




Achieving 25K+ QPS with Sub-100ms P99 latency in ClickHouse for Cost Savings Over ElasticSearch

for the ClickHouse Meetup, Oct-3, 2024, Singapore

Frank Chen, OLAP/Data Infrastructure/Shopee

09/21/2024

- 
- 1 ClickHouse in Shopee**
 - 2 Achievement**
 - 3 Business background**
 - 4 How we achieved it**
 - 5 Future plan**

Typical uses cases of ClickHouse in Shopee

User Behaviour
Analytics

Video Quality
Analytics

Application
Performance Analytics

Anti Fraud Detecting

Realtime Data
warehouse

Business Intelligence

MySQL Replacement

Log Storage

Chatbot
(Vector Search)

...

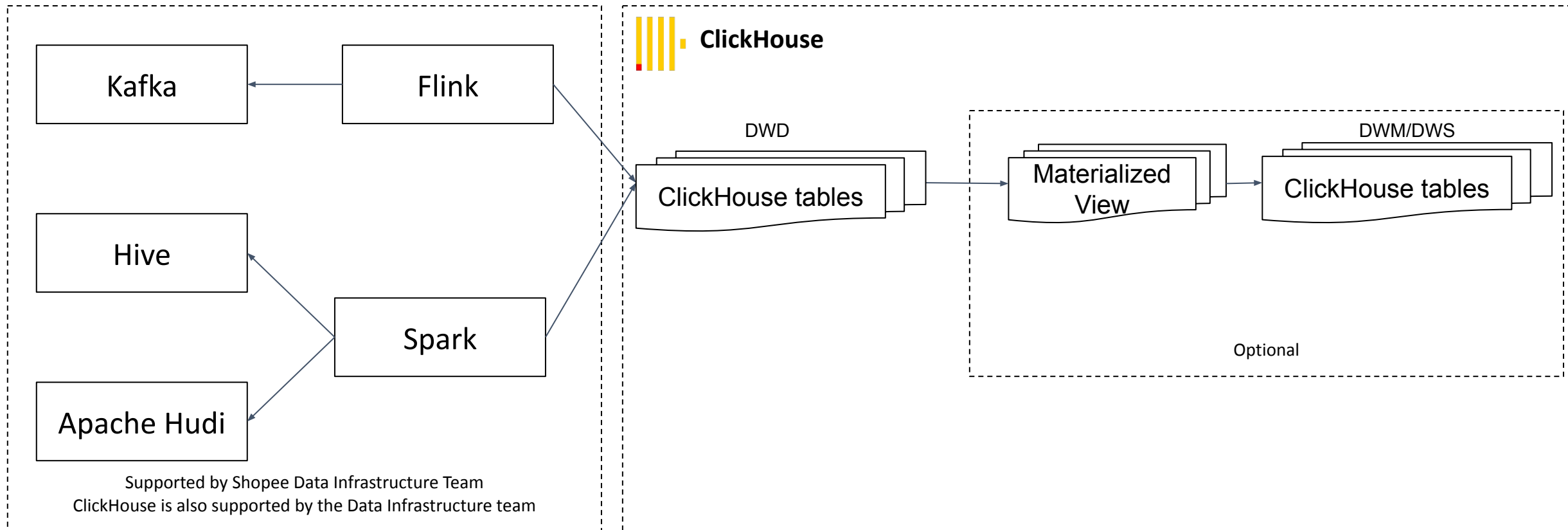


Typical use case – Real Time Data warehouse

Realtime ingestion - Flink

- KafkaEngine is not recommended to use due to
 - Scalability - Scaling out Flink jobs is much easier than scaling out KafkaEngine tables
 - Stability - Resource isolation, KafkaEngine consumes ClickHouse resources

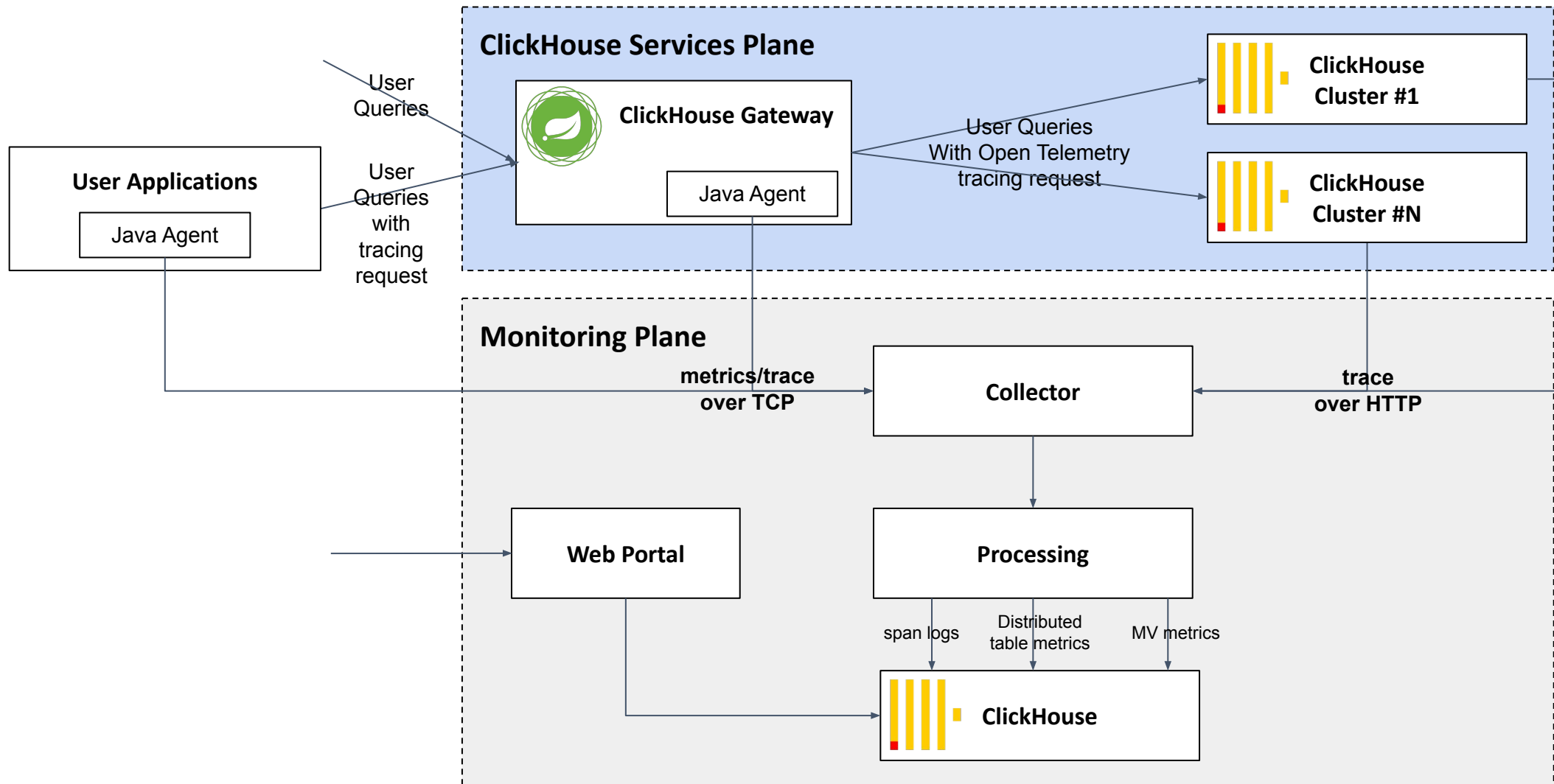
Batch ingestion - Spark





Typical use case – Observability

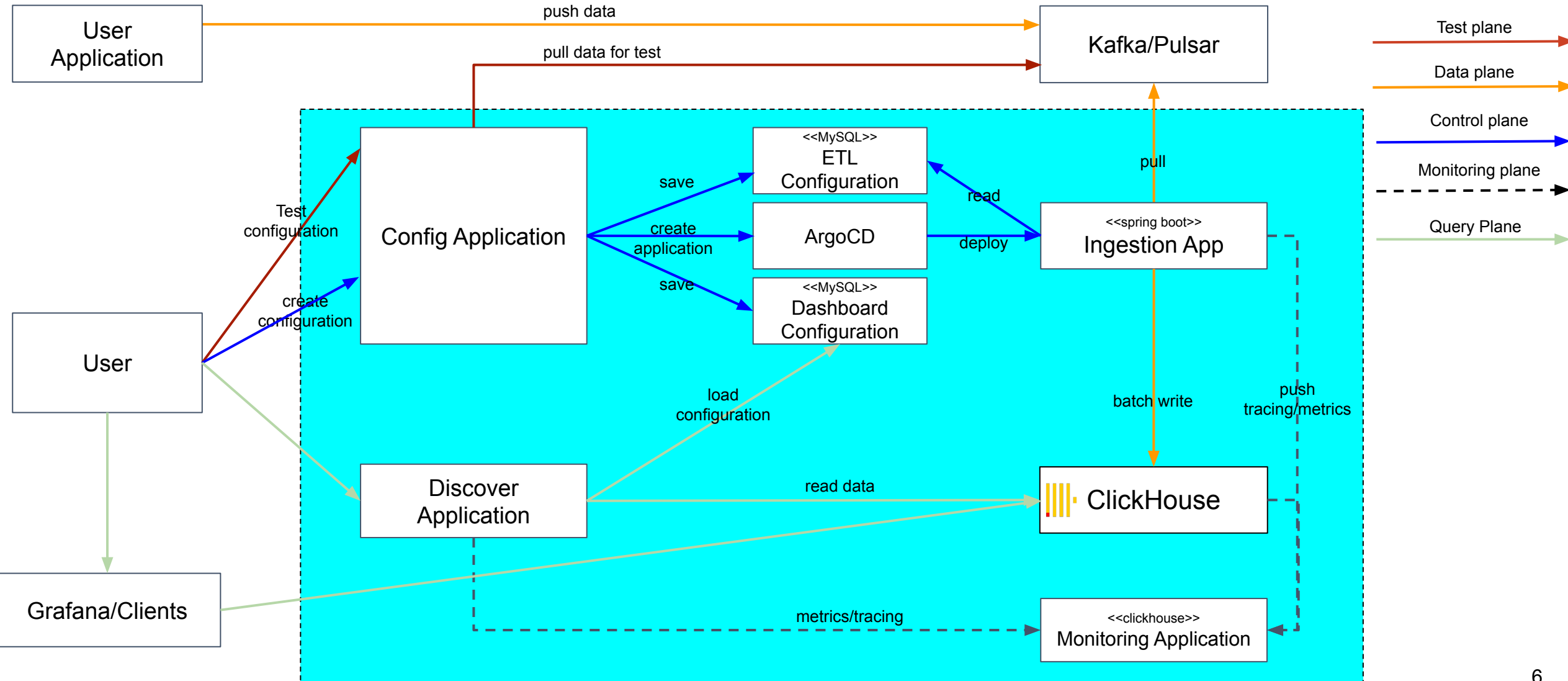
- [Distributed Tracing in ClickHouse](#), Presented in the ClickHouse Meetup @ Shopee on 18th, April, 2024, Singapore
- https://youtu.be/_BVy-V2wy9s





