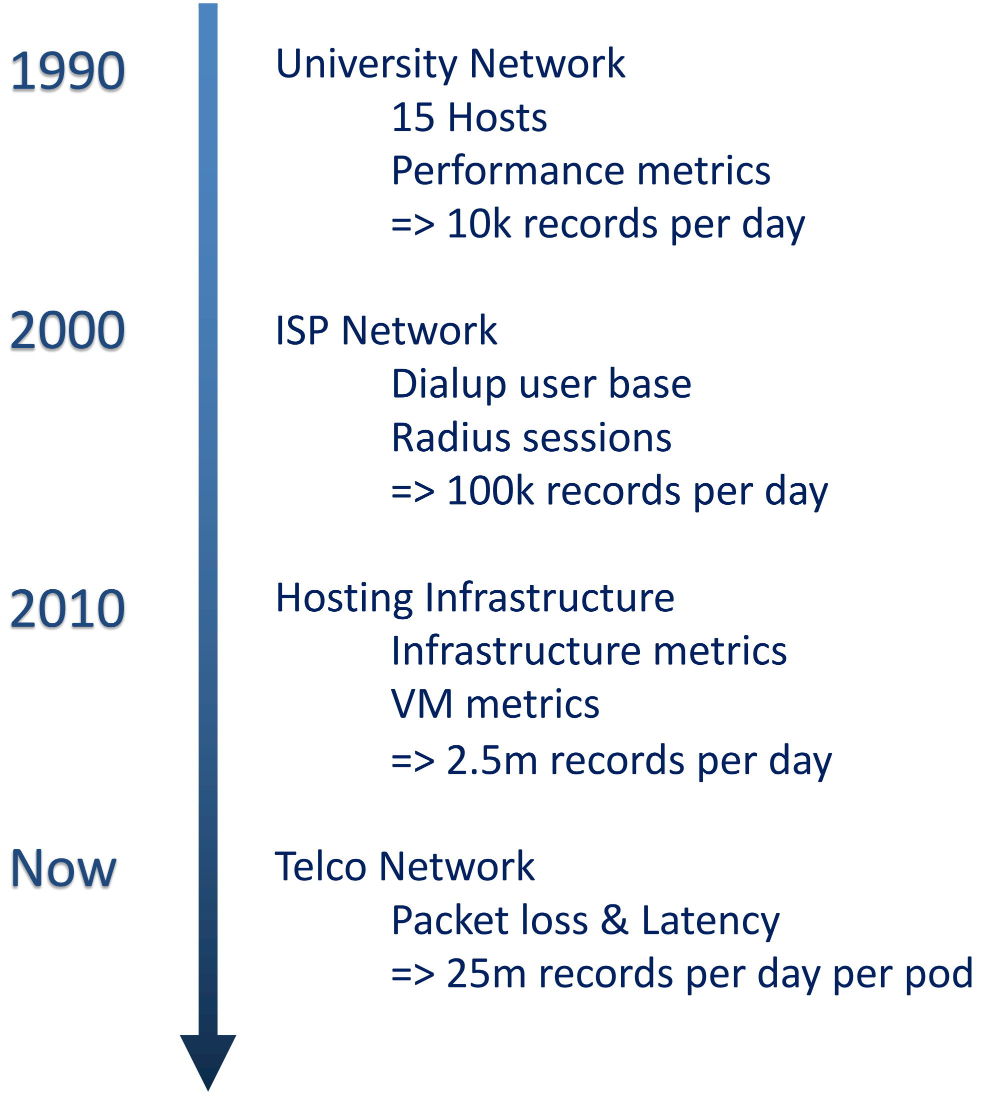
OvertheWire

Timeseries data at scale for the masses

David Hughes

Timeseries Data

Growth over the ages



1990

University Network

- 15 Hosts
- Performance metrics
- => 10k records per day

