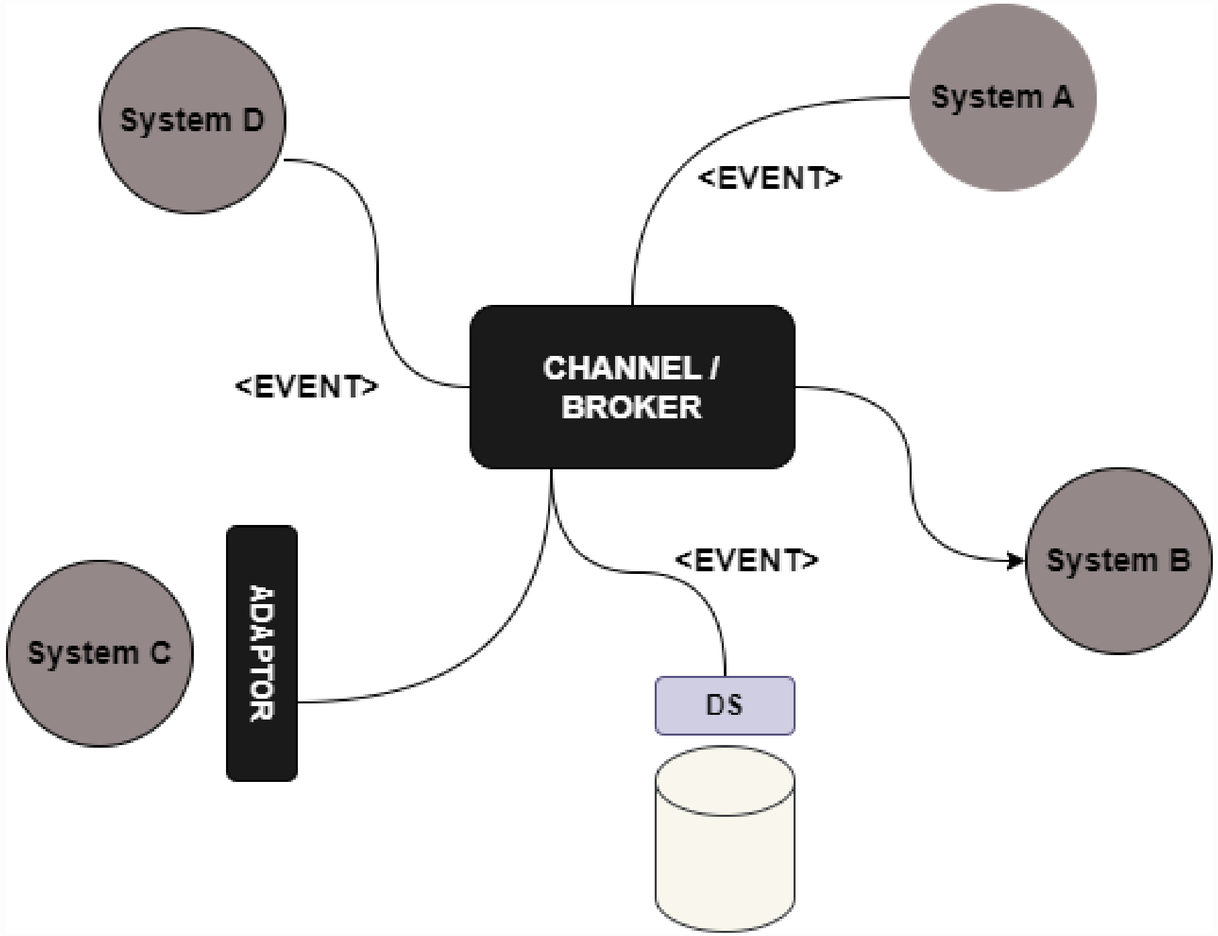
Piloté par les événements

Architecture

Concepts et pratiques



Régis ATEMENGUÉ

@regis_ate www.regisatemengue.com

Objectifs du cours

- Comprendre les concepts fondamentaux de l'architecture Event-Driven
- Concevez un EDA robuste, évolutif et facile à entretenir.
- Utiliser tous les concepts pour implémenter une application basée sur une architecture événementielle

Aperçu du cours

Introduction

Introduction aux événements

Architecture pilotée par les événements (EDA)

Sourcing d'événements et CQRS

Quand utiliser la gestion des événements

Architecture

EDA apatride et avec état

Streaming et EDA

Journalisation et surveillance

Mise en œuvre EDA / Étude de cas

Conclusion

Introduction

Pourquoi apprendre l'architecture événementielle ?

En tant qu'architecte logiciel, il est important de savoir construire des architectures complexes, évolutives et robustes.

Il faut savoir migrer d’un style architectural à un autre.

Passer d'un monolithe à une architecture événementielle

Passer d'un microservice à une architecture de microservices événementiels

- Devient super populaire
- Indépendant de la plateforme
- Extrêmement évolutif

SOA

SOA tente généralement de créer des fonctionnalités métier à l'aide de composants réutilisables qui communiquent via un support découplé tel qu'un réseau ou un bus de services. Nous nous concentrerons sur la SOA avec ESB (Enterprise Service Bus), qui s'apparente aux architectures événementielles.

L'état d'esprit derrière les services d'application est de rendre les fonctionnalités réutilisables, et il s'agit d'une préoccupation centrale de conception dans ce type d'architecture, par opposition aux microservices et pilotés par les événements où les fonctionnalités de partage sont limitées (voire évitées). Alors que la SOA se concentre sur des fonctionnalités abstraites et réutilisables, les microservices et les architectures événementielles se concentrent sur l'organisation de leurs composants autour de domaines.

De plus, dans la SOA, le bus de services a tendance à devenir de plus en plus grand. La logique métier a tendance à être ajoutée au bus plutôt qu'aux services. Le bus assume également plusieurs responsabilités en plus de l'orchestration.

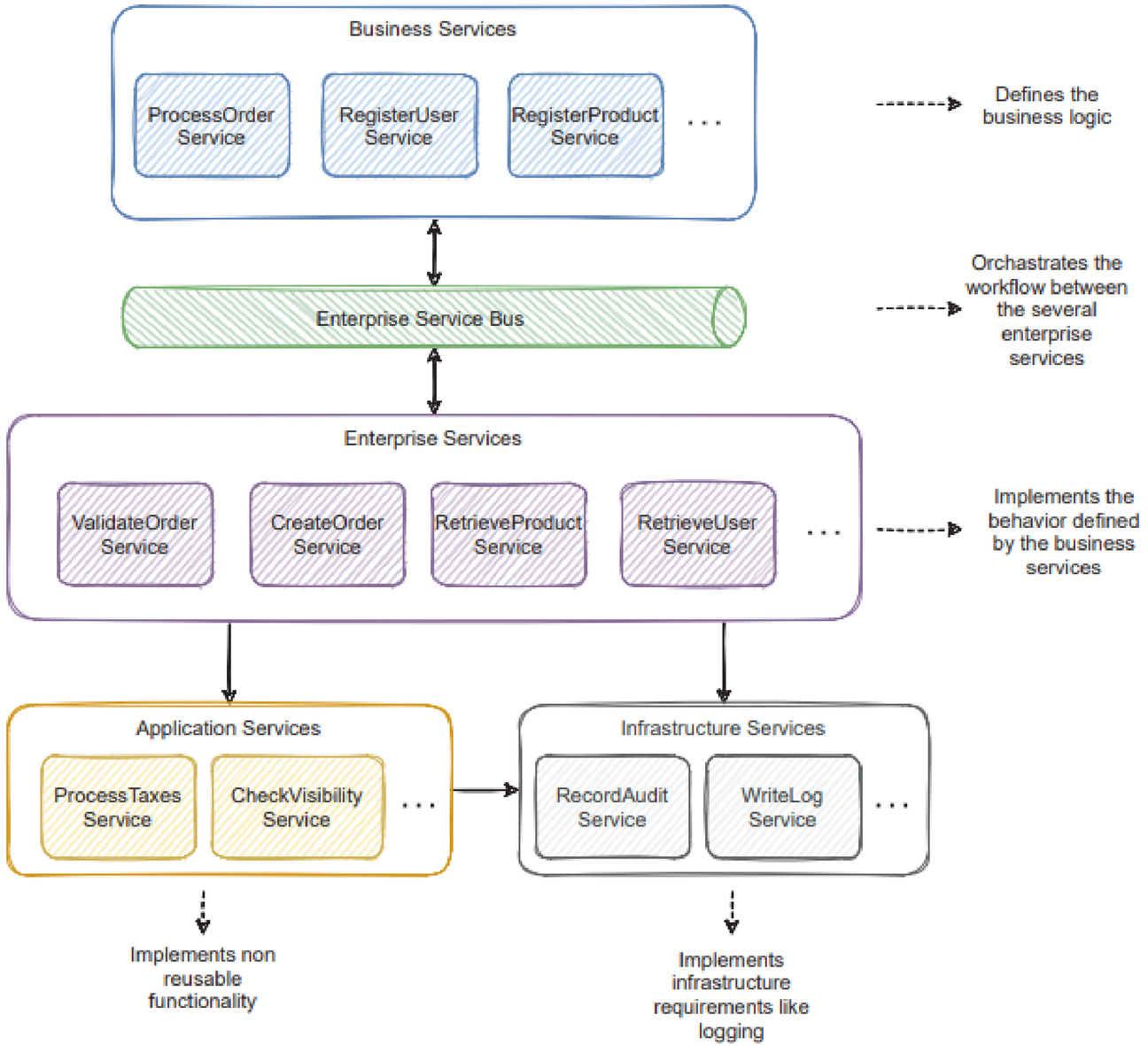


fig: 1 Organisation de service typique des architectures SOA

Microservices

Une architecture de microservices comprend plusieurs services spécifiques qui interagissent ensemble pour accomplir un processus ou un flux de travail donné. Les services sont regroupés selon un contexte délimité et dans le périmètre d'un domaine donné. Ils sont découplés et interagissent les uns avec les autres à l'aide d'un courtier de messages ou de requêtes HTTP.

Contrairement à la SOA où la réutilisabilité est une priorité importante, les microservices évitent le partage. Au lieu de cela, ils se concentrent sur ce domaine et ce contexte délimité.

