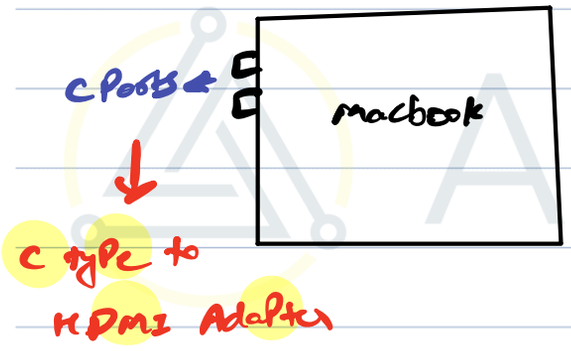
Today's agenda

- Adapter design Pattern
- Facade design Pattern

* Structural design Pattern
  - How to structure our codebase.
    - classes / interface decisions.
    - attributes within a class.

① Adapter design Pattern
- Power Adapter → Convert one form to another

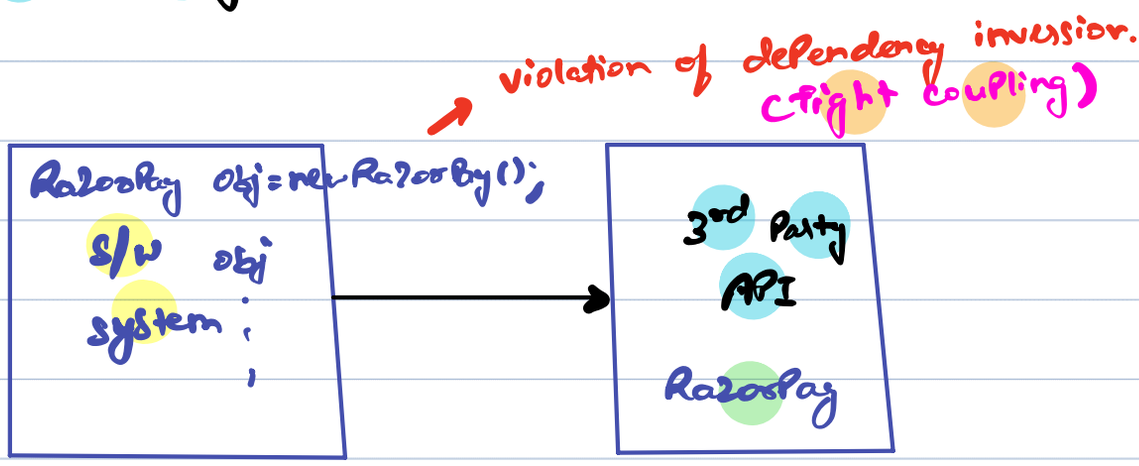


→ 1 C type
⊥ usb type
⊥ HDMI type Slots?

