DESIGN PATTERNS

JÉRÉMY PERROUAULT

INTRODUCTION

INTRODUCTION

PROBLÉMATIQUE

- Problèmes de conception
- Multiples manifestations

- Limitent les évolutions, la maintenabilité

SOLUTIONS

- Design patterns (patrons de conception)
- Chaque pattern répond à une problématique de conception bien précis
- Utilise un vocabulaire commun

- Ils sont fiables, reconnus
- Ils facilitent la conception, la maintenabilité et les évolutions (s'ils sont bien utilisés …)

SOLUTIONS

- Evite de « réinventer la roue »
- Facilite la communication entre les développeurs

TYPES

DIFFÉRENTS TYPES

TYPES

- Architectural patterns
  - MVC ou MVVM par exemple
- Design patterns
  - Issus du livre « Elements of Reusable Object-Oriented Software »
    - Publié en 1994 par Erich Gamme, Richard Helm, Ralph Johnson & John Vlissides
    - Qu'on surnomme « Gang of Four » (ou GoF)
- Idioms (ou coding) patterns
  - Liés à un langage en particulier
- Anti-patterns
  - Pattern mal utilisé, mal approprié

CATÉGORIES

GOF

CATÉGORIES

- Création
  - Rendre le système indépendant de la façon dont les objets sont créés
- Structure
  - Assembler les objets, séparer l'interface de l'implémentation
- Comportement
  - Comment communiquent les objets entre eux

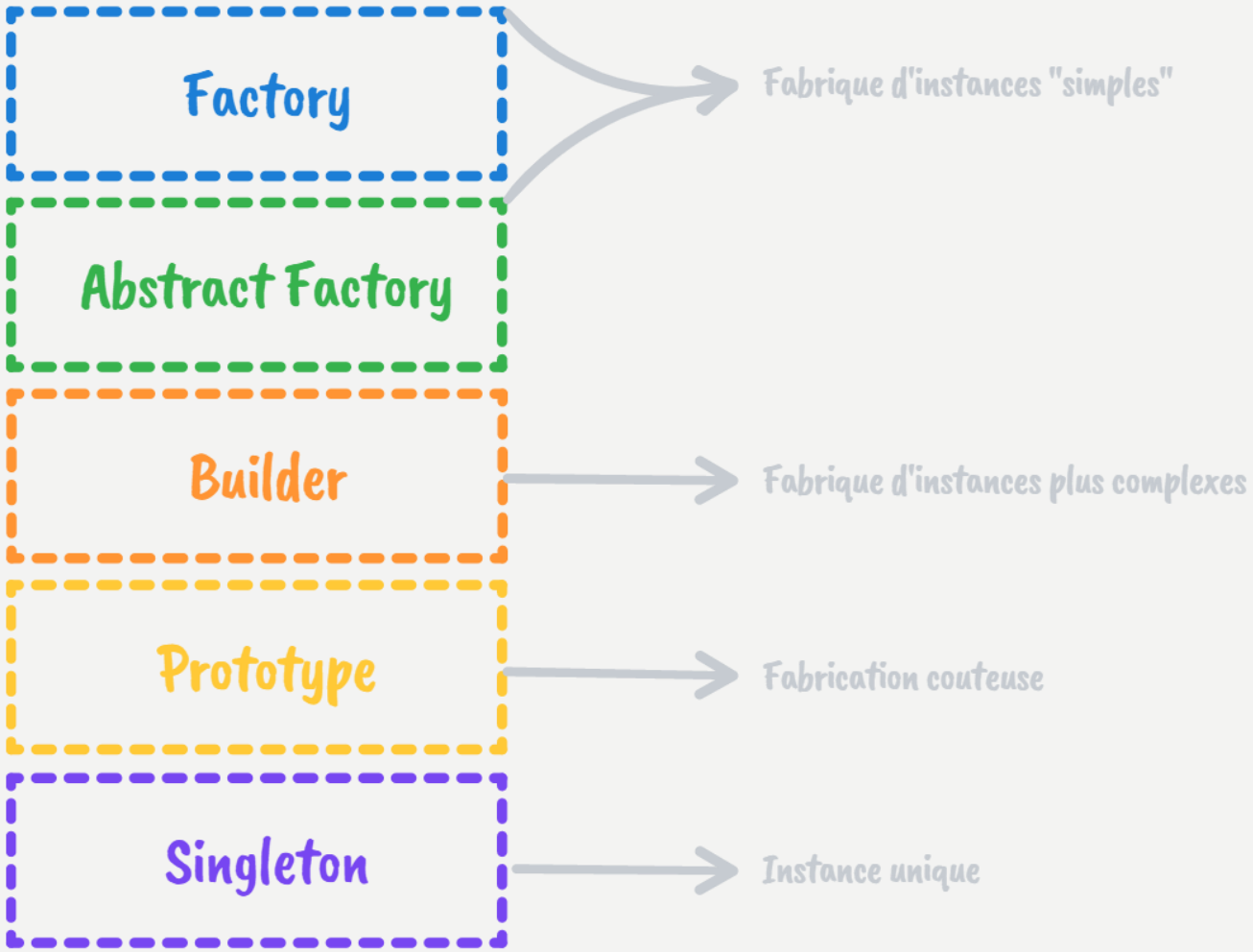
CATÉGORIES

Création	Structure	Comportement
Factory	Adapter	Strategy
Abstract Factory	Bridge	Template Method
Builder	Composite	Chain of Responsability
Prototype	Decorator	Command
Singleton	Facade	Iterator
	Flyweight	Mediator
	Proxy	Mememto
		Observer
		State
		Visitor

CRÉATION

GOF – CRÉATION

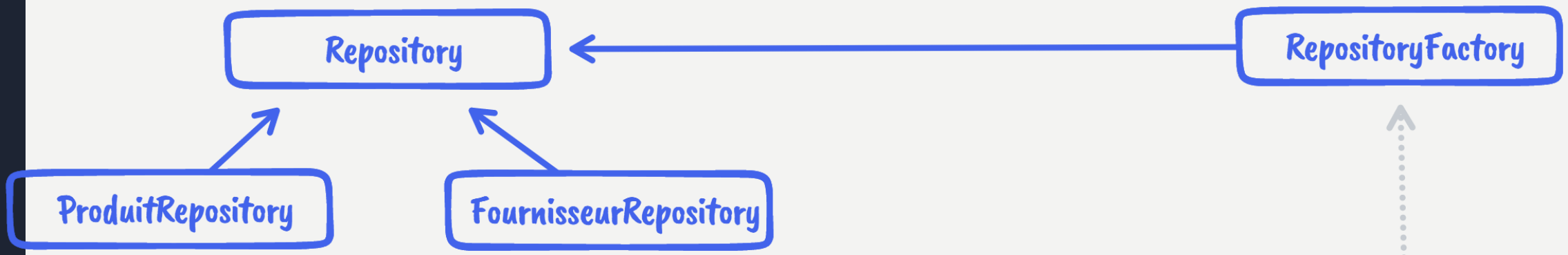
Création


Factory
Fabrique d'instances "simples"
Abstract Factory
Builder
Fabrique d'instances plus complexes
Prototype
Fabrication couteuse
Singleton
Instance unique

CRÉATION – FACTORY

- Problème de conception
  - Instanciation selon les cas, ou instanciation complexe
- Principe
  - Classe « usine » qui instancie d'autres classes
  - Encapsule la logique de création
  - Délégation de l'instanciation
- Exemple d'utilisation
  - Création d'une connexion vers une base de données

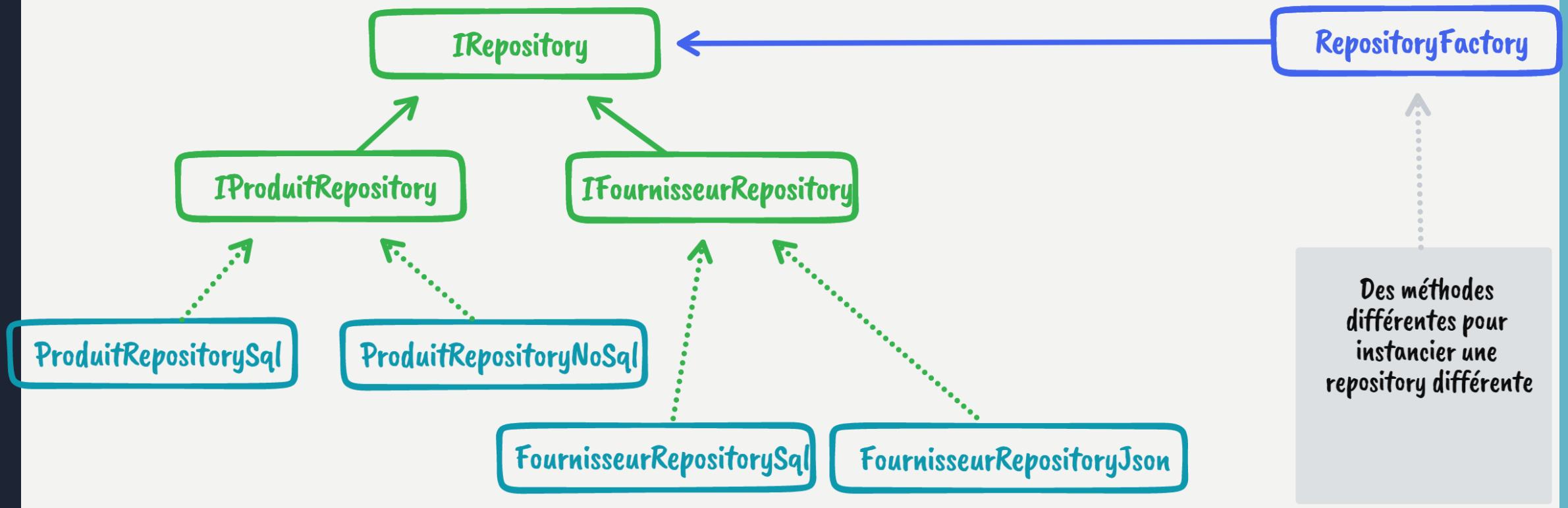
CRÉATION – FACTORY



CRÉATION – ABSTRACT FACTORY

- Problème de conception
  - Disposer d'une interface pour créer des objets d'une même famille sans préciser leurs classes concrètes
- Principe
  - Identique à *Factory*
  - Manipuler les interfaces pour une indépendance par rapport aux classes concrètes
- Exemple d'utilisation
  - Instancier des implémentations de ProduitRepository ou FournisseurRepository
    - Sans se préoccuper de savoir si c'est du SQL, NoSQL, etc.