Ce type d'architecture est fortement découplé puisqu'il existe une frontière physique entre Composants. Ils doivent communiquer via le réseau ; cela pose des limites naturelles à chaque service

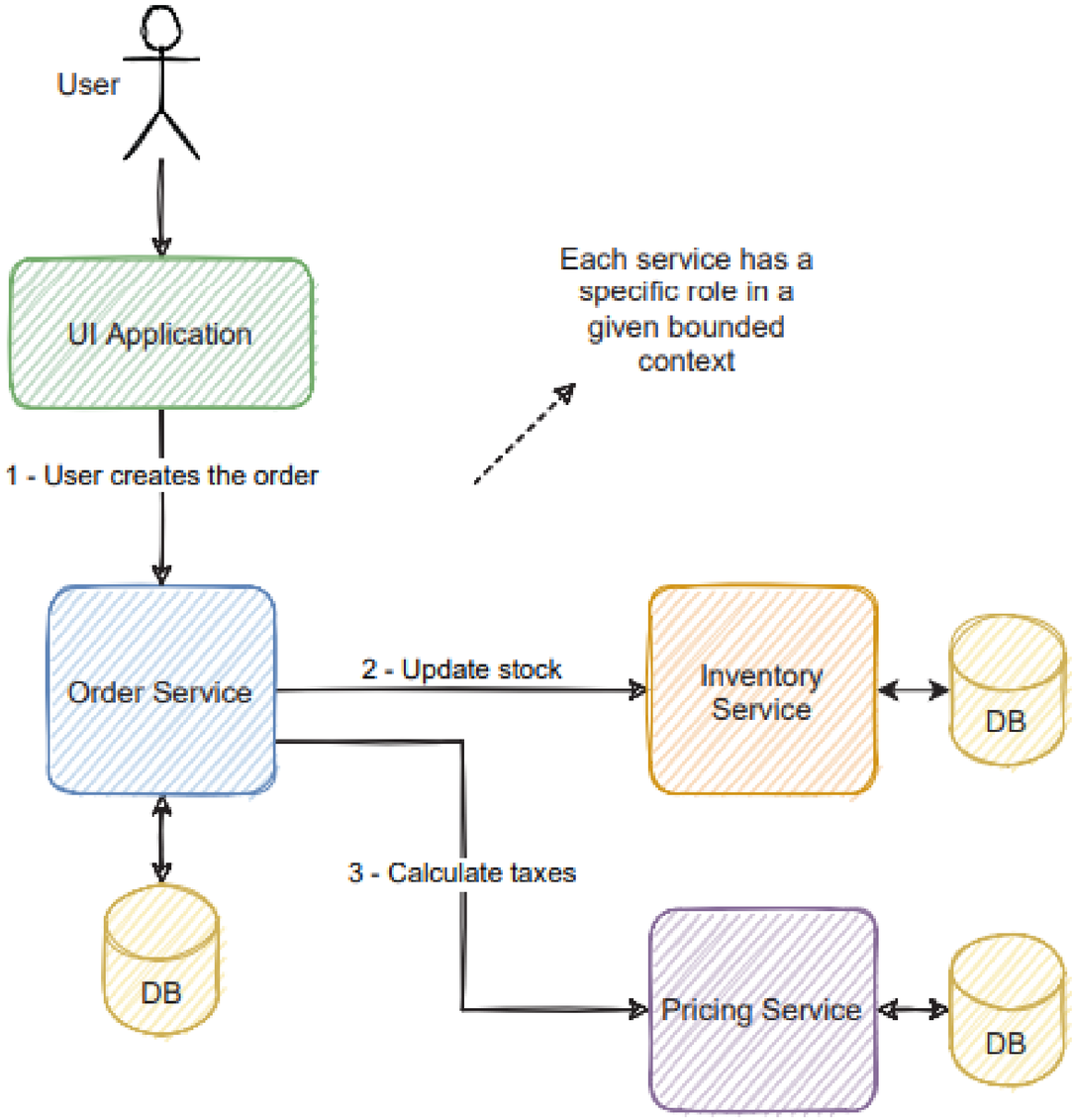


figure 2 :

Un exemple de la même plateforme de commerce électronique avec des microservices spécifiques pour chaque contexte délimité

Communication par microservices

- Peut-être la partie la plus importante de l'architecture des
- microservices. Dicte les performances, l'évolutivité, la mise en œuvre,
- etc. ***L'architecture événementielle gère la partie communication***

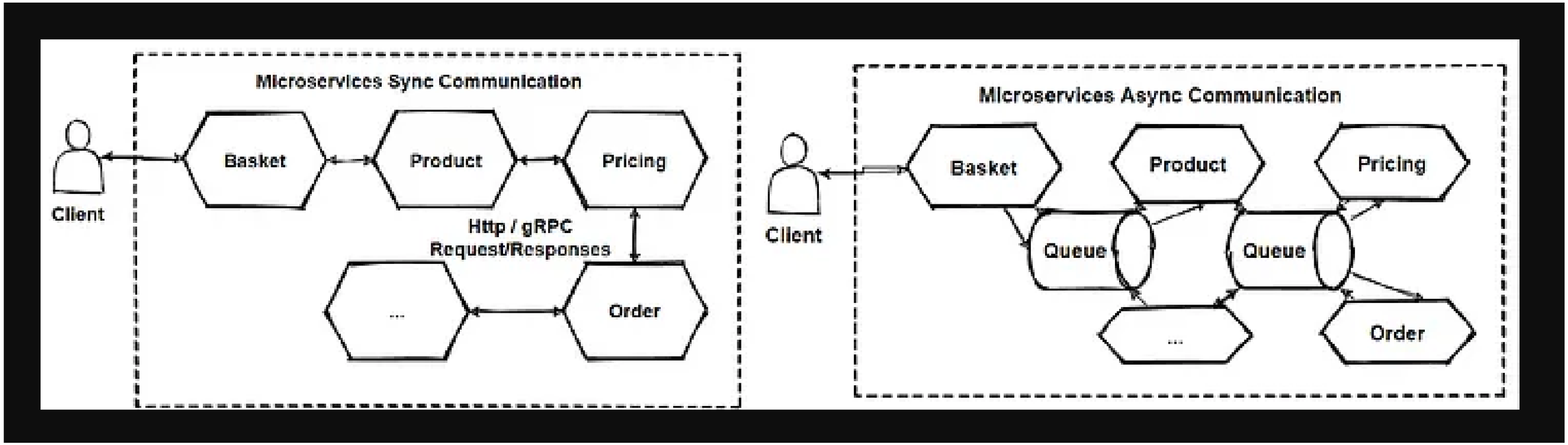


fig 3 : Types de communication de microservices : communication synchronisée ou asynchrone

Introduction aux événements

Commande et requête

- La communication classique entre services
- Services soit
  - ***Envoyer la commande***
  - ***Données de requête***

Commande

- Le service demande à un autre service de faire quelque
- chose. Cela peut être une réponse à la commande, généralement un indicateur de réussite ou d'échec.

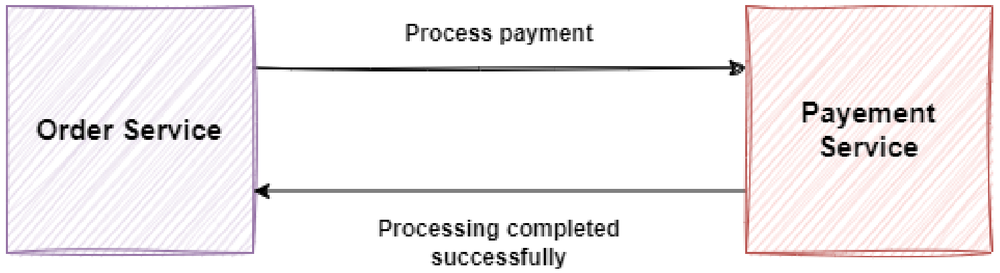


fig 4 : Commande

Requête

- Les services demandent des données à un autre
- service. Il y a toujours une réponse à la requête, contenant les données.

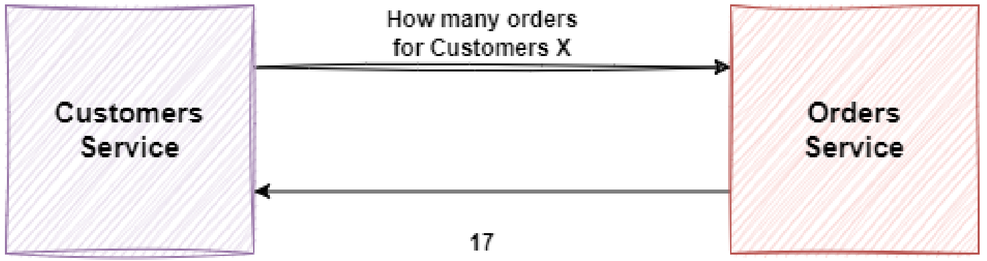


fig 5 : Requête

Problèmes avec la commande et la requête

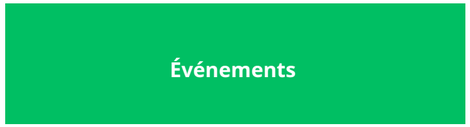
- ***Performance***
  - *Synchrone = le service appelant attend la commande/requête*
- ***Couplage***
  - *Le service appelant appelle un service spécifique*
  - *si le service appelé change - Le service appelant doit également changer*
  - *Plus de travaux, d'entretien*
- ***Évolutivité***
  - *Le service appelant appelle une seule instance d'un service. Si*
  - *cette instance est occupée, les performances sont affectées.*
  - *L'ajout d'une autre instance est possible, mais difficile.*
    - *Ajoutez un équilibreur de charge, des sondes de configuration, etc...*

Événements

Allégorie : le système d'événements du restaurant

Imaginez un restaurant comme un système logiciel et la commande de chaque client comme un événement. Dans cette analogie :

1. **Commande du dîner (événement) :**
   - Chaque fois qu'un client entre dans le restaurant et passe une commande, c'est comme déclencher un événement dans le logiciel. La commande peut porter sur différents plats, représentant différents types d'événements.
2. **Cuisine (gestionnaire d'événements) :**
   - La cuisine est le gestionnaire d'événements. Il écoute les commandes/événements et prépare les plats en conséquence. Lorsqu'une commande est passée, la cuisine commence à cuisiner, répondant à la demande spécifique.
3. **Serveur (Middleware) :**
   - Le serveur fait office de middleware. Une fois que la cuisine a préparé la commande, le serveur la sert au restaurant. Le serveur facilite la communication entre le restaurant (initiateur de l'événement) et la cuisine (gestionnaire de l'événement).
4. **Menu (Types d'événements) :**
   - Le menu représente différents types d'événements. Chaque plat du menu est comme un type d'événement unique. Tout comme un convive peut choisir parmi différents plats, le logiciel peut répondre à différents types d'événements.
5. **Interaction du dîner (action de l'utilisateur) :**
   - Chaque fois qu'un client interagit avec le menu ou demande quelque chose de spécifique, c'est comme une action de l'utilisateur dans un logiciel. Par exemple, un client peut demander un menu, signalant une interaction spécifique.
6. **Spécial du chef (événement par défaut) :**
   - Le plat du chef au menu est comme un événement par défaut. Si un convive ne précise pas un plat particulier, le chef prépare le spécial. De même, le logiciel peut avoir des actions par défaut pour certains événements s'il n'est pas explicitement géré.
7. **Zone d'attente (file d'attente des événements) :**
   - La zone d'attente est comme une file d'attente pour les événements. Les commandes (événements) sont mises en file d'attente et la cuisine les traite dans l'ordre de leur réception.



Indique que **quelque chose est arrivé** dans le système Il n'y a jamais de réponse à l'événement

Les événements sont des enregistrements de choses qui se sont produites. Cela signifie que quoi qu'ils représentent, c'est déjà fait.

