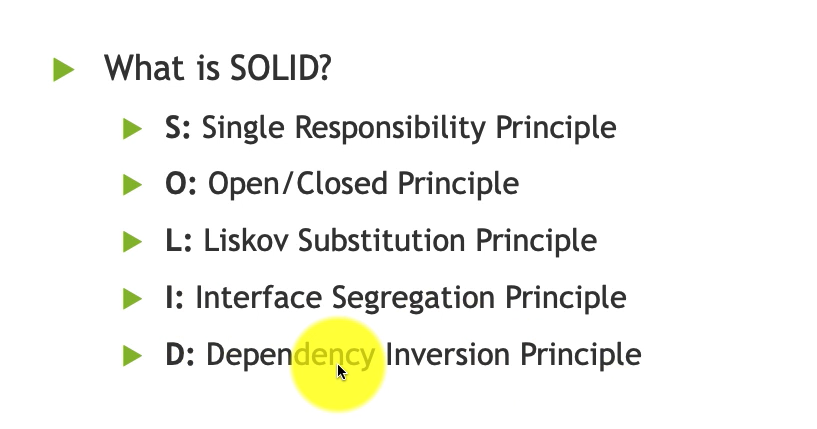
**SOLID**

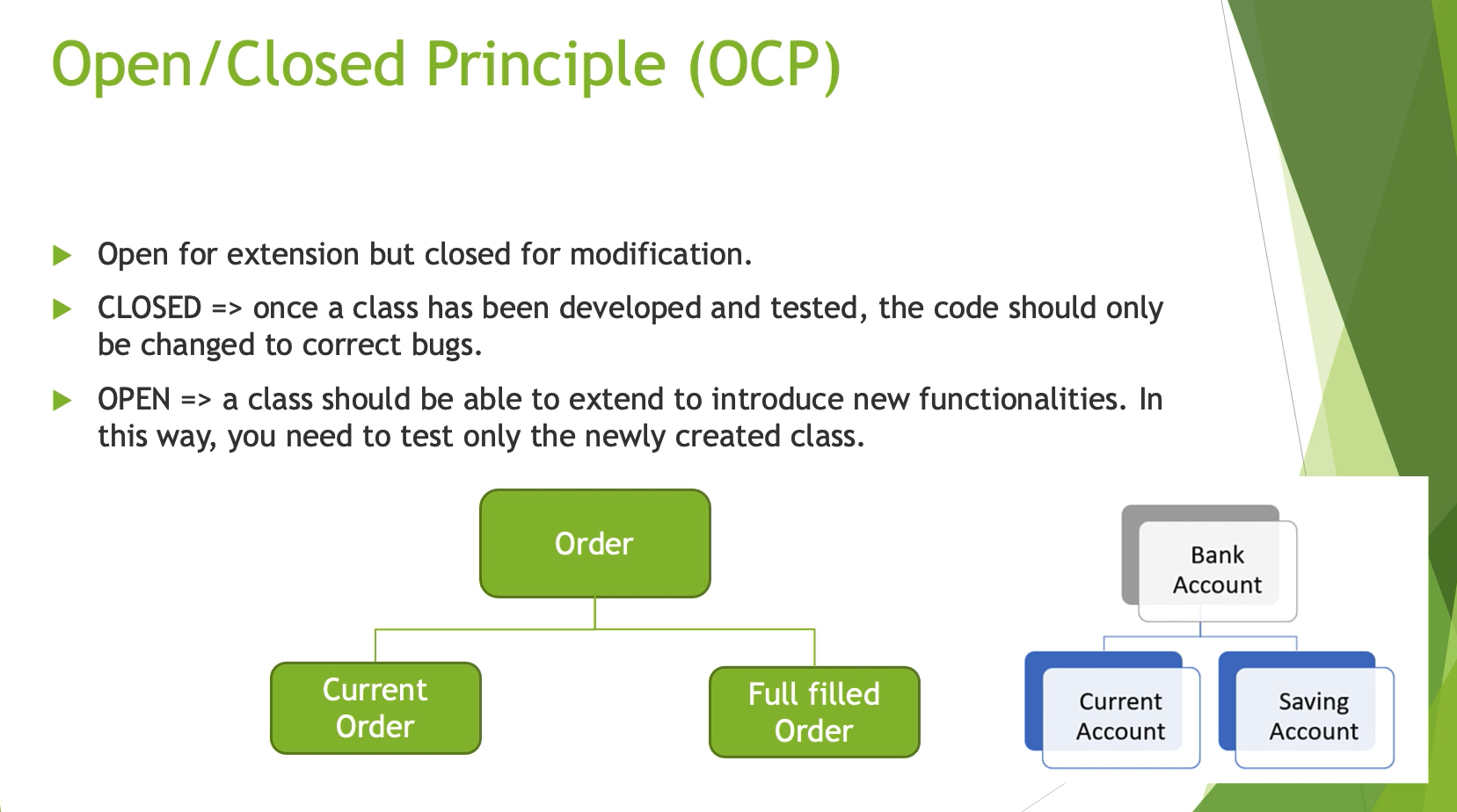


1. Single Responsibility Principle

* 1 Class chỉ nên có 1 và chỉ 1 trách nhiệm

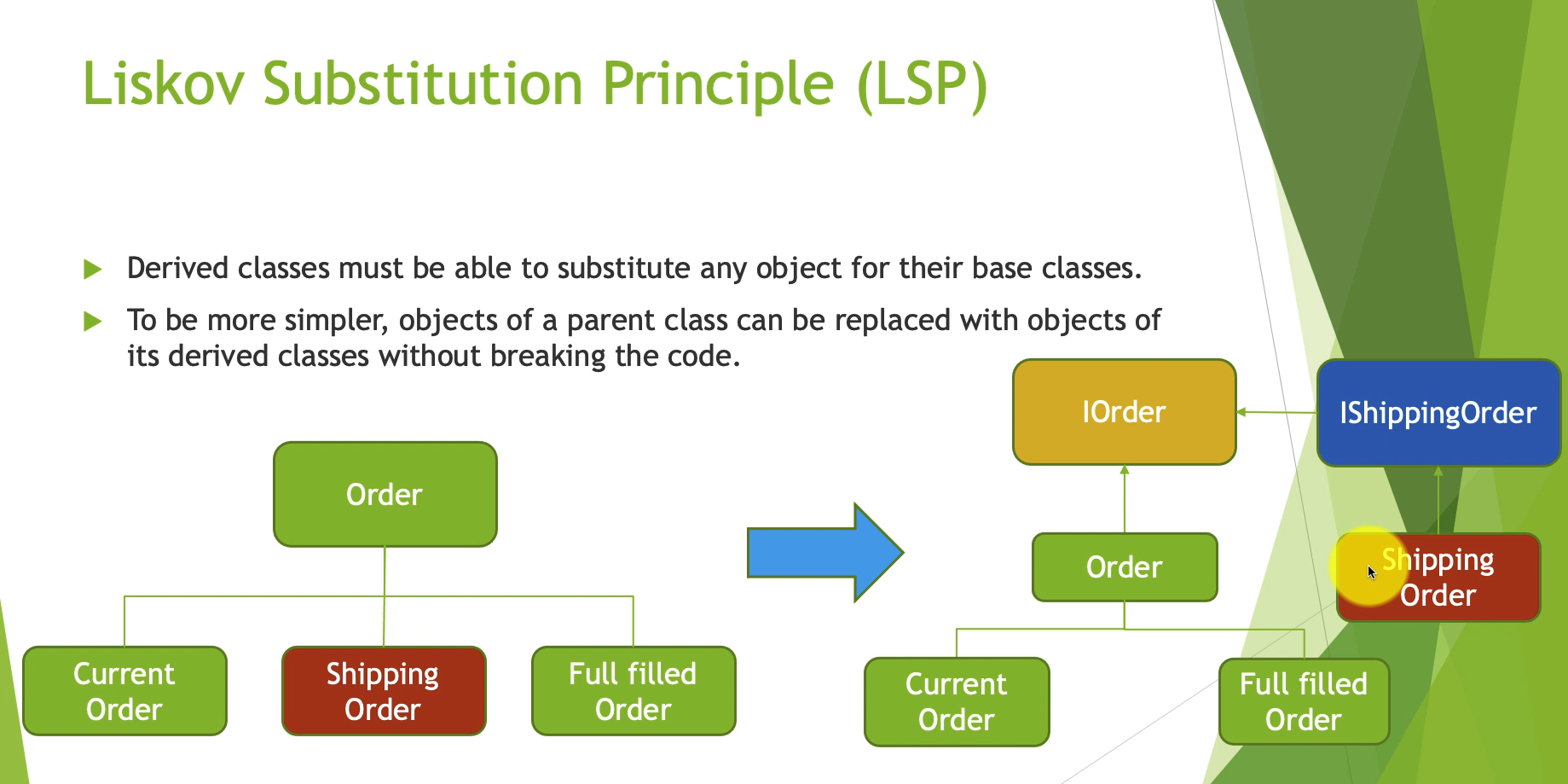
1. Open/Closed Principle

* 1 Class sẵn sàng cho mở rộng thêm 1 tính năng, nhưng nếu sửa thì nó sẽ đóng lại