CRÉATION – ABSTRACT FACTORY



CRÉATION – BUILDER

- Problème de conception
  - Créer un objet complexe en appliquant plusieurs attributs différents
    - Certains attributs peuvent être oublié
    - La validation des informations est peut-être à mettre en place (les constructeurs qui lèvent une Exception sont à éviter)

- Principe
  - Classe « monteur » qui gère les différents attributs, les validations éventuelles, et la création de l'instance
  - Classe « directeur » peut être ajoutée pour réutiliser une configuration particulière

- Exemple d'utilisation
  - Créer des fenêtres IHM qui ont toutes un titre, des boutons, mais chacune avec leur spécificité

CRÉATION – BUILDER

Existant

Pattern


```
new Produit.Builder()
.withNom("Le nom")
.withStock(10)
.build()
```

Autre variante :

```
ProduitBuilder
.nom(..)
.stock(..)
.build()
```

CRÉATION – PROTOTYPE

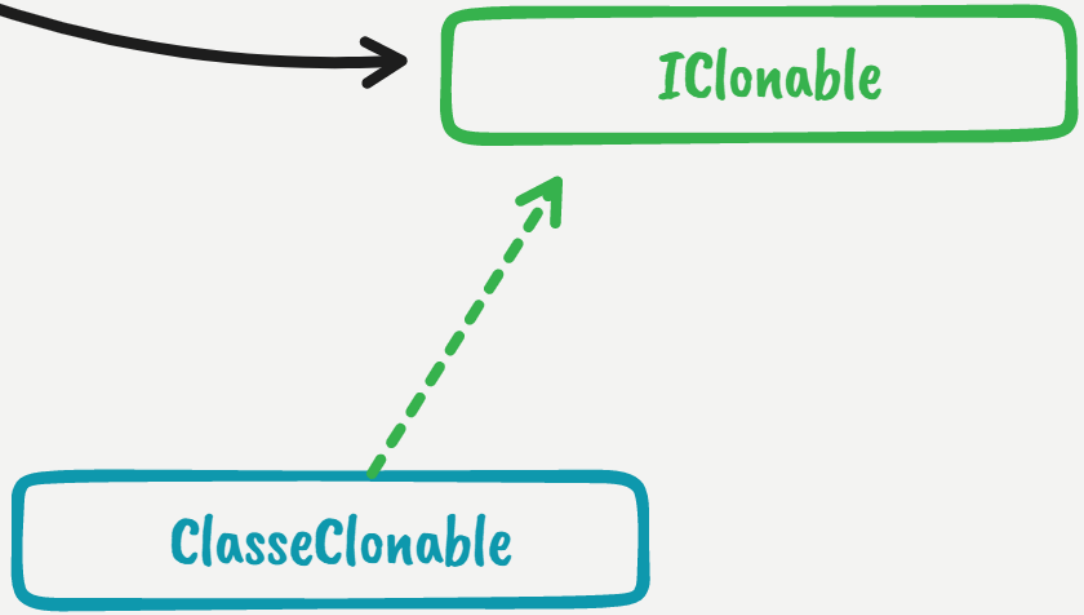
- Problèmes de conception
  - Création d'un objet coûteuse en ressources
  - Création d'objet dont on ignore le type mais dont on dispose d'instance(s)

- Principe
  - Dupliquer l'instance dont on dispose pour créer un nouvel objet

- Exemple d'utilisation
  - Copie d'un élément (Connexion SQL)

CRÉATION – PROTOTYPE

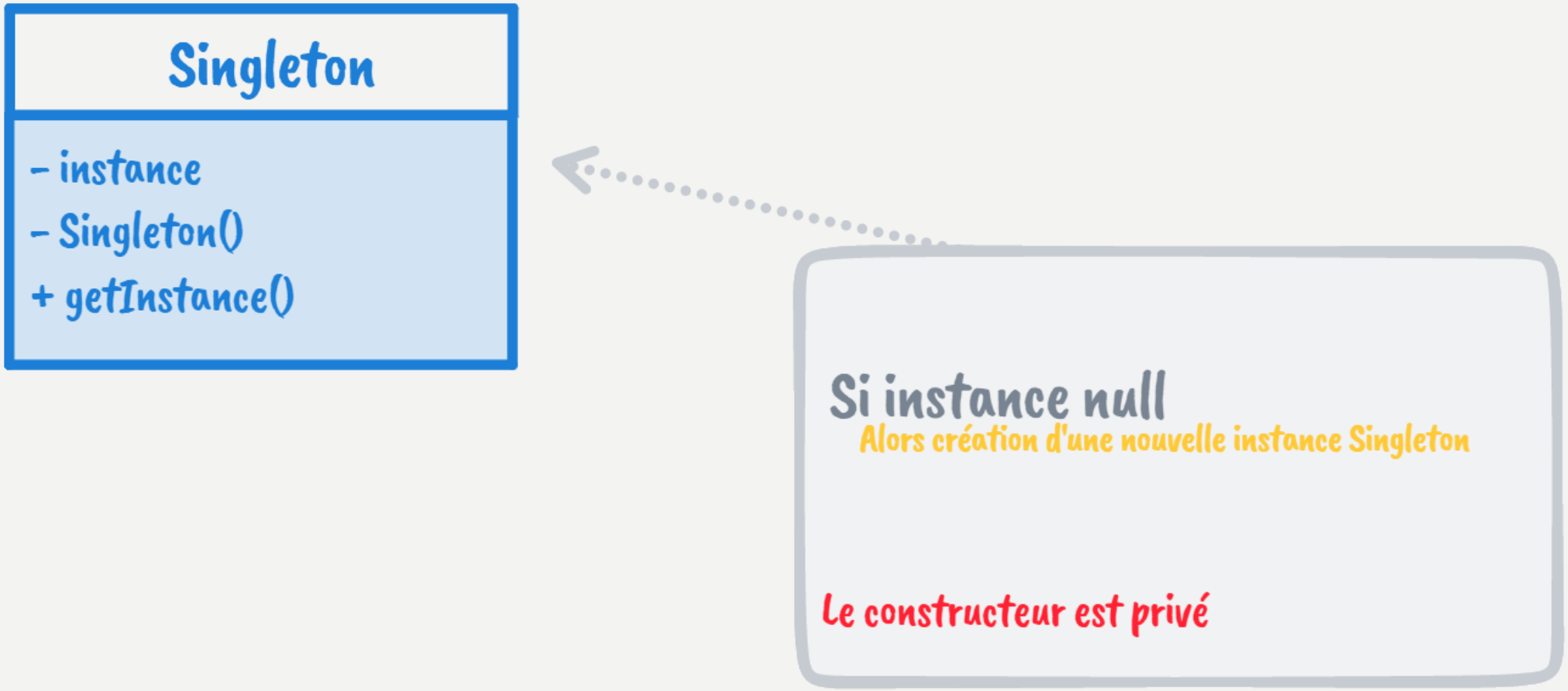
Implémentation d'une méthode "clone" qui copiera les propriétés de l'instance



CRÉATION – SINGLETON

- Problème de conception
  - Instancier une seule fois la classe
- Principe
  - Constructeur privé (ou protégé) pour empêcher l'instanciation depuis l'extérieur
  - Attribut statique *Instance* qui contient l'instance de la classe
  - Méthode statique *getInstance()* qui instancie la classe et la stocke dans *Instance*
- Exemple d'utilisation
  - Un contexte d'utilisation

