HighLoad++ FOUNDATION

Трек Яндекс

YDB: мультиверсионность в распределенной базе данных

Андрей Фомичев,
руководитель YDB

Содержание

01 Обзор YDB

02 Архитектура YDB за 5 минут

03 Распределенные транзакции в YDB

04 Что можно сделать лучше?

05 Что такое MVCC и Snapshot Isolation?

06 Версии строк в YDB

07 Распределенные снапшоты в YDB

08 Исследование производительности

09 Выводы

01

Обзор YDB

YDB

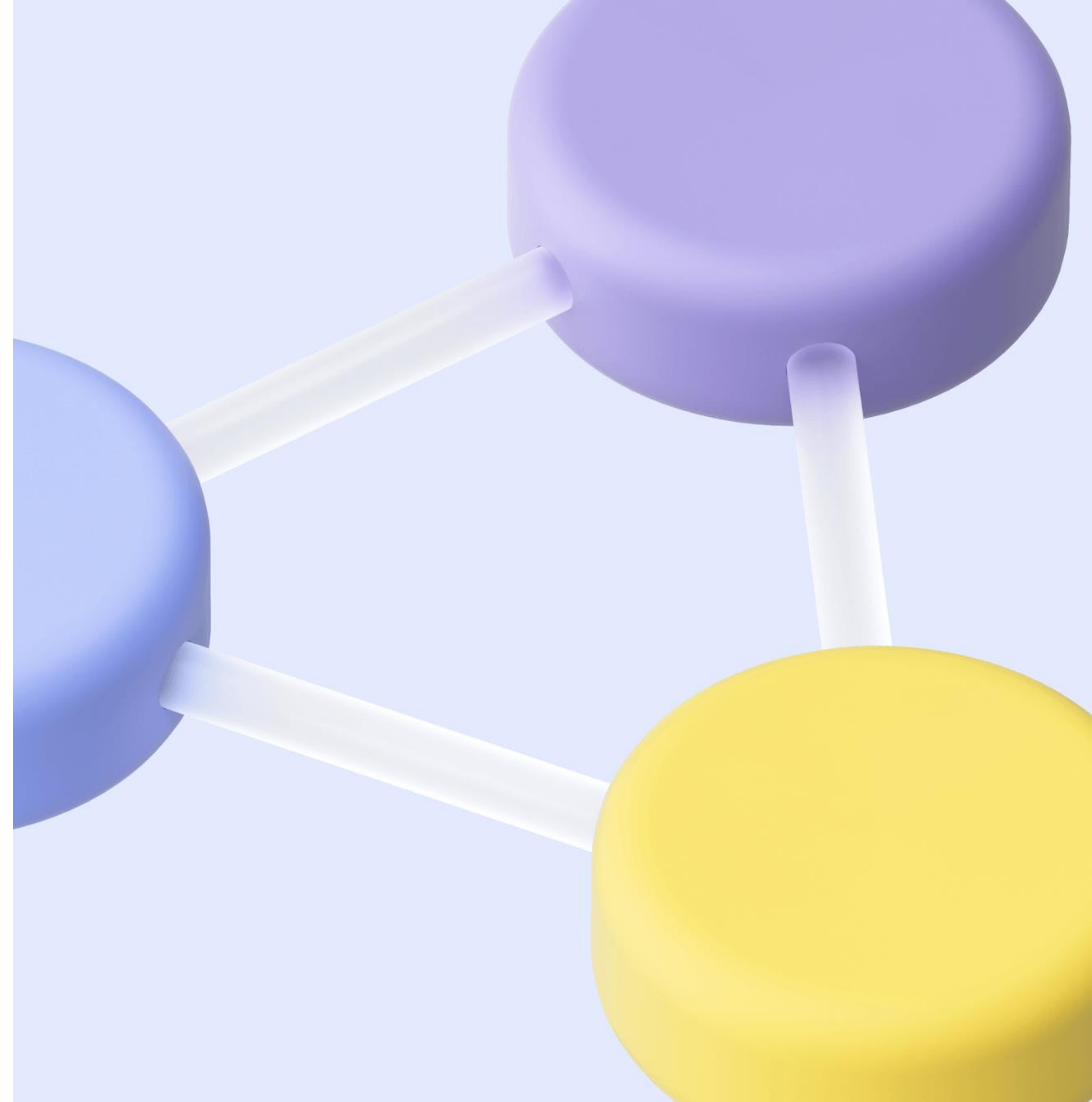
YDB – Open-Source Distributed SQL Database

Distributed SQL означает:

- реляционная СУБД
- работает на кластере
- строгая консистентность

Open-Source

- Apache 2.0 License
- https://github.com/ydb-platform/ydb



Факты про YDB

Consistency & Serializable transaction execution

- CAP-теорема, выбираем CP
- Serializable уровень изоляции транзакций

Highly available

- Работает в нескольких зонах доступности (дата-центрах)
- Выживает после отключения зоны доступности и стойки в другой зоне доступности, не требует участия человека, сохраняет доступность на чтение/запись

Mission critical database

- Подходит для проектов, требующих доступности 24x7
- Не требует наличия окна обслуживания (maintenance window)

Преимущественно OLTP-нагрузка

- Поколоночное хранение, ETL в разработке

Платформа

- Персистентные очереди, сетевые диски, хранение временных рядов и т.п.

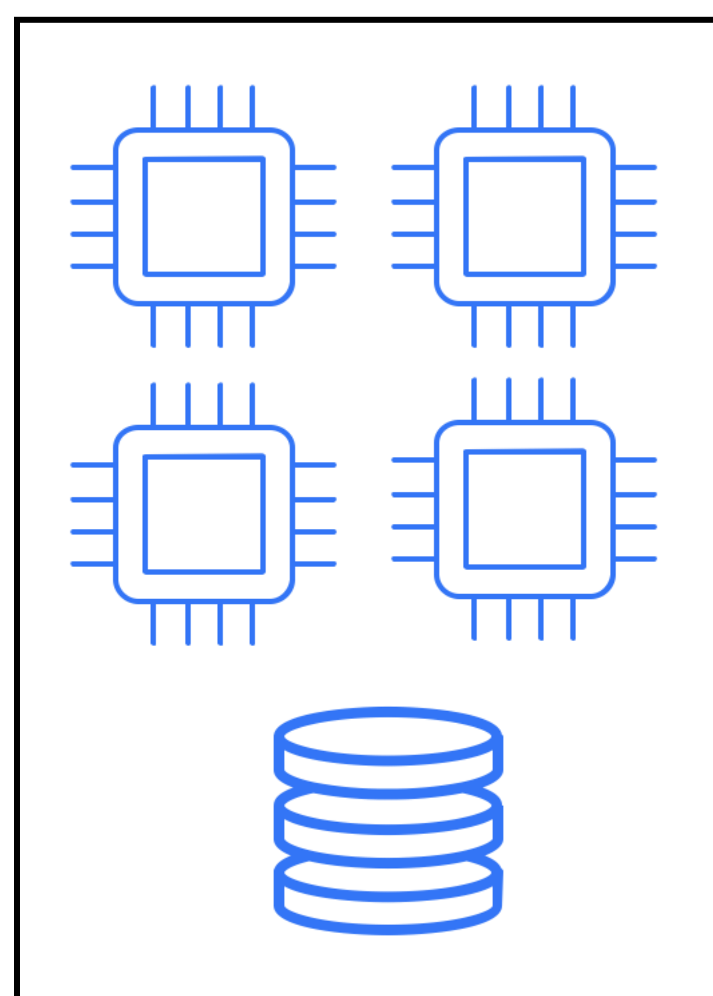
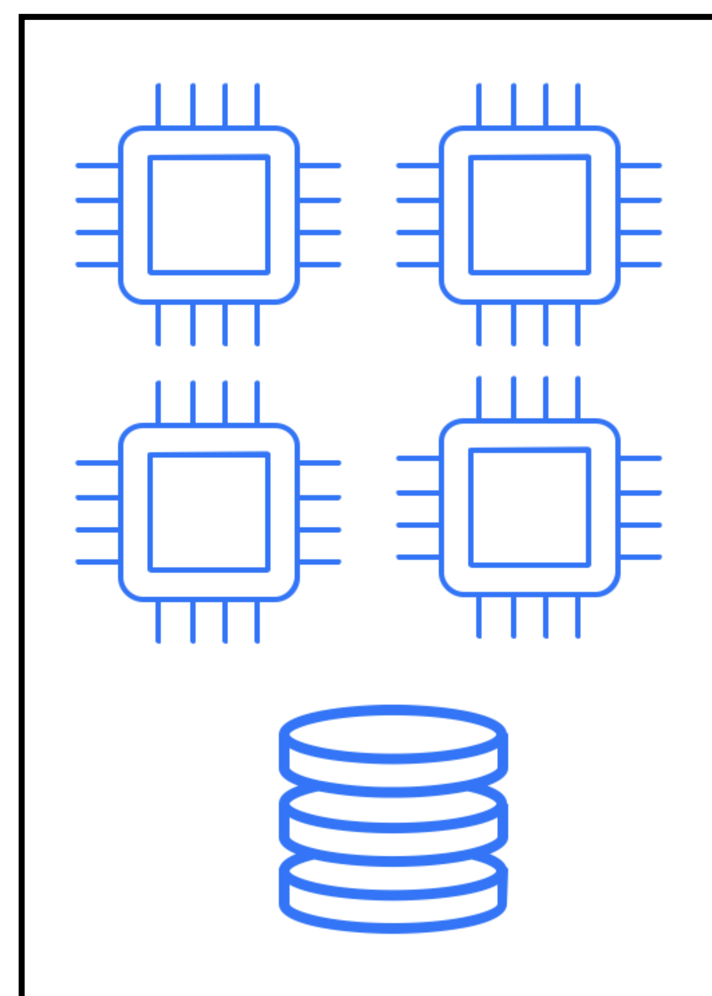
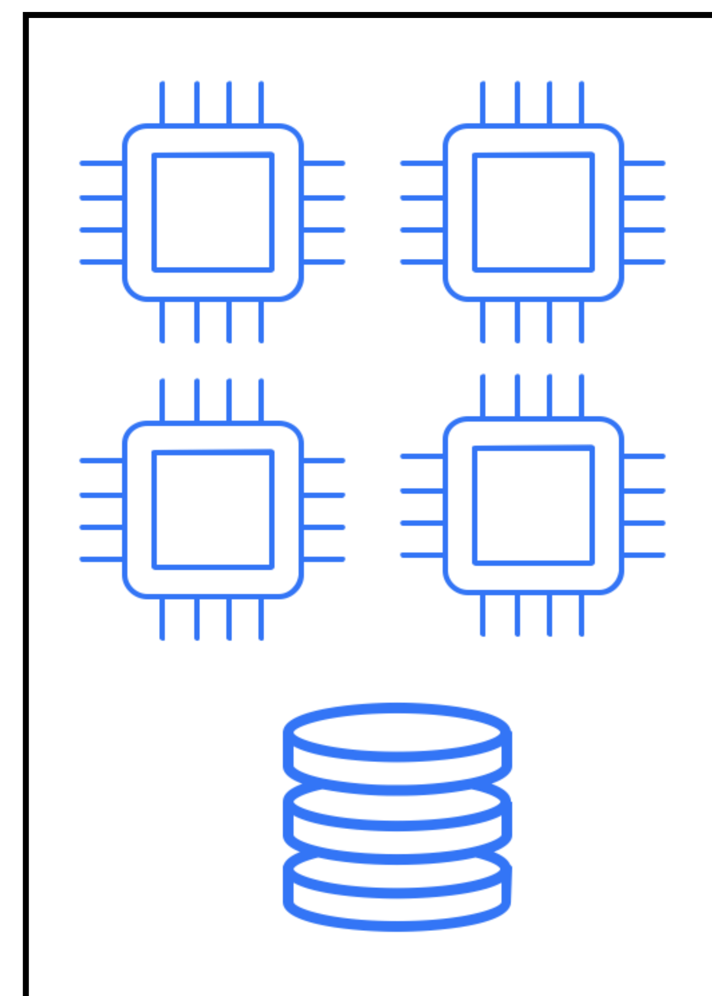
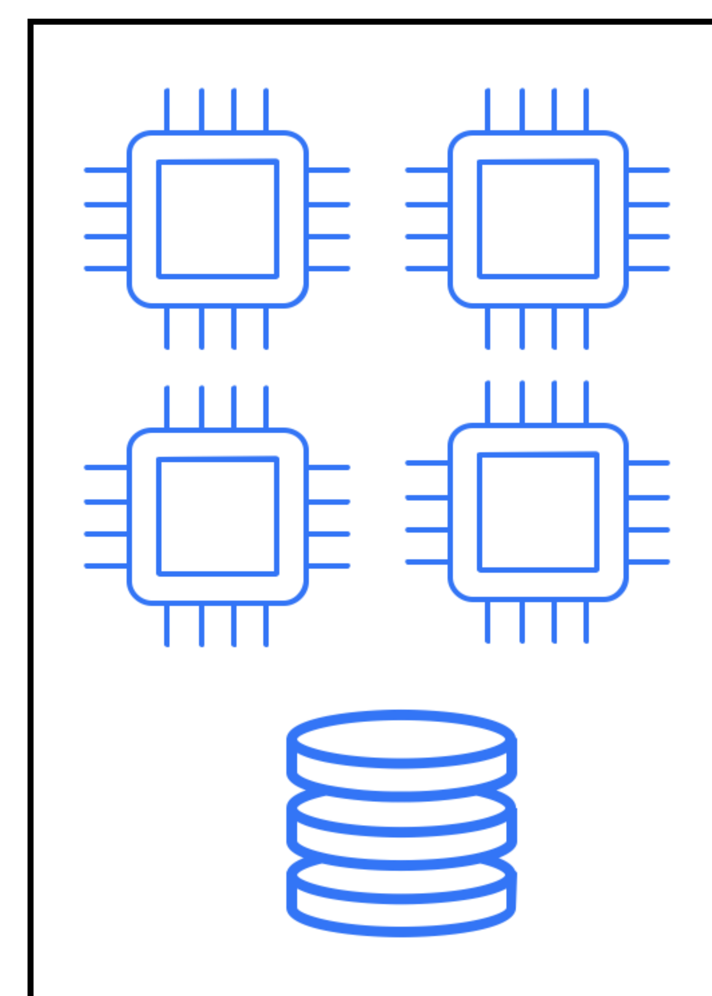
HL 2022 HighLoad++ FOUNDATION
Трек Яндекс

