




Welcome to CoGrammar Parallelism and Concurrency

The session will start shortly...

Questions? Drop them in the chat. We'll have dedicated moderators answering questions.



Coding Interview Workshop Housekeeping

- The use of disrespectful language is prohibited in the questions, this is a supportive, learning environment for all - please engage accordingly.
(Fundamental British Values: Mutual Respect and Tolerance)
- No question is daft or silly - **ask them!**
- There are **Q&A sessions** midway and at the end of the session, should you wish to ask any follow-up questions. Moderators are going to be answering questions as the session progresses as well.
- If you have any questions outside of this lecture, or that are not answered during this lecture, please do submit these for upcoming Academic Sessions. You can submit these questions here: [Questions](#)

Coding Interview Workshop Housekeeping cont.

- For all **non-academic questions**, please submit a query: www.hyperiondev.com/support
- Report a **safeguarding** incident: www.hyperiondev.com/safeguardreporting
- We would love your **feedback** on lectures: [Feedback on Lectures](#)

Skills Bootcamp

8-Week Progression Overview

Fulfil 4 Criteria to Graduation

✓ Criterion 1: Initial Requirements

Timeframe: First 2 Weeks

Guided Learning Hours (GLH):

Minimum of 15 hours

Task Completion: First four tasks

Due Date: 24 March 2024

✓ Criterion 2: Mid-Course Progress

60 Guided Learning Hours

Data Science - **13 tasks**

Software Engineering - **13 tasks**

Web Development - **13 tasks**

Due Date: 28 April 2024

Skills Bootcamp Progression Overview

✓ Criterion 3: Course Progress

Completion: All mandatory tasks,
including Build Your Brand and
resubmissions by study period end
Interview Invitation: Within 4 weeks
post-course
Guided Learning Hours: Minimum of
112 hours by support end date
(10.5 hours average, each week)

✓ Criterion 4: Demonstrating Employability

Final Job or Apprenticeship
Outcome: Document within 12
weeks post-graduation
Relevance: Progression to
employment or related
opportunity



**SKILLS
FOR LIFE**

SKILLS BOOTCAMPS



Department
for Education

CoGrammar

Parallelism and Concurrency

May 2024

Lecture Objectives

- Distinguish between parallelism and concurrency
- Implement concurrent and parallel programming in Python and JavaScript
- Evaluate the impact of parallelism and concurrency on application scalability and performance



Introduction

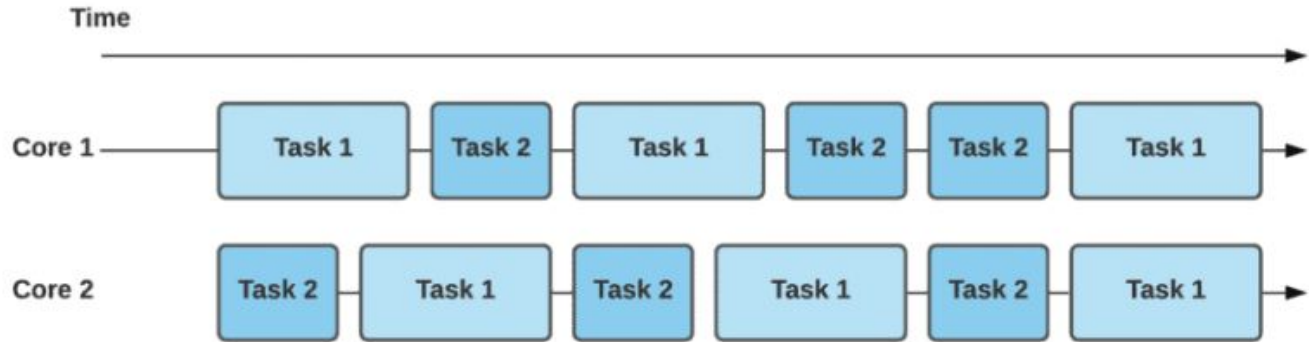
- ❖ In today's computing landscape, leveraging parallelism and concurrency is crucial for improving application performance and responsiveness
- ❖ As hardware capabilities advance, software must adapt to utilize multi-core processors and handle asynchronous operations effectively



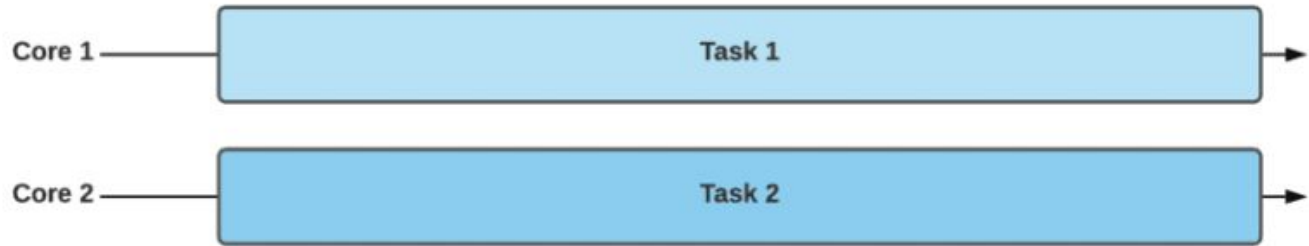
Parallelism vs. Concurrency

- ❖ **Concurrency:** Interleaved execution of multiple tasks, giving the illusion of simultaneous processing
- ❖ **Parallelism:** Simultaneous execution of multiple tasks on different processing units
- ❖ Concurrency is about dealing with multiple tasks at once, while parallelism is about executing multiple tasks at the same time

