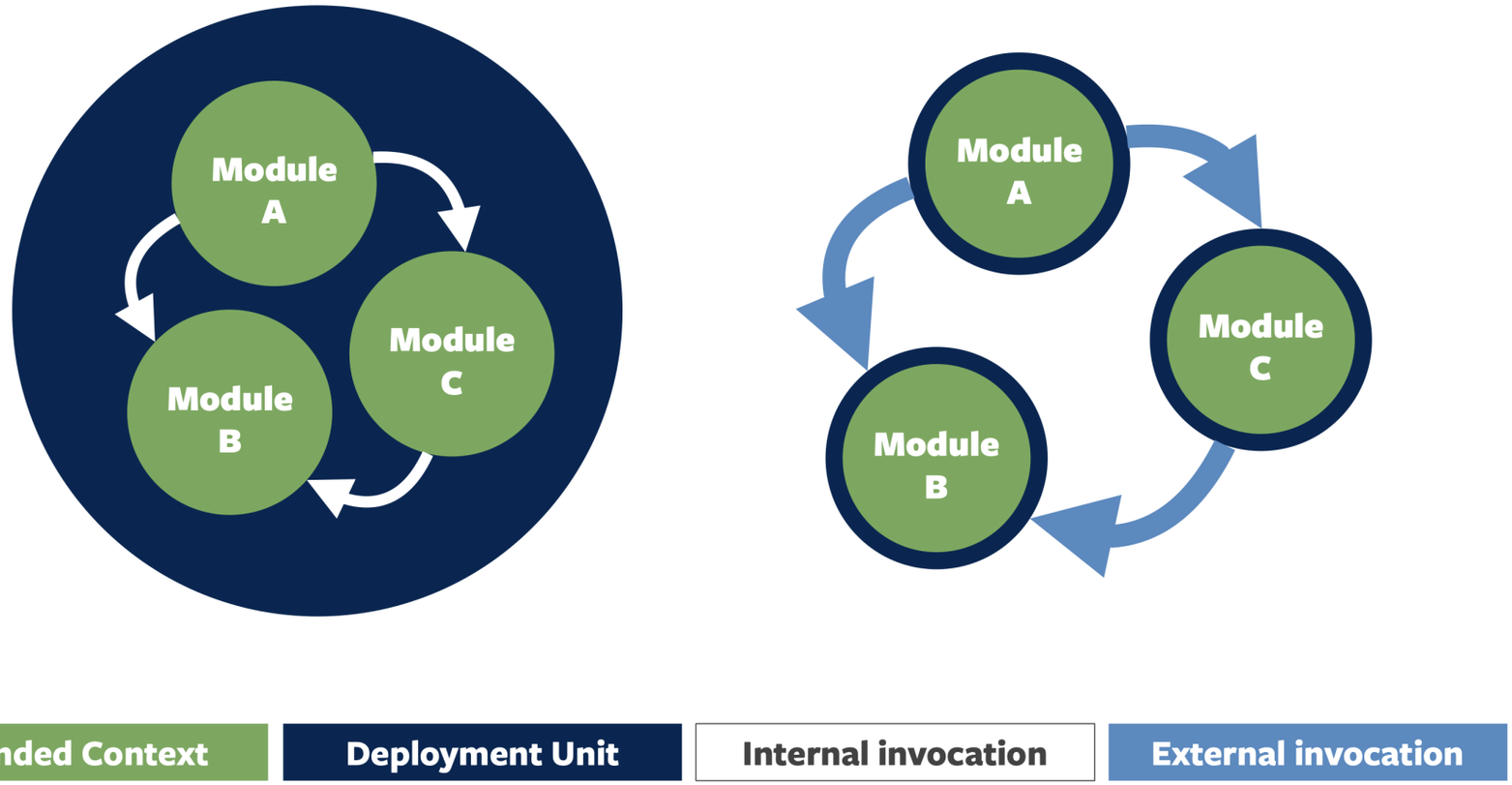
Développement de services Web avancés – partie 2

Backend basé sur une architecture microservices

Principes

- **Décomposer** en plusieurs services, *simples* et *indépendants*, organisés par **fonctionnalités métier**

Monolith VS. Microservice

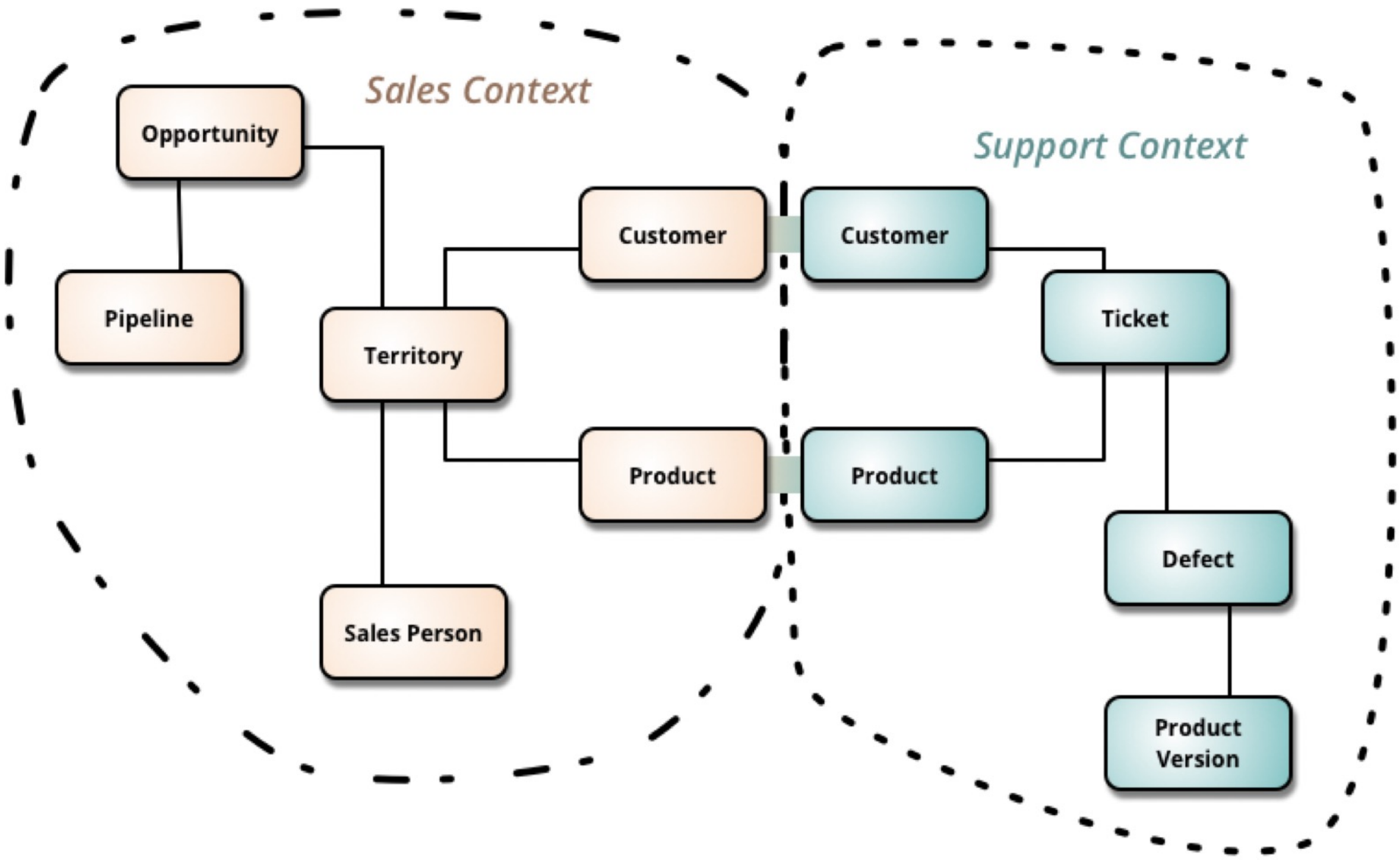


Oliver Drotbohm

Principes (cont.)

