# Yokozuna

## The future of distributed search in Riak

Basho Technologies
Christian Dahlqvist (christian@basho.com)

$ whoami

Name: Christian Dahlqvist

Title: Client Services Engineer

Company: Basho Technologies

Email: christian@basho.com

$./presentation

# define: Yokozuna

A grand champion sumo wrestler

"the official title of the highest rank in Sumo"

In Riak,

- The next generation of search technology
- Direct integration with Solr and Lucene

# Yokozuna Project

- Under Development for ~9 months (so far)
- Engineered by Ryan Zezeski (@rzezeski)
  - Senior Engineer at Basho
    - Designed and developed riak_search
- Recently merged into Basho Org. on GitHub
- Currently 0.5 Status
  https://github.com/basho/yokozuna/blob/master/docs/


riak

# What is Riak?

- Key-Value Store + Extras
- Distributed, horizontally scalable
- Fault-tolerant
- Highly-available
- Built for the Web
- Inspired by Amazon's Dynamo

# Key-Value

- Simple operations - GET, PUT, DELETE
- Value is opaque (mostly), with metadata
- Extras
- Secondary Indexes (2i)
- Links
- Full-text search [riak_search] (optional)
- Map/Reduce

bucket

| key | value |
| --- | --- |
| key | value |
| key | value |
| key | value |

| christian | {first: “Christian”, last: “Dahlqvist”...} |
| --- | --- |
| Metadata | |

# Distributed & Horizontally Scalable

- Default Configuration is optimized for a cluster
- Query load and data are spread evenly
- Add more nodes and get more:
  - ops/second
  - storage capacity
  - compute power (for Map/Reduce)

