async/await: собираем грабли

Андрей Часовских

Broadridge Financial Solutions

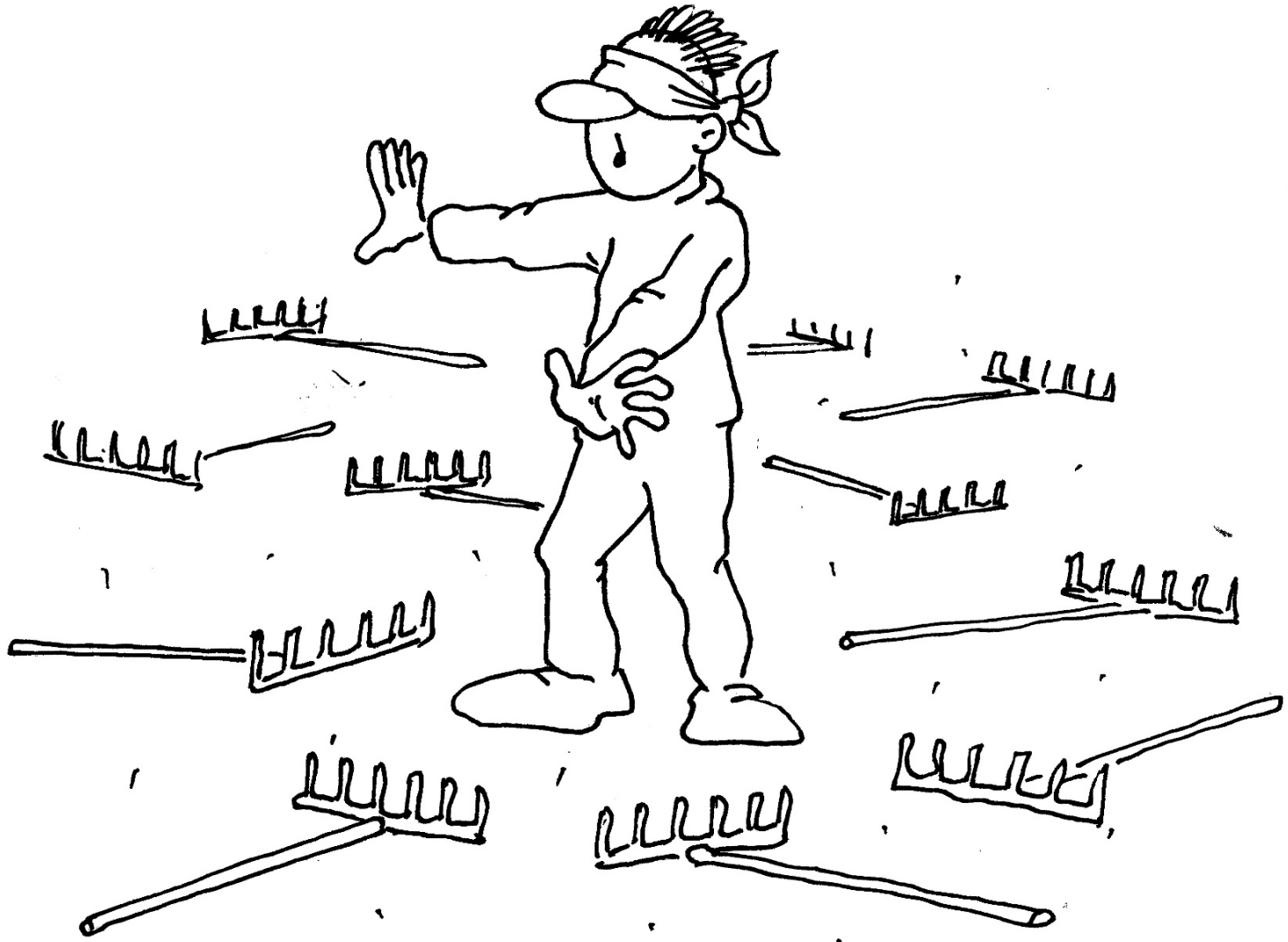
.NEXT 2014 Moscow

Асинхронное программирование

- Управление получаем сразу после вызова операции
- Об окончании операции нас уведомляет коллбэк
- Преимущества
  - Отзывчивость
  - Масштабируемость

