MICROSERVICES

JÉRÉMY PERROUAULT

INTRODUCTION

INTRODUCTION

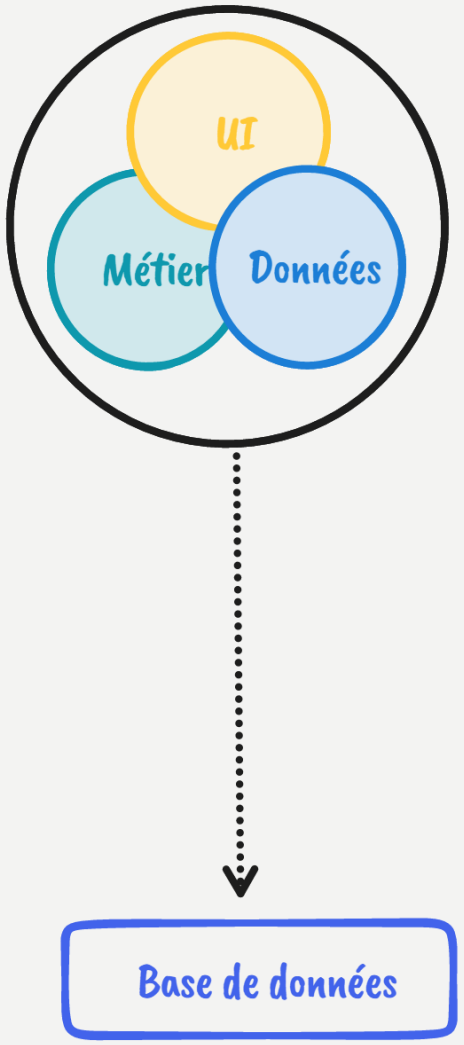
INTRODUCTION

- Les « microservices »
  - Architecture(s)
  - Approche de développement
- Décompose l'application en éléments plus simples et « indépendants »
  - Certains services ont besoin d'autres services : besoin de communication entre eux
- On parlera des « microservices backend »
  - Pour le front, on parle de « micro-frontend », sujet qui ne sera pas abordé ici

