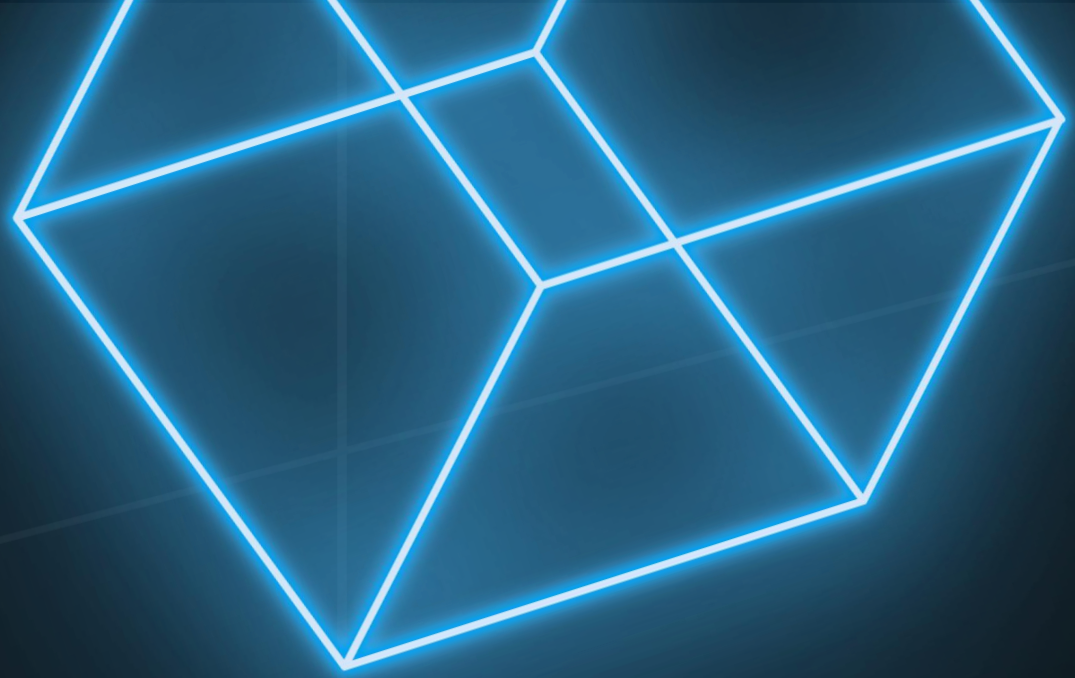
МИР Plat.Form



По следам 1BRC

трюки и подходы к оптимизации производительности

Обо мне

linkedin.com/in/alantsov

Мир Plat.Form

mir-platform.ru

1BRC

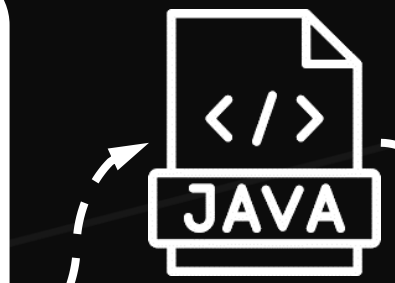
github.com/gunnarmorling/1brc

О чем речь?

- Дана очень простая задача с фиксированными условиями
- Насколько далеко можно зайти в погоне за производительностью?
- Когда и как пользоваться такими подходами в реальном коде?
- Рассмотрим пример похожей задачи из реального опыта

Формулировка задачи

```
Hamburg;12.0
Bulawayo;8.9
Palembang;38.8
Hamburg;13.4
...
```



~ 13 GB

```
Hamburg=
-23.0 18.0 59.2
Bulawayo=
-16.2 26.0 67.3
...
```

Для каждого города из CSV-файла рассчитать min/avg/max температуру
Надо сделать как можно быстрее (минимизируем wall-clock)

Оборудование

- **CPU:** Intel 12900k (8P + 8E = 24 потока исполнения)
- **RAM:** DDR5, 32GB
- **SSD:** NVMe PCIe 4.0 Samsung 980 PRO 512GB
- **OS:** Linux, Ubuntu 22.04 (kernel 6.8.0-40)
- **JDK:** OpenJDK 21 (liberica)

Простейшая реализация

```
record Measurement(String station, double value) { ... }
record ResultRow(double min, double mean, double max) { ... }

Map<String, ResultRow> measurements = new TreeMap<>(
        Files.lines(Paths.get(FILE))
                .map(l -> new Measurement(l.split(";")))
                .collect(groupingBy(Measurement::station, collector)));

System.out.println(measurements);
```

Простейшая реализация

```
record Measurement(String station, double value) { ... }
record ResultRow(double min, double mean, double max) { ... }

Map<String, ResultRow> measurements = new TreeMap<>(
        Files.lines(Paths.get(FILE))
                .map(l -> new Measurement(l.split(";")))
                .collect(groupingBy(Measurement::station, collector)));

System.out.println(measurements);
```

Простейшая реализация

```
record Measurement(String station, double value) { ... }
record ResultRow(double min, double mean, double max) { ... }

Map<String, ResultRow> measurements = new TreeMap<>(
        Files.lines(Paths.get(FILE))
             .map(l -> new Measurement(l.split(";")))
             .collect(groupingBy(Measurement::station, collector)));

System.out.println(measurements);
```

Простейшая реализация

```
record Measurement(String station, double value) { ... }
record ResultRow(double min, double mean, double max) { ... }

Map<String, ResultRow> measurements = new TreeMap<>(
        Files.lines(Paths.get(FILE))
                .map(l -> new Measurement(l.split(";")))
                .collect(groupingBy(Measurement::station, collector)))

System.out.println(measurements)
```

