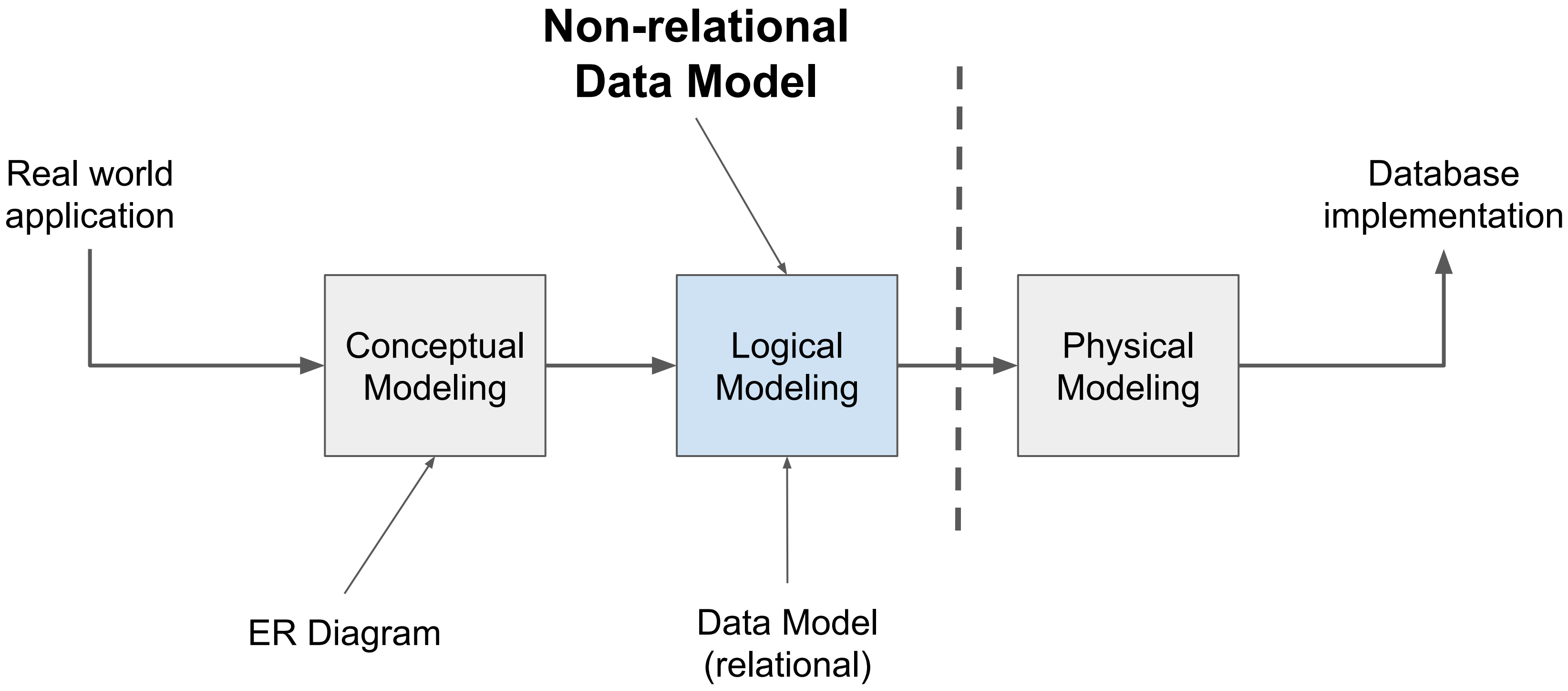
Databases and Big Data

NoSQL

This week



Non-relational Data Model
Real world application
Conceptual Modeling
Logical Modeling
Physical Modeling
Database implementation
ER Diagram
Data Model (relational)

Data Model - Recap

Relational

Key/value

Graph

Document

Array/Matrix

NoSQL

- Started as: no SQL
- Now: Not Only SQL

Event Details

Introduction

This meetup is about "open source, distributed, non relational databases".
Have you run into limitations with traditional relational databases? Don't mind trading a query language for scalability? Or perhaps you just like shiny new things to try out? Either way this meetup is for you.
Join us in figuring out why these newfangled Dynamo clones and BigTables have become so popular lately. We have gathered presenters from the most interesting projects around to give us all an introduction to the field.

