MU5IN852
Bases de Données Large Echelle

data streaming et requêtes

novembre 2024

Objectifs

- Aperçu du data streaming
- Notion de fenêtre sur des flux
- Notion de requêtes continues
  - Jointures sur des fenêtres
- Perspectives

Références

- Conférence internationale SIGMOD 2018
  - Titre de l'article

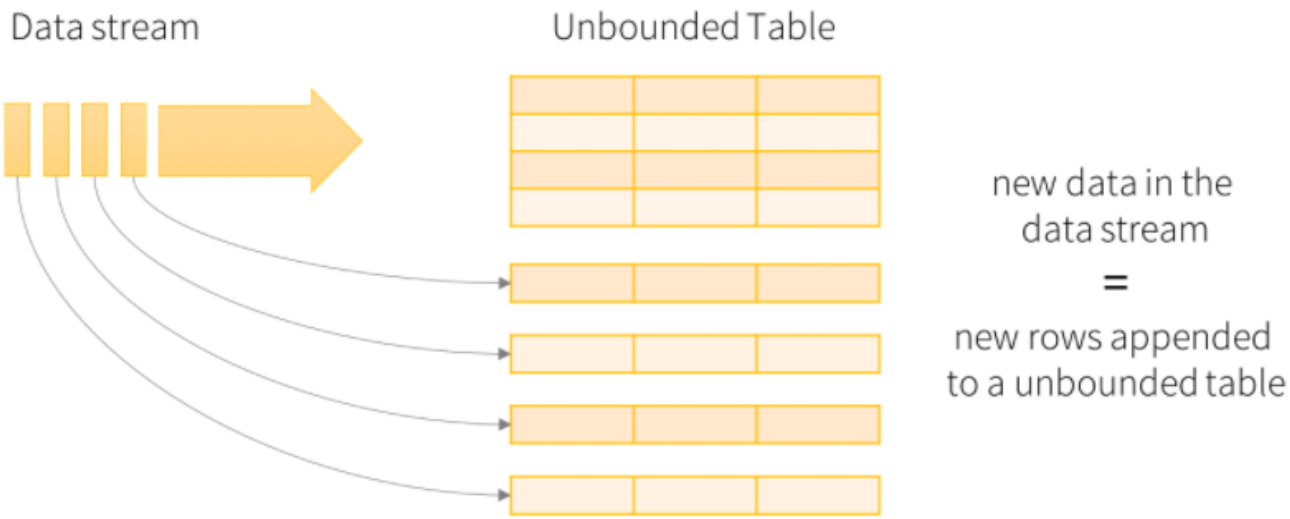
    Structured Streaming: A Declarative API for Real-Time Applications in Apache Spark
  - Auteurs

    Armbrust et al : Databricks, Stanford Univ
  - URL

    https://databricks.com/wp-content/uploads/2018/12/sigmod_structured_streaming.pdf

Contexte : Flux

- Données produites en continu
  - Ensemble ordonné de tuples, de taille infinie
    - Ordre partiel si la source est distribuée
  - Estampille
    - attribut date d'événement ou date d'arrivée



Data stream as an unbounded table

Motivations et Défis

- Requêtes incrémentales complexes à exprimer
  - Besoin de langage déclaratif
- Chaîne de traitement intégrée
- Défis opérationnels
  - Pannes et retards dus aux stragglers
    - tâches « à la traine »
  - Mise à jour des applis traitant un stream
  - Redimensionnement des ressources allouées
- Métriques de performance
  - Débit : nombre de tuples traités par minute

  versus
  - Latence : temps de réponse d'une requête
    - date du résultat de la requête – date d'arrivée de la donnée

