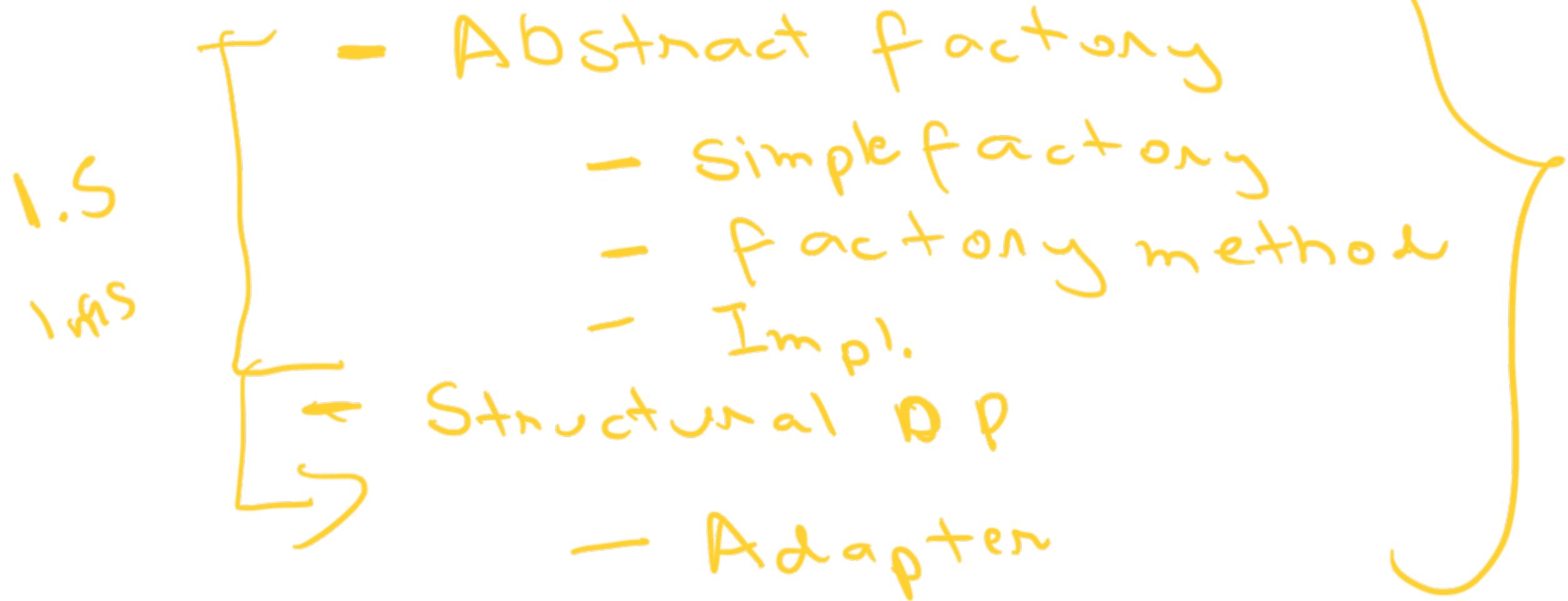


LLD - Abstract Factory and Adapter patterns

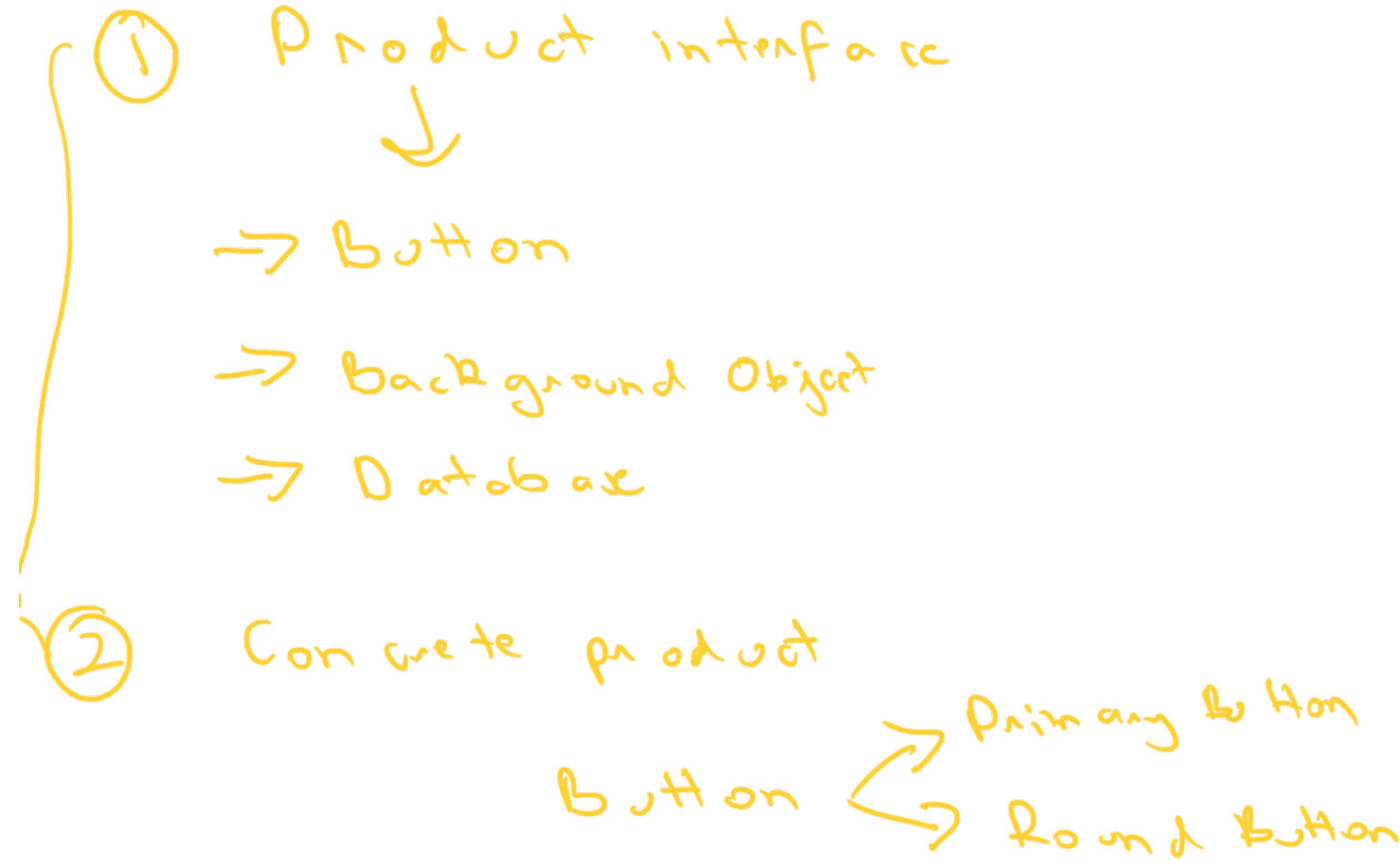
Agenda



① Simple factory

- not a design P
- do not depend on subclasses

- ~~Product interface~~ -
= maintainance right in one



Database → MySQL
→ MongoDB

③ Class - Factory

Button Factory

Database Factory

③.5 Static method

Type → Product interface

Button

Window

DbType

Conditionally create in storage

Static \rightarrow create Btn (Btn Type) {

\Rightarrow Switch (Type) {

\Rightarrow PRIMARY;

\Rightarrow return new PBC)

