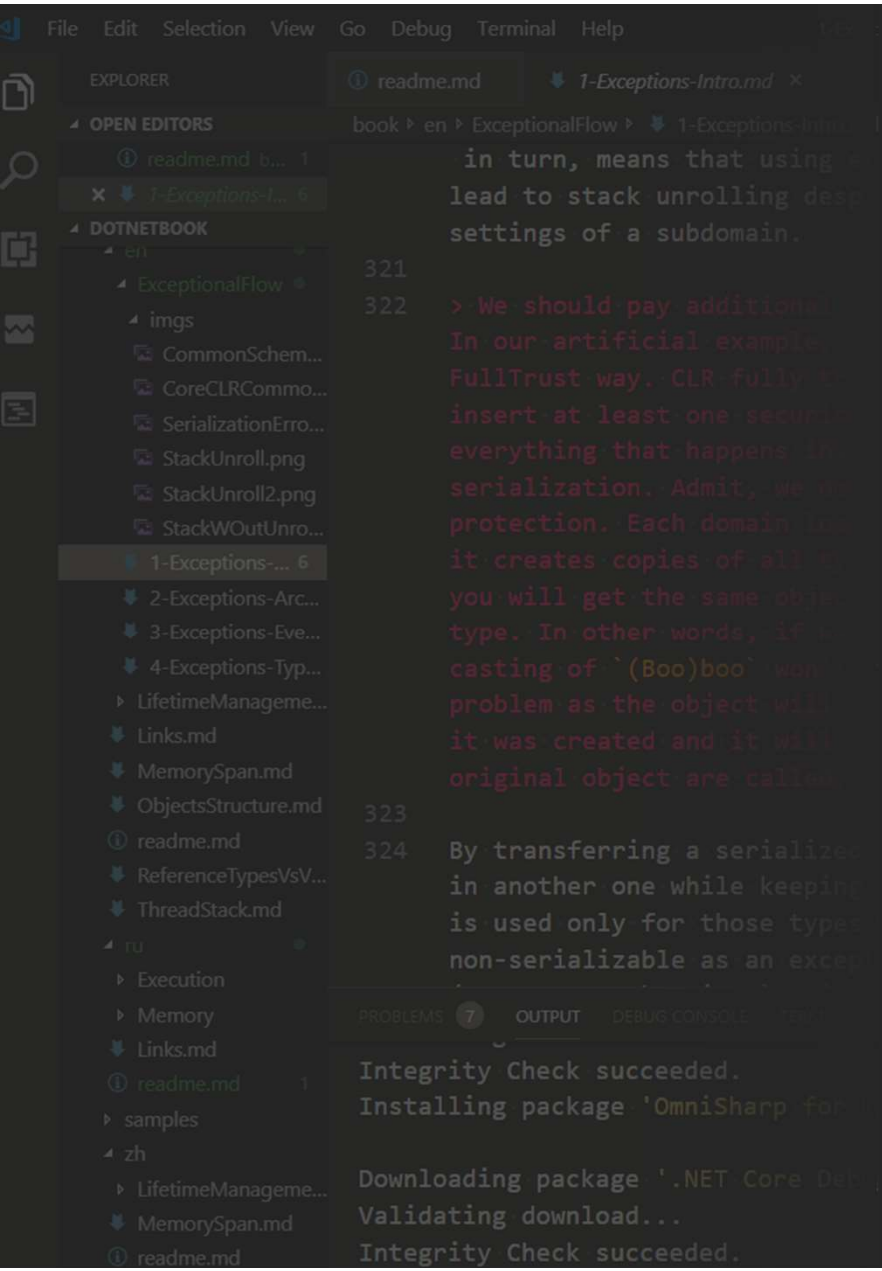
# Разработка собственного пула потоков

## Для получения дополнительного контроля исполнения

https://github.com/sidristij/threading-kit

Сидристый Станислав




# Станислав Сидристый

- R&D Team Lead в Центре Речевых Технологий
- C#, C/C++, C++/CLI когда необходимо
- Книга: https://github.com/sidristij/dotnetbook
- Проект: https://github.com/sidristij/threading-kit
- telegram: @sidristij
- sunex.development@gmail.com

# и целого мира мало

***Первый акт*** *— это презентация основных действующих лиц, зов к странствию, встреча с наставником, принятие зова*

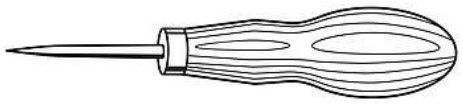
kwanini?

zakaj?

