



Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»

Выпускная квалификационная работа бакалавра

Метод автоматического управления памятью с гарантированным временем выполнения на основе подсчёта ссылок

Студент: Сапожков Андрей Максимович ИУ7-83Б

Научный руководитель: Строганов Юрий Владимирович

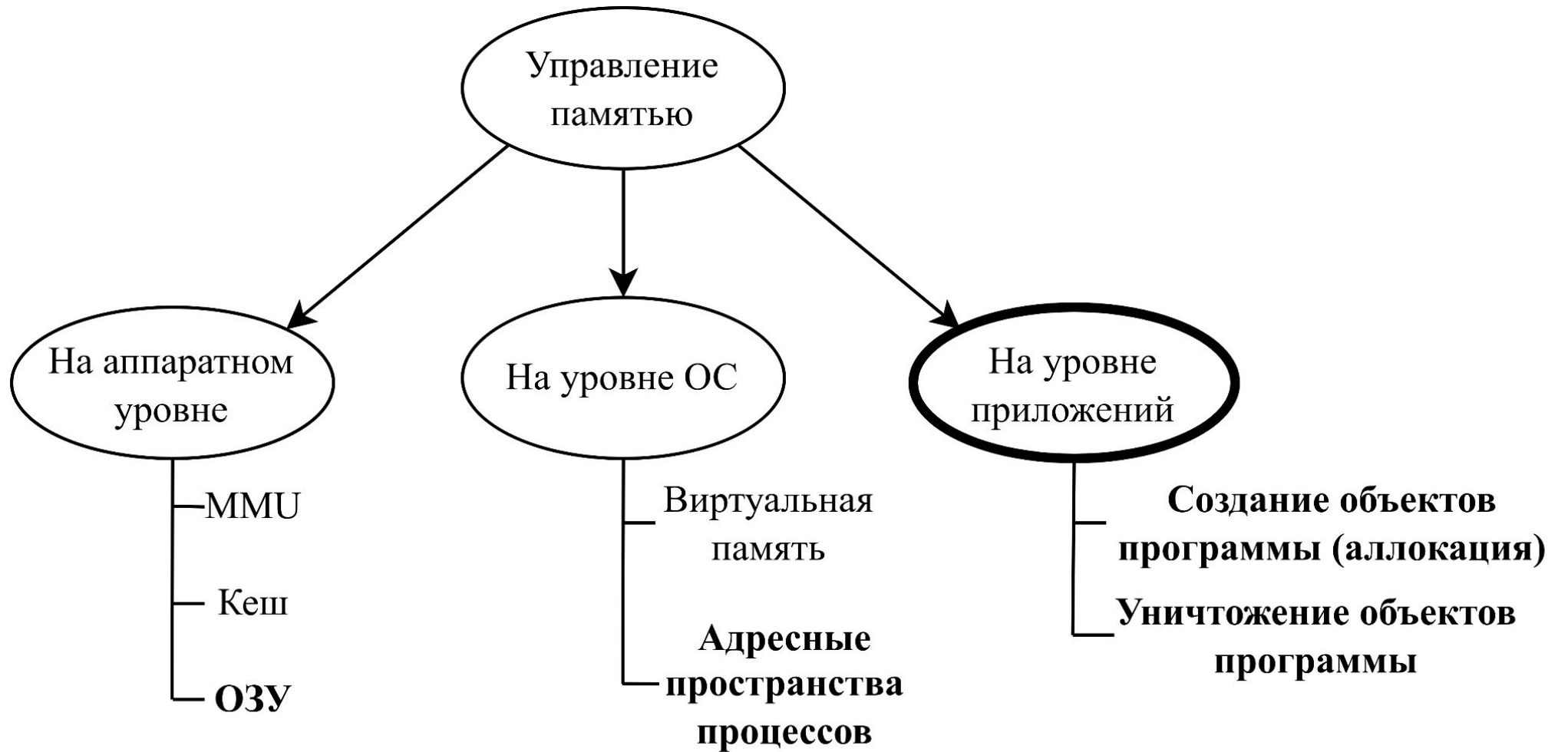
Цель и задачи

Цель — разработка метода автоматического управления памятью с гарантированным временем выполнения на основе подсчёта ссылок.

Задачи:

1. Проанализировать существующие методы распределения памяти в языках программирования с автоматической сборкой мусора.
2. Спроектировать метод автоматического управления памятью.
3. Реализовать спроектированный метод в виде подключаемой библиотеки.
4. Определить области применения реализованного метода.

Управление памятью



Управление памятью на уровне приложений

- Ручное, определяемое человеческим фактором.
- **Автоматическое**, определяемое **алгоритмами распределения памяти**.

```
int main(void) {  
    int *a = new int;  
    // ...  
    *a = 5;  
    // ...  
    delete a;  
}
```

