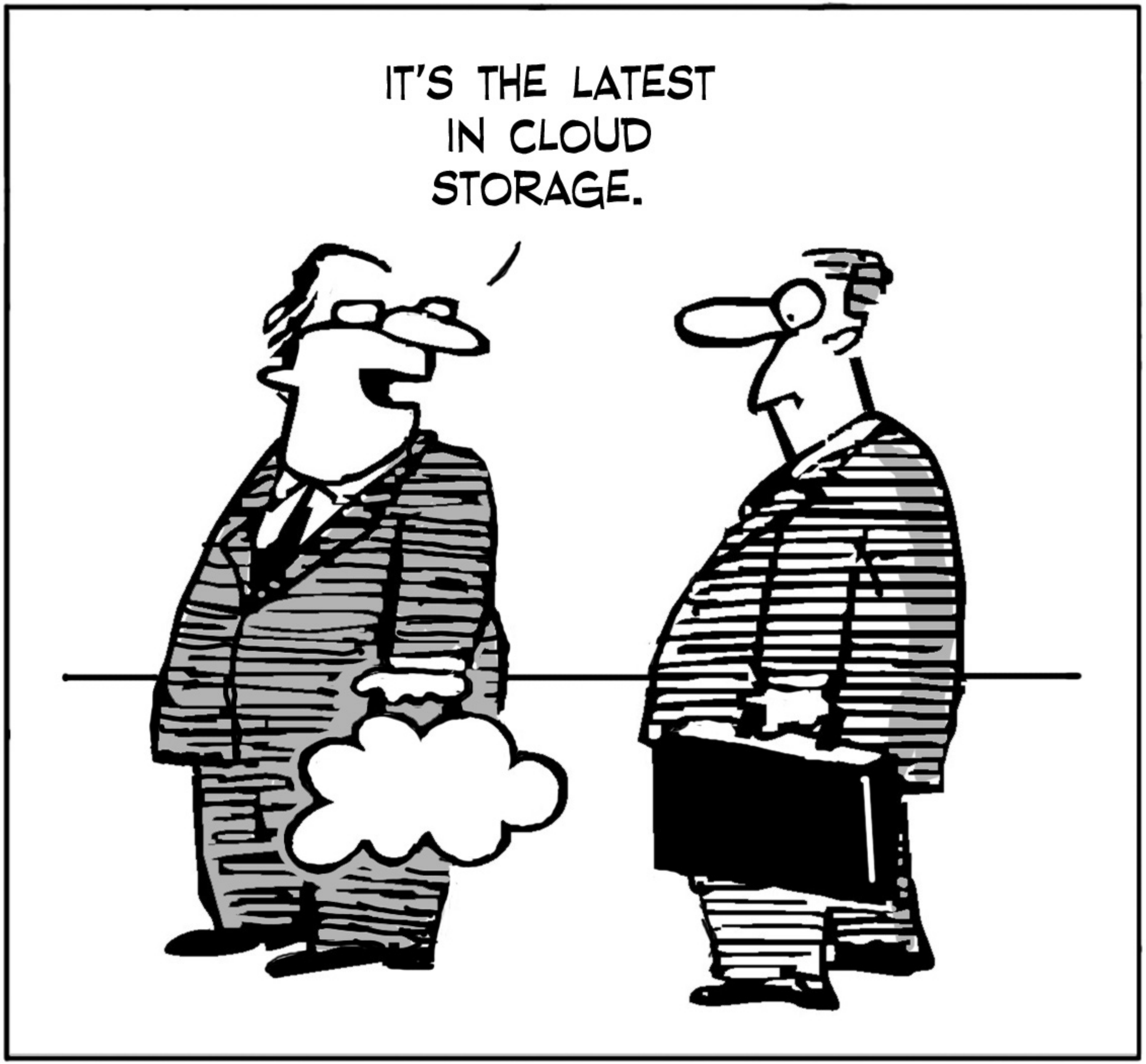
Yes or NoSQL

Andy Street, Casey Link,
David Mazary, Jonathan Berkhahn,
Val Komarov

In partnership with

Booz | Allen | Hamilton

strategy and technology consultants


IT'S THE LATEST IN CLOUD STORAGE.

Problem

- Modern applications have
  - lots of data
  - lots of users
- Need for highly performant, *scalable* database
- Traditional RDMs cannot scale (easily)
  - Why?

