# PV178 – Lab08

Karel Jiránek

# Agenda

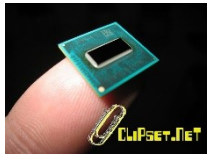
- Thread
- Task
- Locking


BRACE YOURSELF
THREADS ARE COMMING
imgflip.com

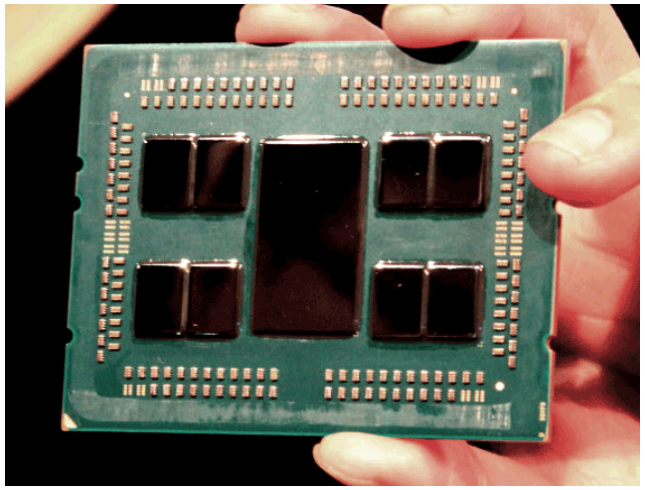
# Thread – HW background

- Procesor má více jader (1 jádro – 64 jader, stolní počítač 2-8 jader, PS5 – 8 jader)
- Grafická karta má stovky až tísíce jader (Nvidia RTX 3080 TI – 10240 jader)
- Jádro má 1-2 vláken (thread)
- Vlákno – nezávislý, paralelní výpočet

*Intel atom (1 core)*




*AMD Threadripper 3990x (64 core)*



# Thread (vlákno)

- Výpočetní vlákno
- Drahé – vytvoření trvá dlouho, prostředky
- Kód, který bude vlákno vykonávat – delegát v konstruktoru
- Běží nezávisle, paralelně (a co 2 vlákna na 1 core procesoru?)
- Schovává výjimky!
- Metody
    - Sleep – čekání X milisekund
    - Start – spuštění kódu vlákna
    - Join – spojení dvou vláken v jedno (zánik vlákna)

