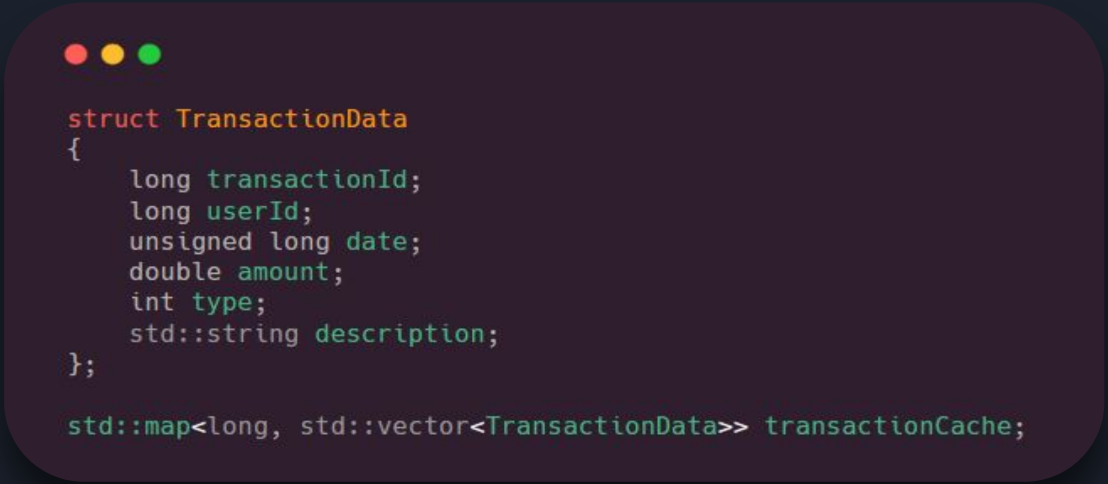
Optimizing Container Synchronization for Frequent Writes

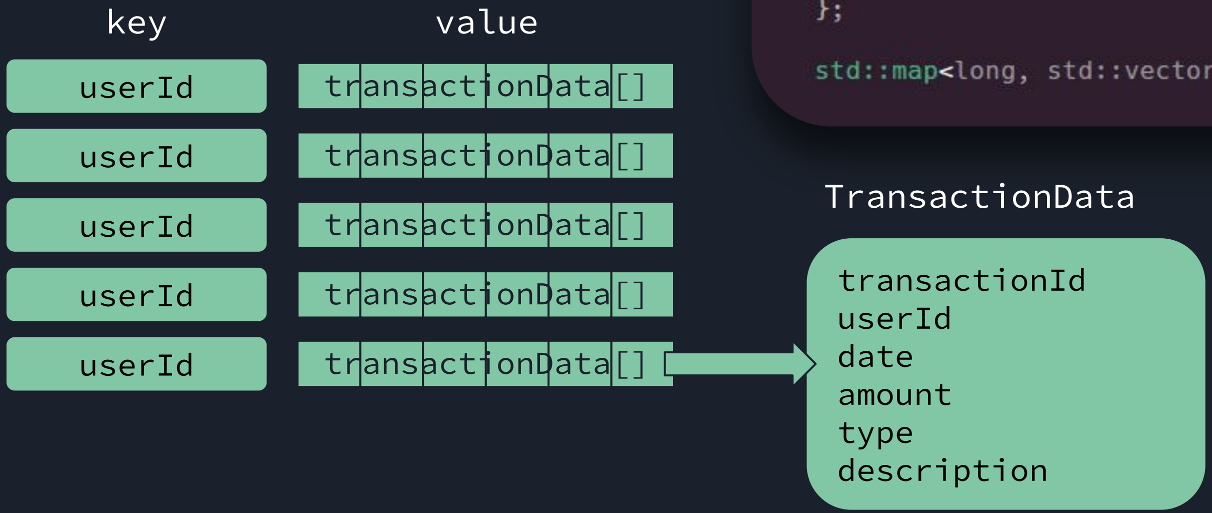
Aleksandr Iskhakov
CTO at Tools For Brokers

Topic: **Synchronization** in multi-threaded architecture

Cache



```
struct TransactionData
{
    long transactionId;
    long userId;
    unsigned long date;
    double amount;
    int type;
    std::string description;
};

std::map<long, std::vector<TransactionData>> transactionCache;
```

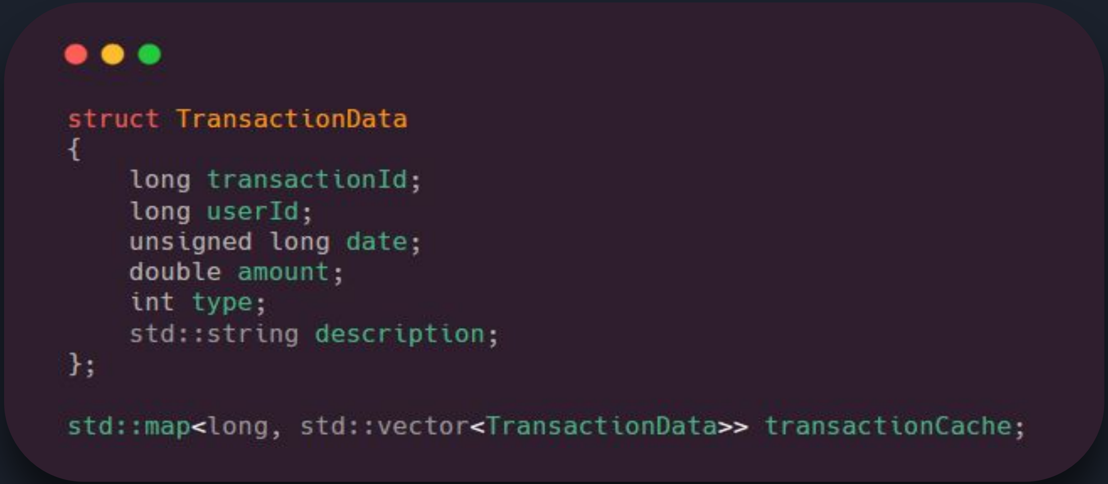


key	value
userId	transactionData[]
userId	transactionData[]
userId	transactionData[]
userId	transactionData[]
userId	transactionData[]