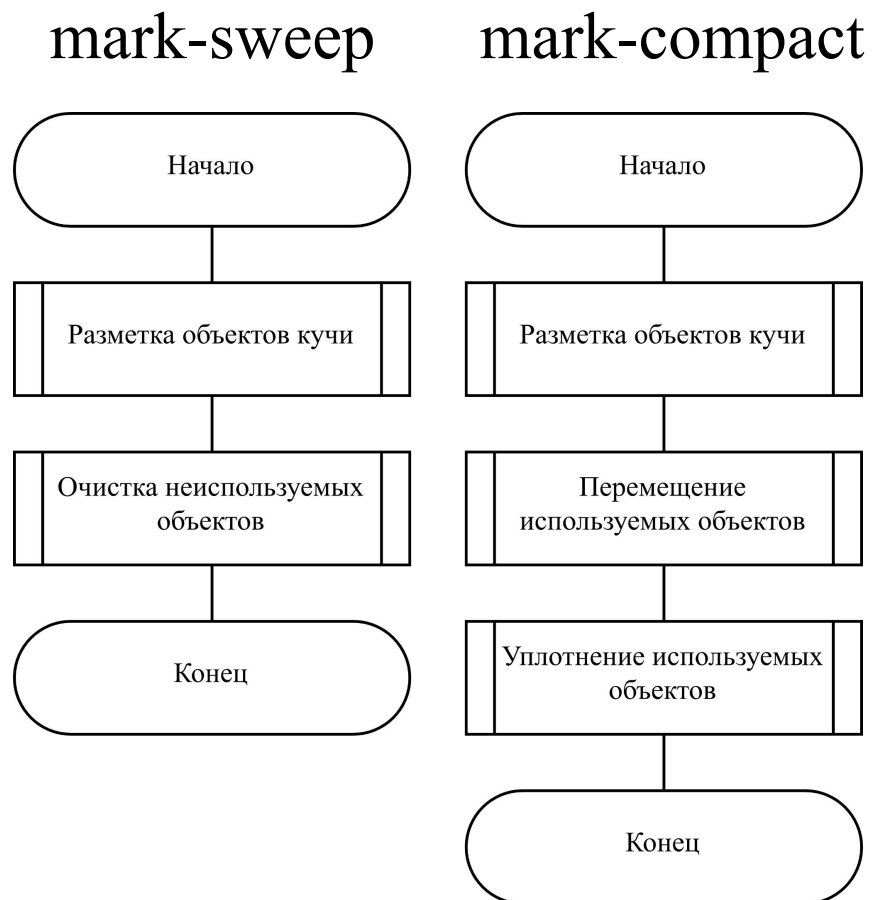
Ручное управление
памятью (C++)

```
package main  
func main() {  
    a := new(int)  
    // ...  
    *a = 5  
    // ...  
}
```

Автоматическое управление
памятью (Golang)

Автоматическая переработка динамически выделяемой памяти

Трассирующая сборка мусора



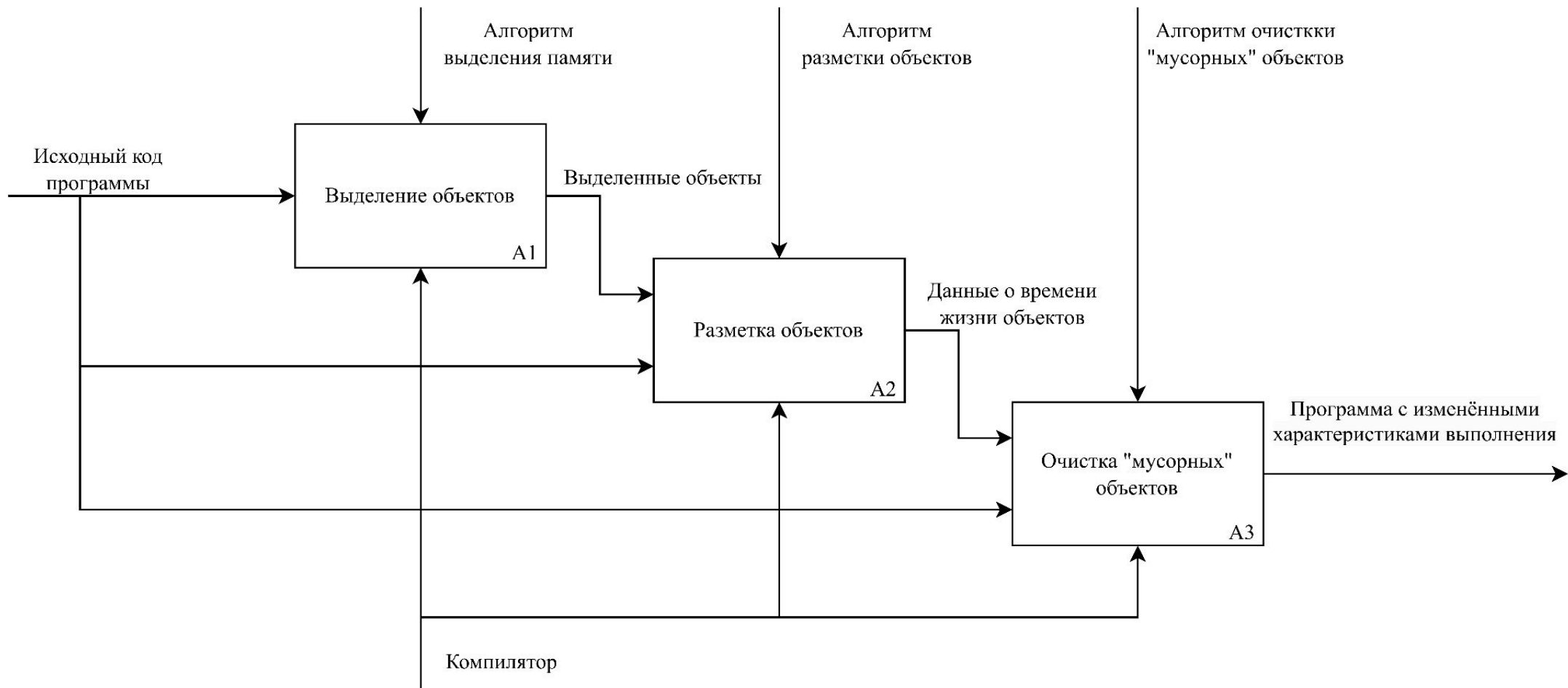
Подсчёт ссылок

- + более равномерное распределение накладных расходов;
- + устойчивость к высоким нагрузкам;
- + масштабируемость по размеру кучи;
- + не требует среды выполнения языка;
- накладные расходы на чтение/запись;
- атомарность операций;
- НЕ ВЫЯВЛЯЮТСЯ ЦИКЛЫ ССЫЛОК.

