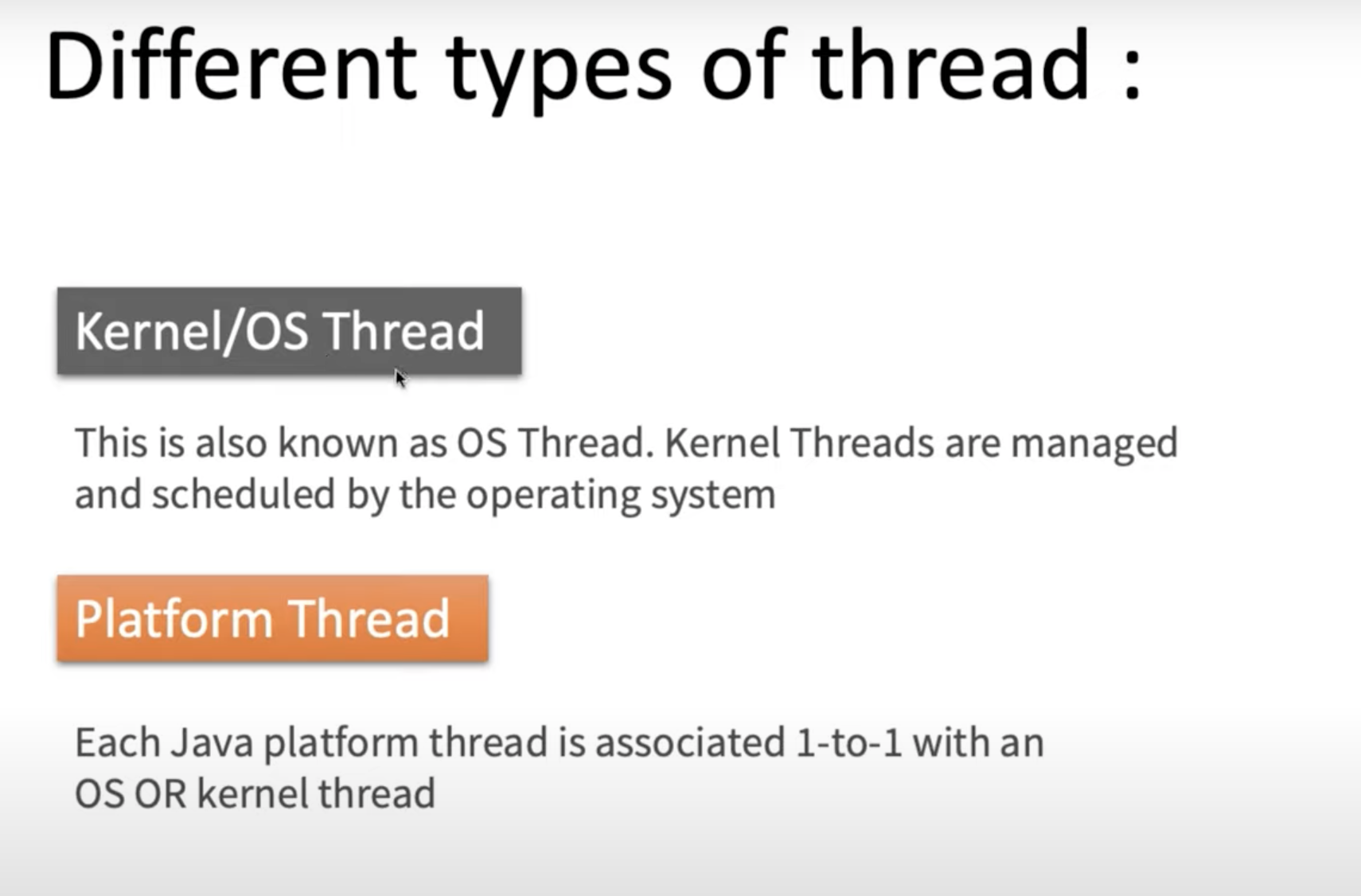
**Virtual Thread**



When we create a thread using **new Thread()** it creates platform-thread and it makes 1:1 mapping with operating system thread to perform any task.

Operating system thread is the one which executes every instruction in the computer system.

## 

**👍 Thread Life Cycle with Platform Thread Pool Below**

**Prior to Java 19: Threading Concepts**

* **Thread-per-Request Model:** Each incoming request was handled by a dedicated thread from a pool.
* **Platform Threads:** These heavyweight threads, tied directly to operating system threads, were expensive to create and manage.
* **Context Switching:** Frequent context switching due to thread blocking (e.g., I/O operations) wasted resources.
* **Limited Scalability:** Excessive threads could exhaust resources and hinder performance.

