '\n')}
```

```
> cramersV(chisq.test(t[,1],t[,2])$observed)
[1] 0.0093161
```

No correlation between songs by song type (original or cover).

"Cramer's V is for nominal data correlations."

```
gremlin> g.V().as('a').out('followedBy').as('b').
           select('a','b').by(out('sungBy').values('name'))
==>[a:Garcia, b:Spencer_Davis]
==>[a:Garcia, b:Weir]
==>[a:Garcia, b:Garcia]
==>[a:Garcia, b:Garcia]
==>[a:Garcia, b:Weir]
==>[a:Spencer_Davis, b:Weir]
==>[a:Spencer_Davis, b:Grateful_Dead]
==>[a:Weir, b:Garcia]
==>[a:Weir, b:All]
==>[a:Weir, b:Garcia]
...
gremlin> f = new File('singer-assortativity.txt')
==>singer-assortativity.txt
gremlin> g.V().as('a').out('followedBy').as('b').
           select('a','b').by(out('sungBy').values('name')).
             each{f.append(it.a.hashCode() + '\t' + it.b.hashCode() + '\n')}
```

"Dah! Now I need an algorithm to turn a String to a unique integer.
Duh -- hashCode()."

"Are songs that follow each other assortative by their singer?"

```
gremlin> g.V().as('a').out('followedBy').as('b').
           select('a','b').by(out('sungBy').values('name'))
==>[a:Garcia, b:Spencer_Davis]
==>[a:Garcia, b:Weir]
==>[a:Garcia, b:Garcia]
==>[a:Garcia, b:Garcia]
==>[a:Garcia, b:Weir]
==>[a:Spencer_Davis, b:Weir]
==>[a:Spencer_Davis, b:Grateful_Dead]
==>[a:Weir, b:Garcia]
==>[a:Weir, b:All]
==>[a:Weir, b:Garcia]
...
gremlin> f = new File('singer-assortativity.txt')
==>singer-assortativity.txt
gremlin> g.V().as('a').out('followedBy').as('b').
           select('a','b').by(out('sungBy').values('name')).
             each{f.append(it.a.hashCode() + '\t' + it.b.hashCode() + '\n')}
```

```
> cramersV(chisq.test(t[,1],t[,2])$observed)
[1] 0.2354349
```

Songs are weakly correlated by singers.

"However, its typically just Jerry and Bobby trading off in ones or twos…"

```
gremlin> g.V().hasLabel('song').
           repeat(out('followedBy').groupCount('m').by('name')).times(8).
             cap('m').
           order(local).by(valueDecr).
           limit(local,10)
```

"Sorta getting bored working on the slides. My improv-vibe died…
oh yea, but I got a nasty classic riff I remember."

"What songs are most central in the concert network?"

"Not PaaaaaageRank again."

```
gremlin> g.V().hasLabel('song').
           repeat(out('followedBy').groupCount('m').by('name')).times(8).
             cap('m').
           order(local).by(valueDecr).
           limit(local,10)
==>PLAYING IN THE BAND=34142246667508
==>ME AND MY UNCLE=32094411419320
==>JACK STRAW=31867238591868
==>EL PASO=29973481580211
==>TRUCKING=29819272116849
==>PROMISED LAND=28663488257022
==>CHINA CAT SUNFLOWER=28569992924918
==>CUMBERLAND BLUES=26320323048221
==>LOOKS LIKE RAIN=26138795229794
==>RAMBLE ON ROSE=26059394903880
gremlin>
```

"Huh, those are the tracks from the 'greatest' Grateful Dead's greatest hits."

https://en.wikipedia.org/wiki/What_a_Long_Strange_Trip_It%27s_Been

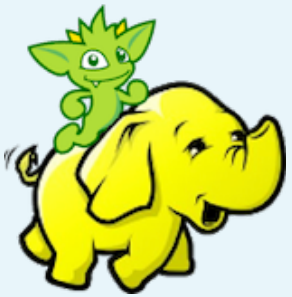
"Rodriguez, M.A., Gintautas, V., Pepe, A., “A Grateful Dead Analysis: The Relationship Between Concert and Listening Behavior,” First Monday 14:1, ISSN:1396-0466, http://arxiv.org/abs/0807.2466, January 2009."

