TIMESCALE

Diana Hsieh

Head of Product
Timescale

TIMESCALE

TIMESCALE

Cockroach LABS

Diana Hsieh

Head of Product
Timescale

TIMESCALE

Sven Klemm

Software Engineering
Timescale


Panel Title
600
500
400
300
200
100
0
10/1
11/1
12/1
1/1
2/1
3/1
4/1
5/1
6/1
7/1
8/1
9/1
signups
Graph
General
Metrics
Axes
Legend
Display
Alert
Time range
Data Source
default
Options
Query Inspector
A
FROM
usage
Time column
"time"
Metric column
none
SELECT
Column: user_id
Aggregate: count
Alias: signups
WHERE
Macro: $__timeFilter
Expr: request_type = 'signup'
GROUP BY
time (24h, none)
Format as
Time series
Edit SQL
Show Help
Generated SQL

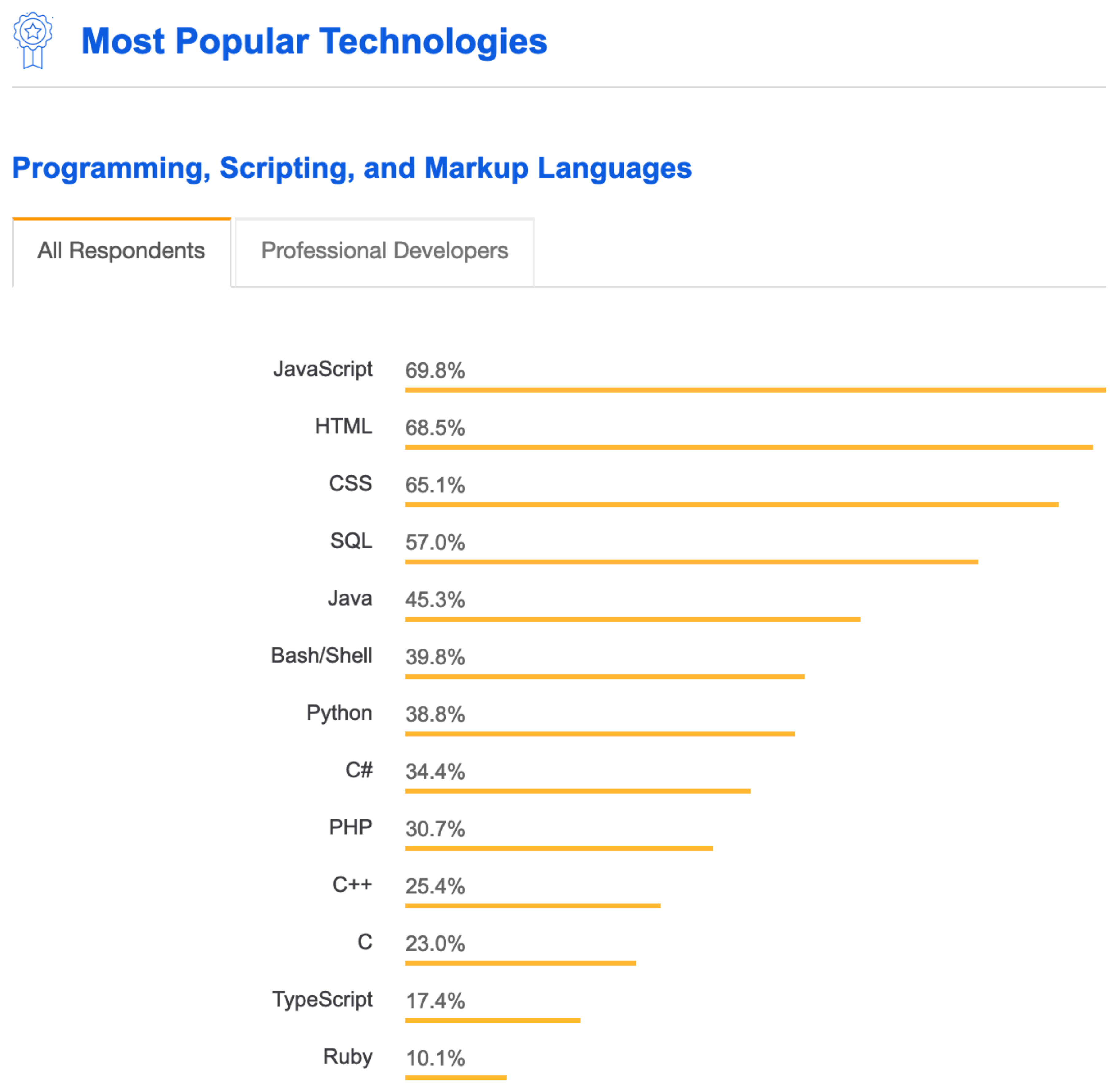

Demo Dashboard
Last 12 hours
Host
grafana.timescaledb.rocks
Memory
1.9 GiB
1.4 GiB
954 MiB
477 MiB
0 B
11:00
12:00
13:00
14:00
15:00
16:00
17:00
18:00
19:00
20:00
21:00
22:00
used
buffers
cache
free
Queries to
default
Add Query
Query Inspector
?
A
FROM
Time column
time
Metric column
none
SELECT
WHERE
GROUP BY
cpu
disk
diskio
docker
docker_container_blkio
docker_container_cpu
docker_container_mem
docker_container_net
docker_container_status
kernel
fers
e
_timeFilter

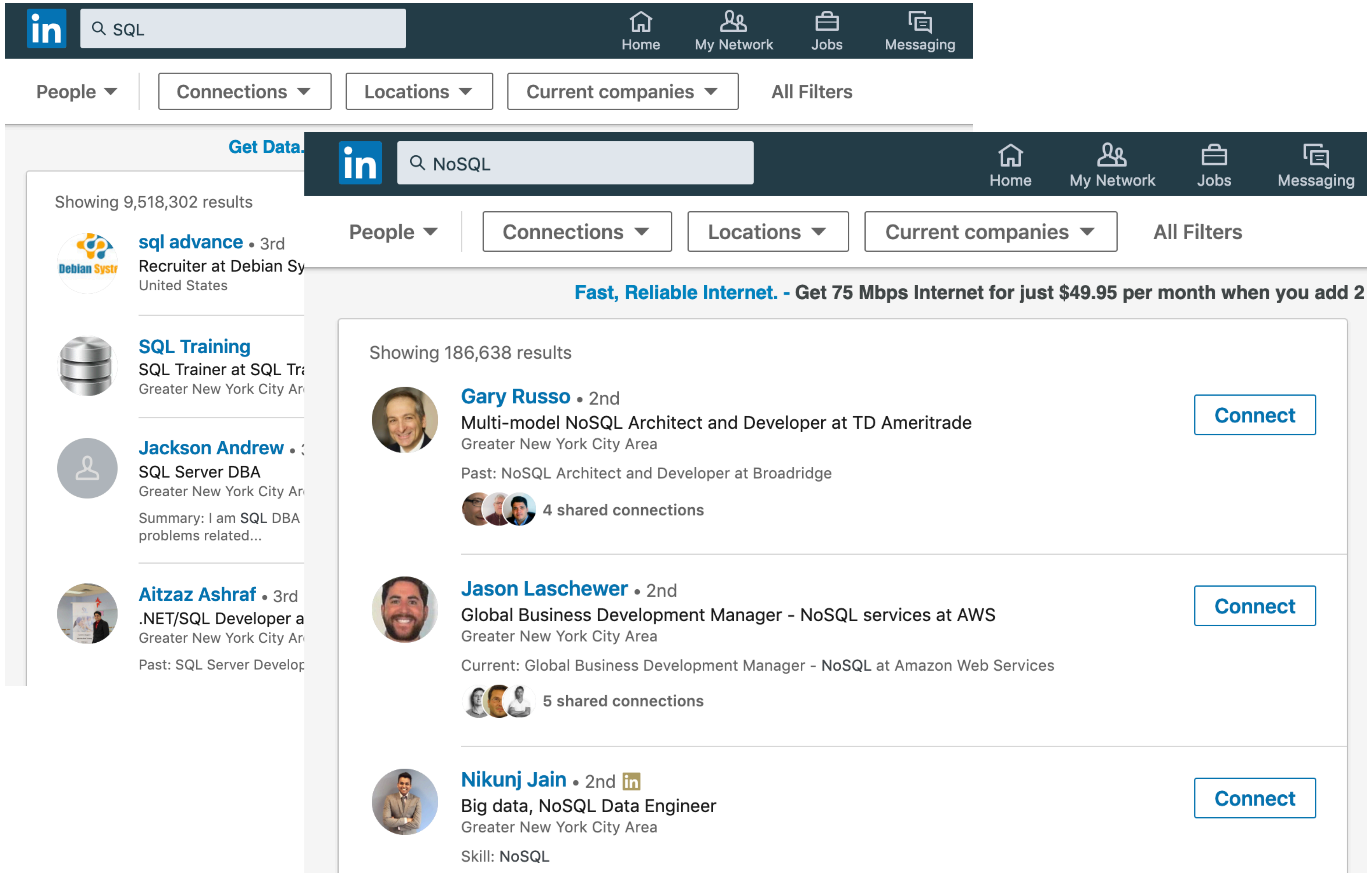

Demo Dashboard
Last 12 hours
Host
grafana.timescaledb.rocks
Memory
1.9 GiB
1.4 GiB
954 MiB
477 MiB
0 B
11:00
12:00
13:00
14:00
15:00
16:00
17:00
18:00
19:00
20:00
21:00
22:00
used
buffers
cache
free
Queries to
default
Add Query
Query Inspector
?
A
FROM
mem
Time column
time
Metric column
none
SELECT
Column: used
Alias: used
Column: buffered
Alias: buffers
Column: cached
Alias: cache
Column: free
Alias: free
WHERE
Expr: host = '$host'
Macro: $__tim
GROUP BY
Format as
Time series
Edit SQL
Show Help
Aggregate Functions
Ordered-Set Aggregate Functions
Window Functions
Alias
Column
Average
Count
Maximum
Minimum
Sum
Standard deviation
Variance
First


