COURS M2

STREAM PROCESSING

PARTIE 2: GESTION DE DONNEES

21.02.2018

UNIVERSITÉ PARIS-EST

Atos Worldwide IT Partner

OBJECTIFS DU COURS

- Comprendre l'utilité de Kafka
- Appréhender l'écosystème autour de Kafka
- Apprendre l'API Apache Kafka Core
  - Topics, Partitions
  - Brokers, Replication, Zookeeper
  - Producer, Consumer, Consumer groups
- Mettre en place une configuration multi broker sur un environnement de travail
- Apprendre des outils visant à accélerer les développements

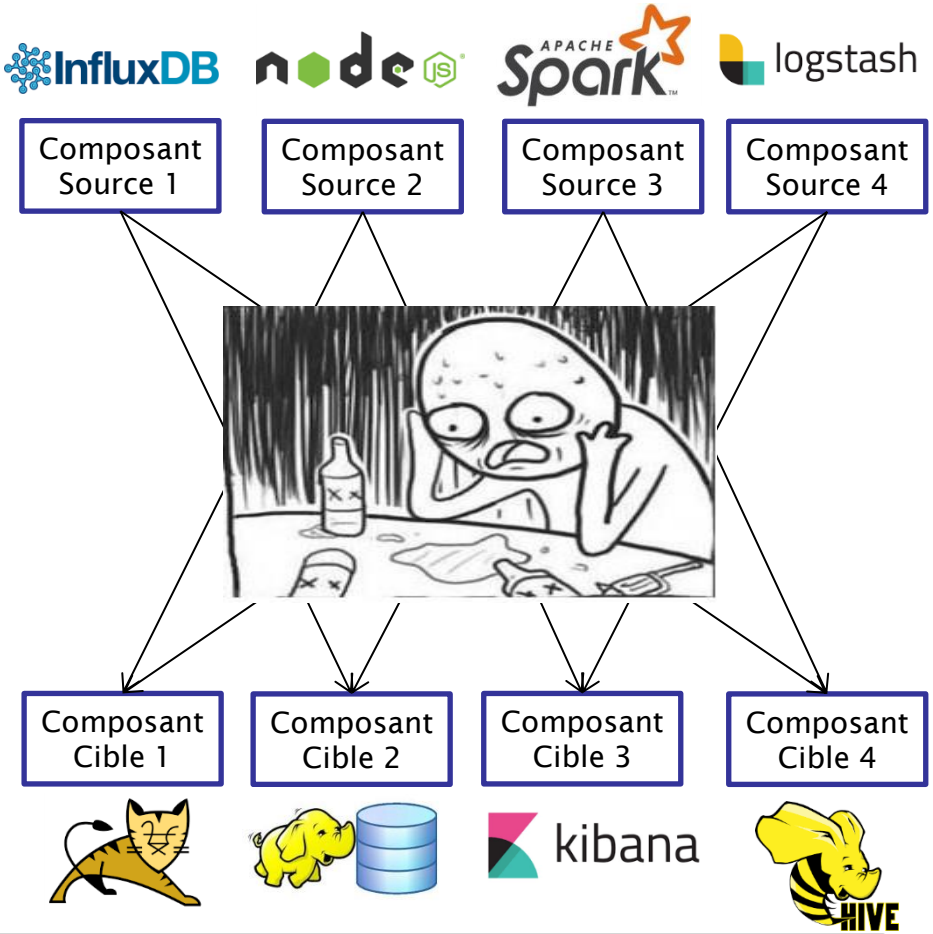
POURQUOI KAFKA ?

- Set up Initial (Chez les bisounours)

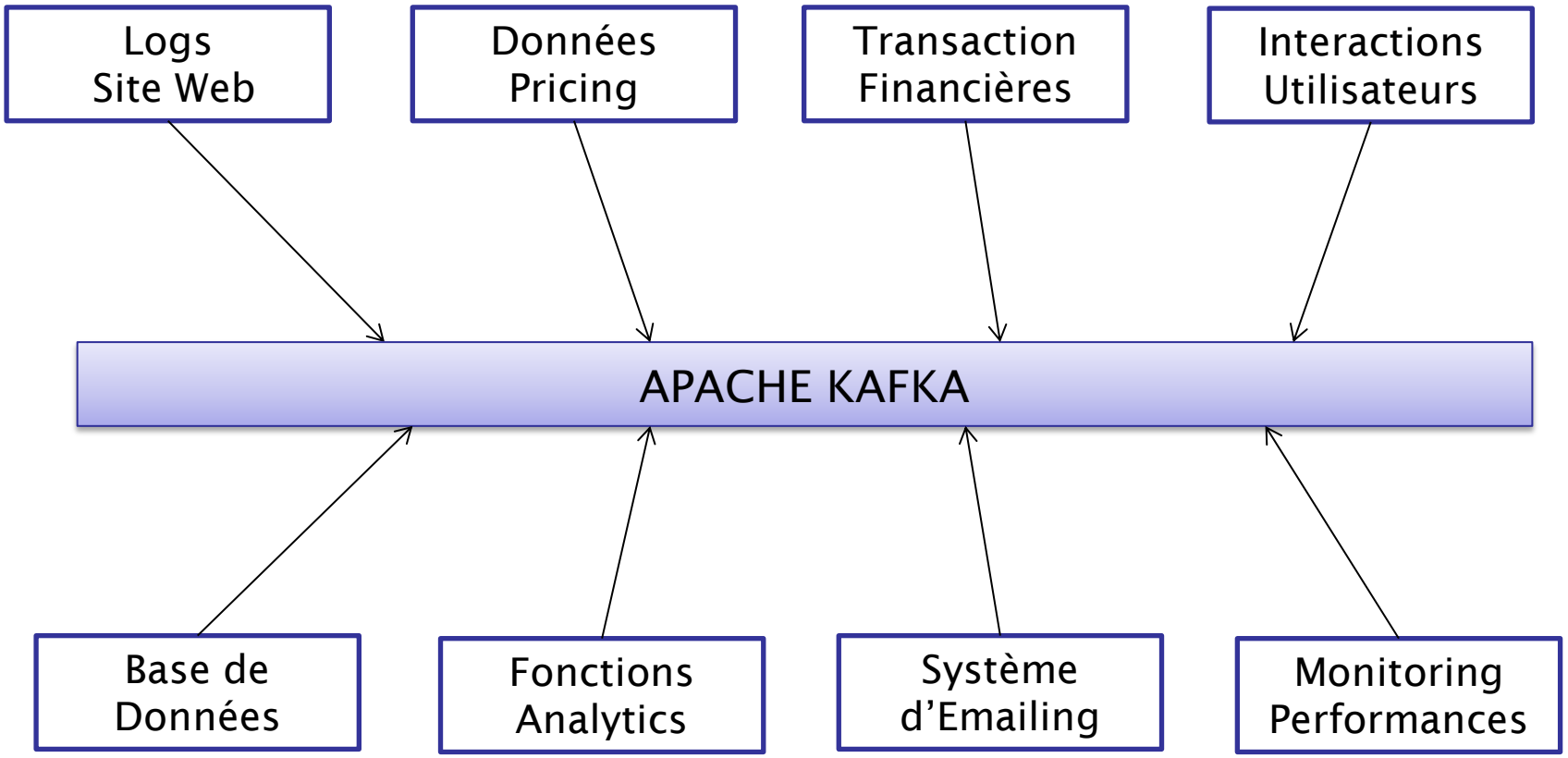
Composant Source

Composant Cible

- Set up Final (La vrai vie)



POURQUOI KAFKA ?



