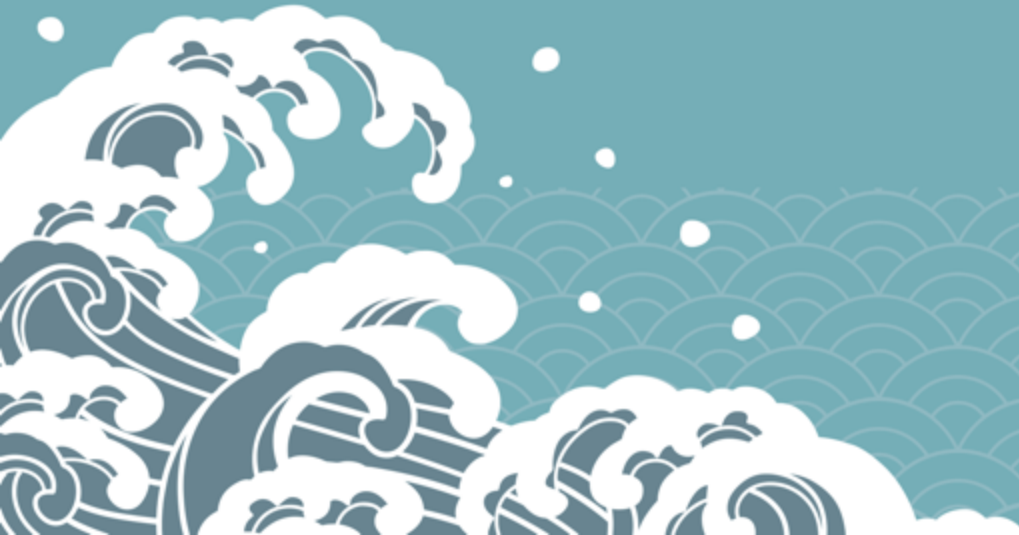




From NoSQL to Mo'SQL

Gordon Guthrie



Lots of people have worked on Riak TS

Andrei Zavada

Andy Till

Bill Soudan

Brett Hazen

Brian McClain

Bryce Kerley

Derek Somogyi

Erik Johnson

Erik Leitch

Heather McKelvey

John Daily

Lauren Rother

Paul Hagan

Pavel Hardak

Seema Jethani

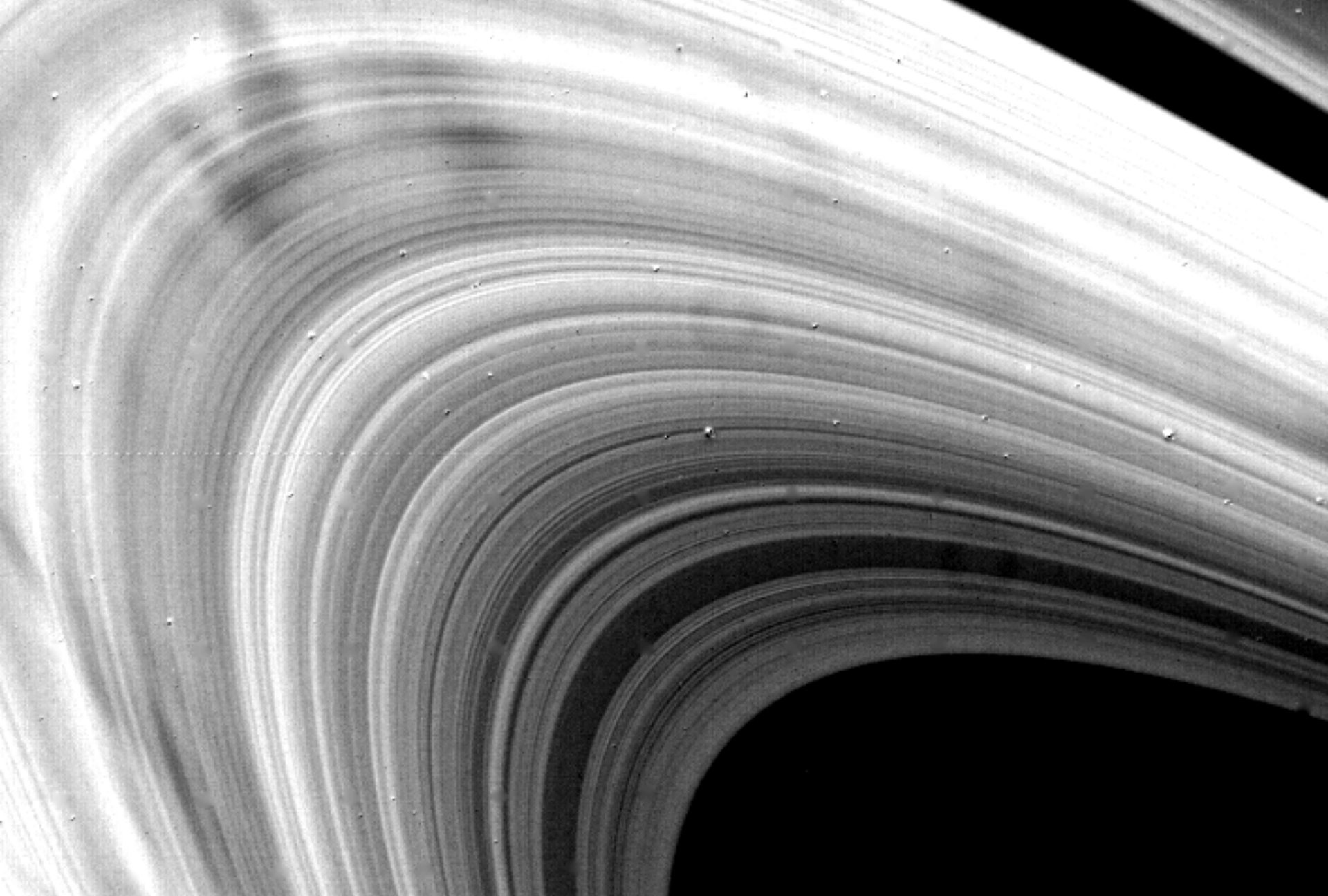


SQL and NoSQL?

