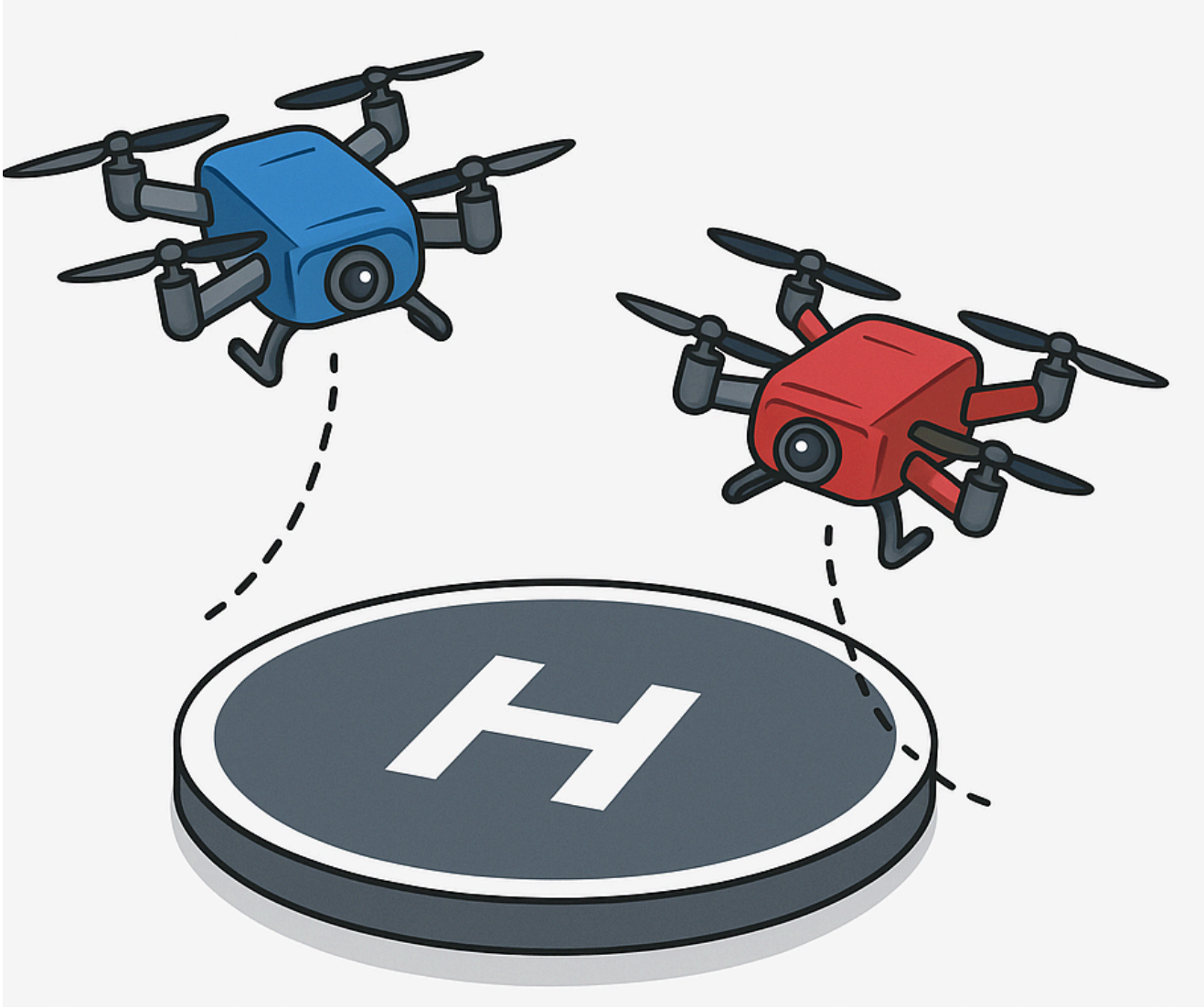
🧠 Schulaufgabe Parallele Programmierung (ChatGPT)

Thema: Synchronisation autonomer Drohnen



🟢 Aufgabe 1 (5 Punkte) – Grundlegender Fehler in der Implementierung

Im Projektordner finden Sie den Quelltext für das Simulationsprogramm der Drohnenstation: `drohnenstation.py`

Das Programm simuliert zwei autonome Drohnen, die im Kreis fliegen und an einer gemeinsamen **Landeplattform** Energie tanken können.
Beide Drohnen starten gleichzeitig, fliegen ihre Runden und versuchen gelegentlich, zu landen.

Wenn eine Drohne landet, wird die Plattform blockiert. Nach dem Start wird sie wieder freigegeben.