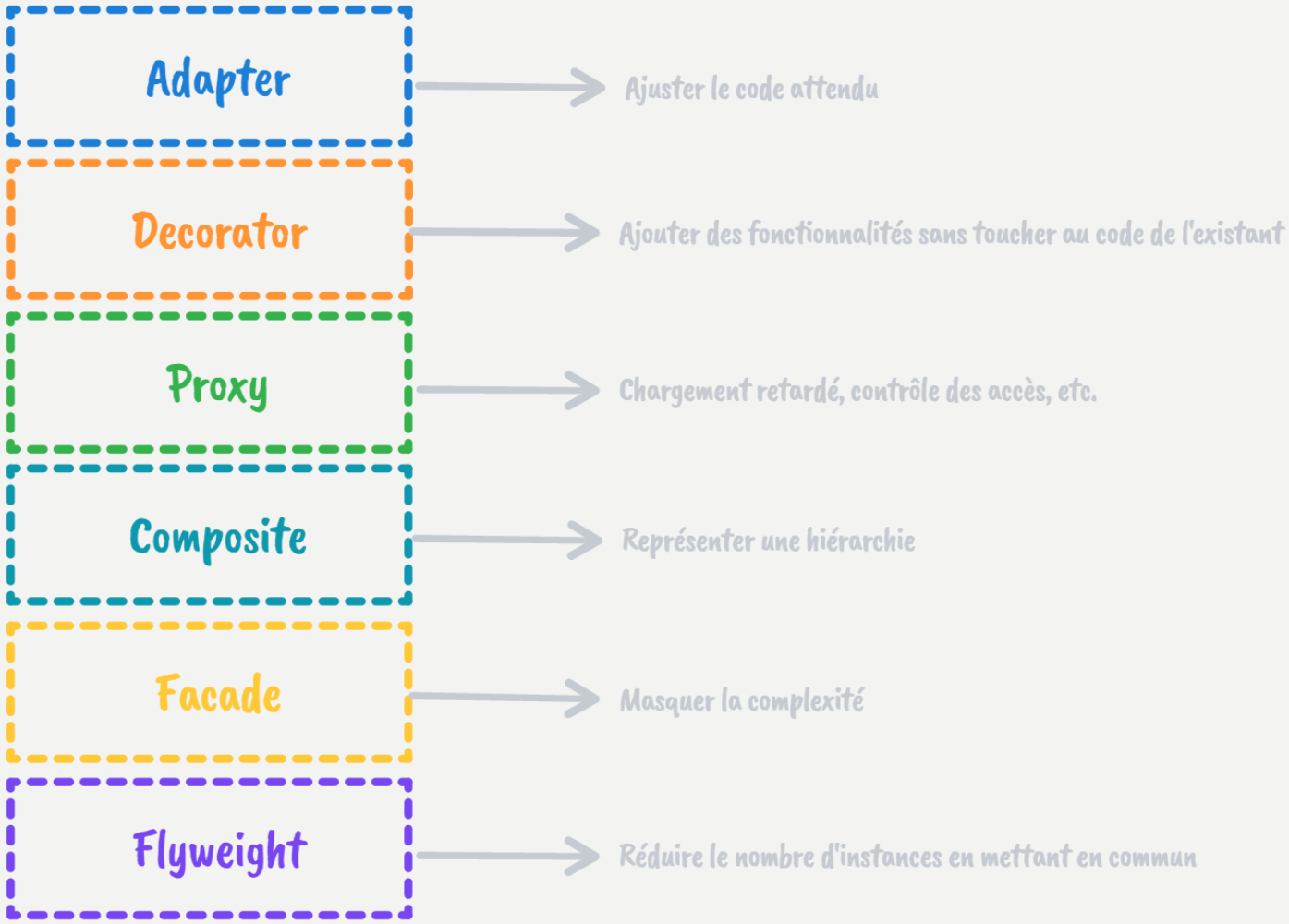
CRÉATION – SINGLETON



STRUCTURE

GOF – STRUCTURE

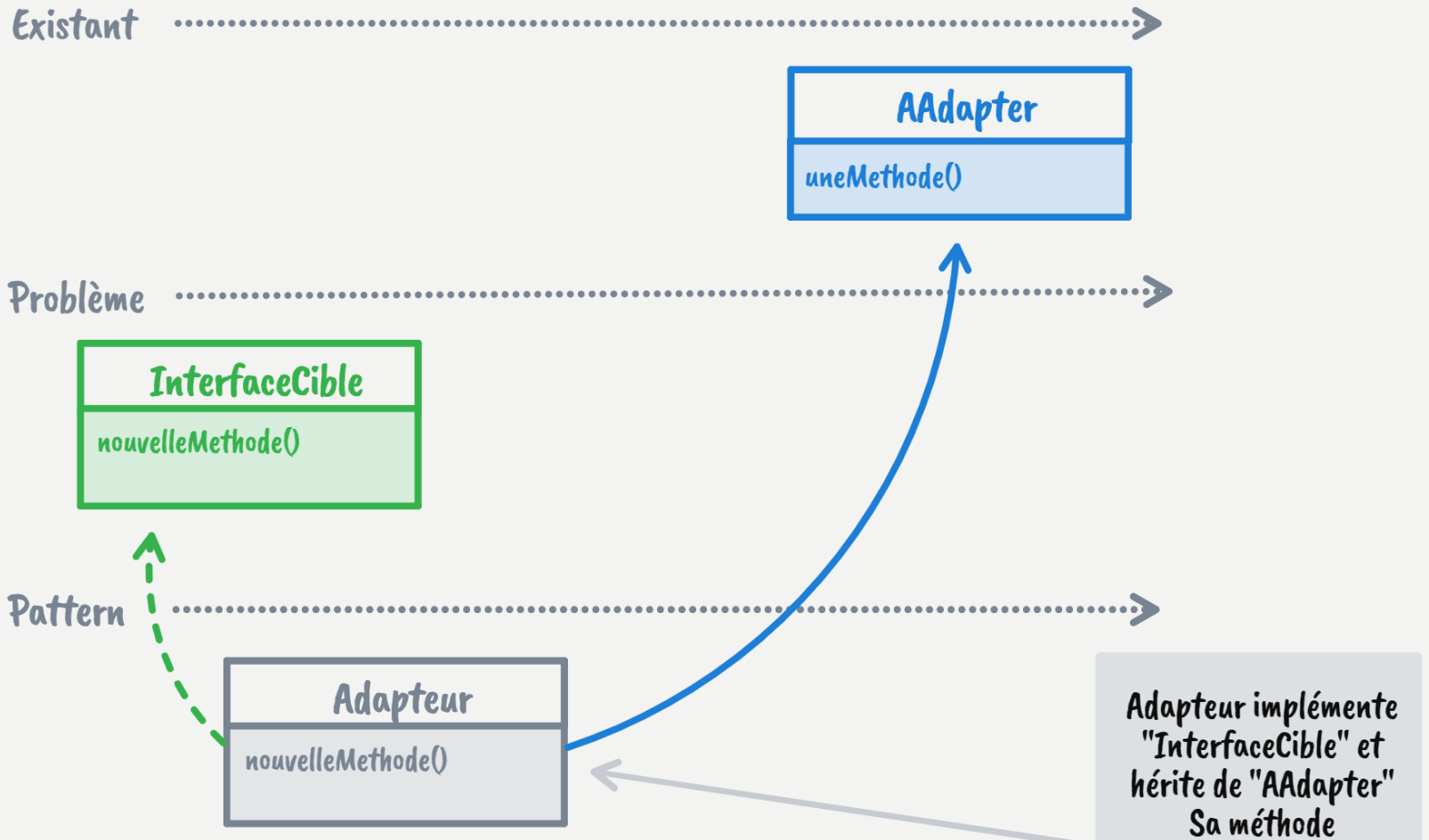
Structure


Adapter
Ajuster le code attendu
Decorator
Ajouter des fonctionnalités sans toucher au code de l'existant
Proxy
Chargement retardé, contrôle des accès, etc.
Composite
Représenter une hiérarchie
Facade
Masquer la complexité
Flyweight
Réduire le nombre d'instances en mettant en commun

STRUCTURE – ADAPTER

- Problème de conception
  - Ajuster l'interface d'un objet à celle attendue par le code client
- Principe
  - Conserver l'instance de la classe adaptée
  - Convertir les appels d'une interface existante vers l'interface implémentée
- Exemple d'utilisation
  - Garder une compatibilité entre versions d'une interface

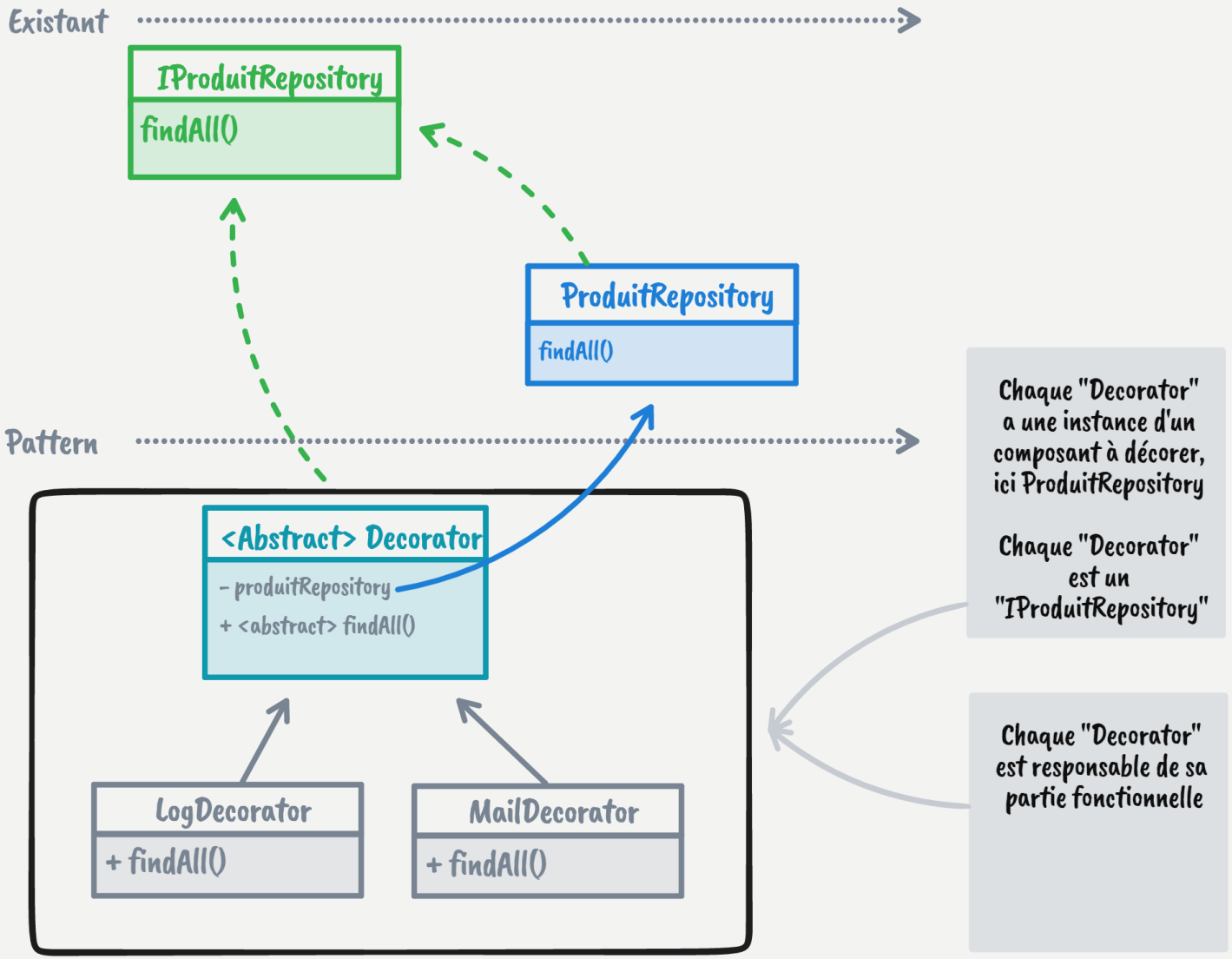
STRUCTURE – ADAPTER



STRUCTURE – DECORATOR

- Problème de conception
  - Rajouter des fonctionnalités à des composants existants sans modifier le code existant
- Principe
  - Encapsuler l'objet existant et y ajouter des nouveaux comportements (responsabilités)
  - Possibilité d'ajouter ou retirer dynamiquement des responsabilités
- Exemple d'utilisation
  - Ajouter de la journalisation ET d'une alerte e-mail