Demo Dashboard
Last 12 hours
Host
grafana.timescaledb.rocks
Memory
1.9 GiB
1.4 GiB
954 MiB
477 MiB
0 B
11:00
12:00
13:00
14:00
15:00
16:00
17:00
18:00
19:00
20:00
21:00
22:00
used
buffers
cache
free
Queries to
default
Add Query
Query Inspector
?
A
SELECT
time AS "time",
used AS "used",
buffered AS "buffers",
cached AS "cache",
free AS "free"
FROM mem
WHERE
host = '$host' AND
$__timeFilter(time)
Format as
Time series
Query Builder
Show Help

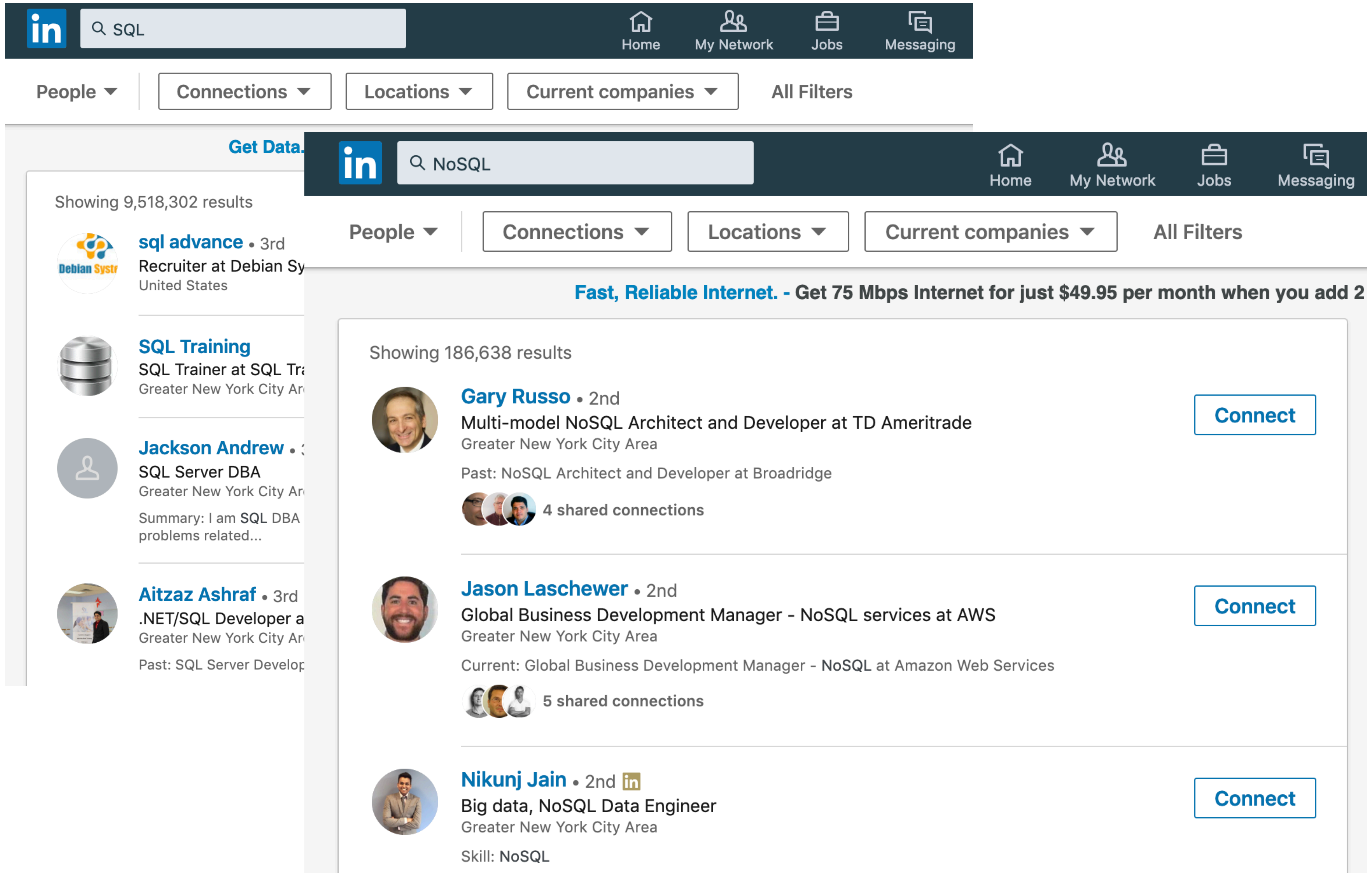

MTA Demo
Last 6 hours
Refresh every 5s
MTA
Off Route Buses
NEW YORK
JERSEY CITY
Leaflet | © OpenStreetMap © CartoDB

