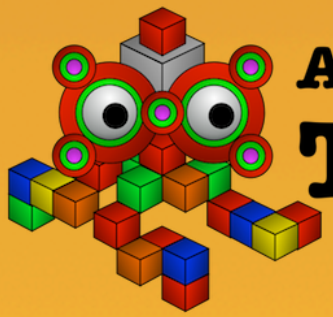
Gremlin's Graph Traversal Machinery



Apache TinkerPop

http://tinkerpop.apache.org

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

DATASTAX

CASSANDRA SUMMIT 2016

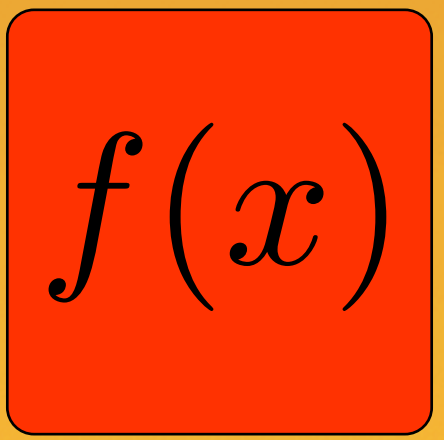
$$f : X \to X$$

The function f is a process that maps a structure of type X to a structure of type X.

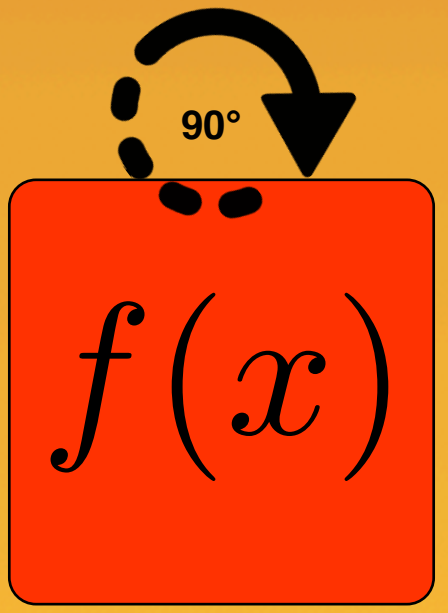
$$f(x_1) = x_2$$

$x_1 \in X$ $x_2 \in X$

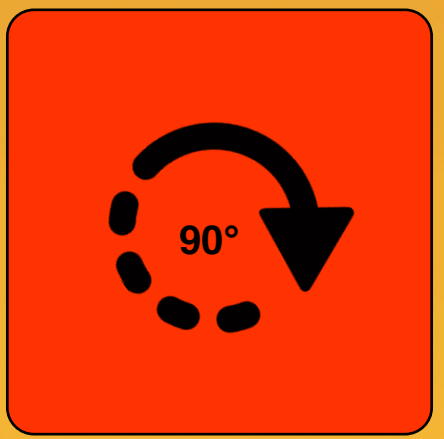
The function f maps the object x1 (from the set of X) to the object x2 (from the set of X).



A **step** is a function.

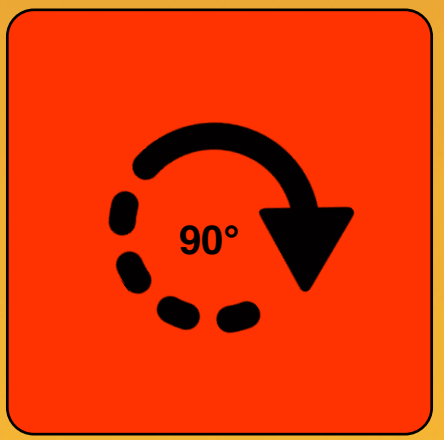


Assume that this step rotates an X by 90°.



The algorithm of the step is a “black box.”

```
class Traverser<V> {
 V value;
}
```



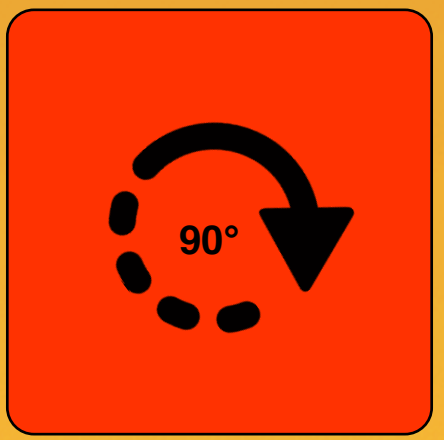

```
class Traverser<V> {
 V value;
}
```

A **traverser** wraps a value of type V.

```
class Traverser<V> {
 V value;
}
```

```
class Traverser<V> {
 V value;
}
```

`Traverser<Integer>`

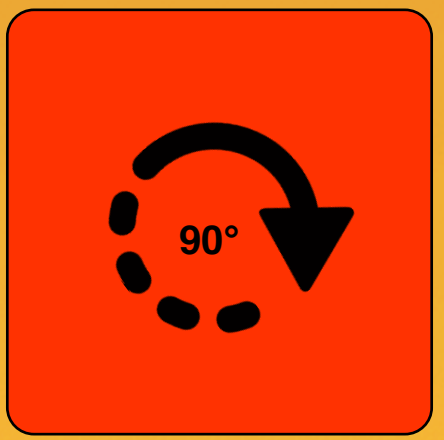


`Traverser<Integer>`

The step maps an integer traverser to an integer traverser.

```
class Traverser<V> {
 V value;
}
```

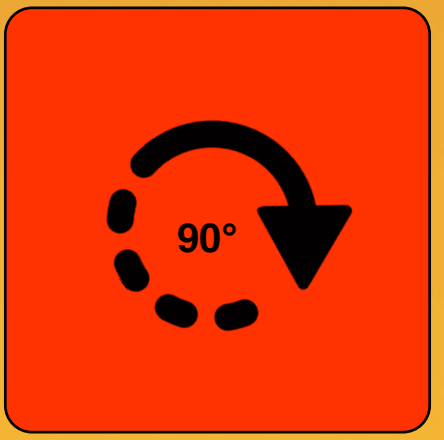
Traverser(0)



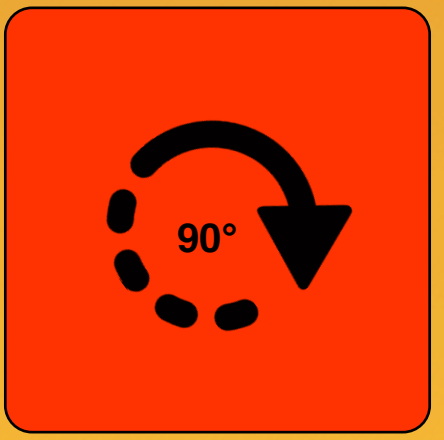

```
class Traverser<V> {
 V value;
}
```

Traverser(90)

A traverser of with a rotation of 0° becomes a traverser with a rotation of 90°.


90°

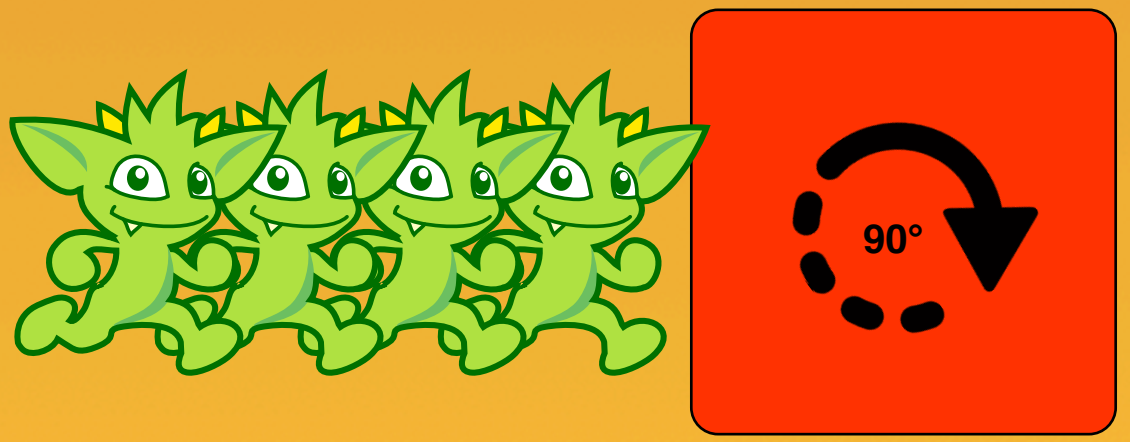
$$T[\mathbb{N}] \rightarrow T[\mathbb{N}]$$



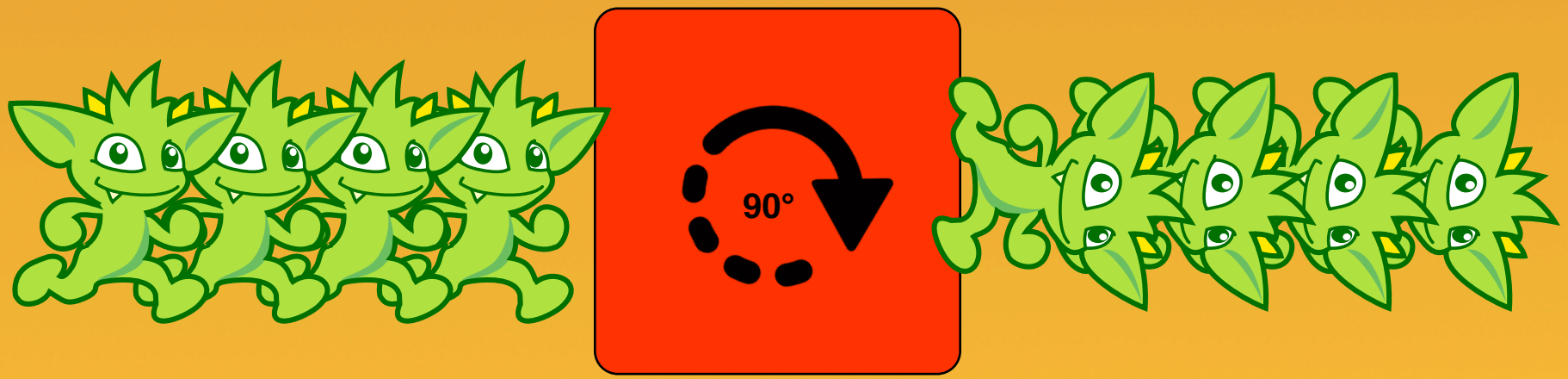

$\in T[\mathbb{N}]$


$\in T[\mathbb{N}]$

Both the input and output traversers are of type Traverser<Integer>.



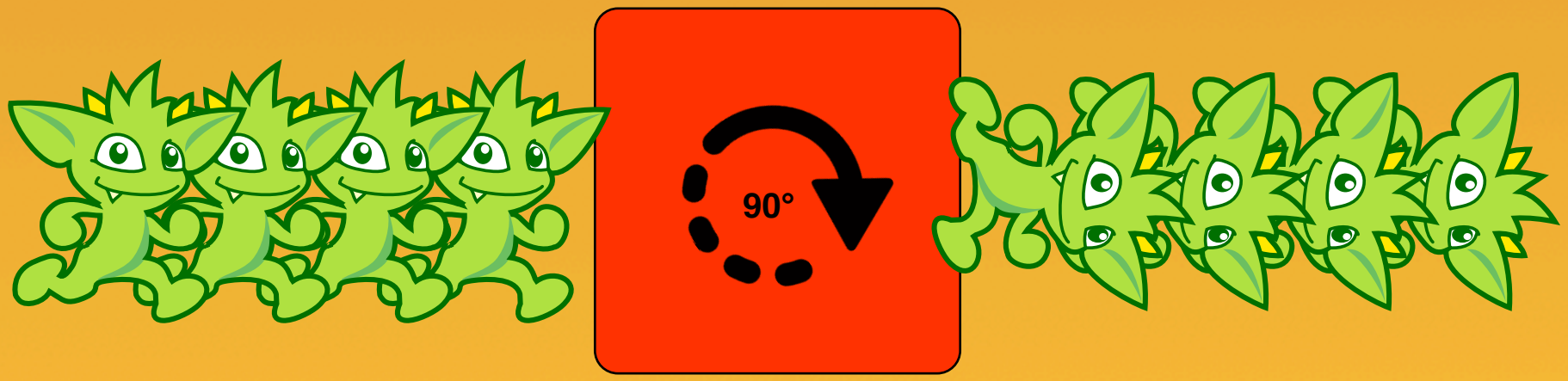
A stream of input traversers…



…yields a stream of output traversers.

```
class Traverser<V> {
 V value;
 long bulk;
}
```

