Riak Search

A Full-Text Search
and Indexing Engine
based on Riak

Erlang Factory · London · June 2010

Basho Technologies
Rusty Klophaus - @rklophaus

Why did we build it?

What are the major goals?

How does it work?

Part One

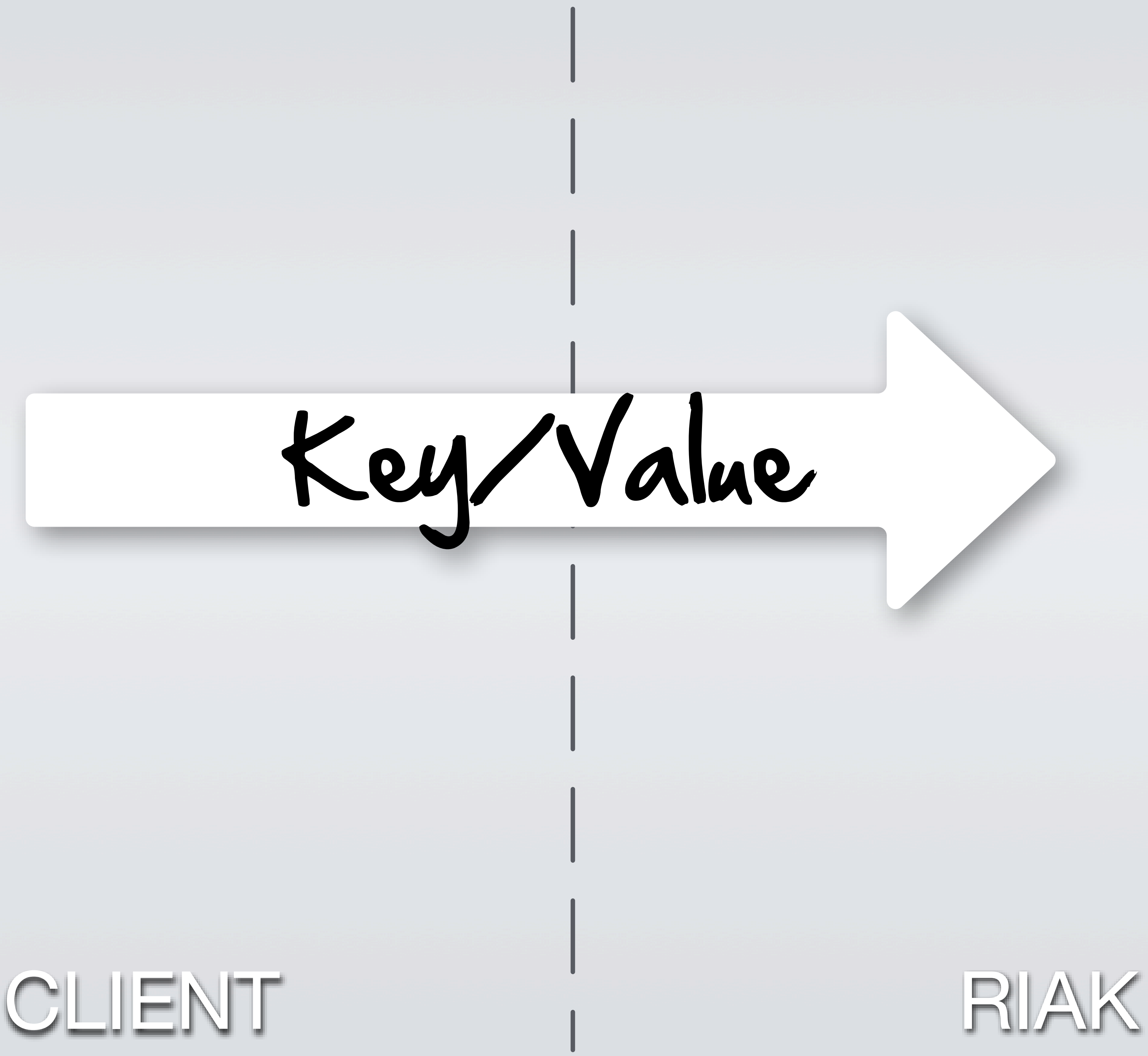
Why did we build Riak Search?

Riak is

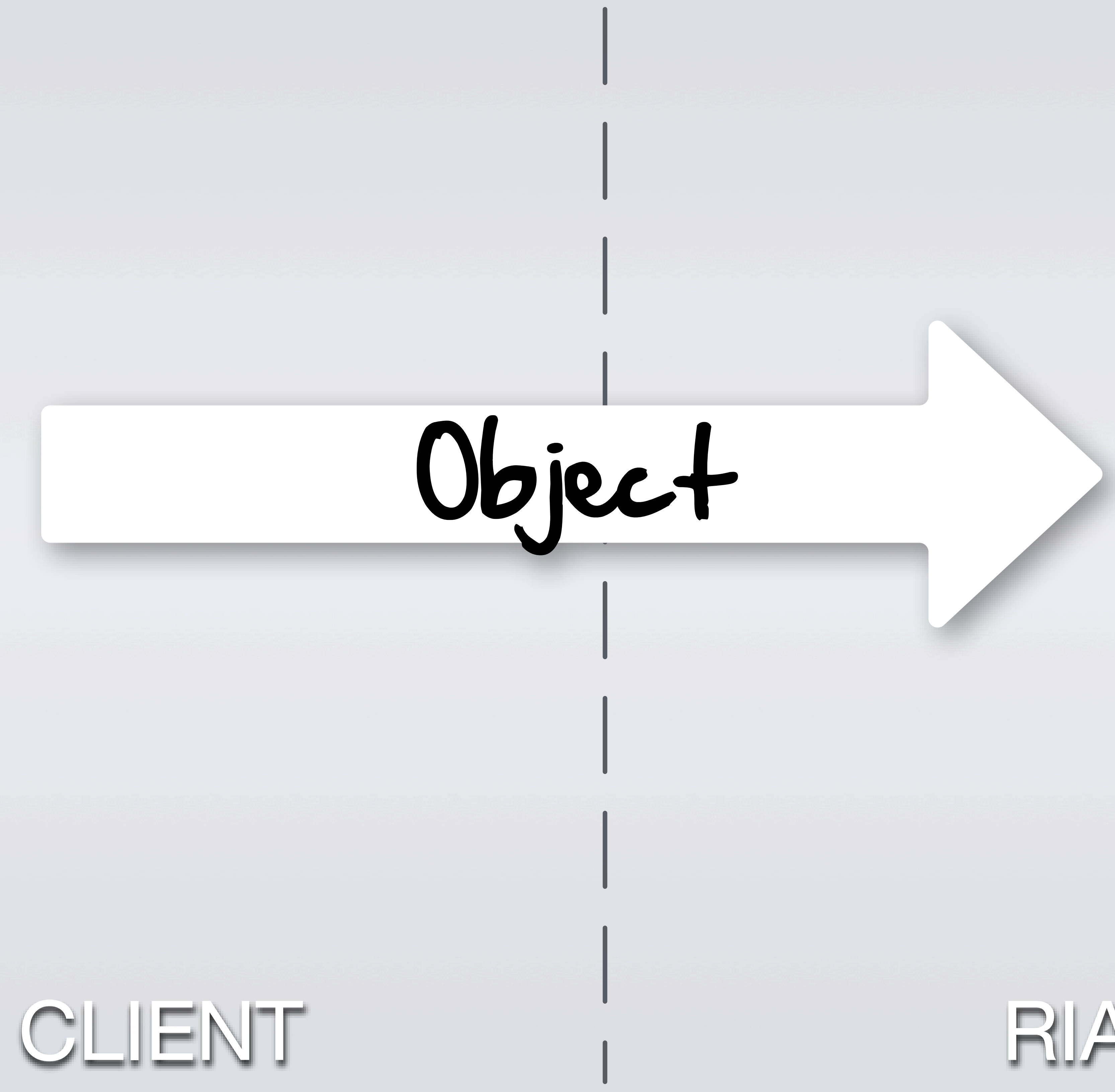
a scalable, highly-available, networked,

open-source key/value store.

