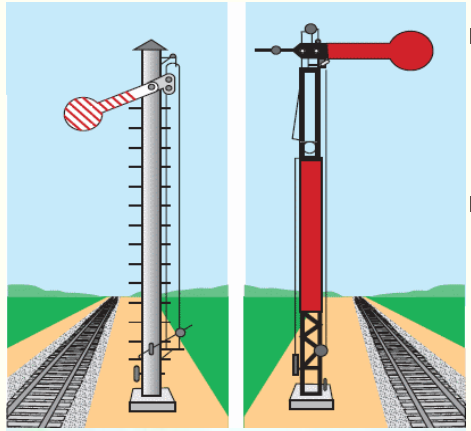
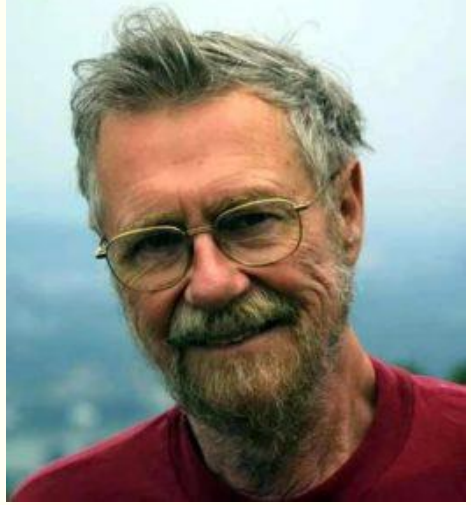
ОСНОВНІ ПОНЯТТЯ БАГАТОПОТОЧНОГО ВИКОНАННЯ КОДУ

Питання 12.1.

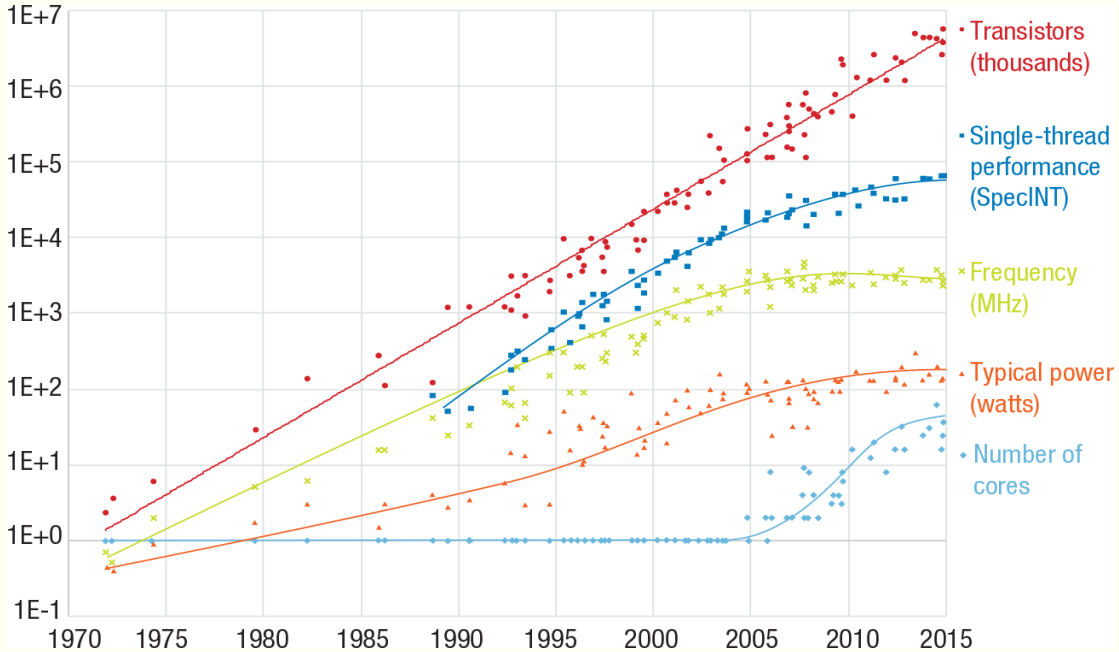
Одночасне виконання (concurrency) vs розпаралелювання (parallelism)



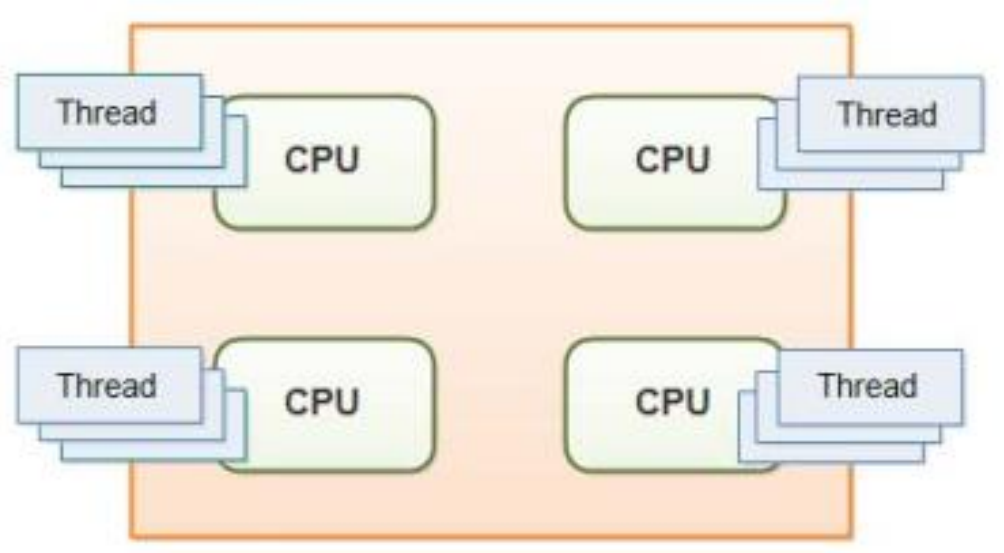


- Термін з'явився в контексті залізної дороги та телеграфу.
  - Потрібно було на одних рейках запускати багато потягів так, щоб вони безпечно пересувались.
- У рамках ***багатопоточних обчислень*** термін почав використовувати Е. Дейкстра.
  - Ввів базові поняття *семафорів, взаємного виключення* (mutual exclusion) та *дедлоків* (deadlock)

Тренди розвитку мікропроцесорів

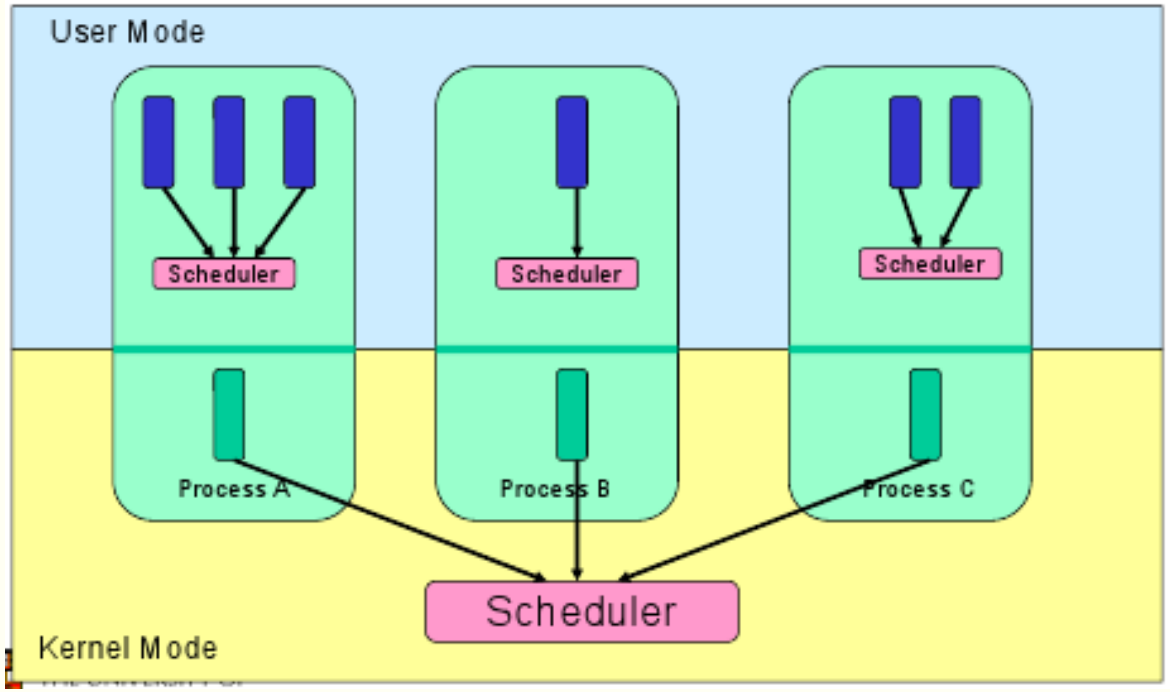


Потоки (thread) і багатопоточність (multithreading)



- ***Потік (thread)*** – впорядкований потік інструкцій, виконання яких може плануватись операційною системою.
  - Потоки зазвичай живуть всередині процесів та складаються з програмного лічильника (program counter), стеку, набору регістрів та ідентифікатору.
  - Потоки в даному розумінні – найменші одиниці виконання, для яких процесор може виділяти час.
- Потоки можу взаємодіяти зі спільними ресурсами, а також з іншими потоками.
- Проблема: коли два потоки володіють спільною пам'яттю (shared memory), а порядок їх виконання не гарантований, можуть з'являтись неточності та помилки.
  - Основна причина – стан гонитви даних (data race).

