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

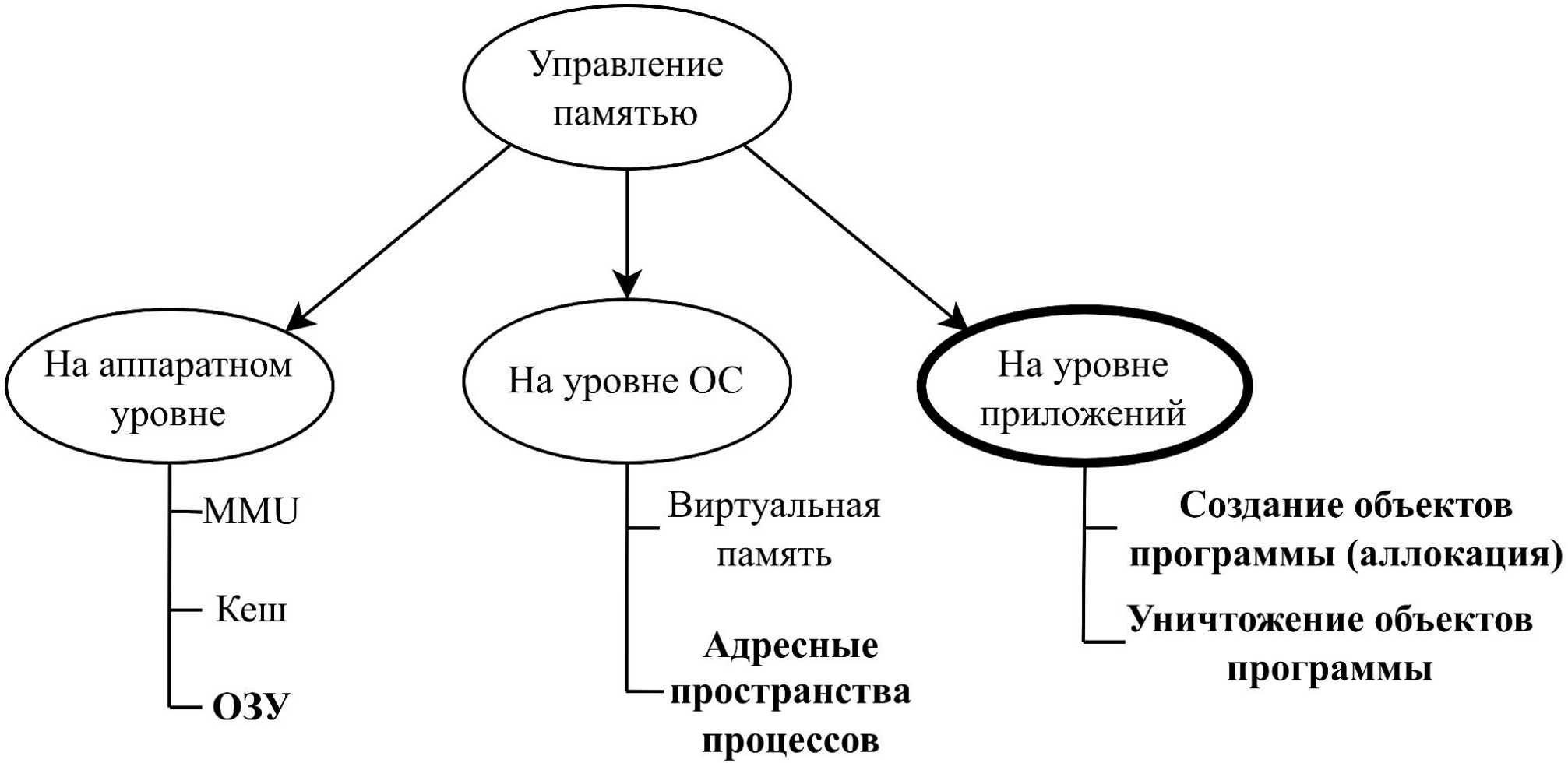
Цель и задачи

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. Сформировать рекомендации по применению 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++)

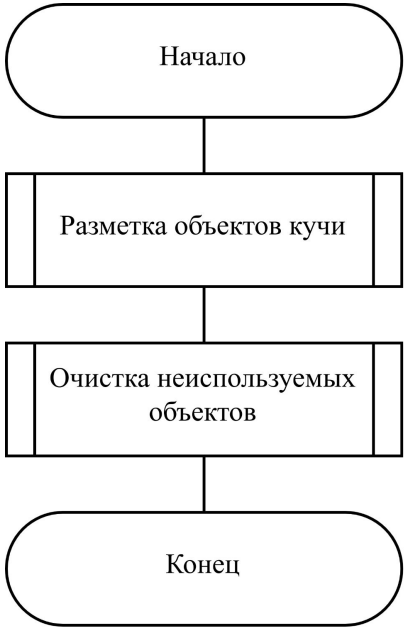
```
package main
func main() {
    a := new(int)
    // ...
    *a = 5
    // ...
}
```

Автоматическое управление памятью (Golang)

