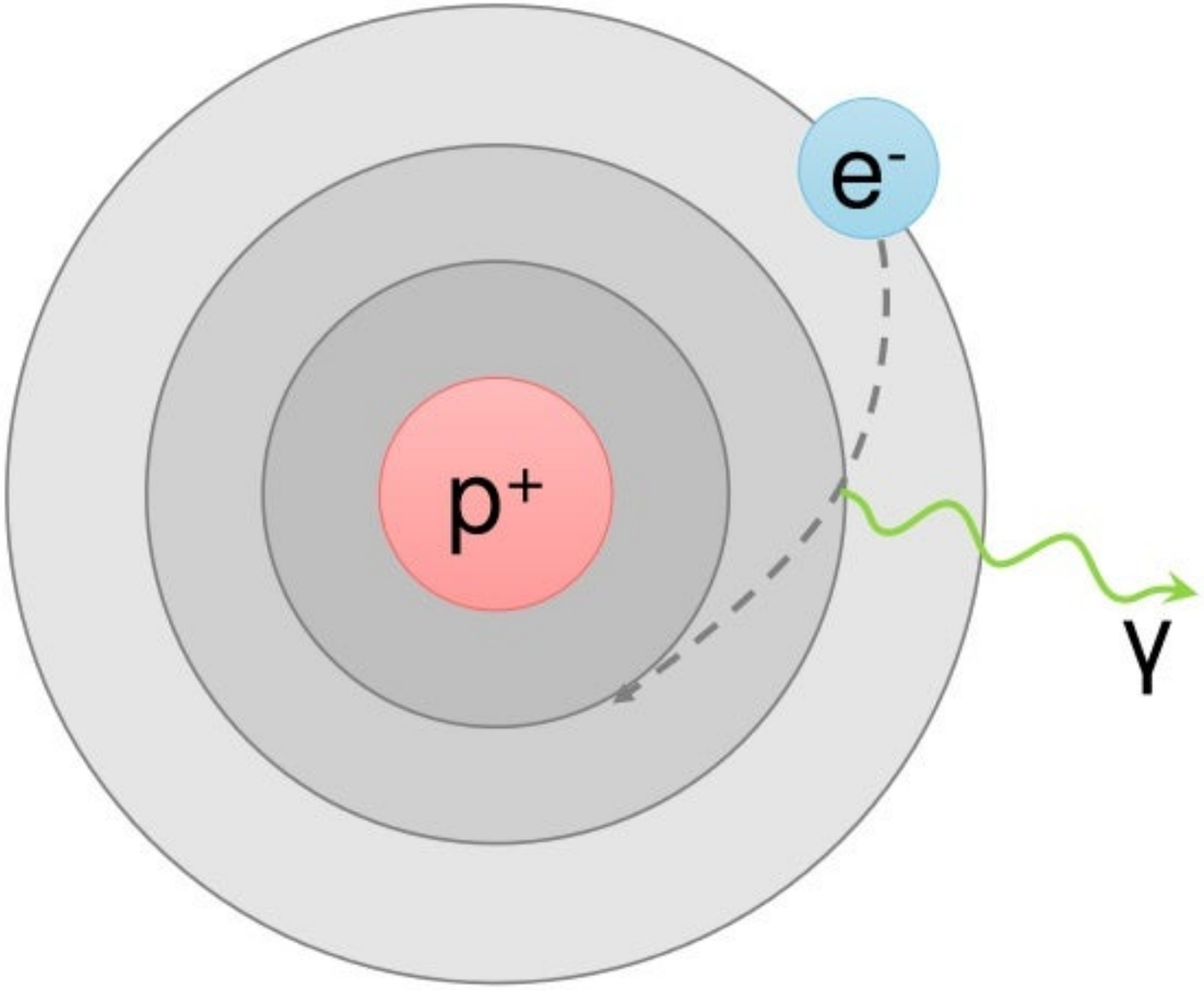

Яндекс

Яндекс

Субъекторная модель

Григорий Демченко. Разработчик YT.


e-
p+


e-
p+
γ

Системы

Сложные системы не описываются совокупностью частей.

ООП

> Когда я говорил про ООП,
> то я не имел в виду C++.

Алан Кей

> Я жалею, что придумал термин «объекты» много лет назад, потому что он заставляет людей концентрироваться на мелких идеях.

> По-настоящему большая идея — это сообщения.