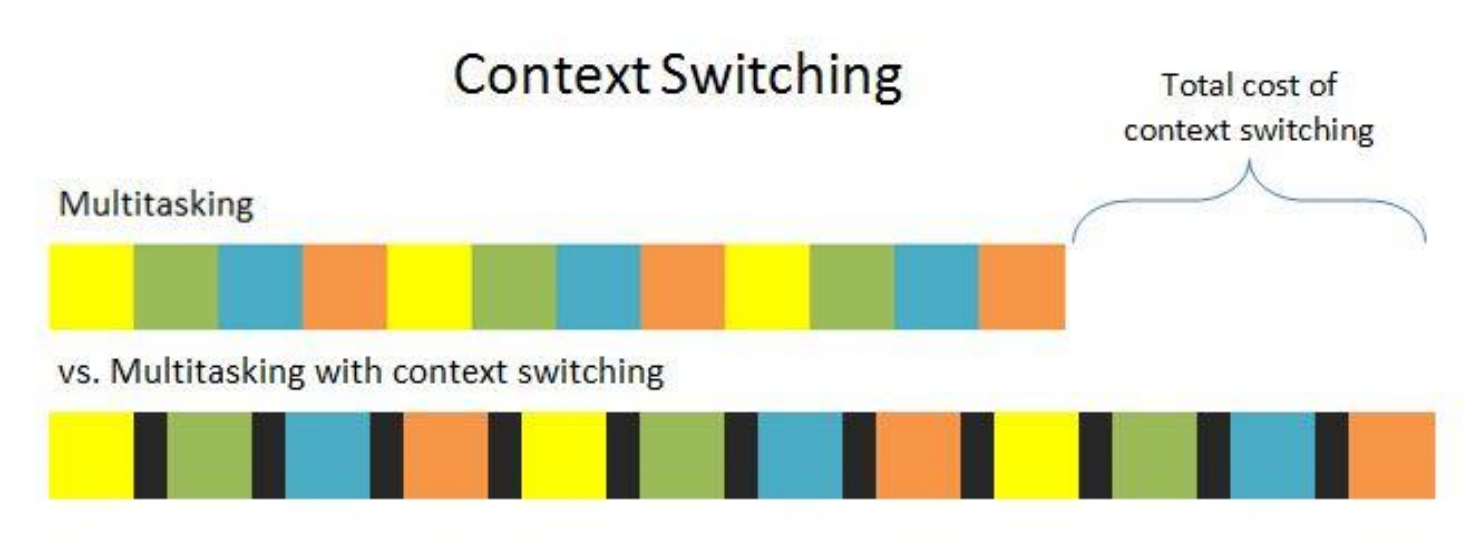
Види потоків



- **Потоки рівня користувача (User-level threads):** можуть активно створюватись, запускатись та вбиватись для різних задач.
  - Python працює з потоками рівня користувача
- **Потоки рівня ядра (Kernel-level threads):** дуже низькорівневі потоки, що працюють з операційною системою.

Поняття багатопоточності (multithreading)

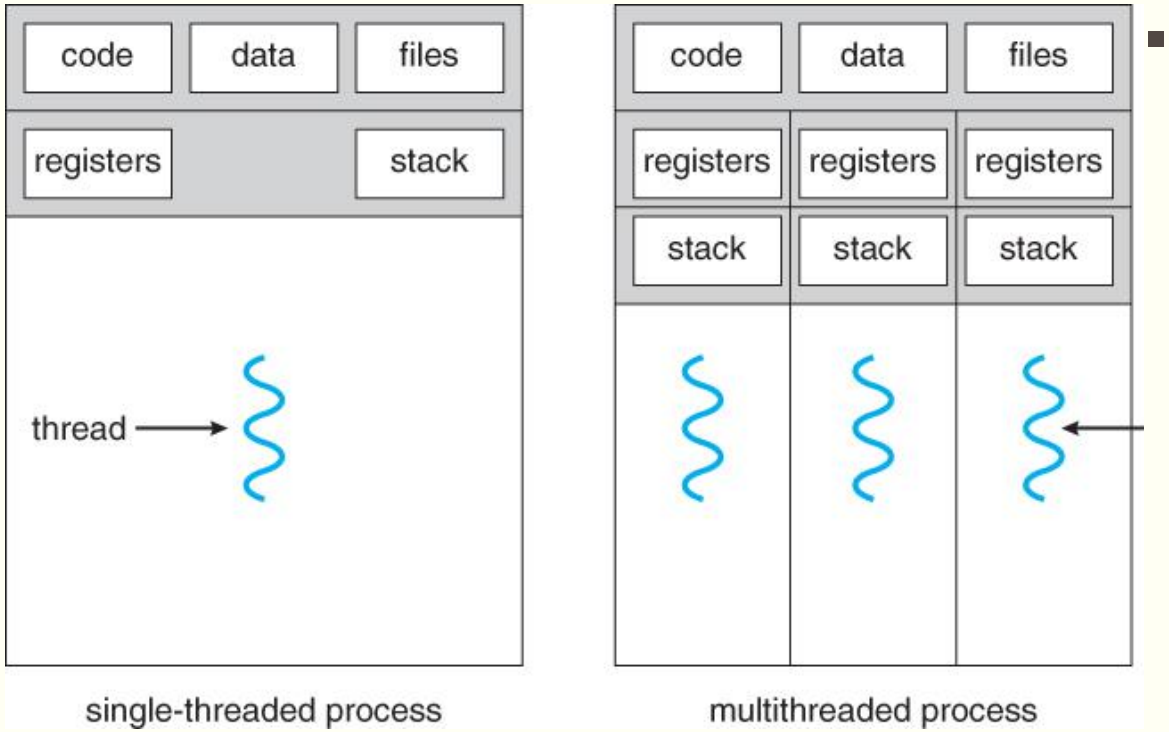
- Зазвичай під багатопоточним процесором розуміють процесор, що вміє виконувати одночасно багато потоків.
  - Окреме ядро здатне дуже швидко ***перемикати контекст*** (switch context) між кількома потоками.
  - Перемикання контексту відбувається через настільки малі проміжки часу, що здається паралельне виконання потоків (хоча це не так).



Переваги та недоліки тредингу

- Переваги
  - Багато потоків чудово підходять для пришвидшення blocking I/O bound програм
  - Потоки легковагові з точки зору використання пам’яті в порівнянні з процесами.
  - Завдяки спільним ресурсам комунікація між потоками спрощується.
- Недоліки
  - Потоки CPython обмежуються global interpreter lock (GIL).
  - Хоч комунікація між потоками спрощується, необхідно дуже обережно реалізувати код, щоб не виникало станів гонитви.
  - Перемикати контекст між багатьма потоками дуже затратно обчислювально. Додавання їх може призвести до зниження загальної продуктивності роботи коду.

Процеси



- За своєю природою дуже схожі на потоки.
  - Проте не прив'язані до конкретного ядра процесора.
  - Можуть працювати з кількома речами в певний момент
  - Містять лише 1 потік – головний (main, primary), проте можуть створювати субпотоки (sub-thread)

За допомогою процесів можна підвищити швидкість виконання програм в деяких сценаріях, коли додатки чутливі до кількості ядер.
Проте, створюючи багато процесів, стикаємось з проблемою міжпроцесної комунікації (**interprocess communication** (**IPC**)), на яку може піти багато часу.

Властивості процесів

- UNIX-процеси створюються операційною системою і зазвичай містять:
  - Process ID, process group ID, user ID, group ID
  - Середовище (Environment)
  - Робочу папку (Working directory)
  - Програмні інструкції (Program instructions)
  - Регістри (Registers)
  - Стек (Stack)
  - Кучу (Heap)
  - Дескриптори файлів (File descriptors)
  - Signal actions
  - Спільні бібліотеки (Shared libraries)
  - Інструменти міжпроцесної взаємодії (Inter-process communication tools: message queues, pipes, semaphores, or shared memory)

Переваги та недоліки процесів

- Переваги:
  - Процеси можуть покращити використання багатоядерних процесорів.
  - Вони краще за багато потоків при виконанні CPU-intensive задач
  - Можна обійти обмеження GIL, породжуючи багато процесів
  - Падаючі процеси не уб’ють програму в цілому
- Недоліки:
  - Немає спільних ресурсів між процесами – необхідно реалізувати в деякій формі міжпроцесну взаємодію (IPC)
  - Потребують більше пам’яті

Багатопроцесна обробка (Multiprocessing)

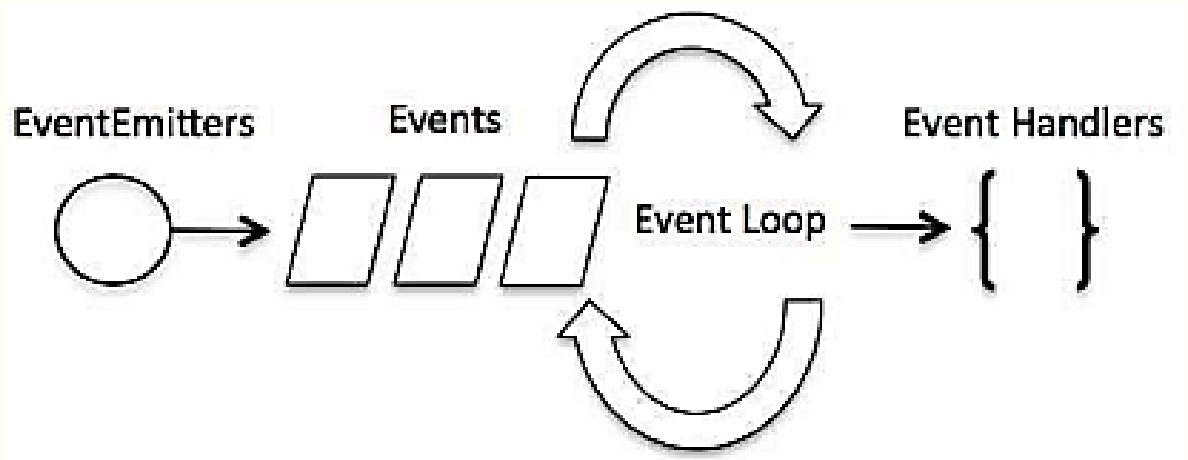
- У Python можна обрати, запускати код, використовуючи багато потоків чи багато процесів.
- Для виводу кількості ядер процесора

```
1  import multiprocessing as mp
2  print(mp.cpu_count())
```