02

Архитектура YDB за 5 минут

Архитектура Share Nothing

Кластер физических или виртуальных машин, архитектура share nothing, commodity hardware

HL 2022 HighLoad++ FOUNDATION
Трек Яндекс

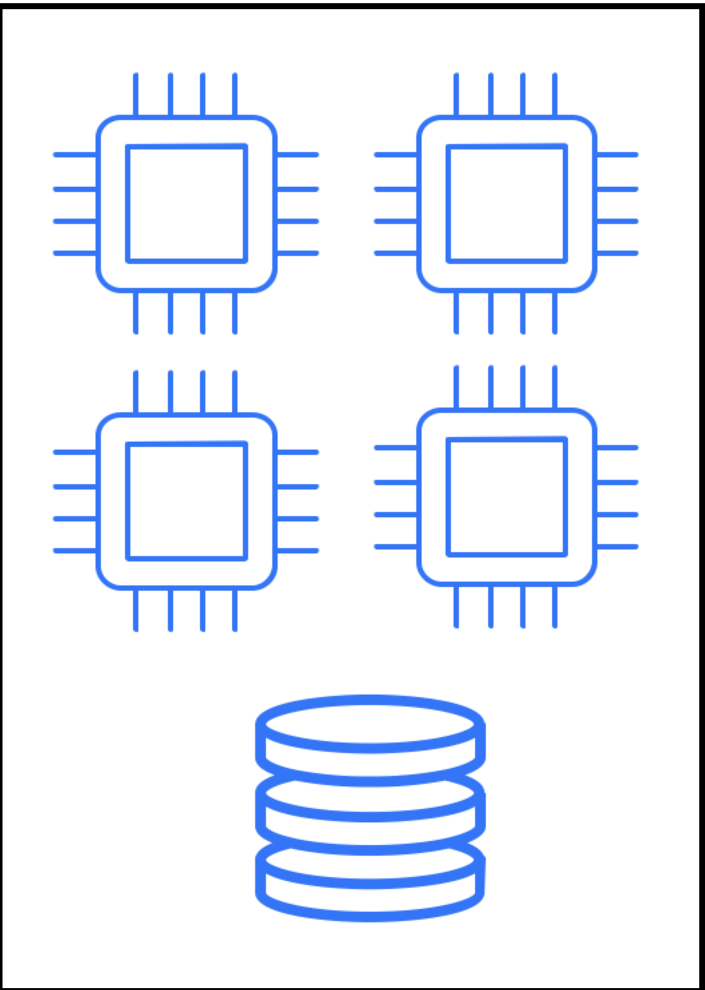
Таблицы и запросы

SQL query

Кластер занимается как хранением данных, так и обработкой пользовательских запросов

Id	Value1	Value2
GX008	8 921	1 114
GX278	827	9
GY045	654	345
SK720	3 445	3 456
SM527	7 668	7 643
UA628	72	3 928

Key	Data
82	8 921
283	827
346	654
1273	3 445



HL 2022 HighLoad++ FOUNDATION
Трек Яндекс

Партицирование таблиц

Id	Value1	Value2
GX008	8 921	1 114
GX278	827	9
GY045	654	345
SK720	3 445	3 456
SM527	7 668	7 643
UA628	72	3 928

DataShard Tablet: GX008, GX278
DataShard Tablet: GY045, SK720
DataShard Tablet: SM527, UA628

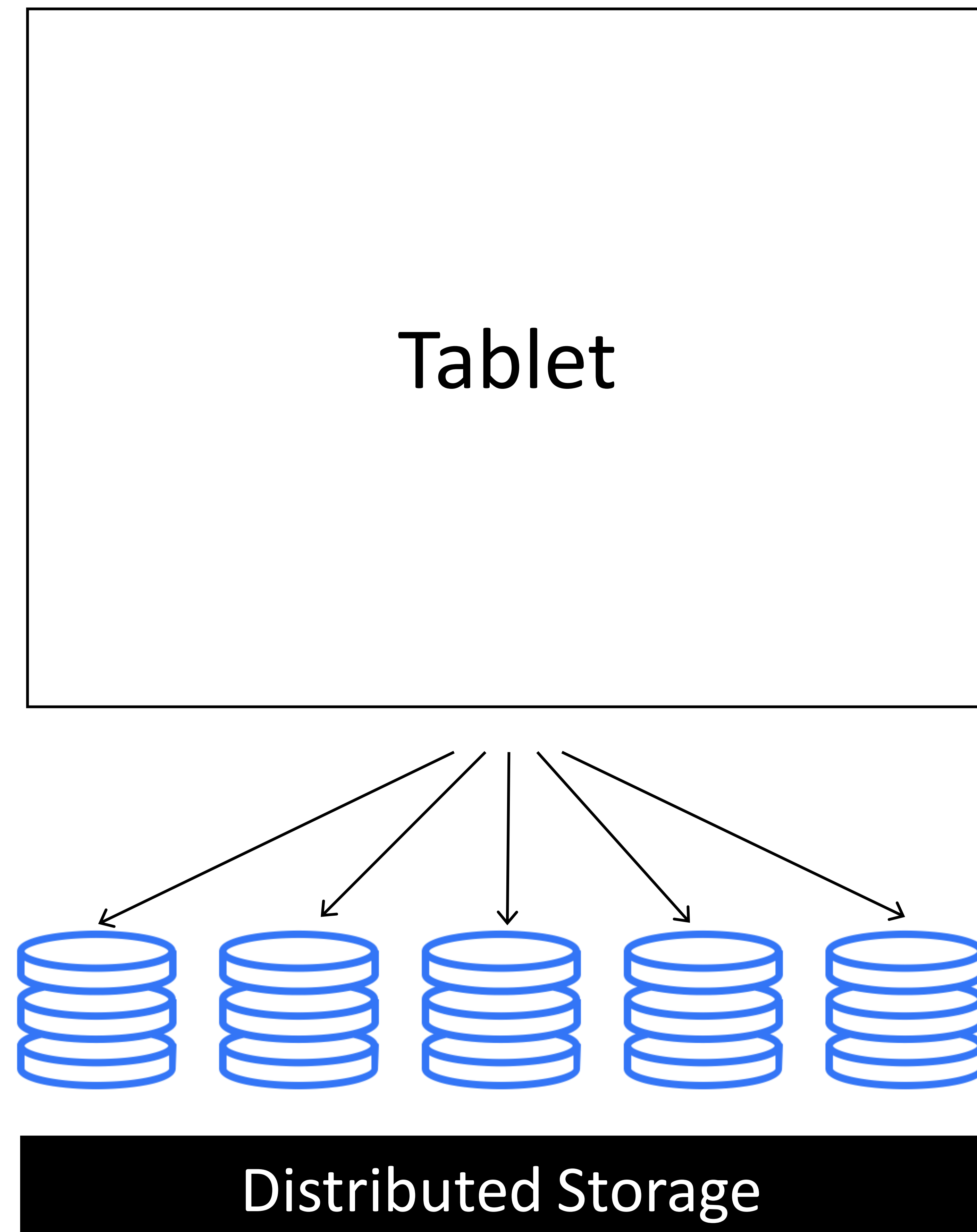
Key	Data
82	8 921
283	827
346	654
1273	3 445

DataShard Tablet: 82, 283
DataShard Tablet: 346, 1273

Пользовательские таблицы разбиваются на партиции, за данные партиции отвечает Таблетка (Tablet)

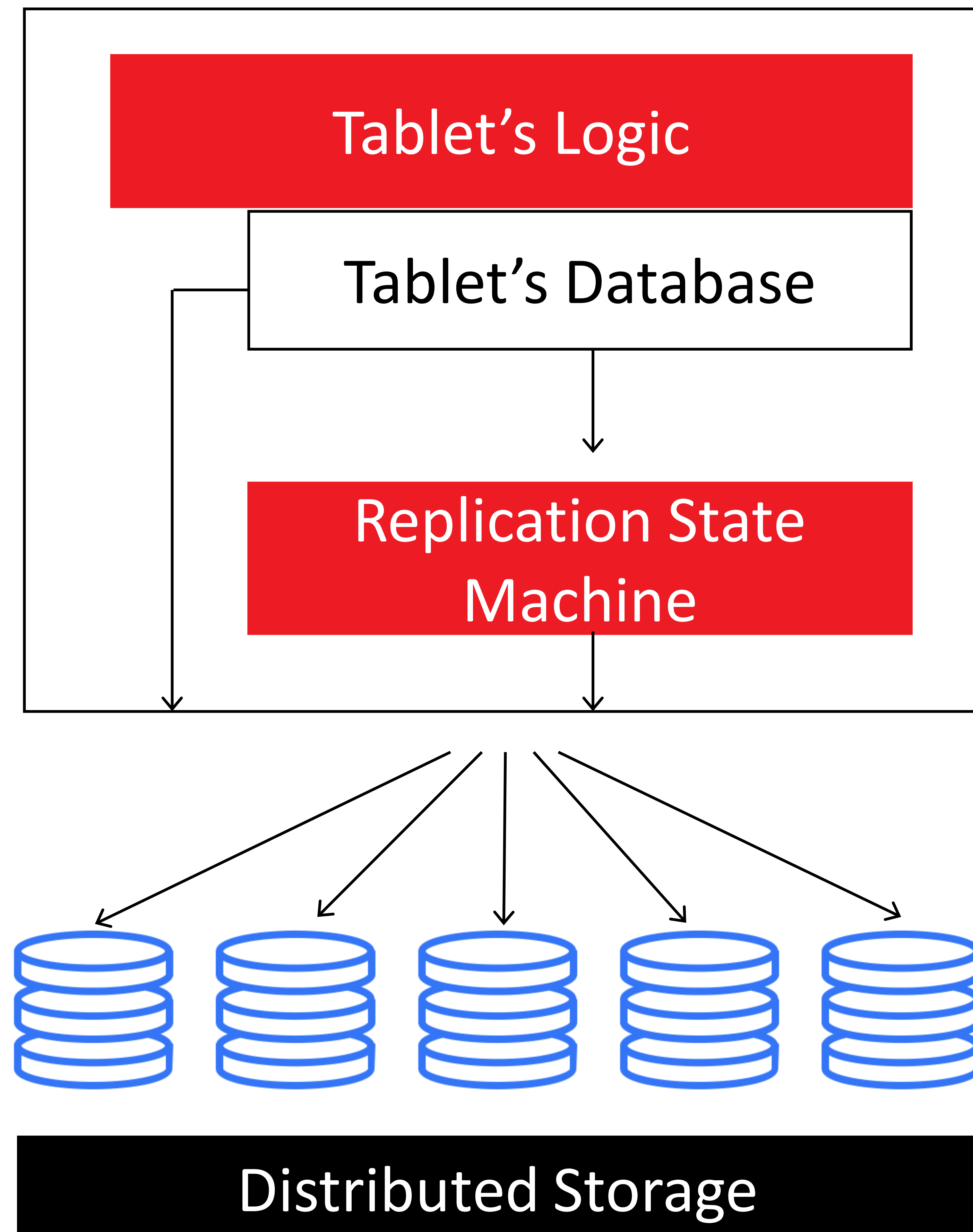
HL 2022 HighLoad++ FOUNDATION
Трек Яндекс

Внутри Tablet (1)



