ESILV
ENGINEERING SCHOOL
DE VINCI PARIS

IoT - Cloud

Thème 8 - Stream

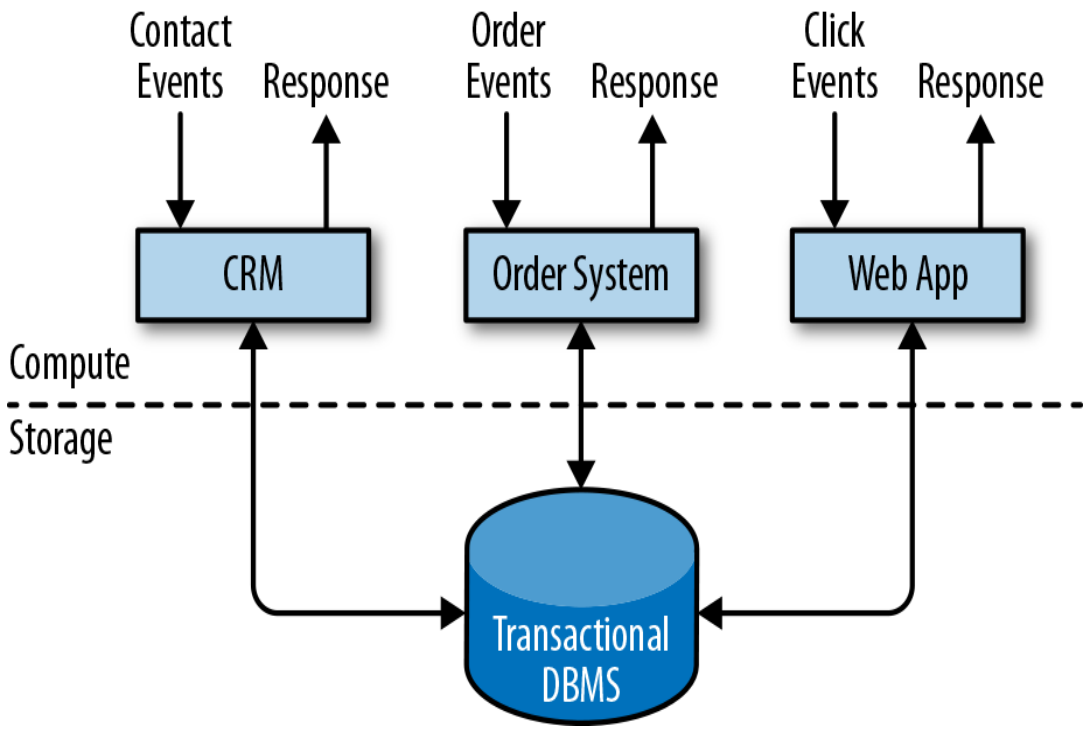
Mary DAL ZUFFO - Gaetan FOURNIES
Mathieu GALIDIE - Jérémie BENCINI

ESILV
ENGINEERING SCHOOL
DE VINCI PARIS

Plan du cours

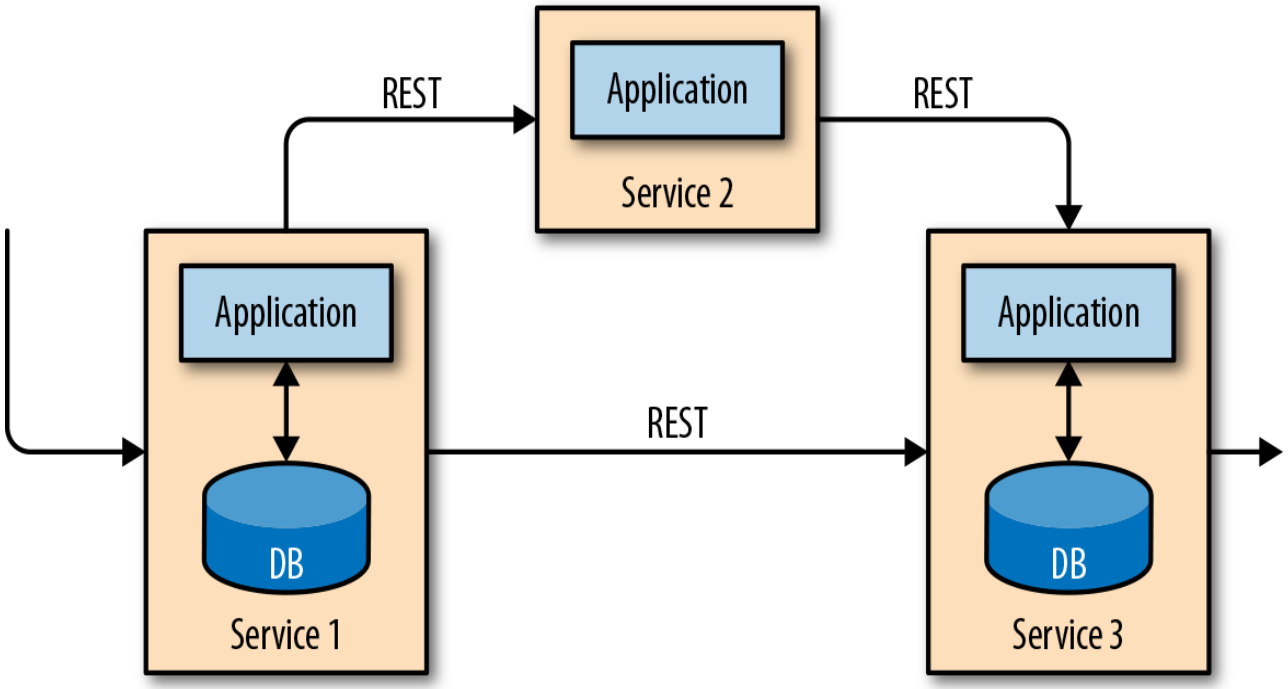
1. **Introduction au Stream**
   - Les infrastructures traditionnelles
   - Le Stream Processing
2. Stream/Batch
   - Batch
   - Comparaison avec le Batch
   - Cas d'usages (particularités du Stream : fenêtre de calcul) Avantages/inconvénients
3. Les différents outils du Stream
   - Flink / Amazon / Google
   - Comment choisir ?
   - Tableau comparatif
4. Focus sur Flink avec une démo