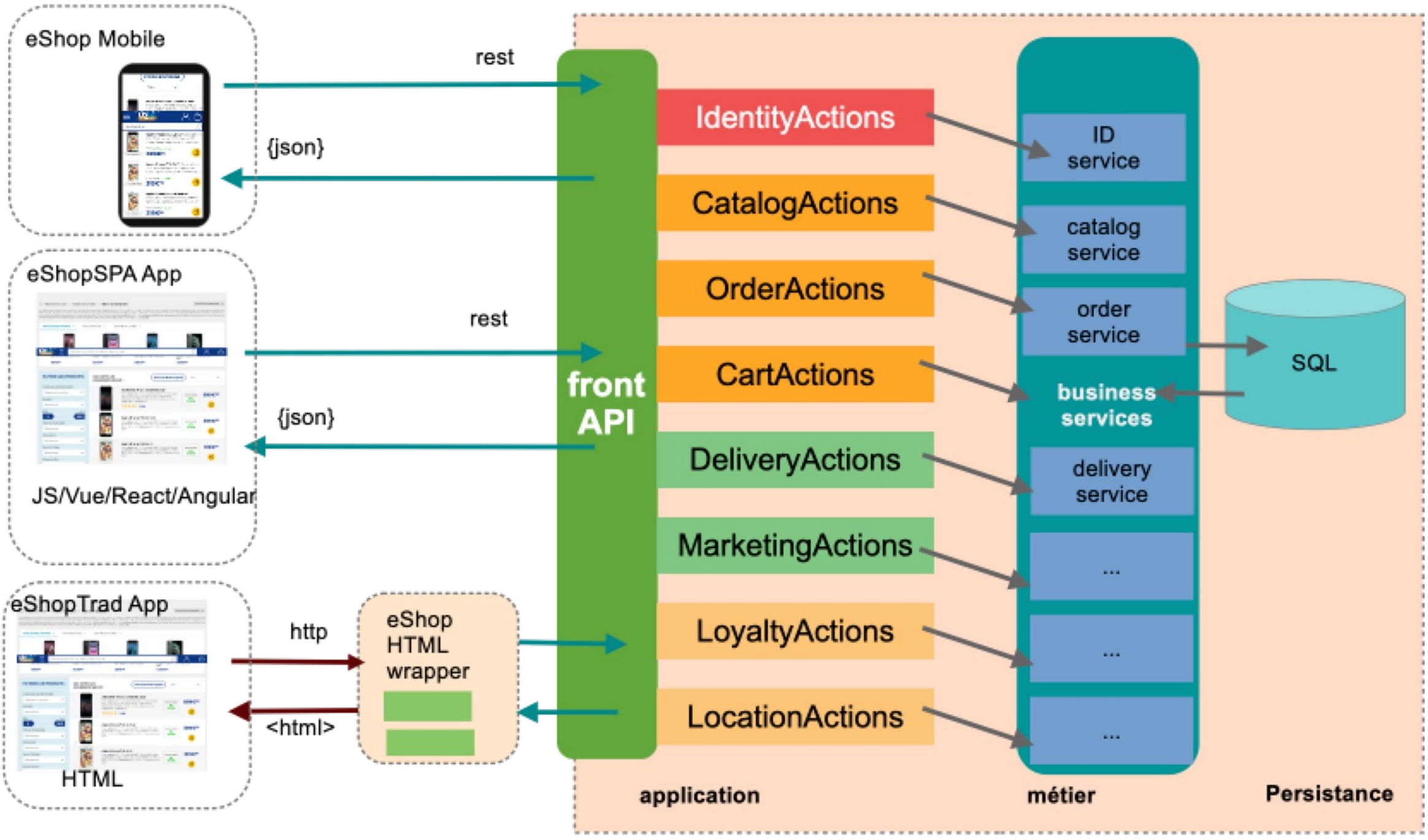
- Modularisation (*microservices*) des fonctionnalités métier (au sens *bounded context*)
- Développés, testés, déployés et mis à l'échelle de façon indépendante (équipes, technologies)
- Fonctionnalités exposées via une API
- Cohésion forte/couplage faible

Bounded contexts

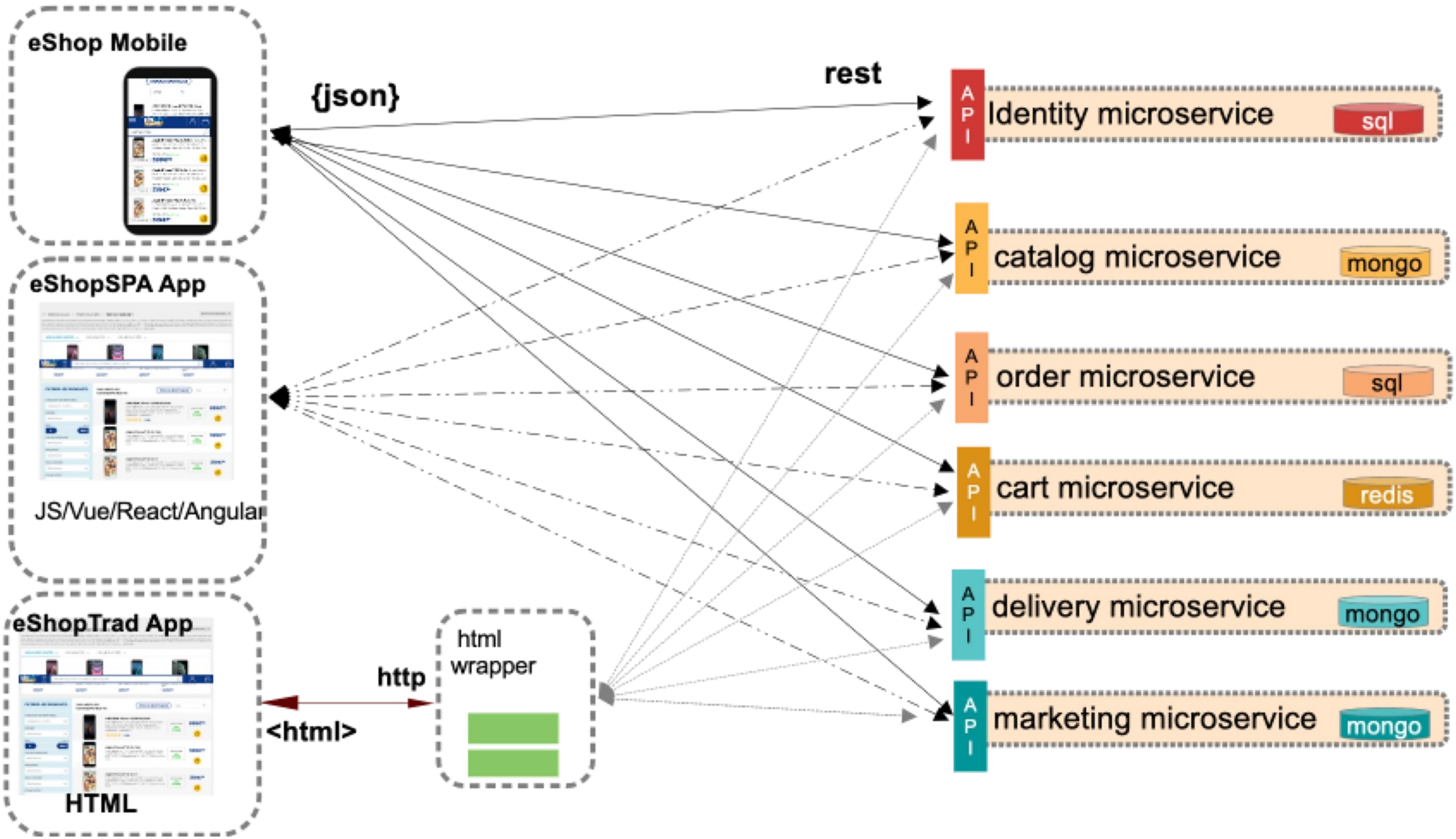


Sam Newman

Architecture SPA monolithique



Architecture microservices



Each has its advantages

Monolith:

- Infrastructure simplicity → Up and running faster
- Code simplicity → don't need to worry about latency, graceful failure, interaction level monitoring, etc.
- Architecture & Org simplicity → no hard boundaries, shared tech stack

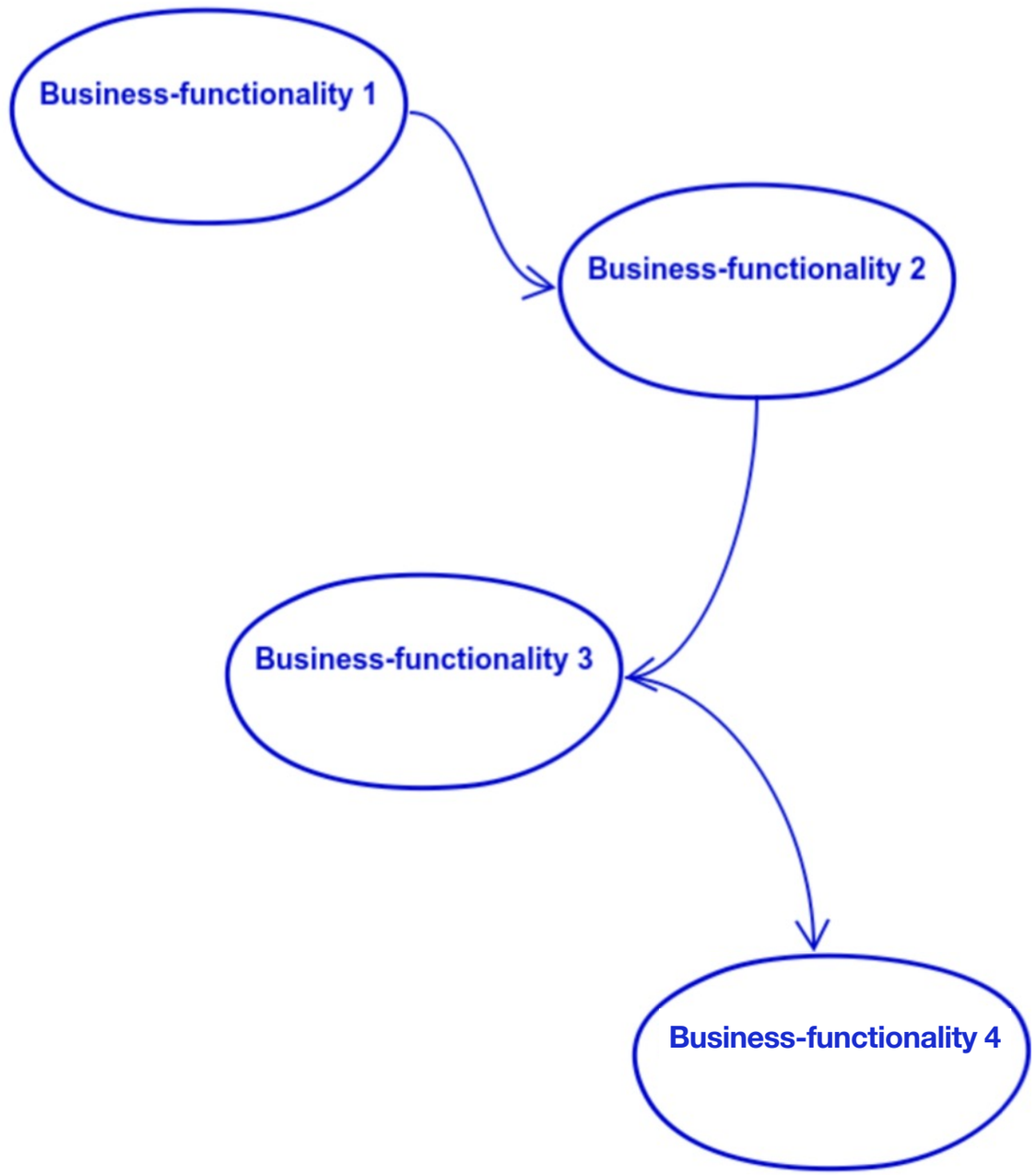
Microservices:

- System ownership → decoupled tech stack (API contracts)
- Smaller components → easier to read code (learn)
- Separation of concerns → fault isolation (easier to troubleshoot)
- Scaling separately → add more to services that are bottlenecks

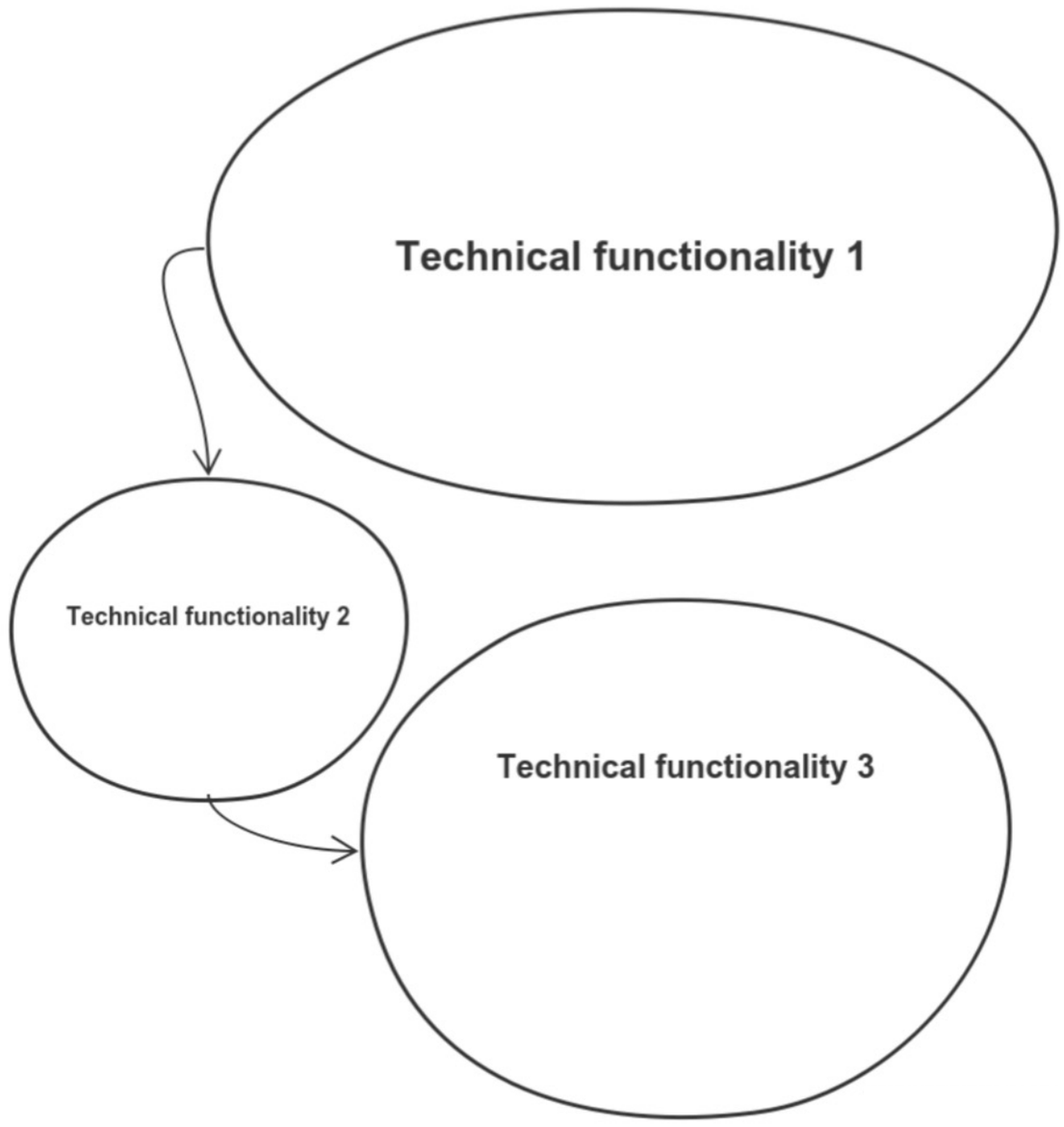
GitHub's Journey from Monolith to Microservices – 2021