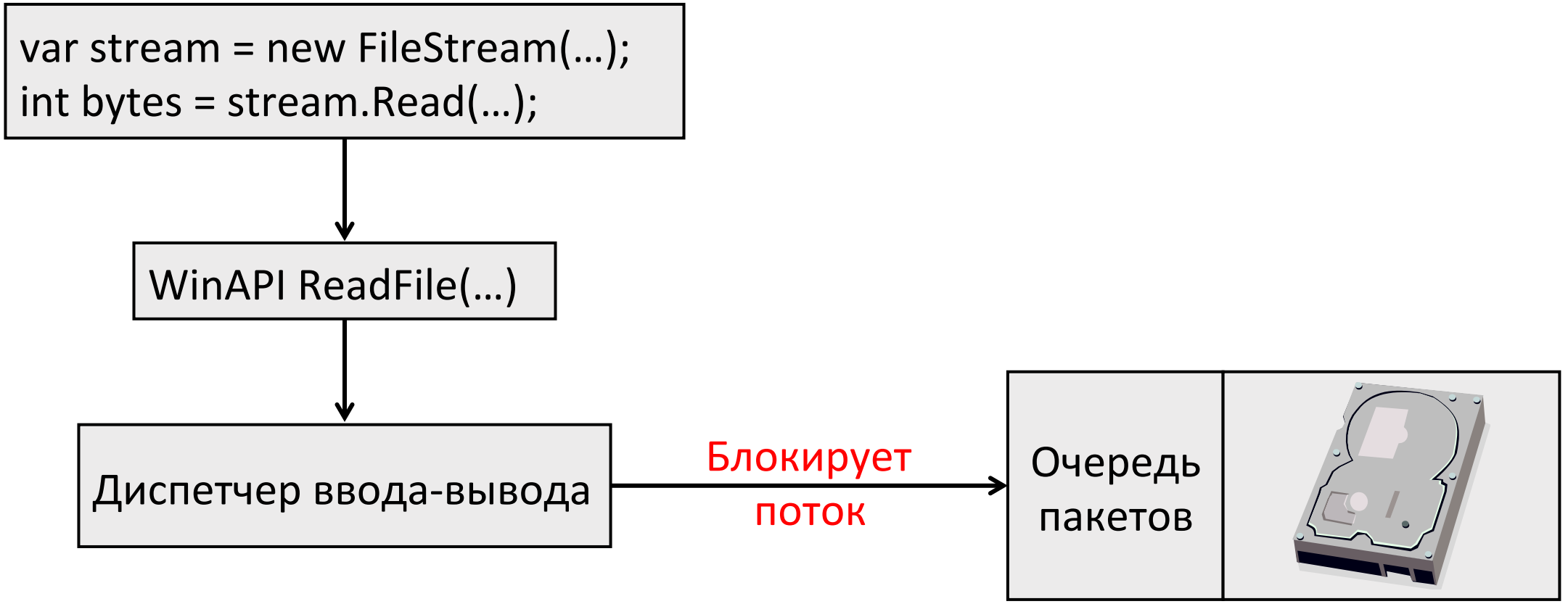
Асинхронное программирование в .NET

- Asynchronous Programming Model (BeginXxx/EndXxx)
- Event-based Asynchronous Programming (XxxAsync(), XxxCompleted)
- Task-based Asynchronous Programming (async/await, Task/Task<T>)
  - Ссылка на документ в конце презентации

