# PeakBench:
# A Benchmark for Scalable Transaction Processing Systems

Chunxi Zhang, Yuming Li, Rong Zhang, **Weining Qian**, Aoying Zhou

School of Data Science and Engineering,
East China Normal University

{cxzhang, liyuming}@stu.ecnu.edu.cn
{rzhang,wnqian,ayzhou}@dase.ecnu.edu.cn


# Outline


Motivation and Background


Architecture


Implementation


Results

# Scalable Transaction

## Phenomenal workloads: Single's Day Promotion

# Phenomenal workloads is common

- 12306 (train ticket buying during the Spring Festival period)
- Black Friday
- Banking systems to support various promotion and second-kill apps.
- ...

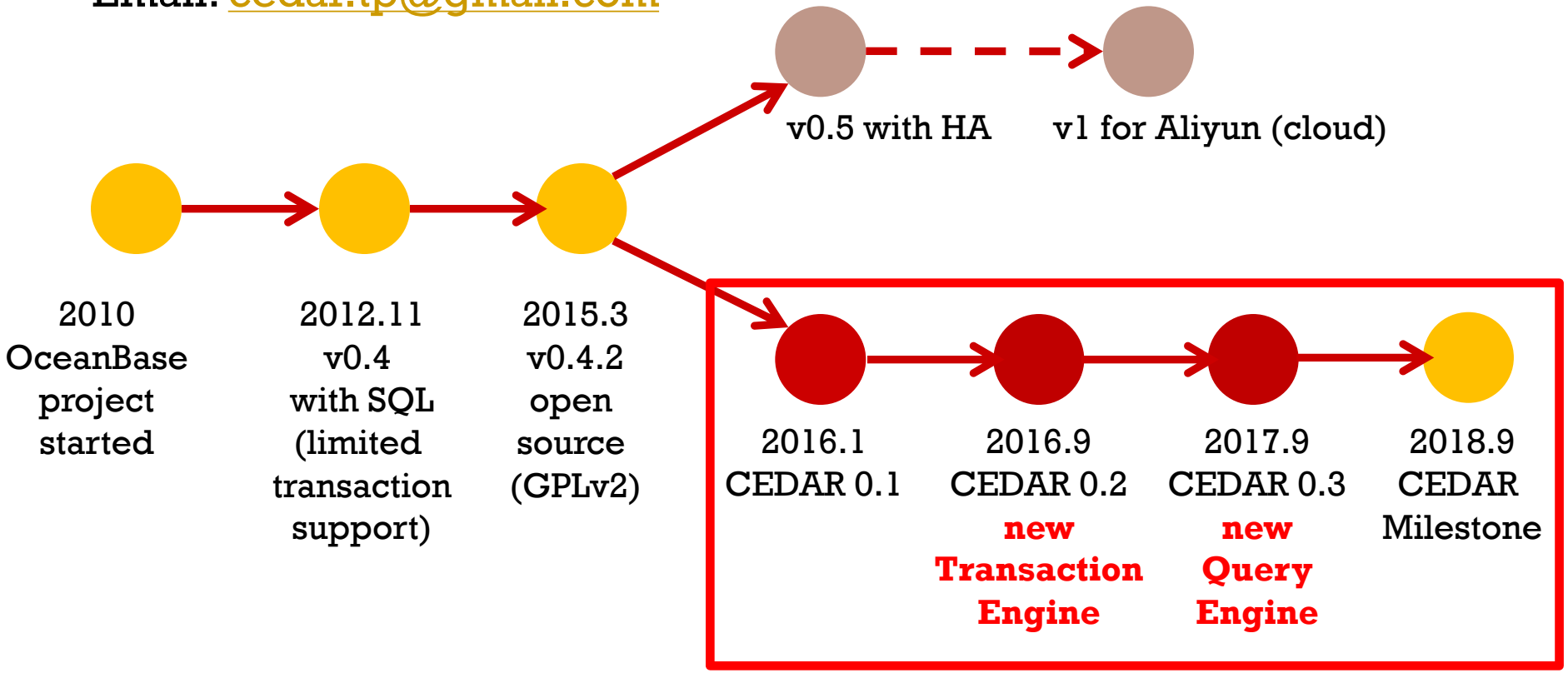
- The key problem is transaction processing in backend paying systems

# What is scalable transaction processing

- Transaction processing
  - Guarantees ACID properties
- Scalability
  - Peak workload >> standard workload in terms of throughput (80x ~ 100x is common)
- Usually depends on scaling-out architecture in stead of scaling-up architecture

# Motivation

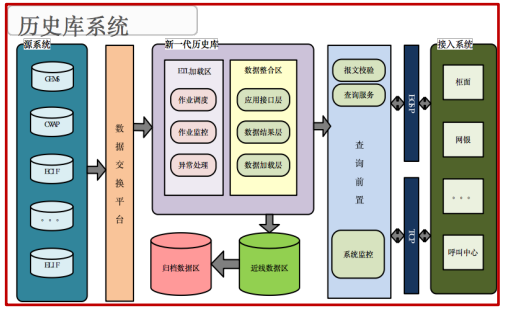
Homepage: https://github.com/daseECNU/CEDAR
Email: cedar.tp@gmail.com



# Applications

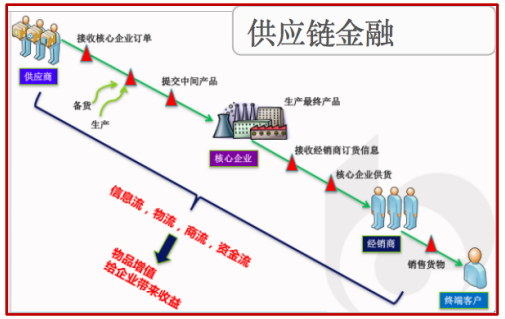

Bank notes recording

Massive datasets



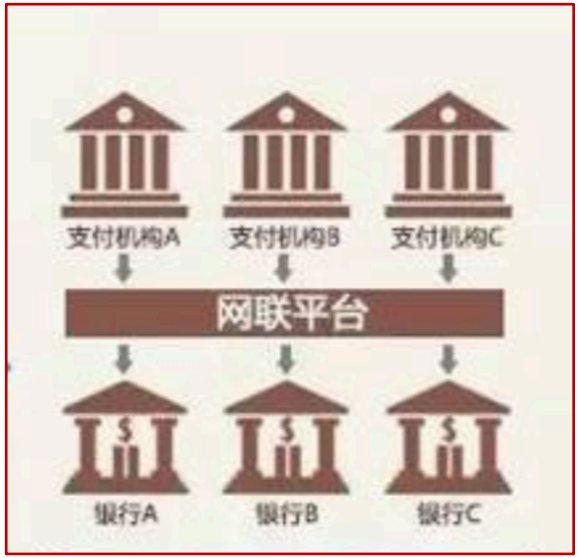

History repository

Class-A app.




Supply-chain finance

Replacing IBM DB2




Netpay

Debt-Credit

Internet-scale

HA

# Transaction Processing is not a New Story

## Database systems

- DB2, Oracle, PostgreSQL, …

## TP Monitors

- CICS, Tuxedo, HP NonStop, …

## NewSQL systems

- H-Store, HANA, MS Hekaton, OceanBase

# Classic and Widely used TP Benchmarks

| Benchmarks | Year | People/Organizations | Descriptions |
|---|---|---|---|
| DebitCredit | 1985 | David DeWitt et al. | TP benchmark, Bank application |
| TPC-C | 1992 | TPC | TP benchmark, Warehouse-centric application |
| TPC-E | 2007 | TPC | TP benchmark, stock brokerage application |
| TATP | 2009 | IBM Software Group | TP benchmark, Caller location system |
| SmallBank | 2009 | University of Sydney | TP benchmark, Bank application |
| YCSB/YCSB++ | 2010/2011 | Yahoo! Research/ Carnegie Mellon University, National Security Agency | TP benchmark |
| CH-benCHmark | 2011 | Dagstuhl Germany | Hybrid workload, TPC-C & TPC-H |

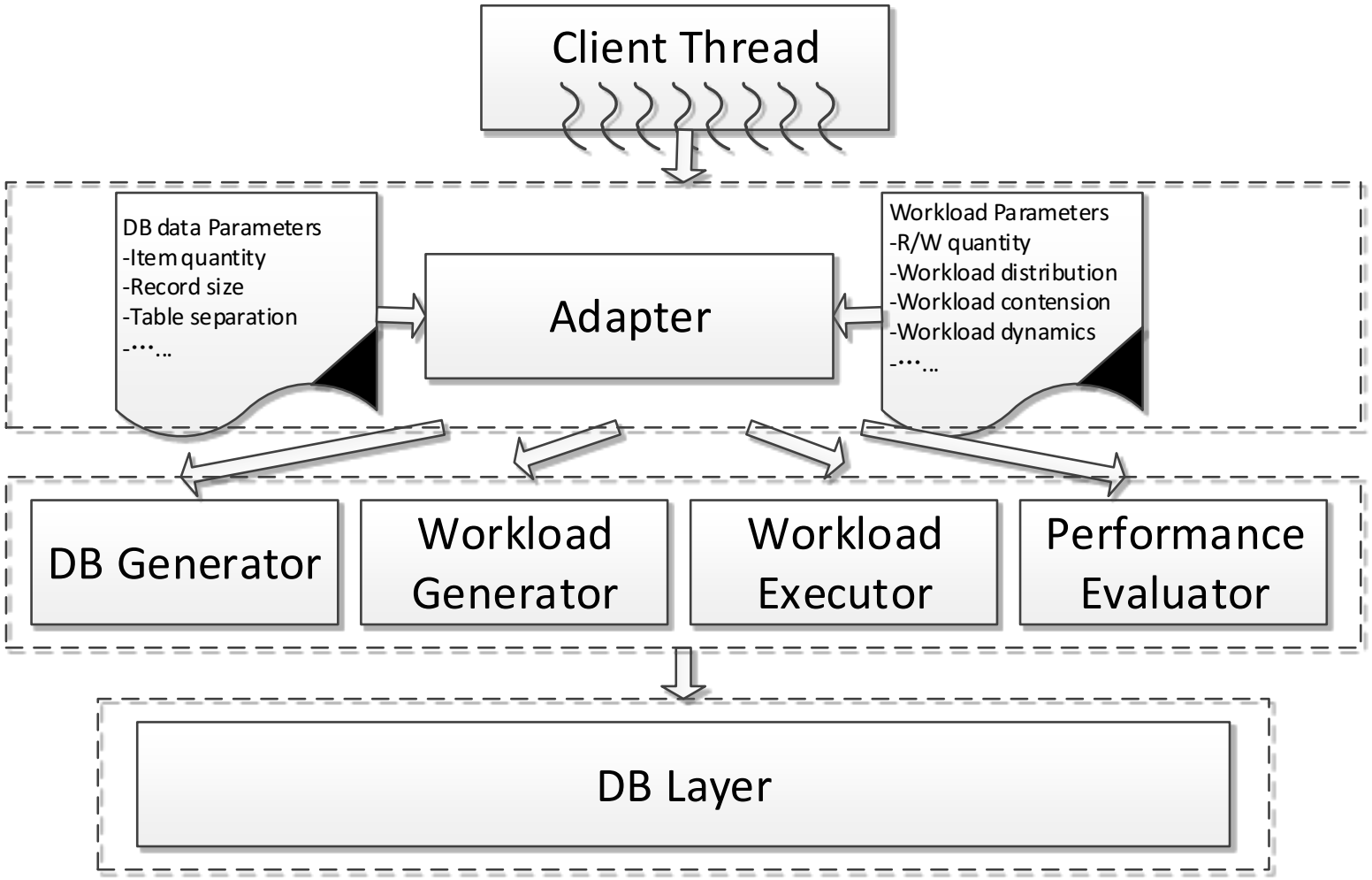
# Why existing benchmarks are not enough?

