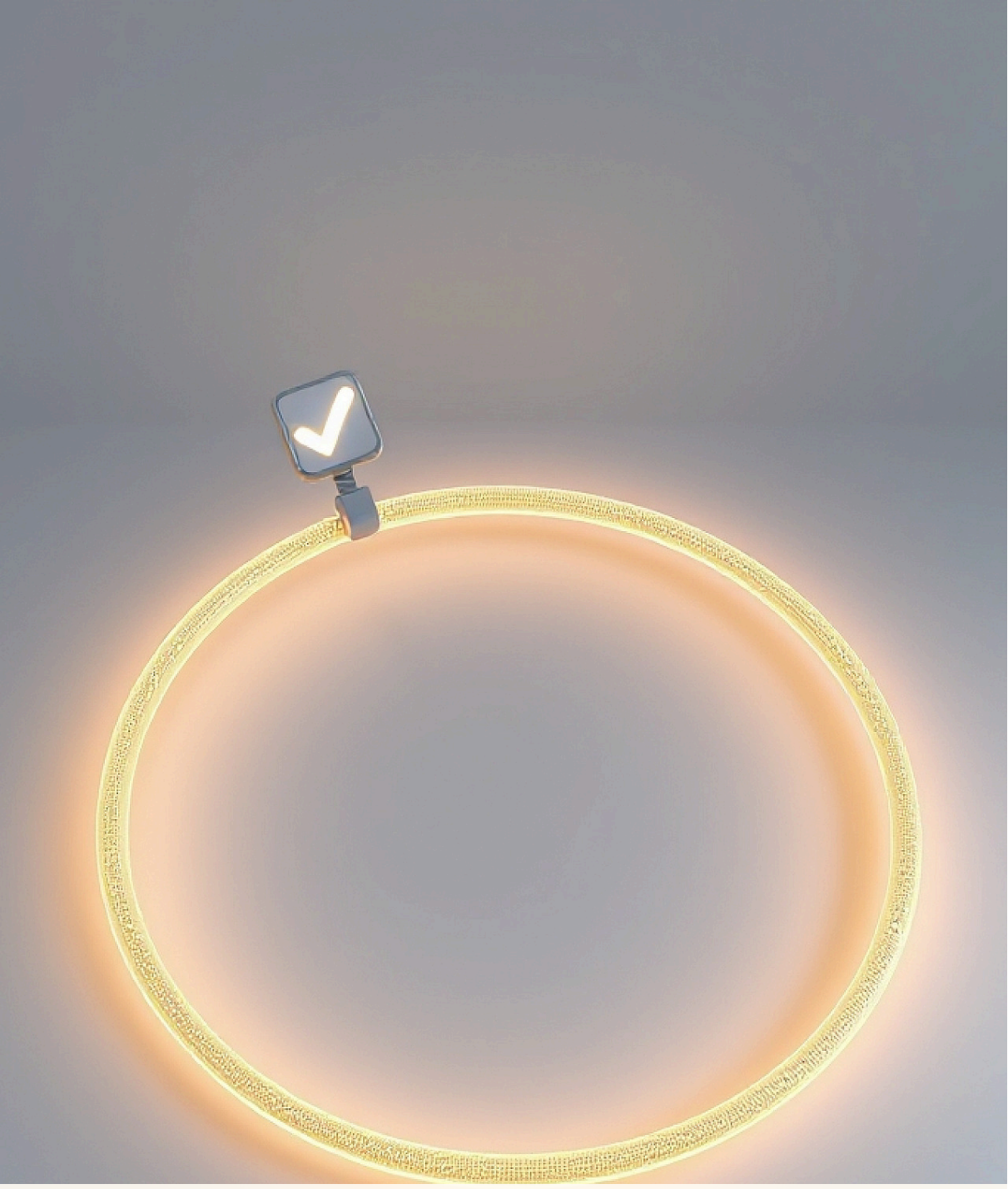
What is an infinite loop?