どうして？

為什麼？

ทำไม?



ngoba?

hvorfor?

# Зачем? Why? Warum?

чаро?

vim li cas?

perché?

چرا؟

kodėl?

왜요?

нигə?

# Тест

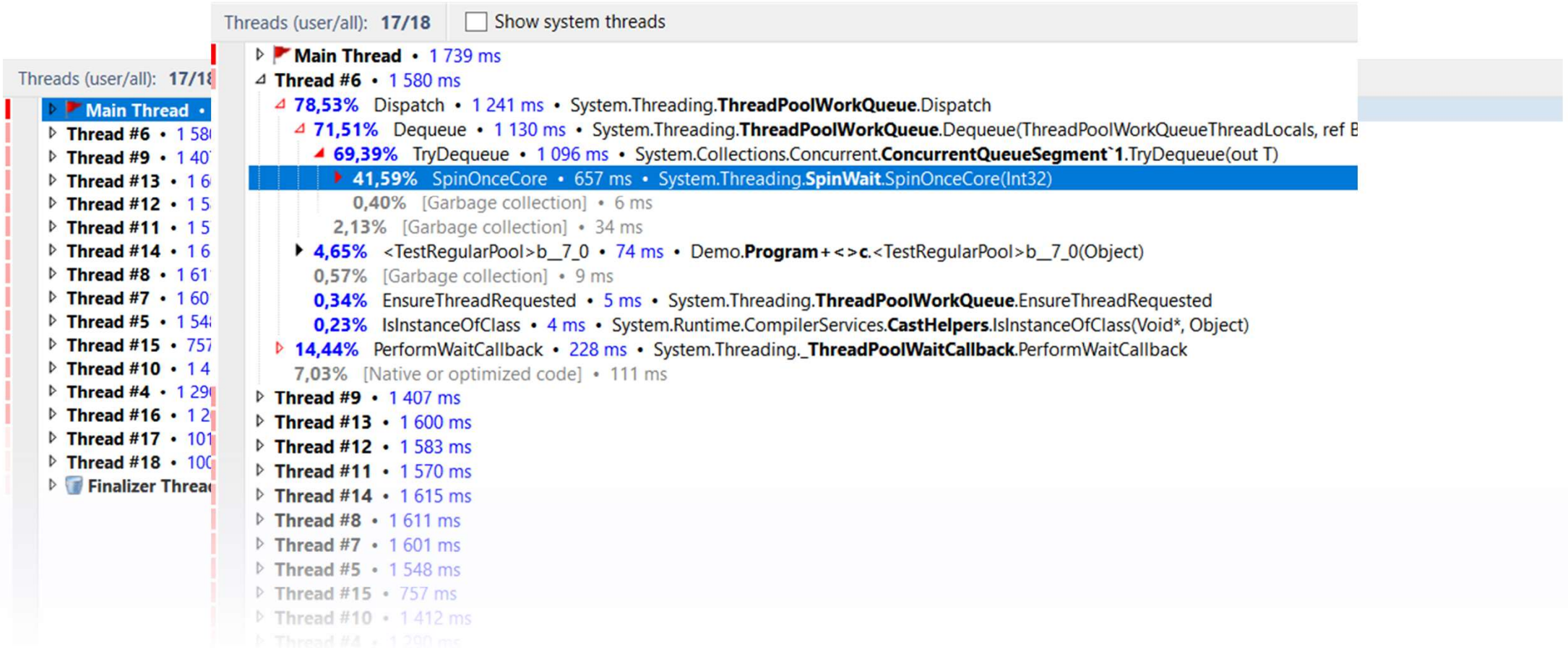
```
private static int TestRegularPool()
{
    var sw = Stopwatch.StartNew();
    var @event = new CountdownEvent(count);

    for (var i = 0; i < count; i++)
    {
        ThreadPool.QueueUserWorkItem((x) ⇒
        {
            x.Signal();
        }, @event, false);
    }

    @event.Wait();
    sw.Stop();
    return (int)sw.ElapsedMilliseconds;
}
```

```
private static int TestSmartPool(SmartThreadPool<object> pool)
{
    var sw = Stopwatch.StartNew();
    var @event = new CountdownEvent(count);

    for (var i = 0; i < count; i++)
    {
        pool.Enqueue((state) ⇒
        {
            ((CountdownEvent)state).Signal();
        }, @event, false);
    }

    @event.Wait();
    sw.Stop();
    return (int)sw.ElapsedMilliseconds;
}
```

