One more way to make backup in Apache Ignite

Николай Ижиков

Николай Ижиков

Platform V DataGrid tech lead

- Apache Ignite PMC
- Apache Kafka contributor

t.me/db_links
t.me/nizhikovTalks
github.com/nizhikov
nizhikov@apache.org

Команда

- Разрабатываем ядро распределенной СУБД с 2017 года
- Являемся основными контрибьюторами в Apache Ignite 2.x
- Вносим большинство правок в код Apache Ignite
- Обладаем полной экспертизой в разработке СУБД
- Умеем мониторить и поддерживать Ignite в ПРОМ. Customer Success Team(L3) – топчик

План доклада

Вступление

- Приложить линейку к... Apache Ignite
- Теория

Существующие способы

- Ребаланс
- CDC
- Полный снапшот
- Инкрементальный снапшот

Дамп кэшей

План доклада

Вступление

Приложить линейку к... Apache Ignite 1

Теория 2

Существующие способы

Ребаланс 3

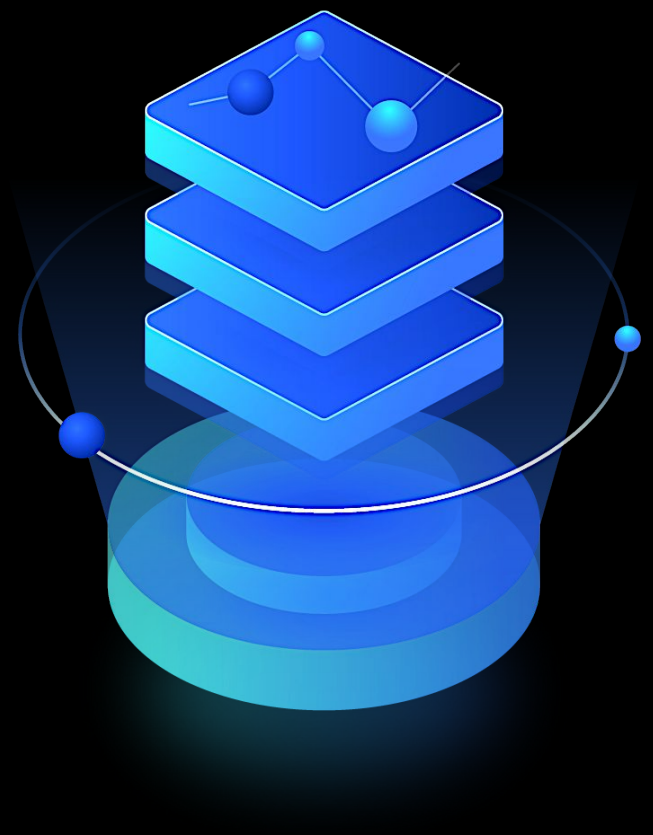
CDC 4

Полный снапшот 5

Инкрементальный снапшот 6

Дамп кэшей

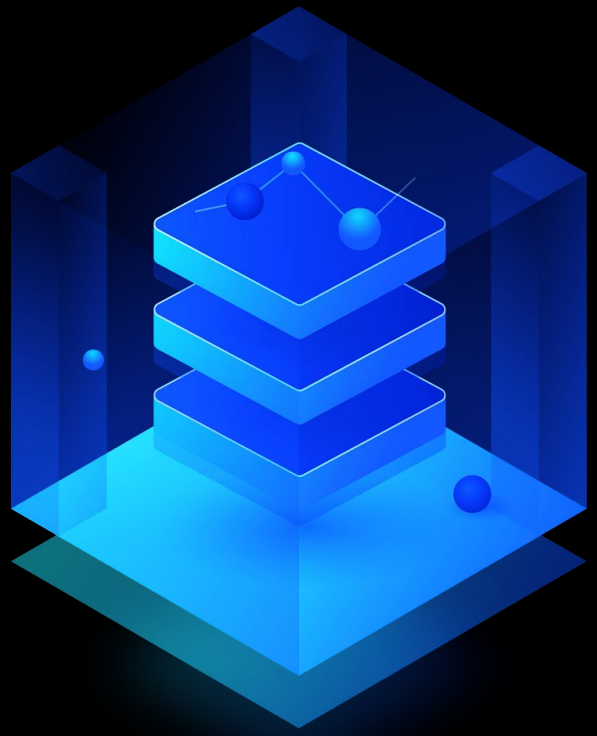
Apache Ignite



Распределенная СУБД

Умеет быстро, транзакционно
и надежно обрабатывать OLTP нагрузку

DataGrid — распределённая OLTP СУБД



DataGrid — форк Apache Ignite

Умеет то же, что и Ignite:

- In-memory режим
- ACID Транзакции и KV api.
- Compute platform.
- SQL и индексы.
- Полные и инкрементальные бэкапы.
- Асинхронная репликация (CDC).

Вообще-то DataGrid может даже больше...

DataGrid в цифрах

Используют крупнейшие Mission Critical+ системы

- ЕПК (единый профиль клиента)
- Эквайринг
- Процессинг
- Карты
- Токенизатор
- Антифрод
- 1000+ кластеров
- 300+ АС

DataGrid в цифрах

Высоконагруженное хранилище данных

Процессинг

32 узла
в кластере

700 ГБ
RAM на узел

40 000
RPS на узел

DataGrid в цифрах

Высоконагруженное хранилище данных

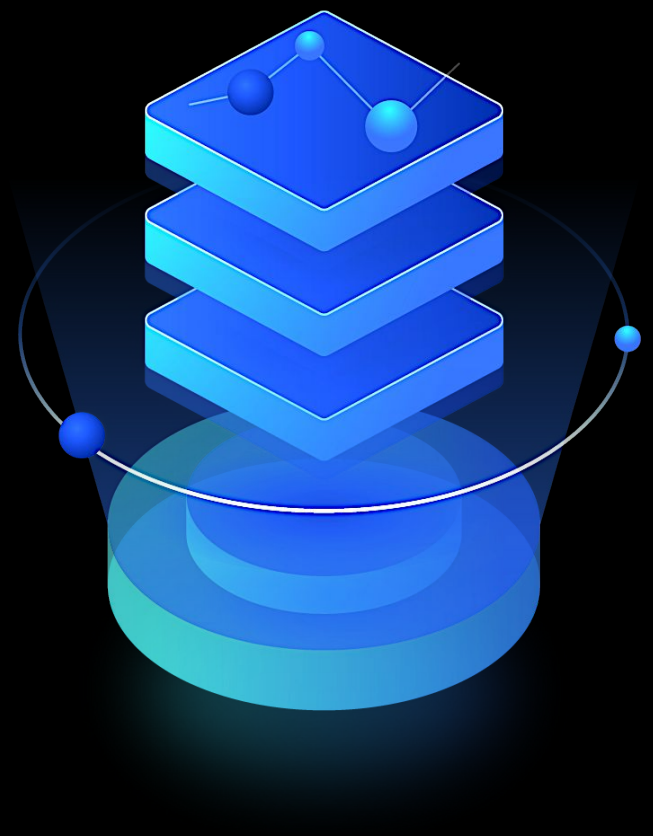
Токенизатор

5 узлов
в кластере

550 ГБ
RAM на узел

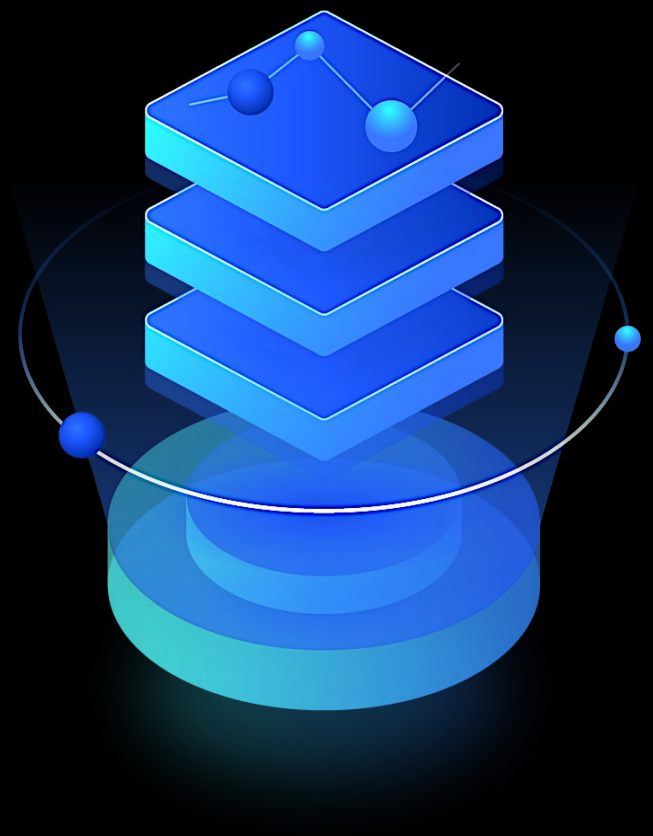
500 000
RPS

База #1



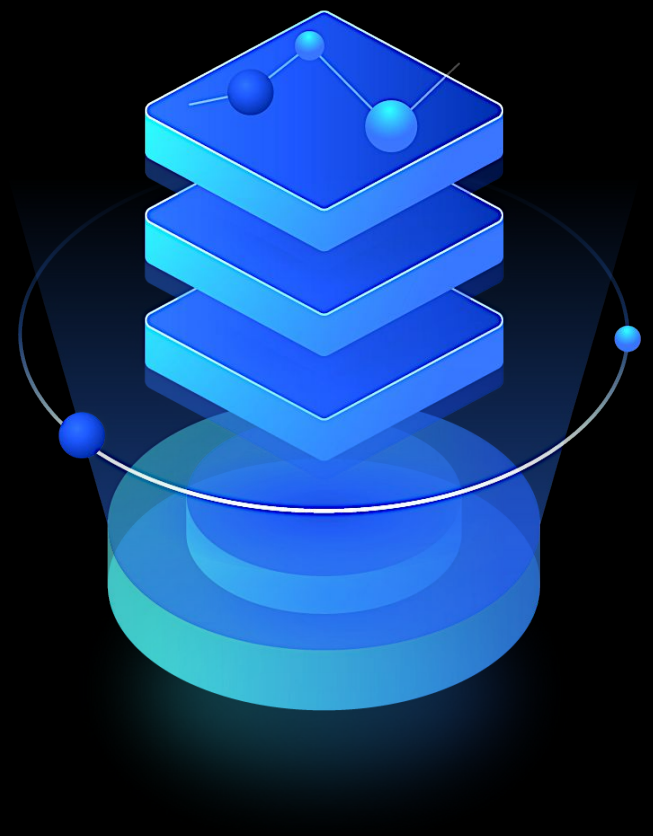
Задача СУБД — хранить данные и отвечать на запросы

База #2



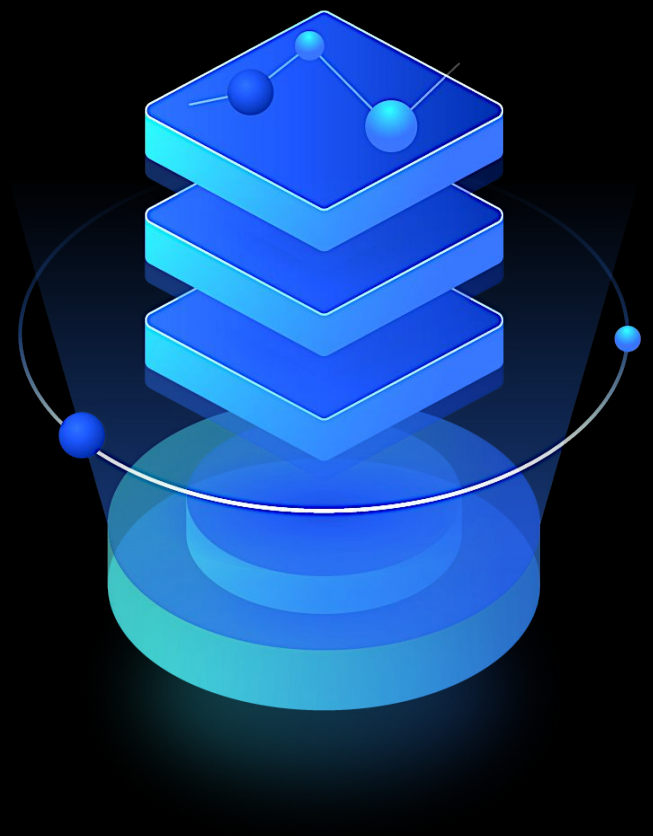
Что бы мы ни делали,
баги случаются, сервера
падают, стойки сгорают,
дата центры бывают
недоступны

База #3



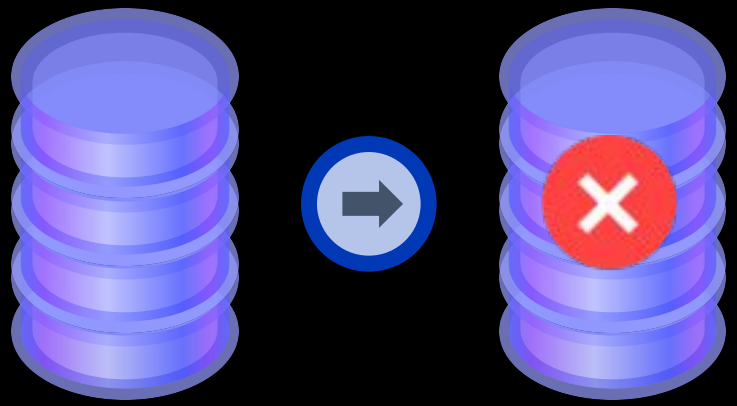
Делайте бэкапы если еще не.

Что?



В докладе проведу обзор способов восстановить данные после инцидентов, а также расскажу о новой фиче – **cache dumps**

Как падает однонодовая СУБД



Причины

- Отказ оборудования
- Проблемы с электричеством
- Баг в СУБД
- Баг в приложении

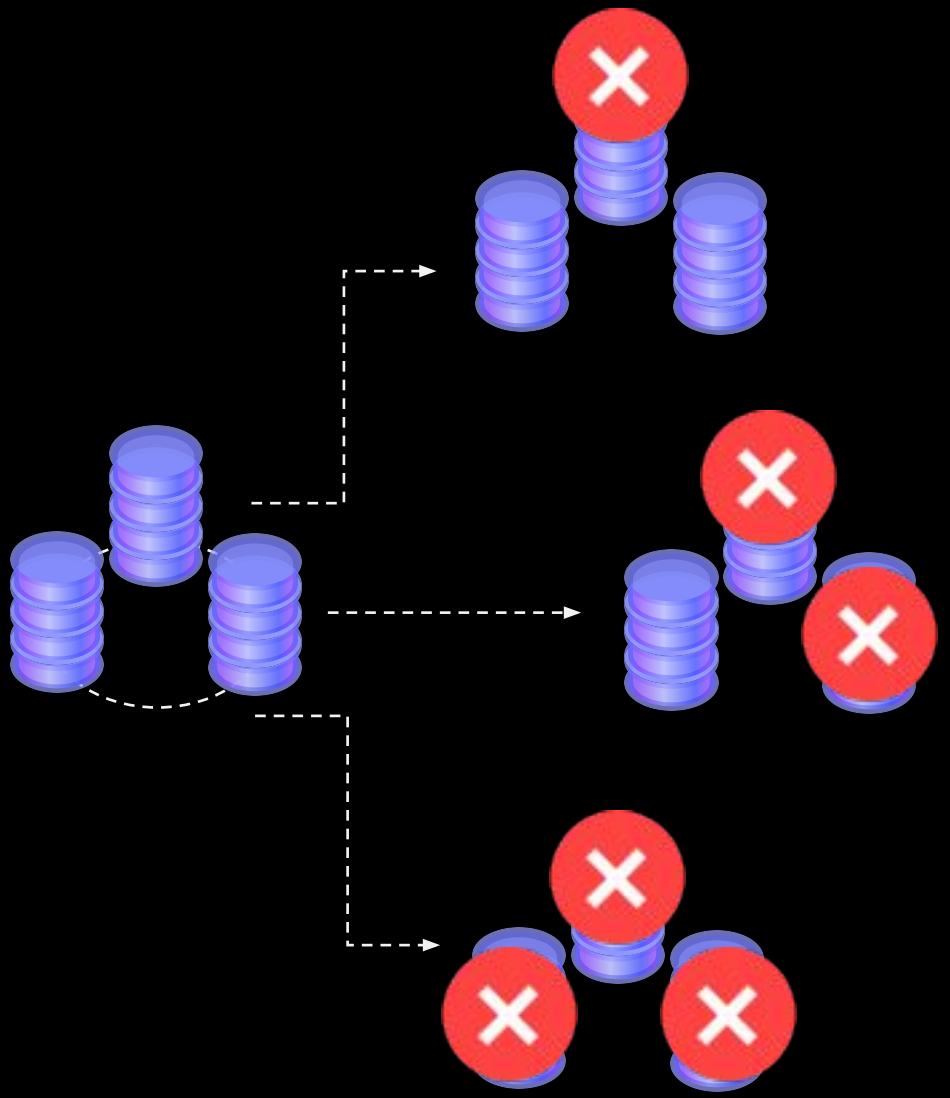
Последствия

- Недоступность — приложение не работает
- Потеря данных

Защита и восстановление

- Шардирование на уровне приложения
- Снапшоты
- Рестарт

Как падает распределенная СУБД



Причины

- Отказ оборудования
- Проблемы с электричеством
- Баг в СУБД
- Баг в приложении

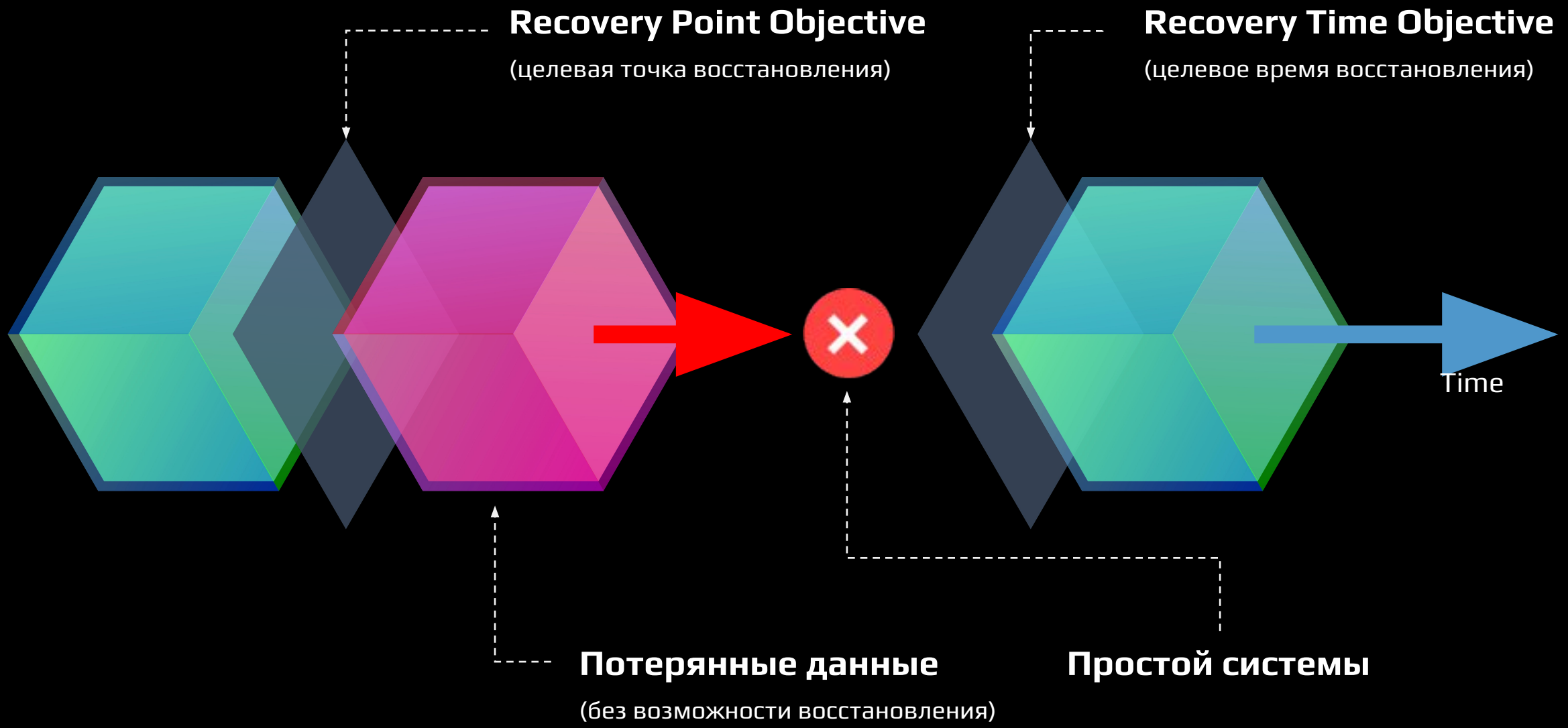
Последствия

- Снижение общего перформанса
- Необходимость системных процессов восстановления

Защита и восстановление

- Правильная настройка
- Добавление узлов в кластер после падения (стандартная операция!)
- Снапшоты