www.yzokul.com.tr

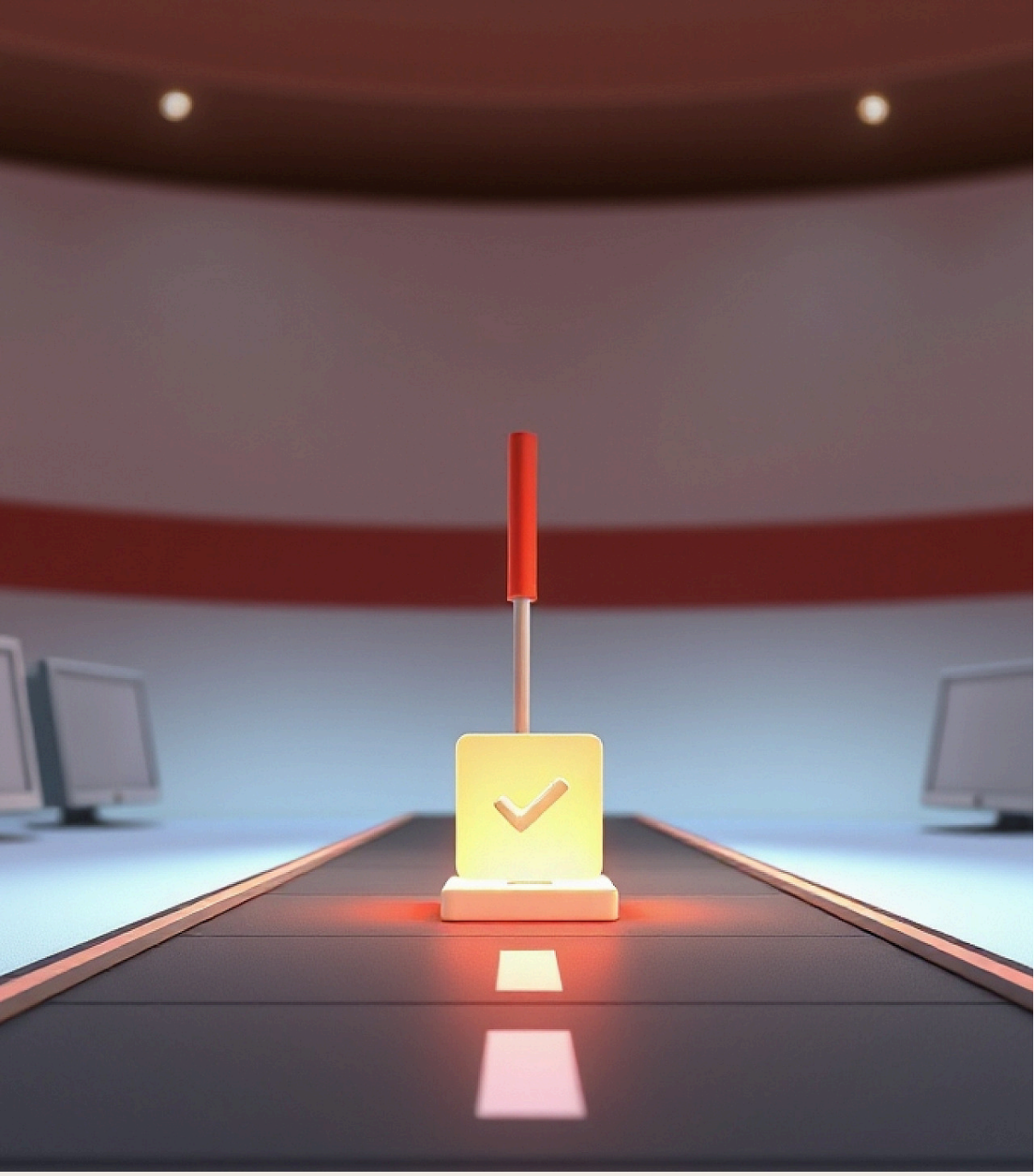
What is an infinite loop?



What is an infinite loop? It is a loop that keeps repeating and never reaches its end condition. So the instructions inside it run again and again, without stopping. The short answer is that an infinite loop never gets to the point where it should finish. That is the basic idea you will explore.



A loop is a set of instructions that runs again and again. Each time around, the same steps are repeated in order. For example, a loop might add one to a number and then check the number. It keeps doing this until something tells it to stop. Loops are common in programming to repeat tasks.



Every loop has an end condition, a check that tells it when to stop. When the check is met, the loop finishes and the program moves on. For example, a loop might stop when a number reaches 10. The end condition is like a finish line: once you cross it, you are done. Without it, the loop would not know when to quit.