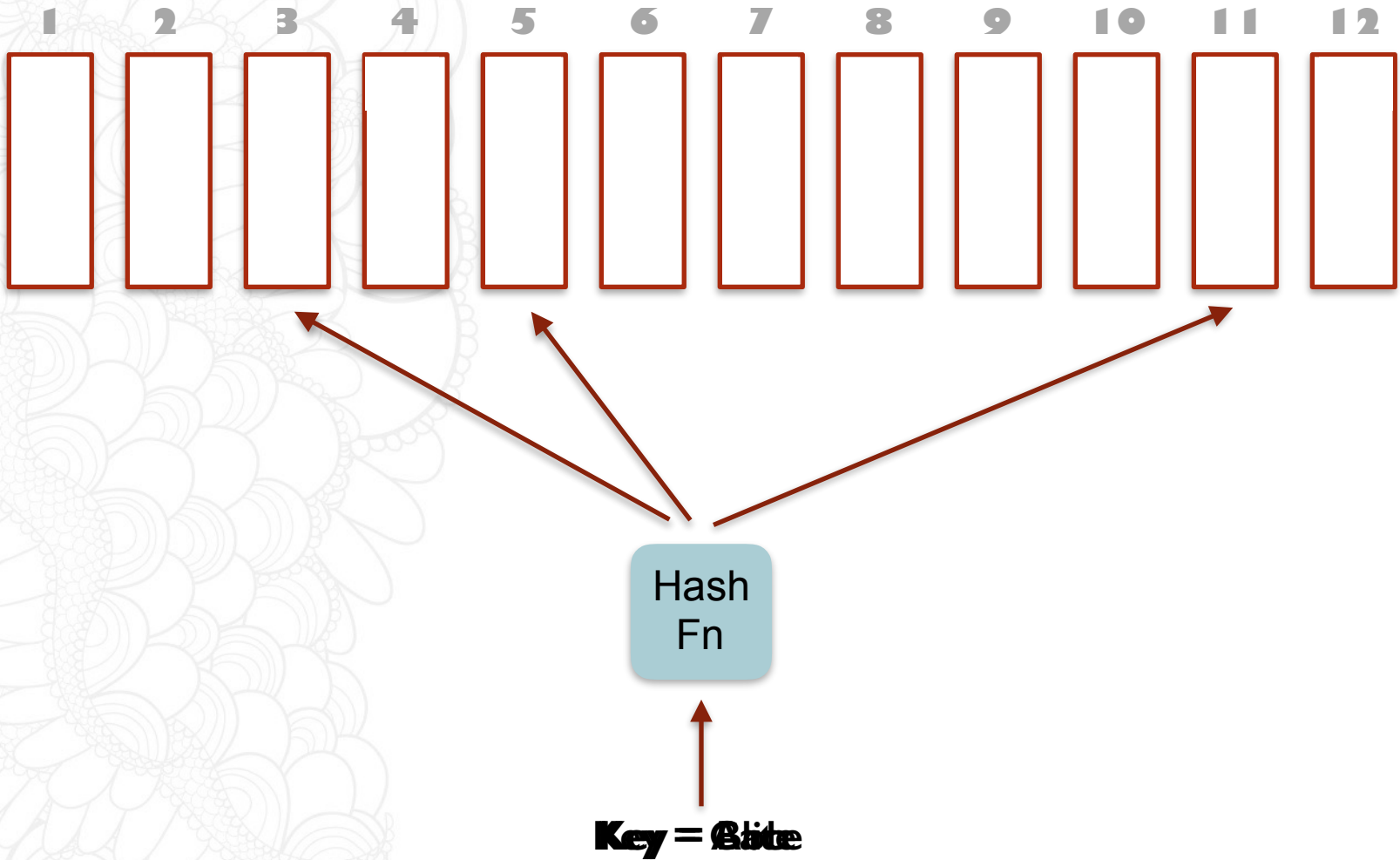
a match made in hell?