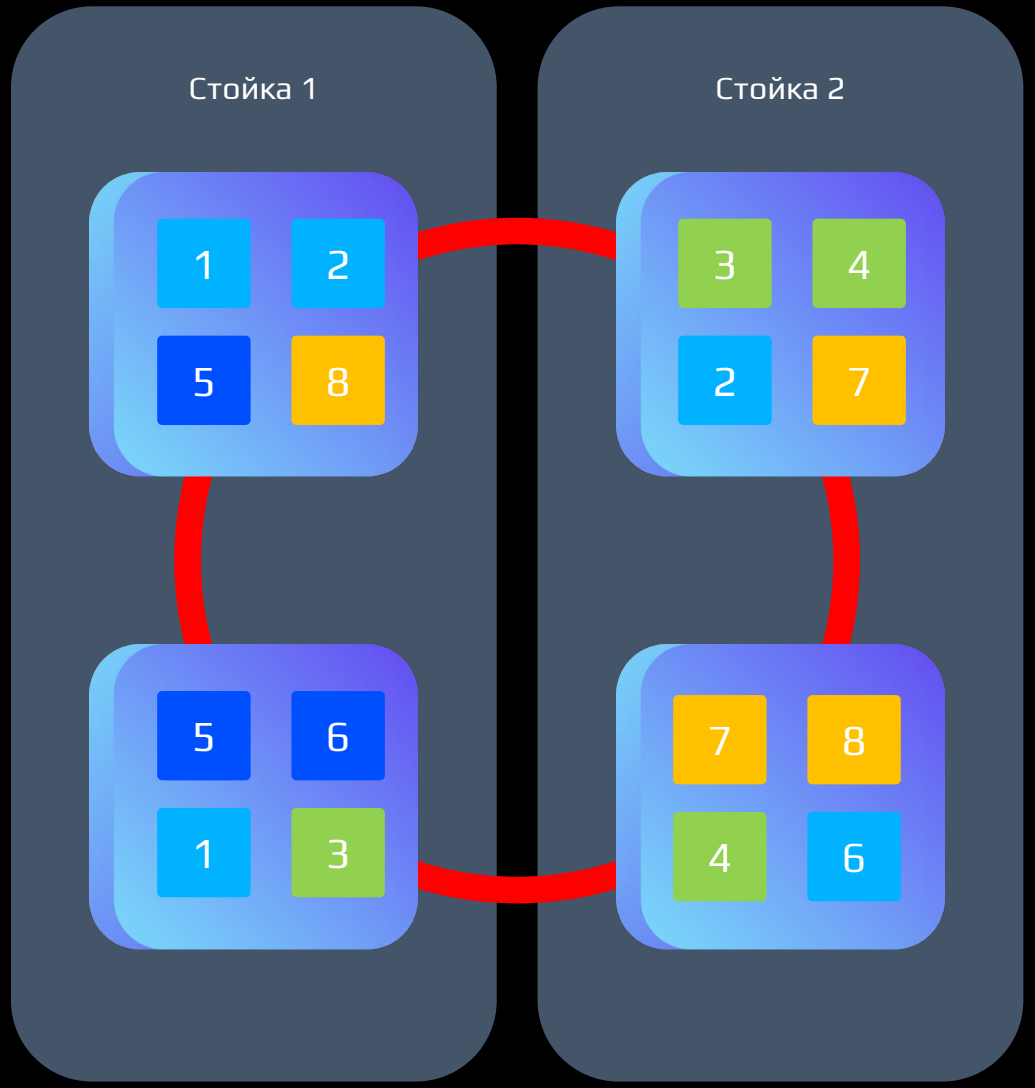
Аварийное восстановление



Ребаланс

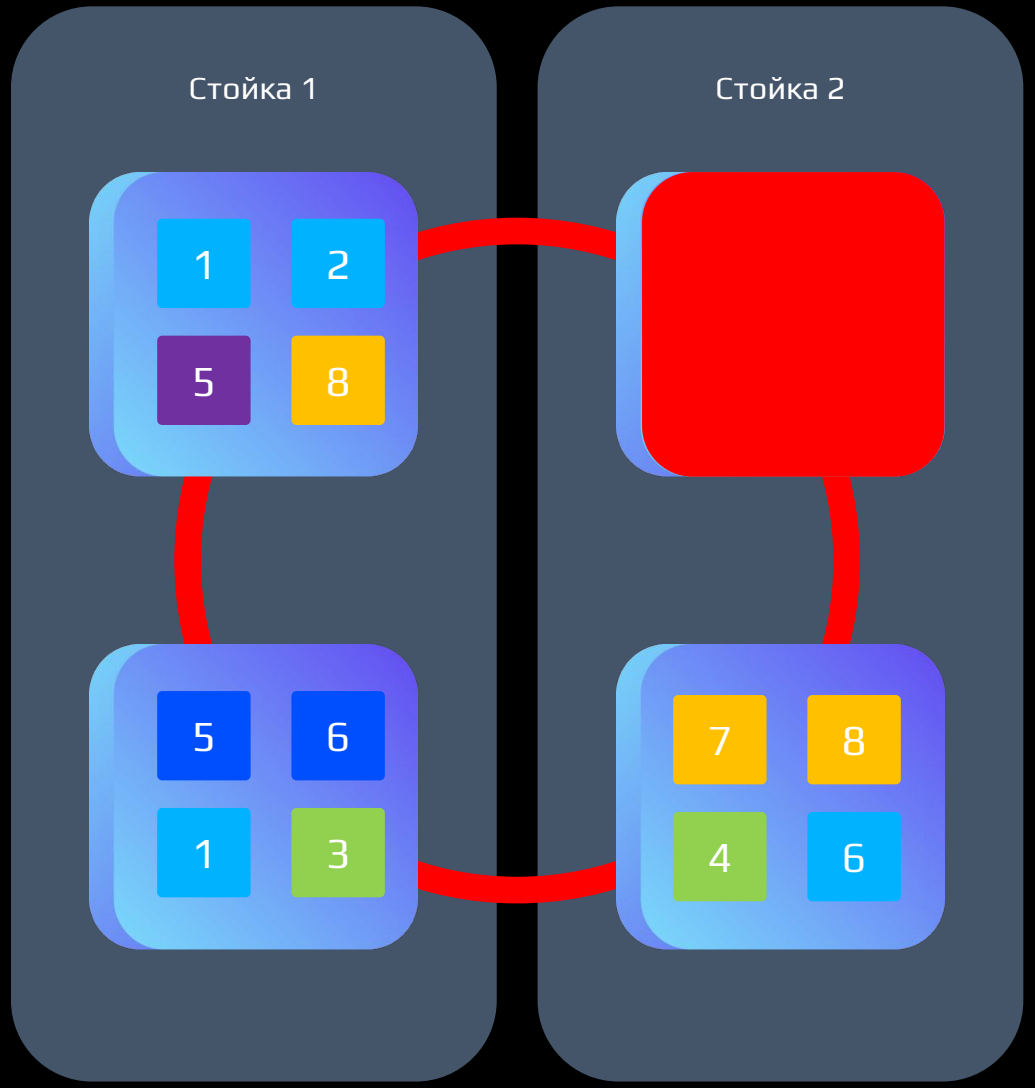
Apache Ignite

Apache Ignite: ребаланс




```
CacheConfiguration {
    affinity → {
        partitions → 8
    },
    backups → 1
}
```

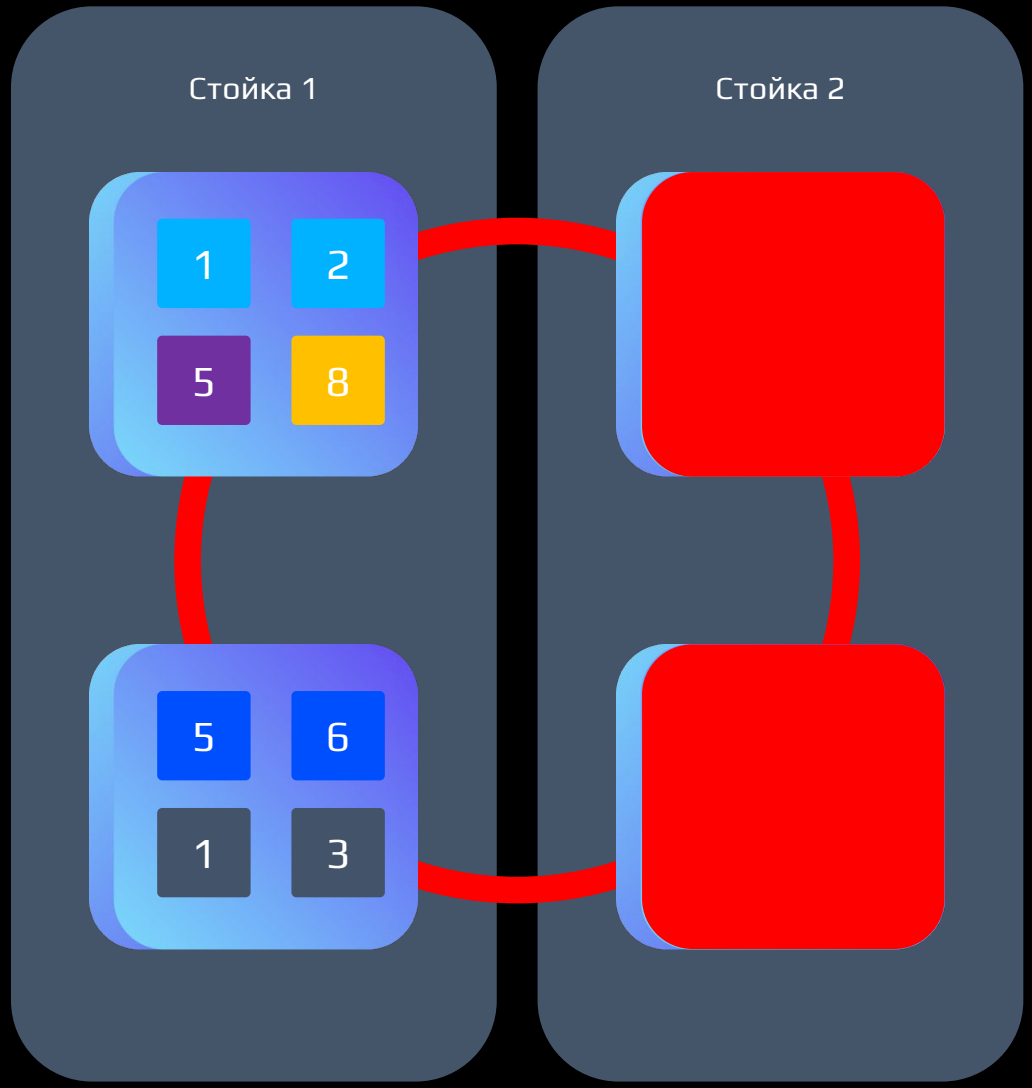
Apache Ignite open source: ребаланс



Остановка узла:

1. Перезапустить узел
2. Ignite сам выполнит ребаланс данных

Apache Ignite open source: ребаланс



Потеря стойки

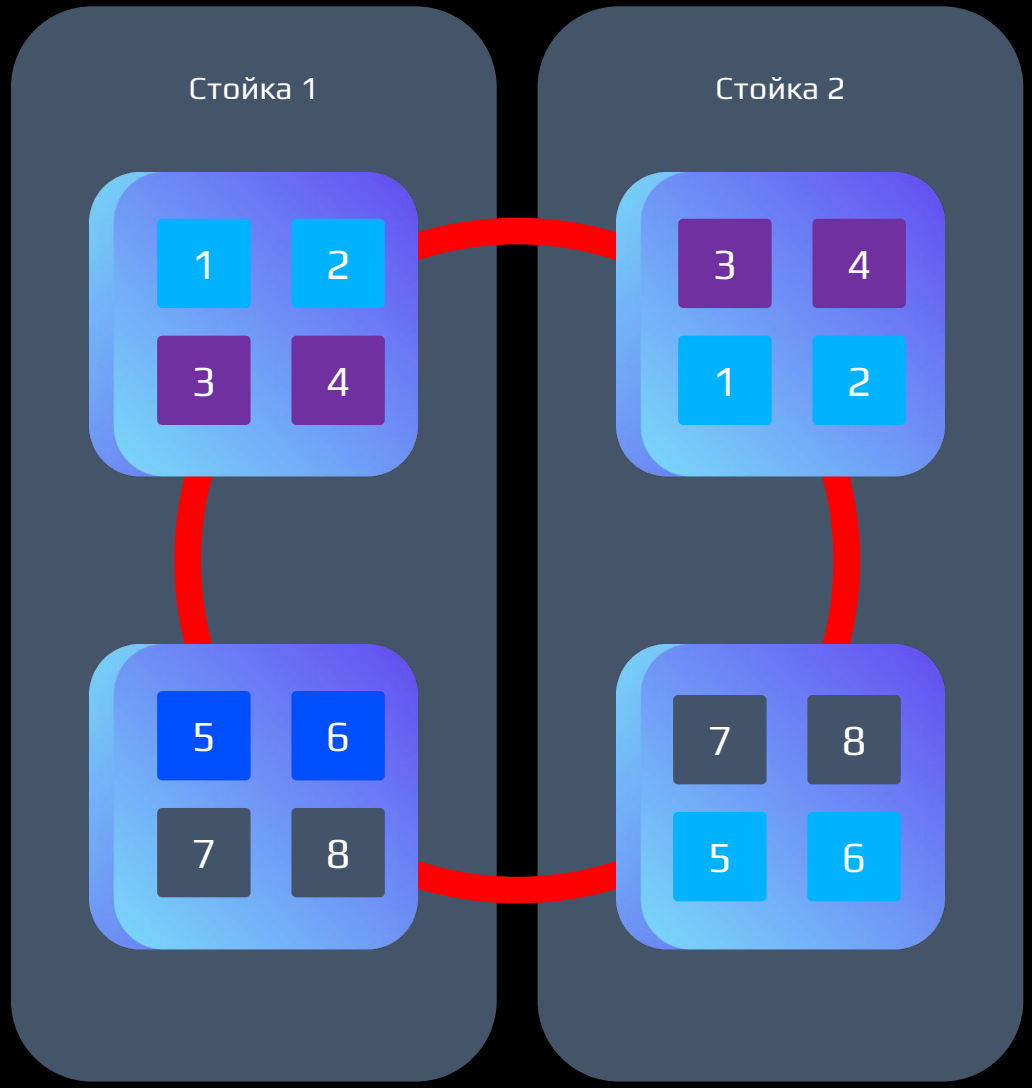
Партиции 4, 7 потеряны

	RPO	RTO	Фича Data Grid	Полное падение	Support inmemory caches	Зачем надо?
Ребаланс	0	0			Да	Восстановиться без простоя при падении узла

Ячейки

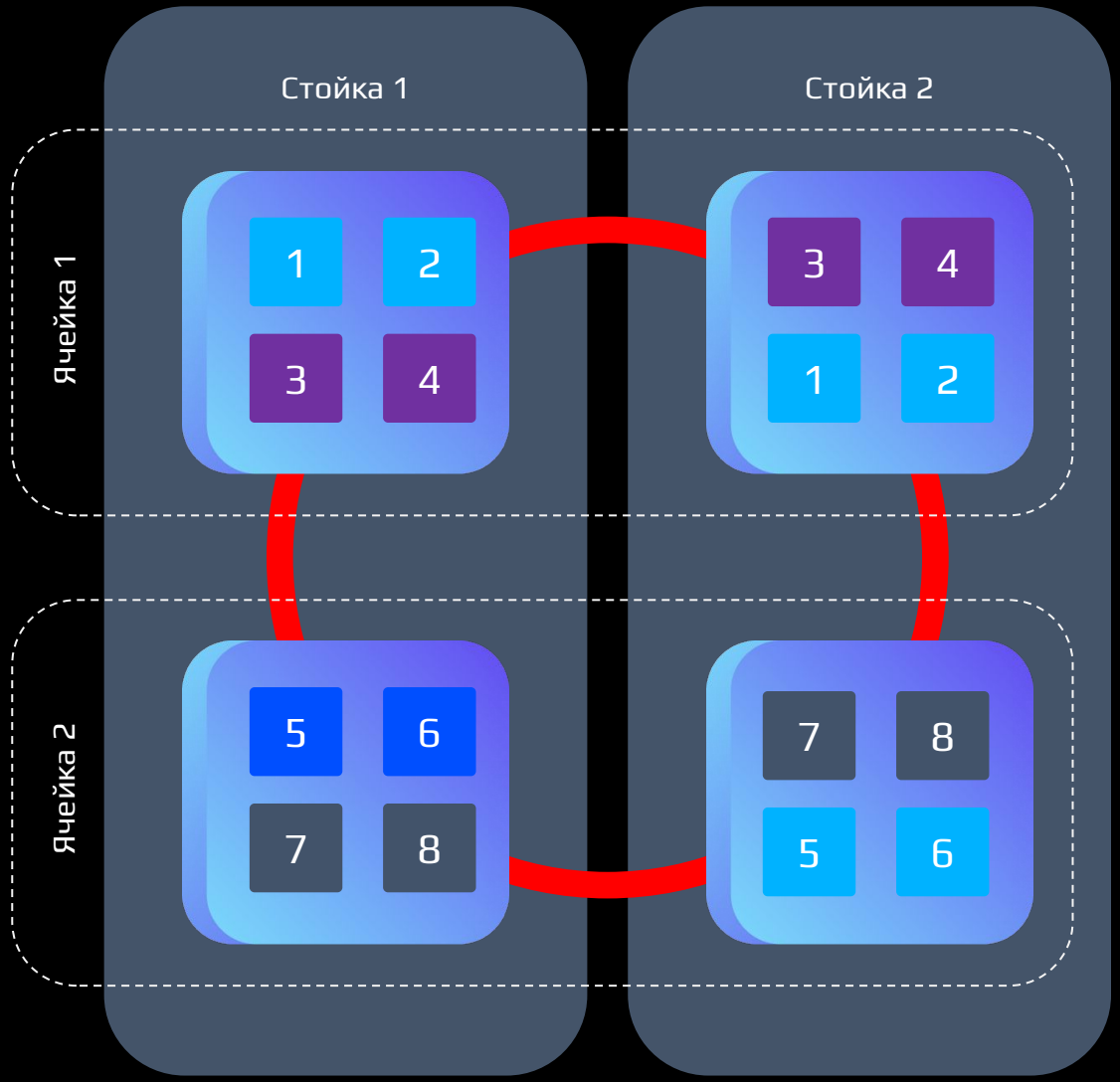
DataGrid

DataGrid: ячейки




```
CacheConfiguration {
    affinity → {
        partitions → 8
        affinityBackupFilter → CellularAffinity
    },
    backups → 1
}
```

