YDB vs. TPC-C: the Good, the Bad, and the Ugly behind High-Performance Benchmarking

Evgenii Ivanov
Yandex Infrastructure

YDB

19 April 2024
Cyprus, Limassol

TIC Tech Internals Conf

YDB vs. TPC-C: the Good, the Bad, and the Ugly behind High-Performance Benchmarking

Evgenii Ivanov
Principal Software Engineer at YDB

Three types of this talk attendees

1

A DBMS developer

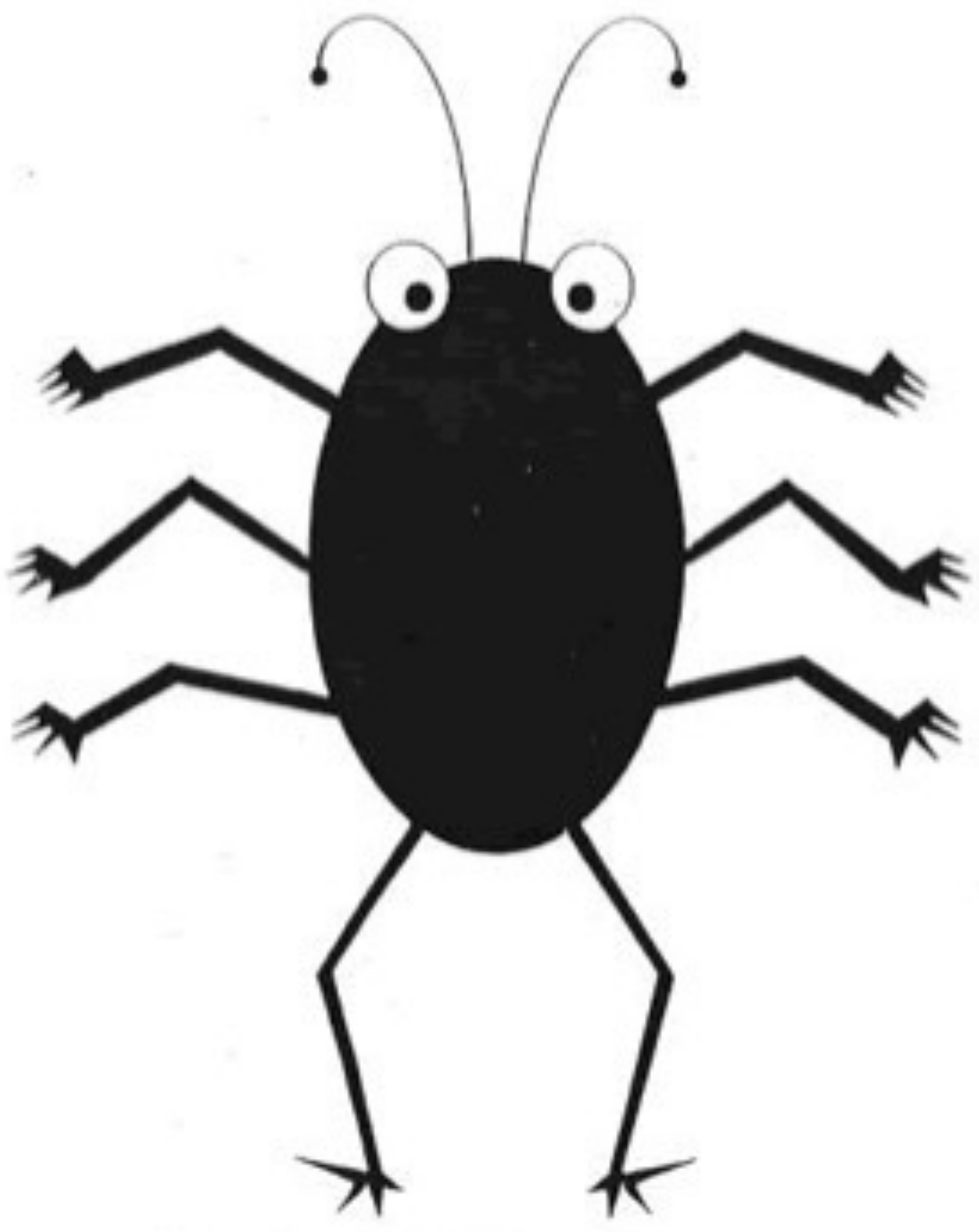
2

A DBMS user (application developer or admin)

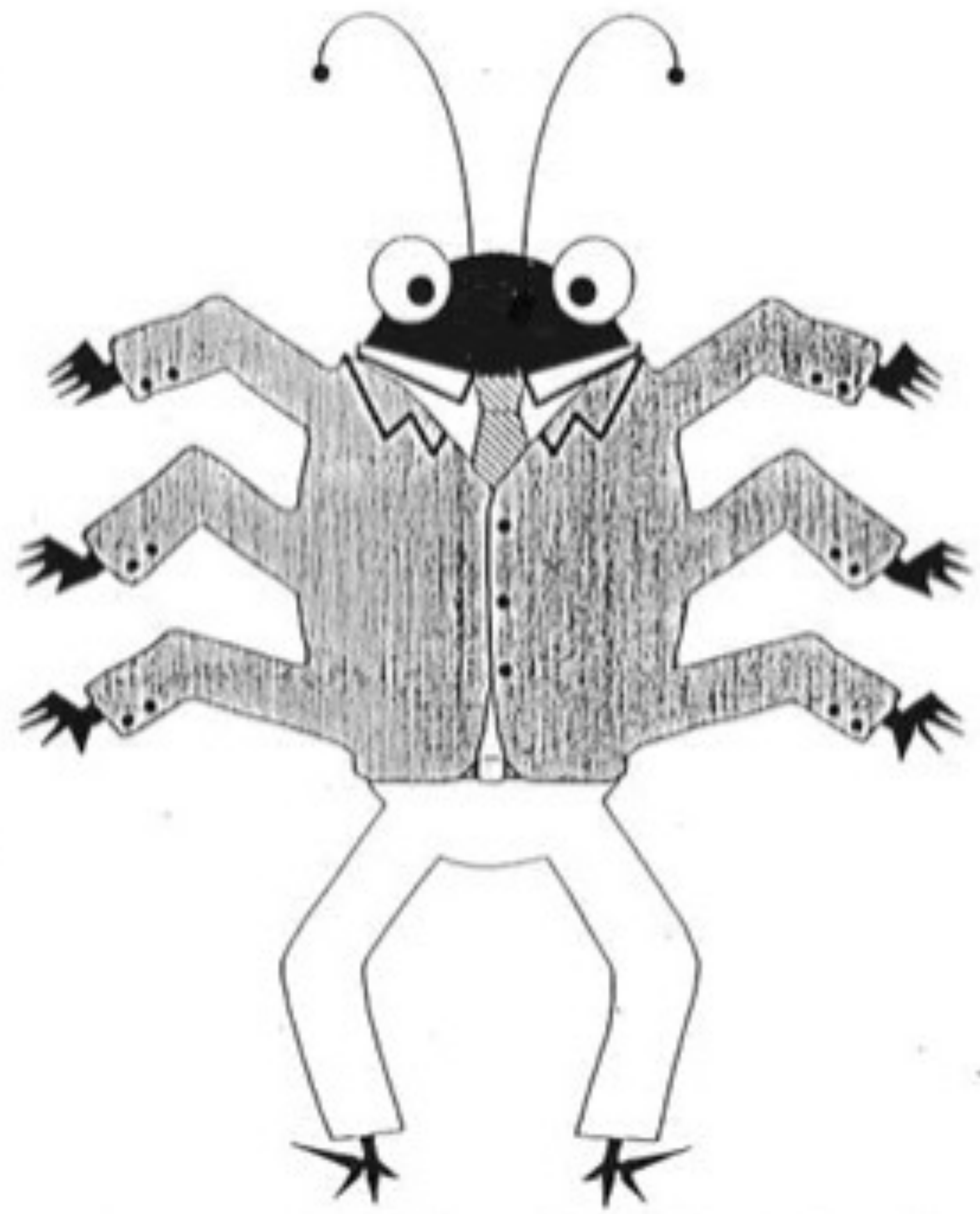
3

A developer without a DBMS dreaming of having one someday

Nothing in common except bugs?