STRUCTURE – DECORATOR



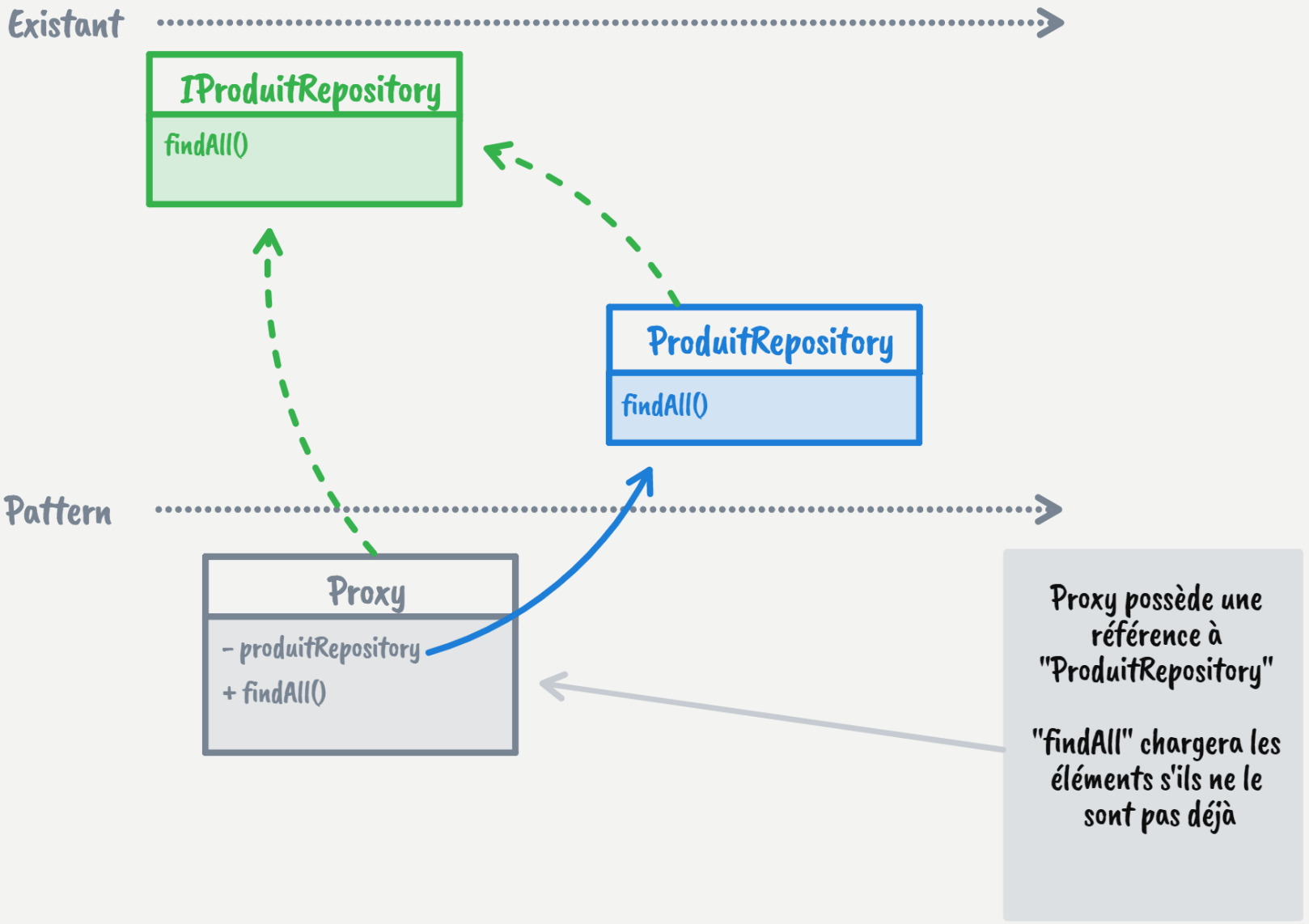
STRUCTURE – PROXY

- Problèmes de conception
  - Donner l'accès à un objet sans avoir à le transmettre
  - Ne charger qu'un cas de besoin (lazy loading)
  - Autoriser l'accès uniquement si les conditions sont satisfaites

- Principe
  - Le proxy est une classe intermédiaire de même abstraction que l'implémentation finale

- Exemples d'utilisation
  - Charger la collection de produits uniquement si on en a besoin
  - Mettre en place un cache
  - Synchronisation

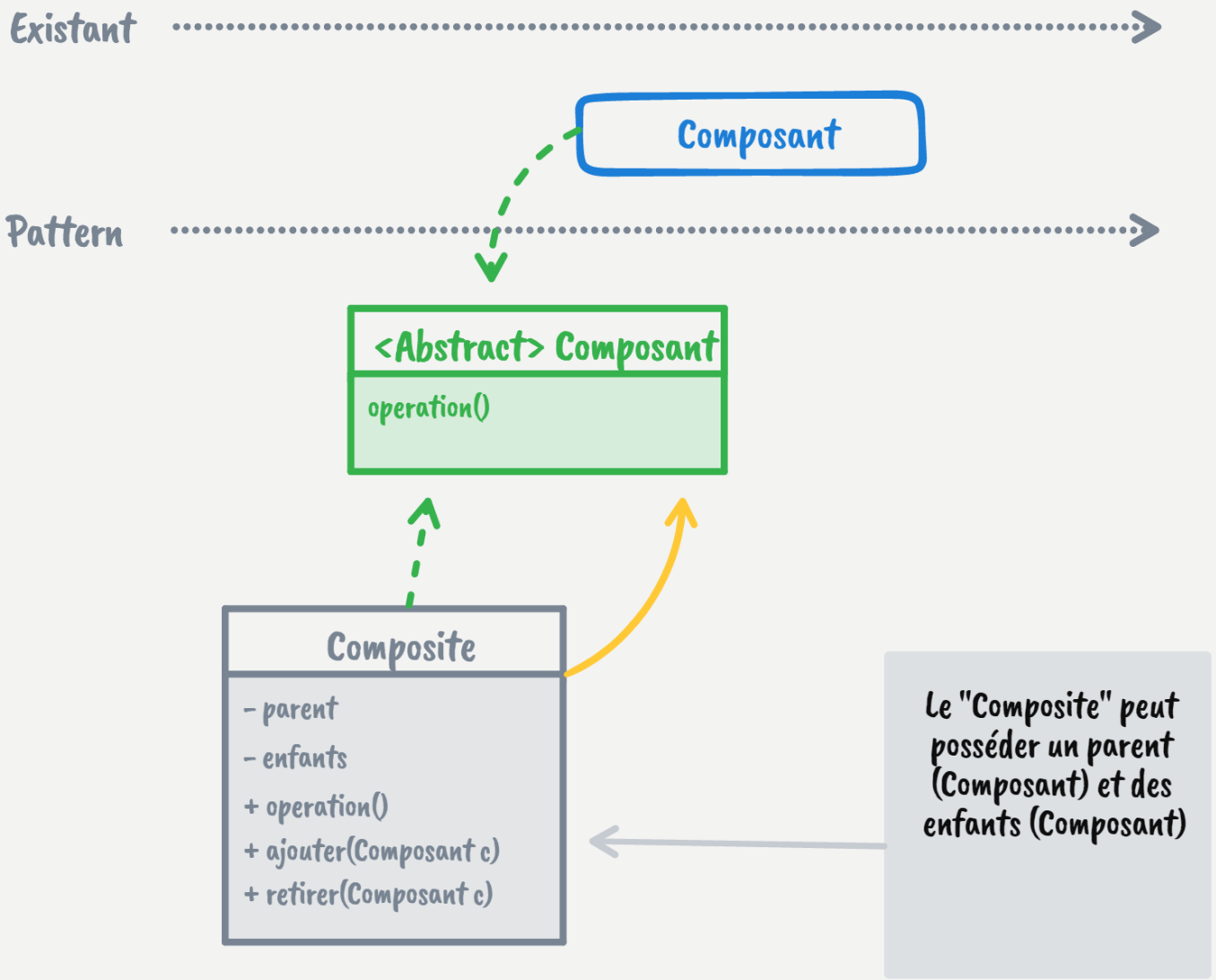
STRUCTURE – PROXY



STRUCTURE – COMPOSITE

- Problème de conception
  - Représenter une hiérarchie
  - Pouvoir manipuler un objet unique et un objet composé de la même façon
- Principe
  - Classe abstraite composant qui peut être unique ou composé
  - Classe unique et classe composé
- Exemple d'utilisation
  - Une liste de catégories qui peuvent contenir d'autres catégories

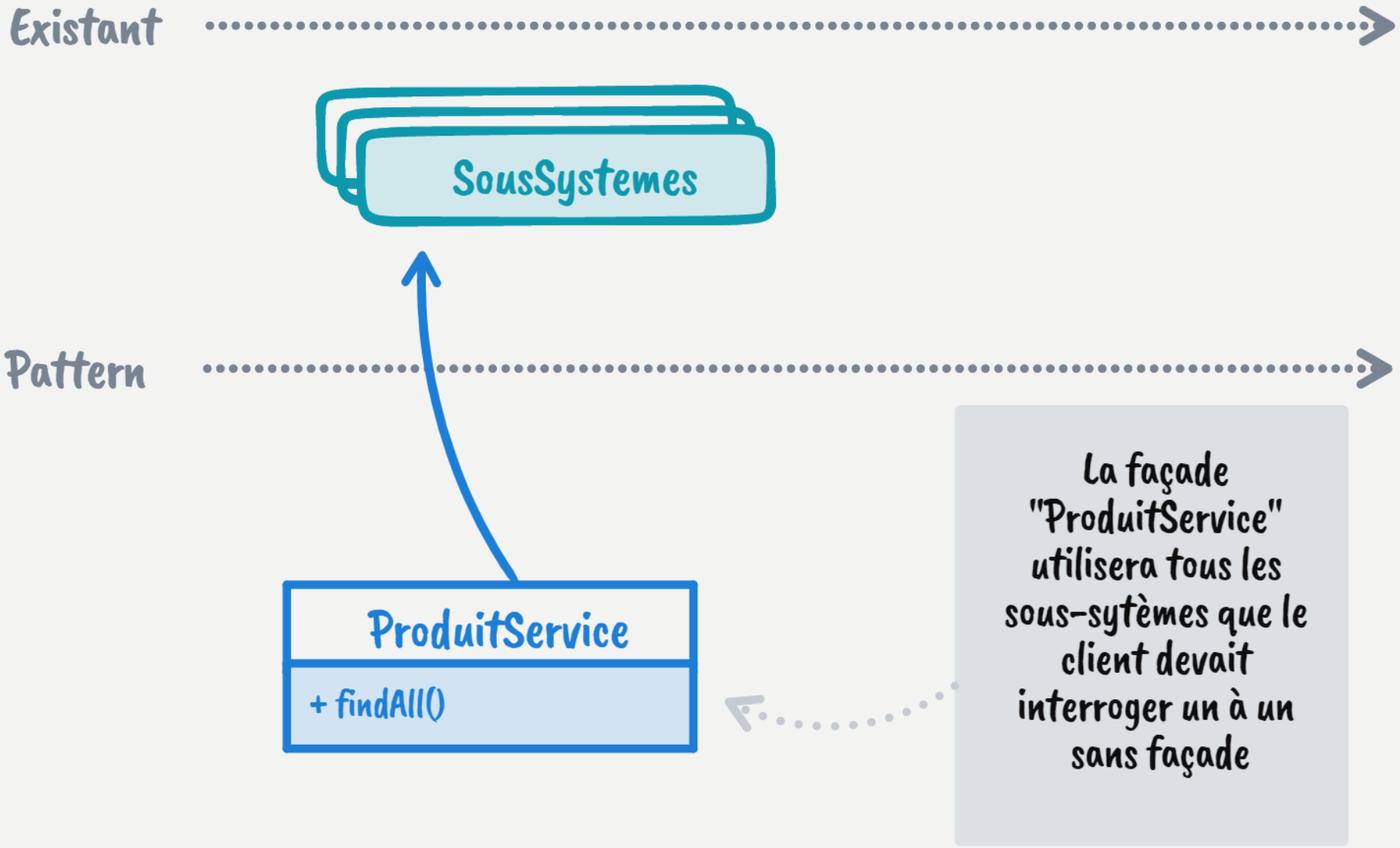
STRUCTURE – COMPOSITE



STRUCTURE – FACADE

- Problème de conception
  - Masquer la complexité d'une API qui répond à un besoin simple
- Principe
  - Encapsuler l'ensemble des interfaces d'un sous-système dans une interface unique
- Exemple d'utilisation
  - Agrégation de services web