\Rightarrow R.

Adv \rightarrow Sub class

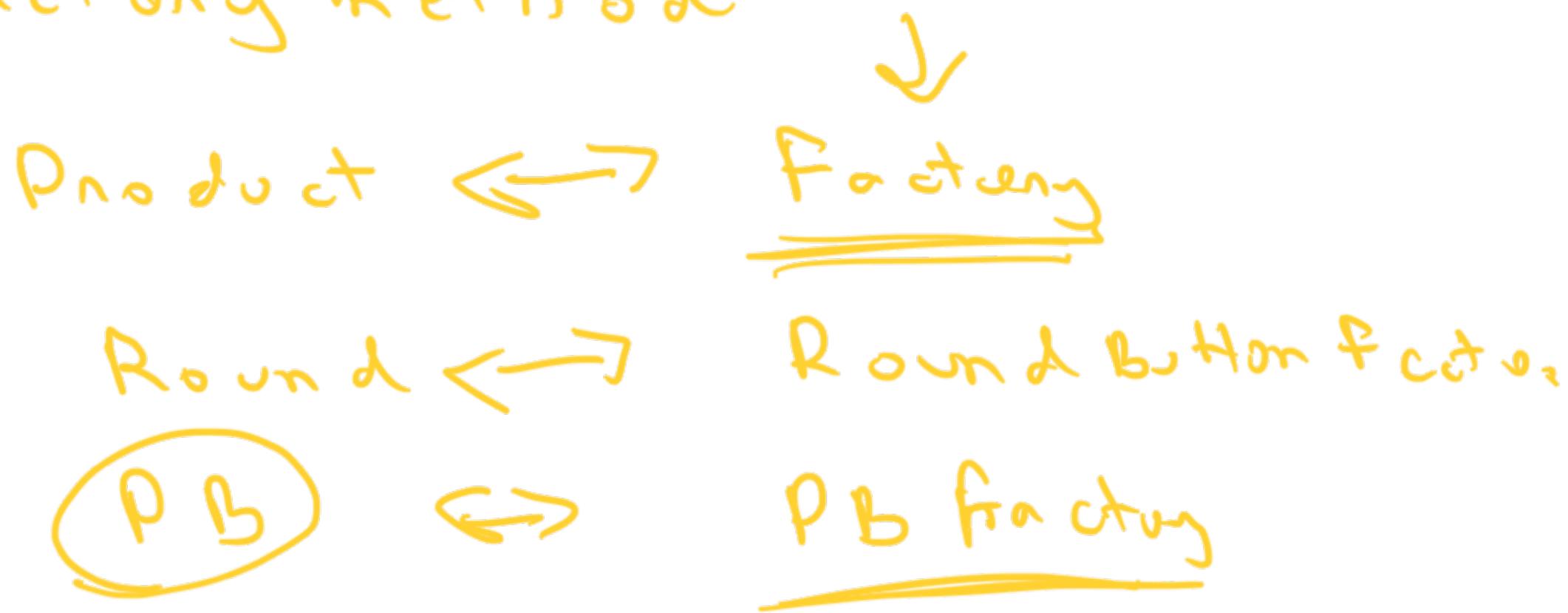
\rightarrow Simple

Down sides

\rightarrow SRF

→ common place
for construction

Factory method



① Product

② Concrete product

③ Factory interface

interface ButtonFactory {
 Button createButton();
}



④ Concrete factories





⑤

Client code

PbFactory → buttonFactory
 = new PbFS

button instance

→ factory.createButton()

Btr , Primary

• getFactory (Primary)

↳ PFactory

• getFactory (Type)

↳ factory

Advantages

| Disadvantages

① S P P ✓

② O C F - ✓

too many class.