Événements

Caractéristiques principales

Quelque chose est arrivé

Asynchrone

Ne renvoie jamais de réponse

Le service appelant n'a aucune idée de qui gère l'événement

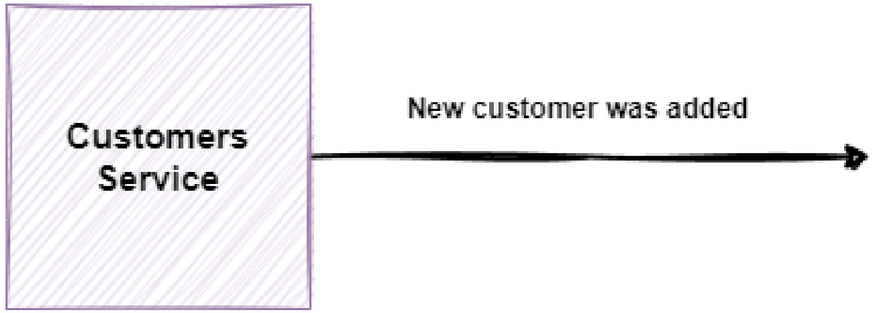


fig 6 : Événement

Événements

Contenu des événements

Deux types de données d'événement (**complet**et**aiguille**)

Complet

- Contient toutes les données pertinentes
- Généralement des données d'entité
- Aucune donnée supplémentaire n'est requise pour le traitement des événements

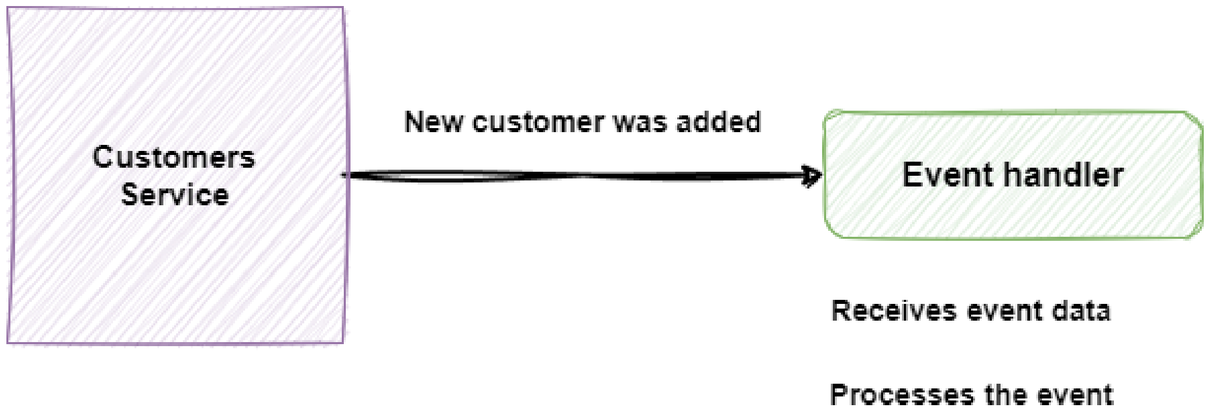


fig 7 : Événement terminé

Événements

Contenu des événements

Deux types de données d'événement (**complet**et**aiguille**)

Aiguille

- Contient un pointeur vers les données complètes de l'entité
- Les données complètes sont généralement stockées dans une base de
- données. Le gestionnaire d'événements doit accéder à la base de données pour recevoir des données complètes.

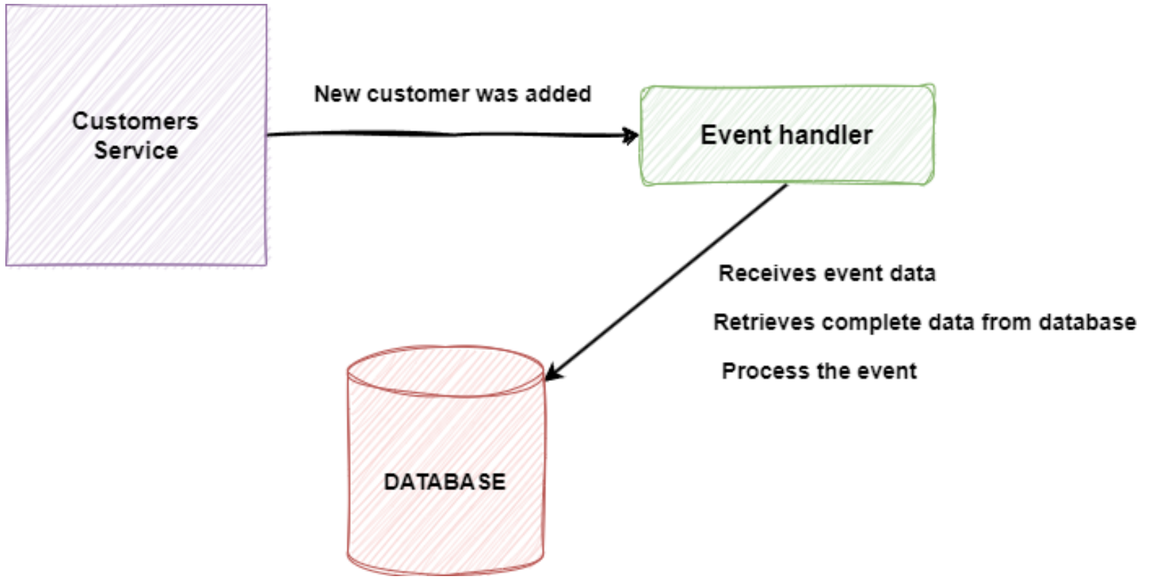


fig 8 : Événement de pointeur

Événements

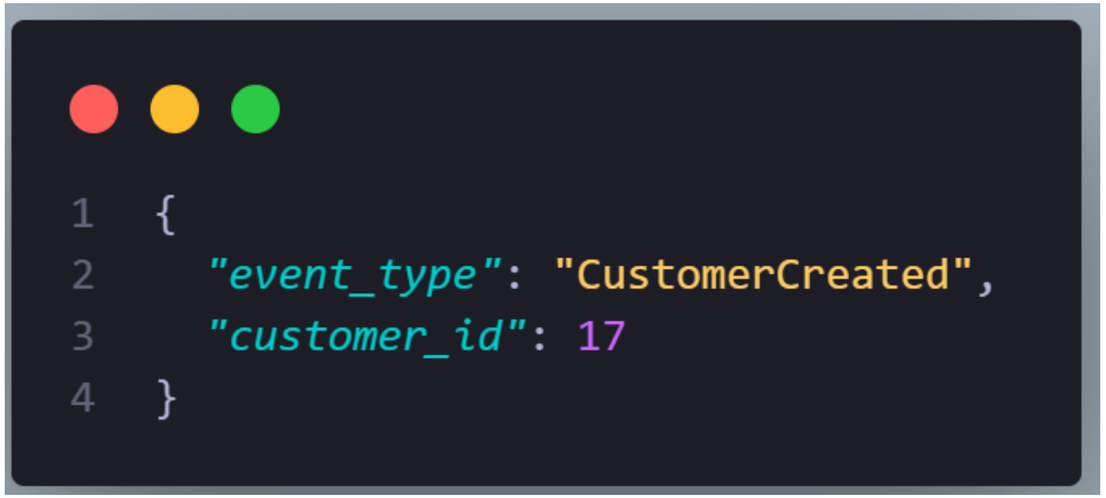
Gestion des événements de pointeur Flowof

Deux types de données d'événement

```
{
    "event_type": "CustomerCreated",
    "customer_id": 17,
    "first_name": "Ateba",
    "last_name": "Jean",
    "joined_at":  "2023-12-12"
}
```

fig 8 : Données complètes

Aiguille



```
{
    "event_type": "CustomerCreated",
    "customer_id": 17
}
```

fig 10 : Données du pointeur

Événements

Complet vs pointeur

Quand utiliser lequel ?

Complet

- La meilleure approche
- Rend l'événement complètement autonome
- Peut sortir des limites du système

Aiguille

- Utiliser quand
  - Les données sont volumineuses
  - Besoin de s'assurer que les données sont à jour
    - En supposant que la base de données est une source unique de vérité

Architecture pilotée par les événements (EDA)

Événements

Complet vs pointeur

Quand utiliser lequel ?

Complet

- La meilleure approche
- Rend l'événement complètement autonome
- Peut sortir des limites du système

Aiguille

- Utiliser quand
  - Les données sont volumineuses
  - Besoin de s'assurer que les données sont à jour
    - En supposant que la base de données est une source unique de vérité

"UN**architecture logicielle**modèle favorisant la production, la consommation et la réaction aux événements

- Wikipédia

Un style d'architecture logicielle qui utilise**événements**comme moyen de**communication**entre les prestations.

Dans une architecture événementielle, plusieurs de ces services interagissent les uns avec les autres pour accomplir un processus de niveau supérieur. Habituellement, l'interaction est composée d'une séquence d'événements entre les différents services

L'architecture événementielle (EDA) est un style architectural dans lequel le flux et le traitement de l'application sont déterminés par des événements. Les composants ou services communiquent en émettant et en réagissant à des événements, favorisant ainsi un couplage lâche entre les différentes parties du système.

Souvent appelé**AED**

Architecture pilotée par les événements

A trois composants principaux

- **Producteur**
- **Canal**
- **Consommateur**

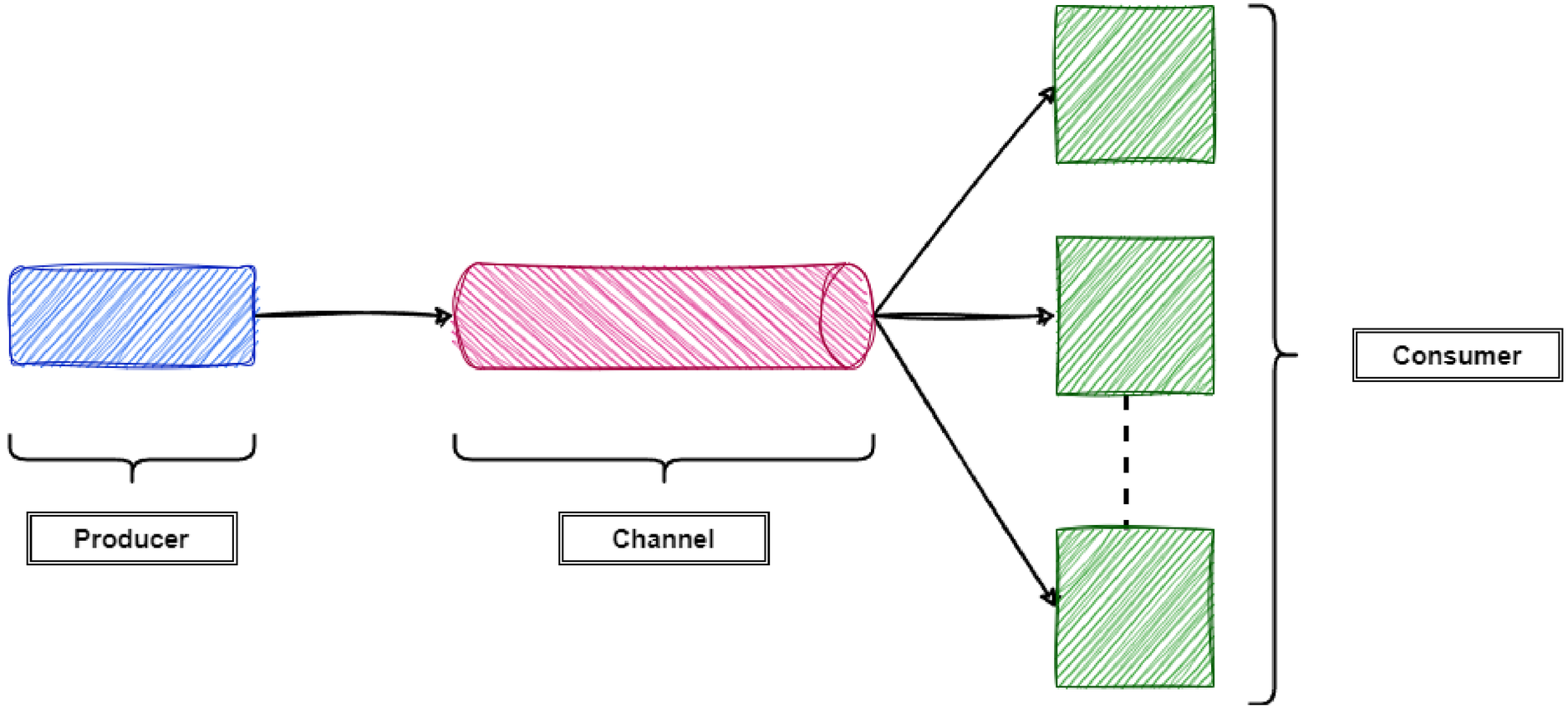


fig 11 : Composants principaux d'EDA

Architecture pilotée par les événements

Producteur

Le composant/service qui envoie l’événement.