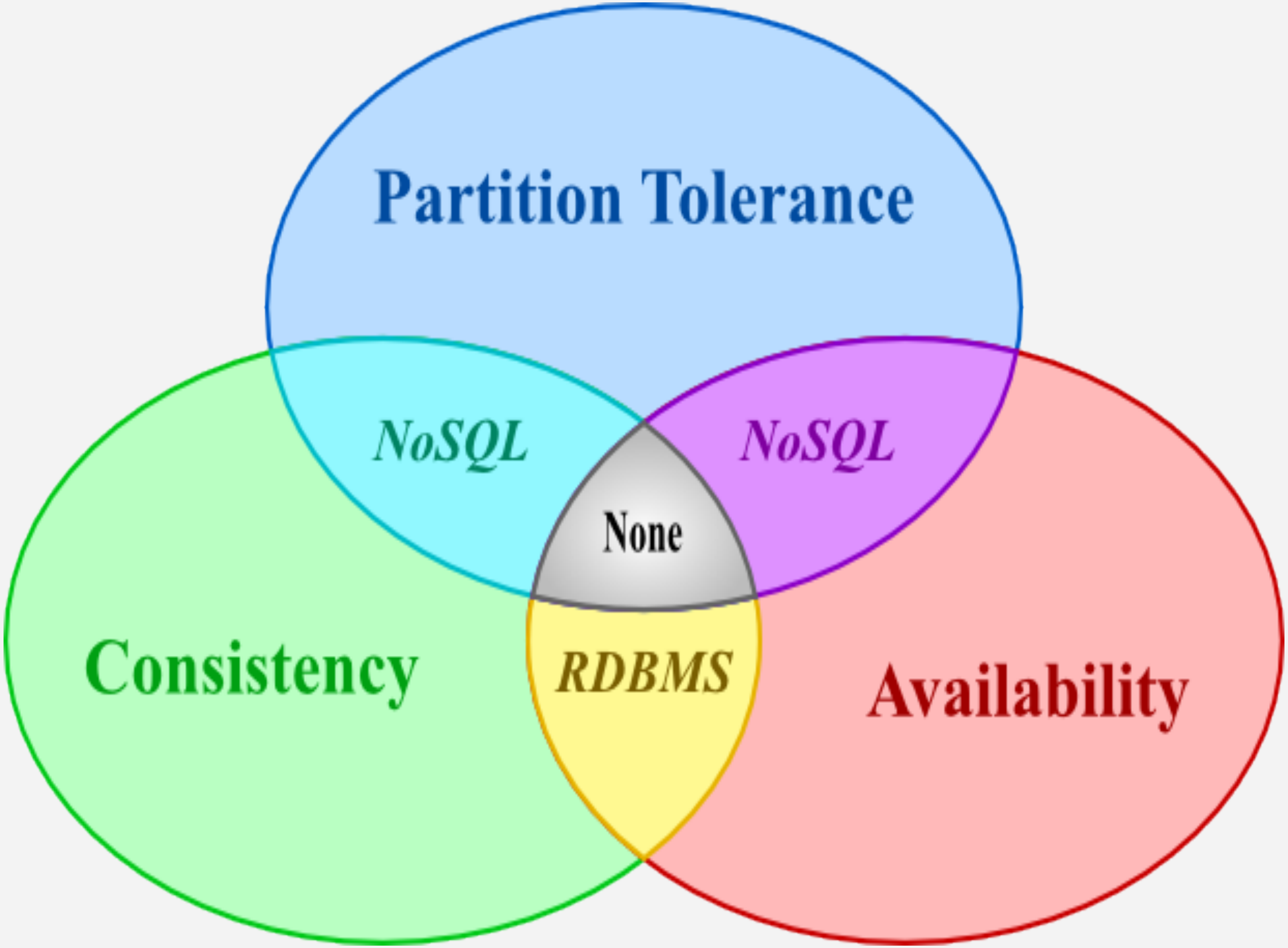
Background

CAP Theorem - *Pick Two*

- Consistency - *do you get the same results?*
- Availability - *can you talk to it?*
- Partition Tolerance - *does it scale?*

- Main issue: Partitioning (i.e., scalability)

- Tradeoff - which to sacrifice?
  - *Consistency* or *Availability*



Venn diagram of the CAP Theorem

NoSQL

Not Only SQL

- Blanket term for any type of structured data storage
- **Goal:** Create DBMS that are scalable
- *Many* different implementations exist, each with strengths and weaknesses
  - Key-value (Dynamo, Cassandra)
  - Tabular (BigTable, HBase)
  - Document (MongoDB)
  - Graph, Object, Multivalue ... and more