  - multiprocessing дає змогу задіяти більше машинних ресурсів
  - Також уникаються обмеження Global Interpreter Lock в Cython
- Недоліки роботи з багатьма процесами:
  - Немає спільного стану (shared state)
  - Нестача комунікації

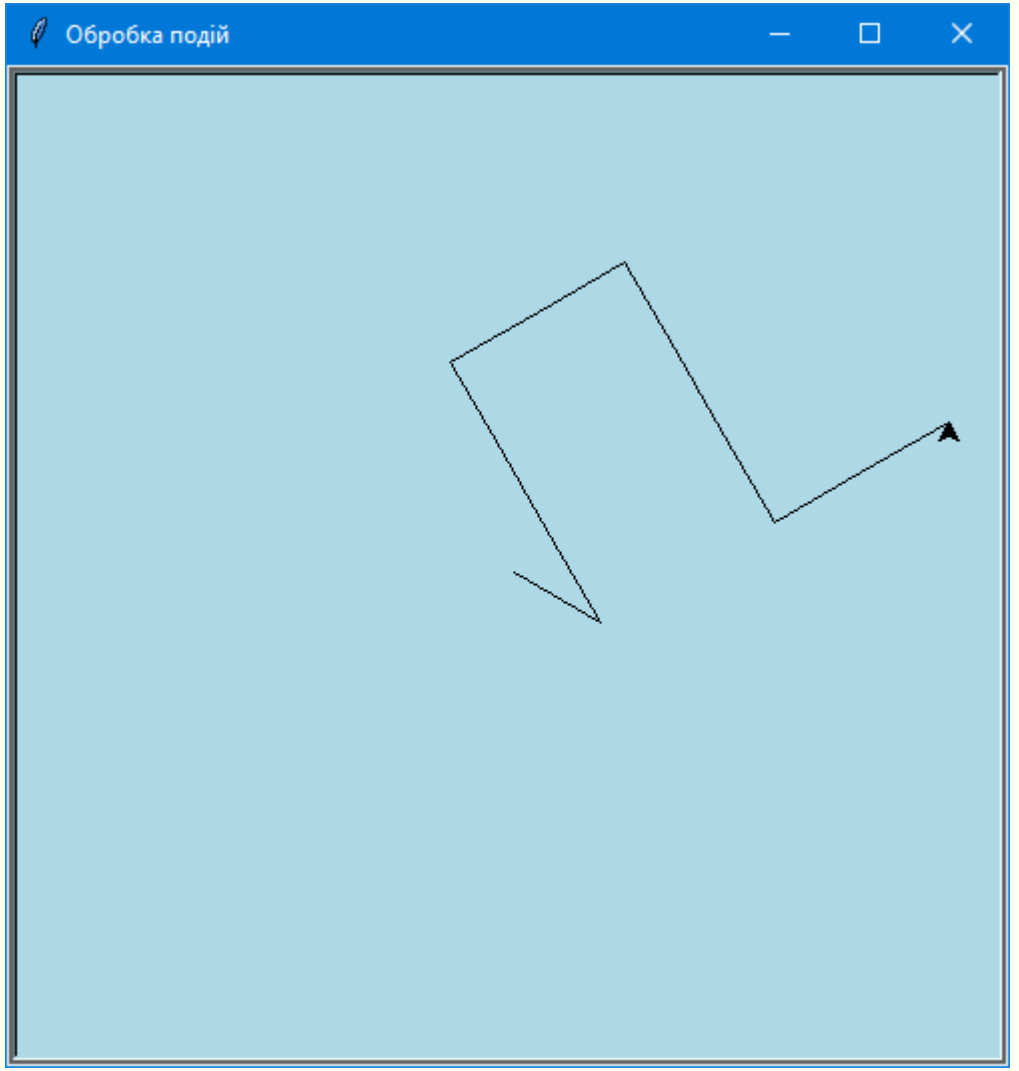
Подійно-орієнтоване програмування

- Кожна здійснена взаємодія може розглядатись як подія (event) чи їх набір, що зазвичай викликає методи зворотного відгуку (callbacks).
  - EventEmitters запускають багато подій, які розглядаються в циклі подій (Event Loop),
  - Подія ставиться у відповідність її обробнику (Event Handler)



Обробка подій у вигляді черепашки

```
1 import turtle
2 turtle.setup(500, 500)
3 window = turtle.Screen()
4 window.title("Обробка подій")
5 window.bgcolor("lightblue")
6 nathan = turtle.Turtle()
7
8 def moveForward():
9   nathan.forward(50)
10
11 def moveLeft():
12   nathan.left(30)
13
14 def moveRight():
15   nathan.right(30)
16
17 def start():
18   window.onkey(moveForward, "Up")
19   window.onkey(moveLeft, "Left")
20   window.onkey(moveRight, "Right")
21   window.listen()
22   window.mainloop()
23
24 start()
```



Реактивне програмування

```
(C:\ProgramData\Anaconda3) C:\Users\spuasson>pip install rx
Collecting rx
  Downloading Rx-1.6.0-py2.py3-none-any.whl (178kB)
    100% |████████████████████████████████| 184kB 2.3MB/s
Installing collected packages: rx
Successfully installed rx-1.6.0
```

- Дуже схоже на подійно-орієнтоване програмування, проте ***фокусується на даних***, а не подіях.
  - *Реагує на зміни в потоці даних* (data stream).
  - Комбінує паттерн Спостерігач (observer), патттерн Ітератор та функціональне програмування.
  - Ми *підписуємось (subscribe) на різні потоки вхідних даних*, а потім *створюємо спостерігачів (observer), які прослуховують (listen) появу конкретної події*.
  - Після спрацювання (trigger) спостерігача запускається код, який відповідає спостерігачу.
- Уявіть датацентр з тисячами серверних стійок, які постійно обчислюють мільйони операцій.
  - Одна з найскладніших проблем – достатнє охолодження стійок, щоб не було пошкоджень через перегрів.
  - Можна встановити багато термометрів по датацентру, щоб перевіряти нагрівання стійок та відправляти показники на центральний комп'ютер неперервним потоком (stream):

```python
import rx
from rx import Observable, Observer

class temperatureObserver(Observer):
    def on_next(self, x):
        print("Температура: %s градусів" % x)
        if (x > 6):
            print("Обережно: температура перевищує рекомендовану межу!")
        if (x == 9):
            print("Датацентр перериває роботу. Температура надто висока")

    def on_error(self, e):
        print("Помилка: %s" % e)

    def on_completed(self):
        print("Всі температури зчитані")

xs = Observable.from_iterable(range(10))
d = xs.subscribe(temperatureObserver())
```

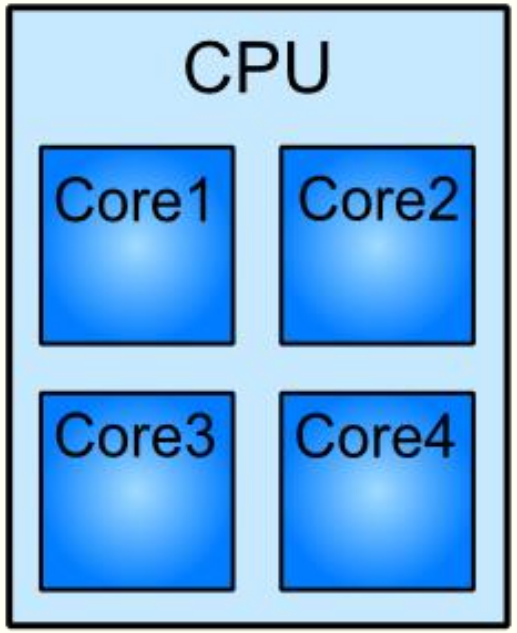
Приклад роботи – датацентр

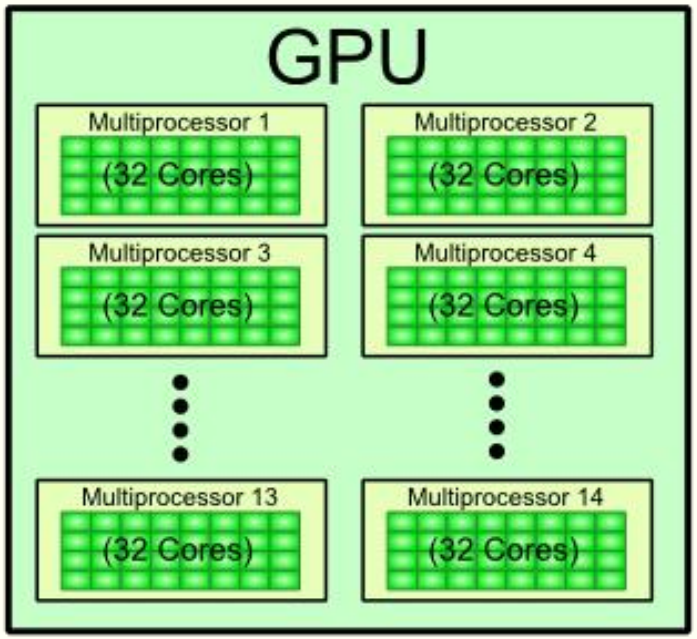
- Клас містить 3 функції:
  - ***on_next:*** викликається кожного разу, коли спостерігач помітить щось нове
  - ***on_error:*** спрацьовує як функція обробки помилок; викликається кожного разу при спостережeіганні помилки.
  - ***on_completed:*** викликається, коли спостерігач стикається з кінцем потоку даних.