## 

**Problems with the Traditional Approach:**

* **Wasted OS Resources:** Threads waiting in the pool unnecessarily locked down OS threads.
* **Inefficient Memory Usage:** Threads consume memory even while idle, leading to potential waste.

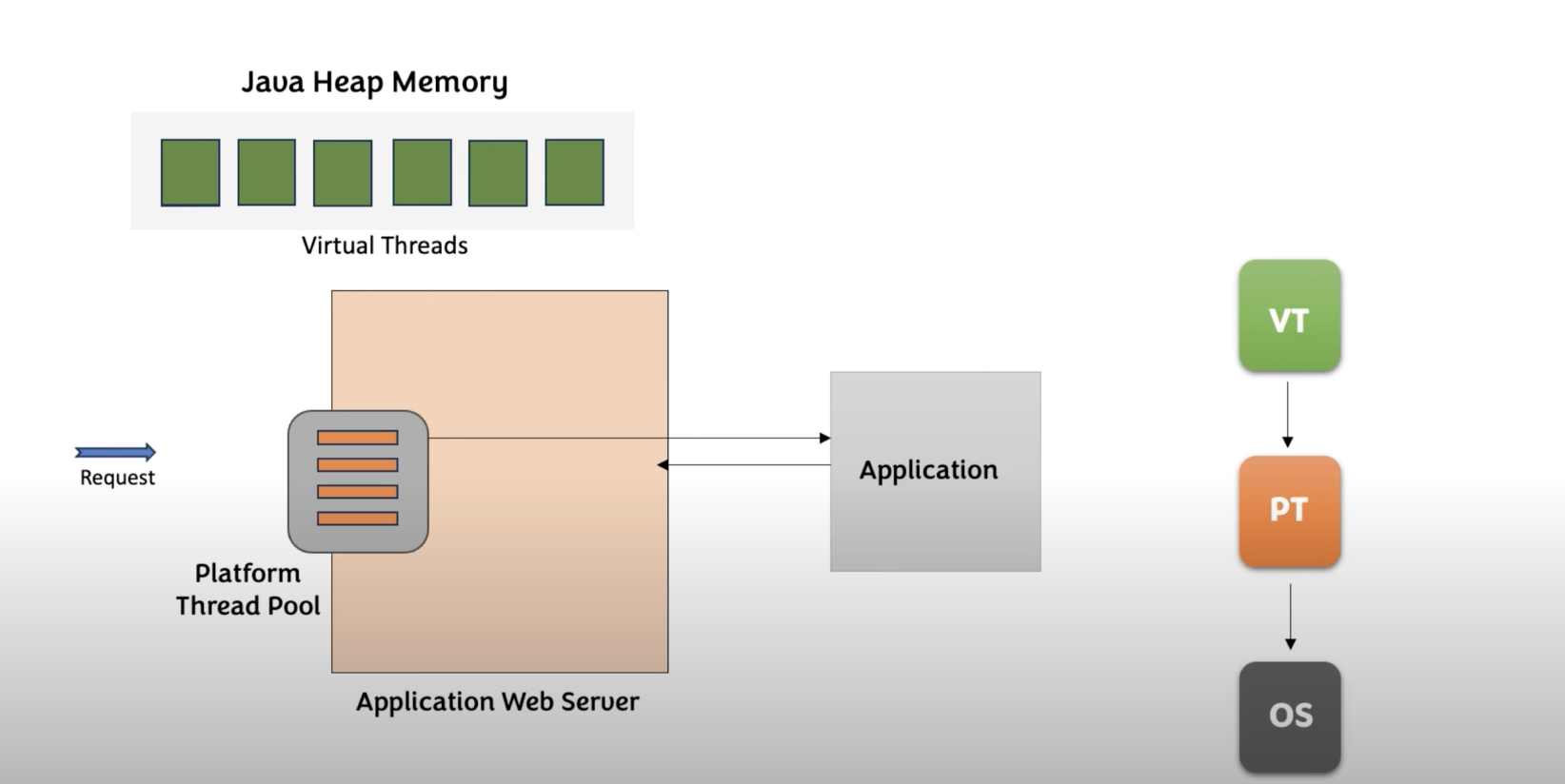
**Thread Life Cycle below with Virtual Thread Pool**

**Virtual Threads (Introduced in Java 19)**

* **Lightweight Threads:** More efficient memory usage compared to platform threads.
* **Enhanced Scalability:** Enables creating and managing a large number of threads without significant resource constraints.
* **Efficient I/O Handling:** When a virtual thread encounters I/O blocking, the JVM switches it with another ready virtual thread on the same OS thread, maximizing efficiency.

