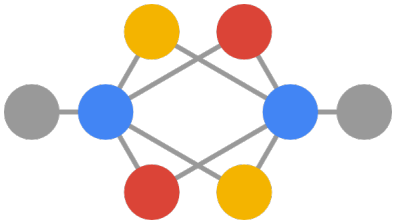
Cayley: Building a Graph Database

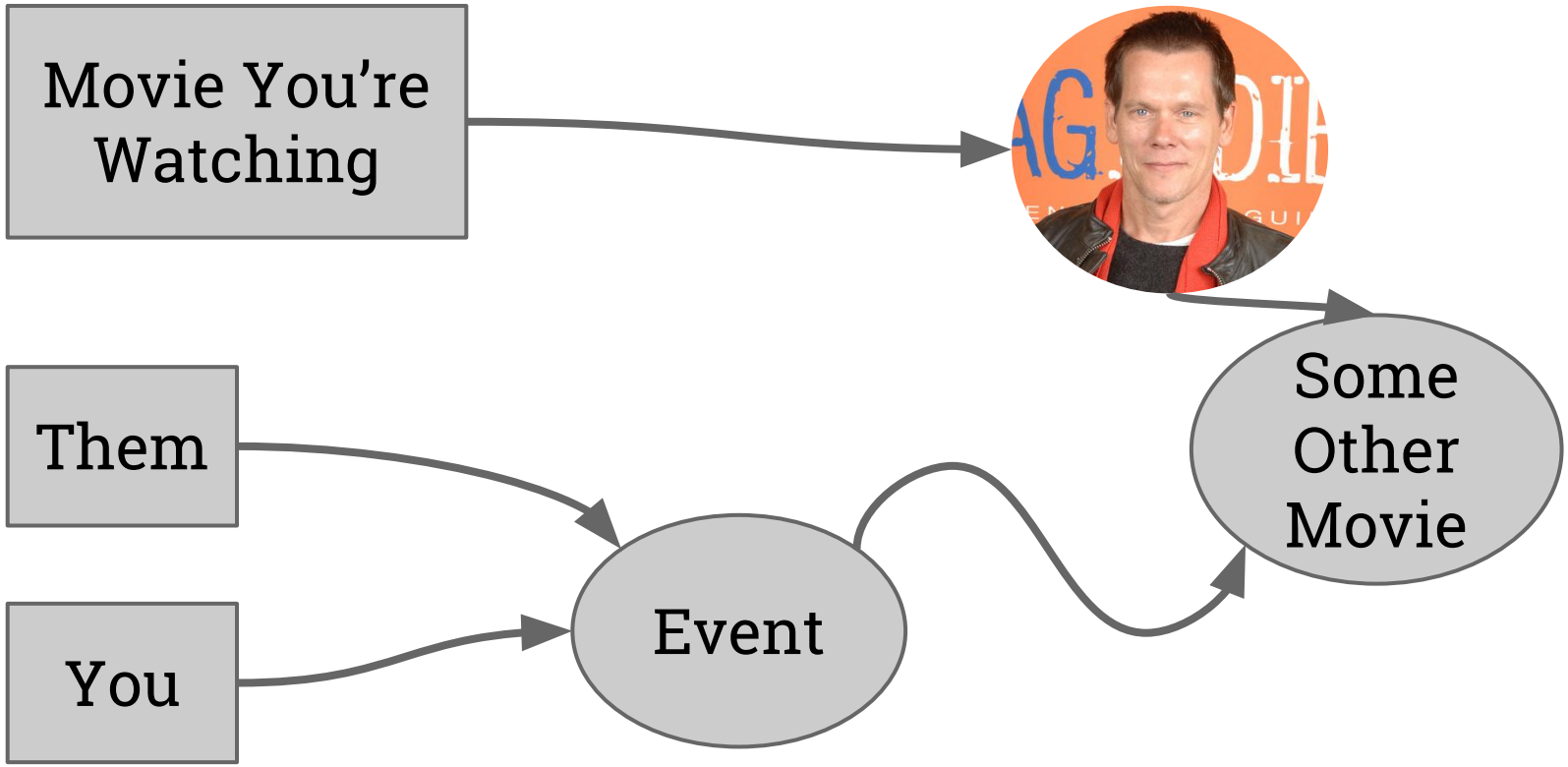


1.

Cayley Motivation

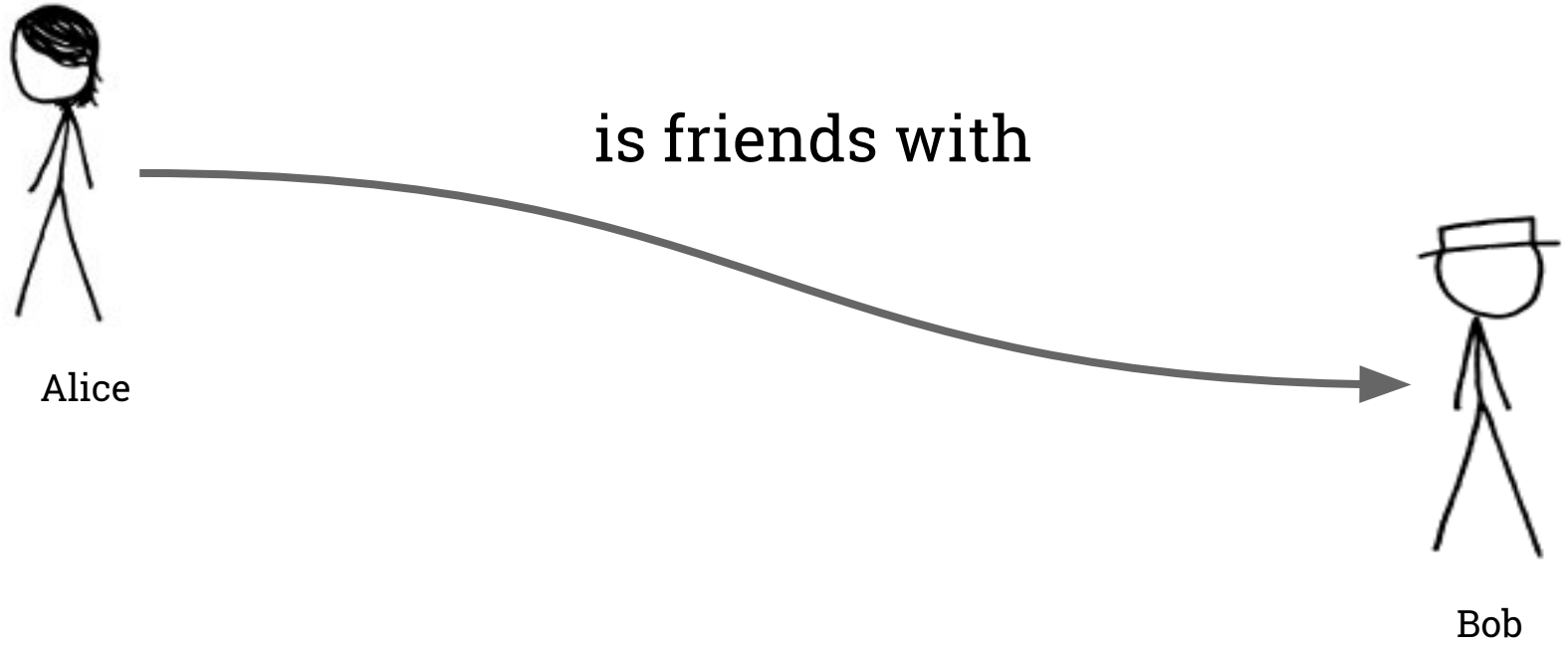
Why build a graph store?

“Oh, he was in that movie we saw that one time.”

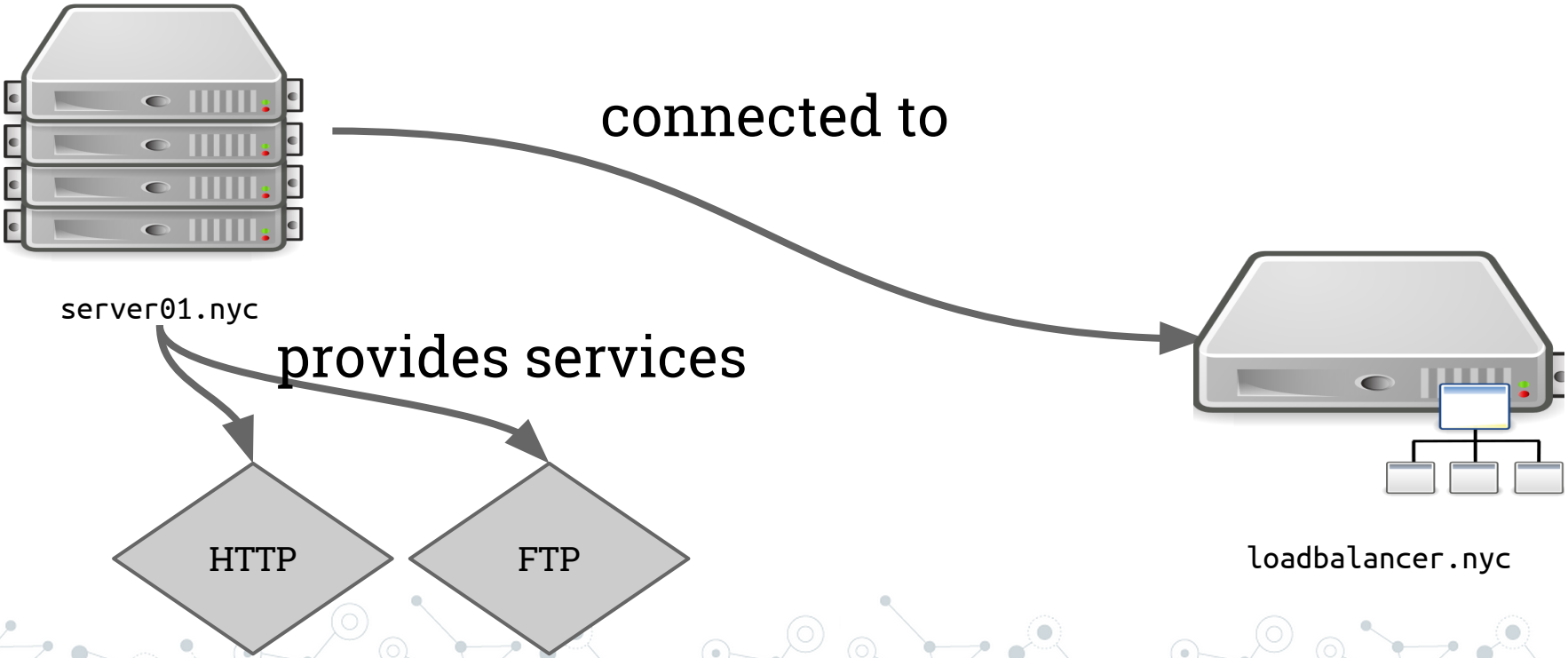

Movie You're Watching
Them
You
Event
Some Other Movie

We naturally think in terms of connections

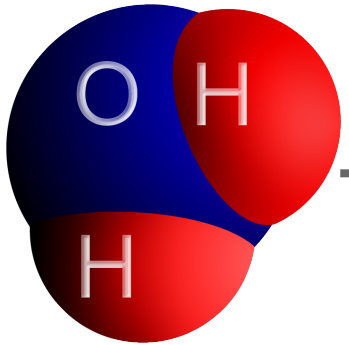
Social Networking



Computer Networking

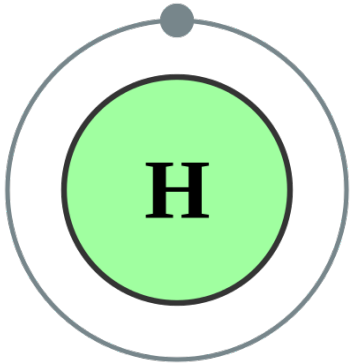


Chemistry



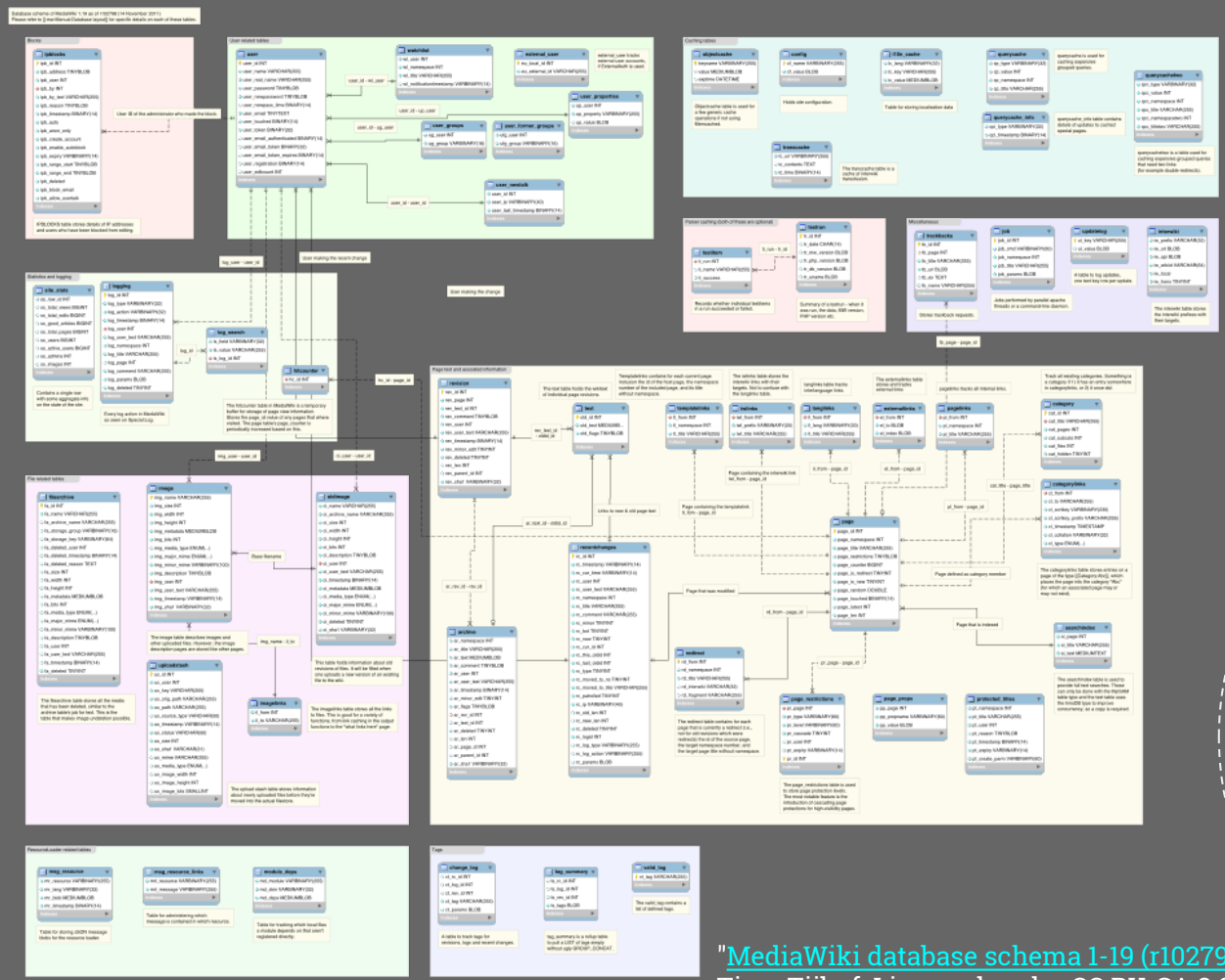
Water

contains element



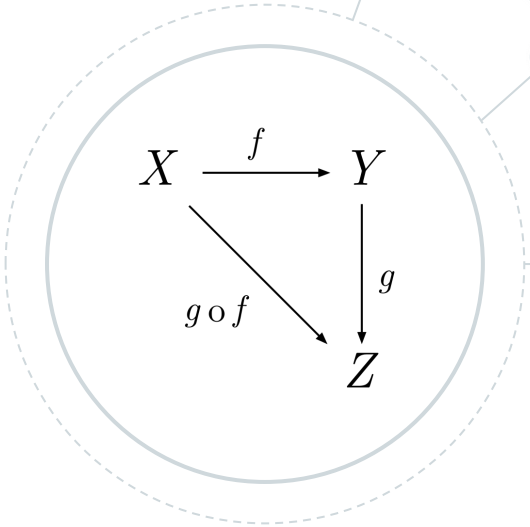
Hydrogen

Today we construct our data in tables



Think of the biggest SQL query you've ever written.