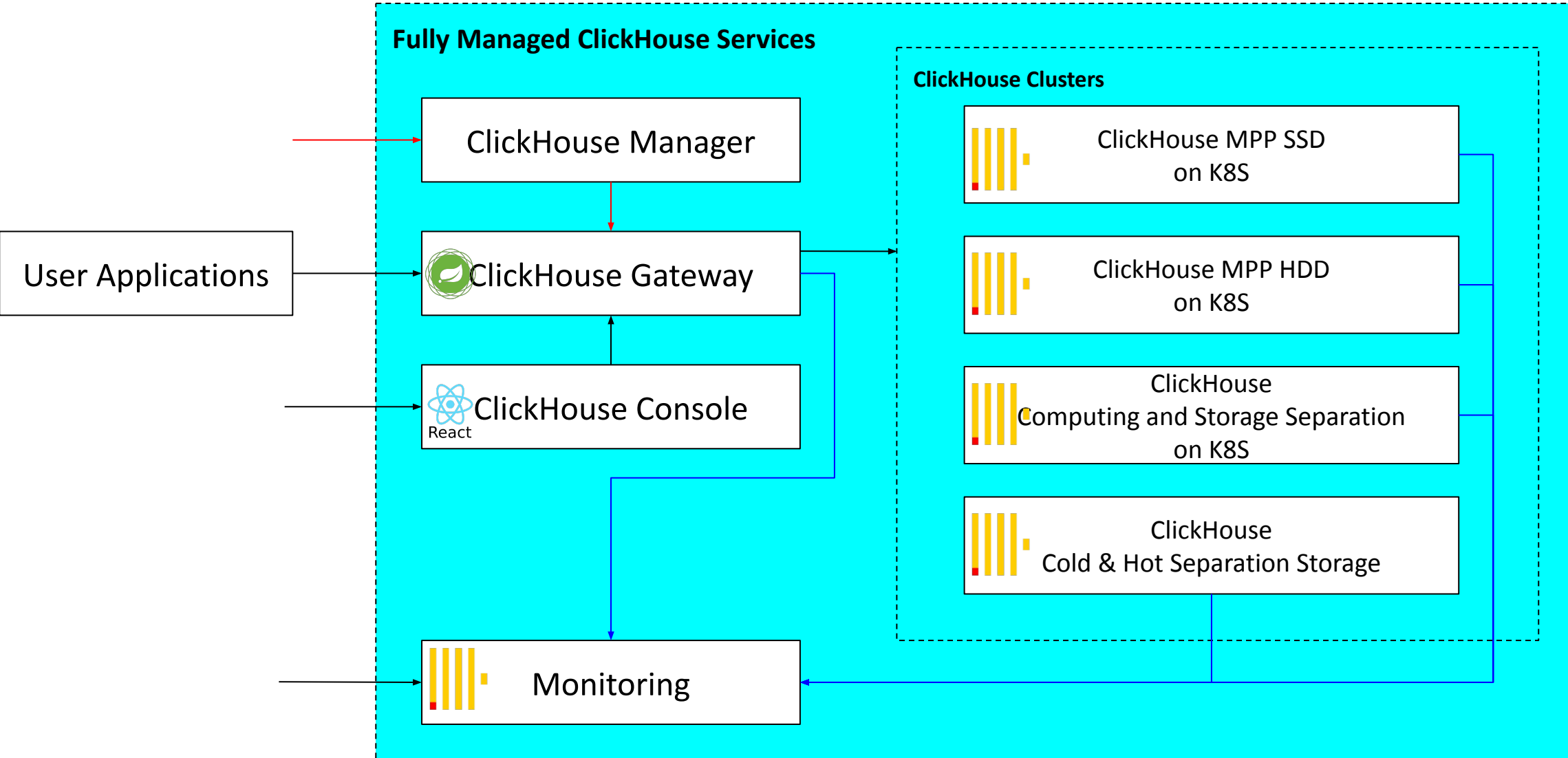
Typical use case – Log storage to Replace ElasticSearch


- How we build PB level log solution on ClickHouse
- Presented in the Database Meetup organized by OceaBase on 10th-Sep, 2024 in Singapore



Architecture of Shopee ClickHouse Service

control plane
data plane
monitor plane



- 
- 1 ClickHouse in Shopee
 - 2 Achievement**
 - 3 Business Background
 - 4 How we achieve it
 - 5 Future plan



Achievement (based on 23.3 LTS)

25K+ QPS
in a single cluster,
10x of other 100+ clusters

50K QPS
at peak

< 15 ms P50 Latency

<100 ms P99 Latency

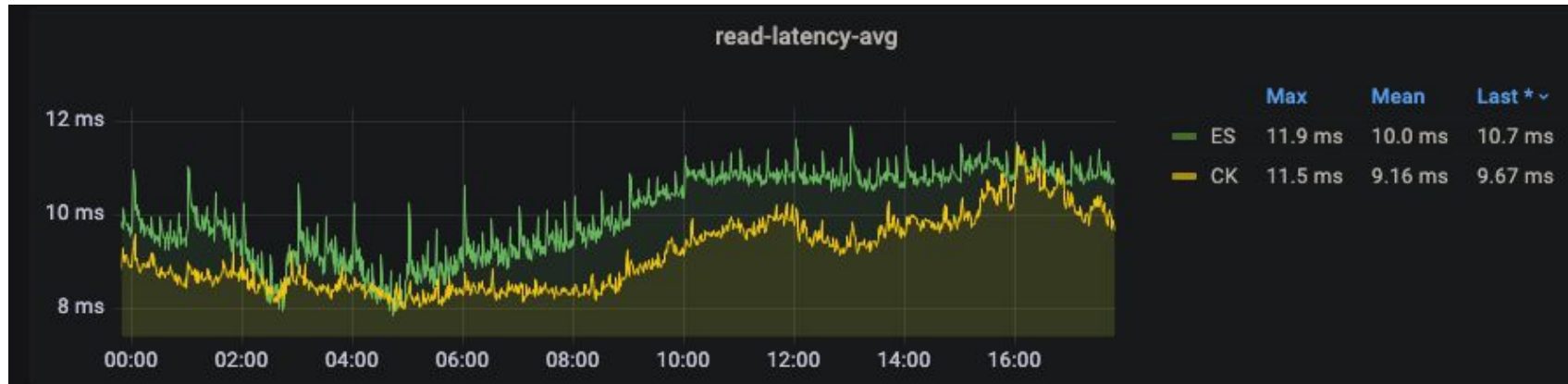
72% CPU Reduced

50% Disk Reduced



Reduced P99 query latency about 56% over ES

P50 Read Latency reduced about **10%**



P99 Read Latency reduced about **56%**





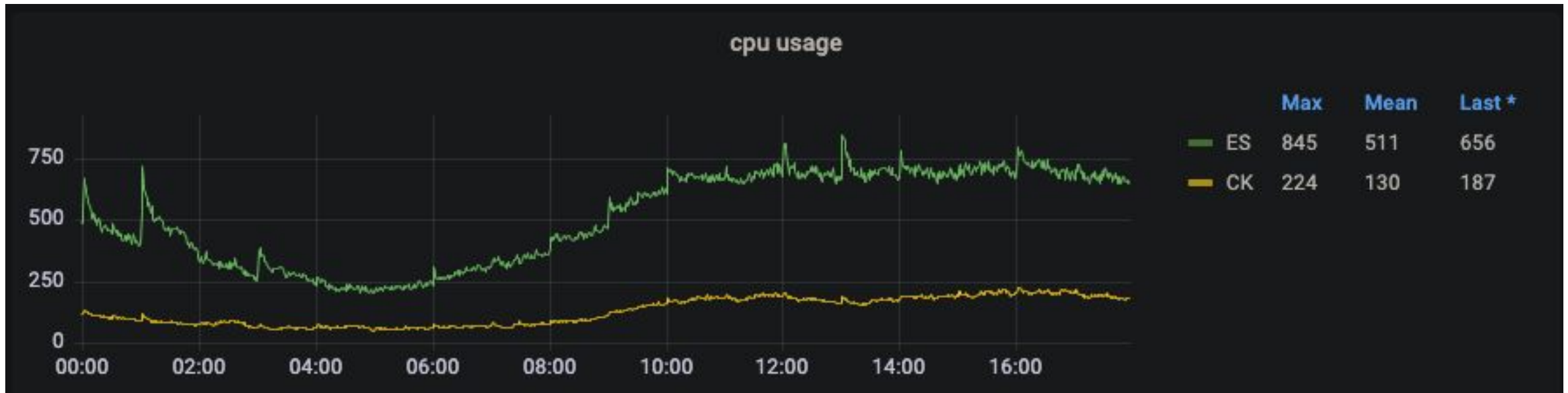
Saved cost over Elastic Search

Use less hardware to serve the same traffic

	ES	CK	Saved
CPU	$64 \text{ core} * 45 = 2880$	$14 \text{ core} * 56 = 784$	72%
Mem	$256 \text{ GB} * 45 = 11520$	$100 \text{ GB} * 56 = 5600$	51%
Disk	$1.7\text{TB} * 45 = 76.5\text{TB}$	$640\text{GB} * 56 = 35.8 \text{ TB}$	53%

Reduced CPU Usage over Elastic Search

- With less CPU cores, the CPU usage is also only **1/3** of ES
- At 25K QPS, the CPU usage on ClickHouse is about 33%





Reduced Memory Usage over Elastic Search


- ClickHouse uses only **12%** of the ES memory to serve all queries
- The Memory usage on ES is about 86% while it's about 24% on CK



Reduced Memory Usage over Elastic Search

ClickHouse only uses **18.7%** of the ES disk usage.