Автоматическая переработка динамически выделяемой памяти

Трассирующая сборка мусора

mark-sweep



mark-compact

Подсчёт ссылок

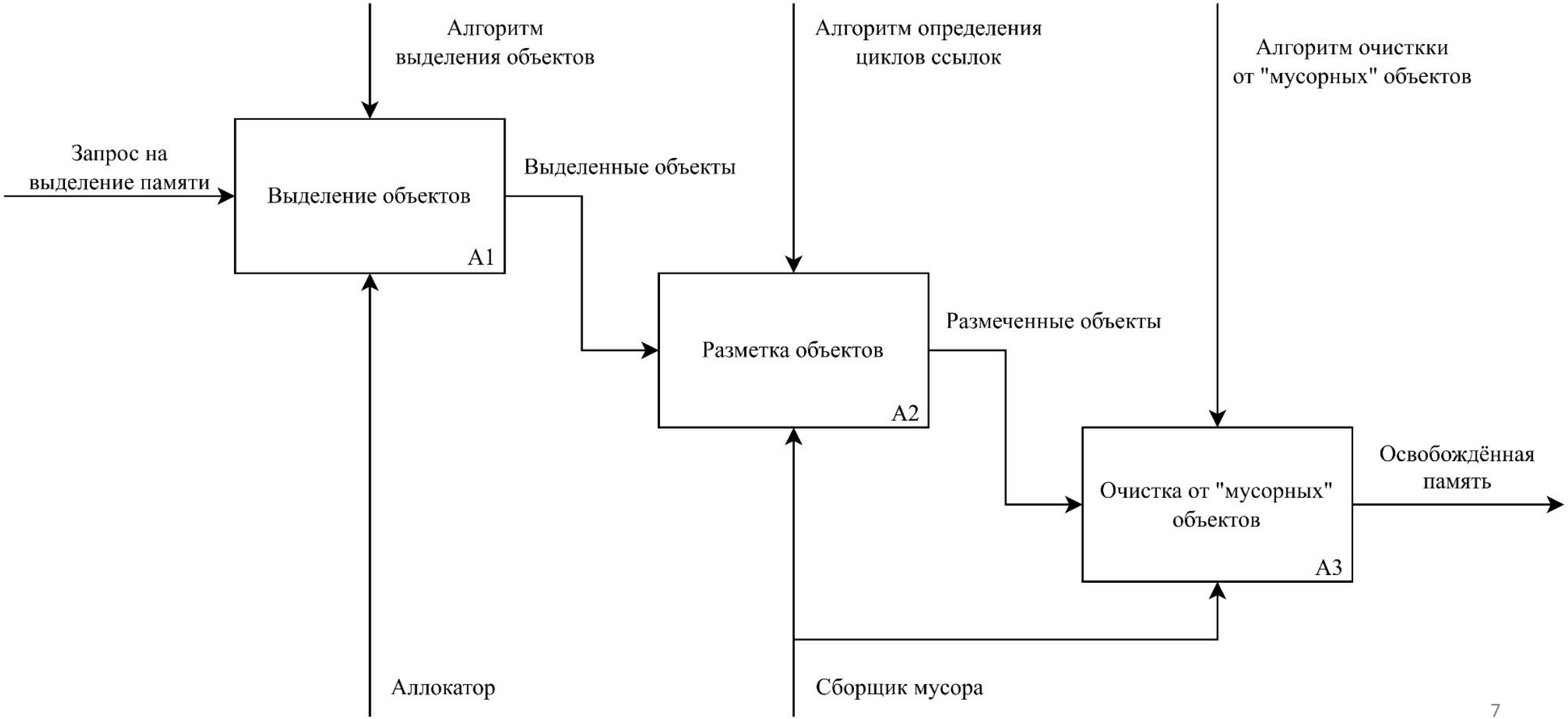
\+ более равномерное распределение накладных расходов;
\+ устойчивость к высоким нагрузкам;
\+ масштабируемость по размеру кучи;
\+ не требует среды выполнения языка;

\- накладные расходы на чтение/запись;
\- атомарность операций;
\- не выявляются циклы ссылок.

