Ускоряем тесты на Python с помощью асинхронности. Весь тестовый прогон за время выполнения одного теста

Некрасов Александр

QA automation engineer

MTC DIGITAL

Асинхронность

И чуть-чуть историй из жизни

Вечер, котики, лапша

DIGITAL

А я сижу и лежу

DIGITAL

Очень важные дела

DIGITAL

Забираем заказ

DIGITAL

Проблемы

- Длительность интеграционных тестов
- Ограниченность ресурсов
- Блокировки

План

- Введение в многозадачность
- Асинхронность в Python
- Как писать асинхронные тесты

Как мы живем

- Асинхронный бэкенд
- Асинхронный фронтенд
- Синхронные тесты?

M T C DIGITAL

```
10 usages
class HttpClient(ABCHttpClient):

    def __init__(self, url: Optional[str] = None, _delay: int = 1) -> None:
        self.url: Optional[str] = url
        self._delay: int = _delay

    4 usages (1 dynamic)
    def get(self, path: str) -> Optional[Union[dict, bool]]:
        print(f'\tSend sync get req to {self.url + path}')
        time.sleep(self._delay)
        return True

    3 usages
    def post(self, path: str, body: Optional[dict] = None) -> Optional[Union[dict, bool]]:
        print(f'\tSend sync post req to {self.url + path}, with body {body}')
        time.sleep(self._delay)
        return True
```

```
10 usages
class DataBase(ABCDataBase):

    def __init__(self, connection: Optional[str] = None, _delay: int = 1) -> None:
        self.connection: Optional[str] = connection
        self._delay: int = _delay

    def select(self, query: Optional[str] = None) -> Optional[Union[dict, bool]]:
        print('select data from database')
        time.sleep(self._delay)
        return True

    3 usages
    def insert(self, query: Optional[str] = None) -> None:
        print('insert data from database')
        time.sleep(self._delay)
```

2 usages
```python
def first_test() -> bool:
    print('First test started')
    # Arrange
    client: HttpClient = HttpClient(url=DOMAIN, _delay=SLEEPING_SEC)
    db: DataBase = DataBase(connection=DB_CON, _delay=SLEEPING_SEC)
    db.insert('inert into ...')
    # Act
    post_res: dict = client.post('/post/req')
    get_res: dict = client.get('/get/req')
    db_data: dict = db.select('select * from ...')
    # Assert
    assert post_res
    assert get_res
    assert db_data
    print('First test finished')
    return True
```

1 usage
```python
def second_test() -> bool:
    print('Second test started')
    time.sleep(SLEEPING_SEC)
    print('Second test finished')
    return True
```

1 usage
```python
def third_test() -> bool:
    print('Third test started')
    time.sleep(SLEEPING_SEC)
    print('Third test finished')
    return True
```

```
2 usages
def first_test() -> bool:
    print('First test started')
    # Arrange
    client: HttpClient = HttpClient(url=DOMAIN, _delay=SLEEPING_SEC)
    db: DataBase = DataBase(connection=DB_CON, _delay=SLEEPING_SEC)
    db.insert('inert into ...')
    # Act
    post_res: dict = client.post('/post/req')
    get_res: dict = client.get('/get/req')
    db_data: dict = db.select('select * from ...')
    # Assert
    assert post_res
    assert get_res
    assert db_data
    print('First test finished')
    return True
```

Введение в многозадачность

Введение в многозадачность

- Многопроцессорность
- Многопоточность
- Асинхронность
- Виды многозадачности

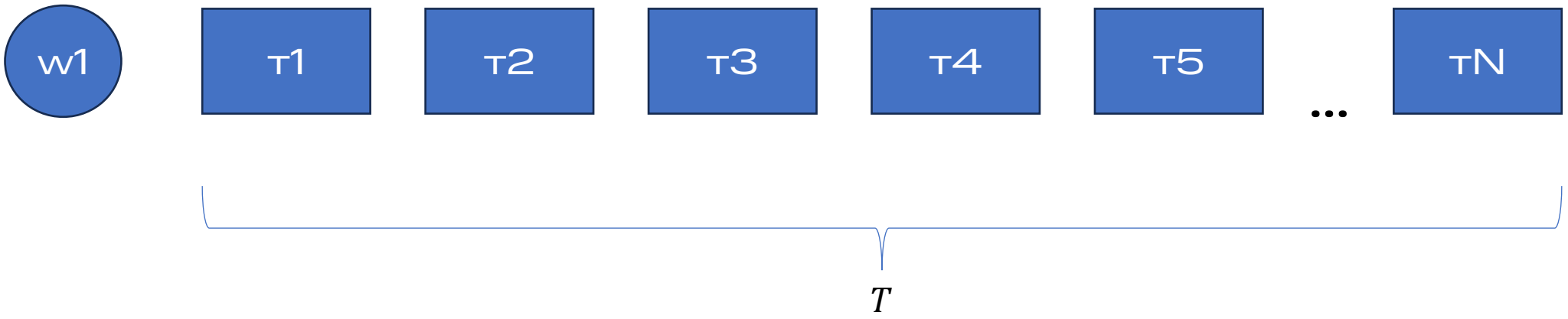
Легенда

- N – Количество задач
- W – Количество работающих воркеров
- $T = \frac{1}{N}\sum t_i$ – Время выполнения N задач в синхронном варианте
- K – Время необходимое на переключение задач