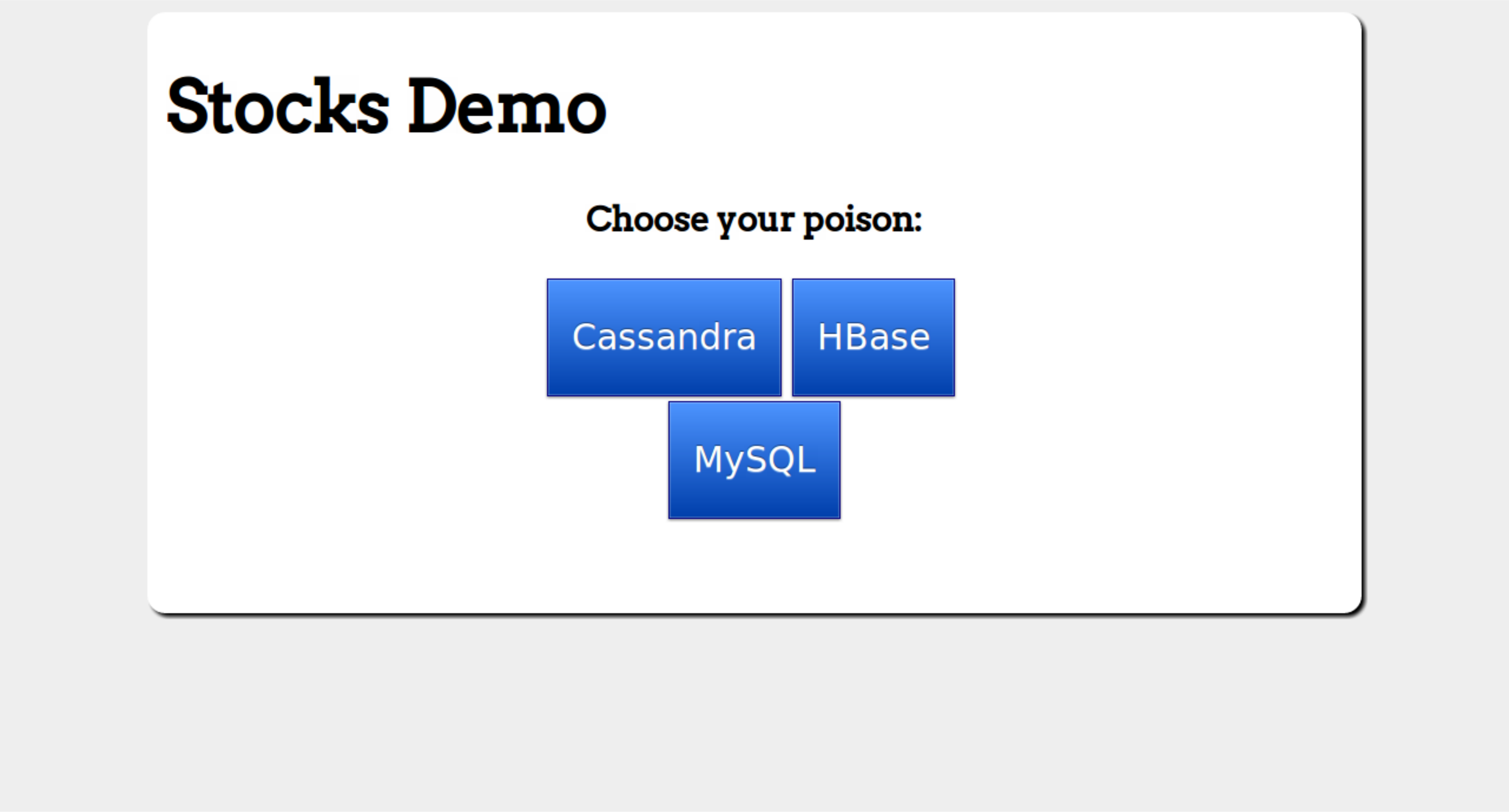
Motivation

- Of our sponsor (BAH)
  - Worthwhile investment?
  - Implementations
  - Developer experience
- Academic motivation
  - Growing technology
  - Benchmarking

NoSqlDemo.com

- NASDAQ Stock Data
  - NASDAQ Exchange Daily 1970-2010 Open, Close, High, Low and Volume
- Query 3 Different Data Sources
  - Compare implementations
- Try it out

