«Под капотом» изолятов и Garbage Collector Dart VM

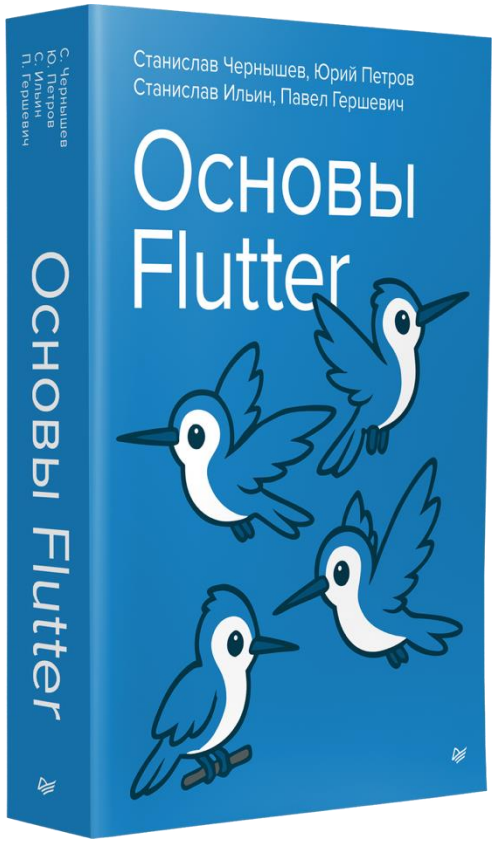
Станислав Чернышев, к.т.н., доцент ГУАП

О спикере

- **Более 10 лет работы в IT (ВПК, МО, проекты на заказ)**
- **Основной вид деятельности – наука и преподавание**
- **Ученики работают в различных компаниях**
- **Автор учебных книг**

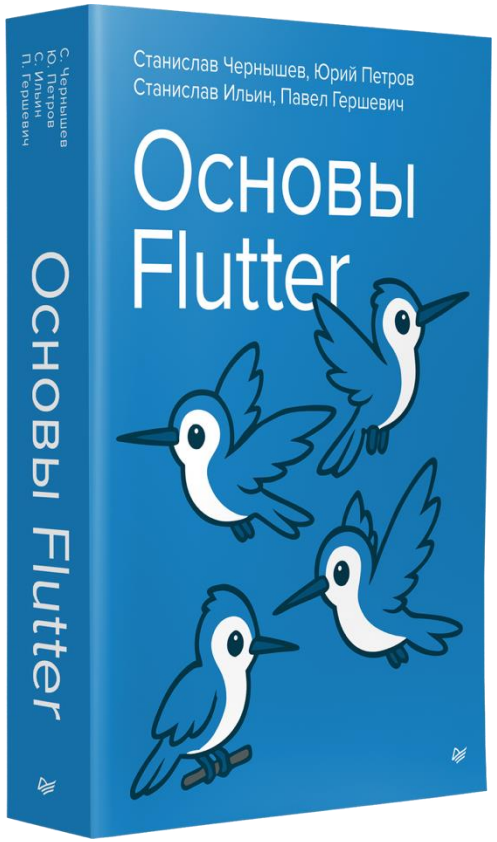
О спикере


Станислав Чернышев
ОСНОВЫ DART
Community edition
2-е издание, исправленное и дополненное
COME TO THE DART SIDE
Партнеры издания
MADBRAINS
Surf
Friflex
Использование данного файла означает, что вы согласны с условиями лицензии, расположенной на следующей странице
2026


Станислав Чернышев, Юрий Петров
Станислав Ильин, Павел Гершевич
Основы Flutter
Основы Flutter

О спикере

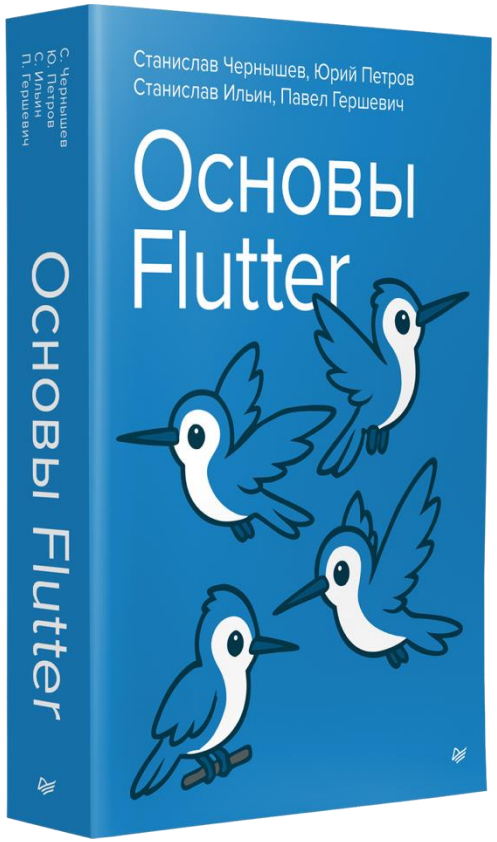

Станислав Чернышев
ОСНОВЫ DART
Community edition
2-е издание, исправленное и дополненное
COME TO THE DART SIDE
Партнеры издания
MADBRAINS
Surf
Friflex
Использование данного файла означает, что вы согласны с условиями лицензии, расположенной на следующей странице
2026


Станислав Чернышев, Юрий Петров
Станислав Ильин, Павел Гершевич
Основы Flutter
Основы Flutter


Станислав Чернышев
DART CONCURRENCY
1-е издание
COME TO THE DART SIDE
Эта книга предназначена для вашего личного пользования.
Пожалуйста, не передавайте ее третьим лицам, за исключением членов своей семьи.
2025

О спикере

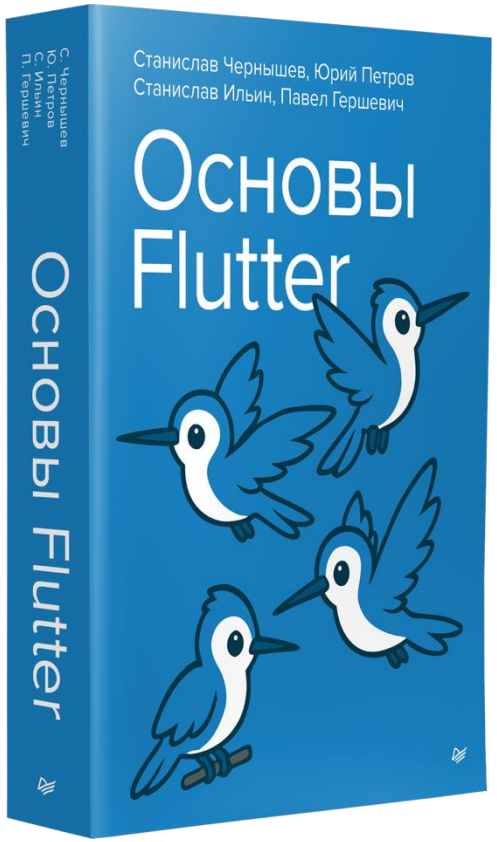

Станислав Чернышев
ОСНОВЫ DART
Community edition
2-е издание, исправленное и дополненное
COME TO THE DART SIDE
Партнеры издания
MADBRAINS
Surf
Friflex
Использование данного файла означает, что вы согласны с условиями лицензии, расположенной на следующей странице
2026


Станислав Чернышев, Юрий Петров
Станислав Ильин, Павел Гершевич
Основы Flutter
Основы Flutter


Станислав Чернышев
DART CONCURRENCY
1-е издание
COME TO THE DART SIDE
Эта книга предназначена для вашего личного пользования.
Пожалуйста, не передавайте ее третьим лицам, за исключением членов своей семьи.
2025

О спикере


Станислав Чернышев
ОСНОВЫ DART
Community edition
2-е издание, исправленное и дополненное
COME TO THE DART SIDE
Партнеры издания
MADBRAINS
Surf
Friflex
Использование данного файла означает, что вы согласны с условиями лицензии, расположенной на следующей странице
2026


Станислав Чернышев, Юрий Петров
Станислав Ильин, Павел Гершевич
Основы Flutter
Основы Flutter


Станислав Чернышев
DART CONCURRENCY
1-е издание
COME TO THE DART SIDE
Эта книга предназначена для вашего личного пользования. Пожалуйста, не передавайте ее третьим лицам, за исключением членов своей семьи.
2025


Станислав Чернышев
АЛГОРИТМЫ И СТРУКТУРЫ ДАННЫХ НА DART
1-е издание
COME TO THE DART SIDE
Эта книга предназначена для вашего личного пользования. Пожалуйста, не передавайте ее третьим лицам, за исключением членов своей семьи.
2025

За что мы любим Dart?

Манхва «Четыре рыцаря»

План, которого будем придерживаться

- **Архитектура потоков Dart VM**
- **Жизненный цикл изолята и Mutator-потока**
- **Представление объектов в памяти**
- **Сборка мусора в молодом поколении**
- **Сборка мусора в старом поколении**
- **Флаги конфигурации**

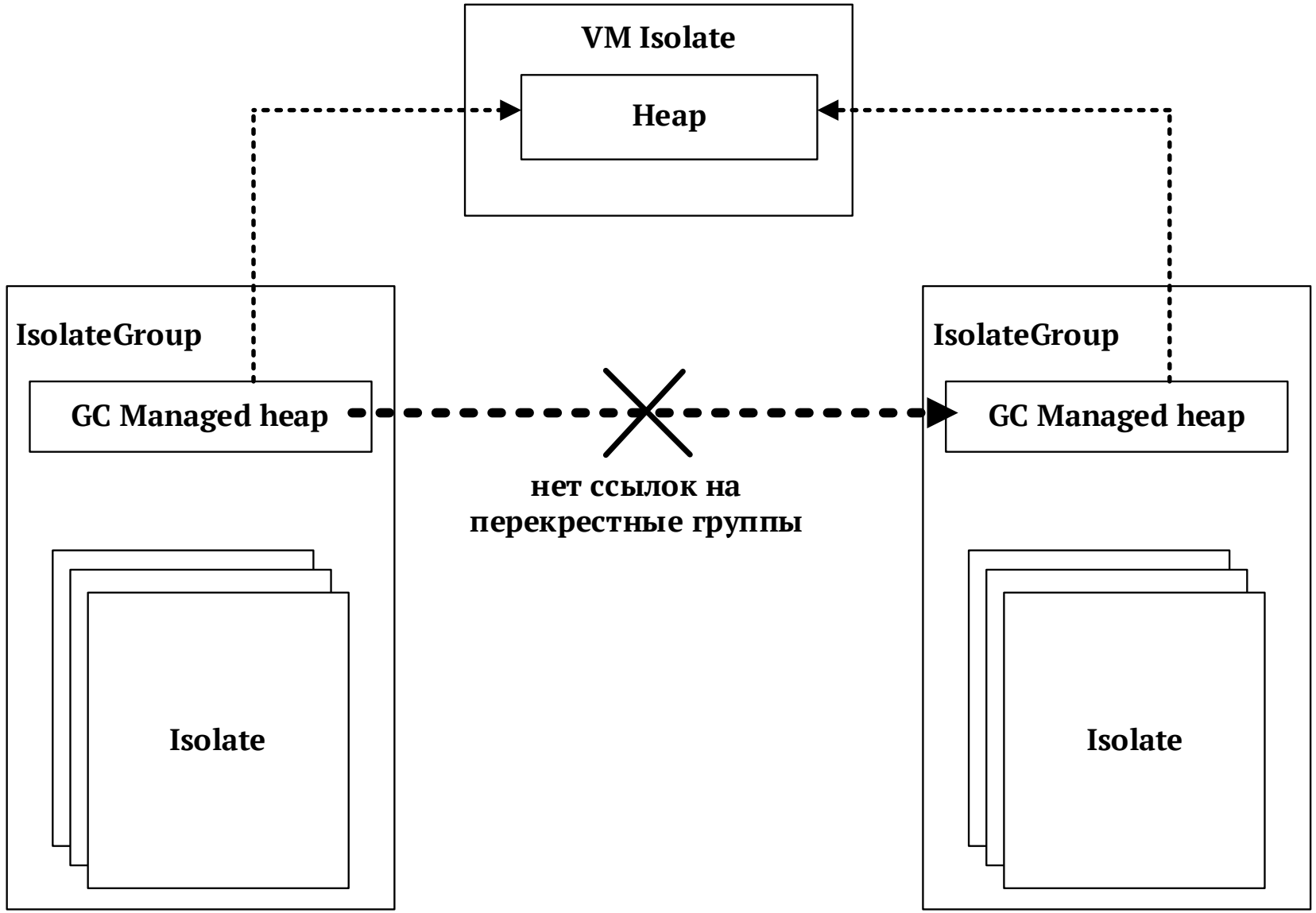

WWW

Архитектура потоков Dart VM

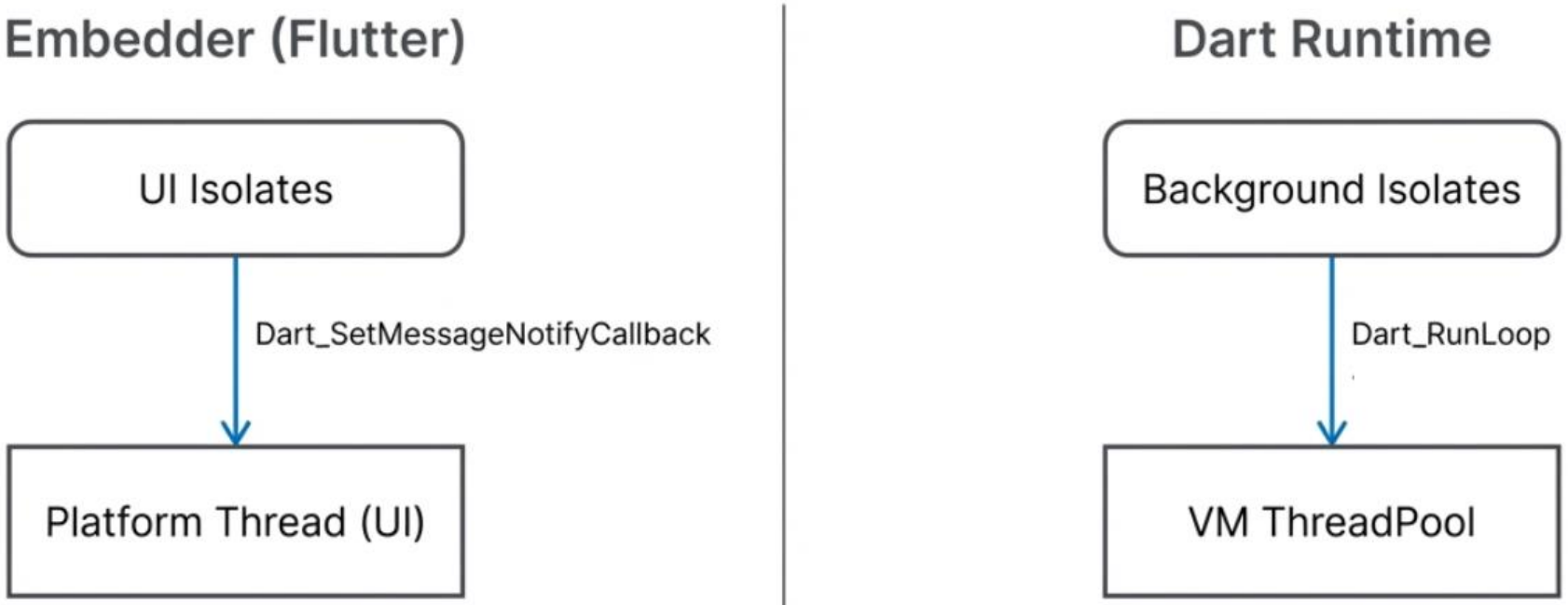
Изолят — это не всегда отдельный поток ОС

Изоляты не привязаны намертво к одному потоку.
Они исполняются в ThreadPool, благодаря чему один поток ОС может переключаться между разными изолятами.