Preliminary schedule

09.45: Doors open
10.00: **Intro session** (Todd Lipcon, Cloudera)
10.40: Voldemort (Jay Kreps, Linkedin)
11.20: Short break
11.30: Cassandra (Avinash Lakshman, Facebook)
12.10: Free lunch (sponsored by Last.fm)
13.10: Dynomite (Cliff Moon, Powerset)
13.50: HBase (Ryan Rawson, Stumbleupon)
14.30: Short break
14.40: Hypertable (Doug Judd, Zvents)
15.20: CouchDB (Chris Anderson, couch.io)
16.00: Short break
16.10: Lightning talks
16.40: Panel discussion
17.00: Relocate to Kate O'Brien's, 579 Howard St. @ 2nd. First round sponsored by Digg

Registration

The event is free but space is limited, please register if you wish to attend.

Location

Magma room, CBS interactive
235 Second Street
San Francisco, CA 94105

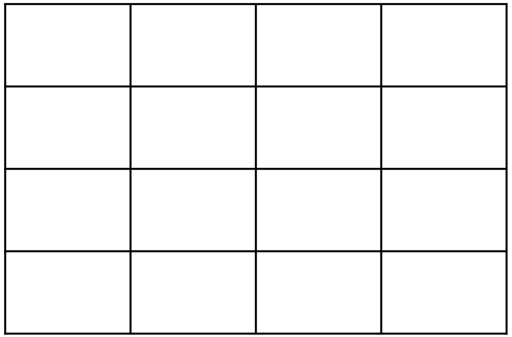
thrudb @thrudb · 23 May 2009
sucks i'm not on the west coast and will not be able to attend #nosql

Todd Lipcon @tlipcon · 23 May 2009
working on slides for the #nosql meetup in June. trying to cover all of dist systems in 40 minutes is not as easy as it sounds.

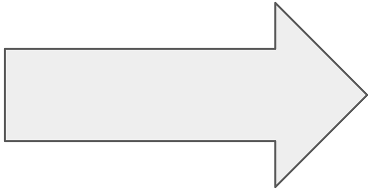
Chris Anderson @jchris · 13 May 2009
It's official - I'll be relaxifying everyone with CouchDB at #NoSQL. Thanks @skr!

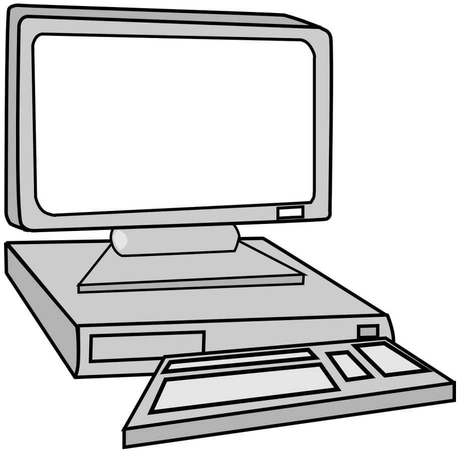
Why NoSQL?