- Tablet – это набор объектов C++, работающих вместе, отвечающих за сегмент данных
- Tablet – ядерная часть YDB
- Tablet предоставляет API для вышележащего уровня, например, для обработчика запросов: 1. insert row; 2. delete row; 3. read row
- Можно думать о Tablet как об адаптере к данным, хранящимся в Distributed Storage
- Tablet обычно имеет волатильные данные, которые можно потерять, например, кэши
- Объект Tablet может умереть и переподняться в том же состоянии на другой машине из лога изменений, хранящегося в Distributed Storage
- На операциях изменения данных Tablet пишет изменения в лог прежде, чем ответить ОК клиенту

Внутри Tablet (2)

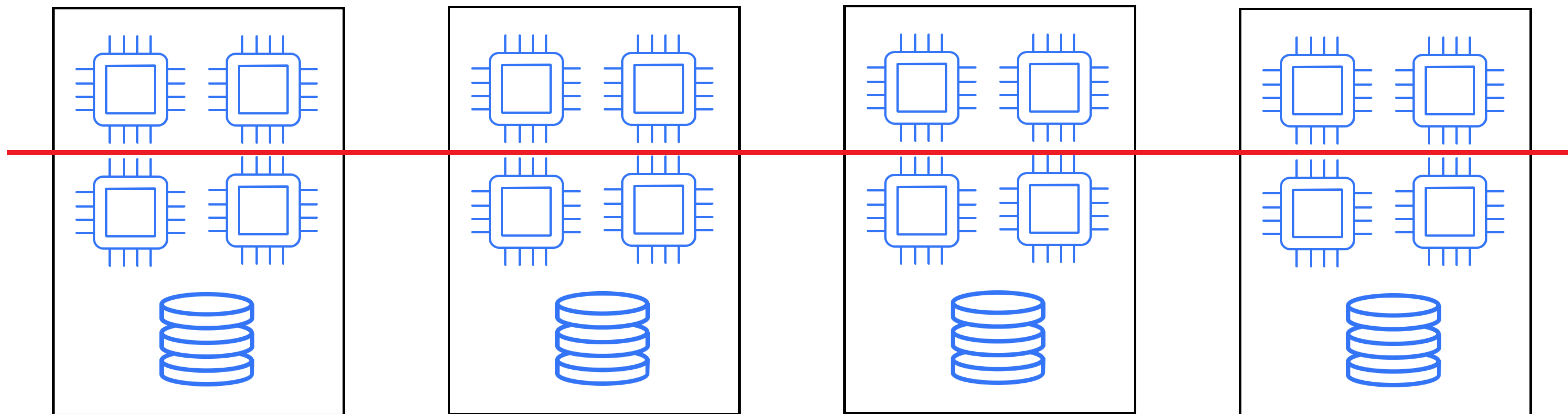


- Replication State Machine (RSM)
1. Пишет изменения в лог в Distributed Storage
2. Восстанавливает свое состояние из лога при падении
3. Предоставляет гарантии, аналогичные RAFT и Paxos
- Tablet's Database
1. Данные хранятся в виде LSM-дерева (Log Structured Merge tree)
2. Реализует ACID-гарантии для данных, за которые отвечает
- Tablet's Logic специфична для типа Tablet
1. Может предоставлять разные API наверх (например, работа с данными или метаданными)
2. Может быть активной сущностью, например, балансировать Tablets по кластеру
- Distributed storage предоставляет надежное хранение данных с избыточностью

HL 2022 HighLoad++ FOUNDATION
Трек Яндекс

Разделение слоев Compute и Storage

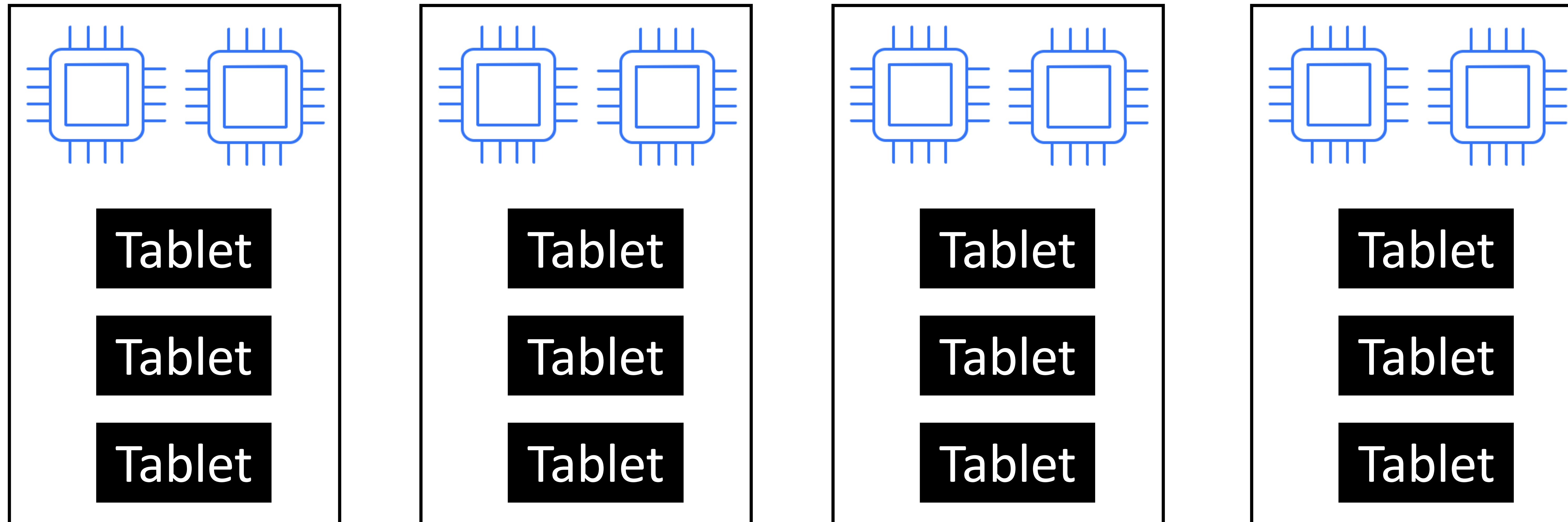
Слои хранения данных и вычислений в YDB разделены, что позволяет растить вычислительные мощности и слой хранения независимо



HL 2022 HighLoad++ FOUNDATION
Трек Яндекс

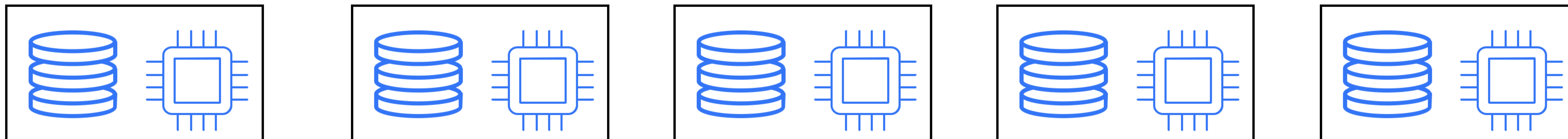
Разделение слоев Compute и Storage

compute nodes

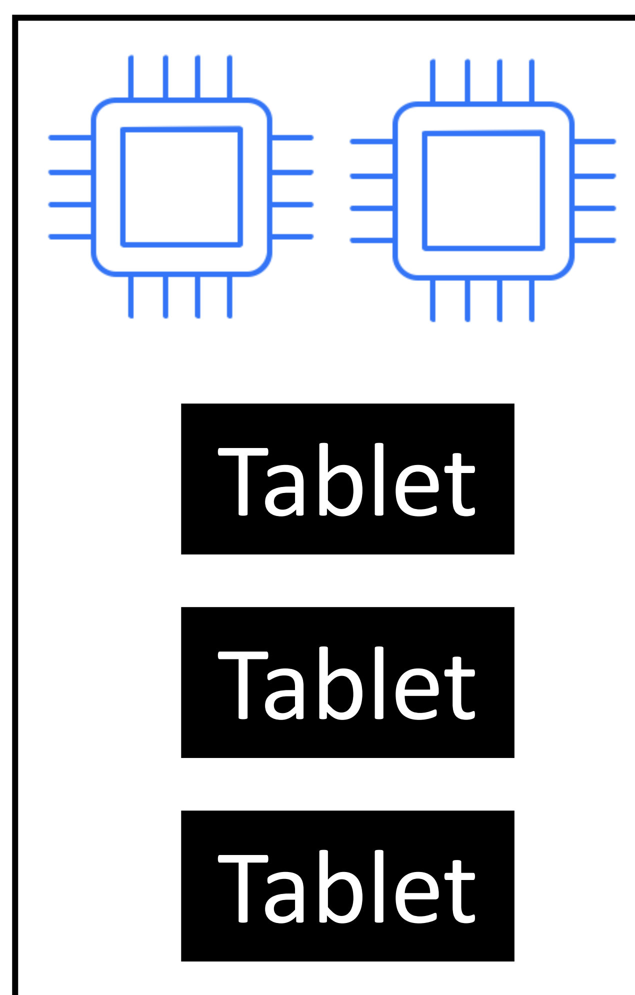


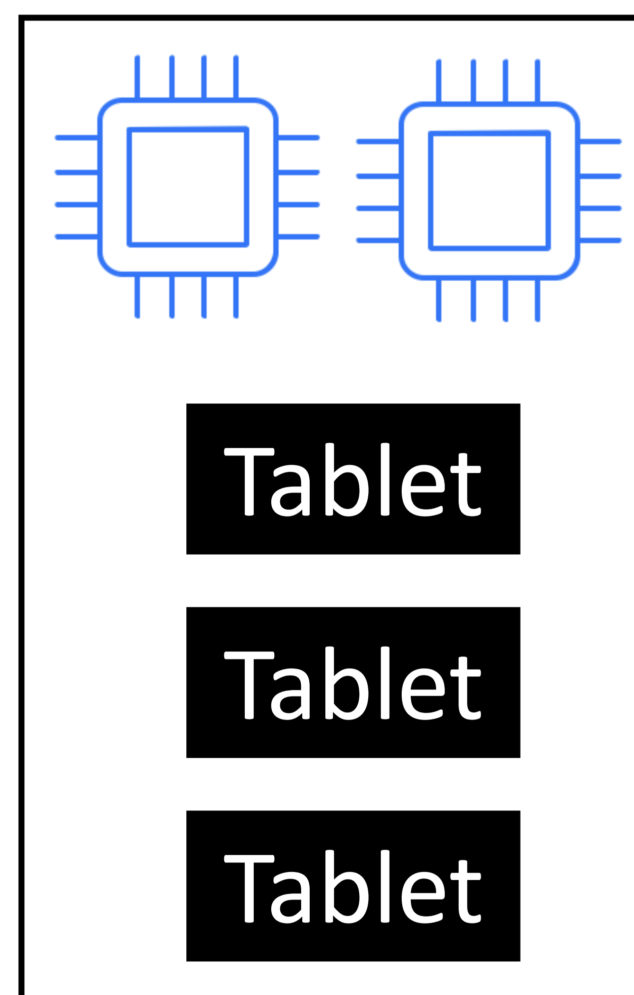
Среды выполнения (Runtime) для Tablets и для выполнения запросов запущены в вычислительных узлах (compute nodes)

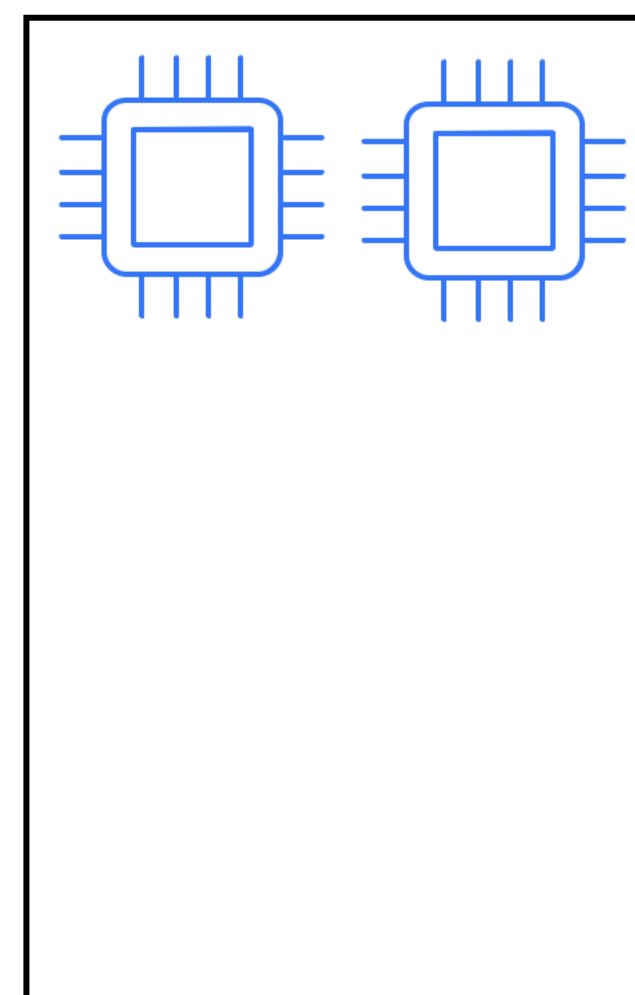
storage nodes



Tablet Balancing







Tablets могу свободно перемещаться между узлами.

