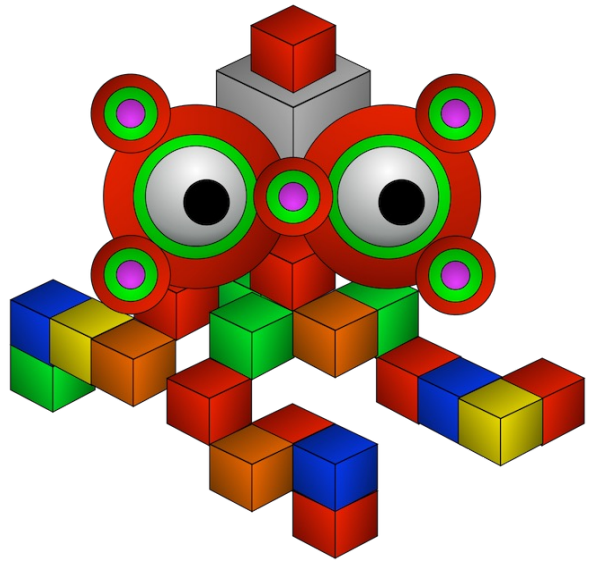


TinkerGQL

Mixing declarative GQL with imperative Gremlin

Cole Greer Improving / Apache TinkerPop

Agenda

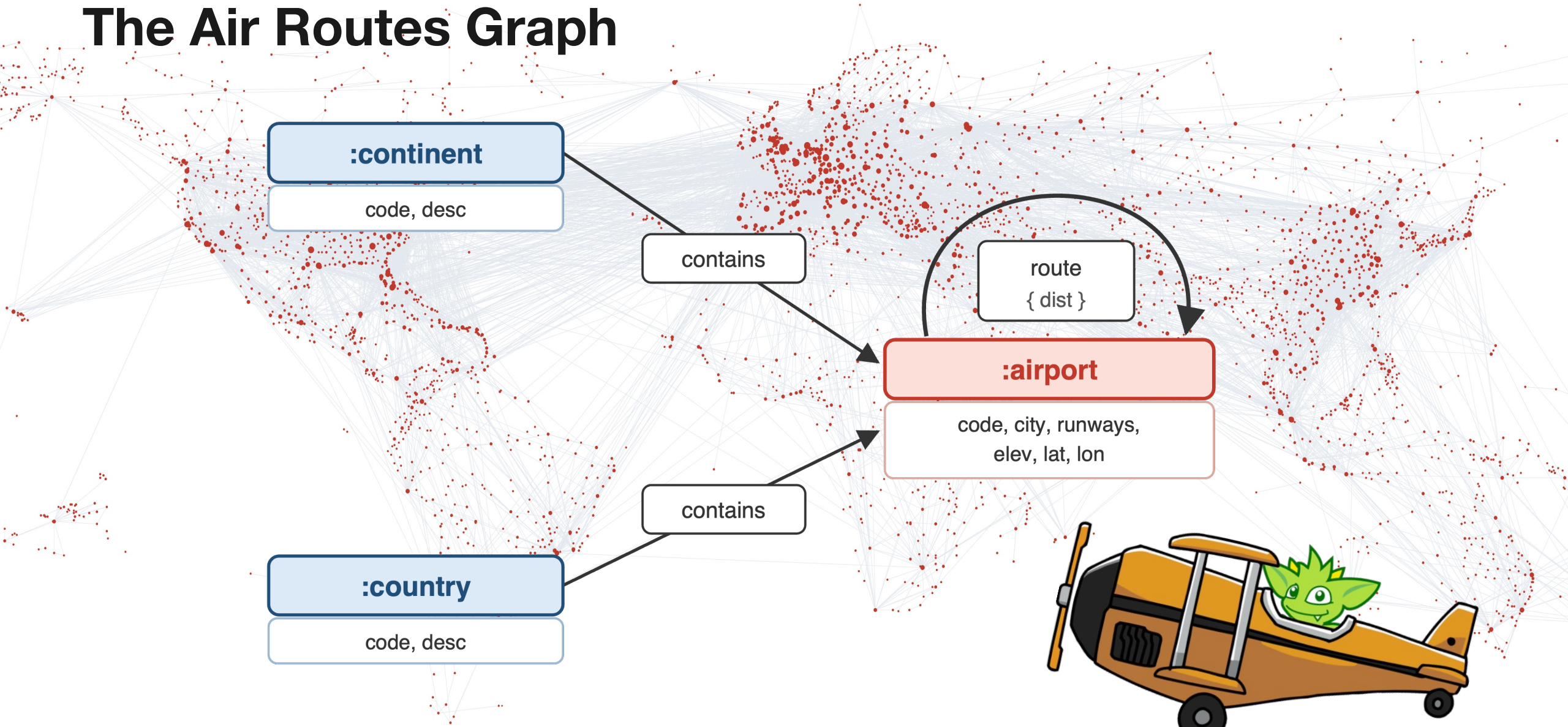
Introduction to Gremlin

My perspective on GQL

TinkerGQL, with worked examples

A look to the Future

The Air Routes Graph



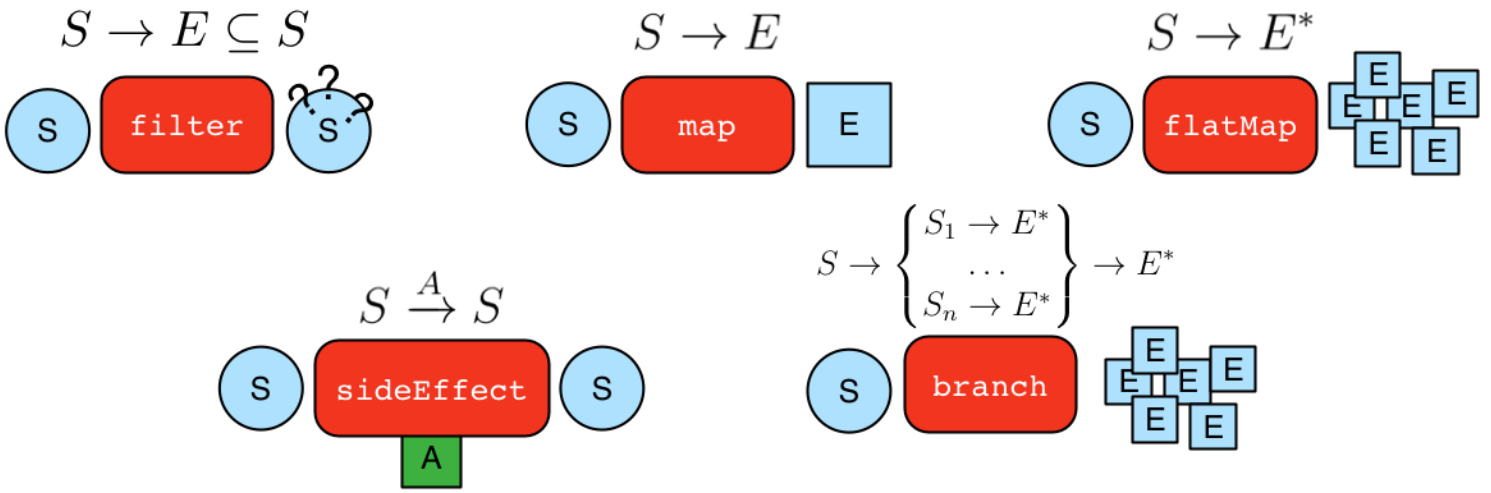
3504 airports, 57645 edges. Every example in this talk runs on it.

Gremlin

Gremlin in One Query

```
g.V()                              // start with every vertex
 .has('airport','code','YVR')      // filter
 .out('route').out('route')        // walk two hops
 .values('code')                   // compute
```

- A traversal is a chain of steps: each one walks, filters, or computes
- Fully open source, built and maintained by the Apache TinkerPop community



Accidental Complexity of Simple Walks

Gremlin

```
g.V().has('airport','code','YVR').
  repeat(out('route').simplePath()).
    emit(loops().is(gte(2))).
    times(4).
  groupCount().by('code').
  order(local).by(values, desc).
  limit(local, 10)
```