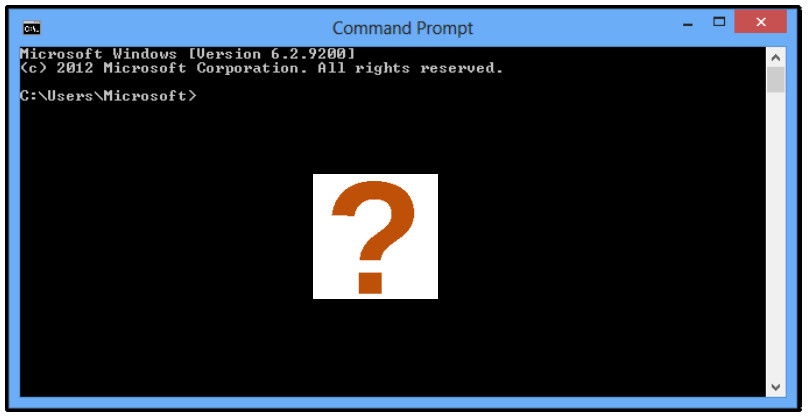
# Thread (vlákno) - příklad

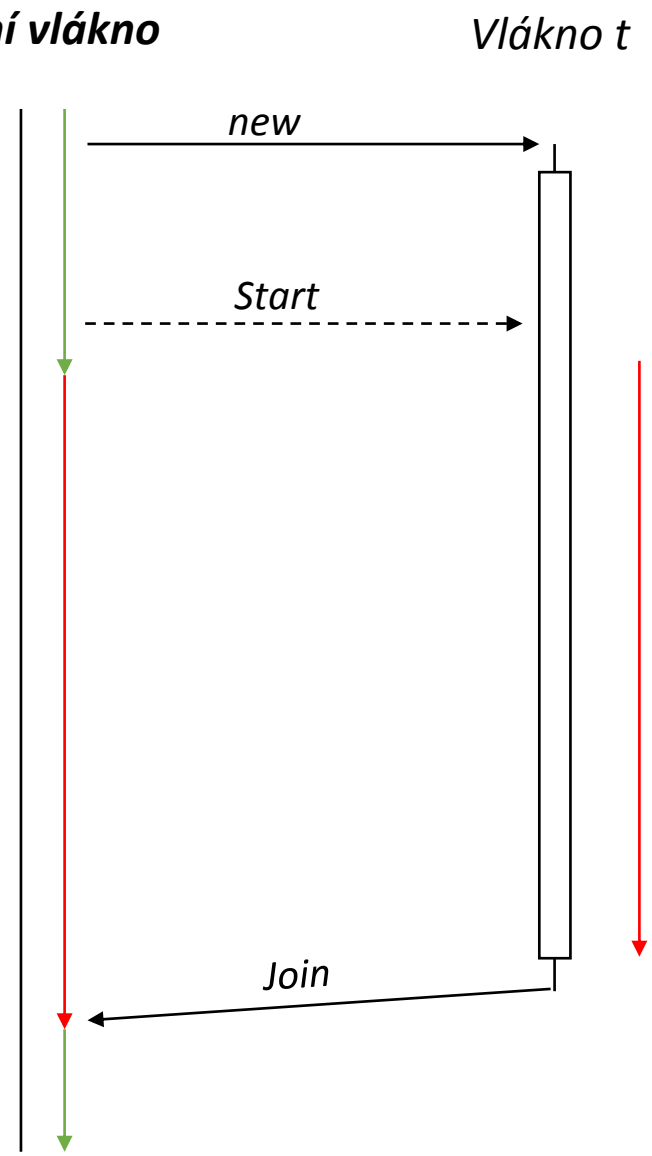
```
var t = new Thread(() => Console.Write("Thread t"))

t.Start()

Console.Write("Main thread"))

t.Join()
```



# **Foreground** vs. background vlákna

- Hlavní vlákno je foreground
- Background vlákno je ukončeno, když už neběží žádné foreground vlákno
- Background vlákno nelze klasicky spojit pomocí .Join() – „fire and forget“
- Background vlákno se používá ve velkých aplikacích
    - velmi těžké sledovat a spravovat všechna vlákna (10+)
    - běžící foreground vlákna by blokovala ukončení programu

