Concurrency and Parallelism in Python

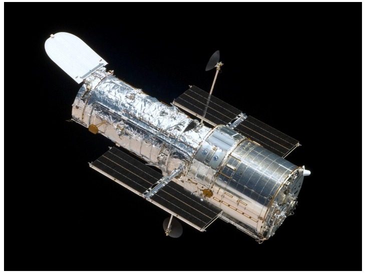
Yingquan Li

Catalog Science Branch (CSB) - Data Management Division (DMD)
Space Telescope Science Institute (STScI)

Software Engineering Roundtable
Wed. 10/1/25 @ 12 Noon EDT

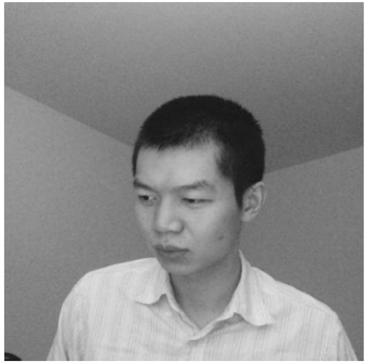
About

STScI | SPACE TELESCOPE SCIENCE INSTITUTE



Hubble Space Telescope (HST)

Transiting Exoplanet Survey Satellite (TESS)



My background is in:

- *Science/Engineering* and *Business*

I've worked in:

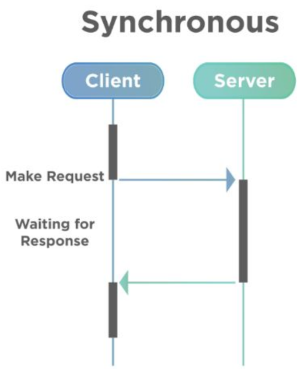
- *Academia, Government, Private Industry*

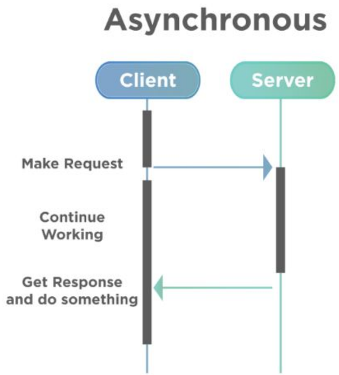
AGENDA

1. Go over the theory:
   - *Synchronous* vs. *Asynchronous Execution*
   - *I/O Bound* vs. *CPU Bound*
   - *Concurrency* vs. *Parallelism*
2. Go over the Python libraries:
   - *asyncio*
   - *threading*
   - *multiprocessing*
   - *concurrent.futures*
3. DEMOs!
4. Words of Wisdom
5. Conclusion & References

Theory!

Synchronous vs. Asynchronous (Programming Styles)





- Synchronous: Tasks must execute one at a time in sequential order; operation must complete fully before the next one begins.
- Asynchronous: Tasks can be initiated without waiting for them to complete; multiple operations can start at the same time.

Image Source

I/O Bound vs. CPU Bound (Programming Bottlenecks)

- Input/Output (I/O) Bound: Situations that spend more time on I/O tasks rather than computation tasks.
- CPU Bound: Situations that spend more time on computation tasks rather than I/O tasks.

I/O Bound Situations	CPU Bound Situations
● Reading files from disk ● Making HTTP requests over the network ● Database queries ● User input ● Downloading files ● API calls	● Mathematical calculations ● Image/video processing ● Data analysis and statistics ● Cryptographic operations ● Machine learning model training ● Sorting large datasets

Concurrency vs. Parallelism (How the Machine Executes Tasks)

- Concurrency: Concurrency means multiple tasks are running and taking turns on the same resource (i.e. 1 CPU core).

- Parallelism: Multiple tasks actually run simultaneously on separate resources (i.e. multiple CPU cores).

- Concurrency ≠ Parallelism!