Синхронный код

DIGITAL

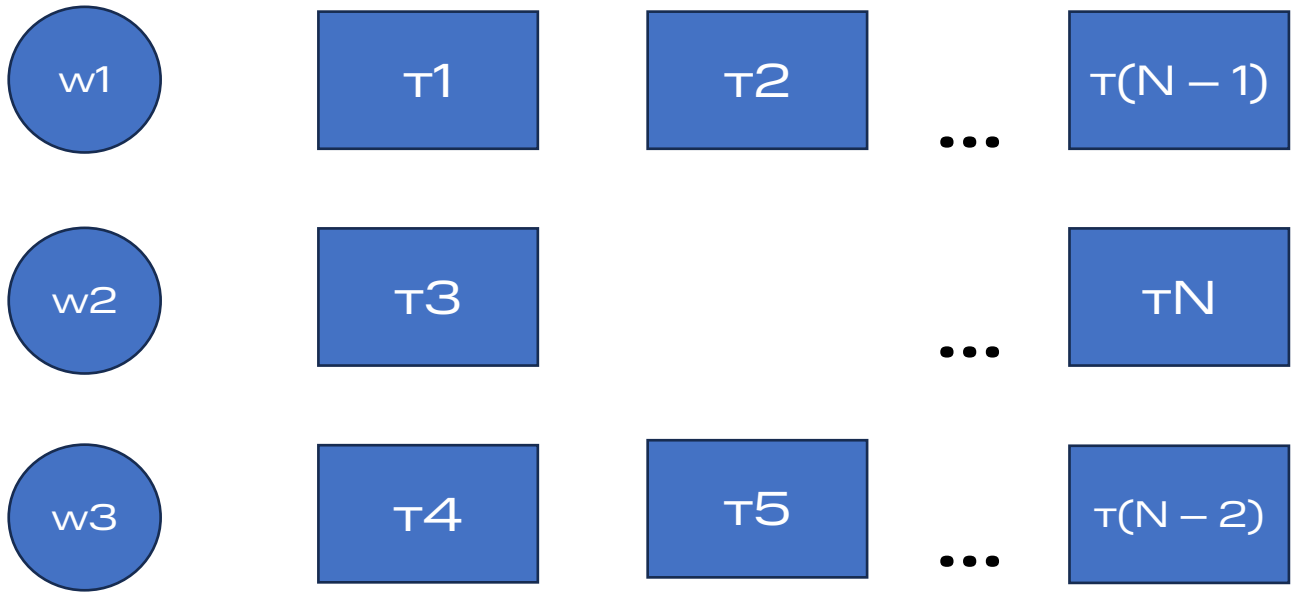
Синхронный код



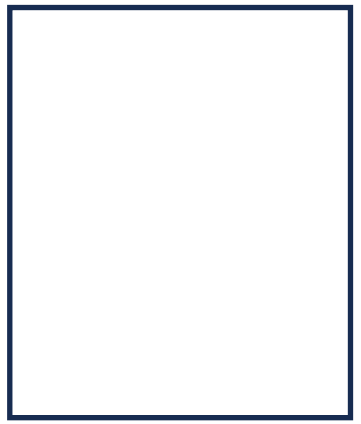
```
1 usage
def main():
    """Входная точка для запуска тестов в один поток, один процесс
    """
    tasks: list[Callable[[], bool]] = [first_test, second_test, third_test]
    print('Start test run')
    start: float = time.time()
    results: list[bool] = [task() for task in tasks]
    end: float = time.time()
    if all(results):
        print('All tests Successfully')
    else:
        print('All test not Successfully')
    print(f'Tests runs in {end - start} sec.')
```

```python
def many_test_main():
    """Входная точка для запуска тестов в один поток, один процесс
    с возможностью корректирования количества запускаемых тестов
    """
    test_count: int = 20
    tests: list[Callable[[], bool]] = [first_test for _ in range(test_count)]
    print('Start test run')
    start: float = time.time()
    results: list[bool] = [test() for test in tests]
    end: float = time.time()
    if all(results):
        print('All tests Successfully')
    else:
        print('All test not Successfully')
    print(f'Tests runs in {end - start} sec.')
```