Infrastructures traditionnelles : Transactionnal Processing



- Les applications sont connectées à des services externes et analyse continuellement des informations.
- Quand un évènement est analysé l'application interagit avec le système de stockage
- Un même DBMS sert plusieurs applications
- Problématique pour l'évolution et la scalability => Microservice

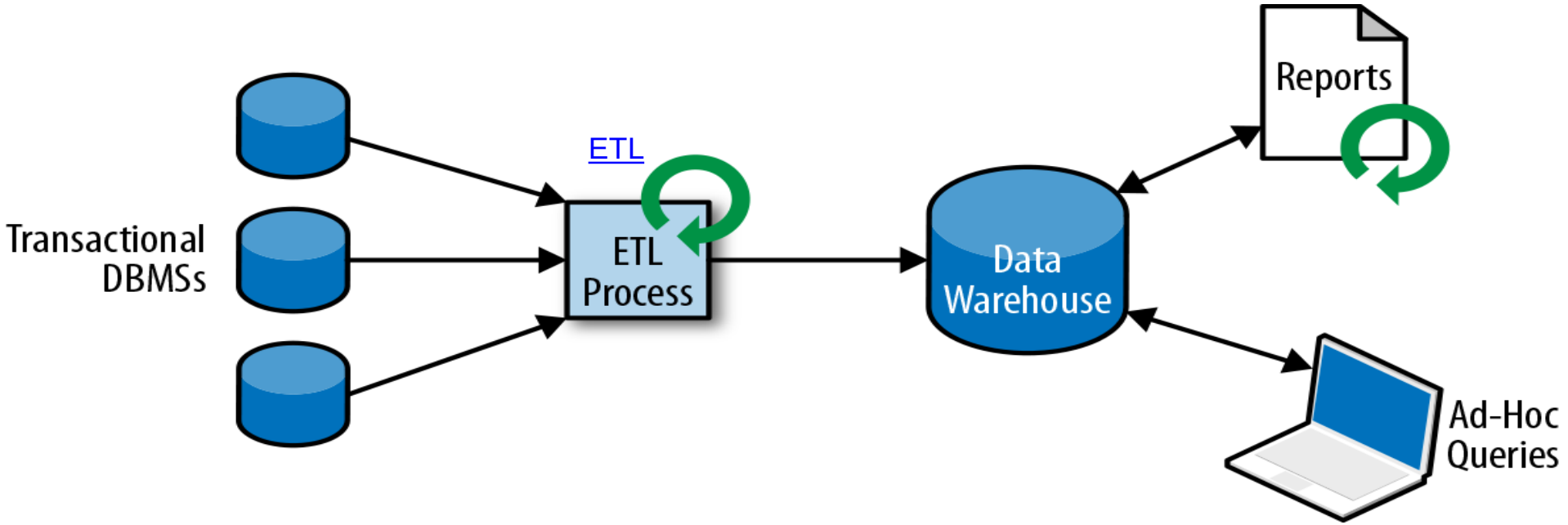
Infrastructures traditionnelles : Transactionnal Processing



- Les **microservices** sont de petites applications indépendantes
- On construit des applications plus complexes en interconnectant plusieurs microservices
- Ils sont déployés sur différents conteneurs et possèdent chacun leur propre technologie