- Architecture
  - Scalable TP is not achieved only by the TP system
- Workloads
  - Should be configurable and scalable
  - Should be: Concurrency, Contention, Skewness, and Dynamics
- Measurements
  - Scalability is a new measure

# SecKill Applications

- SecKill is an application where a large number of users snap up scarce resources in a short period of time, such as "11.11" of Tmall, "ticket booking" during China Spring Festival and "Stock Exchange" applications.
- Three phase of SecKill
  - ReadPhase: Before SecKill start time point, all SecKill items can only be browsed (Read) since customers are searching or keeping monitoring the interested ones. It will be read-heavy.
  - CriticalityPhase: It reaches read peak closing SecKill time. Almost all customers start focusing on the status changing of items and continuously refreshing, when SecKill time approaches.
  - KillPhase: It turns almost all browsers (R) into buy (W) at the point of kill start. It will be write-heavy and contention-intensive, especially for hot items.

# Architecture




Generation control

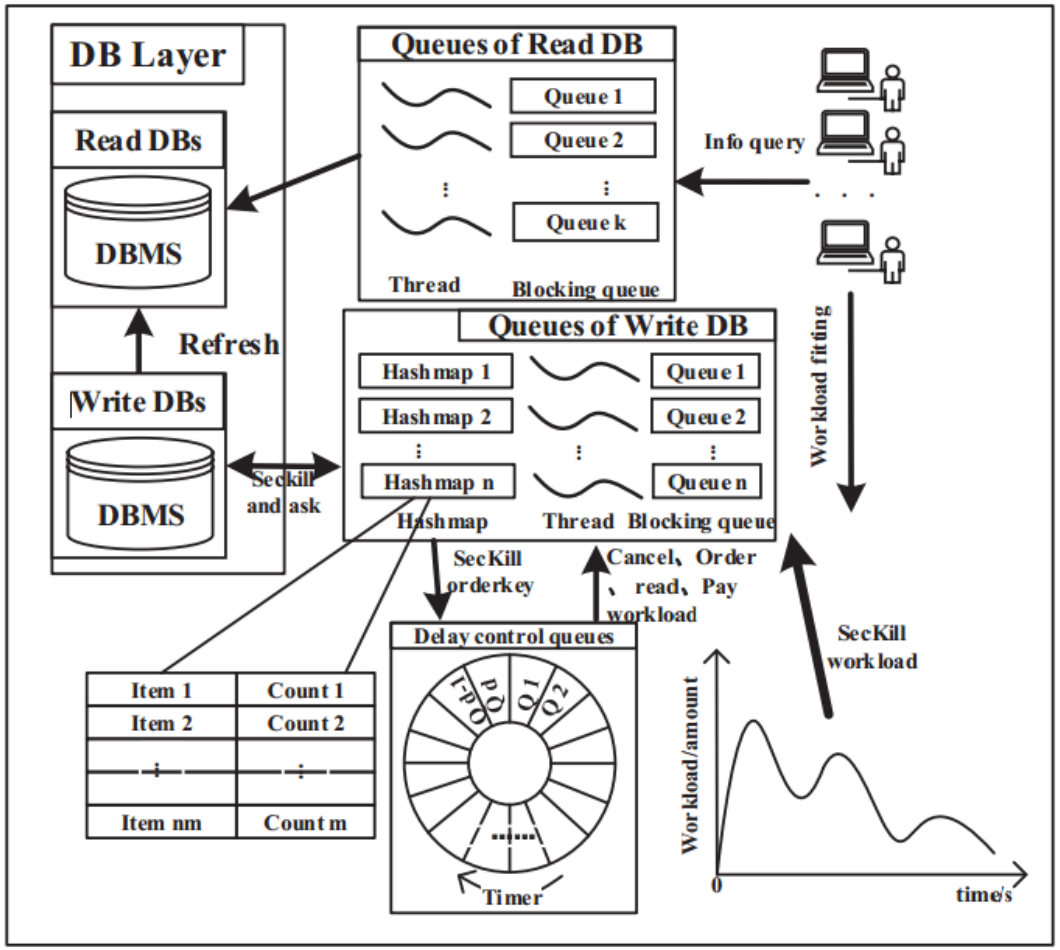
Configuration

Generator

Storage

# Architecture

- Modeling of a SecKill application
- Key: separation of Read DB and Write DB



# Implementations

## Generation Control for Extensibility

- Our data/workload  generators can run on threads, supporting distributed deployment.

## Configuration for Factuality

- Our allows to declare both the static and dynamic demands on data and workloads characteristics.

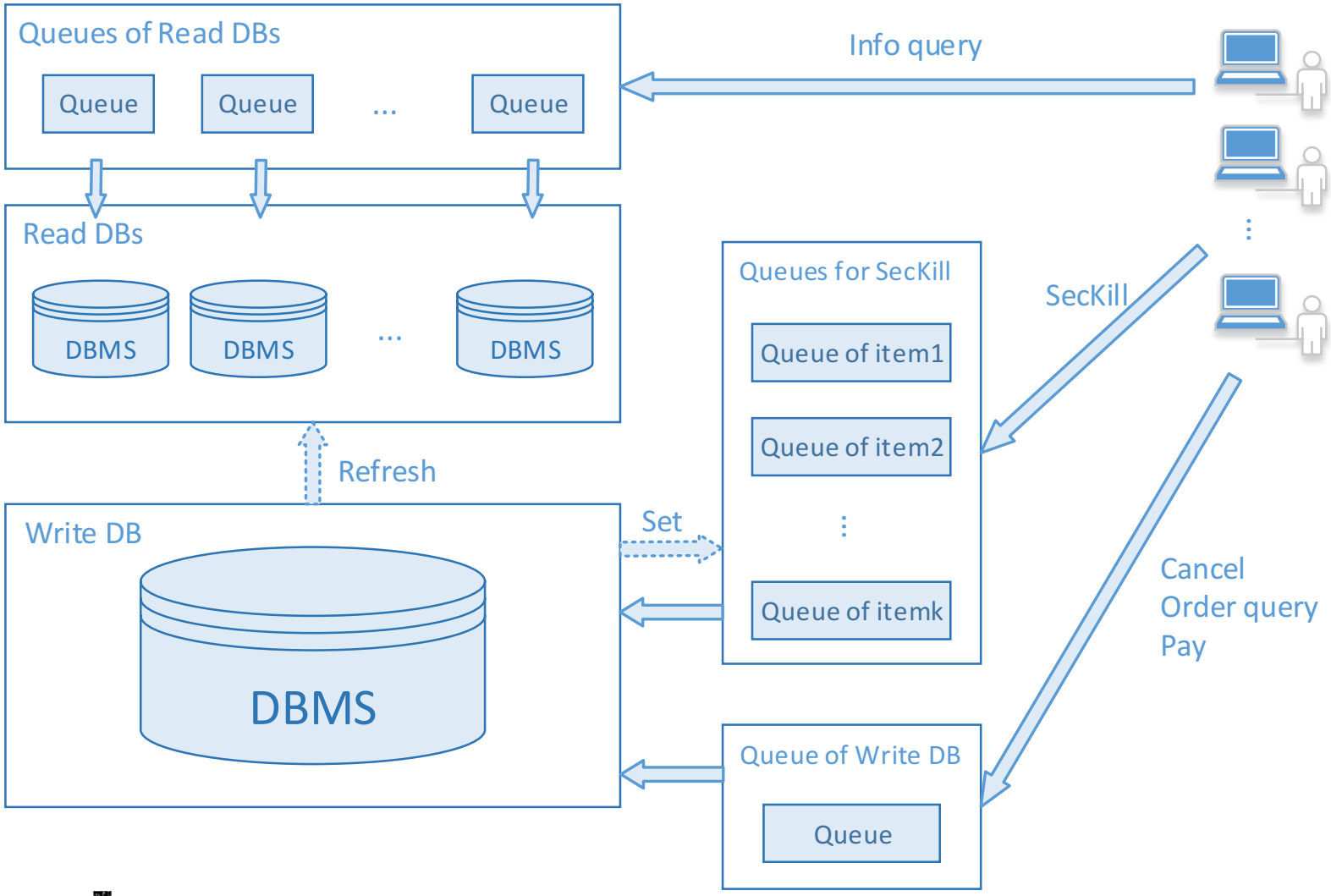
## Executor for Adaptability

- Data generator and workload generator are designed based on requirements assigned.
- We design new evaluation metrics to show system stability under dynamic environment.

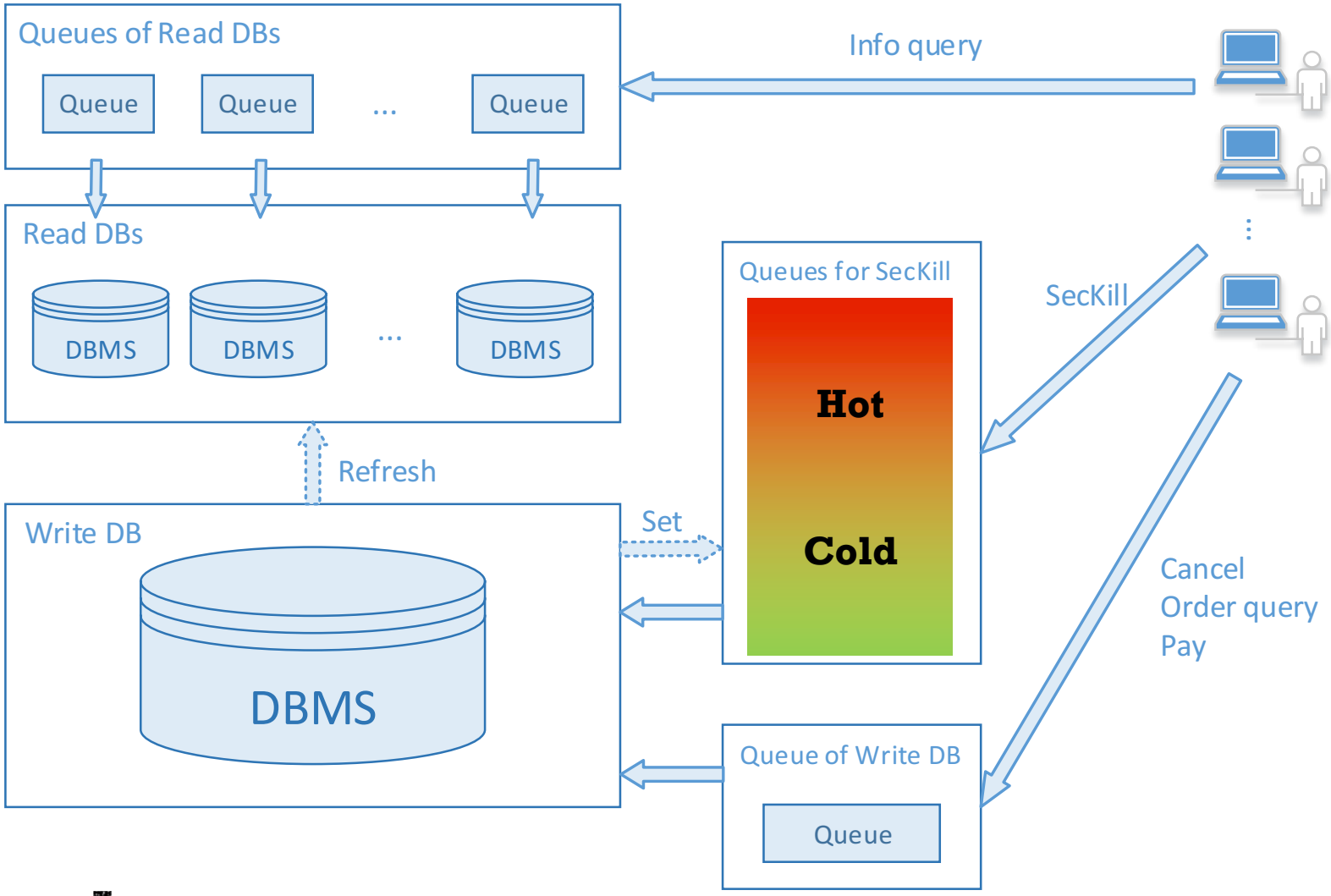
## Storage for Generality:

- Our tool can apply, run and test on different databases with general interfaces.