- У функції on_next() хочемо вивести поточну температуру, а також перевірити, чи отримана температура знаходиться в допустимих межах.
  - Якщо температурне значення відповідає одній з умов, дещо змінюємо його обробку та виводимо опис помилок, що стались.
  - Після оголошення класу продовжуємо створювати fake observable, який містить 10 окремих значень, за допомогою Observable.from_iterable().
  - Останній рядок коду підписує новий екземпляр temperatureObserver на даного спостерігача.

```
In [1]: runfile('C:/Users/spuasson/04_.py', wdir='C:/Users/spuasson')
Температура: 0 градусів
Температура: 1 градусів
Температура: 2 градусів
Температура: 3 градусів
Температура: 4 градусів
Температура: 5 градусів
Температура: 6 градусів
Температура: 7 градусів
Обережно: температура перевищує рекомендовану межу!
Температура: 8 градусів
Обережно: температура перевищує рекомендовану межу!
Температура: 9 градусів
Обережно: температура перевищує рекомендовану межу!
Датацентр перериває роботу. Температура надто висока
Всі температури зчитані
```

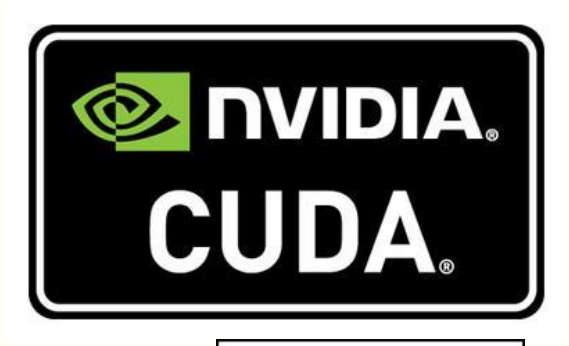
GPU-програмування





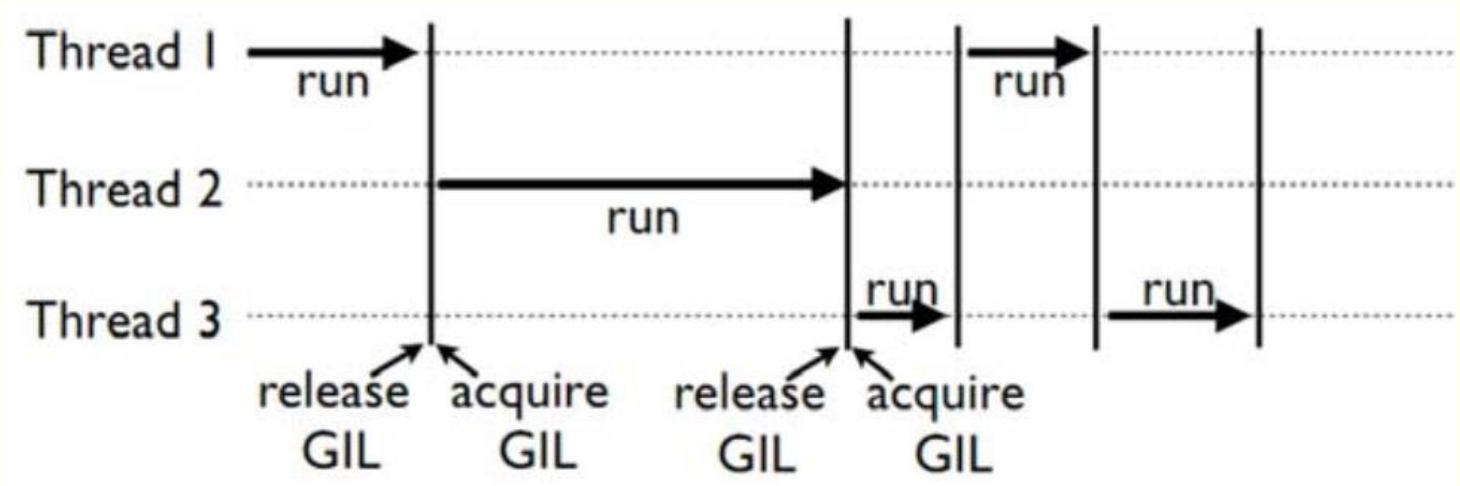
- Хоч GPU мають велику потужність при графічних обчисленнях, вони спроектовані не для забезпечення роботи операційної системи та обчислень загального призначення.
  - CPU мають менше ядер, які спеціально спроектовані у тому числі для швидкого перемикання контексту між задачами.

GPU-програмування

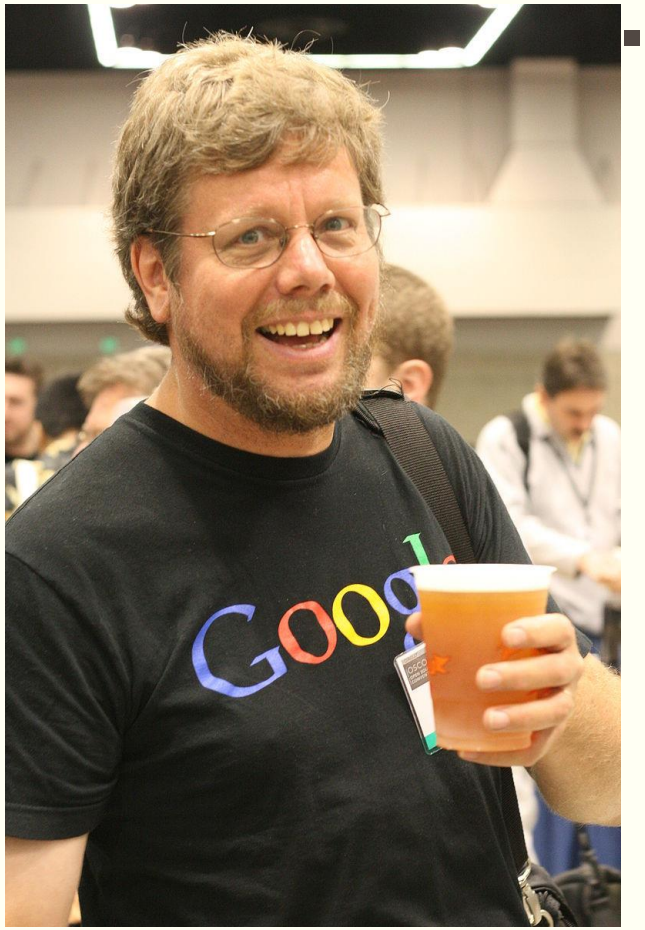


- Бібліотеки, які допомагають абстрагуватись від низькорівневого коду графічних API для задіювання GPU в обчисленнях.
  - PyCUDA дозволяє взаємодіяти з Nvidia CUDA API для паралельних обчислень у Python.
  - OpenCL дозволяє виконувати гетерогенні обчислення (на CPU, GPU, DSP, FPGA та інших видах процесорів та апаратних прискорювачів)
  - Theano – бібліотека, яка дозволяє використовувати GPU, щоб отримувати швидкості, аналогічні С-реалізаціям при розв'язуванні задач, які включають величезні набори даних.

Обмеження Python. Global Interpreter Lock



- GIL – це замок взаємного виключення, який не дає кільком потокам паралельно виконувати Python-код.
  - Цей замок утримується лише одним потоком в будь-який момент часу, тому при потребі потоку виконувати власний код спочатку потік має отримати замок.
  - Кожен потік повинен очікувати та здобувати (acquire) GIL перед тим, як зможе продовжити роботу, а потім звільняти (release) GIL, зазвичай до того, як робота завершена.
  - Дотримується ***випадкової циклічної диспетчеризації (random round-robin)***, гарантій того, який потік першим здобуде замок, немає.



- Guido Van Rossum не проти створення GIL-less вітки мови Python, якщо це не вплине негативно на продуктивність однопоточних додатків.
  - На практиці введення додаткових типів блокувань для забезпечення безпеки потоків сповільнює додаток у понад два рази.
  - Проте є бібліотеки, наприклад NumPy, які можуть виконувати всі свої операції не взаємодіючи з GIL

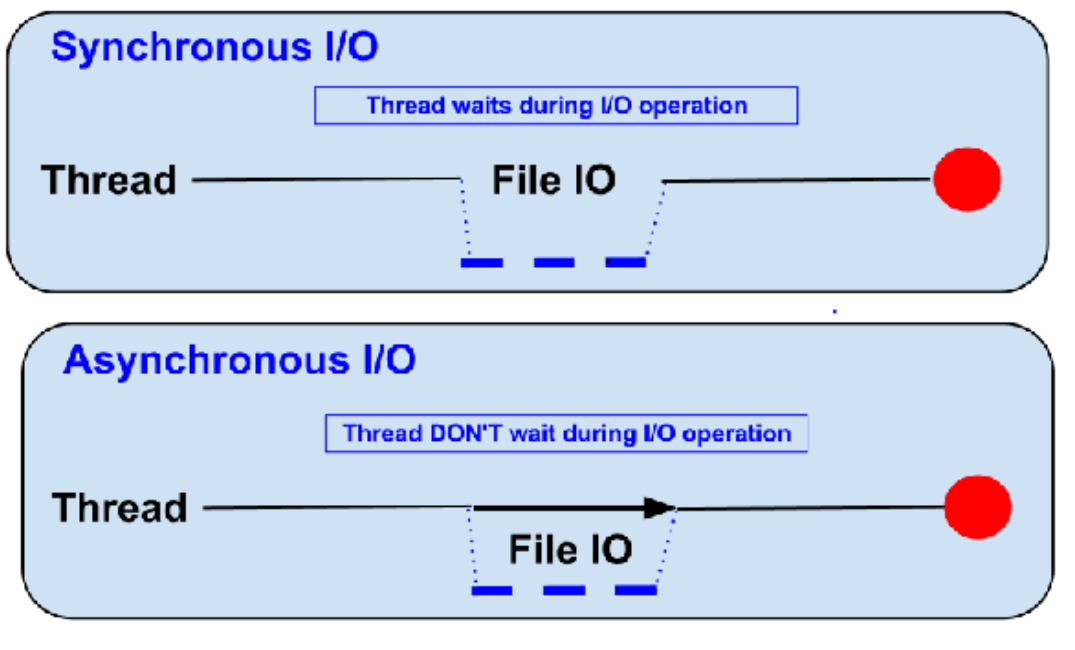
Еквіваленти без GIL

- Jython – реалізація Python, яка працює напряму з платформою Java.
  - Може доповнювати Java в якості скриптової мови для прискорення роботи додатків на рівні Cpython при роботі з великими датасетами.
  - Для більшості випадків одноядерне та багатоядерне виконання коду за допомогою CPython 's single-core execution typically outperforms Jython and its multicore approach.