Système de streaming

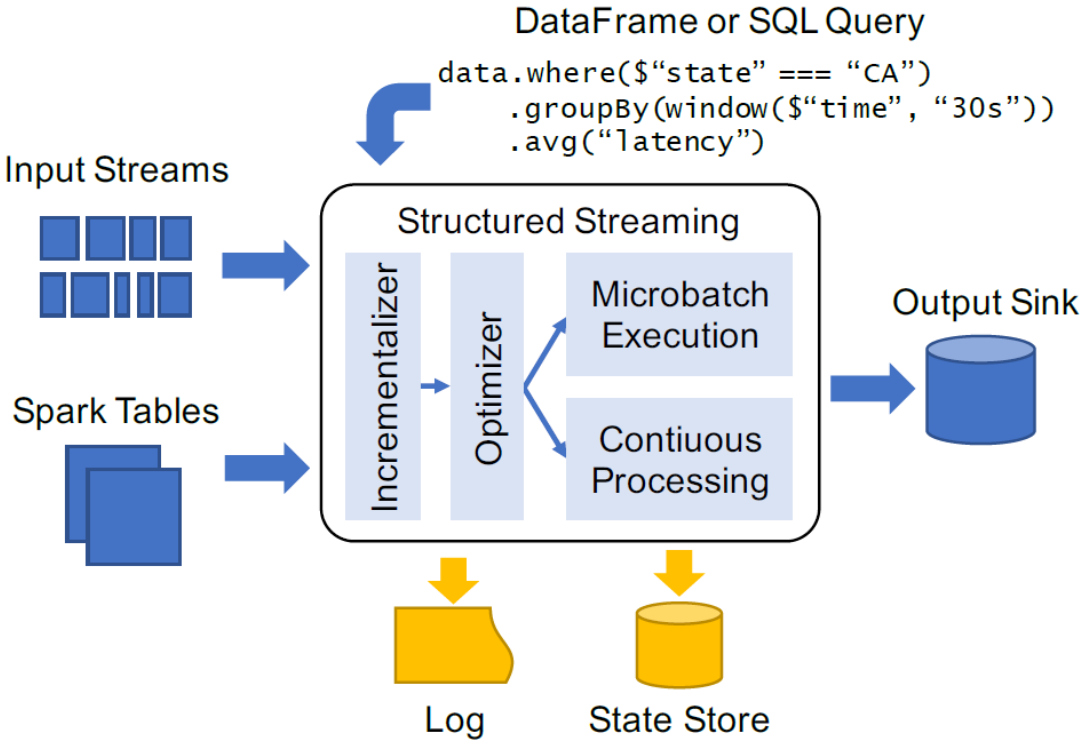
- Système de gestion des flux et des requêtes
  - Scalable : architecture distribuée
- Gestion des flux entrants
  - Tolérance aux pannes
    - Stockage temporaire des flux
    - Possibilité de répéter l'arrivée d'un flux
  - Propriété sémantique
    - Chaque tuple arrive une et une seule fois : « exactly once »
  - Exemple : Kafka
- Gestion des flux sortants
  - Tolérance aux pannes
    - Ecriture indempotente : 1 ou plusieurs invocations d'une écriture produit le même résultat

Apache Spark Structured Stream

- Unbounded Input stream of structured data
- Leverages on existing parallel and distributed processing framework
  - Stream processing = processing a sequence of bounded data
- Windowing
  - split input stream into a sequence of windowed data
  - Applies **parallel** distributed processing for each window
- Input stream may be already distributed
  - E.g., kafka input stream

Architecture du système Structured Streaming

- Données mixtes :
  - Flux dynamiques et/ou tables statiques
- Requêtes déclaratives



Exécution de requêtes

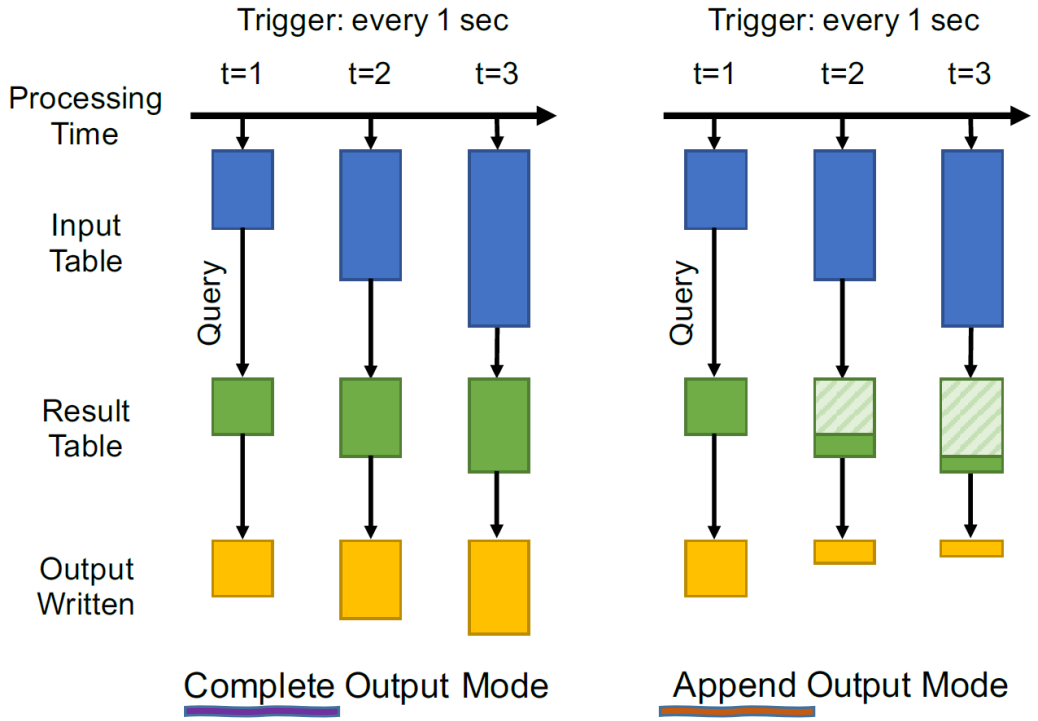
- Requête posée sur toutes les données *d* telles que
  - *d.date <= t*
- Modes d'exécution
  - **Périodique** : exécution toutes les *n* secondes
    - Appelé « micro-batch »