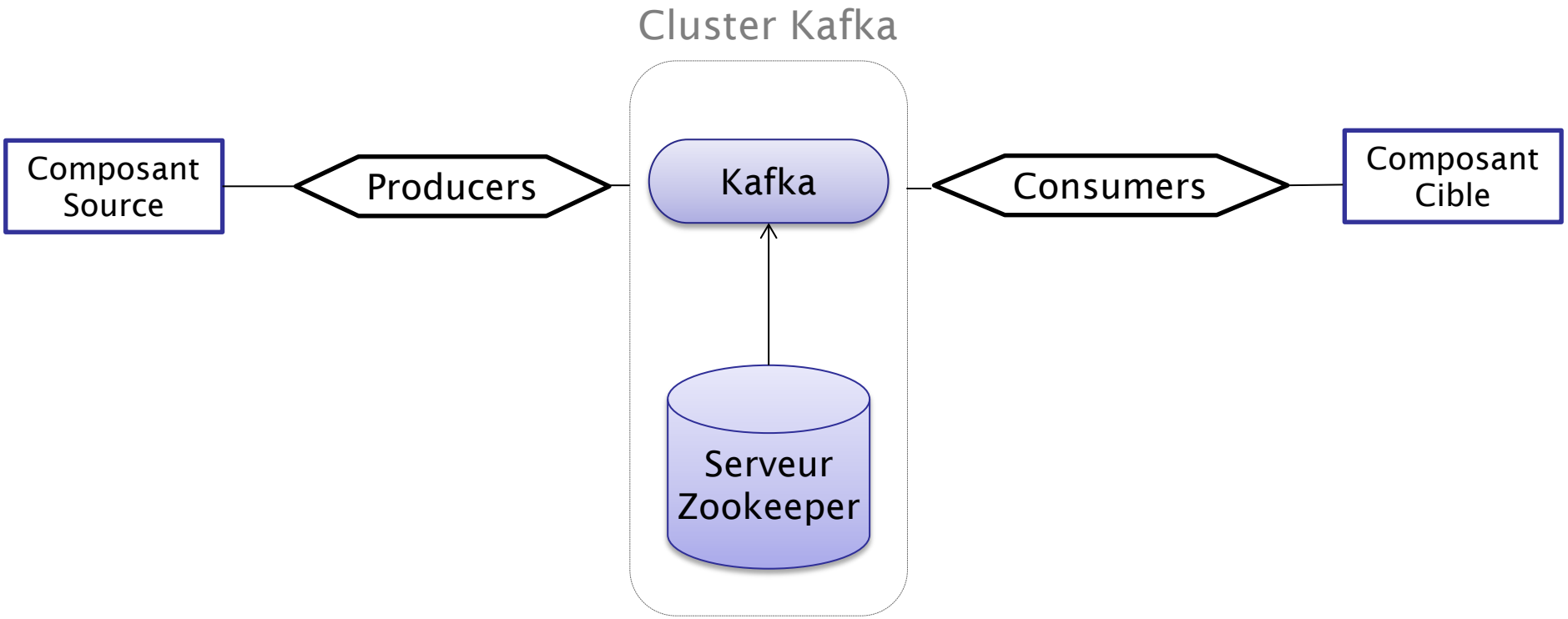
POURQUOI KAFKA ?

- Distribué, architecture résiliente, tolérance aux pannes
- Scalabilité horizontale
- Hautes Performances (latence <10 ms) – Temps réel
- Utilisé par plus de 2000 entreprises:
  - LinkedIn,
  - Netflix,
  - AirBnB,
  - Yahoo,
  - Walmart

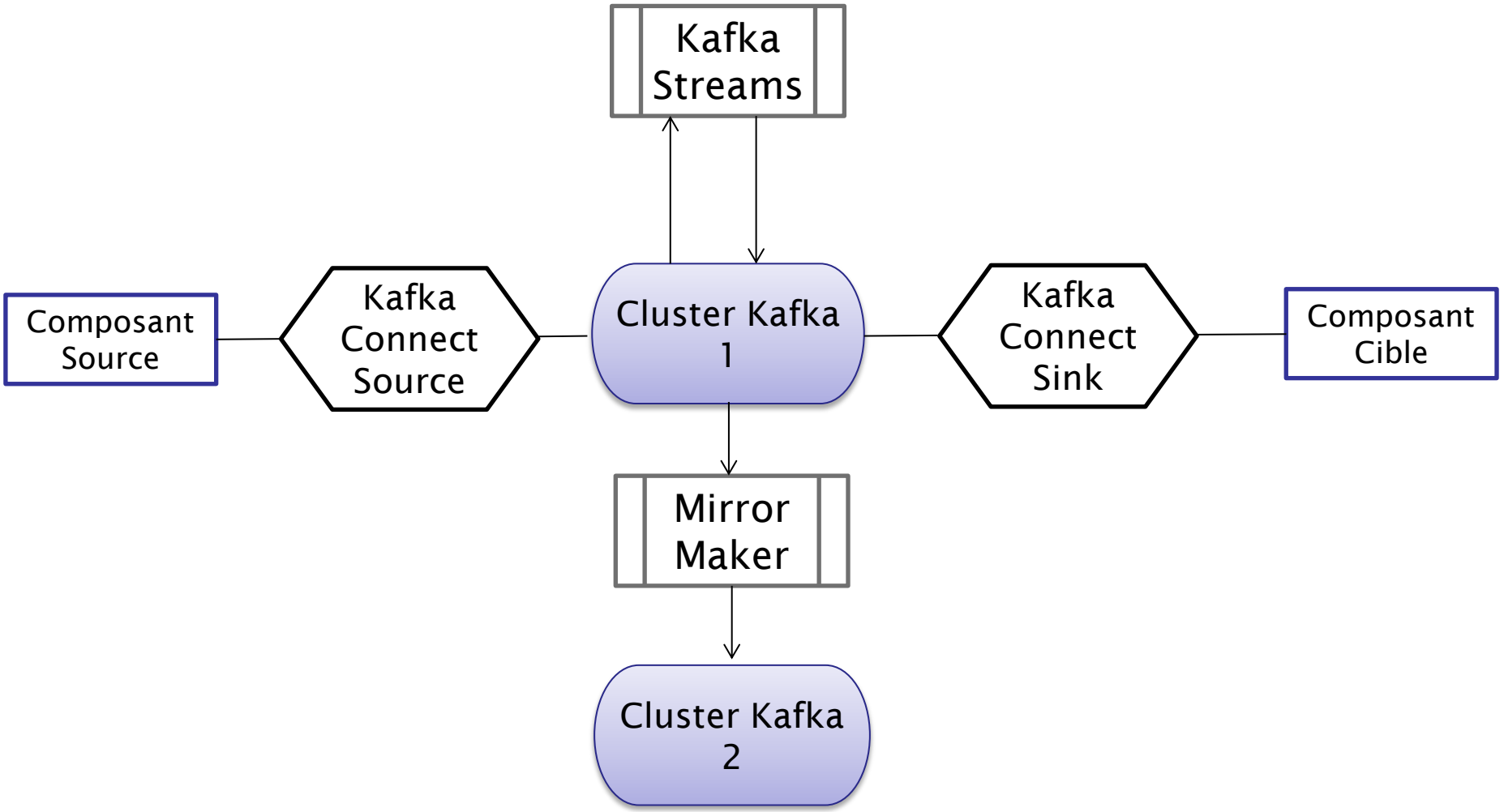
QUAND UTILISER KAFKA ?