Tables are graphs



Shout out to [Spivak](#) and Kent's [Ologs](#) paper

Microservices: Why not both?

(Can we `import` a paradigm?)

2.

Cayley Origins

A very brief history of Freebase and Graphd

Freebase was awesome

And graphd was the engine for the OTG

Giving back to the community

- Freebase shutting down
- Data dumps (and Wikidata)
- Open source some tools…

… but not graphd

graphd was...

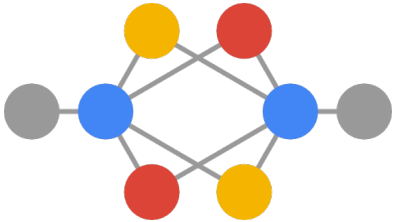
Cool

- Written in C99
- Append-only, eventually consistent
- Dynamic query planning
- Custom disk formats
- Replication
- Battle-tested

...but Unapproachable

- Written in C99
- Human-unfriendly query language
- Custom *everything* -- including poll wrapper, socket sessions, cooperative scheduling.

Cayley



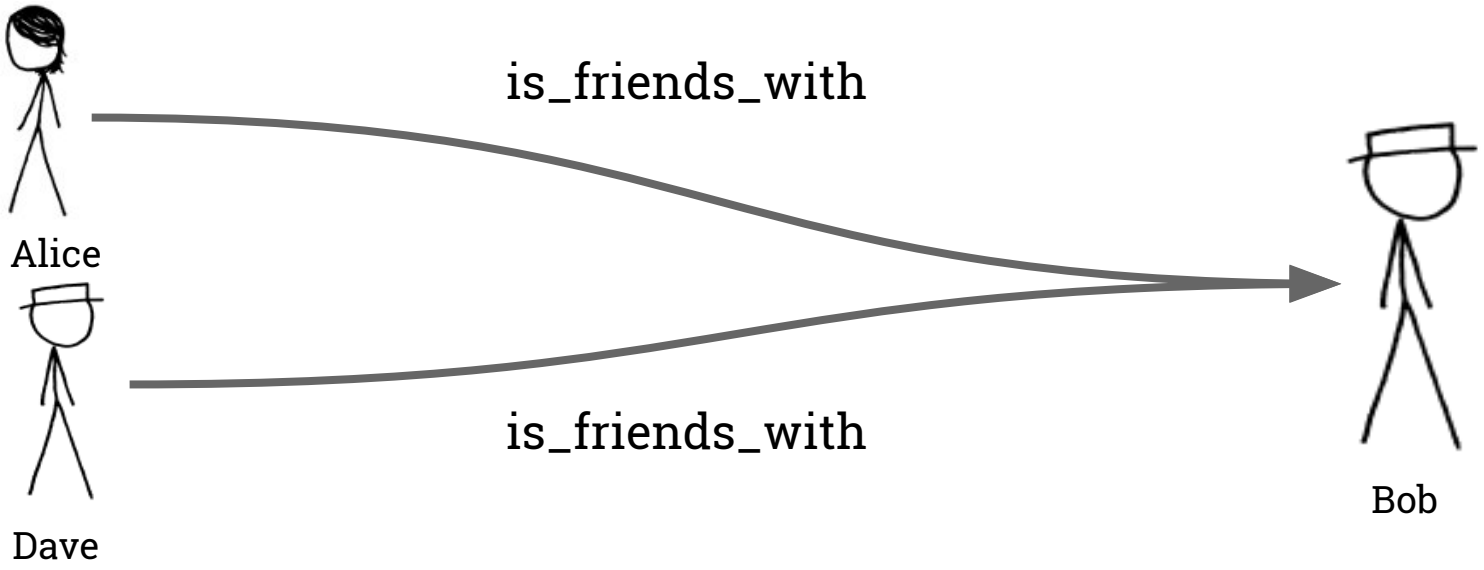
- Open-source the novel parts of graphd
- Use Go's amazing standard library
- At least graphd scale (385m primitives pre-Google)
- Promote the graph paradigm
- Agnostic enough for the future
  - Backends
  - Query languages

3.

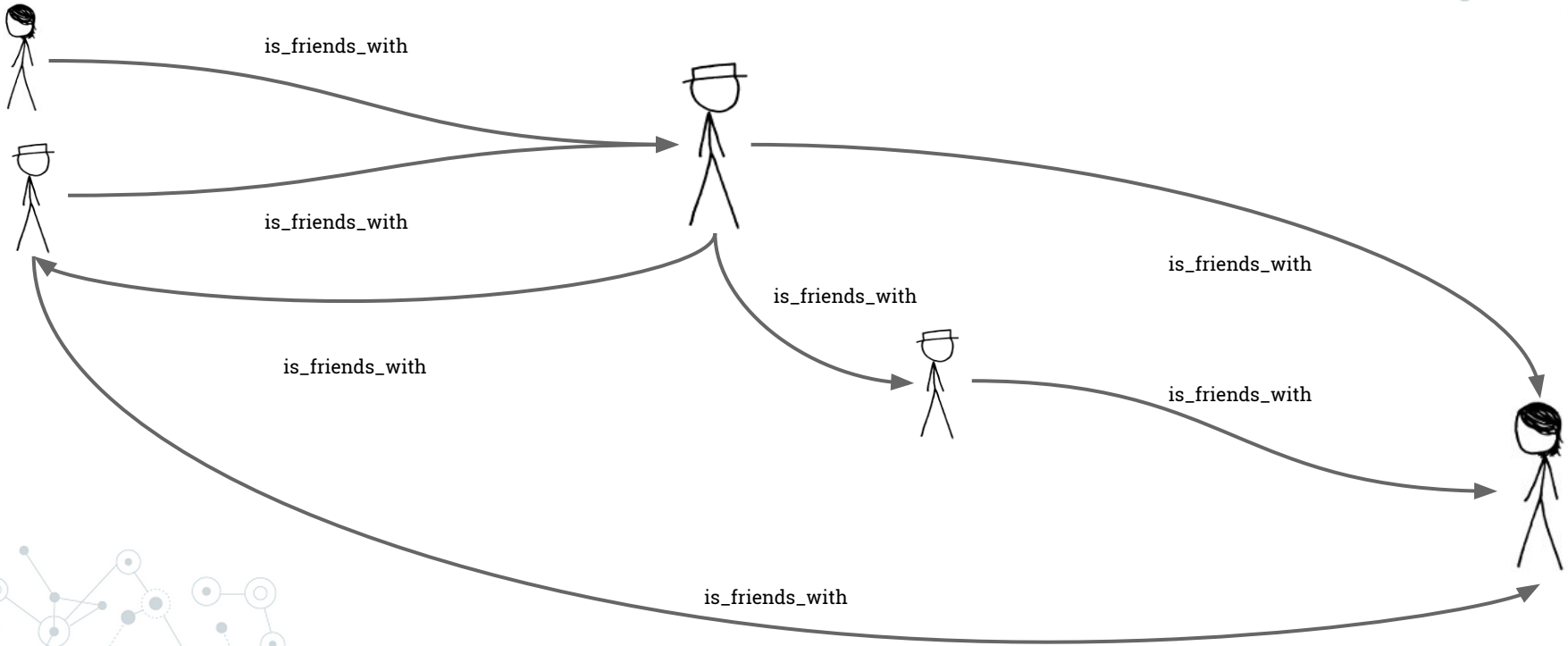
Cayley Architecture

Let’s dig into the nuts and bolts

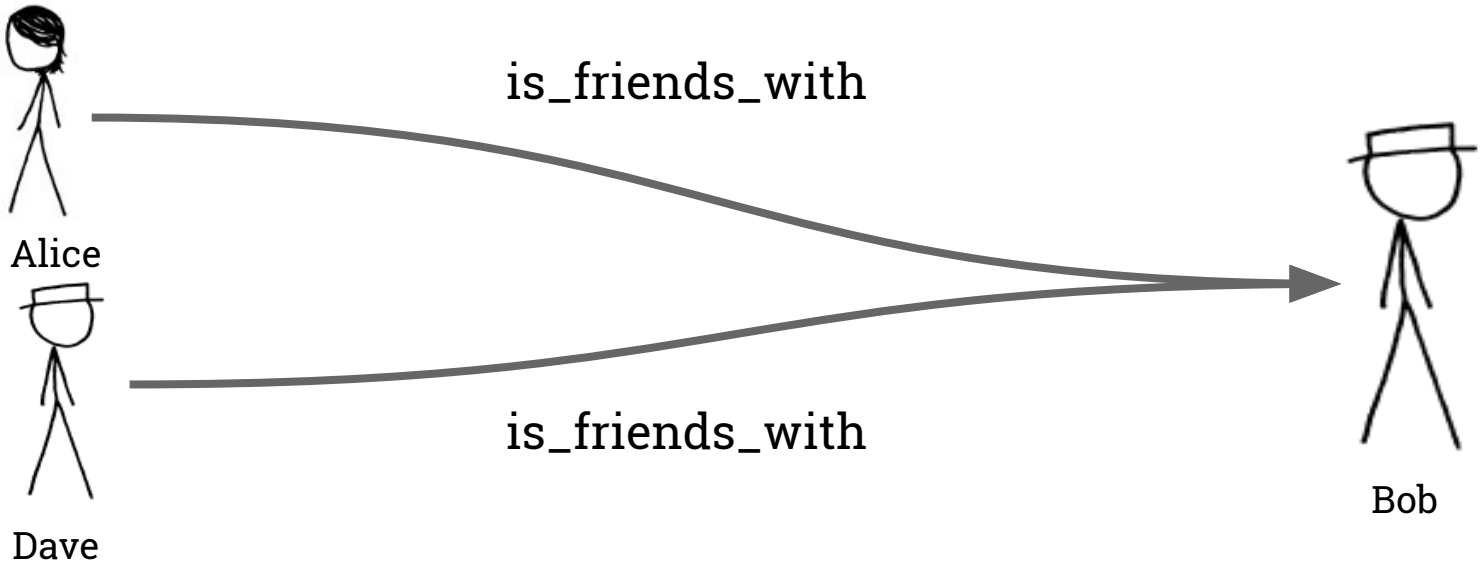
Running Example:



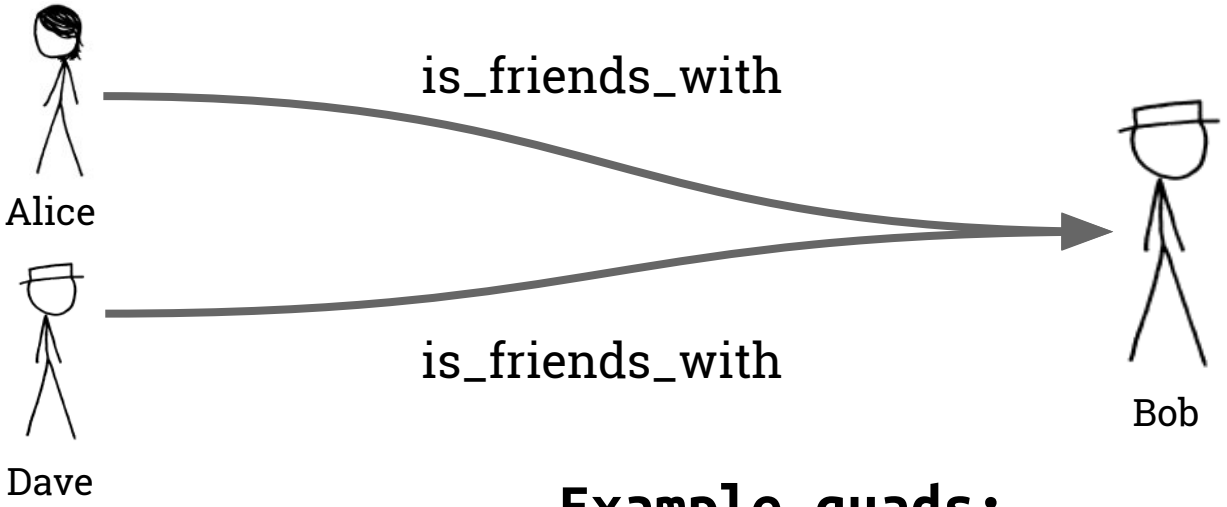
Running Example:



Running Example:



Quads:



Example.quads:

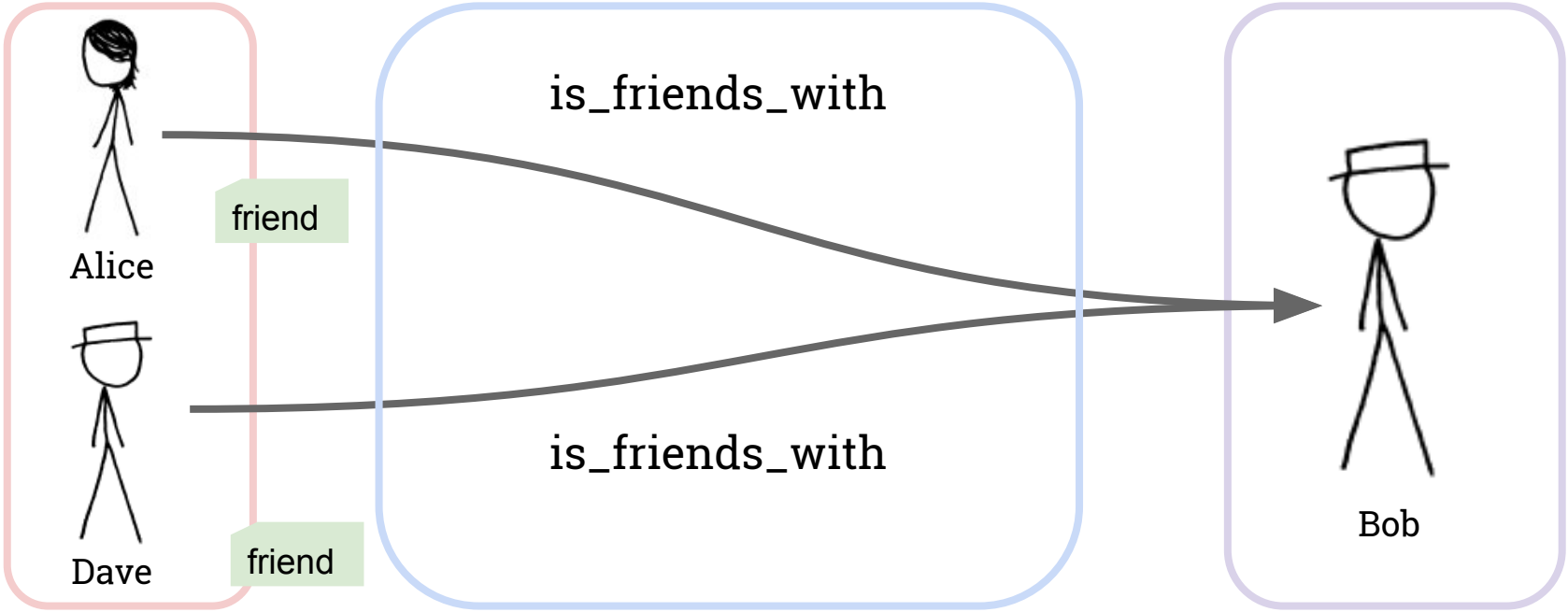
```
Alice is_friends_with Bob .
Dave is_friends_with Bob .
```