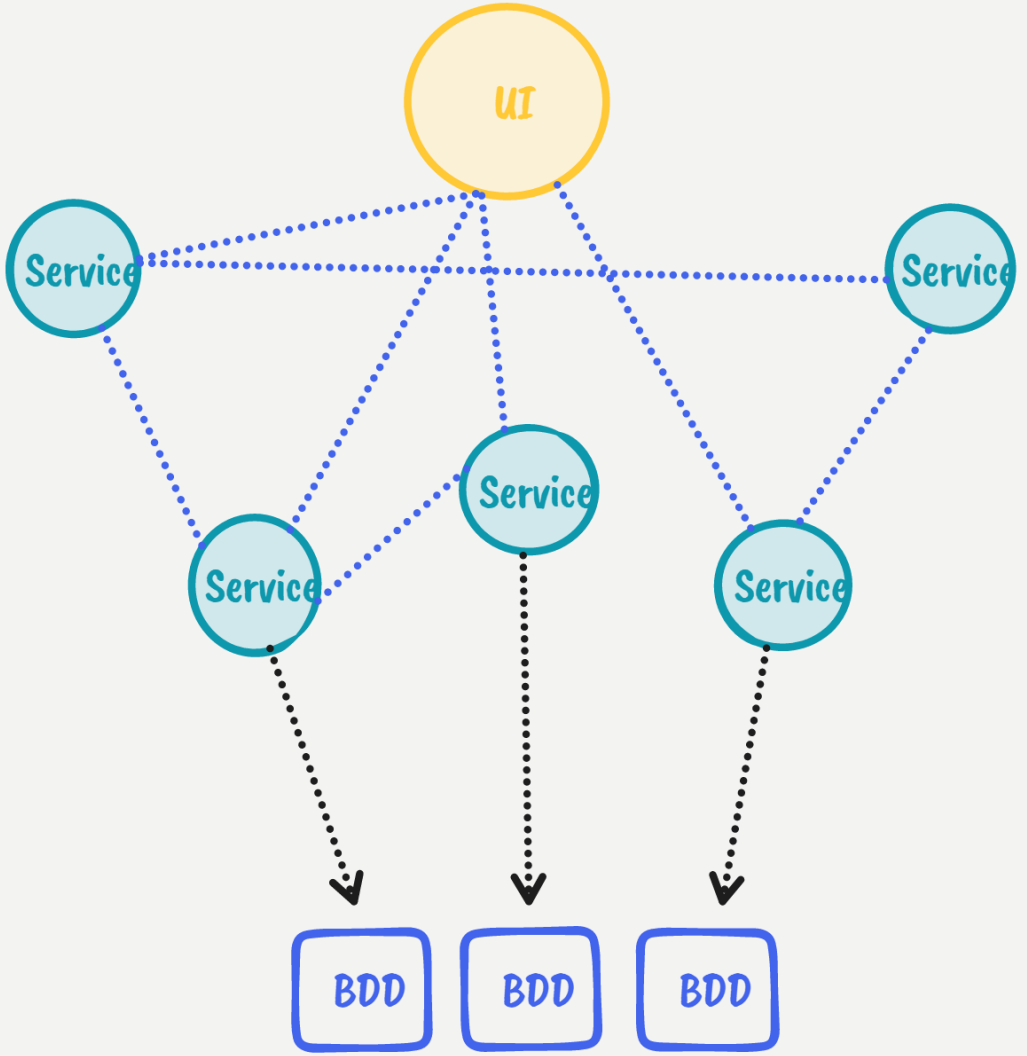
INTRODUCTION

NOTE : Ceci est un exemple.
Il existe une multitude d'architectures en microservices

Architecture Monolithique



Architecture Microservices



INTRODUCTION

- Fonctionnent principalement par **API REST**
  - Chaque service expose des fonctionnalités
  - Chaque service peut consommer une fonctionnalité exposée par un autre service
  - Chaque service peut agréger des informations d'autres services
  - Etc.
- En gardant le même vocabulaire **HTTP REST**
  - GET Récupérer des informations
  - POST Ajouter des informations
  - PUT Modifier des informations
  - PATCH Modifier partiellement des informations
  - DELETE Supprimer des informations

INTRODUCTION

- Les architectures en « microservices » sont généralement plus complexes et plus demandeuses en ressources
  - Patterns
    - Pour nous aider à faire communiquer les microservices entre eux
  - Technologies
    - Ensemble de librairie et frameworks pour nous aider à résoudre les problèmes les plus communs (souvent des implémentations des Patterns)

INTRODUCTION

- Mais présente un réel avantage
  - Séparation des briques fonctionnelles (même si, en développement plus classique, le découpage en modules est possible)
  - Séparation des équipes
  - Evolutivité / maintenabilité
  - Adaptation des ressources
  - Gestion des erreurs

ARCHITECTURE

ARCHITECTURE IDÉALE

ARCHITECTURE

- En microservices, on peut implémenter différents styles d'architectures
  - **MAIS**, et malgré des dépendances inévitables entre services
  - On doit tendre vers des services indépendants les uns des autres (couplage faible)
    - Pour limiter les effets de bord
      - En cas d'évolution, ou de bugs
    - Pour mieux gérer le développement
    - Pour mieux gérer le déploiement
    - Pour mieux gérer la scalabilité
    - Pour mieux gérer les erreurs
  - On doit avoir des services résilients

ARCHITECTURE

- Ce qui veut dire … qu'on refera une classe, pour désigner un même domaine, dans différents services …

```
@Getter @Setter
@ToString
public class Produit {
    private int id;
    private String nom;
    private Double prix;
    private int note;
}
```