Database



database = Database.createC

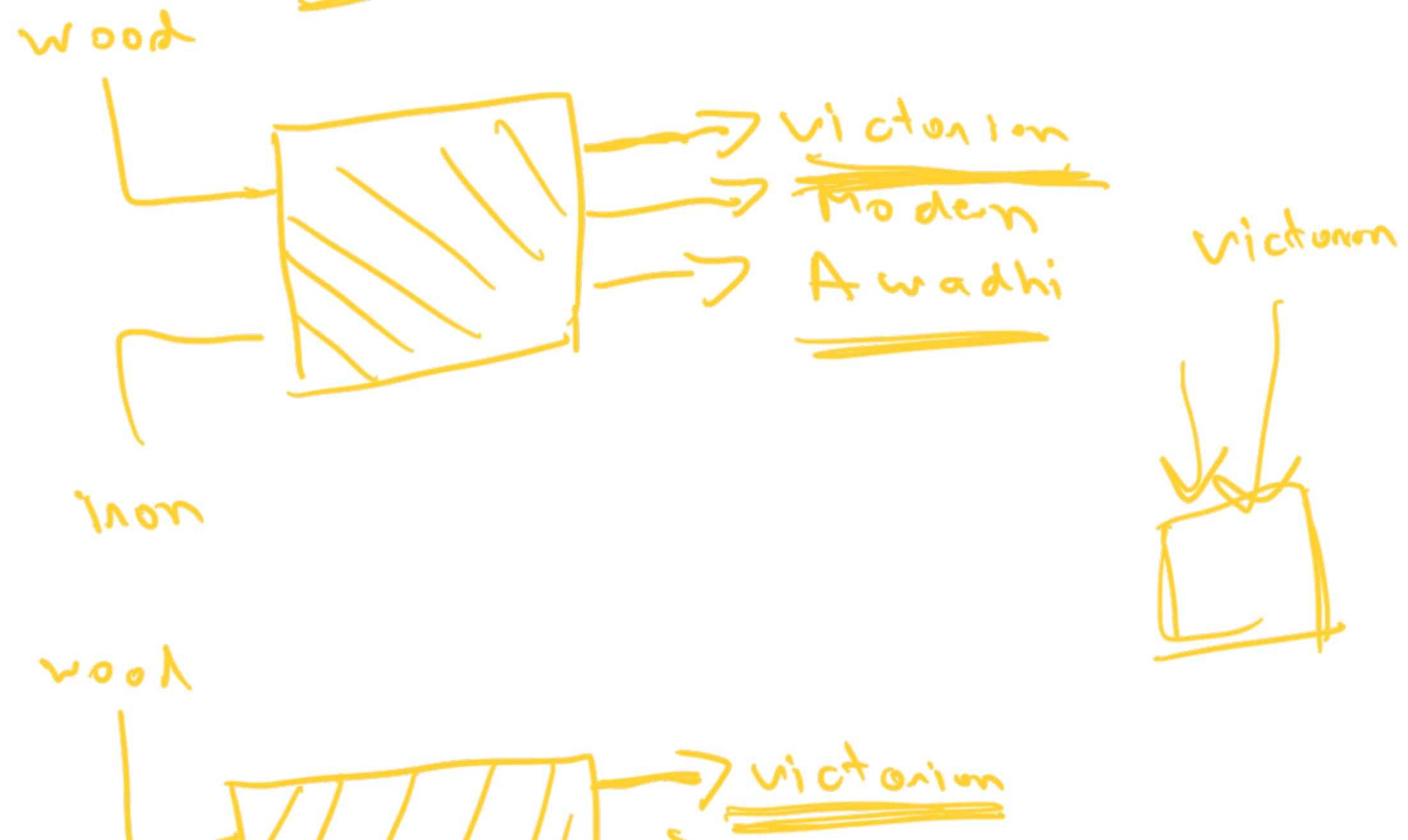
un

DB

mysql: //

mongo: //

Abstract factory





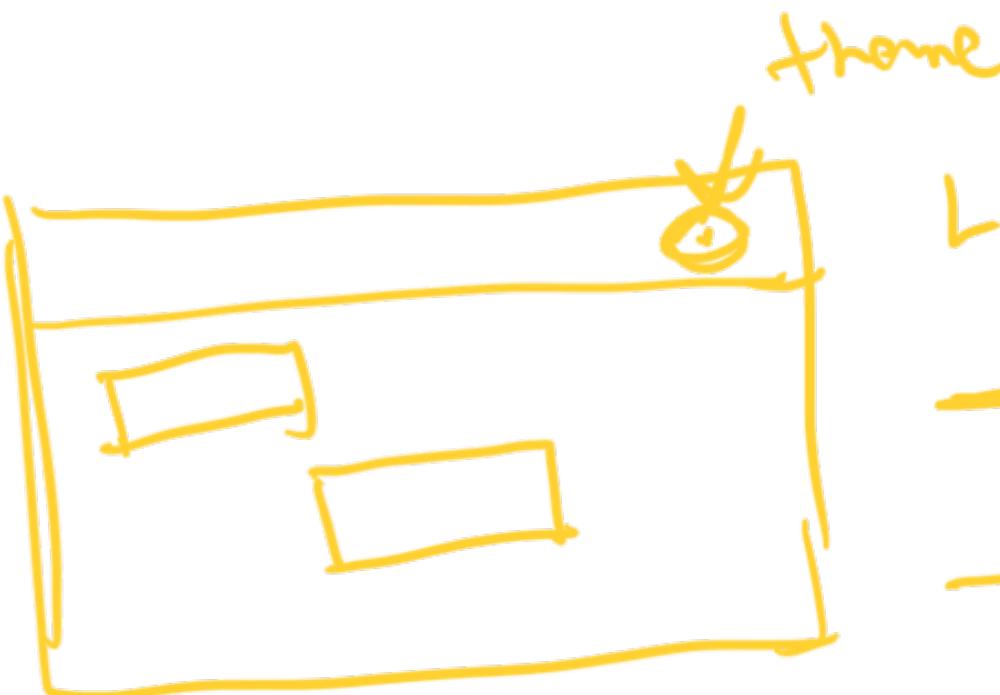
Family of products

Vibration chain + vibration tube
vibration

Abstract factory

→ related family of products

Themes



Light theme

→ Light + Btr

→ Light Checkbox

Dark theme

→ Dark Btr

→ Dark Checkbox