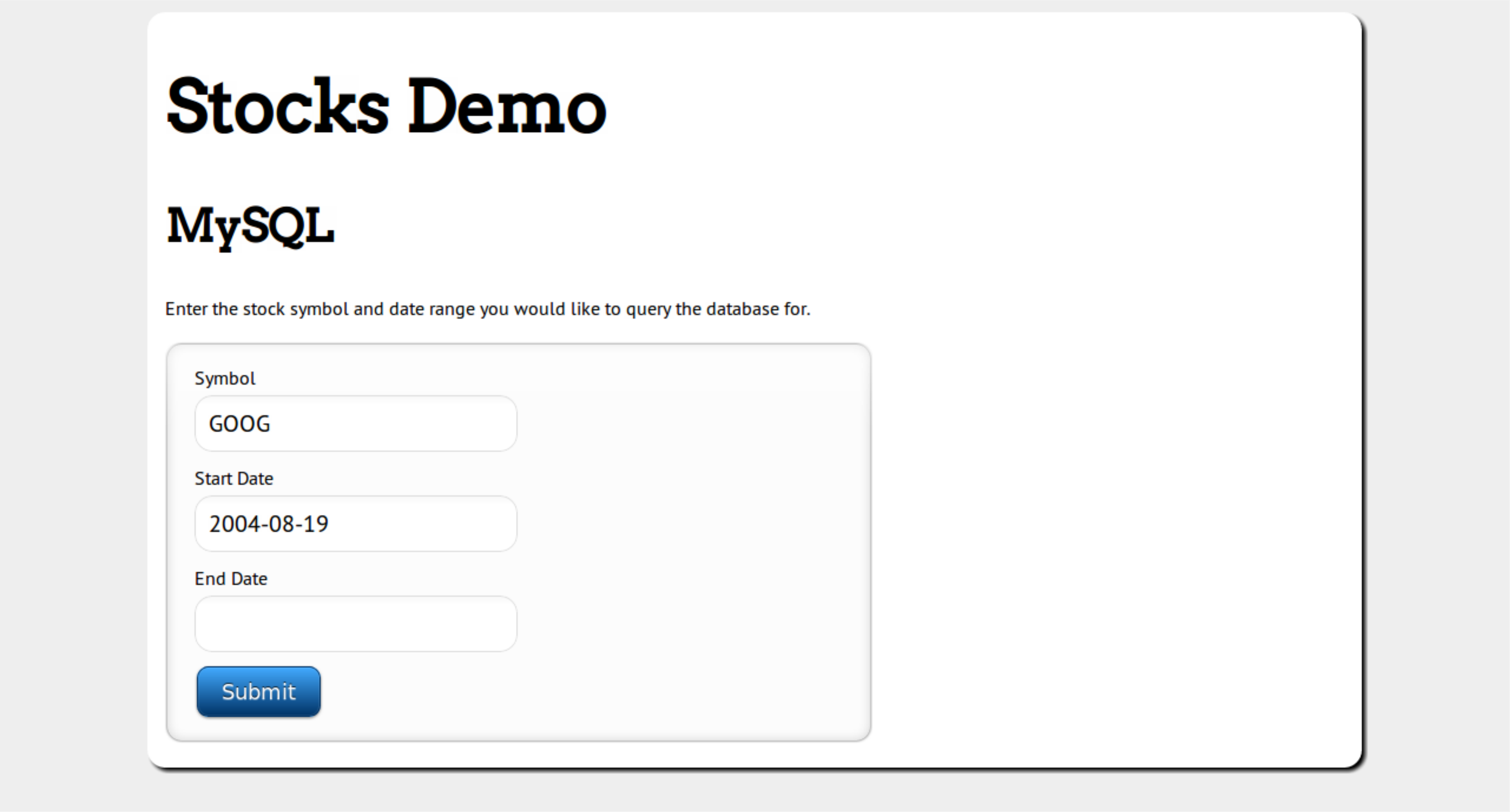
NoSqlDemo.com



Stocks Demo

Choose your poison:

NoSqlDemo.com



Stocks Demo

MySQL

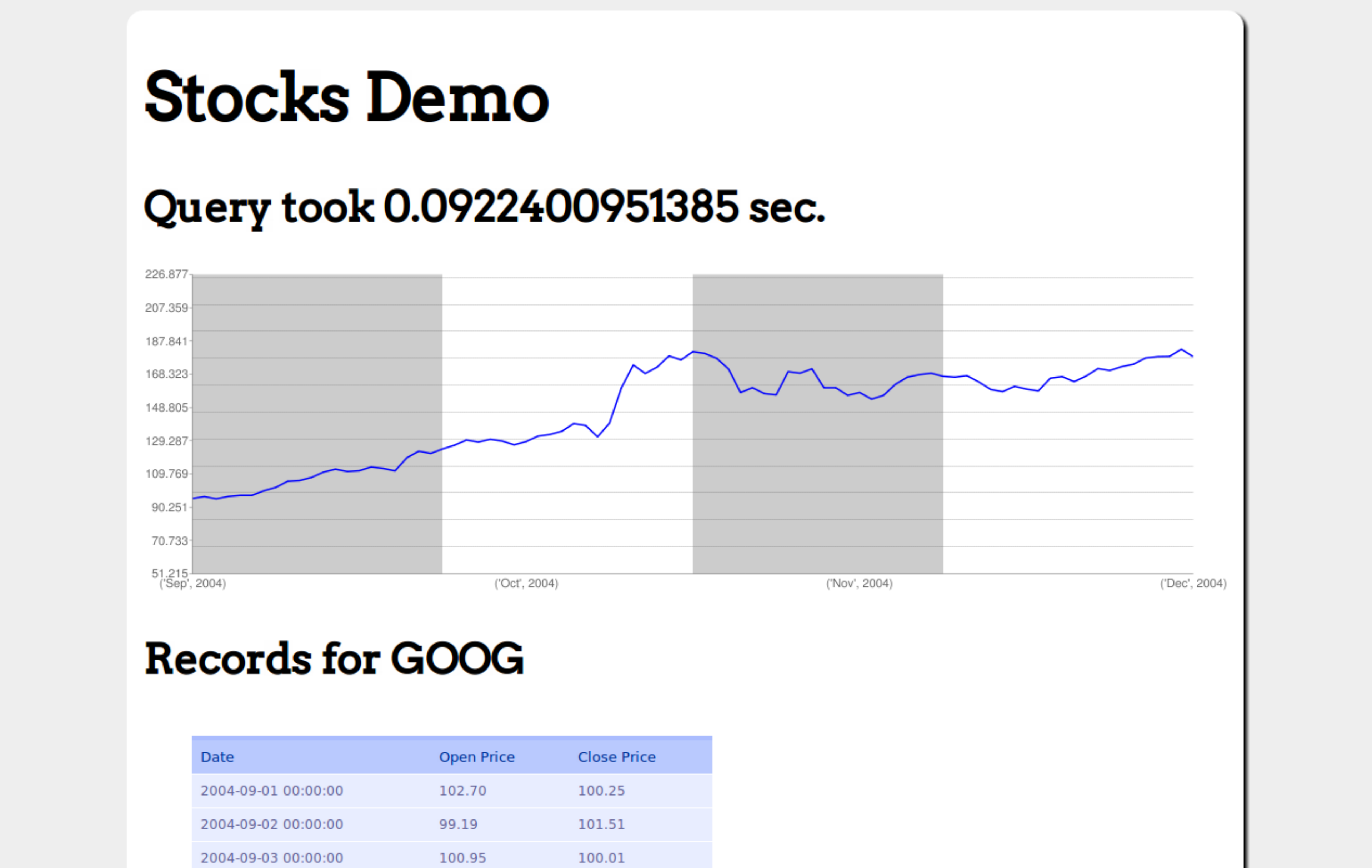
Enter the stock symbol and date range you would like to query the database for.

Symbol: GOOG

Start Date: 2004-08-19

End Date:

NoSqlDemo.com



Stocks Demo

Query took 0.0922400951385 sec.

Records for GOOG

Date	Open Price	Close Price
2004-09-01 00:00:00	102.70	100.25
2004-09-02 00:00:00	99.19	101.51
2004-09-03 00:00:00	100.95	100.01

Cassandra

- Distributed Key-Value Store
- *Eventually Consistent*
- Easy to setup
  - DHT - Peers find each other
- Many language bindings
  - Thrift - Python, Java, PHP, Ruby, etc
- Tunable Consistency

Cassandra Schema

```
Symbol {
  Date 1 {
    open:
    close:
    ...
  }
  ...
}

Date {
  Symbol 1 {
    open:
    close:
    ...
  }
  ...
}
```

```
GOOG {
 2005/01/01 {
   open: 500,
   close: 501
 }
 2005/01/02 {
   open: 501,
   close: 502
 }
}
2005/01/01 {
  GOOG{
   open: 501,
   close: 502
 }
  AAPL {
   open: 501,
   close: 502
 }
}
```