- You have a few tables
- Few GBs max



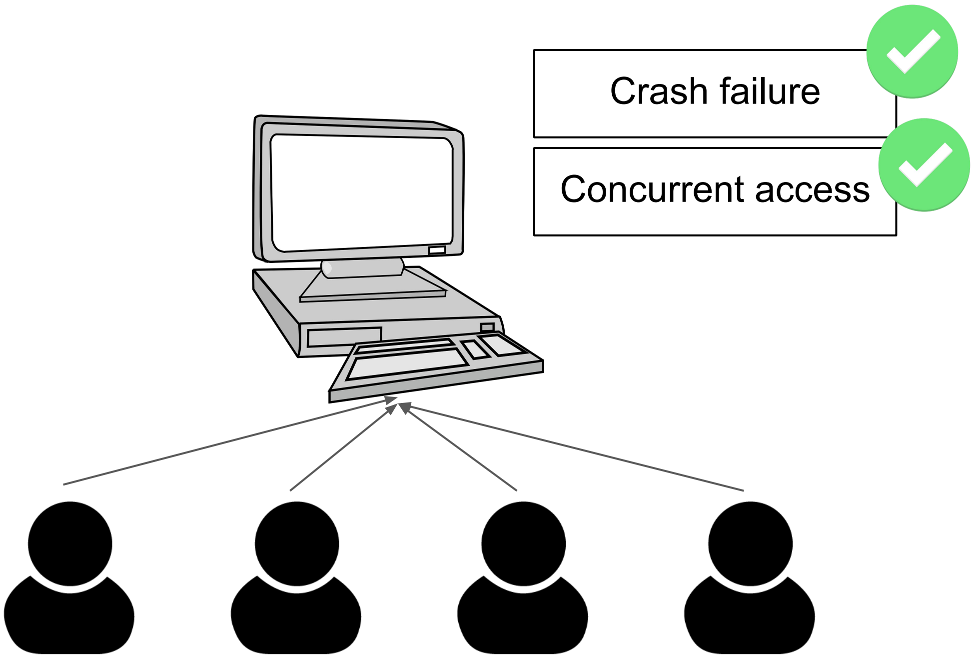
Your data fit on a **single machine**





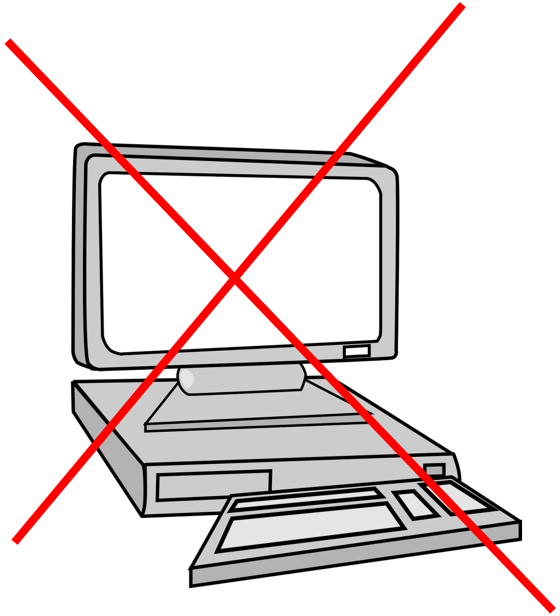
Why NoSQL?

- Or a single server



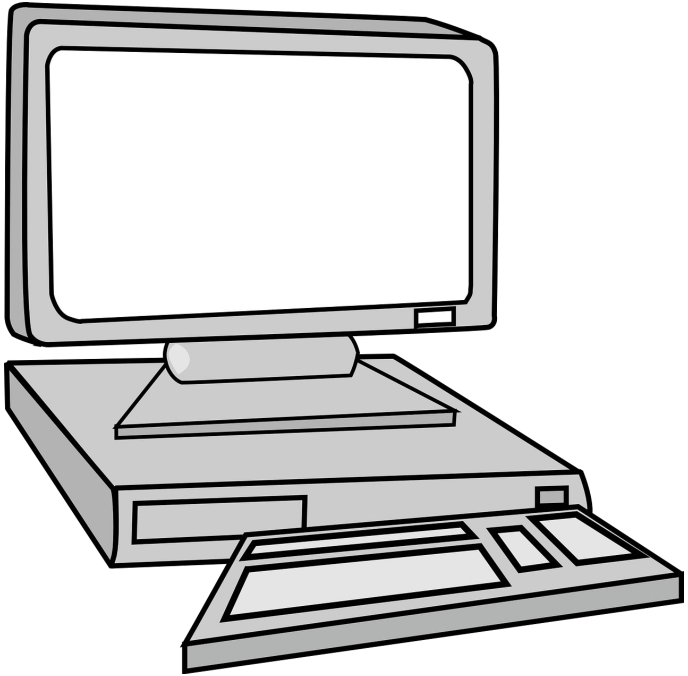
Why NoSQL?

Petabytes do not fit on a single machine



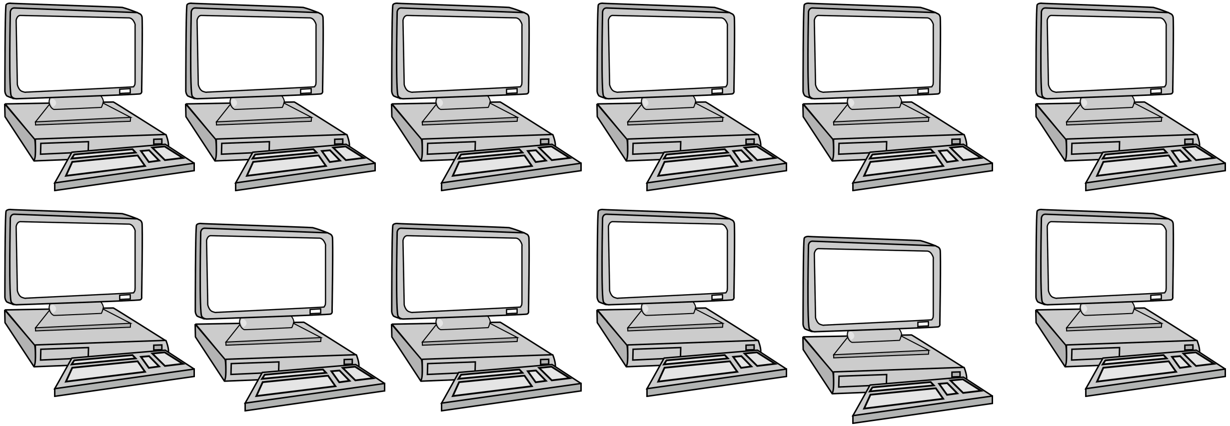
Why NOSQL

- Scale up:
  - Bigger machines
  - Cannot count on it forever
    - Moore Law is slowing down (dead!)