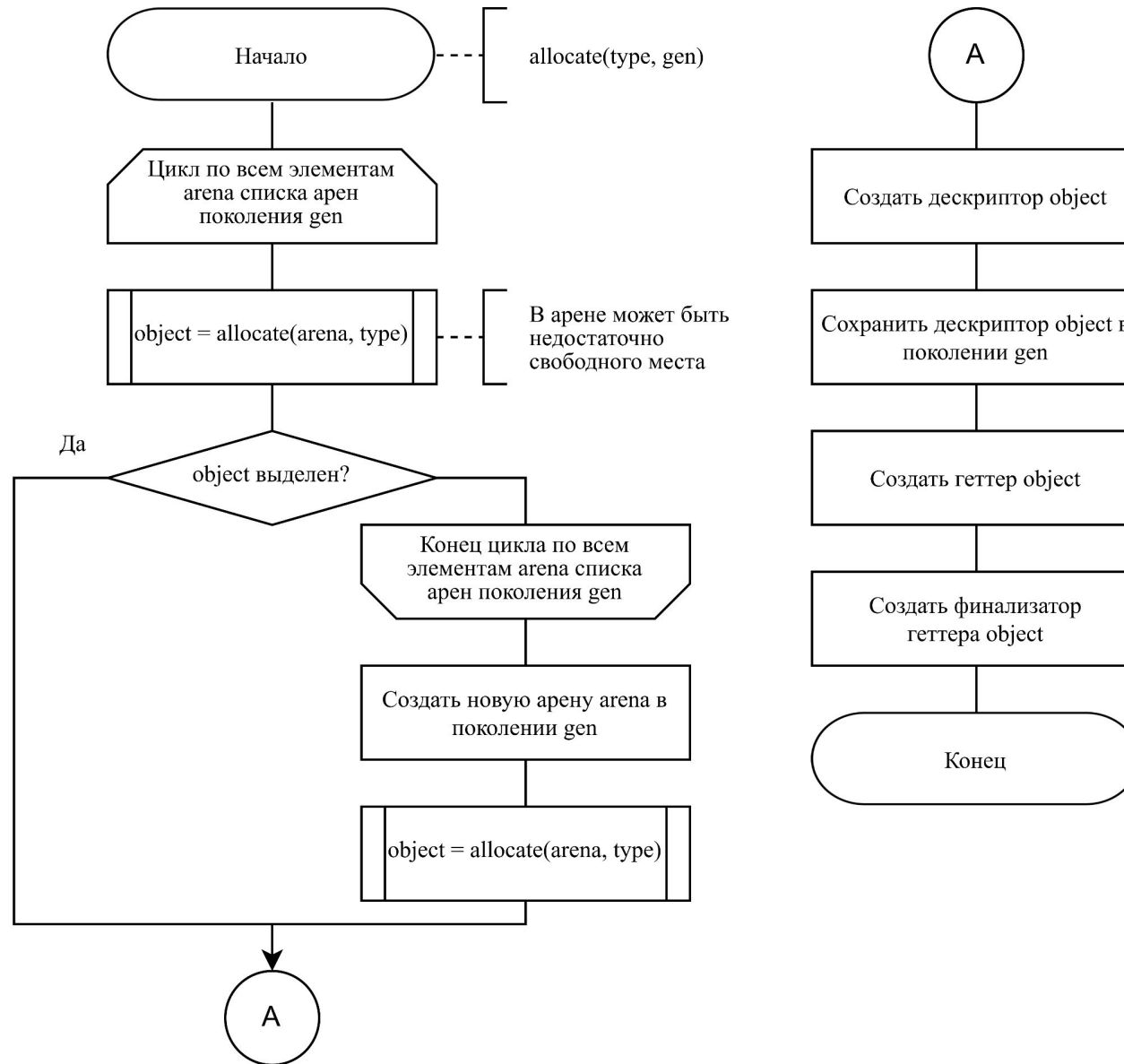
Классификация существующих решений

Язык программирования	Python	Java				JavaScript	C#	Golang
Сборщик мусора	По умолчанию	Serial	Parallel	Garbage-First	ZGC	По умолчанию	По умолчанию	По умолчанию
Разделение объектов на поколения	+	+	+	+	+	-	+	-
Отсутствие хранения вспомогательных данных в объектах	-	+	+	+	+	+	+	+
Использование конкурентной сборки мусора	+	+	+	+	+	-	+	+
Использование параллельной сборки мусора	-	-	+	+	+	-	+	+
Отсутствие остановки потоков мутатора на весь цикл сборки мусора	+	-	-	+	+	-	-	+
Количество остановок потоков мутатора за один цикл сборки мусора	1	1	1	2	1	1	1	2