- Cas général: Systèmes à messageries denses
- Cas pratiques:
  - Suivi d'activité
  - Ingestion de métriques
  - Récupération de Logs
  - Stream processing
  - Découplage des dependances inter-systèmes
  - Integration (seamless!) avec Spark, Flink Storm, Hadoop et bien d'autres …

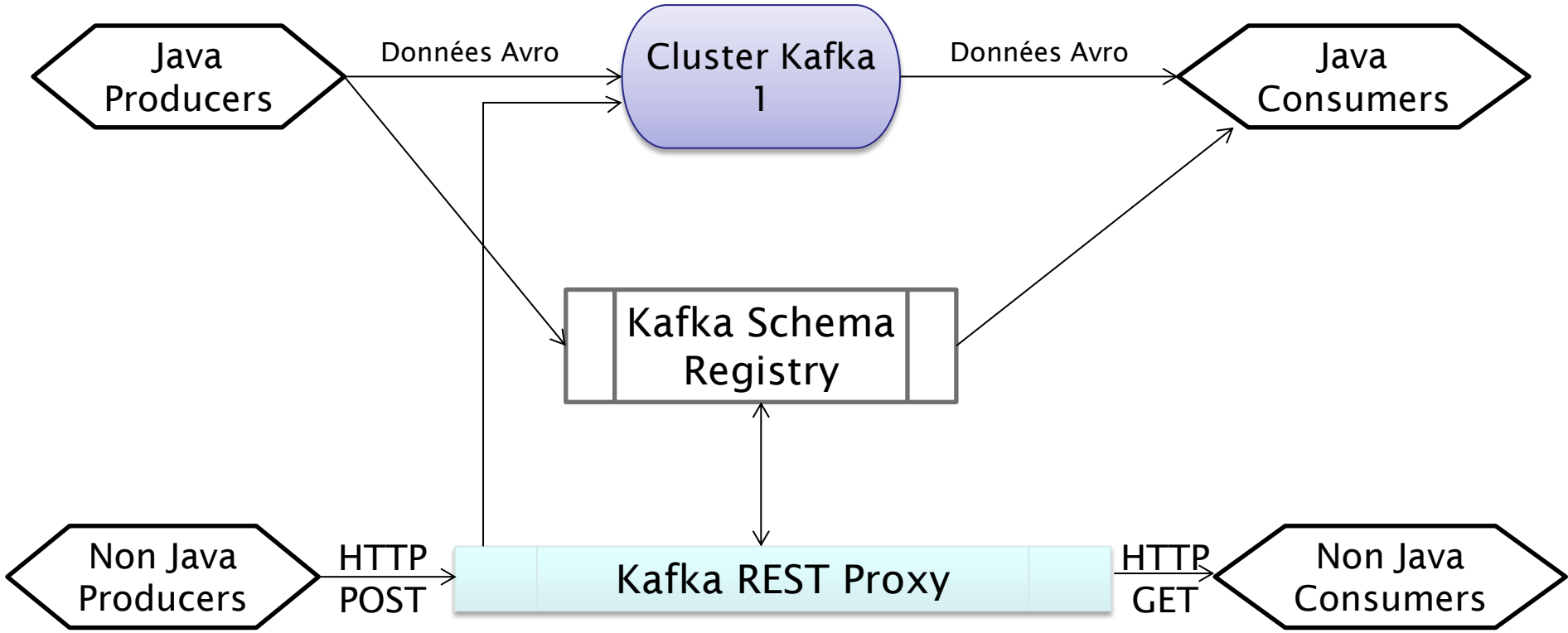
ECO SYSTEME KAFKA
1. Kafka Core



ECO SYSTEME KAFKA
2. Kafka Extended API



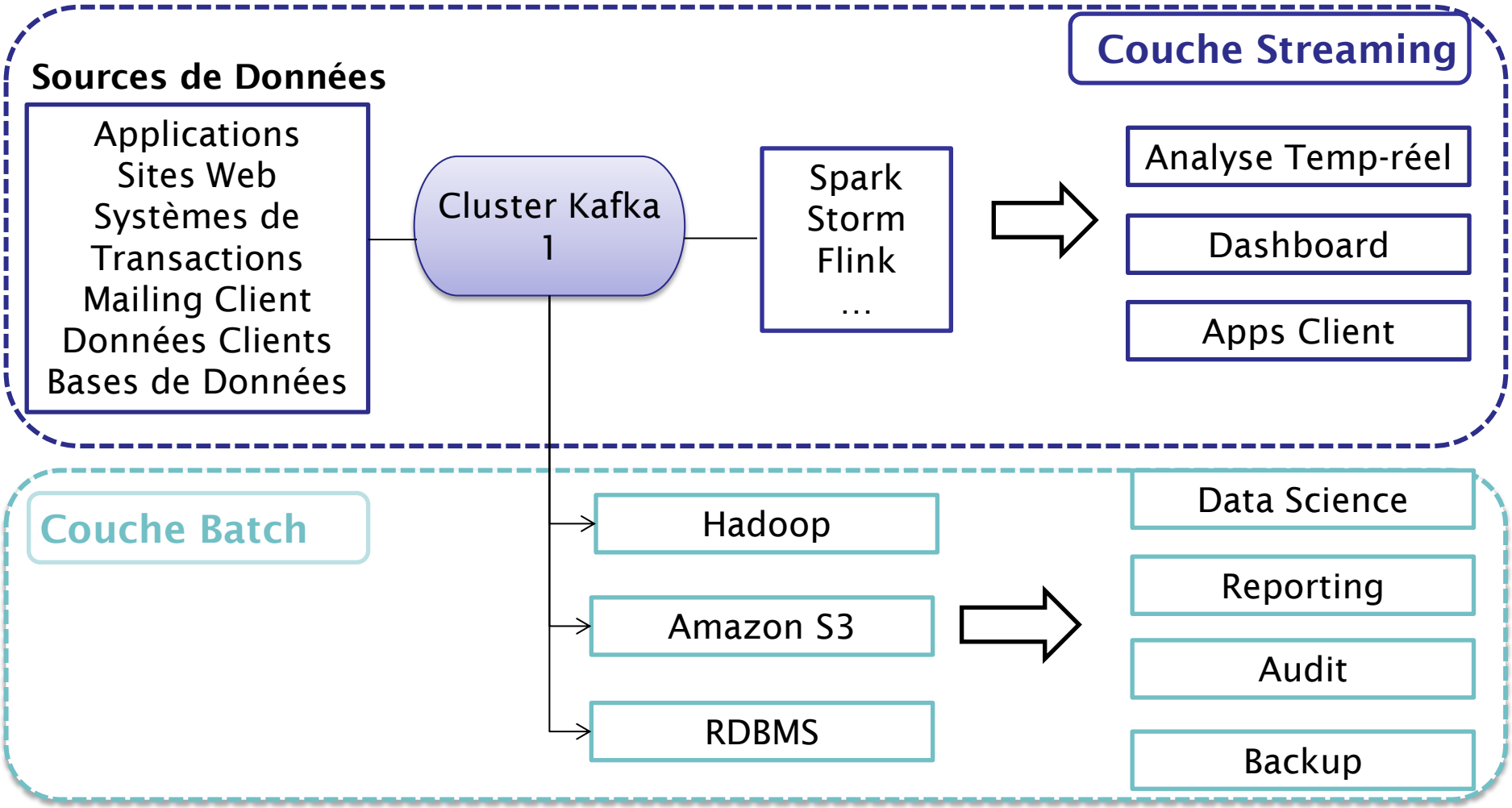
ECO SYSTEME KAFKA
3. Composant Confluents