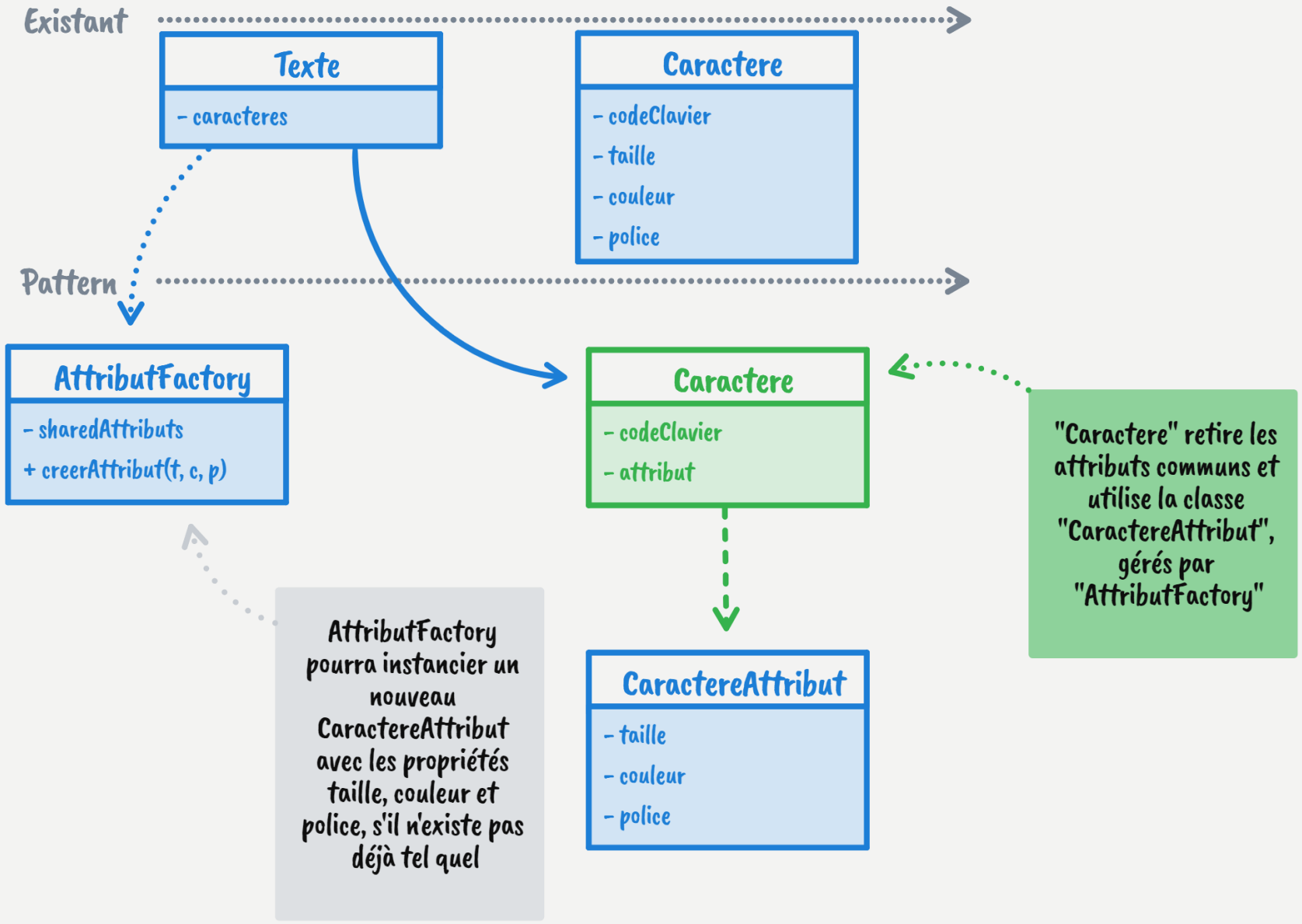
STRUCTURE – FACADE



STRUCTURE – FLYWEIGHT

- Problème de conception
  - Grand nombre d'objets, avec des similitudes dans les valeurs
  - Coût de stockage élevé
- Principe
  - Chercher à réduire le nombre d'instances en mettant à disposition des instances « partagées » (états extrinsèques) et d'autres « non partagées » (états intrinsèques)
- Exemple d'utilisation
  - Editeur de texte, avec chaque caractère qui possède les propriétés couleur, taille, police

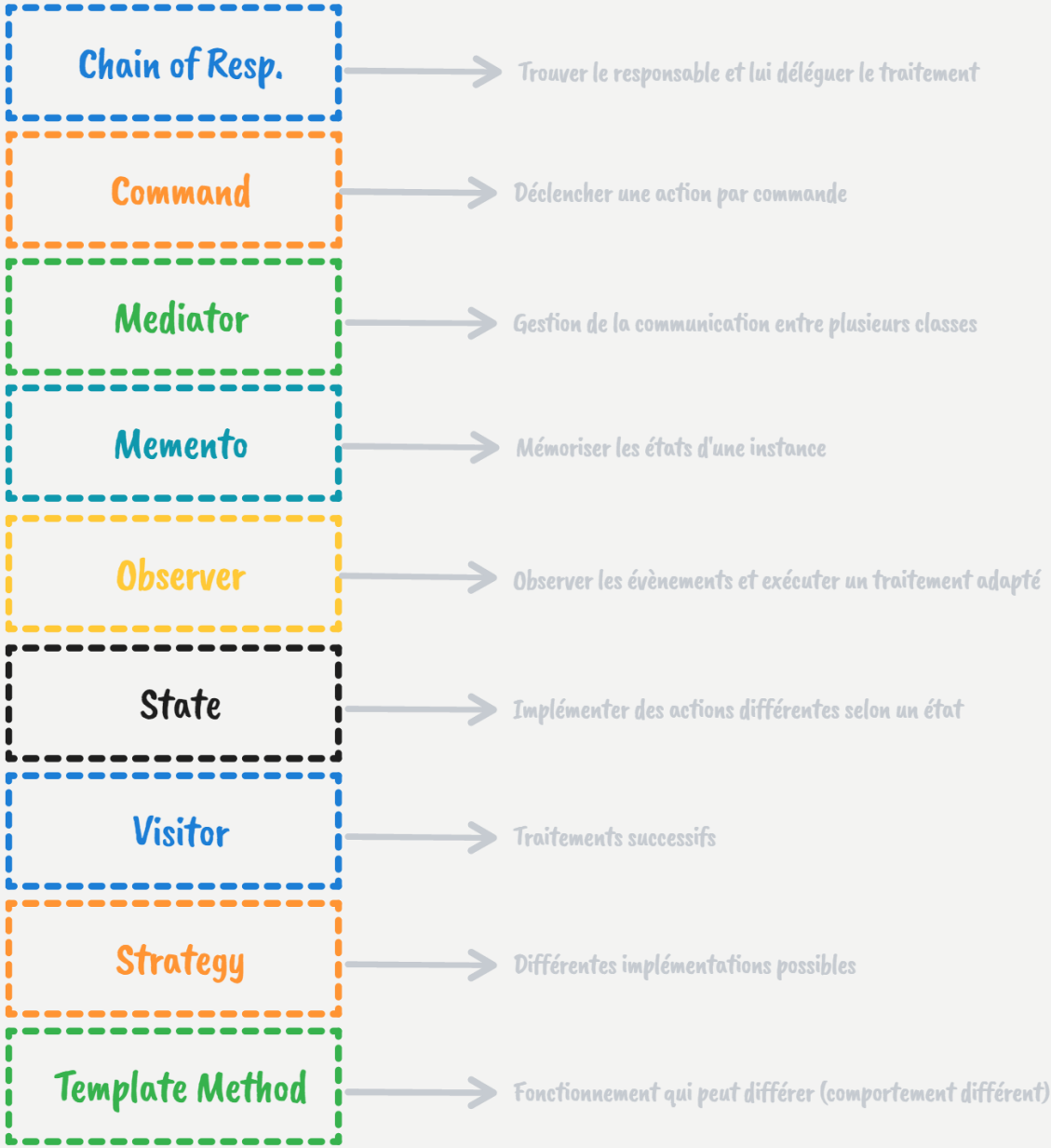
STRUCTURE – FLYWEIGHT



COMPORTEMENT

GOF – COMPORTEMENT

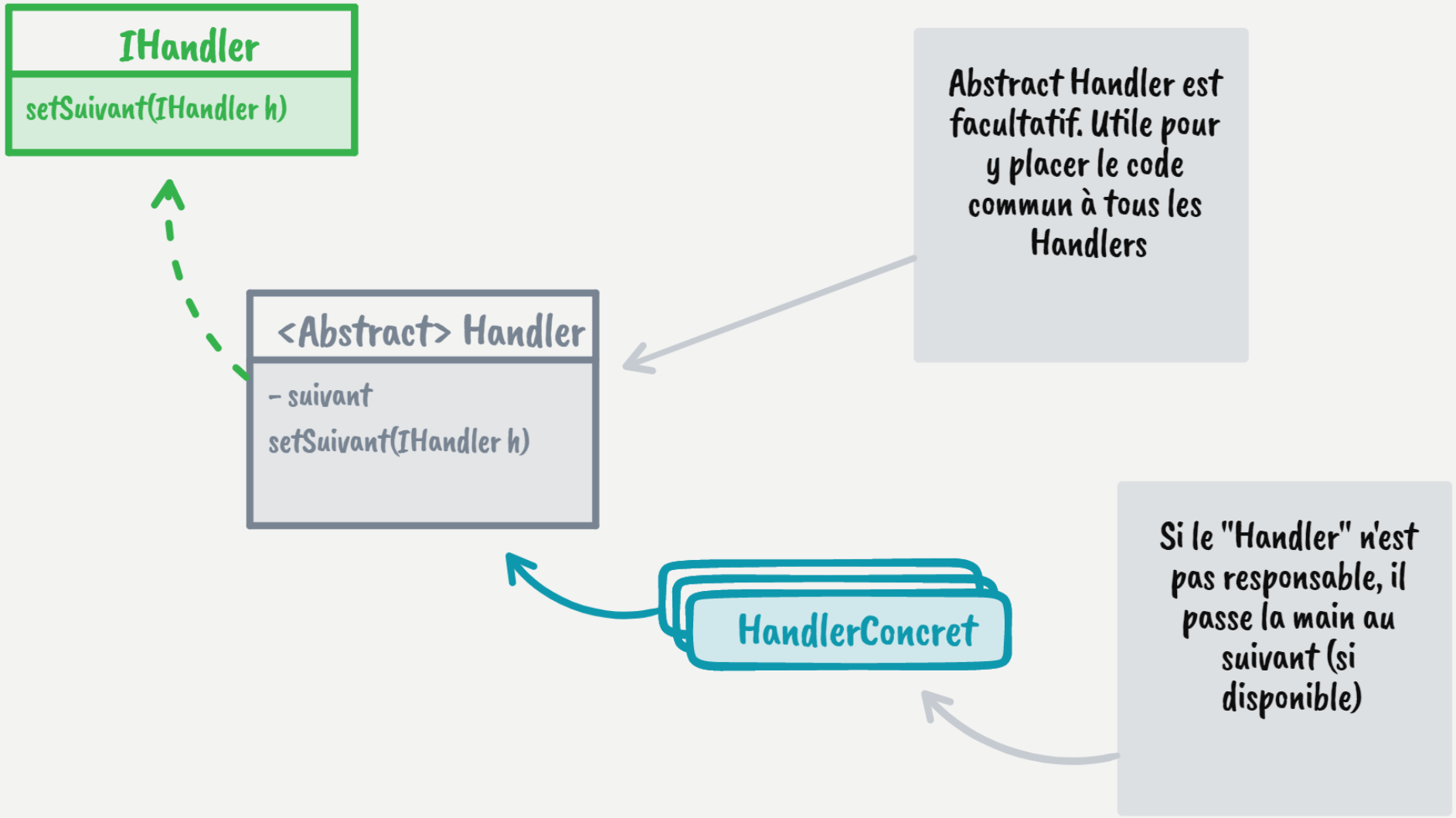
Comportement



COMPORTEMENT – CHAIN OF RESP.

