Offline: Новосибирск

Иван Тимофеев | Старший разработчик

Как мы GC укрощали

Иван Тимофеев

Старший разработчик

Telegram @writelogin

- 4 года в ИТ
- 1 год в Т
- Команда программ лояльности
- Строю дома сеть мечты из микротиков

План повествования

01 Почему мы вообще занялись этим?

02 Какие варианты рассматривались?

03 Немного теории

04 Проведение исследования

05 Анализ полученных результатов

Есть сервис отображения баллов

Необходимо обеспечить 12К rps

Сервис высоконагруженный, используются REST ручки и Virtual threads

Есть требование по Latency

Не выше 0.35 секунды по p99.
«Пики» до 5 секунд

Большие паузы

Паузы GC могут достигать 0.2-0.3 секунды

Стоп стоп, что ты сказал?

SLA, перцентили

Пицца

Мы заказываем пиццу. Что мы хотим получить как голодный заказчик?

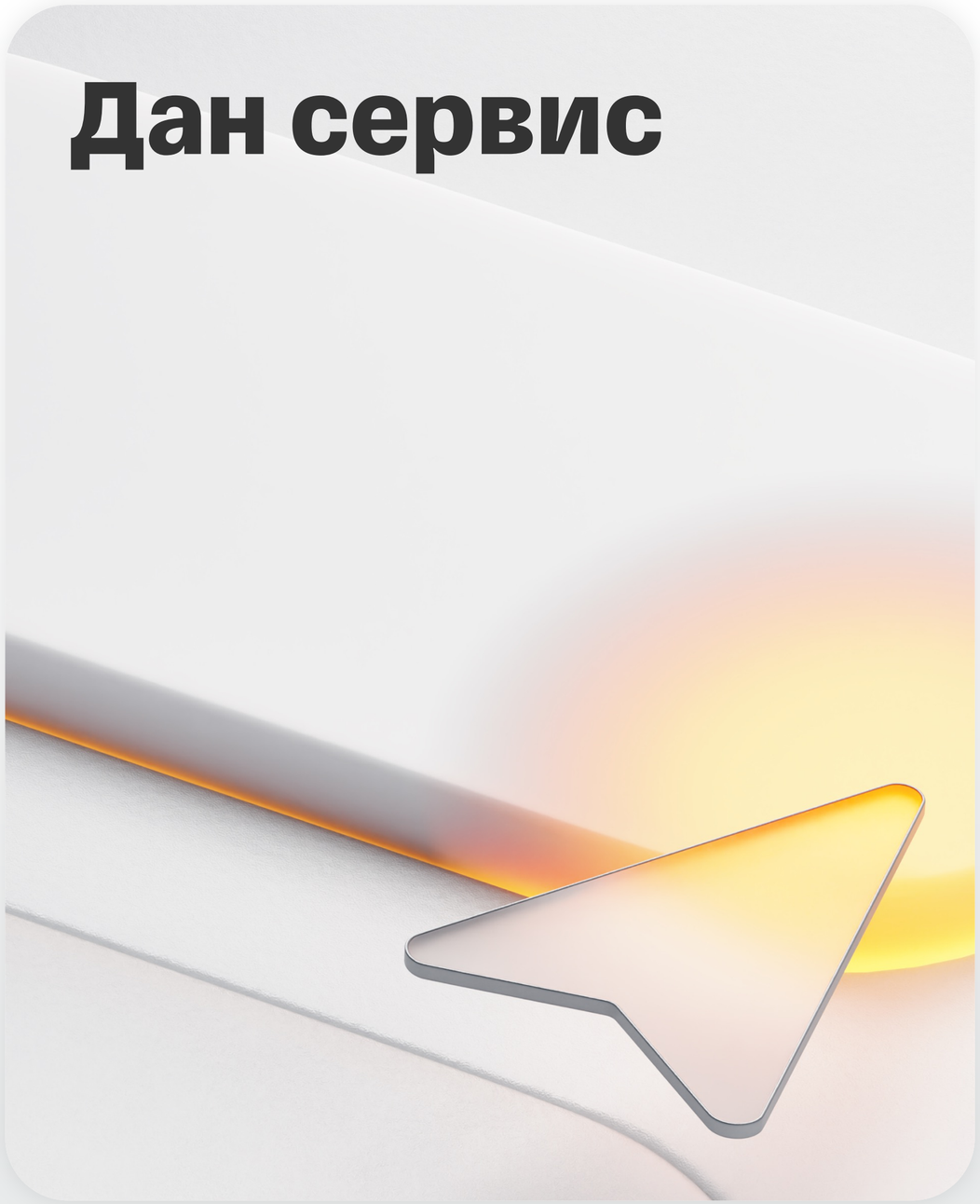
- Долго – найдем нового
- Качество каждый раз разное – найдем нового
- Если от каждого последующего заказа время ожидания растет – найдем нового
- SLA – соглашение о качестве предоставляемых услуг. Пицца не приедет за 30 минут? Будет бесплатной!