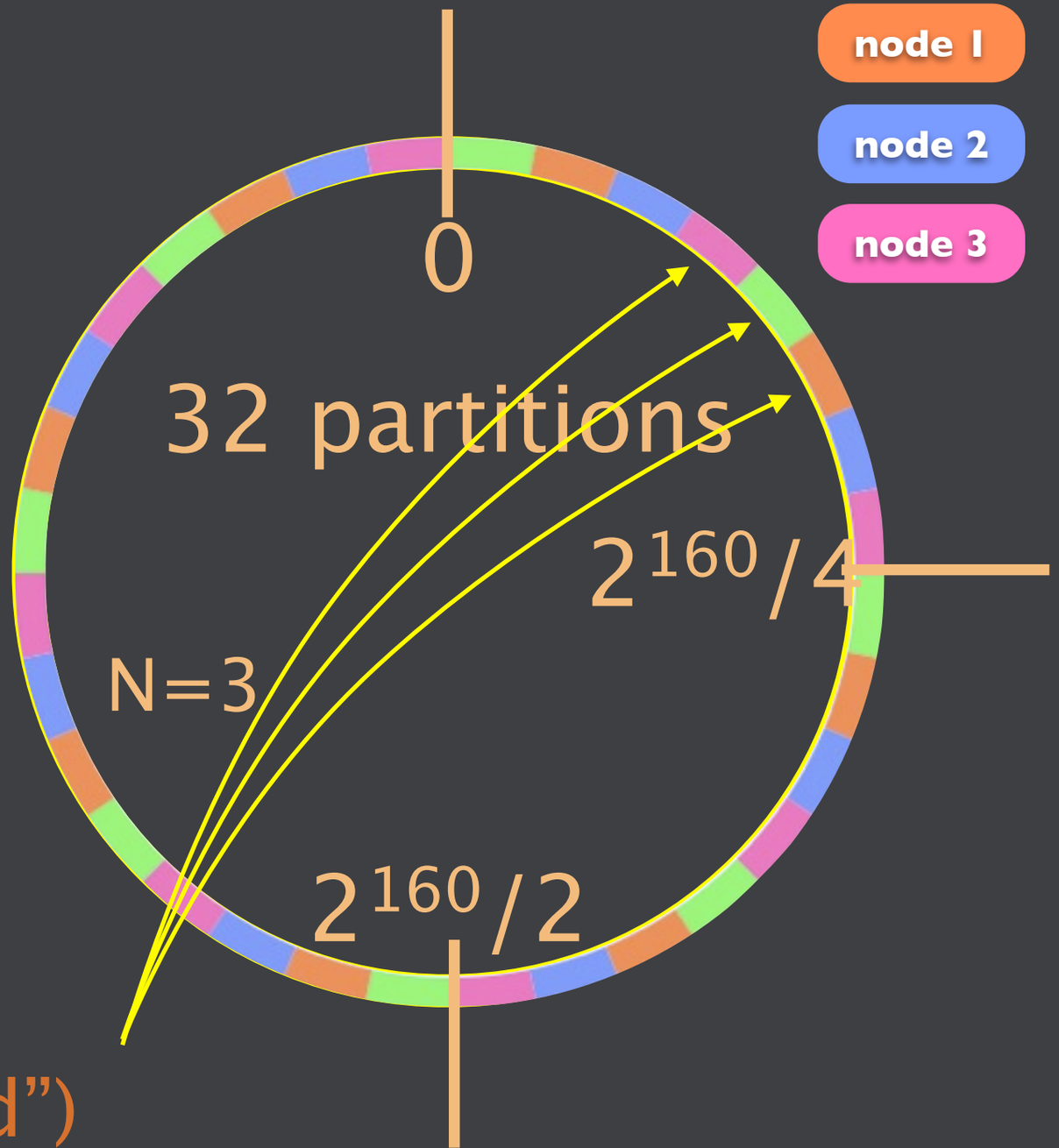
# Inspired by Amazon Dynamo

- Masterless, peer-coordinated replication
- Consistent hashing
- Eventually consistent
- Quorum reads and writes
- Anti-Entropy - Read Repair, Hinted Handoff and Active Anti-Entropy

# Riak Ring

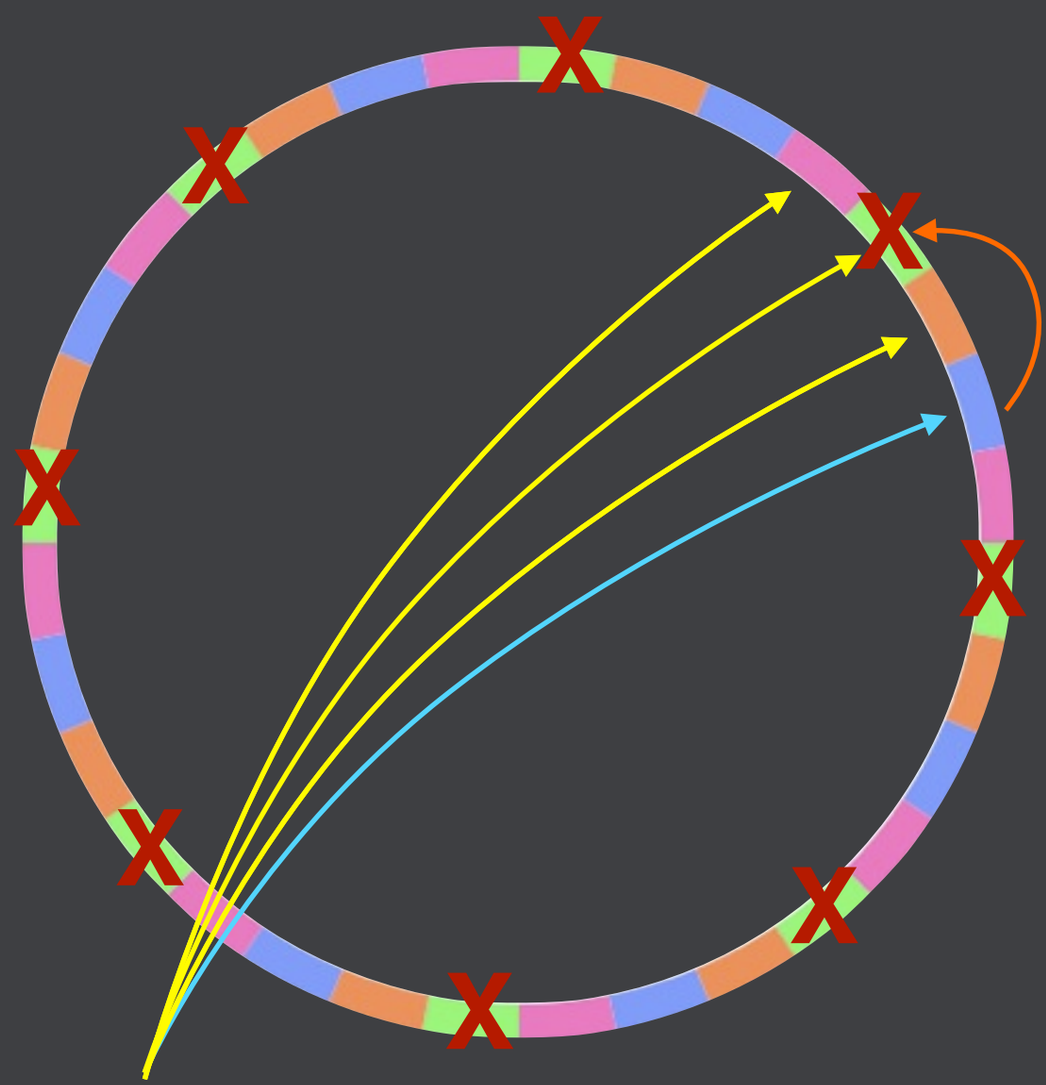
- 160-bit integer keyspace
- divided into fixed number of evenly-sized partitions
- partitions are claimed by nodes in the cluster
- replicas go to the N partitions following the key

