



## What is a Virtual Thread?

A **Virtual Thread** (introduced in Java 21) is a lightweight thread managed by the **JVM**, not the Operating System (OS).

- **Platform Thread (Classic):** 1-to-1 mapping with an OS thread. Heavy (2MB stack), slow to start, limited in number (you can create maybe 10,000 before crashing).
- **Virtual Thread (New):** M-to-N mapping. Millions can exist on a few OS threads. Light (KB stack), cheap to create.

It solves the **"Thread-per-Request" scalability problem**. In the past, if you had 10,000 concurrent users, you needed 10,000 OS threads (impossible). With Virtual Threads, you can have 10 million virtual threads running on just 10 OS threads.<sup>[1] [2]</sup>

## How it Works (In-Depth with "Diagram")

Imagine a **Taxi Company (The JVM)**.

1. **Platform Threads (Carrier Threads):** These are the **Cars**. You only have a few (e.g., 10 cars/cores).
2. **Virtual Threads:** These are the **Passengers**. You have millions of them.

## The "Mounting/Unmounting" Magic

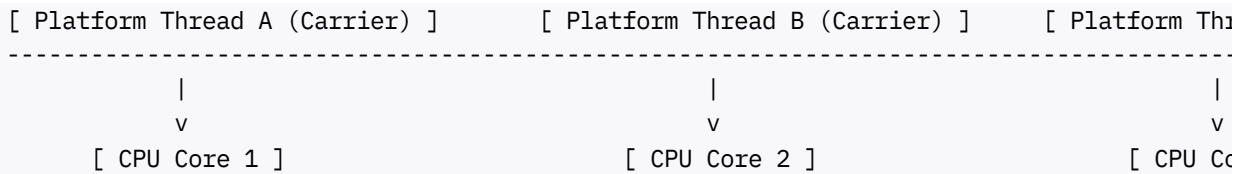
When a Virtual Thread needs to run CPU code, the JVM **"mounts"** it onto a Platform Thread (Passenger gets into a Car).

When the Virtual Thread does **I/O** (e.g., calls database, waits for API response), it **"unmounts"** (Passenger gets out of the Car).

- **Old Way (Platform Threads):** The Car waits at the destination (Database) for the result. The engine runs idle. **Wasteful**.
- **New Way (Virtual Threads):** The Passenger (Virtual Thread) gets out of the Car while waiting for the Database. The Car (Platform Thread) immediately picks up **another Passenger** (another Virtual Thread).<sup>[3] [4]</sup>

## Visual Diagram (Mental Model):

```
[ Virtual Thread 1 (Waiting for DB) ]   [ Virtual Thread 2 (Ready) ]   [ Virtual Thread 3 (Waiting for DB) ]
      |                                   |                                   |
      | (Unmounted - Stored in Heap)     | (Mounted)                       |
      v                                   v                                   v
-----
```



## Java Code Example

You can use `Executors.newVirtualThreadPerTaskExecutor()` to let the JVM create a new virtual thread for *every single task*.

```
import java.util.concurrent.Executors;

public class VirtualThreadDemo {
    public static void main(String[] args) {

        // Old Way: Executors.newFixedThreadPool(100) -> Limited threads
        // New Way: Creates a new virtual thread for EVERY task
        try (var executor = Executors.newVirtualThreadPerTaskExecutor()) {

            for (int i = 0; i < 100_000; i++) { // Launching 100,000 threads!
                int taskId = i;
                executor.submit(() -> {
                    // Simulate IO operation (e.g., DB call)
                    try {
                        Thread.sleep(1000); // The virtual thread unmounts here!
                    } catch (InterruptedException e) {
                        e.printStackTrace();
                    }
                    System.out.println("Task " + taskId + " finished on " + Thread.currentThread().getName());
                });
            }

            // Executor auto-closes and waits for all tasks to finish
        }
    }
}
```

### Output Insight:

You will see output like:

```
Task 1 finished on VirtualThread[#21,]/runnable@ForkJoinPool-1-worker-1
```

Notice `ForkJoinPool-1-worker-1`. That is the **Carrier Thread** (Platform Thread). Even though we have 100,000 tasks, they all run on a small pool of worker threads (equal to your CPU cores).<sup>[5]</sup>  
<sup>[3]</sup>

## Why is this a game changer?

You don't need complex Reactive Programming (WebFlux/RxJava) anymore to handle high concurrency. You can write simple, readable, blocking code (`db.query()`), and the JVM makes it non-blocking under the hood!<sup>[6]</sup>

1. <https://jenkov.com/tutorials/java-concurrency/java-virtual-threads.html>
2. <https://blog.ycrash.io/an-investigative-study-virtual-threads-vs-platform-threads-in-java-23/>
3. <https://rockthejvm.com/articles/the-ultimate-guide-to-java-virtual-threads>
4. <https://engineeringatscale.substack.com/p/what-are-java-virtual-threads>
5. <https://github.com/aliakh/demo-java-virtual-threads>
6. <https://stackoverflow.com/questions/78318131/do-java-21-virtual-threads-address-the-main-reason-to-switch-to-reactive-single>
7. <https://ppl-ai-file-upload.s3.amazonaws.com/web/direct-files/attachments/images/81815274/87f62423-96f2-4071-9802-8f6699e0ecd8/image.jpg>
8. <https://www.geeksforgeeks.org/java/virtual-threads-in-java/>
9. <https://docs.oracle.com/en/java/javase/21/core/virtual-threads.html>
10. [https://www.reddit.com/r/java/comments/1btc92e/how\\_virtual\\_threads\\_gets\\_created\\_in\\_java/](https://www.reddit.com/r/java/comments/1btc92e/how_virtual_threads_gets_created_in_java/)
11. <https://pwrteams.com/content-hub/blog/modern-java-unleashed-virtual-threads-other-game-changing-features-in-jdk-21>
12. <https://openjdk.org/jeps/444>
13. <https://dev.to/24adithya/java-virtual-threads-explained-continuations-carrier-threads-and-scalable-concurrency-2dk5>
14. <https://gist.github.com/matteobertozzi/a6715afc1fe878045a8e1e861e122c12>
15. <https://blog.ycrash.io/java-virtual-threads-quick-introduction/>
16. <https://stackoverflow.com/questions/78094984/difference-between-platform-thread-carrier-thread-and-virtual-thread-in-context>
17. <https://netflixtechblog.com/java-21-virtual-threads-dude-wheres-my-lock-3052540e231d>
18. <https://docs.oracle.com/en/java/javase/20/core/virtual-threads.html>
19. <https://stackoverflow.com/questions/79130627/when-to-use-platform-threads-over-virtual-threads>
20. <https://inside.java/2021/05/10/networking-io-with-virtual-threads/>
21. <https://dzone.com/articles/java-concurrency-evolution-virtual-threads-java21>
22. <https://ppl-ai-file-upload.s3.amazonaws.com/web/direct-files/attachments/images/81815274/429036f4-3bc3-4a7a-8c6f-90246e73b9b9/image.jpg>
23. <https://ppl-ai-file-upload.s3.amazonaws.com/web/direct-files/attachments/images/81815274/24f18c0a-b56e-4862-a619-e2959036a5c2/image.jpg>
24. <https://ppl-ai-file-upload.s3.amazonaws.com/web/direct-files/attachments/images/81815274/97e9ed86-ed11-4035-a18b-1eb05c1bab4e/image.jpg>