Quads:

Example.quads:

```
Alice is_friends_with Bob .
Dave is_friends_with Bob .
```

Space delimited, fourth field optional (and ignore it for now)




```
graph.Vertex().Tag("friend").Out("is_friends_with").Is("Bob")
```

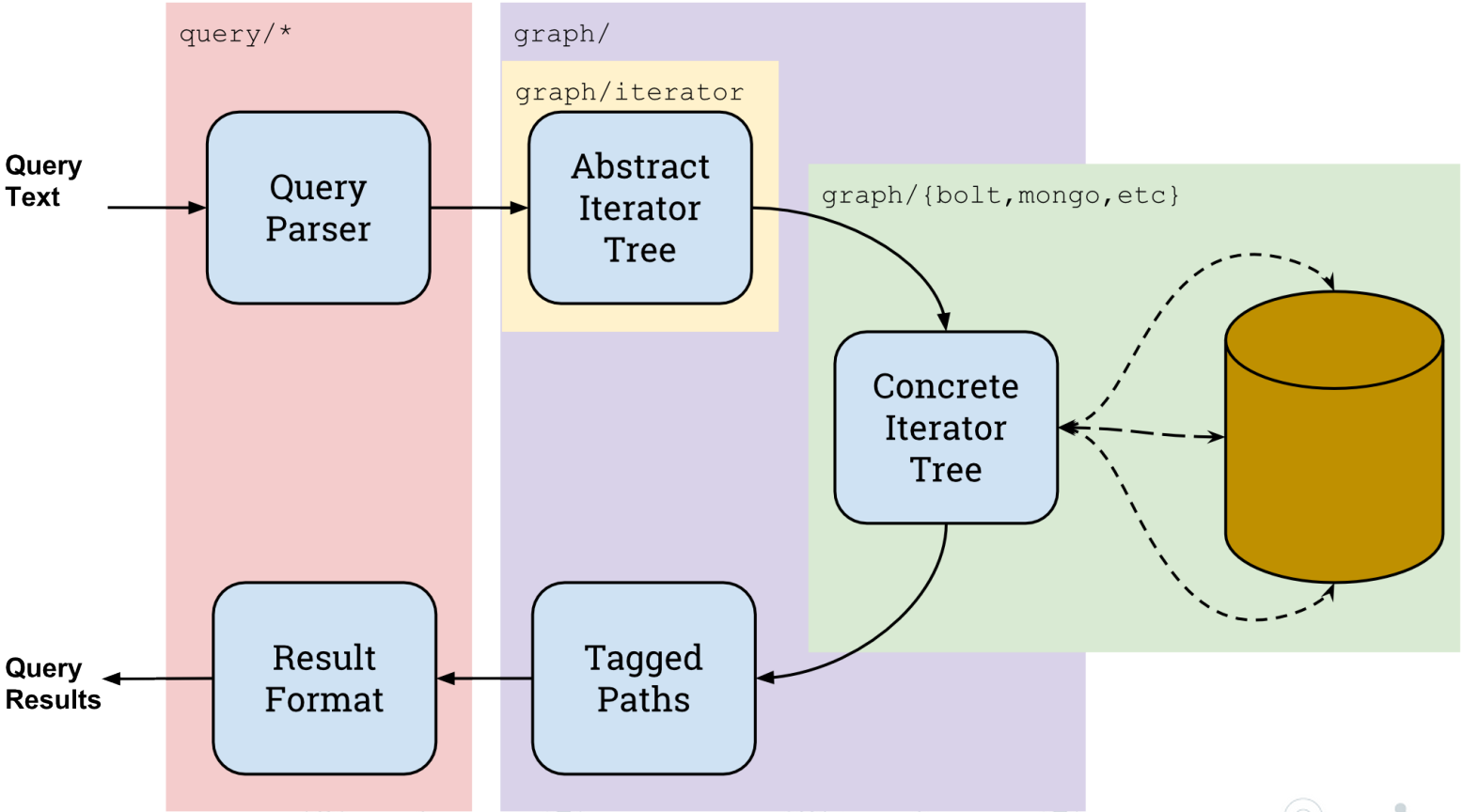
Input:

```
graph.Vertex()
   .Tag(“friend”)
   .Out(“is_friends_with”)
   .Is(“Bob”)
   .All()
```

Output:

```
{
 "result": [
  {
   "friend": "Alice",
   "id": "Bob"
  },
  {
   "friend": "Dave",
   "id": "Bob"
  }
 ]
}
```

Life of a Graph Query



3a.

Query Languages

`query/*`

Role: Parser

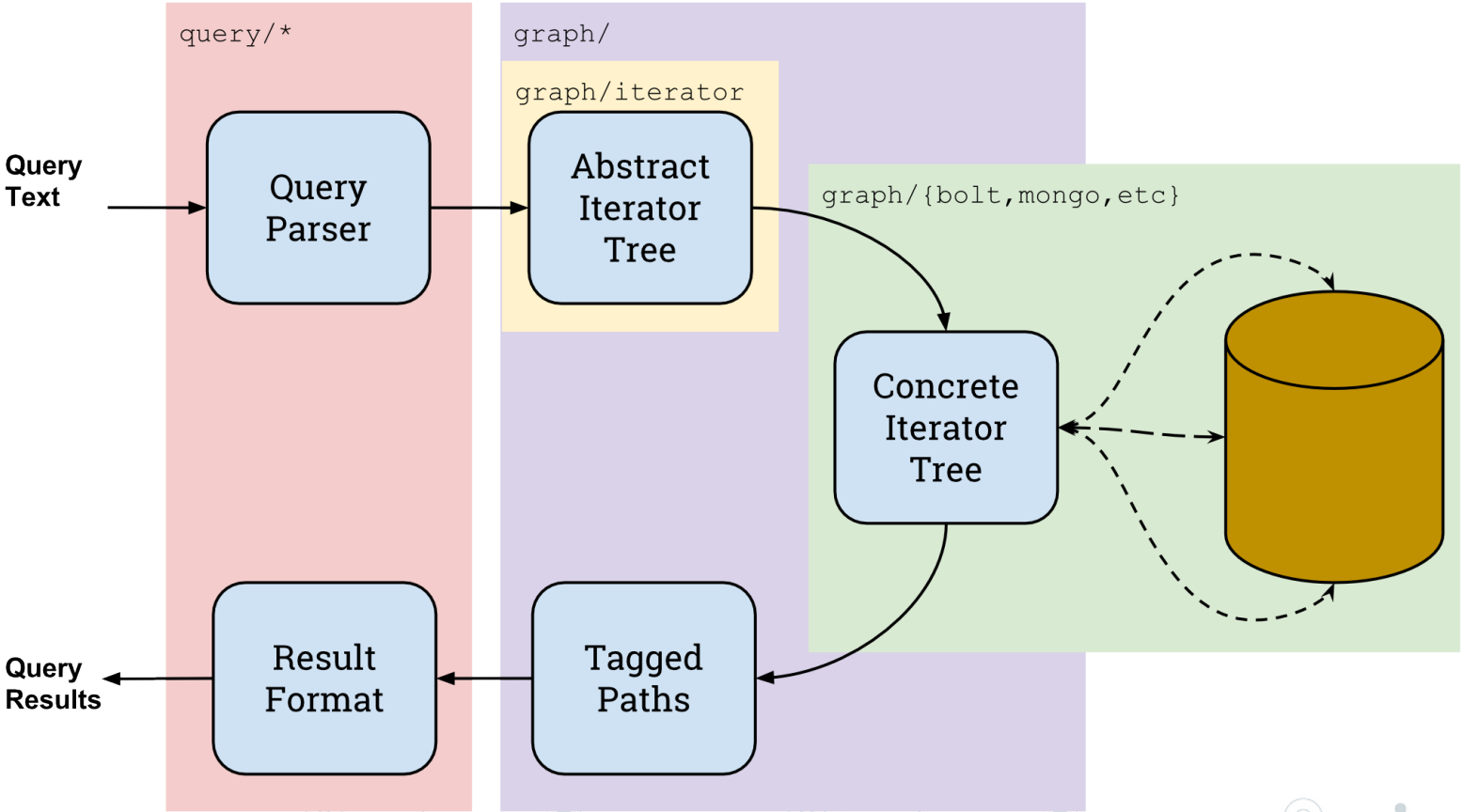
- Create a user session
- Build an abstract iterator tree based on user input.
- Evaluate this tree and return results

Primary Interface:

query/session.go:

```
// Does this even parse, really? Or do I need to keep reading?
Parse(query string) (ParseResult, error)

// Parse and evaluate, giving me paths back on the channel
Execute(query string, res chan interface{}, limit int)

// Stringify said results for the target audience
Format(interface{}) string
```

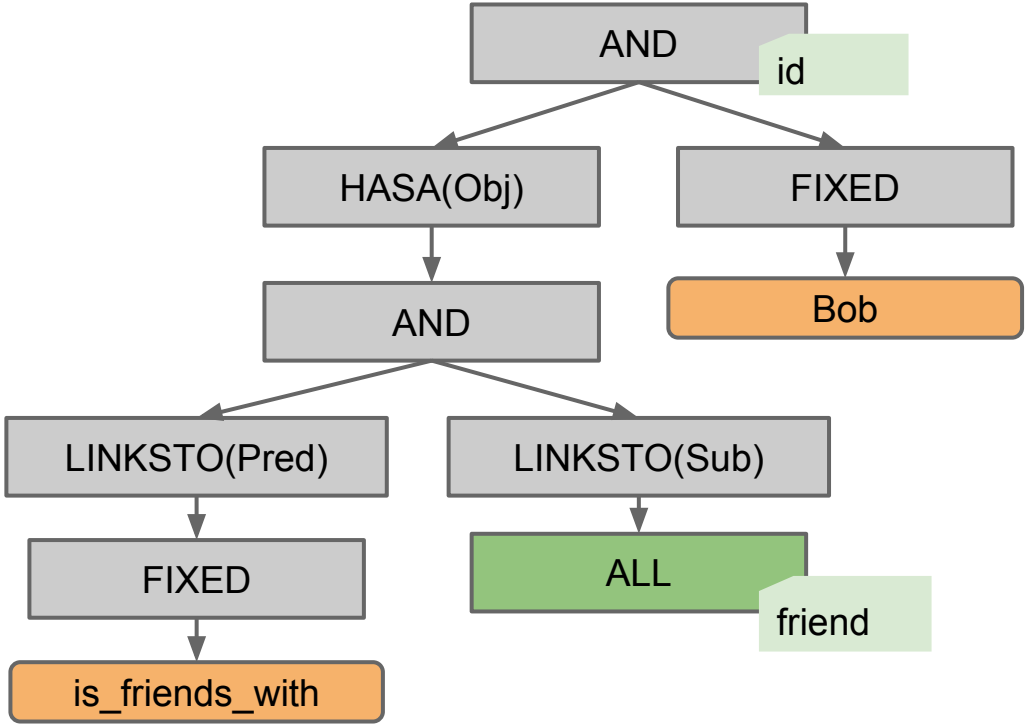
Life of a Graph Query



Input:

```
graph.Vertex()
   .Tag(“friend”)
   .Out(“is_friends_with”)
   .Is(“Bob”)
   .All()
```

Output:



3b.

Abstract Iterators

`graph/iterator`

Role: Represent Sets

- Placeholder for unevaluated sets of either nodes or quads (links)
- Precise set and map operations
- Be backend-independent (“abstract”)
- Hold tags

An iterator glossary

- AND
  - Intersection: *“In this set AND in this set”*
- OR
  - Union: *“In this set OR in this set”*
- FIXED
  - An explicit set in memory, often size 1: *“Exactly ‘A’”*
- ALL
  - All nodes in the database: *“All of it!”*

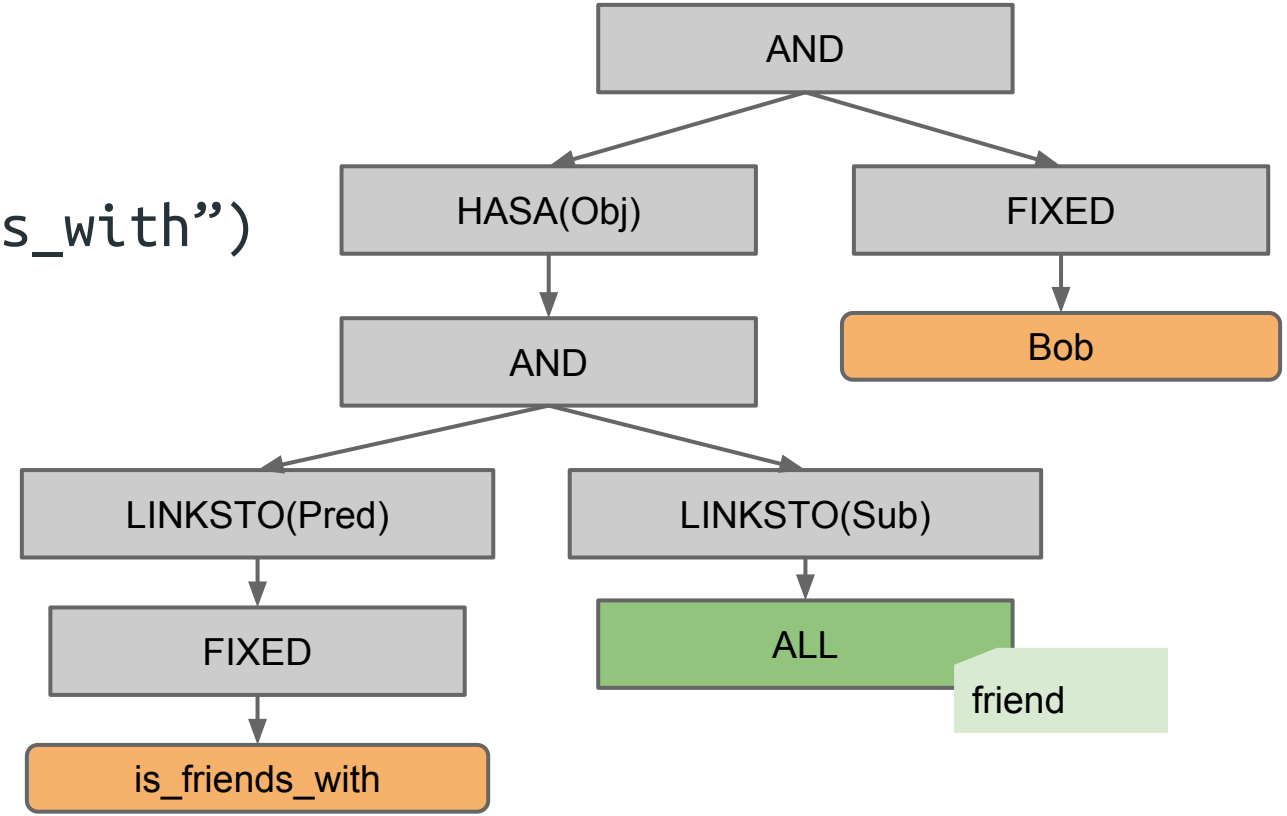
Iterator traversal duals

- LINKSTO
  - Set of Nodes -> Set of Links
- HASA
  - Set of Links -> Set of Nodes