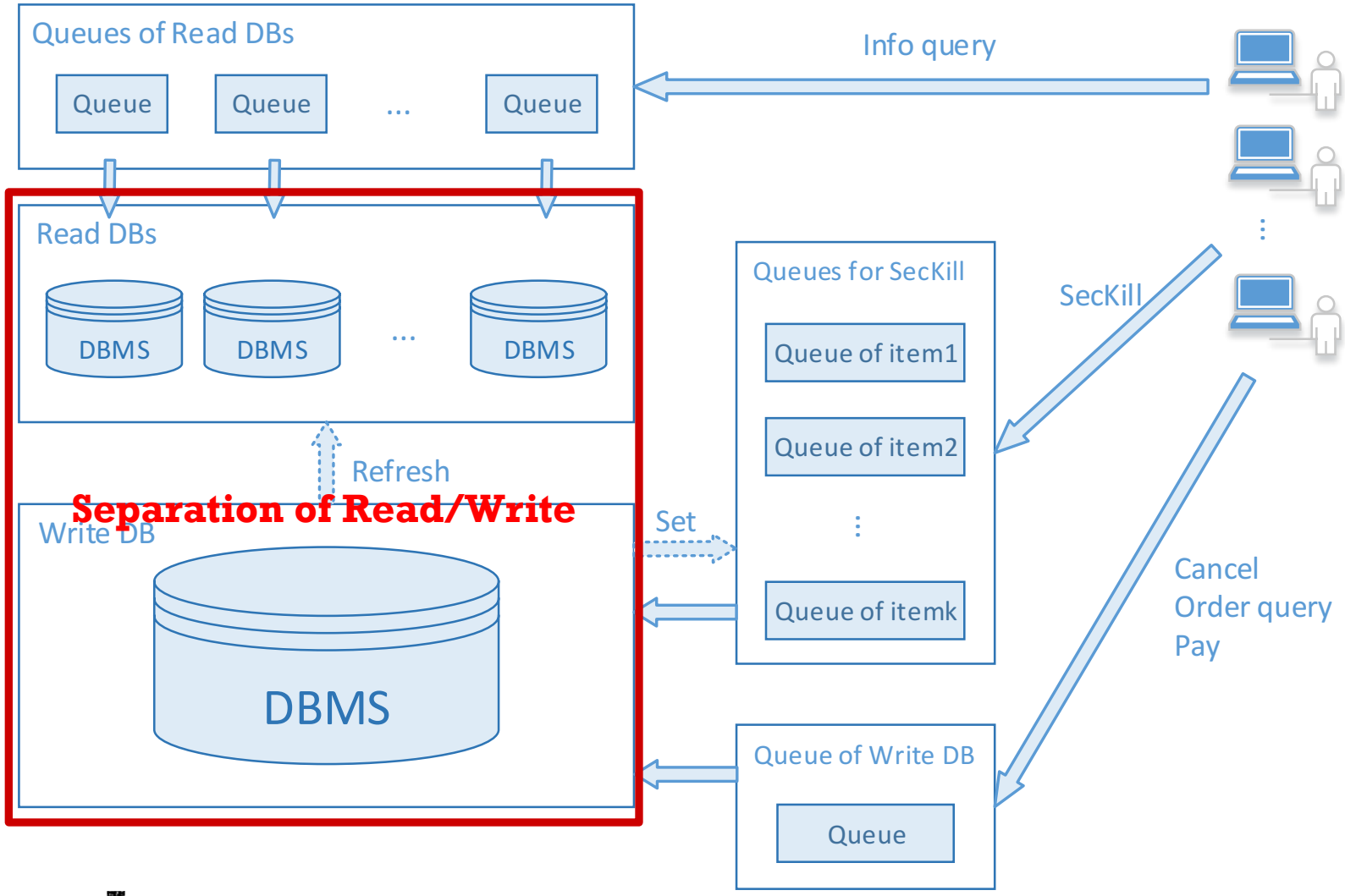
# PeakBench Architecture



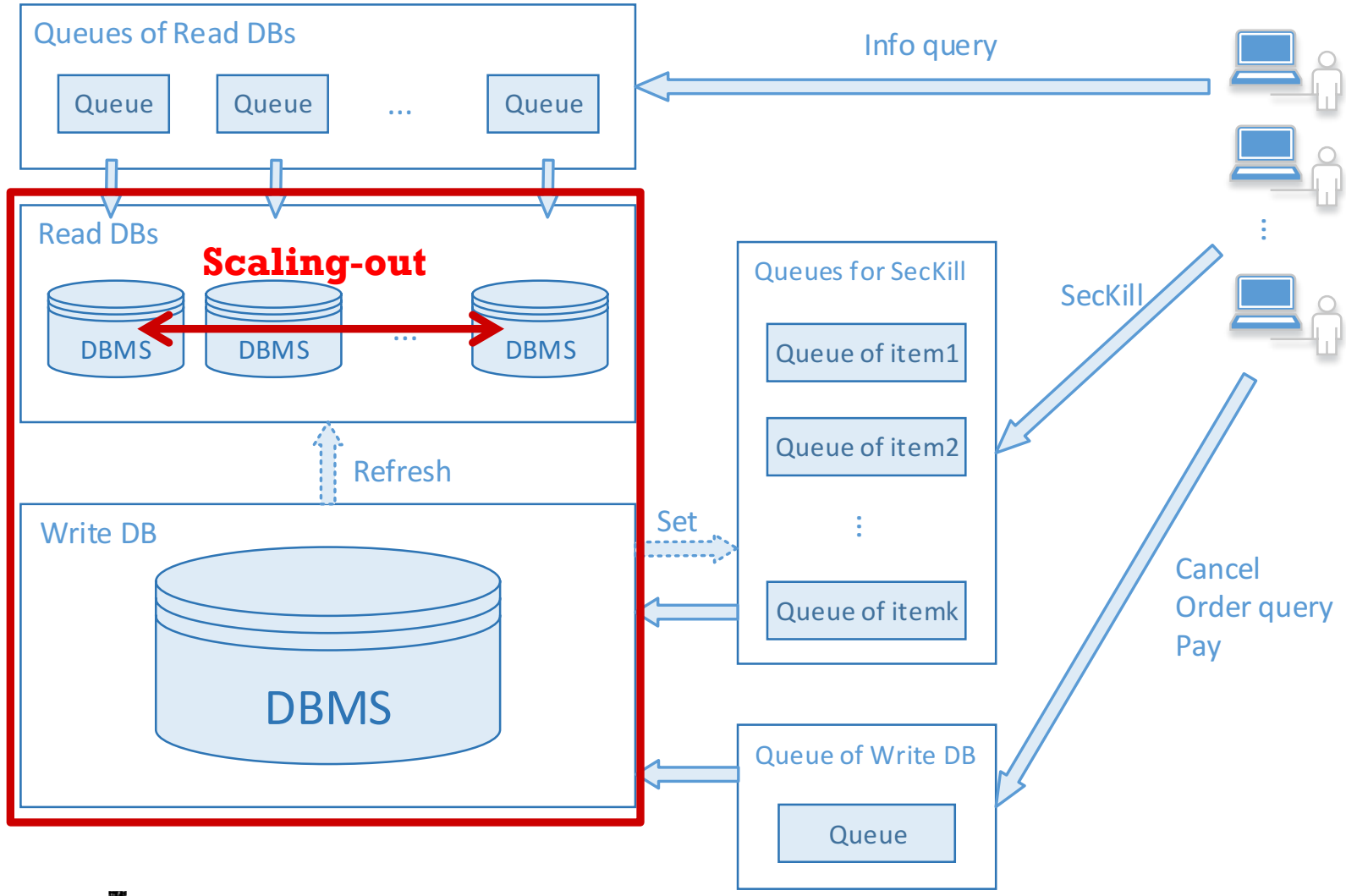
# PeakBench Architecture



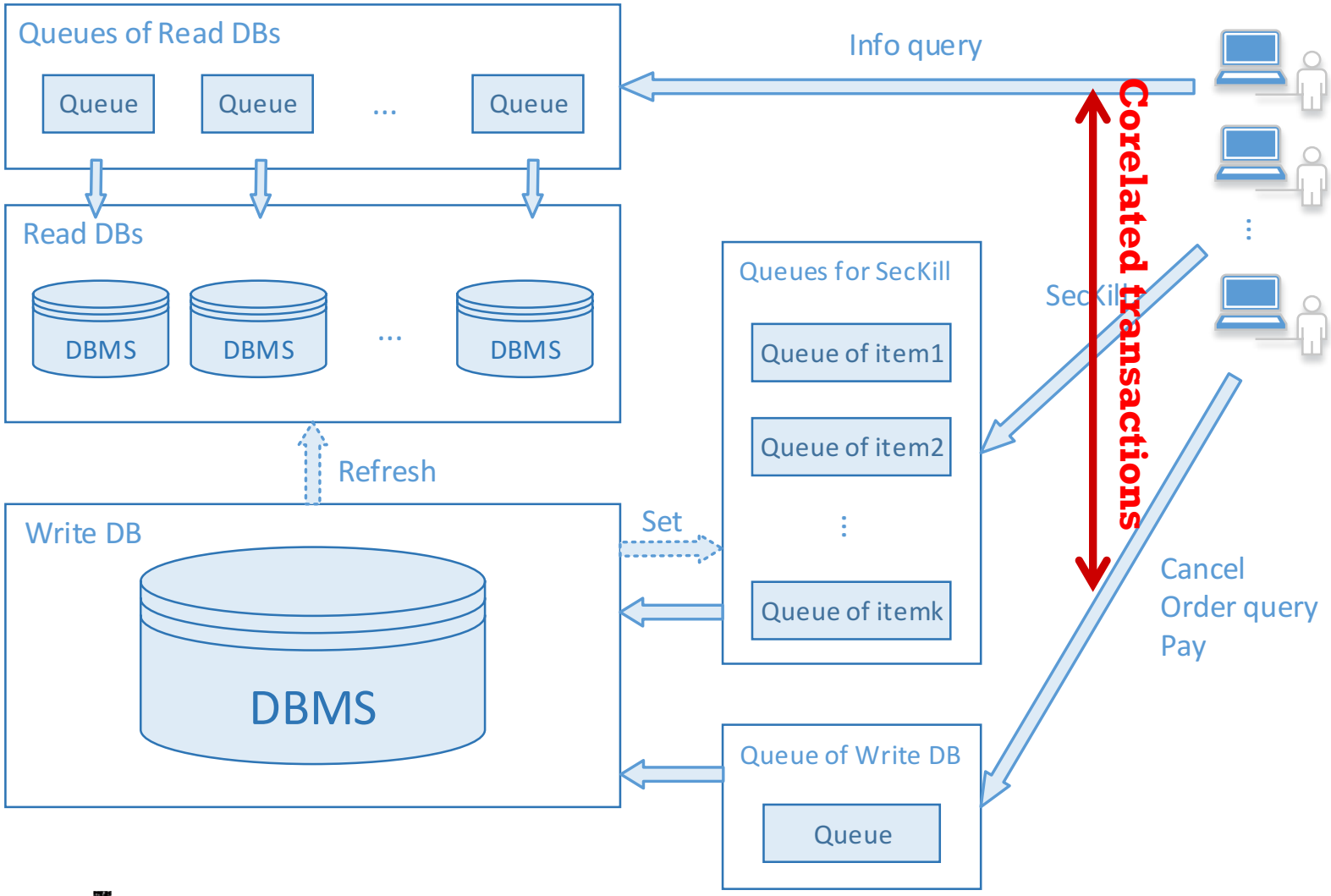
# PeakBench Architecture



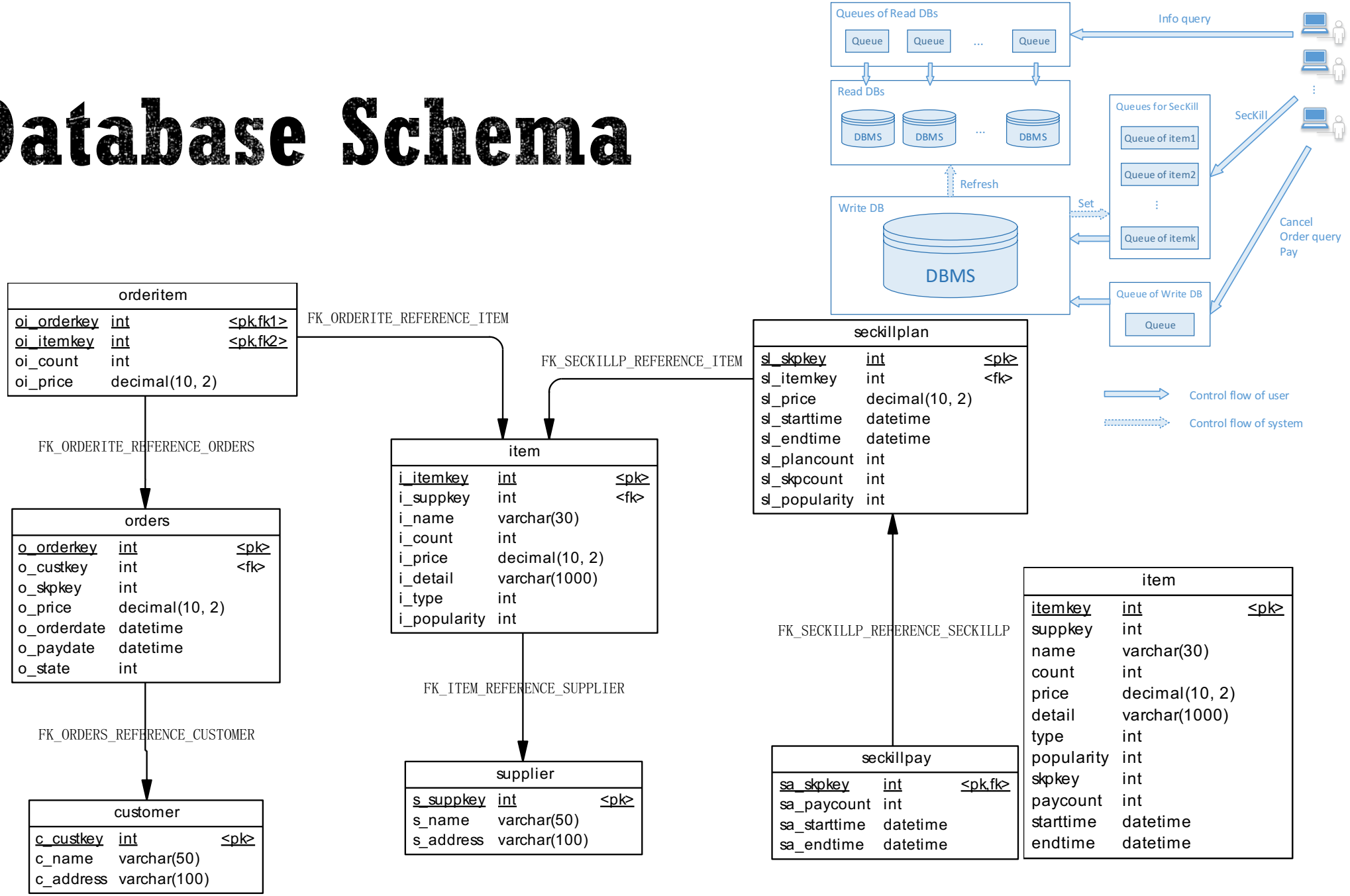
# PeakBench Architecture



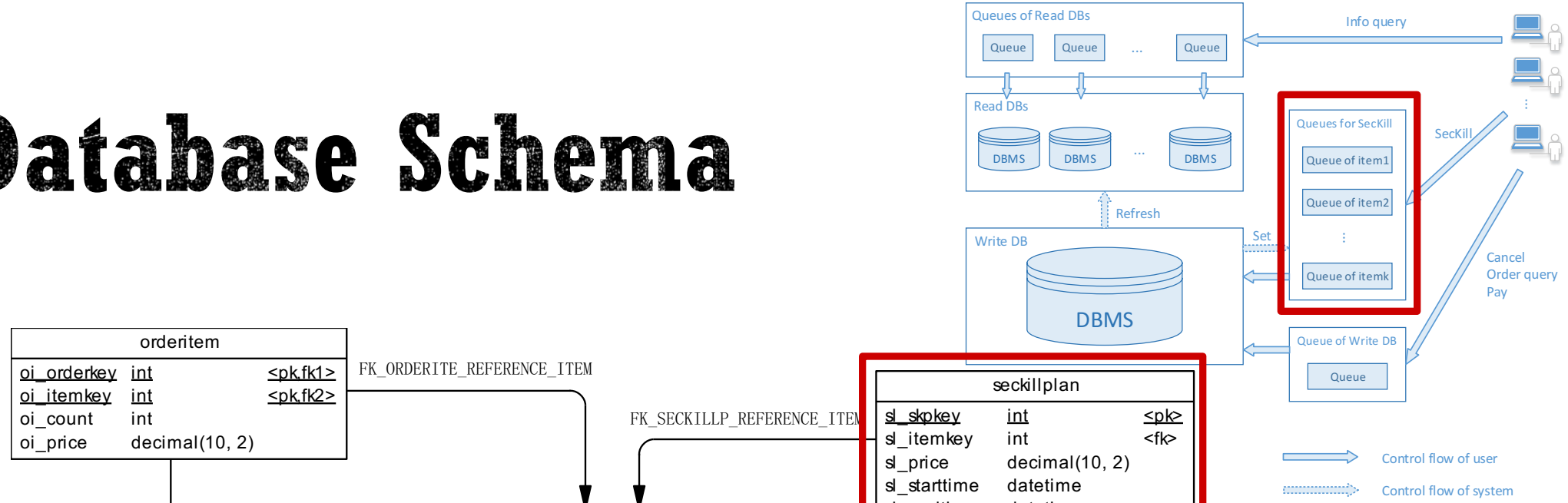