```
gremlin> g.V().hasLabel('song').
           repeat(out('followedBy').groupCount('m').by('name')).times(8).
             cap('m').
           order(local).by(valueDecr).
           limit(local,10)
==>PLAYING IN THE BAND=34142246667508
==>ME AND MY UNCLE=32094411419320
==>JACK STRAW=31867238591868
==>EL PASO=29973481580211
==>TRUCKING=29819272116849
==>PROMISED LAND=28663488257022
==>CHINA CAT SUNFLOWER=28569992924918
==>CUMBERLAND BLUES=26320323048221
==>LOOKS LIKE RAIN=26138795229794
==>RAMBLE ON ROSE=26059394903880
gremlin> clock(10){g.V().hasLabel('song').
           repeat(out('followedBy').groupCount('m').by('name')).times(8).
             cap('m').
           order(local).by(valueDecr).
           limit(local,10)}
==>4040.761404
gremlin>
```

"34 trillion traversers passed through this vertex."

"4 seconds to analyze that many paths -- that is the power of bulking."

```
gremlin> graph = HadoopGraph.open('hadoop-grateful-gryo.properties')
==>hadoopgraph[gryoinputformat->gryooutputformat]
gremlin>
```



"And now note how the same Gremlin queries you've seen thus far can execute across a cluster."

```
gremlin> graph = HadoopGraph.open('hadoop-grateful-gryo.properties')
==>hadoopgraph[gryoinputformat->gryooutputformat]
gremlin> g = graph.traversal(computer(SparkGraphComputer))
==>graphtraversalsource[hadoopgraph[gryoinputformat->gryooutputformat],
sparkgraphcomputer]
gremlin>
```

GraphComputer's are OLAP systems

"Lets use Apache Spark via TinkerPop's SparkGraphComputer."

```
gremlin> graph = HadoopGraph.open('hadoop-grateful-gryo.properties')
==>hadoopgraph[gryoinputformat->gryooutputformat]
gremlin> g = graph.traversal(computer(SparkGraphComputer))
==>graphtraversalsource[hadoopgraph[gryoinputformat->gryooutputformat],
sparkgraphcomputer]
gremlin> g.V().match(
           __.as('a').out('sungBy').as('b'),
           __.as('a').out('writtenBy').as('b'),
           __.as('a').values('name').as('c'),
           __.as('b').values('name').as('d')).
             select('c','d')
```

"_" is required by Gremlin-Groovy as "as" is a keyword in Groovy.

"Which songs were written and sung by the same person?"

```
gremlin> graph = HadoopGraph.open('hadoop-grateful-gryo.properties')
==>hadoopgraph[gryoinputformat->gryooutputformat]
gremlin> g = graph.traversal(computer(SparkGraphComputer))
==>graphtraversalsource[hadoopgraph[gryoinputformat->gryooutputformat],
sparkgraphcomputer]
gremlin> g.V().match(
           __.as('a').out('sungBy').as('b'),
           __.as('a').out('writtenBy').as('b'),
           __.as('a').values('name').as('c'),
           __.as('b').values('name').as('d')).
             select('c','d')
==>[c:ANY WONDER, d:Unknown]
==>[c:WALK DOWN THE STREET, d:Unknown]
==>[c:LEAVE YOUR LOVE AT HOME, d:Unknown]
==>[c:COWBOY SONG, d:Unknown]
==>[c:NEIGHBORHOOD GIRLS, d:Suzanne_Vega]
==>[c:MINDBENDER, d:Garcia_Lesh]
==>[c:EQUINOX, d:Lesh]
==>[c:NO LEFT TURN UNSTONED (CARDBOARD COWBOY), d:Lesh]
==>[c:CHILDHOODS END, d:Lesh]
==>[c:NEVER TRUST A WOMAN, d:Mydland]
...
gremlin>
```

"That was a distributed, declarative graph pattern match query.
This could have been over a 1000 node cluster."

Gremlin-Groovy

```
g.V().match(
  __.as('a').out('sungBy').as('b'),
  __.as('a').out('writtenBy').as('b'),
  __.as('a').values('name').as('c'),
  __.as('b').values('name').as('d')).
     select('c','d')
```


```
SELECT ?c ?d WHERE {
  ?a e:writtenBy ?b .
  ?a e:sungBy ?b .
  ?a v:name ?c .
  ?b v:name ?d
}
```

"Gremlin supports the 'match'-style found in most pattern match languages such as SPARQL."

Gremlin-Groovy

```
g.V().match(
  __.as('a').out('sungBy').as('b'),
  __.as('a').out('writtenBy').as('b'),
  __.as('a').values('name').as('c'),
  __.as('b').values('name').as('d')).
     select('c','d')
```

Uses Apache Jena's SPARQL parser

SPARQL to Gremlin Compiler

```
SELECT ?c ?d WHERE {
  ?a e:writtenBy ?b .
  ?a e:sungBy ?b .
  ?a v:name ?c .
  ?b v:name ?d
}
```

***HOMEWORK**: Use Apache Calcite and convert SQL to a Gremlin traversal.*

Ted Wilmes was here.

"Two different graph languages can be compiled to the Gremlin traversal machine."

