Композиционное тестирование на Python

Максим Кукликов

Обо мне

- 2 года диагностики в промышленности (научная работа)
- 4 года диагностики и тестировании девайсов и запчастей
- 2 года тестирования в вебе
- Разработчик Combidata
- Веду семинары, пишу статьи, менторю юных автоматизаторов

Максим Кукликов

Композиционное тестирование на Python

Максим Кукликов

Для кого доклад?

- Практикующим разработчикам систем тестирования
- Разработчикам которых тестируют
- Ребятам любящим науку 🙃

Композиционное тестирование

Бурдонов Игорь Борисович

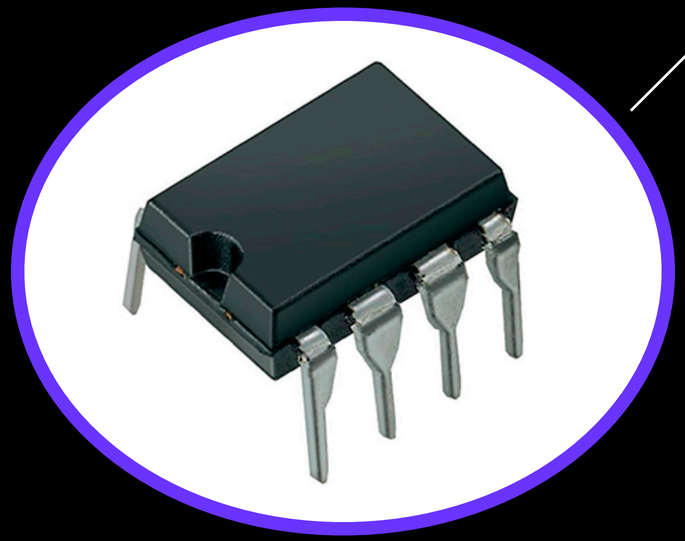
Косачев Александр Сергеевич

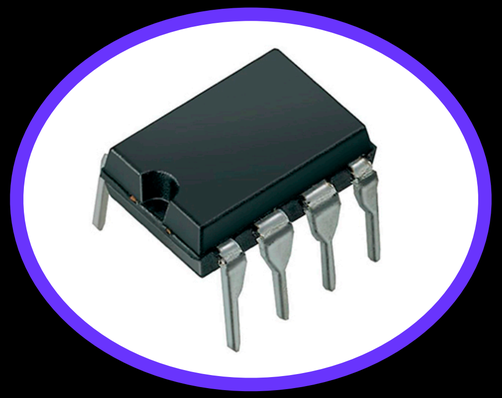
Автоматическое тестирование

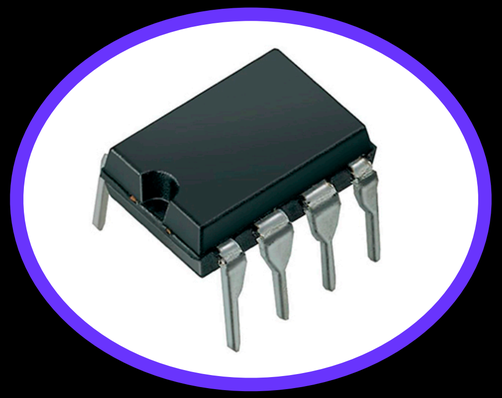
Автоматическое тестирование*

*грубая и краткая история

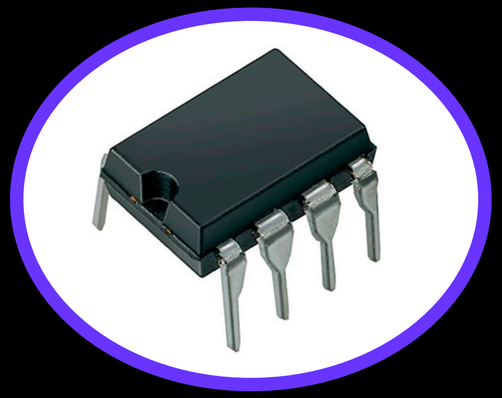
Привет, я Микросхема! 🖐





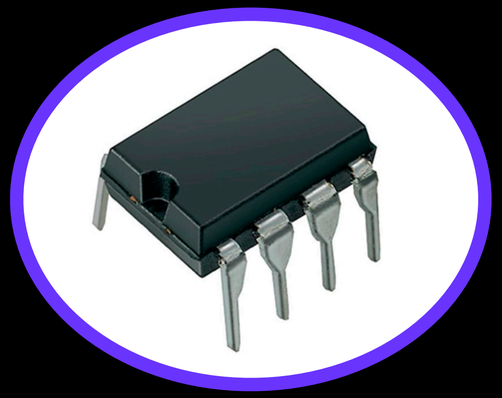


Массово запустили производство

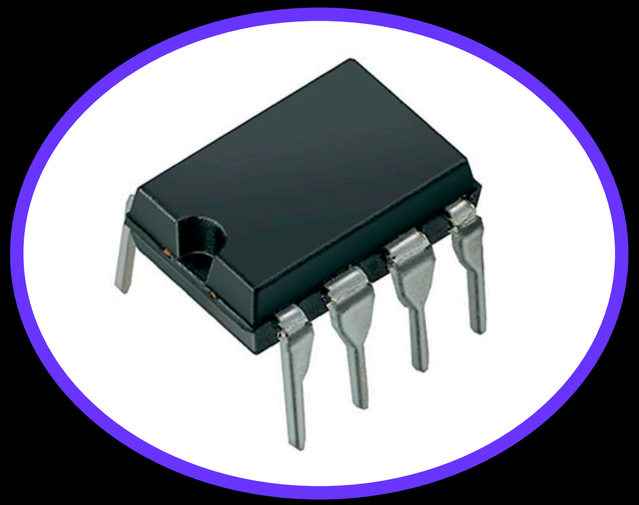


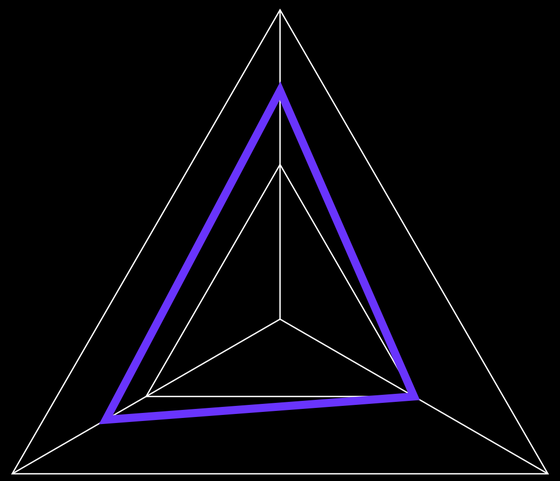
Массово запустили производство

Протестировали вручную



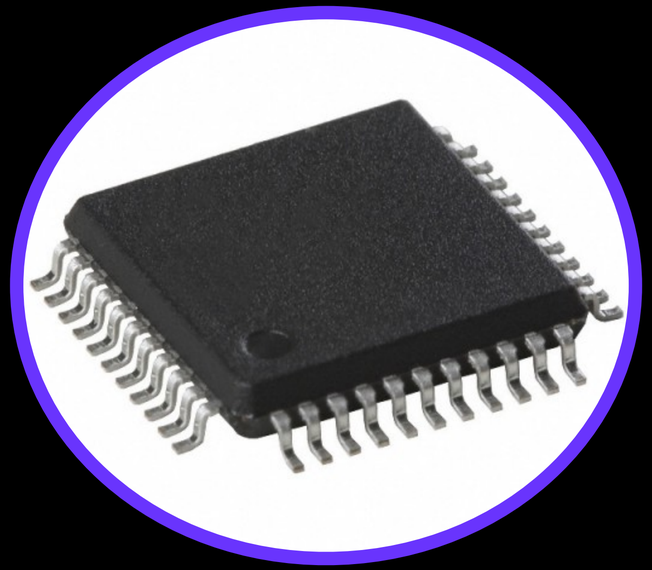
Массово запустили производство → Протестировали вручную → PROFIT

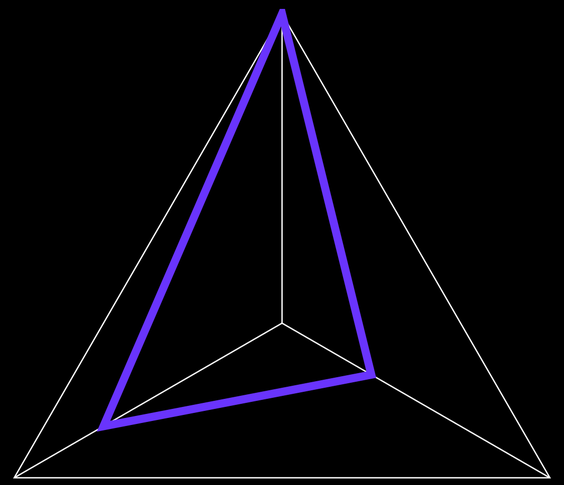




— Ручное тестирование простых и массовых продуктов

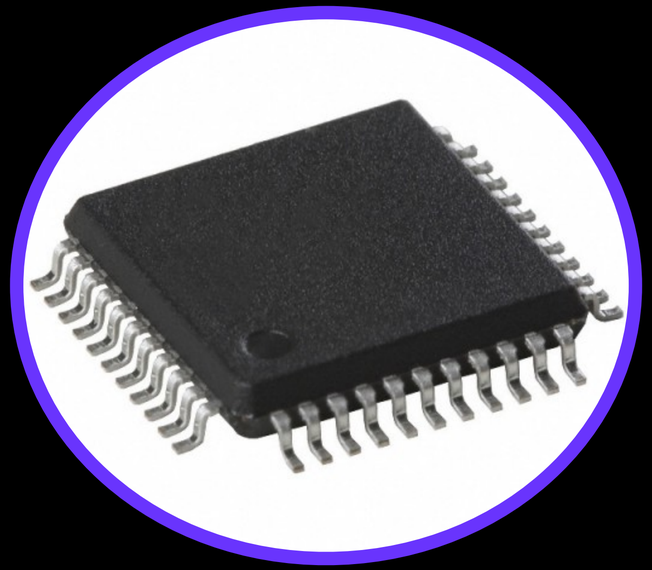
Массово запустили производство → Протестировали вручную → PROFIT





— Ручное тестирование простых и массовых продуктов

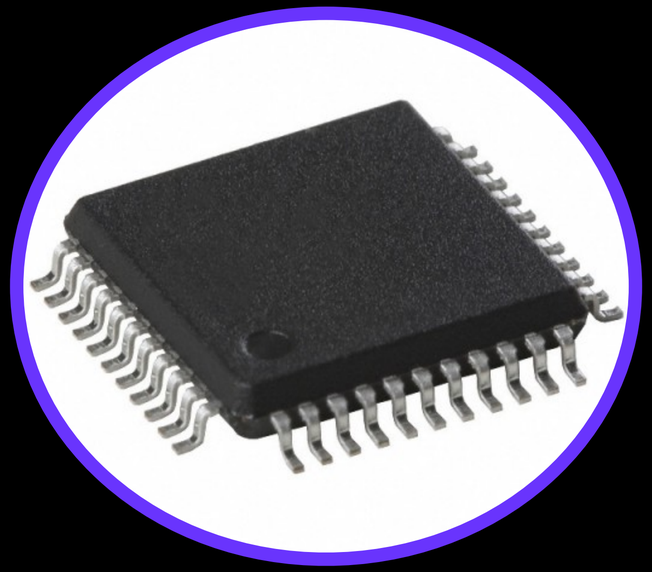
Массово запустили производство → Протестировали вручную → PROFIT

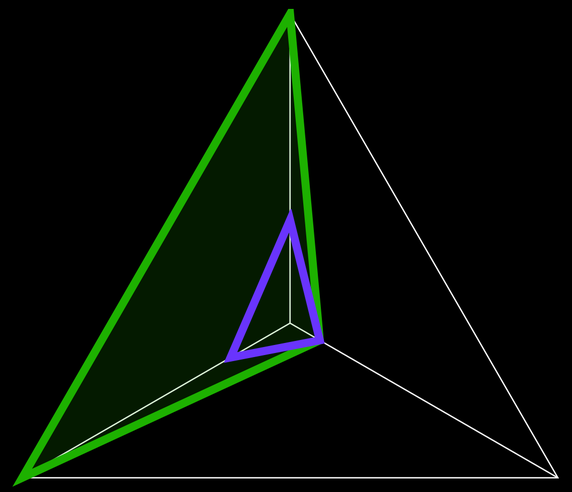


— Ручное тестирование простых и массовых продуктов

— Ручное тестирование сложных и массовых продуктов

Массово запустили производство → Протестировали вручную → ???

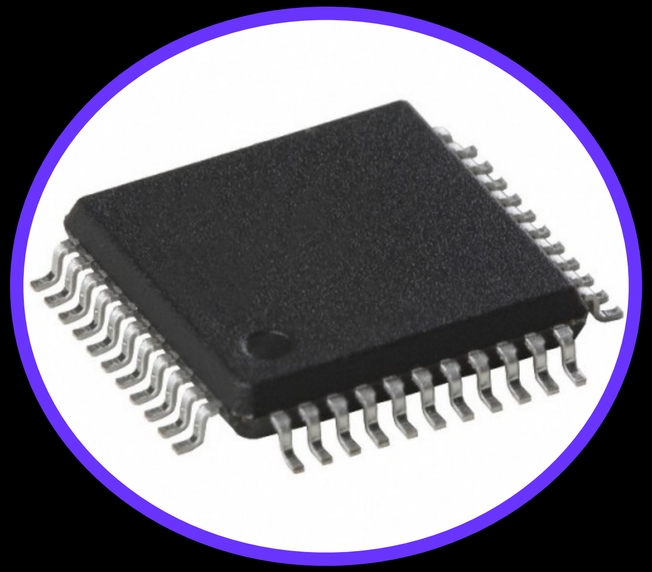


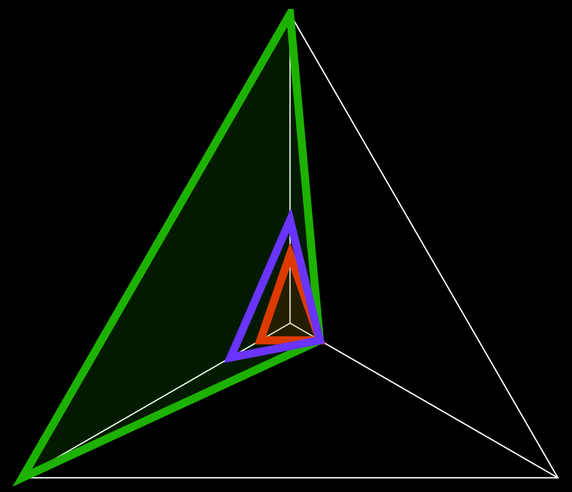


– Ручное тестирование простых и массовых продуктов

– Ручное тестирование сложных и массовых продуктов

Массово запустили производство → Протес… …вали в… … → ???



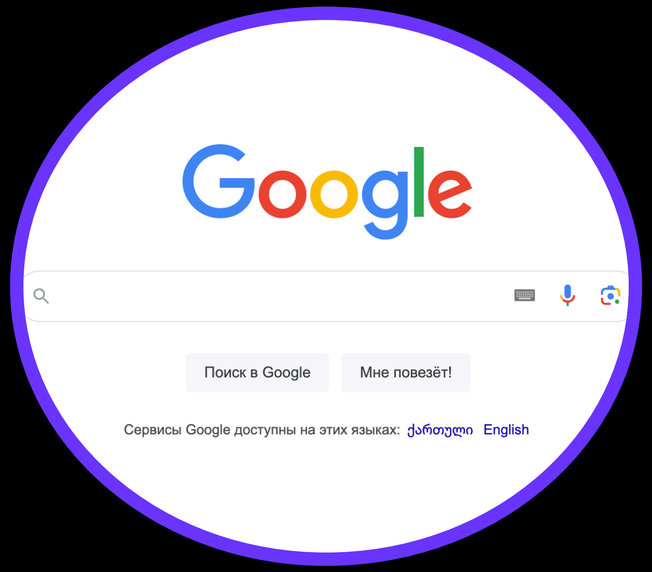


— Ручное тестирование простых и массовых продуктов

— Ручное тестирование сложных и массовых продуктов

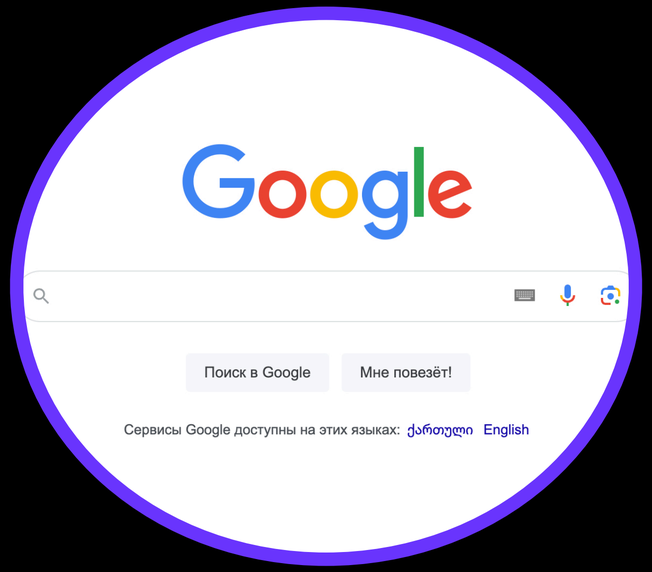
— Автоматическое тестирование сложных и массовых продуктов

Массово запустили производство → Автоматическое тестирование → PROFIT



— Ручное тестирование простых и массовых продуктов
— Ручное тестирование сложных и массовых продуктов
— Автоматическое тестирование сложных и массовых продуктов

Улучшаем продукт → Автоматическое тестирование → PROFIT



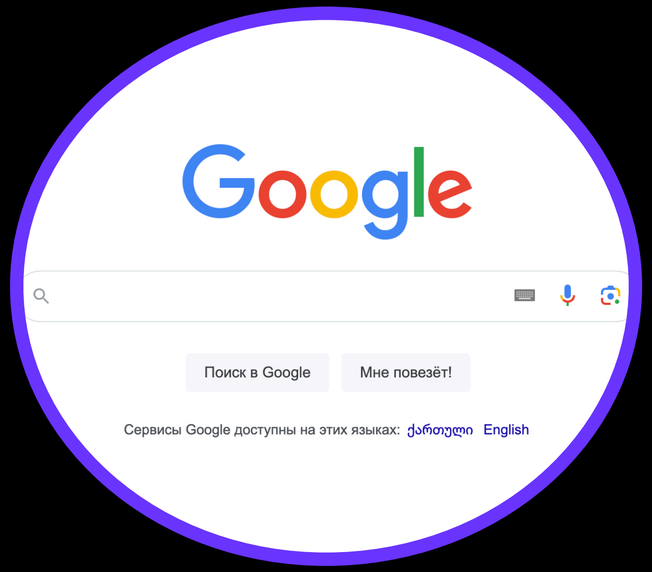
— Ручное тестирование простых и массовых продуктов

— Ручное тестирование сложных и массовых продуктов

— Автоматическое тестирование сложных и массовых продуктов

— Автоматическое тестирование динамических продуктов

Улучшаем продукт → Автоматическое тестирование → PROFIT



— Ручное тестирование простых и массовых продуктов

— Ручное тестирование сложных и массовых продуктов

— Автоматическое тестирование сложных и массовых продуктов

— Автоматическое тестирование динамических продуктов

Улучшаем продукт → Автоматическое тестирование → ???

Автоматическое тестирование

Основновные механники и процессы
и почему они плохо подходят для
динамических продуктов

Академическое тестирование

Основновные механники и процессы
и почему они плохо подходят для
динамических продуктов

Академическое тестирование (End-to-end, Integration, Unit)

Основновные механники и процессы и почему они плохо подходят для динамических продуктов