Souvent appelé éditeur

Envoie généralement un événement signalant quelque chose que le composant a fait

Exemples

Service client------> Nouveau client Ajout d'un service d'inventaire d'événements -------> Événement d'article épuisé

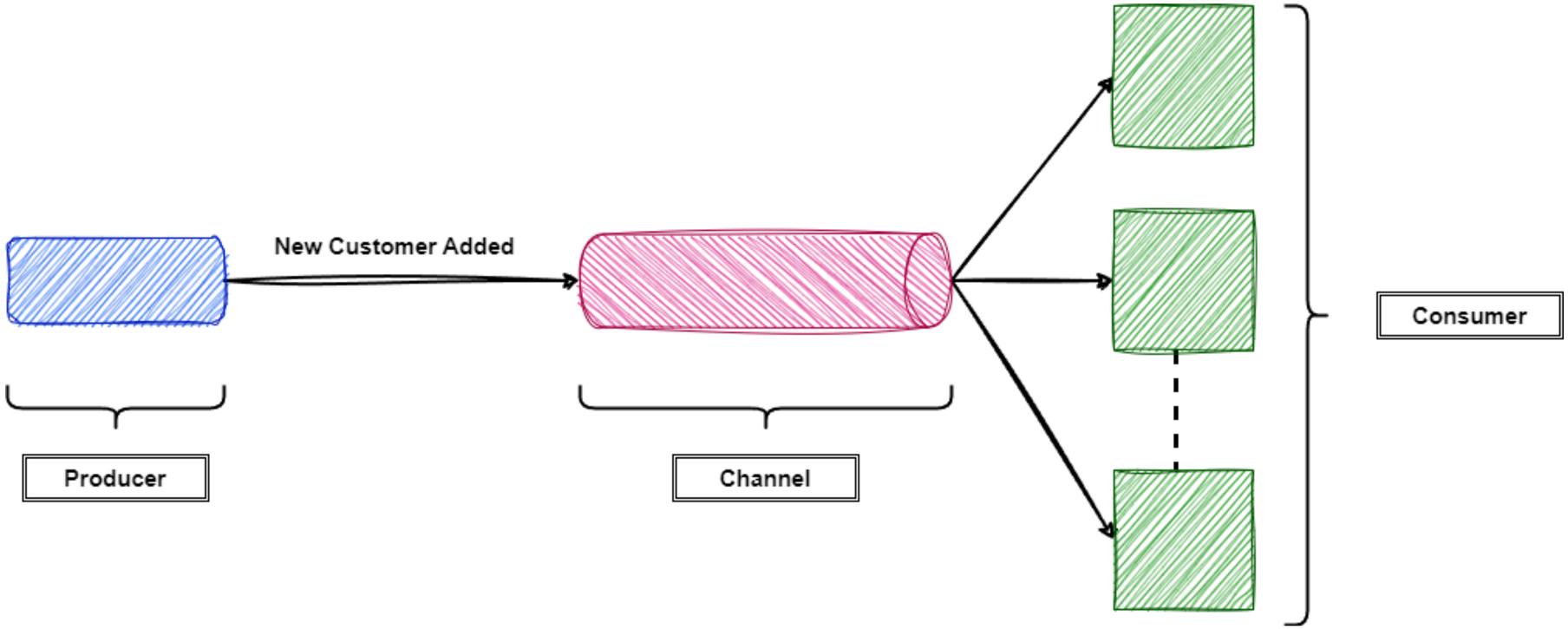


fig 12 : Producteur EDA

Architecture pilotée par les événements

Producteur

Le producteur envoie l'événement à la chaîne

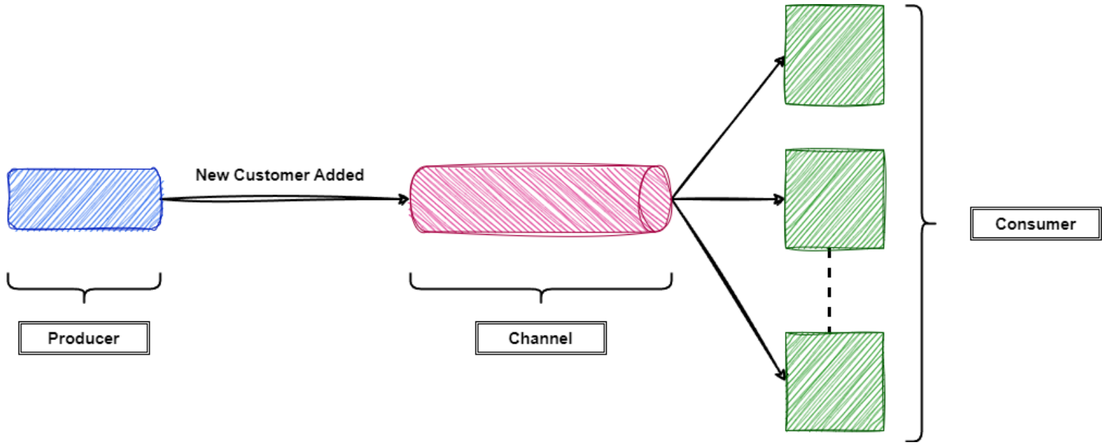


fig 13 : Producteur EDA

La méthode exacte d'appel du canal dépend du canal. Généralement,
on utilise un SDK dédié développé par le fournisseur du canal.

Utilise une sorte d'appel réseau, généralement avec des ports spécialisés et un protocole propriétaire

C'est à dire que RabbitMQ écoute sur le port 5672 et utilise le protocole AMQP. Producer

peut être développé en utilisant n'importe quel langage de développement

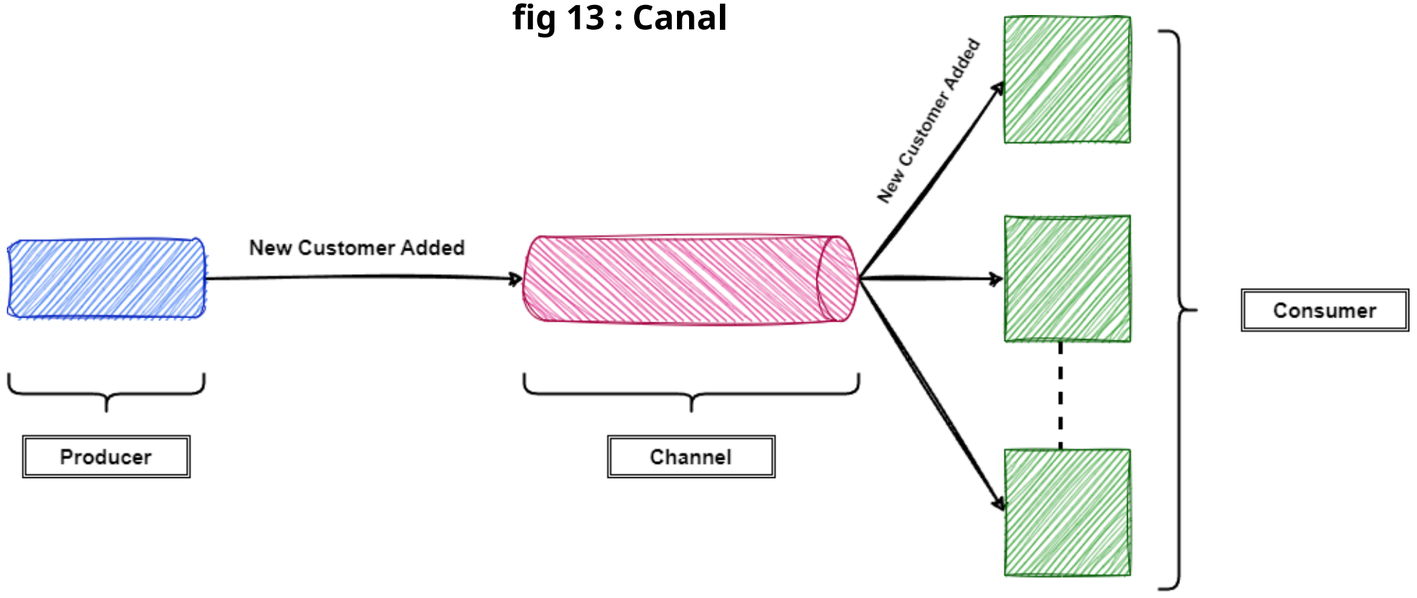
Architecture pilotée par les événements

Canal

- Le composant le plus important de l’EDA
- Responsable de la distribution des événements aux parties concernées
- La chaîne place l'événement dans une file d'attente spécialisée, souvent appelée Topic ou Fanout. Les
- consommateurs écoutent cette file d'attente et notent l'événement.