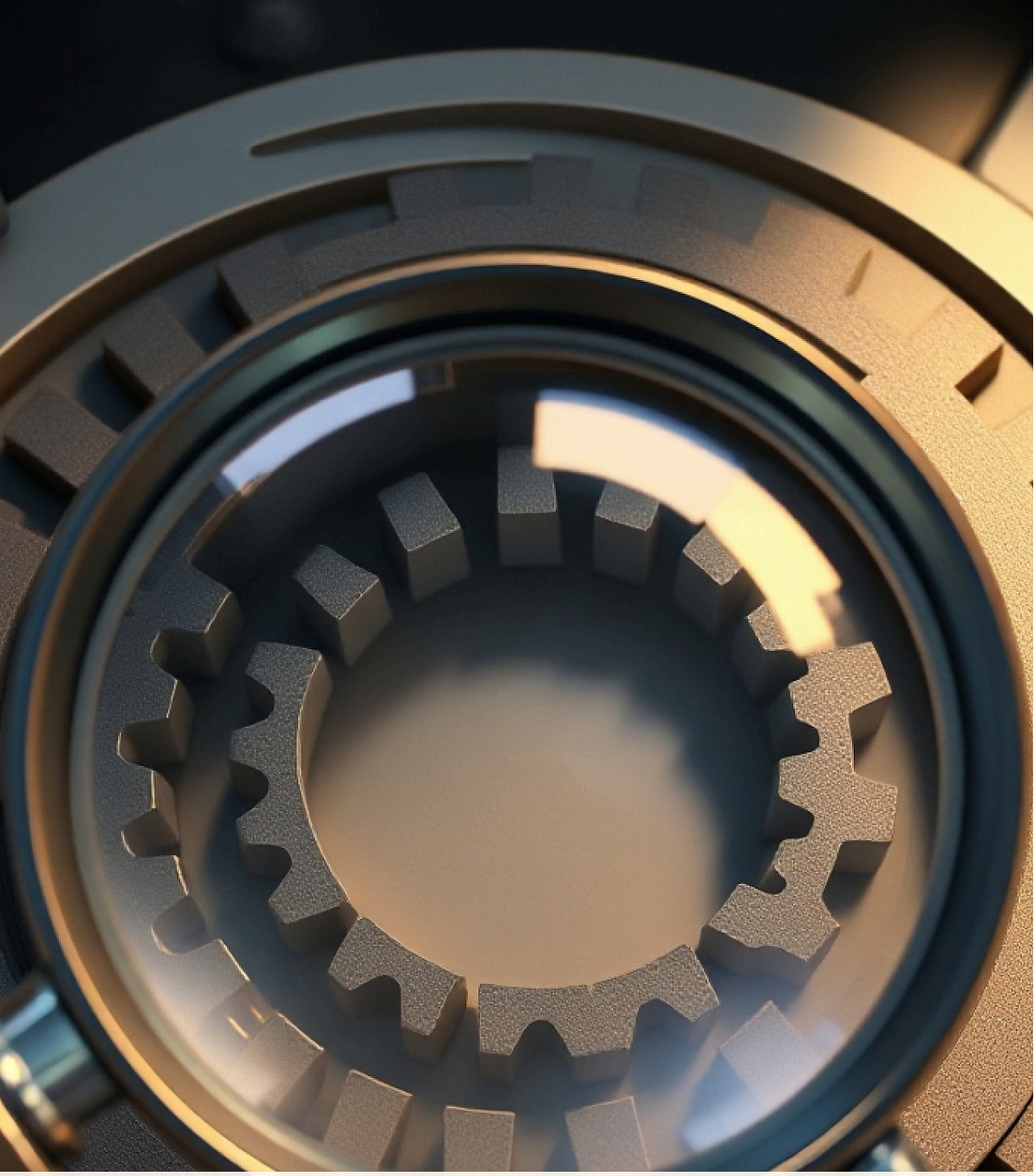
If the end condition never becomes true, the loop keeps going. This is the heart of an infinite loop. The check is done each time, but it always fails, so the loop never exits. For example, if the condition is number equals 10 but the number never changes, the check never passes. The loop runs forever.