Академическое тестирование

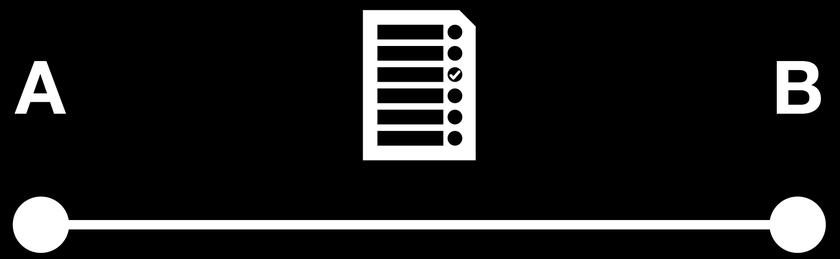
Академическое тестирование

A

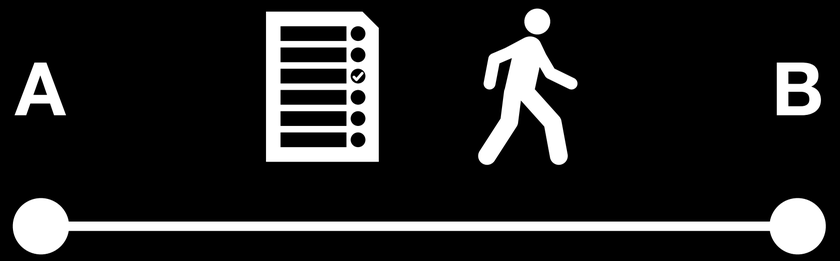
B

Академическое тестирование

Академическое тестирование



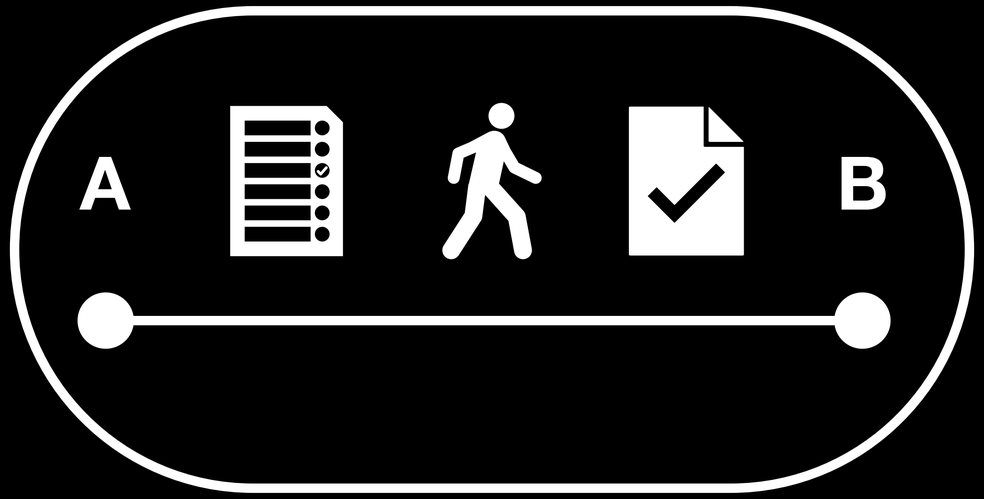
Академическое тестирование



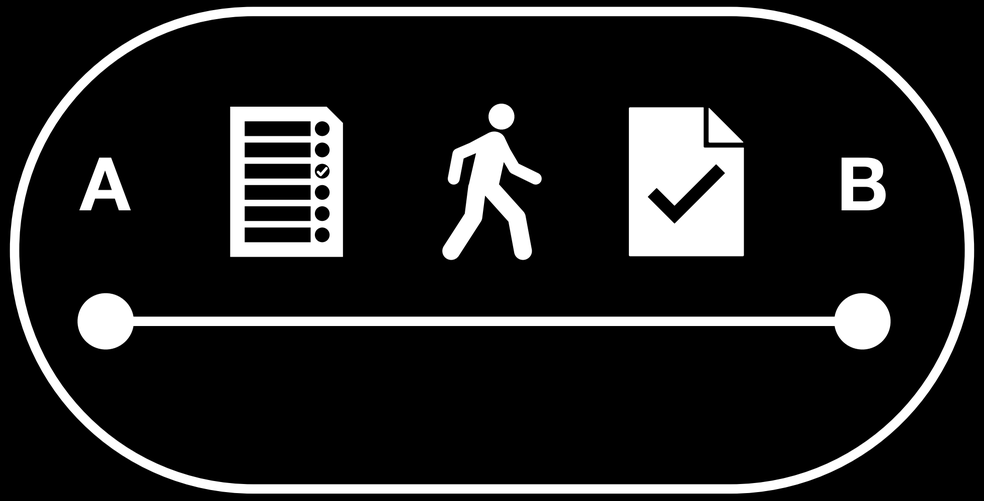
Академическое тестирование

Академическое тестирование

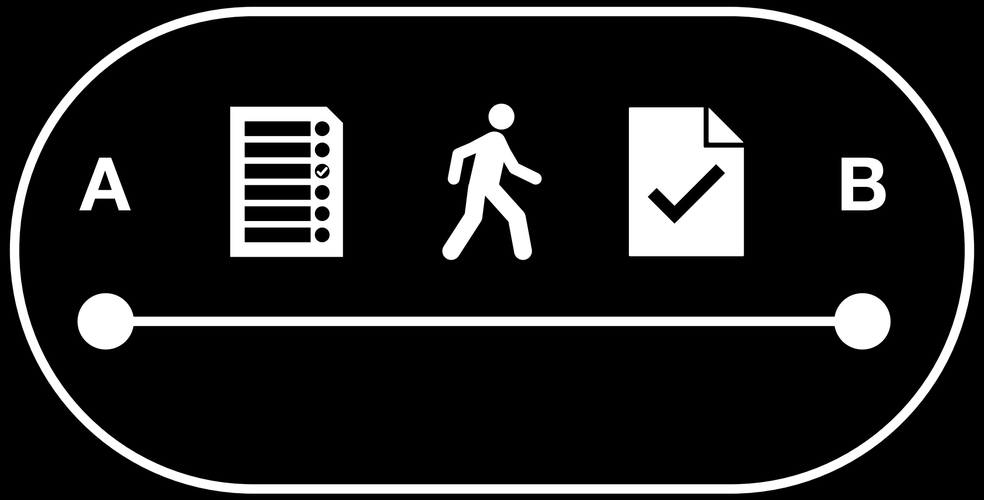
Test №1



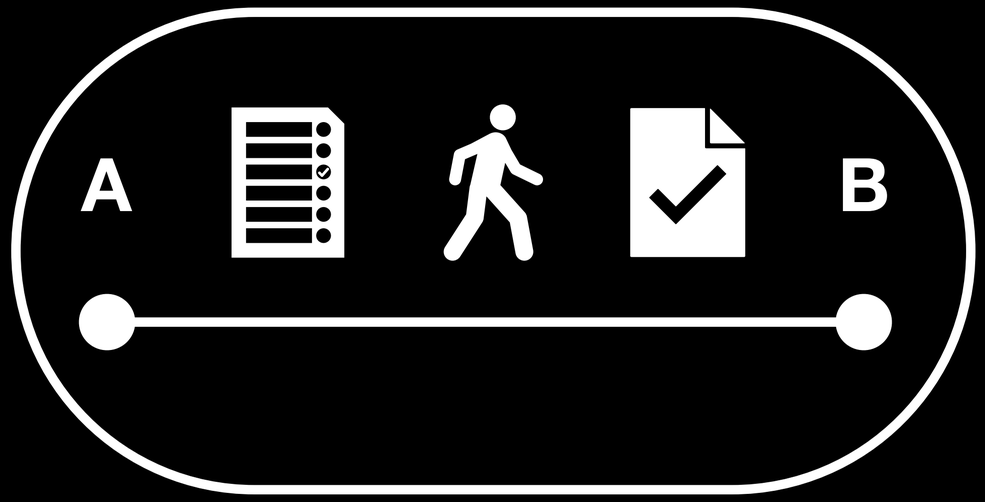
Test №1



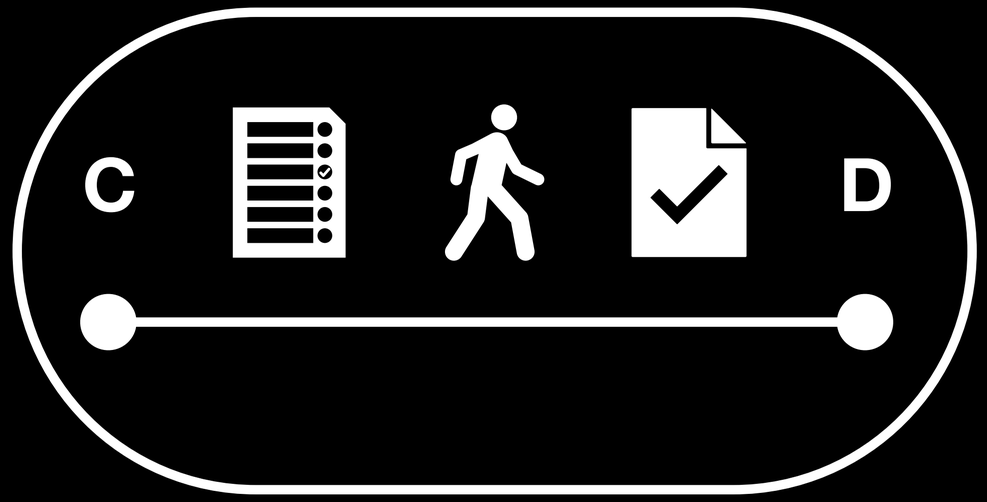
Test №2



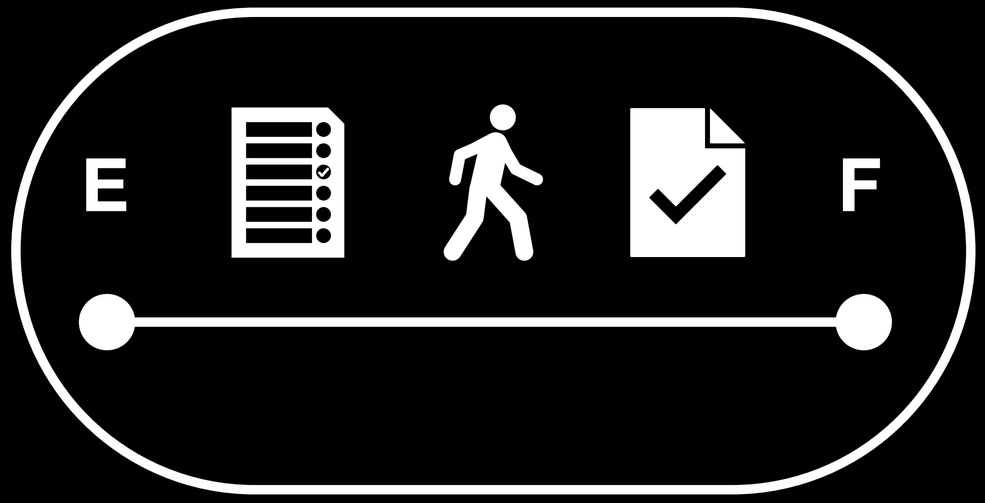
Test №1



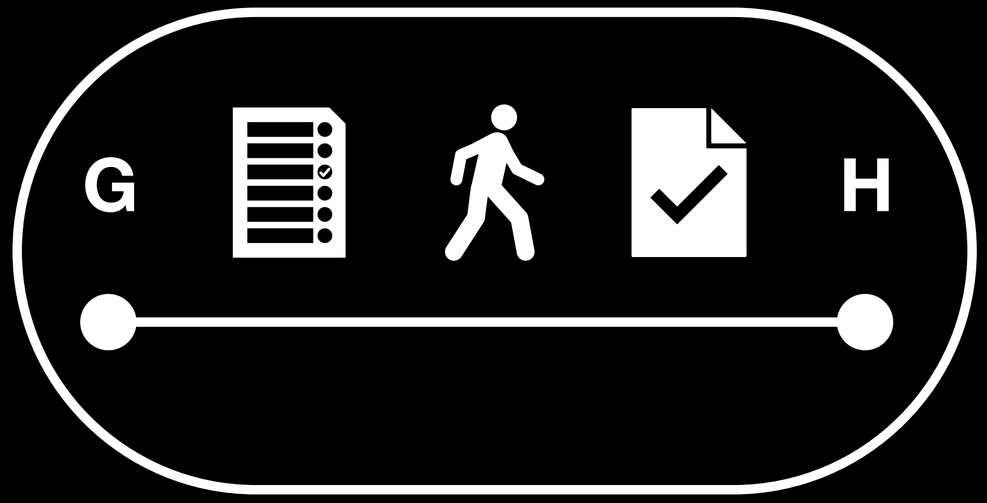
Test №3

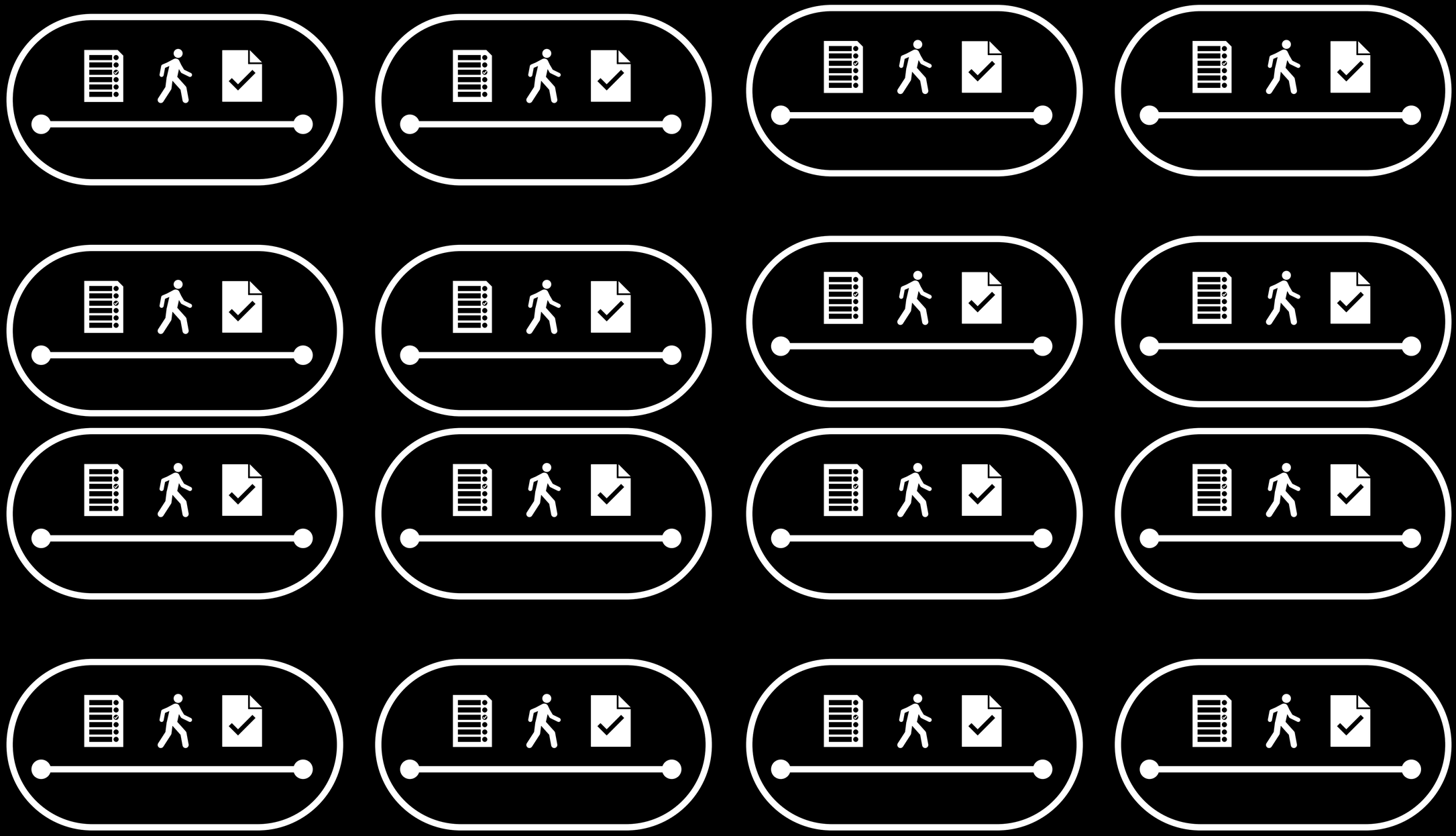


Test №2

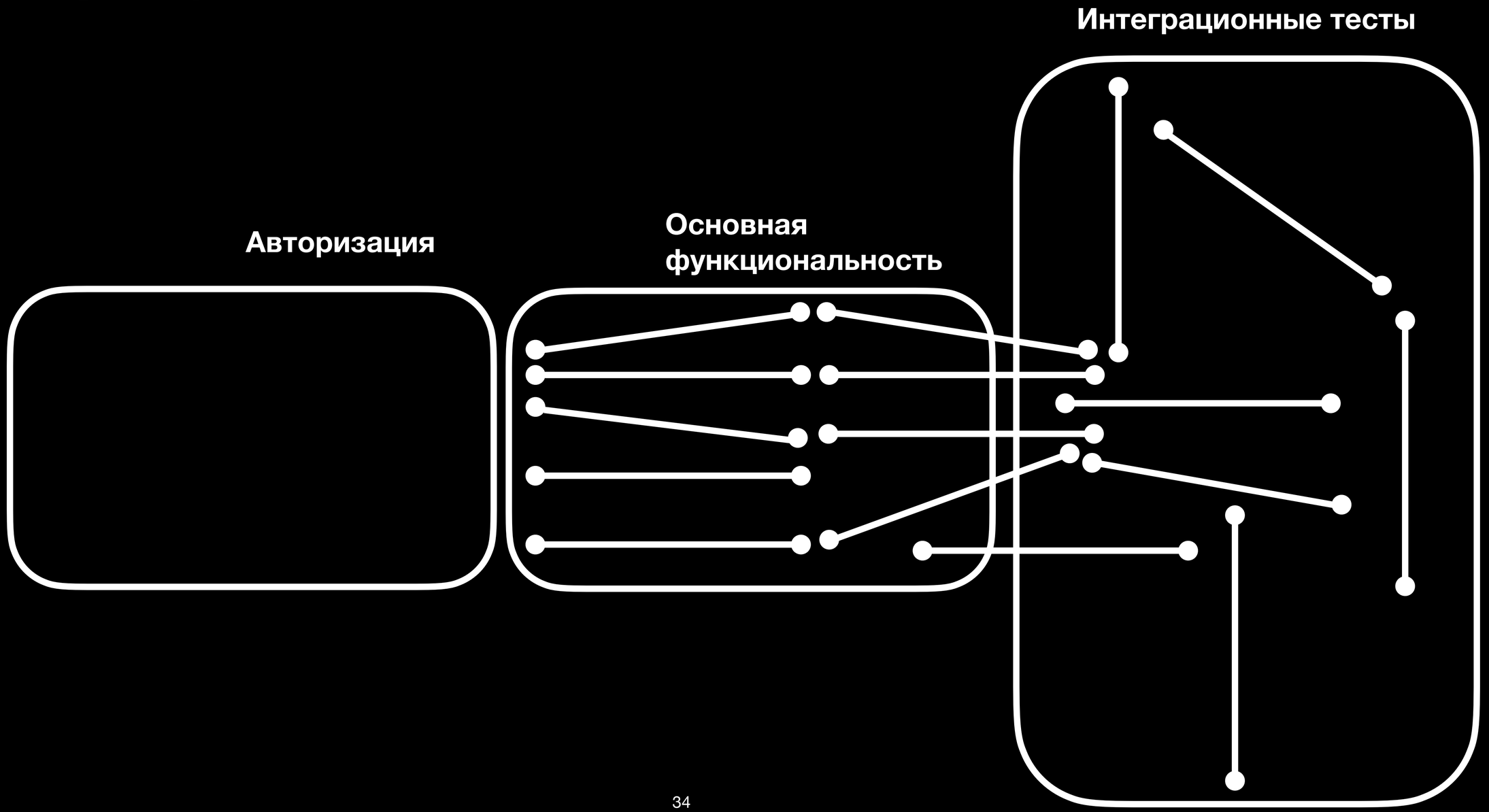


Test №4

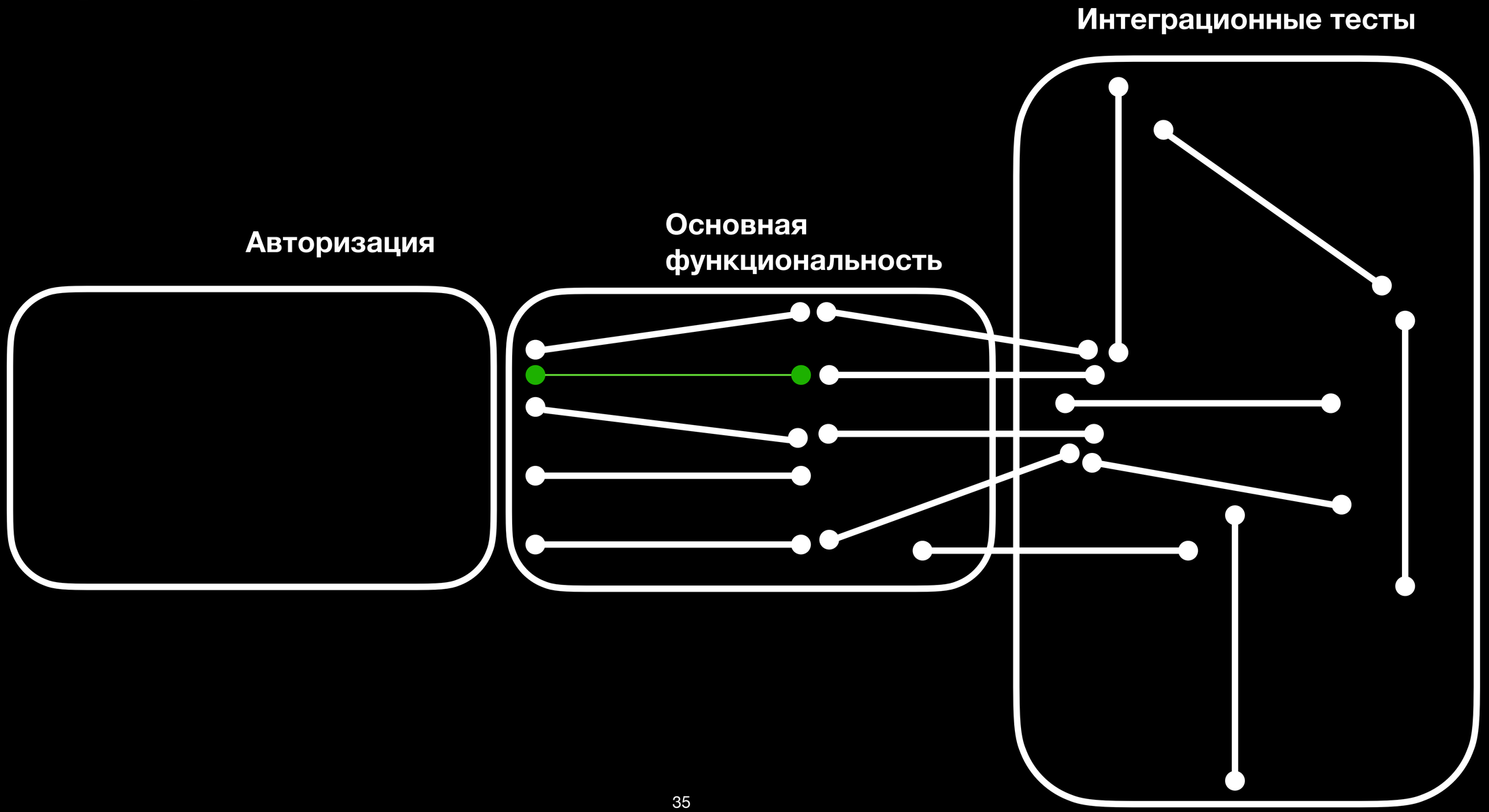




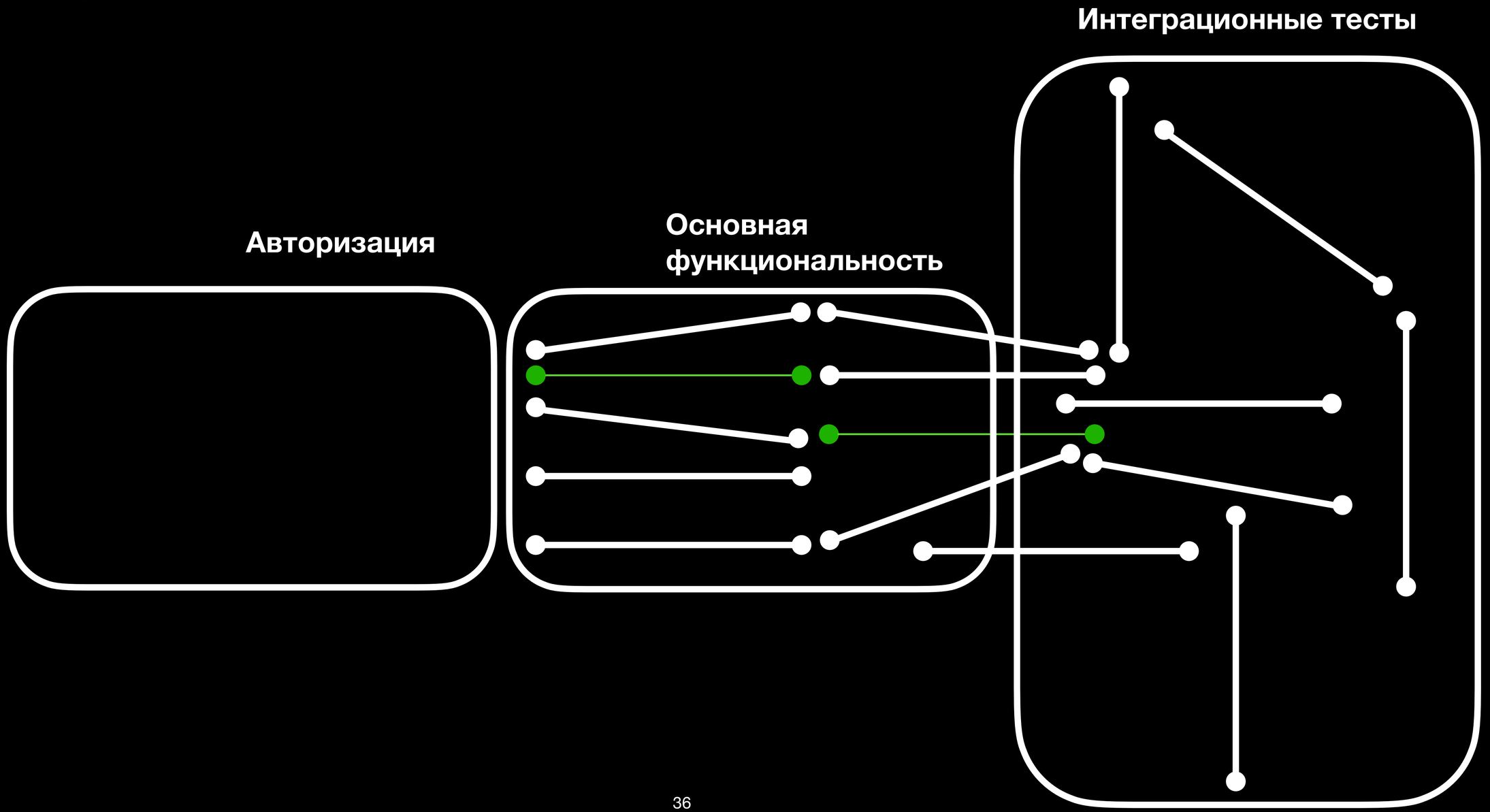
Наш продукт


Интеграционные тесты
Авторизация
Основная
функциональность

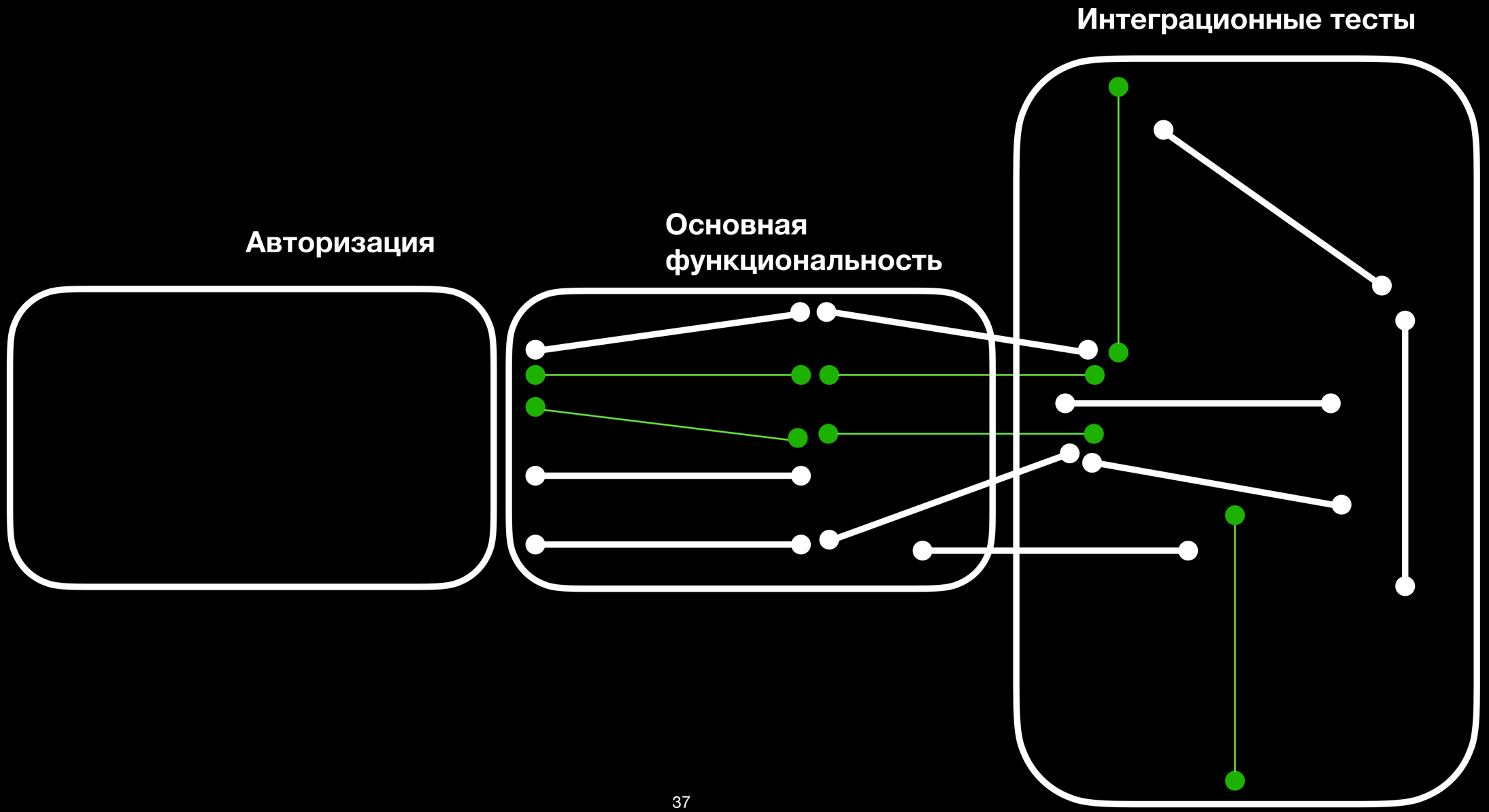
Наш продукт


Интеграционные тесты
Авторизация
Основная
функциональность

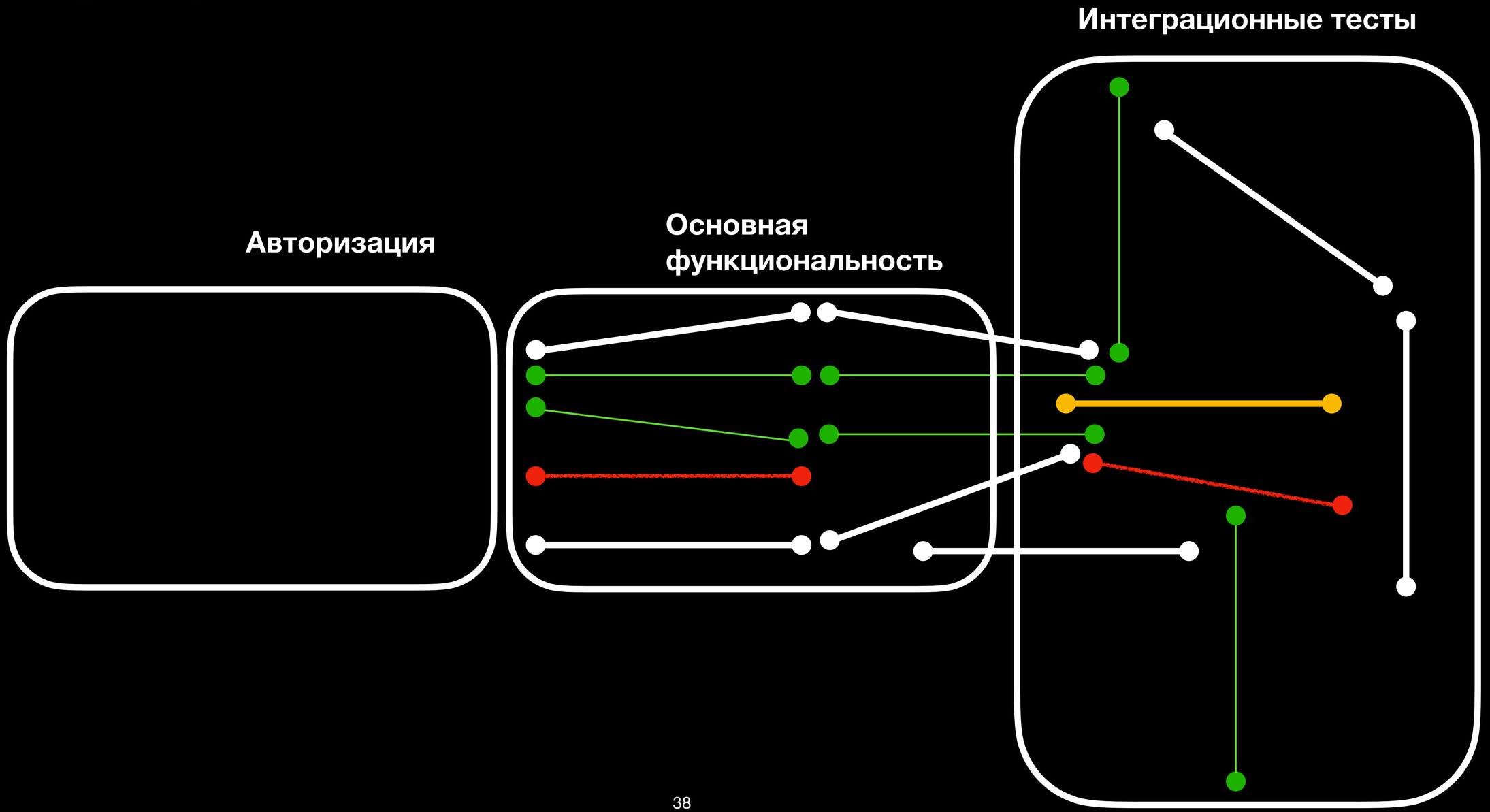
Наш продукт


Интеграционные тесты
Авторизация
Основная
функциональность

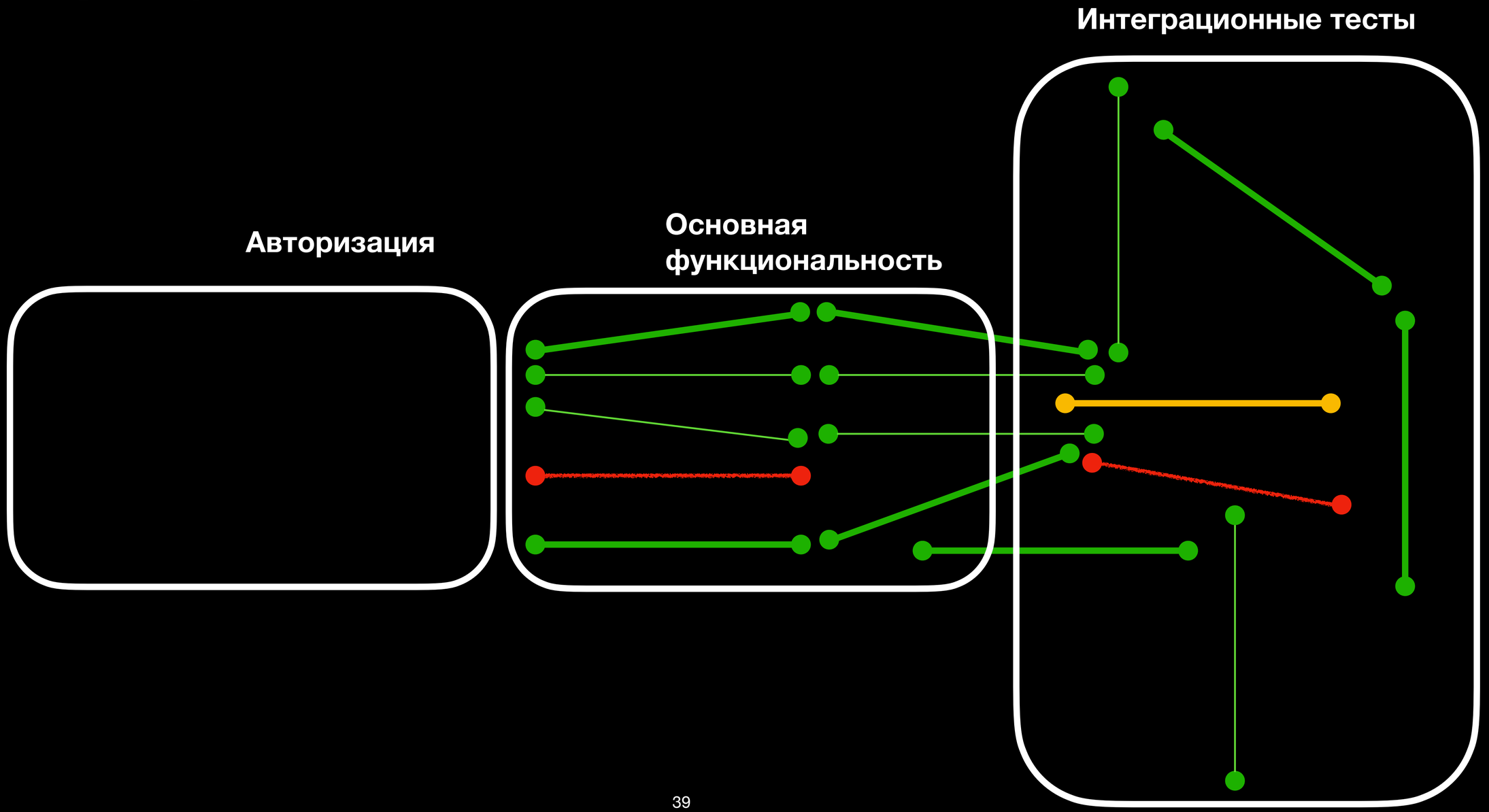
Наш продукт


Интеграционные тесты
Авторизация
Основная
функциональность

Наш продукт


Интеграционные тесты
Авторизация
Основная
функциональность

Наш продукт


Интеграционные тесты
Авторизация
Основная
функциональность

Проблемы

Проблемы “Академического тестирования”

Гибкость :

при изменениях в проекте требуется править каждый тест

```python
import pytest
from selenium import webdriver

# Фикстура для инициализации и завершения WebDriver
@pytest.fixture
def browser():
    driver = webdriver.Chrome()
    yield driver
    driver.quit()

# Тестовый случай: проверка заголовка главной страницы Google
def test_google_title(browser):
    url = "https://www.google.com"
    browser.get(url)
    assert browser.title == "Google"

# Тестовый случай: выполнение поиска в Google
def test_google_search(browser):
    url = "https://www.google.com"
    browser.get(url)
    search_box = browser.find_element("name", "q")
    search_box.send_keys("pytest")
    search_box.submit()
    assert "pytest" in browser.title
```

Проблемы “Академического тестирования”

Отсутствие попарного тестирования :

мы не можем попарно проверять и комбинировать кейсы разных тестов

```
@pytest.mark.parametrize("cc, sq, es", [
    ("com", "news", "cnn.com"),
    ("fr", "nouvelles", "lemonde.fr"),
    ("pl", "wiadomości", "wp.pl")
])
def test_search_by_country(browser, cc, sq, es):
    browser.get(f"https://www.google.{cc}")
    sb = browser.find_element(By.NAME, "q")
    sb.send_keys(sq)
    sb.submit()
    assert es in browser.page_source

@pytest.mark.parametrize("ui, sq, er", [
    ("kitchen knives", "knife", "kitchen knife"),
    ("jungle equipment", "knife", "machete")
])
def test_search_by_interest(browser, ui, sq, er):
    browser.get("https://www.google.com")
    simulate_user_interest(browser, ui)
    sb = browser.find_element(By.NAME, "q")
    sb.send_keys(sq)
    sb.submit()
    assert er in browser.page_source
```