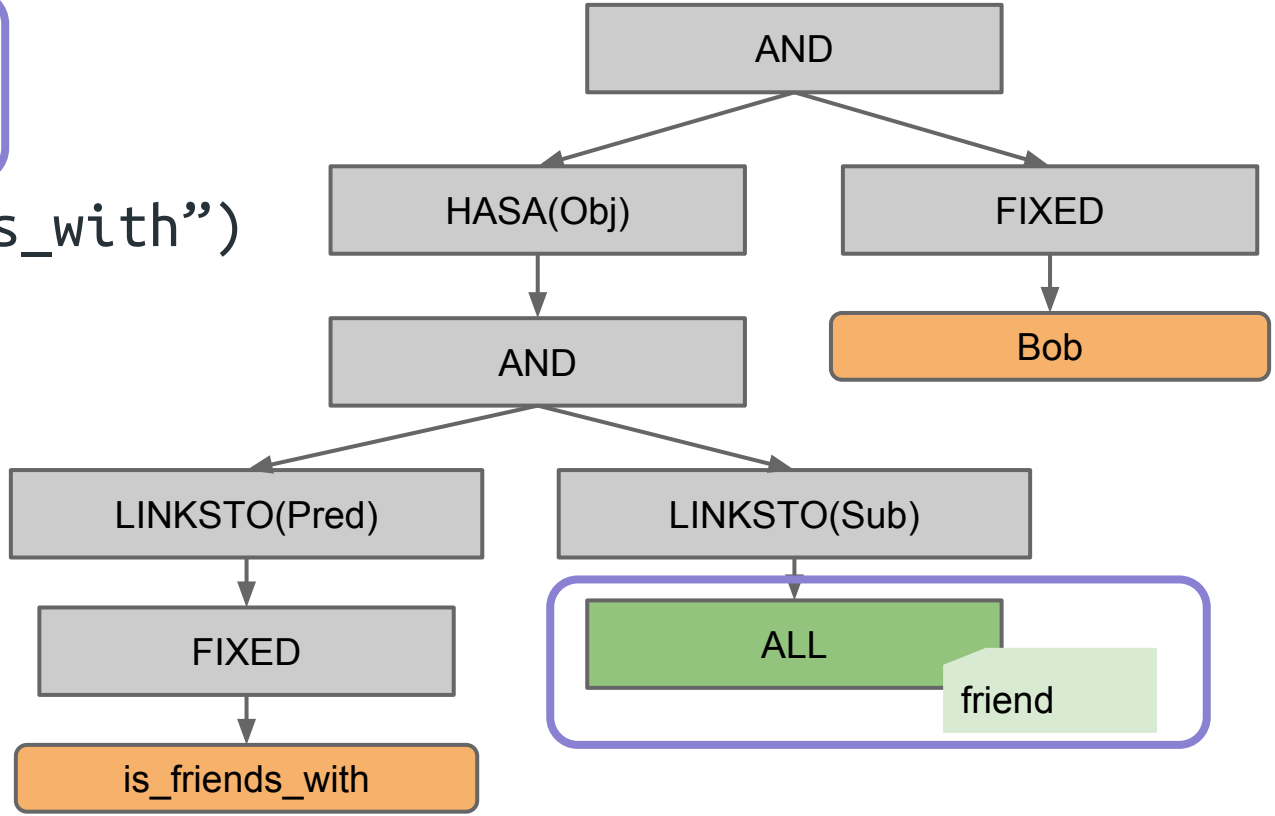
An iterator glossary

- LINKSTO(direction)
  - All quads that have a set of nodes in the direction field
  - *“All LINKS TO Alice where Alice is the subject”*
- HASA(direction)
  - All nodes given by the direction field of a set of quads.
  - *“This link HAS A object, which is Bob”*

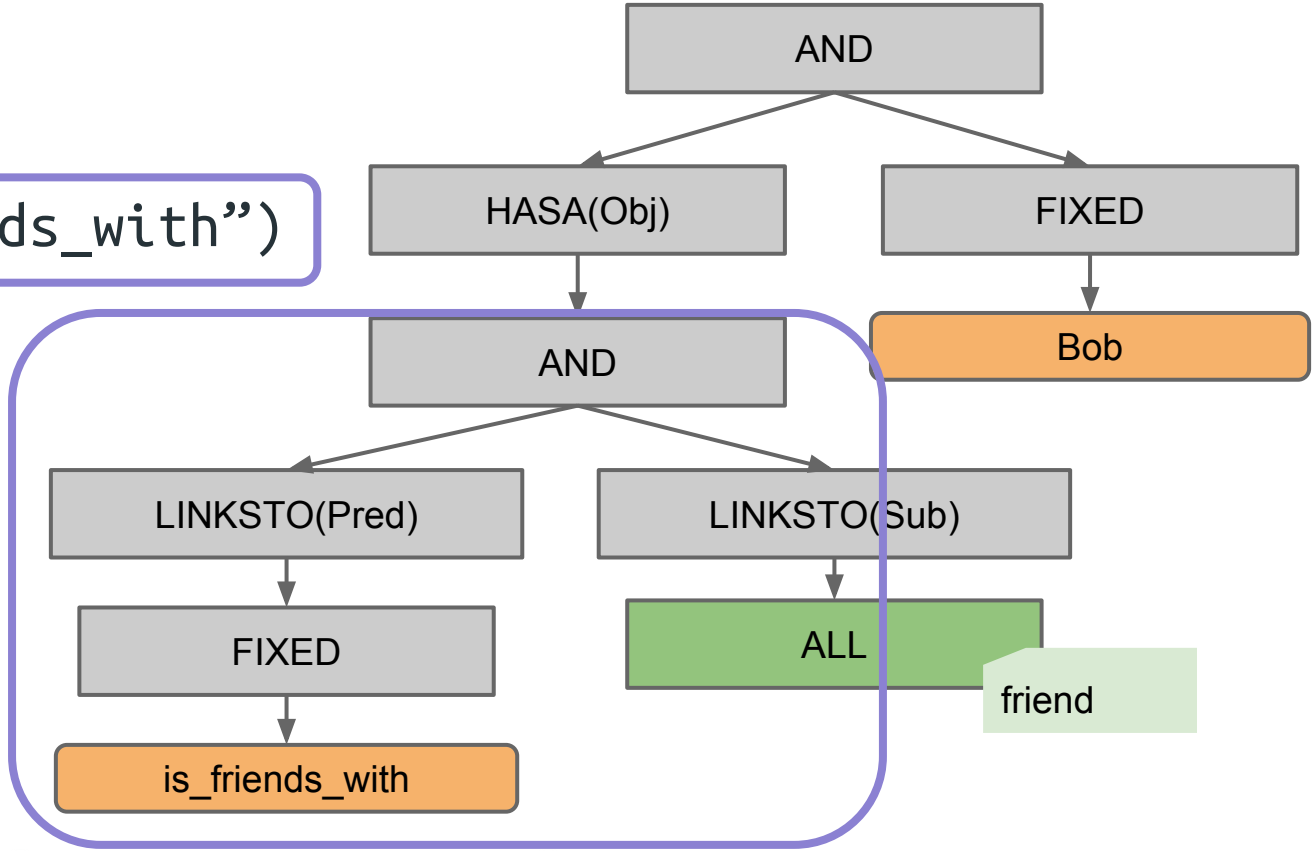
```
graph.Vertex()
    .Tag("friend")
    .Out("is_friends_with")
    .Is("Bob")
```



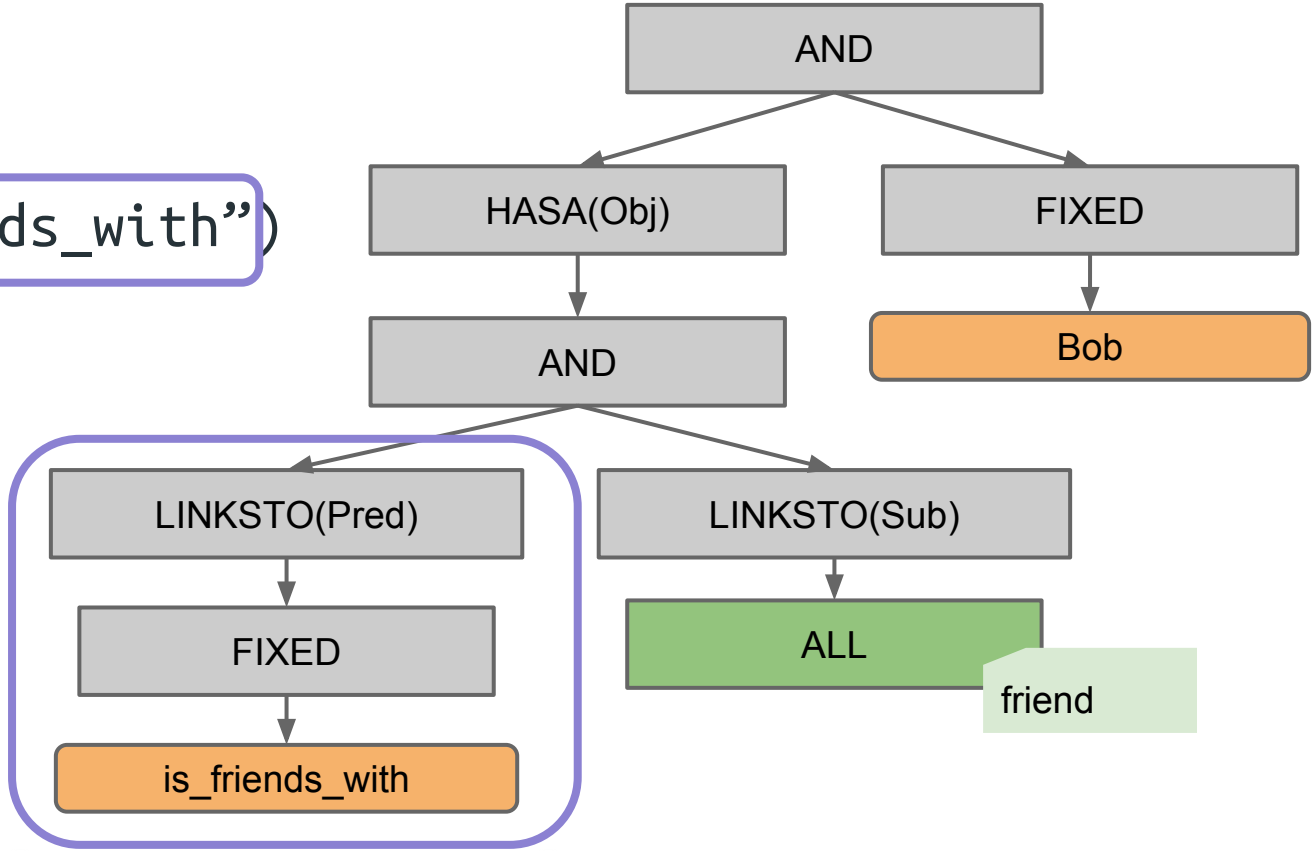
```
graph.Vertex()
    .Tag("friend")
    .Out("is_friends_with")
    .Is("Bob")
```



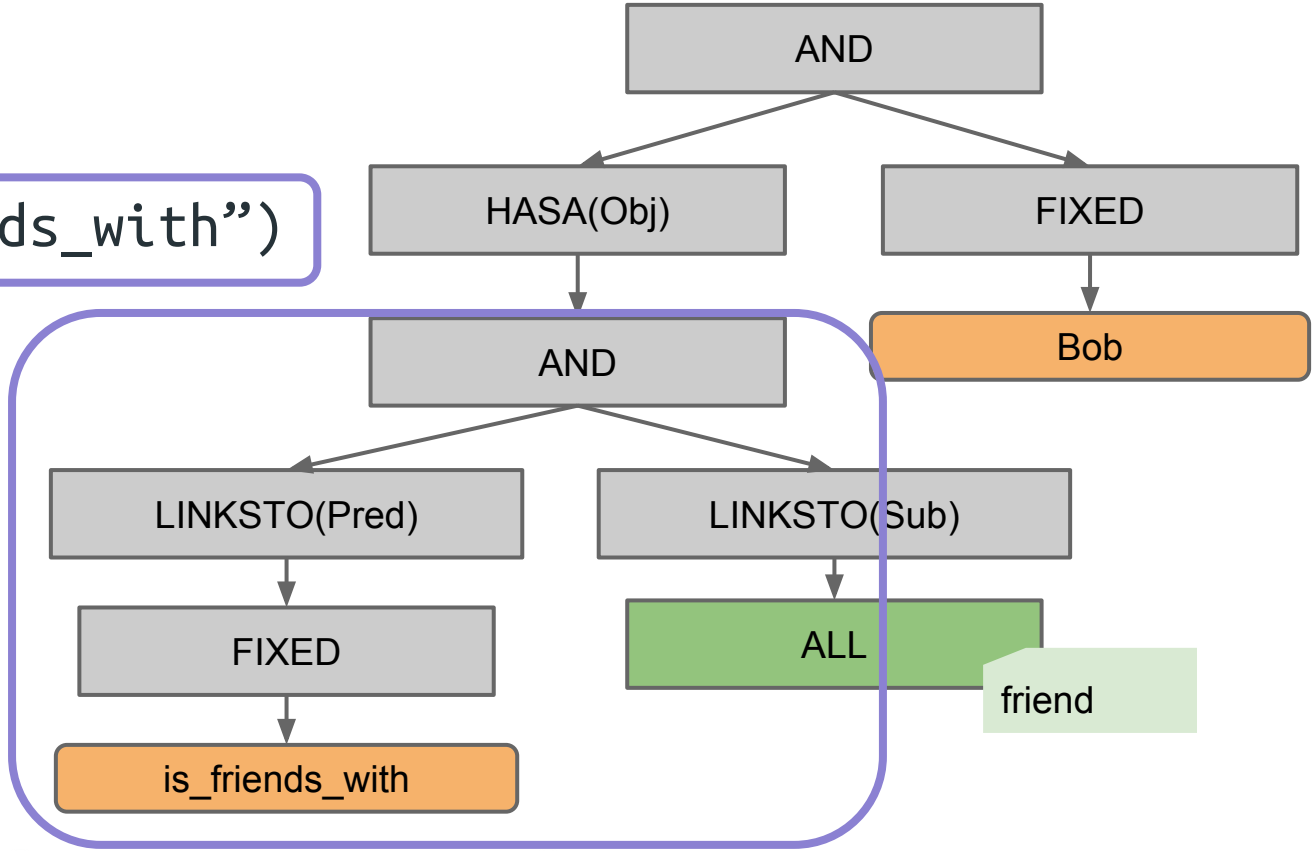
```
graph.Vertex()
    .Tag("friend")
    .Out("is_friends_with")
    .Is("Bob")
```