BUG

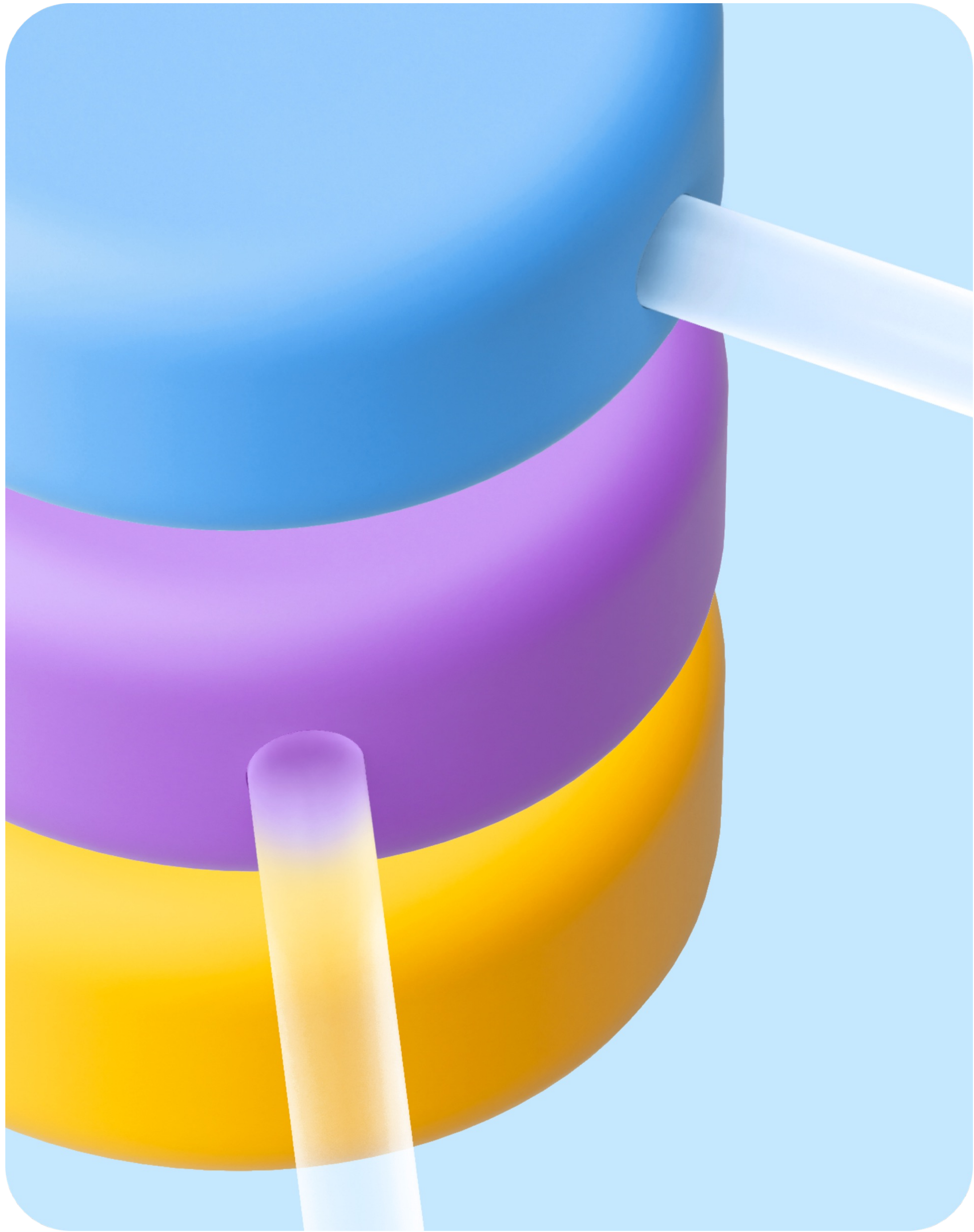


FEATURE

DBMS benchmarks!

Table of contents

YDB overview



YDB

Open-Source Distributed SQL Database

1

Relational DB (mainly OLTP, OLAP is available for testing)

2

Clusters with thousands of servers

3

Apache 2.0 license

4

Star ydb-platform on GitHub

Strictly consistent

1

CAP-theorem —
YDB chooses CP

2

Serializable transaction
execution

Highly available and fault tolerant

1

Multiple availability zones (AZ): automatic recovery

2

YDB is read-write available even after losing an AZ and a rack simultaneously

A mission critical database

1

365x24x7 (366x24x7 when needed)

2

No downtime during a maintenance (e.g. to roll out a new YDB version)

Beyond OLTP: YDB is a platform

1

Column-oriented tables are coming soon and that's not a menace

2

YDB Topic Service (persistent queue)

3

Network Block Store (aka NBS)