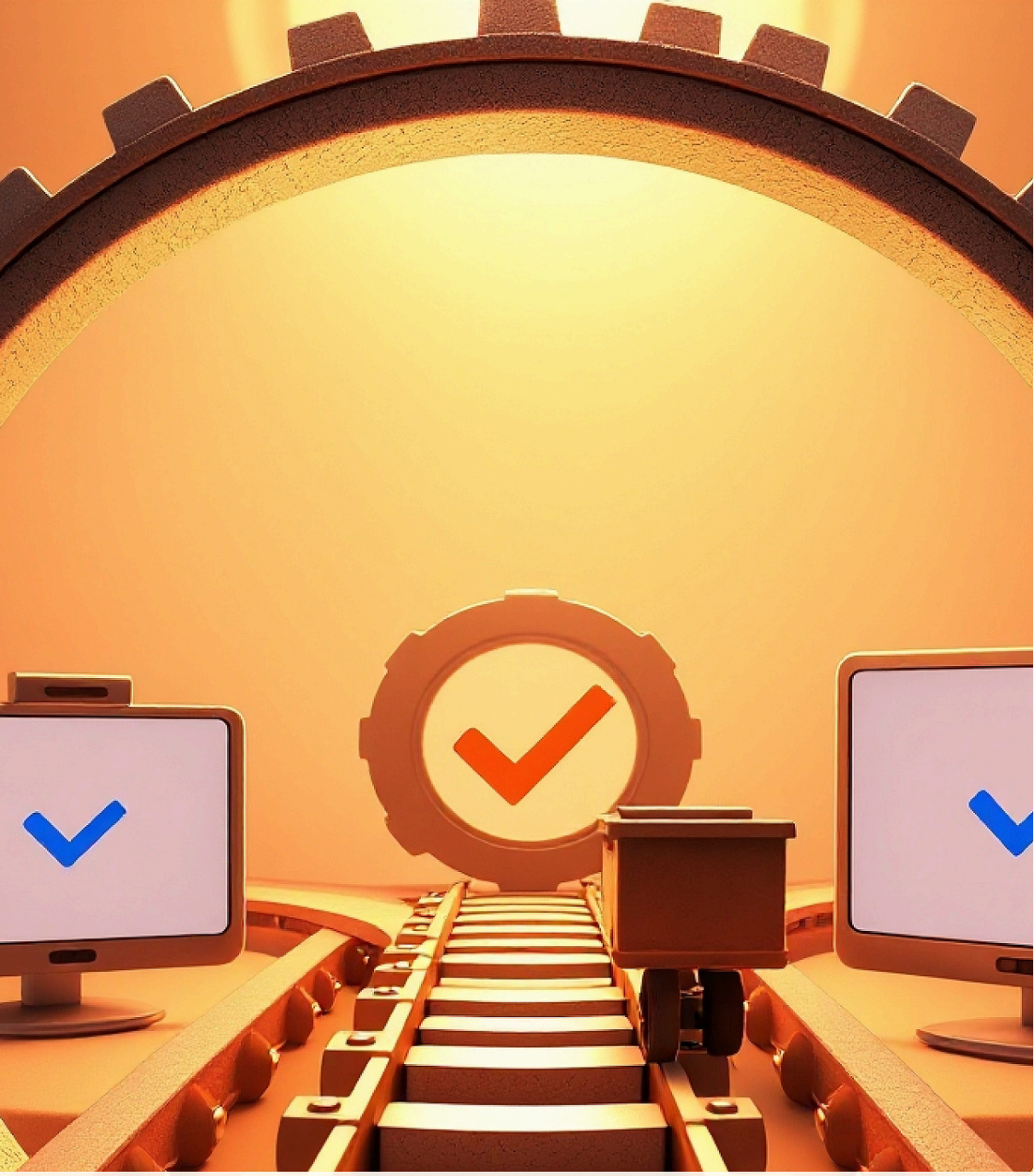
Consider a loop that checks whether a number is less than 10. Inside, it should add one to the number each time. But if the step that adds one is missing, the number stays the same. So the check number < 10 is always true, and the loop never ends. This is a simple mistake that causes an infinite loop.



The mistake happens because the loop forgot to update the value it checks. Without that change, nothing moves toward the end condition. The loop just repeats the same check with the same value. So the condition never becomes false. Updating the value is essential to make progress and eventually stop.



To spot the missing change, look at the loop and find the step that should change the number. In a counting loop, that step is usually an increment like adding one. If that step is missing or wrong, the number will not change. You can trace the loop by hand to see if the value ever updates. If it does not, you have found the problem.