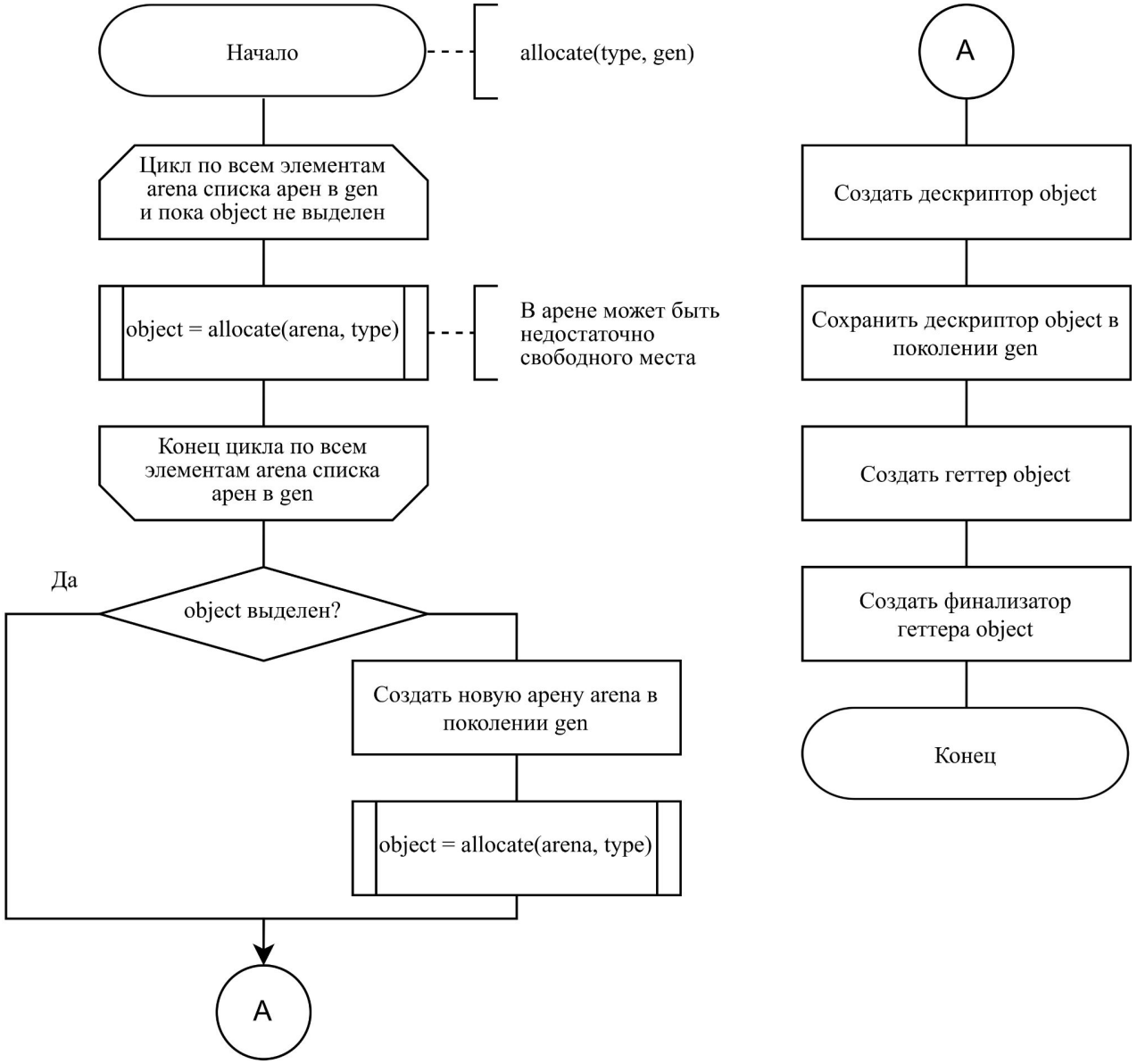
Классификация существующих решений

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**
Возможность встраивания модификаций менеджера памяти	+/-	+/-	+/-	+/-	+/-	+/-	+/-	+

