ИСП РАН

Методы декомпозиции систем и моделирования окружения программных модулей для верификации Си-программ

Диссертация на соискание ученой степени

кандидата физико-математических наук

05.13.11 – математическое и программное обеспечение

вычислительных машин, комплексов и компьютерных сетей

Захаров Илья Сергеевич

Научный руководитель:
д. ф.-м. н., проф. Петренко Александр Константинович

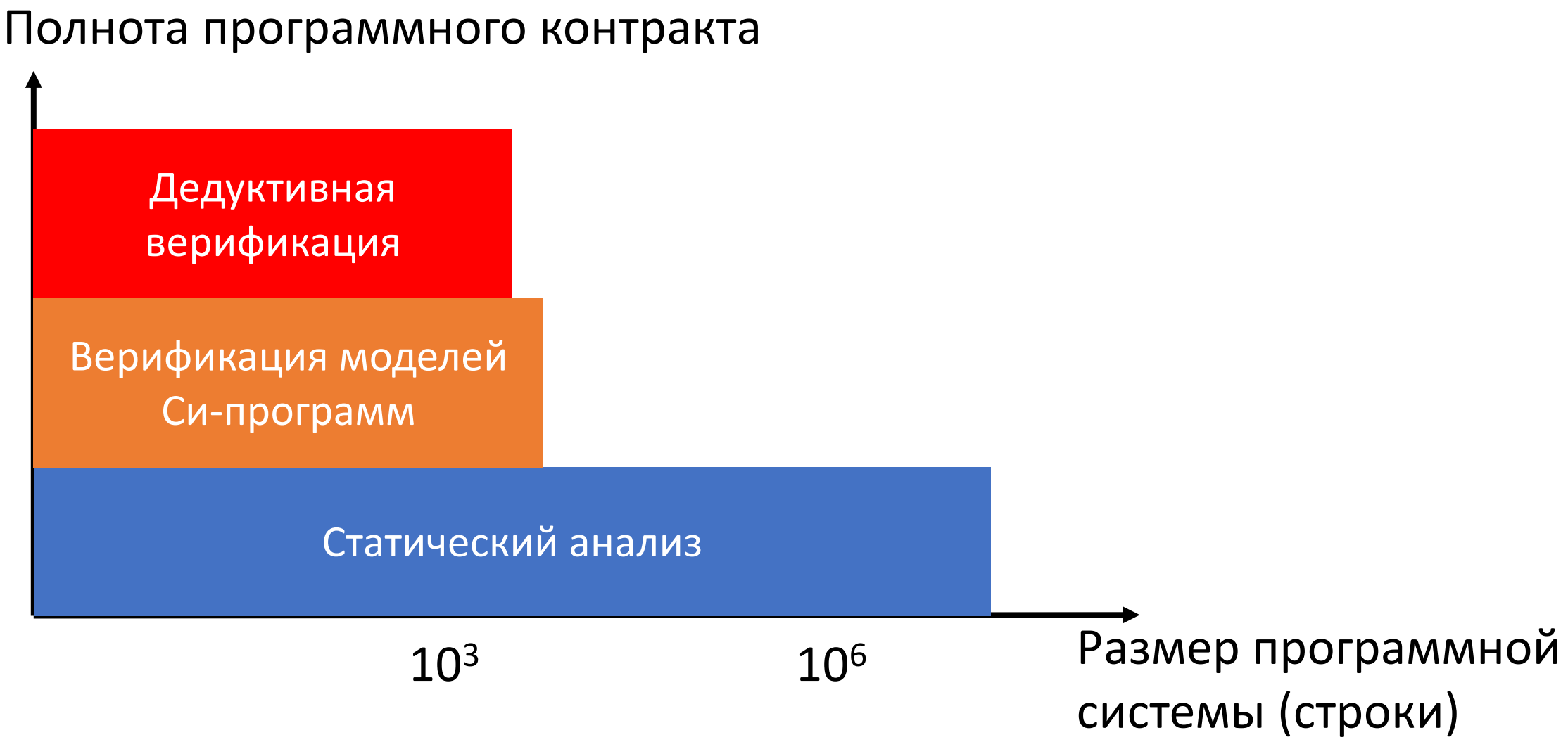
23 мая 2019 года

Крупные программные системы ответственного назначения на языке программирования Си

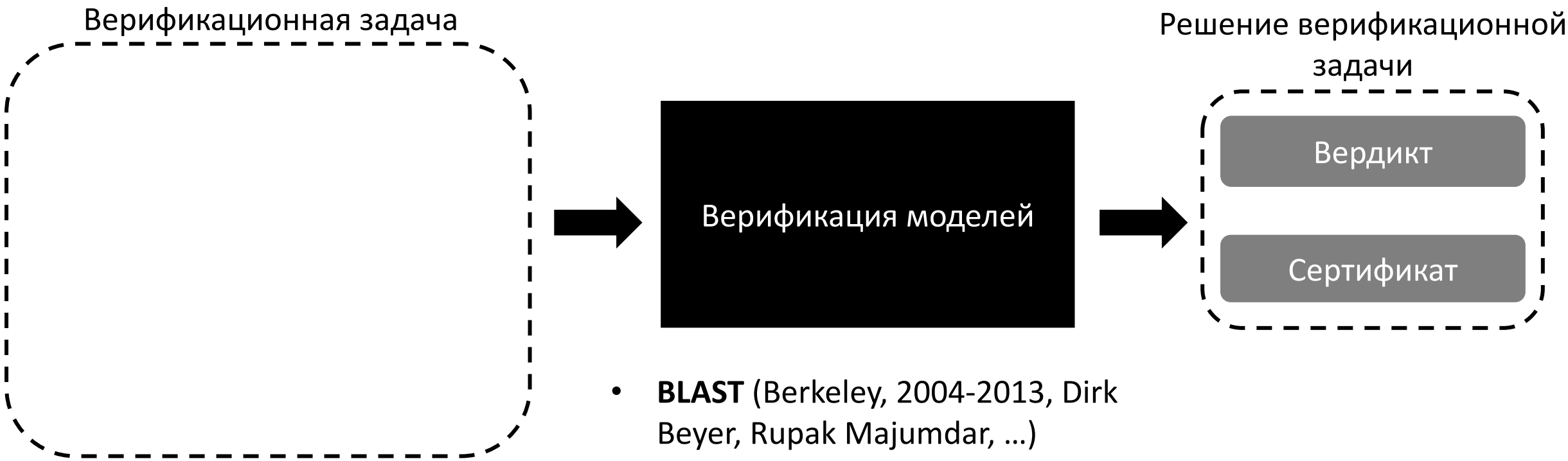
- Операционные системы
- Программное обеспечение встраиваемых систем
- Системы управления базами данных
- Веб-серверы

…

Верификация исходного кода без выполнения

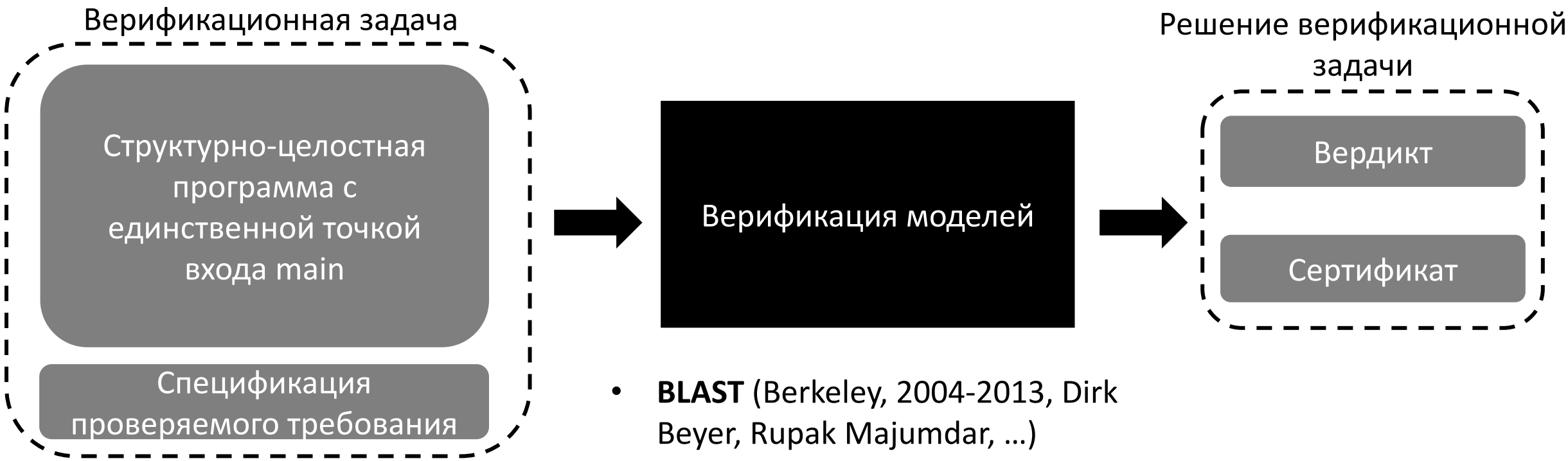


Верификация моделей Си-программ



- **BLAST** (Berkeley, 2004-2013, Dirk Beyer, Rupak Majumdar, …)
- **CPAchecker** (LMU, 2011-н.в., Dirk Beyer, Philipp Wendler, …)
- **CBMC** (Oxford, 2010-н.в., Edmund Clarke, Daniel Kroening, …)
- **SLAM**, **SLAM2**, **Corral** (Microsoft, 2003-н.в., Sriram Rajamani, Tom Ball, …)

Верификация моделей Си-программ



- **BLAST** (Berkeley, 2004-2013, Dirk Beyer, Rupak Majumdar, …)
- **CPAchecker** (LMU, 2011-н.в., Dirk Beyer, Philipp Wendler, …)
- **CBMC** (Oxford, 2010-н.в., Edmund Clarke, Daniel Kroening, …)
- **SLAM**, **SLAM2**, **Corral** (Microsoft, 2003-2017, Sriram Rajamani, Tom Ball, …)

