



# Реализация поддержки асинхронного программирования для фреймворка DSLab

Артём Макогон

21 мая 2023 г.

# Введение

## Описание предметной области

- Большой объем данных/вычислений
- Надежность
- Масштабируемость



# Введение

## Описание предметной области

- Большой объем данных/вычислений
- Надежность
- Масштабируемость



- > Недетерминированные алгоритмы
- > Сложное тестирование





# Введение

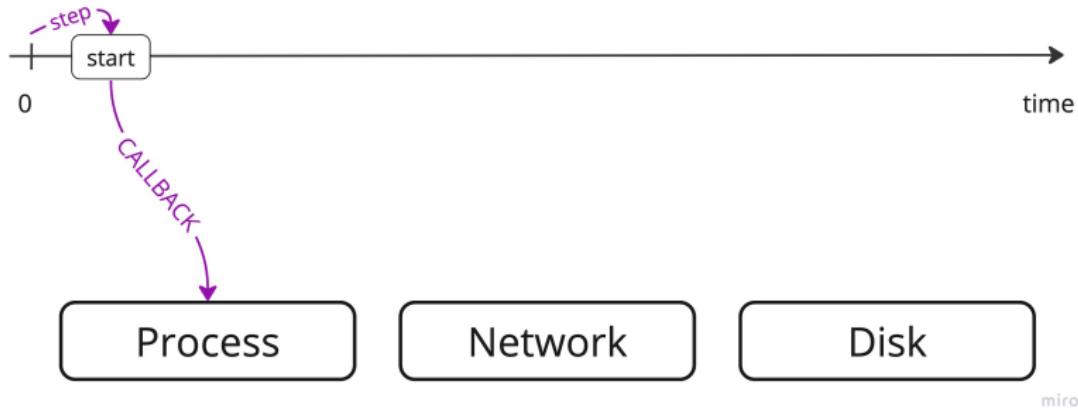
## Дискретно-событийное моделирование



Исполнение симуляции

# Введение

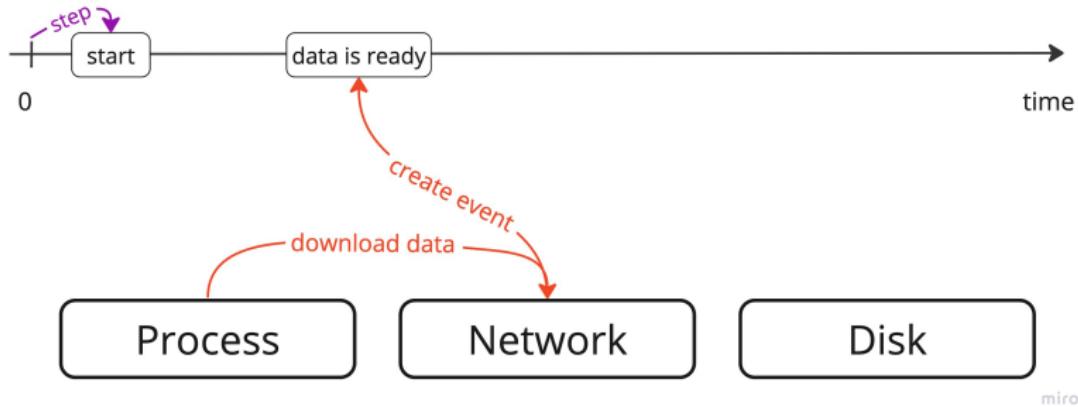
## Дискретно-событийное моделирование



Исполнение симуляции

# Введение

## Дискретно-событийное моделирование

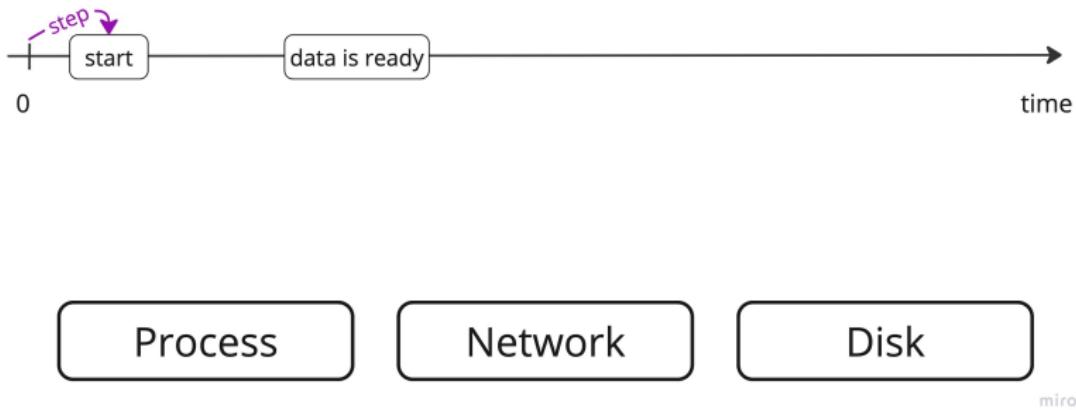


Исполнение симуляции



# Введение

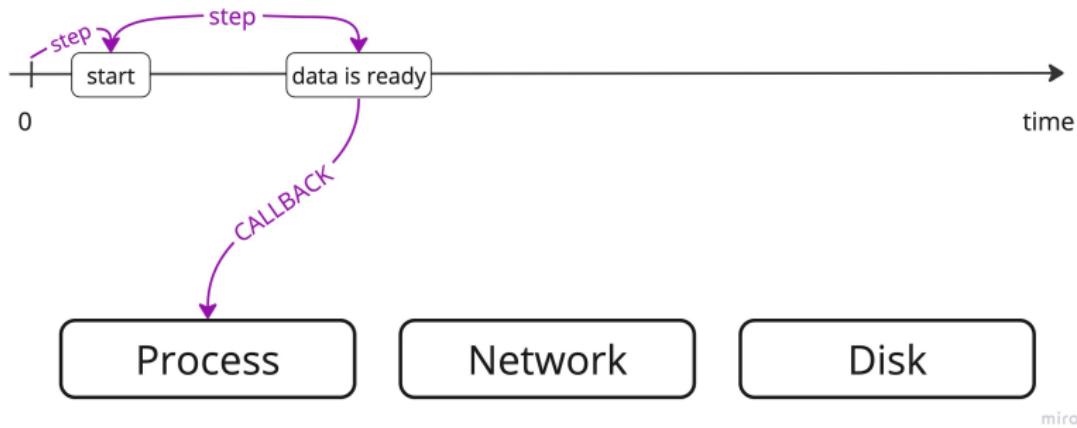
## Дискретно-событийное моделирование



Исполнение симуляции