- IronPython - .NET-еквівалент Jython, працює поверх Microsoft .NET framework.

Приклад: багатопоточне завантаження зображення

- Чудовий приклад для демонстрації багатопоточності у зв'язку з блокуючою (синхронною) природою вводу-виводу.
  - Завантажимо зображення з сайту 10 разів та збережемо у 10 файлів.

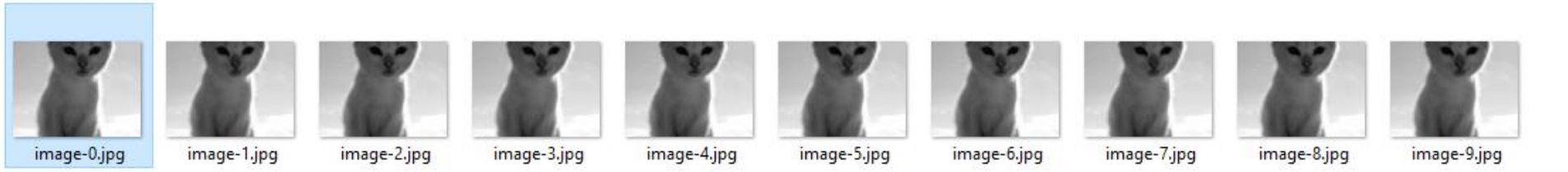


Послідовне завантаження

```python
import urllib.request
import time

def downloadImage(imagePath, fileName):
  print("Downloading Image from ", imagePath)
  urllib.request.urlretrieve(imagePath, fileName)

def main():
  t0 = time.time()
  for i in range(10):
    imageName = "temp/image-" + str(i) + ".jpg"
    downloadImage("http://placekitten.com/g/160/120", imageName)
    print(imageName)

  t1 = time.time()
  totalTime = t1 - t0
  print("Total Execution Time {}".format(totalTime))

if __name__ == '__main__':
  main()
```

Вивід програми

```
Downloading Image from  http://placekitten.com/g/160/120
temp/image-0.jpg
Downloading Image from  http://placekitten.com/g/160/120
temp/image-1.jpg
Downloading Image from  http://placekitten.com/g/160/120
temp/image-2.jpg
Downloading Image from  http://placekitten.com/g/160/120
temp/image-3.jpg
Downloading Image from  http://placekitten.com/g/160/120
temp/image-4.jpg
Downloading Image from  http://placekitten.com/g/160/120
temp/image-5.jpg
Downloading Image from  http://placekitten.com/g/160/120
temp/image-6.jpg
Downloading Image from  http://placekitten.com/g/160/120
temp/image-7.jpg
Downloading Image from  http://placekitten.com/g/160/120
temp/image-8.jpg
Downloading Image from  http://placekitten.com/g/160/120
temp/image-9.jpg
Total Execution Time 4.126962661743164
```



Багатопоточне завантаження

```python
import threading
import urllib.request
import time

def downloadImage(imagePath, fileName):
  print("Downloading Image from ", imagePath)
  urllib.request.urlretrieve(imagePath, fileName)
  print("Completed Download")

def executeThread(i):
  imageName = "temp/image-" + str(i) + ".jpg"
  downloadImage("http://placekitten.com/g/160/120", imageName)
  print(imageName)

def main():
  t0 = time.time()
  # масив, який зберігатиме посилання на всі ваші потоки
  threads = []

  # створимо 10 потоків, додамо їх у масив потоків та запустимо
  for i in range(10):
    thread = threading.Thread(target=executeThread, args=(i,))
    threads.append(thread)
    thread.start()

  # перевірка завершення роботи всіх потоків з масиву
  for i in threads:
    i.join()

  # обчислення загального часу виконання
  t1 = time.time()
  totalTime = t1 - t0
  print("Total Execution Time {}".format(totalTime))

if __name__ == '__main__':
  main()
```

Вивід програми

```
Downloading Image from  http://placekitten.com/g/160/120
Downloading Image from Downloading Image from Downloading Image from   Downloading Image from Downloading Image from
Downloading Image from Downloading Image from  http://placekitten.com/g/160/120http://placekitten.com/g/160/120    Downloading
Image from Downloading Image from http://placekitten.com/g/160/120

http://placekitten.com/g/160/120http://placekitten.com/g/160/120http://placekitten.com/g/160/120http://placekitten.com/g/
160/120
http://placekitten.com/g/160/120http://placekitten.com/g/160/120



Completed Download
temp/image-0.jpg
Completed Download
temp/image-2.jpg
Completed Download
temp/image-1.jpg
Completed Download
temp/image-5.jpg
Completed Download
temp/image-7.jpg
Completed Download
temp/image-3.jpg
Completed Download
temp/image-4.jpg
Completed Download
temp/image-8.jpg
Completed Download
temp/image-9.jpg
Completed Download
Total Execution Time 0.9979584217071533temp/image-6.jpg
```

Властивості багатопоточних систем

- ***Багато дійових сутностей (Multiple actors):***
  - Різні процеси та потоки активно намагаються просунутись у виконанні своїх задач.
  - Можемо мати багато процесів, кожен з яких міститиме по кілька потоків, які намагаються працювати одночасно.
- ***Спільні ресурси (Shared resources):***
  - Представляють пам’ять чи інші ресурси, які сутності використовують, щоб виконати свою роботу.
- ***Правила (Rules):***
  - Конкурентні системи повинні слідувати набору строгих правил, що визначають, коли сутності можуть чи не можуть отримувати замок, доступ до пам’яті, змінювати стан тощо.

Вузькі місця (bottlenecks) вводу-виводу

- Комп'ютер може більше часу чекати на різні операції вводу-виводу, ніж виконувати їх.
  - Зазвичай така ситуація в heavy додатках з підтримкою вводу-виводу.
  - У браузері очікуємо на запити(requests) з мережі довше, ніж їх обробляємо (застосовуємо код з файлів html/css/js).
  - Потрібно або покращити швидкість підсистеми вводу-виводу (купити дороге апаратне забезпечення), або покращити обробку самих запитів на ввід-вивід.
- Розглянемо приклад: ***пошуковий павук (web crawler)***.
  - Мета – обхід (traverse) вебу та індексація сторінок, щоб Google брав їх до уваги під час пошуку.

```
import urllib.request
import time
t0 = time.time()
req = urllib.request.urlopen('http://www.example.com')
pageHtml = req.read()
t1 = time.time()
print("Total Time To Fetch Page: {} Seconds".format(t1-t0))
```

Ускладнимо код: будемо переходити за посиланнями, знайденими на веб-сторінках, щоб проіндексувати їх також

- Можна використовувати бібліотеку BeautifulSoup

```
1 import urllib.request
2 import time
3 from bs4 import BeautifulSoup
4
5 t0 = time.time()
6 req = urllib.request.urlopen('http://www.example.com')
7 t1 = time.time()
8 print("Total Time To Fetch Page: {} Seconds".format(t1-t0))
9 soup = BeautifulSoup(req.read(), "html.parser")
10
11 for link in soup.find_all('a'):
12   print(link.get('href'))
13
14
15 t2 = time.time()
16 print("Total Execeution Time: {} Seconds".format)
```

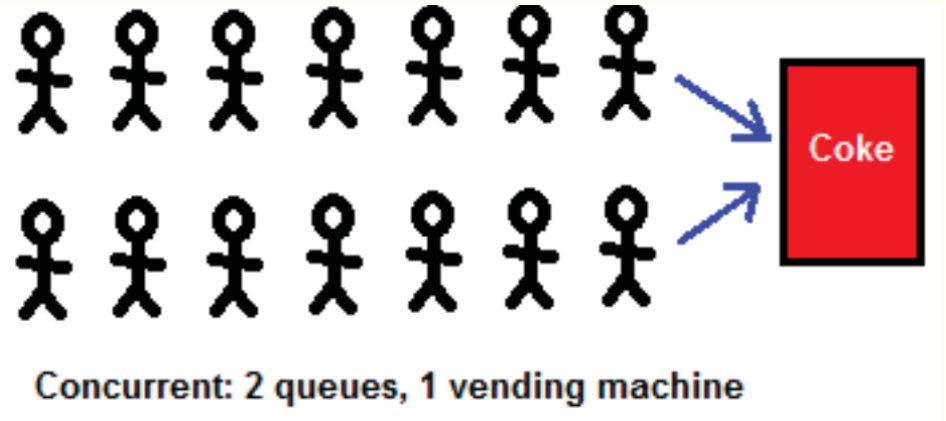
Вивід програми

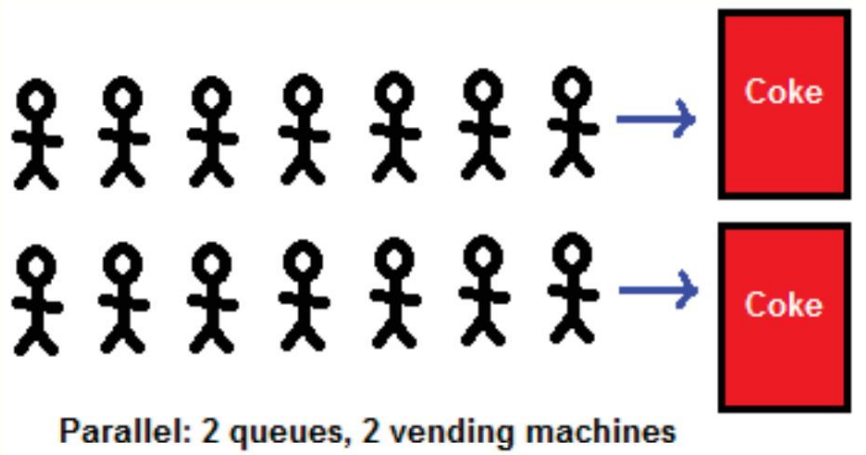
```
Total Time To Fetch Page: 0.26125025749206543 Seconds
http://www.iana.org/domains/example
<built-in method format of str object at 0x00000205920A2CE8>
```

- Час на отримання (fetch) сторінки понад чверть секунди.
  - Павуки мають працювати з мільйонами веб-сторінок, загальний час роботи збільшиться відповідно.
- Головна причина сповільнення – вузьке місце вводу-виводу в програмі.
  - Дуже багато часу чекаємо на мережеві запити, а парсинг отриманої сторінки відбувається швидше за її одержання.

