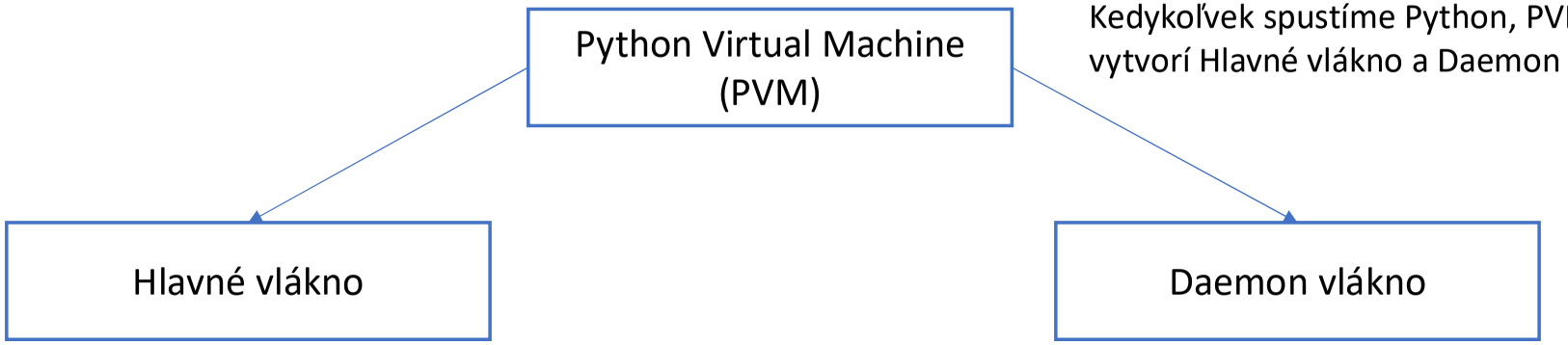
# Daemon vlákna vs. Worker vlákna

- Vlákno v Pythone môže byť nazvané Daemon vlákno (Daemon thread), alebo **bežné**, **Worker**, **vlákno** (Worker thread)
- Keď sa spúšťa Python program, 1 vlákno beží okamžite (hlavné vlákno – Main thread), ktoré inicializuje aplikáciu


- Vieme vytvoriť dcérske vlákna z hlavného vlákna, ktoré sa nazývajú aj ako **aplikačné vlákna**. Hlavné vlákno (Main thread) je posledným vláknom pri skončení behu programu, pretože vykonáva rôzne ukončovacie operácie
- **Daemon vlákna** sú určené ako pomocné vlákna (napr. pre **Garbage collection** proces)
- **Worker vlákna sú** dcérskymi vláknami, ktoré vznikli z Hlavného vlákna (Main thread)

# Daemon vlákna vs. Worker vlákna




Kedykoľvek spustíme Python, PVM vytvorí Hlavné vlákno a Daemon vlákna

Tu vieme vytvoriť ľubovoľný počet Worker vlákien (dcérske vlákna hlavného vlákna)

Vytvorí sa niekoľko Daemon vlákien, napr. pre Garbage collection

# Daemon vlákna vs. Worker vlákna

- **Daemon vlákna** sú vlákna menšej priority, ktoré **bežia na po?adí** vykonať úlohy ako je napr. Garbage collection
- Zvyčajne tvoríme daemon vlákna pre input/output operácie, služby ako sú: NFC, Bluetooth komunikácia, alebo aplikáciu vyhľadávajúca smart hodinky v smartfónoch

- Daemon vlákna sú ukončené PVM, keď všetky Worker vlákna ukončili operácie
- Zároveň platí: Worker vlákna nie sú ukončené, pokiaľ daemon vlákna sú prerušené PVM
- Daemon vlákno sa v Pythone inicializuje parametrom daemon=True

```python
def normal_operation():
    for i in range(1000):
        print("Bežné vlákno je spustené")

def daemon_operation():
    while True:
        print("Dameon vlákno je spustené")

t1 = thr.Thread(target=normal_operation, name="Normal thread")

# Deamon vlakno ukonci operaciu, ked bezne vlakna ukoncia operaciu
# Po 1000 iteraciah bezneho vlakna, Python prerusi / ukonci Daemnon vlakno
# aj napriek tomu, ak je v nekonecnej slucke, a skonci sa program
t2 = thr.Thread(target=daemon_operation, name="Daemon thread", daemon=True)

t1.start()
t2.start()
```

