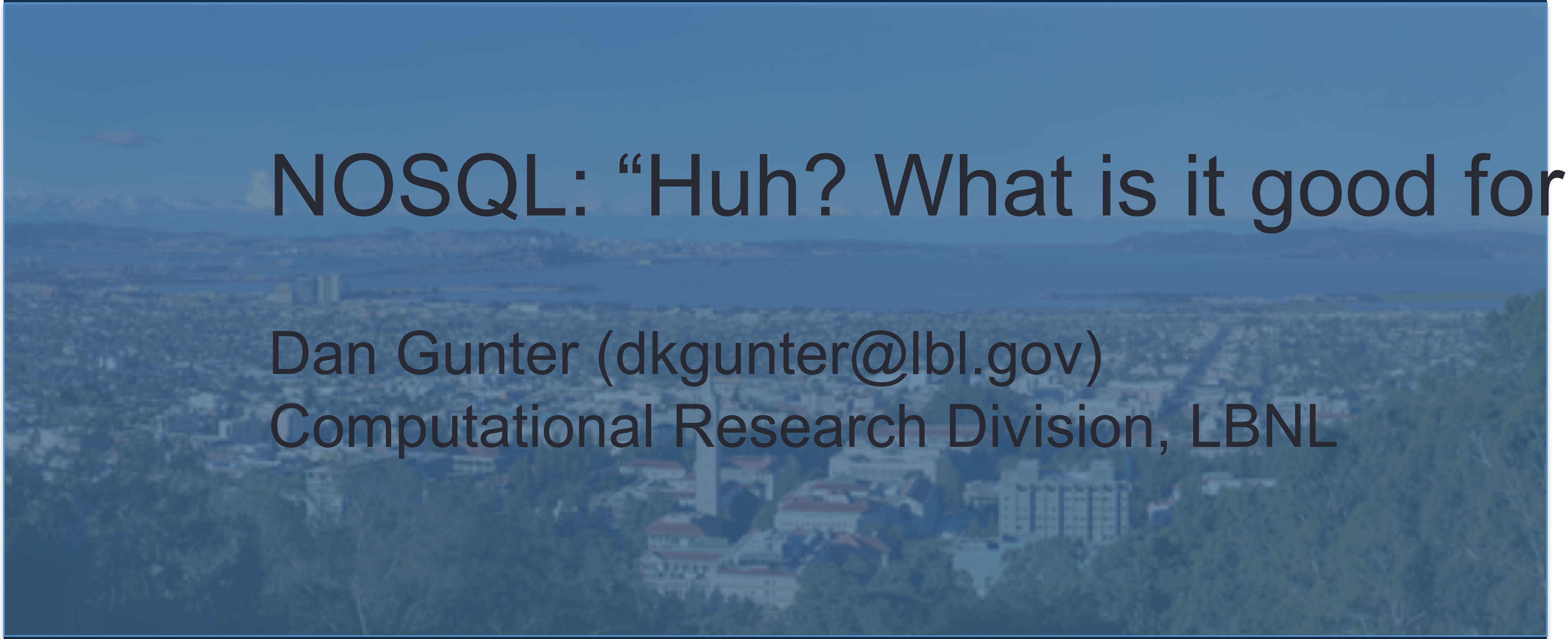
Cloud Computing Workshop
June , 2011
Berkeley, CA



NOSQL: “Huh? What is it good for?”

Dan Gunter (dkgunter@lbl.gov)
Computational Research Division, LBNL

Introduction

- About this talk
  - It is not “hands-on” (sorry)
  - Most of it is history and overview
  - It’s about databases, not explicitly “clouds”
- Why it belongs here
  - Cloud computing and scalable databases go hand-in-hand
  - There are a **lot** of open-source NOSQL projects right now
  - Understanding what they do, and what features of the commercial implementations they’ re imitating, gives insight into scalability issues for distributed computing in general