related family - Theme
-



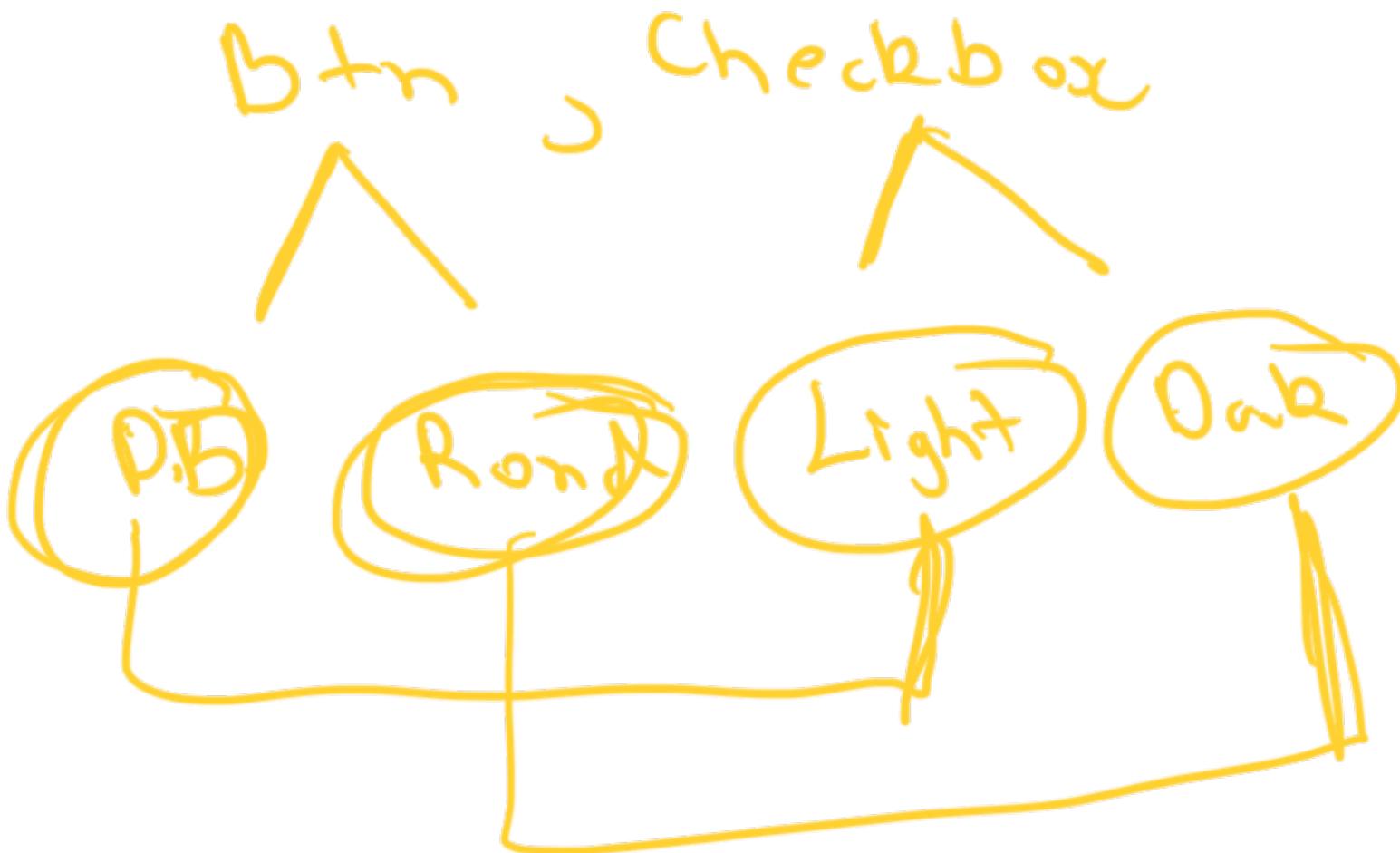
→ Related set of products

⇒ ensure that only products
of some type can be
provided to the user

→ factory of factories:



Abstract factory



PBF

RBF

DCP

LCP

S

→ btnfactory.createBtn()

→ cfactory.createCheckbox()

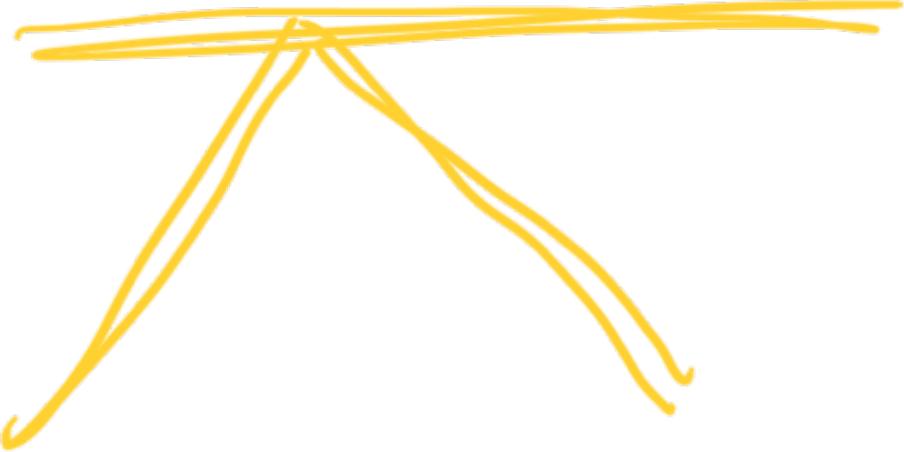
↳

Dark Checkbox + Light Button

Fai → [Light, Light]

→ [Dark, Dark]