- Les détails de mise en œuvre varient énormément d’un canal à l’autre
- **LapinMQ**fonctionne différemment de**Kafka**qui fonctionne différemment des Webhooks, etc.

fig 13 : Canal

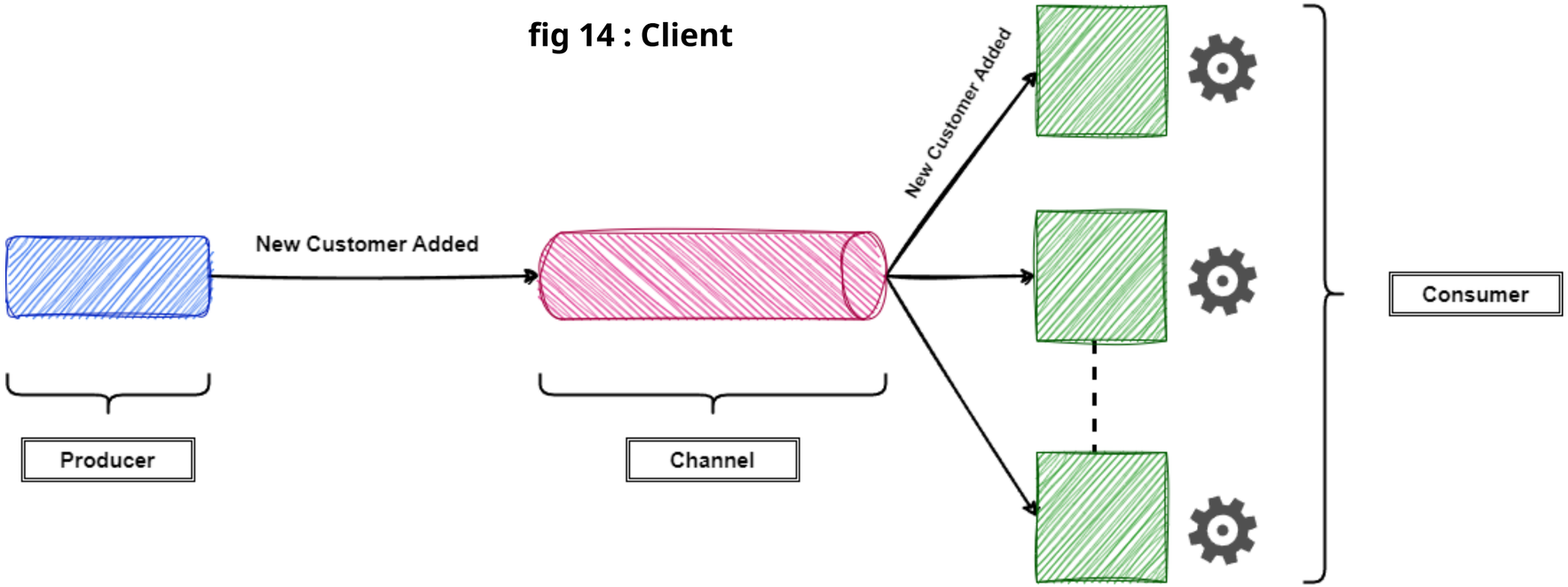


Architecture pilotée par les événements

Client

- Le composant qui reçoit l'événement est envoyé par le producteur et distribué par la chaîne. Peut être
- développé dans n'importe quel langage de développement compatible avec les bibliothèques du canal (le cas
- échéant) Parfois - signale lorsque le traitement est terminé (Ack)
- Le consommateur reçoit et traite l'événement

fig 14 : Client



Architecture pilotée par les événements

Tirer/Pousser

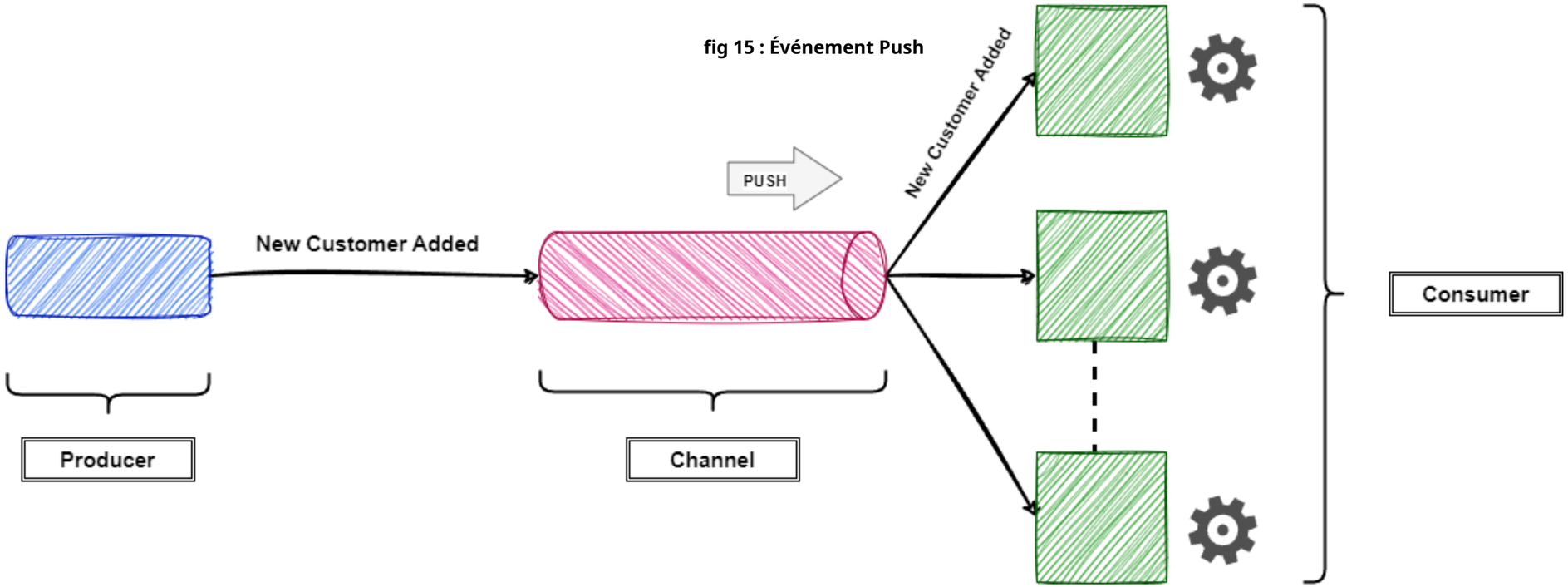
Le client obtient l'événement en utilisant

- **Tirer**
- **Pousser**

La méthode dépend du canal

- La chaîne pousse l'événement vers les Consommateurs

fig 15 : Événement Push



Architecture pilotée par les événements

Tirer/Pousser

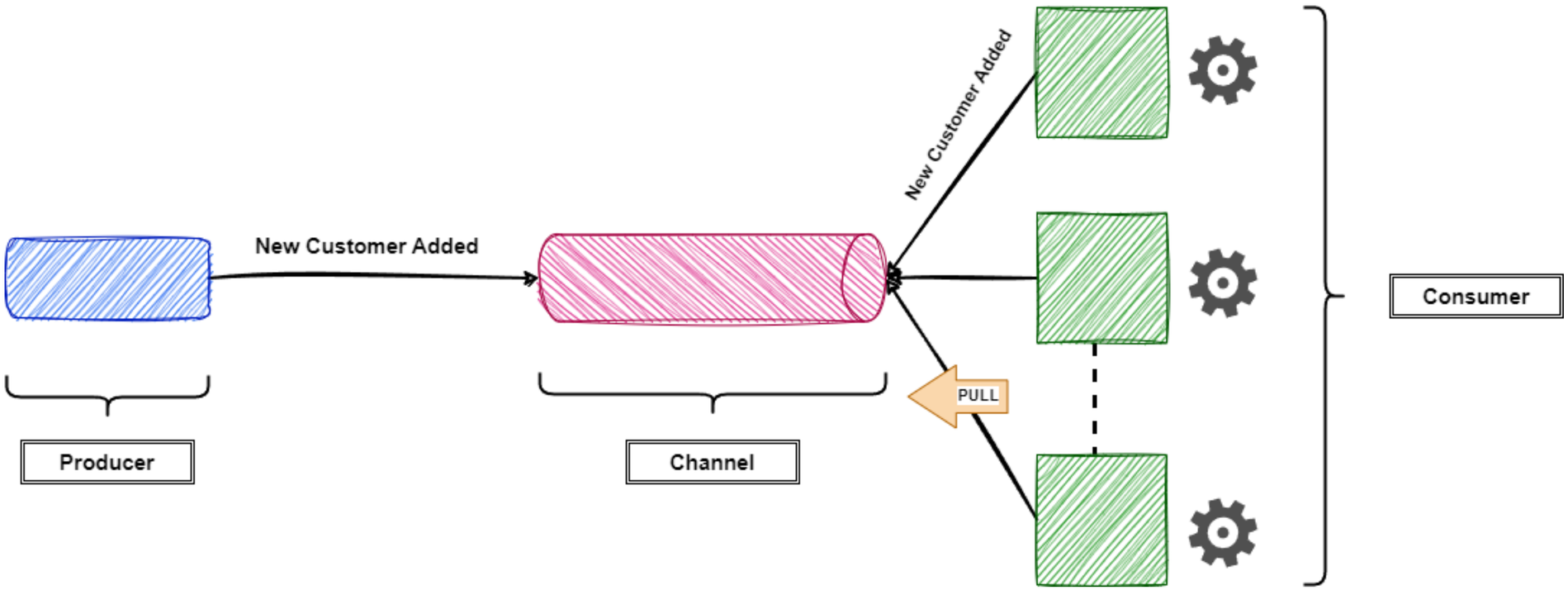
Le client obtient l'événement en utilisant

- **Tirer**
- **Pousser**

La méthode dépend du canal

- Les consommateurs interrogent la chaîne pour de nouveaux événements

fig 16 : Événement d'extraction



Architecture pilotée par les événements

Avantages de l'EDA

- *L'architecture événementielle présente de nombreux avantages par rapport aux autres paradigmes d'architecture*

Performance

- *EDA est une architecture synchrone*
- *Le canal n'attend pas de réponse du client Aucun goulot*
- *d'étranglement en termes de performances*

Couplage

- *Le producteur envoie les événements à la chaîne La chaîne*
- *distribue les événements aux sujets/files d'attente*
- *Les deux n'ont aucune idée de qui écoute l'événement (sauf dans les WebHooks) Pas de*
- *couplage*

Évolutivité

- *De nombreux consommateurs peuvent écouter les événements sur la chaîne.*
- *D'autres peuvent être ajoutés si nécessaire.*
- *La chaîne s'en fiche, le producteur ne le sait pas.*
- *Entièrement évolutif*

EDA contre PubSub

- *L'architecture basée sur les événements est souvent mentionnée avec Pub/Sub*
- *Pub / Sub = Publier et s'abonner*
- *Un modèle de messagerie utilisé par Event-Driven Architecture*