# PeakBench Architecture

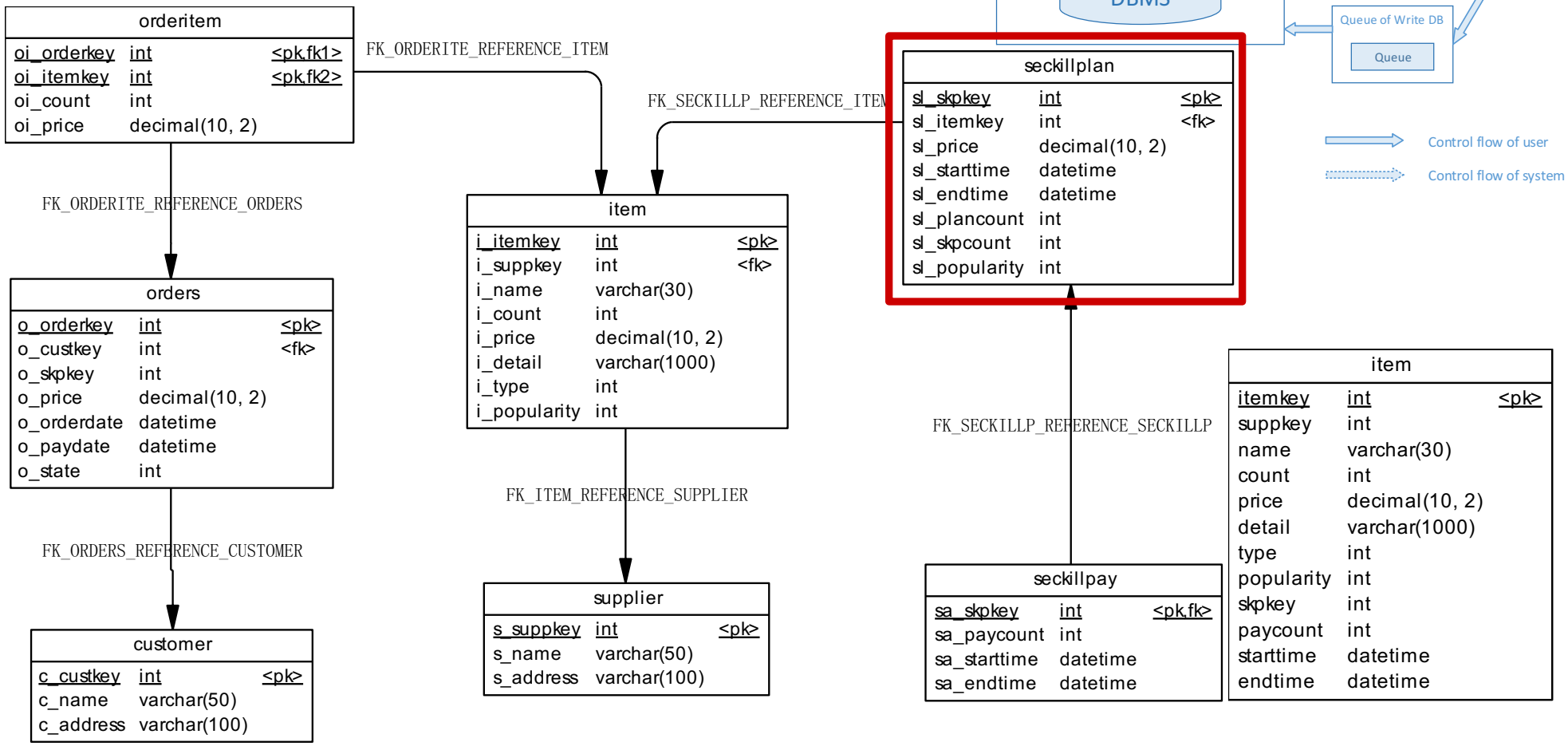


# Database Schema



# Database Schema

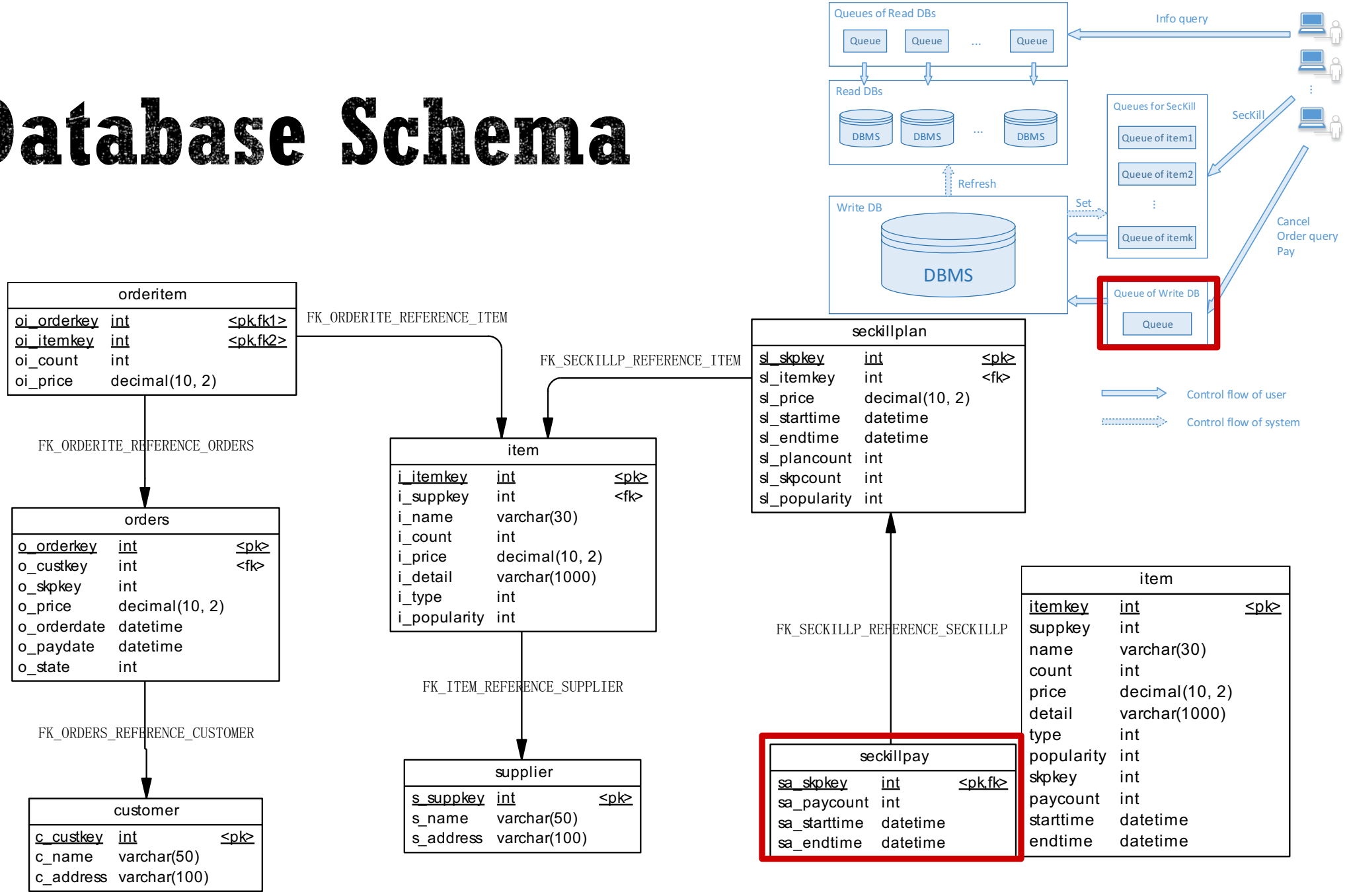



DB for Write

DB for Read

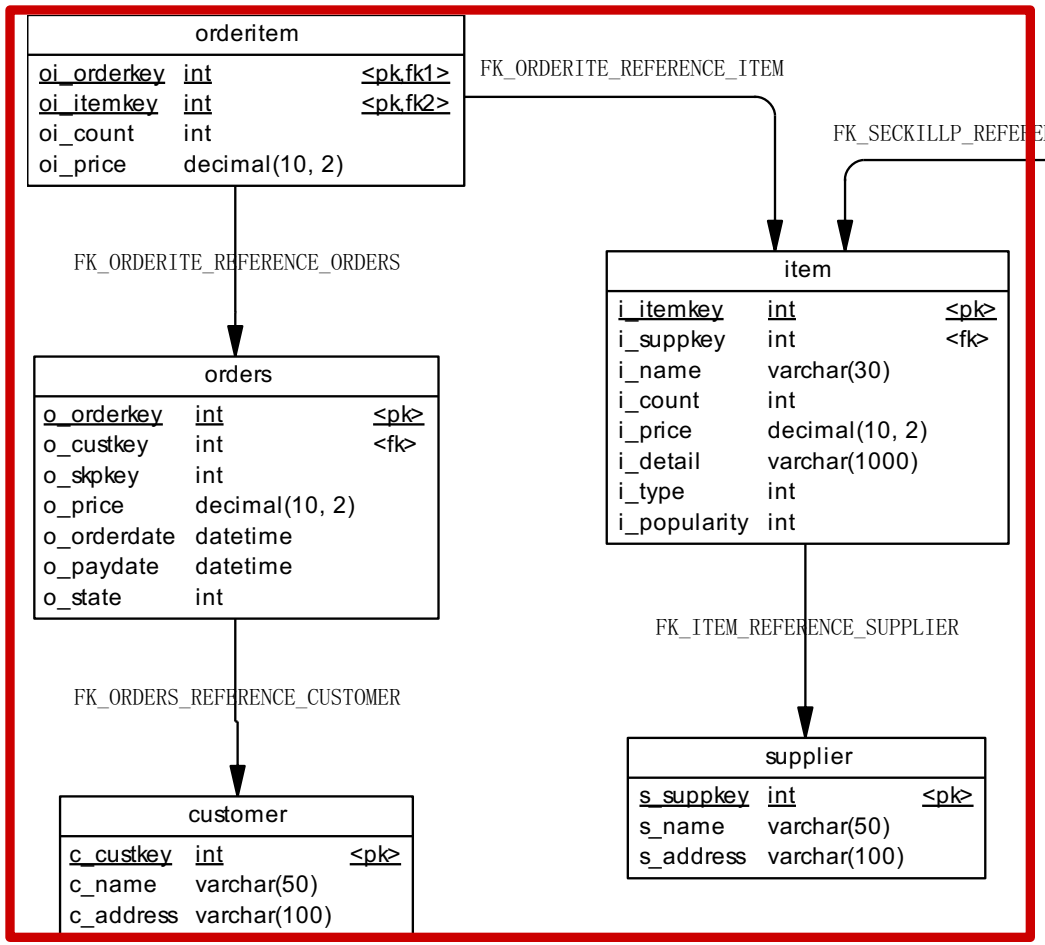
# Database Schema



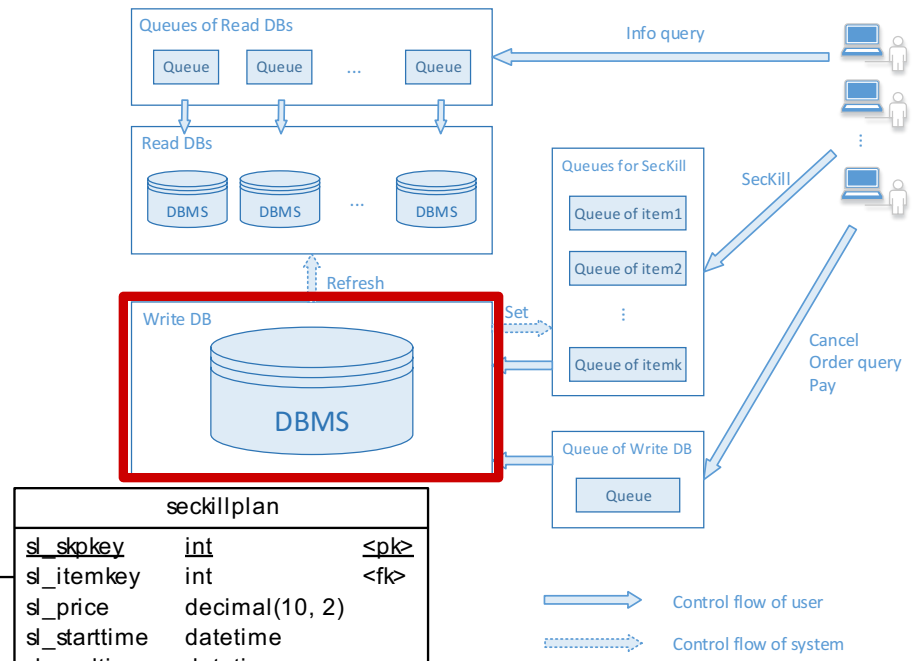

DB for Write

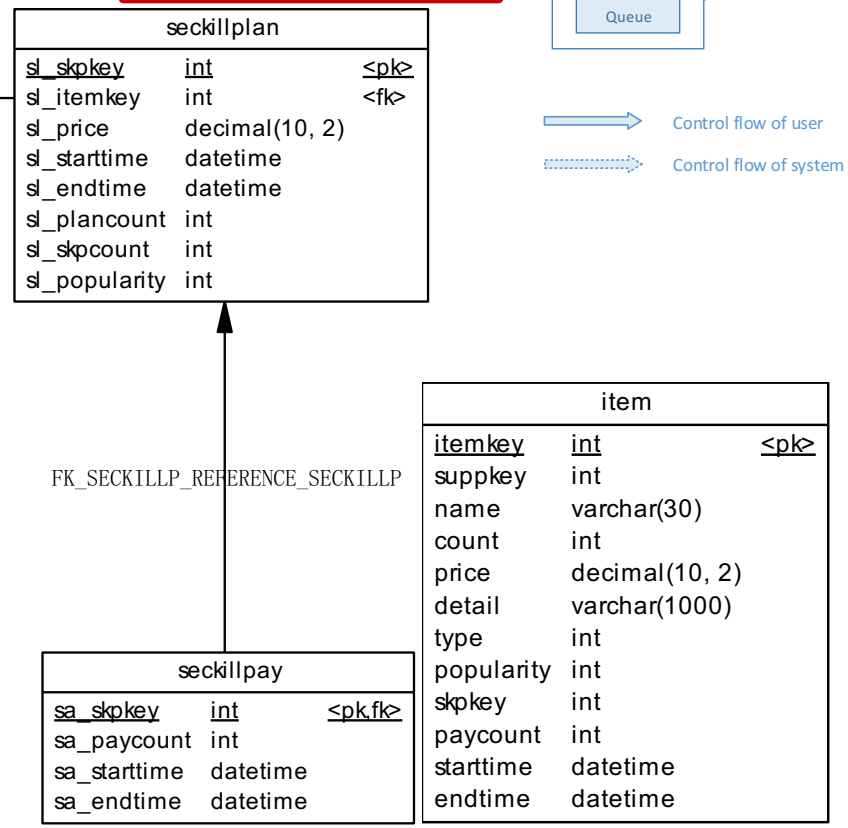