Why NOSQL

- Scale out
  - Many more machines that act as a single machine

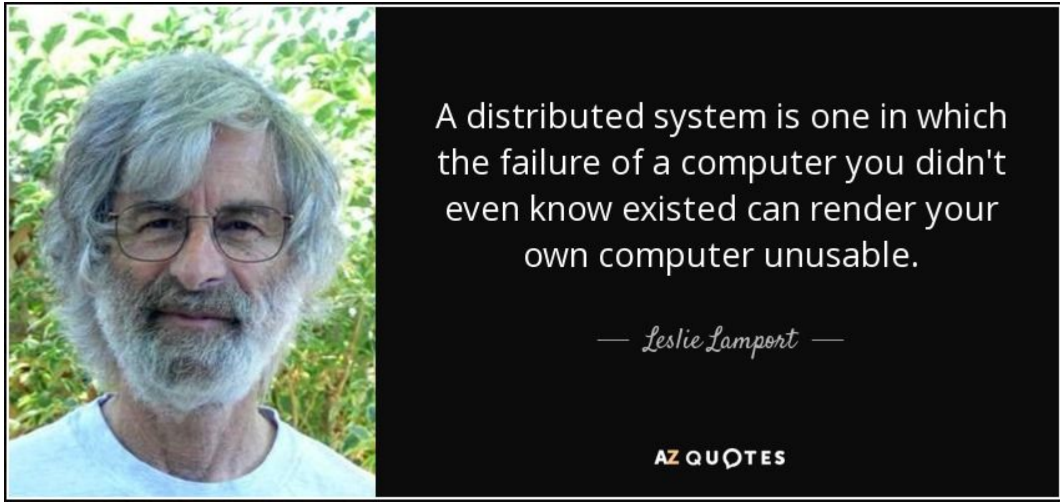


Why NOSQL

- Scale-out is adopted in practice
    - More difficult that it looks

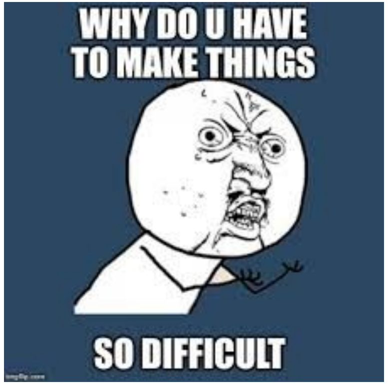
“You can have a second computer **once you’ve shown you know how to use the first one**”

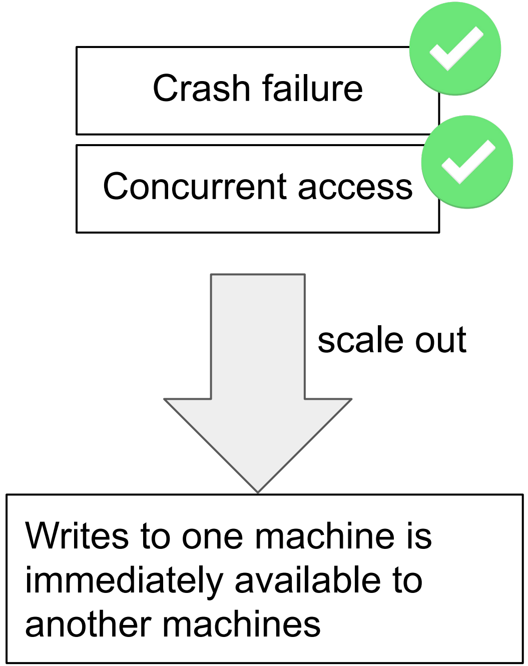
(Paul Barham)



Why NoSQL

- <Massive hand waving>
    - Very difficult to implement the Relation Model over many machines
    - Check out 50.041 if interested



