Asynchronní programování v .NET

Tomáš Jecha
Microsoft MVP

Mail: tomas@jecha.net | Twitter: @jechtom
http://www.jecha.net | http://www.vbnet.cz

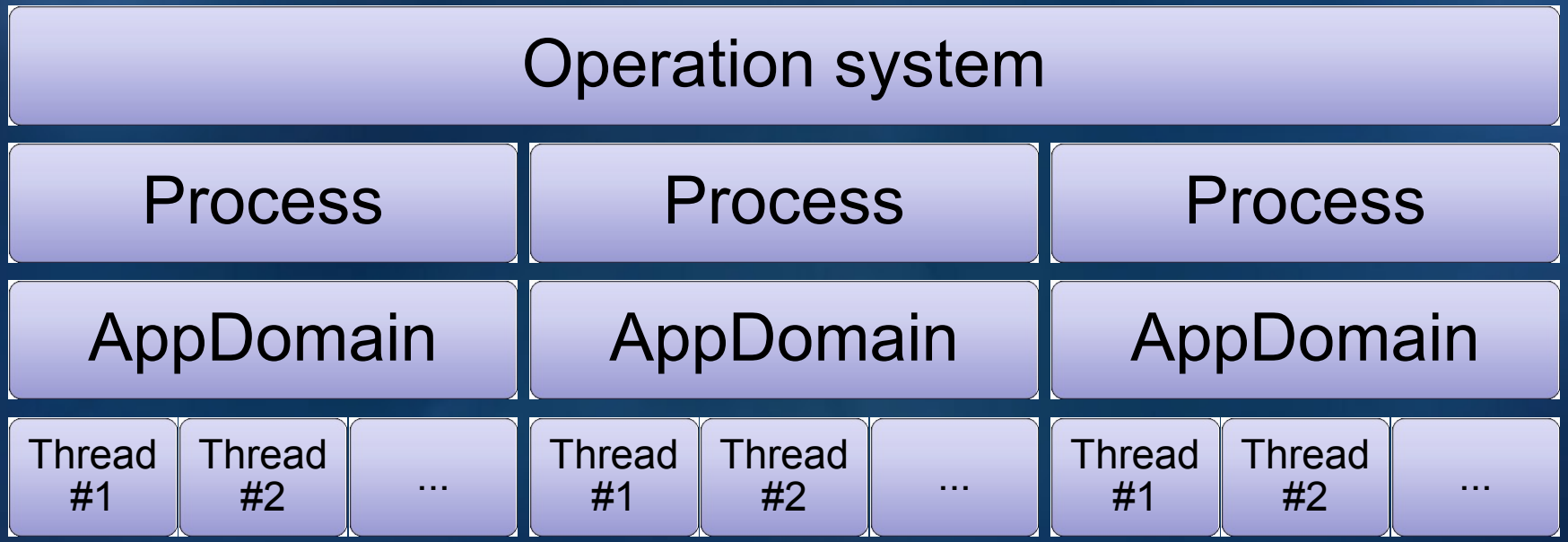
Agenda

- Procesy, vlákna, synchronizační primitiva, ThreadPool
- Task Parallel Library
- Thread-Safe Collections
- Event-based Asynchronous Pattern
- Synchronizační kontext
- WPF aplikace
- Webové aplikace
- async / await