Изоляционная группа (Isolate Group)



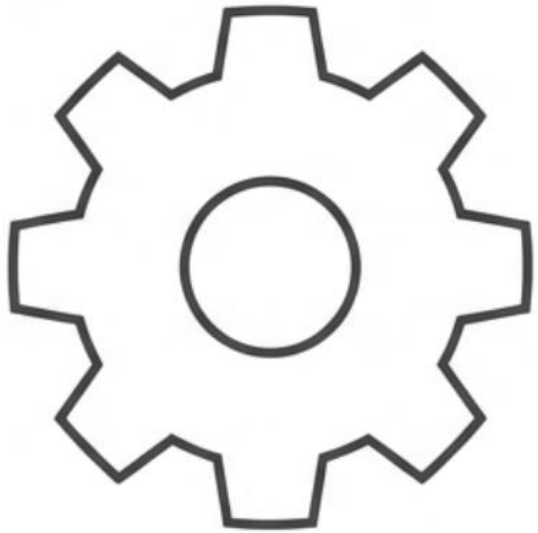
Разделение ответственности: Embedder vs Runtime



- Управляет UI-изолятами.
- Мультиплексирует их выполнение на главном платформенном потоке.

- Управляет фоновыми изолятами ('spawn').
- Исполняет их в собственном пуле потоков (ThreadPool).

За что отвечает ThreadPool?



Global VM ThreadPool

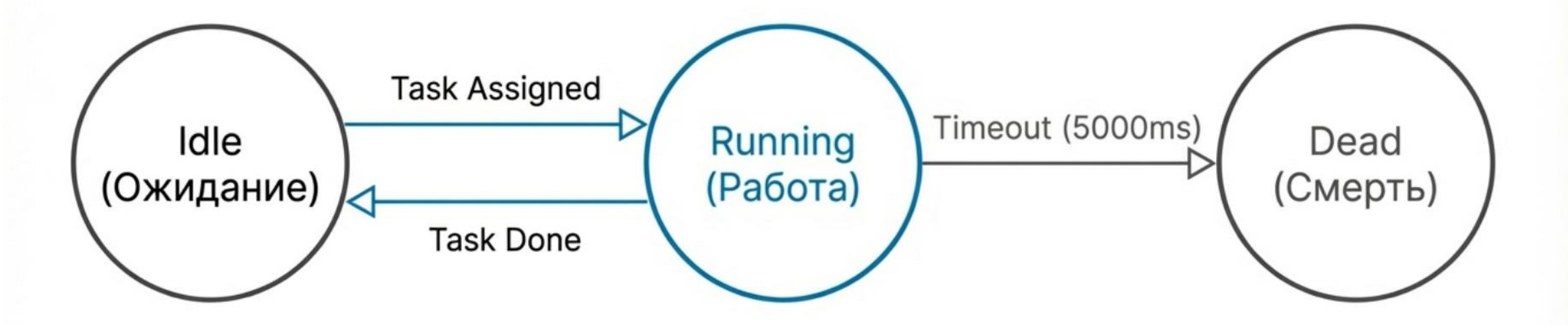
sdk/runtime/vm/thread_pool.cc

- Параллельная сборка мусора (Marker/Sweeper tasks)
- Фоновая JIT-компиляция
- Задачи Service Isolate (DevTools, отладка)
- Обработка Kernel AST

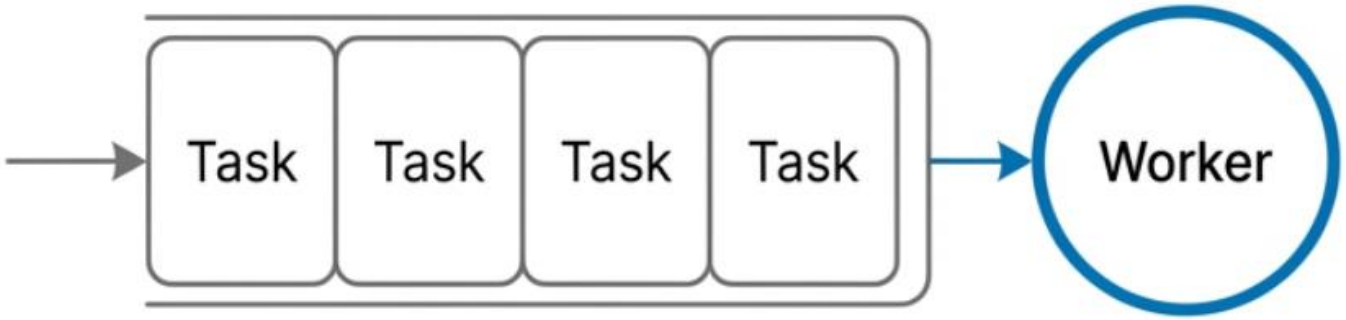
ThreadPool существует весь жизненный цикл VM и не имеет строгих ограничений на количество потоков

Класс Worker - обертка над потоком

- Связывает задачи с операционной системой.
- Один поток ОС может быть связан с разными Worker-объектами в разное время.



Класс Task

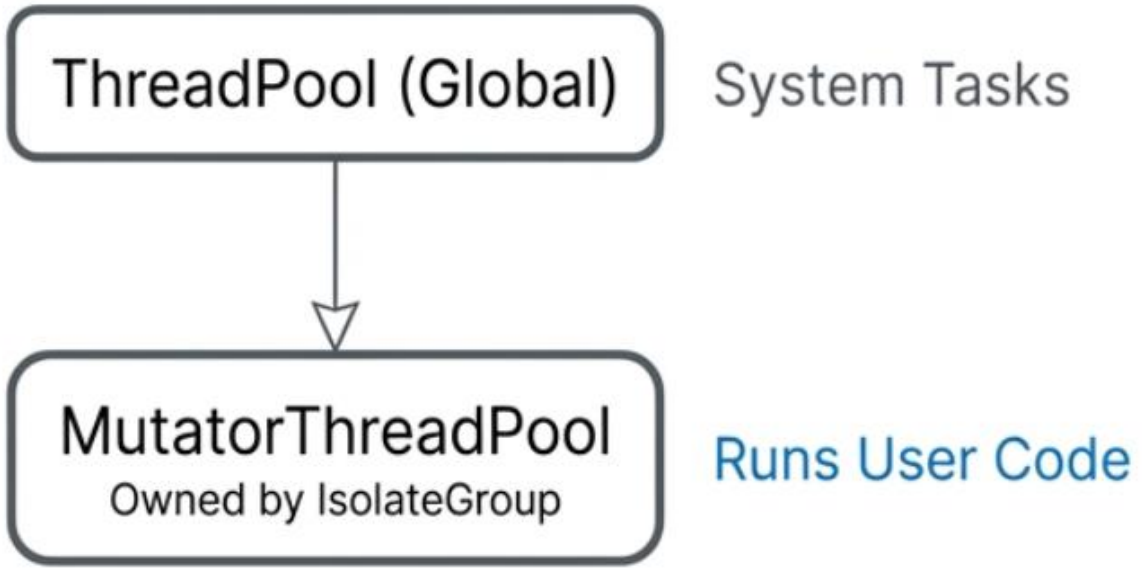


Worker берет Task из очереди и вызывает метод Run()

Типы задач:

- **Задачи управления изолятами (Isolate Management)**
- **Задачи компиляции (Compilation)**
- **Задачи сборки мусора (Garbage Collection)**
- **Задачи обработки сообщений (Message Handling)**
- **Задачи синхронизации и безопасности (Synchronization & Safety)**
- **Тестовые задачи (Testing)**

MutatorThreadPool



- Обычный ThreadPool не имеет права изменять Dart-объекты.
- MutatorThreadPool наследуется от ThreadPool и оперирует пулом потоков, которые выполняют пользовательский Dart-код.
- Имеет строгие ограничения на количество потоков для защиты производительности (TLAB).

Почему мутатор?

Мутатор (Mutator) - *поток, который имеет право изменять (мутировать) граф объектов в куче Dart*

- В каждый момент времени у одного изолята есть только один активный мутатор
- Если мутатор заблокирован нативной задачей, его статус может быть «украден»

Лимит активных мутаторов

MaxMutators = (NewGenSize / TLABSize) / 4 + 2

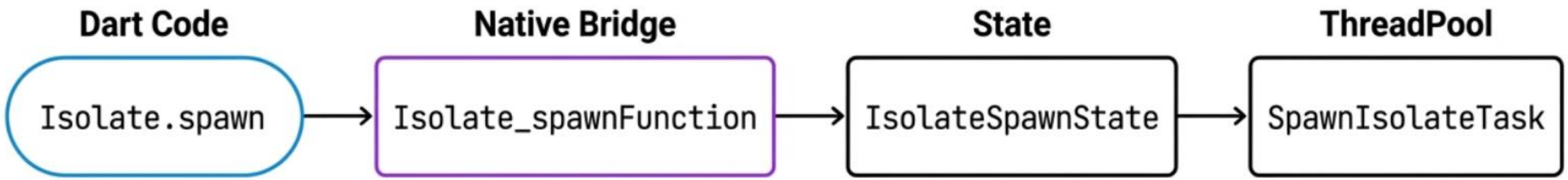
где NewGen Heap = 16 MB, а TLAB = 512 KB

(32 slots) / 4 + 2 = 10 threads

Ограничение необходимо, чтобы потоки-мутаторы не устраивали чрезмерную конкуренцию за локальные буферы аллокации (TLAB), что может приводить к деградации производительности

Жизненный цикл изолята и Mutator-потока

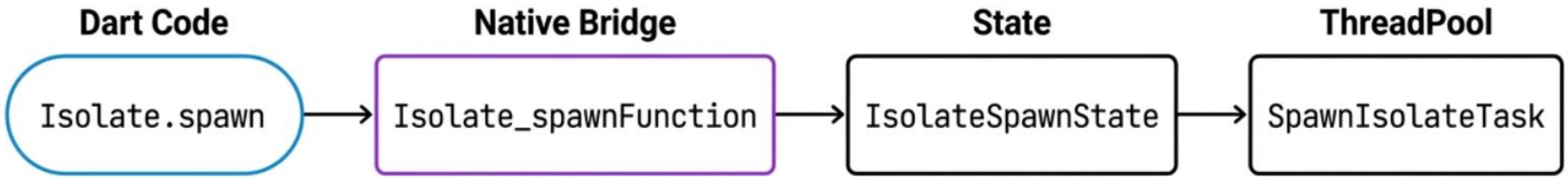
Создание изолята



// sdk/runtime/lib/isolate.cc

```
isolate->group()->thread_pool()->Run<SpawnIsolateTask>(isolate, std::move(state));
```