Пицца

SLA понятно, а что за p99?

- Перцентиль – это метрика «худший показатель в N процентов событий» => pN
- Их любят за стойкость к «выбросам»
- На пицце – у курьера сломалась машина и одна доставка выполнялась 5 часов! (остальные за 30 минут)
- На 100 заказов: 99 за 30 минут и 1 за 300 минут.
  Среднее: 32.7 минуты
  А p99: 30 минут!

Дан сервис



Необходимо обеспечить 12К rps

Сервис высоконагруженный, используются REST ручки и Virtual threads

Есть требование по времени ответа

Не выше 0.35 секунды по p99.

«Пики» до 5 секунд

Большие паузы

Паузы GC могут достигать 0.2-0.3 секунды

Кажется необходимо что-то менять!

* Представим мир, где сеть оптимизирована максимально, запросы в бд идеальны и с предсказуемым временем ответа...

Bending pause times to your will with Generational ZGC

https://netflixtechblog.com/bending-pause-times-to-your-will-with-generational-zgc-256629c9386b

Что отметили коллеги?

Меньше паузы

Длительность пауз 0.001 секунда

Ровнее график

«Плавнее»

Меньше Latency

По p99

Из коробки!

0. Epsilon

Epsilon GC

- Не собирает мусор. Совсем. Даже чуть-чуть
- Можно замерять эталонное быстродействие приложения
- Иногда реально используется – например терминальная утилита
- Если мы владеем безграничным количеством памяти

1. Serial GC

Serial GC

- Сборка в одном потоке и только при полной остановке приложения
- Удобен если ресурсов совсем мало
- Достаточно простой и предсказуемый
- В продакшене не используется

2. Parallel GC

Parallel GC

- Как и на картинке – схож с Serial GC, но умеет в несколько потоков
- Ранее был по умолчанию
- Развитие разработки потребовало более оптимальных алгоритмов
- Пока не работает – приложение может быть даже быстрее ☺


G1
ZGC

3. G1 GC

G1 GC

- По умолчанию сейчас
- Хорошая пропускная способность
- Гибко тюнингуется, поддерживает большие кучи
- Физически делит поколения на регионы и уже их чистит

4. ZGC

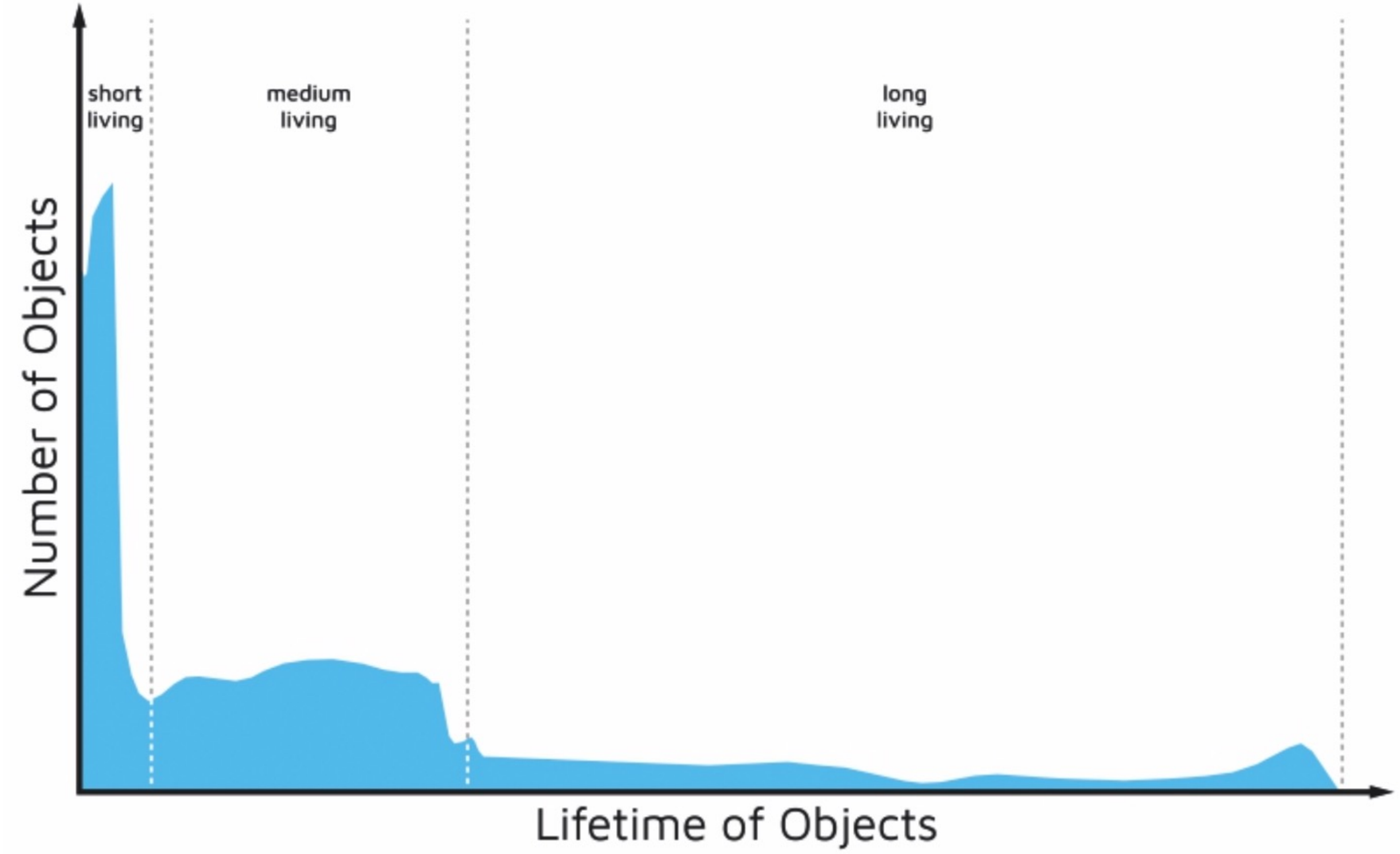
Что ты такое?