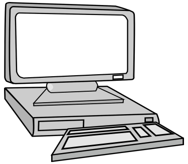


Why NoSQL

- Facebook:
  - Don’t care if I don’t see photos in real-time
  - Care if I can’t upload a photo
- Amazon:
  - Don’t care if my cart forgot an item
  - Care if I can’t add an item

Big companies are obsessed about this

NOSQL



Relational Database

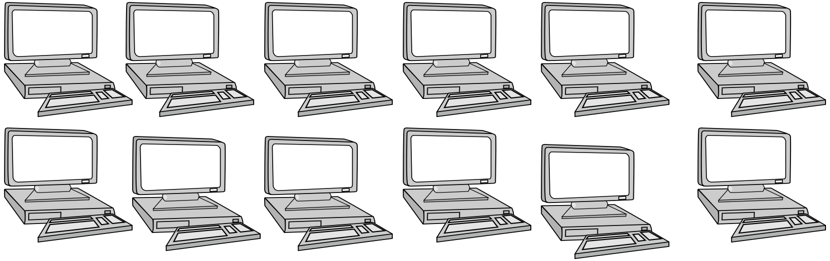
Schema: tables

SQL: join, select, ect.

Correctness

Speed*

Scale out



NoSQL Database

No Schema: just blobs

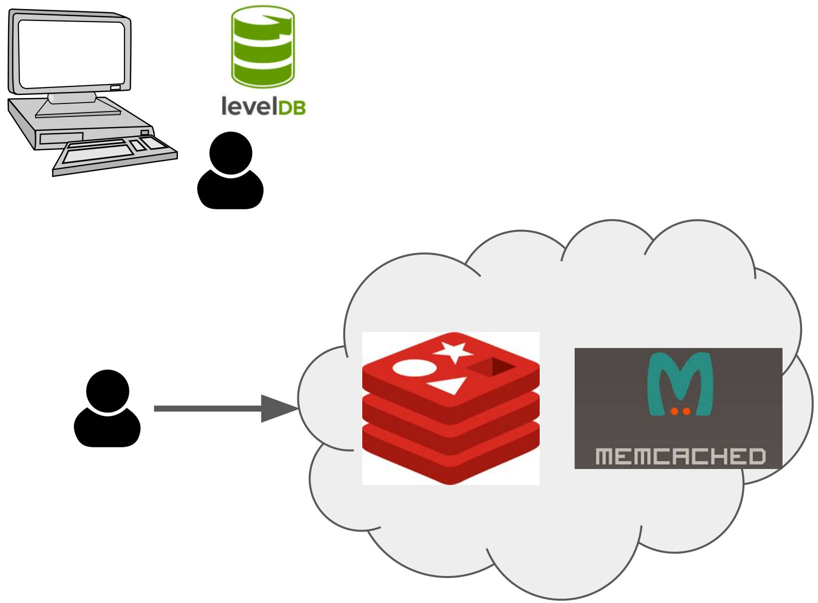
Simple API: put/get

Not always correct

Scalability

Key-Value Store

- Absolute opposite of relational database
- Local:
  - LevelDB
- Remote services
  - Redis
  - Memcached



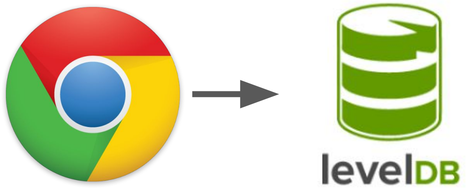
Key-Value Store

- Schema
  - key → value
  - Like a hash table
- Language:
  - put(key, value)
  - get(key)

K1	AAABBBCCC
K2	12340lks 25/11/2019
K3	.=lkj23ois09320-981
K4	abcdo093kjkap

LevelDB

- Backend of Chrome!
- What happen:
    - Put(.) on existing key?
    - Get(.) on non-existent keys?
- Demo with ldb tool