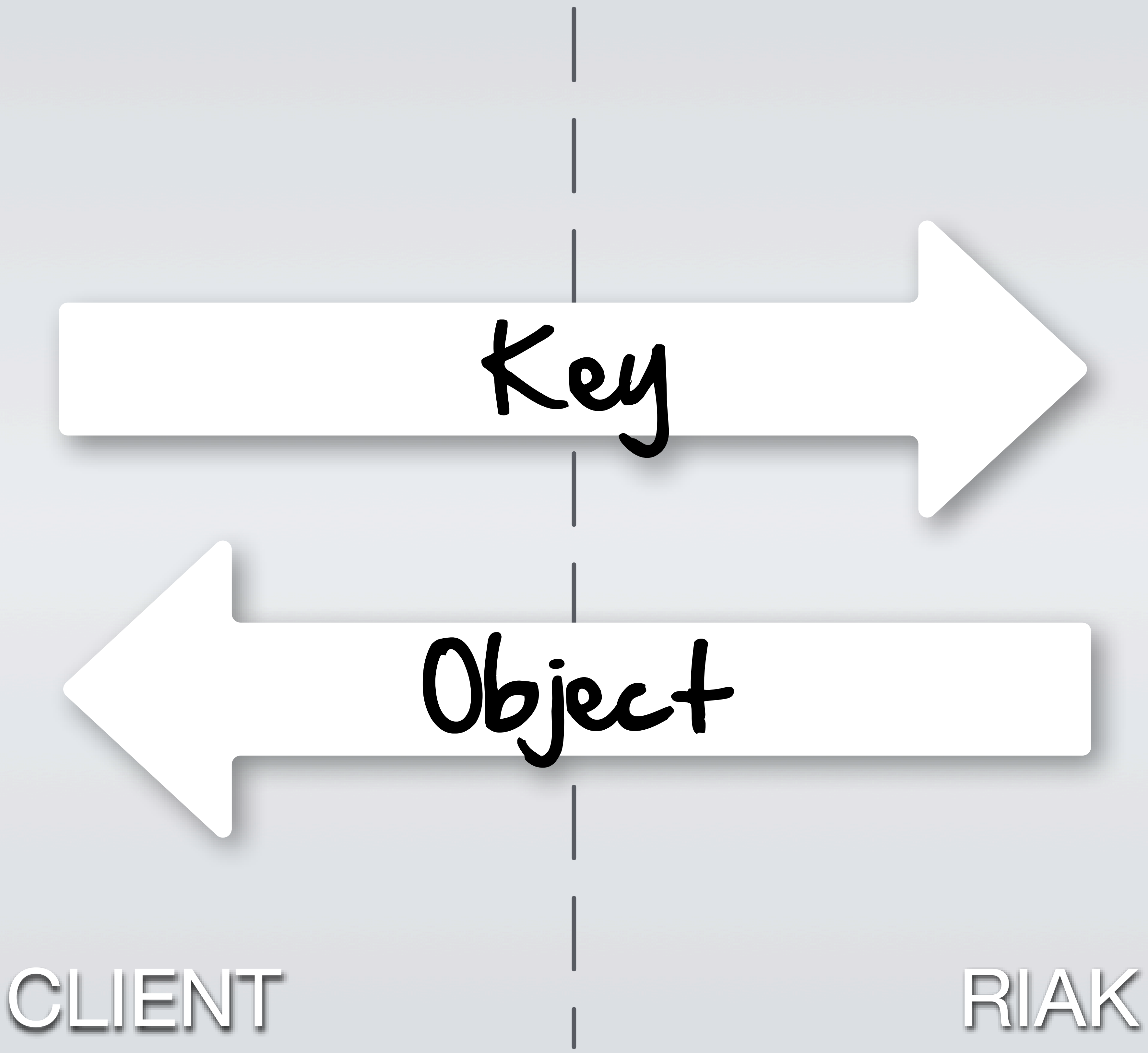
Writing to a Key/Value Store



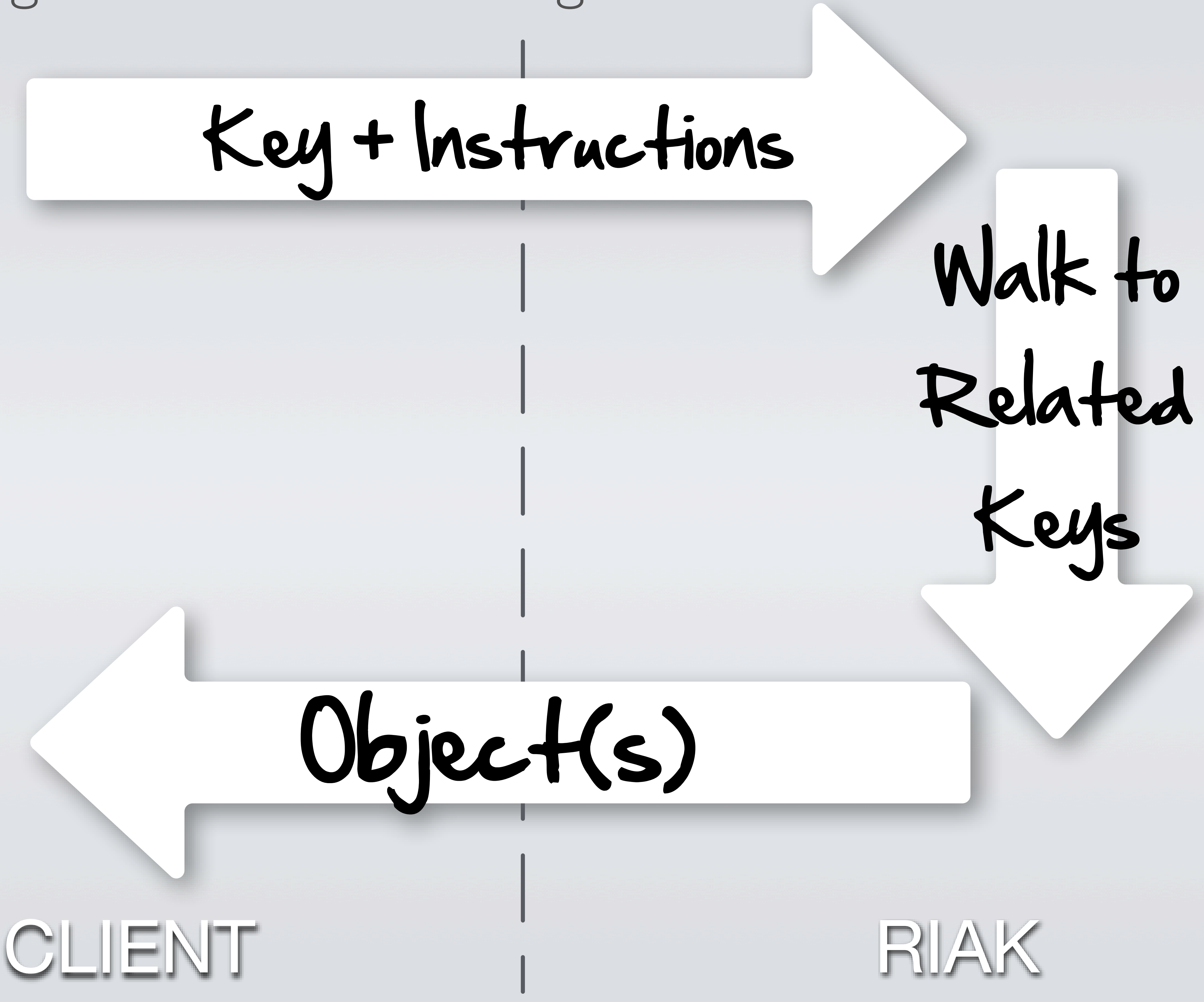
Writing to a Key/Value Store



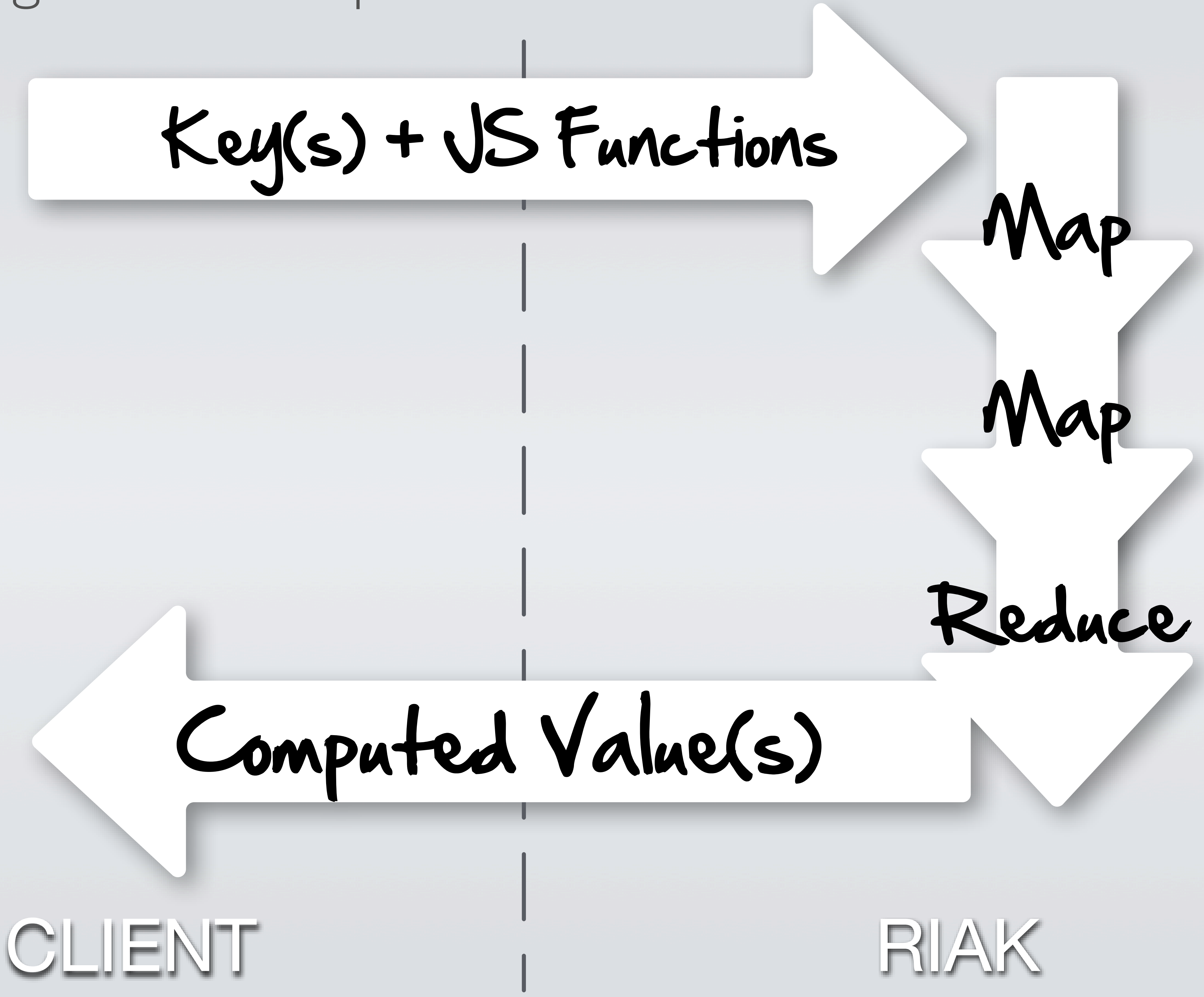
Querying a Key/Value Store



Querying Riak via LinkWalking



Querying Riak via Map/Reduce



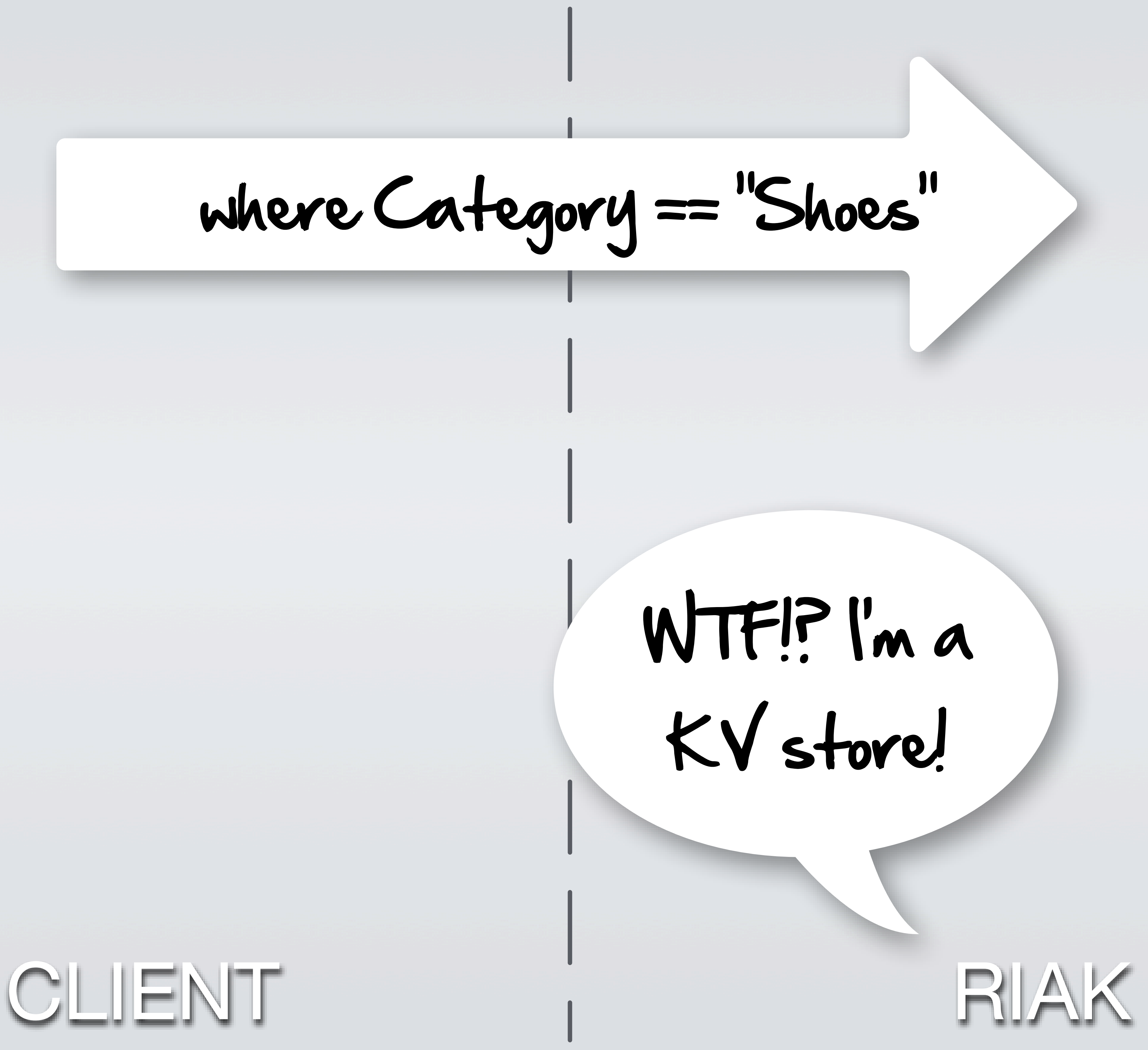
Key/Value Stores

like

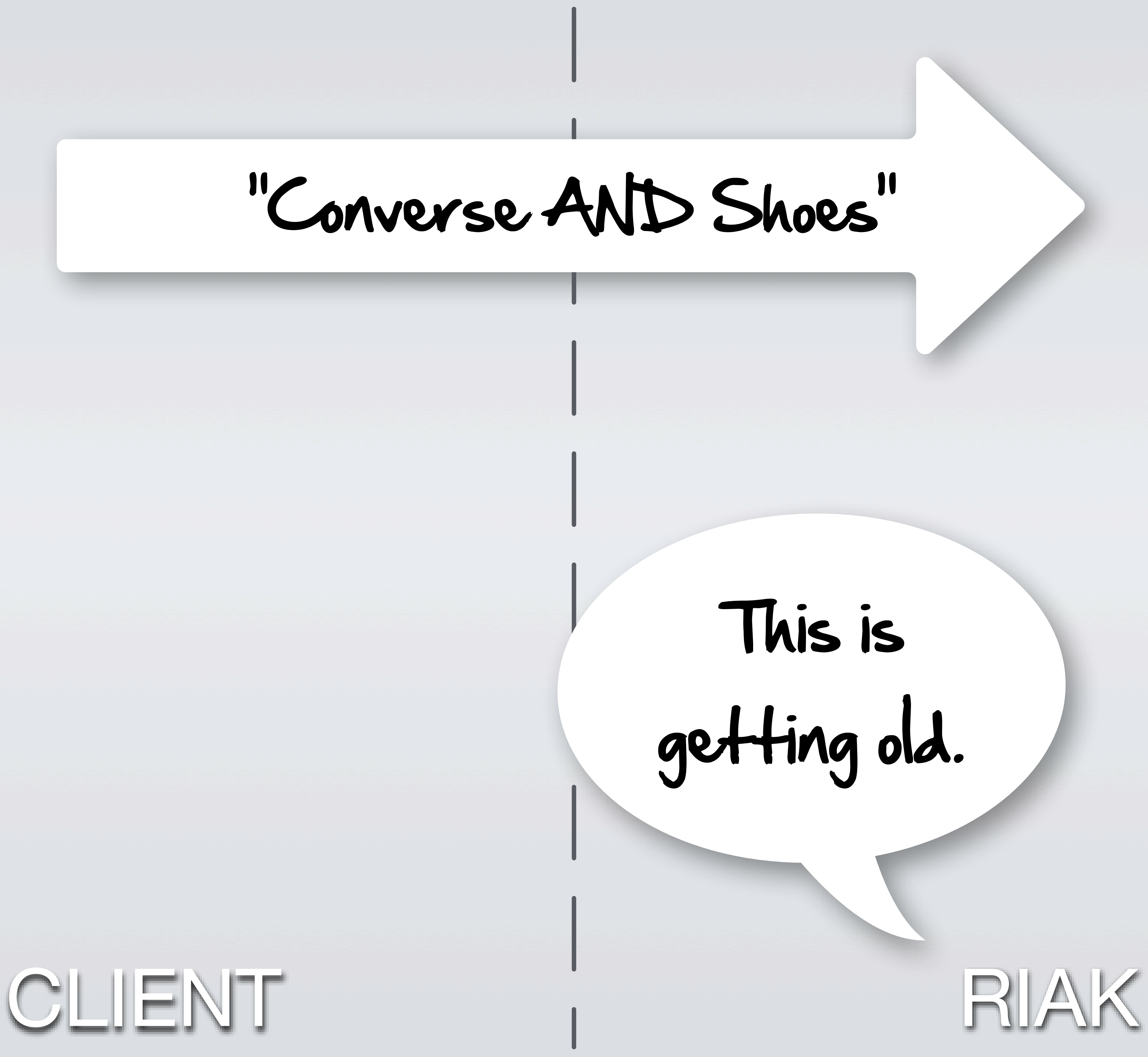
Key-Based Queries

basho

Query by Secondary Index



Full-Text Query



basho

These kinds of queries need an Index.

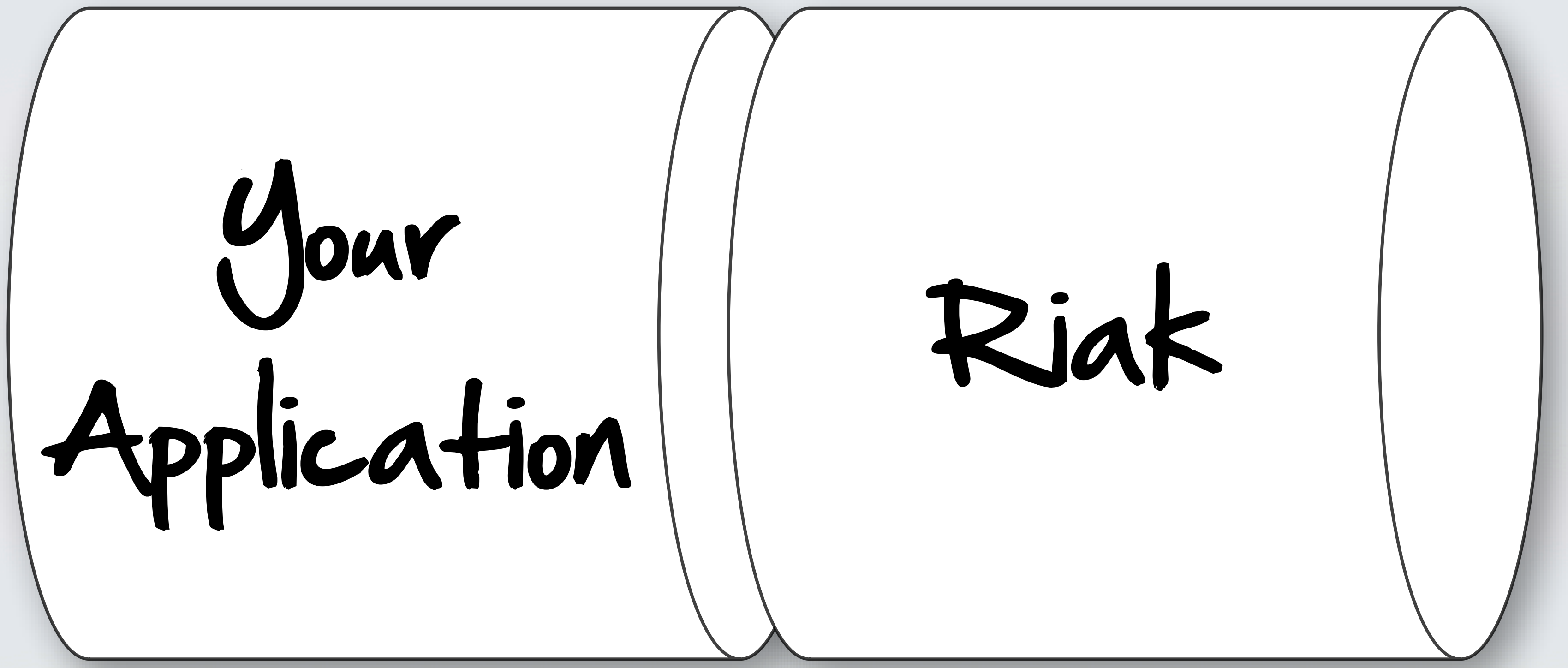
Market Opportunity!

Part Two

What are the major goals of Riak Search?

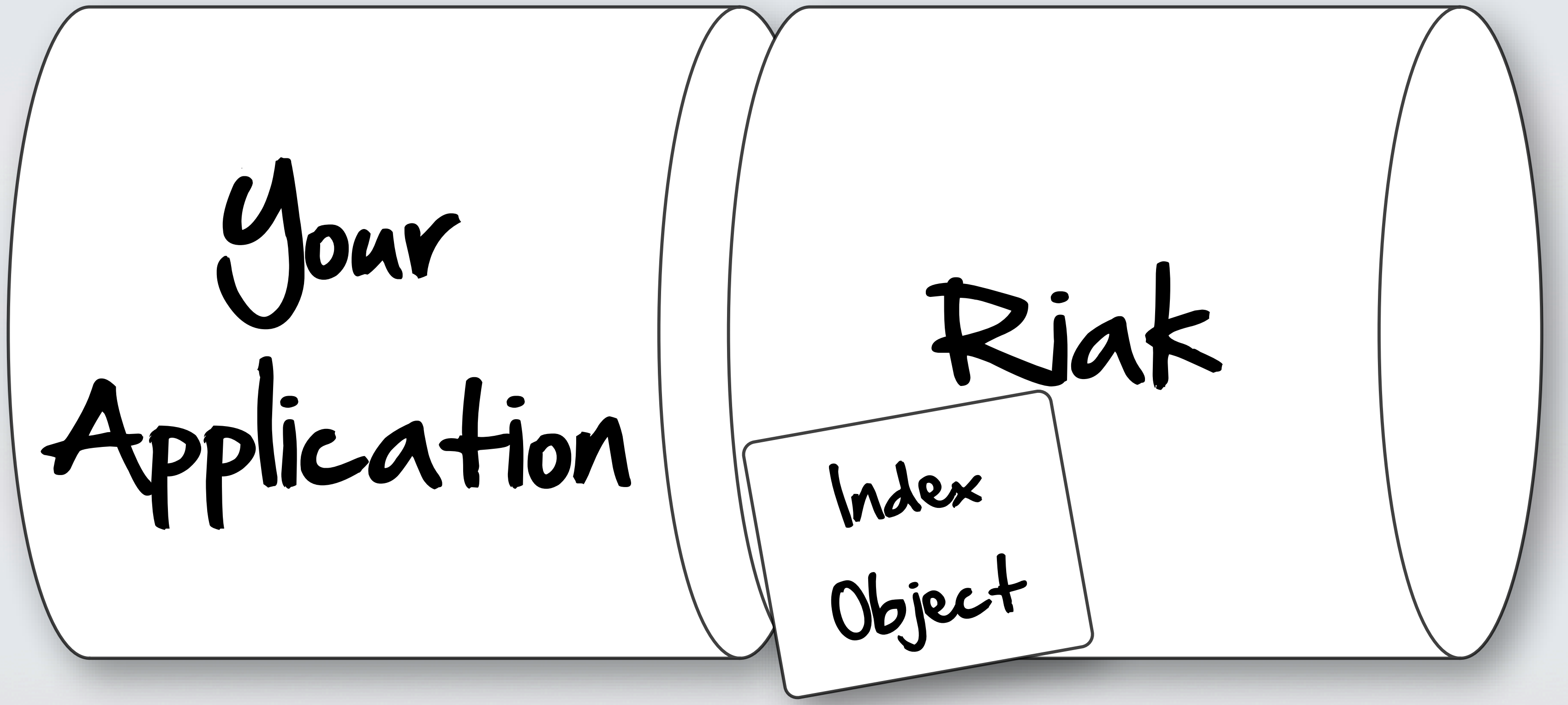
basho

An application built on Riak.

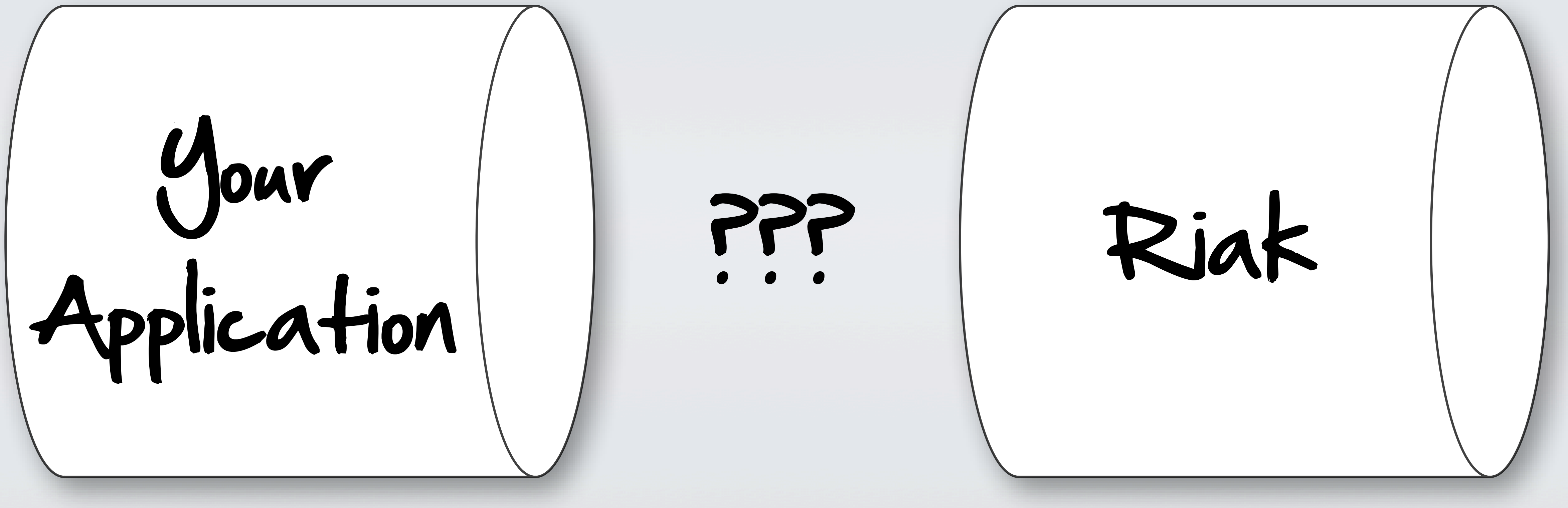


basho

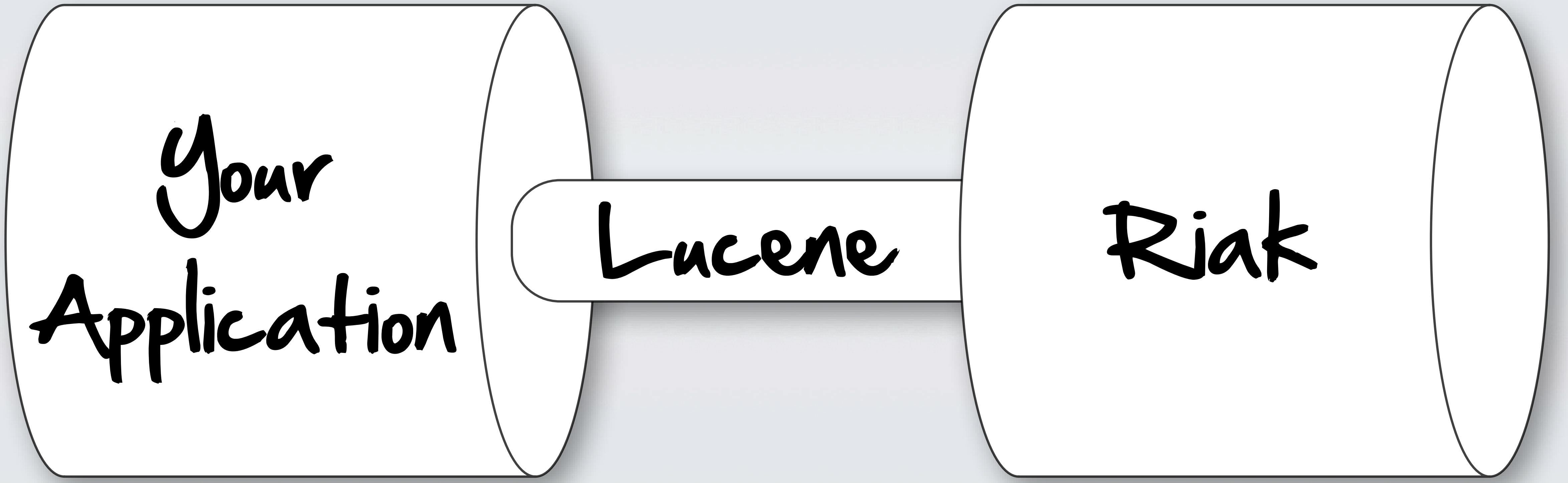
Hrm... I need an index.



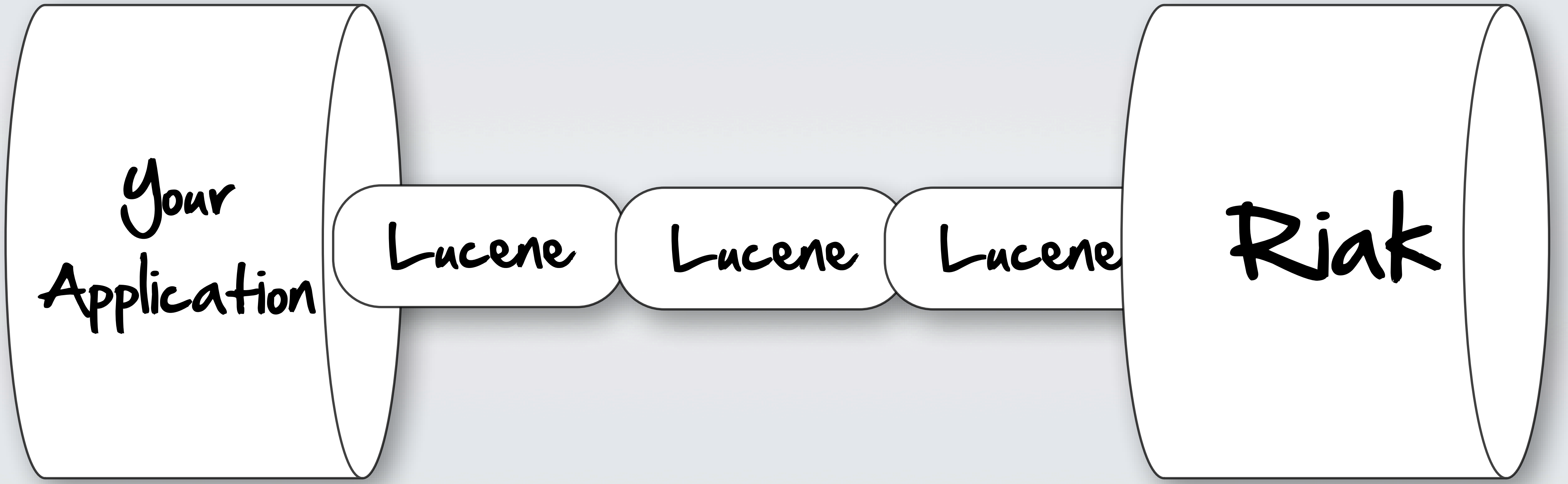
Hrm... I need an index with more features.