4

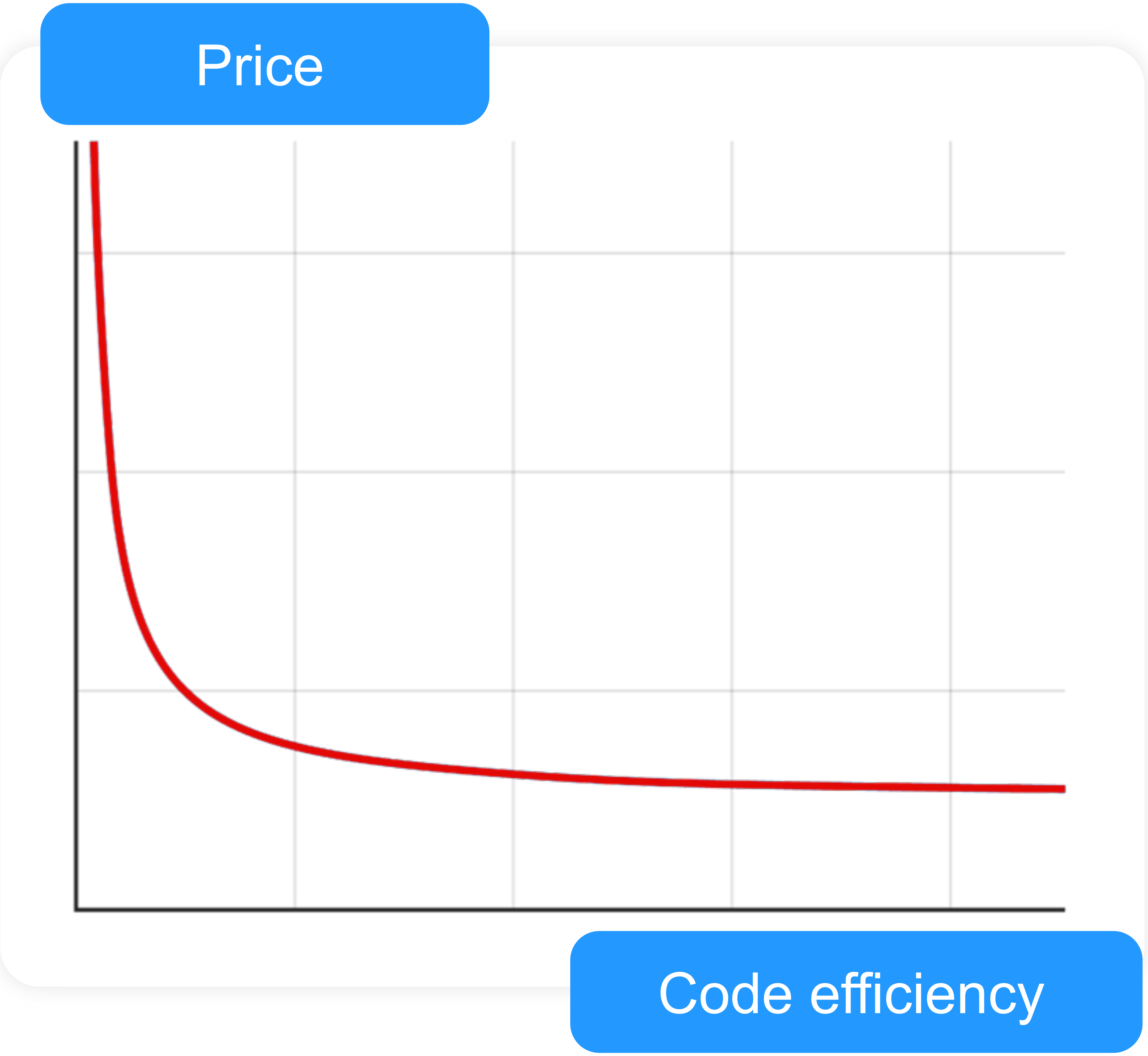
And more

YDB benchmarks evolution

Database performance definition

- **Throughput:** serving infinite number of requests/second
- **Latency:** sending a reply before being requested

The cost of DBMS running



Key focus areas before OSS

1

Scalability without compromising on consistency and fault-tolerance

2

Custom benchmarks

3

Performance tests on a special testing cluster

YDB testing cluster

250

Servers

>1000

Databases

500 TB

of data

After YDB became OSS

1

Focus on efficiency (vs. scalability in the past)

2

Comparison with other open source distributed DBMS

We've started from:

- Yahoo! Cloud Serving Benchmark (YCSB)
- TPC-C — best benchmark for OLTP (and distributed transactions)

Hardware for benchmarking

Distributed vs. Monolithic in Benchmarking Context

- Monolithic databases are limited by single server hardware
- Distributed databases have almost no limits
- Inefficiencies in benchmarks are more crucial: consider overloading DBMS with 16, 128 and 4096 CPU cores

YDB is a benchmark for benchmarks

- **Expectations:** take the benchmark and improve YDB
- **Reality:** take YDB and improve the benchmarks

Yahoo! Cloud Serving Benchmark

1

A popular key-value benchmark

2

Created for NoSQL key-value DBs but still loved by everybody

3

Supports almost all modern databases

Why key-value?

1

A lot of people still need key-value

2

It's easy to analyze the results of YCSB

3

You can't do distributed transactions well if you can't do key-value workloads well

TIC

Key findings

1

Quickly spotted multiple bottlenecks while using YCSB

2

Added YCSB runs to CI as a performance regression test

3

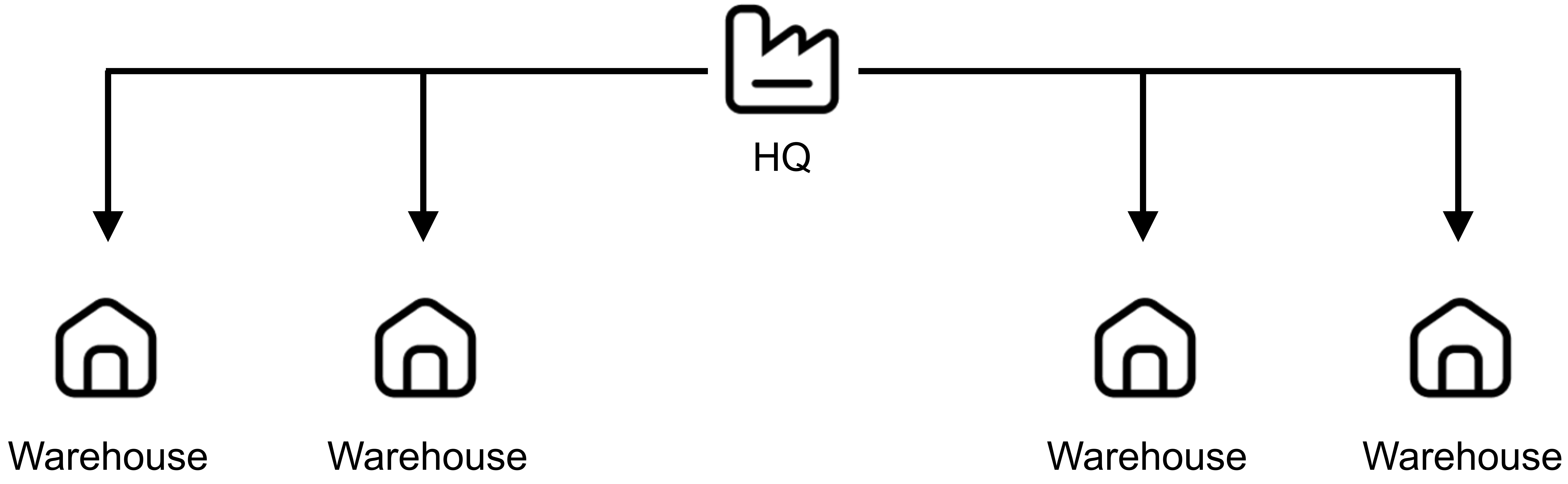
Discovered that YCSB consumes a lot of hardware resources on the client side by its design

TPC-C has the same HW consumption issues

TPC-C

- Since 1992
- «The only objective comparison for evaluating OLTP performance» — CockroachDB
- YugabyteDB and TiDB also stated that TPC-C is the most objective performance measurement of OLTP systems