Проблемы “Академического тестирования”

Высокий порог вхождения:

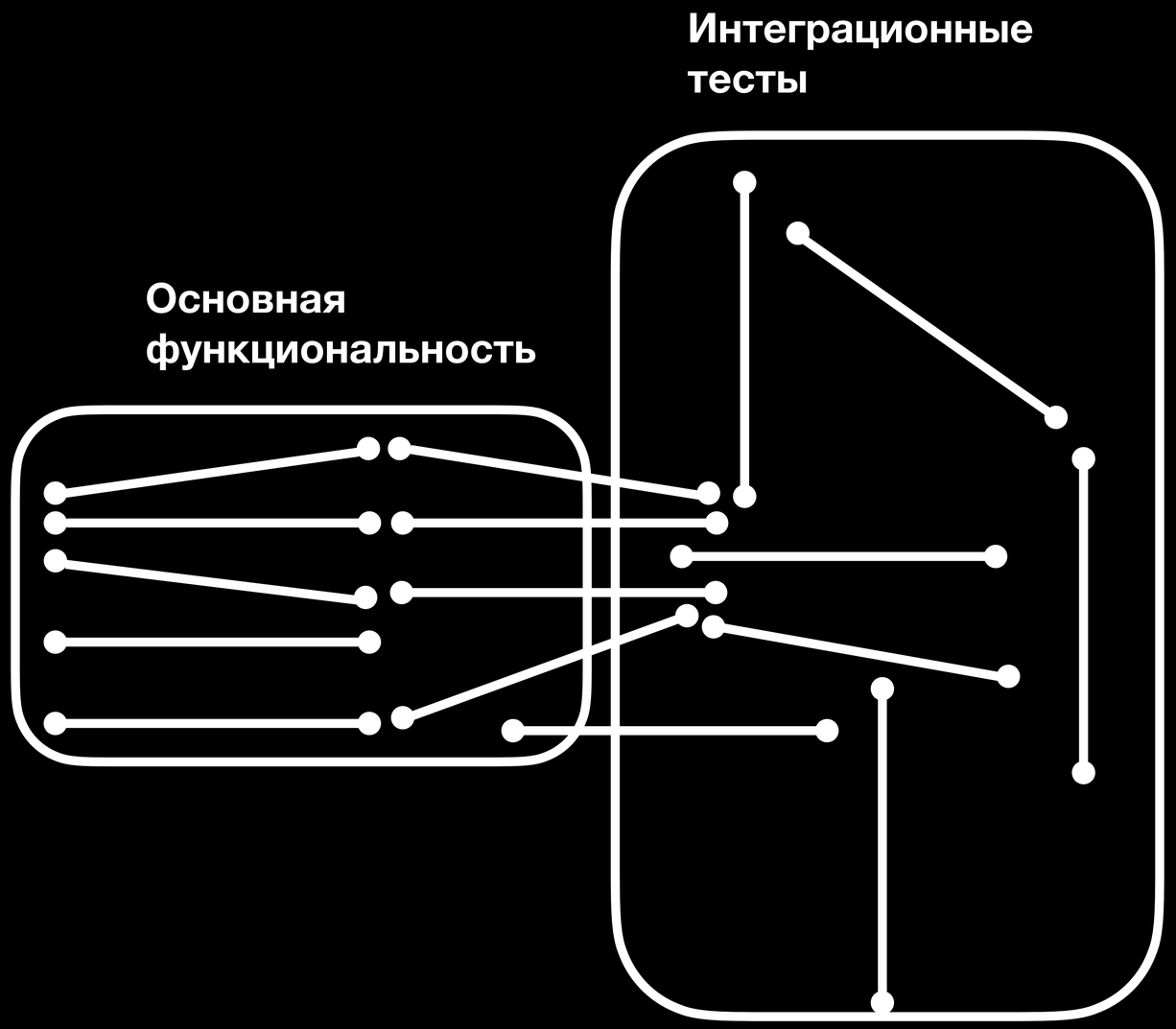
даже для добавления одного кейса или исправления старого понадобится разработчик с экспертизой

```
import unittest
import pytest
import selenium
from selenium.webdriver.common.by import By
from selenium.webdriver.common.keys import Keys
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
import requests
import allure
import locust
import faker
import logging
import Appium-Python-Client
import robotframework
import behave
import Factory_boy
import Hypothesis
import Splinter
import nose
import TestComplete
import mock
import tox
import coverage
import parameterized
import xmlrunner
```

Проблемы “Академического тестирования”

Мы не можем понять как покрыт продукт:

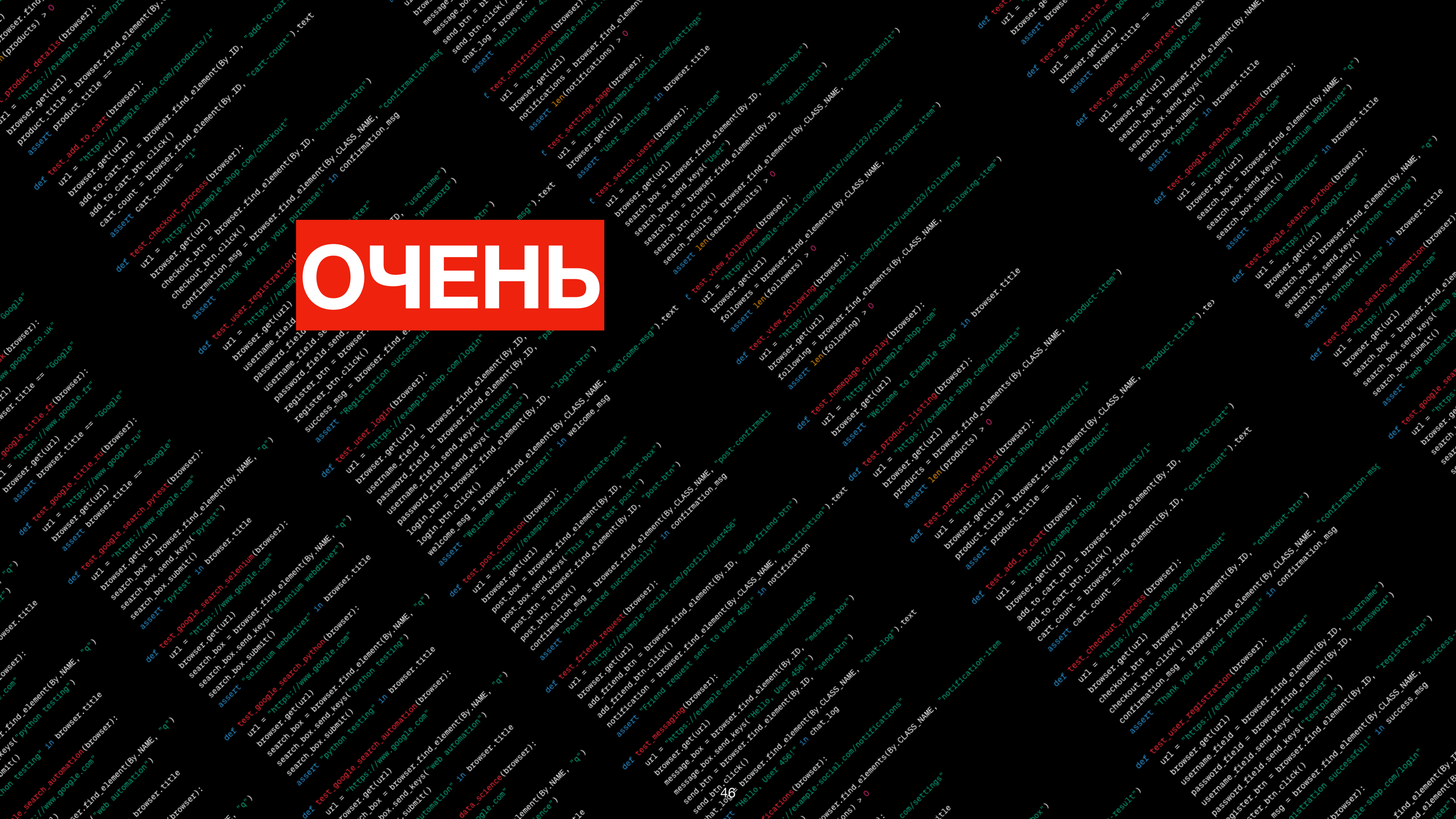
Нет такой структуры у тестов, которая просто и быстро покажет какая функциональность тестируется

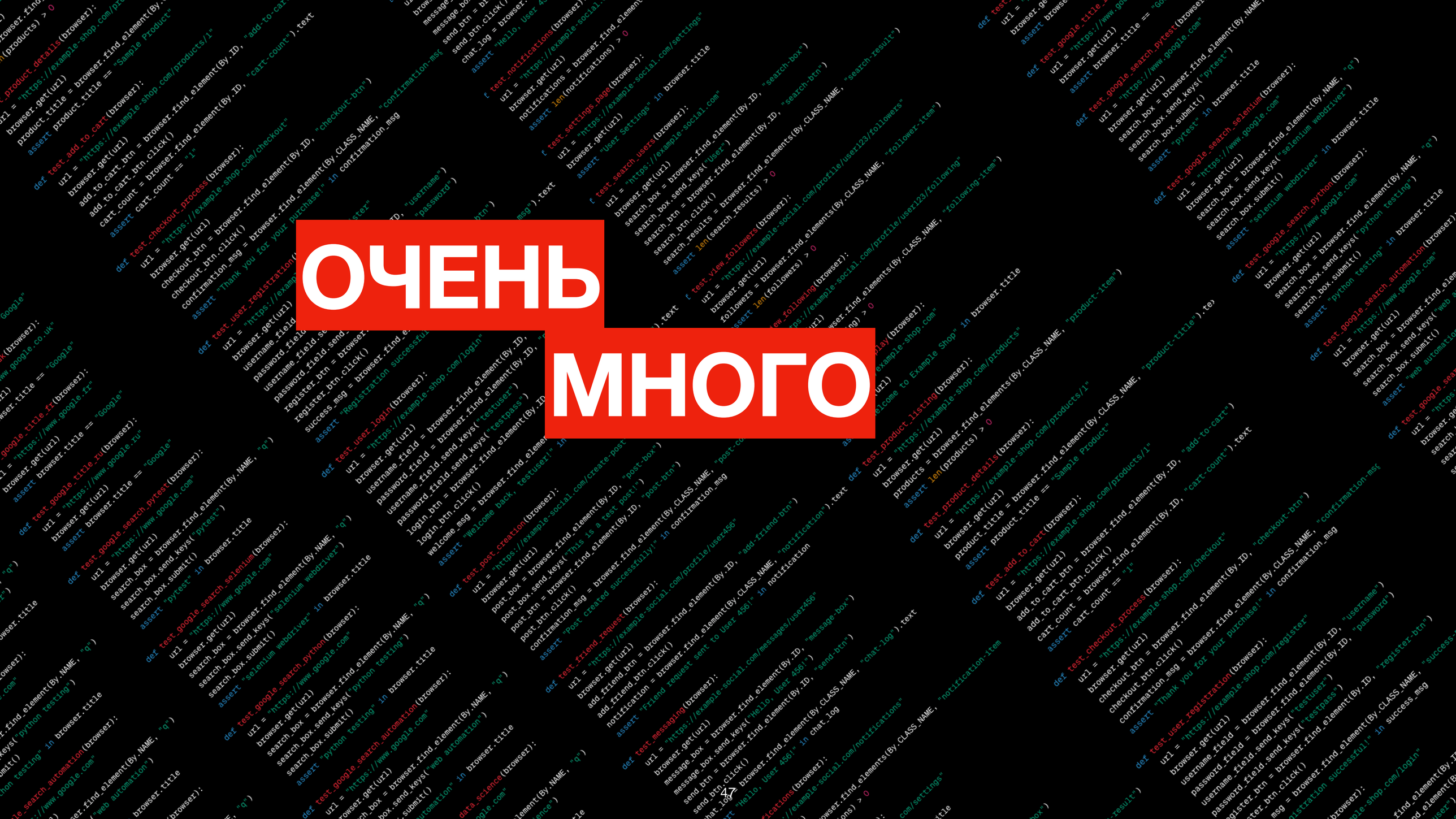


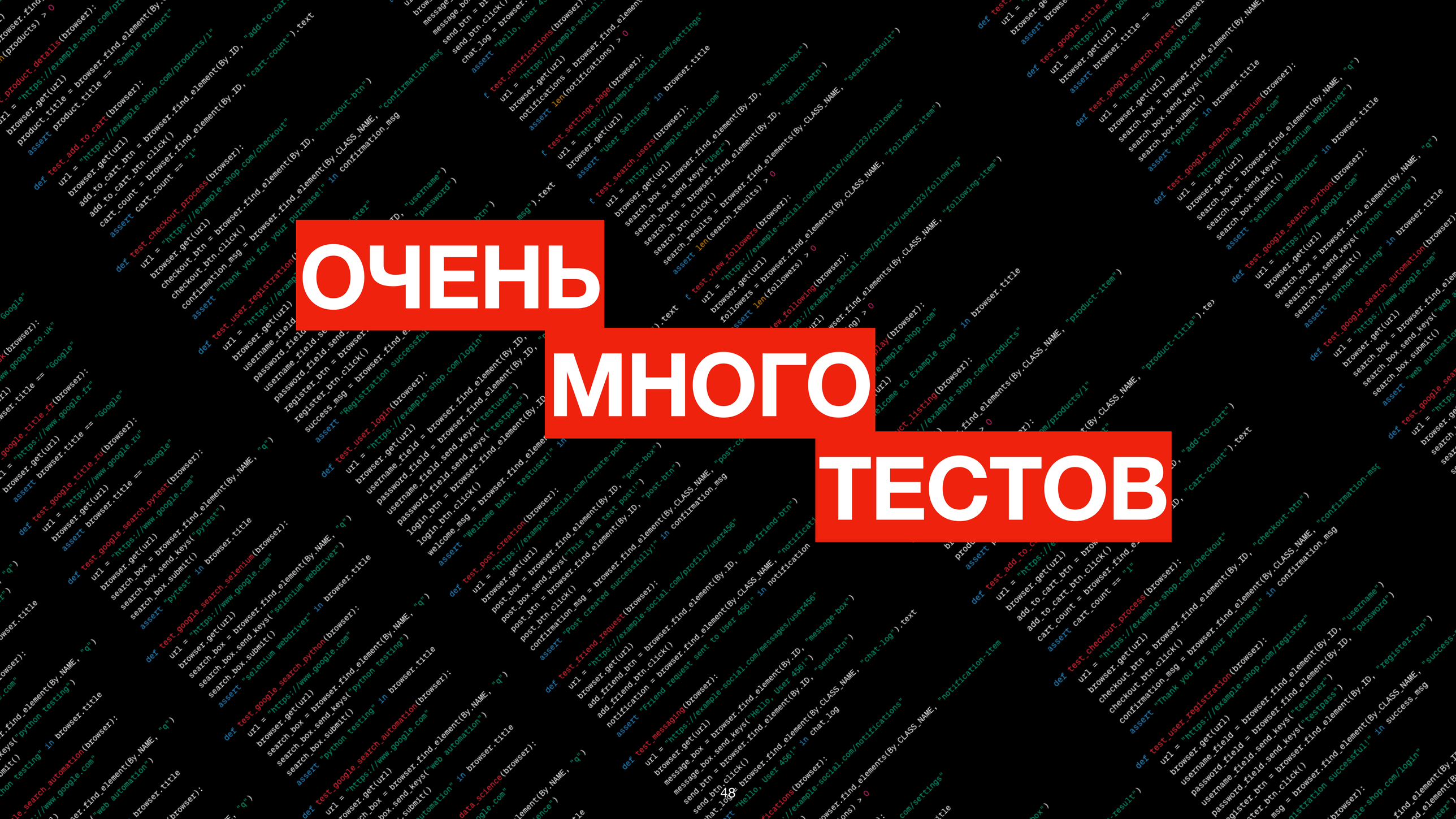
Проблемы “Академического тестирования”

Отсутствие отложенных проверок:

Практически нереально организовать проверку автоматики корректно и в будущем


ОЧЕНЬ


ОЧЕНЬ
МНОГО


ОЧЕНЬ
МНОГО
ТЕСТОВ

Что делать?

Что делать?

Ждать ИИ

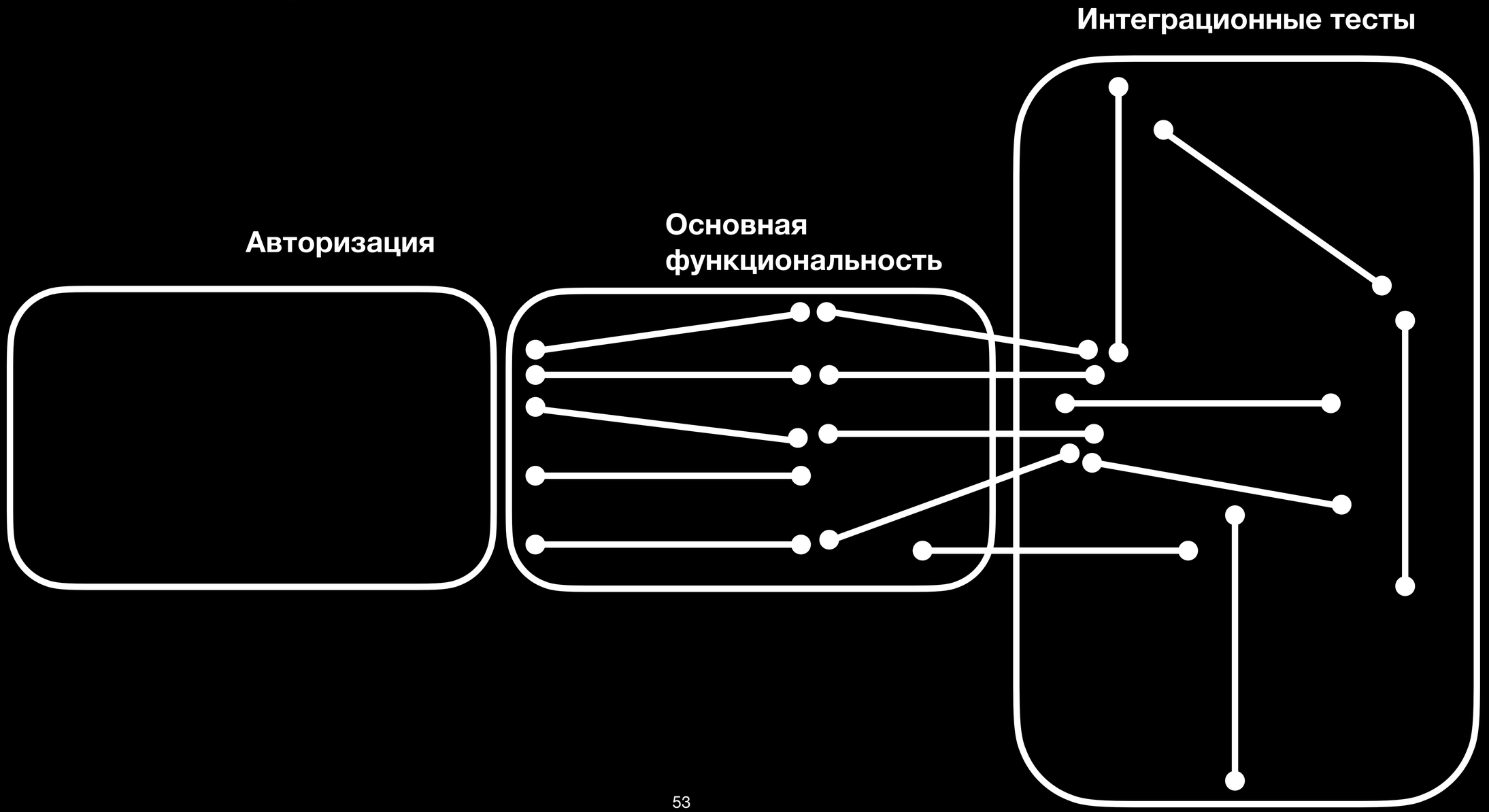
Что делать?

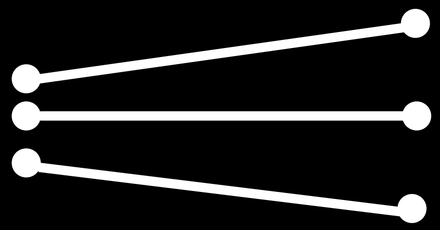
Оптимизировать

Ждать ИИ

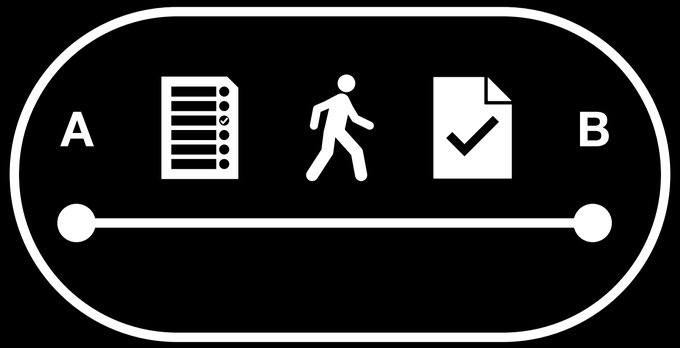
Композиционное тестирование

Почтовый сервис

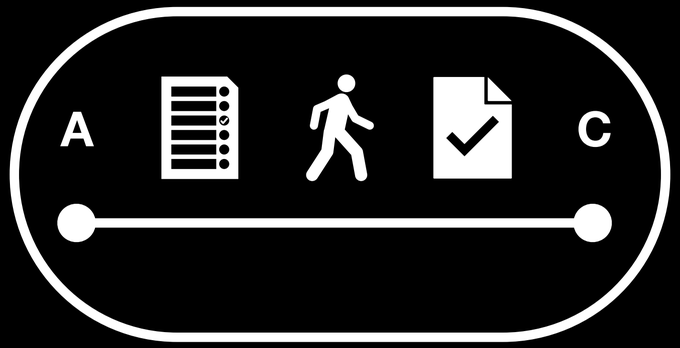




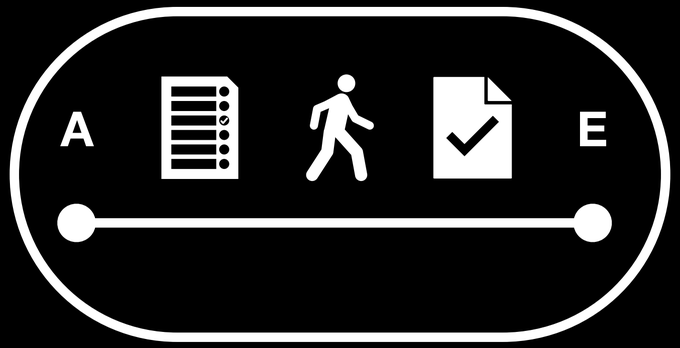
Тест отправки по городу



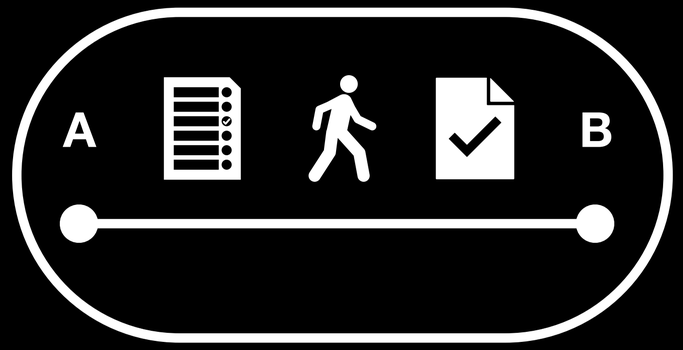
Тест междугородней отправки



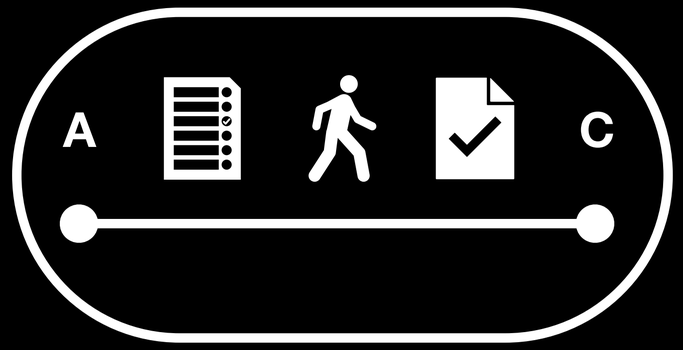
Тест отправки в другую страну



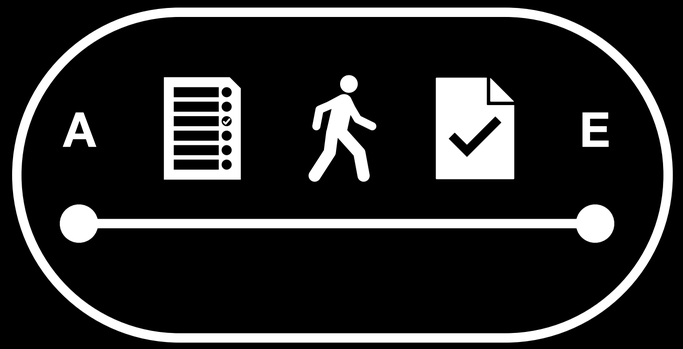
Тест отправки по городу



Тест междугородней отправки



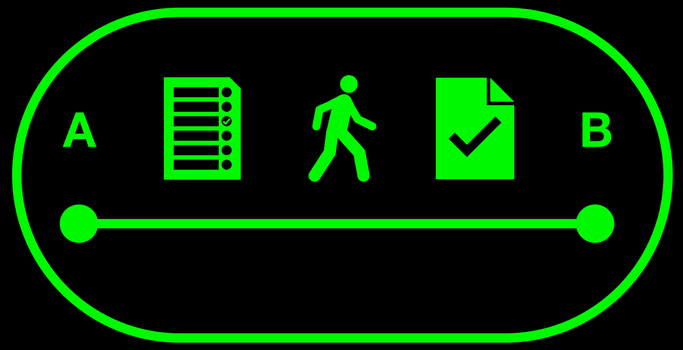
Тест отправки в другую страну



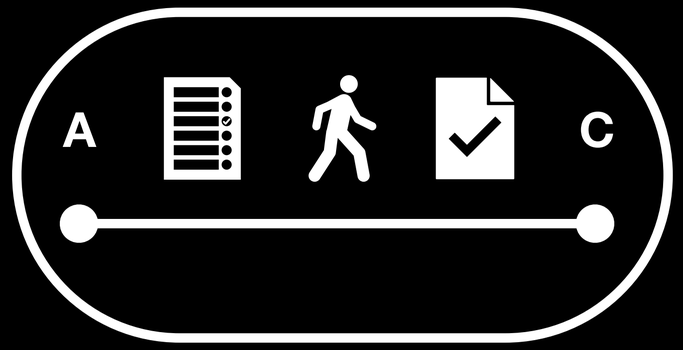
POST: "http://dhl.test.com/api/mail/msk_post"

```
{
   "name": "Иван Иванов",
   "weight": 85083830,
   "rate": "light",
   "dimentions": "2x51x9",
   "address": {
      "street": "ул. Центральная",
      "city": "Москва",
      "zipcode": "101100"
   },
   "phoneNumbers": [
      "+71234590099"
   ]
}
```

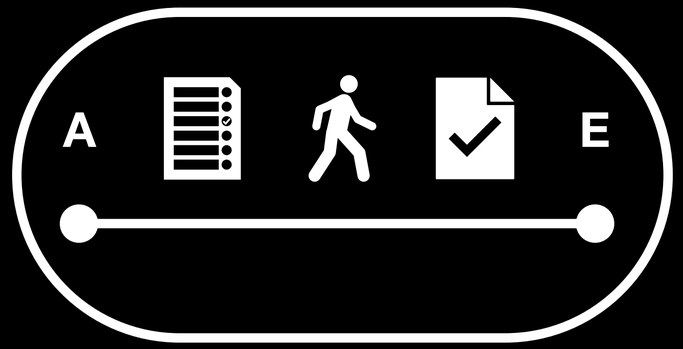
Тест отправки по городу



Тест междугородней отправки



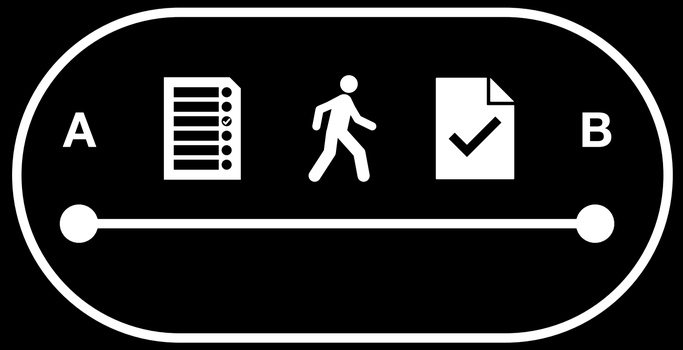
Тест отправки в другую страну



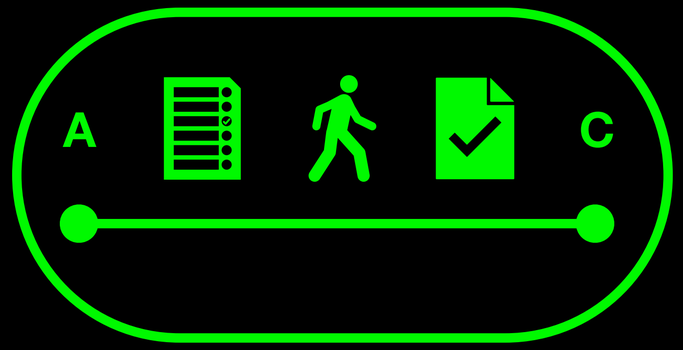
POST: "http://dhl.test.com/api/mail/msk_post"

```
{
   "name": "Иван Иванов",
   "weight": 85083830,
   "rate": "light",
   "dimentions": "2x51x9",
   "address": {
      "street": "ул. Центральная",
      "city": "Москва",
      "zipcode": "101100"
   },
   "phoneNumbers": [
      "+71234590099"
   ]
}
```