Theme factory



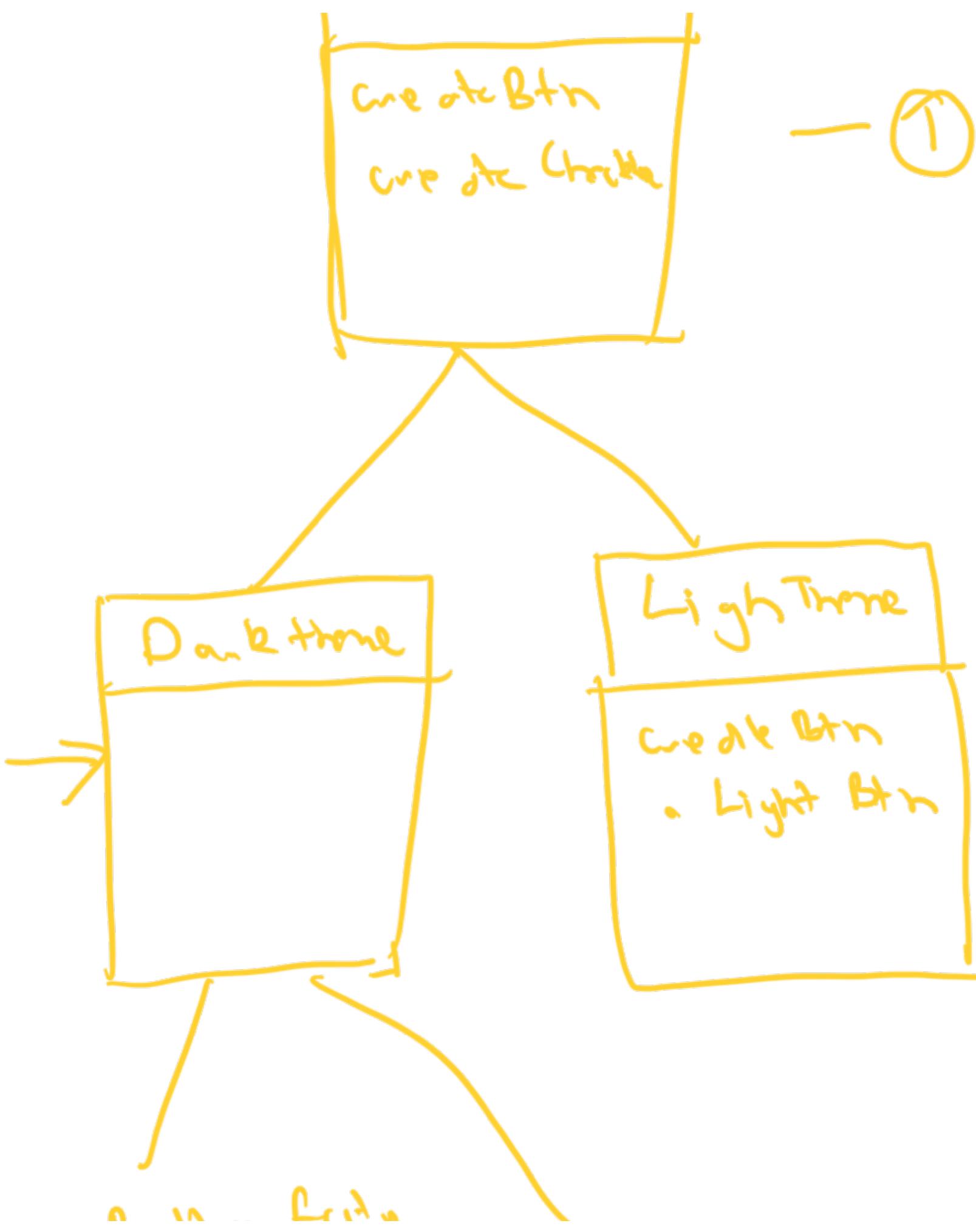
Dark Theme

Light Theme

↳ Create Dark

↳ Create Dark Checkbox

Themefactory



button

Dark

checkbox

Dark

-
- ① Product interface →
 - ② Product concrete class →
 - ③ Abstract factory interface →
 - Create btr
 - Create Checkbox