Architecture microservices (cont.)

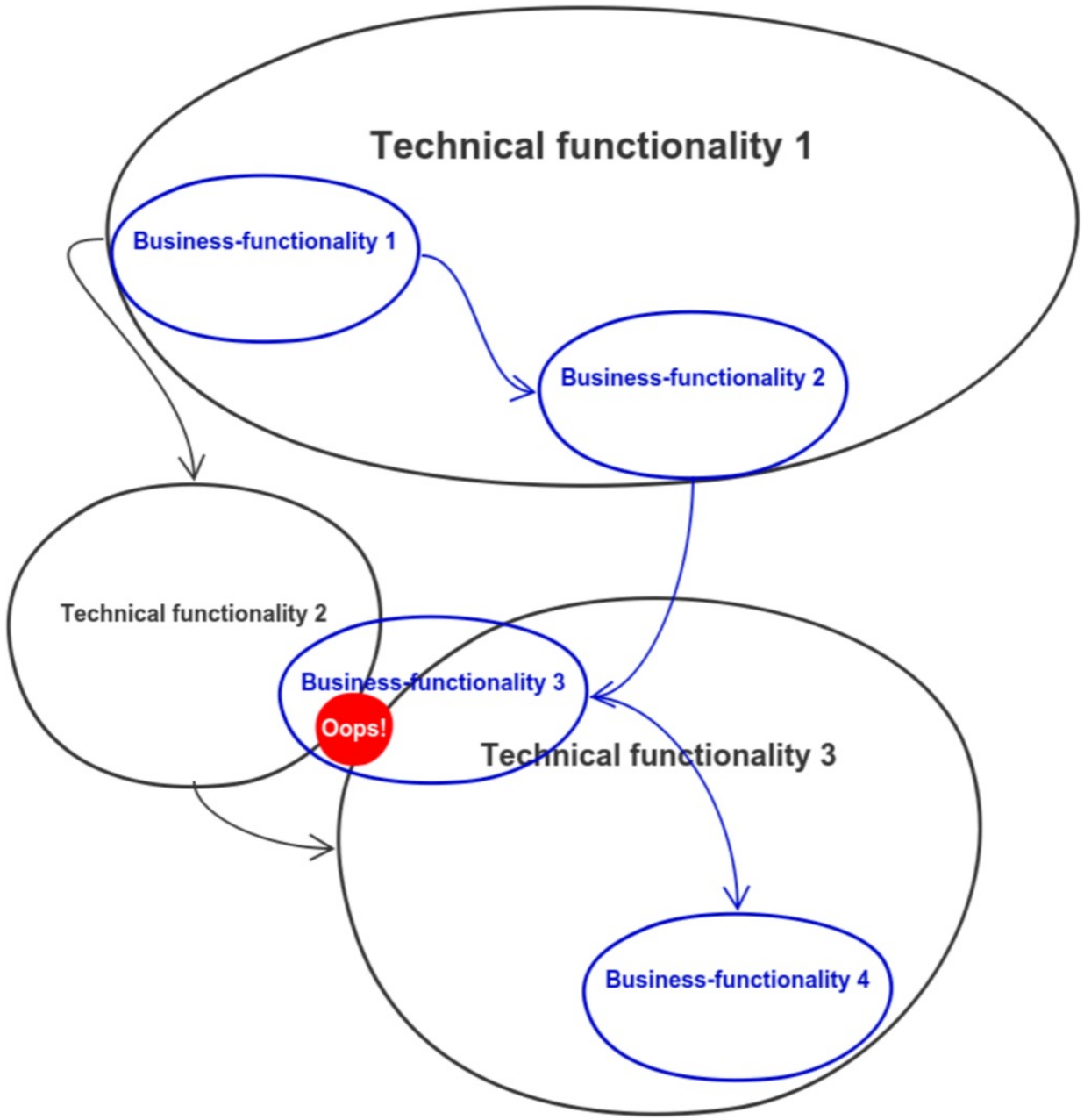
- Difficultés
  - complexe à structurer, tester et maintenir si les microservices se multiplient (automatisation obligatoire)
  - granularité des services
  - monitoring/observabilité
  - distribution des données