Балансировка Tablets очень важна для равномерной утилизации CPU на кластере.

Клиентская балансировка пользовательских сессий выполняет ту же роль для выполнения запросов.

Types of Tablets

DataShard

+ Отвечает за данные партиций пользовательских таблиц

SchemeShard

+ Отвечает за метаданные пользовательских таблиц

Hive

+ Занимается запуском и балансировкой Tablets по кластеру

BlobStorage Controller

+ Отвечает за метаданные Distributed Storage, через него осуществляется управление Distributed Storage

Coordinator and Mediator

+ Участвуют в планировании распределенных транзакций

Etc…

03

Распределенные транзакции в YDB

Пример распределенной транзакции

DataShard Tablet
DataShard Tablet
DataShard Tablet

Id	Value1	Value2
GX008	8 921	1 114
GX278	827	9
GY045	654	345
SK720	3 445	3 456
SM527	7 668	7 643
UA628	72	3 928

Key	Data
82	8 921
283	827
346	654
1273	3 445

DataShard Tablet
DataShard Tablet

```
UPDATE table1 SET Value1=3845 WHERE Id="GY045";
UPDATE table2 SET Data=Data+1 WHERE Key=346;
COMMIT;
```

Как выполнять распределенные транзакции?

2PC (Two-phase Commit)

- Наиболее распространенный протокол выполнения распределенных транзакций
- Недостаток: низкая пропускная способность при конфликтах

YDB использует протокол Calvin

- *Calvin: Fast Distributed Transactions for Partitioned Database Systems by Daniel J. Abadi, Alexander Thomson*
- Calvin позволяет выполнять детерминистические транзакции (of deterministic transactions) без блокировок и конфликтов
- Calvin не может выполнить произвольную транзакцию, которая выражается SQL-запросом, поэтому выполнение транзакций в YDB – больше, чем Calvin-протокол

Что такое детерминистическая транзакция?

Детерминистическая транзакция знает набор ключей, по которым выполняется чтение/запись

```
read A
read B
write C = value(A)+value(B)
```

Что такое детерминистическая транзакция?

Детерминистическая транзакция знает набор ключей, по которым выполняется чтение/запись

```
read A
read B
write C = value(A)+value(B)
```

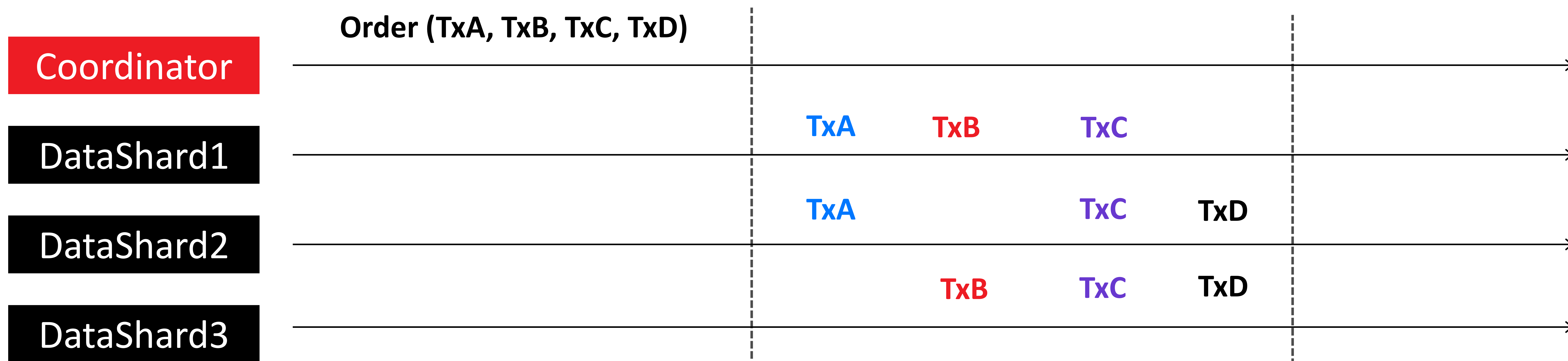
Не все транзакции являются детерминистическими. Пример такой транзакции:

```
read A
read value(A)
read B
write C = value(value(A))+value(B)
```

Как Calvin выполняет детерминистические транзакции?

Транзакции на входе: TxA(DS1, DS2), TxB(DS1, DS3), TxC(DS1, DS2, DS3), TxD(DS2,DS3).

Calvin: если Coordinator упорядочит входящие транзакции, то **не будет конфликта** между транзакциями и мы получим serializable isolation

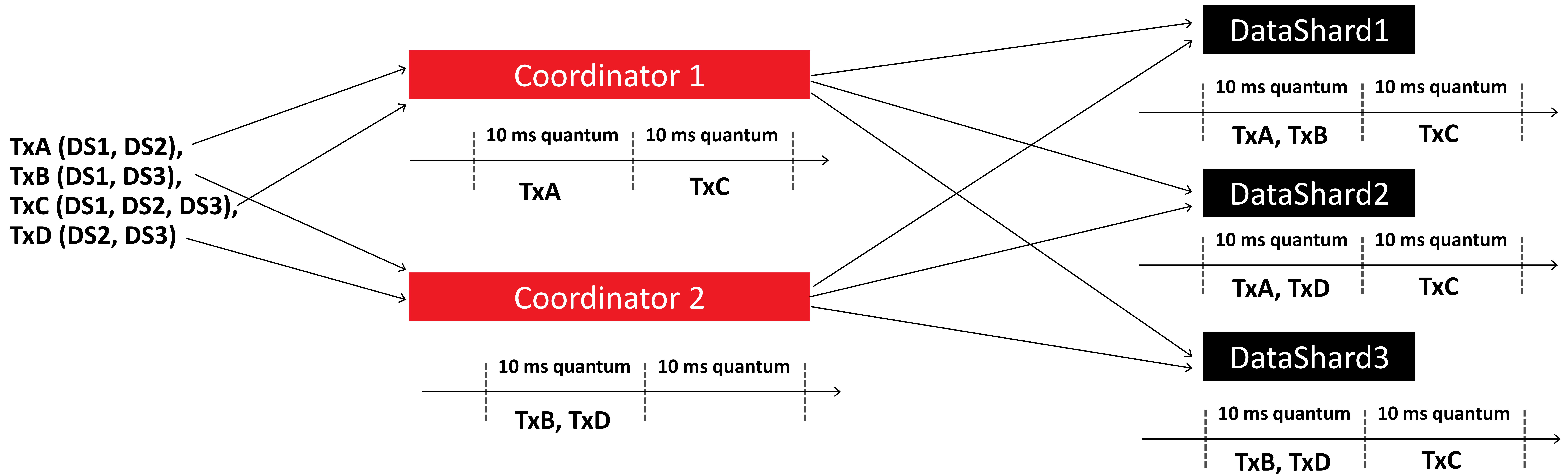


Как YDB выполняет распределенные транзакции?

YDB использует протокол Calvin, но дополнительно решает следующие проблемы:

+ Один координатор для планирования транзакций может быть узким местом
+ Как выполнять транзакции, которые не являются детерминистическими, но исключительно важны в практическом смысле

Система из нескольких координаторов в YDB



HL 2022 HighLoad++ FOUNDATION
Трек Яндекс

Система из нескольких координаторов в YDB

TxA (DS1, DS2),
TxB (DS1, DS3),
TxC (DS1, DS2, DS3),
TxD (DS2, DS3)

Coordinator 1

10 ms quantum | 10 ms quantum

TxA | TxC

Coordinator 2

10 ms quantum | 10 ms quantum

TxB, TxD

DataShard1

10 ms quantum | 10 ms quantum

TxA, TxB | TxC

DataShard2

10 ms quantum | 10 ms quantum

TxA, TxD | TxC

DataShard3

10 ms quantum | 10 ms quantum

TxB, TxD | TxC

Each transaction gets:

`PlanStep – coordinator quantum`

`TxId – unique id`

HL 2022 HighLoad++ FOUNDATION

Трек Яндекс

Многошаговые транзакции в YDB

Вспомним пример транзакции, которая не является детерминистической

```
read A
read value(A)
read B
write C = value(value(A))+value(B)
```

```
1. LOCK(A)
2. LOCK(value(A))
2. LOCK(B)
3. write(C) if LOCKs are not broken
```

YDB использует блокировки (LOCKs) между несколькими шагами выполнения транзакции. Сами шаги являются детерминистическими транзакциями. Блокировки являются оптимистичными.

04

Что можно сделать лучше?

Что можно сделать лучше?

Проблемы с многошаговыми читающими транзакциями

+ Блокировки проверяются на коммите, поэтому мы не знаем о сломанных блокировках (LOCKs) до окончания выполнения запроса
+ Пользователю приходится вызывать commit для исключительно читающих транзакций, иначе нельзя гарантировать непротиворечивость прочитанного
+ Читающие транзакции могут получать ошибку TLI (Transaction Locks Invalidated)
+ Распределенные читающие транзакции планируются координатором, что добавляет latency и требует записи на диск (таков механизм планирования транзакций)

MVCC (Multiversion Concurrency Control) – метод, который позволяет решить проблемы, но

+ MVCC дает накладные расходы на хранение версий данных и на работу с ними
+ Взятие глобального снапшота в распределенной СУБД может быть нетривиально и дорого, надо не испортить текущие сценарии

05

Что такое MVCC (Multiversion Concurrency Control) и Snapshot Isolation?

Multiversion Concurrency Control (MVCC)

Изменения в базе данных версионируются.

Один из способов – к таблицам добавляются системные колонки CreatedVer и DeletedVer, в которых хранится идентификатор транзакции, которая сделала изменение.

key	value
123	“Alice”

key=123	value=“Alice”	CreatedVer=15	DeletedVer=nil

```
15: insert key=123, value="Alice"
```

key=123	value=“Alice”	CreatedVer=15	DeletedVer=29

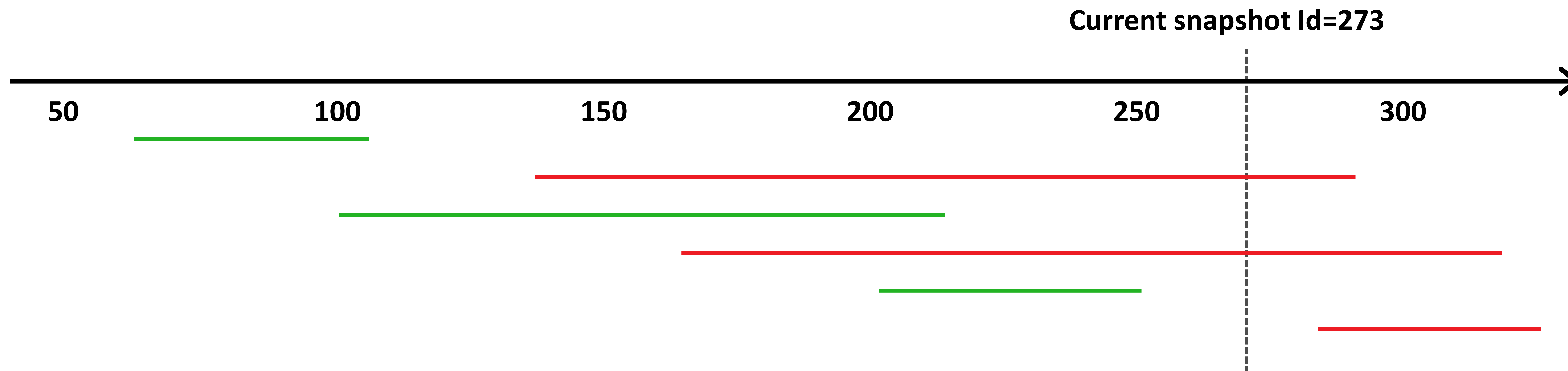
```
29: delete key=123
```

key=123	value=“Alice”	CreatedVer=15	DeletedVer=29
key=123	value=“Bob”	CreatedVer=47	DeletedVer=nil

```
47: insert key=123, value="Bob"
```


HL 2022 HighLoad++ FOUNDATION
Трек Яндекс

MVCC: Snapshot Reads



Новая транзакция берет снапшот
- максимум из закоммиченных идентификаторов транзакций
- идентификаторы выполняющихся транзакций

Снапшот контролирует, какие строки видны транзакции.

HL 2022 HighLoad++ FOUNDATION
Трек Яндекс

Преимущества MVCC

Конфликты read-write исключены!

