

Stack'D

The Monthly Newsletter

What's
inside?

GLOBAL INTERPRETER LOCK(GIL)

Why GIL and how it affects multi-threading?

DIGITAL PSYCHIC

Do corporations really listen through our devices and recommend ads based on that?

THE SCROLL THAT NEVER ENDS

How to break doomscrolling and regain focus

INTERN INSIGHTS

A summer at Amazon...

AND MORE...



A QUICK REWIND

December usually means holidays, celebrations, and a well-earned pause. But at ACM, slowing down isn't really our thing.

Right after Christmas, we launched the 2nd contest of CODEX, bringing together nearly 200 participants. Kudos to our focused marketing efforts and an enthusiastic community always ready to take on new challenges.



DECEMBER

Following up on Prodigy, we made sure the winners received their certificates and prizes, closing the event on a well-deserved high note.

As we step into a brand-new year, we're excited to unravel many more engaging and inspiring events, filled with learning, collaboration, and innovation. Here's to an even more exciting journey ahead!

TECH DIGEST

THE GLOBAL INTERPRETER LOCK (GIL)

You've probably experienced this frustration before. You write a Python program with multiple threads, expecting it to run faster on your multi-core CPU. You launch it, open your system monitor, and... only one core is working. Meanwhile, a friend writes the same logic in C, and all cores light up. What's going on?

The answer lies in a single mechanism called the Global Interpreter Lock, or GIL.

Concurrency vs Parallelism

Before getting into the details, let's clarify two terms that are often confused.

Parallelism means running multiple tasks at the exact same time on different CPU cores.



Concurrency means handling multiple tasks by switching between them quickly.

Concurrency can happen on one or more cores, but for true parallelism,

Concurrency means handling multiple tasks by switching between them quickly.

Concurrency can happen on one or more cores, but for true parallelism, more than one core is required.

Python supports concurrency, but due to the GIL, it does not support true parallel execution of threads in CPython.

Why Does the GIL Exist?

To understand this, travel back to 1991. Python was created in an era when threading wasn't mainstream and multi-core CPUs didn't exist.

Inside the Python interpreter, there are hundreds of shared internal data structures—such as hash maps storing variables, reference counters, symbol tables, and caches. When multiple threads access these structures simultaneously without protection, race conditions occur.

Example:

Two threads try to add variables to the same internal hash map at the same time. Both think the next slot is position 5. Thread A writes its variable there, then Thread B overwrites it. Thread A's variable is lost.

Later, accessing it throws a "Variable not defined" error even though it was clearly defined.

Two Solutions Existed:

- Fine-grained locking: Protect every data structure with its own lock. Incredibly complex, requires massive rewrites, and is prone to deadlocks.
- One global lock: Use a single mutex for the entire interpreter. Simple, minimal code changes, and guaranteed thread safety.

CPython chose option two.

Why Use the GIL in Interpreted Languages?

The GIL isn't inherent to all interpreted languages; it's specific to CPython. Here's why it was chosen.

In the mid-1990s, operating systems introduced threading APIs. Guido van Rossum wanted Python to support threading without forcing developers into C-level complexity. However, redesigning the interpreter for full thread safety would have required rewriting thousands of lines of core code, adding synchronization logic everywhere, introducing subtle race conditions, and potentially slowing down single-threaded programs.

The GIL offered a pragmatic middle ground. It made sense because:

- Multi-core CPUs were rare, so parallelism wasn't a major concern
- I/O concurrency was valuable, and threads could overlap waiting time
- Simplicity mattered—one lock is easier than dozens of fine-grained locks
- C extensions became easier to write, assuming exclusive interpreter access

This approach worked well for years. But starting in the mid-2000s, multi-core processors became standard, and the GIL turned into a performance bottleneck.

How the GIL Affects Multithreading?

Python threads do exist, but only one thread can execute Python code at a time. The impact depends entirely on the type of workload.

1. CPU-Bound Tasks: The GIL Kills Performance

You spawn four threads on a quad-core CPU, each doing intensive computation. You expect all cores to be fully utilized.