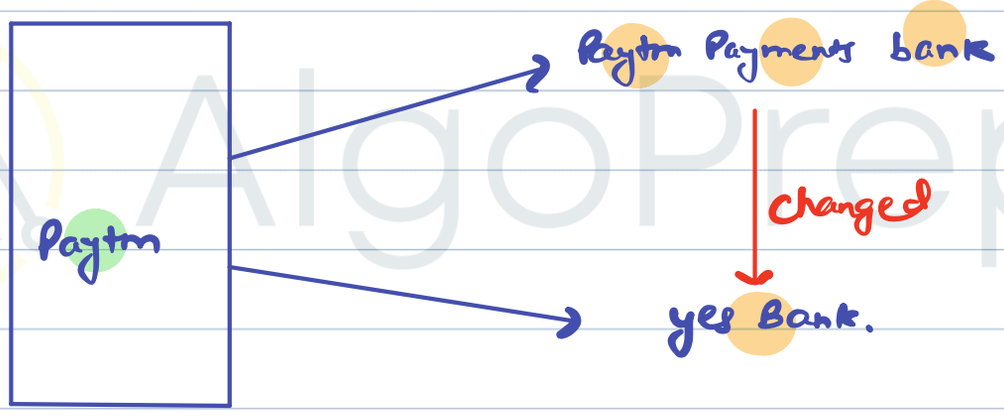
↓

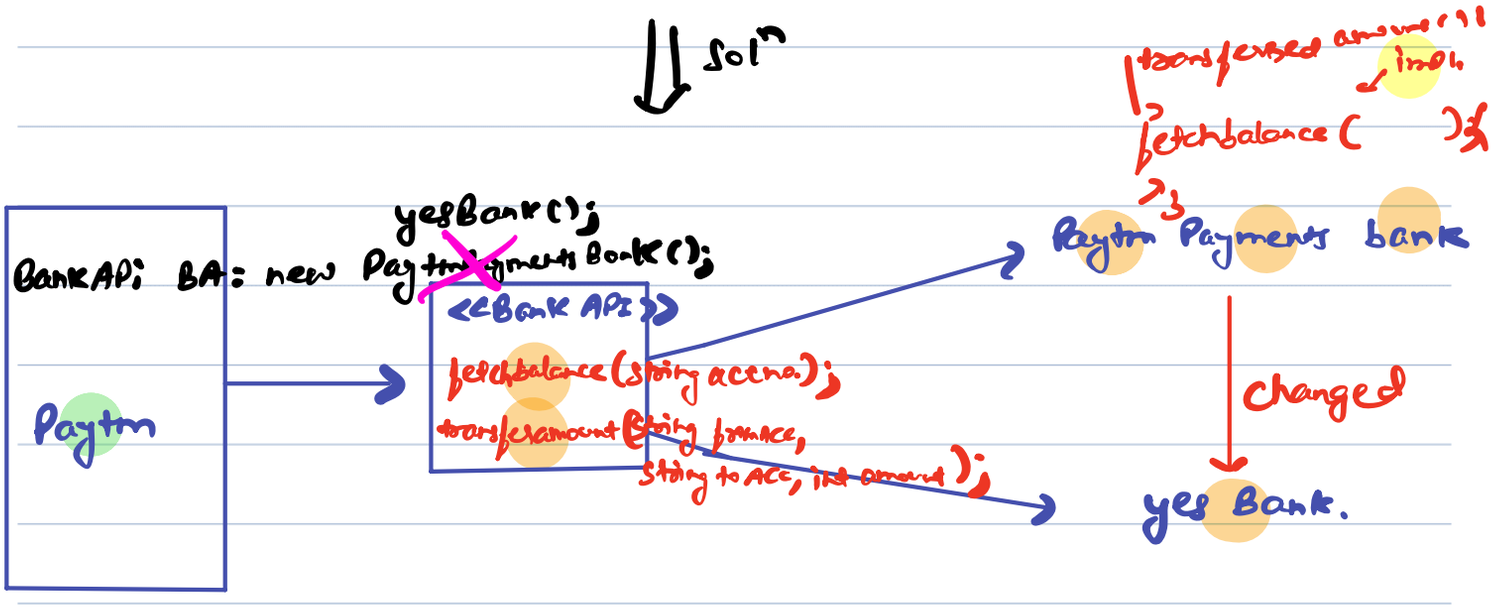
Having only C type makes the design relatively easy.