Алан Кей

Reader

Рассмотрим класс Reader:

```
class Reader {
public:
    Buffer read(Range range, const Options& options);

    // и другие методы ...
};
```

Превращение

В отдельностоящую функцию:

```
Buffer read(Reader* this,
            Range range,
            const Options& options);
```

Reader

Вернемся к классу:

```
class Reader {
public:
    Buffer read(Range range, const Options& options);

    // и другие методы ...
};
```

Использование

Вызов метода класса:

```
auto buffer = reader.read(range, options);
```

Использование

Трансформация вызова:

```
reader
    <- read(range, options)
    -> buffer;
```

Сообщения

```
struct InReadMessage {
    Range range;
    Options options;
};

struct OutReadMessage {
    Buffer buffer;
};

reader
    <- InReadMessage{range, options}
    -> OutReadMessage;
```

Сообщения

Создание

Упаковка вызова метода в сообщение:

```
template<typename T_base>
struct ReaderAdapter : T_base {
    Buffer read(Range range, const Options& options) {
        return T_base::call([range, options](Reader& reader) {
            return reader.read(range, options);
        });
    }
};
```

Тождественное преобразование

```
template<typename T>
struct BaseValue {
protected:
    template<typename F>
    auto call(F&& f) {
        return f(t);
    }
private:
    T t;
};

ReaderAdapter<BaseValue<Reader>> reader;  // Reader reader;
auto buffer = reader.read(range, options);
```

Синхронизация

```
template<typename T_base, typename T_locker>
struct BaseLocker : private T_base {
protected:
    template<typename F>
    auto call(F&& f) {
        std::unique_lock<T_locker> _{lock_};
        return f(static_cast<T_base&>(*this));
    }

private:
    T_locker lock_;
};
```

Использование

В качестве объекта синхронизации:

```
ReaderAdapter<BaseLocker<Reader, std::mutex>> reader;

auto buffer = reader.read(range, options);
```

Универсальный адаптер

Обобщение подхода:

```
DECL_ADAPTER(Reader, read)


AdaptedLocked<Reader, std::mutex> reader;
auto buffer = reader.read(range, options);
```

Сопрограммы

Перепланирование

```
struct Spinlock {
    void lock() {
        while (lock_.test_and_set(std::memory_order_acquire)) {
            reschedule();
        }
    }

    void unlock() {
        lock_.clear(std::memory_order_release);
    }

private:
    std::atomic_flag lock_ = ATOMIC_FLAG_INIT;
};
```