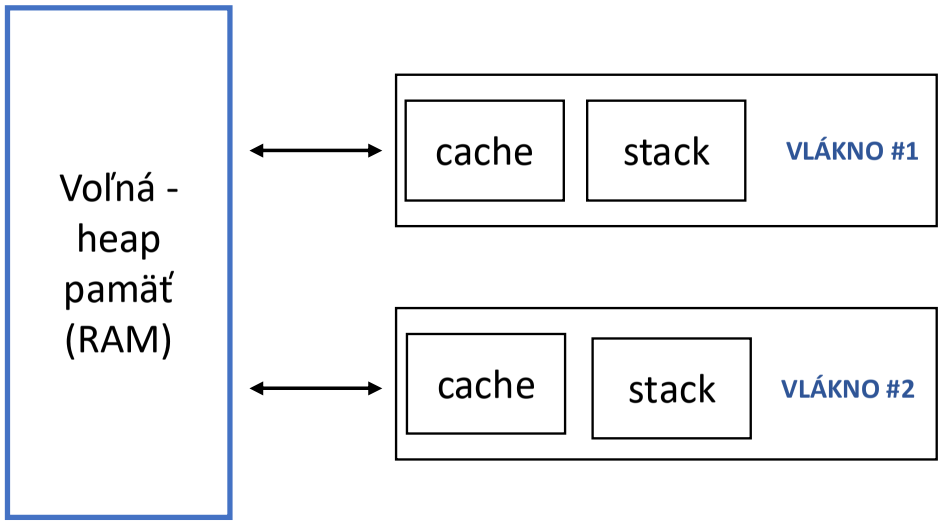
# Manažment pamäte

- Sú **procesy** a **vlákna** (odľahčené procesy)
- **Dôležitý** rozdiel medzi procesom a vláknom je, vlákna (rovnakého procesu) bežia v zdieľanej pamäti, zatiaľ čo procesy bežia v separátnej pamäti

- **Zásobníková pamäť** (stack memory): menšia, rýchlejšia, uložené lokálne premenné, argumenty metód, volania funkcií
- **Voľná pamäť** (heap memory/RAM): väčšia, pomalšia, sú v nej uložené objekty tak dlho, pokiaľ existuje na ne referencia odkiaľkoľvek z aplikácie

- Každé vlákno má vlastnú zásobníkovú pamäť, pričom všetky vlákna zdieľajú spoločnú voľnú pamäť (heap memory)
- **Synchroni?ácia** – jej hlavný zmysel je zdieľanie prostriedkov (zdrojov) bez vzájomného ovplyvňovania sa

# Manažment pamäte




Vlákno #1 a #2 má vlastnú stack a cache pamäť, ale spoločnú voľnú pamäť (heap memory)

# Manažment pamäte

- Z **príkladu** by sme očakávali atomickosť operácie s výsledkom 2 milióny, čo sa ale nestane
- V skutočnosti prebiehajú 2 pod-operácie (načítanie premennej z pamäte s inkrementáciou, a zápis hodnoty do pamäte)
- **Čo sa stane:**
  - Vlákno 1 zoberie hodnotu premennej z pamäte, inkrementuje +1
  - **Zároveň**, v rovnakom čase, vlákno 2 zoberie hodnotu premennej z pamäte inkrementuje +1
  - Hodnota premennej je rovná 1. Táto hodnota sa zapíše do pamäte.

```python
x = 0
def increment():
    global x
    x = x + 1

def operation():
    for i in range(1000000):
        increment()

t1 = thr.Thread(target=operation, name="Thread #1")
t2 = thr.Thread(target=operation, name="Thread #2")

t1.start()
t2.start()

# Pockame, kym operacie skoncia
t1.join()
t2.join()

# x = 2-miliony ?
# Nie, x = 1460120
print(f"Operácie skončili, x je rovné {x}")
```