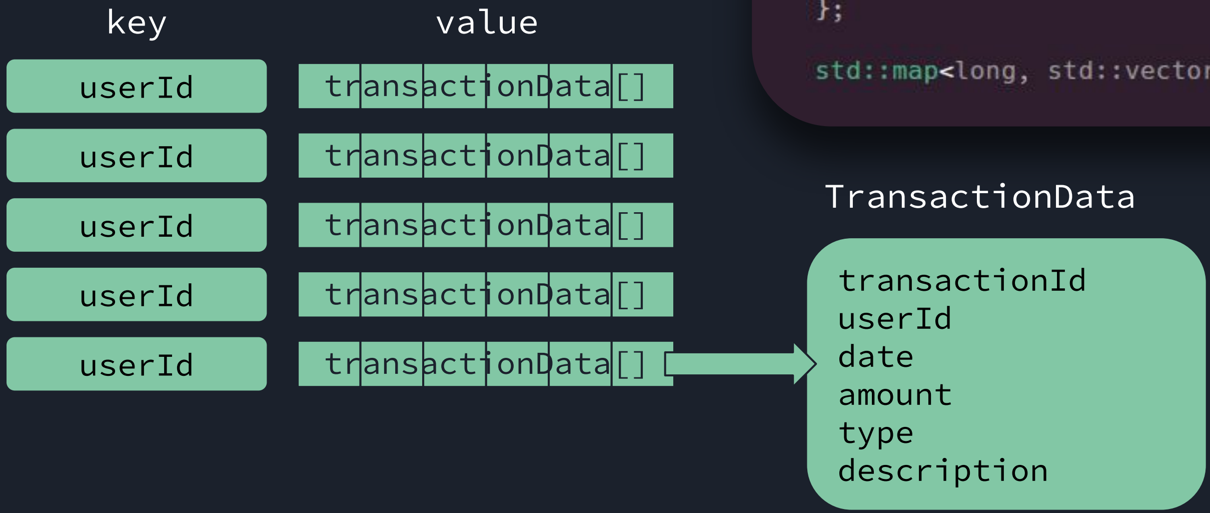
TransactionData

transactionId
userId
date
amount
type
description

Cache



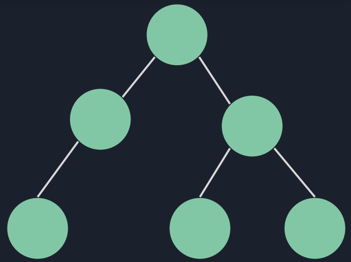
```
struct TransactionData
{
      long transactionId;
      long userId;
      unsigned long date;
      double amount;
      int type;
      std::string description;
};

std::map<long, std::vector<TransactionData>> transactionCache;
```



key	value
userId	transactionData[]
userId	transactionData[]
userId	transactionData[]
userId	transactionData[]
userId	transactionData[]

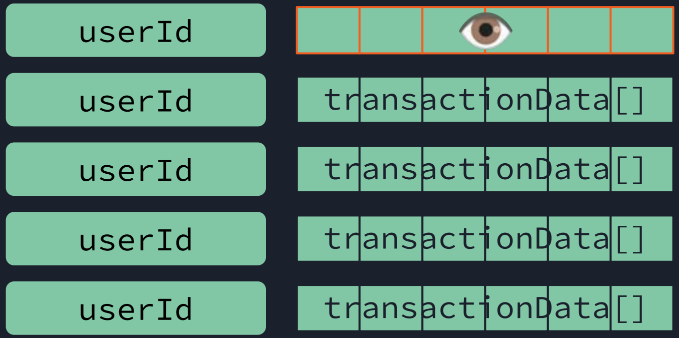
TransactionData

transactionId
userId
date
amount
type
description



Cache operations

Read Write Pop



userId 👁

userId transactionData[]

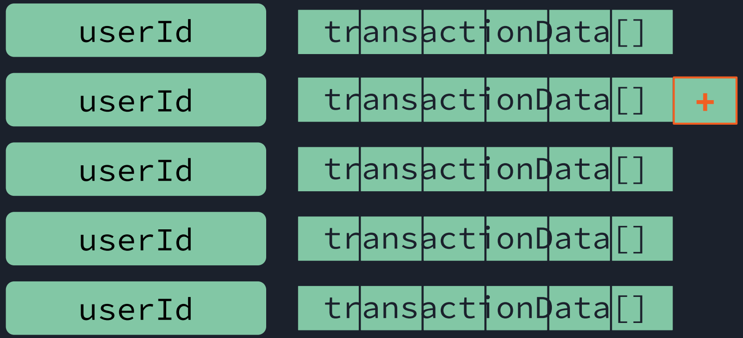
userId transactionData[]

userId transactionData[]

userId transactionData[]