Создание изолята

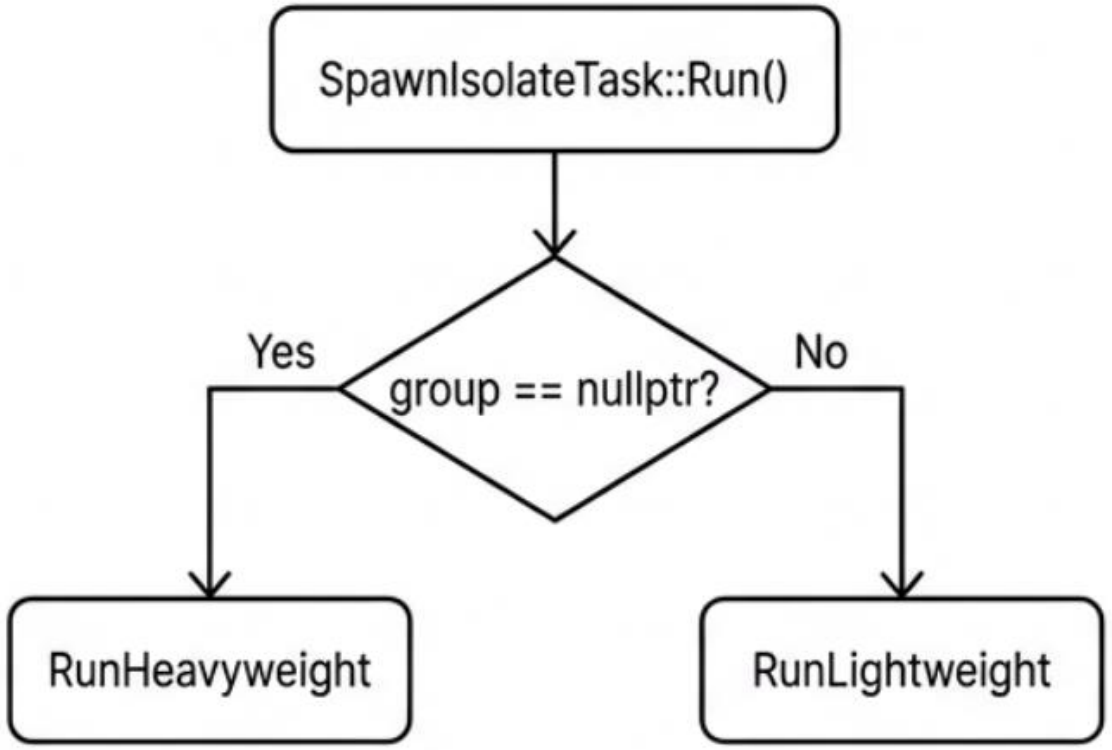


1. Native Bridge –> любой запуск (spawn, 'run, 'compute') транслируется в нативный вызов Isolate_spawnFunction

2. State Container -> создается объект IsolateSpawnState, содержащий в себе все необходимые данные для запуска.

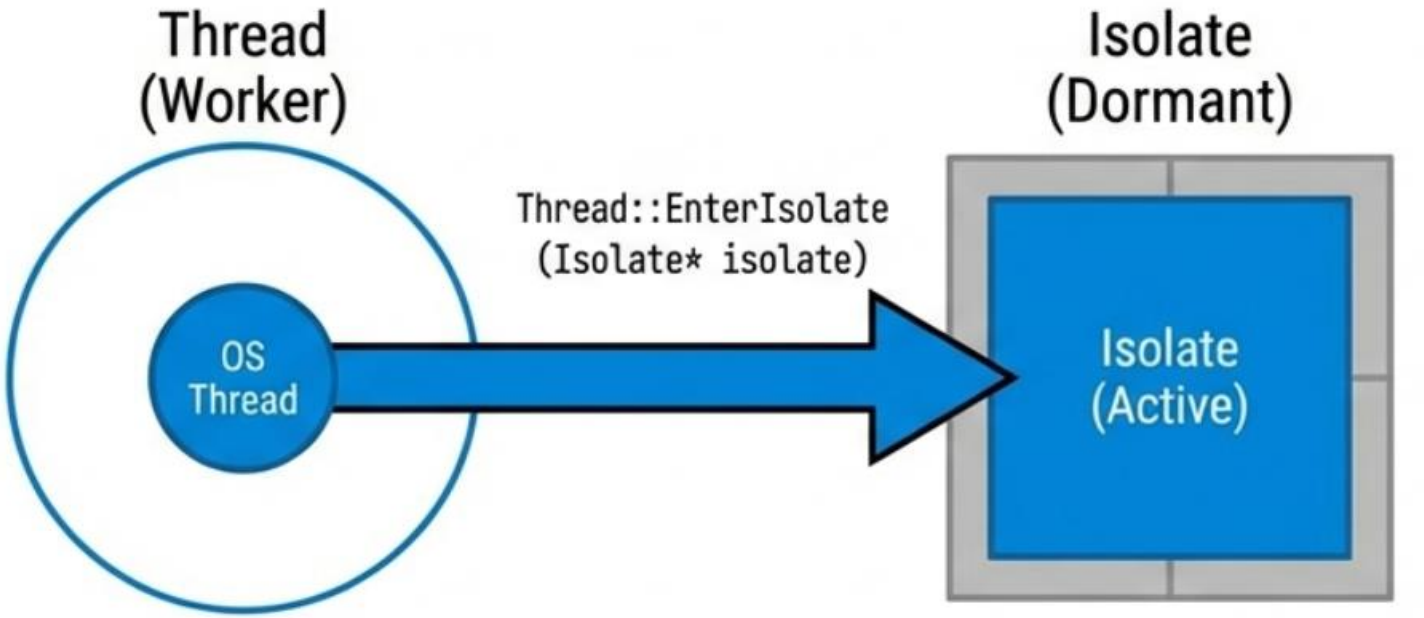
3. Task Queue -> создается задача, которая будет выполнена первым свободным воркером MutatorThreadPool

Стратегия аллокации ресурсов при запуске задачи




```
// sdk/runtime/lib/isolate.cc
if (group== nullptr) {
    RunHeavyweight(name); // Новая группа, своя куча
} else {
    RunLightweight(name); // Текущая группа, общая куча
}
```

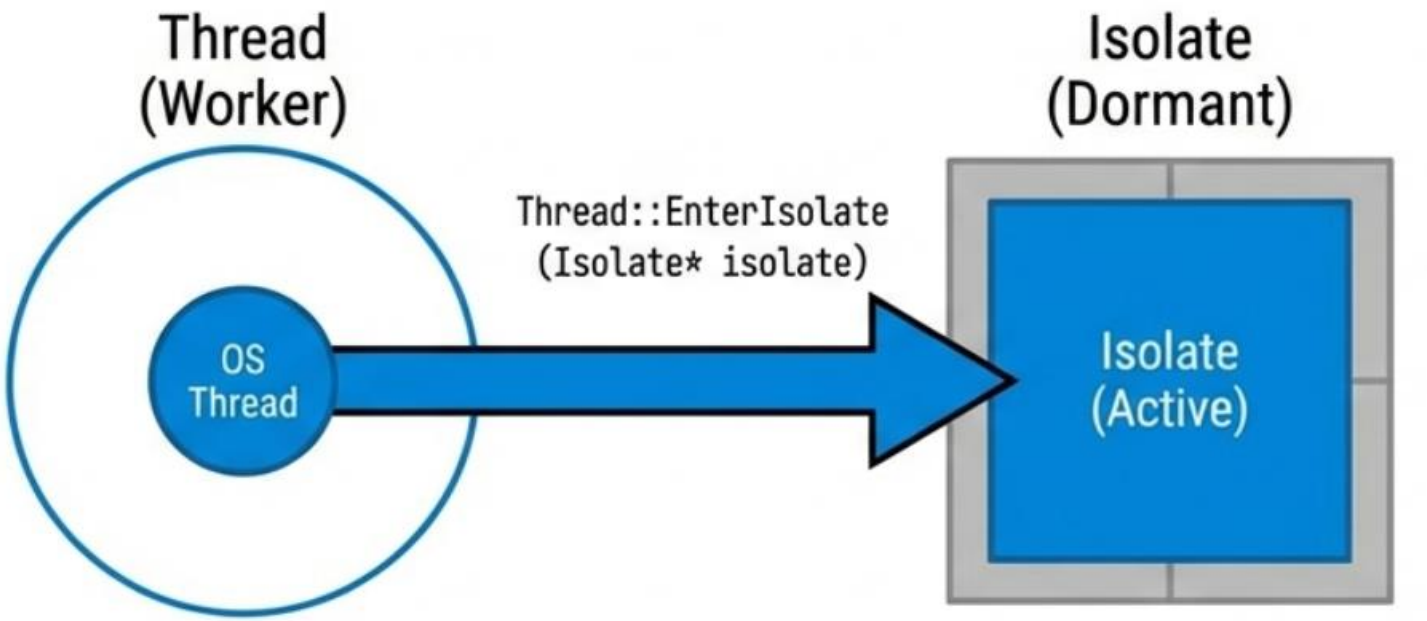
Назначение потока изоляту



// sdk/runtime/vm/thread.cc

```
if (!(is_nested_reenter & isolate->mutator_thread) ->OwnsSafepoint))) {
    group-IncreaseMutatorCount(nullptr, is_nested_reenter, false);
}
```

Назначение потока изоляту

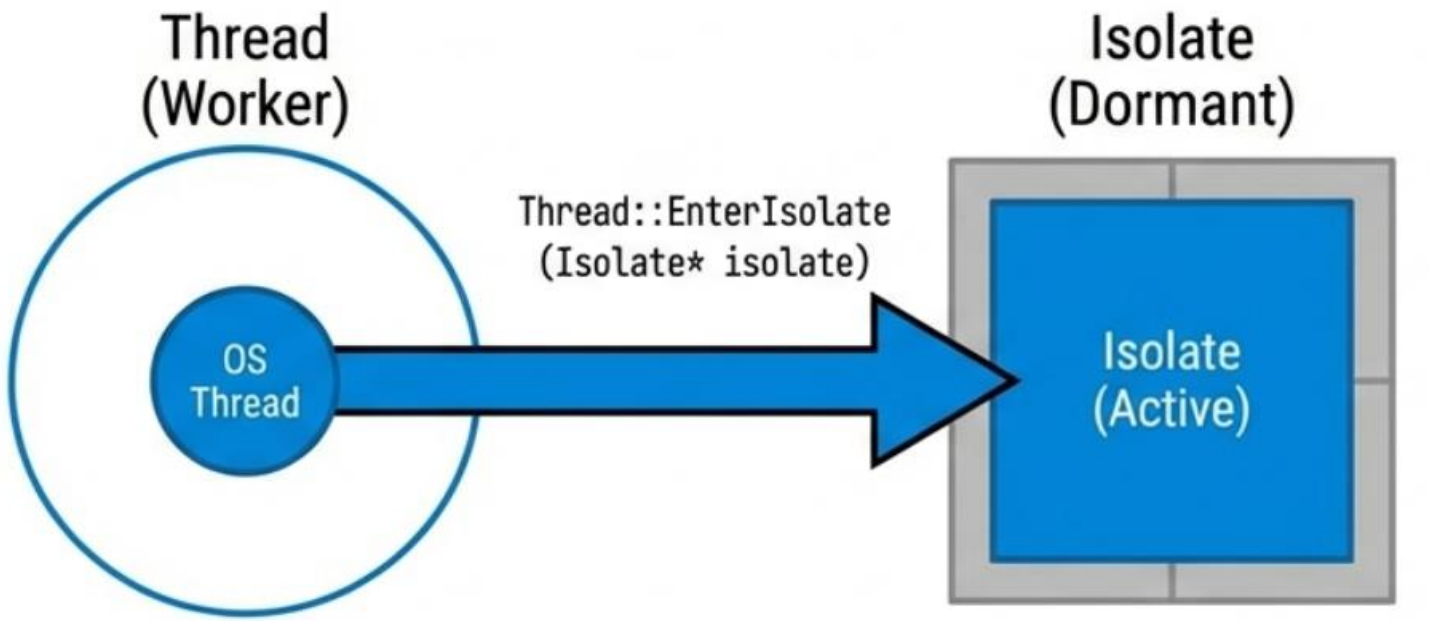


1. **Check Resumability.** Проверка флага is_resumable, позволяющая понять есть ли уже контекст?

// sdk/runtime/vm/thread.cc

```
if (!(is_nested_reenter & isolate-›mutator_thread) ->OwnsSafepoint))) {
    group-IncreaseMutatorCount(nullptr, is_nested_reenter, false);
}
```

Назначение потока изоляту



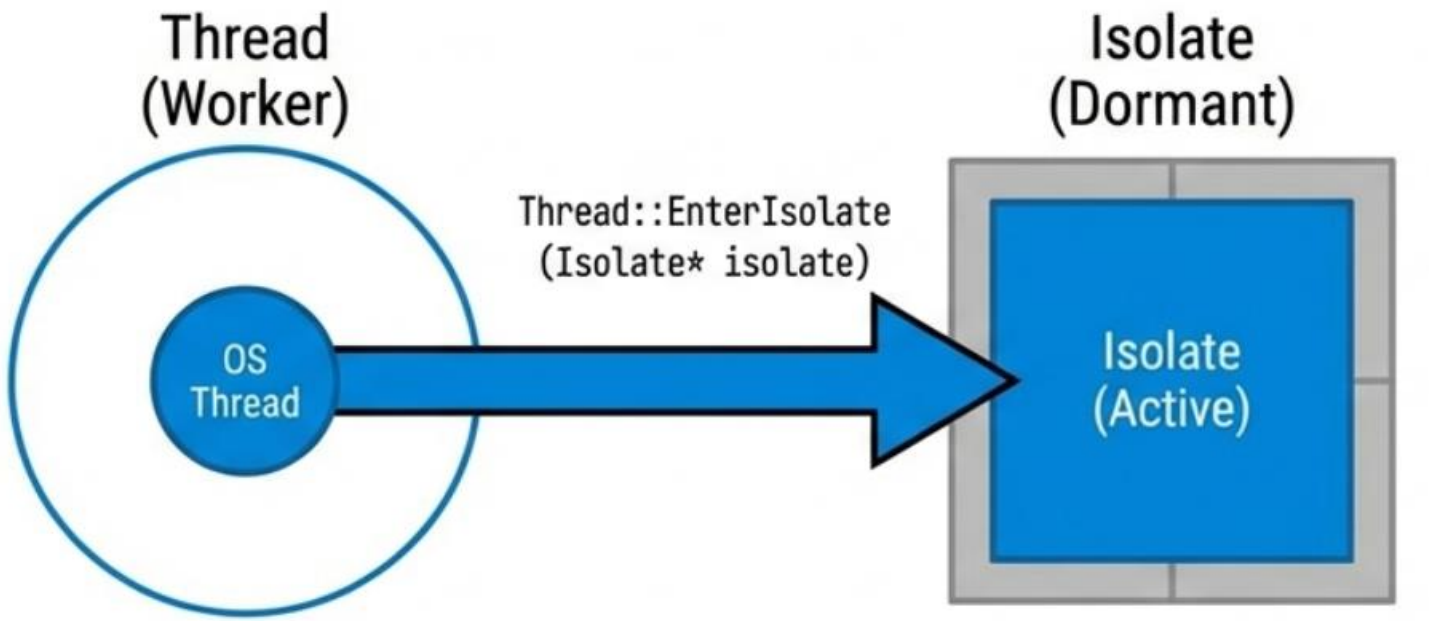
1. **Check Resumability.** Проверка флага is_resumable, позволяющая понять есть ли уже контекст?

2. **Registration.** Вызов IncreaseMutatorCount, который уведомляет группу о новом активном работнике.

// sdk/runtime/vm/thread.cc

```
if (!(is_nested_reenter & isolate-›mutator_thread) ->OwnsSafepoint))) {
    group-IncreaseMutatorCount(nullptr, is_nested_reenter, false);
}
```

Назначение потока изоляту



1. **Check Resumability.** Проверка флага is_resumable, позволяющая понять есть ли уже контекст?