What actually happens looks like this:

- Thread 1 acquires the GIL and executes Python code while other cores sit idle
- The GIL switches to Thread 2, then Thread 3, then Thread 4
- Threads take turns rather than running in parallel

Only one core ever executes Python bytecode at any given moment. There's no speed-up, and sometimes performance even degrades due to context-switching overhead.

In contrast, a C program doing the same work can use all four cores simultaneously and finish in roughly one-quarter the time.

Solutions for CPU-bound work:

- Multiprocessing: Separate processes with separate GILs run in true parallel
- Native libraries: NumPy and pandas release the GIL during heavy computation
- Alternative interpreters: Jython and IronPython do not use a GIL

2. I/O-Bound Tasks: The GIL Doesn't Matter

For workloads involving network requests, file reads, or database queries, the GIL is far less of a problem.

When a thread makes a blocking I/O call, the interpreter releases the GIL, allowing other threads to run.

While one thread waits for data, others continue executing. This overlap of waiting time provides real concurrency benefits.

Where threading works well:

- Web servers handling multiple client connections
- Web scrapers fetching multiple URLs
- API clients making simultaneous requests
- Database applications running multiple queries

Conclusion:

The Global Interpreter Lock exists to protect CPython's shared internal state using a single, simple guarantee. This design made threading safe and practical long before multi-core CPUs were common.

The GIL isn't mysterious—it's an old design decision that continues to enforce its original rules in a modern, multi-core world.

BUST THE MYTH

MYTH: "MY PHONE LISTENS TO MY CONVERSATIONS FOR ADS"

Common Experience:

You're telling a friend you need a new laptop. You haven't searched for it. You haven't typed it. Yet, twenty minutes later, you open Instagram and boom, an ad for the exact MacBook you discussed. The conclusion seems obvious: "Zuckerberg is listening".

Reality:

Tech companies don't need to listen to you. In fact, recording 24/7 audio, uploading massive .wav files to the cloud, and processing natural language for billions of users would be a computational nightmare, expensive, battery-draining, and data-heavy. The truth is far more impressive and terrifying. Their predictive models are so mathematically accurate, they know what you want before you even articulate it.

How the "Digital Psychic" Actually Works:

1. The "Social Graph" & Location Mirroring

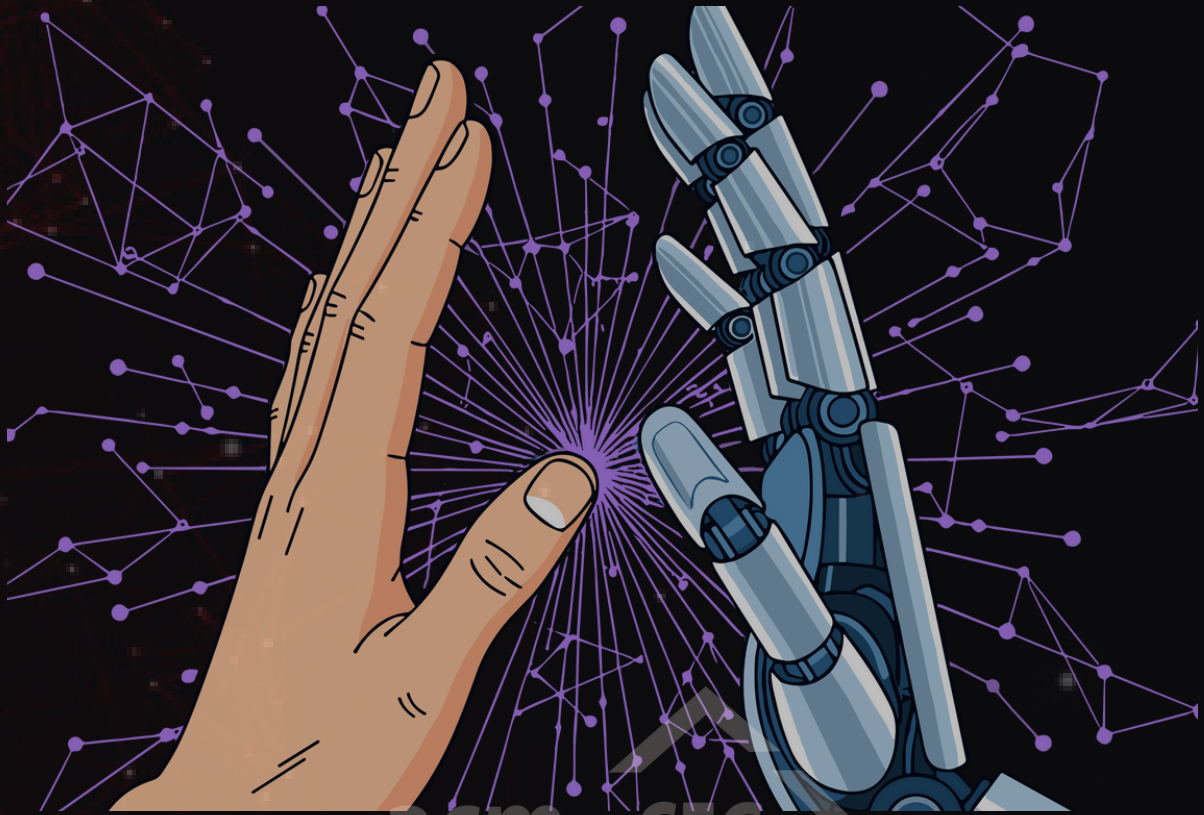
- Advertisement algorithms rely heavily on who you are physically near.