Quelques rappels sur
RESTful

Infrastructures traditionnelles : Analytical Processing



- Les données sont stockées dans différentes bases de données transactionnelles
- Les données sont répliquées dans la Data Warehouse où elles seront requêtées (Batch)
- 2 types de requêtes :
  - Rapports périodiques
  - Requêtes ad hoc

Pour plus d'infos sur les structures traditionnelles, c'est par ici

Le Stream Processing : Stateless/Statefull

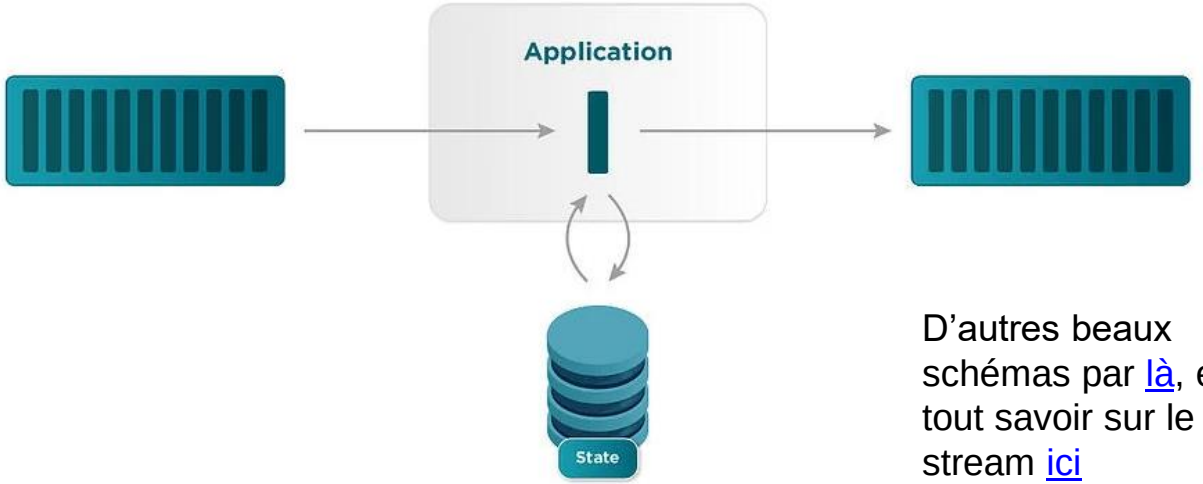
Stateless Stream Processing

La façon dont est analysé chaque évènement est indépendante des évènements précédents

Stateful Stream Processing

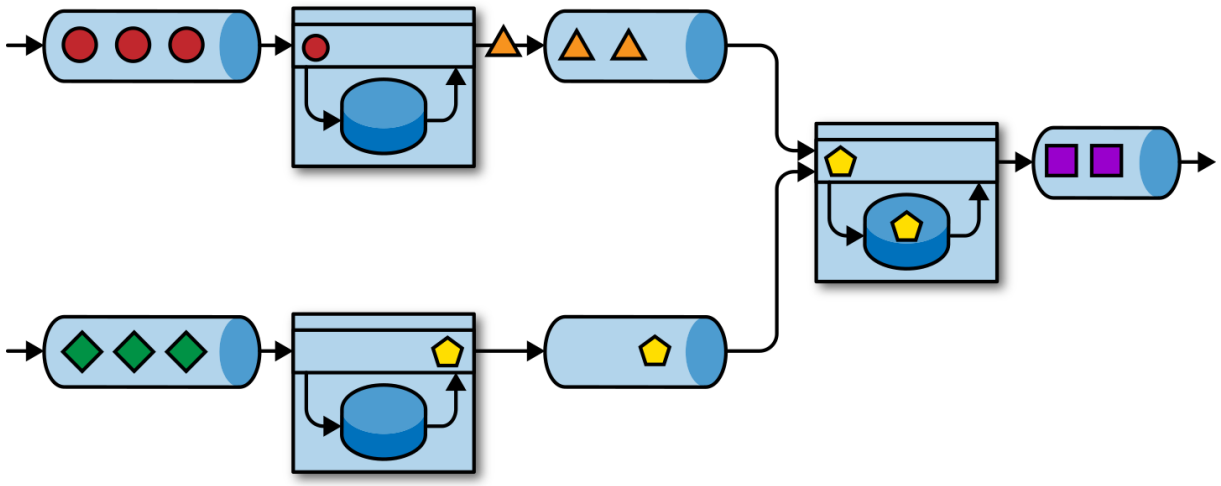
Un *state/contexte* est conservé entre les évènements. Les évènements précédents peuvent influer sur l'analyse des suivants.