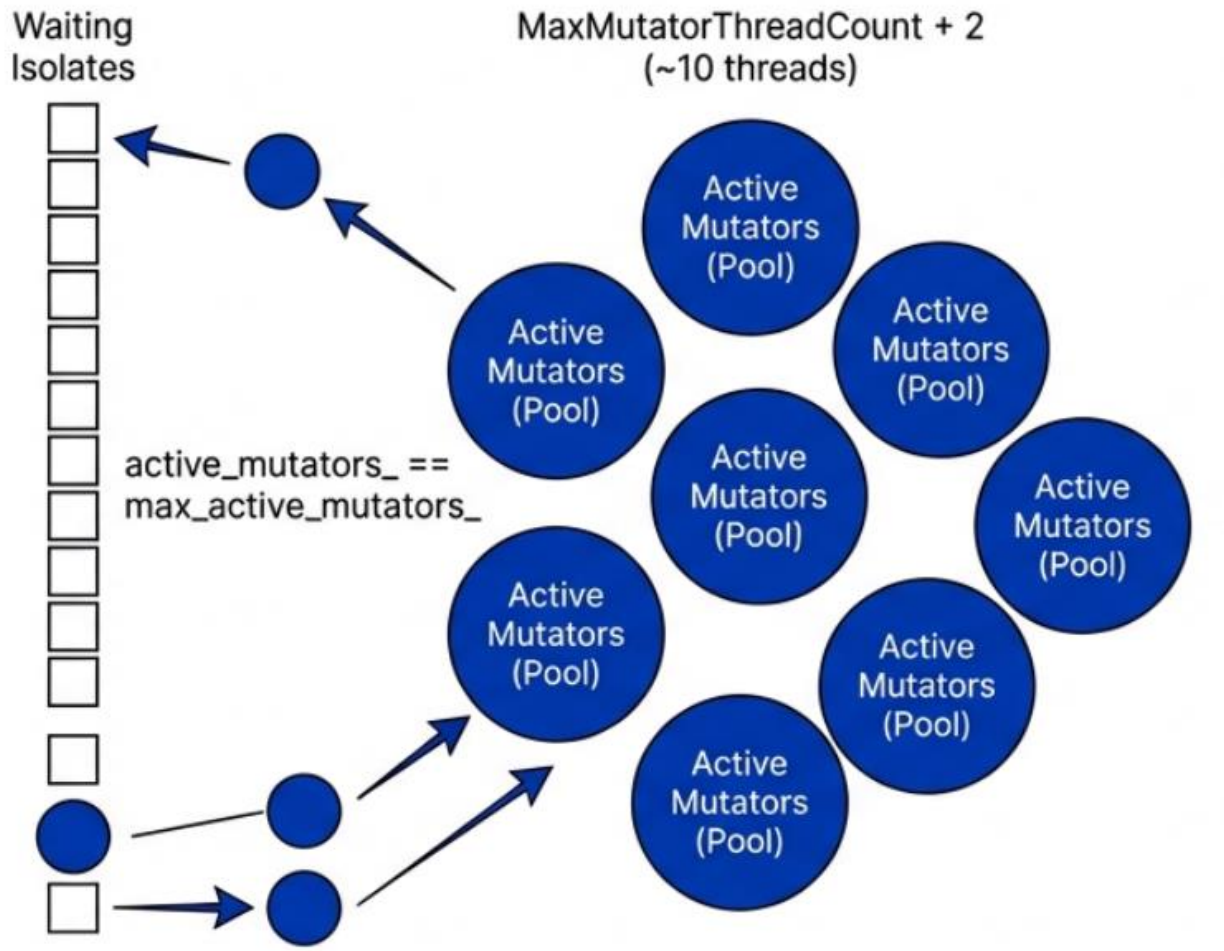
2. **Registration.** Вызов IncreaseMutatorCount, который уведомляет группу о новом активном работнике.

3. **Setup.** Инициализация DartMutatorState и переход изолята из состояния покоя в состояние выполнения.

// sdk/runtime/vm/thread.cc

```
if (!(is_nested_reenter & isolate-›mutator_thread) ->OwnsSafepoint))) {
    group-IncreaseMutatorCount(nullptr, is_nested_reenter, false);
}
```

Мультиплексирование потоков



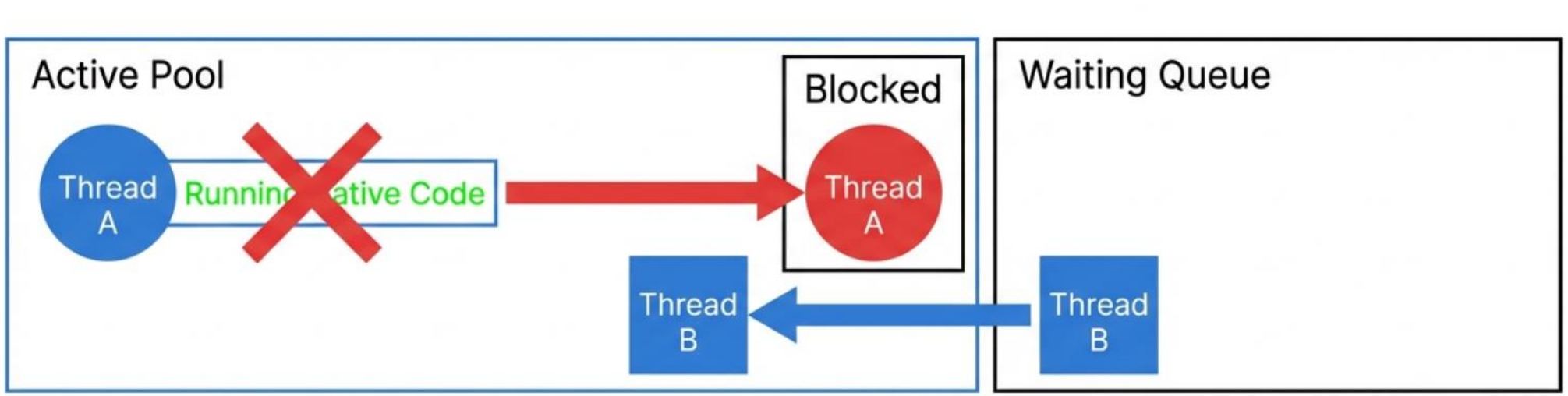
Если все слоты заняты, то новые потоки блокируются.

Даже если вы создали 1000 изолятов, **ТОЛЬКО** ~10 из них будут единомоментно выполняются физически (потоком ОС).

Мультиплексирование потоков. Таймауты

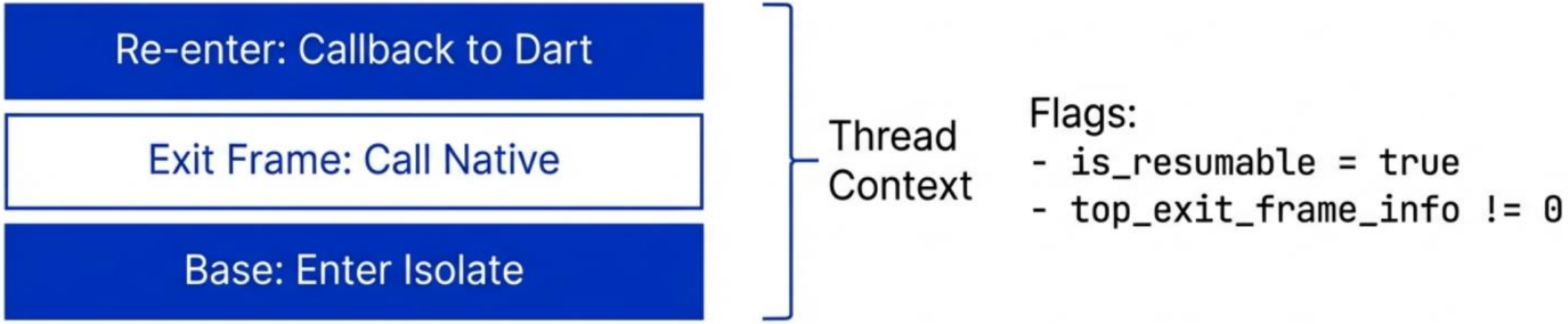
1. Новый поток пытается войти в MutatorThreadPool
2. Лимит max_active_mutators_ исчерпан
3. Поток переходит в состояние ожидания (блокируется)
4. Если через 120 мс место не освободилось, то запускается механизм вытеснения

Мультиплексирование потоков. Вытеснение



Длительные синхронные циклы на чистом Dart блокируют вытеснение

Мультиплексирование потоков. Вложенные переключения



При is_nested_reenter счетчик мутаторов в группе не увеличивается, так как поток «технически» продолжает владеть контекстом

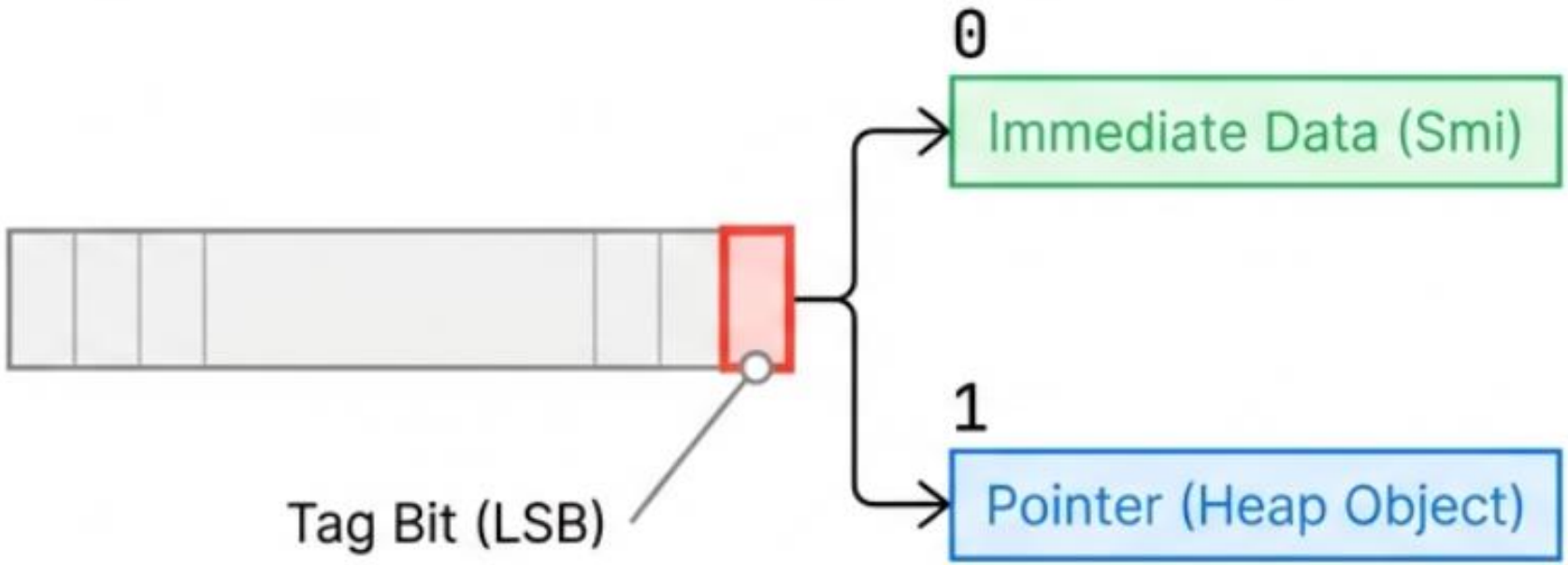
Мультиплексирование потоков. Thread

```
const intptr_t kMaxSuspendedThreads = 20;
auto group = thread->isolate_group();

return group->thread_registry()->active_isolates_count() < kMaxSuspendedThreads;
```

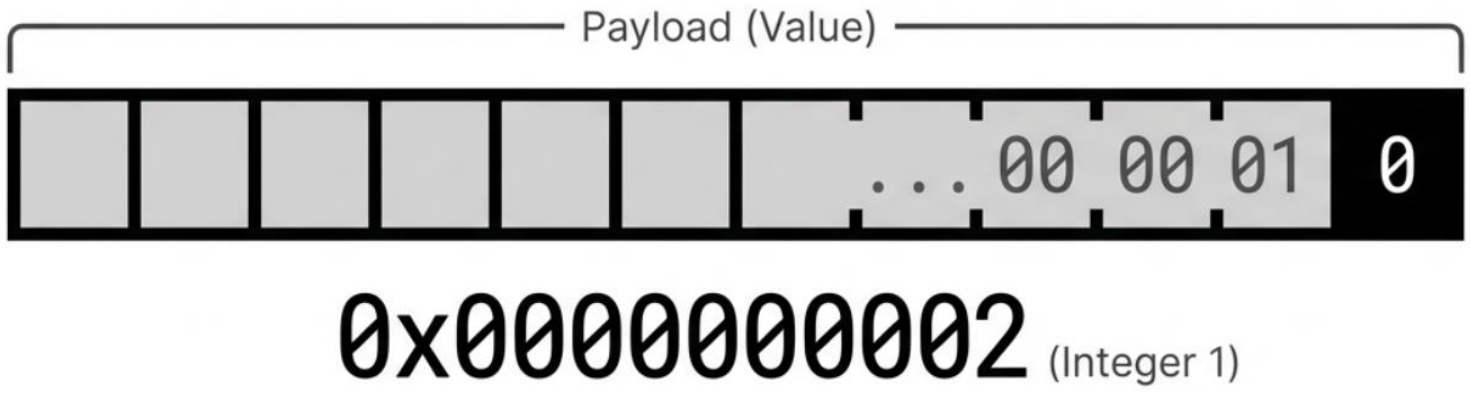
Представление объектов в памяти

Тегированные указатели



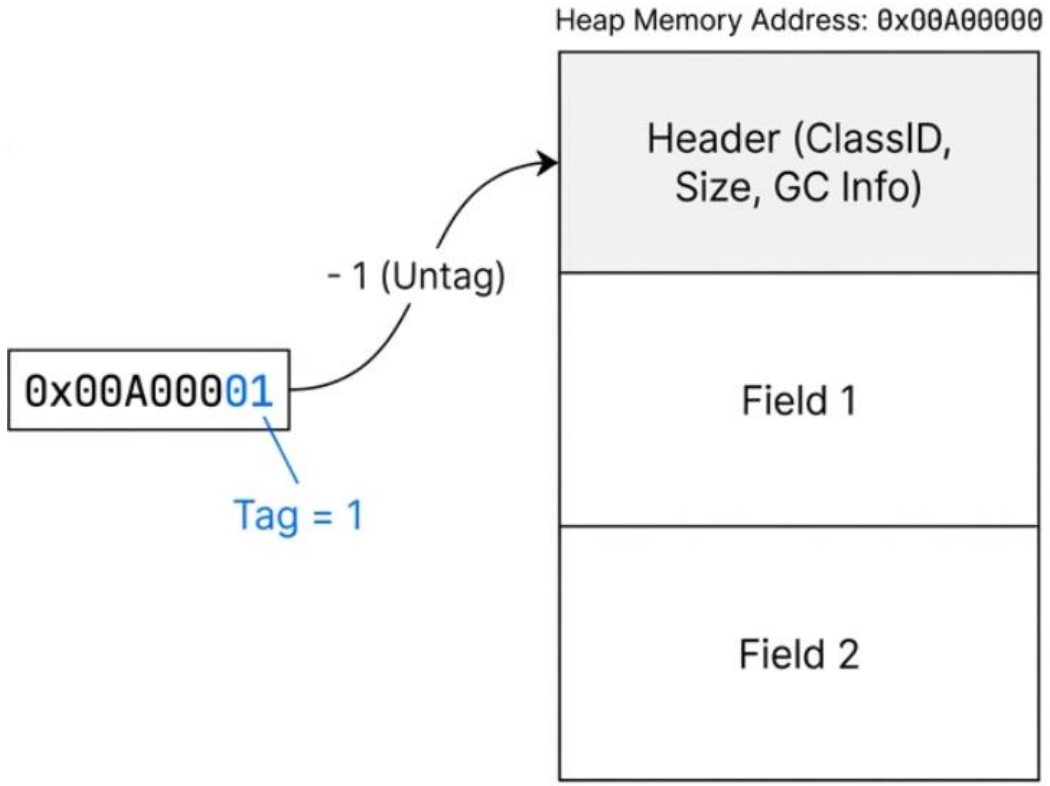
Информация о типе зашита в младший бит указателя

Smi (Small Integers)