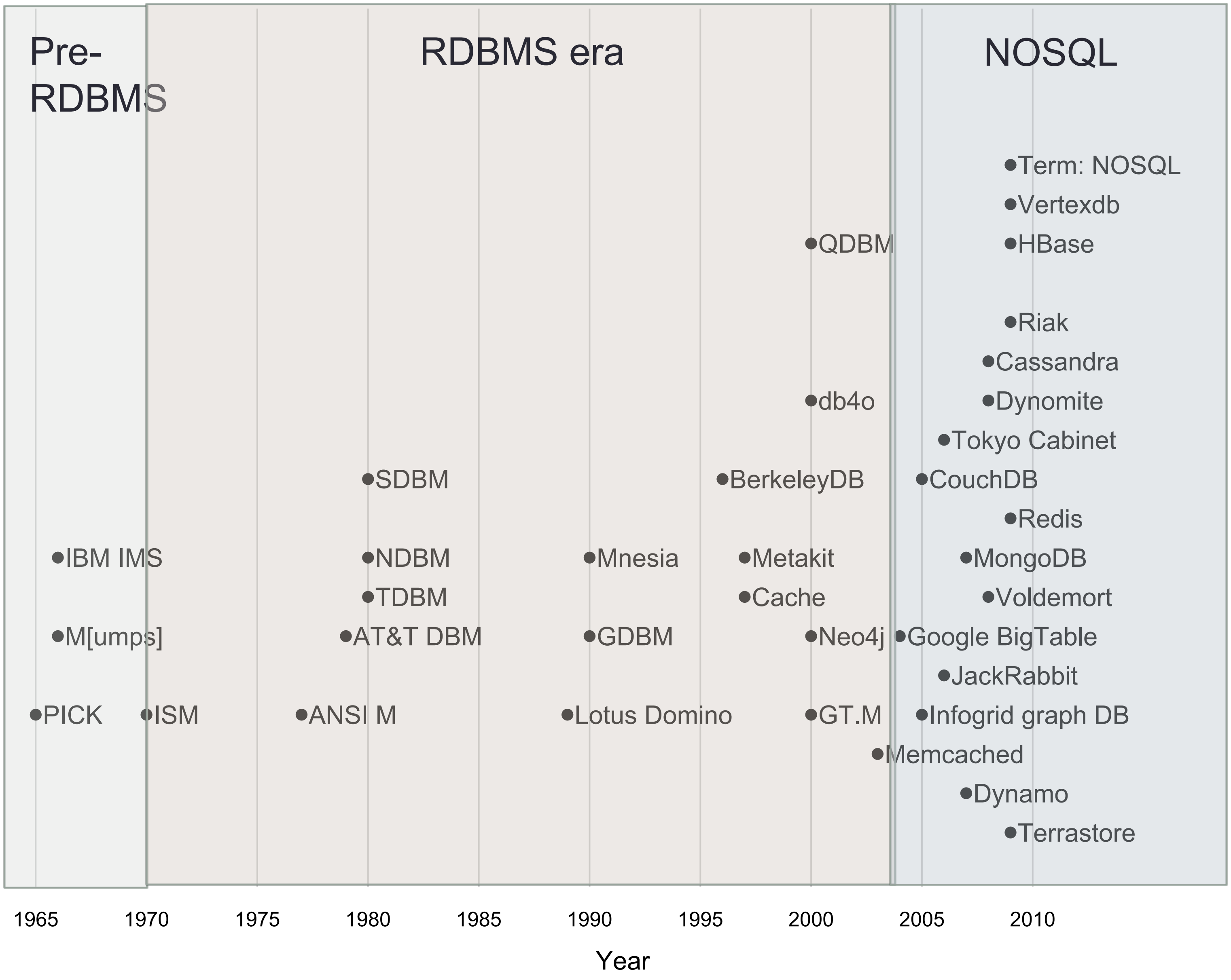
Outline

- Introduction
- History
- Theory
- Technologies
- Data modeling
- Conclusions

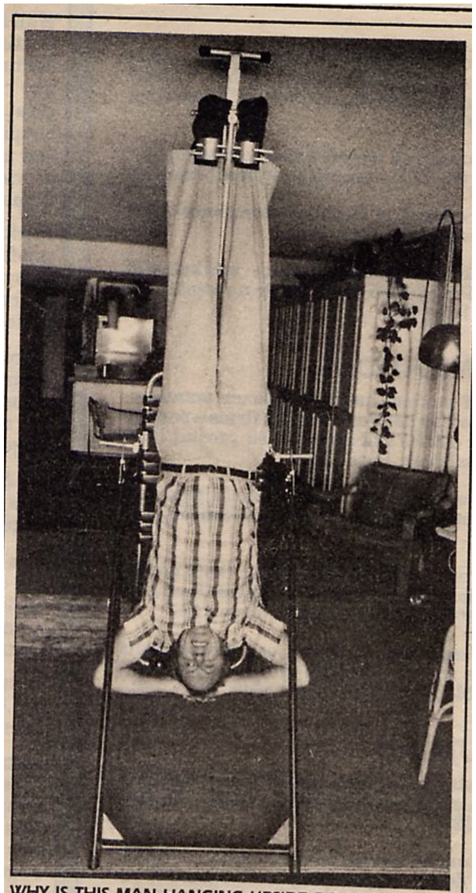
Terminology: NOSQL / Schemaless / …

- First: not terribly important or deep in meaning
- But "NOSQL" has gained currency
  - Original, and best, meaning: **Not Only SQL**
    - Wikipedia credits it to Carlo Strozzi in 1998, re-introduced in 2009 by Eric Evans of Rackspace
    - May use non-SQL, typically simpler, access methods
    - Don't need to follow all the rules for RDBMS'es
  - Lends itself to "No (use of) SQL", but this is misleading
- Also referred to as "schemaless" databases
  - Implies dynamic schema evolution

NOSQL History



Pre-relational data stores



WHY IS THIS MAN HANGING UPSIDE DOWN?: Because he's Dick Pick, and virtually nothing he does is orthodox. Pick, of Pick operating system fame, claims he thinks better this way. With his track record, who's to argue? For more on Pick's style and substance, see page 146.

- Hierarchical storage and sparse multi-dimensional arrays
- MUMPS (Massachusetts General Hospital Utility Multi-Programming System), later ANSI **M**
  - sparse multi-dimensional array
  - global variables, prefixed with “^”, are automatically persisted:

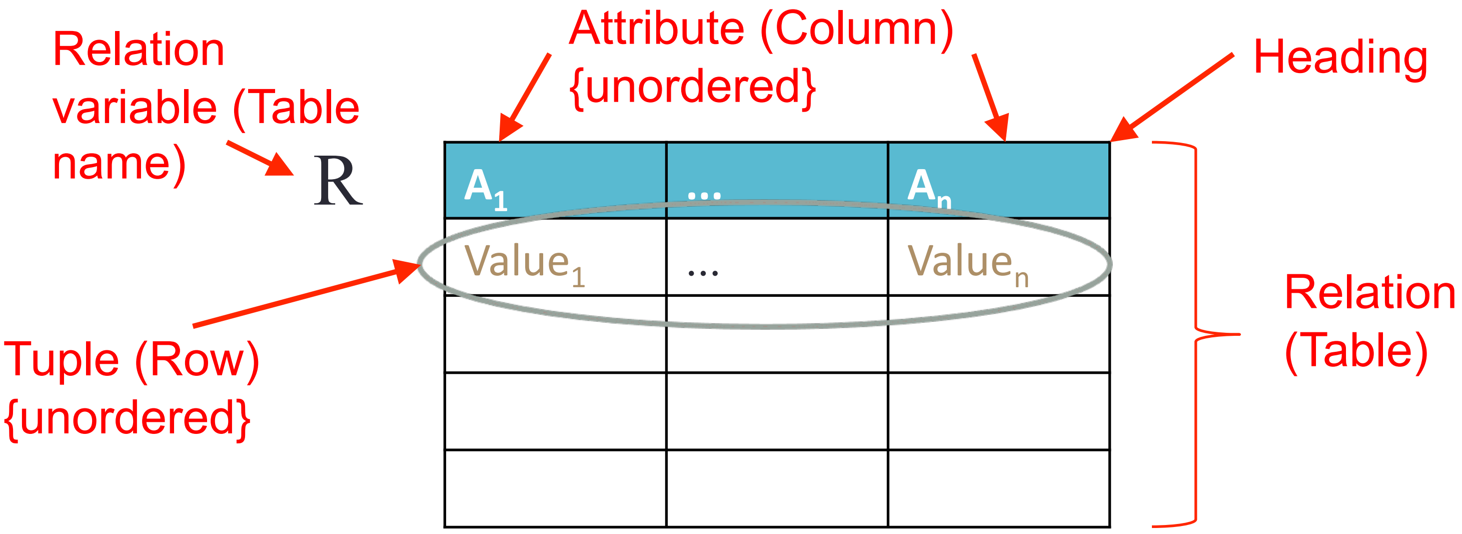
```
^Car(“Door”,”Color”) = “Blue”
```

- “Pick” OS/database
  - everything is hash table
- IBM Information Management System (IMS), [DB1]

Source: Computer Systems News, 11/28/83

The Relational Model

- Introduced with E. F. Codd's 1970 paper "*A Relational Model of Data for Large Shared Data Banks*"
- Relational algebra provided declarative means of reasoning about data sets
- SQL is loosely based on relational algebra



Relation variable (Table name) → R

Attribute (Column) {unordered}

Heading

Tuple (Row) {unordered}

Relation (Table)

A_1	…	A_n
Value_1	…	Value_n

Recent NOSQL database products

Key/Value Store

- Memcached
- Redis
- Tokyo Cabinet
- Dynamo
- Dynomite
- Riak
- Project Voldemort

Columnar *or* Extensible record

- Google BigTable
- HBase
- Cassandra
- HyperTable

Document Store

- CouchDB
- MongoDB
- SimpleDB
- Lotus Domino
- Mnesia

Graph DB

- Neo4j
- FlockDB
- Infinite-Graph

Why now, NOSQL cow?

- G*lobal* internet companies (Google, Amazon, Yahoo!, FaceBook, etc.) hit limitations of standard RDBMS solutions for one or more of:
  - Extremely high transaction rates
  - Dynamic analysis of huge volumes of data
  - Rapidly evolving and/or semi-structured data

- And: these companies – unlike the financial and health services industries using **M** and friends – did not see need for “ACID” guarantees
  - Didn’t want to run z/OS on mainframes (nutters!)
  - “Open world” model, i.e. networks break your $&#!

Theory: The CAP Theorem

- Introduced by Eric Brewer in a PODC keynote on July 2000, thus also known as "Brewer' s Theorem"
- **CAP** = **C**onsistency, **A**vailability, **P**artition-tolerance
  - Theorem states that in any "shared data" system, i.e. any distributed system, you can have *at most* 2 out of 3 of CAP (at the same time)
  - This was later proved formally (w/asynchronous model)

All robust distributed systems live here

Forfeit partition-tolerance	Forfeit availability	Forfeit consistency
Single-site databases, cluster databases, LDAP	Distributed databases w/ pessimistic locking, majority protocols	Coda, web caching, DNS, **Dynamo**

CAP, ACID, and BASE

- RDBMS systems and research focus on ACID: **A**tomicity, **C**onsistency, **I**solation, and **D**urability
  - concurrent operations act as if they are serialized
- Brewer: this is one end of a *spectrum*, one that sacrifices Partition-tolerance and Availability for Consistency
- So, at the other end of the spectrum we have BASE: **B**asically **A**vailable **S**oft-state with **E**ventual consistency
  - Stale data may be returned
  - Optimistic locking (e.g., versioned writes)
  - Simpler, faster, easier evolution

ACID

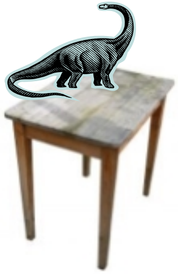
BASE

Pioneers

- Google BigTable
- Amazon Dynamo

These implementations are **not** publicly available, but the distributed-system techniques that they integrated to build huge databases have been imitated, to a greater or lesser extent, by every implementation that followed.

Google BigTable



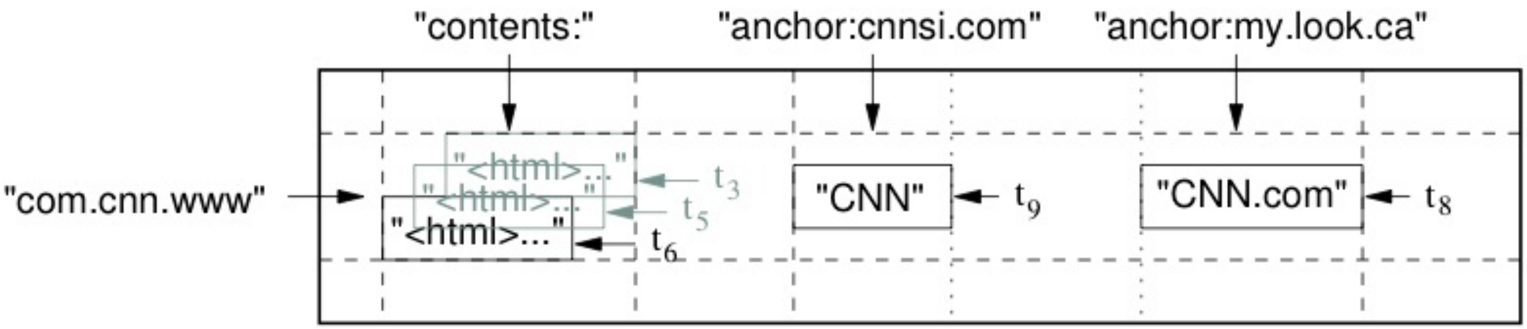
- Internal Google back-end, scaling to thousands of nodes, for
  - **web indexing,** Google Earth, Google Finance
- Scales to petabytes of data, with highly varied data size & latency requirements
- Data model is (3D) sparse, multi-dimensional, sorted map

```
(row_key, column_key, timestamp) ->
  string
```