Верификация моделей модулей программных систем

Цель

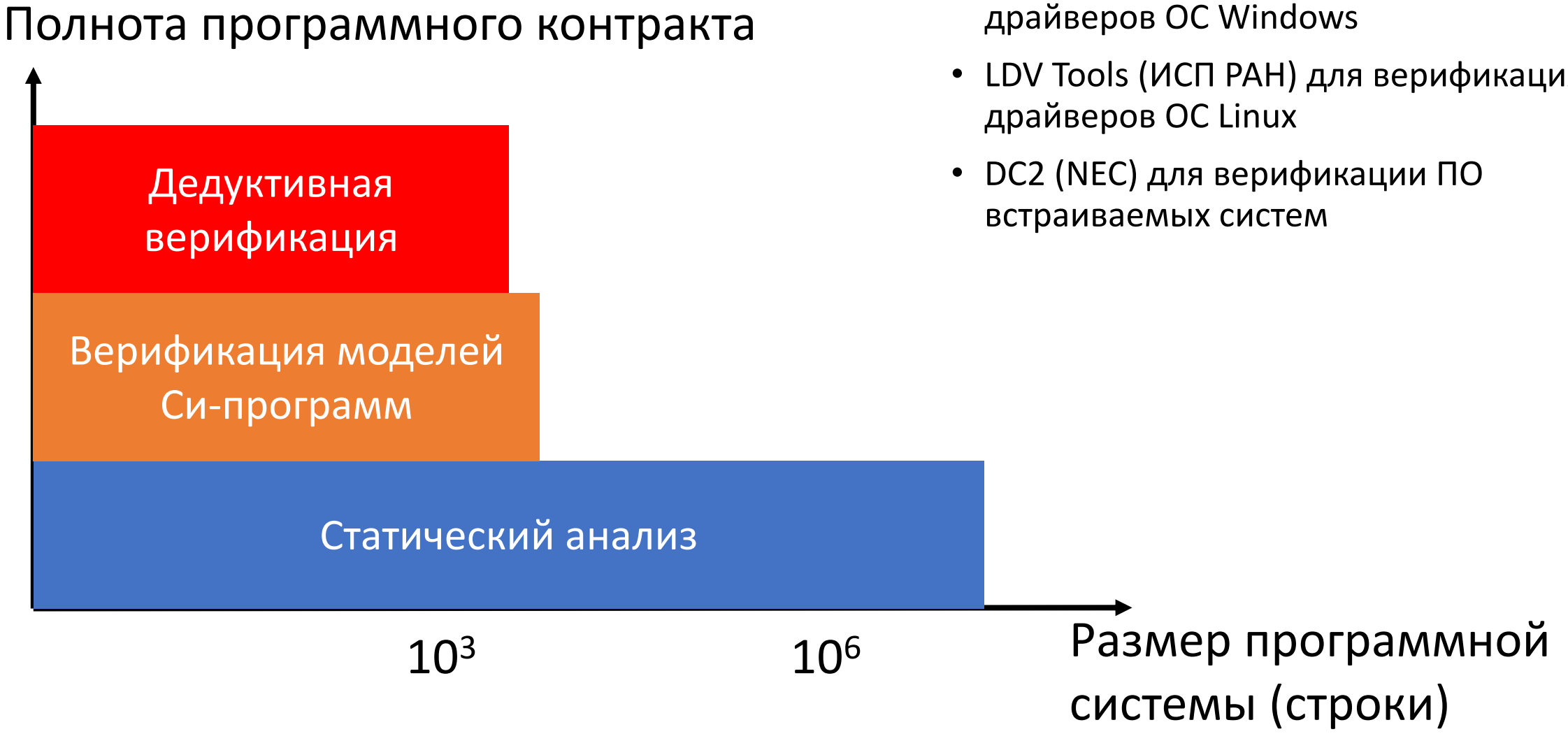
Развитие методов верификации крупных программных систем на языке программирования Си с использованием существующих инструментов верификации моделей программ для сокращения трудоемкости и сроков верификации при помощи автоматизации:

- декомпозиции системы на модули
- синтеза моделей их окружения

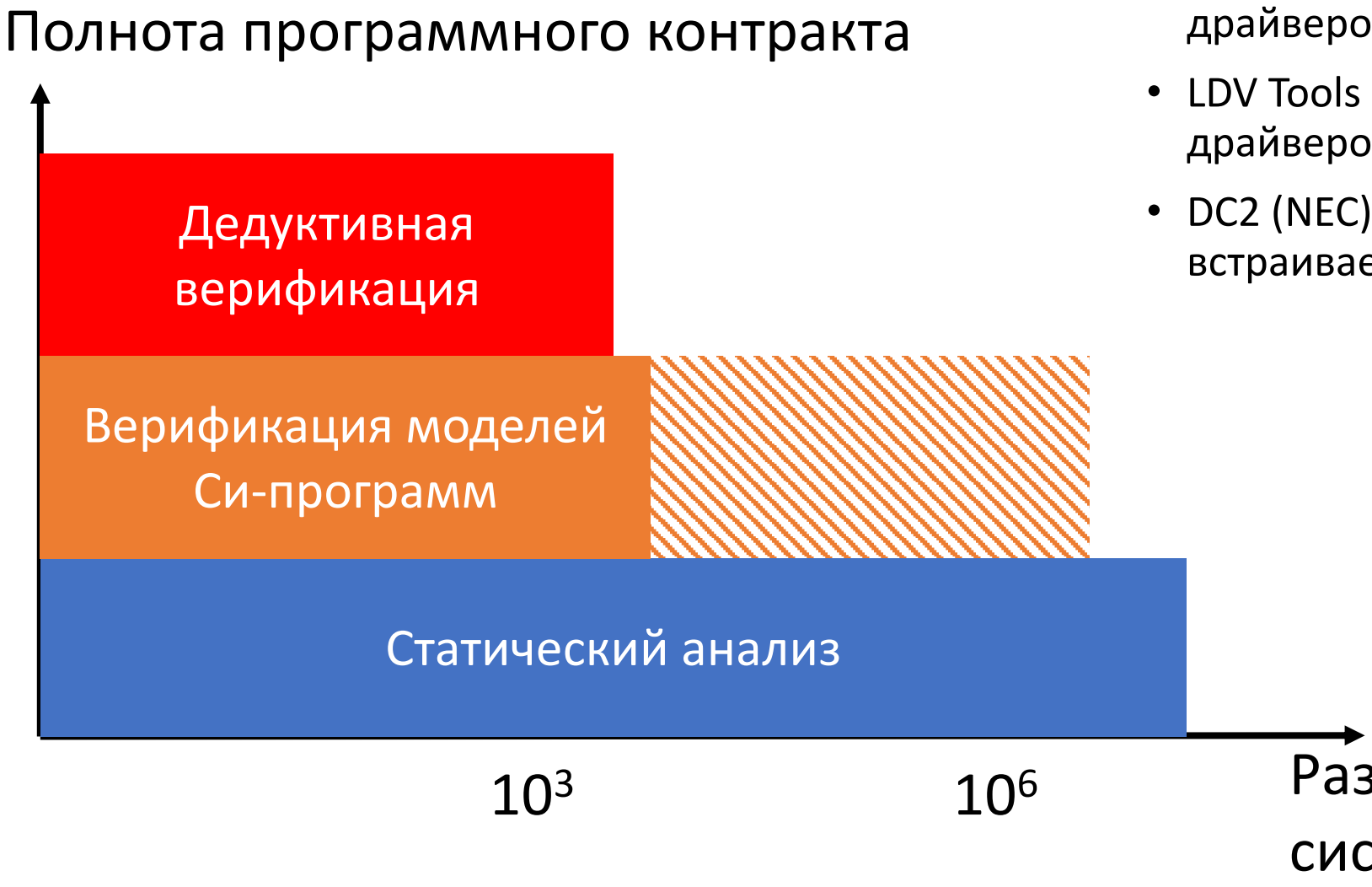
Развитие методов верификации крупных программных систем

Существующие решения:

- SDV (Microsoft) для верификации драйверов ОС Windows
- LDV Tools (ИСП РАН) для верификация драйверов ОС Linux
- DC2 (NEC) для верификации ПО встраиваемых систем



Развитие методов верификации крупных программных систем



Существующие решения:

- SDV (Microsoft) для верификации драйверов ОС Windows
- LDV Tools (ИСП РАН) для верификация драйверов ОС Linux
- DC2 (NEC) для верификации ПО встраиваемых систем

Задачи

- Разработать метод настраиваемой автоматизированной декомпозиции Си-программ на модули
- Разработать метод автоматизированного синтеза моделей окружения модулей, позволяющий адаптировать процесс синтеза для проверки разных видов требований и программ
- Разработать архитектуру системы верификации Си-программ, выполняющей генерацию и решение набора верификационных задач при помощи автоматизированной декомпозиции системы на модули и синтеза моделей окружения модулей

Выносимые на защиту положения

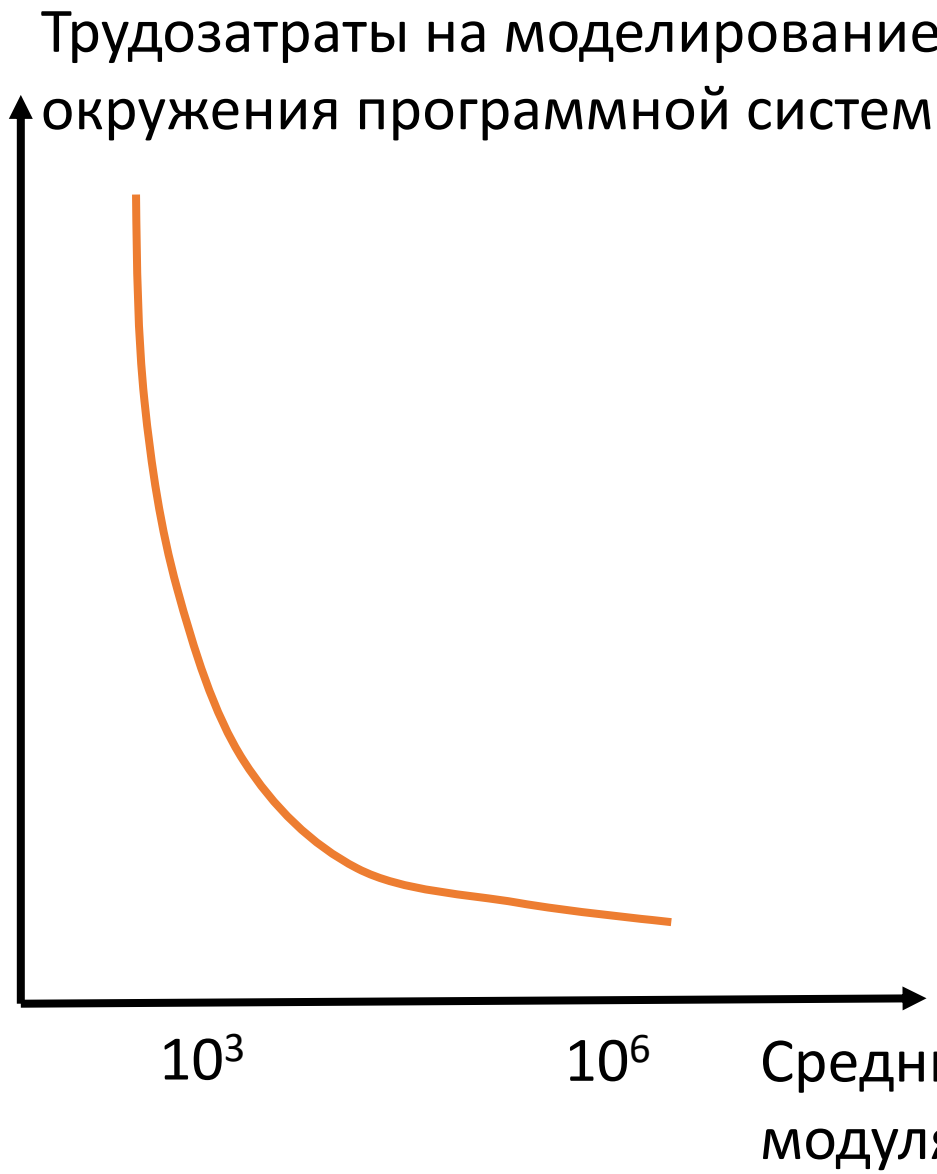
- **Метод** автоматизированной декомпозиции Си-программ на модули
- **Метод** спецификации моделей окружения на основе композиции систем переходов
- **Метод** автоматизированного синтеза моделей окружения модулей, позволяющий адаптировать процесс синтеза для проверки разных видов требований и программ