Многопроцессорность / Многопоточность

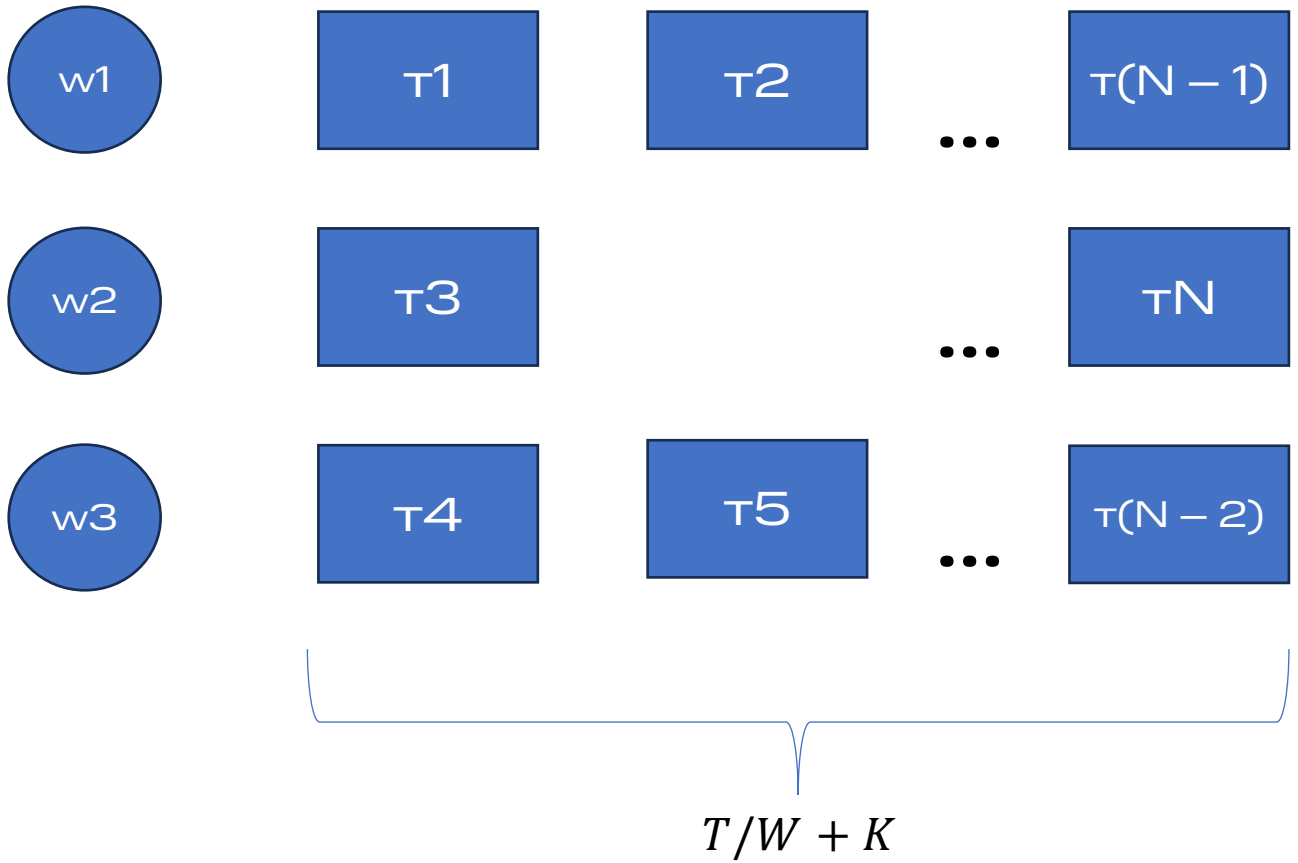


Выполненные



M T C DIGITAL

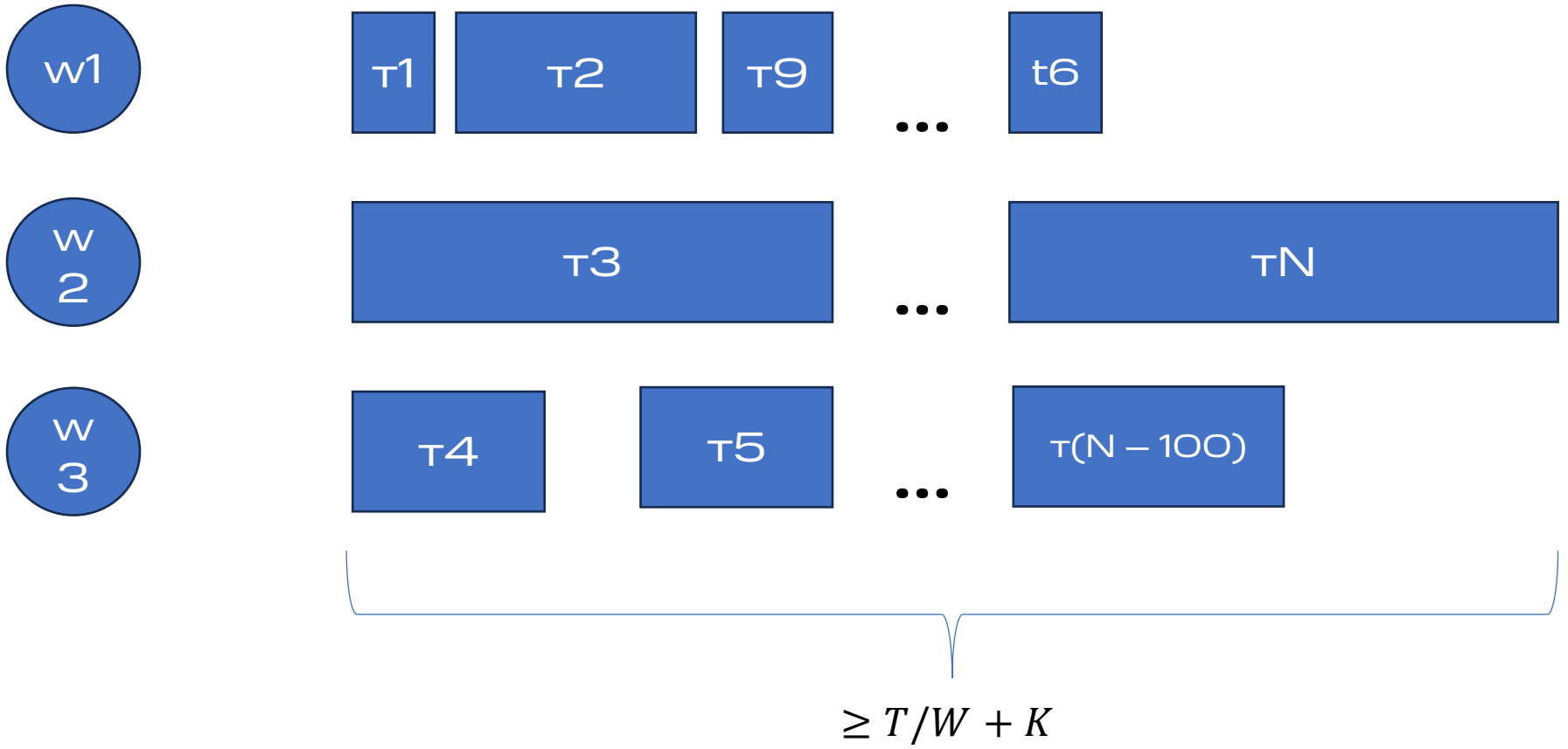
Многопроцессорность / Многопоточность



DIGITAL

```
1 usage
def many_test_main():
    """Входная точка для запуска тестов на нескольких ядрах процессора
    с возможностью корректирования количества запускаемых тестов
    """
    test_count: int = 20
    tests: list[Callable[[], bool]] = [first_test for _ in range(test_count)]
    with Pool(PROCESS_COUNT) as p:
        print('Start test run')
        start: float = time.time()
        runner: list[multiprocessing.pool.ApplyResult] = [p.apply_async(test) for test in tests]
        results: list[bool] = [task.get() for task in runner]
        end: float = time.time()
    if all(results):
        print('All tests Successfully')
    else:
        print('All test not Successfully')
    print(f'Tests runs in {end - start} sec.')
```

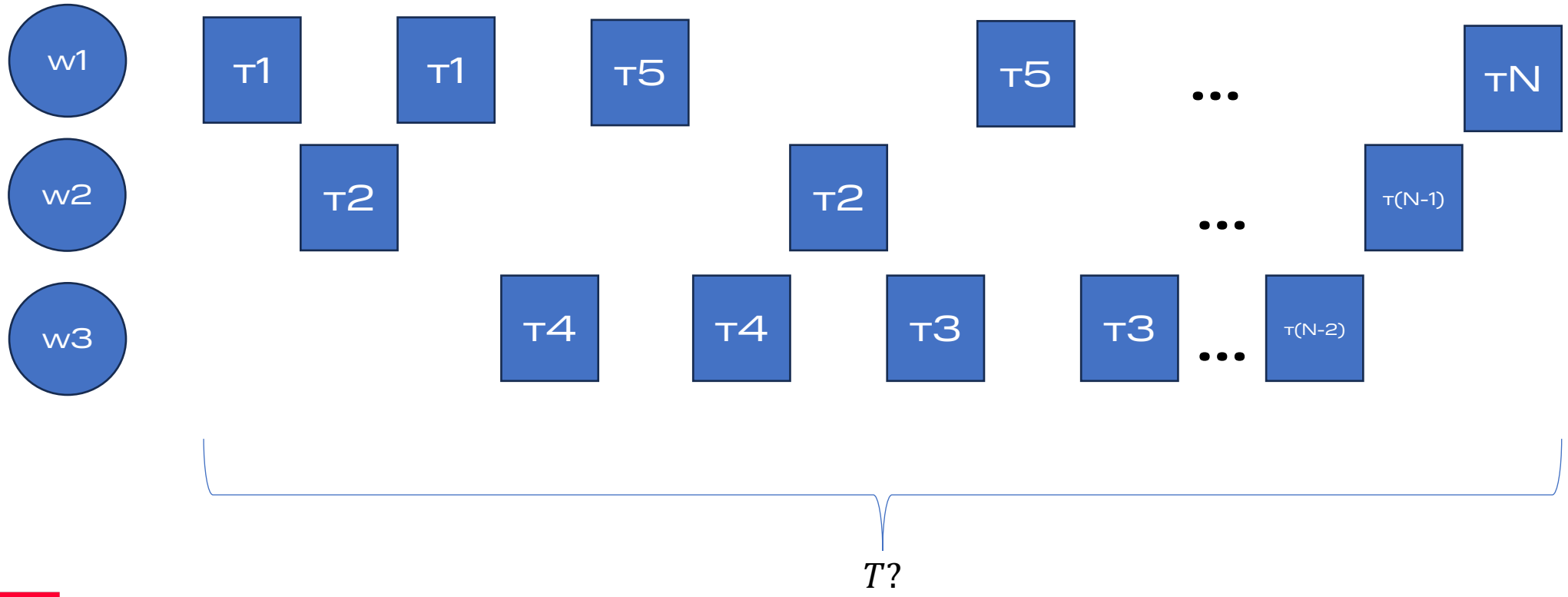
Многопроцессорность / Многопоточность



$$\geq T/W + K$$

DIGITAL

Многопоточность (python)



DIGITAL

Задача 1

Время

```python
def fibonacci(number: int) -> int:
    if number < 2:
        return number
    return fibonacci(number - 1) + fibonacci(number - 2)
```

Этот кот будет все время потреблять ресурсы процессора

Задача 2

Время

Обработка данных
(задействует CPU)

Отправка запроса
(просто ожидает ответа)

```
for i_data in data:
    if not is_correct(i_data):
        i = convert(i_data)
    model.add(convert(i_data))

resp = request.post(url=url, json=model.to_json())

for i_data in resp:
    if not is_correct(i_data):
        i = convert(i_data)
    resp_model.add(convert(i_data))
```

Обработка данных
(задействует CPU)

MTC DIGITAL

Задача 2

Во время выполнения этого кота процессор прохлаждается

Время

```
for i_data in data:
    if not is_correct(i_data):
        i = convert(i_data)
    model.add(convert(i_data))
```

```
resp = request.post(
    url=url,
    json=model.to_json()
)
```

```
for i_data in resp:
    if not is_correct(i_data):
        i = convert(i_data)

resp_model.add(convert(i_data))
```

Этот кот будет все время потреблять ресурсы процессора

MTC DIGITAL

Задача 3

Время

Обработка данных
(задействует CPU)

Отправка запроса
(просто ожидает ответа)

```
for i_data in data:
    if not is_correct(i_data):
        i_data = convert(i_data)
    model.add(convert(i_data))

query = cursor.execute('SELECT * FROM `table_1`')

id_ = query.fetchone().get('id')
resp=request.post(url=url.format(id_),json=model.to_json())
```

Отправка запроса
(просто ожидает ответа)

Задача 3

Этот кот будет все время потреблять ресурсы процессора

Время

```
for i_data in data:
    if not is_correct(i_data):
        i_data =
convert(i_data)
    model.add(convert(i_data))
```

```
query = cursor.execute(
    'SELECT * FROM `table_1`'
)
```

```
id_ = query.fetchone().get('id')

resp = request.post(
    url=url.format(id_),
    json=model.to_json()
)
```