State : information conservée pour un usage futur, un contexte



D'autres beaux schémas par là, et pour tout savoir sur le stateful stream ici

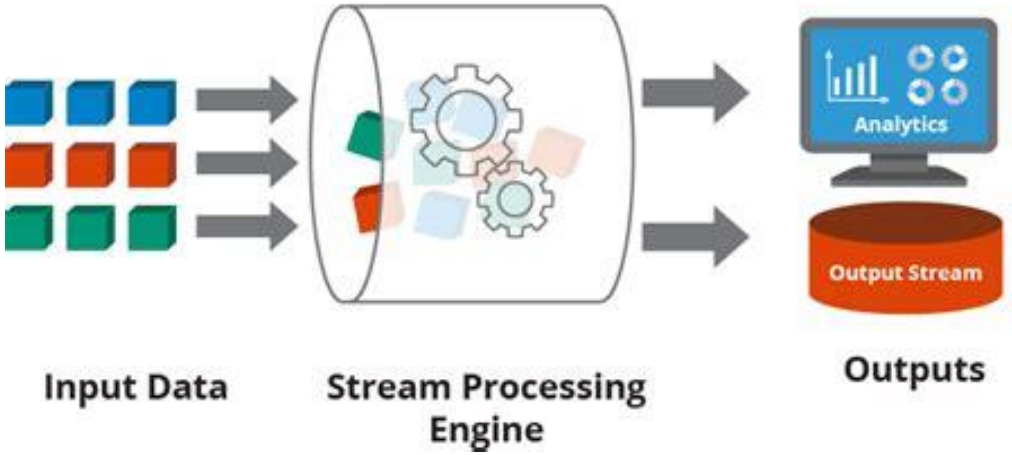
Le Stream Processing : Event-Driven application



Event logs = un broker

- Evolution des microservices
- Les applications communiquent entre elles via des *event Logs*
- Stockage dans des local *state*
- Chaque application est stateful et gère localement son state
- Une application peut être scale localement

Le Stream Processing



“ Stream processing is a programming paradigm defining applications which, when receiving a sequence of data, treat it as a **collection of elements**, or datapoints, and rather than group and process them together, **process each datapoint by itself** ”

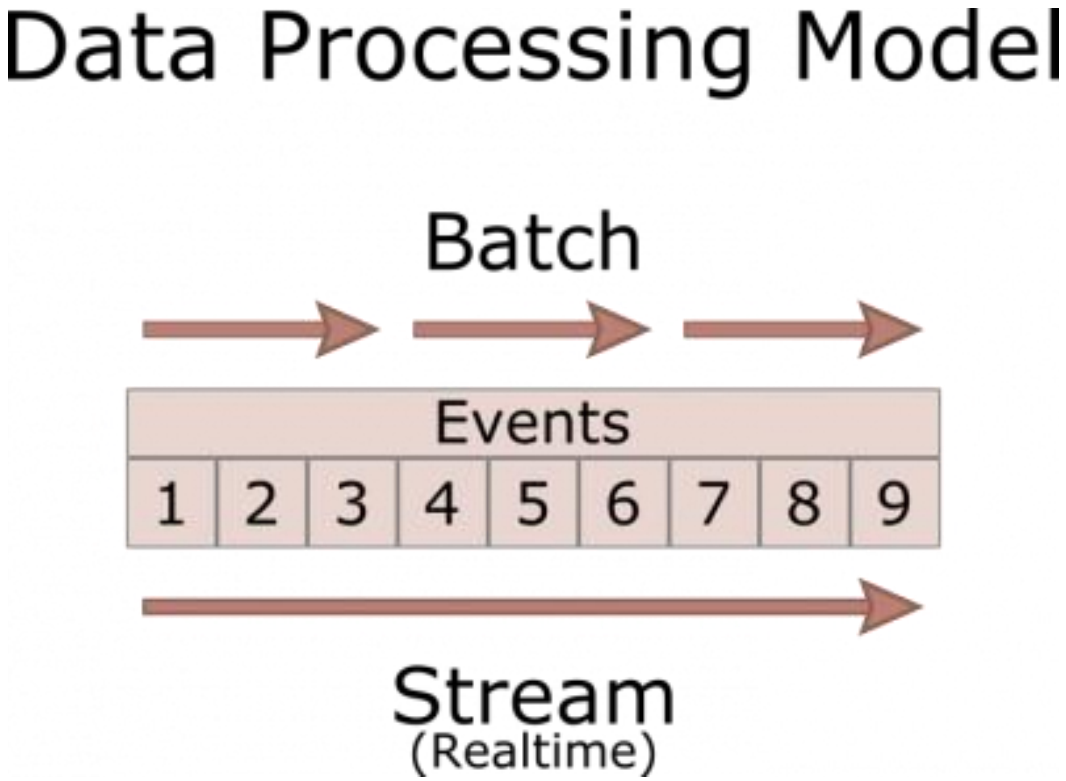
- Les applications réagissent aux évènements instantanément
- Permet de gérer un grand volume de données
- Adapté aux time series et aux données continues
- Décentralise l’infrastructure

Batch ou Stream ?

