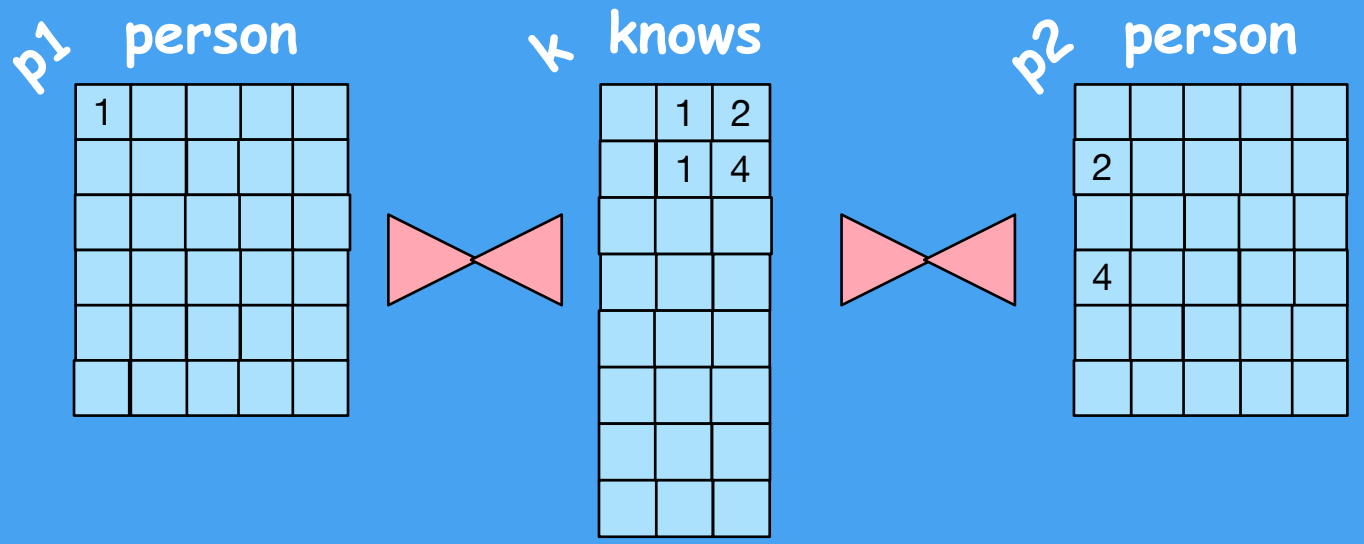
Gremlin 101.3
on your
FM Dial

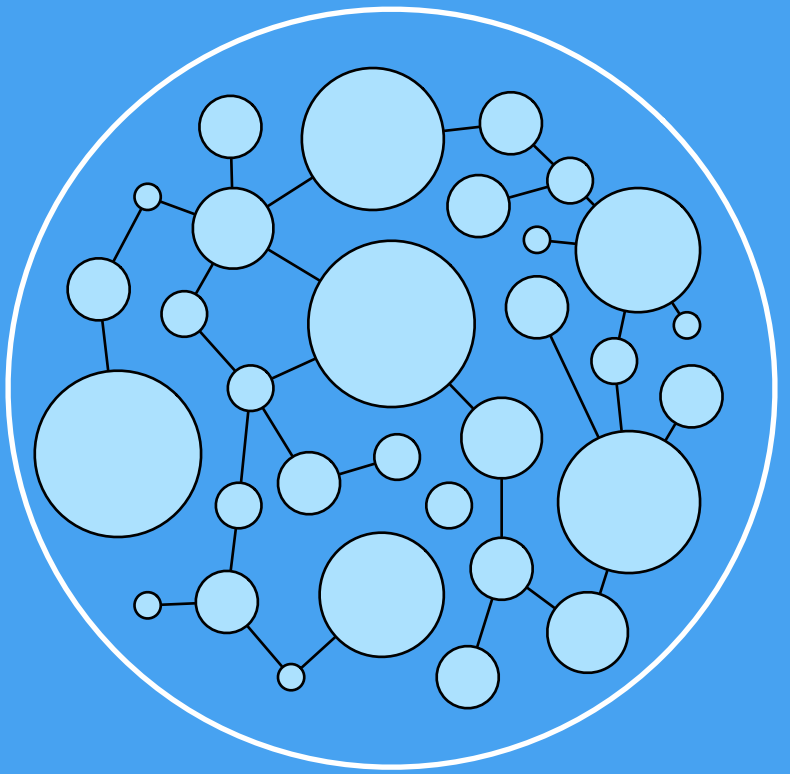
Marko A. Rodriguez
DATASTAX

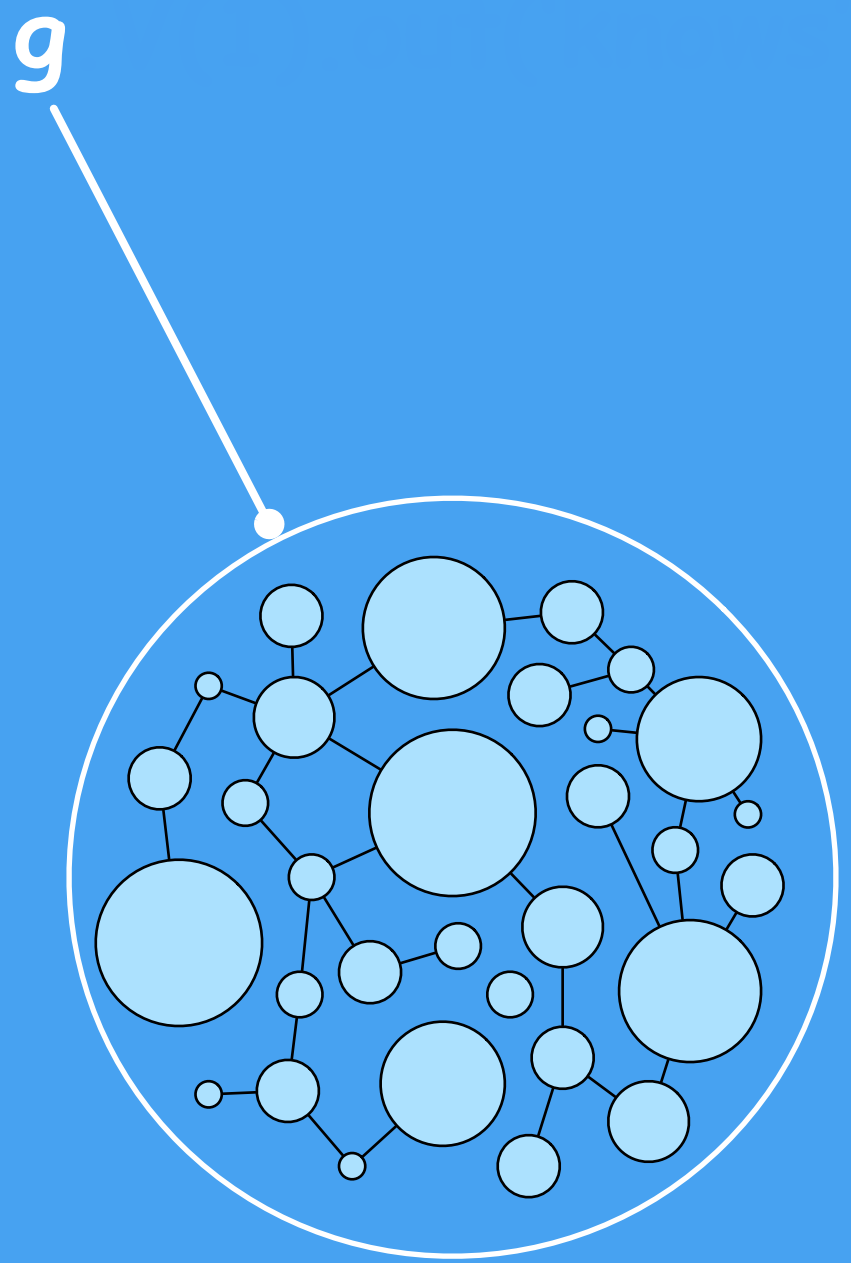
Who are person 1's friends?

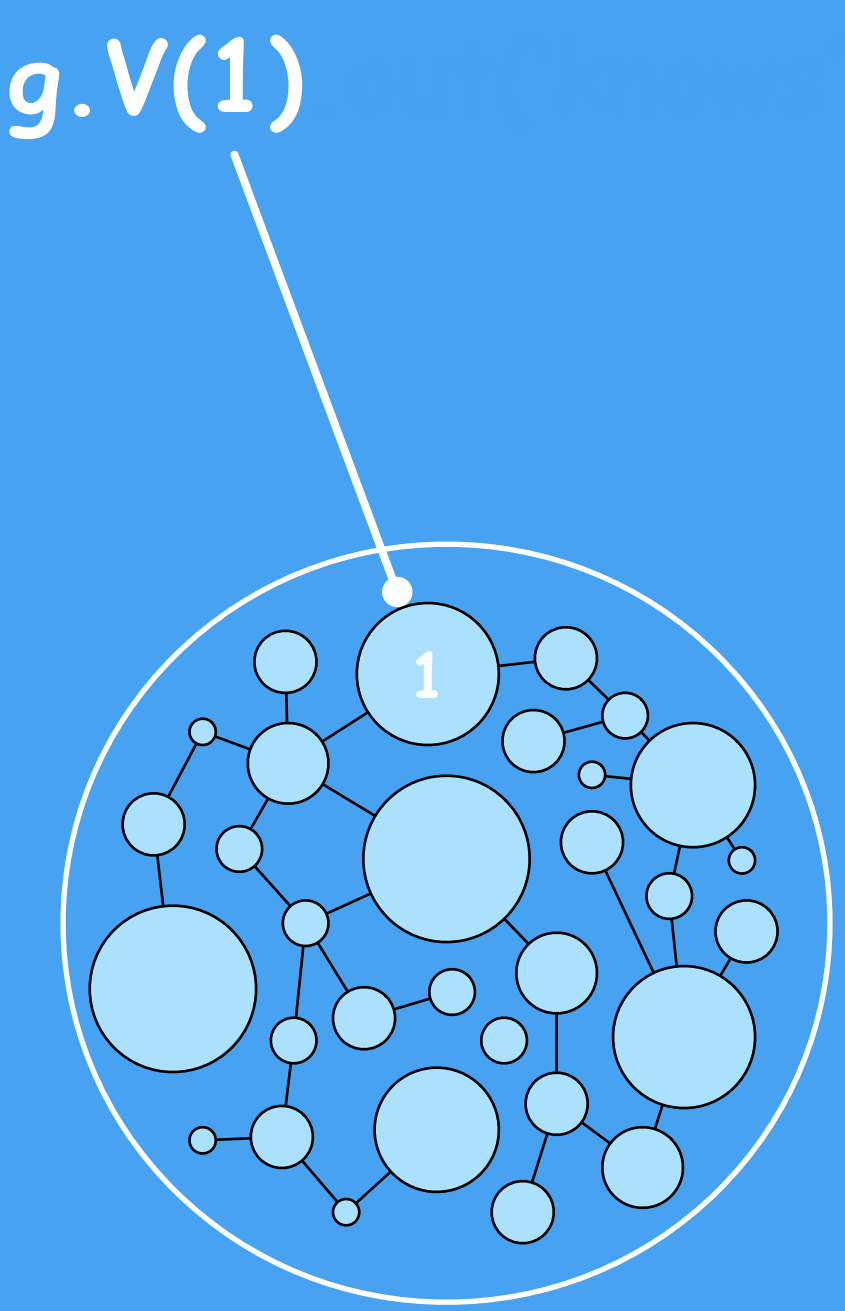
```
SELECT p2.*
      FROM person p1
INNER JOIN knows k
        ON k.out = p1.id
INNER JOIN person p2
        ON p2.id = k.in
     WHERE p1.id = 1
```



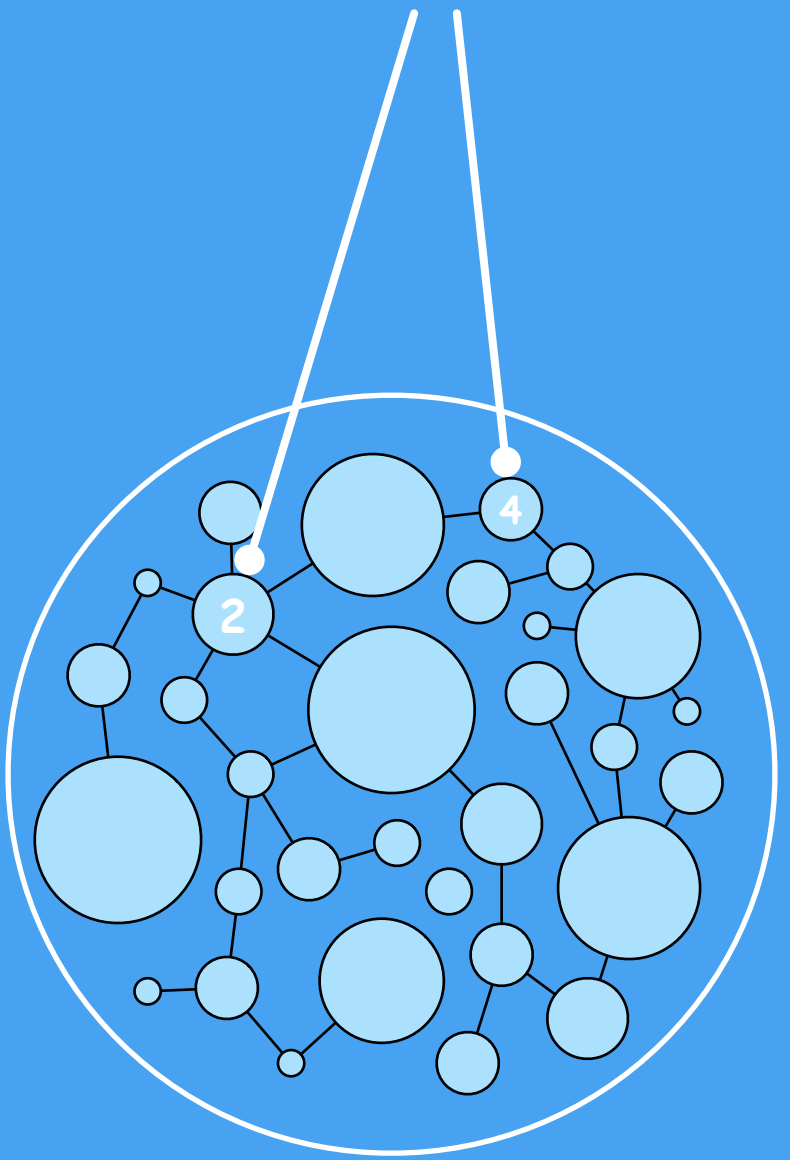
g.V(1).out('knows')



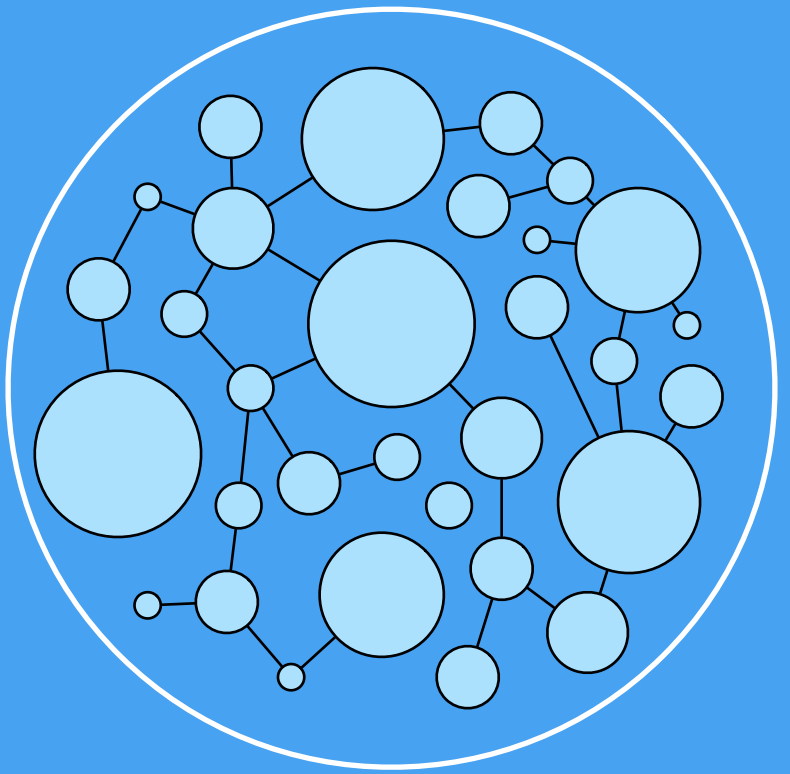

g

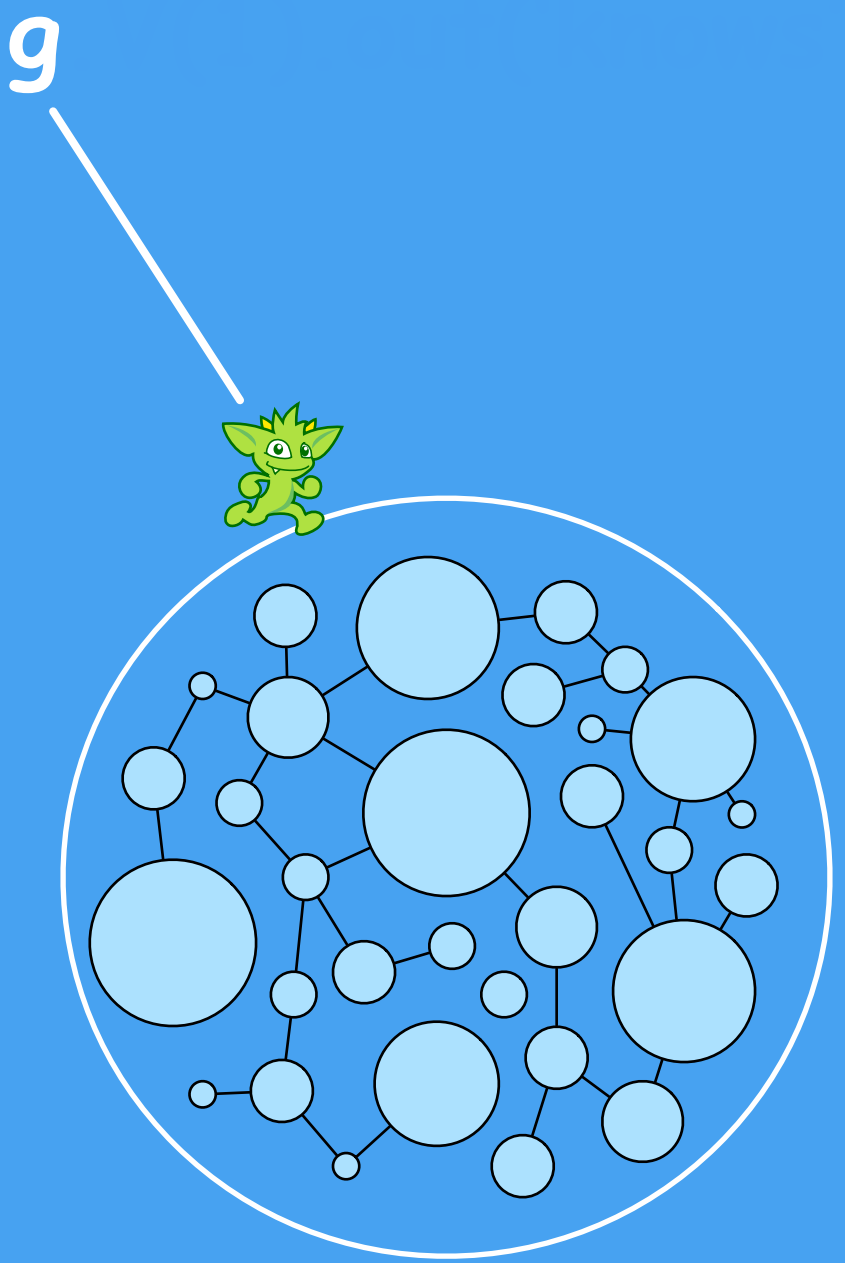

g.V(1)
1

g.V(1).out('knows')

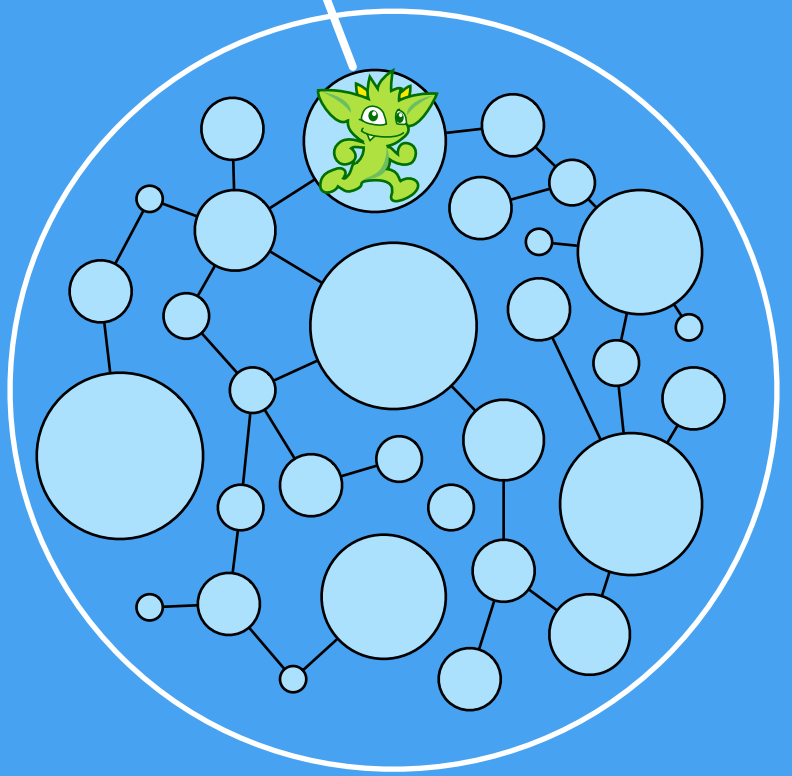


g.V(1).out('knows')

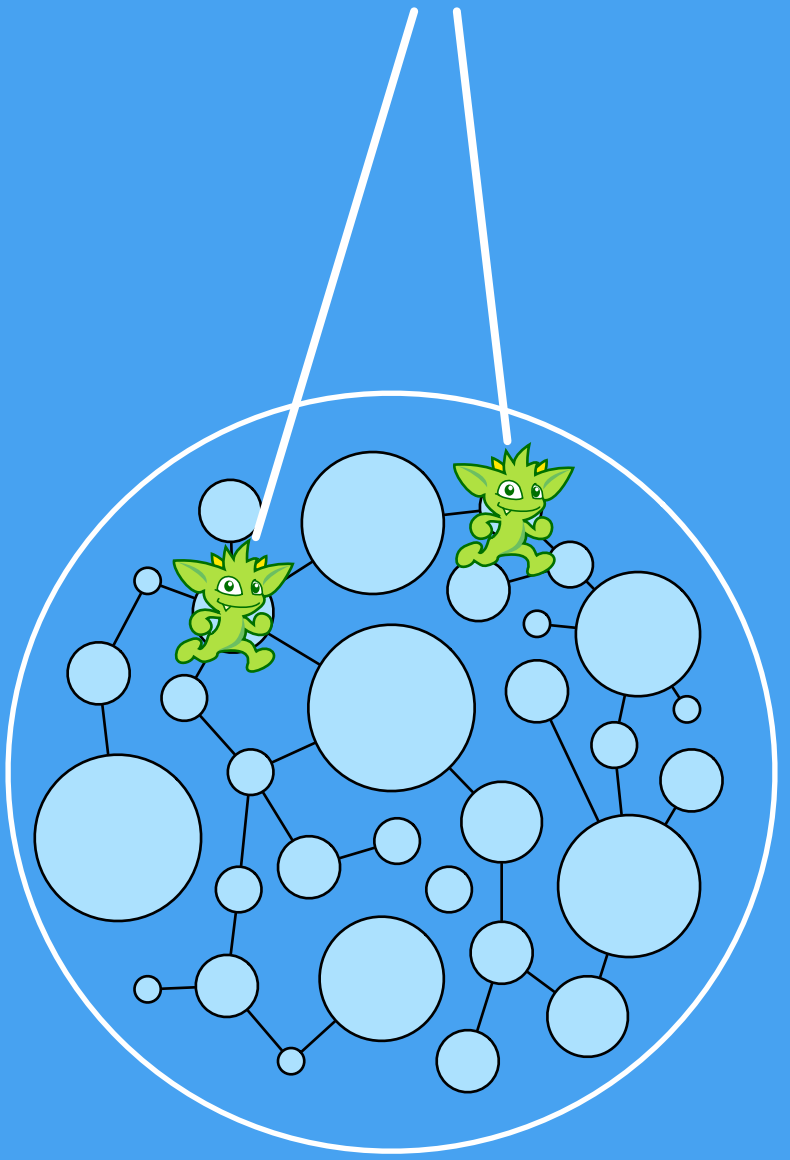



g

g.V(1)



g.V(1).out('knows')



$$G \longleftarrow_{\mu} t \in T \overset{\psi}{\longrightarrow} \Psi$$

Rodriguez, M.A., "The Gremlin Graph Traversal Machine and Language," ACM Proceedings of the 15th Symposium on Database Programming Languages, 2015.

http://arxiv.org/abs/1508.03843

Traversers

$$G \longleftarrow \mu \;\; t \in T \;\; \psi \longrightarrow \Psi$$

Graph Traversal

Traversers

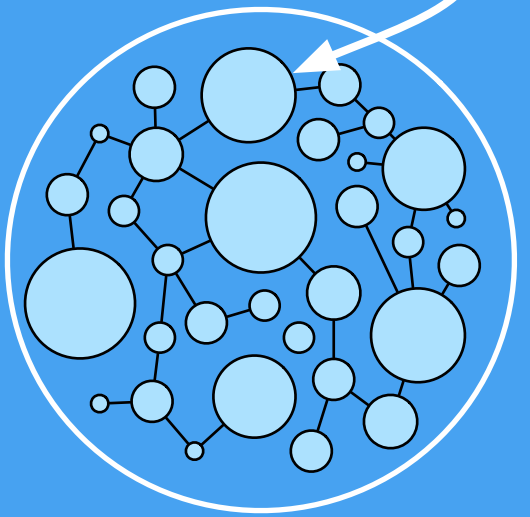

$G \longleftarrow \mu \; t \in T \; \psi \longrightarrow \Psi$
Graph
object
traverser
step
Traversal

Ψ
Traversal
Steps, SideEffects

T
Traversers
Locations, Bulk, Sack, Path, Loops

G
Graph
Vertices, Edges, Properties

g.V(1).out('knows')

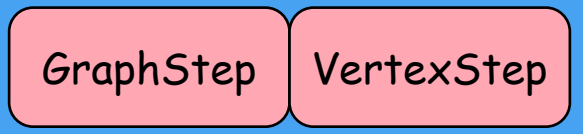


The Traversal

g.V(1).out('knows')

g.V(1).out('knows')

compiles to



g.V(1).out('knows')

compiles to

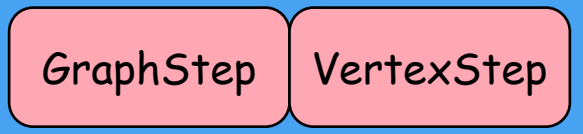
GraphStep | VertexStep

executes as

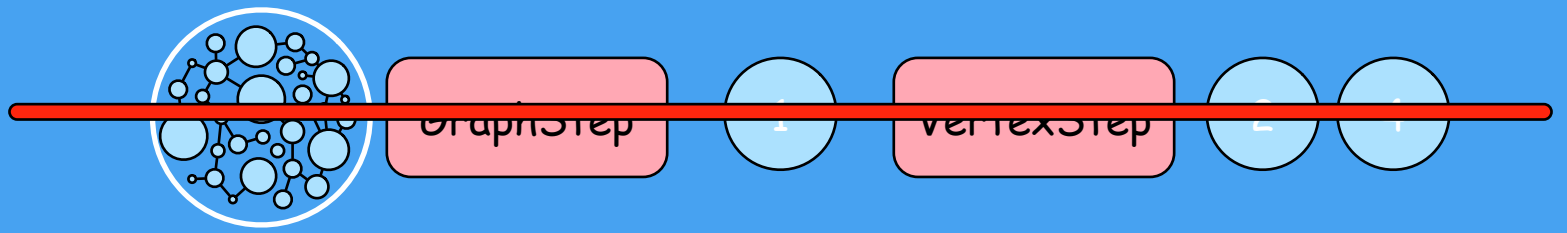
GraphStep → 1 → VertexStep → 2 → 4

g.V(1).out('knows')

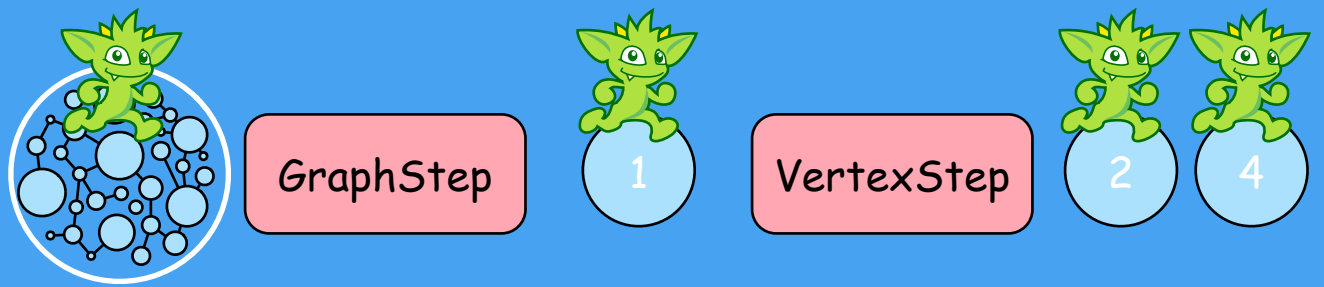
compiles to



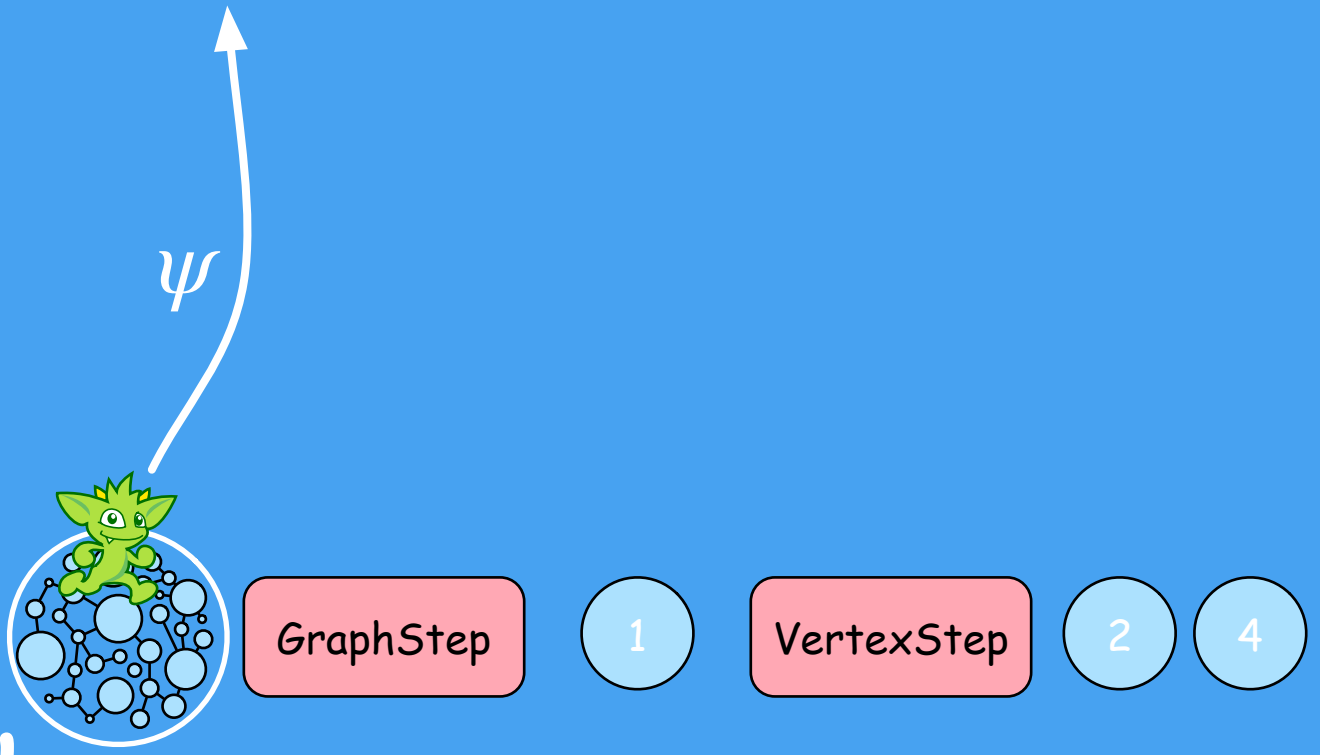
executes as



executes as

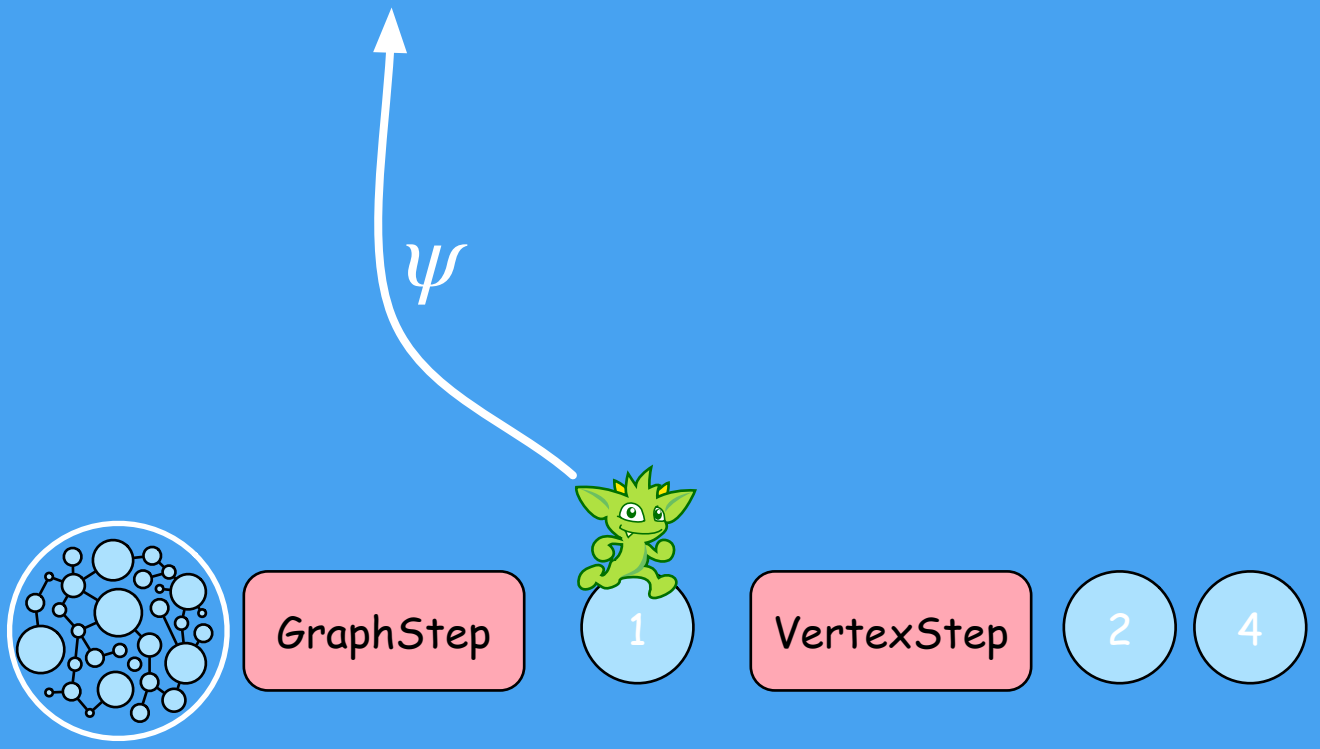


g.V(1).out('knows')



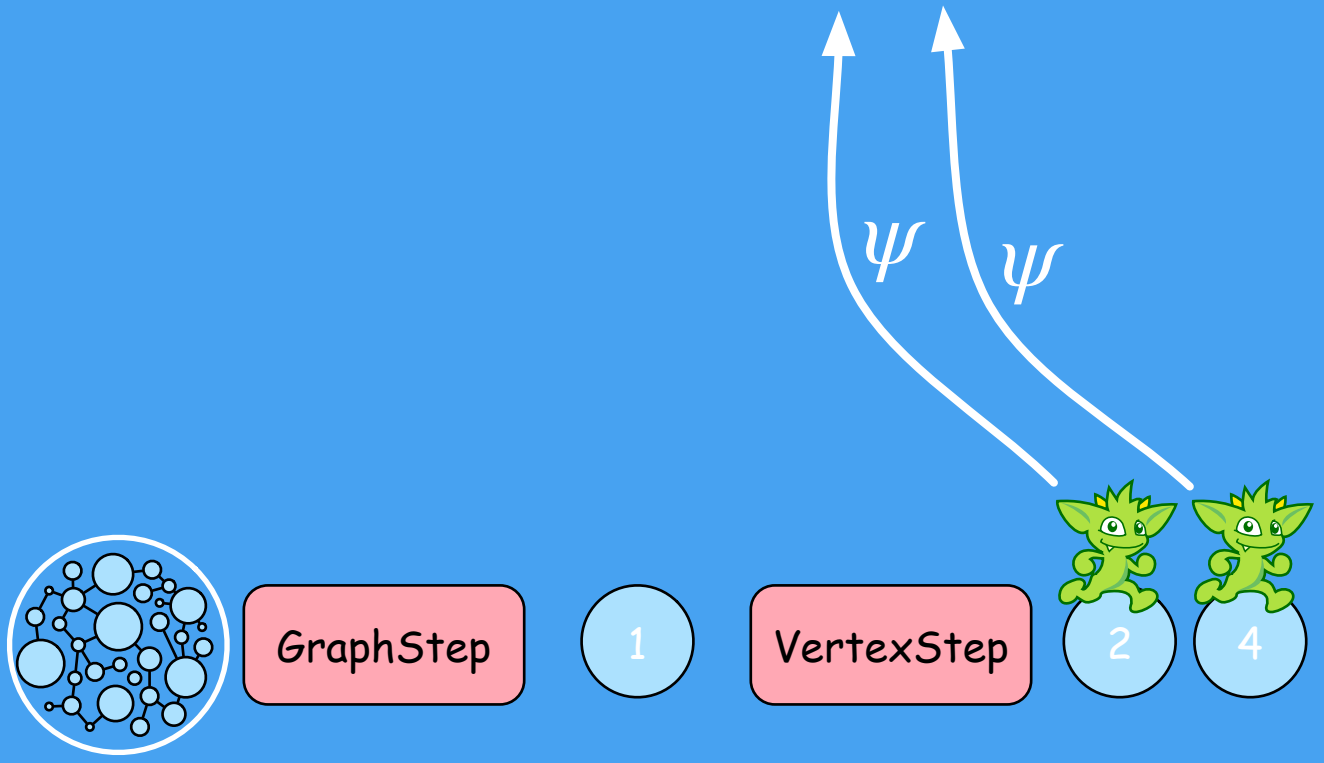
→Graph

g.V(1).out('knows')



Graph → Vertex

g.V(1).out('knows')



Vertex→Vertex

g.V(1).out('knows')

$$\mathrm{V}_1 : G \rightarrow V$$

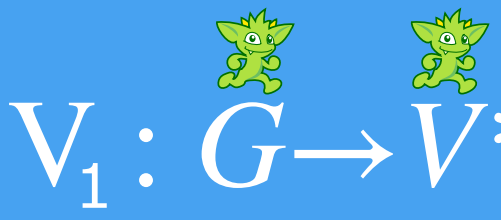
$$\mathrm{out}_{\mathrm{knows}} : V \rightarrow V$$

g.V(1).out('knows')

$V_1 : G \rightarrow V^*$

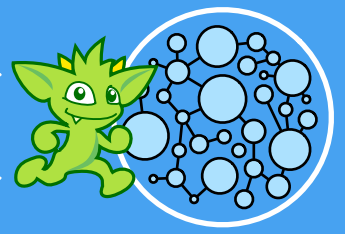
$out_{knows} : V \rightarrow V^*$

g.V(1).out('knows')



g.V(1).out('knows')

$V_1 : G \rightarrow V^*$

$V_1($  $)$

$out_{knows} : V \rightarrow V^*$

g.V(1).out('knows')

$V_1 : G \rightarrow V^*$

$\text{out}_{\text{knows}} : V \rightarrow V^*$

g.V(1).out('knows')

$V_1 : G \rightarrow V^*$

$out_{knows} : V \rightarrow V^*$

$out_{knows}($  $)$

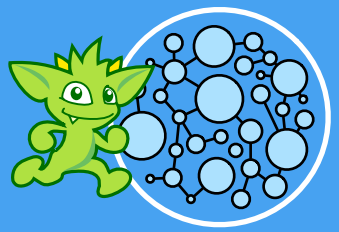
g.V(1).out('knows')

$V_1 : G \rightarrow V^*$

$\text{out}_{\text{knows}} : V \rightarrow V^*$

g.V(1).out('knows')

$$\text{out}_{\text{knows}}(\text{V}_1(\quad)) =$$

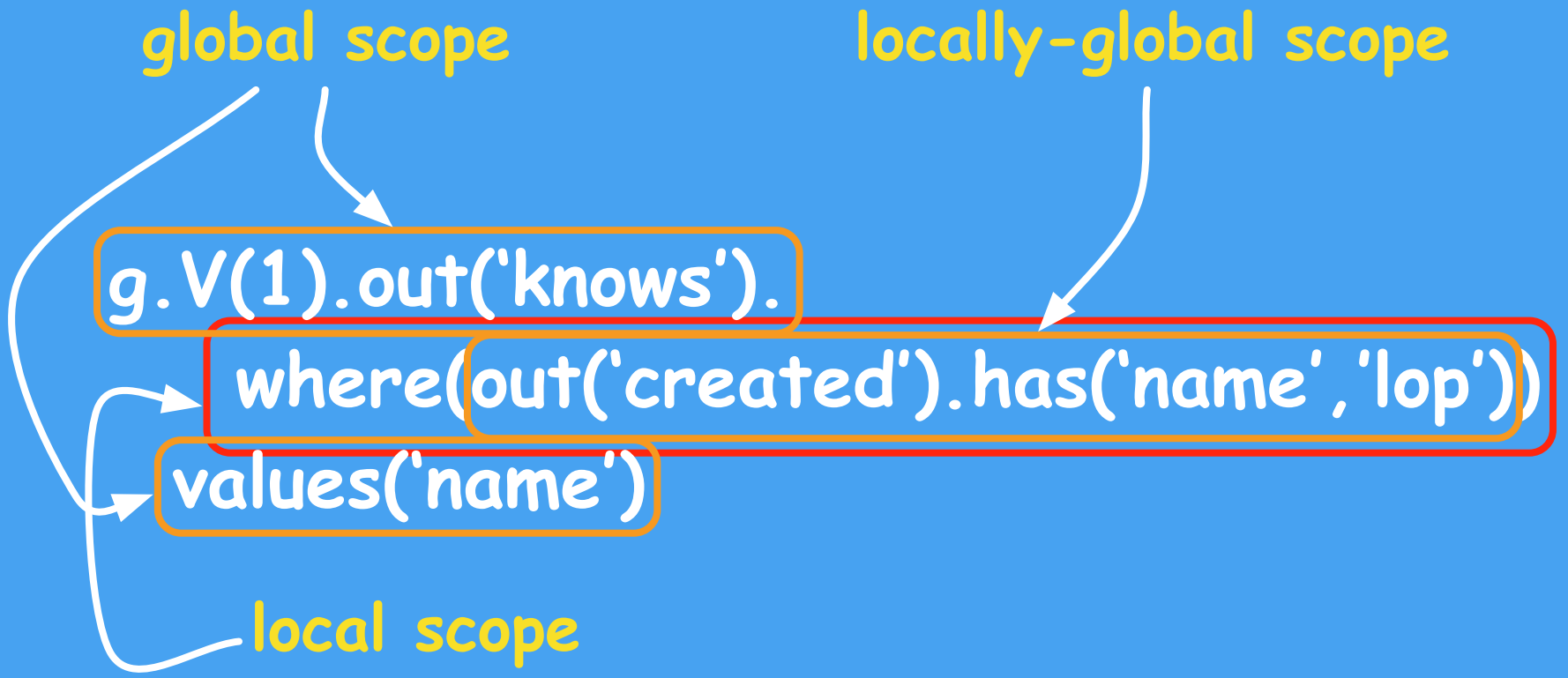


What are the names of the people that person 1 knows who created LinkedProcess (lop)?

```
g.V(1).out('knows').where(out('created').has('name','lop')).values('name')
```

What are the names of the people that person 1 knows who created LinkedProcess (lop)?

```
g.V(1).out('knows').
    where(out('created').has('name','lop'))
  values('name')
```

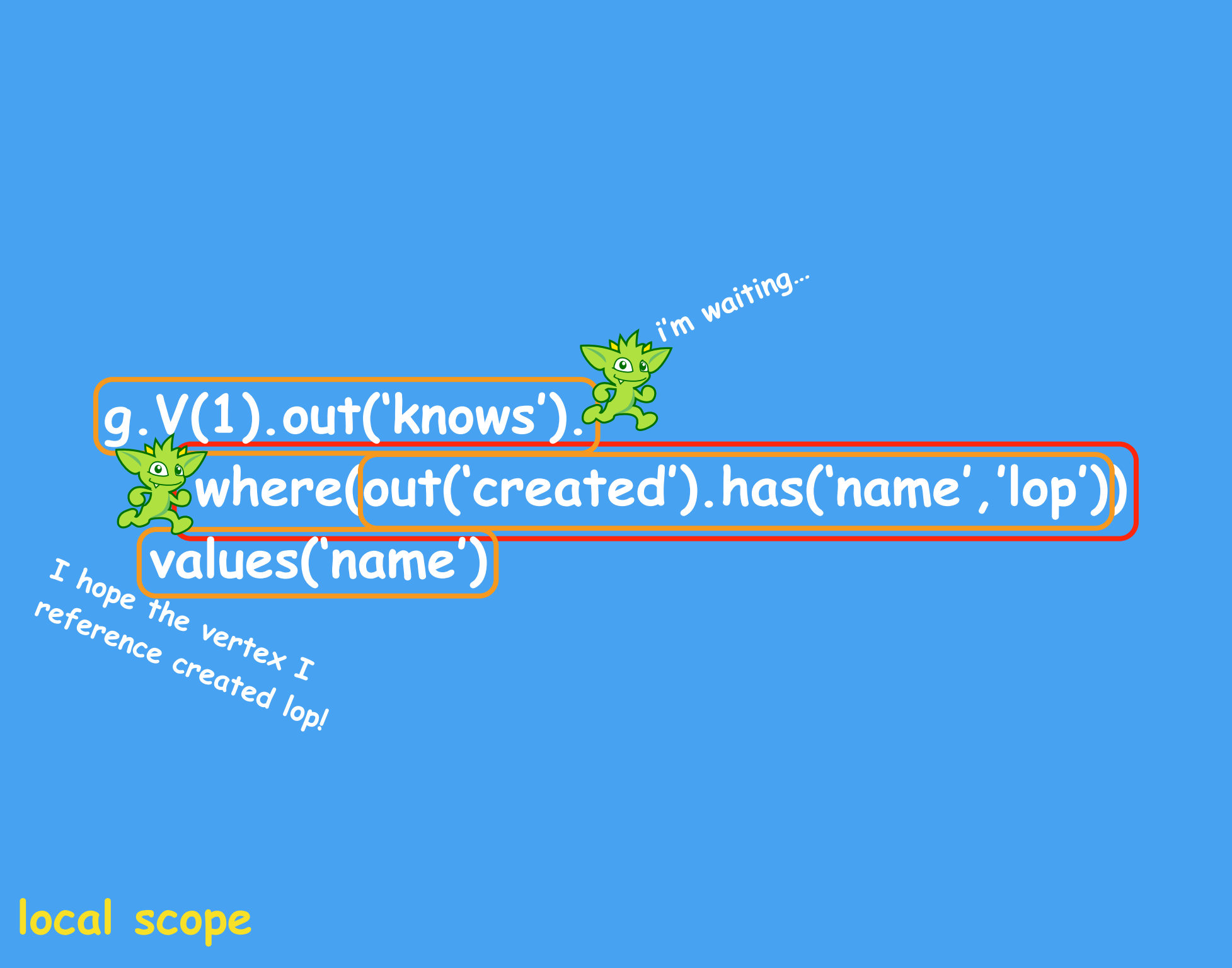

global scope
locally-global scope
g.V(1).out('knows').
where(out('created').has('name','lop'))
values('name')
local scope