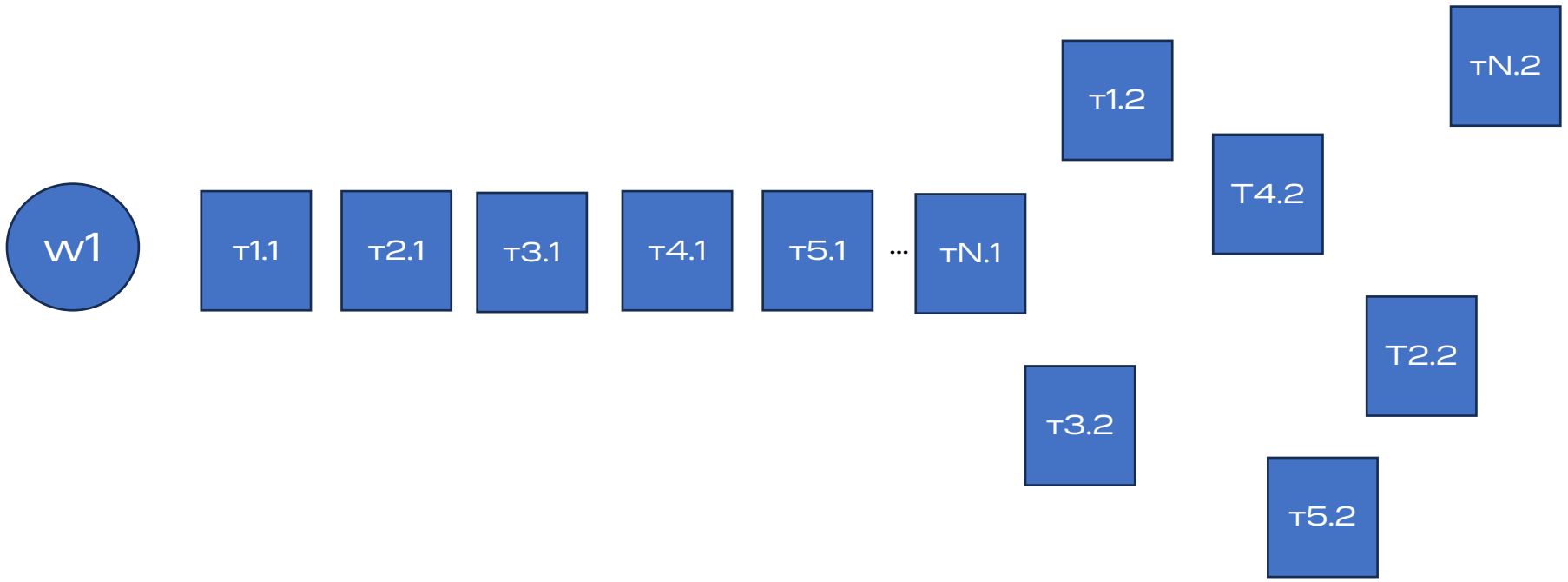
Во время выполнения этого кота процессор прохлаждается

DIGITAL

Легенда ++

- N – Количество задач
- W – Количество работающих воркеров
- $T = \frac{1}{N}\sum t_i$ – Время выполнения N задач в синхронном варианте
- K – Время, необходимое на переключение задач
- D – Суммарное время, которое задачи ждут ответа от IO операции
- P – Суммарное время когда внутри задач выполняется счетный код

Асинхронность

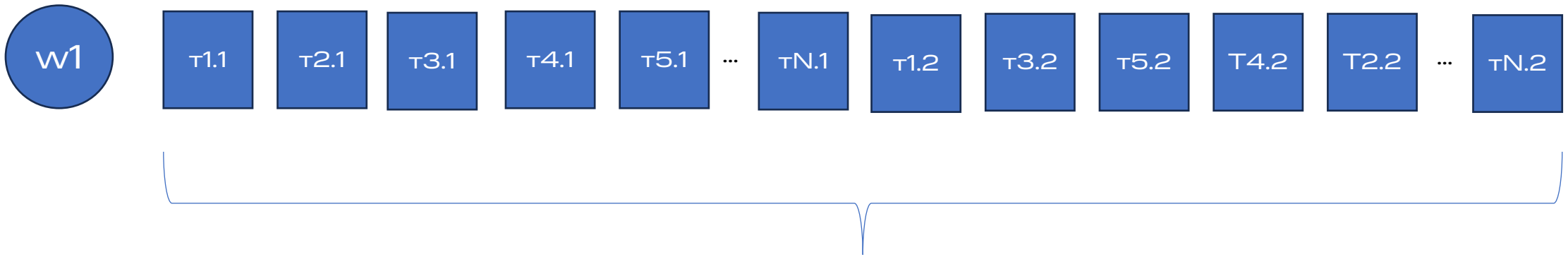


Выполненные

?

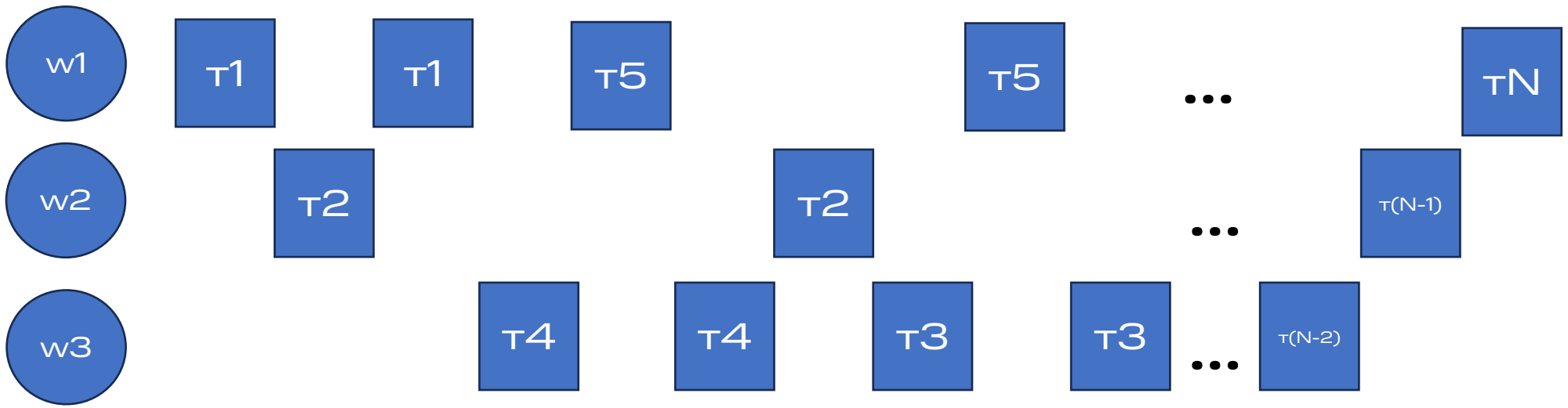
DIGITAL

Асинхронность



$$T - D + \max(D_i) + K = P + \max(D_i) + K$$

Многопоточность (python)