La définition de nos **besoins** définit quelle méthode va être utilisée.

Le **Batch** permet de traiter une **large quantité de données**, l'architecture est simple à mettre en place.
Ce système comporte une **grande latence** des données.

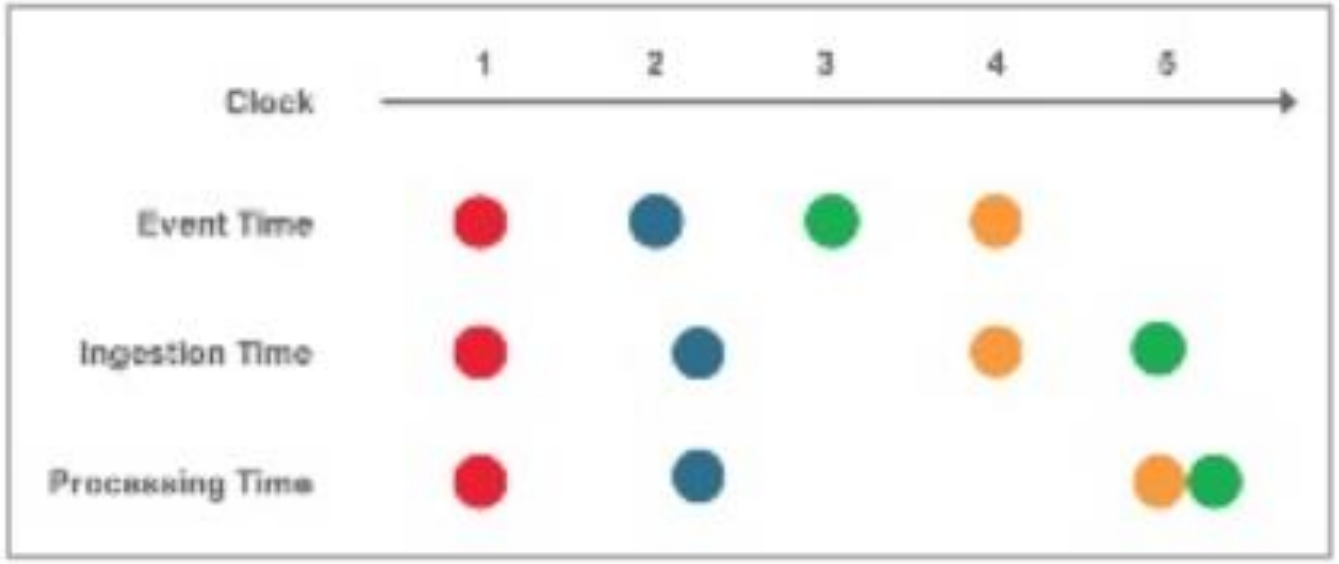
Le **Stream** traite les données en **temps-réel**, l'architecture est plus complexe mais permet d'avoir une **faible latence** et donc un traitement en live.



Le temps

Le **timestamp** définit la variable de **temps** d'une donnée.

- **Event time** : définit par l'évènement, dépendant de la donnée.
- **Ingestion time** : définit quand la donnée est ingérée dans le système.
- **Processing time** : définit quand la donnée est traitée.



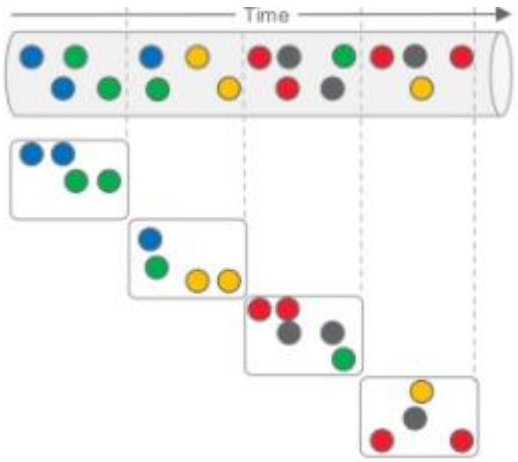
Cf slides 27 à 30 ici.