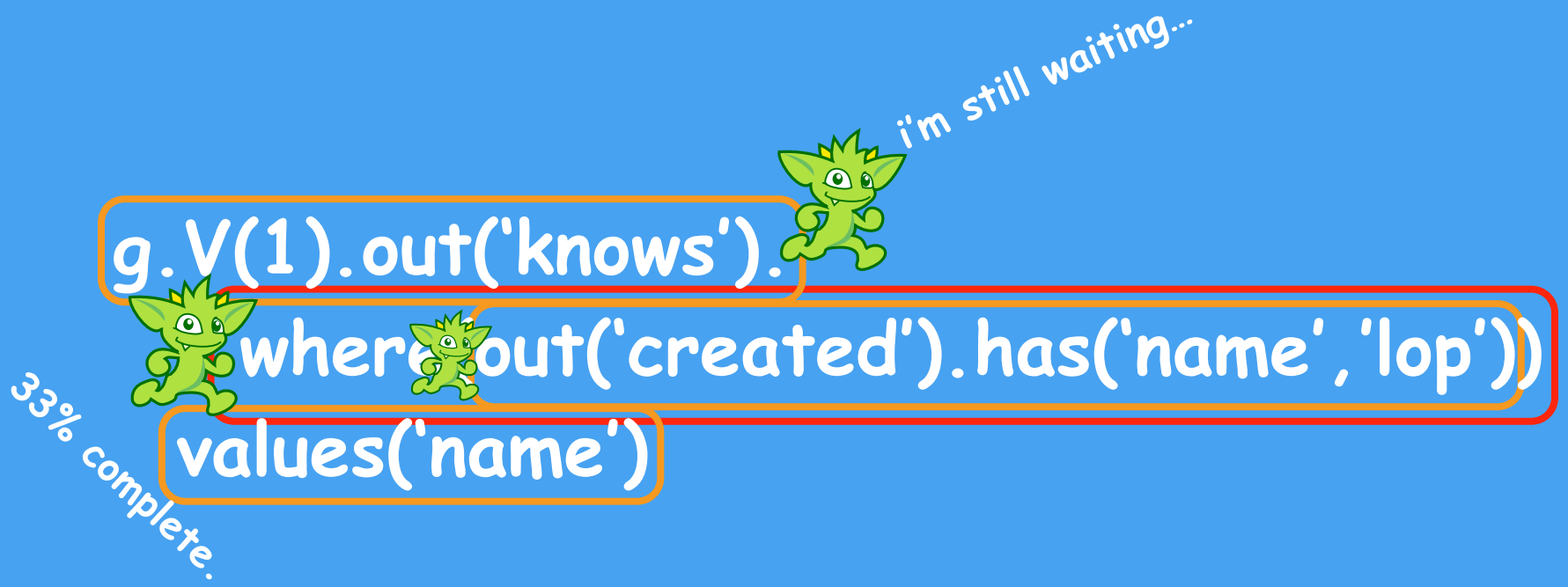
```
g.V(1).out('knows').
    where(out('created').has('name','lop')).
  values('name')
```

global scope

```
g.V(1).out('knows').
    where(out('created').has('name','lop'))
  values('name')
```

global scope

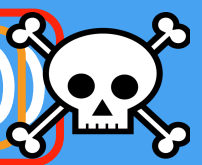

i'm waiting...
g.V(1).out('knows').
where(out('created').has('name','lop'))
values('name')
I hope the vertex I reference created lop!
local scope



locally-global scope

locally-global scope


```
g.V(1).out('knows').
  where(out('created').has('name','lop')).
  values('name')
```



local scope

```
g.V(1).out('knows').
  where(out('created').has('name','lop')).
  values('name')
```

my turn finally!

local scope

```
g.V(1).out('knows').
    where(out('created').has('name','lop')).
    values('name')
```

prosper my child.

locally-global scope

```
g.V(1).out('knows').
  where(out('created').as('name','lop')).
  values('name')
```



one successful offspring is all I need.

locally-global scope

```
g.V(1).out('knows').
  where(out('created').as('name','lop
  values('name')
```

woo hoo!

boo yea!

live and learn :(

locally-global scope

```
g.V(1).out('knows').
  where(out('created').has('name','lop')).
  values('name')
```

I reference a vertex that created lop!

local scope

```
g.V(1).out('knows').
    where(out('created').has('name','lop')).
    values('name')
```

global scope

```
g.V(1).out('knows').
    where(out('created').has('name','lop')).
    values('name')
```

I reference a result.

global scope

Gremlin Language Traversal

```
g.V(1).out('knows').
    where(out('created').has('name','lop'))
  values('name')
```

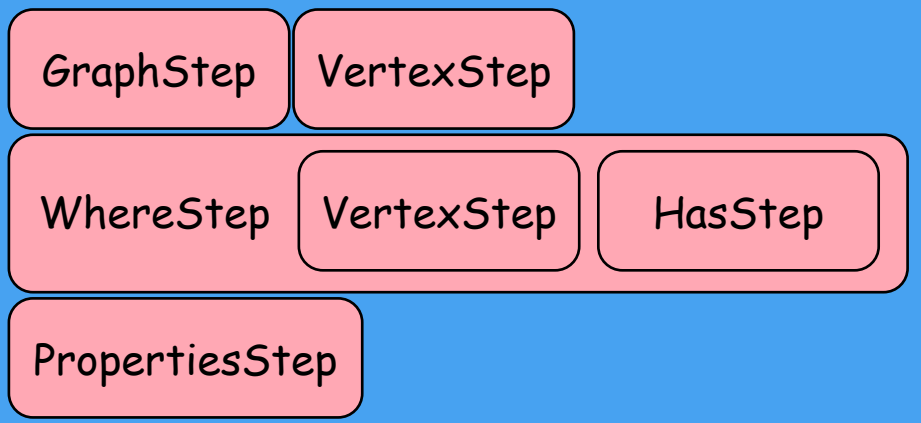
traversal = op((object | traversal)*)*

Gremlin Bytecode Traversal

```
[[V,1][out,knows]
  [where,[[out,created],[has,name,lop]]]
[values,name]]
```

traversal = [[op,(object | traversal)*]*]

Gremlin Machine Traversal



traversal = step[(object | traversal)*]*

language

```
g.V().has('name','marko').
  repeat(outE().identity().inV()).times(2).
  name.groupCount()
```

translate

bytecode

```
[[V],[has,name,eq(marko)],
 [repeat,[[outE],[identity],[inV]]],[times,2],
 [values,name],[groupCount]]
```

compile

machine code

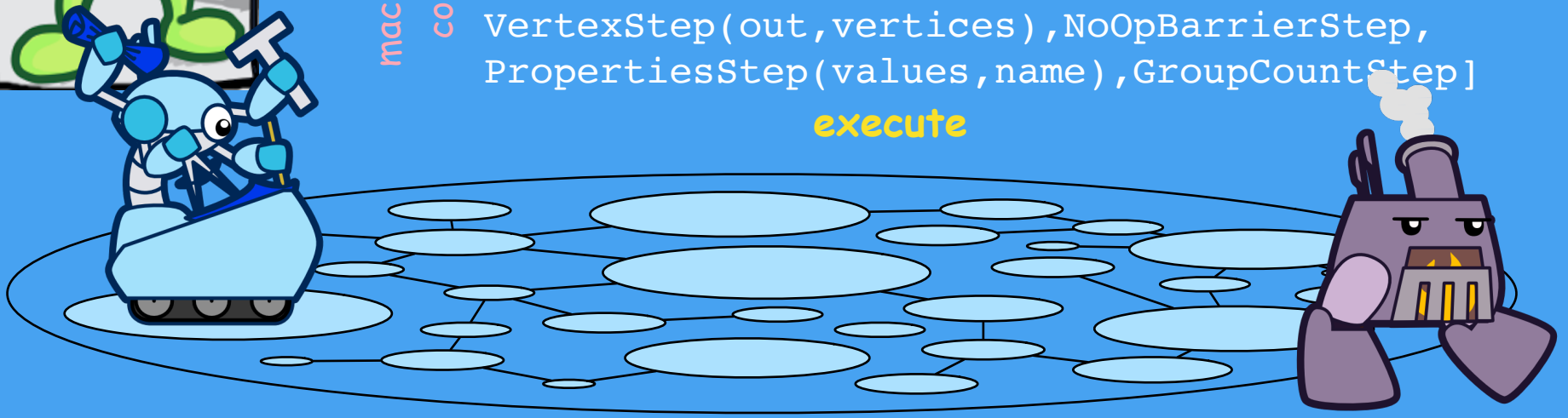
```
[GraphStep,HasStep(name,eq(marko)),
 RepeatStep([VertexStep(out,edges),IdentityStep,
             EdgeVertexStep(in)],2),
 PropertiesStep(values,name),GroupCountStep]
```

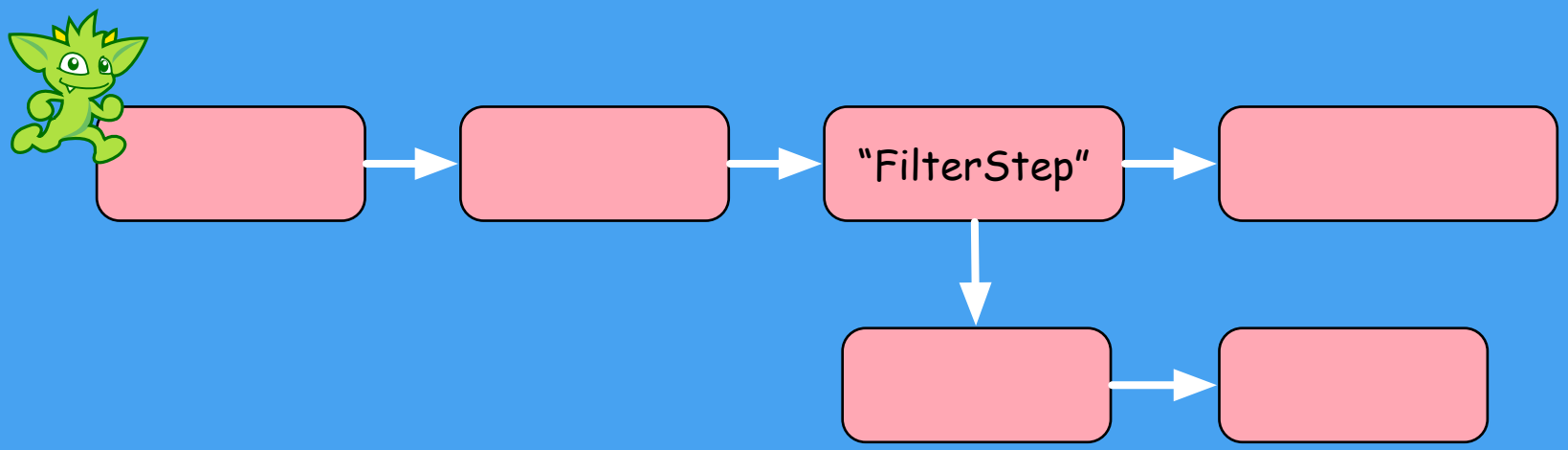
optimize

machine code

```
[ProviderGraphStep(name,eq(marko)),
 VertexStep(out,vertices),NoOpBarrierStep,
 VertexStep(out,vertices),NoOpBarrierStep,
 PropertiesStep(values,name),GroupCountStep]
```

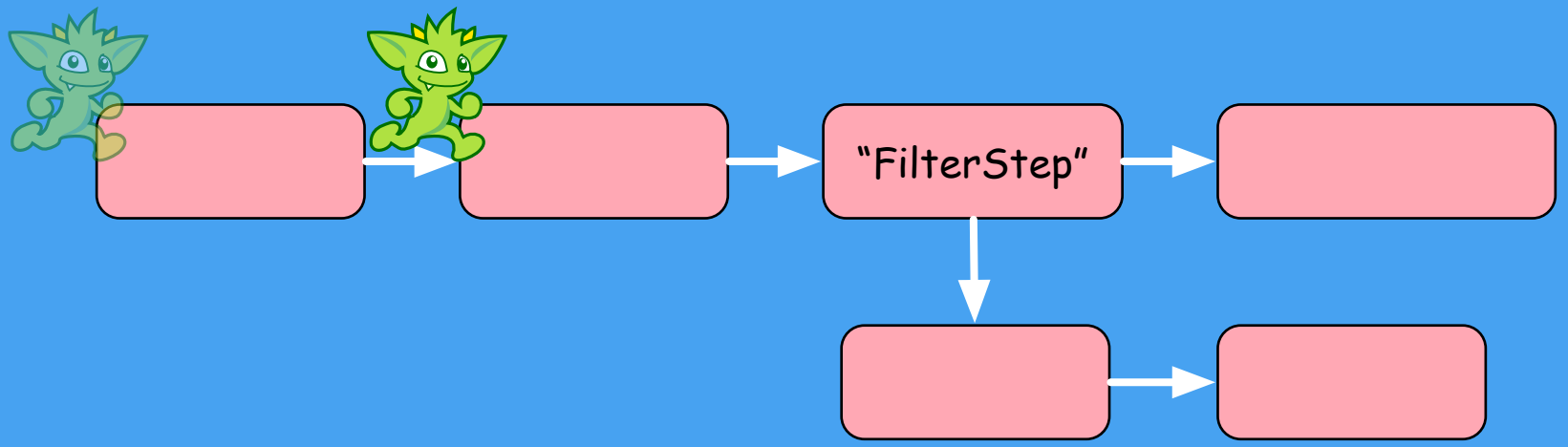
execute

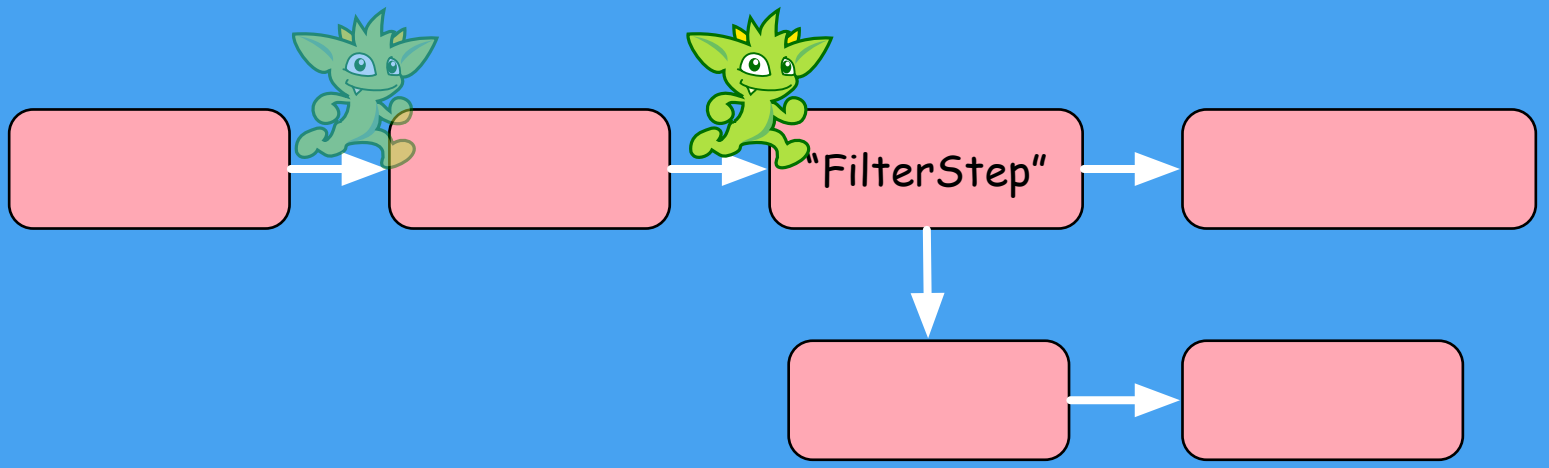


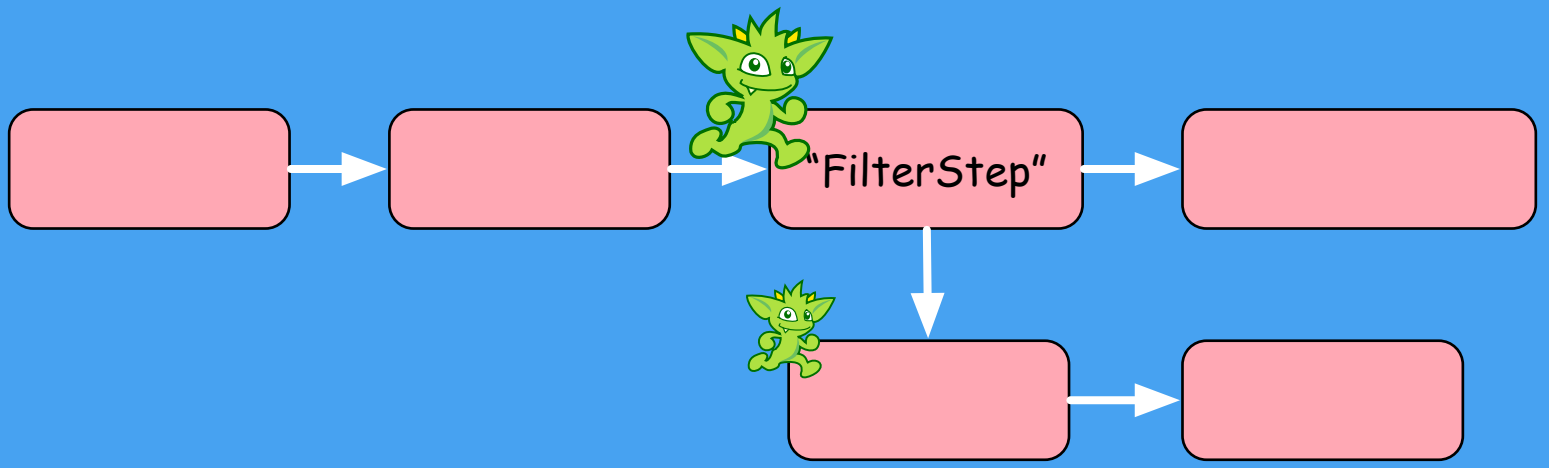


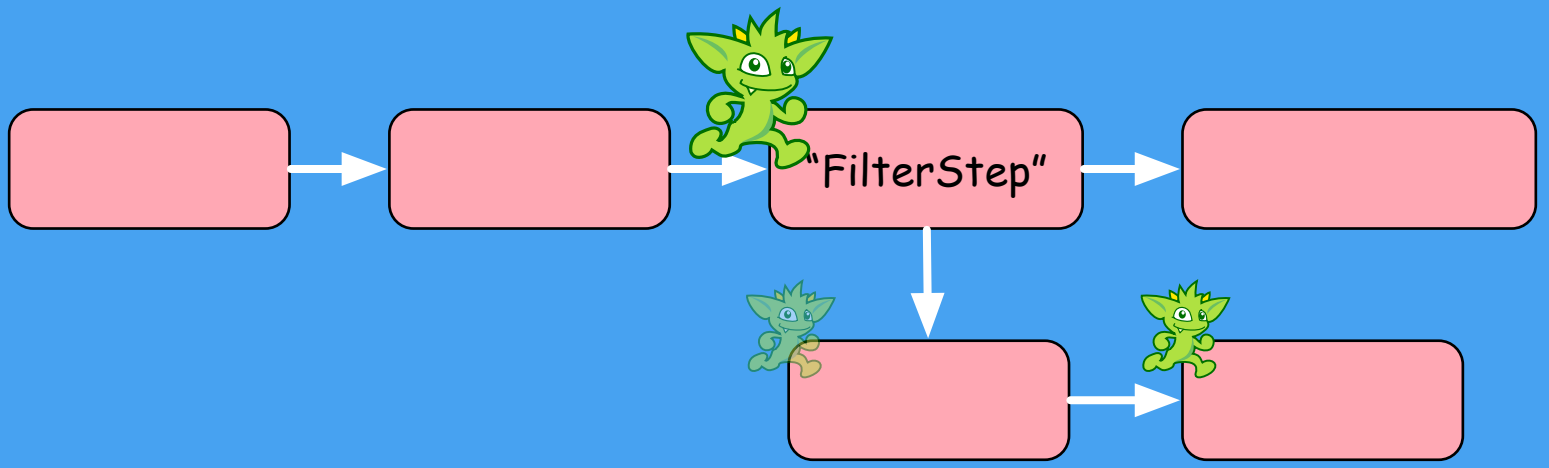
Rodriguez, M.A., "A Gremlin Implementation of the Gremlin Traversal Machine," DataStax Engineering Blog, October 2016.

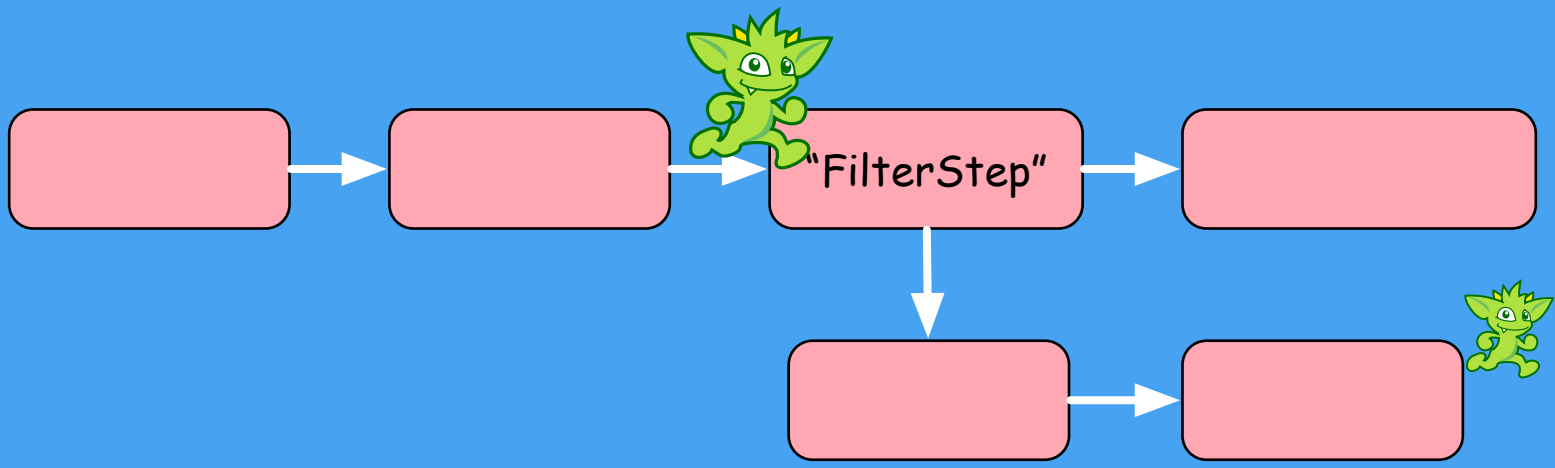
https://www.datastax.com/dev/blog/a-gremlin-implementation-of-the-gremlin-traversal-machine

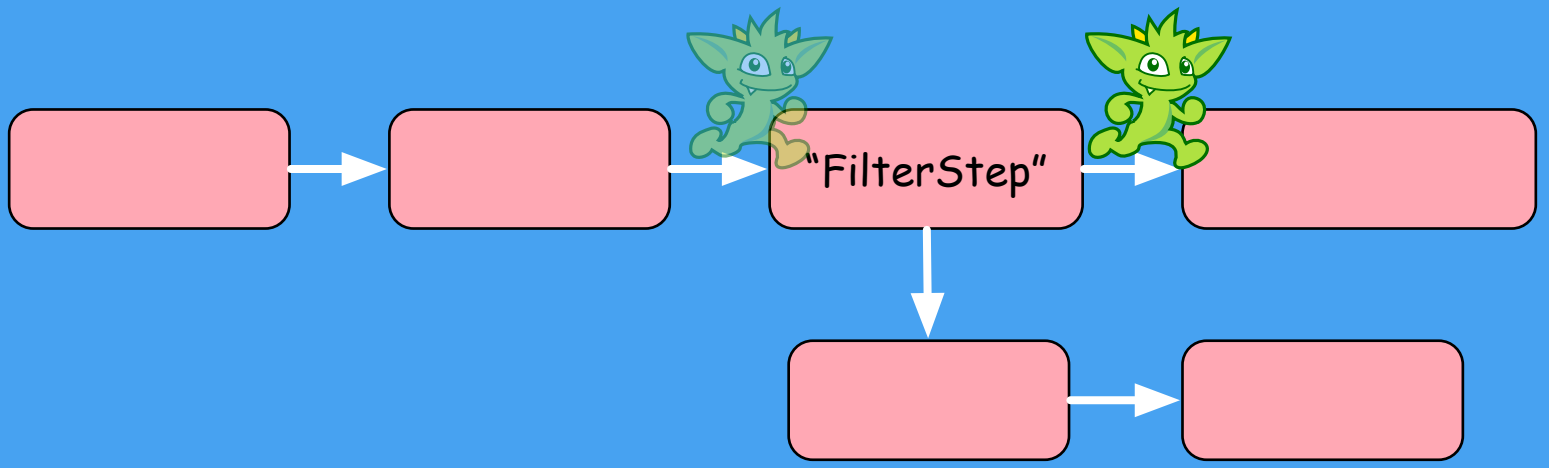

"FilterStep"

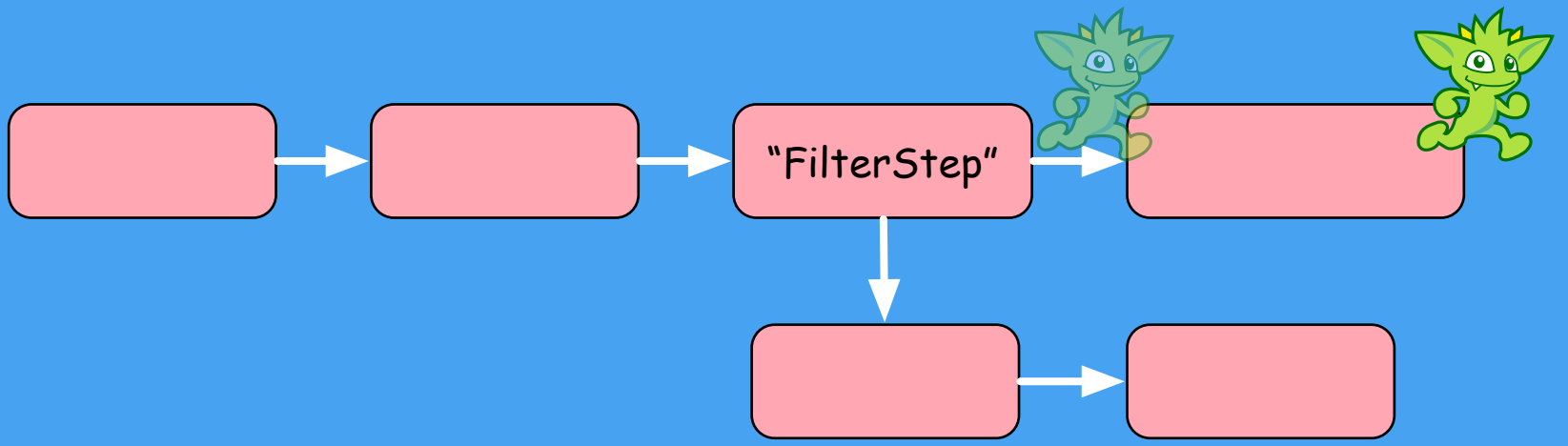

"FilterStep"

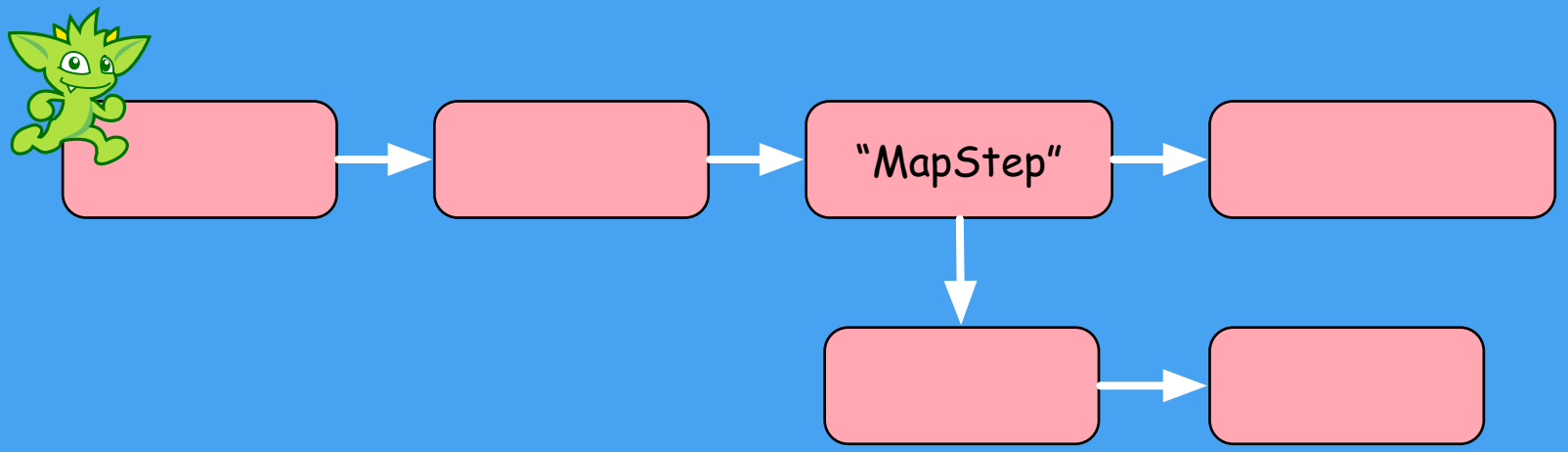

"FilterStep"

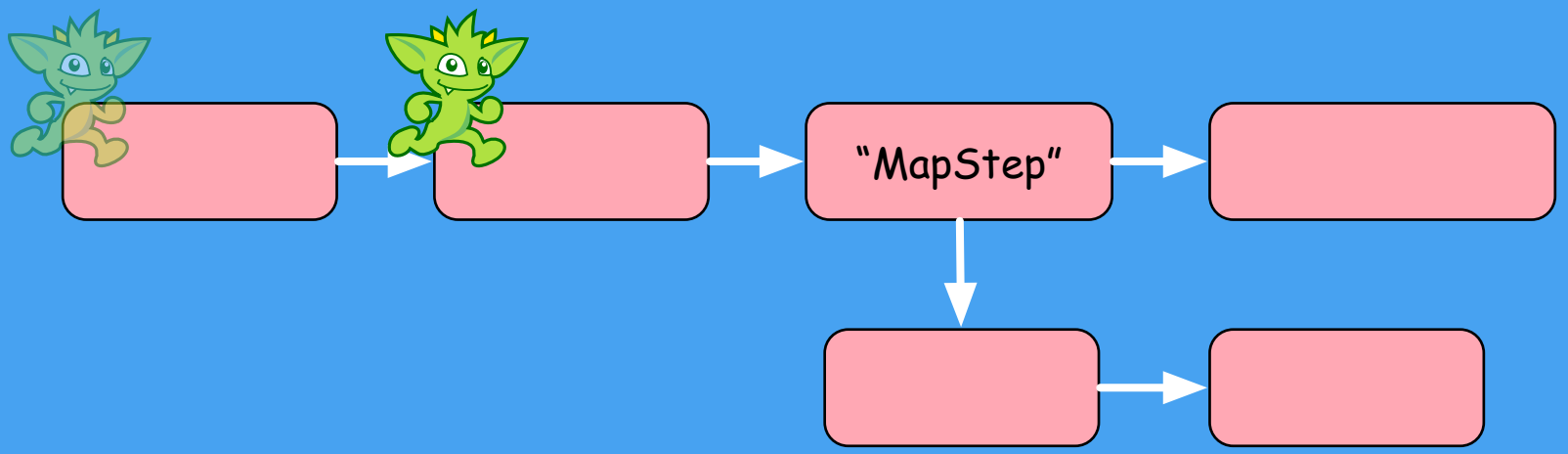

"FilterStep"

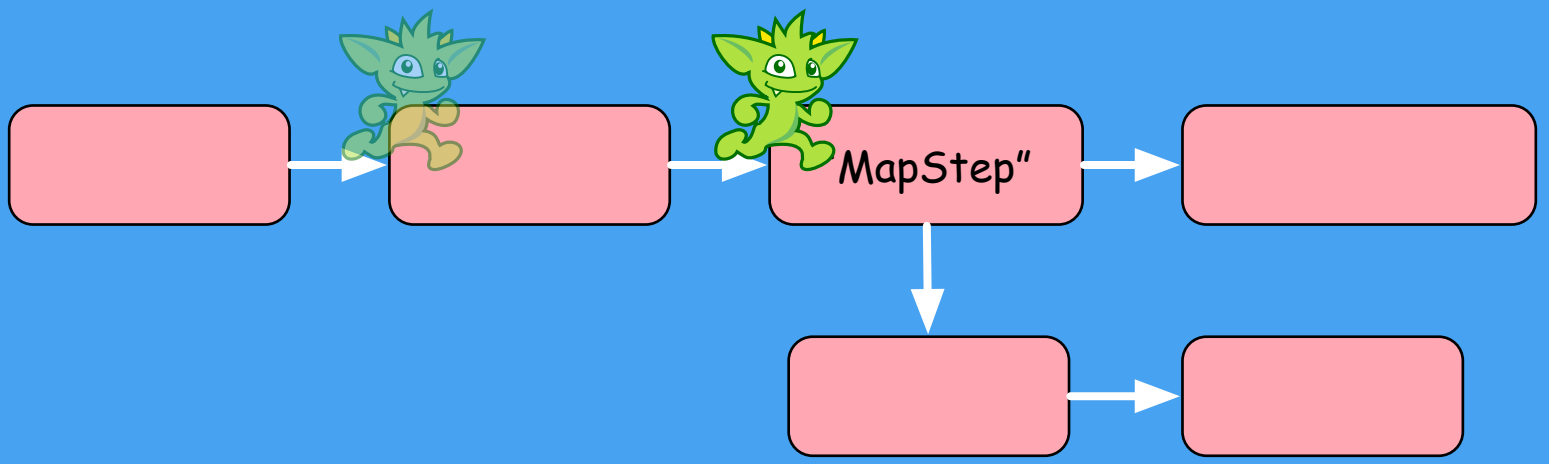

"FilterStep"

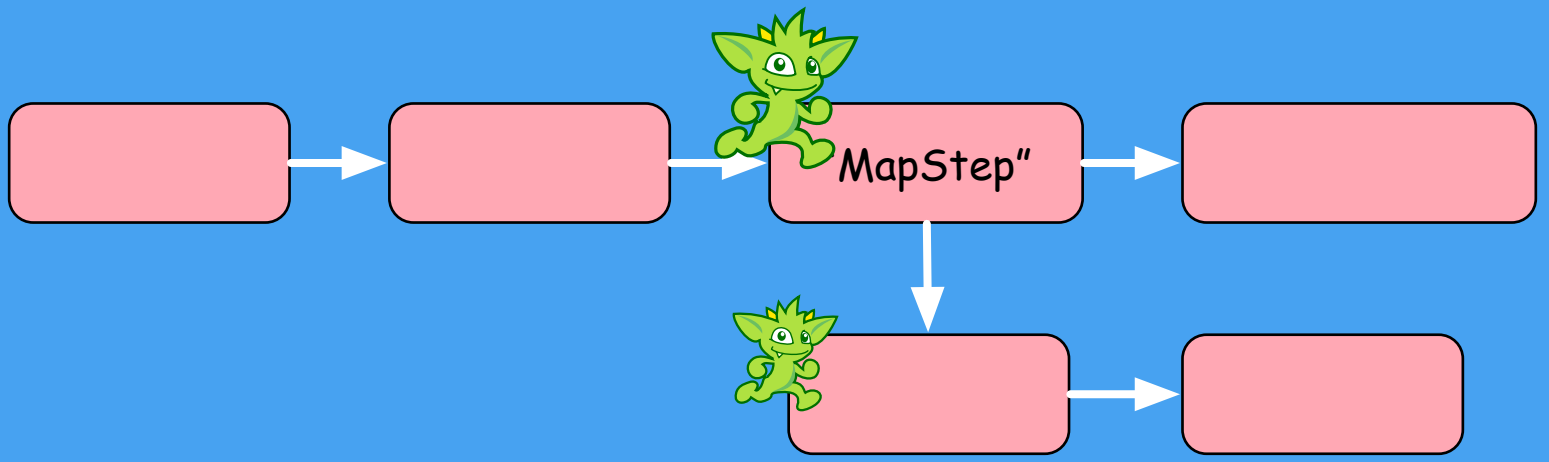

"FilterStep"

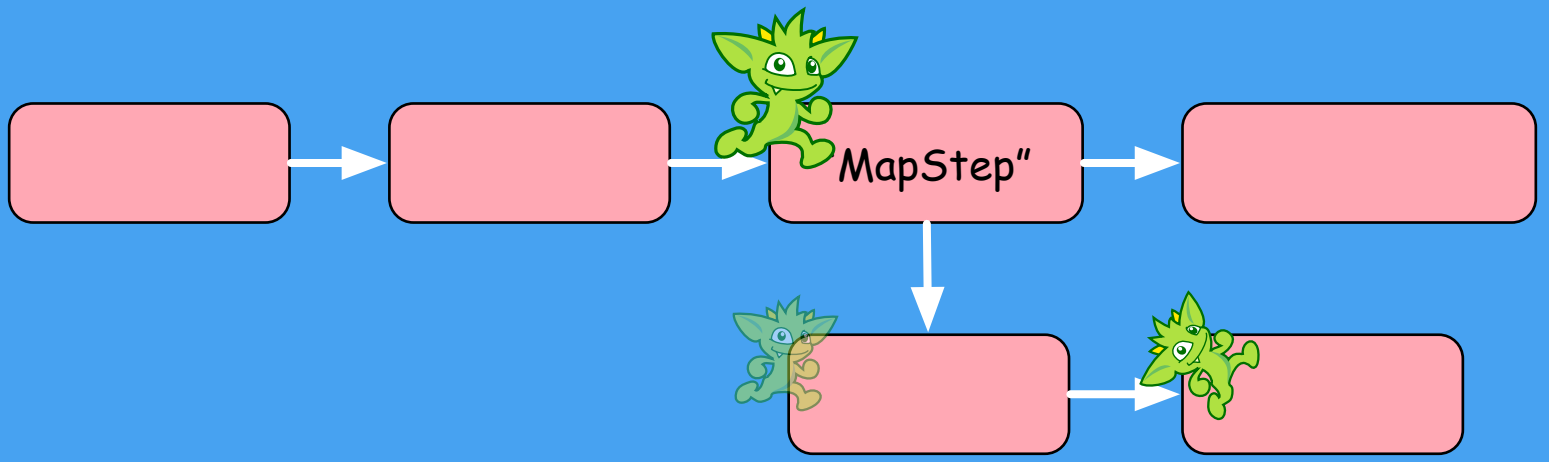

"FilterStep"

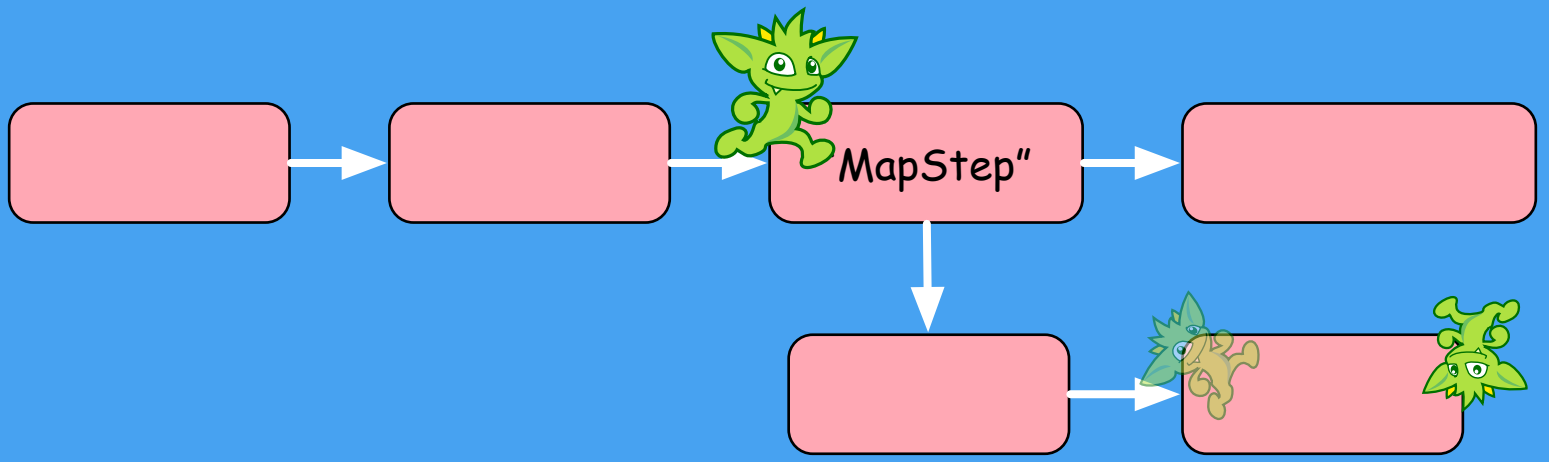

"MapStep"

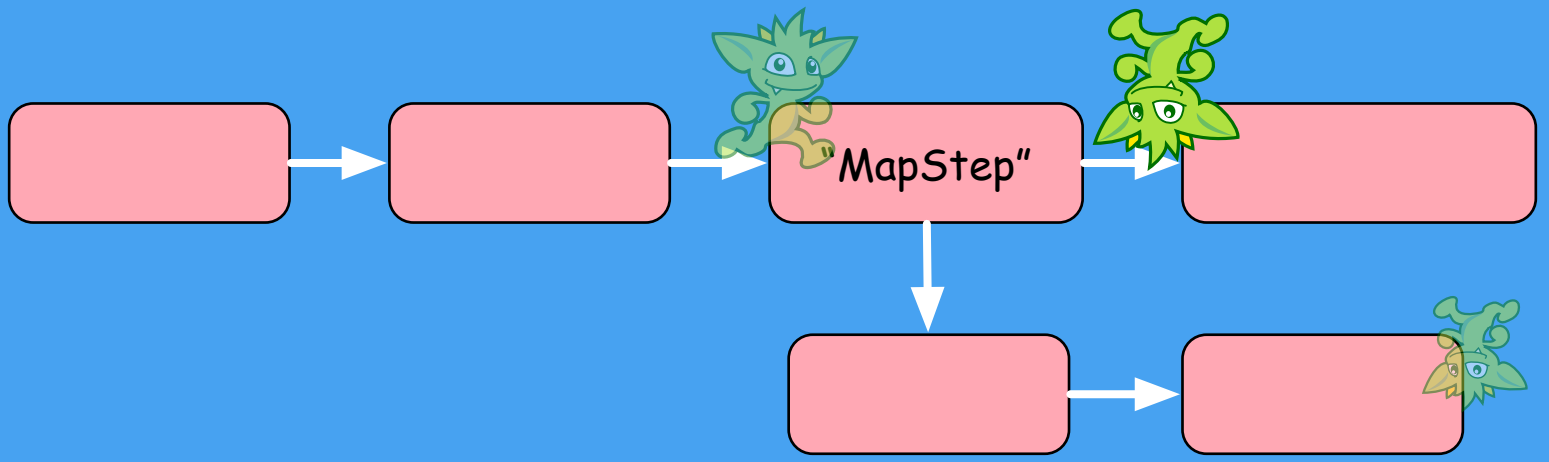

"MapStep"

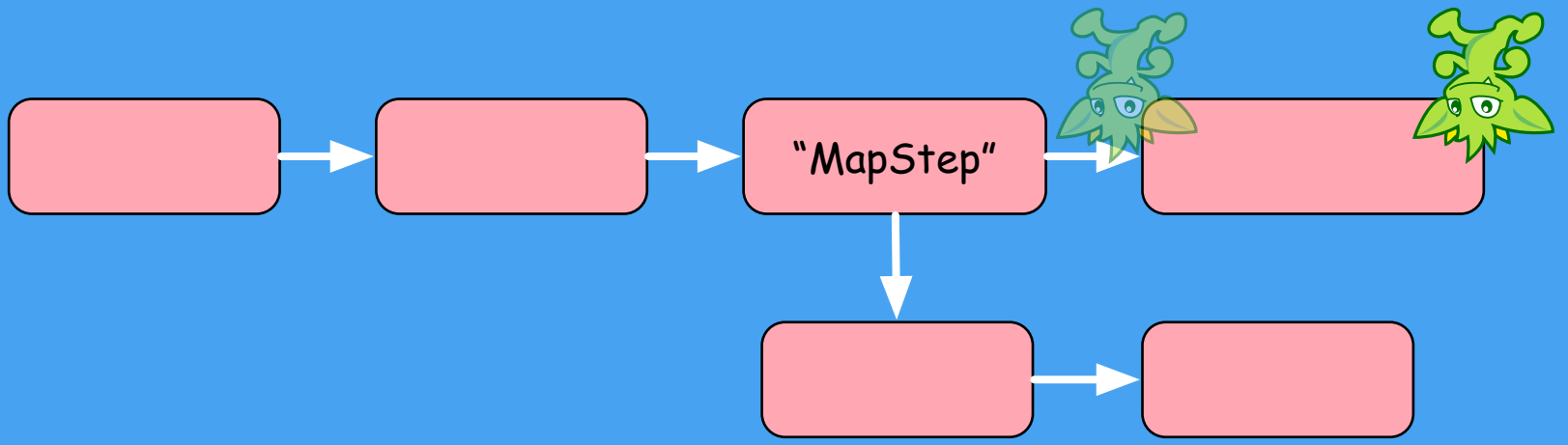

MapStep"


MapStep"

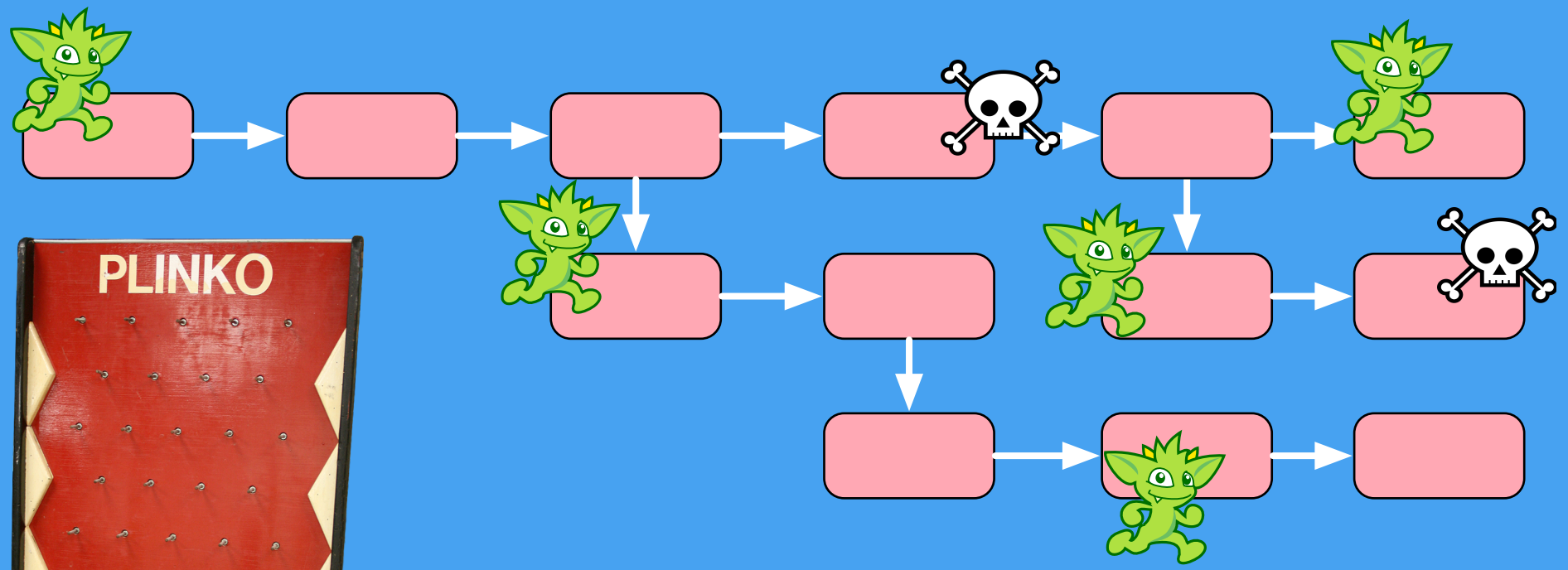

MapStep"