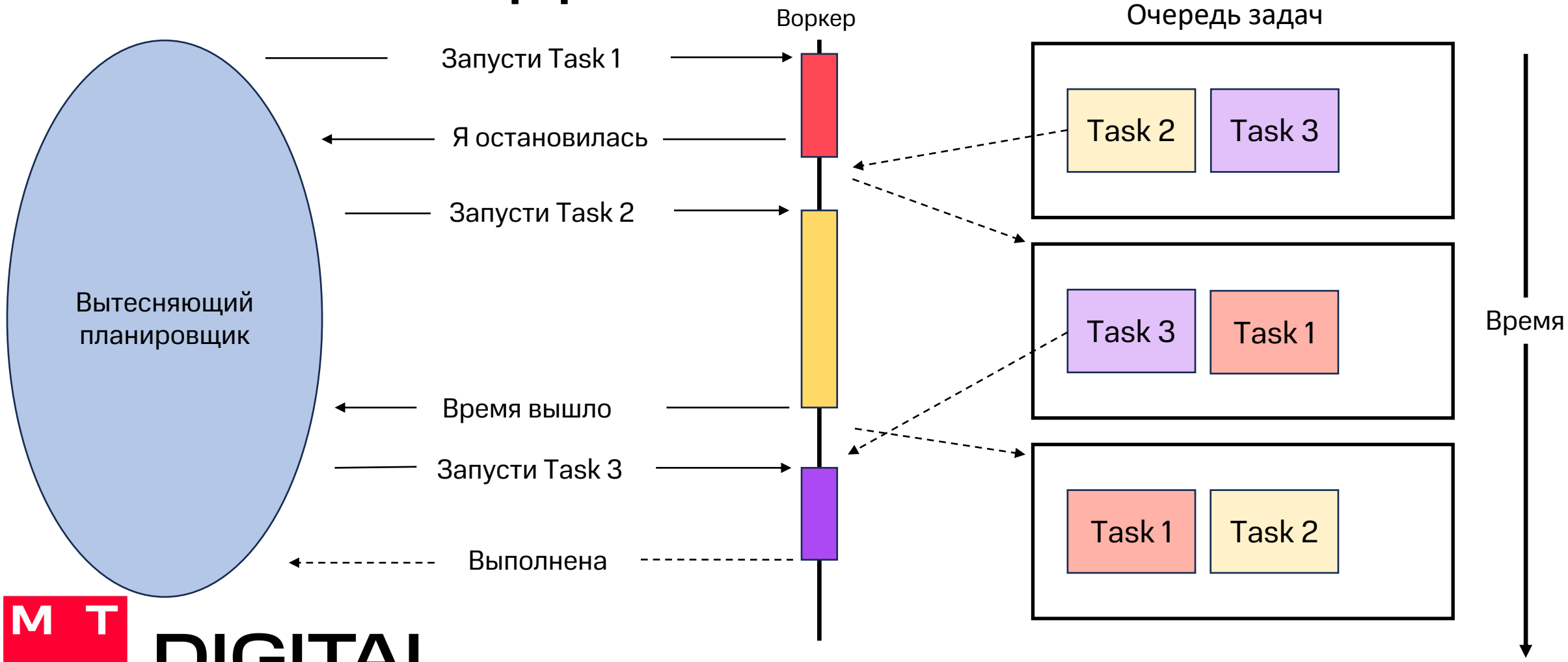


$$T/W + K + P$$

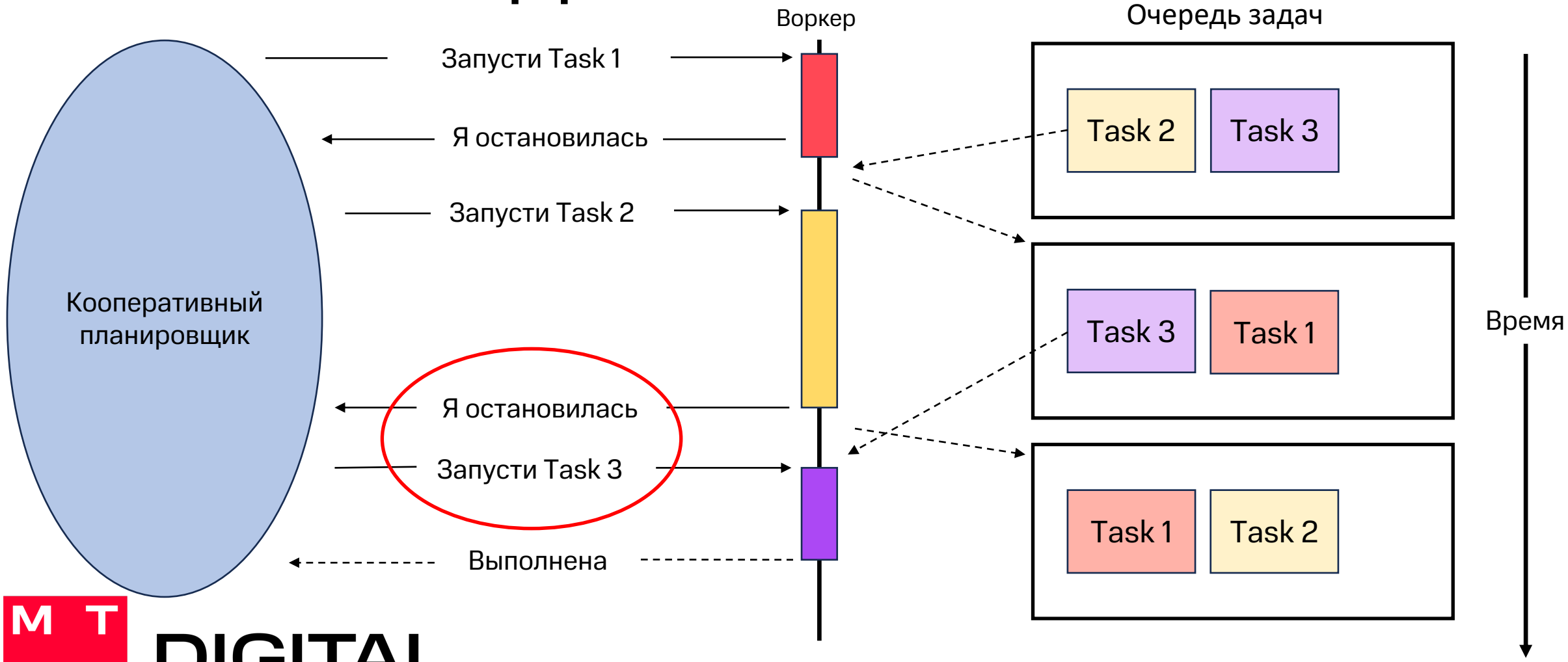
Многозадачность

- Вытесняющая многозадачность
- Кооперативная многозадачность

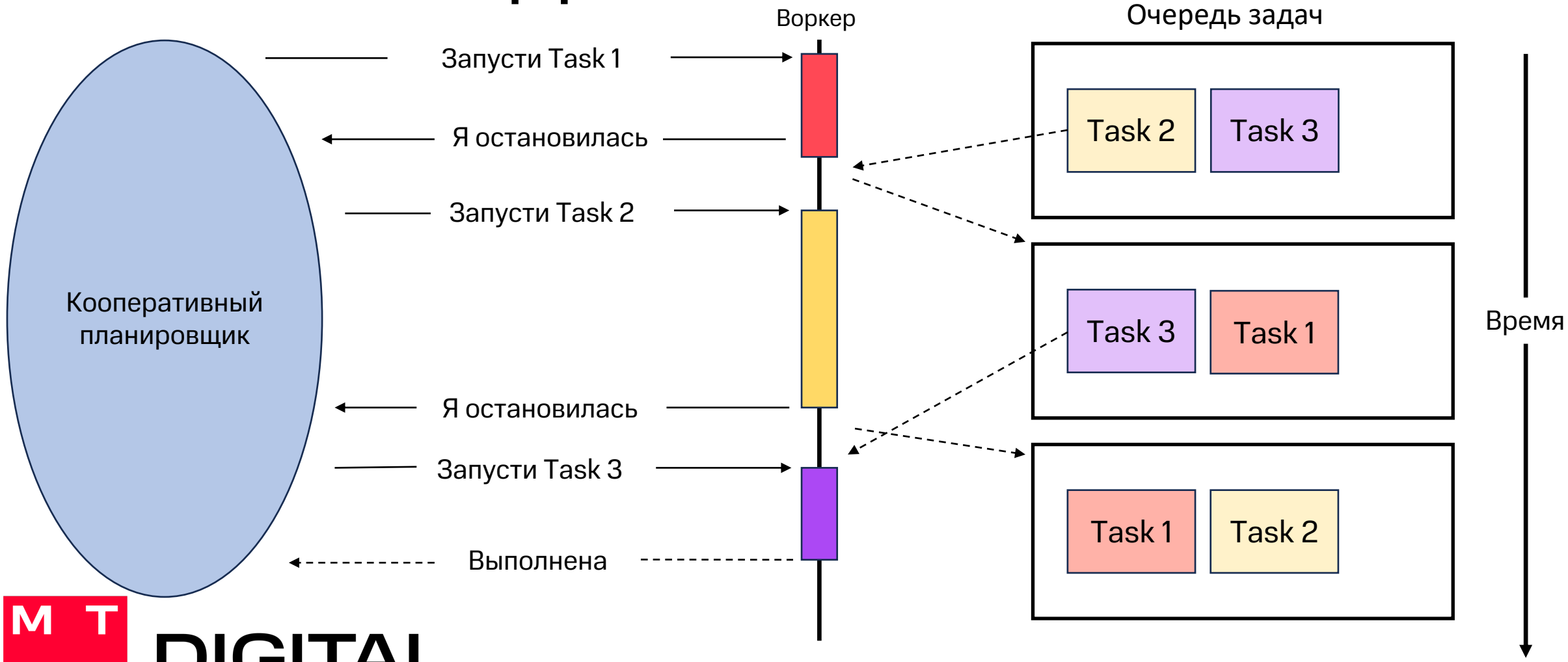
Вытесняющая многозадачность



Кооперативная многозадачность

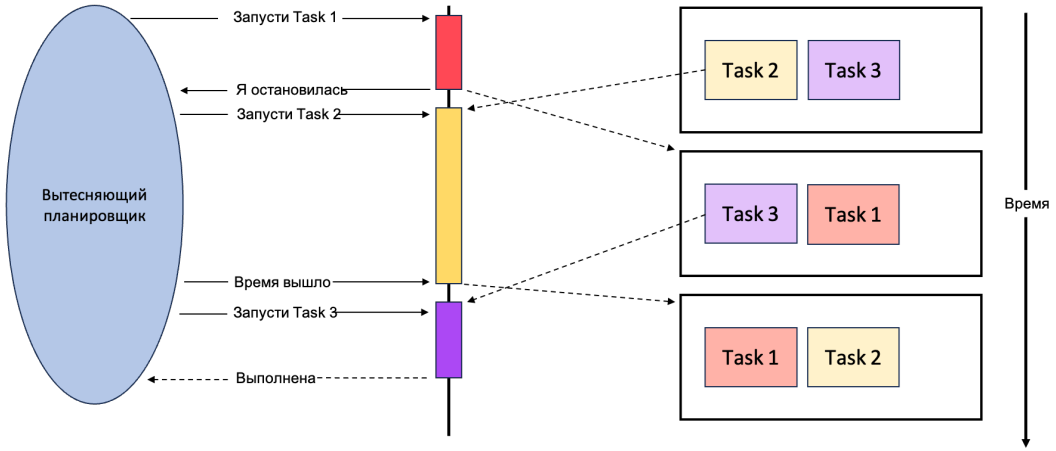


Кооперативная многозадачность



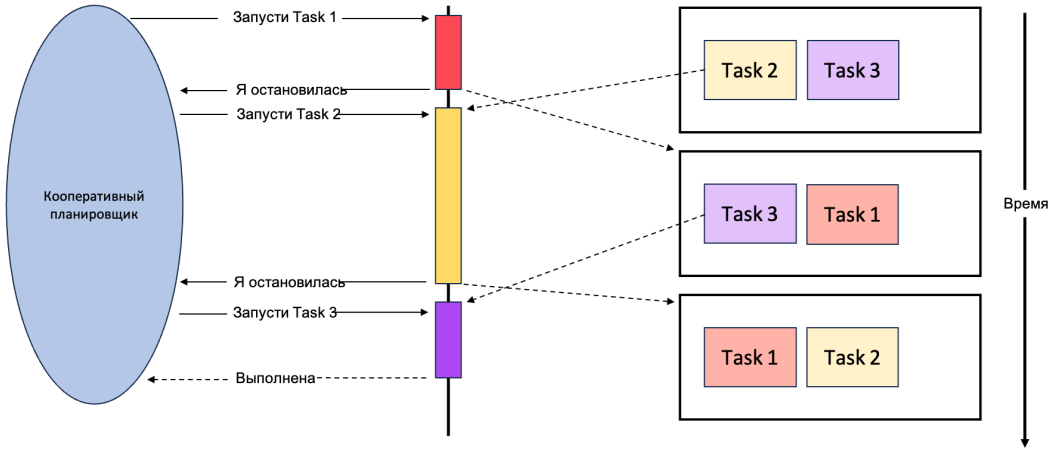
Так в чем же разница?

Вытесняющая



- Task 1 – Process 1
- Task 2 – Process 2
- Task 3 – Process 3

Кооперативная



- Task 1 – Process 1
- Task 2 – Process 1
- Task 3 – Process 1

Мини-итоги

- Вспомнили понятия асинхронности, многопроцессорности / многопоточности
- Поняли из чего состоят задачи
- Узнали преимущества разных типов многозадачности

Введение в asyncio

Введение в asyncio

Документация:

- asyncio is a library to write **concurrent** code using the **async/await** syntax.
- asyncio is used as a foundation for multiple Python asynchronous frameworks that provide high-performance network and web-servers, database connection libraries, distributed task queues, etc.

DIGITAL

Объекты asyncio

- Coroutine
- Future
- Task

Coroutine

```
async def first_test() -> bool:
    ...
    return True
```

Future

```
async def main():
    my_future = asyncio.Future()
    ...
    my_future.set_result(42)
    ...
    res = await my_future
```

Task

```
async def main():
    task_1 = asyncio.create_task(first_test())
    ...
    res = await task_1
```

```
3 usages
async def first_test() -> bool:
    print('First test started')
    # Arrange
    client: AsyncHttpClient = AsyncHttpClient(url=DOMAIN, _delay=SLEEPING_SEC)
    db: AsyncDataBase = AsyncDataBase(connection=DB_CON, _delay=SLEEPING_SEC)
    await db.insert('inert into ...')
    # Act
    post_res: dict = await client.post('/post/req')
    get_res: dict = await client.get('/get/req')
    db_data: dict = await db.select('select * from ...')
    # Assert
    assert post_res
    assert get_res
    assert db_data
    print('First test finished')
    return True
```

```
4 usages
class AsyncHttpClient(ABCHttpClient):

    def __init__(self, url: Optional[str] = None, _delay: int = 1) -> None:
        self.url: Optional[str] = url