Concurrent

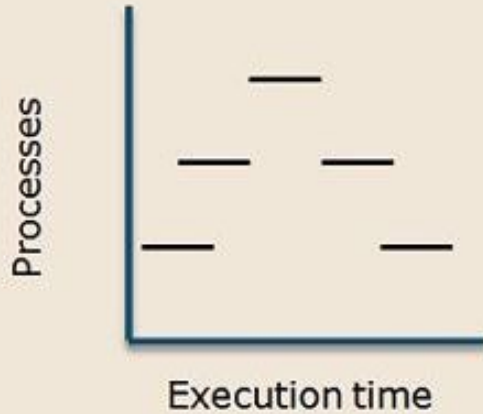


Parallel



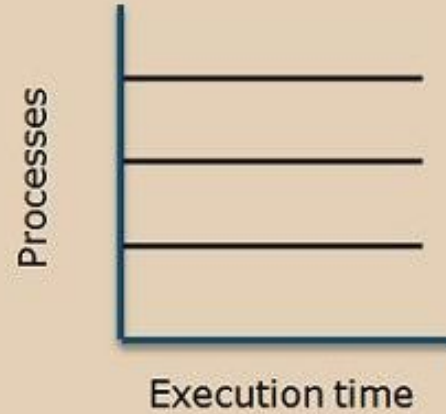
Source: <https://www.baeldung.com/cs/concurrency-vs-parallelism>

CONCURRENCY



VS

PARALLELISM



Source: <https://techdifferences.com/difference-between-concurrency-and-parallelism.html>



Parallelism vs. Concurrency

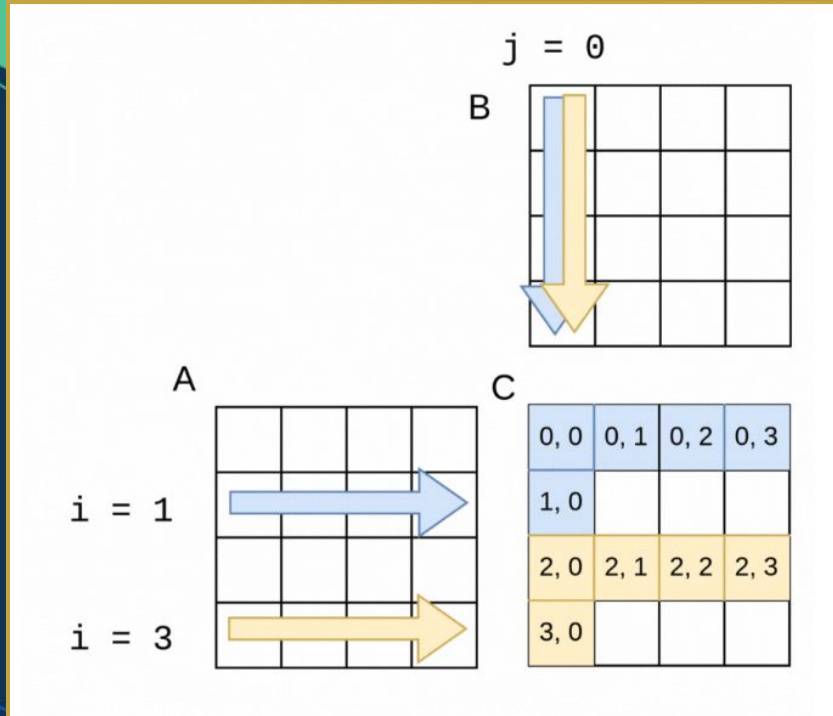
- ❖ Key differences
 - Parallelism focuses on executing tasks simultaneously, while concurrency focuses on managing and coordinating multiple tasks
 - Parallelism requires hardware support (multi-core processors), while concurrency can be achieved on single-core systems



Parallel Computing

- ❖ Parallel computing involves the simultaneous execution of multiple tasks on different processing units
- ❖ It aims to improve performance by distributing workload across multiple processors or cores
- ❖ Example: Matrix multiplication can be parallelised by distributing the computation of each element across multiple processors





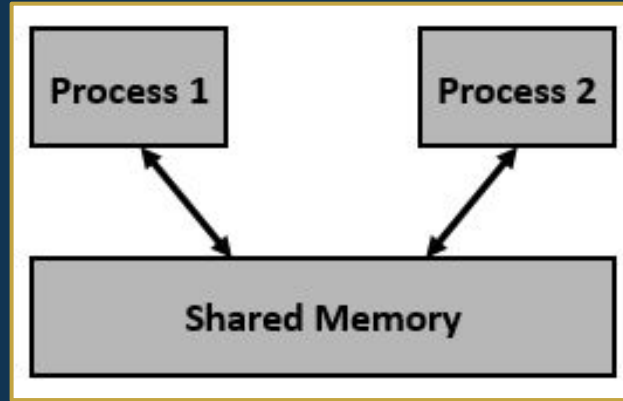
Here we have two processes doing matrix multiplication to speed it up

Source: <https://www.learnpdc.org/PDCBeginners/5-applications/matrix-multiply.html>



Parallel Architectures

- ❖ Shared Memory
 - Multiple processors share a common memory space (e.g., multi-core CPUs)
 - Processors can communicate and synchronise through shared variables
 - Advantages: Low communication overhead, easy to program
 - Disadvantages: Limited scalability, potential for race conditions



Source:

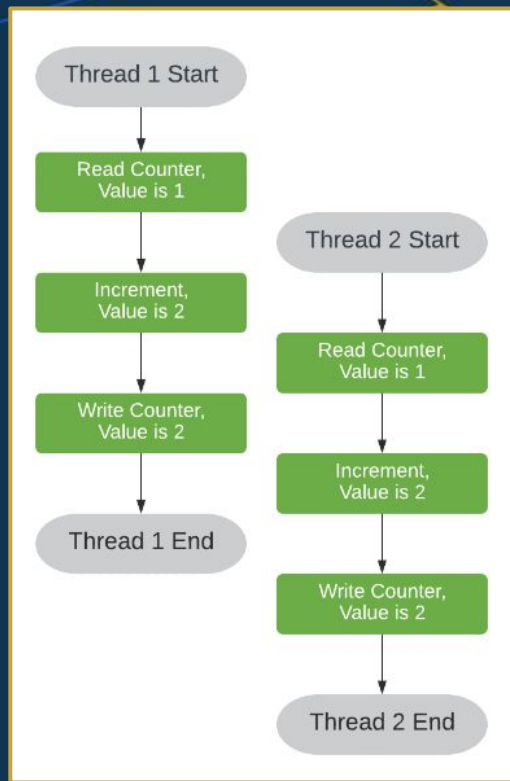
https://www.tutorialspoint.com/inter_process_communication/inter_process_communication_shared_memory.htm

When race conditions occur

A race condition occurs when two threads access a shared variable at the same time. The first thread reads the variable, and the second thread reads the same value from the variable. Then the first thread and second thread perform their operations on the value, and they race to see which thread can write the value last to the shared variable. The value of the thread that writes its value last is preserved, because the thread is writing over the value that the previous thread wrote.


Both threads read the same initial value (1) before either has a chance to write back the incremented value. They then both write 2, even though the counter should have been incremented twice to 3.

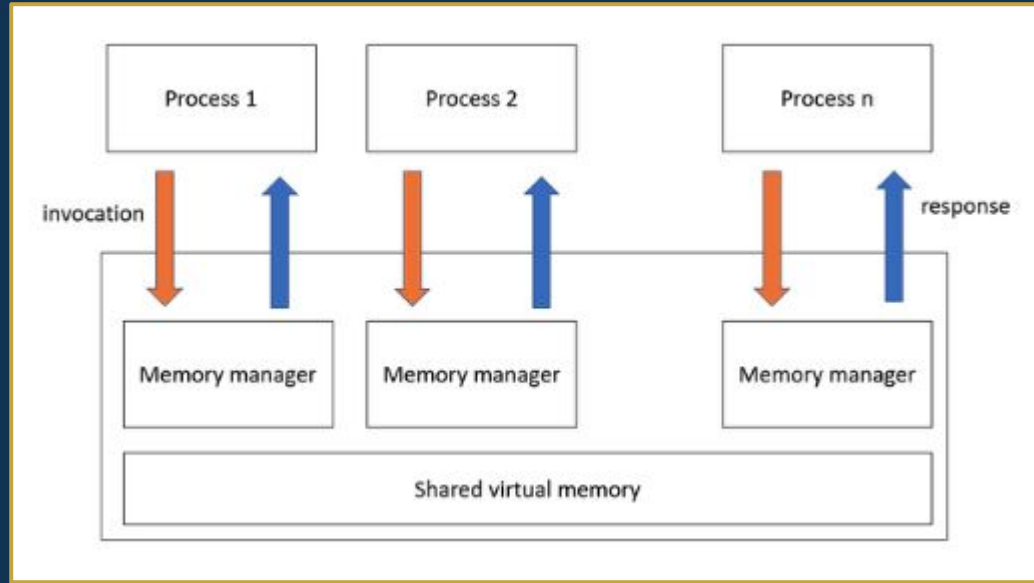
Source: <https://www.baeldung.com/cs/race-conditions>





Parallel Architectures

- ❖ Distributed Memory
 - Each processor has its own local memory, and communication occurs through a network
 - Processors communicate by sending messages over the network
 - Advantages: High scalability, no shared memory bottlenecks
 - Disadvantages: Higher communication overhead, more complex programming model
- 



Source: <https://www.geeksforgeeks.org/what-is-distributed-shared-memory-and-its-advantages/>

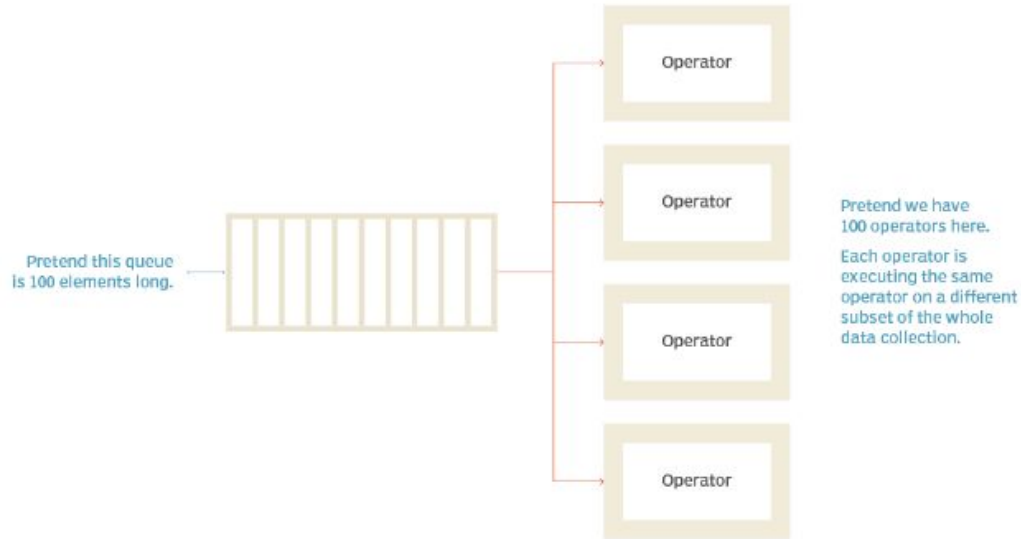


Parallel Programming Models

- ❖ Data Parallelism:
 - The same operation is performed on multiple data elements simultaneously (e.g., SIMD operations)
 - Each processing unit works on a portion of the data
 - Example: Applying a filter to an image, where each pixel can be processed independently