  ou
  - Continue : exécution à chaque nouveau tuple entrant

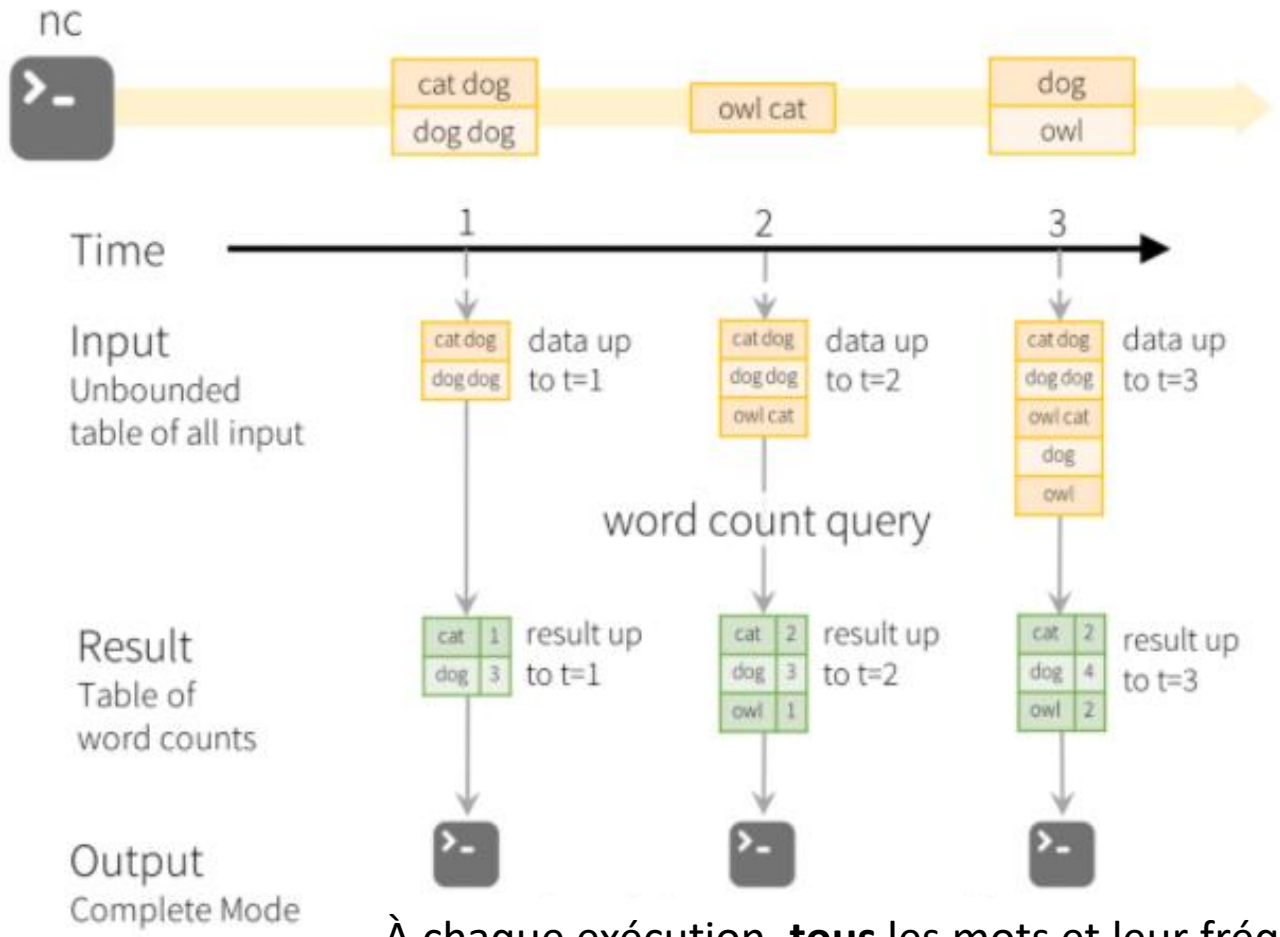
Exécution périodique : mode de sortie

Exécution périodique de la requête avec trois modes de sortie possibles :

- **Complete** : Résultat complet à chaque instant t
- **Append** : Résultat = seulement les nouveaux tuples
- Update : Résultat = les tuples à modifier ou à ajouter



Mode de sortie « Complete »
Exemple de requête « word count »