PROCESY A VLÁKNA

Proces v .NET

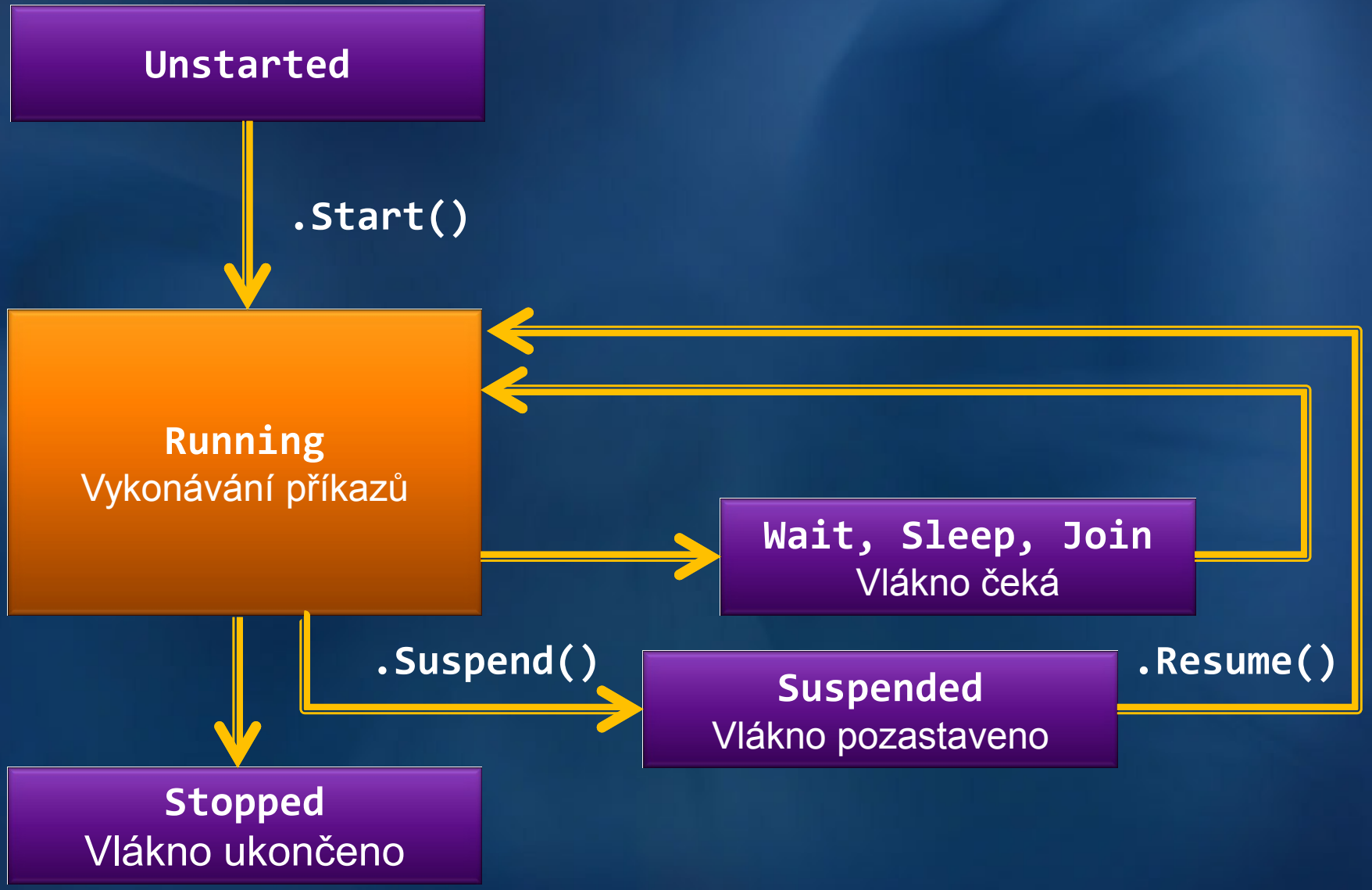
- Operační systém hostuje procesy
- Proces = izolovaná instance aplikace (.exe)
- Uvnitř procesu se vytváří Application Domain
  - Hostitel pro .NET aplikaci a její vlákna
  - Data jsou mezi vlákny v AppDomain sdílená



Vlákno v .NET

- Třída: *System.Threading*.**Thread**
- Aktuální vlákno: Thread.**CurrentThread**
- Lze nastavit:
  - Delegát spuštění
  - Parameter spouštění
  - Name
  - Priority
  - IsBackground

Životní cyklus vlákna v .NET

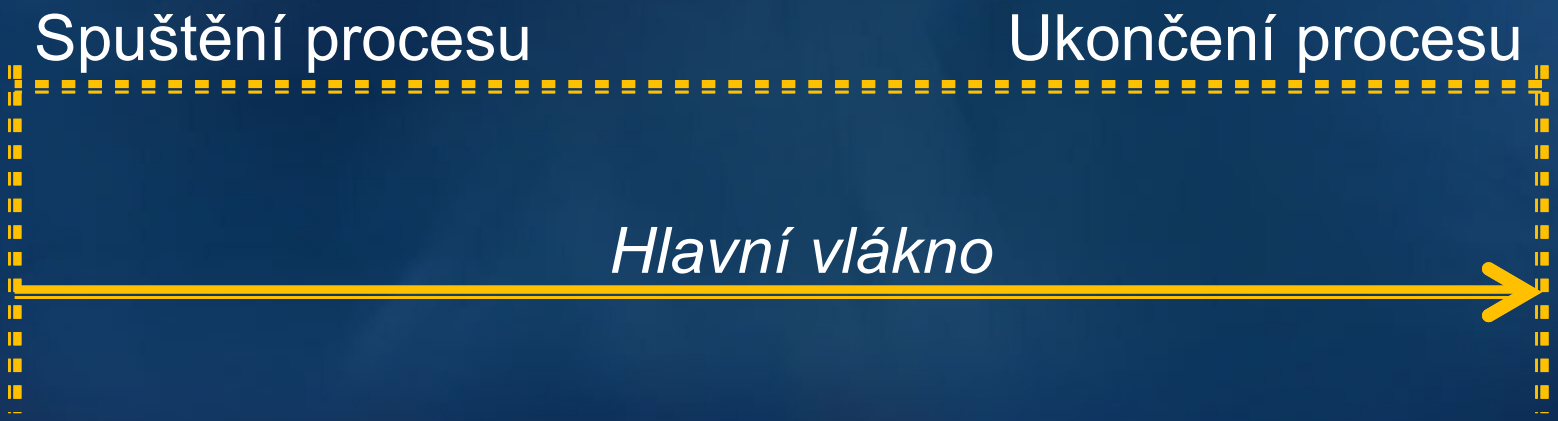


Typy čekání vlákna

- Wait
  - Čekání na impuls z jiného vlákna
  - Třídy dědící z **`WaitHandle`**
  - Nebo využití **`lock`** (**Monitor**)
- Sleep
  - Čekání specifický čas
  - **`Thread.Sleep`**`(TimeSpan.FromSeconds(5))`
- Join
  - Čekání na ukončení jiného vlákna
  - *`vlákno`*`.`**`Join`**`()`