- MVCC предоставляет возможность читать из базы данных консистентно без конфликтов с пишущими транзакциями

MVCC позволяет реализовать уровень сериализации транзакций Snapshot Isolation

- *“Snapshot isolation is a guarantee that all reads made in a transaction will see a consistent snapshot of the database, and the transaction itself will successfully commit only if no updates it has made conflict with any concurrent updates made since that snapshot.” (Wikipedia)*
- Популярный (и максимальный) уровень изоляции транзакций во многих СУБД, считается практичным решением

Isolation in Database Systems

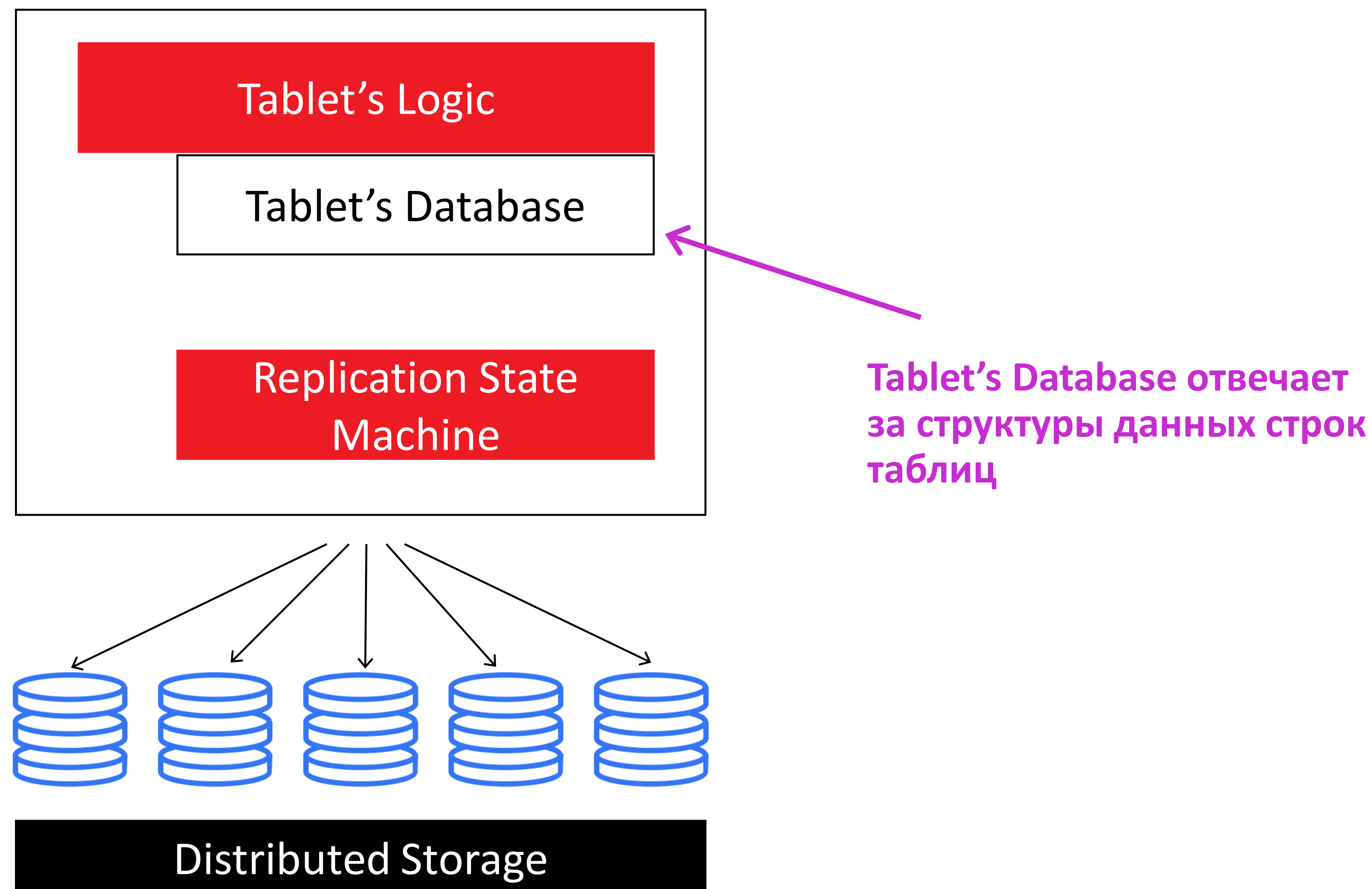
	Dirty read	Unrepeatable read	Lost updates	Phantoms	Write skew
Read uncommitted	✓	✓	✓	✓	✓
Read committed	✗	✓	✓	✓	✓
Repeatable read	✗	✗	✗	✓	✓
Snapshot isolation	✗	✗	✗	✗	✓
Serializable	✗	✗	✗	✗	✗
Strict Serializable	Serializable + Linerializable				

HL 2022 HighLoad++ FOUNDATION
Трек Яндекс

06

Версии строк в YDB

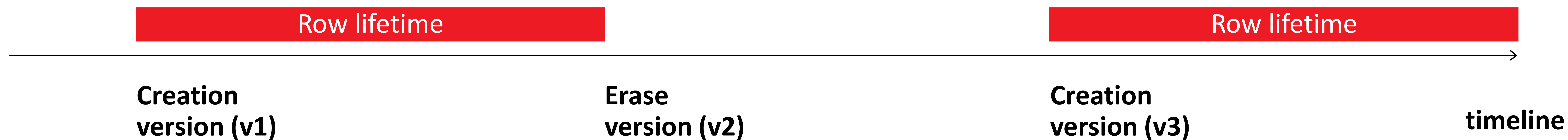
Добавляем версии строк в Tablet's Database



Tablet's Database отвечает за структуры данных строк таблиц

HL 2022 HighLoad++ FOUNDATION
Трек Яндекс

Жизненный цикл строк таблицы в YDB

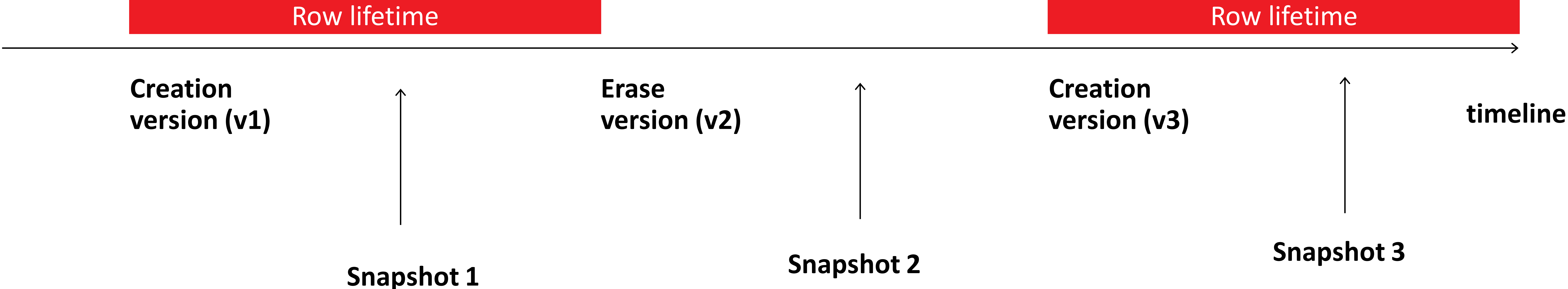


- Каждая строка в базе имеет 2 версии

1. Версия создания
2. Версия удаления

HL 2022 HighLoad++ FOUNDATION
Трек Яндекс

Жизненный цикл строк таблицы в YDB



- Каждая читающая (потенциально распределенная) транзакция берет снапшот базы данных в момент начала выполнения

Версия создания/удаления строки

```
struct TRowVersion {
        ui64 PlanStep; // coordinator time in ms
        ui64 TxId; // transaction id, that is unique for distributed transactions
        friend constexpr bool operator<(const TRowVersion& a, const TRowVersion &b) {
                return a.PlanStep < b.PlanStep ||
                        (a.PlanStep == b.PlanStep && a.TxId < b.TxId);
        }
};
```

TRecord – структура данных строки

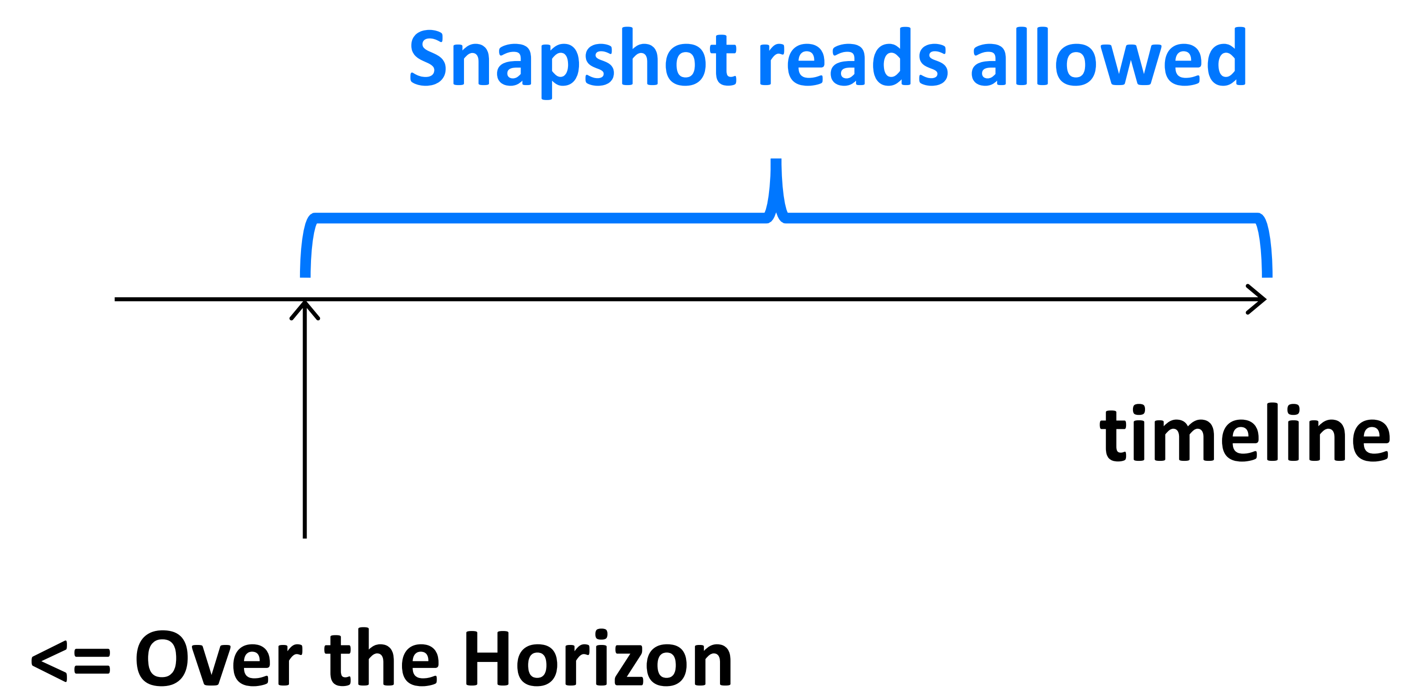
Erased	Version	History		RowOperation	Fixed Size Data	TRowVersion	TRowVersion	Variable Size Data
1 byte header					Size depends on table schema	Optional 16 bytes	Optional 16 bytes	

Интересные факты

- Мультиверсионирование можно включать и выключать на базе => можем
1. выбирать оптимальные настройки на базе
2. сравнивать производительность
- Чтение данных последней версии на мультиверсионированной базе стоит столько же, сколько без MVCC
1. Чтение предыдущих версий требует похода в history block

HL 2022 HighLoad++ FOUNDATION
Трек Яндекс

”За горизонтом”



- Мы ограничиваем то, насколько глубоко в прошлом можно читать данные
- Таким образом нет необходимости хранить версии данных «за горизонтом»
- Если в таблице большинство данных записано «давно», то YDB не тратит дисковое пространство на хранение старых версий (после compaction)

Оценка Disk Overhead на мультиверсионирование

+ До 32 bytes дополнительно на строку
1. 16 bytes на версию создания
2. 16 bytes на версию удаления
3. Если строка не удалена, то мы не тратим место на версию удаления
+ Дополнительно хранятся предыдущие версии строки
1. Размер этих строк зависит от паттерна нагрузки на базу
+ Версии строк «за горизонтом» не хранятся (после compaction)
1. Версии строк хранятся для недавно записанных/удаленных строк

Что насчет автоматического и ручного VACUUM?