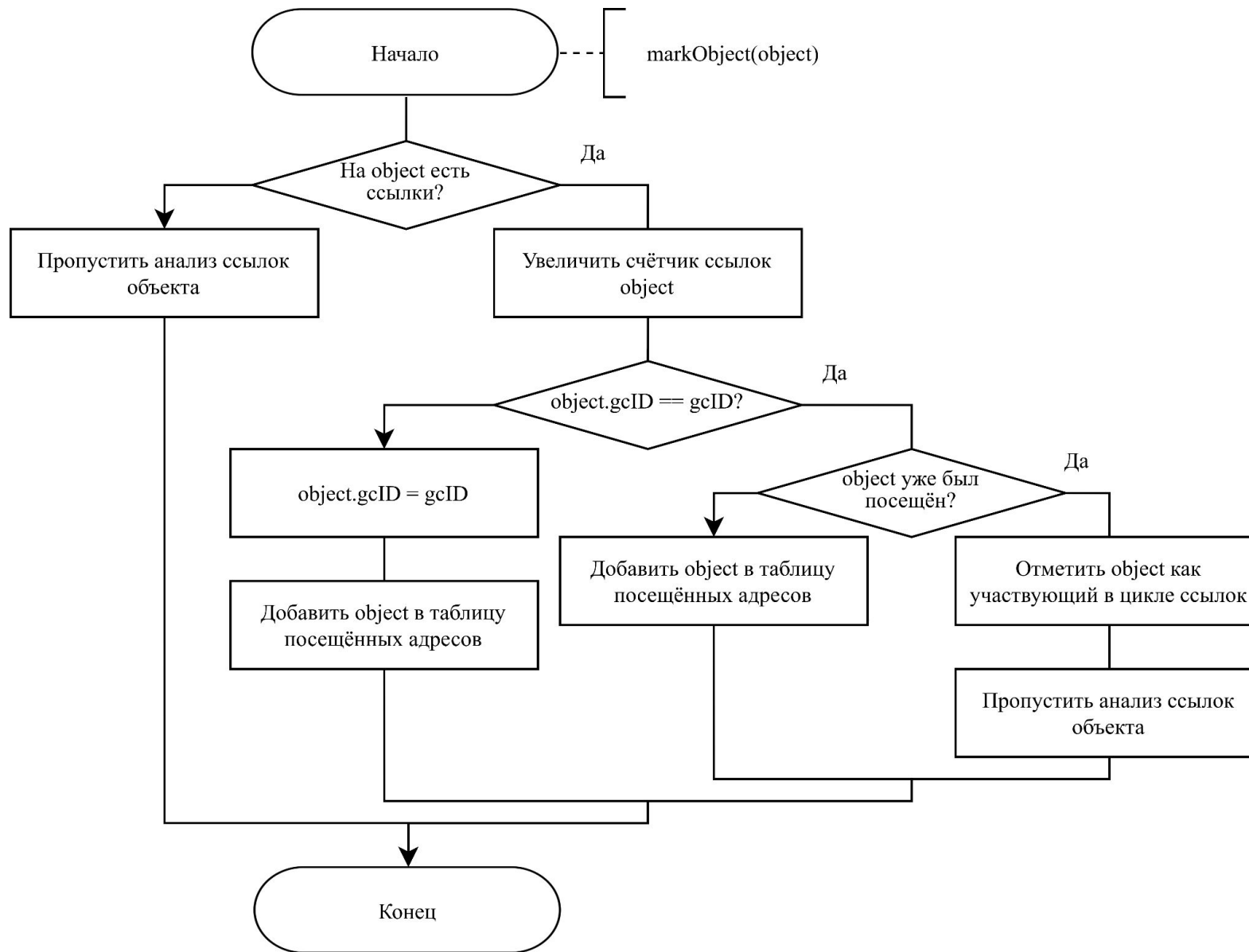
Метод автоматического управления памятью с гарантированным временем выполнения на основе подсчёта ссылок