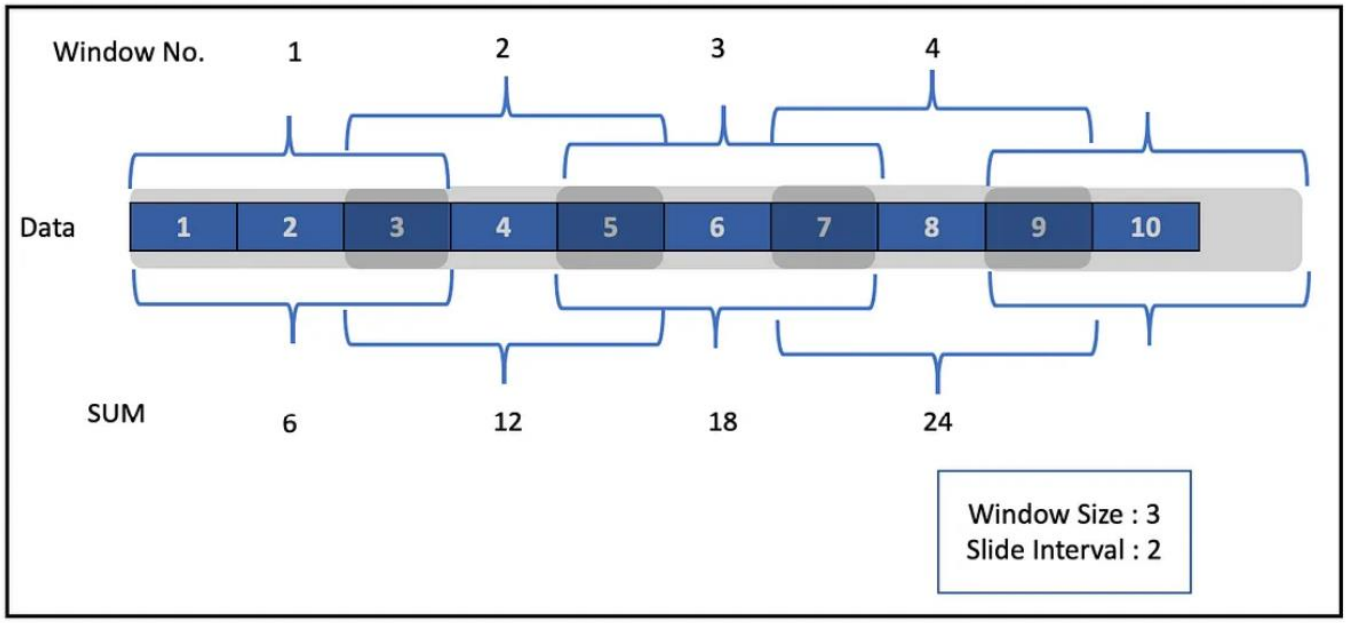


À chaque exécution, **tous** les mots et leur fréquence sont produits

Fenêtrage temporel : GROUP BY WINDOW

- basé sur la date de l'événement
  - Attribut du flux entrant
- Syntaxe : GROUP BY WINDOW(attr, taille, décalage)
  - attribut de type date
  - taille de la fenêtre
  - décalage entre les dates de début de deux fenêtres consécutives
- Recouvrement partiel des fenêtres consécutives
  - si décalage < taille