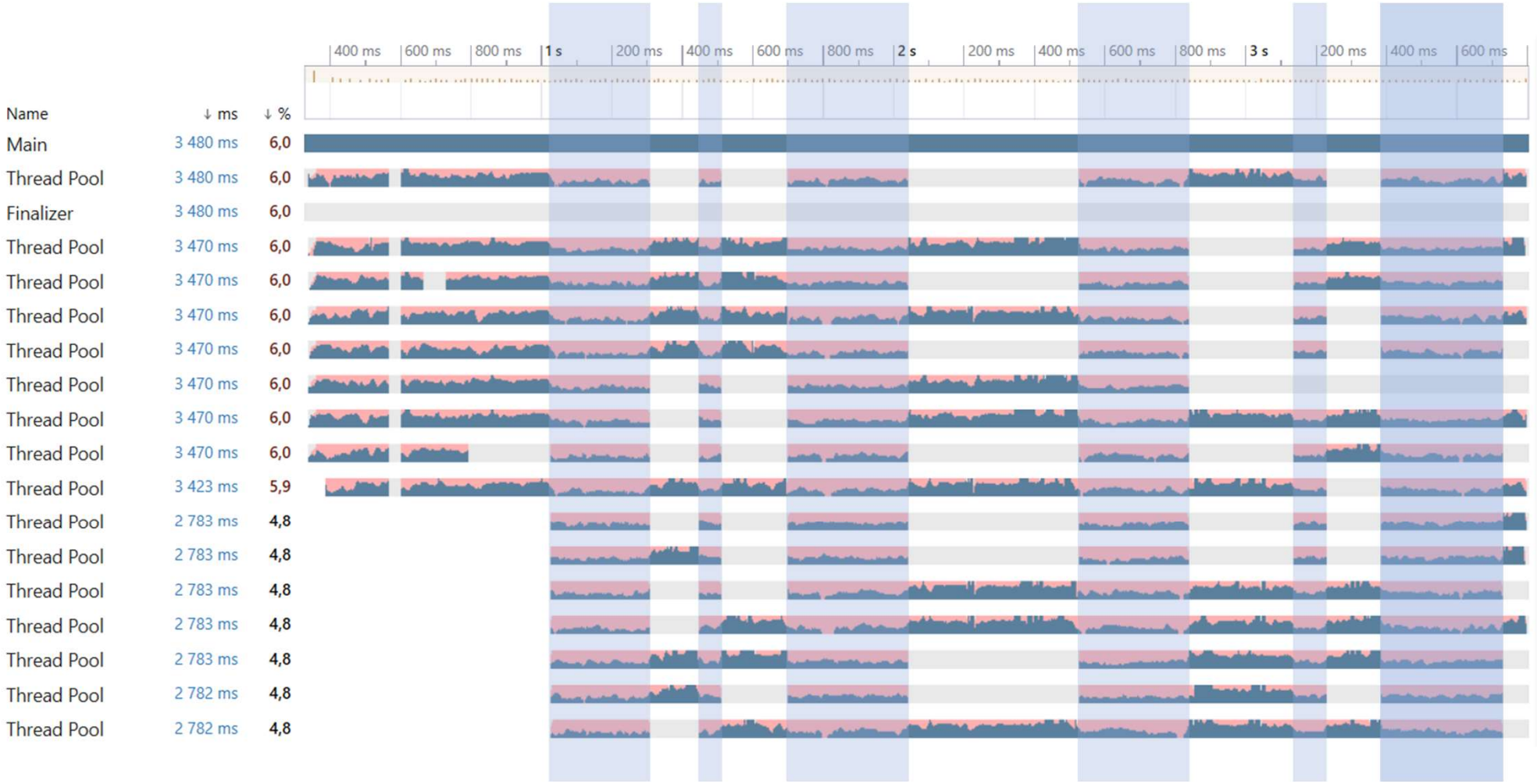
~112,1 мсек

- 28%

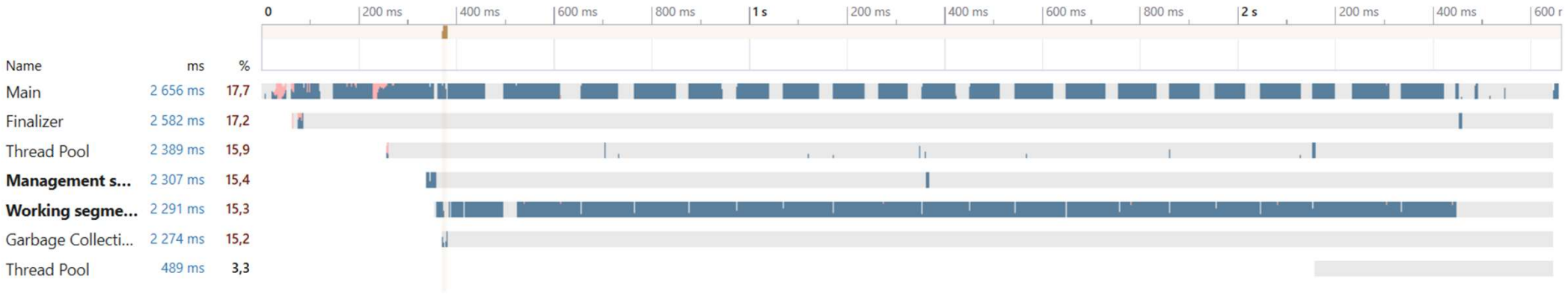
~80,5 мсек


Threads (user/all): 17/18
Show system threads
Main Thread • 1 739 ms
Thread #6 • 1 580 ms
78,53% Dispatch • 1 241 ms • System.Threading.ThreadPoolWorkQueue.Dispatch
71,51% Dequeue • 1 130 ms • System.Threading.ThreadPoolWorkQueue.Dequeue(ThreadPoolWorkQueueThreadLocals, ref E
69,39% TryDequeue • 1 096 ms • System.Collections.Concurrent.ConcurrentQueueSegment`1.TryDequeue(out T)
41,59% SpinOnceCore • 657 ms • System.Threading.SpinWait.SpinOnceCore(Int32)
0,40% [Garbage collection] • 6 ms
2,13% [Garbage collection] • 34 ms
4,65% <TestRegularPool>b__7_0 • 74 ms • Demo.Program+<>c.<TestRegularPool>b__7_0(Object)
0,57% [Garbage collection] • 9 ms
0,34% EnsureThreadRequested • 5 ms • System.Threading.ThreadPoolWorkQueue.EnsureThreadRequested
0,23% IsInstanceOfClass • 4 ms • System.Runtime.CompilerServices.CastHelpers.IsInstanceOfClass(Void*, Object)
14,44% PerformWaitCallback • 228 ms • System.Threading._ThreadPoolWaitCallback.PerformWaitCallback
7,03% [Native or optimized code] • 111 ms