GQL

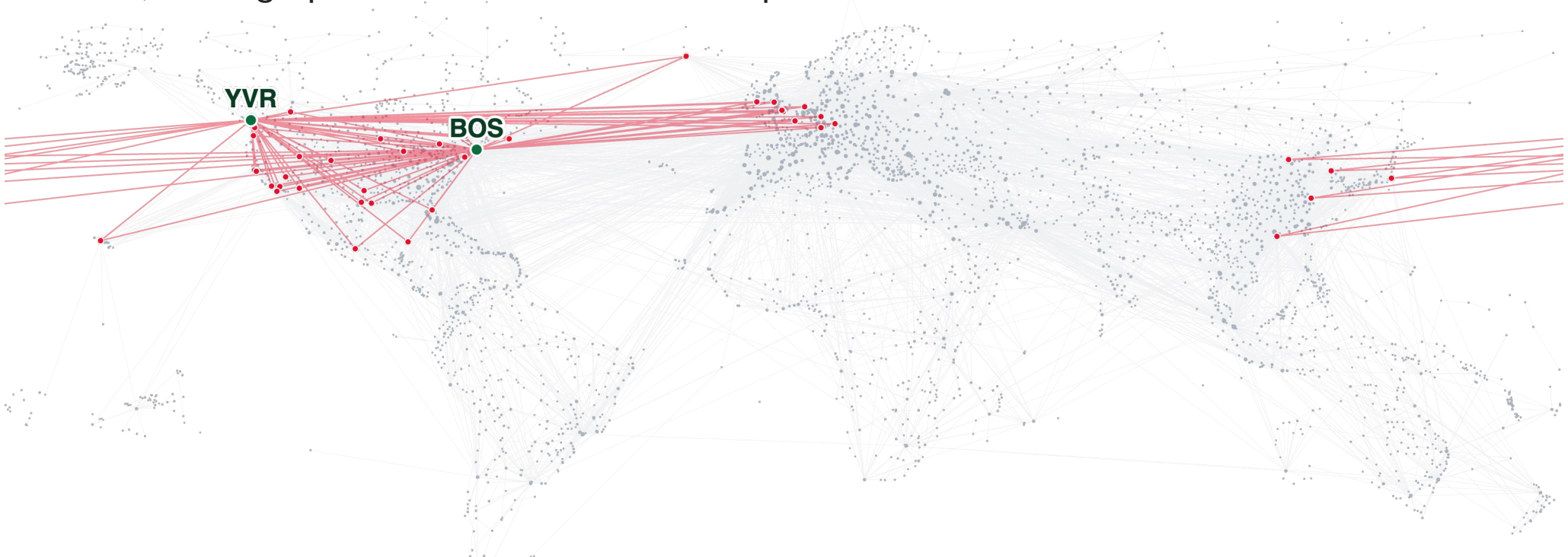
```
MATCH (a:airport {code: 'YVR'})-[r:route]->{2,4}(b:airport)
RETURN b.code, count(*) AS routings ORDER BY routings DESC LIMIT 10
```

GQL

Why GQL Is Awesome

```
MATCH (yvr:airport {code:'YVR'})-[:route]->(hub)-[:route]->(bos:airport {code:'BOS'})
```

- The query is a picture of the data you want
- One ISO standard, portable across engines
- Readable at a glance, teachable in minutes
- Declarative, leaving optimization concerns to the planner



Hard or Impossible in Pure GQL

Stop walking when the running cost exceeds a budget

```
g.withSack(0).V().has('airport','code','YVR').
  repeat(outE('route').sack(sum).by('dist').inV().
         filter(sack().is(lt(3000)))).
    until(has('code','BOS'))
```

Write a computed value as you iterate, feeding the next hop

```
g.withSack(1.0).V().has('airport','code','YVR').
  repeat(outE('route').sack(mult).by('dist').inV().
         property('load', sack())).times(3)
```

Also out of reach: arbitrary computation inline, and calling a service mid-traversal.

Declarative Languages Get Supplemented

A declarative language says what you want. Something else computes.

- GQL has CALL
- Cypher has procedures

TinkerGQL puts both halves in one query.