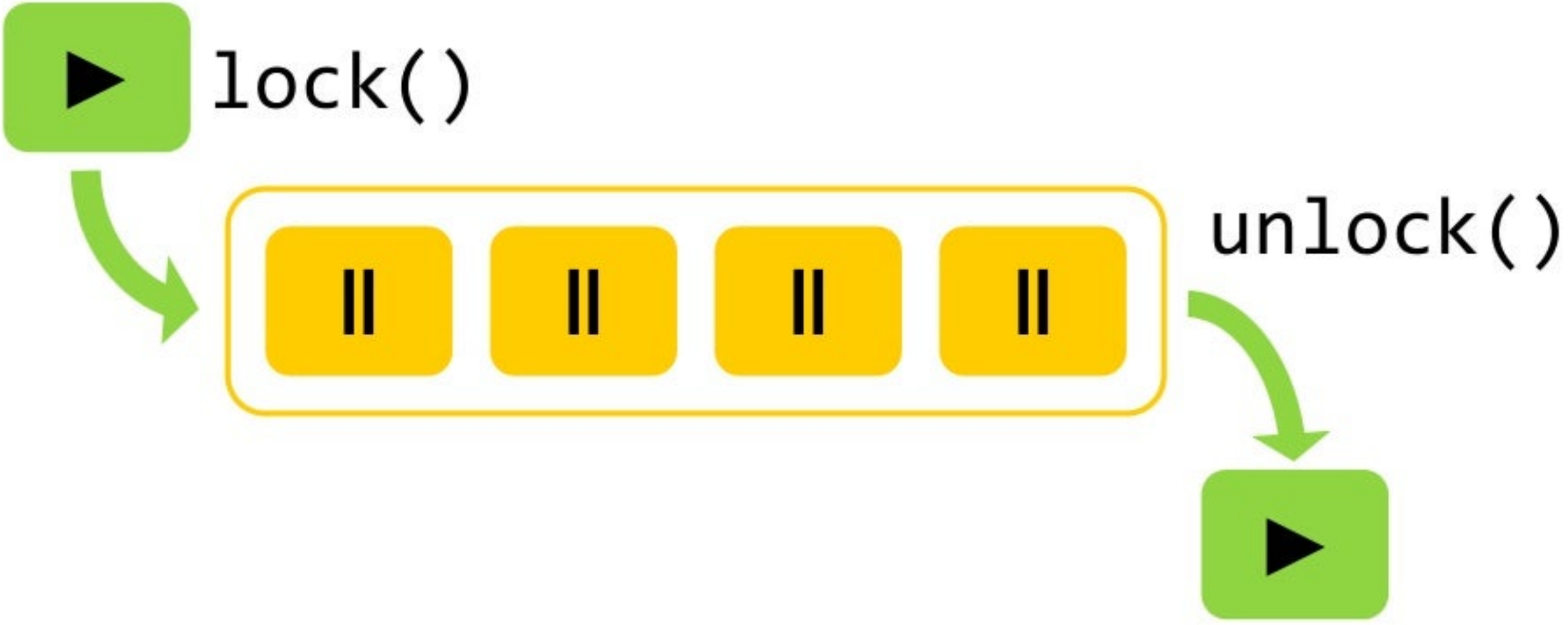
CoSpinLock

Используем как объект синхронизации:

```
template <typename T>
using CoSpinlock = AdaptedLocked<T, synca::Spinlock>;

CoSpinlock<Reader> reader;
auto buffer = reader.read(range, options);
```

Асинхронный мьютекс



CoMutex

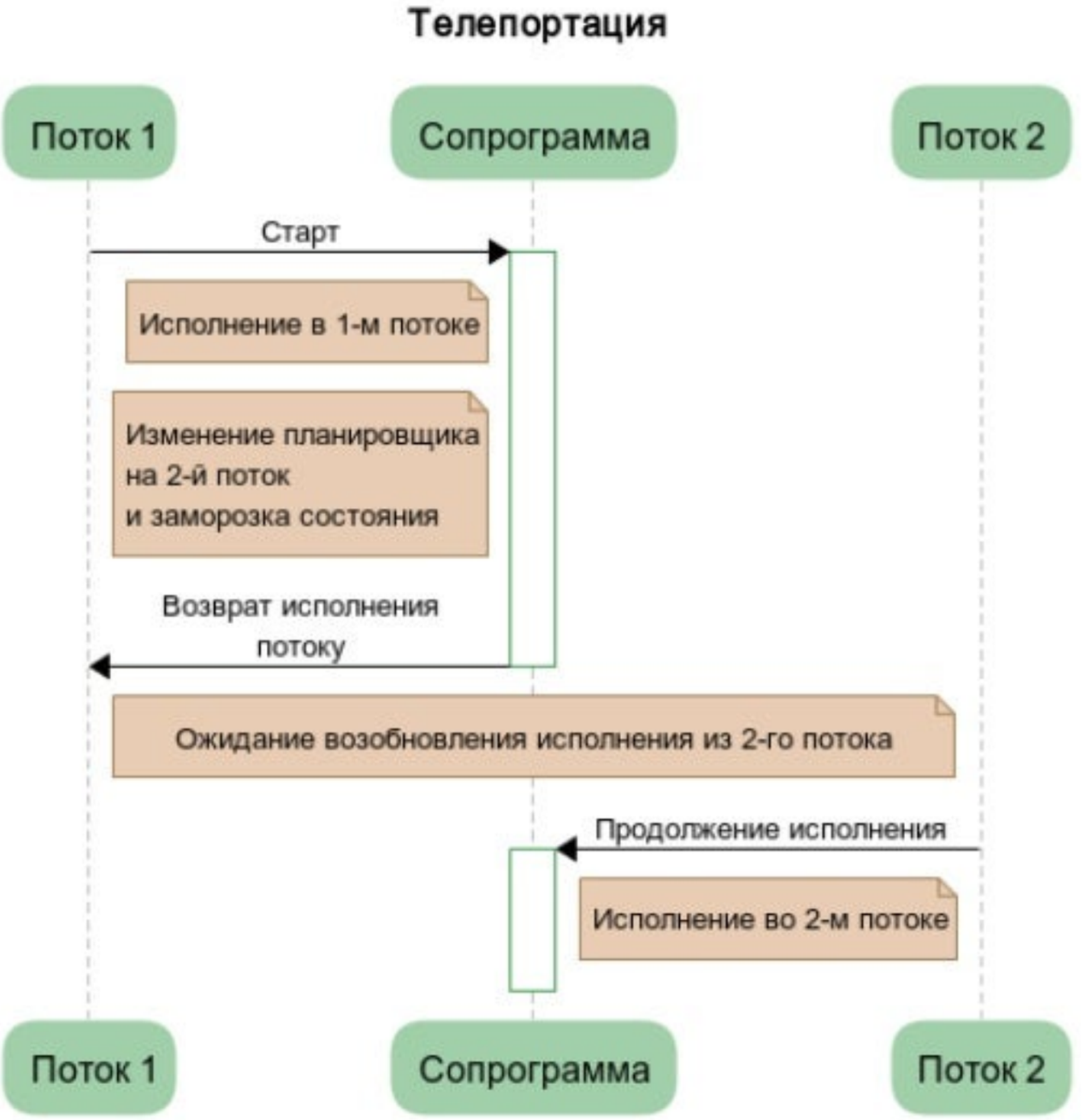
Синхронизация через мьютекс с FIFO гарантией:

```
CoMutex<Reader> reader;

auto buffer = reader.read(range, options);
```