DataGrid: ячейки



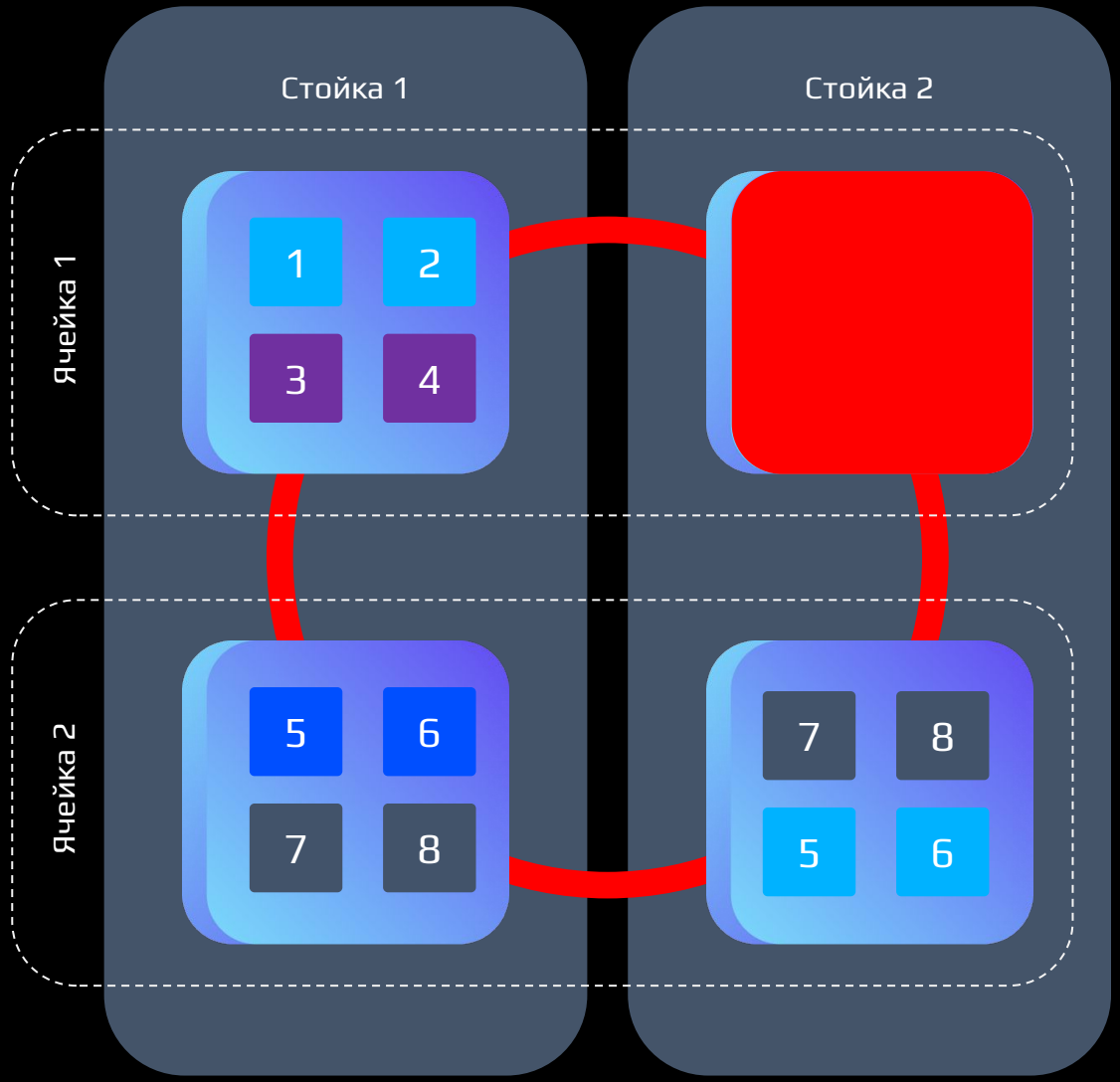

```
BaselineNode → {
  consistentId → “Node01”,
  attributes → {
    “cell” → “ячейка 1”
  }
}
```

```
BaselineNode → {
  consistentId → “Node02”,
  attributes → {
    “cell” → “ячейка 1”
  }
}
```

```
BaselineNode → {
  consistentId → “Node03”,
  attributes → {
    “cell” → “ячейка 2”
  }
}
```

```
BaselineNode → {
  consistentId → “Node04”,
  attributes → {
    “cell” → “ячейка 2”
  }
}
```

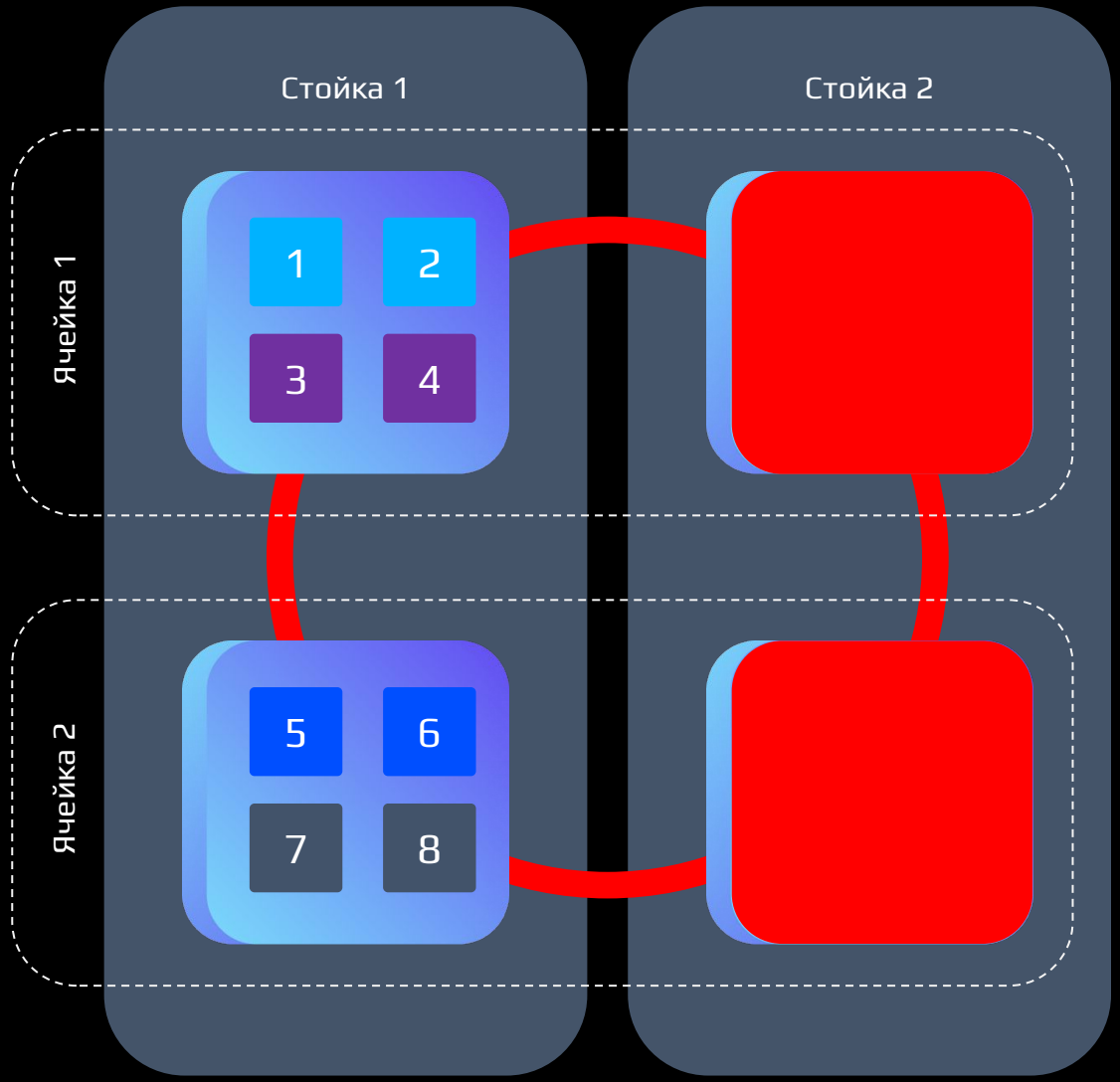
DataGrid: ячейки



Остановка узла:

1. Перезапустить узел.
2. Ignite сам отребалансирует данные.

DataGrid: ячейки



Потеря стойки

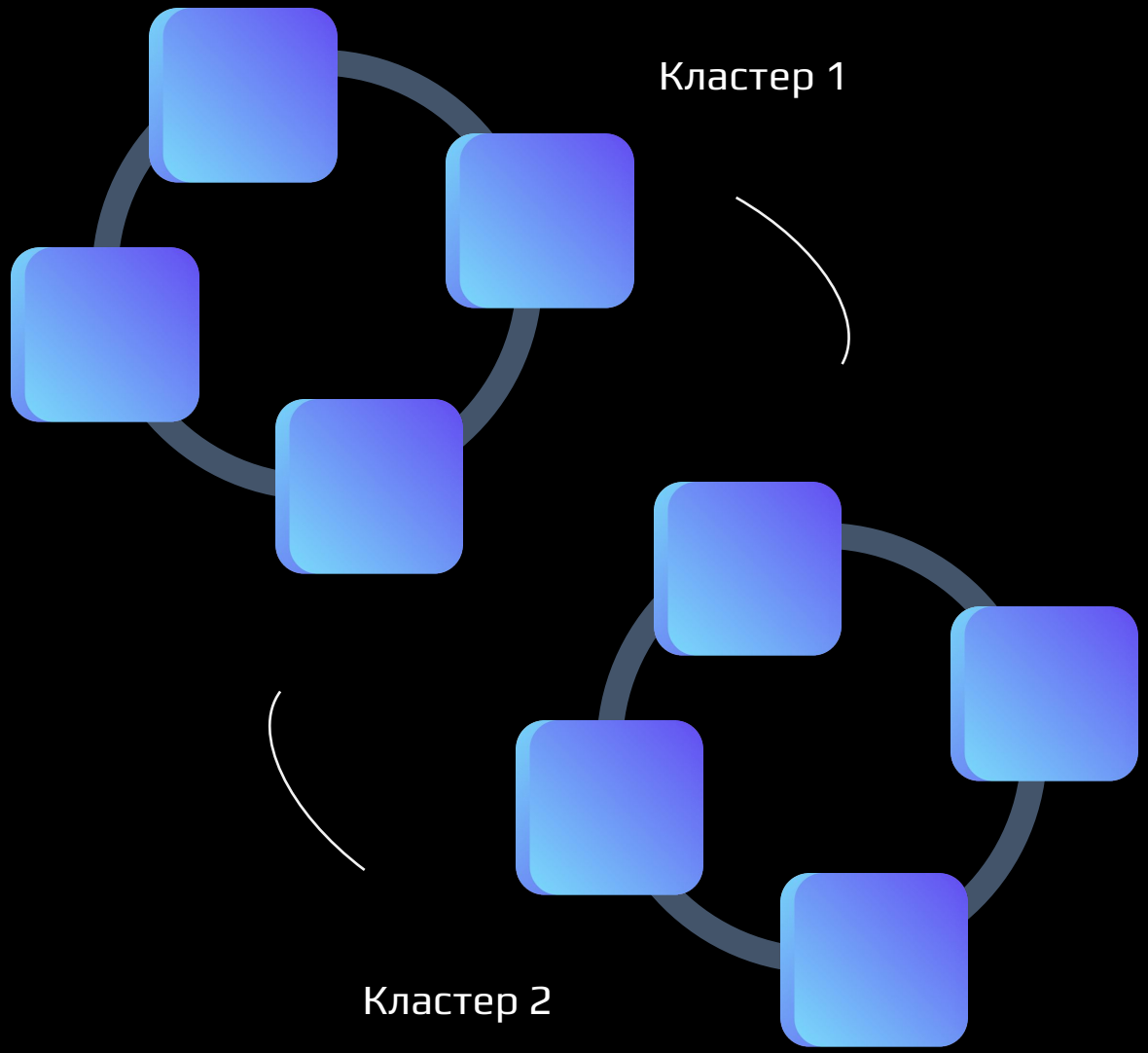
Все партиции в кластере!

1. Поменять стойку
2. Перезапустить узлы
3. Ignite сам отребалансирует данные (полный ребаланс)

	RPO	RTO	Фича Data Grid	Полное падение	Support inmemory caches	Зачем надо?
Ребаланс	0	0			Да	Восстановиться без простоя при падении узла
Ячейки	0	0	Да		Да	Восстановиться без простоя без стойки


CDC
Apache Ignite

Apache Ignite: CDC

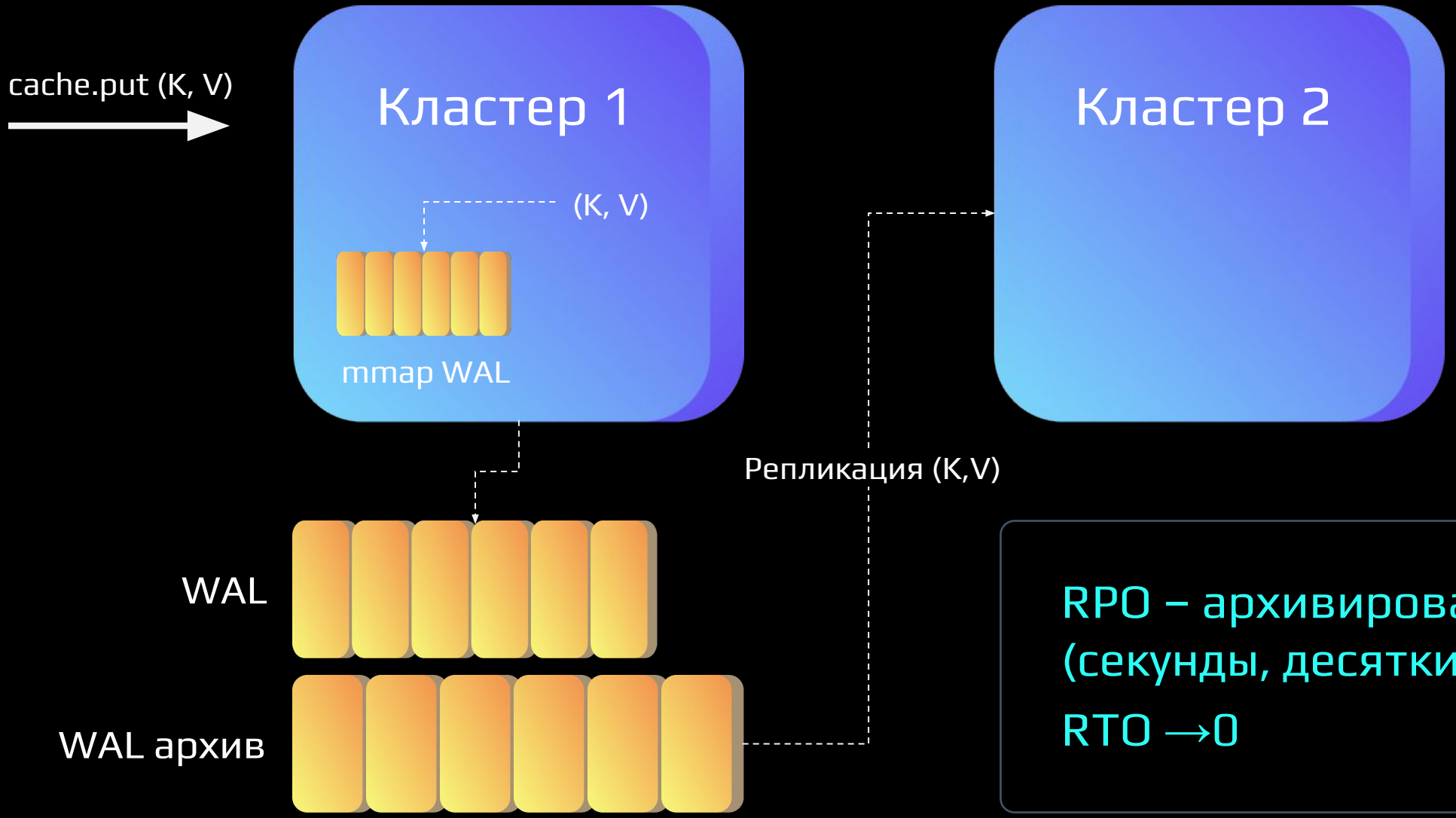


1. CDC обеспечивает асинхронную логическую репликацию данных
2. CDC поставляет фреймворк для написания своей логики репликации
3. В Apache Ignite реализована репликация через тонкого клиента, толстого клиента, Kafka
4. https://t.me/nizhikovTalks/28

RPO → секунды

RTO → 0

Apache Ignite: CDC



RPO – архивирование WAL (секунды, десятки секунд)