- ZGC – Z Garbage Collector (именно Z, не Zettabyte!)
- Почти все процессы жизненного цикла в многопотоке. STW на Mark Start, Mark End, Relocate Start.
- Также чистит кучу за счет вашего приложения
- В JDK 25 поколенческий режим стал «по умолчанию» и начали deprecate non-generational режима
- В идеале разработчики не будут «тюнить» GC
- Время пауз 0.001 на кучах до 16 ТБ

STW?

Поколения

Weak Generational Hypothesis



Weak Generational Hypothesis

Молодые вперед

Большинство объектов умирают молодыми

Старые реже

Чем старше объект, тем меньше шанс, что он умрет

Почему бы не чистить по-разному?

Разный подход к каждому поколению

А теперь вернемся к тесту

Окружение

- **Железо:** M3 Pro, 36 RAM
- **Способ развертывания:** 2 Docker контейнера с ограничением по ресурсам:
  --cpus=”2" -m "2g"
- **Стэк:** Java 21, Spring Boot 3.3.6, Spring Data JDBC
- **Инструмент нагрузочного теста:** Grafana K6
- **Сценарий:**
  Прогрев – 15 минут на 100 RPS
  Разгон – 10 минут 100 –> 300
  Основная часть – 30 минут 300 RPS

На чем проводим тест?

Первый подопытный – сервис с синхронными ручками

- Сервис с большим числом REST – ручек – основная бизнес-цель
- Множество походов в БД, но все уже оптимизированы
- Имеет «ламинарные» показатели по RPS
- В рамках тестирования будем осознано «перегружать» сервис

RPS

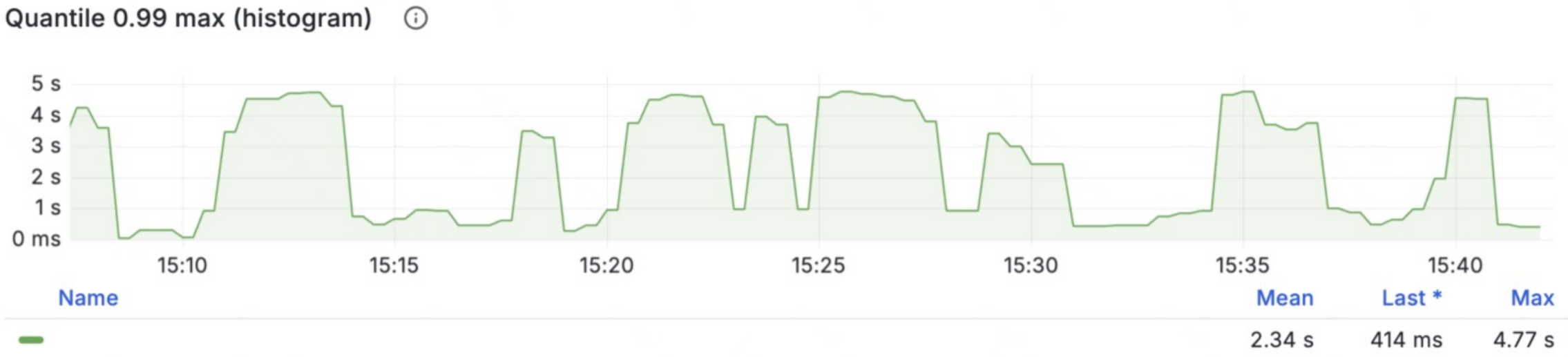
G1


RPS per uri
300
250
200
150
15:10
15:15
15:20
15:25
15:30
15:35
15:40
320
300
280
260
240
220
18:45
18:50
18:55
19:00
19:05
19:10
19:15