Découpage orienté *business*



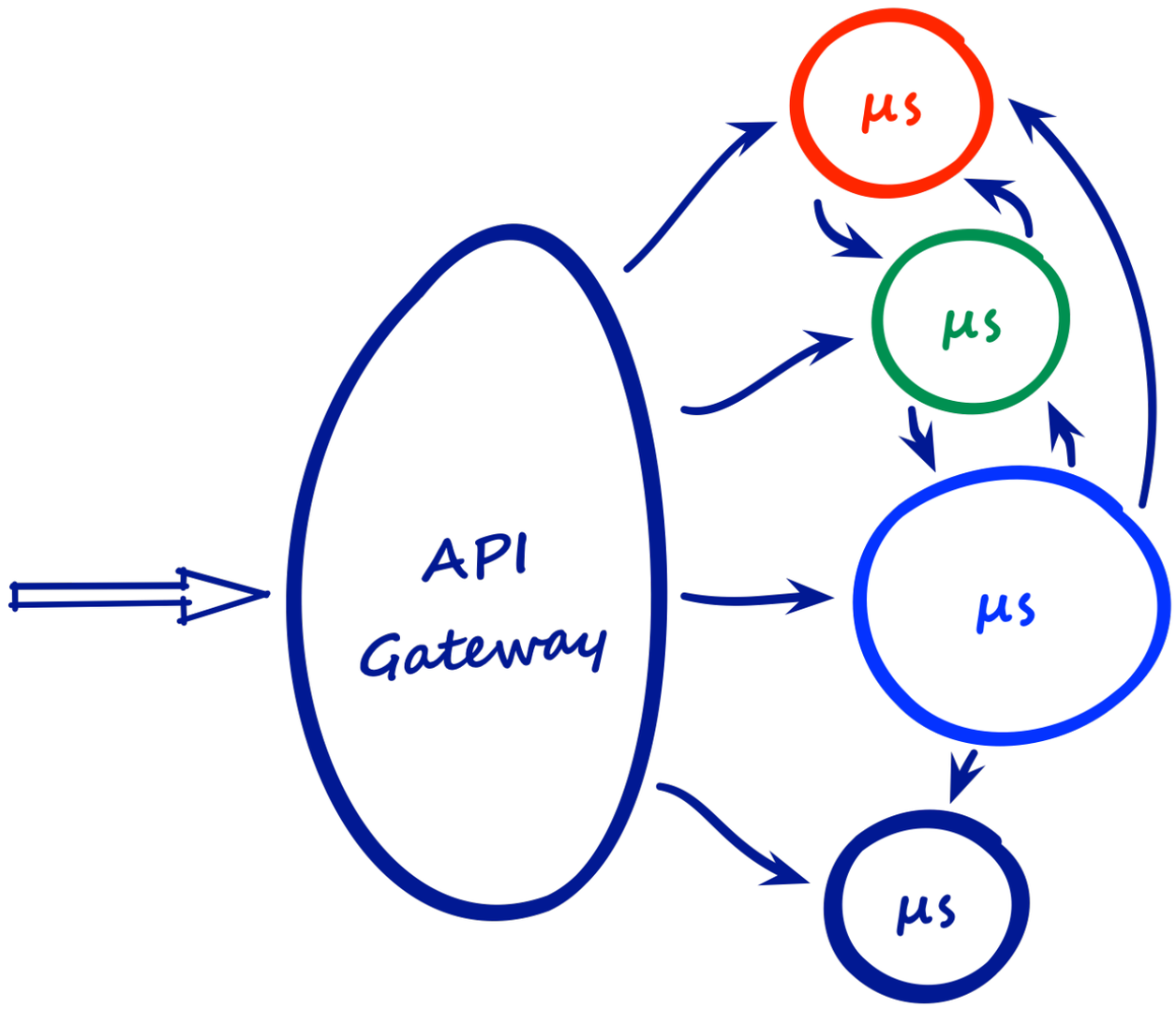
Découpage orienté technique


Technical functionality 1
Business-functionality 1
Business-functionality 2
Technical functionality 2
Business-functionality 3
Oops!
Technical functionality 3
Business-functionality 4

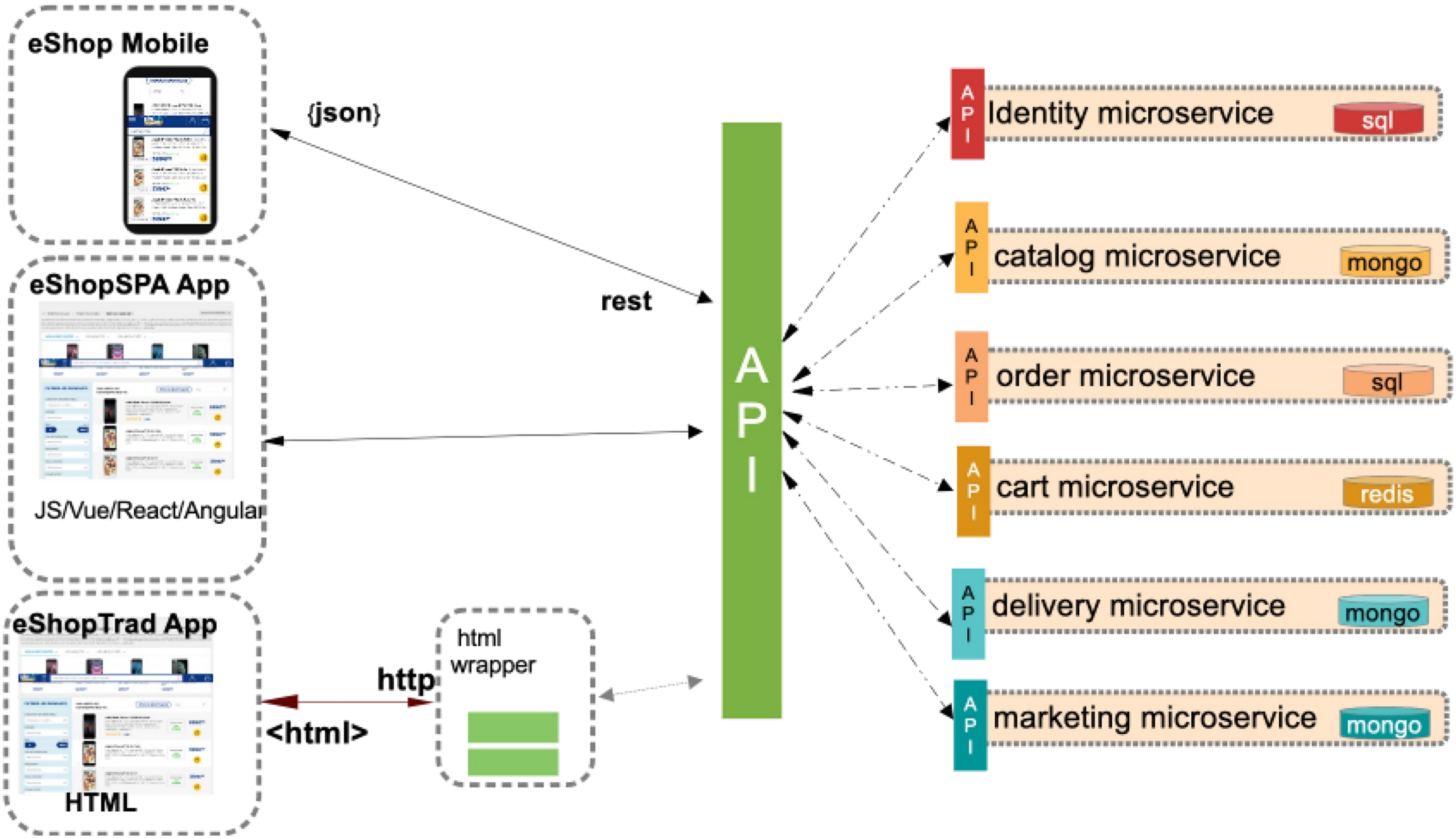
Problème !

API Gateway

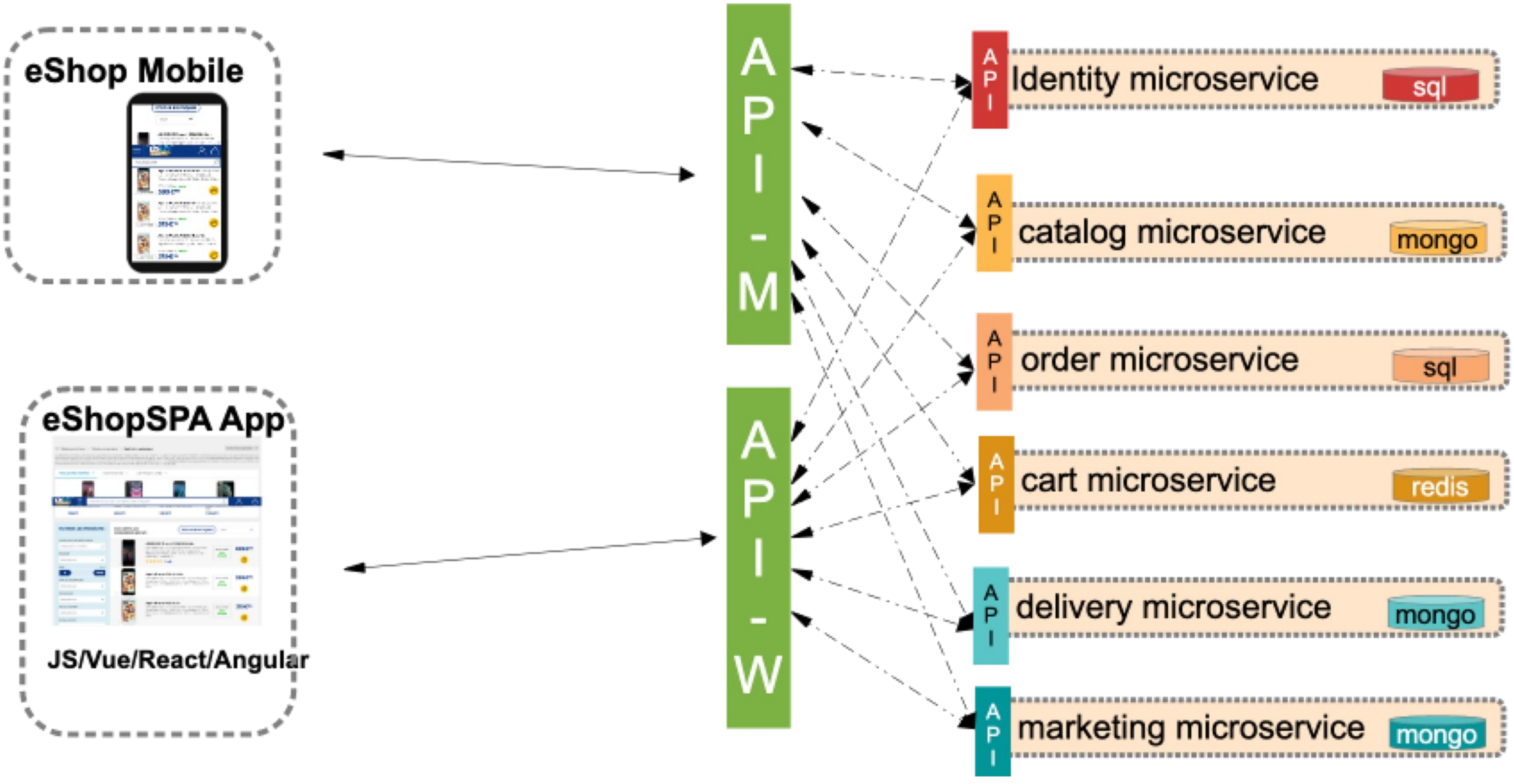
- Pour simplifier l'utilisation d'un backend composé de nombreux services, on utilise le pattern ***façade*** en déployant une ou plusieurs API Gateway