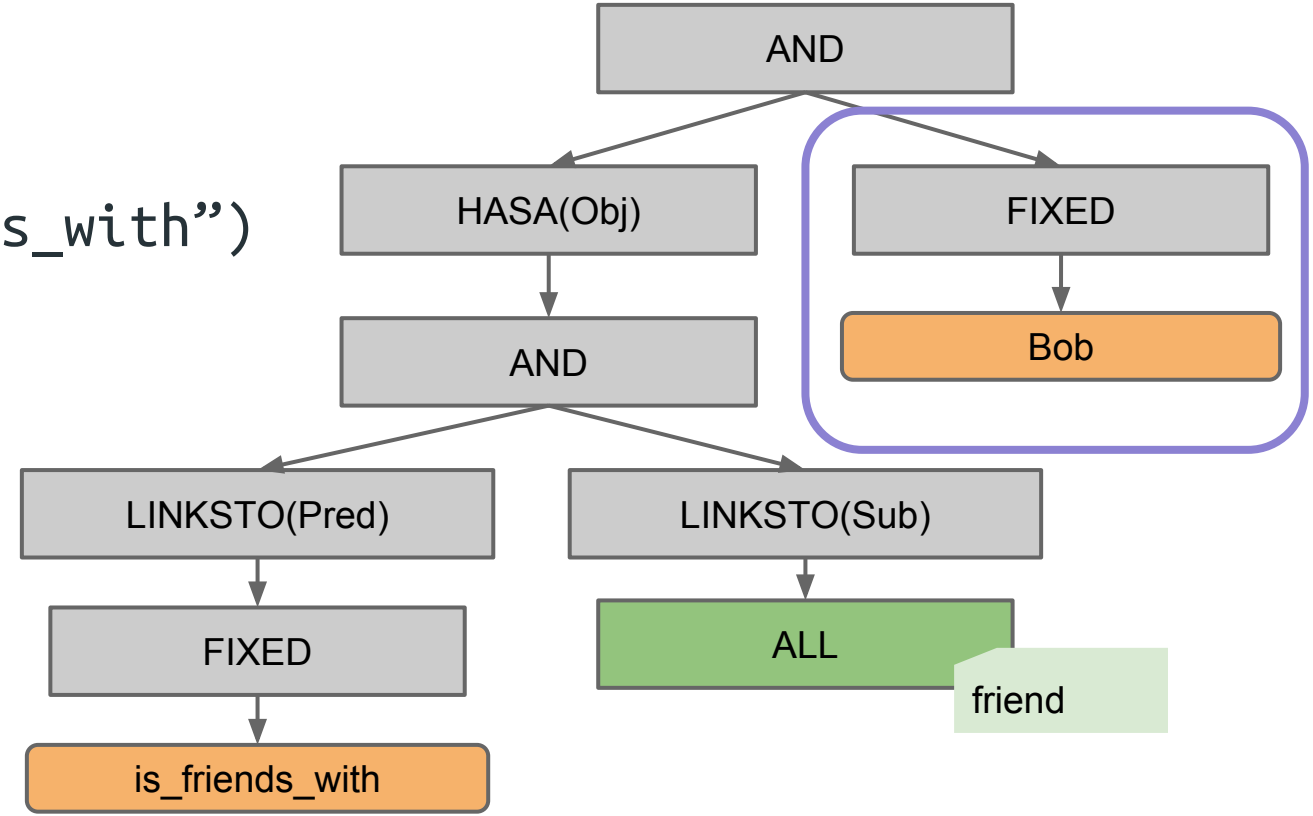
```
graph.Vertex()
    .Tag("friend")
    .Out("is_friends_with")
    .Is("Bob")
```



```
graph.Vertex()
    .Tag("friend")
    .Out("is_friends_with")
    .Is("Bob")
```



```
graph.Vertex()
    .Tag("friend")
    .Out("is_friends_with")
    .Is("Bob")
```

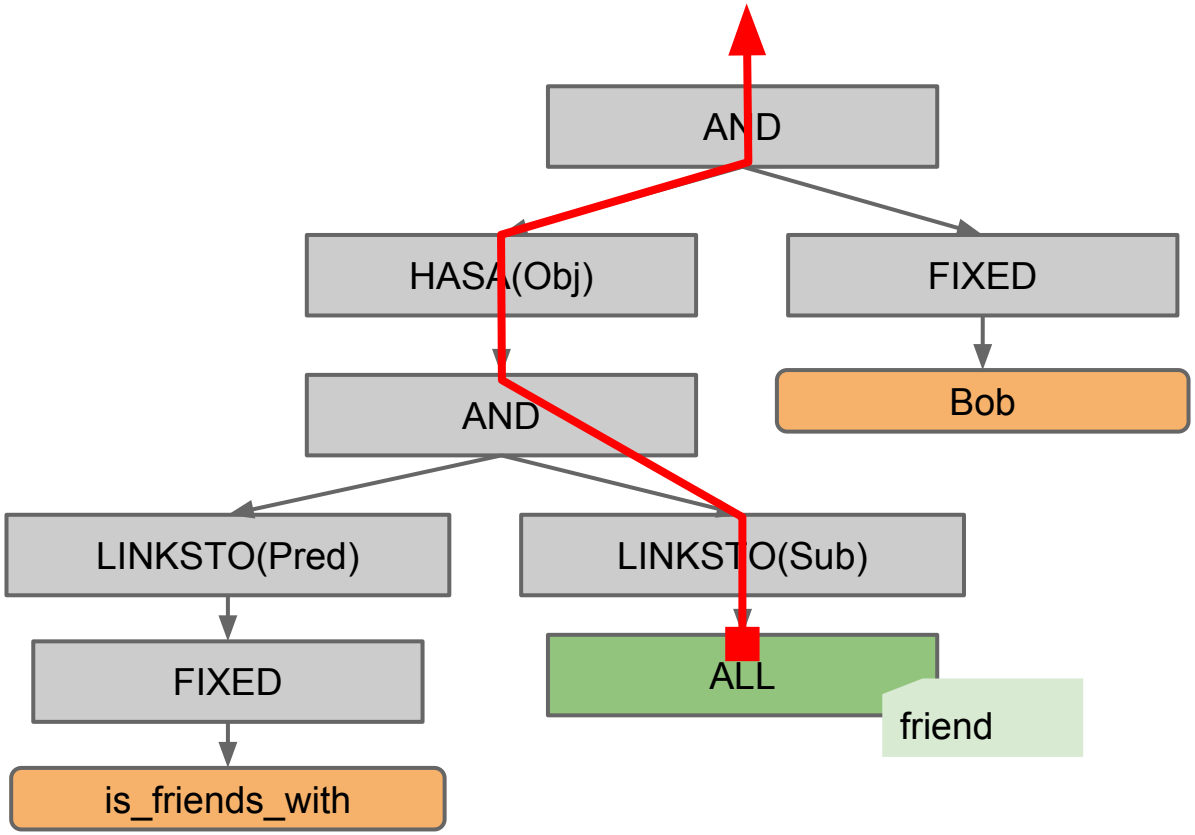


Using Iterators:

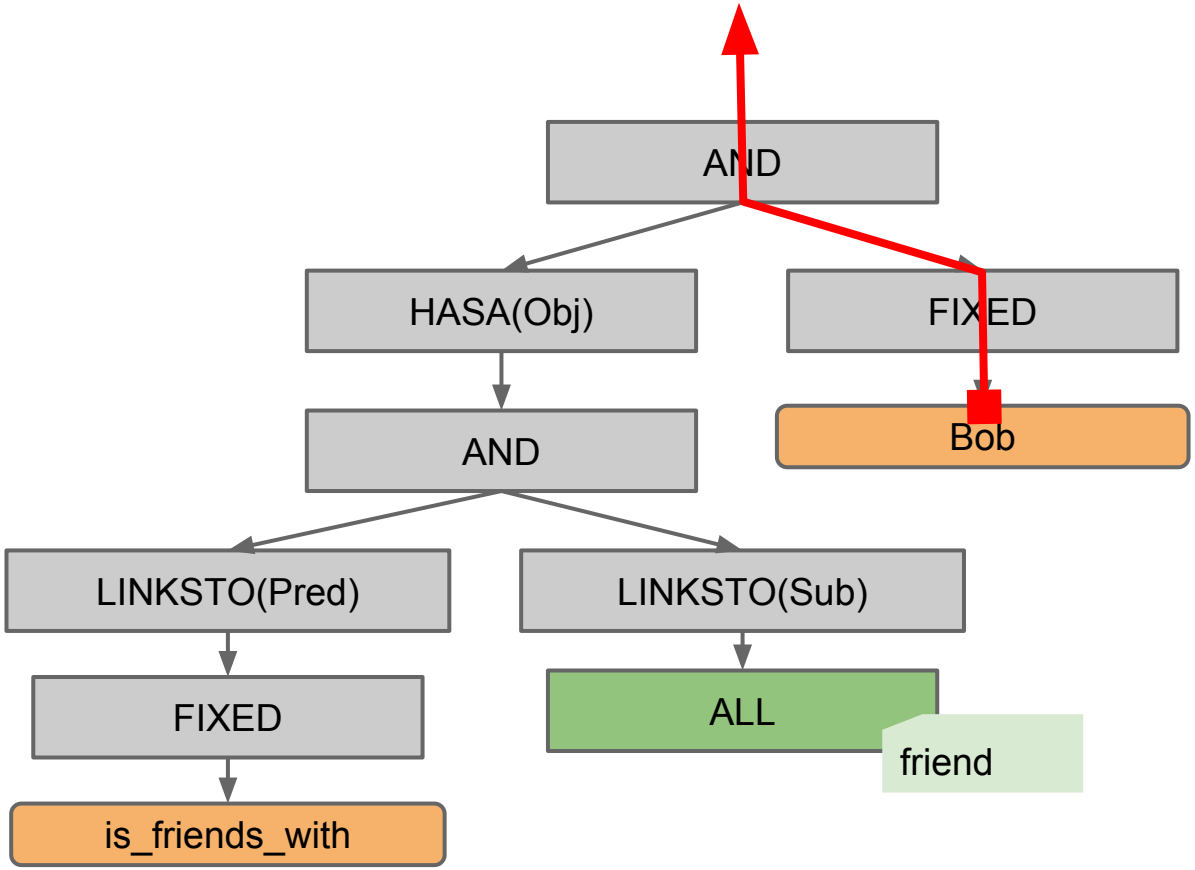
graph/iterator.go:

```
// Get the Value token at the current iterator position
Result() Value

// Move the iterator to the next result value
Next() bool

// Is there some other path in the tree that would
// give us the same Result()? If so, advance the subiterators.
NextPath() bool

// Does the set this iterator represents contain this token?
Contains(Value) bool
```

Avoid the naive path

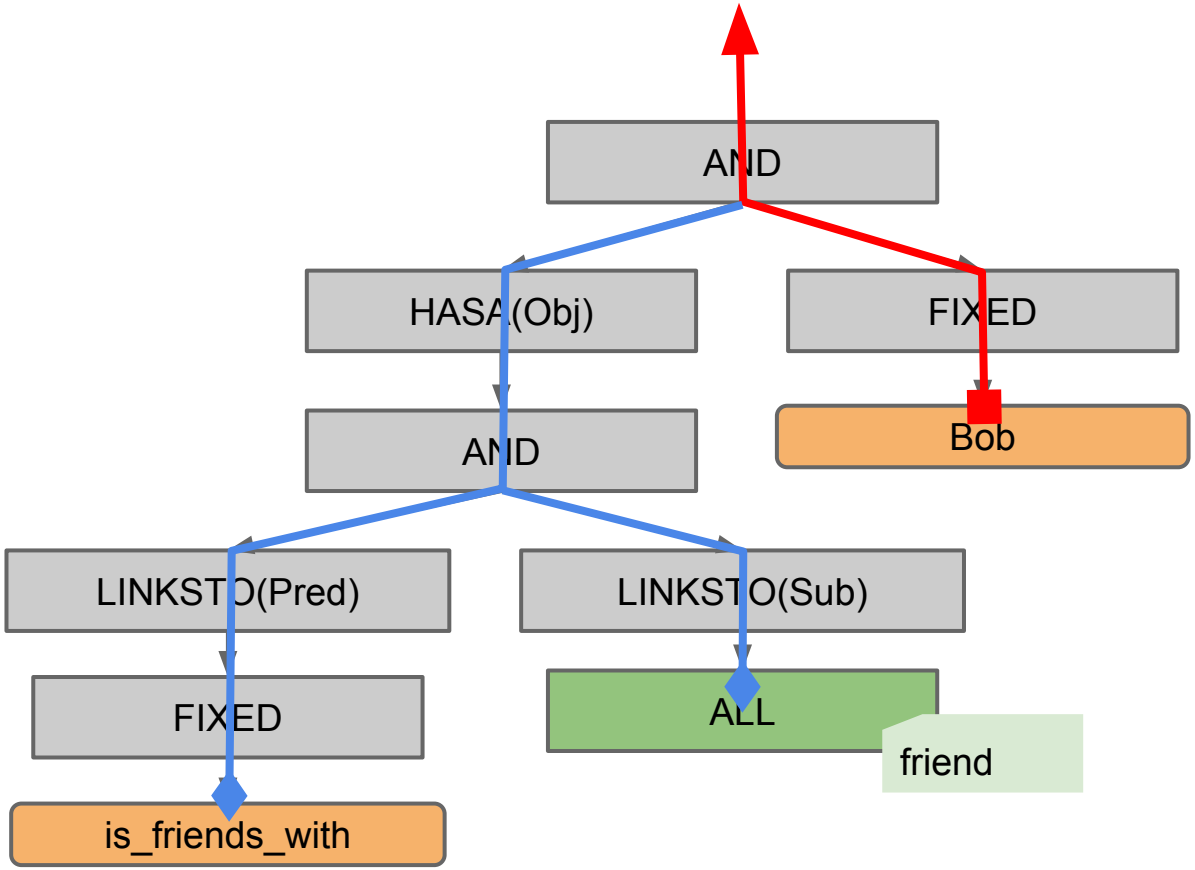


Next()
from the easy part...

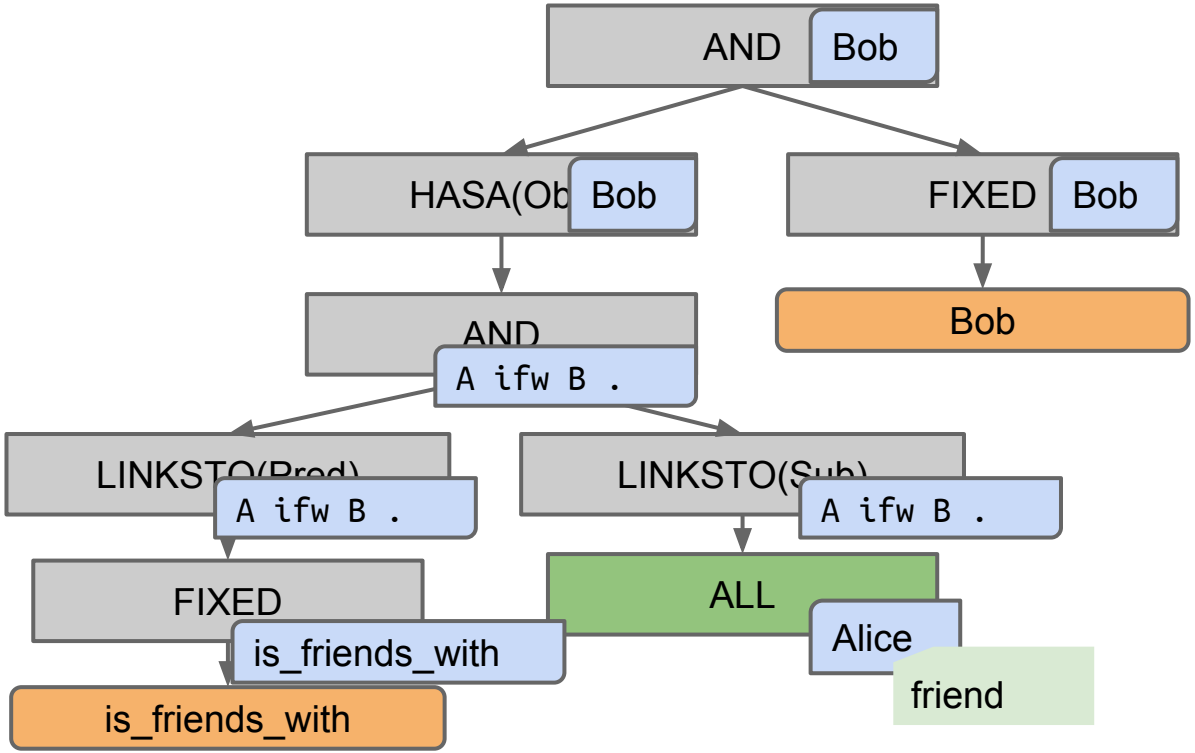


Contains()