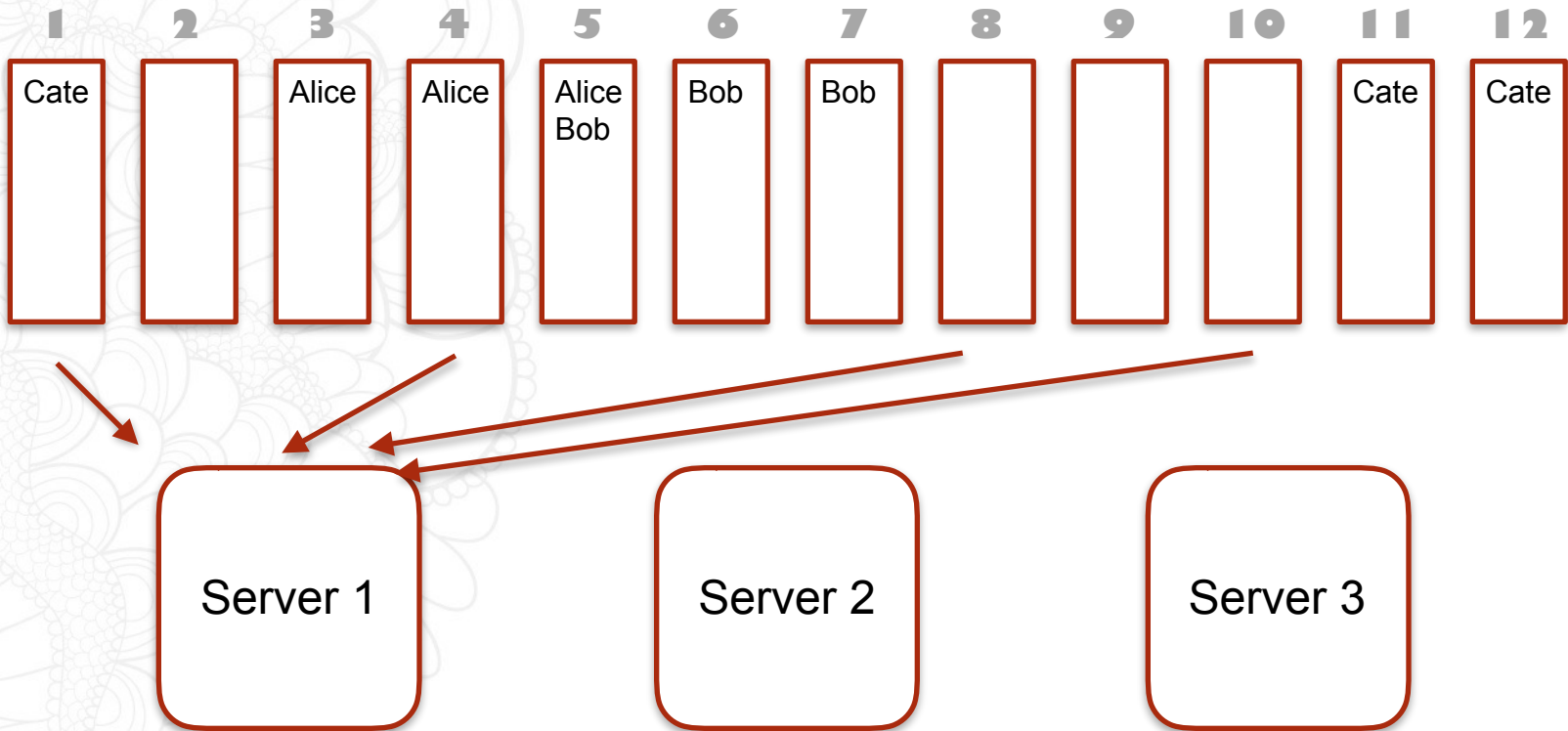
```
select key, value from bucket where key='ModelT';
```



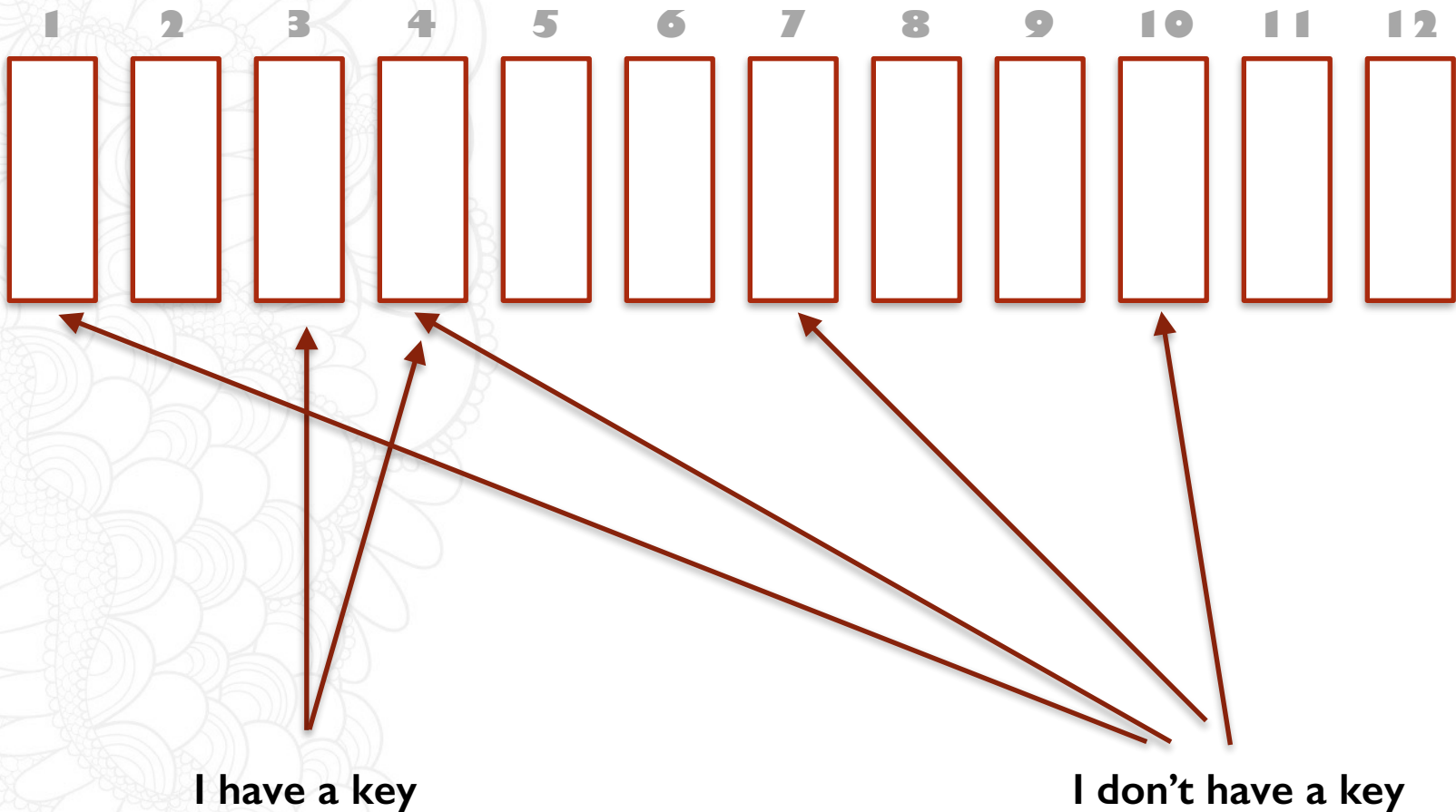
The ring itself is a pain right up yer bahooky in slides