- YDB хранит данные в виде LSM-дерева
- Это означает, что на данных периодически запускается compaction
- Compaction в том числе подчищает старые версии данные и удаляет версии строк, которые нет необходимости хранить
- В YDB есть фоновый compaction, он запускается периодически, если не срабатывают условия по дисбалансу LSM-дерева
- Можно запустить compaction вручную
- Эти процедуры обеспечивают аналог VACUUM

07

Распределенные снапшоты в YDB

Взятие снапшота БД для Read Tx

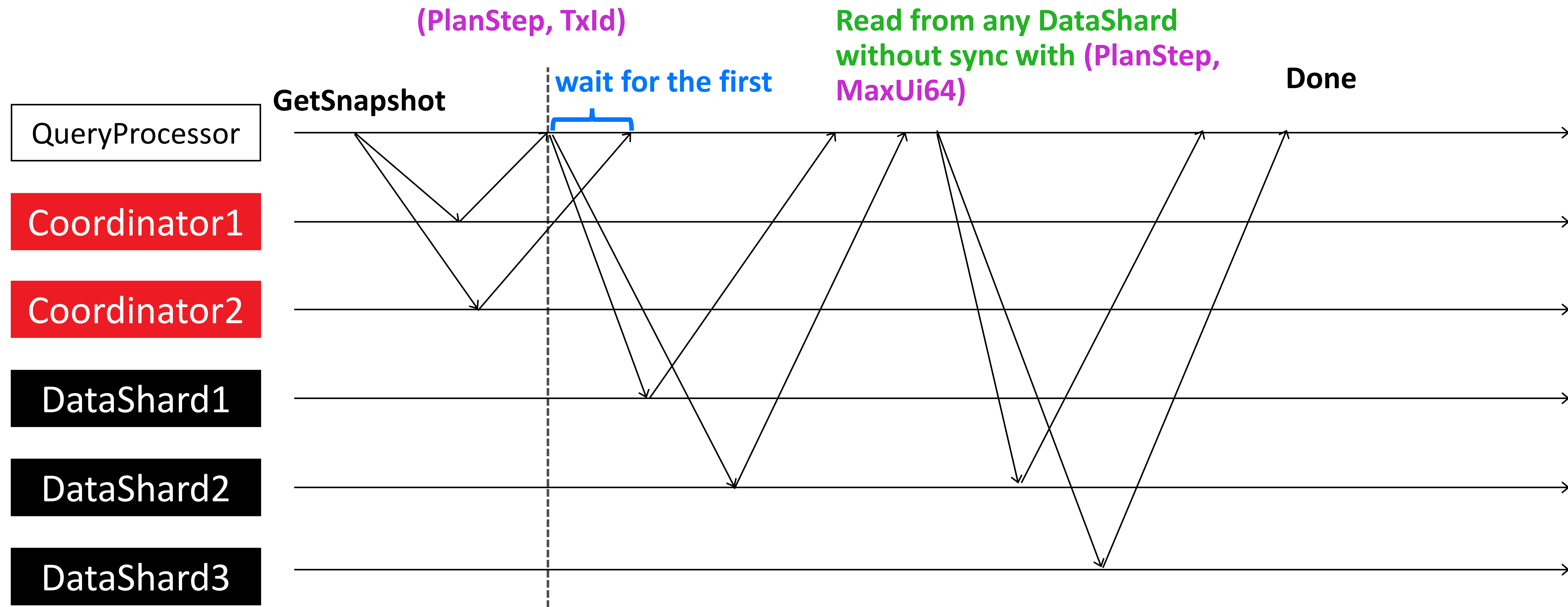
Традиционные реляционные СУБД обычно имеют глобальный Sequencer

- Инкрементируется по приходу новой транзакции
- Легко взять текущее значение из оперативной памяти

В распределенных СУБД все сложнее

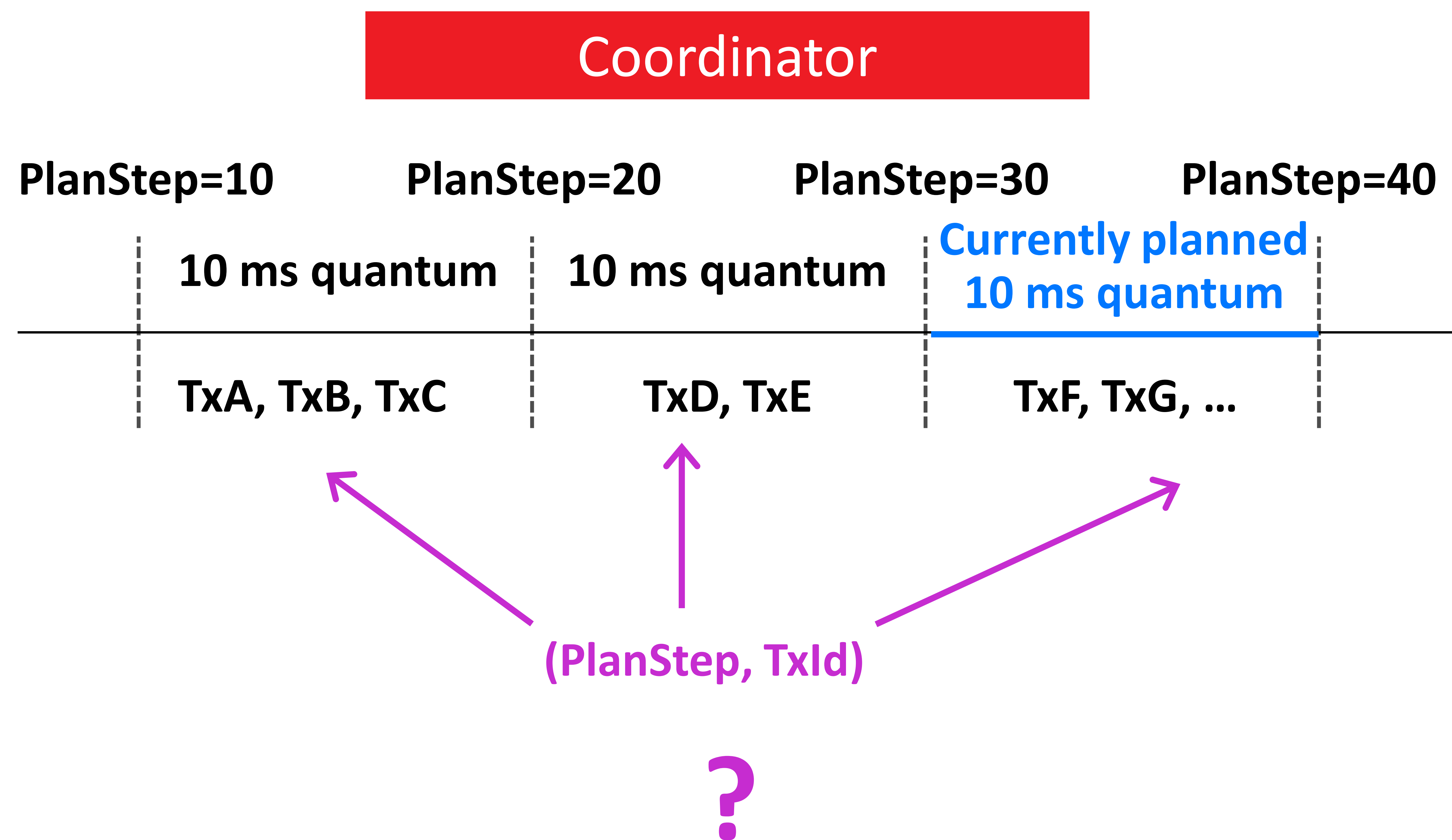
- В YDB нет понятия единого sequencer'a (это плохо масштабируется)
- Координаторы – это те сущности, в которых есть понятие «времени» и его продвижения

Взятие и использование снапшота БД для Read Tx



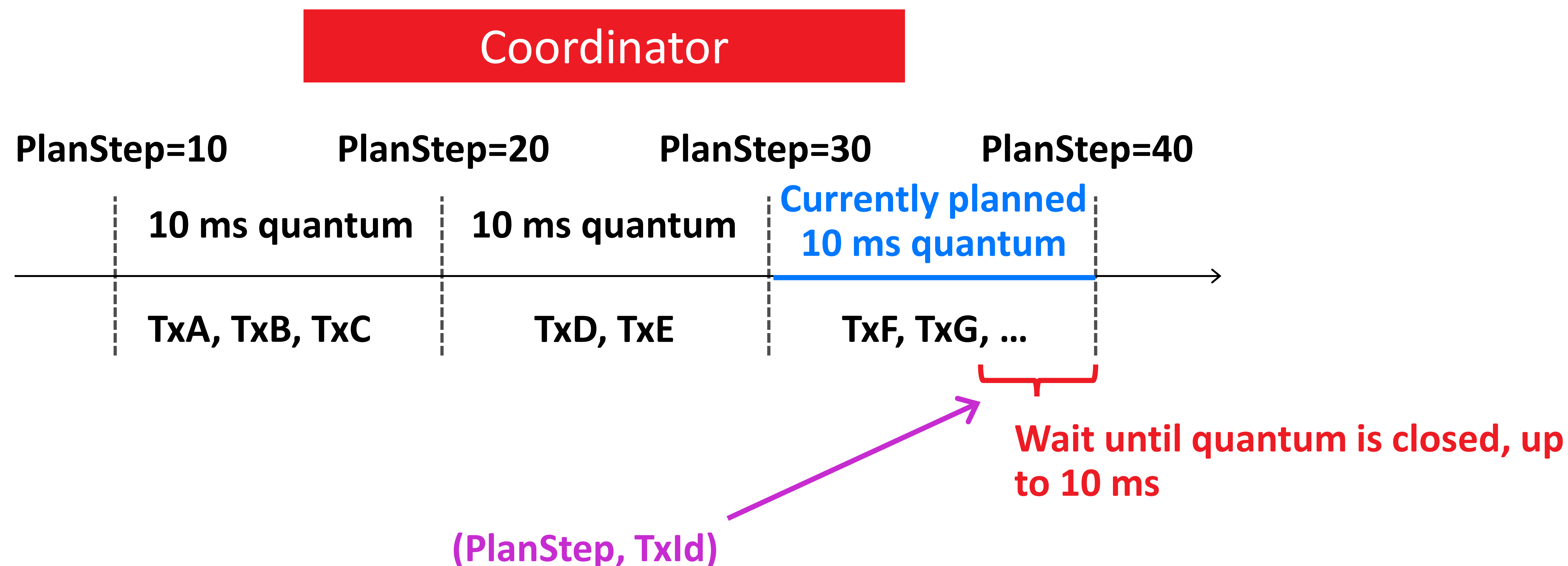
HL 2022 HighLoad++ FOUNDATION
Трек Яндекс

Как Coordinator отвечает на GetSnapshot?



HL 2022 HighLoad++ FOUNDATION
Трек Яндекс

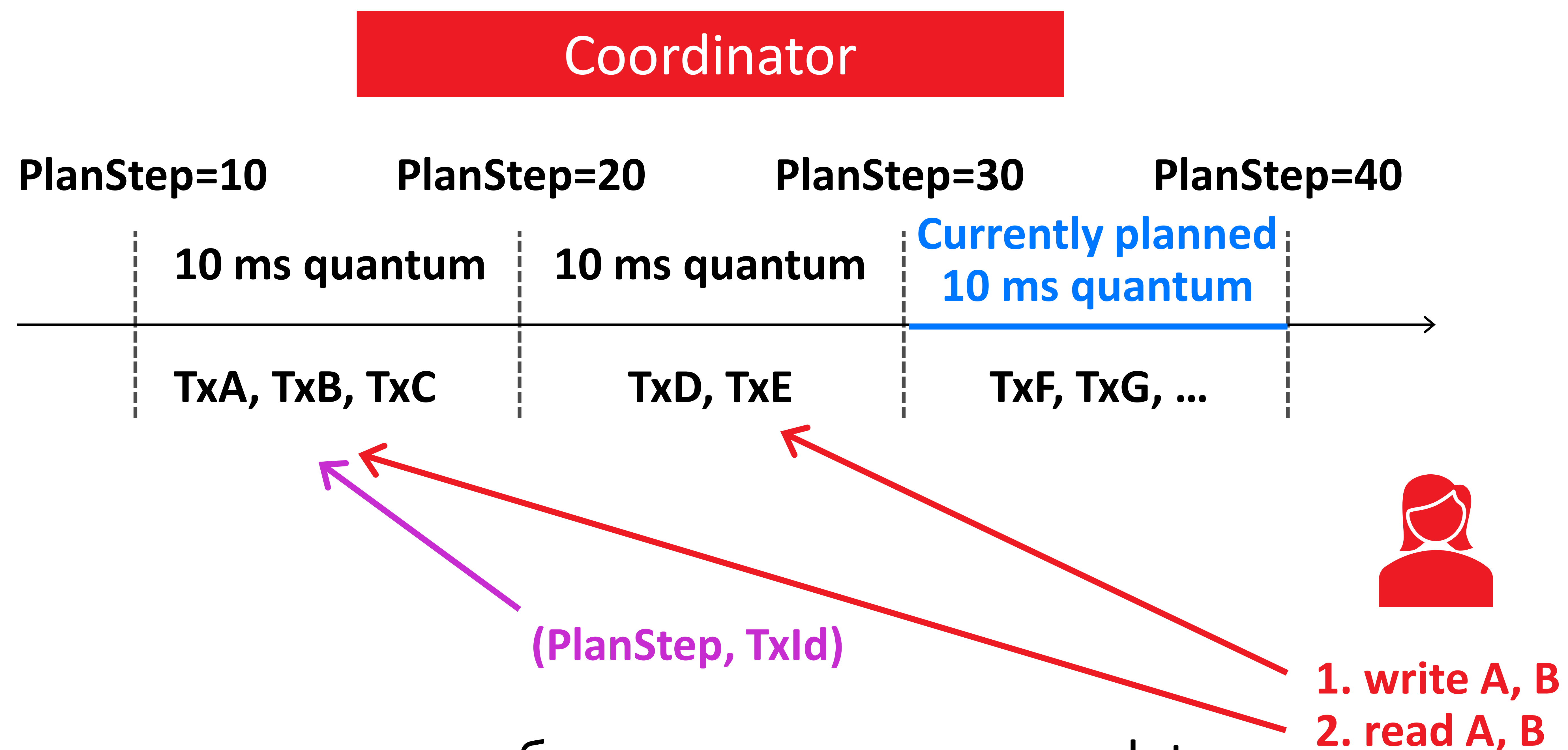
Как Coordinator отвечает на GetSnapshot?