- Problème de conception
  - Plusieurs objets sont capables de traiter la demande, mais on ne sait pas lequel
- Principe
  - Transmettre l'information successivement (chaîner l'information) aux différents objets jusqu'à trouver le responsable
- Exemples d'utilisation
  - Rechercher un code produit qui peut se trouver dans plusieurs systèmes de gestion
  - Mettre en place différents mécanismes de filtres pour une requête HTTP

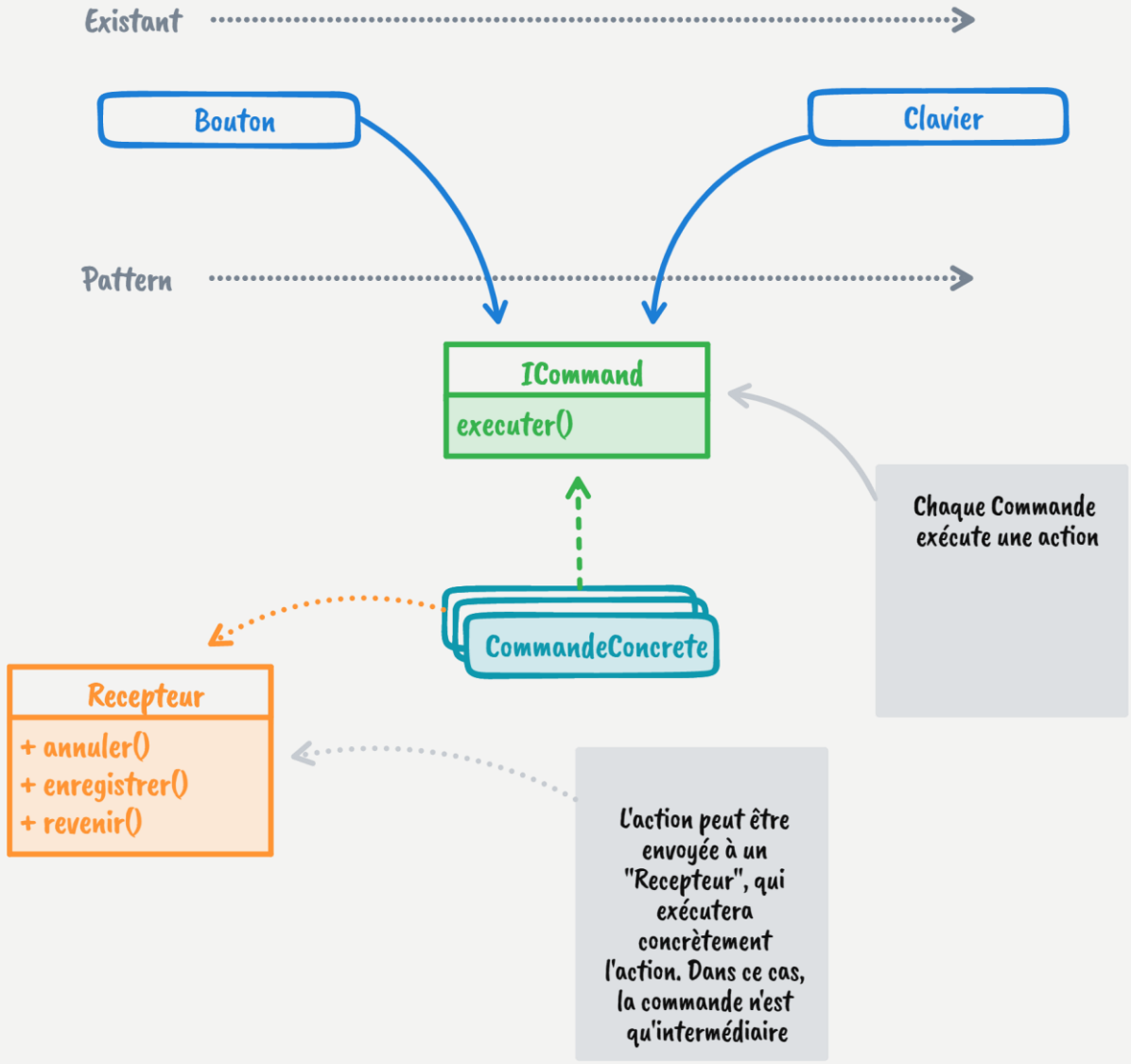
COMPORTEMENT – CHAIN OF RESP.



COMPORTEMENT – COMMAND

- Problèmes de conception
  - Code d'action (au clique sur bouton par exemple) dans le code du clique
- Principe
  - Principe de séparation des préoccupations
  - Interface commande implémentée par les commandes
  - Un invocateur (bouton par exemple) déclenche une commande
- Exemples d'utilisation
  - Boutons, raccourcis clavier et menu (invocateurs) qui aboutissent à la même fonctionnalité

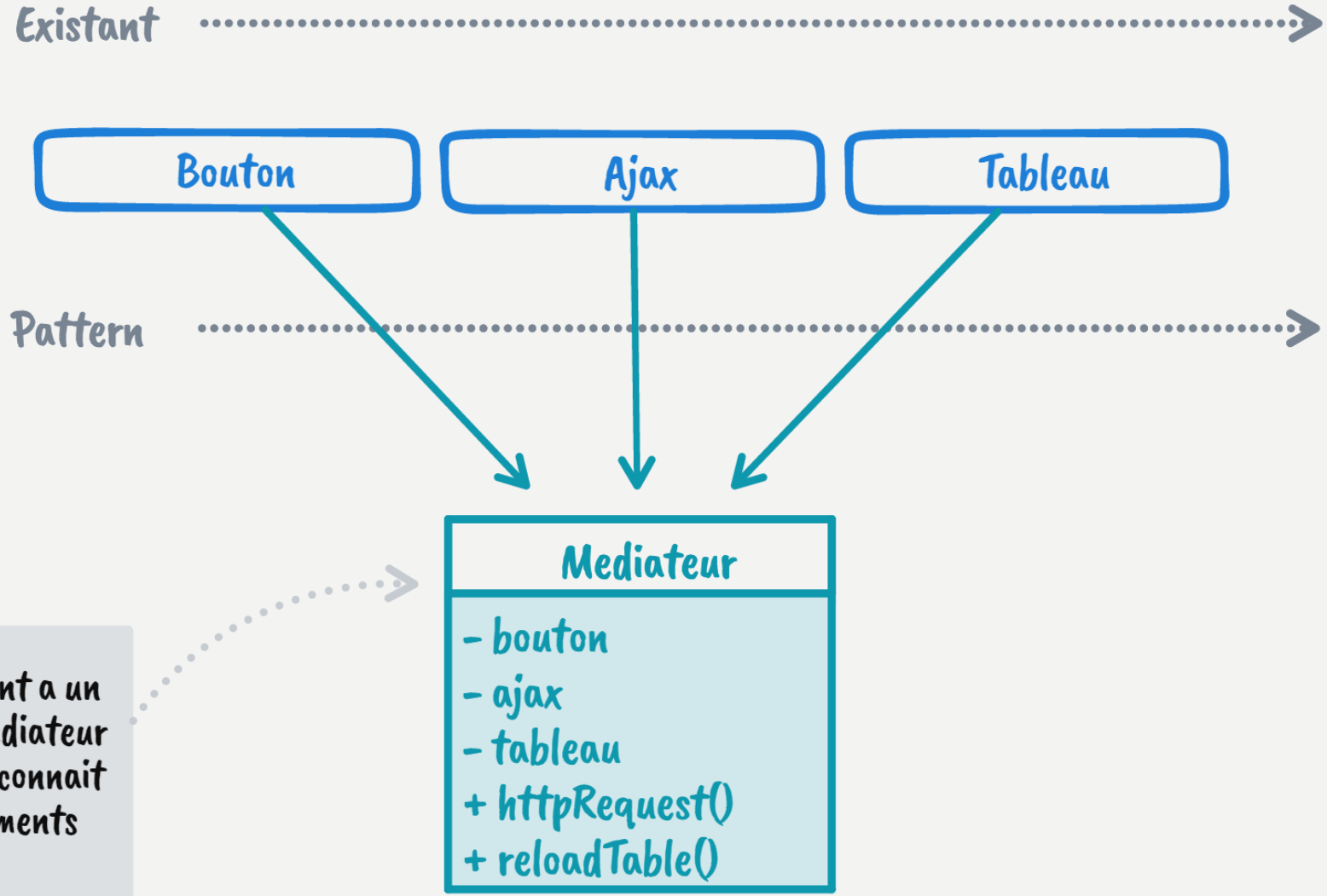
COMPORTEMENT – COMMAND



COMPORTEMENT – MEDIATOR

- Problème de conception
  - Les classes ont des dépendances fortes entre elles
- Principe
  - Plutôt que chaque classe soit dépendante de chaque autre classe
    - Mettre en place une classe dédiée à la gestion de la communication
  - Le médiateur est intermédiaire, il fonctionne comme un « routeur »
- Exemple d'utilisation
  - Un bouton, qui au clique, envoie une requête Ajax, qui elle-même change le contenu du tableau