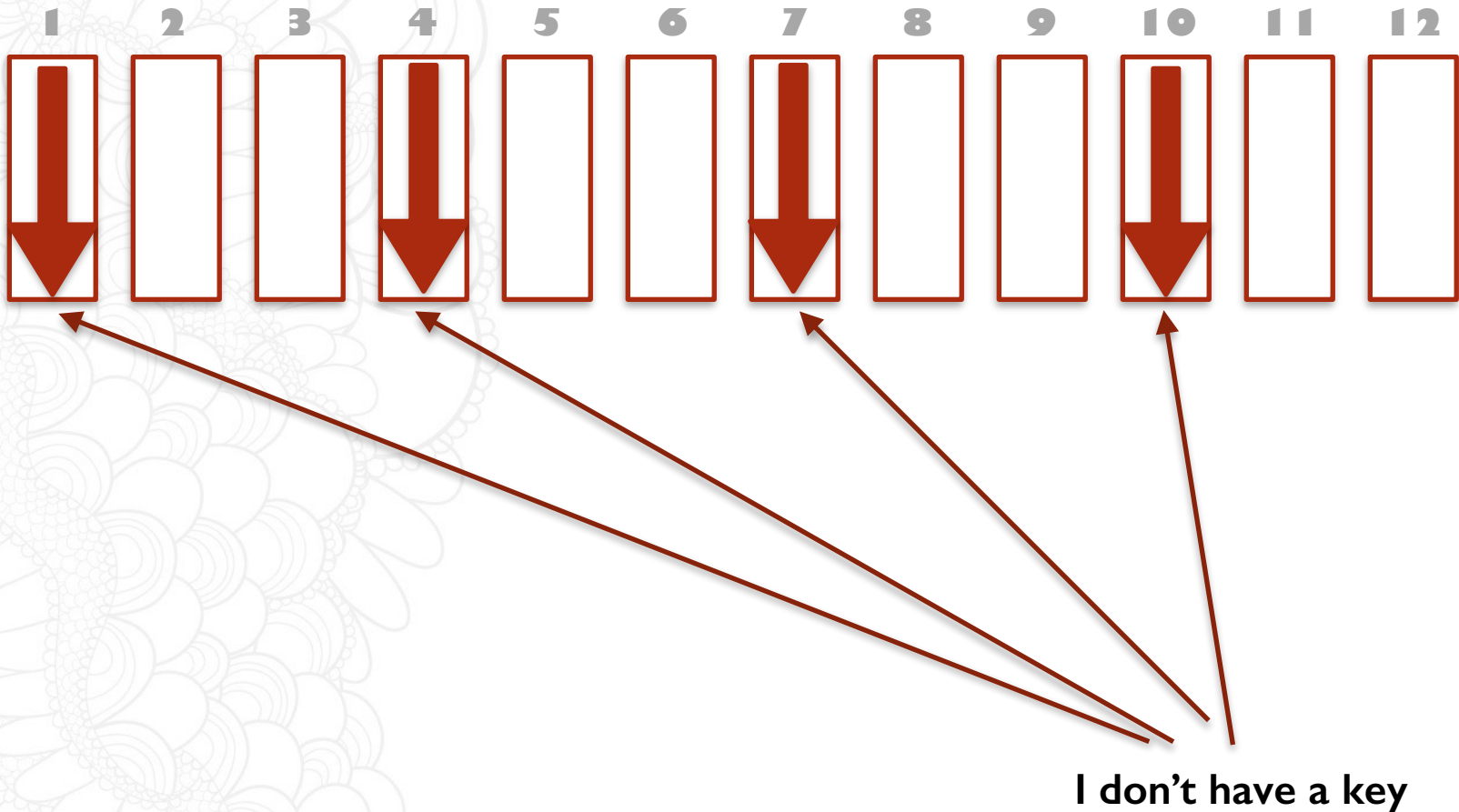
The ring is logical on physical nodes



This give you two query modes



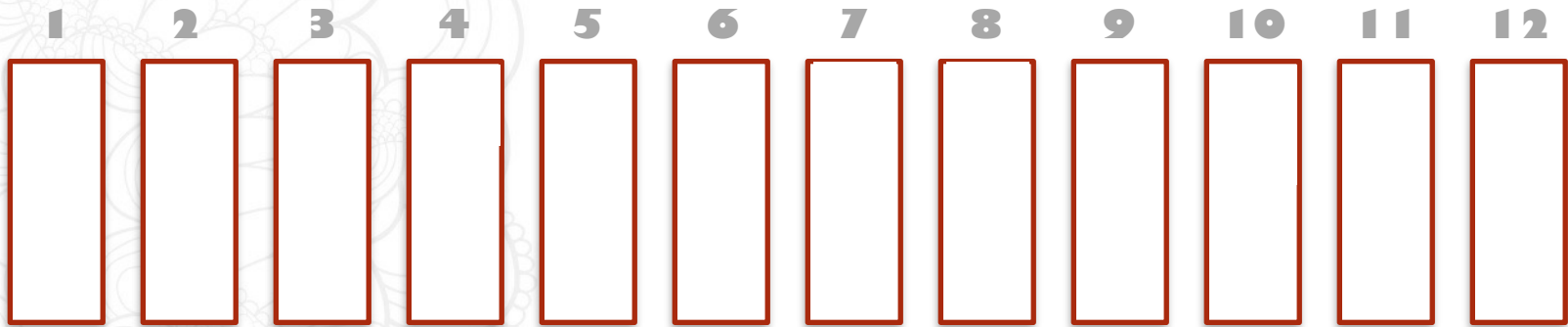
And its not just about the travelling



In summary

- You can talk to **2 servers** if you have a key
- You must talk to **all servers** if you don't

Lets see how TS works



Hash
Fn



Key = (Alice, quantum(Time, 10, 's'))

Different access patterns

- You can talk to **2 servers** if you want to query the data across 1 quantum