Data parallelization illustrated



Source:

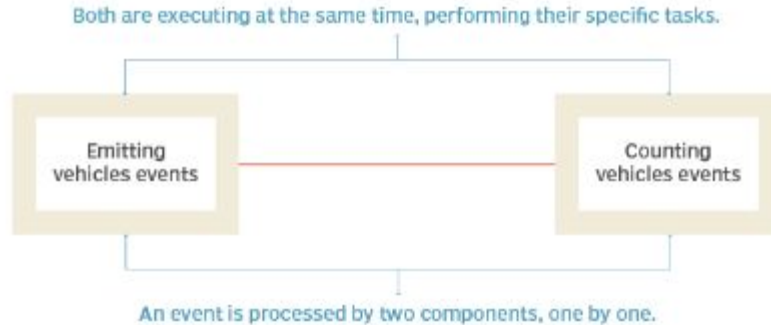
<https://www.techtarget.com/searchdatamanagement/post/How-parallelization-works-in-streaming-systems>



Parallel Programming Models

- ❖ Task Parallelism:
 - Different tasks or functions are executed in parallel (e.g., multi-threading)
 - Each processing unit executes a different task or function
 - Example: Executing multiple independent queries on a database simultaneously

Task parallelism illustrated



Source:

<https://www.techtarget.com/searchdatamanagement/post/How-parallelization-works-in-streaming-systems>




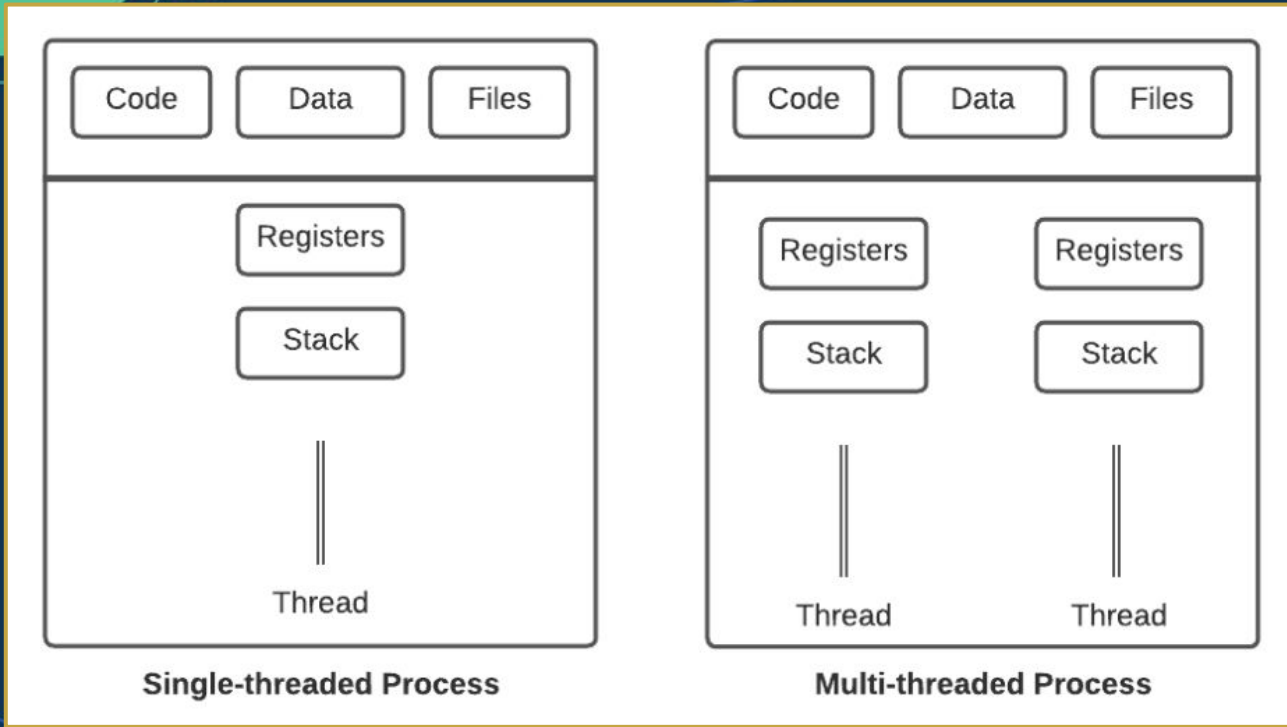
Concurrency - Threads and Processes

- ❖ Threads:
 - Lightweight units of execution within a single process, sharing the same memory space
 - Each thread has its own program counter, stack, and local variables
 - Threads can communicate through shared memory



Concurrency - Threads and Processes

- ❖ Processes:
 - Separate instances of a program with isolated memory spaces, communicating through inter-process communication (IPC) mechanisms
 - Each process has its own memory space, program counter, stack, and file descriptors
 - Processes communicate through IPC mechanisms like pipes, sockets, and shared memory
- 