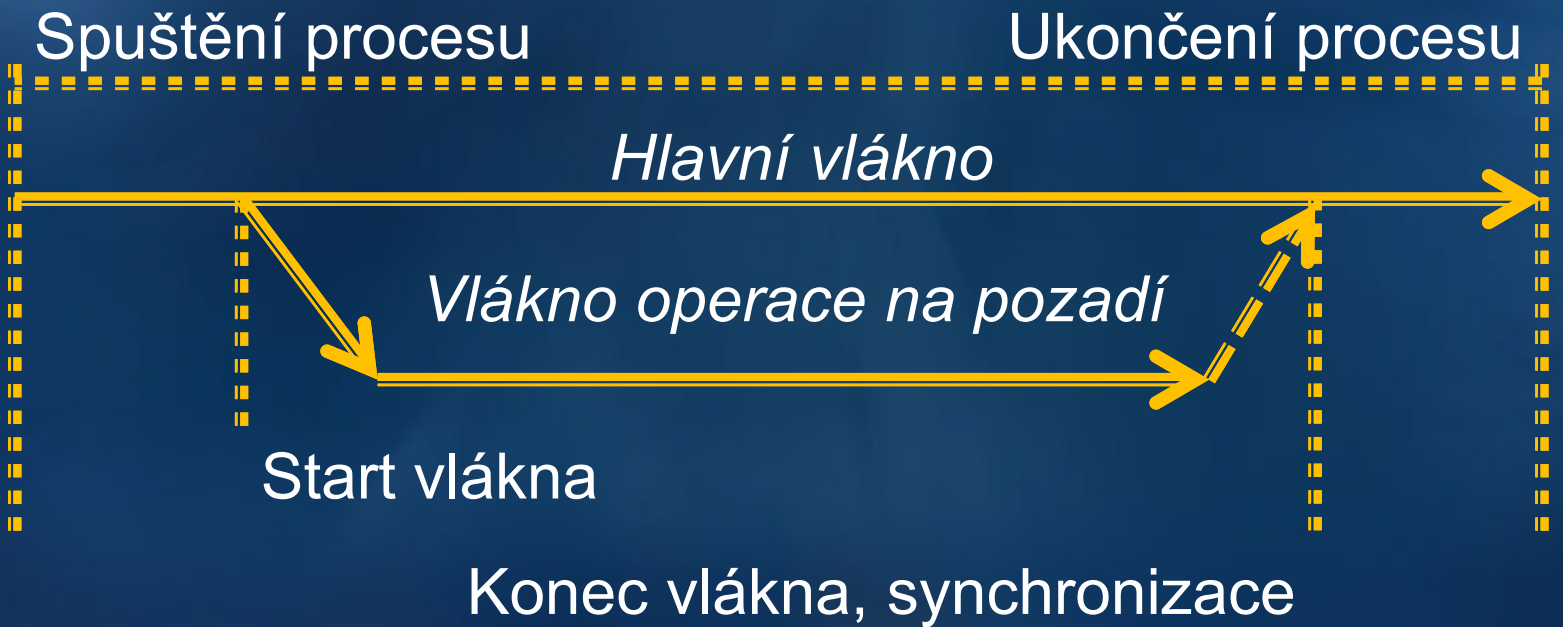
Nejčastější způsoby vzniku vlákna

- Startem aplikace
  - Vytváří se při startu procesu, končí jeho uzavřením
  - Například konzolová aplikace, WinForms / WPF aplikace
  - Řídí celý běh aplikace, označuje se jako „Main Thread“