hash(“user_id”)



# Disaster Scenario

- Node fails
- Requests go to fallback
- Node comes back
- “Handoff” – data returns to recovered node
- Normal operations resume



hash(“user_id”)

# Anti-Entropy

- Read-repair corrects inconsistencies on read only.
- Active Anti-Entropy uses Merkle trees to compare data in partitions and periodically ensure consistency.
- Active Anti-Entropy runs as a background process

# Riak Use Cases

- Reliability, flexibility, scalability
- Retail Data
- Session Data
- Serving Advertising
- Log and Sensor Data
- Wherever low latency increases revenue

# Back to Yokozuna...

# What is Solr?

- Advanced Full-Text Search Capabilities
- Optimized for High Volume Web Traffic
- Standards Based Open Interfaces - XML, JSON and HTTP
- Near Real-time indexing
- Flexible and Adaptable with XML configuration
- Extensible Plugin Architecture
- Built on Lucene

# What is Lucene?

- High performance indexing, over 150GB/hour on modern hardware
- small RAM requirements -- only 1MB heap
- incremental indexing as fast as batch indexing
- index size roughly 20-30% the size of text indexed

# What is Lucene?

- ranked searching -- best results returned first
- many powerful query types: phrase queries, wildcard queries, proximity queries, range queries and more
- fielded searching (e.g. title, author, contents)
- sorting by any field
- multiple-index searching with merged results
- allows simultaneous update and searching
- flexible faceting, highlighting, joins and result grouping
- fast, memory-efficient and typo-tolerant suggesters

# Turn-Key Integration

- Solr bundled with Riak, zero-configuration and management to use
- supervise Solr process, start/stop/restart
- expose and present canonical Solr query interface
- use existing Solr clients to query Riak
  (take advantage of existing libraries and ecosystem)

# The Application

- An Erlang application
- Made up of process and library modules
- Has a supervision tree
- Sits alongside Riak K/V

# Intermediary

- Converts K/V data into Solr documents
  - Introspection via an “Extractor”
- Translates Solr queries to distributed Solr queries
- Constantly communicates with K/V to verify object/index divergence

# Why another Search?

**riak_search**

- Users confused/expected solr query support
- Less rich features/analyzers/language support
- Bad performance/resource usage for certain types of queries
- Basho is not focused on innovation in search technology

