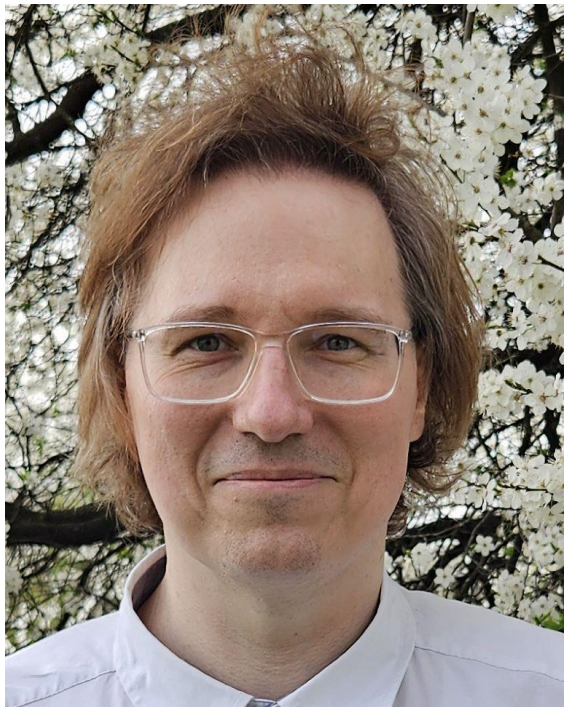
.NET - Programowanie równoległe ~~i trochę rozproszone~~ kiedy (i jak)

Tomasz Kopacz

35 minut



Demos: https://github.com/tkopacz/msts2024-net-parallel

“Standard” computers… (and servers)

2005	2015	2024
Pentium 4, HT, 3.74 GHz	i5, 6 cores + HT, 3.2 GHz	I7/i9, 20 + HT, 3.4-5 GHz
512 MB, 1GB SOMETIMES	8 GB	32/128 GB
10GFLOPS | 10K MIPS	1TFLOPS | 100K MIPS	100TFLOPS | 1M MIPS
Xeon, 2 cores+ HT, 3.6 GHz	E5, 8 cores + HT, 2.4 GHz	Gold, 28 cores + HT, 2.6 GHz
4 GB	32 GB	256 GB
1 048 576 4KB records	8 388 608 4KB records	67 108 864 4KB records
		IN RAM!
		67M (256 is not too much!)

3.5 GHz = 1 cycle every 1/3 500 000 000 second = 2.85 ns
*Speed of light = 3 * 108 m / sec. 30 cm in 1 nanosecond!*

Imagine – 100GHz, cycle 0.01ns, size 0.3 mm ……
(“slowly” quantum physics)

Business – real one

News webpage/sec

Wikipedia: 30K / sec, 300 machines, 100/machine

Twitter: 800 connection per second

NYT: 200-300 per sec, 10K peak

Orders/day

Restaurant: 50 (EU) - 2000 (China) per day

Grocery, Retail: 1000

E-commerce: 5000

Estimations!

- Amazon: 33M
  - 200M Prime Members
- Allegro: 0.7M

Search, scale events

There are 0.6 million Google searches per second

100K/sec in Azure Event Hub (IoT, telemetry etc)

But:

Intelligent Thermometer

- 1 reading per 30 seconds

Hardware....

PCI-E

- 3.0: 15.8 GB/sec
- 4.0: 31.5 GB/sec
- 5.0: 63 GB/sec
- (7.0,224) 242 GB/s

NVME: 3-12GB / sec

SSD/SATA: 0.5GB / sec

DDR5: 64GB/s

ETH 10Gbit: 1 GB / sec

8,112,077,412 people on Earth

≈ $6*10^9$ mobiles + $3*10^9$ „PC”

+ traffic from bots ☺

Clemens Vasters • 1st — 2d (edited)

Principal Architect, Messaging Services and Standards, Microsoft

Azure Event Hubs handles roughly 120 million requests per second. No typo. We have quite a few customers who do more than 100K/sec just at the messaging level.