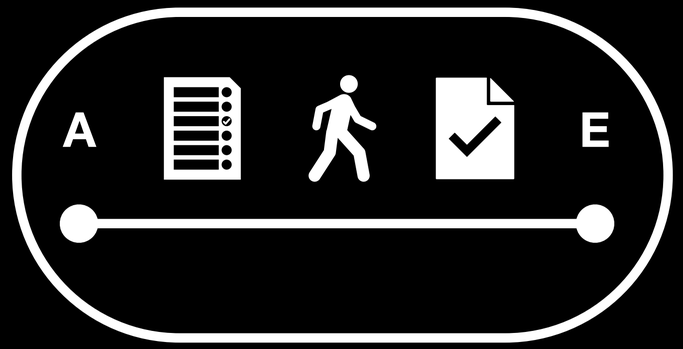
Тест отправки по городу



Тест междугородней отправки



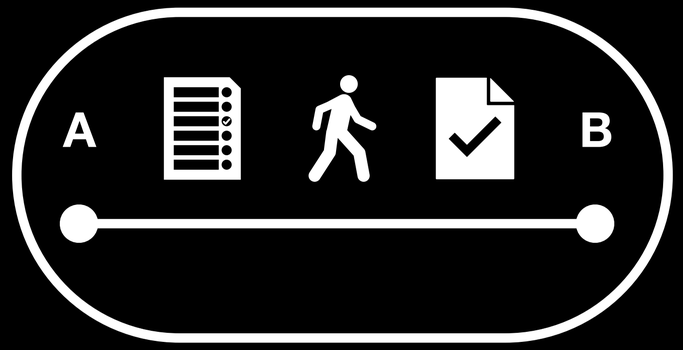
Тест отправки в другую страну



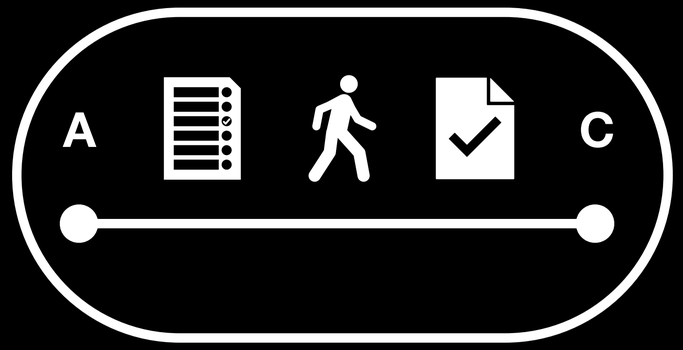
POST: "http://dhl.test.com/api/mail/msk_post"

```
{
   "name": "Иван",
   "weight": 7180,
   "rate": "light",
   "dimentions": "3x2x4",
   "address": {
      "street": "ул. Красно-белая",
      "city": "Железногорск",
      "zipcode": "101207"
   },
   "phoneNumbers": [
      "+7-123-456-78-90"
   ]
}
```

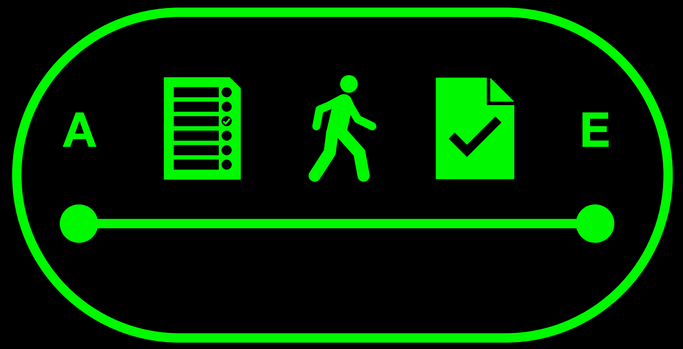
Тест отправки по городу



Тест междугородней отправки



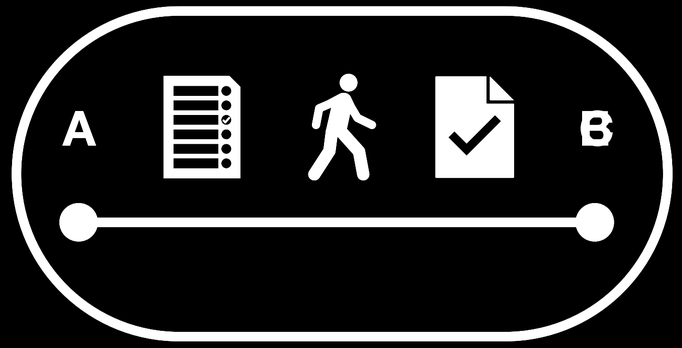
Тест отправки в другую страну



POST: "http://dhl.test.com/api/mail/msk_post"

```
{
   "name": "Иван-Петр",
   "weight": 7180,
   "rate": "heavy",
   "dimentions": "3x5x2",
   "address": {
      "street": "Abashidze st.",
      "city": "Tbilisi",
      "zipcode": "1100"
   },
   "phoneNumbers": [
      "+71234567890",
      "+95509876543321"
   ]
}
```

Тест отправки почты



Тест отправки почты

POST: "http://dhl.test.com/api/mail/post"

Тест отправки почты

POST: "http://dhl.test.com/api/mail/post"

GET: "http://dhl.test.com/api/mail"

Тест отправки почты

POST: "http://dhl.test.com/api/mail/post"

GET: "http://dhl.test.com/api/mail"

```
{
   "name": "Иван",
   "weight": 4354,
   "rate": "loyal",
   "dimentions": "3x13x9",
   "address": {
      "street": "ул. Центральная",
      "city": "Москва",
      "zipcode": "101101"
   },
   "phoneNumbers": [
      "+7-123-456-78-90"
   ]
}
```

Тест отправки почты

POST: "http://dhl.test.com/api/mail/post"

GET: "http://dhl.test.com/api/mail"

```
{
   "name": "Иван Иванов",
   "weight": 4354,
   "rate": "light",
   "dimentions": "3x43x8",
   "address": {
      "street": "тракт Старый",
      "city": "Москва",
      "zipcode": "101101"
   },
   "phoneNumbers": [
      "+71234589999"
   ]
}
```

Тест отправки почты

POST: "http://dhl.test.com/api/mail/post"

GET: "http://dhl.test.com/api/mail"


```
{
   "name": "Анна-Мария",
   "weight": 34860108,
   "rate": "loyal",
   "dimentions": "4165x372836x19633",
   "address": {
      "street": "ул. Красно-белая",
      "city": "Москва",
      "zipcode": "101100"
   },
   "phoneNumbers": [
      "+7-123-456-78-90"
   ]
}
```

Тест отправки почты

POST: "http://dhl.test.com/api/mail/post"

GET: "http://dhl.test.com/api/mail"

```
{
   "name": "Иван-Петр",
   "weight": 34860108,
   "rate": "heavy",
   "dimentions": "86x8x6",
   "address": {
      "street": "ул. Красно-белая",
      "city": "Москва",
      "zipcode": "101101"
   },
   "phoneNumbers": [
      "+7 123 456 78 90"
   ]
}
```

Тест отправки почты

POST: "http://dhl.test.com/api/mail/post"

GET: "http://dhl.test.com/api/mail"


```
{
   "name": "Артемий Лебедев",
   "weight": 4354,
   "rate": "light",
   "dimentions": "7x8x1",
   "address": {
      "street": "тракт Старый",
      "city": "Москва",
      "zipcode": "101101"
   },
   "phoneNumbers": [
      "+71234567890",
      "+70987654321"
   ]
}
```

Тест отправки почты

POST: "http://dhl.test.com/api/mail/post"

GET: "http://dhl.test.com/api/mail"


```
{
   "name": "Артемий Лебедев",
   "weight": 4354,
   "rate": "light",
   "dimentions": "7x8x1",
   "address": {
      "street": "тракт Старый",
      "city": "Москва",
      "zipcode": "101101"
   },
   "phoneNumbers": [
      "+71234567890",
      "+70987654321"
   ]
}
```

Тест отправки почты

POST: "http://dhl.test.com/api/mail/post"

GET: "http://dhl.test.com/api/mail"

```
{
    "name": "Иван Иванов",
    "weight": 169,
    "rate": "light",
    "dimentions": "34x53x5",
    "address": {
        "street": "ул. Центральная",
        "city": "Москва",
        "zipcode": "101101"
    },
    "phoneNumbers": [
        "+71234567890",
        "+70987654321"
    ]
}
```

Тест отправки почты

POST: "http://dhl.test.com/api/mail/post"

GET: "http://dhl.test.com/api/mail"


```
{
   "name": "Иван",
   "weight": 579626,
   "rate": "light",
   "dimentions": "695x82742x4812126",
   "address": {
      "street": "ул. Центральная",
      "city": "Нижний Новгород",
      "zipcode": "101207"
   },
   "phoneNumbers": [
      "1234567890"
   ]
}
```

Тест отправки почты

POST: "http://dhl.test.com/api/mail/post"

GET: "http://dhl.test.com/api/mail"

```
{
   "name": "Анна-Мария",
   "weight": 169,
   "rate": "light",
   "dimentions": "6917x212x795",
   "address": {
      "street": "проспект Мира",
      "city": "Москва",
      "zipcode": "101101"
   },
   "phoneNumbers": [
      "+71234589999"
   ]
}
```

Движки для таких тестов на Python

Tavern

Hypothesis

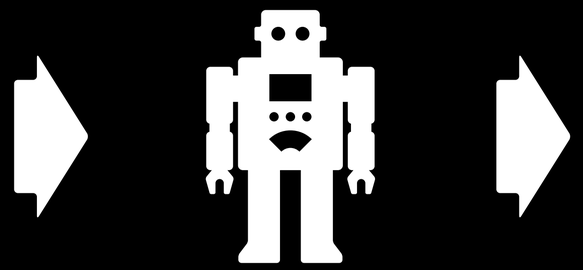
Pytest-cases

Combidata

Combidata

Combidata

```
{
    "cases": cases_dict,
    "workflow": testing_flow,
    "tools": my_test_tools,
    "template": web_data
}
```



1) Кейс прошел успешно
2) Тест упал
3) Кейс прошел успешно
4) Неверный результат

.....

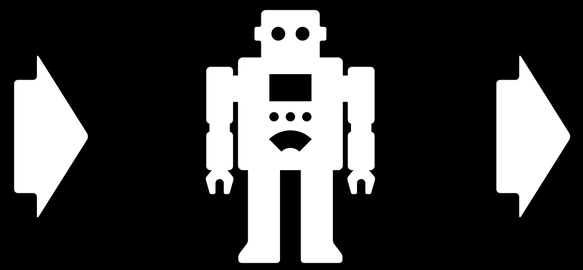
n) Кейс прошел успешно

"cases"

```
unit_dict["cases"]["rate"] = {
    "HV": {
        "value": "heavy",
        "requirements": {"weight": "Hig"},
        "name": "Грузовой тариф"
    },
    "LI": {
        "value": "light",
        "requirements": {"weight": "Gen"},
        "name": "Перевозка курьером"
    },
    "LO": {
        "value": "loyal",
        "requirements": {"dimentions": "Gen"},
        "name": "Спецпредложение"
    }
}
unit_dict["cases"]["weight"] = {
    "Gen": {
        "gen_func": random.randint,
        "options": {"a": 100, "b": 10000},
        "name": "Базовый тариф (вес)"
    },
    "Hig": {
        "gen_func": random.randint,
        "options": {"a": 10000, "b": 100000000},
        "name": "Грузовой тариф (вес)"
    }
}
```

Combidata

```
{
    "cases": cases_dict,
    "workflow": testing_flow,
    "tools": my_test_tools,
    "template": web_data
}
```



1) Кейс прошел успешно
2) Тест упал
3) Кейс прошел успешно
4) Неверный результат

.....

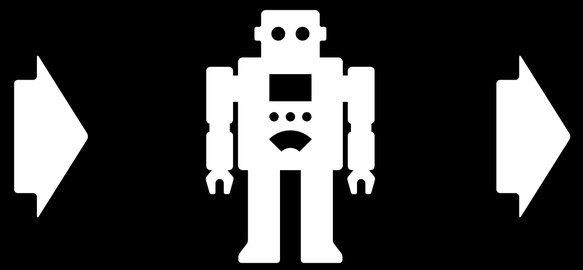
n) Кейс прошел успешно

"workflow"

```
{
    "standard": (ST_COMBINE, ST_GENERATE, ST_FORM, SEND, CHECK_WRI),
    "error": (ST_COMBINE, ST_GENERATE, ST_FORM, SEND, CHECK_WRO),
    "international": (ST_COMBINE, ST_GENERATE, ST_FORM, SEND, CHECK_INT),
    "generate": (ST_COMBINE, ST_GENERATE),
    "tell": [(ARC, MODAL), (CLEAR, CHECK_WRI)]
}
```

Combidata

```
{
    "cases": cases_dict,
    "workflow": testing_flow,
    "tools": my_test_tools,
    "template": web_data
}
```



1) Кейс прошел успешно
2) Тест упал
3) Кейс прошел успешно
4) Неверный результат

.....

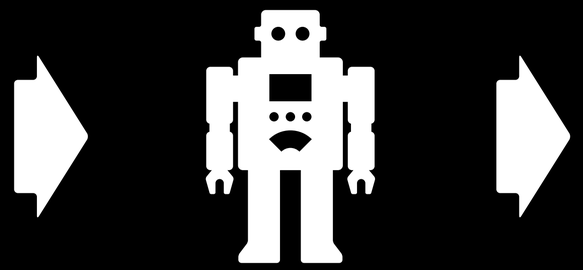
n) Кейс прошел успешно

"tools"

```
"selenium": selen,
"playwright": pw,
"dev_console": console_dev,
"logger": clogger,
"creds": CRED
```

Combidata

```
{
    "cases": cases_dict,
    "workflow": testing_flow,
    "tools": my_test_tools,
    "template": web_data
}
```



1) Кейс прошел успешно
2) Тест упал
3) Кейс прошел успешно
4) Неверный результат

.....

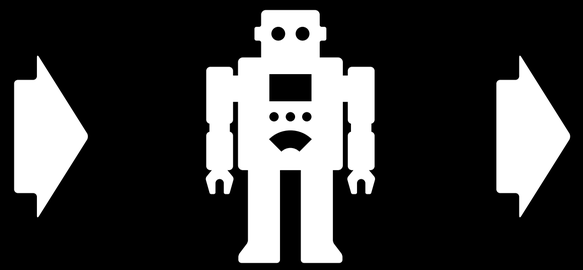
n) Кейс прошел успешно

"template"

```
{"name": "name",
 "weight": "age",
 "price": "price",
 "dimentions": "dimentions",
 "address": {
    "street": "street",
    "city": "city",
    "zipcode": "zipcode"
                        },
 "phoneNumbers": "phoneNumbers"}
```

Combidata

```
{
    "cases": cases_dict,
    "workflow": testing_flow,
    "tools": my_test_tools,
    "template": web_data
}
```



1) Кейс прошел успешно
2) Тест упал
3) Кейс прошел успешно
4) Неверный результат

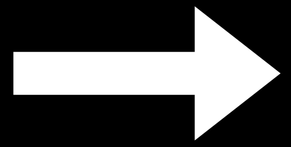
.....

n) Кейс прошел успешно

Пример стандартного запуска

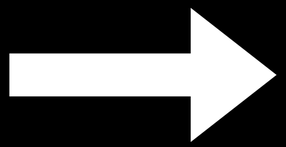
```
unit_dict["cases"]["weight"] = {
    "Gen": {
        "gen_func": random.randint,
        "options": {"a": 100, "b": 10000},
        "name": "Базовый тариф (вес)"
    },
    "Hig": {
        "gen_func": random.randint,
        "options": {"a": 10000, "b": 100000000},
        "name": "Грузовой тариф (вес)"
    }
}
unit_dict["cases"]["dimentions"] = {
    "Gen": {
        "gen_func": re_generate,
        "value": r"[1-9]{1,3}x[1-9]{1,3}x[1-9]{1,2}",
        "name": "Базовый тариф (размер)"
    },
    "Hig": {
        "gen_func": re_generate,
        "value": r"[1-9]{3,8}x[1-9]{3,8}x[1-9]{3,8}",
        "requirements": {"weight": "Hig"},
        "name": "Грузовой тариф (размер)"
    }
}
unit_dict["cases"]["street"] = {
    "StandardStreet": {
        "value": "ул. Центральная",
        "name": "Стандартное название улицы"
    },
    "StreetWithNumber": {
        "value": "ул. Центральная, 10",
        "name": "Улица с номером дома"
    },
    "StreetWithHyphen": {
```

Пример стандартного запуска



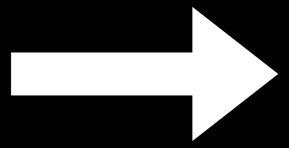
```
unit_dict["cases"]["weight"] = {
    "Gen": {
        "gen_func": random.randint,
        "options": {"a": 100, "b": 10000},
        "name": "Базовый тариф (вес)"
    },
    "Hig": {
        "gen_func": random.randint,
        "options": {"a": 10000, "b": 100000000},
        "name": "Грузовой тариф (вес)"
    }
}
unit_dict["cases"]["dimentions"] = {
    "Gen": {
        "gen_func": re_generate,
        "value": r"[1-9]{1,3}x[1-9]{1,3}x[1-9]{1,2}",
        "name": "Базовый тариф (размер)"
    },
    "Hig": {
        "gen_func": re_generate,
        "value": r"[1-9]{3,8}x[1-9]{3,8}x[1-9]{3,8}",
        "requirements": {"weight": "Hig"},
        "name": "Грузовой тариф (размер)"
    }
}
unit_dict["cases"]["street"] = {
    "StandardStreet": {
        "value": "ул. Центральная",
        "name": "Стандартное название улицы"
    },
    "StreetWithNumber": {
        "value": "ул. Центральная, 10",
        "name": "Улица с номером дома"
    },
    "StreetWithHyphen": {
```

Пример стандартного запуска



```
unit_dict["cases"]["weight"] = {
    "Gen": {
        "gen_func": random.randint,
        "options": {"a": 100, "b": 10000},
        "name": "Базовый тариф (вес)"
    },
    "Hig": {
        "gen_func": random.randint,
        "options": {"a": 10000, "b": 100000000},
        "name": "Грузовой тариф (вес)"
    }
}
unit_dict["cases"]["dimentions"] = {
    "Gen": {
        "gen_func": re_generate,
        "value": r"[1-9]{1,3}x[1-9]{1,3}x[1-9]{1,2}",
        "name": "Базовый тариф (размер)"
    },
    "Hig": {
        "gen_func": re_generate,
        "value": r"[1-9]{3,8}x[1-9]{3,8}x[1-9]{3,8}",
        "requirements": {"weight": "Hig"},
        "name": "Грузовой тариф (размер)"
    }
}
unit_dict["cases"]["street"] = {
    "StandardStreet": {
        "value": "ул. Центральная",
        "name": "Стандартное название улицы"
    },
    "StreetWithNumber": {
        "value": "ул. Центральная, 10",
        "name": "Улица с номером дома"
    },
    "StreetWithHyphen": {
```

Пример стандартного запуска

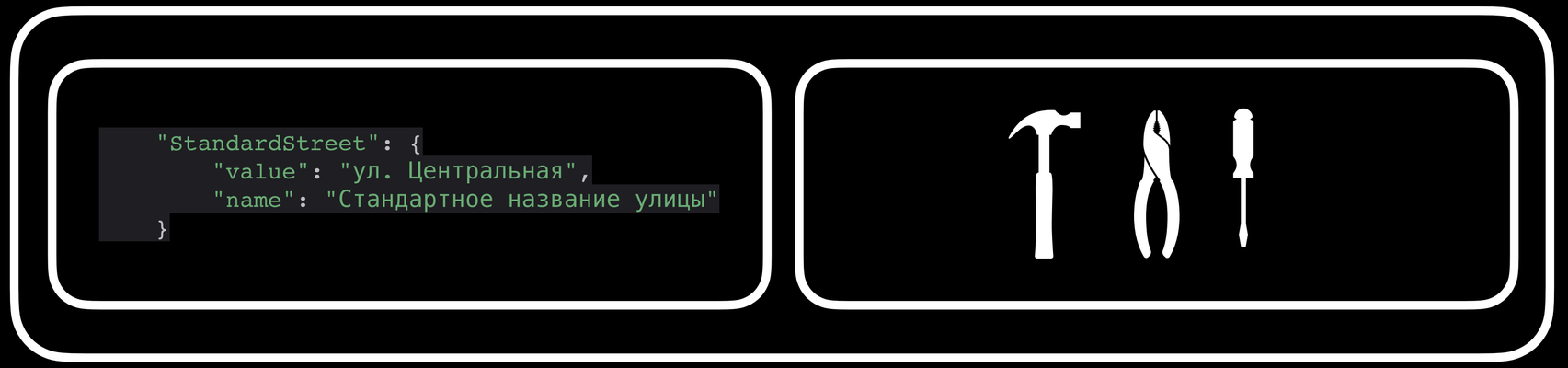


```
unit_dict["cases"]["weight"] = {
    "Gen": {
        "gen_func": random.randint,
        "options": {"a": 100, "b": 10000},
        "name": "Базовый тариф (вес)"
    },
    "Hig": {
        "gen_func": random.randint,
        "options": {"a": 10000, "b": 100000000},
        "name": "Грузовой тариф (вес)"
    }
}
unit_dict["cases"]["dimentions"] = {
    "Gen": {
        "gen_func": re_generate,
        "value": r"[1-9]{1,3}x[1-9]{1,3}x[1-9]{1,2}",
        "name": "Базовый тариф (размер)"
    },
    "Hig": {
        "gen_func": re_generate,
        "value": r"[1-9]{3,8}x[1-9]{3,8}x[1-9]{3,8}",
        "requirements": {"weight": "Hig"},
        "name": "Грузовой тариф (размер)"
    }
}
unit_dict["cases"]["street"] = {
    "StandardStreet": {
        "value": "ул. Центральная",
        "name": "Стандартное название улицы"
    },
    "StreetWithNumber": {
        "value": "ул. Центральная, 10",
        "name": "Улица с номером дома"
    },
    "StreetWithHyphen": {
```

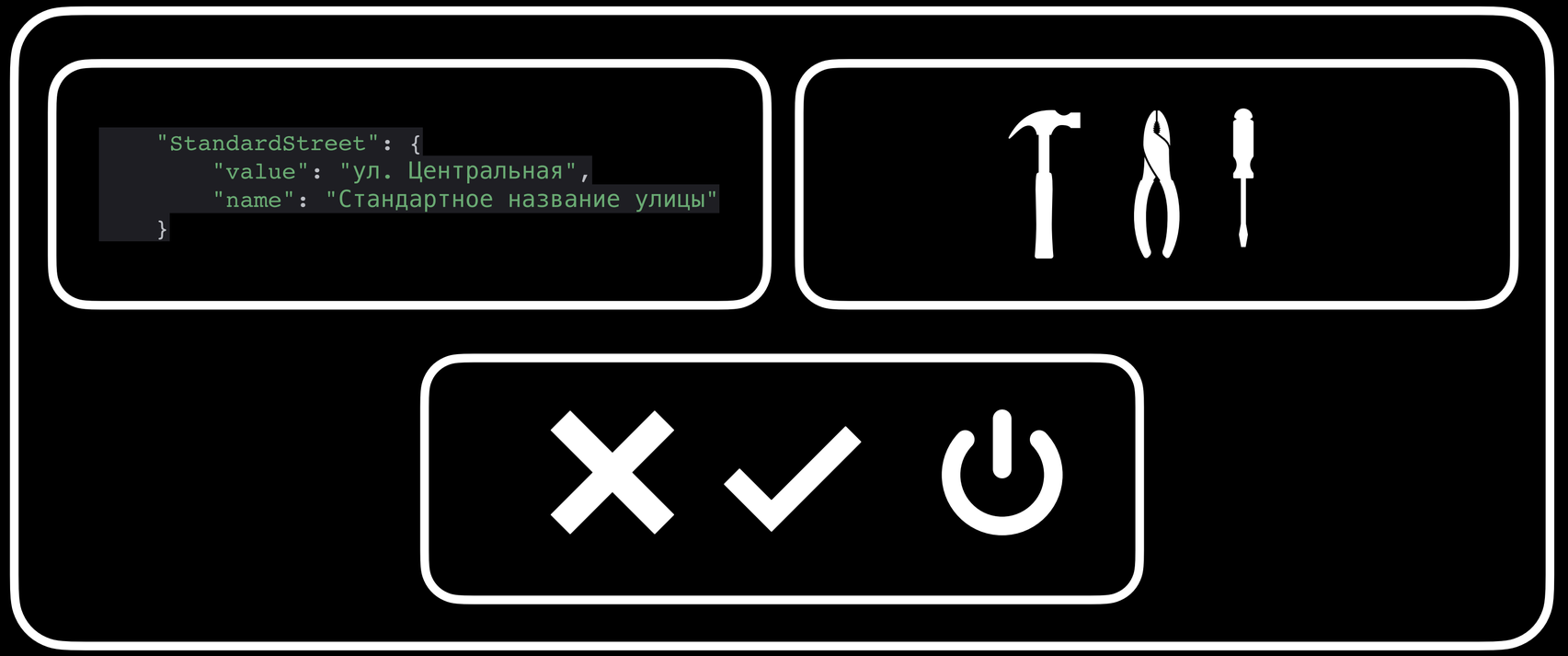
Пример стандартного запуска

```
    "StandardStreet": {
        "value": "ул. Центральная",
        "name": "Стандартное название улицы"
    }
```