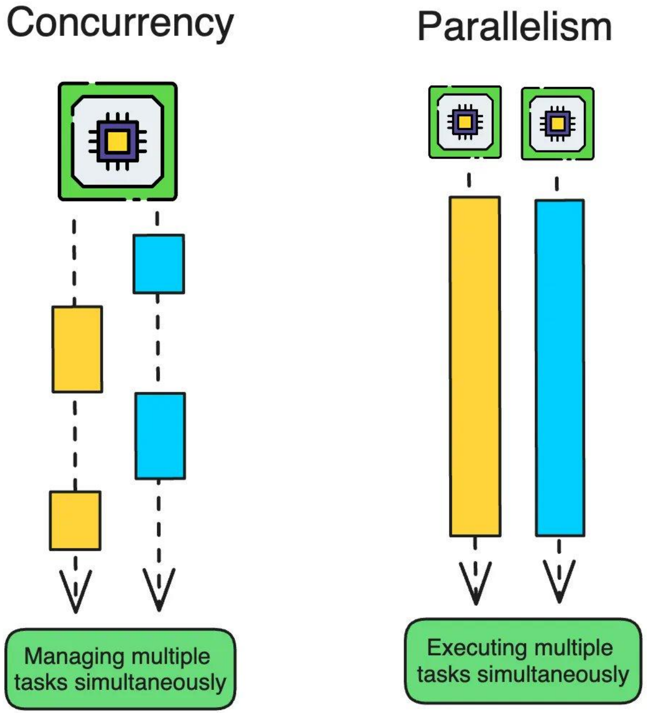

Concurrency
Parallelism
Managing multiple tasks simultaneously
Executing multiple tasks simultaneously

Image Source

Python Libraries!

asyncio

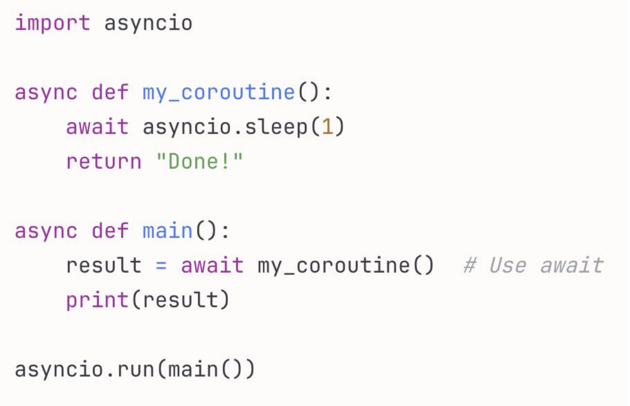
- Python library used to write concurrent code using async/await syntax.

- asyncio is best for tasks that wait a lot (i.e. network I/O).

- asyncio is also best for managing many writing tasks.

- Handles I/O bound computing. Five key concepts: Event loop, coroutines, tasks, futures, and synchronization.

Visual Diagrams (asyncio)



```python
import asyncio

async def my_coroutine():
    await asyncio.sleep(1)
    return "Done!"

async def main():
    result = await my_coroutine()   # Use await
    print(result)

asyncio.run(main())
```

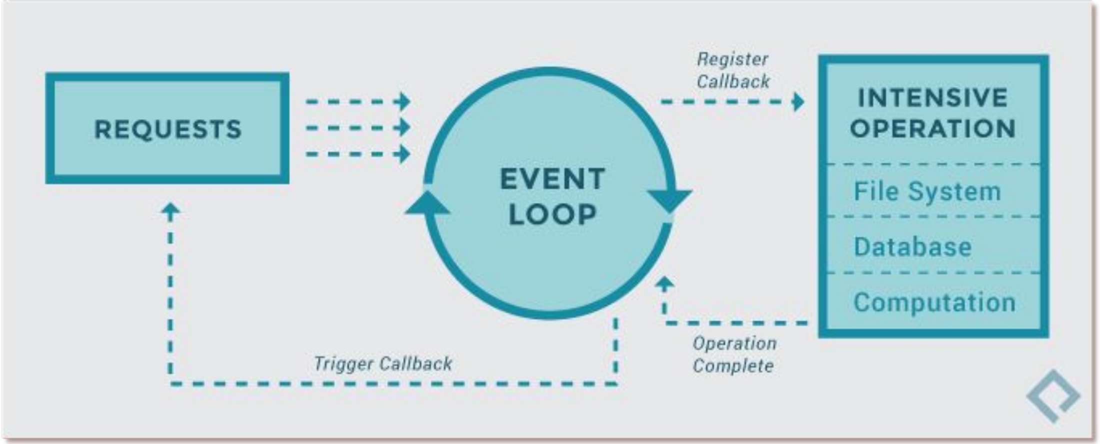
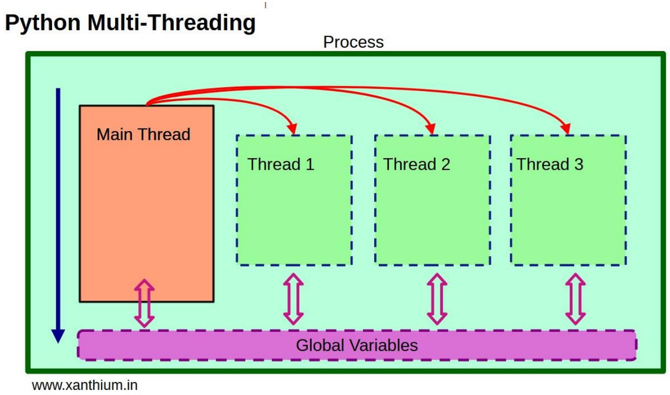