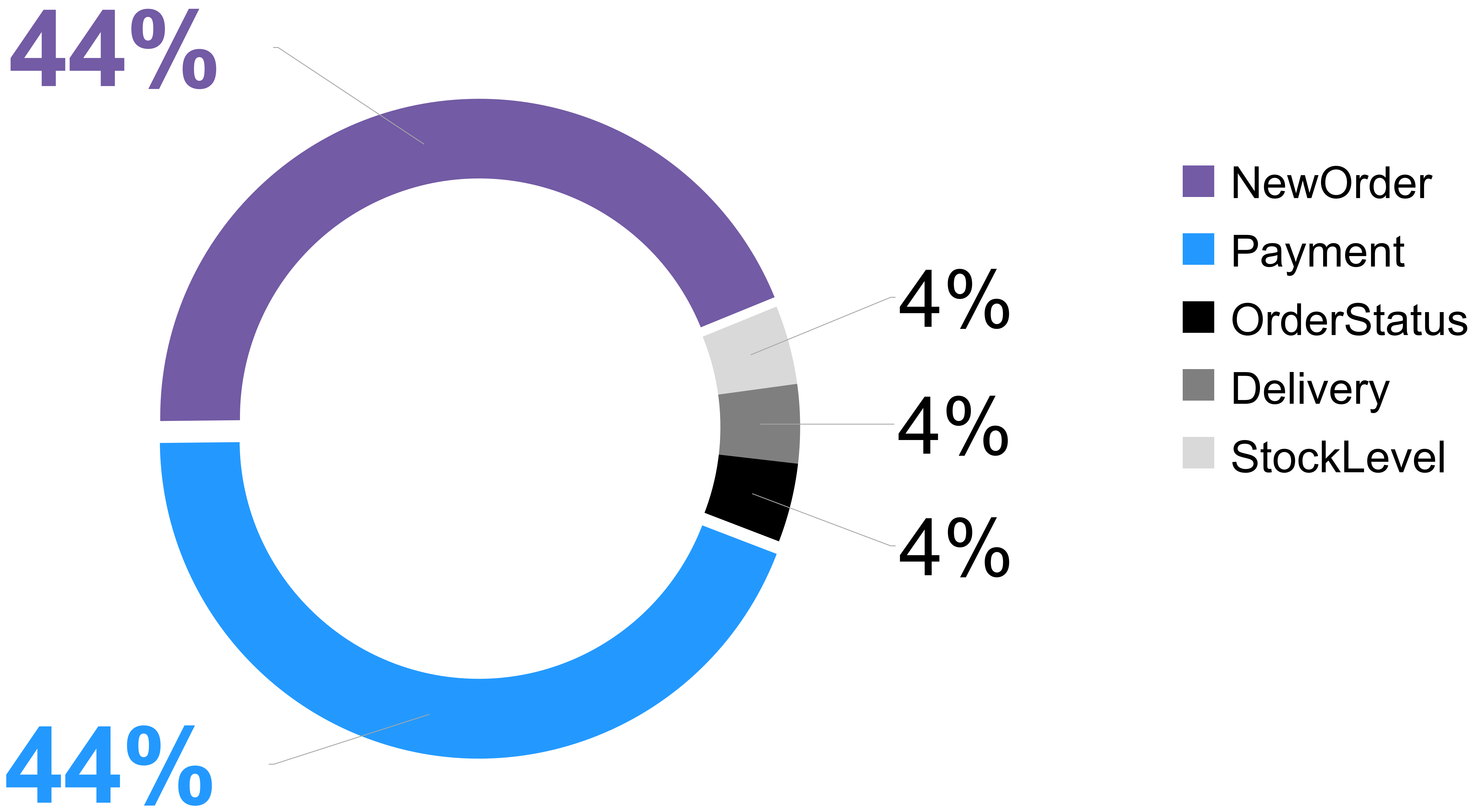
Simulates an e-commerce organization



TPC-C logic in a nutshell

- Number of warehouses is a parameter
- Each warehouse serves 10 districts (around 100 MB of data)
- Each district has a terminal
- Customers use a terminal for orders and payments
- Sometimes customers check the order status
- Delivery is handled by database as well
- Warehouses rarely make inventorization

TPC-C transactions



TPC-C transactions

Require serializable level of isolation

Require multi-step transactions

Are write intensive workload (2:1 writes/reads)

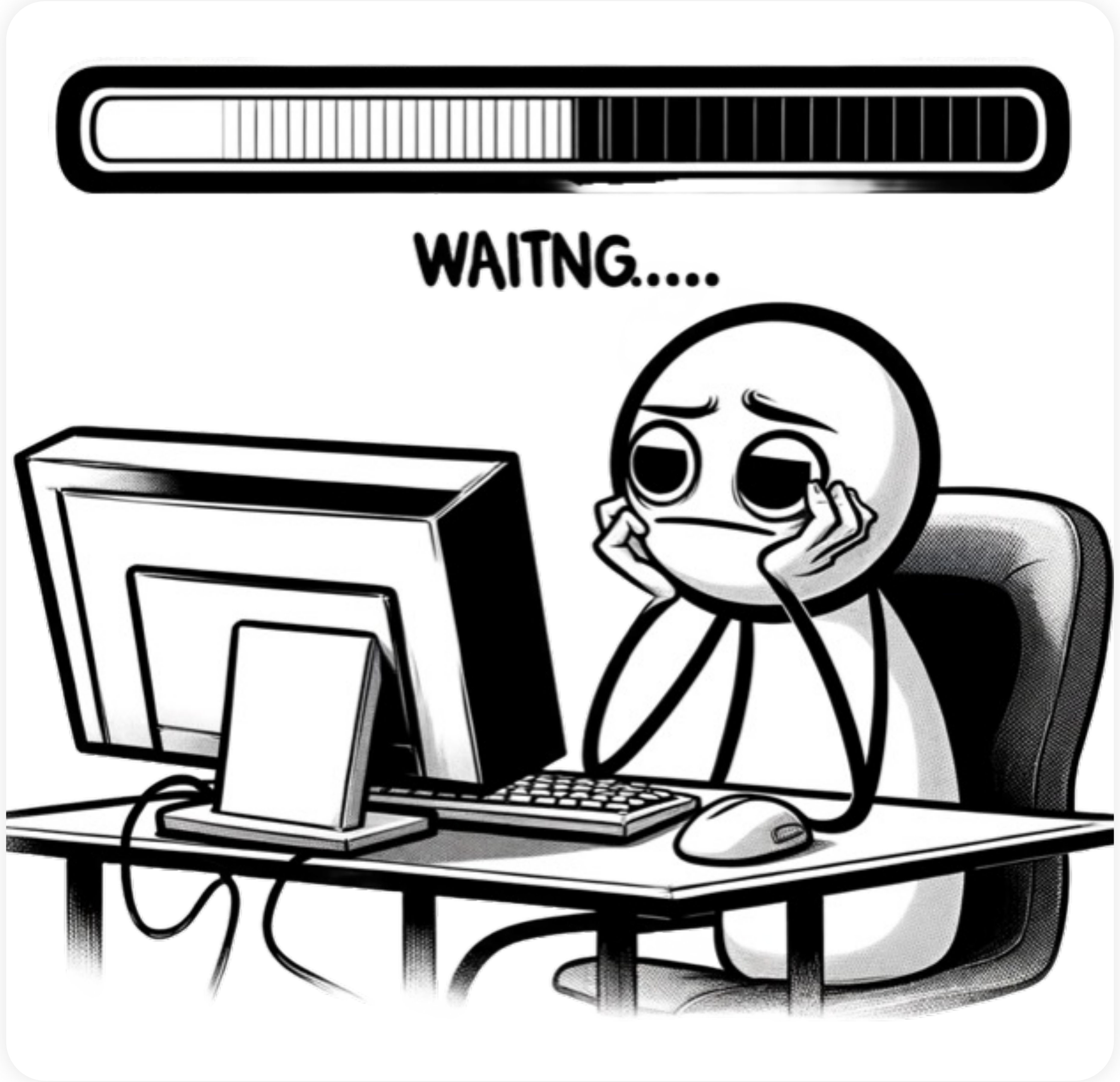
Benchmark measures the number of New Order transactions per minute — tpmC