DB for Read

# Database Schema



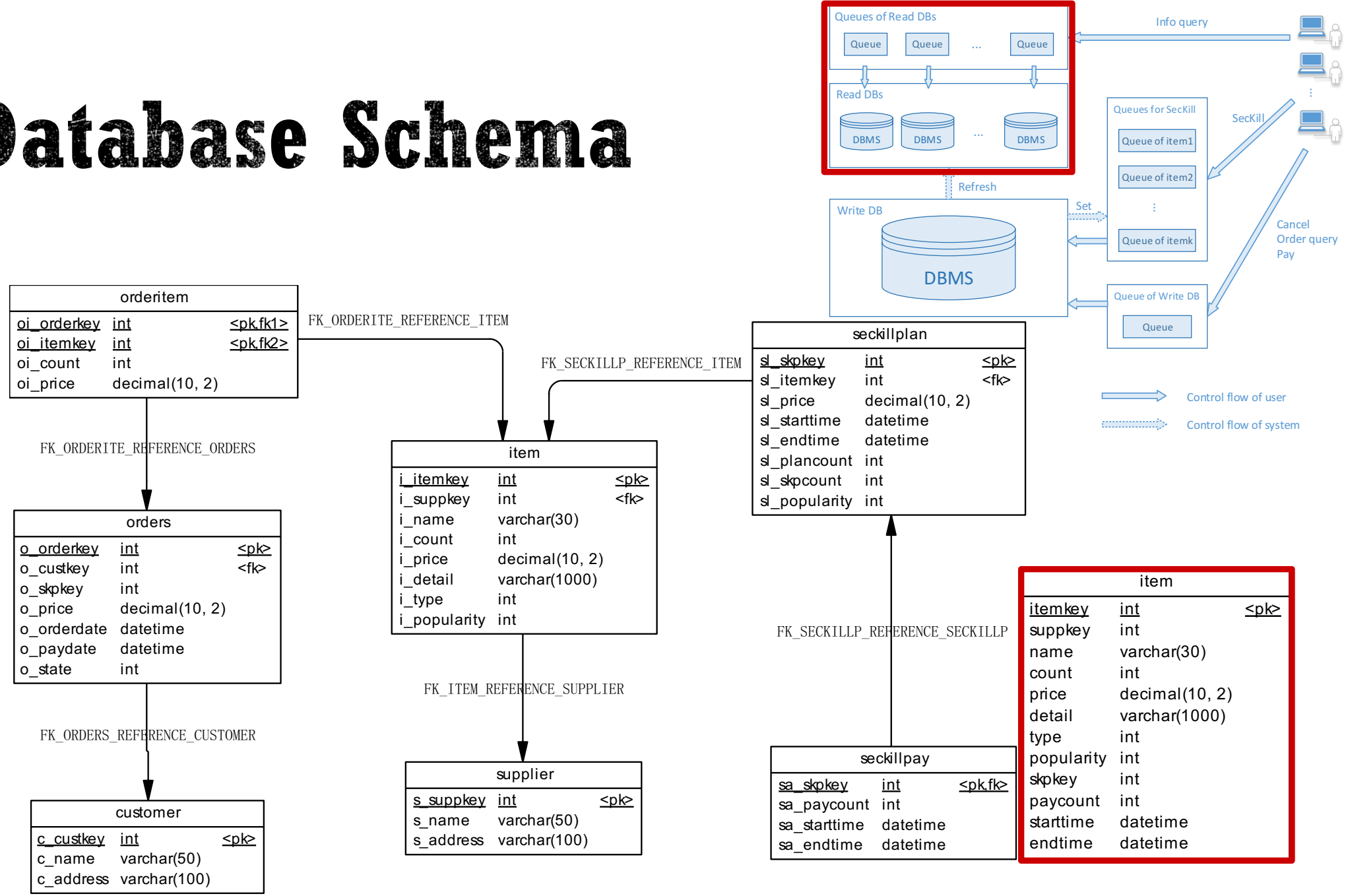

DB for Write




DB for Read

# Database Schema




DB for Write

DB for Read

# Workloads

## Non-SecKill Tx.

- Submit a transaction
- Payment
- Cancel a transaction
- Query a transaction
- Sync. the ReadDB with the WriteDB