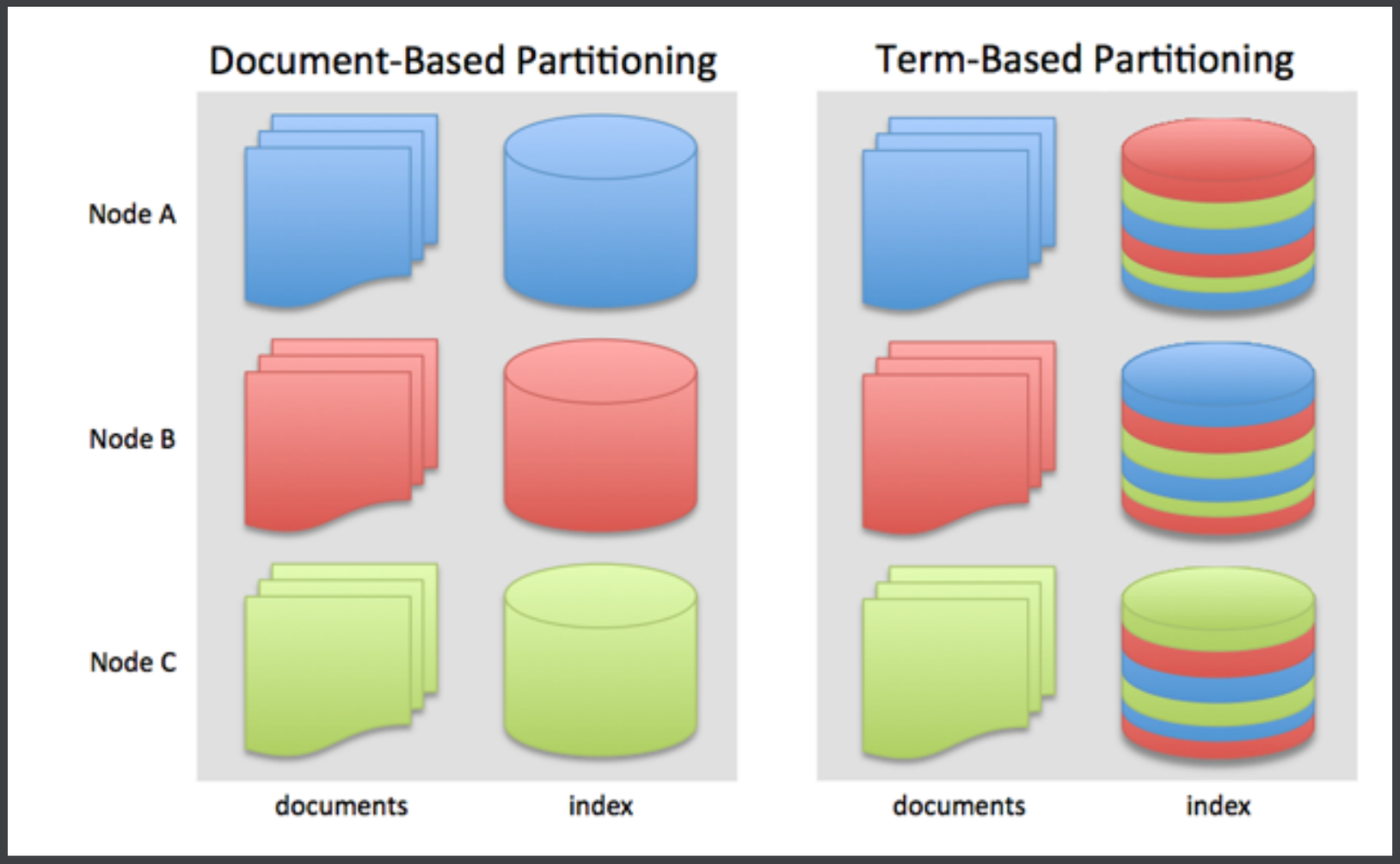
# Term-based Partitioning

- Indexes partitioned by the “term”s queried on
- Term “A” will appear on machine A even if the data is on machine B
- Great for single term queries
  - Only asks one node for the answer
  - Short-queries, or queries with disjunctions perform very well

# Document-based Partitioning

- Indexes live alongside the data being indexed
- Machine A stores indexes for all of its data
- Machine B stores indexes for all of its data
- To query the entire data set both indexes must be accessed
- Term-based partitioning can lead to spikey index distribution

# Document vs Term




http://blog.clipboard.com/2012/03/18/0-Milking-Performance-From-Riak-Search

# Why Solr?

- Great analyzer/language support (more than 30 languages out of the box)
- Features; ranking, faceting, highlighting, geospatial queries, and much more.
- Built upon great foundations, Lucene
- Active community innovating in search space