Concrete factory classes

→ Dark theme factory

→ Light theme factory

Creational

→ Singleton → Single instance

→ builder — ✓

→ Prototype → Object creation perform once

→ Factory → ? + registry

→ different types of objects

→ ctor - 1o variable

- build

- ~~Object~~ build

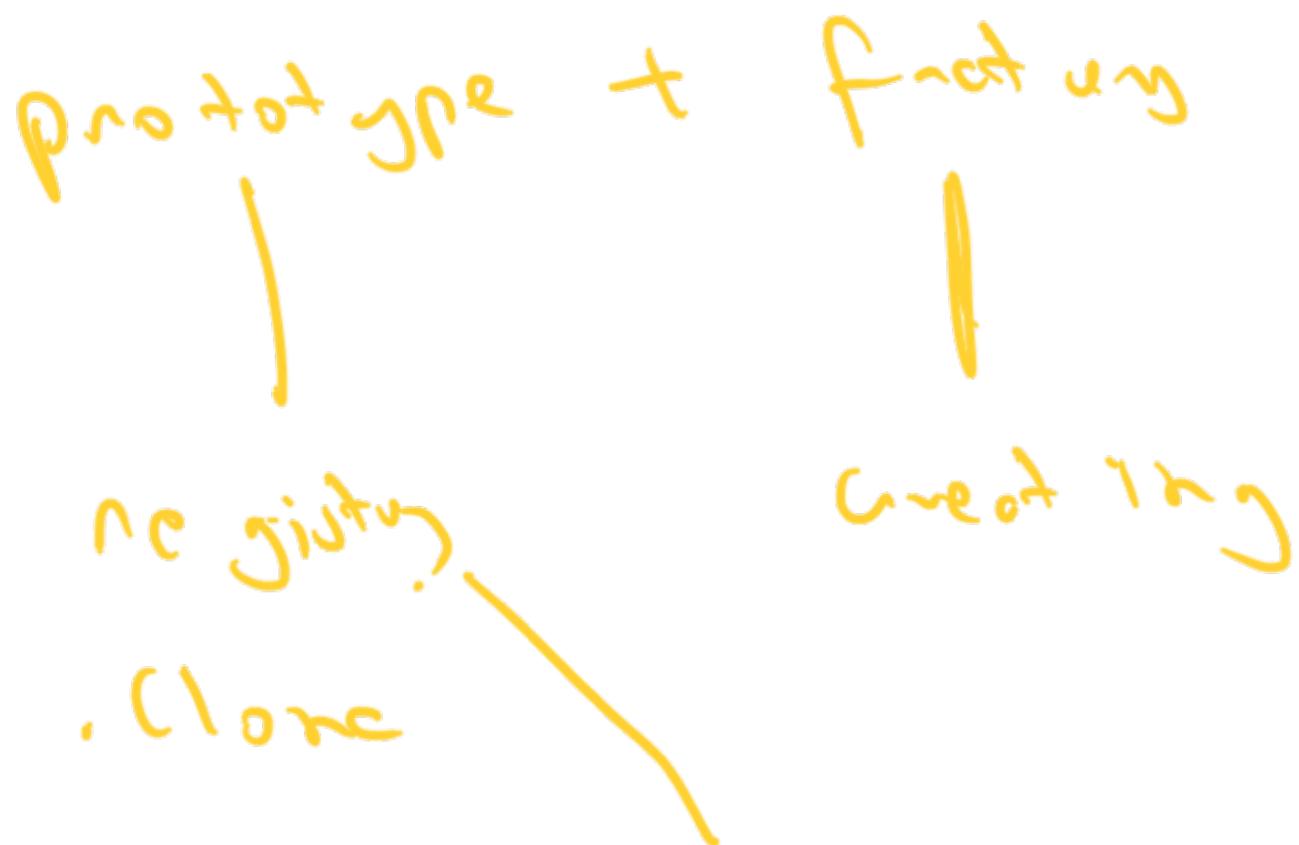
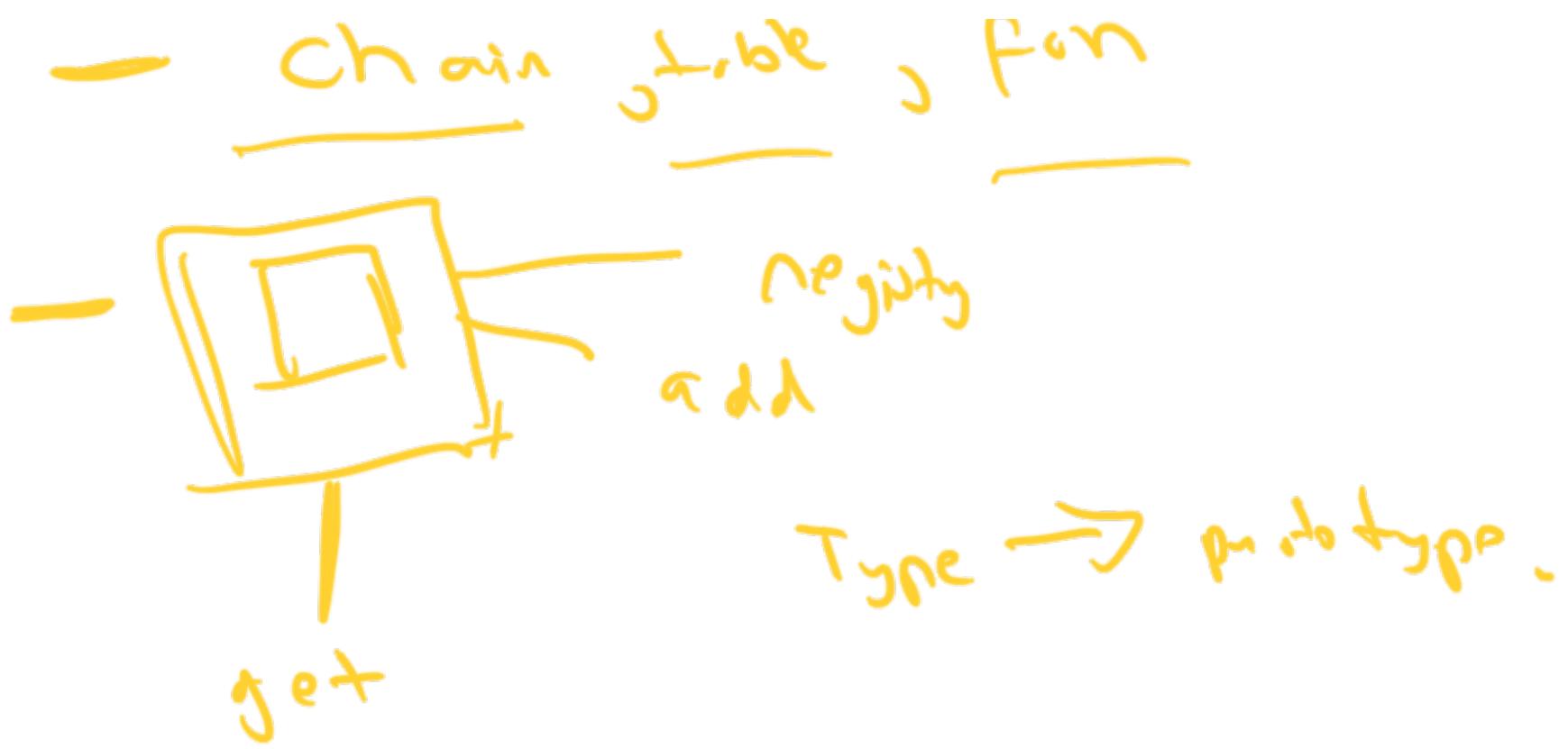
- beautiful