MapStep"


"MapStep"


"MapStep"

Chained and Nested Steps



Step Types

Map: one-to-one

FlatMap: one-to-many

Filter: one-to-one.or.none

SideEffect: one-to[?]-one

Barrier: many-to-many

Reducer: many-to-one

Map

label()

person

Map

label()

person

FlatMap

out('knows')

Map

label()

person

FlatMap

out('knows')

Filter

hasLabel('android')

Map

label()

person

FlatMap

out('knows')

Filter

hasLabel('android')

SideEffect

x=[]

store('x')

Map

label()

person

Barrier

barrier()

FlatMap

out('knows')

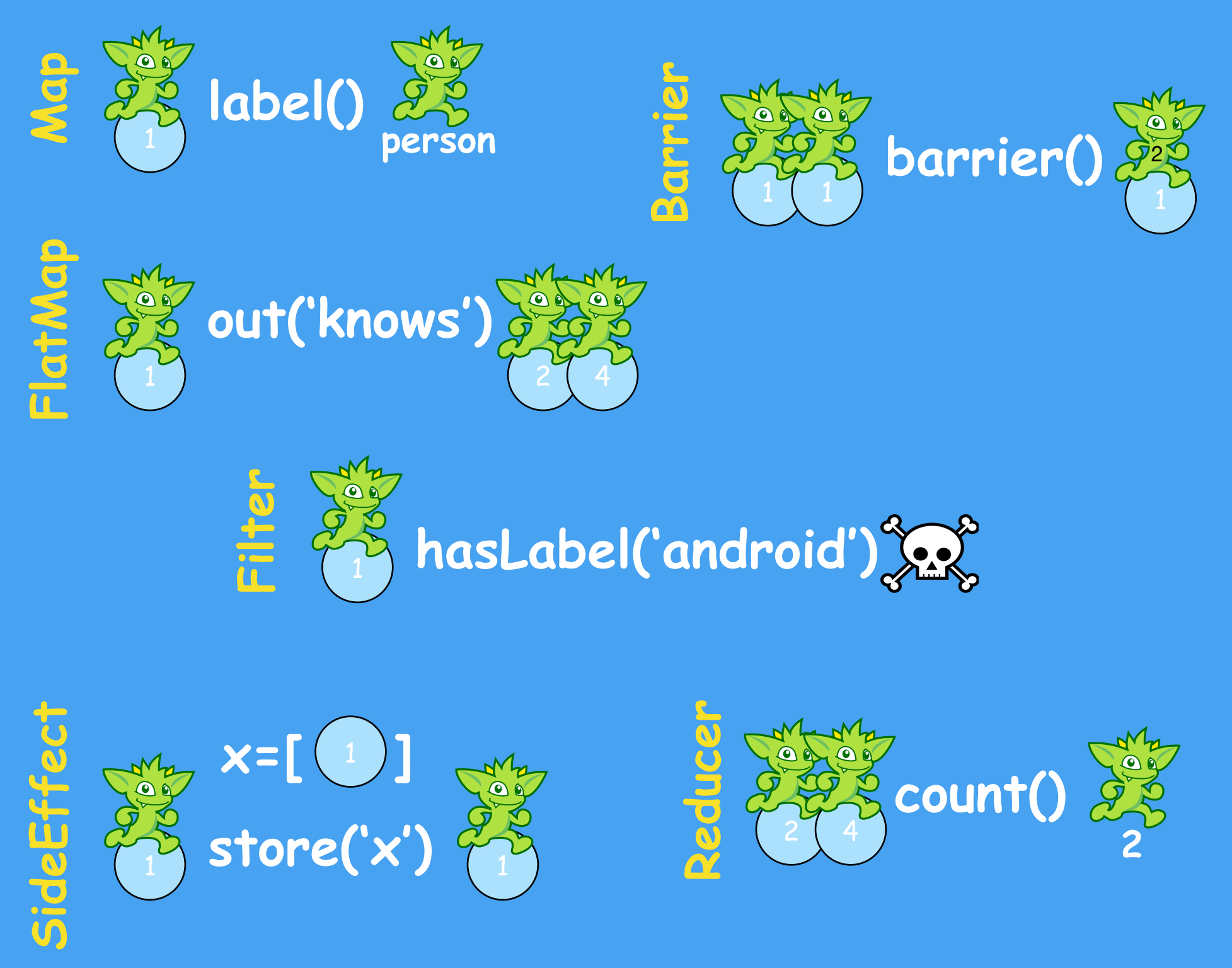
Filter

hasLabel('android')

SideEffect

x=[]

store('x')


Map
label()
person
Barrier
barrier()
FlatMap
out('knows')
Filter
hasLabel('android')
SideEffect
x=[ ]
store('x')
Reducer
count()
2

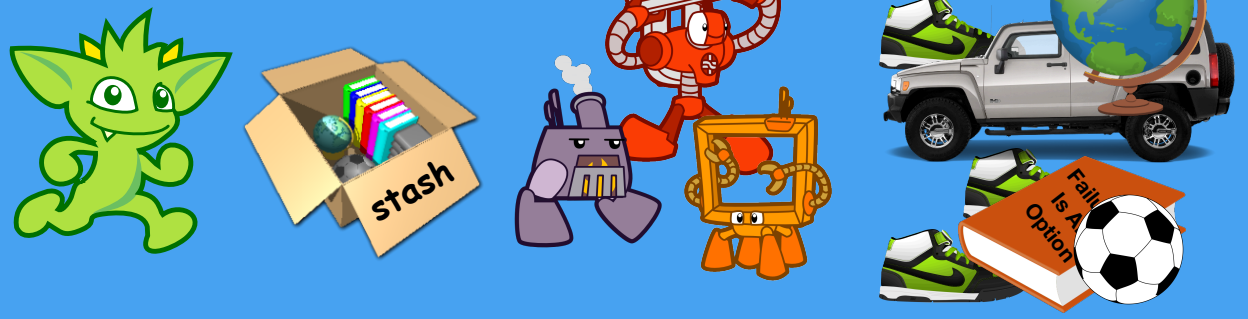
```
g.V().has('name','gremlin')
```

```
g.V().has('name','gremlin').
  out('bought').aggregate('stash')
```

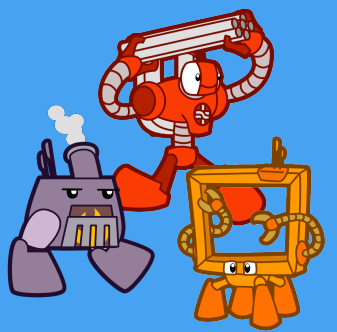
```
g.V().has('name','gremlin').
  out('bought').aggregate('stash').
  in('bought')
```

```
g.V().has('name','gremlin').
  out('bought').aggregate('stash').
  in('bought').has('name',neq('gremlin'))
```

```
g.V().has('name','gremlin').
  out('bought').aggregate('stash').
  in('bought').has('name',neq('gremlin')).
  out('bought')
```

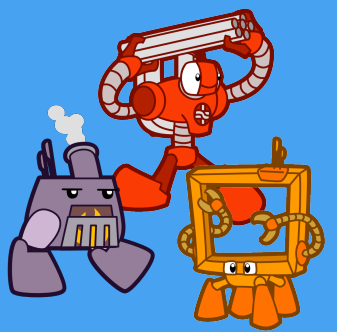


```
g.V().has('name','gremlin').
  out('bought').aggregate('stash').
  in('bought').has('name',neq('gremlin')).
  out('bought').not(within('stash'))
```

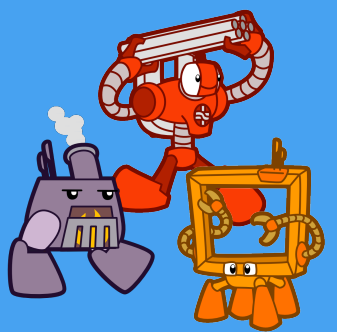


```
g.V().has('name','gremlin').
  out('bought').aggregate('stash').
  in('bought').has('name',neq('gremlin')).
  out('bought').not(within('stash')).
  groupCount()
```

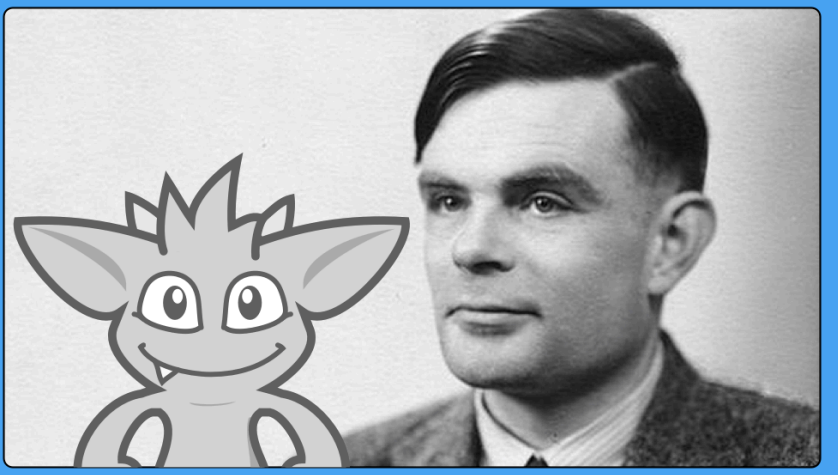
```
g.V().has('name','gremlin').
  out('bought').aggregate('stash').
  in('bought').has('name',neq('gremlin')).
  out('bought').not(within('stash')).
  groupCount().
  order(local).by(values,decr)
```



```
g.V().has('name','gremlin').
  out('bought').aggregate('stash').
  in('bought').has('name',neq('gremlin')).
  out('bought').not(within('stash')).
  groupCount().
  order(local).by(values,decr).
  select(keys).unfold().limit(1)
```



Turing Completeness



repeat().until()	do-while
until().repeat()	while-do
choose()	if-then-else
choose().option()	switch
as()	var set
select()	var get

Every modern programming language supports function chaining and nesting.

g.V(1).out("knows")

g.V(1).out('knows')

g.V(1).out('knows')

Quiz

out('created')

repeat(out()).times(2)

count()

id()

groupCount()

label()

groupCount('m')

where('a',eq('b'))

property('name','marko')

has('age')

barrier()

values('name')

out('created')

flatmap: vertex → vertex*

repeat(out()).times(2)

count()

id()

groupCount()

label()

groupCount('m')

where('a',eq('b'))

property('name','marko')

has('age')

barrier()

values('name')

out('created')

flatmap: vertex → vertex*

repeat(out()).times(2)

flatmap: vertex → vertex*

count()

reducer: object* → long

id()

map: element → object

groupCount()

label()

groupCount('m')

where('a',eq('b'))

property('name','marko')

has('age')

barrier()

values('name')

out('created')

flatmap: vertex → vertex*

repeat(out()).times(2)

flatmap: vertex → vertex*

count()

reducer: object* → long

id()

map: element → object

groupCount()

reducer: object* → map<object,long>

label()

map: element → string

groupCount('m')

side-effect: object →(map<object,long>) object

where('a',eq('b'))

filter: object → object

property('name','marko')

has('age')

barrier()

values('name')

out('created')

flatmap: vertex → vertex*

repeat(out()).times(2)

flatmap: vertex → vertex*

count()

reducer: object* → long

id()

map: element → object

groupCount()

reducer: object* → map<object,long>

label()

map: element → string

groupCount('m')

side-effect: object →(map<object,long>) object

where('a',eq('b'))

filter: object → object

property('name','marko')

side-effect: element →(element) element

has('age')

filter: element → element

barrier()

barrier: object* → object*

values('name')

flatmap: element → string*

The traversal guides the traverser down the long and winding graph.

T

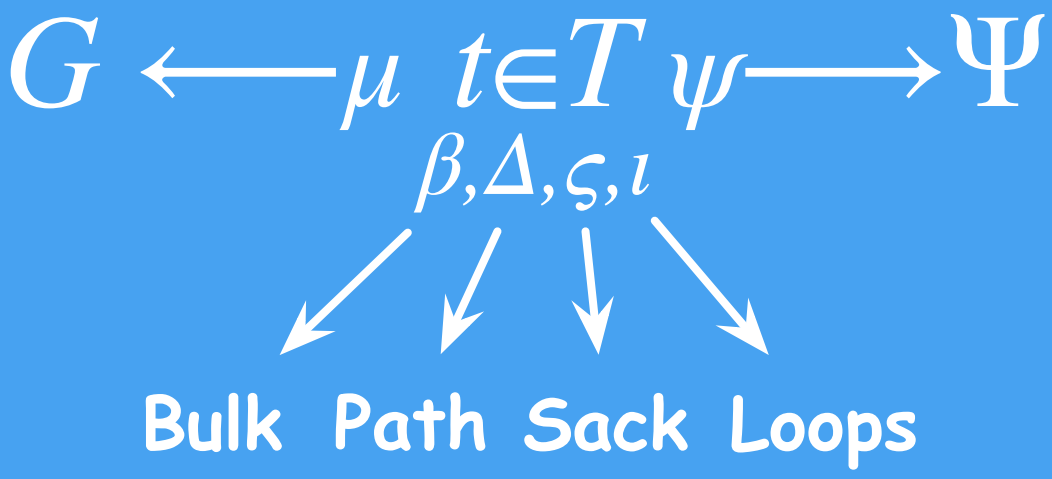
The Traversers

Traversers

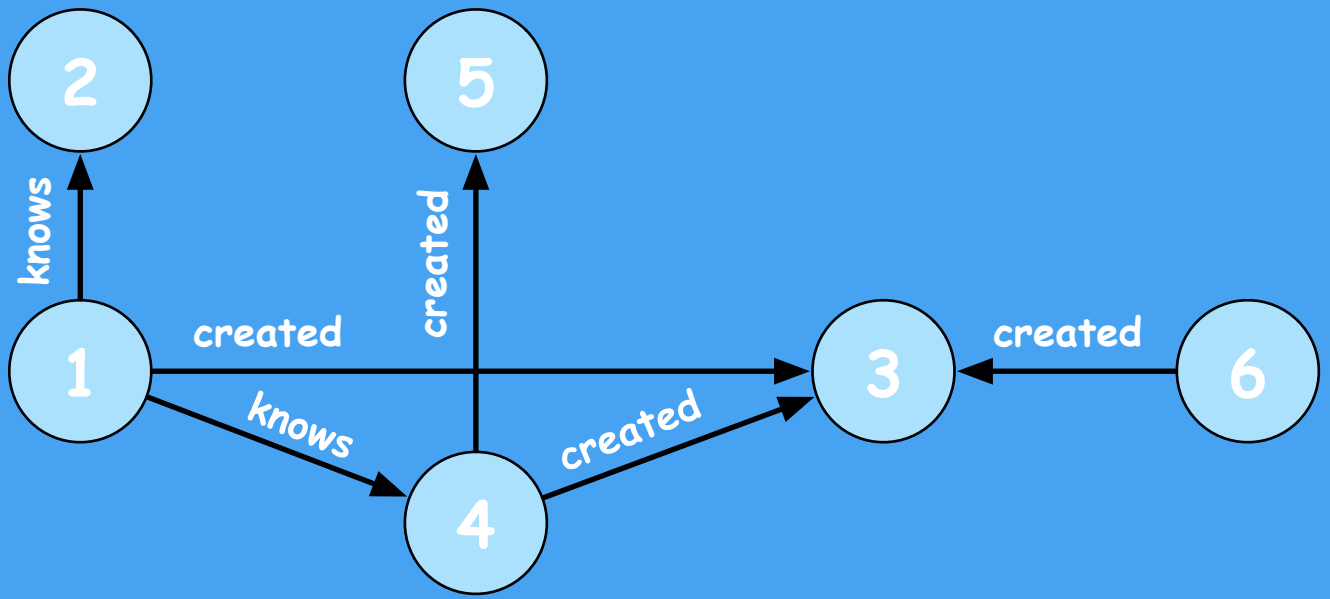
$$G \longleftarrow \mu \;\; t \in T \;\; \psi \longrightarrow \Psi$$

Graph Traversal

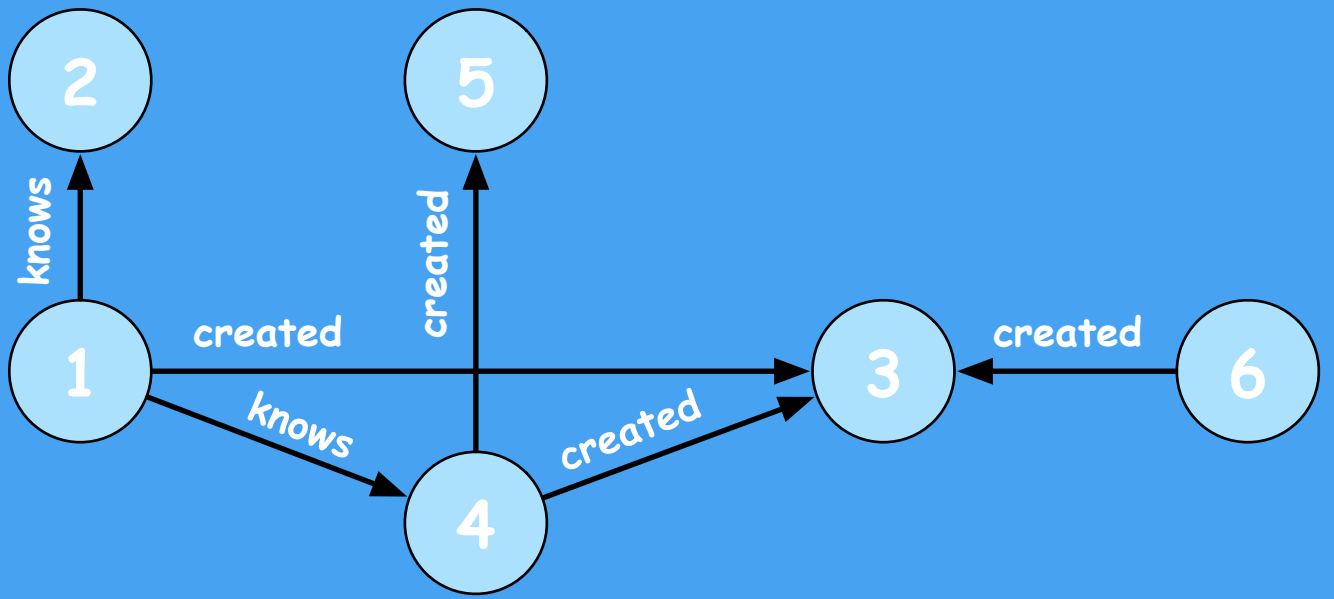
Traversers


$G \longleftarrow \mu \; t \in T \; \psi \longrightarrow \Psi$
$\beta, \Delta, \varsigma, \iota$
Bulk
Path
Sack
Loops

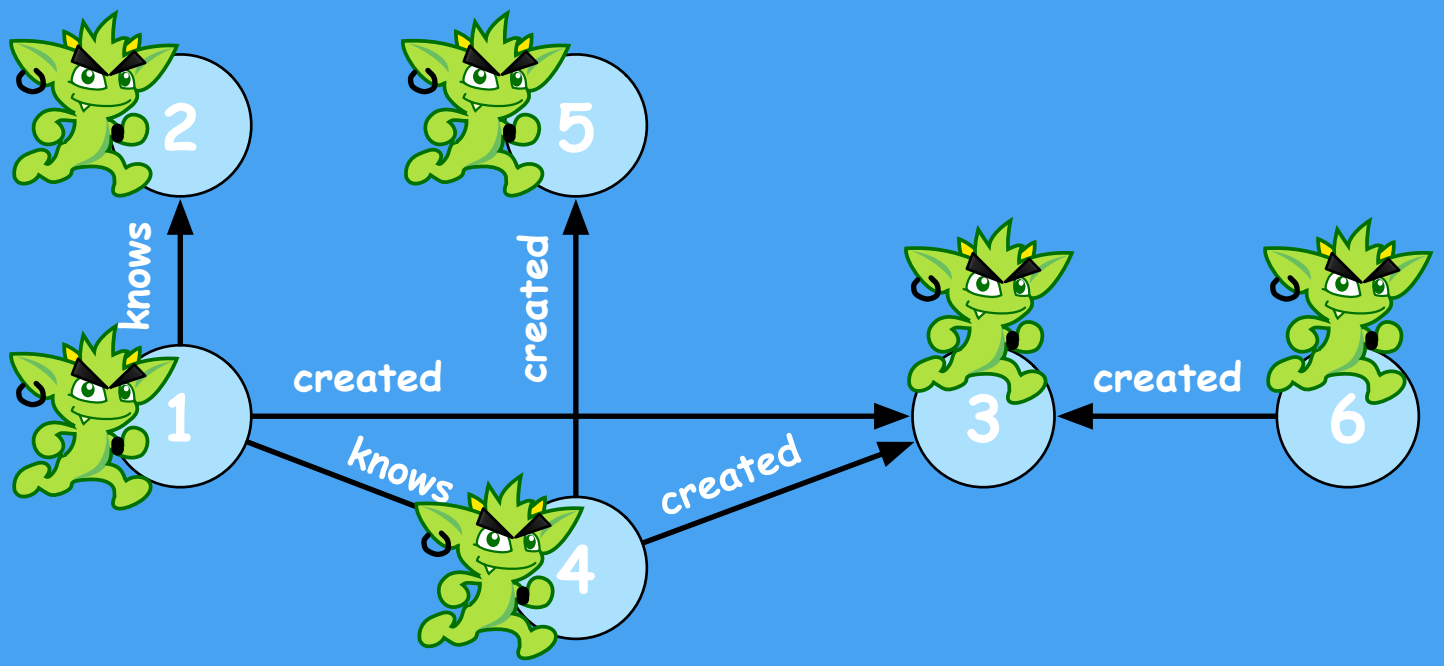
g.V().both().id()



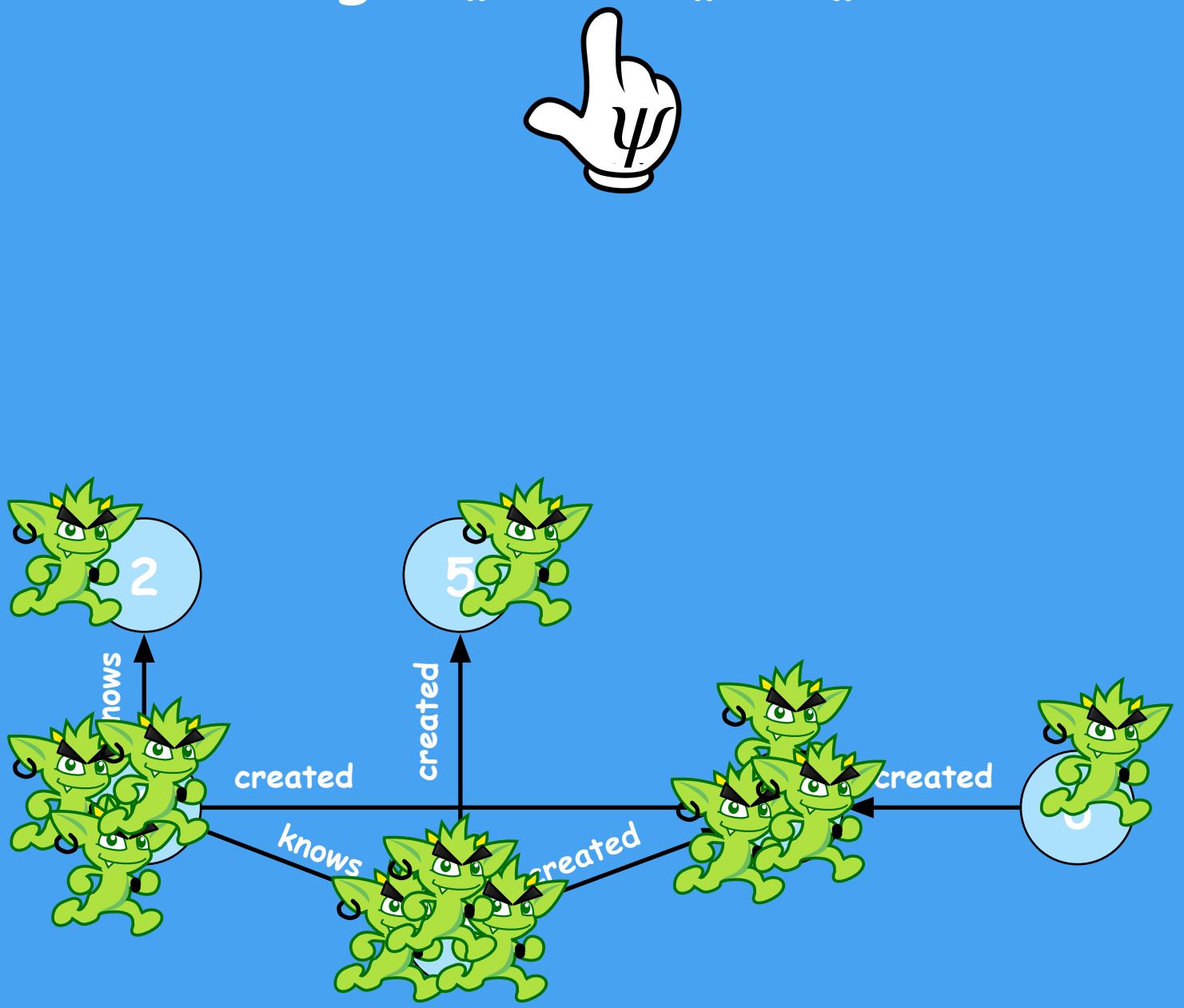
g.V().both().id()



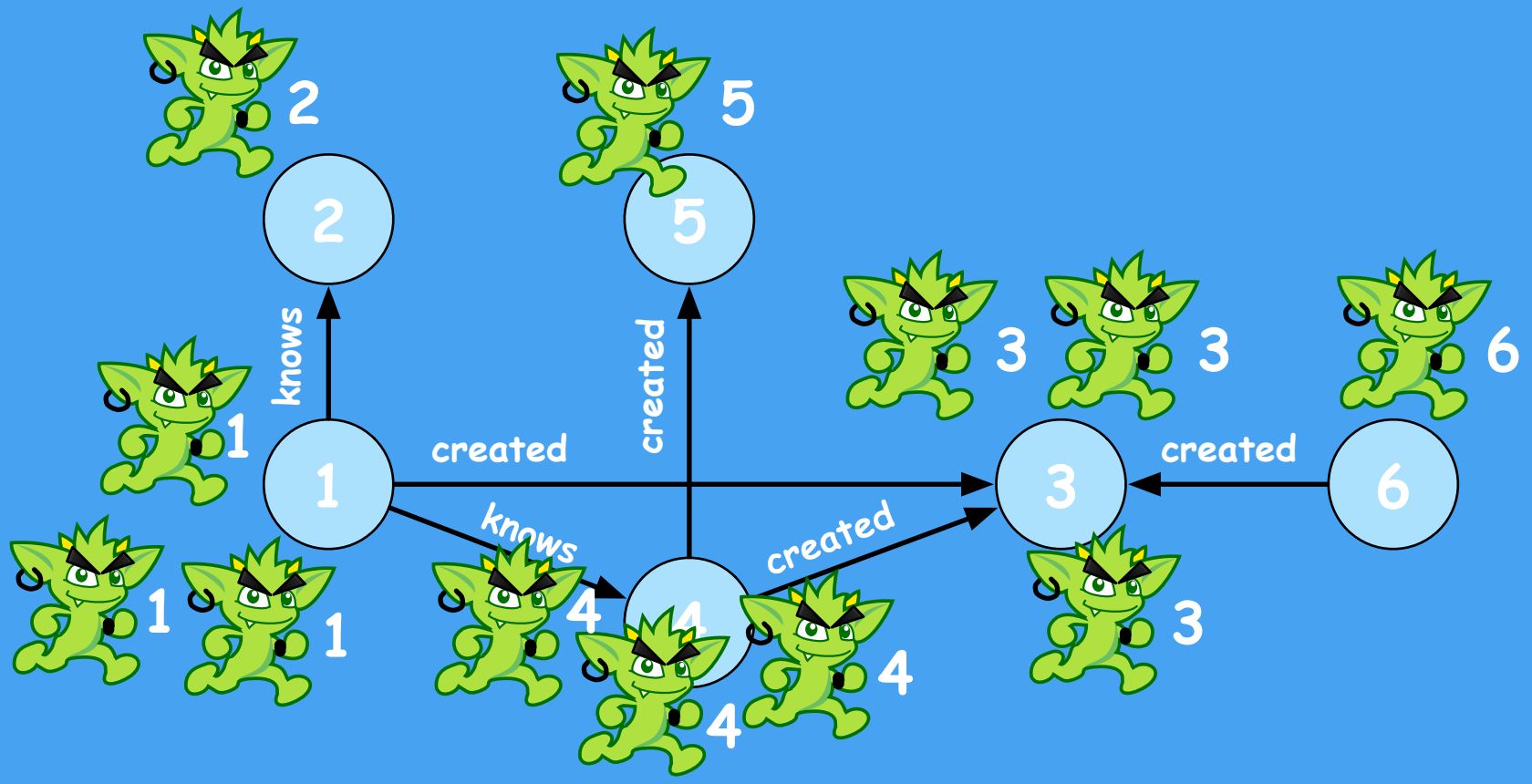
g.V().both().id()



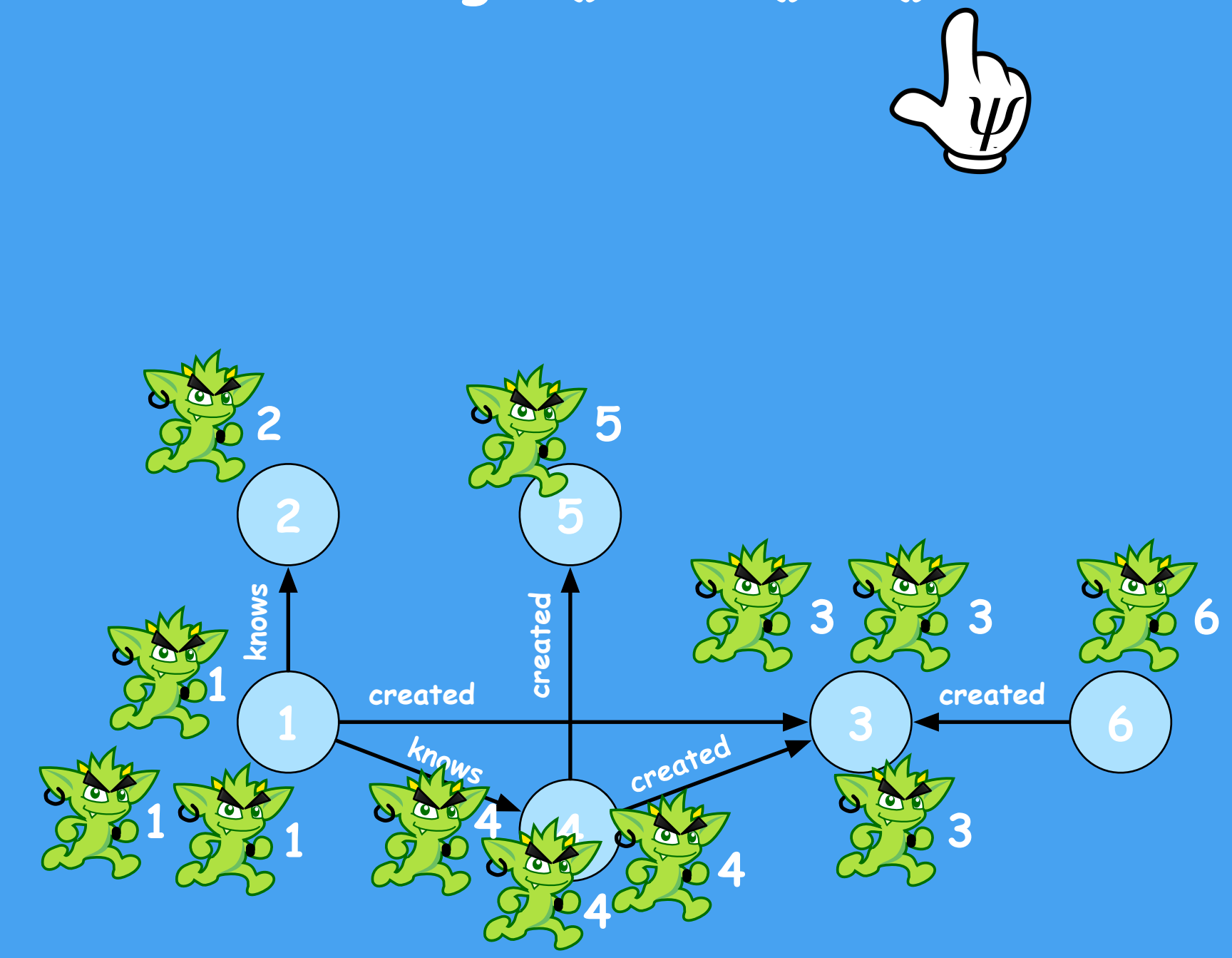
g.V().both().id()

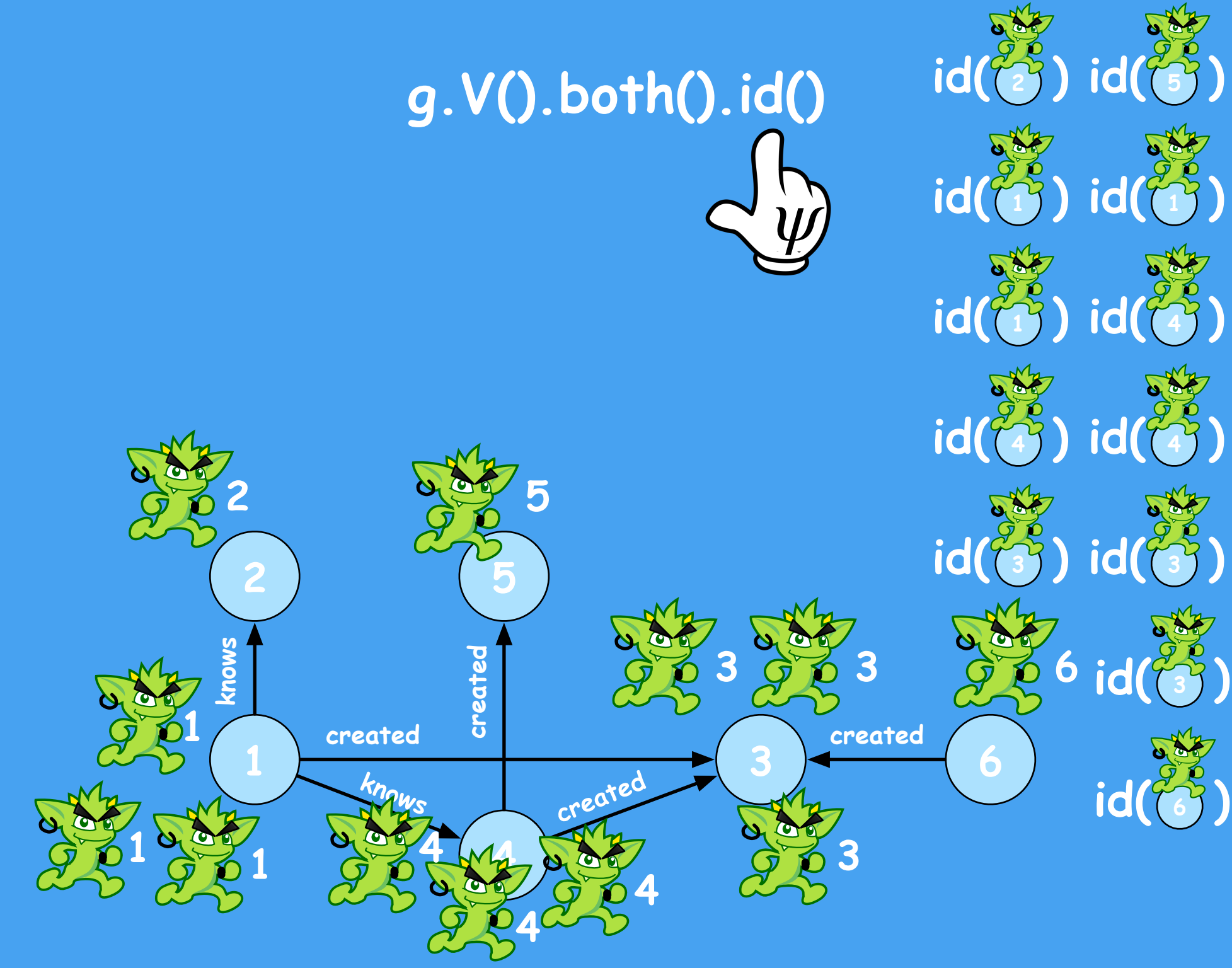


g.V().both().id()

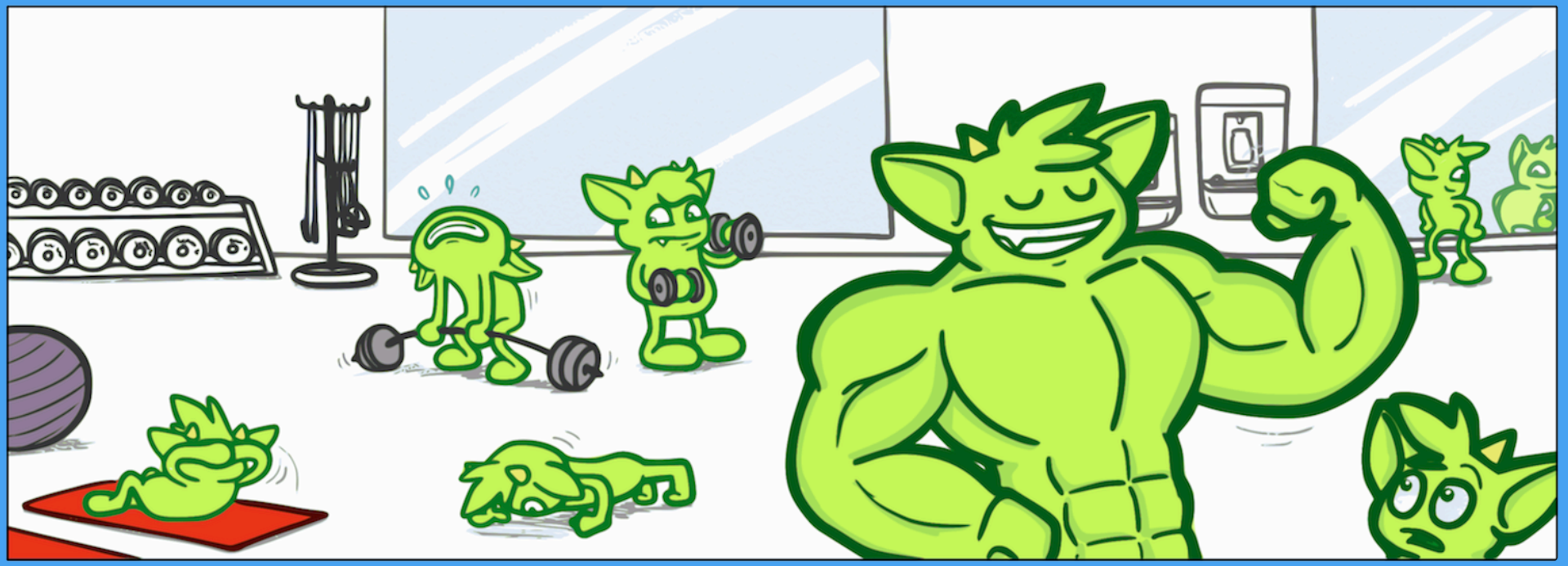

2
5
knows
created
1
created
3
created
6
knows
created
1
1
1
4
4
4
3
3
4

g.V().both().id()


2
5
knows
created
1
created
3
created
6
knows
created
1
1
1
4
4
4
3
3
3
6
==>2
==>5
==>1
==>1
==>1
==>4
==>4
==>4
==>3
==>3
==>3
==>6

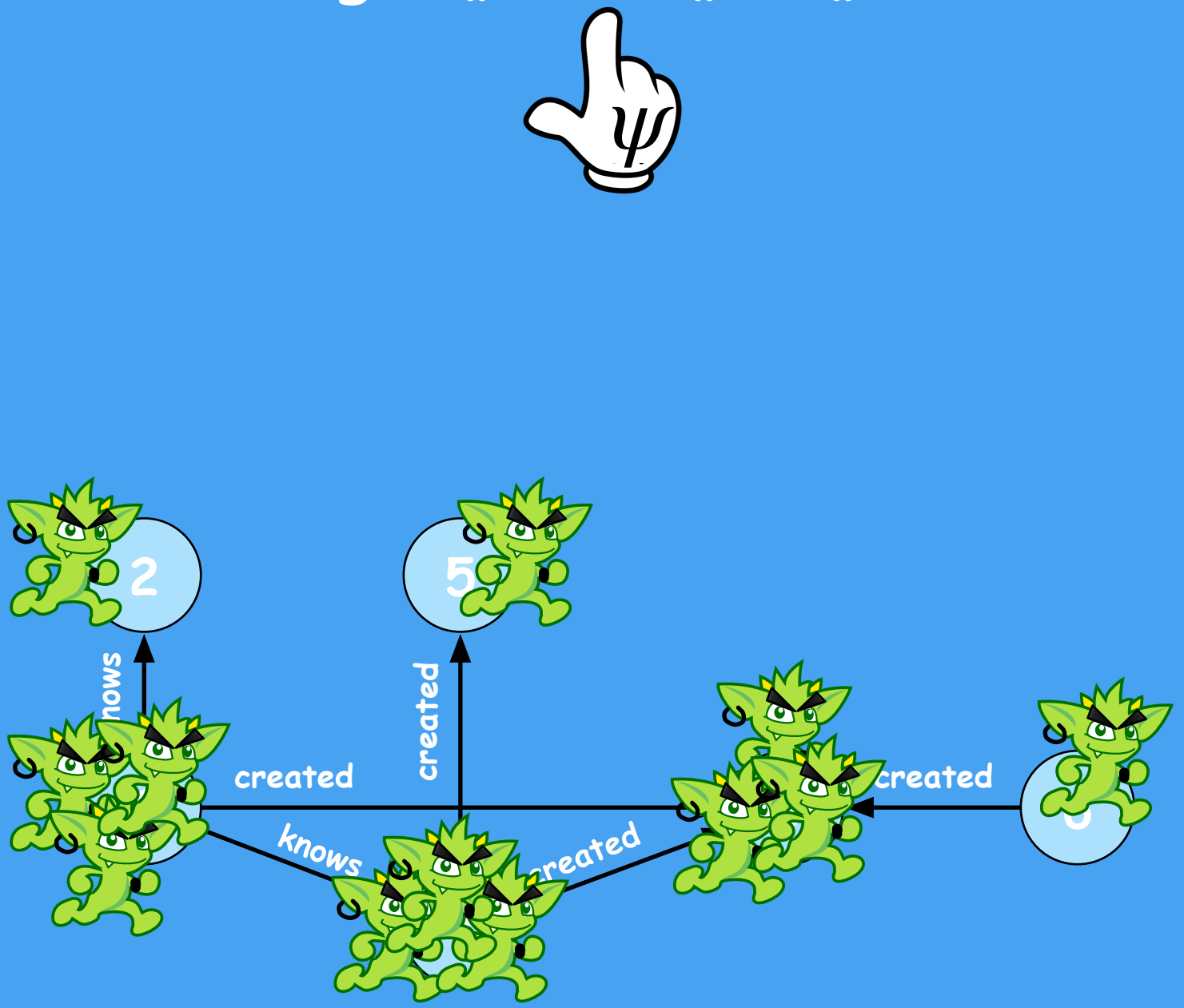

g.V().both().id()
id(2) id(5)
id(1) id(1)
id(1) id(4)
id(4) id(4)
id(3) id(3)
id(3)
id(6)
knows
created
created
knows
created
created

Its time to bulk up.