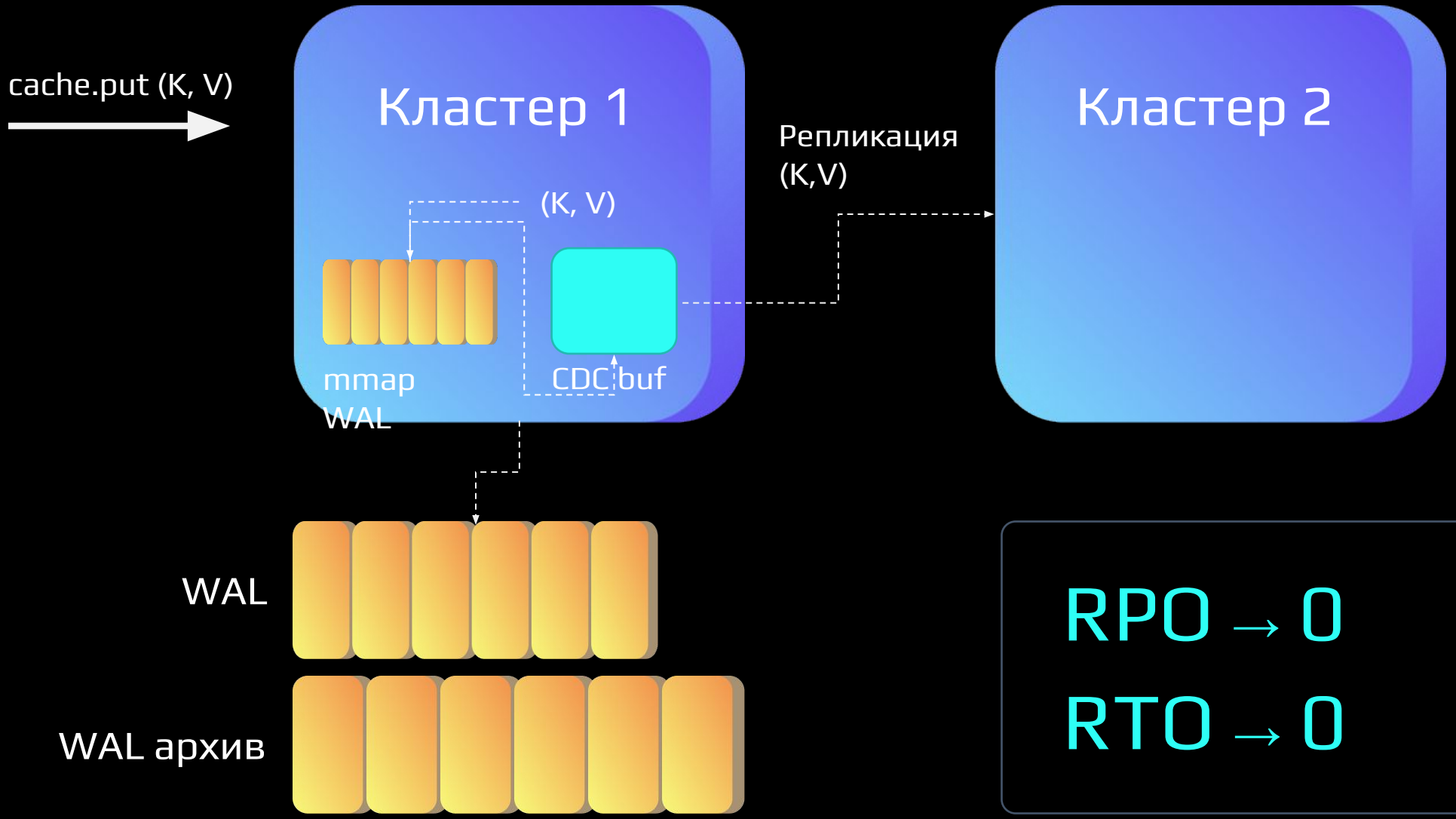
RTO →0

	RPO	RTO	Фича Data Grid	Полное падение	Support inmemory caches	Зачем надо?
Ребаланс	0	0			Да	Восстановиться без простоя при падении узла
Ячейки	0	0	Да		Да	Восстановиться без простоя без стойки
CDC	0	Время обработки сегмента		Частично	Да	Multi DC replication

Online CDC

DataGrid

DataGrid: online CDC



RPO → 0

RTO → 0

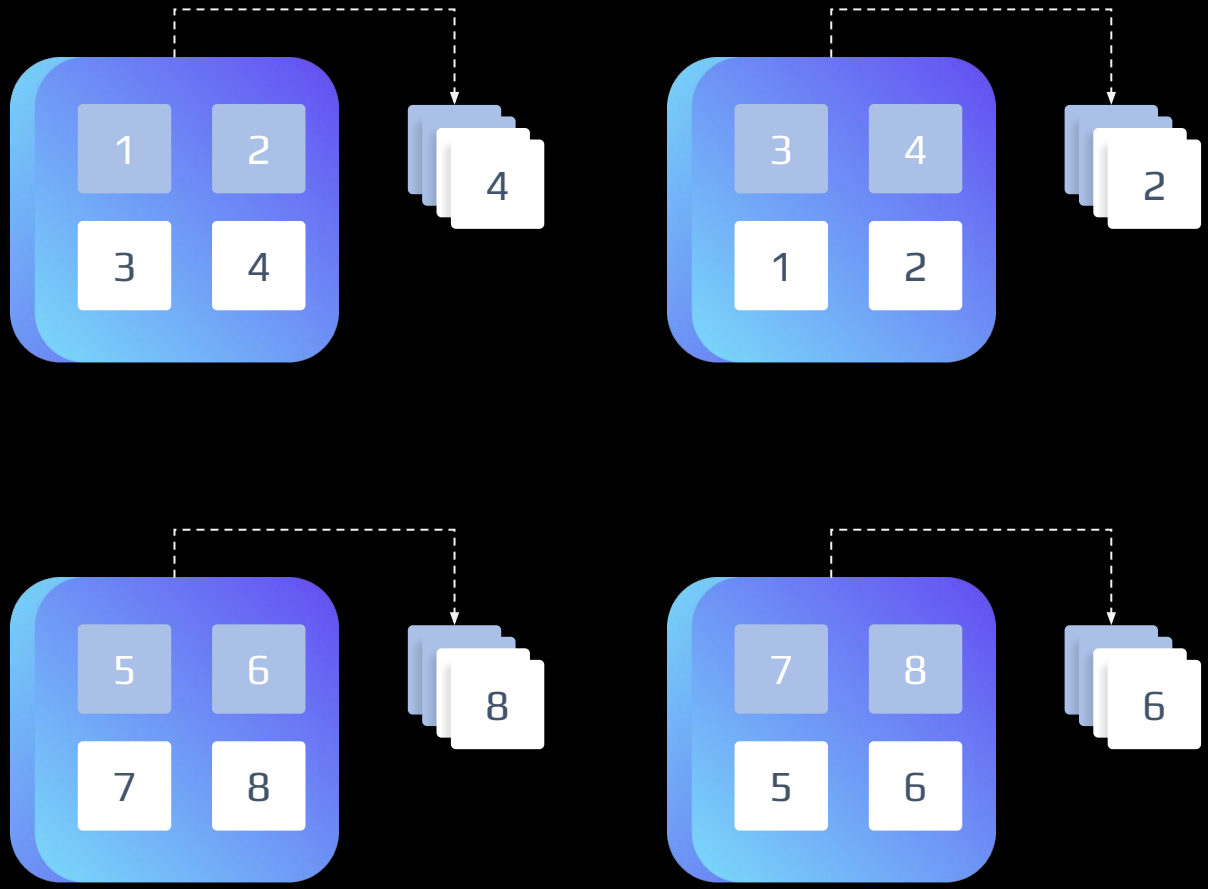
	RPO	RTO	Фича Data Grid	Полное падение	Support inmemory caches	Зачем надо?
Ребаланс	0	0			Да	Восстановиться без простоя при падении узла
Ячейки	0	0	Да		Да	Восстановиться без простоя без стойки
CDC	0	Время обработки сегмента		Частично	Да	Multi DC репликация
Online CDC	0	0	Да	Частично	Да	Multi DC репликация с меньшей задержкой

Снапшот

Apache Ignite

Apache Ignite: алгоритм снятия снапшота

1. Администратор дает команду.
2. Системный процесс делает stop the world (PME).
3. Во время StW нет транзакций — данные консистентны, можно инициализировать писателей снапшота.
4. После checkpoint начинают работу Copy On Write storage. Перед изменением страницы сохраняем старую версию в дельта файл.
5. Копируем все партиции на всех нодах, сверху накладываем дельта-файл.



Apache Ignite: алгоритм снятия снапшота

Полная резервная копия всех партиций

Преимущества

- RTO ≈ 0
- Восстановление на другую топологию.

Недостатки

- Большой объем.
- Дорого снимать — StW, CoW, RAM, IO.
- RPO — часы.
- Только для persistent кэшей.

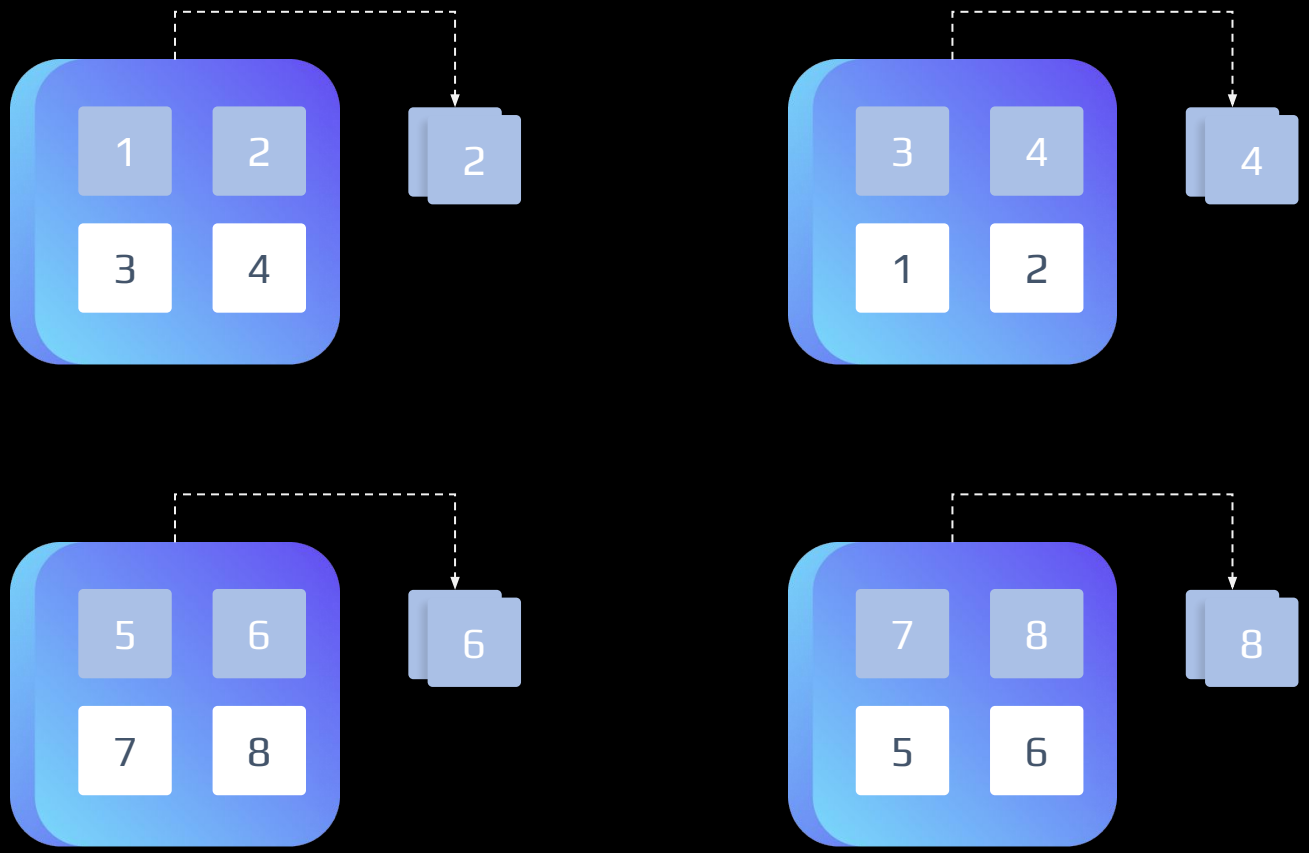
	RPO	RTO	Фича Data Grid	Полное падение	Support inmemory caches	Зачем надо?
Ребаланс	0	0			Да	Восстановиться без простоя при падении узла
Ячейки	0	0	Да		Да	Восстановиться без простоя без стойки
CDC	0	Время обработки сегмента		Частично	Да	Multi DC репликация
Online CDC	0	0	Да	Частично	Да	Multi DC репликация с меньшей задержкой
Полный снапшот	Часы	0		Да		Восстановить PDS при падении кластера

Only-primary снапшот

Apache Ignite

DataGrid: only-primary снапшот

Резервная копия primary-партиций



Преимущества

- Кратно меньший размер по сравнению с полным

Недостатки

- Все еще дорого
- RTO время ребаланса
- RPO – часы
- Нужен ребаланс на восстановлении
- Только для persistent кэшей

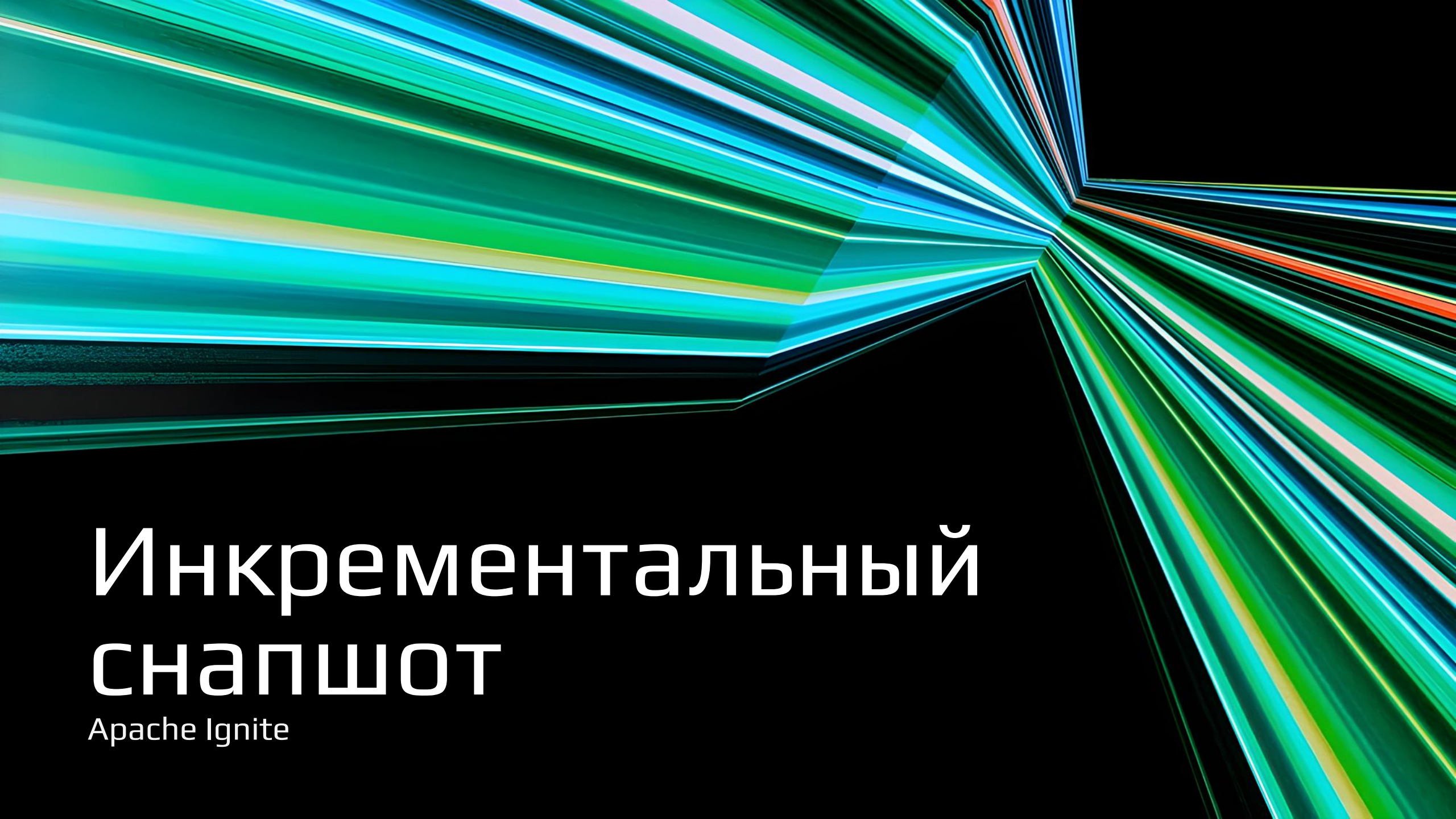
Снапшот — итоги

RPO = часы

RTO 0 + ребланс
(в случае only-primary)

Переживает остановку кластера

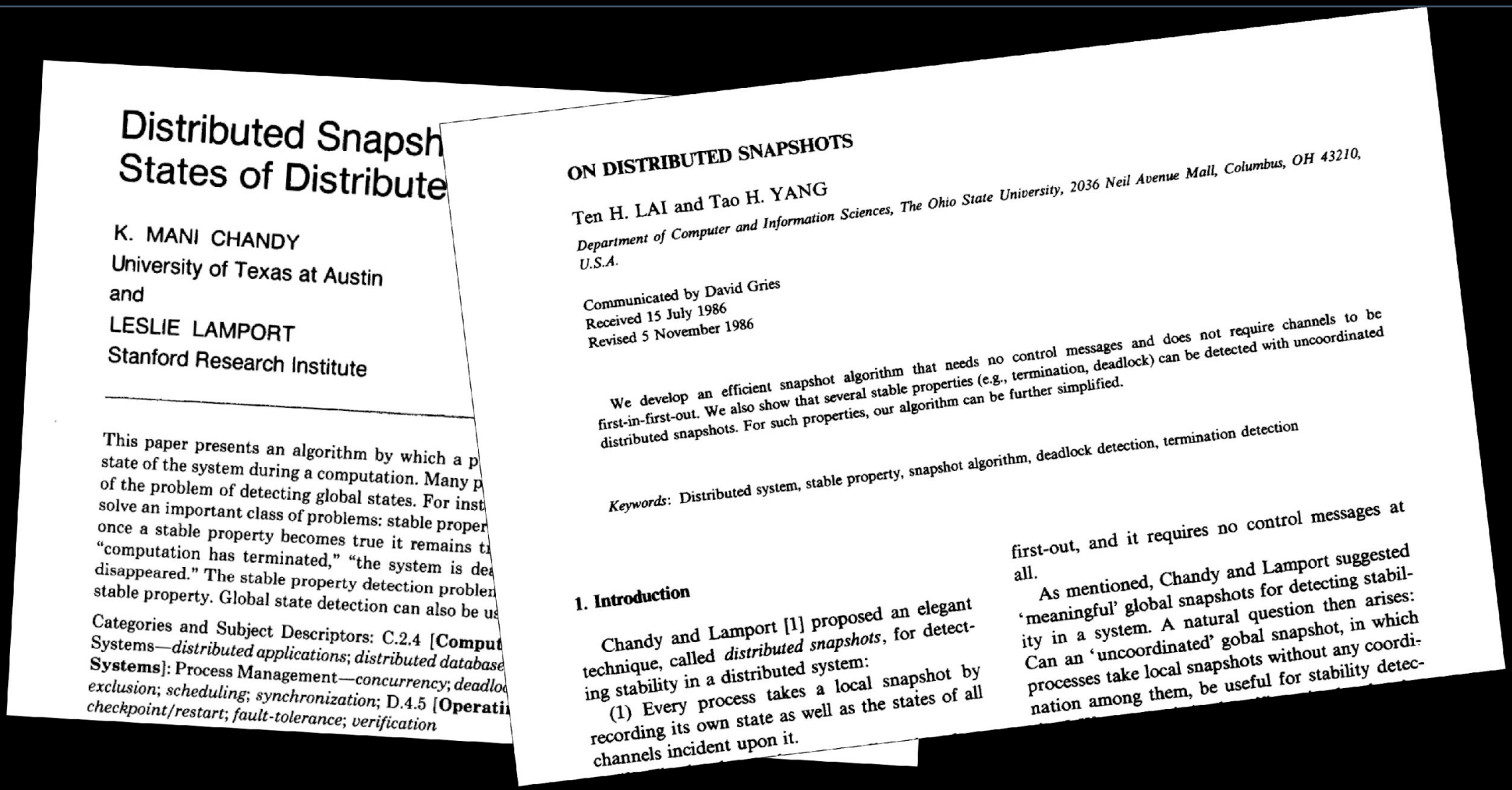
	RPO	RTO	Фича Data Grid	Полное падение	Support inmemory caches	Зачем надо?
Ребаланс	0	0			Да	Восстановиться без простоя при падении узла
Ячейки	0	0	Да		Да	Восстановиться без простоя без стойки
CDC	0	Время обработки сегмента		Частично		Multi DC репликация
Online CDC	0	0	Да	Частично		Multi DC репликация с меньшей задержкой
Полный снапшот	Часы	0		Да		Восстановить PDS при падении кластера
Полный снапшот. only-primary	Часы	Время ребаланса	Да	Да		Хранить меньше



Инкрементальный снапшот

Apache Ignite

Consistent Cut



Distributed Snapsh
States of Distribute

K. MANI CHANDY
University of Texas at Austin
and
LESLIE LAMPORT
Stanford Research Institute

This paper presents an algorithm by which a p
state of the system during a computation. Many p
of the problem of detecting global states. For inst
solve an important class of problems: stable proper
once a stable property becomes true it remains t
"computation has terminated," "the system is dea
disappeared." The stable property detection proble
stable property. Global state detection can also be us

Categories and Subject Descriptors: C.2.4 [Comput
Systems—*distributed applications*; *distributed database*
Systems]: Process Management—*concurrency*; *deadlo*
exclusion; *scheduling*; *synchronization*; D.4.5 [**Operati**
checkpoint/restart; *fault-tolerance*; *verification*

ON DISTRIBUTED SNAPSHOTS

Ten H. LAI and Tao H. YANG

Department of Computer and Information Sciences, The Ohio State University, 2036 Neil Avenue Mall, Columbus, OH 43210, U.S.A.