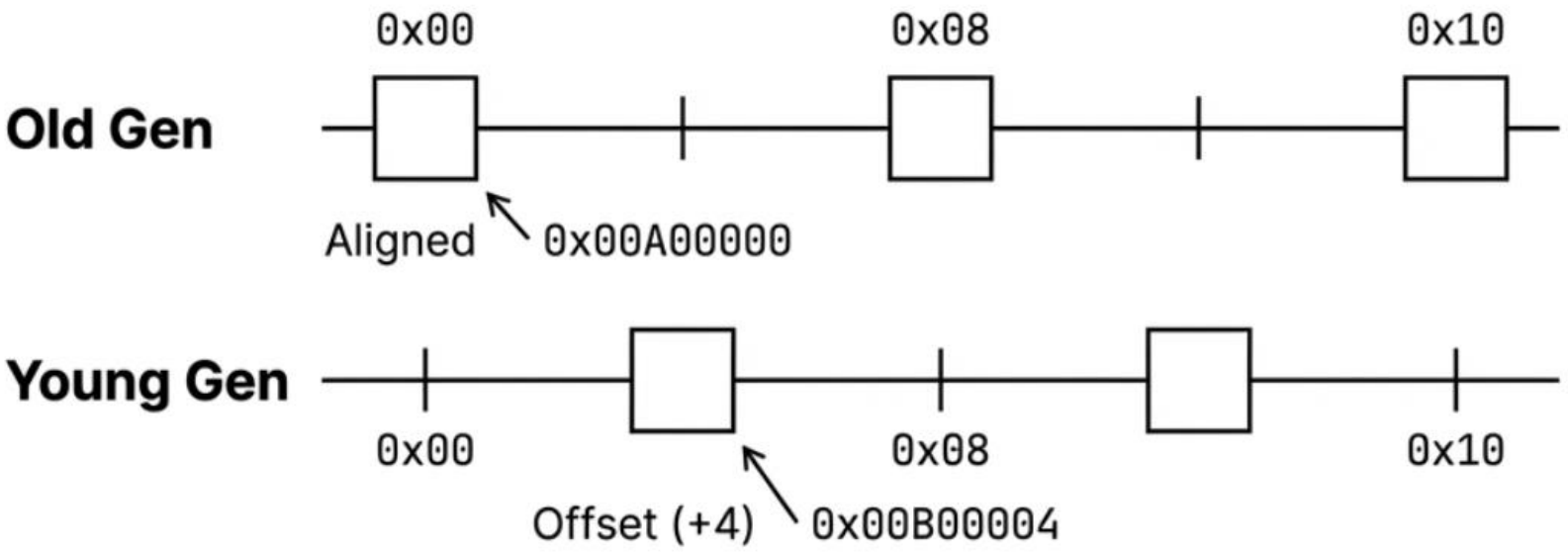
Если младший бит равен 0, Dart VM трактует оставшиеся биты не как адрес, а как само значение числа

Heap Objects



Если младший бит равен 1, то указатель содержит адрес объекта в памяти

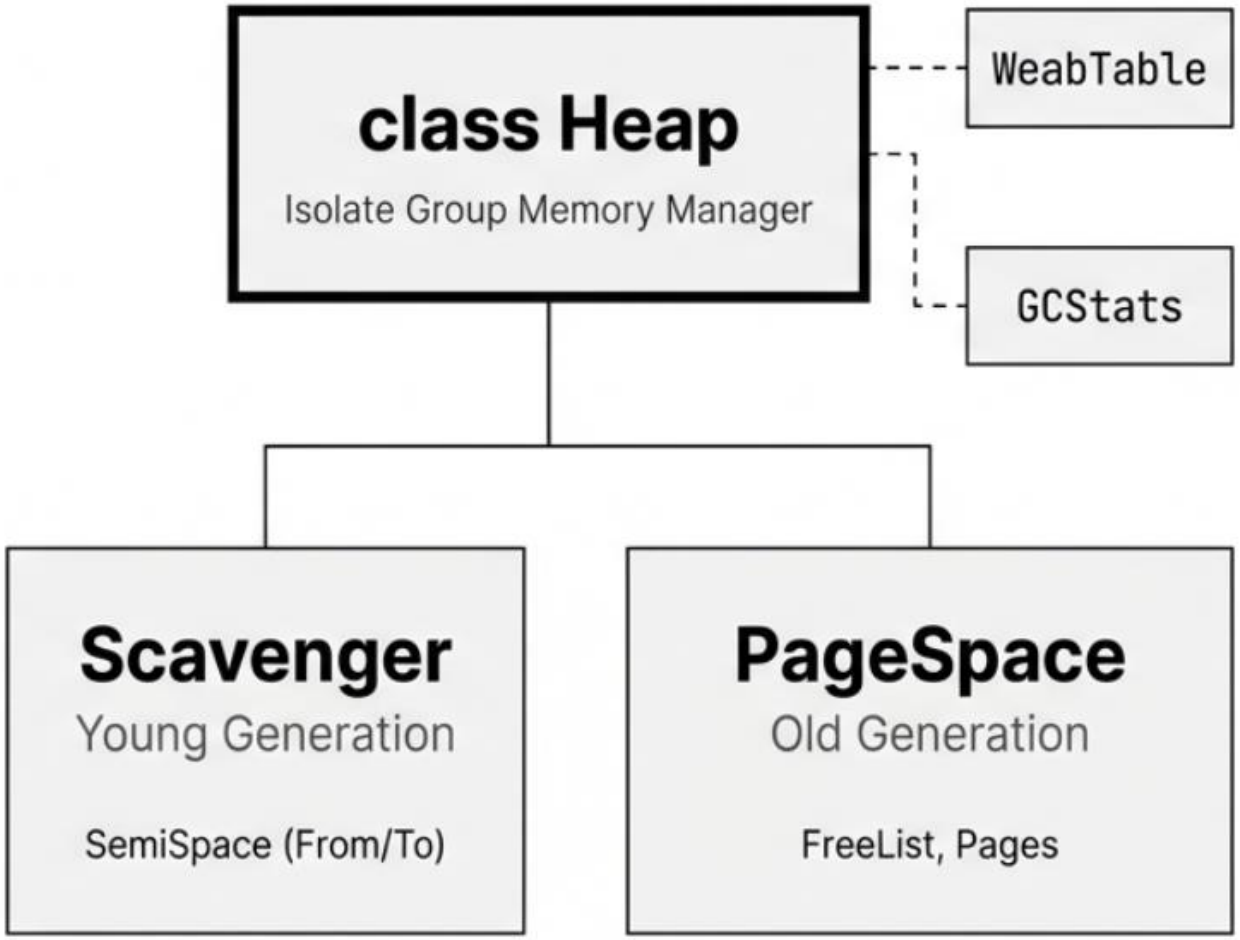
Heap Objects. Выравнивание



addr % double_word == 0

addr % double_word == word

Класс Heap

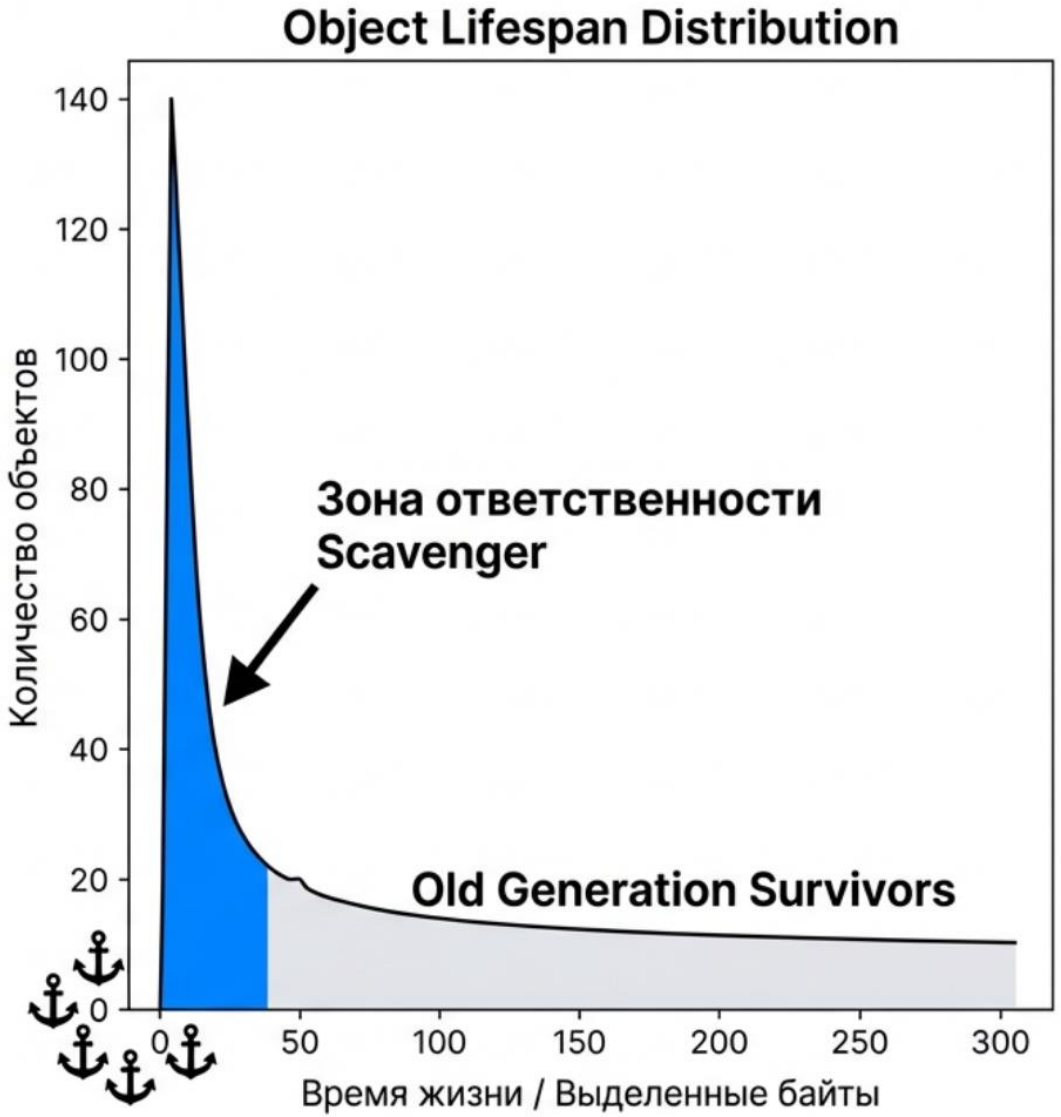


sdk/runtime/vm/heap/heap.h

Управляет двумя основными пространствами памяти через указатели:

1. Scavenger - «молодое поколение» (Young Gen)
2. PageSpace - «старое поколение» (Old Gen)

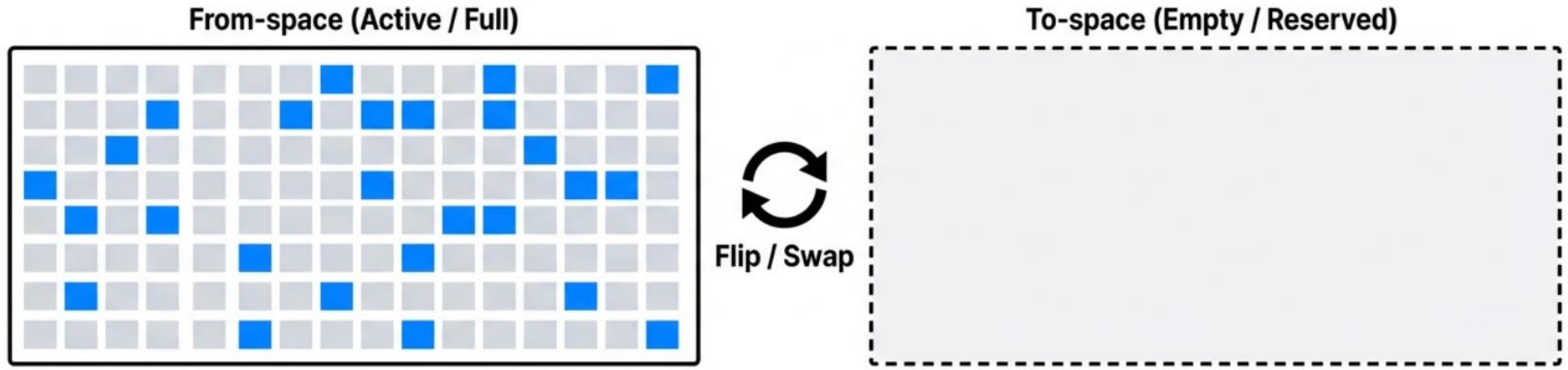
Зачем 2 поколения?



Большинство создаваемых объектов живут очень недолго (временные переменные, итераторы) и лишь немногие остаются в памяти до завершения программы

Сборка мусора в молодом поколении

Scavenger. Алгоритм Чейни и два полупространства (To/From)

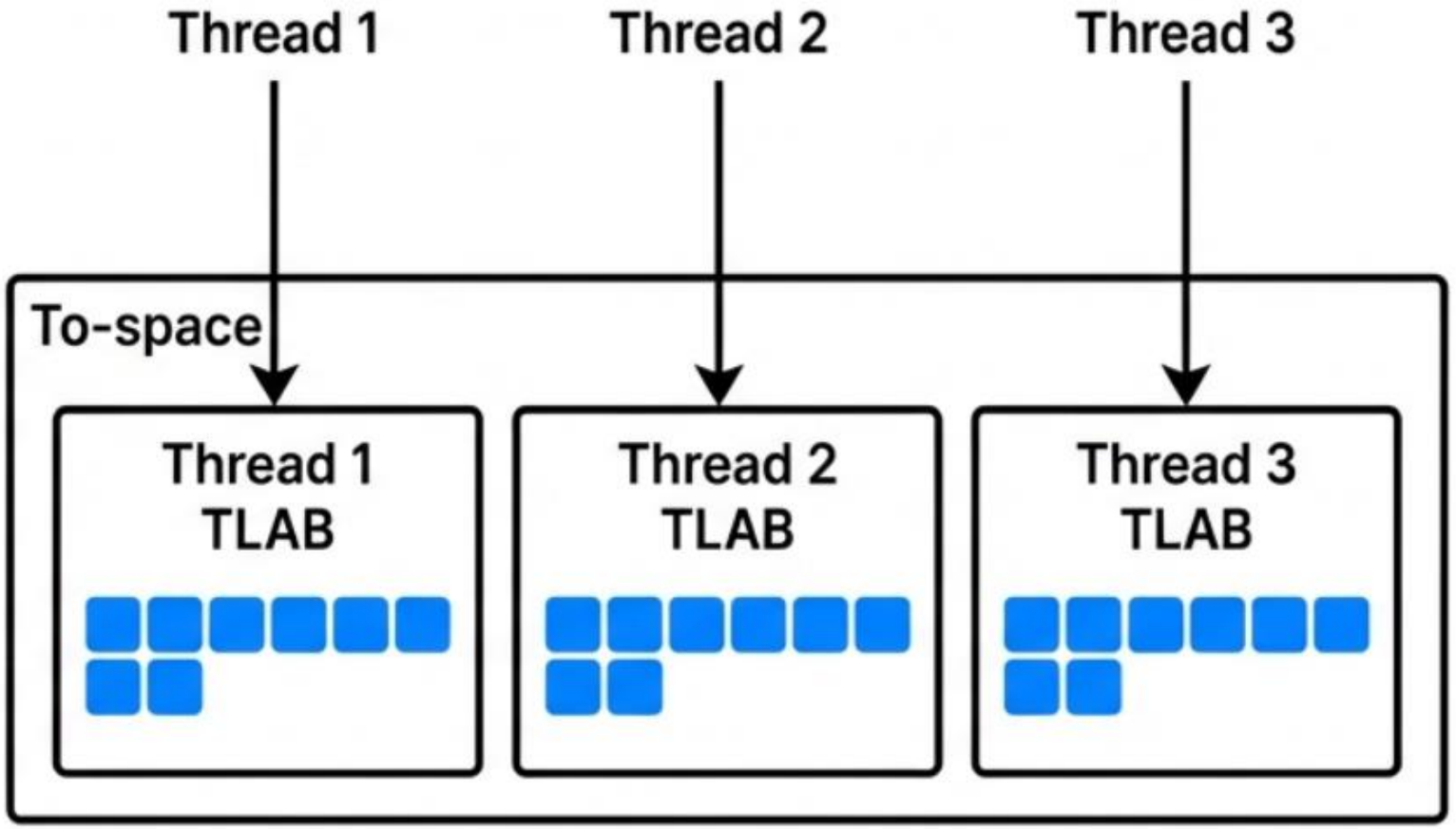


To-space (Активное)
Пространство, где в данный момент размещаются новые объекты

From-space (Резервное)
Пространство, используемое для копирования выживших объектов во время сборки мусора