→ minimal, self documenting

→ not duplicated

↳ Prototype

Prototype - clone an object when creating
one for each



Singleton

break

$$6:09 = 6:15$$

$$10:45$$

Theory



Builder DD

LLD

→ use case

→ Design Tic Tac Toe



COPX

SOP →

Adopter

Decorator

Fly weight

b0p \rightarrow Ob server

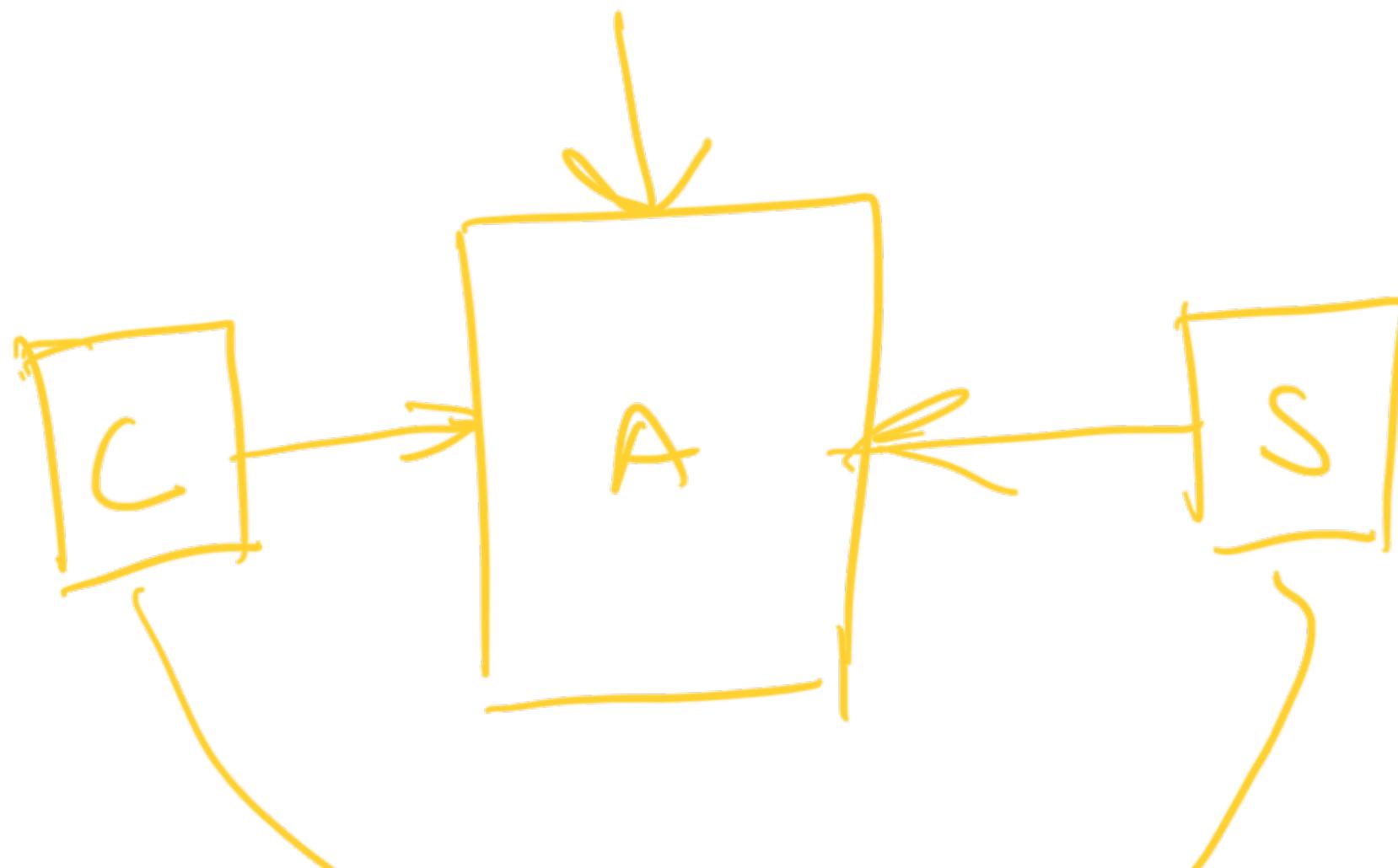
\Rightarrow Strategy

Case Studies

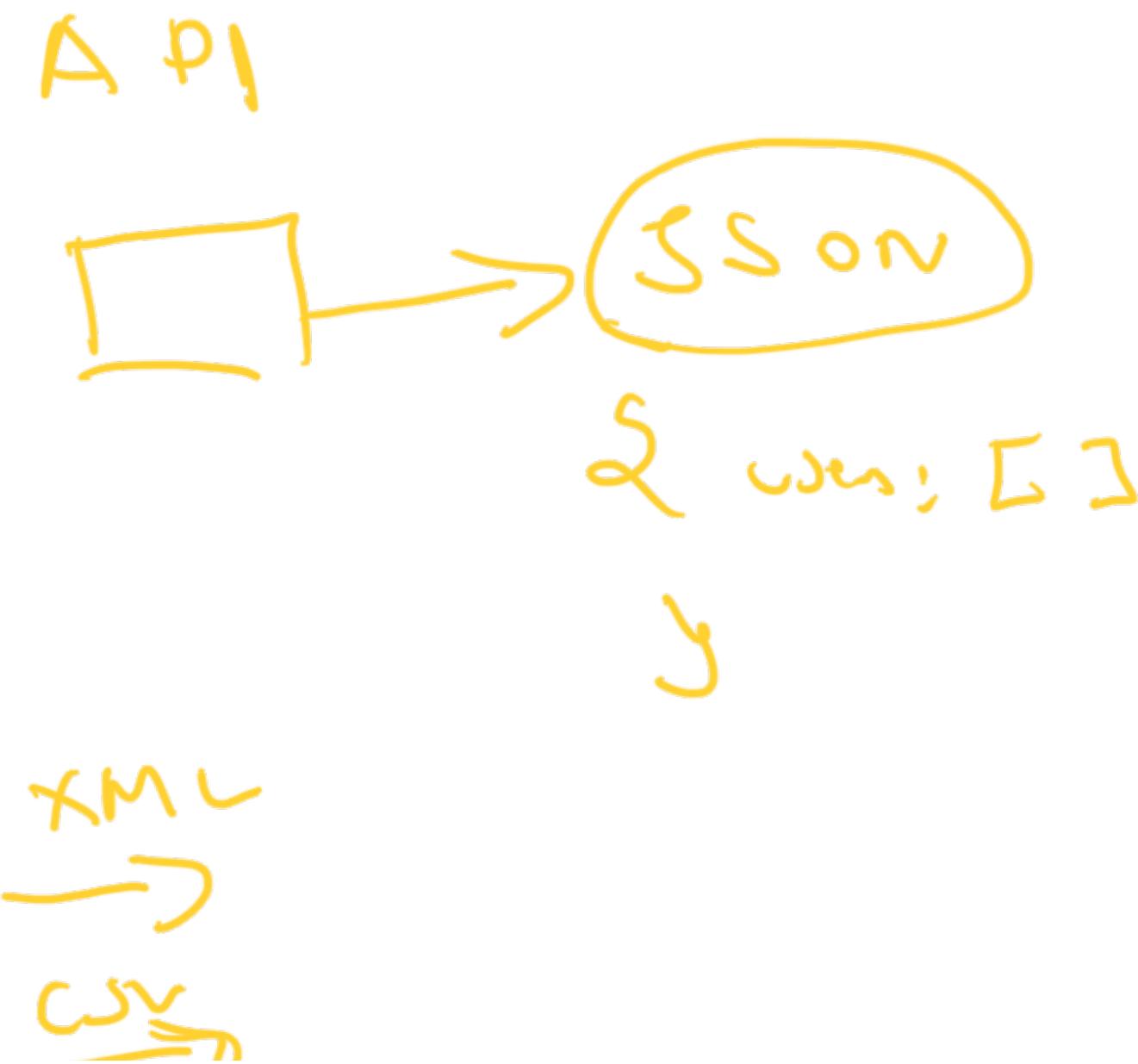
- Real world - Pen
- Game - Tic Tac Toe
- MS - Parking
 - email
 - bus
 - split