SQL is a well-known querying language

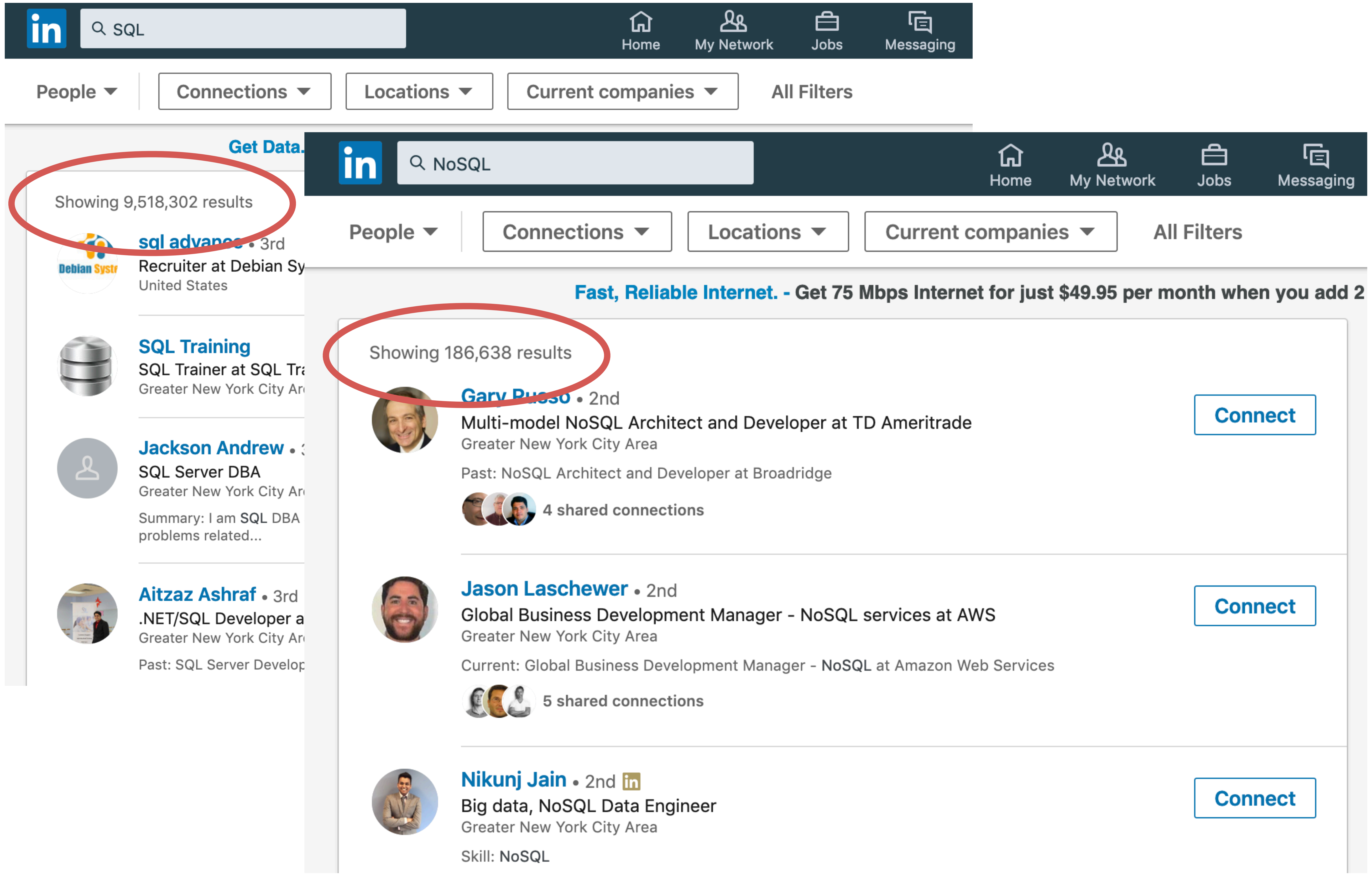




SQL is a well-known querying language



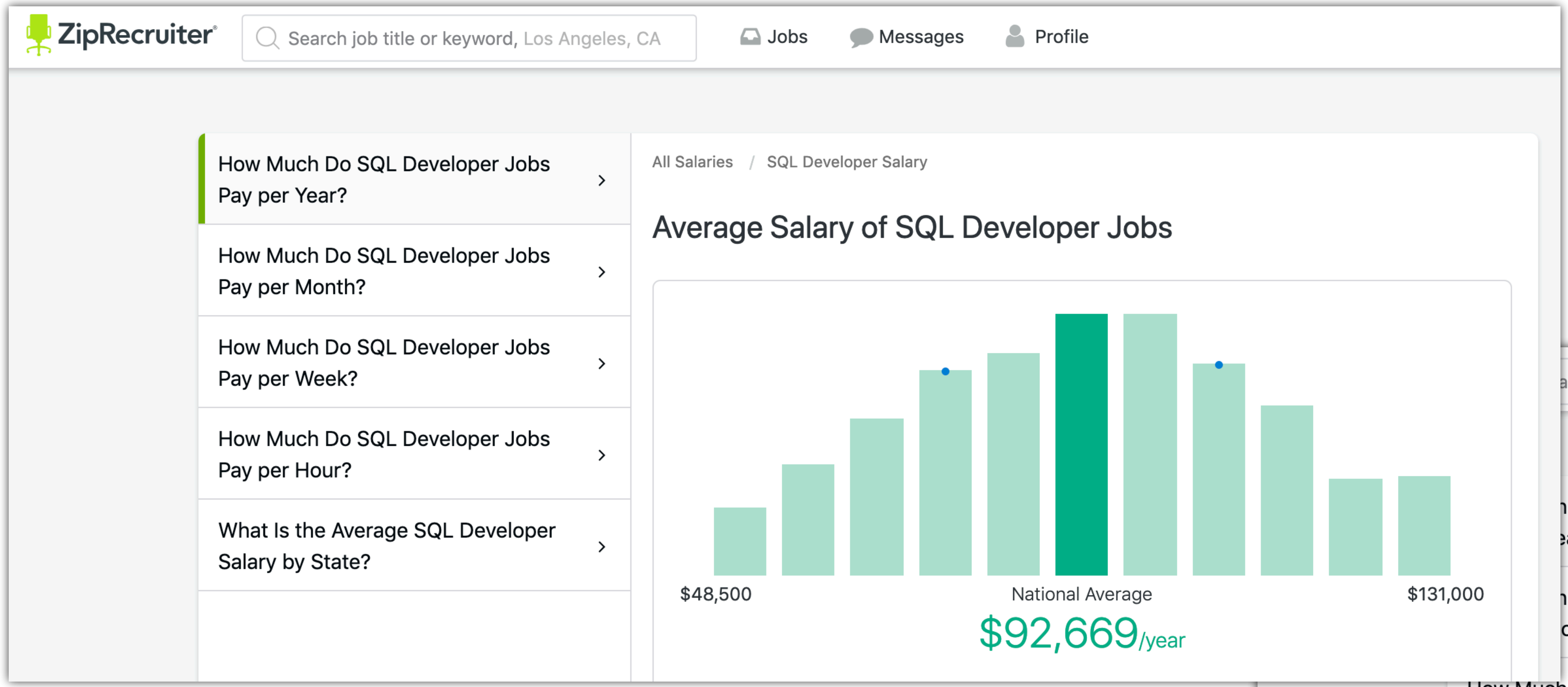
SQL is a well-known querying language



SQL is a well-known querying language

SQL is a well-known querying language

SQL is a well-known querying language



Data must be accessible to provide value

New database, new integrations

Reinforced data silos due to lack of inter-operability between systems

New database, new language

Increased training costs and talent gap to query disparate silos

Brittle architecture

Huge operational cost to manage multiple database solutions

De-centralized infrastructure

Hard to analyze data that spans across multiple silos

COMCAST

“ Initially my colleagues were skeptical when I suggested storing metrics for our ***120 petabyte data center*** *in a relational database, but after replacing the prior NoSQL database with TimescaleDB* ***we couldn’t be more happy*** *with the performance.*

Because TimescaleDB is an extension of PostgreSQL, we’re starting to expand the scope of the metrics storage to ***power executive dashboards*** *and advanced analytical functions that our prior NoSQL solution couldn’t support.”*

- Chris Holcombe, Production Dev Engineer

But.. SQL doesn’t scale…

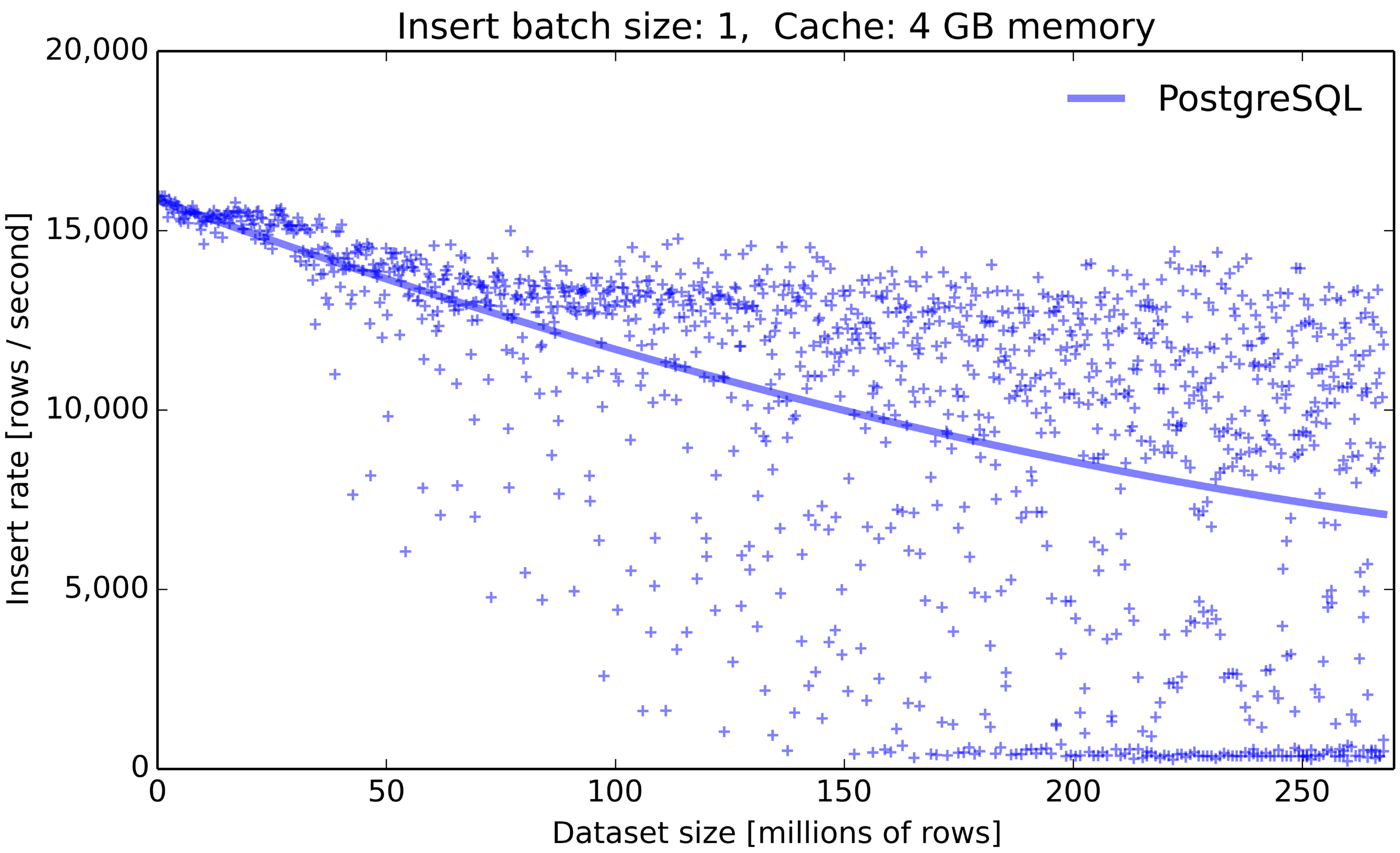
Traditional SQL databases were built for transactional workloads

	State snapshot	Time-series
Financial Data	Account balance	Transaction history
DevOps	Capacity utilized	Usage over time
IoT	# of online devices	Devices over time
SaaS Applications	Total users	Logins over time

Time-series data requires a different kind of database

- Fast ingest of weakly ordered data
- Large storage requirements
- Time-series specific analytics
- Time-series specific data management

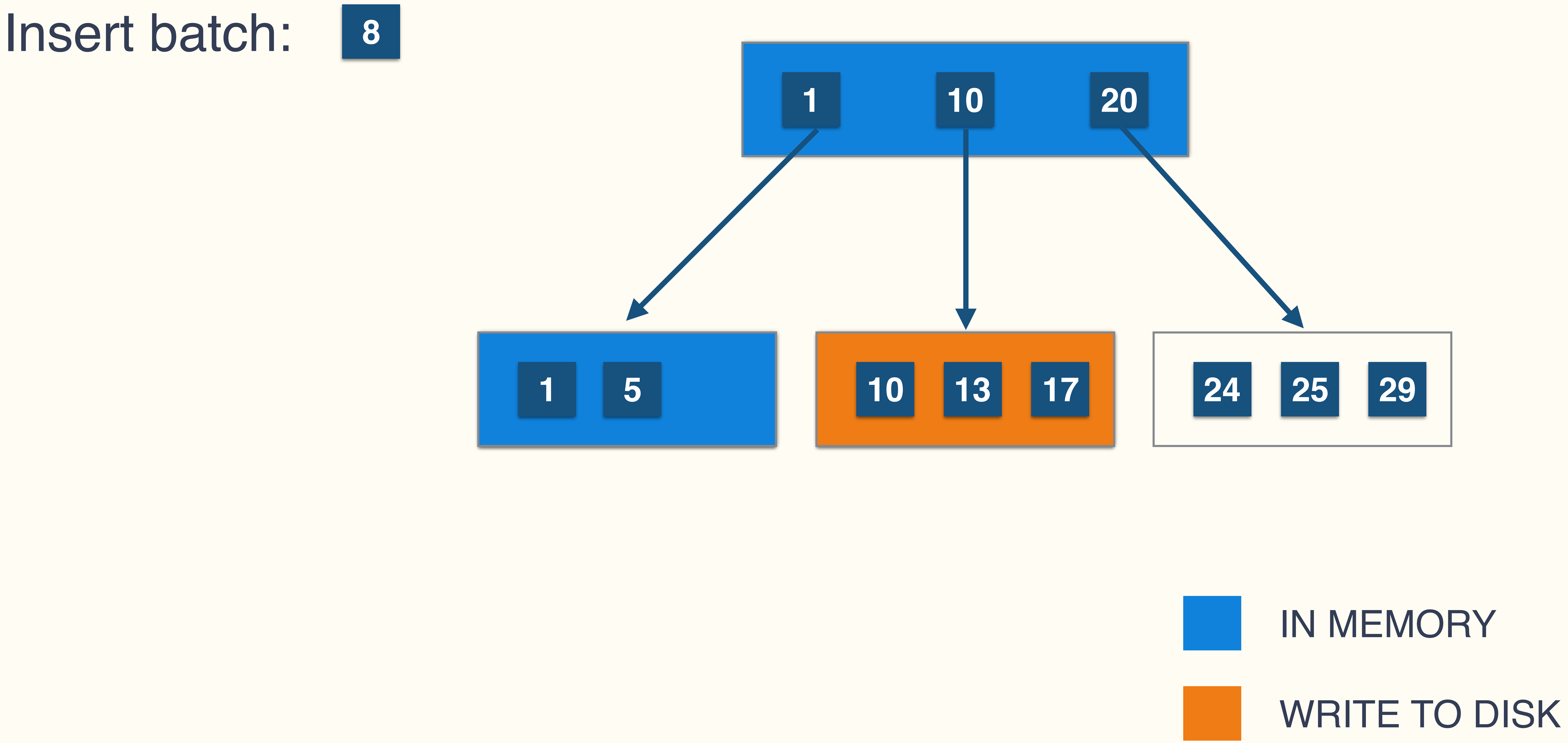
PostgreSQL is hard to scale



Postgres 9.6.2 on Azure standard DS4 v2 (8 cores), SSD (premium LRS storage)