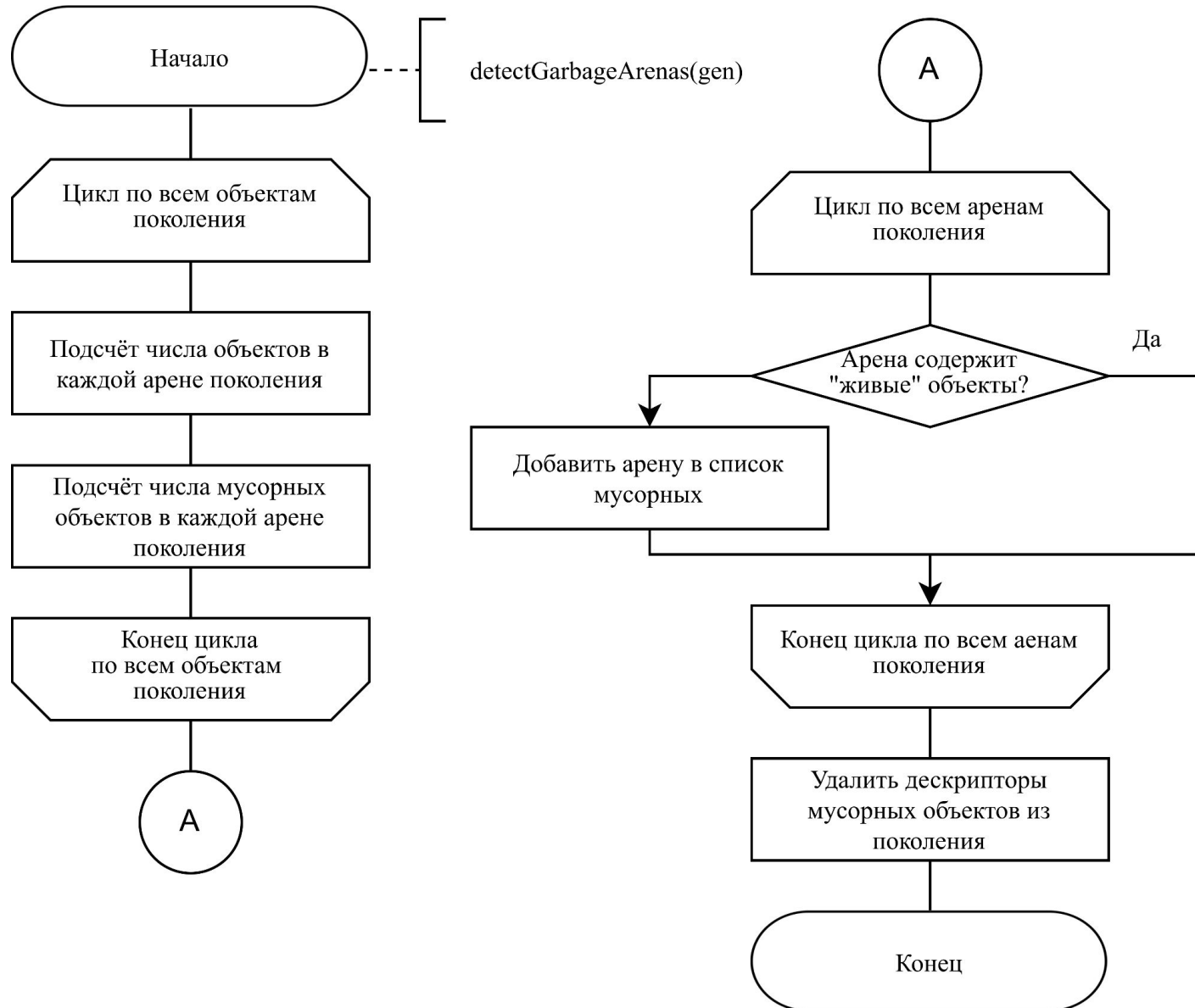
Алгоритм выделения памяти



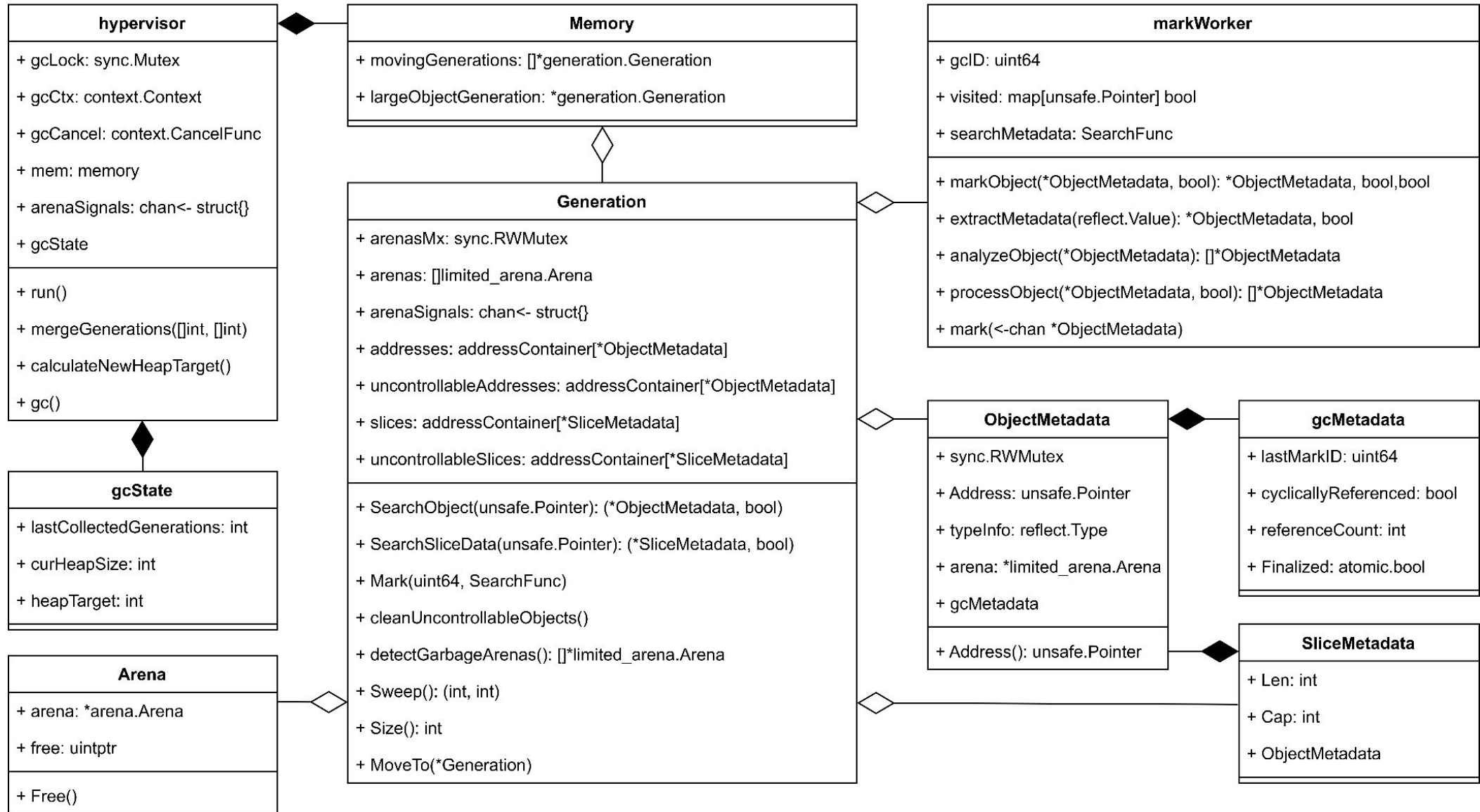
Алгоритм разметки



Алгоритм очистки



Структура ПО



Использование метода

```
> go get github.com/Inspirate789/alloc
```

```
package main

func main() {
    object := new(int)
    println(*object)
    *object = 7
    println(*object)
}
```