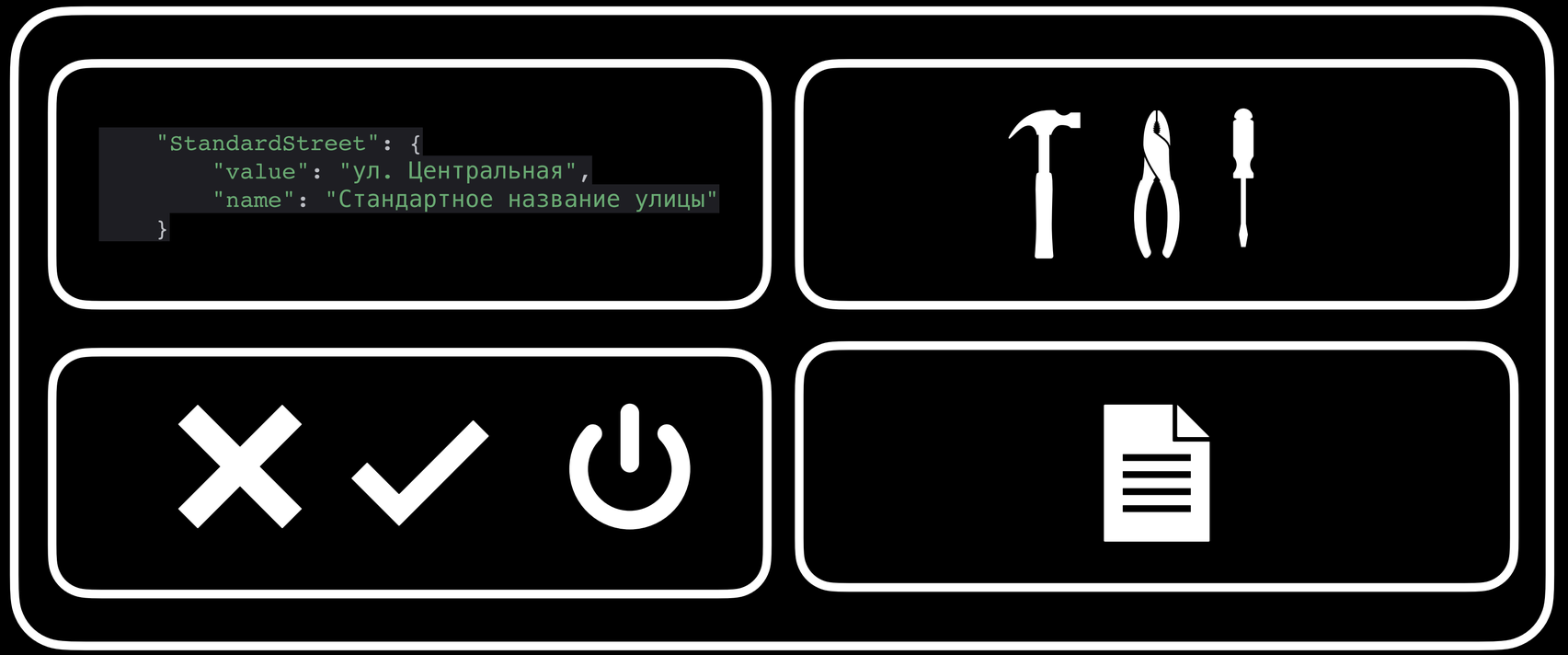
Пример стандартного запуска



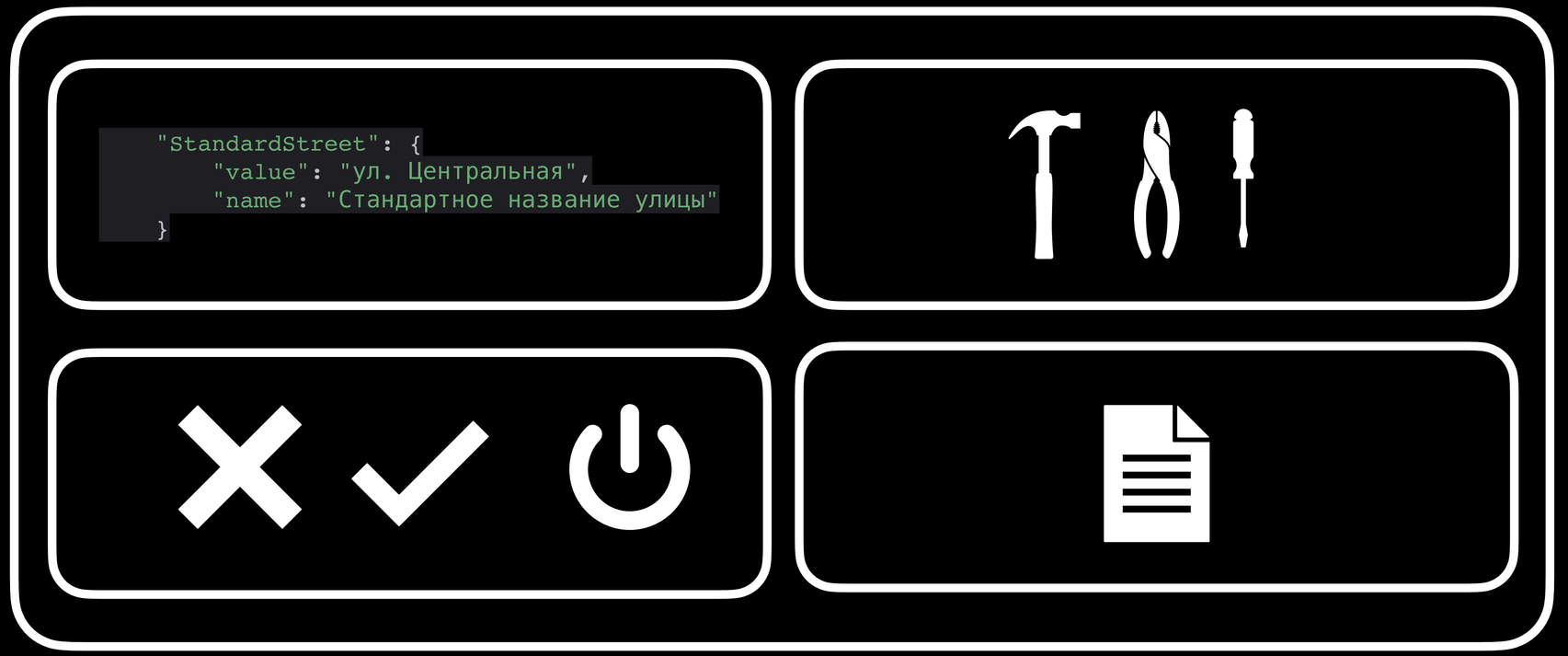
Пример стандартного запуска



Пример стандартного запуска

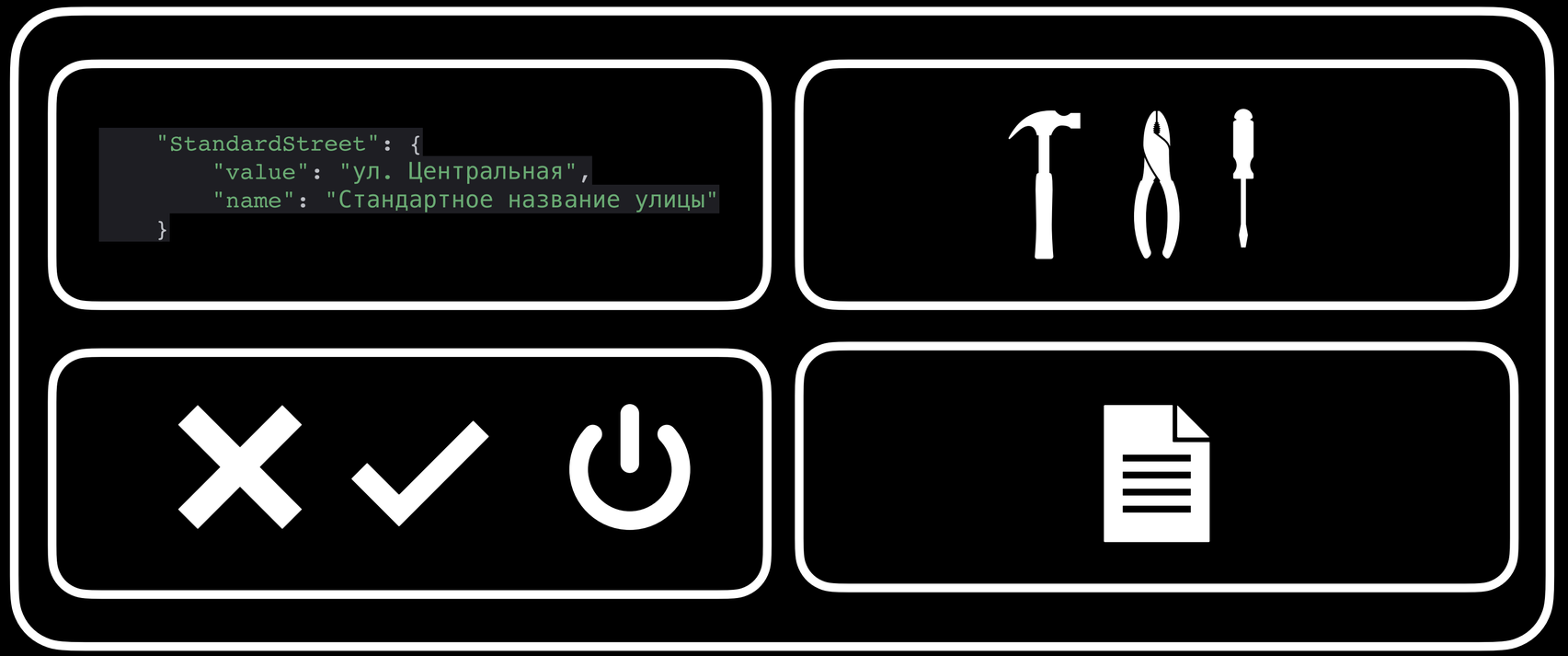


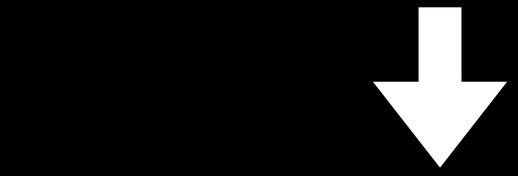
Пример стандартного запуска




```
"standard": (ST_COMBINE, ST_GENERATE, ST_FORM, SEND, CHECK_WRI)
```

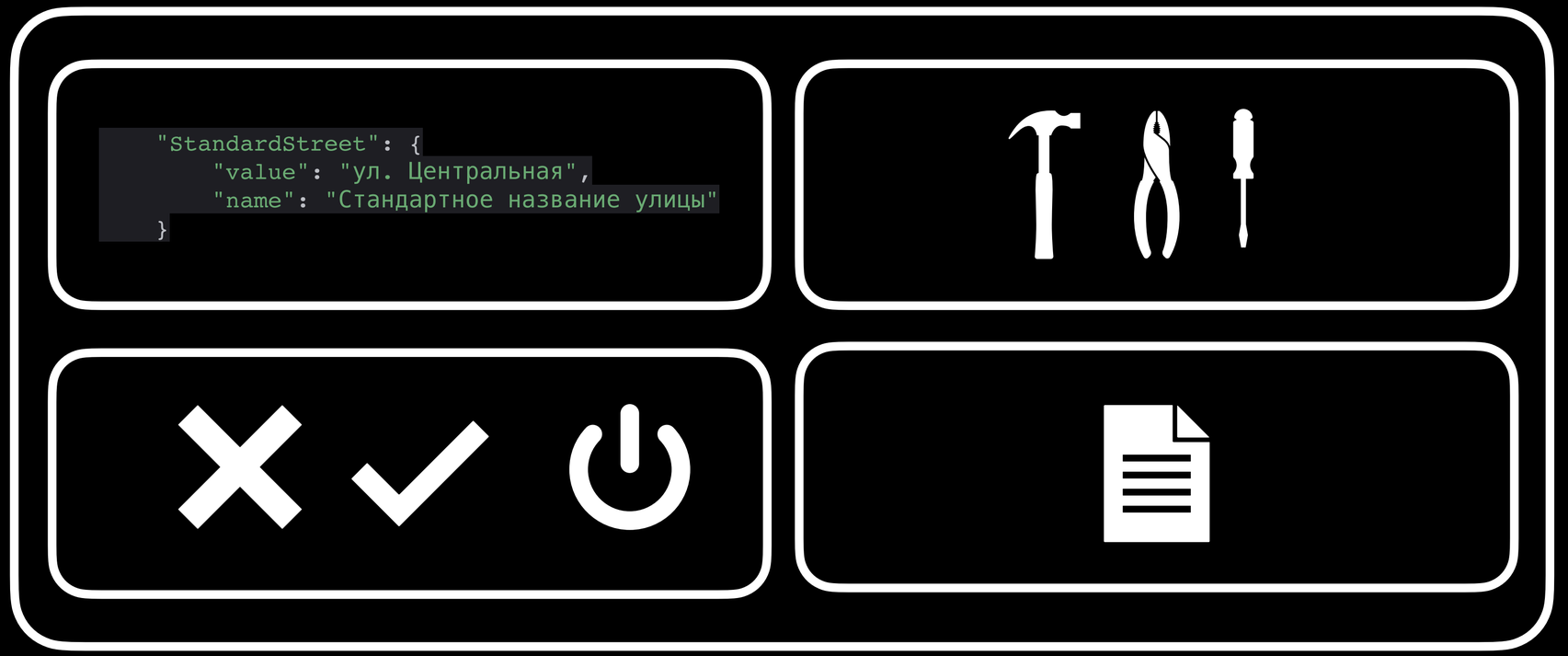
Пример стандартного запуска





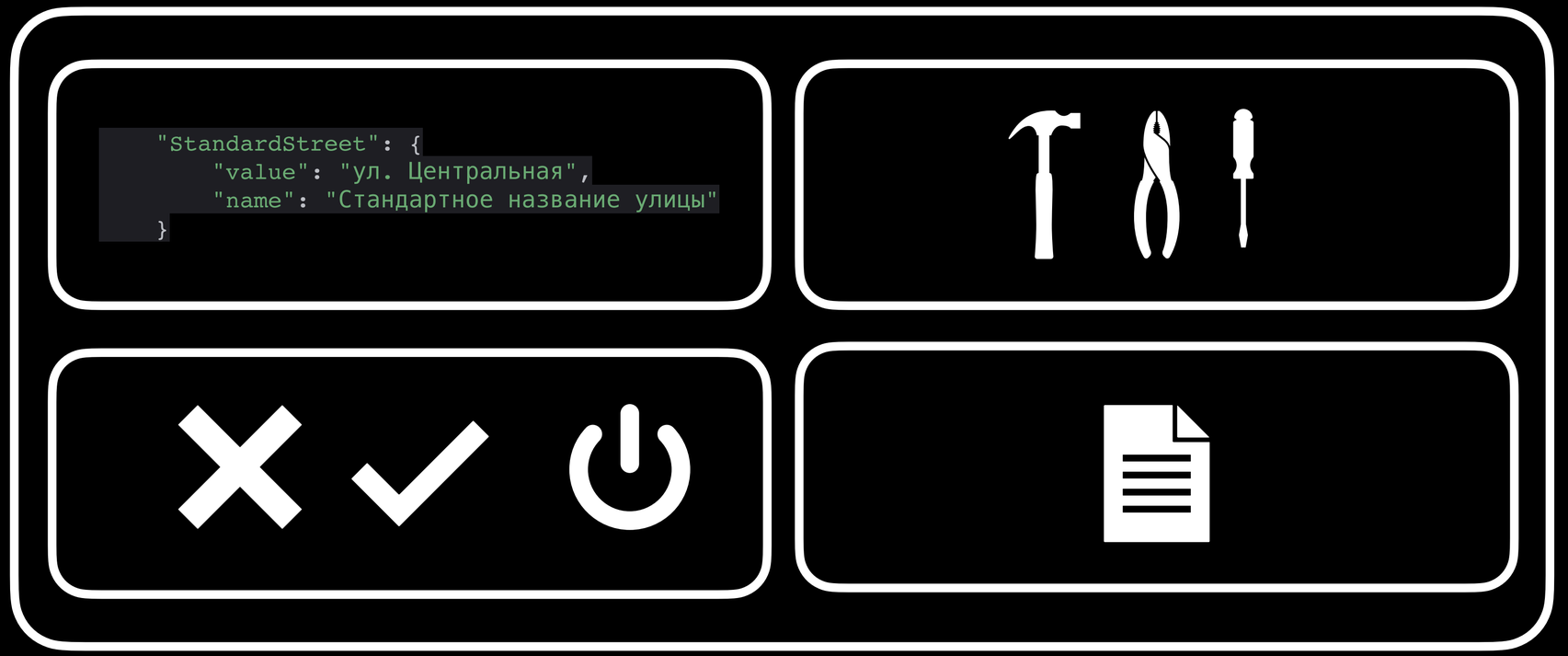
```
"standard": (ST_COMBINE, ST_GENERATE, ST_FORM, SEND, CHECK_WRI)
```

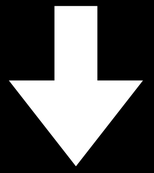
Пример стандартного запуска




```
"standard": (ST_COMBINE, ST_GENERATE, ST_FORM, SEND, CHECK_WRI)
```

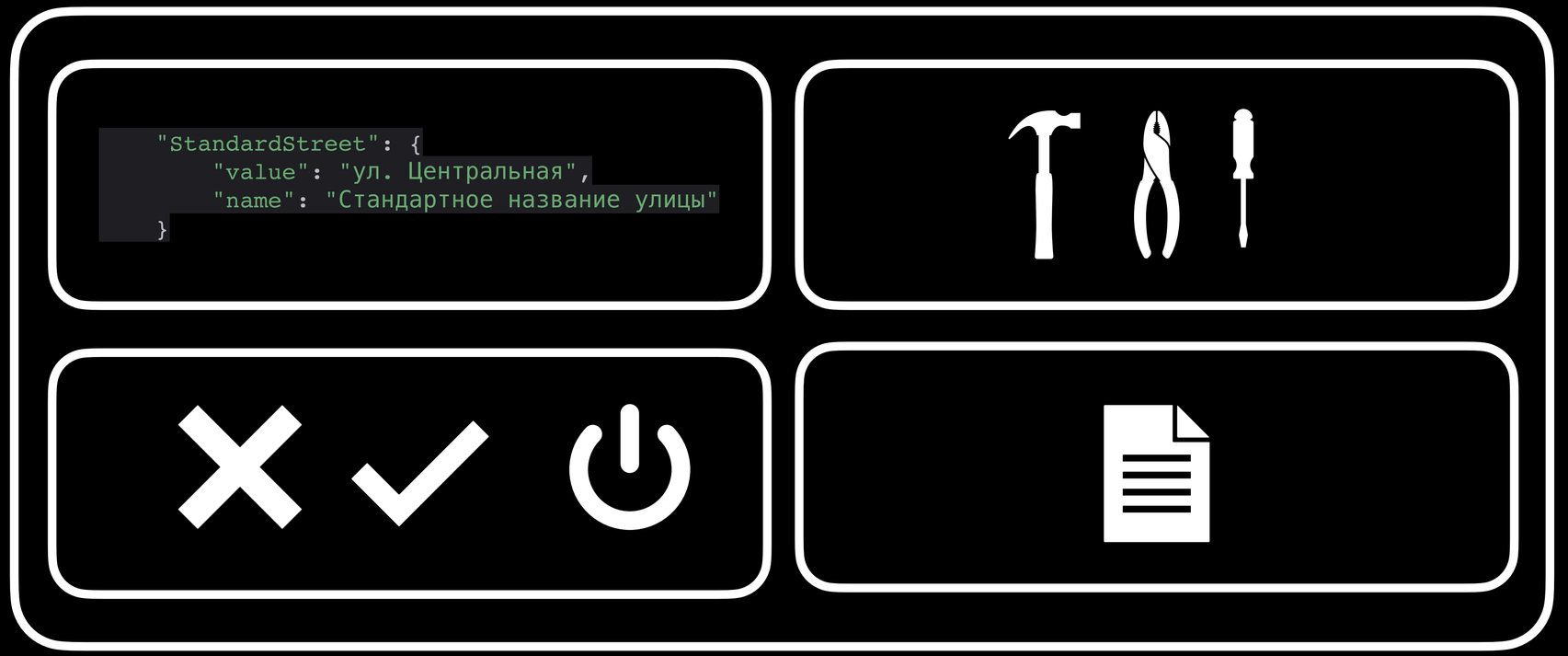
Пример стандартного запуска





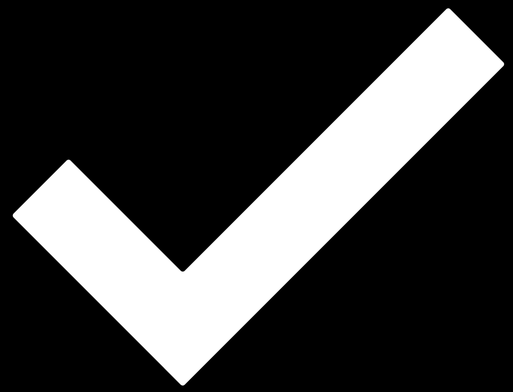
```
"standard": (ST_COMBINE, ST_GENERATE, ST_FORM, SEND, CHECK_WRI)
```

Пример стандартного запуска

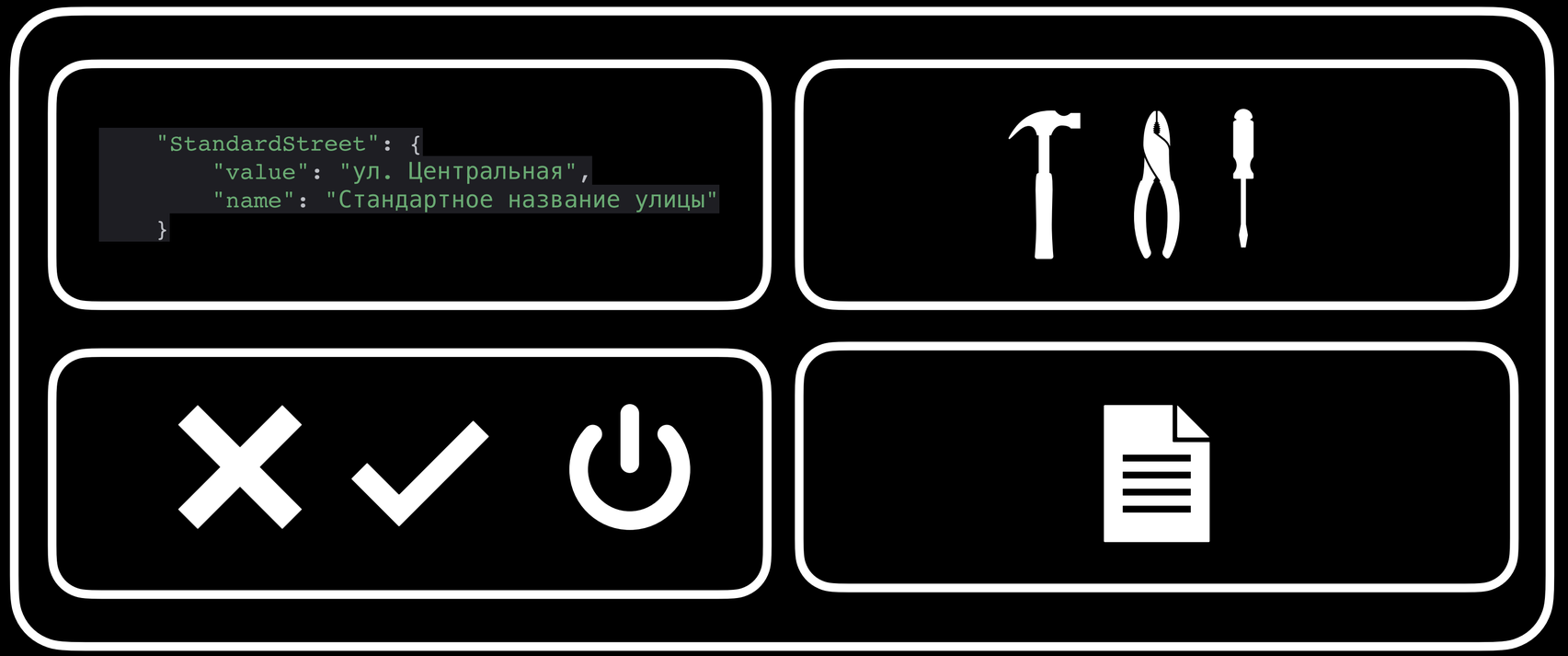


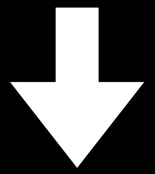

```
"standard": (ST_COMBINE, ST_GENERATE, ST_FORM, SEND, CHECK_WRI)
```

Результат



Пример стандартного запуска





```
"standard": (ST_COMBINE, ST_GENERATE, ST_FORM, SEND, CHECK_WRI)
```

`ST_COMBINE`

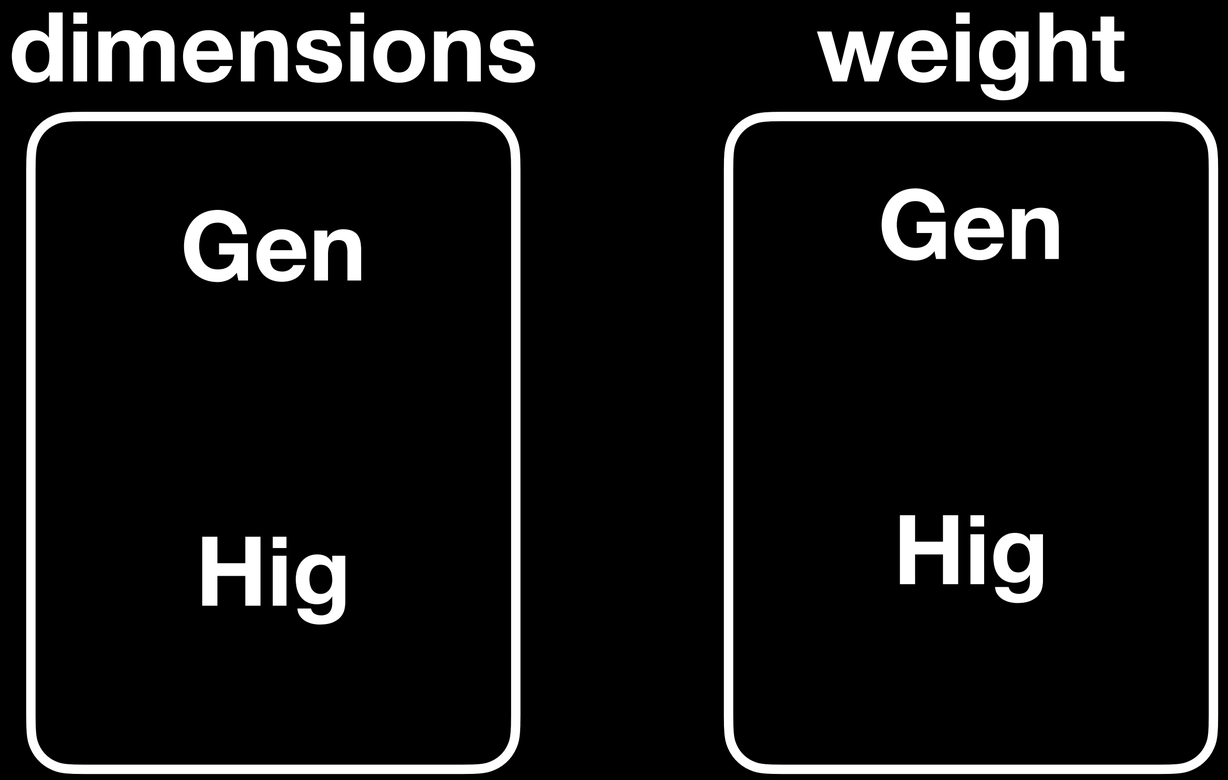
Или как комбинировать кейсы многомерными графами

ST_COMBINE

```
unit_dict["cases"]["dimentions"] = {
    "Gen": {
        "gen_func": re_generate,
        "value": r"[1-9]{1,3}x[1-9]{1,3}x[1-9]{1,2}",
        "name": "Базовый тариф (размер)"
    },
    "Hig": {
        "gen_func": re_generate,
        "options": r"[1-9]{3,8}x[1-9]{3,8}x[1-9]{3,8}",
        "requirements": {"weight": "Hig"},
        "name": "Грузовой тариф (размер)"
    }
}
```

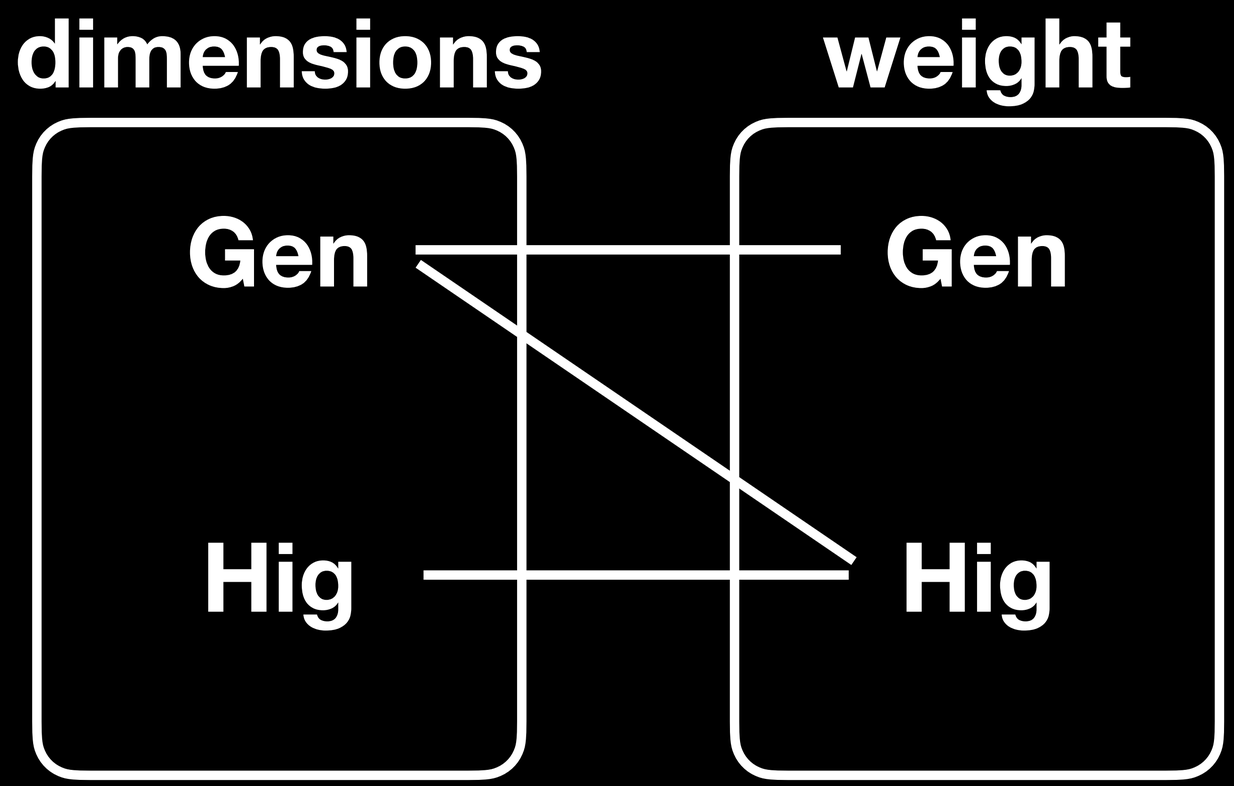
```
unit_dict["cases"]["weight"] = {
    "Gen": {
        "gen_func": random.randint,
        "options": {"a": 100, "b": 10000},
        "name": "Базовый тариф (вес)"
    },
    "Hig": {
        "gen_func": random.randint,
        "options": {"a": 10000, "b": 100000000},
        "name": "Грузовой тариф (вес)"
    }
}
```

ST_COMBINE

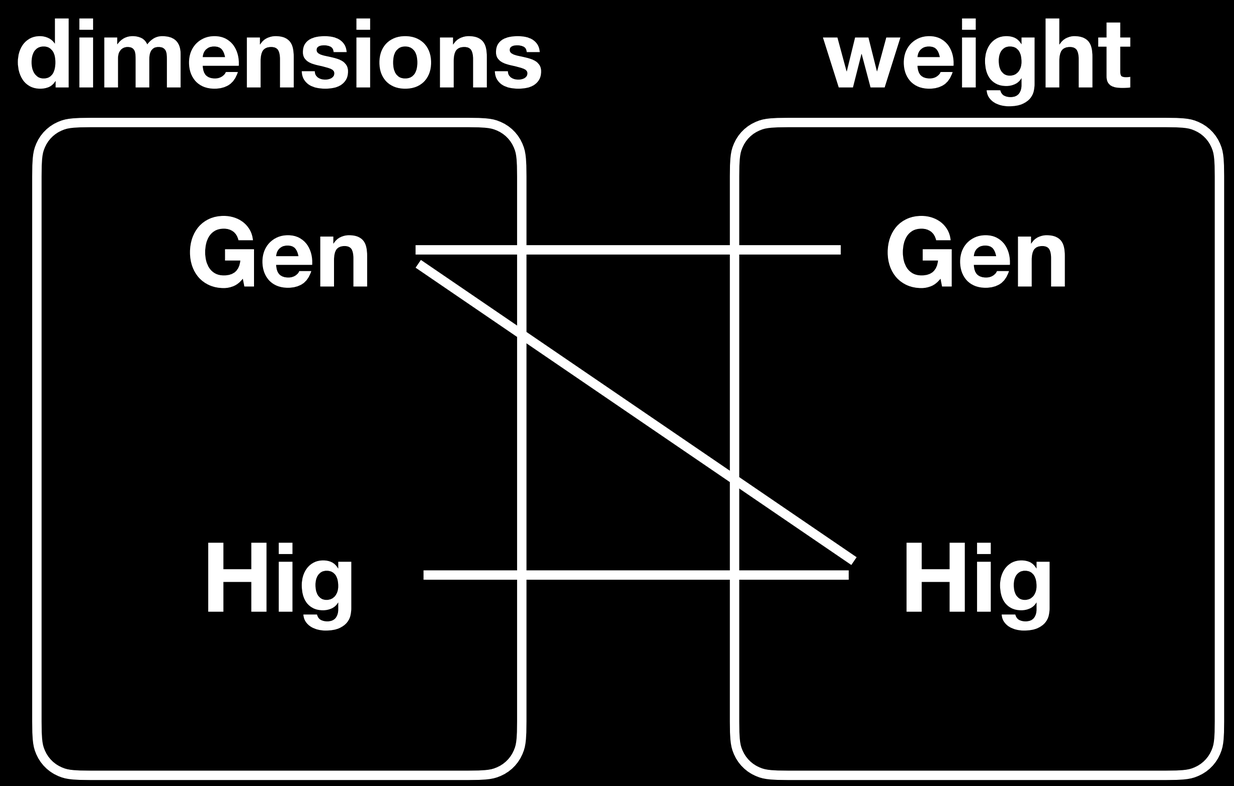


ST_COMBINE

```
"Hig": {
        "gen_func": re_generate,
        "options": r"[1-9]{3,8}x[1-9]{3,8}x[1-9]{3,8}",
        "requirements": {"weight": "Hig"},
        "name": "Грузовой тариф (размер)"
    }
```