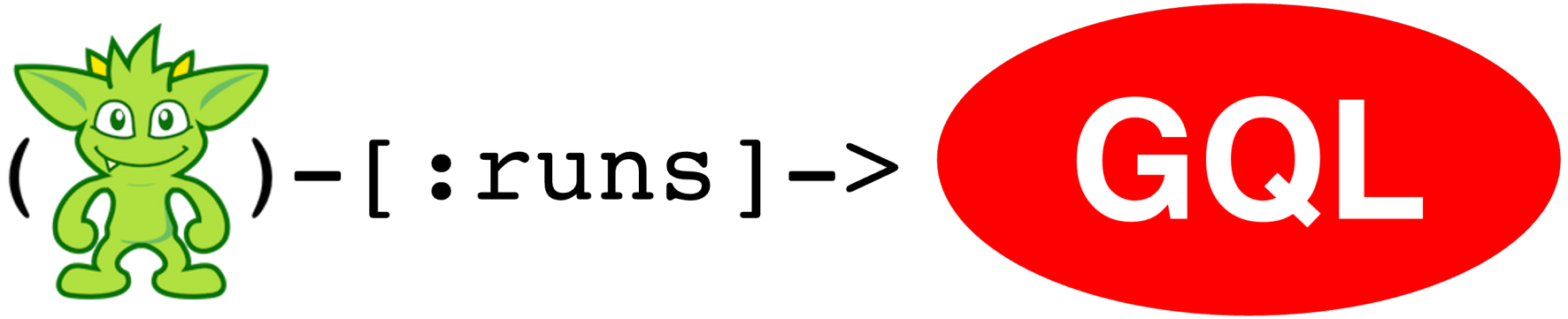
TinkerGQL

TinkerGQL: Scope

A minimal subset of ISO GQL MATCH, in the optional `gql-gremlin` module.

Supported Node patterns, three edge directions, parameters, comma joined patterns, and quantified relationships: `{m,n}` `{m}` `{m,}` `+`

Not supported WHERE, RETURN, INSERT, SET, DELETE, inequalities, OPTIONAL MATCH, aggregation



Bindings Are the Currency

```
g.match("MATCH (a:airport {code:'YVR'})-[:route]->(b:airport)")

// ==> {a:v[YVR], b:v[SEA]}
// ==> {a:v[YVR], b:v[LAX]}    ... about a hundred rows
```

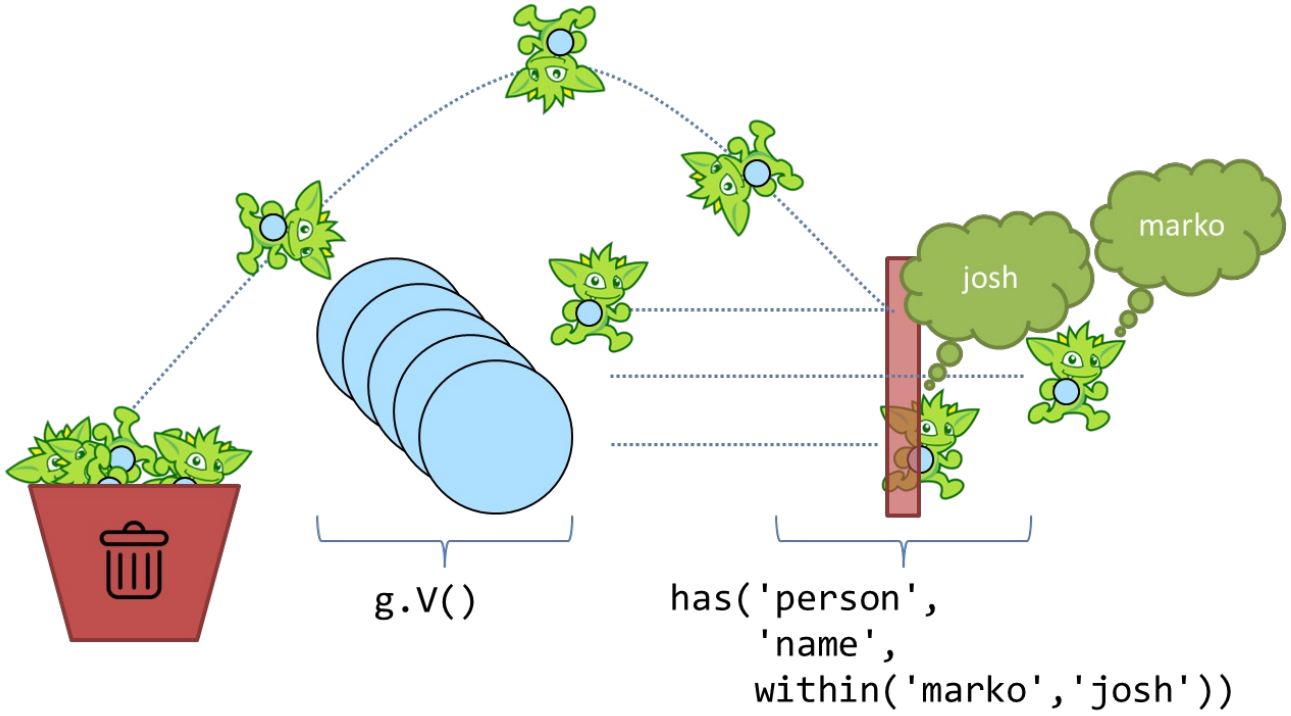
```
g.match("MATCH (a:airport {code:'YVR'})-[:route]->(b:airport)").
  select('b').by('code')

// ==> SEA, LAX, SFO, YYZ, ...
```