Image Source

threading

- Python library that allows a single process to spawn multiple threads (atomic units of a process).
- Threads can run in parallel within the same application; *data is shared*.
- Handles I/O bound computing that's less CPU intensive (faster I/O).
- Be careful of the Global Interpreter Lock (GIL)! The GIL ensures only one thread controls the interpreter; may get in the way.
- Be wary of: race conditions, dead/live locks, and resource starvation.

Visual Diagrams (threading)



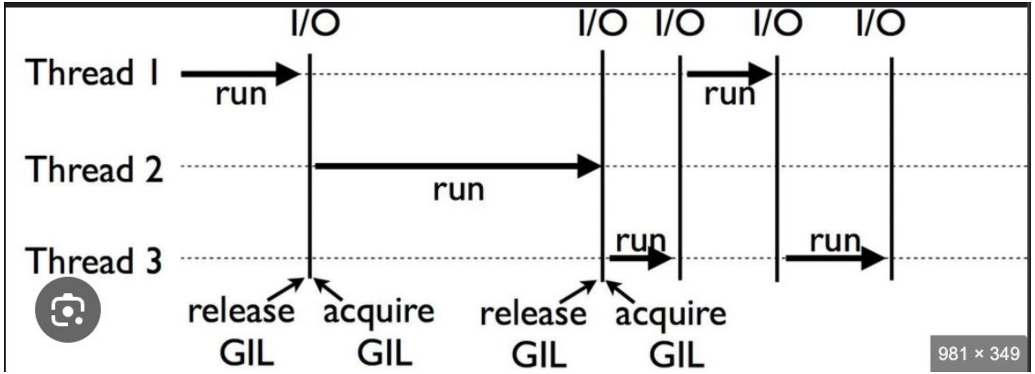


Image1 Source, Image2 Source

multiprocessing

- Python library that allows multiple processes to run on different processors (parallelism).

- True parallelism that bypasses the GIL.

- Used for optimizing performance on CPU-heavy tasks.

- Handles CPU bound tasks. Powerful feature: Pool object allows parallel execution of a function across multiple input values.

Visual Diagram (multiprocessing)