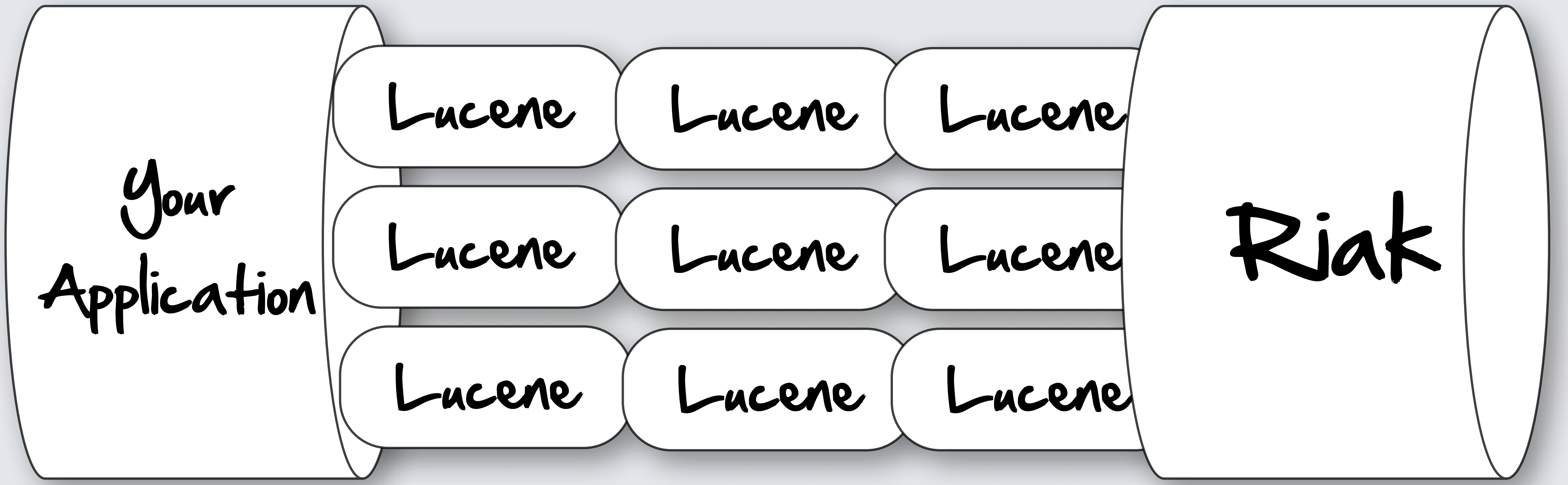
Lucene should do the trick...



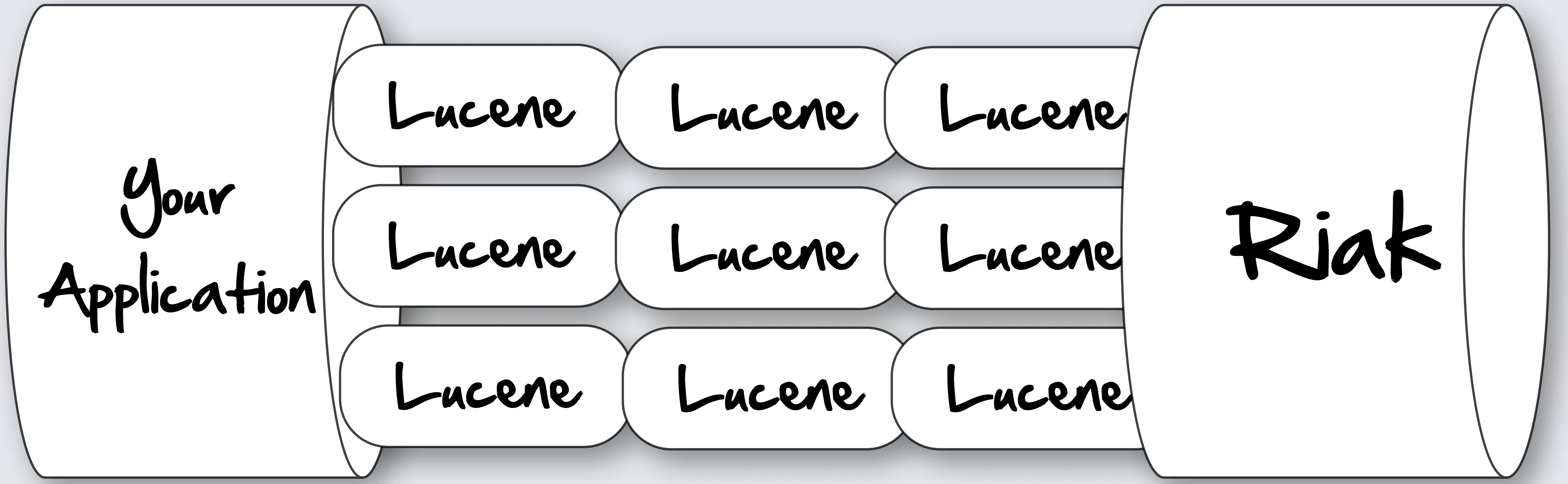
...shard to add more storage capacity...



...replicate to add more throughput.

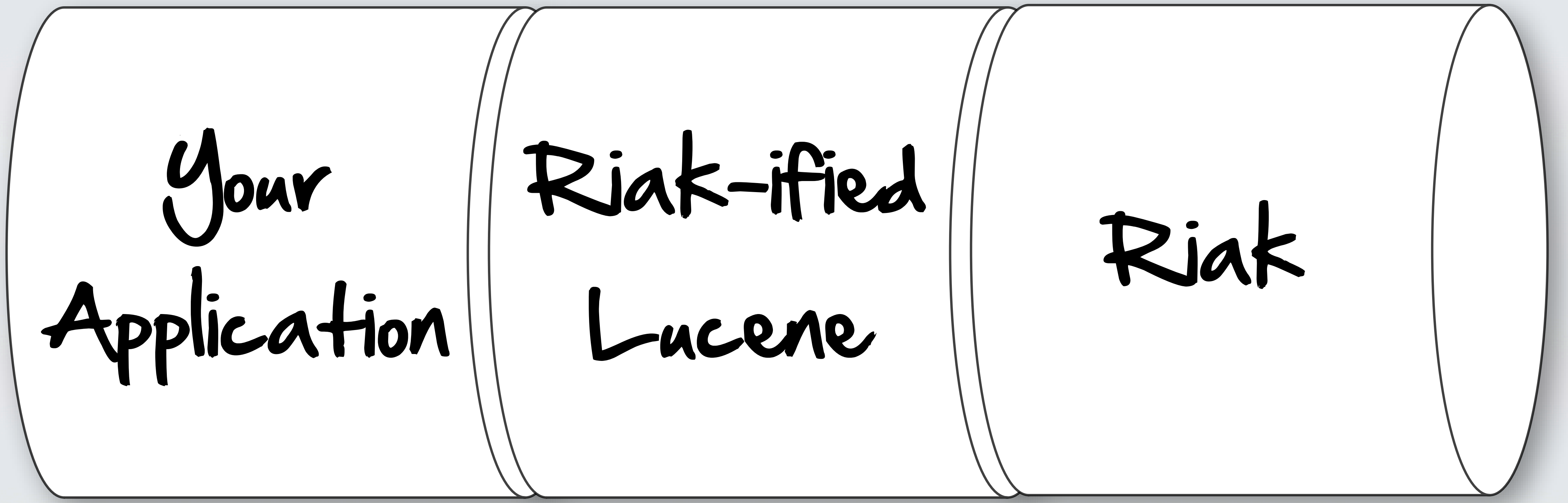


...replicate to add more throughput.

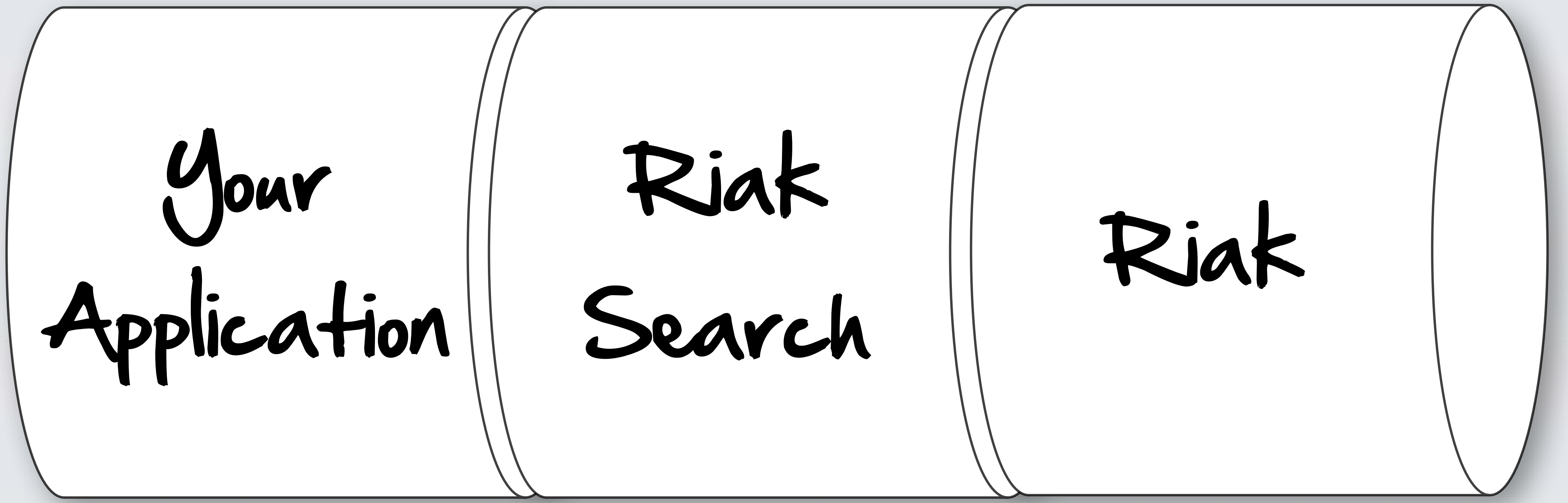


Operations nightmare!

What do we really want?



What do we really want?



Functionality? Be like Lucene (and more).

- Lucene Syntax
- Leverages Java Lucene Analyzers
- Solr Endpoints
- Integration via Riak Post-Commit Hook (Index)
- Integration via Riak Map/Reduce (Query)
- Near-Realtime
- Schema-less

Operations? Be like Riak.

- No special nodes
- Add nodes, get more compute and storage
- Automatically load balance
- Replicas for durability and performance
- Index and query in parallel
- Swappable storage backends

basho

Part Three

How do we do it?

A Gentle Introduction to Document Indexing

basho

The Inverted Index

Document

Inverted Index

#1

Every dog has
his day.

```
day, 1
dog, 1
every, 1
has, 1
his, 1
```

The Inverted Index

Documents

#1 Every dog has his day.

#2 The dog's bark is worse than his bite.

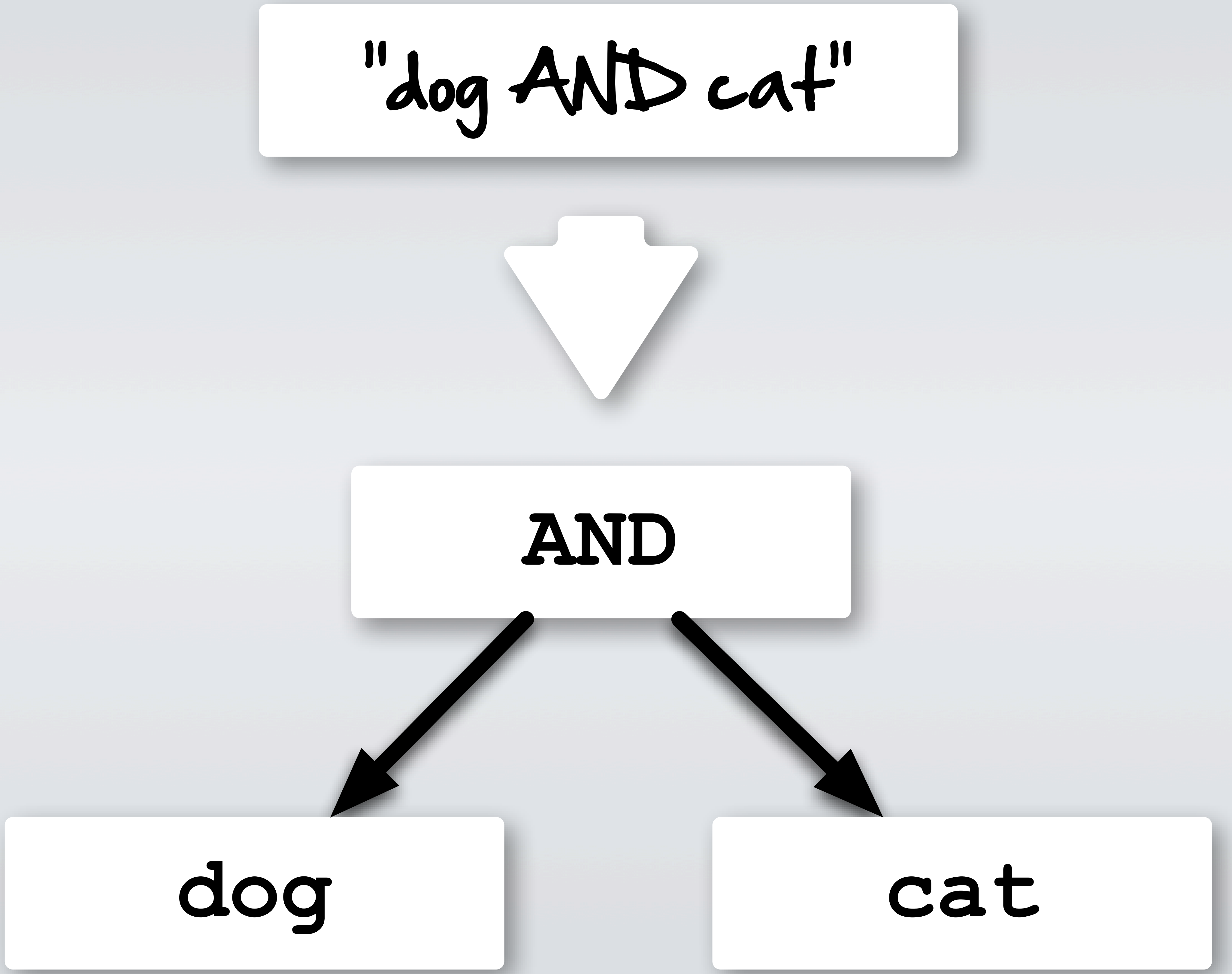
#3 Let the cat out of the bag.

#4 It's raining cats and dogs.

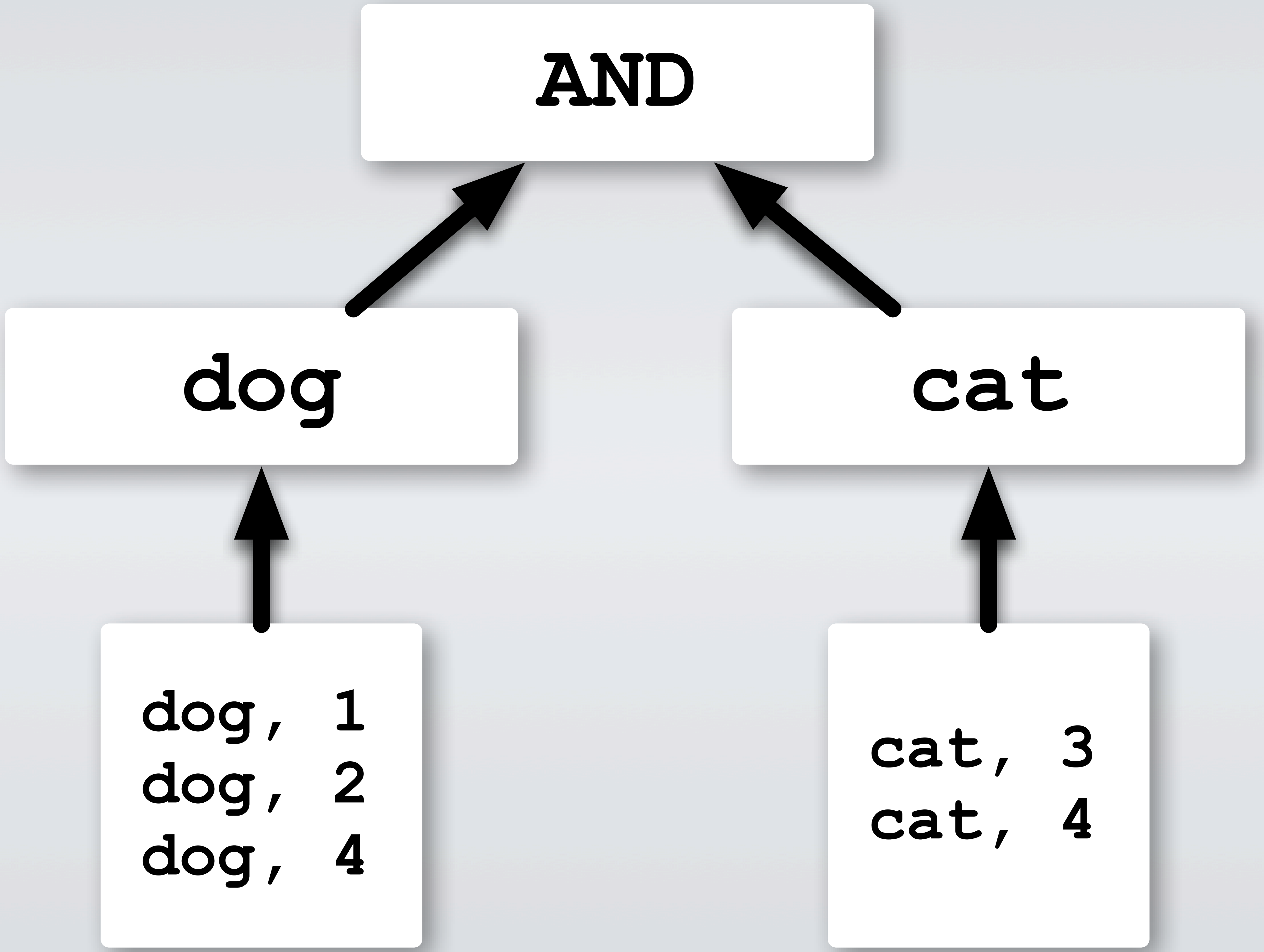
Combined Inverted Index

```
and, 4
bag, 3
bark, 2
bite, 2
cat, 3
cat, 4
day, 1
dog, 1
dog, 2
dog, 4
every, 1
has, 1
...
```

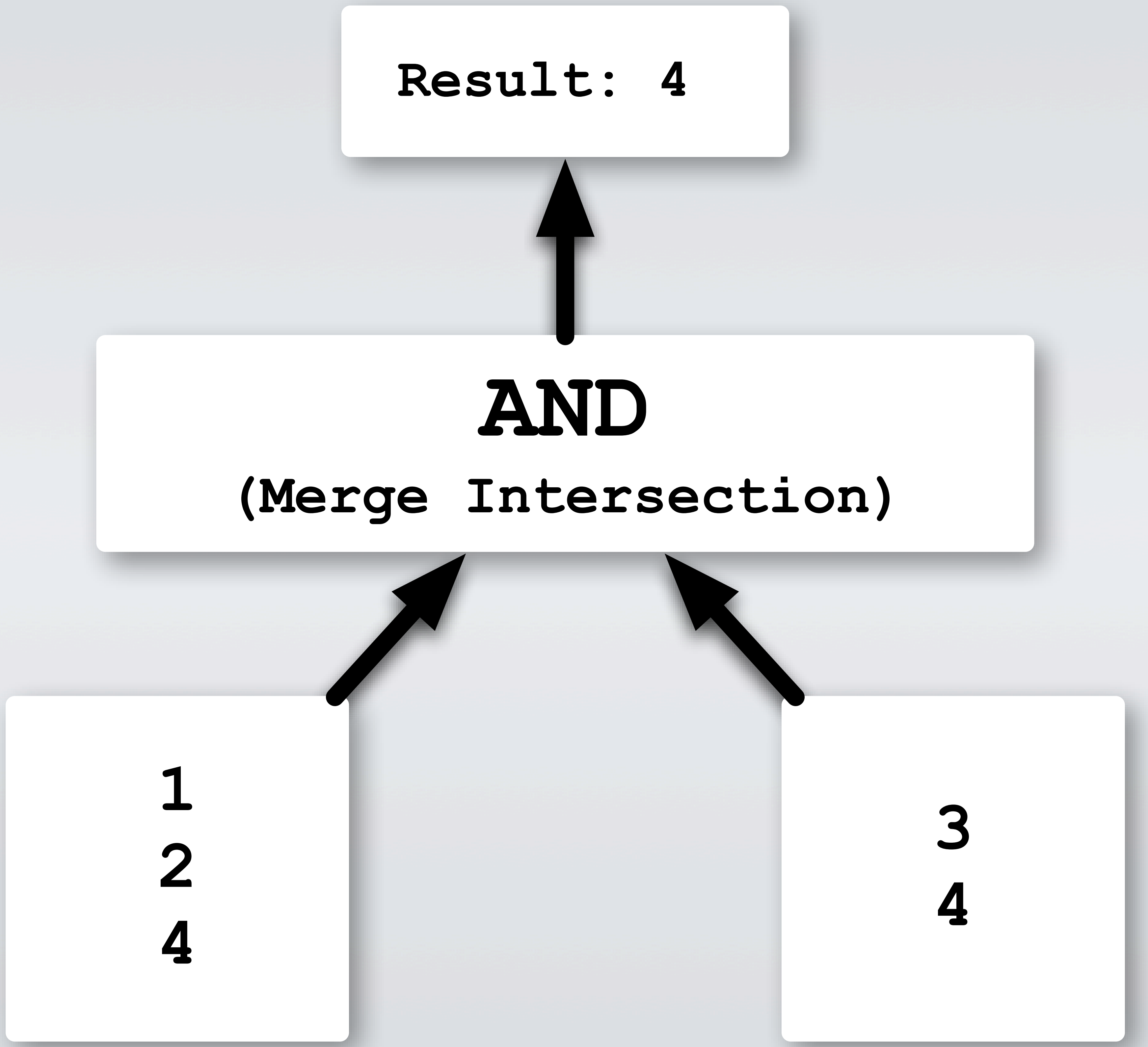
At Query Time...



