Riak Search

John Muellerleile
Basho Technologies

Erlang Factory 2010
San Francisco

Agenda

- Introduction
- A Search Tale
- Riak Search

“I used to make stuff”

(but now I am a database hacker)

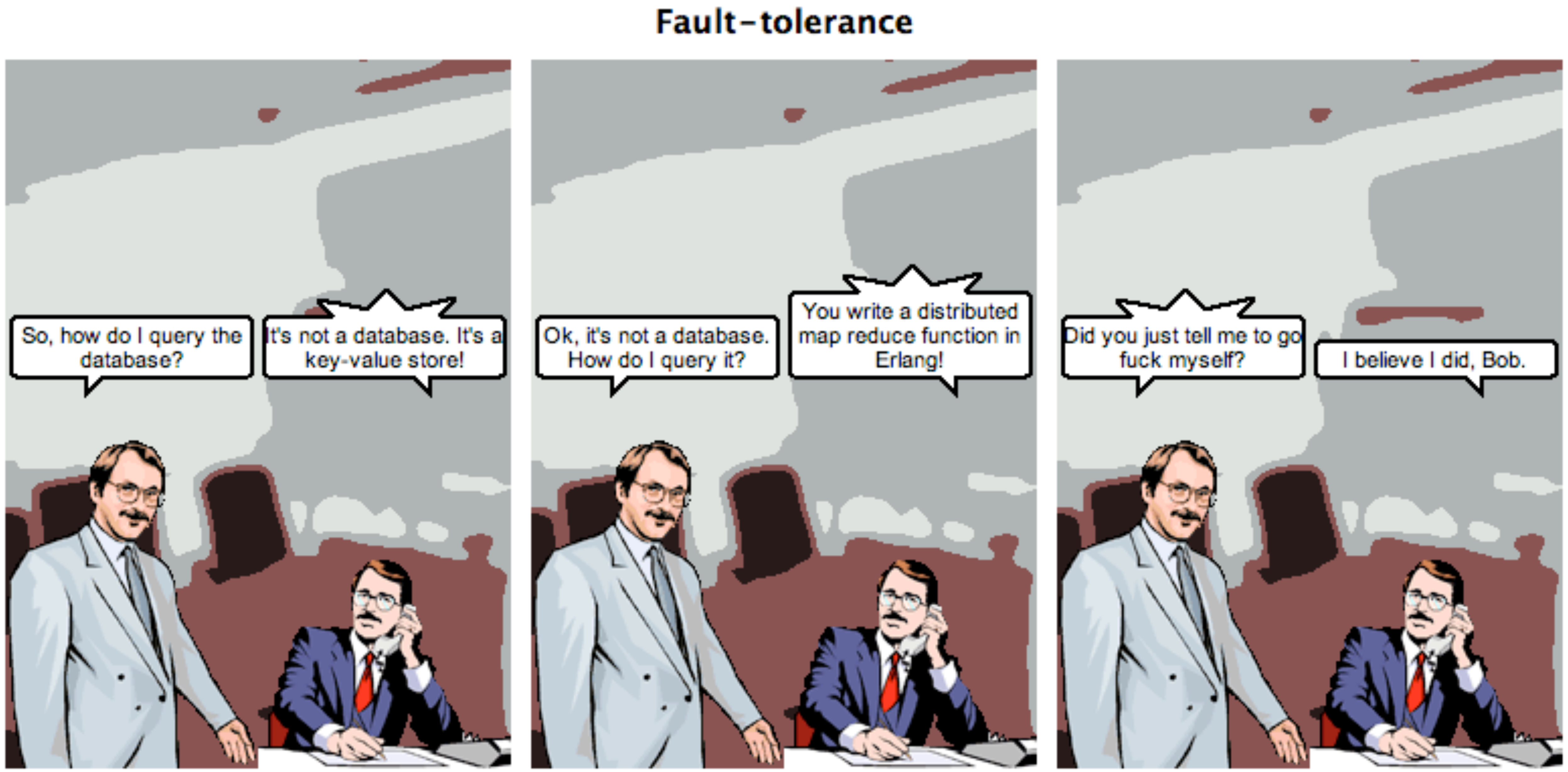
- You are not your primary key
- Having to roll your own indexes = not OK*
- Time spent was not on *developing your app*

* choosing to is different matter altogether

A moment of weakness

“My way of joking is to tell the truth;
it’s the funniest joke in the world”
George Bernard Shaw

How do I query it?


Fault-tolerance
So, how do I query the database?
It's not a database. It's a key-value store!
Ok, it's not a database. How do I query it?
You write a distributed map reduce function in Erlang!
Did you just tell me to go fuck myself?
I believe I did, Bob.

