Using PostgreSQL, Prometheus & Grafana for Storing, Analyzing and Visualizing Metrics

Erik Nordström, PhD
Core Database Engineer

hello@timescale.com · **github.com/timescale**

Why PostgreSQL?

- Reliable and familiar (ACID, Tooling)
- SQL: powerful query language
- JOINs: combine time-series with other data
- Simplify your stack: avoid data silos

TimescaleDB:
PostgreSQL for **time-series data**

Common Complaints

👎

- Hard or impossible to scale
- Need to define schema
- SQL too complex or has poor support for querying time-series
- Vacuuming on DELETE
- **No Grafana support**

TimescaleDB + Prometheus

- Scales for time-series workloads
- Automatic scheme creation
- Advanced analytics with full SQL support and time-oriented features
- No vacuuming with drop_chunks()
- Grafana support via Prometheus or PostgreSQL data sources (since v4.6)

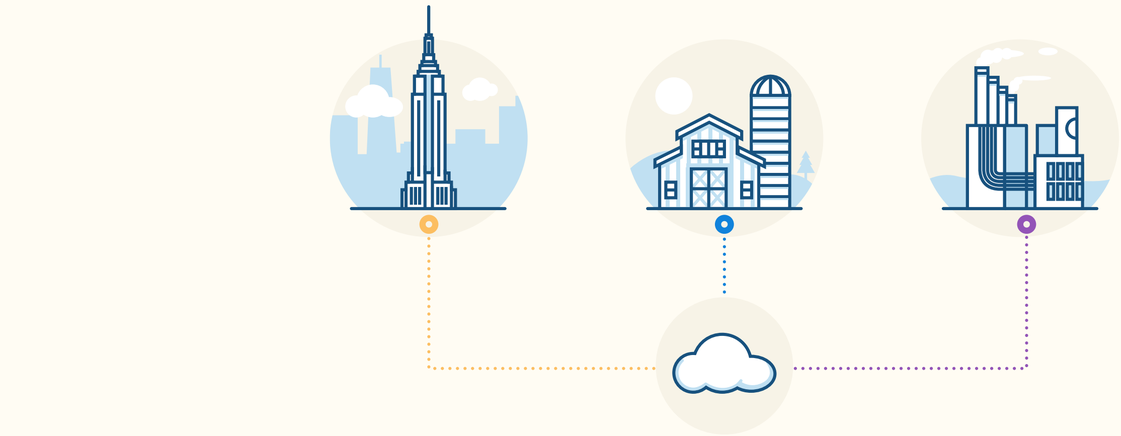
How it works

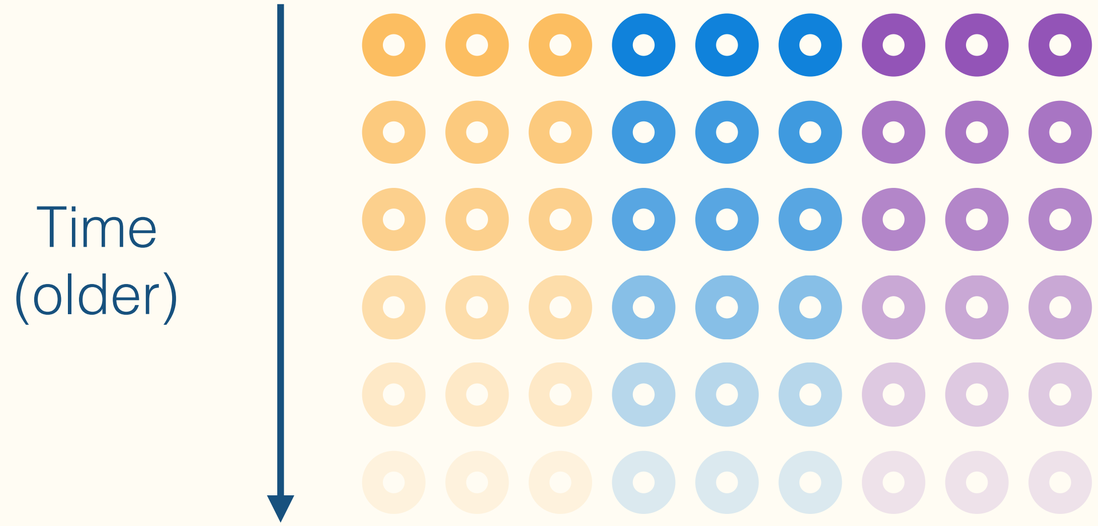
Collecting metrics with Prometheus

Remote storage API

TimescaleDB / PostgreSQL

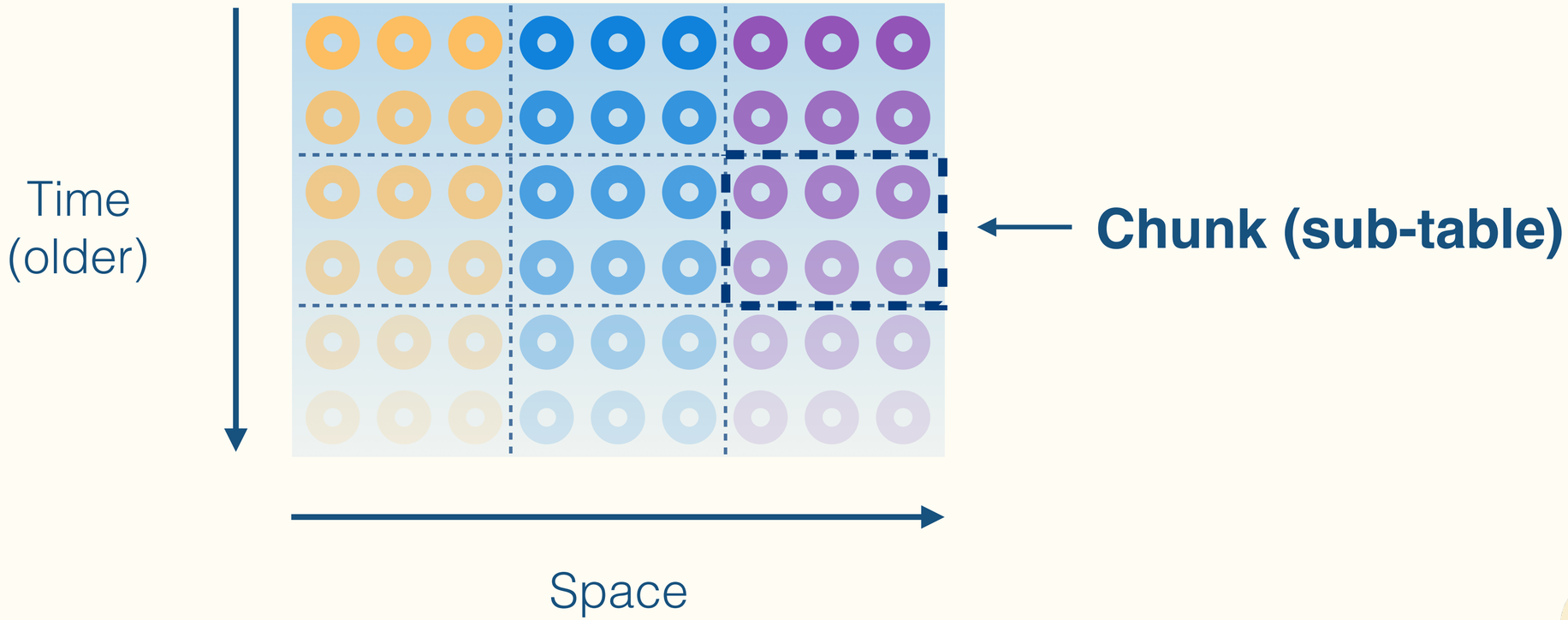
- Adapter
- pg_prometheus




Time
(older)