Each row has 12 columns (1 timestamp, indexed 1 host ID, 10 metrics)

Indexes need to be updated on inserts

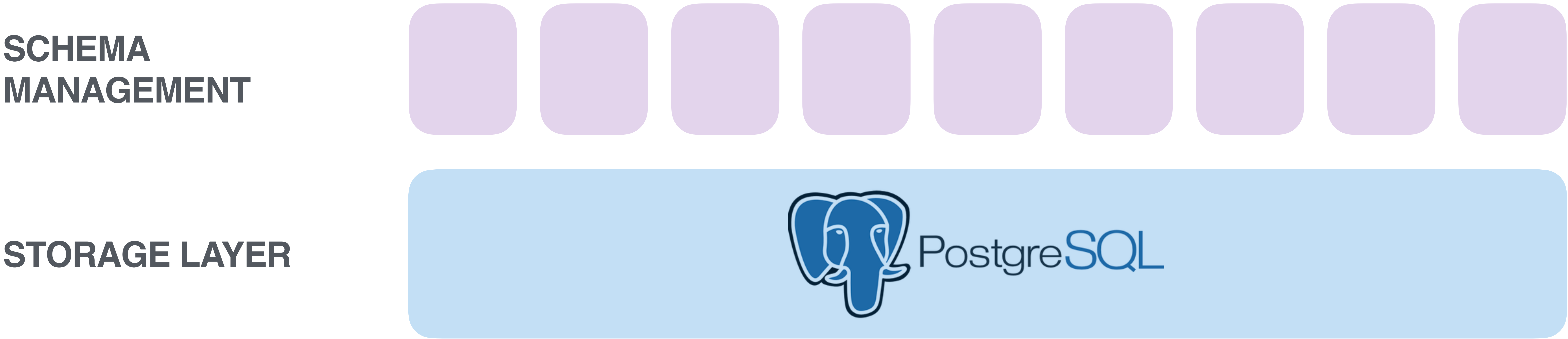


Making SQL for time-series possible

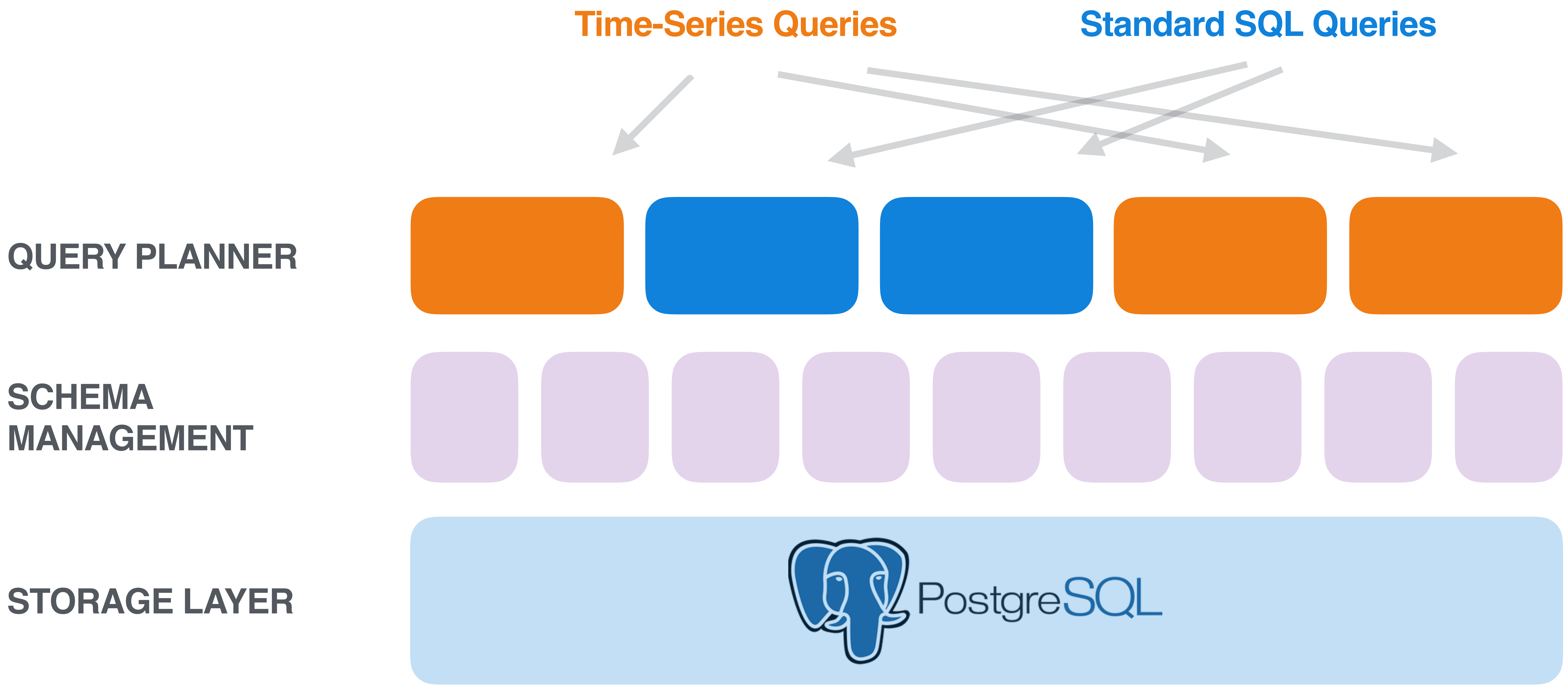
STORAGE LAYER

PostgreSQL

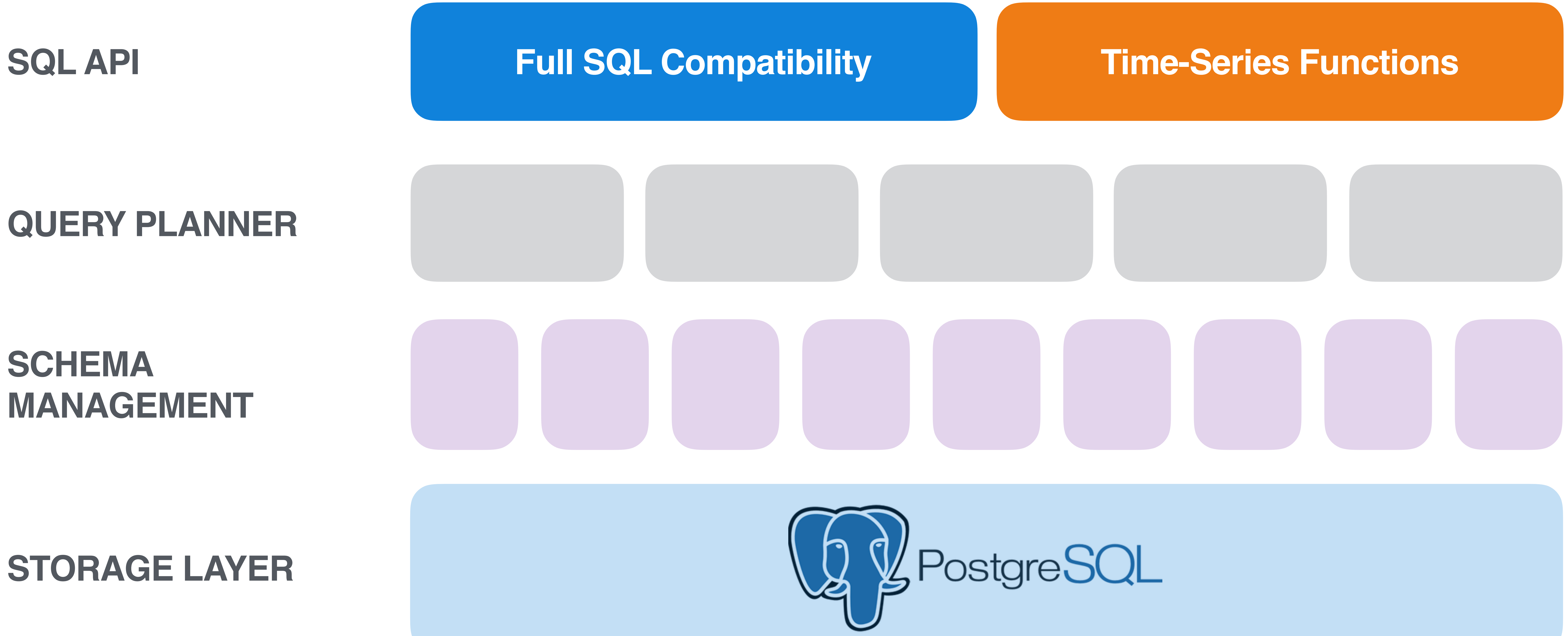
Making SQL for time-series possible



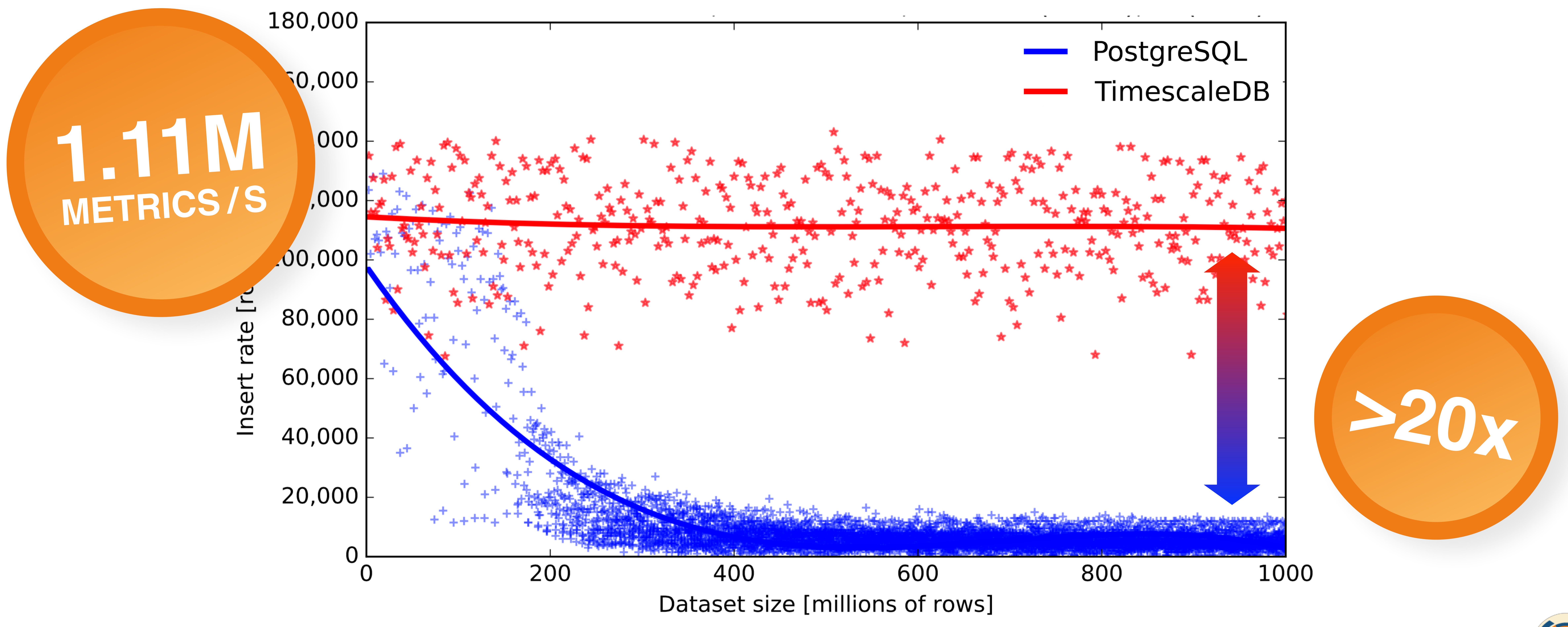
Making SQL for time-series possible



Making SQL for time-series possible



TimescaleDB vs PostgreSQL



TimescaleDB 0.5, Postgres 9.6.2, Azure standard DS4 v2 (8 cores), SSD (LRS storage)
Each row has 12 columns (1 timestamp, indexed 1 host ID, 10 metrics)

How would I go about modeling this?

Our MTA demo

Rider Metrics

Rider Metadata

MTA time-series

PostGIS metadata

Riders Schema

Rider Metadata

```
mta=> \d riders
              Table "public.riders"
   Column    |  Type   | Collation | Nullable | Default
-------------+---------+-----------+----------+---------
 rider_id    | text    |           | not null |
 ny_resident | boolean |           |          |
Indexes:
    "riders_pkey" PRIMARY KEY, btree (rider_id)
```

Rider Metrics

```
                       Table "public.riders_metrics"
  Column   |           Type           | Collation | Nullable | Default
-----------+--------------------------+-----------+----------+---------
 time      | timestamp with time zone |           | not null |
 route_id  | text                     |           | not null |