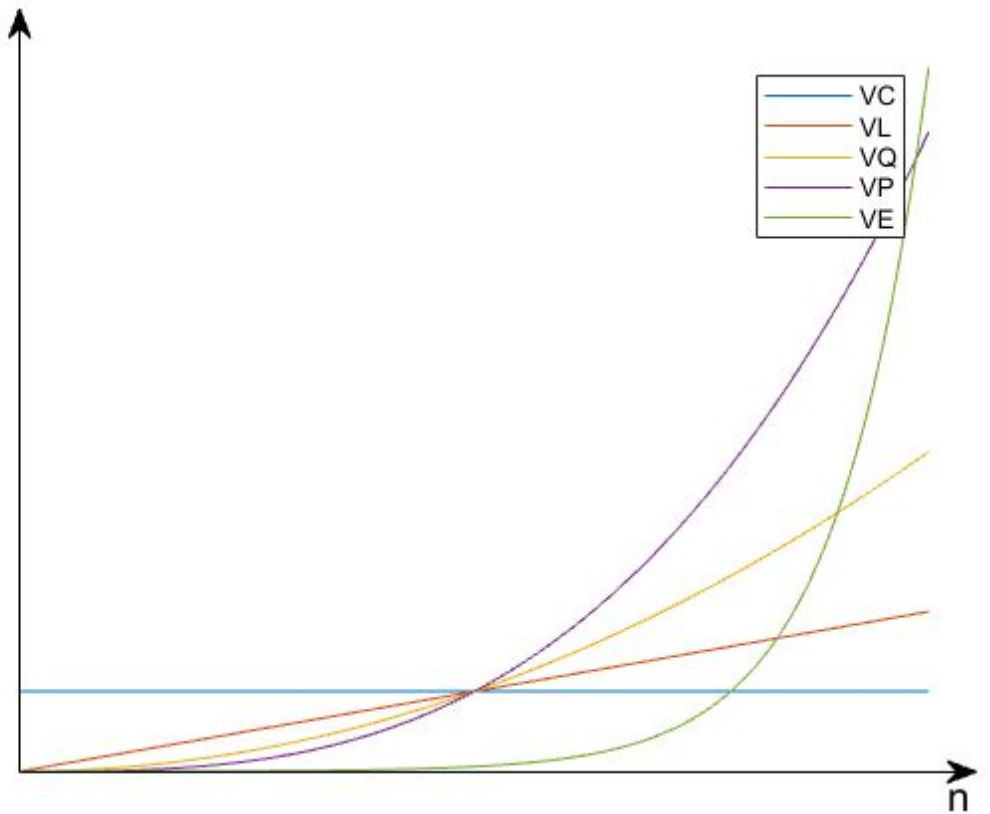
Встроенные функции
Golang

```
package main
import "github.com/Inspirate789/alloc"
func main() {
    object := alloc.New[int]()
    println(*object.Get())
    *object.Get() = 7
    println(*object.Get())
}
```

Реализованный метод

Классы алгоритмов по требованиям к дополнительной памяти

- **V0**: $V_t(D) = 0$ (не рассматривается)
- **VC**: $V_t^{\wedge}(n) = \text{const} \neq 0$;
- **VL**: $V_t^{\wedge}(n) = \Theta(n)$;
- **VQ**: $V_t^{\wedge}(n) = \Theta(n^2)$;
- **VP**: $V_t^{\wedge}(n) = \Theta(n^k)$, $k > 2$;
- **VE**: $V_t^{\wedge}(n) = \Theta(e^{\lambda n})$, $\lambda > 0$;