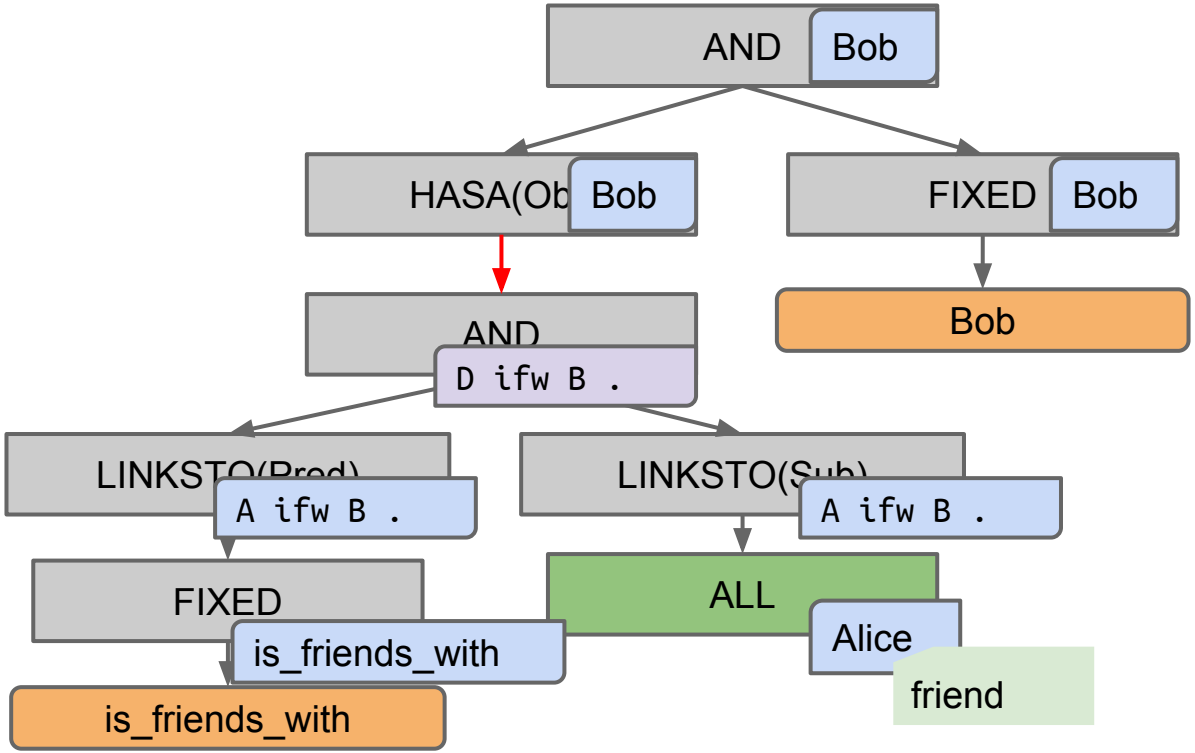
checks the rest



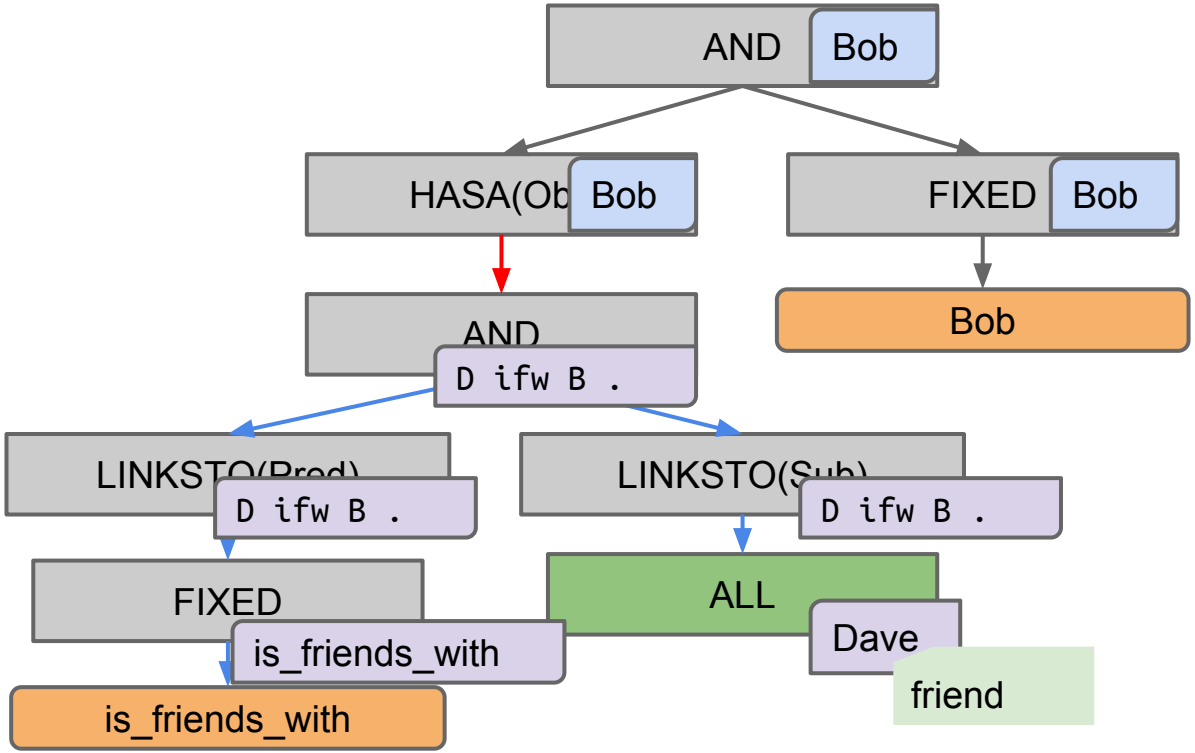
Result()


AND
Bob
HASA(Ob
Bob
FIXED
Bob
AND
A ifw B .
Bob
LINKSTO(Prod)
A ifw B .
LINKSTO(Sub)
A ifw B .
FIXED
is_friends_with
ALL
Alice
friend
is_friends_with

NextPath() to find the other friends



NextPath() to find the other friends



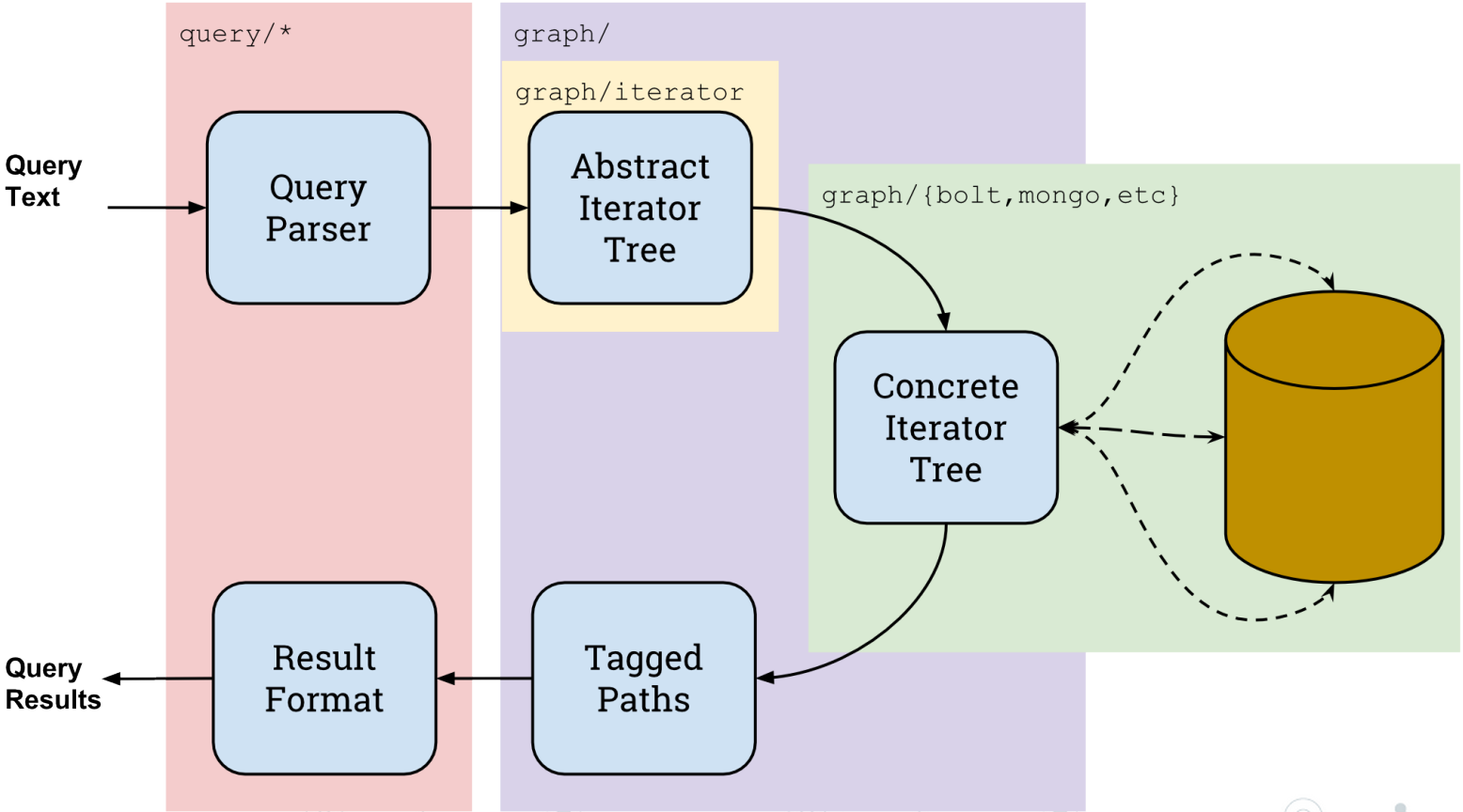
3c.

Quadstores and Concrete Iterators

aka, Backends

`graph/{bolt,mongo,leveldb,memstore}`

Life of a Graph Query



Role: Store the Data

- Translate quad operations to the particular backend
- Translate opaque database identifiers
- Optimize the query based on backend features

Most translation happens in backend iterators

SQL

Key/Value

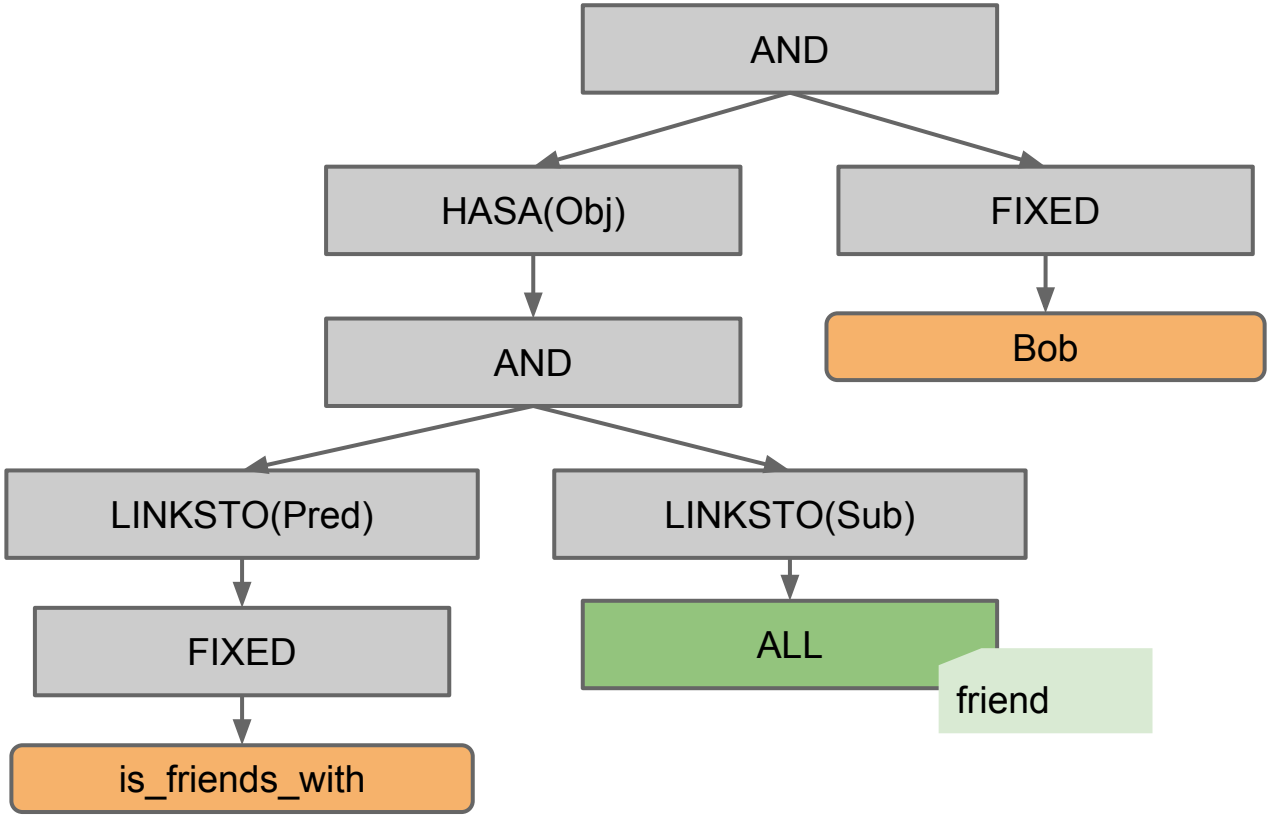
Documents

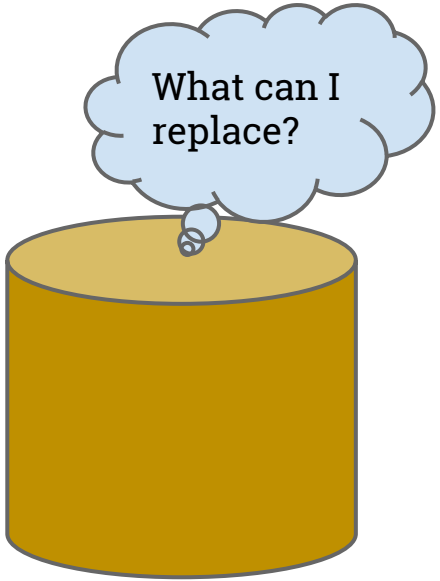
etc...