2000

ISP Network

- Dialup user base
- Radius sessions
- => 100k records per day

2010

Hosting Infrastructure

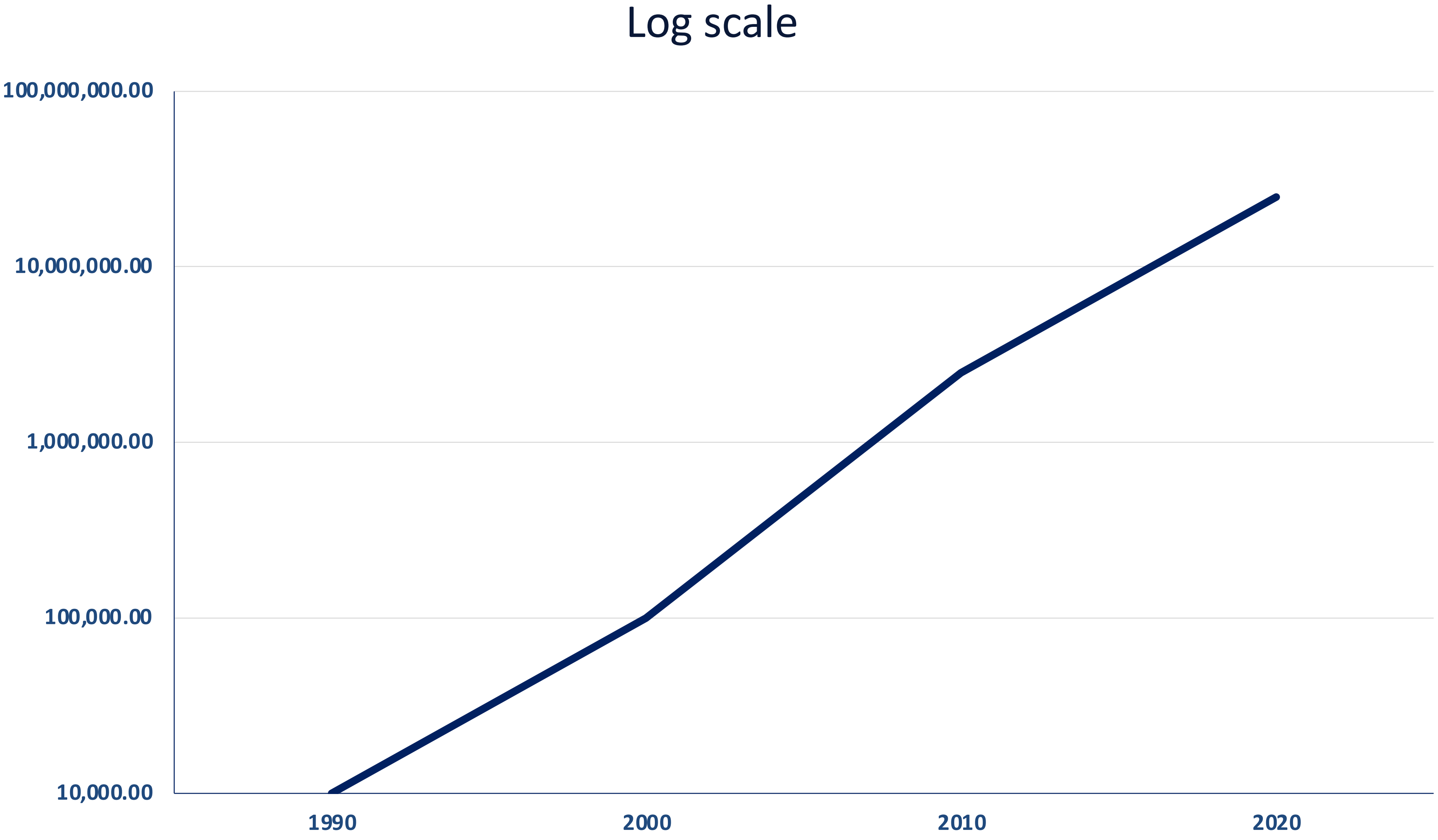
- Infrastructure metrics
- VM metrics
- => 2.5m records per day