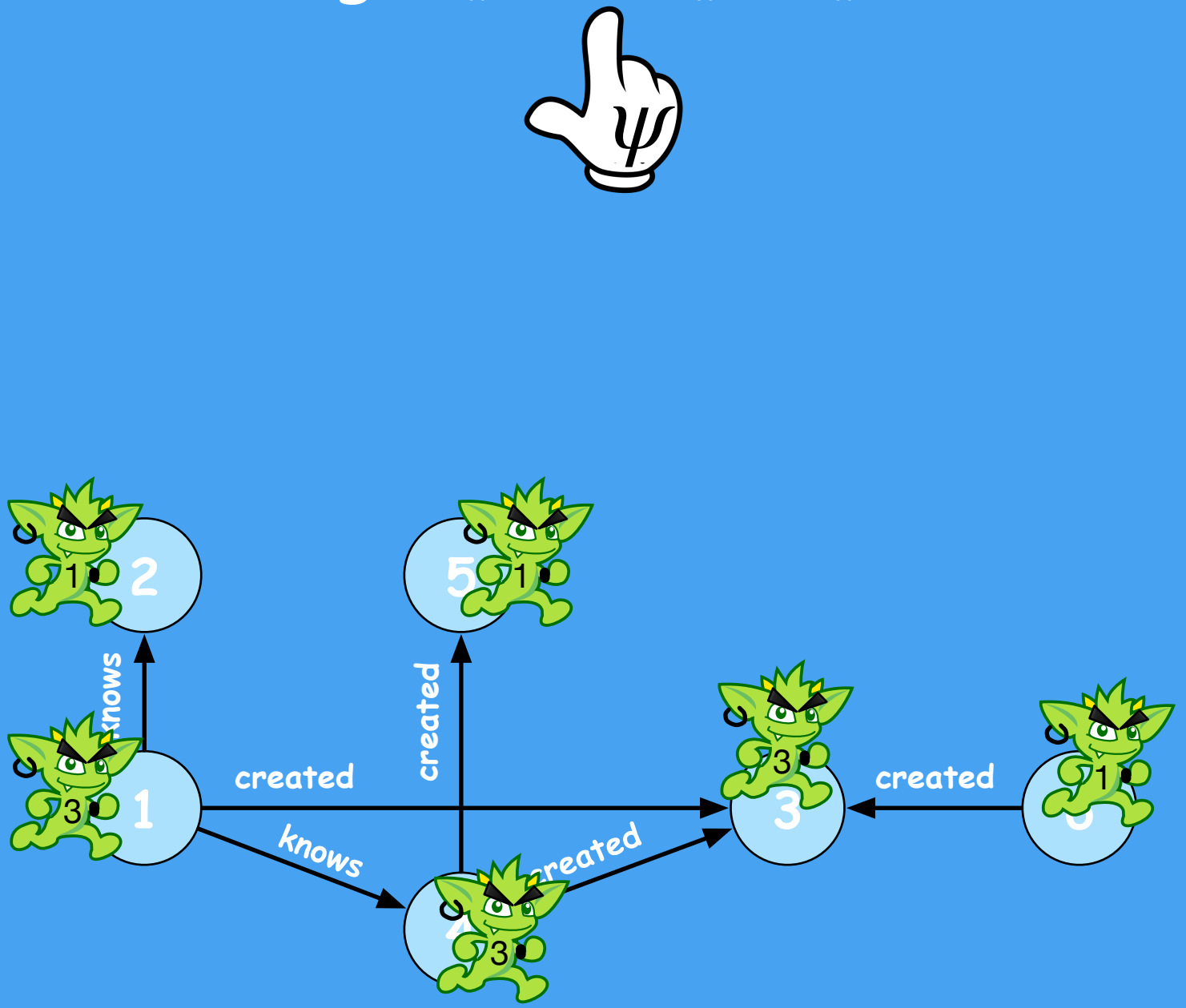


Rodriguez, M.A., "The Beauty of Bulking," Gremlin-Users Mailing List, June 2015.
https://twitter.com/twarko/status/606513003090092035

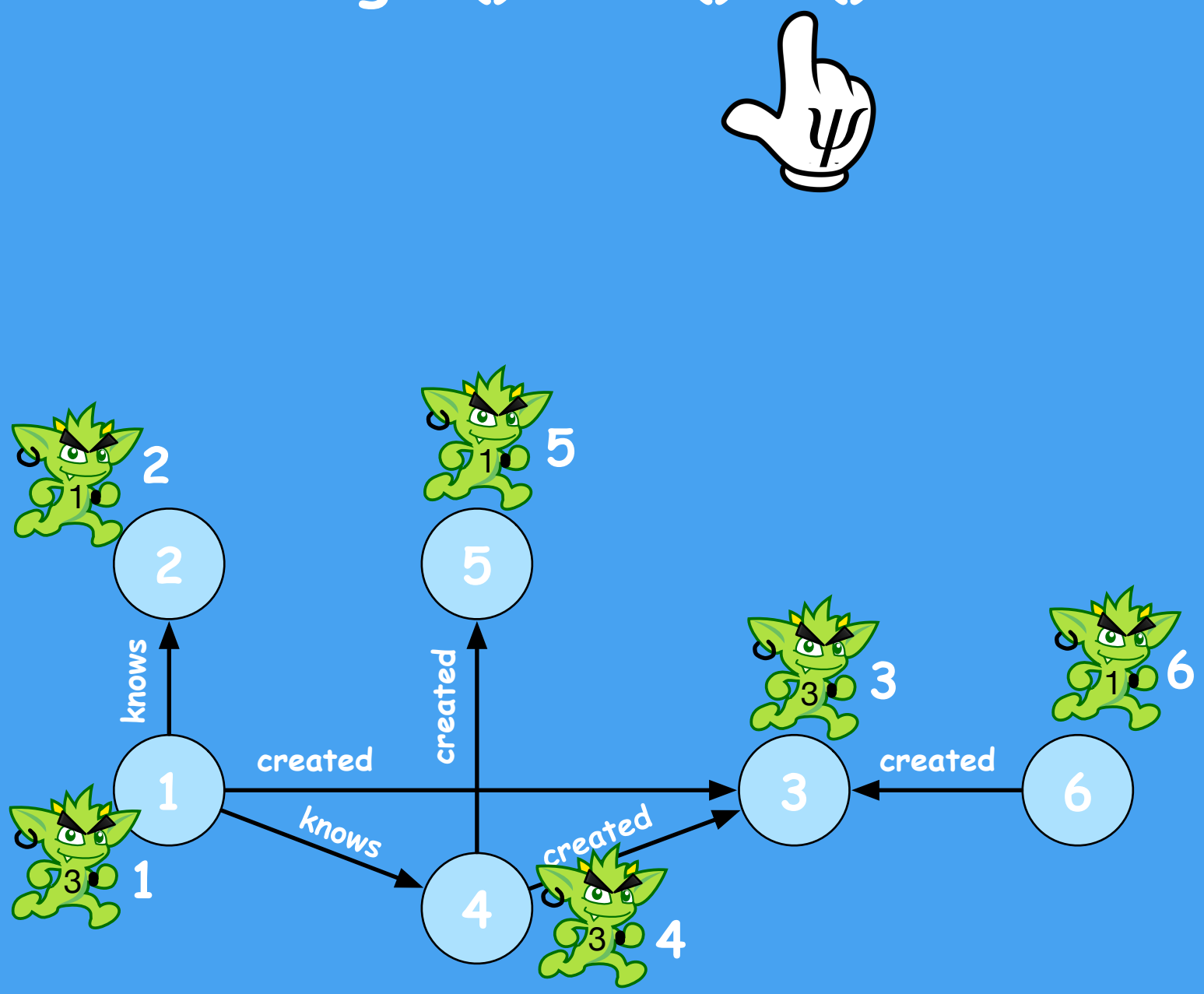
g.V().both().id()



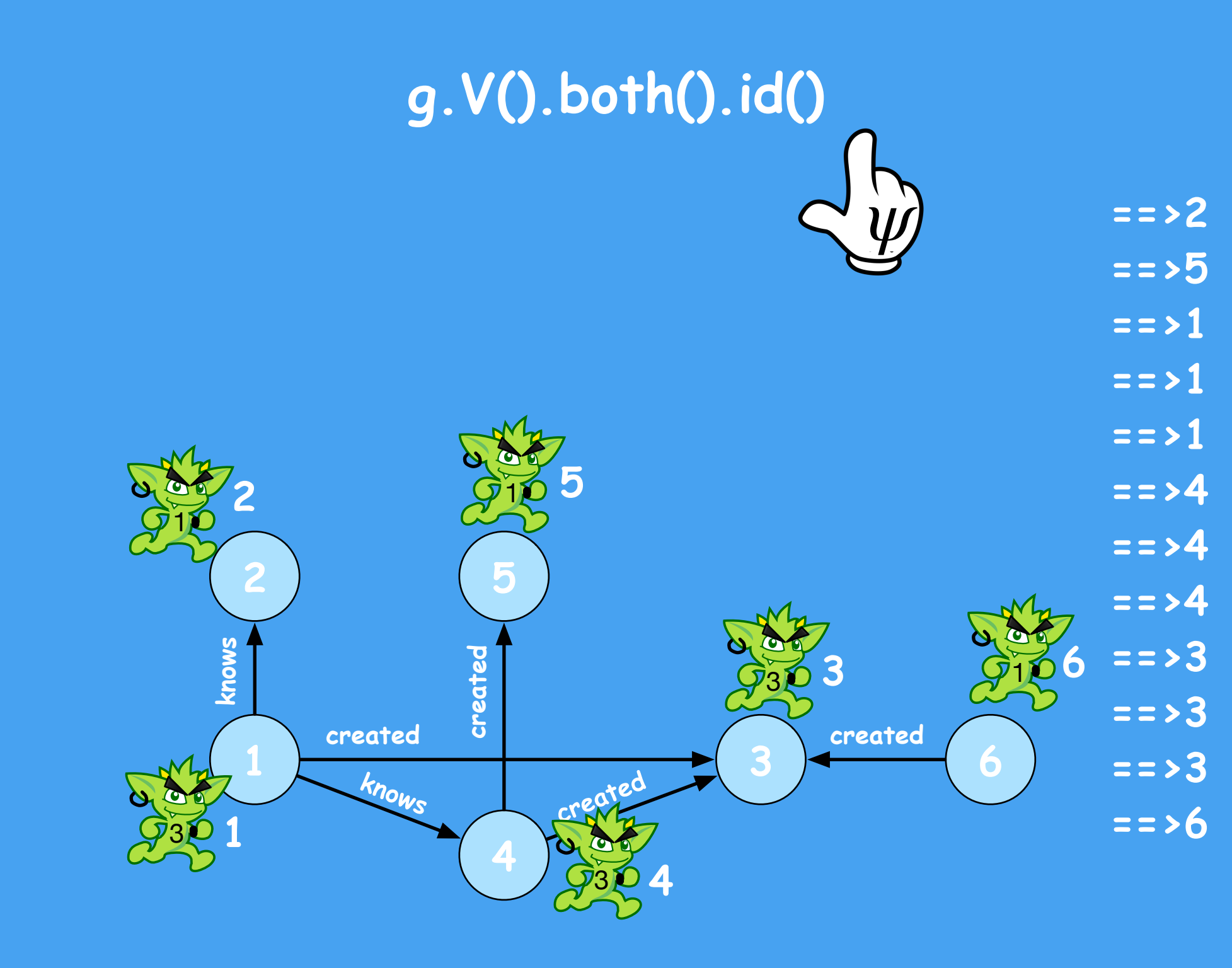
g.V().both().id()



g.V().both().id()

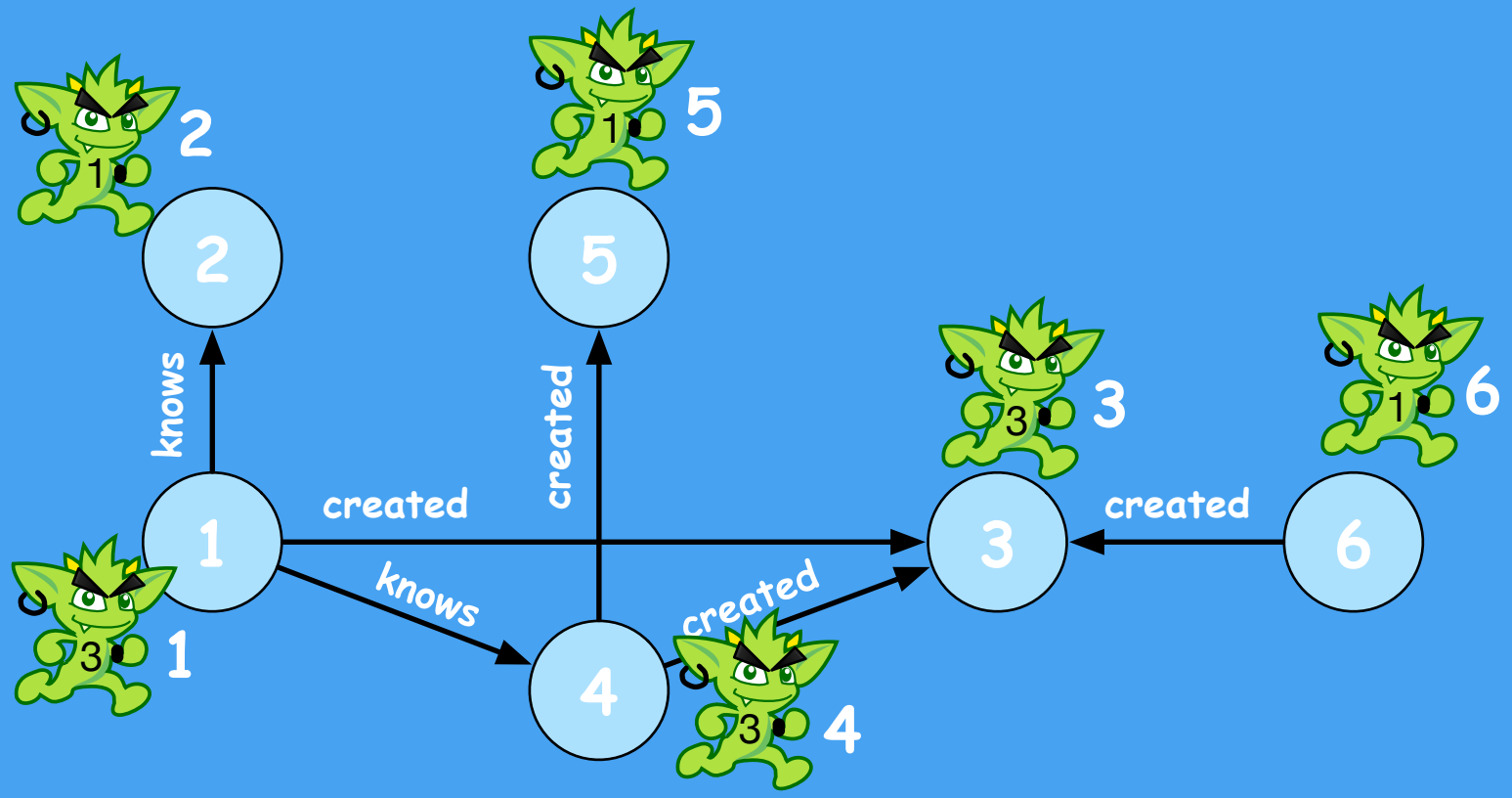

2
1
5
1
knows
created
created
3
3
1
6
created
1
knows
created
3
1
3
4
4

g.V().both().id()


==>2
==>5
==>1
==>1
==>1
==>4
==>4
==>4
==>3
==>3
==>3
==>6
knows
created
created
knows
created
created
1
2
3
4
5
6

g.V().both().id()

1/2 as many function calls with bulking!



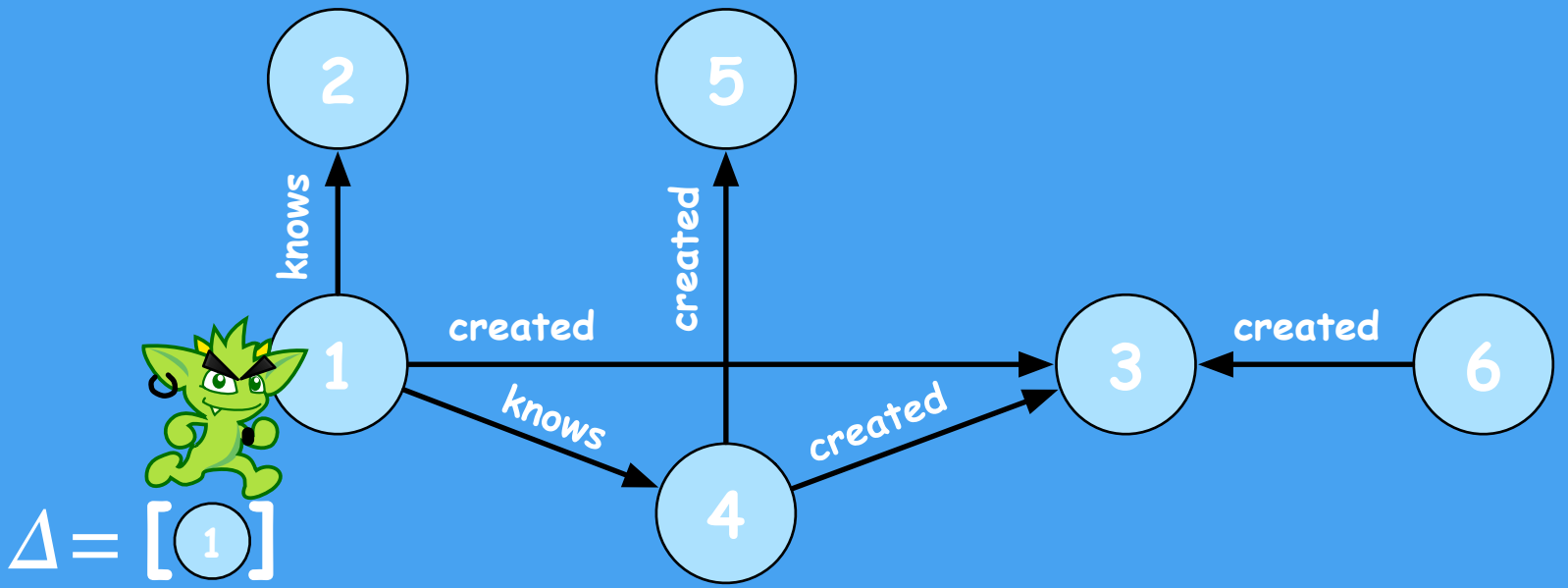
g.V().both().id()

optimizes to

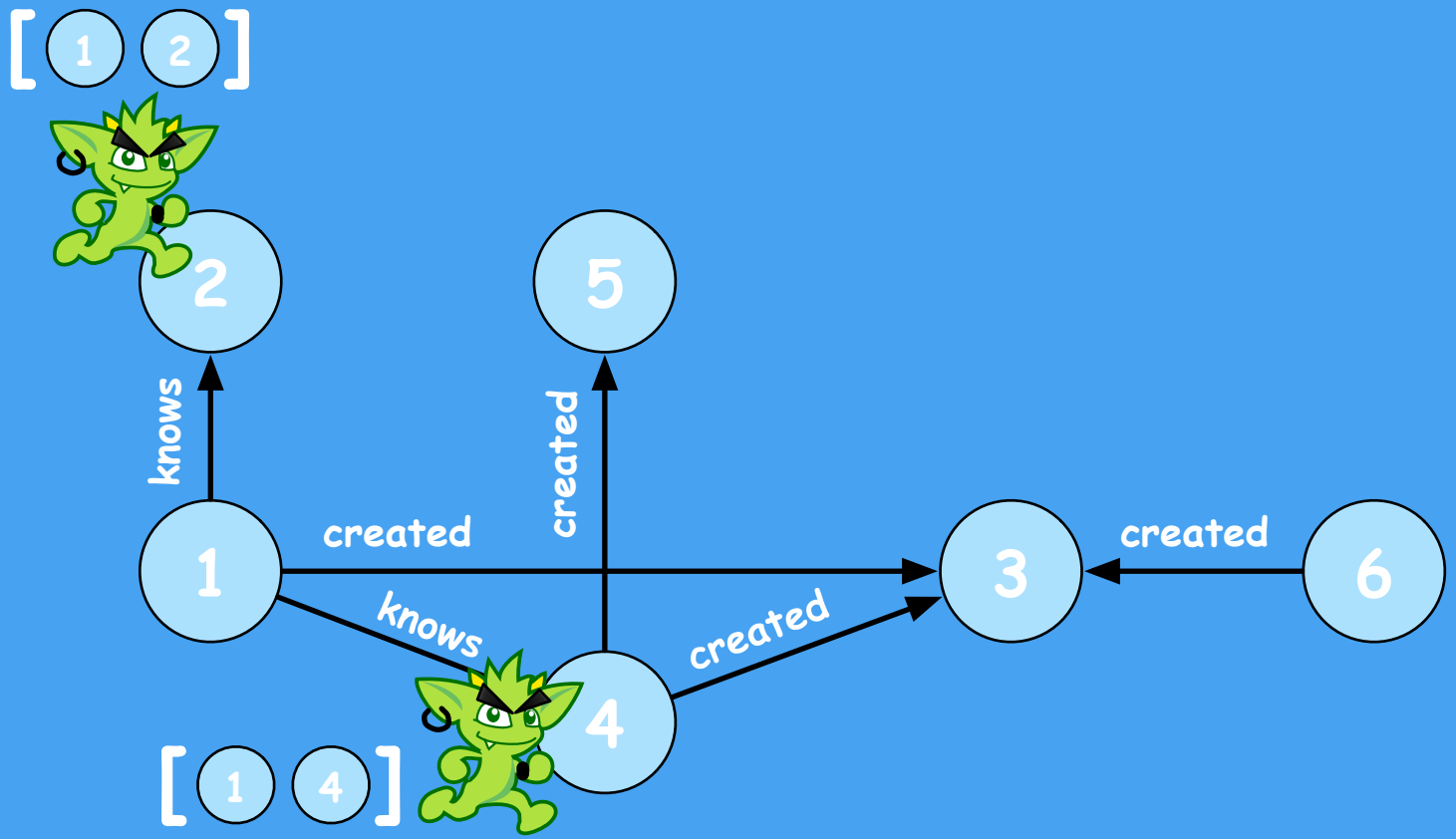
g.V().both().barrier().id()

```
gremlin> g.V().both().id().explain()
==>Traversal Explanation
==========================================================================================================================
Original Traversal                 [GraphStep(vertex,[]), VertexStep(BOTH,vertex), IdStep]

ConnectiveStrategy              [D]   [GraphStep(vertex,[]), VertexStep(BOTH,vertex), IdStep]
MatchPredicateStrategy          [O]   [GraphStep(vertex,[]), VertexStep(BOTH,vertex), IdStep]
FilterRankingStrategy           [O]   [GraphStep(vertex,[]), VertexStep(BOTH,vertex), IdStep]
InlineFilterStrategy            [O]   [GraphStep(vertex,[]), VertexStep(BOTH,vertex), IdStep]
RepeatUnrollStrategy            [O]   [GraphStep(vertex,[]), VertexStep(BOTH,vertex), IdStep]
PathRetractionStrategy          [O]   [GraphStep(vertex,[]), VertexStep(BOTH,vertex), IdStep]
IncidentToAdjacentStrategy      [O]   [GraphStep(vertex,[]), VertexStep(BOTH,vertex), IdStep]
AdjacentToIncidentStrategy      [O]   [GraphStep(vertex,[]), VertexStep(BOTH,vertex), IdStep]
RangeByIsCountStrategy          [O]   [GraphStep(vertex,[]), VertexStep(BOTH,vertex), IdStep]
LazyBarrierStrategy             [O]   [GraphStep(vertex,[]), VertexStep(BOTH,vertex), NoOpBarrierStep(2500), IdStep]
TinkerGraphCountStrategy        [P]   [GraphStep(vertex,[]), VertexStep(BOTH,vertex), NoOpBarrierStep(2500), IdStep]
TinkerGraphStepStrategy         [P]   [TinkerGraphStep(vertex,[]), VertexStep(BOTH,vertex), NoOpBarrierStep(2500), IdStep]
ProfileStrategy                 [F]   [TinkerGraphStep(vertex,[]), VertexStep(BOTH,vertex), NoOpBarrierStep(2500), IdStep]
StandardVerificationStrategy    [V]   [TinkerGraphStep(vertex,[]), VertexStep(BOTH,vertex), NoOpBarrierStep(2500), IdStep]

Final Traversal                    [TinkerGraphStep(vertex,[]), VertexStep(BOTH,vertex), NoOpBarrierStep(2500), IdStep]
```

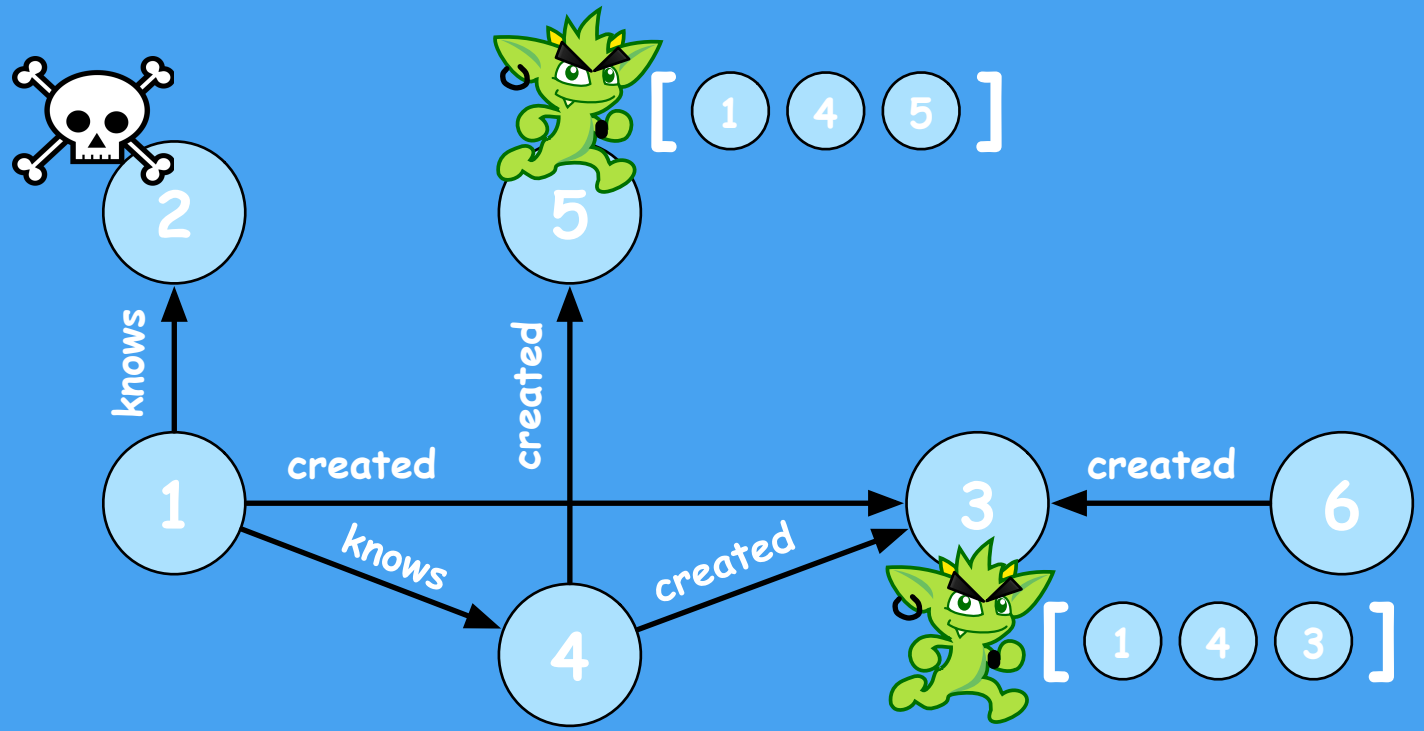
g.V(1).out('knows').out('created').path()



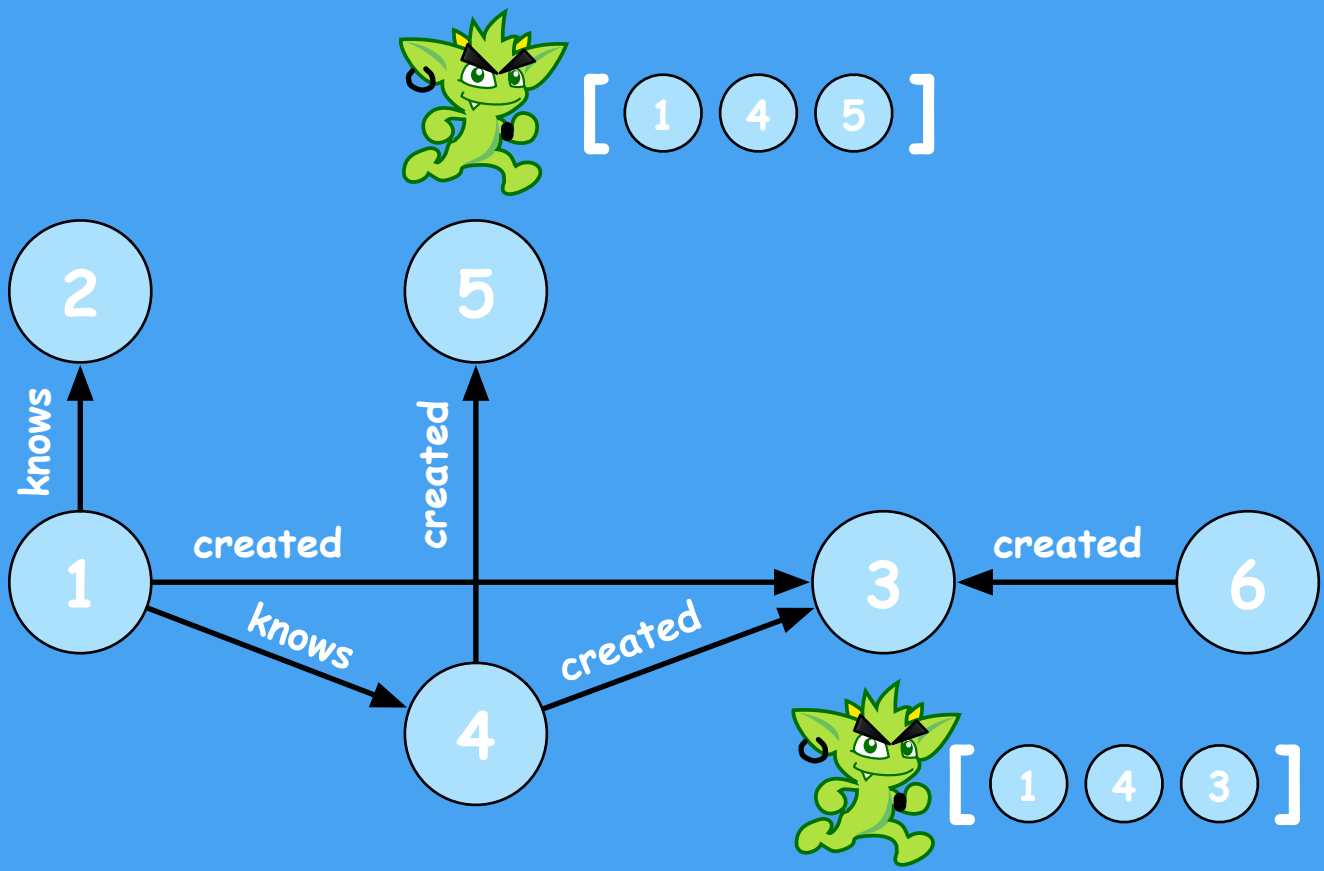
g.V(1).out('knows').out('created').path()



g.V(1).out('knows').out('created').path()

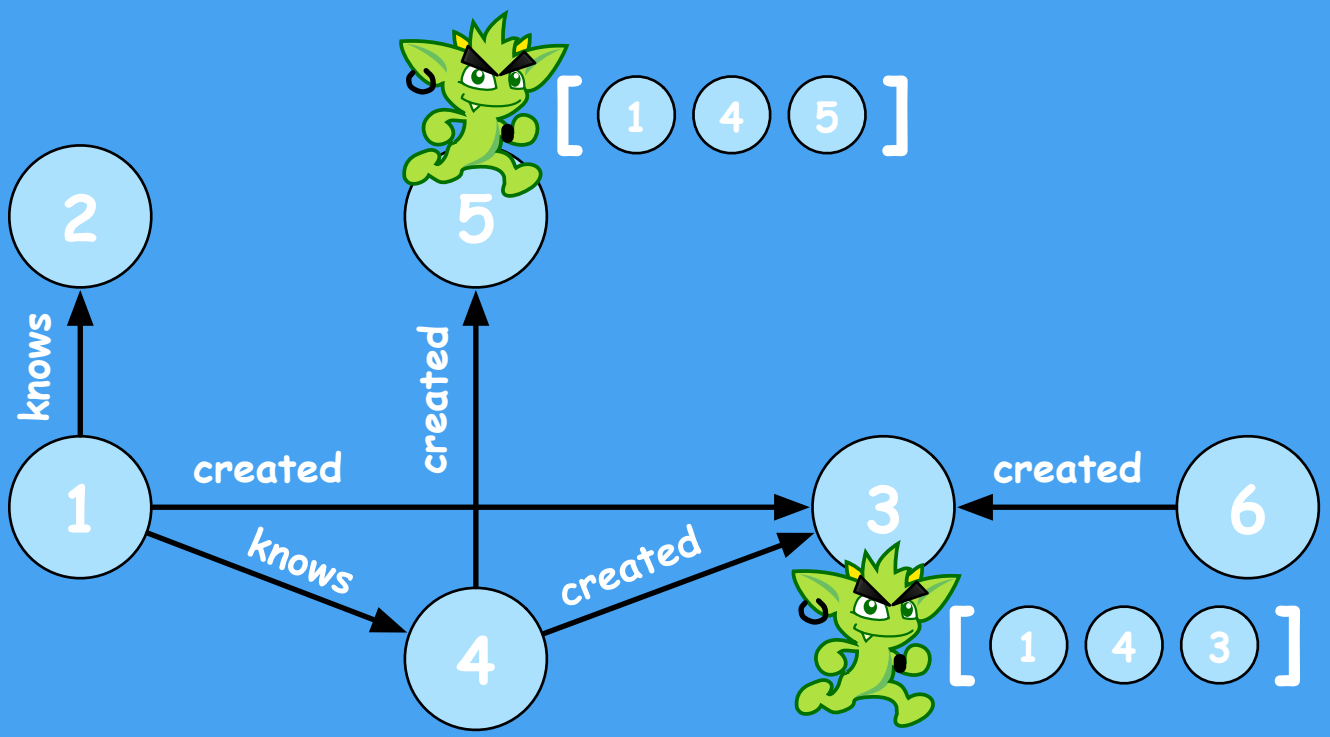


g.V(1).out('knows').out('created').path()

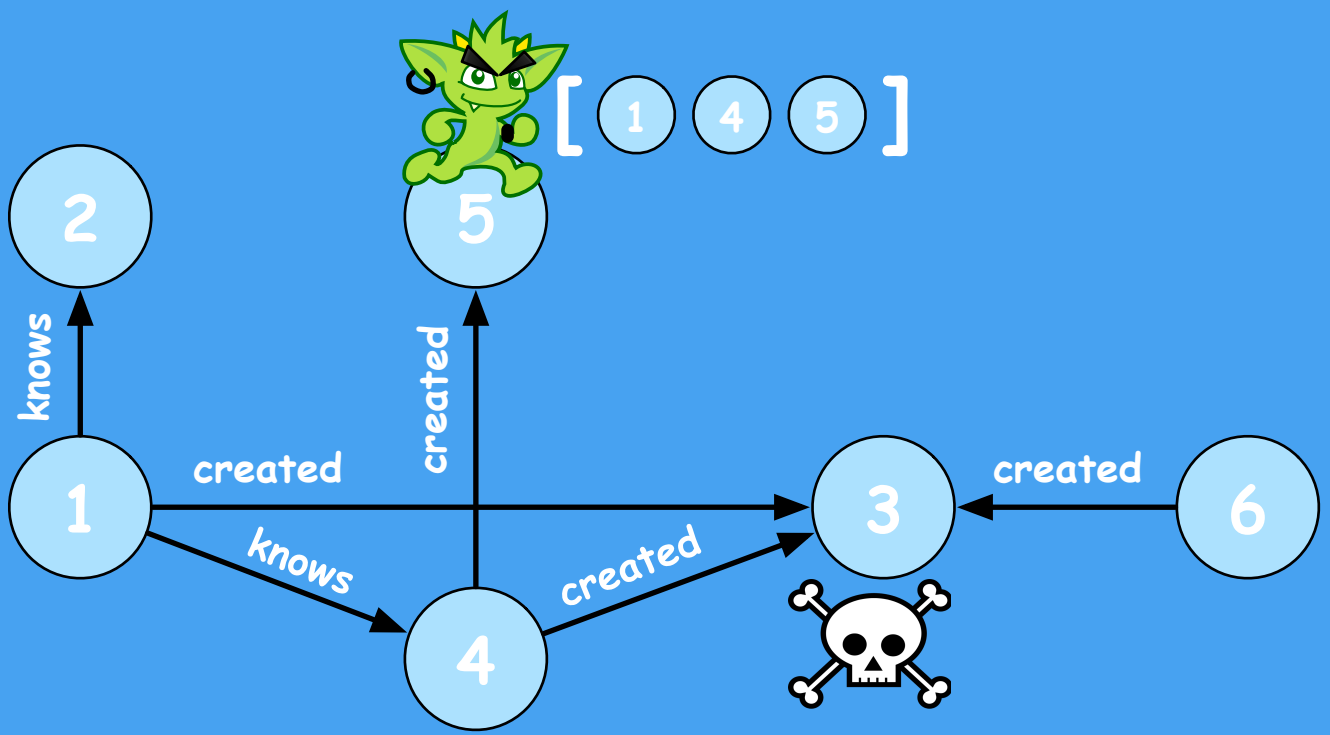


Its time to get crazy with locally scoped traversers.

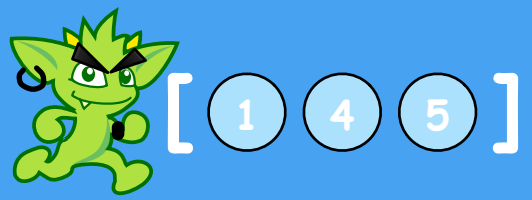
```
g.V(1).out('knows').out('created').limit(1).
path().by(inE().count())
```

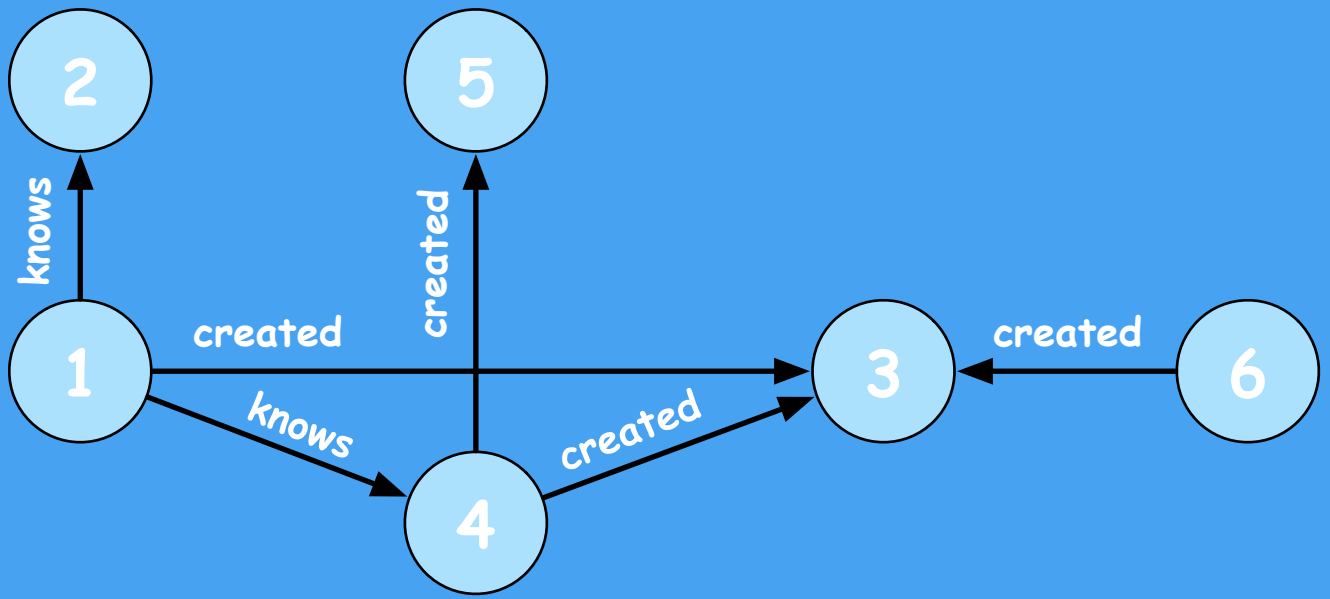


```
g.V(1).out('knows').out('created').limit(1).
  path().by(inE().count())
```

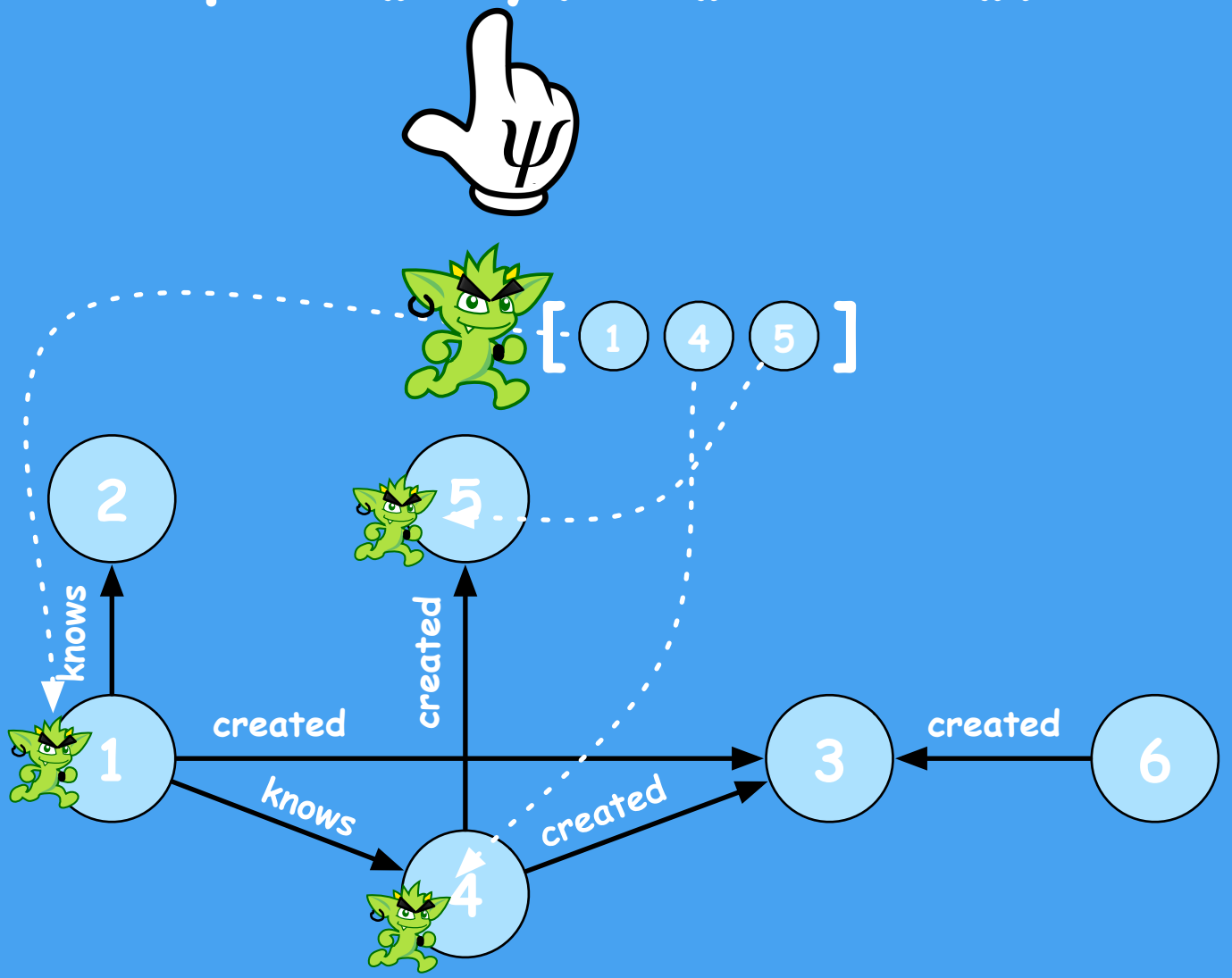


g.V(1).out('knows').out('created').limit(1).
path().by(inE().count())

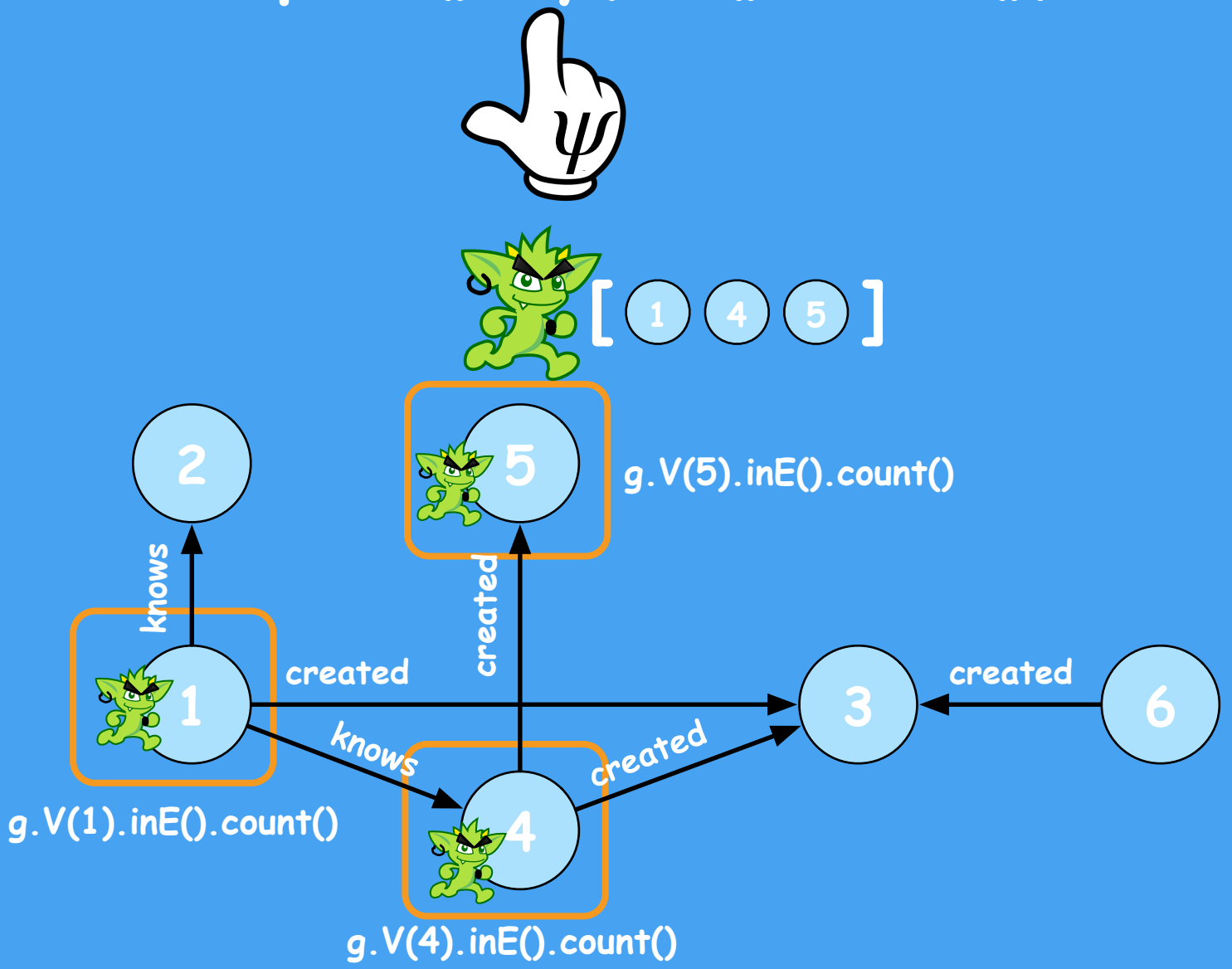




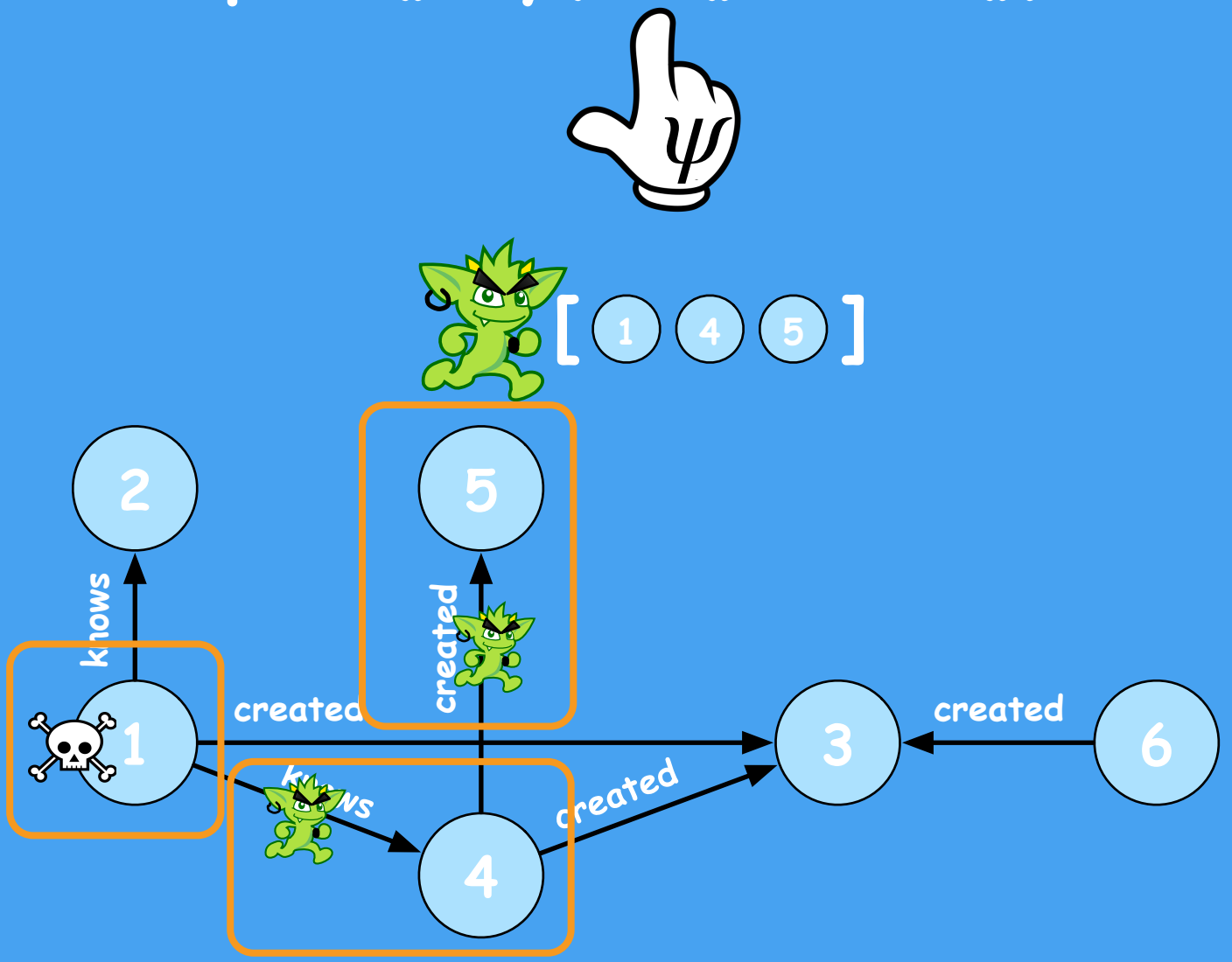
g.V(1).out('knows').out('created').limit(1).
path().by(inE().count())