- 
- 1 ClickHouse in Shopee
 - 2 Achievement
 - 3 Business background**
 - 4 How we achieve it
 - 5 Future plan

Business background and Query patterns

constantly changing

billion records

Shop Id	Item Id	attrib 1	attrib 2	attrib n
1	1
1	2
1	3
2	1
3	4
4	1
...

```
SELECT * FROM table WHERE shop_id = xxx AND item_id = xxx
```

```
SELECT count() FROM table WHERE shop_id = xxx AND item_id =  
xxx AND xxx
```

- **Distributed + Replacing** merge tree is used
- **Replacing** merge tree is used for data update
- At query phase, we generally use the **FINAL** modifier to deduplicate records

Initial table design

```
ENGINE = ReplicatedReplacingMergeTree('{layer}-{shard}/tab_local',{replica}',  
cqs_version)  
ORDER BY (shop_id, item_id)
```


```
ENGINE = Distributed('cluster', 'default', 'tab_local', cityHash64(shop_id))
```

Initial Cluster Configuration

- 3 shards, 2 replicas each shard. In another word, in a total 6 replicas for performance evaluation.
- 8 CPU cores and up to 40 cores each replica

Initial Performance

- only **100 QPS**

- 
- 1 ClickHouse in Shopee
 - 2 Achievement
 - 3 Business Background
 - 4 How we achieve it**
 - 5 Future plan



Challenges

Query/Server Settings

Fine tuning to maximize performance under such scenario

Schema Design

Maximize the performance by proper schema optimization

ClickHouse Kernel

Avoid extra lock contention

Kubernetes

How to reduce impact on K8S

Gateway

Maximize the performance and improve error handling

NOTE: This is not the timeline that we applied optimizations



Settings - Improved QPS 5 times by limiting the number of query threads

Fact

- ClickHouse tries to use as many threads as possible to maximize performance for analytic queries
- Under high concurrency, too many threads causes performance penalty due to threads context switching
- Factors that affect the number of query threads
 - number of data parts that needs to be scanned
 - number of marks that needs to be scanned
 - rows that needs to be scanned

By simply setting *max_threads* to 4 improves the QPS from 100 to 500

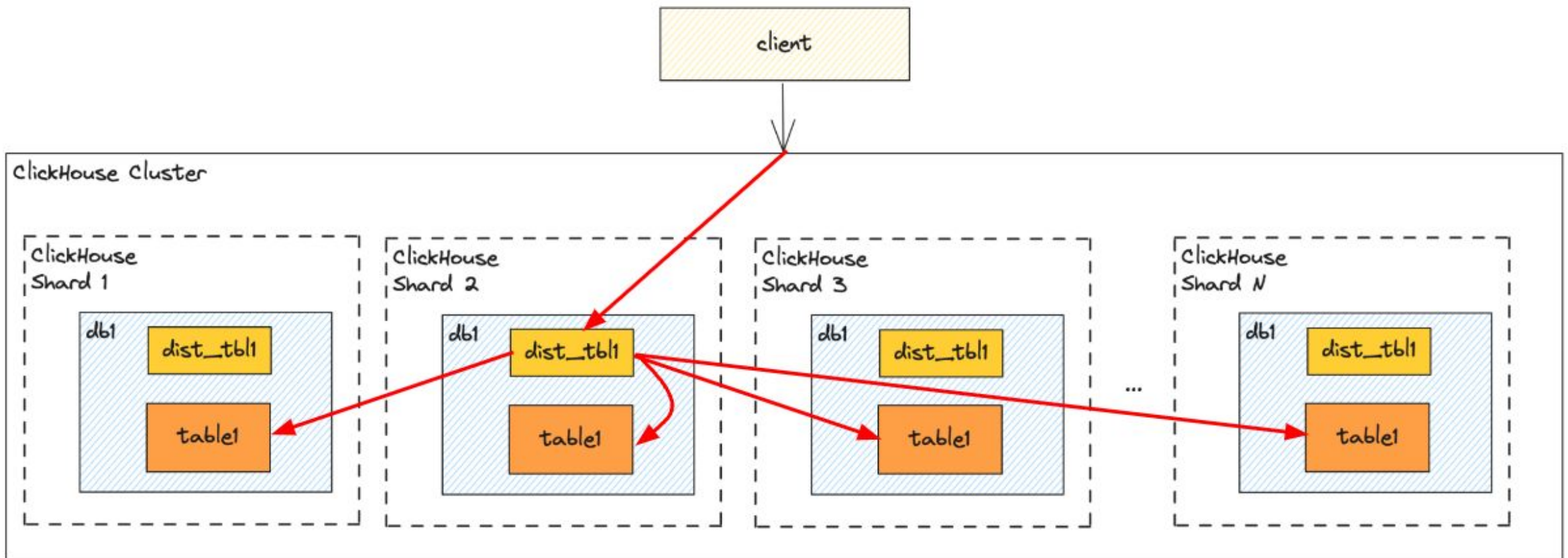
max_threads	Used CK Cores(40 Limited)	Used CK Mem	Avg Latency(ms)	P90 Latency(ms)
32	20c	17%	40	94
16	13c	17%	28	99
8	11c	17%	24	93
4	10c	17%	23	92



Settings - Improved QPS 3 times by applying query phase shard pruning(1)

Fact

- Use **Distributed** table engine to query data from all shards
- **Distributed** table engine needs to dispatch sub-queries to **ALL** shards to execute query on data of each shard

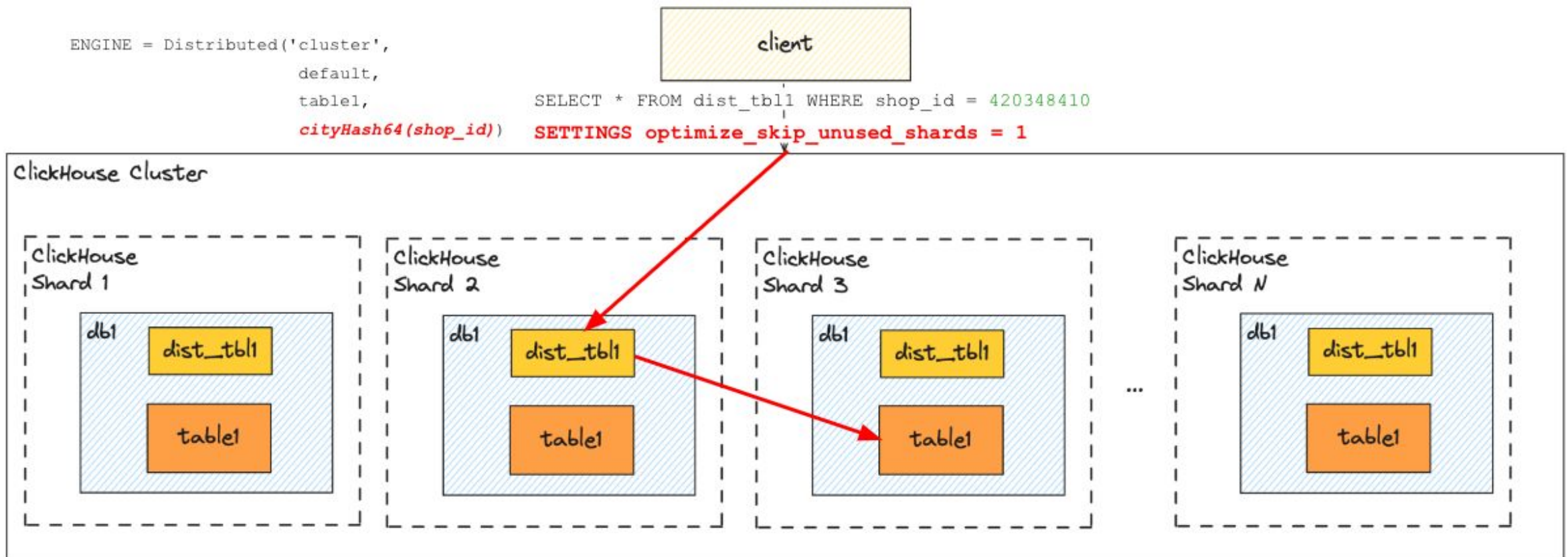




Settings - Improved QPS 3 times by applying query phase shard pruning(2)

Data Characteristics in this use case

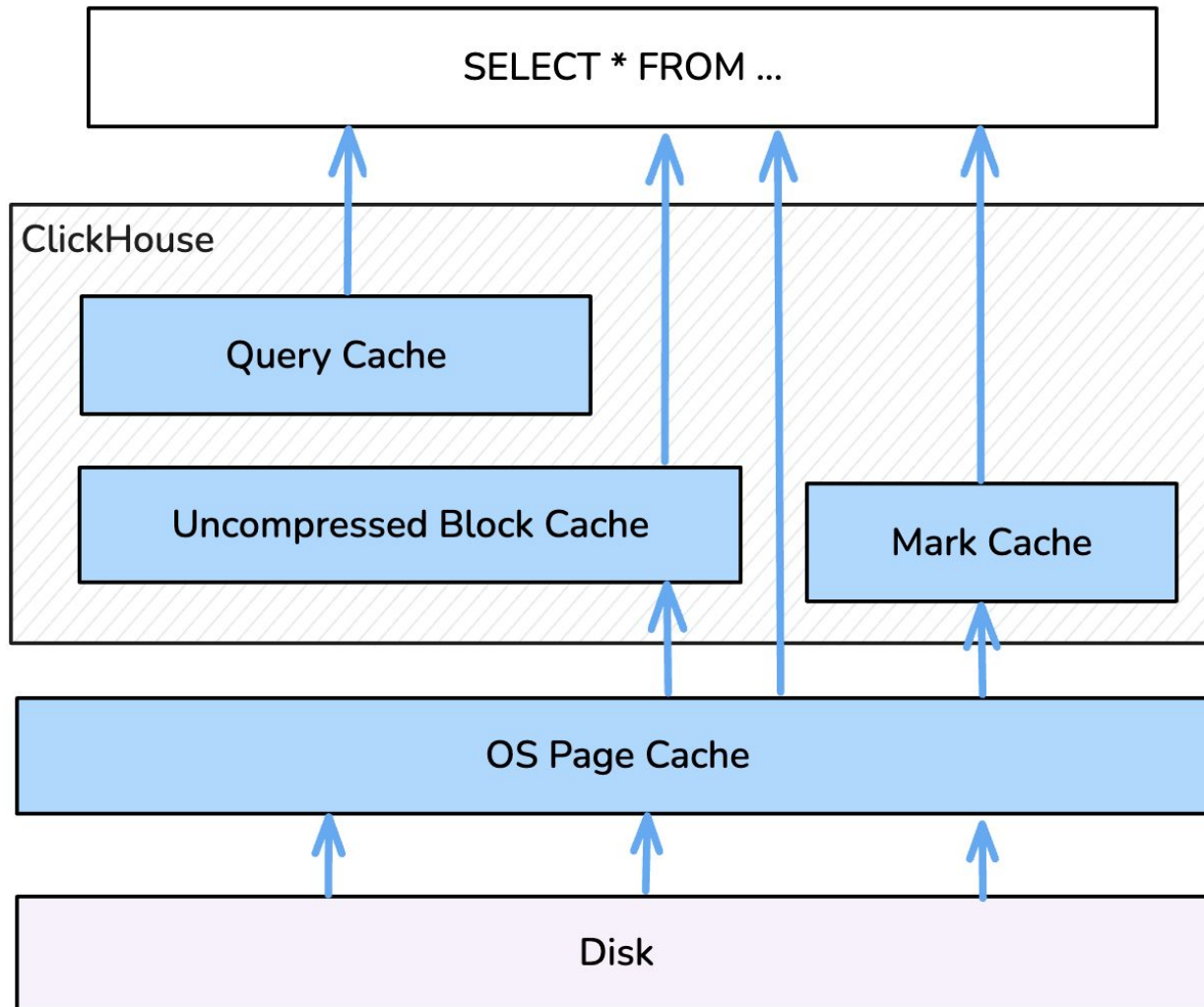
- data is written through Distributed table engine
- data is sharded by a customized sharding key
- queries come with a filter on that specific sharding key