Cache operations

Read **Write** Pop



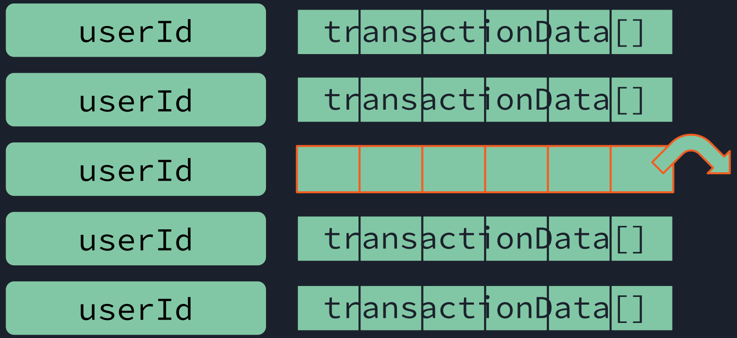
userId	transactionData[]	
userId	transactionData[]	+
userId	transactionData[]	
userId	transactionData[]	
userId	transactionData[]	

Cache operations

Read

Write

Pop



userId | transactionData[]

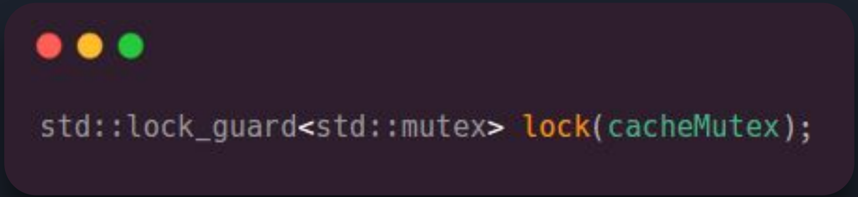
userId | transactionData[]

userId |

userId | transactionData[]

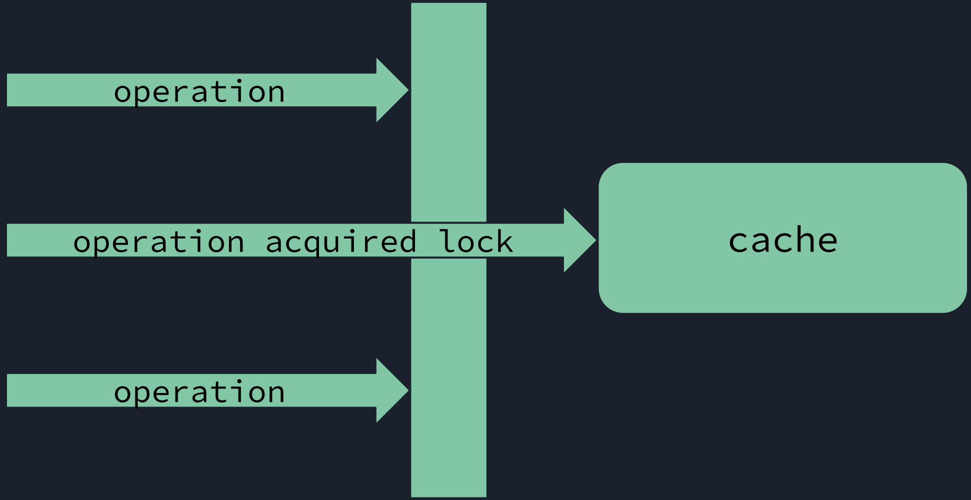
userId | transactionData[]

Mutex



```
std::lock_guard<std::mutex> lock(cacheMutex);
```

One operation at a time



operation

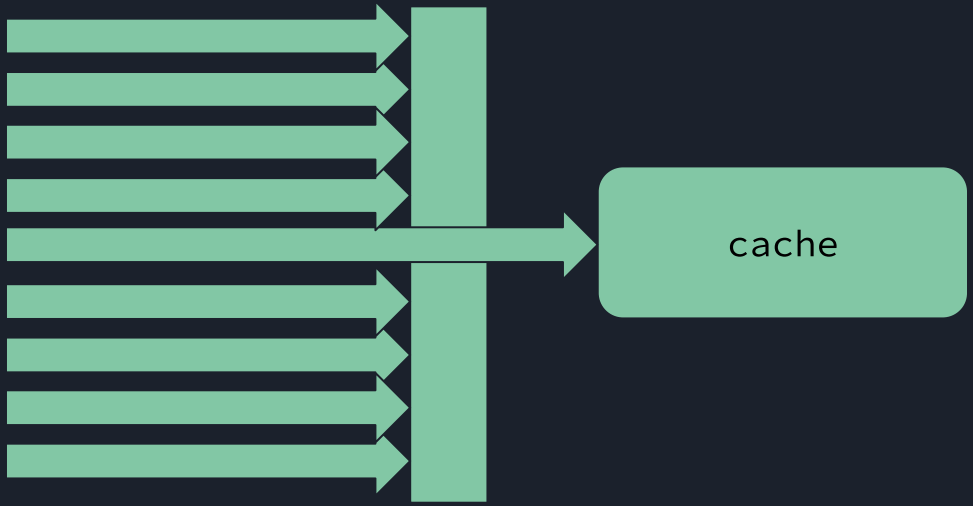
operation acquired lock

operation

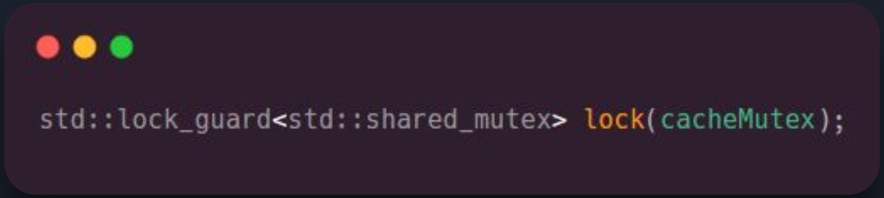
cache

Mutex

Load increases...