- Technologies:
  - Google File System, to store data across 1000's of nodes
    - 3-level indexing with **Tablets**
  - **SSTable** for efficient lookup and high throughput
  - Distributed locking with **Chubby**

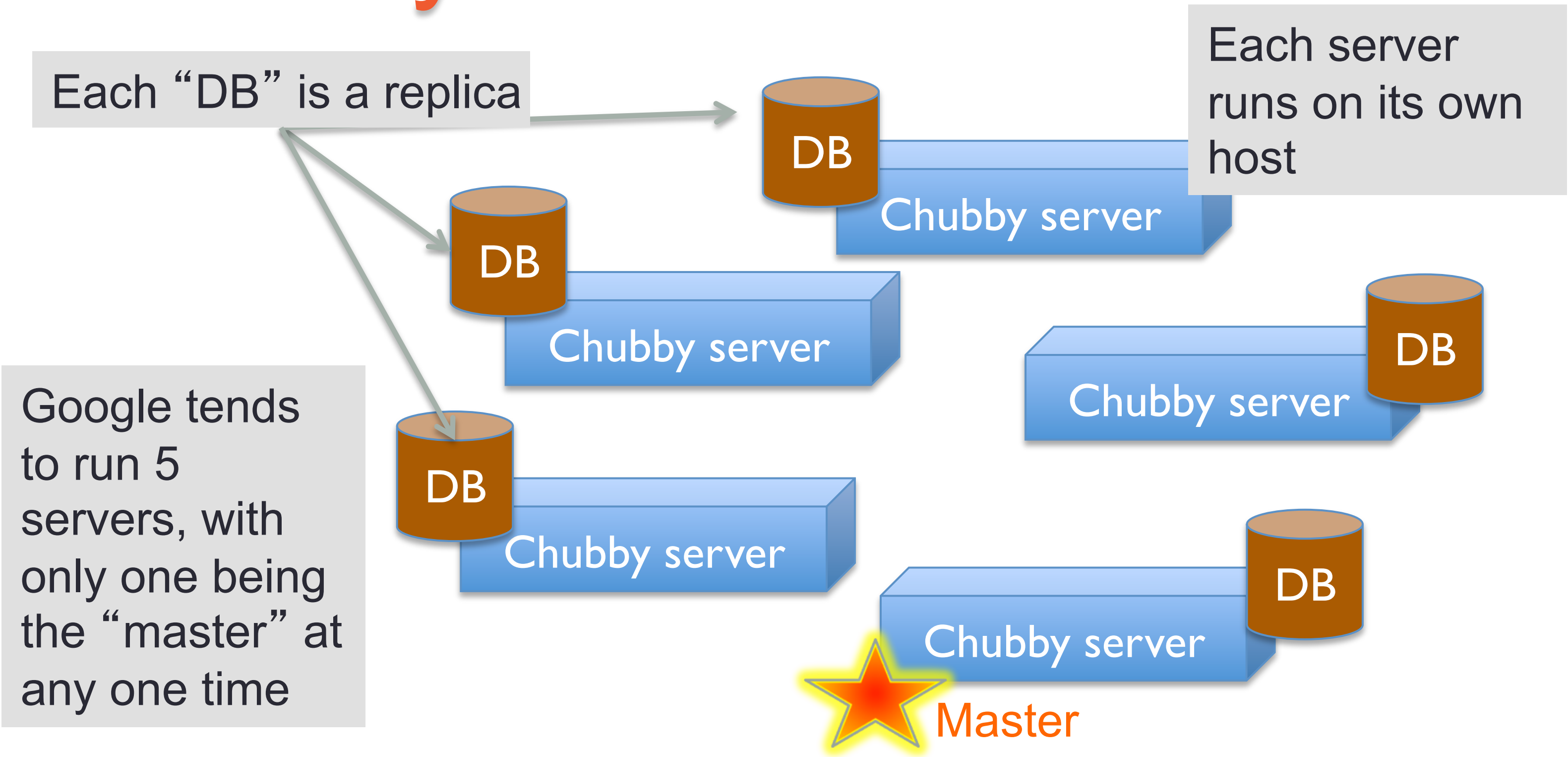
BigTable Data Model

> *Google's Bigtable is essentially a massive, distributed 3-D spreadsheet. It doesn't do SQL, there is limited support for atomic transactions, nor does it support the full relational database model. In short, in these and other areas, the Google team made design trade-offs to enable the scalability and fault-tolerance Google apps require.*
>
> - Robin Harris, StorageMojo (blog), 2006-09-08