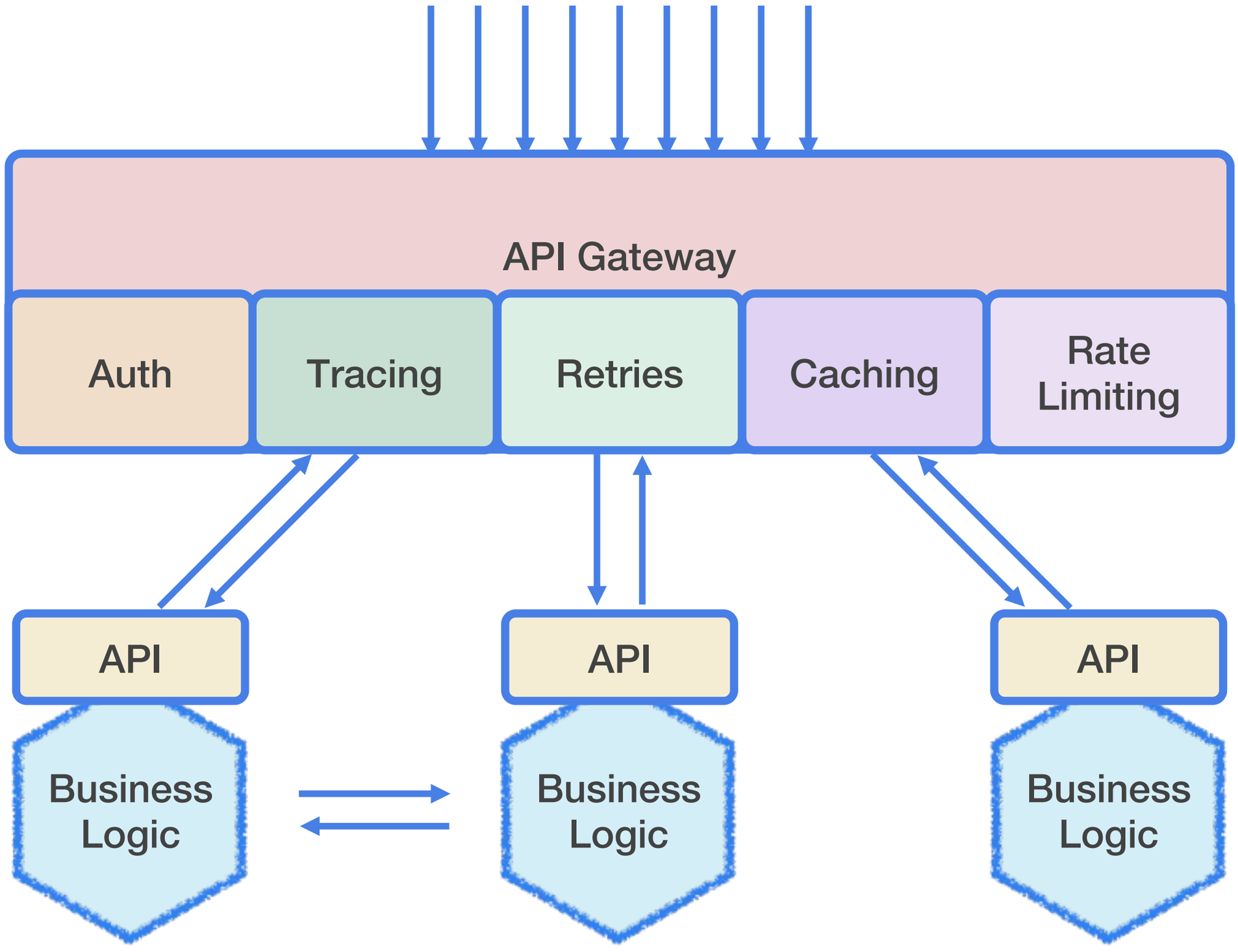
API Gateway (cont.)



Plusieurs Gateways



Principes


API Gateway
Auth
Tracing
Retries
Caching
Rate Limiting
API
Business Logic
API
Business Logic
API
Business Logic

Avantages

- Sécurité renforcée
  - authentification centralisée
  - chiffrement des communications
- Gestion de trafic optimisée
  - équilibrage de charge (*load balancer*), *rate limiter*
- Performances
  - mise en cache des réponses
  - transformation des requêtes/réponses