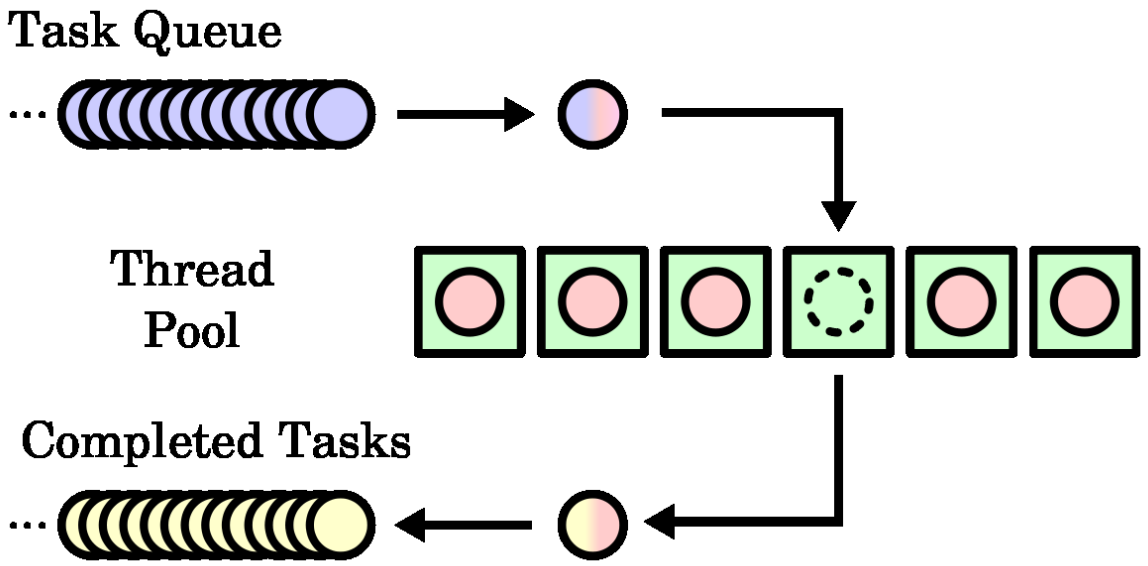
# Vlákna

# Samostatná práce

- Threads

# ThreadPool

- „Skladiště vláken”
- Efektivně spravuje vlákna - úlohy spouštěné na vláknech
- Příklad: `ThreadPool.QueueUserWorkItem(Console.WriteLine, "123");`



# Task

- Úspornější a efektivnější než Thread
- Neschovávají výjimky!
- X Tasků != X fyzických výpočetních vláken (více tasků může běžet na 1 fyzickém vláknu)
- Na ukončení Tasku se dá čekat synchronně nebo **asynchronně**
- Lze programově (hezky) ukončit pomocí tokenu
- Tasky lze řetězit – po tom skončí jeden, začne druhý (lze i podmíněně)

# Task – vytvoření, spuštění a čekání

- Vytvoření `var t = new Task(() => Console.Write("Task"))`
- Spuštění – `t.Start()`
- Vytvoření a spuštění – `Task.Run(… => …)`
- Synchronní čekání
  - Čekání na skončení 1 task: `.Wait()`, `.Result`
  - Čekání na skončení N tasků: `Task.WaitAll(tasks)`

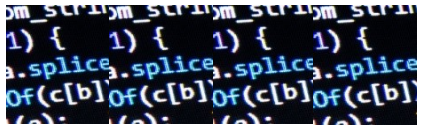
# Tasks

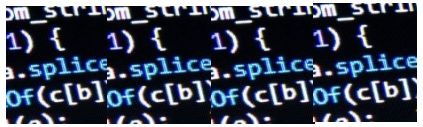
# Samostatná práce

- Tasks

# Task vs. Thread

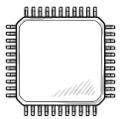
*kód*

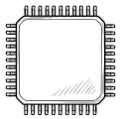


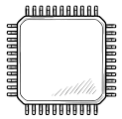


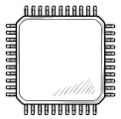
## *Thread*

*Core 1* *Core 2* *Core 3* *Core 4*





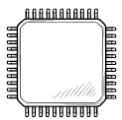


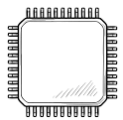


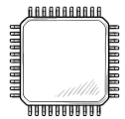
```
for (int i = 0; i < 4; i++) { new Thread(() => {...}).Start(); }
```

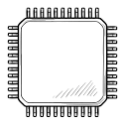
## *Task*

*Core 1* *Core 2* *Core 3* *Core 4*









```
for (int i = 0; i < 4; i++) { new Task(() => {...}).Start(); }
```

# Lock, mutex, sempahore

- `Lock` – klíčové slovo, řídí přístup ke sdíleným prostředkům, 0/1, **NENÍ** sdílen mezi systémovými procesy
- `Mutex` – třída, funkce jako lock, sdílen mezi systémovými procesy
- `Semaphore` – třída, funkce jako mutex, X vláken může vejít

# Samostatná práce

- Locking

# Concurrent Collections

- Normální kolekce nejsou vláknově bezpečné – Concurrent Collections ANO!
- `ConcurrentBag` – thread safe `Collection`
- `ConcurrentDictionary` – thread safe `Dictionary`
- `ConcurrentQueue` - thread safe `Queue`

# Blocking Collections

- Wrapper pro thread safe kolekce
- Operace jsou blokovány dokud je není možné vykonat
  - Take – blokuje program dokud nelze odebrat prvek
  - Add – blokuje dokud nelze přidat prvek

```
new BlockingCollection<int>(new ConcurrentBag<int>(), 10)
```

*Max kapacita*

# Na doma

- Projít si příklady na ConcurrentCollections v Lab08_Solution
- Udělat samostatnou práci  ConcurrentCollections