ZGC

p99

G1

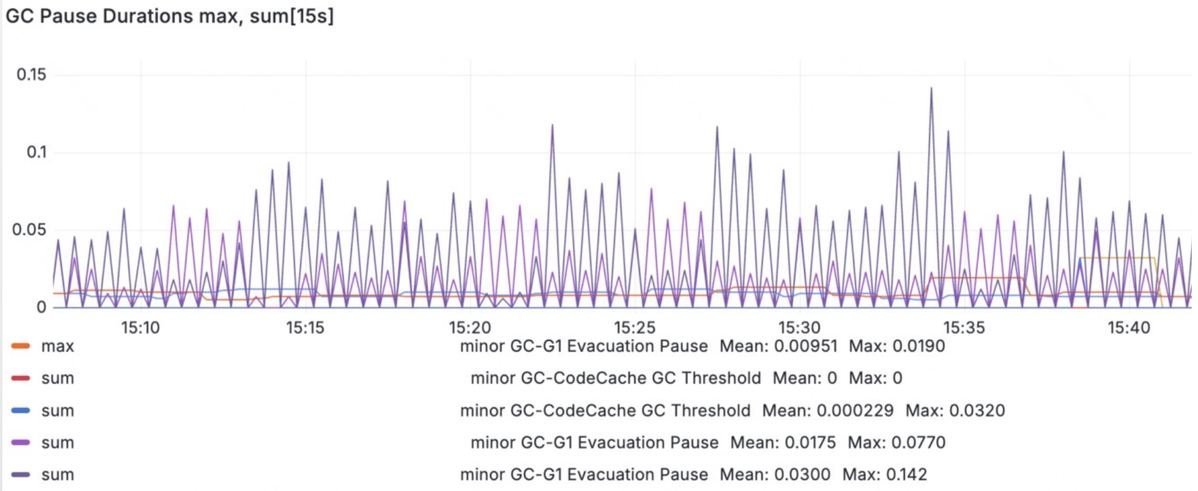

Quantile 0.99 max (histogram)
5 s
4 s
3 s
2 s
1 s
0 ms
15:10
15:15
15:20
15:25
15:30
15:35
15:40
Name
Mean
Last *
Max
2.34 s
414 ms
4.77 s

ZGC

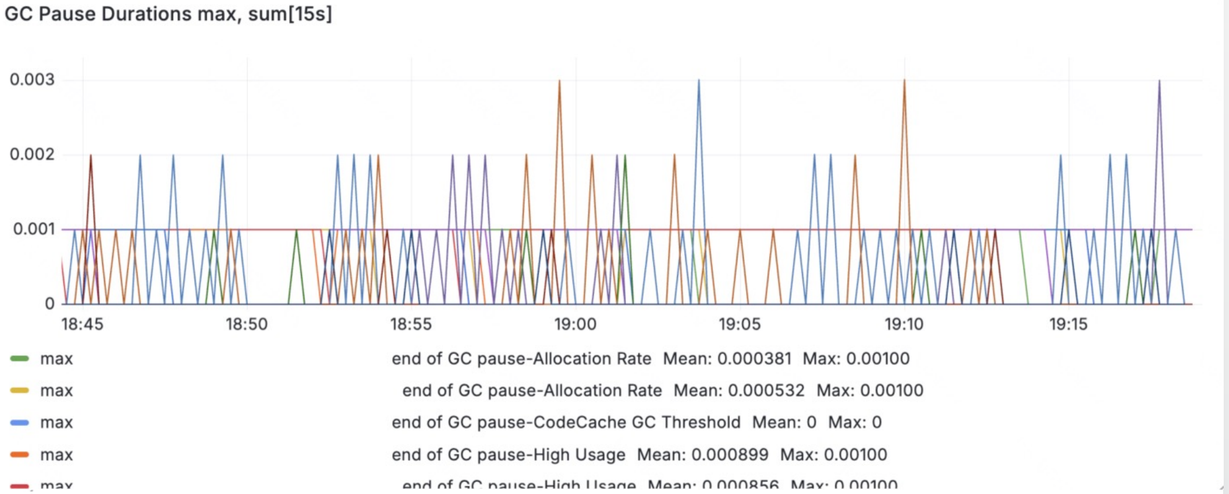

Quantile 0.99 max (histogram)
4 s
3 s
2 s
1 s
0 ms
18:45
18:50
18:55
19:00
19:05
19:10
19:15
Name
Mean
Last *
Max
926 ms
580 ms
4.09 s