Thread #9 • 1 407 ms
Thread #13 • 1 600 ms
Thread #12 • 1 583 ms
Thread #11 • 1 570 ms
Thread #14 • 1 615 ms
Thread #8 • 1 611 ms
Thread #7 • 1 601 ms
Thread #5 • 1 548 ms
Thread #15 • 757 ms
Thread #10 • 1 412 ms
Threads (user/all): 17/18
Main Thread •
Thread #6 • 1 58
Thread #9 • 1 40
Thread #13 • 1 6
Thread #12 • 1 5
Thread #11 • 1 5
Thread #14 • 1 6
Thread #8 • 1 61
Thread #7 • 1 60
Thread #5 • 1 54
Thread #15 • 757
Thread #10 • 1 4
Thread #4 • 1 29
Thread #16 • 1 2
Thread #17 • 101
Thread #18 • 100
Finalizer Threa

# ThreadPool


400 ms
600 ms
800 ms
1 s
200 ms
400 ms
600 ms
800 ms
2 s
200 ms
400 ms
600 ms
800 ms
3 s
200 ms
400 ms
600 ms
Name
↓ ms
↓ %
Main 3 480 ms 6,0
Thread Pool 3 480 ms 6,0
Finalizer 3 480 ms 6,0
Thread Pool 3 470 ms 6,0
Thread Pool 3 470 ms 6,0
Thread Pool 3 470 ms 6,0
Thread Pool 3 470 ms 6,0
Thread Pool 3 470 ms 6,0
Thread Pool 3 470 ms 6,0
Thread Pool 3 470 ms 6,0
Thread Pool 3 423 ms 5,9
Thread Pool 2 783 ms 4,8
Thread Pool 2 783 ms 4,8
Thread Pool 2 783 ms 4,8
Thread Pool 2 783 ms 4,8
Thread Pool 2 783 ms 4,8
Thread Pool 2 782 ms 4,8
Thread Pool 2 782 ms 4,8