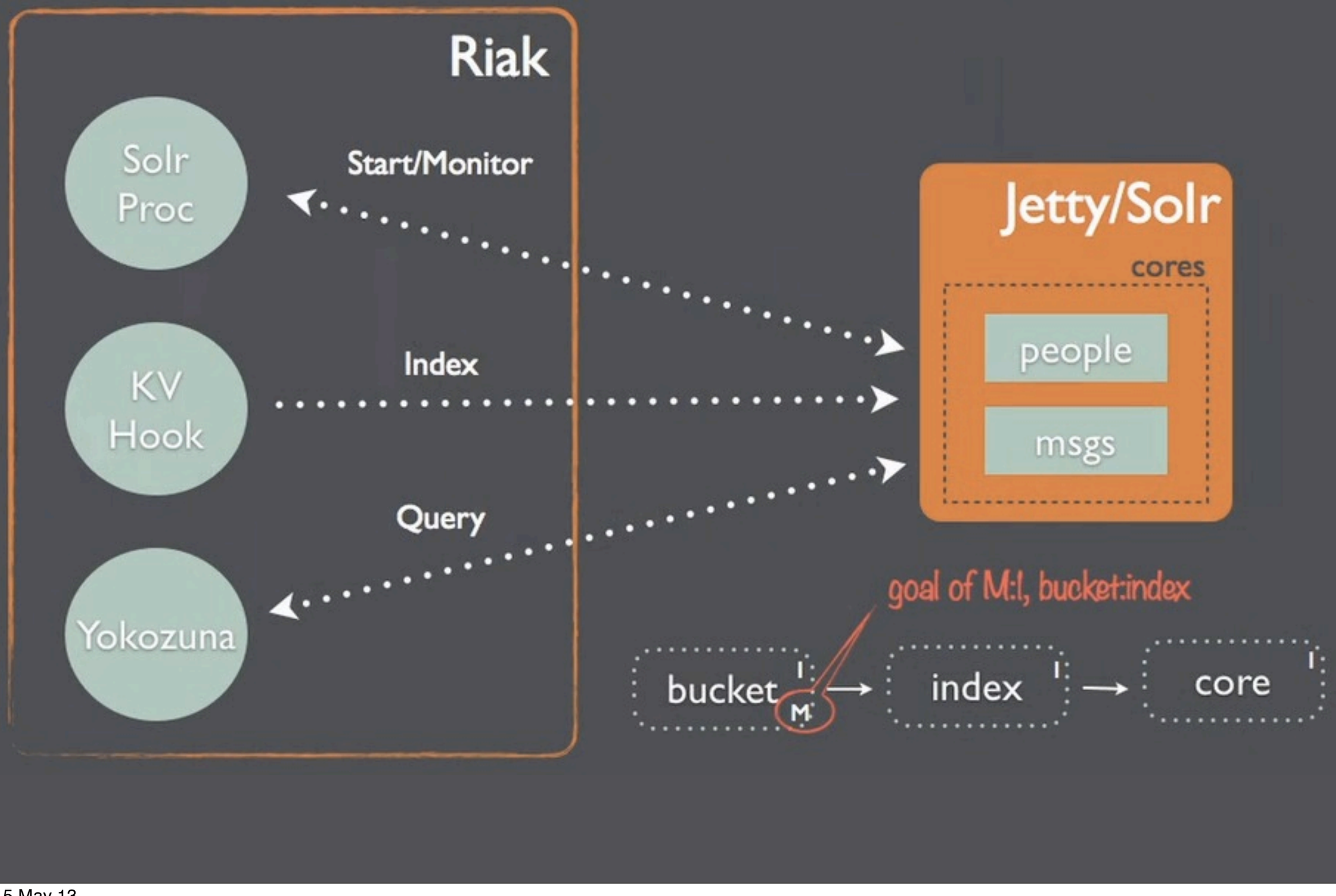
# Improve retrieval in Riak

- Riak excels at storing data at scale
- Solr excels at indexing and querying data
- Riak's existing query technology
  - K/V – GET, PUT, DELETE
  - Secondary Indexes (explicit indexing) EQUALS and RANGE
- Query *very* large data sets most efficiently