Now

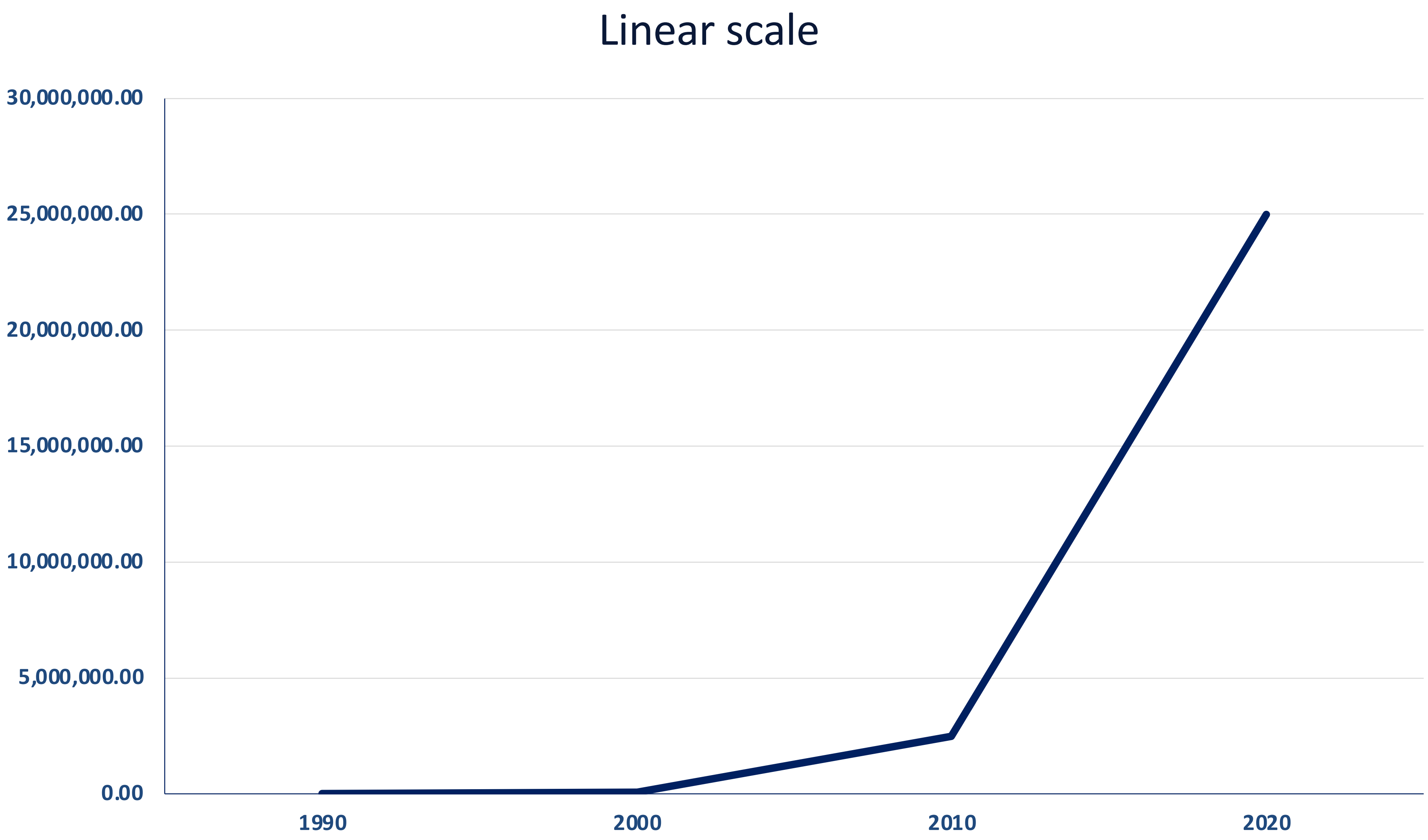
Telco Network

- Packet loss & Latency
- => 25m records per day per pod

Data Growth

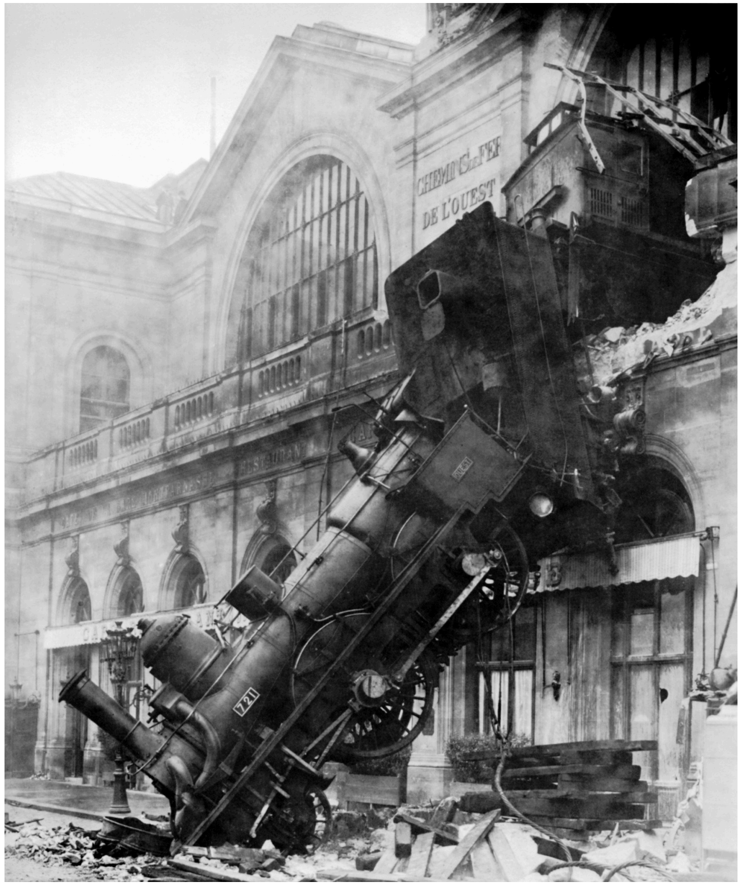

Log scale
100,000,000.00
10,000,000.00
1,000,000.00
100,000.00
10,000.00
1990
2000
2010
2020

Data Growth


Linear scale
30,000,000.00
25,000,000.00
20,000,000.00
15,000,000.00
10,000,000.00
5,000,000.00
0.00
1990
2000
2010
2020

Data Growth

Train wreck scale



Dealing with Data

The decade of NoSQL

- “Web Scale”
- Non Tabular
  - ✓ More Flexible
  - ✓ Higher Performance
- Designed to scale-out or cluster
- So many forms to choose from
- Just blame Google 😁

A NoSQL For Any Occasion

- **Key Value Store**
  E.g. Memcache
  Simple and high performance.
  Perfect for local caches

- **Structured Store**
  E.g. Redis
  A Key Value Store with typed data.
  Also great for local caches

- **Document Database**
  E.g. MongoDB
  Key Value where the value is a document, and each document may have a different structure.

- **Graph Database**
  E.g. Neo4J
  If your world looks like a graph then this is the business !!

Tabular data by any other name

is still tabular

```
People : [
        {
                first: ”Kanye”,
                last: “West”,
                age: 42
        },
        {
                first: ”Ed”,
                last: “Sheeran”,
                age: 28
        },
        {
                first: ”Bono”,
                age: 59
        }
]
```

First	Last	Age
Kanye	West	42
Ed	Sheeran	28
Bono		59

My Data Wish List

1. Tabular data
2. SQL Interface
3. APIs for every language I can think of
4. Grow in data depth without performance hit
5. Grow in data width without performance hit

Deliver the x10 scale promise without sacrificing performance, flexibility, or ease of use