- Add another 2 for 2 quanta
- eventually must talk to **all servers**

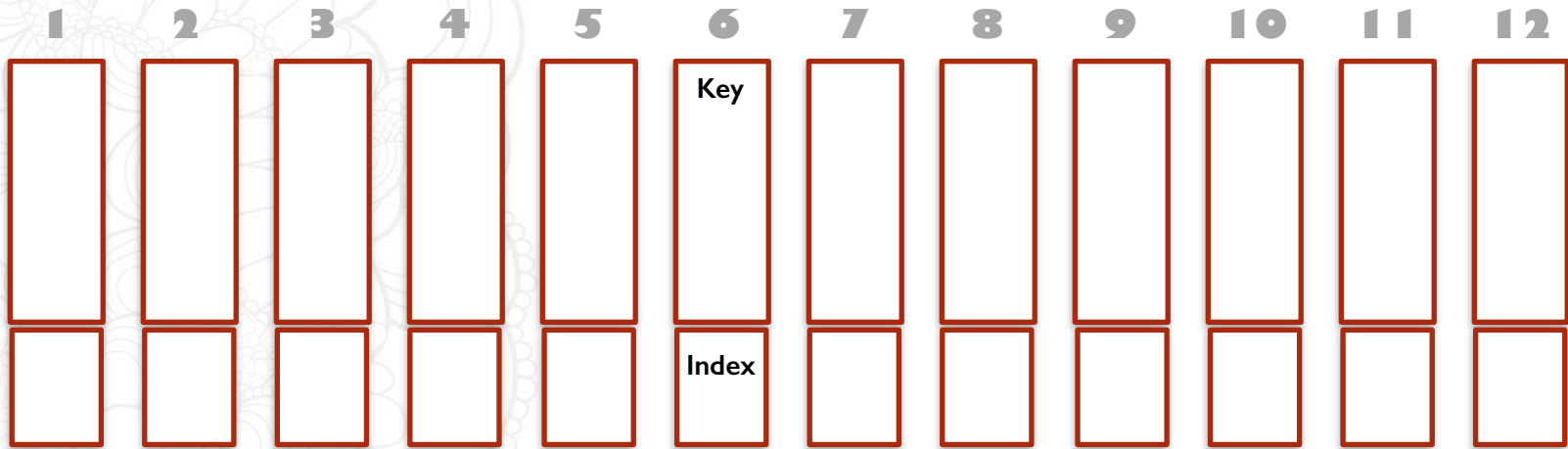
There are trade-offs

- You can make the quanta **bigger** which means less trips to read more data
- **but** your write pattern gets lumpier with higher risk of hot spots



**Can we
improve that?**

We have $2i$ indices



To get a list of keys that match an index you visit $1/3$ of nodes + 1 and make an index read