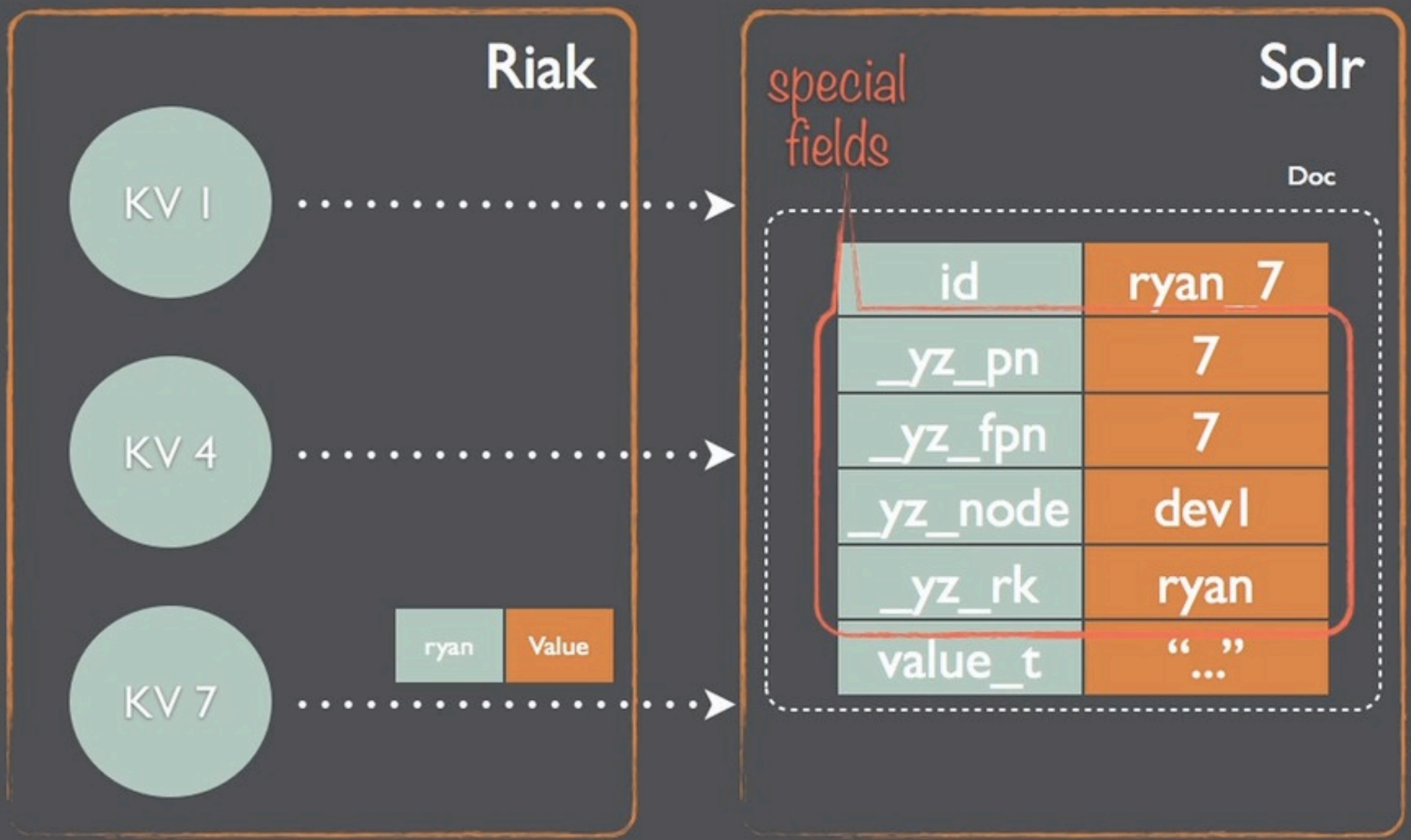
# Combining Riak and Solr

- High availability, fault tolerance, scale out/in
- Efficient indexing, mature feature set, known entity and ecosystem
- Make Solr piggyback off dynamo architecture
- Make Riak searchable at scale in a dependable way