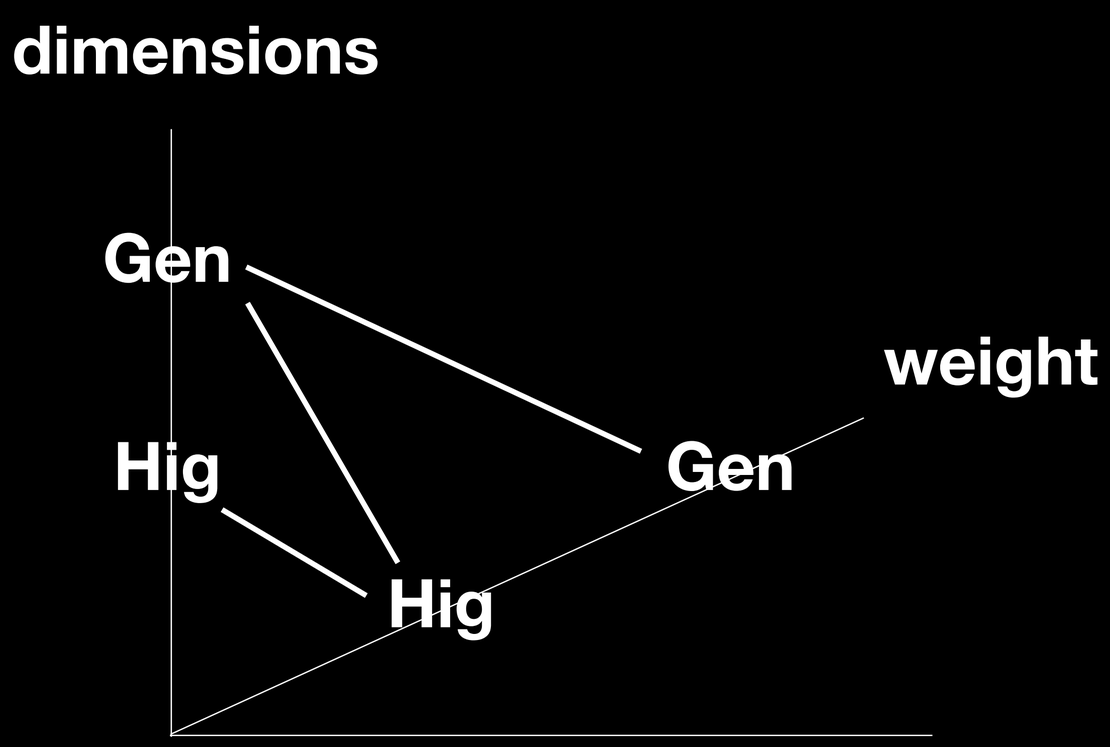


ST_COMBINE



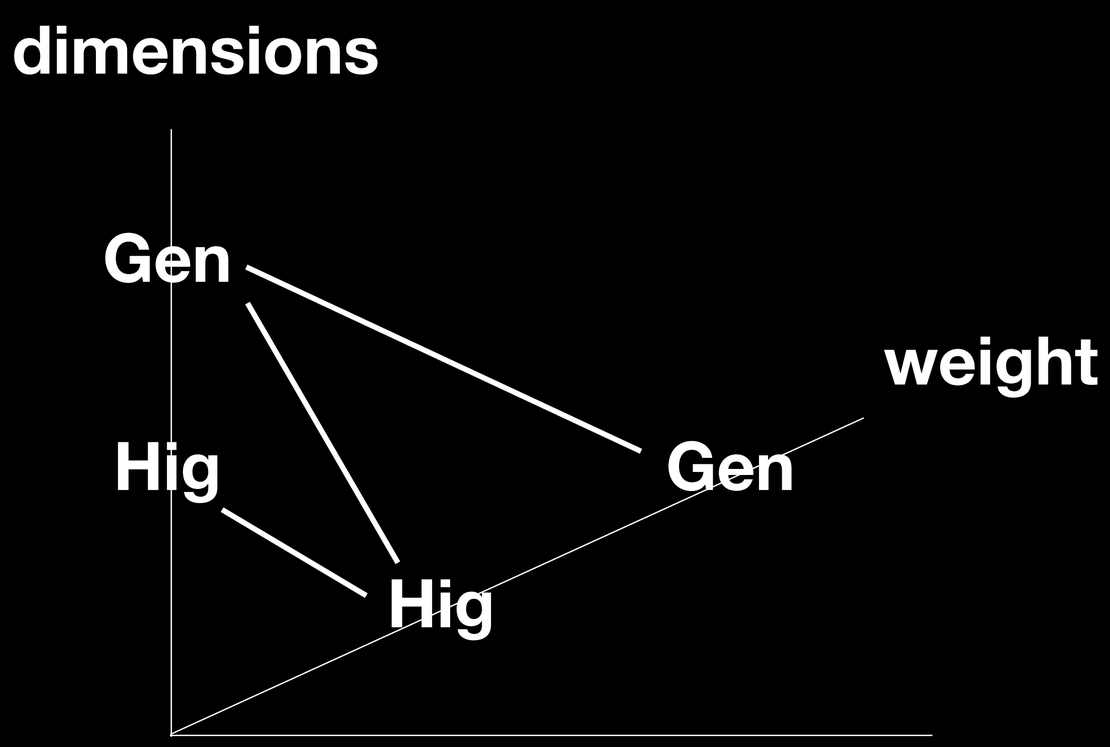

```
{"dimensions": "Gen",
 "weight": "Gen"}

{"dimensions": "Gen",
 "weight": "Hig"}

{"dimensions": "Hig",
 "weight": "Hig"}

{"dim      ons": "Hig",
 "wei      ": "Gen"}
```

ST_COMBINE



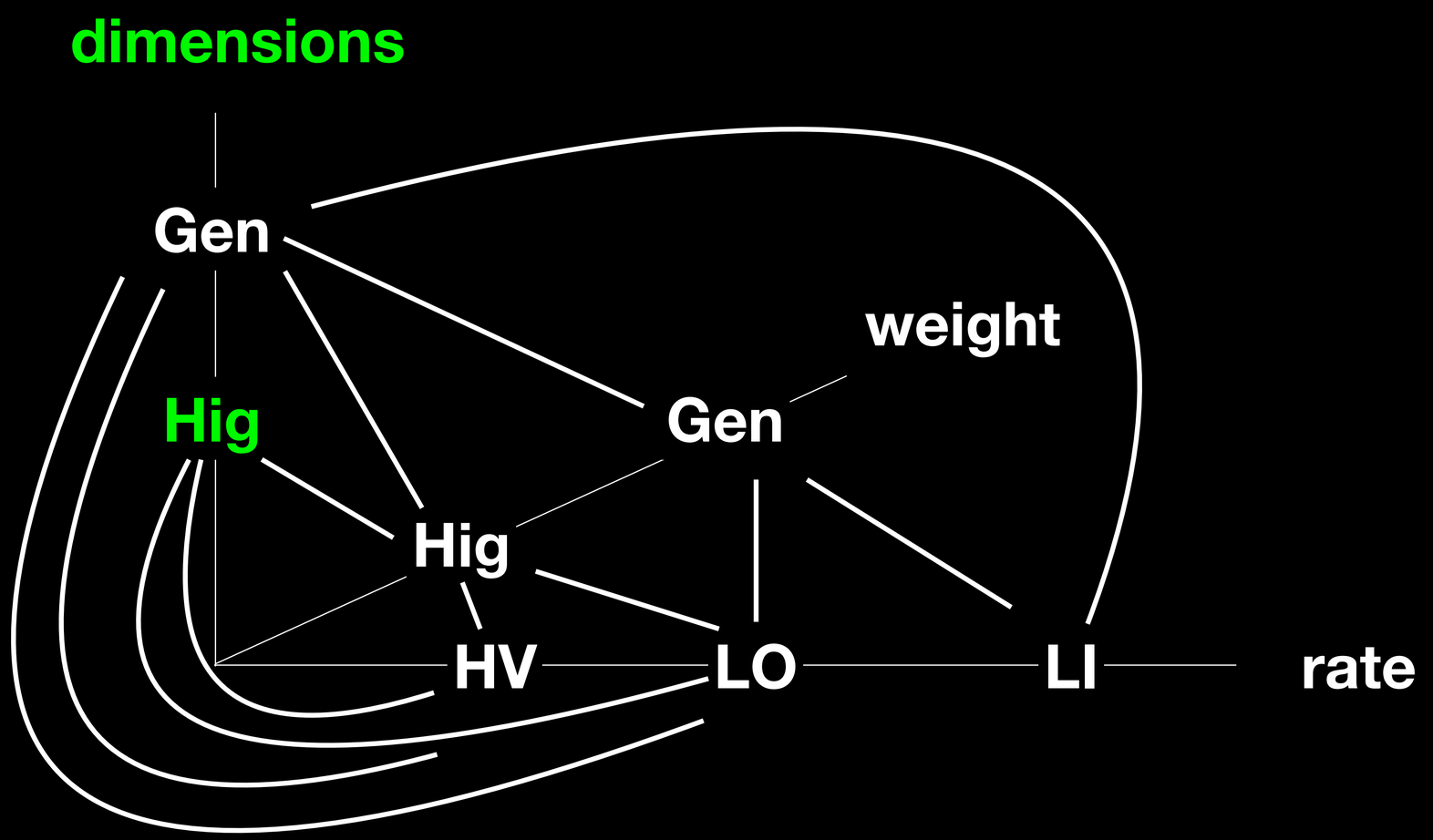

```
unit_dict["cases"]["rate"] = {
    "Heavy": {
        "value": "heavy",
        "requirements": {"weight": "Hig"},
        "name": "Грузовой тариф"
    },
    "Light": {
        "value": "light",
        "requirements": {"weight": "Gen"},
        "name": "Перевозка курьером"
    },
    "Loyal": {
        "value": "loyal",
        "requirements": {"dimentions": "Gen"},
        "name": "Спецпредложение"
    }
}
```

ST_COMBINE

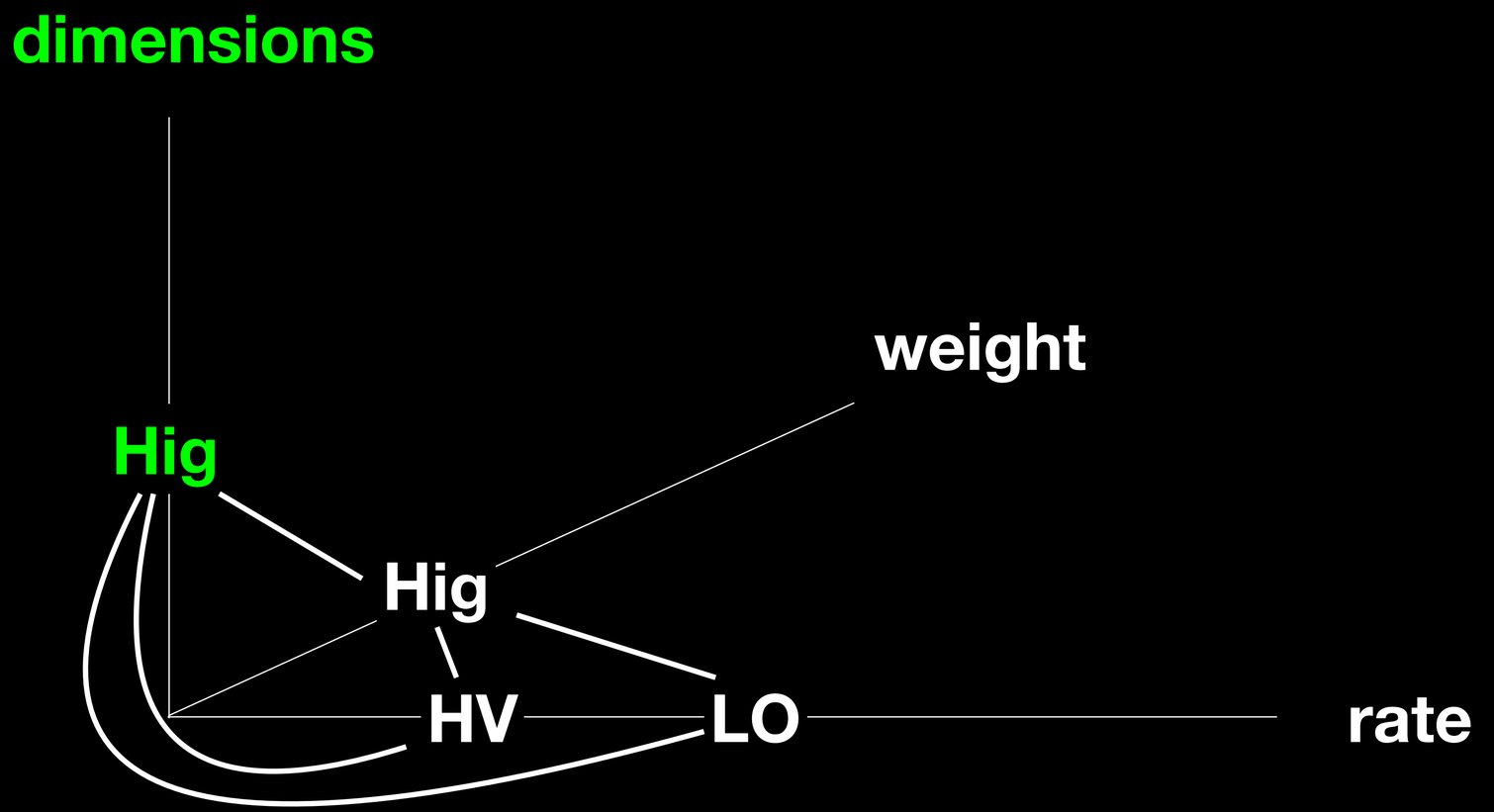



```
unit_dict["cases"]["rate"] = {
    "Heavy": {
        "value": "heavy",
        "requirements": {"weight": "Hig"},
        "name": "Грузовой тариф"
    },
    "Light": {
        "value": "light",
        "requirements": {"weight": "Gen"},
        "name": "Перевозка курьером"
    },
    "Loyal": {
        "value": "loyal",
        "requirements": {"dimentions": "Gen"},
        "name": "Спецпредложение"
    }
}
```

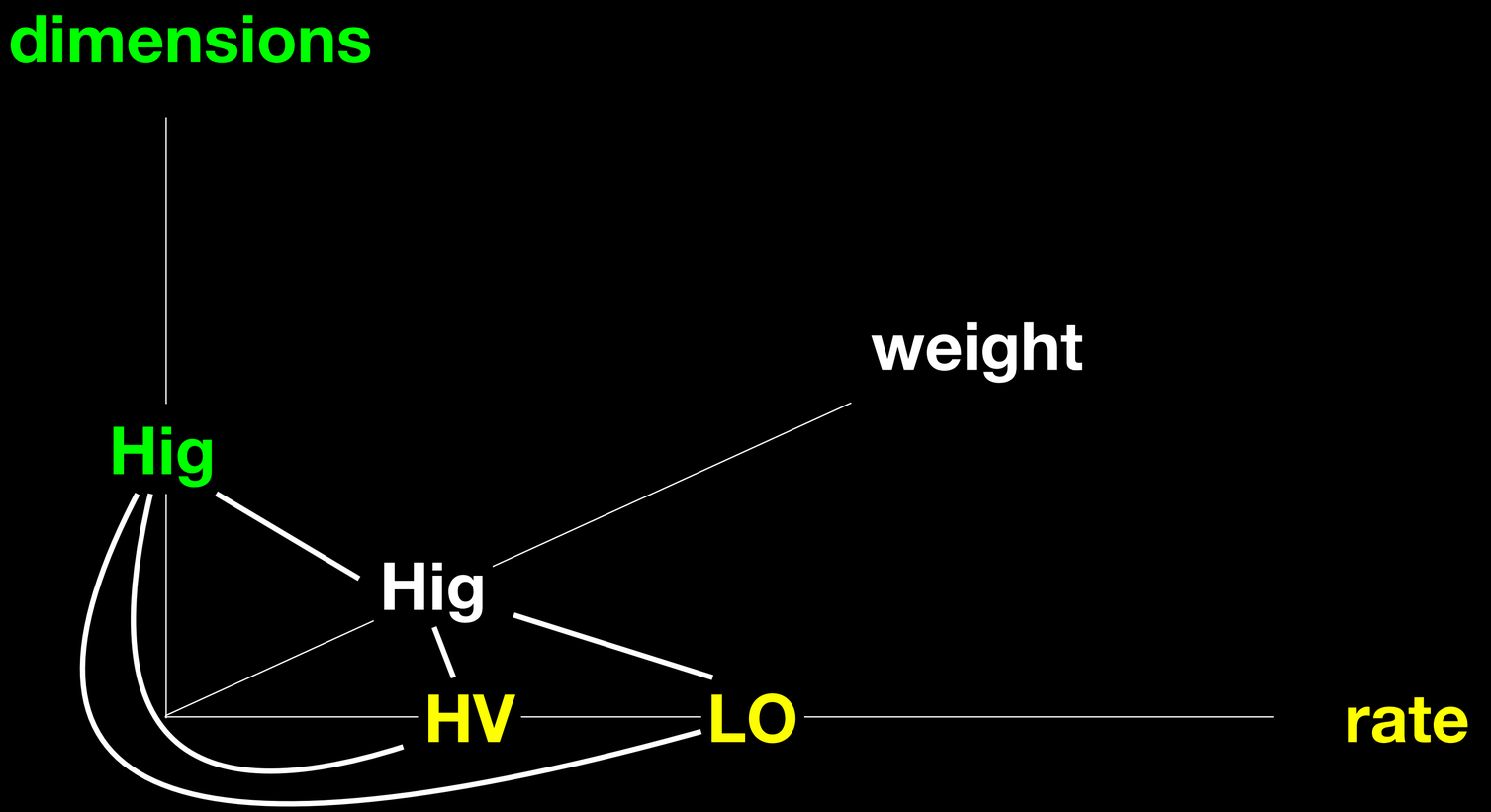
ST_COMBINE



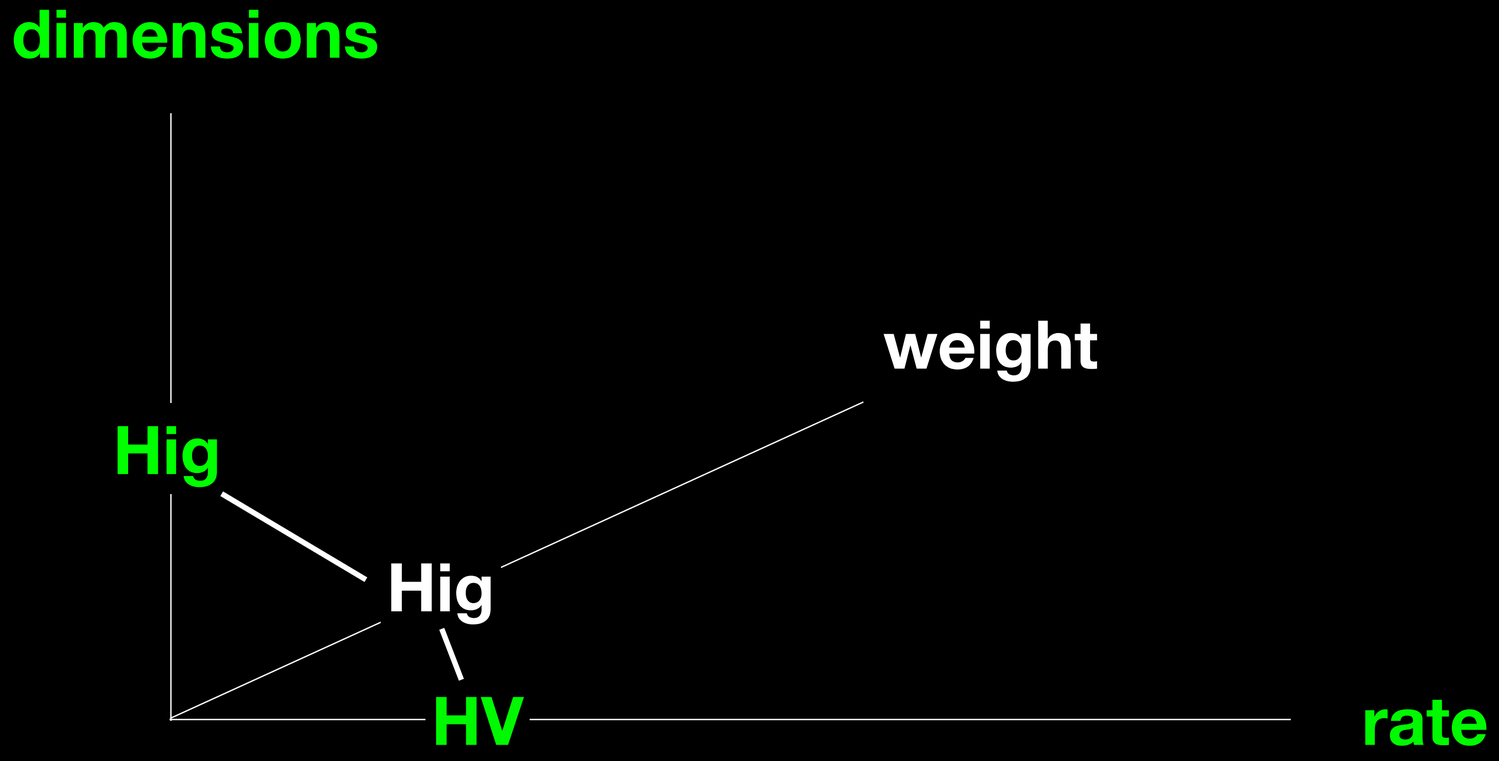
ST_COMBINE



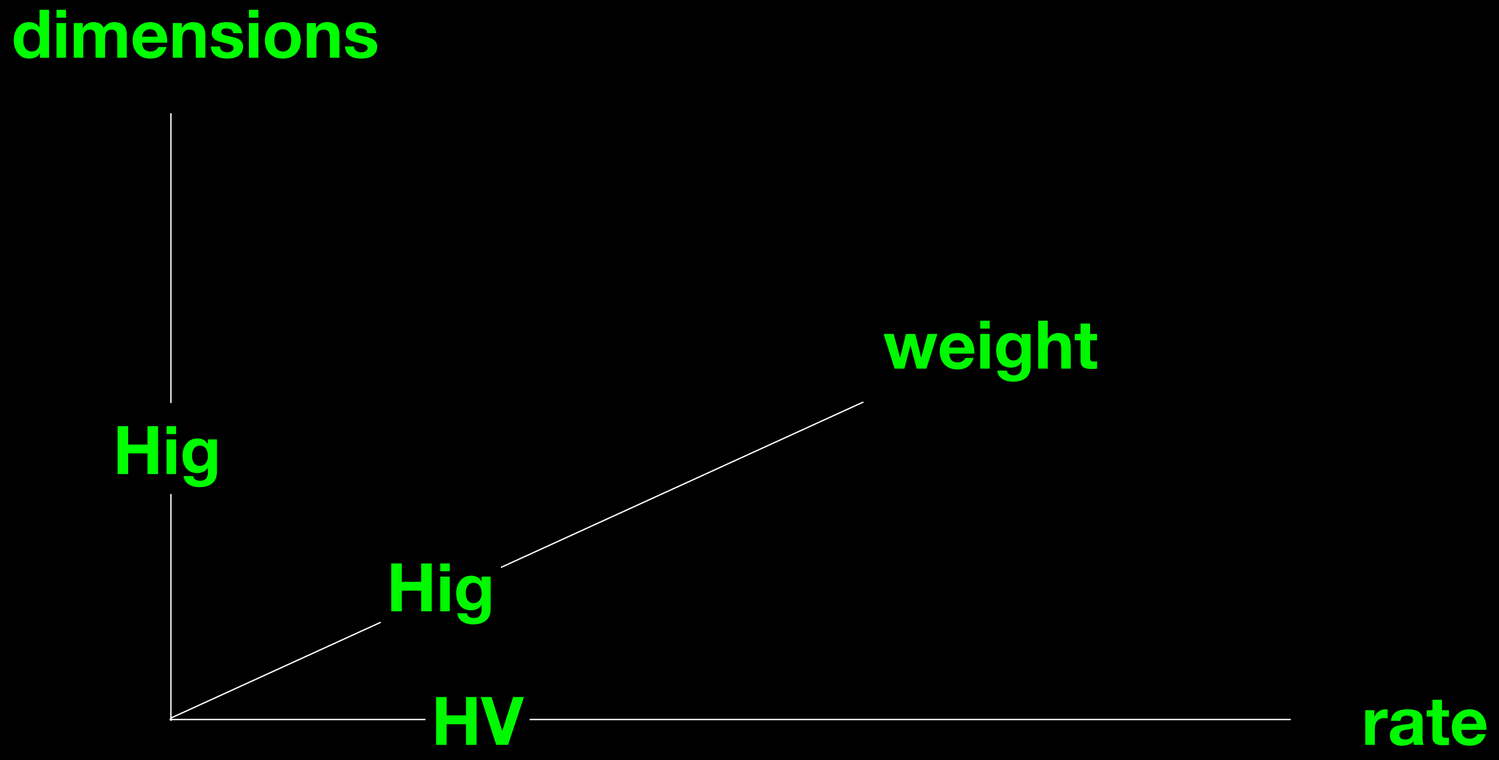
ST_COMBINE



ST_COMBINE



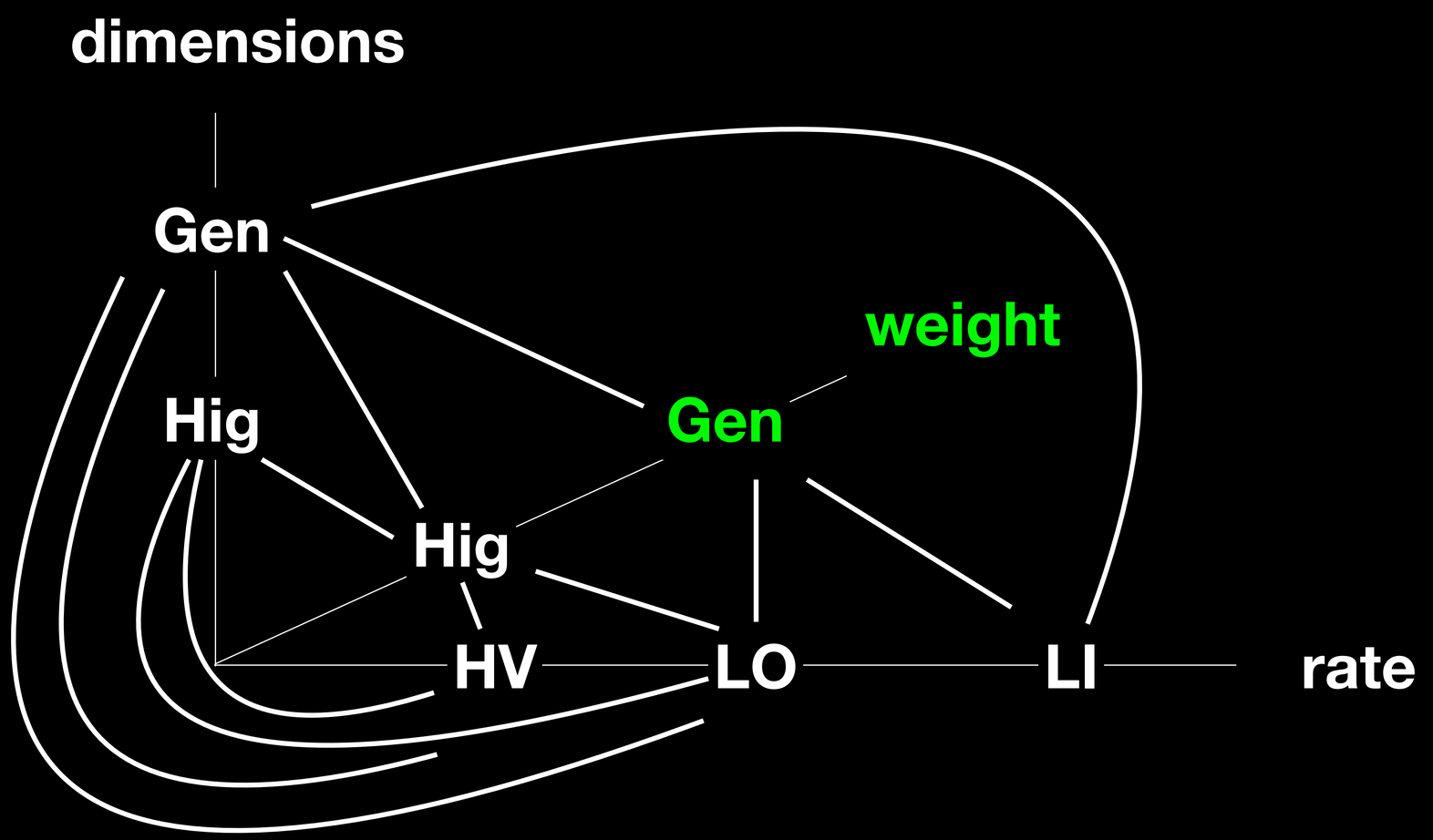
ST_COMBINE



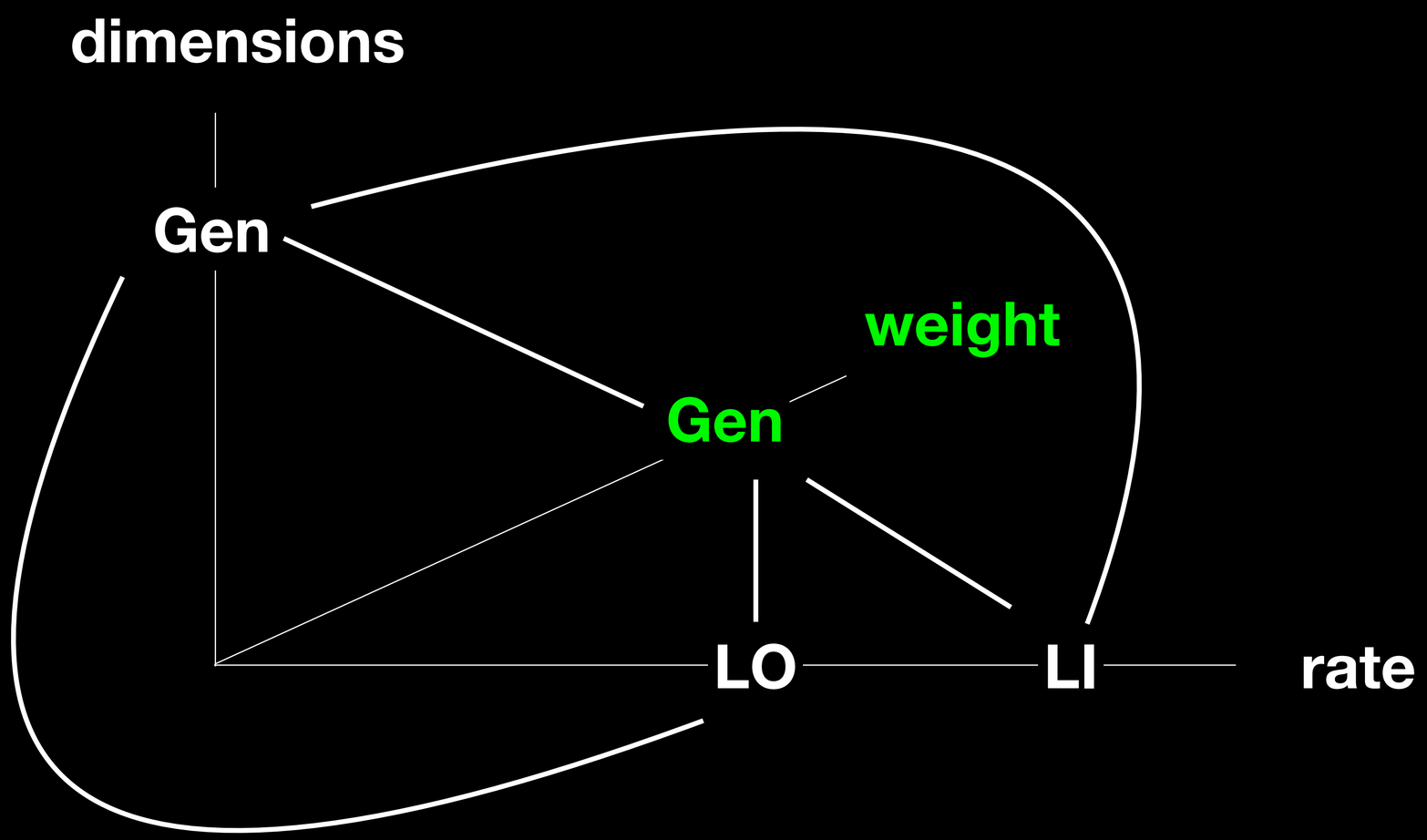
ST_COMBINE

```
{"dimensions": "Hig",
 "weight": "Hig",
 "rate": "HV"}
```