Source: <https://www.baeldung.com/cs/process-vs-thread>



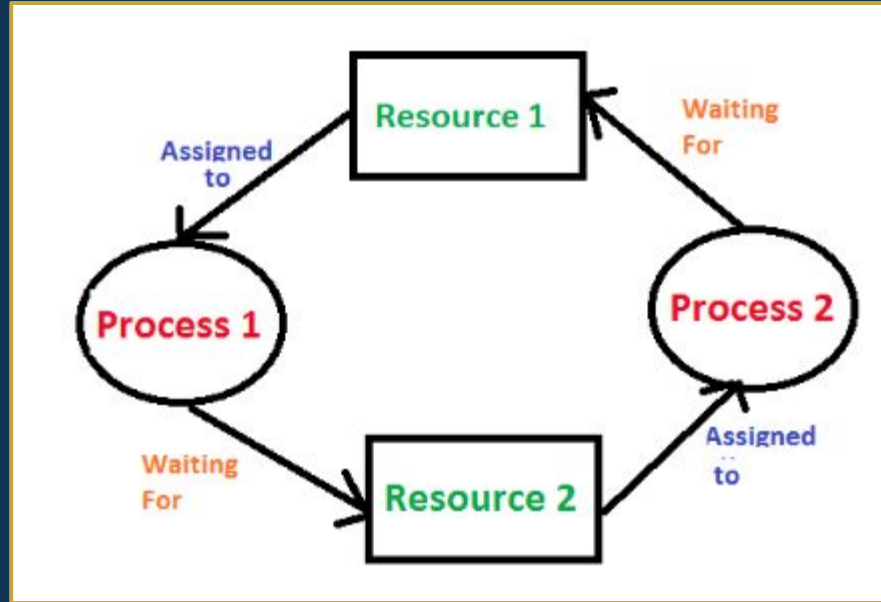
Challenges in Concurrent Programming

- ❖ Race Conditions:
 - Unexpected behaviour due to unsynchronised access to shared resources
 - Occurs when multiple threads access and manipulate shared data concurrently, leading to non-deterministic results



Challenges in Concurrent Programming

- ❖ Deadlocks:
 - Situations where two or more processes are unable to proceed because each is waiting for the other to release a resource
 - Occurs when there is a circular dependency of locks or resources among threads

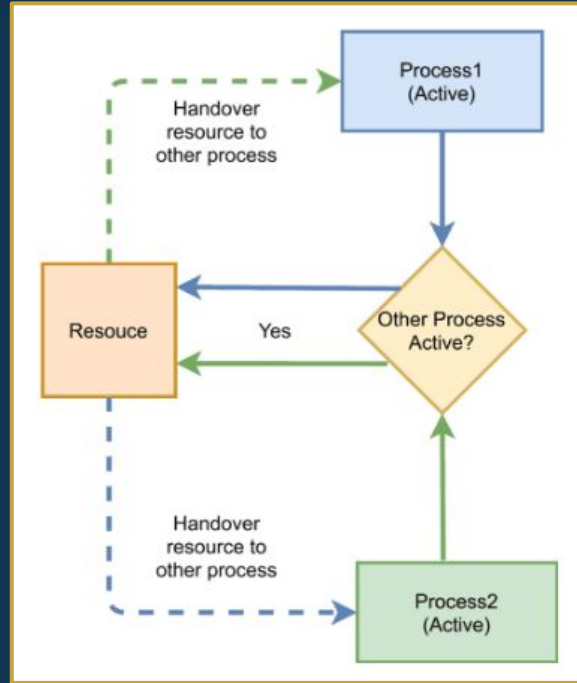


Source: <https://www.geeksforgeeks.org/introduction-of-deadlock-in-operating-system/>



Challenges in Concurrent Programming

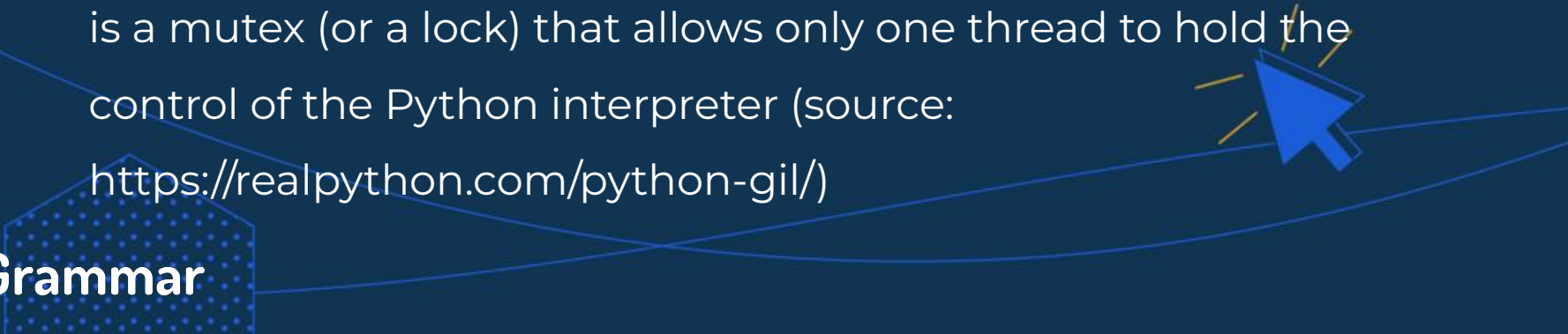
- ❖ Livelocks:
 - Similar to deadlocks, but processes continuously change their state without making progress
 - Occurs when threads repeatedly respond to each other's actions without advancing