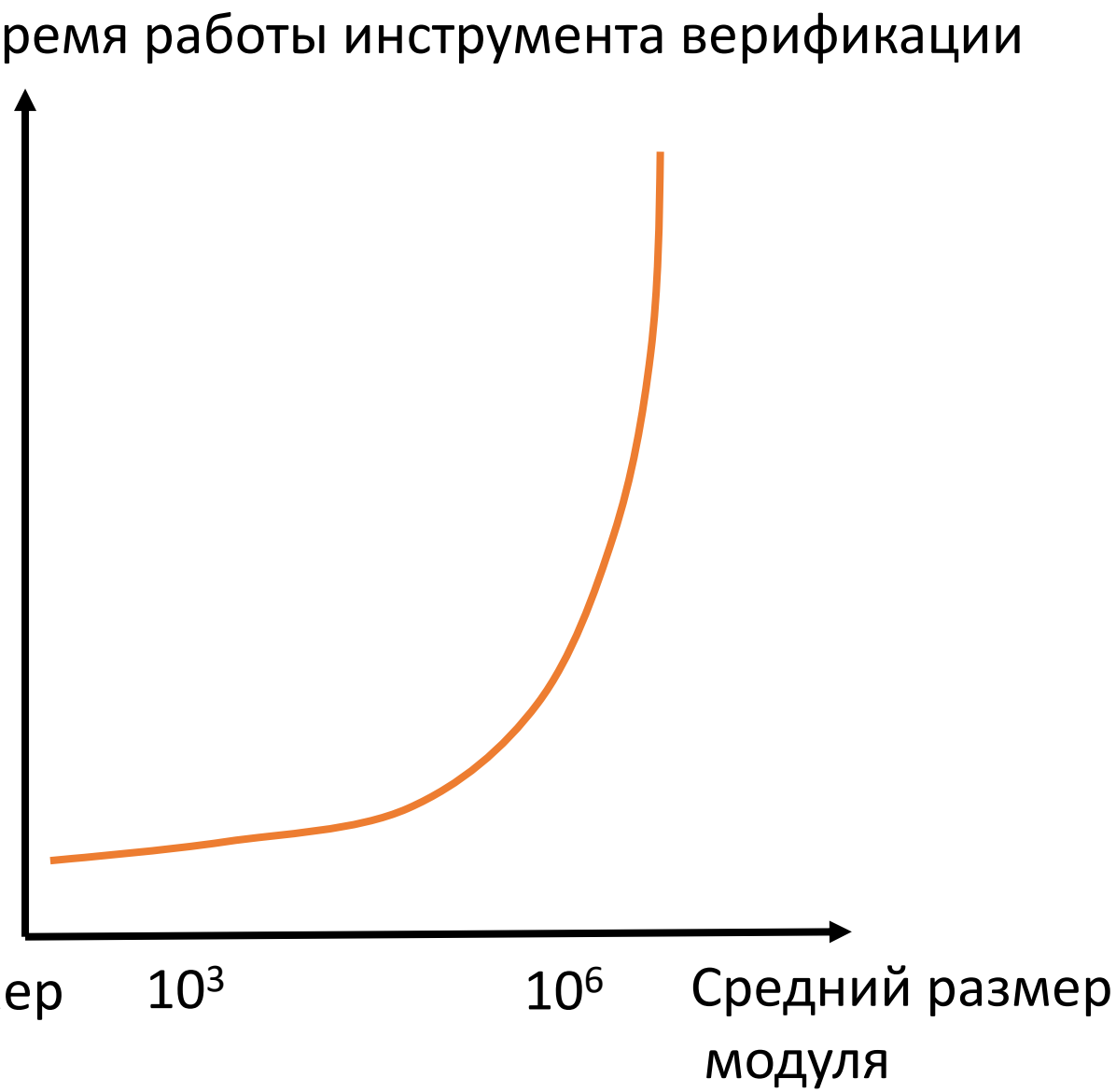
Метод автоматизированной декомпозиции Си-программ на модули

Компоненты программных систем

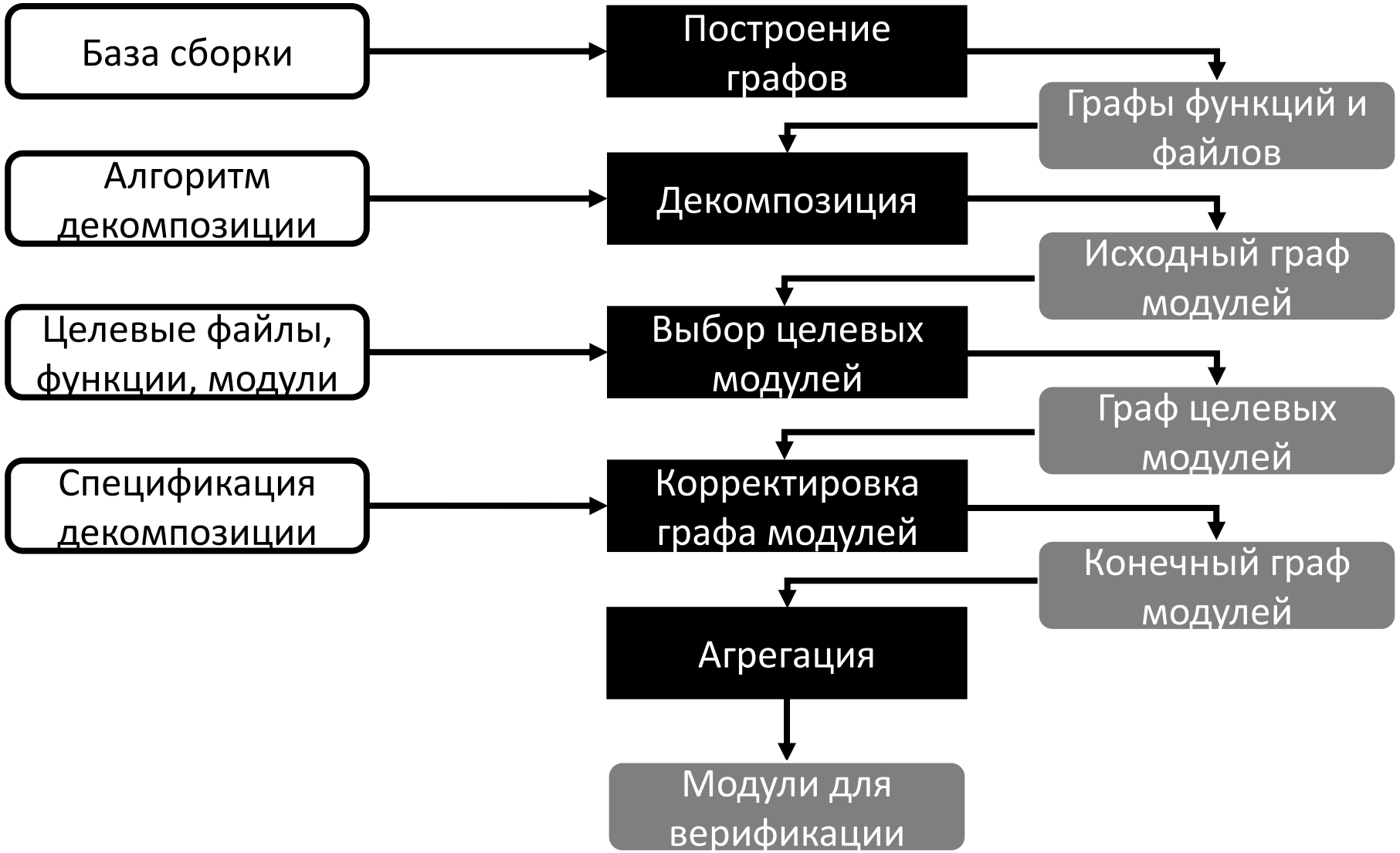
Программа	Размер программы (10^6 строк)	Количество компонентов
Ядро ОС Linux	15	5000
Веб-сервер Apache	1,5	150
Библиотека GTK	1,0	200
Видеопроигрыватель VLC	0,5	80
BusyBox	0,2	300

Зависимость времени верификации и трудоемкости моделирования окружения от размера модулей