Time-space partitioning

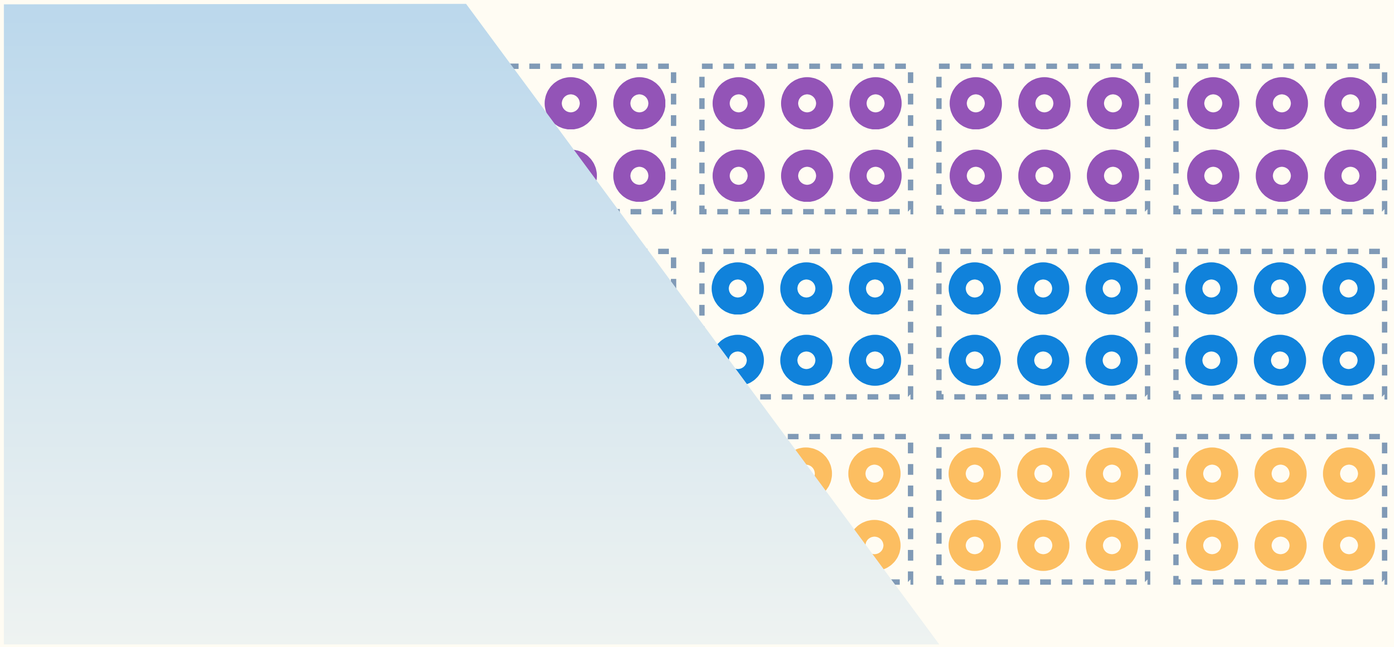
(for both scaling up & out)



The Hypertable Abstraction

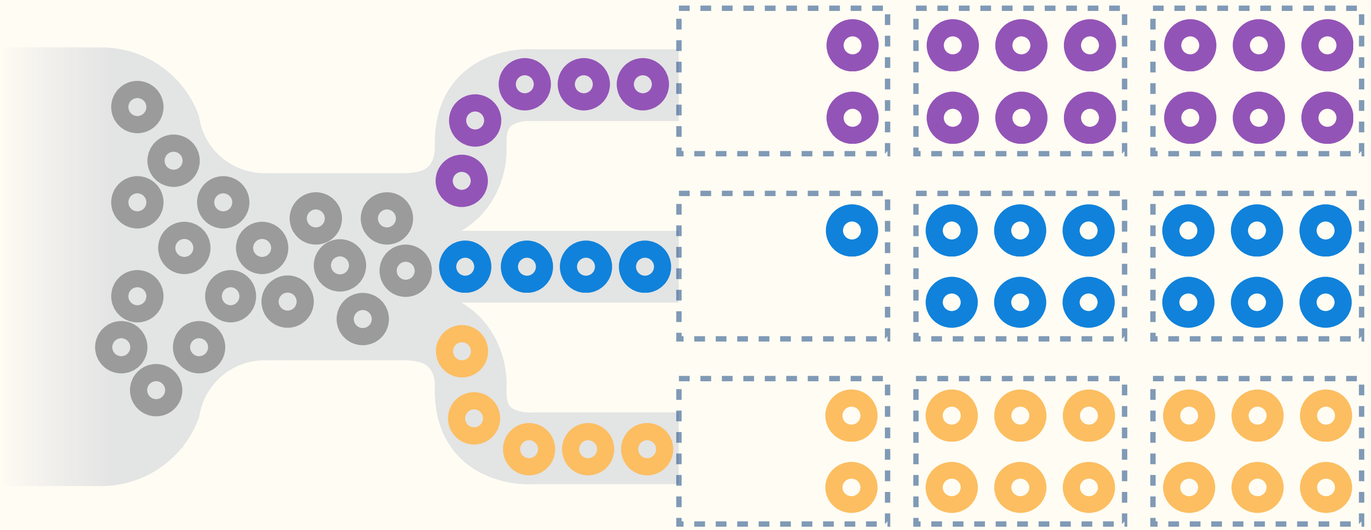
Hypertable

- Triggers
- Constraints
- Indexes
- UPSERTs
- Table mgmt



Chunks

Automatic Space-time Partitioning



Easy to Get Started

```
CREATE TABLE conditions (
      time timestamptz,
      temp float,
      humidity float,
      device text
);

SELECT create_hypertable('conditions', 'time', 'device', 4,
chunk_time_interval => interval '1 week');

INSERT INTO conditions VALUES ('2017-10-03 10:23:54+01', 73.4,
40.7, 'sensor3');


SELECT * FROM conditions;

           time            | temp | humidity | device
---------------------------+------+----------+---------
 2017-10-03 11:23:54+02 | 73.4 |     40.7 | sensor3
```