# Введение

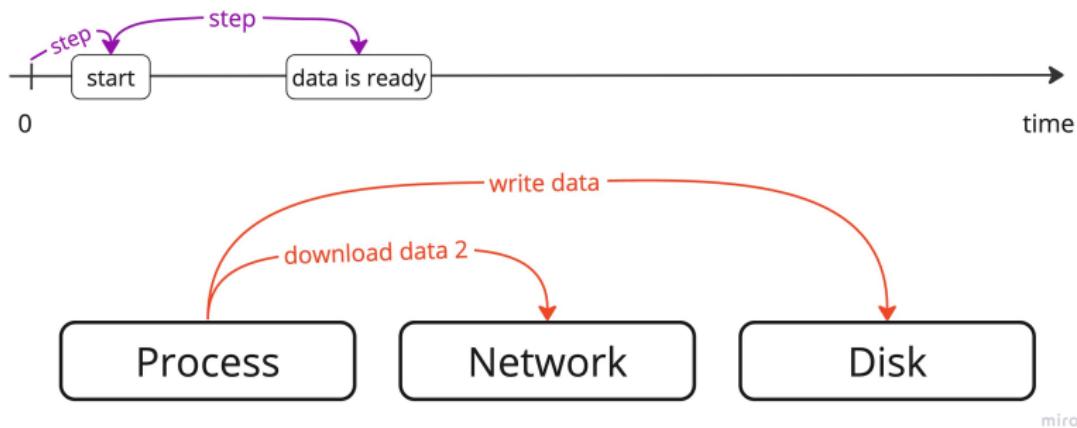
## Дискретно-событийное моделирование



Исполнение симуляции

# Введение

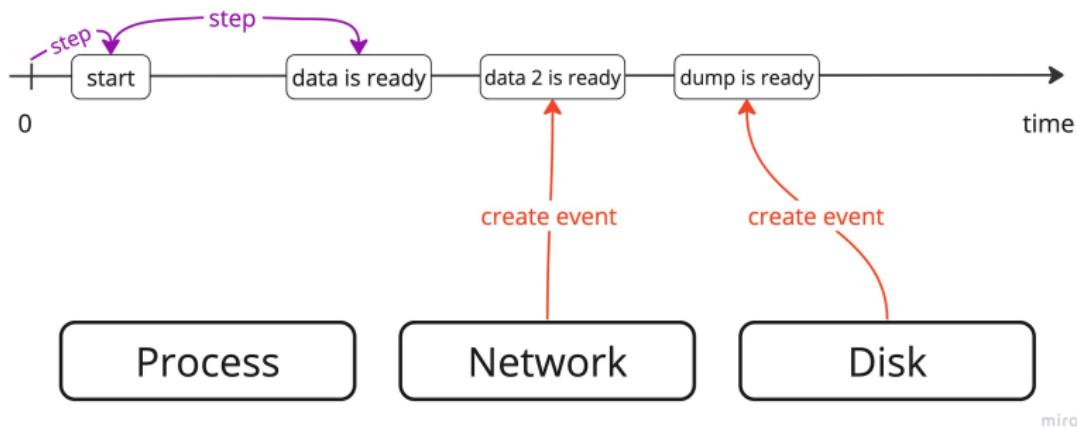
## Дискретно-событийное моделирование



Исполнение симуляции

# Введение

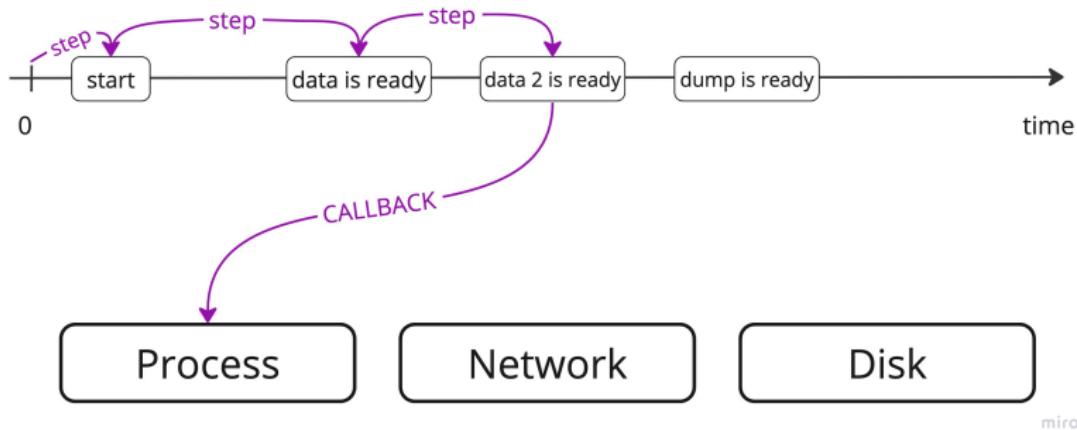
## Дискретно-событийное моделирование



Исполнение симуляции

# Введение

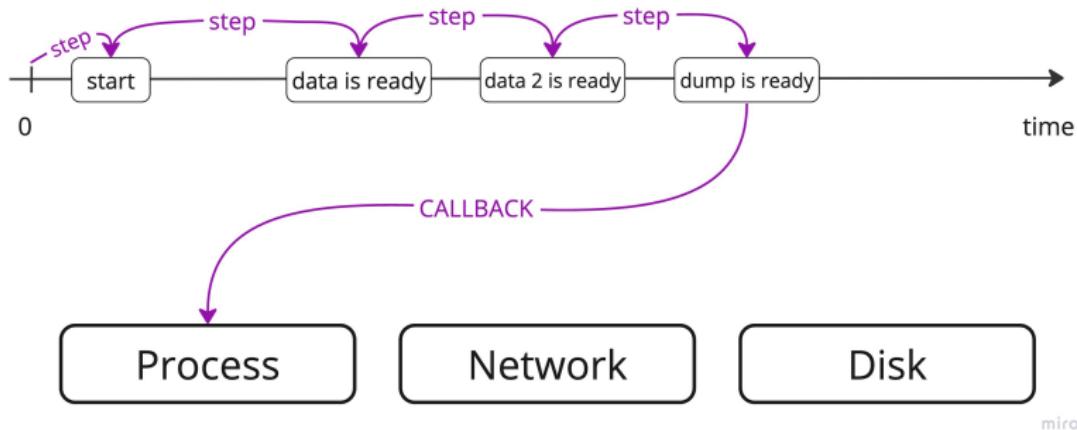
## Дискретно-событийное моделирование



Исполнение симуляции

# Введение

## Дискретно-событийное моделирование

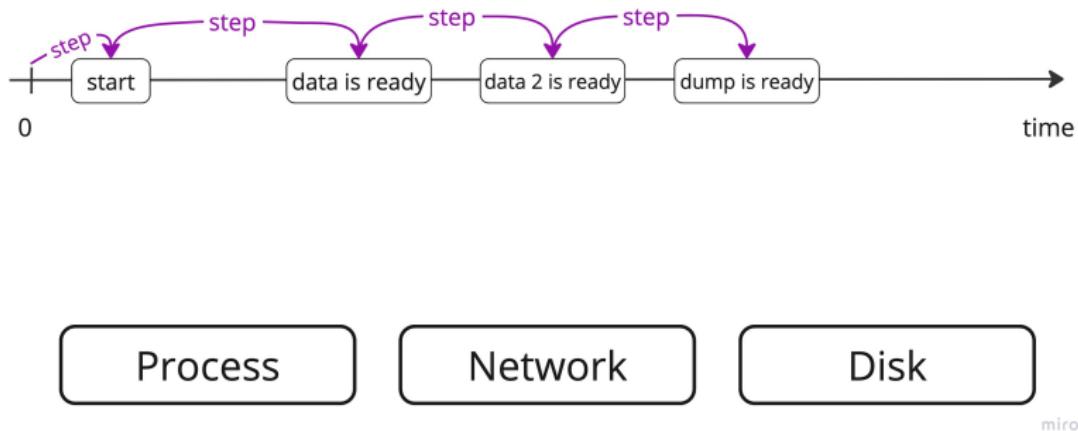


Исполнение симуляции



# Введение

## Дискретно-событийное моделирование

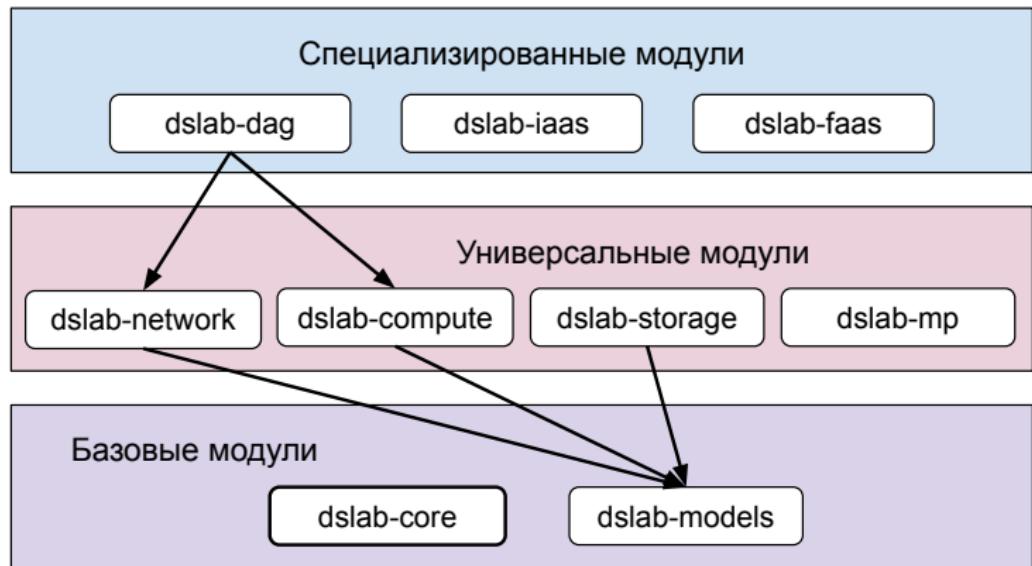


Исполнение симуляции



# Введение

## Архитектура проекта DSLab





# Введение

Callback модель. Фрагментированная реализация Process.

```
fn on(&mut self, event: Event) {  
    cast!(match event.data {  
        Start {} => {  
            self.on_start();  
        }  
        DownloadCompleted {data} => {  
            self.on_download_completed(data);  
        }  
        DiskWriteCompleted => {  
            self.on_disk_write_completed();  
        }  
    })  
}
```

Реагирование на события.

```
impl Process {  
    fn on... {  
          
    }  
    fn on... {  
          
    }  
    fn on... {  
          
    }  
}
```

Реализация  
callback-ов



# Введение

## Преимущества асинхронного подхода

```
async fn add_file_to_storage(some_file) {
    send_file_to_all_replicas(some_file);
    result = wait_confirmation_from_all().await;

    if result.has_quorum {
        send_commit_to_replicas(result.nodes);
        wait_commit_confirmation_from(result.nodes).await;

        send_ok_message_to_user();
    } else {
        send_reject_message_to_user();
    }
}
```