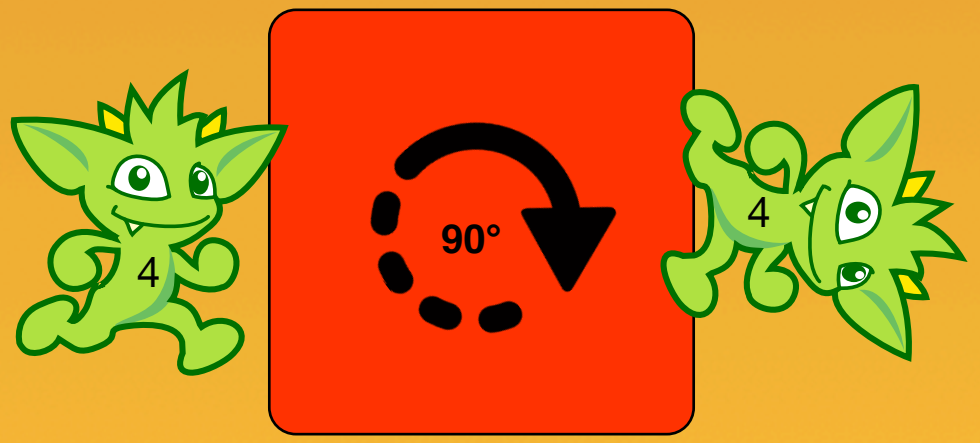
```
class Traverser<V> {
 V value;
 long bulk;
}
```



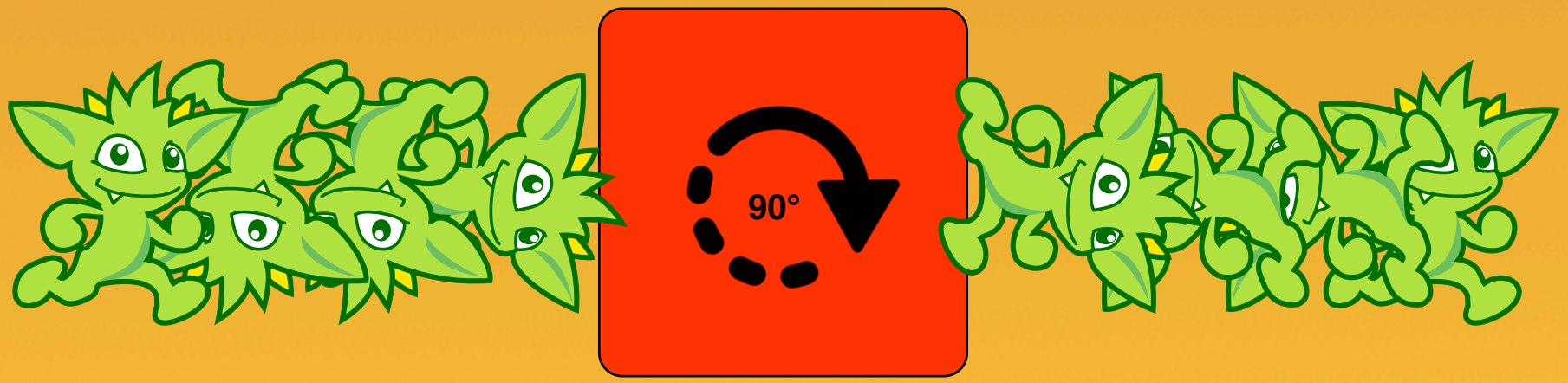
A traverser can have a **bulk** which denotes how many V values it represents.

```
class Traverser<V> {
 V value;
 long bulk;
}
```

```
class Traverser<V> {
 V value;
 long bulk;
}
```



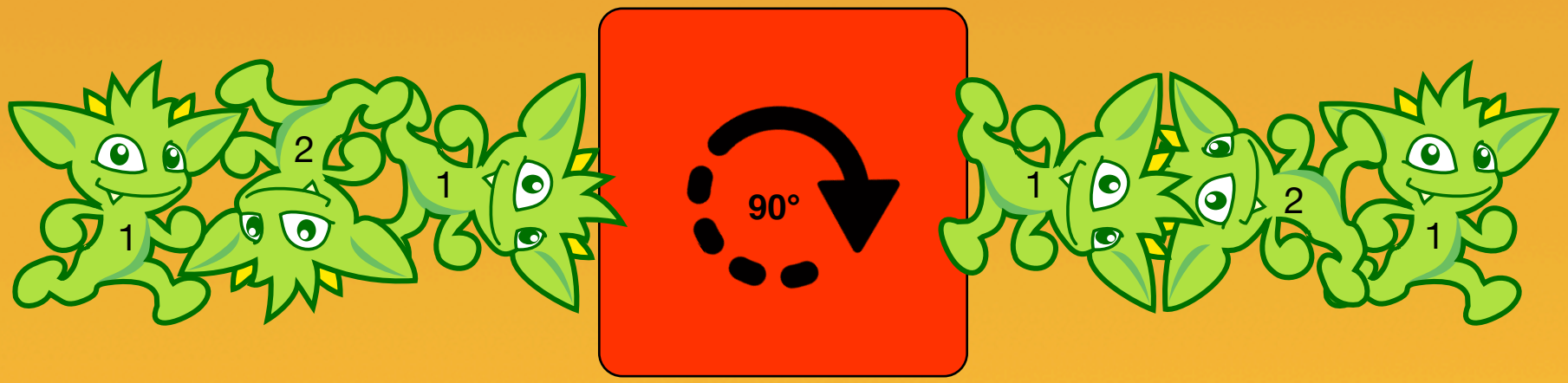
Bulking groups identical traversers to reduce the number of evaluations of a step.



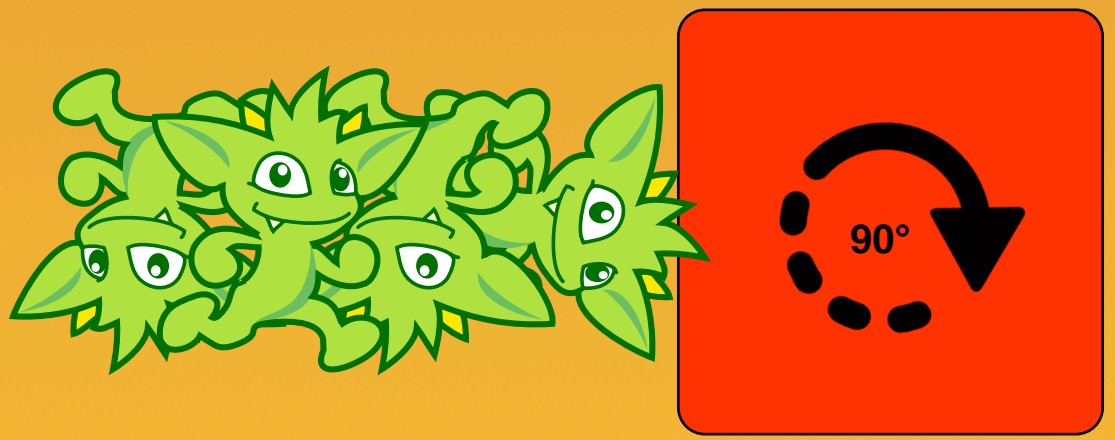
A variegated stream of input traversers yields a variegated stream of output traversers.

```
class Traverser<V> {
 V value;
 long bulk;
}
```

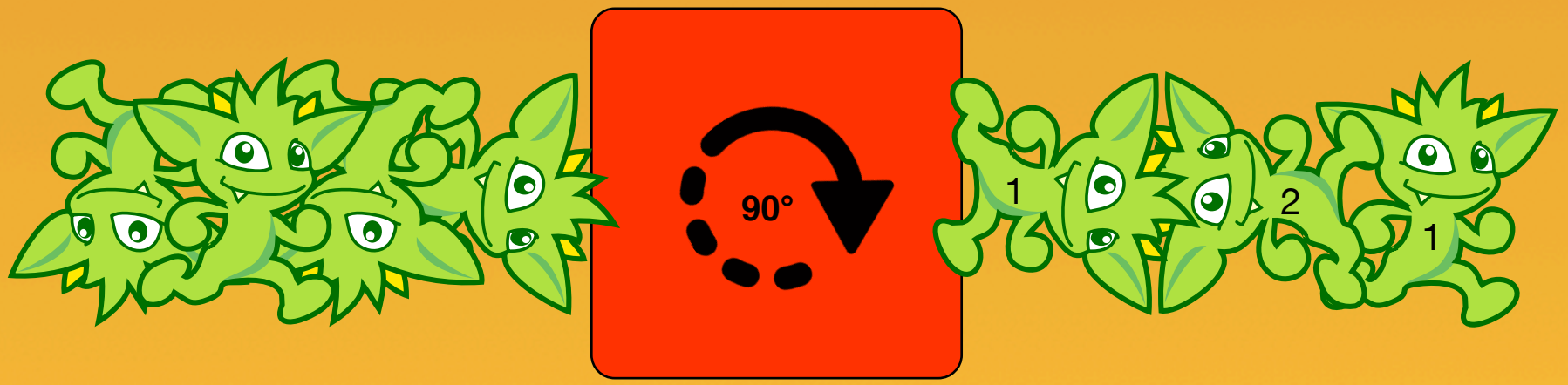
```
class Traverser<V> {
 V value;
 long bulk;
}
```



Bulking can reduce the size of the stream.



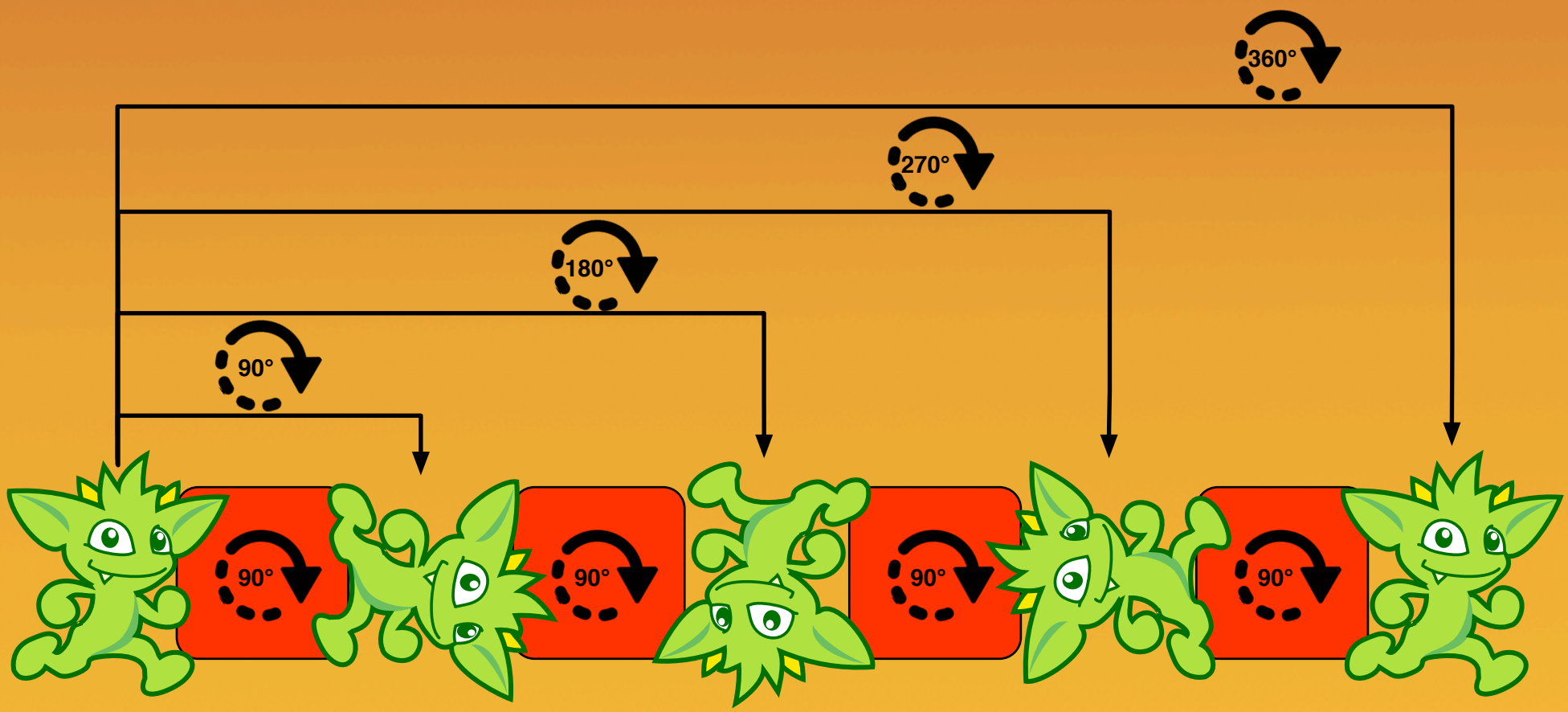
If the order of the stream does not matter…



total bulk = 4
total count = 4

total bulk = 4
total count = 3

…then reordering can increase the likelihood of bulking.