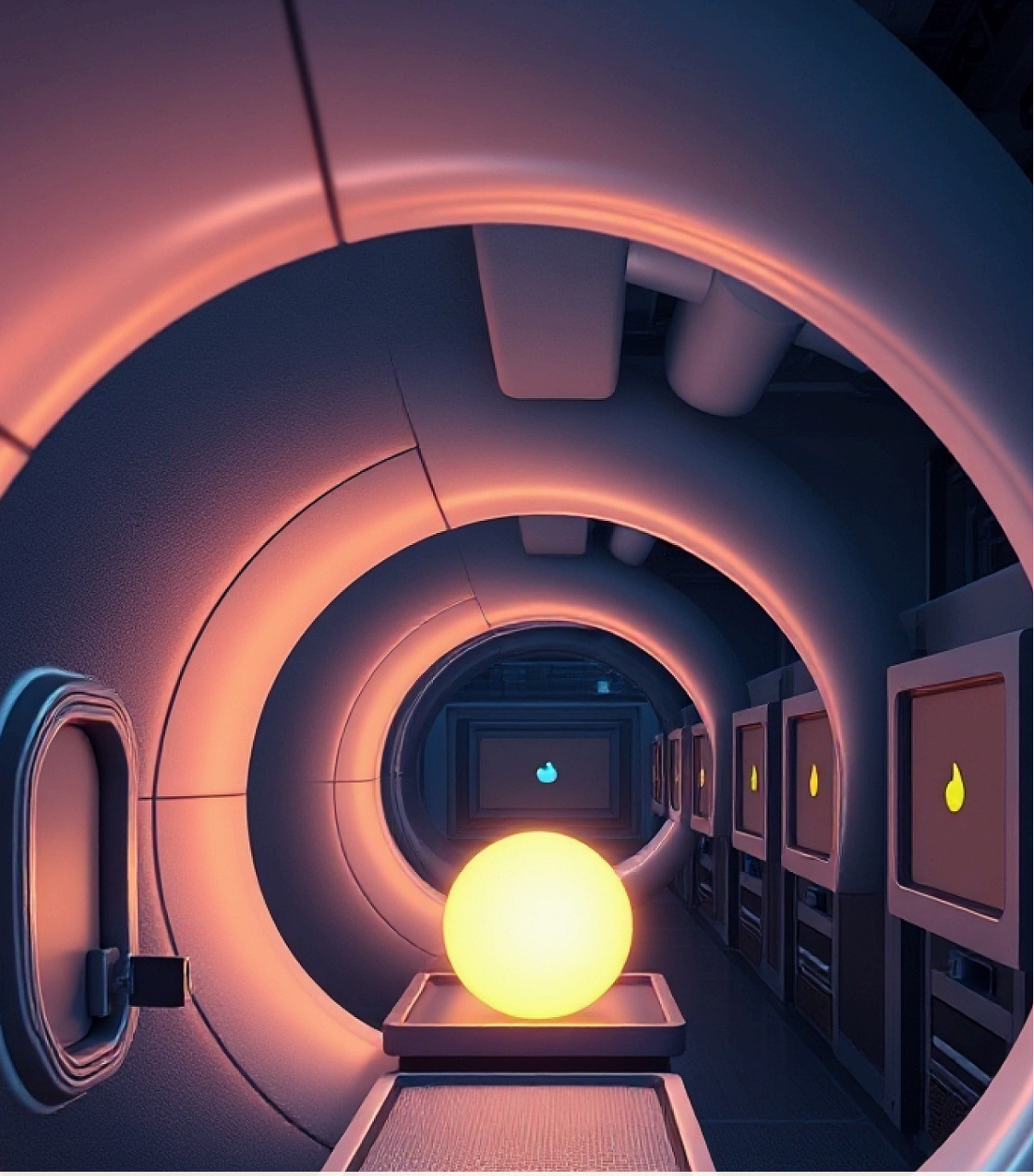
To fix the mistake, add or correct the step that changes the number each time. For example, include a line that adds one to the number inside the loop. Now the number increases, and eventually the check number < 10 becomes false. The loop can then end normally. Always ensure the loop makes progress toward its end condition.