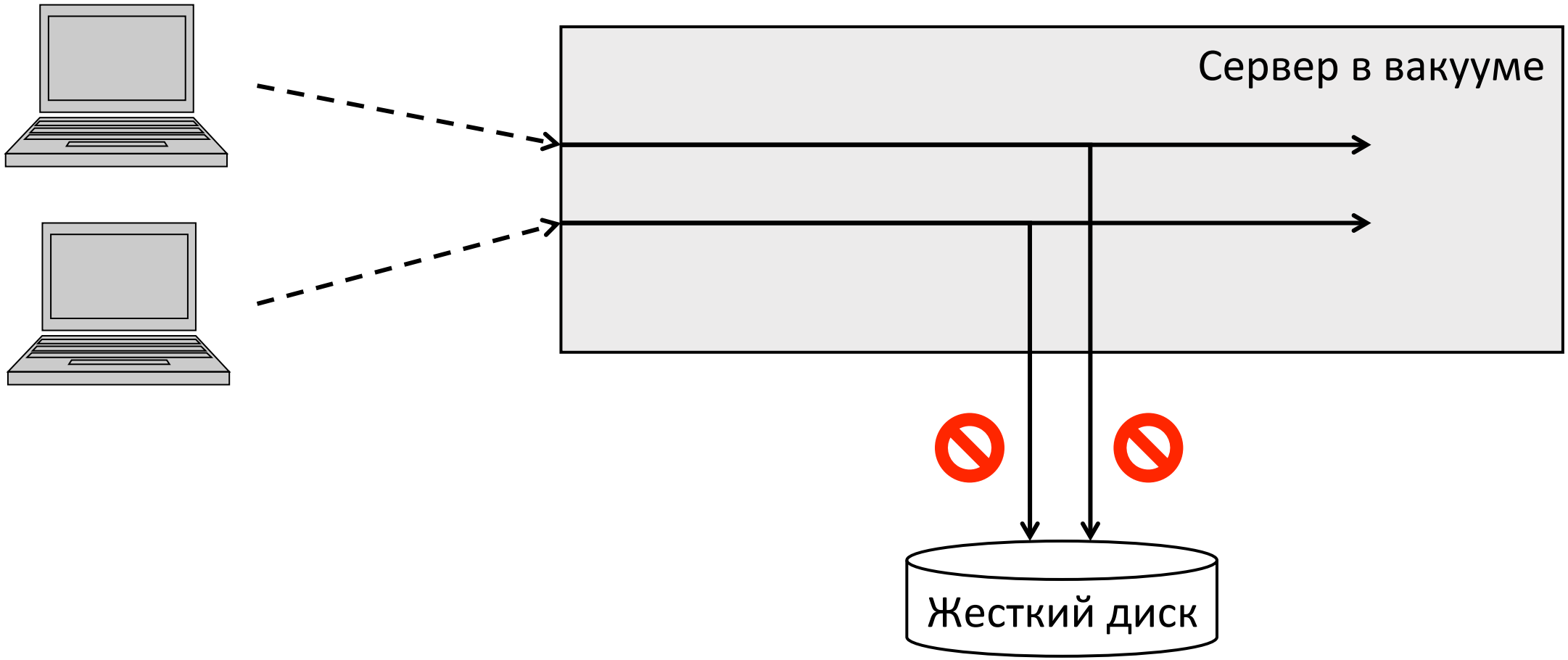


Асинхронность != многопоточность

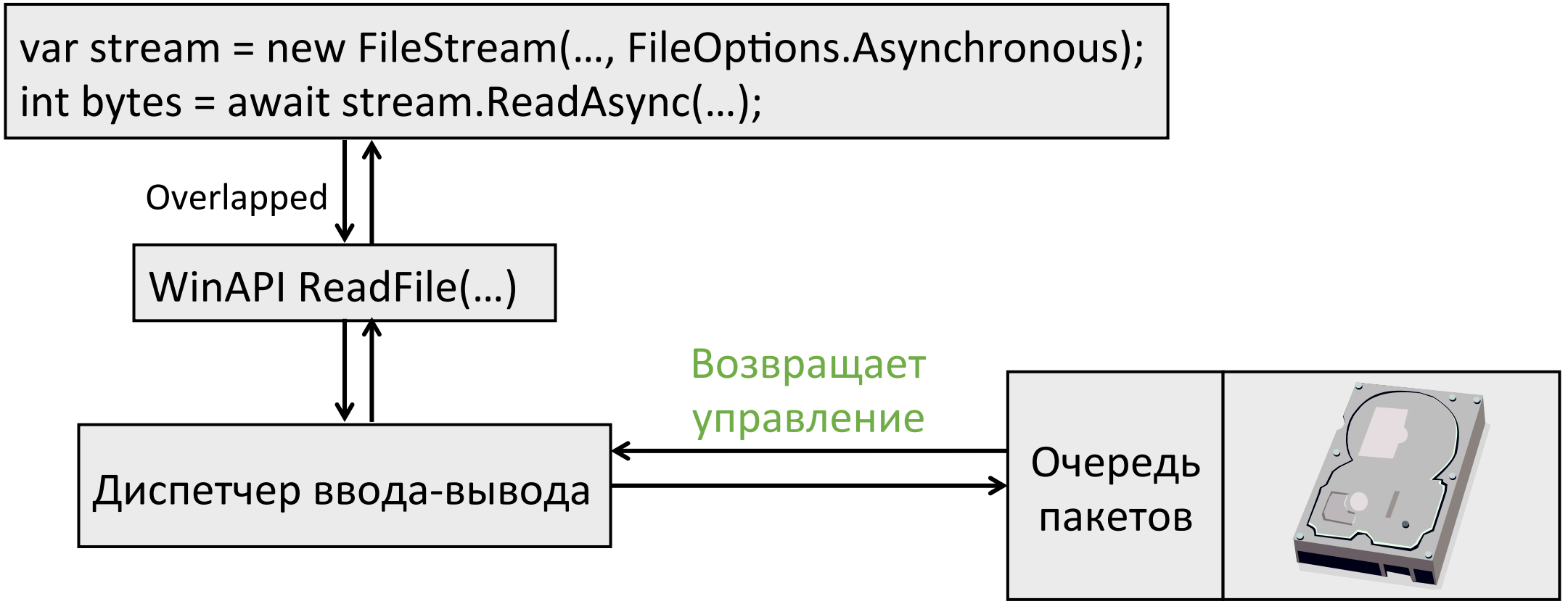
Синхронный ввод-вывод



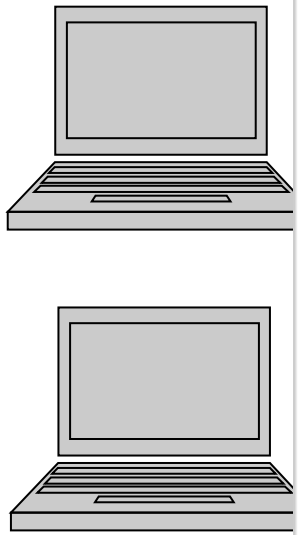
Синхронный ввод-вывод: пример

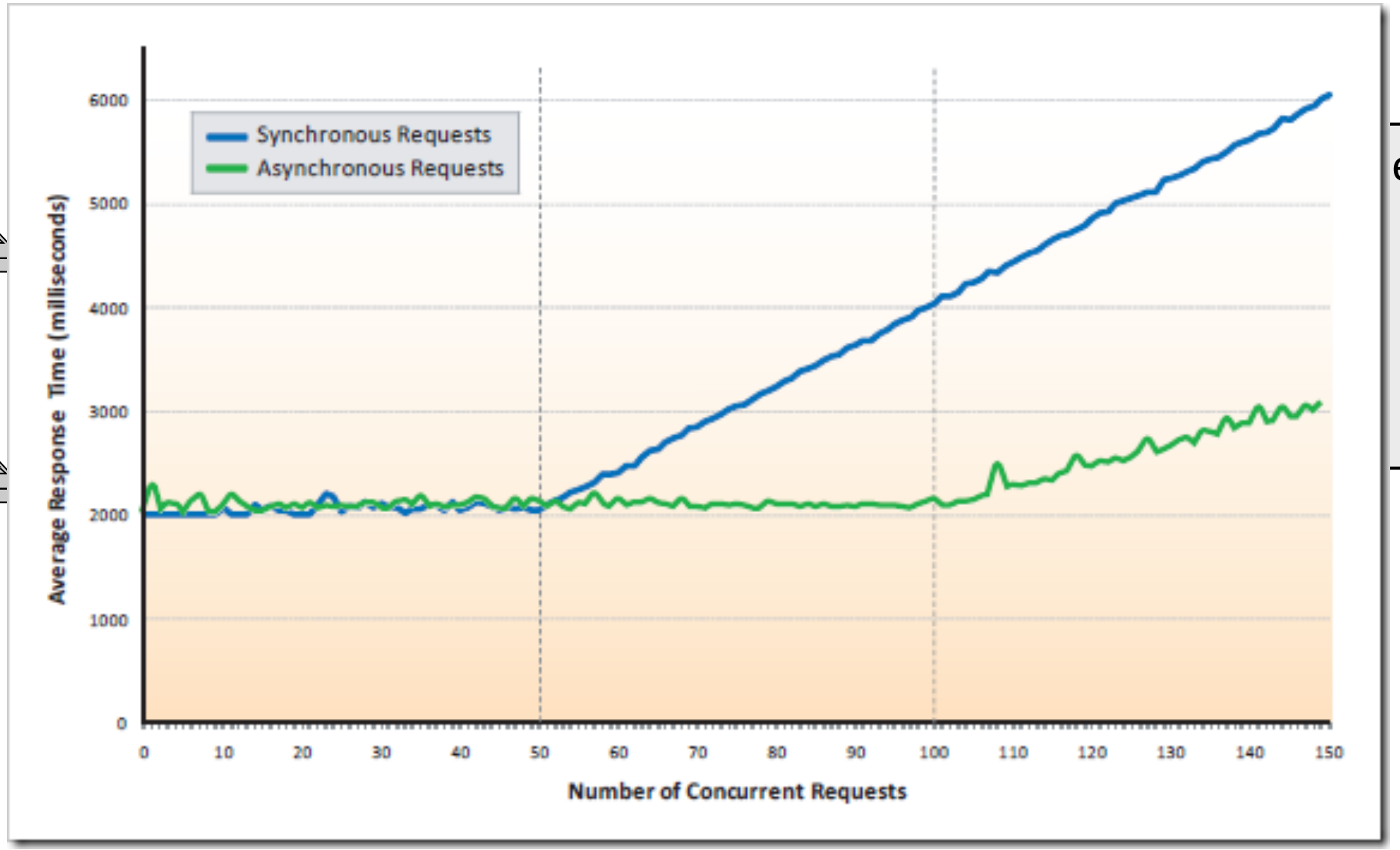


Асинхронный ввод-вывод



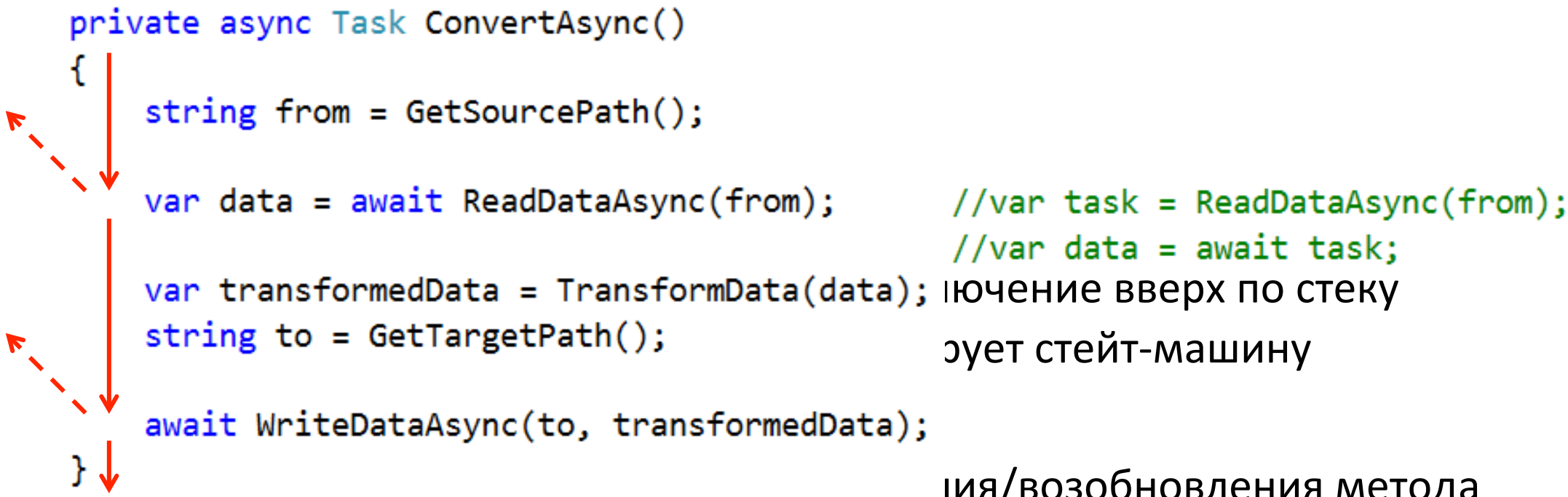
Асинхронный ввод-вывод: пример





e

async/await



ючение вверх по стеку

ует стейт-машину

ия/возобновления метода

- Извлекает результат или исключение из таска

Реализация

```
async Task FooAsync()
{
    M1();
    await BarAsync();
    M2();
}

// примерный сгенерированный код
Task FooAsync()
{
    var sm = new FooAsyncStruct();
    sm.this = this;
    sm.builder = AsyncTaskMethodBuil
    sm.state = -1;
    …
    sm.MoveNext();
    return sm.builder.Task;
}
```

```
struct FooAsyncStruct : IAsyncStateMachine {
    public AsyncTaskMethodBuilder builder;