You've seen the movie

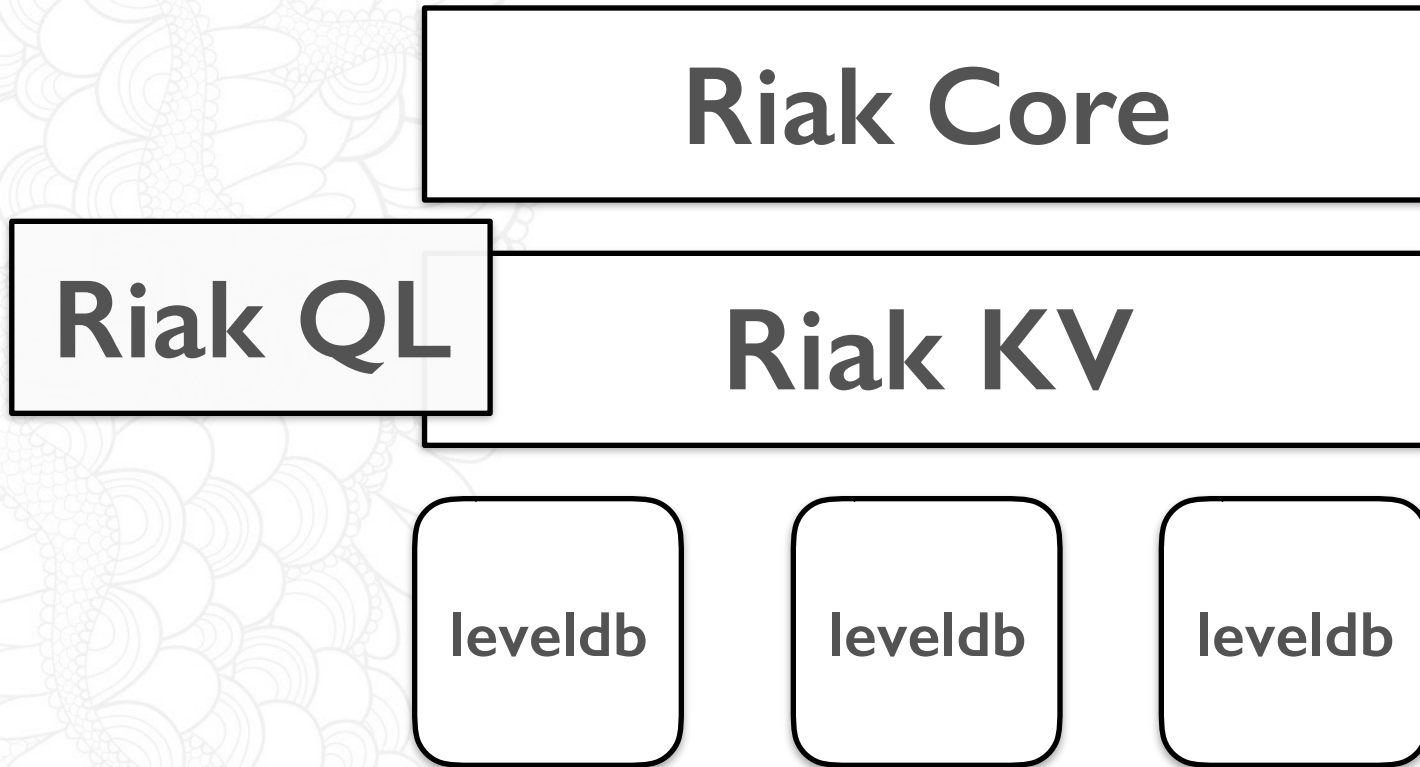


The SQL

Why SQL

- Everybody knows it/low barrier to entry
- Good tooling
- Its a declarative language, but extendable

High level architecture



KV stores are know-nothing wrt values

```
CREATE TABLE GeoCheckin
(
  id          SINT64    NOT NULL,
  time       TIMESTAMP NOT NULL,
  region     VARCHAR   NOT NULL,
  state      VARCHAR   NOT NULL,
  weather    VARCHAR   NOT NULL,
  temperature DOUBLE,
  PRIMARY KEY (
    (id, QUANTUM(time, 15, 'm')),
    id, time
  )
)
```