# SmartThreadPool*






* не содержит ГМО
перед использованием посоветуйтесь с врачом

история про серийное

# убийство

# Решаемые проблемы

- Контроль lifetime потока:
  - Пользовательская оболочка с OnStarted, OnStopping для потока
  - OnRun(PoolWork item) => item.Run(_connection);
    для проброса состояния из оболочки потока
- Внешнее управление количеством потоков
- Повышение скорости за счёт уменьшения борьбы за очередь
- Сдерживание «страха» перед большой очередью задач
- Авторазблокирование

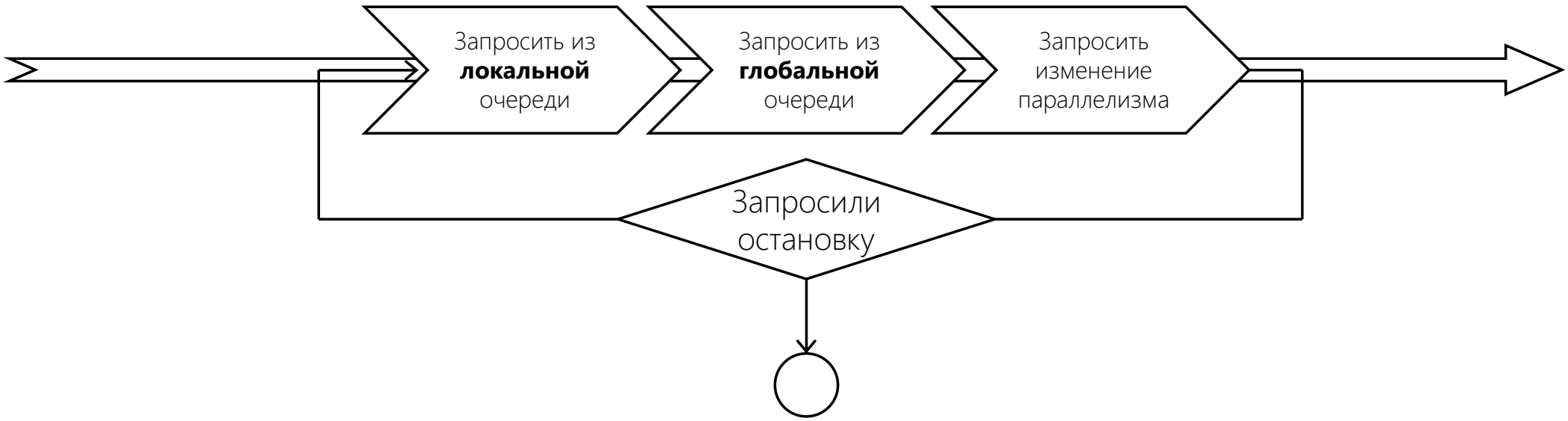
# Пользовательская логика

```
protected override void OnStarted()
{
    _connection = ConnectionProvider.Create();
}

protected override Task OnRun(PoolWork poolWork)
{
    return poolWork.Run(_connection);
}

protected override void OnStopping()
{
    _connection.Dispose();
}
```