There is no RETURN. The step emits a binding map, and `select()` reads it.

Gremlin Filters What TinkerGQL Cannot

```
g.match("MATCH (a:airport {code:'YVR'})-[:route]->(b:airport)").
  where(select('b').by('runways').is(gte(3))).
  select('b').by('code')

// ==> LAX, SFO, YYZ, ORD, ...
```

TinkerGQL tests equality only. The inequality lives in `where()`.



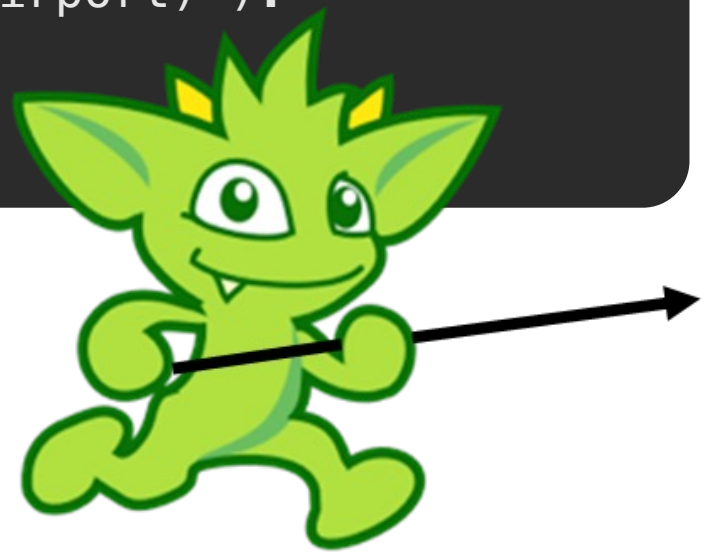
Keep Walking From a Binding

```
g.match("MATCH (c:country {code:'CA'})-[:contains]->(a:airport)").
  select('a').
  out('route').dedup().count()

// ==> 1163
```

The pattern picks the seed set. Gremlin carries on from there.

```
g.match("MATCH (a:airport {code:'YVR'})-[r:route]->(b:airport)").
  select('r').values('dist').mean()

// ==> 1876.4
```

Edges bind too, and their properties feed the computation.



GQL Simplifies Pattern Matching

Pure Gremlin

```
g.V().has('airport','code','YVR').
  repeat(out('route').simplePath()).
    emit(loops().is(gte(2))).
    times(4).
  groupCount().by('code').
  order(local).by(values, desc).
  limit(local, 10)
```

Gremlin with TinkerGQL

```
g.match("MATCH (a:airport {code:'YVR'})-[:route]->{2,4}(b:airport)").
  select('b').
  groupCount().by('code').
  order(local).by(values, desc).
  limit(local, 10)
```

The walk became a pattern. The computation stayed in Gremlin.

Provider Extensibility

TinkerPop ships the reference grammar and engine. TinkerGraph uses it out of the box.

Providers can swap in:

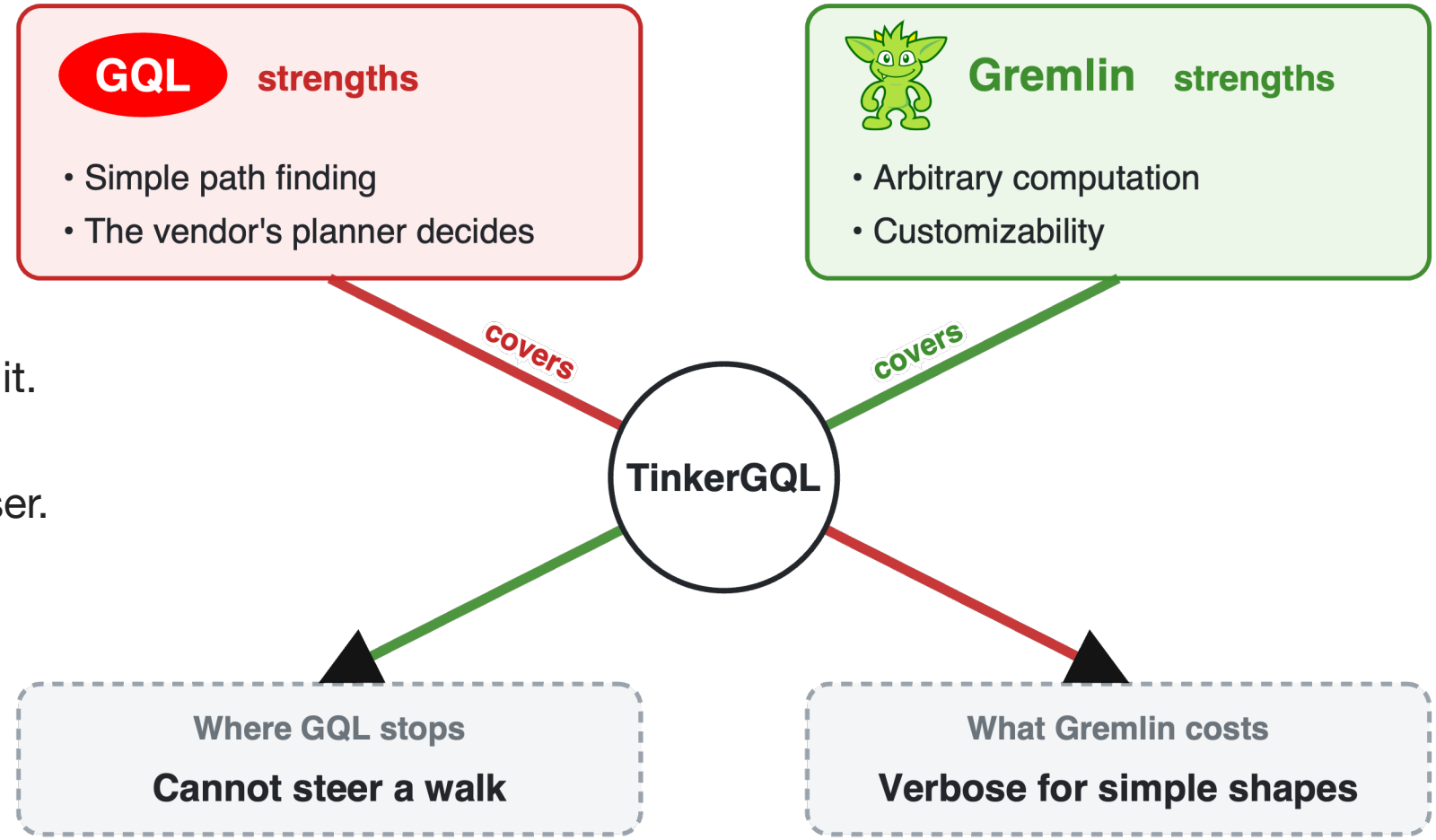
- their own optimizations
- their own engine
- an entirely separate language

Current Limitations

- Still in beta
- Variable length paths are currently WIP.
- Composition runs one way. The pattern cannot be seeded from upstream.
- Missing support for OPTIONAL MATCH

Takeaways

- Pattern matching describe shape. It does not compute.
- Gremlin is where the computation goes.
- TinkerPop 4 puts GQL at the front and leaves Gremlin behind it.
- The seam is one decision: the pattern emits an ordinary traverser.



Use GQL for the shape of the data. Use Gremlin for everything you do with it.

What About the Other Direction?

Gremlin as a subroutine, through GQL's own CALL

```
MATCH (a:airport {code: 'YVR’})
CALL gremlin {
  withSack(0).repeat(outE('route').sack(sum).by('dist').inV()).times(2).
    has('code','BOS').project('route','miles').by(path()).by(sack())
} YIELD route, miles
RETURN route, miles ORDER BY miles LIMIT 1
```

Or bound as a value expression

```
LET onward = gremlin(b) { out('route').dedup().count() }
```