Bài 3: Review cấu trúc bộ nhớ và Đề xuất phương án cải thiện (30 phút)

* Xem đoạn code sau, chỉ ra ít nhất 2 vấn đề và mô tả phương pháp cải thiện:

tag\_log = []

def log(tag\_id, cnt, timestamp):

tag\_log.append((tag\_id, cnt, timestamp))

**✅ Vấn đề 1: Bộ nhớ sẽ tăng lên không ngừng (Memory Leak)**

**Nguyên nhân:**  
Danh sách tag\_log sẽ tiếp tục tăng kích thước mỗi khi hàm log() được gọi. Nếu ứng dụng chạy liên tục hoặc có nhiều lần gọi hàm log(), bộ nhớ sẽ bị chiếm dụng đáng kể và không bao giờ được giải phóng.

**Phương án cải thiện:**

* Sử dụng một cấu trúc dữ liệu như collections.deque với tham số maxlen để giới hạn kích thước của log. Khi số phần tử vượt quá giới hạn, phần tử cũ nhất sẽ bị loại bỏ tự động:

from collections import deque

tag\_log = deque(maxlen=1000) # Lưu tối đa 1000 bản ghi log

def log(tag\_id, cnt, timestamp):

tag\_log.append((tag\_id, cnt, timestamp))

**✅ Vấn đề 2: Không đảm bảo dữ liệu được lưu trữ theo thời gian thực (Data Integrity)**

**Nguyên nhân:**  
Nếu log() được gọi từ nhiều luồng (threads) hoặc tiến trình (processes), tag\_log có thể gặp xung đột dữ liệu do không có cơ chế bảo vệ (ví dụ: khóa dữ liệu).

**Phương án cải thiện:**

* Sử dụng threading.Lock để đảm bảo tính toàn vẹn của dữ liệu khi nhiều luồng cùng ghi vào tag\_log:

from collections import deque

from threading import Lock

tag\_log = deque(maxlen=1000)

log\_lock = Lock()

def log(tag\_id, cnt, timestamp):

with log\_lock:

tag\_log.append((tag\_id, cnt, timestamp))

Nếu ứng dụng là đa tiến trình (multiprocessing), thay Lock bằng multiprocessing.Manager() để chia sẻ dữ liệu giữa các tiến trình.