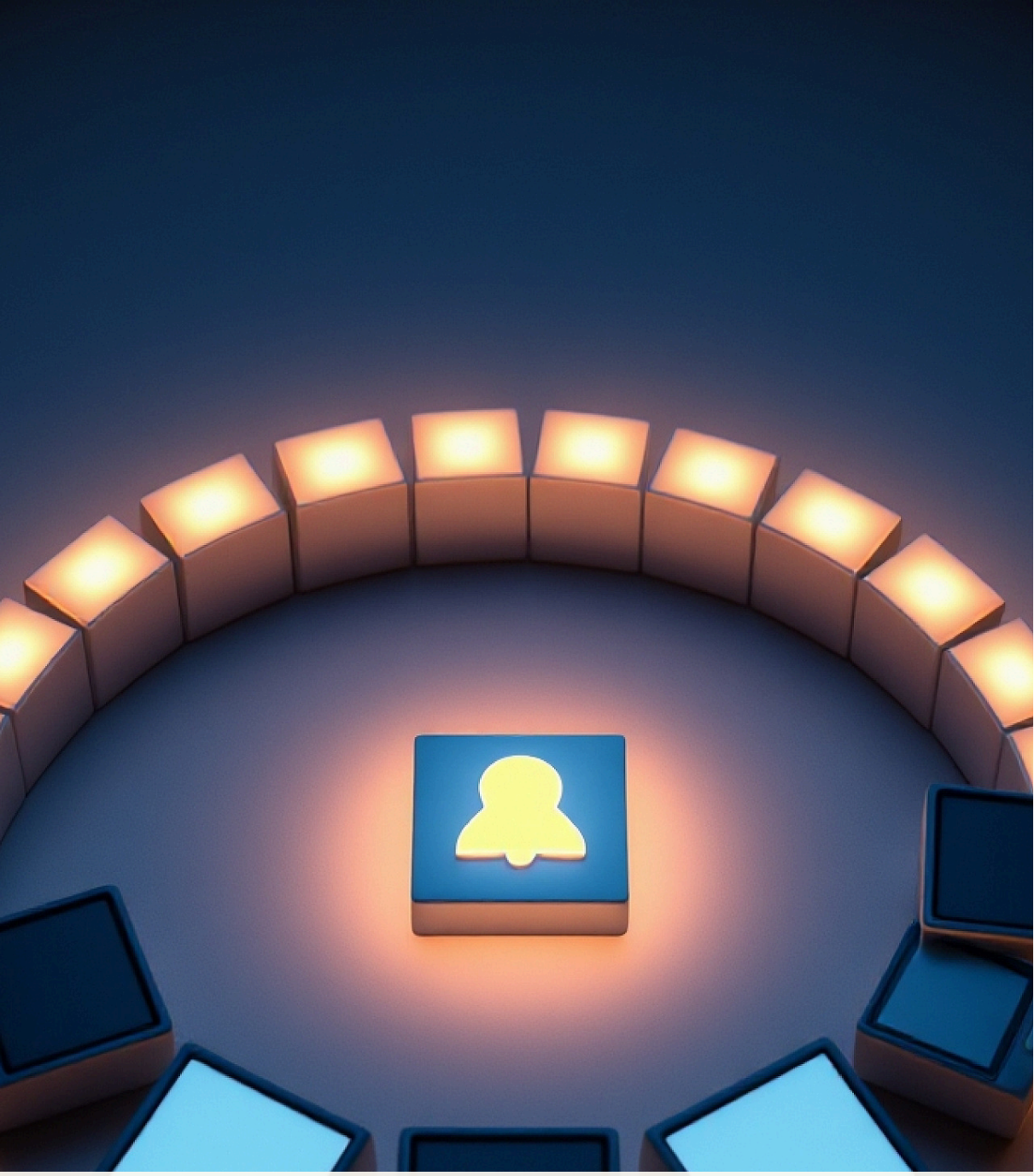
Sometimes people want a loop to keep running. It is not always a mistake. In some programs, an infinite loop is used on purpose to keep the program alive and responsive. For example, a game or a server needs to run continuously. So deliberate infinite loops are common and useful.



A game's main loop runs continuously to keep the game alive. It checks for player input, updates the game state, and draws the screen. This loop repeats many times per second. It does not have a fixed end condition because the game should run until the player quits. So the loop is infinite by design.