Source: http://labs.google.com/papers/bigtable-osdi06.pdf

Chubby and Paxos



Each "DB" is a replica

Each server runs on its own host

Google tends to run 5 servers, with only one being the "master" at any one time

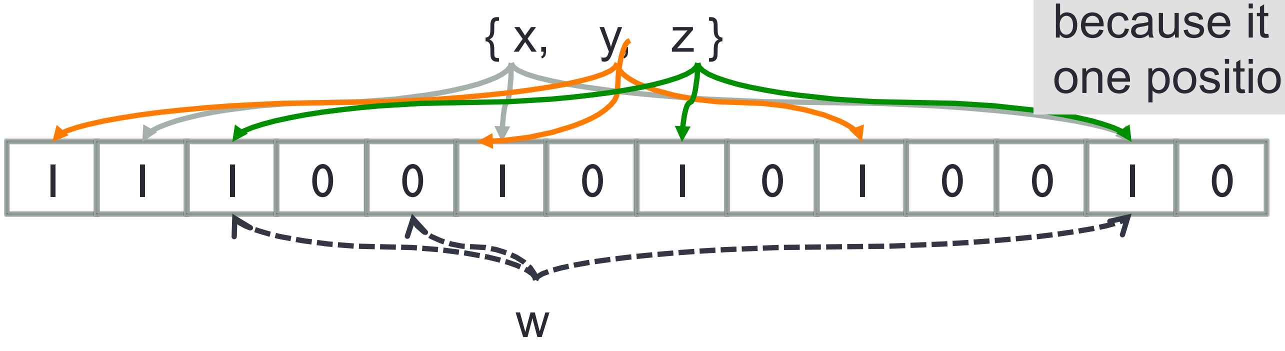
- Chubby is a distributed locking service. Requests go the current Master. If the Master fails, Paxos is used to elect a new one

Tablets and SSTables

- Tablets represent contiguous groups of rows
  - Automatically split when grow too big
  - One “tablet server” holds many tablets
- 3-level indexing scheme similar to B+-tree
  - Root tablet -> Metadata tablets -> Data (leaf) tablets
  - With 128MB metadata tablets, can addr. 2^{34} leaves
- Client communicates directly with tablet server, so data does not go through root (i.e. locate, then transfer)
  - Client also caches information
- Values written to memory, to disk in a commit log; periodically dumped into read-only **SSTables**. Better throughput at the expense of some latency

Bloom Filters for fast lookups

- What is a Bloom filter?
  - Each element is hashed multiple times
  - Constant time: if an element is *not* in a set
  - Can only say “no” with certainty
- Here, tests if an **SSTable** has a row/column pair
  - NO: Stop
  - YES: Need to load & retrieve data anyways
- Useful optimization in this space..



1	1	1	0	0	1	0	1	0	1	0	0	1	0

w is not in { *x*, *y*, *z* } because it hashes to one position with a 0

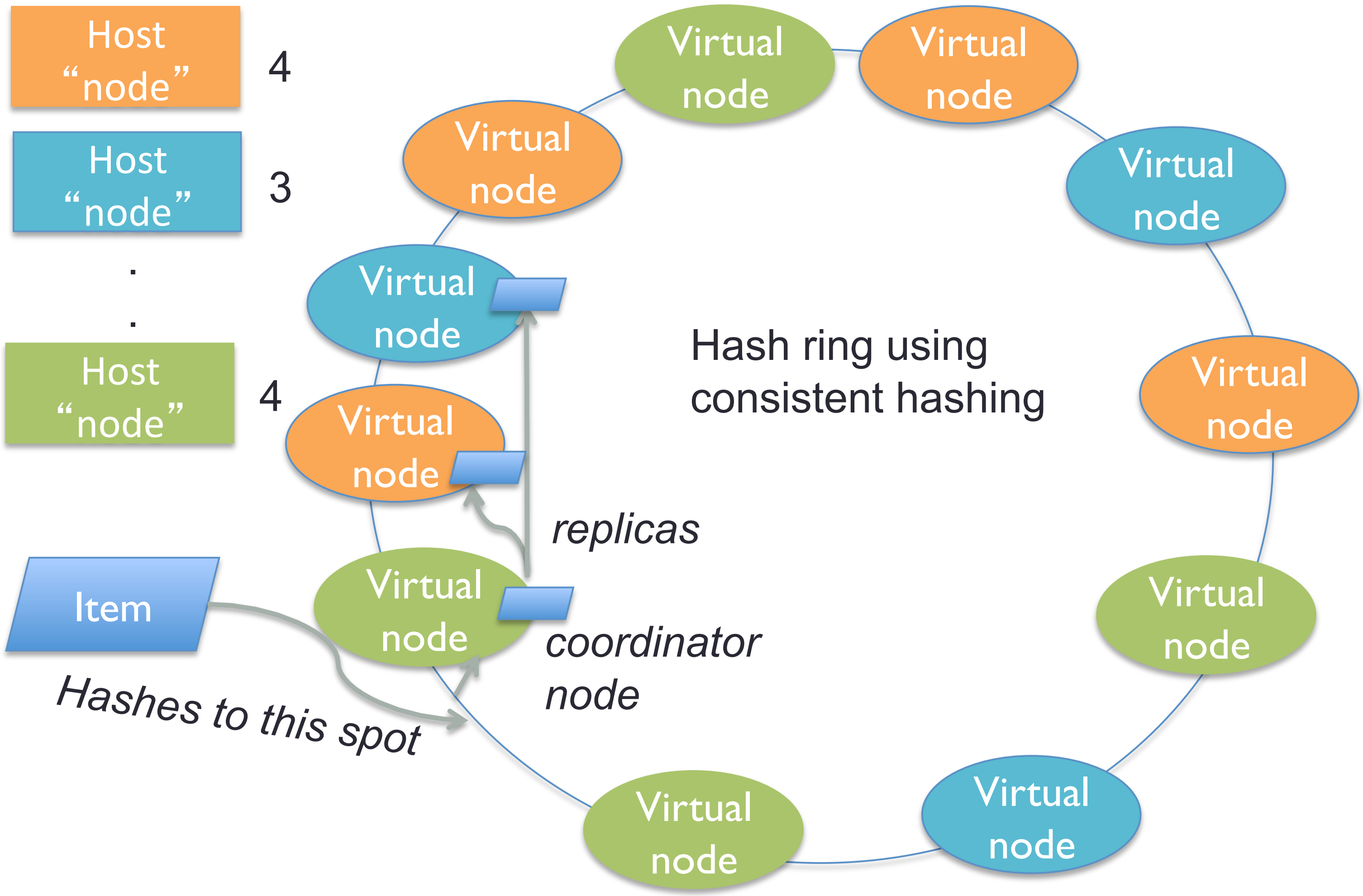
What about CAP?