Псевдокод асинхронного взаимодействия нод в симуляции



# Введение

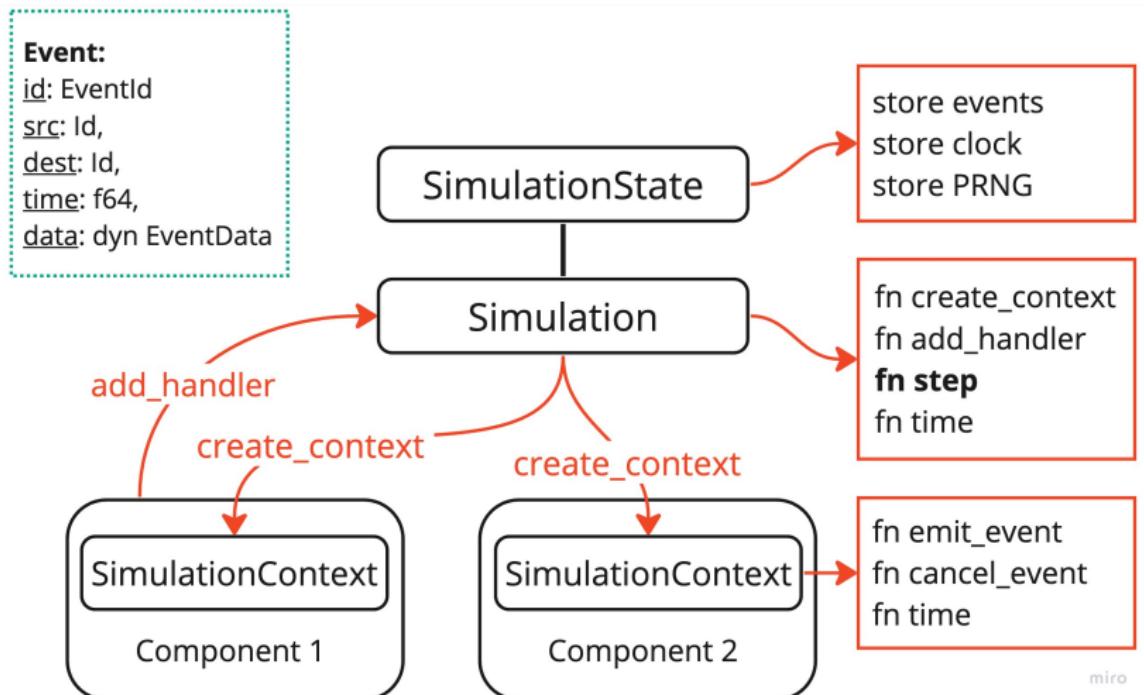
## Постановка задачи

Цель проекта – реализация поддержки асинхронного программирования для фреймворка DSLab. Для этого необходимо:

- Реализовать эффективное асинхронное расширение для существующего ядра `dslab-core`. Поддержать возможности:
  - запустить асинхронную задачу;
  - «заснуть» на определенное время внутри симуляции;
  - асинхронно ждать событий от других компонент;
  - комбинировать ожидания с помощью методов `futures`.
- Добавить примеры, написать документацию и тесты.

# Введение

## Дизайн и структура ядра dslab-core



miro

Внутреннее устройство dslab-core



# Управление исполнением

## Stackless-корутины

```
fn on_start_action() {  
    // do 1  
}  
  
fn on_first_event() {  
    // do 2  
}  
  
fn on_second_event() {  
    // do 3  
}  
  
/* ... */  
  
on_start_action();  
  
async fn action() {  
    // do 1  
  
    wait_for_first_event().await;  
  
    // do 2  
  
    ⇌  
    wait_for_second_event().await;  
  
    // do 3  
}  
  
/* ... */  
  
let future = action();  
future.poll(**/);
```

Синхронный код (разрезан на части разработчиком)

Асинхронный код (разрезан на части компилятором)



# Новое API для пользователей

## Расширение SimulationContext

- Запустить асинхронную задачу:

```
> fn spawn(&self, future: impl Future<Output = ()>)
```

- «Заснуть» на определенное время:

```
> async fn async_wait_for(&self, timeout: f64)
```

- Дождаться события с «данными» типа T от компонента src:

```
> async fn async_handle_event<T>(src: Id) -> (Event, T)
```

```
> async fn async_wait_for_event<T>(src: Id, timeout: f64) ->  
    AwaitResult<T>
```

- Дождаться события с «данными» T, деталями details от src:

```
> async fn async_detailed_handle_event<T>(src: Id, details: u64)  
    -> (Event, T)
```

```
> async fn async_detailed_wait_for_event<T>(src: Id, details:  
    u64, timeout: f64) -> AwaitResult<T>
```