* CLOSED: Khi 1 class đã được developed và tested thì code chỉ nên được thay đổi để sửa lỗi, chứ không được sửa hay thêm tính năng
* OPEN: 1 Class có thể được phép thêm tính năng mới. Nên chỉ cần test các tính năng, function mới thôi, không cần đụng tới những gì đã làm rồi



1. Liskov Substitution Principle

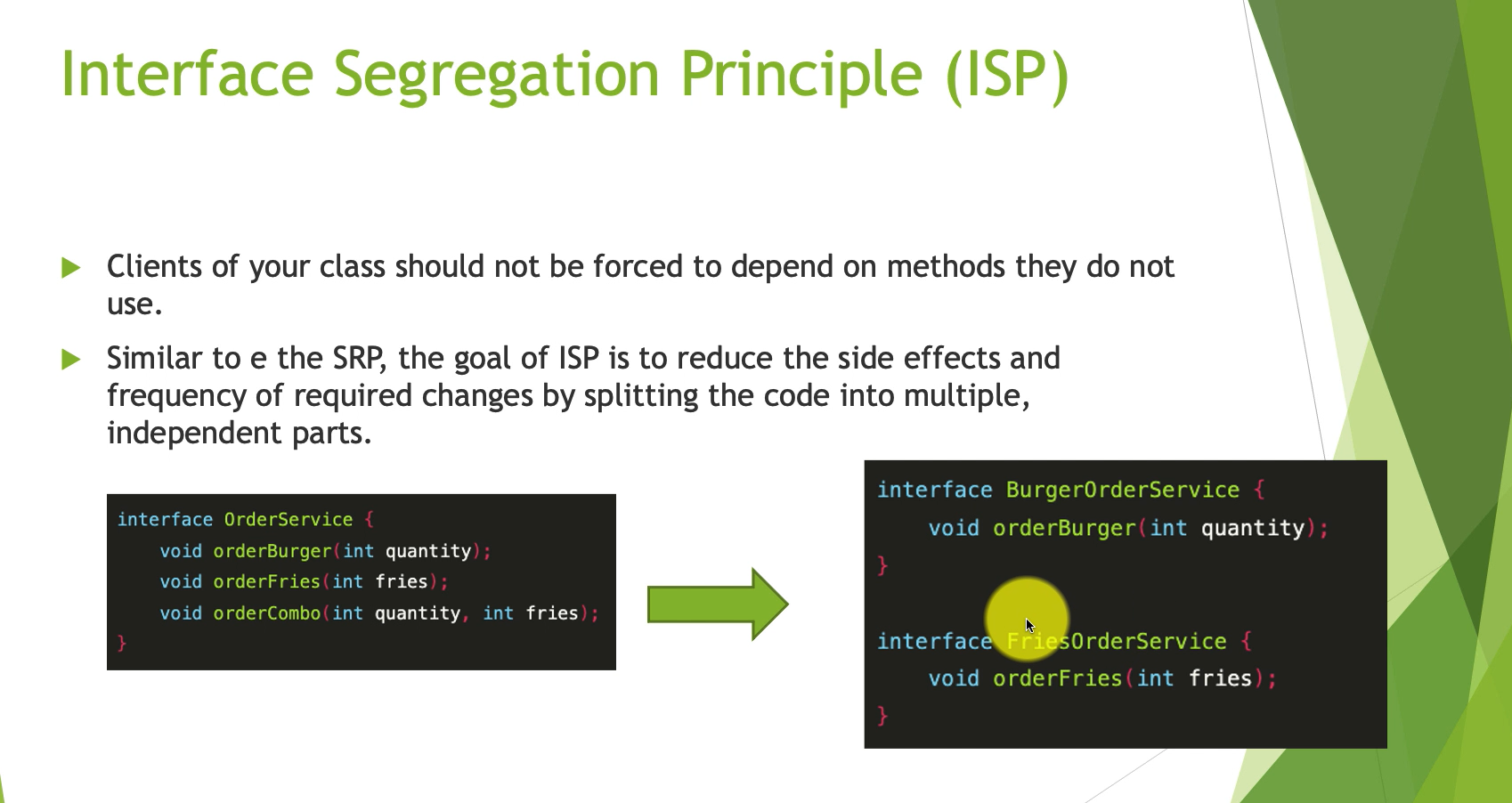
* Các class con có thể thay thế bất kì đối tượng nào của class cha
* Đối tượng của class cha có thể bị thay thế bởi đối tượng của các class (lớp) dẫn xuất mà không làm ảnh hưởng gì tới thiết kế thổng thể, tới code