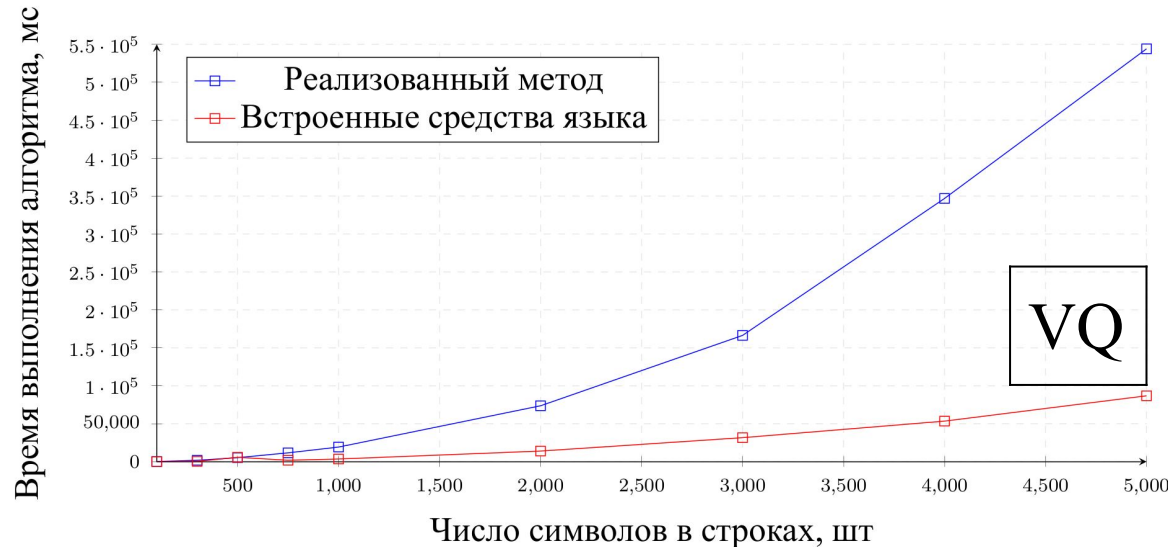
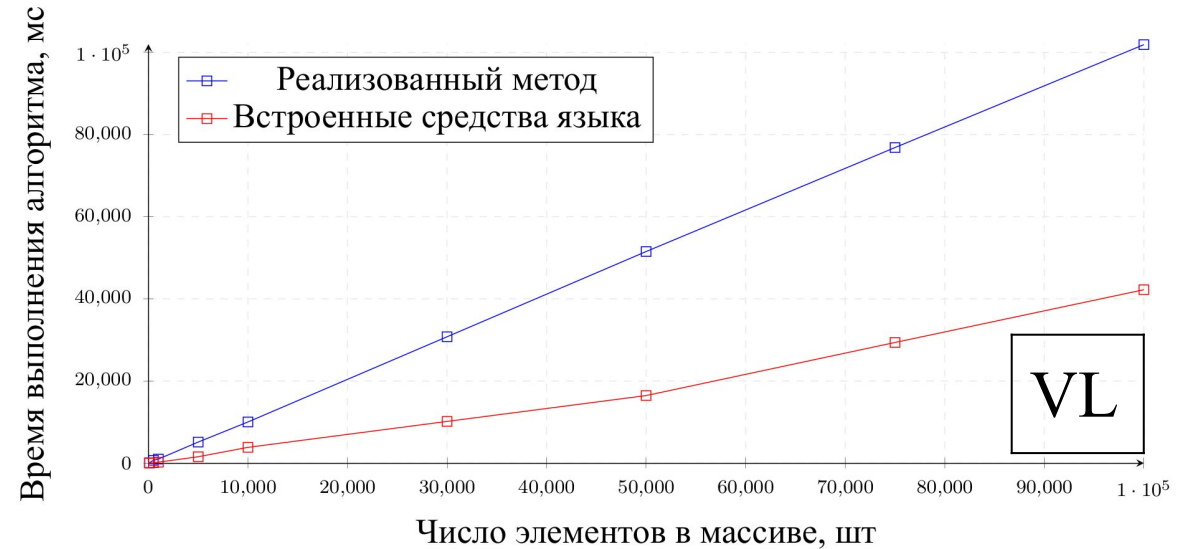
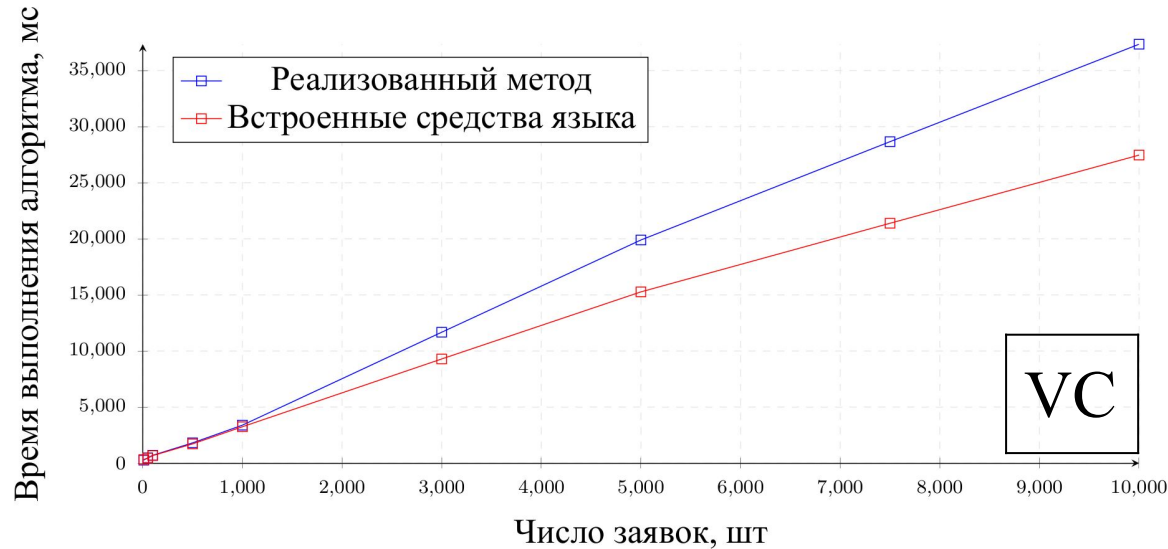


где D_n – вход алгоритма длины n ,

$V_t(D)$ – дополнительная память алгоритма,

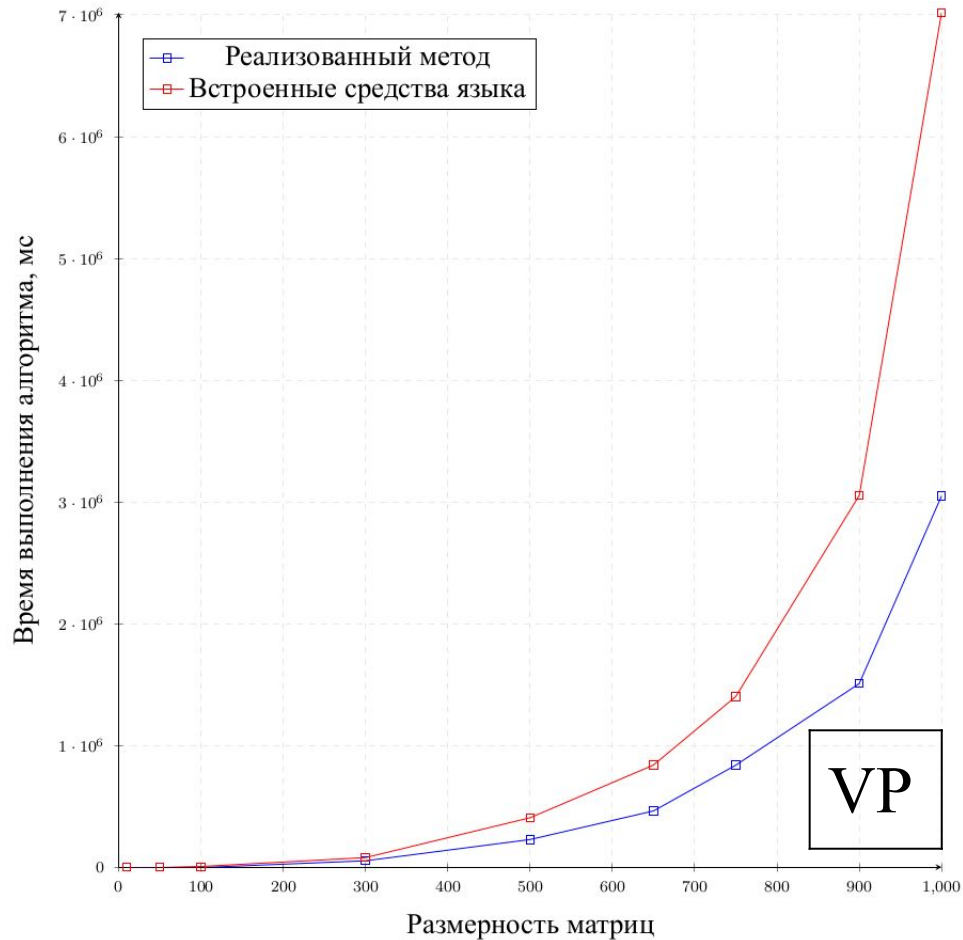
$V_t^{\wedge}(n) = \max V_t(D)$ – дополнительная память алгоритма в худшем случае для всех входов длины n .