# Новое API для пользователей

## Детализированное ожидание. Пример

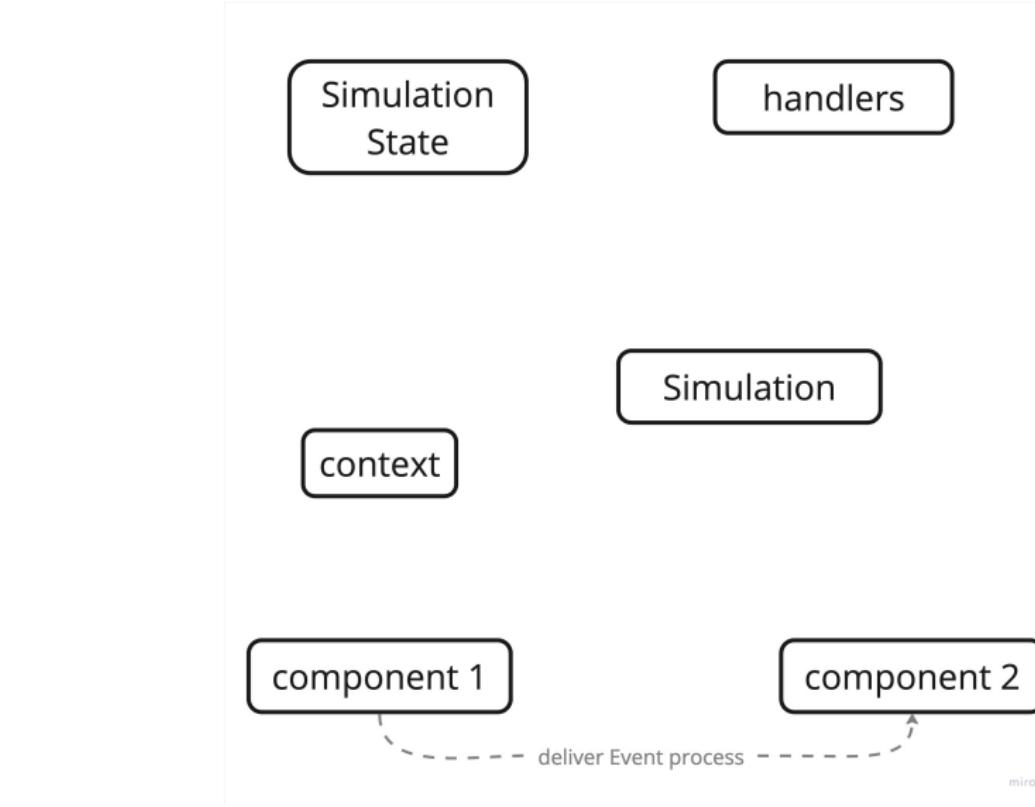
```
async fn process_data_transfer_request(
    &self, request: Request) {
    let request_id = self.network.send(request);

    self.ctx.
        async_detailed_handle_event::<DataTransferCompleted>(
            self.network.id,
            request_id,
        ).await;
}
```