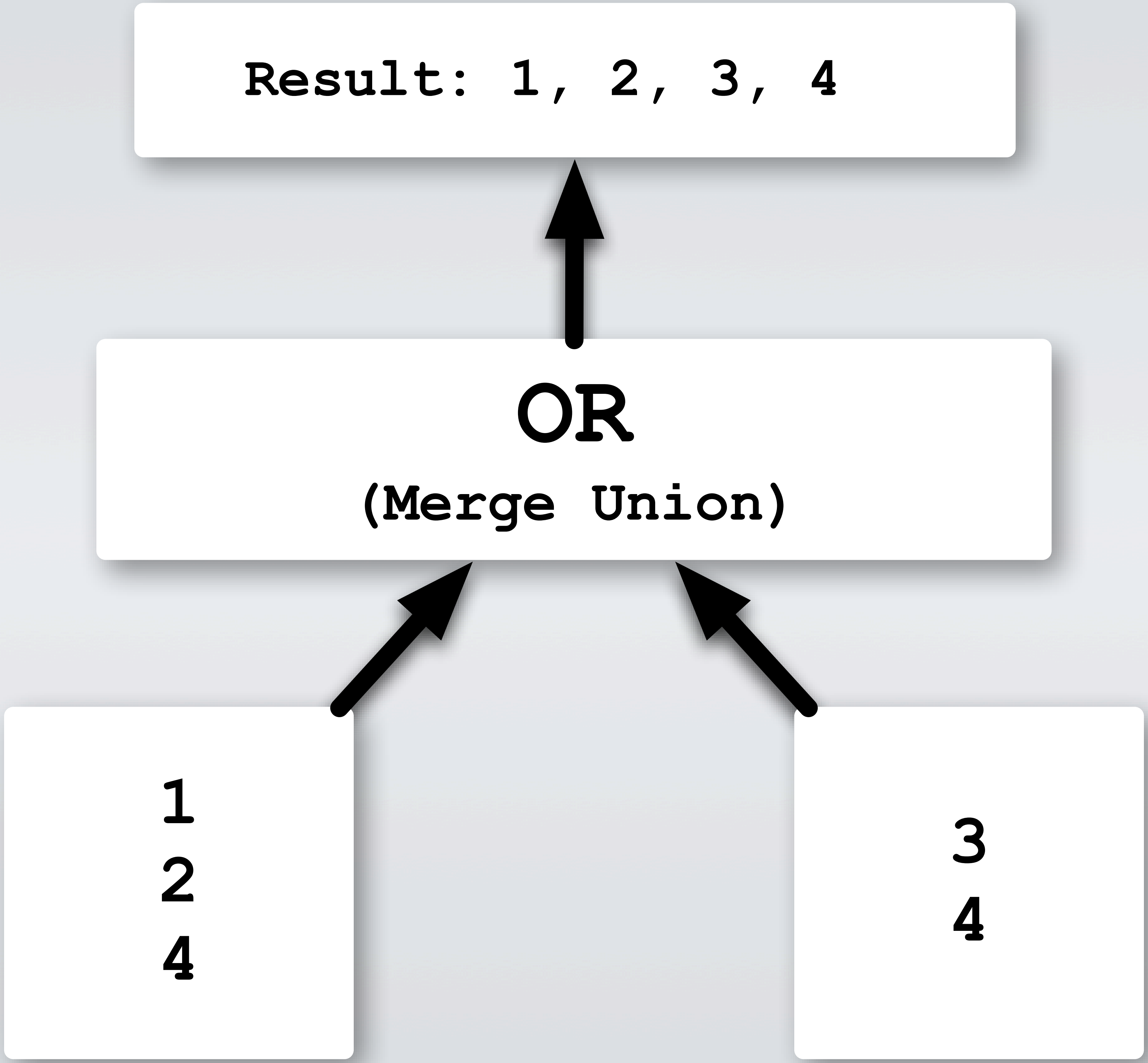
At Query Time...



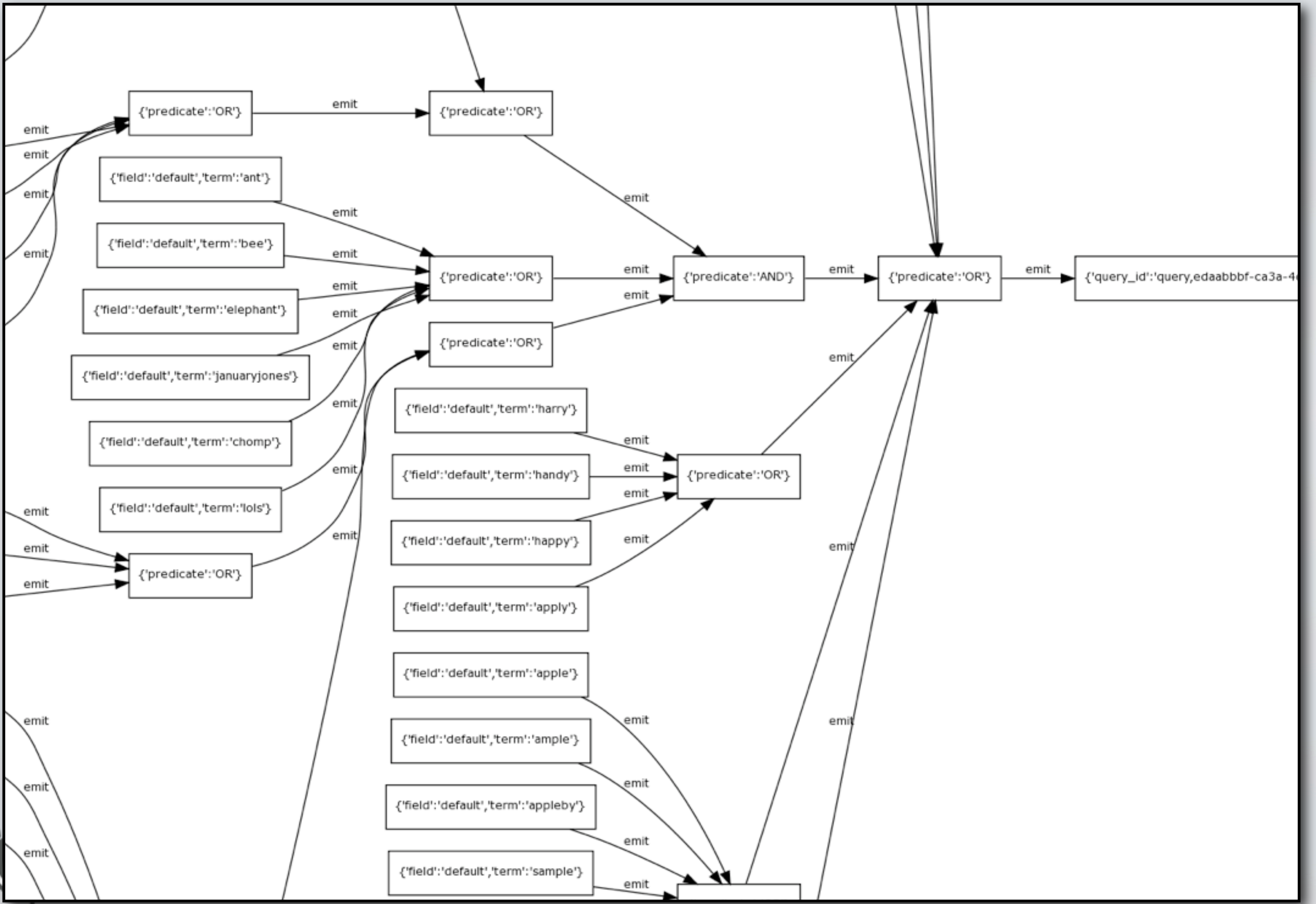
At Query Time...



At Query Time...



Complex Behavior from Simple Structures


{'predicate':'OR'}
emit
{'predicate':'OR'}
{'field':'default','term':'ant'}
{'field':'default','term':'bee'}
{'field':'default','term':'elephant'}
{'field':'default','term':'januaryjones'}
{'field':'default','term':'chomp'}
{'field':'default','term':'lols'}
{'predicate':'OR'}
{'predicate':'AND'}
{'predicate':'OR'}
{'query_id':'query,edaabbbf-ca3a-4
{'predicate':'OR'}
{'field':'default','term':'harry'}
{'field':'default','term':'handy'}
{'field':'default','term':'happy'}
{'field':'default','term':'apply'}
{'predicate':'OR'}
{'predicate':'OR'}
{'field':'default','term':'apple'}
{'field':'default','term':'ample'}
{'field':'default','term':'appleby'}
{'field':'default','term':'sample'}

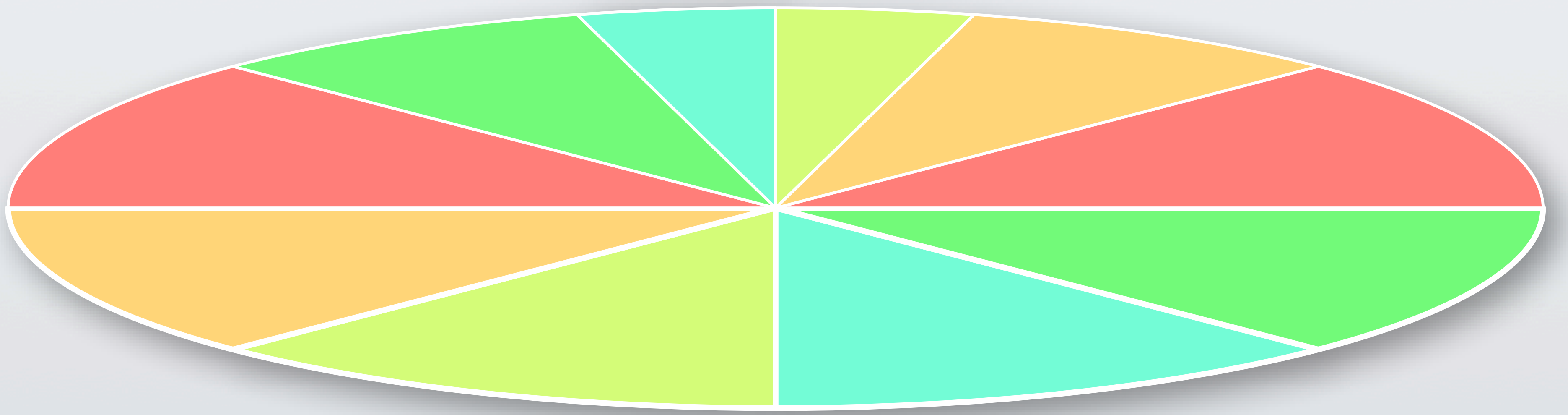
Storage Approaches...

basho

Riak Search uses Consistent Hashing to store data on Partitions

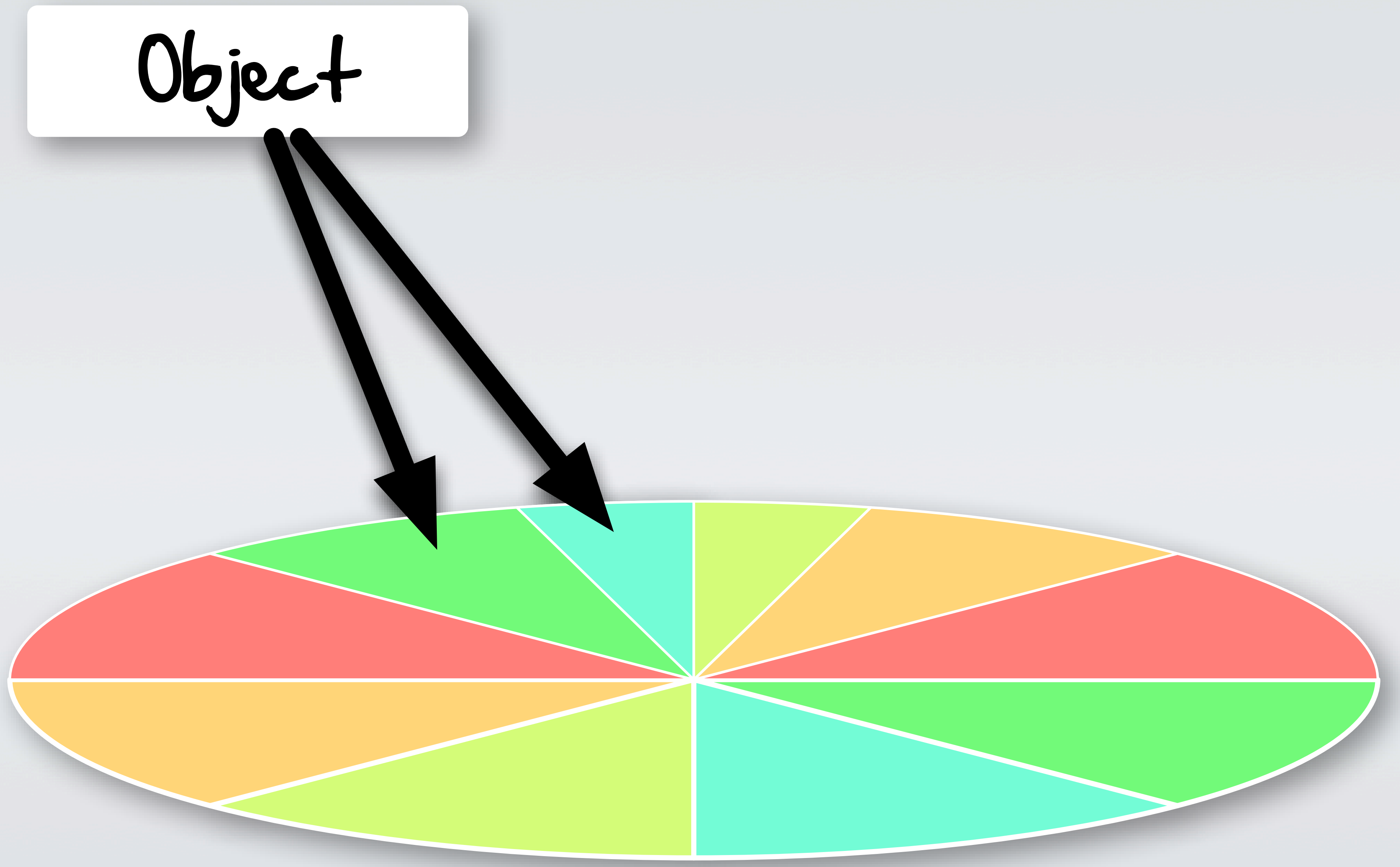
Introduction to Consistent Hashing and Partitions

```
Partitions = 10
Number of Nodes = 5
Partitions per Node = 2
Replicas (NVal) = 2
```



basho

Introduction to Consistent Hashing and Partitions



Document Partitioning

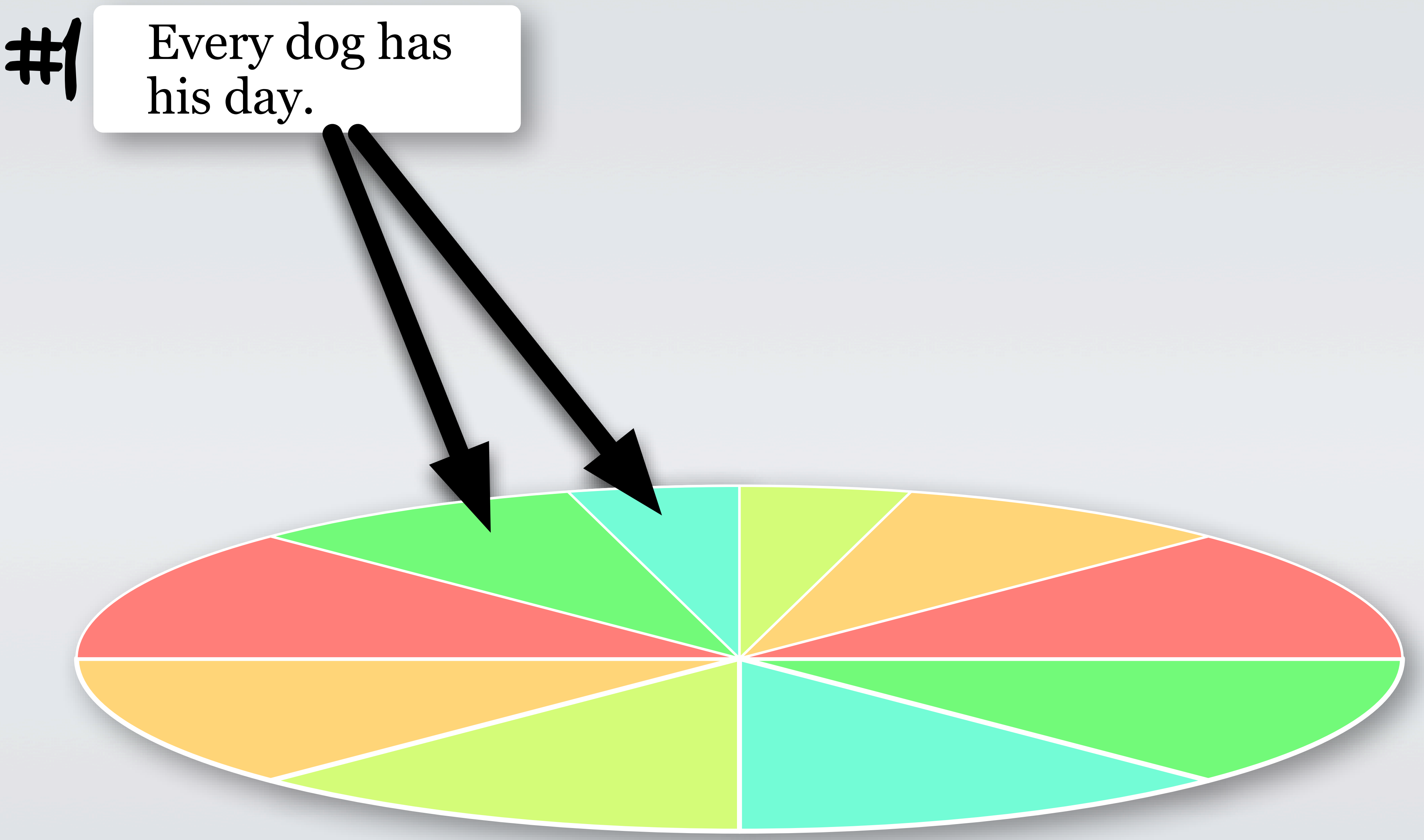
vs.