Communicated by David Gries
Received 15 July 1986
Revised 5 November 1986

We develop an efficient snapshot algorithm that needs no control messages and does not require channels to be first-in-first-out. We also show that several stable properties (e.g., termination, deadlock) can be detected with uncoordinated distributed snapshots. For such properties, our algorithm can be further simplified.

Keywords: Distributed system, stable property, snapshot algorithm, deadlock detection, termination detection

1. Introduction

Chandy and Lamport [1] proposed an elegant technique, called *distributed snapshots*, for detecting stability in a distributed system:

(1) Every process takes a local snapshot by recording its own state as well as the states of all channels incident upon it.

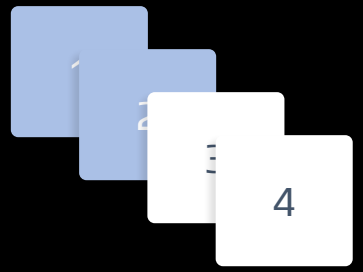
first-out, and it requires no control messages at all.

As mentioned, Chandy and Lamport suggested 'meaningful' global snapshots for detecting stability in a system. A natural question then arises: Can an 'uncoordinated' gobal snapshot, in which processes take local snapshots without any coordination among them, be useful for stability detec-

https://habr.com/ru/companies/sberbank/articles/753404/

Как Apache Ignite хранит данные?

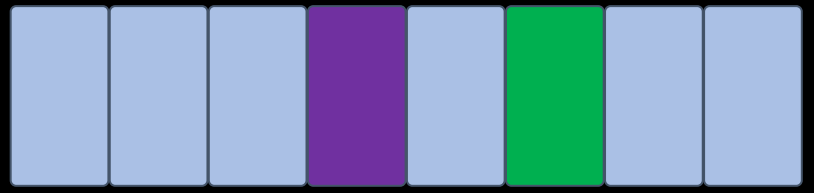
1



Партиции обслуживают операции:

- Каждая партиция – B+Tree, в узлах страницы.
- Чтение страницы за 1 IO операцию.
- Запись на диск – random write
  => копим изменения и checkpoint’им.
- Не хранят историю операций
  => глобальная блокировка для гарантий консистентности в снапшоте.

2

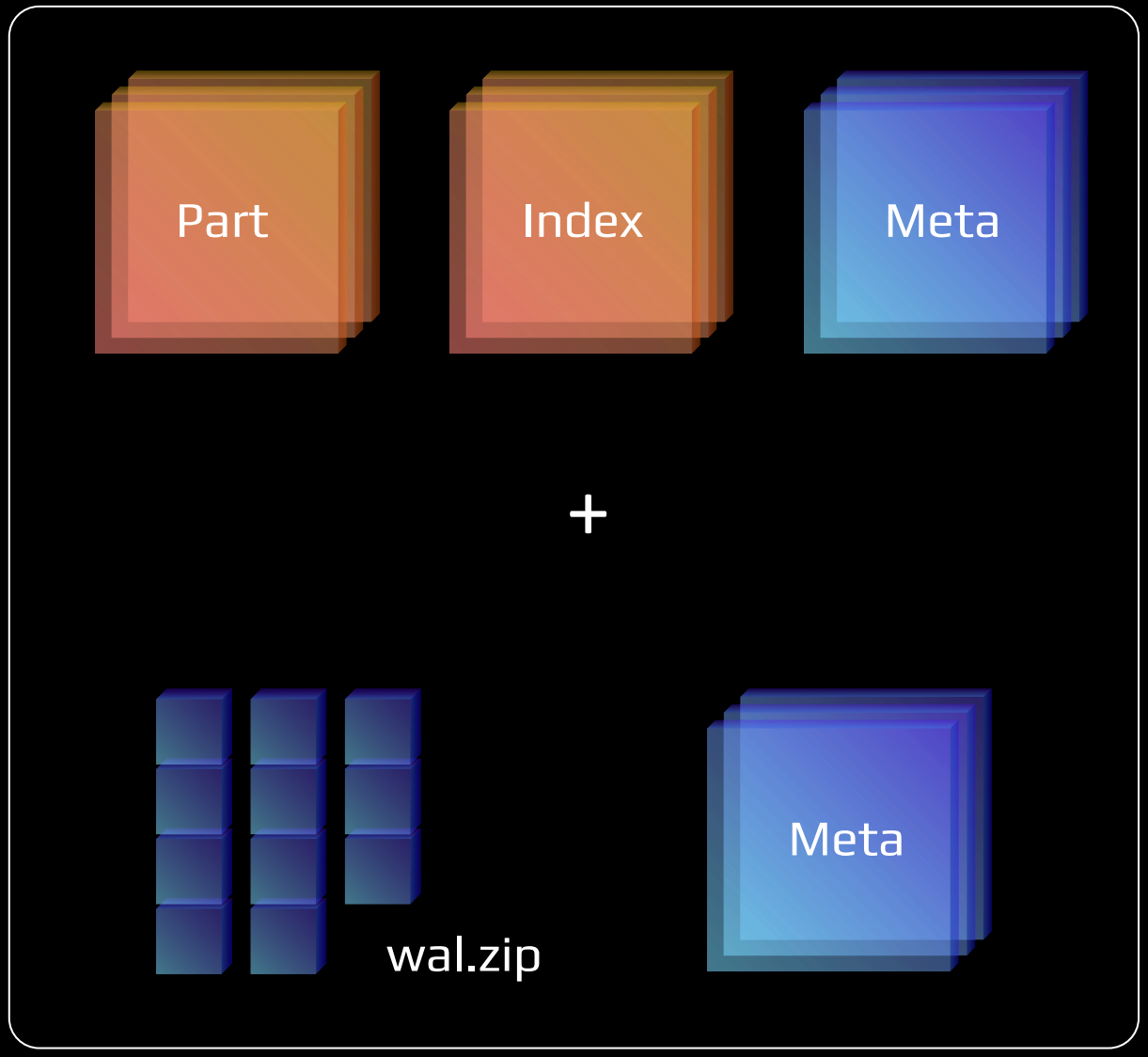


Write Ahead Log (WAL):

- Эффективная упреждающая запись update’ов в последовательный журнал операций.
- Используется для восстановления данных в партициях с момента checkpoint’а.
- Хранит историю операций
  => блокировка не нужна?

Apache Ignite: инкрементальный снапшот

- Сделали полный снапшот.
- Через время доложили к нему накопившиеся WAL’ы и мету.
- Можно делать несколько инкрементов.



Apache Ignite: инкрементальный снапшот

Полный

- Полная копия данных
- Тяжелая операция: PME, IO, RAM
- RPO и RTO – часы

Инкрементальный

- Только изменения в данных.
- Легкая неблокирующая операция (Нет StW).
- Предсказуемый RPO – минуты.
- Overhead на RTO – минуты.
- Транзакционно консистентный.
- Формирует точку восстановления по алгоритму Consistent Cut.

	RPO	RTO	Фича Data Grid	Полное падение	Support inmemory caches	Зачем надо?
Ребаланс	0	0			Да	Восстановиться без простоя при падении узла
Ячейки	0	0	Да		Да	Восстановиться без простоя без стойки
CDC	0	Время обработки сегмента		Частично	Да	Multi DC репликация
Online CDC	0	0	Да	Частично	Да	Multi DC репликация с меньшей задержкой
Полный снапшот	Часы	0		Да		Восстановить PDS при падении кластера
Полный снапшот. only-primary	Часы	Время ребаланса	Да	Да		Хранить меньше
Инкрементальный снапшот	Минуты	Время обработки WAL		Да		Снимать чаще

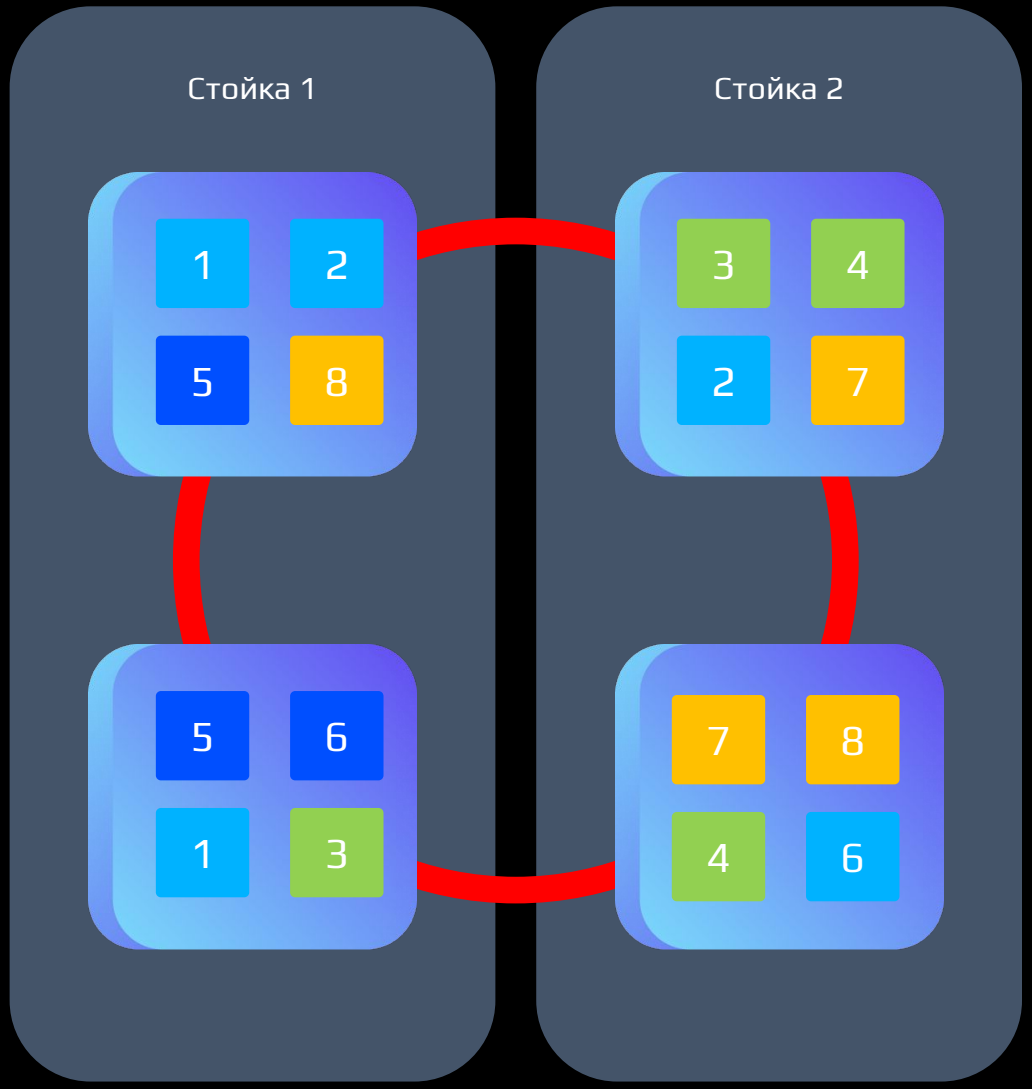
	RPO	RTO	Фича Data Grid	Полное падение	Support inmemory caches	Зачем надо?
Ребаланс	0	0			Да	Восстановиться без простоя при падении узла
Ячейки	0	0	Да		Да	Восстановиться без простоя без стойки
CDC	0	Время обработки сегмента		Частично	Да	Multi DC репликация
Online CDC	0	0	Да	Частично	Да	Multi DC репликация с меньшей задержкой
Полный снапшот	Часы	0		Да		Восстановить PDS при падении кластера
Полный снапшот. only-primary	Часы	Время ребаланса	Да	Да		Хранить меньше
Инкрементальный снапшот	Минуты	Время обработки WAL		Да		Снимать чаще

Cache dump

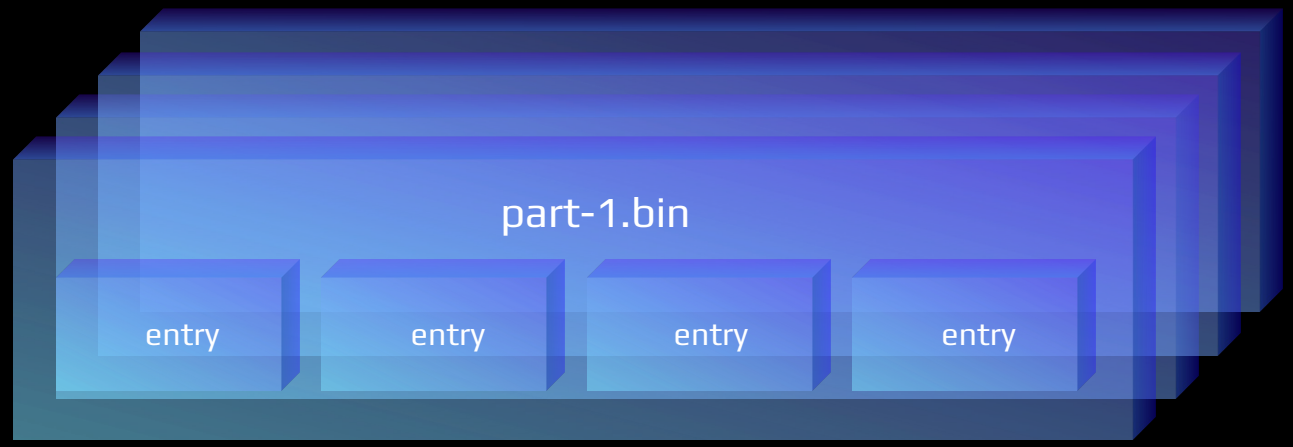
Apache Ignite

Cache dump: задачи

1. Уметь бекапить in-memory кеши.
2. Делать это консистентно.
3. С учетом транзакций.
4. Не прерывать пользовательскую нагрузку.
5. Восстанавливать данные на любую. топологию.
6. Поддержка CLI.
7. Nice to have - API offline обработки.