ARCHITECTURE

- … en langages différents, ou identiques

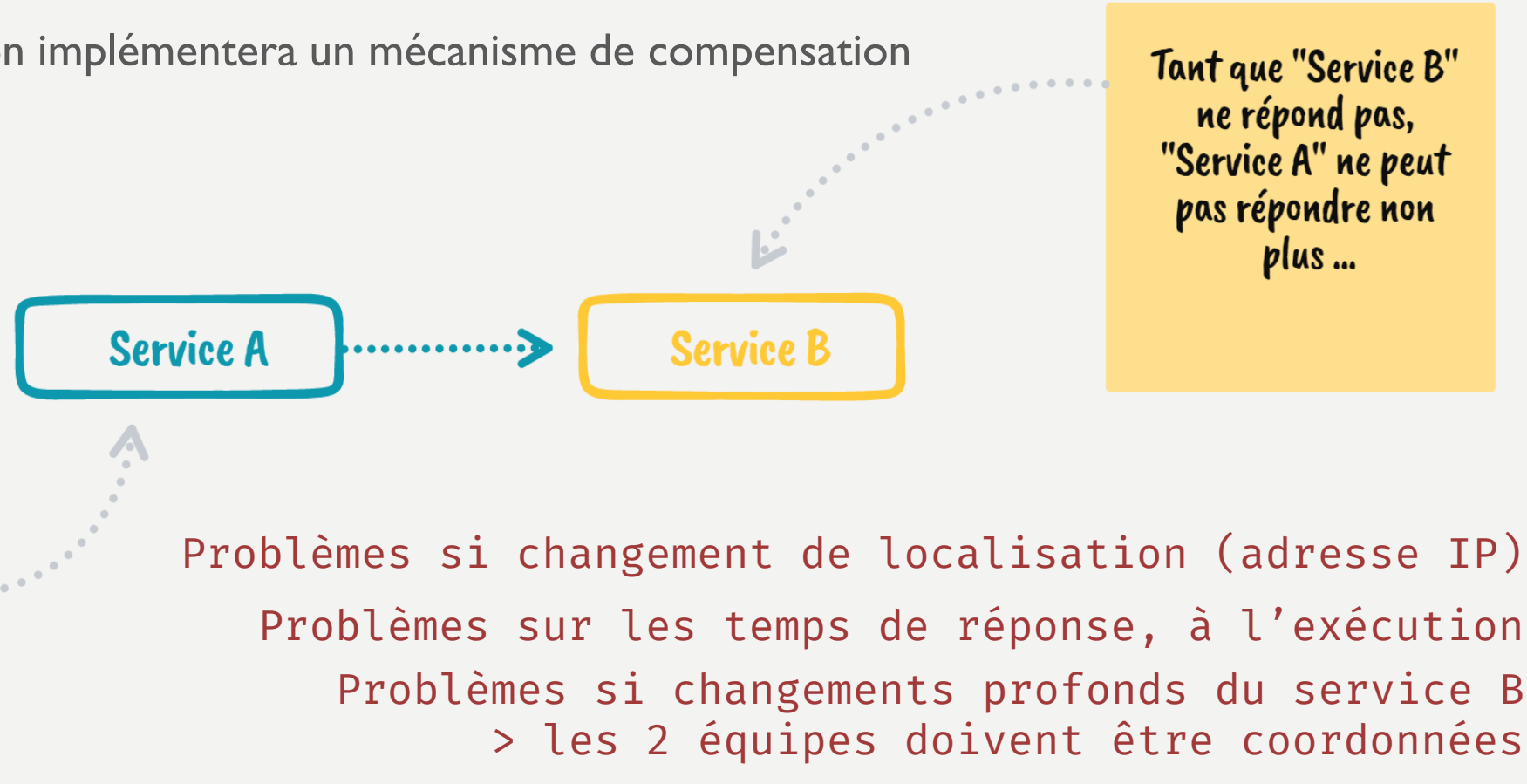
```
public class Produit
{
    public int Id { get; set; }
    public string? Nom { get; set; }
    public double? Prix { get; set; }
    public int Note { get; set; }
}
```

ARCHITECTURE

- Et le principe DRY ? Tenté d'utiliser une bibliothèque partagée ?
  - Vous vous retrouverez avec un couplage fort
    - Une modification, (ex : ajout d'une nouvelle information) aura des répercutions dans tous les services
    - Et occasionnera éventuellement des défaillances

ARCHITECTURE

- Ce qui veut dire aussi … qu'on évitera ce genre de situation …
  - Où un service dépend d'un autre en l'appelant directement
- Ou du moins, qu'on implémentera un mécanisme de compensation



Problèmes si changement de localisation (adresse IP)