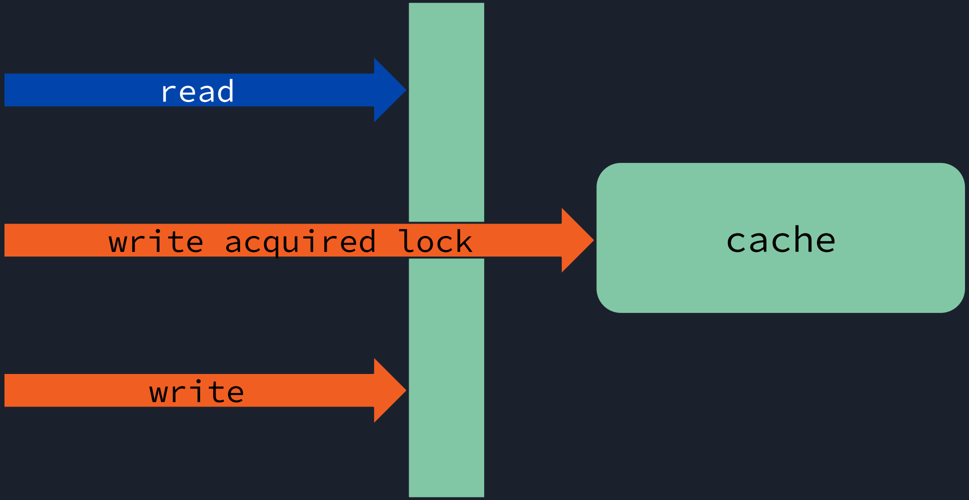


Shared mutex (RW Lock)



```
std::lock_guard<std::shared_mutex> lock(cacheMutex);
```

1. Unique locking



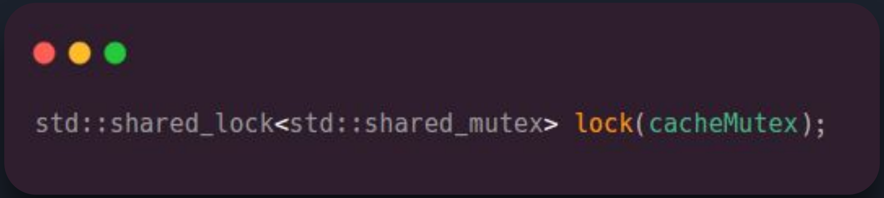
read

write acquired lock

cache

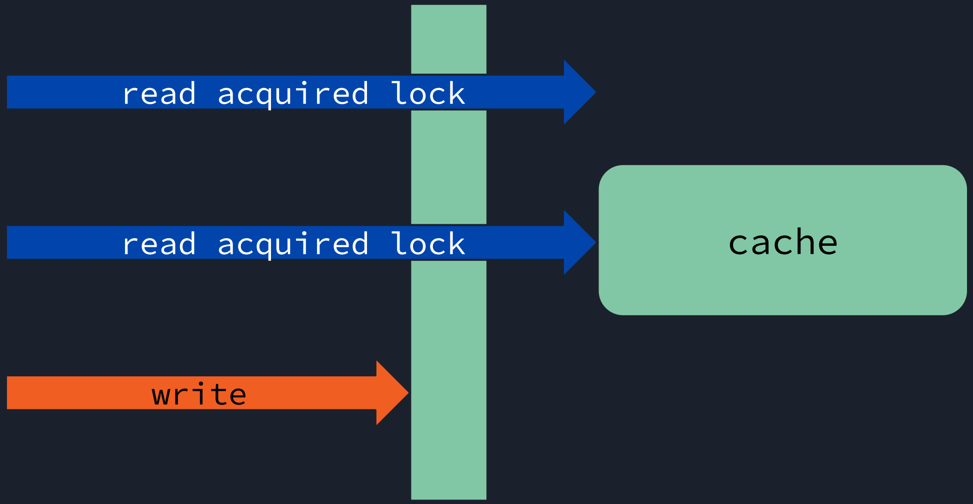
write

Shared mutex (RW Lock)



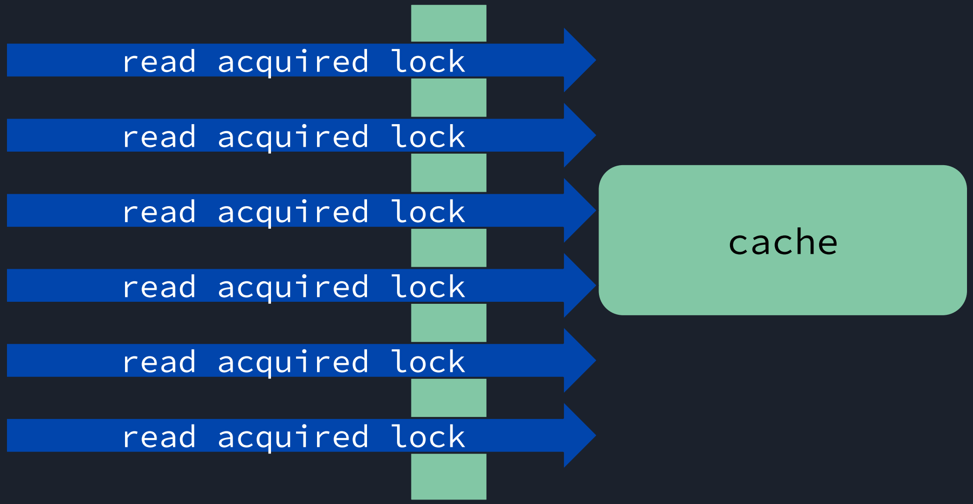
```
std::shared_lock<std::shared_mutex> lock(cacheMutex);
```

