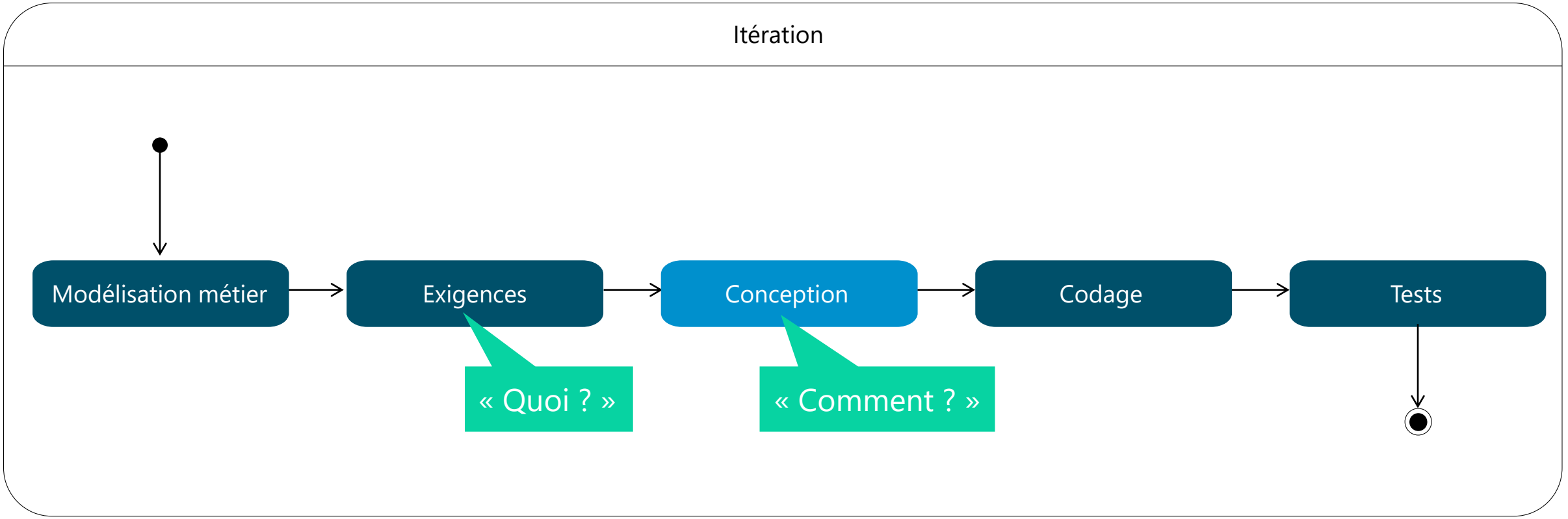
Analyse et conception

Module 05 – La conception

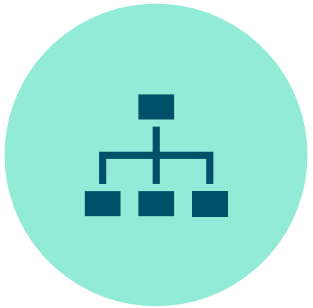
Objectifs

- Comprendre le concept de conception dans un projet informatique
- Notions essentielles d’architecture
- Notion de Design Pattern

La phase de conception



Objectifs de la phase de conception



ARCHITECTURE LOGICIELLE / CONCEPTION DE HAUT NIVEAU

PRINCIPES DE CONCEPTION / DESIGN PATTERNS

CONCEPTION DÉTAILLÉE : PASSER DES EXIGENCES AU CODE

Objectifs de la phase de conception

Qualité

- Faciliter l'écriture du code grâce à une architecture la plus claire et concise possible
- Garantir la robustesse du logiciel même en cas de refactoring