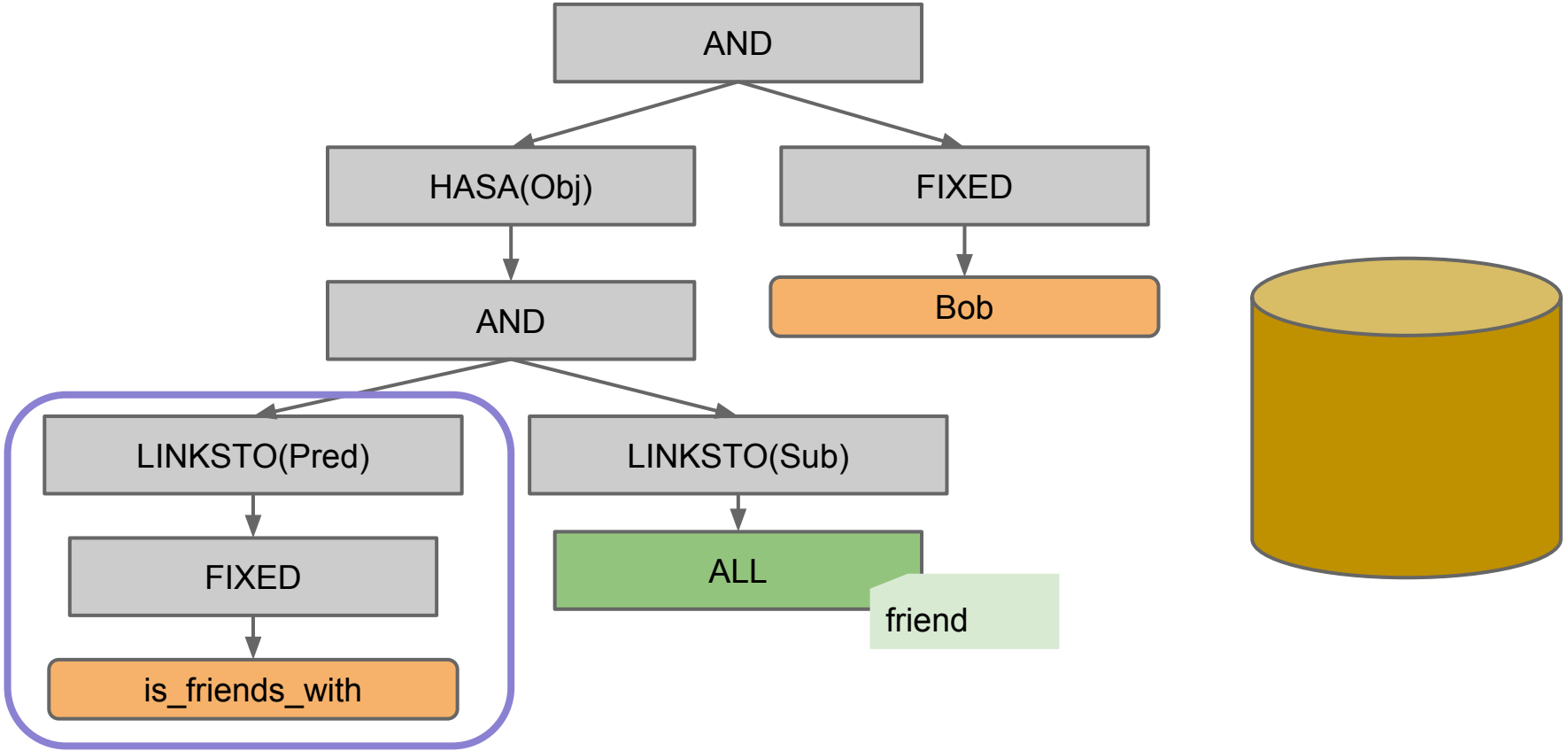
Primary Interface:

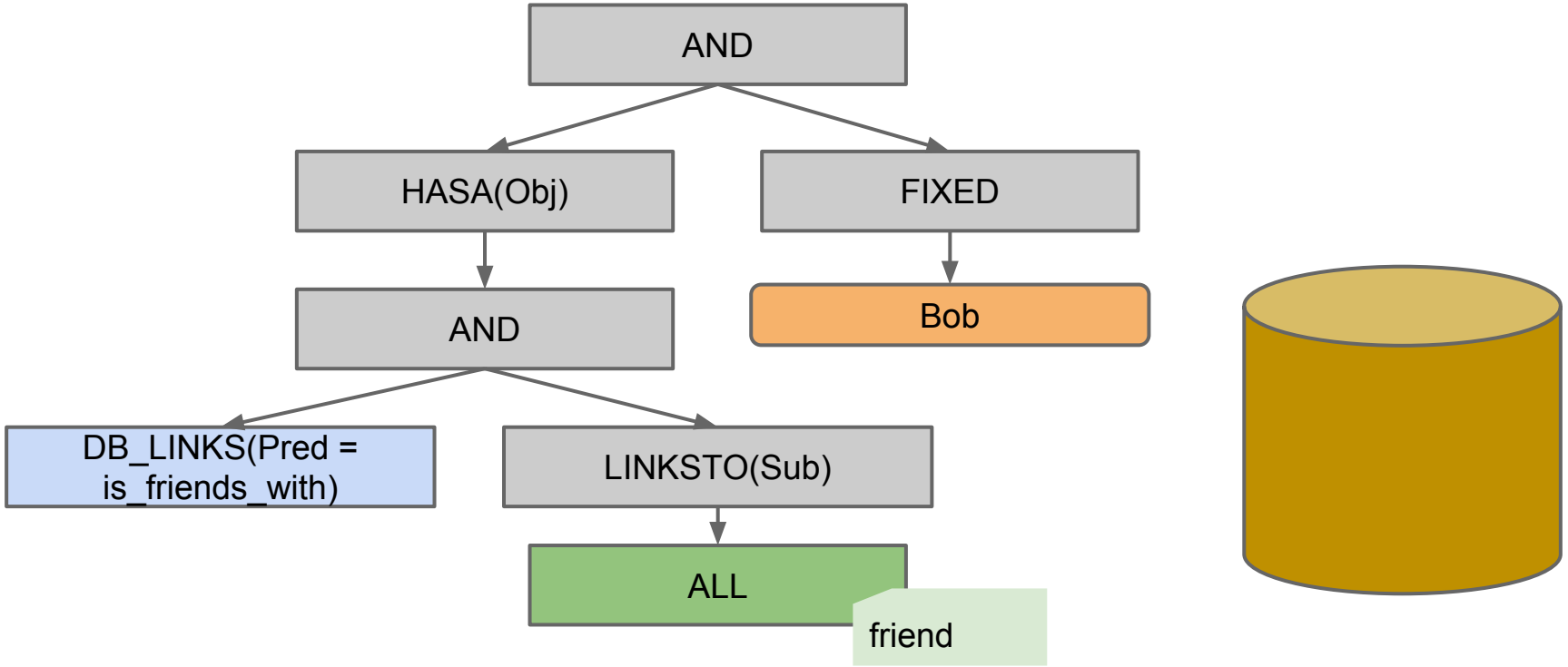
graph/quadstore.go:

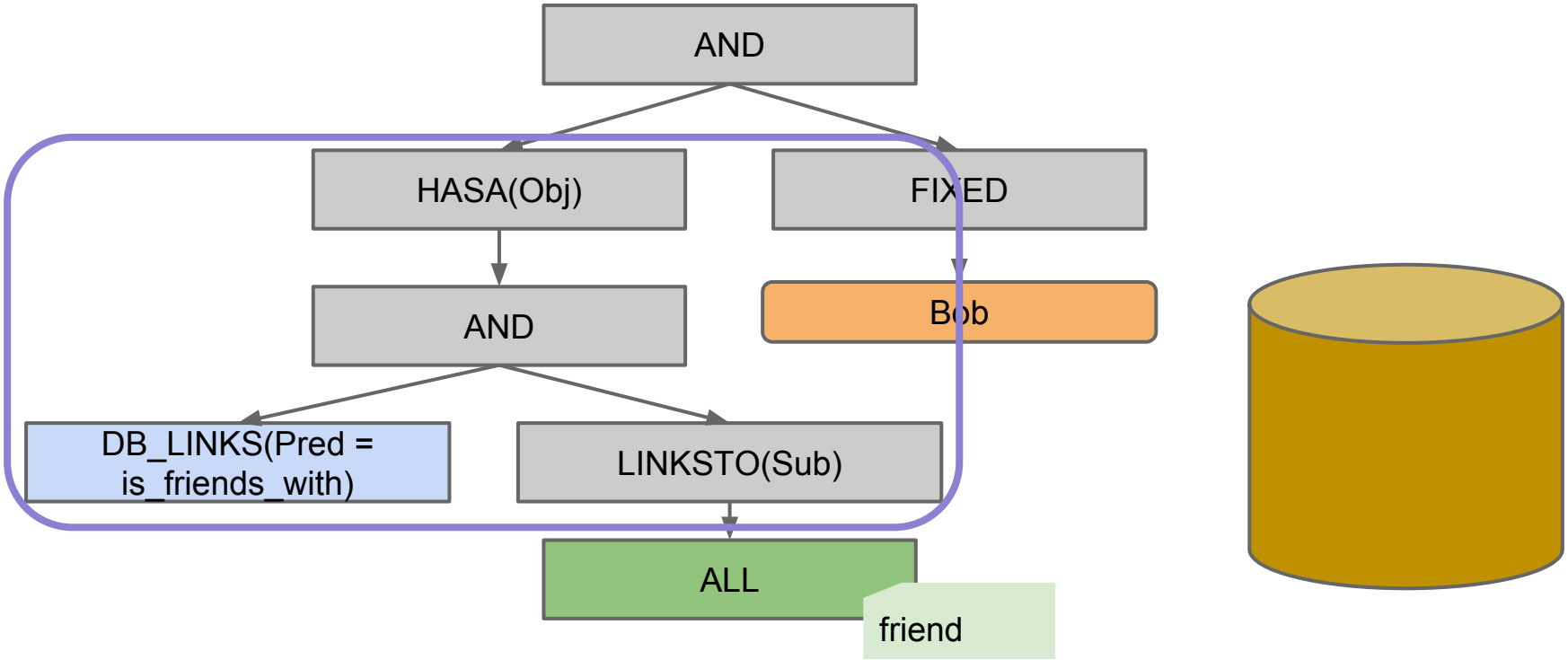
```
// Iterate links that have a given node token in a given field
// For use in HASA and LINKSTO
QuadIterator(quad.Direction, Value) Iterator

// Returns an ALL iterator for this particular datastore
NodesAllIterator() Iterator

// Make optimization decisions (via replacement)
OptimizeIterator(it Iterator) (Iterator, bool)

        // Translate the opaque values
        ValueOf(string) Value
        NameOf(Value) string
        Quad(Value) quad.Quad
```

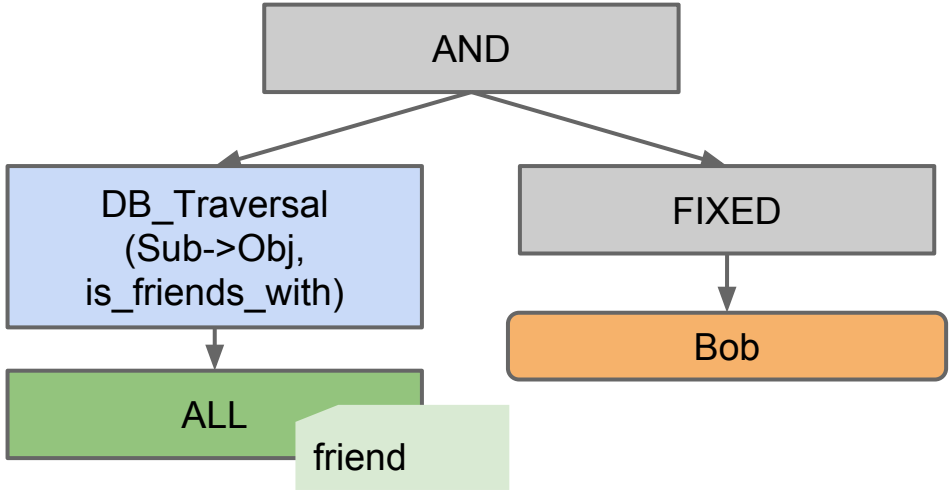

AND
HASA(Obj)
FIXED
AND
Bob
LINKSTO(Pred)
LINKSTO(Sub)
FIXED
ALL
friend
is_friends_with

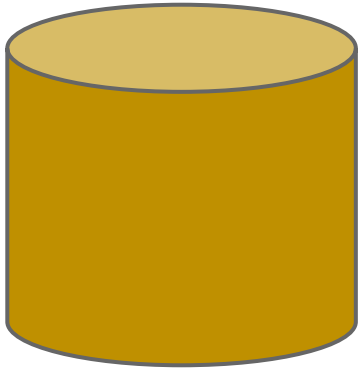

What can I replace?


AND
HASA(Obj)
FIXED
AND
Bob
LINKSTO(Pred)
LINKSTO(Sub)
FIXED
ALL
friend
is_friends_with


AND
HASA(Obj)
FIXED
AND
Bob
DB_LINKS(Pred = is_friends_with)
LINKSTO(Sub)
ALL
friend

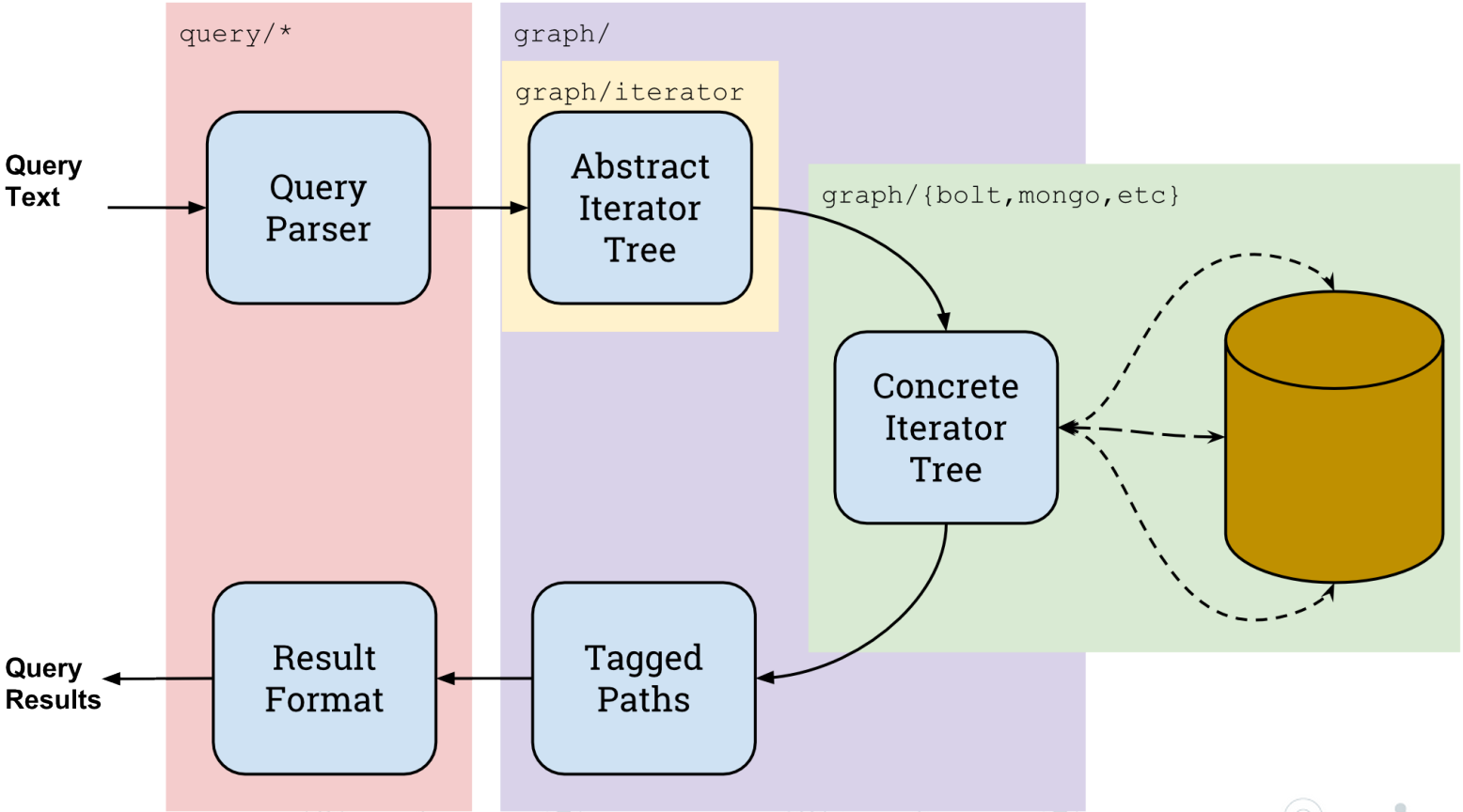

AND
HASA(Obj)
FIXED
AND
Bob
DB_LINKS(Pred = is_friends_with)
LINKSTO(Sub)
ALL
friend