CMU Benchbase

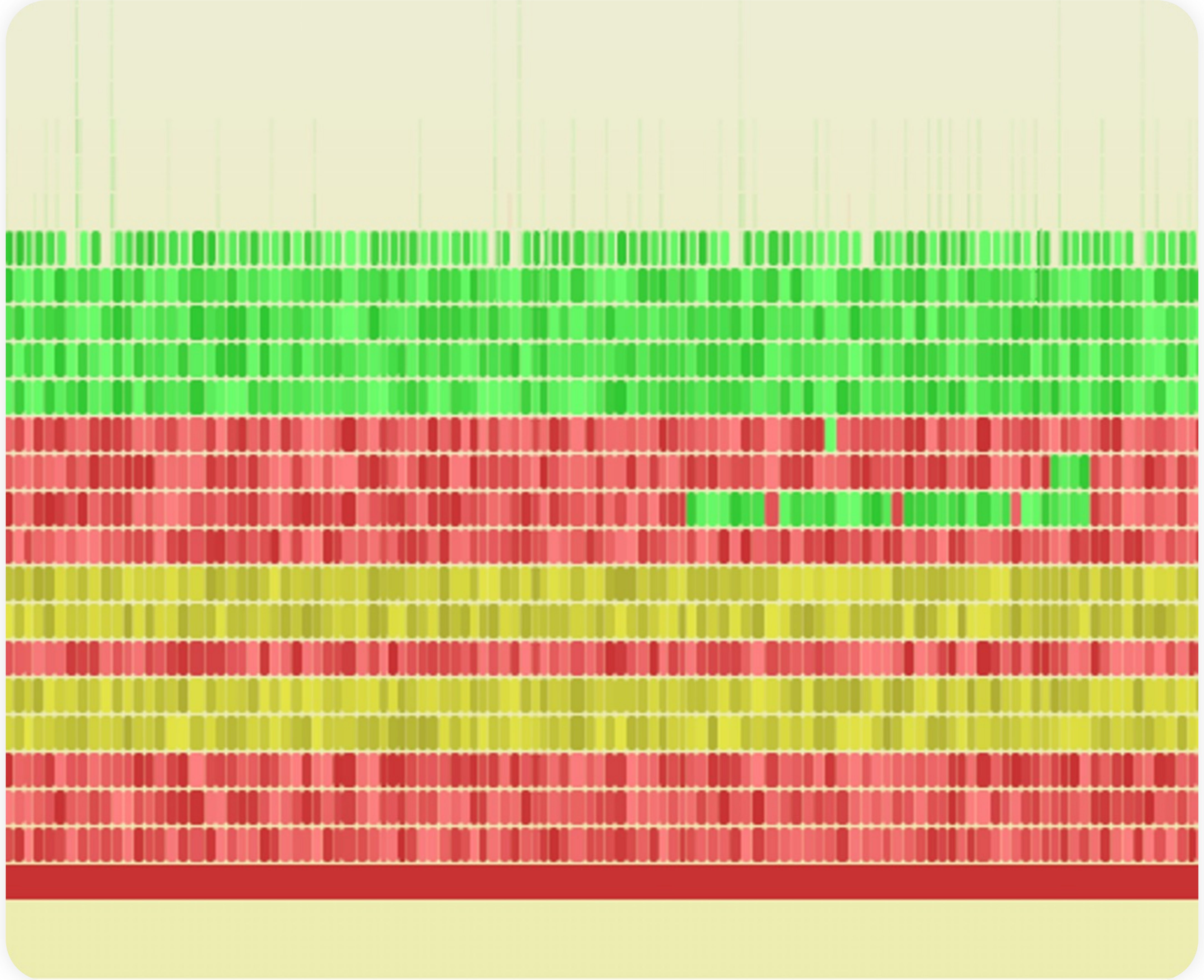
- Multi-DBMS SQL Benchmarking Framework via JDBC
- Developed by Carnegie Mellon under Andy Pavlo’s supervision
- It’s easy to add new DBMS and benchmarks
- The only well known TPC-C implementation
- YugabyteDB uses Benchbase fork
- We had to fork too (with a goal to upstream the YDB support)

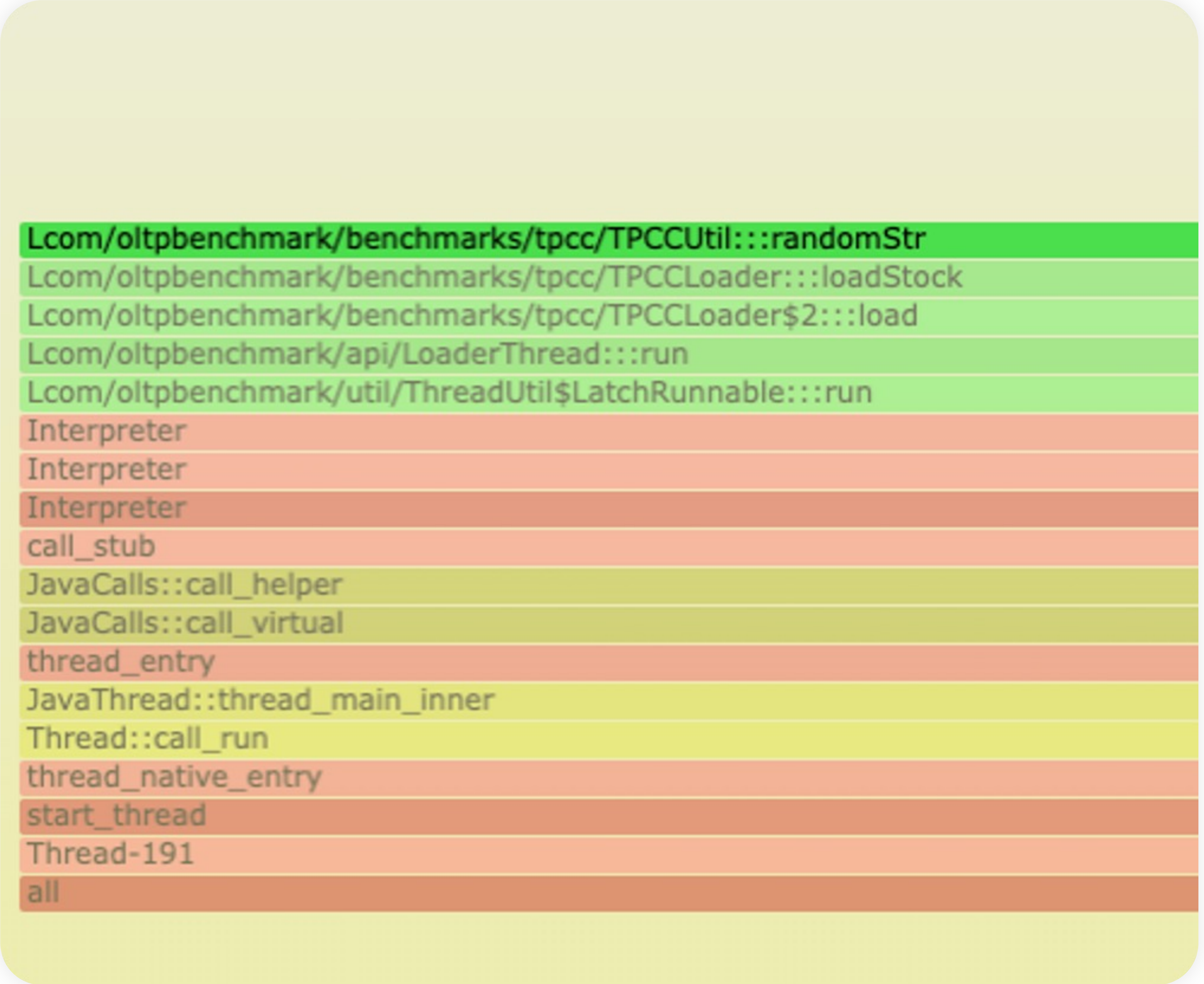
Issue 1: Data import via INSERT

- Terabytes of initial data
- Usually DBMSs have a faster import operations like bulk upsert in YDB
- Waiting for hours to import the data