This is not the FINAL solution that we use, but gave us hint to optimize further



Settings - Enabled block cache to reduce CPU usage to reduce 16% CPU usage



- Uncompressed cache is used to cache decompressed data from disk, is NOT enabled by default
- Query Cache is not mature on 23.3



Settings - Uncompressed Cache & Query Cache

- Uncompressed cache reduced 16% CPU usage at a cost of default 8GiB memory cache configuration

Enabled	QPS	Used CPU Cores	Used Mem	Avg Latency	P90 Latency
0	1K	12c	17%	35ms	94ms
1	1K	10c	17%	19ms	90ms

- Query Cache aims to improve the latency, but didn't see significant improvement

Enabled	QPS	Used CPU Cores	AVG Latency	P90	P99	P995
0	2K	8c	5ms	9ms	76ms	88ms
1	2K	7.5c	5ms	9ms	78ms	90ms



Schema Design - Choose a proper partition size

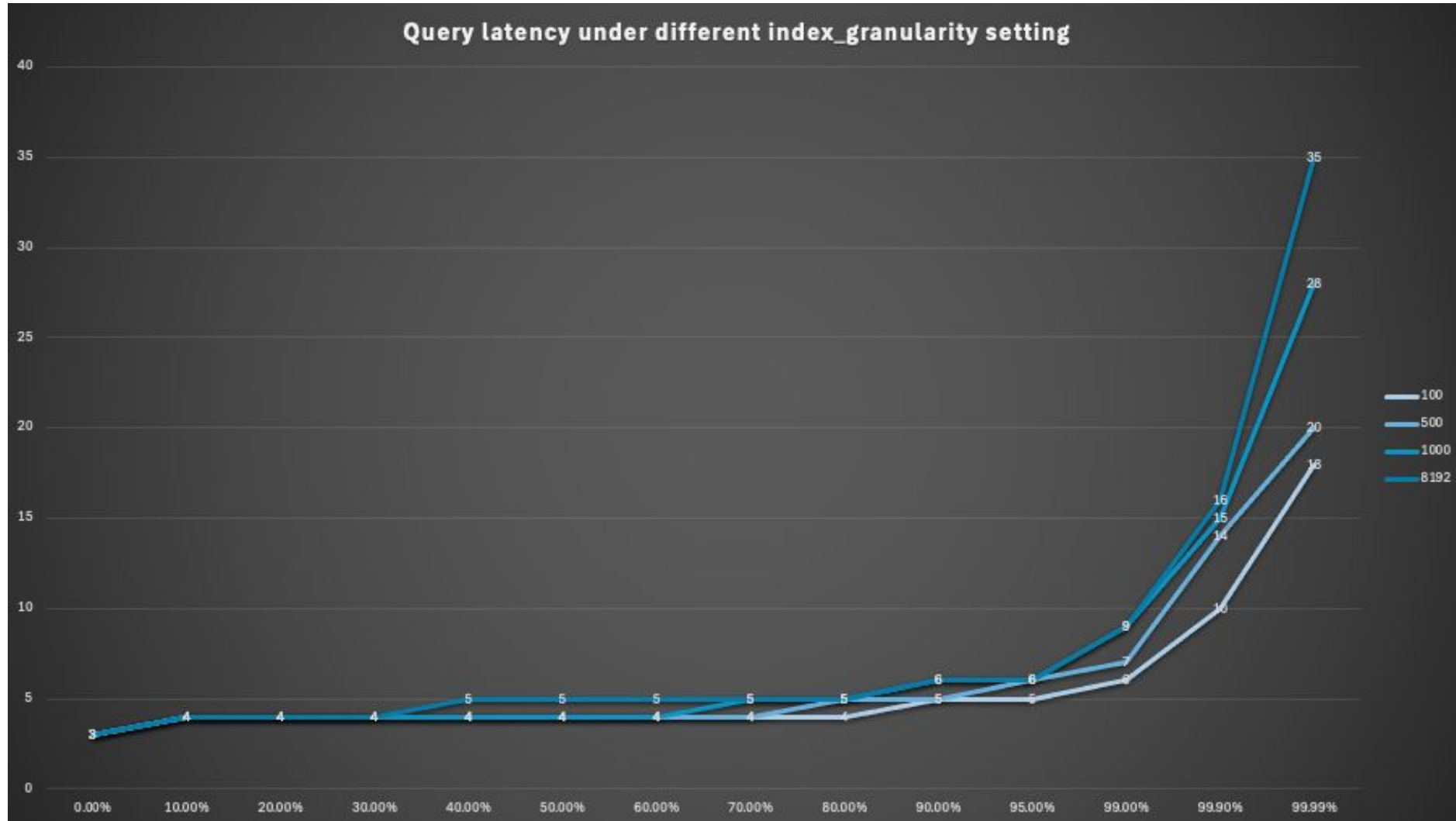
```
PARTITION BY toInt64(shop_id / 10) % N
```

partition num	CK Used Cores	CK mem	avg delay(ms)	p90(ms)
5	11c	17%	18	88
10	10c	17%	18	90
20	10c	17%	17	88

- The table below shows us that the query latency has nothing to do with the number of partition numbers.
- Even under the 20 partitions case, the average latency and P90 latency decreases 1 millisecond, we think the improvement is trivial.

Schema Design - Lowered the granularity

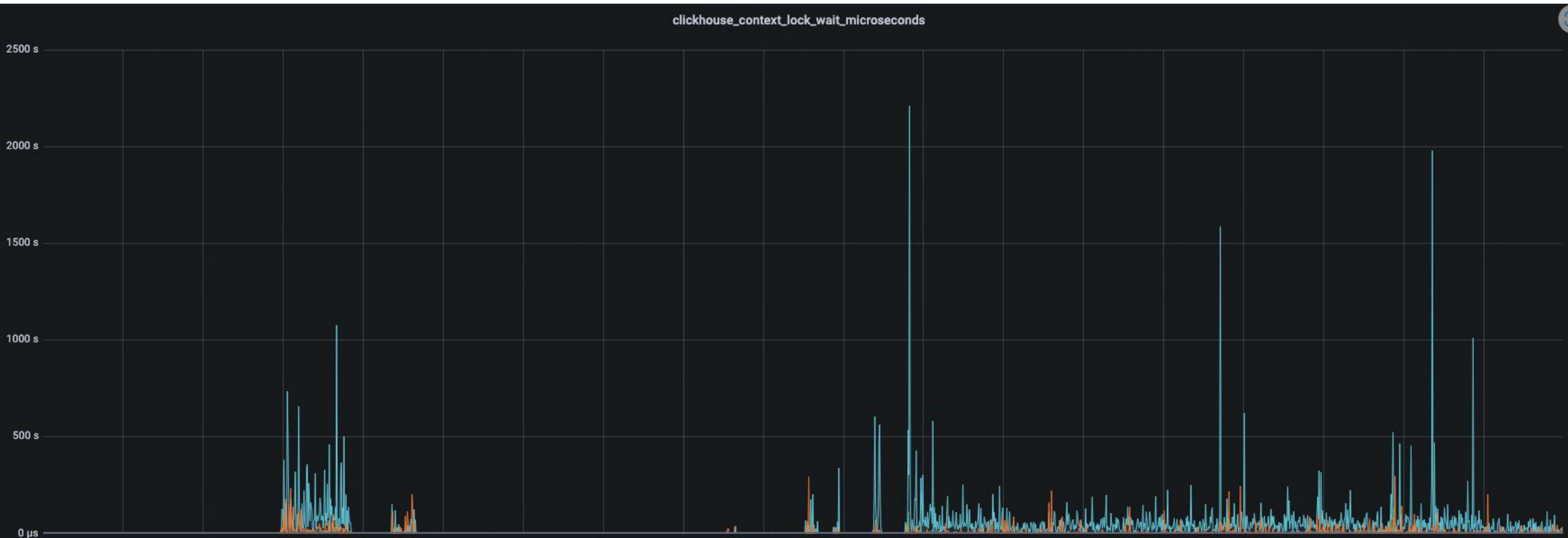
- This case is a point query case, means that lowered granularity might help for latency
- When the granularity is reduced from default 8192 to 100, the P99 latency was reduced about 50%