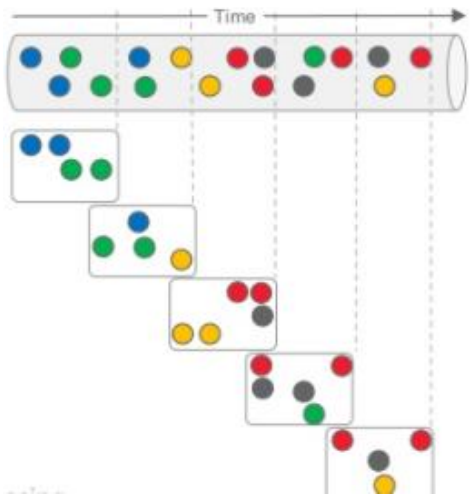
Stream Processing Window

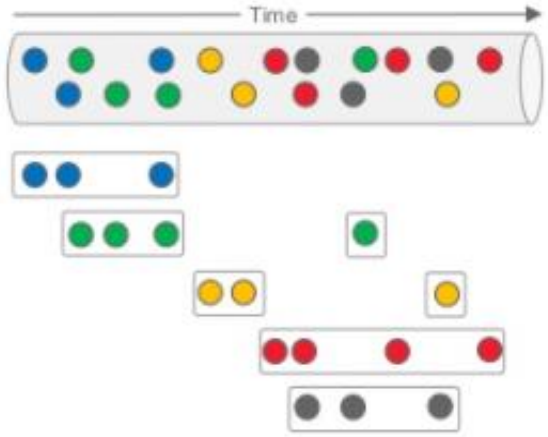
Les **fenêtres** permettent de compartimenter les données pour éviter des **problèmes de mémoire** du processeur.

Plusieurs types de fenêtres existent :

- **Fixed window** : fenêtres de même taille et pas d'enchevêtrement.
- **Sliding window** : fenêtres de même taille et une durée d'intervalle de glissement (< durée d'une fenêtre).
- **Session Window** : fenêtres de tailles différentes et définies par des identifiants de session.





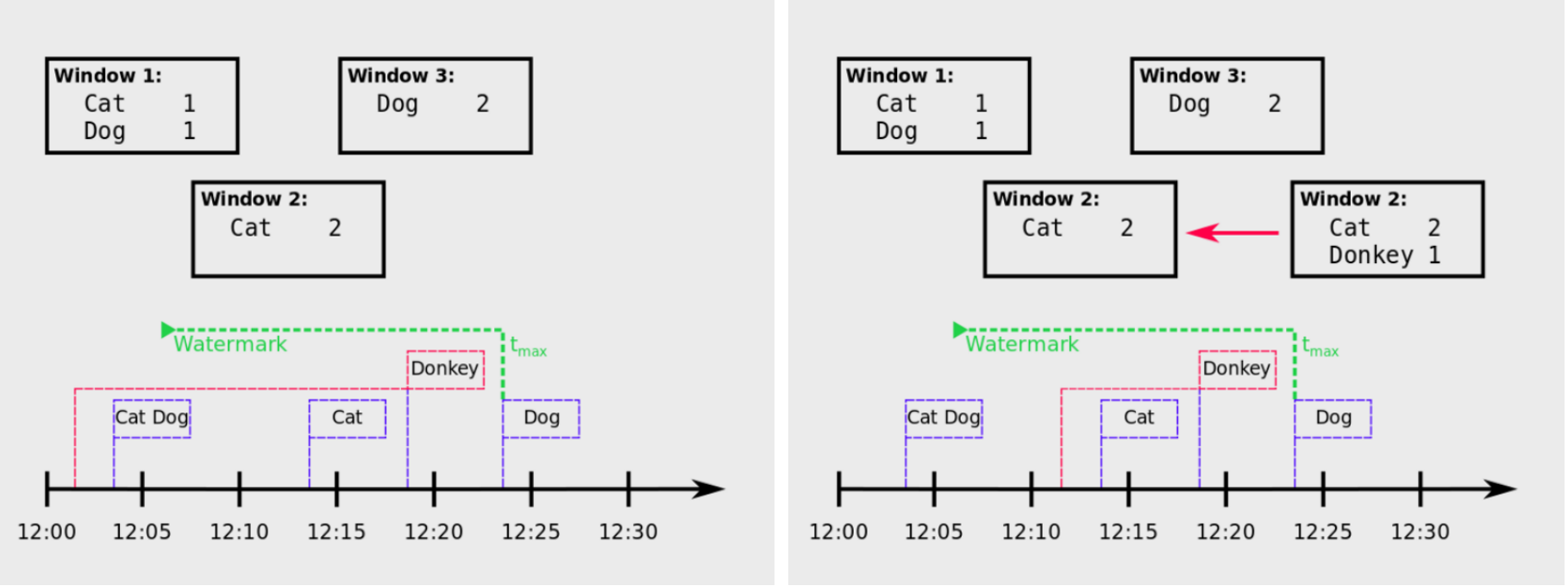


Watermarks

Un **watermark** est un outil qui permet de définir quand **tous les évènements qui le précèdent** sont considérés comme **acquis**.

Il permet de répondre au **retard** qu'un système de stream processing peut avoir.

Il est défini par une **durée** pendant laquelle les données peuvent encore arrivées.



Cf ici pour plus d'informations.