```
gremlin> :install com.datastax sparql-gremlin 0.1
==>Loaded: [com.datastax, sparql-gremlin, 0.1]
gremlin>  :plugin use datastax.sparql
==>datastax.sparql activated
gremlin>
```

"Install the SPARQL-Gremlin compiler developed by Daniel Kuppitz of DataStax."

```
gremlin> :install com.datastax sparql-gremlin 0.1
==>Loaded: [com.datastax, sparql-gremlin, 0.1]
gremlin>  :plugin use datastax.sparql
==>datastax.sparql activated
gremlin> :remote connect datastax.sparql g
==>SPARQL[graphtraversalsource[hadoopgraph[gryoinputformat->gryooutputformat],
sparkgraphcomputer]]
gremlin>
```

"Given that SPARQL is NOT Groovy, it is passed to the cluster as a remote String."

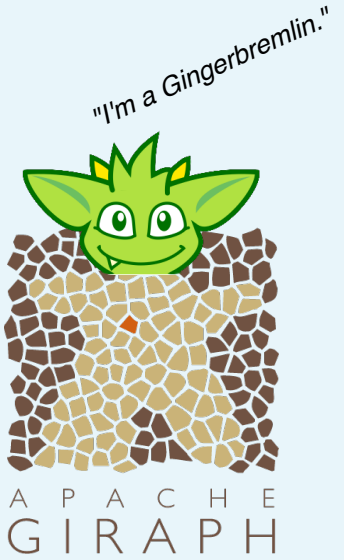
```
gremlin> :install com.datastax sparql-gremlin 0.1
==>Loaded: [com.datastax, sparql-gremlin, 0.1]
gremlin>  :plugin use datastax.sparql
==>datastax.sparql activated
gremlin> :remote connect datastax.sparql g
==>SPARQL[graphtraversalsource[hadoopgraph[gryoinputformat->gryooutputformat],
sparkgraphcomputer]]
gremlin> :> SELECT ?c ?d WHERE {
             ?a e:writtenBy ?b .
             ?a e:sungBy ?b .
             ?a v:name ?c .
             ?b v:name ?d }
==>[c:ANY WONDER, d:Unknown]
==>[c:WALK DOWN THE STREET, d:Unknown]
==>[c:LEAVE YOUR LOVE AT HOME, d:Unknown]
==>[c:COWBOY SONG, d:Unknown]
==>[c:NEIGHBORHOOD GIRLS, d:Suzanne_Vega]
==>[c:MINDBENDER, d:Garcia_Lesh]
==>[c:EQUINOX, d:Lesh]
==>[c:NO LEFT TURN UNSTONED (CARDBOARD COWBOY), d:Lesh]
==>[c:CHILDHOODS END, d:Lesh]
==>[c:NEVER TRUST A WOMAN, d:Mydland]
...
gremlin>
```

"SPARQL was just executed over Apache Spark by way of the Gremlin traversal machine."