Beim letzten Update trat jedoch ein Fehler auf: **Beide Drohnen landen gleichzeitig auf der Plattform.**

Analysieren Sie den Code (Klassen `Drohenstation`, `Drohne`, `Plattform`) und **finden Sie die fehlerhafte Synchronisation.**
Beschreiben Sie in ein bis zwei Sätzen, **warum der Fehler auftritt**, und **verbessern Sie den Code**, sodass nie zwei Drohnen gleichzeitig landen.

> 💡 Hinweis: Die Plattform verhält sich wie eine Ampel in der alten Aufgabe. Eine Drohne sollte landen dürfen, wenn `plattform.frei == True`.

🟡 Aufgabe 2 (14 Punkte) – Synchronisation mit Peterson

Der bisherige Code funktioniert zwar, nutzt aber keine garantierte Synchronisation.
Sie sollen daher die Landeregelung mit dem **Peterson-Algorithmus** absichern.

Erstellen Sie dazu eine neue Klasse:

```
class DrohnenstationPeterson:
    ...
```

Implementieren Sie innerhalb dieser Klasse:

- zwei Threads (Drohne A und Drohne B),
- je eine **Flag**-Variable pro Drohne,
- und eine **Turn-Variable** nach Peterson.

Nur eine Drohne darf gleichzeitig auf der Plattform landen.
Nutzen Sie `time.sleep()` mit Zufallswerten, um das Verhalten zu simulieren.

🟠 Aufgabe 3 (9 Punkte) – Implementierung ohne aktives Warten

Wie Sie wissen, nutzen sowohl Dekker als auch Peterson **aktives Warten**.
Sie sollen nun eine Version ohne aktives Warten implementieren, die mit Python-Mechanismen (`Lock` oder `Condition`) arbeitet.

Kopieren Sie Ihre Lösung aus Aufgabe 2 und nennen Sie die Klasse:

```
class DrohnenstationSynchronisation:
    ...
```

Nutzen Sie:

- `threading.Lock` oder `threading.Condition`
- um sicherzustellen, dass **nur eine Drohne gleichzeitig landet**,
- und dass die andere Drohne erst dann landen darf, **wenn die Plattform wieder frei ist.**

🔵 Aufgabe 4 (21 Punkte) – Plattform für mehrere Drohnen

Ihr Chef ist begeistert von der stabilen Simulation und möchte das System erweitern: Mehrere Drohnen (beliebige Anzahl) sollen gleichzeitig fliegen und **nacheinander** auf der Plattform landen dürfen.

Erstellen Sie dazu eine Klasse:

```
class DrohnenstationQueue:
    ...
```

Die Klasse soll:

- mehrere Threads (Drohnen) starten,
- eine gemeinsame `Plattform` verwalten,
- eine **Warteschlange (Queue)** verwenden, um die Reihenfolge der Landungen zu regeln.

Nutzen Sie für die Synchronisation eine `Condition`, die sicherstellt,
dass immer **nur die erste Drohne der Warteschlange** landen darf.

> Bonus (2 Punkte): Zeigen Sie beim Ablauf der Simulation im Terminal an, **welche Drohne gerade landet** und **welche warten muss.**

✍️ Hinweise

- Verwenden Sie nur die Standardbibliothek (`threading` , `time` , `random` , `queue`).
- Nutzen Sie keine globalen Variablen.
- Kommentieren Sie Ihren Code sinnvoll.
- Achten Sie auf reproduzierbares Verhalten (z. B. `random.seed(0)` für Tests).

🧩 Datei: `drohnenstation.py` *(Startcode für Aufgabe 1)*

```
"""
drohnenstation.py
Fehlerhafte Ausgangsdatei zur Schulaufgabe Parallele Programmierung
Simulation: Zwei Drohnen teilen sich eine Landeplattform
```

```
"""

import threading
import time
import random


class Plattform:
    def __init__(self):
        self.frei = True  # Plattform ist anfangs frei


class Drohne(threading.Thread):
    def __init__(self, name: str, plattform: Plattform):
        super().__init__()
        self.name = name
        self.plattform = plattform

    def run(self):
        while True:
            print(f"{self.name} fliegt ihre Runde im Luftraum …")
            time.sleep(random.uniform(0.2, 0.6))

            print(f"{self.name} möchte landen …")
            # ❌ FEHLER: keine Synchronisation – beide Drohnen können
gleichzeitig landen!
            if self.plattform.frei:
                print(f"{self.name} landet auf der Plattform.")
                self.plattform.frei = False
                time.sleep(random.uniform(0.3, 0.6))
                print(f"{self.name} startet wieder.")
                self.plattform.frei = True
            else:
                print(f"{self.name} muss warten – Plattform ist belegt.")

            time.sleep(random.uniform(0.2, 0.5))


if __name__ == "__main__":
    plattform = Plattform()
    d1 = Drohne("Drohne A", plattform)
    d2 = Drohne("Drohne B", plattform)

    d1.start()
    d2.start()
```