A Search Tale

DIY Search in Three Acts

Act 1: “I Love Lucene”

- “Life is good!”
- Dedicated server + backup
- Not growing quickly, relatively static data set
- Fast & predictable

Act 2: "Cluster Luck"

- "I love a good challenge!"
- A few shards+backups, some clever scripts
- Performance is good enough
- Nobody notices when the indexing master fails, it's just a couple of documents
- Amazon EC2 appreciates your business

Act 3: “SNAFU”

- Lots of shards
- Operational nightmare
- Diminishing returns: indexing, querying
- High “hit by bus” factor
- Scripts no longer viewed as “clever”
- Amazon EC2 holiday cards are now hand-written

Where does it hurt?

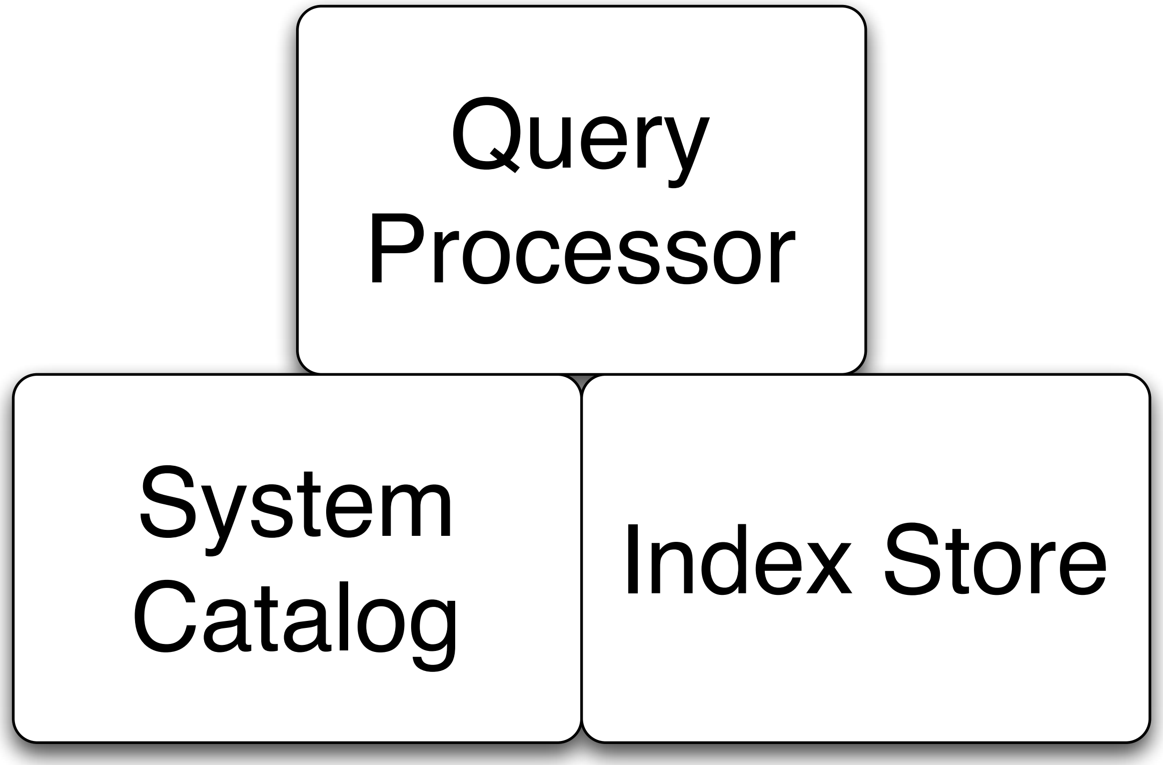
- Complexity: more shards, more problems
- Failure: cases range from “oops” to “*Epic*” (and are not mutually exclusive)
- Diminishing returns: indexing, querying
- Increasing cost: Ops, Dev, Opportunity

Riak Search Goals

- Decentralized: no SPOF
- Distributed: only worry about local data
- Homogeneity: all nodes do the same thing
- Value: Adding nodes adds performance & capacity
- Flexibility: Query platform that's efficient and extensible