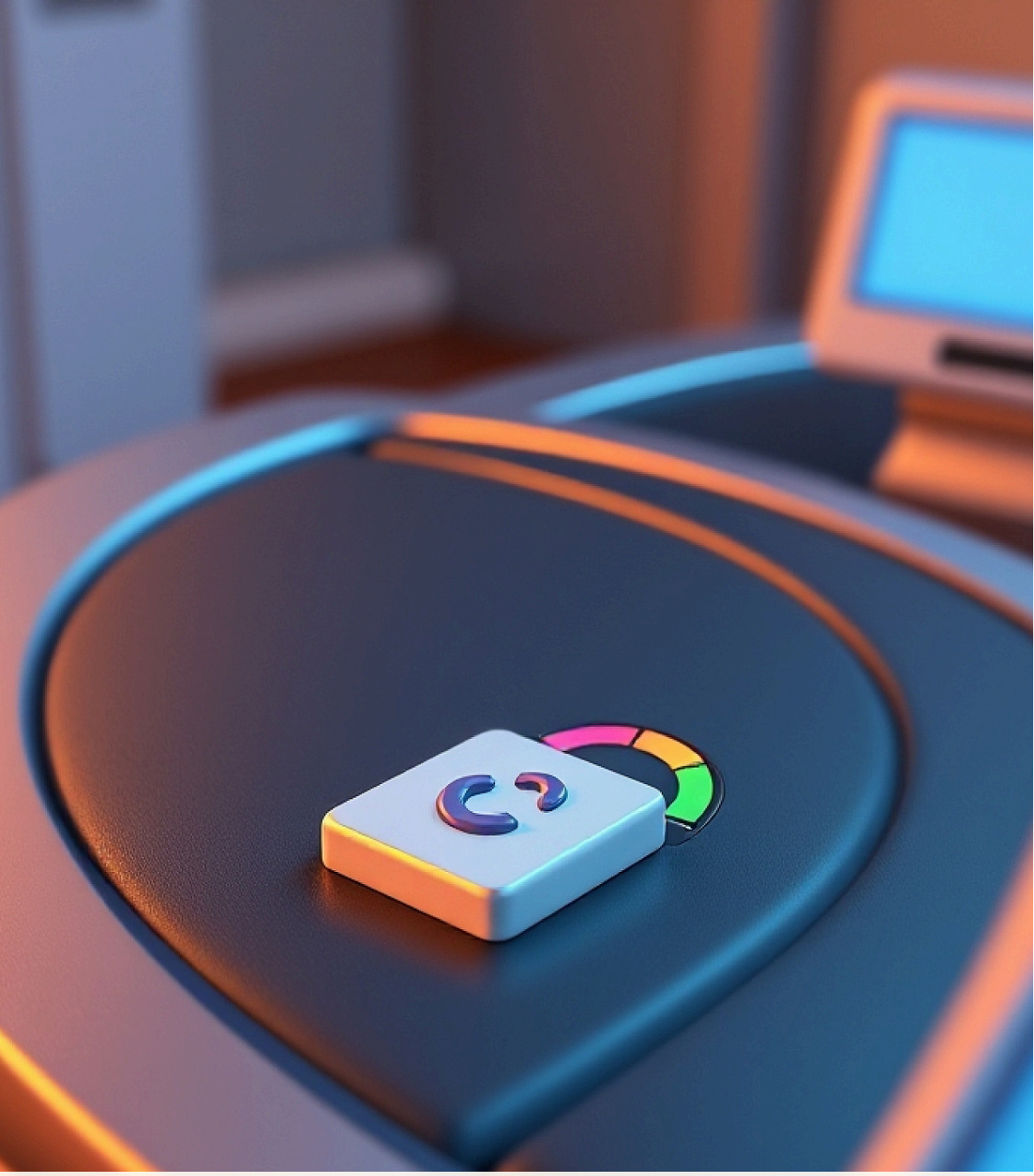
A server's main loop runs continuously to stay ready for requests. It waits for incoming connections, handles them, and then goes back to waiting. This loop repeats as long as the server is running. It has no planned end because the server should always be available. So it is another deliberate infinite loop.



These loops wait for events, like a key press or a request. They repeat so they are always ready when something happens. Instead of stopping after one event, they keep listening. This makes the program responsive and able to handle many events over time. Waiting for events is a key reason to use a deliberate infinite loop.