2. Shared locking



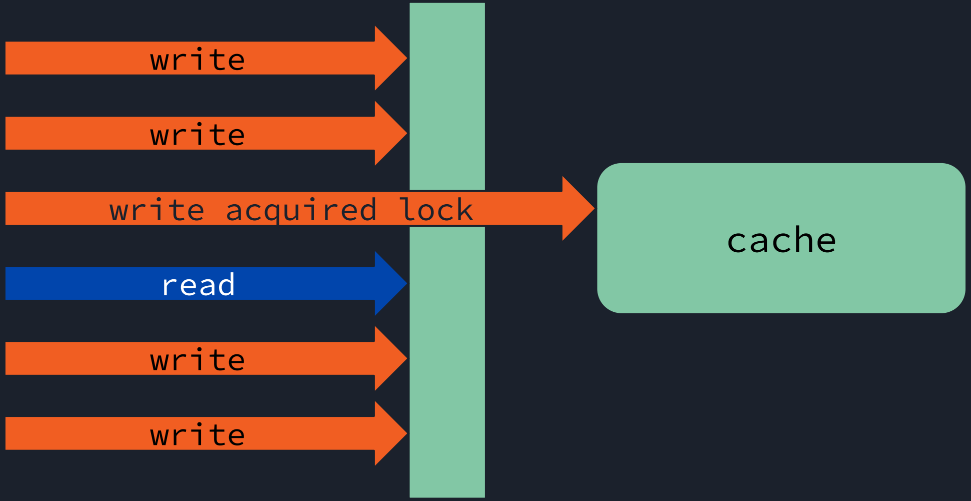
Shared mutex (RW Lock)

Works like a charm in read-heavy environments…



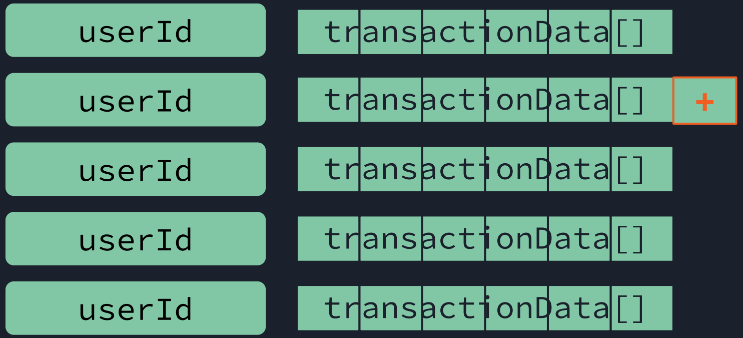
Shared mutex (RW Lock)

…but what about write-heavy?



Renewed example

Write



userId transactionData[]

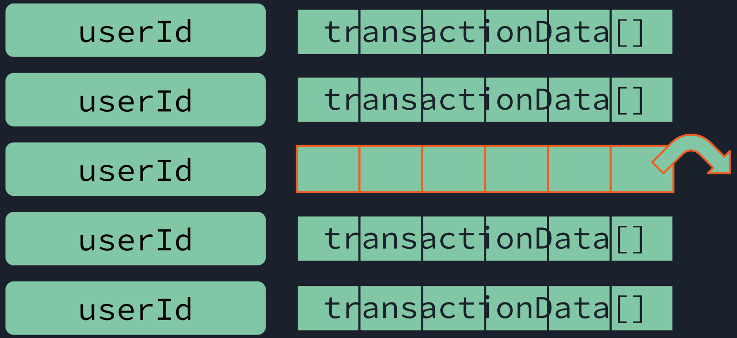
userId transactionData[] +

userId transactionData[]

userId transactionData[]

userId transactionData[]

Pop



userId transactionData[]

userId transactionData[]

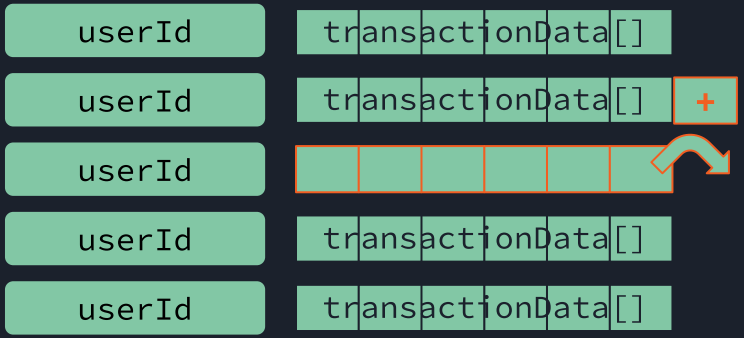
userId

userId transactionData[]

userId transactionData[]