        self._delay: int = _delay

    2 usages (1 dynamic)
    async def get(self, path: str) -> Optional[Union[dict, bool]]:
        print(f'\tSend async get req to {self.url + path}')
        await asyncio.sleep(self._delay)
        return True

    1 usage
    async def post(self, path: str, body: Optional[dict] = None) -> Optional[Union[dict, bool]]:
        print(f'\tSend async post req to {self.url + path}, with body {body}')
        await asyncio.sleep(self._delay)
        return True
```

```
4 usages
class AsyncDataBase(ABCDataBase):

    def __init__(self, connection: Optional[str] = None, _delay: int = 1) -> None:
        self.connection: Optional[str] = connection
        self._delay: int = _delay

    async def select(self, query: Optional[str] = None) -> Optional[Union[dict, bool]]:
        print('select data from database')
        await asyncio.sleep(self._delay)
        return True

    1 usage
    async def insert(self, query: Optional[str] = None) -> None:
        print('insert data from database')
        await asyncio.sleep(self._delay)
```

```
1 usage
async def main_with_locking():
    """Входная точка для запуска асинхронных тестов без использования задач
    """
    tests: list[Callable[[], Coroutine]] = [first_test, second_test, third_test]
    print('Start test run')
    start: float = time.time()
    results: list[bool] = [await test() for test in tests]
    end: float = time.time()
    if all(results):
        print('All tests Successfully')
    else:
        print('All test not Successfully')
    print(f'Tests runs in {end - start} sec.')
```

```python
async def main_without_locking():
    """Входная точка для запуска асинхронных тестов с использованием задач
    """
    tests: list[Callable[[], Coroutine]] = [first_test, second_test, third_test]
    tasks: list[asyncio.Task] = [asyncio.create_task(test()) for test in tests]
    print('Start test run')
    start: float = time.time()
    await asyncio.sleep(0)
    results: list[bool] = [await task for task in tasks]
    end: float = time.time()
    if all(results):
        print('All tests Successfully')
    else:
        print('All test not Successfully')
    print(f'Tests runs in {end - start} sec.')
```