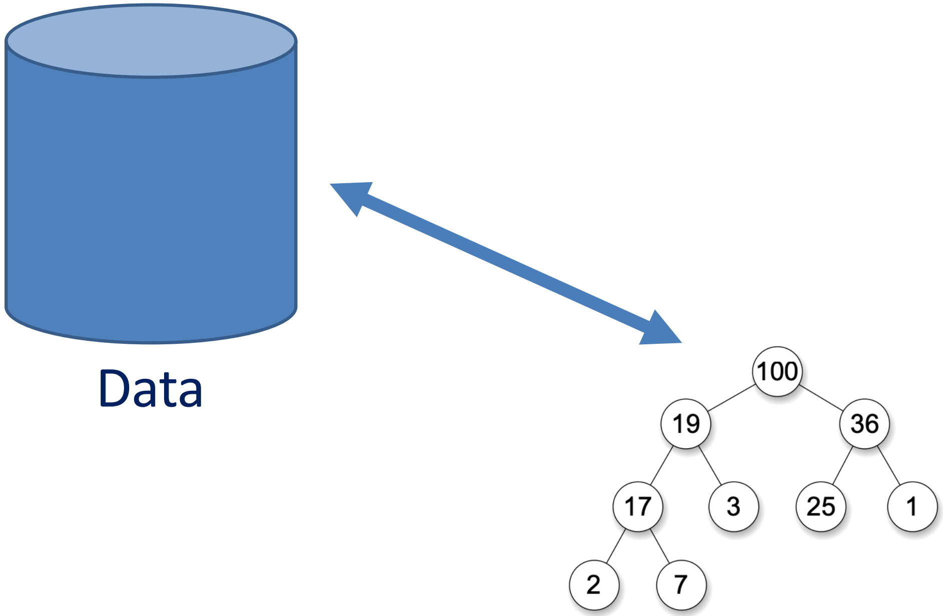
TimescaleDB

A PostgreSQL extension

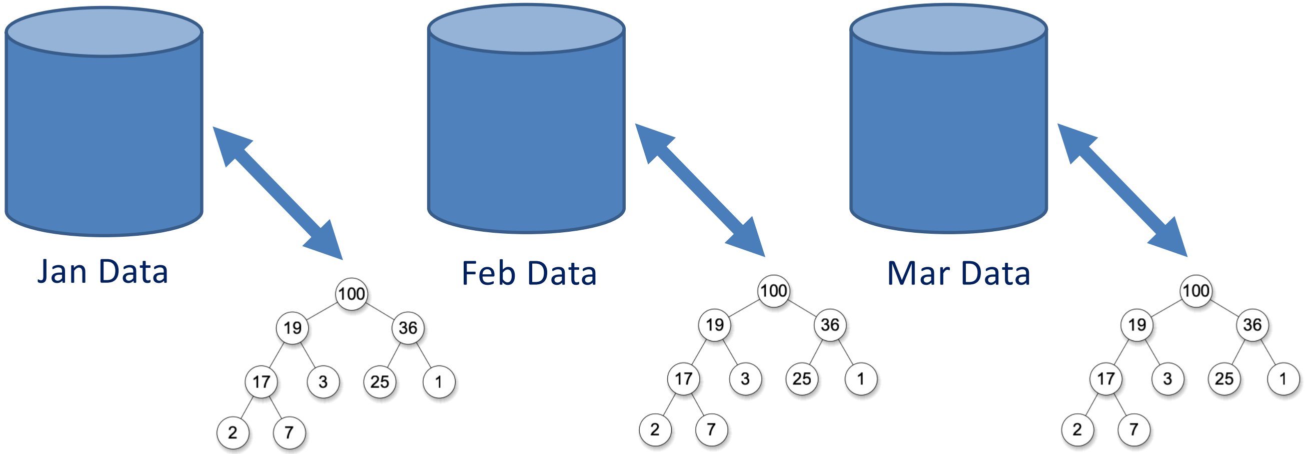
Timescale

- Specifically written for time series data at scale
- Plugs seamlessly into the Postgres query planner
- Hides the complexity from the developer. Standard SQL
- Works with any tool that works with PostgreSQL
- No retraining if your teams already use PostgreSQL
- Opensource or Commercial

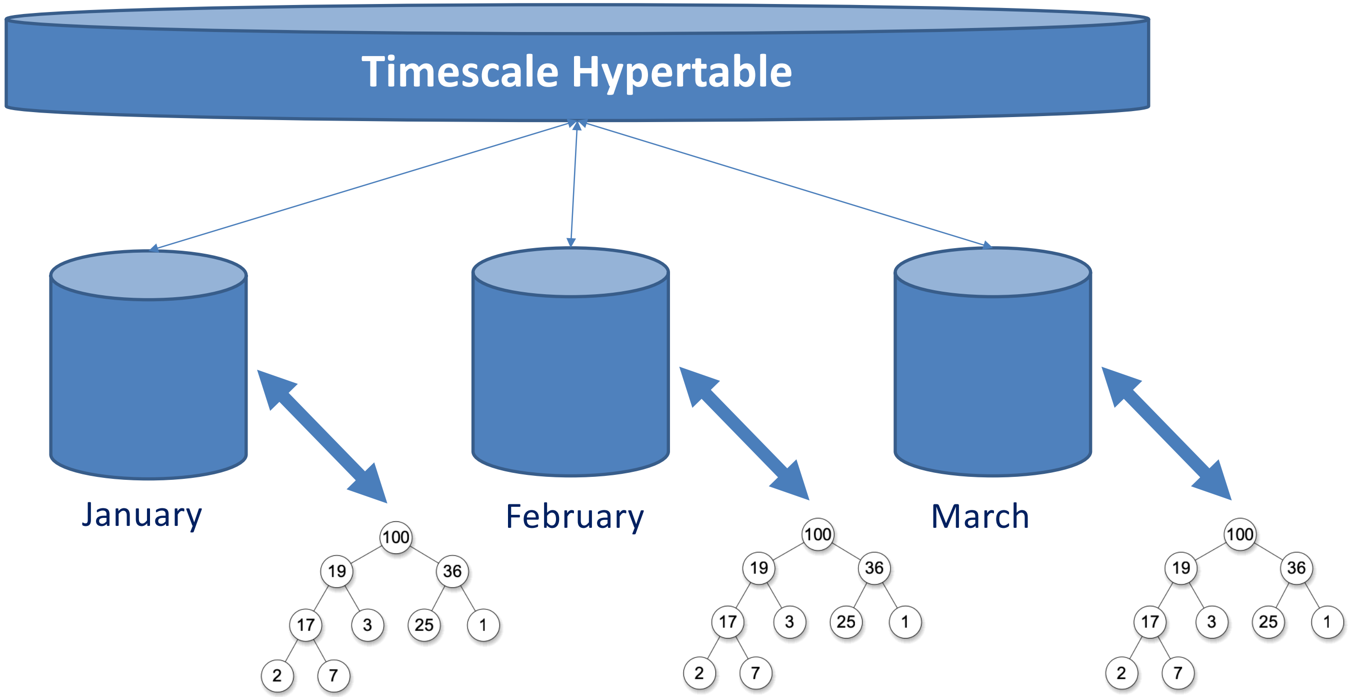
Data partitioning or “Chunking”