- For bookkeeping tasks, Chubby's replication allows tolerance of node failures (**P**) and consistency (**C**) at the price of availability (**A**), during time to elect a new master and synchronize the replicas.
- Tablets have "relaxed consistency" of storage, GFS:
  - A single master that maps files to servers
  - Multiple replicas of the data
  - Versioned writes
  - Checksums to detect corruption (with periodic handshakes)

Amazon: Dynamo

- Used by Amazon's "core services", for very high **A** and **P** at the price of **C** ("eventual consistency")
- Data is stored and retrieved solely by key (key/value store)
- Techniques used:
    - *Consistent hashing* – for partitioning
    - *Vector clocks* – to allow MVCC and read repairs rather than write contention
    - *Merkle trees*—a data structure that can diff large amounts of data quickly using a tree of hash values
    - *Gossip* – A decentralized information sharing approach that allows clusters to be self-maintaining
- Old techniques; but not used before at this scale with very high reliability constraints

Dynamo data partitioning and replication



Eventual consistency and sloppy quorum

- **R** = Number of healthy nodes from the *preference list* (roughly, list of "next" nodes on hash ring) needed for a read
- **W** = Number of healthy nodes from preference list needed for a write
- **N** = number of replicas of each data item
- You can tune your performance
  - R << N, high *read availability*
  - W << N, high *write availability*
  - R + W > N, consistent, but *sloppy quorum*
  - R + W < N, at best, *eventual consistency*
- *Hinted handoff* keeps track of the data "missed" by nodes that go down, and updates them when they come back online

2011 HPC and Cloud Computing Workshop

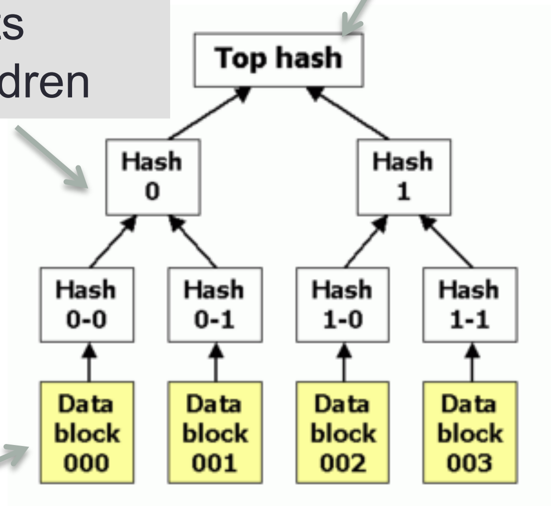
BERKELEY LAB
LAWRENCE BERKELEY NATIONAL LABORATORY

Replica synchronization with Merkle trees

- When things go really bad, the “hinted” replicas may be lost and nodes may need to synchronize their replicas
- To make synchronization efficient, all the keys for a given virtual node are stored in a *hash tree* or *Merkle tree* which stores data at the leaves and recursive hashes in the nodes
- Same hash => Same data at leaves

If two top hashes match, then the trees are the same

Each node is a hash of its children



For Dynamo, the “data” are the keys stored in a given virtual node

Infrastructure (at scale) is fractal

• Why didn't Amazon or Google just run a big machine with something like GT.M, Vertica, or KDB (etc.)?

• The answer must be partially to do something new, but partially that it wasn't *just* shopping carts or search

Their whole infrastructure is dynamic, and pieces of it are splitting off and growing, and sub-pieces of those pieces are later breaking off and also growing larger, etc. etc.

- This ability to **be effective at multiple scales** is crucial to the rise in NOSQL (schemaless) database popularity

The Gold Rush

Columnar *or* Extensible record

- Google BigTable
- HBase
- Cassandra
- HyperTable

Key/Value Store

- Memcached
- Redis
- Tokyo Cabinet
- Dynamo
- Dynomite
- Riak
- Project Voldemort

Document Store

- CouchDB
- Mnesia
- Lotus Domino
- MongoDB

Graph

- Neo4j
- FlockDB
- InfiniteGraph

Key/Value Store

Memcached

Redis

Tokyo Cabinet

Dynamo

Dynomite

Riak

Project Voldemort

- Basic operations are simply get, put, and delete
- All systems can distribute keys over nodes
- Vector clocks are used as in Dynamo (or just locks)
- Replication: common
- Transactions: not common
- Multiple storage engines: common

Project Voldemort

Type	Key/Value Store
License	Apache 2.0
Language	Java
Company	Linked-In
Web	project-voldemort.com

- Dynamo-like features:
  - Automatic partitioning with consistent hashing
  - MVCC with vector clocks
  - Eventual consistency (N, R, and W)
- Also:
  - combines cache with storage to avoid sep. cache layer
  - **pluggable storage layer**
    - RAM, disk, other..