Метод автоматического управления памятью на основе подсчёта ссылок



Алгоритм выделения объектов

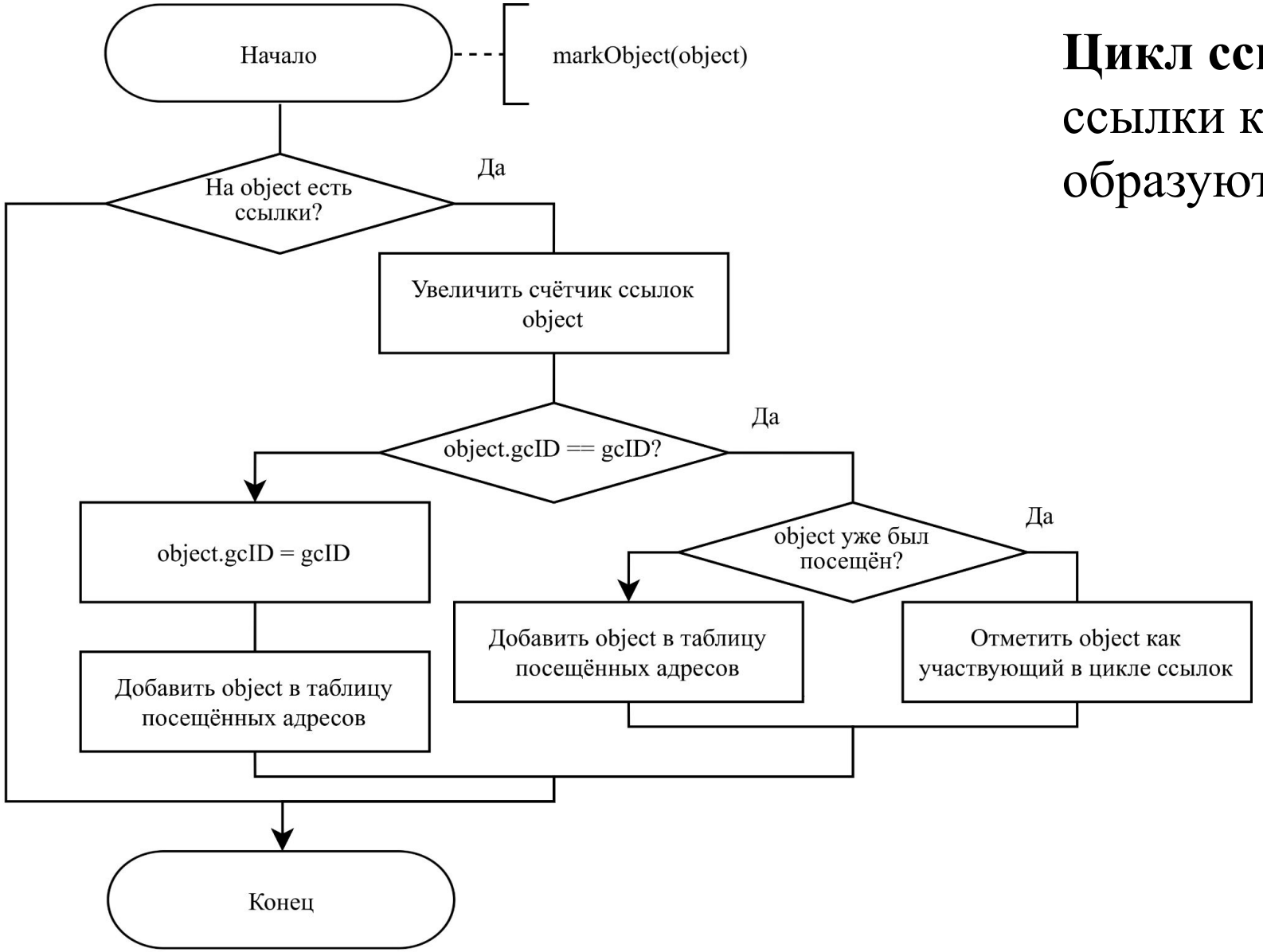


Арена – область кучи фиксированного размера.

Геттер – функция, позволяющая получить доступ к объекту.

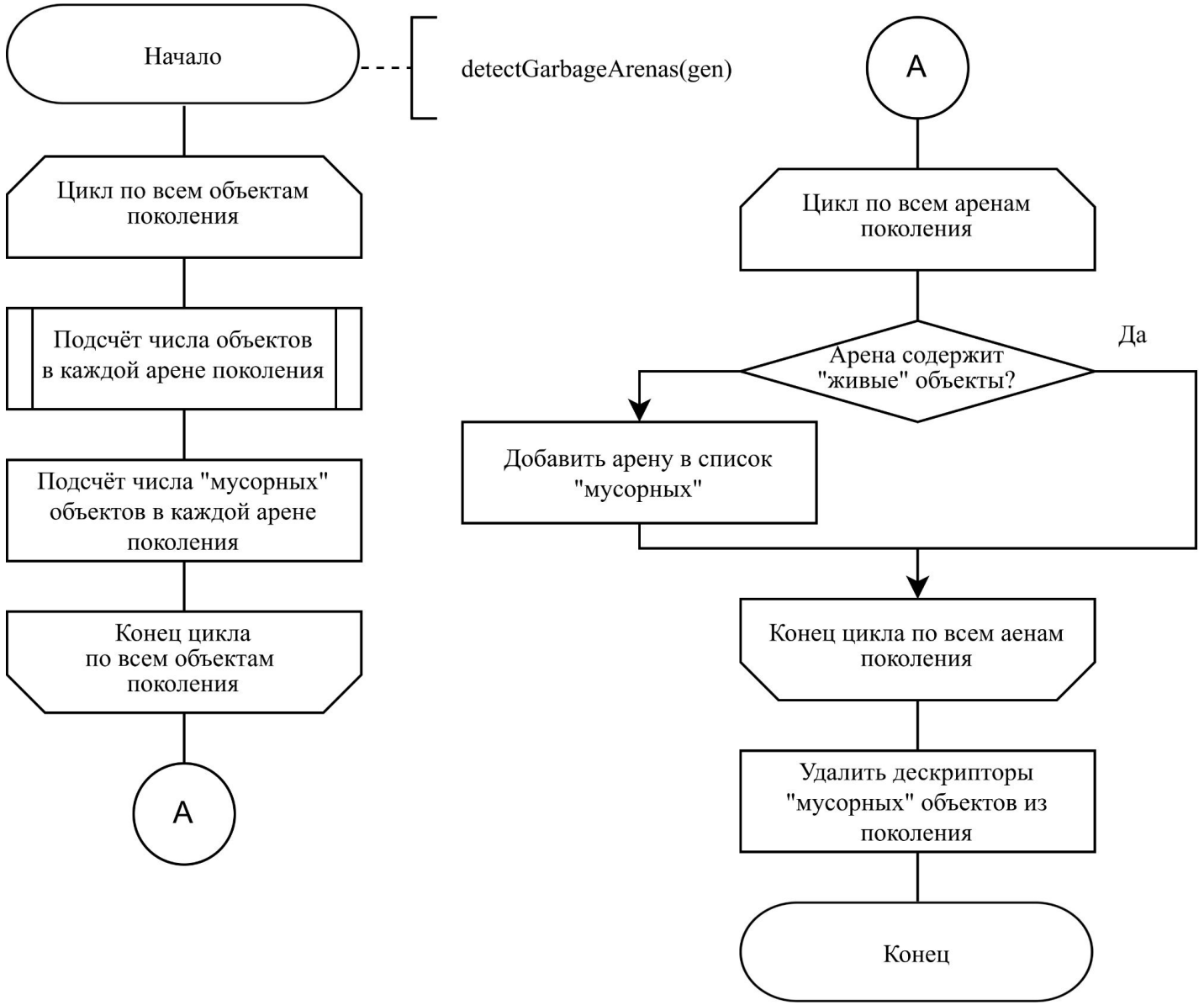
Финализатор – функция, которая запускается, когда сборщик мусора определяет, что рассматриваемый объект недоступен в программе.