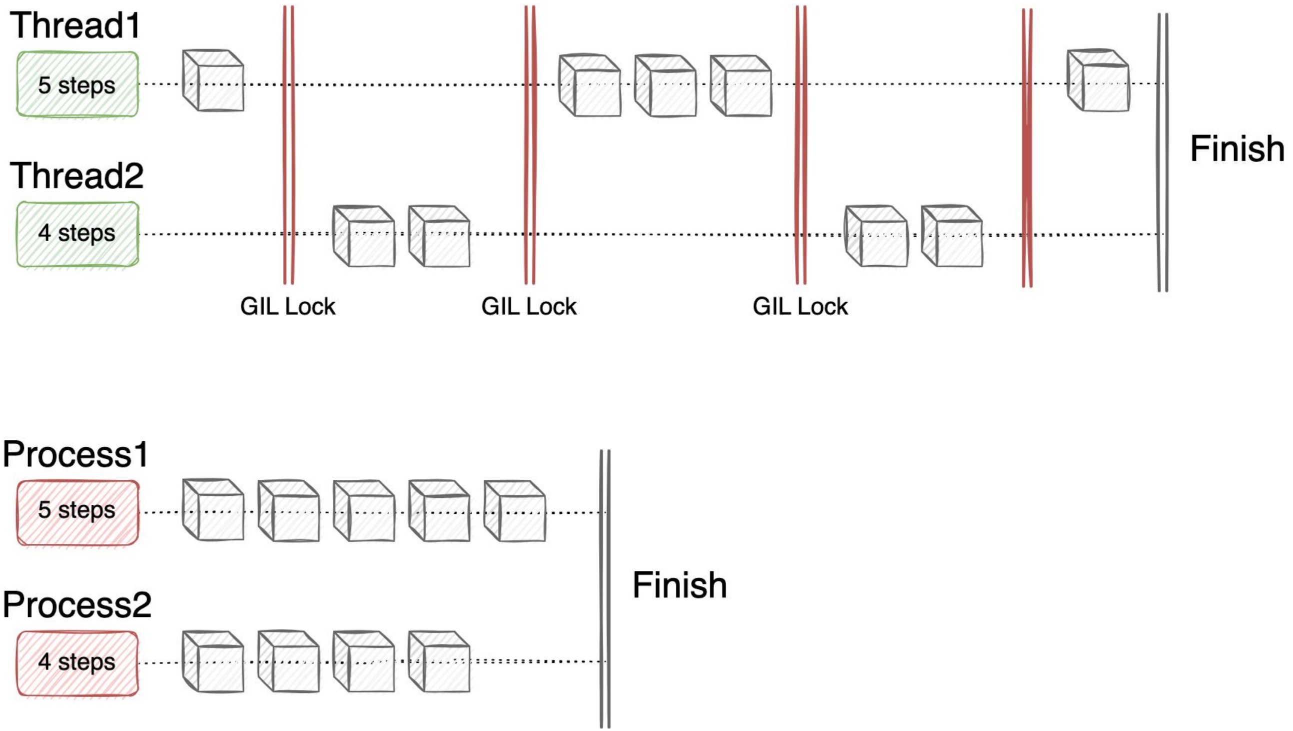


Image Source

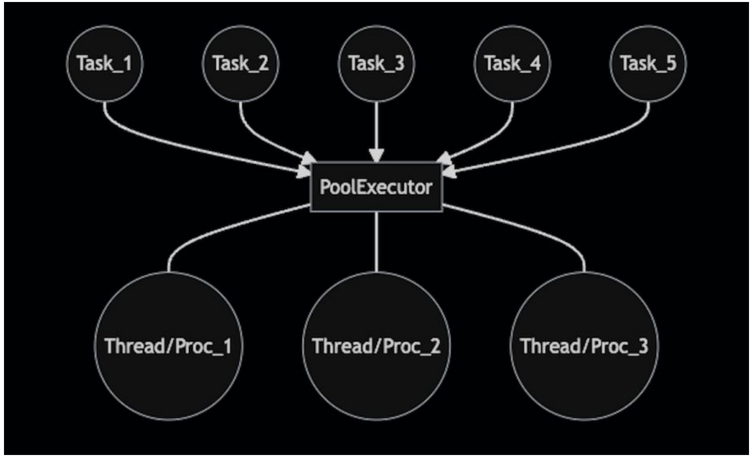
concurrent.futures

- Python library that offers a more modern way to orchestrate multiple *threads* and *processes* in a pool.

- Simpler to manage *threads* and *processes* using concurrent.futures than to do so manually!

- Use when you have synchronous code you want to make concurrent; use when you need true parallelism (i.e. processes running on multiple CPU cores).

Visual Diagrams (concurrent.futures)