Kernel - Minimized internal lock to improve QPS from 1500 to 2000+

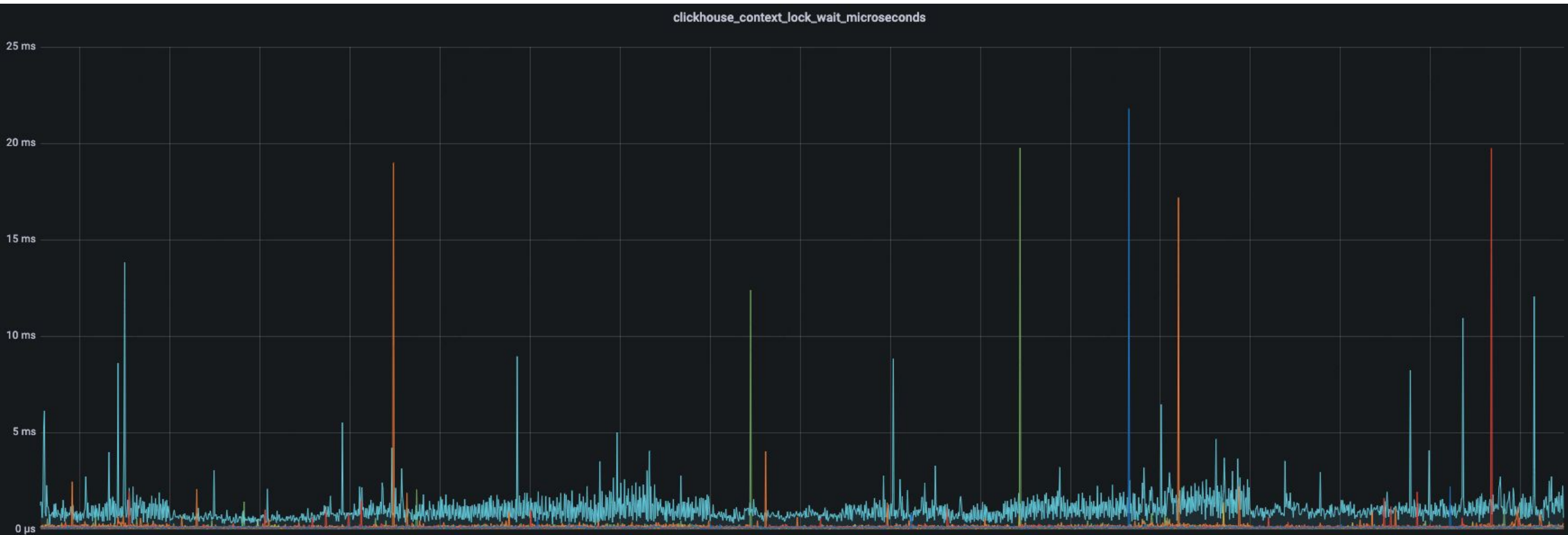
- ClickHouse internally accesses a global or query level context very frequently, and a lock is used for each access method
- Under high concurrency, this causes unstable query latency and limit the concurrent performance





Kernel - Minimized internal lock to improve QPS from 1500 to 2000+

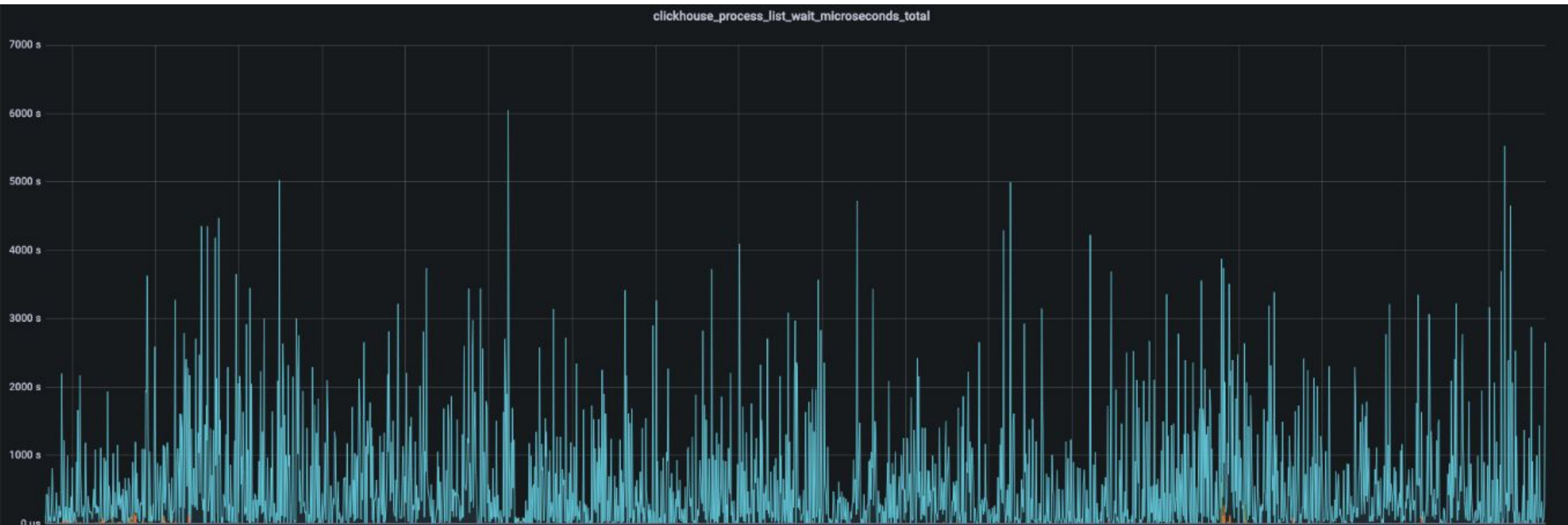
- During query execution, most of access is a READ operation, a READ-WRITE lock can be used to reduce lock contention
- Under the same QPS pressure, the max of total time spending on waiting was reduced to milliseconds from thousands of seconds





Kernel - Minimized ProcessList lock contention(1)

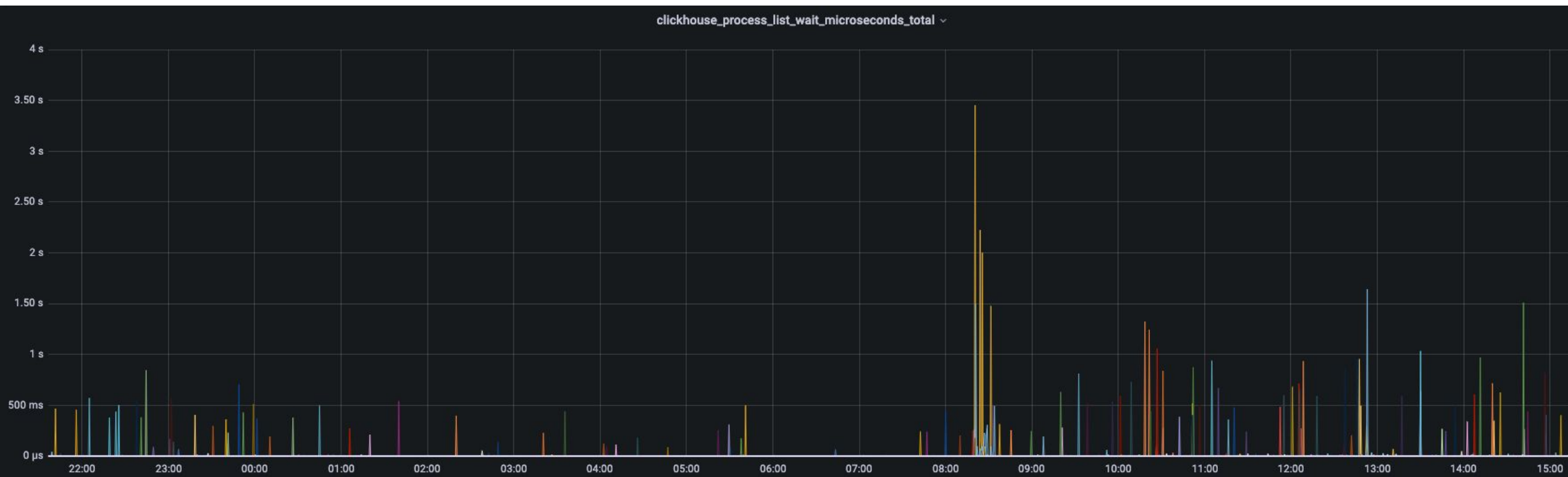
- Similar to previous problem, there's a ProcessList in ClickHouse to maintain info of running queries
 - It will be accessed every time when a query comes in and a query finishes
- In Shopee, the ProcessList was refactored to track INSERT and SELECT queries respectively
 - Under high concurrency, the original design brought in performance problem





Kernel - Minimize ProcessList lock contention(2)

- The total time spending on waiting for LOCK is reduced from thousands of seconds to seconds