Pause Duration

G1

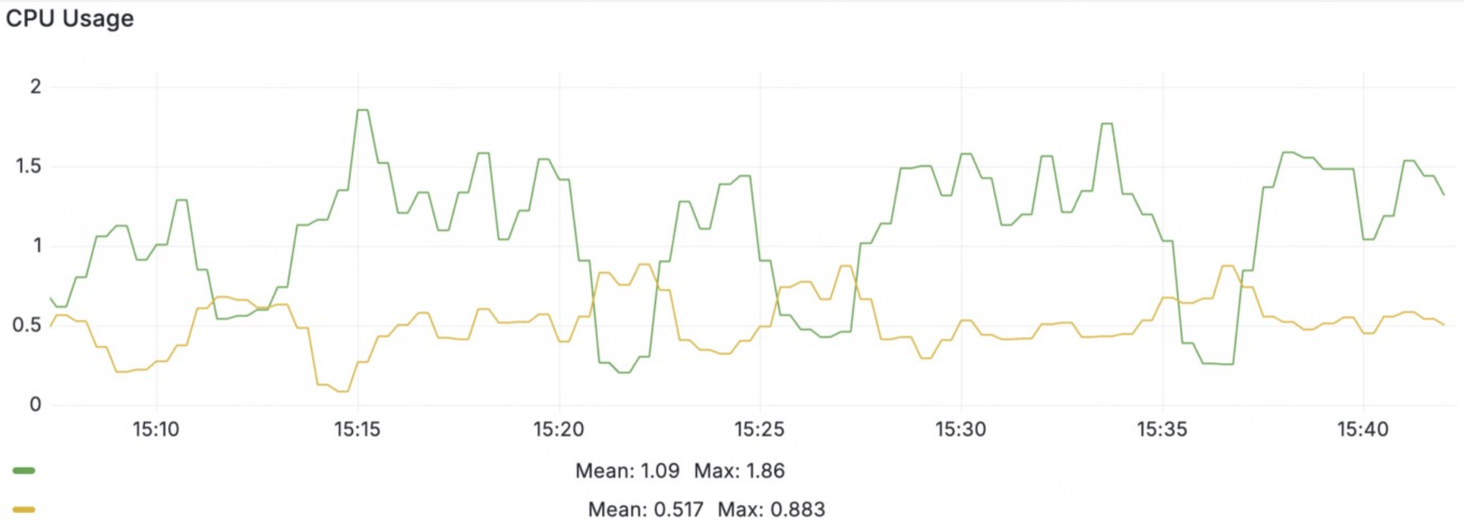


ZGC

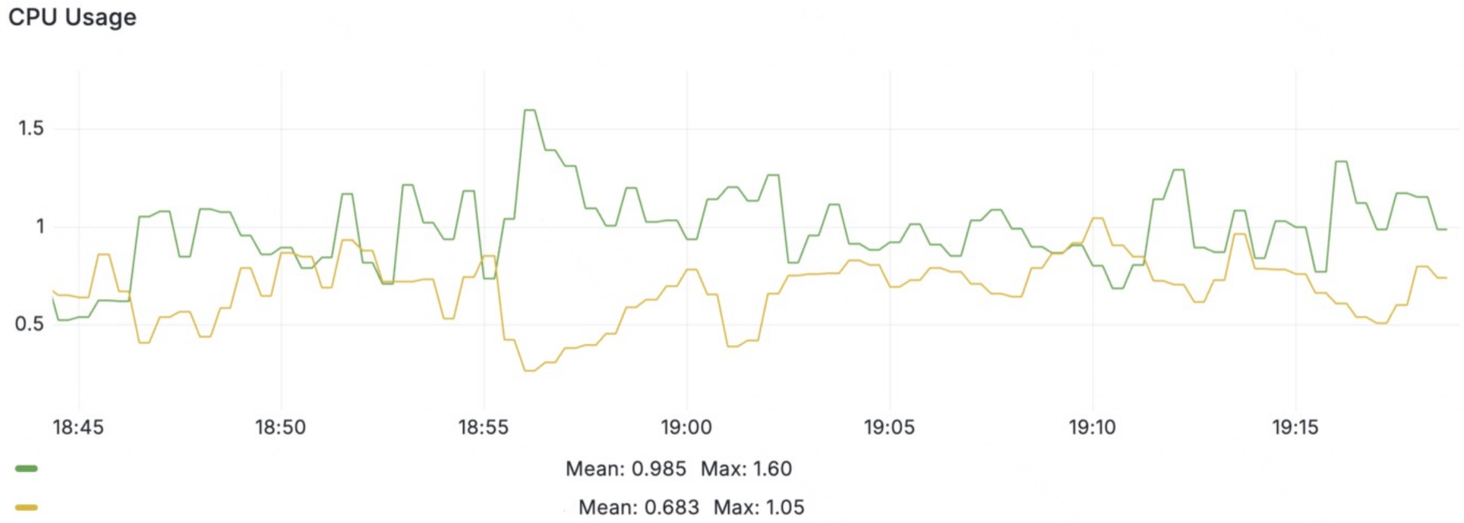


CPU Usage

G1



ZGC



Что мы увидели?

Длительность пауз

Меньше на порядок!