ECO SYSTEME KAFKA
4. Outils Admin & Monitoring

- **Topics UI** *(Landoop):* Suivre le contenu des topics
- **Schema UI** *(Landoop):* Explorer la Schema Registry
- **Connect UI** *(Landoop):* Création et suivi des tâches Connect
- **Kafka Manager** *(Yahoo):* Gestion globale du cluster
- **Burrow** *(Linkedin):* Vérification des lags Consumers
- **Exhibitor** *(Netflix):* Configuration Zookeeper, Monitoring, Backup
- **Kafka Monitor** *(Linkedin):* Health Check
- **Kafka Tools** (Linkedin): Administration brokers et topics
- ***Kafkat*** *(AirBnB):* Administration brokers et topics
- **JMX Dump** : Métrologie des brokers
- **Control Centre /Auto Data Balancer/Replicator** *(Confluent):* licence payante

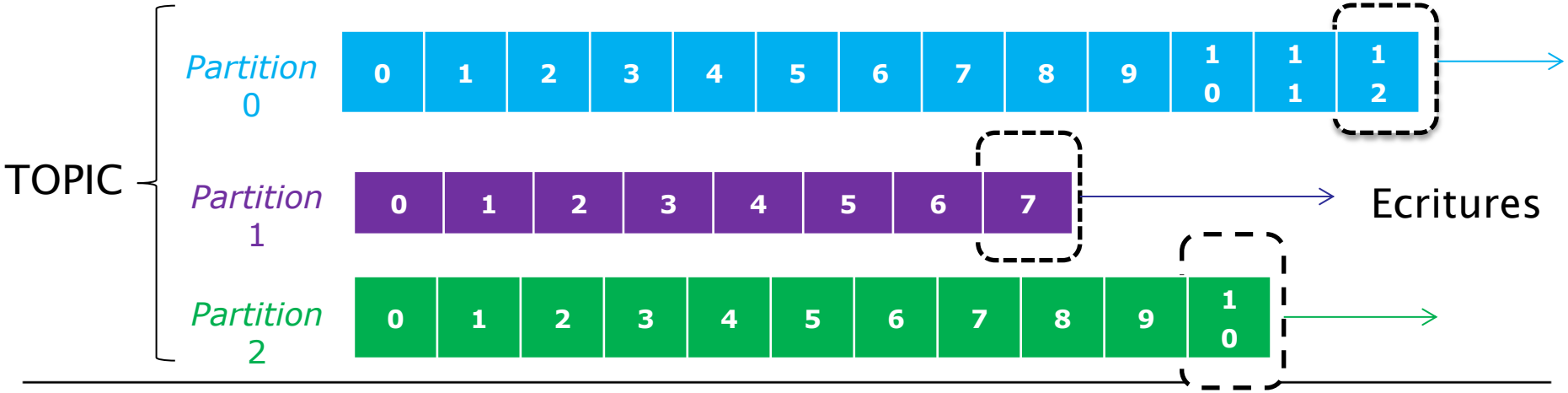
KAFKA DANS LE MONDE PROFESSIONNEL



TOPICS ET PARTITIONS

- Topics: Un flux nommé d'enregistrement
  - Similaire à une table dans une base de données
  - Nombre illimité de topics
  - Un topic est défini par son nom (unique !)

- Les topics sont divisés en partitions:
  - Chaque partition est ordonnée
  - Chaque message dans une partition dispose d'un ID incrémenté (offset)

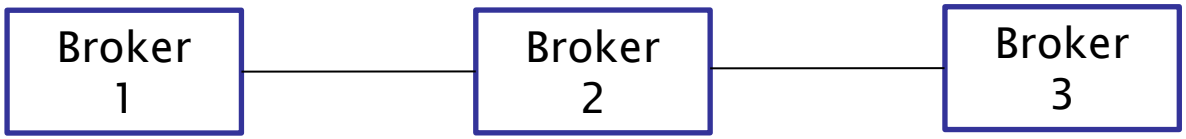


TOPICS ET PARTITIONS

- L'offset n'a de sens que dans le cadre d'une partition spécifique
  - L'offset 22 dans la partition A ne représente pas la même donnée que l'offset 22 dans la partition B
- L'ordre n'est garanti qu'au sein de la parition (pas entre partitions !)
- Quand une donnée est écrite dans une partition, elle ne peut être changée (principe d'immutabilité)
- Les données sont assignées aléatoirement à une partition sauf si une clef est fournie
- Le nombre de partitions par topic n'est pas limité