**Key Points:**

* Virtual threads reside in the Java heap, not directly tied to OS threads.
* Virtual threads are mapped to platform threads only when executing "real work."
* OS threads are not allocated to platform threads until real work needs to be executed.



If there is no real work being executed, all threads are detached from each other Virtual Thread gets detached from the platform thread and then platform thread will be detached from the OS thread.

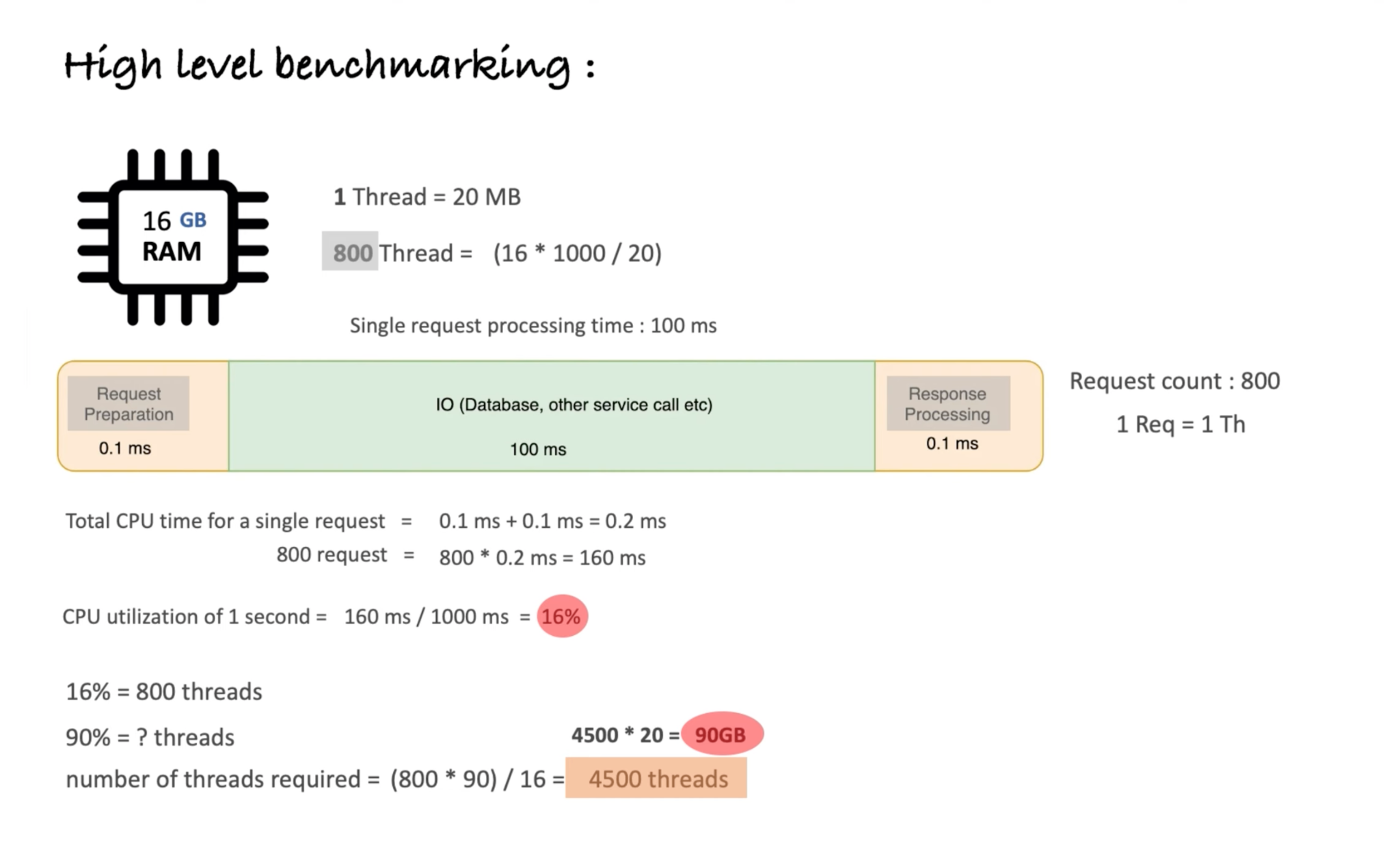
## 

## 

**Benefits:**

* **Improved CPU utilization and throughput.**
* **Enhanced memory management.**
* **Simplified concurrent programming.**

The below calculation is for Platform thread you can see to run 4500 threads we need around 90 gb memory but in case of virtual thread this is not the case virtual thread only takes bytes to be stored in heap area.