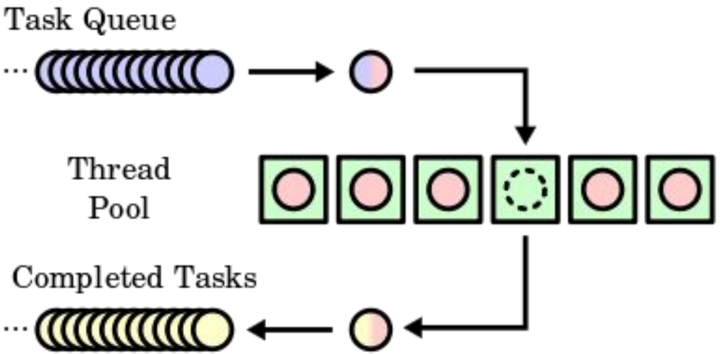


Image1 Source, Image2 Source

When to Use Which Library

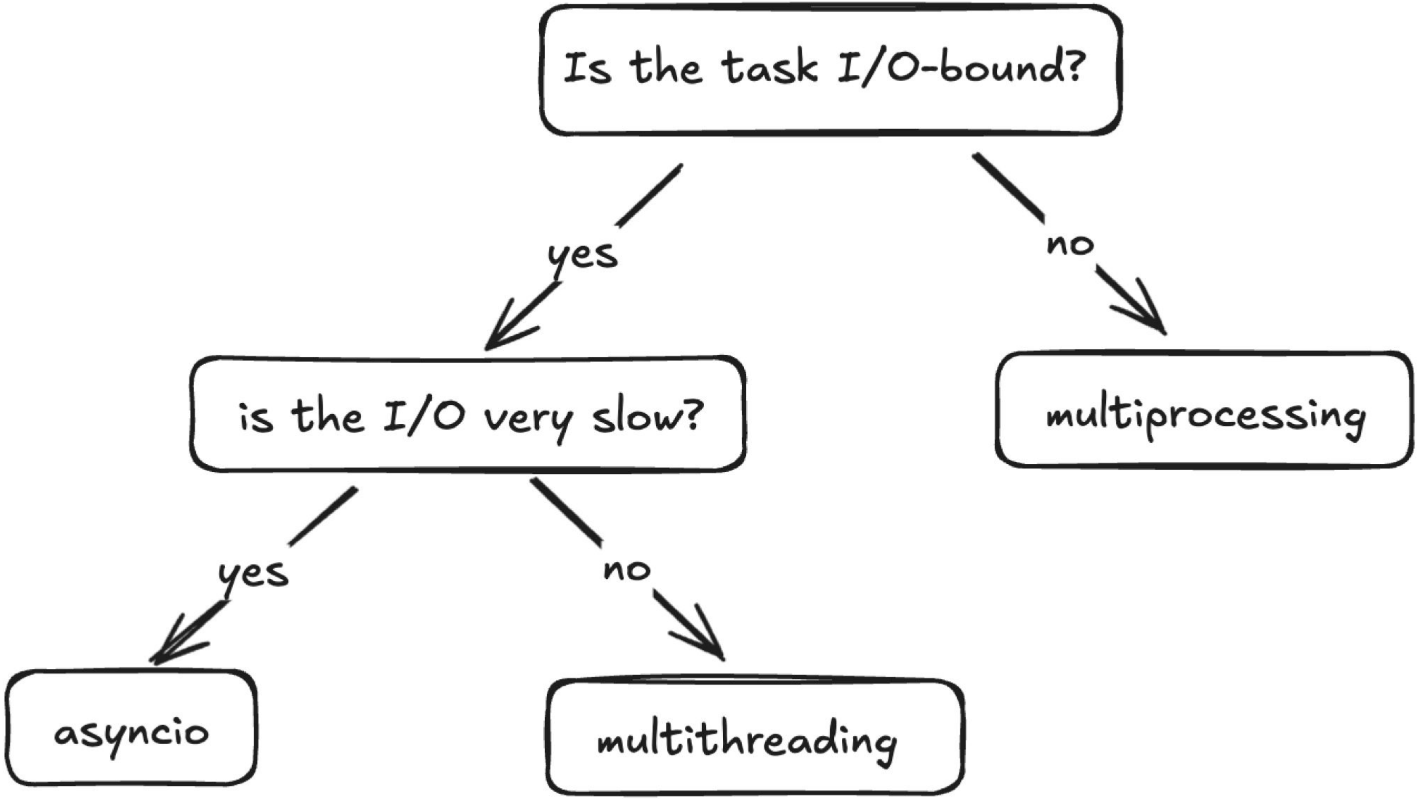