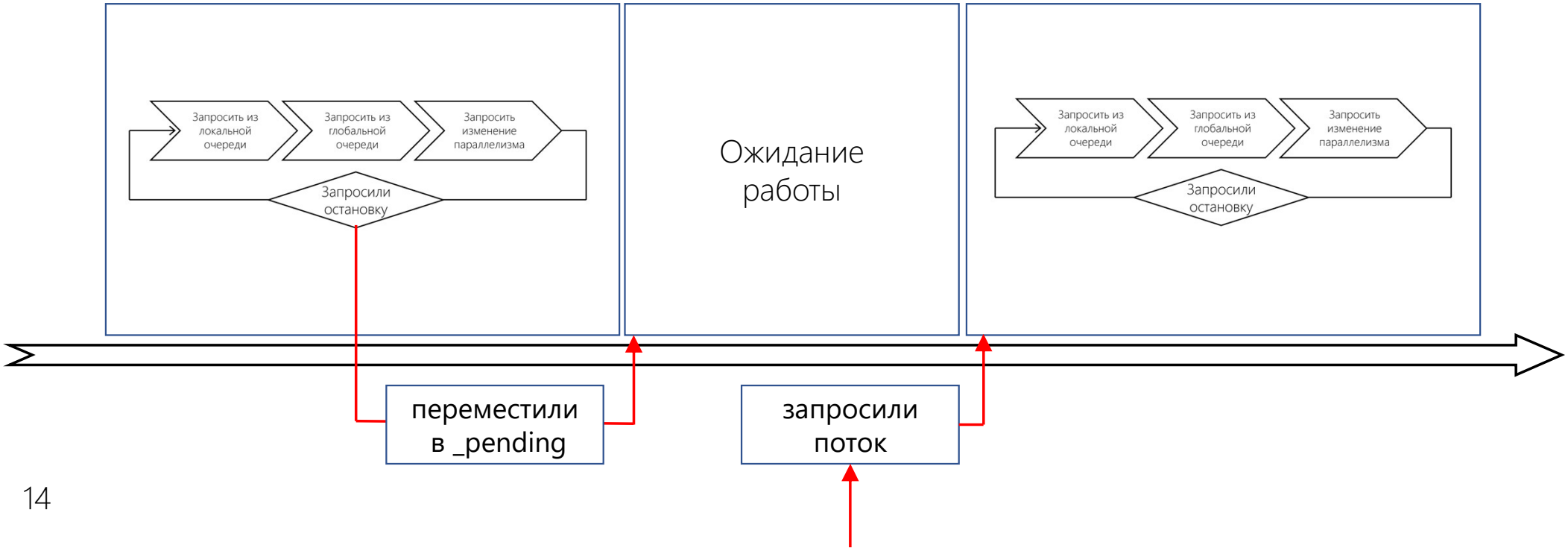
# Разработка

***Второй акт*** *— это самая большая, основная часть истории. Здесь вызов, брошенный антагонистом, заставляет героя действовать. К середине второго акта он уже не может повернуть назад, как бы ему этого ни хотелось. Во второй половине второго акта многократно возрастают ставки и риски. И к концу второго акта герой терпит большое поражение, оказываясь в максимальной опасности, практически в безвыходной ситуации.*