**Example Calculation:**

* **Total RAM:** 16 GB
* **Platform Thread Memory Footprint:** 20 MB
* **Maximum Threads (Assuming all memory is used for threads, which is unrealistic):** 16 GB / 20 MB/thread = 800

1 thread is taking 20 MB of memory. So, with a 16 GB RAM capacity, how many threads can my system accommodate?

Given:

- 16 GB RAM = 16 \* 1024 MB = 16,384 MB

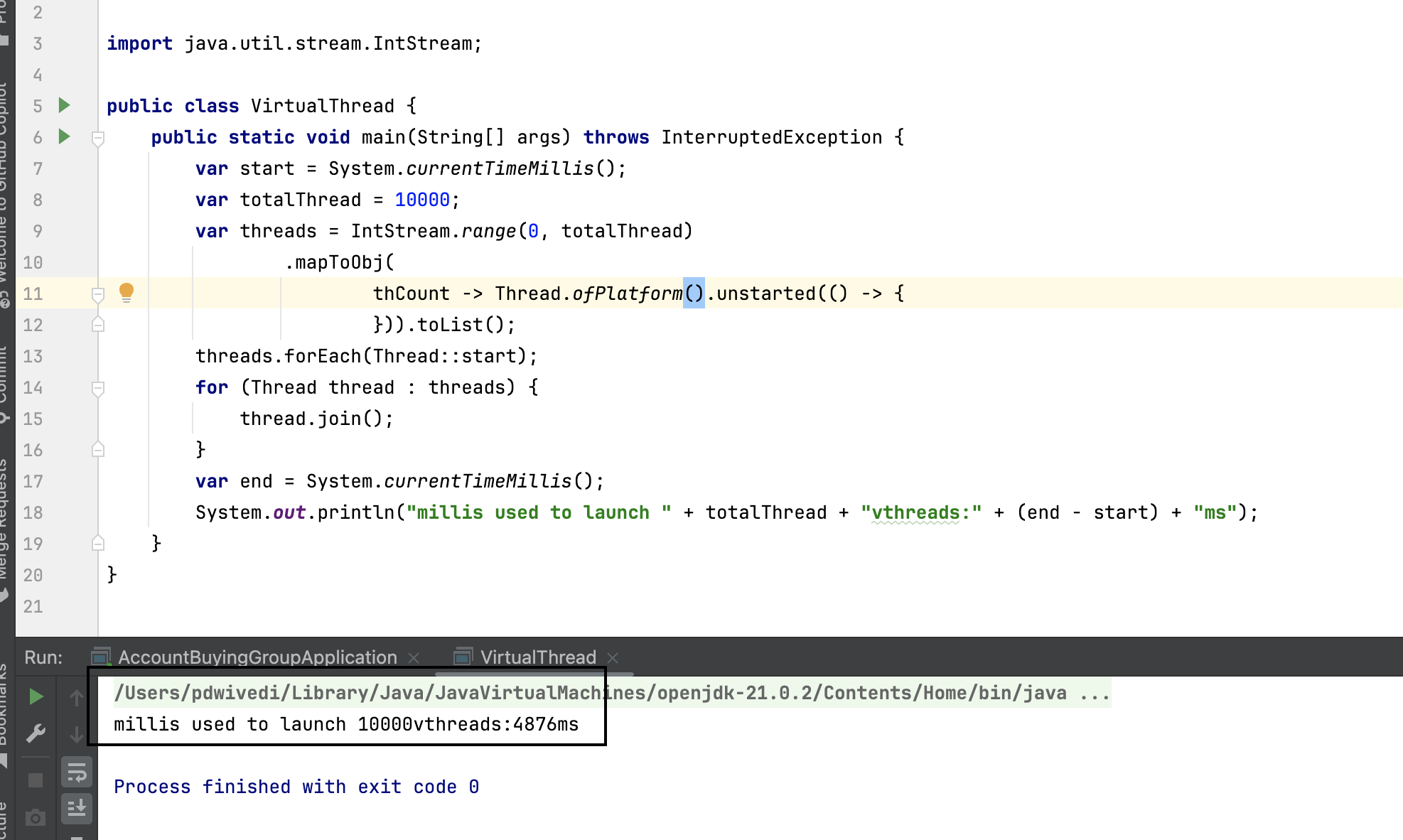
- Memory per thread = 20 MB

Total threads = 16,384 MB / 20 MB ≈ 819 threads.