Количество пауз

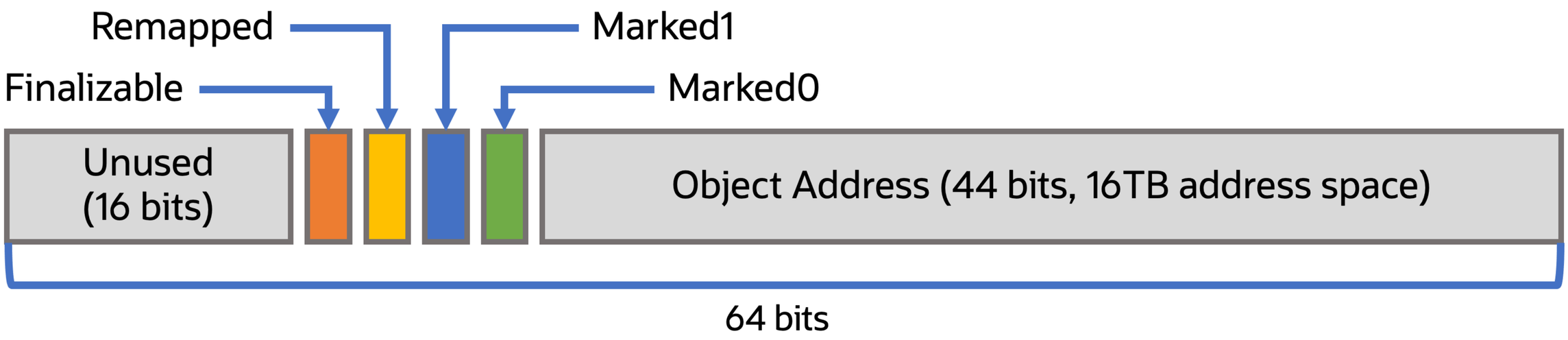
Больше на порядок

CPU Usage

Не изменился

Как так получилось?