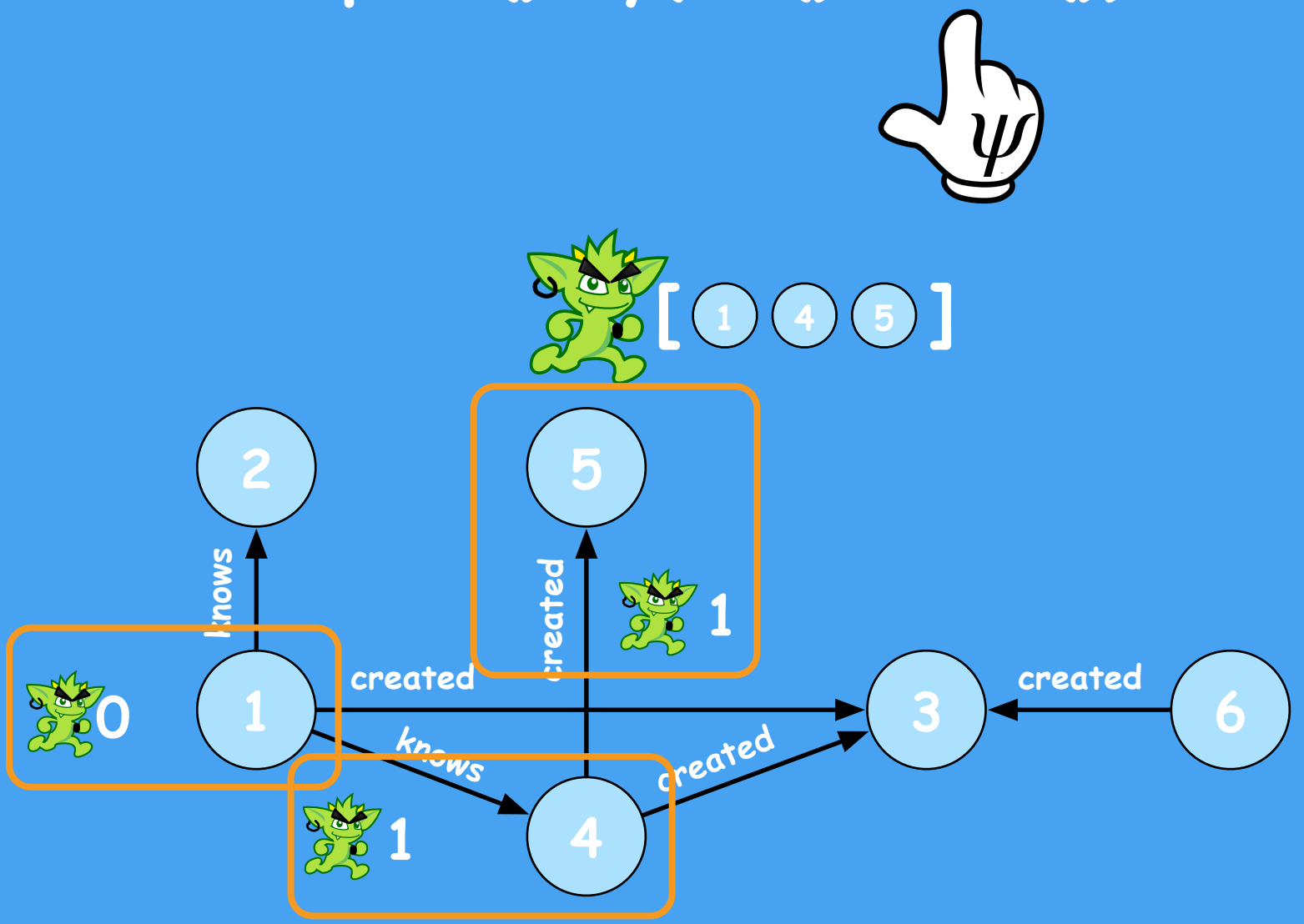
g.V(1).out('knows').out('created').limit(1).path().by(inE().count())



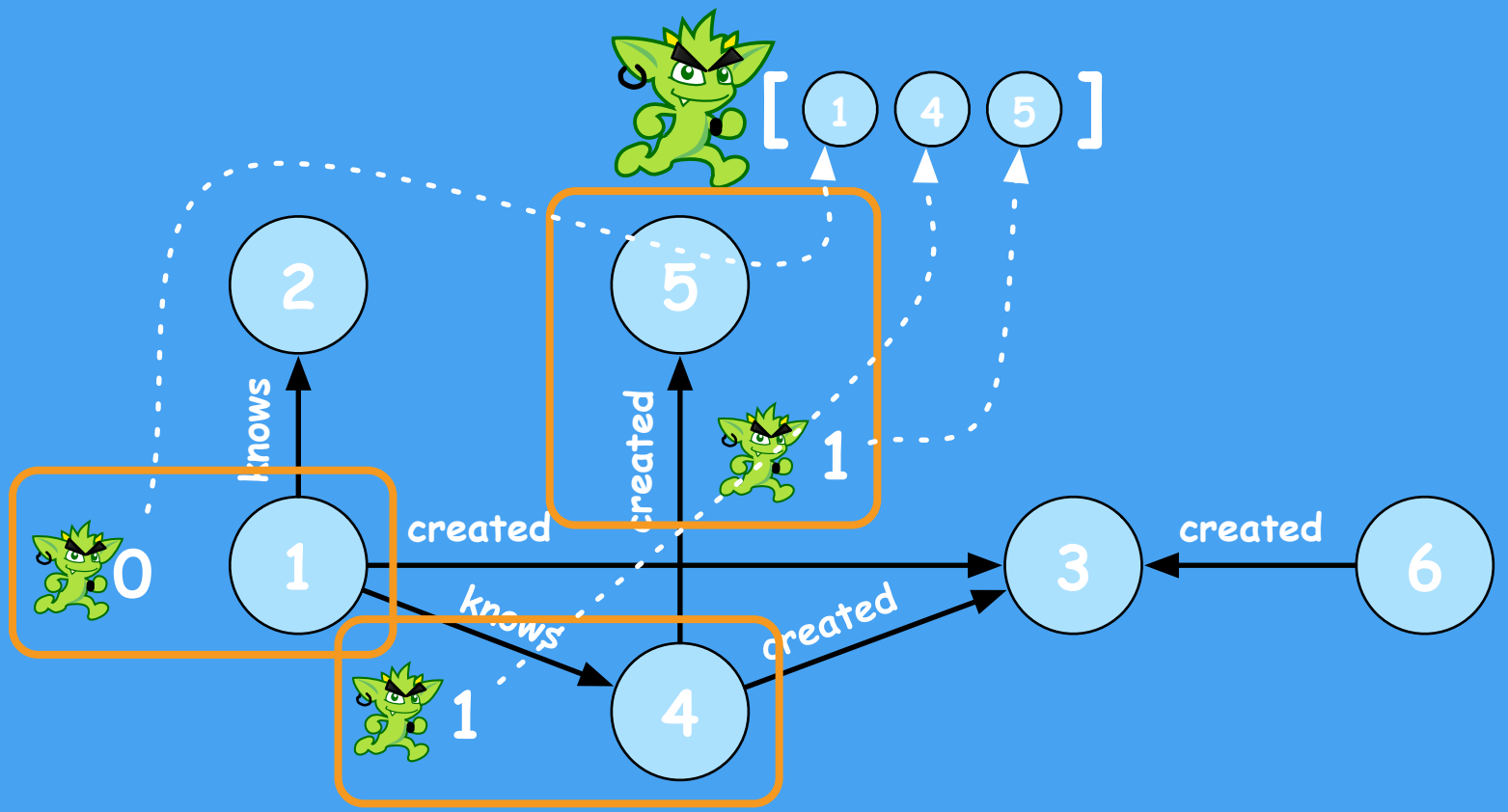
g.V(1).out('knows').out('created').limit(1).
path().by(inE().count())



g.V(1).out('knows').out('created').limit(1).
path().by(inE().count())

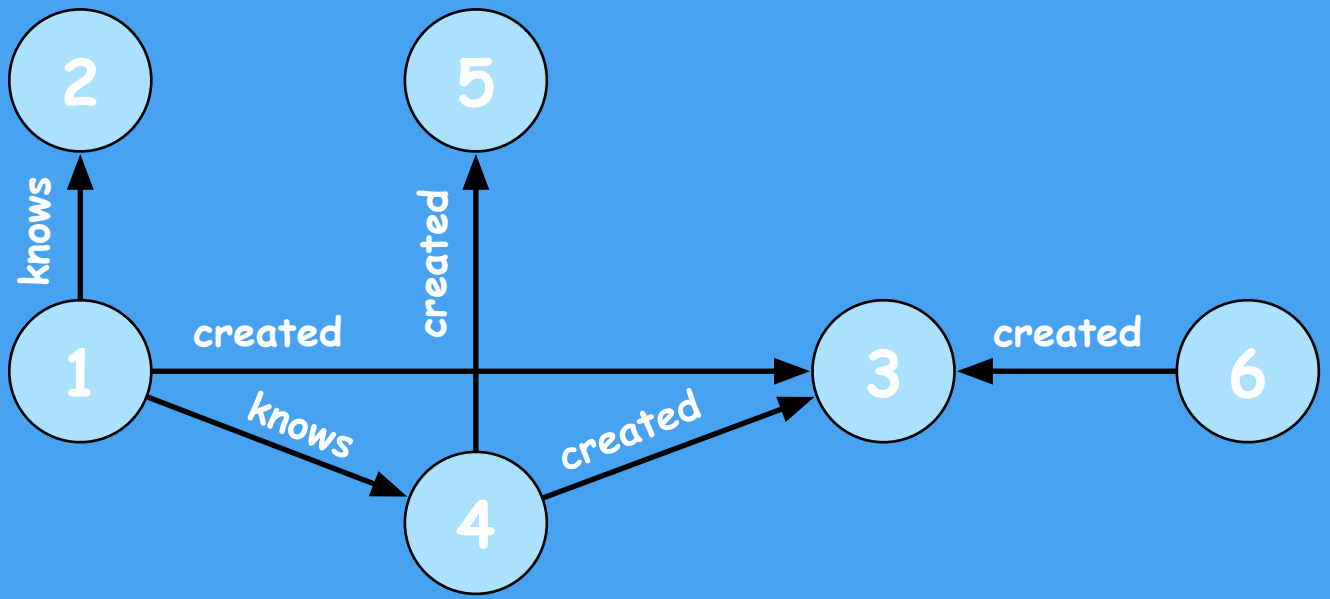


g.V(1).out('knows').out('created').limit(1).
path().by(inE().count())

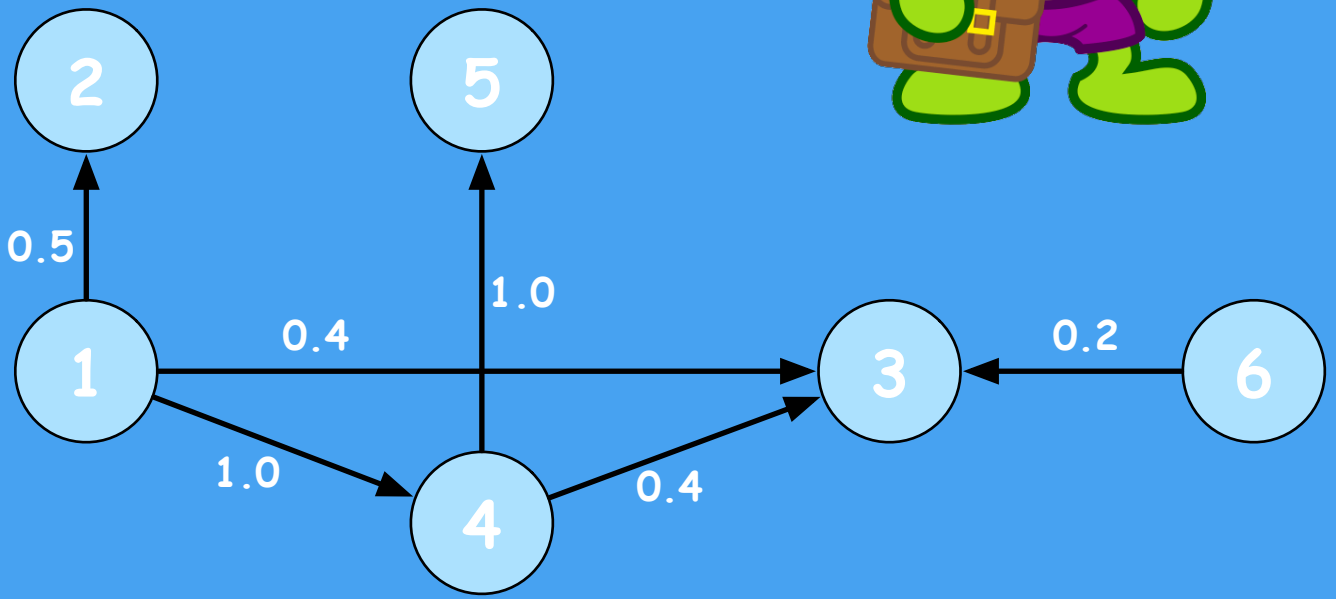


```
g.V(1).out('knows').out('created').limit(1).
  path().by(inE().count())
```

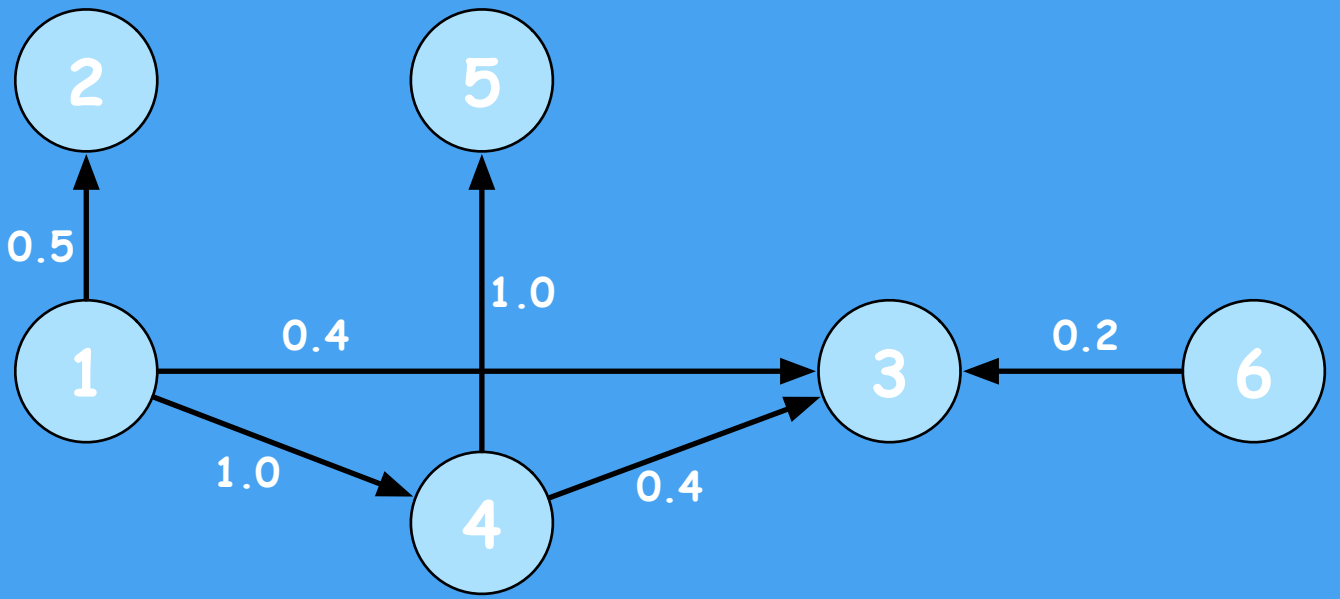
[0 1 1]



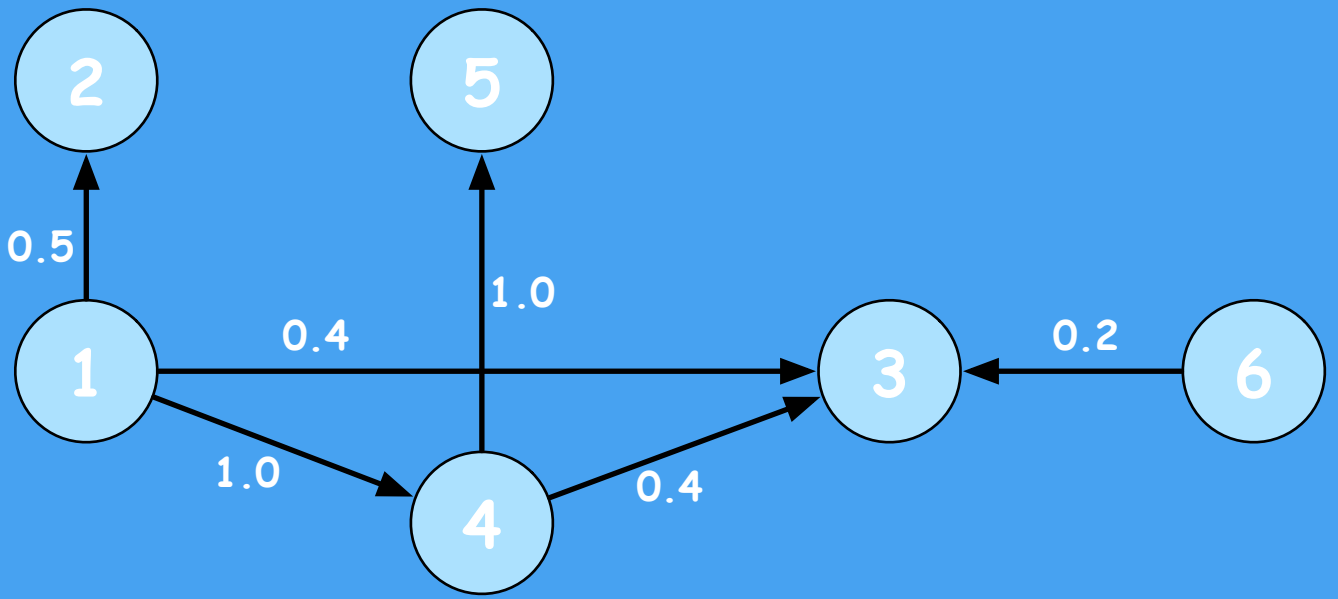
```
g.withSack(1).V(1).
  repeat(outE().sack(mult).by('weight').inV()).
  times(2).sack()
```



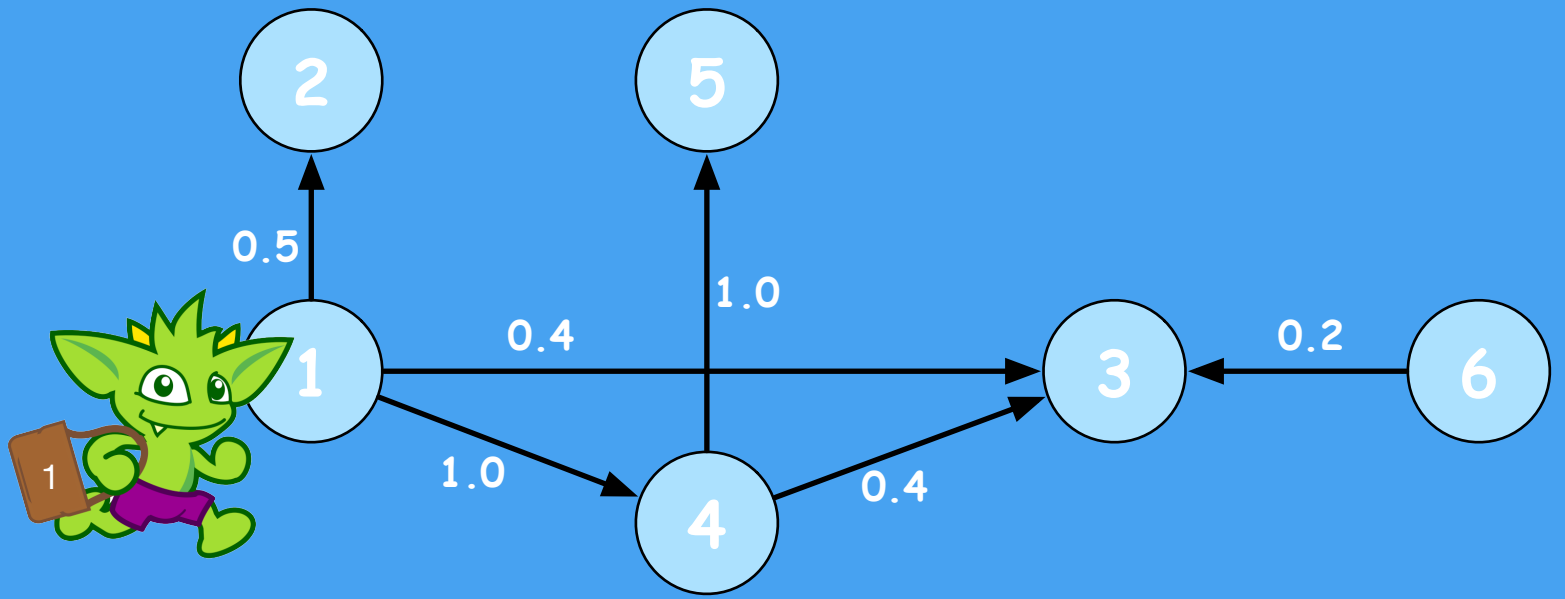
```
g.withSack(1).V(1).
  repeat(outE().sack(mult).by('weight').inV()).
  times(2).sack()
```



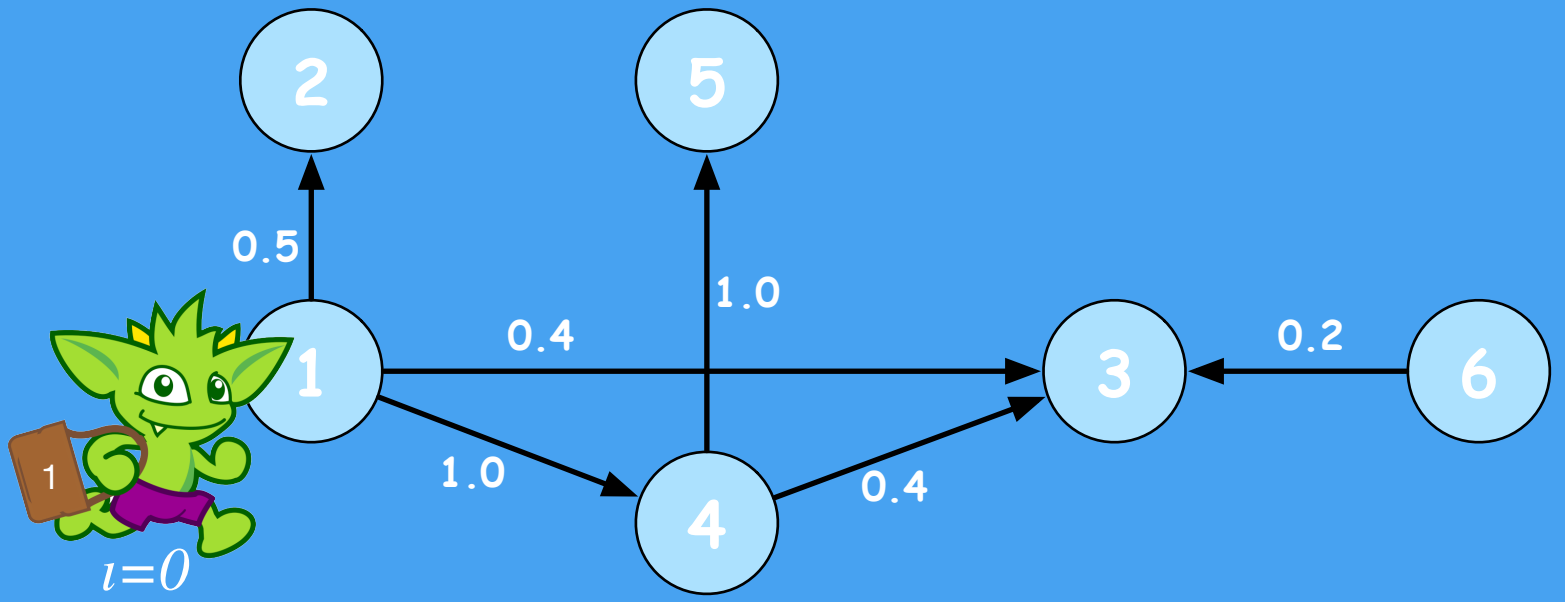
```
g.withSack(1).V(1).
  repeat(outE().sack(mult).by('weight').inV()).
  times(2).sack()
```



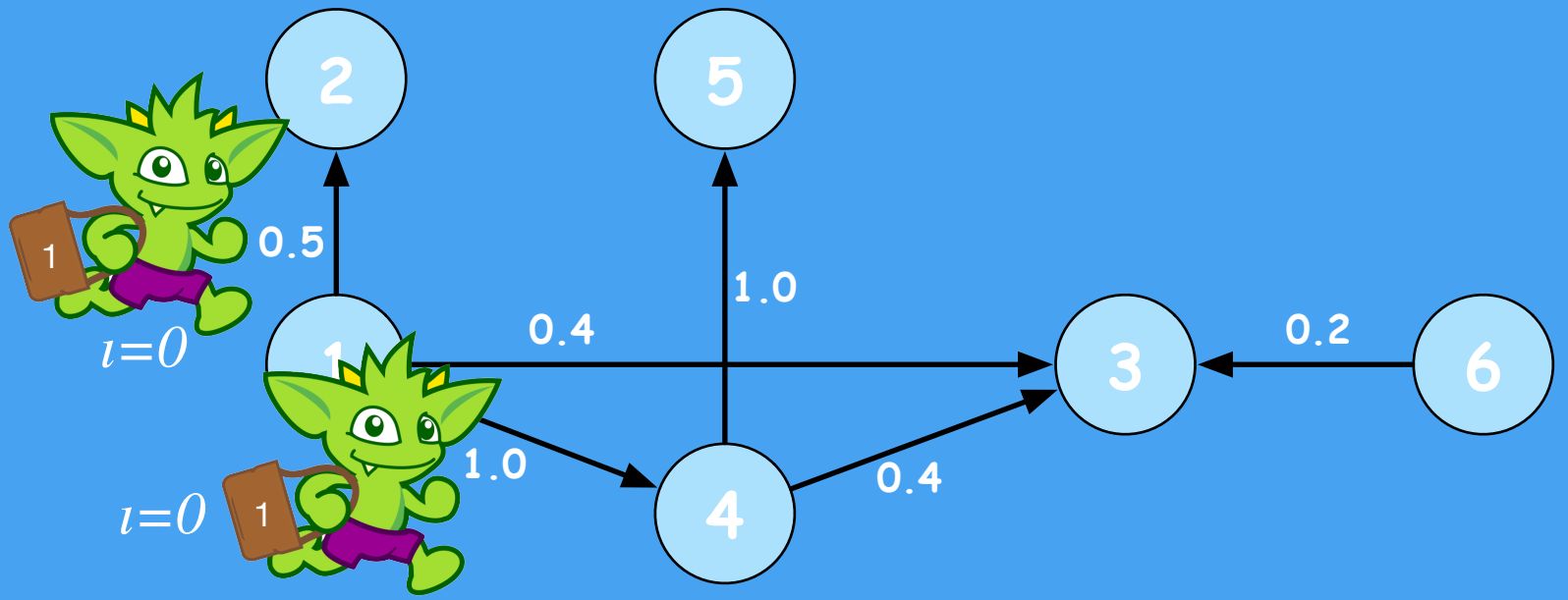
```
g.withSack(1).V(1).
  repeat(outE().sack(mult).by('weight').inV()).
  times(2).sack()
```



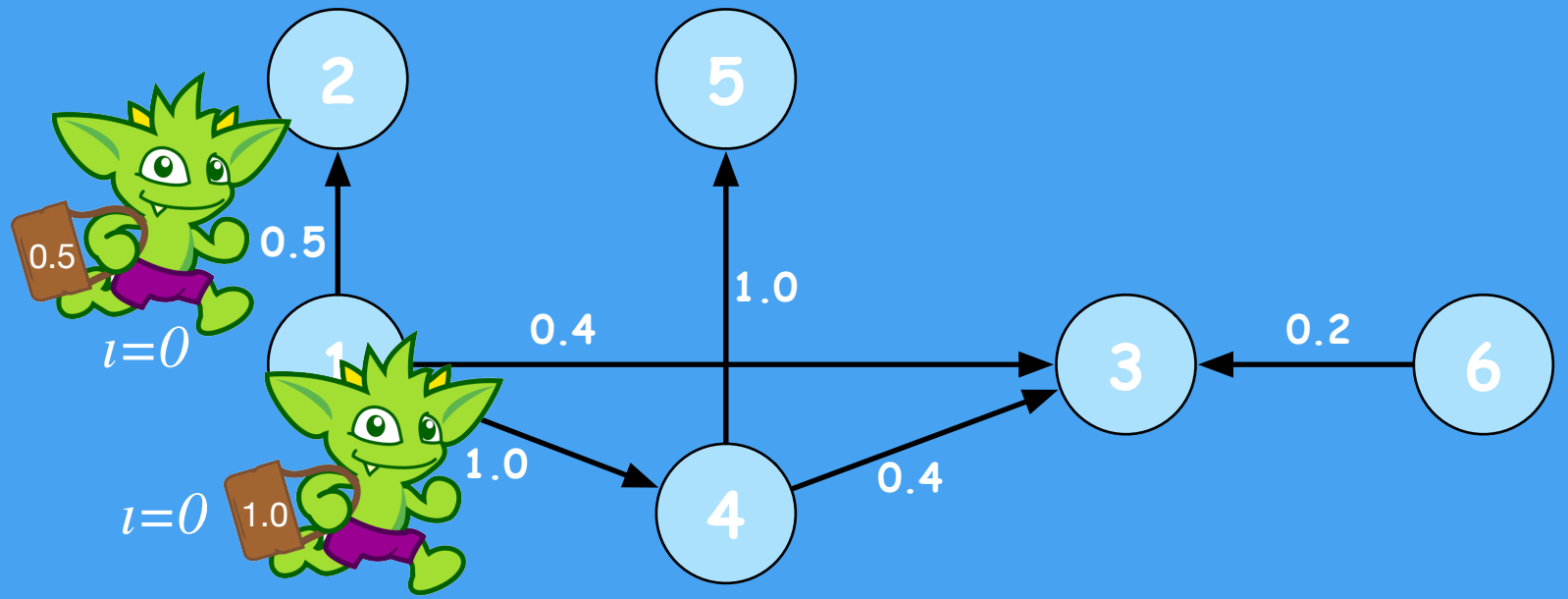
```
g.withSack(1).V(1).
  repeat(outE().sack(mult).by('weight').inV()).
  times(2).sack()
```



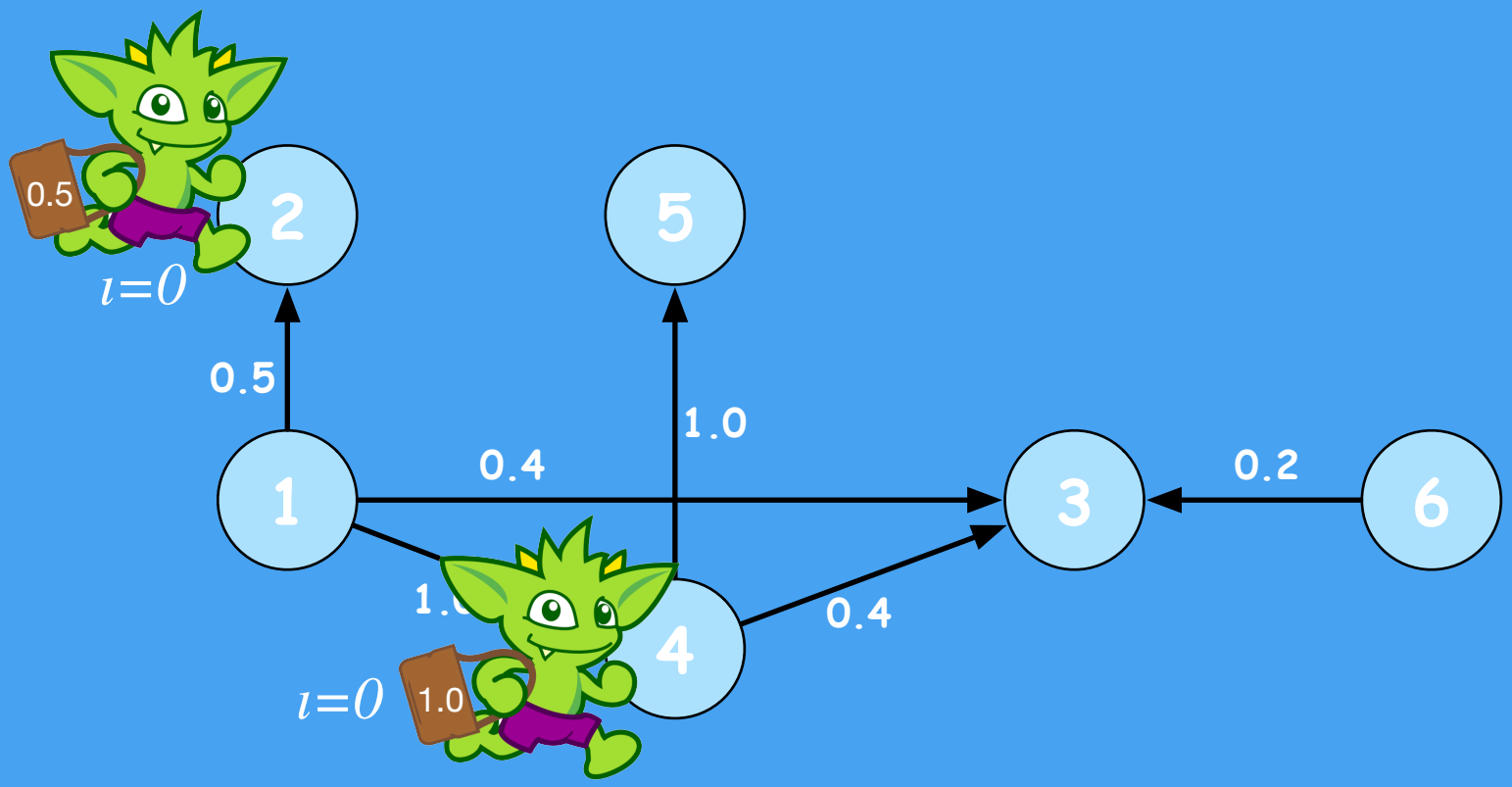
```
g.withSack(1).V(1).
  repeat(outE().sack(mult).by('weight').inV()).
  times(2).sack()
```



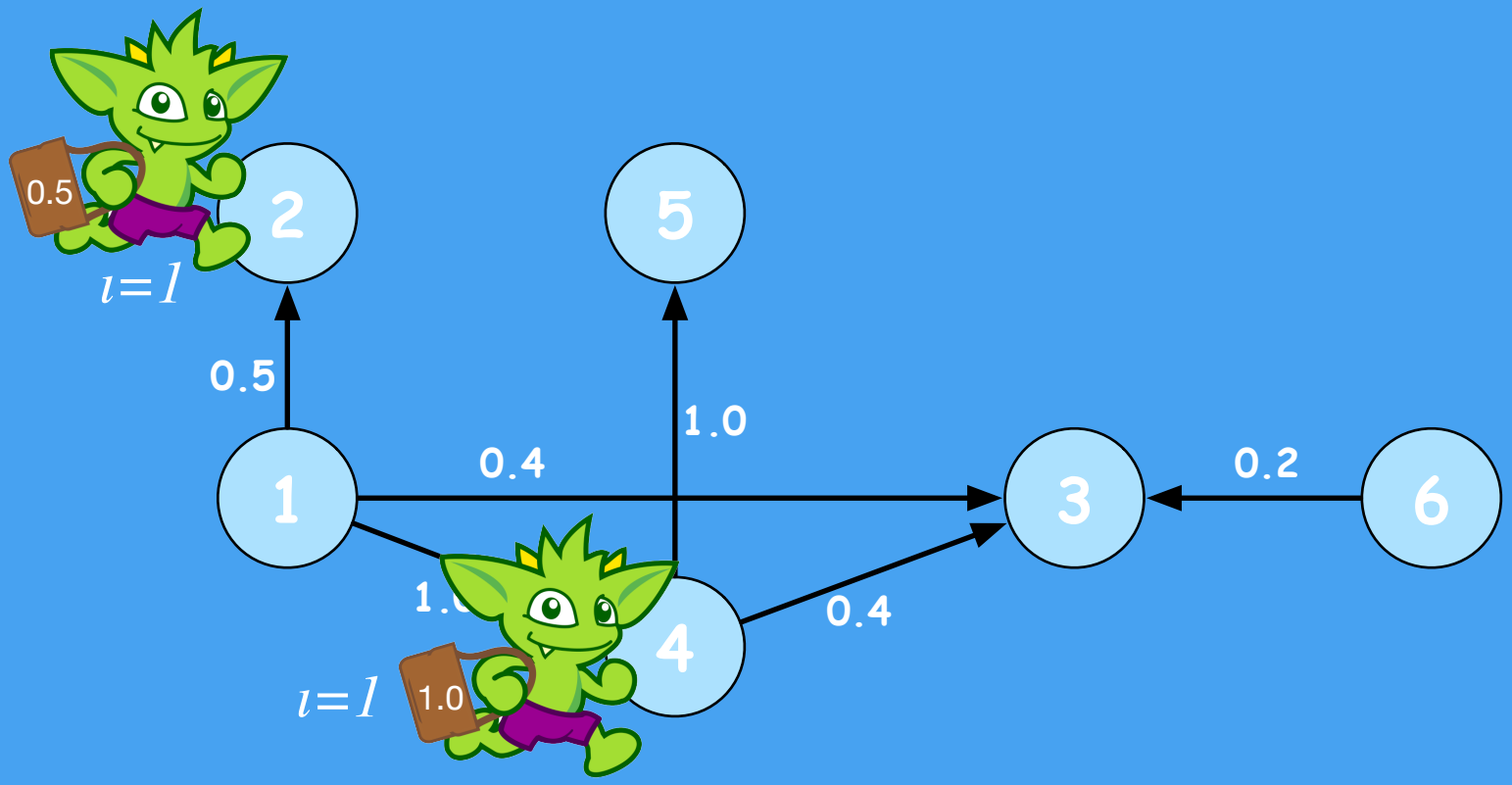
```
g.withSack(1).V(1).
  repeat(outE().sack(mult).by('weight').inV()).
  times(2).sack()
```



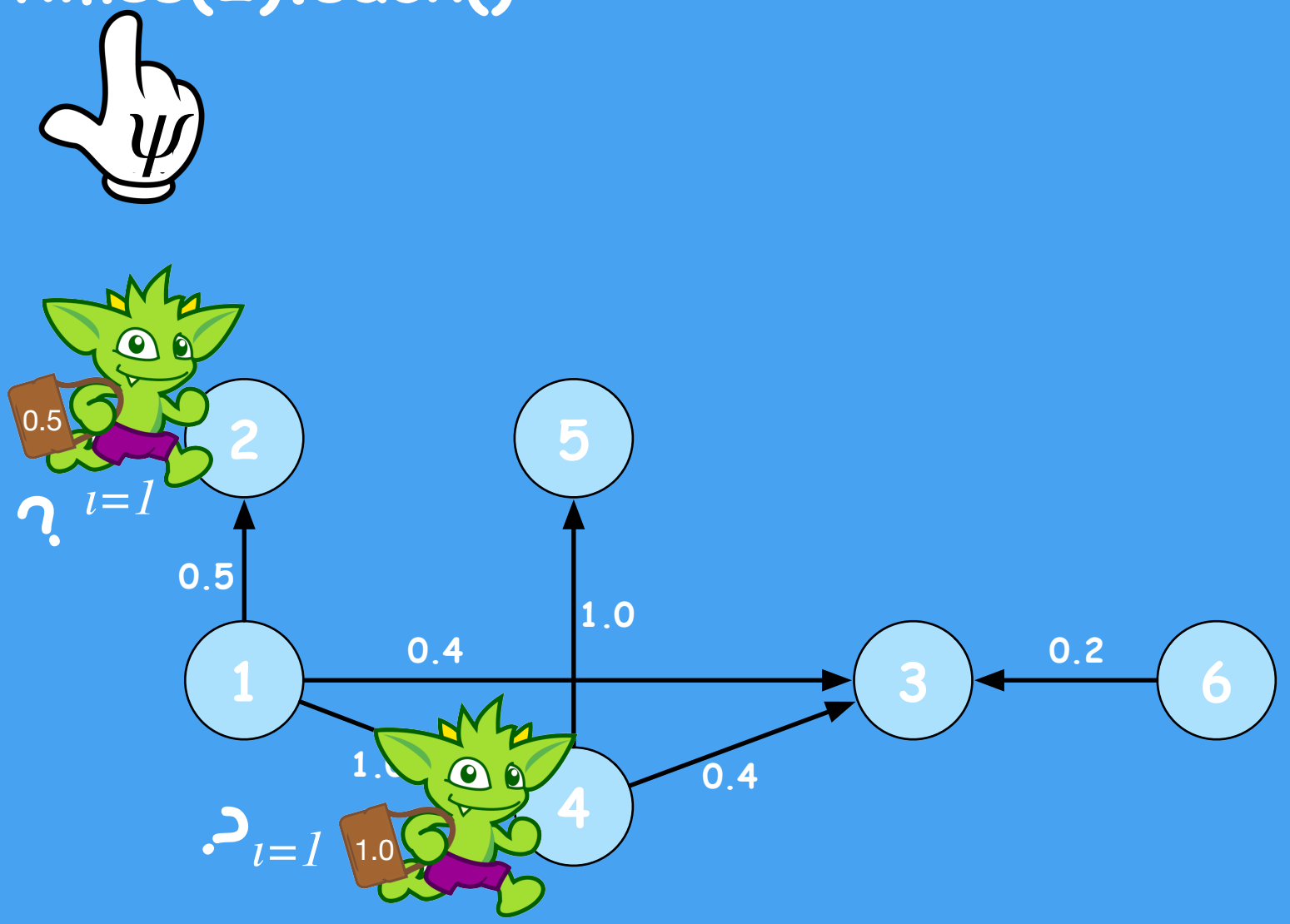
```
g.withSack(1).V(1).
  repeat(outE().sack(mult).by('weight').inV()).
  times(2).sack()
```



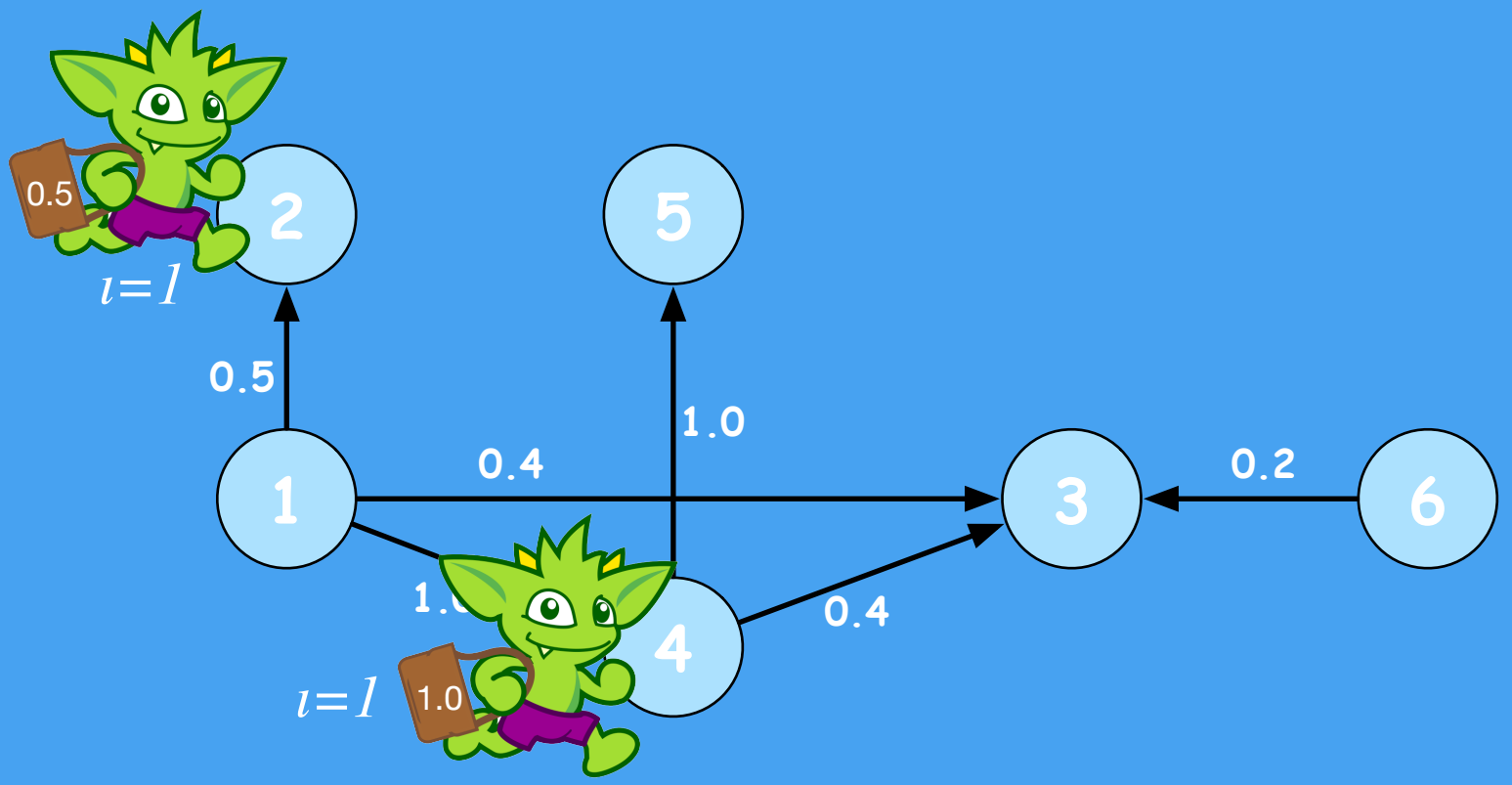
```
g.withSack(1).V(1).
  repeat(outE().sack(mult).by('weight').inV()).
  times(2).sack()
```