The Flip (Смена ролей)
В начале цикла сборки мусора эти пространства меняются местами

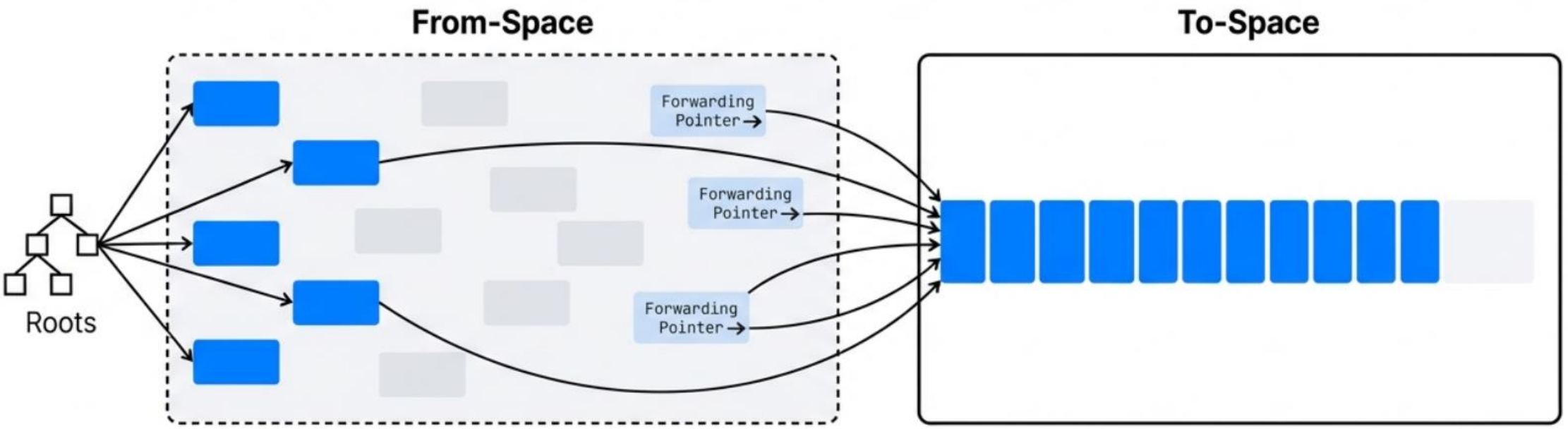
TLAB (Thread-Local Allocation Buffer)



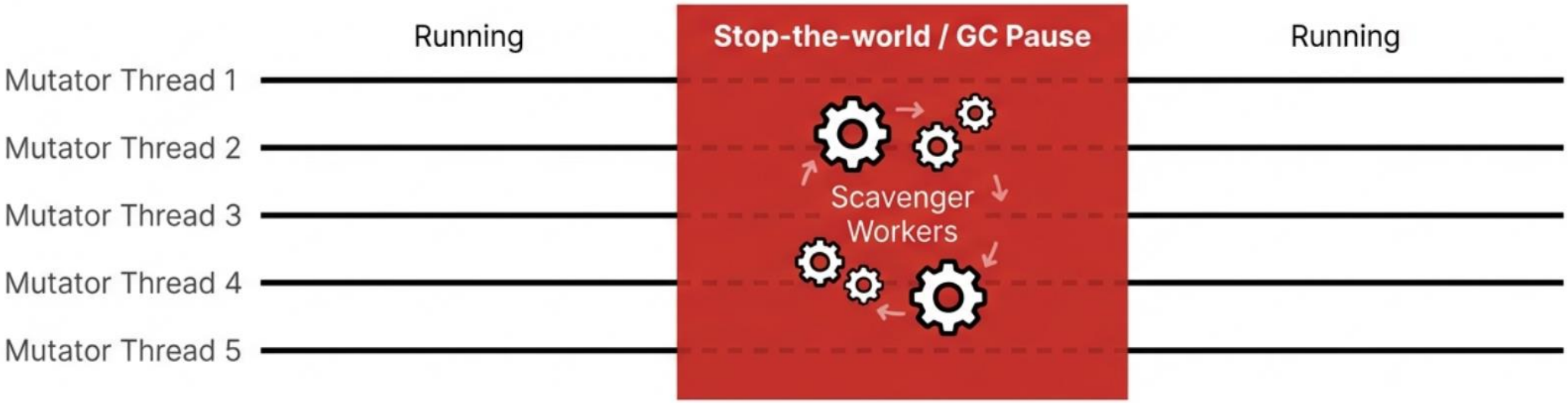
Каждому потоку выделяется персональный буфер (по умолчанию 512 КБ) внутри «To-space»

Блокировка производится только в тот момент, когда TLAB заполнен и поток просит у Scavenger новую страницу

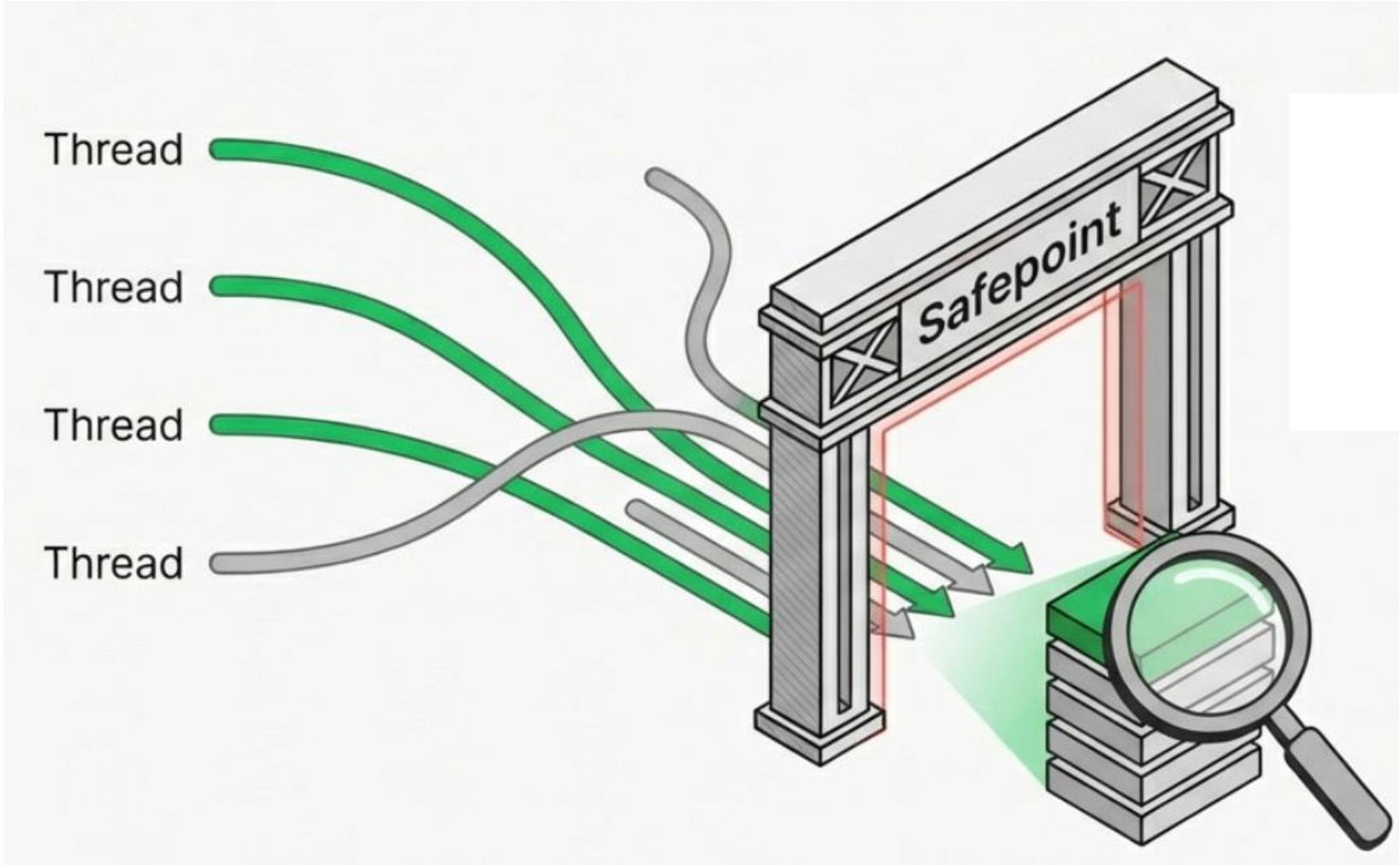
Как живые объекты перемещаются в To-space



«Остановка мира»

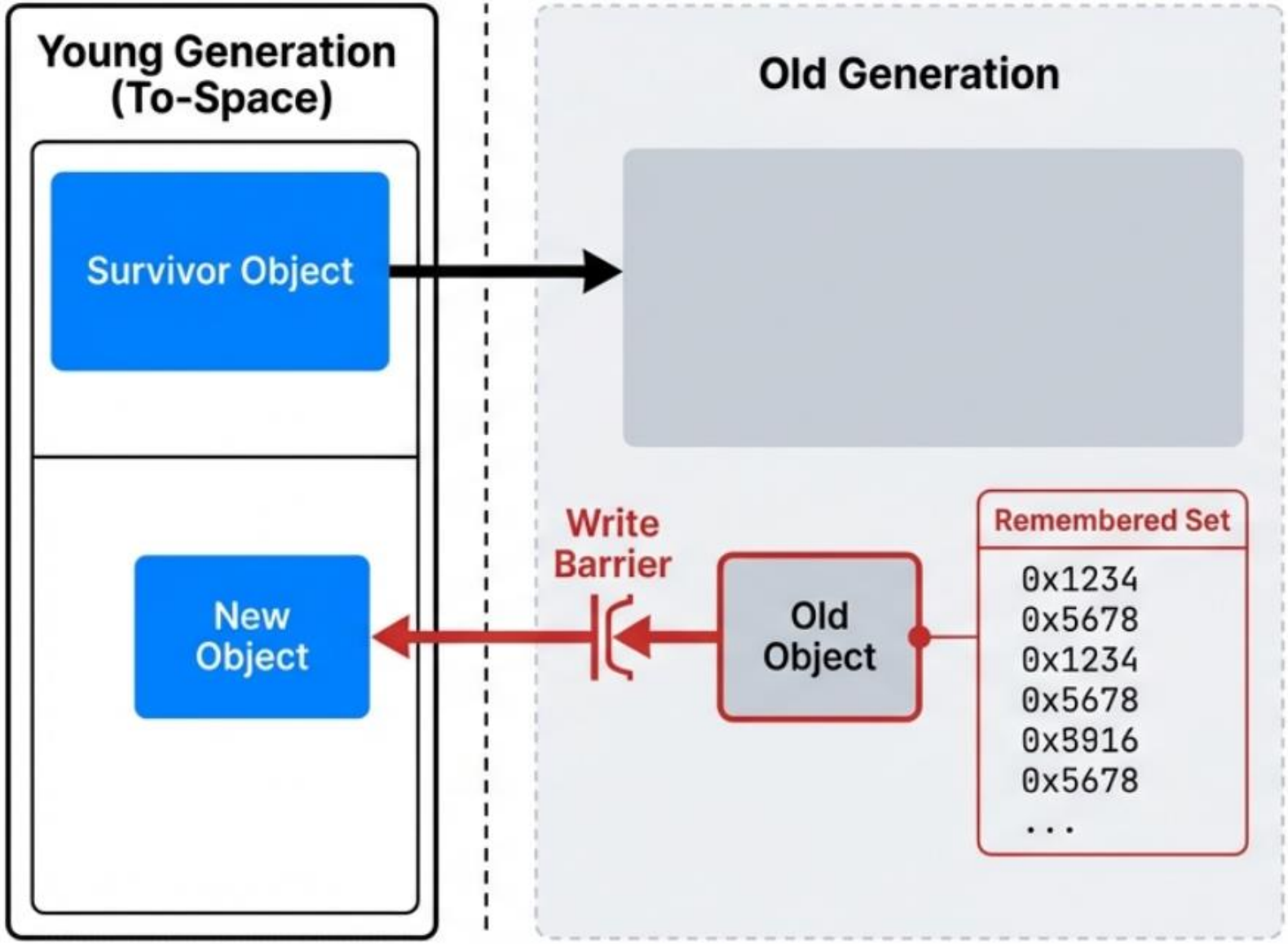


Safepoints



В точке остановки не должно быть «голых» указателей в регистрах

Продвижение и барьеры записи

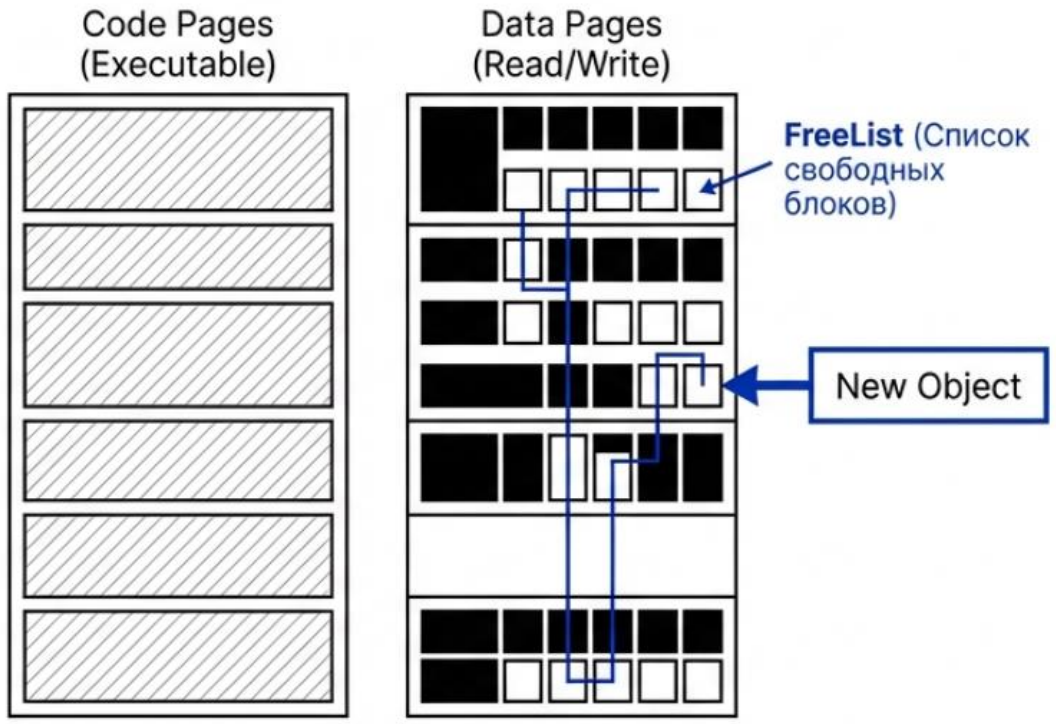


Если объект пережил несколько циклов сборки мусора, он переводится в разряд старичков