Мини-итоги

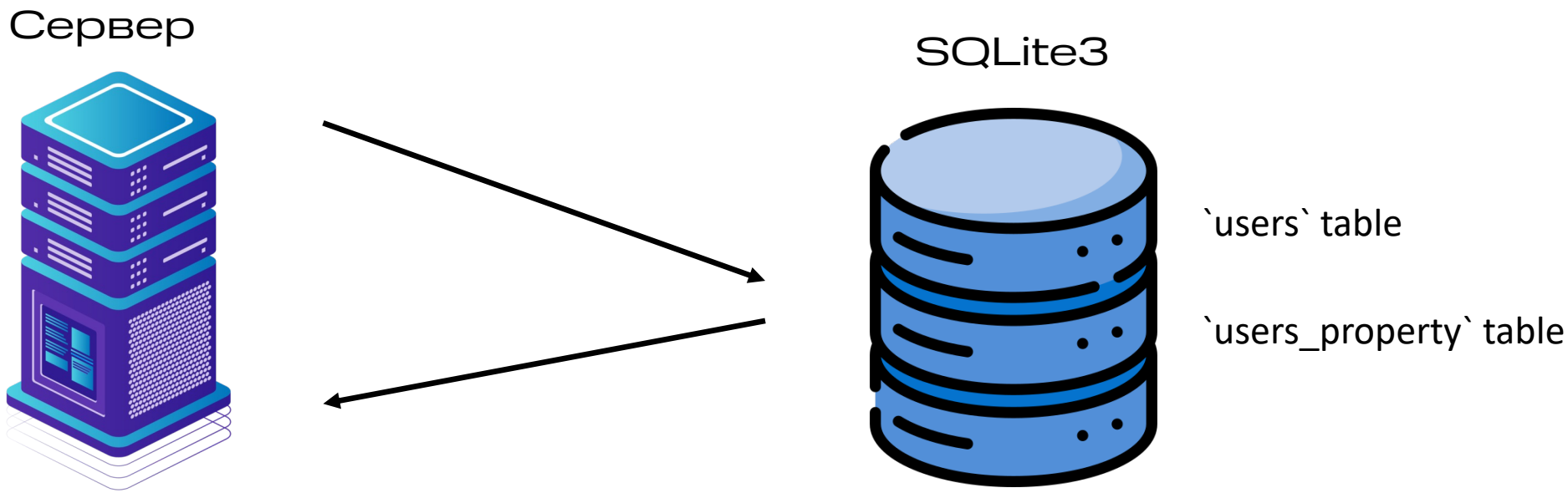
- Использовали базовый функционал asyncio
- Узнали, что такое coroutine, future, task
- Написали первое асинхронное приложение

Как писать асинхронные тесты

План

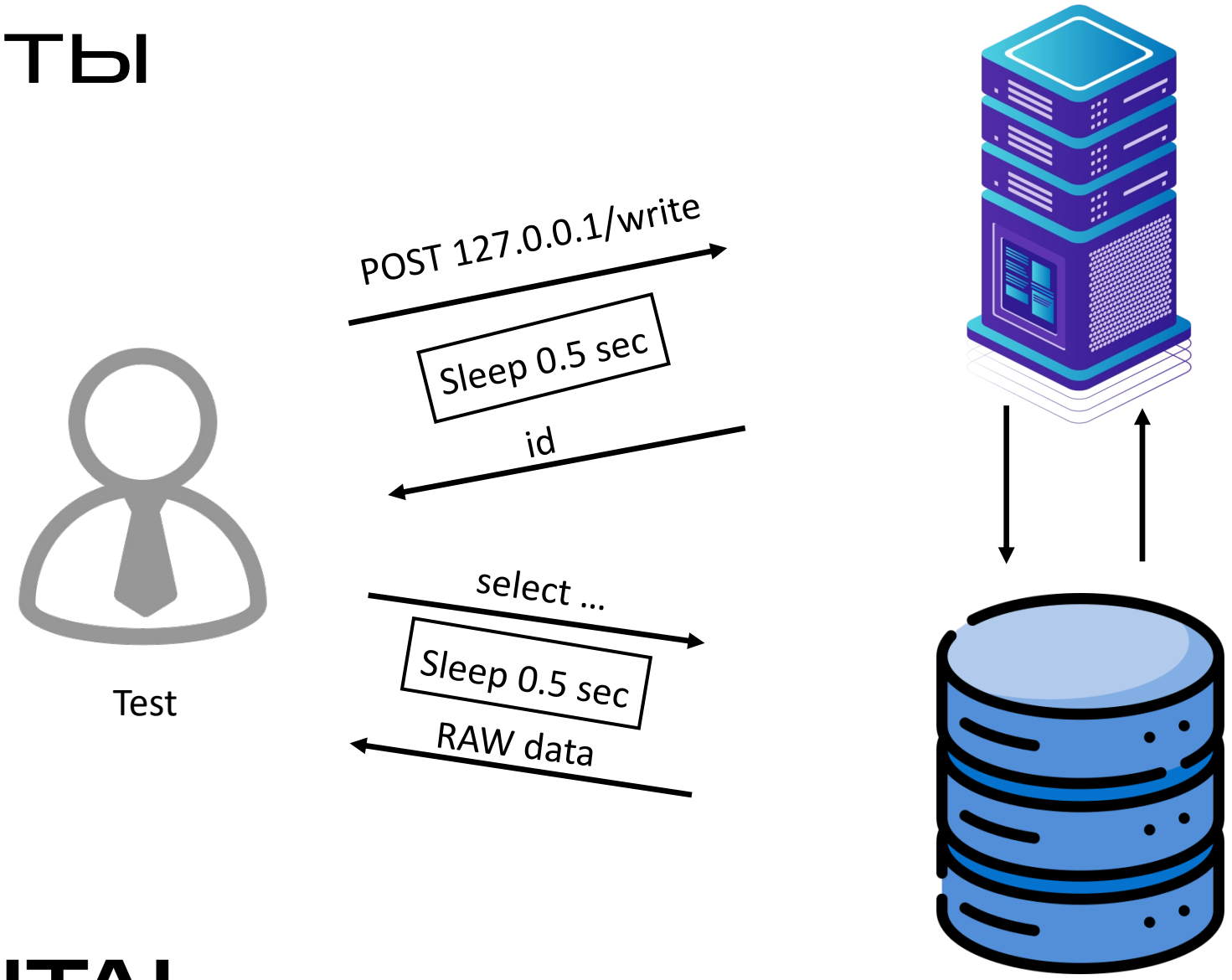
- Пишем первый тест
- Ускоряем асинхронные тесты
- Украшаем асинхронные тесты