Edit: Made a correction to 1/7th of what I had originally written since I looked up into wrong column -- weekly instead of daily -- and we usually don't calculate req/sec at that level. Sorry. The number is still huge :)

Options to run paralel code in .NET

(normal Threads)

(async/await)

Tasks and Task Parallel Library

Parallel.For | Parallel.ForEach | await foreach (IAsyncEnumerable)

PLINQ (Data Parallelism)

Data Flow

Reactive Extension (Rx)

Scenarios

Server-side, console

Also:

One Blazor
One Windows Forms

What are the limits for concurrent programming

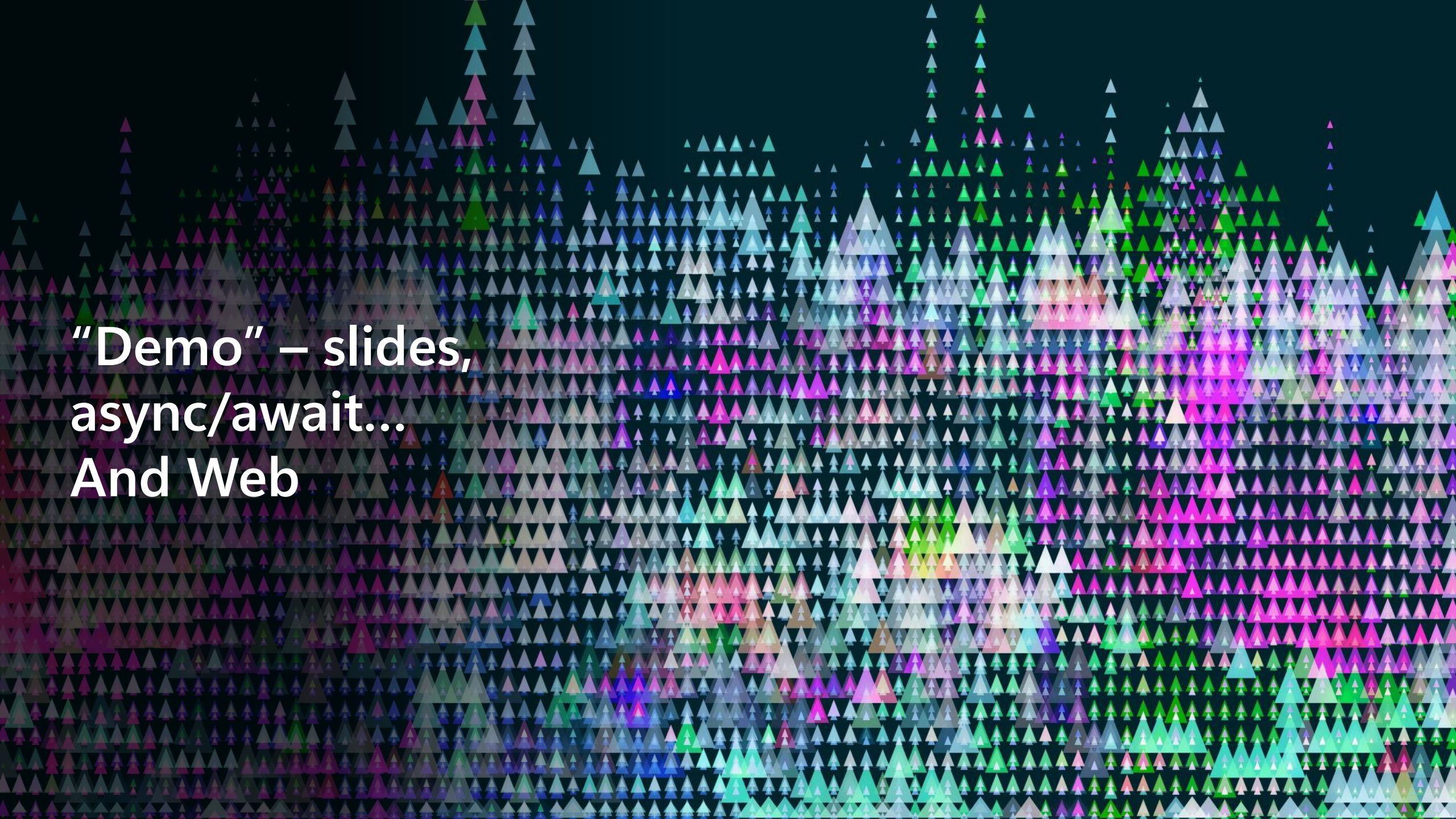
Local Hardware

- *CPU/Cores/Threads*
- *Sockets number*
- *I/O throughput*
- *Memory (rare)*

Remote (external) services

- *SQL DTU, Max RU/s* (CosmosDB), Tokens/second....
- *Network throughput*

Our own mind / knowledge / skills!