Repartitioning is Simple

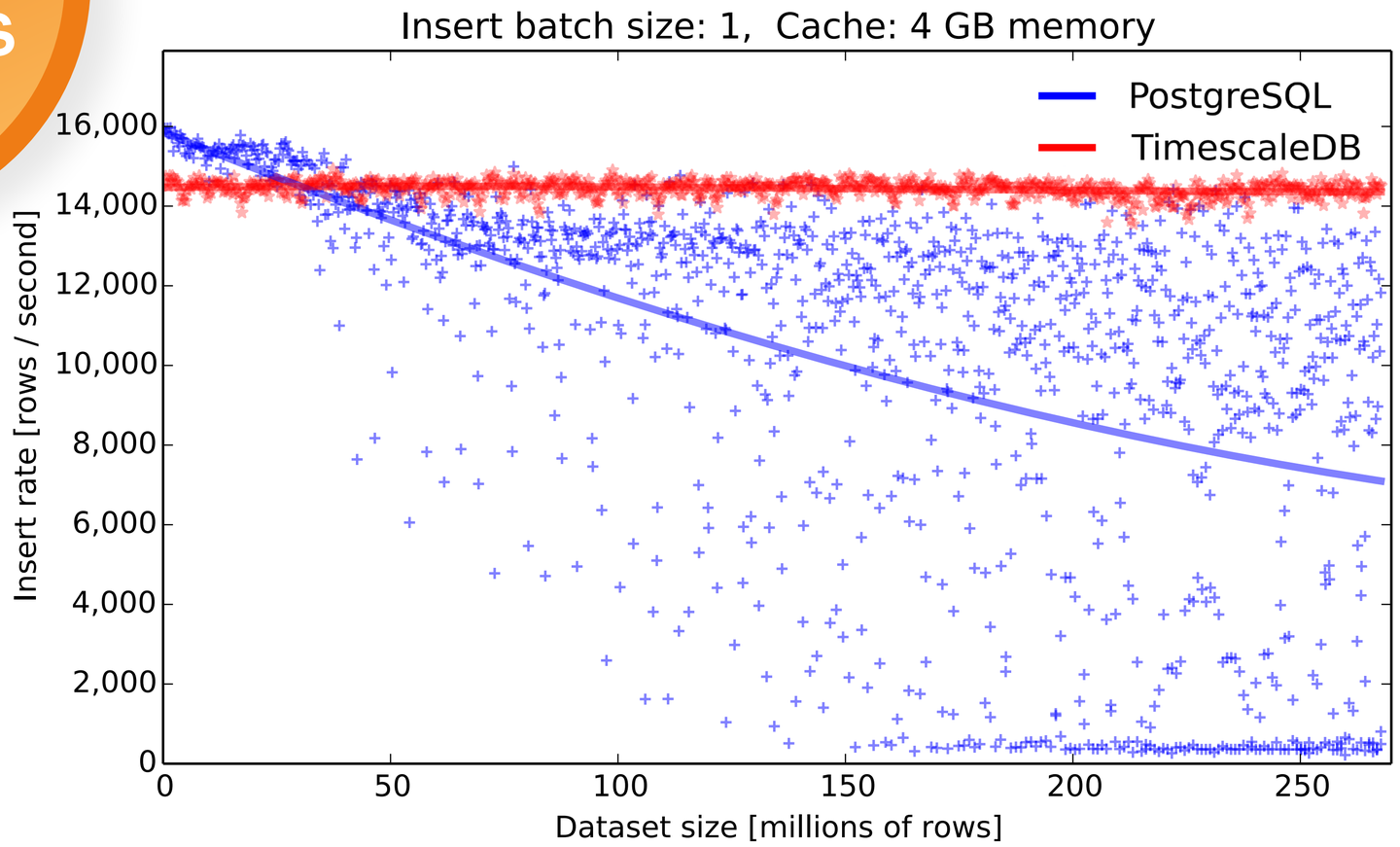
```
— Set new chunk time interval
SELECT set_chunk_time_interval('conditions', interval '24 hours’);


— Set new number of space partitions
SELECT set_number_partitions('conditions', 6);
```