A **traversal** is a list of one or more steps.

$$f : X \to Y$$

$$h : Y \to Z$$

Different functions can yield different mappings between different domains and ranges.

$$Y\ y = f(x)$$

$$Z\ z = h(y)$$

The output of f can be the input to h because the range of f is the domain of h (i.e. Y).

$$y = f(x)$$

$$z = h(y)$$

$$y = f(x)$$

$$z = h(y)$$

$$f(x) = y$$

$$h(y) = z$$

$$f(x) = y$$

$$h(y) = z$$

$$f(x) \not{h}(y) = z$$

$$f(x)\ h \qquad = z$$

$$x \quad f \quad h \quad = z$$

$$x \cdot f \cdot h = z$$

readable

$$x \cdot f \cdot h = z$$

$$\equiv$$

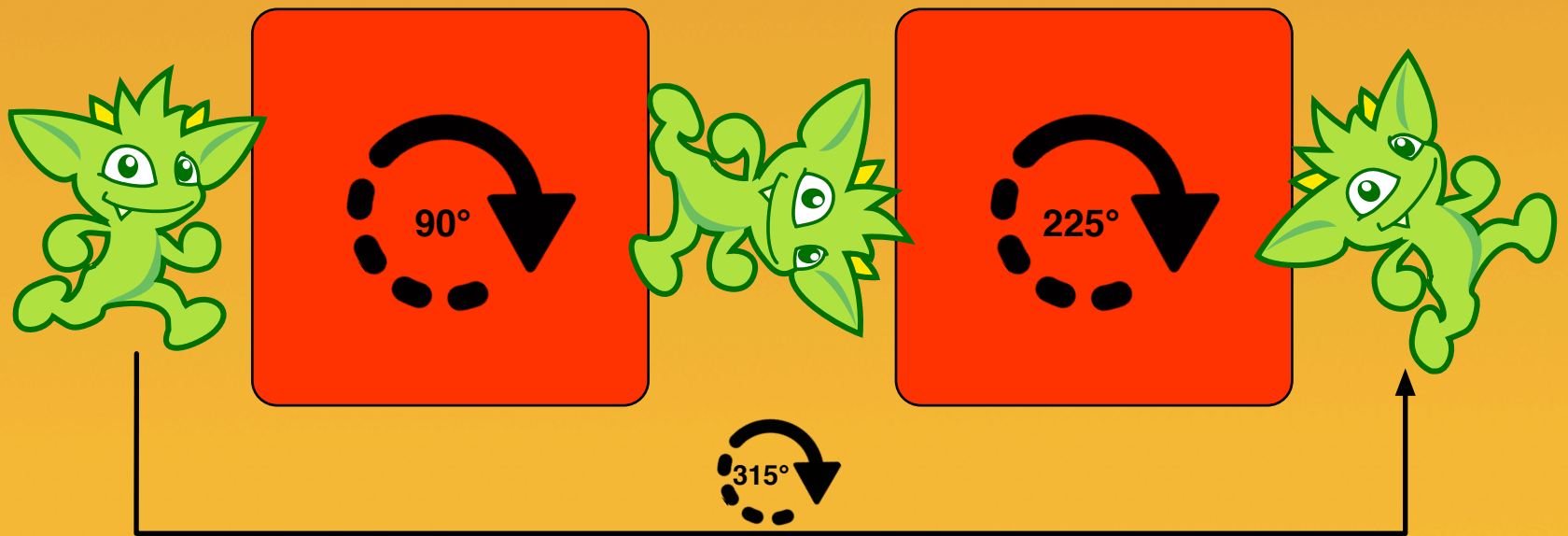
unreadable

$$h(f(x)) = z$$

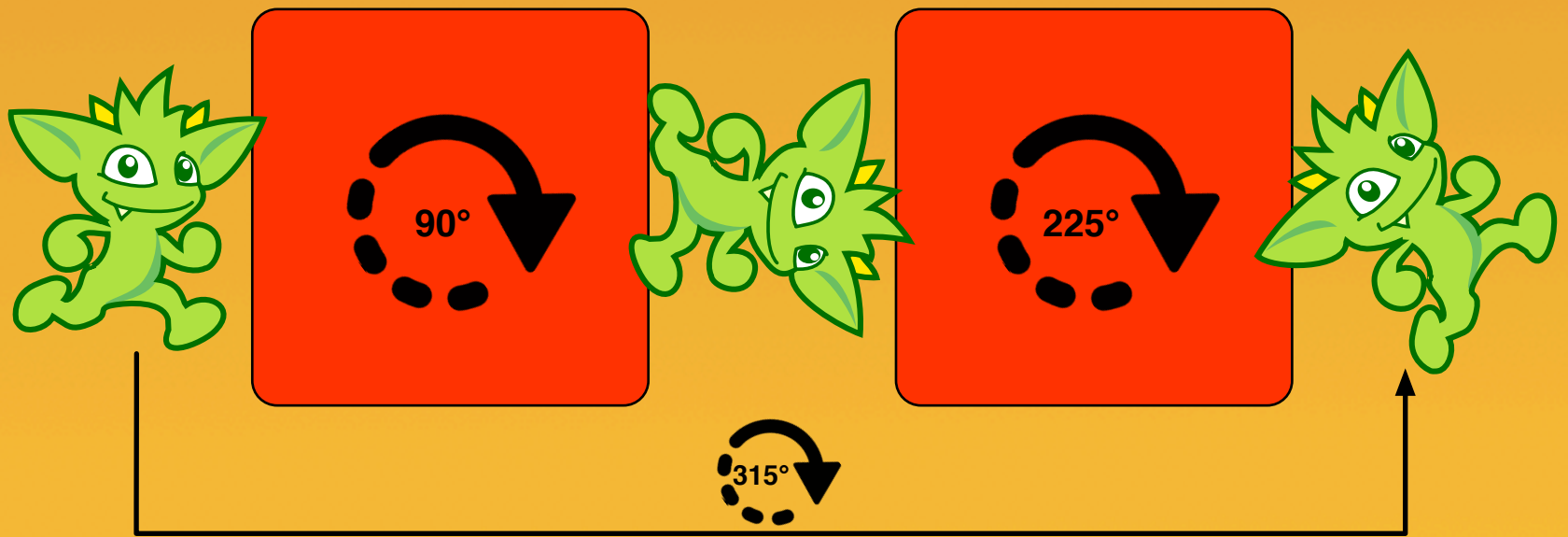
$$z = x \cdot f \cdot h$$

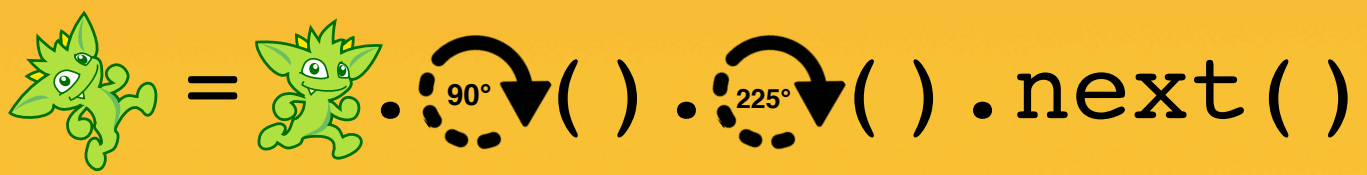
$$z = x \cdot f \cdot h$$

```
z = x.f().h().next()
```

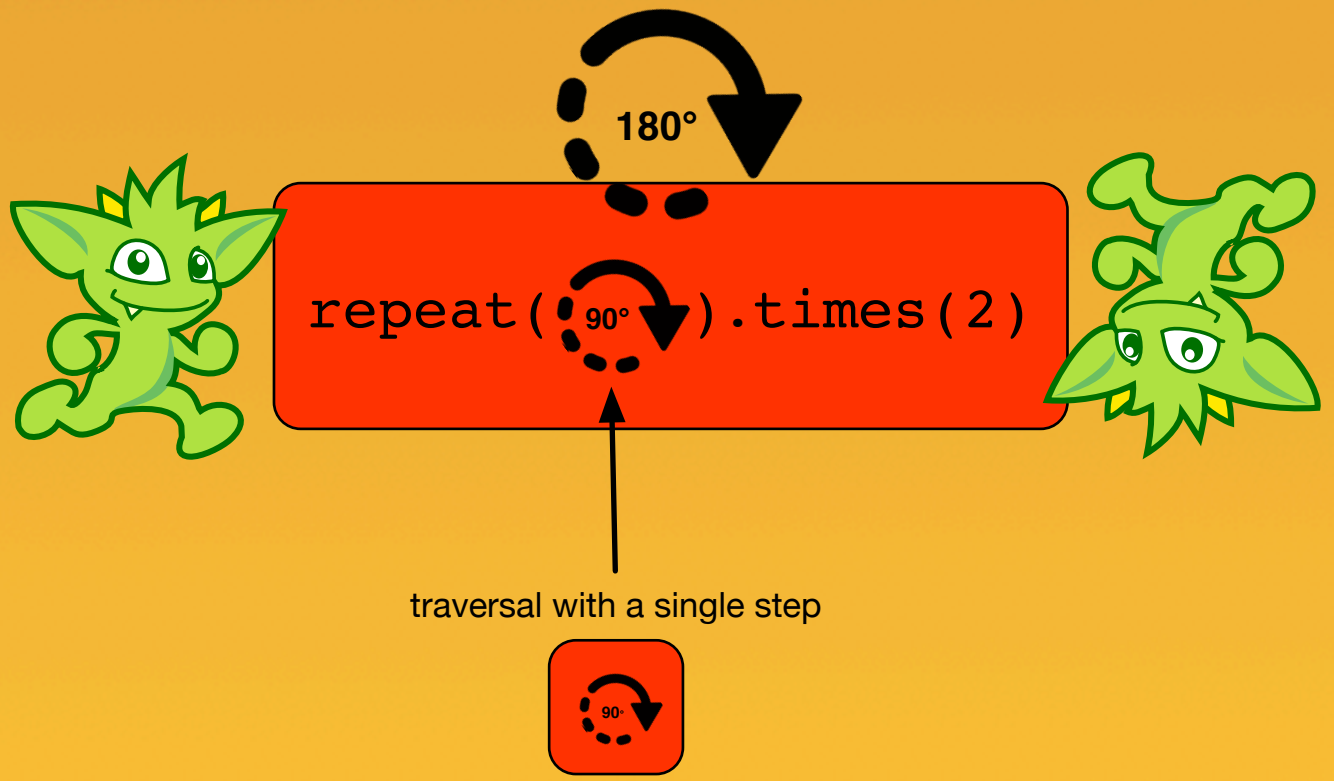


Different types of steps exist and their various compositions yield query expressivity.





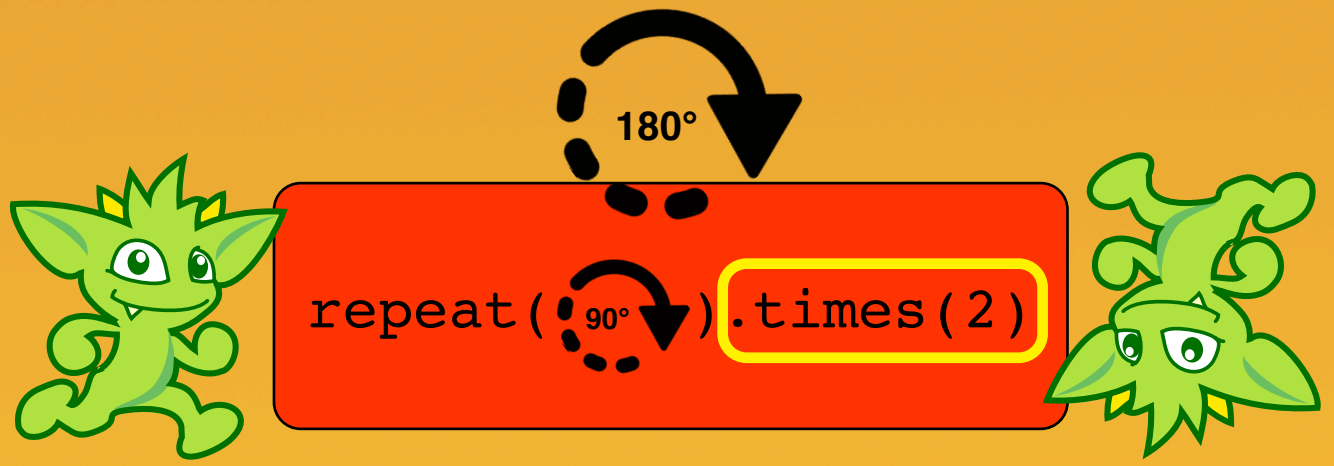
Steps process traverser streams/flows and their composition is a traversal/iterator.



Anonymous traversals can serve as step arguments.