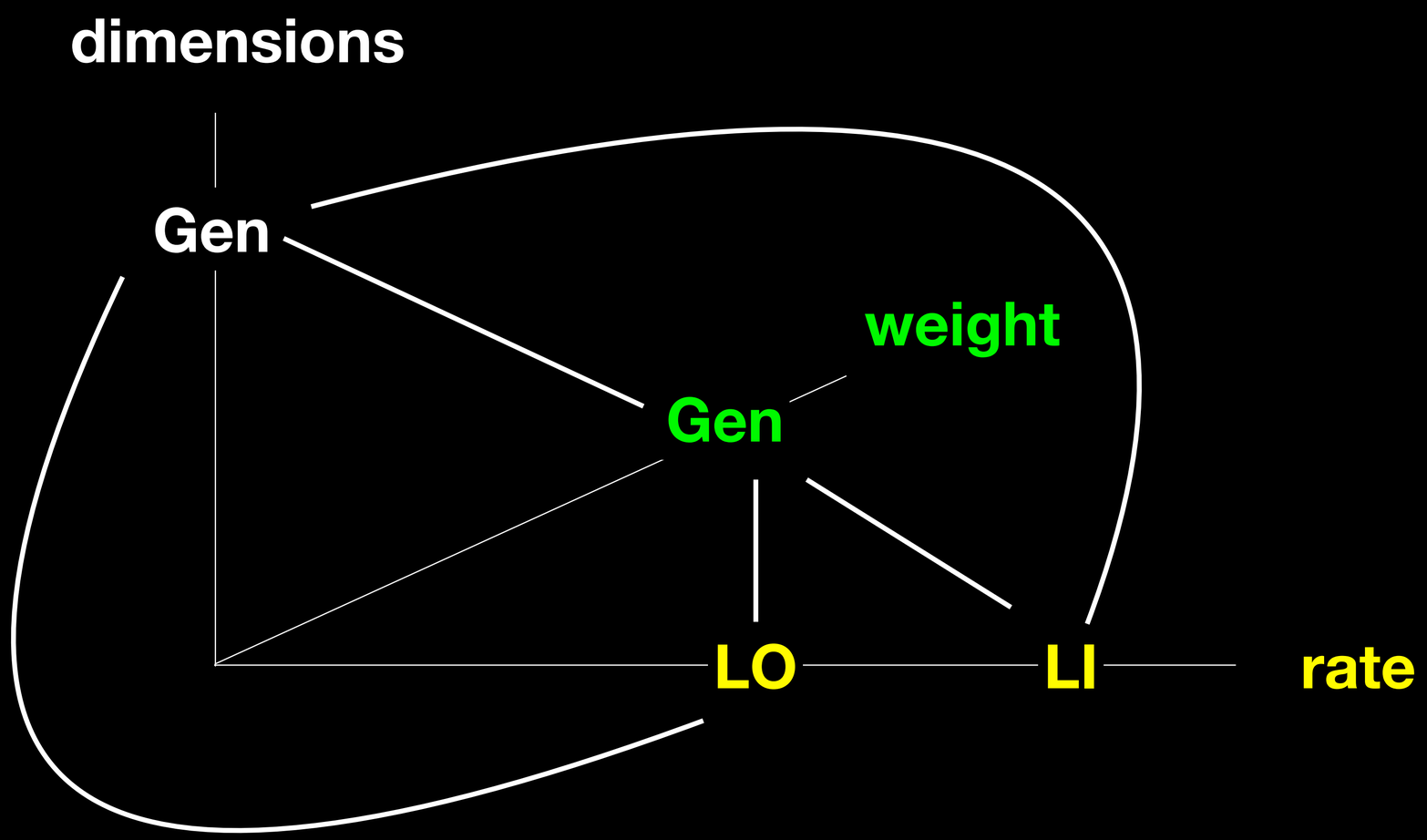
ST_COMBINE



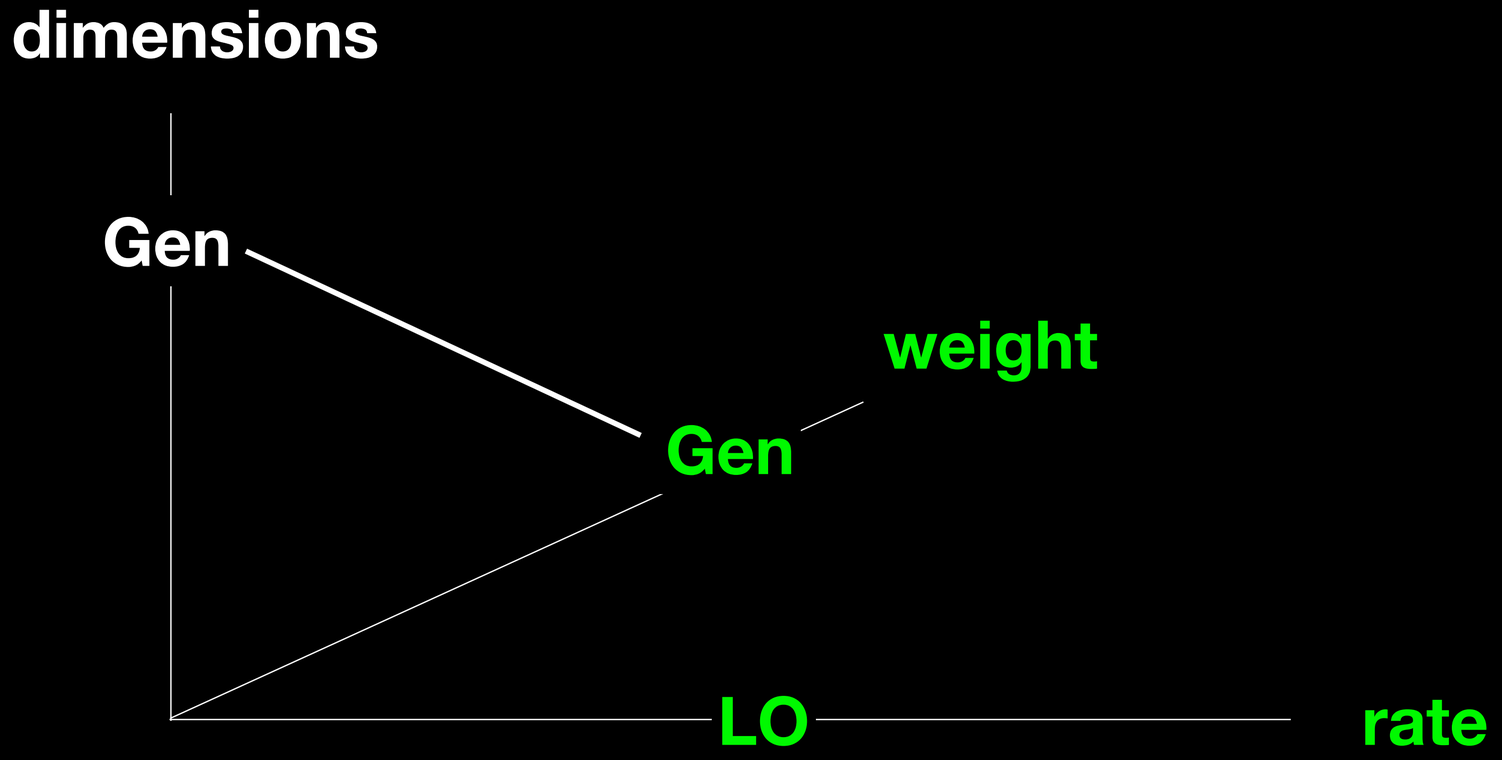
ST_COMBINE



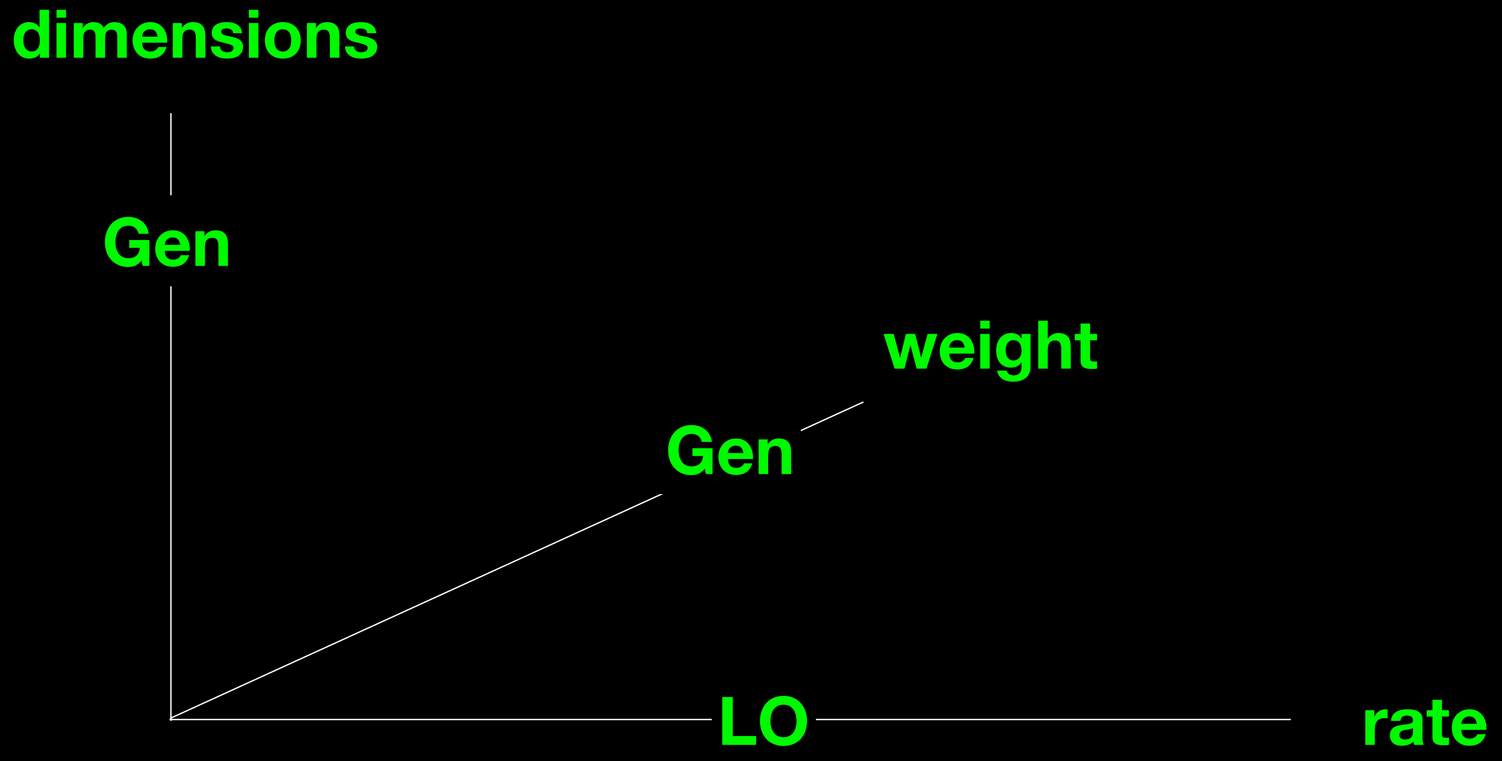
ST_COMBINE



ST_COMBINE



ST_COMBINE



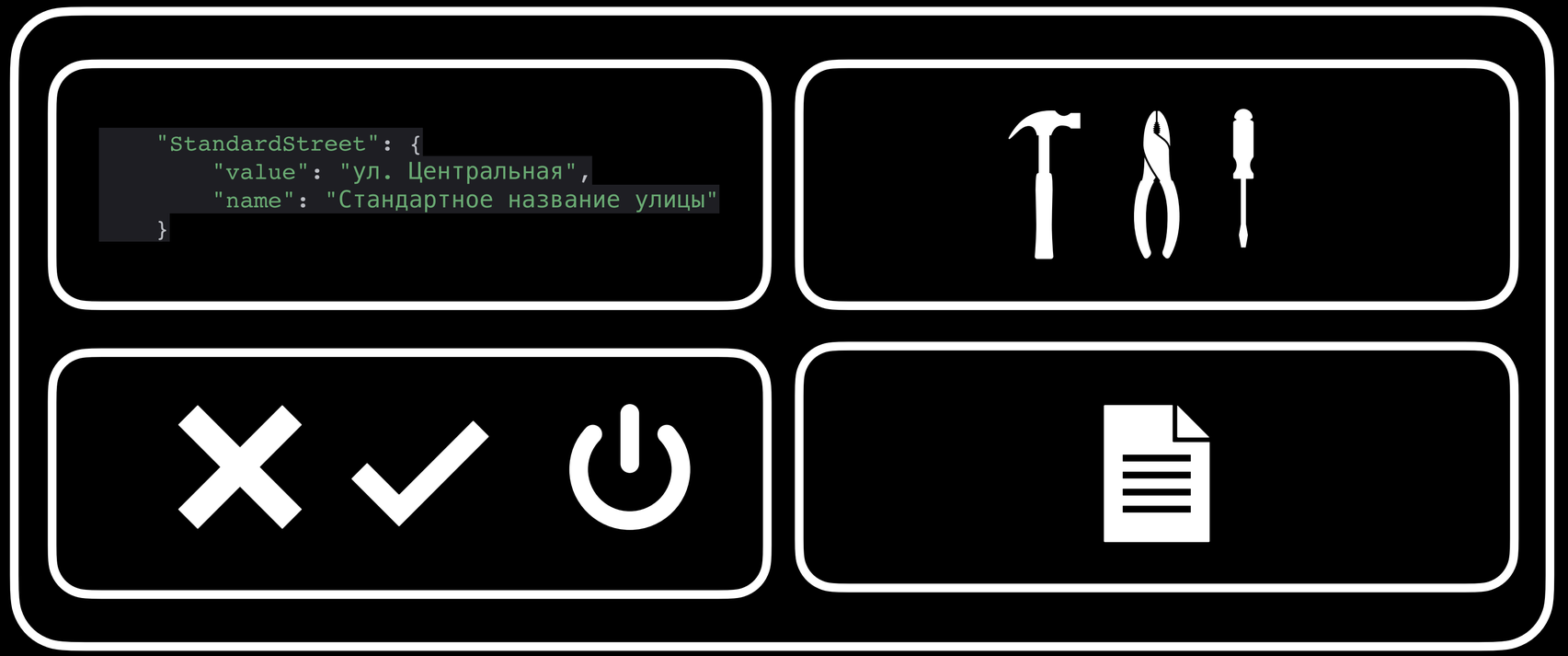
ST_COMBINE

```
{"dimensions": "Gen",
 "weight": "Gen",
 "rate": "LO"}
```

ST_COMBINE

N мерные графы

Пример стандартного запуска




```
"standard": (ST_COMBINE, ST_GENERATE, ST_FORM, SEND, CHECK_WRI)
```

`ST_GENERATE`

Или как отказаться от статических данных

ST_GENERATE

```
{"dimensions": "Gen",
 "weight": "Gen",
 "rate": "LO"}
```

ST_GENERATE

```
{"dimensions": {
        "gen_func": re_generate,
        "options": r"[1-9]{3,8}x[1-9]{3,8}x[1-9]{3,8}",
        "requirements": {"weight": "Hig"},
        "name": "Грузовой тариф (размер)"
    },
 "weight": {
        "gen_func": random.randint,
        "options": {"a": 10000, "b": 100000000},
        "name": "Грузовой тариф (вес)"
    },
 "rate": {
    "value": "heavy",
    "requirements": {"weight": "Hig"},
    "name": "Грузовой тариф"
}}
```

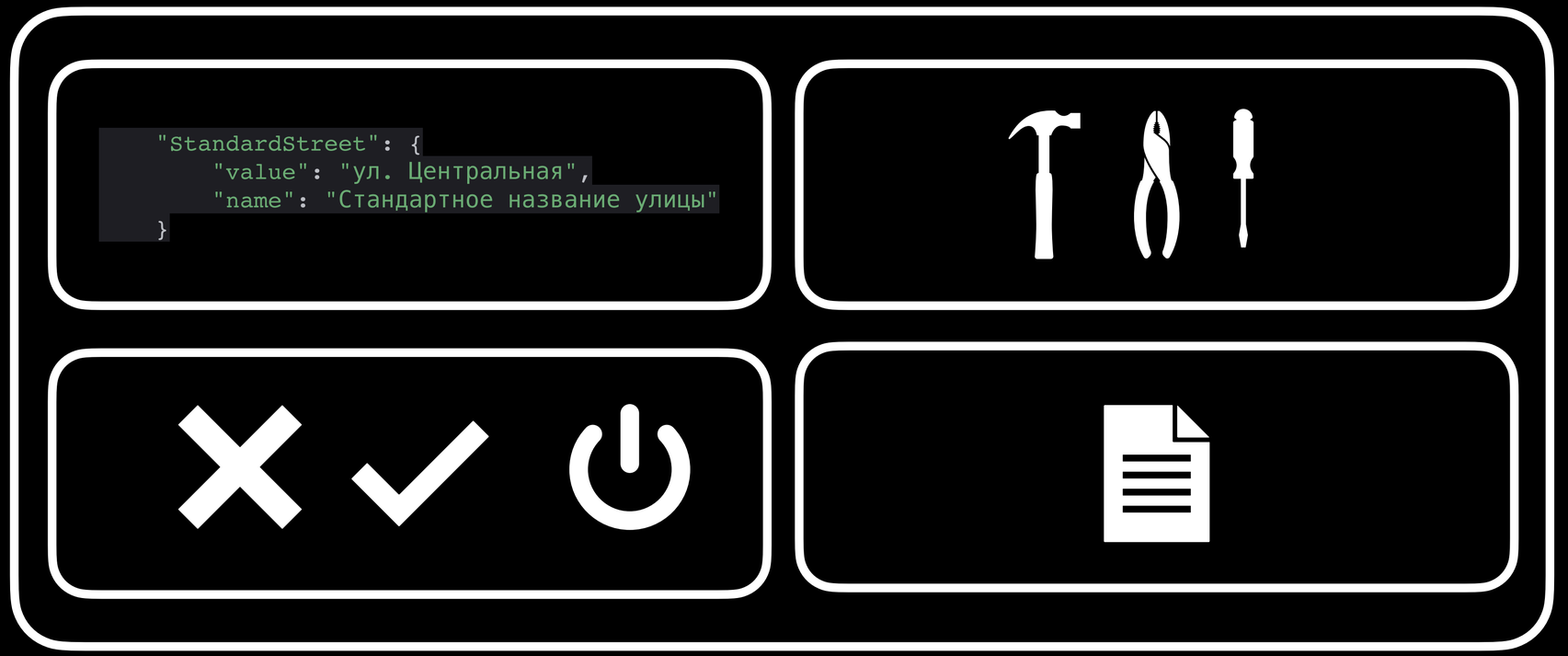
ST_GENERATE

```
{"dimensions":re_generate(r"[1-9]{3,8}x[1-9]{3,8}x[1-9]{3,8}"),
 "weight": random.randint(a=10000, b=100000000),
 "rate":"heavy"}
```

ST_GENERATE

```
{"dimensions":"1879x5674x3954"),
 "weight": 156890,
 "rate":"heavy"}
```

Пример стандартного запуска




```
"standard": (ST_COMBINE, ST_GENERATE, ST_FORM, SEND, CHECK_WRI)
```

`ST_FORM`

Или как удобно форматировать по шаблону

ST_FORM

```
{"dimensions":"1879x5674x3954"),
 "weight": 156890,
 "rate":"heavy"}
```

ST_FORM

```
{"dimensions":"1879x5674x3954"),
 "weight": 156890,
 "rate":"heavy"}
```

ST_FORM

Сгенерированные данные

```
{
    "name": "Анна-Мария",
    "zipcode": "101101",
    "weight": 156890,
    "rate": "heavy",
    "city": "Железногорск",
    "dimentions": "1879x5674x3954",
    "phoneNumbers": [
        "+71234567890",
        "+70987654321"
    ],
    "street": "ул. Центральная, 10"
}
```

Шаблон

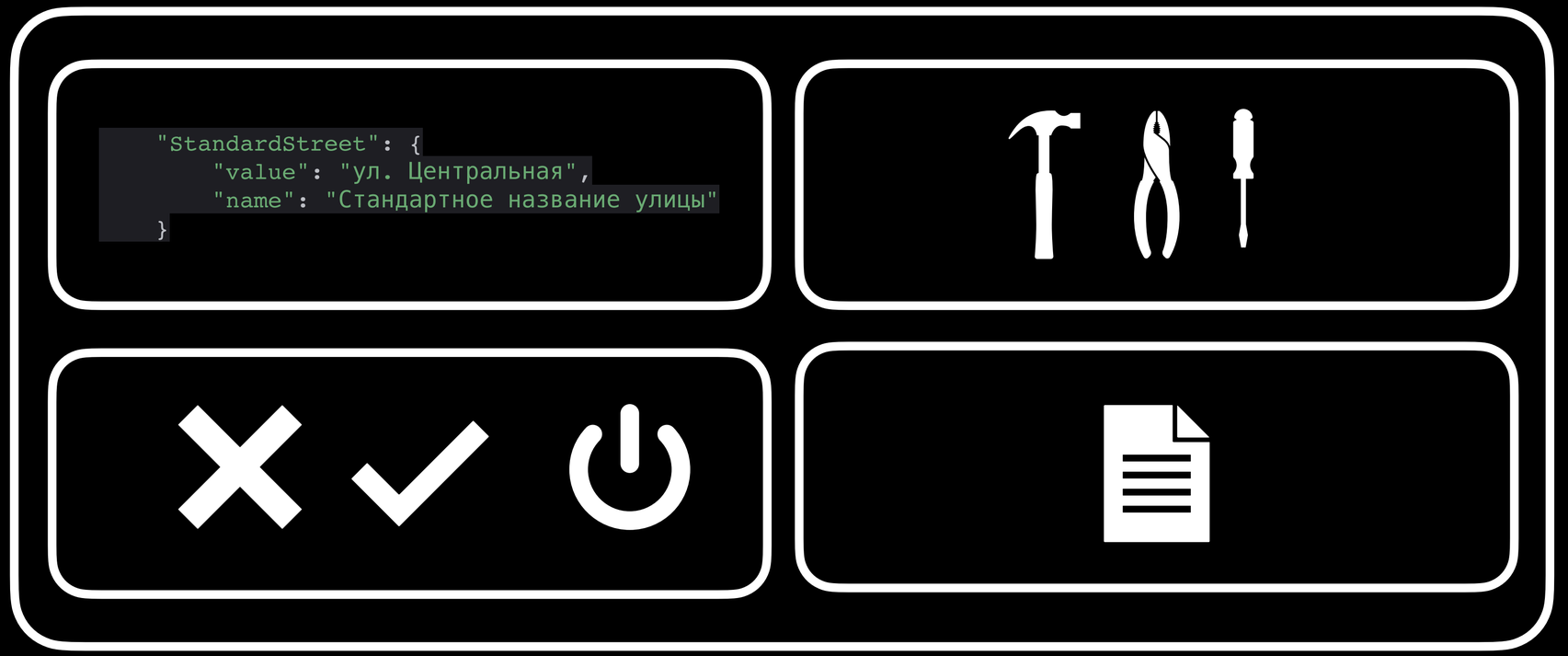
```
{
"name": "name",
"weight": "weight",
"rate": "rate",
"dimentions": "dimentions",
"address": {
   "street": "street",
   "city": "city",
   "zipcode": "zipcode"
                          },
"phoneNumbers": "phoneNumbers"
}
```

ST_FORM

Сформированные данные

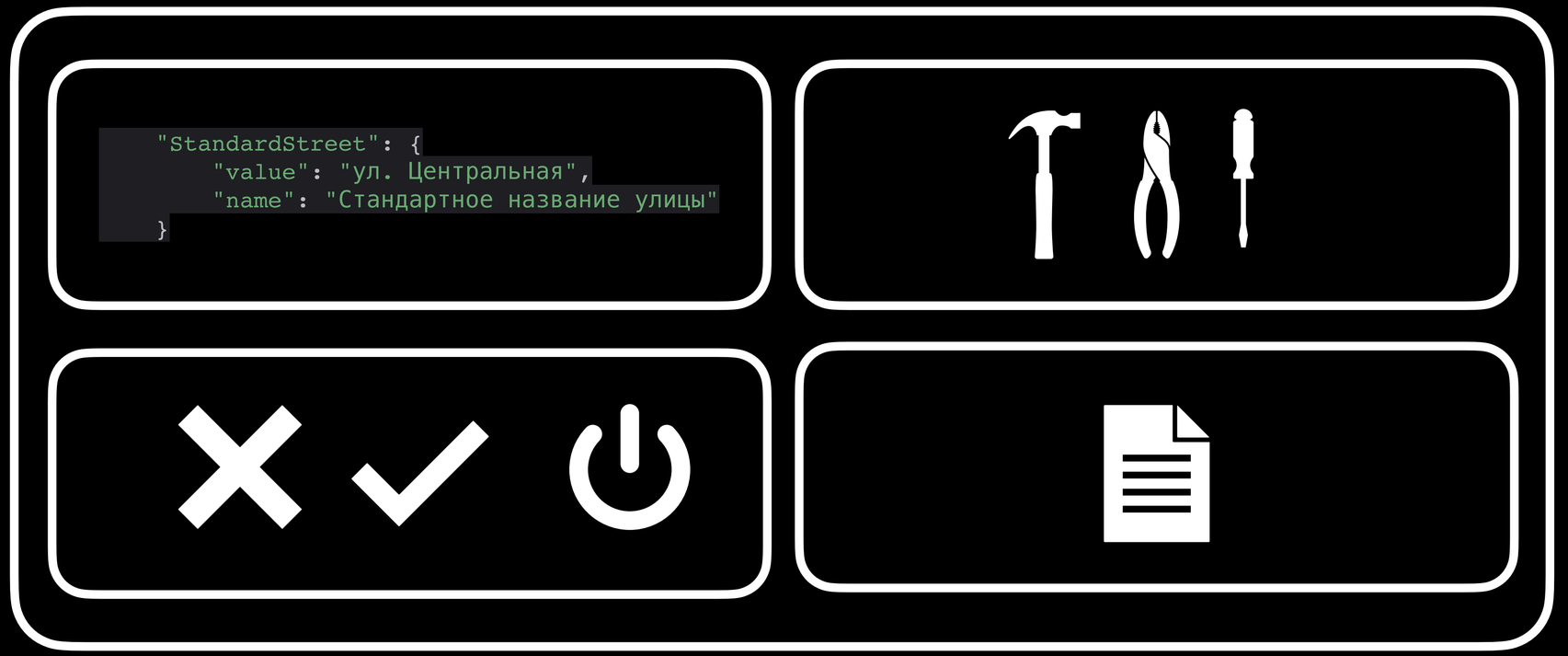

```
{
"name": "Анна-Мария",
"weight": 156890,
"rate": "heavy",
"dimentions": "1879x5674x3954",
"address": {
   "street": "ул. Центральная, 10",
   "city": "Железногорск",
   "zipcode": "101101"
                          },
"phoneNumbers": [
        "+71234567890",
        "+70987654321"
    ]
}
```