Source: <https://www.baeldung.com/cs/deadlock-livelock-starvation>



Parallelism in Python

- ❖ The multiprocessing module allows spawning processes to achieve parallelism
 - ❖ It provides a similar interface to the threading module but enables true parallelism by bypassing the GIL
 - ❖ The Python Global Interpreter Lock or GIL, in simple words, is a mutex (or a lock) that allows only one thread to hold the control of the Python interpreter (source: <https://realpython.com/python-gil/>)
- 

```
import multiprocessing
import time

def worker(process_id):
    print(f"Process {process_id} started")
    time.sleep(2)
    print(f"Process {process_id} finished")

if __name__ == '__main__':
    processes = []
    for i in range(3):
        process = multiprocessing.Process(target=worker, args=(i,))
        processes.append(process)
        process.start()

    for process in processes:
        process.join()
```

```
Process 1 started
Process 0 started
Process 2 started
Process 0 finished
Process 2 finished
Process 1 finished
```



Parallelism in Python

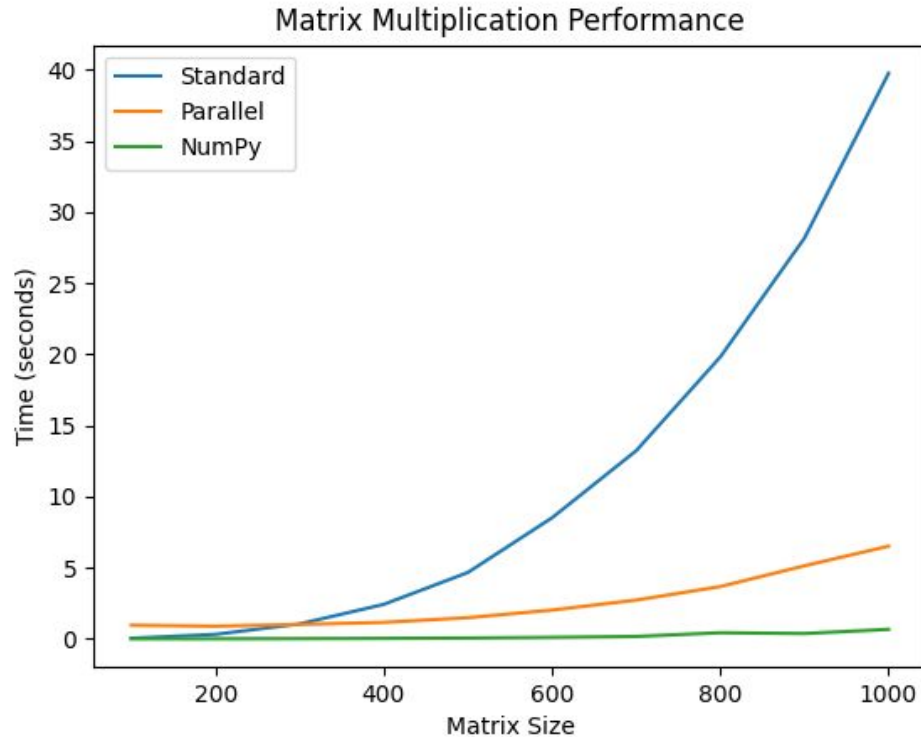
- ❖ Parallel Libraries in Python
 - NumPy: Provides vectorised operations and efficient numerical computations
 - Pandas: Offers data parallelism through its DataFrame and Series objects
 - Joblib: Enables easy parallelisation of Python code

```
def multiply_row(args):
    row_a, matrix_b = args
    return [sum(a*b for a, b in zip(row_a, col_b)) for col_b in zip(*matrix_b)]

def matrix_multiply(matrix_a, matrix_b):
    return list(map(multiply_row, [(row_a, matrix_b) for row_a in matrix_a]))

def parallel_matrix_multiply(matrix_a, matrix_b):
    with multiprocessing.Pool() as pool:
        return pool.map(multiply_row, [(row_a, matrix_b) for row_a in matrix_a])

def numpy_matrix_multiply(matrix_a, matrix_b):
    return np.dot(matrix_a, matrix_b)
```



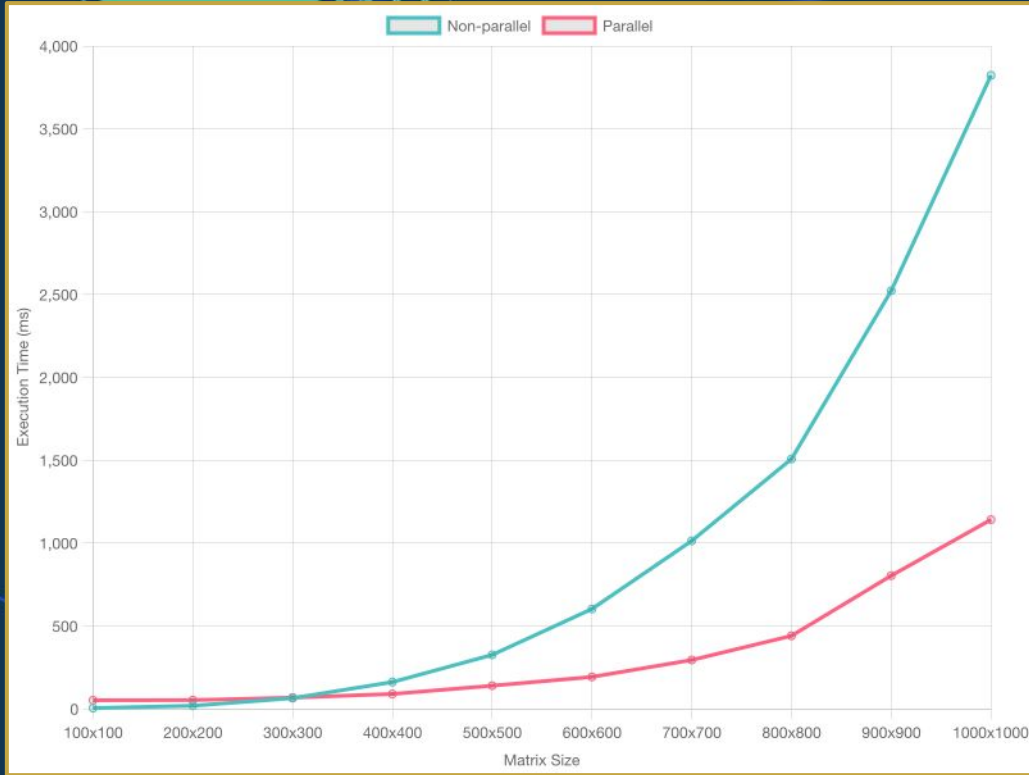
Even though Parallel multiplication improved upon standard multiplication, the highly efficient numpy library took the win by far (this experiment was done on a 2021 M1 Pro 16GB)