Evolutivité

- Permettre l'extension du logiciel en évitant les régressions

Maintenance

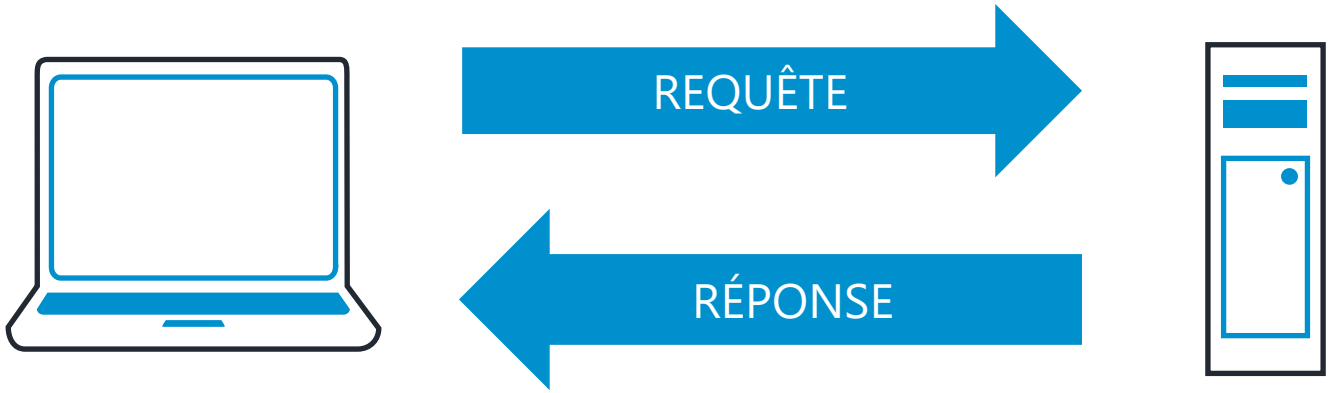
Réutilisabilité des composants

Architecture client-serveur

Architecture N-Tiers

SOA

Architecture client-serveur

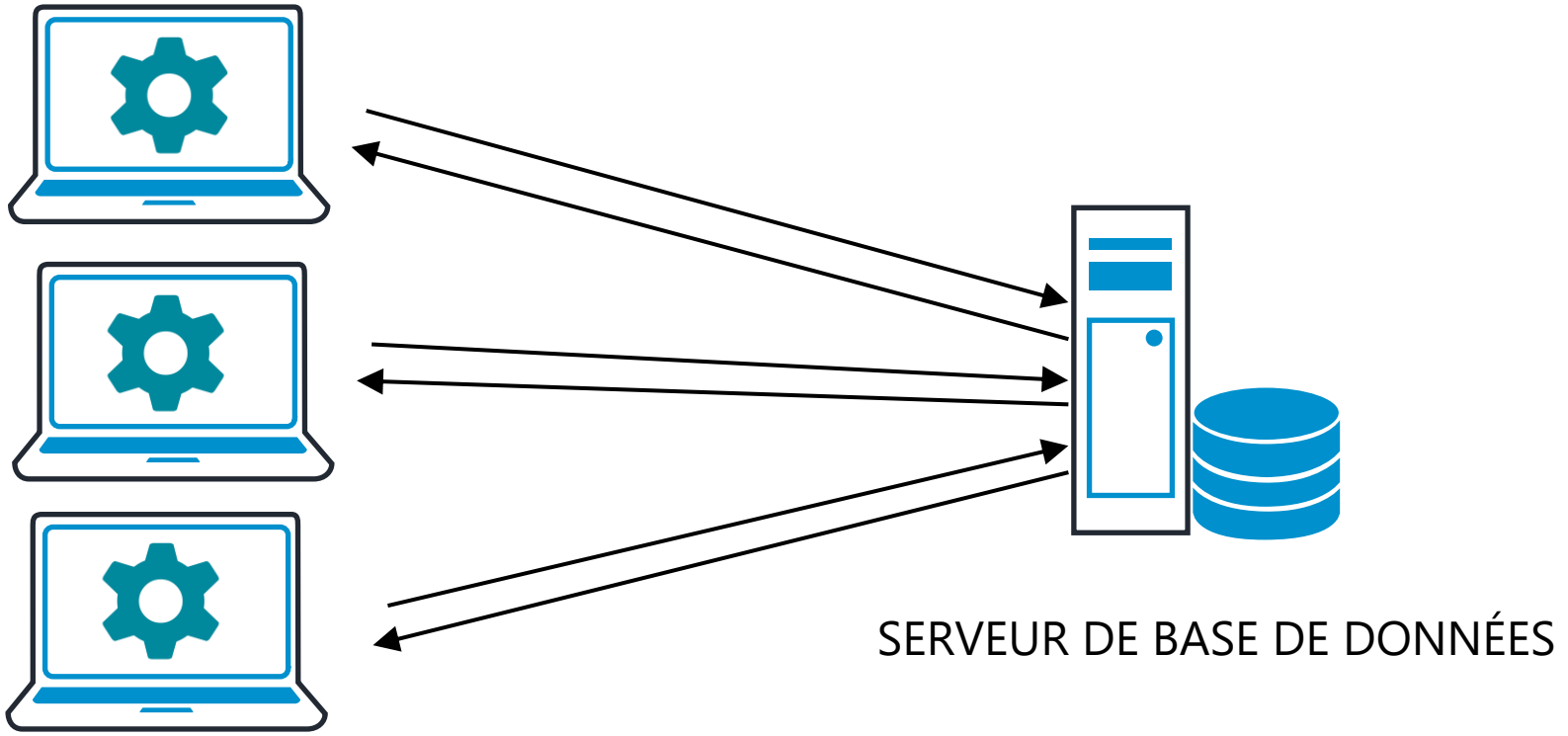


CLIENT

SERVEUR

Exemple :