PG10 requires a lot of **manual work**

```
CREATE TABLE conditions (
    time timestamptz,
    temp float,
    humidity float,
    device text
);

CREATE TABLE conditions_p1 PARTITION OF conditions
      FOR VALUES FROM (MINVALUE) TO ('g')
      PARTITION BY RANGE (time);
CREATE TABLE conditions_p2 PARTITION OF conditions
      FOR VALUES FROM ('g') TO ('n')
      PARTITION BY RANGE (time);
CREATE TABLE conditions_p3 PARTITION OF conditions
      FOR VALUES FROM ('n') TO ('t')
      PARTITION BY RANGE (time);
CREATE TABLE conditions_p4 PARTITION OF conditions
      FOR VALUES FROM ('t') TO (MAXVALUE)
      PARTITION BY RANGE (time);
-- Create time partitions for the first week in each device partition
CREATE TABLE conditions_p1_y2017m10w01 PARTITION OF conditions_p1
      FOR VALUES FROM ('2017-10-01') TO ('2017-10-07');
CREATE TABLE conditions_p2_y2017m10w01 PARTITION OF conditions_p2
      FOR VALUES FROM ('2017-10-01') TO ('2017-10-07');
CREATE TABLE conditions_p3_y2017m10w01 PARTITION OF conditions_p3
      FOR VALUES FROM ('2017-10-01') TO ('2017-10-07');
CREATE TABLE conditions_p4_y2017m10w01 PARTITION OF conditions_p4
      FOR VALUES FROM ('2017-10-01') TO (‘2017-10-07');

-- Create time-device index on each leaf partition
CREATE INDEX ON conditions_p1_y2017m10w01 (time);
CREATE INDEX ON conditions_p2_y2017m10w01 (time);
CREATE INDEX ON conditions_p3_y2017m10w01 (time);
CREATE INDEX ON conditions_p4_y2017m10w01 (time);

INSERT INTO conditions VALUES ('2017-10-03 10:23:54+01', 73.4, 40.7,
‘sensor3');
```