Простейшая реализация

```
record Measurement(String station, double value) { ... }
record ResultRow(double min, double mean, double max) { ... }

Map<String, ResultRow> measurements = new TreeMap<>(
        Files.lines(Paths.get(FILE))
             .map(l -> new Measurement(l.split(";")))
             .collect(groupingBy(Measurement::station, collector)))

System.out.println(measurements)
```

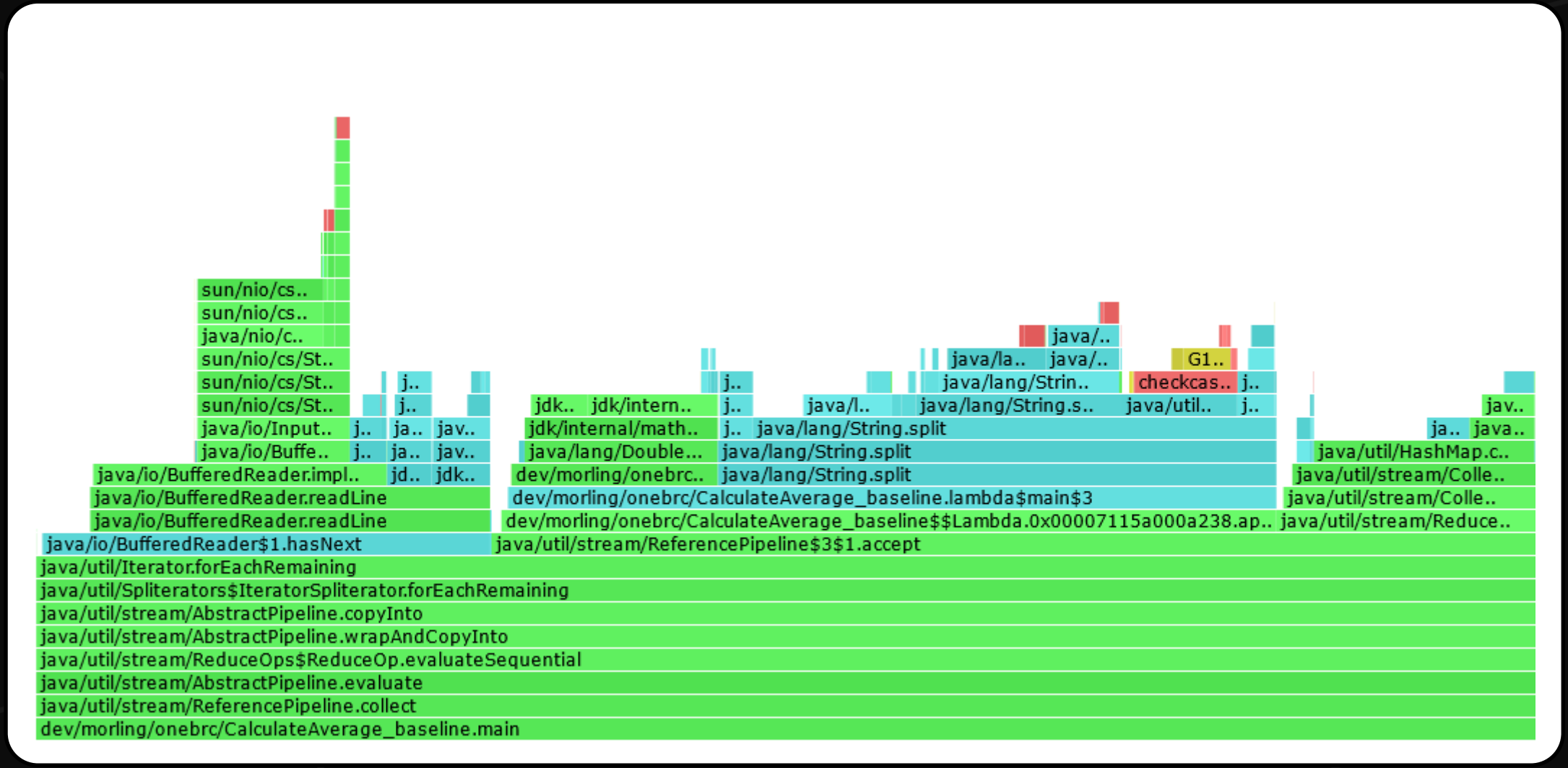
~100 секунд

Простейшая реализация