Розуміння паралелізму

- ***Паралелізм*** – виконання двох або більше дій одночасно.
  - На відміну від багатопоточності (concurrency), в якій відбувається просування двох або більше things одночасно.
  - Для справжнього паралелізму потрібні багатоядерні процесори.





Вузькі місця в роботі процесора

- Зазвичай зворотні відносно вузьких місць вводу-виводу.
  - З’являються в додатках з великою кількістю обчислень чи інших обчислювально дорогих задач.
  - Швидкість роботи таких програм прив’язана до частоти процесора.
- Якщо швидкість обробки даних значно нижча за швидкість їх отримування, маємо вузьке місце, пов’язане з процесором (CPU).

Одноядерні процесори

- Виконують лише один потік.
  - Проте швидко перемикають контекст між багатьма потоками виконання тисячі раз за секунду.
  - Перемикання контексту включає зберігання всієї необхідної інформації для потоку в конкретний момент часу, а потім її відновлення в інший момент часу.
- При написанні багатопоточних додатків у Python перемикання контексту досить дороге обчислювально.
  - Цього ніяк не уникнути, тому сучасні ОС націлені на оптимізацію перемикання контексту.

Частота процесора

- Одне з ключових обмежень додатку, запущеного на одному ядрі.
  - Мається на увазі кількість тактів процесора кожну секунду.
- Проте транзистори за своїм розміром уже підходять до фізичної межі.
  - При подальшому зменшенні транзисторів ми стикнемось з ефектами квантового тунелювання.
  - У зв’язку з фізичними обмеженнями потрібно шукати інші методи нарощування комп’ютерних потужностей.

Модель масштабованості (scalability) Мартеллі

- Представляє три типи проблем та програм:
  - 1 ядро: однопоточні та однопроцесні програми.
  - 2-8 ядер: багатопоточні та багатопроцесні програми.
  - 9+ ядер: розподілені обчислення.
- Відносно скоро ми дійдемо до межі частот для 2-8-ядерних систем, і тоді доведеться шукати інші методи нарощування потужності, на зразок багатопроцесорних систем або навіть розподілених обчислень.
  - Якщо задачу слід вирішувати швидко, проте вона вимагає значних обчислювальних потужностей, можливим вирішенням є розподілені обчислення на багатьох машинах з багатьма запущеними екземплярами програми.
  - Великі промислові системи, які обробляють мільйони запитів – основні в цій категорії.

Time-sharing – планувальник задач (task scheduler)

- Один з найважливіших компонентів ОС.
  - Забезпечує, щоб кожна задача (task) мала шанс виконатись.
  - Проте, де та коли запуститься задача, чітко не визначено.
  - Якщо запустити два ідентичних процеси один за одним, гарантії того, що перший запущений процес завершиться теж першим немає.

```python
import threading
import time
import random

counter = 1

def workerA():
  global counter
  while counter < 1000:
    counter += 1
    print("Worker A is incrementing counter to {}".format(counter))
    sleepTime = random.randint(0,1)
    time.sleep(sleepTime)

def workerB():
  global counter
  while counter > -1000:
    counter -= 1
    print("Worker B is decrementing counter to {}".format(counter))
    sleepTime = random.randint(0,1)
    time.sleep(sleepTime)

def main():
  t0 = time.time()
  thread1 = threading.Thread(target=workerA)
  thread2 = threading.Thread(target=workerB)

  thread1.start()
  thread2.start()

  thread1.join()
  thread2.join()

  t1 = time.time()

  print("Execution Time {}".format(t1-t0))

if __name__ == '__main__':
  main()
```

Недетермінована поведінка

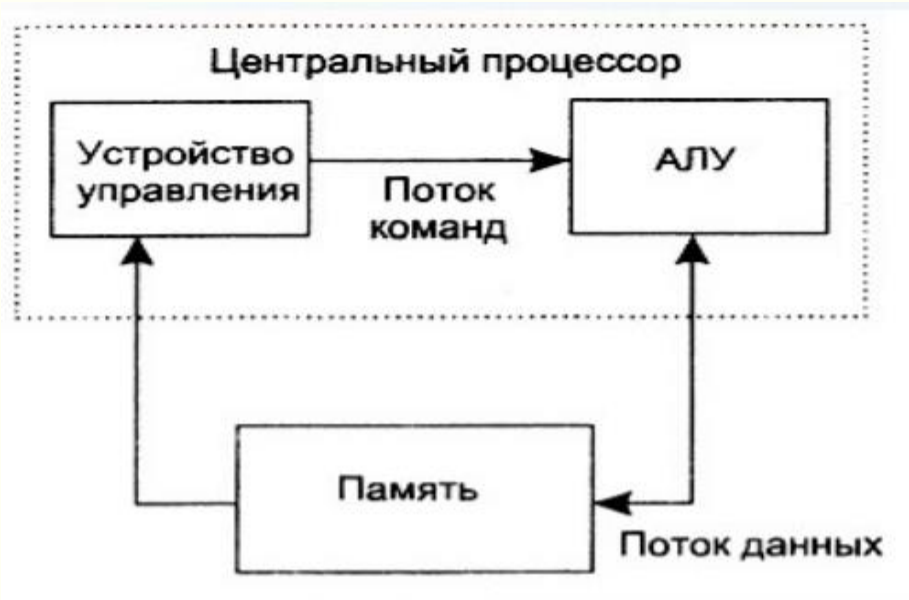
- Маємо два конкуруючих потоки, у кожного з яких свої цілі: декрементація до –1000 або навпаки – інкрементація до 1000.
  - З одноядерним процесором є ймовірність, що worker A встигне виконати завдання до того, як почне роботу worker B.
  - Це ж можна сказати і про worker B.
  - Третя можливість: планувальник задач продовжуватиме нескінченно перемикати контекст між worker A та worker B.

Архітектура системи

- Існує кілька архітектурних стилів пам’яті для різних сценаріїв використання.
  - Один може бути чудовим для паралельних обчислень, проте не підходити для звичайних домашніх комп’ютерів.
- Часто використовується таксономія Майкла Флінна (1972):
  - SISD: single instruction stream, single data stream
  - SIMD: single instruction stream, multiple data stream
  - MISD: multiple instruction stream, single data stream
  - MIMD: multiple instruction stream, multiple data stream

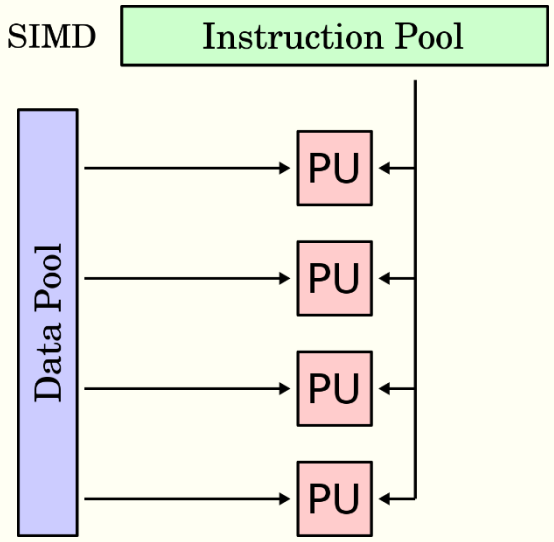
SISD

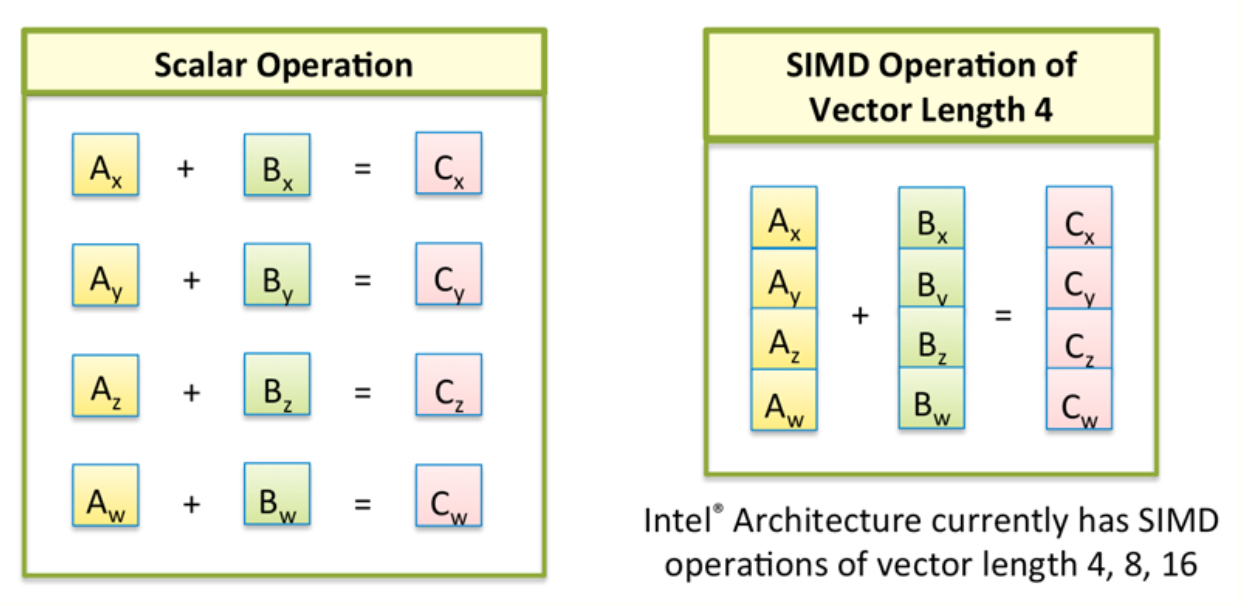
- Цільова спрямованість – однопроцесорні системи.
  - Ці системи мають один послідовний потік даних та один процесор, який виконує цей потік.
  - Описує класичні машини фон Неймана, а до появи багатоядерних процесорів представляла типовий домашній комп'ютер.
  - Такі системи не здатні на паралелізм даних або інструкцій, тому графічні процесори дуже обмежувались подібними системами.