Renewed example

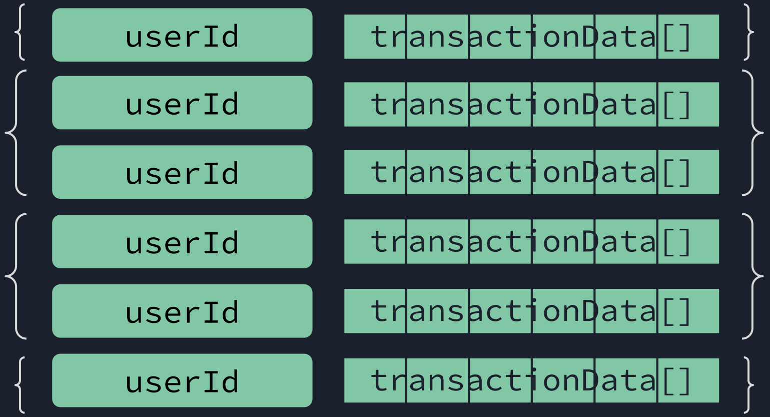
Let’s look at the data structure



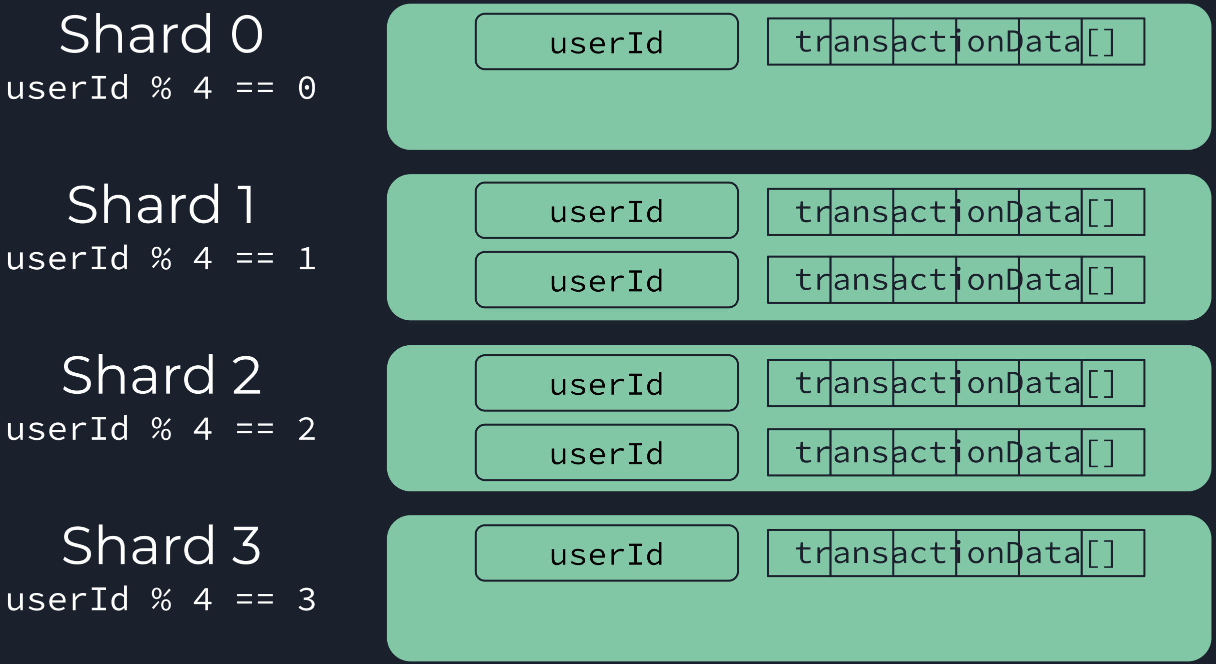
(simultaneous work by different users would be a great help)

Sharding

Sharding is a horizontal partitioning of data



Sharding



Shard 0

`userId % 4 == 0`

userId

transactionData[]

Shard 1

`userId % 4 == 1`

userId

transactionData[]

userId

transactionData[]

Shard 2

`userId % 4 == 2`

userId

transactionData[]

userId

transactionData[]

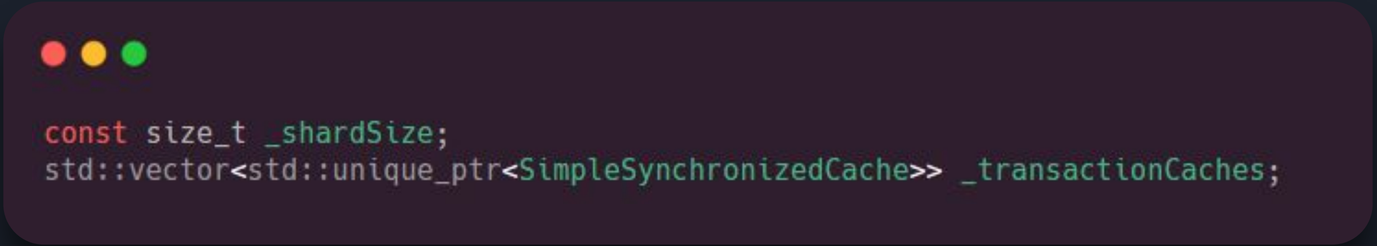
Shard 3

`userId % 4 == 3`

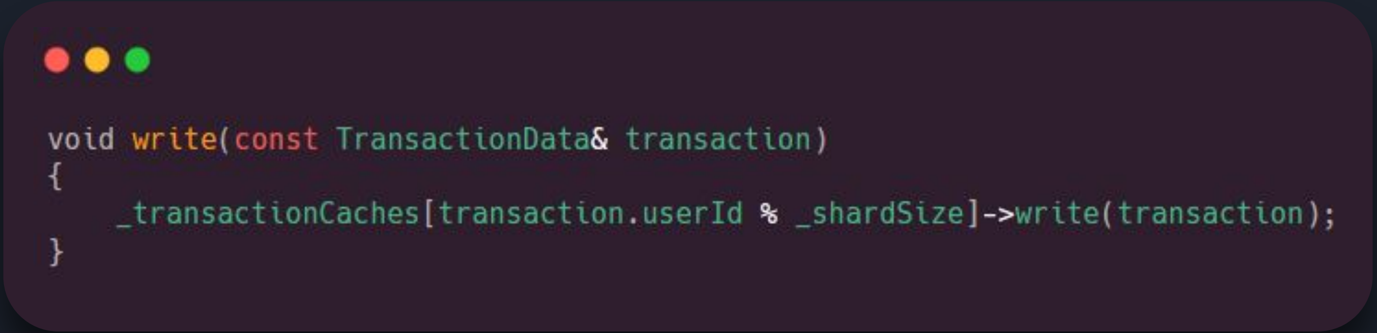
userId

transactionData[]