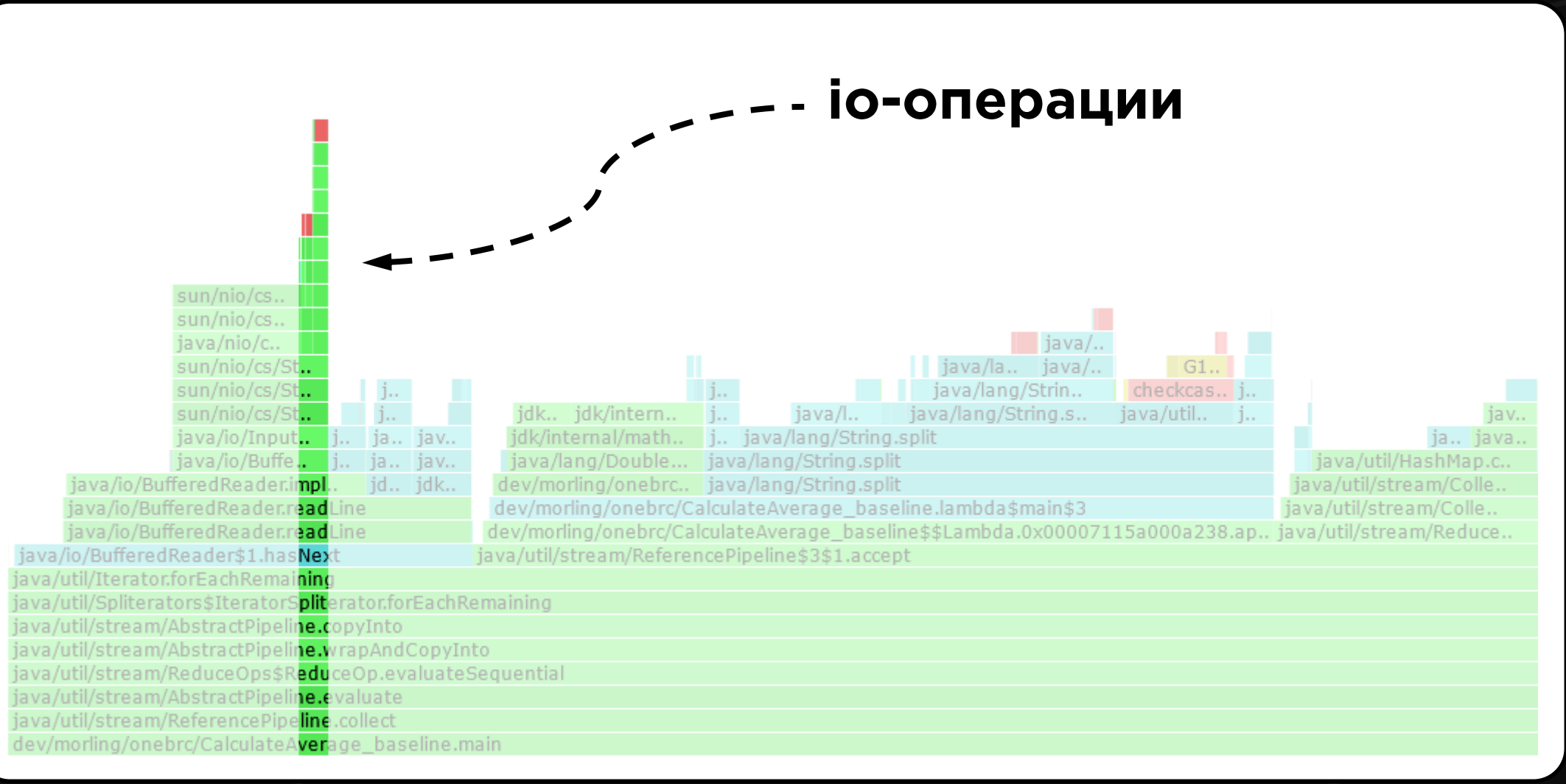
```
record Measurement(String station, double value) { ... }
record ResultRow(double min, double mean, double max) { ... }

Map<String, ResultRow> measurements = new TreeMap<>(
        Files.lines(Path.of(FILE))
                .map(l -> new Measurement(l.split(";")))
                .collect(groupingBy(Measurement::station, collector)))

System.out.println(measurements)
```

Это io-bound или cpu-bound задача?