Image Source

When to Use Which Library

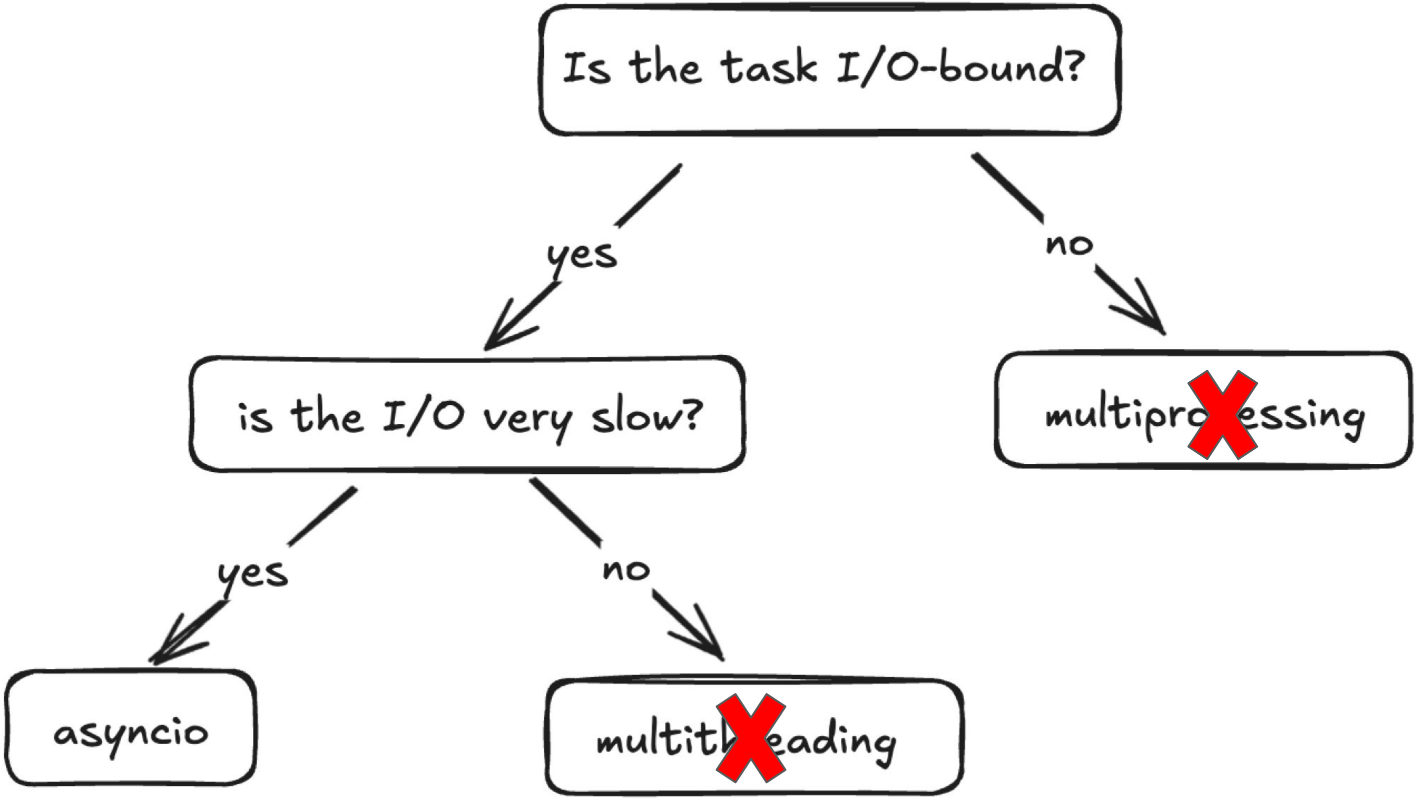
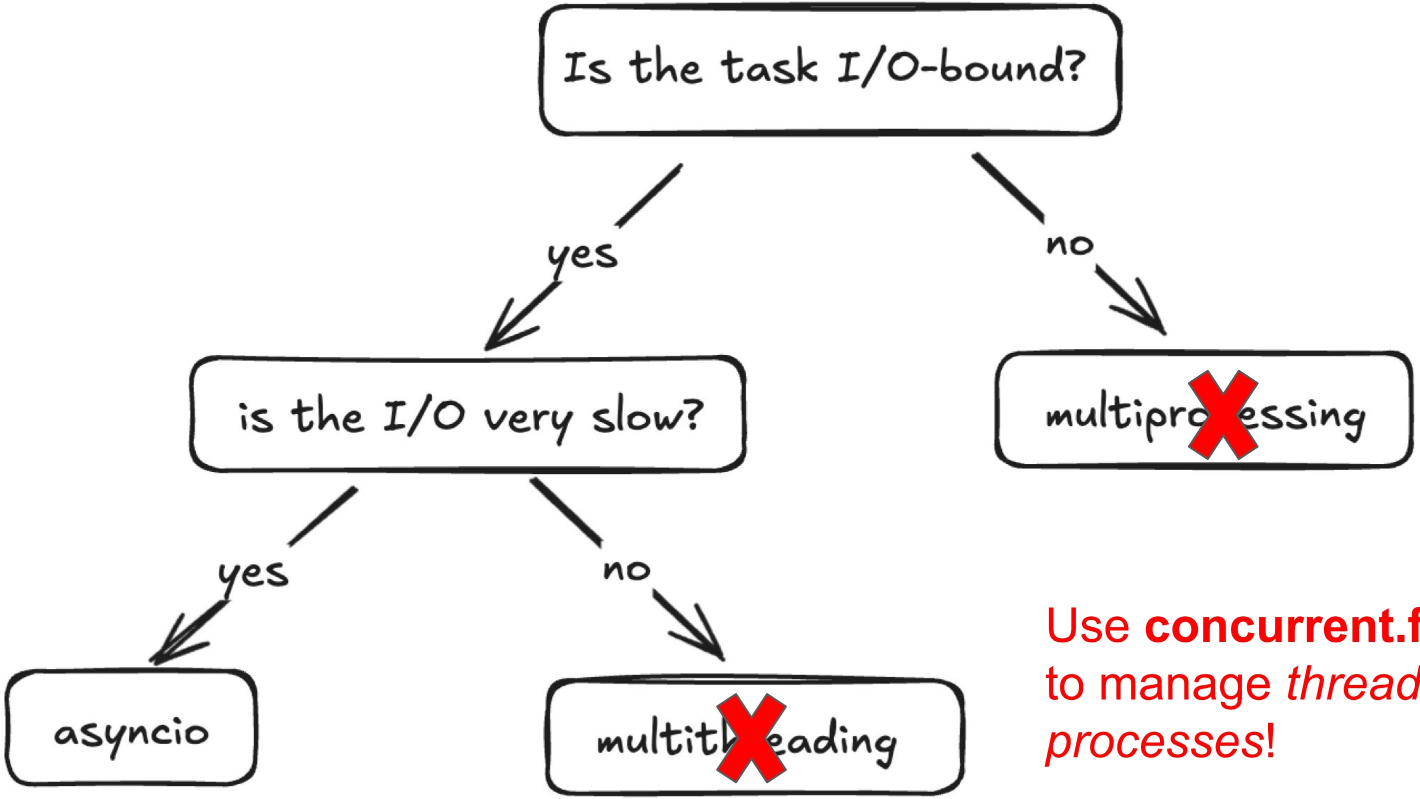