“Demo” – slides,
async/await…
And Web

```
app.MapGet("/long", () =>
{
    var forecast = Enumerable.Range(1, 500).Select(index =>
        new LongRecord
        (
            new string('t', index *2 + Random.Shared.Next(255)),
            new string('d', index * 3 + Random.Shared.Next(255)),
            1000 + Random.Shared.Next(2000)
        ))
        .ToArray();
    return forecast;
});
```

Sync Web Api

Async Web Api

```
app.MapGet("/longasync", async () =>
{
    return await Task.Run(() =>
    {
        var forecast = Enumerable.Range(1, 500).Select(index =>
        new LongRecord
        (
            new string('t', index * 2 + Random.Shared.Next(255)),
            new string('d', index * 3 + Random.Shared.Next(255)),
            1000 + Random.Shared.Next(2000)
        ))
        .ToArray();
        return forecast;
    }
    );
});
```

Numbers… (ACA, 1vCPU + 1GB, all defaults)

400 vUsers

Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	**4000**	**0**	**0.00%**	**3666.02**	**103**	**29388**	**2712.00**	**7291.80**	**9576.25**	**13189.95**	**79.62**	**60541.72**	**16.95**
Sync	4000	0	0.00%	3666.02	103	29388	2712.00	7291.80	9576.25	13189.95	79.62	60541.72	16.95

Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	**4000**	**0**	**0.00%**	**3995.92**	**369**	**27644**	**3294.00**	**7281.60**	**8832.75**	**12063.70**	**70.57**	**53660.71**	**14.06**
Async	4000	0	0.00%	3995.92	369	27644	3294.00	7281.60	8832.75	12063.70	70.57	53660.71	14.06

450 vUsers, Async – **2x** more errors than sync (and will grow **FASTER**)

Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	**4500**	**19**	**0.42%**	**1475.63**	**428**	**24961**	**3211.00**	**9528.90**	**10250.90**	**13385.36**	**75.83**	**57414.85**	**16.08**
Sync	4500	19	0.42%	1475.63	428	24961	3211.00	9528.90	10250.90	13385.36	75.83	57414.85	16.08

Requests	Executions			Response Times (ms)							Throughput	Network (KB/sec)	
Label	#Samples	FAIL	Error %	Average	Min	Max	Median	90th pct	95th pct	99th pct	Transactions/s	Received	Sent
Total	**4500**	**49**	**1.09%**	**7526.70**	**134**	**249254**	**6166.50**	**17408.20**	**22149.65**	**25108.06**	**15.91**	**11966.40**	**3.16**
Async	4500	49	1.09%	7526.70	134	249254	6166.50	17408.20	22149.65	25108.06	15.91	11966.40	3.16

Normal Threads: Thread, and ThreadPool

Thread = OS Thread. Mapping 1:1 to OS Threads

```
Thread t01 = new Thread(() => Console.WriteLine($"Hello
{Thread.CurrentThread.ManagedThreadId}"));
t01.Start();
```