Avantages

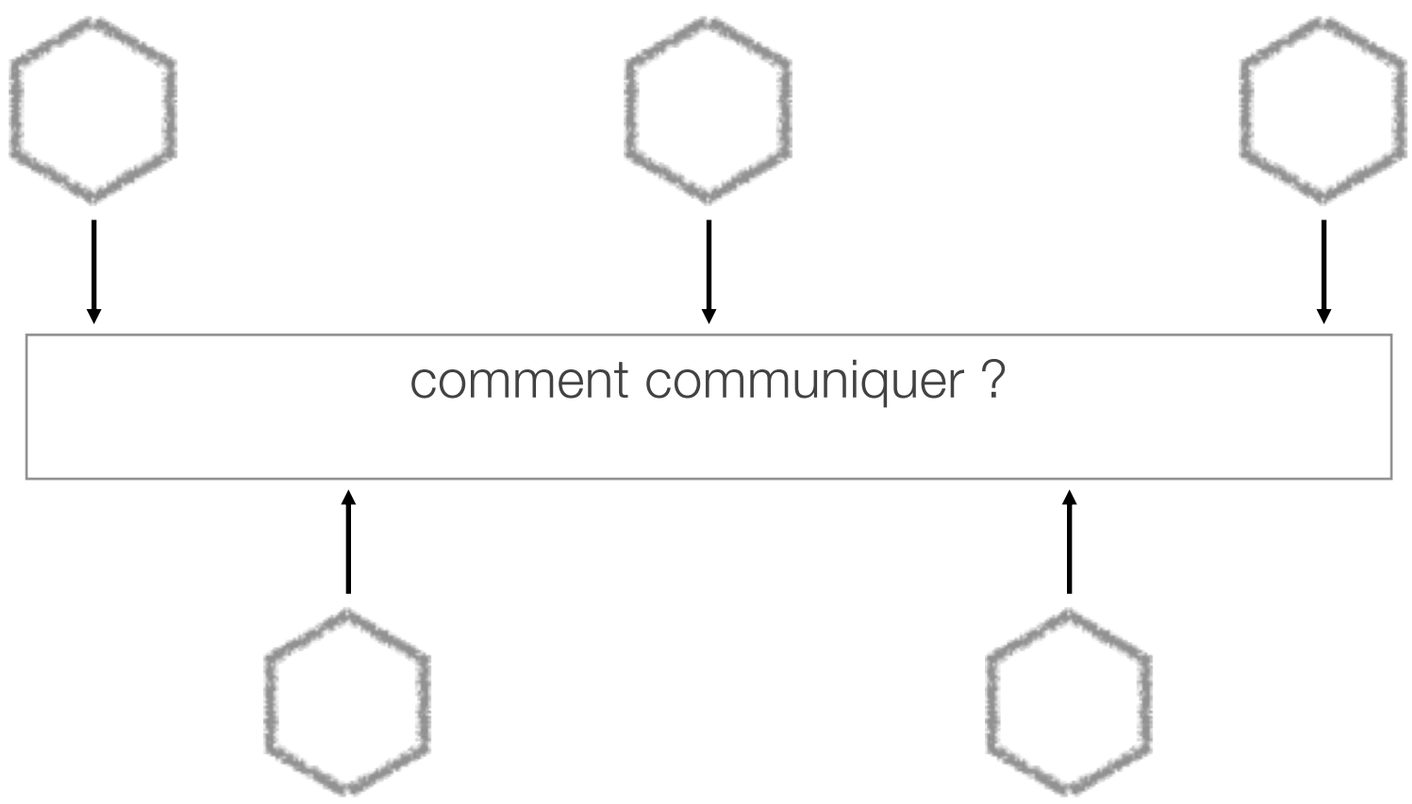
- Monitoring/observabilité
  - collecte de métriques et suivi de performances
  - détection des anomalies
- Simplification de l'architecture
  - point d'entrée unique
  - découplage des services
  - facilité d'intégration (protocoles par ex.)

Difficultés

- Complexité technique
  - configuration, maintenance
- Risques
  - point de défaillance unique
  - latence supplémentaire
  - performances
- Coûts

Communication

- Nécessité de communiquer entre microservices
  - pour se coordonner
  - pour échanger des données
- Gateway ou microservices ↔ microservices



Deux modes de communication

- Communication **synchrone**
  - un composant interagit avec un autre composant et doit attendre sa réponse pour poursuivre son exécution
- Communication **asynchrone**
  - un composant interagit avec un autre et peut poursuivre son exécution sans attendre la réponse

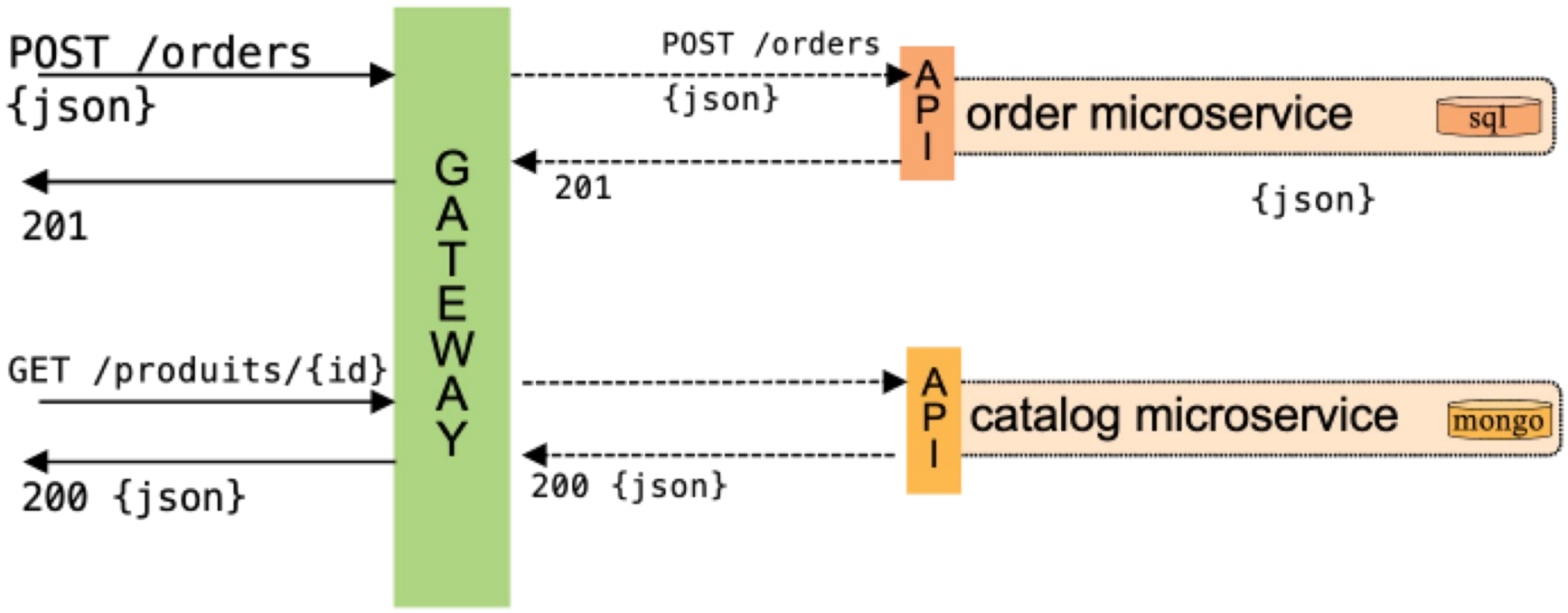
Exemple

- Traiter une commande d'un client
  - la gateway transfère la requête vers le microservice de gestion des commandes, attend la réponse puis répond
  - le microservice de gestion des commandes
    - interroge le microservice de catalogue pour obtenir des informations sur les produits (référence, prix, disponibilité) — il a besoin de la réponse pour poursuivre son action
    - puis transfère la commande au microservice de préparation/livraison - il n'a pas besoin de réponse
    - transférer une confirmation au microservice d'envoi de mail et au microservice SMS pour informer le client — il n'a pas besoin de réponse

Communication synchrone

- Un composant interagit avec un autre et attend la réponse pour terminer ses actions
  - la gateway transfère une requête vers le service **commande**, et attend la réponse du service pour retourner une réponse
  - le service commande interroge le service **catalogue** pour obtenir le tarif et la disponibilité d'un produit et finaliser la création de la commande
- Requête-réponse sur une API Restful ou GraphQL
  - gateway ↔ microservices: communication directe
  - microservices ↔ microservices : communication au travers de la gateway