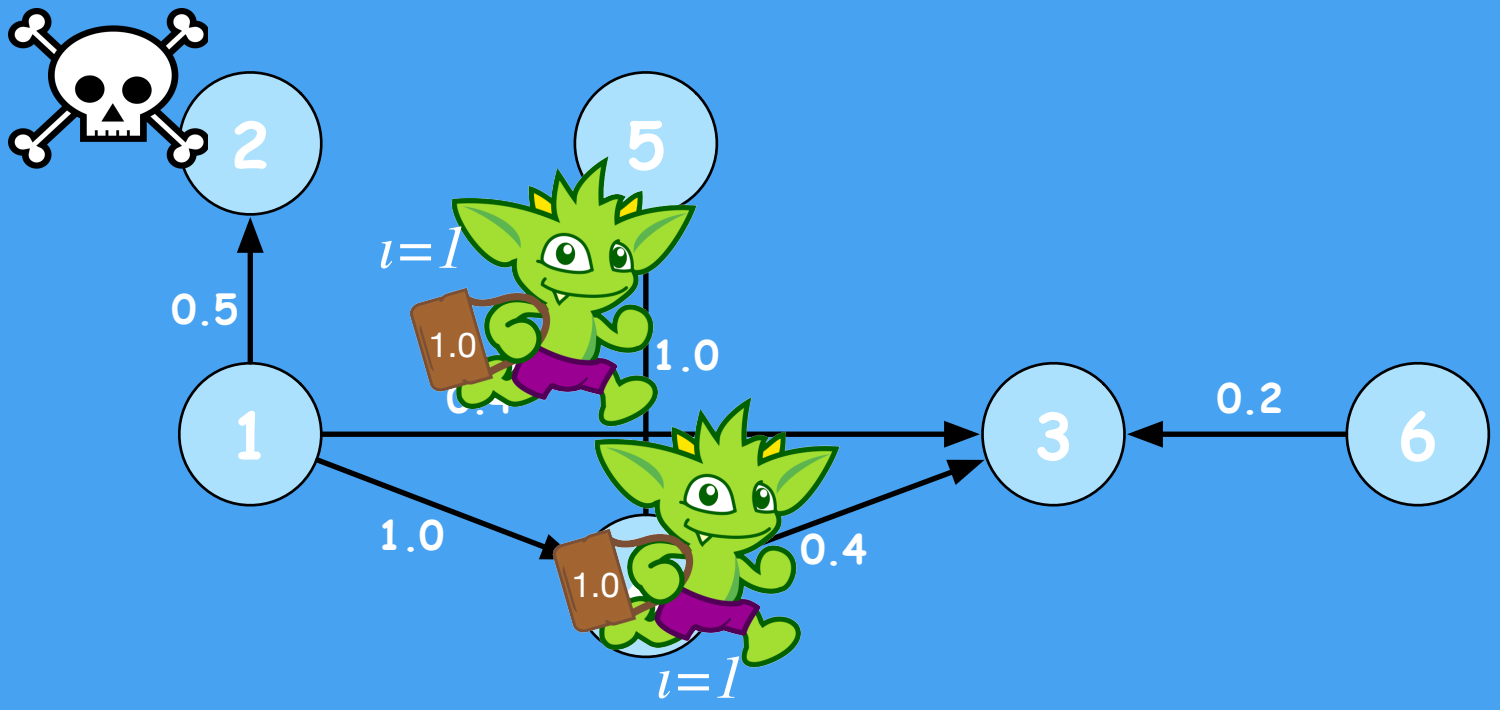
```
g.withSack(1).V(1).
  repeat(outE().sack(mult).by('weight').inV()).
  times(2).sack()
```



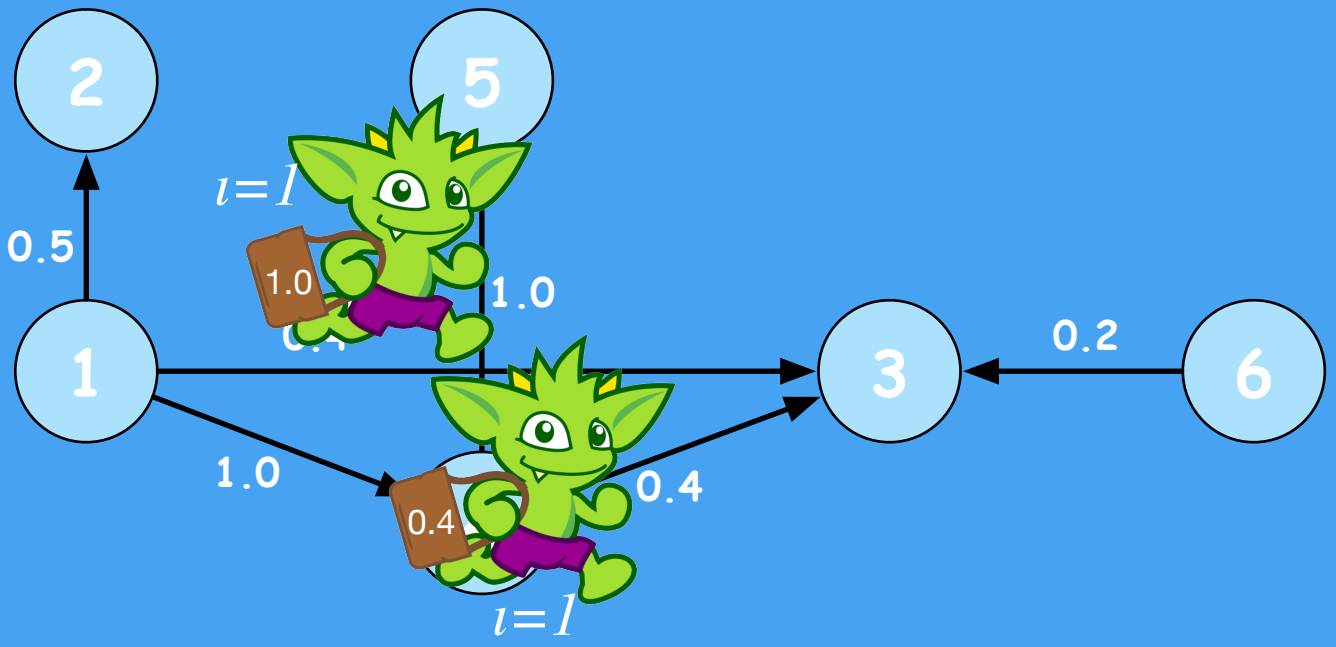
```
g.withSack(1).V(1).
  repeat(outE().sack(mult).by('weight').inV()).
  times(2).sack()
```



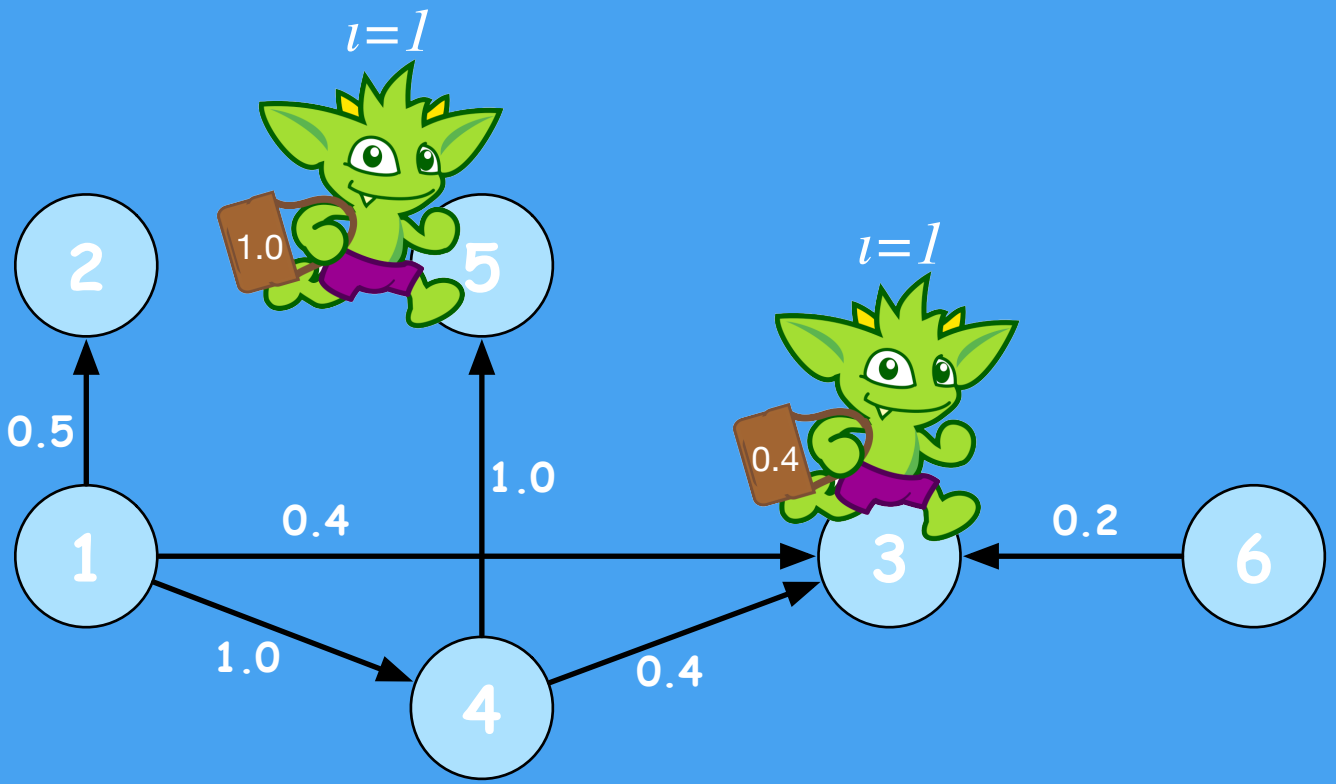
```
g.withSack(1).V(1).
  repeat(outE().sack(mult).by('weight').inV()).
  times(2).sack()
```



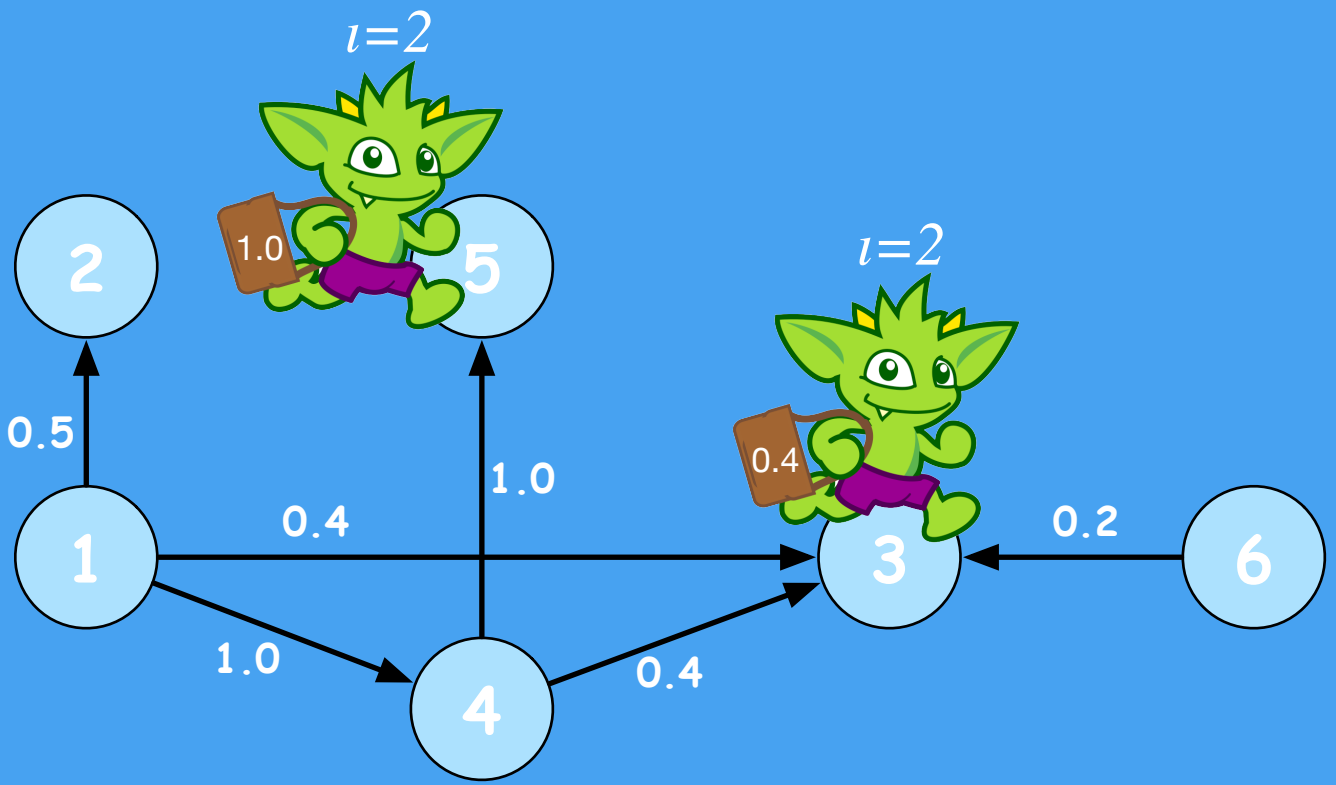
```
g.withSack(1).V(1).
  repeat(outE().sack(mult).by('weight').inV()).
  times(2).sack()
```



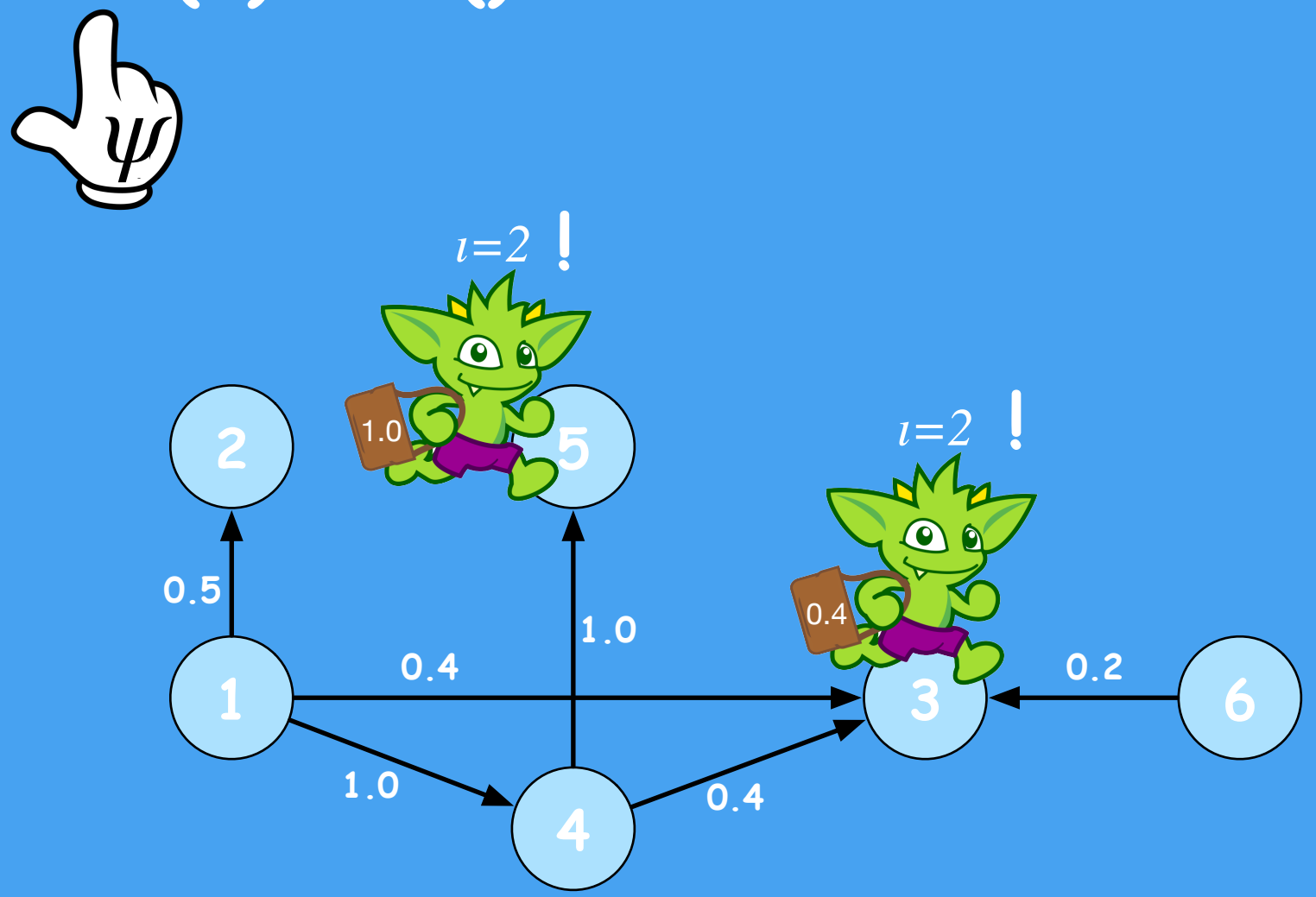
```
g.withSack(1).V(1).
  repeat(outE().sack(mult).by('weight').inV()).
  times(2).sack()
```



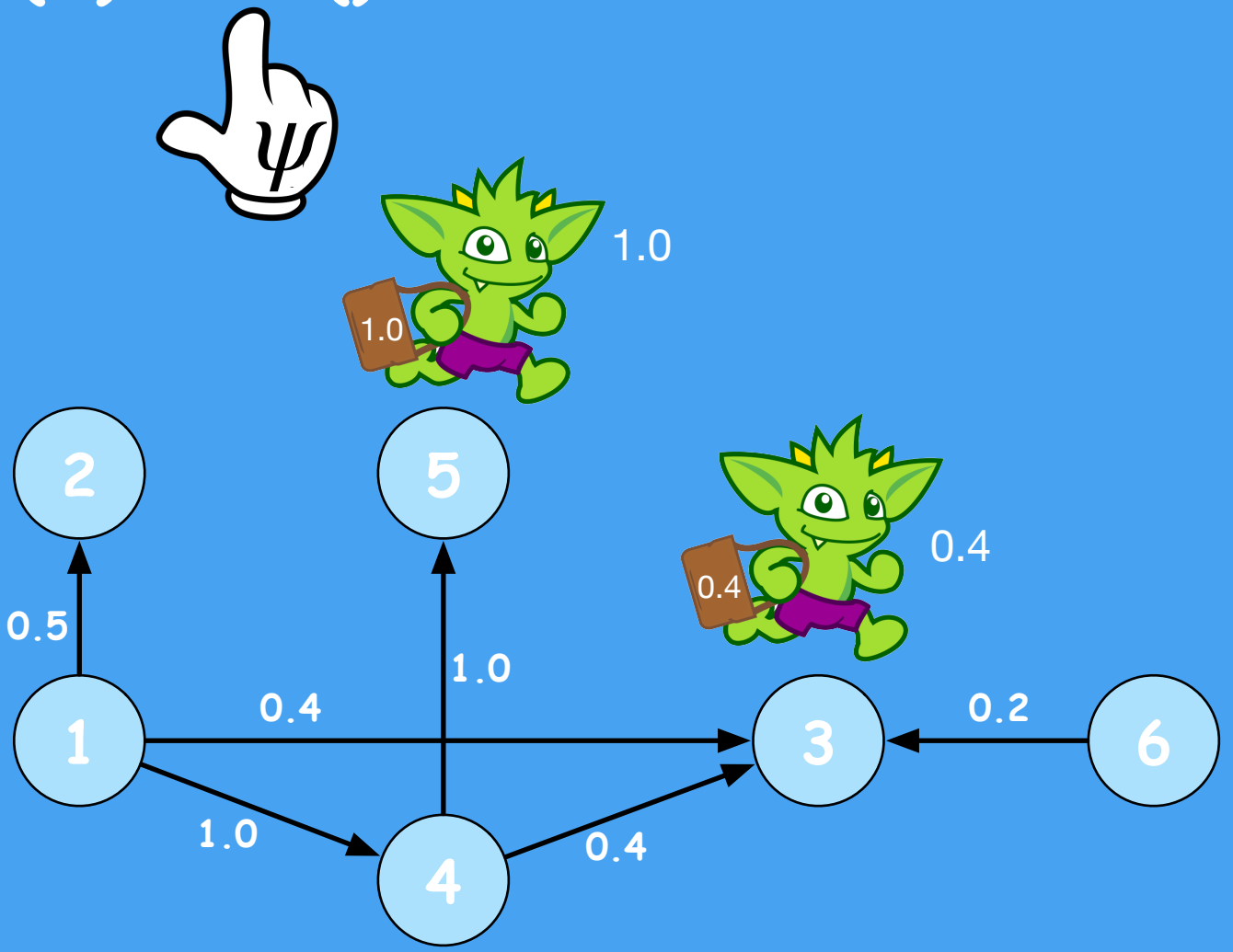
```
g.withSack(1).V(1).
  repeat(outE().sack(mult).by('weight').inV()).
  times(2).sack()
```



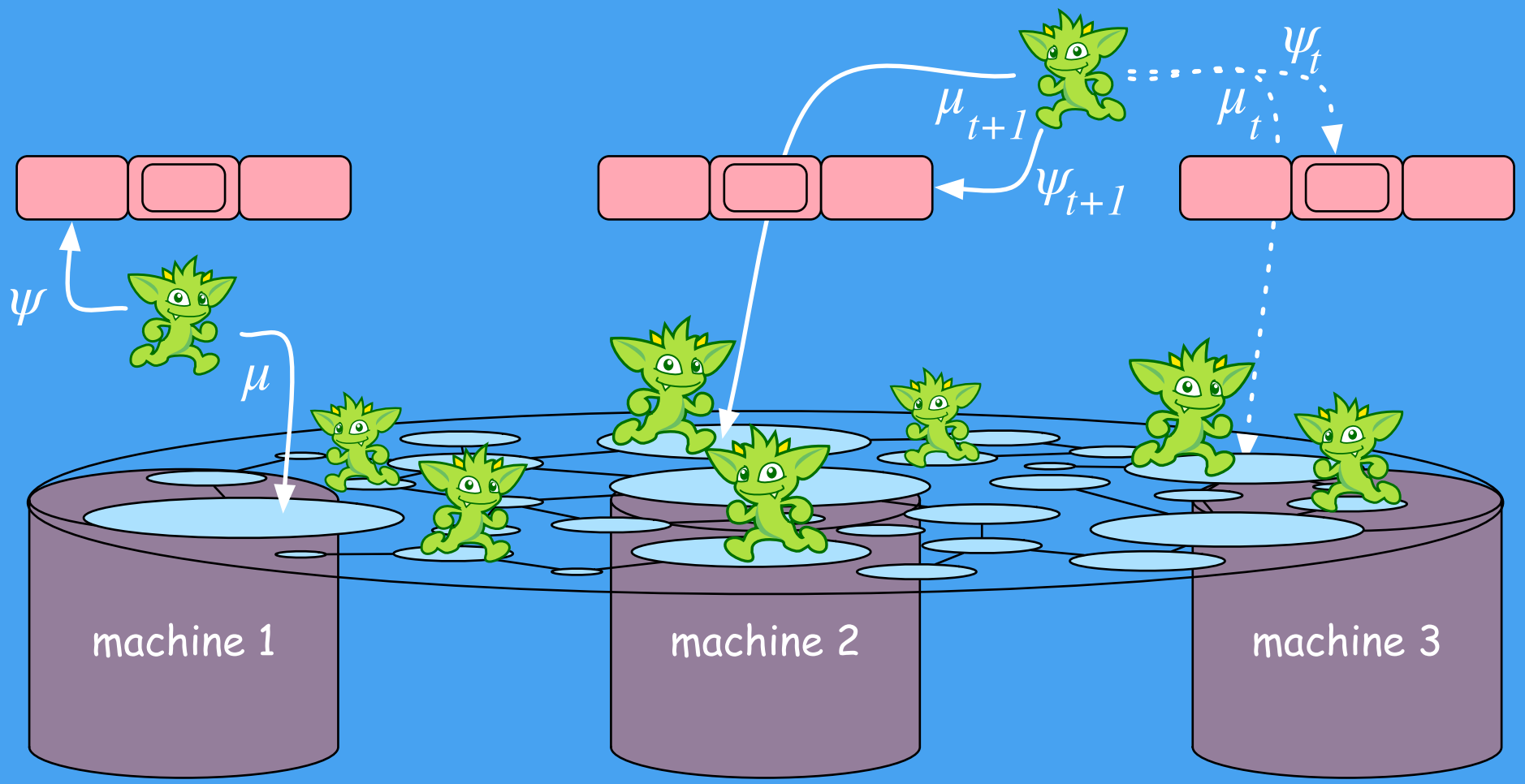
```
g.withSack(1).V(1).
  repeat(outE().sack(mult).by('weight').inV()).
  times(2).sack()
```



```
g.withSack(1).V(1).
  repeat(outE().sack(mult).by('weight').inV()).
  times(2).sack()
```



The Gremlin traversal machine is a distributed virtual machine — traversers can independently move around the graph (across machines) and through the traversal.



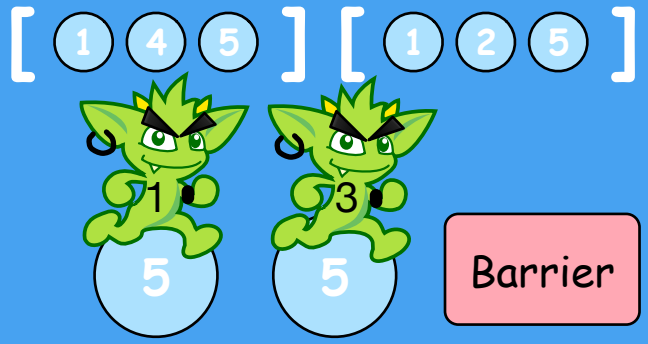
Quiz

To Bulk or Not To Bulk

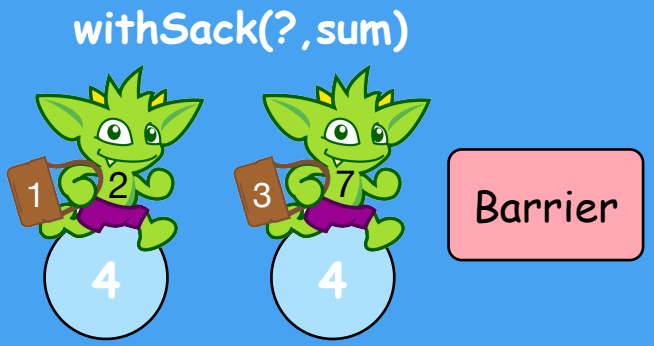
Barrier

To Bulk or Not To Bulk

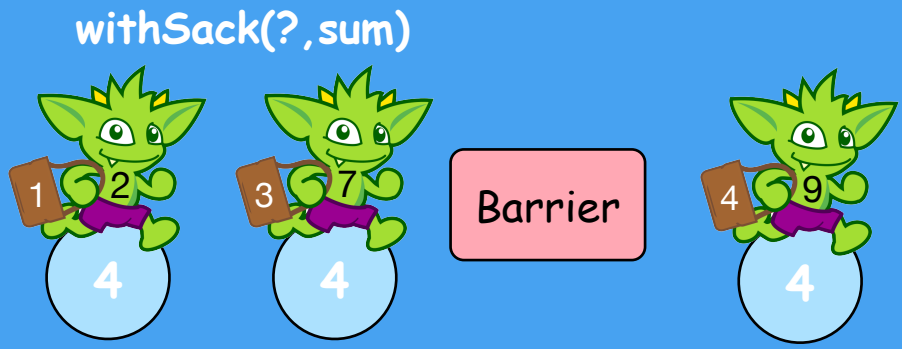
To Bulk or Not To Bulk

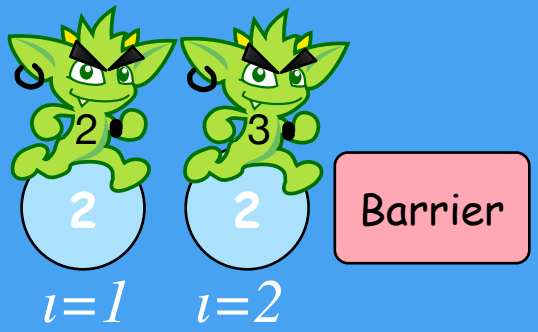


To Bulk or Not To Bulk

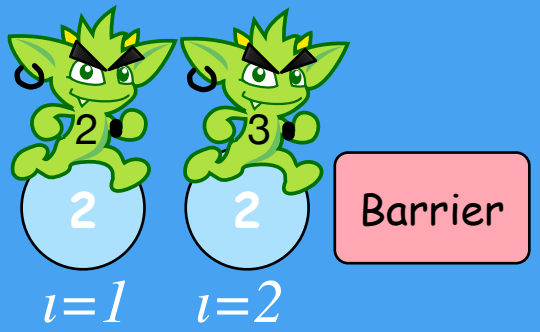


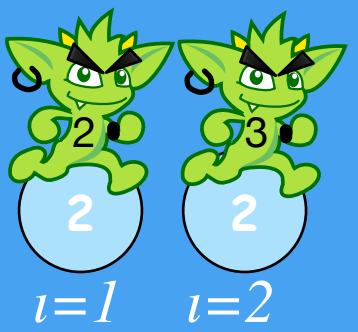
To Bulk or Not To Bulk

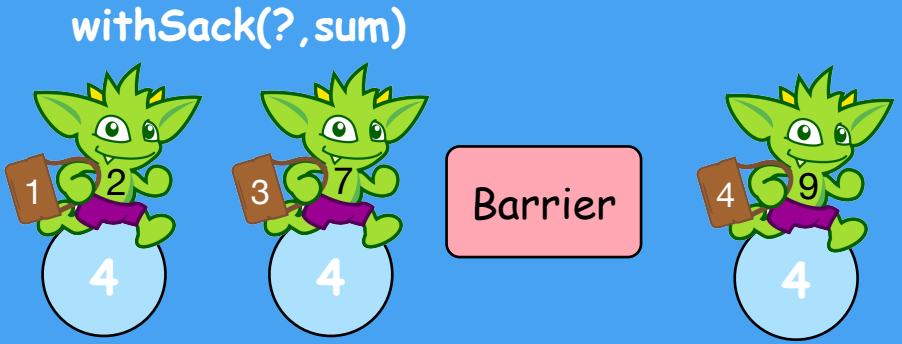




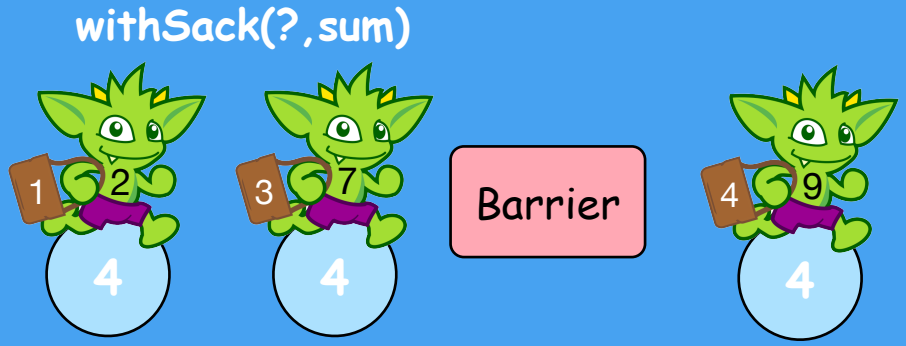
To Bulk or Not To Bulk

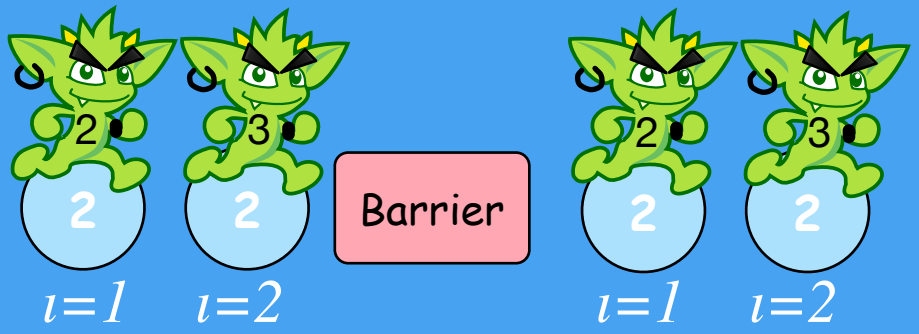




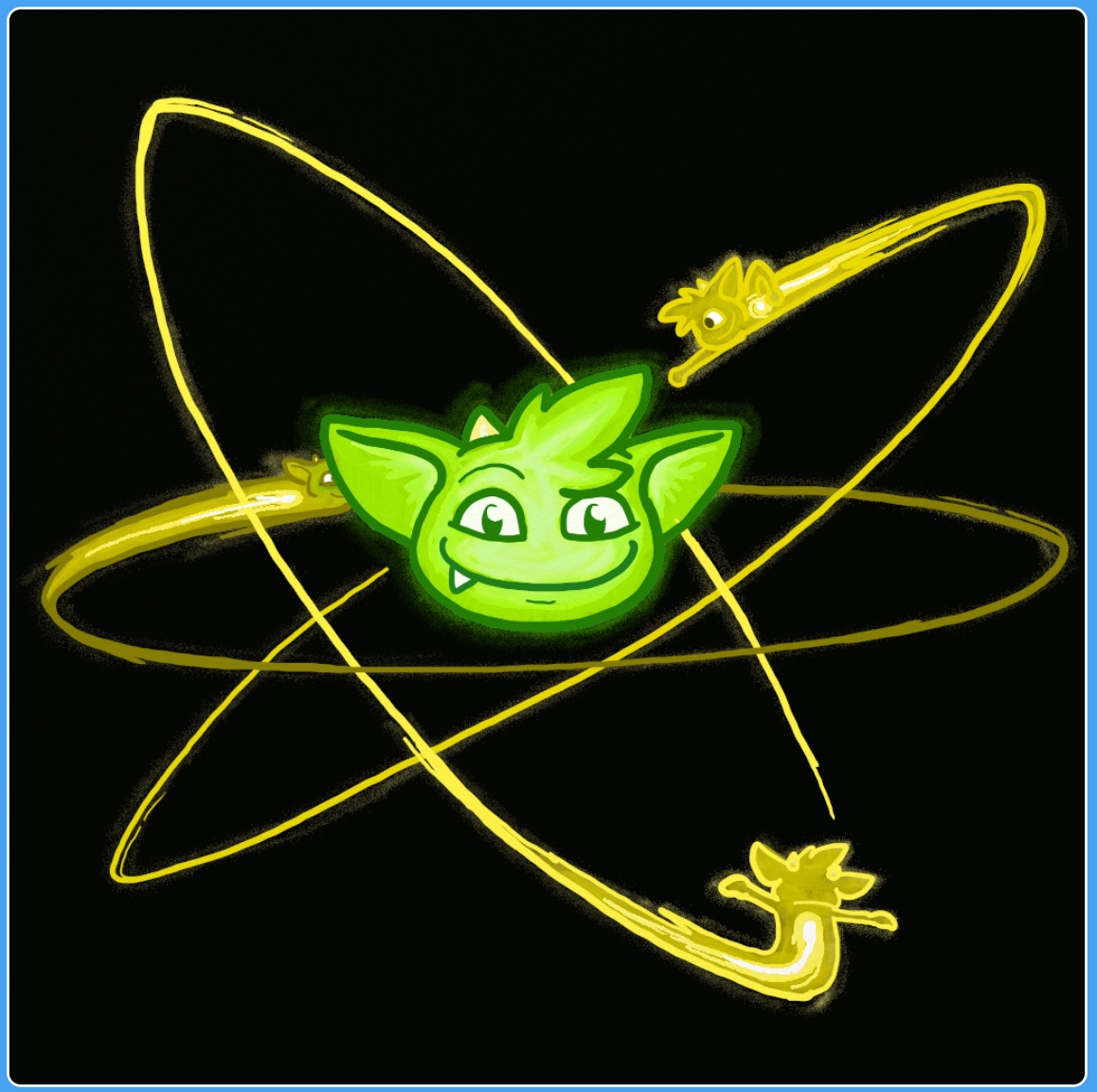


To Bulk or Not To Bulk





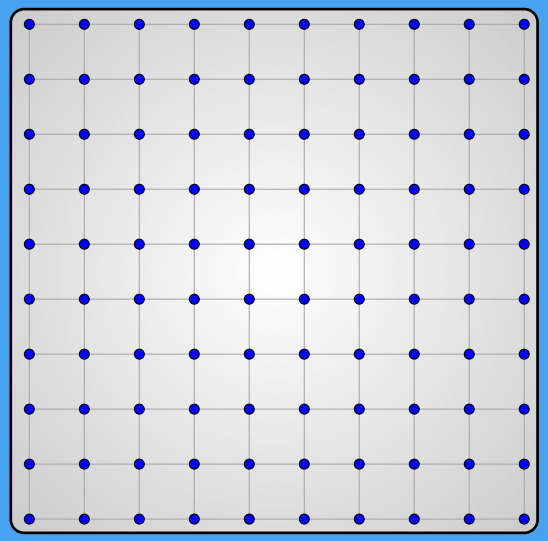
The traverser is an atomic unit of computing — moving through both the traversal and the graph.



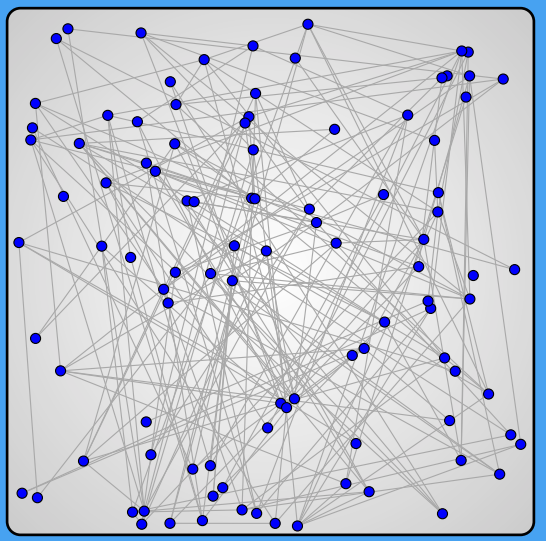
G

The Graph

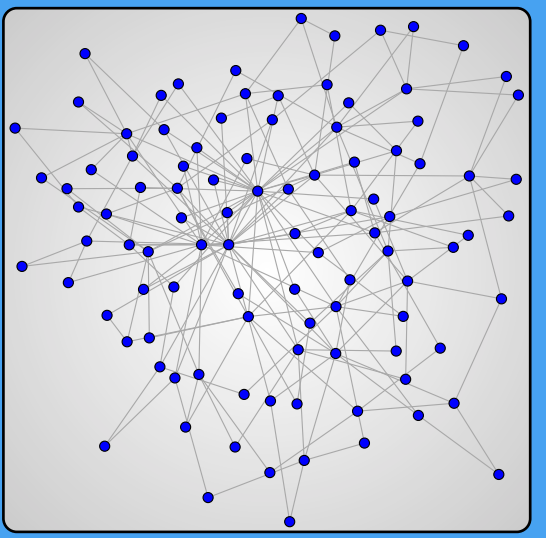
Lattice



Scale-Free

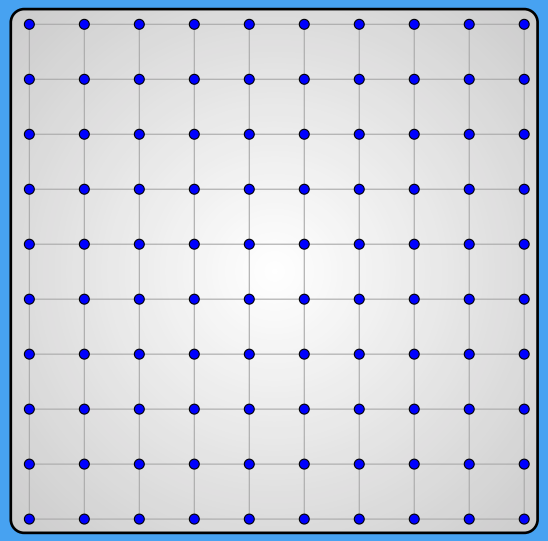


Random

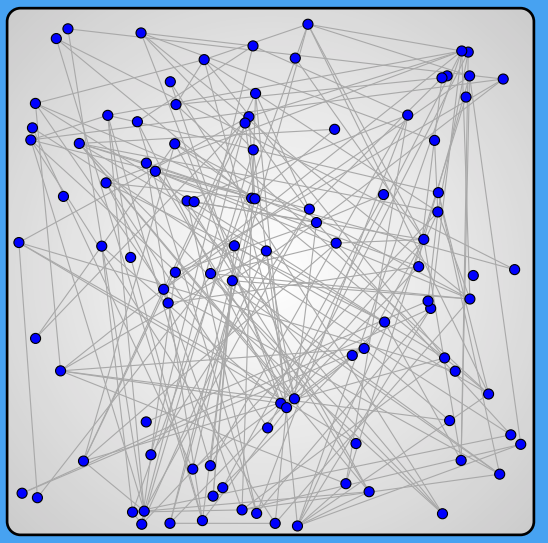


Rodriguez, M.A., "Graphoendodonticology," DataStax Blog, March 2017.
https://www.datastax.com/2017/03/graphoendodonticology

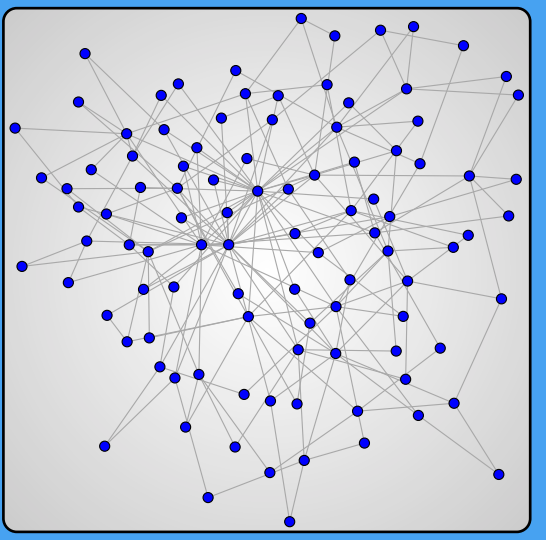
Lattice

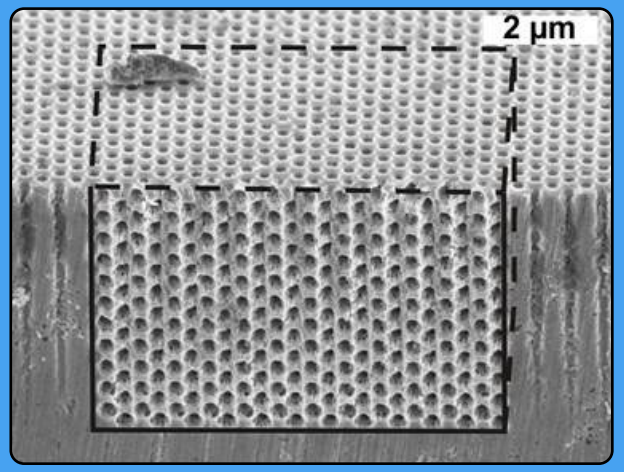


Scale-Free



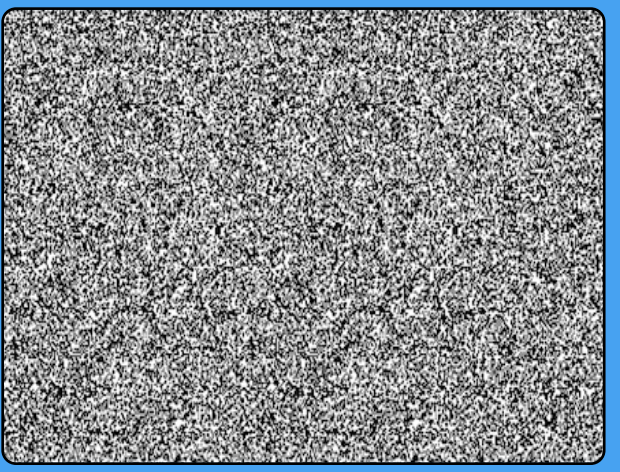
Random





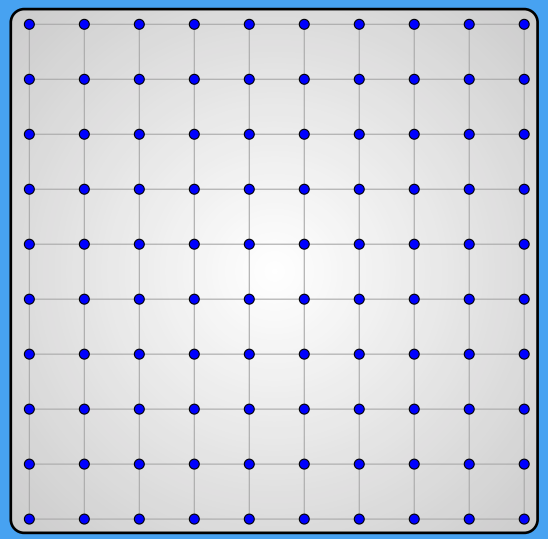
Diamond

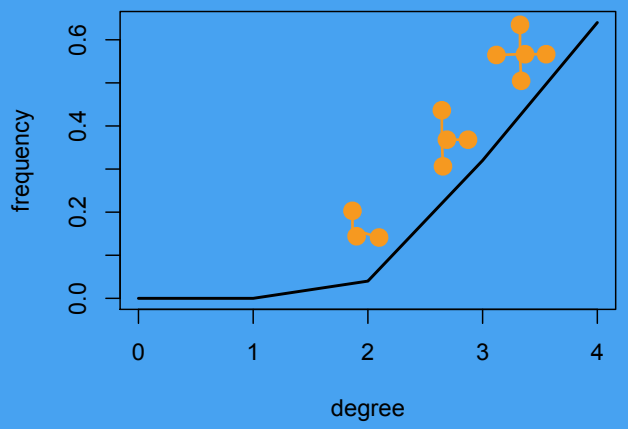
Jackson Pollock



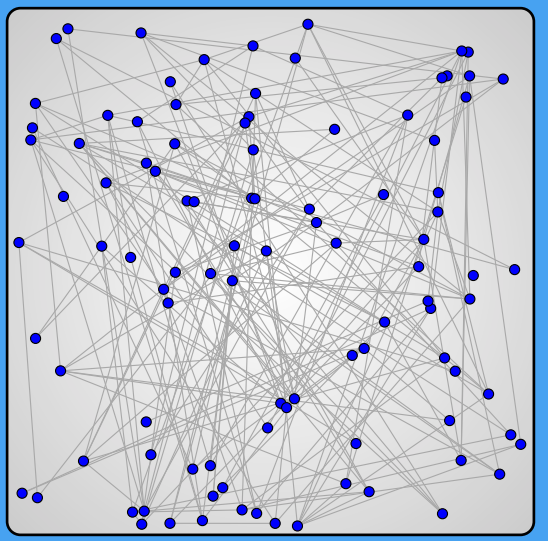
TV Fuzz

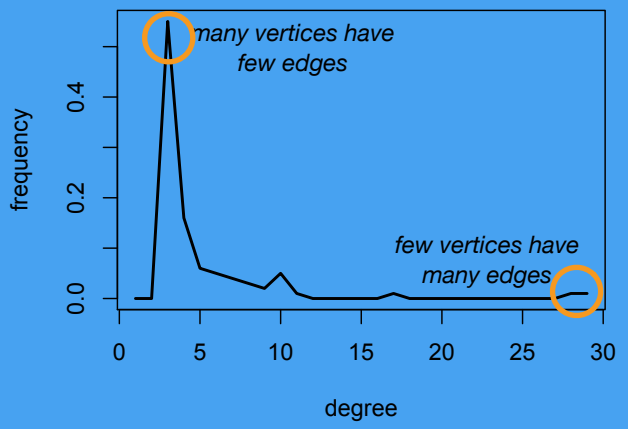
Lattice



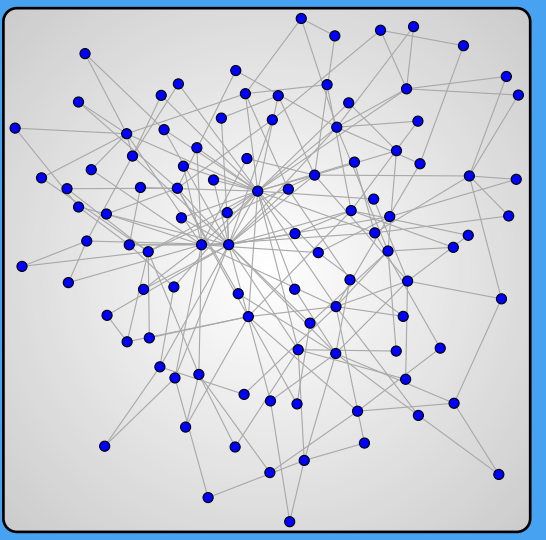


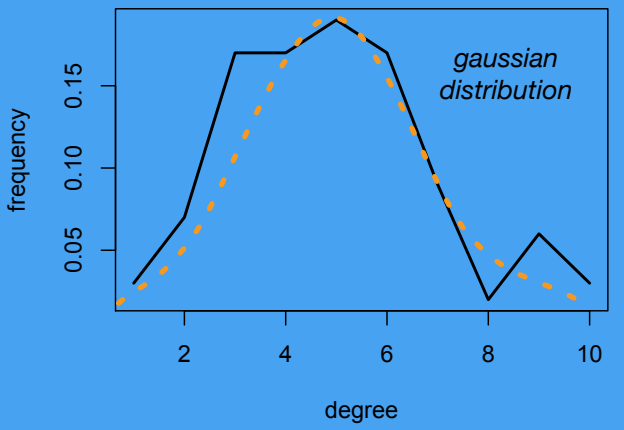
Scale-Free





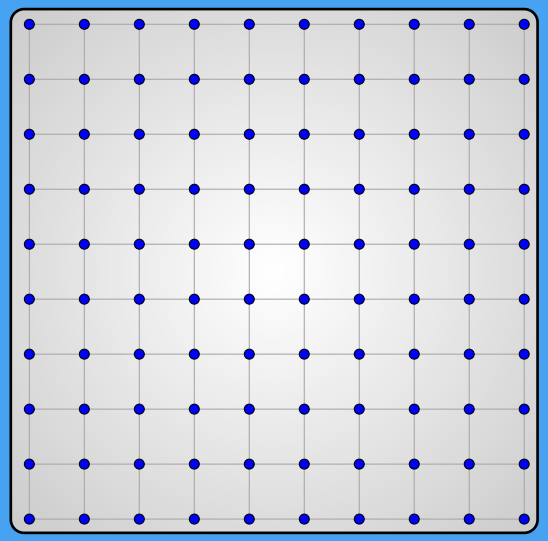
Random



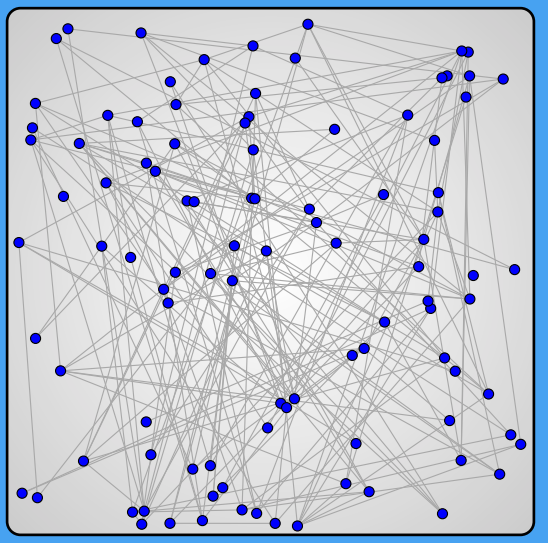



```
g.V().groupCount().by(both().count())
```