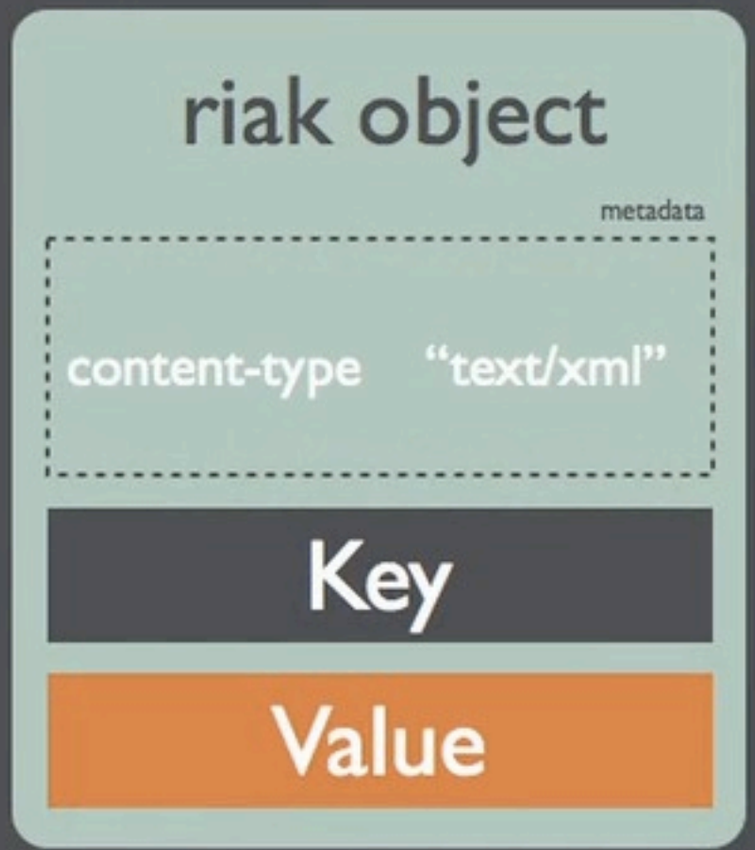
# One Solr Instance per Node



# All Partitions, One Solr



# Extraction on Media Type





```
<doc>
  <person_name_s>Ryan Zezeski</person_name_s>
  <person_bio_t>...</person_bio_t>
  ...
</doc>
```

# Query -> Dist. Query

search/people?q=zezeski

Node A

Riak

yz_cover:plan("people")

solr/people/select?shards=...&fq=...&q=zezeski

Solr

distributed search

NodeB

Solr

NodeB

Solr

NodeC

Solr

# Workflow

1. Query Yokozuna (exposed by Riak)
2. Solr query converted to distributed Solr query (but acts like single instance)
3. Access appropriate shards and filter query
4. Solr, distributes query based on canonical parameters
5. Solr, returns results proxied to caller by Riak