l'architecture client lourd – serveur de base de données



CLIENT LOURD
(type Visual Basic, C, Pascal)

Architecture client-serveur

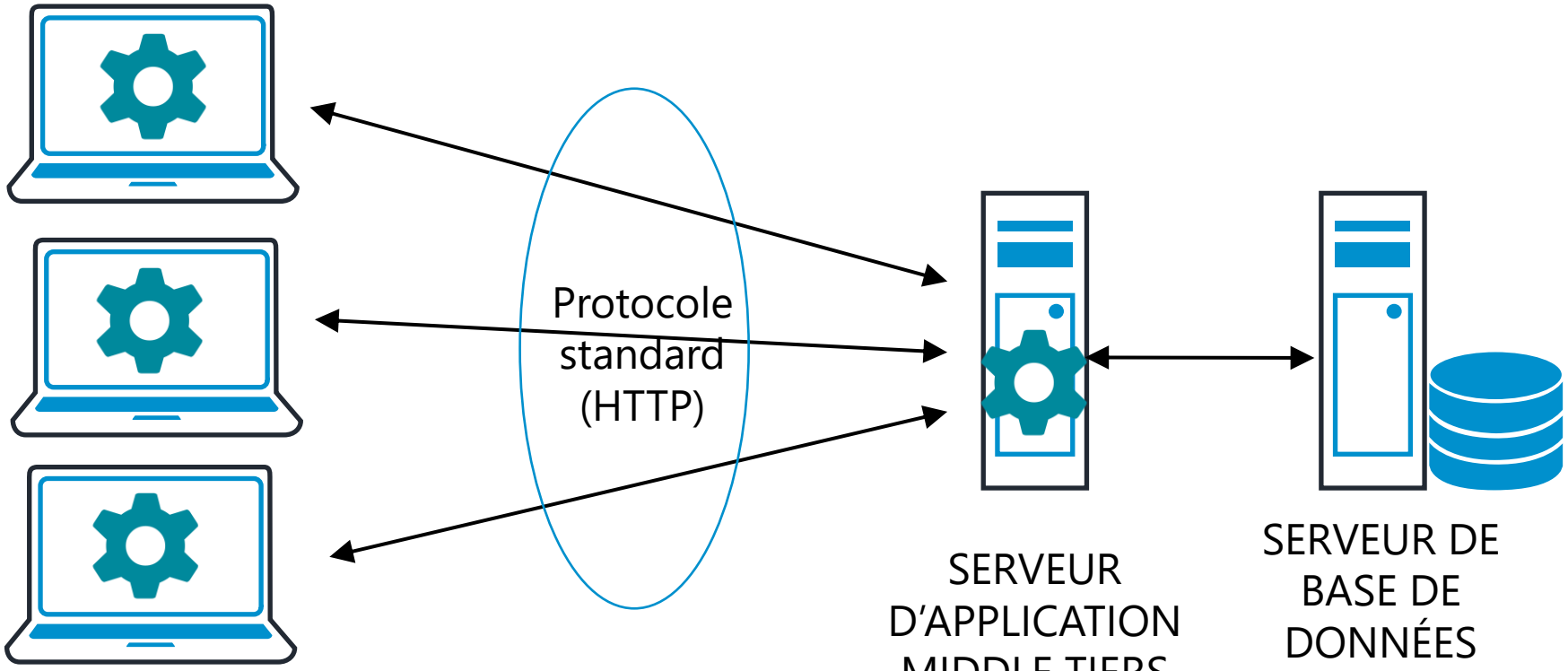
Avantages

- Les données sont centralisées permettant le contrôle de leur cohérence
- Plusieurs applications différentes peuvent communiquer via la base de données