INSERT performance

144K METRICS / S



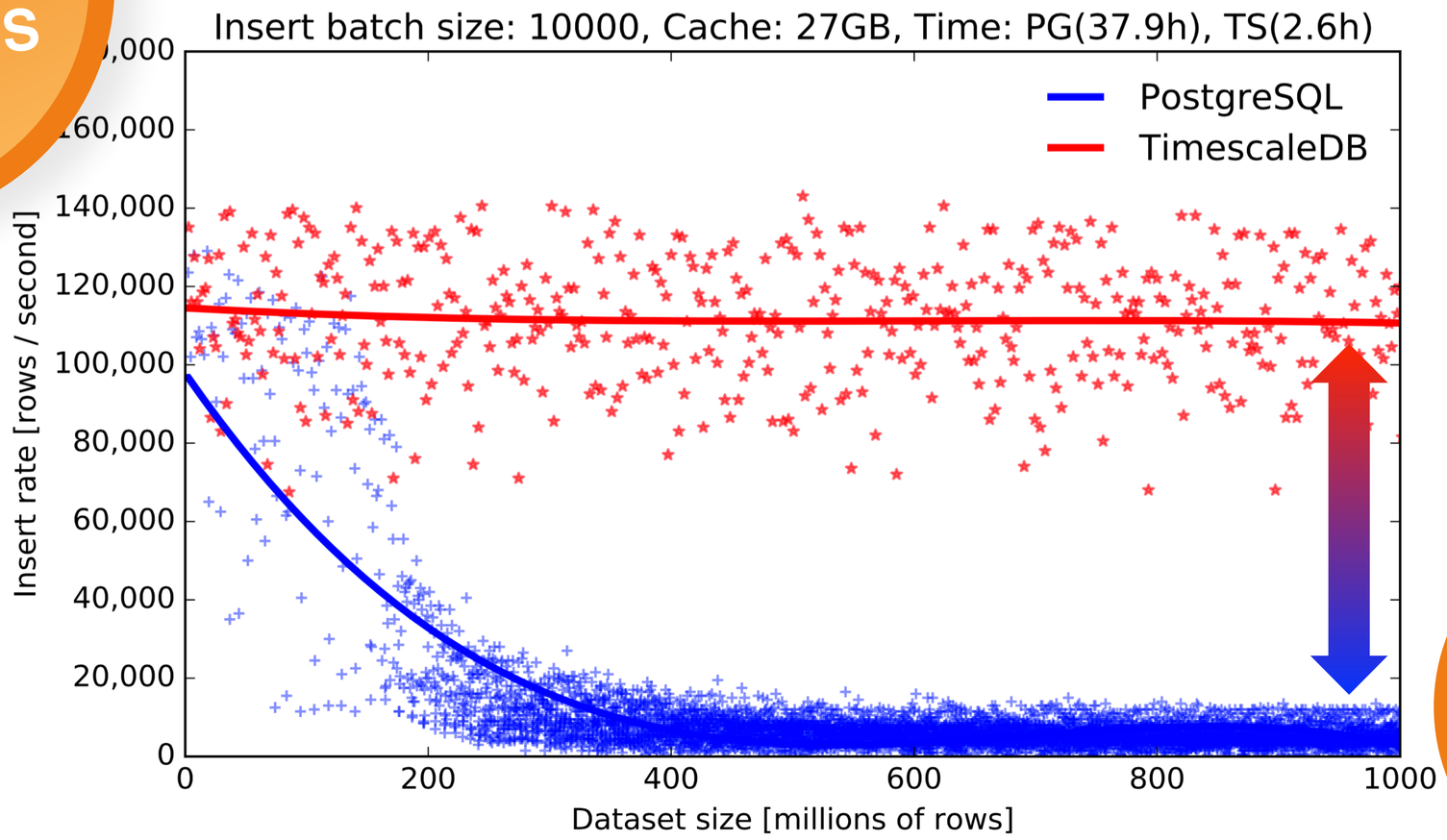
14.4K INSERTS / S

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)