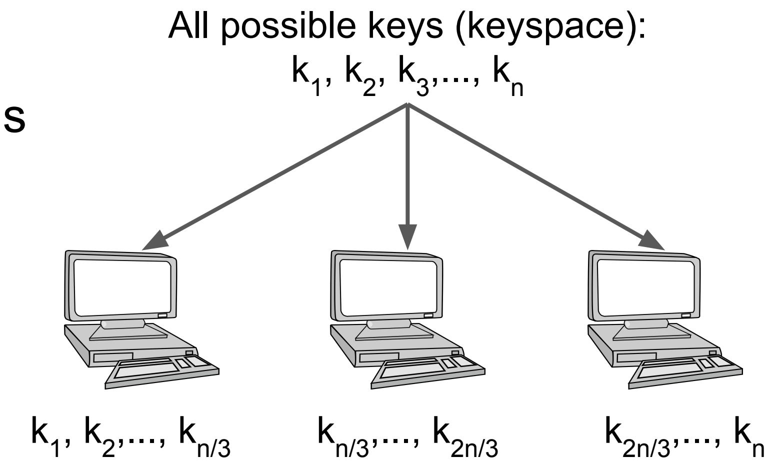



```
%./ldb db
>ls
apple1   apple2   apple3   apple4   apple5   apple6   apple7   apple8   apple9   foobar1  foobar2
foobar3  foobar4  foobar5  foobar6  foobar7  foobar8  foobar9  quxx     zebra1   zebra2   zebra3
zebra4   zebra5   zebra6   zebra7   zebra8   zebra9
>
zebra1   zebra2   zebra3   zebra4   zebra5   zebra6   zebra7   zebra8   zebra9
>get zebra1
```

Key-Value Store

don't need to search through column, like a hashmap

- The good
  - Very simple to use, behave like a map
  - Very handy in many types of applications (Web applications)
  - Very fast, like a map
  - Very scalable
    - Partition the keys into disjoint sets
    - Then distribute them over many machines



Key-Value Store

- The bad
  - You only have Put(.) and Get(.)
  - Range queries, like `select * from Grade where gpa > 3.5;`
  - Join queries not supported.

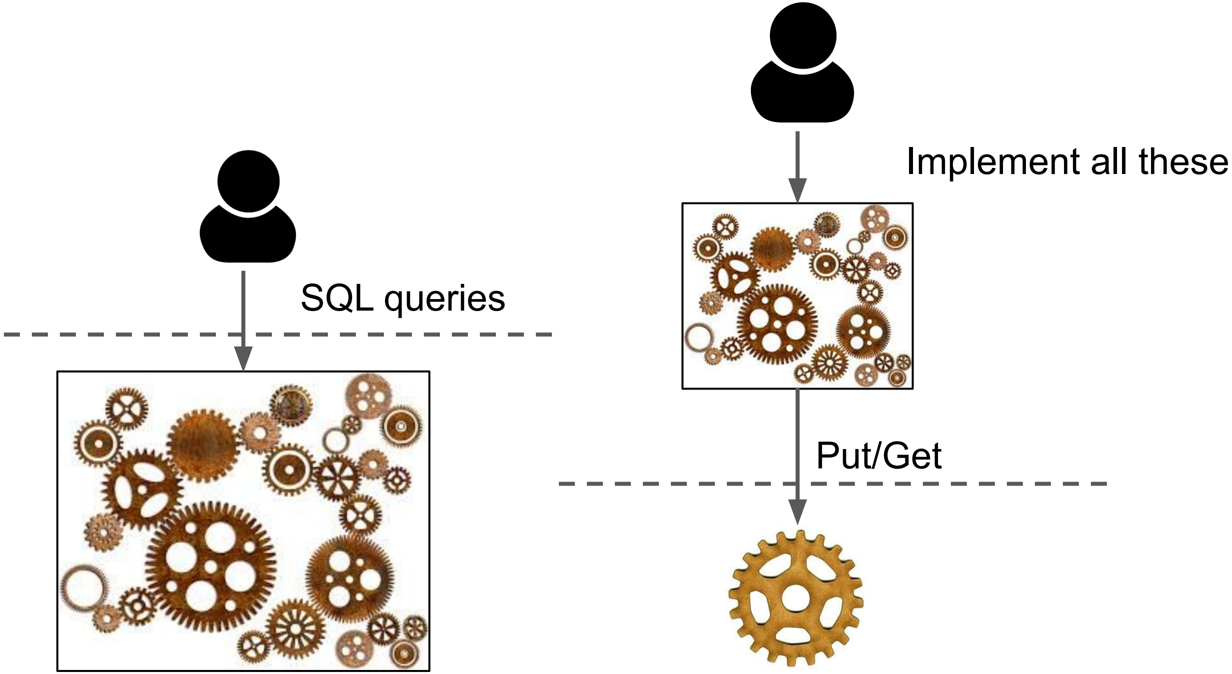
```
for each key in keyspace:
  if (int)key[value] > 3.5:
    print(key)
```

Key-Value Store

no discrimination on the data types

- The ugly:
  - Treat everything as a binary string (blob, no structure)
  - If your value contains structure, e.g., (age, name, salary)
    - Too bad for you
  - Push complexity to someone else!
    - Price will be paid, question is where, and by whom
  - Joins and range queries are costly