 rider_id  | integer                  |           | not null |
```

MTA Schema

MTA time-series

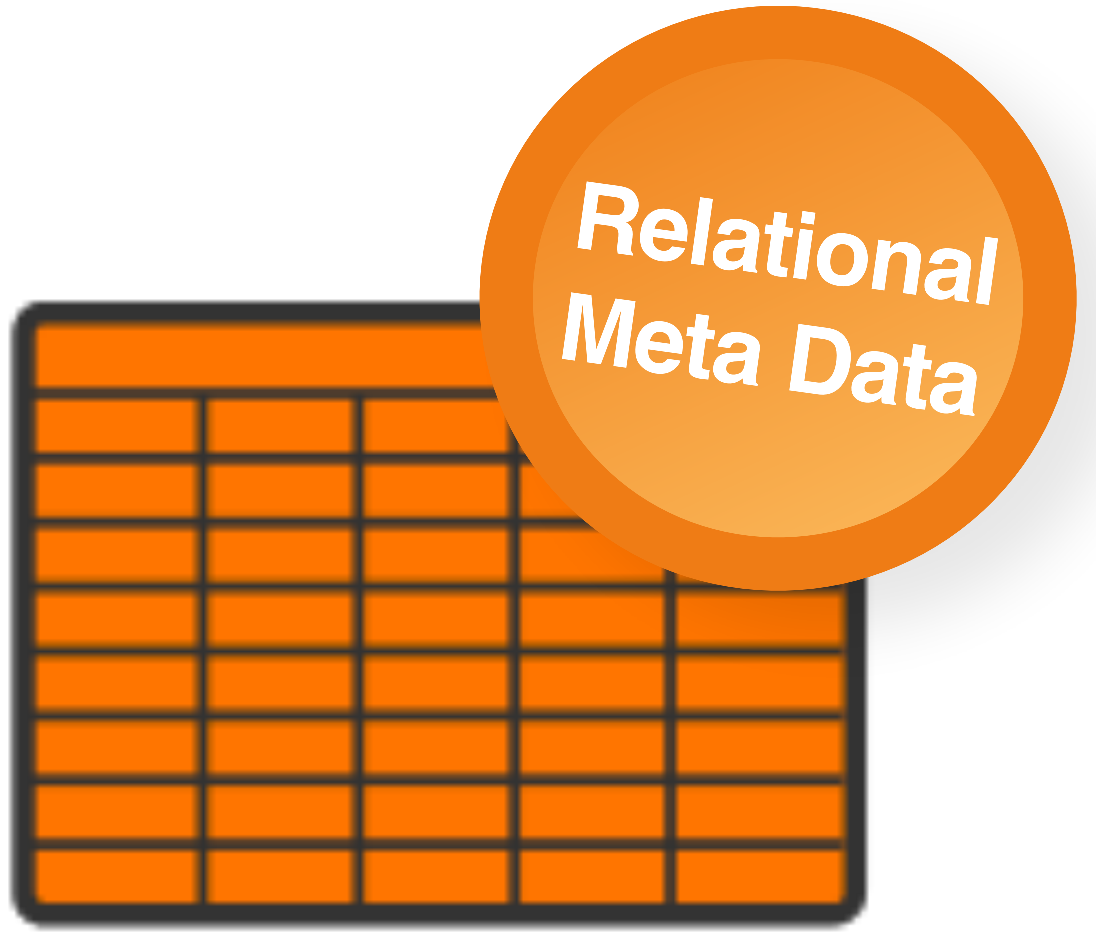
PostGIS metadata

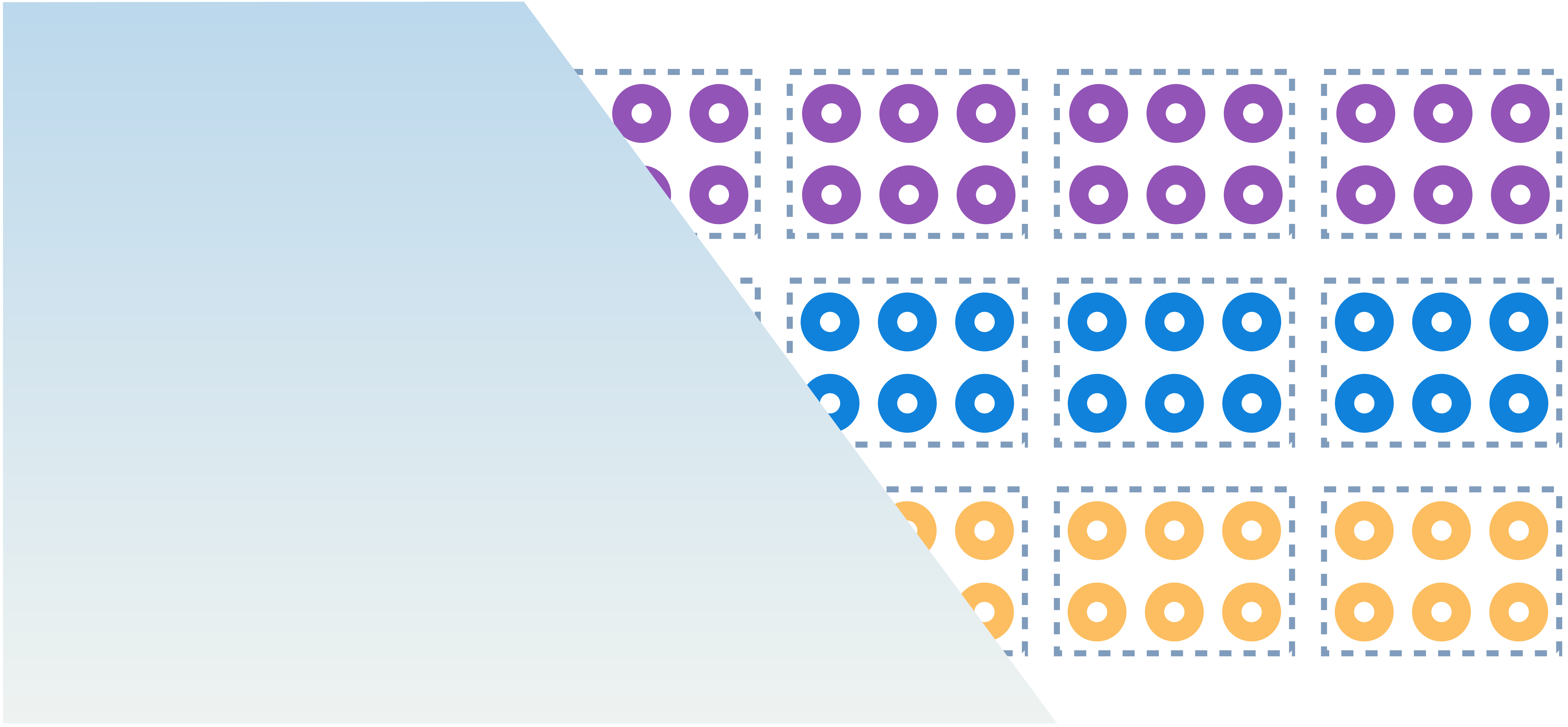
```
mta=> \d route_geofences
                  Table "public.route_geofences"
  Column  |          Type          | Collation | Nullable | Default
----------+------------------------+-----------+----------+---------
 route_id | character varying(254) |           |          |
 geom     | geometry               |           |          |
Indexes:
    "route_geofences_geom_idx" gist (geom)
    "route_geofences_route_id_idx" btree (route_id)
```

```
                          Table "public.mta"
  Column  |           Type           | Collation | Nullable | Default
----------+--------------------------+-----------+----------+---------
 vid      | text                     |           |          |
 time     | timestamp with time zone |           | not null |
 route_id | text                     |           |          |
 bearing  | numeric                  |           |          |
 geom     | geometry(Point,4326)     |           |          |
Indexes:
    "idx_mta_geom" gist (geom)
    "idx_mta_route_id" btree (route_id)
    "mta_time_idx" btree ("time" DESC)
Triggers:
    ts_insert_blocker BEFORE INSERT ON mta FOR EACH ROW EXECUTE PROCEDURE _timescaledb_internal.insert_blocker()
Number of child tables: 31 (Use \d+ to list them.)
```