Structural Design pattern

Adapter



Software Engineering



==>
/api-JSON

/api-XML

Adopter Layer



| uses





A / option

Payment Gate way:

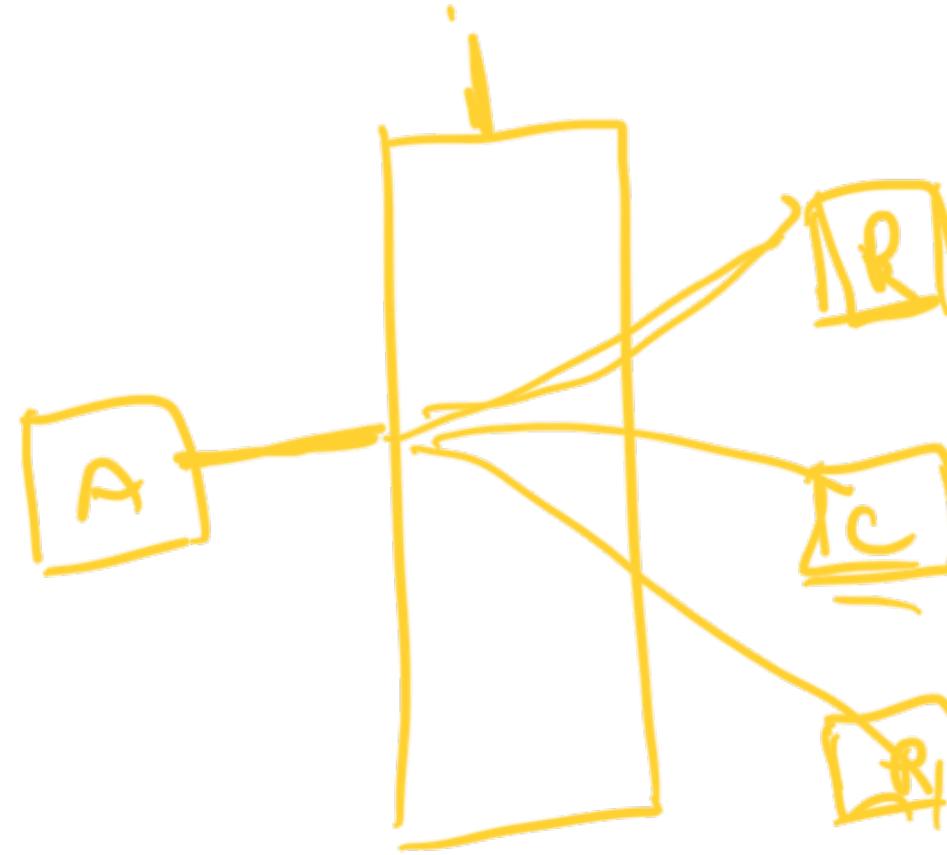
Pay on Del

Cheque

... -

→ PayU
~~PayU~~

→ Cash Free



Query tuning

→ monitoring

↓ > DB

↓ > Log

↓

25

→ Frontend - application monitoring
→ Backend - new relic, sentry
→ SQL - slow queries

EXPLAIN / ANALYZE

→ Query execution plan

→ Scan? → Table scan }
→ Rows? → Index scan }

→ Index → cost candidate