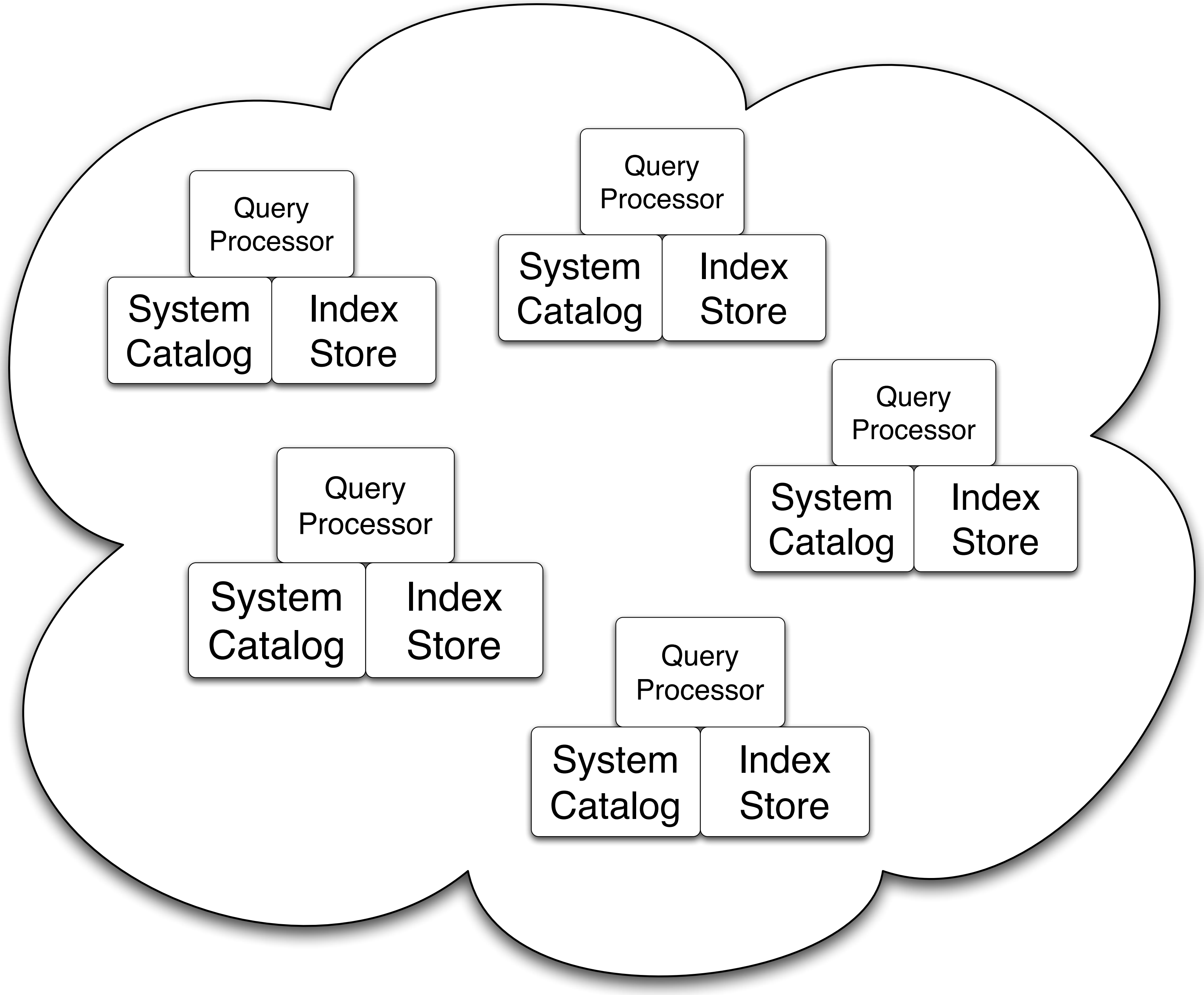
Architecture

Riak Search Node



Keep it simple & foster emergence

No Unique Snowflakes



Index

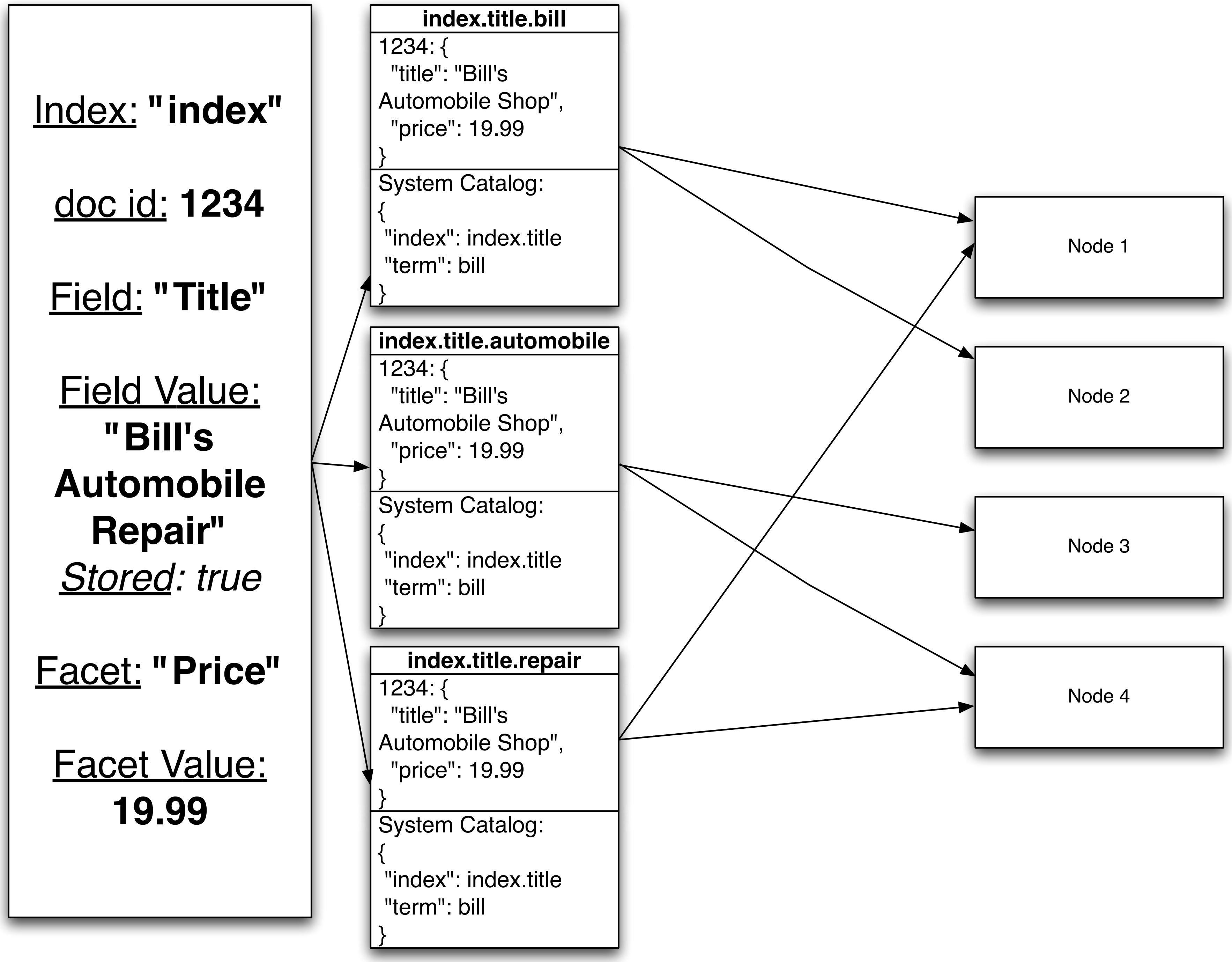
- “Column-*esque*”, use as many as you want
- Every column has a primary key
- Every key points to at least a “doc id”
- Can also store “facets”

System Catalog

- Searchable index of node-local columns
- Can contain arbitrary metadata
- Built to be small & fast; always in-memory
- e.g., query:
  ```
  +index:”products”
  +term:auto*
  +category:”books”
  ```
  result:
  ```
  products.automobile,
  products.auto, products.automata,
  products.autoclave, [...]
  ```

Indexing Crash Course

Tokenize ➡ Term ➡ Column ➡ Store



Index: "index"
doc id: 1234
Field: "Title"
Field Value: "Bill's Automobile Repair"
Stored: true
Facet: "Price"
Facet Value: 19.99
index.title.bill
1234: { "title": "Bill's Automobile Shop", "price": 19.99 }
System Catalog: { "index": index.title "term": bill }
index.title.automobile
1234: { "title": "Bill's Automobile Shop", "price": 19.99 }
System Catalog: { "index": index.title "term": bill }
index.title.repair
1234: { "title": "Bill's Automobile Shop", "price": 19.99 }
System Catalog: { "index": index.title "term": bill }
Node 1
Node 2
Node 3
Node 4