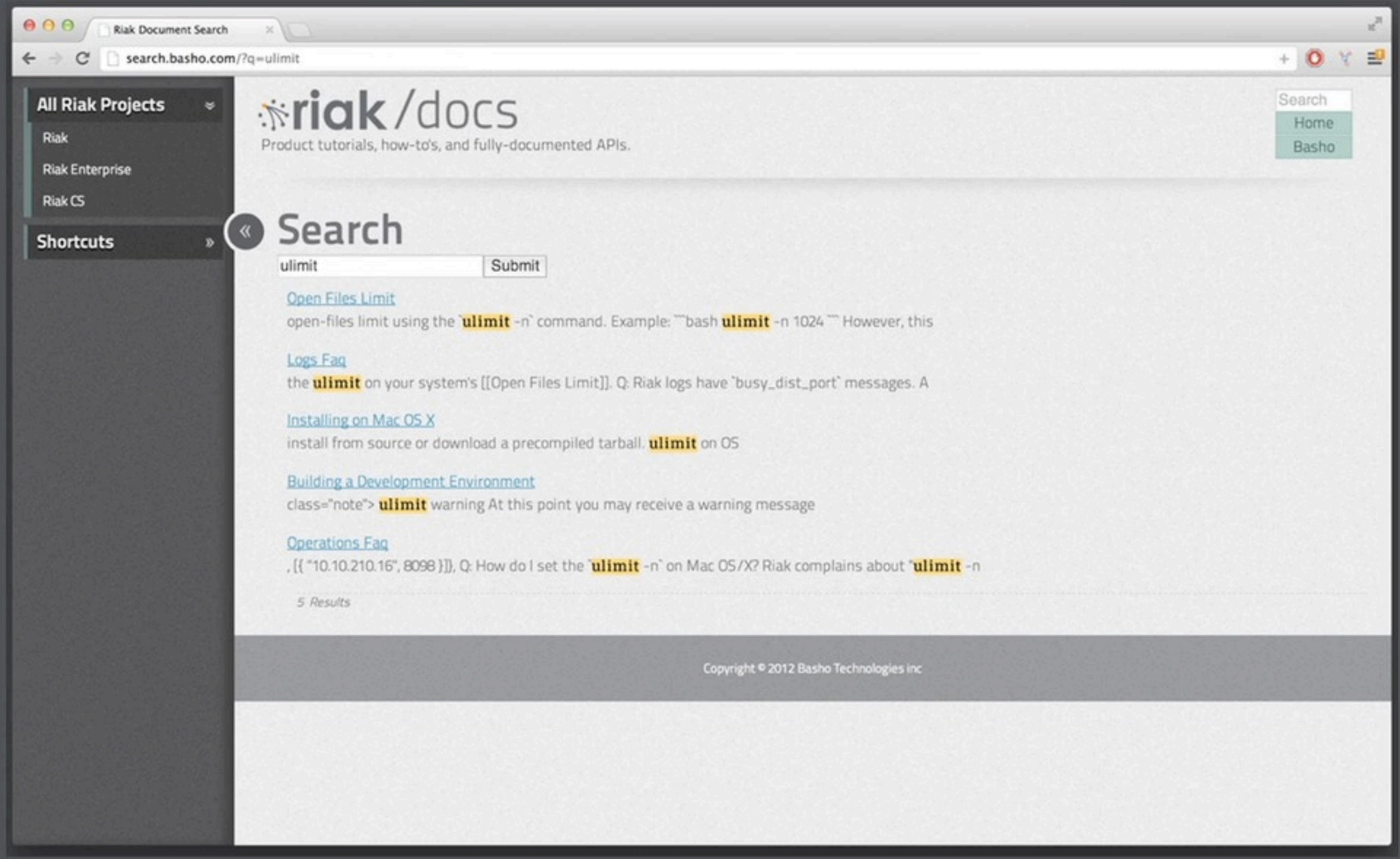
# Using Yokozuna

- Current Options:
  - Pre-Built AMIs for Amazon EC2
  - Build from Source
  - Ask Basho, we'll help you get started
- Query for data as you would with Solr:

```
http://localhost:8098/search/people?
q=natty&wt=json
```

# Current Users

- Some OSS users, particularly those wanting to do geospatial querying
- Many evaluating as alternative to ElasticSearch
- Basho, uses Yokozuna for Basho Docs




# POWERED BY YOKOZUNA/SOLR

# Conclusion

- Any (distributed) search query from Solr
- Introspect and index K/V data
- Synchronize indexes and “master” data
- Yokozuna is likely to be available as a technical preview following release of Riak 1.4 (approx. June 2013)

  Available today:
  https://github.com/basho/yokozuna

# Basho Technologies

- Founded in 2008 by a group of engineers and executives from Akamai Technologies, Inc.
- Design large scale distributed systems
- Develop Riak, open-source distributed database
- Specialize in storing critical information, with data integrity

# RICON 2013

A Distributed Systems Conference for Developers

## RICON | WEST

*San Francisco*

October ...

## RICON | EAST

*New York City*

May 13th-14th at New World Stages in world-famous Midtown Manhattan.

Tickets | More Info

## RICON | EUROPE

*London*

Provisionally scheduled for November 2013

# Questions?

Christian Dahlqvist, christian@basho.com

# Want to know more?

We will come and give a Riak tech talk at your organisation or group:

bit.ly/RiakTechTalk