High CPU usage by benchmark itself





Multithreaded benchmark with a single lock

1

Import threads generate random strings

2

They share the same `java.util.Random` object

3

Had to change to `ThreadLocalRandom`

Issue 2: One warehouse terminal — one thread

- Our minimal setup — **15K warehouses**
- 15K warehouses — **150K terminals**

Sync vs. Async

- We want concurrency without too many threads
- It’s hard to write async programs in old languages
- Future/Promise model
- Goroutines — simple and efficient
- Java virtual threads — Java’s attempt to go Go

Issue 3: Benchmark consumes too much RAM

40 MB

Single warehouse

15 000

Warehouses

600 GB

RAM

Initial benchmark requirements to run 15K warehouses

150K

Threads

600 GB

RAM

To test YDB running on 3 servers, we used 5 servers to run the benchmark (each 128 cores and 512 GB RAM)

Scaling out

- DBMS with 9, 15, 30, 60, 81 servers
- YDB, CockroachDB, YugabyteDB
- Single TPC-C run in AWS costs **$10,000**
- Multiple runs?

Minimum changes — maximum benefit

1 Java virtual threads (Java >= 21)

2 1 terminal — 1 **virtual** thread

3 Aggregate transaction history

- 6 MB RAM per warehouse (instead of 40)
- 1 CPU core per 1000 warehouses
- 15K warehouses — 90 GB RAM, 15 CPU cores

Deadlock for free

1 Number of sessions is limited

2 Some vthreads hold session waiting for network I/O and loose carrier thread

3 Other vthreads call `Object.wait`() to obtain a session and block carrier thread

- Java virtual threads is a silver bullet for Russian roulette
- Very easy to get deadlock

Our fork and upstream

1

github.com/ydb-platform/tpcc

2

We plan to upstream the improvements

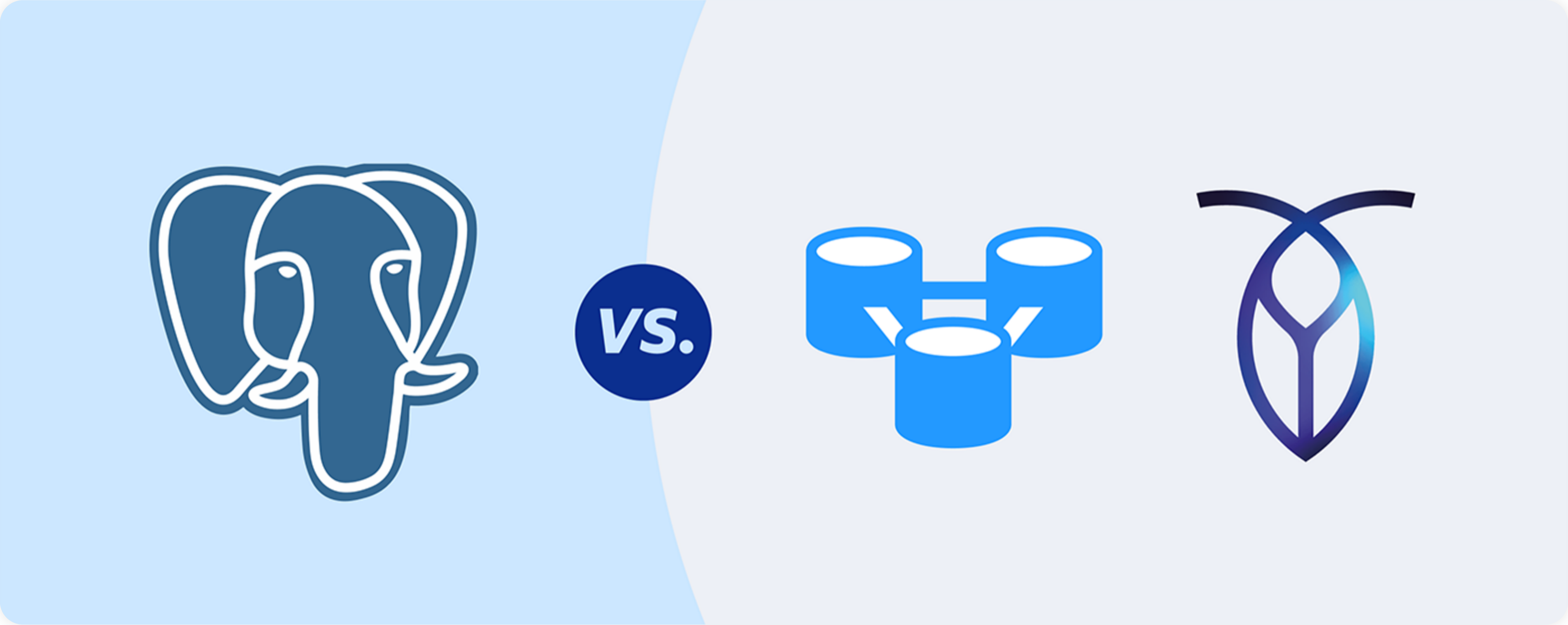
What happens when you compare them?

vs.

vs.

PostgreSQL appears

PostgreSQL vs. Distributed



Yet another Benchbase fork

1

https://github.com/ydb-platform/tpcc-postgres

2

Everything we've discussed + HikariCP

Test setup: 3 servers

- 128 logical cores: 2x32-cores Intel Xeon Gold 6338 CPU @ 2.00GHz with hyper-threading turned on
- 4xNVMe disks
- 512 GB RAM
- 50 Gb network
- Transparent hugepages turned on (huge pages in case of PostgreSQL)
- Ubuntu 20.04.3 LTS