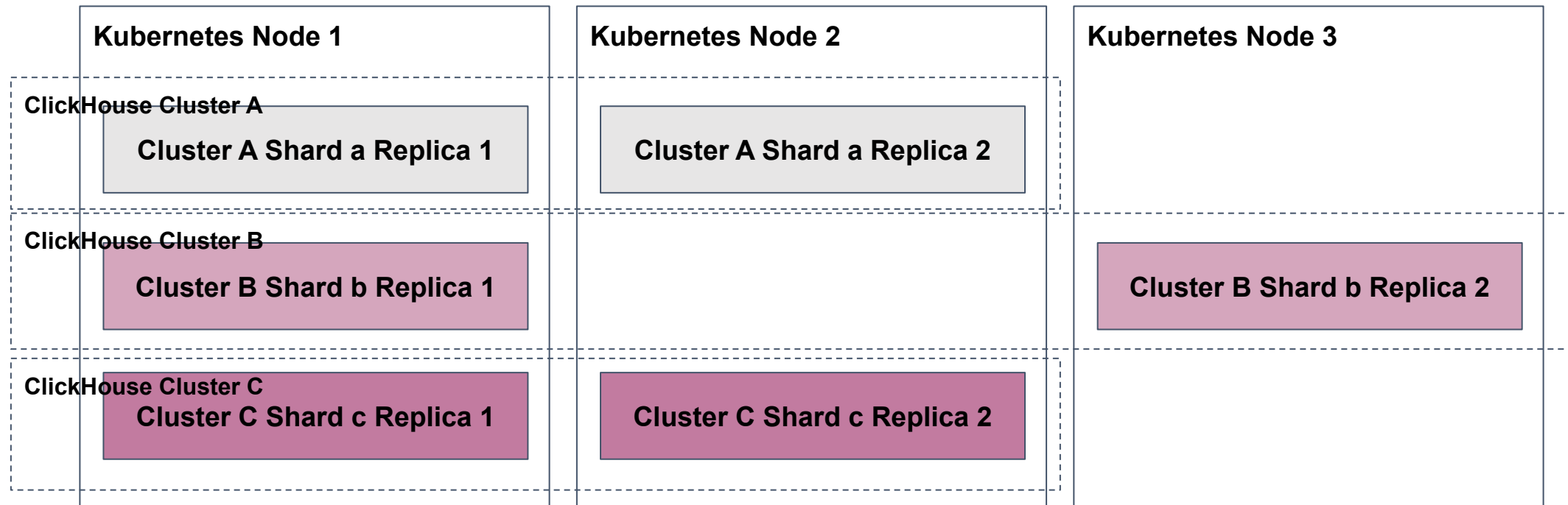




Kubernetes - IO isolation to stabilize query latency

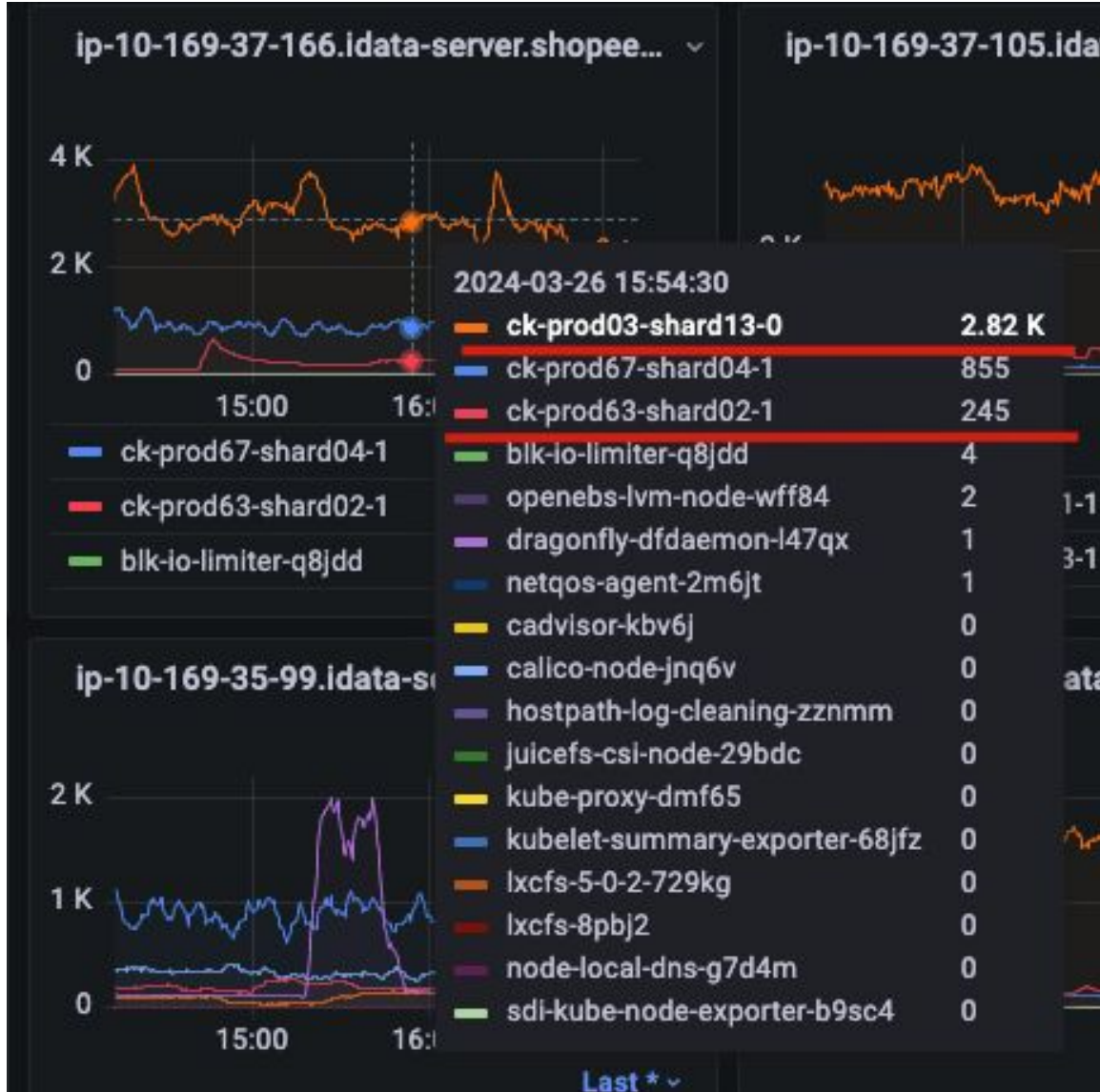
ClickHouse clusters are deployed on top of K8S

- The K8S node is much powerful to deploy multiple ClickHouse POD
- Two replicas of same shards are NOT deployed on the same K8S node to ensure availability
- PODs from different ClickHouse clusters might deploy on the same K8S node
- K8S provides CPU/Memory isolation, but does not provide IO isolation
- Some businesses have heavy IO load which might reduce the IO performance on the K8S node





Kubernetes - IO isolation to stabilize query latency(2)



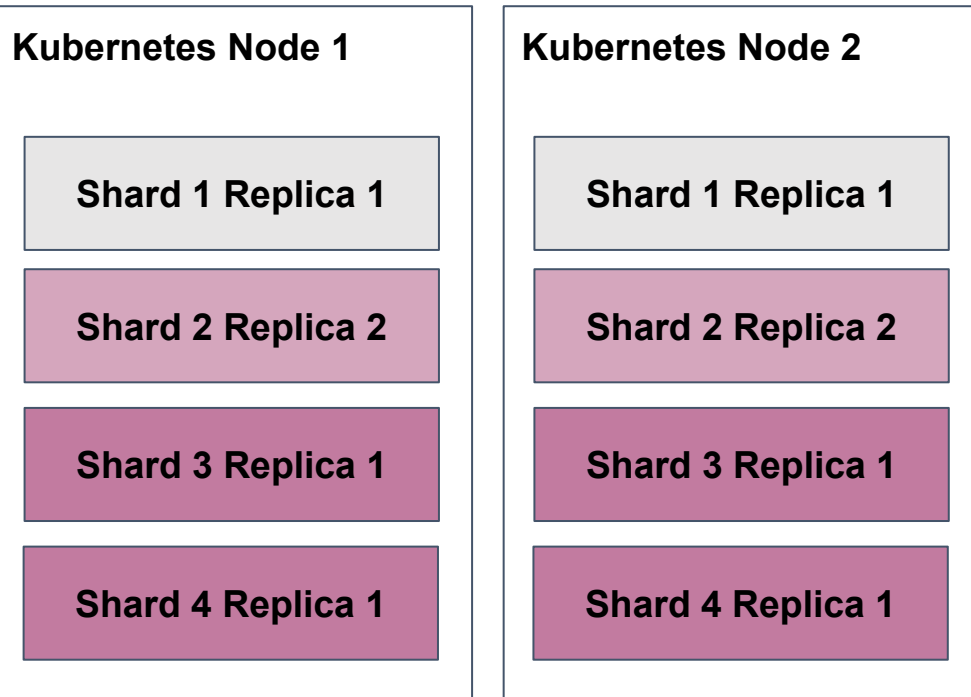
- prod03-shard13-0 was a POD of a cluster that has very heavy IO usage
- prod63-shard02-1 was the POD of the cluster that is used to serve the high concurrency
- Queries on shard02-1 had higher latency than other replicas due to longer IO wait



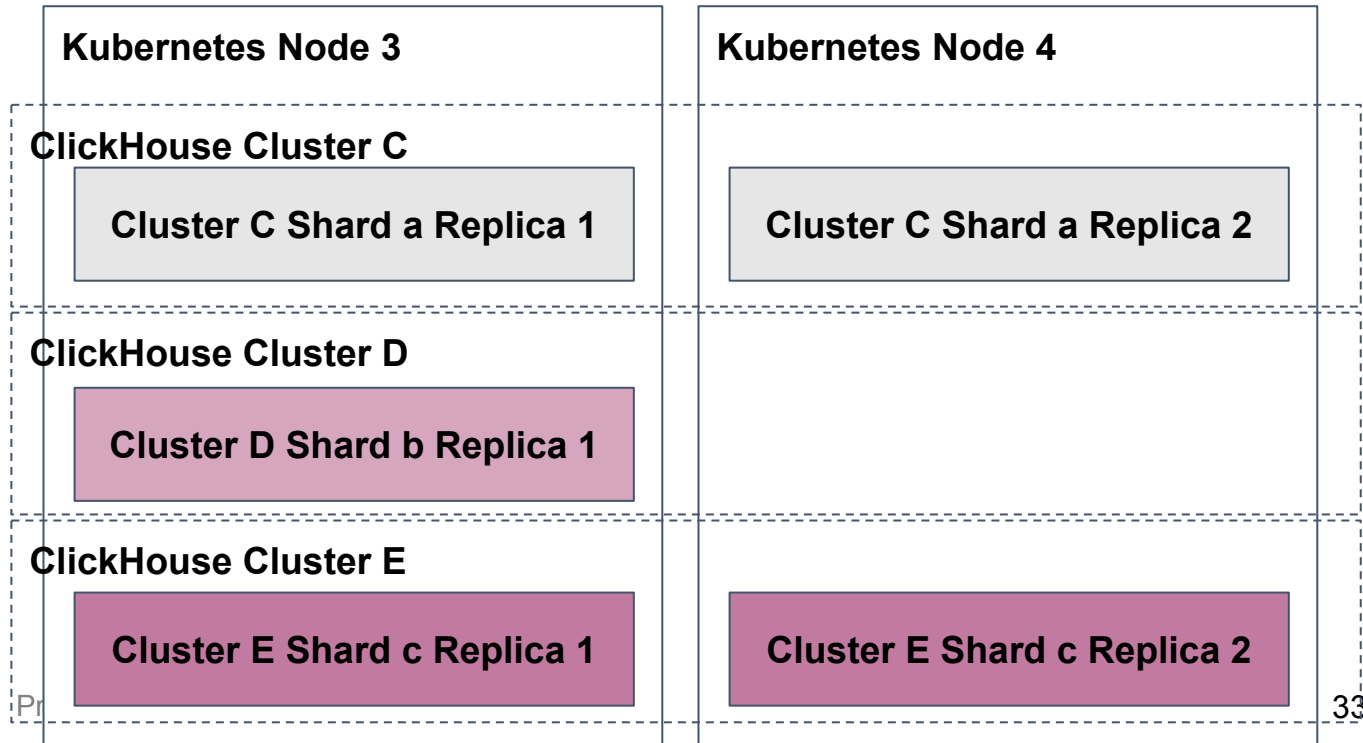
Kubernetes - IO isolation to stabilize query latency(3)

- We applied IO limiting at the K8S layer, but considering the business is much different from OLAP business, we imposed an isolation at K8S node level
 - Cluster A can only SELECT nodes 1,2 for deployment
 - Other clusters can NOT select nodes 1, 2 for deployment
- How many PODs are deployed on a single node?
 - Based on SSD performance and concurrency needs, 4 PODs on a single server
 - The node has enough CPU & Memory for 4 PODs

ClickHouse Cluster A (demonstration only, not the real deployment)