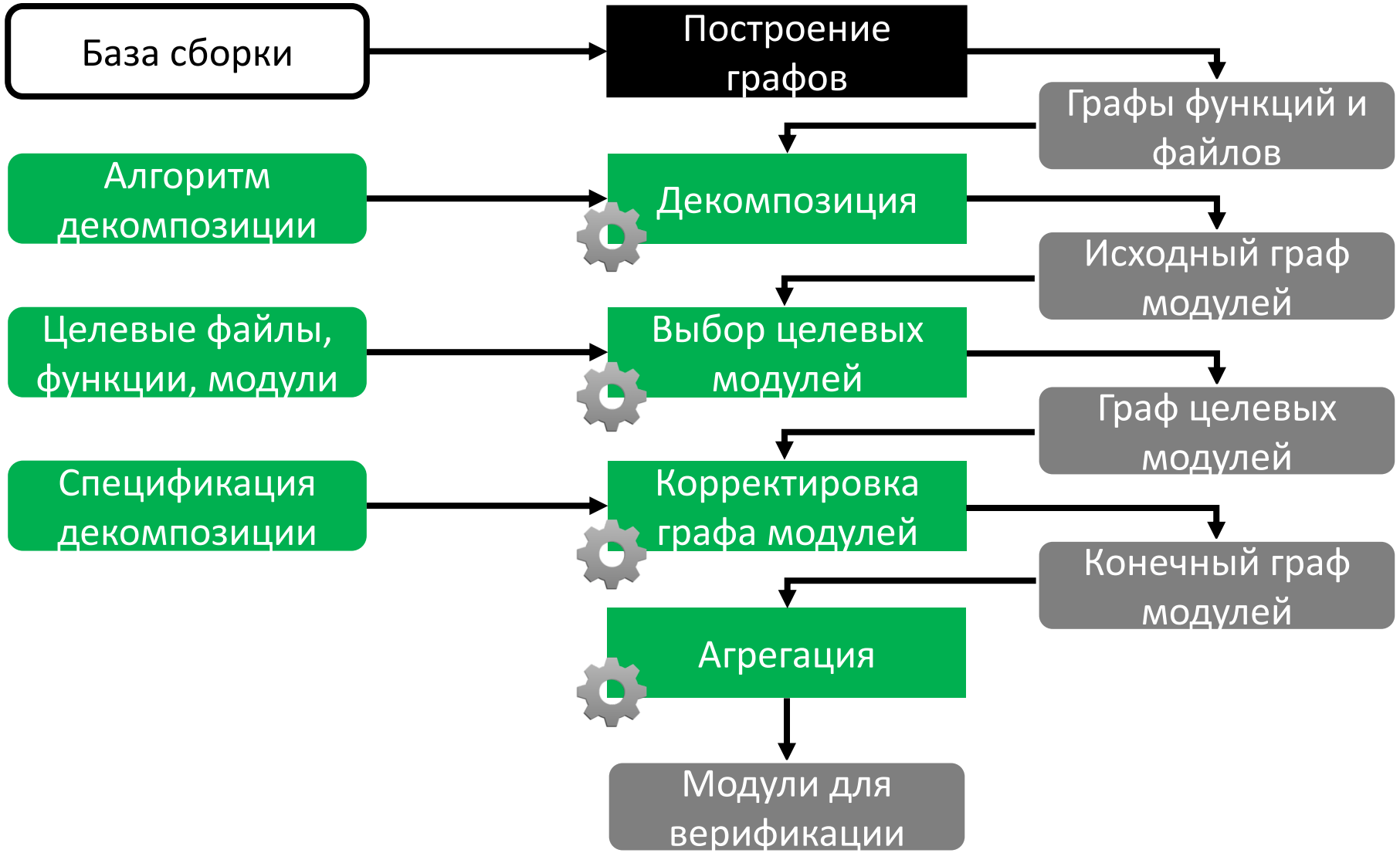




Метод автоматизированной декомпозиции Си-программ на модули

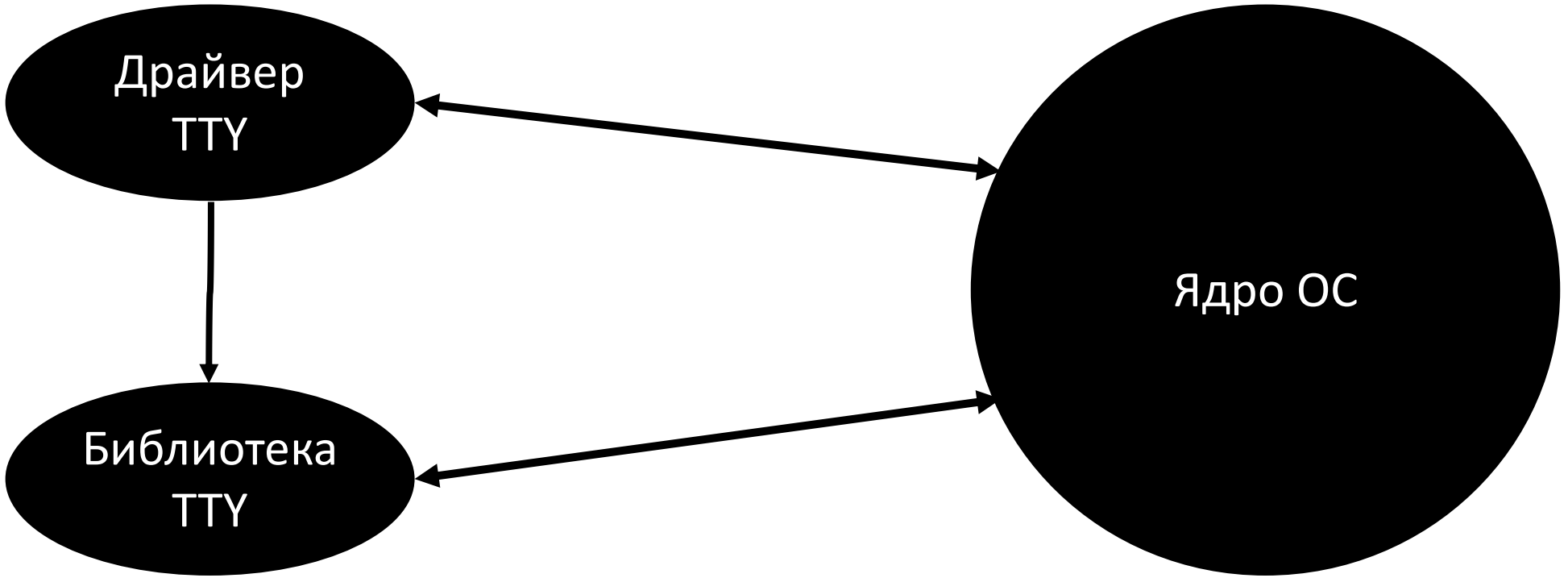


Метод автоматизированной декомпозиции Си-программ на модули

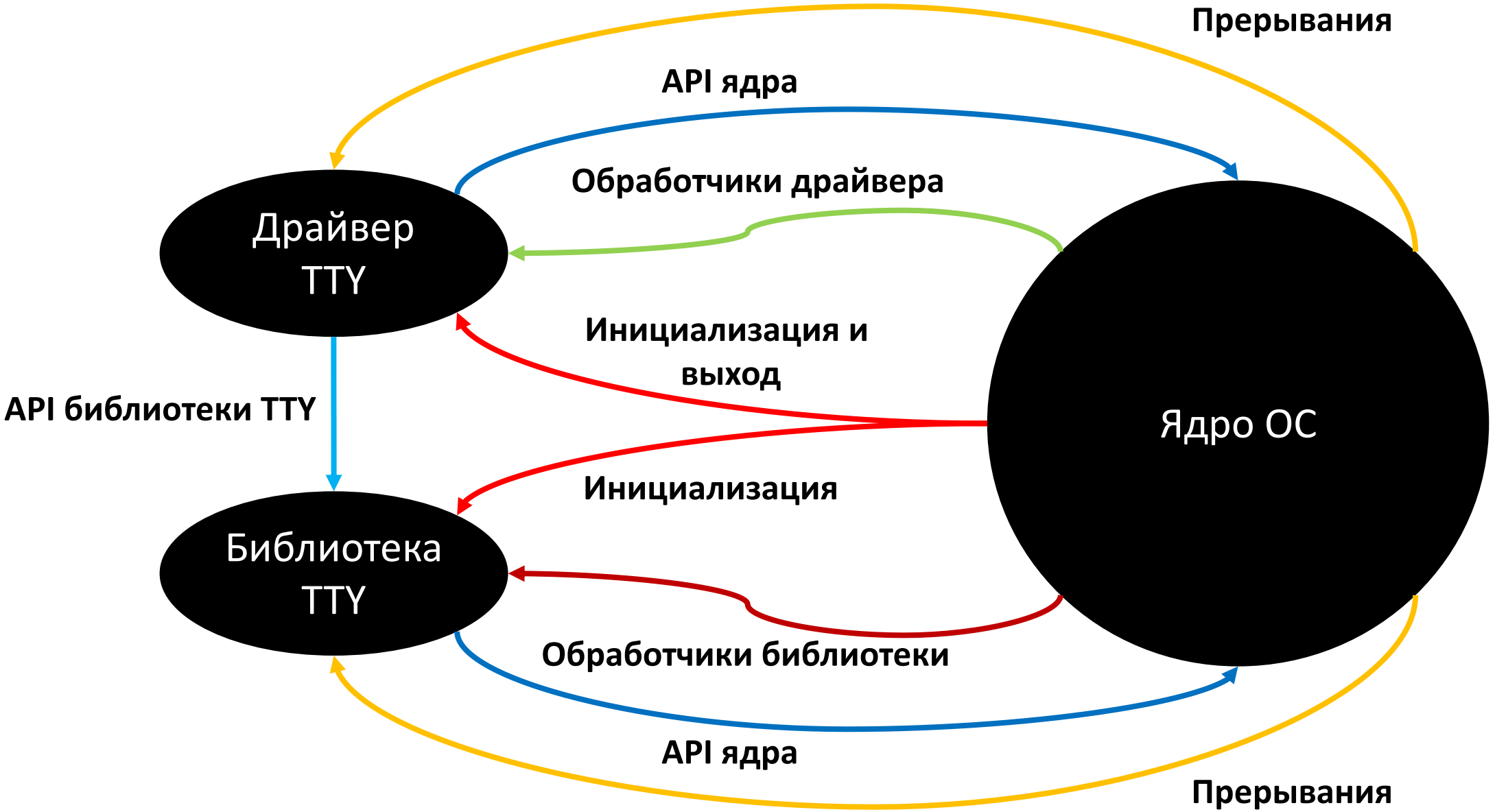


Метод спецификации моделей окружения на основе композиции систем переходов

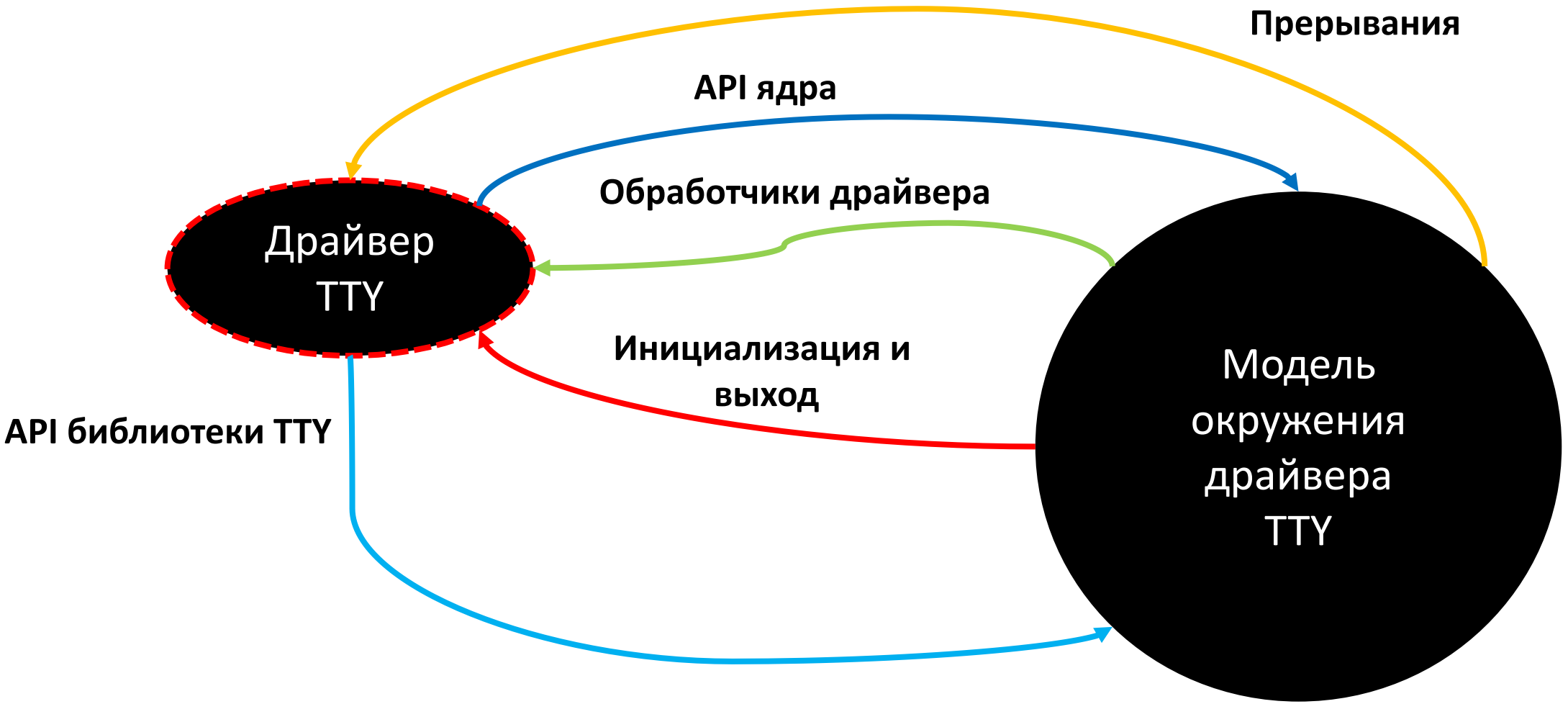
Пример выбора варианта декомпозиции



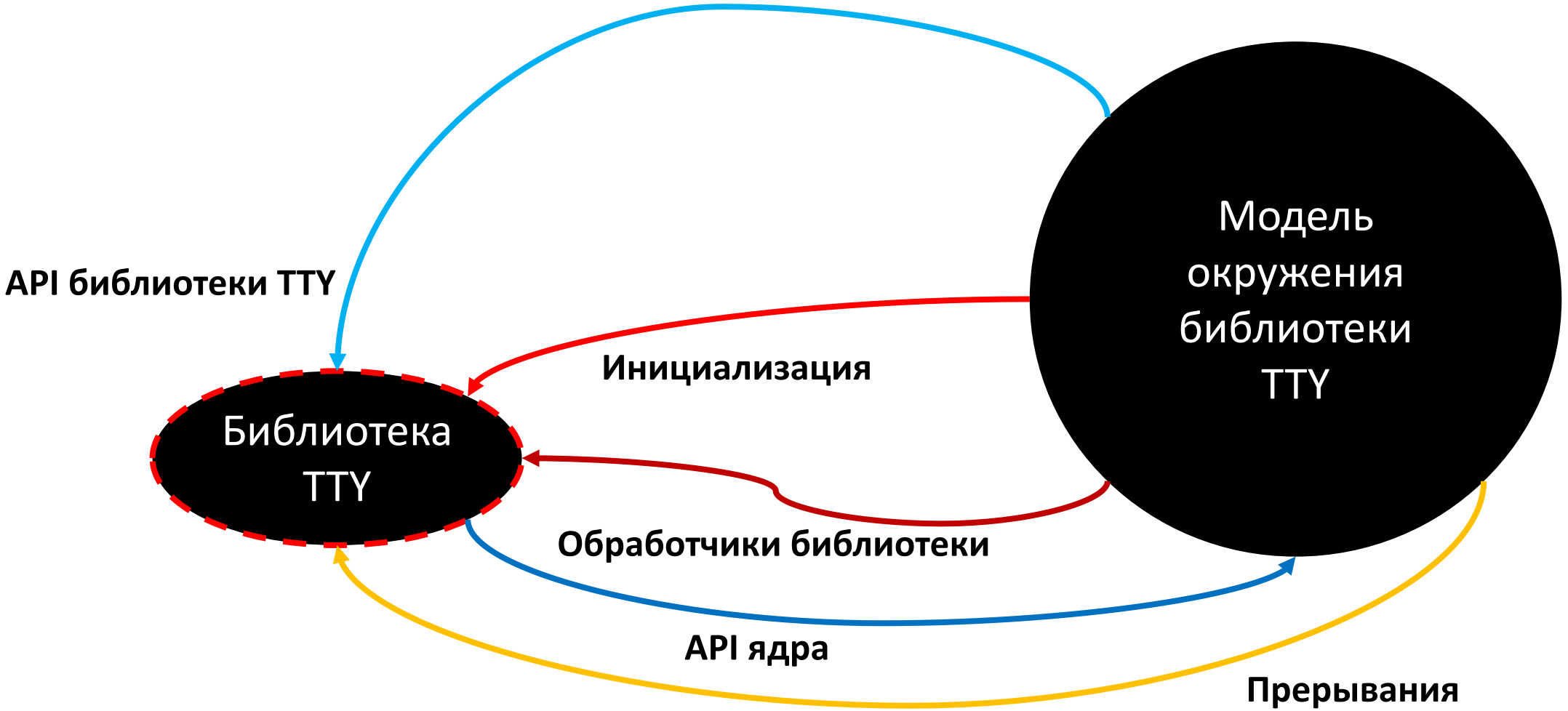
Взаимодействие драйвера и библиотеки с ядром



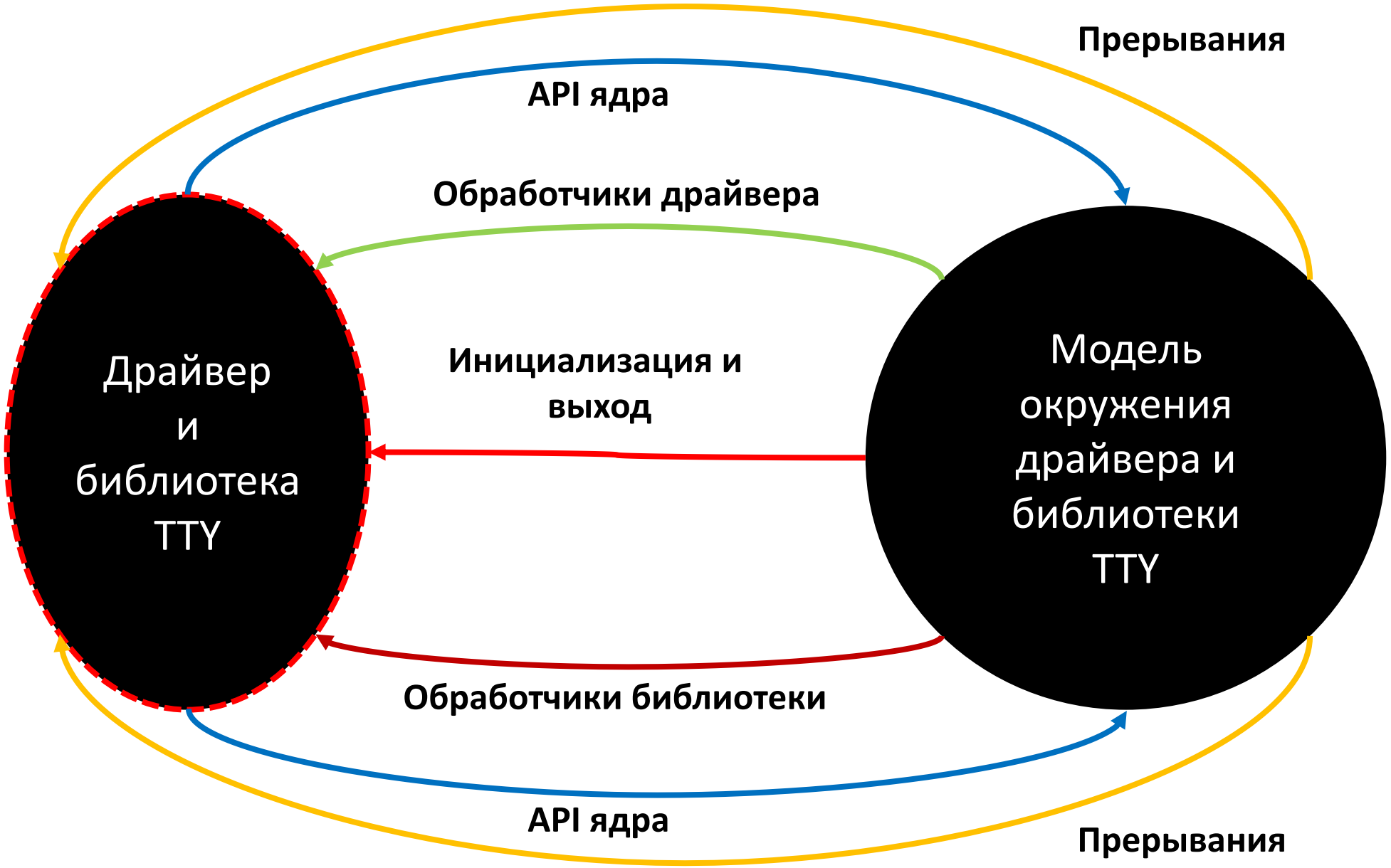
Пример выбора варианта декомпозиции 1 (1)



Пример выбора варианта декомпозиции 1 (2)



Пример выбора варианта декомпозиции 2



Предлагаемый подход

- Независимо друг от друга специфицировать модели сценариев взаимодействия модуля с окружением, описывая
  - порядок событий
  - зависимости по данным
  - ограничения на контекст
- Строить модель как параллельная композицию сценариев с возможностью переиспользования спецификации отдельных сценариев
- Учитывать при композиции возможность появления новых ограничений в зависимости от выполненной декомпозиции системы на модули

Спецификация сценариев взаимодействия модуля и окружения

$M = < P, E >, E = \mathcal{E}_1 \,||\, \ldots \,||\, \mathcal{E}_n$

P – вспомогательные функции на языке Си

$\mathcal{E}_i = < V, A, a_0, R >$ – модель сценария взаимодействия (потока или функции)

V – множество переменных состояния

A – множество действий:

- Посылка или получение сигнала согласно модели рандеву
- Блок кода на языке Си

a_0 – начальное действие

$R: A \times A$ – отношение переходов

```
void init_scenario(void) {
    // ----- State vars -----
    int ret;

    // ----- Transition Relation -----
    // Block Action 1 begin
    ret = tty_init();
    // Block Action 2 end

    if (__VERIFIER_nondet_int())  {
        // Send Action 2 begin
        __VERIFIER_assume(ret == 0);
        send(“reg_driver");
        // Send Action 2 end
     }
}
```

Отсутствие влияния трансляции моделей окружения на возможность обнаружения ошибок

Лемма 1. Истинность предиката пути, на котором могут выполняться подряд две независимые операции не зависит от порядка выполнения данных операций между собой.

Лемма 2. На эквивалентность блоков не влияют независимые операции выполняемые параллельно в других процедурах.