INSERT performance

1.11M METRICS / S



>20x

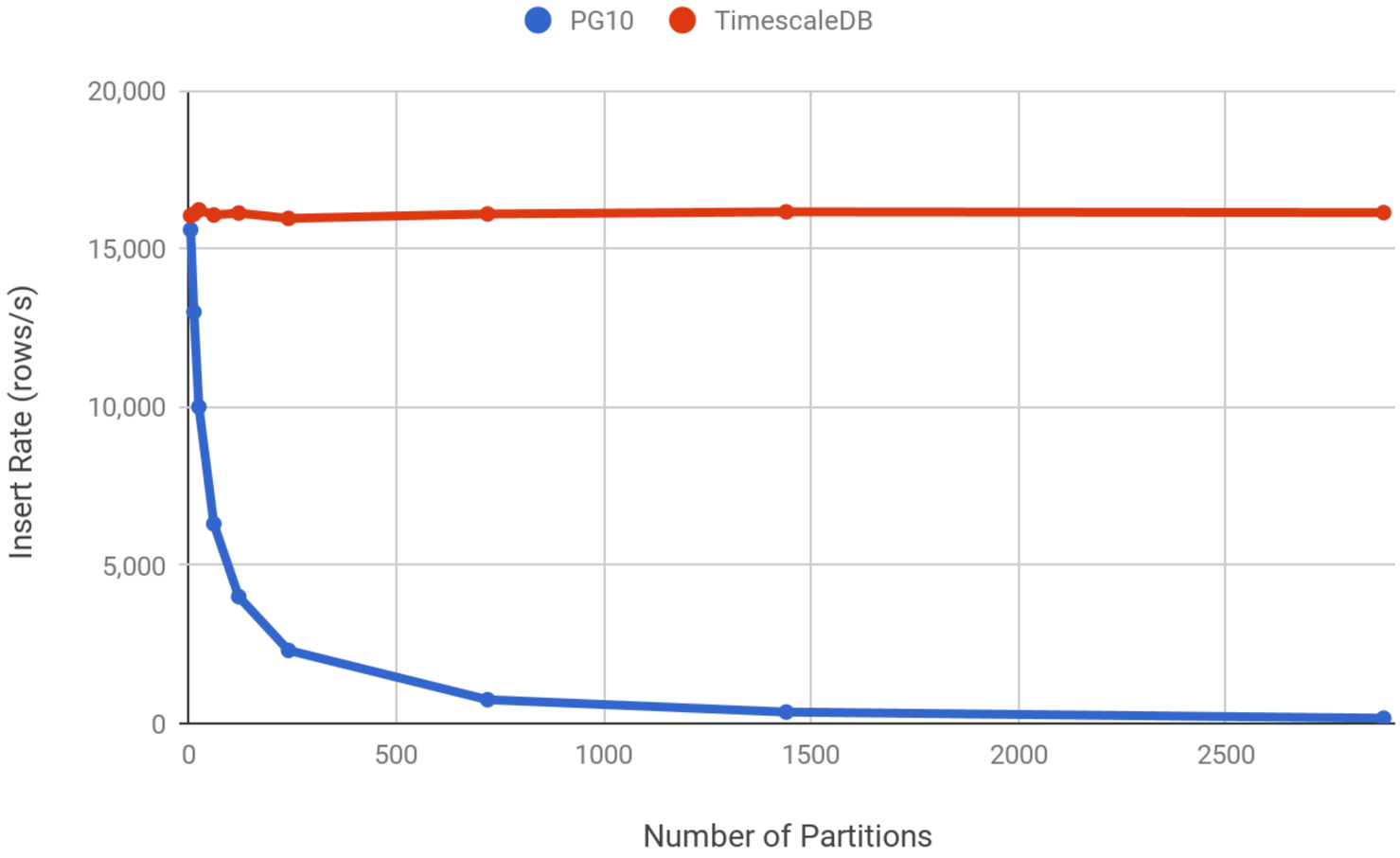
Postgres 9.6.2 on Azure standard DS4 v2 (8 cores), SSD (premium LRS storage)