DBMS should survive a single server failure

PostgreSQL has two sync replicas

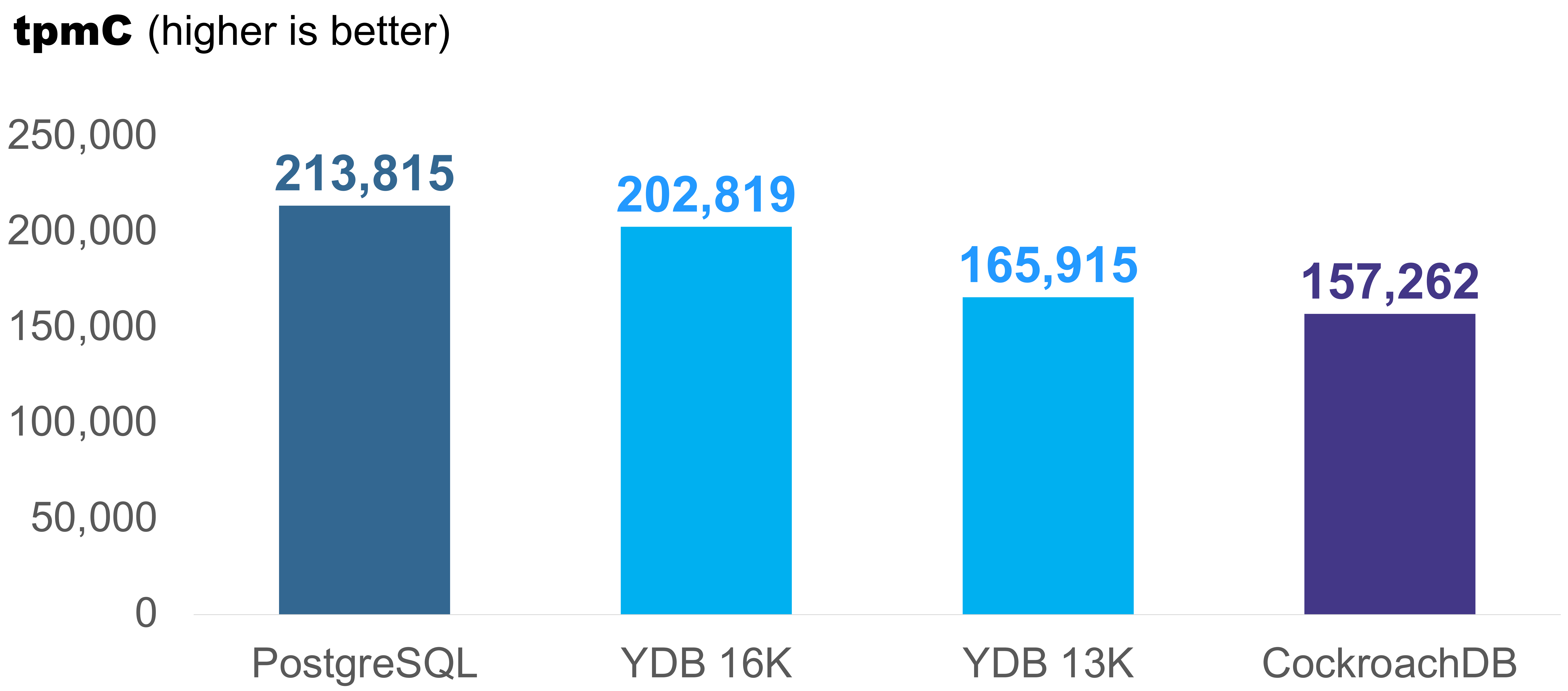
CockroachDB and YDB use replication factor 3

Infinite PostgreSQL configurations

Postgres configurations summary

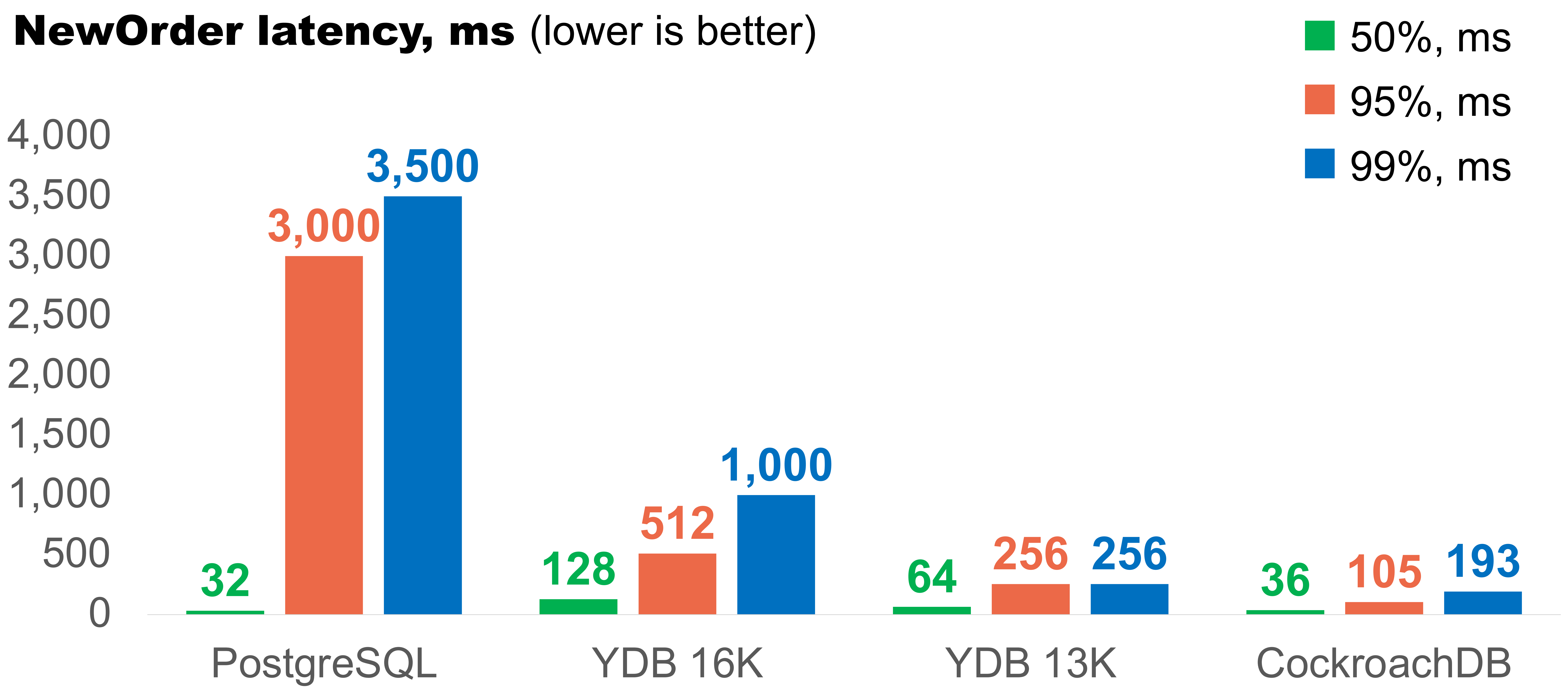
- Postgres “fault intolerant” setups are extremely performant
- Sync replication is a huge bottleneck and limits vertical scalability
- More information can be found in the YDB blog

tpmC* (throughput)