Term Partitioning

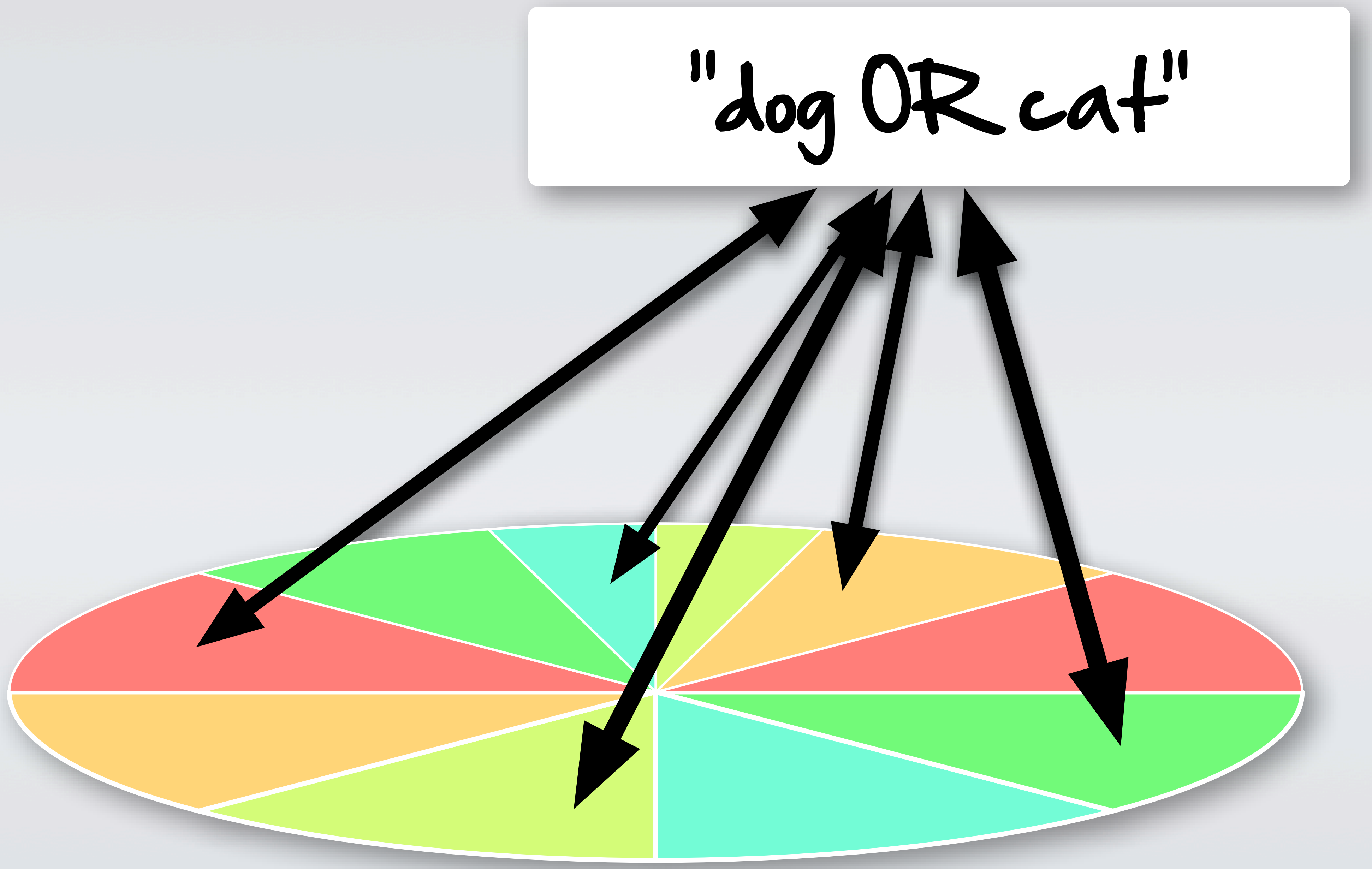
...and the
Resulting Tradeoffs

basho

Document Partitioning @ Index Time



Document Partitioning @ Query Time



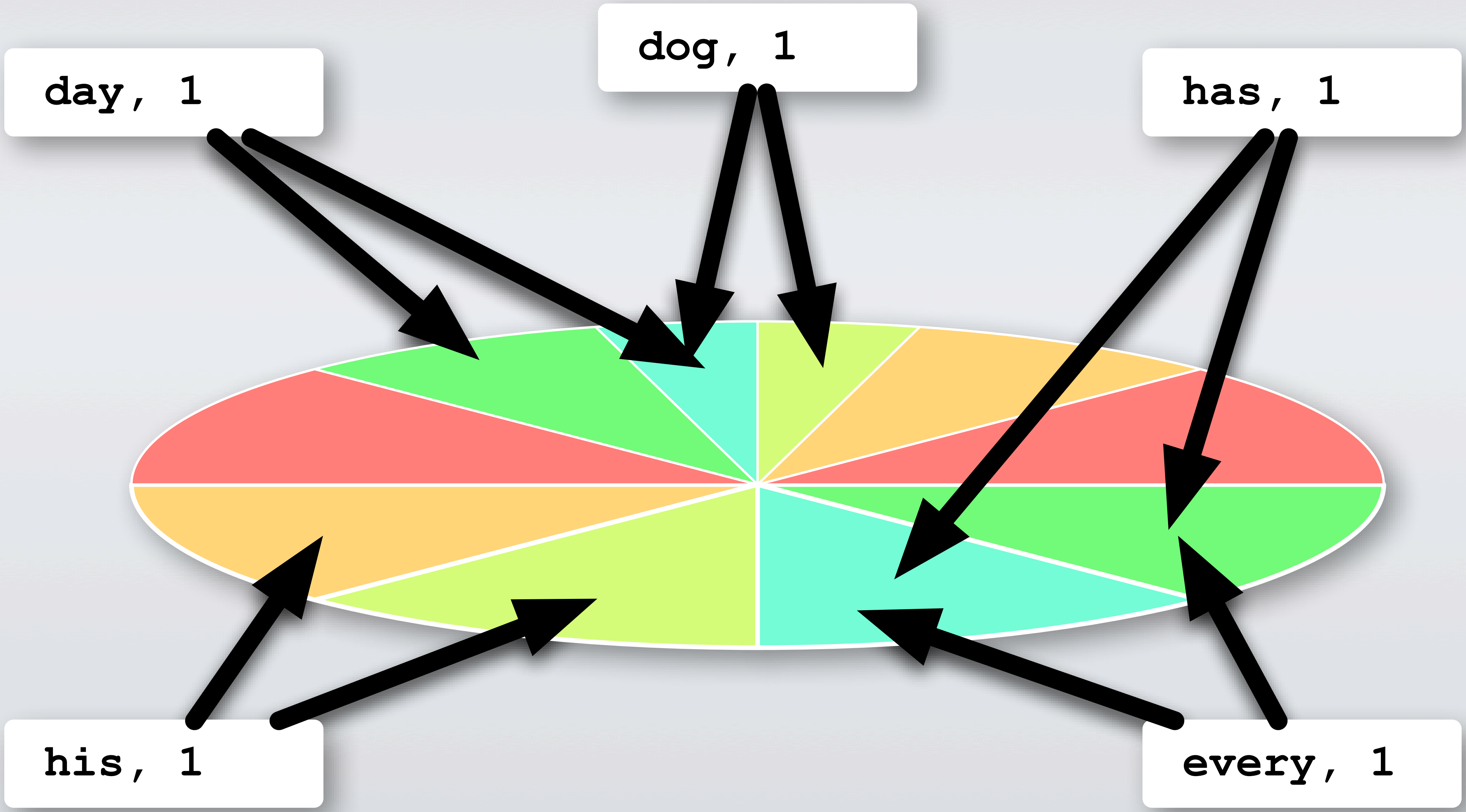
basho

Term Partitioning @ Index Time

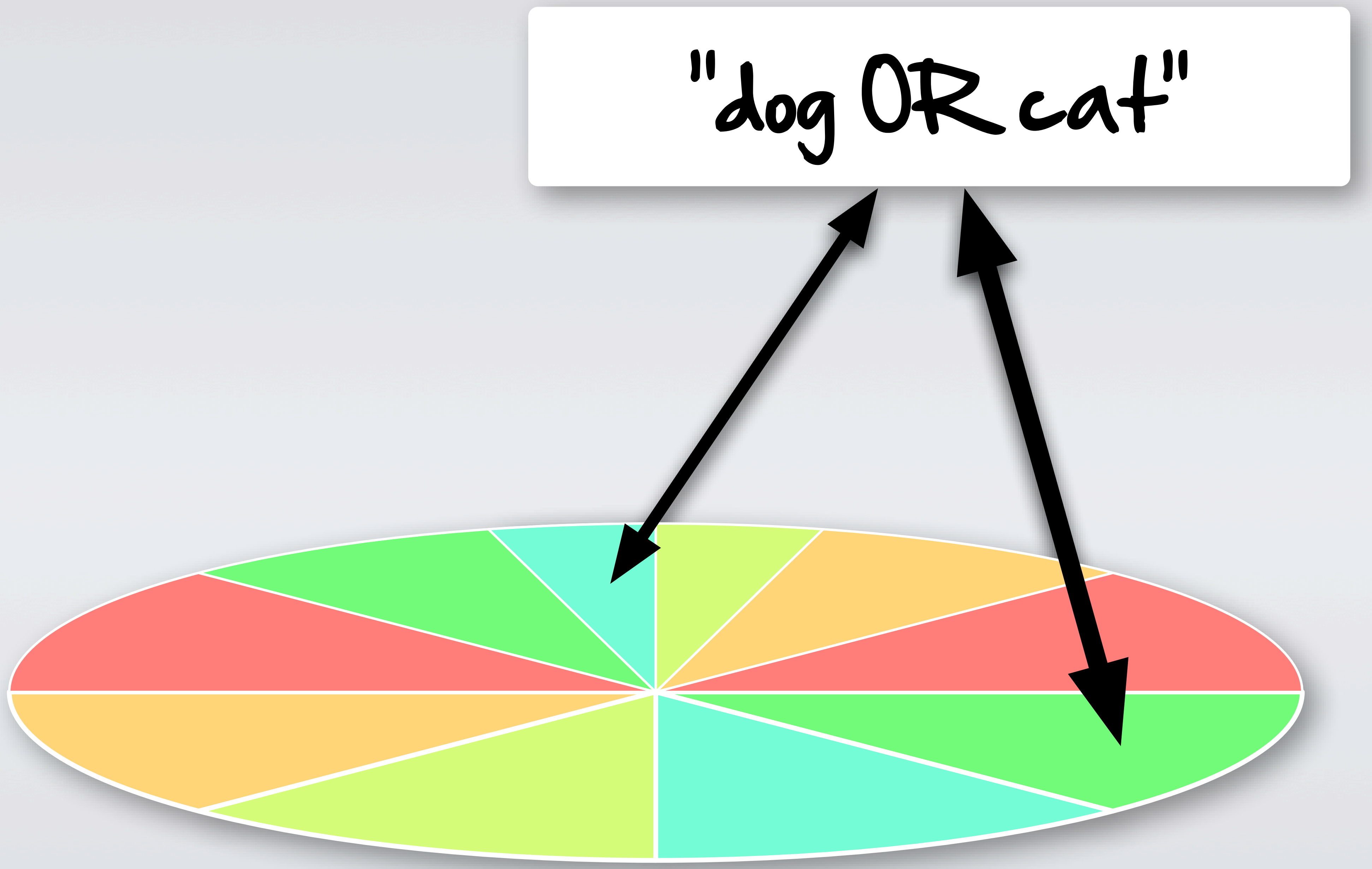
#1 Every dog has his day.

```
day, 1
dog, 1
every, 1
has, 1
his, 1
```

Term Partitioning @ Index Time



Term Partitioning @ Query Time



basho

Tradeoffs...

Document Partitioning	Term Partitioning
+ Lower Latency Queries	- Higher Latency Queries
- Lower Throughput	+ Higher Throughput
- Lots of Disk Seeks	- Hotspots in Ring (the "Obama" problem)

basho

Riak Search: Term Partitioning

Term-partitioning is the most viable approach for our beta clients' needs: high throughput on Really Big Datasets.

Optimizations:

- Term splitting to reduce hot spots
- Bloom filters & caching to save query-time bandwidth
- Batching to save query-time & index-time bandwidth

Support for either approach eventually.

basho

Diving Deeper:
The Lifecycle of a Query

Parse the Query

The Query

```
meeting AND (face OR phone)
```

basho

The Query as an Erlang Term (Parse w/ Leex and Yecc)

```
[{land, [
    {term,"meeting",[]},
    {lor,[
        {term,"face",[]},
        {term,"phone",[]}
    ]}
]}]
```

basho

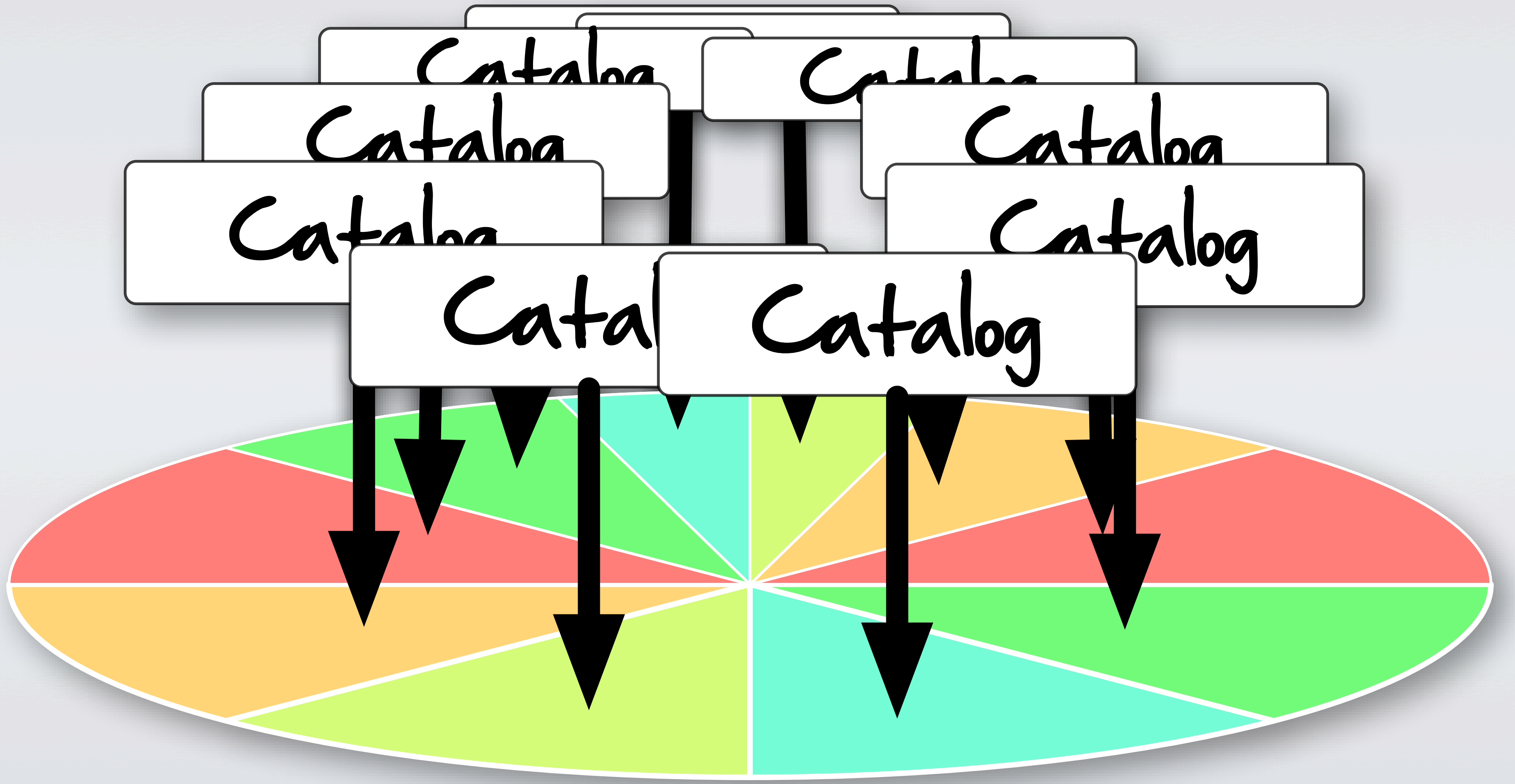
The Query as a Graph

$$\begin{tikzcd}
 & \texttt{\#land} \arrow[dl] \arrow[d] & \\
\texttt{\#term} \arrow[d] & \texttt{\#lor} \arrow[d] \arrow[dr] & \\
\texttt{"meeting"} & \texttt{\#term} \arrow[d] & \texttt{\#term} \arrow[d] \\
 & \texttt{"phone"} & \texttt{"face"}
\end{tikzcd}$$