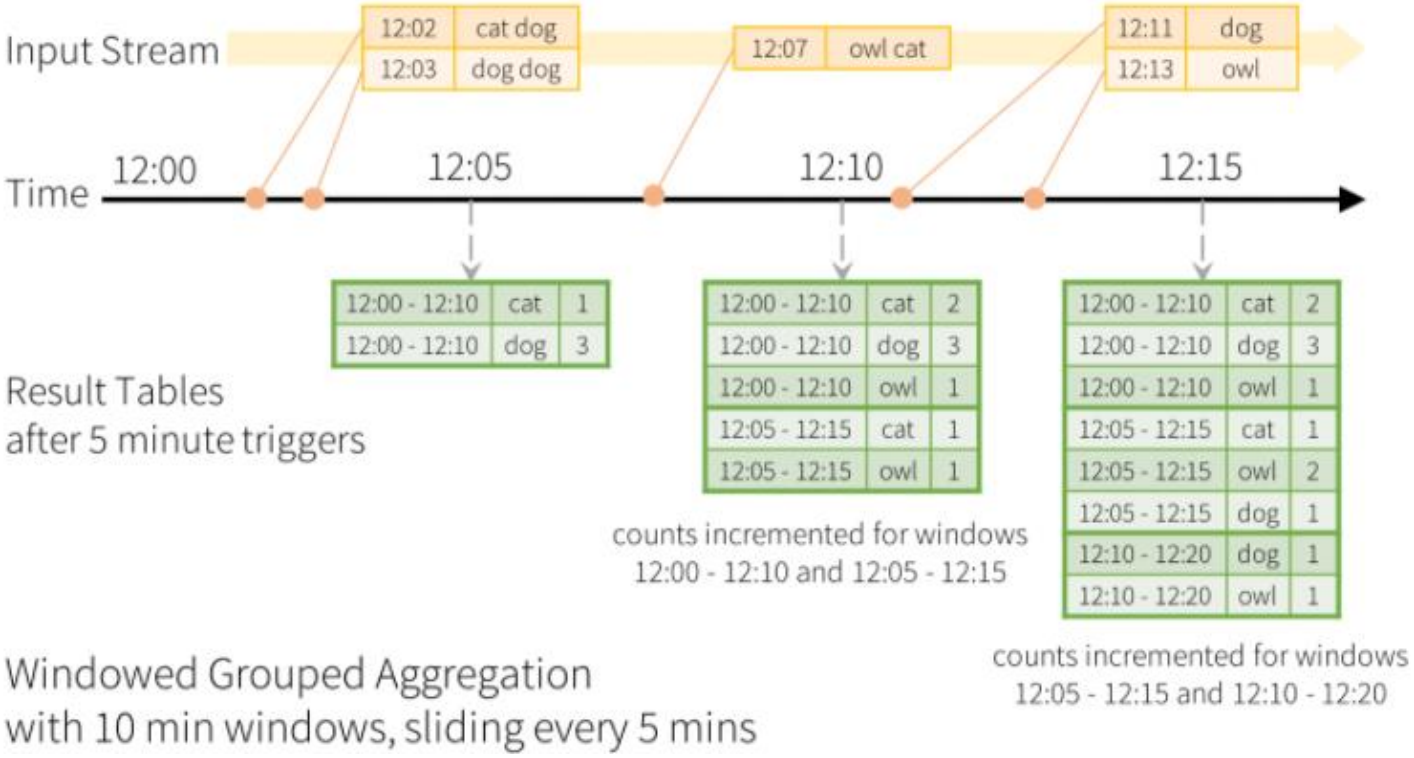
Exemple de fenêtre avec recouvrement



https://medium.com/expedia-group-tech/apache-spark-structured-streaming-operations-5-of-6-40d907866fa7

Fenêtre temporelle avec recouvrement

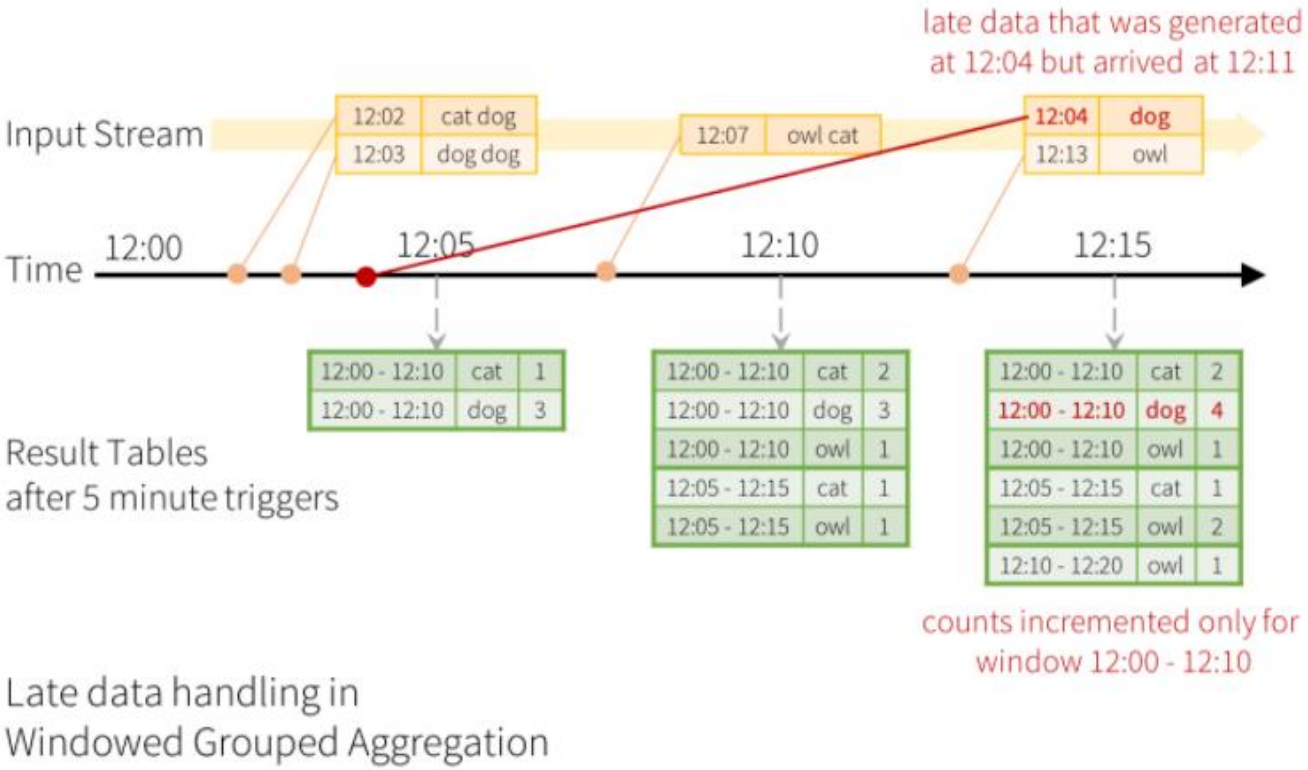
- Taille (ou durée) de la fenêtre : 10 min
- Décalage : 5 min