Problèmes sur les temps de réponse, à l'exécution

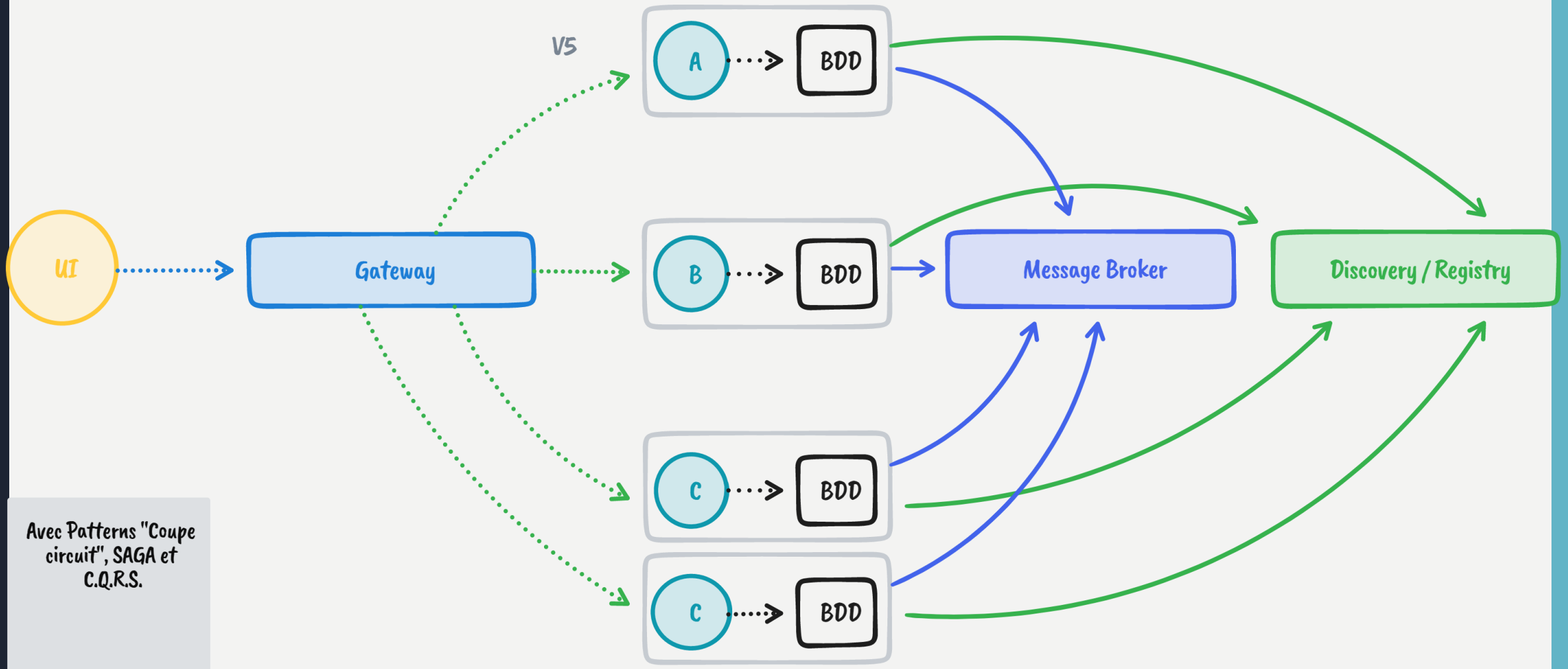
Problèmes si changements profonds du service B
> les 2 équipes doivent être coordonnées

ARCHITECTURE

- Une base de données, ou plusieurs bases de données ?
  - Un partage des ressources peut limiter les performances d'un ou des services
  - Une modification dans la structure des données peut impacter les services qui en consomme une partie (idem classes partagées)
- DONC, on privilégiera d'utiliser une base de données par service
  - En ayant conscience de perdre l'avantage des transactions et des contraintes référentielles
    - Qu'il faudra compenser par d'autres mécanismes

ARCHITECTURE

- Pour finalement tendre vers cette architecture



Avec Patterns "Coupe circuit", SAGA et C.Q.R.S.