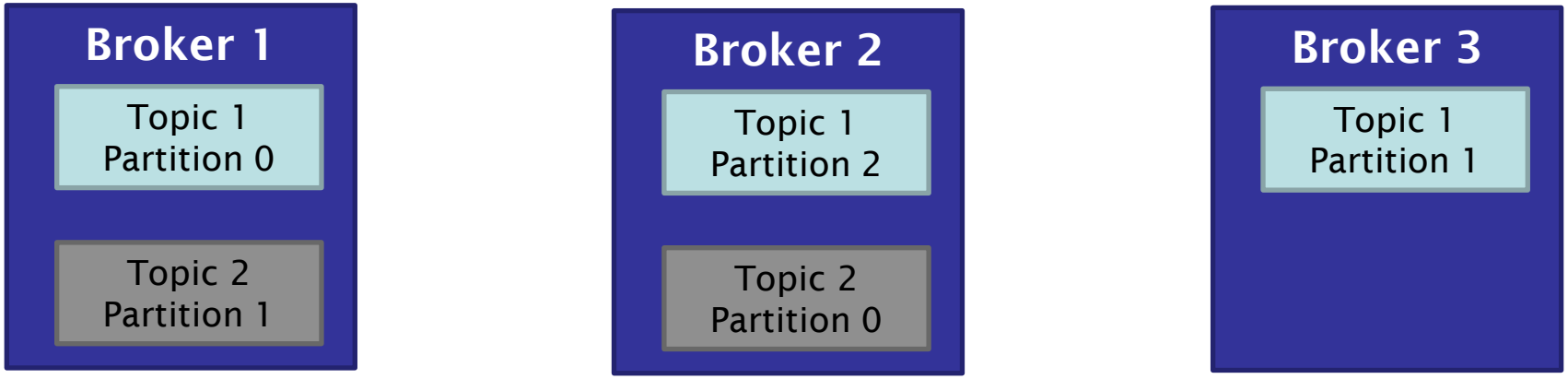
BROKERS

- Un cluster Kafka est composé de plusieurs brokers (servers)
- Chaque broker est identifié par un ID (integer unique)
- Chaque broker contient des partitions de topics donnés
- Lors de la connexion à un broker, on est connectés à l'ensemble du cluster
- Le nombre classique (et minimal) de brokers est estimé à 3 (toujours en nombre impair), certains cluster industriels disposent de plus 100 brokers.



BROKERS ET TOPICS

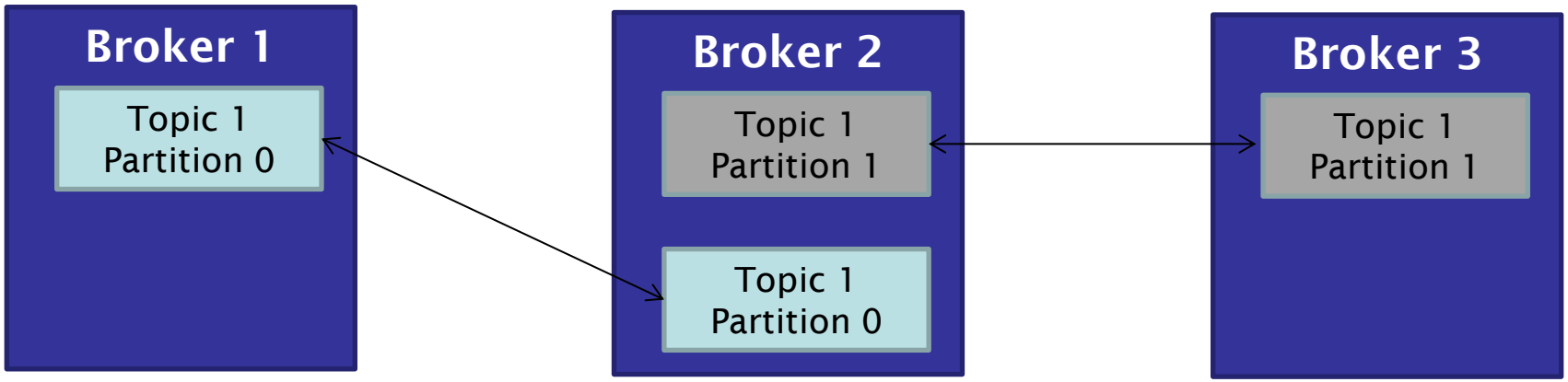
- Exemple de 2 topics (3 partitions / 2 partitions)



- Les données sont distribuées et le broker 3 n'a pas de données pour le topic 2

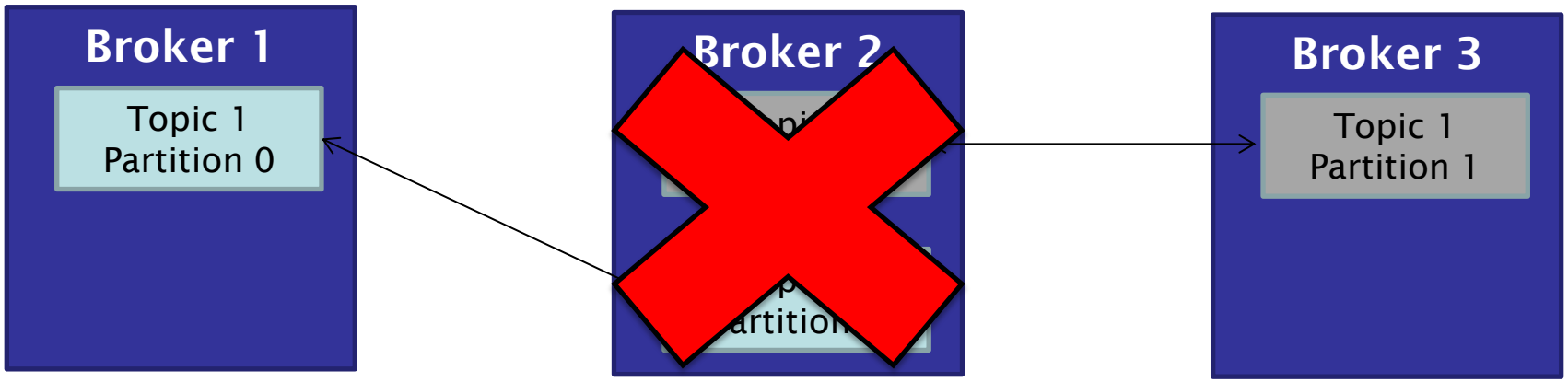
FACTEUR REPLICATION DE TOPICS

- Les topics doivent avoir un facteur de réplication > 1 (généralement entre 2 et 3)
- Si un broker tombe en panne, un autre broker peut servir les données
- **Exemple:** Topic avec 2 partitions et un facteur de replication de 2