Взятие снапшота может быть запланировано как обычная распределенная транзакция.

Недостаток: пенальти по latency вплоть до периода планирования

Как Coordinator отвечает на GetSnapshot?

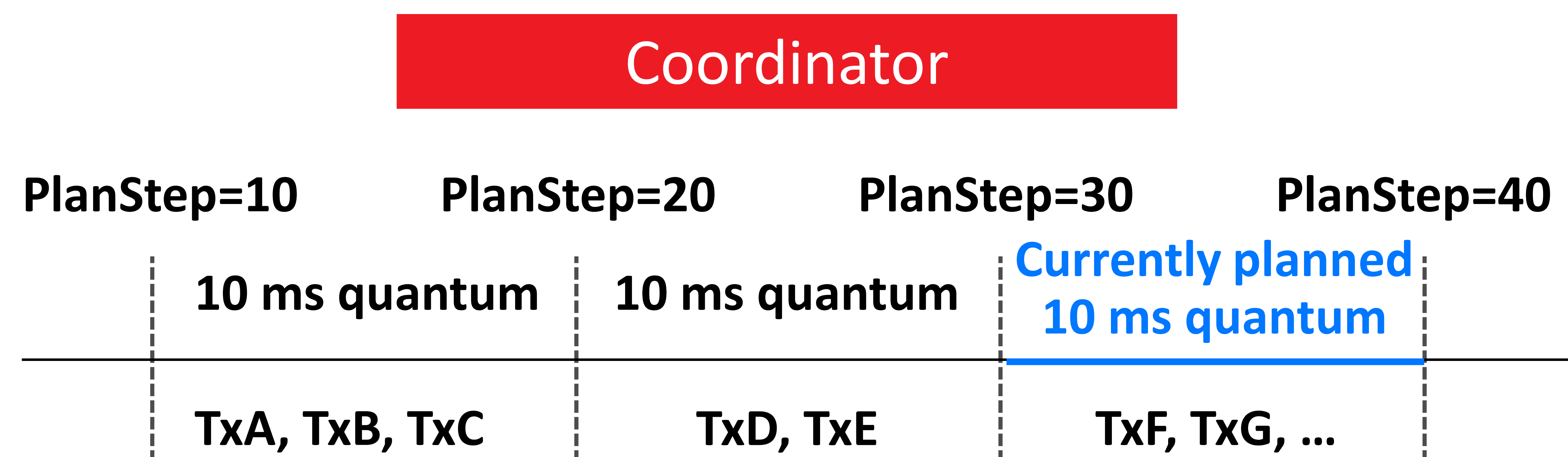


Взятие снапшота в далеком прошлом избавляет от пенальти по latency.

Однако может нарушать свойство ‘read your writes’.

HL 2022 HighLoad++ FOUNDATION
Трек Яндекс

Как Coordinator отвечает на GetSnapshot?



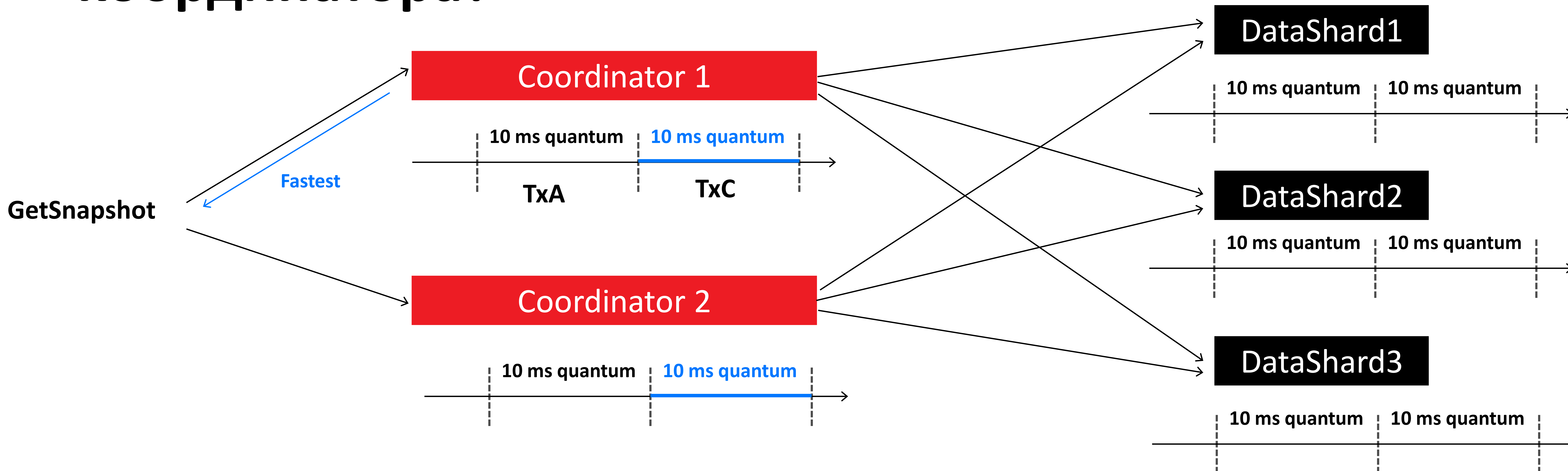
(PlanStep, TxId) = (LastCommittedPlanStep, Max<ui64>())

+ Гарантирует, что данные в снапшоте не поменяются
+ Гарантирует свойство ‘read your writes’, а именно: транзакции

1. либо еще в полете
2. либо закоммичены с (PlanStep, TxId) < (LastCommittedPlanStep, Max<ui64>())

HL 2022 HighLoad++ FOUNDATION
Трек Яндекс

Почему достаточно ждать только быстрейшего координатора?



Настенные часы могут отличаться у Coordinator1 и Coordinator2.

Однако все DataShards дожидаются ответа от _всех_ координаторов.

HL 2022 HighLoad++ FOUNDATION
Трек Яндекс

Оптимизация Immediate Reads and Writes

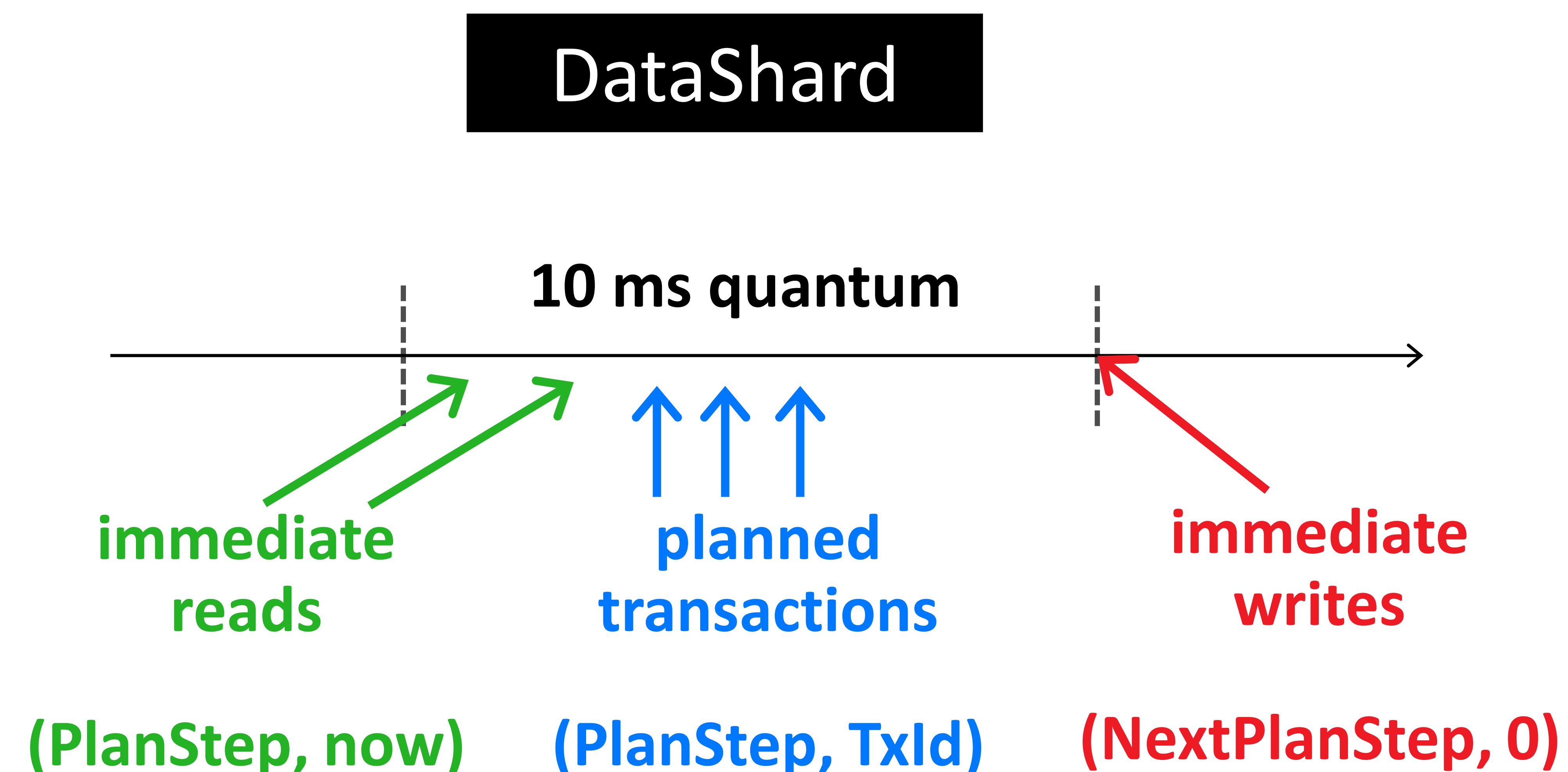
- В YDB есть оптимизация для транзакций, которые затрагивают только один DataShard
- Одношардовые транзакции называются immediate reads и immediate writes

1. select; commit
2. insert/upsert; commit

- Immediate-транзакция направляется на выполнение непосредственно в шард без планирования на координаторе
- Immediate-транзакция существенно выигрывает по latency по сравнению с планируемыми транзакциями (например, единицы микросекунд против десятков микросекунд)