# Synchronizácia - zámky


- **Synchroni?ácia** zabezpečí, že žiadne 2 vlákna spustia rovnaký blok kódu (tzv. kritickú oblasť kódu)
- **Kritická oblasť** je časť programu, v ktorom sú sprístupňované zdieľané zdroje
- Tzv. **race condition (súbeh)** sa vyskytuje, pokiaľ aspoň **2** vlákna pristupujú k zdieľaným (rovnakým) zdrojom
- Veľmi ťažké debugovať programy s race condition (určité obdobie fungujú správne, neskôr nastane nekonzistencia)
- **Synchroni?ácia** a **?ámky** vedia pracovať s **race condition** problémom
- Použitím zámku (Lock) alebo **mutex**, vlákno #1 získa zámok, uzamkne operáciu v kritickej oblasti. Ostatné vlákna čakajúce na operáciu, sú blokované. Akonáhle je operácia dokončená, vlákno zámok okamžite uvoľní, čím môže iné vlákno získať zámok a zamknúť prístup ku kritickej sekcii pre ostatné vlákna. Ak súčasne viac vlákien chce získať zámok, je zaručené, **vždy len 1** vlákno získa zámok
- Výsledok predošlého príkladu, použitím **?ámku** (Lock), vráti 2 milióny

```
from threading import Lock
x = 0

# Iba 1 vlakno moze ziskat zamok v rovnakom case
# Ked zamok ziska, ine vlakno musi cakat, kym bude znova zamok dostupny
lock = Lock()
def increment():
    global x
    # Toto je kriticka sekcia programu
    # Vlakna sa mozu dostat do blokovaneho stavu
    # Musia cakat, kym sa zamok opat neuvolni
    lock.acquire()
    x = x + 1
    lock.release()

def operation():
    for i in range(1000000):
        increment()

t1 = thr.Thread(target=operation, name="Thread #1")
t2 = thr.Thread(target=operation, name="Thread #2")
```