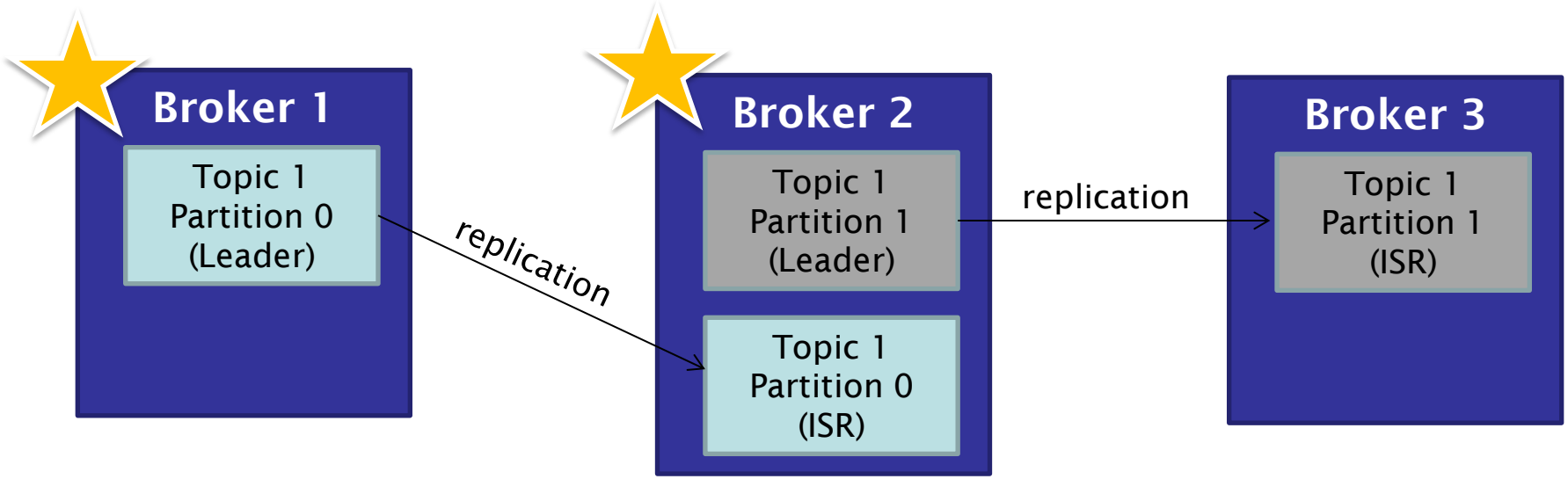
FACTEUR REPLICATION DE TOPICS

- Exemple: le broker 2 tombe en panne
- Résultat: Le brokers 1 et 3 peuvent continuer à fournir les données
- **Exemple:** Topic avec 2 partitions et un facteur de replication de 2



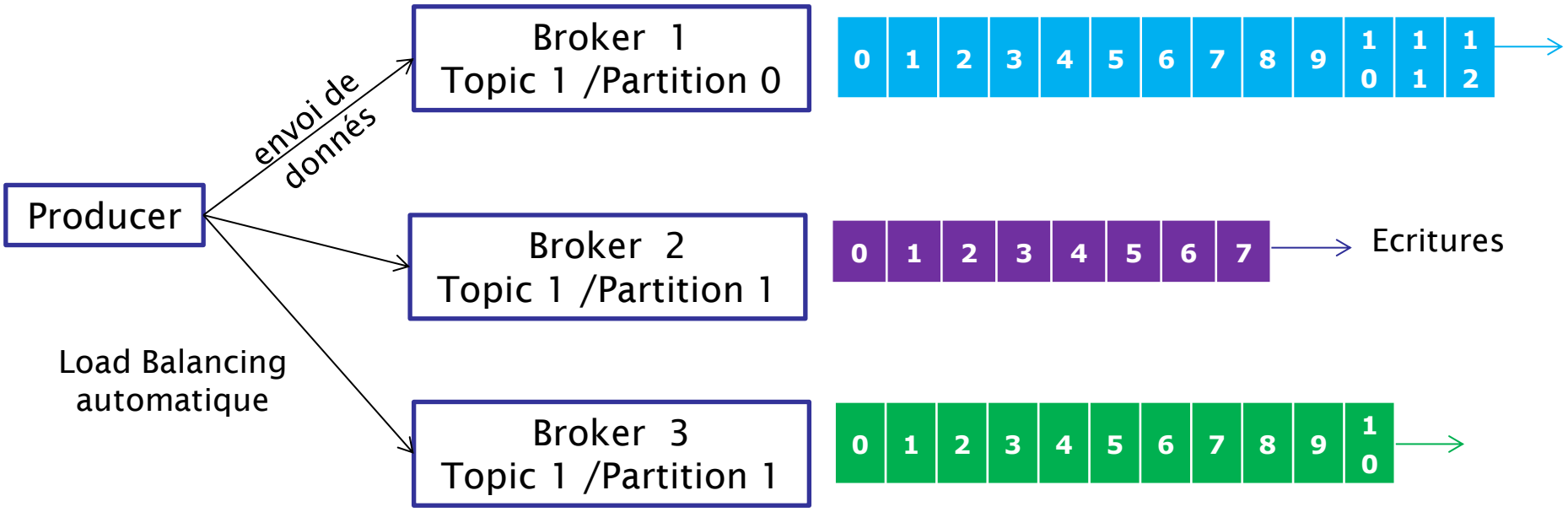
LEADER DE PARTITION

- Durant un run donné, seul un broker peut être leader pour une partition
- Seul ce leader peut recevoir et fournir les données pour une partition
- Les autres brokers s'occupent de synchroniser les données
- Conséquence: Chaque partition dispose d'un leader et de multiple ISR (in-sync replica)



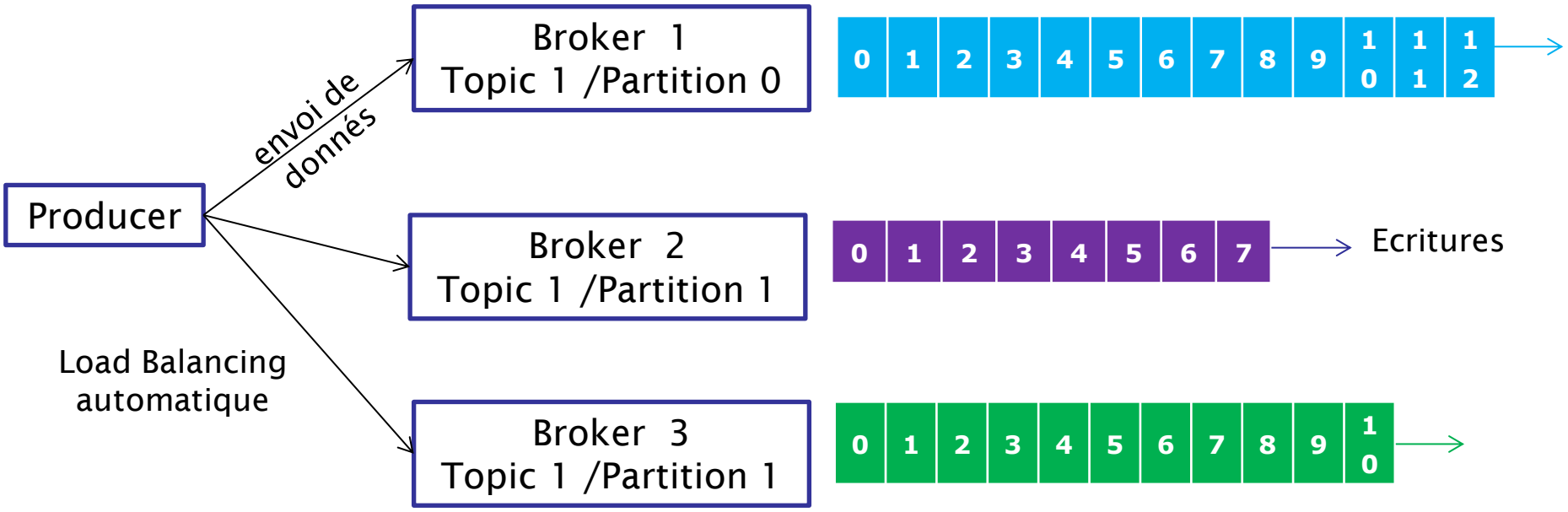
PRODUCERS

- Les producers écrivent les données sur un/des topic
- Ils doivent spécifier un nom de topic et un broker auquel se connecter, Kafka s'occupe nativement du routage des données au bon endroit.