Parallelism in Javascript

- ❖ Web Workers allow running scripts in the background, separate from the main JavaScript thread
- ❖ They enable parallel processing in web browsers

```
function multiplyMatrices(matrixA, matrixB, startRow, endRow) {  
  const colsA = matrixA[0].length;  
  const colsB = matrixB[0].length;  
  
  const result = [];  
  for (let i = startRow; i < endRow; i++) {  
    result[i - startRow] = [];  
    for (let j = 0; j < colsB; j++) {  
      let sum = 0;  
      for (let k = 0; k < colsA; k++) {  
        sum += matrixA[i][k] * matrixB[k][j];  
      }  
      result[i - startRow][j] = sum;  
    }  
  }  
  
  return result;  
}
```



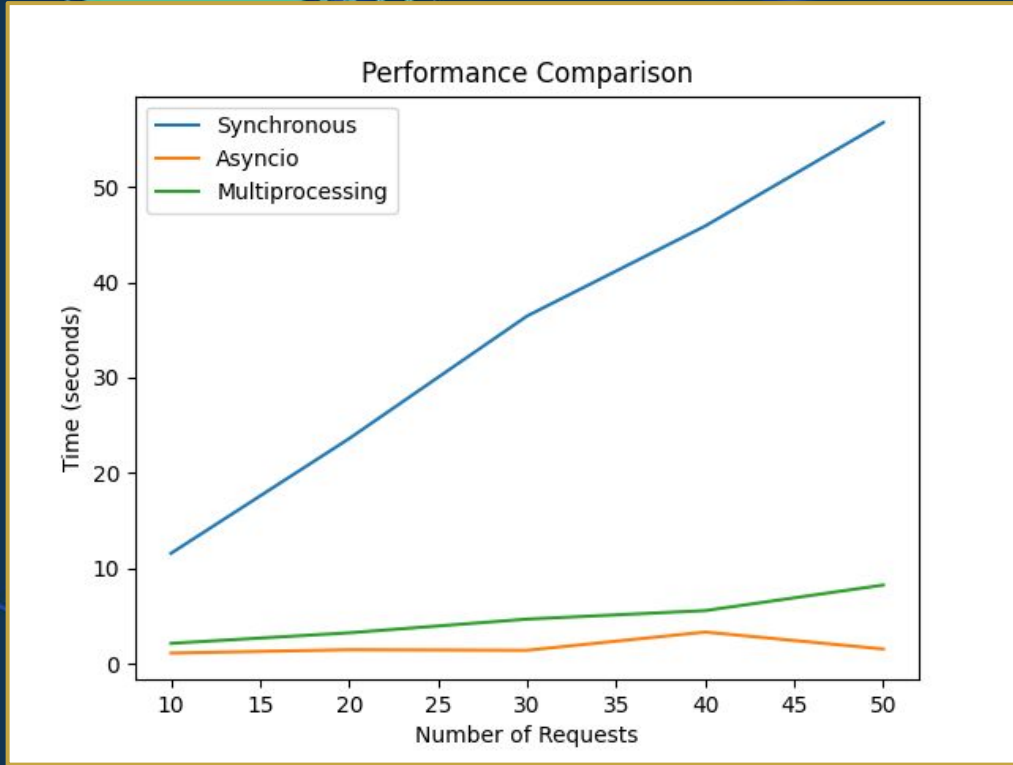
Parallelisation becomes more effective as the number of operations increases (this experiment was done on a 2021 M1 Pro 16GB)