Inconvénients

- Le serveur centralisé est un goulot d'étranglement limitant la montée en charge
- Nécessite une installation du client lourd
  - Lourdeur des déploiements
  - Problèmes éventuels de configuration des machines clientes
- Le client lourd peut devenir trop complexe
  - Couplage de domaines fonctionnels différents

Architecture 3 TIERS



Architecture 3 TIERS

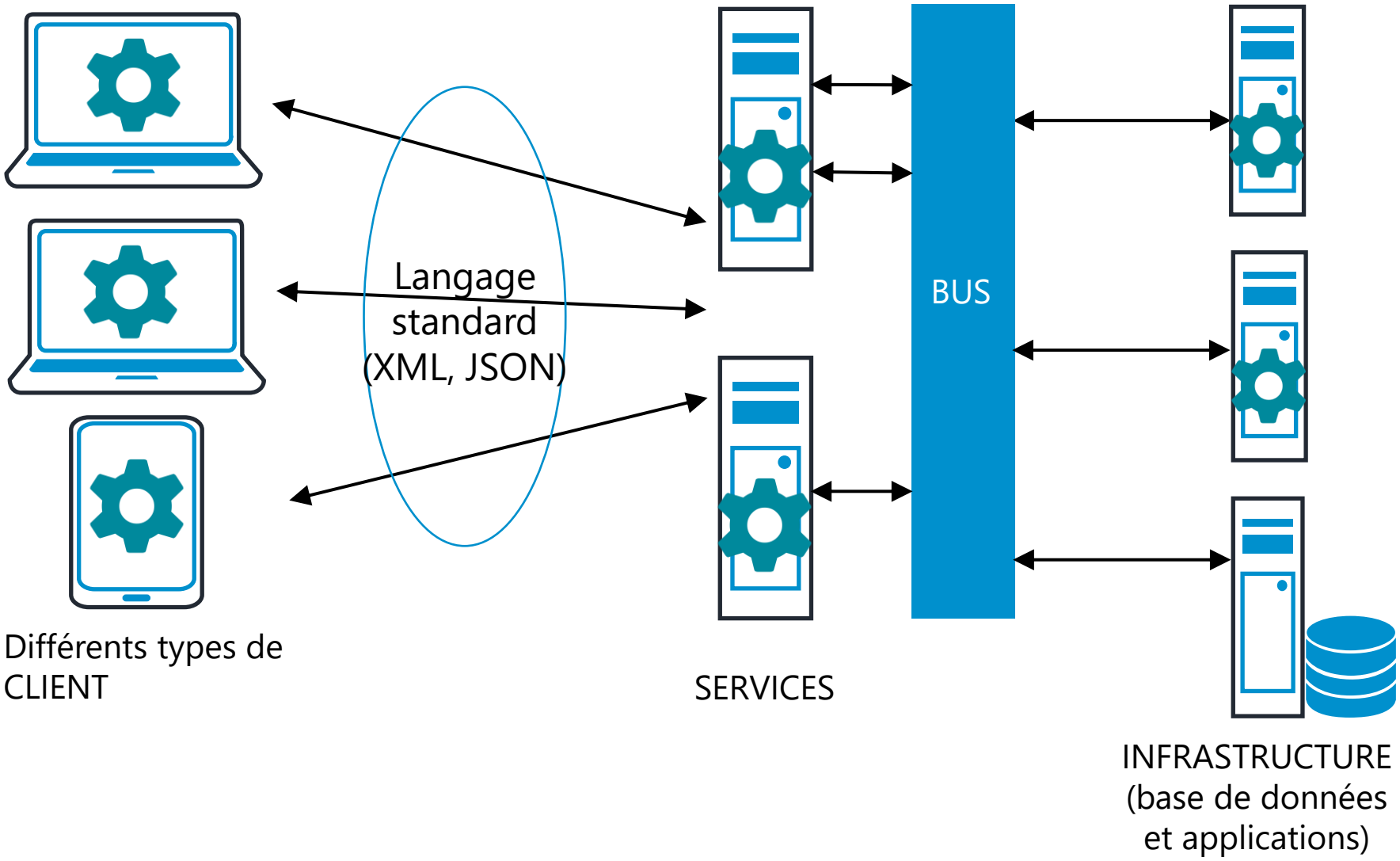
Avantages

- Les traitements sont centralisés sur le serveur
  - Déploiements facilités
  - Maintenance facilitée
- Plusieurs applications différentes peuvent communiquer via la base de données

Inconvénients

- Le serveur centralisé est un goulot d'étranglement limitant la montée en charge
- L'application peut couvrir différents domaines fonctionnels aboutissant à une application dite monolithique difficile à faire évoluer

Architecture SOA



Architecture SOA

- NOTION DE SERVICE
  - Fonction ayant une signification métier
  - Fonction ayant un objectif unique
  - Fonction mutualisée (potentiellement partagée par différentes applications)

- INTEROPÉRABILITÉ
  - Clients et services sont agnostiques de la technologie de l'autre partie
  - Couplage faible entre les composants