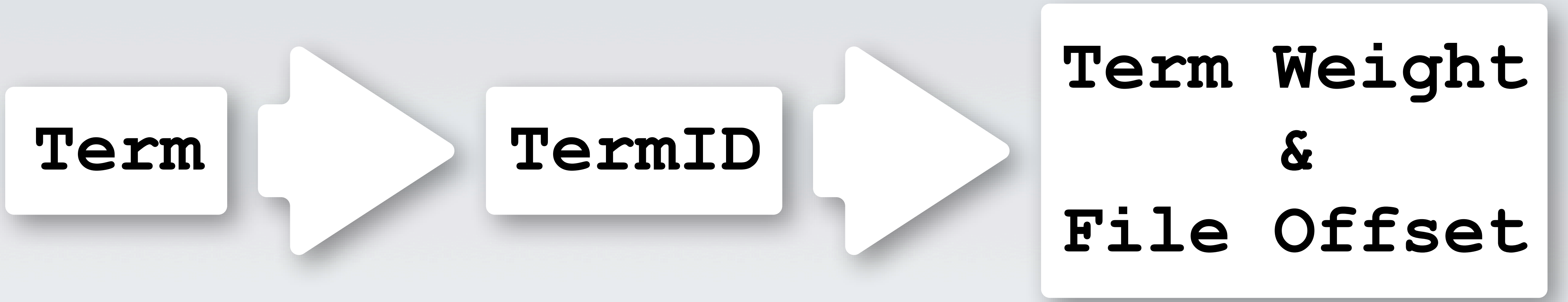
Plan the Query

basho

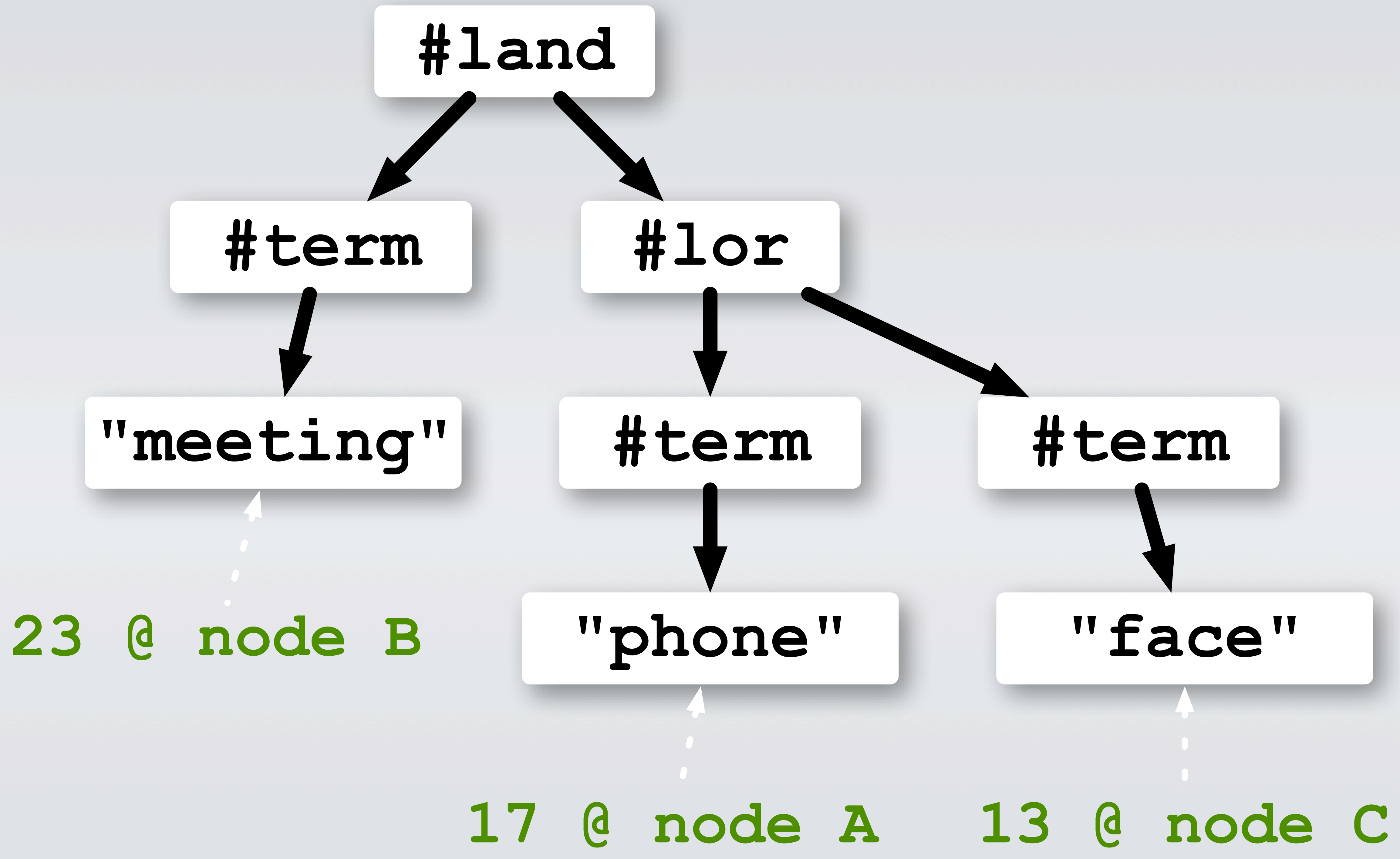
System Catalog



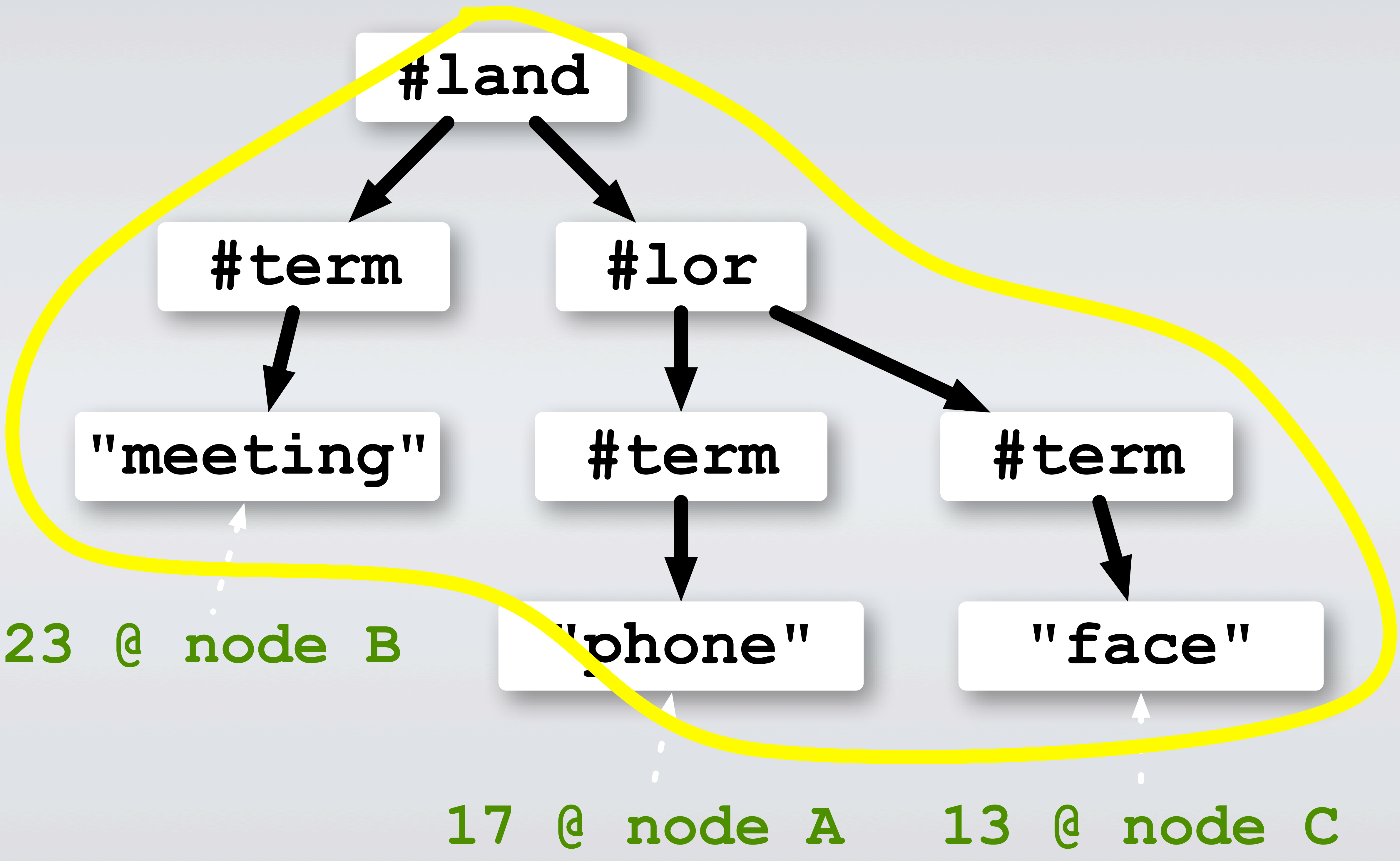
System Catalog



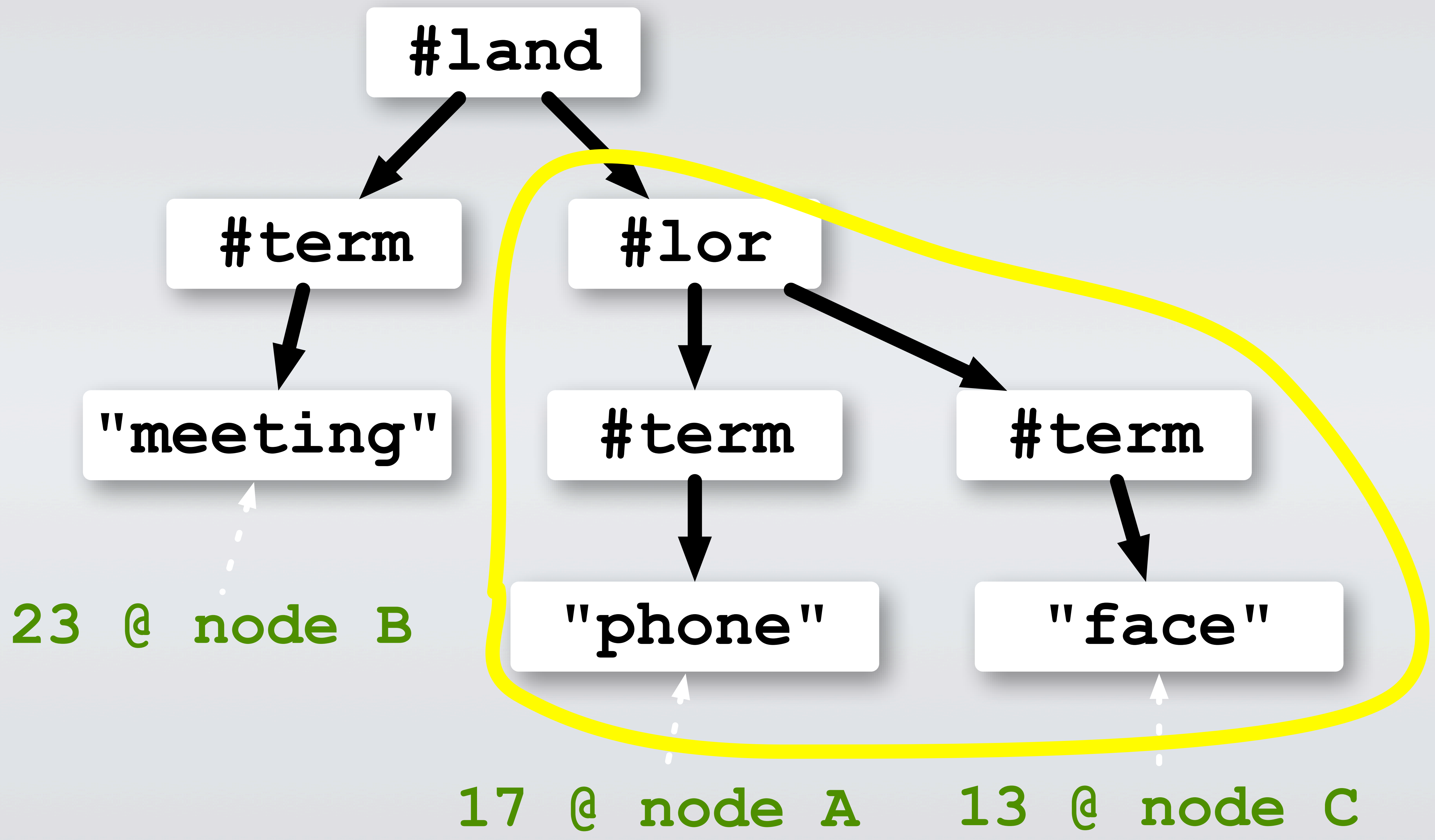
Consult the System Catalog for Term/Node Weights



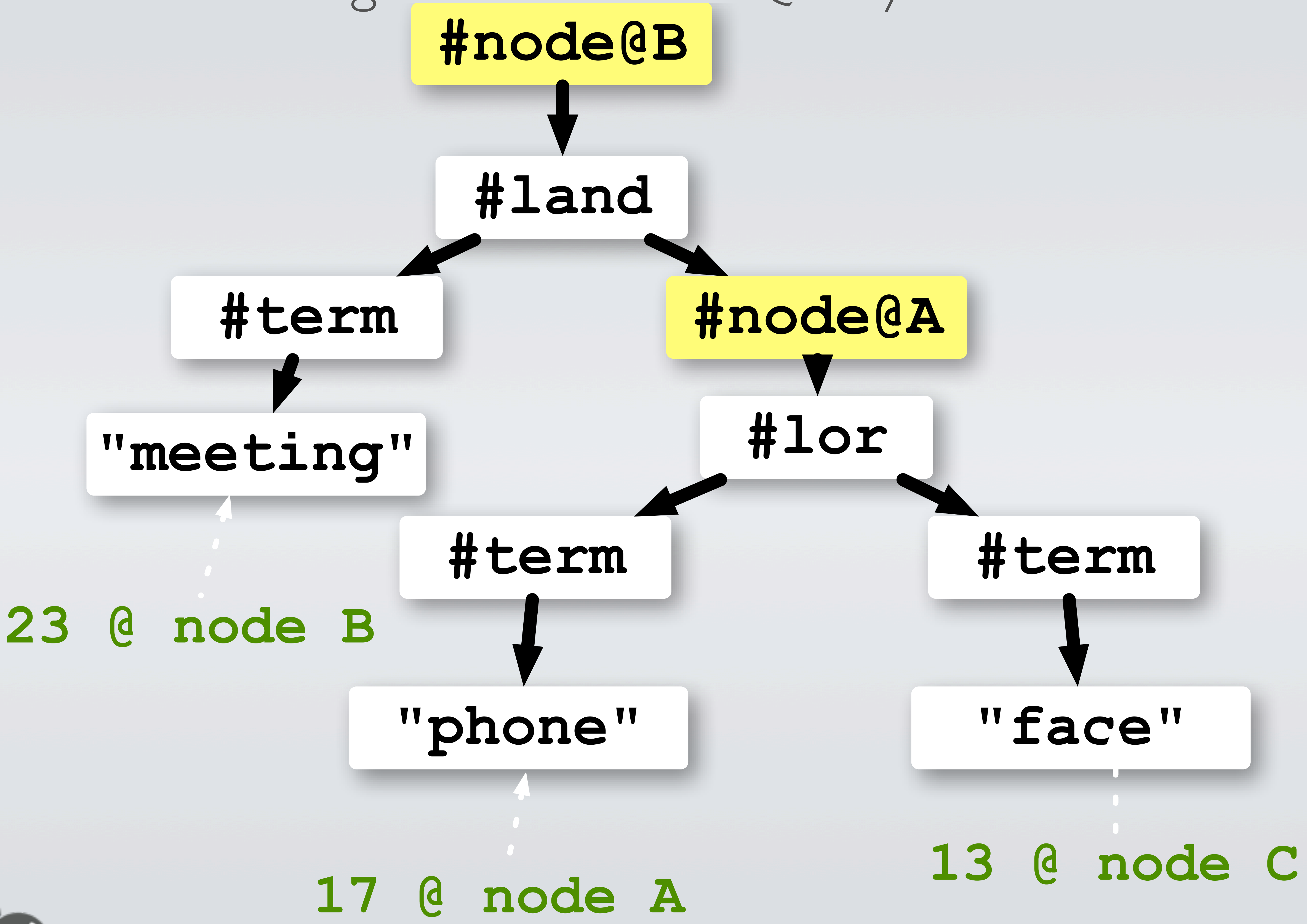
Use Term Weights to Plan the Query



Use Term Weights to Plan the Query



Use Term Weights to Plan the Query

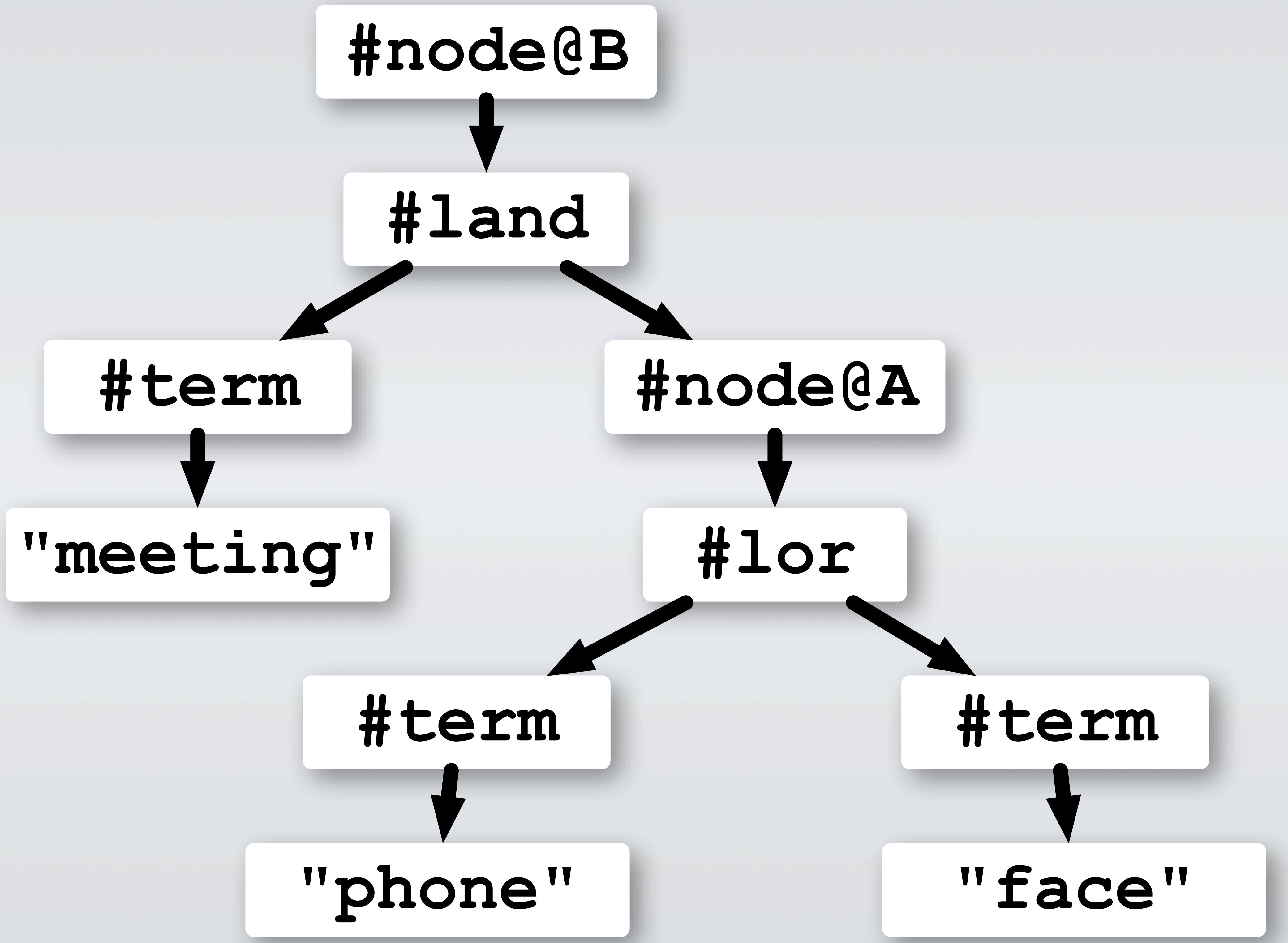


The Node-Assigned Query as an Erlang Term

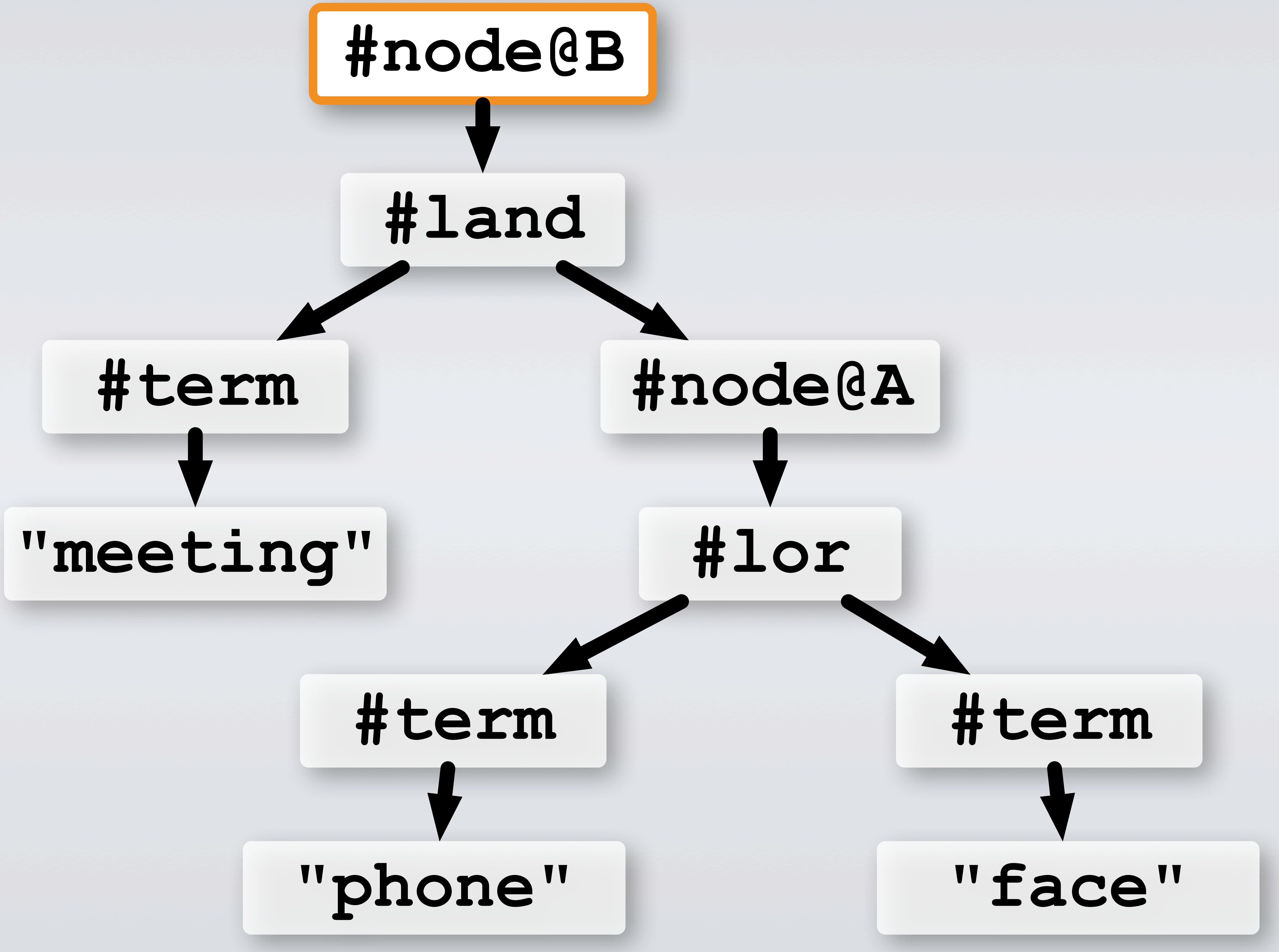
```
[{node,
    {land, [
        {node,
            {lor, [
                {term,{"email","body","face"}, [
                    {node_weight,'node_c@127.0.0.1', 13}
                ]},
                {term,{"email","body", "phone"}, [
                    {node_weight,'node_a@127.0.0.1', 17}
                ]}
            ]},
            'node_a@127.0.0.1'
        },
        {term, {"email","body","meeting"}, [
            {node_weight,'node_b@127.0.0.1', 23}
        ]}
    ]},
    'node_b@127.0.0.1'
}]
```