Приложение



DIGITAL

Тесты


POST 127.0.0.1/write
Sleep 0.5 sec
id
select ...
Sleep 0.5 sec
RAW data
Test

Библиотеки

- aiohttp
- aiosqlite
- requests
- pytest

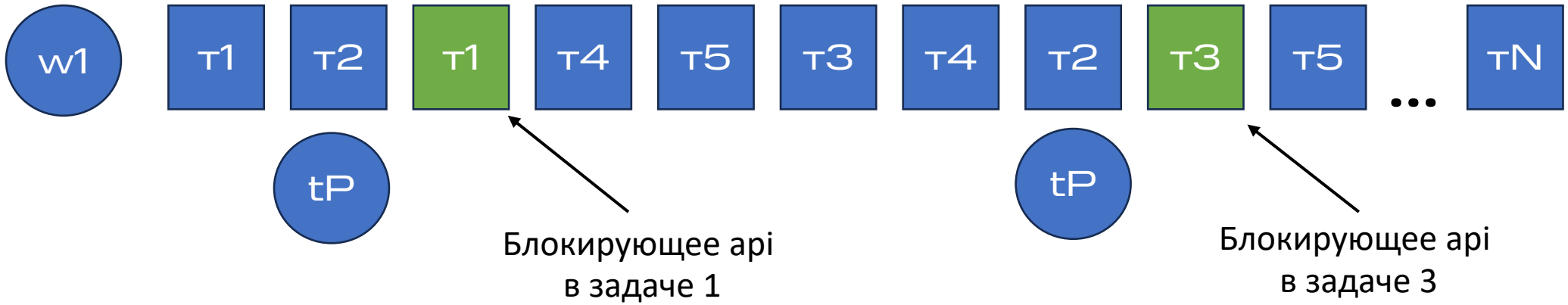
pytest-asyncio-cooperative

pip install pytest-asyncio-cooperative

DIGITAL

Лайфкодинг

Использование блокирующих api



DIGITAL

HEISENBUG

Слоеный фреймворк автотестирования на стеке. Архитектура, примеры на практике и подводные камни

Роман Помелов

VK / Skillbox

Как перейти на асинхронные тесты

- Полный рефакторинг

- Замена синхронного api

DIGITAL

Реальные сравнения

Sync vs Async

Страхование > Запуски > backend_tests - 2dce0bff

38 160

Обзор Результаты тестов Ошибки Графики **Временная шкала**

0 0s - 1m 47s 1m 47s

0s 1m 08s 2m 48s 4m 28s 6m 0... 7m 48s 9m 28s 11m 0... 12m ... 14m ... 16m ... 17m ... 19m ... 21m ... 2...

runner-tmtbrvfy-project-165172-concurrent-0

Страхование > Запуски > backend_tests - 393b7a67

37 1 160

Обзор Результаты тестов Ошибки Графики **Временная шкала**

0 0s - 1ms 1ms

0s 17s 796ms 37s 796ms 57s 796ms 1m 17s 1m 37s 1m 57s 2m 17s

runner-kvfetmxj-project-165172-concurrent-0



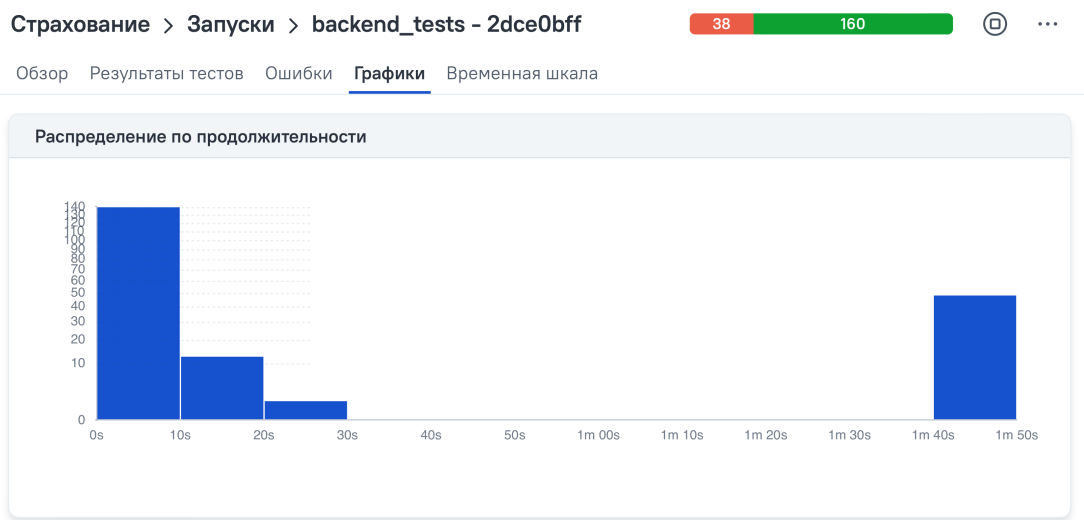




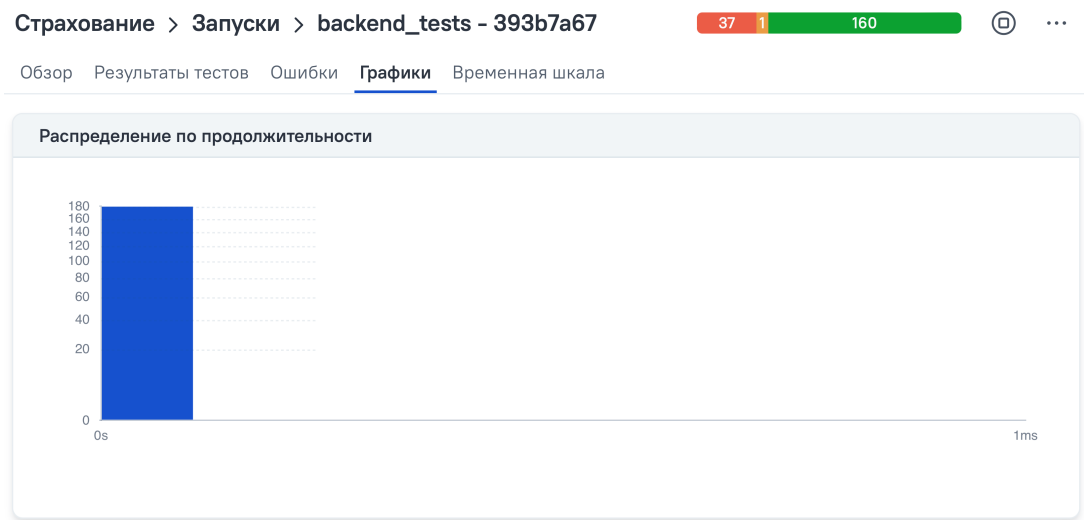

0s 17s 796ms 37s 796ms 57s 796ms 1m 17s 1m 37s 1m 57s 2m 17s

Распределение по продолжительности

Sync:



Async:



MTC DIGITAL

Взаимодействие с Allure

Что работает

- Логирование тестов
- Логирование параметров в параметризованных тестах
- Логирование трассировки

Что не работает

- Шаги
- Прикрепление чего-либо (attach)
- Аргументы передаваемые в функции в тестах

Результаты перехода на асинхронность

- Сократилась суммарная длительность выполнения асинхронных тестов
- Исчезла необходимость в большом количестве ресурсов
- Исчезли потенциальные ограничения тестов

M T C DIGITAL

Плюсы

И минусы

Плюсы асинхронных тестов

- Быстрее работают на слабом железе
- Позволяют запускать больше параллельных тестов
- Мы знаем в какой момент выполнение одного теста может переключиться на выполнение другого
- Операционная система перестает заниматься менеджментом тестов
- Появляется возможность выполнения долгих тестов без увеличения длительность всего прогона

Минусы асинхронных тестов

- Усложнение кодовой базы
- Усложнение процесса дебаггинга
- Необходимость перехода на асинхронные api
- Требуется атомарность тестов
- Allure некорректно отображает асинхронные тесты

Итоги:

- Узнали разницу между вытесняющей и кооперативной многозадачностями
- Написали первую асинхронную программу
- Научились ускорять тестовые прогоны с помощью асинхронности

Спасибо за внимание!

Тут вы можете найти код из доклада →

Тут вы можете найти меня →

@POKSLIUN

MTC DIGITAL