Sharding

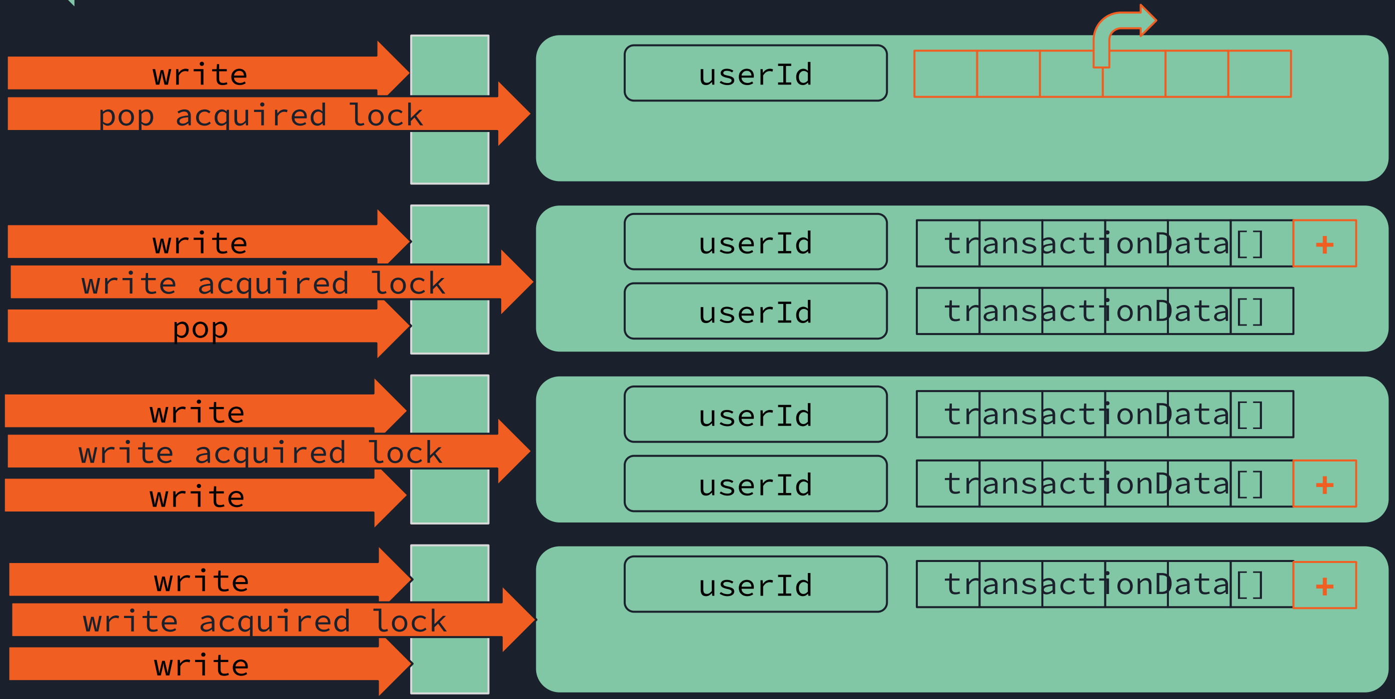


```
const size_t _shardSize;
std::vector<std::unique_ptr<SimpleSynchronizedCache>> _transactionCaches;
```

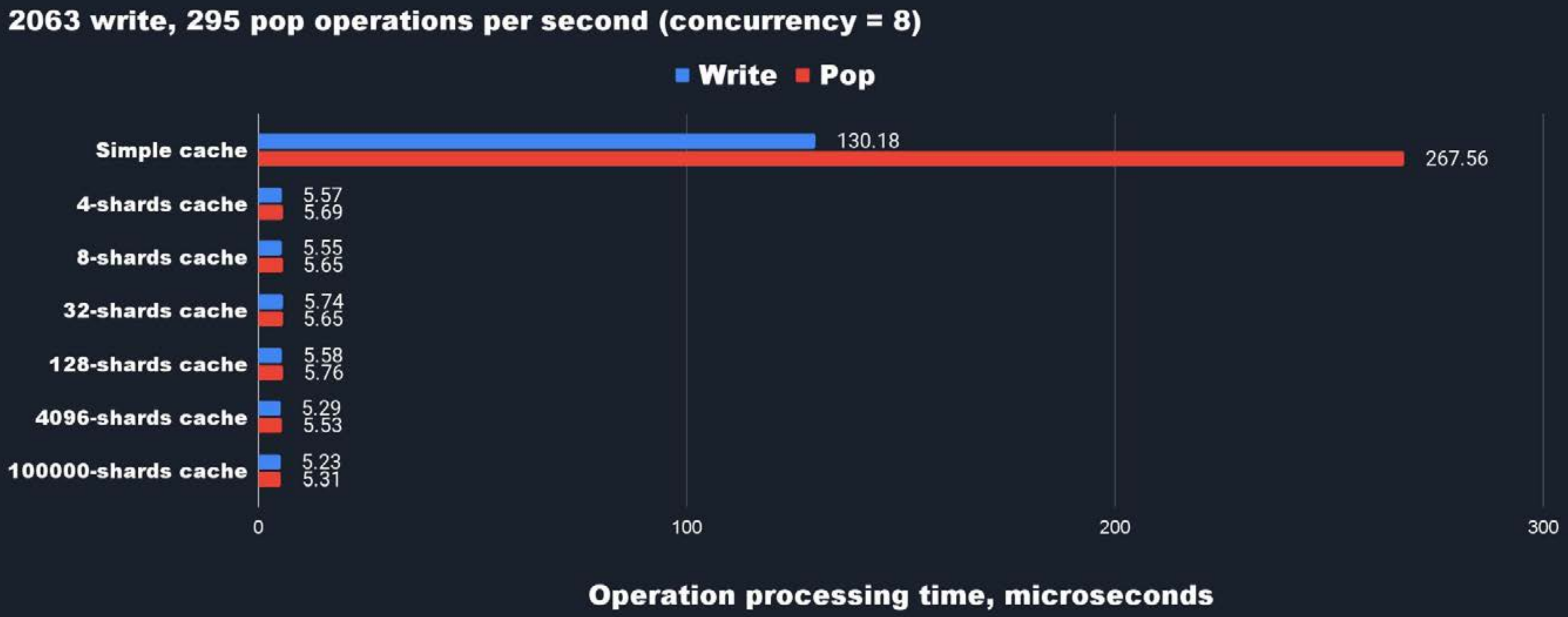


```
void write(const TransactionData& transaction)
{
    _transactionCaches[transaction.userId % _shardSize]->write(transaction);
}
```