Проигрыш по времени выполнения алгоритмов в сравнении со встроенными средствами Golang

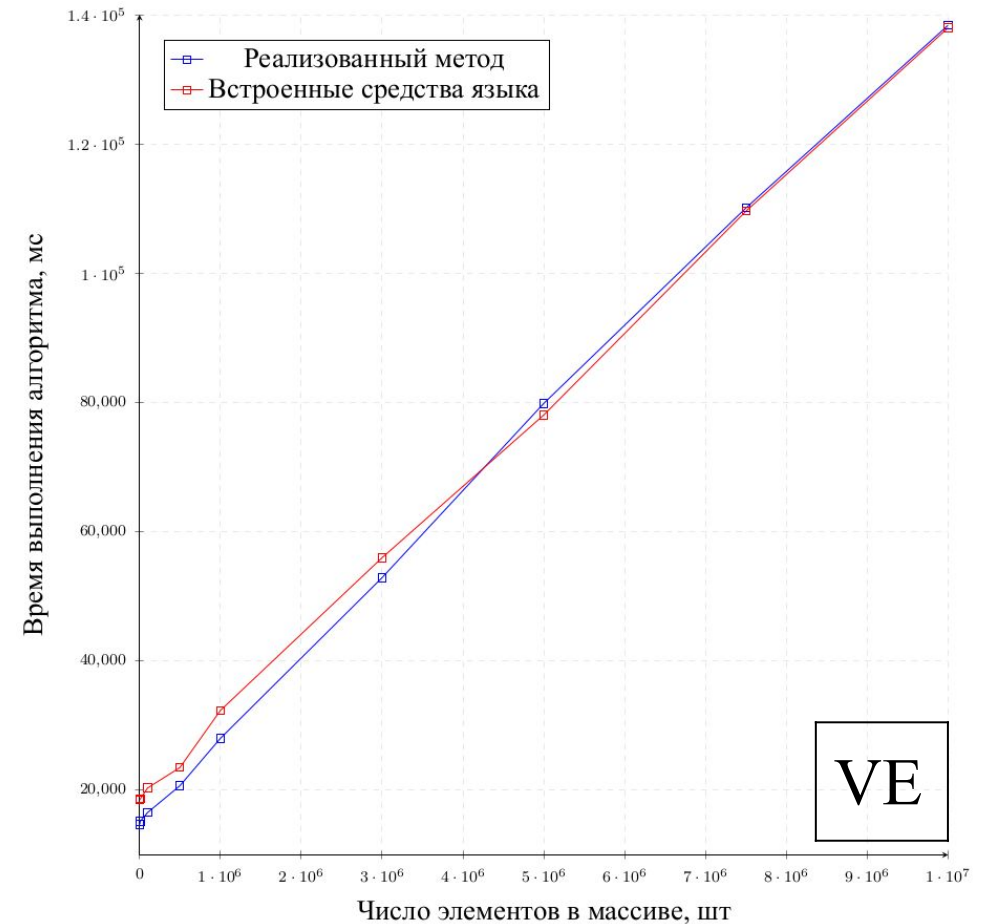


VC: фиксированная доп. память;
VL: доп. память линейно
зависит от длины входа;
VQ: доп. память квадратично
зависит от длины входа.

Выигрыш по времени выполнения алгоритмов в сравнении со встроенными средствами Golang



VP: доп. память полиномиально надквадратично зависит от длины входа



VE: доп. память экспоненциально зависит от длины входа

Заключение

Цель работы достигнута: был разработан и реализован метод автоматического управления памятью. Все поставленные задачи были выполнены:

- Проанализированы существующие методы распределения памяти в языках программирования с автоматической сборкой мусора.
- Проведена классификация алгоритмов по требованиям к дополнительной памяти: выделены классы **V0**, **VC**, **VL**, **VQ**, **VP** и **VE**.
- Спроектирован и реализован метод автоматического управления памятью.
- Сравнительный анализ выявил области применения реализованного метода: алгоритмы класса **VP** и **VE**.

Дальнейшее развитие

- Исследование стабильности менеджера памяти.
- Исследование фрагментации кучи при использовании метода.
- Внедрение метода во встроенный сборщик мусора языка Golang.

Публикации

- TODO