Remembered Set - список старых объектов, которые ссылаются на молодые

Сборка мусора в старом поколении

Организация памяти в старшем поколении



Неизменяемые инструкции, машинный код

Изменяемые Dart-объекты

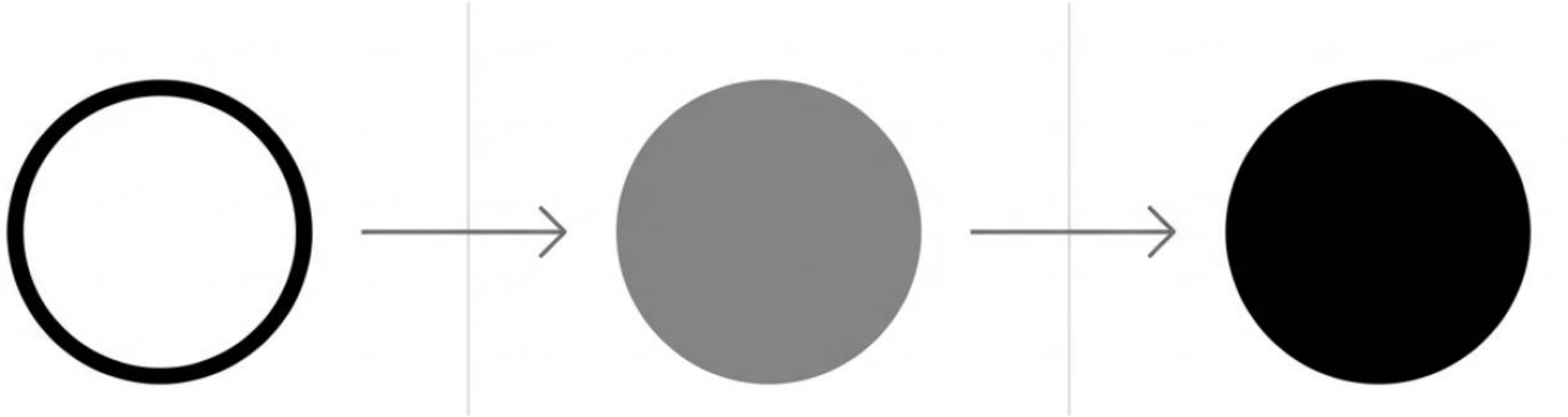
Старшее поколение реализовано в виде набора страниц памяти под управлением класса PageSpace

Стратегии сборки мусора

1. Concurrent Mark + Concurrent Sweep (по умолчанию)

2. Concurrent Mark + Parallel Compact

Трехцветная маркировка

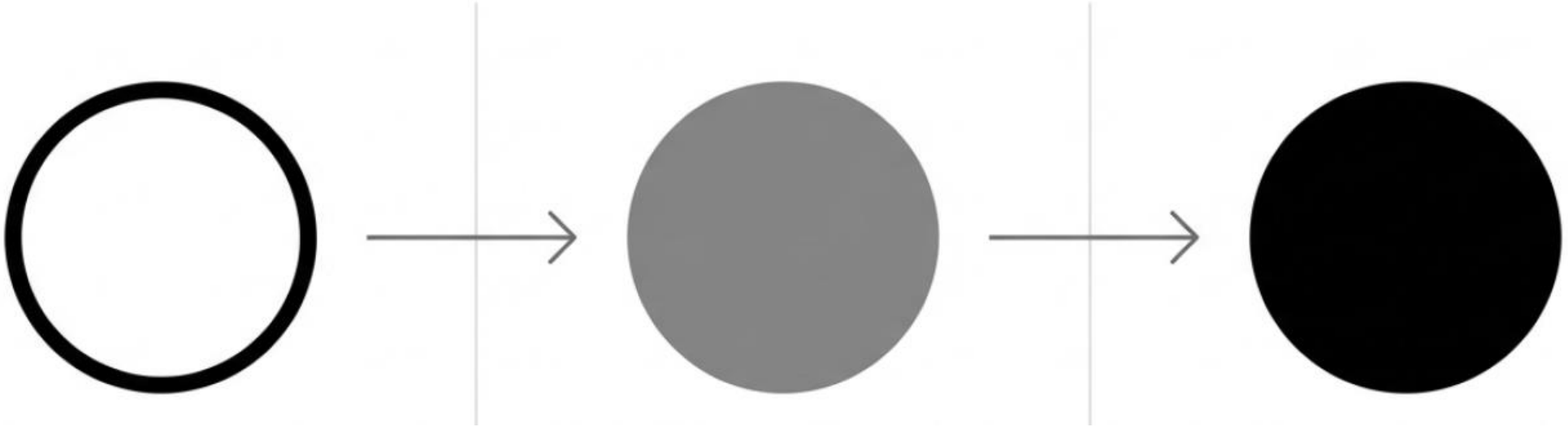


Кандидаты на удаление

Рабочий список (worklist)

Гарантировано живые

Трехцветная маркировка



Кандидаты на удаление

В начале цикла все объекты – белые и если они остаются таковыми в конце маркировки, то это мусор

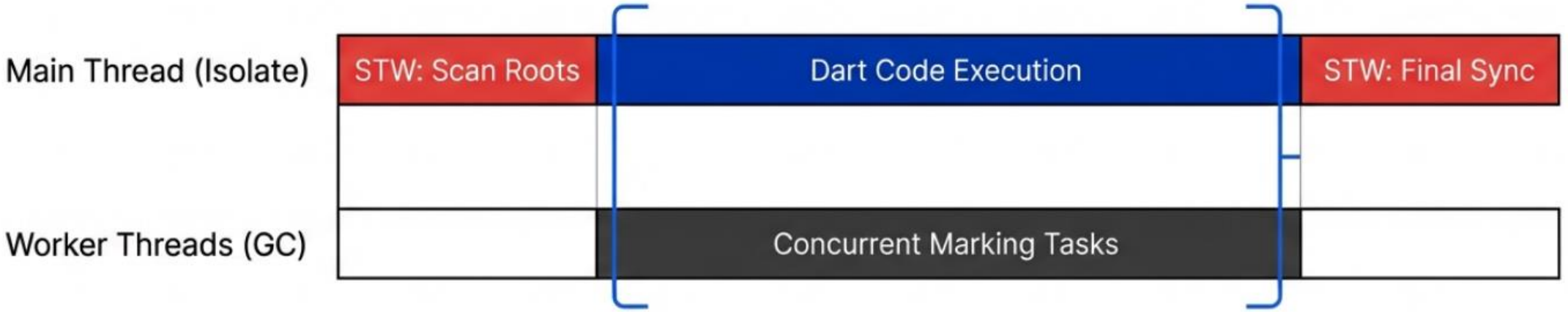
Рабочий список (worklist)

Объект достижим из корней, но его дочерние ссылки еще не просканированы (помещается в MarkingStack)

Гарантировано живые

Объект и все его ссылки просканированы

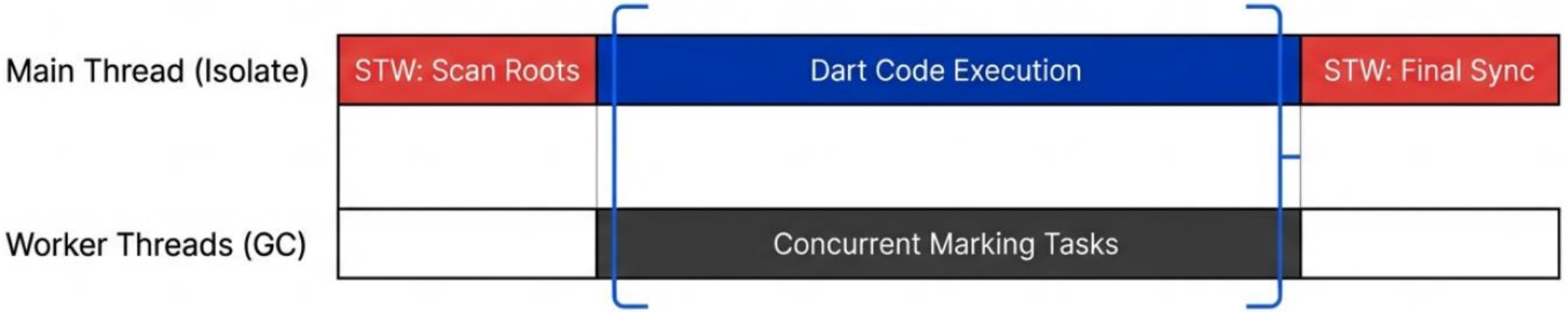
Трехцветная маркировка



1. Начальная маркировка

Стартует после краткой паузы и сканирует только «корни», окрашивая объекты в серый цвет

Трехцветная маркировка



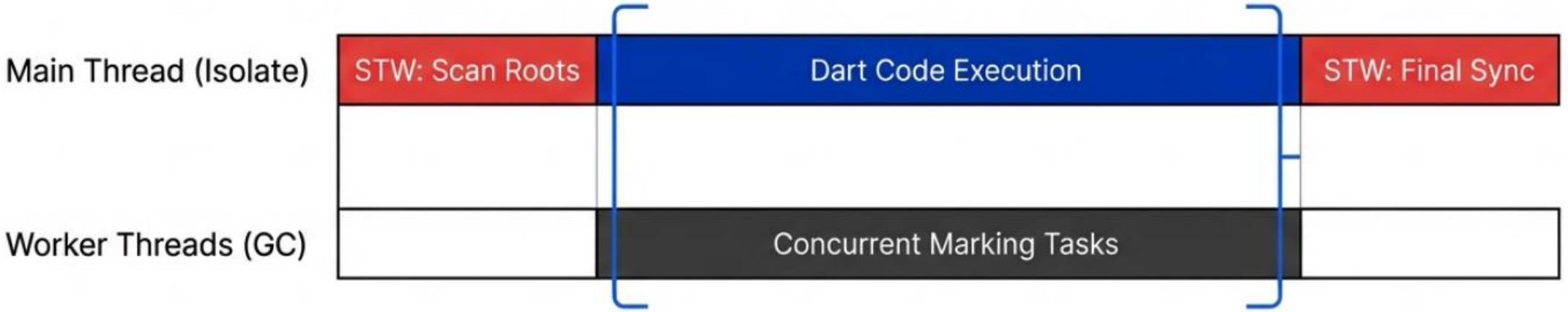
1. Начальная маркировка

Стартует после краткой паузы и сканирует только «корни», окрашивая объекты в серый цвет

2. Конкурентная фаза

Потоки-маркировщики извлекают объекты из MarkingStack и красят их

Трехцветная маркировка



1. Начальная маркировка

Стартует после краткой паузы и сканирует только «корни», окрашивая объекты в серый цвет