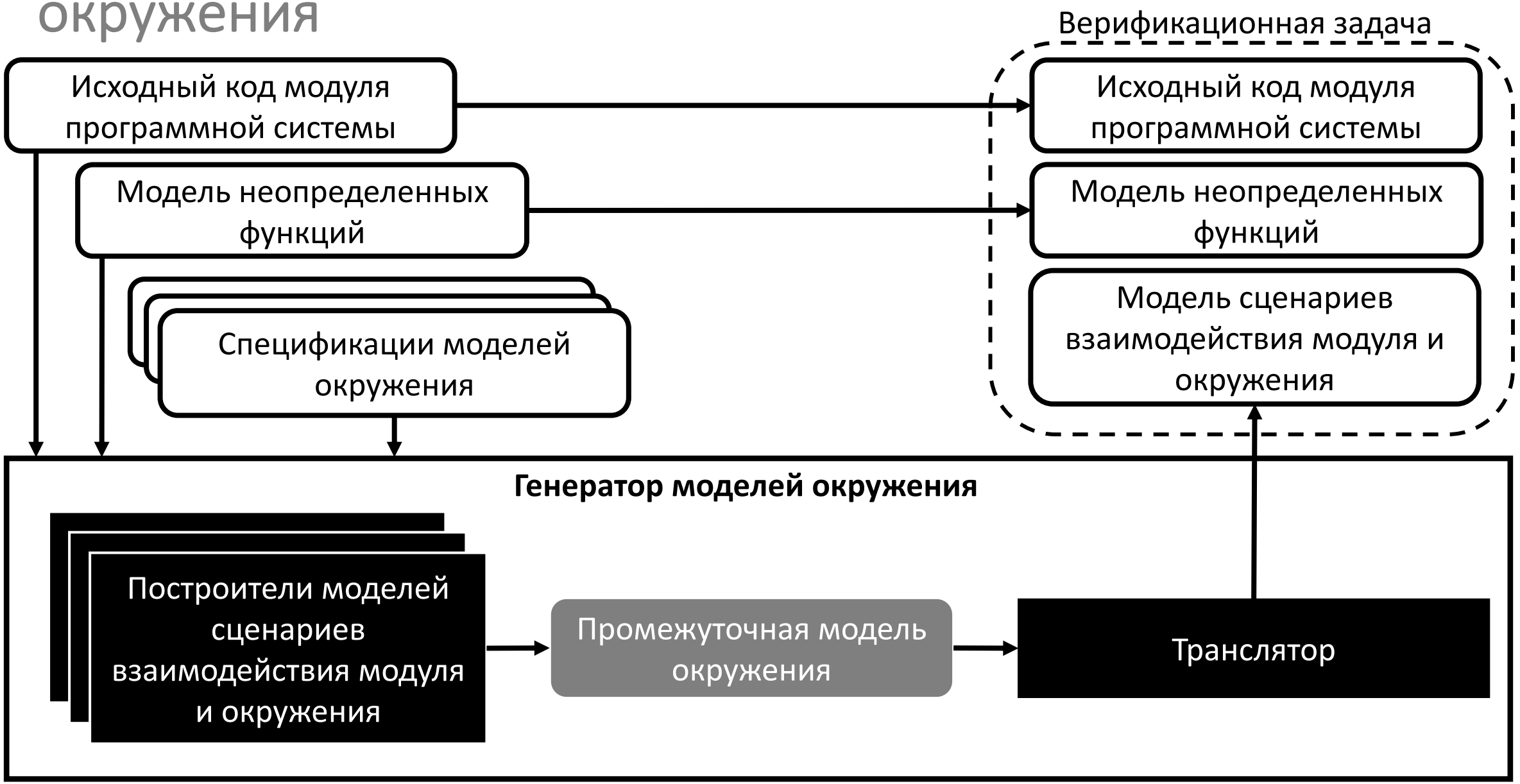
Лемма 3. Предикат пути через блоки кода с захваченной блокировкой принимает истинное значение только в том случае, если на этом пути операции между метками входа и выхода каждого блока под блокировкой может одновременно выполнять только один поток.

Лемма 4. На пути выполнения блока из операций $flag$ и $recv$ приема сигнала d всегда происходит посылка данного сигнала из другого потока.

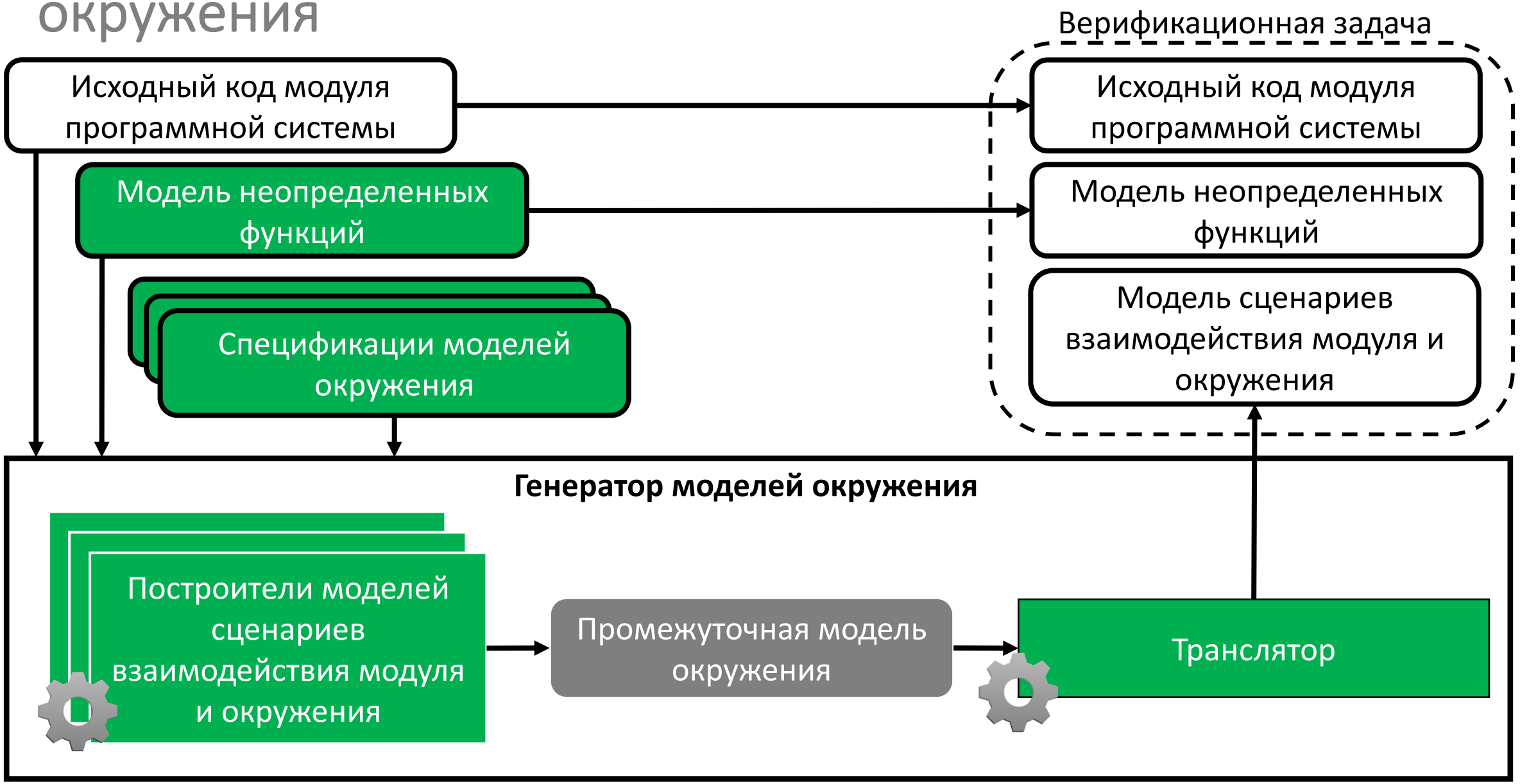
Теорема 1. Множества достижимых ошибочных состояний исходной программы на языке ELZ и результата ее трансляции на язык LZ изоморфны.

Метод автоматического синтеза моделей окружения модулей

Метод автоматизированного синтеза моделей окружения



Метод автоматизированного синтеза моделей окружения



Результаты практического применения

Система верификации Klever

Система верификации Klever позволяет

- Адаптировать процесс верификации для проверки
  - Разных видов программ
    - Драйверы и подсистемы ядра ОС Linux
    - Апплеты BusyBox
    - Драйверы ОСРВ Zephyr
  - Разных типов требований
    - Отсутствие гонок по данным
    - Корректность работы с памятью
    - Корректность работы с программным интерфейсом
- Параллельно синтезировать и решать верификационные задачи
  - На одной вычислительной машине, включая виртуальные машины OpenStack
  - На вычислительном кластере под управлением VerifierCloud

Трудоемкость верификации драйверов и подсистем ОС Linux

Этап	Драйверы Serial (30KLOC, 100% покрытие)	Все драйверы (3MLOC, 51% покрытие)	Подсистемы (1MLOC, 45% покрытие)	Итого
Разработка стратегий декомпозиции	0,25 чел. мес. (100LOC Python)	-	0,25 чел. мес. (100LOC Python)	0,5 чел. мес. (200LOC Python)
Разработка построителей моделей сценариев	3 чел. мес. (3KLOC Python)	-	0,5 чел. мес. (500LOC Python)	3,5 чел. мес. (3,5KLOC Python)
Разработка спецификаций моделей окружения	4,5 чел. мес. (7KLOC DSL)	5,5 чел. мес. (10KLOC DSL)	-	10 чел. мес. (17KLOC DSL)
Итого	7,75 чел. мес.	5,5 чел. мес.	0,75 чел. мес.	14 чел. мес.

Время верификации драйверов ядра ОС Linux

Целевые модули	Верификационных задач	2 процессорных ядра	4 процессорных ядра	30 * 4 процессорных ядра
Serial драйверы	1,5 тыс.	5 часов	2.7 часов	30 минут
Все драйверы	100 тыс.	25 дней	8 дней	11 часов

Сравнение точности Klever и LDV Tools

Система верификации	Вер. задач	Ошибки	Ложные срабатывания	Доказана корректность	Нет вердикта
Корректность работы с интерфейсами ядра					
Klever	1470	1	**5**	1432	16
LDV	1470	0	**31**	1421	9
Корректность работы с памятью и отсутствие гонок по данным					
Klever	98	**5**	11	52	30
LDV	-	-	-	-	-