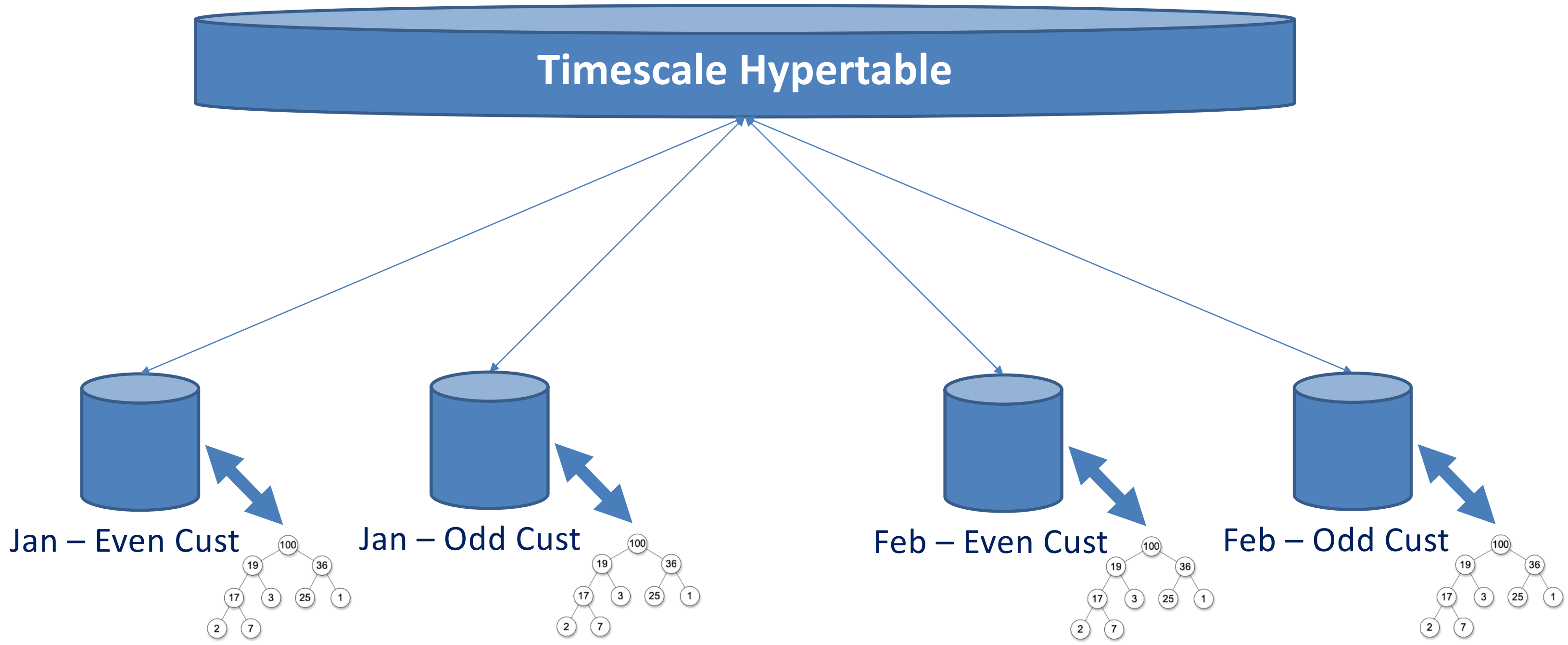
Data partitioning or “Chunking”