    public SomeType this;
    public int state;
    …

    private void MoveNext() {
        try {
            switch (state) {
                …
            }
        }
        catch (Exception ex) {
            builder.SetException(ex);
            return;
        }
        builder.SetResult();
    }
}
```

```
async Task FooAsync()
{
    M1();
    await BarAsync();
    M2();
}
```

```
// примерный код того, что происходит внутри каждого состояния
M1();

var task = BarAsync();
var awaiter = task.GetAwaiter();

Action postback = () => {
    awaiter.GetResult();
    M2();
}

if (awaiter.IsCompleted) {
    postback();
}
else {
    var context = SynchronizationContext.Current;
    if (context == null) {
        context = new SynchronizationContext();
    }
    var contextCopy = context.CreateCopy();
    awaiter.OnCompleted(() => contextCopy.Post(postback, null));
}
```

Синхронное выполнение

Захват контекста

Контекст пула потоков

Заблуждения

- async методы выполняются в другом потоке
- С помощью await метод выполняется в другом потоке
  - await вообще не запускает методы
- Продолжение метода выполняется в другом потоке
  - Не всегда: ожидаемый объект уже завершен или однопоточный контекст не был захвачен

«async/await **позволяет использовать модель синхронного программирования** для написания кода, выполняющего операции ввода-вывода без блокировки потоков.
И т.о. создавать более отзывчивое и масштабируемое программное обеспечение.»

Джеффри Рихтер

async void

- Исключения выбрасываются в вызывающий контекст
  - Могут завершить процесс
- Не узнаем об окончании операции
- Только для обработчиков событий и подобных штук

Асинхронные лямбды

- Это Action или Func<Task>?
- Action == async void
- Проверять тип делегата!

```
Task t = null;
SomeMethod(() => t = FooAsync());
await t;
```

Таск, которого не ждут

- Не узнаем об исключениях или узнаем слишком поздно
  - Могут завершить процесс
  - TaskScheduler.UnobservedTaskException
- Не узнаем об окончании операции

Блокировка таска

- Блокируем текущий поток
- Возможен дедлок

Используйте Task.ConfigureAwait(false)

- Не захватываем текущий контекст
  - Минимизируем переключения между потоками
- Для библиотечных вызовов
- Захват контекста – для вызовов верхнего уровня

Async all the way

- Нежелательно смешивать синхронный и асинхронный код
- async/await стремится распространяться по коду
- Избавляемся от простых ошибок

Спасибо!

- Pfx Team http://blogs.msdn.com/b/pfxteam/
- Stephen Cleary http://blog.stephencleary.com/
- Lucian Wischik http://blogs.msdn.com/lucian
- TAP http://aka.ms/tap

http://andreycha.info

http://github.com/andreycha/DotNext2014Moscow