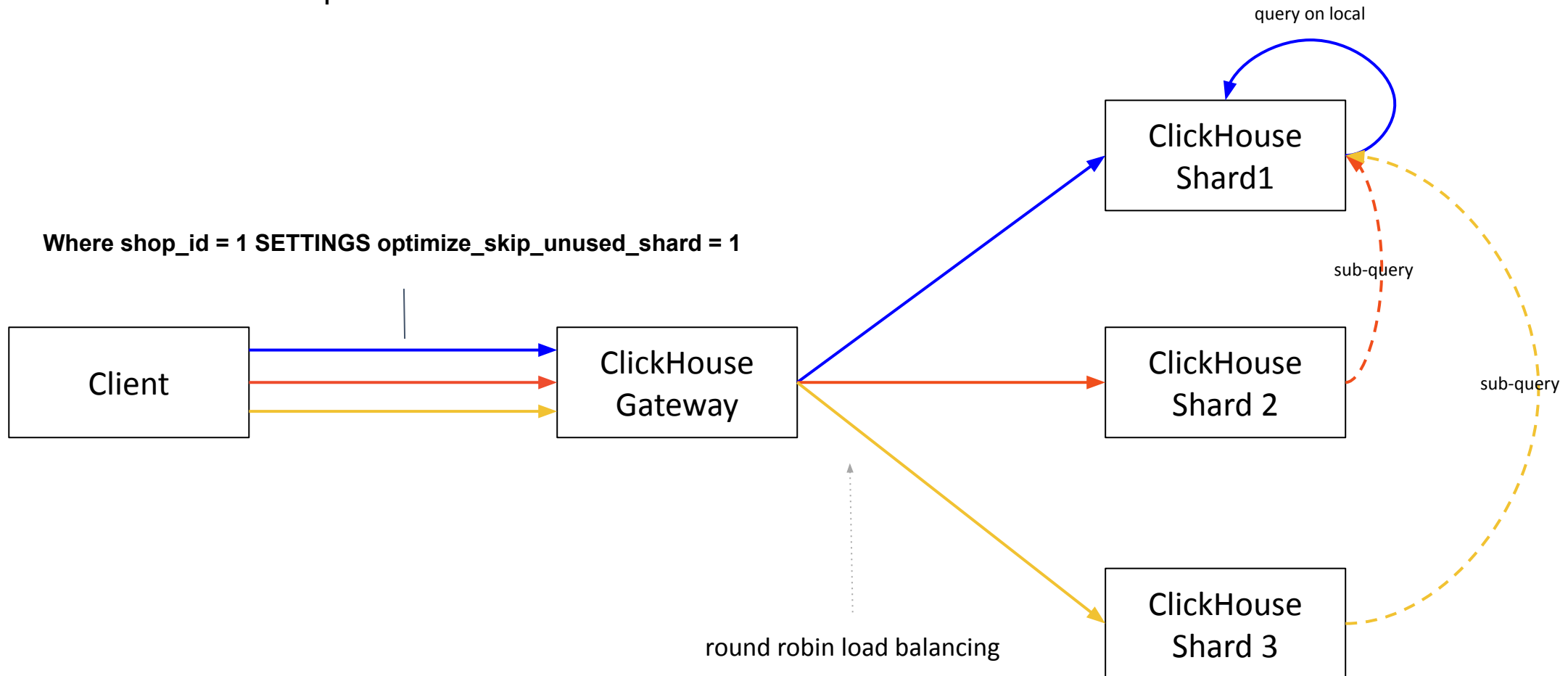
K8S Cluster





Gateway - Support of sharding routing to achieve sub100ms P99 latency

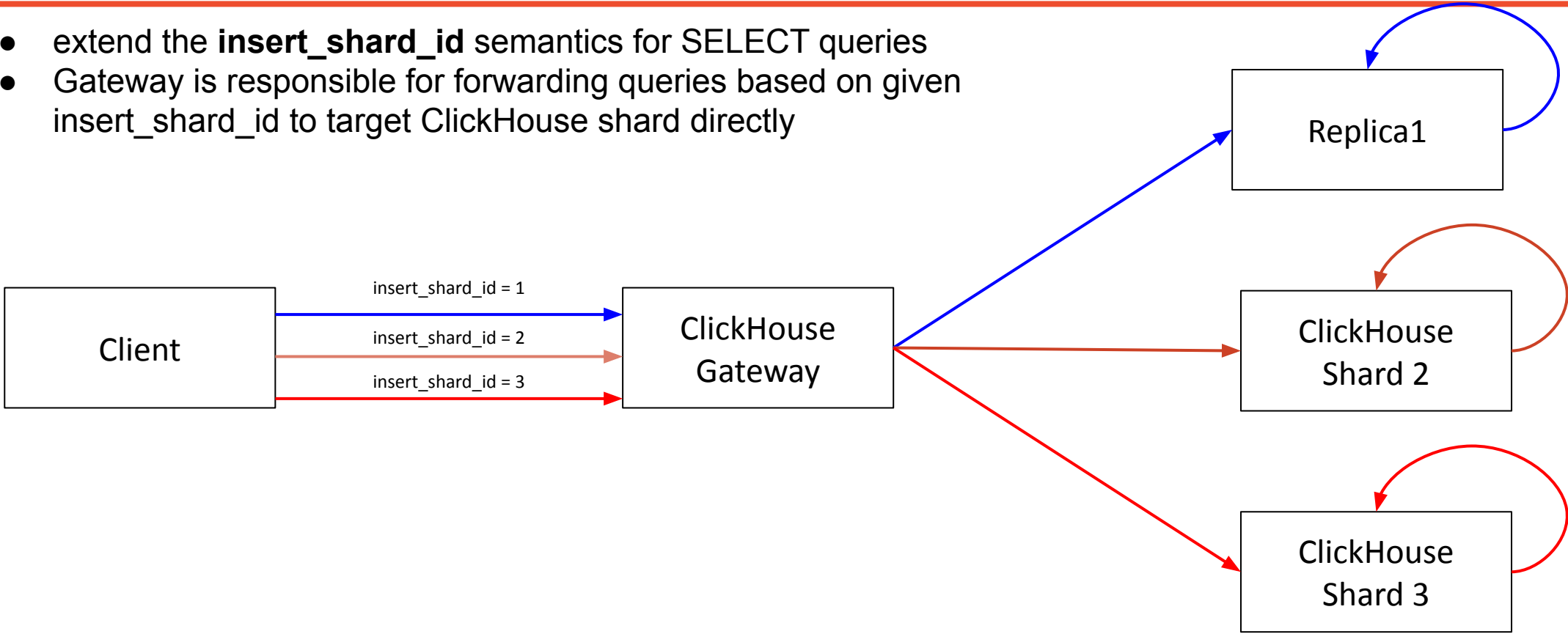
- When the shard pruning setting is turned on, we observed that QPS was increased significantly, but latency was not stable
- One reason is that, query on distributed table might involve extra forwarding inside ClickHouse, increasing number of sub-queries





Gateway - Support of sharding routing to achieve sub100ms P99 latency(2)

- extend the `insert_shard_id` semantics for SELECT queries
- Gateway is responsible for forwarding queries based on given `insert_shard_id` to target ClickHouse shard directly

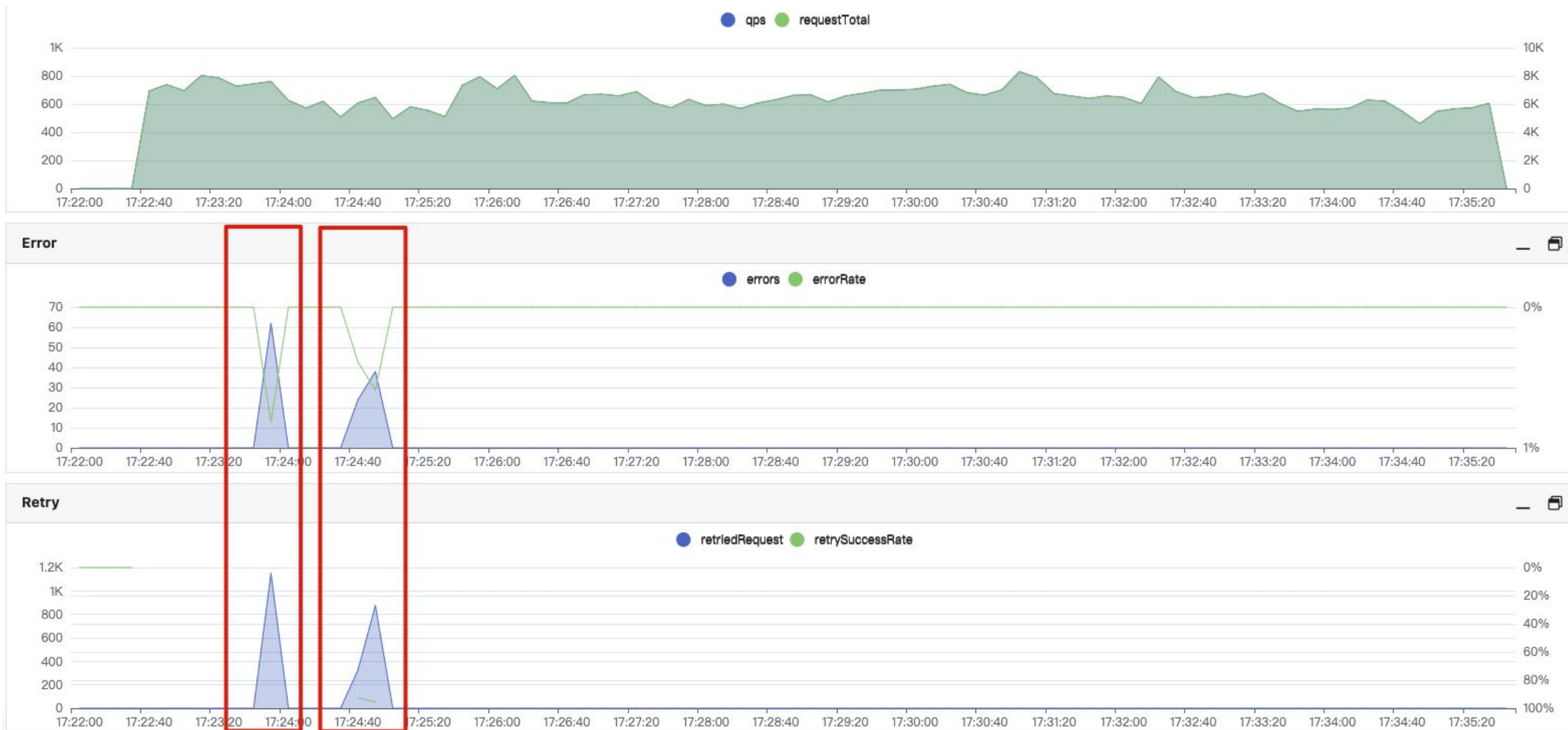


	QPS	Used CPU Cores	P50(ms)	P90(ms)	P99(ms)	P995(ms)
Before	2K	22	27	90	1500	3000
After	2K	18	11	71	98	99