…select("a","b").by("name")

…groupCount().by(out().count())

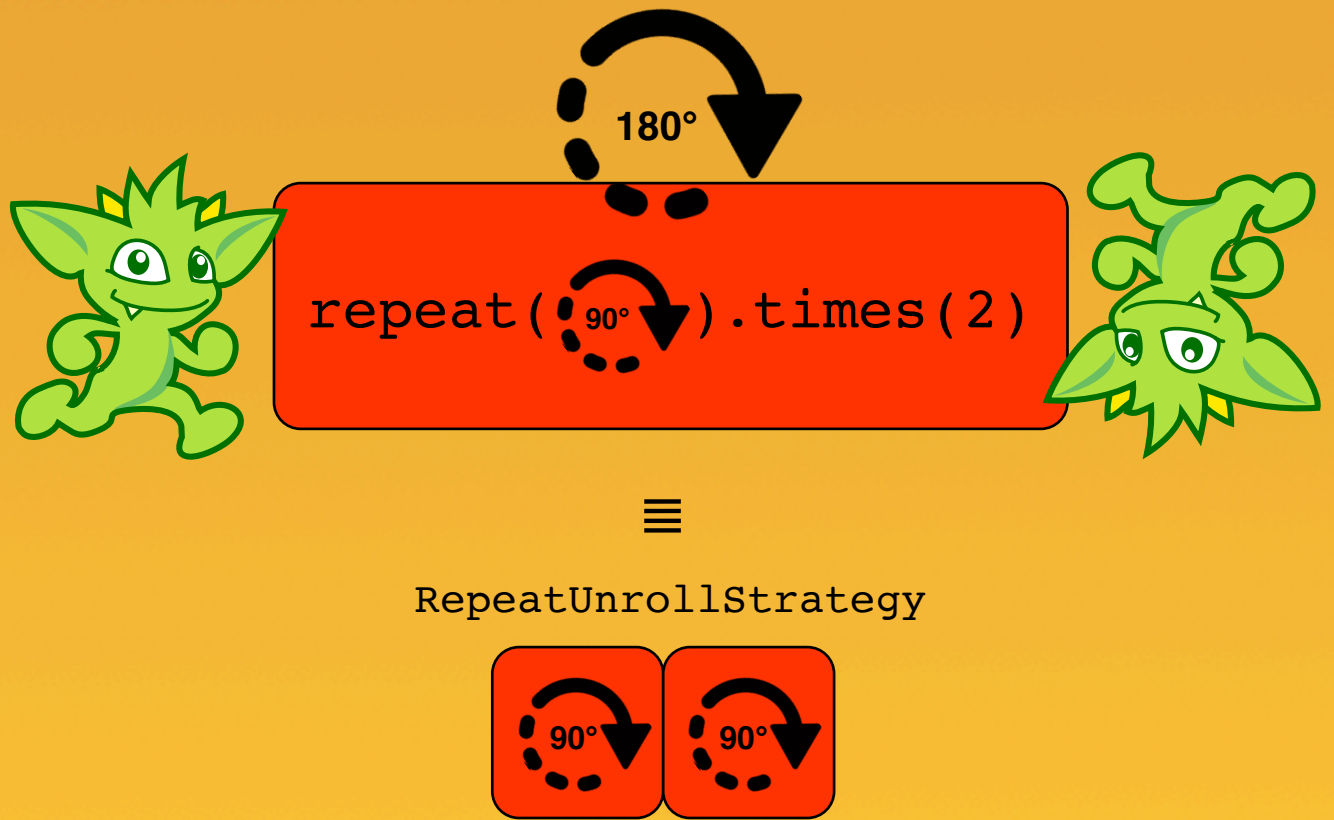


…addE("knows").from("a").to("b")

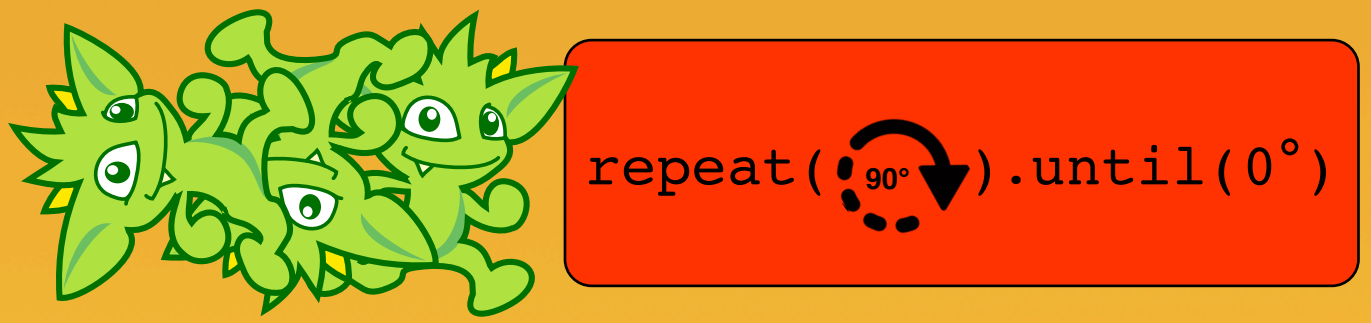
…order().by("age",decr)

Some functions serve as **step-modulators** instead of steps.

```
interface TraversalStrategy {
 void apply(Traversal traversal);
 Set<TraversalStrategy> applyPrior();
 Set<TraversalStrategy> applyPost();
}
```



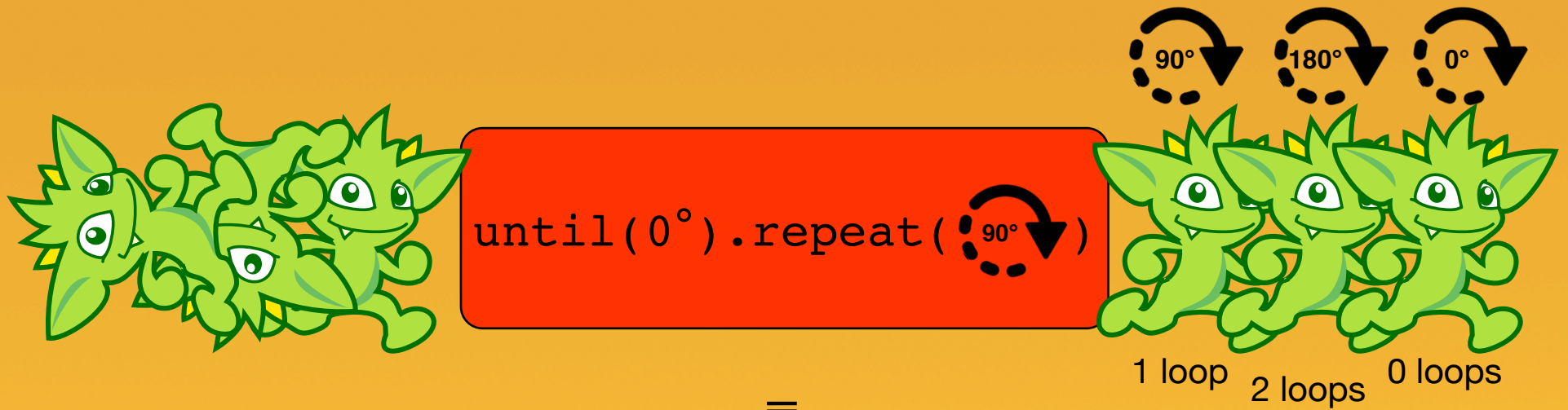
During optimization, **traversal strategies** may rewrite a traversal to a more efficient, semantically equivalent form.



Continuously process a traverser until some condition is met.

≡

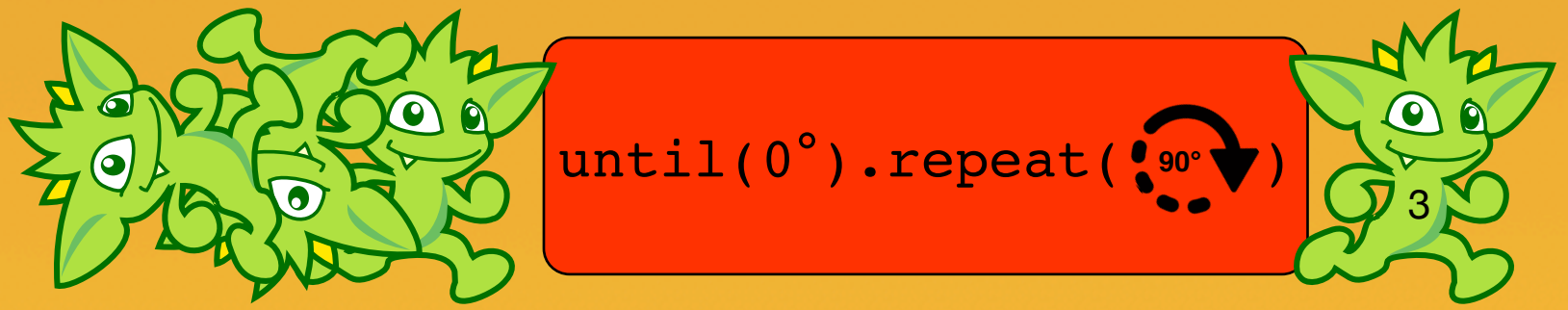
repeat().until() provides do/while-semantics.



≡

while(x !=) do 

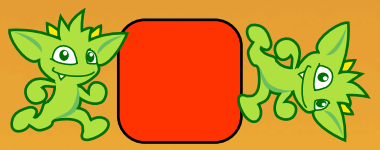
until().repeat() provides while/do-semantics.



Even if the traversers in the input stream can not be bulked, the output traversers may be able to be.

`map(x)`

one-to-one



`flatMap(x)`

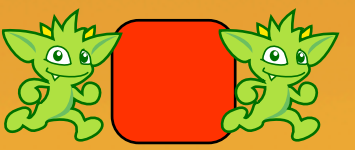
one-to-many

`filter(x)`

one-to-(one-or-none)

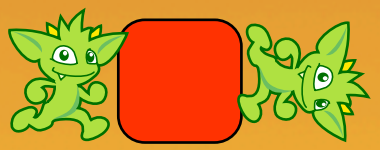
`sideEffect(x)`

one-to-same



map(x)

one-to-one



```
select(“a”,”b”)
values(“age”)
path()
id()
order()
label()
mean()
sum()
count()
groupCount()
```

flatMap(x)

one-to-many

```
out(“knows”)
in(“created”)
values(“name”)
match(x,y,z)
properties()
outE(“knows”)
V()
```

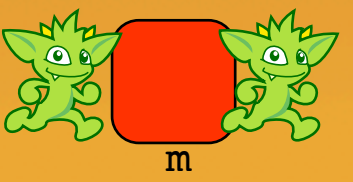
filter(x)

one-to-(one-or-none)

```
has(“name”,”gremlin”)
where(“a”,eq(“b”))
and(x,y)
or(x,y)
coin(0.5)
sample(10)
simplePath()
dedup()
```

sideEffect(x)

one-to-same




```
groupCount(“m”)
store(“m”)
tree(“m”)
subgraph(“m”)
group(“m”)
```

many-to-one (reducers)

* A collection of examples. Not the full step library.

`map(x)` one-to-one

`flatMap(x)` one-to-many

`filter(x)` one-to-(one-or-none)

`sideEffect(x)` one-to-same

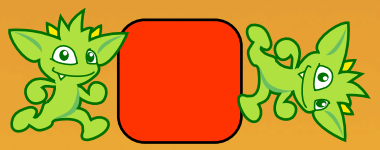

```
out("knows")
                    has("name","gremlin")
values("age")
path()



label()
                                                group("m")

          outE("knows")
                              simplePath()
               V()
groupCount()
```

map(x) — one-to-one

flatMap(x) — one-to-many

filter(x) — one-to-(one-or-none)

sideEffect(x) — one-to-same

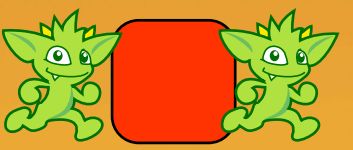


$T[V] \rightarrow T[V]^*$

`out("knows")`

$T[V \cup E] \rightarrow \emptyset \cup T[V \cup E]$

`has("name","gremlin")`



`m`

$T[V \cup E] \rightarrow T[\mathbb{N}^+]$

`values("age")`

`path()`

$T[?] \rightarrow T[\text{path}]$

`label()`

$T[V \cup E] \rightarrow T[\text{string}]$

$T[V] \rightarrow T[E]^*$

`outE("knows")`

$T[?] \rightarrow T[?]$

`group("m")`

$T[?] \rightarrow \emptyset \cup T[?]$

`simplePath()`

$T[G] \rightarrow T[V]^*$

`V()`

$T[?] \rightarrow T[\text{map}[?, \mathbb{N}^+]]$

`groupCount()`

```
g.V().has("name","gremlin").
  out("knows").values("age").
  groupCount()
```

$T[V] \rightarrow T[V]^*$
`out("knows")`

$T[V \cup E] \rightarrow \emptyset \cup T[V \cup E]$
`has("name","gremlin")`

$T[V \cup E] \rightarrow T[\mathbb{N}^+]$
`values("age")`

$T[G] \rightarrow T[V]^*$
`V()`

$T[?] \rightarrow T[\text{map}[?, \mathbb{N}^+]]$
`groupCount()`

```
g.V().has("name","gremlin").
  out("knows").values("age").
  groupCount()
```

`V()`	$T[G] \rightarrow T[V]^*$
`has("name","gremlin")`	$T[V \cup E] \rightarrow \emptyset \cup T[V \cup E]$
`out("knows")`	$T[V] \rightarrow T[V]^*$
`values("age")`	$T[V \cup E] \rightarrow T[\mathbb{N}^+]$
`groupCount()`	$T[?] \rightarrow T[\text{map}[?, \mathbb{N}^+]]$

Steps can be composed if their respective domains and ranges match.