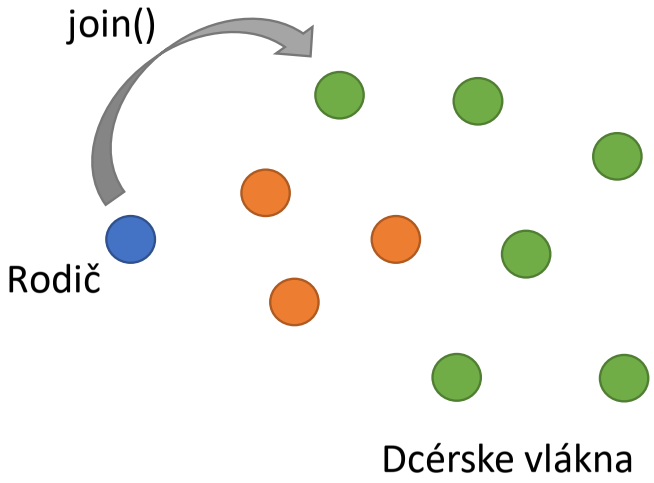
# Synchronizácia - zámky

- V Pythone môžeme použiť buď štandardnú triedu **threading.Lock**, alebo **threading.RLock**, ktorá je tzv. zámkom opätovného prístupu
- štandardný zámok (**Lock**) je možné získať v reálnom čase **ra**, následne sa musí uvoľniť. Uvoľnený môže byť akýmkoľvek vláknom
- na druhej strane vieme v reálnom čase získať zámok **RLocks** niekoľkokrát pre rovnaké vlákno, bez ohľadu na to, že už bol získaný
- **RLock** musí byť vrátený toľkokrát, koľkokrát bol získaný
- **Rlock** zámky môžu byť uvoľnené len vláknom, ktoré ich získalo
- Všeobecne platí, že vlákno nemôže získať zámok vlastnený iným vláknom. Ale dané vlákno môže získať zámok, ktorý už vlastní. Umožnenie vlákna získať rovnaký zámok viackrát sa nazýva **opätovná synchroni ácia účastníka**. A to je presne to, čo sa deje v **Pythone** v triede **RLocks** - rovnaké vlákno môže získať zámok viackrát, ak ho už v danom momente vlastní
- opätovnou synchronizáciou účastníka sa v aplikácii **vyhneme deadlockom**

```
from threading import Lock
#from threading import RLock

lock = Lock()
#lock = RLock()

# Standardny Lock nevie ziskat znova zamok sam od seba
# Rlock ano
lock.acquire()
#lock.acquire()

print("Vypísaný text na konzolu")

lock.release()
lock.release()
```

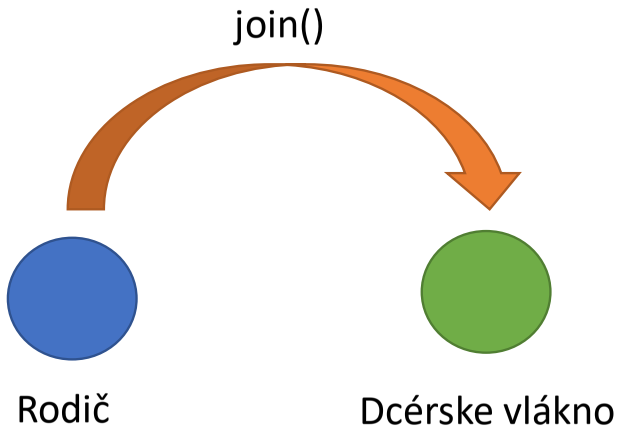
# Synchronizácia - join()

- join() je funkcia, kde vlákno #1 volá join() na vlákno #2, #3, #4, ... a čaká, kým dané vlákno nedokončí svoju úlohu
- join() je ďalšou súčasťou synchronizácie vlákien
- Rodičovské vlákno volá dcérske vlákno na vykonanie operácie
- Kým dcérske vlákno nedokončí operáciu, rodičovské vlákno čaká (je blokované)
- Keď dcérske vlákno operáciu ukončí, join sa okamžite uvoľní, rodičovské vlákno je odblokované
- Pri úlohách, kde pracuje viac vlákien nad 1 úlohou použijeme rodičovské vlákno, ktoré volá dcérske vlákna. Rodičovské vlákno vždy čaká na dokončenie operácie každého dcérskeho vlákna
- Rodič volá dcérske vlákno #1, ostatné dcérske vlákna a rodič čakajú, dcéra #1 končí úlohu, rodič sa zobudí a volá dcérske vlákno #2. Vlákno #2 dokončí úlohu, rodič sa zobudí a volá dcérske vlákno #3, ...
- Volaním veľkého počtu vlákien z hlavného vlákna, ktoré sa majú spustiť, vieme dosiahnuť použitím cykla while alebo for



# Synchronizácia - join()

```python
import time
from threading import Thread

def child():
    print("Dcérske vlákno pracuje...")
    # Funkcia sleep() uspi dcerske vlakno na 5 sekund
    time.sleep(5)
    print("Dcérske vlakno skončilo...")


def parent():
    t = Thread(target=child, args=([]))
    t.start()
    print("Rodičovské vlákno čaká...")
    # Funkcia join() s volitelnym parametrom sec., po n sekundach
    # sa rodicovske vlakno odblokuje a viac nebude cakat na
    # dokoncenie operacie vlakna dcerskeho
    t.join(2)
    # V tomto priklade bude po 2 sek. rodicovske vlakno odblokovane
    print("Rodičovské vlákno je odblokované...")


parent()
```




**Výstupom je:**
Dcérske vlákno pracuje...
Rodičovské vlákno čaká...
Rodičovské vlákno je odblokované...
Dcérske vlákno skončilo...