Алгоритм определения циклов ссылок



Цикл ссылок – группа объектов, ссылки которых друг на друга образуют замкнутый цикл.

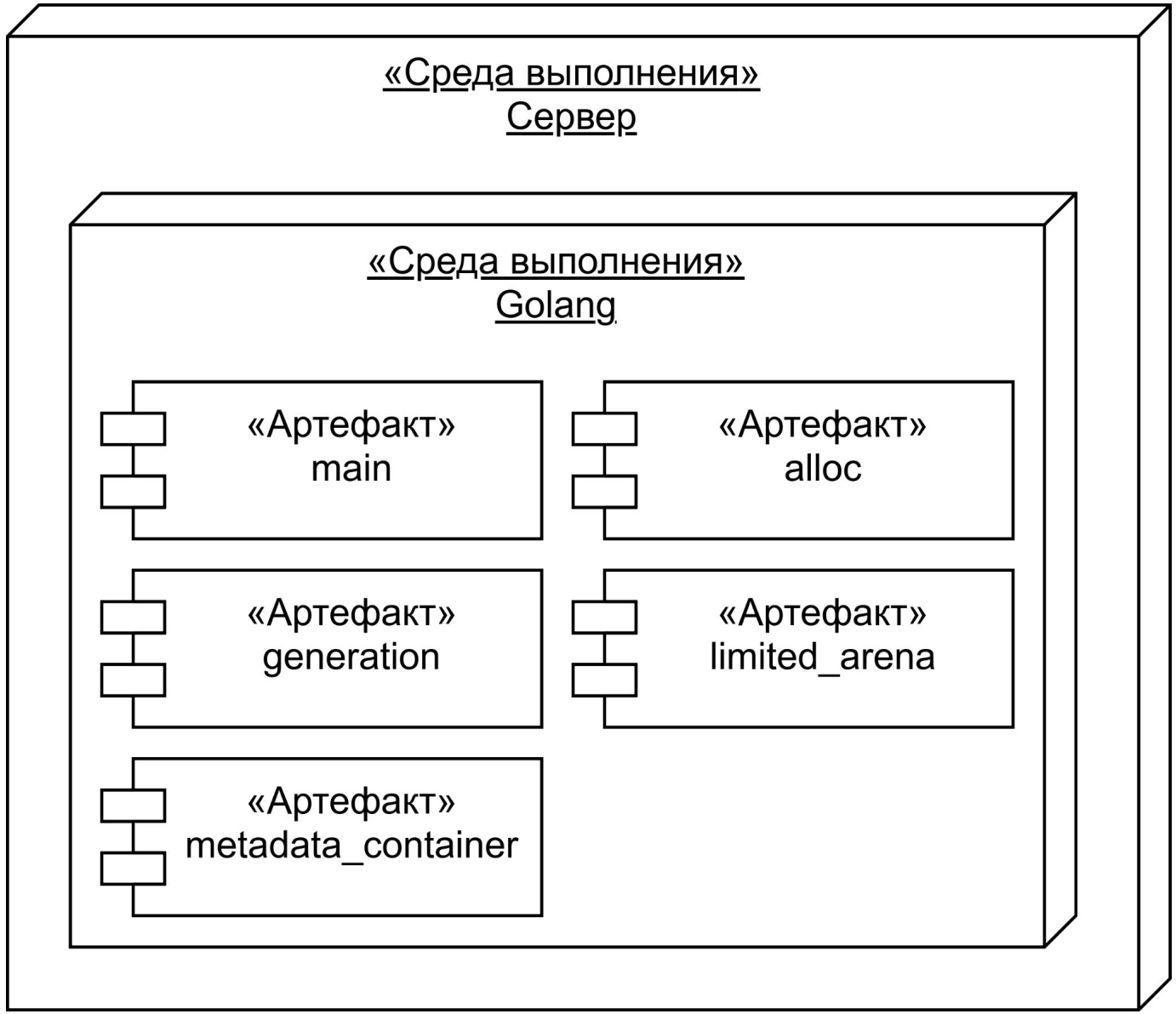
Алгоритм очистки от «мусорных» объектов



«Живой» объект – объект, который **достижим** из основной программы.

«Мусорный» объект – объект, который **не достижим** из основной программы.