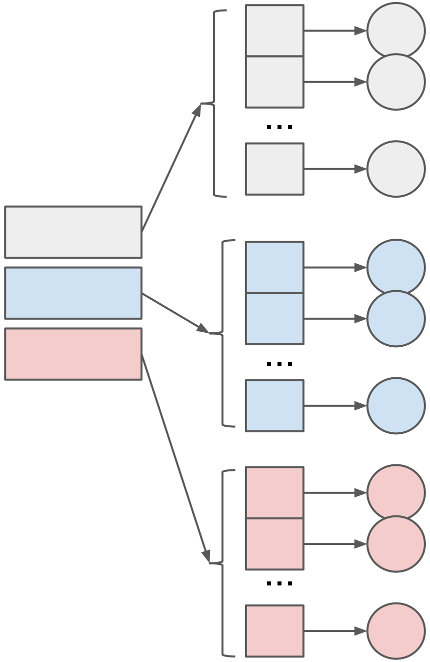
Key-Value Store



More Structured Key-Value Store

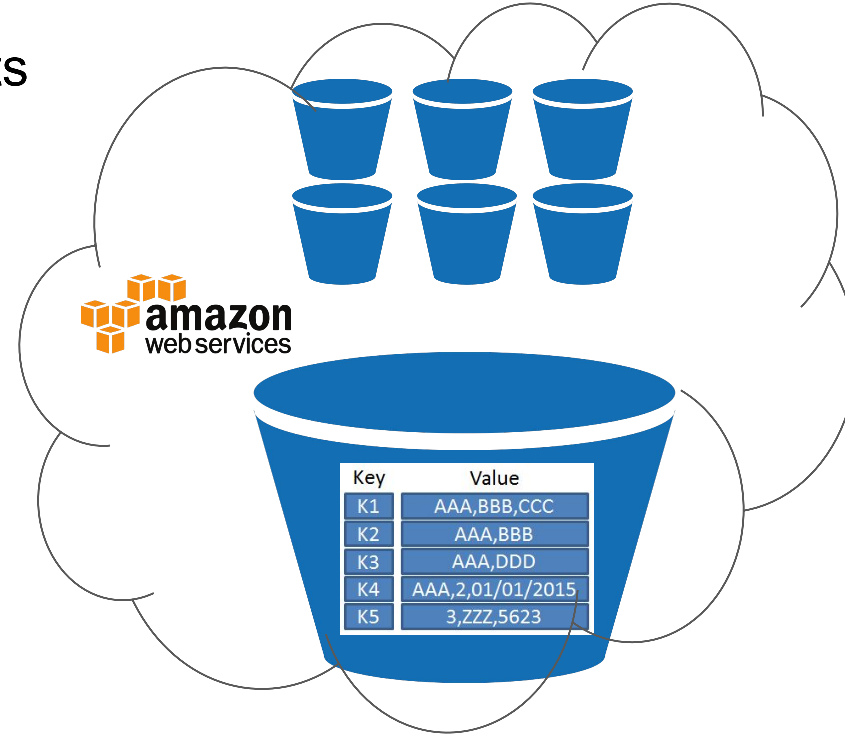
- LevelDB:
  - Flat keyspace
  - An object is identified by a single key
  - Like a typical hashtable
- (A bit) more structure
  - Organize keys into groups
  - An object is identified by (group key, object key)
  - Like a 2-level hash table

All keys are equal, just like hashmap



S3

- Amazon service
- All data divided in independent buckets
- Each bucket is a key-value store
- `put((bucket_key, object_key), value)`
- `get((bucket_key, object_key))`



S3

Loss of 1 object per 10K years

Durability & Data Protection

Q: How durable is Amazon S3?

Amazon S3 Standard, S3 Standard–IA, S3 One Zone-IA, and S3 Glacier are all designed to provide 99.999999999% durability of objects over a given year. This durability level corresponds to an average annual expected loss of 0.000000001% of objects. For example, if you store 10,000,000 objects with Amazon S3, you can on average expect to incur a loss of a single object once every 10,000 years. In addition, Amazon S3 Standard, S3 Standard-IA, and S3 Glacier

RocksDB

- Built on LevelDB
  - Used inside Facebook
- Column families
  - Grouping keys in a column family
- Demo with Rocksdb's ldb tool

Column Families provide a way to logically partition the database. Some interesting properties:
- Atomic writes across Column Families are supported. This means you can atomically execute Write({cf1, key1, value1}, {cf2, key2, value2}).
- Consistent view of the database across Column Families.
- Ability to configure different Column Families independently.
- On-the-fly adding new Column Families and dropping them. Both operations are reasonably fast.