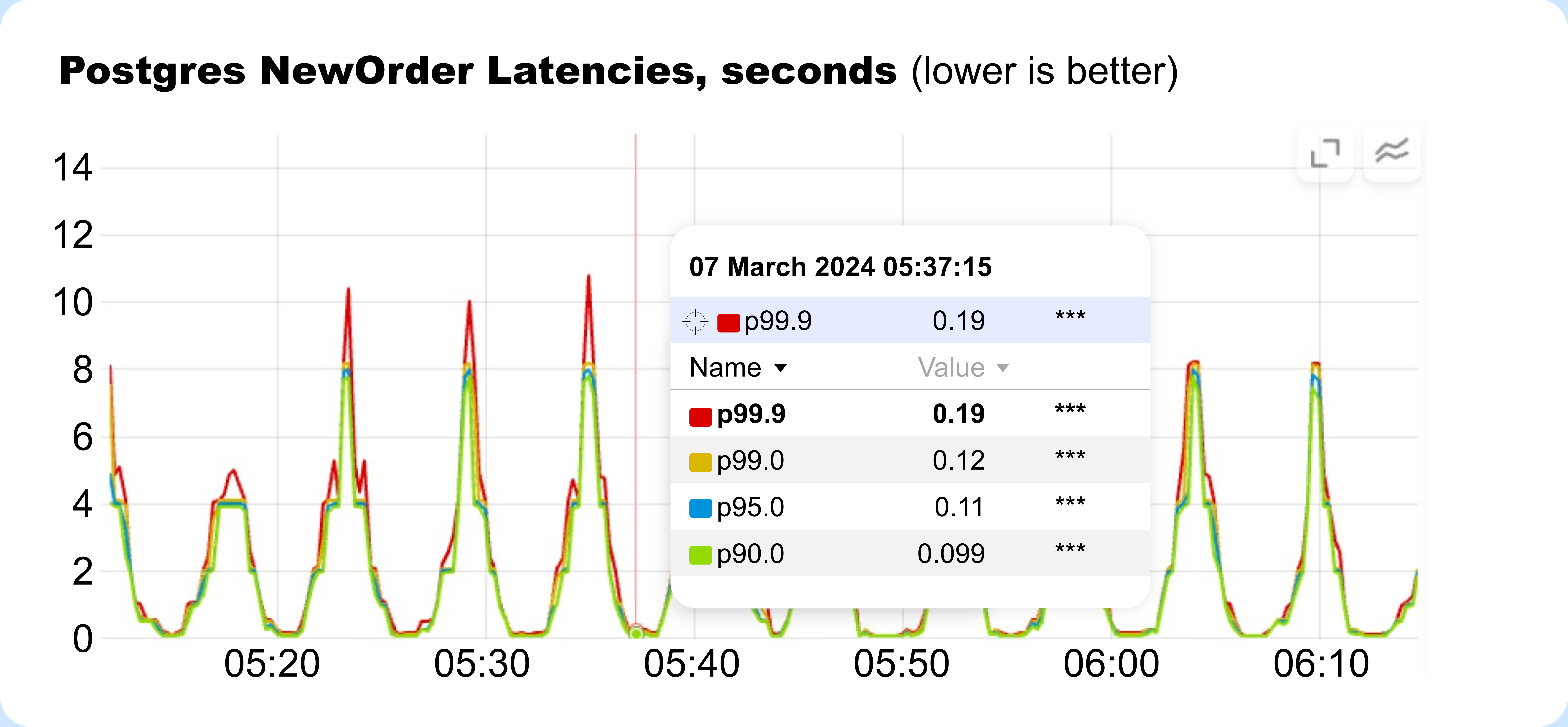
** The results are not officially recognized TPC results and are not comparable with other TPC-C test results published on the TPC website.*

Latency

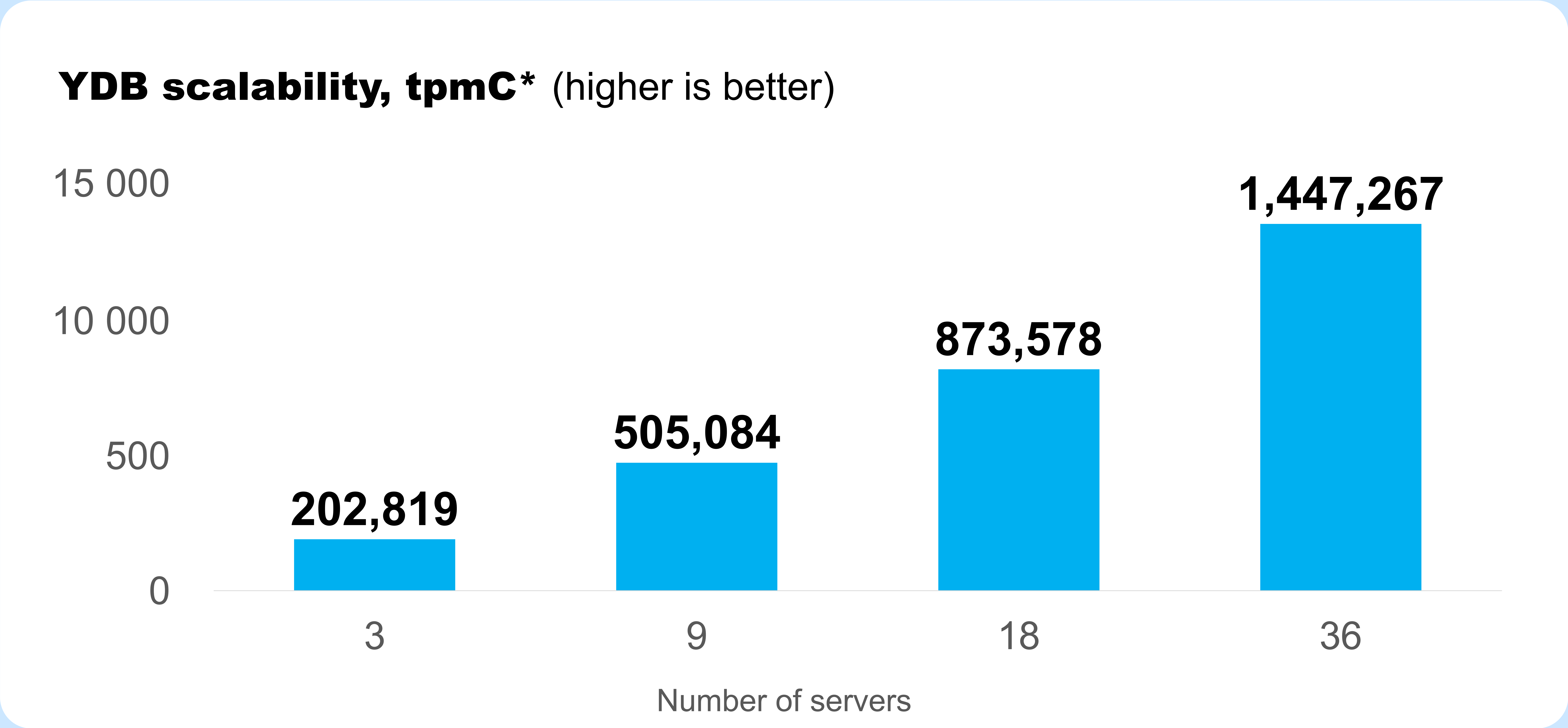


NewOrder latency, ms (lower is better)
50%, ms
95%, ms
99%, ms
4,000
3,500
3,000
2,500
2,000
1,500
1,000
500
0
32
3,000
3,500
128
512
1,000
64
256
256
36
105
193
PostgreSQL
YDB 16K
YDB 13K
CockroachDB

NewOrder latency in Postgres



YDB scalability



** The results are not officially recognized TPC results and are not comparable with other TPC-C test results published on the TPC website.*

TPC-C results summary

1

PostgreSQL wins attaining 5% more tpmC than YDB

2

PostgreSQL exhibits significantly higher latency

3

YDB holds a 29% tpmC advantage over CockroachDB

4

Distributed DBMSs can be easily scaled by adding commodity hardware

Conclusions

- Be ready to improve OSS benchmarks
- Implement benchmarks
  in a way, that they don't consume more
  resources than DBMS
- YCSB and TPC-C are great benchmarks
- PostgreSQL might not be enough,
  and distributed DBMSs shine even
  in clusters with just three servers

YDB blog, community,
presentations, recordings

Please leave your feedback

- Be ready to improve OSS benchmarks
- Implement benchmarks
  in a way, that they don't consume more
  resources than DBMS
- YCSB and TPC-C are great benchmarks
- PostgreSQL might not be enough,
  and distributed DBMSs shine even
  in clusters with just three servers

Evgenii Ivanov, @eivanov89
Principal Software Engineer at YDB

YDB