Separate meta data from time series

Speed up queries and inserts by storing values that describe time series metrics in relational tables

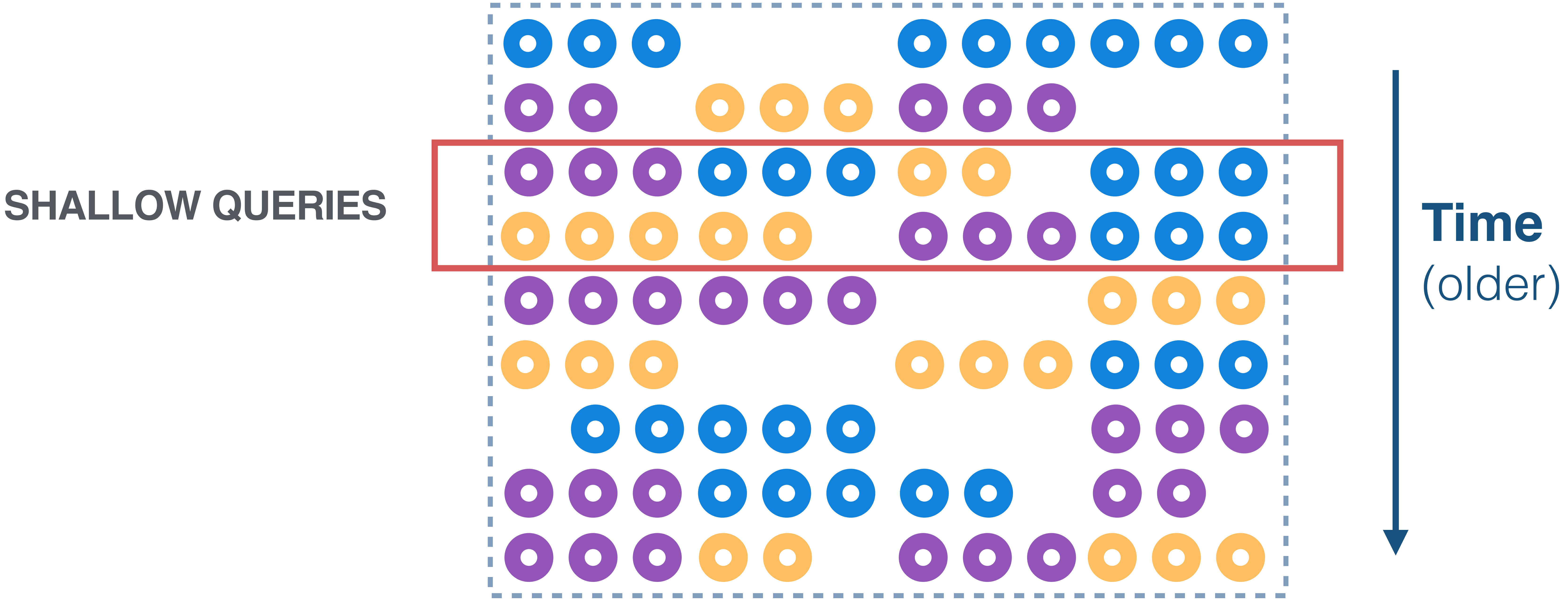




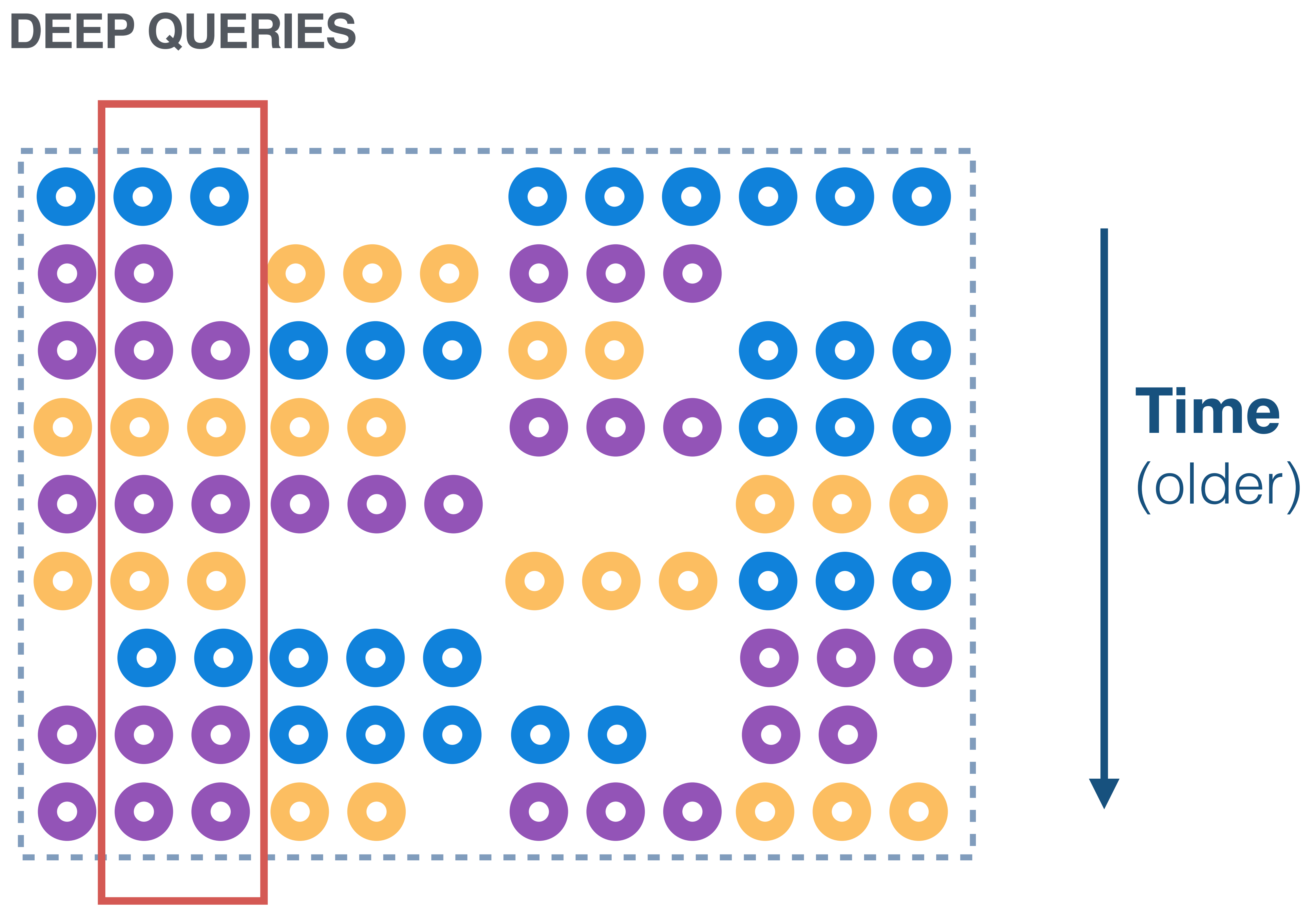
Store metrics more efficiently with metadata tables

```
webinar=# select * from hardware_metadata limit 5;
 server_id | location |       start_date       |                                        description
-----------+----------+------------------------+------------------------------------------------------------------------------------------------
         1 | {DE}     | 2015-01-22 00:00:00+00 | server1: some random and long text about this server that no one needs, except when they need it
         2 | {FR}     | 2010-08-08 00:00:00+00 | server2: some random and long text about this server that no one needs, except when they need it
         3 | {CN}     | 2012-05-14 00:00:00+00 | server3: some random and long text about this server that no one needs, except when they need it
         4 | {CN}     | 2010-10-14 00:00:00+00 | server4: some random and long text about this server that no one needs, except when they need it
         5 | {FR}     | 2016-10-15 00:00:00+00 | server5: some random and long text about this server that no one needs, except when they need it
(5 rows)
```

