**Q) You receive an issue ticket stating that the game FPS has drastically reduced, and we are not achieving the desired 11ms frame time. How would you systematically approach diagnosing and fixing this issue?**

My first instinct is to verify and reproduce the issue. If the problem is consistent across different hardware configurations then it is likely caused by something within the game itself. However, if I fail to reproduce the issue, then it is likely a problem outside the scope of the developer.

For sake of discussion, let's assume that the problem is reproducible and across different hardware configurations we are not hitting the target FPS.

In this case, I would first check **where the frame time is being spent**—whether we are **GPU-bound or CPU-bound**.

- If working in **Unreal Engine**, I'd start by enabling the **stat unit** command to break down frame timings (Game, Draw, GPU). This would immediately tell me which subsystem is the culprit.
- In **Unity**, I would check the **Profiler** to identify whether the CPU is spending too much time in game logic or whether rendering calls are the primary issue.

Once I have a general idea of whether the game is CPU- or GPU-bound, I would drill down further into the specific subsystems.

<u>If the game is CPU-bound:</u>

The first thing I'd check is whether the **main thread** is overwhelmed. I would:

- Look for expensive **tick functions** running every frame.
- Profile **physics calculations**—large numbers of rigid bodies, complex collision checks, or high-frequency updates can cause slowdowns.
- Inspect **pathfinding and AI logic**—pathfinding on a large number of agents can cause severe performance degradation.
- Examine **memory allocations**—frequent allocations and deallocations lead to garbage collection hitches. Object pooling should be used for frequently spawned entities like bullets, particles, or enemies.

<u>If the game is GPU-bound:</u>

If the GPU is struggling to maintain frame time, it's usually due to inefficient rendering. Here's how I'd approach it:

- **Check the number of draw calls**—excessive draw calls indicate that the engine is not batching objects efficiently.
- **Analyze shader complexity**—fragment shaders with heavy computations can slow down rendering, especially if overdraw is high. If a significant portion of the screen is covered by transparent objects, **sorting and limiting transparent rendering passes** is necessary.
- **Optimize LODs**—high-polygon models being rendered at all distances is a common mistake.
- **Reduce dynamic shadow calculations**—real-time shadows can be a massive GPU burden.