If your phone's GPS places you next to a friend for 45 minutes, the algorithm links your profiles temporarily..

- Scenario: Your friend spent the last week searching for "gaming laptops." When you meet him, the algorithm thinks, "User A (Friend) influences User B (You). User A wants a laptop. Therefore, User B might now want a laptop."
- You see the ad not because you spoke, but because your friend searched for it before you even met

2. Shadow Data & "Off-Platform" Activity

- It's not just about what you like on Facebook. It's about where you've been on the rest of the web. Almost every website you visit has a Meta Pixel or Google Tag embedded in it.
- **Prediction:** You didn't search for "vacations," but you paused for 3 seconds on a photo of a beach on Instagram (Dwell Time) and bought sunscreen on Amazon yesterday. The AI combines these disconnected data points to assign you a "High Probability Travel" score, triggering flight ads immediately.



3. The Frequency Illusion (Baader-Meinhof Phenomenon)

- This is a glitch in your brain, not your phone. You are exposed to roughly around hundreds of ads every single day. Your brain filters out 99% of them as "noise."
- When you talk about something (like "running shoes"), that topic moves to your brain's "active attention" list. Suddenly, you notice the shoe ad that was already there yesterday, but you ignored it then. The algorithm is throwing random darts but you only remember the one that hits the bullseye.

Conclusion:

"You aren't being recorded. You are being simulated."

Companies have built a "Digital Twin" of you based on your clicks, swipes, and location history. This simulation is so accurate that it doesn't need to hear your voice to know what you're thinking, it simply calculates the probability of your next desire.

"They didn't hear you say it. They calculated that you were about to."