Проверялся набор из 49 Serial драйверов, объем исходного кода составил 30 KLOC

Основные результаты работы

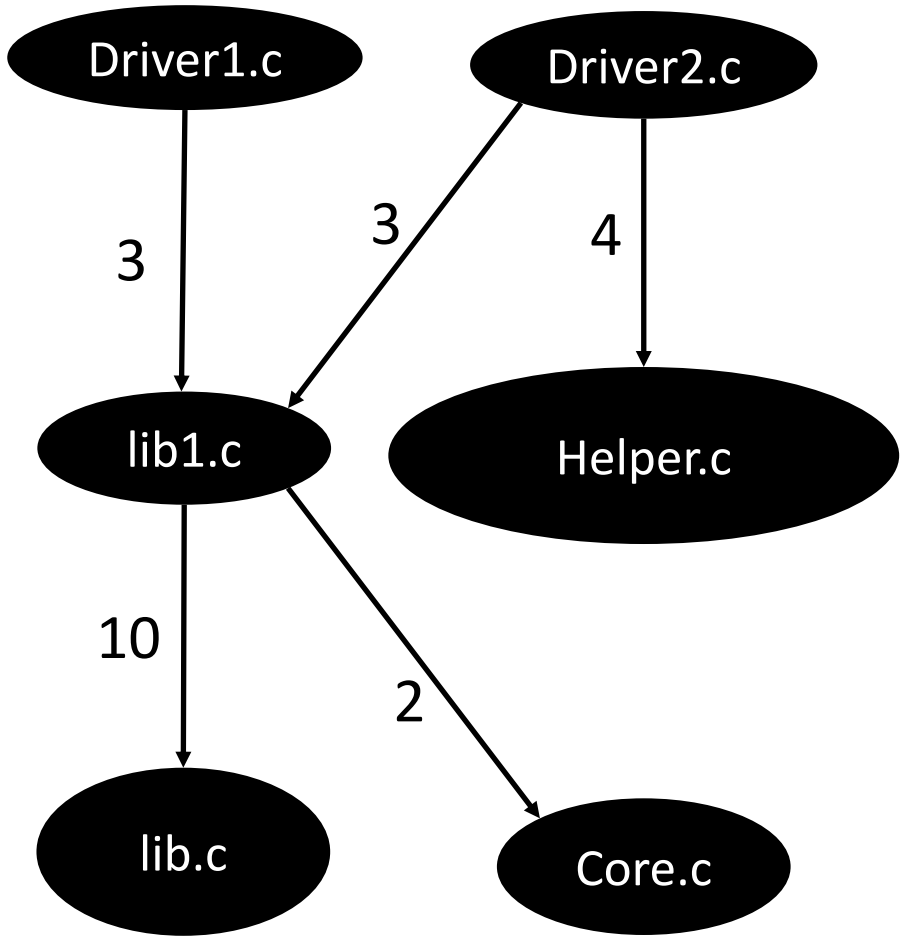
- Разработан **метод автоматизированной декомпозиции Си-программ на модули**
- Разработан **метод спецификации моделей окружения модулей** на основе композиции систем переходов
- Разработан **метод автоматизированного синтеза моделей окружения** модулей, позволяющий адаптировать процесс синтеза для проверки разных видов требований и программ

На основе предлагаемых методов была **реализована система верификации Klever**, предназначенная для проверки требований к программным системам на языке программирования Си. Система используется в проектах отдела Технологий программирования ИСП РАН им. В.П. Иванникова и позволила выявить сотни ошибок в исходном коде различных программных систем.