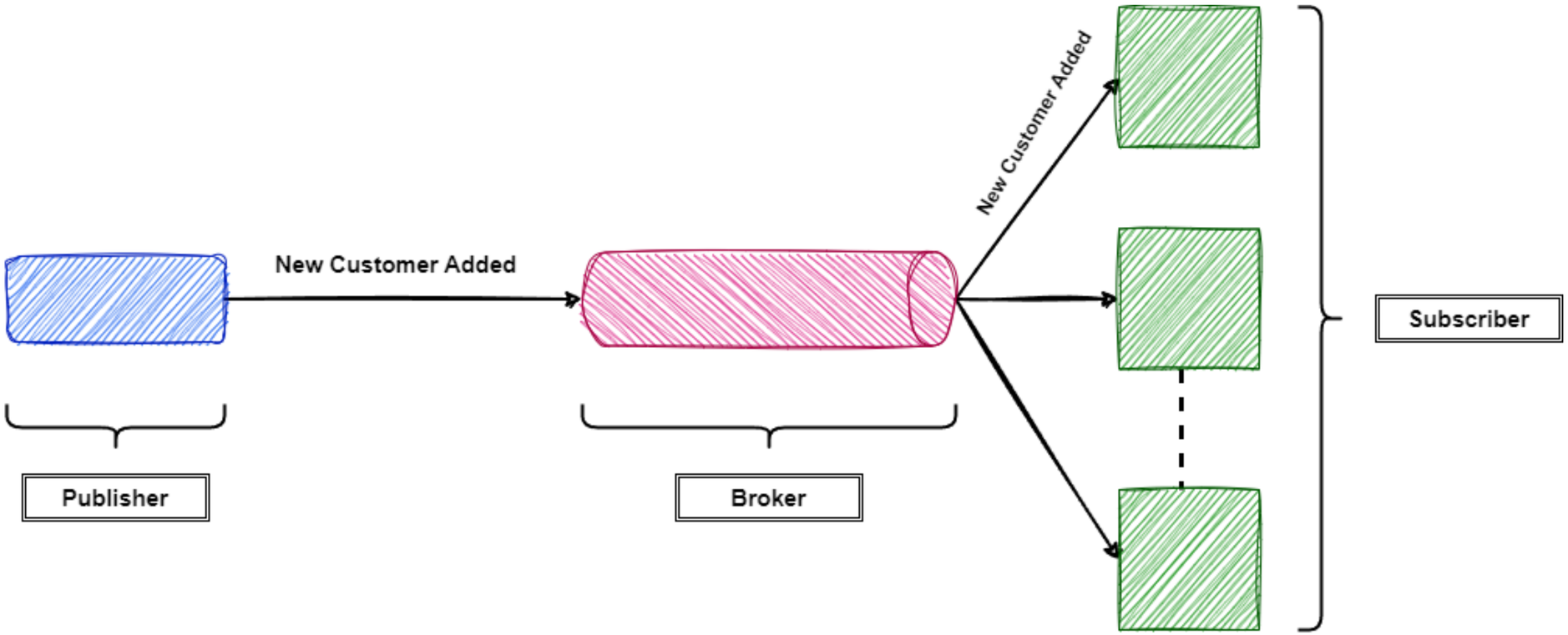


figure 17 : Pub/Sub

EDA contre PubSub

- L'architecture basée sur les événements et Pub/Sub sont extrêmement similaires
- Différence principale
  - EDA décrit toute l'architecture du système Pub/Sub
  - est un modèle de messagerie utilisé par le système
    - Pas exclusivement

Par exemple

- *Mon architecture événementielle utilise principalement Pub/Sub pour la communication interservices, mais je dispose de quelques API REST pour les requêtes synchrones.*

Commande en EDA

- Les moteurs de messagerie garantissent souvent l'ordre des messages

- Populaire principalement dans les files d'attente traditionnelles

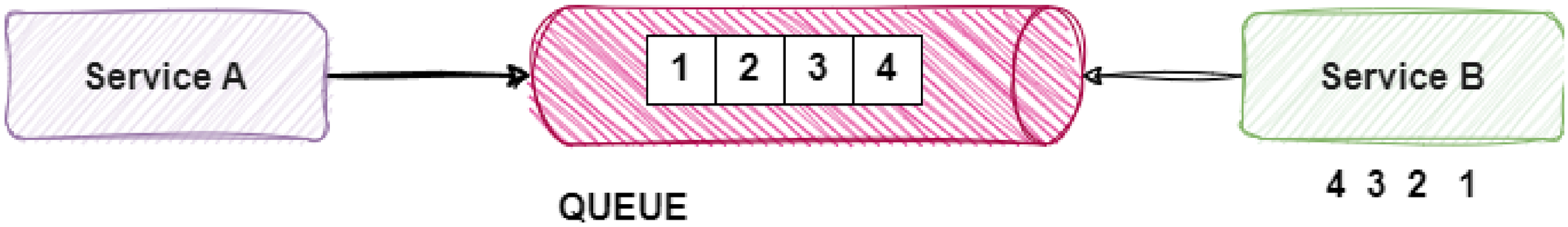


fig 18 : File d'attente

Commande en EDA

- Avec EDA (en particulier avec Pub/Sub), la commande n'est pas toujours garantie. La commande peut
- être affectée par la latence du consommateur, les performances du code, etc.

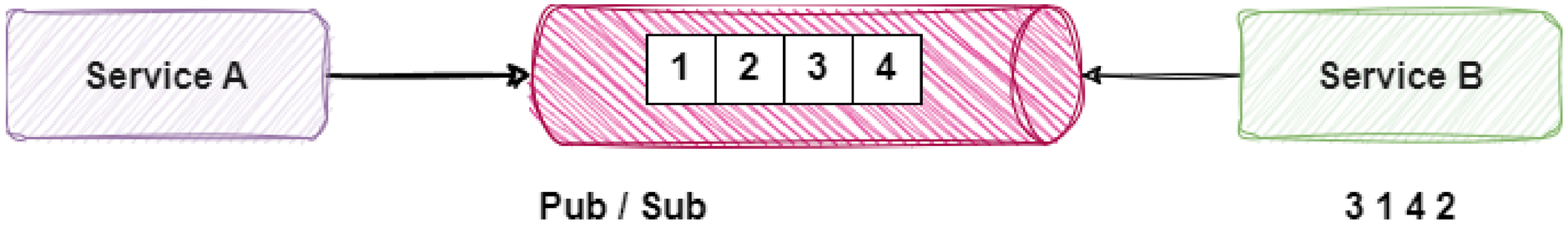


figure 19 : Pub/Sub

- si la commande est importante, assurez-vous de sélectionner un canal qui prend en charge cette fonctionnalité.
- Exemples
  - RabbitMQ le prend en charge
  - Le signal R ne fonctionne pas

Orchestration et chorégraphie

L'architecture événementielle utilise généralement l'une des deux approches suivantes pour interagir avec d'autres composants logiciels.

Orchestration

Chorégraphie

Orchestration

L'architecture événementielle utilise généralement l'une des deux approches suivantes pour interagir avec d'autres composants logiciels.

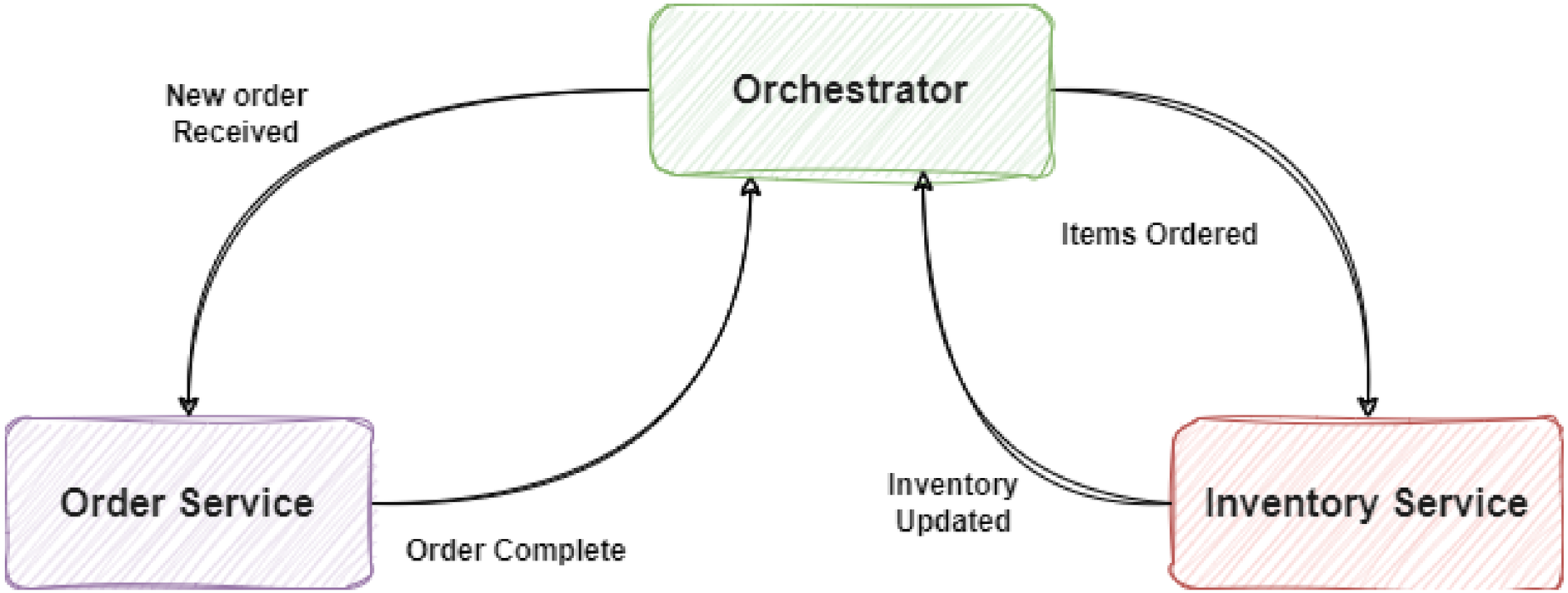


fig 20 : Orchestrateur

Chorégraphie

L'architecture événementielle utilise généralement l'une des deux approches suivantes pour interagir avec d'autres composants logiciels.

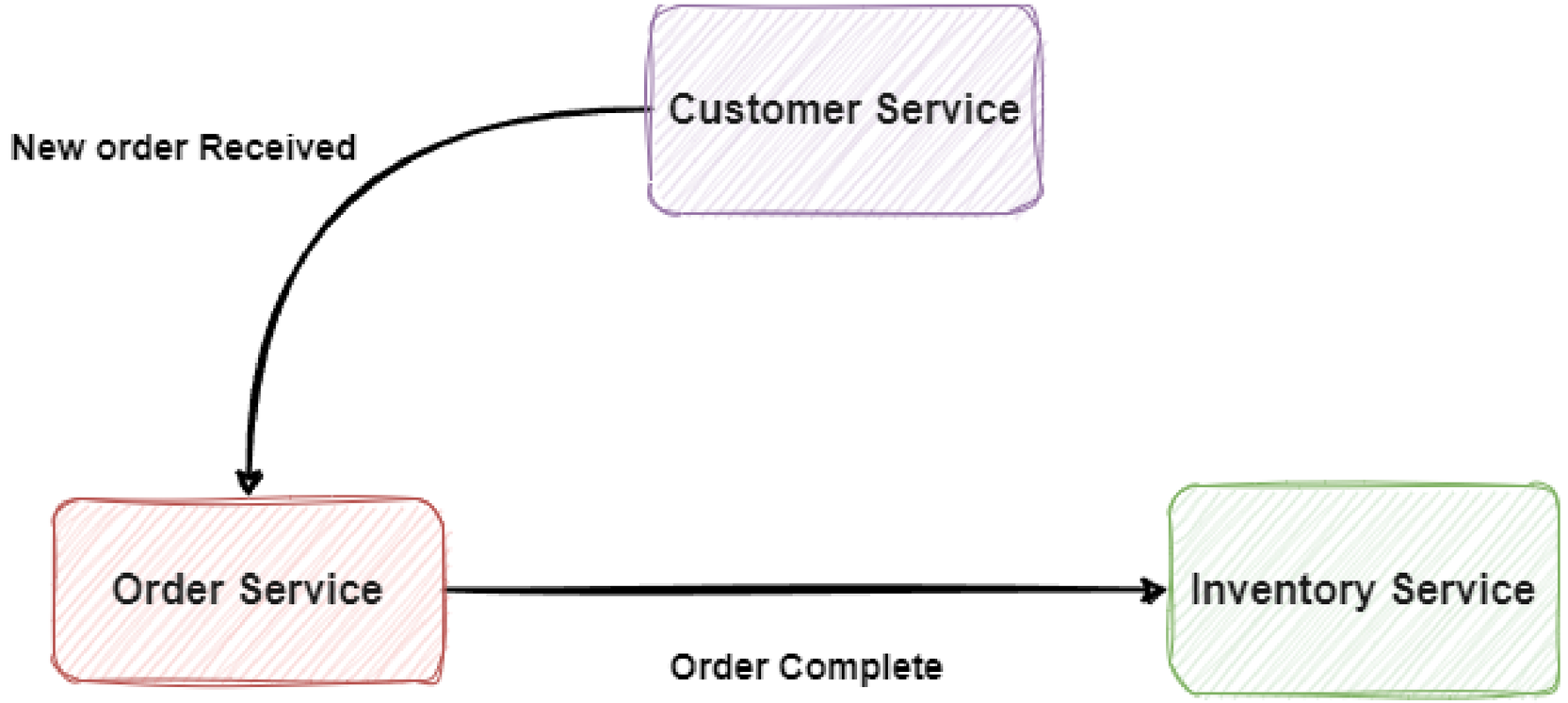


fig 21 : Chorégraphie

Orchestration

Avantages

- Simplicité
- Contrôle centralisé
- Visibilité
- Facilité de dépannage

Les inconvénients

- Couplage serré
- Point de défaillance unique
- Difficulté à ajouter, supprimer et remplacer des microservices
- Aérien

Chorégraphie

Avantages

- Couplage lâche
- Facilité d'entretien
- Contrôle décentralisé
- Communication asynchrone

Avantages

- Complexité
- Manque de contrôle central
- Manque de visibilité
- Difficulté à résoudre les problèmes

Sourcing d'événements et CQRS

Diffusion d'événements

Diffusion d'événements

Jusqu'à présent, nous avons parlé de l'architecture basée sur les événements.