Riak

Type	Key/Value Store
License	Open-Source
Language	Erlang
Company	Basho
Web	wiki.basho.com/display/RIAK/Riak/

Map/reduce with the Python API

```
result = self.client
    .add(bucket.get_name())
    .map(“Riak.mapValuesJson”)
    .reduce(“Riak.reduceSum”)
    .run()
```

- Dynamo-like features:
  - Consistent hashing
  - MVCC with vector clocks
  - Eventual consistency (N, R, and W)
- Also:
  - Hadoop-like M/R queries in either JS or Erlang
  - REST access API

Columnar *or* Extensible record

Google BigTable

HBase

Cassandra

HyperTable

- All share BigTable data model
  - rows and columns
  - "column families" that can have new columns added
- Consistency models vary:
  - MVCC
  - distributed locking
- Need to run on a different back-end than BigTable (GFS ain't for sale)

Cassandra

Type	Extensible column store
License	Apache 2.0
Language	Java
Company	Apache Software Foundation
Web	cassandra.apache.org

Widely used: Facebook, Twitter, Digg, Reddit, Rackspace

- Marriage of BigTable and Dynamo
  - Consistent hashing
  - Structured values
  - Columns / column families
  - Slicing with predicates
  - Tunable consistency:
    - W = 0, Any, 1, Quorum, All
    - R = 1, Quorum, All
  - Write commit log, memtable, and uses SSTables

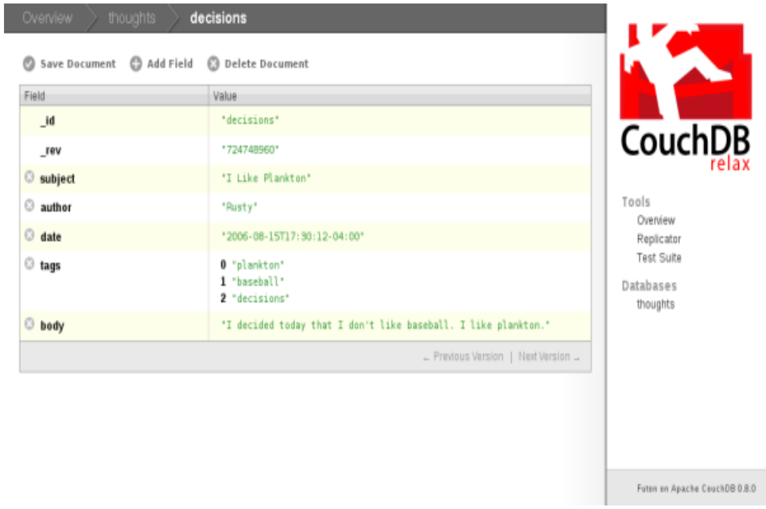
Document Store

- CouchDB
- MongoDB
- SimpleDB
- Lotus Domino

- Store objects
  - not really documents; think: nested maps
- Varying degrees of consistency, but not ACID
- Allow queries on data contents (M/R or other)
- May provide atomic read-and-set operations

CouchDB

Type	Document store
License	Apache 2.0
Language	Erlang
Company	Apache Software Foundation
Web	couchdb.org



Field	Value
_id	"decisions"
_rev	"724748960"
subject	"I Like Plankton"
author	"Rusty"
date	"2006-08-15T17:30:12-04:00"
tags	0 "plankton" 1 "baseball" 2 "decisions"
body	"I decided today that I don't like baseball. I like plankton."

- Objects are grouped in "collections"
- REST API
  - *Elegant*, but slow
- Read scalability through async. replication with eventual consistency
- No sharding
- Incrementally updated MR "views"
- ACID? Uses MVCC and flush on commit. So, kinda..

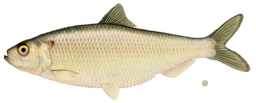
MongoDB

Type	Document store
License	GPL
Language	C++
Company	10gen
Web	mongodb.org

- Groups objects in collections within a database
- Data stored in Binary JSON format called BSON
- *Replication* for HA
- *Sharding* to scale reads and writes
- M/R queries, **rich JSON query language**
- User-defined indexes on fields of the objects
- Atomic update modifiers can
  - increment value
  - modify-if-current
- Journaled writes for performance

Mnesia

Type	Document store
License	EPL*
Language	Erlang
Company	Ericsson
Web	www.erlang.org
Papers	http://www.erlang.se/publications/mnesia_overview.pdf

* Mozilla Public License modified to conform with laws of Sweden (more )

- Stores data in "tables"
  - Data stored in memory
  - Logged to selected disks
- Replication and sharding
- Queries are performed using Erlang list comprehensions (?!)
- User-defined indexes on fields of the objects
- Transactions are supported (but optional)
- Optimizing query compiler and dynamic "rule" tables
- Embedded in Erlang OTP platform (similar to *Pick*)

Benchmarks?

- Tough, since the whole point of NOSQL databases is to not be a one-size-fits-all solution
  - e.g., Hard to quantify different styles of consistency
- Yahoo Cloud-Serving Benchmark
  - http://research.yahoo.com/node/3202
- Do not fool yourself that scalability is going to be magically easy
  - "MongoDB is web scale": http://www.youtube.com/watch?v=b2F-DItXtZs

Data Modeling

I remember having late night meetings about tables, normalization and migration and how best to represent the data we have for each packet capture. For a startup, these kinds of late night meetings are critical in establishing a bond amongst the engineers who are just learning to work with each other. NoSQL destroys this human aspect in a number of ways.

http://labs.mudynamics.com/2010/04/01/why-nosql-is-bad-for-startups/

Example from distributed monitoring

- Consider semi-structured input like:

```
ts      = 2010-02-20T23:14:06Z
event   = job.state
wf_uuid= 8bae72f2-etc
state   = JOB_SUCCESS
name    = create_dir_montage
job_submit_seq=1
```

- If the fields are likely to change*, or new types of data will appear, how to model this kind of data?