## SecKill Tx.

- Submit a transaction
- Payment
- Cancel a transaction
- Query a transaction
- Expiring a SecKill transaction
- Sync. the ReadDB with the WriteDB
- Clean the SecKill DBs

# Workload

- Read Workload
  - Customers can search based on item id (Q1), item categories (Q2), keywords (Q3), range of prices (Q4) or shops for viewing the detailed item information (Q5).
  - Q6 is applied only on R/W separation architecture model for synchronizing between RDB and WDB guaranteeing consistency.
  - We create the following secondary index table on the Item (or RItem) table to improve query efficiency and optimize query response delay.

# Workload

- Update workload
  - Submit Order Transaction
  - Pay Order Transaction
  - Cancel Order Transaction
  - Select Order Transaction
  - Overdue Order Transaction
- Our workloads are all abstracted from Alibaba's actual seconds kill business.

# Workload Generation

- Set 20% items in the Table Item_ID as hot items, while other 80% ones are common items.
- 80% workloads on those 20% hot items are hot ones.
- Workloads satisfies Zipfian distribution.
- Warm the ReadDB 5min before the SecKill. Throw the peak workload to the ReadDB when SecKill begins, and then decay it (update every 5sec).
- The WriteDB is only accessed after the SecKill begins. The distribution of workload is similar to that for the ReadDB.

# Measurements

- QPS
- TPS
- Stability

## Some results

1

| item表规模 | mysqld CPU | QPS | Update time | avg time1 | avg time2 | avg time5 |
|---|---|---|---|---|---|---|
| 200万 | 300%-400% | 4222 | 196 ms | 3218 us | 6222 us | 5926 us |
| 500万 | 300%-400% | 4216 | 195 ms | 3214 us | 6324 us | 5988 us |
| 1000万 | 300%-400% | 4128 | 200ms | 3216us | 6485us | 6022us |
| 2000万 | 300%-400% | 4189 | 196 ms | 3321 us | 7155 us | 6010 us |

2

| 更新周期 | mysqld CPU | QPS | Update time | avg time1 | avg time2 | avg time5 |
|---|---|---|---|---|---|---|
| 0.5s | 450%-550% | 4103 | 207ms | 3252us | 6498us | 6049us |
| 1s | 400%-500% | 4115 | 198ms | 3150us | 6556us | 6087us |
| 2s | 300%-400% | 4128 | 200ms | 3216us | 6485us | 6022us |

3

| 每次更新数据量 | mysqld CPU | QPS | Update time | avg time1 | avg time2 | avg time5 |
|---|---|---|---|---|---|---|
| 0 | 300%-400% | 4135 | | 3213 us | 6496 us | 6044us |
| 500 | 300%-400% | 4133 | 110ms | 3213us | 6475us | 6026us |
| 1000 | 400% - 500% | 4128 | 200ms | 3216us | 6485us | 6022us |
| 1500 | 400% - 500% | 4118 | 291ms | 3229 us | 6489 us | 6028 us |
| 2000 | 400% - 500% | 4124 | 387ms | 3238 us | 6473 us | 6033 us |
| 2500 | 400% - 500% | 4117 | 479ms | 3243us | 6484us | 6050us |
| 3000 | 400% - 500% | 4113 | 573ms | 3258 us | 6493 us | 6048 us |
| 5000 | 550% - 650% | 4100 | 954ms | 3252us | 6489us | 6060us |
| 10000 | 已无法完成数据更新 | | | | | |

4

| 读负载类型 | mysqld CPU | QPS | Update time | avg time1 | avg time2 | avg time3 | avg time4 | avg time5 |
|---|---|---|---|---|---|---|---|---|
| 三种 | 400% - 500% | 4128 | 198ms | 3216us | 6485us | | | 6022us |
| 四种 | 400% - 500% | 4115 | 199ms | 3229 us | 6405 us | | 10571 us | 5967 us |
| 五种 | 500% - 600% | 4184 | 187ms | 2916us | 5530us | 55033us | 9852us | 5165us |

5

| 读负载线程数 | mysqld CPU | QPS | avg time1 | avg time2 | avg time5 |
|---|---|---|---|---|---|
| 2 | 100% - 200% | 2105 | 388us | 1522us | 1417us |
| 5 | 250% - 350% | 3448 | 635us | 2241us | 2073us |
| 10 | 300% - 400% | 4071 | 1509us | 3287us | 3095us |
| 20 | 300% - 400% | 4132 | 3210us | 6462us | 6054us |
| 30 | 300% - 400% | 4139 | 4574us | 9953us | 9246us |
| 50 | 300% - 400% | 4081 | 5460us | 18565us | 16978us |
| 100 | 300% - 400% | 4149 | 6739us | 35342us | 31965us |
| 200 | 300% - 400% | 4140 | 8089us | 85098us | 76409us |
| 300 | 300% - 400% | 4146 | 9480us | 121549us | 108623us |

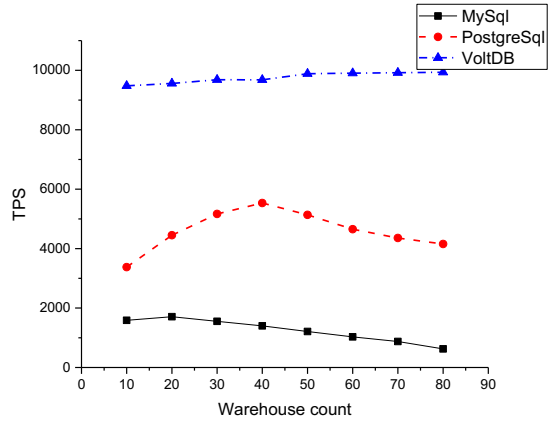
# Benchmarking MySQL, PostgreSQL and VoltDB

- Clusters with 8 nodes configured in RAID-5 on CentOS v.6.5, which are equipped with 2 Intel Xeon E5-2620 @ 2.13 GHz CPUs, 130GB memory and 3 TB HDD disk.
- Database: MySQL14.14、PostgreSQL 9.5.1 and VoltDB 7.8.2.
- Configurations of PeakBench

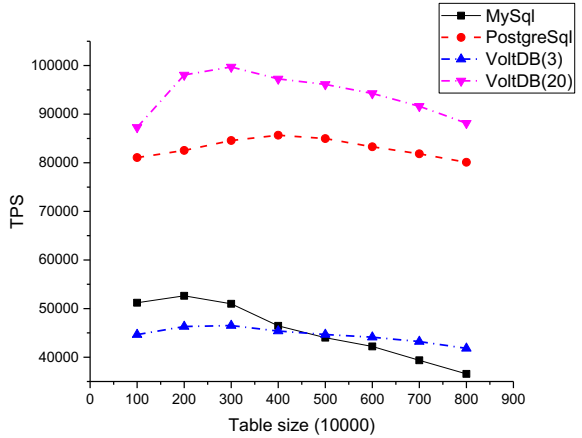
| Table | Size (tuples) |
|---|---|
| Customer | $8 \cdot 10^4$ |
| Supplier | $5 \cdot 10^2$ |
| Item | $10^5$ |
| Order | $4 \cdot 10^5$ |
| OrderItem | $8 \cdot 10^5$ |
| SecKill | $10^2$ |
| scaleFactor | 10 |
| skScaleFactor | 10 |

| Parameter | Value |
|---|---|
| CR | 50 % |
| CI | 5 |
| Connection Thread for MySql and PG | 80 |
| Connection Thread for VoltDb | 60 |
| Partition Number for VoltDB | 3 |
| *zipfian(s,N)* | s=2,N=10 |
| Submit Order size (SuO) | $10^6$ |
| $\alpha$ and $\beta$ | 0.5& 0.5 |