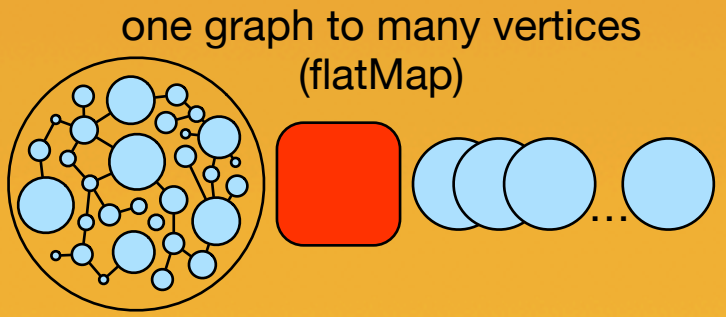
```
g.V().has("name","gremlin").
  out("knows").values("age").
  groupCount()
```

`V()`	$T[G] \rightarrow T[V]^*$
`has("name","gremlin")`	$T[V] \rightarrow \emptyset \cup T[V]$
`out("knows")`	$T[V] \rightarrow T[V]^*$
`values("age")`	$T[V] \rightarrow T[\mathbb{N}^+]$
`groupCount()`	$T[\mathbb{N}^+] \rightarrow T[\text{map}[\mathbb{N}^+, \mathbb{N}^+]]$

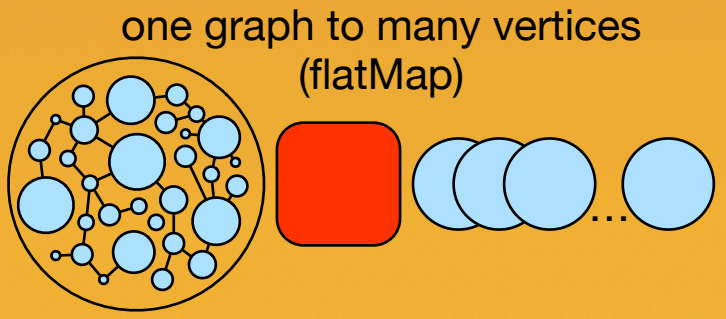
```
g.V().has(“name”,”gremlin”).
  out(“knows”).values(“age”).
  groupCount()
```

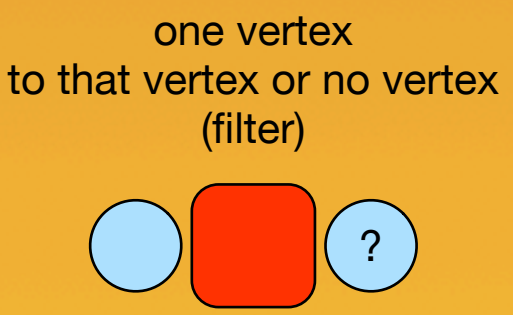
What is the distribution of ages of the people that Gremlin knows?

```
g.V().has(“name”,”gremlin”).
   out(“knows”).values(“age”).
   groupCount()
```

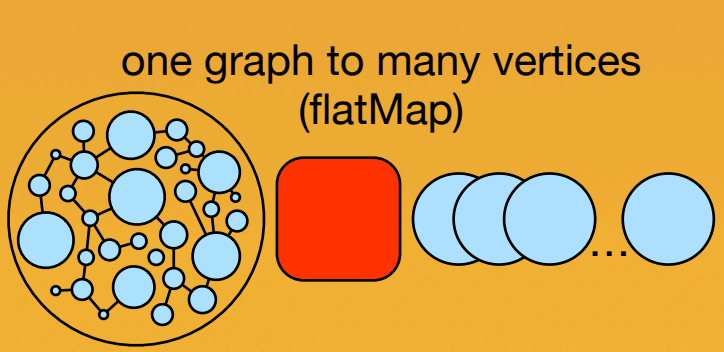


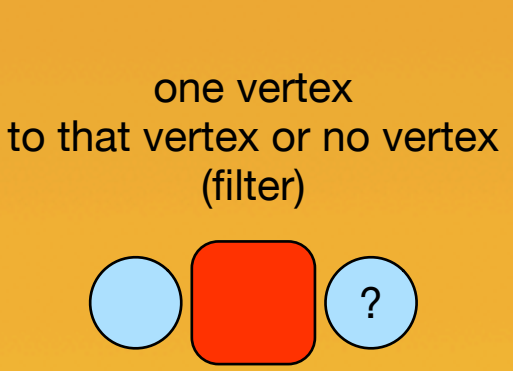
```
g.V().has(“name”,”gremlin”).
  out(“knows”).values(“age”).
  groupCount()
```

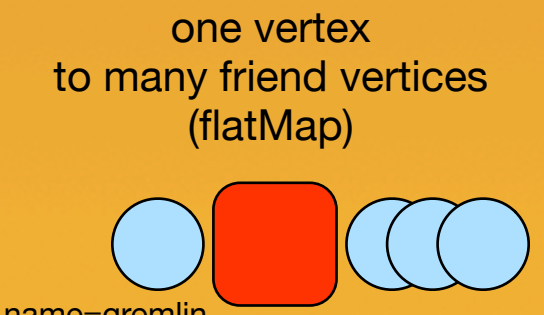




```
g.V().has("name","gremlin").
  out("knows").values("age").
  groupCount()
```

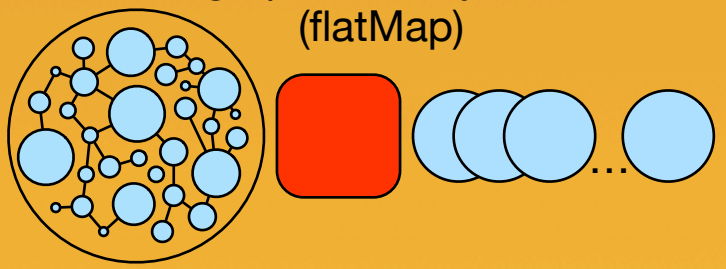




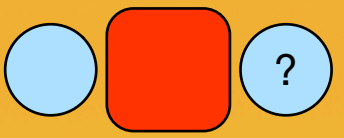


```
g.V().has("name","gremlin").
  out("knows").values("age").
  groupCount()
```

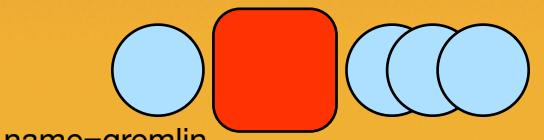
one graph to many vertices
(flatMap)



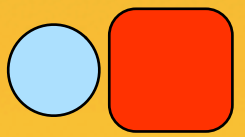
one vertex
to that vertex or no vertex
(filter)



one vertex
to many friend vertices
(flatMap)

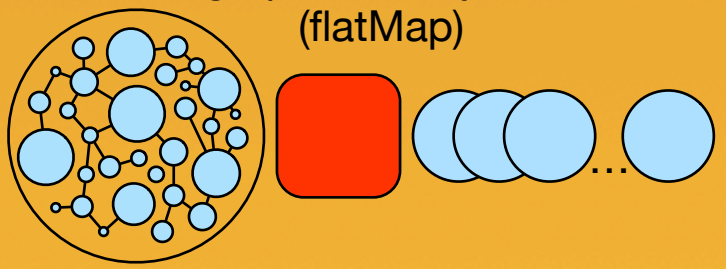


one vertex to
one age value
(map)

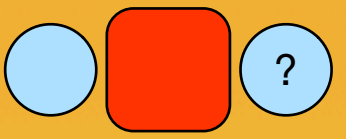


```
g.V().has("name","gremlin").
  out("knows").values("age").
  groupCount()
```

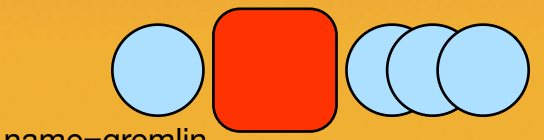
one graph to many vertices
(flatMap)



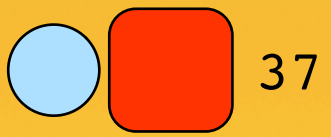
one vertex
to that vertex or no vertex
(filter)



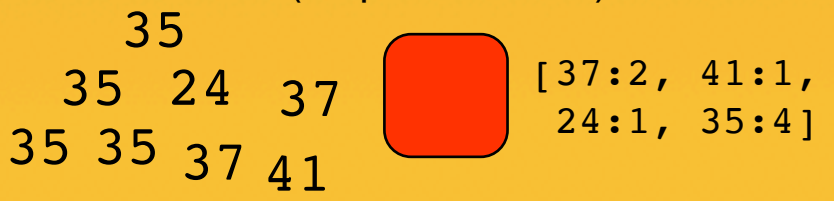
one vertex
to many friend vertices
(flatMap)



one vertex to
one age value
(map)



many age values
to an age distribution
(map — reducer)



The Gremlin Traversal Language

The Gremlin Traversal Machine

Gremlin Traversal Language

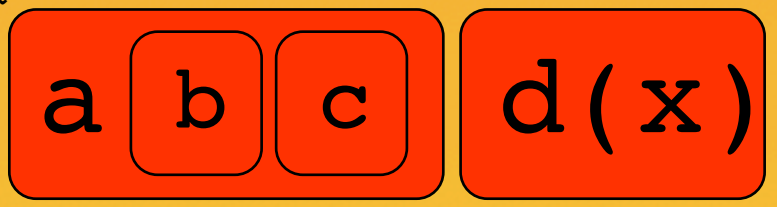
function composition

Traversal creation via
step composition

fluent methods

```
a().b().c()
```

function nesting



Step parameterization via
traversal and constant nesting

method arguments

```
a(b().c()).d(x)
```

Any language that supports function composition and function nesting can **host** Gremlin.

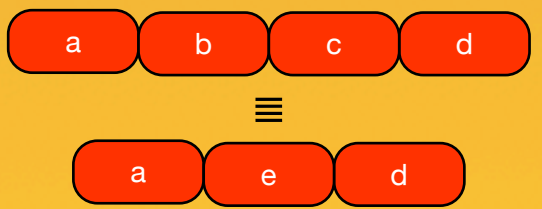
Gremlin Traversal Machine

```
class Traverser<V> {
 V value;
 long bulk;
}
```

```
class Step<S,E> {
  Traverser<E> processNextStart();
}
```

$f(x)$

```
class Traversal<S,E> implements Iterator<E> {
  E next();
  Traverser<E> nextTraverser();
}
```




```
interface TraversalStrategy {
 void apply(Traversal traversal);
 Set<TraversalStrategy> applyPrior();
 Set<TraversalStrategy> applyPost();
}
```

The fundamental constructs of Gremlin's machinery.

Gremlin-Java

```
g.V(1).
  repeat(out("knows")).times(2).
  groupCount().by("age")
```

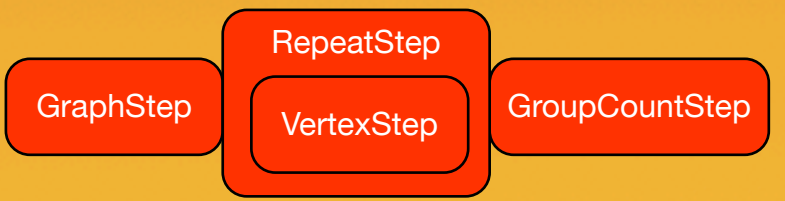
translates

Bytecode

```
[[V, 1]
[repeat, [[out, knows]]]
[times, 2]
[groupCount]
[by, age]]
```

compiles

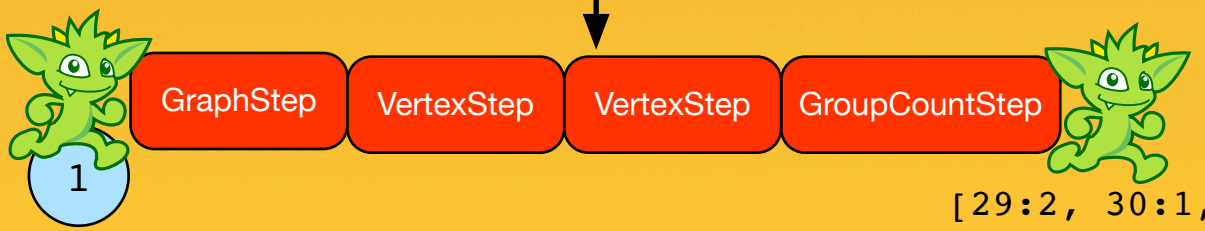
Traversal



Traversal Strategies

optimizes

executes



Gremlin-Python

```
g.V(1).
  repeat(out('knows')).times(2).
  groupCount().by('age')
```