Image Source

When to Use Which Library



Use **concurrent.futures** to manage *threads* and *processes*!

Image Source

DEMOs!

Words of Wisdom (realpython.com)!

Wisdom Key Point #1

“The first step of this process is deciding if you should use a concurrency module… concurrency always comes with extra complexity and can often result in bugs that are difficult to find.”

Wisdom Key Point #2

"Hold out on adding concurrency until you have a known performance issue and then determine which type of concurrency you need. As **Donald Knuth** has said, 'Premature optimization is the root of all evil (or at least most of it) in programming.'"

Wisdom Key Point #3

"Once you've decided that you should optimize your program, figuring out if your program is I/O-bound or CPU-bound is a great next step. Remember that I/O-bound programs are those that spend most of their time waiting for something to happen, while CPU-bound programs spend their time processing data or crunching numbers as fast as they can."

Wisdom Punchline!

“Use asyncio when you can, threading or concurrent.futures when you must.”

THANK YOU FOR YOUR ATTENTION!

Email: **yli12313@vt.edu**
LinkedIn: Yingquan Li
(*I only use LI Messages*)

"In almost every computation a great variety of arrangements for the succession of the processes is possible, and various considerations must influence the selections amongst them for the purposes of a calculating engine." - **Ada Lovelace**

Image Source

References

- Articles:
  - Speed Up Your Python Program With Concurrency
- Class:
  - Speed Up Python With Concurrency
- Documentation:
  - The Python Standard Library Documentation
- Videos:
  - Asyncio in Python - Full Tutorial
  - Python Threading Tutorial: Run Code Concurrently Using the Threading Module
  - Python Multiprocessing Tutorial: Run Code in Parallel Using the Multiprocessing Module