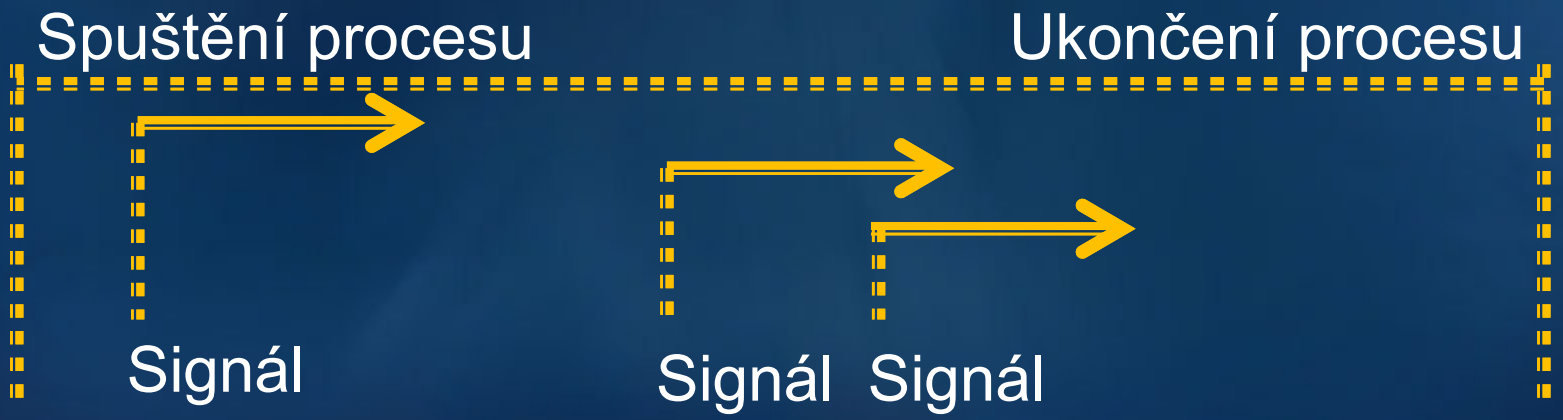
Nejčastější způsoby vzniku vlákna

- Operace na pozadí
  - Vlákno se vytváříme, když je potřeba a po dokončení se zruší
  - Vhodné na operace, které nesmí blokovat hlavní vlákno



Nejčastější způsoby vzniku vlákna

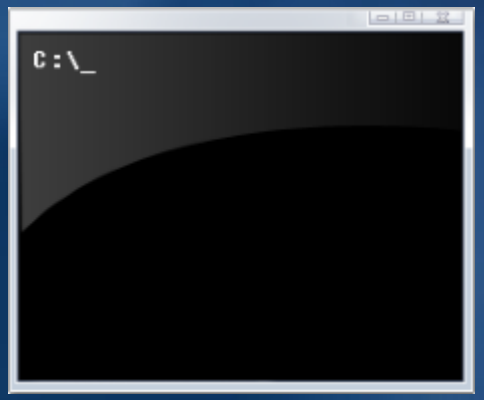
- Reakce na signál
  - Vlákno se vytváří jako reakce na konkrétní signál
  - Signál může být například:
    - Příchozí HTTP požadavek (každý požadavek = vlákno) – webová app.
    - Volání WCF služby
    - Vypršení Timeru – opakování úkolů ve Windows Service



Foreground / Background threads

- Při běhu aplikace se chovají zcela stejně
  - Stejná priorita
  - Stejné chování a stavy
- Jediný rozdíl – ukončení aplikace:
  - Aplikace sama končí, pokud jsou ukončena všechna Foreground Threads
  - Zbývající Background Threads se ukončí násilně (přerušení vyjímkou)
- Vlastnost **`IsBackgroundThread`**

Konzolová aplikace



- Nejjednodušší model
- Při startu aplikace:
  - Vytvoří a spustí se „Main Thread“
- Režim: Multi-threaded apartments
  - K třídě „Console“ můžeme přistupovat ze všech vláken
  - Blokace vlákna neblokuje celou aplikaci (čekání na vstup, otevření souboru atp.)

Konzolová aplikace
Vytvoření vlákna
Vyčkávání - Sleep
Rozdíl mezi foreground a background vlákny
Vyčkávání - Join

DEMO

Synchronizační postupy - **lock**

- Klauzule:
  lock(objekt)
  { *synchronizovaný kód* }
- Více vláken nemůže najednou vykonávat synchronizovaný kód

- Vnitřně využívá třídu **Monitor**:
  Monitor.Enter(objekt);
  try { *synchornizovaný kód* }
  finally { **Monitor.Exit**(objekt); }