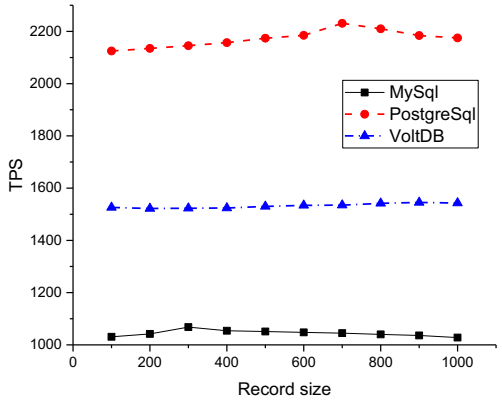
# Contention Controls



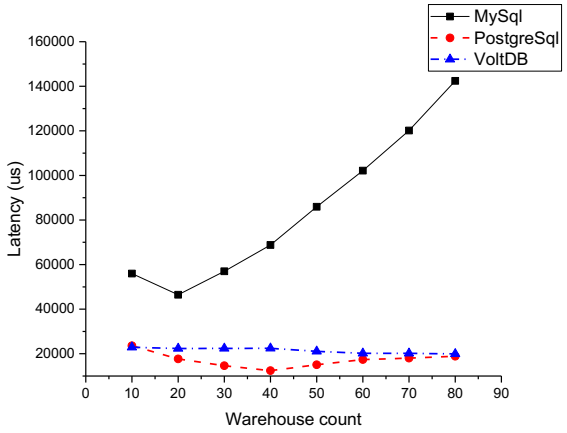

TPS@TPCC



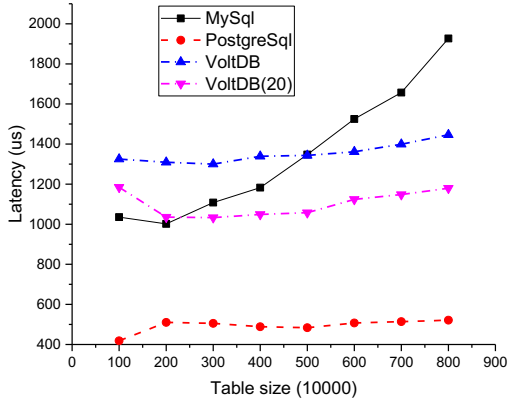

TPS@YCSB



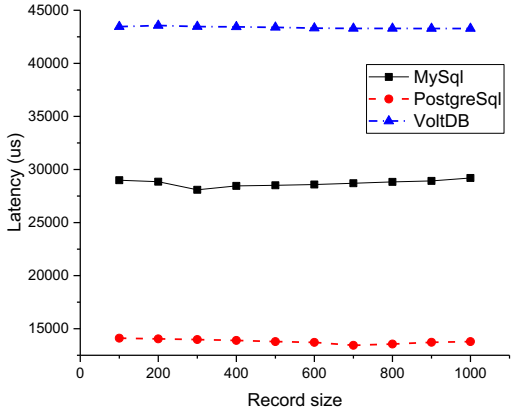

TPS@Smallbank




Latency@TPCC

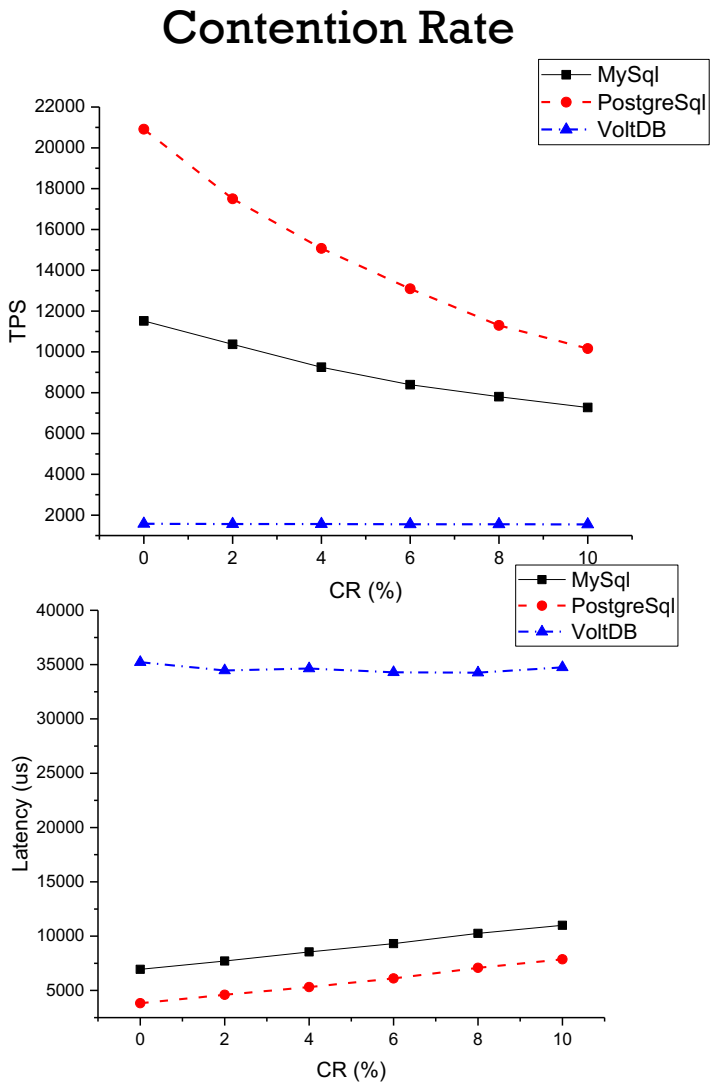


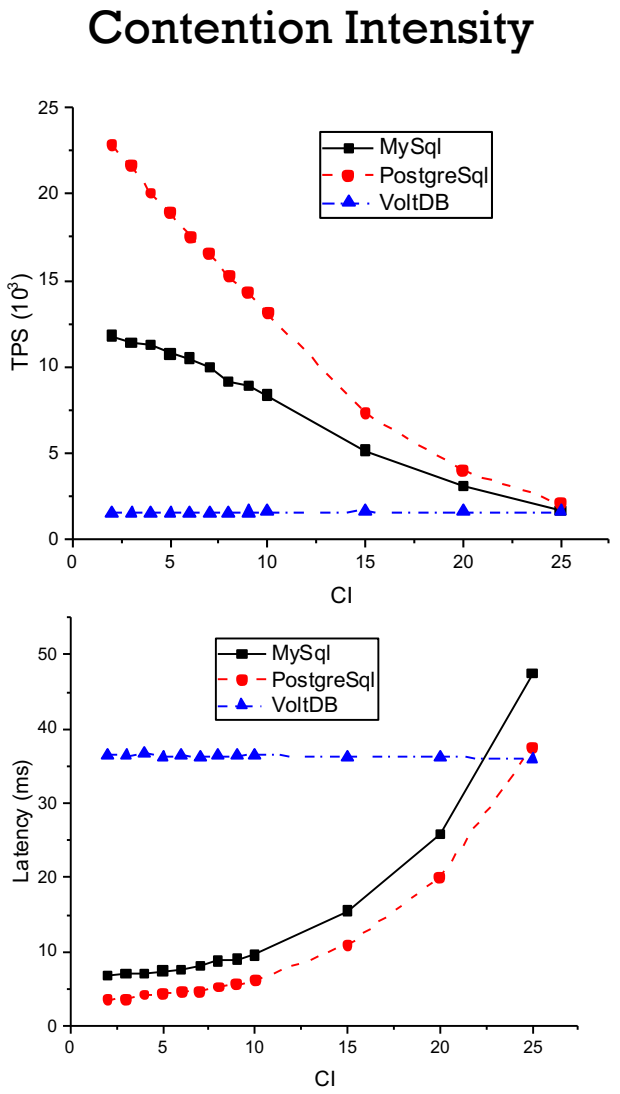

Latency@YCSB



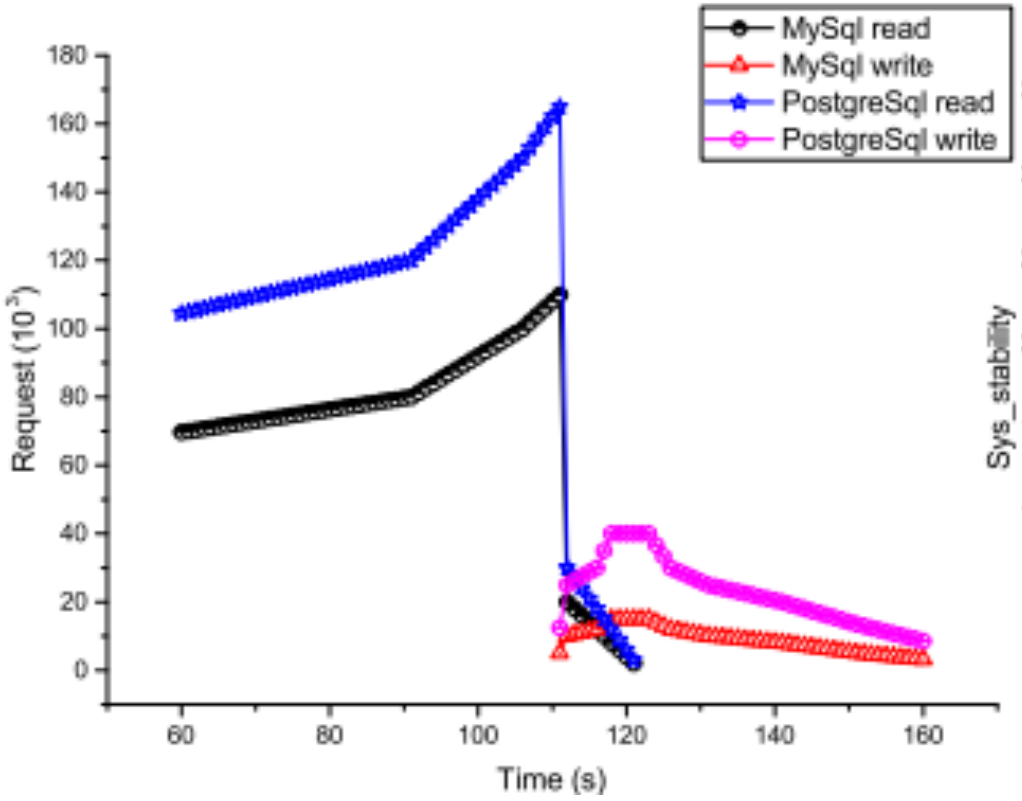

Latency@Smallbank

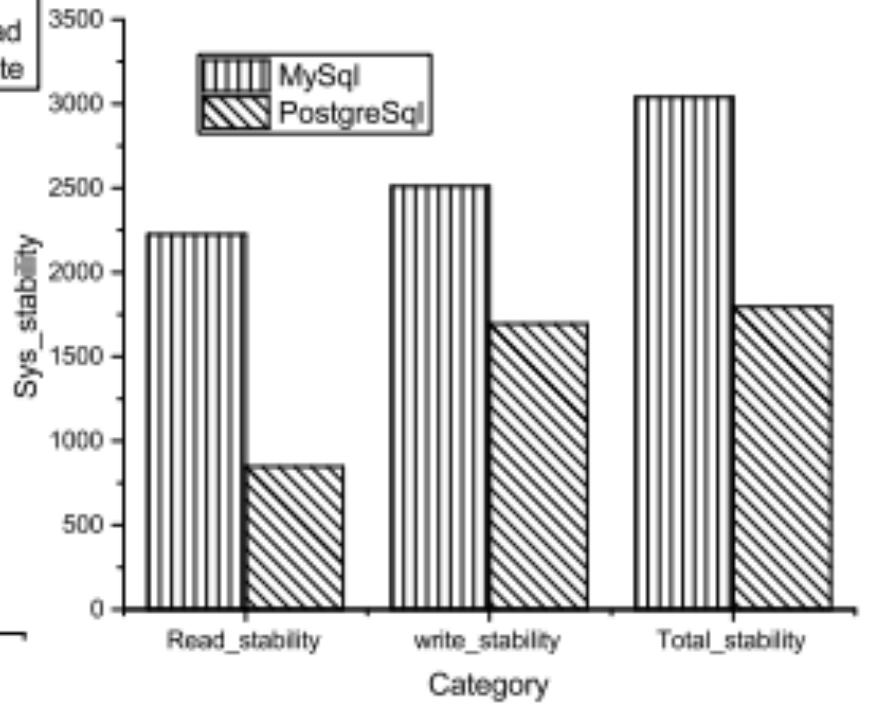
# Results: w. Different Contention Rate/Intensity