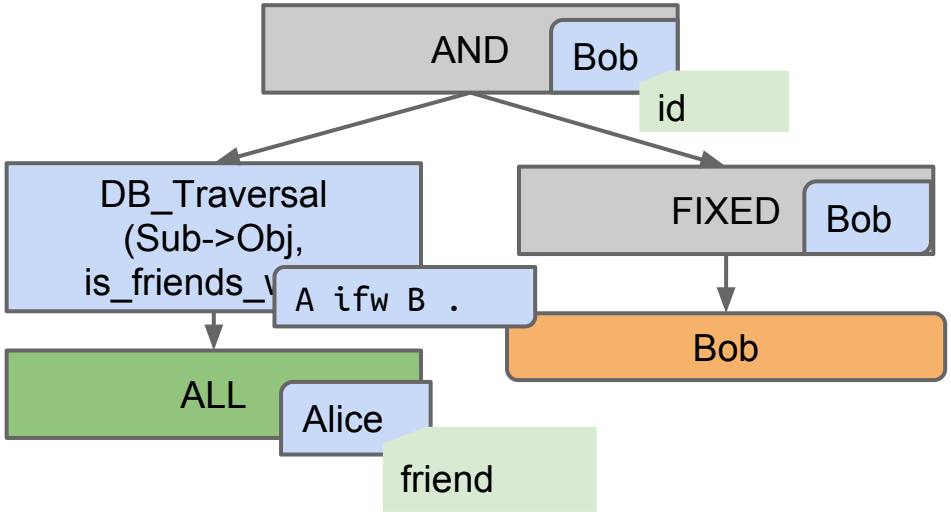




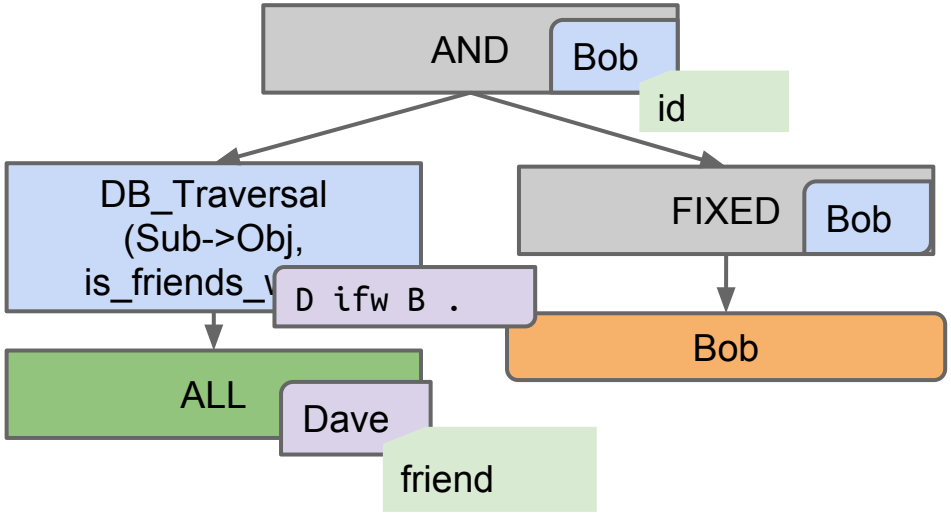
Concrete Iterator Tree

Life of a Graph Query






```
map[string]string{
   “id”: “Bob”,
   “friend”: “Alice”,
}
```




```
map[string]string{
  “id”: “Bob”,
  “friend”: “Dave”,
}
```

Input:

```
graph.Vertex()
  .Tag(“friend”)
  .Out(“is_friends_with”)
  .Is(“Bob”)
  .All()
```

Output:

```
{
 "result": [
  {
   "friend": "Alice",
   "id": "Bob"
  },
  {
   "friend": "Dave",
   "id": "Bob"
  }
 ]
}
```

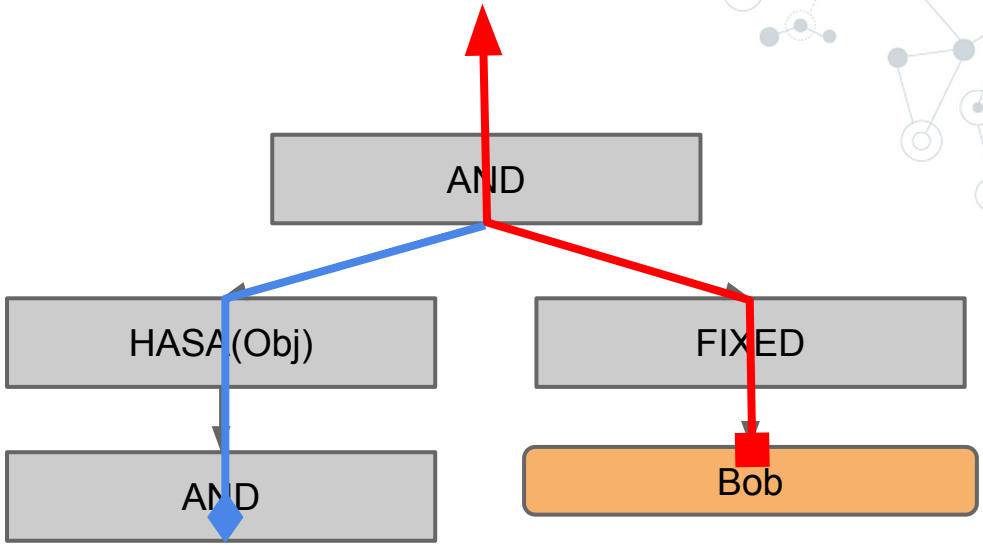
4.

Here there be dragons

Three interesting corners of the codebase

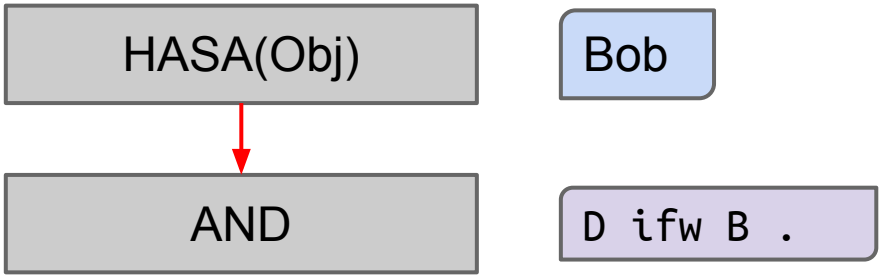
and_iterator_optimize.go

Choosing the best tree branches to extract with `Next()` or check with `Contains()`



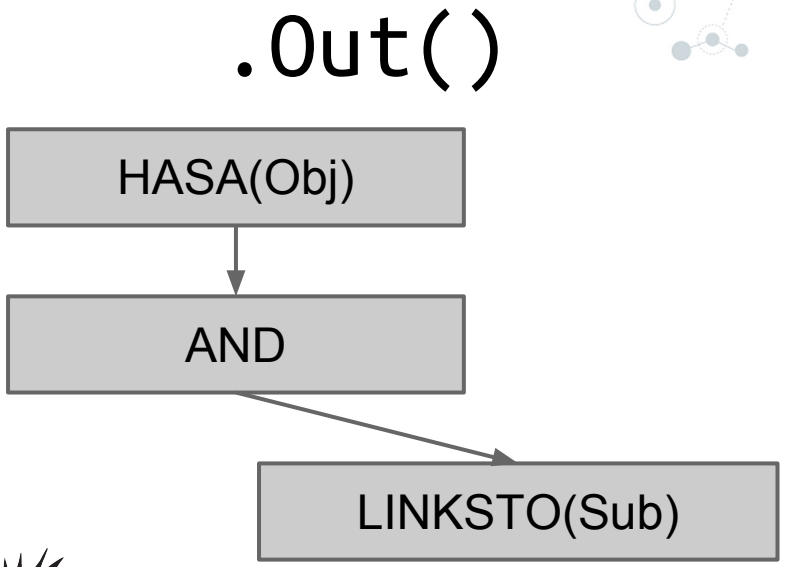
HasA.NextContains()

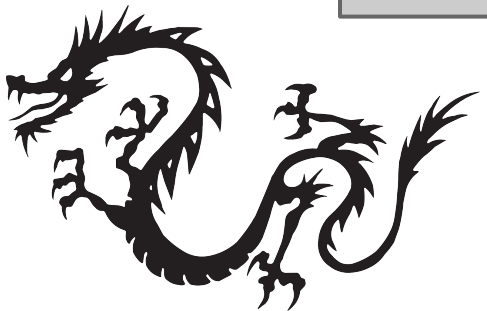
HASA iterators are the potential fan-in point; keeping track of alternate paths is important and subtle



buildIteratorTreeHelper()

Defines the tree patterns built for each of the various graph methods exposed in Javascript



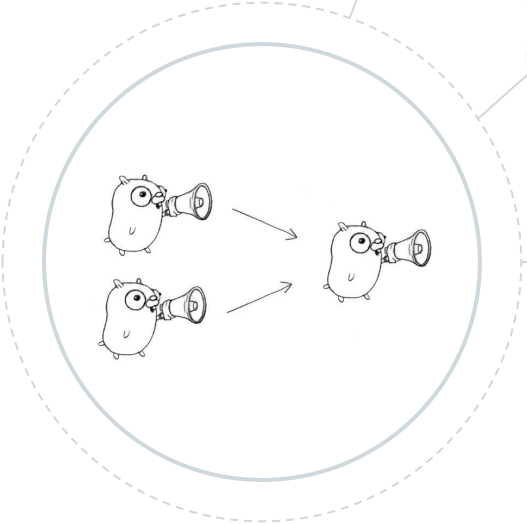


5.

Future Work

Building an even better graph

Parallel Queries



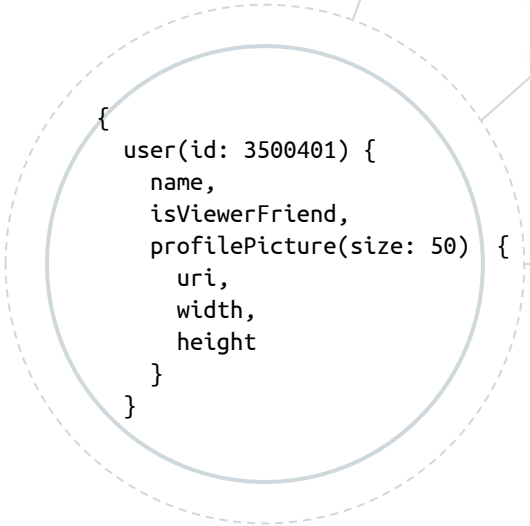
Evaluate two (reasonably sized) query branches simultaneously

“Perfect” Set Sizes

Precompute/estimate some cardinalities via HyperLogLog

New Query Languages

SPARQL, Facebook's GraphQL, etc.

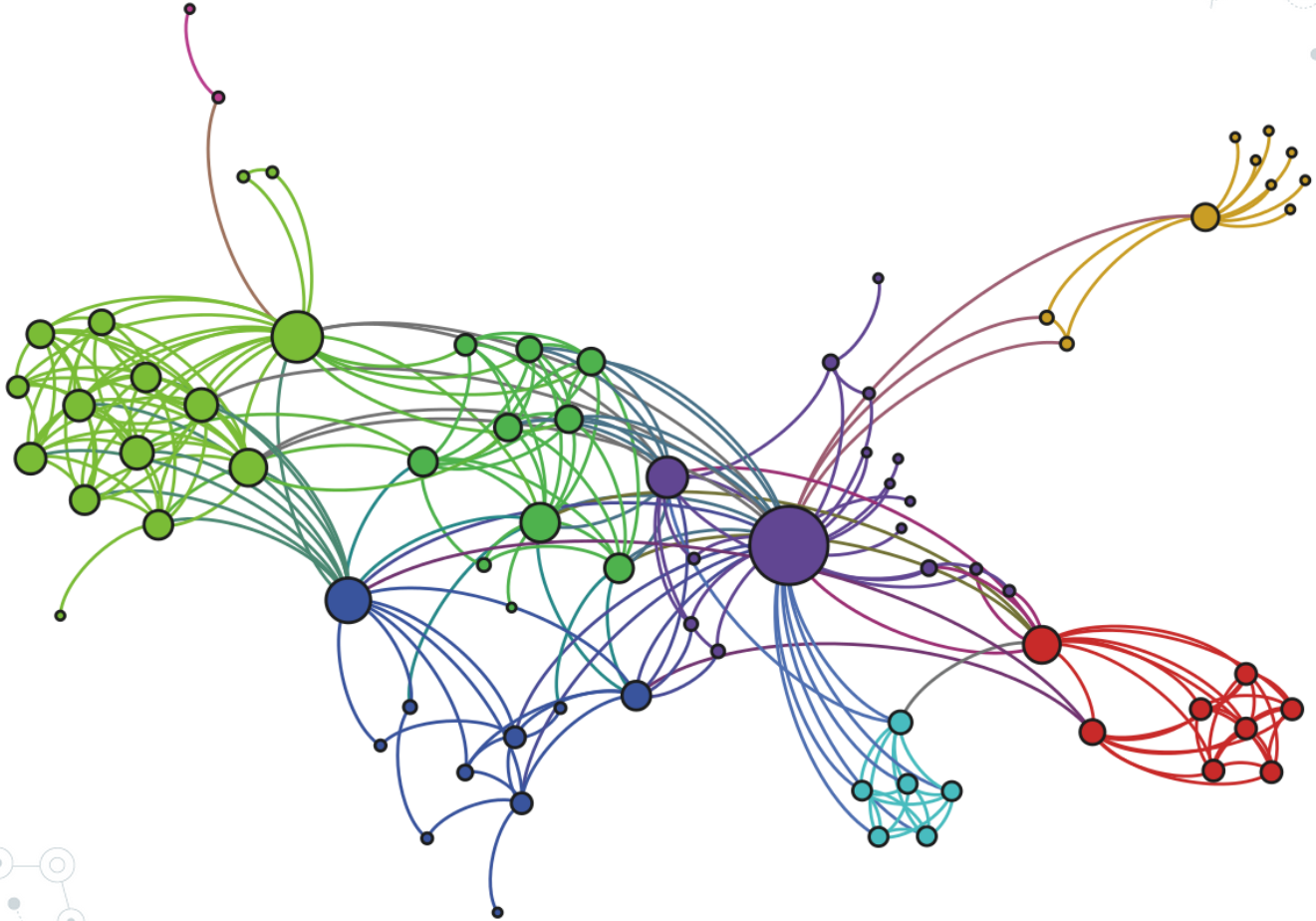


Optional Type Safety

Metadata links declared on predicates

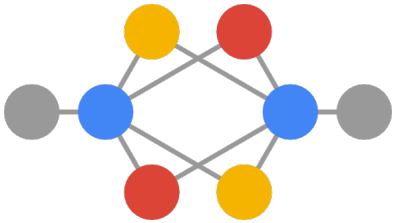
Many more useful things...

- Cassandra backend (PR for review)
- Full text indexing
- Graph-shaped iterators (generalizing trees)
- Single-master replication
- Longer term: experimental indexes and sharding



“Les Miserables” Social Graph

Questions?



github.com/google/cayley

@barakmich
@cayleygraph

Credits

Special thanks to all the people who made and released these awesome resources for free:

- Presentation template by SlidesCarnival
- Alice, Bob and Dave by Randall Munroe
- Kevin Bacon by SAGIndie (via Wikipedia)
- Server Icons by RRZEIcons (via Wikipedia)
- Connections by Langwitches (via Flickr)
- Hydrogen and Water (via Wikipedia)
- square-peg-round-hole-21 by Yoel Ben-Avraham (Flickr)
- All dragons CC0