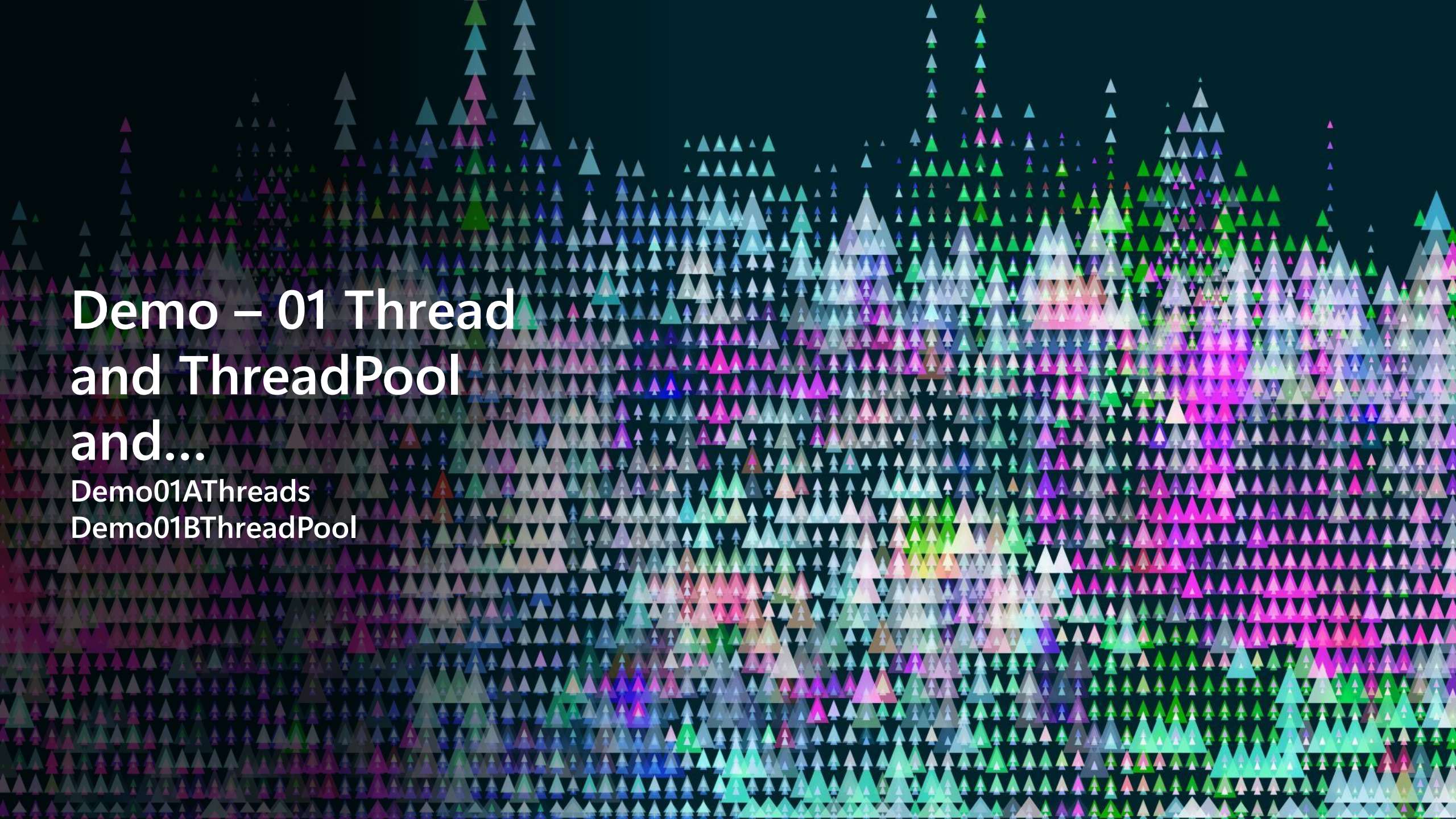
ThreadPool = pool of threads

```
ThreadPool.QueueUserWorkItem
```

Important .Abort, .Kill –deprecated!

Use CancellationTokenSource and CancellationToken

(or kill new PROCESS)


Demo – 01 Thread
and ThreadPool
and…
Demo01AThreads
Demo01BThreadPool

Threads/Task/Pool - [app].runtimeconfig.json | Env

System.GC.Server: (WS – one CPU, true, SRV – dedicated threads)

System.GC.Concurrent: (background), one per logical vCPU

System.GC.HeapAffinitizeMask, System.GC.CpuGroup, System.GC.HeapCount, System.GC.HeapHardLimit, LOH | SOH | POH Settings, System.GC.Name (custom), System.GC.ConserveMemory

DOTNET_Thread_UseAllCpuGroups | DOTNET_Thread_AssignCpuGroups

System.Threading.ThreadPool.MinThreads | System.Threading.ThreadPool.MaxThreads

System.Threading.ThreadPool.UseWindowsThreadPool

System.Threading.ThreadPool.Blocking.* : sync-over-async case & block, AVOID (in general)

Also: AppContext.SetSwitch (some settings)

*Ps. COMPlus_ =>DOTNET_*

Task Parallelism, Data Parallelism: Task Parallel Library

Why “threads” are BAD?