SIMD

- Найкраще підходять для роботи з системами обробки мультимедійного контенту.

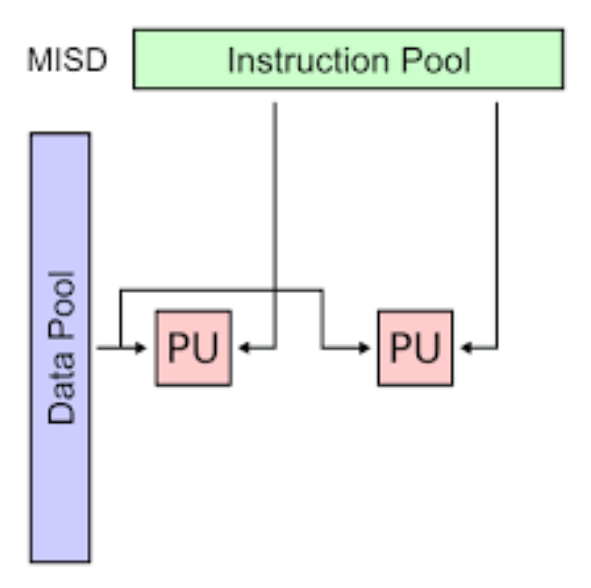




- Представляється в графічних процесорах.
  - В OpenGL-програмуванні існують об'єкти - Vertex Array Objects (VAO), які зазвичай містять багато Vertex Buffer Objects, які описують 3D-об'єкт у грі.
  - За потреби, наприклад, перемістити персонажа, позицію кожного елементу кожного Vertex Buffer object потрібно заново обчислити і дуже швидко.

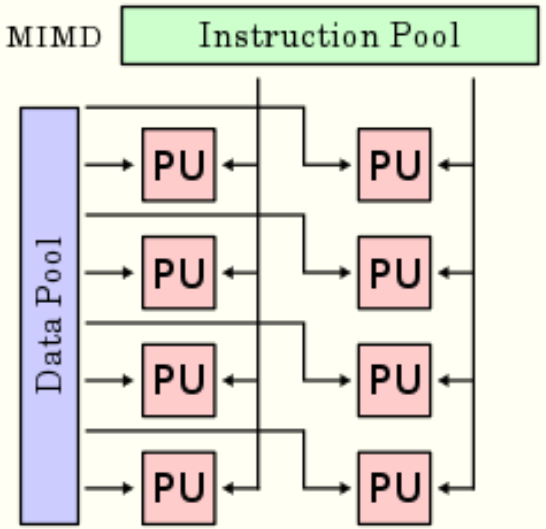
MISD

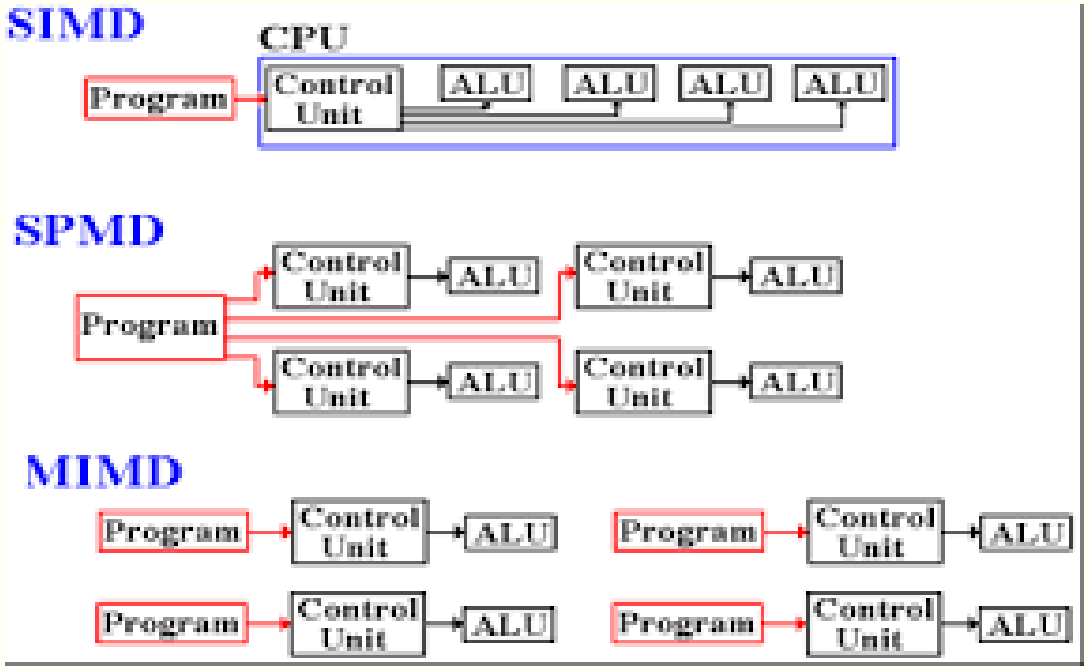
- Multiple instruction streams, single data streams – дещо нелюбима архітектура без реальних прикладів комерційних систем.



MIMD

- Найбільш різноманітна архітектура, включає всі сучасні багатоядерні процесори.
  - Кожне ядро такого процесора здатне паралельно з іншими та незалежно від них виконувати інструкції.
  - На відміну від SIMD-машин, пристрої на базі архітектури MIMD здатні запускати ***кілька окремих операцій*** для багатьох наборів даних паралельно.



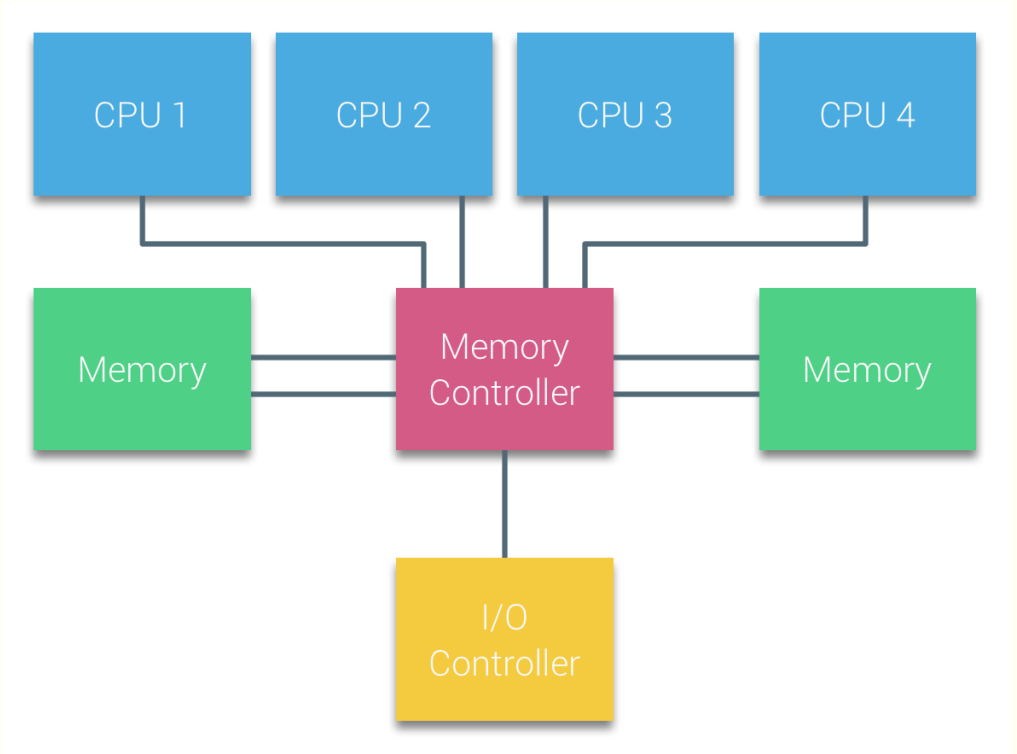


Архітектурні стилі пам’яті комп’ютера

- Один з найбільших викликів – швидкість доступу до даних.
  - Якщо немає достатньо швидкого доступу до даних, виникає вузьке місце в роботі програм незалежно від якості проектування системи.
- Проектувальники постійно шукають способи спрощення розробки паралельних рішень для задач.
  - Один із способів покращення – надавати єдиний фізичний адресний простір, до якого можуть звертатись ядра процесора.
  - Це спрощує код, оскільки не треба звертати увагу на безпеку потоків.
- Різні архітектури використовуються у широкому діапазоні сценаріїв.
  - Дві основні архітектури, які найчастіше застосовують проектувальники, відповідають або шаблону Uniform Memory Access (UMA), або Non-uniform memory access (NUMA).