Klauzule lock – synchronizace kritických částí

DEMO

Synchronizační postupy - Events

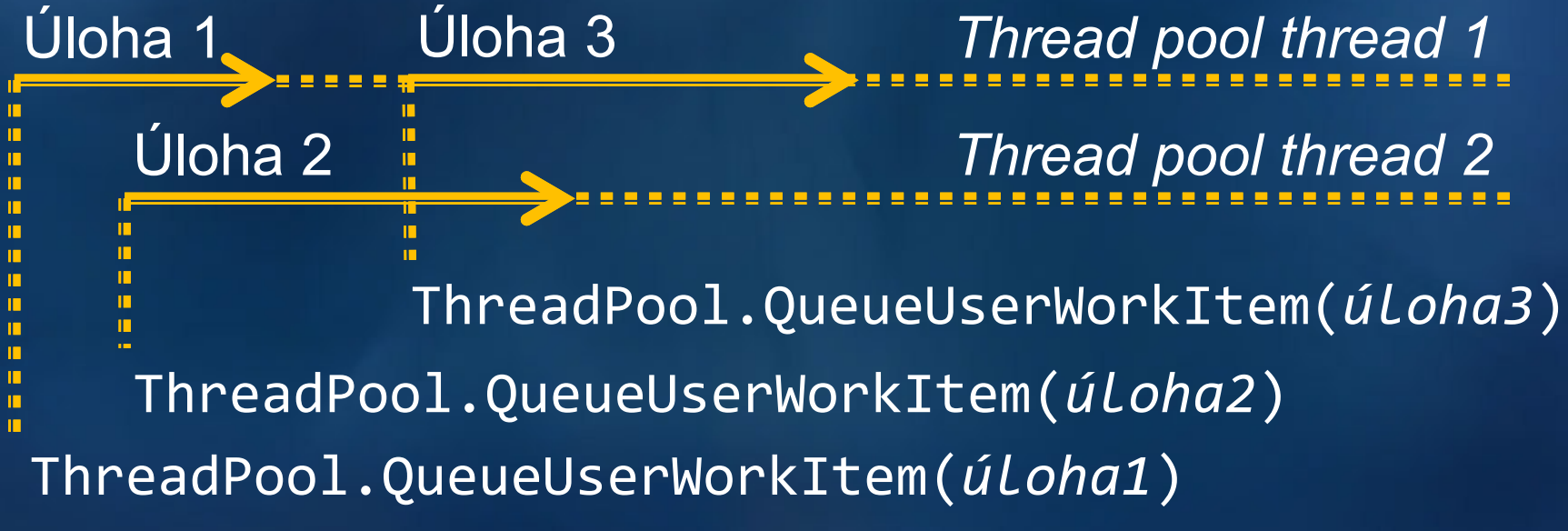
- Dědí z `EventWaitHandle`, existují:
  - **`ManualResetEvent`**
  - **`AutoResetEvent`**
- Základní příkazy:
  - **`Set`** - nastavuje stav na *True*
  - **`Reset`** * - nastavuje stav na *False*
  - **`WaitOne`** – čeká na stav *True*

Synchronizační události

DEMO

Thread Pool

- Poskytuje fond vláken, které je možné využít k provádění kódu
- Jen Background threads
- Recykluje ukončená vlákna
- Alokuje nová vlákna v případě nedostatku
- Ideální pro větší množství kratších operací



.NET Framework 4.0

TASK PARALLEL LIBRARY

Task Parallel Library

- Abstrakce nad vlákny

Task Parallelism

- Práce s objektem úlohy (Task)

Data Parallelism

- Paralelizace dotazů (PLINQ)

Task Parallel Library - Tasks

- Pracuje s „Task“ (!= vlákno)
- Do jisté míry podobné jako „Thread“
- Hlavní rozdíly:
  - Task podobně jako ThreadPool vlákna recykluje
  - Task nelze ukončit násilně
  - Task je vždy na pozadí
  - Task se nemusí ihned spustit (čekání při přetížení)
  - Task dokáže navazovat události na sebe
  - Task dokáže sledovat vzniklé vyjímky

Vytvoření a vyvolání Task
CancellationTokenSource
Sledování událostí „Parallel Tasks“

DEMO

Paralelizace dotazů (PLINQ)

- Nabízí LINQ to objects se zpracováním nad více vlákny
- Významné příkazy:
  - AsParallel
  - WithCancellation(*cancelToken*)
  - WithDegreeOfParallelism
  - WithExecutionMode
  - WithMergeOptions

Představení PLINQ

Nastavení chování dotazu

DEMO

.NET Framework 4.0

THREAD-SAFE COLLECTIONS

Thread-Safe Collections

- Namespace: `System.Collections.Concurrent`
- Třídy:
  - `BlockingCollection<T>`
  - `ConcurrentQueue<T>`
  - `ConcurrentStack<T>`
  - `ConcurrentBag<T>`
  - `ConcurrentDictionary<TKey,TValue>`
- Některé části realizovány pomocí: `SpinWait`; `SpinLock`

Thread-Safe Collections

- Namespace: `System.Collections.Concurrent`
- Třídy:
  - `BlockingCollection<T>`
  - `ConcurrentQueue<T>`
  - `ConcurrentStack<T>`
  - `ConcurrentBag<T>`
  - `ConcurrentDictionary<TKey,TValue>`

Thread-Safe Collections

- Některé části realizovány pomocí: `SpinWait; SpinLock`
- Zapouzdření častých operací:
  - `AddOrUpdate` – value / Func
  - `GetOrAdd` - value / Func
- Způsoby procházení:
  - Read-only enumeration
  - GetConsumingEnumerable
- Ne vždy budou vyhovovat, často je zapotřebí synchronizovat navazující operace

Návrhový vzor návrhu asynchronních komponent

EVENT-BASED ASYNCHRONOUS PATTERN

Synchronní operace

```
public class ProcessingClass
{

public int Foo();

}
```

Signatury podle event-based asynchronous patternu

```
public class ProcessingClass
{

public int Foo();
public void FooAsync();
public void FooAsync(object state);
public event
     FooCompletedEventHandler
     FooCompleted;
}
```