ALGO OF THE MONTH

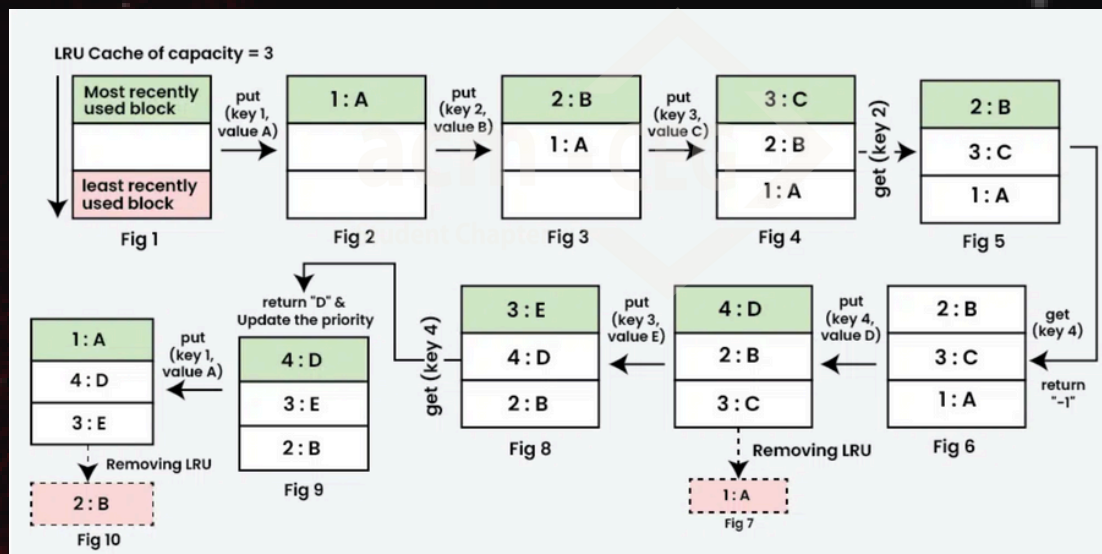
LRU CACHE ALGORITHM

Have you ever opened Spotify and felt like it somehow knew what song you wanted next? It is not because the app knows you personally. It is because it remembers what you used recently.

This simple idea brings us to the **LRU Cache Algorithm**, short for Least Recently Used.

LRU is all about deciding what to forget when space is limited. When a system cannot store everything, it does not remove data randomly. Instead, it removes the data that has not been used for the longest time.

How does LRU work?



Think about your study table with very little space.

The books you use often stay right in front of you. The books you have not touched in days slowly move to the side. When the table becomes full, the book you used the least recently is the first one to be removed.

- LRU follows the exact same logic. Every time some data is accessed, it is marked as recent. When memory becomes full, the least recently accessed data is removed. This way, the system keeps the most useful data close by.
- Today, LRU works quietly inside many systems we use every day.

Coming back to Spotify, they have developed their own Cache Library called `SPTPersistentCache` which utilized LRU eviction strategy under the hood. LRU helps the app stay fast and responsive. Songs, playlists, and recently played tracks are cached so they load instantly when you open the app. When the cache becomes full, tracks you have not listened to in a long time are removed, making space for the ones you play often.

SPOTIFY RECOMMENDATION ALGORITHM

The diagram illustrates the Spotify Recommendation Algorithm architecture, showing the flow of data from user interactions to the final recommendation output.

Input Data Sources:

- Access Point (User Interactions)
- Radio
- Artist Page
- Playlist
- Discover

Data Processing and Storage:

- Kafka:** Acts as a central data hub, receiving data from the input sources and distributing it to the Recommendations Pipeline.
- HDFS (Hadoop Distributed File System):** Stores large volumes of data, including:
 - Play Logs
 - Track Metadata
 - News, Blogs, and Text
- Recommendations Pipeline:** Processes data from Kafka and HDFS to generate recommendations. It utilizes:
 - Batch CF Models (Collaborative Filtering)
 - Batch NLP Models (Natural Language Processing)
 - Batch Audio Models
- Cassandra:** A distributed database used for storing user profiles, playlist data, and recommendation results.

Output:

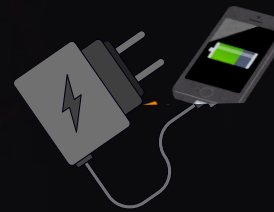
- Discover Weekly:** The final recommendation playlist generated by the system.

TIP OF THE MONTH

Assignments, exams, projects. All lined up? But the brain asks for excuses. Just 5 minutes, 10 minutes, and you find that your whole day is already gone. Days roll by and you scroll through the sun and the moon yet those 5 minutes never come to an end. Sounds familiar? Then here are some tips to get rid of doomscrolling.

1. Turn your phone screen to grayscale

Colors are addictive. Grayscale makes the brain bored faster. Hence, scrolling reduces naturally.



2. Keep social apps off your home screen

Cutting out these extra hovers over social media apps saves a lot of time.

3. Use one alarm, not an infinite number of alarms

Multiple snoozes train your brain to ignore alerts giving you procrastination over your purpose.



4. Pick your charging spots wisely

Have a charging port usually away from the bed. This alone reduces a lot of scrolling before sleep and helps in better sleep as well



5. Use "Do Not Disturb" as a ritual

Turn on "Do Not Disturb" every day only during a specific time period instead of random hits. This helps create a mental boundary rather than temporary cut-offs.