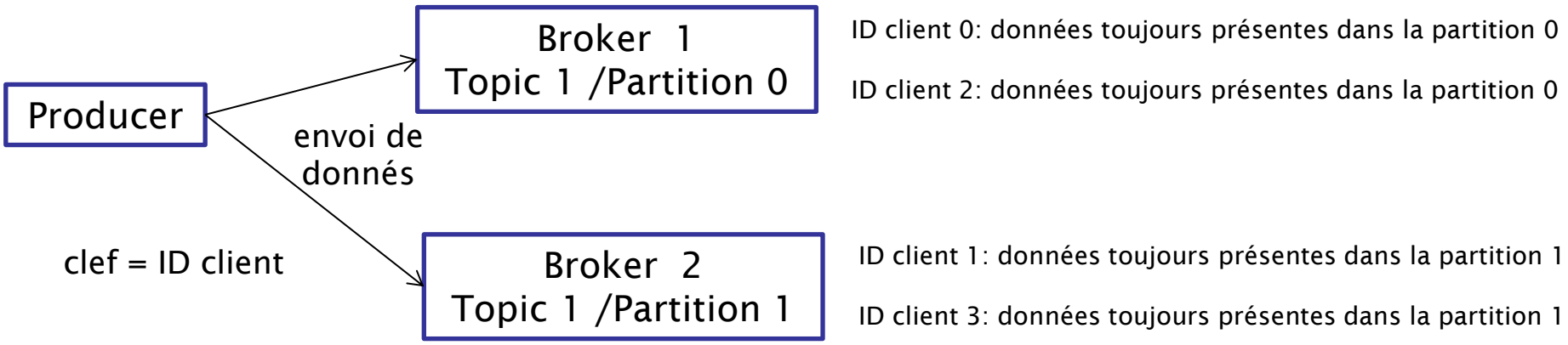
ACKNOWLEDGEMENTS

- Les producers peuvent demander la confirmation des écritures de données *(acknowledgment)*
  - **Acks = 0:** Les producers ne vont pas attendre de confirmation (possibilité de pertes de données)
  - **Acks = 1:** Les producers vont attendre la confirmation du leader (limitation des pertes de données)
  - **Acks = all**: Confirmation du leader + replicas (aucune perte de données)



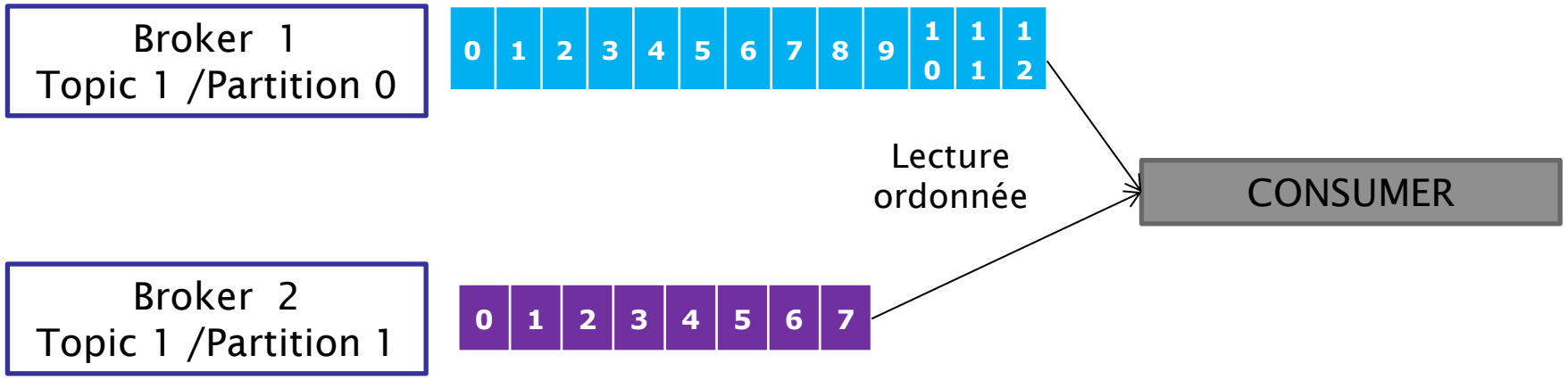
CLEFS DE MESSAGES

- Les producers peuvent envoyer une clef avec le message
- Si une clef est envoyée, le producer a alors la garantie que tous les messages pour cette clef iront toujours à la même partition
- Cette procédure garantit l'ordre (ordering) pour une clef données



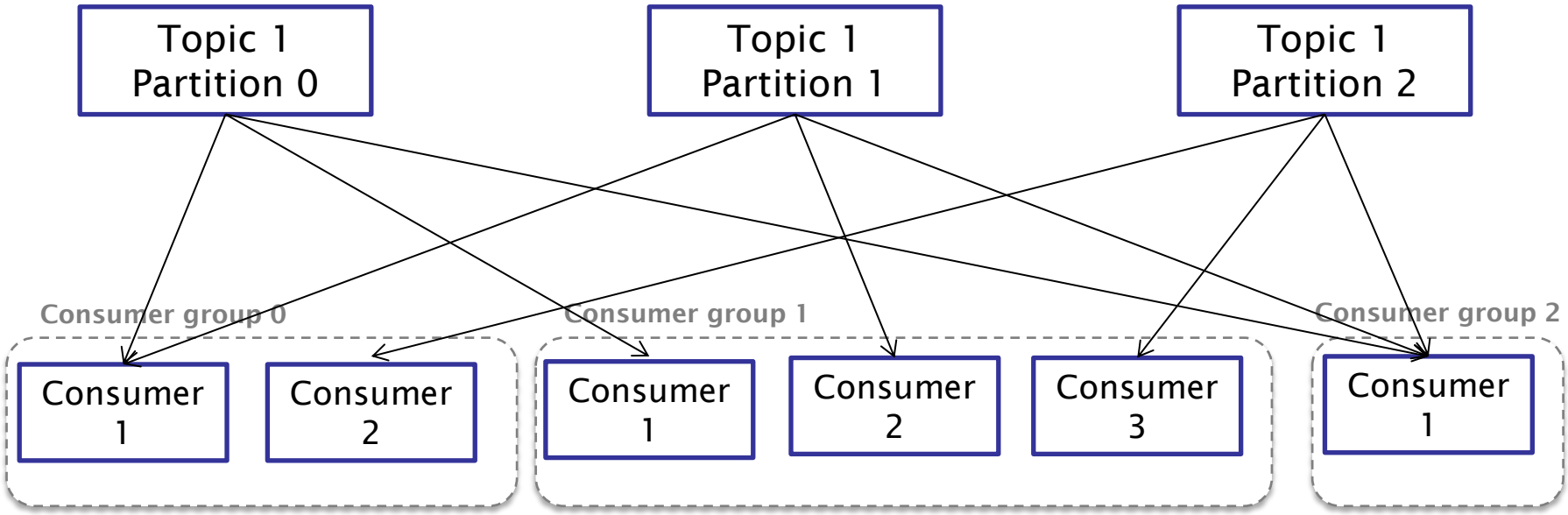
CONSUMERS

- Les consumers lisent les données à partir d'un topic donné
- Ils doivent spécifier un nom de topic, et un broker auquel se connecter. Kafka s'occupe nativement d'extraire les données du bon broker.
- La lecture des données se fait de manière ordonnée dans chaque partition mais en parallèle entre les partitions.