- IMPLÉMENTATIONS
  - Bus de service
    - Entreprise Service Bus
    - Message Broker (Ex. : Apache Kafka)
  - Web Oriented Architecture
    - Web services

- Objectifs de la phase de conception
- Architecture logicielle / Conception de haut niveau
- **Principes de conception / Design Patterns**
- Conception détaillée : passer des exigences au code

- Respect des principes SOLID :
  - S : Single Responsability Principle
  - O : Open / Closed Principle
  - L : Liskov Substitution Principle
  - I : Interface Segregation Principle
  - D : Dependency Inversion Principle

Single Responsibility Principle (SRP)

- Chaque objet est en charge d'une seule responsabilité, laquelle doit être encapsulée dans la classe
- Une responsabilité peut être définie comme une raison de changer
- Une classe ne doit être susceptible de changer que pour une raison

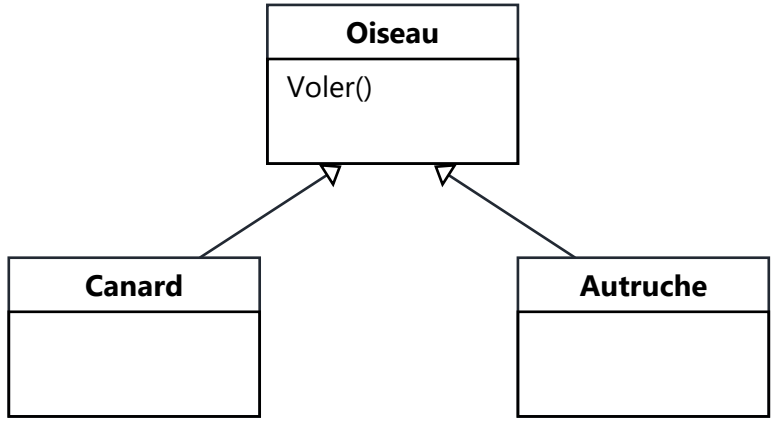
- Exemple :
  - Un composant logiciel qui crée et imprime un rapport. Ce composant peut être changé pour 2 raisons : soit le contenu du rapport change, soit le format de l'impression change. Le SRP préconise de séparer ces 2 responsabilités permettant de limiter les impacts du changement.