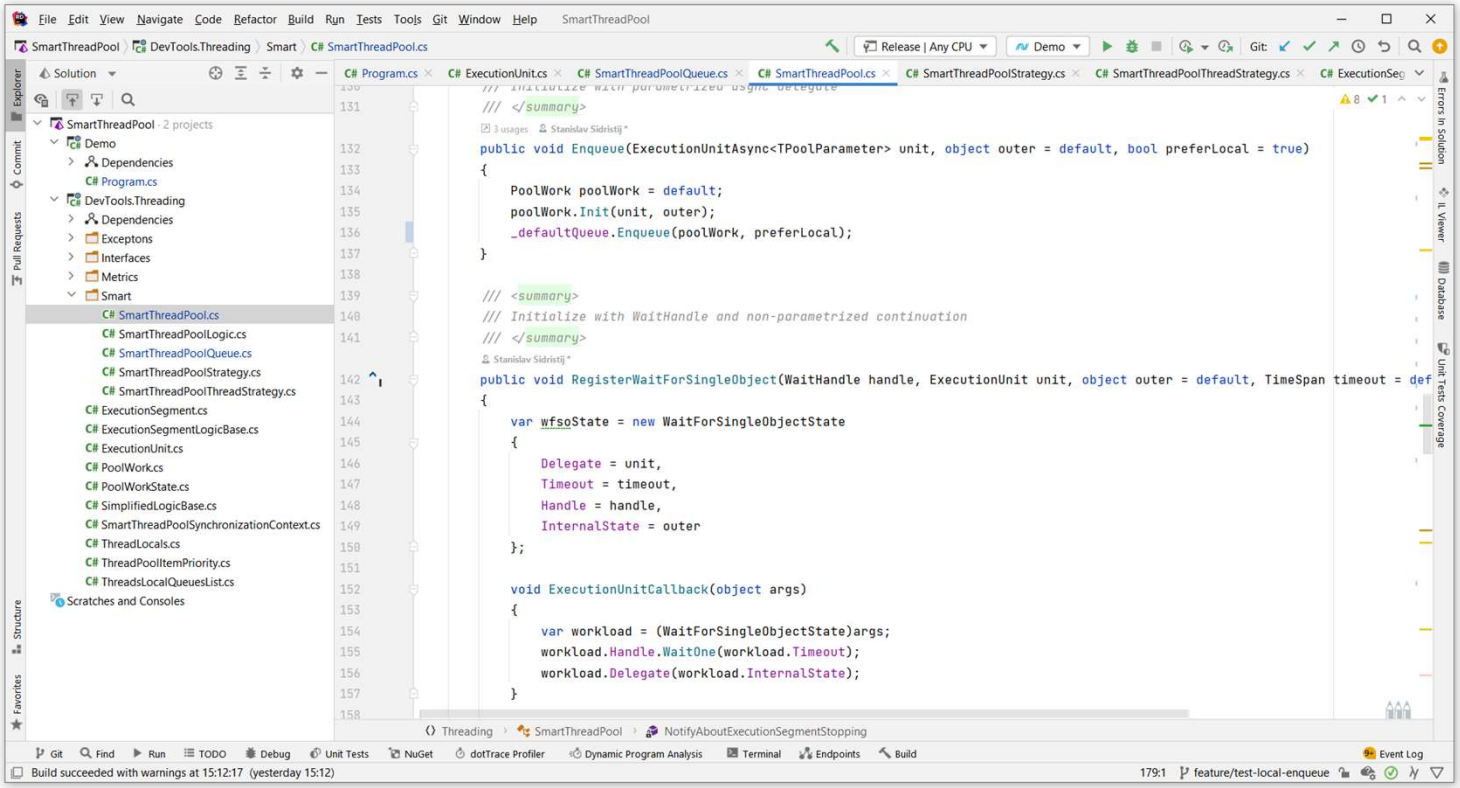
# Очередь отрезков исполнения



# Очередь отрезков исполнения



# Код


File Edit View Navigate Code Refactor Build Run Tests Tools Git Window Help SmartThreadPool
SmartThreadPool DevTools.Threading Smart SmartThreadPool.cs
Release | Any CPU Demo Git:
Solution
SmartThreadPool · 2 projects
Demo
Dependencies
Program.cs
DevTools.Threading
Dependencies
Exceptons
Interfaces
Metrics
Smart
SmartThreadPool.cs
SmartThreadPoolLogic.cs
SmartThreadPoolQueue.cs
SmartThreadPoolStrategy.cs
SmartThreadPoolThreadStrategy.cs
ExecutionSegment.cs
ExecutionSegmentLogicBase.cs
ExecutionUnit.cs
PoolWork.cs
PoolWorkState.cs
SimplifiedLogicBase.cs
SmartThreadPoolSynchronizationContext.cs
ThreadLocals.cs
ThreadPoolItemPriority.cs
ThreadsLocalQueuesList.cs
Scratches and Consoles
Program.cs ExecutionUnit.cs SmartThreadPoolQueue.cs SmartThreadPool.cs SmartThreadPoolStrategy.cs SmartThreadPoolThreadStrategy.cs
/// </summary>
3 usages Stanislav Sidristij
public void Enqueue(ExecutionUnitAsync<TPoolParameter> unit, object outer = default, bool preferLocal = true)
{
PoolWork poolWork = default;
poolWork.Init(unit, outer);
_defaultQueue.Enqueue(poolWork, preferLocal);
}
/// <summary>
/// Initialize with WaitHandle and non-parametrized continuation
/// </summary>
Stanislav Sidristij
public void RegisterWaitForSingleObject(WaitHandle handle, ExecutionUnit unit, object outer = default, TimeSpan timeout = def
{
var wfsoState = new WaitForSingleObjectState
{
Delegate = unit,
Timeout = timeout,
Handle = handle,
InternalState = outer
};
void ExecutionUnitCallback(object args)
{
var workload = (WaitForSingleObjectState)args;
workload.Handle.WaitOne(workload.Timeout);
workload.Delegate(workload.InternalState);
}
Threading SmartThreadPool NotifyAboutExecutionSegmentStopping
Git Find Run TODO Debug Unit Tests NuGet dotTrace Profiler Dynamic Program Analysis Terminal Endpoints Build Event Log
Build succeeded with warnings at 15:12:17 (yesterday 15:12)
179:1 feature/test-local-enqueue

# Выводы

# Выводы

- Если замечать проблемы, придёт понимание, как их решать
- Можно менять стандартные, привычные механизмы
- Решение может получиться лучше чем у профи в .NET Team
- Если не попробуешь – никогда не узнаешь.


QA