Execute the Query

Spawn the Query Processes

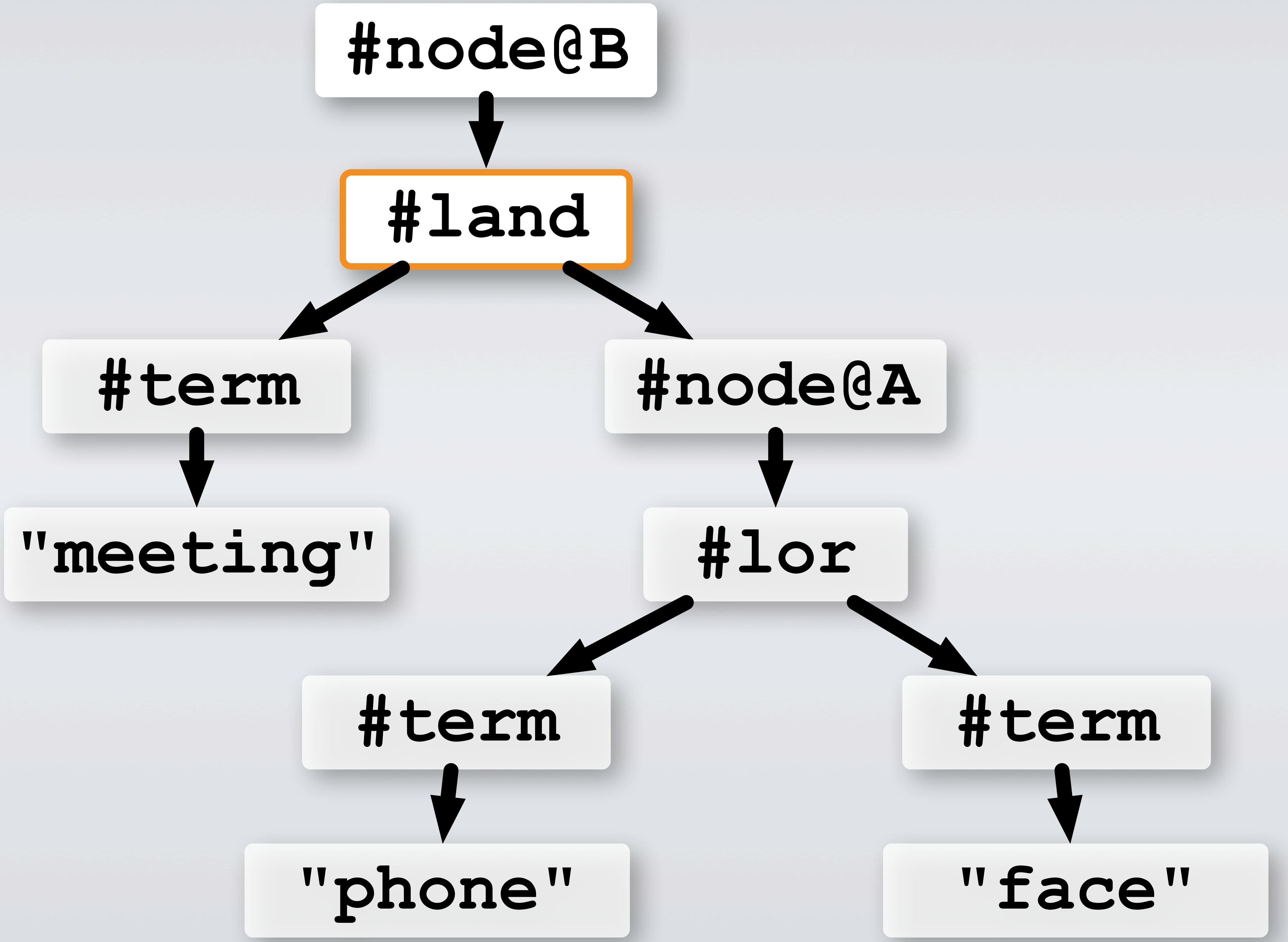


Spawn the Query Processes

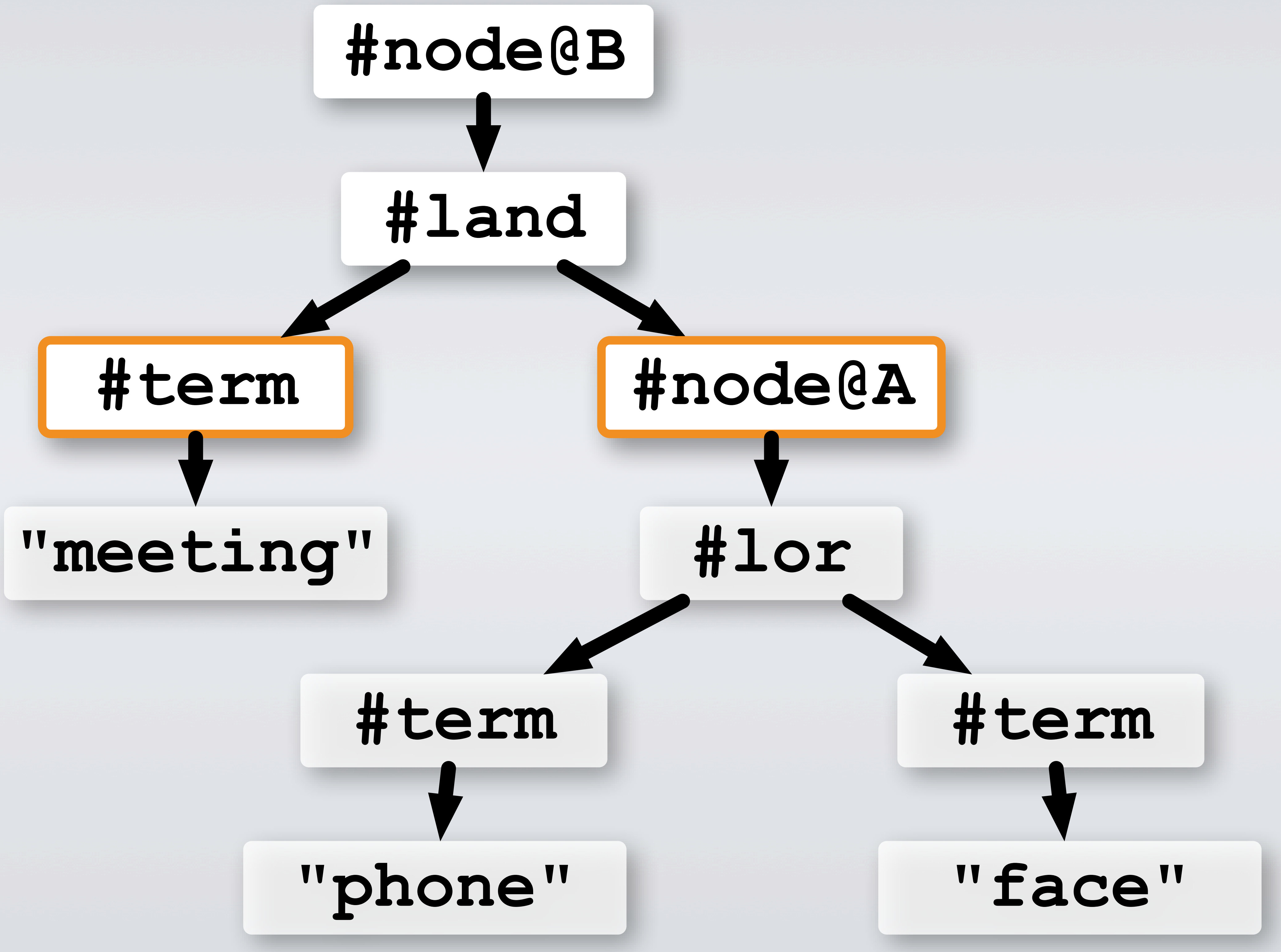


basho

Spawn the Query Processes

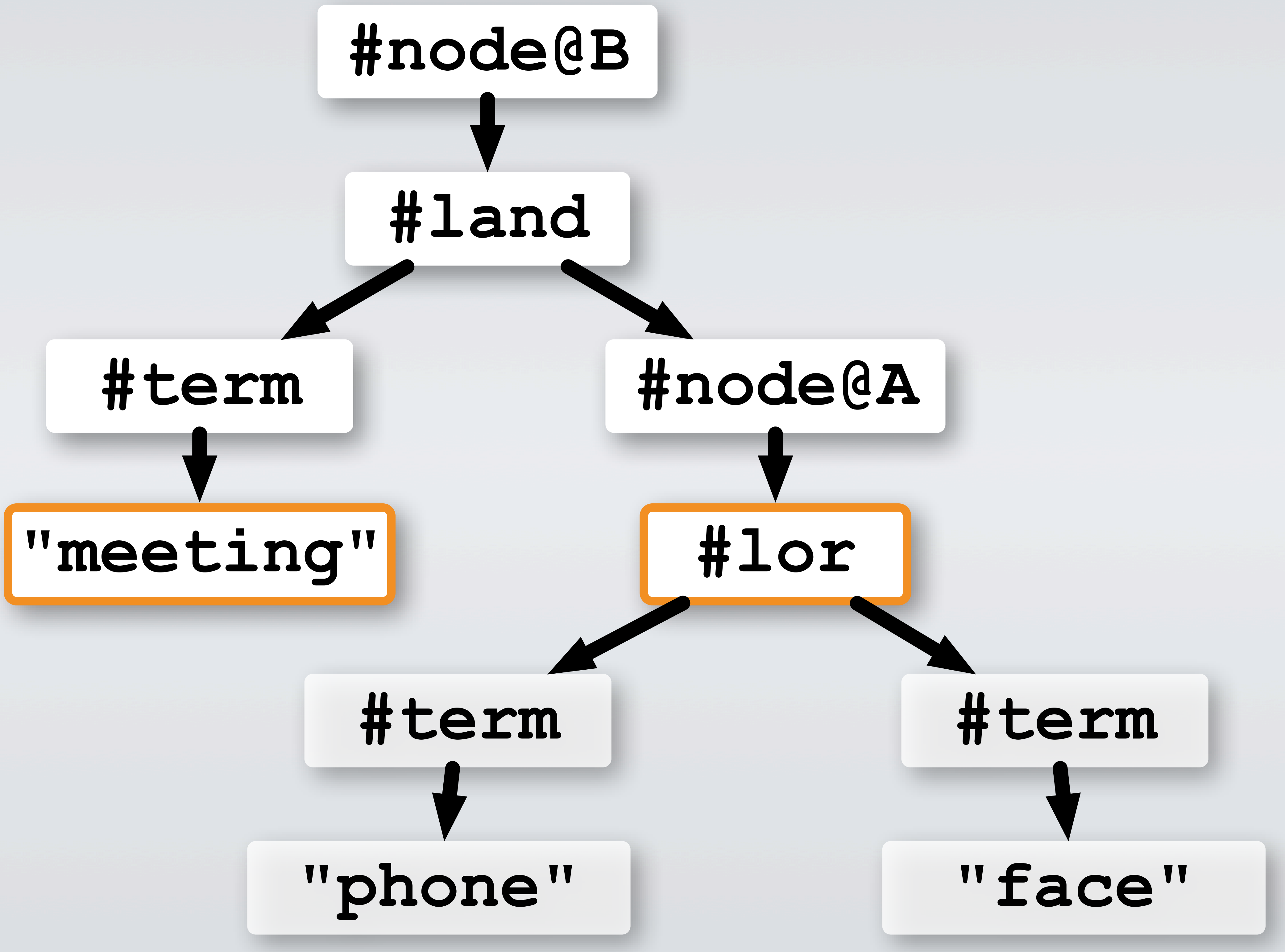


Spawn the Query Processes

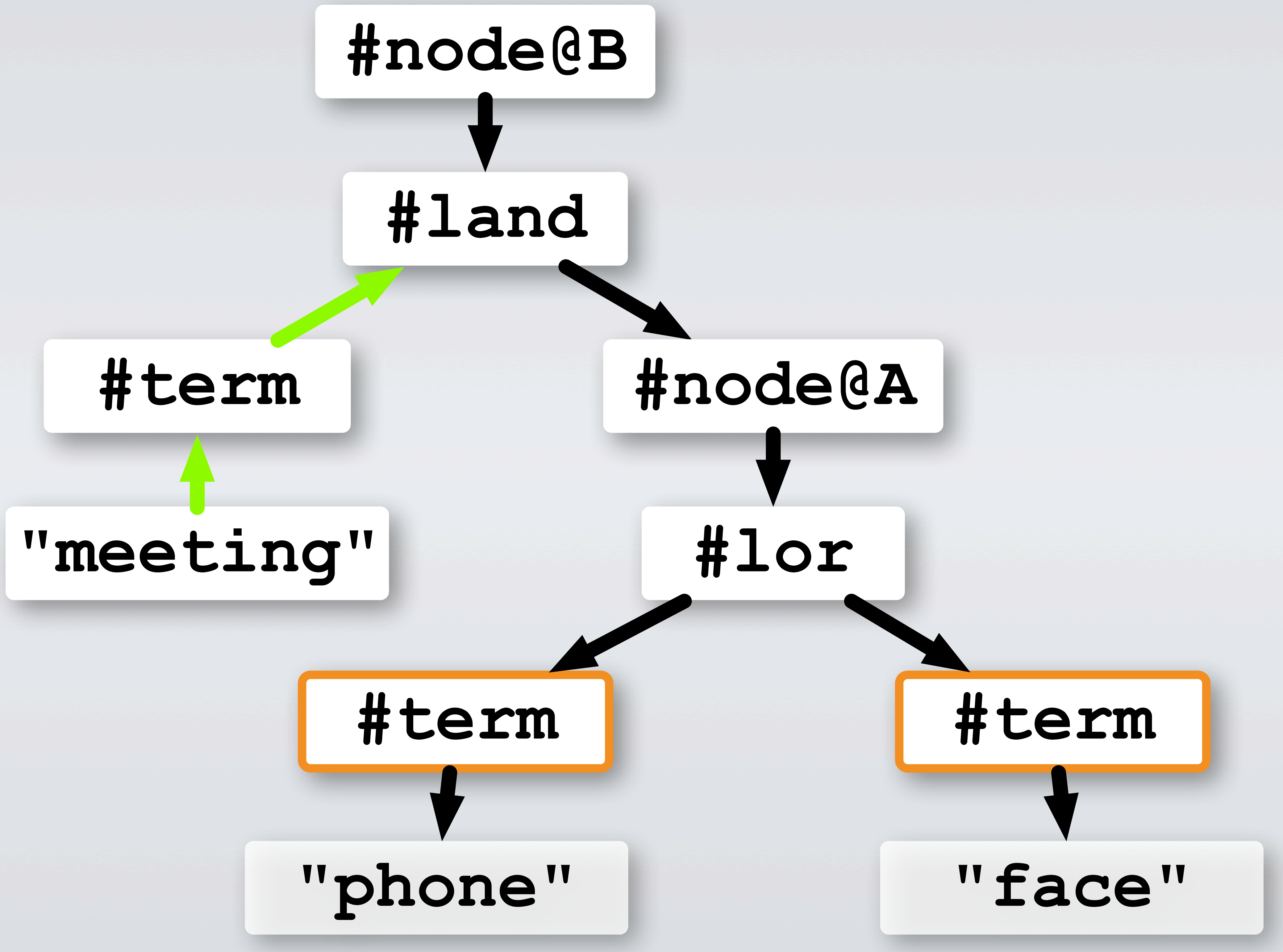


basho

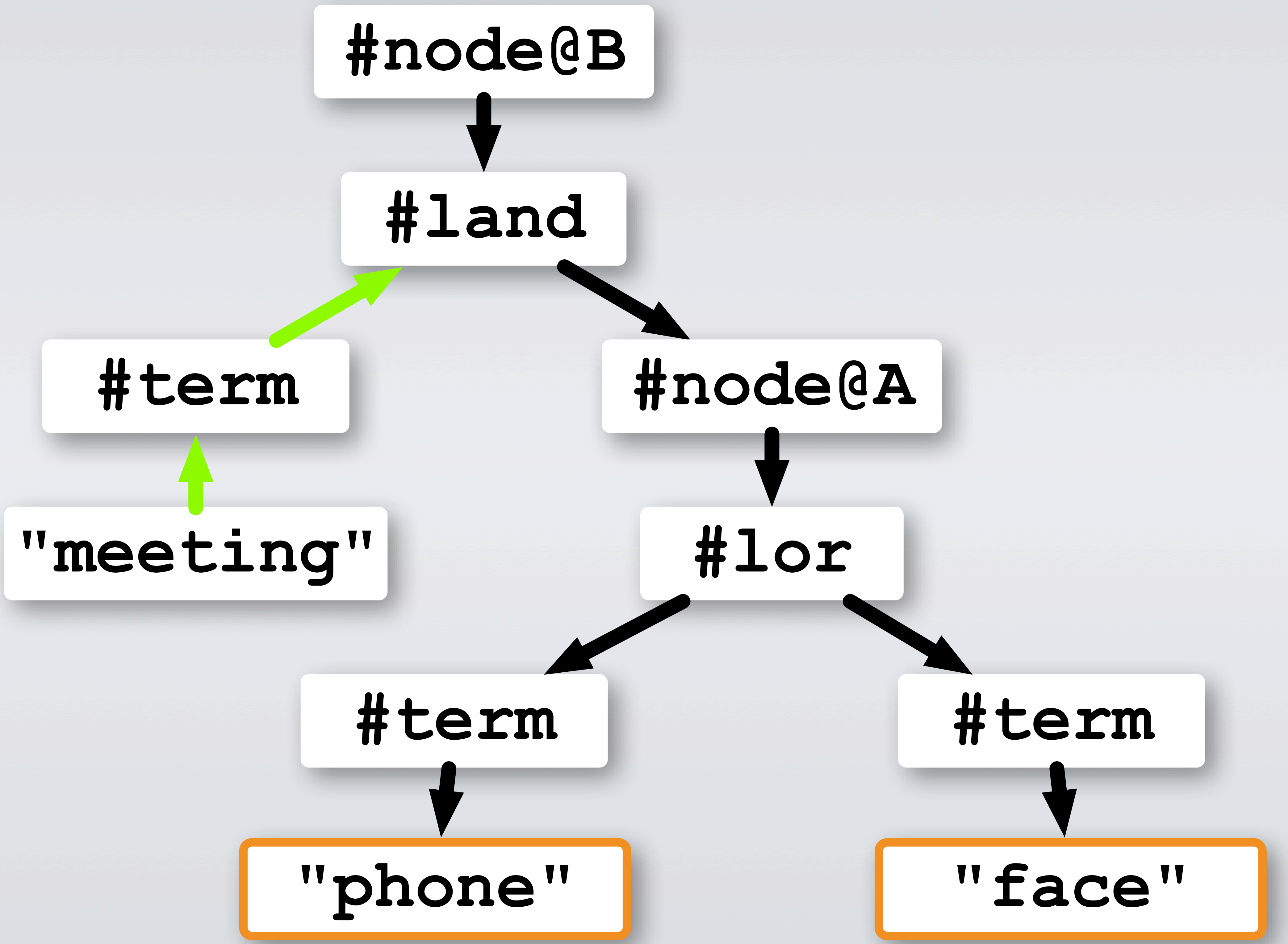
Spawn the Query Processes



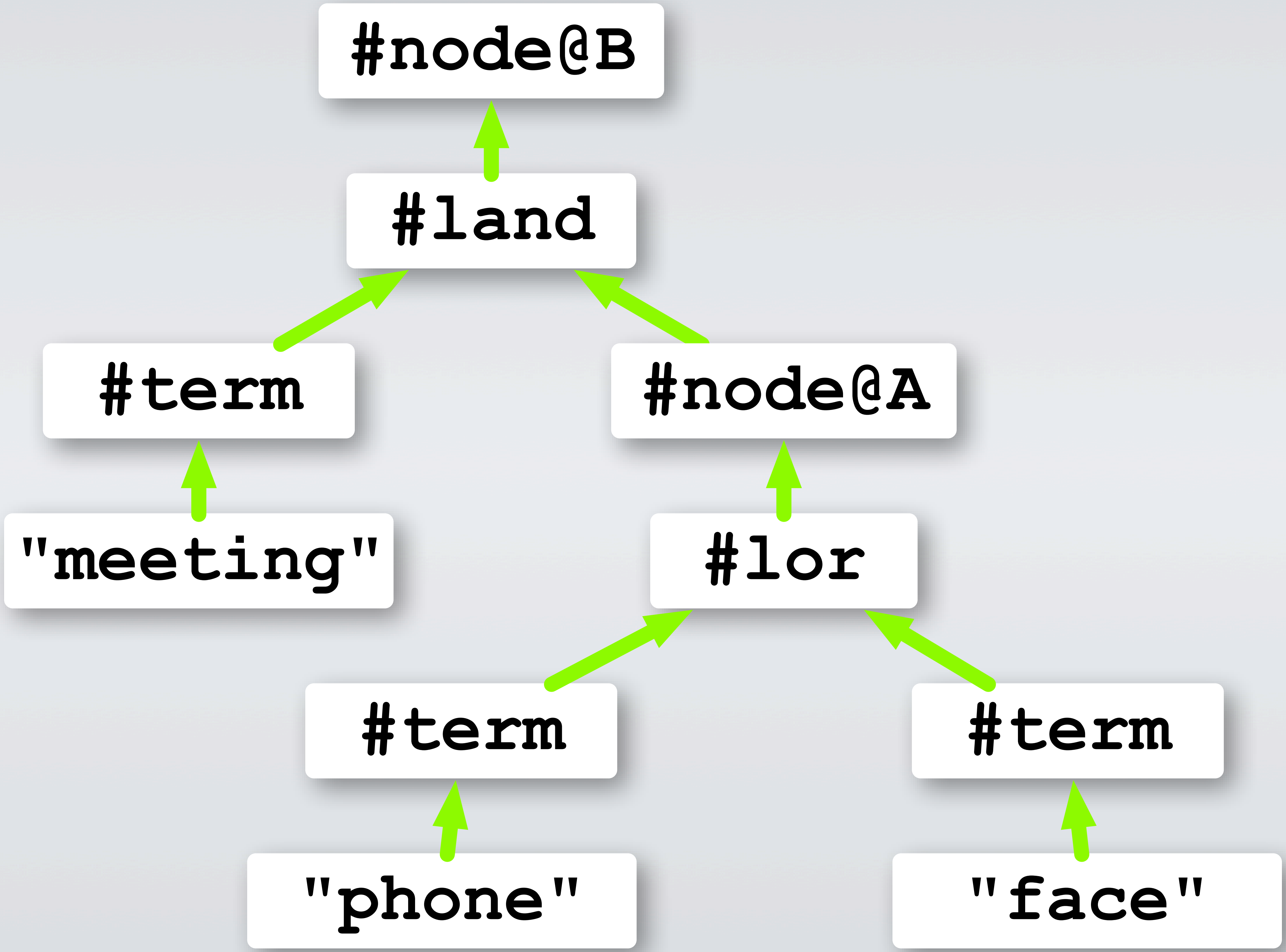
Spawn the Query Processes & Stream the Results