load Erlang ddl
helper module
as .beam

leex/yecc

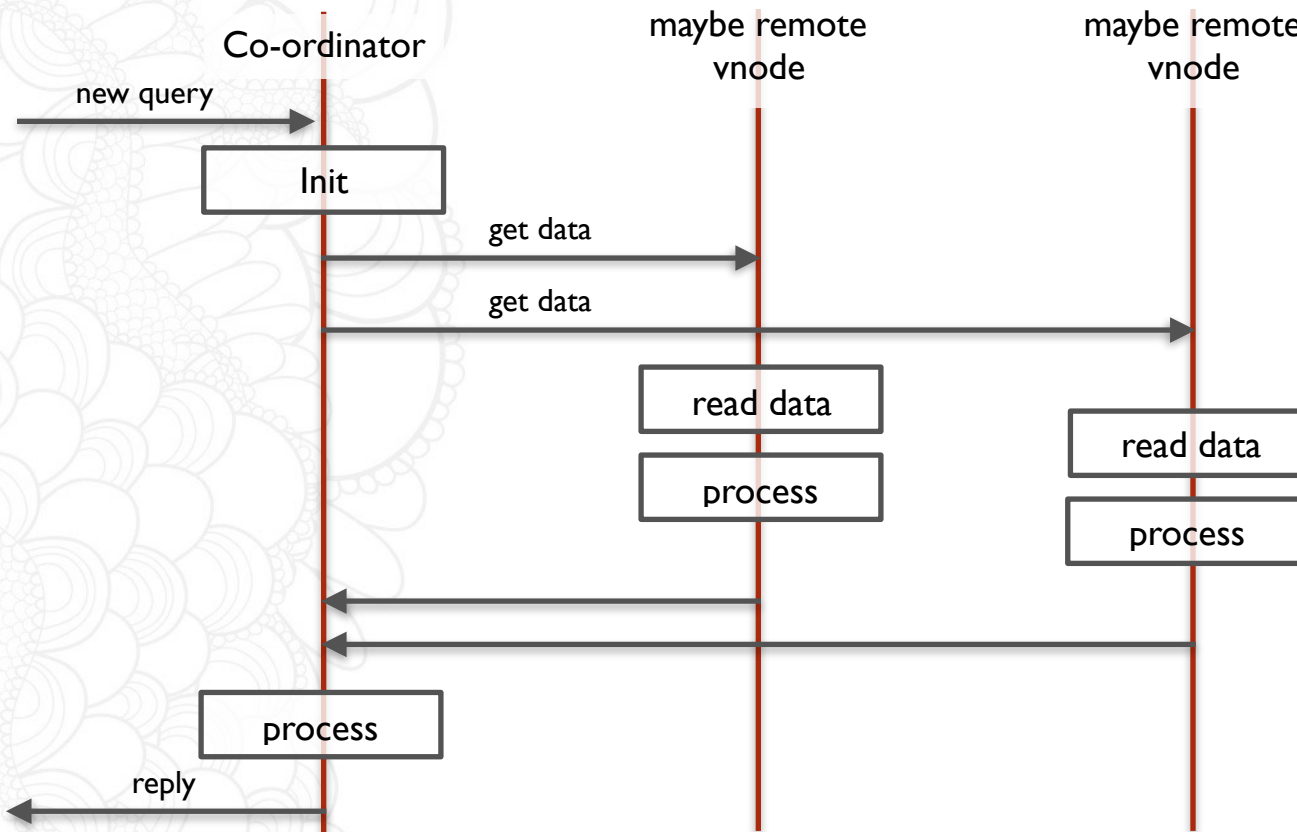
Erlang data
structure
(a record)

distribute round
cluster with
riak_core

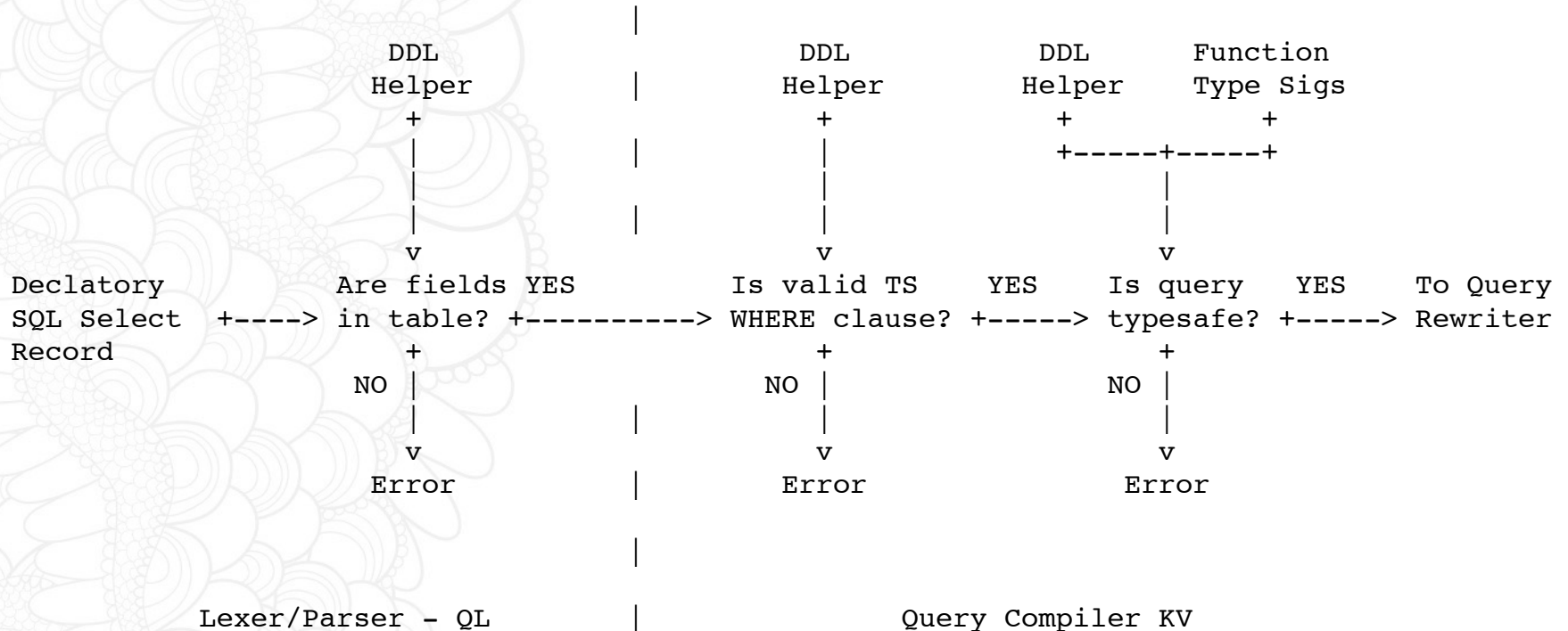
merl

riak core
metadata

The query system is distributed



Quite complex query validation process



Unroll all the SQL

```
SELECT AVG(temp) FROM mytimeseries WHERE family = 'myfamily' AND series = 'myseries'
AND timestamp > 1233 AND timestamp < 6789 AND temp > 18;
```