Open / Closed Principle

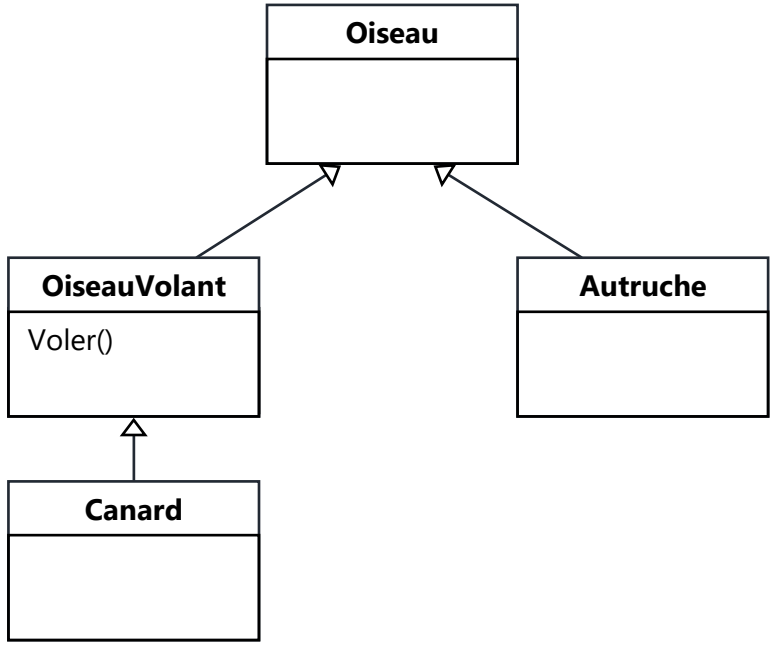
- Principe d'ouverture / fermeture
  - Les entités logicielles (classes, modules, fonctions, etc.) doivent être ouvertes aux extensions, mais fermées aux modifications
  - Version de B. Meyer :
    - Une classe ne peut être modifiée que pour corriger des erreurs
    - L'ajout de nouvelles fonctionnalités ne se fait que par la création de nouvelles classes (par exemple des sous-classes de la première)
  - Version polymorphique :
    - On utilise au maximum les interfaces figées. Mais elles peuvent être implantées librement et augmentées par héritage

- Principe de substitution de Liskov
  - Les classes dérivées doivent pouvoir se substituer à leurs classes parentes

Exemple ne respectant pas le principe de Liskov



Version corrigée



Interface Segregation Principle

- Principe de ségrégation des interfaces
  - Une interface doit ne comporter que des méthodes en rapport avec l'interface elle-même, de façon à ce que les clients d'une interface ne connaissent que les méthodes en rapport avec cette interface
  - Aucun client d'une interface ne doit dépendre de méthodes (de l'interface) qu'il n'utilise pas

Dependency Inversion Principle

- Principe d'inversion des dépendances
  - Dans les architectures classiques, les composants de haut niveau dépendent des composants de bas niveau sur lesquels ils reposent. Le principe d'inversion des dépendances établit au contraire que :
    - Les modules de haut niveau ne doivent pas dépendre des modules de bas niveau. Les deux doivent dépendre d'abstractions
    - Les abstractions ne doivent pas dépendre de détails, mais les détails doivent dépendre des abstractions

Les patrons de conception (Design Pattern)

- Les patterns répondent à des problèmes de conception récurrents dans le cadre de la programmation objet.
- Ils représentent des solutions connues et éprouvées dont la conception provient de l'expérience des programmeurs.
- Ils sont basés sur les bonnes pratiques de la programmation objet.
- 23 patterns décrits dans l'ouvrage de référence du « GoF » répartis dans 3 catégories : construction, structuration et comportement.
- Chaque pattern est nommé, décrit, représenté sous la forme d'un diagramme de classe.
- Quelques Design Patterns :
  - Singleton
  - Observer
  - State
  - Factory
  - MVC