* Ex:Shipping Order có thể thay thế (triển khai, viết lại) luôn các phương thức mà class Order đưa ra mà không làm ảnh hưởng gì đến thiết kế tổng thể
* Ex: IOrder có hàm CalculateTotalAmount() Cả class Current Order và class Shipping Order đều triển khai hàm này. Nhưng class Shipping Order có cộng thêm phí ship thì có thể triển khai dễ dàng mà không ảnh hưởng tới các class con của IOrder hay class con khác

1. Interface Segregation Principle

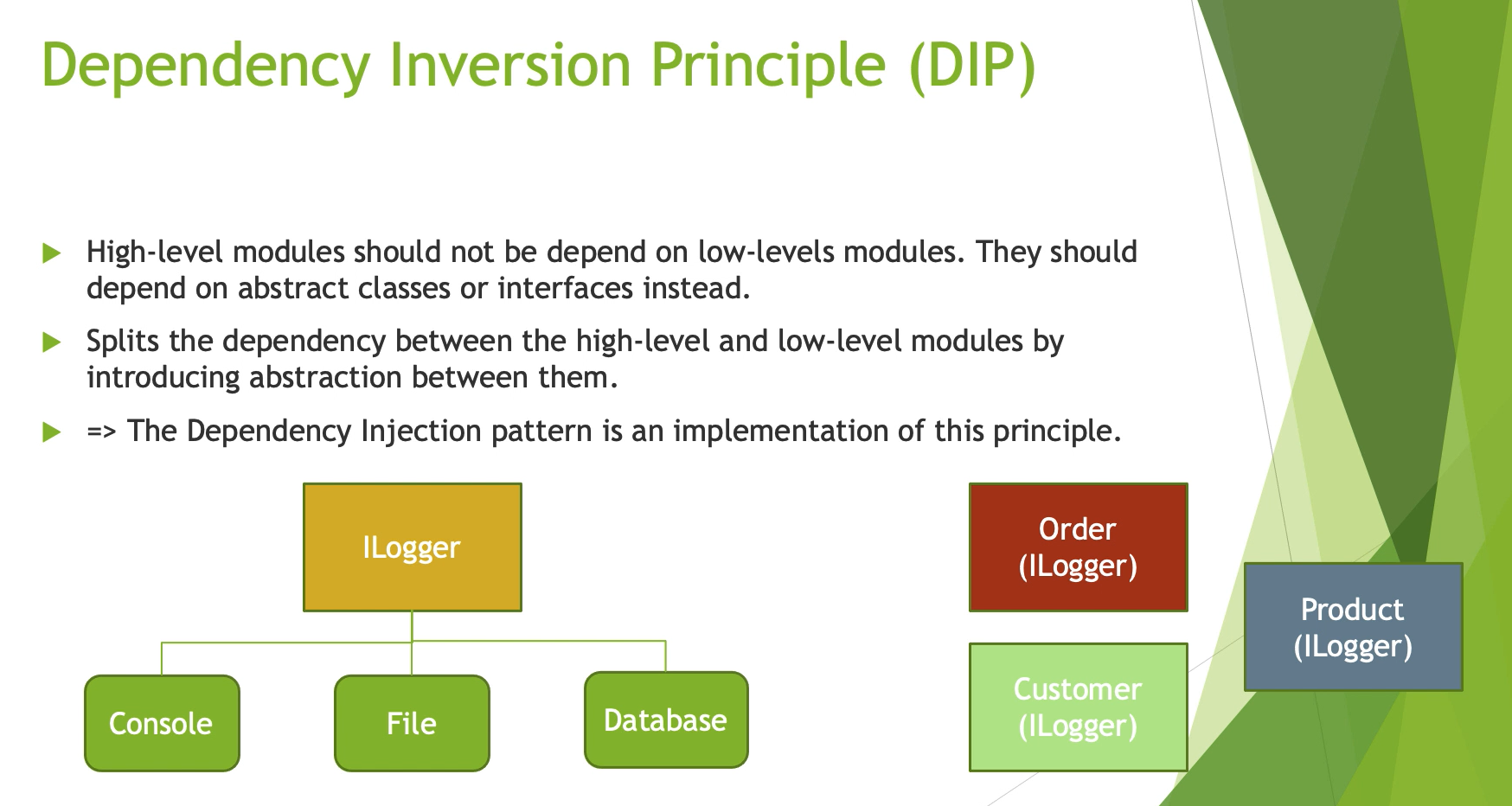
* Nguyên lý tách biệt các interface
* Các client của các class không nên bị ép buộc phải phụ thuộc vào các method mà họ không sử dụng
* Ex: Class implement 1 interface nhưng không nên triển khai hết tất cả các hàm mà interface đó đưa ra nếu không dùng tới hay không cần thiết



* Ex: 1 class kế thừa interface OrderService nhưng chỉ cần triển khai và sử dụng phương thức orderBurger nhưng lại phải triển khi vả 2 phương thức còn lại => Không nên => Nên tách ra thành các interface riêng biệt cho từng phương thức

1. Dependency Inversion Principle

* Các module cấp cao không nên phụ thuộc vào các module cấp thấp. Mà nên phụ thuộc vào các class abstract hoặc là interface (depend on abstract classes or interfaces instead)
* Các class chính nên inject các class abstract hoặc các interface thay vì new 1 đối tượng trực tiếp



* Ex: Khi ở controller muốn sử dụng repository thì inject các interface IRepository thay vì new Repository()
* Ex: Trên hình, muốn tạo ra 1 logger 1 console, 1 log ra file, 1 log vào DB… Ta nên tạo ra 3 đối tượng và triển khai Ilogger.. ở nơi cần sử dụng thì chỉ cần inject các đối tượng mà ta vừa triển khai…. Ví dụ: cần log ra file thì inject đối tượng FileLogger đã được triển khai các thuộc tính của ILogger (ILogger chỉ đưa ra các phương thức LogInfor, LogError, LogWarning và các lớp con cần triển khai các phuong thức này)

**Command Query Responsibility Segregation (CQRS)**

* Command: Write data (Các câu lệnh create, update, delete) nên được tách biệt với phần query
* Query: Read data (READ) chỉ read data