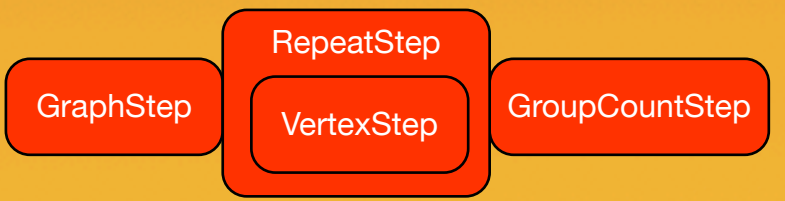
translates

Bytecode

```
[[V, 1]
[repeat, [[out, knows]]]
[times, 2]
[groupCount]
[by, age]]
```

compiles

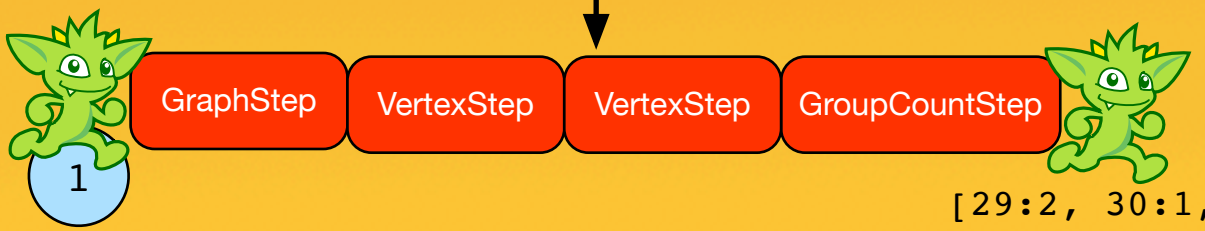
Traversal

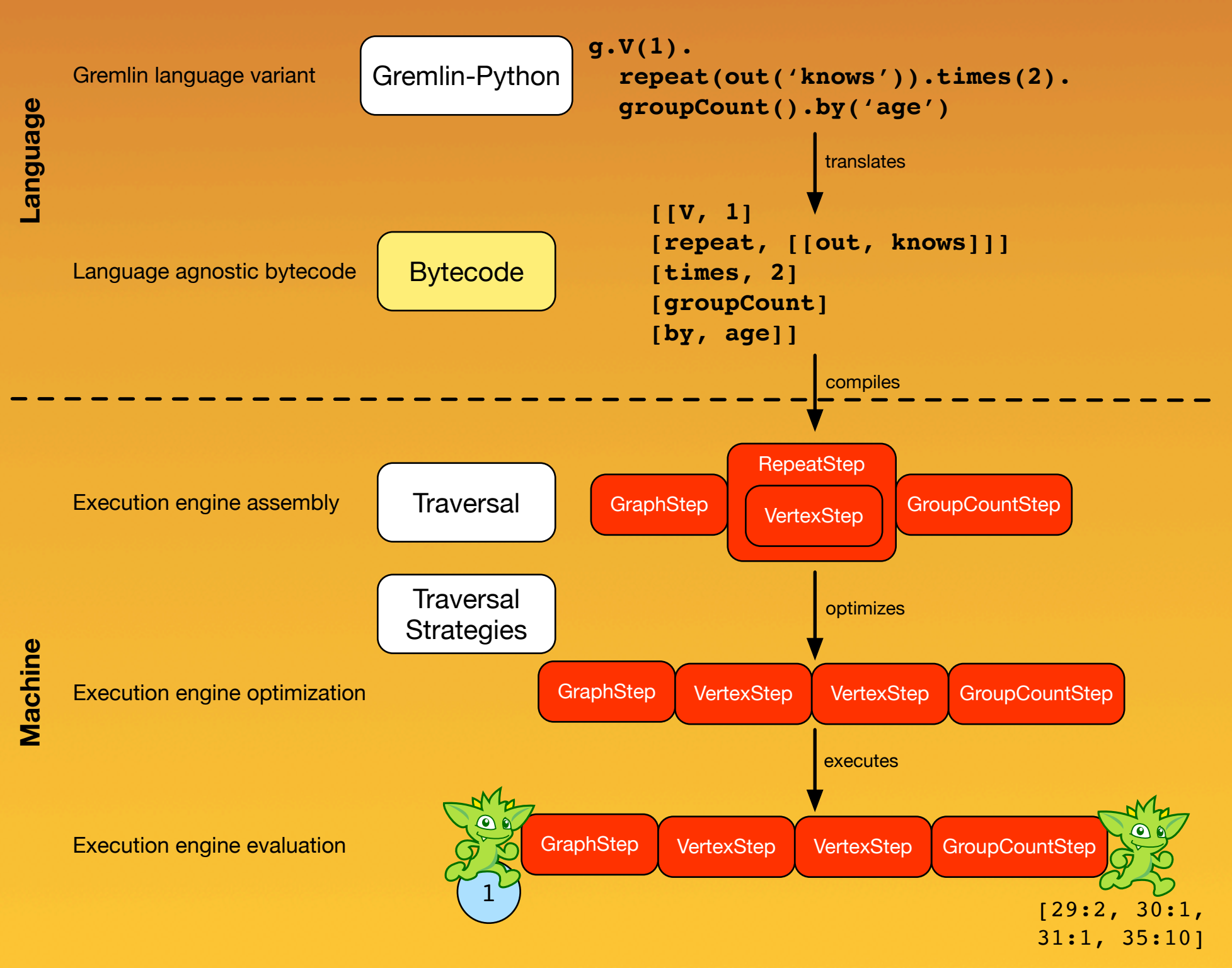


Traversal Strategies

optimizes

executes




Language
Gremlin language variant
Gremlin-Python
g.V(1).
repeat(out('knows')).times(2).
groupCount().by('age')
translates
Language agnostic bytecode
Bytecode
[[V, 1]
[repeat, [[out, knows]]]
[times, 2]
[groupCount]
[by, age]]
compiles
Machine
Execution engine assembly
Traversal
RepeatStep
GraphStep
VertexStep
GroupCountStep
Traversal Strategies
optimizes
Execution engine optimization
GraphStep
VertexStep
VertexStep
GroupCountStep
executes
Execution engine evaluation
1
GraphStep
VertexStep
VertexStep
GroupCountStep
[29:2, 30:1,
31:1, 35:10]

Language

Gremlin-Python

Gremlin-Java

Gremlin-Ruby

JavaScript

Gremlin-JavaScript

Gremlin-Groovy

Gremlin-Scala

Clojure

Gremlin-Clojure

Bytecode

Machine

Java-based Implementation

? ? ?

?-based Implementation

```
Python 2.7.2 (default, Oct 11 2012, 20:14:37)
[GCC 4.2.1 Compatible Apple Clang 4.0 (tags/Apple/clang-418.0.60)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> from gremlin_python.structure.graph import Graph
>>> from gremlin_python.driver.driver_remote_connection import DriverRemoteConnection
```

CPython

Gremlin-Python

```
Python 2.7.2 (default, Oct 11 2012, 20:14:37)
[GCC 4.2.1 Compatible Apple Clang 4.0 (tags/Apple/clang-418.0.60)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> from gremlin_python.structure.graph import Graph
>>> from gremlin_python.driver.driver_remote_connection import DriverRemoteConnection
>>> graph = Graph()
>>> g = graph.traversal().withRemote(DriverRemoteConnection('ws://localhost:8182','g'))
```

CPython

DriverRemoteConnection

Gremlin-Python

```
Python 2.7.2 (default, Oct 11 2012, 20:14:37)
[GCC 4.2.1 Compatible Apple Clang 4.0 (tags/Apple/clang-418.0.60)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> from gremlin_python.structure.graph import Graph
>>> from gremlin_python.driver.driver_remote_connection import DriverRemoteConnection
>>> graph = Graph()
>>> g = graph.traversal().withRemote(DriverRemoteConnection('ws://localhost:8182','g'))
# nested traversal with Python slicing and attribute interception extensions
>>> g.V().hasLabel("person").repeat(both()).times(2).name[0:2].toList()
[u'marko', u'marko']
```

Gremlin-Python

CPython

Bytecode

DriverRemoteConnection

JVM

Apache TinkerPop

Gremlin Traversal Machine

```
Python 2.7.2 (default, Oct 11 2012, 20:14:37)
[GCC 4.2.1 Compatible Apple Clang 4.0 (tags/Apple/clang-418.0.60)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> from gremlin_python.structure.graph import Graph
>>> from gremlin_python.driver.driver_remote_connection import DriverRemoteConnection
>>> graph = Graph()
>>> g = graph.traversal().withRemote(DriverRemoteConnection('ws://localhost:8182','g'))
# nested traversal with Python slicing and attribute interception extensions
>>> g.V().hasLabel("person").repeat(both()).times(2).name[0:2].toList()
[u'marko', u'marko']
# a complex, nested multi-line traversal
>>> g.V().match( \
...     as_("a").out("created").as_("b"), \
...     as_("b").in_("created").as_("c"), \
...     as_("a").out("knows").as_("c")). \
...   select("c"). \
...   union(in_("knows"),out("created")). \
...   name.toList()
[u'ripple', u'marko', u'lop']
>>>
```