Использование детализированного асинхронного ожидания

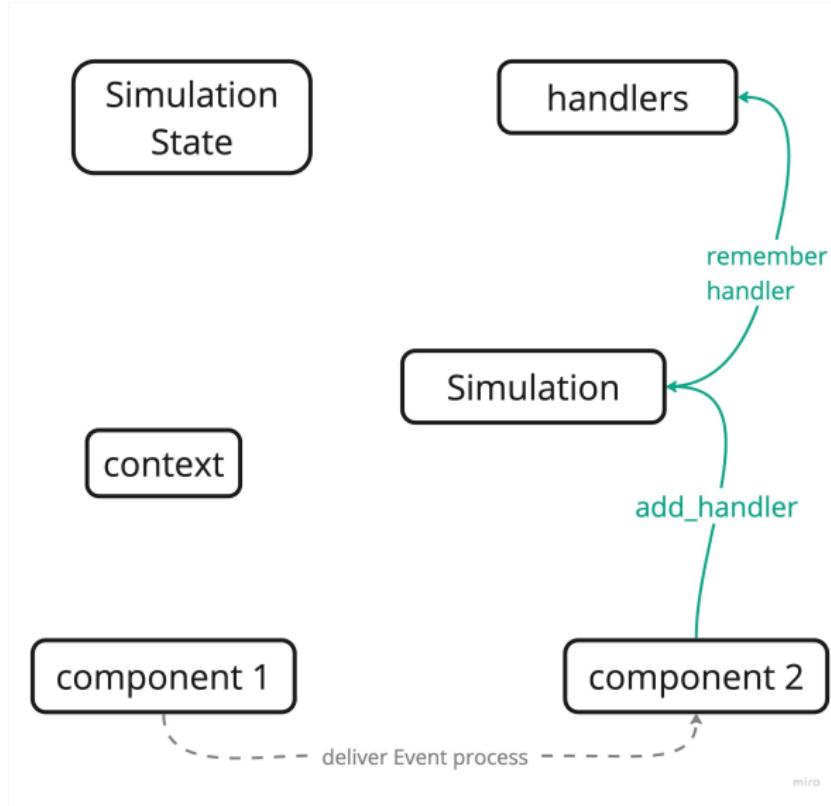
# Внутреннее устройство и реализация

## Callback-based модель



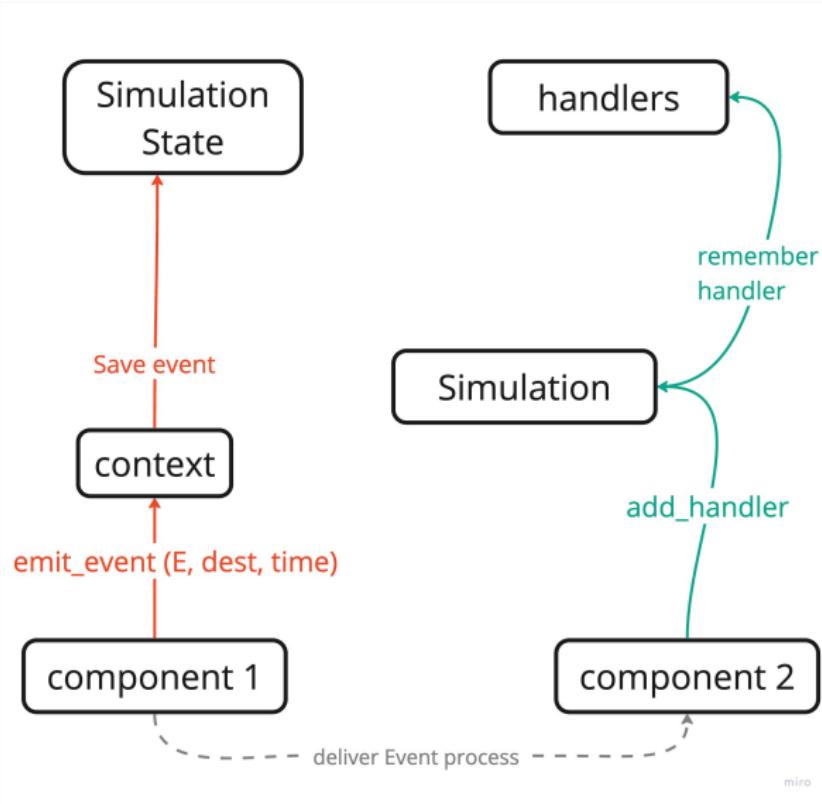
# Внутреннее устройство и реализация

## Callback-based модель



# Внутреннее устройство и реализация

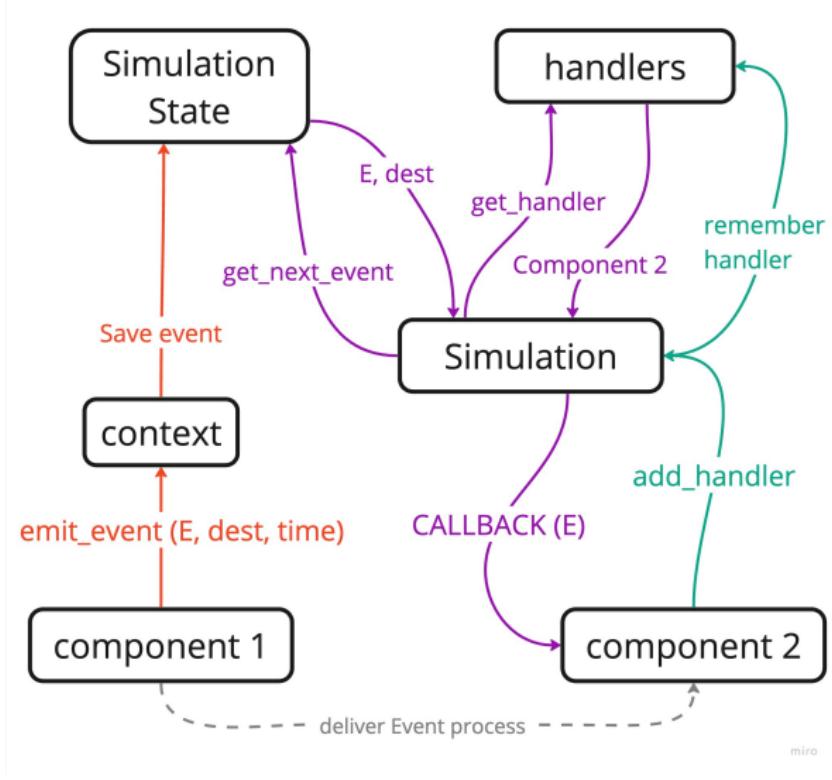
## Callback-based модель



miro

# Внутреннее устройство и реализация

## Callback-based модель

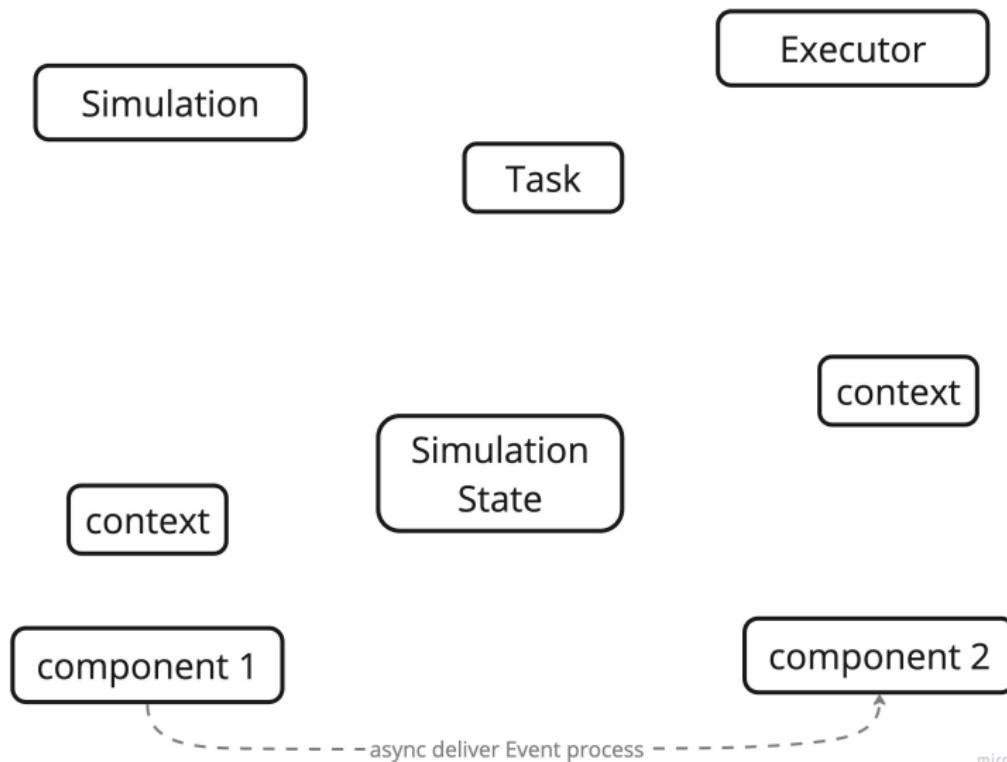


miro



# Внутреннее устройство и реализация

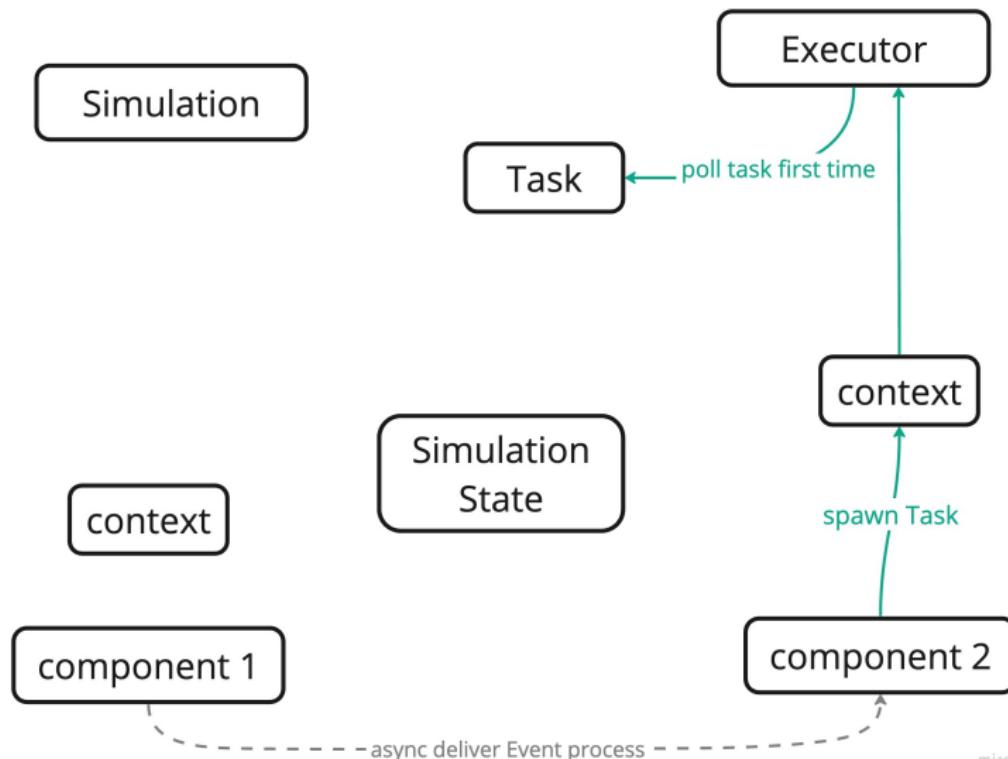
## Асинхронная модель



miro

# Внутреннее устройство и реализация

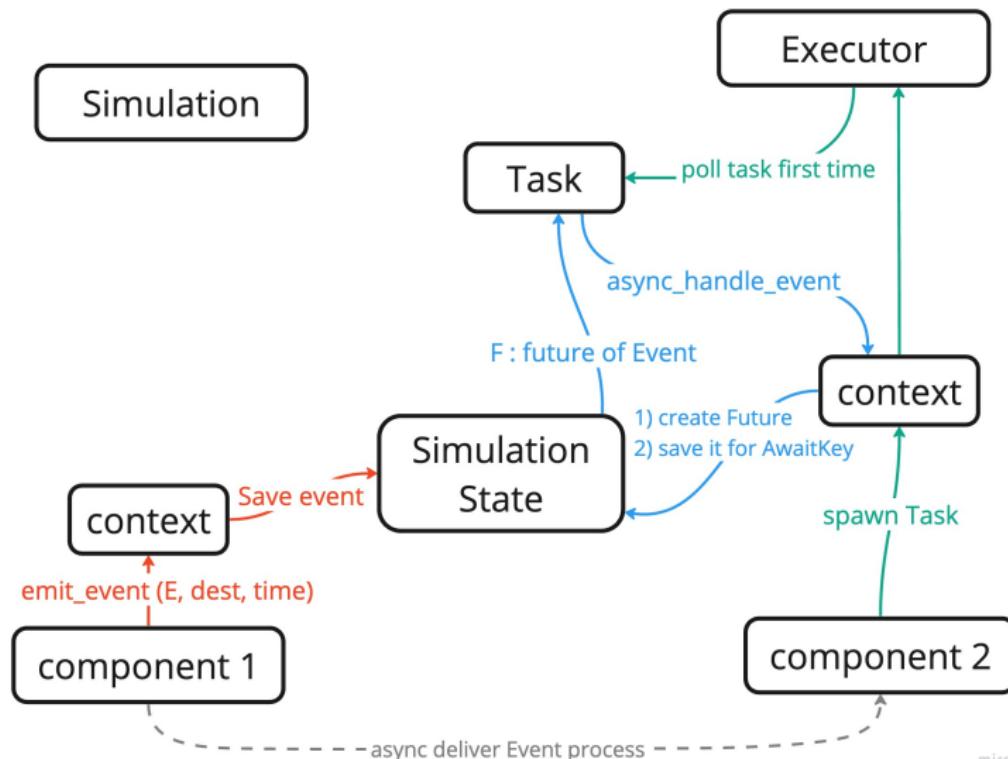
## Асинхронная модель



miro

# Внутреннее устройство и реализация

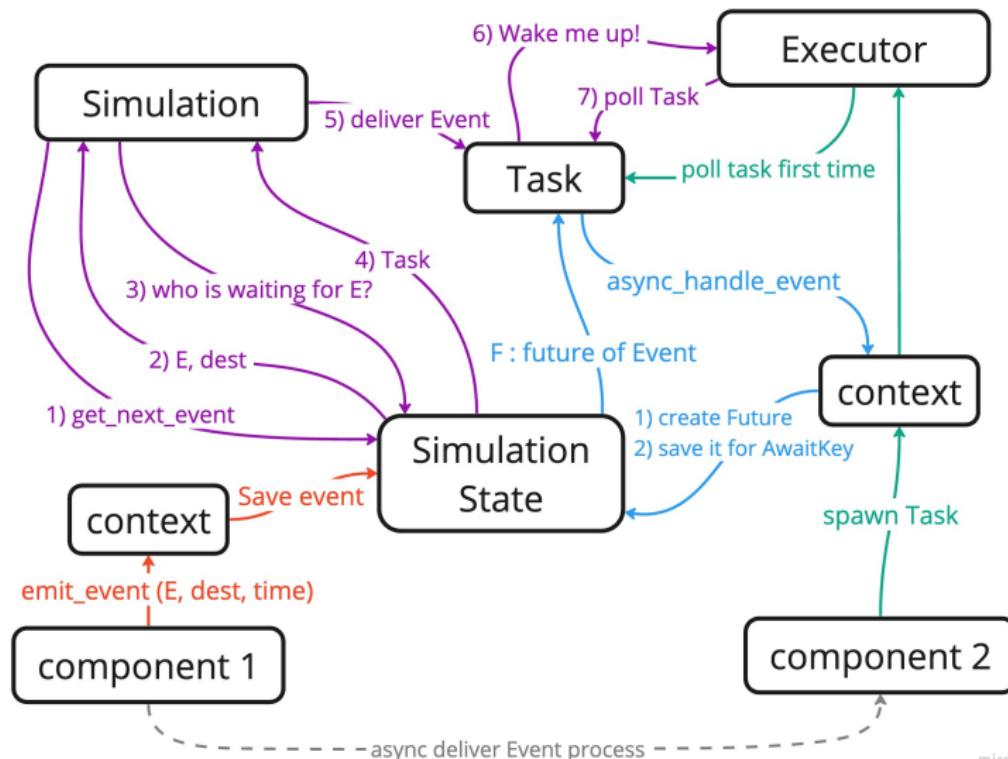
## Асинхронная модель



miro

# Внутреннее устройство и реализация

## Асинхронная модель

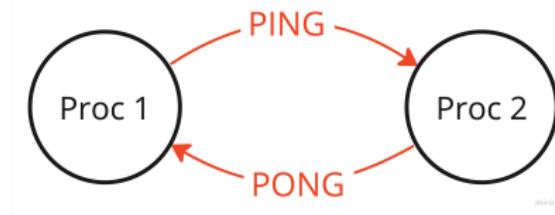


# Тестирование

## Ping-pong. Производительность

Hosts	100000
Peers per host	100
Iterations	100

Аргументы