Event-based Asynchronous Pattern

- Slouží k zapouzdření asynchronních operací
- Jednotná signatura
- AsyncOperationManager
  - Řízení asynchronních operací
- AsyncOperation
  - Sledování stavu asynchronních operací
- Asynchronní operace může i nemusí mít synchronní variantu
- Zvážit implementaci:
  - Cancel, IsBusy, ProgressChanged

Event-based Asynchronous Pattern – implementace a použití

DEMO

SYNCHRONIZAČNÍ KONTEXT

Synchronizační kontext

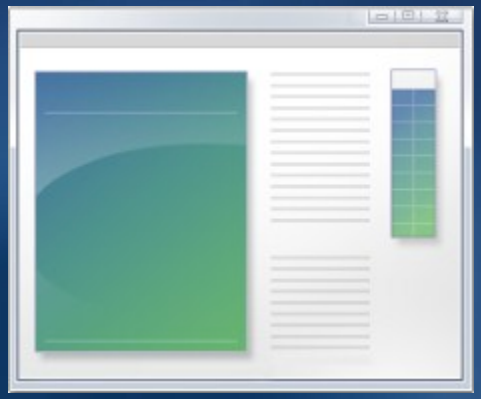
- Zajišťuje synchronizaci
- Každý typ aplikace jej implementuje jinak
  - SynchronizationContext
  - WindowsFormsSynchronizationContext
  - DispatcherSynchronizationContext
  - AspNetSynchronizationContext
- Send – synchronní odeslání
- Post – asynchronní odeslání
- Aktuální: `SynchronizationContext.Current`

Synchronizační kontext

- SynchronizationContext
  - Defaultní chování
  - Send – přímé volání, Post – ThreadPool
- WindowsFormsSynchronizationContext
  DispatcherSynchronizationContext
  - Synchronizace volání Send a Post do UI vlákna
- AspNetSynchronizationContext
  - Předávání HTTP context do dalších vláken, vyčkávání na dokončení async. operací