HBase

- Modeled after Google's BigTable
- Runs on HDFS
- Consistent and Partition-tolerant:
  - Single writer
  - NameNode is single PoF
- Can MapReduce run natively

Development

- PyBase, an API based on Pycassa
- Converted our CassandraModel to HBaseModel
- Adjusted for the differences in HBase's structure
    - Scanner instead of Get
    - Configure the Scanner correctly, parse through the results
- Difficulties with proprietary data types
- Very little documentation of PyBase

HBase Schema

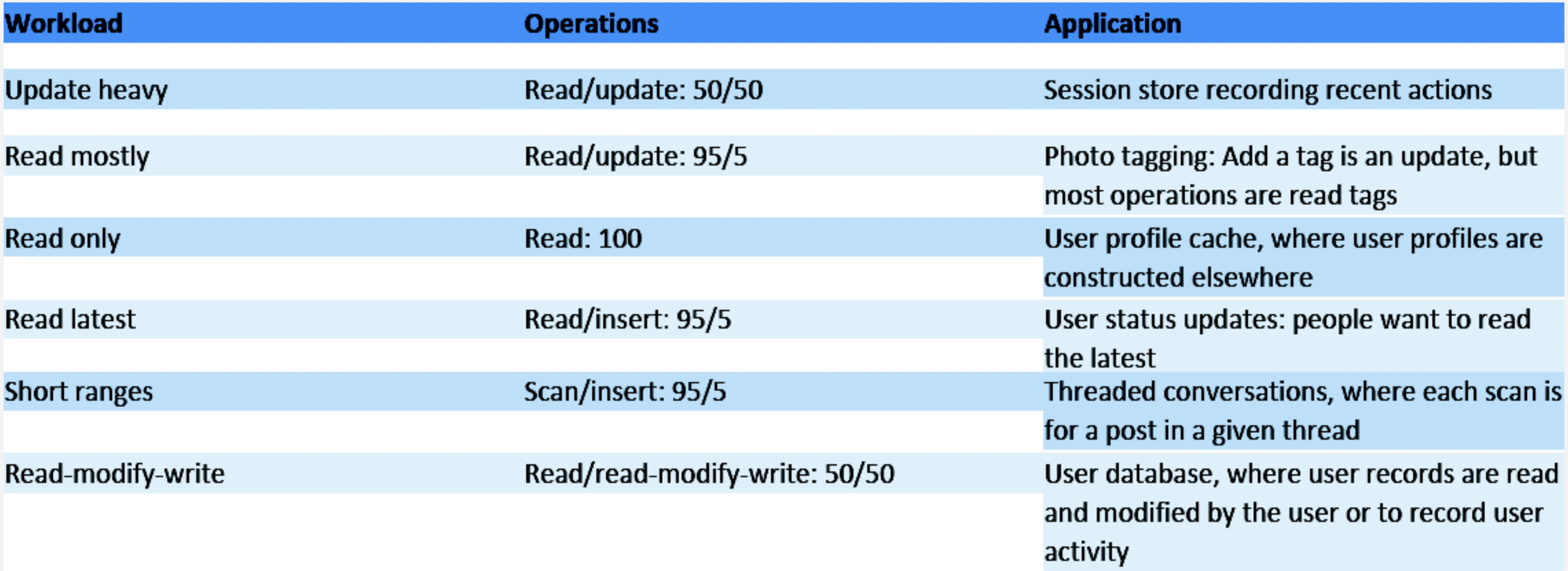
Table	Row	Column Families	
		Price	Volume
		Columns	
Stocks	<symbol><date>	price: open	
		price: close	
		price: high	
		price: low	
Dates	<date><symbol>	price: open	
		price: close	
		price: high	
		price: low	
		Column Family	
		Symbols	
		Column	
Symbols	<first letter<	Symbol: <symbol>	