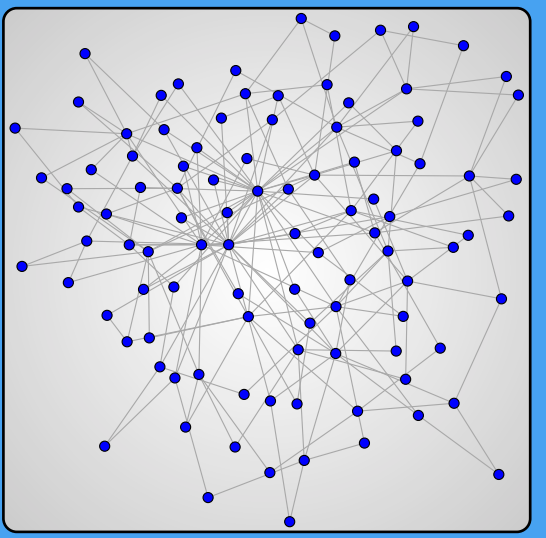
Lattice

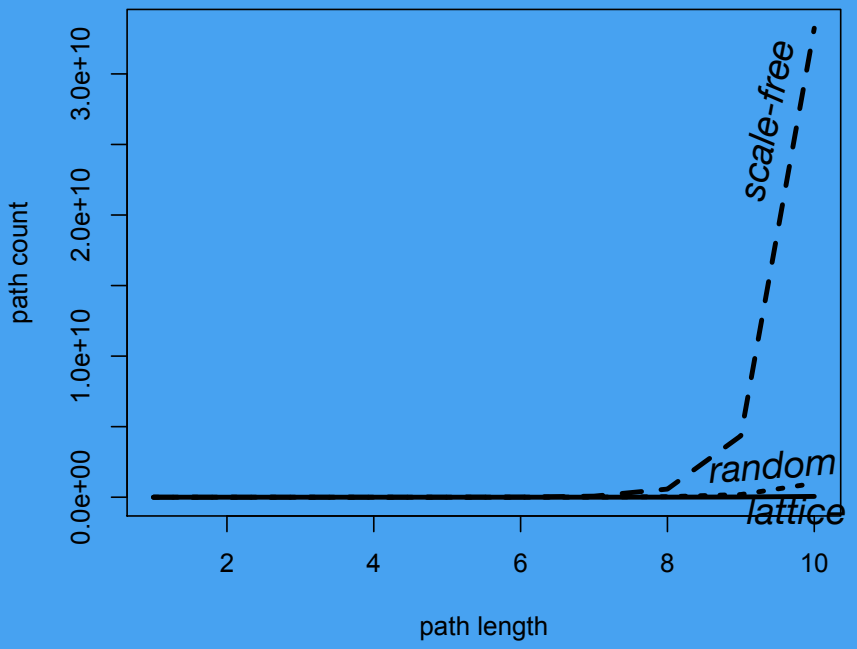


Scale-Free

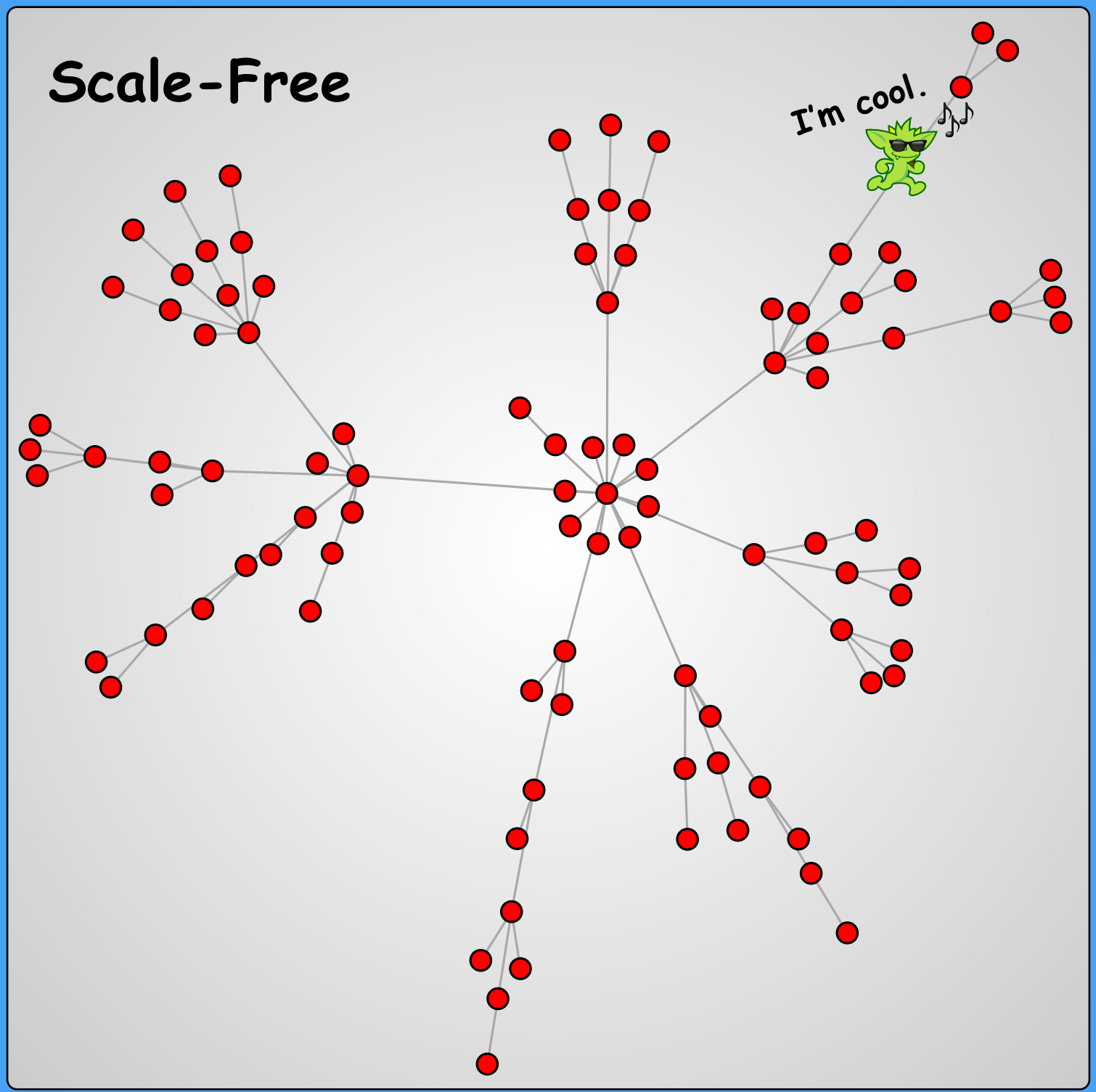


Random

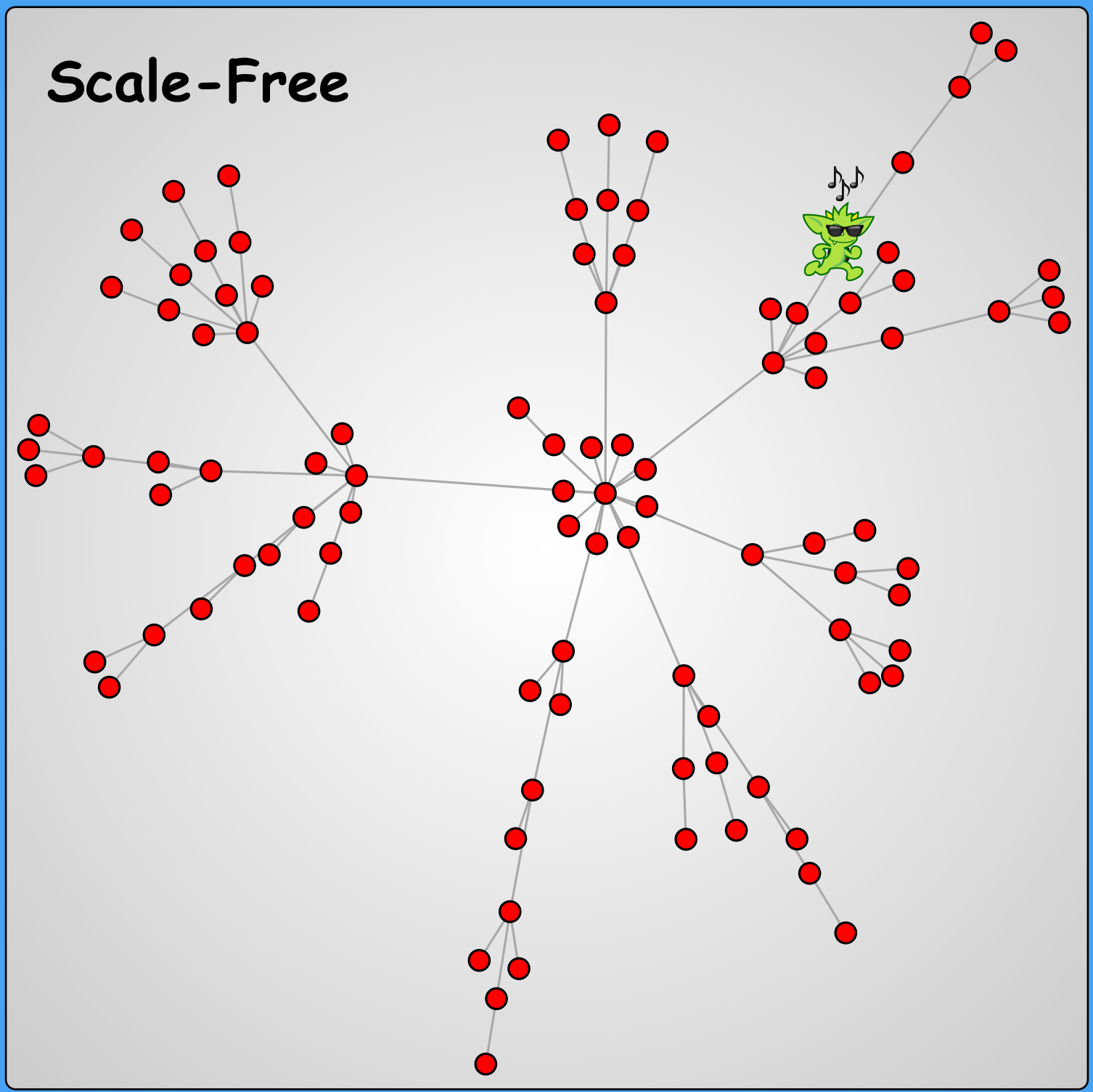




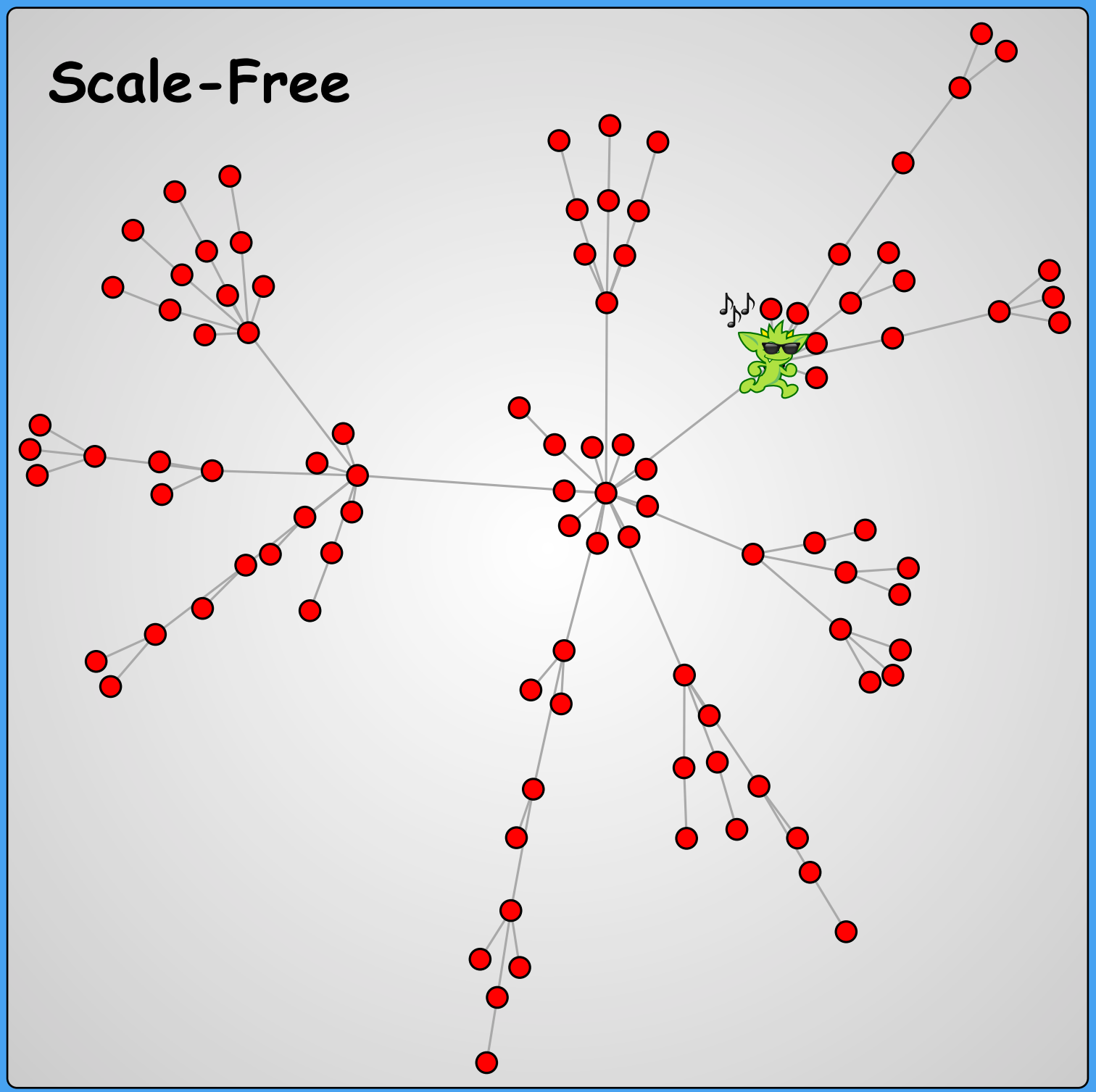

```
g.V().repeat(both()).times(x).count()
```


Scale-Free
I'm cool.

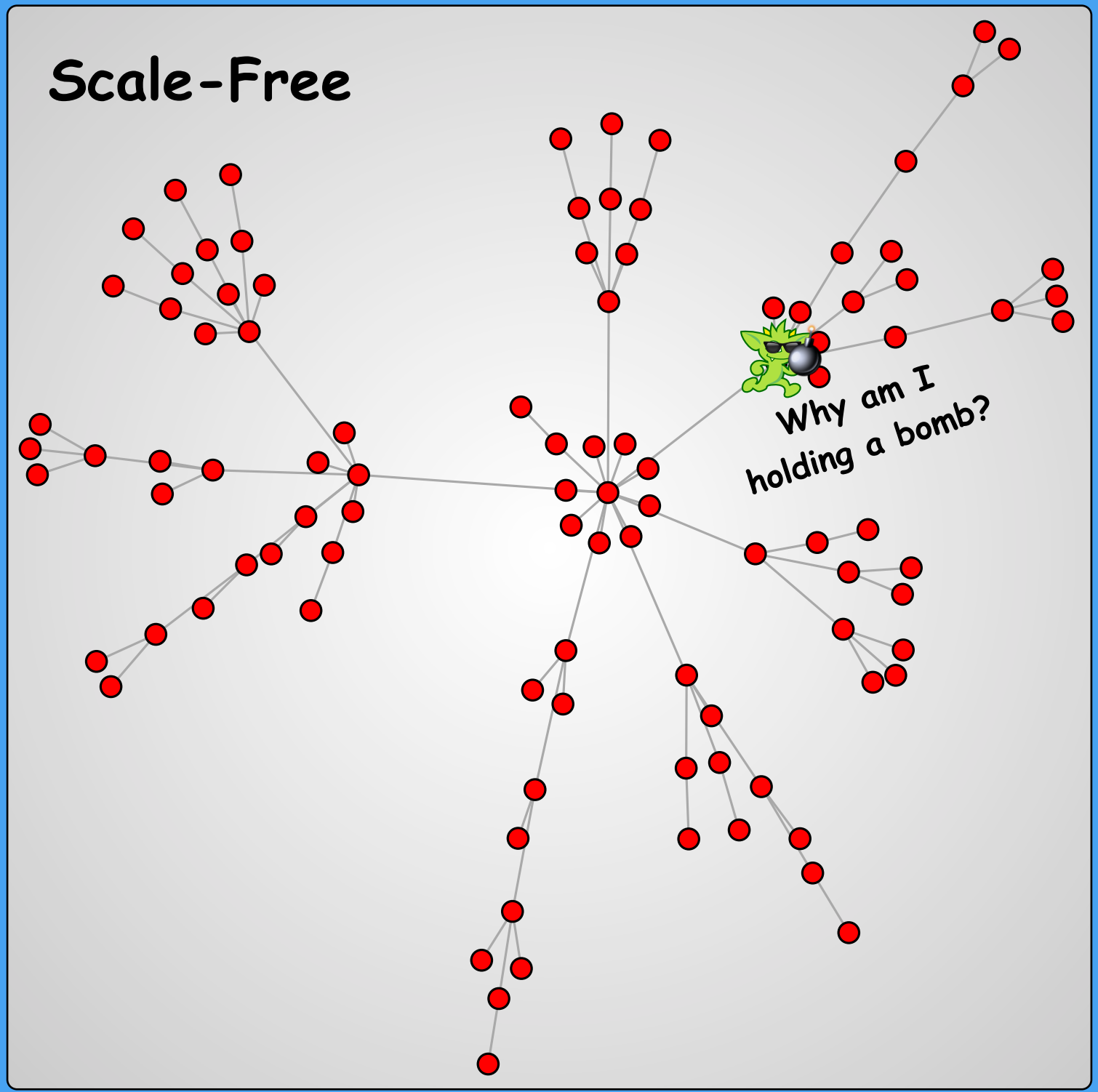
Scale-Free

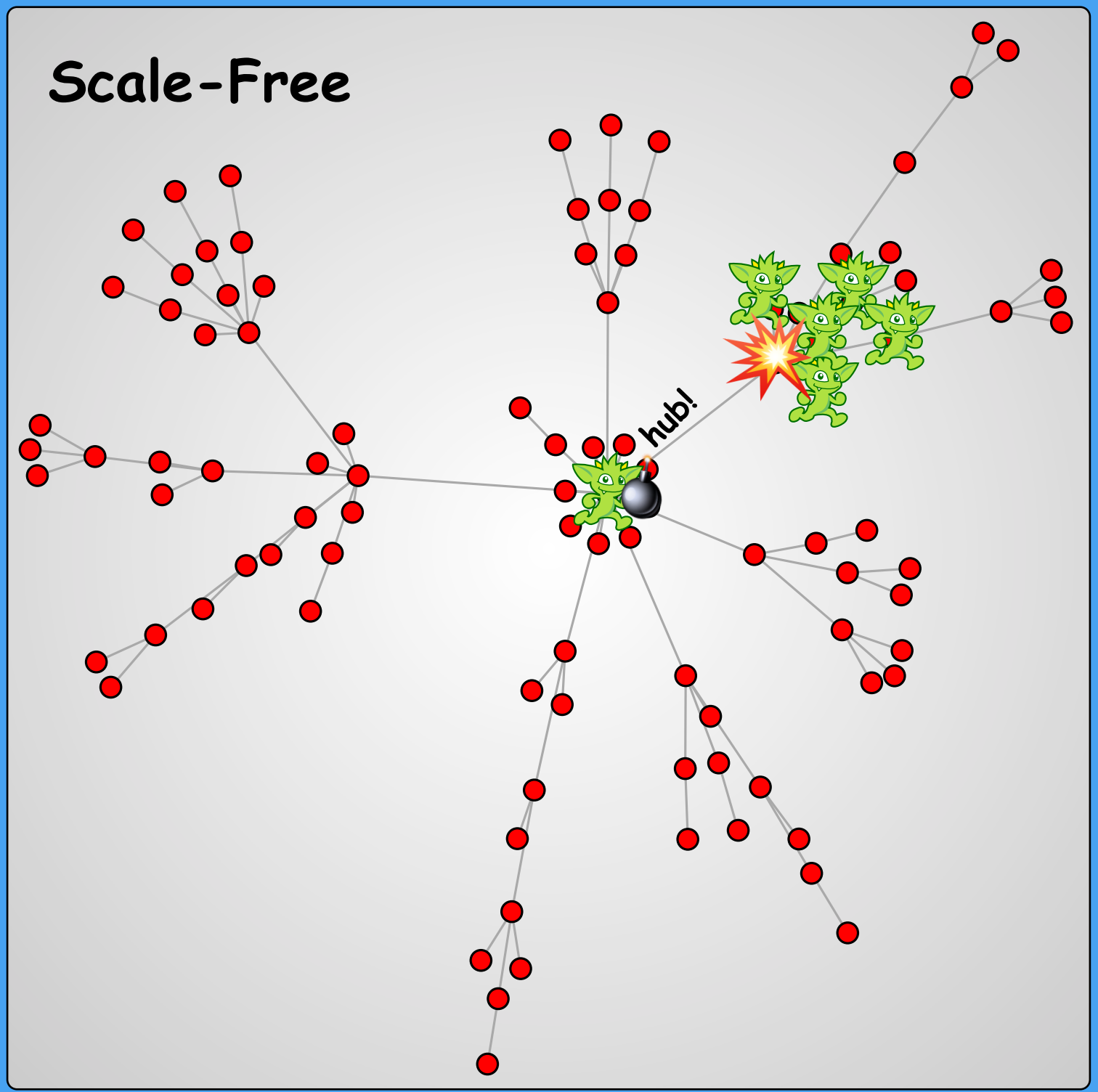


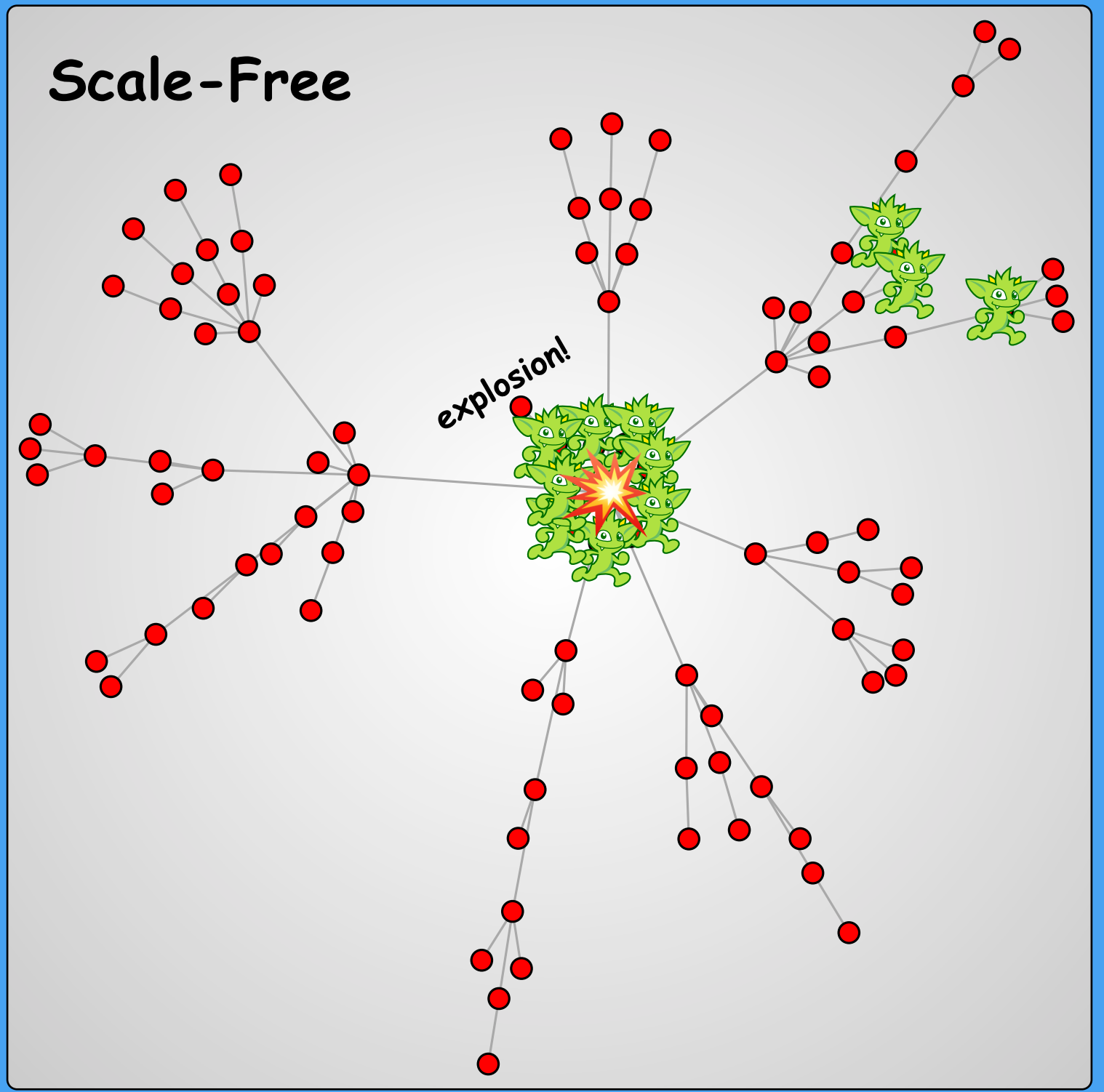
Scale-Free

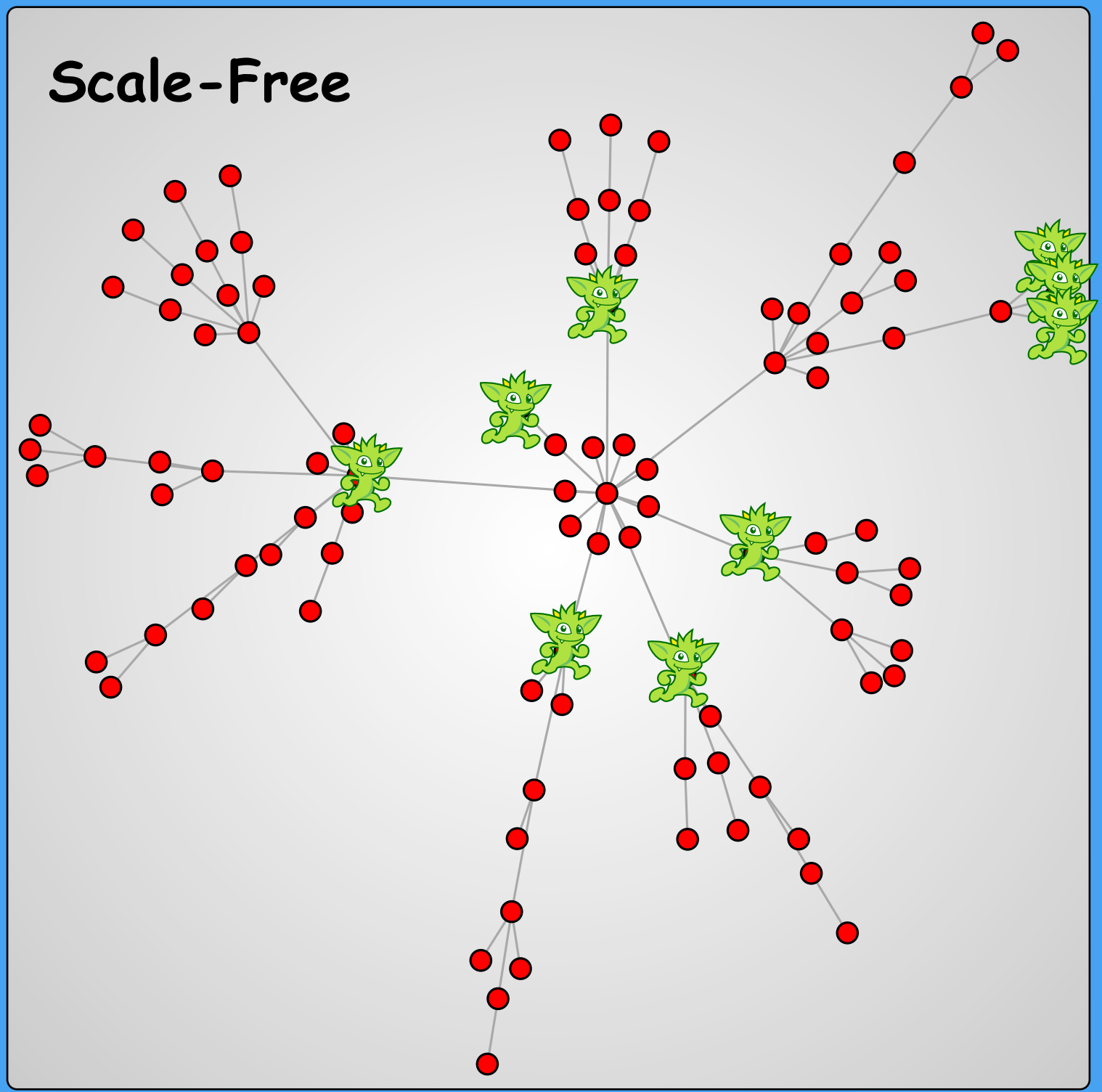


Scale-Free

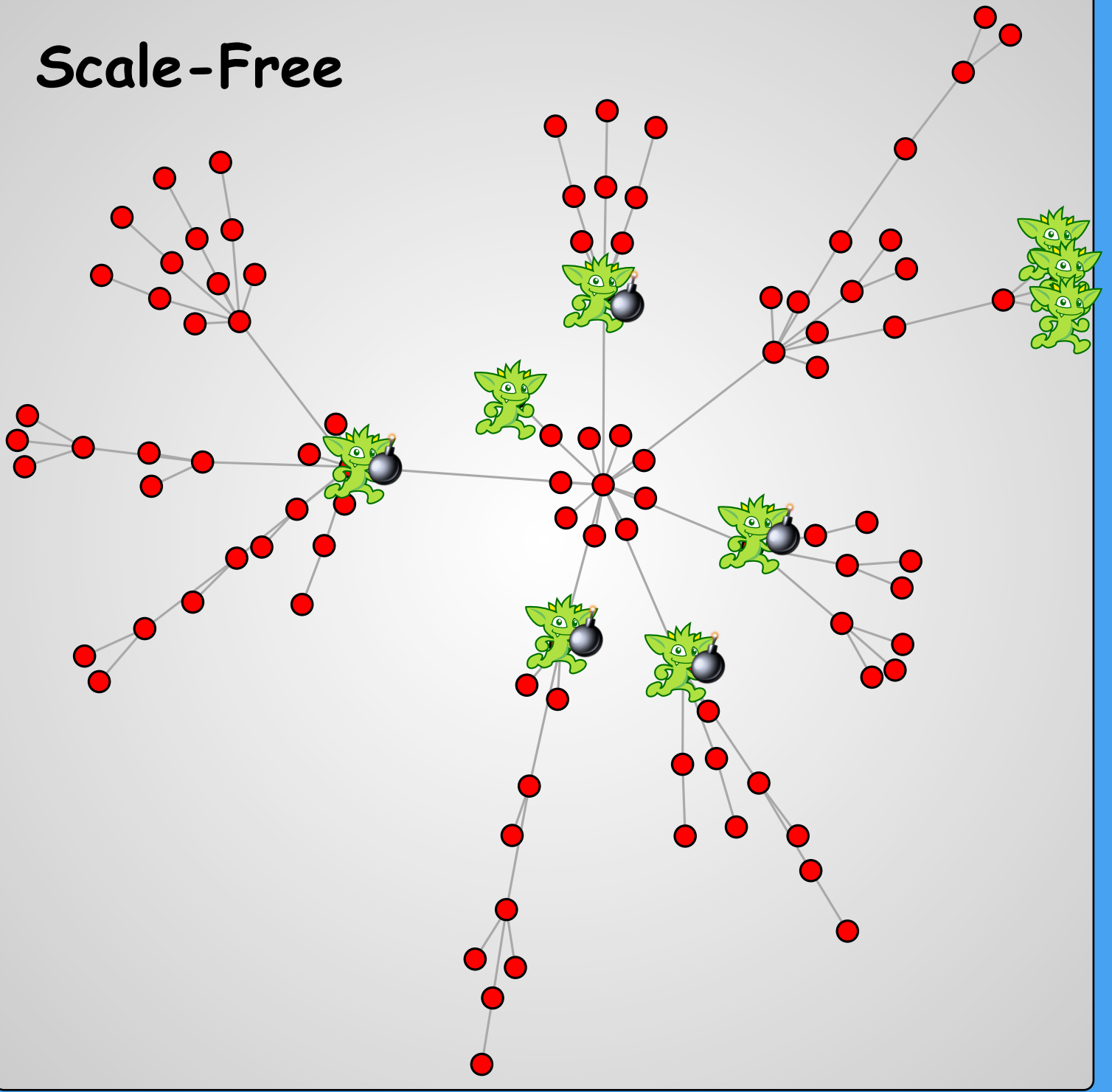


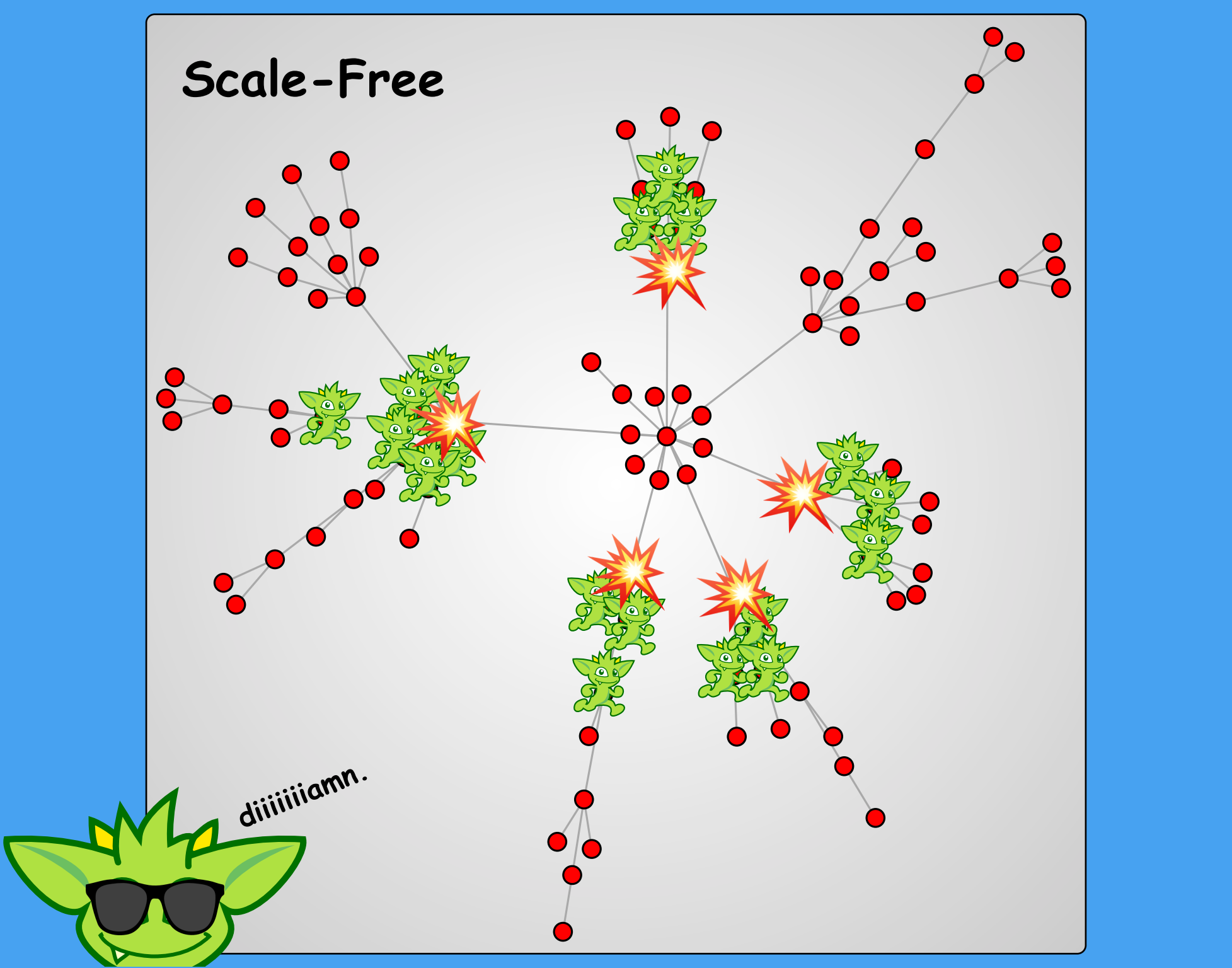

Scale-Free
hub!


Scale-Free
explosion!


Scale-Free

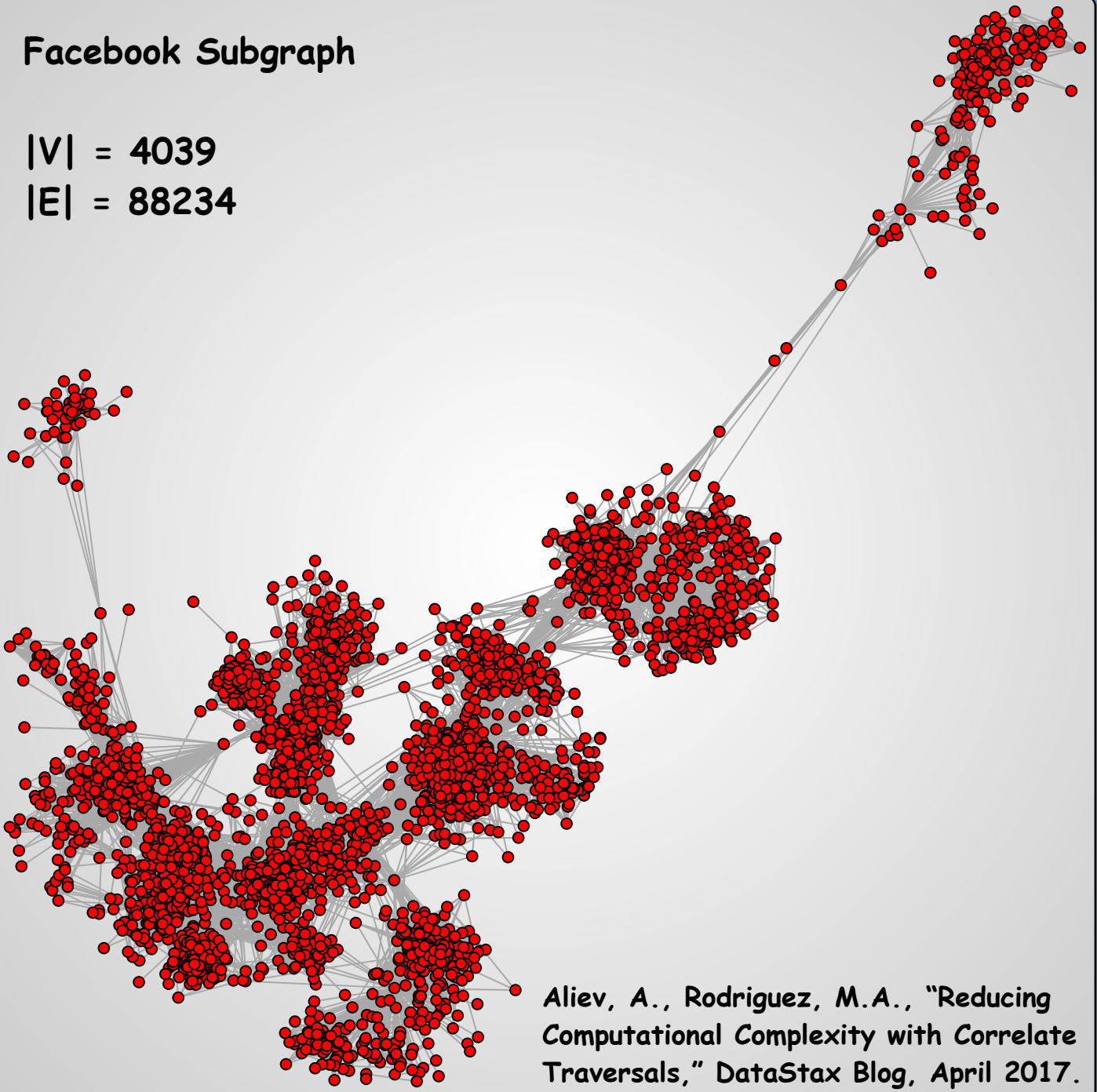
Scale-Free




Scale-Free
diiiiiiiamn.

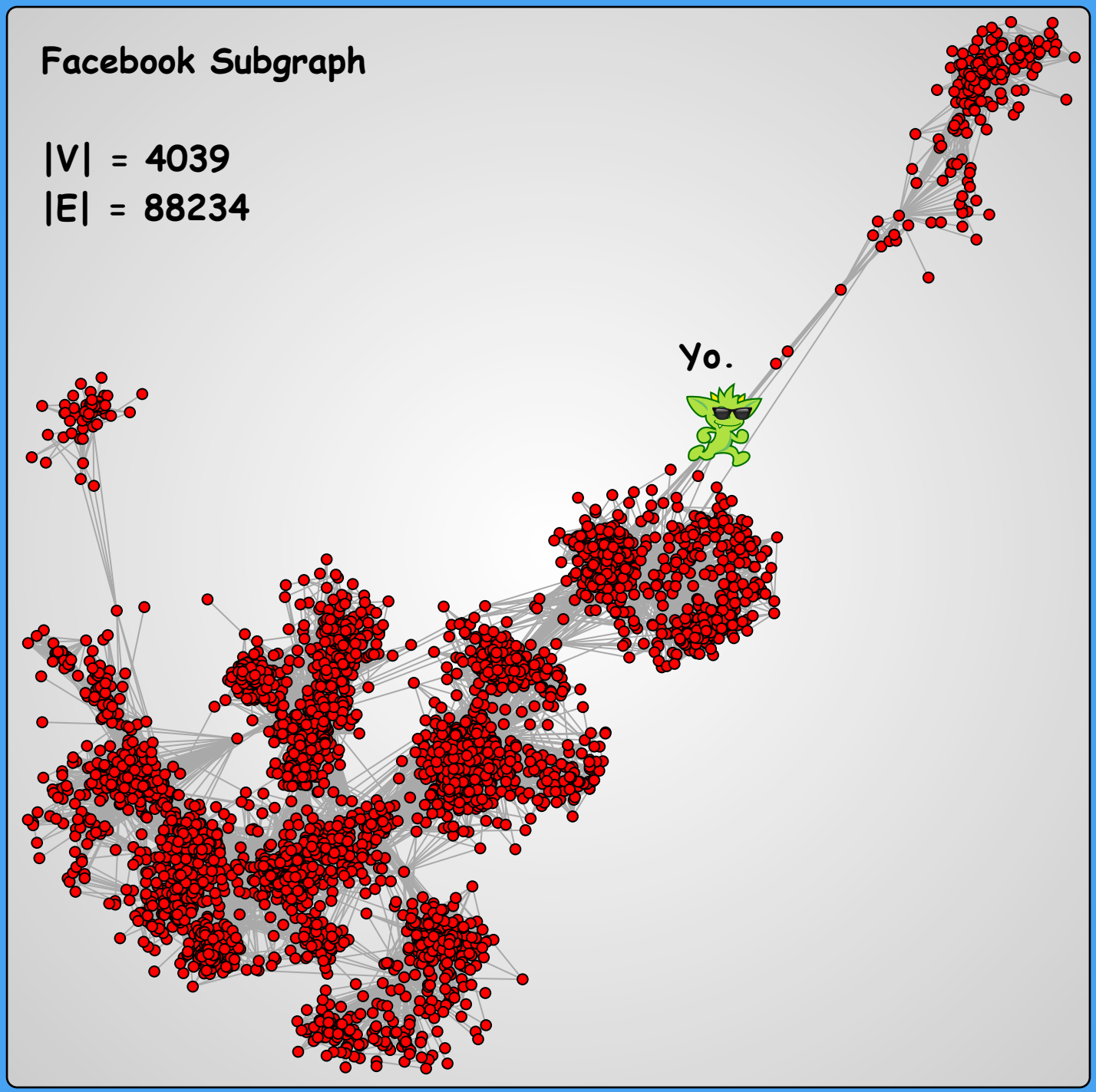
Facebook Subgraph

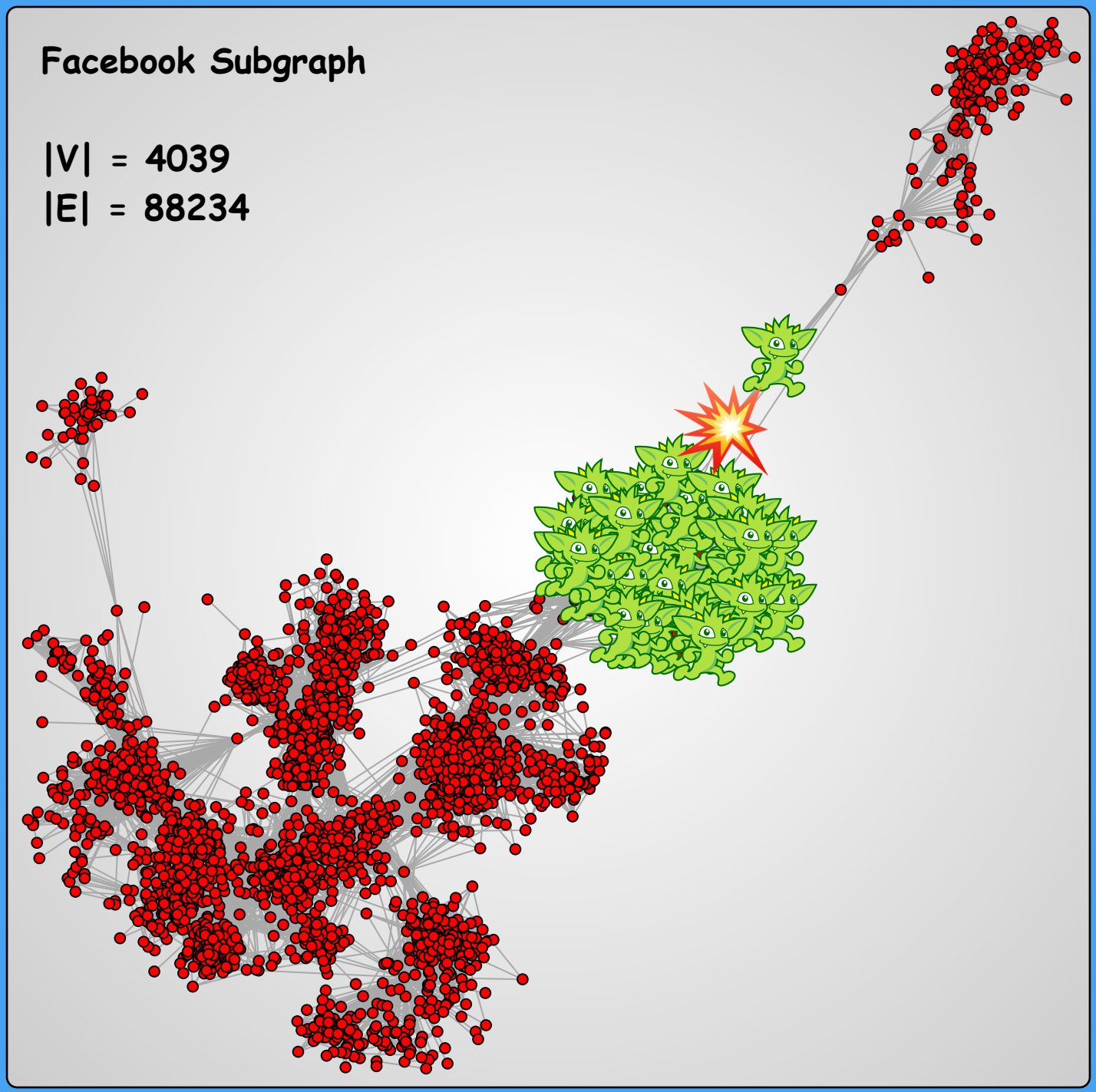
|V| = 4039
|E| = 88234



Aliev, A., Rodriguez, M.A., "Reducing Computational Complexity with Correlate Traversals," DataStax Blog, April 2017.