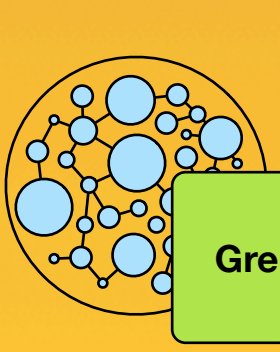
Gremlin-Python

CPython

Bytecode

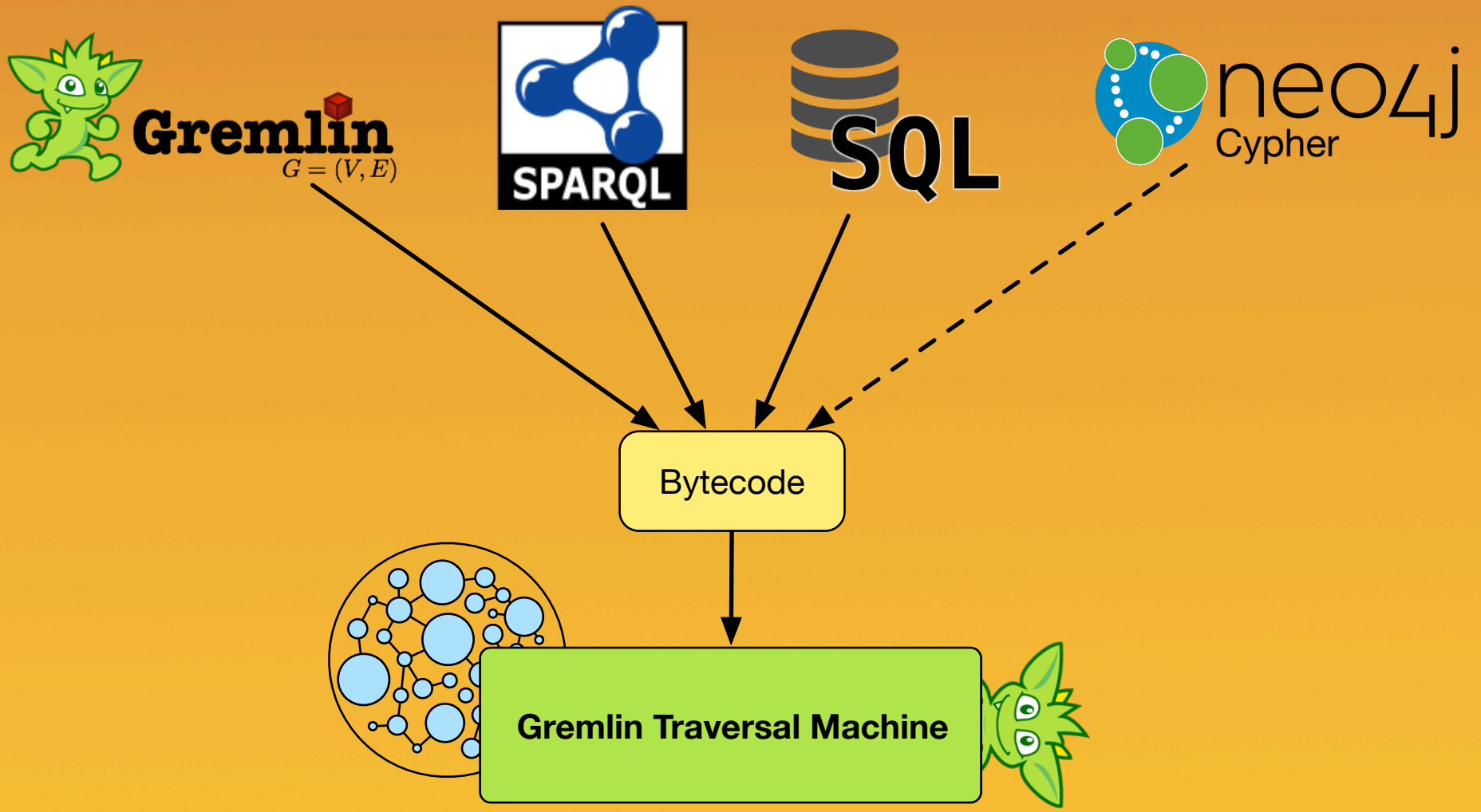
DriverRemoteConnection

JVM



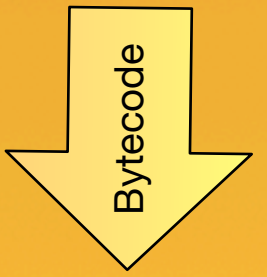
Apache TinkerPop

Gremlin Traversal Machine

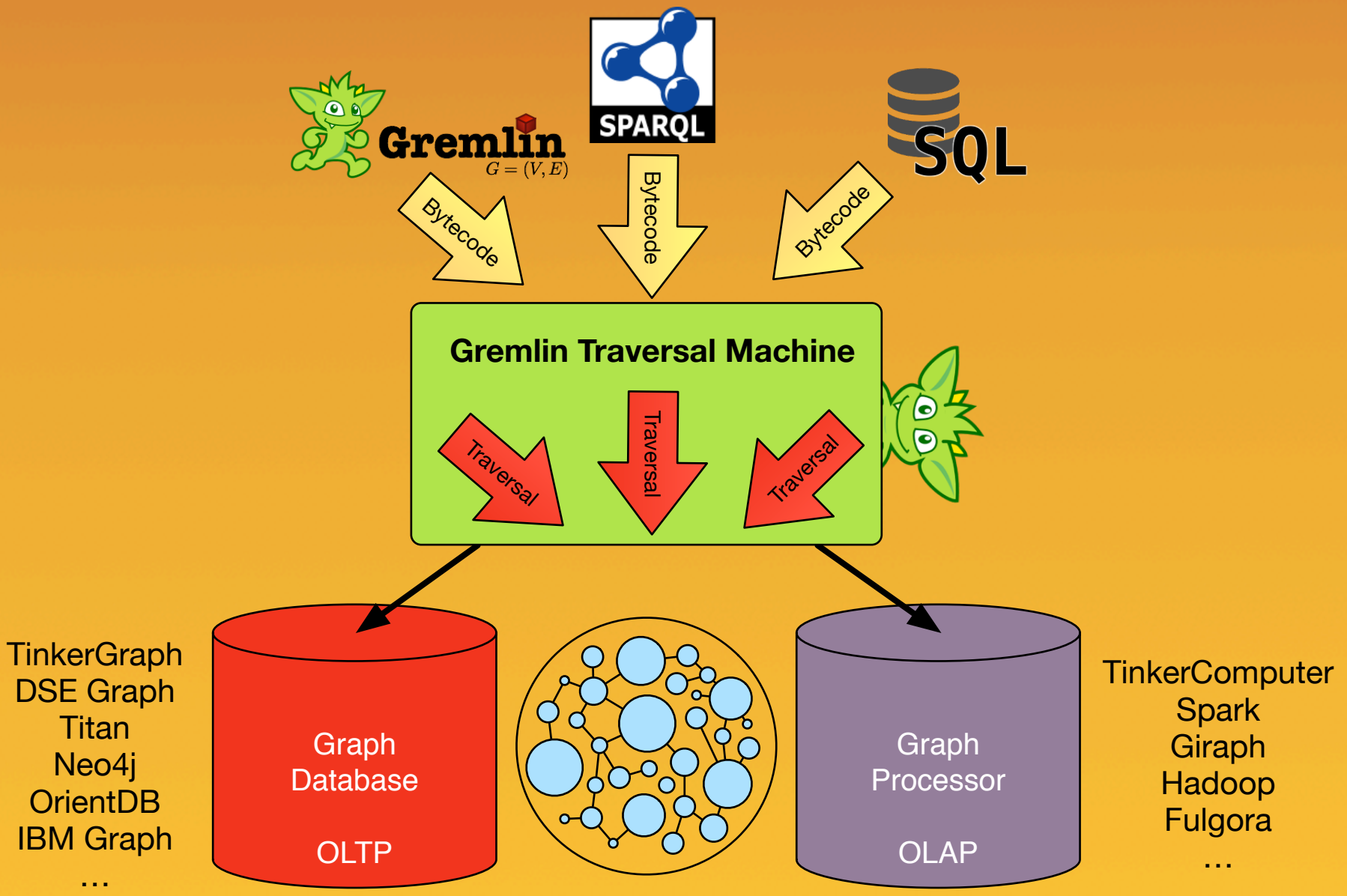


Distinct query languages (not only Gremlin language variants) can generate bytecode for evaluation by any OLTP/OLAP TinkerPop-enabled graph system.

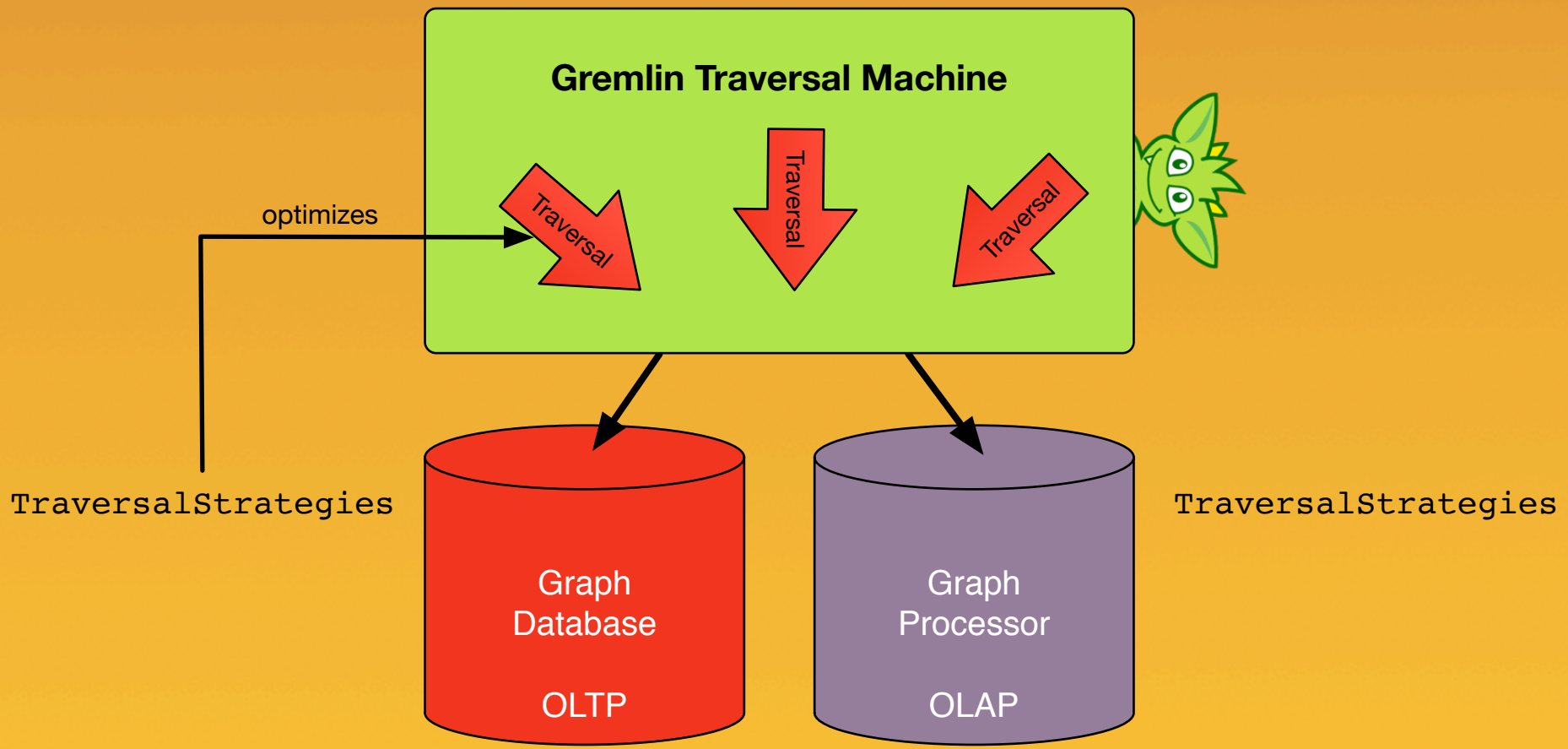

```
SELECT DISTINCT ?c
WHERE {
  ?a v:label "person" .
  ?a e:created ?b .
  ?b v:name ?c .
  ?a v:age ?d .
    FILTER (?d > 30)
}
```




```
[
  [V],
  [match,
    [[as, a], [hasLabel, person]],
    [[as, a], [out, created], [as, b]],
    [[as, a], [has, age, gt(30)]]],
  [select, b],
  [by, name],
  [dedup]
]
```

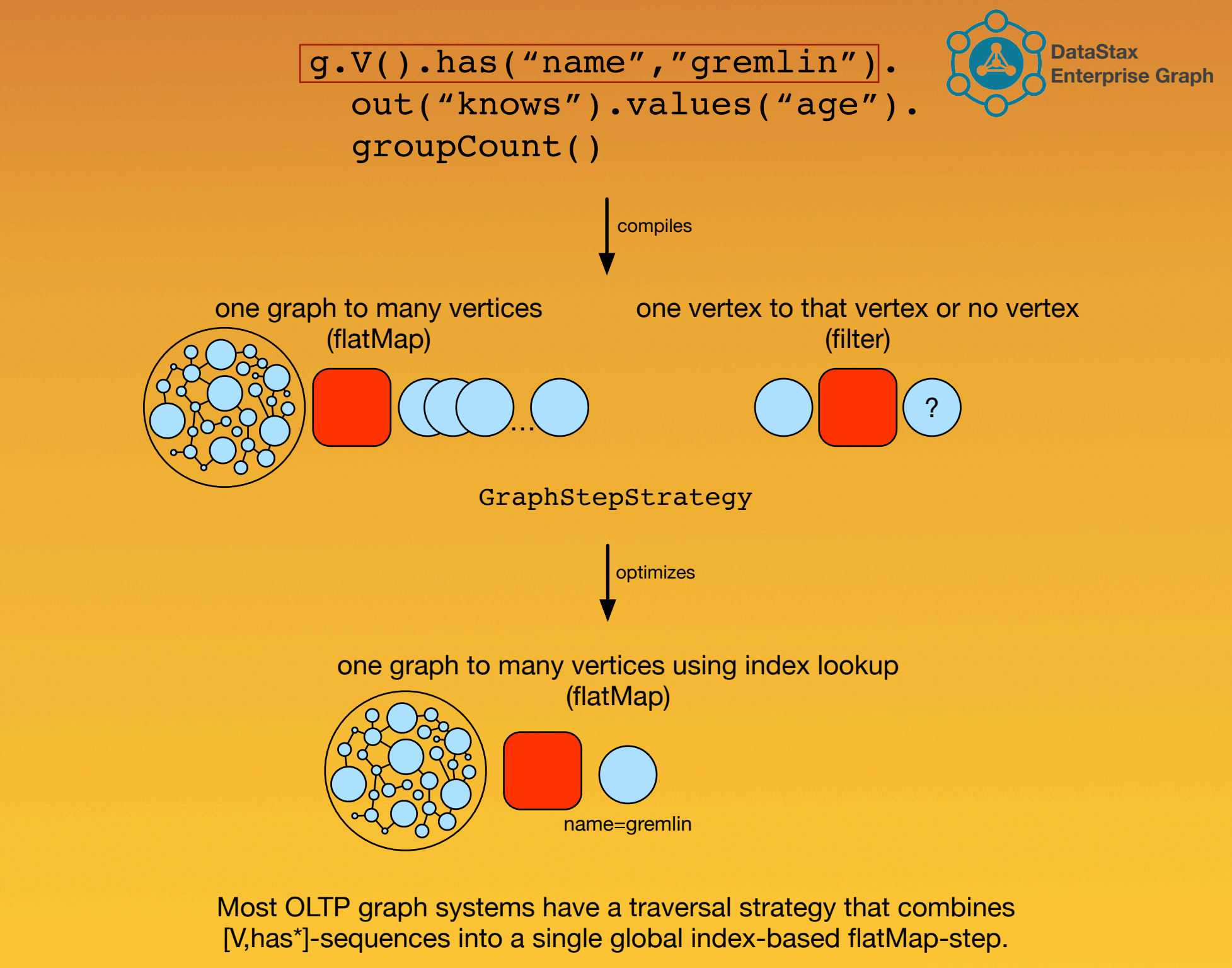


Gremlin traversals can be executed against OLTP graph databases and OLAP graph processors. That means that if, e.g., SPARQL compiles to bytecode, it can execute both OLTP and OLAP.