MySQL

Consistent and Available

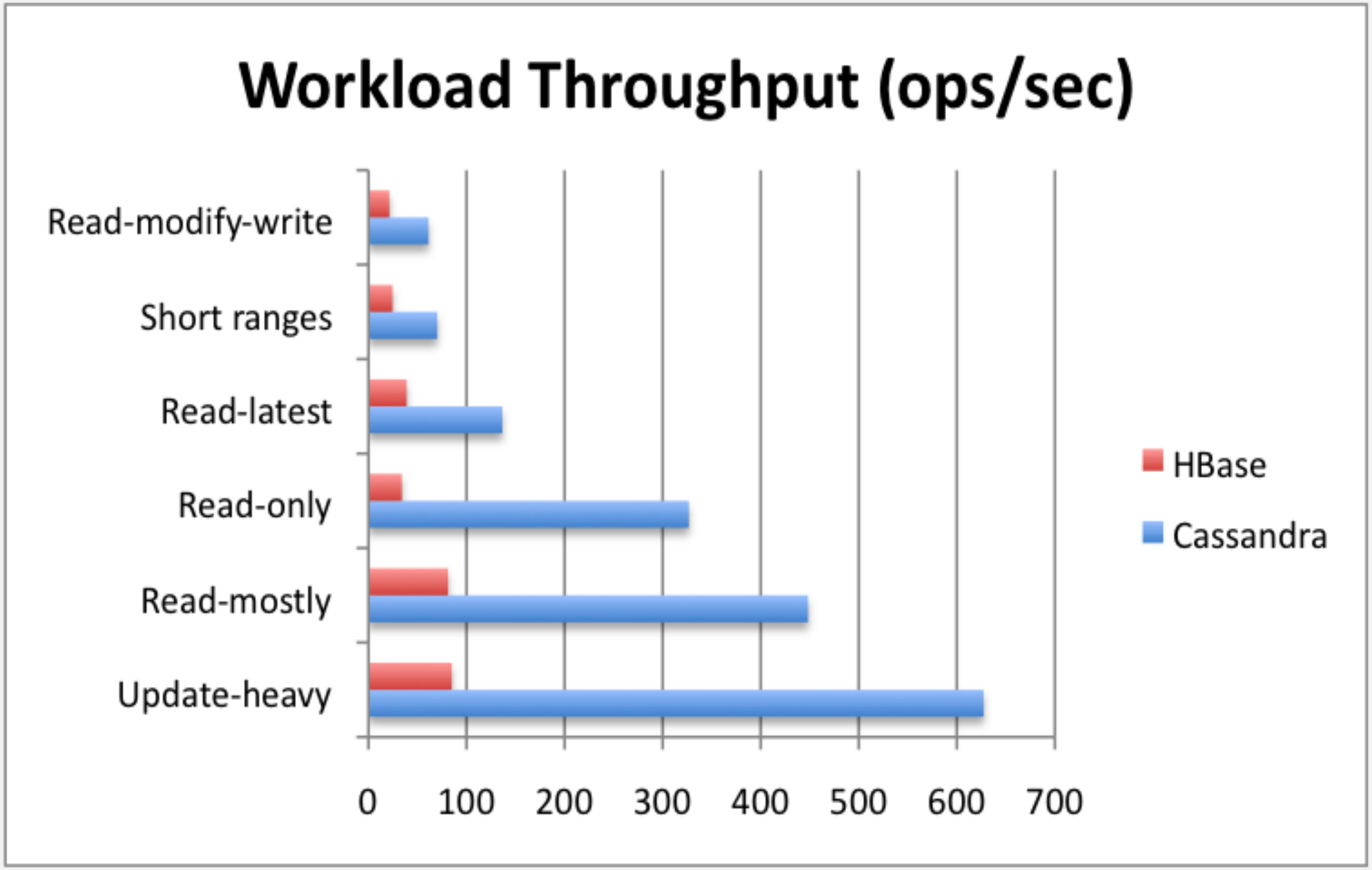
- "Traditional"
- Easy Setup (sudo apt-get install mysql-server)
- Simple Schema
  - Direct import from CSV
  - Flat table
- Unparalleled Support
- High and Low level API support for many languages
- Doesn't scale well
- Further improvements through caching (memcached) and mirroring (Linux-HA project)