Gateway - Eliminate query errors during cluster upgrading

- Lots of query Retries or ERRORS observed at the Gateway side during the restarting of ClickHouse cluster shard
- This is NOT acceptable for such large online traffic business



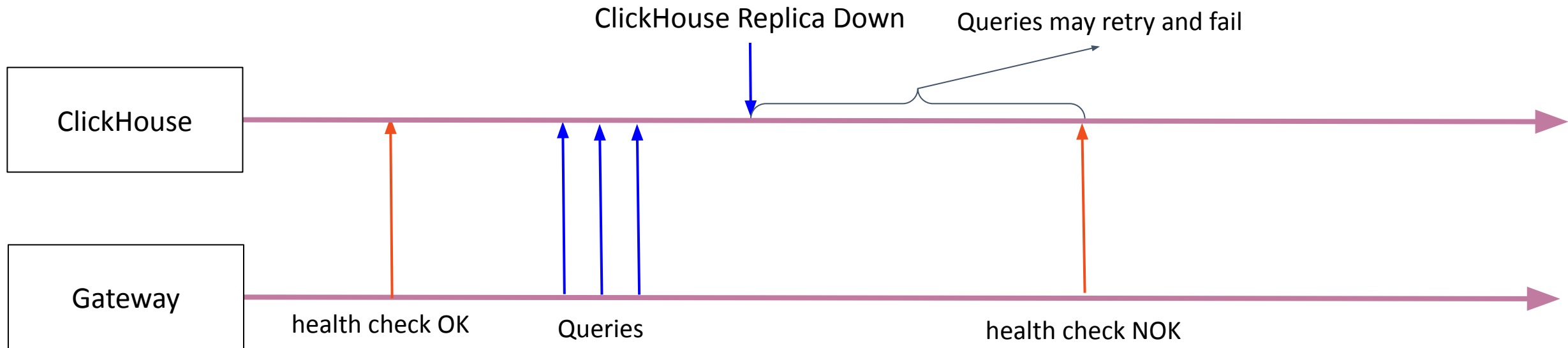


Gateway - Eliminate query errors during cluster upgrading(2)

- This problem is about how Gateway detects health status of backend ClickHouse replicas



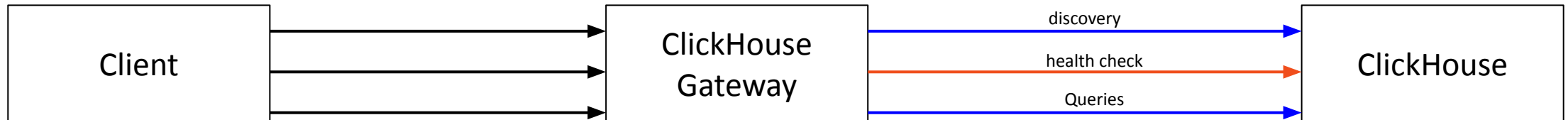
- Health check on every replica is performed separately from queries in a fixed interval
- There's a small window when the backend ClickHouse replica is down while new health check has not been issued to mark that replica as unhealthy



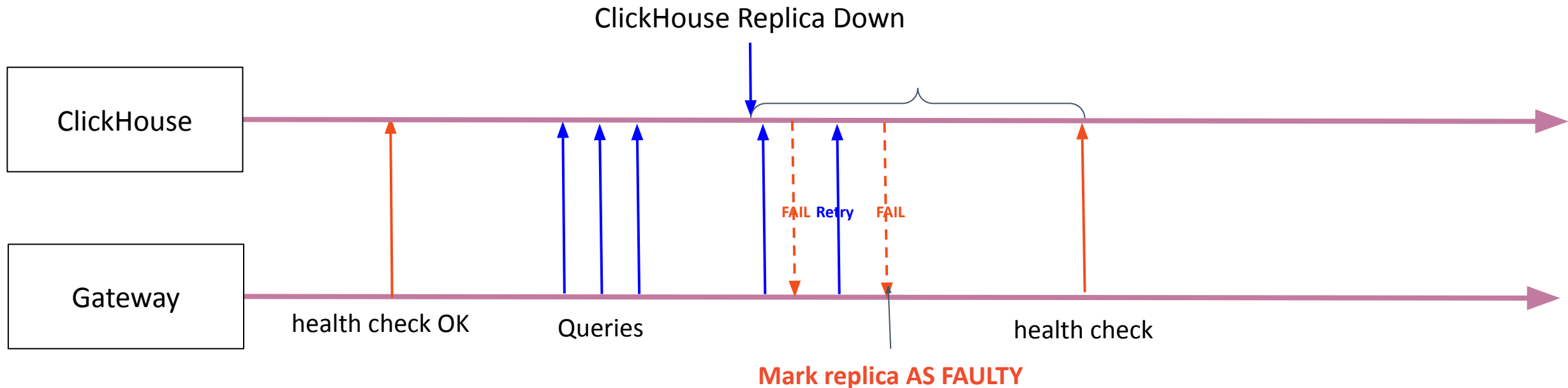


Gateway - Eliminate query errors during cluster upgrading(2)

- This problem is about how Gateway detects health status of backend ClickHouse replicas



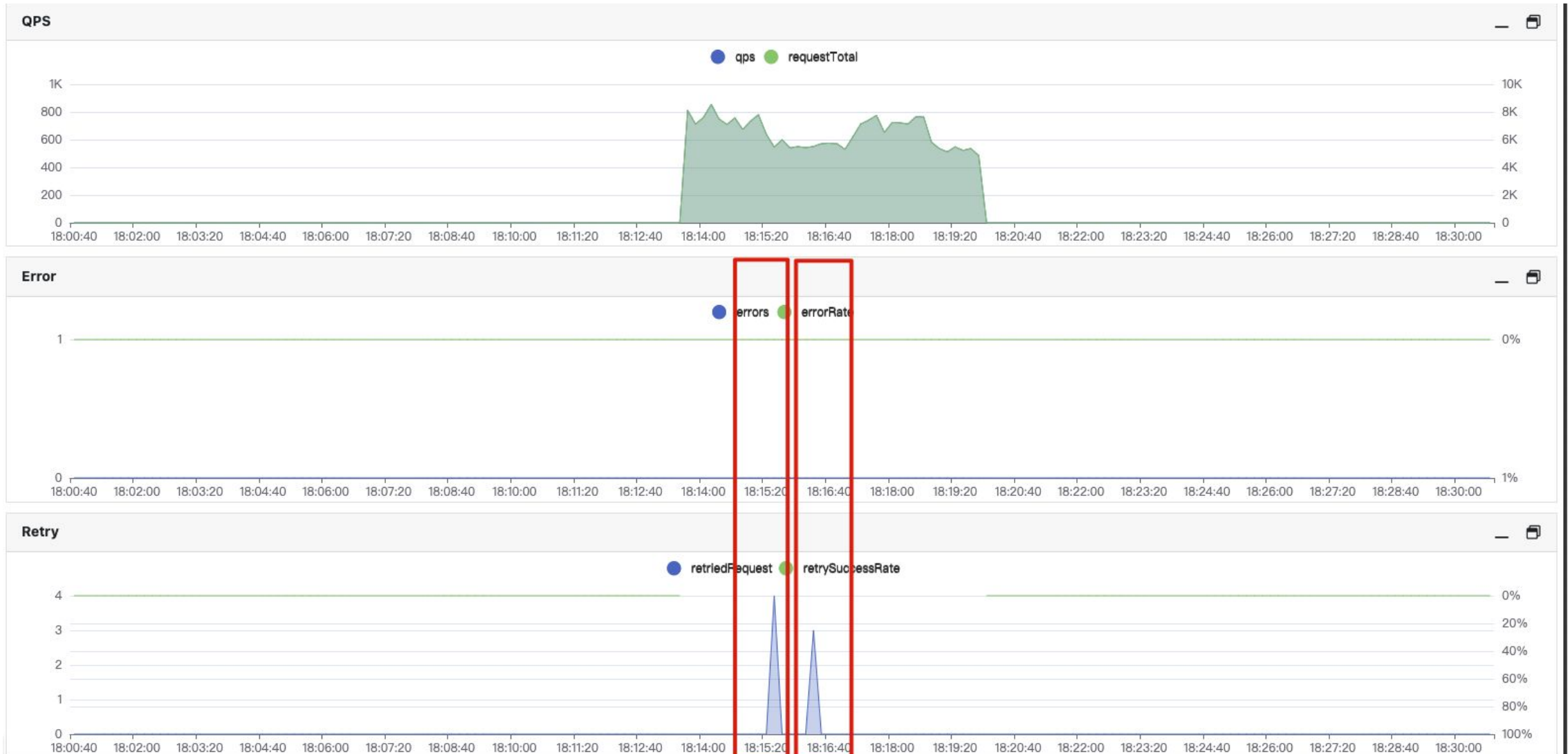
- Health check on every replica is performed separately from queries in a fixed interval
- There's a small window when the backend ClickHouse replica is down while new health check has not been issued to mark that replica as unhealthy






Gateway - Eliminate query errors during cluster upgrading(3)

- Only very few retries were observed when a shard restarts



Private & Confidential

- Client Side
 - Organize INSERT statement by PARTITION and INSERT into shard directly
 - Tried LIMIT BY/GROUP BY as alternative of FINAL for query phase deduplication, but FINAL in this case is the best one
- ClickHouse
 - Increased keep-alive timeout to maintain longer connections between Gateway and Kernel to reduce connection set up time
 - Query Cache, which did NOT help in this case
- ClickHouse Gateway
 - Configure to use fast-fail in case of high load
 - Upgrade from JDK 11 to JDK21 to try ZGC, but led to NEGATIVE performance, min latency increased about 10%
- Other
 - Improved ClickHouse metric export SQL to reduce resource utilization

- 
- 1 ClickHouse in Shopee
 - 2 Achievement
 - 3 Business Background
 - 4 How we achieved it
 - 5 **Future plan**



There're more work that need to be done in the core of ClickHouse

- `index_granularity`
 - The `index_granularity` has been lowered down to match the data characteristics, however, they're few shops that have extra large records, this optimization leads to negative performance problem
- Data updating
 - ReplacingMergeTree is used for updating, which brings in some query performance penalty because of **FINAL**
- Data deletion
 - Lightweight deletion still has very poor performance and requires extra optimization
 - Deleting several rows sometimes took seconds which is not acceptable in this case



This achievement is a team work under the collaboration of the following colleagues.

Name	Team
Chaoguang Qin	Listing team, Marketplace
Teng Ma	OLAP Team, Data Infrastructure
Pengdong Zhong Jinming Guo	SRE Team, Data Infrastructure



Thanks