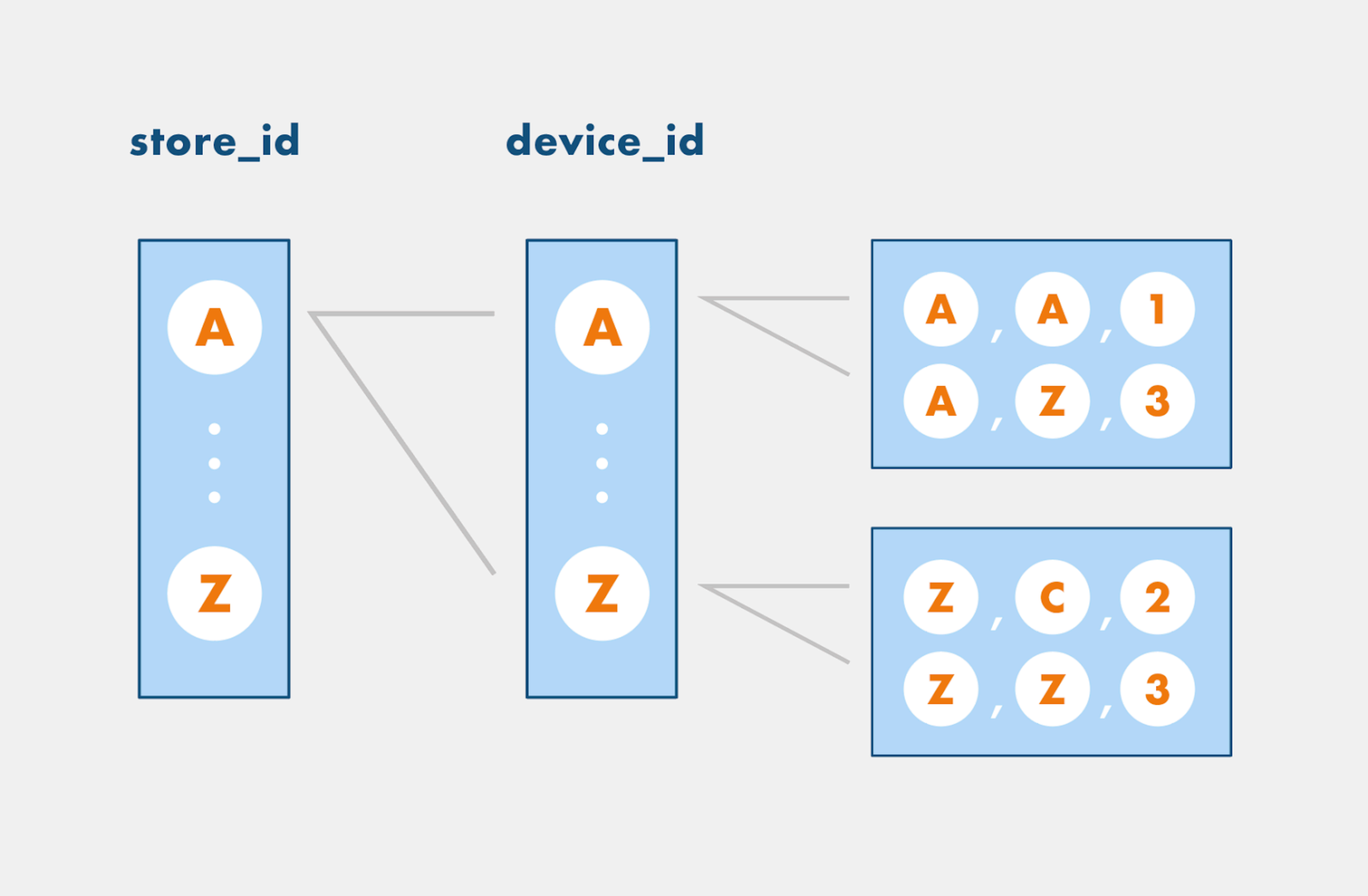
Optimize for your query patterns



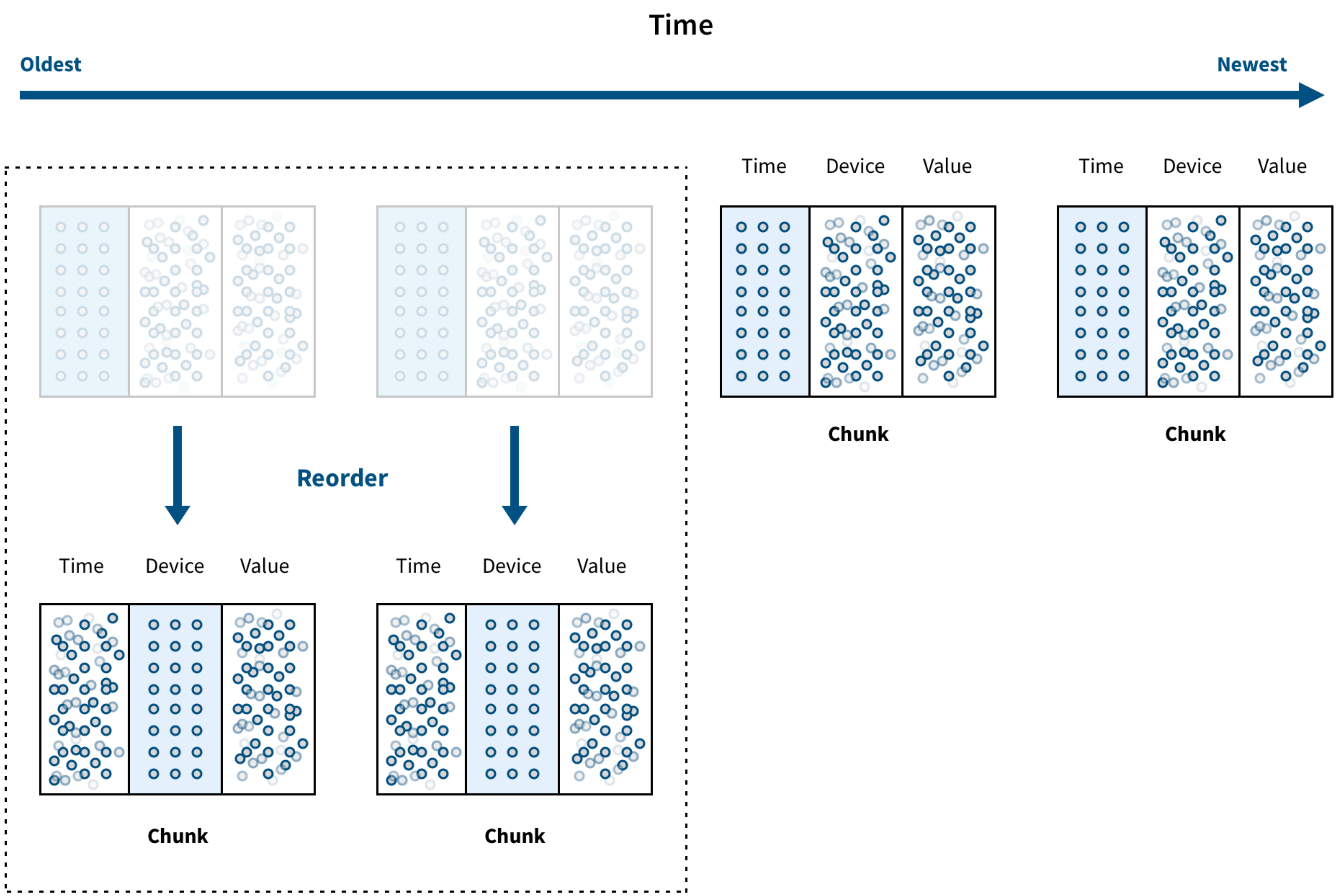
Optimize for your query patterns



Leverage indexes…

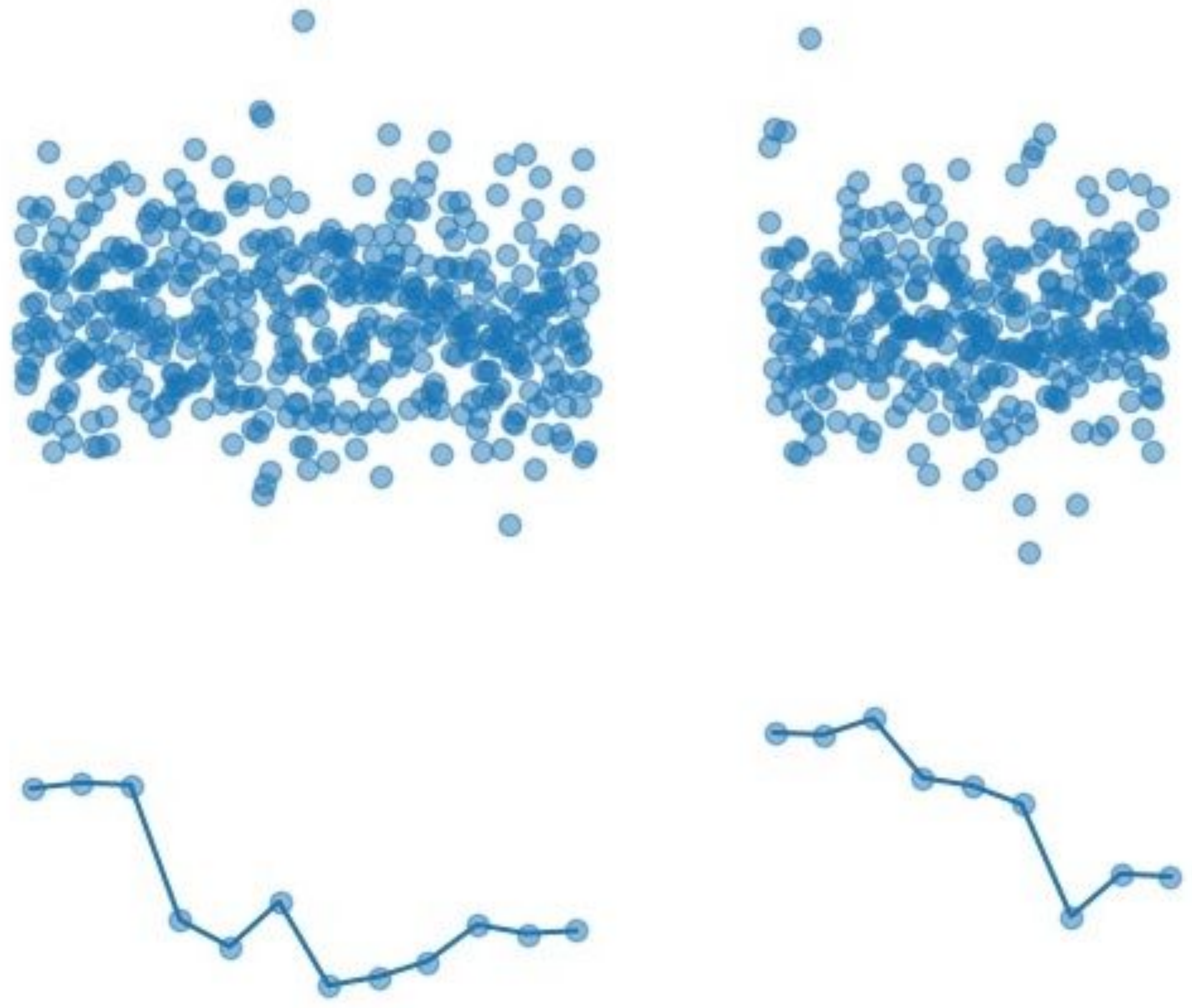


…or optimize how data is written on disk



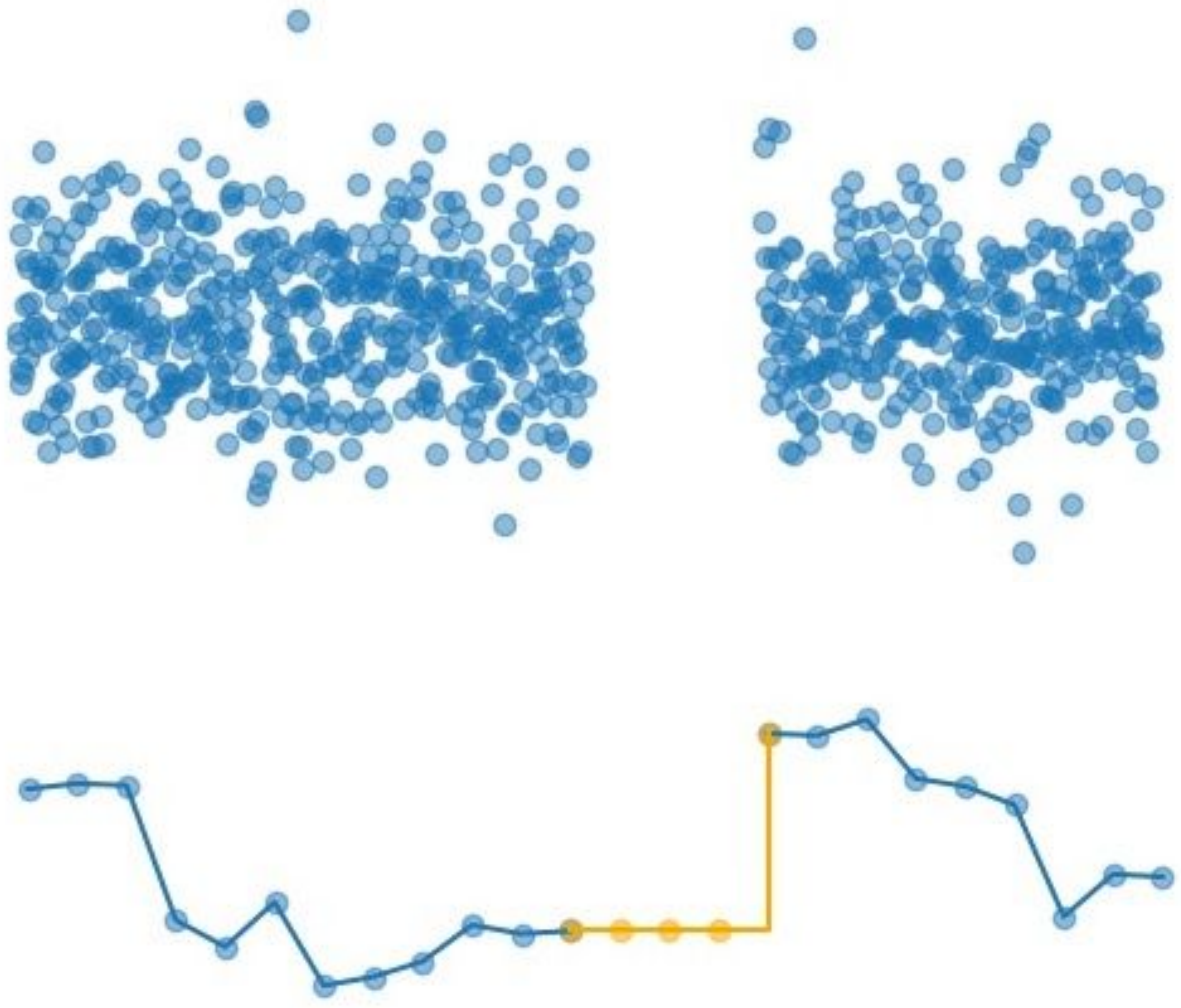
Leverage TimescaleDB functions

```
SELECT
   time_bucket('1 minute', time) as minute,
   avg(value)
FROM observations
GROUP BY minute
ORDER BY minute;
```



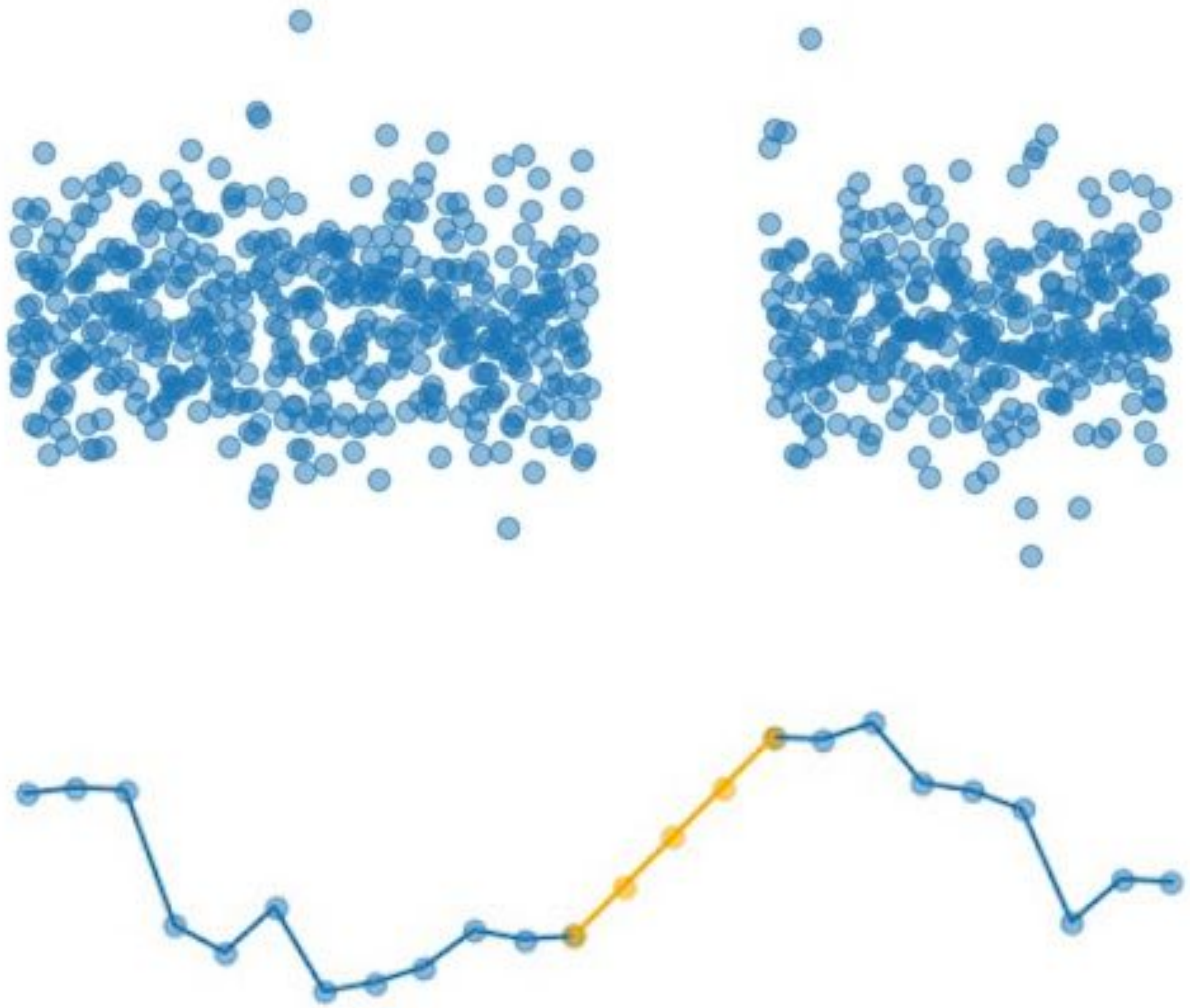
Leverage TimescaleDB functions

```
SELECT
  time_bucket_gapfill(
    ‘1 minute’, time,
    start => '2019-01-21 9:00',
    finish => '2019-01-21 17:00’) as minute
  locf(avg(value))
FROM observations
GROUP BY minute;
```



Leverage TimescaleDB functions

```
SELECT
  time_bucket_gapfill(
    ‘1 minute’, time,
    start => '2019-01-21 9:00',
    finish => '2019-01-21 17:00’) as minute
  interpolate(avg(value))
FROM observations
GROUP BY minute;
```



Open Source (Apache 2.0)

- github.com/timescale/timescaledb

Join the Community

- slack.timescale.com