- Does not scale as well as you think. Thread/Context switching over head
- Thread starvation. You own synchronization

TPL handles: partitioning, the scheduling of threads on the ThreadPool, cancellation support, state management, …

Also: Concurrent collections – to make life a little bit simpler…

Parallel.For / ForEach

PLINQ: 2 syntaxes – similar to SQL and adopting fluent API approach

async / await - for – I/O Bound work, like:

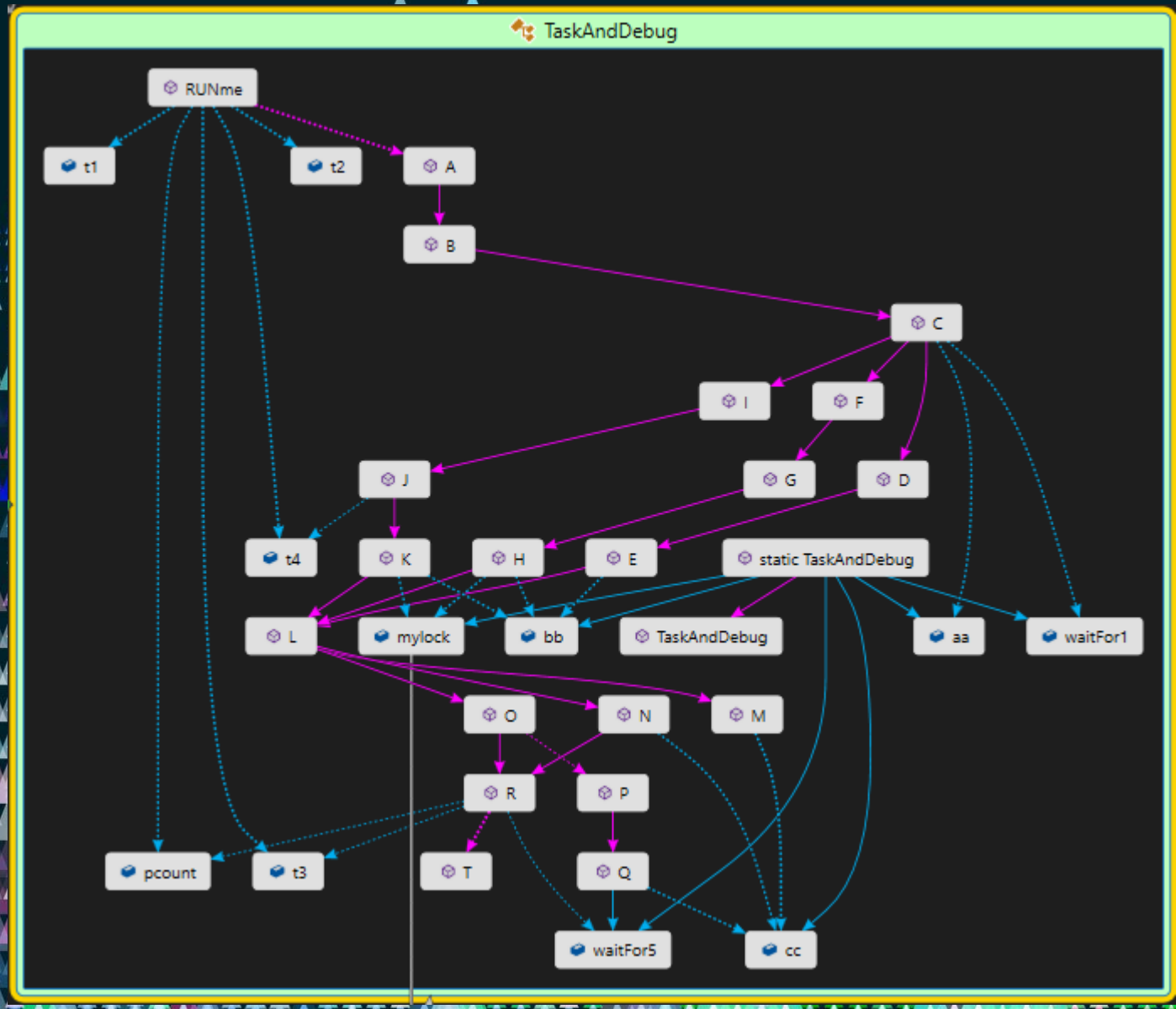
- HTTPClient, Disk I/O, Socket I/O. Also: Task.Run !
- Not (always) new thread. Thread Context – the same (or different) :
  `taskA.ConfigureAwait(true)`