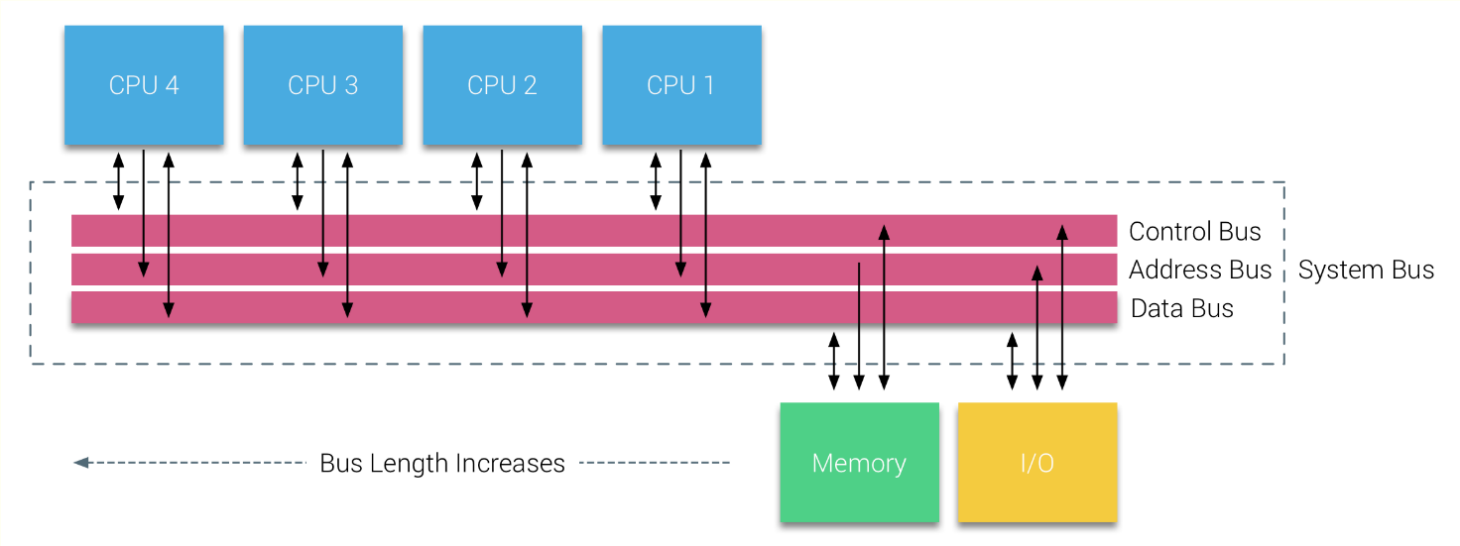
UMA (Uniform Memory Access)

- Застосовує спільний прості пам’яті (shared memory space), який може рівноймовірно використовуватись будь-якою кількістю ядер.
  - Незалежно від приналежності ядра воно зможе напряму звертатись до місця в пам’яті незалежно від його віддаленості.
    - Також цю архітектуру називають Symmetric Shared-Memory Multiprocessors (SMP).

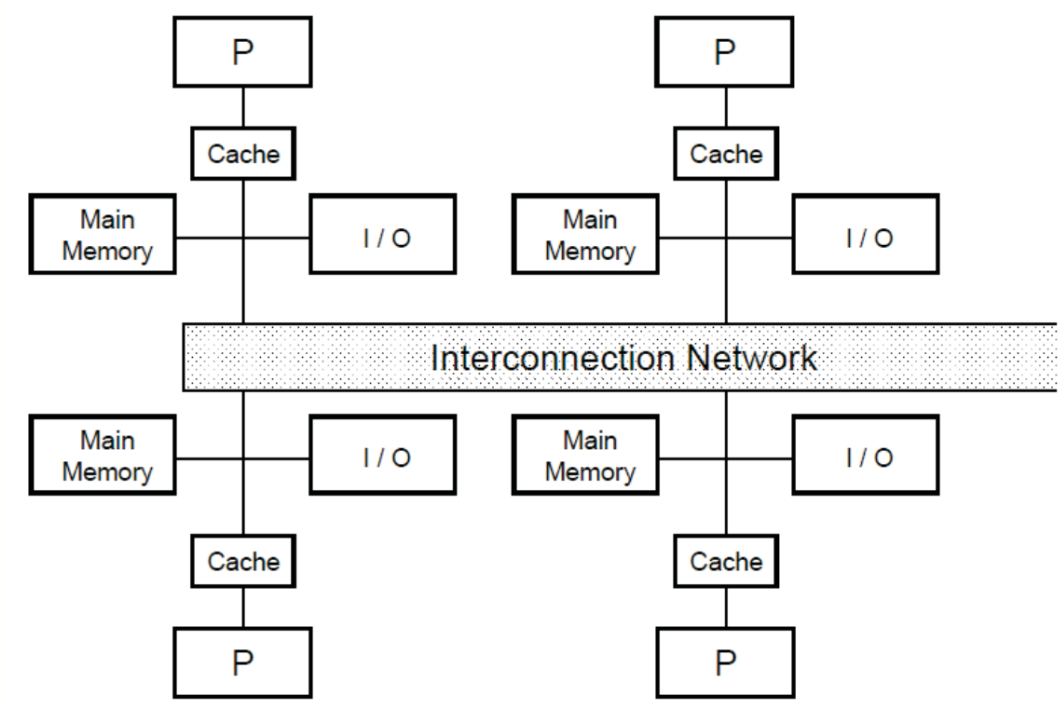


Переваги та недоліки UMA

- Переваги:
  - Кожне звернення до оперативної пам’яті відбувається за фіксований, точний період часу.
  - Кеш когерентний та узгоджений.
  - Проектування апаратного забезпечення простіше.
- Недолік:
  - UMA-системи застосовують одну шину пам’яті, від якої всі системи мають до пам’яті доступ, що вказує на проблеми масштабованості.



NUMA



NUMA (Non-uniform Memory Access) – архітектура, в якій деякий доступ до пам'яті може бути швидшим за інші залежно від процесора, який виконує звернення.

- Може бути пов'язано з розташуванням процесора відносно пам'яті.
- Кожен процесор володіє власним кешем, підсистемою вводу-виводу, доступом до основної пам'яті.
- Процесор під'єднано до interconnection network

Основна перевага NUMA:

- NUMA-машини краще масштабуються, ніж UMA-пристрої.

Недоліки NUMA:

- Недетермінований час доступу до пам'яті може коливатись від дуже швидкого, за умови, що пам'ять локальна, до набагато повільнішого для віддаленої в розташуванні пам'яті.
- Процесори повинні спостерігати за змінами, що були здійснені іншими процесорами; чим більше процесорів, тим більше часу це займає.

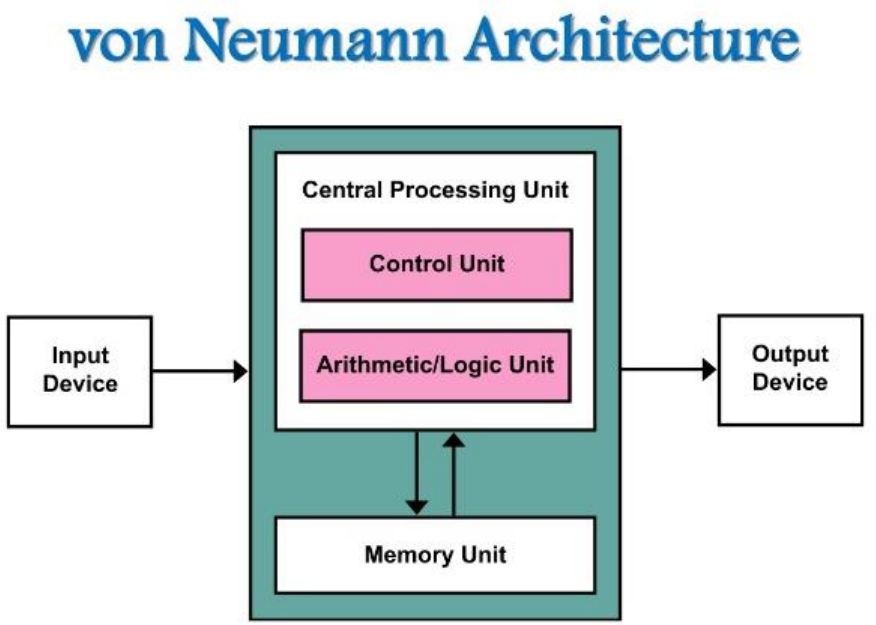
ДЯКУЮ ЗА УВАГУ!

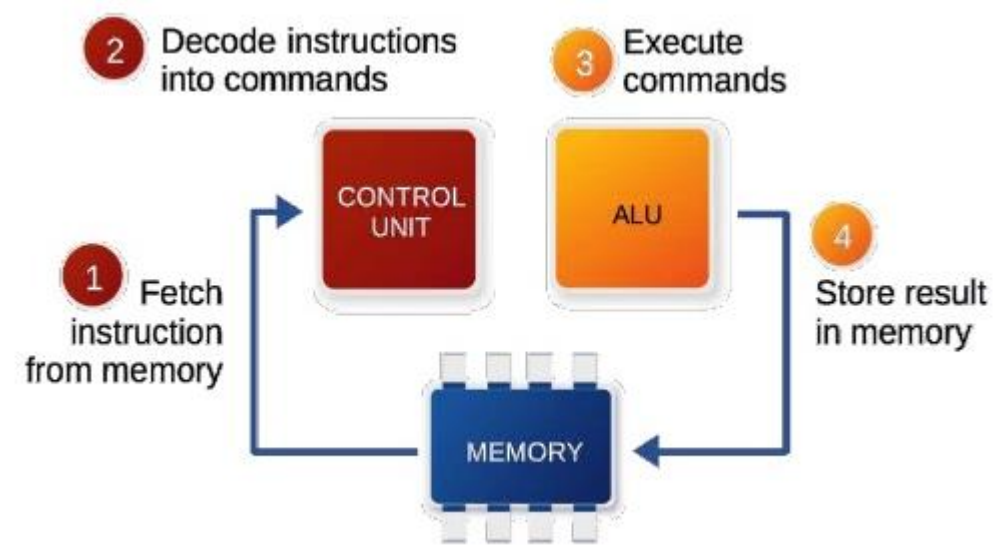
Наступне питання: Життєвий цикл потоку

Багатоядерні процесори

- Кожне ядро містить все необхідне для виконання послідовності збережених інструкцій.
- У кожного ядра власний цикл, який складається з таких процесів:
  - ***Отримання (Fetch):*** одержання інструкцій з пам'яті. Лічильник команд (program counter, PC) ідентифікує місце, з якого починати виконання наступного кроку.
  - ***Декодування (Decode):*** ядро конвертує отриману інструкцію в сигнальну послідовність, яка запускає різні частини процесора.
  - ***Виконання (Execute):*** запускається декодована інструкція, результати її виконання зберігаються в регістр процесора.

Архітектура фон Неймана





- На багатьох ядрах можна незалежно виконувати кілька циклів Fetch -> Decode -> Execute.
  - Стиль архітектури дозволяє створювати більш продуктивні програми, що підтримують паралельне виконання.