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

TimescaleDB vs. PG10

Insert Performance as # Partitions Increases

(batch size = 1 row)



Query Performance

	Speedup
Simple column rollups	**0-20%**
GROUPBYs	**20-200%**
Time-ordered GROUPBYs	**400-10000x**
DELETEs	**2000x**

https://blog.timescale.com/timescaledb-vs-6a696248104e

How data is stored

pg_prometheus

Prometheus Data Model in PostgreSQL

- New data type **prom_sample**: <time, name, value, labels>

```
CREATE TABLE metrics (sample prom_smaple);
INSERT INTO metrics
VALUES (‘cpu_usage{service=“nginx”,host=“machine1”} 34.6 1494595898000’);
```

- Scrape metrics with CURL:

```
curl http://myservice/metrics | grep -v “^#” | sql -c “COPY metrics FROM STDIN”
```

Querying raw samples

```
SELECT * FROM metrics;
sample
_____________________________________________________________________________

cpu_usage{service=“nginx”,host=“machine1”} 34.600000 1494595898000
```

```
SELECT prom_time(sample) AS time, prom_name(sample) AS name, prom_value(sample) AS value,
prom_labels(sample) AS labels from metrics;

          time          |    name    |   value    |                  labels
________________________+____________+____________+_____________________________________________

2017-05-12 15:31:38+02 | cpu_usage |    34.6    | {“host”: “machine1”,”service”: “nginx”}
```