Apache Ignite: Cache dump



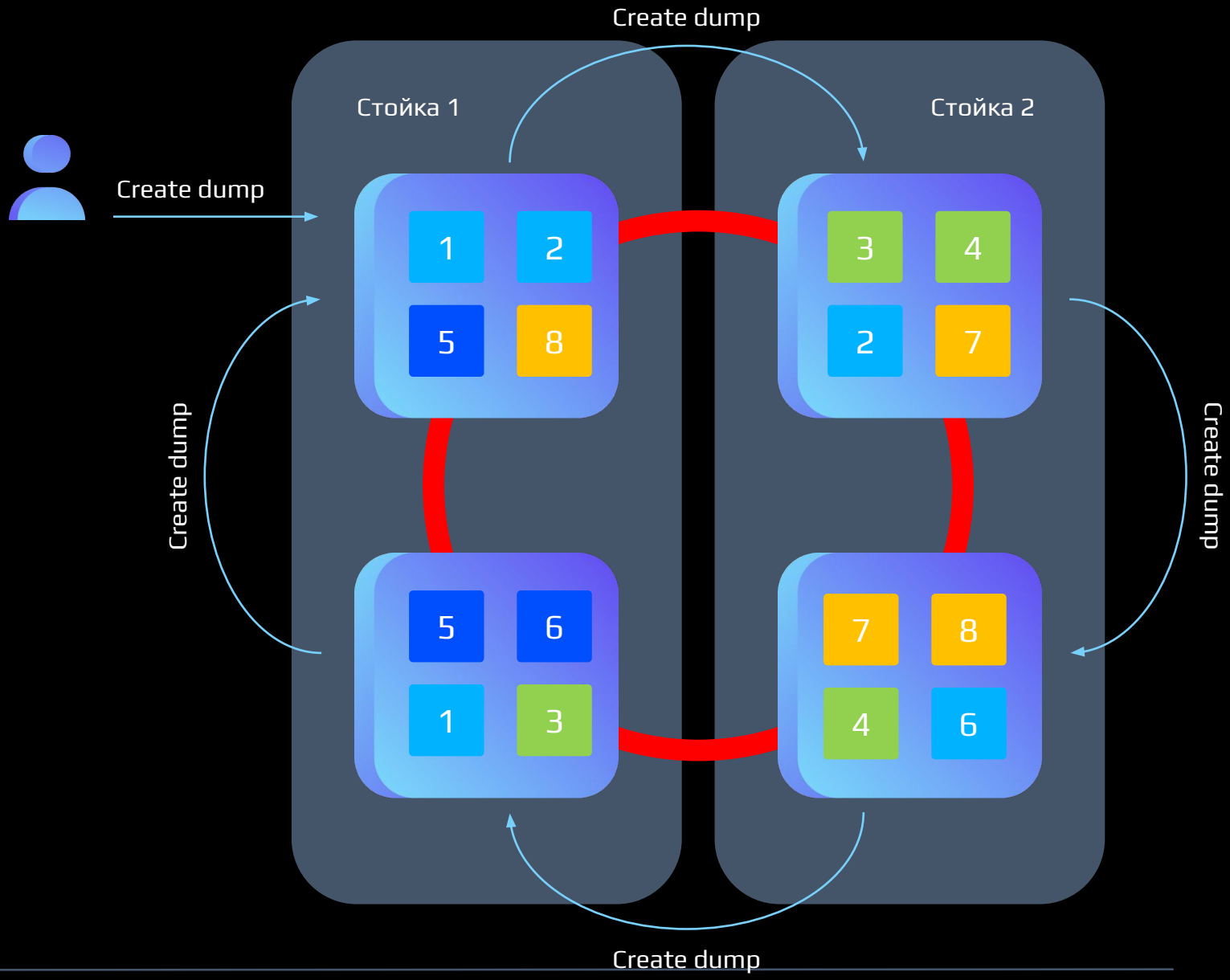
metadata

cache configs

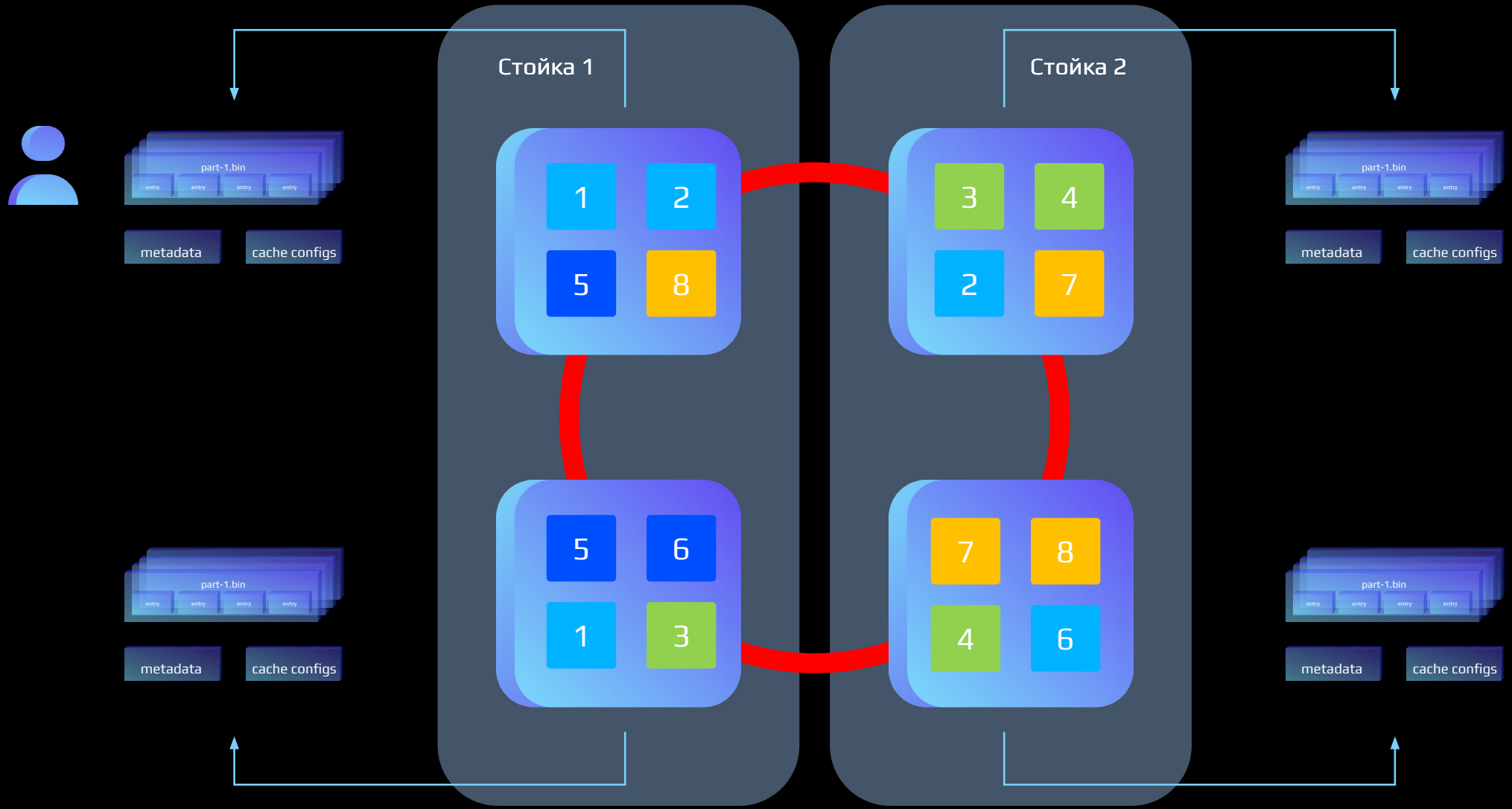
- Структура файла — список записей одна за одной, а не страницы данных.
- Метаданные.
- Конфигурации кэшей.

Apache Ignite: алгоритм снятия дампа

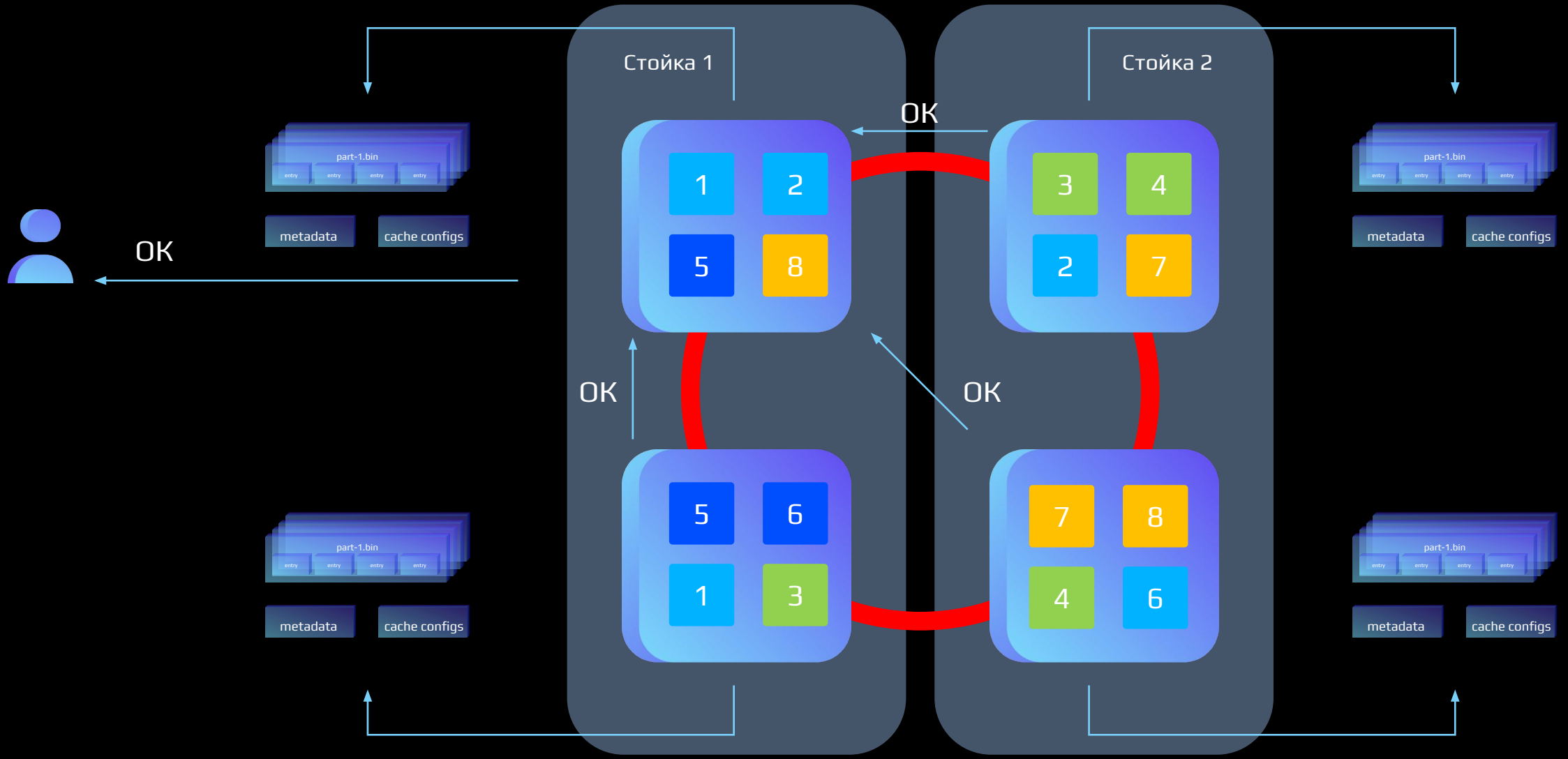
1. Администратор дает команду.
2. Системный процесс делает stop the world (PME).
3. Во время StW — данные консистентны, можно инициализировать писателей дампа.
4. Для обеспечения консистентности устанавливаем listener’ы на изменения кеша.
5. Пускаем транзакции.
6. Итерируемся по партиции и записываем row by row.
7. Изменившиеся данные записываем listener’ом.



Apache Ignite: алгоритм снятия дампа

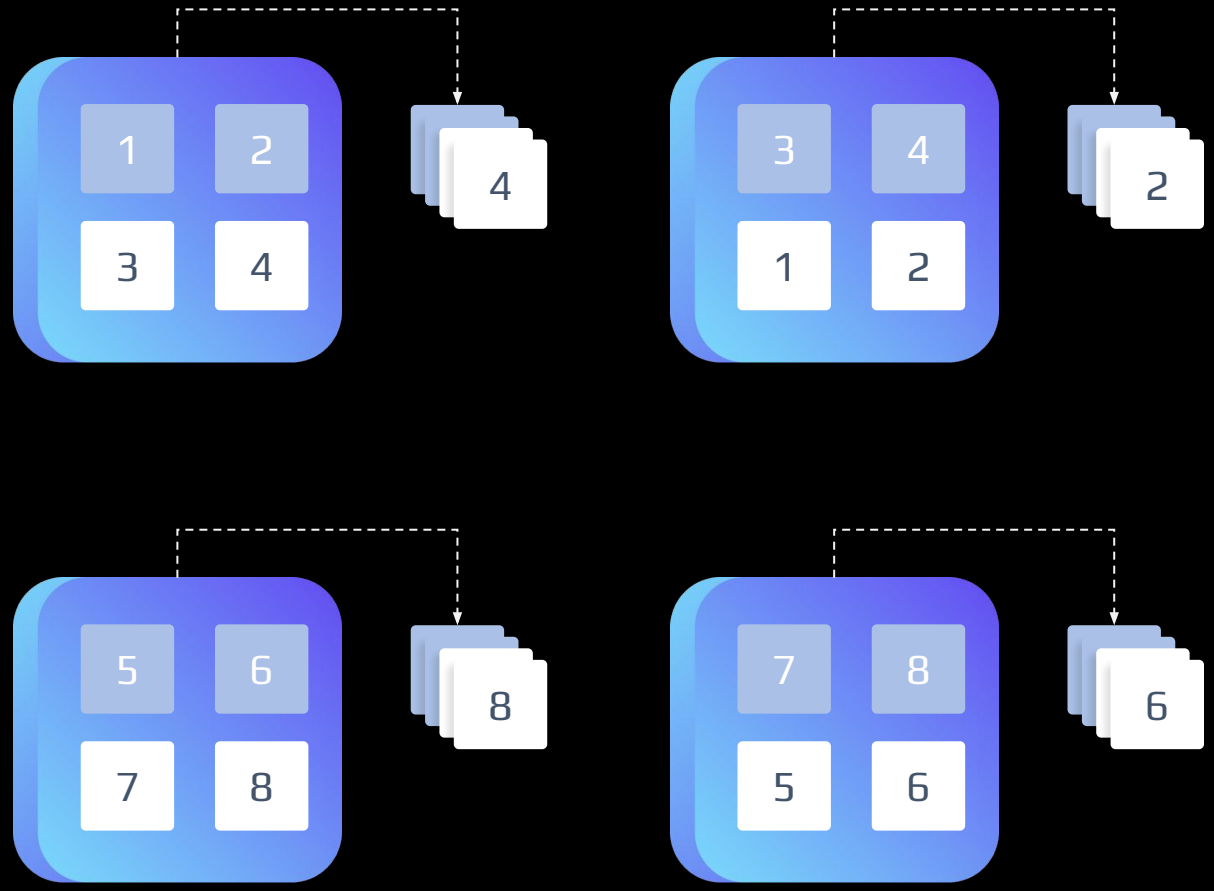


Apache Ignite: алгоритм снятия дампа



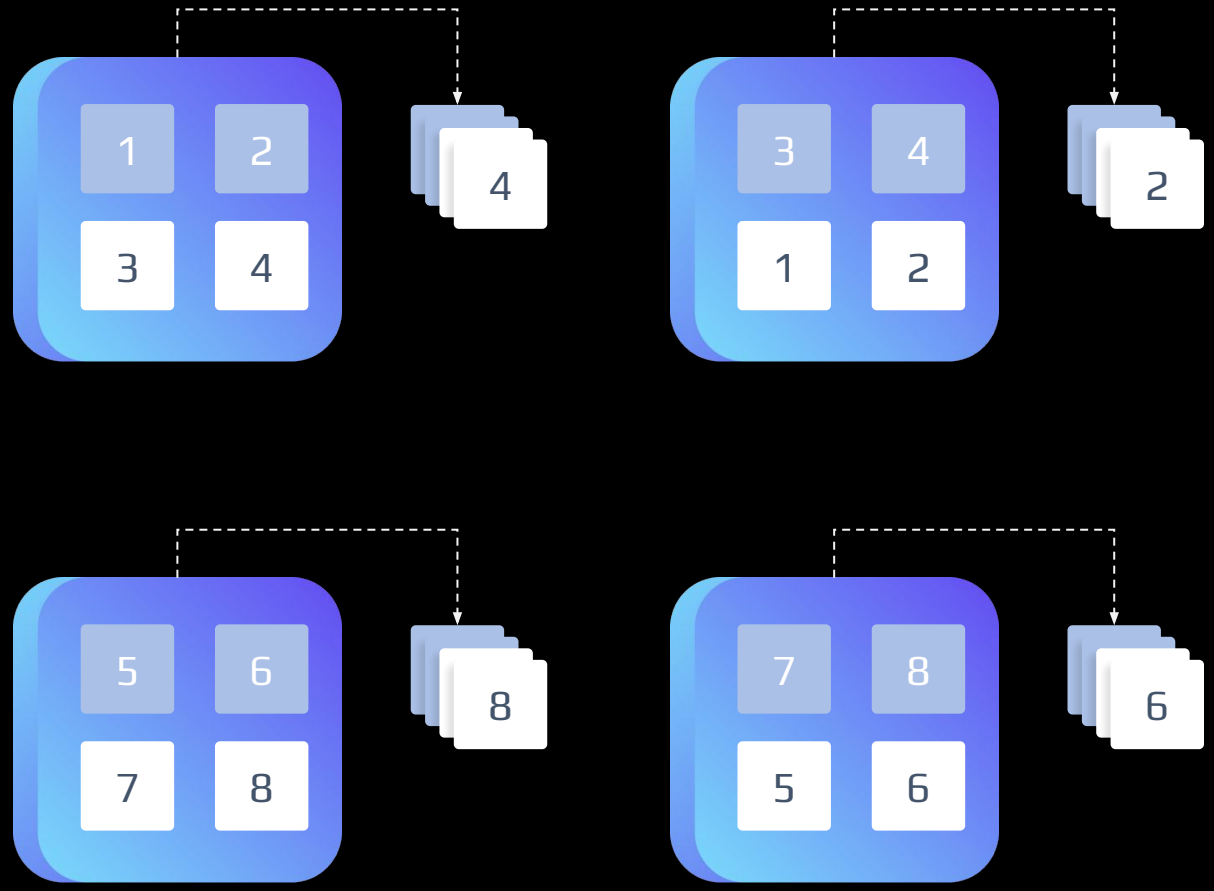
Apache Ignite: снятие дампа - тонкости

1. Откатиться, если где-то ошибка.
2. Не перегрузить ноды системным процессом - снятие дампа паразитная нагрузка.

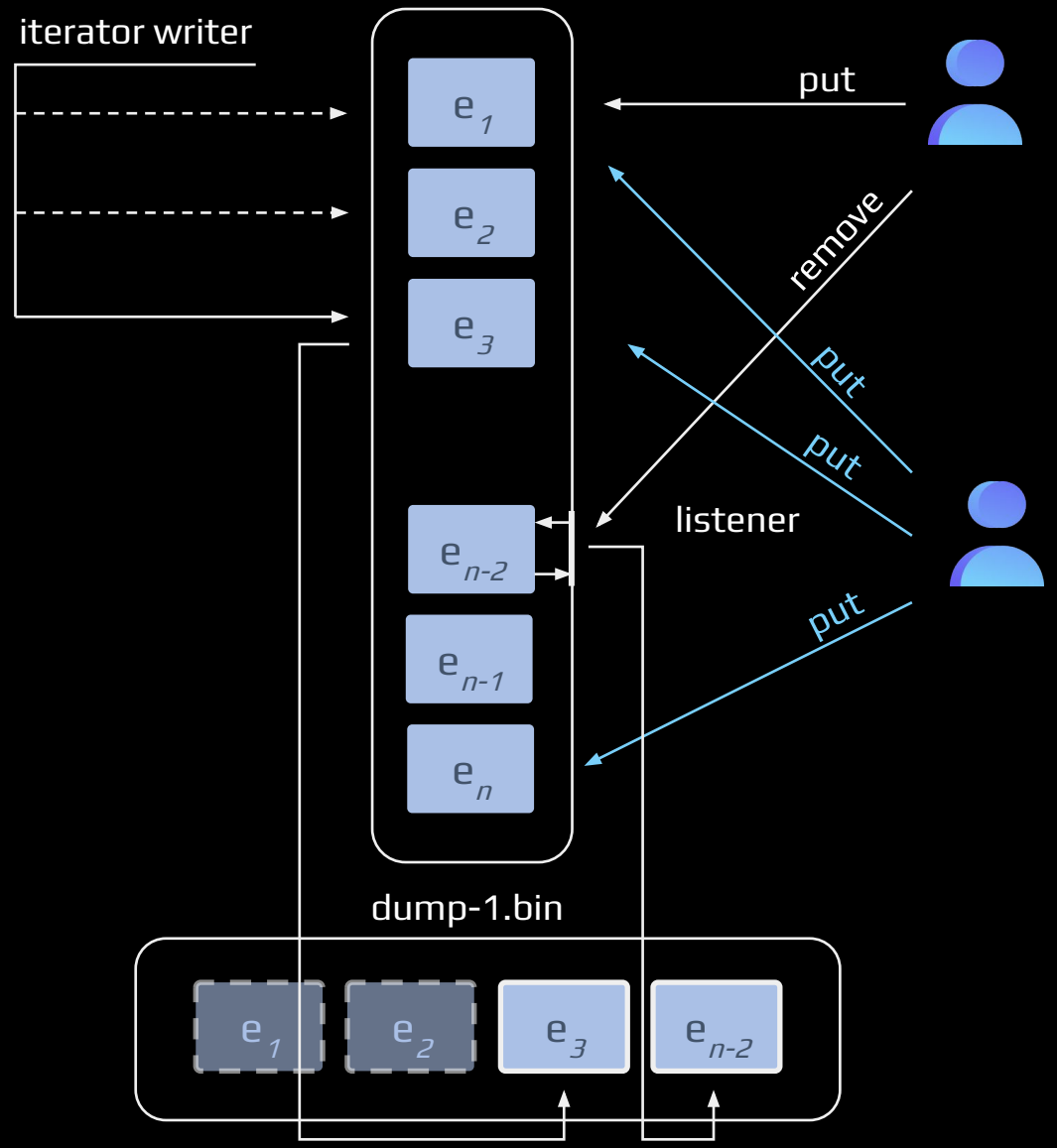


Apache Ignite: снятие дампа - тонкости

1. Откатиться, если где-то ошибка.
   - **DistributedProcess**
2. Не перегрузить ноды системным процессом - главное пользовательские операции!
   - **SnapshotExecutor - по задаче на каждую партицию.**
   - **Rate limiter - ограничивает скорость записи на ноде.**

