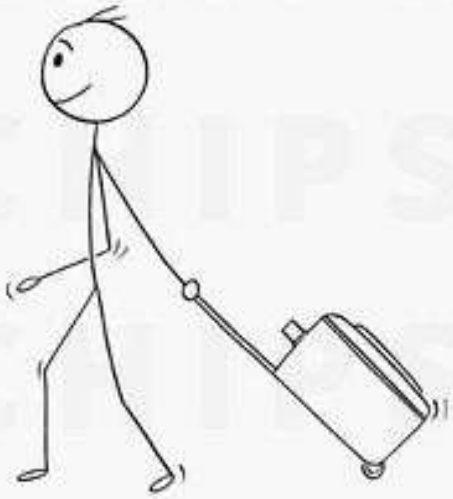
DESIGN PATTERN

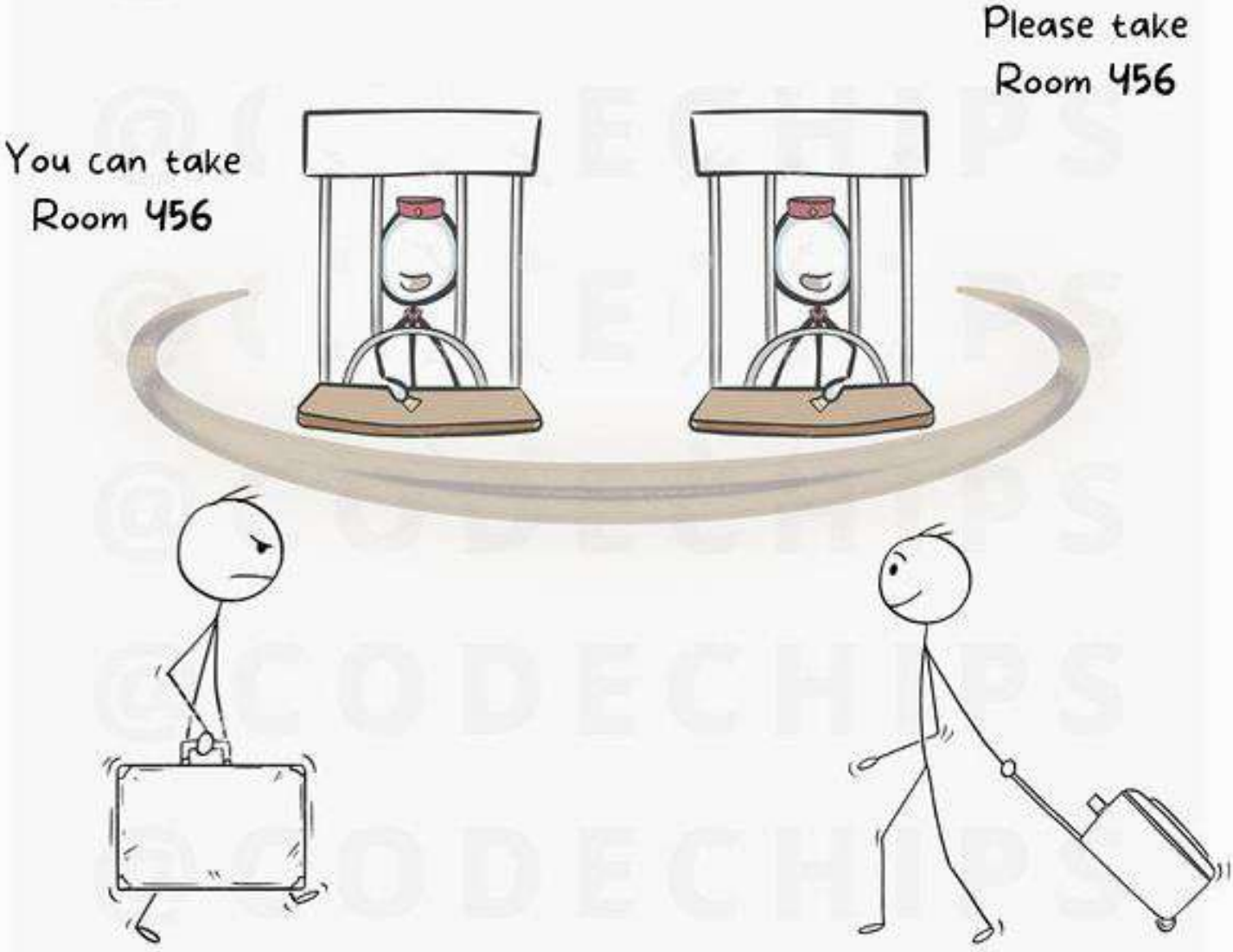
SINGLETON

@codechips
Cody Dev
popupdev04@gmail.com

Let's say you want to **book a hotel room**



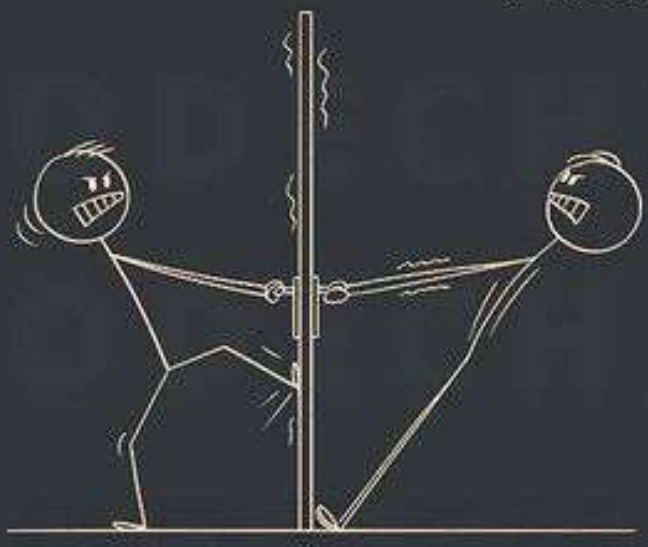
There are **multiple counters** and you book a room from one of them



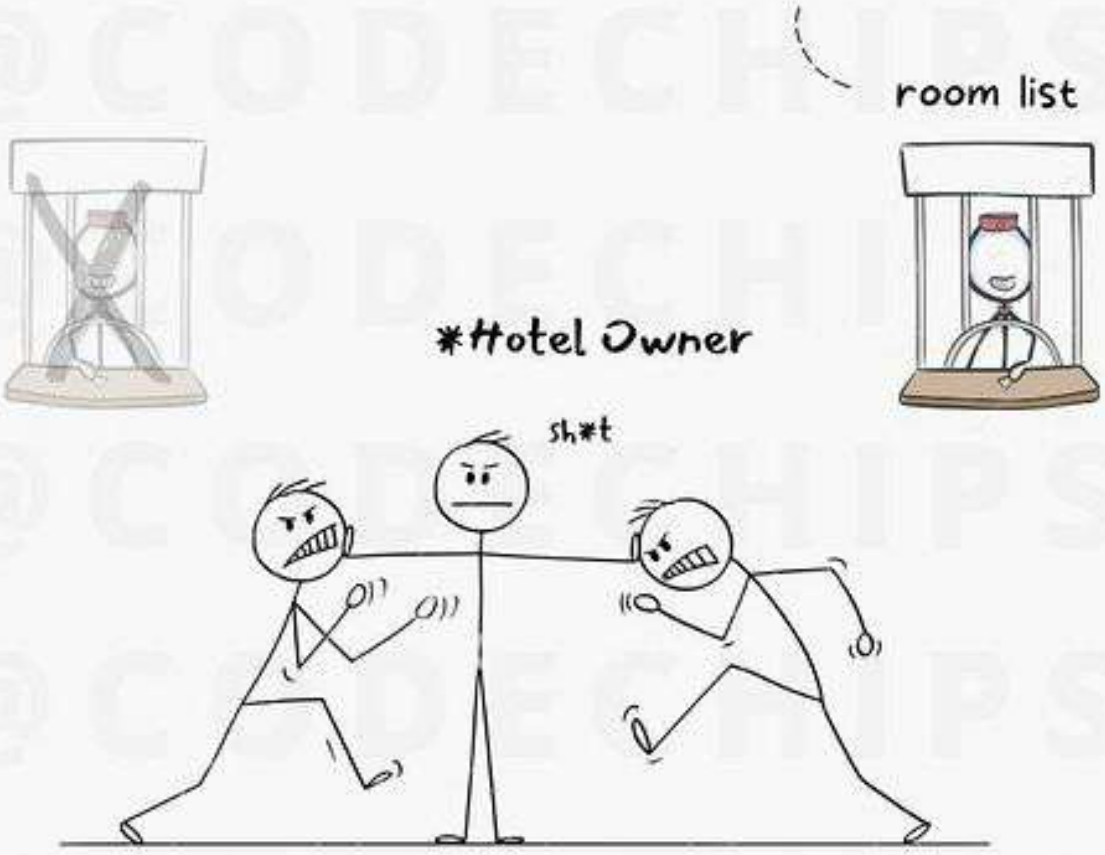
But the same room got booked to another person too

Oh what else do you think I did!@#

I booked this room



This wouldn't have happened if there is only a **single counter** as a global point to **control access** to the shared resource



In software world, you may need to

- Ensure that a class has just a single instance
- Provide a global access point to that instance

eg: a database or a file

How to implement Singleton Pattern?

1.Make the default constructor private, to prevent other objects from using the new operator with the Singleton class.

```
private constructor Database() is
    // Some initialization code, such as the actual
    // connection to a database server.
    // ...
```

2. Create a static creation method that acts as a constructor

```
// The static method that controls access to the singleton
// instance.
public static method getInstance() is
    if (Database.instance == null) then
        acquireThreadLock() and then
            // Ensure that the instance hasn't yet been
            // initialized by another thread while this one
            // has been waiting for the lock's release.
            if (Database.instance == null) then
                Database.instance = new Database()
    return Database.instance
```

When to use this Design Pattern?

Use the Singleton pattern when a class in your program should have just a **single instance** available to all clients

The Singleton pattern disables all other means of creating objects of a class except for the special creation method