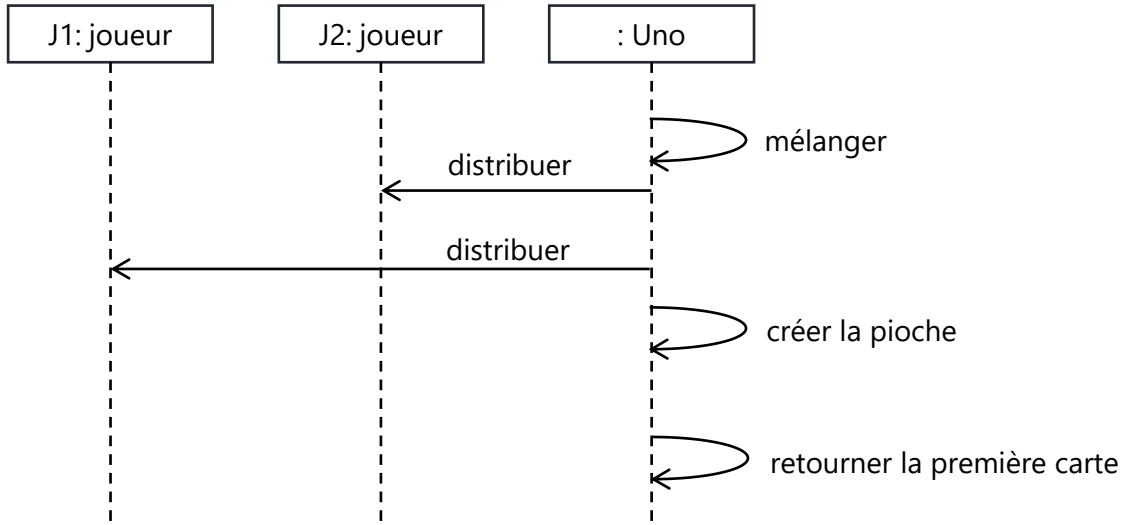
- Définition : les fausses bonnes idées !
- Exemples :
  - Le copier-coller
  - Le code spaghetti
  - Le blob : classe ayant trop de responsabilités
  - …

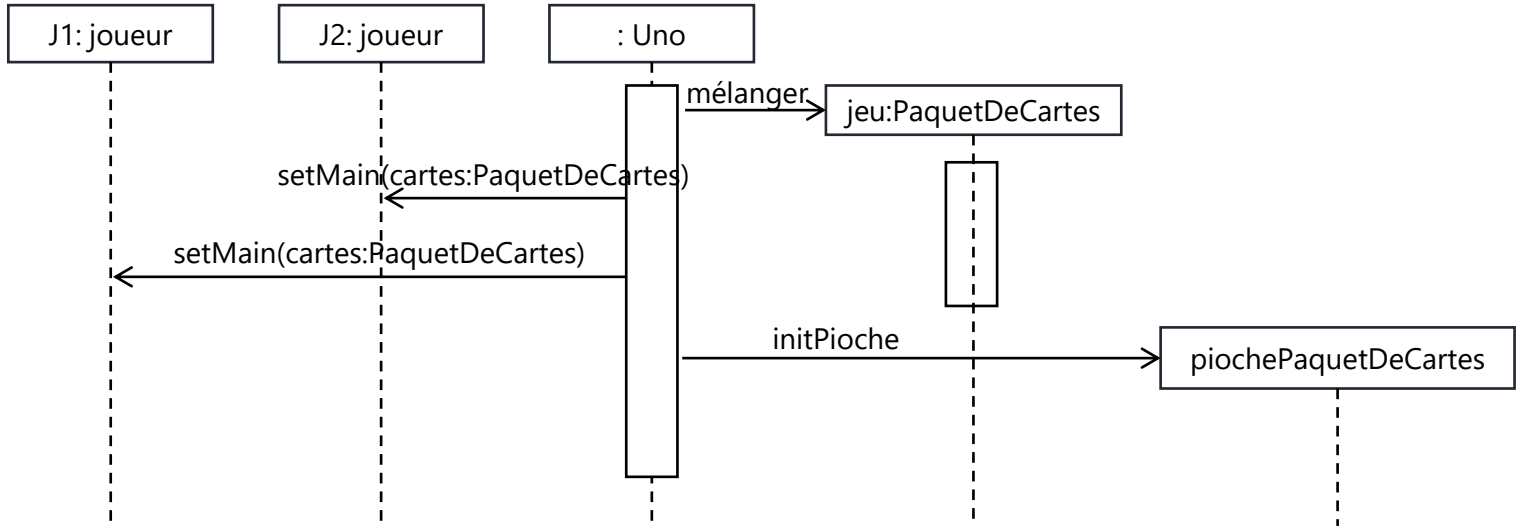
  (cf. https://fr.wikipedia.org/wiki/Antipattern)

- Objectifs de la phase de conception
- Architecture logicielle / conception de haut niveau
- Principes de conception / Design Patterns
- **Conception détaillée : passer des exigences au code**

Découvrir les objets avec le diagramme de séquence

Diagramme de séquence système
DSS-01 – initializer la partie de UNO





Découvrir les objets avec le diagramme d'état