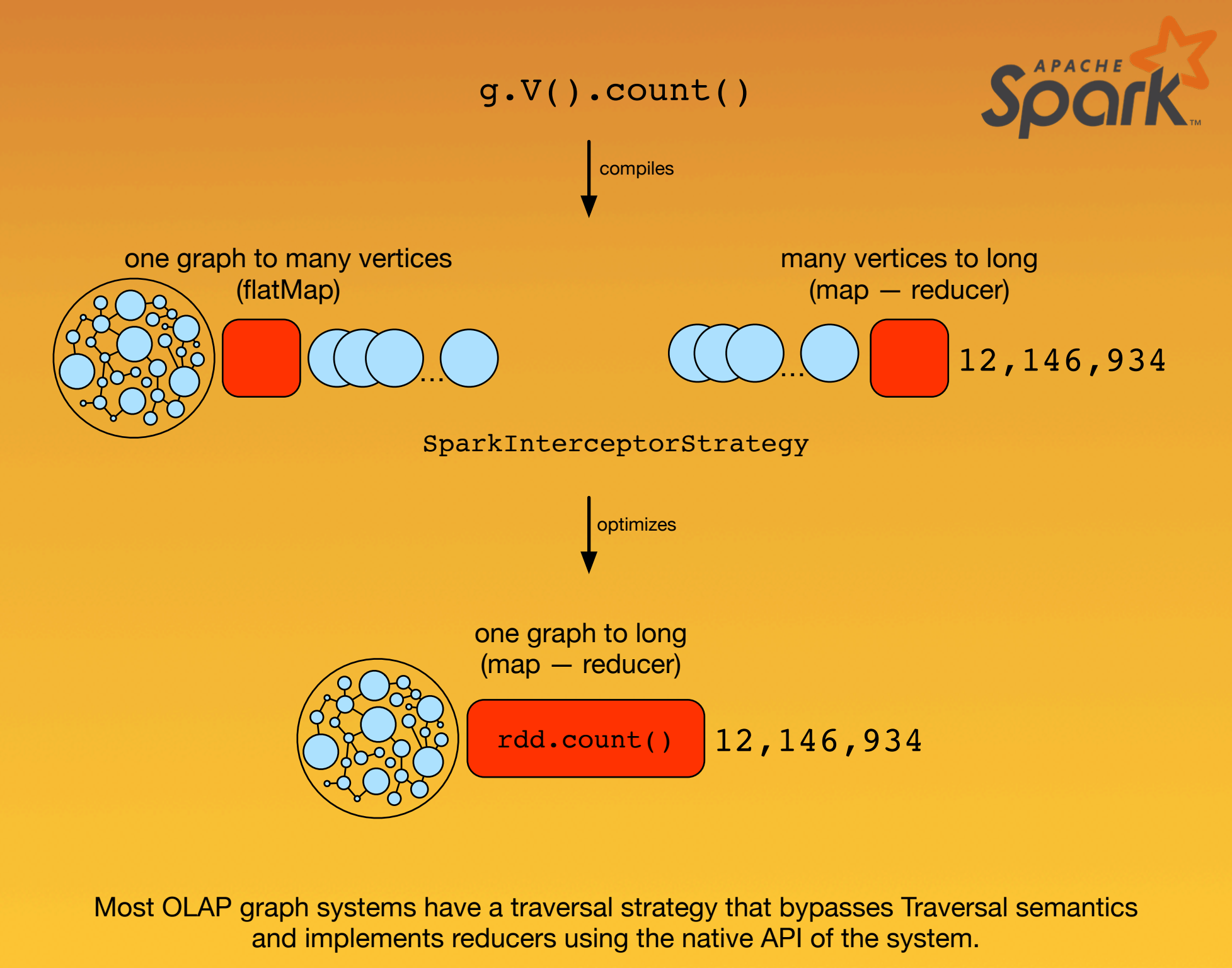


Graph system providers (OLTP/OLAP) typically have custom strategies registered with the Gremlin traversal machine that take advantage of unique, provider-specific features.

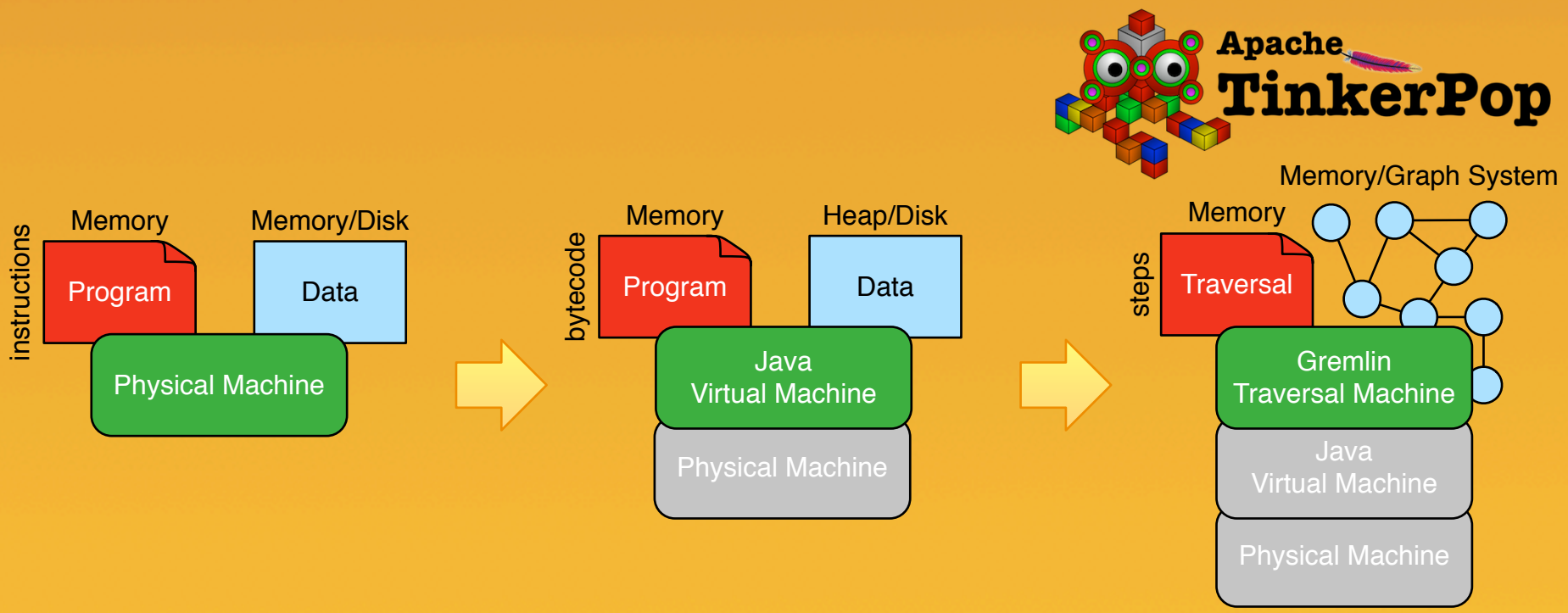
```
g.V().has("name","gremlin").
  out("knows").values("age").
  groupCount()
```



Most OLTP graph systems have a traversal strategy that combines [V,has*]-sequences into a single global index-based flatMap-step.



Most OLAP graph systems have a traversal strategy that bypasses Traversal semantics and implements reducers using the native API of the system.



From the physical computing machine to the Gremlin traversal machine.

Stakeholders

Application Developers

Language Providers

- Gremlin Language Variant
- Distinct Query Language

Graph System Providers

- OLTP Provider
- OLAP Provider

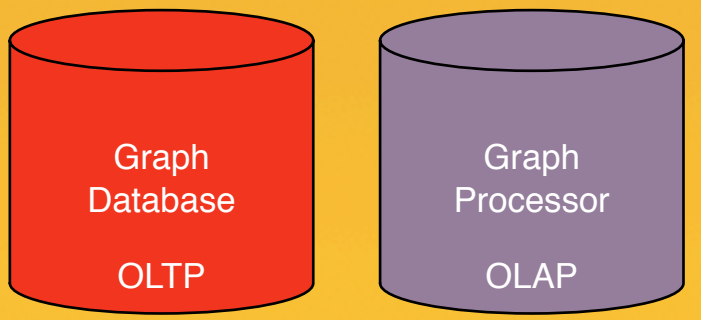
Stakeholders

Application Developers

One query language for all OLTP/OLAP systems.

Gremlin

$G = (V, E)$



Real-time and analytic queries are represented in Gremlin.

Stakeholders

Application Developers

- One query language for all OLTP/OLAP systems.
- No vendor lock-in.

DataStax Enterprise Graph

OrientDB®

Spark

neo4j

hadoop

APACHE GIRAPH

Apache TinkerPop as the JDBC for graphs.

Stakeholders

Application Developers

- One query language for all OLTP/OLAP systems.
- No vendor lock-in.
- Gremlin is embedded in the developer's language.

SQL in Java

```
ResultSet result = statement.executeQuery(
  "SELECT name FROM People \n" +
  " ORDER BY age \n" +
  " LIMIT 10")
```

vs.

Gremlin-Java

```
Iterator<String> result =
  g.V().hasLabel("person").
    order().by("age").
    limit(10).values("name")
```

No "fat strings." The developer writes their graph database/processor queries in their native programming language.

Stakeholders

Language Providers

Gremlin Language Variant

Distinct Query Language

➕ Easy to generate bytecode.

```
GraphTraversal.getMethods()
 .findAll { GraphTraversal.class == it.returnType }
 .collect { it.name }
 .unique()
 .each {
   pythonClass.append(
"""  def ${it}(self, *args):
    self.bytecode.add_step("${it}", *args)
    return self
""")}
```

Gremlin-Python's source code is programmatically generated using Java reflection.

Stakeholders

Language Providers

Gremlin Language Variant
Distinct Query Language

- Easy to generate bytecode.
- Bytecode executes against TinkerPop-enabled systems.

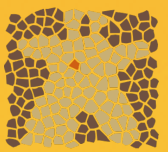
DataStax
Enterprise Graph

OrientDB®

Spark

neo4j

hadoop



APACHE
GIRAPH

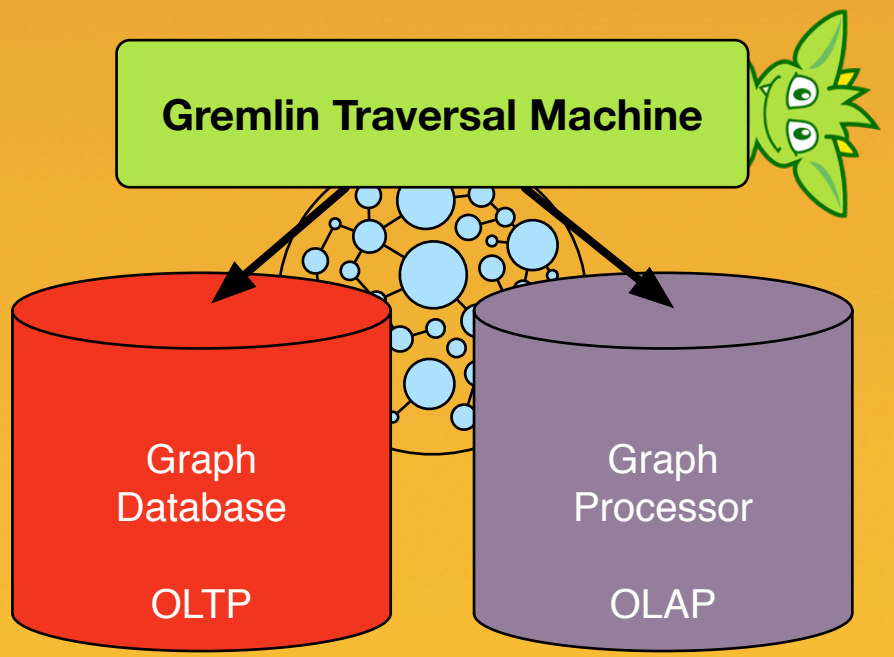
Language providers write a translator for the Gremlin traversal machine, not a particular graph database/processor.

Stakeholders

Language Providers

Gremlin Language Variant
Distinct Query Language

- Easy to generate bytecode.
- Bytecode executes against TinkerPop-enabled systems.
- Provider can focus on design, not evaluation.



The language designer does not have to concern themselves with OLTP or OLAP execution. They simply generate bytecode and the Gremlin traversal machine handles the rest.

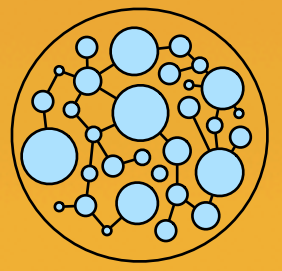
Stakeholders

Graph System Providers

- OLTP Provider
- OLAP Provider

Easy to implement core interfaces.

`Graph`



`Vertex`

`Edge`

`Transaction`

`Property`
`key=value`

`TraversalStrategy`

Stakeholders

Graph System Providers

- OLTP Provider
- OLAP Provider

- Easy to implement core interfaces.
- Provider supports all provided languages.

The provider automatically supports all query languages that have compilers that generate Gremlin bytecode.

Stakeholders

Graph System Providers

OLTP Provider
OLAP Provider

- Easy to implement core interfaces.
- Provider supports all provided languages.
- OLTP providers can leverage existing OLAP systems.

DataStax Enterprise Graph

Spark

DSE Graph leverages SparkGraphComputer for OLAP processing.

Stakeholders

Application Developers

- One query language for all OLTP/OLAP systems.
- No vendor lock-in.
- Gremlin is embedded in the developer's language.

Language Providers

Gremlin Language Variant
Distinct Query Language

- Easy to generate bytecode.
- Bytecode executes against TinkerPop-enabled systems.
- Provider can focus on design, not evaluation.

Graph System Providers

OLTP Provider
OLAP Provider

- Easy to implement core interfaces.
- Provider supports all provided languages.
- OLTP providers can leverage existing OLAP systems.

Thank you.

http://tinkerpop.apache.org
http://www.datastax.com/products/datastax-enterprise-graph