Query Processor

- Parser
- Query Intermediate Language
- Planner
- Executor

Parser

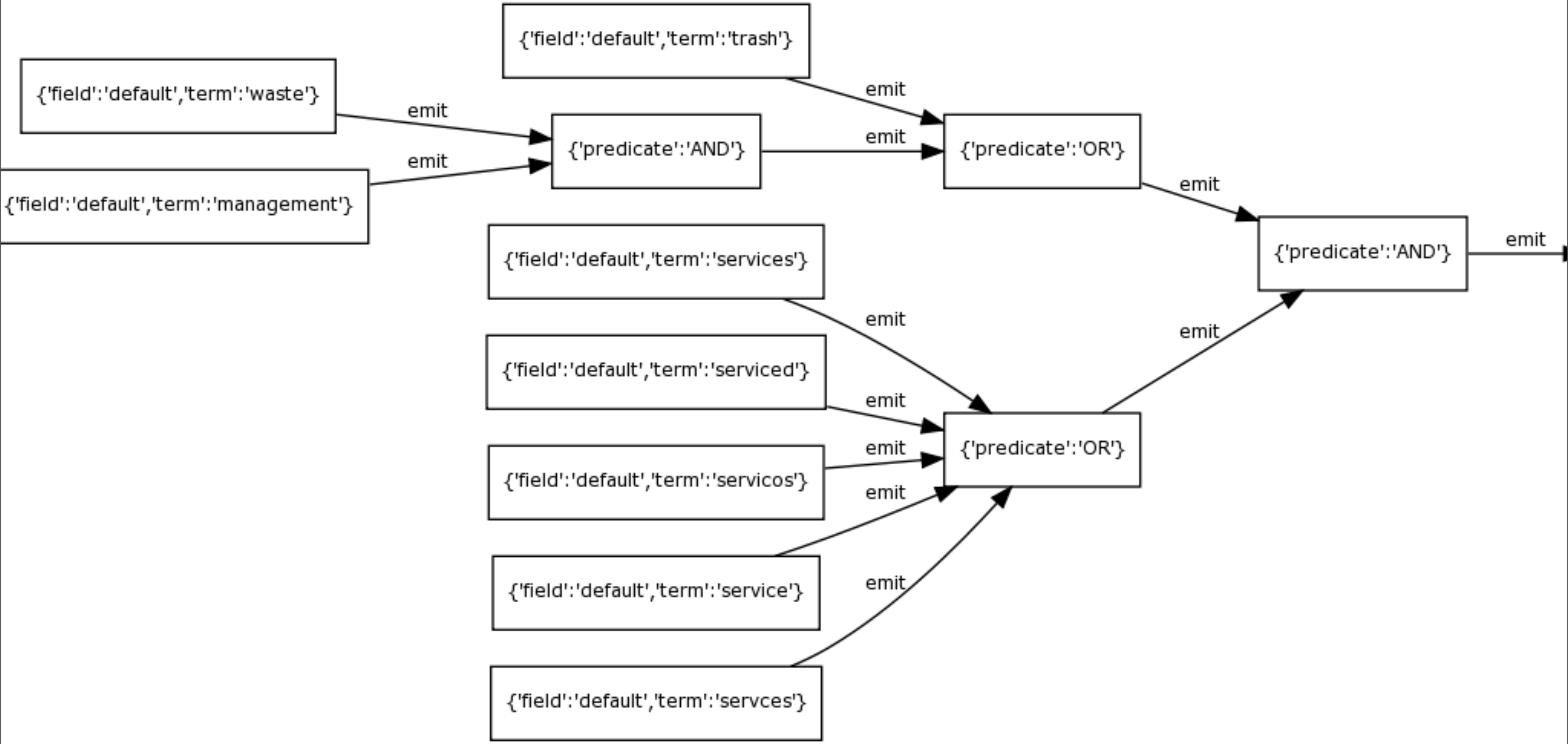
- Parsers produce networks of "*primitives*" the planner and executor can deal with
- Implement multiple query languages
- Use different query languages with the same data in many cases
- Lucene syntax is implemented

QIL

Query Intermediate Language

- QIL is used to express data flows
- A network of simple components: *producers*, *filters* and *accumulators*

You like DAGs?


{'field':'default','term':'trash'}
{'field':'default','term':'waste'}
{'field':'default','term':'management'}
{'predicate':'AND'}
{'predicate':'OR'}
{'predicate':'AND'}
{'field':'default','term':'services'}
{'field':'default','term':'serviced'}
{'field':'default','term':'servicos'}
{'field':'default','term':'service'}
{'field':'default','term':'servces'}
{'predicate':'OR'}
emit

QIL Vertices

- Producers: stream node-local data (`get_all`, ...)
- Filters: *AND*, *OR*, *NOT*, ...
- Accumulators: collect and/or stream results