Triggers

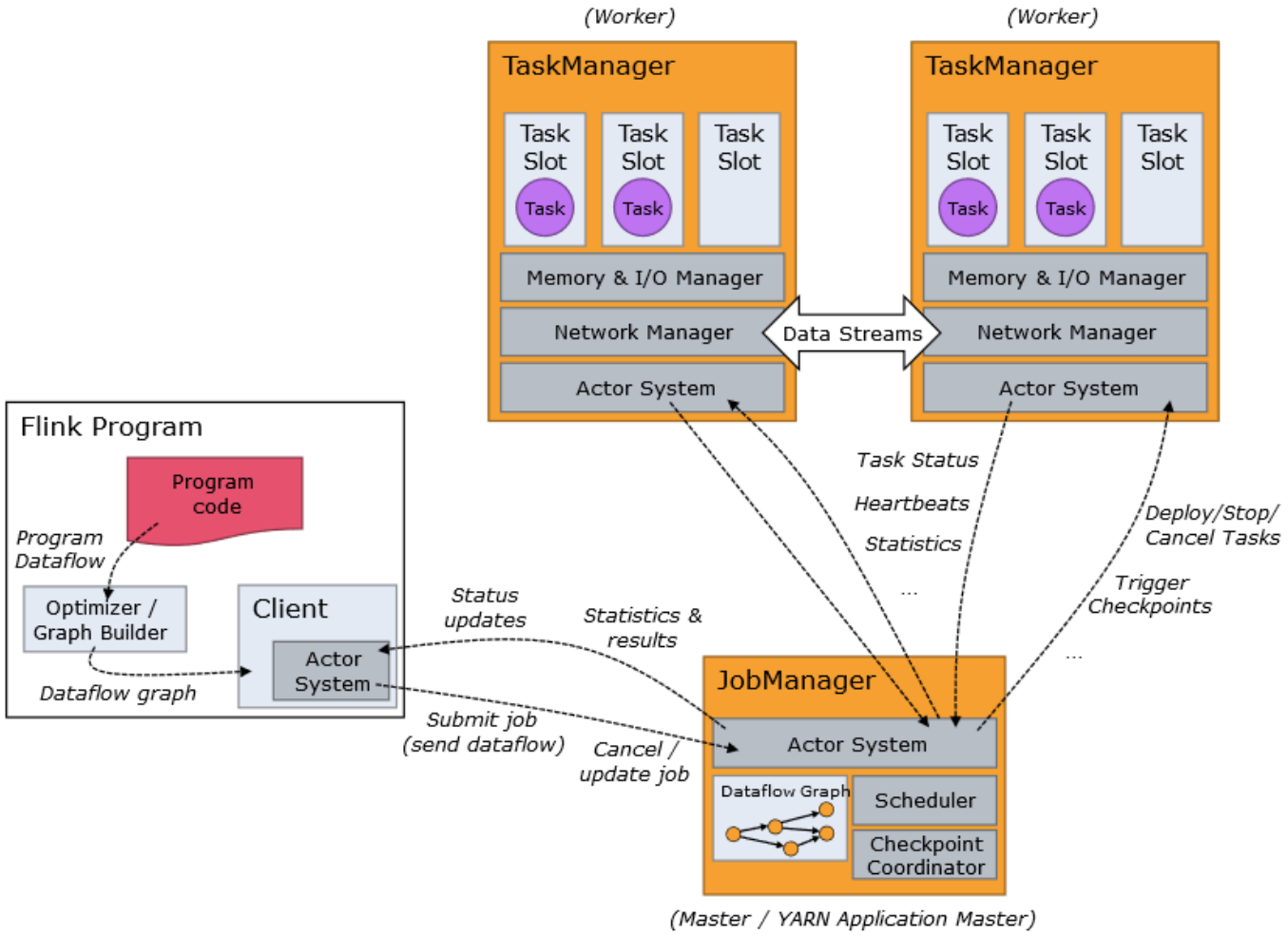
Un **trigger** permet de déclencher le **calcul le résultat final** de la fenêtre.

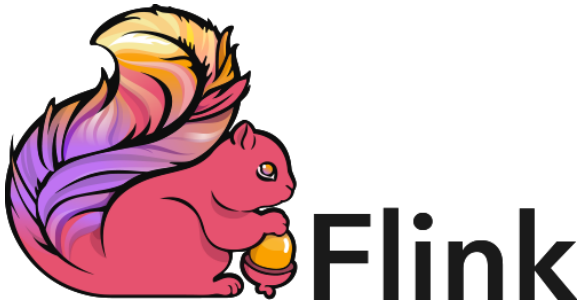
- **Watermark progress** : calcul le résultat final de la fenêtre quand le watemark dépasse les limites d'une fenêtre.
- **Event time progress** : calcul les résultats des fenêtres plusieurs fois, avec un interval spécifique, au fur et à mesure que le watermark avance.
- **Processing time progress** : calcul les résultats des fenêtres plusieurs fois, avec un interval spécifique mesuré par rapport au temps de calcul.
- **Punctuations** : dépend de la donnée.

Cf ici pour plus d'informations sur le windowing.