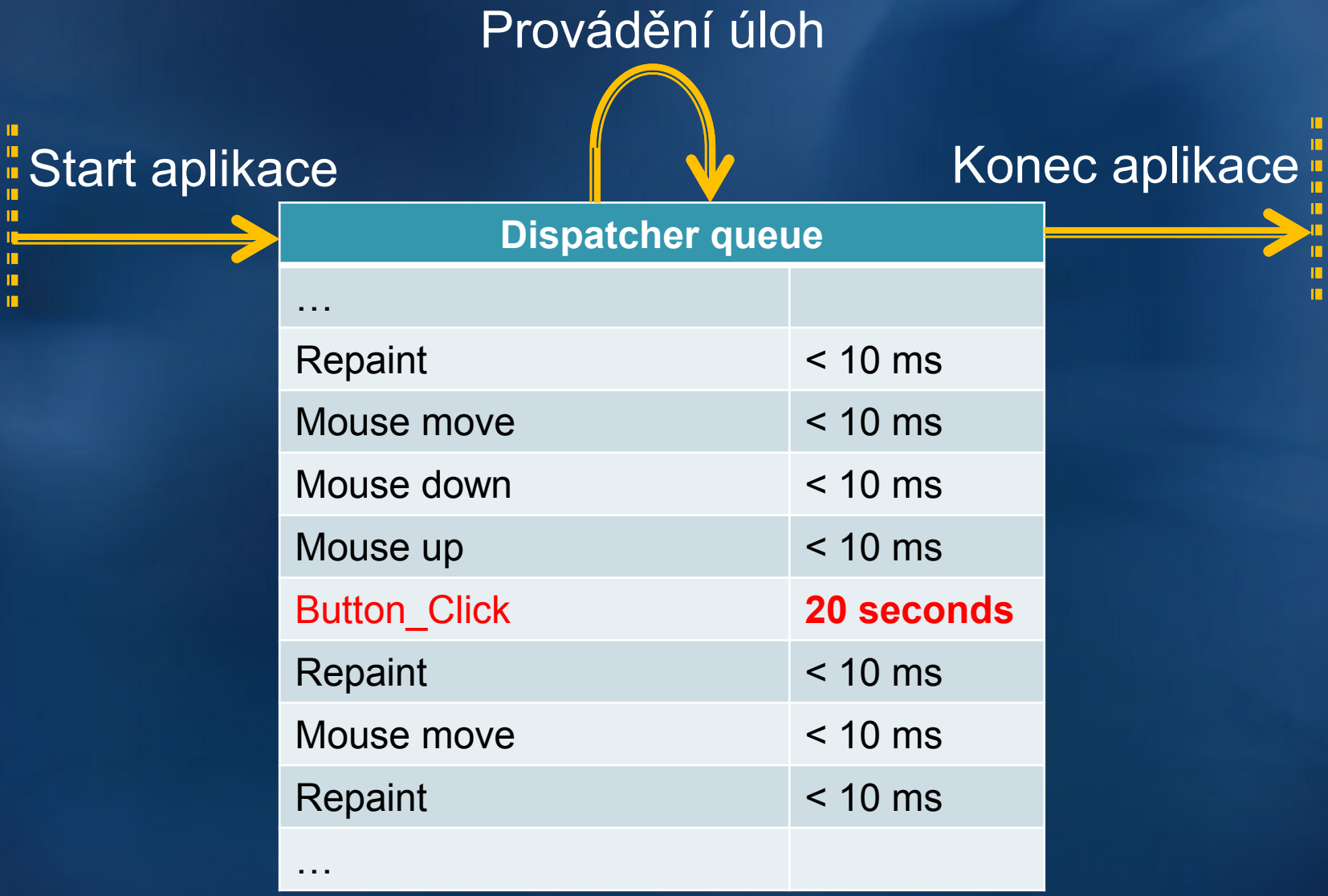
WPF APLIKACE

WPF aplikace



- Single-threaded apartments
  - *„Jedno vlákno vládne všem“*
  - Výhradně hlavní komunikuje s UI objekty
  - Delší operace řeší samostatná vlákna
- Hnací motor – `Dispatcher`
  - Synchronní fronta úloh
  - Na většinu úloh reaguje přímo WPF
    - Vykreslování
    - Reakce uživatelského prostředí
  - Ukončením hlavního vlákna aplikace končí
  - `DispatcherSynchronizationContext`

Hlavní vlákno - Dispatcher



Dispatcher queue	
…	
Repaint	< 10 ms
Mouse move	< 10 ms
Mouse down	< 10 ms
Mouse up	< 10 ms
Button_Click	**20 seconds**
Repaint	< 10 ms
Mouse move	< 10 ms
Repaint	< 10 ms
…	

Dispatcher

- Každý element uživatelského prostředí ve WPF obsahuje vlastnost **`Dispatcher`**
- **`Dispatcher.Invoke(…)`**
  - Přidá do seznamu úloh nové volání
  - Pro oznamování událostí do hlavního vlákna
- Ne všechny úlohy mají stejnou váhu
  - Nastudovat prioritu úloh