Спасибо за внимание

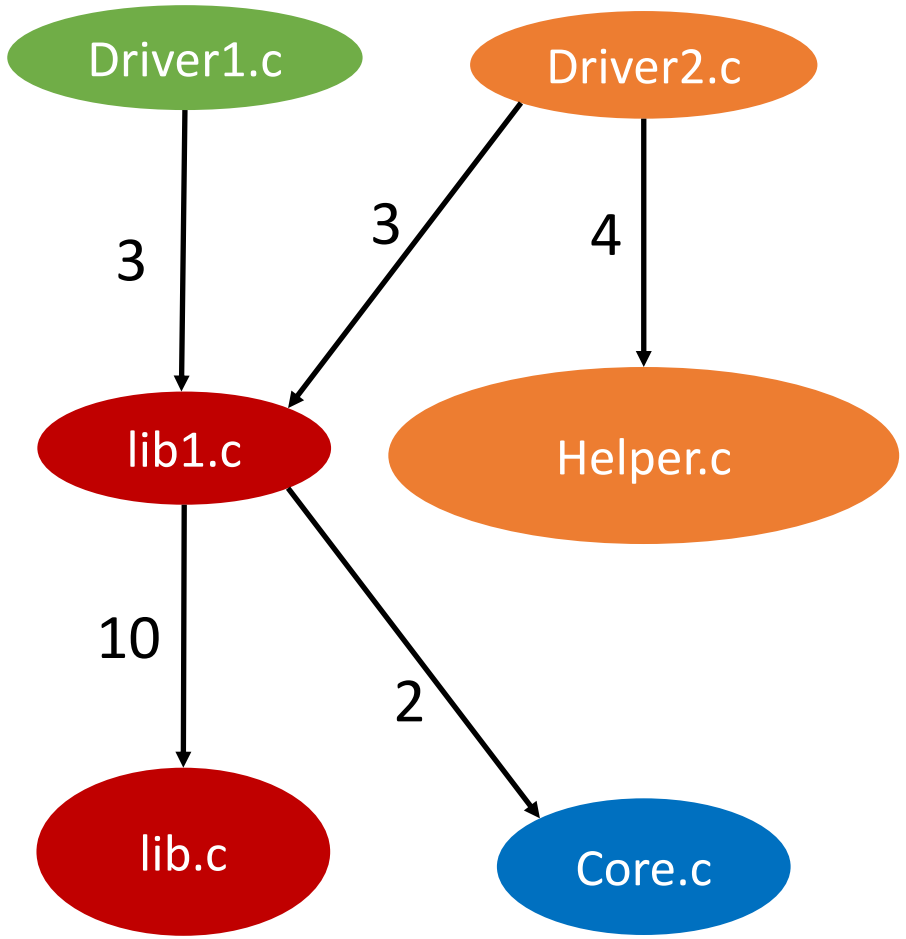
Пример декомпозиции программы

Сценариев вызова точек входа: 22



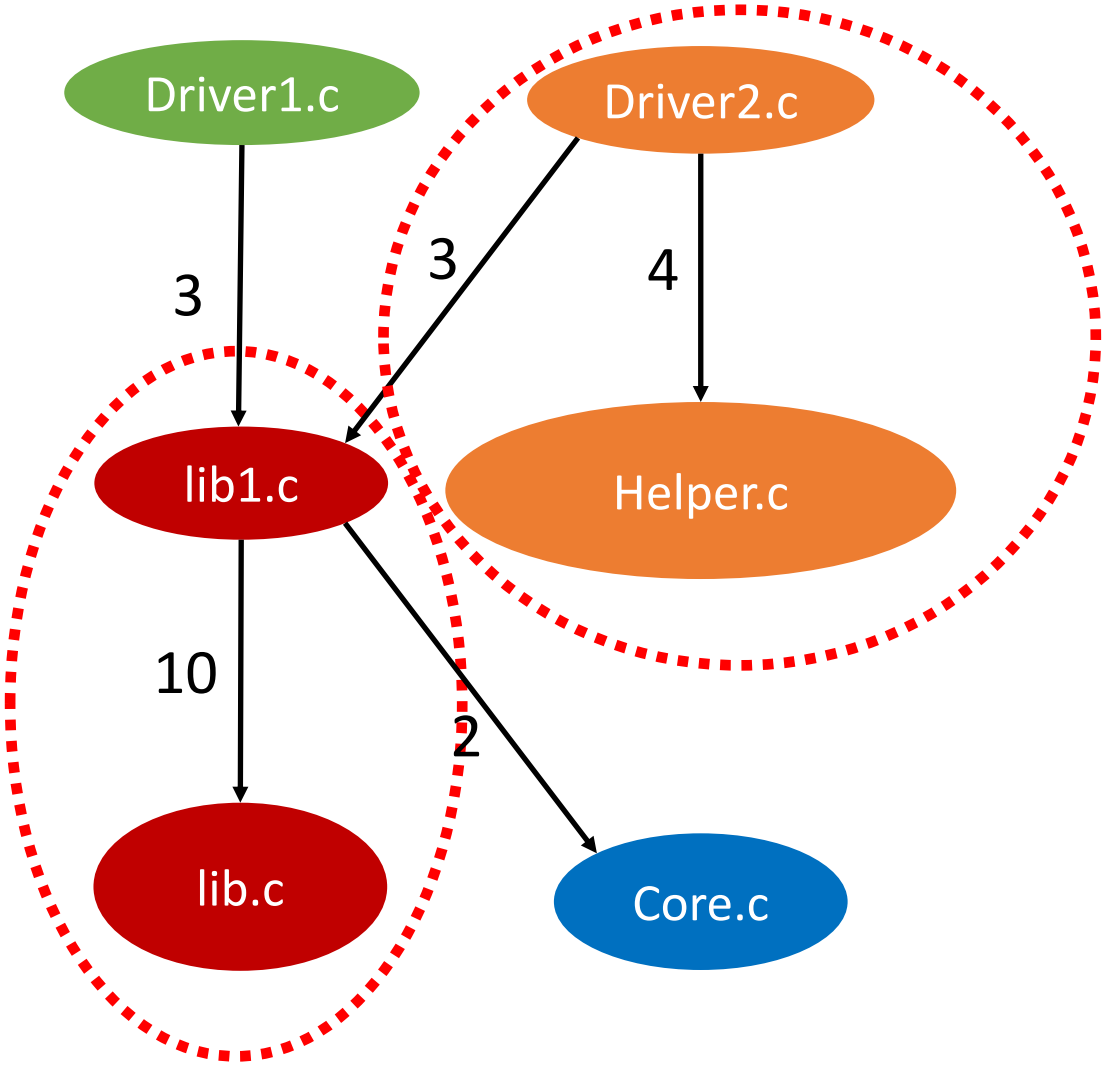
Пример декомпозиции программы

Сценариев вызова точек входа: 22



Пример декомпозиции программы

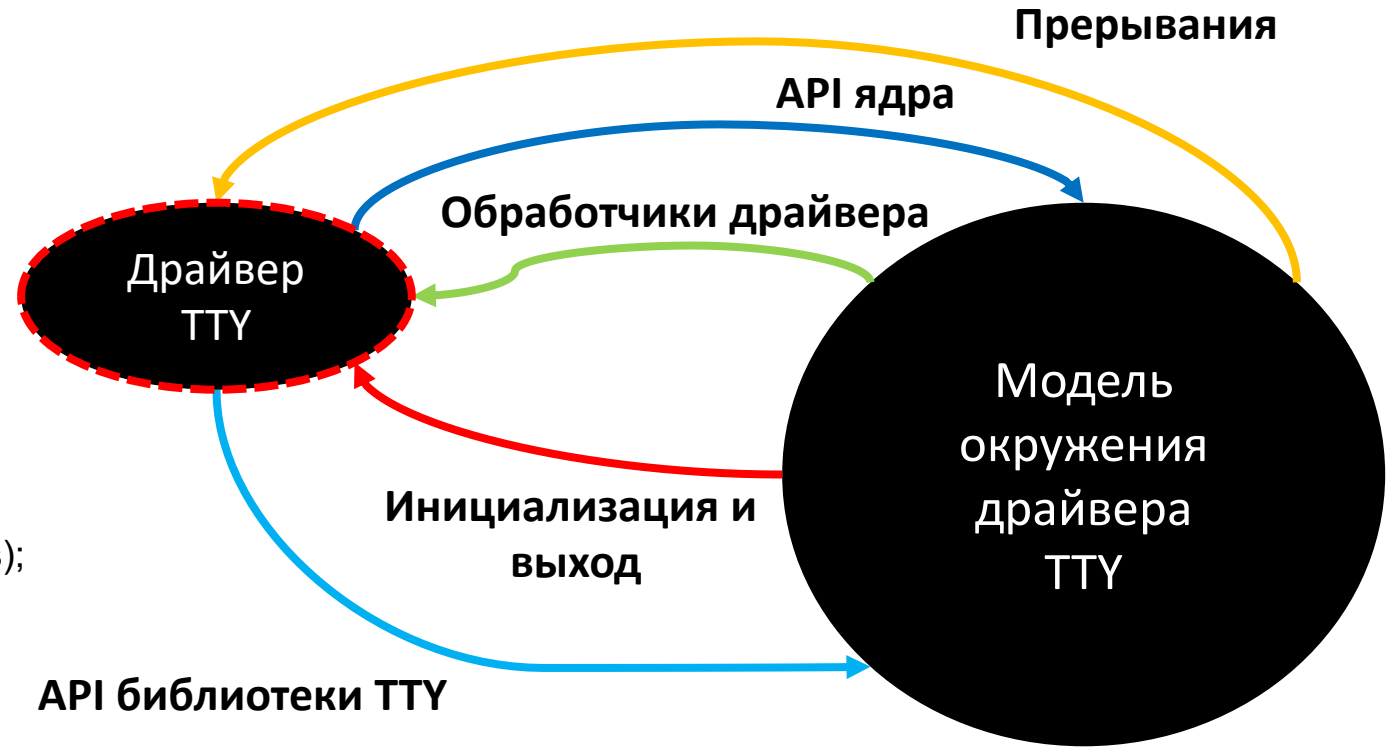
Сценариев вызова точек входа: 8



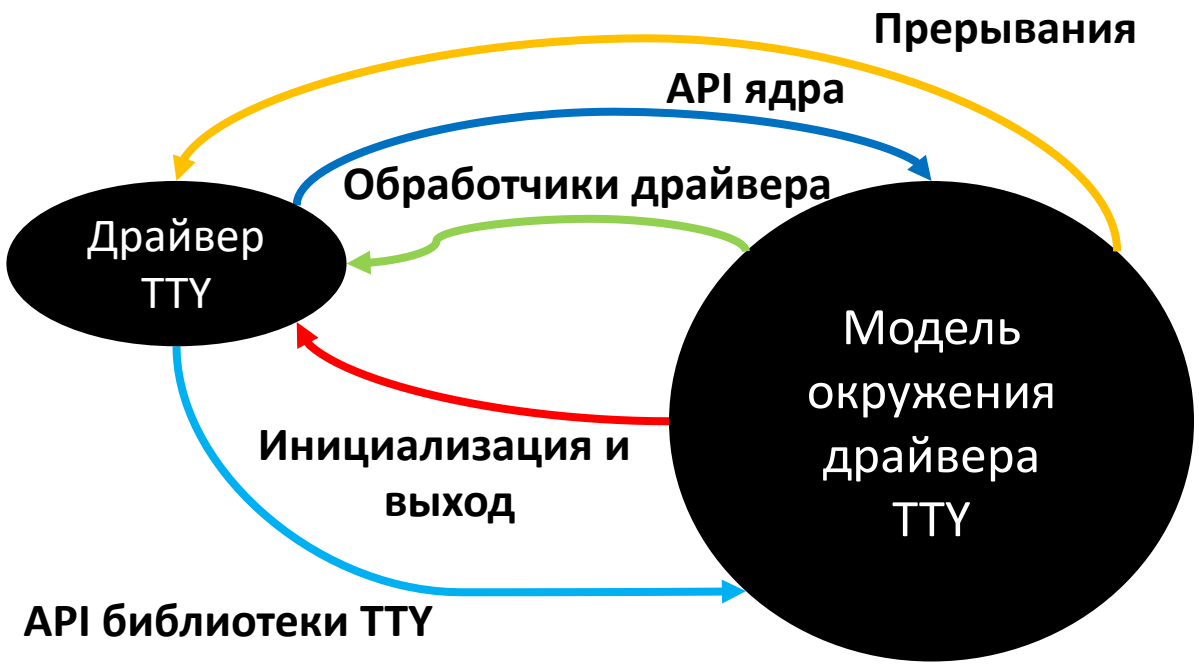
Сценарии использования драйвера

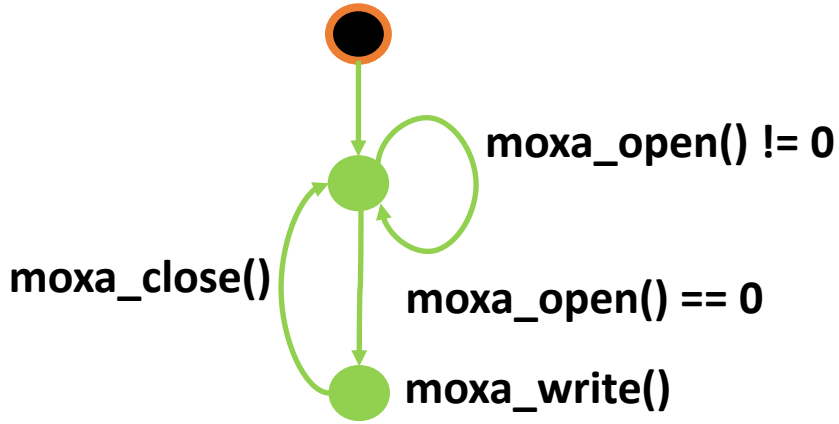
```
static const struct tty_operations moxa_ops = {
        .open = moxa_open,
        .close = moxa_close,
        .write = moxa_write
};
static struct tty_driver *moxaDriver;

static int __init moxa_init(void) {
        int retval = 0;
        moxaDriver = tty_alloc_driver(…);
        if (IS_ERR(moxaDriver))
                return PTR_ERR(moxaDriver);
        tty_set_operations(moxaDriver, &moxa_ops);
        if (tty_register_driver(moxaDriver)) {
                put_tty_driver(moxaDriver);
                return -1;
        }
        return 0;
}
static void __exit moxa_exit(void) {
        tty_unregister_driver(moxaDriver)
        put_tty_driver(moxaDriver);
}
module_init(moxa_init);
module_exit(moxa_exit);
```

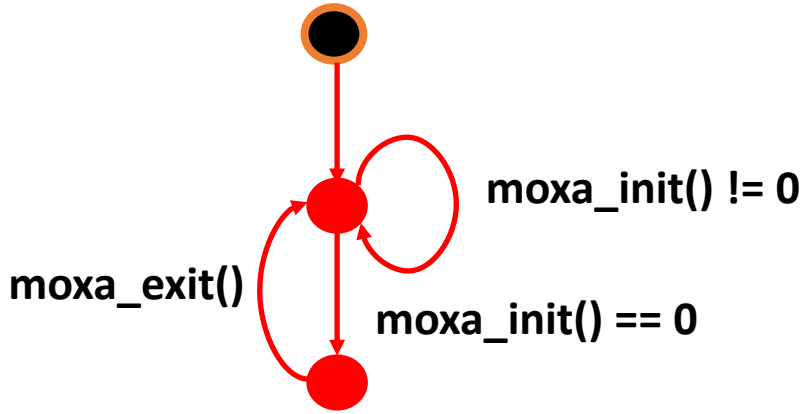


Сценарии взаимодействия драйвера и окружения



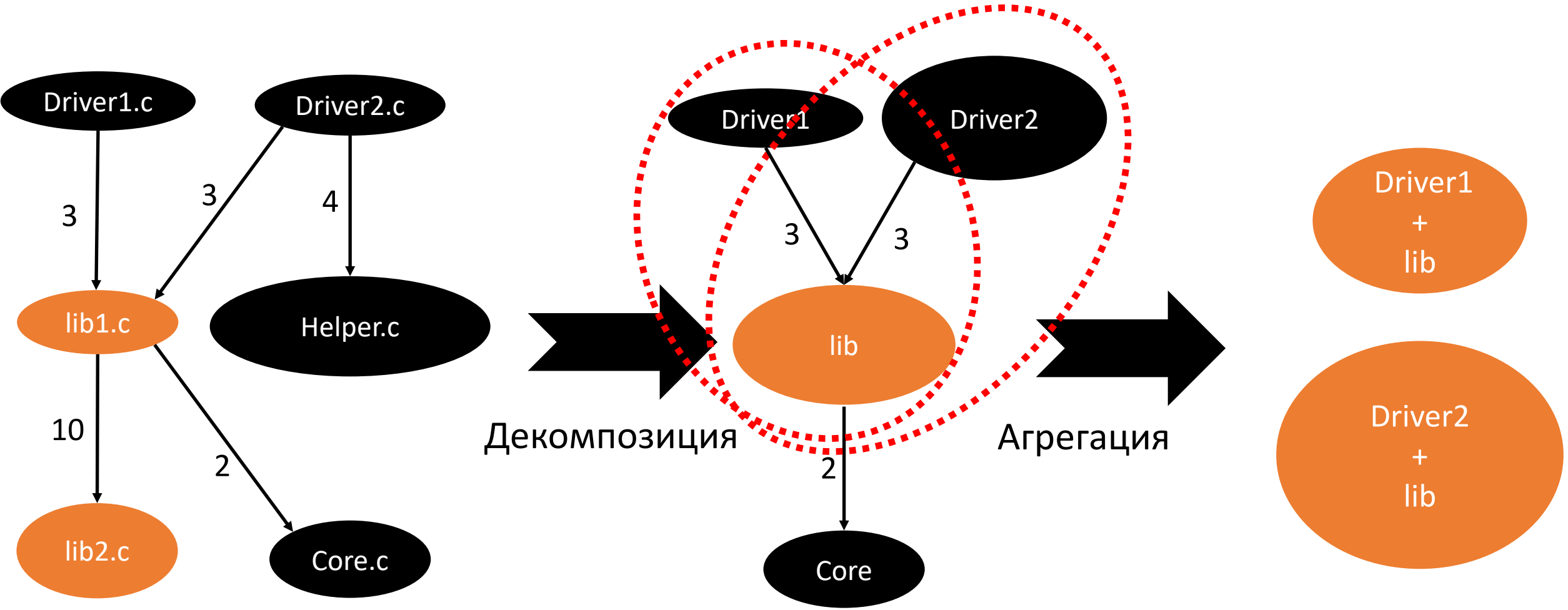


Сценарий вызова обработчиков



Сценарий инициализации драйвера

Агрегация модулей программы



Трудоемкость и точность верификации апплетов BusyBox

Этап	Трудозатраты
Разработка стратегий декомпозиции	0,25 чел. мес. (100LOC Python)
Разработка построителей моделей сценариев	0 чел. мес. -
Разработка спецификаций моделей окружения	0,25 чел. мес. (200LOC DSL)
Итого	0,5 чел. мес.

Вердикт	Число апплетов
Ложное срабатывание	6
Ошибка	1
Ошибок не найдено	159
Нет вердикта	133