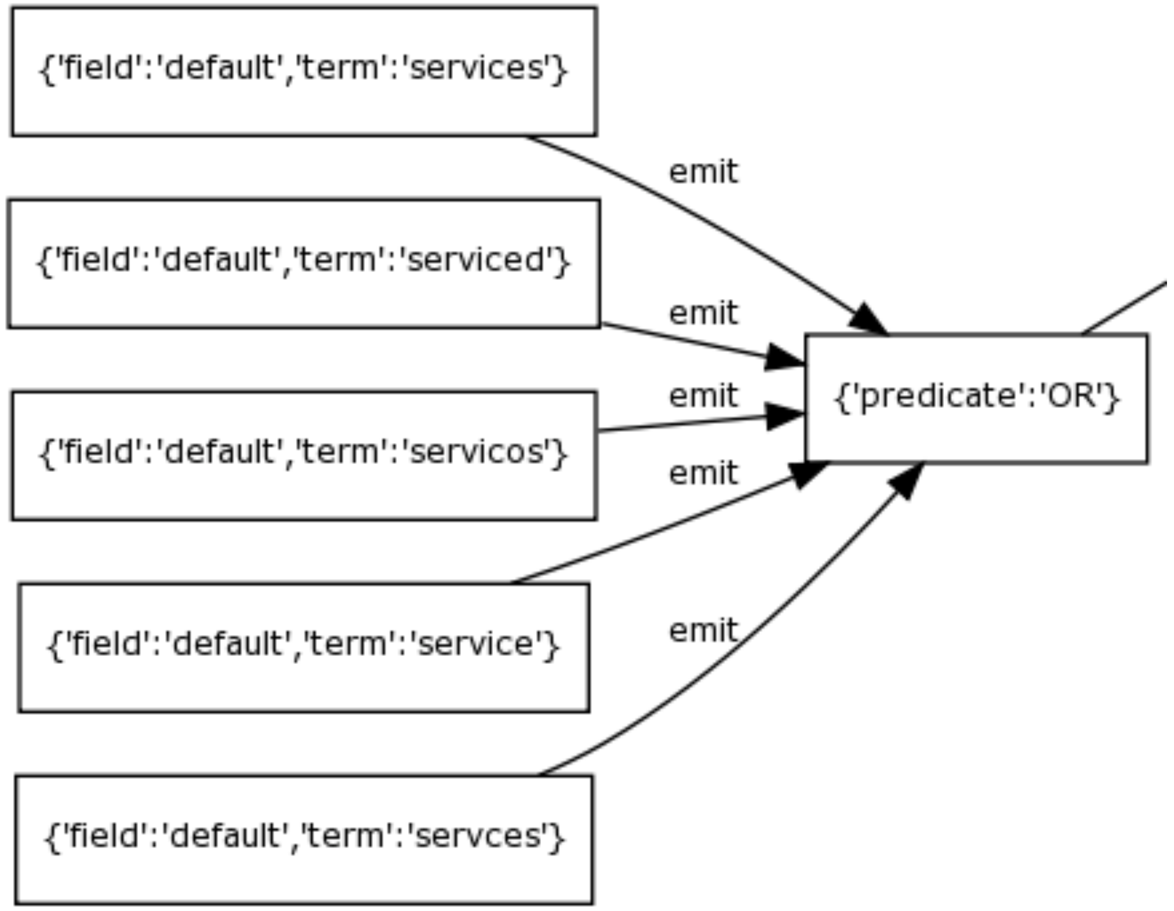
QIL Edges

- A stream of tuples
- One format

Expansions

A Fuzzy Decomposition

services~0.8


{'field':'default','term':'services'}
{'field':'default','term':'serviced'}
{'field':'default','term':'servicos'}
{'field':'default','term':'service'}
{'field':'default','term':'servces'}
emit
emit
emit
emit
emit
{'predicate':'OR'}