Asynchronní zpracování ve WPF

DEMO

ASP.NET APLIKACE

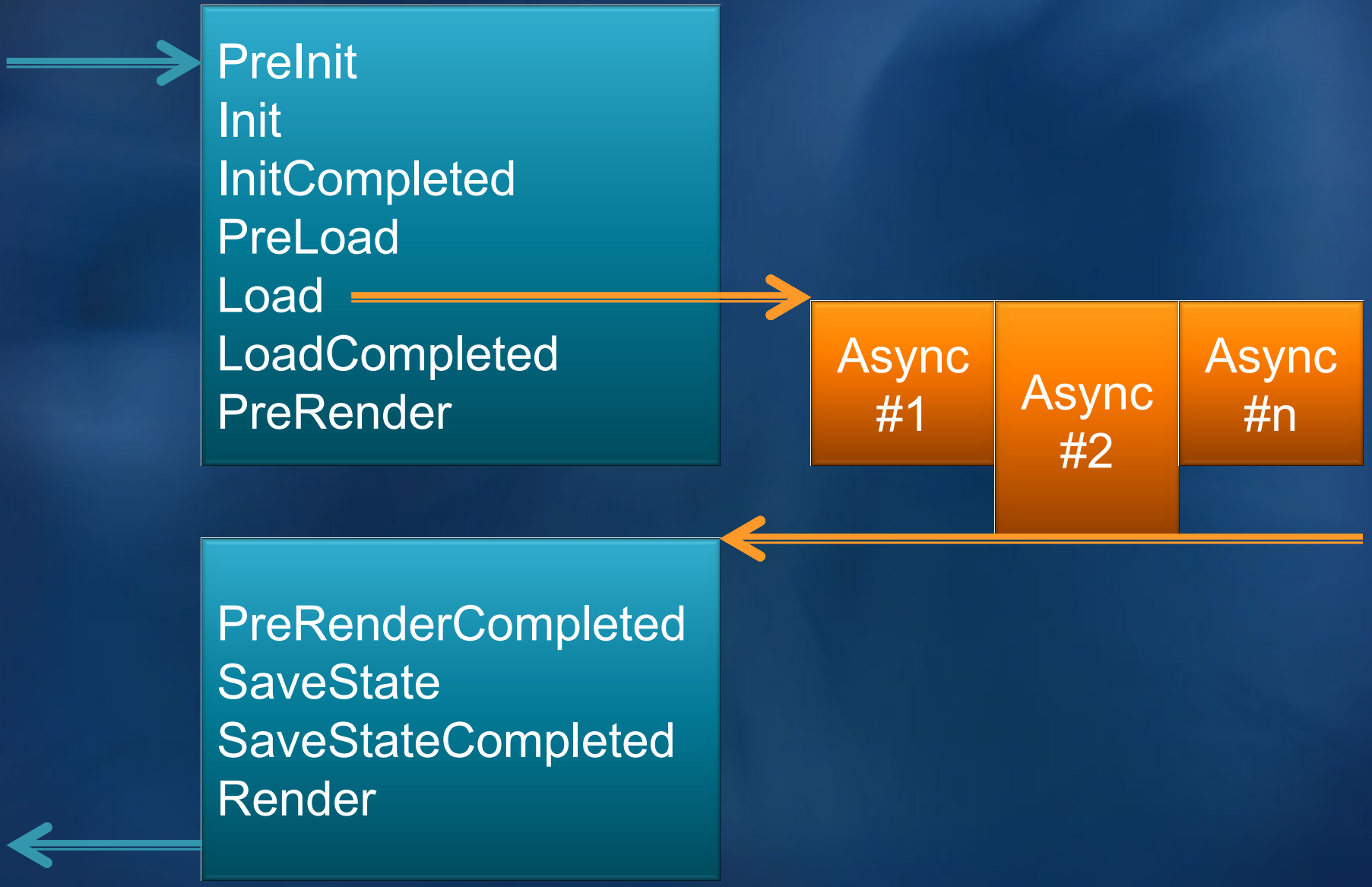
Webová aplikace

- 1 vlákno = 1 požadavek
  - Při synchronním zpracování
- Vlákna přiděluje **`ThreadPool`**
  - Podobně funguje *WCF* pro příchozí požadavky
- Není vhodné vytvářet vlákna pracující mimo požadavek
  - Recyklace
  - Start on demand

Automatická recyklace vláken

DEMO

Asynchronní volání v ASP.NET



Asynchronní volání v ASP.NET

- Lze vyvolávat asynchronní operace
- ASP.NET čeká před vykreslením stránky na všchny asynchronní operace
- Stránku obsluhují 2 thready
- Stránku je třeba označit `IsAsync="true"`

Vytvoření asynchronního volání při generování ASPX stránky

DEMO

.NET Framework 4.5

ASYNC / AWAIT

Synchronní kód

```
private byte[] DoSomeMagic()
{
      byte[] data;
      lblStatus.Text = "Získávám data...";
      data = LoadData();
      lblStatus.Text = "Rozbaluji...";
      data = UnpackData(data);
      lblStatus.Text = "Rozbaleno";
      return data2;
}
```

Asynchronní kód

```
private void DoSomeMagic()
{    lblStatus.Text = "Získávám data...";
     byte[] data;
     ThreadPool.QueueUserWorkItem((o) =>
     {    data = LoadData();
          context.Post((o2) => {
          lblStatus.Text = "Rozbaluji...";
          }, null);
          data = UnpackData(data);
          context.Post((o2) => {
          /* akce po dokončení */ }, null);
     }, null);
}
```

Asynchronní kód - async&await

```
private async Task<byte[]> DoSomeMagic()
{
      byte[] data;
      lblStatus.Text = "Získávám data...";
      data = await LoadData();
      lblStatus.Text = "Rozbaluji...";
      data = await UnpackData(data);
      lblStatus.Text = "Rozbaleno";
      return data;
}
```

async / await

- Novinka v .NET 4.5 (C# 5 i VB.NET 11)
- Nová klíčová slova: „async“ a „await“
- Psaní asynchronních volání
  - Eliminace callbacků
  - Čitelnější zápis
  - Automatická synchronizace
- Návratová hodnota je Task<*ReturnType*>
- Kompiler mění strukturu „async“ metody

Asynchronní kód - async&await

```
private async Task<byte[]> DoSomeMagic()
{
    byte[] data;
    lblStatus.Text = "Získávám data...";
    data = await LoadData();
    lblStatus.Text = "Rozbaluji...";
    data = await UnpackData(data);
    lblStatus.Text = "Rozbaluji...";
    return data;
}
```

Ihned synchronně

Thread Pool

Synchronization Context

Timery

`System.Threading.Timer`

- Využívá nativní funkce
- Události spouští pomocí `ThreadPool`

`System.Timers.Timer`

- Vnitřně `System.Thread.Timer`
- Vlastnost `SynchronizationObject`

`System.Windows.Forms.Timer`

- Pro Windows Forms (synchronizace do UI vlákna)

`System.Windows.Threading.DispatcherTimer`

- Pro WPF (synchronizace do UI vlákna)

Doporučení při multi-threading programování

- 2x více vláken != 2x rychlejší zpracování
  - Čas potřebný k synchronizaci
  - Ne všechny operace lze paralerilovat
- Co nejjednodušší konstrukce
- Obtížně ladění a reprodukce chyb
  - Promýšlet možné stavy paralelních operací, i když na testovacím stroji nenastávají
- Vymezit části, které přistupují k prostředkům z více vláken
- Naučit se ladit vlákna ve Visual Studiu

Tomáš Jecha
Mail: tomas@jecha.net | Twitter: @jechtom
http://www.jecha.net | http://www.vbnet.cz