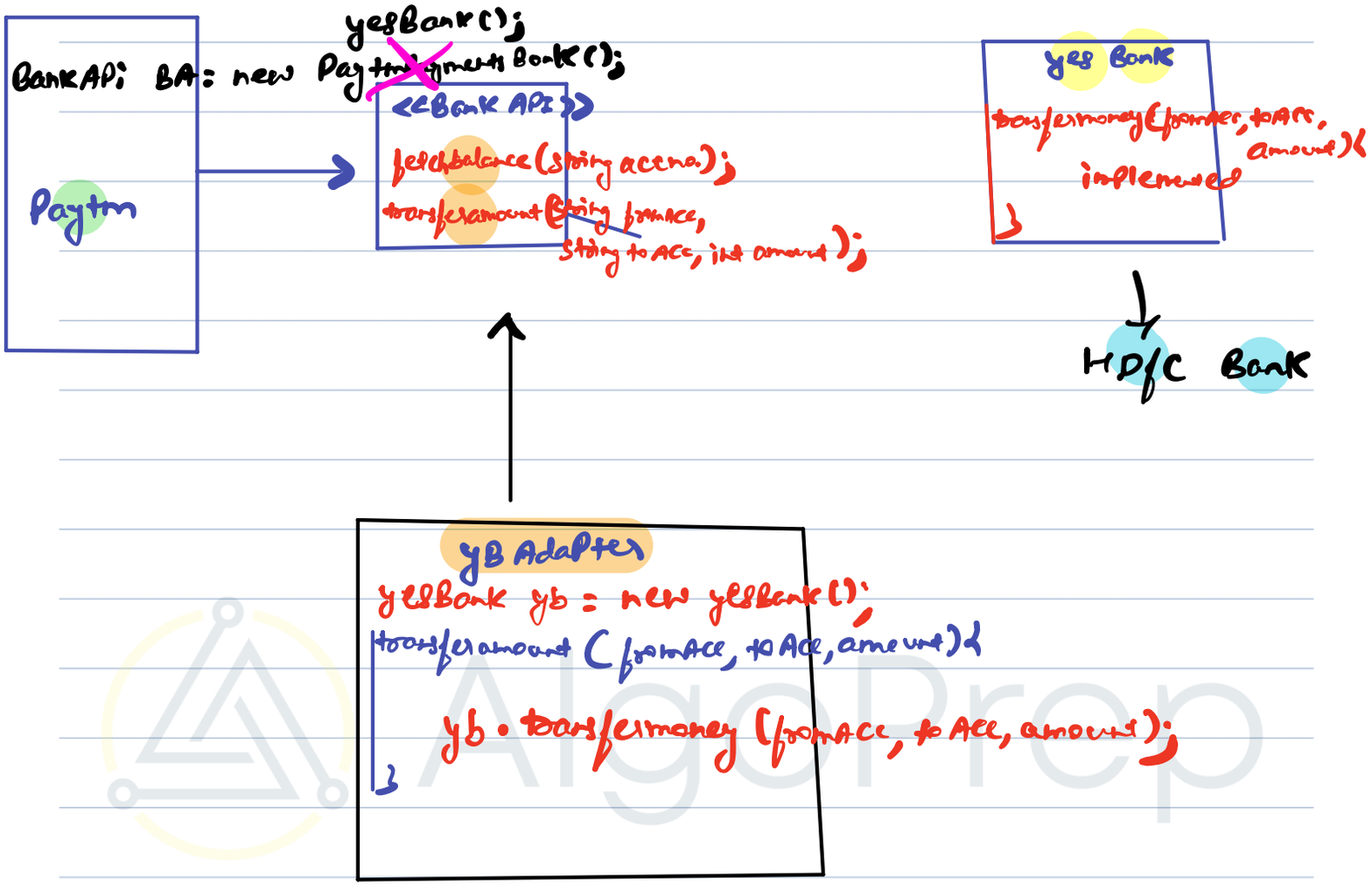
Adapter design Pattern



→ Paytm Problem Statement.






BankAPI BA = new PaytmBank();
yesBank();
Paytm
<<Bank API>>
fetchBalance(String accNo);
transferAmount(String fromAcc, String toAcc, int amount);
yes Bank
transferMoney(fromAcc, toAcc, amount){
implemented
HDFC Bank
YB Adapter
yesBank yb = new yesBank();
transferAmount(fromAcc, toAcc, amount){
yb.transferMoney(fromAcc, toAcc, amount);

→ How to we Adapters

↳ 1. Whenever we are connecting to a 3rd Party APIs, Create an interface with all the required methods.

2. Create an adapter class which is going to act as a bridge between yours and 3rd Party service Providers.

→ Calendly:

↳ 3:00 Pm 4:00 Pm

↳ 3:15 3:30 3:45 4:00

↳ name

↳ email

3rd Party APIs.

↳ google meet / zoom

↳ Calendar

:

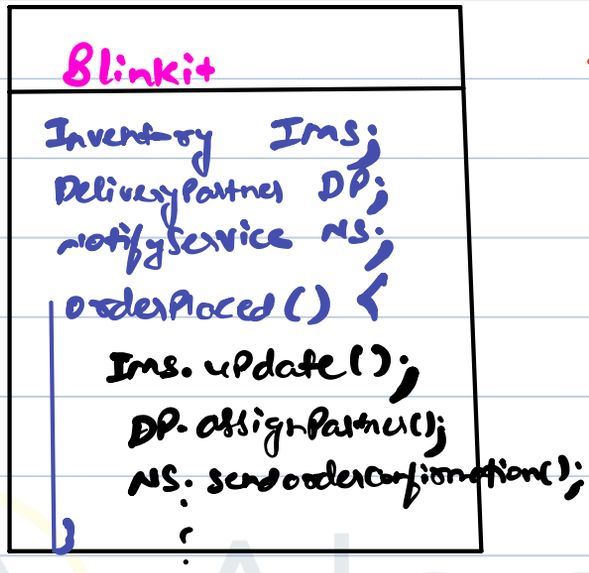
:

→ Cal.com → A competitor of Calendly with open sourced codebase.