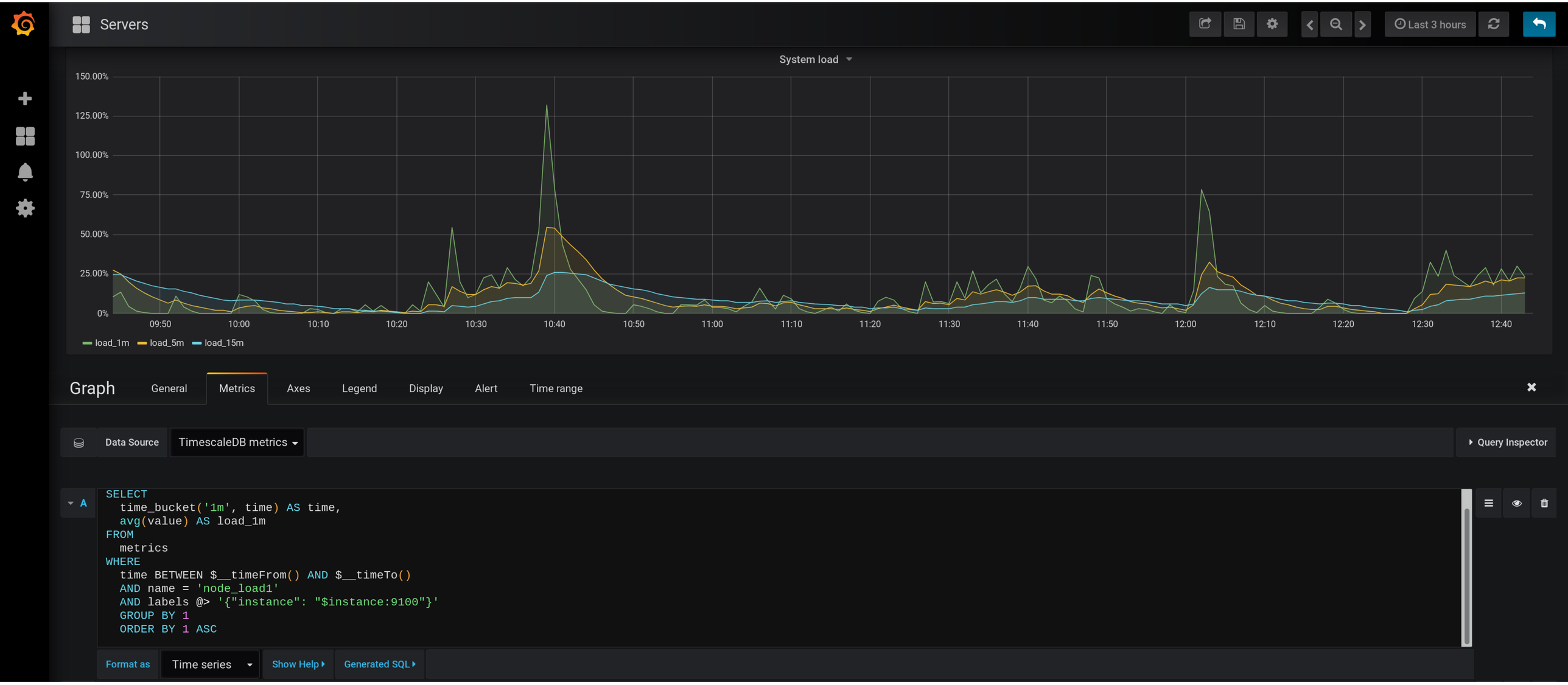
Normalized data storage

```
SELECT create_prometheus_table('metrics');
```

- Normalizes data:
  - values table
  - labels table (jsonb)
- Sets up proper indexes
- Convenience view for inserts and querying
  - columns: | sample | time | name | value | labels |

Easily query view

```
SELECT sample
FROM metrics
WHERE time > NOW() - interval ’10 min’ AND
  name = ‘cpu_usage’ AND
  Labels @> ‘{“service”: “nginx”}’;
```


Servers
Last 3 hours
System load
150.00%
125.00%
100.00%
75.00%
50.00%
25.00%
0%
09:50
10:00
10:10
10:20
10:30
10:40
10:50
11:00
11:10
11:20
11:30
11:40
11:50
12:00
12:10
12:20
12:30
12:40
load_1m
load_5m
load_15m
Graph
General
Metrics
Axes
Legend
Display
Alert
Time range
Data Source
TimescaleDB metrics
Query Inspector
A
SELECT
time_bucket('1m', time) AS time,
avg(value) AS load_1m
FROM
metrics
WHERE
time BETWEEN $__timeFrom() AND $__timeTo()
AND name = 'node_load1'
AND labels @> '{"instance": "$instance:9100"}'
GROUP BY 1
ORDER BY 1 ASC
Format as
Time series
Show Help
Generated SQL

Data Source | TimescaleDB metrics

A

```
SELECT
  time_bucket('1m', time) AS time,
  avg(value) AS load_1m
FROM
  metrics
WHERE
  time BETWEEN $__timeFrom() AND $__timeTo()
  AND name = 'node_load1'
  AND labels @> '{"instance": "$instance:9100"}'
  GROUP BY 1
  ORDER BY 1 ASC
```

Format as | Time series | Show Help | Generated SQL

Variables > Edit

General

Name	instance	**Type**	Query
Label	optional display name	**Hide**	

Query Options

Data source	TimescaleDB metrics	**Refresh**	On Dashboard Load
Query	SELECT DISTINCT labels->'instance' FROM metrics_labels WHERE metric_name = 'node_boot_time';		
Regex	/([0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}):([0-9]+)/		
Sort	Disabled		

Preview of values (shows max 20)

172.31.12.186	172.31.39.219	172.31.32.167	172.31.0.78	172.31.5.159	172.31.25.156	172.31.9.219	172.31.3.161

slack-login.timescale.com

Open-source projects

github.com/timescale/timescaledb

github.com/timescale/pg_prometheus

github.com/timescale/prometheus-postgresql-adapter

hello@timescale.com · **github.com/timescale**