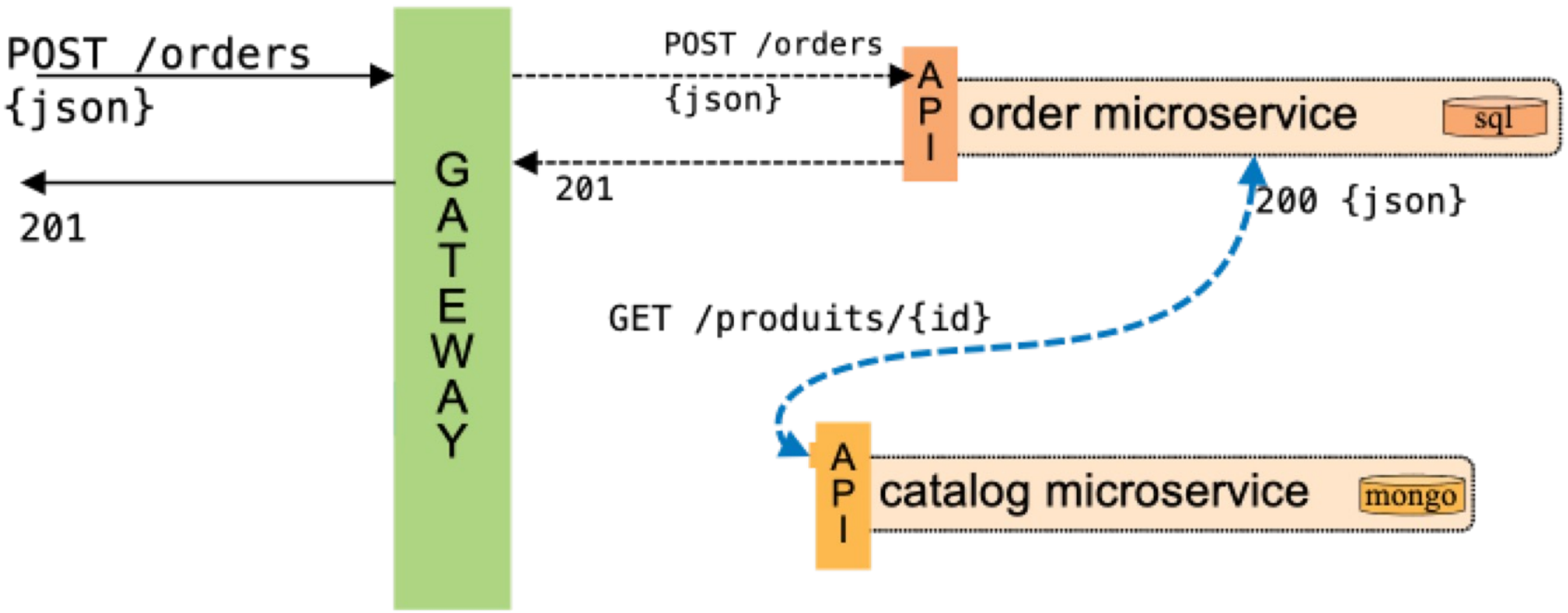
Gateway ↔ microservices



- La gateway a besoin de connaître les microservices et leur localisation

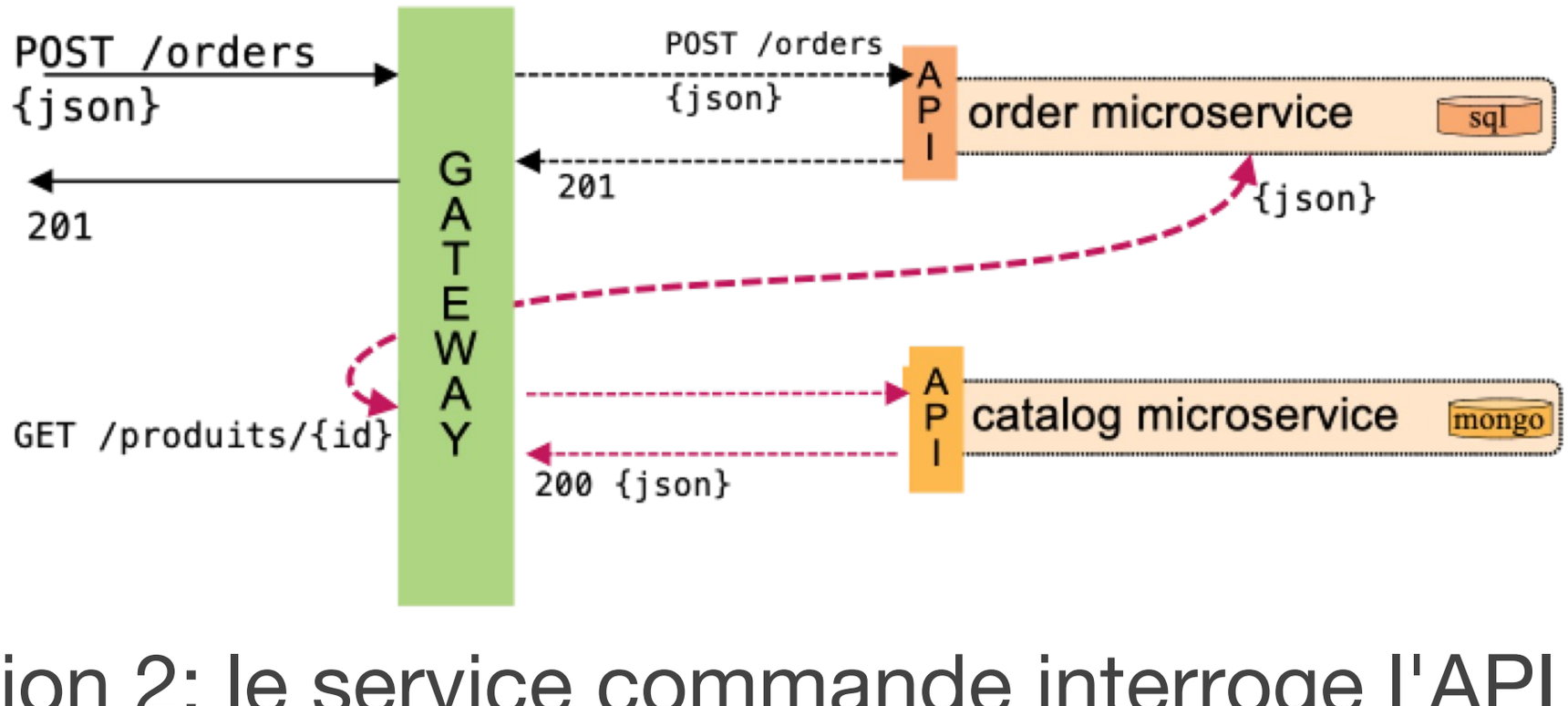
Microservices ↔ microservices

- Le service de création de commandes doit interroger le catalogue



- Solution 1: le service commande interroge directement l'API catalogue
- Inconvénient: le microservice commande doit connaître cette API et sa localisation

Microservices ↔ microservices



- Solution 2: le service commande interroge l'API catalogue au travers de la gateway
- Avantage: seule la gateway a besoin de connaître les microservices
- Inconvénient: dans certains cas il faudra transporter un token d'authentification pour interroger la gateway

Communication asynchrone

- Étudiée ultérieurement

Infrastructure

- Développement
  - docker-compose permet de déployer les microservices dans un ensemble de conteneurs docker sur une seule machine
- Production
  - docker-compose pour de petites applications
  - pour une montée en charge, kubernetes permet de gérer la répartition des microservices sur plusieurs machines

Réalisation en PHP/Slim

- Utiliser une librairie client HTTP conforme PSR-7 pour les messages de requêtes/réponses
- Intérêt : les requêtes / réponses de la librairie ont le même type que les requêtes/réponses de Slim
- Exemple de librairie : guzzlehttp/guzzle
  - https://docs.guzzlephp.org/en/stable/

Exemple: action de la gateway (Guzzle)

Un client guzzle est injecté dans l'action

```php
class GatewayConsulterPraticienAction
{
    private ClientInterface $remote_praticien_service;

    public function __construct(ClientInterface $client)
    {
        $this->remote_praticien_service = $client;
    }

    public function __invoke(ServerRequestInterface $request,
                             ResponseInterface $response,
                             array $args): ResponseInterface
    {
        $id = $args['id'];
        $response = $this->remote_praticien_service->get("praticiens/$id");

        return $response;
    }
}
```

La réponse guzzle est une réponse Slim

Requête synchrone vers le micro-service praticiens