CAREER CORNER



I am Prasanna Kumar, a final-year Computer Science and Engineering student at CEG, and I had the amazing opportunity to intern at Amazon as a Software Development Engineering Intern during Summer 2025. It was truly a great learning experience that helped me understand how software systems work in the real world beyond what we study in college.

During my internship, I worked on improving and automating some of Amazon's internal tools that help teams monitor and manage data jobs efficiently. I also helped redesign a few dashboards that display important performance metrics, making them easier to understand and more accurate. One of the interesting parts of my work was setting up automatic alerts and notifications — these alerts inform teams if something goes wrong or takes longer than expected, helping them fix issues quickly. Through these tasks, I learned how large-scale systems are maintained and how even small improvements can make a big difference.

The office environment was very friendly and motivating. My mentor was extremely helpful and guided me whenever I faced challenges. My manager and team members always encouraged me to explore new ideas and solve problems on my own. My peers were also very supportive. Discussing ideas and learning together made the experience even more enjoyable.

One of my biggest learnings was the importance of Amazon's "Dive Deep" principle — always try to understand the problem fully before jumping to a solution. Also, networking and teamwork matter a lot. Talking to people from different teams helped me gain new perspectives on how things work inside such a big organization.

To all my juniors and readers, I would say — stay curious and never hesitate to ask questions. Try to solve problems independently before seeking help, but don't be afraid to reach out when you need support. Learn to communicate, collaborate, and take ownership of your work. These qualities will help you grow not just as a coder but as a complete engineer.

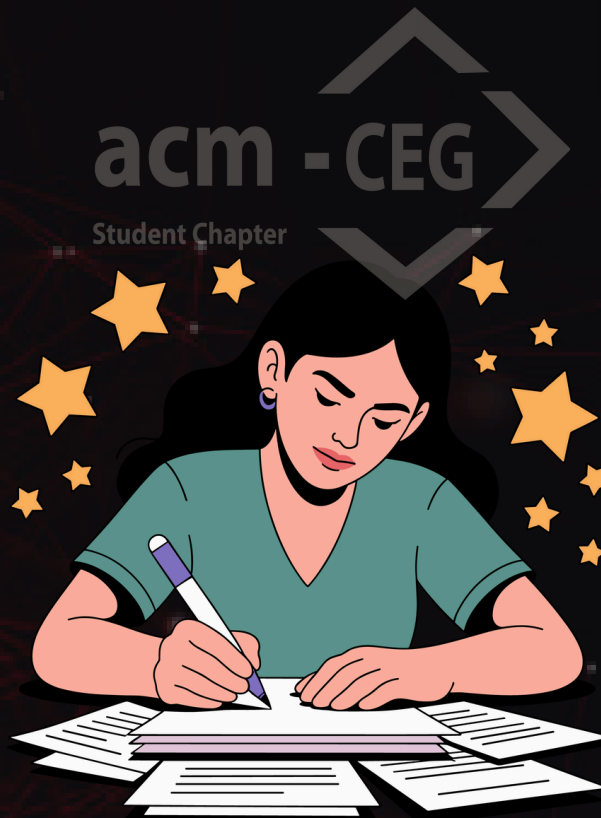
READER'S AREA

Have a fresh story to tell? An experience to share? Or maybe an unique idea that you just thought randomly?

Whether it's a project you're proud of, insights from a tech trend, or even a quirky take on student life - We'd love to hear from you! This space is for our community, by our community.

Send us your write-ups, articles, or even short reflections. Who knows? Your piece might just inspire, inform, or entertain hundreds of readers in the next issue of our newsletter.

Share your content with us (acm.ceg2025@gmail.com) and become a part of our growing conversation!





ACM Student Chapter of CEG



auceg.acm.org

Credits

Content Contributors:

Prasanna Kumar, Shree Vaishnavi K, Roopa Varshini, Sanjay Kumaran

Editor:

Niranjan K

Design & Layout:

Gabriella D, Kavinyaya P

References:

<https://youtu.be/UXwoAKB-SvE?si=yemPZLW7gF5Ociln>