Colored Pointers



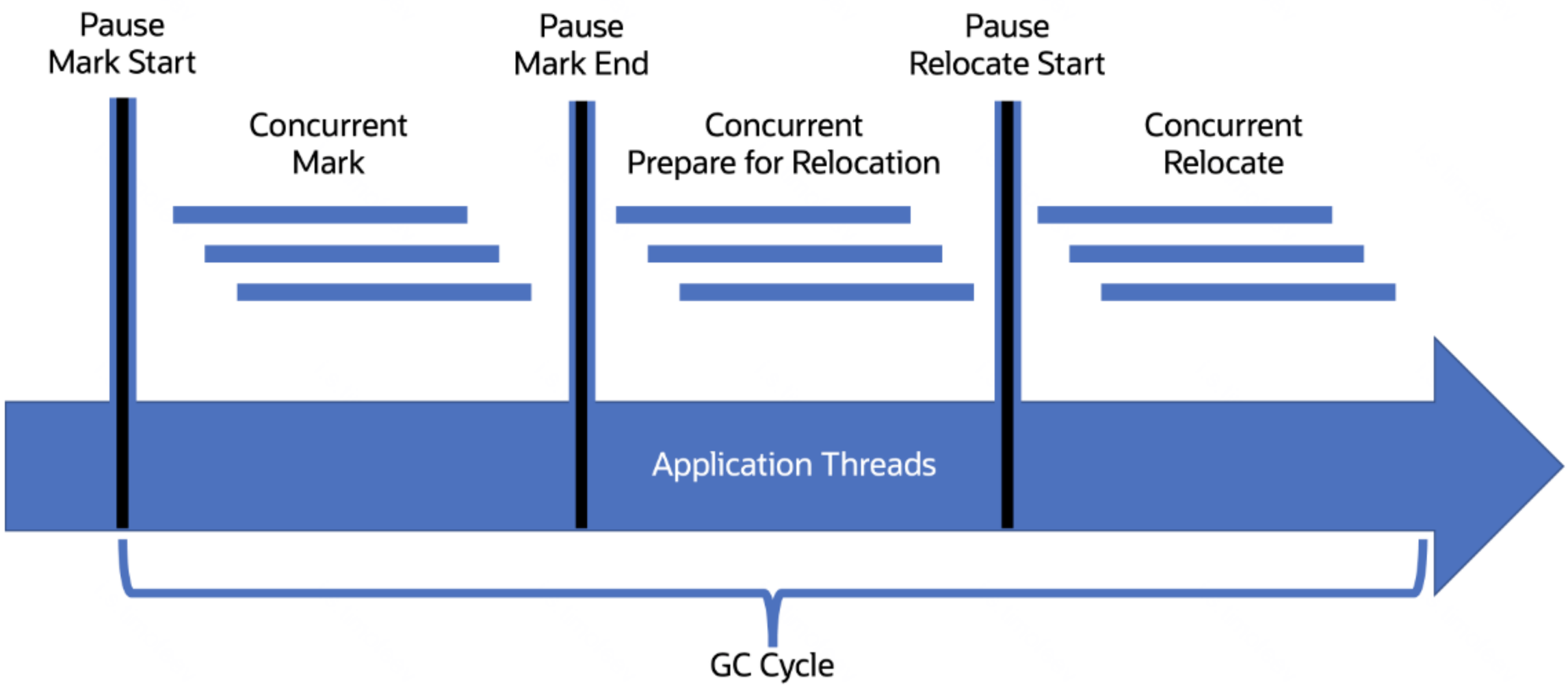
Load Barriers

- C2
- Код вокруг геттеров на ссылки из кучи

C2.cpp

```
inline zaddress ZBarrier::barrier(...) {
    if (finalizable) {
        barrier<is_marked_or_null_fast_path,
mark_barrier_on_finalizable_oop_slow_path>(p, o);
    } else {
        const uintptr_t addr = ZOop::to_address(o);
        if (ZAddress::is_good(addr)) {
            // Mark through good oop
            mark_barrier_on_oop_slow_path(addr);
        } else {
            // Mark through bad oop
            barrier<is_good_or_null_fast_path,
mark_barrier_on_oop_slow_path>(p, o);
    }
  }
}
```

STW


Pause
Mark Start
Pause
Mark End
Pause
Relocate Start
Concurrent
Mark
Concurrent
Prepare for Relocation
Concurrent
Relocate
Application Threads
GC Cycle

Что нового приносит JDK 25 LTS?

01

JDK-8350441

ZGC: Overhaul Page Allocation

02

JDK-8357443

ZGC: Optimize Old Page Iteration in Remap Remembered Phase

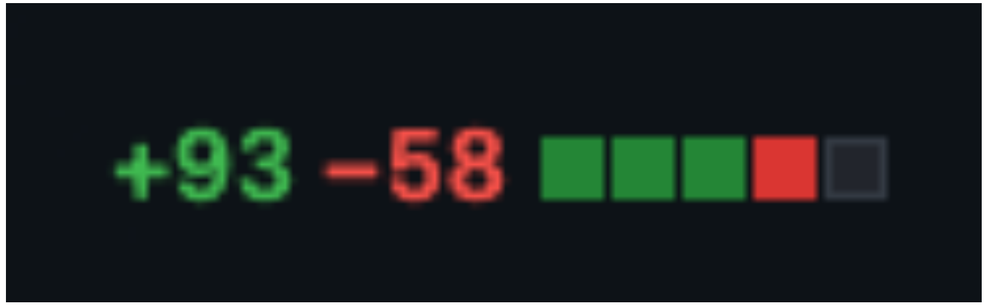
03

JEP 474

ZGC: Generational Mode by Default

JDK-8357443

- 90 строчек изменений
- Небольшая доработка метаданных
- P.S. Где-то тут я осознал теорию заговора, что нами управляют плюсовики...



JDK-8357443

- 90 строчек изменений
- Небольшая доработка метаданных
- P.S. Где-то тут я осознал теорию заговора, что нами управляют плюсовики...

zRemembered.hpp

```
// This iterator uses the "found old" optimization to skip having to
//iterate
// over the entire page table. Make sure to check where and how the
//FoundOld
// data is cycled before using this iterator.
class ZRemsetTableIterator {
private:
  ZRemembered* const            _remembered;
  BitMap* const                 _bm;
  ZPageTable* const             _page_table;
  const ZForwardingTable* const _old_forwarding_table;
  volatile BitMap::idx_t        _claimed;

public:
  ZRemsetTableIterator(ZRemembered* remembered, bool previous);

  bool next(ZRemsetTableEntry* entry_addr);
};
```

JDK-8357443

Benchmark

Document1

```
4 GB MaxHeapSize
                    Original        Patch
Default threads
mac:                0.27812         0.0507
win:                0.9485          0.10452
linux-x64:          0.53858         0.092
linux-x64 NUMA:     0.89974         0.15452
linux-aarch64:      0.32574         0.15832

4 threads
mac:                0.19112         0.04916
win:                0.83346         0.08796
linux-x64:          0.57692         0.09526
linux-x64 NUMA:     1.23684         0.17008
linux-aarch64:      0.334           0.21918

1 thread:
mac:                0.19678         0.0589
win:                1.96496         0.09928
linux-x64:          1.00788         0.1381
linux-x64 NUMA:     2.77312         0.21134
linux-aarch64:      0.63696         0.31286
```

И снова наш опыт

И снова наш опыт

95th pct, ms	99th pct, ms
98	178

95th pct, ms	99th pct, ms
137	226

И что делать?

ZGC

Нужен минимальный Latency

REST-сервисы, обрабатывающие клиентские запросы

VS

G1

Нужна высокая пропускная способность

Сервисы, обрабатывающие Batch, Kafka-консьюмеры

Можно ли подкрутить GC?

-XX:ZUncommit

Не отдаем память

-XX:ZProactive

Используем малую нагрузку для доп фаз

-XX:UseDynamicNumberOfThreads

Даем GC возможность динамически выбирать количество тредов

-XX:ZFragmentationLimit

Порог фрагментации

Можно ли подкрутить GC?

-XX:ZUncommit

Не отдаем память

-XX:ZProactive

Используем малую нагрузку для доп фаз

-XX:UseDynamicNumberOfThreads

Даем GC возможность динамически выбирать количество тредов

-XX:ZFragmentationLimit

Порог фрагментации

Можно ли подкрутить GC?