Диаграмма развёртывания ПО

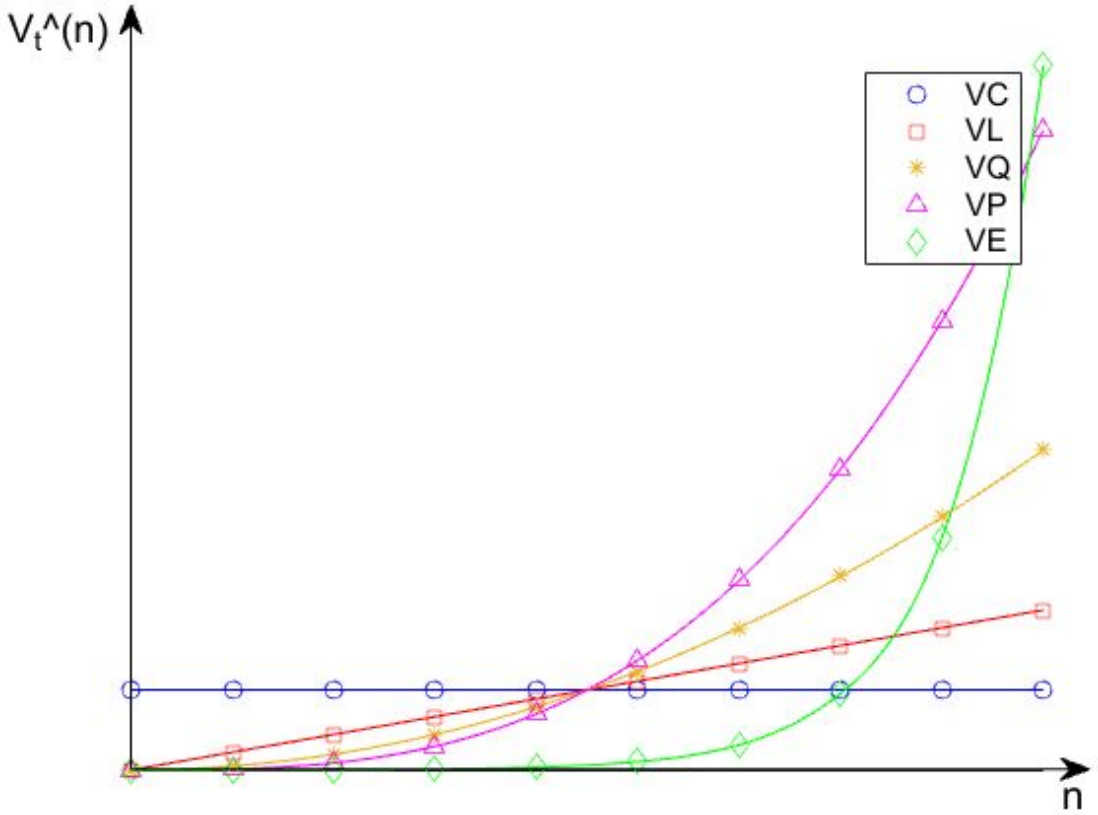


Дескриптор объекта

Поле	Размер в байтах
Мьютекс	24
Адрес объекта	8
Метаданные о типе	16
Указатель на арену памяти	8
Идентификатор последней разметки	8
Флаг участия в цикле ссылок	1
Счётчик ссылок	8
Флаг финализации	4
Итого (с учётом выравнивания)	**88**