Первичный анализ

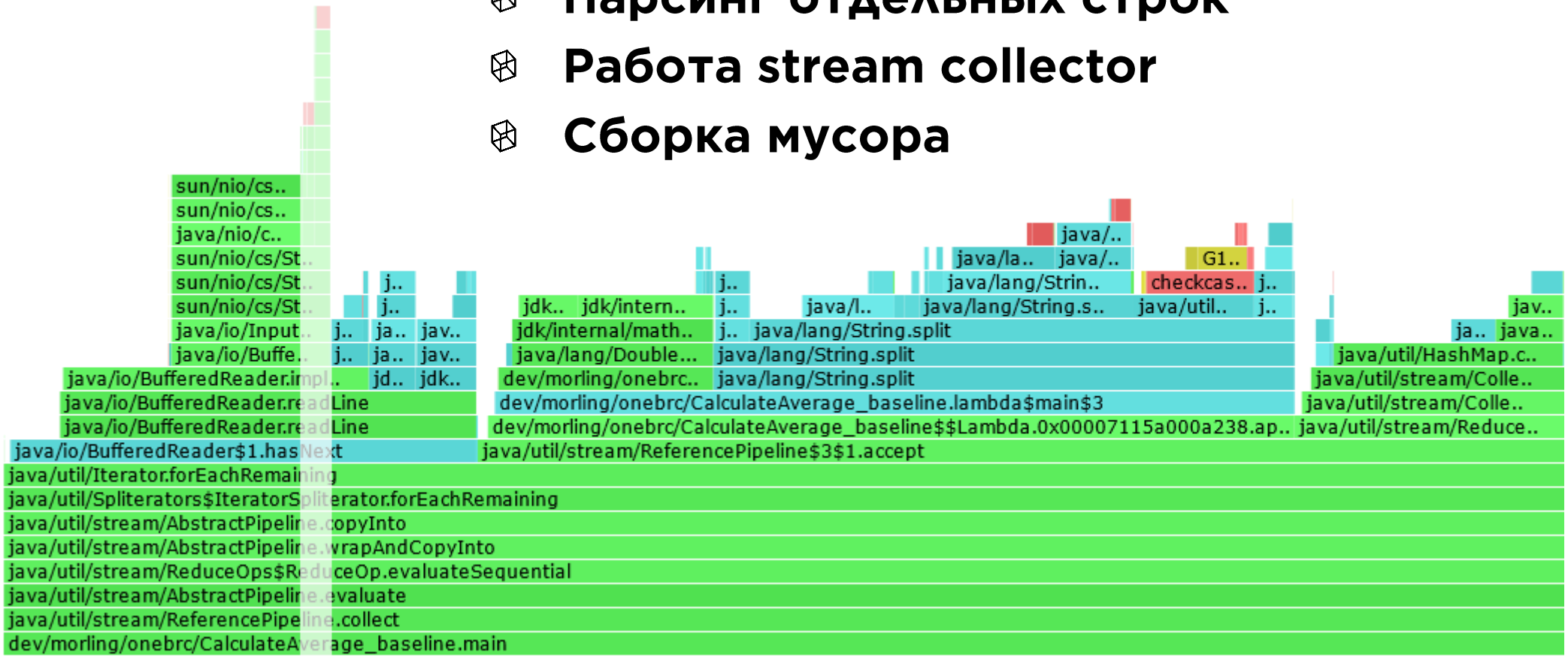


Первичный анализ



Первичный анализ

- Парсинг отдельных строк
- Работа stream collector
- Сборка мусора



Baseline + параллельность

- Просто делим файл на 24 куска одинакового размера
- Каждый поток обрабатывает только свой кусок
- Для каждого города в каждом куске файла считаем min / max / sum / count
- Объединить результаты в конце для каждого города

Чтение из файла – какое API выбрать?

- `FileInputStream` и `FileChannel`
  потоковые, не безопасны из нескольких потоков

- `MappedByteBuffer` (вызов FileChannel.map)
  с помощью ОС представляем часть файла как массив

  ограничение на размер массива – не более 2 Гб

  возможность использования ОС-специфичного функционала

  библиотеки с удобными интерфейсами – например, AtomicBuffer из agrona

- `MemorySegment` (Foreign Function and Memory API)
  нет ограничения на размер массива в 2 Гб

Параллельность - реализация

```
Map<String, ResultRow> process() throws Exception {
    FileChannel channel = new RandomAccessFile(FILE, "r").getChannel()
    long size = channel.size()
    MemorySegment segment = channel.map(READ_ONLY, 0, size, Arena.global())

    int nThreads = 24
    var workerResults = new HashMap/*<String, Row>*/[nThreads]
    for (int i = 0; i < nThreads; i++)
        workerResults[i] = processForChunk(segment, i)

    return mergeWorkerResults(workerResults)
}
```

Параллельность - реализация

```
Map<String, ResultRow> process() throws Exception {
    FileChannel channel = new RandomAccessFile(FILE, "r").getChannel()
    long size = channel.size()
    MemorySegment segment = channel.map(READ_ONLY, 0, size, Arena.global())

    int nThreads = 24
    var workerResults = new HashMap/*<String, Row>*/[nThreads]
    for (int i = 0; i < nThreads; i++)
        workerResults[i] = processForChunk(segment, i)

    return mergeWorkerResults(workerResults)
}
```

Параллельность - реализация

```
Map<String, ResultRow> process() throws Exception {
    FileChannel channel = new RandomAccessFile(FILE, "r").getChannel()
    long size = channel.size()
    MemorySegment segment = channel.map(READ_ONLY, 0, size, Arena.global())

    int nThreads = 24
    var workerResults = new HashMap/*<String, Row>*/[nThreads]
    for (int i = 0; i < nThreads; i++)
        workerResults[i] = processForChunk(segment, i)

    return mergeWorkerResults(workerResults)
}
```

Параллельность - реализация

```
Map<String, ResultRow> process() throws Exception {
    FileChannel channel = new RandomAccessFile(FILE, "r").getChannel()
    long size = channel.size()
    MemorySegment segment = channel.map(READ_ONLY, 0, size, Arena.global())

    int nThreads = 24
    var workerResults = new HashMap/*<String, Row>*/[nThreads]
    for (int i = 0; i < nThreads; i++)
        workerResults[i] = processForChunk(segment, i)

    return mergeWorkerResults(workerResults)
}
```

Параллельность - реализация

```
Map<String, ResultRow> process() throws Exception {
    FileChannel channel = new RandomAccessFile(FILE, "r").getChannel()
    long size = channel.size()
    MemorySegment segment = channel.map(READ_ONLY, 0, size, Arena.global())

    int nThreads = 24
    var workerResults = new HashMap/*<String, Row>*/[nThreads]
    for (int i = 0; i < nThreads; i++)
        workerResults[i] = processForChunk(segment,  i)

    return mergeWorkerResults(workerResults)
}
```