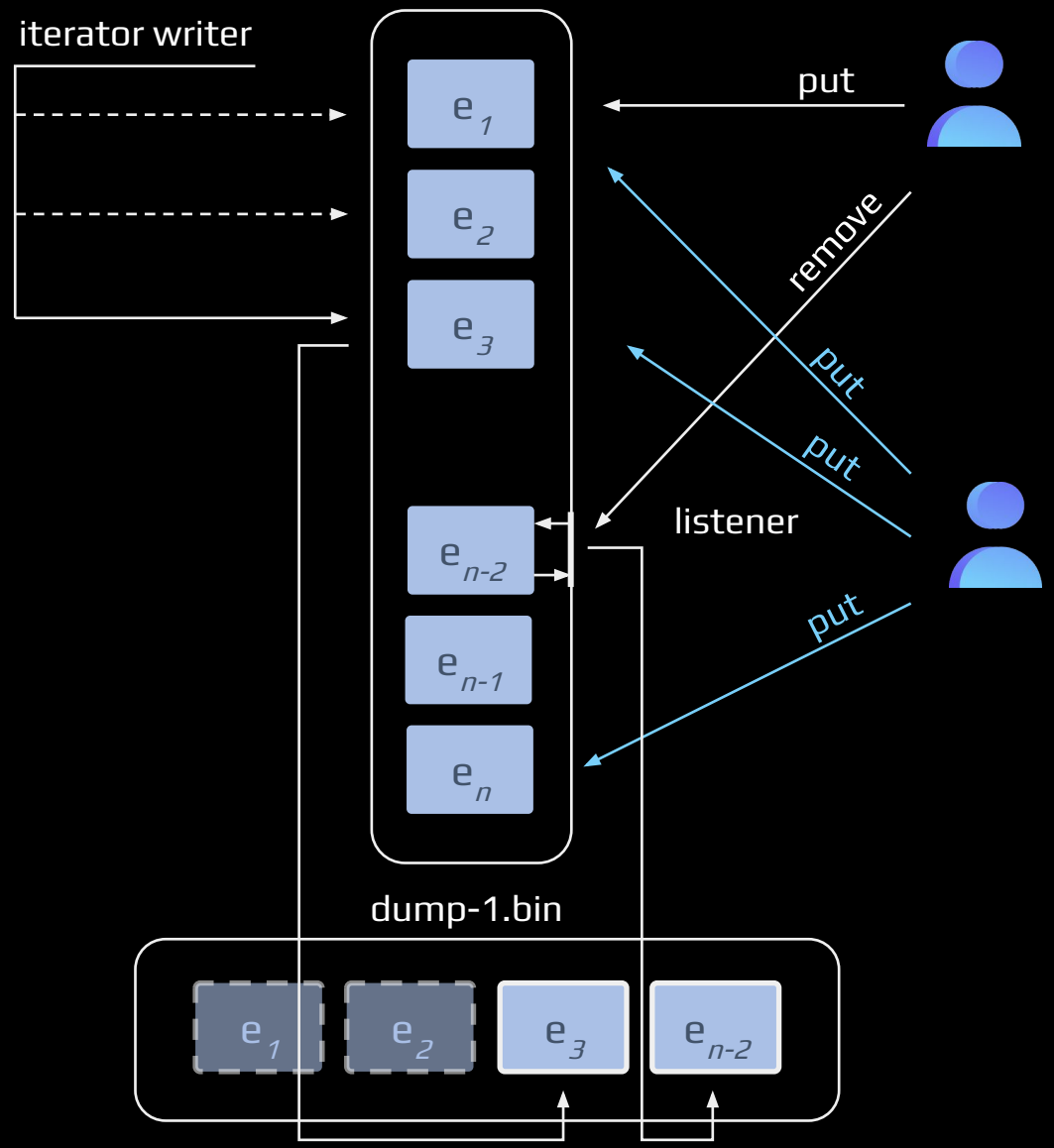


Дамп одной партиции



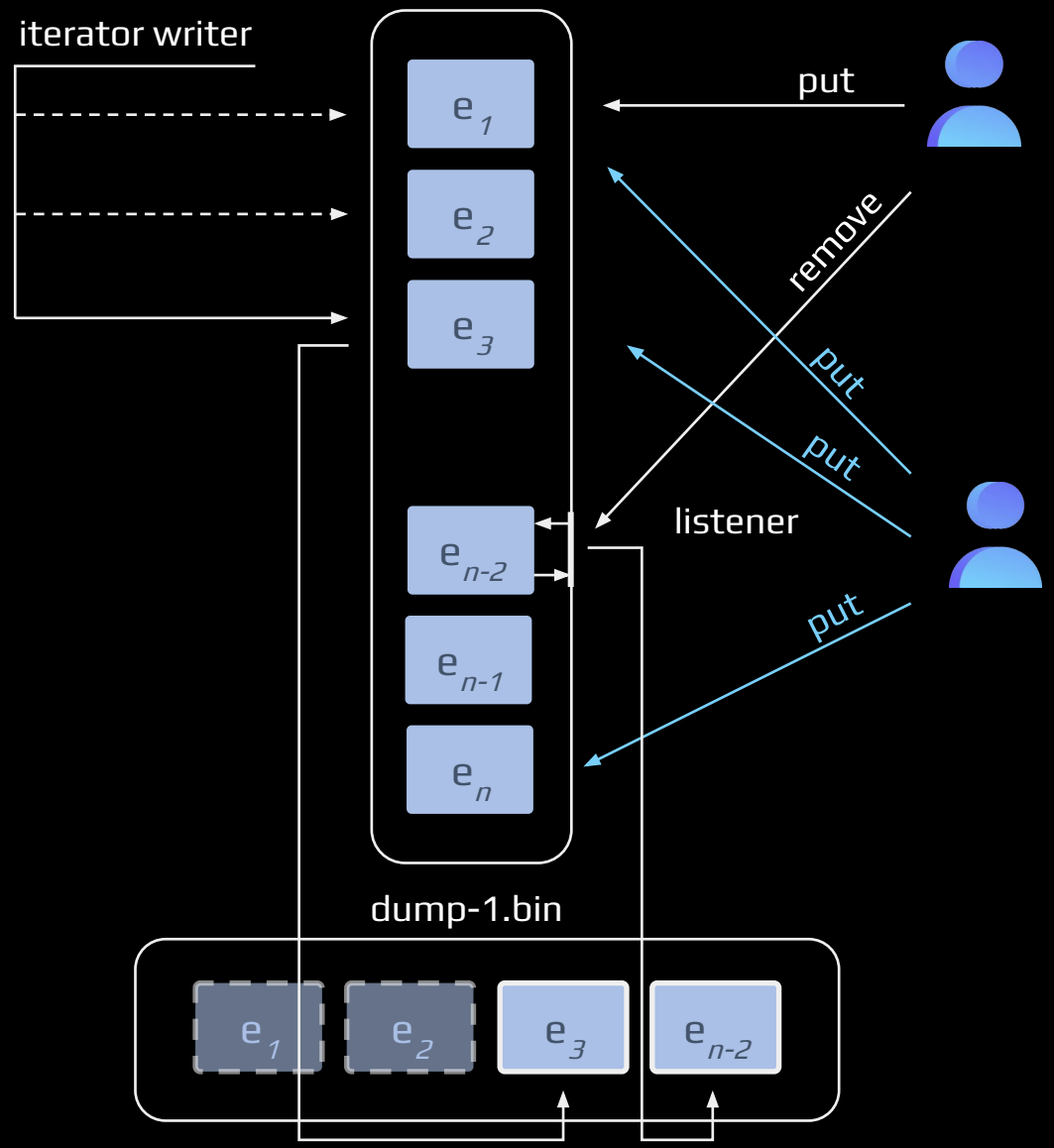
Эффективная фильтрация

Просто запоминать все ключи которые изменились за время снятия дампа очень дорого по памяти.



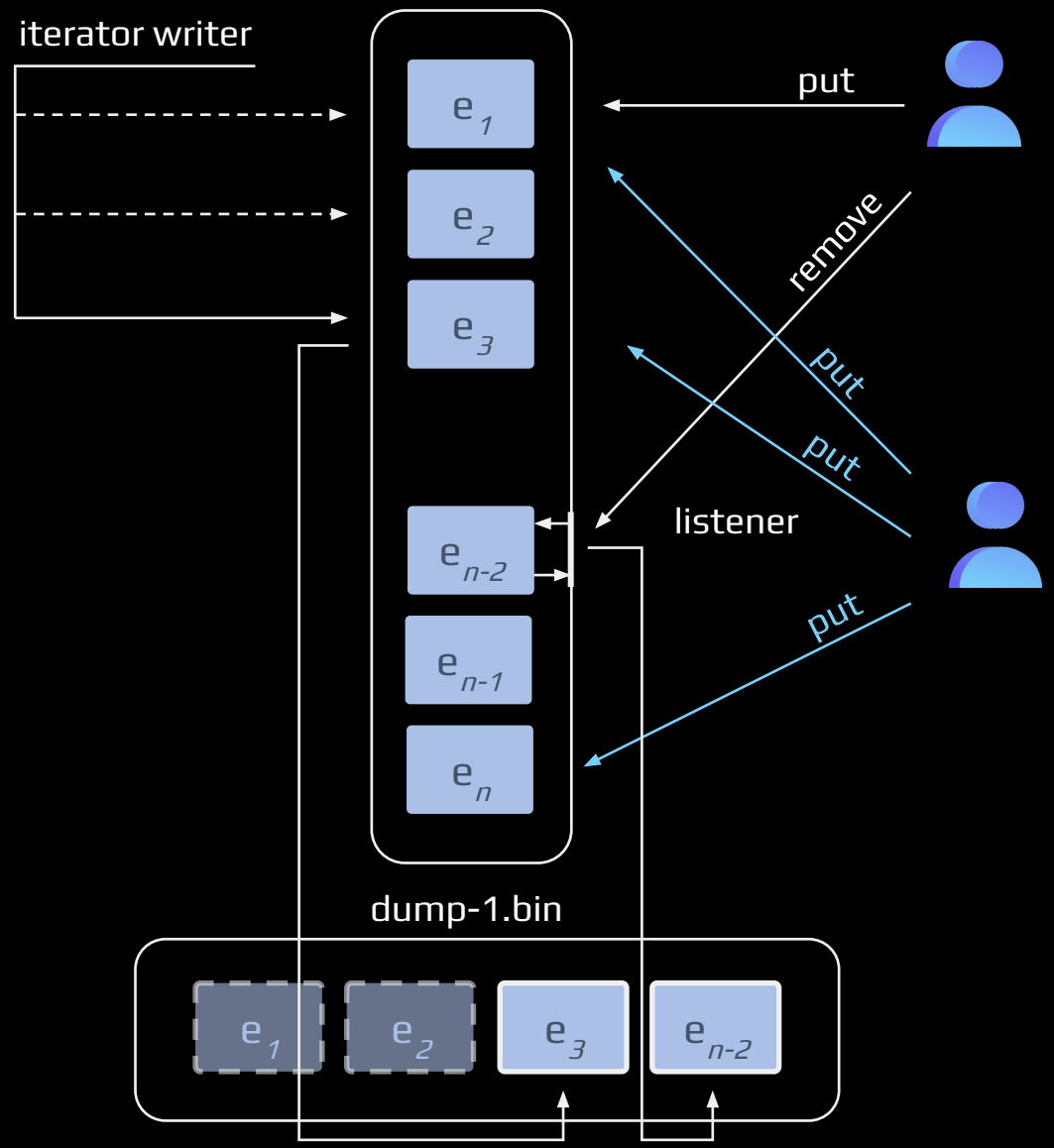
Эффективная фильтрация - требования

1. Итератор по партиции нетранзакционный - видим изменившиеся данные.



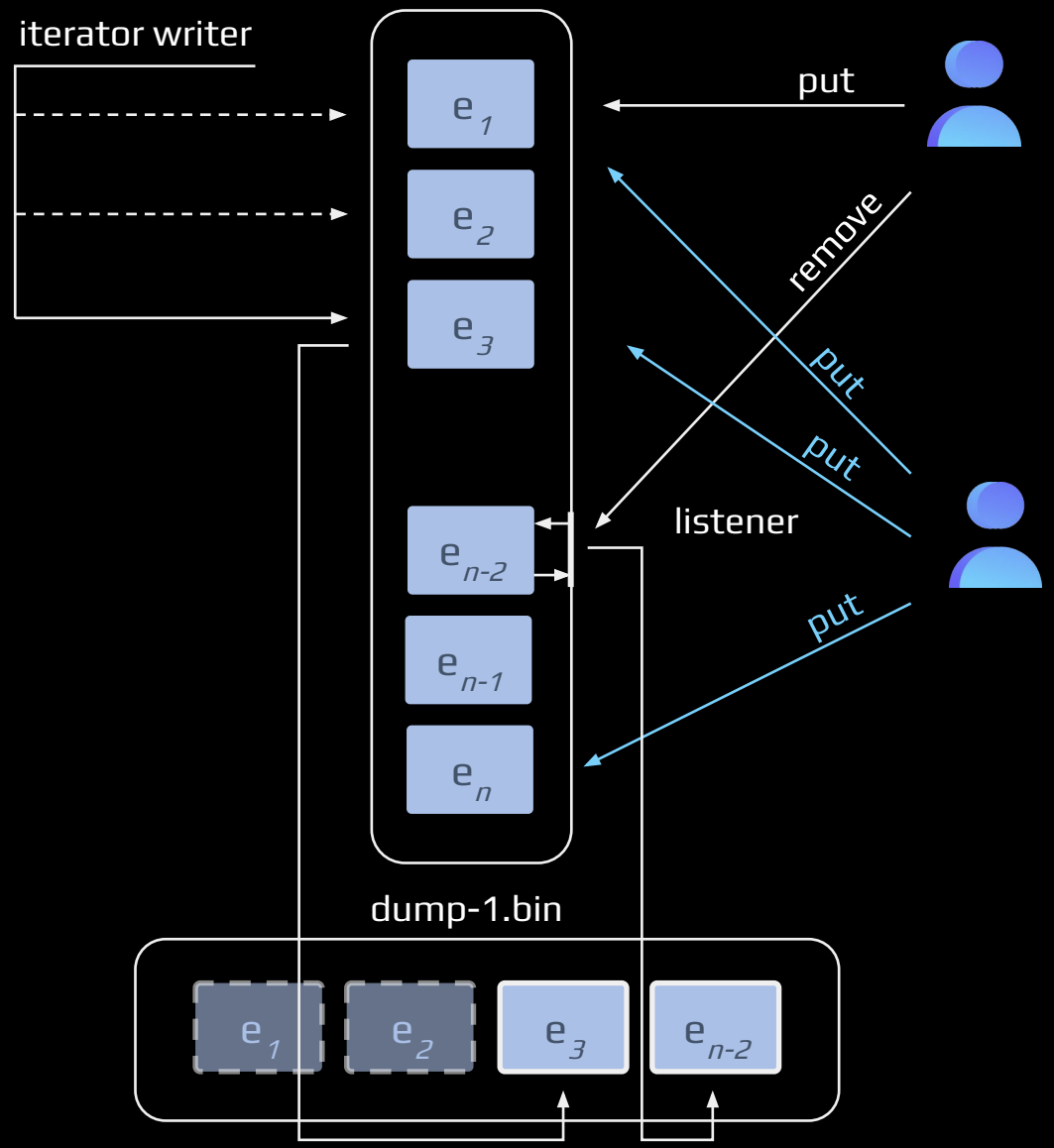
Эффективная фильтрация

1. Итератор по партиции нетранзакционный - видим изменившиеся данные.
2. Listener в потоке который обслуживает запрос на изменение данных.



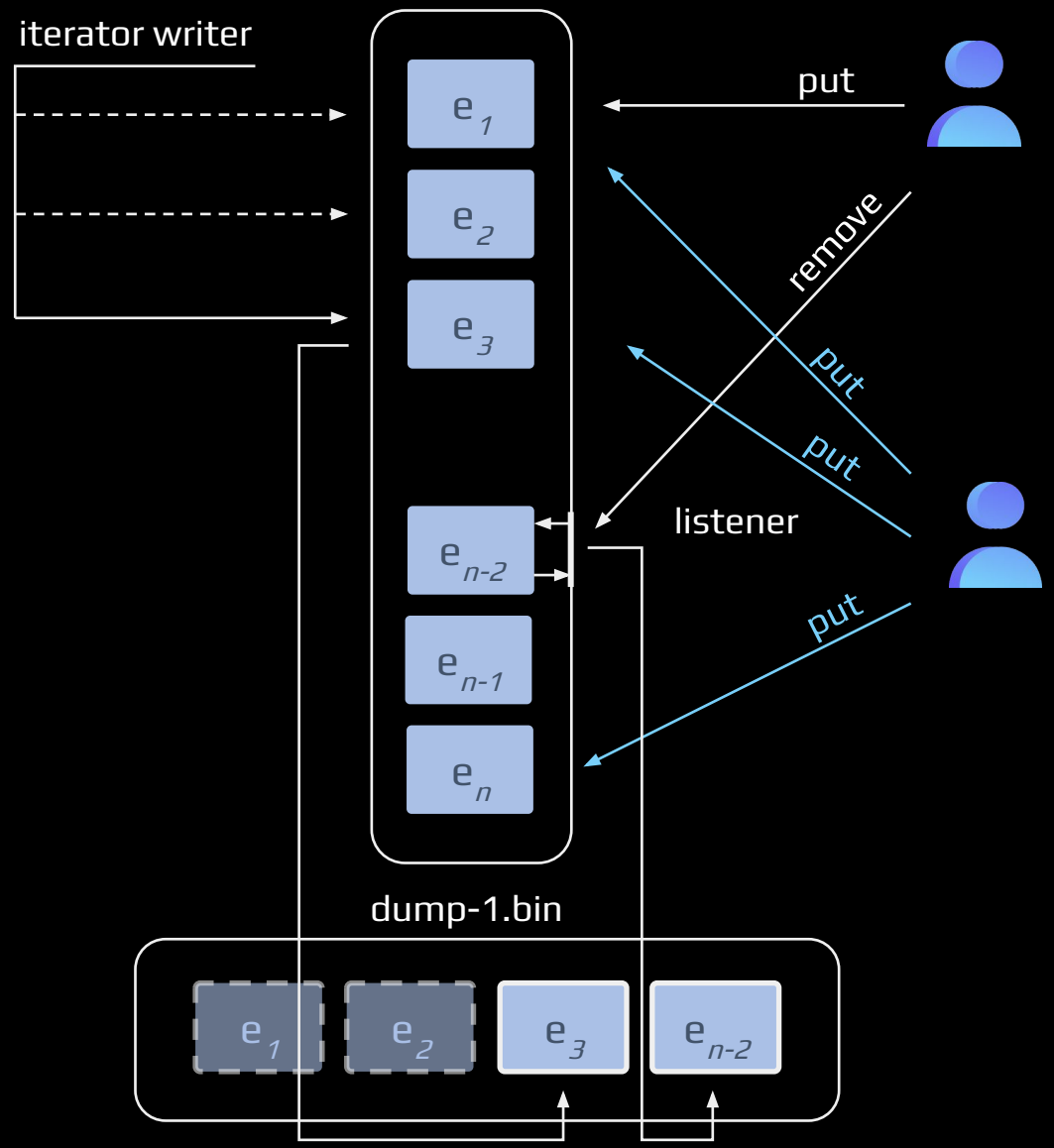
Эффективная фильтрация

1. Итератор по партиции нетранзакционный - видим изменившиеся данные.
2. Listener в потоке который обслуживает запрос на изменение данных.
3. Итератор сортированный(B+ дерево).



Эффективная фильтрация

1. Итератор по партиции нетранзакционный - видим изменившиеся данные.
2. Listener в потоке который обслуживает запрос на изменение данных. Должен сохранить entry ДО изменения.
3. Итератор сортированный(B+ дерево).
4. У каждой записи есть уникальная версия. Версия выдается на primary.



Работа итератора

1. Фильтруем по версии на primary.
2. Запоминаем последнее записанное итератором.
3. Фильтруем все обработанное listener’ом.