Open source : La Suite Apache

Apache Flink Architecture

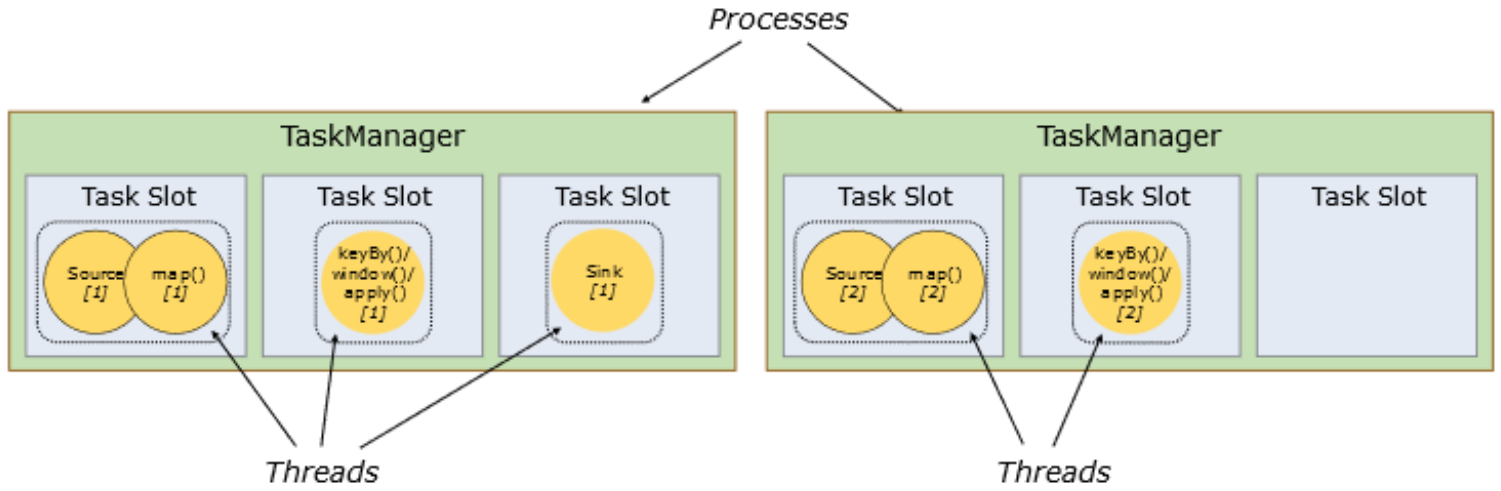




- Architecture **Maître esclave**
- Un **JobManager**, planifie les tâches
- Les **TaskManagers**, exécutent les tâches

https://flink.apache.org/

Apache Flink



- Le Task Slot : un espace mémoire isolé dédié à l'exécution des ***Threads***
- Un TaskManager peut avoir un ou plusieurs Task slots.
- Différente façon de déployer Flink (standalone, Docker…)

Amazon Kinesis

Avantages

- Puissance de traitement en temps réel
- Aucun serveur à gérer
- Paiement en fonction de l'utilisation

Cas d'utilisation

- ETL de streaming
- Analytique en temps réel
- Traitement dynamique des événements

https://docs.aws.amazon.com/streams/latest/dev/key-concepts

Google Dataflow

Avantages

- Analyse rapide des flux de données
- Opérations et gestion simplifiées
- Réduction du coût total de possession

Cas d'utilisation

- IA en temps réel : Détection d'anomalies / Reconnaissance de formes / Prévisions de prédiction
- Traitement des données de capteurs et de journaux

https://cloud.google.com

Comment choisir ?

Quels Avantages des données diffusées en continu ?

Quel coût et budget ?

Exemples de données diffusées en continu : Une société d'énergie solaire / Une société de jeux en ligne / Un établissement financier

Tableau comparatif : IT Central Station

https://www.itcentralstation.com

Apache Flink

https://www.itcentralstation.com/product_reviews/apache-flink-review-120830-by-rahul-agarwal?tid=pdf_cat_2612

Amazon Kinesis

https://www.itcentralstation.com/product_reviews/amazon-kinesis-review-123196-by-pablo-giner?tid=pdf_cat_2612

Google dataflow

https://www.itcentralstation.com/products/comparisons/apache-nifi_vs_google-cloud-dataflow?tid=pdf_cat_2612

Focus sur Flink avec une démo

- Téléchargement de la dernière version bin de flink : ici
- Changement du port de flink : <rootFlink>/conf/flink-conf.yaml
  - Rest.port: <numero du port>

- Démarrage de flink : <flink-home>/bin/start-cluster.sh
- Lancement du fichier : <flink-home>/bin/flink run <javaClass>.jar --input <wordInput.txt> --output <outputText>
- Regarder les logs : cat <outputText>

- tutoriel utilisé