Ordre d'arrivée des données

- Hypothèse :
  - L'ordre d'arrivée peut être différent de l'ordre obtenu en triant les données par date croissante (l'attribut date servant d'estampille temporelle).
    - date d'arrivée ≠ date du tuple
    - les données peuvent arriver « en retard » ou « en avance »
- Problème :
  - Peut-on garantir la complétude des résultats sur des flux potentiellement infinis ?

Illustration du problème

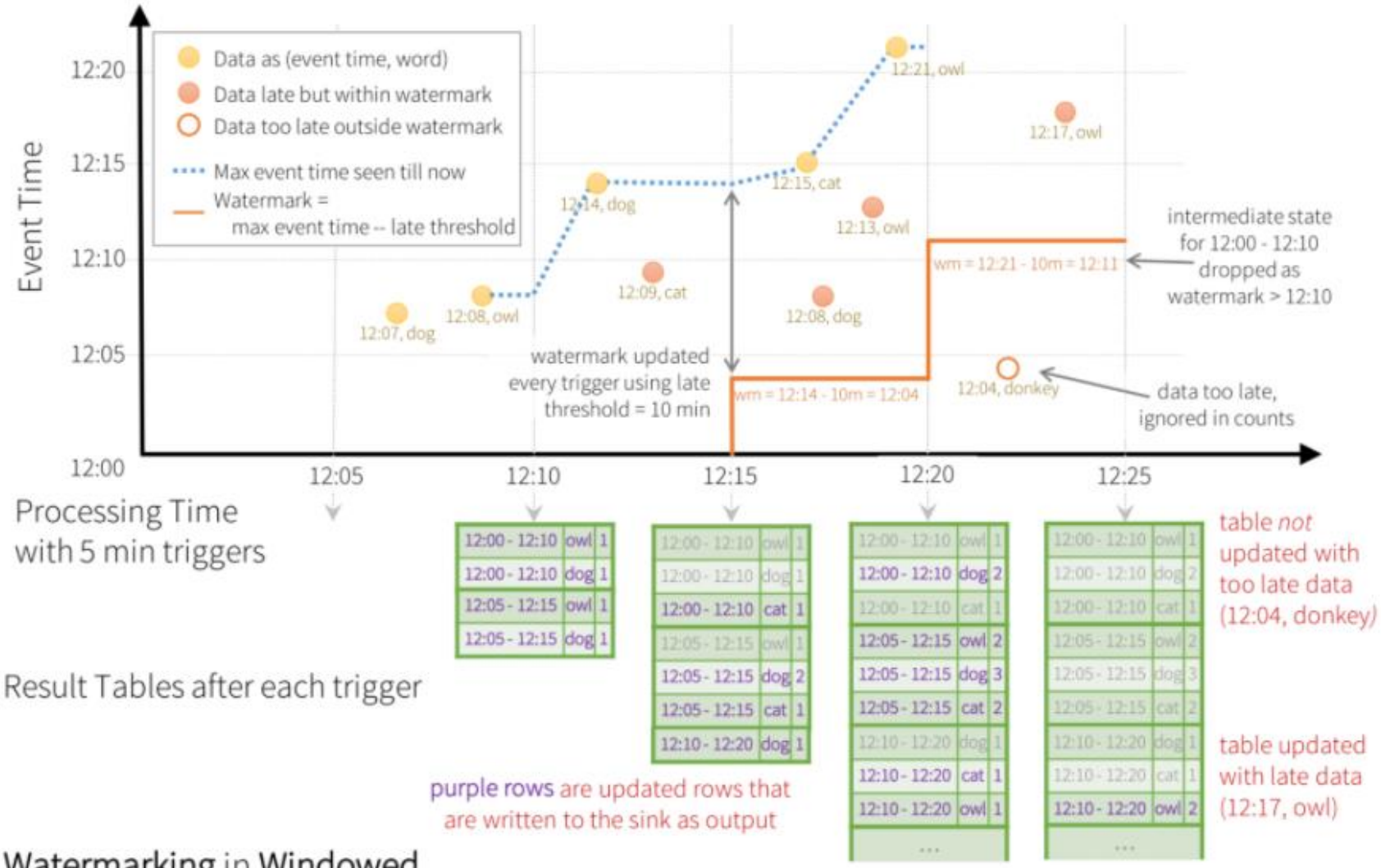


Borner le flux pour ignorer les données trop tardives

Flux borné : WITH WATERMARK

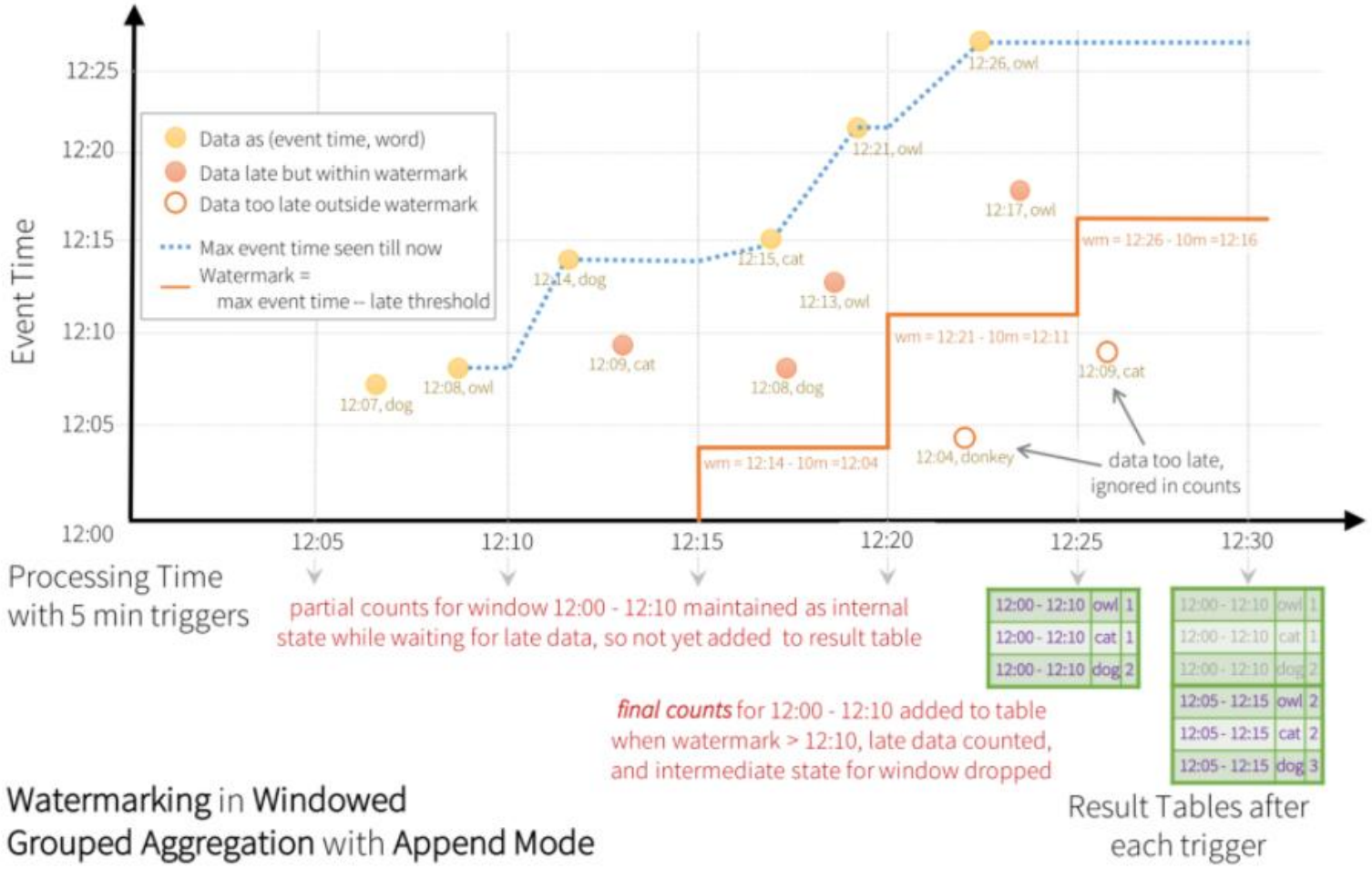
- Solution : spécifier une **contrainte** sur la date d'événement par rapport à la date courante
- Syntaxe : WITH WATERMARK attribut, retard
  - attribut de type date
  - retard toléré = longueur de l'intervalle de validité d'un flux
- Condition requise
  - date tuple > date max – retard toléré
  - date max : date maximale parmi les tuples déjà arrivés
    - c'est la borne supérieure de l'intervalle de validité
    - Elle peut être supérieure à la date courante…
    - c'est la date max des tuples arrivés avant la *précédente* exécution de la requête.
    - Ne prend pas en compte les tuples arrivés entre la précédente exécution et l'exécution courante.