COMPORTEMENT – MEDIATOR

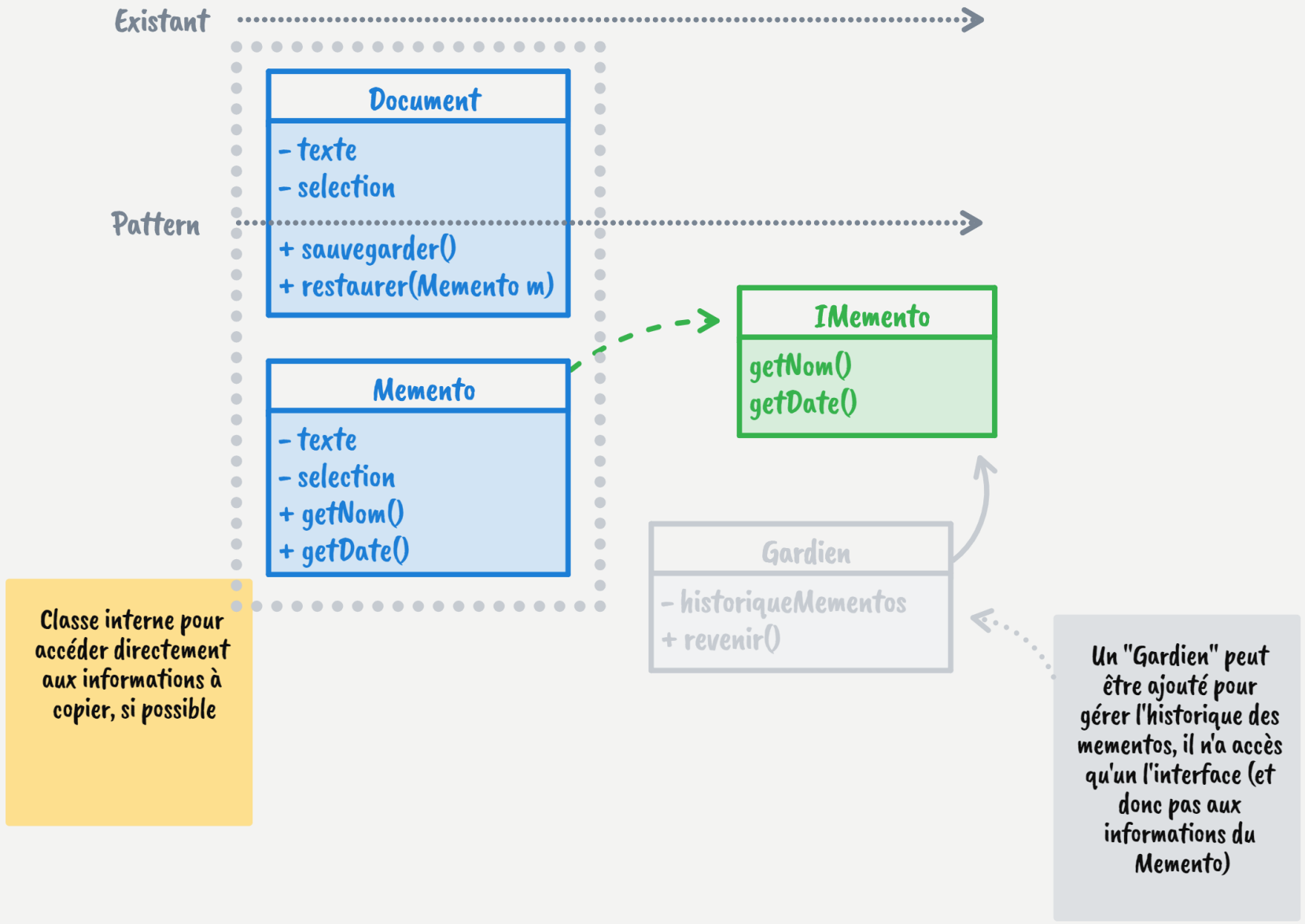


Chaque élément a un lien vers le Mediateur
Le Mediateur connait tous les éléments

COMPORTEMENT – MEMENTO

- Problème de conception
  - Sauvegarder l'état interne d'un objet en respectant l'encapsulation pour la restaurer plus tard
- Principe
  - Un Memento qui stock l'état d'une classe
  - Un gardien qui gère une liste de Memento (historique des états)
- Exemple d'utilisation
  - Fonctionnalités Undo / Redo

COMPORTEMENT – MEMENTO

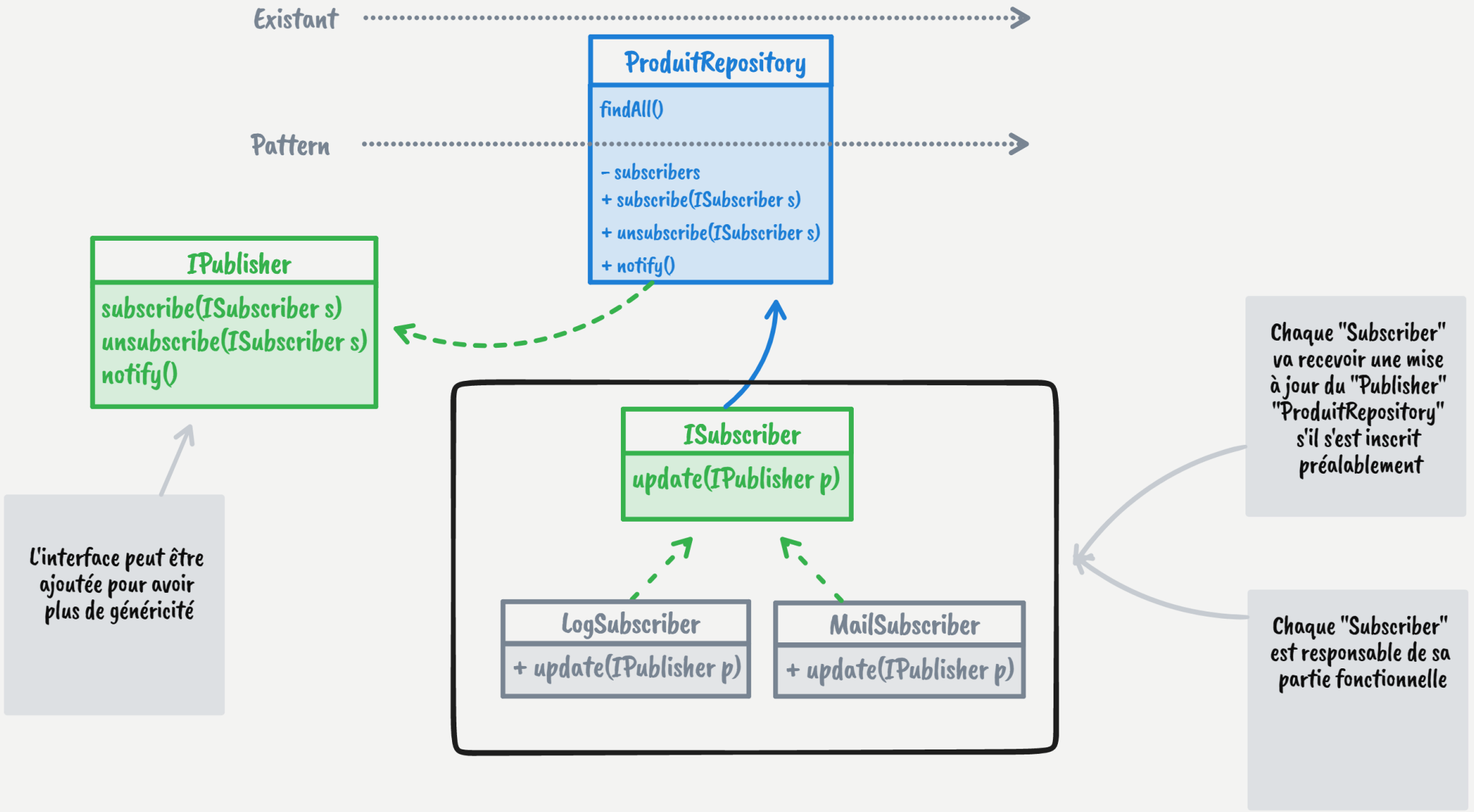


COMPORTEMENT – OBSERVER

- Problèmes de conception
  - Appeler une ou des méthode(s) après une action
  - Appel réalisé à chaque fois qu'on en a besoin, parce qu'on sait pas trop où placer ça

- Principe
  - Objet observé par d'autres objets
  - Méthodes
    - Attach
    - Detach
    - Notify
- Exemple d'utilisation
  - Envoyer une alerte mail et effectuer une journalisation après une action