<----Erlang Coordinator----->

<----LevelDB C++ Code----->

<----Network----->

```
+ FROM
|
| SELECT SUM(STemp)/SUM(NoTemp)
|
| GROUP BY []
|
| ORDER BY []
|
+ WHERE []
```

Chunk1

Chunk2

```
+ FROM mytable on vnode X
|
| SELECT SUM(temp) AS STemp, COUNT(temp) AS NoTemp
|
| GROUP BY []
|
| ORDER BY []
|
+ WHERE + start_key = {myfamily, myseries, 1233}
|       | end_key   = {myfamily, myseries, 4000}
+ temp > 18
|
+ FROM mytable on vnode Y
|
| SELECT SUM(temp) AS STemp, COUNT(temp) AS NoTemp
|
| GROUP BY []
|
| ORDER BY []
|
+ WHERE + start_key = {myfamily, myseries, 4001}
|       | end_key   = {myfamily, myseries, 6789}
+ temp > 18
```

Query rewriting in a nutshell

declarative SQL
(decorated with execution hints)

transform syntax

preserve semantics

Query Plan
(executable fragments)

Schematic SQL Operations

Data On Disk

```
+-----+-----+-----+
| Col1  | Col2  | Col3  |
| Type1 | Type2 | Type3 |
+-----+-----+-----+
```

```
+-----+-----+-----+
| Val1a | Val1b | Val1c |
+-----+-----+-----+
| Val2a | Val2b | Val2c |
+-----+-----+-----+
| Val3a | Val3b | Val3c |
+-----+-----+-----+
```


All the fragments meet this pattern - row ops

Col1	Col2	Col3
Type1	Type2	Type3

Operation
←-----+

Col1	Col2	Col3
Type1	Type2	Type3

Val1a	Val1b	Val1c
Val3a	Val3b	Val3c
Val6a	Val6b	Val6c
Val5a	Val5b	Val5c

WHERE
GROUP BY
ORDER BY
LIMIT
DISTINCT
HAVING

Val1a	Val1b	Val1c
Val2a	Val2b	Val2c
Val3a	Val3b	Val3c
Val4a	Val4b	Val4c
Val5a	Val5b	Val5c
Val6a	Val6b	Val6c

Row and column operations

ColX	ColY
Type1	Type2

Operation

←-----+

Val1X	Val1Y
Val2X	Val2Y

SELECT

Col1	Col2	Col3
Type1	Type2	Type3

Val1a	Val1b	Val1c
Val2a	Val2b	Val2c
Val3a	Val3b	Val3c

Column Name Vector Ops

ColX	ColY	ColZ
Type1	Type2	Type3

Operation

←-----+

Val1a	Val1b	Val1c
Val2a	Val2b	Val2c
Val3a	Val3b	Val3c
Val4a	Val4b	Val4c

AS

Col1	Col2	Col3
Type1	Type2	Type3

Val1a	Val1b	Val1c
Val2a	Val2b	Val2c
Val3a	Val3b	Val3c
Val4a	Val4b	Val4c

Executable Fragments

```
{where, [  
  {and_,  
    {'=', <<"sequence_number">>, {integer, 2321}},  
    {'=', <<"time">>, {integer, 1400497861762723}}  
  ]  
}}
```

YASL's all the way down

How much SQL?

- **SELECT**
- **WHERE**
- **GROUP BY**
- **ORDER BY/LIMIT** is being worked on
- functions:
 - **AVG/MEAN**
 - **MAX**
 - **MIN**
 - **SUM**
 - **COUNT**
 - **STDDEV/STDDEV_SAMP**
 - **STDDEV_POP**

What does 'decorated with execution hints mean'?

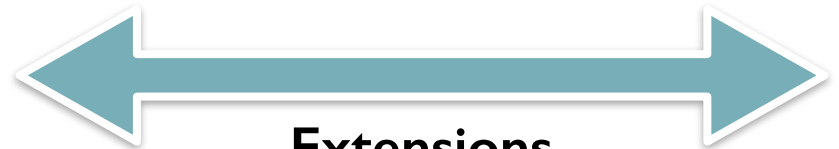
```
SELECT * FROM mytable;
```

```
SELECT * FROM mytable LIMIT 1000;
```

```
SELECT * FROM mytable WITH frobulate=on;
```



**Standard SQL
works in Tools**



**Extensions
Set as table defaults**

What does the future hold?



Riak has other sorts of co-located data

- CRDT sets look like colocation
 - 100,000 elements in a CRDT set
 - written to a vnode under a key
- performance issues
 - monolithic object
 - read 100,000 element set from disk
 - operate on it
 - write it back to disk

Enter big sets!

How would that work?

Table In Shell

ColX	ColY
Type1	Type2

SQL Query



Val1X	Val1Y
Val2X	Val2Y

Data On Disk

Table Schemas

Col1	Col2	Col3
Type1	Type2	Type3

Set off Maps

Val1a	Val1b	Val1c
Val2a	Val2b	Val2c
Val3a	Val3b	Val3c

High level architecture

Riak Core

Riak QL **Big Sets/Big Maps**

leveldb

leveldb

leveldb

...and because maps are recursive and can contain sets which can be maps

- we have prototyped subsets of relational queries
 - left or inner joins

Much excites!

You still have the 2 query paths...

**Talk to all
servers**



**Talk to 2
servers**

Can we 'steal' some of the causality information from Delta ops and use that to build single point access eventually consistent indices?

Dunno!