Пример стандартного запуска



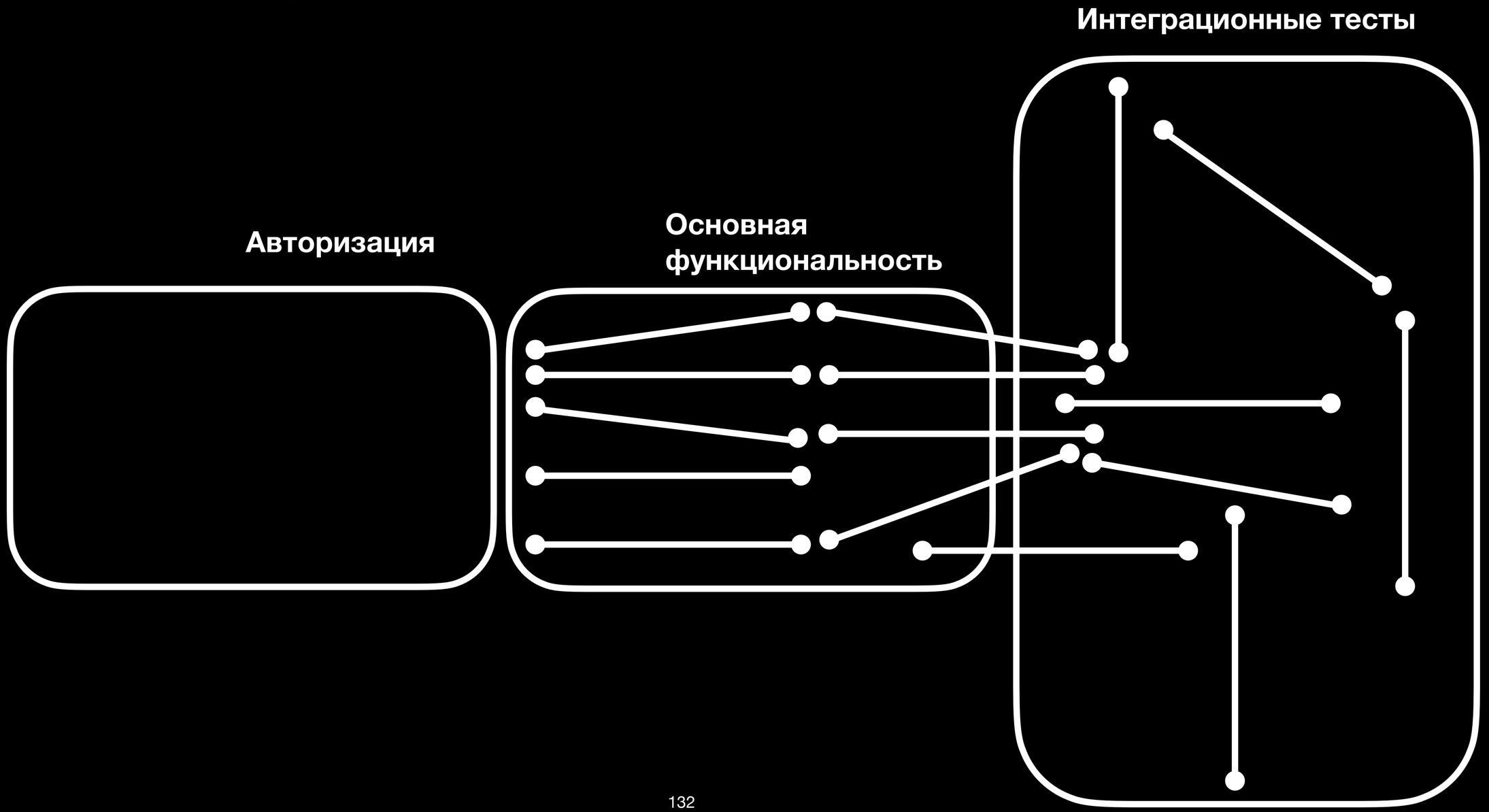

```
"standard": (ST_COMBINE, ST_GENERATE, ST_FORM, SEND, CHECK_WRI)
```

Пример стандартного запуска

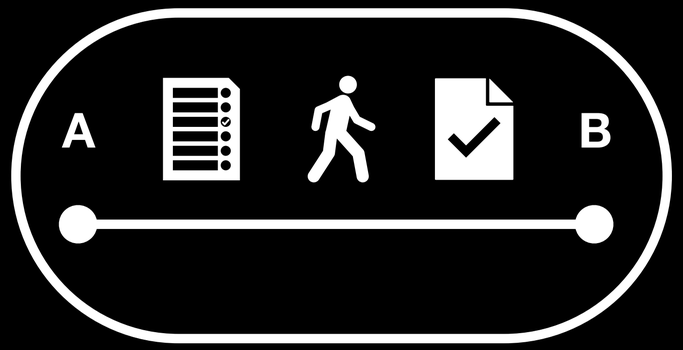



```
"standard": (ST_COMBINE, ST_GENERATE, ST_FORM, SEND, CHECK_WRI)
```

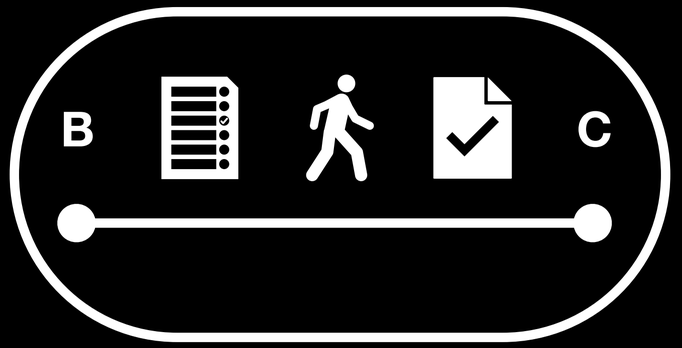
Почтовый сервис



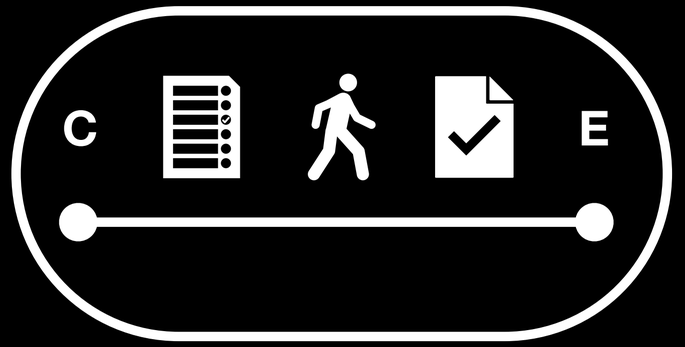
Тест отправки почты



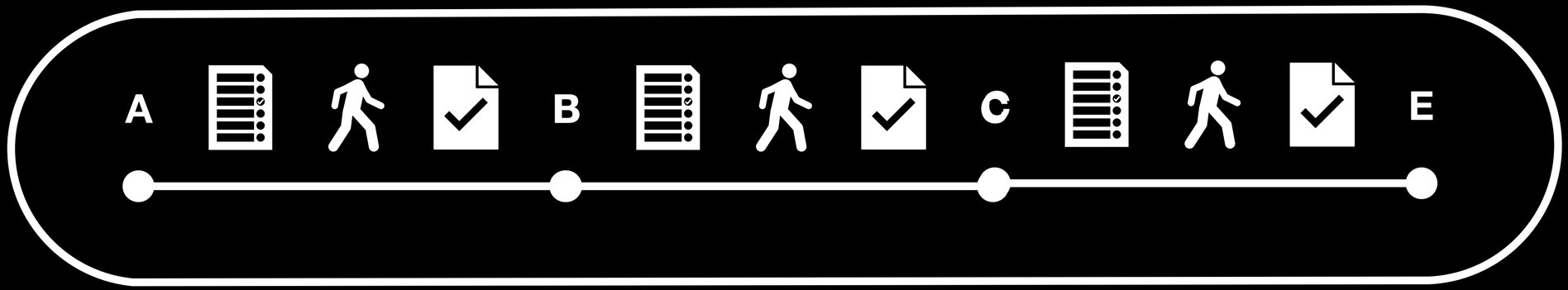
Тест присвоения трек-номера



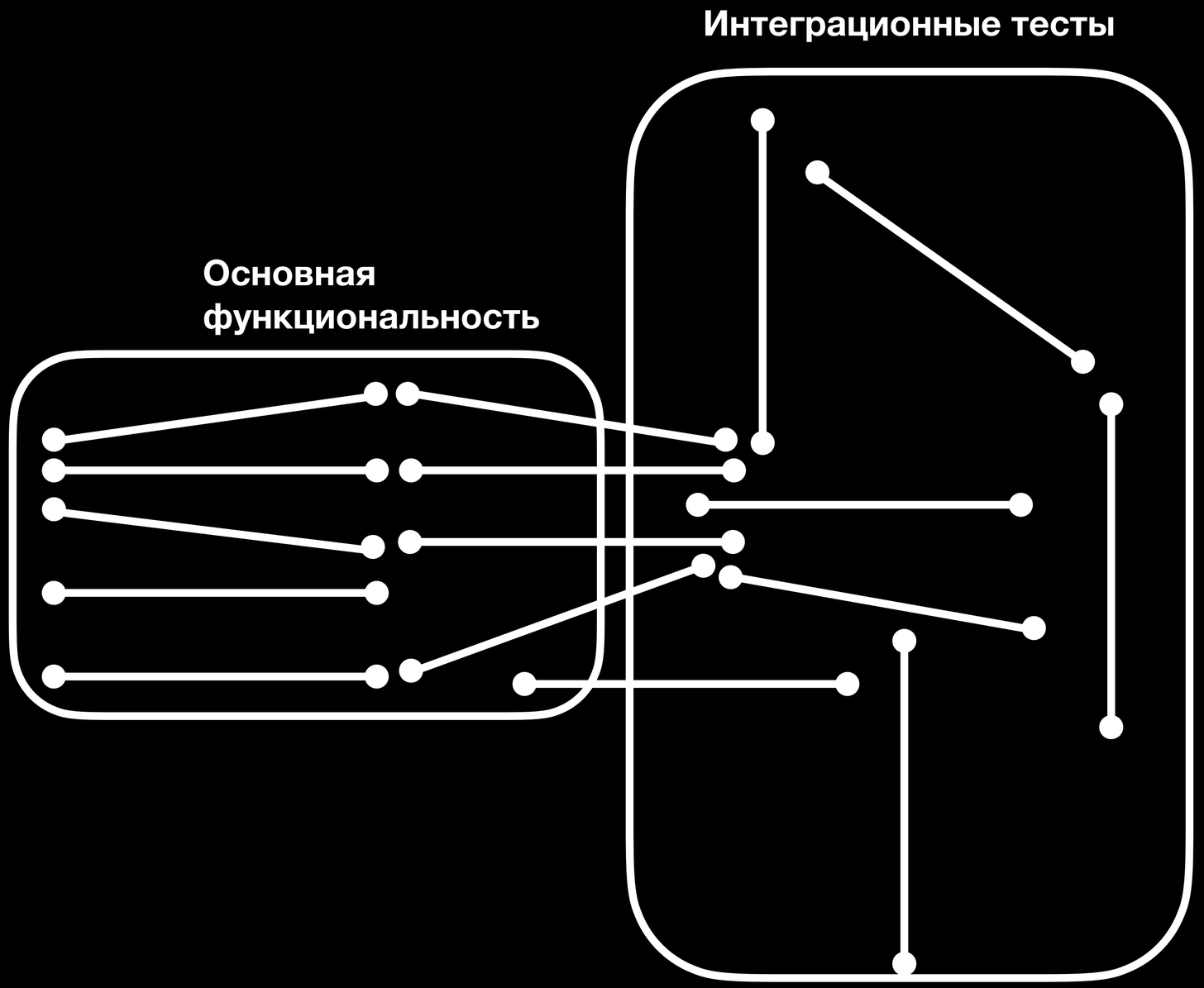
Тест отслеживания по трек-номеру

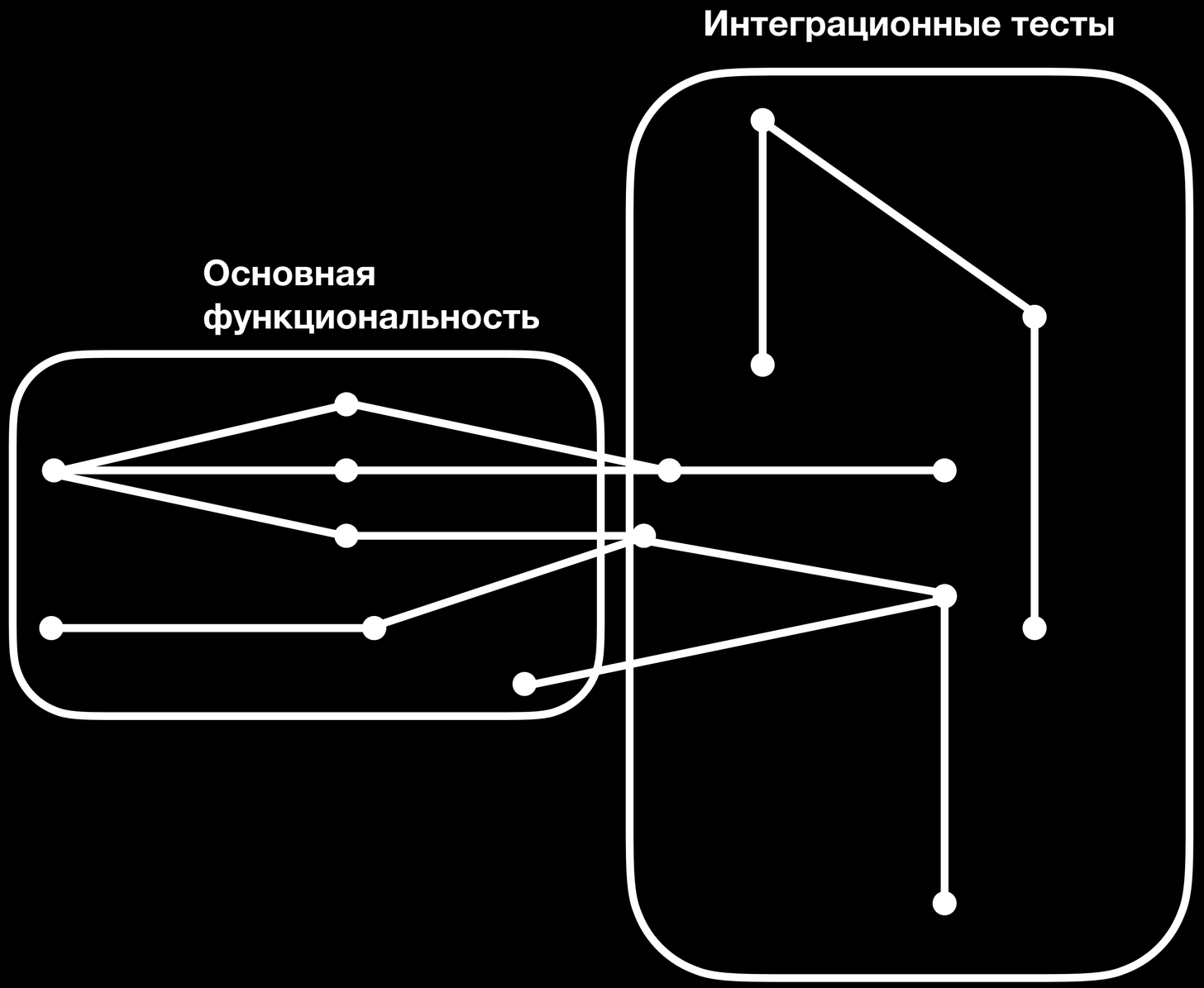


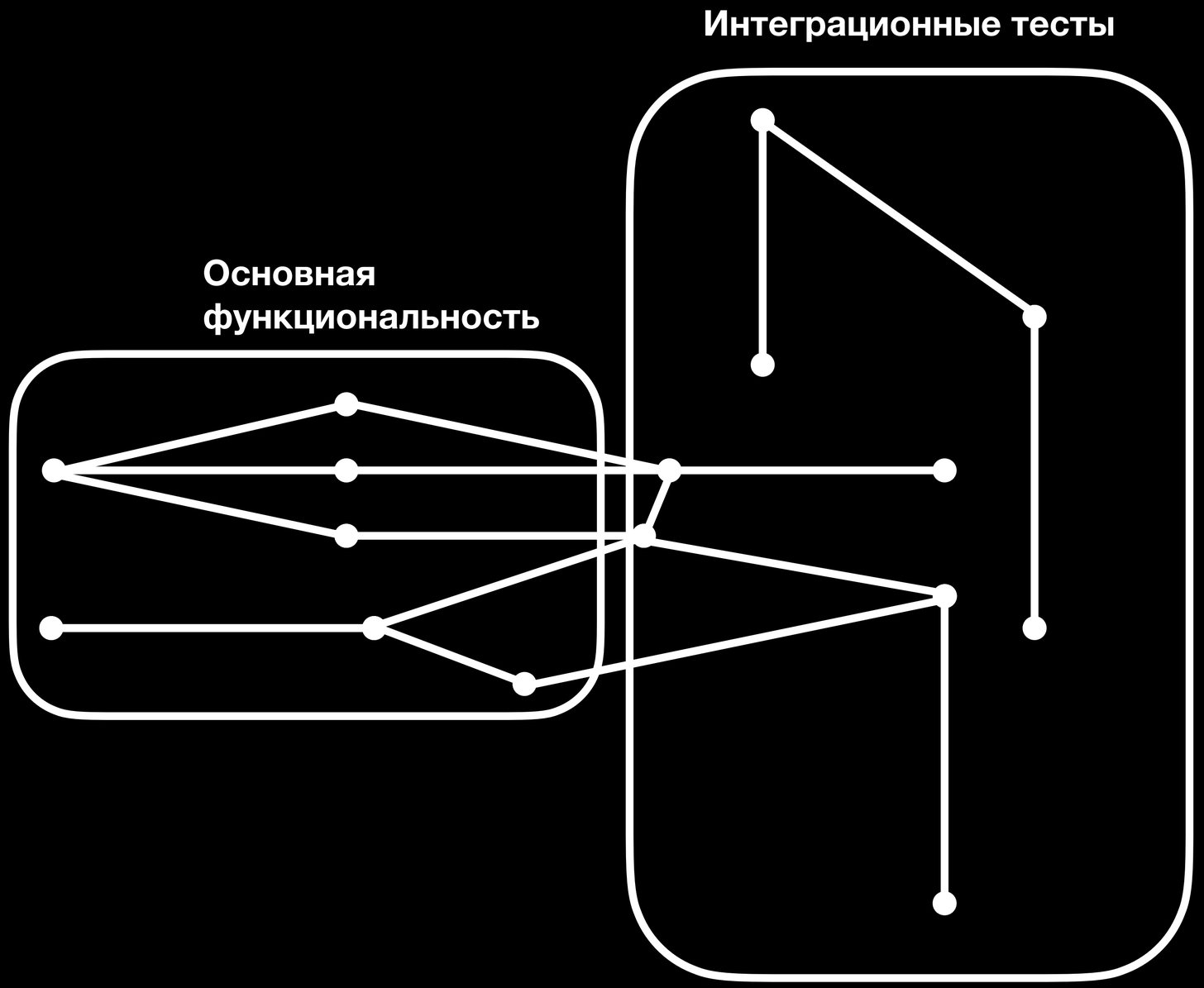
Рабочий процесс приема и отслеживания посылки

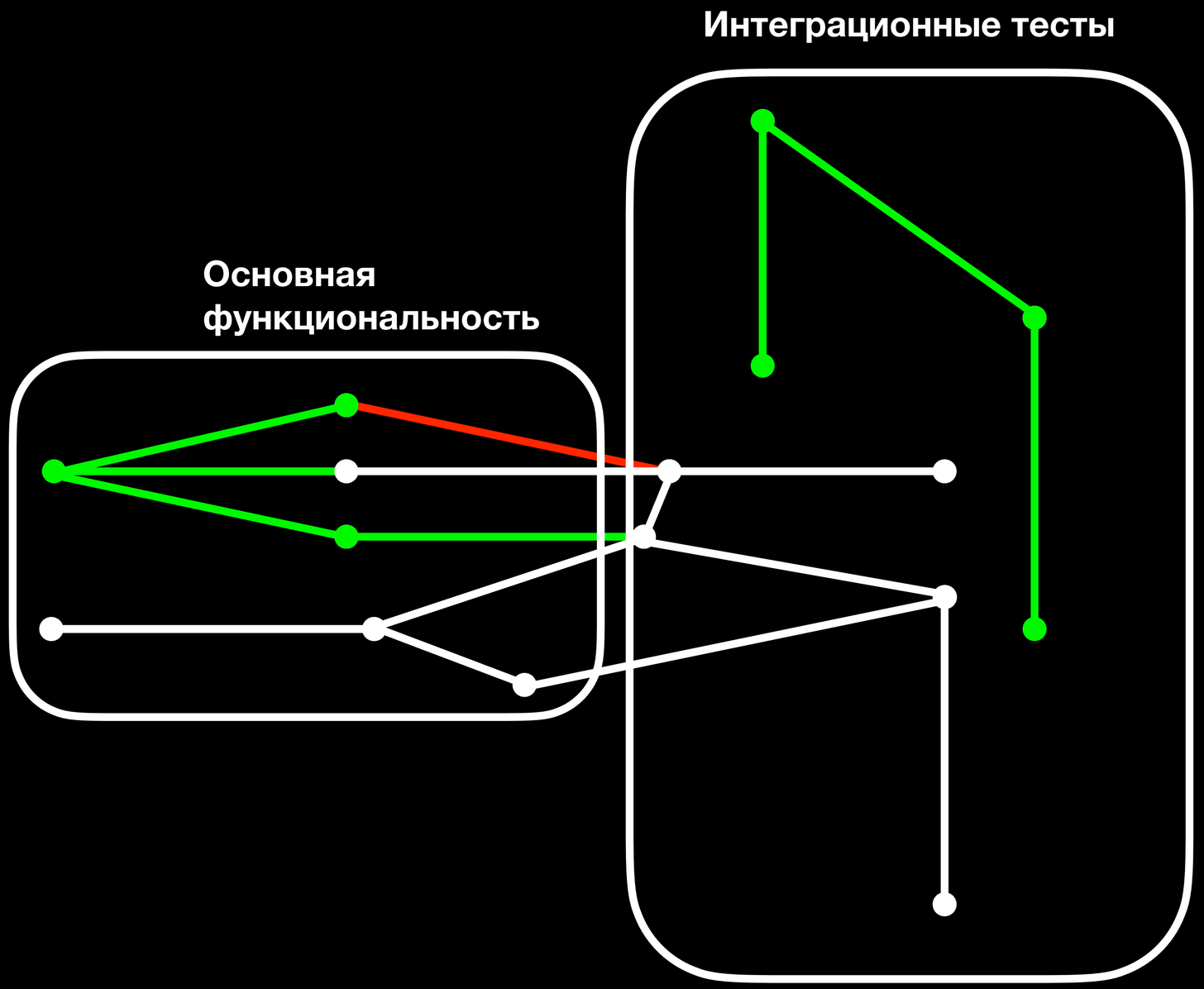


Алексей Родионов «Тестирование на основе сетей Петри»


Интеграционные тесты
Основная
функциональность

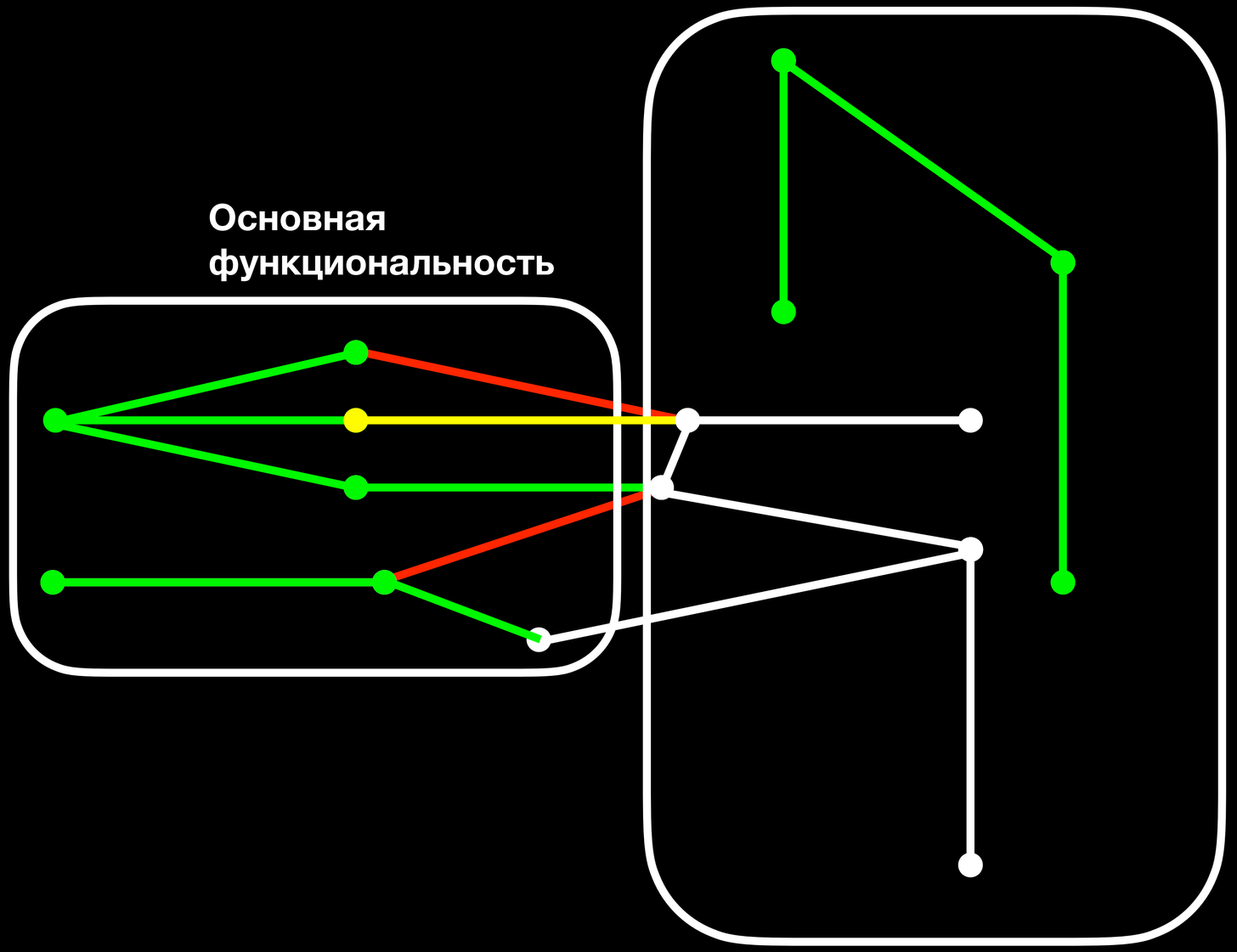

Интеграционные тесты
Основная
функциональность

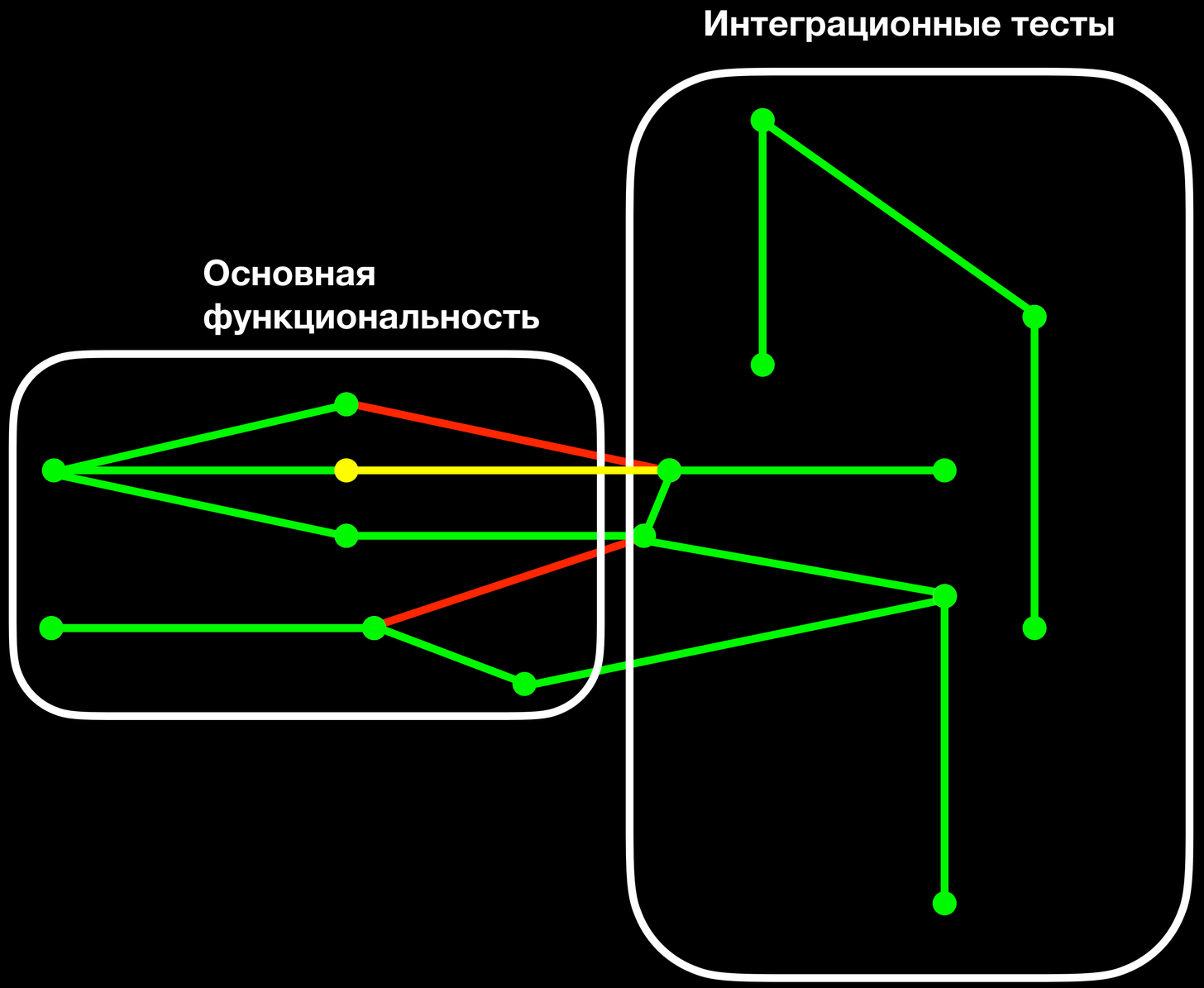

Интеграционные тесты
Основная
функциональность


Интеграционные тесты
Основная
функциональность

Интеграционные тесты

Основная
функциональность




Интеграционные тесты
Основная
функциональность

Минусы композиционного тестирования

1) Сложность
2) Ресурсоемкость
3) Риск ложных срабатываний
4) Зависимость от компонентов
5) Трудоемкость отладки

Плюсы композиционного тестирования

1) Сериализуемость
2) Переиспользование тестов и кейсов
3) Мок-система
4) Попарное тестирование
5) Систематизация
6) Инструмент
7) Огромный потенциал для AI
8) Масштабируемость
9) Идеально для тестов API

Мок-система

Тренажер тестирования Web API

GitHub

NotSecret.site

Плюсы композиционного тестирования

1) Сериализуемость
2) Переиспользование тестов и кейсов
3) Мок-система
4) Попарное тестирование
5) Систематизация
6) Инструмент
7) Огромный потенциал для AI
8) Масштабируемость
9) Идеально для тестов API

Композиционное тестирование на Python

Максим Кукликов