Телепортация

- 2 планировщика
- Сопрограмма



Телепортация через портал

```
template <typename T_base>
struct BaseSerializedPortal : T_base {
    BaseSerializedPortal() : tp_(1) {}

protected:
    template <typename F>
    auto call(F&& f) {
        synca::Portal _{tp_};
        return f(static_cast<T_base&>(*this));
    }

private:
    mt::ThreadPool tp_;
};
```

CoSerializedPortal

Синхронизация через портал:

```
CoSerializedPortal<Reader> reader;

auto buffer = reader.read(range, options);
```

Alone

```
template <typename T_base>
struct BaseAlone : T_base {
    BaseAlone(mt::IScheduler& scheduler) : alone_{scheduler} {}

protected:
    template <typename F>
    auto call(F&& f) {
        synca::Portal _{alone_};
        return f(static_cast<T_base&>(*this));
    }

private:
    synca::Alone alone_;
};
```

CoAlone

Синхронизация через `synca::Alone`, который гарантирует, что ни один обработчик не будет запущен параллельно с другим:

```
CoAlone<Reader> reader;

auto buffer = reader.read(range, options);
```

Канал

```
template <typename T_base>
struct BaseChannel : T_base {
    BaseChannel() {
        synca::go([this] { loop(); });
    }

private:
    void loop() {
        for (auto&& action : channel_) { action(); }
    }

    synca::Channel<Handler> channel_;
};
```

Channel: реализация вызова

```
template <typename F>
auto call(F&& f) {
    Result<decltype(f())> result;
    synca::DetachableDoer doer;
    channel_.put([&] {
        try {
            result = f(static_cast<T_base&>(*this));
        } catch (std::exception&) {
            result.setCurrentError();
        }
        doer.done();
    });
    doer.wait();
    return result.get();
}
```

CoChannel

Синхронизация через канал `synca::Channel`:

```
CoChannel<Reader> reader;

auto buffer = reader.read(range, options);
```

Способы синхронизации в пользовательском пространстве

1. `CoSpinlock`
2. `CoMutex`
3. `CoSerializedPortal`
4. `CoAlone`
5. `CoChannel`

Субъектор

Определение

```
#define BIND_SUBJECTOR(D_type, D_subjector, ...)                \
    template <>                                                 \
    struct subjector::SubjectorPolicy<D_type> {                 \
        using Type = D_subjector<D_type, ##__VA_ARGS__>;        \
    };

template <typename T>
struct SubjectorPolicy {
    using Type = CoMutex<T>;
};

template <typename T>
using Subjector = typename SubjectorPolicy<T>::Type;
```

Использование

Можно использовать любой из 5-ти способов синхронизации:

```
DECL_ADAPTER(Reader, read, open, close)
BIND_SUBJECTOR(Reader, CoAlone)


Subjector<Reader> reader;
auto buffer = reader.read(range, options);
```

Асинхронный вызов

```
// declaration part
struct Network {
    void send(const Packet& packet);
};
DECL_ADAPTER(Network, send)
BIND_SUBJECTOR(Network, CoChannel)

// usage part
void sendPacket(const Packet& packet) {
    Subjector<Network> network;
    network.async().send(packet);     // асинхронный запуск
    // ...
    network.wait();                   // ожидание завершения
}
```

> CoChannel + async() ≡ акторы

Субъекторная модель

1. Глубокая абстракция исполнения.
2. Интеграция синхронизации.
3. Неинвазивность.
4. Защищенность от ошибок многопоточного проектирования.
5. Поздняя оптимизация.
6. Эффективность.
7. Ясность, чистота и простота кода.

Модель упрощается при обобщении

Григорий Демченко

Спасибо за внимание!

Григорий Демченко

Разработчик YT

 gridem@yandex-team.ru

 github.com/gridem/Subjector

 github.com/gridem/Synca

 habrahabr.ru/users/gridem

 gridem.blogspot.com