HL 2022 HighLoad++ FOUNDATION
Трек Яндекс

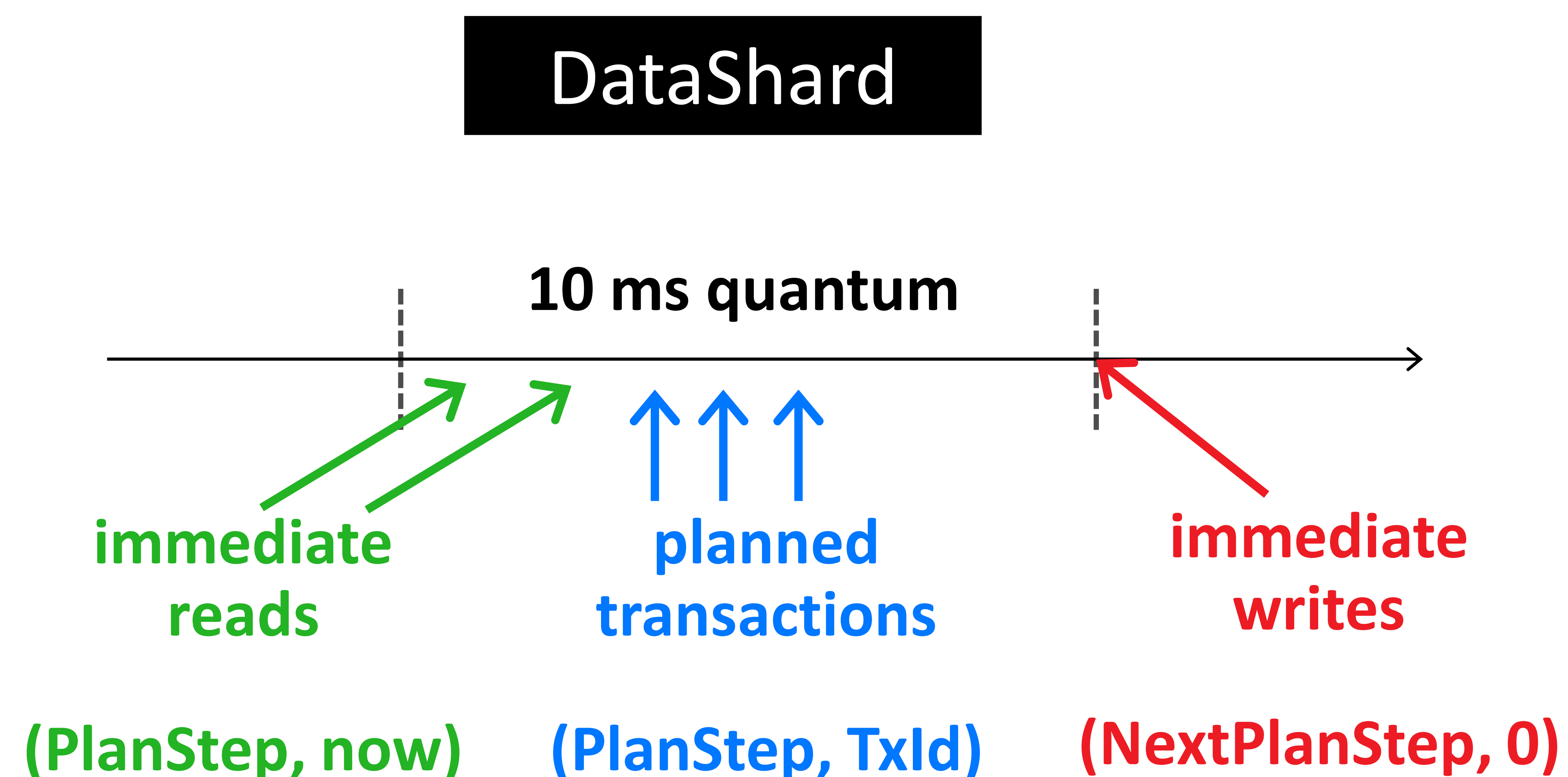
Immediate Transactions and Snapshot Reads



- Immediate reads выполняются как можно раньше
- Планируемые (распределенные) транзакции выполняются в соответствии в полученным от координатора (PlanStep, TxId)

HL 2022 HighLoad++ FOUNDATION
Трек Яндекс

Immediate Transactions and Snapshot Reads



+ Immediate writes

1. У нас нет (PlanStep, TxId) от координатора для них
2. Если выполнить immediate write сейчас, это может сломать snapshot reads
3. В итоге YDB отдает приоритет snapshot read и выполняет immediate write в момент (NextPlanStep, 0)
4. Только если есть snapshot reads в очереди

+ В итоге

1. Для Key-Value-нагрузки snapshot read не влияют на latency
2. Смешанная нагрузка дает latency пенальти для immediate writes

HL 2022 HighLoad++ FOUNDATION
Трек Яндекс

Что-то еще?

Пришлось добавить в YDB несколько глубоких оптимизаций

+ Лизы (leases) координатора

1. Координатор отвечает на GetSnapshot без записи на диск

+ Unprotected Reads

1. Гарантии DataShard, что снапшот не поменяется после рестарта этого DataShard

И в итоге оно заработало хорошо!

HL 2022 HighLoad++ FOUNDATION
Трек Яндекс

08

Исследование производительности

Производительность Read Queries

Graph

Share... | Configure... | Pin | Open in Monitoring

2022-05-11 –2022-05-11 | Now | < < | > > | + | − | Hour | Day | Week | Month | Year

12K
10K
8K
6K
4K
2K
0K

20:56 20:58 21:00 21:02 21:04 21:06 21:08 21:10 21:12 21:14 21:16 21:18 21:20 21:22 21:24 21:26 21:28 21:30 21:32 21:34 21:36 21:38 21:40 21:42 21:44 21:46 21:48 21:

Single Key Read
(Before)

Two Tables Join
(Before)

Single Key Read
(Snapshot reads)

Two Tables Join (Snapshot Read)

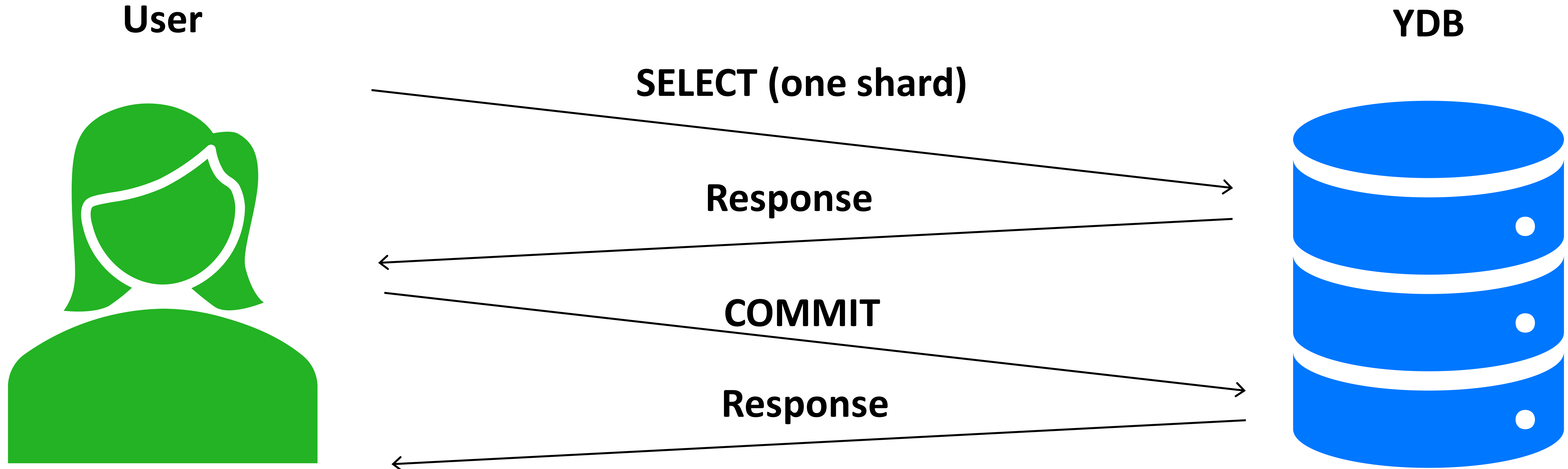
		bin	Min	Max	Last	Avg	Sum
	✓	all (35)					
■	✓	1	0.000K	5.966K	0.000K	1.151K	229.125K

1 18.2%
6 7.2%
5 6.7%
2 22.4%

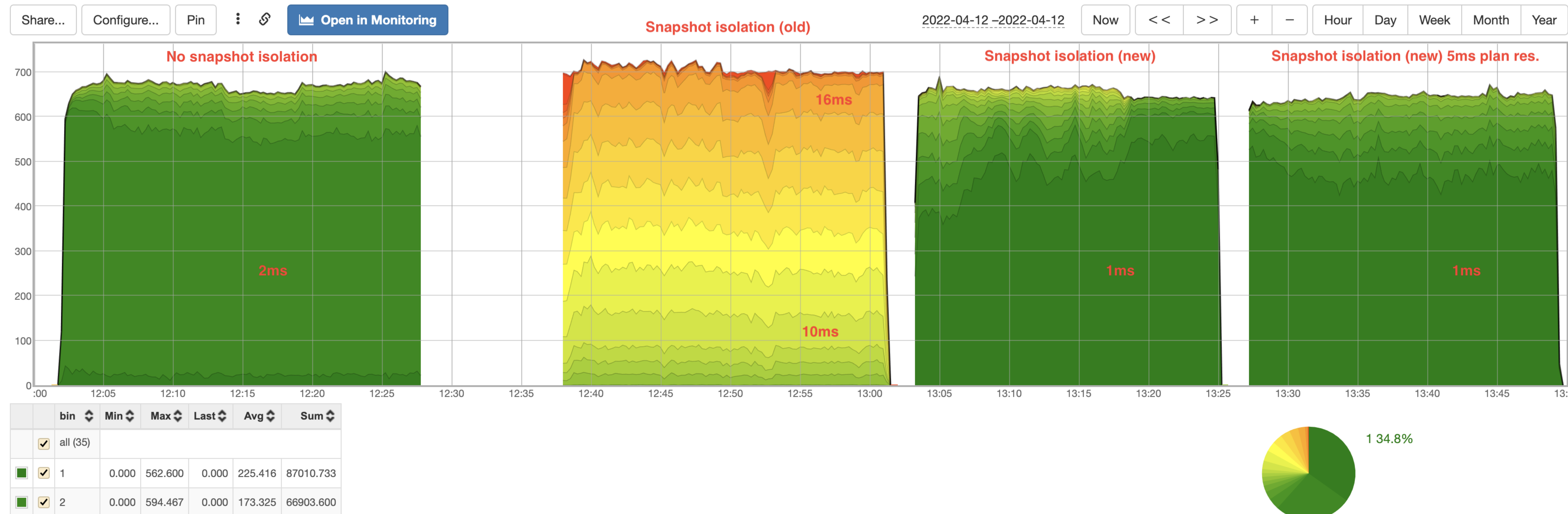
HL 2022 HighLoad++ FOUNDATION

Трек Яндекс

“Open Read” Query – худший сценарий



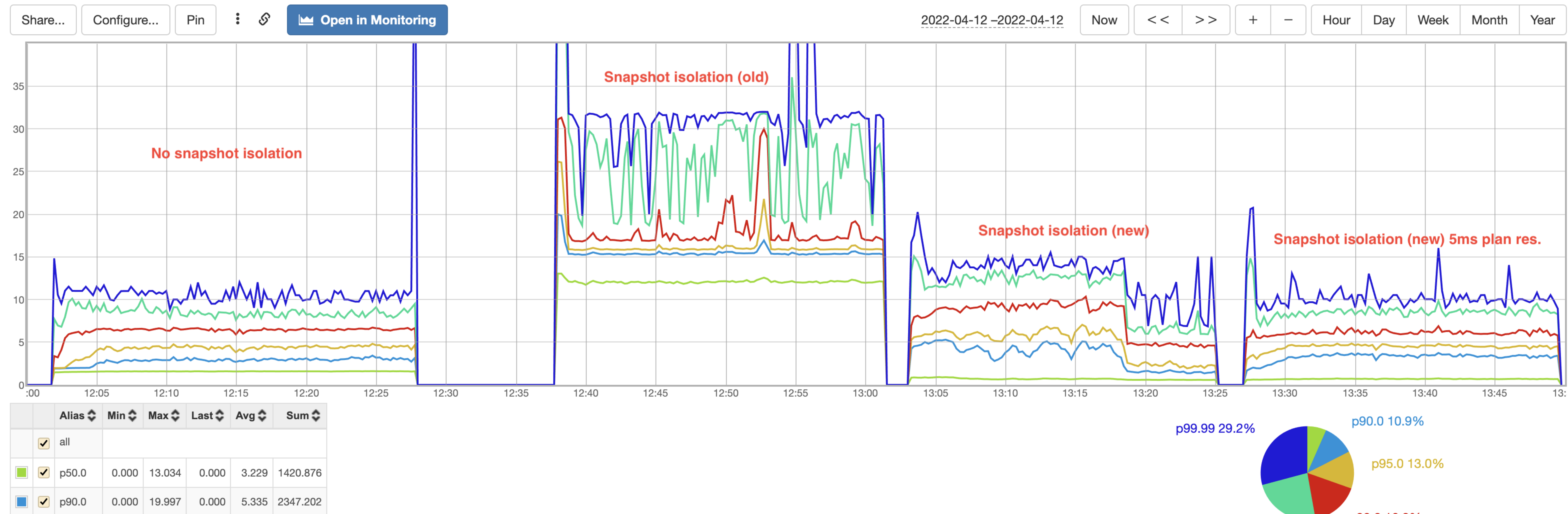
“Open Read” Query Latency



		bin	Min	Max	Last	Avg	Sum
	✓	all (35)					
■	✓	1	0.000	562.600	0.000	225.416	87010.733
■	✓	2	0.000	594.467	0.000	173.325	66903.600

HL 2022 HighLoad++ FOUNDATION
Трек Яндекс

“Open Read” Query Latency Percentile



		Alias	Min	Max	Last	Avg	Sum
	✓	all					
	✓	p50.0	0.000	13.034	0.000	3.229	1420.876
	✓	p90.0	0.000	19.997	0.000	5.335	2347.202

HL 2022 HighLoad++ FOUNDATION
Трек Яндекс

09

Выводы

Выводы

Мы добавили поддержку MVCC в YDB

Мы добавили поддержку Snapshot Read

+ Пользователь читает консистентные данные без вызова commit и проверки сломанных локов

Уровень изоляции в YDB по-прежнему serializable

+ Подумаем про Snapshot Isolation

Мы протестировали производительность

Мы разблокировали новые сценарии

+ Консистентные вторичные индексы без распределенных транзакций

Андрей Фомичев

Руководитель YDB