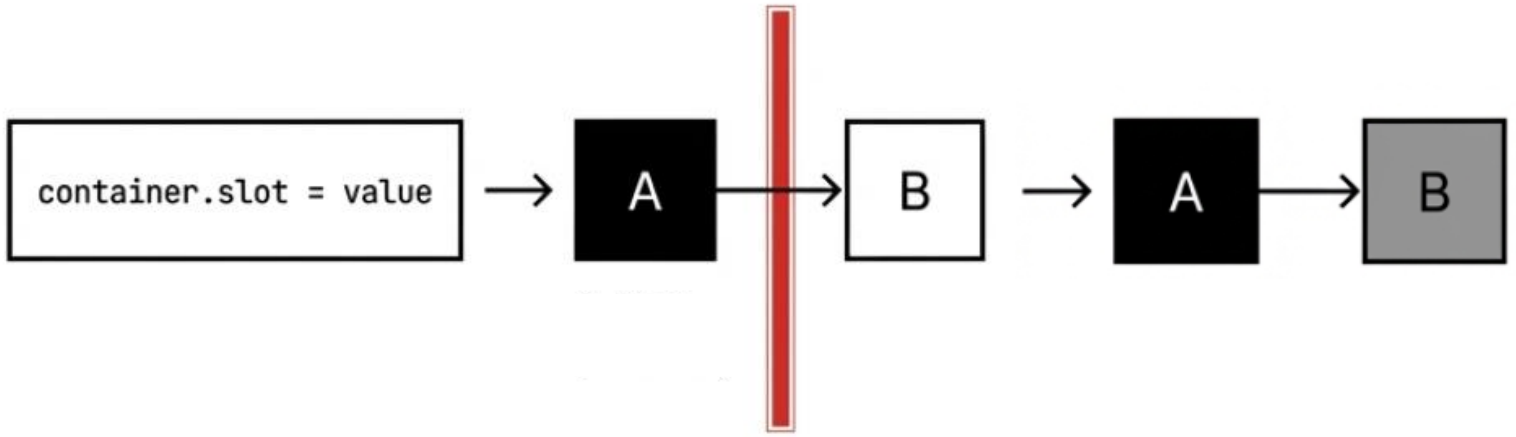
2. Конкурентная фаза

Потоки-маркировщики извлекают объекты из MarkingStack и красят их

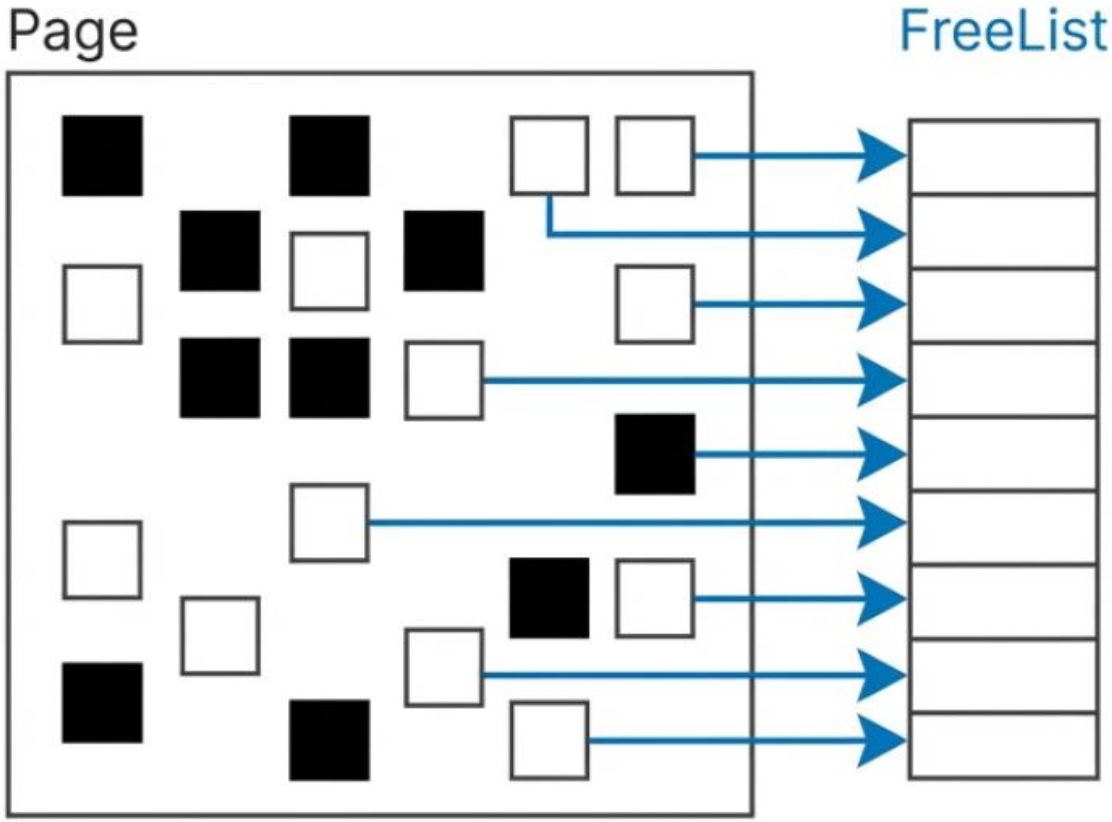
3. Финальная синхронизация

Повторная краткая пауза в работе исполняемого кода для обработки изменений, накопленных за время работы

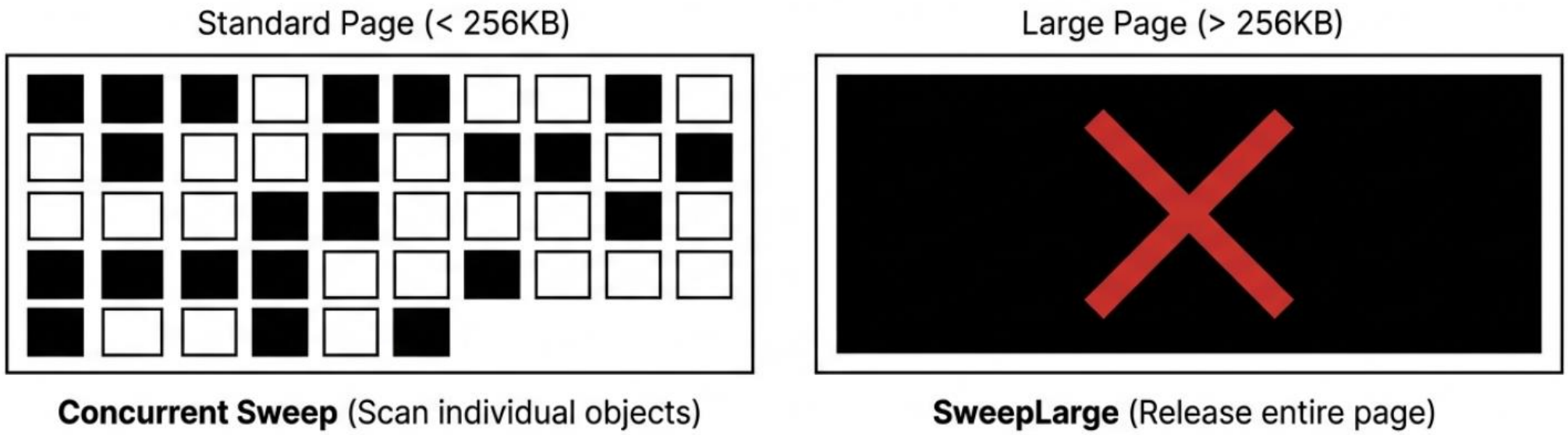
Барьеры записи



Фоновая очистка (Concurrent Sweep)



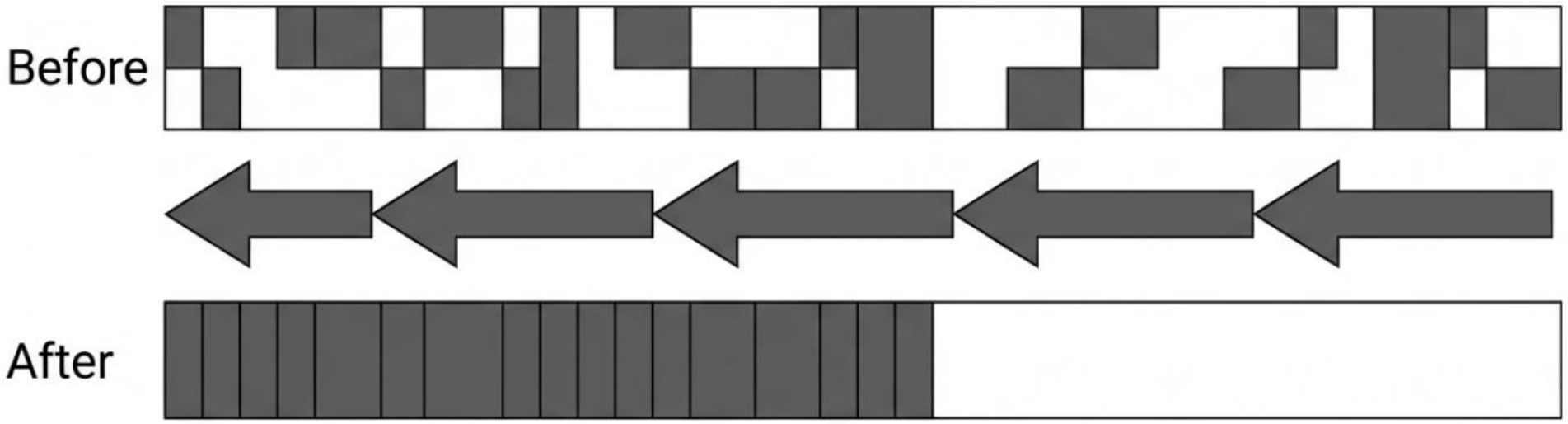
SweepLarge



Операция быстрая, но требует «остановки мира»

Основная проблема Concurrent Sweep - фрагментация

Parallel Compact



Флаги конфигурации Dart VM

--new-gen-semi-max-size=<value>

// JIT
> dart --new-gen-semi-max-size=32 run bin\main.dart

// Конфигурация runtime Dart VM запускаемого исполняемого файла
> bin\main.exe --new-gen-semi-max-size=32

Флаг устанавливает максимальный размер (в МБ) для полупространства (semi-space) в молодом поколении сборщика мусора. Влияет на количество выделяемых под изоляционную группу активных потоков-мутаторов

--new_gen_growth_factor=<value>

// JIT
> dart --new_gen_growth_factor=4 run bin\main.dart

// Конфигурация runtime Dart VM запускаемого исполняемого файла
> bin\main.exe --new_gen_growth_factor=4

Флаг позволяет задать коэффициент увеличения емкости кучи молодого поколения при его расширении

--marker-tasks=<value>

```
// JIT
> dart --marker-tasks=4 run bin\main.dart

// Конфигурация runtime Dart VM запускаемого исполняемого файла
> bin\main.exe --marker-tasks=4
```

Флаг устанавливает количество потоков-маркировщиков для GC старого поколения. По умолчанию равен 2. Если зададите 0, то вся работа будет выполняется в основном потоке.

Выводы

Выводы

- Изолят всегда исполняется только одним потоком-мутатором

Выводы

- Изолят всегда исполняется только одним потоком-мутатором
- Количество потоков мутаторов ограничено

Выводы

- Изолят всегда исполняется только одним потоком-мутатором
- Количество потоков мутаторов ограничено
- Хочешь больше потоков-мутаторов? Используй флаг --new-gen-semi-max-size

Выводы

- Изолят всегда исполняется только одним потоком-мутатором
- Количество потоков мутаторов ограничено
- Хочешь больше потоков-мутаторов? Используй флаг --new-gen-semi-max-size
- Сборка мусора в молодом поколении работает быстро, но требуется «остановка мира»

Выводы

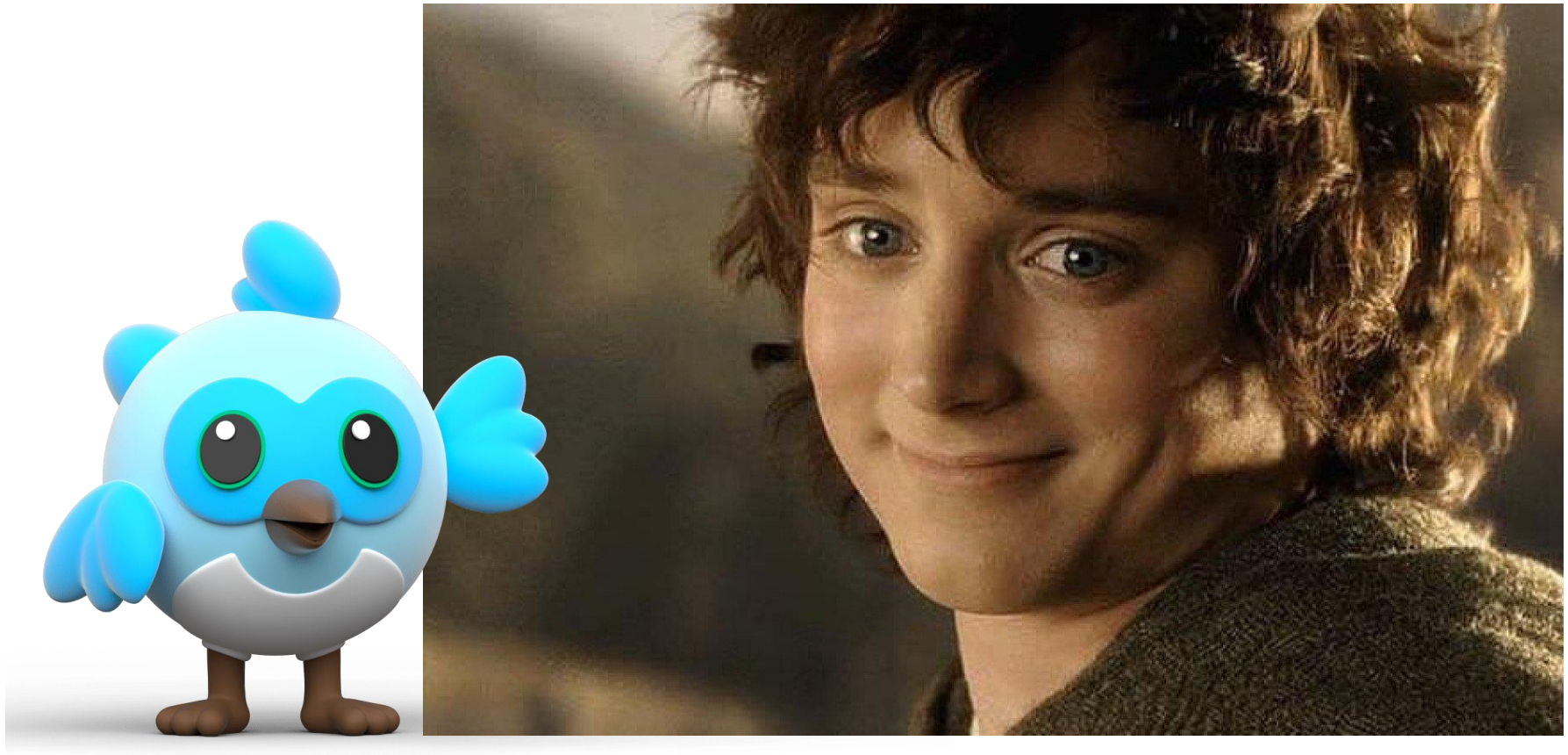
- Изолят всегда исполняется только одним потоком-мутатором
- Количество потоков мутаторов ограничено
- Хочешь больше потоков-мутаторов? Используй флаг --new-gen-semi-max-size
- Сборка мусора в молодом поколении работает быстро, но требуется «остановка мира»
- Сборка мусора в старшем поколении зависит от фрагментации памяти. По умолчанию – Mark-Sweep, когда все плохо – Mark-Compact

Выводы

- Изолят всегда исполняется только одним потоком-мутатором
- Количество потоков мутаторов ограничено
- Хочешь больше потоков-мутаторов? Используй флаг --new-gen-semi-max-size
- Сборка мусора в молодом поколении работает быстро, но требуется «остановка мира»
- Сборка мусора в старшем поколении зависит от фрагментации памяти. По умолчанию – Mark-Sweep, когда все плохо – Mark-Compact
- Избегайте длительных синхронных операций в изолятах

Выводы

- Изолят всегда исполняется только одним потоком-мутатором
- Количество потоков мутаторов ограничено
- Хочешь больше потоков-мутаторов? Используй флаг --new-gen-semi-max-size
- Сборка мусора в молодом поколении работает быстро, но требуется «остановка мира»
- Сборка мусора в старшем поколении зависит от фрагментации памяти. По умолчанию – Mark-Sweep, когда все плохо – Mark-Compact
- Избегайте длительных синхронных операций в изолятах
- Не стоит бездумно поднимать большое количество изолятов



COME TO THE DART SIDE