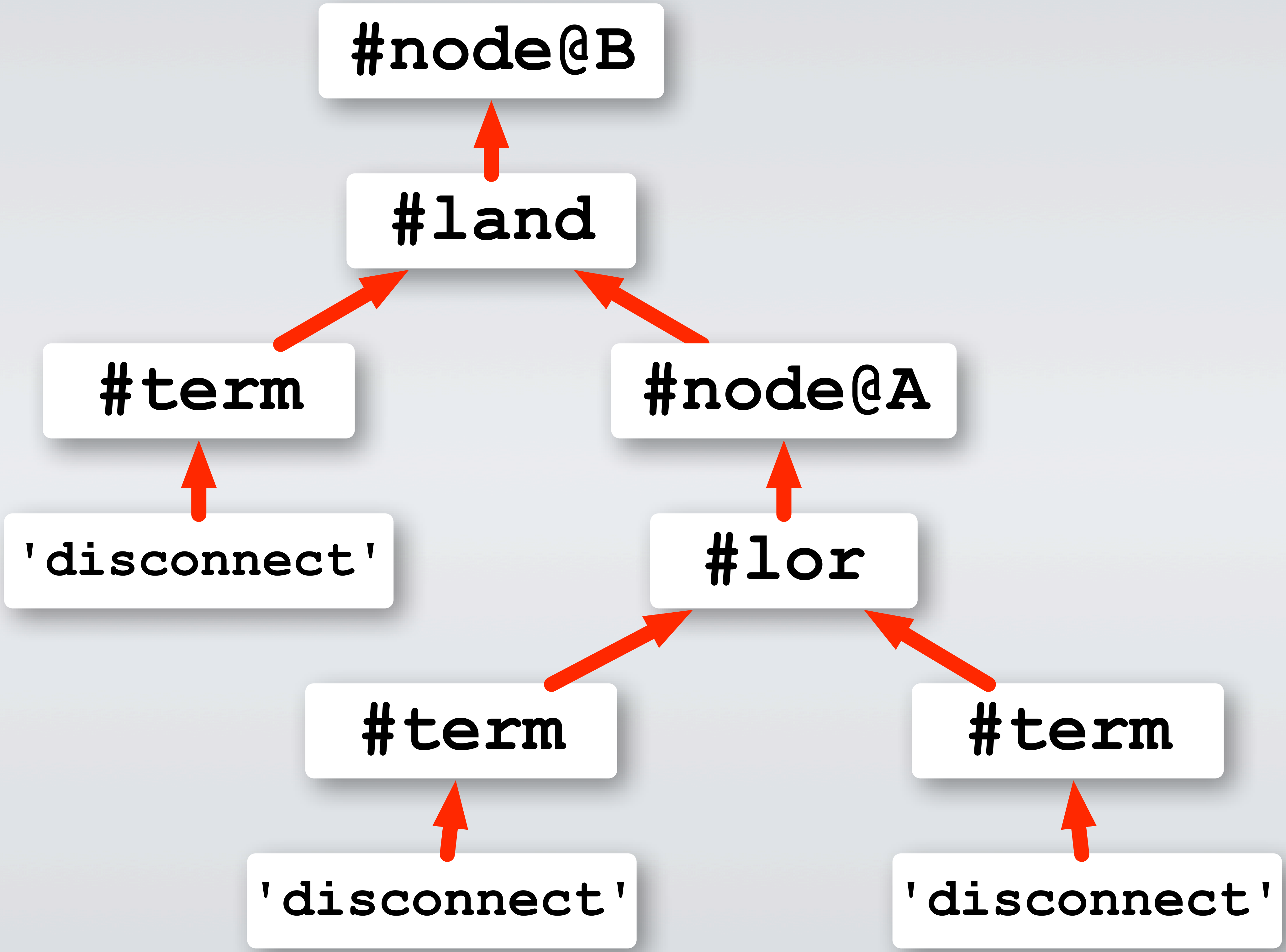
Spawn the Query Processes & Stream the Results



Spawn the Query Processes & Stream the Results



Terminate When Finished



Message Format

The Message Format

```
Message ::
    {results, [Result]} |
    {results, disconnect}

Result ::
    {DocID, Properties}

DocID ::
    term()

Properties ::
    proplist()
```

The Message Format

```
{results, [
    {375, []},
    {961, [{color, "red"}]},
    {155, [{pos, [1,2,5]}]}
]}
```

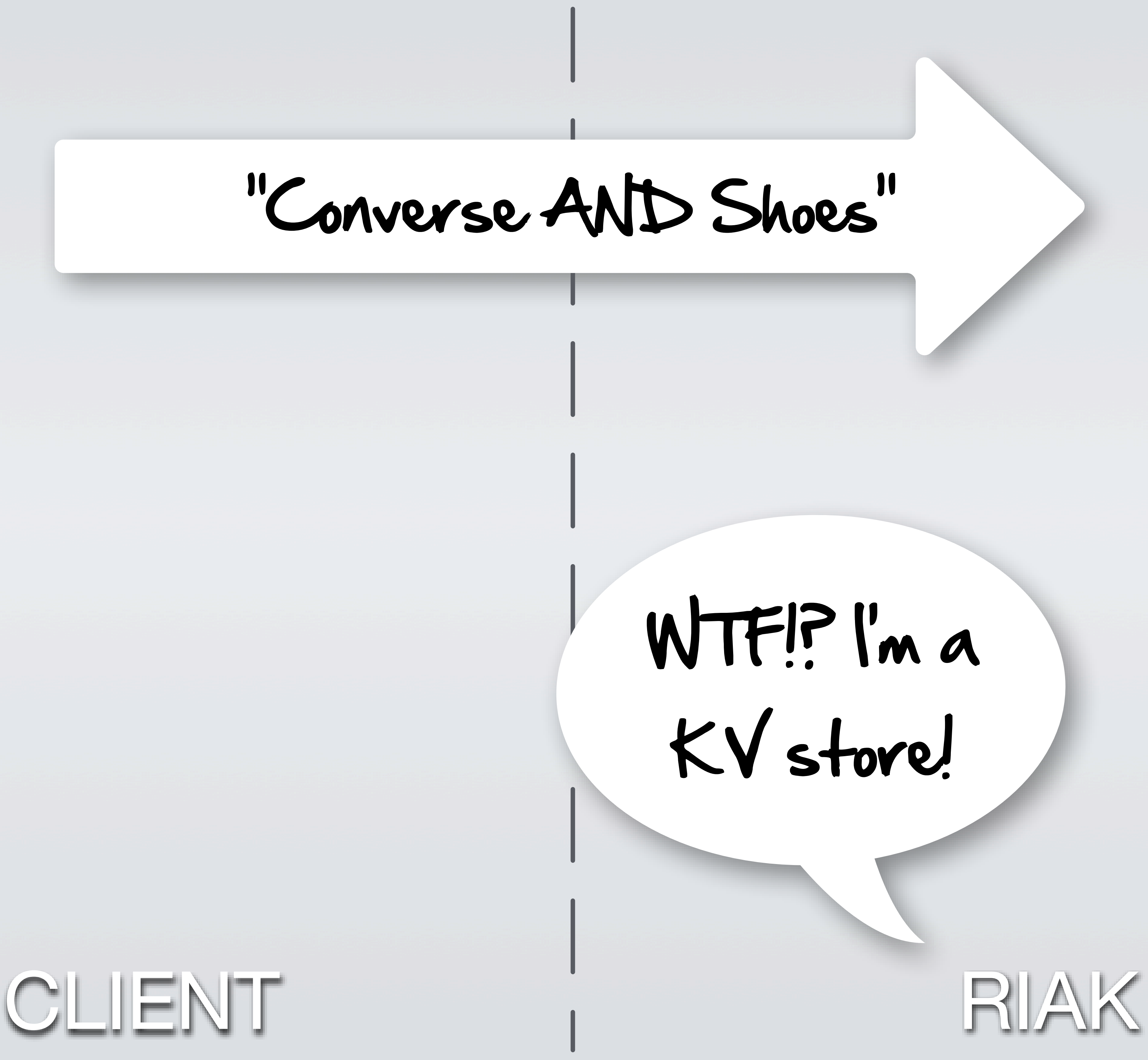
Yay for Erlang!

- Clean lines between load balancing and logic, single- and multi-node look the same
- Easy to create new operators, rapid development of experimental features
- Linked processes make cleanup a breeze
- Significant code reduction over early Java prototypes

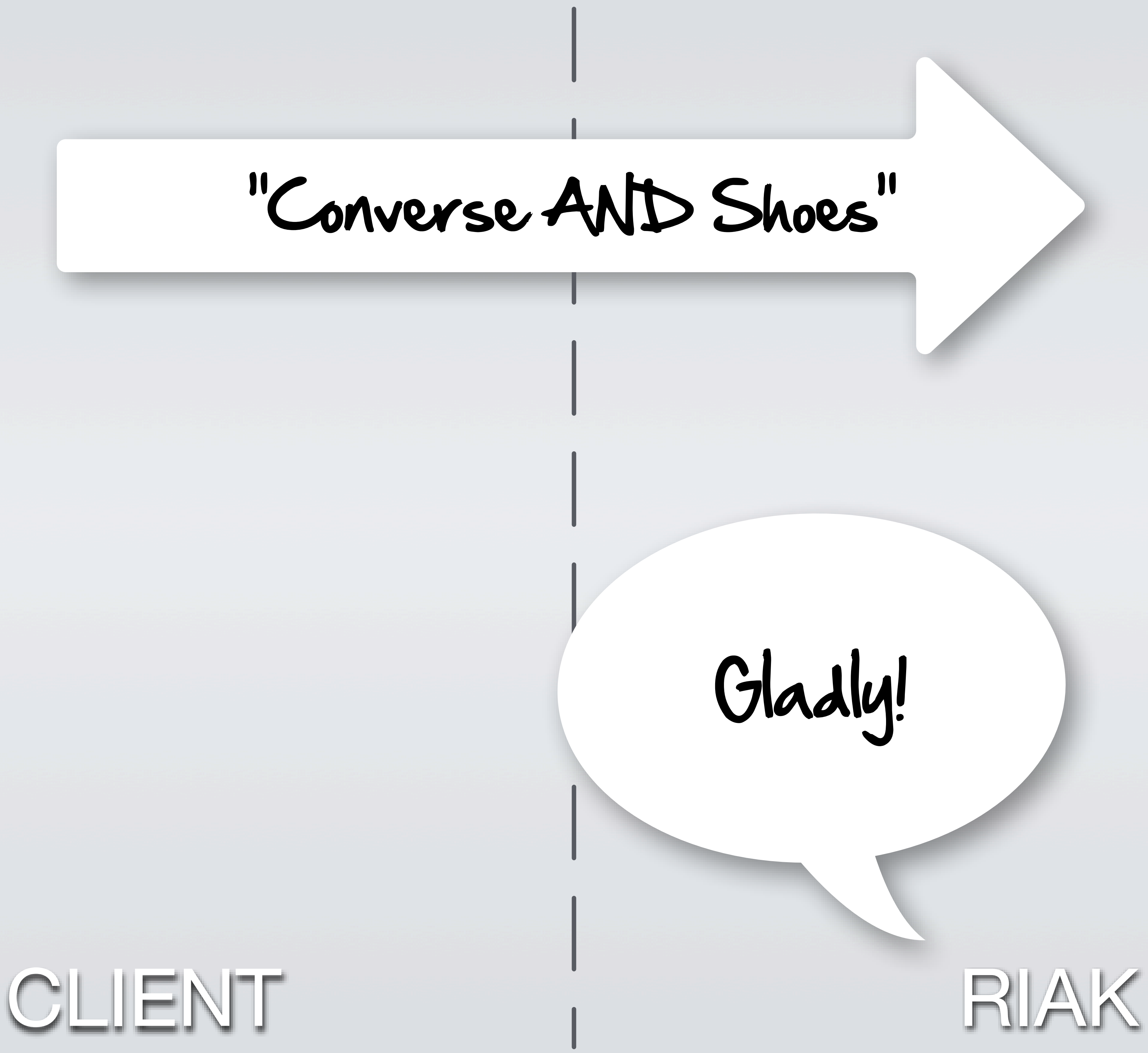
Part Four

Review

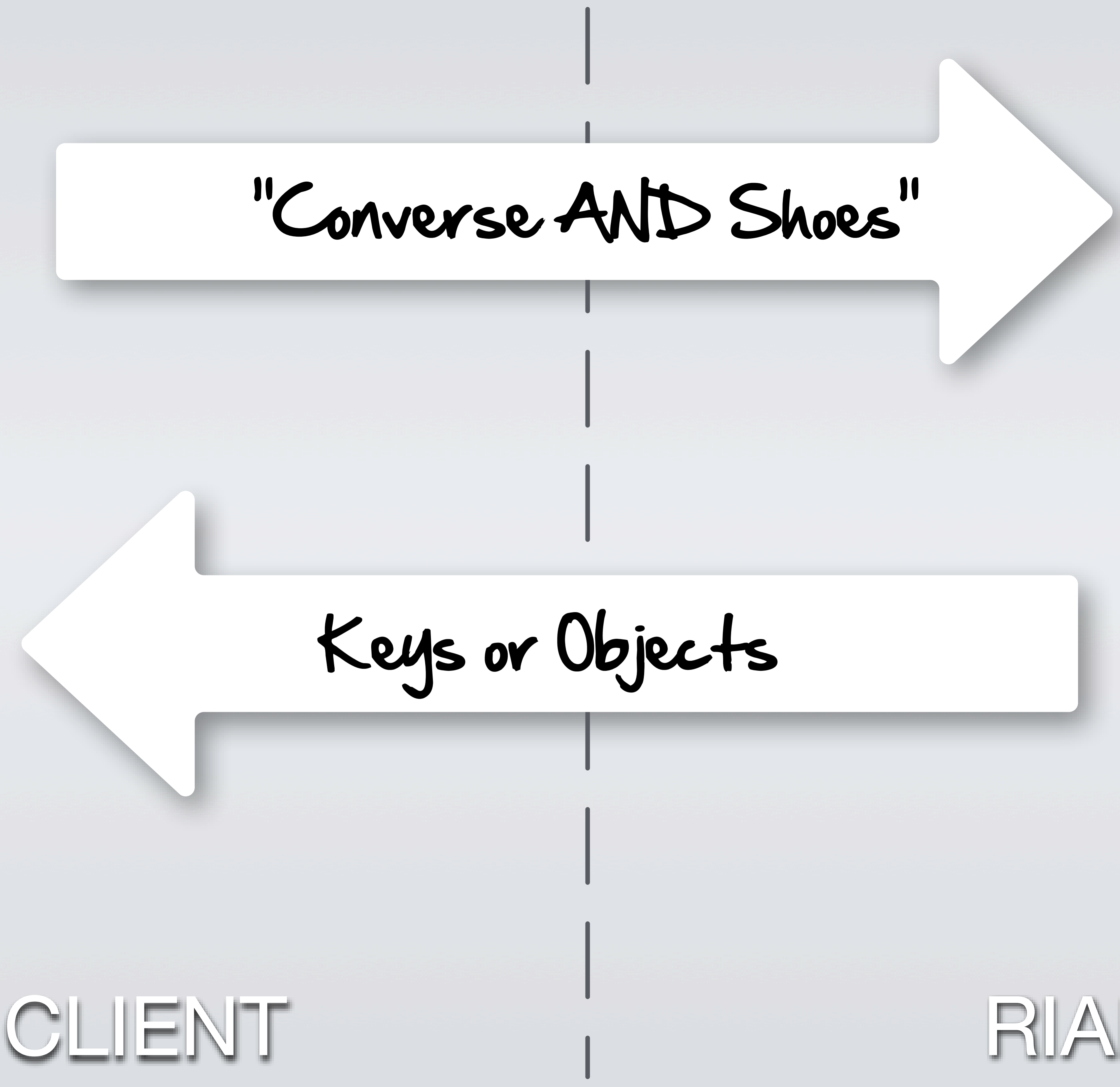
Riak Search turns this...



...into this...



...into this...

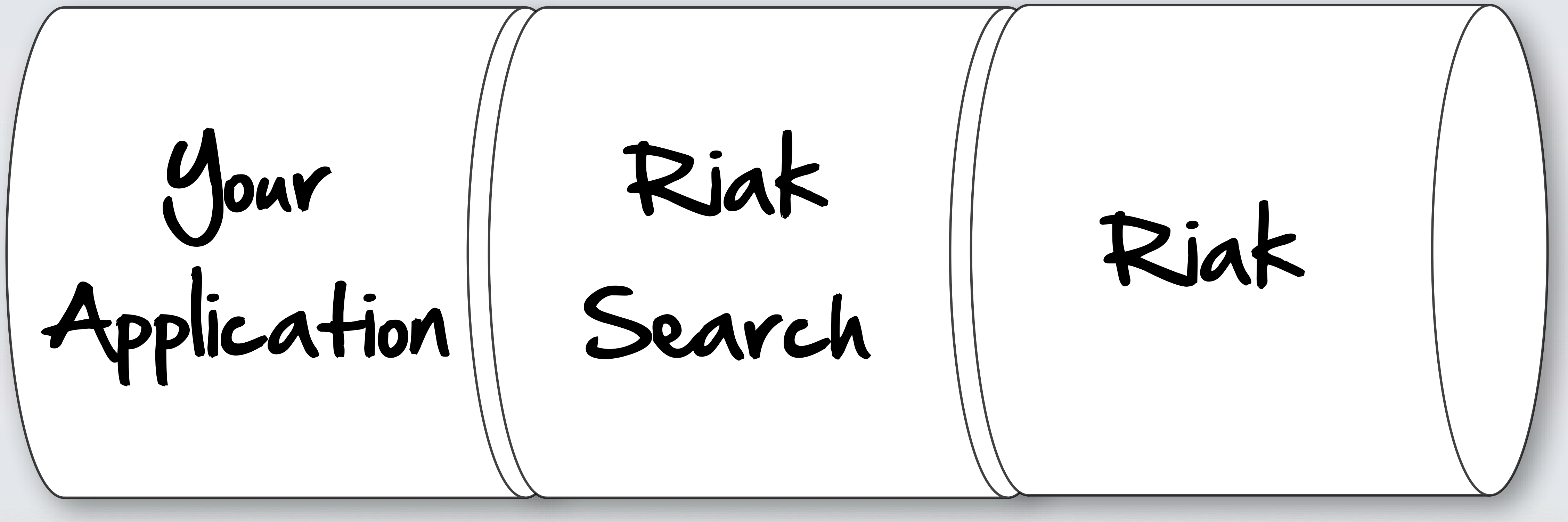


CLIENT

RIAK

basho

...while keeping operations easy.



Thanks! Questions?

basho

Search Team:

John Muellerleile - @jrecursive

Rusty Klophaus - @rklophaus

Kevin Smith - @kevsmith

Currently working with a small set of Beta users.

Open-source release planned for Q3.

www.basho.com