Параллельность - реализация

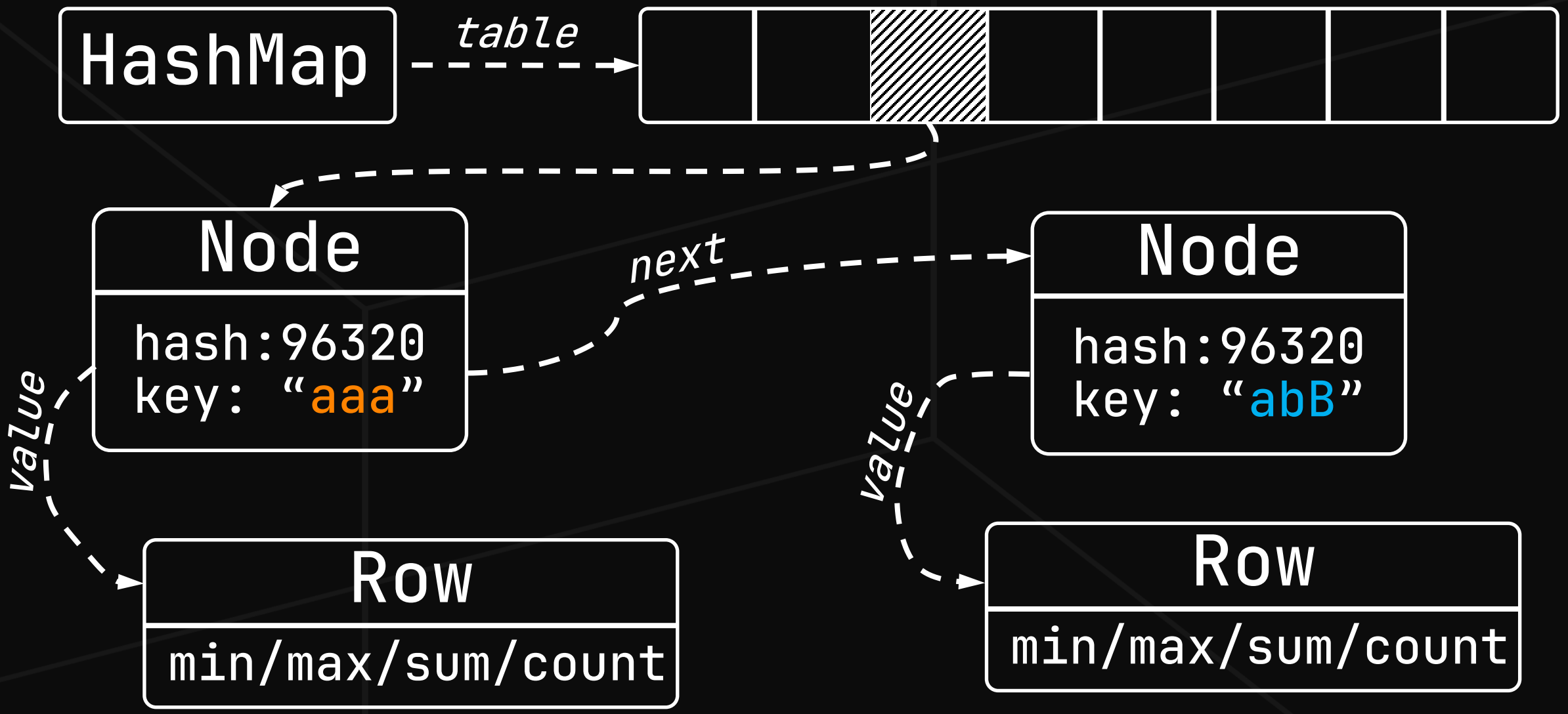
```
Map<String, ResultRow> process() throws Exception {
    FileChannel channel = new RandomAccessFile(FILE, "r").getChannel()
    long size = channel.size()
    MemorySegment segment = channel.map(READ_ONLY, 0, size, Arena.global())

    int nThreads = 24
    var workerResults = new HashMap/*<String, Row>*/[nThreads]
    for (int i = 0; i < nThreads; i++)
        workerResults[i] = processForChunk(segment,  i)

    return mergeWorkerResults(workerResults)
}
```