Illustration du watermarking (1/2)



Watermarking in **Windowed Grouped Aggregation** with **Update Mode**

Illustration du watermarking (2/2)



Jointure de flux

- Problème:
  - A la date courante, est-ce qu'on a assez d'information pour déterminer le résultat de la jointure ?
    - Peut-on joindre les tuples qui viennent d'arriver?

- Plusieurs aspects à considérer
  - Dynamicité :
    - Jointure entre 1 flux et une table
    - Jointure entre 2 flux
  - Type de jointure
    - Jointure standard (inner join)
    - Jointure externe (left/right outer join)

Jointure entre un flux et une table

- Jointure interne : *inner join*
  - Syntaxe: Flux JOIN Table
  - Chaque tuple du flux peut être comparé avec ceux de la table
- Jointure **externe**
  - Syntaxe
    - **flux à gauche :** Flux LEFT JOIN Table
    - **ou flux à droite :** Table RIGHT JOIN Flux
  - Le résultat contient les tuples du flux sans correspondance avec la table
  - Limitation :
    - LEFT JOIN impossible si le flux est à droite : ~~Table LEFT JOIN Flux~~
      - et réciproquement, right join impossible si le flux est à gauche
    - Car on ne peut pas savoir à l'avance si un tuple de la table va joindre avec les prochains tuples du flux non encore arrivés

Jointure entre deux flux

A chaque évaluation de la jointure :

- Déterminer quelles sont les données à comparer dans chaque flux
  - Ignorer les données qui arrivent trop en retard
    - Définir une watermark sur chaque flux
- Empêcher que la quantité de données à comparer n'augmente indéfiniment
  - Ecarter les données trop anciennes
    - ajouter un prédicat comparant les dates des tuples
    - WHERE date1 BETWEEN date2 AND date2 + interval 10 minutes
  - Ce prédicat définit une « fenêtre » car il détermine la date minimale à conserver dans chaque flux
    - Remarque : la jointure n'est pas une agrégation donc on ne définit pas explicitement de fenêtre

Requêtes incrémentales

- Evaluation incrémentale sur des fenêtres avec recouvrement
  - Eviter les calculs redondants
  - Etat transmis entre deux évaluations consécutives de la même requête
- Extensibilité et efficacité
  - Fonction définie par l'utilisateur (UDF) implantant une méthode de calcul incrémental

Calcul incrémental: exemple (1/2)

- Table T(*d* date, *a* int)
- Requête S avec agrégation associative sur une fenêtre glissante de 15' avec un décalage de 5' :
  - S: select sum(a) … from T group by window(d, 15, 5)
  - date initiale t_0
  - date ti = t_0 + 5i
  - La fenêtre Wi couvre la période [t_i, t_i + 15[
  - La requête produit le résultat S_i pour chaque fenêtre W_i
    - S_i est la somme des valeurs de a dans la fenêtre Wi
    - S_i = sum({ r.a | $r.d \in \mathbf{Wi}$ })

Calcul incrémental: exemple (2/2)

- On découpe les fenêtres en intervalles consécutifs disjoints de 5 minutes
  - $\mathbf{wi} = [\mathbf{t_i},\ \mathbf{t_i} + \mathbf{5}\ [$
- *Soit s_i la somme sur chaque intervalle wi*
  - s_i = sum({ r.a | $\mathbf{r.d} \in \mathbf{wi}$ })
  - Pour i > 2: on maintient les résultats intermédiaire s_{i-2} et s_{i-1}
- On a $W_i = w_i \cup w_{i+1} \cup w_{i+2}$
- Le résultat de la requêtes est $S_i = s_i + s_{i+1} + s_{i+2}$
  - s_{i+1} et s_{i+2} sont déjà calculés et seul s_i est un nouveau terme
- or $S_{i-1} = s_{i-1} + s_i + s_{i+1}$ équivaut à $s_i + s_{i+1} = S_{i-1} - s_{i-1}$
- Donc $S_i = S_{i-1} - s_{i-1} + s_{i+2}$

Biblio et perspectives

- Spark Structured Streaming
  - (rappel) article SIGMOD 2018 : Structured Streaming
  - Programming guide
    - https://spark.apache.org/docs/latest/structured-streaming-programming-guide.html
- Académique :
  - VLDB 2015 : Google DataFlow Model
  - BigData 2018 : BigSR: real-time expressive RDF stream reasoning on modern Big Data platforms
    - https://ieeexplore.ieee.org/document/8621947
  - EDBT 2020 Tutorial: Declarative Languages for Big Streaming Data
    - https://openproceedings.org/2020/conf/edbt/paper_T1.pdf
  - VLDB 2023 Erebus : Explaining the Outputs of Data Streaming Queries (article + code)
    - https://www.vldb.org/pvldb/vol16/p230-palyvos-giannas.pdf
  - VLDB Journal 2023: A survey on the evolution of stream processing systems
    - https://link.springer.com/article/10.1007/s00778-023-00819-8