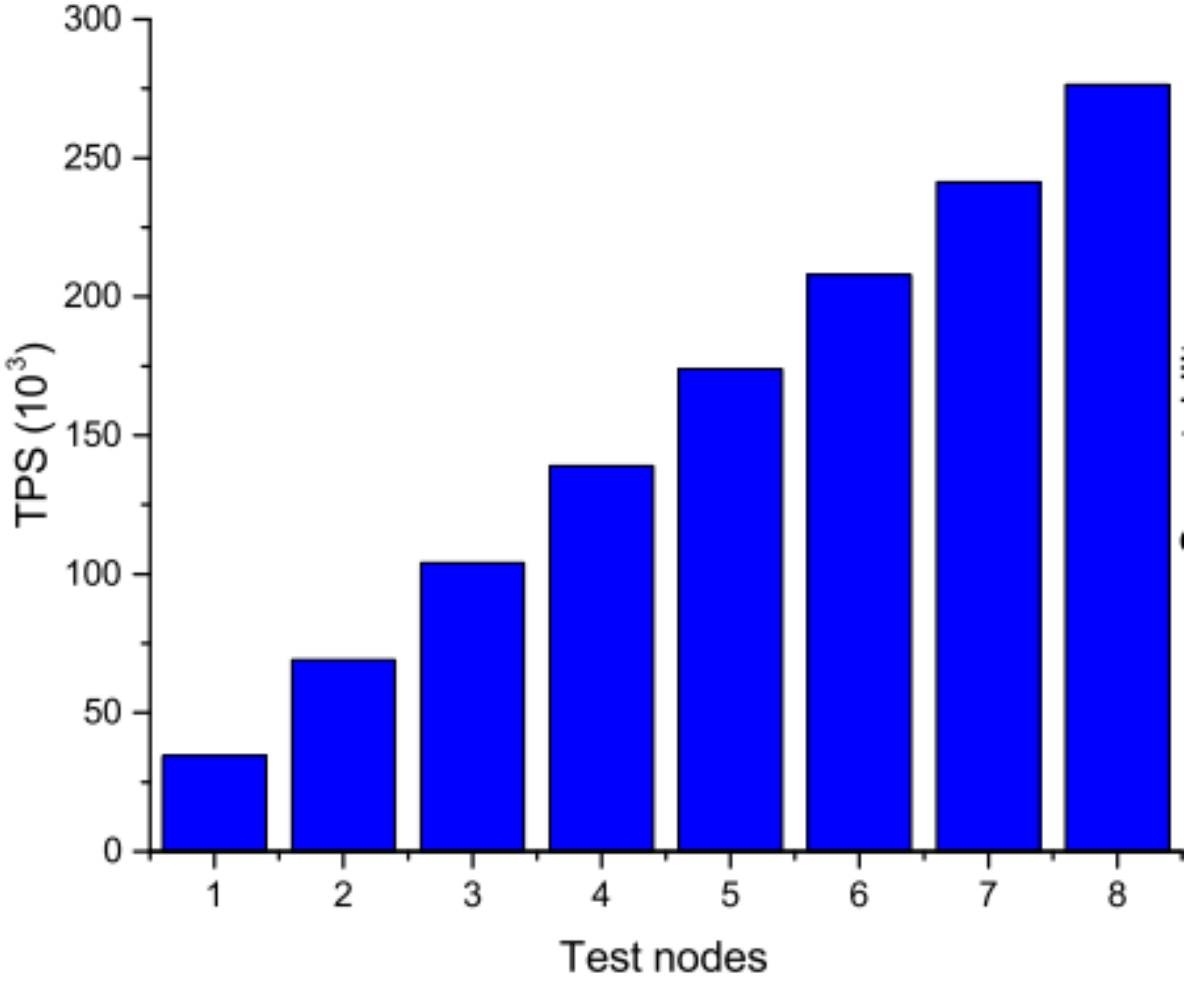
# Results: Dynamics and Stability



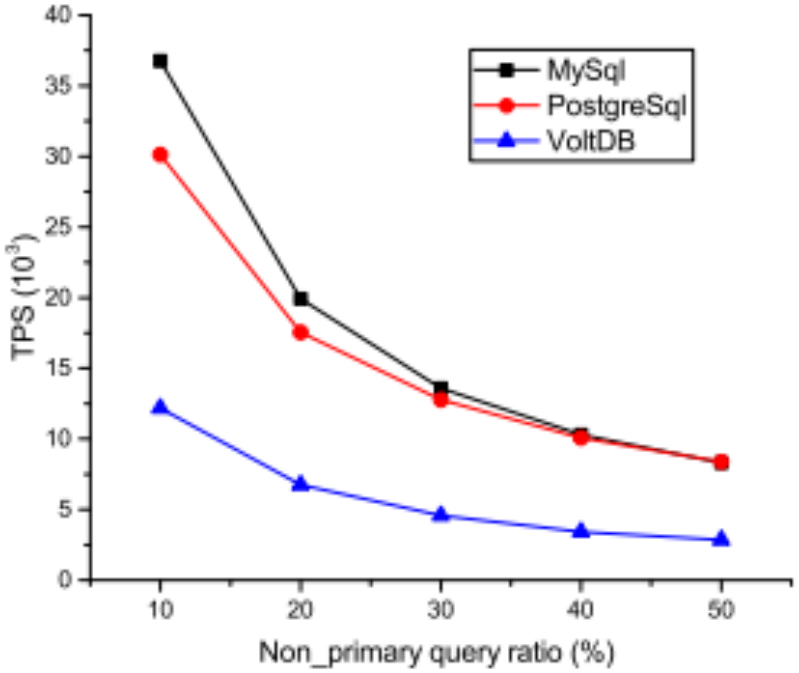




$$Sys\_Stability = \alpha \star \delta T \star \beta \star \delta L_c / \bar{L}_c, \alpha + \beta = 1.$$

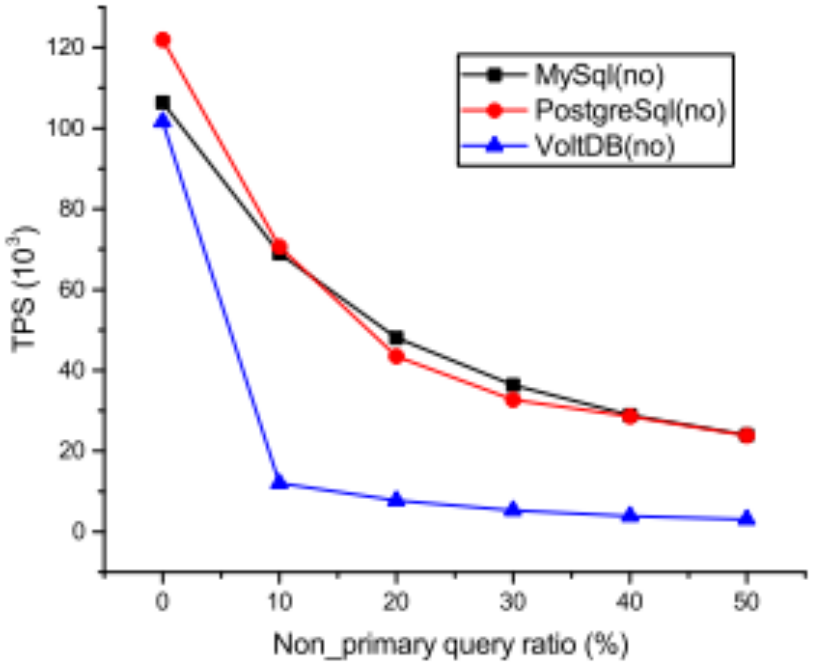
# Results: Scalable Generator



# Results: Read Workload

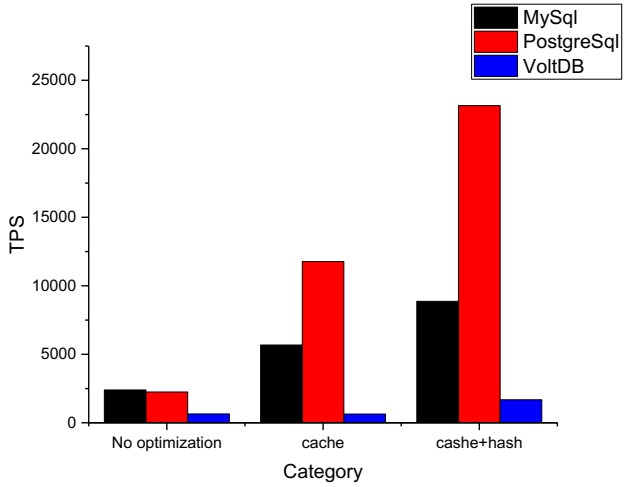


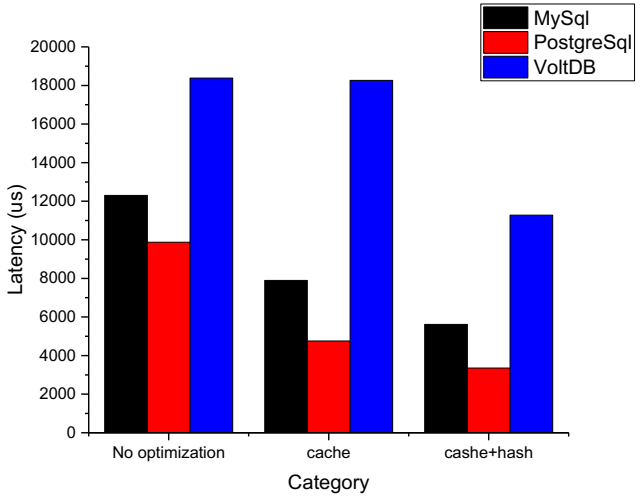

(a) $Q_1$-$Q_5$

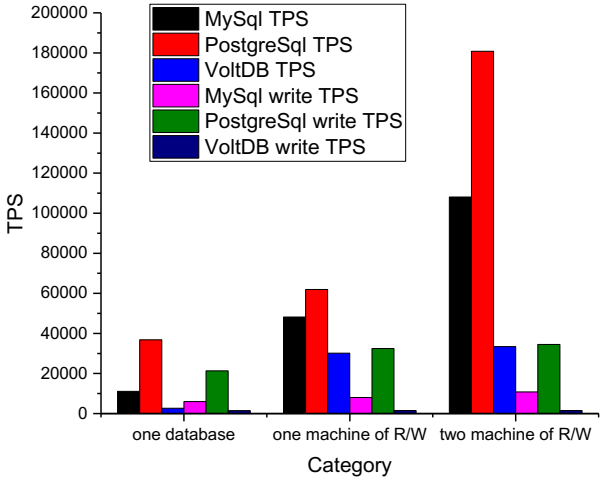


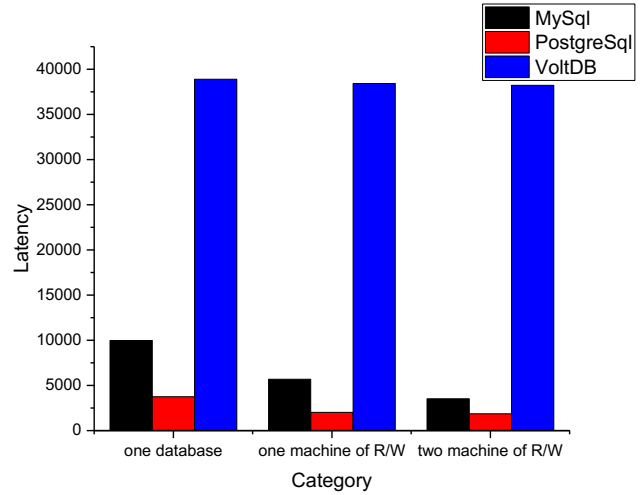

(b) Without $Q_3$ and $Q_4$

# Results: On Architecture of the Application



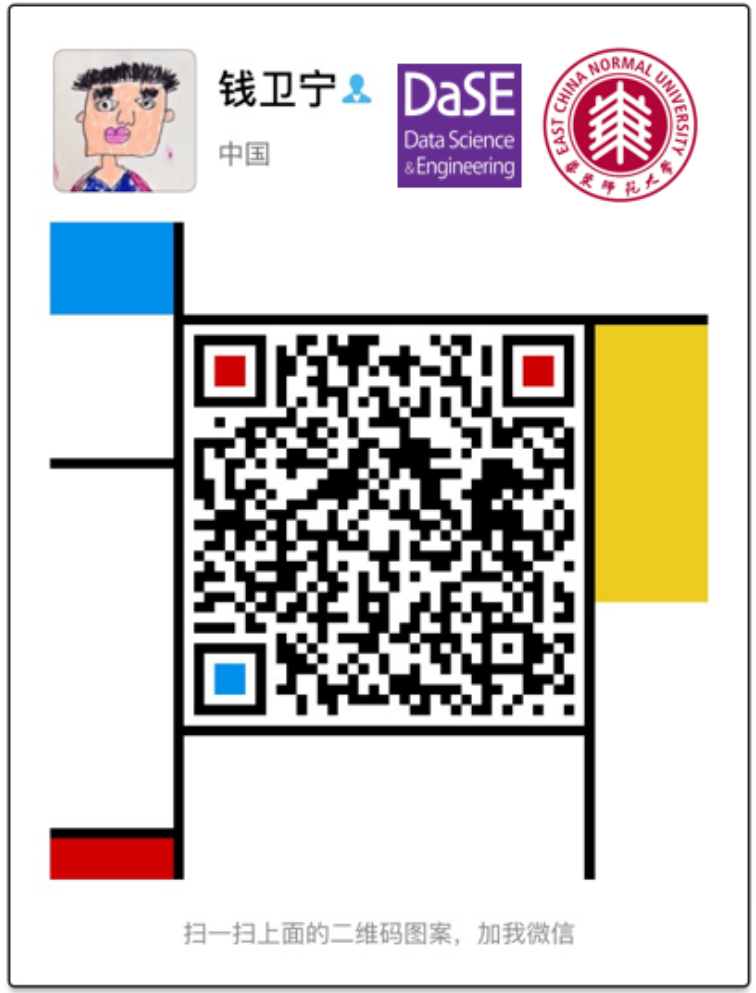



# Summary

- Scalable TP benchmark is needed for various new TP applications
- Not only for single systems, but also for applications/architectures
- We present a benchmark proposal for such applications
- https://github.com/daseECNU/DB-Testing

# Thanks !


钱卫宁
中国
DaSE
Data Science
& Engineering
EAST CHINA NORMAL UNIVERSITY
扫一扫上面的二维码图案，加我微信