~10 секунд

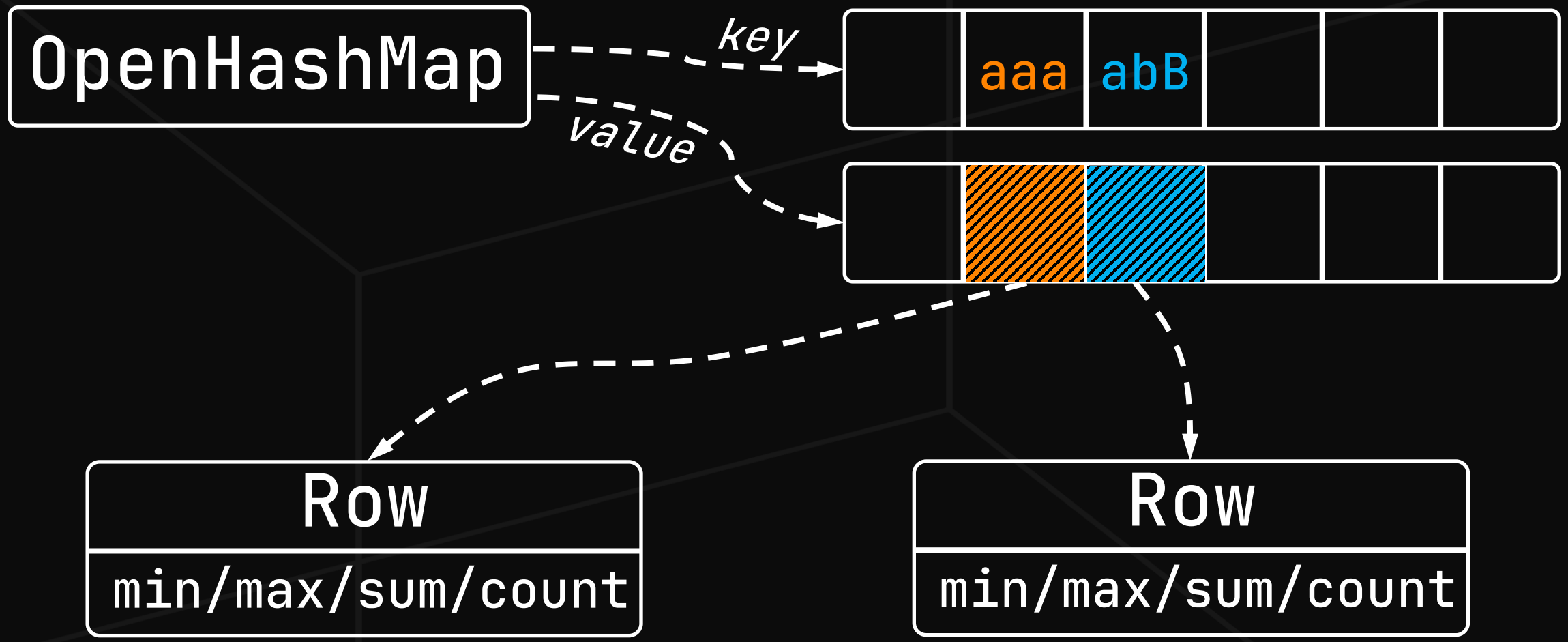
Выбор
оптимальной структуры данных

- Каждый поток обрабатывает свой кусок файла и складывает в свою локальную хэш-мапу
- В хэш-мапе: город → кортеж (min, max, sum, count)
- В конце, делаем агрегацию по всем хэш-мапам (для каждого города)
- Как обрабатывать коллизии?

Метод цепочек



Открытая адресация



Идеальное хэширование

- Подбор для заранее известного набора ключей такой хэш-функции, которая гарантирует отсутствие коллизий

```
(String cityName) -> Math.abs(cityName.hashCode()) % 10819
```

- Нужно знать заранее набор ключей, их должно быть немного
- Было запрещено в рамках конкурса

OpenFlatHashMap

Отдельная запись – 128 байт

имя города: 100 байт
хэш имени: 8 байт
длина имени: 4 байта
min / max: по 2 байта
sum: 8 байт
count: 4 байта

OpenFlatHashMap

- Используем один массив байтов большого размера
- Каждый 128 байт – это отдельная запись
- При обращении к полям записей используем индекс, умноженный на 128 байт + смещение
- Минимальное использование косвенной адресации
- Надо писать руками, но ждем Project Valhalla
- Как замерить? Например: *perf stat*

Work stealing

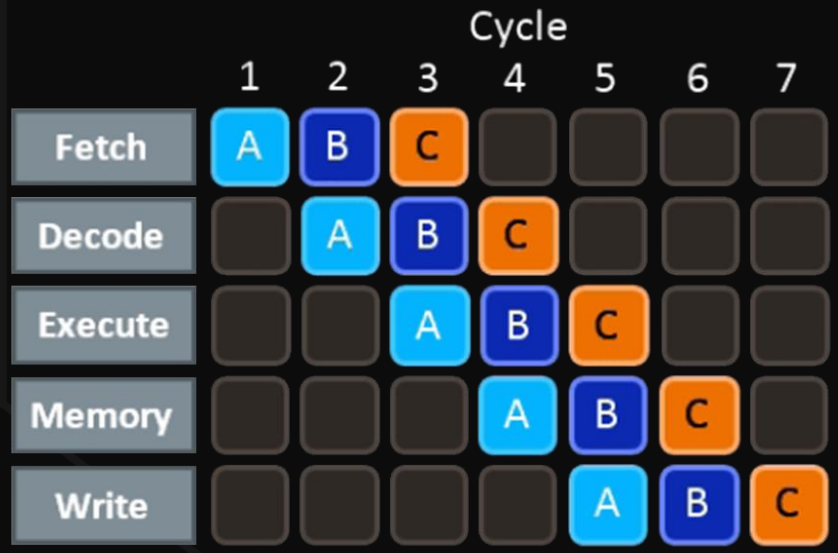
- **Изначально делили файл на 24 куска**
  8P + 8E = 24 потока исполнения
  P core: производительные
  E core: энергоэффективные

- **Не гибридная архитектура?**
  производительность «одинаковых» ядер разная

- **Надо резать на большее количество задач**
  слишком маленький размер задачи – большой overhead
  слишком большой размер задачи – какие-то ядра простаивают
  нужно подбирать экспериментально

Instruction-Level Parallelism

- Модель мышления: процессор выполняет команды последовательно, одна за другой
- Реальность: команды разбиваются на раздельные стадии выполнения
- Стадии выполняются параллельно
- Параллелизм на уровне отдельных команд



Привязка к формату данных

```
long parseDataPoint(LongConsumer consumer, MemorySegment seg, long offset) {
    long word = seg.get(JAVA_LONG_LT, offset)
    int decimalSepPos = Long.numberOfTrailingZeros(~word & 0x10101000)
    int shift = 28 - decimalSepPos
    long signed = (~word << 59) >> 63
    long designMask = ~(signed & 0xFF)
    long digits = ((word & designMask) << shift) & 0x0F000F0F00L
    long absValue = ((digits * 0x640a0001) >>> 32) & 0x3FF
    long value = (absValue ^ signed) - signed
    consumer.accept(value)
    return offset + (decimalSepPos >>> 3) + 3
}
```

Привязка к формату данных

```
long parseDataPoint(LongConsumer consumer, MemorySegment seg, long offset) {
    long word = seg.get(JAVA_LONG_LT, offset)
    int decimalSepPos = Long.numberOfTrailingZeros(~word & 0x10101000)
    int shift = 28 - decimalSepPos
    long signed = (~word << 59) >> 63
    long designMask = ~(signed & 0xFF)
    long digits = ((word & designMask) << shift) & 0x0F000F0F00L
    long absValue = ((digits * 0x640a0001) >>> 32) & 0x3FF
    long value = (absValue ^ signed) - signed
    consumer.accept(value)
    return offset + (decimalSepPos >>> 3) + 3
}
```