1. Blob
2. Placeholders
3. Entity-Attribute-Value

RDBMS
“anti-patterns”

In the year since I first wrote this slide, they changed!

What’ s wrong with EAV?

- It’ s terrible, trust me: I tried it
- You end up with queries that look like this to just extract a bunch of fields that started out in the same log line:

```
select e.time, user.value user,
host.value host, dest.value dest,
      nbytes.value nbytes, dur.value
dur, type.value type
from event e
join attr user on e.id = user.e_id
join attr host on e.id = host.e_id
join attr dest on e.id = dest.e_id
join attr nbytes on e.id =
nbytes.e_id
join attr dur on e.id = dur.e_id
join attr type on e.id = type.e_id
join attr code on e.id = code.e_id
where
e.name = 'FTP_INFO'
and host.name = 'host'
and dest.name = 'dest'
and nbytes.name = 'nbytes'
and dur.name = 'dur'
and type.name = 'type'
and user.name = 'user'
and (code.name = 'code' and
code.value = '226')
```

What about queries?

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.

Query languages

- Yes, Erlang

```
query [E.name || E <- table(employee),
                 E.sex = female]
end
```

- And Map/Reduce

```
function(doc) {
    if (doc.last_name) {
        emit(doc.last_name, doc);
    }
}
function find_users_whose_last_names_start_with(db, query) {
    var matches;
    matches = db.view('users/last_names',
                      { startkey: query,
                        endkey:   query + "\u9999" });
    return matches.rows.map(dot('value'));
}
```

More Query Languages

- ..and JSON (MongoDB)

  {last_name: 'Smith'}, {'ssn': 1}

- all the key/value stores
- PIG (see tutorial!)
- etc.

- Is there any way this can be unified?
  - Why would one want to do that?

NOSQL == CoSQL?

SQL

- Children point to parents
- Closed world
- Entities have identity (extensional)
- Necessarily strongly typed
- Synchronous (ACID) updates across multiple rows
- Environment coordinates changes (transactions)
- Value-based, strong reference (referentially consistent)
- Not compositional
- Query optimizer

co-SQL

- Parents point to children
- Open world
- Environment determines identity (intensional)
- Potentially dynamically typed
- Asynchronous (BASE) updates within single values
- Entities responsible to react to changes (eventually consistent)
- Computation-based, weak reference (expect 404)
- Compositional
- developer/pattern

Source: A Co-Relational Model for Large Shared Data Banks
DOI:10.1145/1924421.1924436

Your data on NOSQL

- You need to think about this going in; you are throwing away much of the elegance of relational query optimization
  - need to weigh against costs of static schemata
- Holistic approach:
  - Spend lots of time on **logical** model, understand problem!
  - What degree of normalization makes sense?
  - Is your data well-represented as a hash table? Is it hierarchical? Graph-like?
  - What degree of consistency do you really need? Or maybe multiple ones?

Example from Mat. Science

- Use MongoDB for:
  - Input data (materials)
  - Queue manager
  - Provenance
  - Output data (calculated energies)
- Moved from MySQL database with tons of tables requiring tons of joins
- Use de-normalized Mongo schema
  - queries got considerably simpler

Sample query for Mat. Sci.

- Every crystal that has (Li or Na or K), (Mn), (O or S or F or Si), plus one other element except (Zn or Ni or Fe or Cu or Co or Br or Re or Y) (nelements = 4) with volume less than X, Mn has oxidation state 3+, Mn has 6-fold coordination, and crystal system is monoclinic. X = 500 'wyckoff' is a proxy for 6-fold coordination, oxidation state

```
{ "space_group.crystal_system" : "monoclinic",
"lattice.volume" : { "$lt" : 500 },
"element_names" : {"$all" : ['Mn'],"$size" : 4},
"atoms" : { "$elemMatch" : { "oxidation" : 3 } },
"$where" : "match_all(
this.element_names, ['Li', 'Na', 'K'], ['Mn'], ['O',
'S', 'F', 'Si'])”,
element.names : { “$nin” ['Zn', 'Ni', 'Fe', 'Cu',
'Co', 'Br', 'Re', 'Y’]} }
```

2011 HPC and Cloud Computing Workshop

BERKELEY LAB
LAWRENCE BERKELEY NATIONAL LABORATORY

Conclusions

Not
R.I.P.
RDBMS

- Anyone who says RDBMS is dead (and means it) is an idiot
- SQL is mostly a red herring
  - Can be layered on top of NOSQL, e.g. BigQuery and Hive
  - Ad-hoc joins? Really?
- What's really interesting about NOSQL is scalability (given relaxed consistency) and flexibility
  - incremental scalability from local disk to large degrees of parallelism in the face of distributed failure
  - easier schema evolution, esp. important at the "development" phase, which is often longer than anyone wants to admit
- Whether we should move towards the One True Database or a Unix-like ecosystem of tools is mostly a matter of philosophical bent; certainly both directions hold promise

Selected references

- Cattell's overview of "scalable datastores"
  - ◦ http://cattell.net/datastores/
- BigTable
  - ◦ http://labs.google.com/papers/bigtable.html
- Stonebraker et al. on columnar vs. map/reduce
  - ◦ http://database.cs.brown.edu/sigmod09/benchmarks-sigmod09.pdf
- NOSQL "summer reading": http://nosqlsummer.org/
  - ◦ "path throgh them": http://doubleclix.wordpress.com/2010/06/12/a-path-throug-nosql-summer-reading/
- Varley's Master's Thesis on non-relational db's (modeling)
  - ◦ http://ianvarley.com/UT/MR/Varley_MastersReport_Full_2009-08-07.pdf

Why do we care about Mnesia / OTP?

- Database for RabbitMQ (distributed messaging behind S3)
- Erlang see   g a popularity i   l-computing space

- Mmm.. tas