Классы алгоритмов по требованиям к дополнительной памяти

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

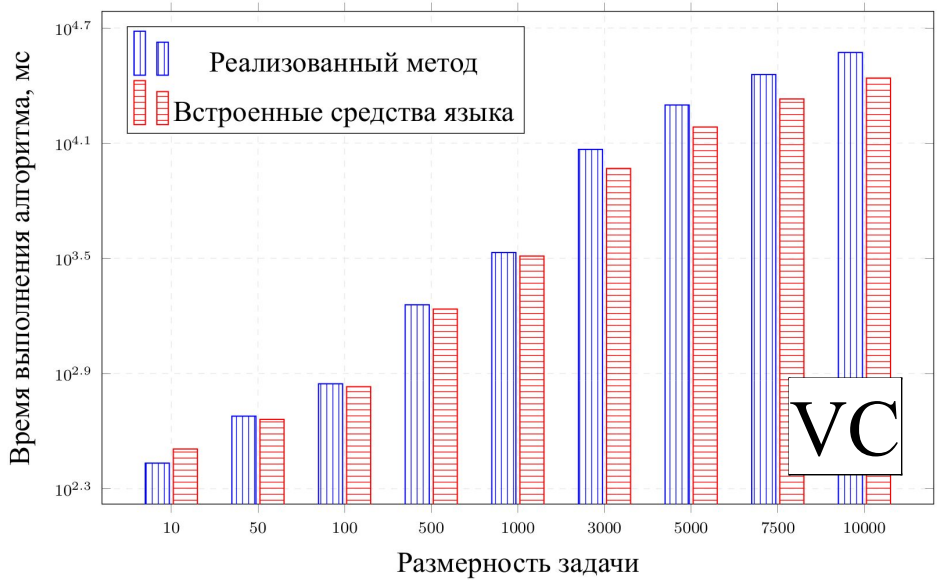


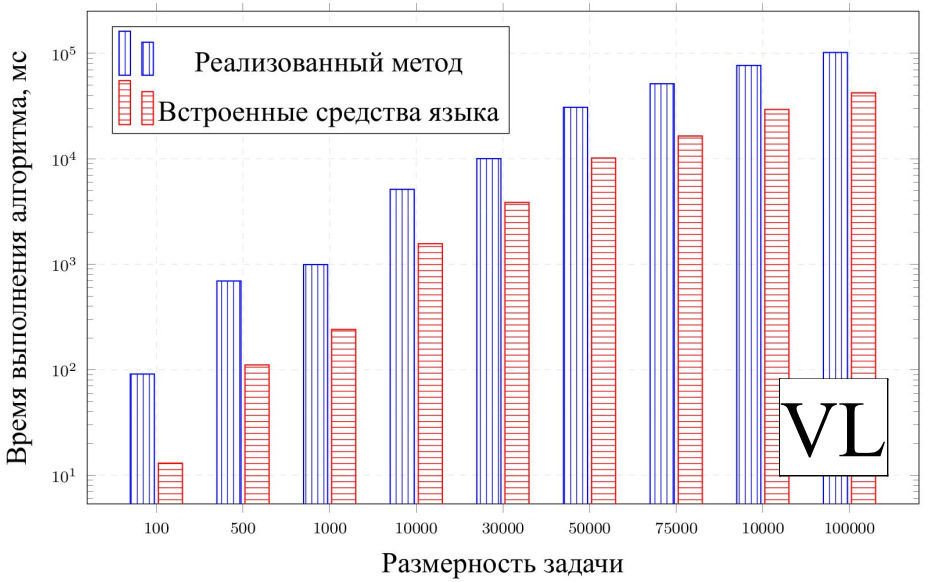
где D_n – вход алгоритма длины n,

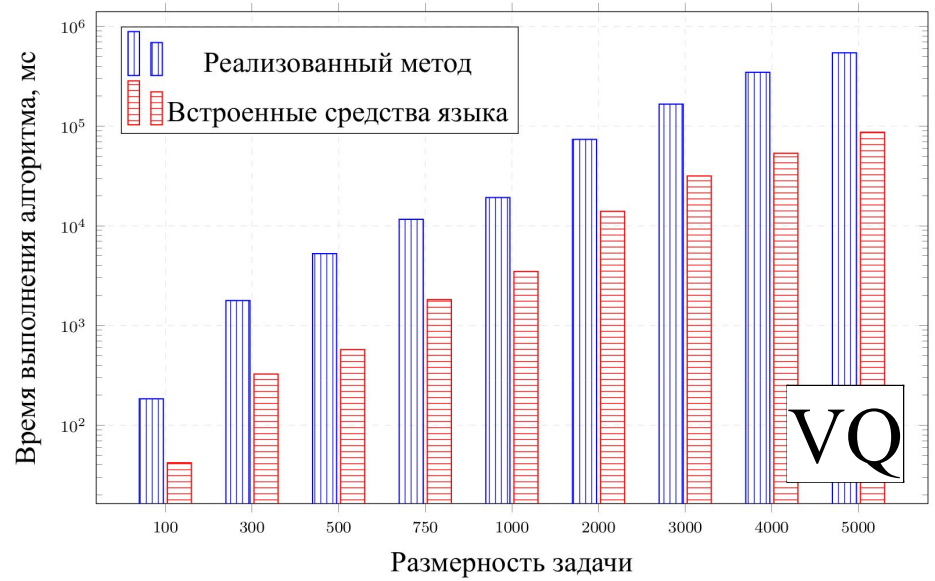
$V_t(D)$ – дополнительная память алгоритма,

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

Проигрыш по времени выполнения алгоритмов в сравнении со встроенными средствами Golang