Concurrency in Python

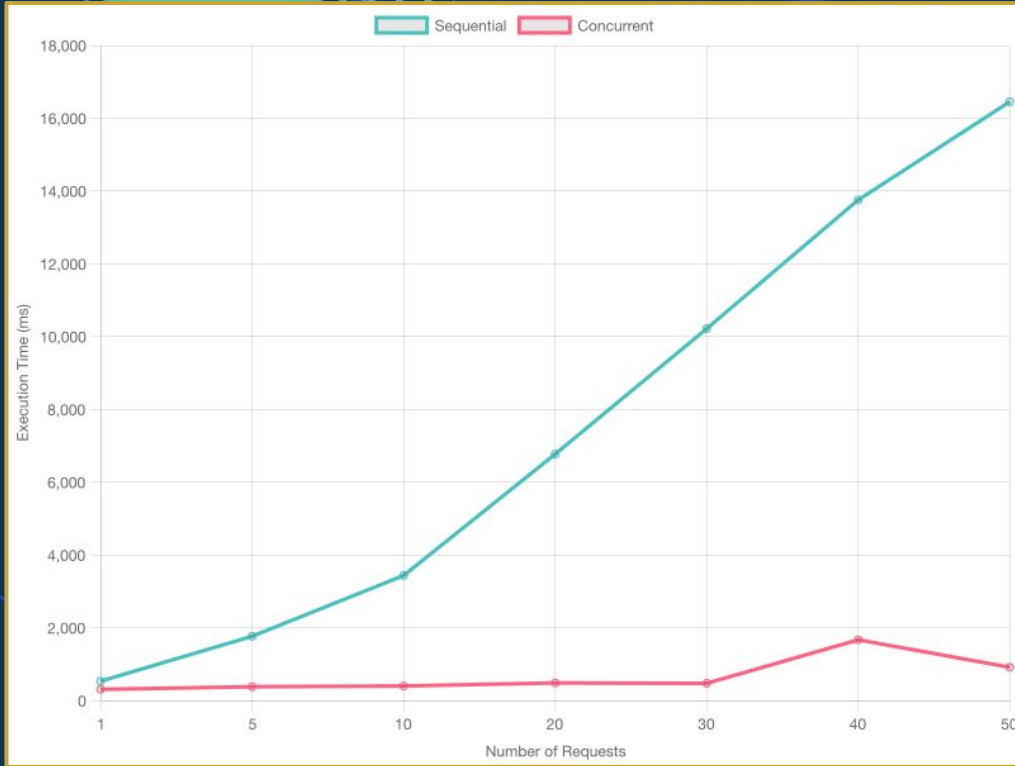
- ❖ The `asyncio` module provides tools for writing asynchronous code using the `async/await` syntax



Once again the python library `asyncio` beats even the efficient concurrent solution, although even more importantly the synchronous solution takes significantly longer (this experiment was done on a 2021 M1 Pro 16GB)

Concurrency in Javascript

- ❖ JavaScript's event-driven, non-blocking I/O model enables asynchronous programming
- ❖ Async/await provides a more concise and readable way to work with Promises
 - A promise represents an asynchronous operation whose result will come in the future (source: <https://www.freecodecamp.org/news/javascript-promises-es-async-await-and-promise-methods/>)



Concurrency performs much better than sequential execution for these api calls (this experiment was done on a 2021 M1 Pro 16GB)



What is the key difference between parallelism and concurrency?

- A. Task execution
- B. Hardware requirements
- C. Processing focus
- D. No significant difference

What is the key difference between parallelism and concurrency?

- A. **Task execution**
- B. Hardware requirements
- C. Processing focus
- D. No significant difference

Which of the following is an advantage of shared memory parallel architectures?

- A. High scalability
- B. No shared memory bottlenecks
- C. Low communication overhead
- D. More complex programming model

Which of the following is an advantage of shared memory parallel architectures?

- A. High scalability
- B. No shared memory bottlenecks
- C. Low communication overhead**
- D. More complex programming model

What can lead to race conditions in concurrent programming?

- A. Synchronized data access
- B. Circular lock dependencies
- C. Unsynchronized data access
- D. Threads responding without advancing

What can lead to race conditions in concurrent programming?

- A. Synchronized data access
- B. Circular lock dependencies
- C. Unsynchronized data access**
- D. Threads responding without advancing

Which Python module allows spawning processes to achieve parallelism by bypassing the Global Interpreter Lock (GIL)?

- A. threading
- B. asyncio
- C. multiprocessing
- D. concurrent.futures



Which Python module allows spawning processes to achieve parallelism by bypassing the Global Interpreter Lock (GIL)?

- A. threading
- B. asyncio
- C. multiprocessing**
- D. concurrent.futures



In JavaScript, what does a promise represent?

- A. A synchronous operation whose result is immediately available
- B. An asynchronous operation whose result will come in the future
- C. A way to achieve parallelism in web browsers
- D. A method for coordinating multiple threads

In JavaScript, what does a promise represent?

- A. A synchronous operation whose result is immediately available
- B. An asynchronous operation whose result will come in the future**
- C. A way to achieve parallelism in web browsers
- D. A method for coordinating multiple threads

Portfolio Assignment: SE

Binary Search Tree Library

Objective: Develop a library for efficient manipulation of binary search trees in Python or JavaScript

Portfolio Assignment: SE

Requirements:

- ❖ Implement a BST library with methods for insertion, deletion, searching, and traversal
- ❖ Optimize the library for performance
- ❖ Write unit tests
- ❖ Create a README file with documentation

Portfolio Assignment: DS

Decision Tree Visualisation

Objective: Create a Python script that visualises a decision tree using a given dataset

Portfolio Assignment: DS

Requirements:

- ❖ Use libraries like scikit-learn and matplotlib to build and visualise a decision tree
- ❖ Analyse the impact of different hyperparameters on the tree structure and performance
- ❖ Provide a Jupyter Notebook with well-documented code and insights

Portfolio Assignment: WD

Heap-based Priority Queue

Objective: Implement a priority queue using a heap in a web-based application

Portfolio Assignment: WD

Requirements:

- ❖ Create a web application that allows users to add and remove elements from a priority queue
- ❖ Implement the priority queue using a heap data structure
- ❖ Design an intuitive user interface
- ❖ Deploy the application and provide a link to the live demo

CoGrammar

Q & A SECTION

**Please use this time to ask
any questions relating to the
topic, should you have any.**

Thank you for attending



Department
for Education

CoGrammar