Redis

- In memory vs. disk
    - All data DRAM
    - Very, very fast

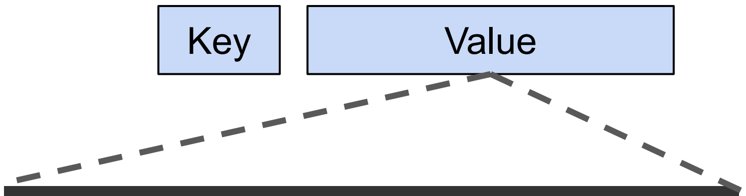
Numbers Everyone Should Know

L1 cache reference	0.5 ns
Branch mispredict	5 ns
L2 cache reference	7 ns
Mutex lock/unlock	100 ns
Main memory reference	100 ns
Compress 1K bytes with Zippy	10,000 ns
Send 2K bytes over 1 Gbps network	20,000 ns
Read 1 MB sequentially from memory	250,000 ns
Round trip within same datacenter	500,000 ns
Disk seek	10,000,000 ns
Read 1 MB sequentially from network	10,000,000 ns
Read 1 MB sequentially from disk	30,000,000 ns
Send packet CA->Netherlands->CA	150,000,000 ns

Google

Redis

- More structure
  - Value has richer type than just blob
  - List
  - Set
  - Even key-value collection




```
List: ["a", "b", "c", "c",...]
Set: {"a", "1", "b"..{
Hash: {"a" => "1",
       "1" => "093",
       "test" = "succeed",
       ...}
```

Redis

- Very popular backend for Web applications
  - Because every ms counts
  - Memcached’s cousin
  - Too popular that it’s in Ubuntu’s standard *apt* repositories

Redis

- Demo with redis-cli

```
lpush course 12345
lpush course istd_50043
llen course
lindex course 0
```

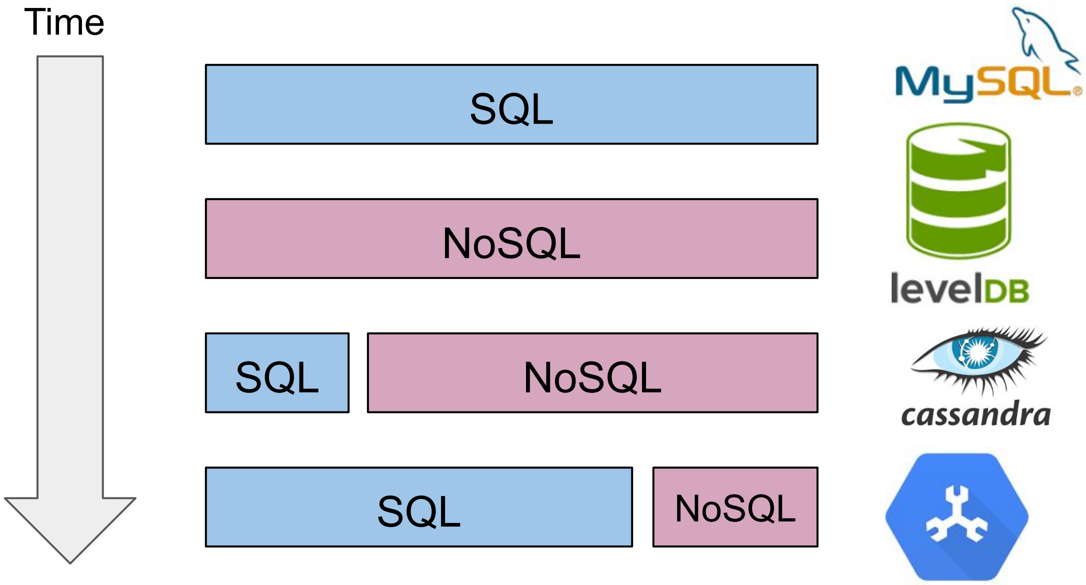
```
hset student age 25
hset student gpa 3.5
hvals student
hkeys student
```

Summary

- NoSQL chooses scalability over everything else
- Key-value store:
  - LevelDB: bare bone
  - S3/RocksDB: column families
  - Redis: in-memory + remote service + typed values.

Final Remark

On History



NoSQL still extremely useful

But people want ease-of-use

- Appreciate the value of SQL more and more