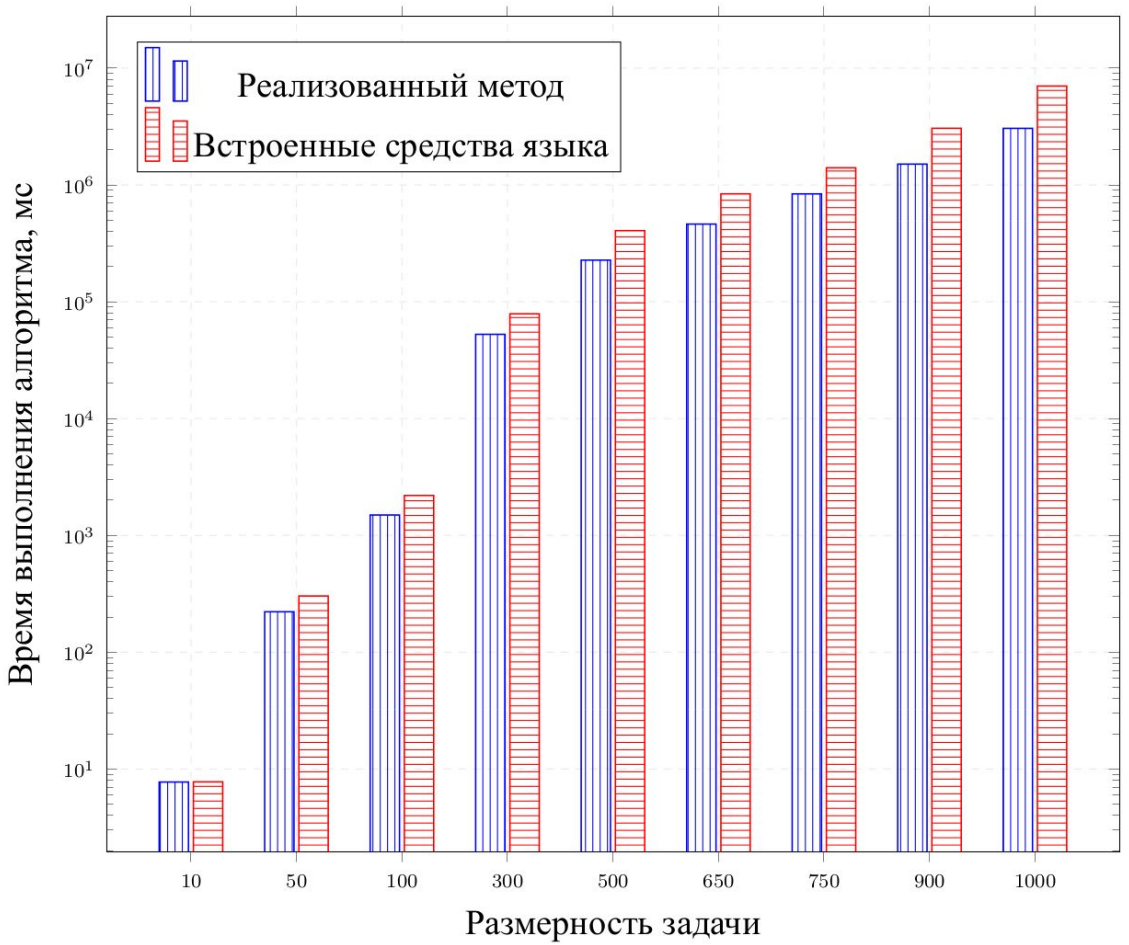


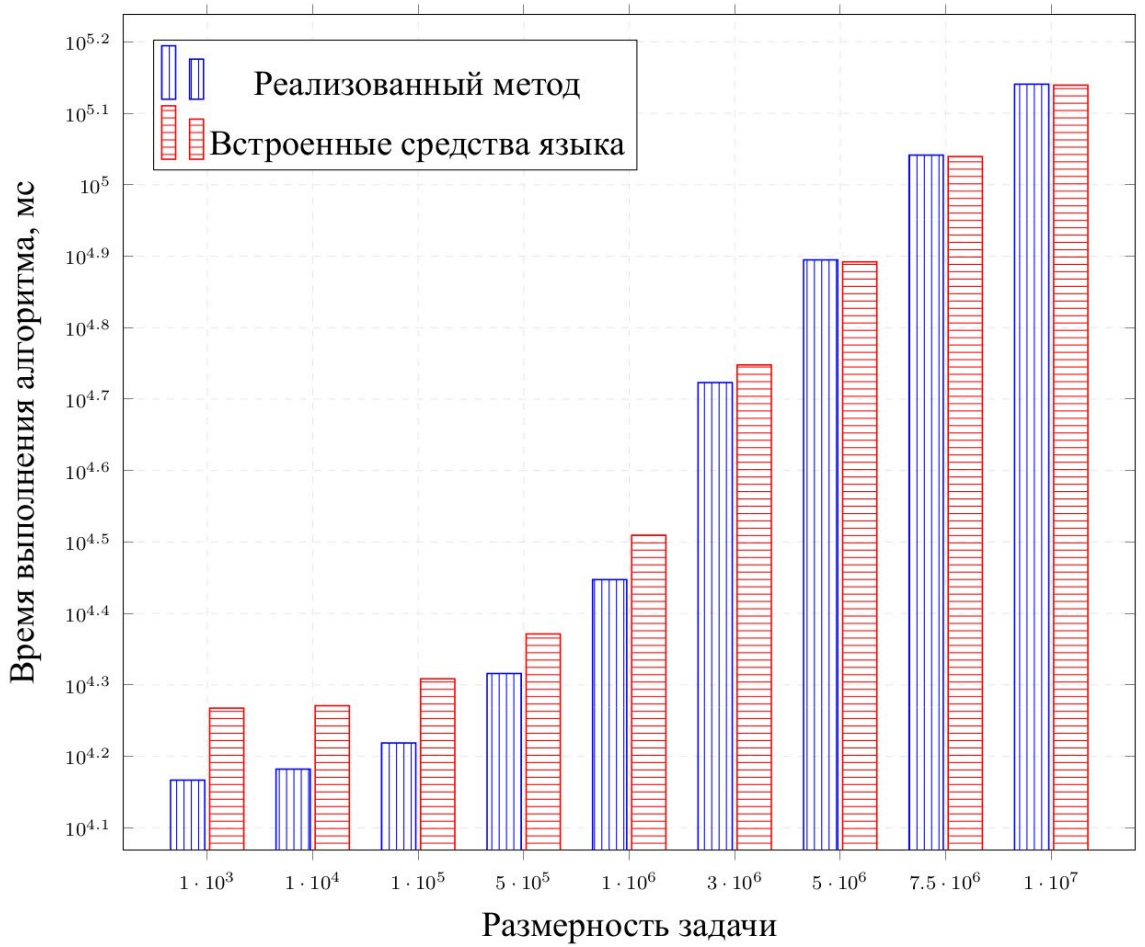




VC: фиксированная доп. память;
VL: доп. память линейно зависит от длины входа;
VQ: доп. память квадратично зависит от длины входа.

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.

1. Проанализированы существующие методы распределения памяти в языках программирования с автоматической сборкой мусора.
2. Спроектирован метод автоматического управления памятью.
3. Спроектированный метод реализован в виде подключаемой библиотеки.
4. Сформированы рекомендации по применению реализованного метода: алгоритмы классов **VP** и **VE**.

Дальнейшее развитие

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

Результаты работы поданы для участия в международной научной конференции «Математические методы в технике и технологиях ММТТ-37».