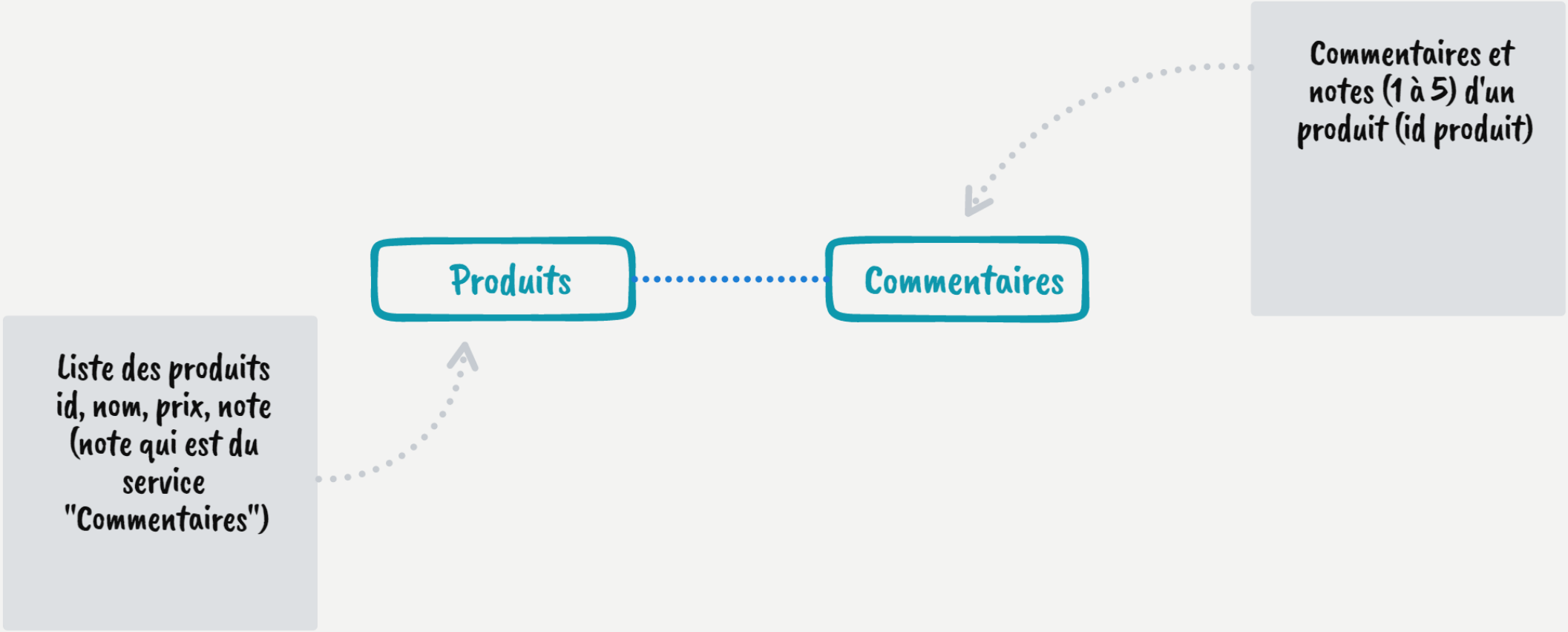
APPLICATION

DEUX PREMIERS SERVICES

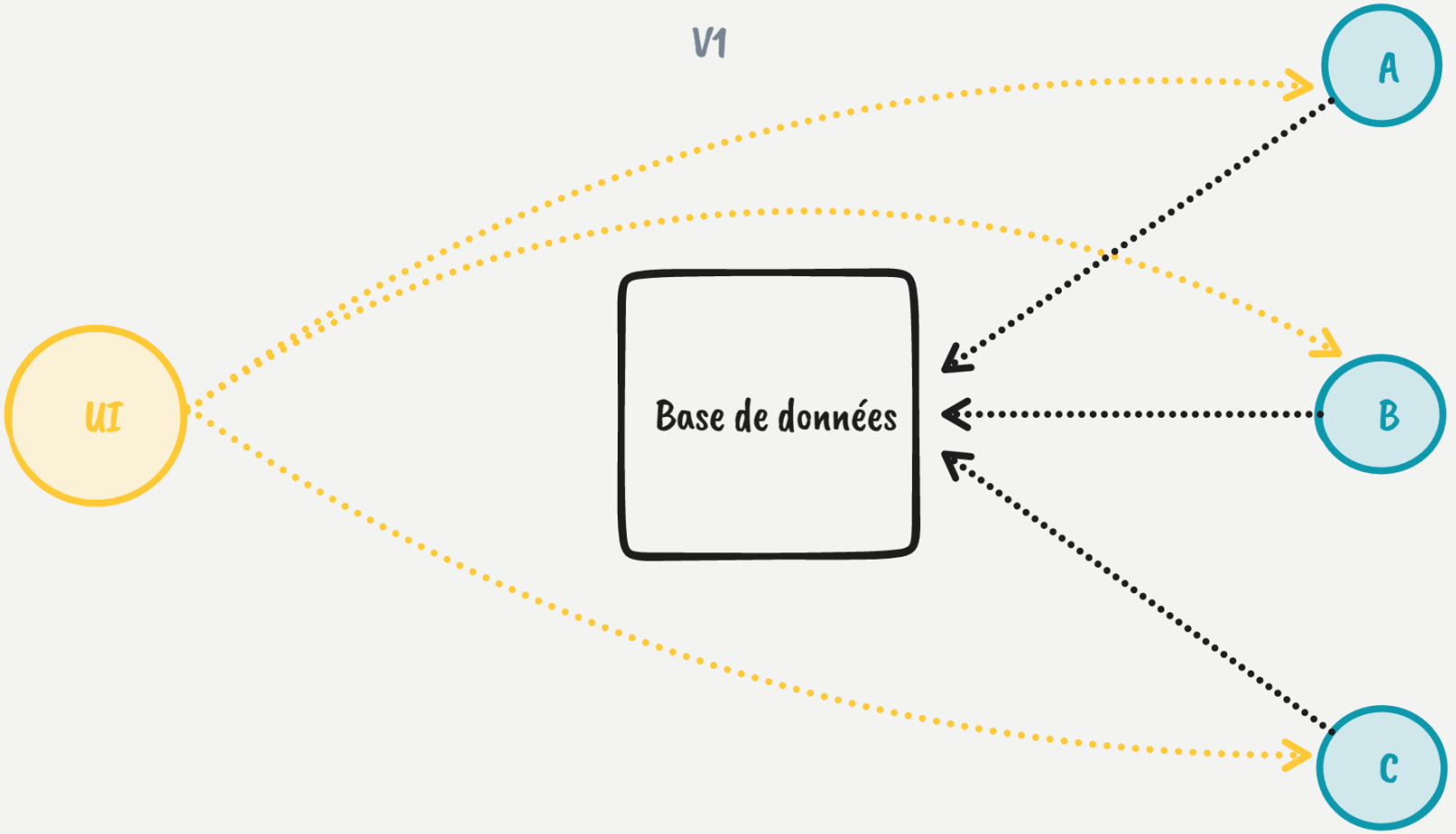
APPLICATION

- Fabrication d'une application e-commerce
  - Des produits
  - Des commentaires
  - Des clients
  - Des commandes
  - Des paiements
  - Des expéditions

APPLICATION



APPLICATION



APPLICATION

- Modèle de données
  - Produit : id, nom, prix, notable
  - Commentaire : id, texte, note (0 à 5), produit

- Règle de gestion
  - Un commentaire ne peut exister QUE si le produit est « notable »
  - Un produit ne peut pas être supprimé si des commentaires existent pour lui

APPLICATION

- Service Produit
  - Récupérer la liste des produits, avec la note moyenne de chacun produit
  - Récupérer un produit, avec sa note moyenne et ses commentaires
  - Ajouter un produit
  - Modifier un produit
  - Supprimer un produit

- Service Commentaire
  - Récupérer un commentaire, avec le nom du produit
  - Ajouter un commentaire pour un produit
  - Modifier un commentaire
  - Supprimer un commentaire

APPLICATION

- Mise en place de l'architecture précédente
  - En JAVA SPRING BOOT Service "Produits"
  - En C# ASP .NET Service "Commentaires"

- Implémenter **Swagger** sur les 2 projets
- Désactiver le HTTPS si activé
- Utiliser une base de données embarquée, ou PostgreSQL, ou MySQL, ou …