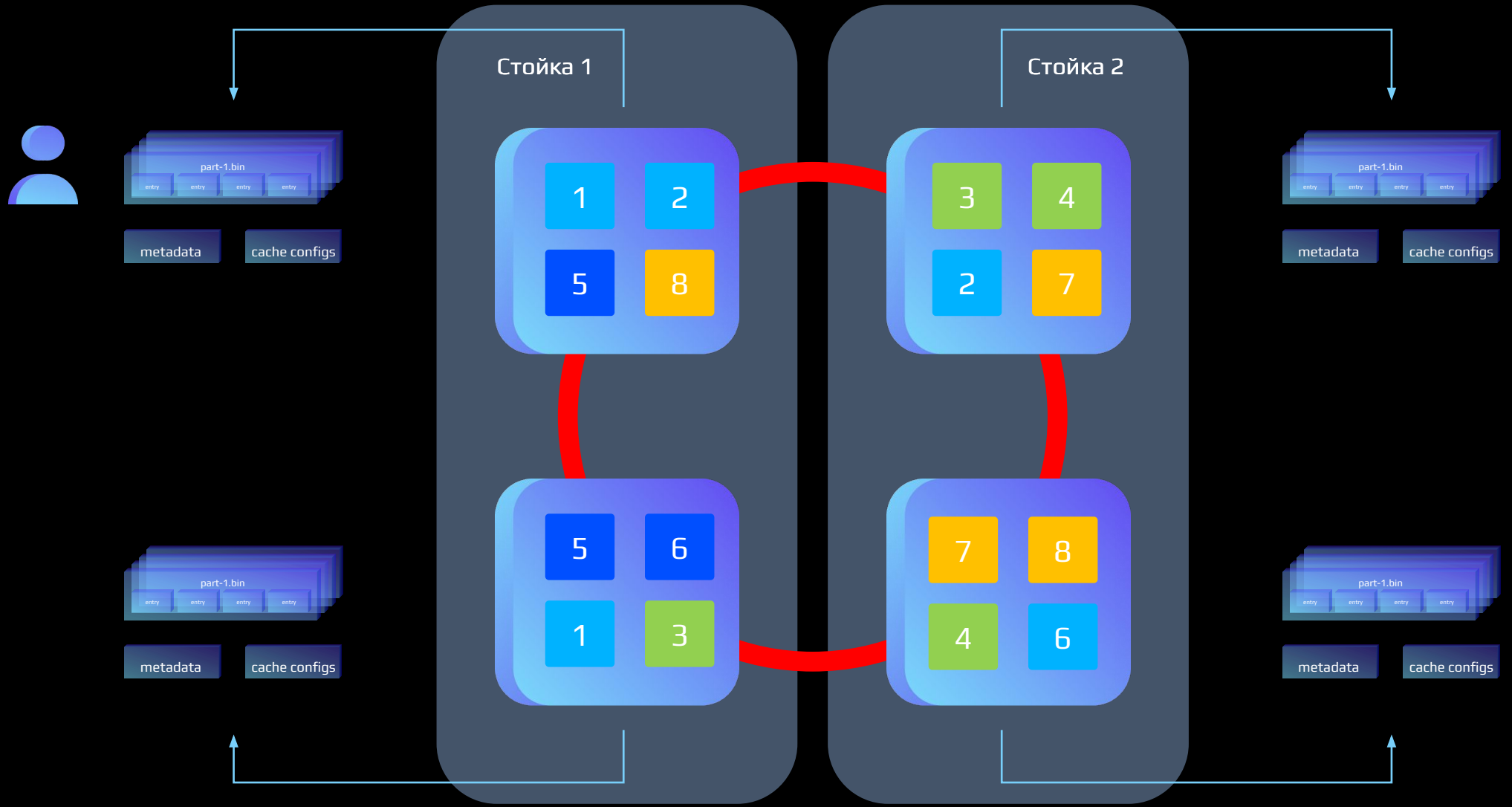
```
if (afterStart(version))
    return;

synchronized {
    iterLastWritten = entry;
    if (writtenByListener.contains(key))
        return;
    writeToFile(entry);
}
```

Работа listener’а

1. Фильтруем по версии на primary.
2. Пропускаем обработанное итератором - нужно сравнить ключи.
3. Запоминаем измененное.
4. Отбрасываем все новое - созданное после начала снятия дампа.

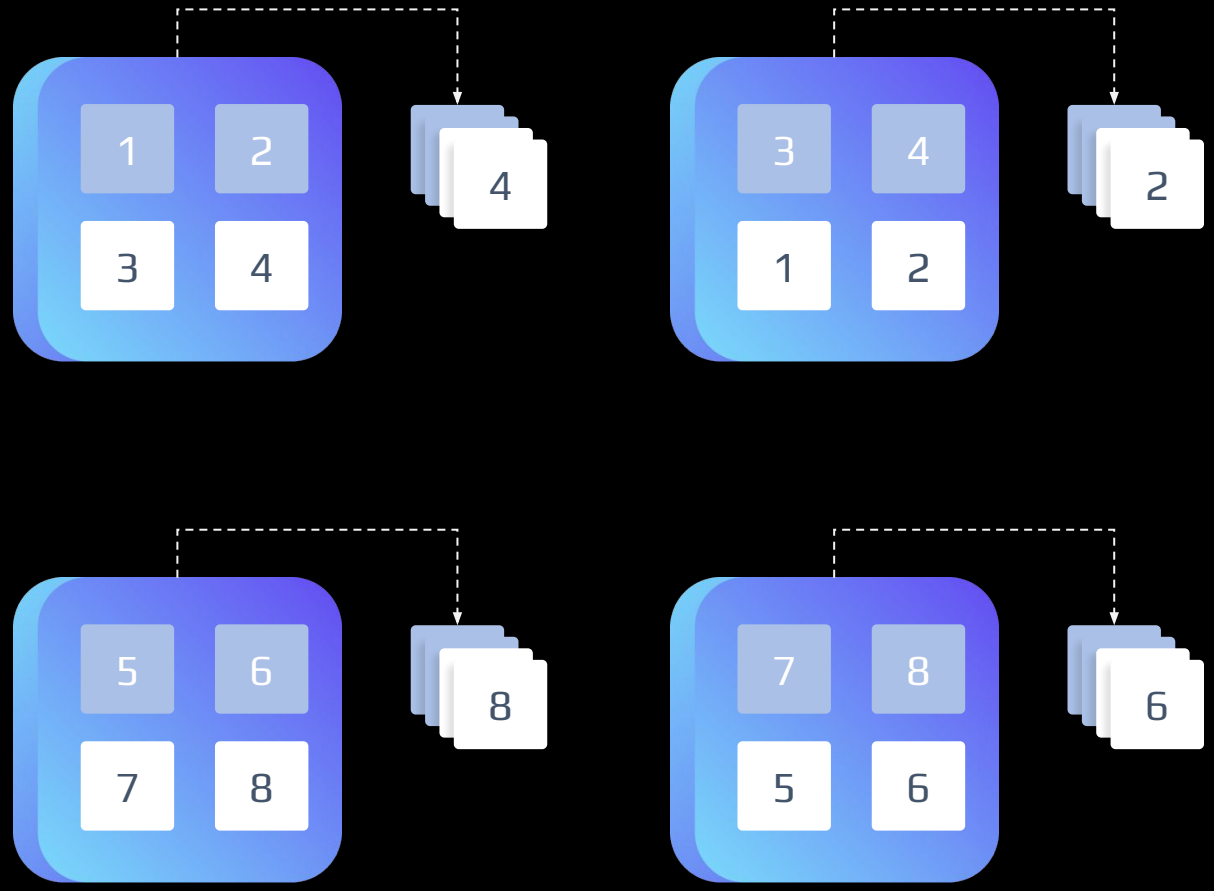
```
if (closed || afterStart(prevVersion))
    return;

synchronized {
    if (iterLastWritten.compare(key) >= 0)
        return;
    if (!writtenByListener.add(key))
        return;
    if (prevVal == null)
        return;
    writeToFile(entry);
}
```

Запустили распределенный алгоритм

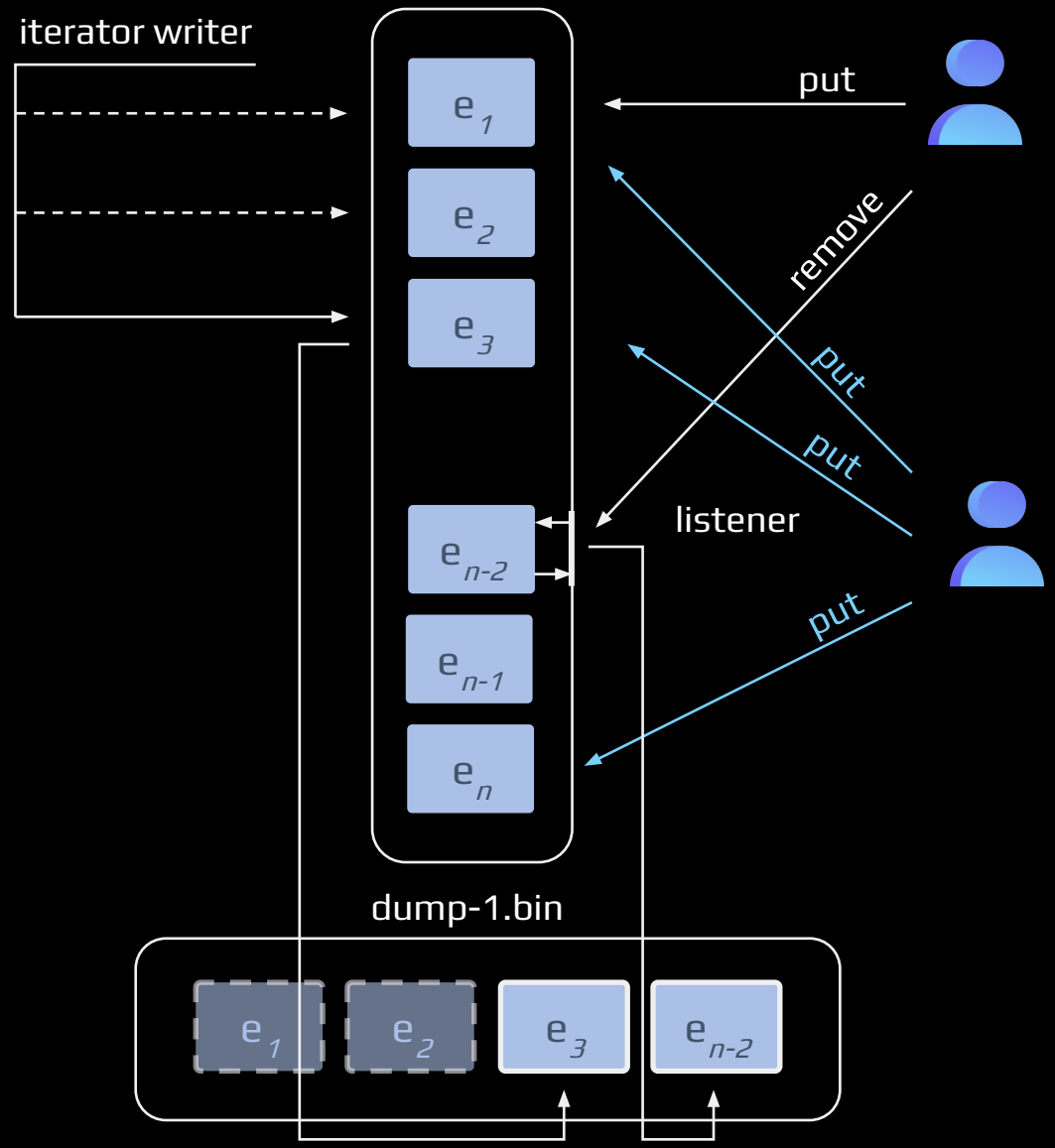


Дампим все на ноде

1. Задачи по дампу каждой партиции в SnapshotExecutor.
2. Нагрузка на диск ограничина rate limiter'ом.

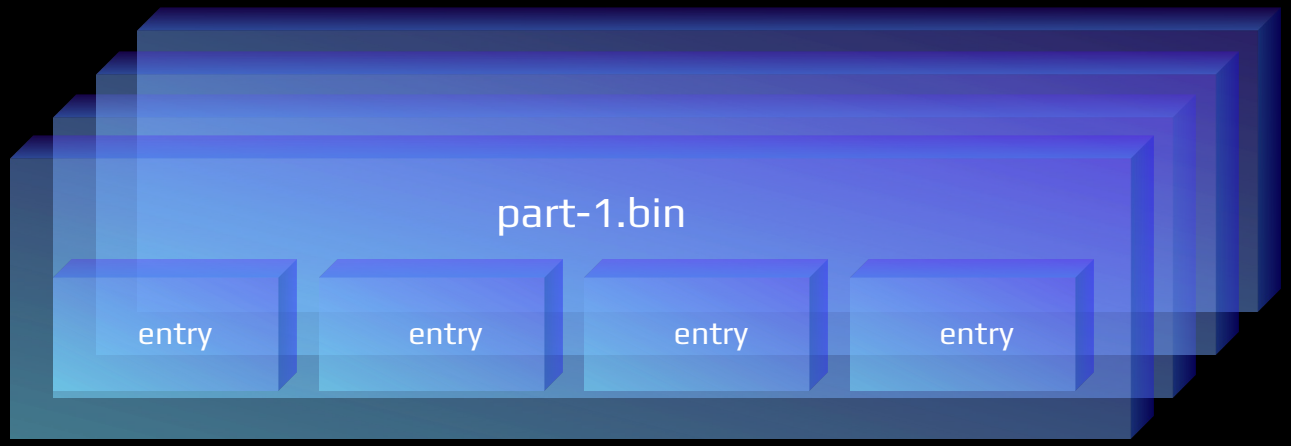


Каждую партицию пишем максимально эффективно



Ура!
Дамп транзакционно консистентный на начало снятия и не убивает перформанс.

API



metadata

cache configs

Из этих файликов надо восстановить данные

API

```
interface DumpConsumer {
    void start();
    void onMeta(Iterator<Metadata> meta);
    void onConfigs(
        Iterator<CacheConfig> cfg
    );
    void onPartition(
        int cache,
        int part,
        Iterator<DumpEntry> data
    );
    void stop();
}
```

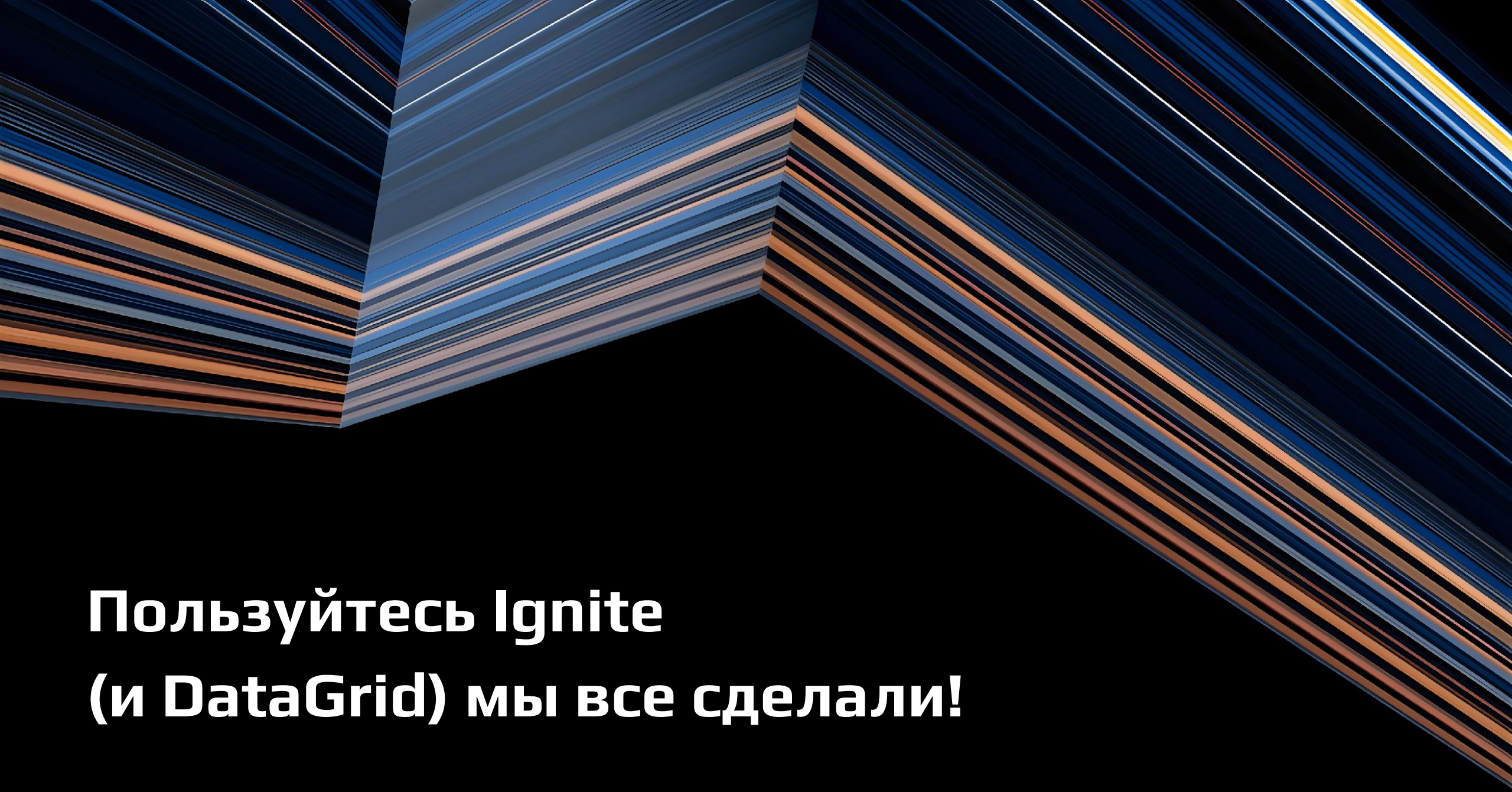
```
class DumpReaderConfig {
  String dir;
  DumpConsumer cnsmr;
  int threadCnt;
}

class DumpReader {
  DumReader(DumpReaderConfig cfg);
  void run() {
    ...
  }
}
```

Data Grid: CLI

Утилита - ignite-dump-reader:

- В удобном виде позволяет запустить open source DumpReader.
- Восстановление данные через тонкий клиент, клиентскую ноду.
- Вывод данных в json формате.
- Запуск собственного DumpConsumer.
- Восстановление состояния CDC.

	RPO	RTO	Фича Data Grid	Полное падение	Support inmemory caches	Зачем надо?
Ребаланс	0	0			Да	Восстановиться без простоя при падении узла
Ячейки	0	0	Да		Да	Восстановиться без простоя без стойки
CDC	0	Время обработки сегмента		Частично	Да	Multi DC репликация
Online CDC	0	0	Да	Частично	Да	Multi DC репликация с меньшей задержкой
Полный снапшот	Часы	0		Да		Восстановить PDS при падении кластера
Полный снапшот. only-primary	Часы	Время ребаланса	Да	Да		Хранить меньше
Инкрементальный снапшот	Минуты	Время обработки WAL		Да		Снимать чаще
Cache dump	Часы	Минуты	Да	Да	Да	In-memory cache snapshots



Пользуйтесь Ignite (и DataGrid) мы все сделали!

PLATFORM V

Спасибо!