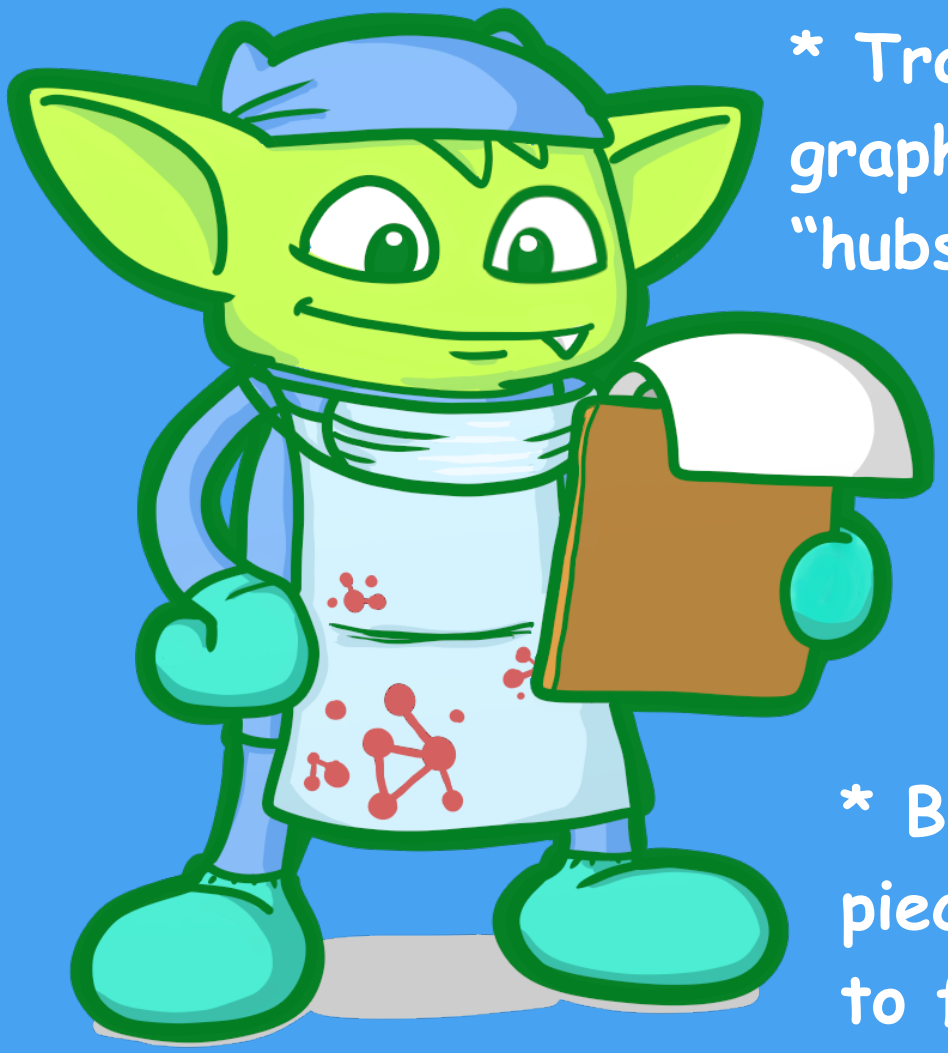
https://www.datastax.com/2017/04/reducing-computational-complexity-with-correlate-traversals


Facebook Subgraph
|V| = 4039
|E| = 88234
Yo.


Facebook Subgraph
|V| = 4039
|E| = 88234

So you want to be a graph surgeon?

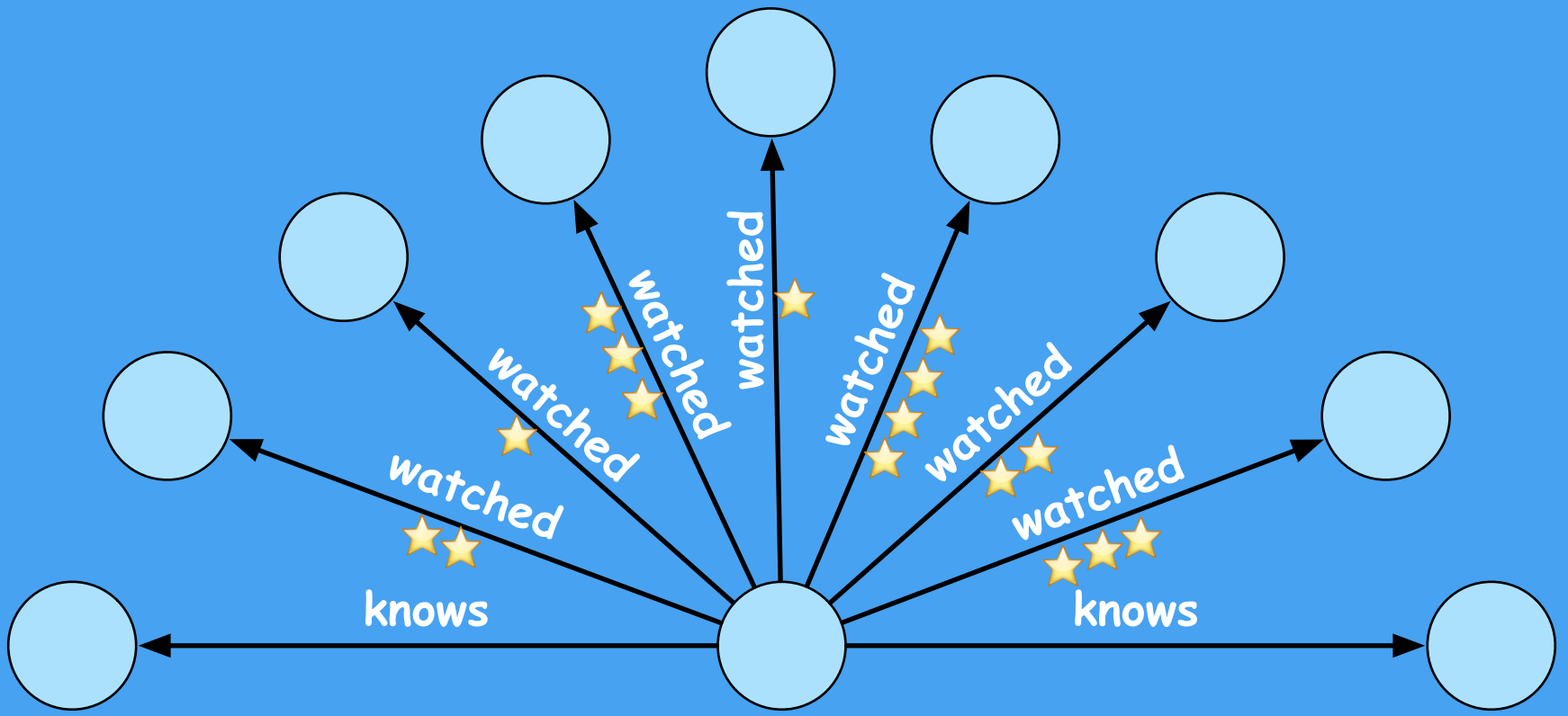
* The same traversal on different graphs can have wildly different performance characteristics.

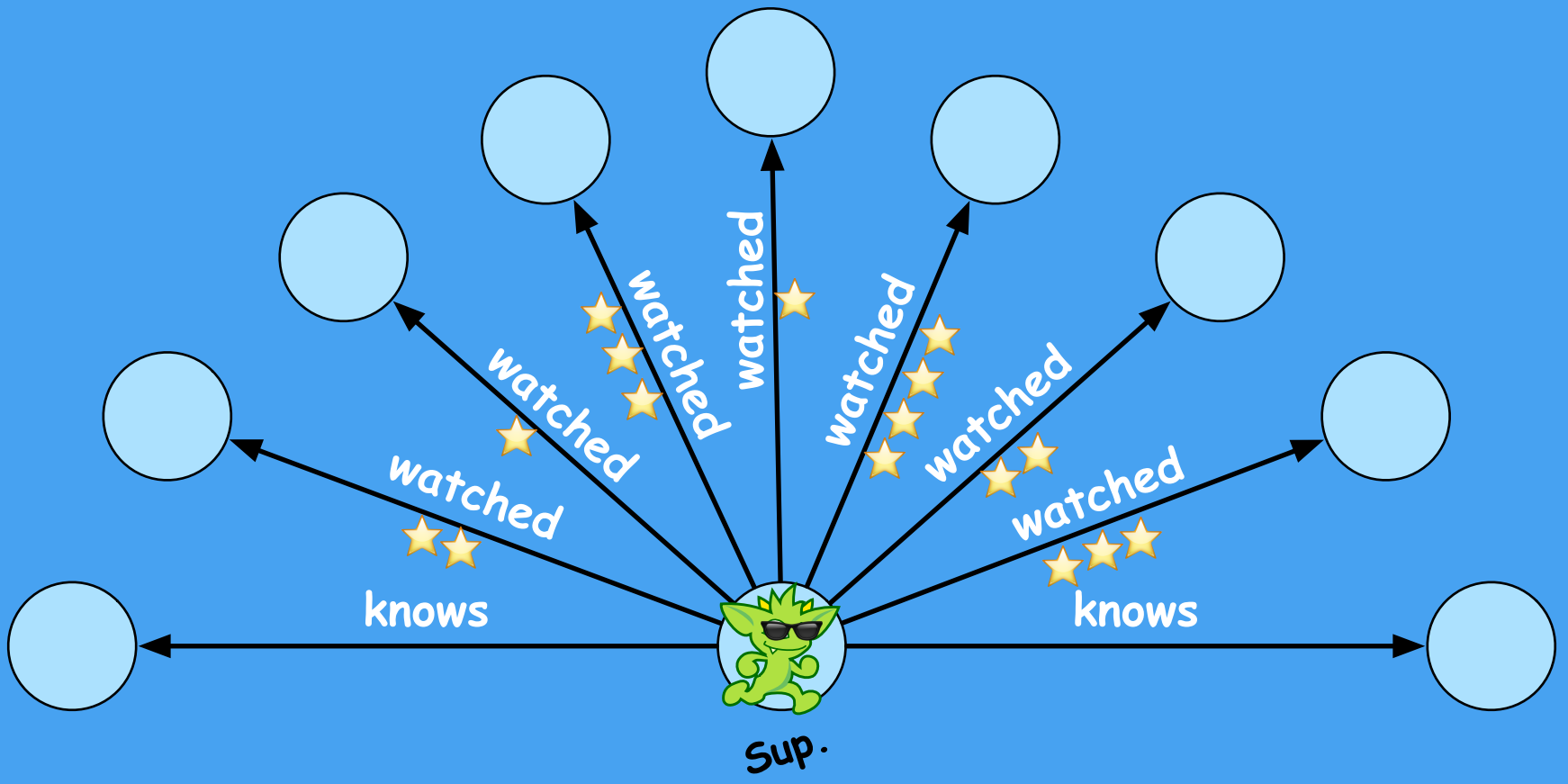


* Traversals on scale-free/natural graphs can easily explode due to "hubs" (super nodes).

* Use Gremlin to study the graph's statistics before using Gremlin to query the graph.

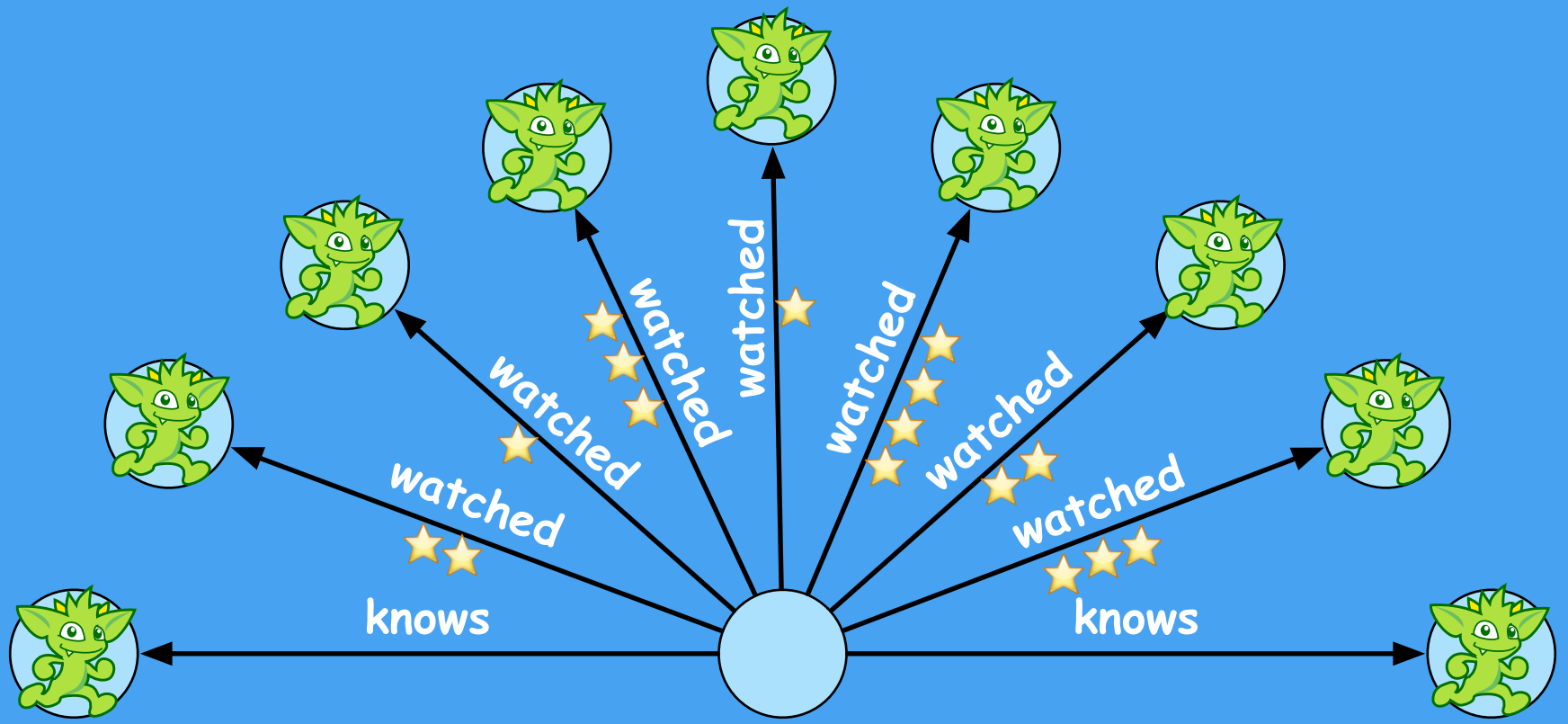
* Build your traversals up piecewise and use profile() to find bottlenecks.


watched
watched
watched
watched
watched
watched
knows
knows

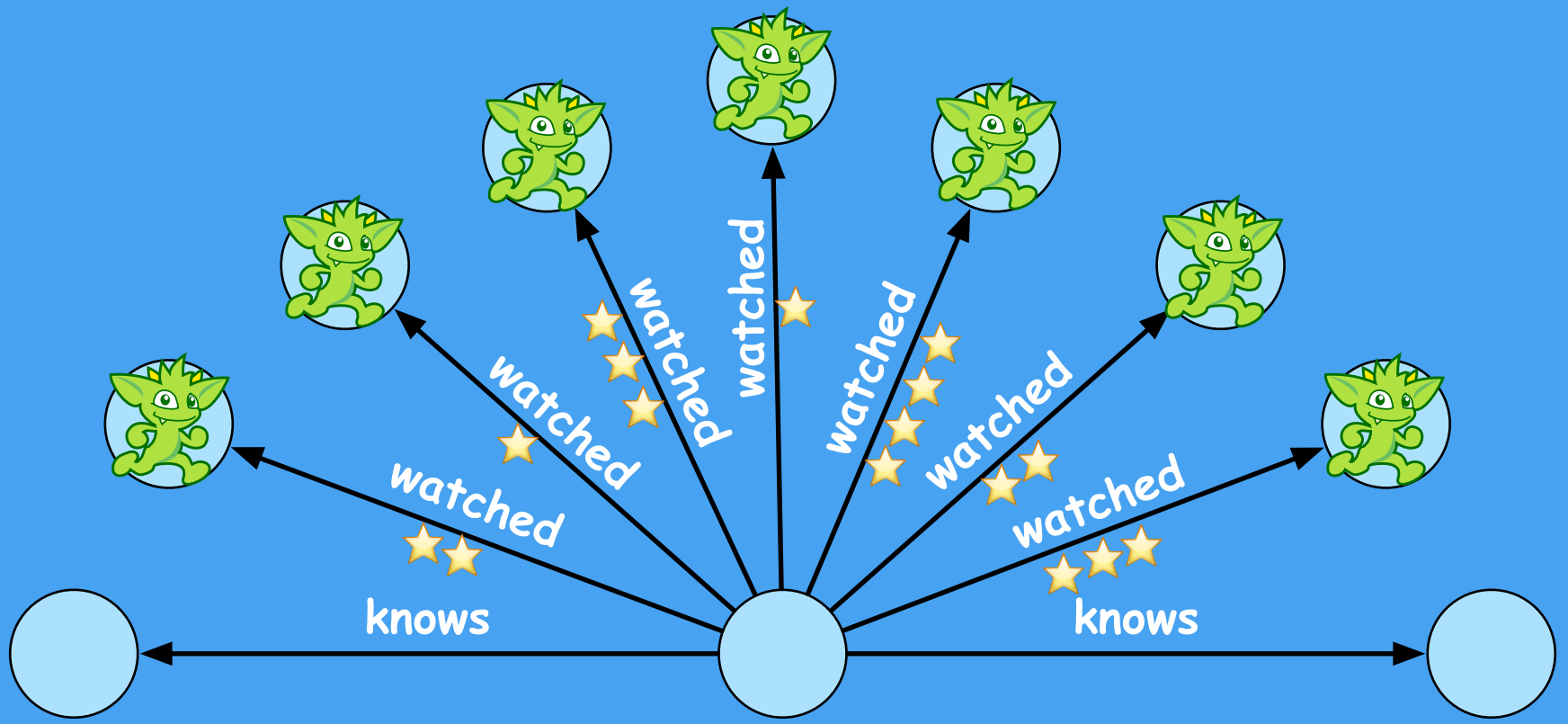

watched
watched
watched
watched
watched
watched
knows
knows
Sup.

g.V(1).count()

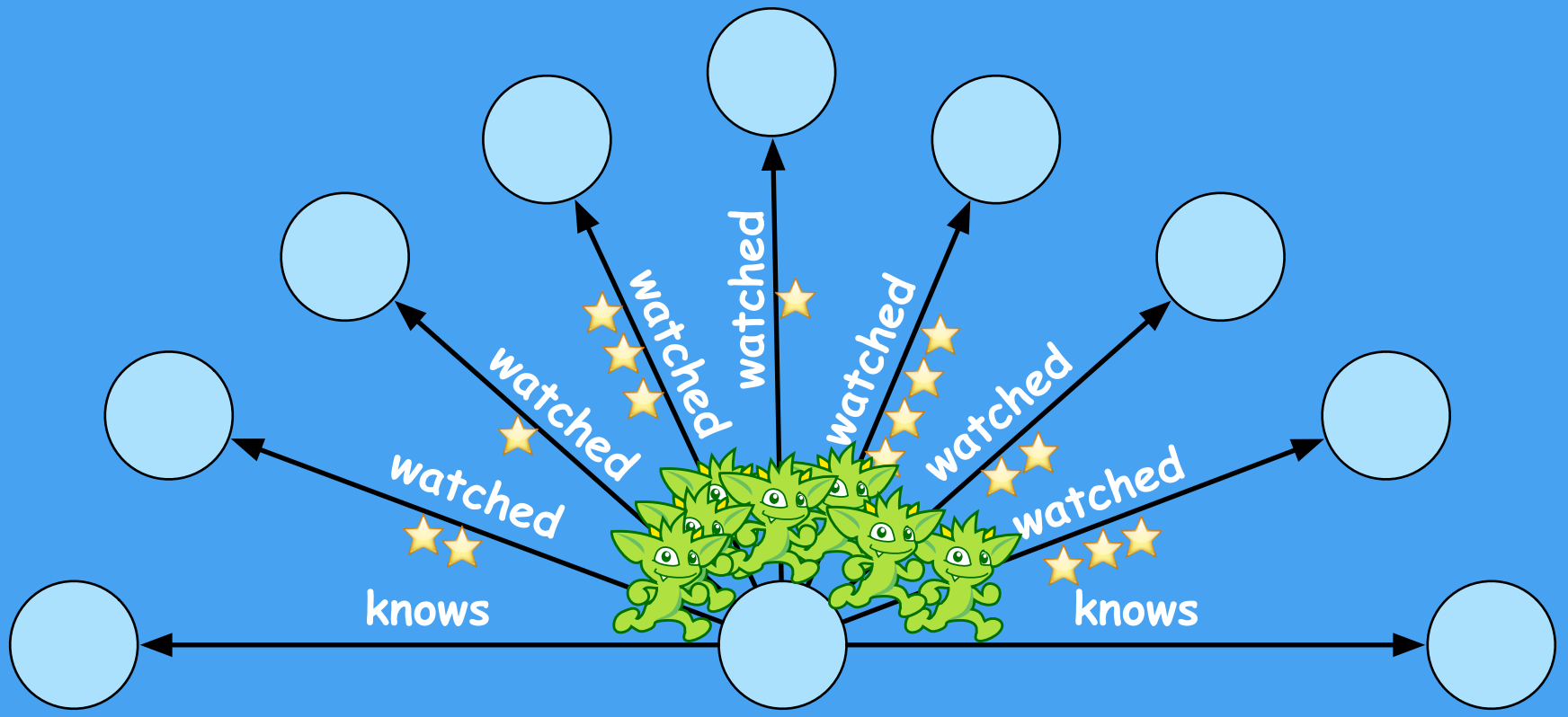
1



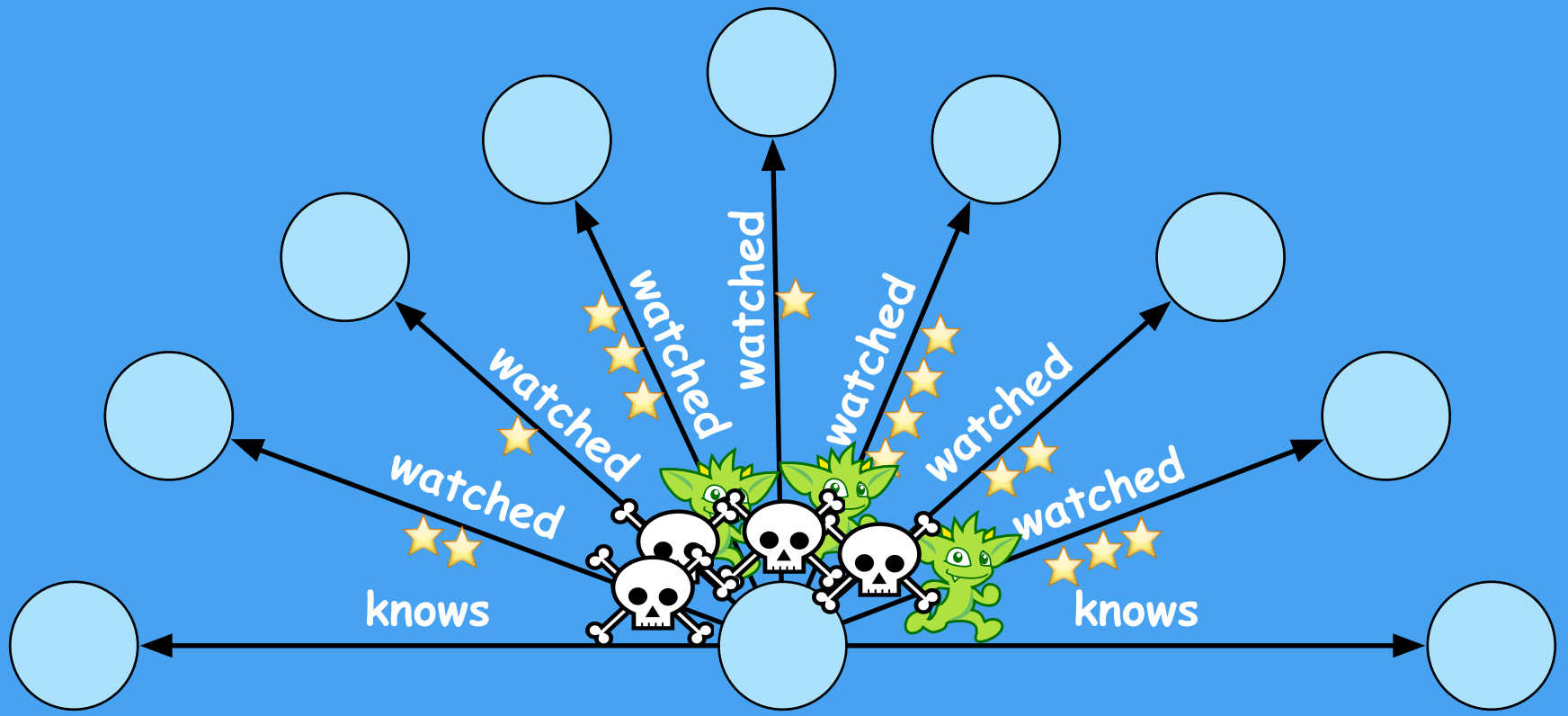
g.V(1).out().count()



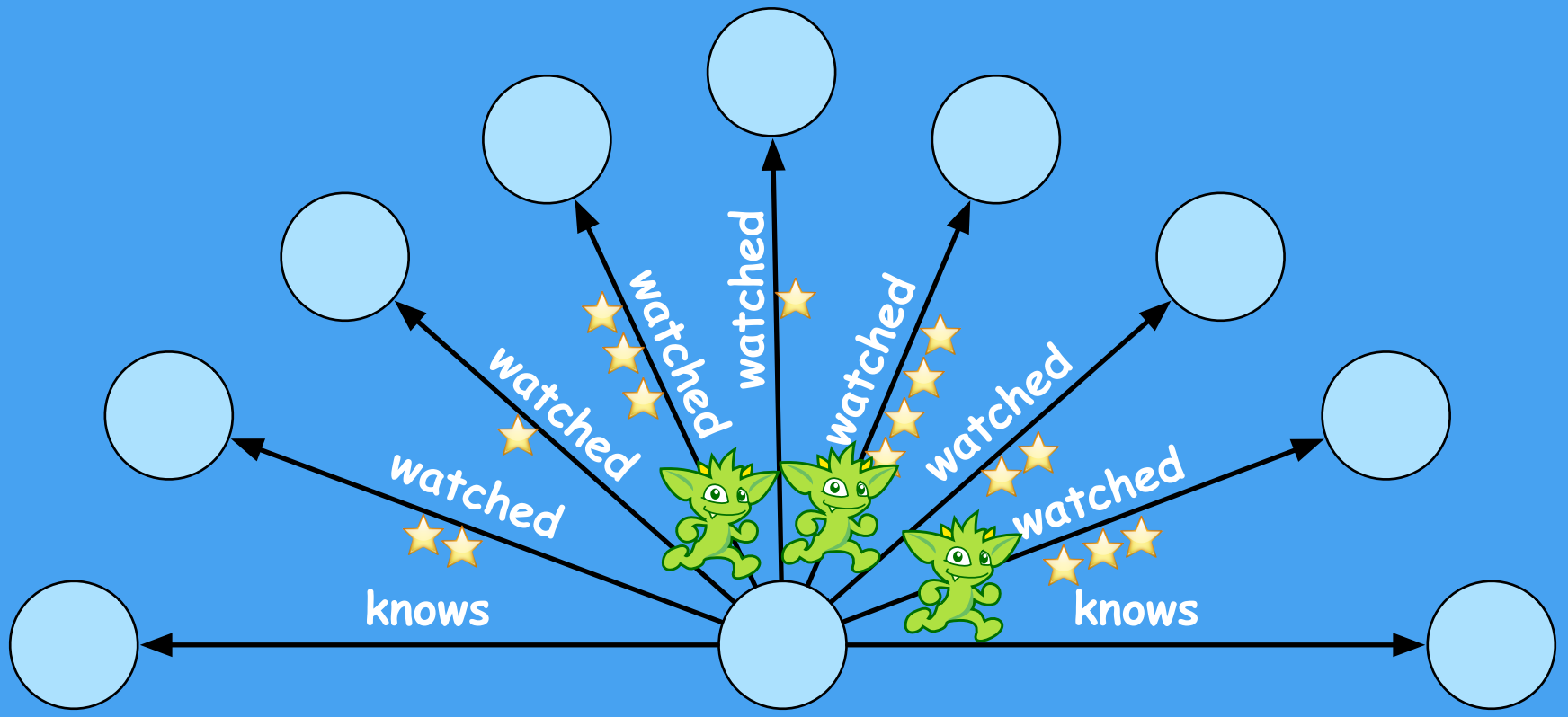
g.V(1).out('watched').count()



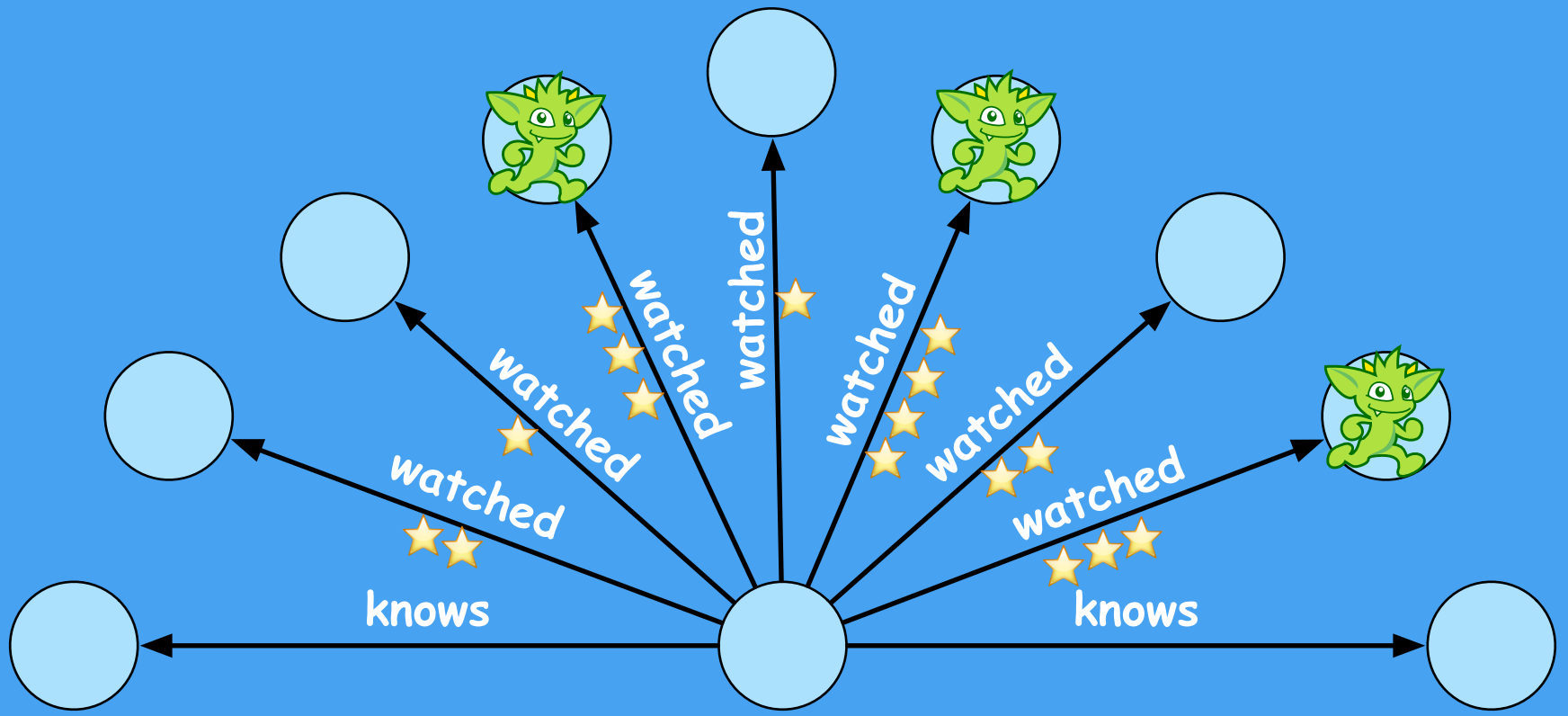
g.V(1).outE('watched')



g.V(1).outE('watched').has('stars',gte(3))

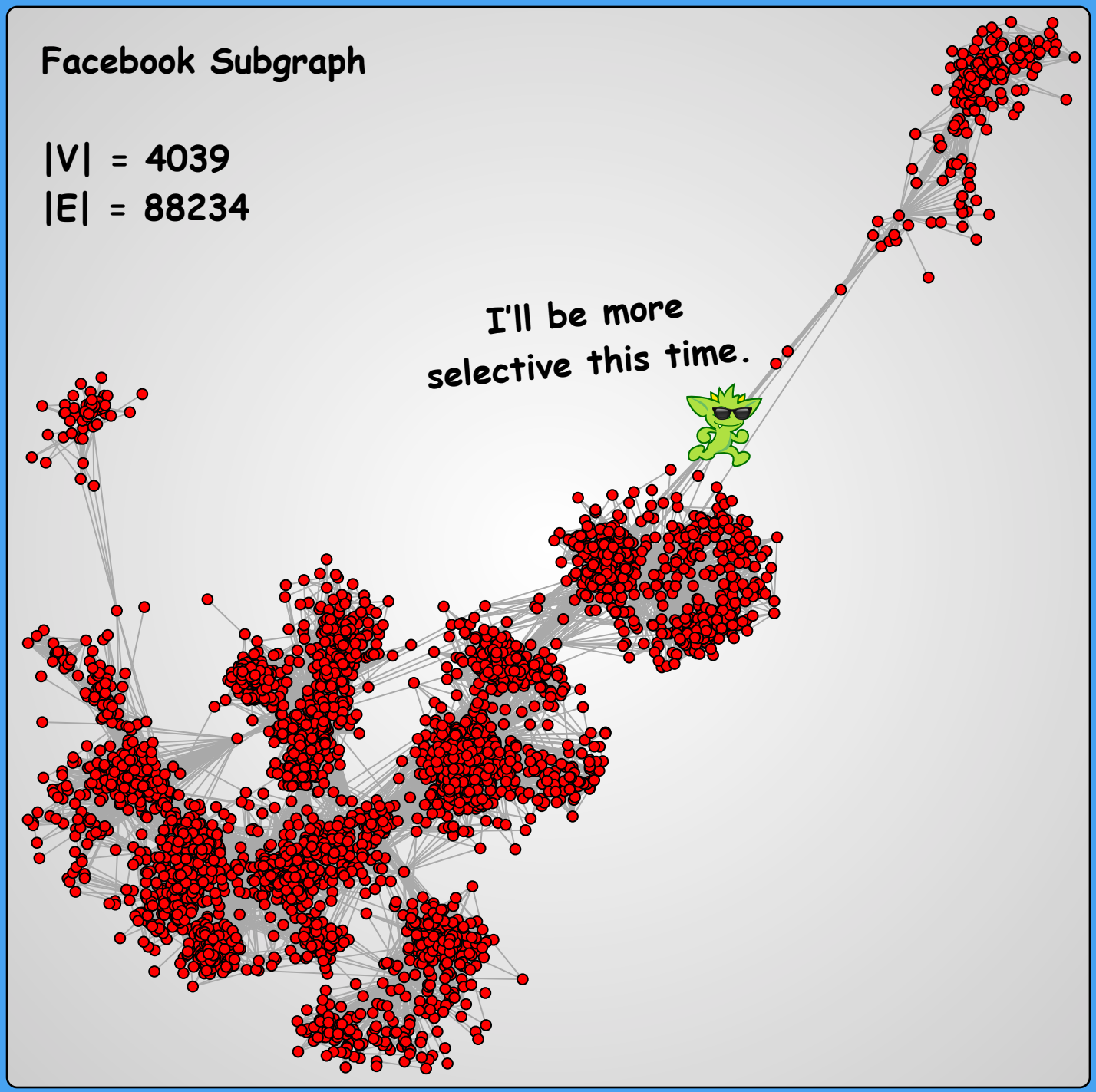


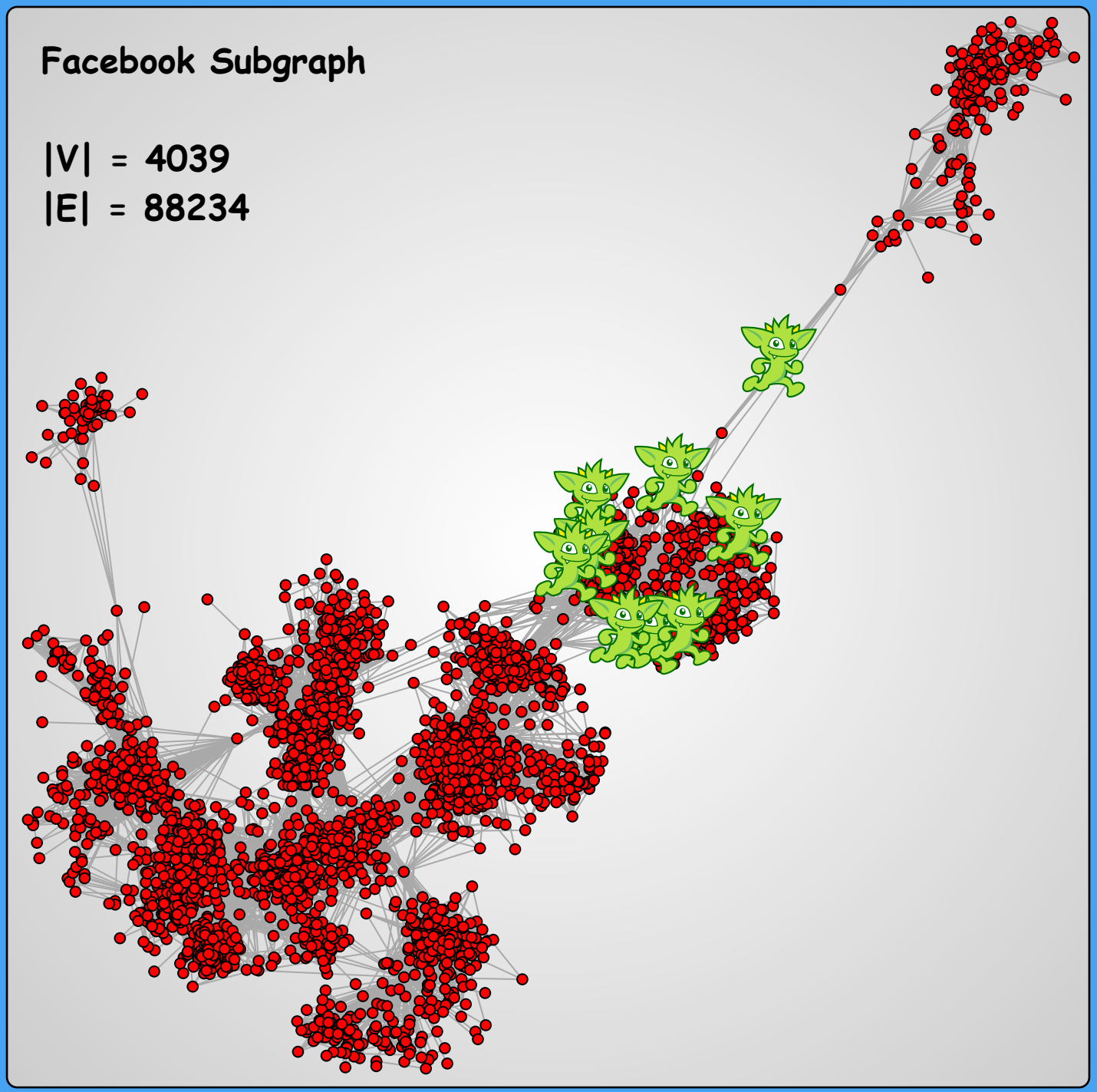
g.V(1).outE('watched').has('stars',gte(3))




```
g.V(1).outE('watched').has('stars',gte(3)).inV().count()
```

3


Facebook Subgraph
|V| = 4039
|E| = 88234
I'll be more
selective this time.


Facebook Subgraph
|V| = 4039
|E| = 88234

Quiz

Controlling Traverser Populations

out()

out('label')

outE().has('key',value).inV()

local(out().limit(10))

local(out().coin(0.5))

local(outE().sample(10).by('weight').inV())

Controlling Traverser Populations

out()

only edges in one direction

out('label')

outE().has('key',value).inV()

local(out().limit(10))

local(out().coin(0.5))

local(outE().sample(10).by('weight').inV())

Controlling Traverser Populations

out()

only edges in one direction

out('label')

only edges w/ label

outE().has('key',value).inV()

local(out().limit(10))

local(out().coin(0.5))

local(outE().sample(10).by('weight').inV())

Controlling Traverser Populations

out()

only edges in one direction

out('label')

only edges w/ label

outE().has('key',value).inV()

only edges w/ property condition

local(out().limit(10))

local(out().coin(0.5))

local(outE().sample(10).by('weight').inV())

Controlling Traverser Populations

out()

only edges in one direction

out('label')

only edges w/ label

outE().has('key',value).inV()

only edges w/ property condition

local(out().limit(10))

only the first 10 edges

local(out().coin(0.5))

local(outE().sample(10).by('weight').inV())

Controlling Traverser Populations

out()

only edges in one direction

out('label')

only edges w/ label

outE().has('key',value).inV()

only edges w/ property condition

local(out().limit(10))

only the first 10 edges

local(out().coin(0.5))

only 50% of edges

local(outE().sample(10).by('weight').inV())

Controlling Traverser Populations

out()

only edges in one direction

out('label')

only edges w/ label

outE().has('key',value).inV()

only edges w/ property condition

local(out().limit(10))

only the first 10 edges

local(out().coin(0.5))

only 50% of edges

local(outE().sample(10).by('weight').inV())

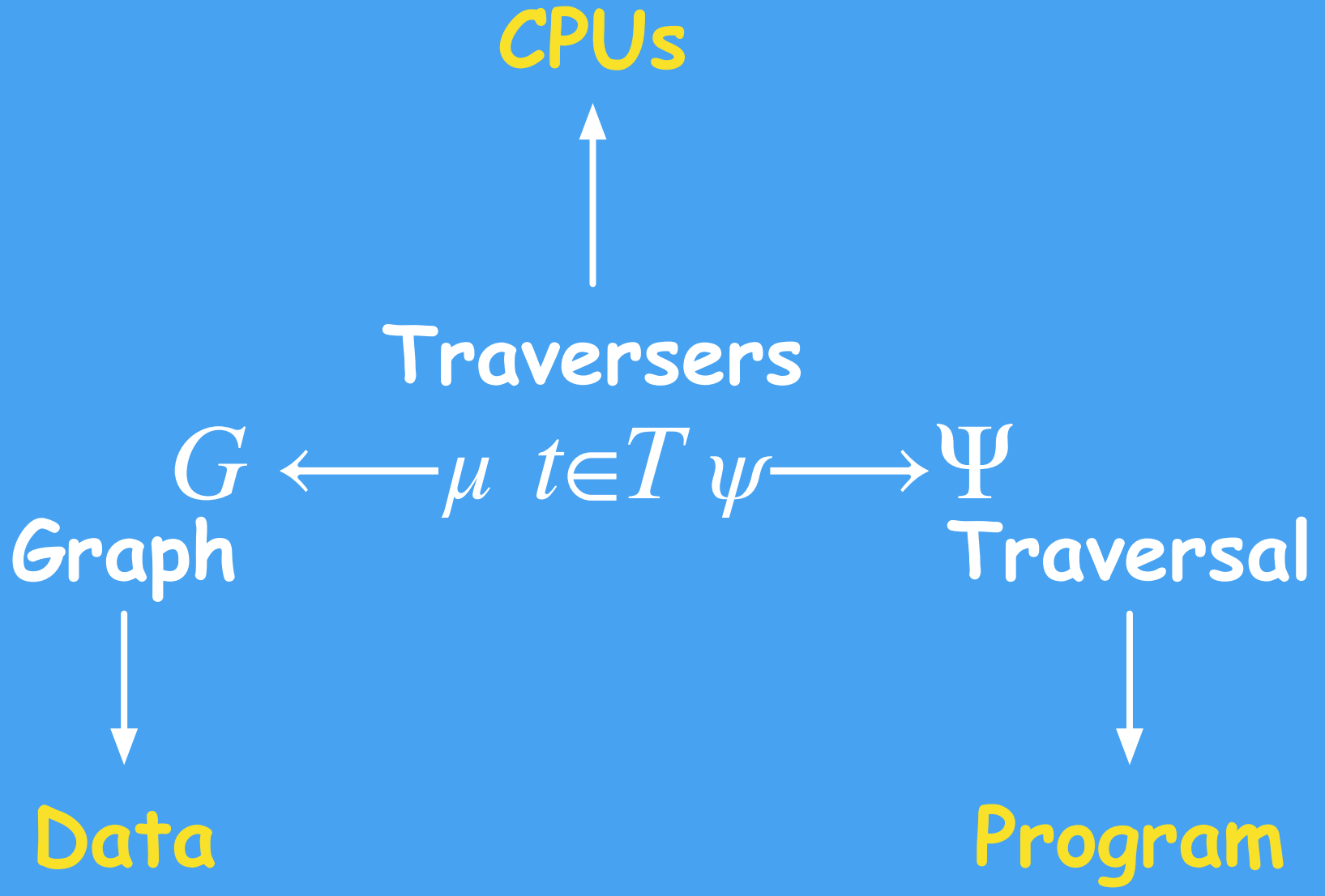
only 10 edges biased by weight

The computational complexity of the traversal is a function of the structural complexity of the graph.

Traversers

$$G \longleftarrow \mu \;\; t \in T \;\; \psi \longrightarrow \Psi$$

Graph Traversal


CPUs
Traversers
$G \longleftarrow \mu \; t \in T \; \psi \longrightarrow \Psi$
Graph
Traversal
Data
Program

The show is over.

Please tip your waitress and ~~drive~~ traverse home safe!