```
gremlin> g = graph.traversal(computer(GiraphGraphComputer))
==>graphtraversalsource[hadoopgraph[gryoinputformat->gryooutputformat],
giraphgraphcomputer]
gremlin>
```

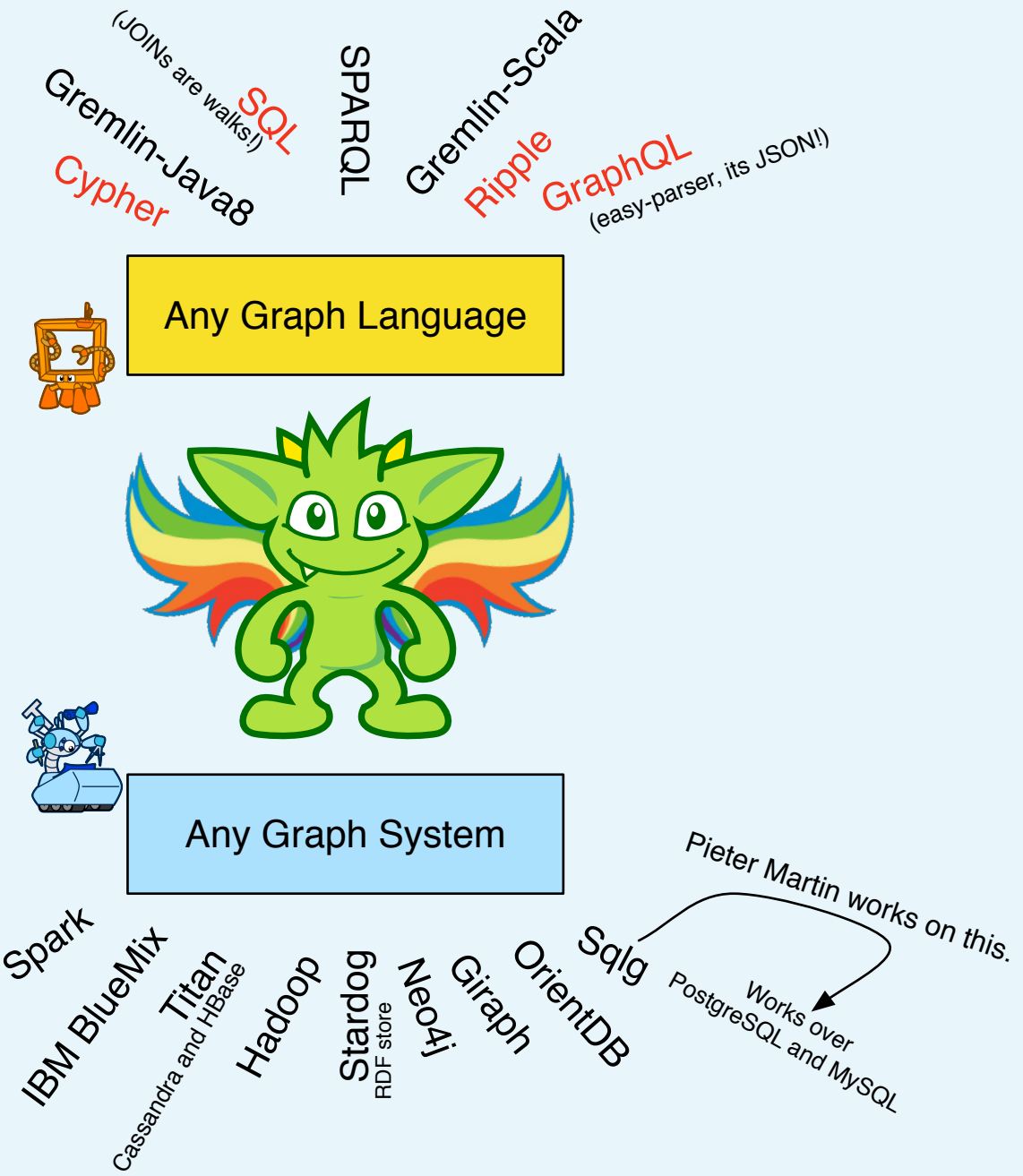
"Use a different GraphComputer."

```
gremlin> g = graph.traversal(computer(GiraphGraphComputer))
==>graphtraversalsource[hadoopgraph[gryoinputformat->gryooutputformat],
giraphgraphcomputer]
gremlin>  g.V().match(
            __.as('a').out('sungBy').as('b'),
            __.as('a').out('writtenBy').as('b'),
            __.as('a').values('name').as('c'),
            __.as('b').values('name').as('d')).
              select('c','d')
INFO  org.apache.hadoop.mapreduce.Job  - Running job: job_1445539638801_0011
INFO  org.apache.hadoop.mapreduce.Job  -  map 50% reduce 0%
INFO  org.apache.hadoop.mapreduce.Job  -  map 100% reduce 0%
INFO  org.apache.hadoop.mapreduce.Job  - Counters: 55
   ...
   Giraph Stats
      Superstep 1 GiraphComputation (ms)=2022
      Superstep 2 GiraphComputation (ms)=1994
      Superstep 3 GiraphComputation (ms)=999
      Superstep 4 GiraphComputation (ms)=1001
      Superstep 5 GiraphComputation (ms)=1254
      Total (ms)=21425
   ...
==>[c:IT MUST HAVE BEEN THE ROSES, d:Hunter]
==>[c:EASY WIND, d:Hunter]
==>[c:WHATLL YOU RAISE, d:Hunter]
==>[c:CRYPTICAL ENVELOPMENT, d:Garcia]
==>[c:CREAM PUFF WAR, d:Garcia]
==>[c:DRUMS, d:Grateful_Dead]
...
gremlin>
```



"Gremlin-Groovy just executed over Apache Giraph by way of the Gremlin traversal machine."

RED = "No known compiler."



* Many system providers are still on TinkerPop2 and thus, haven't migrated to TinkerPop3.

Keep on Traversin'...

Thank you….

Association for Computing Machinery

The Apache Software Foundation
http://www.apache.org/

DATASTAX

Rodriguez, M.A., Kuppitz, D., Yim, K., “Tales From the TinkerPop," DataStax Engineering Blog, 2015.
http://www.datastax.com/dev/blog/tales-from-the-tinkerpop