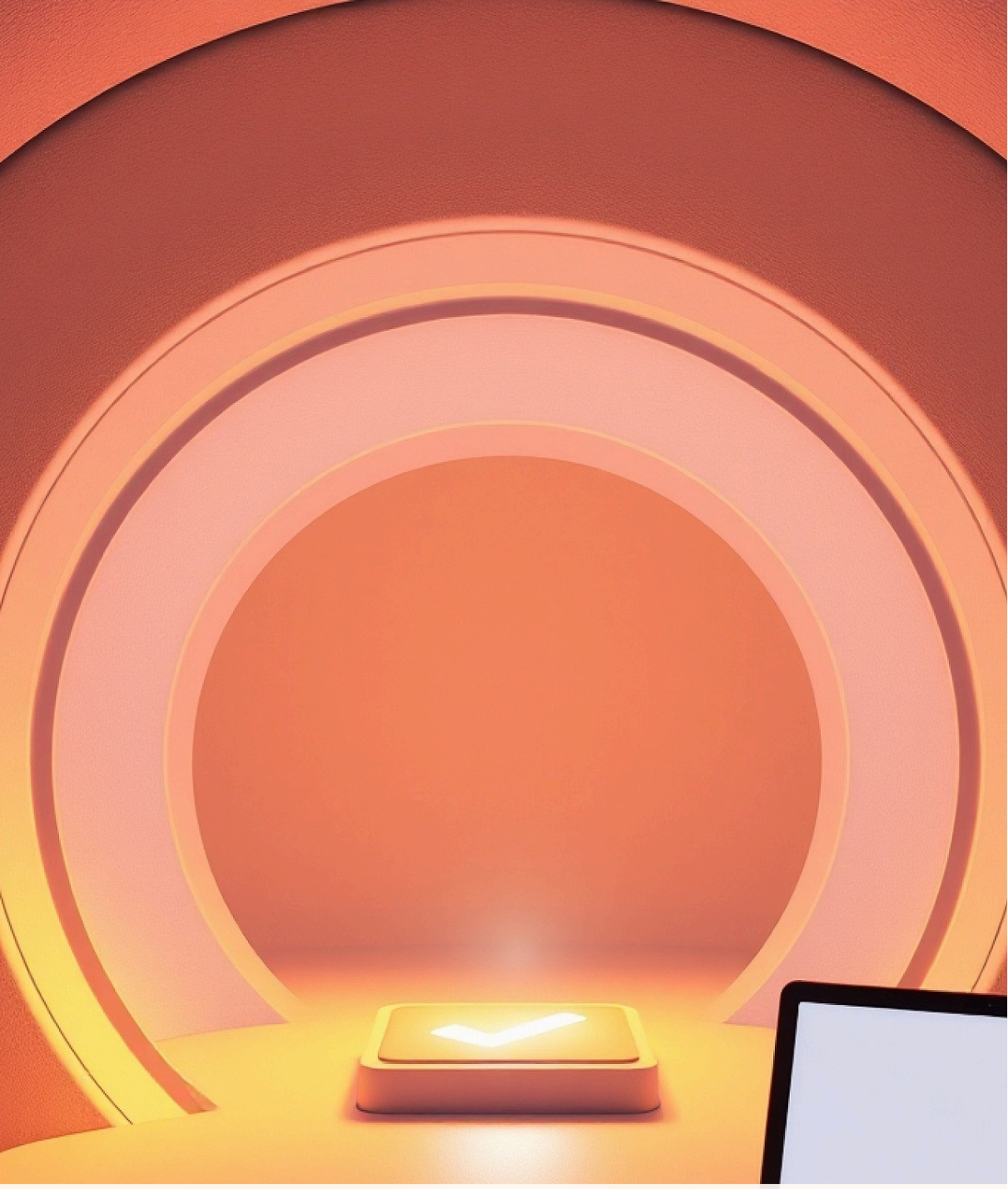
Even a deliberate loop must be able to stop when needed. There should be a way to end it, such as a quit command or a shutdown signal. For example, a game loop stops when the player chooses to exit. A server loop stops when the server is shut down. Without a way to stop, the program could run forever and be hard to control.



A running loop uses the computer's time and power. A good loop should not spin uselessly or waste resources. For example, a loop that constantly checks for input without pausing can use a lot of CPU. Better designs wait efficiently for events or include a small sleep. This keeps the program responsive without wasting energy.



Review the usual errors: forgetting to change the checked value, and letting a deliberate loop run with no way to stop it. Also, avoid loops that waste resources by spinning without pause. Check both the end condition and the update step. For deliberate loops, ensure there is a clear exit. These checks help you avoid problems.



An infinite loop never reaches its end condition. It can be a mistake, like forgetting to update a value, or on purpose, like a game or server loop. But even deliberate loops must stay stoppable and not waste resources. Remember to check for updates and provide a way to stop. That is the answer.