Demo – 02 TPL

Demo02TPL

Demo02TPLExceptions

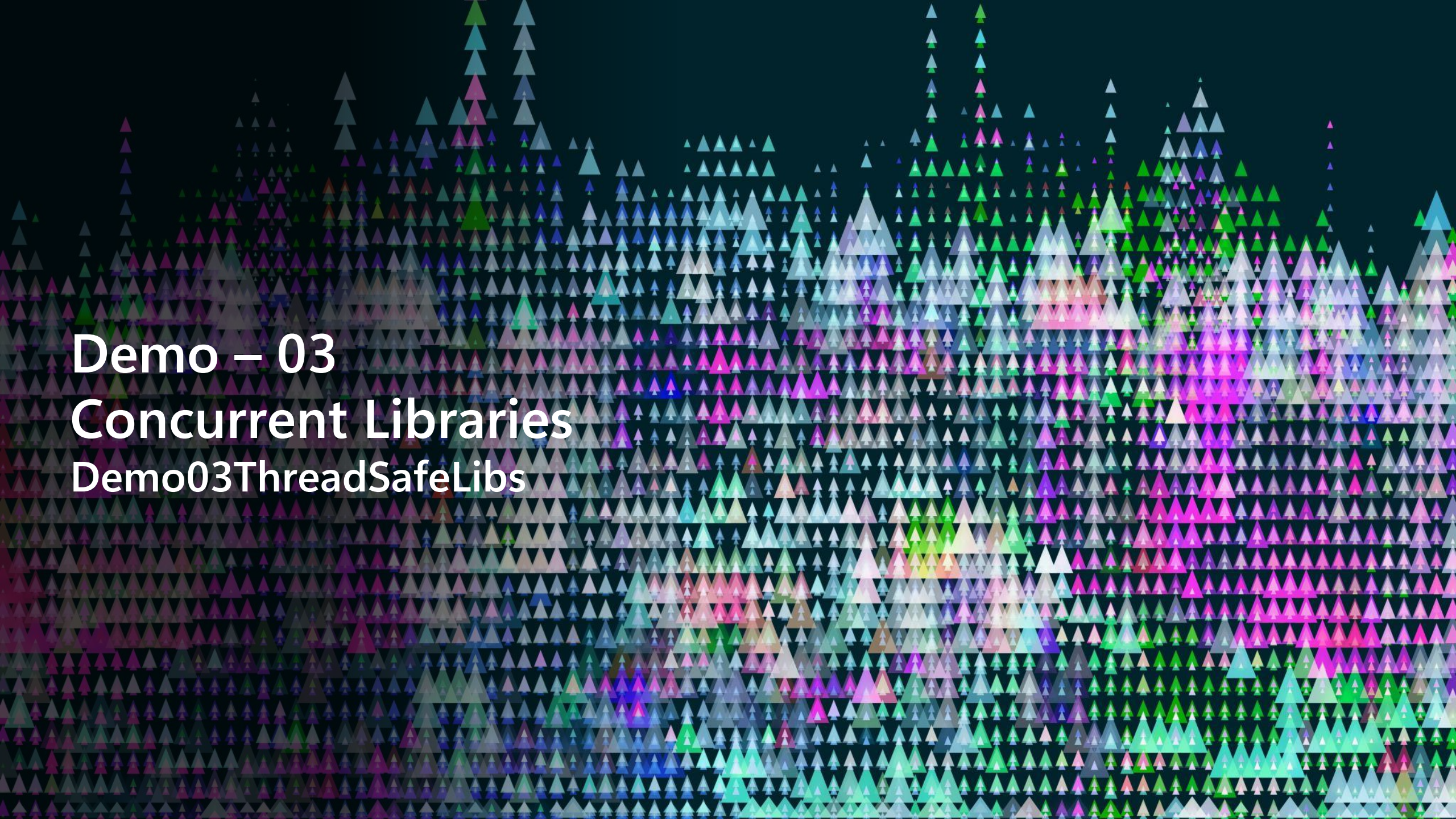
Demo02TPLSynchronization



Case – Blazor

Server and client-side rendering

Blazor-SimplestDemo



Demo – 03
Concurrent Libraries
Demo03ThreadSafeLibs


Demo – 04 PLINQ
Demo04PLINQ
Demo04PLINQSetPixel

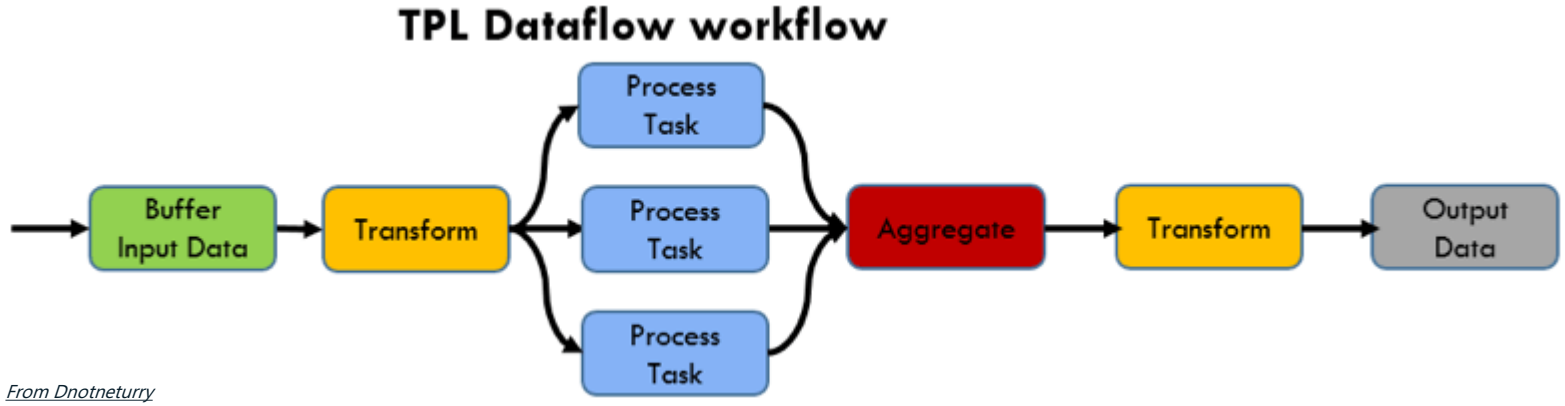
TPL Dataflow

Message passing and parallelizing CPU-intensive **AND** I/O intensive application.
High throughput **AND** low latency.

Concept like actor-based programming (Orleans, Akka)

- In-process message passing, ETL like pipelines, not distributed
- VERY similar semantic
- Helpful to avoid deadlocks
- Ps. Dataflows are also in Power Query – very similar semantic

Blocks: Source, Propagator, Target, Pipeline



From Dnotneturry

Demo – 05 Data Flow

Demo05DataFlow

Rx – Reactive Extension for .NET

.NET library for processing event streams – here. Observable pattern

Better Events, modern Streams…

```
IObservable bigTrades = trades.Where(trade => trade.Volume > 1_000_000);
bigTrades.Subscibe( …)
```

(Like IEnumerable, but will add a new element when available)

Fully integrated with ecosystem: combination with TPL Dataflow, (P)LINQ etc

Scenarios:

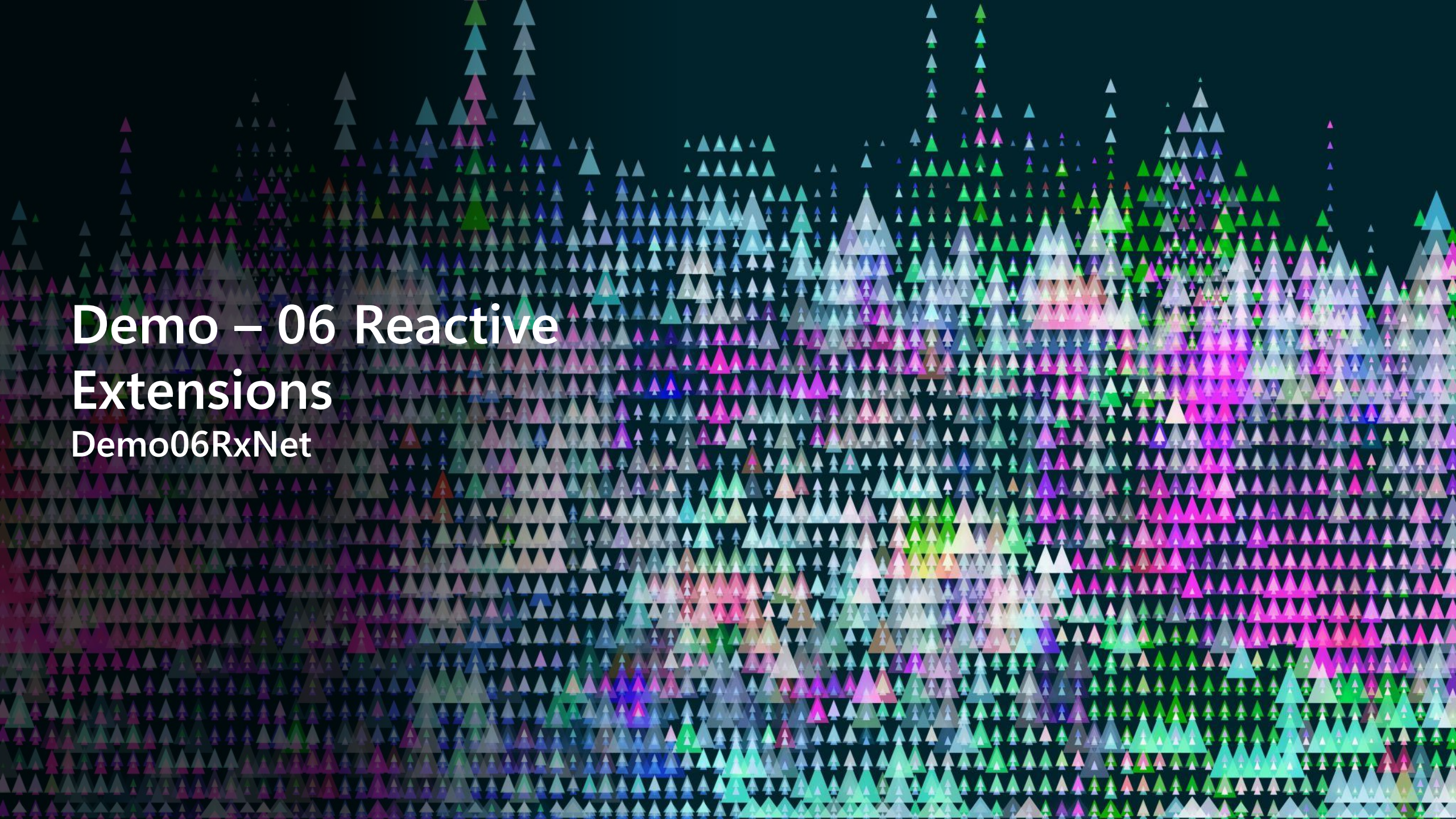
Complex Event Processing, **CQRS**, Integration with IoT Brokers, UI!, Tracking in **real time** (Web, GPS, ….). Domain Model Events. Broadcasting.

Event passing vs complex thread topologies

Also, REAQTOR (Rx + state, durability)

Also, integration with Blazor (and others UI frameworks!)

Also (project from 2010) many additional integrations…



Demo – 06 Reactive Extensions

Demo06RxNet

Summary – how to “think”

Use mix of Data and Task Parallelism (map-reduce, fork-join etc)

- Mandatory Data Parallelism for larger sets

Use Task Parallel Library – “serialize” parallel code

- async/await for all I/O bound. Actors “like” approach – maybe DataFlow | Reactive Extensions

Think before “rewriting”

Treat “TPL” as a **standard** toolbox

- Classic threads – only for exams/tests/jobs interviews ;) or – libraries. Time to **write tests**!

Prepare code – avoid “globals”, keep local calculation etc…

Supportive materials:

- .NET docs - https://github.com/dotnet/docs
- Excellent - https://github.com/dotnet/runtime/tree/main/docs
- https://github.com/dotnet/performance

FEEDBACK

.NET - Programowanie równoległe i trochę rozproszone - kiedy (i jak)

Tomasz Kopacz

https://mstechsummit.pl/user.html#!/lecture/MSTS24-19c1/rate

Microsoft

Questions?

tkopacz@microsoft.com

Demos: https://github.com/tkopacz/msts2024-net-parallel