YCSB

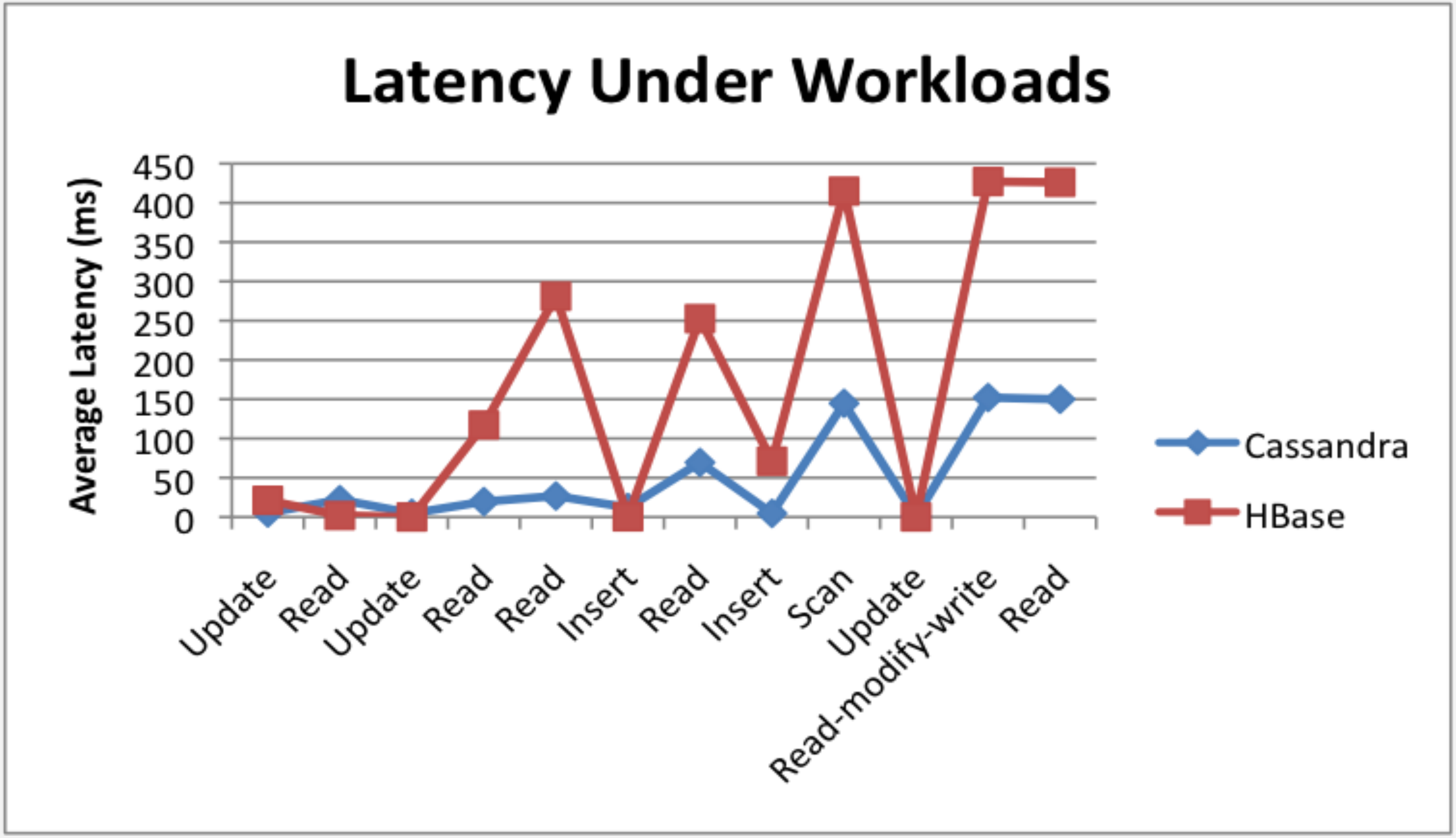


Workload	Operations	Application
Update heavy	Read/update: 50/50	Session store recording recent actions
Read mostly	Read/update: 95/5	Photo tagging: Add a tag is an update, but most operations are read tags
Read only	Read: 100	User profile cache, where user profiles are constructed elsewhere
Read latest	Read/insert: 95/5	User status updates: people want to read the latest
Short ranges	Scan/insert: 95/5	Threaded conversations, where each scan is for a post in a given thread
Read-modify-write	Read/read-modify-write: 50/50	User database, where user records are read and modified by the user or to record user activity

YCSB


Workload Throughput (ops/sec)
Read-modify-write
Short ranges
Read-latest
Read-only
Read-mostly
Update-heavy
0
100
200
300
400
500
600
700
HBase
Cassandra

YCSB


Latency Under Workloads
Average Latency (ms)
450
400
350
300
250
200
150
100
50
0
Update
Read
Update
Read
Read
Insert
Read
Insert
Scan
Update
Read-modify-write
Read
Cassandra
HBase

Conclusions

- Know your data
  - What queries do you want to make?

- Understand your solution's Data Model

- Watch out for EC2

MySQL Setup

1. Install from repo (sudo apt-get install mysql-server)
2. Configure binding addr and port
3. Create Database
4. Import stocks data from CSV files (via..LOAD DATA INFILE)
5. Create internal hash indexes (via..CREATE INDEX)
6. Use MySQLdb and Python to marshall DB data
7. Write webapp

Questions?