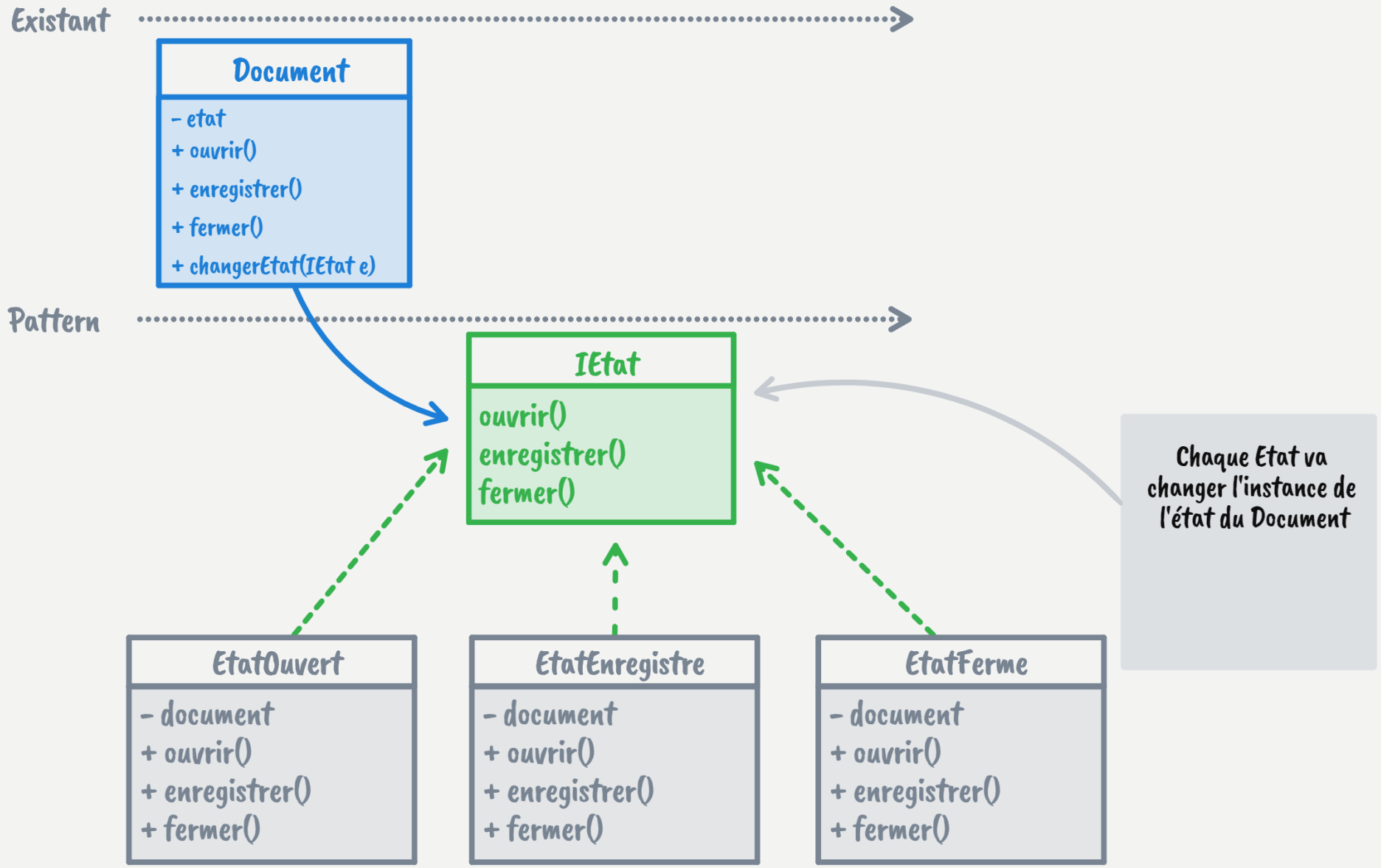
COMPORTEMENT – OBSERVER



COMPORTEMENT – STATE

- Problème de conception
  - Changer le comportement d'un objet selon son état interne
- Principe
  - Mettre en place une interface contenant une opération par état possible
  - Implémenter autant de classes que d'états possibles
- Exemple d'utilisation
  - Selon l'état d'un document (ouvert, modifié, sauvegardé et fermé), certaines méthodes sont bloquées

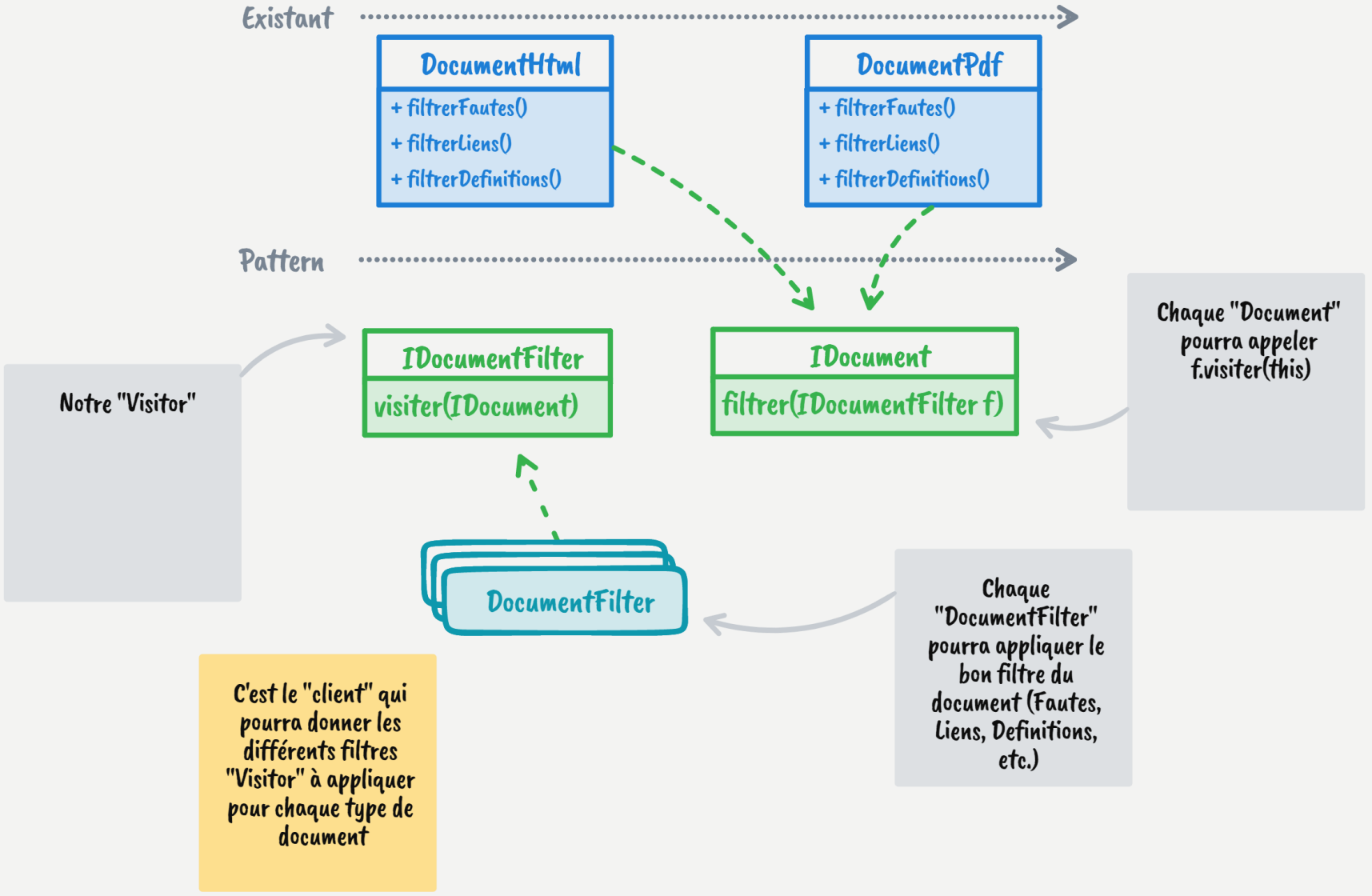
COMPORTEMENT – STATE



COMPORTEMENT – VISITOR

- Problème de conception
  - Réaliser des opérations sur les éléments d'un objet sans connaître à l'avance le résultat à obtenir
- Principe
  - Utiliser un objet tiers qui sera capable d'obtenir le résultat souhaité à partir des données
- Exemple d'utilisation
  - DocumentHTML susceptible de contenir des définitions, fautes d'orthographes, liens obsolètes, qu'il est nécessaire de traiter avant affichage

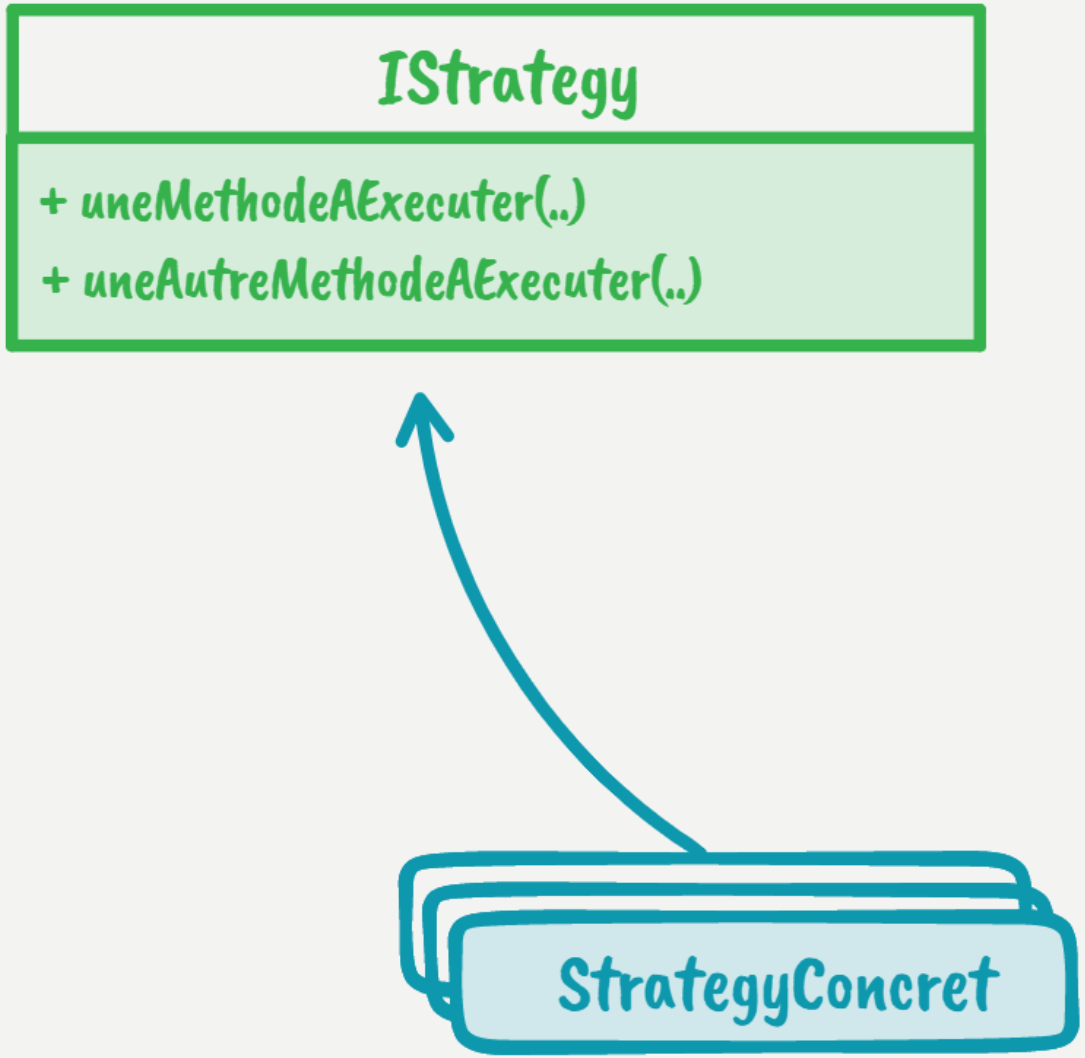
COMPORTEMENT – VISITOR



COMPORTEMENT – STRATEGY

- Problème de conception
  - Traiter une donnée en fonction du contexte, selon différents algorithmes possibles
  - Ces algorithmes ne sont connus qu'au dernier moment
- Principe
  - Une interface générale
  - Une classe par spécificité
  - Déléguer le traitement à un objet tiers qui implémente l'algorithme à utiliser
- Exemple d'utilisation
  - Contrôleur qui retourne un flux HTML, un flux JSON ou une erreur 404

COMPORTEMENT – STRATEGY



COMPORTEMENT – TEMPLATE METHOD

- Problème de conception
  - Fonctionnement global qui peut différer selon les cas (différents algorithmes)
- Principe
  - Une classe abstraite générale
  - Une classe dérivée par spécificité
- Exemple d'utilisation
  - Un fichier est ouvert, enregistré et fermé, mais il peut être enregistré au format HTML (texte) ou texte (suite d'octets).

COMPORTEMENT – TEMPLATE METHOD