Подробнее:

questdb.io/blog/1brc-merykittys-magic-swar

Unsafe

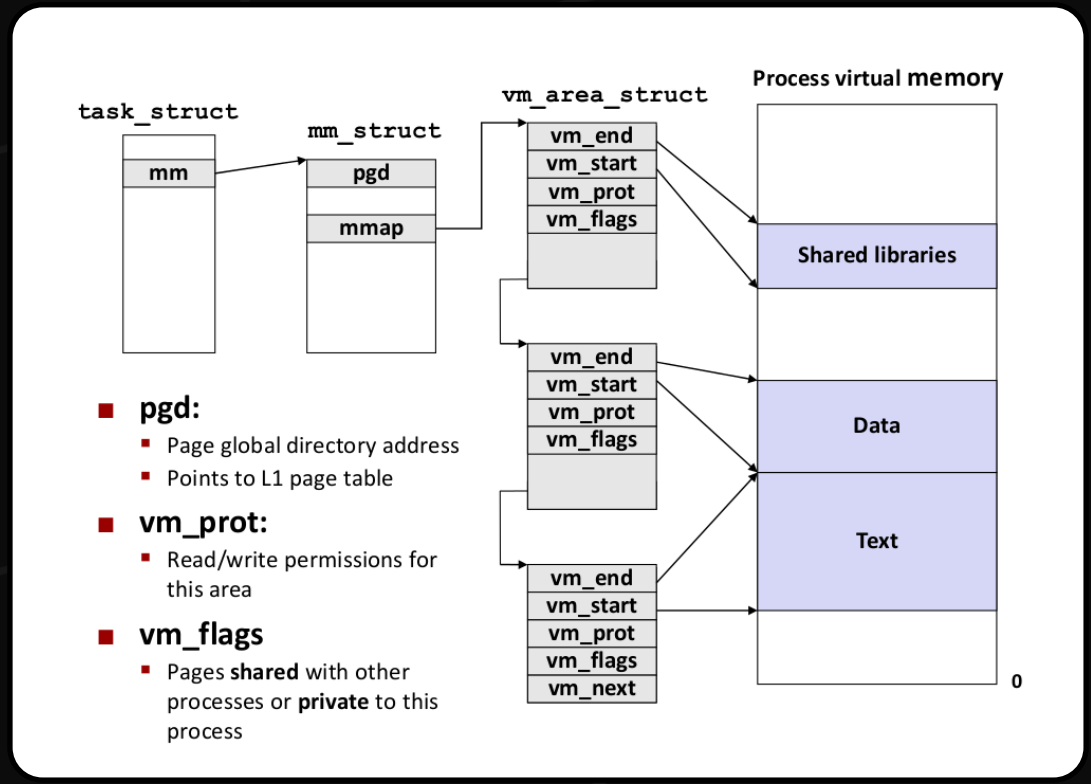
- Нужно очень хорошо подумать, стоит ли оно того
- Доступно в некоторых библиотеках как готовый компонент с интерфейсом

  например, `UnsafeBuffer` из *agrona*
- JEP-471: Deprecate the Memory-Access Methods in sun.misc.Unsafe for Removal

  замена – `VarHandle` и `MemorySegment`

Особенности mmap/munmap

- **Пользуемся** `mmap`
  внутри ядра обновляются соответствующие структуры данных для хранения информации об адресном пространстве процесса
- **При завершении процесса** – тратим время на их очистку
- **В 1BRC:** костыль с запуском нового процесса или вызов `cleaner` на `ByteBuffer`
- **Альтернативный вариант:** `madvice + MADV_DONTNEED`



Волшебный madvice

- Хинт для ОС относительно используемой памяти:
  `MADV_NORMAL , MADV_RANDOM, MADV_SEQUENTIAL, MADV_WILLNEED, MADV_DONTNEED`

  \+ специфичные для Linux флаги

- Использование в реальных проектах:

  Lucene
  github.com/apache/lucene/pull/13196

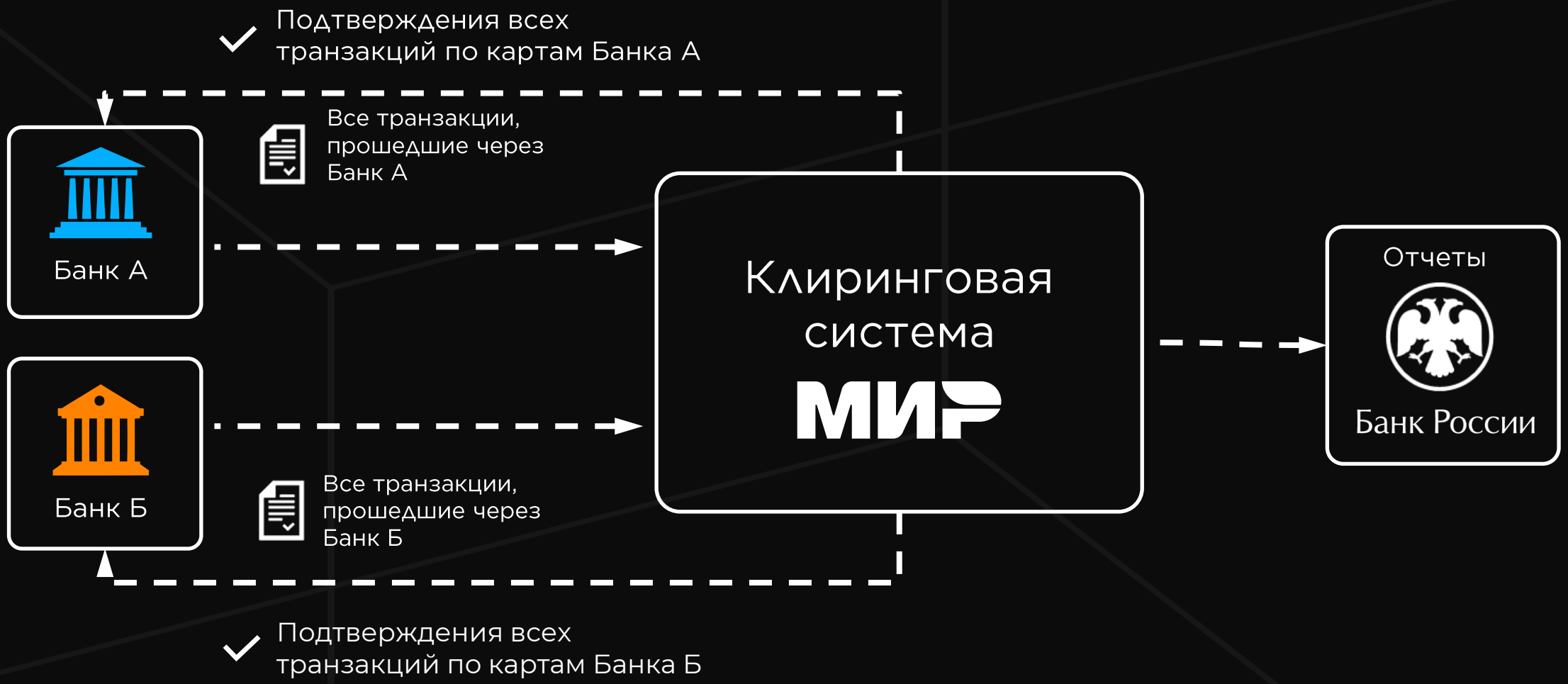
  Camunda
  github.com/camunda/camunda/issues/11377

Lucene

Camunda

В какой момент
надо было остановиться?

Пример похожей задачи



Пример похожей задачи

МИР current: 

МИР candidate: 

Нужно проверить, что сгенерированные файлы логически эквивалентны

Занимало более 8 часов на больших объемах, после оптимизаций ~ 30 минут

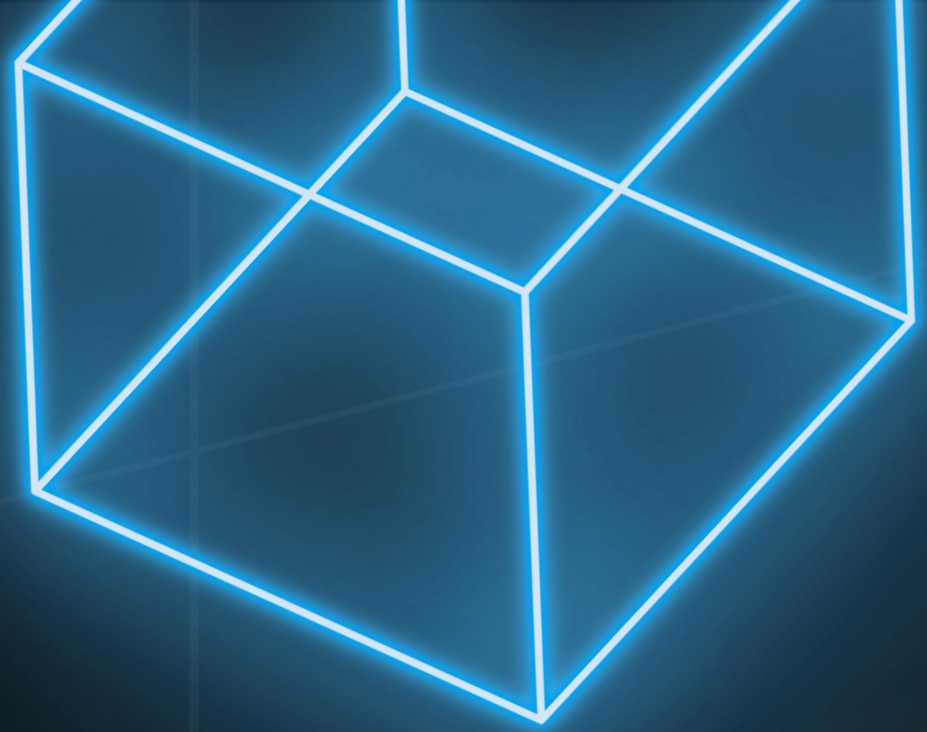
Обработка отдельного файла

- Первый файл разбивается на куски
- Куски файла обрабатываются отдельными потоками, транзакции складываются в локальные хэш-таблицы
- Хэш-таблицы объединяются, уникальные транзакции можно обрабатывать параллельно (95% от всех)
- Уникальные - в lock-free open hash map
- Обрабатываем второй файл, разбивая на куски, проверяем логическую эквивалентность транзакций из второй файла и хэш-мапы, полученной из первого файла
- Не уникальные проверяем последовательно (но их мало)

Итоги

- Посмотрели на разные подходы к оптимизации производительности
- Специализированный код ценой меньшей гибкости становится быстрее
- Проблемы с производительностью должны решаться по мере их важности
- Пользуйтесь инструментами!

МИР Plat.Form



Спасибо

за внимание!