-XX:ZUncommit

Не отдаем память

-XX:ZProactive

Используем малую нагрузку для доп фаз

-XX:UseDynamicNumberOfThreads

Даем GC возможность динамически выбирать количество тредов

-XX:ZFragmentationLimit

Порог фрагментации

Можно ли подкрутить GC?

-XX:ZUncommit

Не отдаем память

-XX:ZProactive

Используем малую нагрузку для доп фаз

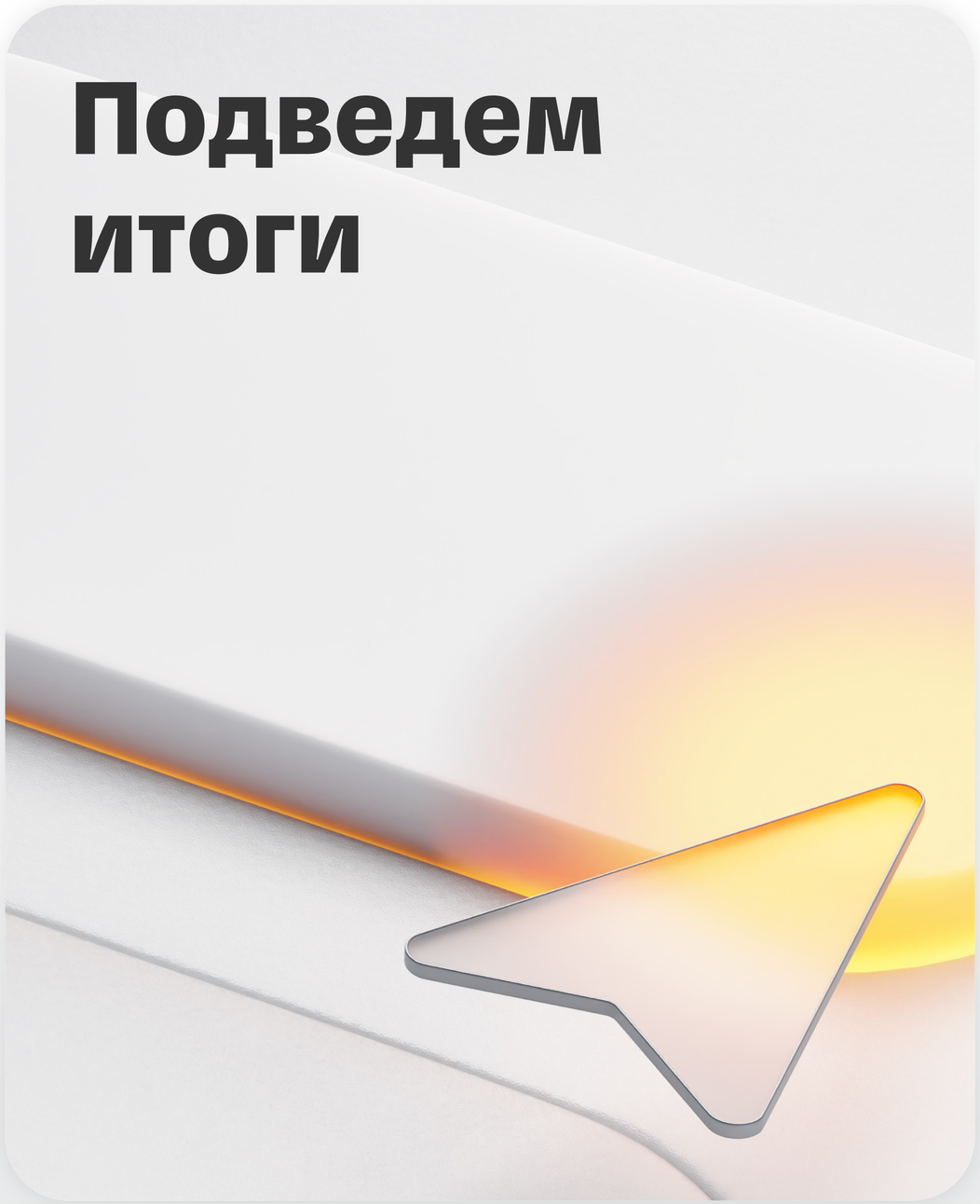
-XX:UseDynamicNumberOfThreads

Даем GC возможность динамически выбирать количество тредов

-XX:ZFragmentationLimit

Порог фрагментации

Подведем итоги



Нет идеального GC

Нет «золотой пули», способной решать всем все задачи

Поколенческая теория

Снижает нагрузку на железо. Оптимизирует очистку.

STW неизбежен

Все GC имеют разный STW, но он всегда есть

No-code
Python
JavaScript
Node.js
Swift
Mobile App
Development
Innovation
PHP
Java
Analysis
Planning

Спасибо за внимание!

Ваши вопросы