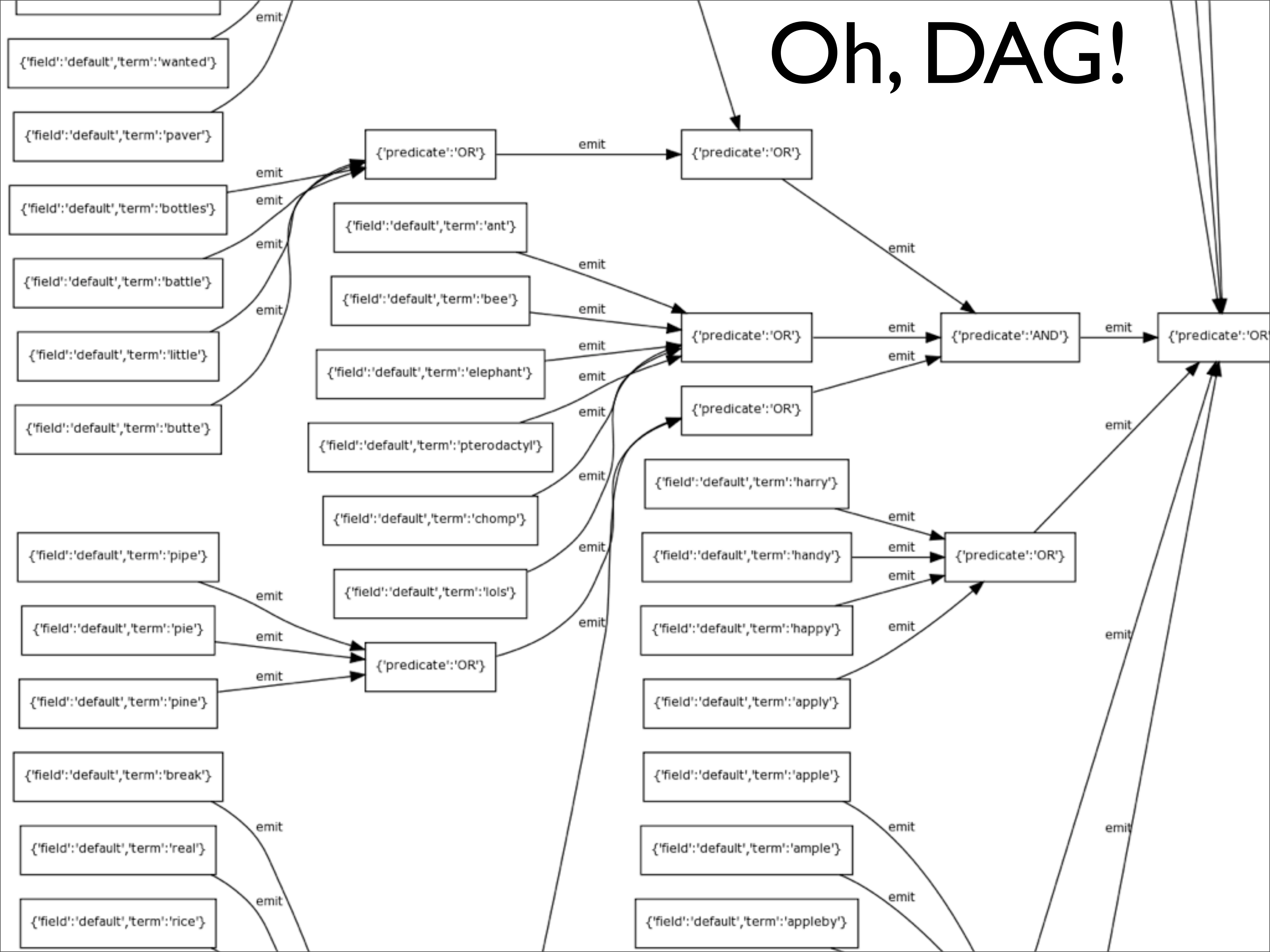
Expansions

- *Range* ➡ "OR" of all existing indexes in the range
- *Wildcard*, *regex* ➡ "OR" of all existing columns that match
- Expansion is an easy way to express a *relationship*, not just string variations (related terms, synonyms, etc.)
- Expansions can be applied ad-hoc