* Facade design Pattern

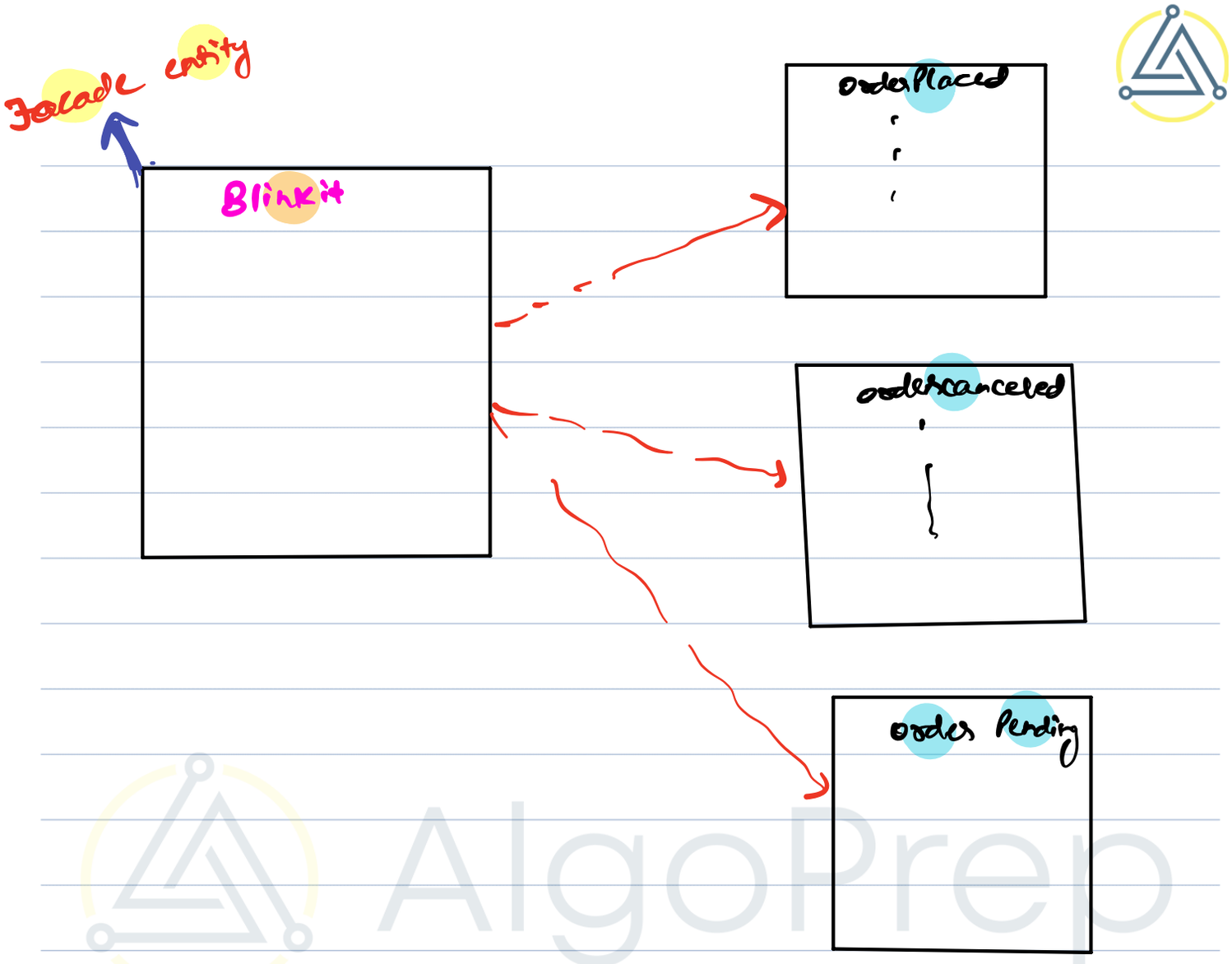
↳ outer face

→



→ SRP violation.

→ long code.


Facade entity
Blinkit
orderPlaced
orderCanceled
orders Pending