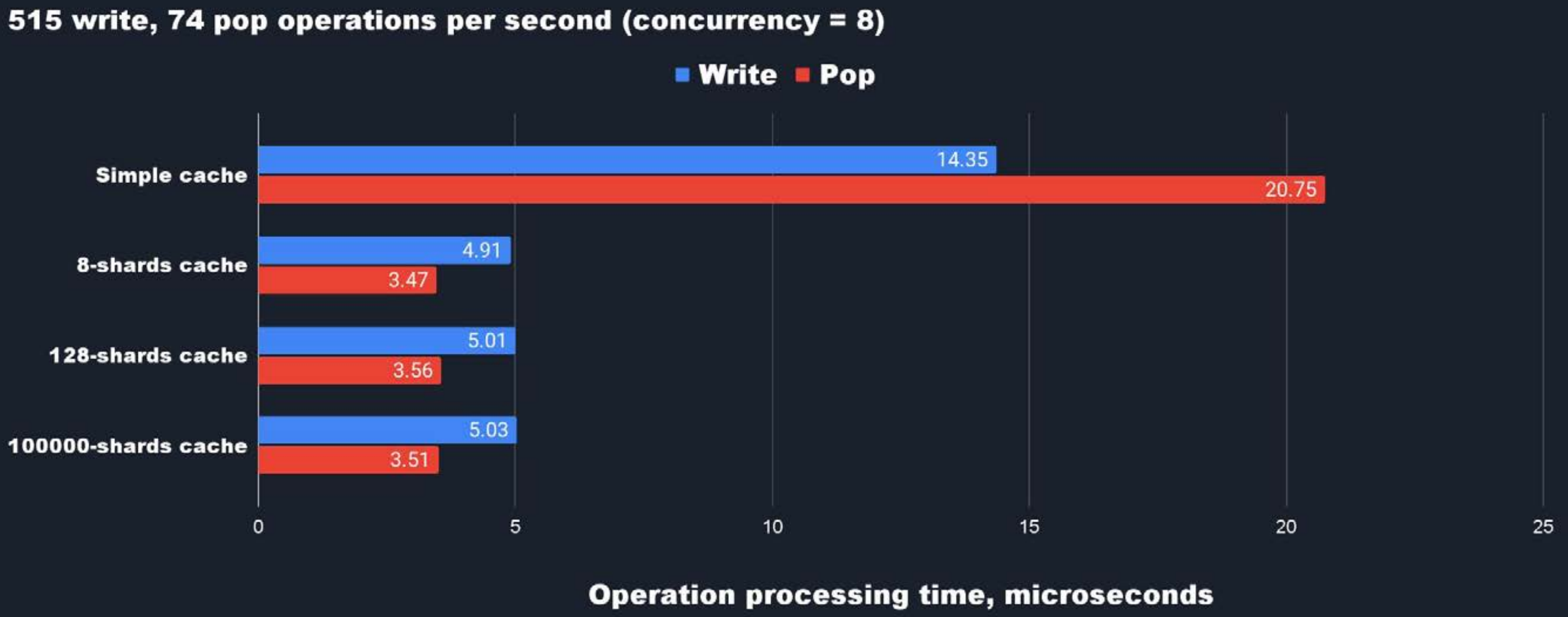
Sharding


write
pop acquired lock
userId
write
write acquired lock
pop
userId
transactionData[]
+
userId
transactionData[]
write
write acquired lock
write
userId
transactionData[]
userId
transactionData[]
+
write
write acquired lock
write
userId
transactionData[]
+

Sharded cache vs simple cache



Sharded cache vs simple cache



Key takeaways

- Sharding can be used in optimization of write-heavy multithreaded environments
- Analyzing the data structure can provide insights in ways to optimize synchronization
- Premature optimization isn’t useful — it’s crucial to know the expected load

Thank you!

Discussed cache implementations and tests

alex-iskh/ShardedCache

Questions?

aiskh

alex_iskh@outlook.com