Quelque chose s'est produit

Un événement a été créé

Quelqu'un a écouté l'événement et l'a géré

Le streaming consiste à permettre un flux d’événements distribué et ordonné.

Ce ne sont pas les mêmes, mais partagent des caractéristiques similaires

Diffusion d'événements

Les moteurs de streaming d'événements publient un flux d'événements, par exemple la télémétrie à partir de capteurs, le journal système, etc.

Les événements sont publiés dans un « flux » Les consommateurs s'abonnent à un flux spécifique

Les événements sont conservés dans un flux pendant une durée spécifiée

Les consommateurs peuvent récupérer les événements envoyés dans le passé (généralement jusqu'à quelques jours)

Les moteurs de streaming peuvent être utilisés comme base de données centrale. Une source unique de vérité

Tous les événements ne sont pas nécessairement gérés

Certains pourraient ne pas être pertinents

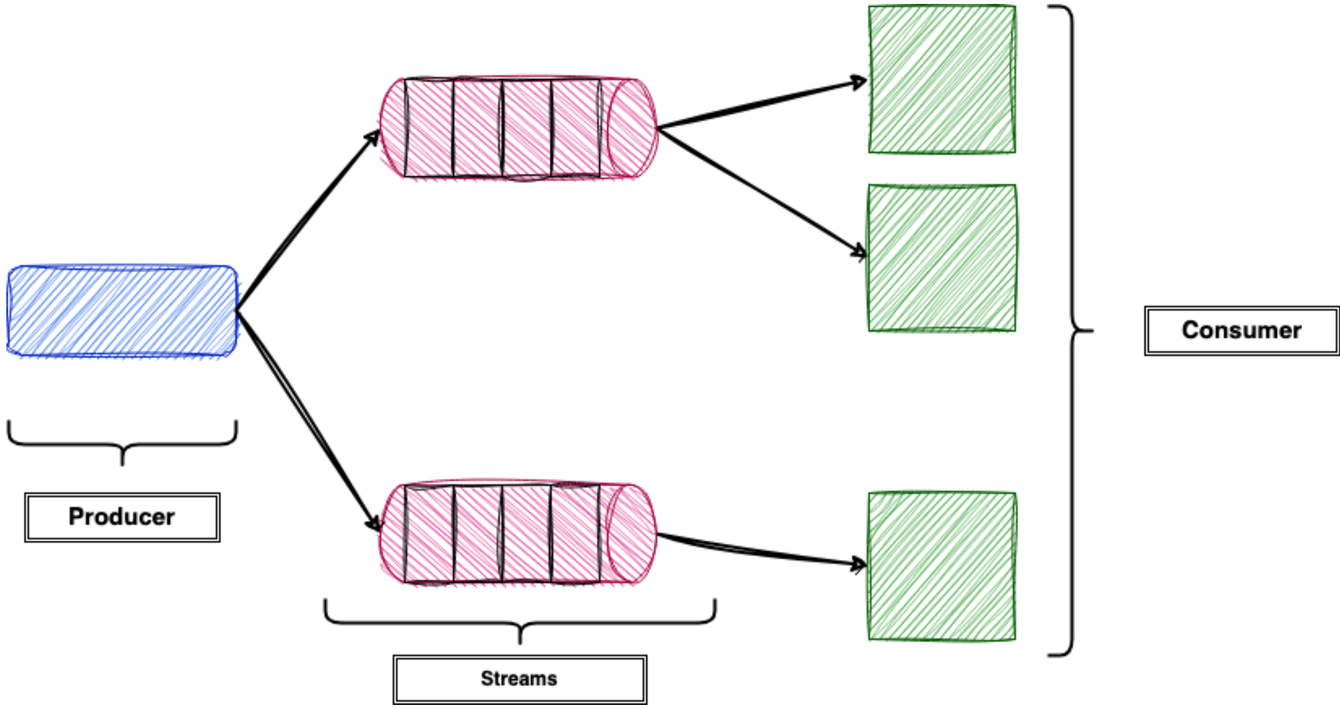


fig 22 Diffusion d'événements

Streaming d'événements et EDA

- Habituellement utilisé pour les événements générés en dehors du système. Les
- événements sont conservés
- Tous les événements ne sont pas gérés
- Charge élevée

- Habituellement utilisé pour les événements se produisant à l'intérieur du système. Les
- événements ne sont pas conservés.
- Tous les événements sont gérés Pas
- de charge élevée

Quand utiliser le streaming d’événements

- Lorsque le système doit gérer un flux d’événements extérieurs

- Quand les événements doivent être conservés pour une utilisation future

- Lorsqu'une charge élevée est attendue

Implémentation du streaming d'événements

Utiliser des outils de streaming spécialisés

Le plus remarquable

Journalisation et surveillance

Journalisation et surveillance

Crucial pour avoir une vue globale de ce qui se passe dans le système

Extrêmement important dans tout système distribué

Particulièrement difficile en EDA

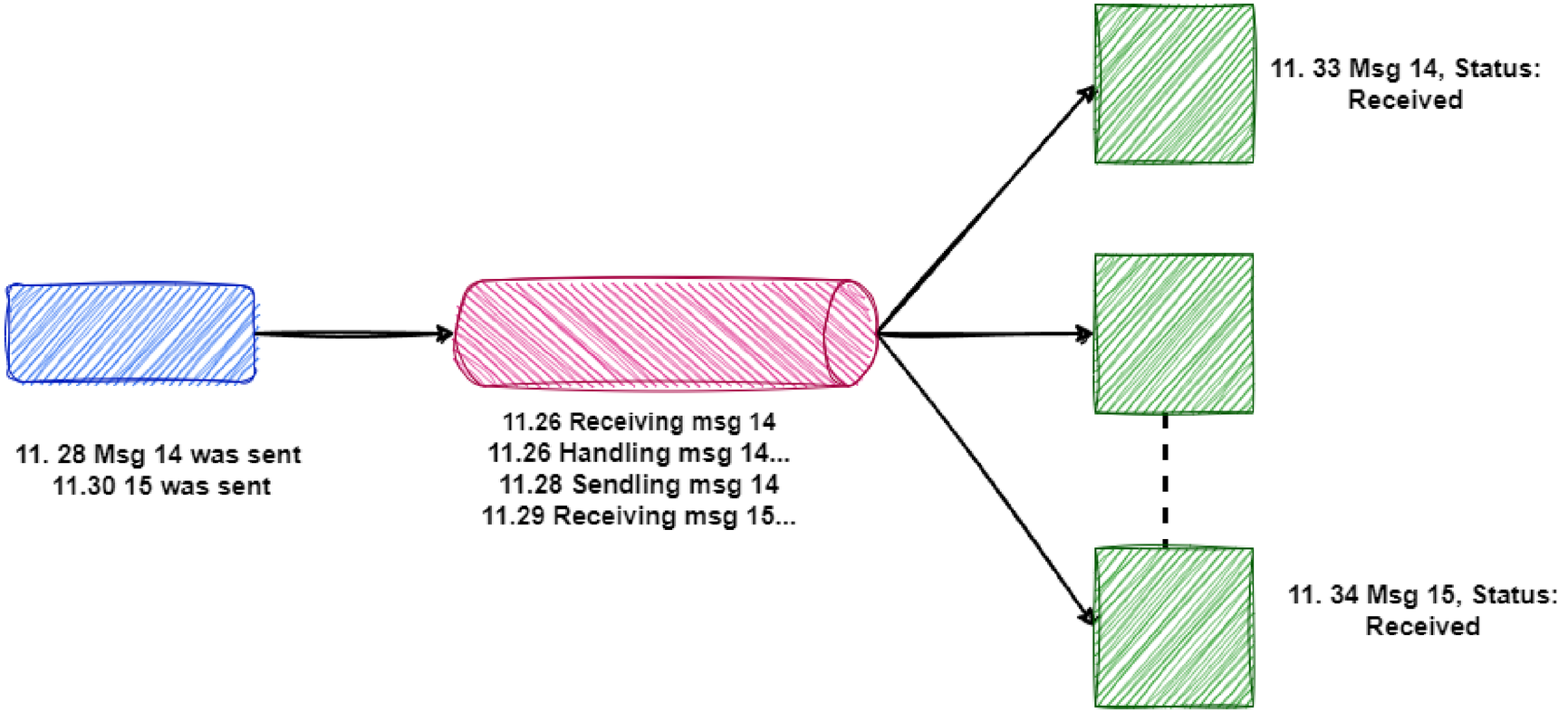


fig 23 Journaux

Different formats

Timestamps not synced

Different destinations

Many records hard to trace

Journalisation et défis dans EDA

- Dans EDA, les composants sont distribués et non couplés. Il est difficile
- d'obtenir un journal unifié et ordonné par ordre chronologique. Chaque
- composant écrit le journal vers une destination différente. Les formats de
- journal sont différents.
- Les horodatages ne sont pas synchronisés