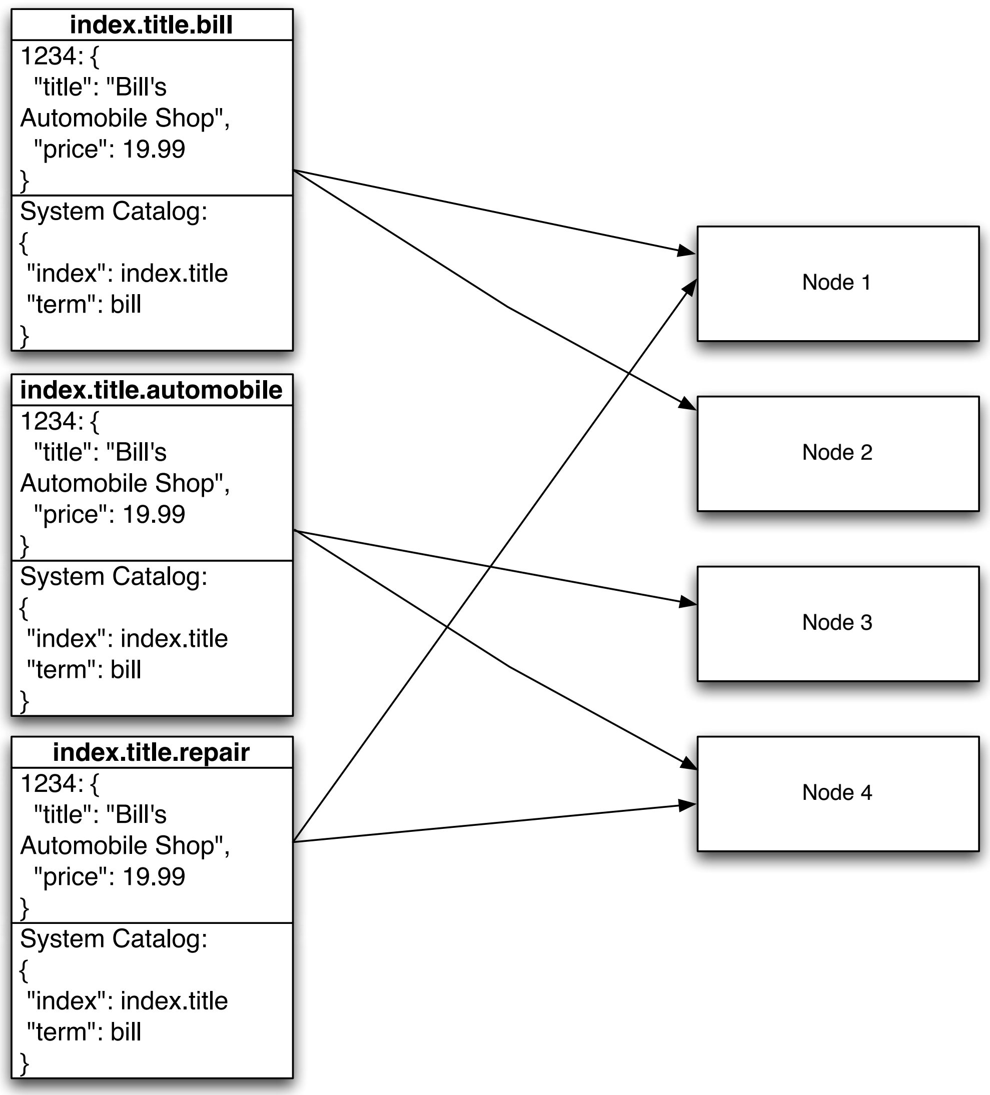
Keep It Simple & Foster Emergence

- All components produce and/or expect the same messages
- Easy to reason about, extend
- *Networks of primitives built up by expansion produce complex behaviors*

Oh, DAG!


{'field':'default','term':'wanted'}
{'field':'default','term':'paver'}
{'field':'default','term':'bottles'}
{'field':'default','term':'battle'}
{'field':'default','term':'little'}
{'field':'default','term':'butte'}
{'field':'default','term':'pipe'}
{'field':'default','term':'pie'}
{'field':'default','term':'pine'}
{'field':'default','term':'break'}
{'field':'default','term':'real'}
{'field':'default','term':'rice'}
{'predicate':'OR'}
{'field':'default','term':'ant'}
{'field':'default','term':'bee'}
{'field':'default','term':'elephant'}
{'field':'default','term':'pterodactyl'}
{'field':'default','term':'chomp'}
{'field':'default','term':'lols'}
{'field':'default','term':'harry'}
{'field':'default','term':'handy'}
{'field':'default','term':'happy'}
{'field':'default','term':'apply'}
{'field':'default','term':'apple'}
{'field':'default','term':'ample'}
{'field':'default','term':'appleby'}
{'predicate':'AND'}
emit

```
title:bill
+title:auto
+price:[18.00 TO 20.00]
```



1. **Locate** columns
2. **Plan** query

2a. optimize for co-located node-local data

2b. optimize filter process placement

3. **Execute** plan

3a. start accumulators

3b. start filters

3c. start producers

QIL-ler Features

- Designed to be extended, customized
- Built to pipeline
- Lots of opportunity for optimization,

  e.g., *logically factor*: favor node-coincident indexes for expansions, even partially, to reduce unnecessary tuple streams

Why Erlang?

- Queries execute as process networks
- Each process an automaton, shared nothing
- Stability in a storm of short lived processes
- Sane ways to reason about and manage them
- Erlang: a perfect fit!

Project Status

- Prototype being supported for a small group of users, beta program is closed
- Production version has been in the works for awhile, though no availability date (yet)
- It will be open source (Apache)
- *I can’t wait* to get it into your hands!

Thanks!

- direct: johnm at basho dot com
- twitter: @jrecursive
- inquiry: riak@basho.com
- Basho Technologies: http://basho.com/
- search team: @rklophaus @kevsmith @hobbyist @dizzyco @jon_meredith