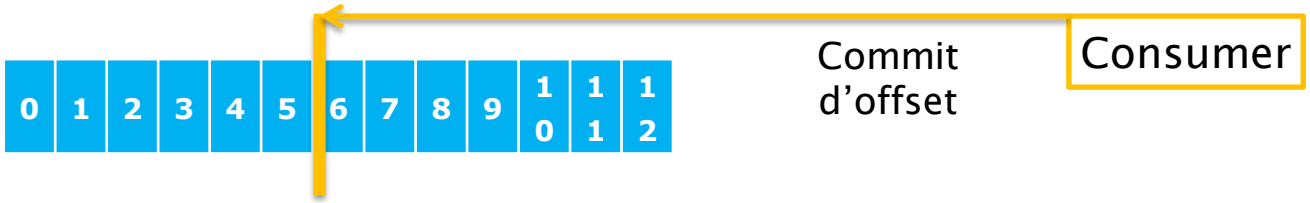
CONSUMERS GROUPS

- Les consumers lisent les données par groupe
- Chaque consumer au sein d'un groupe lit les données exclusivement d'une partition
- Il n'est pas possible d'avoir plus de consumers que de partitions (sinon certains seraient inactifs)



CONSUMERS OFFSETS

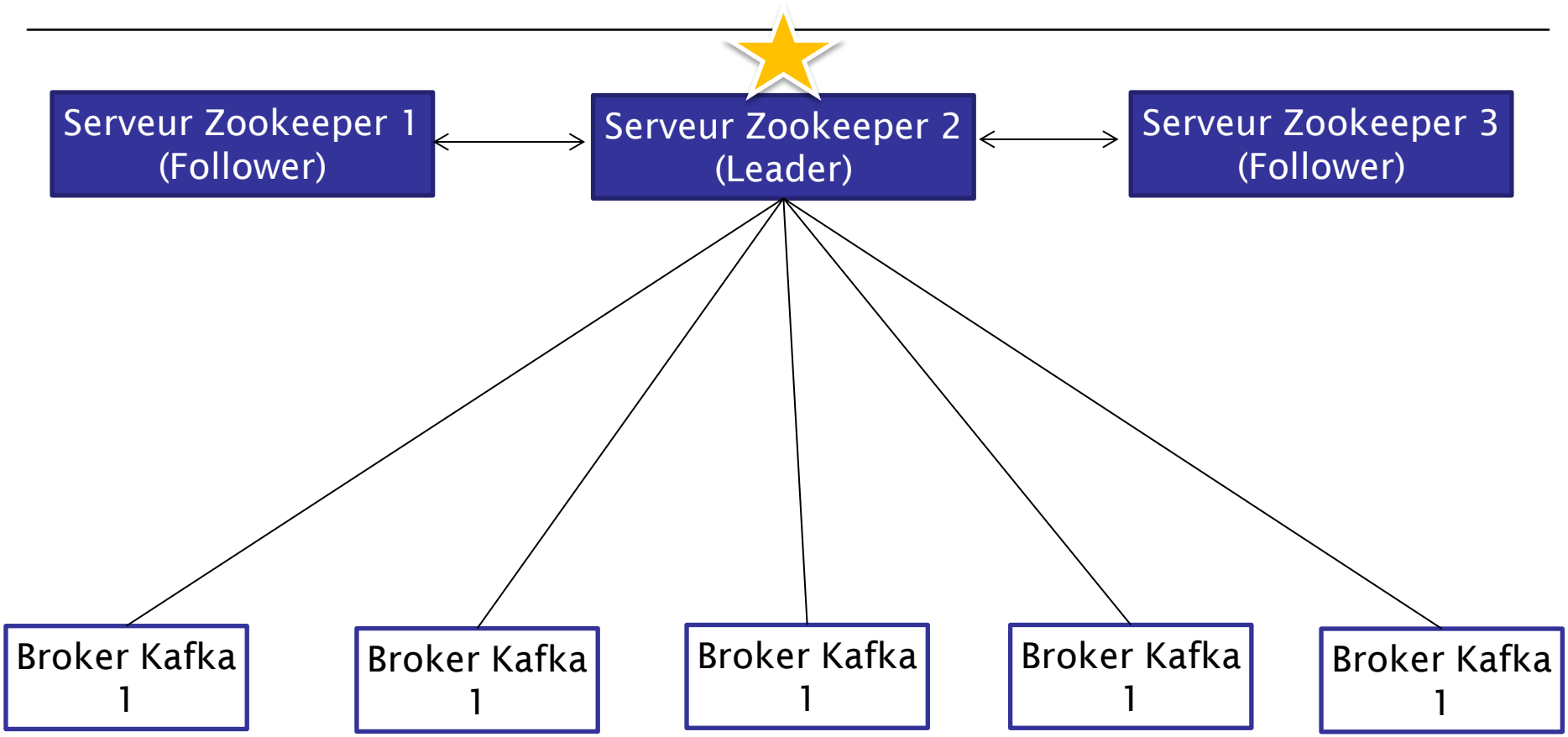
- Kafka stocke les offsets auquels un consumer group s'est abonné pour ses lectures
- Les commtis d'offset sont stockés dans un topic « __consumer-offsets »
- Quand un consumer a traité les données sur un certain broker, il doit commité ses offsets
- Si un processus de consommation tombe, il pourra relire à l'endroit exact où il est tombé en panne.



ZOOKEEPER

- Server de gestion des brokers (sauvegarde une liste des brokers)
- Il permet l'election d'un leader pour les partitions
- Il envoie des notifications à Kafka lors de changements d'état (e.g nouveau topic, un broker qui tombe, un topic supprimé, etc)
- Kafka ne peut fonctionner sans Zookeeper
- Zookeeper opère en « odd quorum » (cluster de server de 3, 5, 7, etc)
- Zookeeper dipose également d'un leader, les autres serveurs sont des « followers »

ZOOKEEPER


Serveur Zookeeper 1 (Follower)
Serveur Zookeeper 2 (Leader)
Serveur Zookeeper 3 (Follower)
Broker Kafka 1
Broker Kafka 1
Broker Kafka 1
Broker Kafka 1
Broker Kafka 1

SEMANTIQUE LIVRAISON DE DONNEES

- Rappel: les consumers choisissent quand ils commitent les offsets
- **At most once:** les offsets sont commités dès que le message est reçu. Si le traitement ne se passe pas comm prévu, le message sera perdu (il ne sera pas lu à nouveau)
- **At least once:** Les offsets sont commités après que le message soit traité. Si le traitement ne se passe pas comme prévu, le message sera relu à nouveau. Cela peut entraîner des doublons de traitement. **S'assurer que le traitement reste idempotent est primordial.**
- **Exactly once:**Très compliqué à atteindre, requiert une ingénierie poussée.
- **Conséquence:** Généralement, la cible reste « at least once » en s'assurant de l'idempotence.