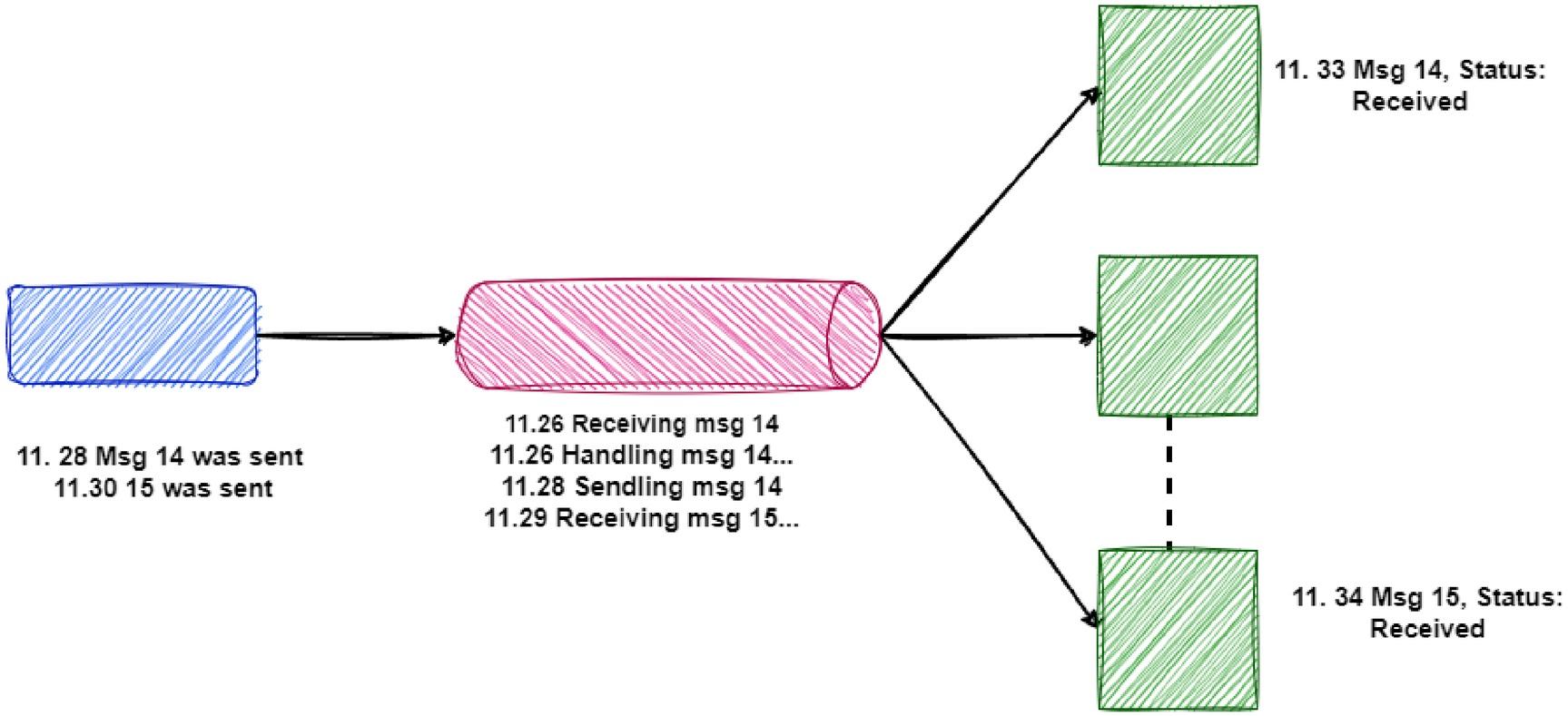


fig 23 Journaux

Different formats

Timestamps not synced

Different destinations

Many records hard to trace

Identifiant de corrélation

- Au début de chaque transaction, un identifiant unique est joint au message. Cet identifiant est
- enregistré dans le cadre de l'enregistrement du journal
- Permet le traçage à travers les composants

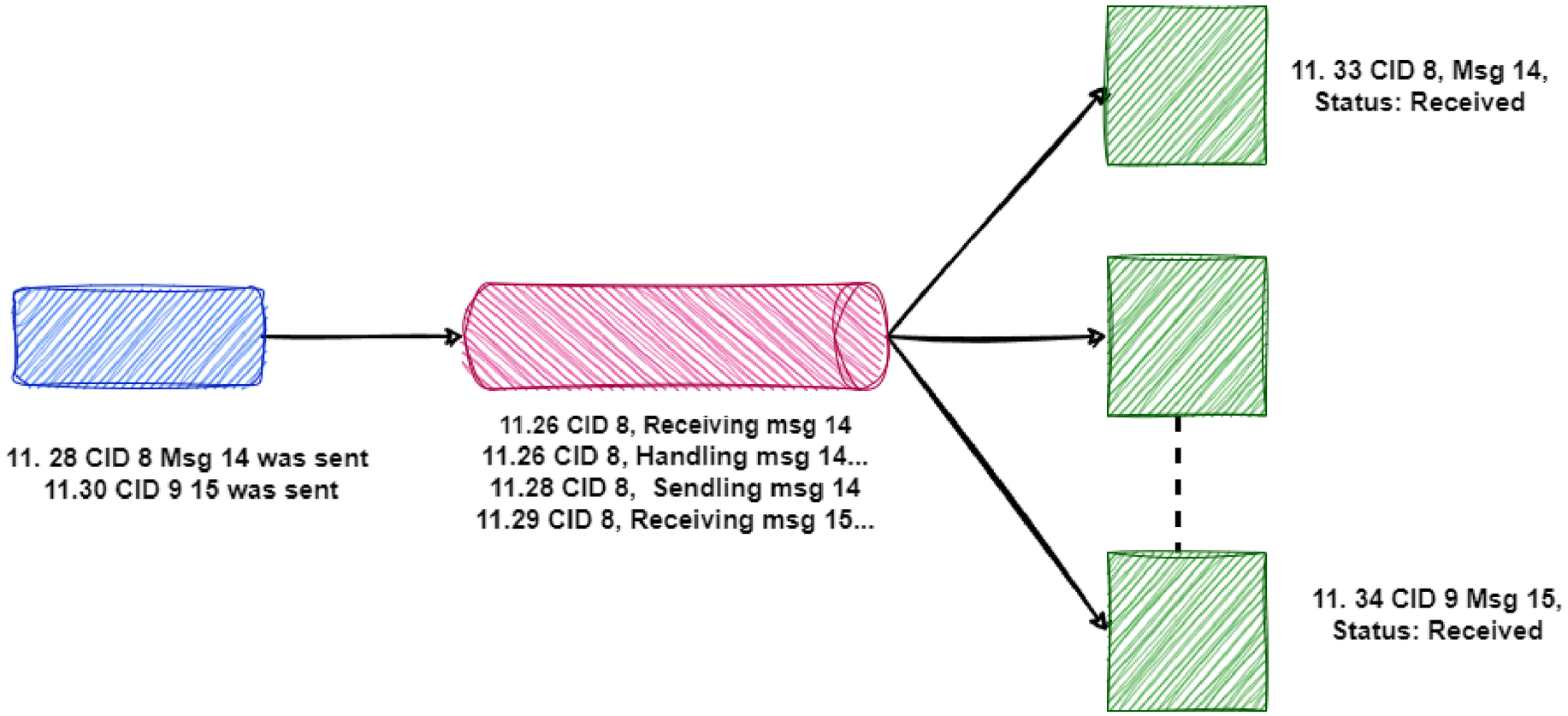


fig 24 ID de corrélation

Moteur de journalisation centralisé

- Le moteur central utilisé pour regrouper tous les enregistrements de journaux possède
- des convertisseurs qui convertissent entre les formats de journaux
- Excellentes capacités d'analyse et de visualisation. Peut
- utiliser le canal existant pour le transport des journaux.

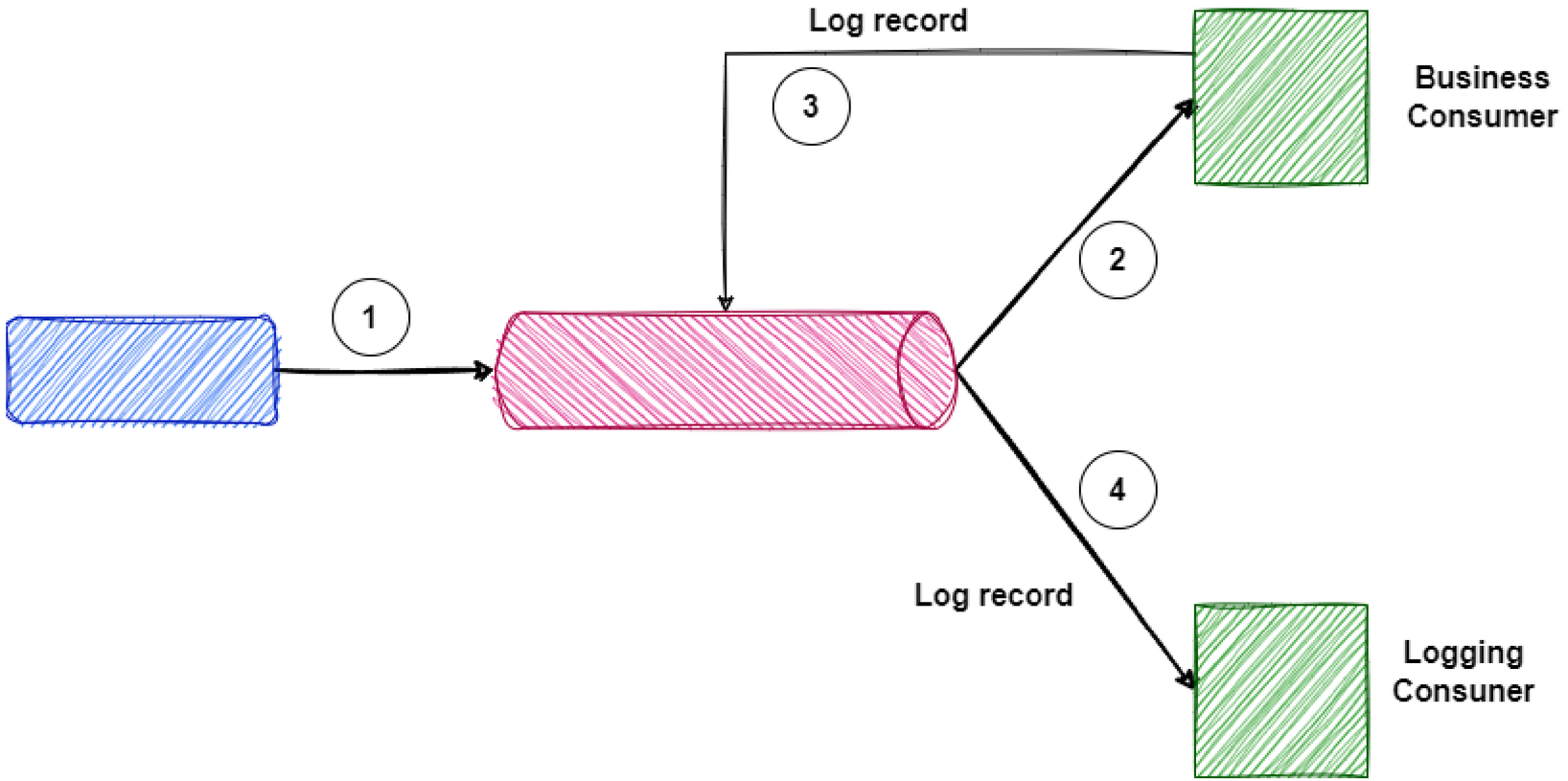


fig 24Journalisation centralisée

Ce qui doit être enregistré

Les journaux doivent refléter le comportement du système

Non juste des erreurs !!

Devrait être capable de rejouer les transactions à l'aide des journaux

Enregistrer:

- Déclencheur d'événement
- Contenu de l'événement (y compris l'identifiant de corrélation)
- Réception de l'événement
- Gestion des événements terminée.
- Toute erreur

Cas de mise en œuvre et d’étude