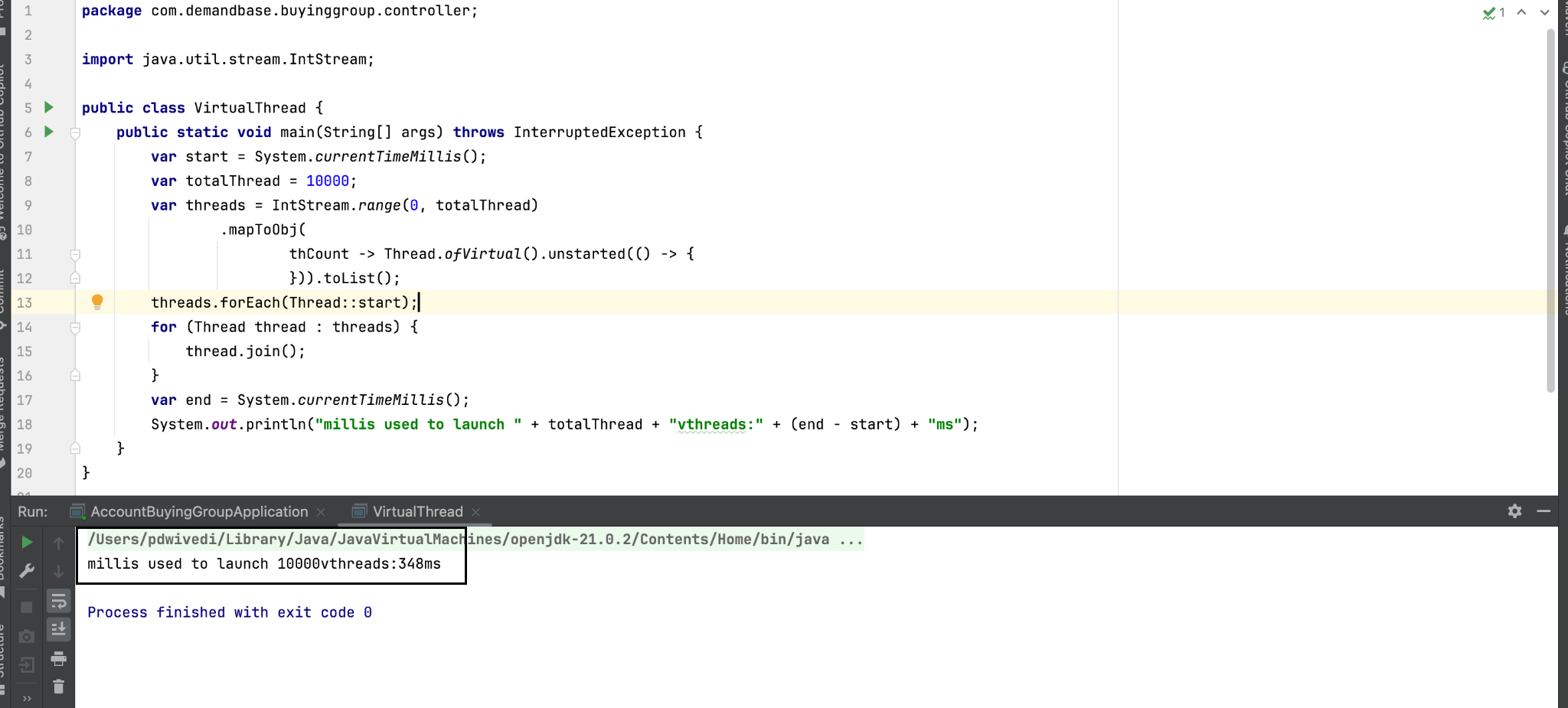
Therefore, my system can accommodate approximately 819 threads with 16 GB of RAM.

**Virtual threads can handle significantly more concurrent requests due to their lightweight nature and efficient resource utilization.  
  
  
  
Java Program Below to demonstrate how much responseTime will be saved when we use Virtual Thread**

**Code using platform thread and it’s response time below:**

****

**Code using Virtual thread and it’s response time below:**

****

**So, you can clearly see the difference between platform and Virtual thread in platform thread time taken to process 10000 thread is 4876 ms and in virtual thread it’s only 348 ms.**

**To enable Virtual Threads in Spring Boot, you can add the following key-value pair in the application.properties file:**

**spring.threads.virtual.enabled=true**

**If you want to compare the difference, you can comment out the above property in the application.properties file and hit one of your APIs. Then, you can run the following command to observe the difference:**

**ab -n 1600 -c 400 {{yourAPIEndpoint}}**

**For example:**

**ab -n 1600 -c 400 http://localhost:8081/buyingGroup/healthCheck**

**Run the above command in the terminal after adding and removing the property from the .properties file, and observe the results:**

**Time taken for tests: 5.842 seconds**