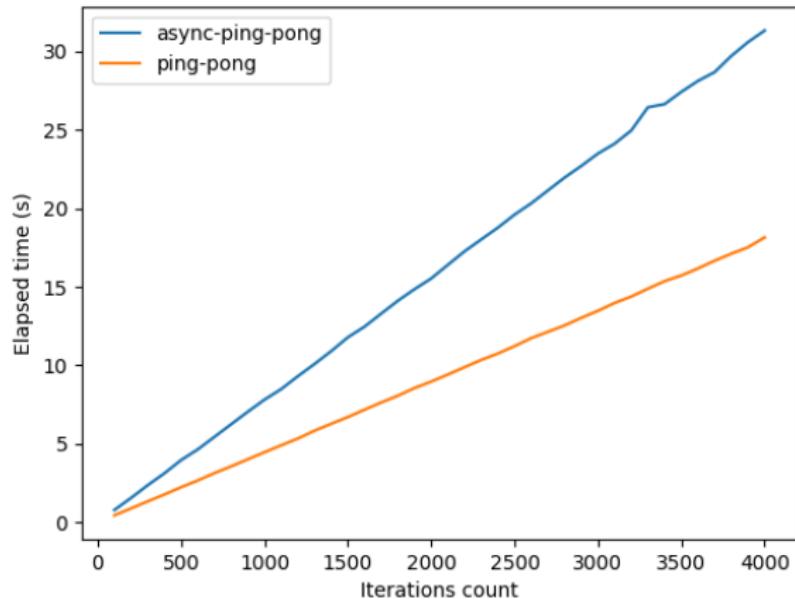


Example	Elapsed time	Events/s	Iterations/s
async-ping-pong	16.45s	1234103	6.08
ping-pong	8.20s	2452670	12.20

Сравнение производительности

# Тестирование

## Ping-pong. Производительность



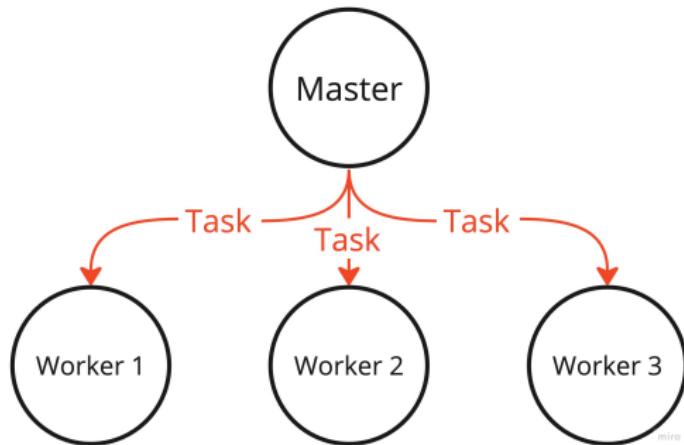
Сравнение производительности

# Тестирование

Master-workers. Производительность

Hosts	100
Tasks	100000

Аргументы



Example	Elapsed time	Scheduling time	Events/s
async-master-workers	5.97s	4.31s	285059
master-workers	5.19s	4.31s	328392

Сравнение производительности



# Тестирование

## Master-workers. Сравнение кода компонента Worker

```
pub struct Worker {  
    tasks: HashMap<u64, TaskInfo>,  
    computations: HashMap<u64, u64>,  
    reads: HashMap<u64, u64>,  
    writes: HashMap<u64, u64>,  
    downloads: HashMap<usize, u64>,  
    uploads: HashMap<usize, u64>,  
}  
  
fn on_task_request(request);  
fn on_data_read_completed(r_id);  
fn on_comp_started(comp_id);  
fn on_comp_finished(comp_id);  
fn on_data_write_completed(r_id);  
  
fn on_data_transfer_completed(*);
```

master-workers

```
fn on_task_request(req) {  
    self.ctx.spawn(  
        self.process_task(req));  
}  
  
async fn process_task(&self, req:  
    TaskRequest) {  
    let mut task = TaskInfo {req};  
  
    self.download_data(&task).await;  
    self.read_data(&task).await;  
    self.run_task(&task).await;  
    self.write_data(&task).await;  
    self.upload_result(&task).await;  
}
```

async-master-workers



- ✓ Реализовать эффективное асинхронное расширение для существующего ядра `dslab-core`
- ✓ Добавить примеры использования нового функционала высокоуровневыми компонентами.
- ✓ Написать документацию нового API и покрыть реализацию тестами.



- ✓ Реализовать эффективное асинхронное расширение для существующего ядра `dslab-core`
  - ✓ Добавить примеры использования нового функционала высокоуровневыми компонентами.
  - ✓ Написать документацию нового API и покрыть реализацию тестами.
- 
- **Stateful**-процесс  $\implies$  используем асинхронный подход.
  - **Stateless**-процесс  $\implies$  используем модель callback-ов.

# Вопросы