Data partitioning or “Chunking”



Data partitioning or “Chunking”



Hypertables in action

- “Depth” of data should not impact performance. If you keep 10 years of data it will just consume storage until someone queries it.
- Lookup performance requires indices to be loaded. Current indices will be in RAM as you will be writing to the time series in the current period.
- Lookups are commonly grouped. E.g. most customers look at their monitoring for the current month. Locality of lookups keeps indices cached.
- Concurrent writes and reads are handled efficiently.

Hypertables in action

Example Benchmark

Voip CDR records (very wide data records)

3 Million records per day

6 months data (i.e. approx. 550 million records)

Test 1

No concurrent data inserts

Query 1 : Get 1st 50 of 1000 records for a client in a given month

=> Data returned in under 2 seconds

Query 2 : Get 2nd 50 records same client and same month

=> Data returned in 15 msec

Hypertables in action

Test 2

Base insert rate of 5,000 records per minute

Query 1 : Get 1st 50 of 1000 records for a client in a given month

=> Data returned in under 2 seconds

Query 2 : Get 2nd 50 records same client and same month

=> Data returned in 15 msec

Test 3

Base insert rate of 50,000 records per minute

Query 1 : Get 1st 50 of 1000 records for a client in a given month

=> Data returned in under 3 seconds

Query 2 : Get 2nd 50 records same client and same month

=> Data returned in 25 msec

Scale out

"Unlimited" data width

Scale-out Option 1

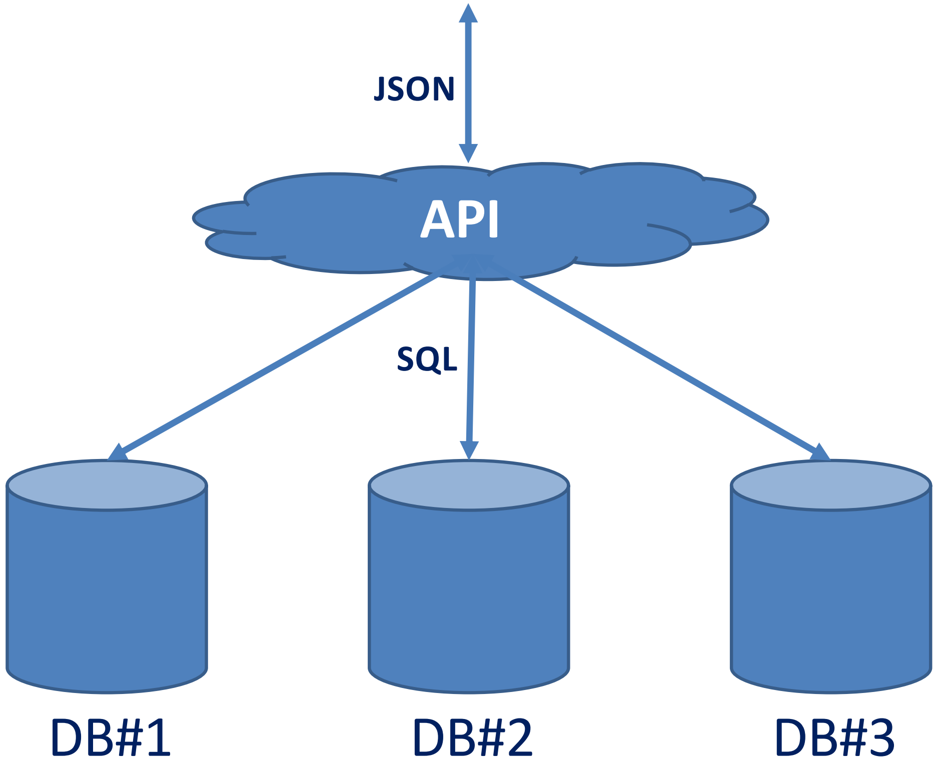
Using TimescaleDB Features

- Clustering in beta
- Uses combination of data nodes and access nodes
- Access nodes route queries to data node

Scale-out Option 2

Simple solution using “home grown sharding”

Scale out via “API Sharding”



Thanks !

Any questions ?

(remember there’s coffee outside ☺)